

2nd
Revised &
Updated
Edition

BlockChain

From Concept to Execution

With 10 Blockchains, 3 DLTs, 182 MCQs, 70 Diagrams &
Many Sample Codes



DEBAJANI MOHANTY



2nd

Revised &
Updated
Edition

BlockChain

From Concept to Execution

With 10 Blockchains, 3 DLTs, 182 MCQs, 70 Diagrams &
Many Sample Codes



DEBAJANI MOHANTY



Blockchain From Concept to Execution

***With 10 Blockchains, 3 DLTs, 182
MCQs,
70 Diagrams & Many Sample Codes***

***2nd
Revised & Updated Edition***

Debajani Mohanty



www.bpbonline.com

SECOND EDITION 2022

Copyright © BPB Publications, India

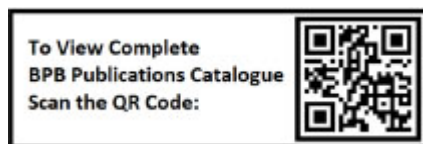
ISBN: 978-93-89423-42-6

All Rights Reserved. No part of this publication may be reproduced, distributed or transmitted in any form or by any means or stored in a database or retrieval system, without the prior written permission of the publisher with the exception to the program listings which may be entered, stored and executed in a computer system, but they can not be reproduced by the means of publication, photocopy, recording, or by any electronic and mechanical means.

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true to correct and the best of author's and publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but publisher cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners but BPB Publications cannot guarantee the accuracy of this information.



www.bpbonline.com

Dedicated to

My Mother Mrs Nirupama Mohanty

About the Author



Debajani Mohanty, one of the top 30 Blockchain Influencers of India 2021 (Singapore Fintech News), Top 100 Global Social Influencer (Piktale awards), and Blockchain Catalysts of India 2019-2020 (Coin Crunch), is a Solution Architect and author of 5 Amazon bestseller Blockchain books, some of which are translated into German and Chinese to reach a wider mass. Pioneer Blockchain solutions developed by Debajani in different verticals have attracted global attention in the Fintech, e-Governance, and Telecom verticals. Debajani works as a Chief Architect at EarthId and is a keynote speaker at many national and international innovation summits.

Twitter ID: <https://twitter.com/debimr75>

Linkedin ID: <https://www.linkedin.com/in/debajanimohantypmp/>

Acknowledgements

I am indebted to Prasanna Lohar, Anthony Day, Sharat Chandra, Shiv Aggarwal, Rajesh Dhuddu, Richard G Brown, Srinivas Mahankali, Rohas Nagpal, and many more thought leaders, whose posts I regularly followed while authoring this book.

I owe special thanks to my mother, husband, and daughters for helping me with the household chores during the long Covid-19 lockdown in India. It's due to their sincere and kind efforts that I could concentrate on my work wholeheartedly. I salute all the healthcare workers, cleaners, suppliers, and police who have been working relentlessly; because of them, we have been able to concentrate on our work with peace of mind. Let Science, Technology, Medicine, and Farming have just one goal – prosperity of the world that we live in. Because of the pandemic, I have learned this lesson – We are “One”.

Preface

Any idea why I am so much hooked up to Blockchain?

Weird but true, the Universe itself is decentralized in nature and pretty messed up. There is no single master, no single force, and no particular nation that can be termed as the most powerful.

The beauty of any emerging technology lies in the fact that nobody is an expert here. The space is ever changing and evolving manifolds over the passing years. Even the very mysterious, all-wise father of Blockchain, Satoshi Nakamoto, has to stay relevant in the market through regular learning. There was a time when the compact 109 pager “Blockchain from concept to execution” remained a sweetheart among the Blockchain lovers. Now, the time has come for a fresh start, and it starts 100 steps ahead from where it ended in the last version.

Today, the Blockchain comes with many variations, including shared ledger, distributed ledger, mutable ledger etc. In addition to that, there are adjoining technologies as the layer-2 set up and low code environments for smart contracts. Knowing them all and matching to the individual’s requirement is a must for the future IT industry.

This new version is thoughtfully designed to match the need of the students and experts alike.

Phase I covers the most widely adopted Blockchains of today. The first chapter starts from the very basic concepts of Blockchain that everyone should learn. The remaining chapters of this phase discuss some of the most popular Blockchains of today.

Phase II further looks over the popular public inter-operable Blockchains in the market. It also explores the competitive study between the different public Blockchains and inter-operable Blockchains.

Phase III illustrates the private permissioned DLTs that are adopted by the organizations. The final chapter in this phase also comes with a comparative study to help the reader choose one over the other.

Phase IV describes some of the most popular industry use cases as of today.

Phase V gives a guideline on how an industry can fast-track the Blockchain adoption and some research area of tomorrow.

The book is a super-fast read and hence can save a lot of time for the learners. It would be most suitable to the business leaders and architects to understand the capabilities and utilization of these frameworks and help them choose the right one for their respective business need in the projects. Also, many of these frameworks are still evolving, hence I would keep on updating the book as they do.

The previous version of “Blockchain from Concept to Execution” consistently remained the bestseller on Amazon for close to 2 years and sold more than 10,000 copies. Hope this version too would do similar wonders, ending up being a compact companion of every Blockchain enthusiast for a quick reference.

Downloading the code bundle and coloured images:

Please follow the link to download the
Code Bundle and the **Coloured Images** of the book:

<https://rebrand.ly/2700b0>

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Did you know that BPB offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.bpbonline.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at business@bpbonline.com for more details.

At www.bpbonline.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on BPB books and eBooks.

BPB is searching for authors like you

If you're interested in becoming an author for BPB, please visit www.bpbonline.com and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

The code bundle for the book is also hosted on GitHub at <https://github.com/bpbpublications/Blockchain-From-Concept-to-Execution>. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at <https://github.com/bpbpublications>. Check them out!

PIRACY

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at business@bpbonline.com with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit www.bpbonline.com.

REVIEWS

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased

opinion to make purchase decisions, we at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit www.bpbonline.com.

Table of Contents

1. Introduction to Blockchain

1.1 The Blockchain Market

1.2 Evolution of Blockchain Technology and Hype

1.3 Birth of Bitcoin

1.4 Bitcoin vs. Previous Generation Electronic Money

1.5 Key Concepts

1.5.1 Cryptography, Encryption, and Decryption

1.5.1.1 Symmetric Cryptography

1.5.1.2 Asymmetric Cryptography

1.5.1.2.1 RSA

1.5.1.2.2 DSA

1.5.1.2.3 Elliptic curve cryptography and ECDSA

1.5.1.2.4 Edward curve Cryptography and EdDSA

1.5.1.2.5 Ed25519

1.5.2 Hashing

1.5.3 Wallet

1.5.3.1 Online wallets

1.5.3.2 Desktop wallets

1.5.3.3 Mobile wallets

1.5.3.4 Hardware wallets

1.5.3.1.5 Paper wallets

1.5.3.1.6 Non-custodial wallets

1.5.3.1.7 Custodial wallets

1.5.4 Decentralization

1.5.4.1 Centralized System

1.5.4.2 Distributed System

1.5.4.3 Decentralized System

1.5.5 Consensus

1.5.5.1 Proof of Work (PoW)

1.5.5.2 Proof of Stake (PoS)

1.5.5.3 Delegated Proof of Stake (DPOS)

1.5.5.4 Practical Byzantine Fault Tolerance or PBFT

1.5.5.5 Tendermint

- [1.5.5.6 Direct Acrylic Graph](#)
 - [1.5.6 Nodes](#)
 - [1.5.7 Shared Replicated Ledger](#)
 - [1.5.8 Smart Contracts](#)
 - [1.5.9 Native Token](#)
 - [1.5.10 Gossip Network for Message Propagation](#)
 - [1.5.11 Hard Fork and Soft Fork](#)
 - [1.6 How Bitcoin Blockchain Works](#)
 - [1.6.1 Mining](#)
 - [1.6.2 Proof of Work](#)
 - [1.6.3 Consensus with Proof of Work](#)
 - [1.6.4 DAPPS](#)
 - [1.7 Threats & Challenges in Public Blockchains](#)
 - [1.7.1 Double Spending](#)
 - [1.7.2 Sybil Attack](#)
 - [1.7.3 DDoS Attack](#)
 - [1.8 Permission Levels](#)
 - [1.9 Does your Use Case need a Blockchain?](#)
 - [1.9.1 When to use a RDBMS](#)
 - [1.9.2 Where to Use a Public Blockchain](#)
 - [1.9.3 Where to Use a Private/Permissioned Blockchain](#)
 - [1.10 Cap Theorem or The Scalability Trilemma](#)
 - [1.11 Bitcoin, The Environment Villain](#)
 - [1.12 Areas of research and improvement](#)
 - [1.12.1 Consensus Model](#)
 - [1.12.2 Scalability and Performance](#)
 - [1.12.3 Transaction Fees](#)
 - [1.12.4 Layer-2 Solutions](#)
 - [1.12.4.1 Sharding](#)
 - [1.12.4.2 State Channels](#)
 - [1.12.4.3 Parachains](#)
 - [1.12.4.4 Sidechains](#)
 - [1.12.4.5 Rollups](#)
 - [1.13 Reason for the Volatile Price of Crypto](#)
 - [1.13.1 Utility](#)
 - [1.13.2 Limited vs. Unlimited Supply](#)
 - [1.13.3 Hoarding](#)

[1.14 Market Capitalization](#)

[Summary](#)

[References](#)

[Quiz](#)

[Answers](#)

PHASE I – PUBLIC BLOCKCHAINS

2. Ethereum

[2.1 Consensus Model](#)

[2.2 Transaction fees](#)

[2.3 L2 Solutions](#)

[2.4 Scalability and Performance](#)

[2.5 Development](#)

[2.5.1 Why Learn Solidity?](#)

[2.5.2 Solidity vs. Who?](#)

[2.5.3 Solidity Features](#)

[2.5.4 REMIX browser](#)

[2.5.5 Write a Small Program](#)

[2.5.5.1 Compiling](#)

[2.5.5.2 Deployment](#)

[2.5.6 Application Binary Interface](#)

[2.5.7 Layout of Solidity source code](#)

[2.5.7.1 SPDX License Number](#)

[2.5.7.2 Version](#)

[2.5.7.3 Import File](#)

[2.5.7.4 Contract](#)

[2.5.7.5 Libraries](#)

[2.5.7.6 Interfaces](#)

[2.5.7.7 Events](#)

[2.5.7.8 Order of Elements in Solidity File](#)

[2.5.7.9 Order of Elements in a Contract](#)

[2.5.8 Functions in Solidity](#)

[2.5.8.1 Constructor](#)

[2.5.8.2 Setter\(\) and Getter\(\)](#)

[2.5.8.3 Return Values](#)

[2.5.8.4 Order of Functions](#)

2.5.9 Variables

2.5.9.1 State variable o9

2.5.9.2 Local Variable

2.5.9.3 Global variables

2.5.10 Types

2.5.10.1 Value Type

2.5.10.1.1 Integers

2.5.10.1.2 Fixed Numbers

2.5.10.1.3 Boolean

2.5.10.1.4 Byte

2.5.10.1.5 String Literals

2.5.10.1.6 Enum

2.5.10.1.7 Address

2.5.10.2 Reference Type

2.5.10.2.1 Array

2.5.10.2.2 Struct

2.5.10.2.3 Mappings

2.5.10.3 Constant and Immutable State Variables

2.5.11 Operators in Solidity

2.5.11.1 Arithmetic Operator

2.5.11.2 Relational Operators

2.5.11.3 Logical Operators

2.5.11.4 Bitwise Operator

2.5.11.5 Assignment Operators

2.5.11.6 Conditional (or ternary) Operators

2.5.12 Gas and Operation Cost of Ethereum

2.5.12.1 Ether

2.5.12.2 Gas vs. Ether

2.5.12.3 Gas limit

2.5.12.4 Gas price

2.5.12.5 Future of Gas

2.5.13 Storing Data in Solidity

2.5.13.1 Storage

2.5.13.2 Memory

2.5.13.3 Calldata

2.5.13.4 Stack

2.5.14 Control of Flow in Solidity

- [2.5.14.1 if-else Condition](#)
 - [2.5.14.2 while-do Loop](#)
 - [2.5.14.3 do-while Loop](#)
 - [2.5.14.4 for Loop](#)
 - [2.5.14.5 break Statement](#)
 - [2.5.14.6 Continue Statement](#)
 - [2.5.14.7 Conditional \(or ternary\) Operators](#)
- [2.5.15 View Function](#)
- [2.5.16 Pure Function](#)
- [2.5.17 Fallback Function](#)
- [2.5.18 Error Handling](#)
 - [2.5.18.1 Require](#)
 - [2.5.18.2 Revert](#)
 - [2.5.18.3 Assert](#)
 - [2.5.18.4 Try/Catch](#)
- [2.5.19 Events](#)
 - [2.5.19.1 Event Logs in Solidity](#)
 - [2.5.19.2 Output](#)
- [2.5.20 Object Oriented Approach of Solidity](#)
 - [2.5.20.1 Encapsulation](#)
 - [2.5.20.2 Abstraction](#)
 - [2.5.20.3 Inheritance](#)
 - [2.5.20.4 Polymorphism](#)
- [2.5.21 Abstract Contracts and Interfaces](#)
 - [2.5.21.1 Abstract Contract](#)
 - [2.5.21.2 Interface](#)
- [2.5.22 Creating and Destroying Contracts](#)
 - [2.5.22.1 Destruction of a Contract](#)
- [2.5.23 Lab 1](#)
- [2.5.24 Lab 2](#)
- [2.5.25 Ethereum Tokens and ERC Standards](#)
 - [2.5.25.1 Ethereum Request for Comments](#)
 - [2.5.25.2 ERC20](#)
- [2.5.26 Truffle and Ganache](#)
 - [2.5.26.1 Truffle](#)
 - [2.5.26.2 Ganache](#)

[2.6 Wallet](#)

[2.7 Live Projects](#)

[2.8 Altcoins](#)

[2.9 Summary](#)

[References](#)

[Quiz](#)

[Answers](#)

3. Hedera Hashgraph

[3.1 Consensus Model – Proof of Stake](#)

[3.2 Transaction fees](#)

[3.3 L2 Solutions](#)

[3.4 Scalability and Performance](#)

[3.5 Development](#)

[3.6 Wallet](#)

[3.7 Live Projects](#)

[3.8 Summary](#)

[References](#)

4. Tezos

[4.1 Consensus Model – Liquid Proof of Stake](#)

[4.2 Transaction fees](#)

[4.3 L2 Solutions](#)

[4.4 Scalability and Performance](#)

[4.5 Development](#)

[4.6 Live Projects](#)

[4.7 Summary](#)

[References](#)

5. Cardano

[5.1 Consensus Model – Proof of Stake](#)

[5.2 Transaction fees](#)

[5.3 L2 Solutions](#)

[5.4 Scalability and Performance](#)

[5.5 Development](#)

[5.5.1 Plutus](#)

[5.5.2 Marlowe](#)

[5.5.3 Rosetta](#)

[5.6 Wallet](#)
[5.7 Live Projects](#)
[5.8 Summary](#)
[References](#)

6. Algorand

[6.1 Consensus Model – Pure Proof of Stake](#)
[6.1.1 Block Proposal](#)
[6.1.2 Soft Vote](#)
[6.1.3 Certify Vote](#)
[6.2 Transaction fees](#)
[6.3 L2 Solutions](#)
[6.4 Scalability and Performance](#)
[6.5 Development](#)
[6.6 Wallet](#)
[6.7 Live Projects](#)
[6.8 Summary](#)
[References](#)

7. Solana

[7.1 Consensus Model – Proof of History & Proof of Stake](#)
[7.2 Transaction fees](#)
[7.3 L2 Solutions](#)
[7.4 Scalability and Performance](#)
[7.5 Development](#)
[7.6 Wallet](#)
[7.7 Live Projects](#)
[7.8 Summary](#)
[References](#)

8. Avalanche

[8.1 Consensus Model – Proof of Stake](#)
[8.2 Transaction fees](#)
[8.3 L2 Solutions](#)
[8.4 Scalability and Performance](#)
[8.5 Development](#)
[8.6 Wallet](#)

[8.7 Live Projects](#)

[8.8 Summary](#)

[References](#)

PHASE II – INTEROPERABLE BLOCKCHAINS

9. Polygon

[9.1 Consensus Model – Proof of Stake](#)

[9.2 Transaction fees](#)

[9.3 L2 Solutions](#)

[9.4 Scalability and Performance](#)

[9.5 Development](#)

[9.6 Wallet](#)

[9.7 Live Projects](#)

[9.8 Summary](#)

[References](#)

10. Polkadot

[10.1 Consensus Model – Nominated Proof of Stake](#)

[10.1.1 Parachain Slots](#)

[10.2 Transaction fees](#)

[10.3 L2 Solutions](#)

[10.4 Scalability and Performance](#)

[10.5 Development](#)

[10.6 Wallet](#)

[10.7 Live Projects](#)

[10.8 Summary](#)

[References](#)

11. Cosmos

[11.1 Consensus Model – Proof of Stake Tendermint](#)

[11.2 Transaction fees](#)

[11.3 L2 Solutions](#)

[11.4 Scalability and Performance](#)

[11.5 Development](#)

[11.6 Wallet](#)

[11.7 Live Projects](#)

[11.8 Summary.](#)
[References](#)

[12. Comparison of Blockchains](#)

[12.1 Comparison of Public Blockchains – Ethereum, Hashgraph, Cardano, Algorand, Avalanche, and Solana](#)

[12.2 Comparison of Interoperable Blockchains – Polkadot vs Cosmos vs Polygon](#)

[12.3 Use Cases on Interoperable Blockchains](#)

[12.4 Why the organizations stay away from public Blockchains?](#)

[Quiz](#)

[Answers](#)

[PHASE III – PRIVATE PERMISSIONED DLT](#)

[13. Hyperledger Fabric](#)

[13.1 Key Concepts](#)

[13.1.1 The Ledger](#)

[13.1.2 Identity](#)

[13.1.3 Membership Service Provider \(MSP\).](#)

[13.1.4 Policies](#)

[13.1.5 Peers](#)

[13.1.6 Channel](#)

[13.1.7 Orderer](#)

[13.1.7.1 Solo](#)

[13.1.7.2 Apache Kafka](#)

[13.1.7.3 Raft](#)

[13.1.8 Smart Contract \(Chaincode\).](#)

[13.2 Consensus and Fabric End-To-End Transaction Flow](#)

[13.2.1 Stage 1 Proposal and Endorsement](#)

[13.2.2 Stage 2 Ordering](#)

[13.2.3 Stage 3 Validation & Commit](#)

[13.3 Private Data](#)

[13.4 Interoperability with Oracles](#)

[13.5 Performance and Scalability.](#)

[13.6 Industry Adoption](#)

[13.7 Development](#)

[13.8 Deployment on Cloud](#)

[13.9 Summary](#)

[References](#)

14. R3 Corda

[14.1 Key Concepts](#)

[14.1.1 Network](#)

[14.1.2 Ledger](#)

[14.1.3 State](#)

[14.1.4 Transactions](#)

[14.1.5 Notary](#)

[14.1.6 Time Window](#)

[14.1.7 Attachments](#)

[14.1.8 Contracts](#)

[14.1.9 Interoperability with Oracles](#)

[14.1.10 Consensus](#)

[14.2 CordApp](#)

[14.3 Corda End-To-End Transaction Flow](#)

[14.4 Corda Accounts Model](#)

[14.5 Token SDK](#)

[14.6 Corda Network](#)

[14.7 Corda Opensource vs. Corda Enterprise](#)

[14.8 Performance and Scalability](#)

[14.9 Industry Adoption](#)

[14.10 Development](#)

[14.11 Deployment on Cloud](#)

[14.12 Summary](#)

[References](#)

15. Consensys Quorum

[15.1 Key Concept](#)

[15.2 Consensus - RAFT](#)

[15.2.1 Privacy Manager](#)

[15.2.2 Transaction Manager](#)

[15.2.3 Enclave](#)

[15.3 Consensys Quorum End to End Transaction Flow](#)

[15.4 Industry Adoption](#)

[15.5 Development](#)
[15.6 Deployment on Cloud](#)
[15.7 Summary](#)
[References](#)

16. Comparison of Hyperledger Fabric, R3 Corda, and Consensus Quorum

[References](#)
[Quiz](#)
[Answers](#)

PHASE IV – USE CASES

17. Decentralized Identity

[17.1 Digital Identity](#)
[17.2 Evolution of Digital Identity](#)
[17.2.1 Centralized Identity](#)
[17.2.2 Federated Identity](#)
[17.2.3 User Centric Identity](#)
[17.2.4 Decentralized Identity \(DID\)](#)
[17.3 World Wide Web Consortium](#)
[17.4 Data sharing types](#)
[17.5 DID Use Cases](#)
[17.5.1 Use Case - Remote Voting](#)
[17.5.2 Use Case – Smart Agriculture](#)
[17.6 DID for Privacy Programs](#)
[17.7 DID Products](#)
[17.8 Summary](#)
[References](#)
[Quiz](#)
[Answers](#)

18. Tokenization, DeFi, NFT and CBDC

[18.1 Decentralized Finance, Tokenization and NFT](#)
[18.1.1 NFT Craze for Digital Contents](#)
[18.1.2 Blockchain & DLT Platforms](#)
[18.2 Stablecoins](#)

[18.2.1 Pros and Cons of Stablecoins](#)
[18.3 CBDC](#)

[18.3.1 CBDC Types](#)

[18.3.1.1 Wholesale CBDC](#)

[18.3.1.2 Retail CBDC](#)

[18.3.1.2.1 Indirect CBDC](#)

[18.3.1.2.2 Direct CBDC](#)

[18.3.1.2.3 Hybrid CBDC](#)

[18.3.2 Blockchain & DLT Platforms](#)

[18.3.3 Research on CBDC](#)

[18.4 Summary](#)

[References](#)

[Quiz](#)

[Answers](#)

[19. Blockchain and 5G for IoT](#)

[19.1 How IoT Works](#)

[19.2 Nextgen IoT](#)

[19.3 Summary](#)

[References](#)

[PHASE V – Blockchain Adoption and Future](#)

[20. Production and Beyond](#)

[20.1 Challenges in Blockchain projects](#)

[20.2 Blockchain CoE Set-Up](#)

[20.2.1 Accelerators in Blockchain adoption](#)

[20.2.2 Roles](#)

[20.2.3 Skills](#)

[20.2.4 Factors to Consider for Production](#)

[20.3 Blockchain Standards](#)

[20.3.1 W3C](#)

[20.3.2 Open Standards](#)

[20.3.3 European Standardization Organization](#)

[20.3.4 ISO/TC 307](#)

[20.4 Quantum Computing, The Fear Factor](#)

[20.5 Summary](#)

[References](#)

[Glossary.](#)

[Index](#)

CHAPTER 1

Introduction to Blockchain

Blockchain is one of the latest revolutions of the 21st century, many of whose fruits are yet to be harvested.

In this chapter, we will cover the introduction to Blockchain, the hype of Blockchain, and all the inherent and adjoining technologies that the decentralized architectures are dependent on. This knowledge would help us in the later chapters to understand the inherent mechanisms of different Blockchain protocols and choose the one that best suits our business needs.

1.1 The Blockchain Market

As per a 2020 report by Markets and Markets, “the global Blockchain market to be \$3.0B in 2020 and is projected to reach \$39.7B by 2025. The market growth can be attributed to the increasing number of venture funding and investments in the Blockchain technology and the increasing popularity of the Blockchain in retail and supply chain management.”

This report is seconded by another market analysis report by “Grand view research” that has the opinion that, “The global Blockchain technology market size was valued at USD 3.67 billion in 2020. It is expected to expand at a compound annual growth rate (CAGR) of 82.4% from 2021 to 2028.”

While North America has been the major investor that contributes to nearly 40% of the global revenue, we can find regional interest as well. And though the interest started with Banking and Fintech, today every business vertical seems to be looking for opportunities with the Blockchain technologies.

1.2 Evolution of Blockchain Technology and Hype

Blockchain is already a decade old, and hence, for the people who look for opportunities with this technology, it's absolutely needed to understand the latest developments and the trends in this space.

One of the best place to do so is to keep a close eye on the “Gartner Hype Cycle” that registers all the emerging technologies and religiously follows their developments. Any such technology as per Gartner would first go through five phases namely innovation trigger, followed by a peak of inflected expectation, trough of disillusionment, slope of enlightenment, and finally the plateau of productivity. Different technologies or use cases take different amount of time to travel from the innovation trigger till productivity. [Figure 1.1](#) and [Figure 1.2](#) shows the Gartner hype cycles for the past 2 years, where they have especially highlighted the Blockchain use cases and their estimated timelines for reaching productivity, as shown as follows:

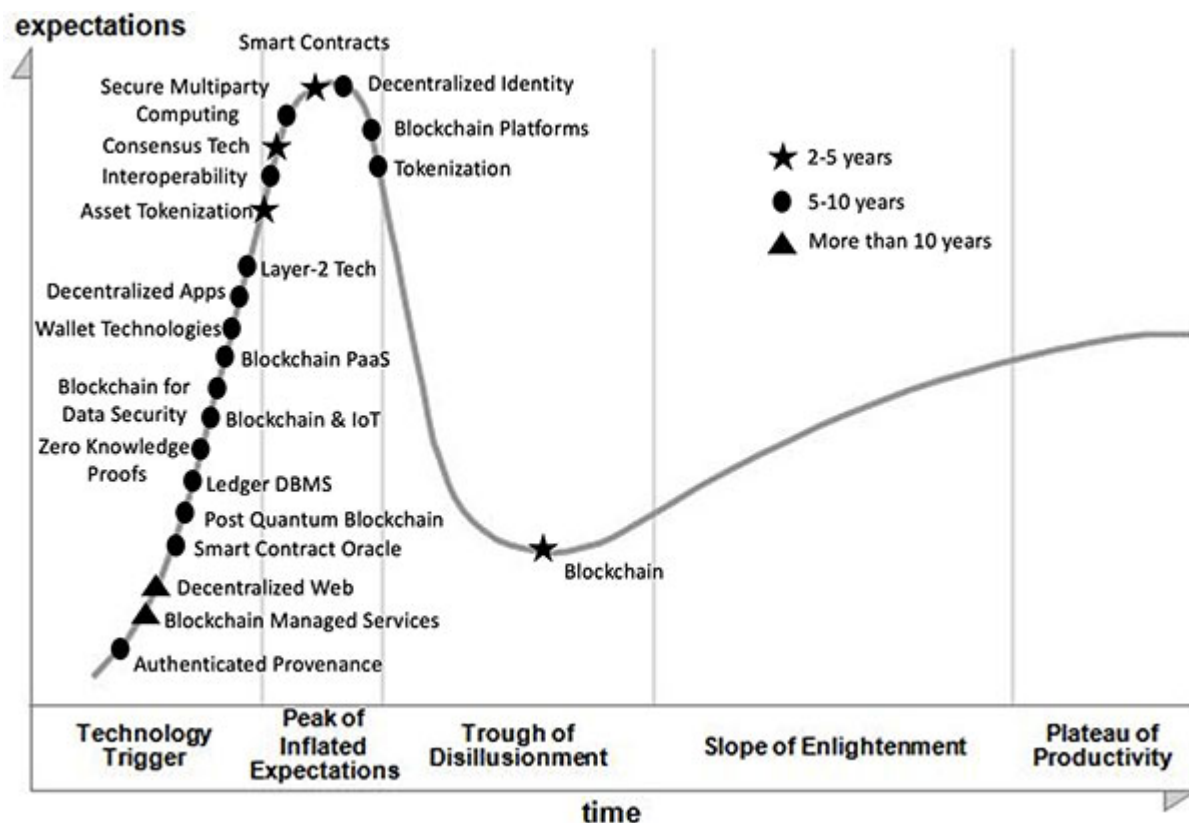


Figure 1.1: Gartner Hype Cycle for Blockchain 2020 (Source: Gartner)

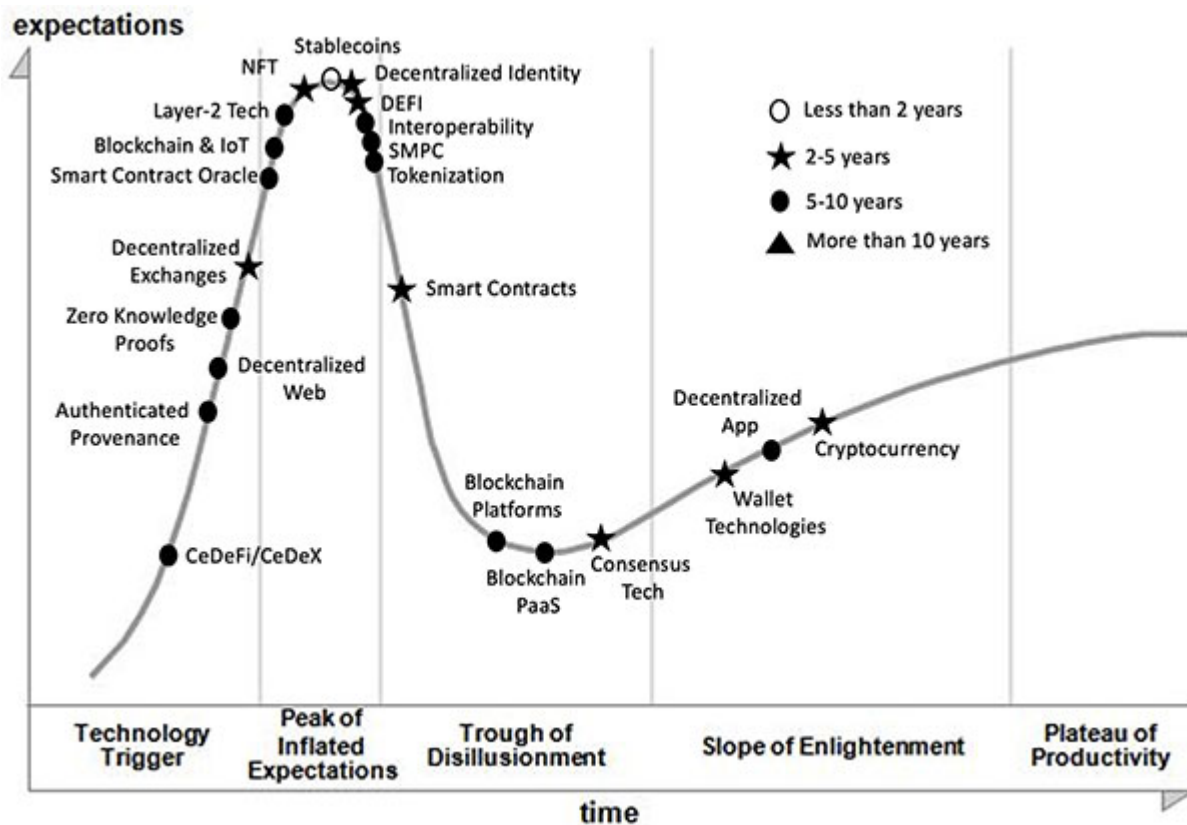


Figure 1.2: Gartner Hype Cycle for Blockchain 2021 (Source: Gartner)

Blockchain was introduced for the first time in Gartner Hype Cycle in 2016, and in just under five years, many of its use cases are in production. This signifies that in comparison to many others emerging technologies, the adoption of Blockchain is much faster. In the later chapters, we will read about many of the Blockchain use cases along with their supporting products and technologies that would help us understand the entire ecosystem with clarity.

1.3 Birth of Bitcoin

Unlike some of the other emerging technologies as Artificial Intelligence, IoT, Robotics etc., Blockchain is a relatively new technology that was introduced almost a decade ago through a cryptocurrency called “Bitcoin”. In 2008, someone named Satoshi Nakamoto published a whitepaper called “Bitcoin: A Peer-to-Peer Electronic Cash System”, that described the idea of Bitcoin

cryptocurrency and Blockchain as the new technology to back such a currency. Within a year, Nakamoto released the first version of Bitcoin in the market that was also associated with the first implementation of the Blockchain so far. The first block called the “Genesis Block” was added to the Blockchain network on 3rd January 2009, and the first Bitcoin transaction happened on 12th January 2009. By year 2010, Satoshi vanished from the scene totally stopping his involvement in the associated activities. The price of the Bitcoin as well as the demand of the Blockchain as a technology, however, never looked back. In just over a decade, Bitcoin has observed exponential growth from zero to roughly \$2+ trillion which is approximately 0.5% of the global wealth. The person behind the name, Satoshi Nakamoto, the father of Bitcoin, still remains a mystery. Was he a man, a woman, a group of people, an alien, or even a time traveller from the future, who gifted us this superior technology, and then simply vanished? The genius always prefers to stay away from the limelight, or does he?

1.4 Bitcoin vs. Previous Generation Electronic Money

Bitcoin is not the first digital money, nor will it be the last. Since the 1980s, many people have tried their hands on digital or electronic currencies but have failed due to technical or regulatory reasons. Organizations such as PayPal, eCash, WebMoney, Liberty Reserve, Payoneer, cashU etc., tried to achieve similar results but through centralized ecosystems which come with their own shortcomings such as the following:

- Possibility of mass hacking
- Single point of failure
- High transaction fees

What made Bitcoin a shining star in the space of electronic funds transfer was its unique features. A few surprising benefits because of which Bitcoin became a milestone in payments are as follows:

- Immunity to Fraud

- No single point of failure
- Low transaction fees
- Universally acceptable
- Instantaneous Settlements
- Prevents Identity Theft

Today, there are more than 6000 cryptocurrencies being traded in the market and most of them use the concept of decentralization in one way or the other. All thanks goes to Satoshi and his concept of decentralized architecture in Blockchain, which is the essence behind the growth of cryptocurrencies.

1.5 Key Concepts

Unlike popular belief, Blockchain is not a new technology; rather it's an amalgamation of many existing technologies along with a few new. Hence, let's explore the basic elements and key concepts of Blockchain before diving deeper into the next chapters.

1.5.1 Cryptography, Encryption, and Decryption

Cryptography is the backbone of every Blockchain, and hence, it's expected that we should have sound knowledge of its basic principles first.

Cryptography is the science of secure communication where two people can send and receive information openly using some sort of algorithm, so that the others cannot interpret the message even if they have access to it. The concept of Cryptography is nothing new. One of the earliest mention can be found as early as around 100 BC when the Roman general Julius Caesar used such techniques to send messages to his generals. Over the years, though many new cryptographic algorithms have been developed, but so have their hacking methodologies.

Every algorithm in Cryptography is divided into two different phases, which are as follows:

- **Encryption** or transforming the original data to a cryptic or uninterpretable format.
- **Decryption** or converting the encrypted data back to its original form.

In each cryptographic algorithm, we use two different types of cryptographic keys for the encryption and decryption of the data, which are as follows:

- Symmetric cryptography
- Asymmetric cryptography

[1.5.1.1 Symmetric Cryptography](#)

In symmetric cryptography, there is only one cryptographic key that is used for both encrypting as well as decrypting the data. This approach is easier and yet has limited functionalities and use cases.

[1.5.1.2 Asymmetric Cryptography](#)

Public-key cryptography, also known as asymmetric cryptography, uses a pair of keys, which are as follows:

- Public key, that can be visible to others
- Private key, that stays only with the owner

While one key can be used to lock the message, the other can unlock it. These two keys can be used in a variety of different ways; look at the following example:

- Alice wishes to send a secret message to Bob. Alice would encrypt the transaction with Bob's public key and only Bob can decrypt it using his own private key.
- Alice signs a document using her private key, also known as "**Digital Signature**". The others who know Alice's public key can validate that the document is actually signed by Alice.

The following are the different types of asymmetric cryptography:

[1.5.1.2.1 RSA](#)

Introduced in 1977, Rivest–Shamir–Adleman or RSA is one among the first public-key Cryptography used for secure data transmission. This cryptography depends on solving a factoring problem with two large prime numbers. Though RSA is tough to breach, compared to DSA, it's faster for its signature validation and slower in generation. Being there for a long time, RSA is most widely used and best supported.

1.5.1.2.2 DSA

Digital Signature Algorithm or DSA, a variant of the Schnorr and ElGamal signature was proposed by the National Institute of Standards and Technology (NIST) in 1991. Based on a discrete logarithmic problem, it's still used by Federal Information Processing Standard for digital signatures. In comparison to RSA, it's faster for signature generation but slower for validation.

1.5.1.2.3 Elliptic curve cryptography and ECDSA

Discovered in 1985, Elliptic curve cryptography or ECC is a contemporary powerful approach to public-key cryptography that relies on mathematical elliptic curves. It's adopted by Bitcoin and many other cryptocurrencies and Blockchain for creating smaller, faster, and more efficient cryptographic keys.

Elliptic Curve implementation of DSA or ECDSA has managed to provide similar security levels as RSA with a shorter encryption key. Hence, it needs less computing power leading to a faster processing than the keys of the previous generation. One of the disadvantages though is that it makes the size of the encrypted messages much bigger. Also, it's much more complex to implement in comparison to RSA.

1.5.1.2.4 Edward curve Cryptography and EdDSA

In Cryptography, Edward curves are a family of Elliptic curves introduced in 2008 that has been utilised in Edwards-curve Digital Signature Algorithm or EdDSA. Using intractable discrete logarithmic problems, EdDSA is safer than the previous cryptographic versions as DSA and ECDSA. Being one of the latest work in cryptography, it

lacks standardization and yet has enough use in the market as the others.

1.5.1.2.5 Ed25519

Ed25519 is a type of EdDSA signature algorithm and yet depends on SHA-512/256 hashing and Curve25519 elliptic curve. It's one of the latest digital signatures and is available in the market only since 2011. It's safer and faster than DSA, ECDSA, and EdDSA, plus comes with a much shorter version than the conventional keys. As per some cryptography experts, it can be broken by quantum computers in the future; but till then, it seems to be the safest and the best.

Note: *Certificate authorities work on the principles of asymmetric cryptography to issue digital certificates and also validate them.*

Note: *X.509 is an International Telecommunication Union (ITU) standard for public key certificates that binds an identity to a public key using a digital signature. It's heavily used in different Blockchains and DLT networks.*

1.5.2 Hashing

Hashing is one of the ways in which using encryption algorithm, a string can be converted into another fixed length string as the output; however, the output string cannot be converted back to the original value. Hence, hashing is irreversible and always comes with unique outputs with the same length of characters. Hashing has been heavily used in the past in storing the passwords in the databases, so that none would be able to decipher the original value, limiting the chances of fraud.

National Institute of Standards and Technology (NIST) has published a group of cryptographic hash functions also as a standard for the U.S. Federal Information Processing Standard (FIPS). Some of them are MD5, SHA-256, SHA-512 etc.

1.5.3 Wallet

Most Blockchains and cryptocurrencies are associated with a private space for storing the user's cryptographic keys or private data that helps the user in sending and receiving the digital currency with the others. These wallets are popularly known as cryptocurrency wallets or crypto wallets. Such wallets digitally store a user's public and private keys and programmatically helps in sending and receiving digital currency or sensitive information. Let's explore the different types of wallets and their pros and cons on the basis of the type of device they are associated with.

1.5.3.1 Online wallets

Online wallets, as the name suggests, are wallets where the cryptographic keys are saved to an online application and can be accessed through a browser. It's one of the most convenient ways to store and use the keys on, as per the requirement. Some of the examples of such wallets are Exodus, Ledger Nano X, Multibit etc. Being available on the online websites, these wallets are prone to security threats. Hence, most online wallets have strong password policies as well as multi-factor authentication system that might come in different combinations of password, biometrics, and OTP.

1.5.3.2 Desktop wallets

Desktop wallets are offline wallets that can be downloaded and saved on a computer. We just need to make sure that the computer is secure enough and password protected, also that it is not shared with others. Desktop wallets are far more secure than the online wallets; however, the data stored in such wallets are often not portable.

1.5.3.3 Mobile wallets

Mobile wallets are similar to desktop wallets; however, the underlying data is portable. Hence, they are the most popular form of wallets in today's world.

1.5.3.4 Hardware wallets

Similar to a mobile wallet, the data in a hardware wallet is portable, secure, and convenient to use. However, they cannot be used on the fly with other apps, something that we can do with the mobile wallets.

1.5.3.1.5 Paper wallets

Paper wallets can also be used to store the keys in the printed form or as a QRCode. They can be stored in such a way that none but the owner has access to the data. They are useful in a low-tech environment, especially for users who do not have a mobile phone or desktop or access to the Internet. However, the drawback is that the data in such wallets cannot be integrated with the online apps and hence can't be fully utilised in all scenarios.

Crypto wallets can also be categorized on the basis of their type of ownership.

1.5.3.1.6 Non-custodial wallets

The non-custodial type of wallet gives a user the complete ownership of private keys and associated cryptocurrencies. While this approach gives absolute security to the users, at the same time, it's risky, as some unexperienced first-time users may lose their private keys and the underlying assets along with it if the password is lost.

1.5.3.1.7 Custodial wallets

A custodial wallet is a different type of wallet where the user relies on a third party to store the private keys as a custodian. While with the custodial wallet, the responsibility of the user is far less in handling the keys, this solution might not work for all use cases as the user needs to fully trust the third party for the security of the keys. Hence, these wallets are considered less secure than the non-custodial types, and yet they are preferred by many, especially those who either are not tech savvy, or do not have the time to handle it all on their own.

Most crypto exchanges offer web based or mobile app based custodial wallets. These wallets also come with password reset and recovery features, similar to what we have in most websites.

1.5.4 Decentralization

Before learning about Blockchain, it's absolutely important to understand the difference between a centralized vs. the other types of ecosystems. [Figure 1.3](#) demonstrates the differences in their architectures, as shown as follows:

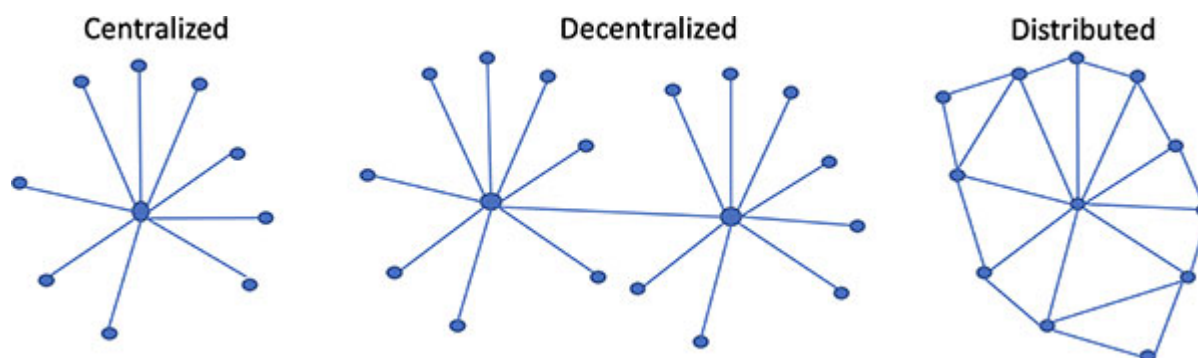


Figure 1.3: Centralized vs. Decentralized vs. Distributed servers

Let's explore each of them in detail.

1.5.4.1 Centralized System

A centralized system is managed by a centralized server that handles all the external requests on its own. Centralized ecosystems are relatively simpler to architect and control. However, the disadvantages are the possibility of a single point of failure, mass hacking, massive load, and monopoly resulting in a higher fees for the transaction.

1.5.4.2 Distributed System

A distributed system refers to the **location**, i.e. the different components of the ecosystem such as data bases, servers etc., are located at different locations, possibly with their backups, so that if one goes down, the other one can take over. The advantages are high scalability, low latency, and fault tolerance. The distributed

systems are not a new concept and have been running in production for decades. Please note that a distributed system can also be centralized as the ownership of the ecosystem can still be in the hands of one party.

1.5.4.3 Decentralized System

Decentralization, on the other hand, is associated with **control**. In a decentralized ecosystem, the decisive power is divided among many or, let's just say, that the ecosystem is together owned by many. When it comes to location, the decentralized ecosystem can have one or many different locations of the data and their respective ownership.

Table 1.1 is a comparative chart on the advantages and disadvantages of the three ecosystems, as follows:

Type of Ecosystem	Advantages	Disadvantages
Centralized	<ul style="list-style-type: none"> • Dedicated resources, hence less expensive • Simple to architect, configure, develop, test, debug, and maintain • Peace of mind with centralized consensus 	<ul style="list-style-type: none"> • Single point of failure and mass hacking possible • Monopoly possible resulting in higher fees for transaction • Performance and scalability always a bottleneck, and hence inappropriate for bigger ecosystems with higher complexity
Distributed	<ul style="list-style-type: none"> • Can be highly scalable horizontally • Fault tolerant • Low latency 	<ul style="list-style-type: none"> • More complex to architect, configure, develop, test, debug, and maintain • Synchronization and consistency are always a challenge
		<ul style="list-style-type: none"> • Network availability can face a challenge
Decentralized	<ul style="list-style-type: none"> • Scales moderately and 	<ul style="list-style-type: none"> • More complex to architect,

	horizontally <ul style="list-style-type: none"> • Chances of monopoly is less • Regulation and maintenance is a perpetual challenge 	configure, develop, test, debug, and maintain <ul style="list-style-type: none"> • Consensus of all players may face a challenge • Hence synchronization and consistency won't be as smooth as the centralized systems
--	---	--

Table 1.1: Comparison of features of centralized, distributed, and decentralized servers

1.5.5 Consensus

Consensus is a process through which the owner of an ecosystem agrees for a transaction to get committed in the data store. In a centralized system, it hardly matters as the ownership itself is centralized. However, in Blockchains, it's pretty complex as there are multiple stakeholders of the ecosystem and they all have to agree on the validity of the transaction. Let's discuss the different types of consensus models that have been used so far in the different decentralized ecosystems.

1.5.5.1 Proof of Work (PoW)

PoW is the original consensus process of Bitcoin and Ethereum where every node is an equal participant in the consensus process. The final consensus is arrived at after the fastest validator node wins, followed by a voting process of all the nodes. Such a model is slow as well as resource intensive, and hence, not fit for use by most of the industry use cases.

1.5.5.2 Proof of Stake (PoS)

Proof of Stake is an energy-saving consensus model that has been adopted in some of the later Blockchain protocols that appeared in the market post Bitcoin. Here, only the nodes that have invested some stake in the Blockchain network are eligible to participate in the validating transactions. In this process, they get rewarded with

the native currency of the Blockchain. It's usually believed that people who have invested in a network heavily would be less likely to harm the network on their own. Such consensus process is easily faster, scalable, and more energy efficient than PoW.

1.5.5.3 Delegated Proof of Stake (DPOS)

In DPOS, only some elected subsets of all the stakeholder nodes are allowed to validate the transactions. This process of election keeps on happening in regular time interval, so that all the nodes would get their turn to be validators at some interval. Such a consensus process is even faster and more efficient than PoS, as less number of nodes take the decision as they take lesser time to reach a joint consensus.

1.5.5.4 Practical Byzantine Fault Tolerance or PBFT

Before discussing PBFT, we must understand BFT or "Byzantine Fault Tolerance". BFT refers to the Byzantine Generals' Problem, widely used in computer science for a war situation where multiple generals of the same party must agree on a single war strategy to attack a city; however, it also considers the risk that some of these generals might be corrupt or not trustworthy.

When it comes to a distributed system, BFT is a feature for fault tolerance or resistance to failure, even if some of the nodes fail or act maliciously. There are many different solutions to handle this type of issue; however, PBFT is considered as the gold standard. Practical Byzantine Fault Tolerance or PBFT is a mechanism through which a collective decision can still be made by all the nodes even in a war-like difficult situation. In PBFT, at one time, there can be one leader node and the rest will work as followers; also, the follower nodes can take their turn to be a leader node. When a client sends a request, it is attended by the leader node of that time. The leader node broadcasts the message to all the follower nodes who send the responses back to the client. The client has to wait till the "N+1" responses from different nodes, where "N" is the maximum number

of possible nodes. In the Blockchain and DLT world, this consensus mechanism became extremely popular when it was adopted by **Hyperledger Fabric**.

1.5.5.5 Tendermint

Tendermint is an opensource software that brings the combination of a Byzantine Fault Tolerant Blockchain consensus engine and a p2p networking protocol. It is used and popularized by the **Cosmos** interoperable Blockchain.

1.5.5.6 Direct Acrylic Graph

A Direct Acyclic Graph or DAG is a similar network of nodes, just like Blockchain; however, its underlying consensus mechanism is far different. In Mathematics, DAG is a special type of graph whose edge moves in just one direction and can't go back, hence the name is justified. In DAG, for any new transaction to be submitted, at least two previous transactions have to be confirmed. Hence, more the number of transactions that are submitted, more would be confirmed, and hence the chain would grow faster. Obviously, DAGs would be faster and much more scalable than many of the Blockchains and would require negligible fees (as there are no miners). One of the most popular DLTs leveraging the benefits of DAG is **Hedera Hashgraph**.

1.5.6 Nodes

In Blockchain, the nodes are devices or platforms connected to the common network that are owned by different individuals or organizations or parties. The developers can run their services on these independent platforms.

1.5.7 Shared Replicated Ledger

Each of the nodes on Blockchain are backed by an independent datastore. As already discussed, all these nodes collectively reach a consensus to accept or reject a request for a transaction. Also,

doesn't matter which node tries to add the data, it gets replicated across all the nodes in the Blockchain network in near real time.

1.5.8 Smart Contracts

Smart contracts are self-executing contracts written to the code. In other words, they are business logics or algorithms that are deployed on all the nodes of the Blockchain. These contracts are mostly used to validate and execute the transaction request as per the previously programmed logic. Please note that smart contracts can be used to automate credible transactions between different parties without the need of intermediaries or manual interventions, that are often required in traditional transactions. In some use cases, we can also make smart contracts execute only on a subset of nodes, rather than on all.

1.5.9 Native Token

Most public Blockchains are associated with a native token that is defined by the network's protocol. The creation, transfer, redeeming, or burning of these tokens can be handled by smart contracts. A part of the token is usually allocated to the validators of the network. At the same time, all the participants, whosoever wishes to run a decentralized application on the Blockchain network, has to pay in terms of the token. These payments in tokens also keep the malicious actors at bay and keeps the public network safe against the DDoS attacks.

Ideally, these tokens can be purchased at the crypto exchanges with traditional or fiat currencies such as USA, GBP etc. The name of some of the biggest crypto exchanges are Binance, Coinbase Exchange, ZT, CuCoin, Upbit, FTX, Bitbank, Bitso, Kraken, Bitbank, IndoEx etc.

Some of these exchanges also allow to trade between different major cryptocurrencies such as Bitcoin and Ether etc. Please note that these tokens are also cryptocurrencies that get traded in the exchange and their prices go up or down just like traditional assets depending upon the demand and supply in the market. Some of the

examples of these cryptocurrencies are Bitcoin, Ether, Cardano, Polkadot, XRP etc.

Please note that the native tokens are not the same as general tokens (fungible or non-fungible) that you might have heard of. The latter can be created and utilized on a Blockchain platform with the help of programming. We shall discuss more on them in [Chapter 18](#).

1.5.10 Gossip Network for Message Propagation

[Figure 1.4](#) illustrates how most Blockchain networks consist of a huge number of untrusting nodes who receive the data updates through a gossip network, as shown as follows:

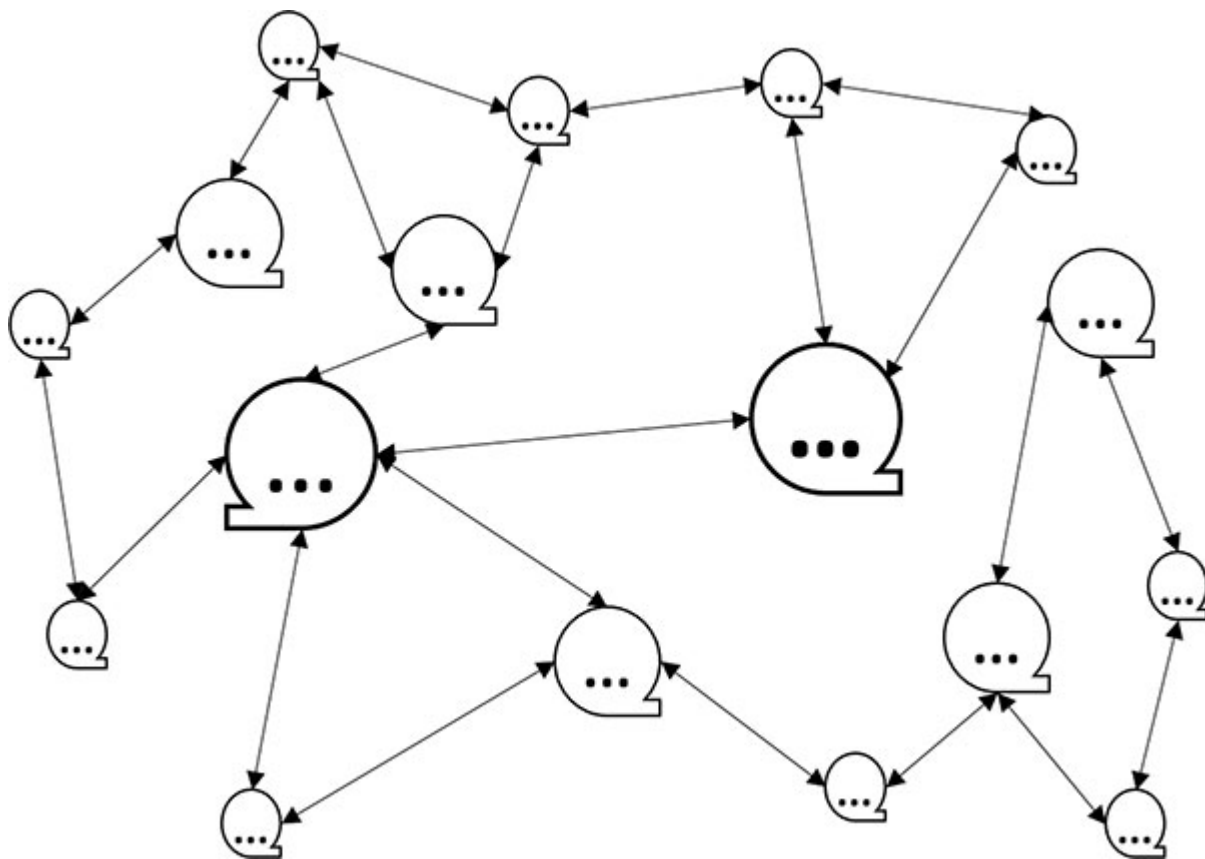


Figure 1.4: Gossip Network

This network purely relies on peer-to-peer messaging, and hence, any data gets communicated just like a forest fire or how an epidemic spreads itself.

1.5.11 Hard Fork and Soft Fork

Public Blockchains often undergo forks. These forks are of two types, i.e., hard fork and soft fork. Hard fork is a serious change in a Blockchain ecosystem where the primary Blockchain splits into more than one protocol with different cryptocurrencies.

As shown in [Figure 1.5](#), a hard fork is an important concept in Blockchain that can occur by accident or with careful planning. When different parties in a Blockchain disagree on certain concepts of the Blockchain, that leads to the emergence of different hard forks. Bitcoin Classic, Bitcoin Cash, Bitcoin Gold etc., are the different hard forks of Bitcoin. Refer to [Figure 1.5](#), as follows:

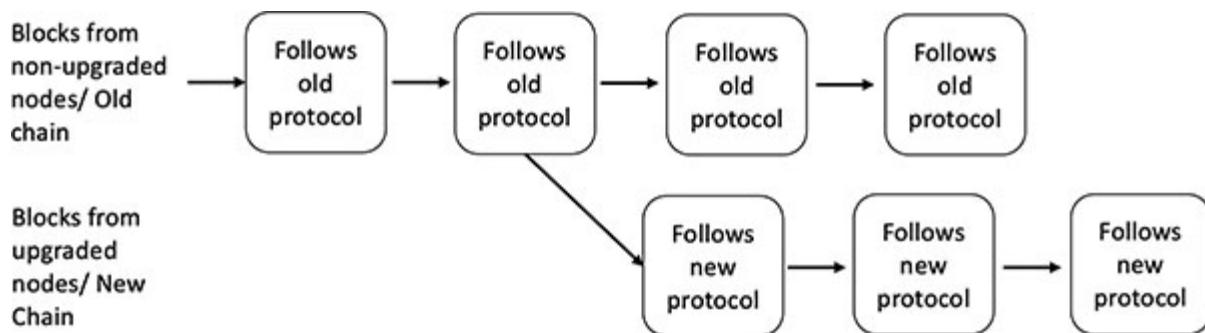


Figure 1.5: Hard Fork

As represented in [Figure 1.6](#), Soft forks are backward-compatible temporary divergences from the main chain, where some previously valid transaction blocks are invalidated. It happens when the old nodes do not agree to a rule followed by the newly upgraded nodes. Refer to [Figure 1.6](#), as follows:

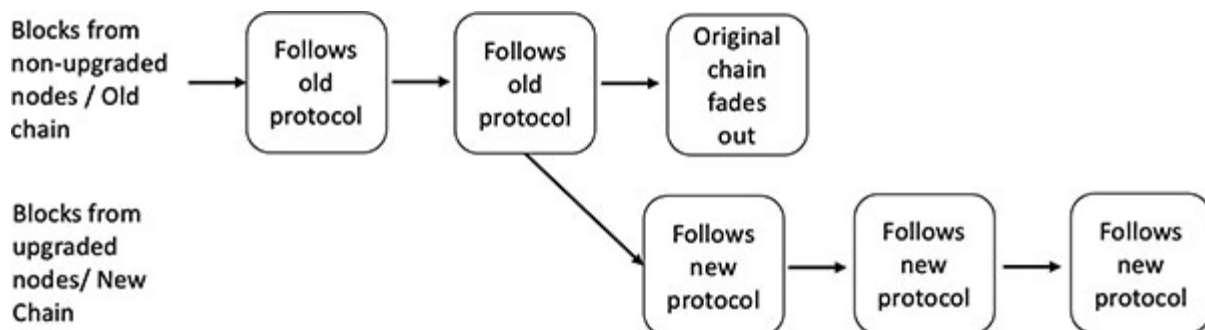


Figure 1.6: Soft Fork

After discussing all the basic components of Blockchain, it's time to give a definition to Blockchain.

Blockchain is a distributed ledger with a decentralized consensus where the participant nodes can send the transaction requests to the ledger that can be added and replicated to the other nodes in near real time. The data in a Blockchain is immutable or can be added in the append-only mode; hence, it's transparent and cryptographically secure.

1.6 How Bitcoin Blockchain Works

The description of Blockchain that we just covered is as per the original concept of the technology as introduced by Satoshi in Bitcoin. Let's now explore how the data is added to such a Blockchain ledger. The Blockchain transactions are linked to each other in a unique data structure known as the Merkle tree. As shown in [Figure 1.7](#), the hash of each transaction is calculated and paired with the hash of the next transaction. Now, the collective hash of these two hashes is again paired with another collective hash. This process is repeated till we reach the root hash, which is also known as the **Merkle root** that gets saved to the Block header. Refer to [Figure 1.7](#) as follows:

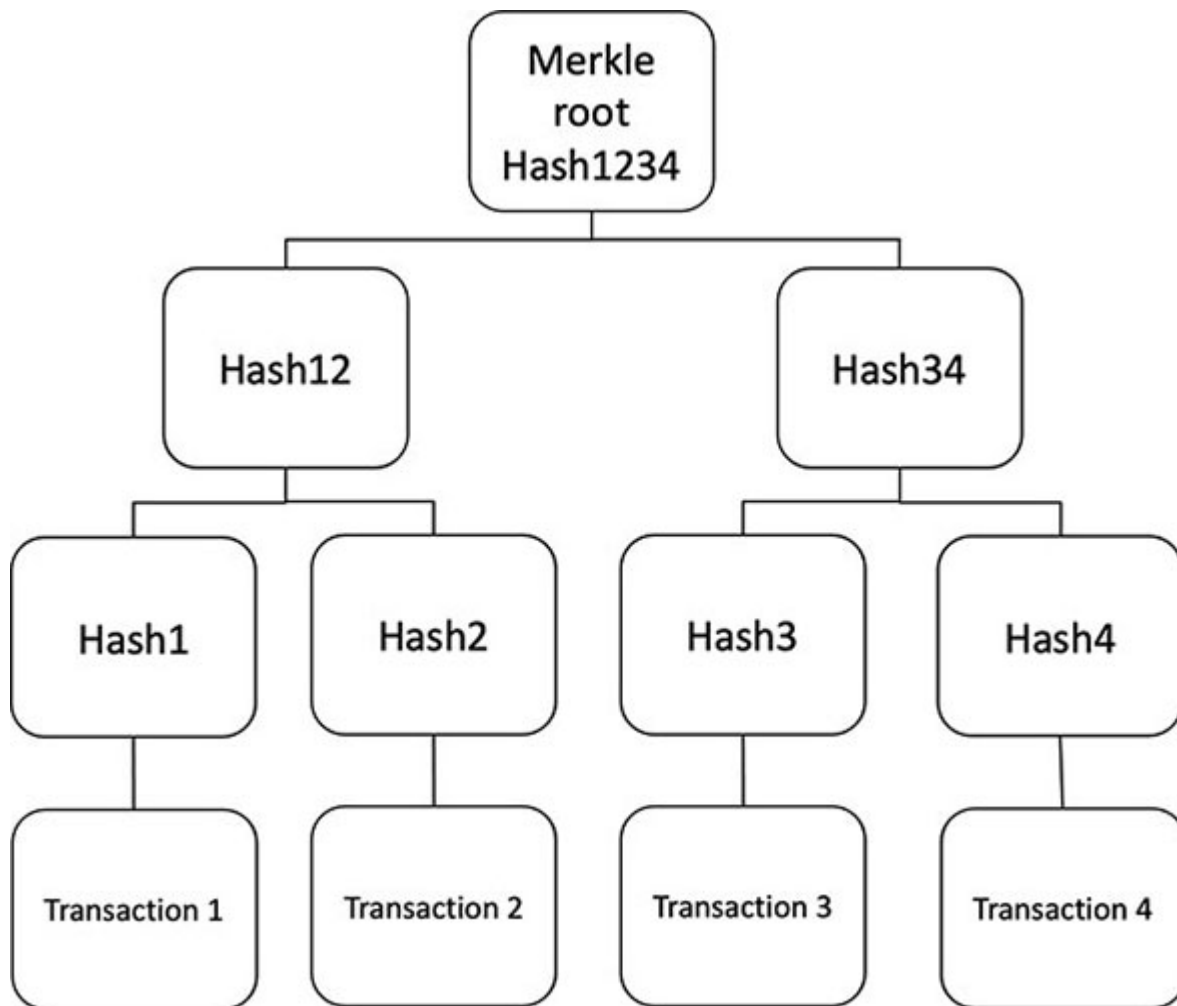


Figure 1.7: Merkle root

As the name suggests, a Blockchain is a chain of blocks, where each block points to a previous block. As shown in [Figure 1.8](#), each block has a block Header that in turn consists of the Merkle root, the hash of the last block, and a nonce. If any of the data changes, then the Merkle root of the block would alter and that would not match with the hash in the next block which is stored as the “Hash of previous block”. Refer to [Figure 1.8](#), as follows:

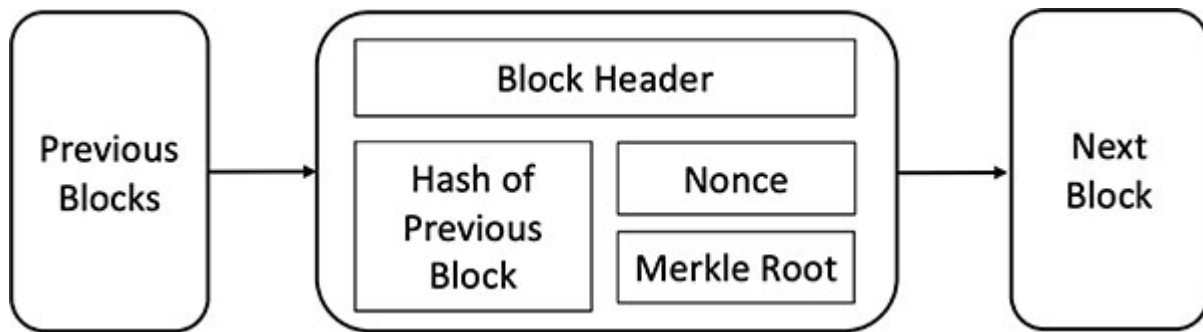


Figure 1.8: Blockchain or a linked chain of blocks

Thus, the integrity of the data is maintained and the data once added can be considered as tamper-proof. Please note that the nonce or “number only used once” is a randomly generated number for each block and is mostly used in cryptographic communication and information technology.

1.6.1 Mining

We all know that Blockchain is a distributed ledger with more than one owner. Now saying so, the following questions come up:

- How the transactions would be validated?
- Which among the owners would be responsible to validate the transactions?
- Even if some of the owners agree to do the job, what would they gain from such an intensive process?

Well, the answer is that all the Blockchains have a decentralized consensus process to validate the transactions where all or more than one parties get involved. For the Bitcoin network, the owner of the nodes themselves participate to earn a fraction of the token. So, mining is a process of the validation of transactions, adding them to the ledger, and confirming the same to the rest of the nodes in the network. Mining is intentionally designed to be resource-intensive and difficult, so that the number of blocks found each day by the miners remains steady and the spam attacks could be controlled. So, mining serves the following two purposes:

1. To verify the legitimacy of a transaction, or avoiding the so-called double-spending.
2. To create new digital currencies by rewarding the miners for performing the previous task.

1.6.2 Proof of Work

We already explored how easy it is to add a block to an existing Blockchain. In one minute, thousands, if not millions, of blocks could be added to the Blockchain network with such a process. In order to minimize this, so that there is no spam and a block is properly validated with adequate time, a mining mechanism called “proof of work” is added to the system.

Individual blocks must contain a proof of work to be considered valid. This proof of work is verified by the other Bitcoin nodes, each time receiving a block. Bitcoin uses the hashcash proof-of-work function, which is described as follows:

- Transactions are bundled together into what we call a block.
- The miners verify that the transactions within each block are legitimate.
- To do so, the miners should solve a mathematical puzzle known as the proof-of-work problem.
- A reward is given to the first miner who solves each block’s problem.
- The verified transactions are stored in the public Blockchain.
- When a miner finally finds the right solution, he/she announces it to the whole network, at the same time, receiving a cryptocurrency prize (the reward) provided by the protocol.

As the difficulty level increases over time, the miners use more powerful machines to calculate the correct hash as soon as possible to come out as the winner. Currently, the time taken in the Bitcoin Blockchain to add a block is around 10 minutes.

1.6.3 Consensus with Proof of Work

As soon as a new transaction is added, it is broadcasted to all the nodes and the mining nodes start working to solve the proof-of-work puzzle. Whosoever finishes it first, broadcasts it to the others. The nodes accept the block only if all the transactions in it are valid and not already spent. The transaction is confirmed and added to the Blockchain only if a minimum percentage of all the nodes accept it. For the Bitcoin, originally the percentage was 51%, matching the democratic voting models in the world. However, soon it was observed that more than 51% percent of the computing power in the Bitcoin network lies with a handful of miners today.

51% attack is a well-known attack, especially in Bitcoin or any other similar Blockchain network, where one or a group of nodes try to take full control of the network by acquiring the ownership of more than 50% of the nodes.

Hence, people soon tried to increase this consensus percentage from 51% to a higher percentage. Ripple is another Blockchain product that today uses an 80% consensus model.

Note, Bitcoin is not the only cryptocurrency that works on the Proof-of-work consensus model; there are hundreds of others such as Ethereum 1.0, Dogecoin, Litecoin, Bitcoin Cash, Ethereum Classic, Monero etc.

A complete list of the PoW based Blockchain protocols can be found at the following link: <https://coinmarketcap.com/view/pow/>

1.6.4 DAPPS

In the preceding example, we saw how the data or a block carrying the transaction(s) are added to a Blockchain. But that's not all. A Blockchain works in the Decentralized mode and the technology is known as DAPPS or Decentralized Applications. It's implemented on a network where every node keeps either a whole or a partial copy of the Blockchain. This leads to a greater transparency and no single point of failure for the transactions.

1.7 Threats & Challenges in Public Blockchains

Public Blockchains are open for all and hence the chances of cyber-attacks as well as many other threats are to be addressed before it is eligible for production.

1.7.1 Double Spending

Unlike the fiat/physical cash, which cannot be spent twice, one can play with the crypto-currencies trying to spend the same money twice in a quick succession. Let's explain this with an example.

Tom and Harry agree to do a transaction worth \$10 where Tom has to pay Harry the amount. Tom and Laura too agree for another transaction worth \$10 where Tom has to pay Laura the amount.

But Tom only has \$10 in his account. He first sends the money to Harry. We already saw that the transaction is evaluated by the miners and it takes some time to get confirmed. But before the confirmation, he sends the amount again to Laura. So, now we have two unconfirmed transactions in the pool. Such a type of fraud is called double spending.

In order to avoid this, Blockchain keeps a timestamp of each transaction. So, for the sender to revert the first transaction and send it again, he needs to tamper with the Blockchain data by altering the whole chain, considering the fact that the hashes are saved in the following blocks, which is next to impossible.

1.7.2 Sybil Attack

[*Figure 1.9*](#) illustrates a sybil attack, where a hacker somehow gets the access to multiple nodes instead of one in a public Blockchain, and tries to manipulate the control of the network, as shown as follows:

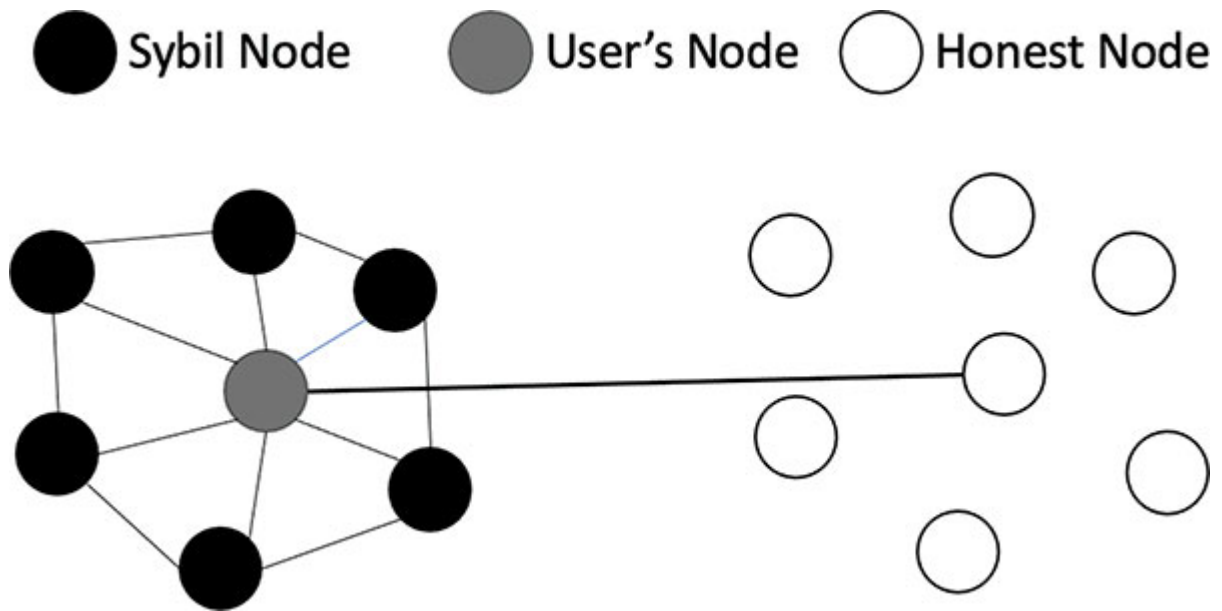


Figure 1.9: Sybil Attack

Such manipulations can result in a 51% attack in the Bitcoin or similar networks running with a PoW consensus. Such attacks can be prohibited in a permissioned Blockchain after a background verification of each participant.

1.7.3 DDoS Attack

[Figure 1.10](#) illustrates the DDoS or a distributed denial of service, which is a cyber-attack where the hacker uses bots to send a huge number of requests to the specific server and totally consume all of its resources, as shown as follows:

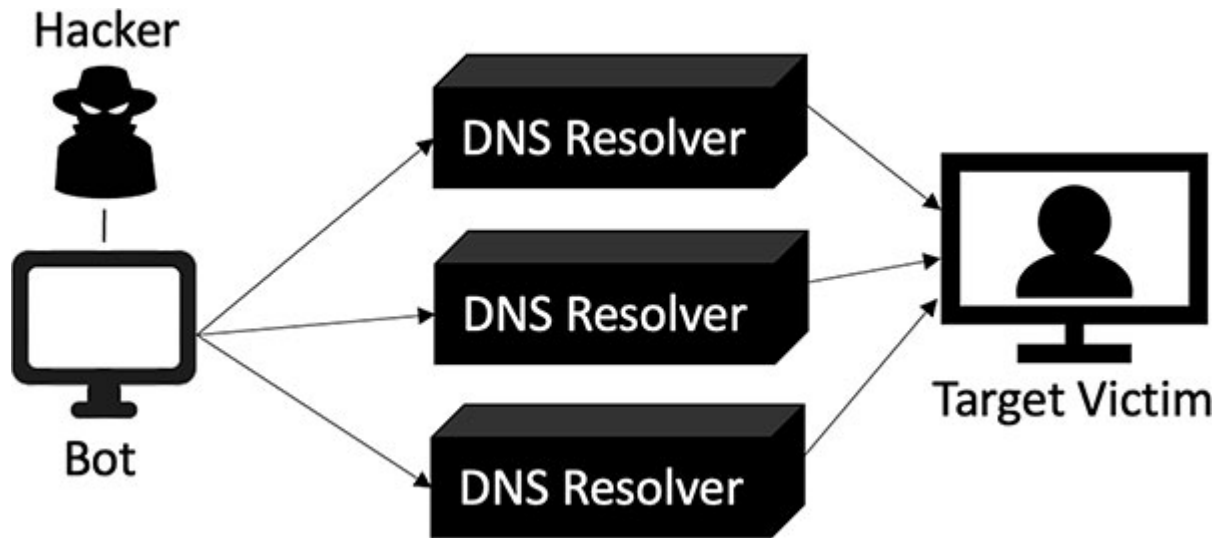


Figure 1.10: DDoS Attack

If the requests are more than the maximum capacity that the server can handle, then the server goes unavailable for further requests. This is pretty common in the Internet shopping sites or organizations offering online services. Many Blockchain backed cryptocurrency exchanges have faced such attacks and have gone offline from time to time. Hence, many public Blockchains have started moving from a permissionless to a permissioned architecture. Let's now discuss the different levels of permissions in the Blockchain protocols.

1.8 Permission Levels

As shown in [Figure 1.11](#), DLTs can be divided into four different types on the basis of their permission levels, as follows:

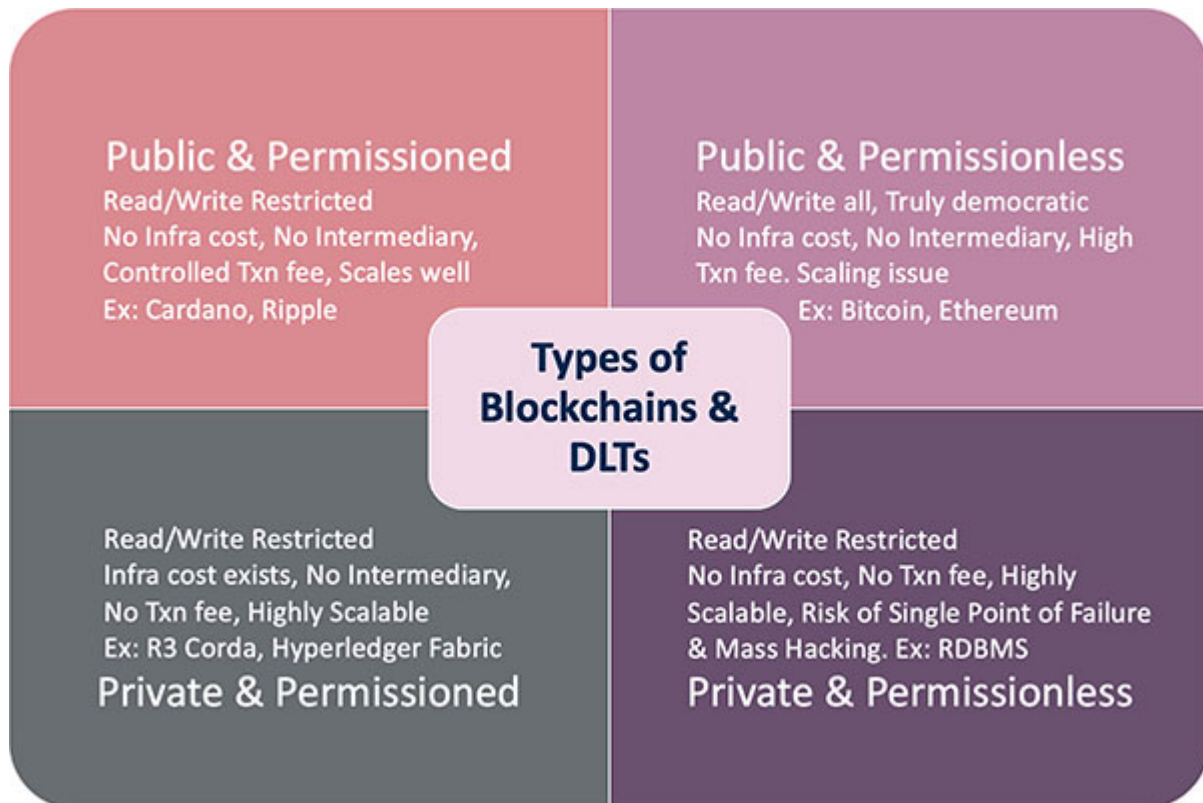


Figure 1.11: Types of Blockchain

Public Blockchains are often permissionless or open for all, i.e., one does not need the permission of any authority to join them. The original Bitcoin Blockchain network is such an example. Some public Blockchains are also public permissioned. But the private DLTs are mostly permissioned or built with only trusted parties or participants, and hence their consensus models are also of different types. Obviously, they are used for different types of use cases.

1.9 Does your Use Case need a Blockchain?

So far, the reader got some exposure to the Blockchain and the magic that it can create. So, if a Blockchain is that charismatic, then why are the big players not replacing their existing databases to Blockchain? The answer is simple. A Blockchain might be a brand new out-of-the-world framework, but it is not meant to address all kind of storage requirements. Let's find out where it works and where it doesn't.

1.9.1 When to use a RDBMS

As shown in [Figure 1.12](#), we opt for a traditional database if the following are true:

- We have all the data from a single party or organization.
- We can have a single access control for data validation.
- There is no issue of trust.

Refer to [Figure 1.12](#), as follows:

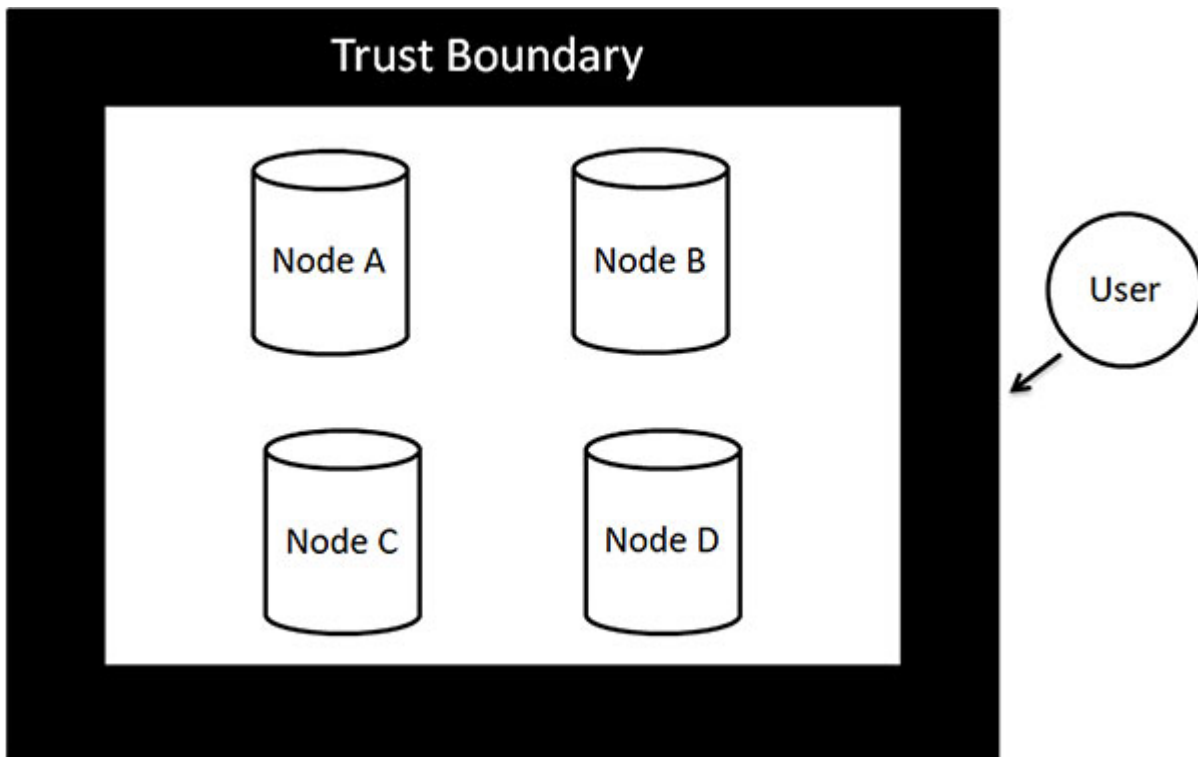


Figure 1.12: Use RDBMS for single organization with multiple subunits

1.9.2 Where to Use a Public Blockchain

When the data is not private, for example, some business scenarios need the data to be broadcast to all without boundary. Cryptocurrencies as Bitcoin and Ethereum I are the best examples. Social media, e-Auction systems etc., also fall into this category.

1.9.3 Where to Use a Private/Permissioned Blockchain

In the financial organizations and banks, there is a lot of exchange of data between multiple parties; however, they wish to share that data only with those who are concerned. Also, if you need permissions in a workflow, then the private and permission based DLT framework is right for you, as shown in [Figure 1.13](#) as follows:

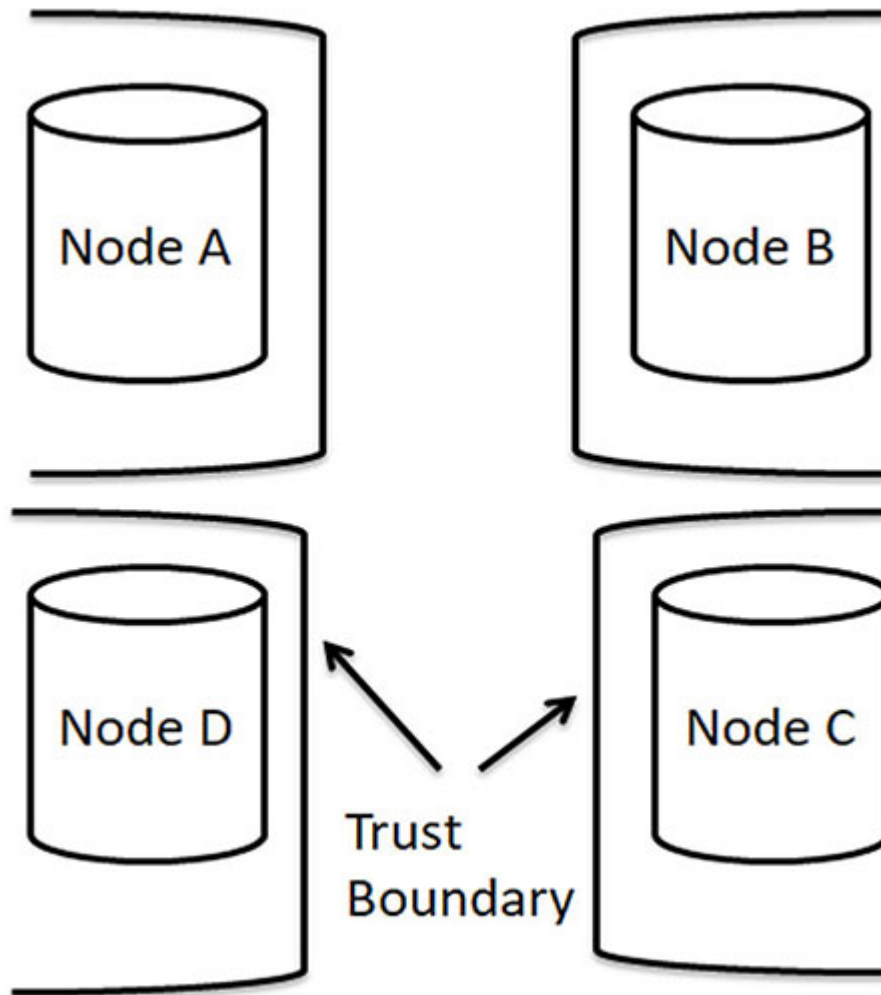


Figure 1.13: Use Private & Permissioned DLT for organizations with individual trust boundary

1.10 Cap Theorem or The Scalability Trilemma

As the business grows, our ecosystems start getting more complex with multiple databases. Also, there would be a need for

synchronization between them with adequate consistency and availability. The Cap theorem, also known as “The Scalability Trilemma”, is a belief in computer science about the distributed data stores as per which we can achieve only two among the three properties, i.e., Consistency, Availability, and Partition tolerance. Hence, different kinds of DBMS, RDBMS, Blockchains, and DLTs are built to match their respective non-functional requirements, sacrificing one property that is relatively less significant, as shown in [Figure 1.14](#) as follows:

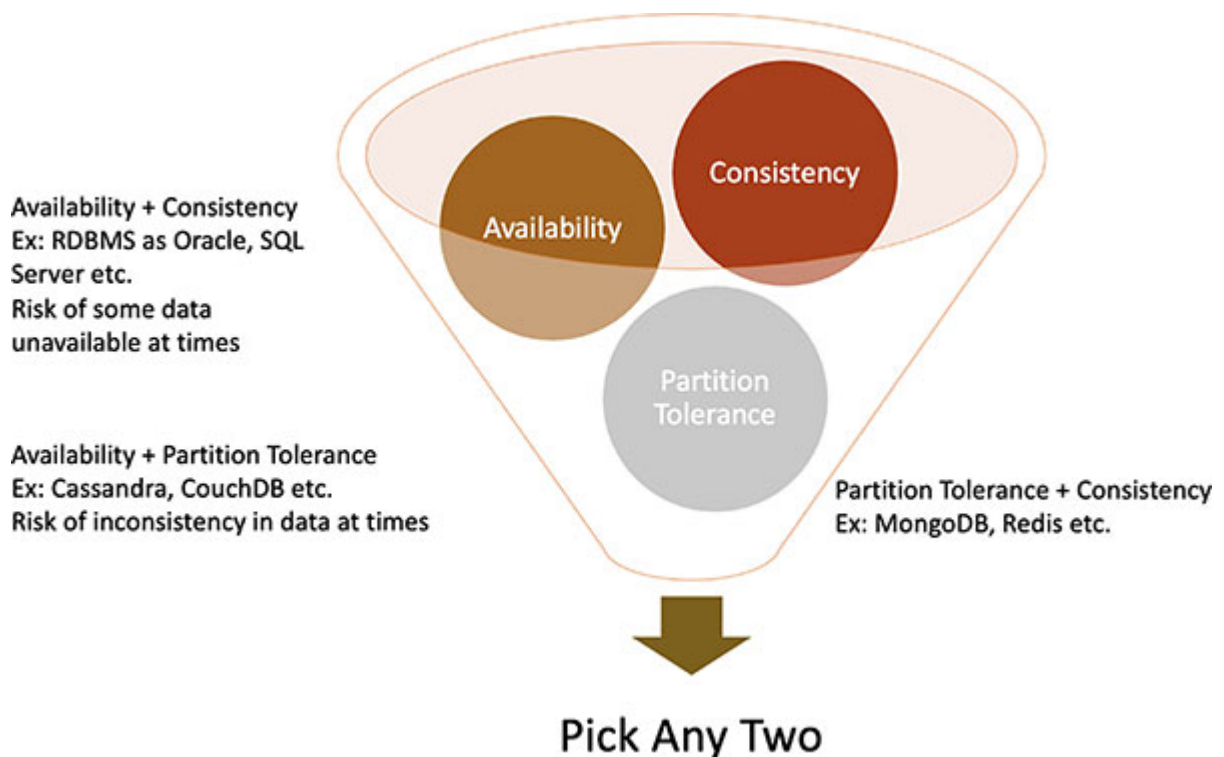


Figure 1.14: CAP Theorem

1.11 Bitcoin, The Environment Villain

Irrespective of Bitcoin’s huge popularity, this cryptocurrency is also heavily criticized for its mining policy that sucks a huge amount of energy.

As per a recent report by Business Insider, “Bitcoin mining consumes around 91 terawatt-hours of electricity annually. That’s more annual electricity use than all of Finland, which is a country of

5.5 million people. That's almost 0.5% of all electricity consumption worldwide, and a 10 times jump from just five years ago".

Such concerns has led the Tesla founder, Elon Musk to not accept any payment in terms of Bitcoin for any business transactions in the organization. Apart from many regulations and security concerns, Bitcoin has serious technical limitations such as slow execution time and high transaction fees. Transactions in the Bitcoin network take 10 minutes to get committed and the average transaction fee is almost \$23 today. Whosoever has worked in Bitcoin would be very well aware of its throughput which is about 24 transactions per second or TPS which is too slow to be considered for payments in the mainstream.

Obviously, such limitations would be a road-blocker for a Bitcoin or a similar Blockchain network to get into the mainstream, as they cannot compete with the legacy of the centralized systems such as VISA that runs almost at 24,000 TPS. Hence, it's only obvious that the next generation of Blockchains have to overcome such limitations by improving their capabilities with better architecture.

1.12 Areas of research and improvement

The following are some of the areas where most of the Blockchain protocols have worked to come up with a better speed for execution with a lower cost:

- Consensus models
- Scalability, Performance (i.e., faster finality and high transactions per second or TPS)
- Transaction fees
- L2 Solutions

1.12.1 Consensus Model

The primary reason for the Bitcoin's poor performance is because of its consensus model. The PoW consensus model works on a voting system that needs to wait to gather 51% consensus from the nodes

before the blocks are committed to the Blockchain network. The later Blockchain networks have tried to adopt many different and unique consensus processes that can make a decentralised system work faster, as represented in [Figure 1.15](#) as follows:

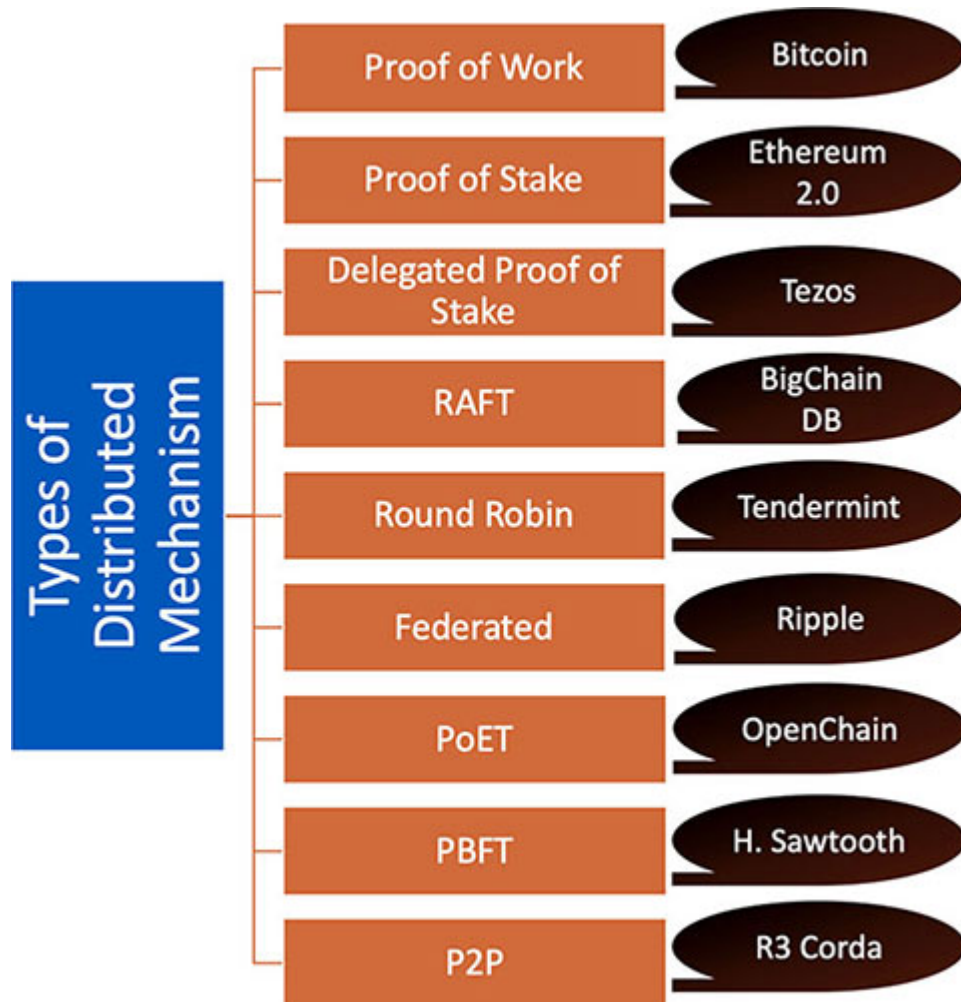


Figure 1.15: Types of Consensus Models in Blockchain and DLTs

We will discuss them in the later chapters along with the Blockchain protocols where they are implemented.

1.12.2 Scalability and Performance

We already know that cryptocurrencies were introduced in the market as an alternate payment mechanism. Currently, VISA is a global standard for payment processing and processes almost 1700 transactions per second; however, it claims to be able to handle up

to 24000 TPS in case of need. Hence, if cryptocurrencies are to be considered as a mainstream payment solution, they must be able to handle scalability and performance issues to be at par.

1.12.3 Transaction Fees

The fees for a transaction in the Bitcoin network keeps fluctuating as the price of the associated crypto changes. This is a major concern for a business as they would have no idea about how much it would be at the time of the business transactions. In many next-generation public Blockchains, this part has been stabilized by improving the consensus model.

1.12.4 Layer-2 Solutions

In the public Blockchain networks, often we have two layers, i.e., Layer-I (L1) and Layer-II (L2). While L1 is the consensus of the basic or the main Blockchain layer, L2 refers to any additional network that supports the L1 in order to scale. For example, Bitcoin is currently using the Lightning network that works on top of the Bitcoin network to enable the private transactions between parties. Similarly, many new L2 solutions are being used in the next-generation public Blockchain networks nowadays, in order to meet the requirements of scalability.

The second layer scaling solution can be implemented using many different options; the following is a list that represents the most popular ones:

- Sharding
- State Channels
- Parachains
- Sidechains
- Rollups

Let's explore them one by one.

1.12.4.1 Sharding

A shard means a small part of a whole. Sharding is heavily used in databases where a large database is partitioned into smaller units for faster and easier processing. Shards can also be located in different parts of the world, carrying area-specific customer data for security and faster access. Hence, sharding, in certain cases, is considered as a Layer 1 solution.

When it comes to Blockchain, sharding helps distribute the workload across the network reducing the workload on the nodes, leading to a much faster processing. On the down side, it might induce certain security concerns as it breaks the Blockchain into multiple shards and is prone to attacks.

1.12.4.2 State Channels

As represented in [Figure 1.16](#), a State channel is a mechanism where a group of users interact with each other outside the Blockchain network for a number of transactions through a channel. Only the final outcome is reported to the main ledger. Refer to [Figure 1.16](#), as follows:

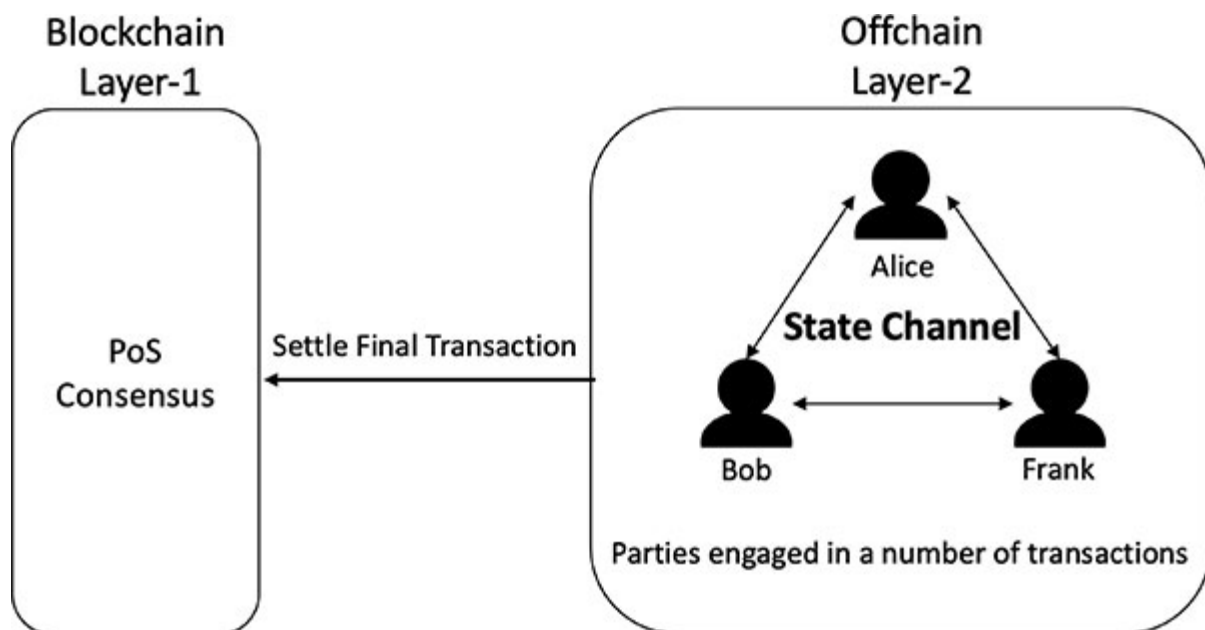


Figure 1.16: State Channel

Bitcoin's lightning network, Ethereum's Raiden network, Cardano's Hydra, and Neo Blockchain's Trinity work on such architecture.

1.12.4.3 Parachains

As shown in [Figure 1.17](#), Parachains or parallel chains work in parallel to the main Blockchain ecosystem. The beauty of this architecture is that this ecosystem can be extremely scalable as each Parachain can share the load and execute the transactions. Also, they can interact with the other Parachains as well as the main chain at the same time through messaging.

This model has been used by Polkadot interoperable Blockchain to be discussed in [Chapter 10](#).

Refer to [Figure 1.17](#), as follows:

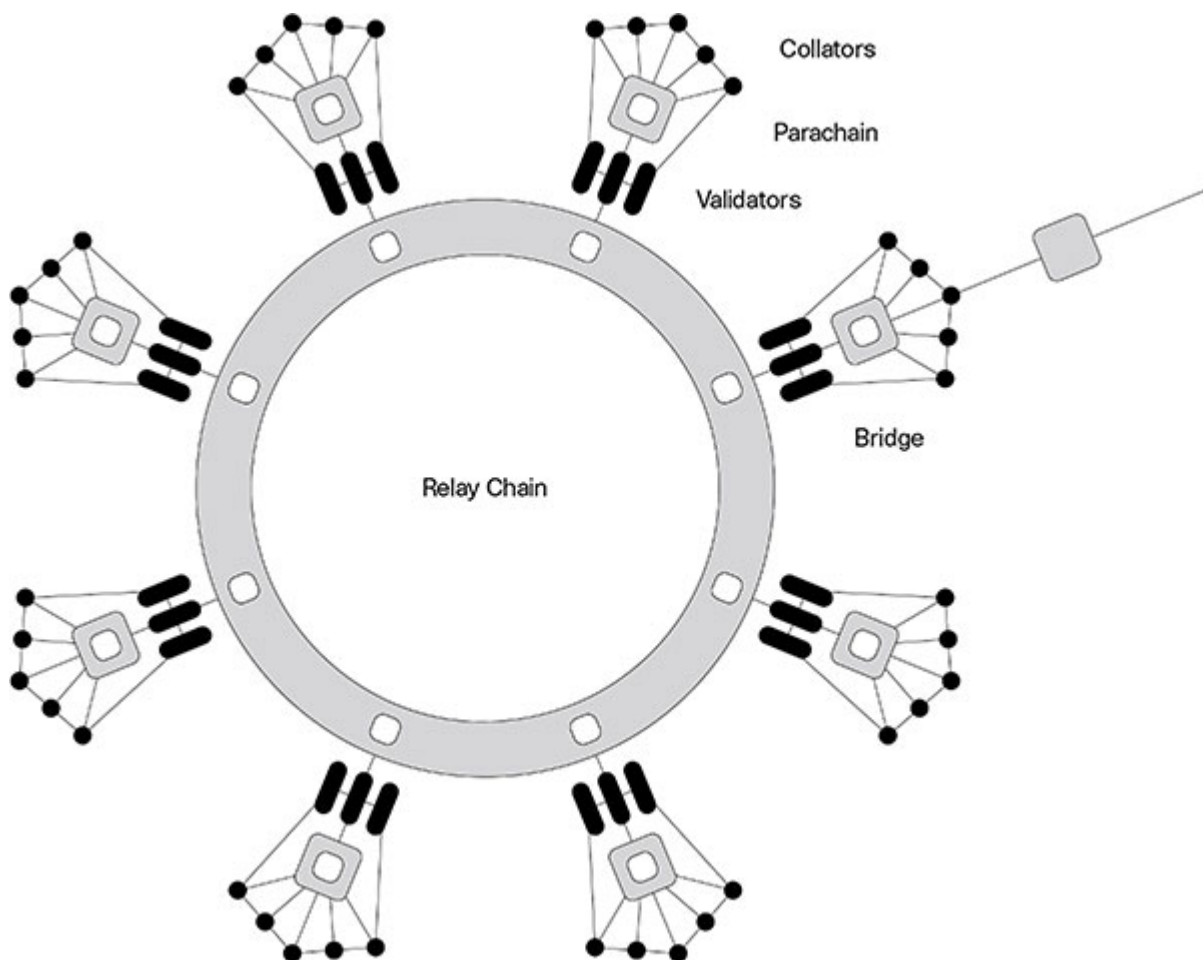


Figure 1.17: Parachains (Source: <https://wiki.polkadot.network/docs/getting-started>)

1.12.4.4 Sidechains

As demonstrated in [Figure 1.18](#), Sidechains are another type of architecture where different Blockchain networks work in parallel to the mainchain, as follows:

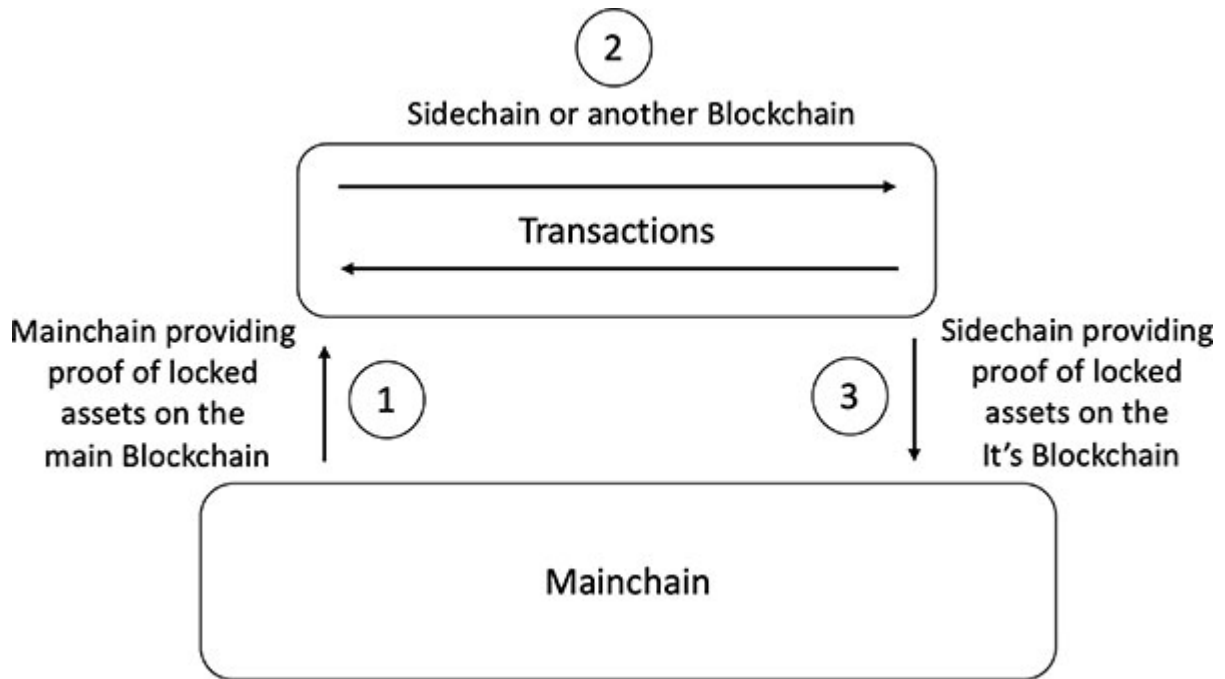


Figure 1.18: Sidechain Architecture

However, in this model, they are pretty much decoupled and work independently on their own. All the sidechains get connected to the mainchain through the two-way gates. They are pretty popular in the market and are widely adopted by many leading Blockchain solutions.

Some of the most popular examples of Sidechains are Microsoft's Sidetree solution for Decentralized Identity ION, Plasma Sidechain on Ethereum, Liquid Sidechain for Bitcoin, Cosmos etc.

1.12.4.5 Rollups

Rollups is another L2 architecture where the transactions get executed off-chain; however, the smart contracts as well as the proof of the transactions reside on the mainchain. There are two types of

Rollups popular in the market, which are Zero-Knowledge Rollups or ZK-Rollups and Optimistic Rollups.

In ZK-Rollups, the actual transaction happens offchain whereas only the proof of transaction is sent to the mainchain. For leveraging this solution, ZK-Rollups use a feature called **Zero-Knowledge-Proof**. Optimistic Rollups are similar to ZK-Rollups, however, as the name suggests, the L2 sends all the transactions to L1 without verification, i.e., with a practice of optimism. The dispute resolution happens on L1 later and the transactions are reverted in case of the detection of fraud. Of course, ZK-Rollups are faster than Optimistic Rollups as the validation occurs along with the updates of the transaction hand in hand.

Please note that the Rollups are far more scalable than the plasma L2 technologies. While in plasma, the proof is sent to the mainchain on per transaction basis, in case of the Rollups, many transactions are combined into one transaction offline and the collective proof is sent to the mainchain.

The Rollups model is fast, scalable, and would need far less as the transaction fees. Interoperable Blockchain **Polygon** or Matic uses this approach for scaling.

L2 technologies are still a new concept and needs far more research. They may sound too good to be true. However, we must note that with the L2 technologies, we are again moving towards some level of centralization.

Please also note that the scalability and performance of the Blockchain network is also associated with the consensus process and L2 solutions. As the decision making process would be shifted from PoW to the others and the transaction finality is achieved faster with the L2 solutions, we would see significant improvement in this space.

In my opinion, after the rise of private permissioned DLTs in the last 4-5 years, 2022 will be the breakout year for public

Blockchains. The cryptocurrency market cap is already top \$2 Trillion and they may soon have an impact on the economic, political, and social lives of all of us in one way or the other.

1.13 Reason for the Volatile Price of Crypto

"I am not a fan of Bitcoin and other Cryptocurrencies, which are not money, and whose value is highly volatile and based on thin air," Trump wrote on Twitter in 2019.

There are many cryptocurrency critics in the market including the former US president Donald Trump. One of the major reasons is the extreme volatile price fluctuation of the crypto. Just like any other stock in the stock exchanges, the price of the cryptocurrencies vary every day. Though the price fluctuation highly depends upon the demand and supply of crypto in the market, have you ever wondered why there would be a hike in demand? Let's evaluate the parameters.

1.13.1 Utility

Cryptocurrencies also work as gas or the fuel to run the network with different consensus mechanisms. Hence, more the adoption of a Blockchain network in the market, more would be the demand and higher would be the price.

1.13.2 Limited vs. Unlimited Supply

Cryptocurrencies also come with different types of supplies. While some have a maximum cap, others do not, and hence, there is some amount of inflation involved in the crypto world.

1.13.3 Hoarding

Sometimes, certain accounts buy and hold large amount of a particular crypto artificially raising its price.

Cryptocurrency is still a new world, and hence, it needs strict regulation rules just like the stock exchanges do. Unless the prices are under control, many organizations would not prefer to work with cryptos for the associated risks in the ever-changing transaction fees.

1.14 Market Capitalization

The market capitalisation of a cryptocurrency is a primary indicator for the investors to measure the safety and stability to trade in comparison to its peers.

Why does market capitalization matter?

Because, the higher the market cap, the higher is its growth potential in the global crypto market. Also, higher adoption indicates its ability to survive in the volatile environment of the cryptocurrency world.

As the technology matures, different protocols have been introduced in the market and some of them are playing really well with the solutions that can achieve most of the non-functional requirements listed earlier. So, in the upcoming chapters, let's explore the different types of Blockchains and DLT networks, and learn what problems they can solve.

Summary

In this chapter, we covered the following topics:

- Evolution and hype of Blockchain.
- Introduction to Bitcoin and its differentiators from the previous generation's payment systems.
- Key concepts of Blockchain as cryptography, hashing, wallet types, decentralisation, consensus, smart contracts, tokens, forks, threats, permission levels etc.
- Areas of research and improvement for Blockchain, for example, consensus models, scalability, transaction fees, and L2 technologies.
- How to choose the right crypto.

References

- Blockchain market size, share and global market forecast - <https://www.marketsandmarkets.com/Market-Reports/blockchain-technology-market-90100890.html>
- Blockchain technology market share report - <https://www.grandviewresearch.com/industry-analysis/blockchain-technology-market>.
- What are the Cryptocurrency Layer 2 Scaling Solutions? - <https://coinmarketcap.com/alexandria/article/what-are-cryptocurrency-layer-2-scaling-solutions>
- What is Layer 2 Scaling Solutions & Why It Is Required? - <https://medium.com/crypto-wisdom/what-is-layer-2-scaling-solutions-why-it-is-required-66b8dbf3bc9c>
- ZK-Rollups - <https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/zk-rollups/>
- Cryptocurrency prices, charts, and market capitalization - <https://coinmarketcap.com/>
- Blockchain Consensus - <https://devopedia.org/blockchain-consensus>

Quiz

Q1: What is/are the challenge(s) with Bitcoin?

- A. Not scalable
- B. Does not have any inherent value
- C. Not energy efficient
- D. All of the above

Q2: What are custodial and non-custodial wallets?

- A. In a custodial wallet, the user relies on a third party to store the private keys.
- B. A non-custodial type of wallet gives a user complete ownership of the private keys.

- C. Both of the above
- D. None of these

Q3: Which of the following attacks may lead to a 51% attack?

- A. Phishing attack
- B. DDoS attack
- C. Sybil attack
- D. All of these

Q4: Sybil attacks can be completely avoided in which of the following?

- A. Public permissionless Blockchain
- B. Public permissioned Blockchain
- C. Private permissioned DLT
- D. All of these

Q5: How does a gossip protocol transmit messages?

- A. It broadcasts the messages to all the nodes.
- B. It transmits messages on peer-to-peer basis.
- C. Both of these
- D. None of the above

Q6: While buying crypto, what all parameters should you keep in mind?

- A. Utility
- B. Total number of crypto in supply
- C. Market capitalization
- D. All of the above

Q7: You are in a process of selecting an L2 technology that is the fastest. Which among the following would you select?

- A. Plasma network
- B. ZK Rollups

- C. Optimistic Rollups
- D. All can scale equally well

Q8: Parallel chain L2 technology has been implemented in which of the following Blockchains?

- A. Solana
- B. Cardano
- C. Polkadot
- D. None of the above

Q9: ZK Rollups and Optimistic Rollups L2 technologies have been implemented in which of the following interoperable Blockchains?

- A. Polkadot
- B. Cosmos
- C. Polygon
- D. None of the above

Q10: Which of the following is a consensus model?

- A. Proof of Stake
- B. Round Robin
- C. P2P
- D. All of the above

Q11: Which of the following cryptocurrencies work on the PoW consensus model?

- A. Bitcoin
- B. Monero
- C. Dogecoin
- D. All of the above

Q12: Which among the following is true for asymmetric cryptography?

- A. RSA is faster than DSA in signature validation

- B. DSA is faster than RSA in signature generation
- C. Ed25519 is safer and faster than DSA, ECDSA, & EdDSA
- D. All of the above

Q13: Which among the following is an implementation of the “Direct Acrylic Graph” consensus?

- A. Ethereum
- B. Hedera Hashgraph
- C. Solana
- D. All of the above

Q14: Which among the following is a backward compatible temporary fork?

- A. Hard Fork
- B. Soft Fork
- C. Both of the above
- D. None of these

Q15: Tendermint consensus is applied on which of the following Blockchains?

- A. Cosmos
- B. Hedera Hashgraph
- C. Ethereum
- D. Solana

Q16: Bitcoin’s lightning network, Ethereum’s Raiden network, and Neo Blockchain’s Trinity fall under which type of L2 technology?

- A. Sharding
- B. State Channels
- C. Sidechains
- D. Rollups

Q17: Microsoft’s Sidetree solution for Decentralized Identity ION, Plasma Chains, Cosmos etc., fall under which type of L2

technology?

- A. Sharding
- B. State Channels
- C. Sidechains
- D. Rollups

Q18: Cardano's Hydra L2 solution falls under which type of L2 technology?

- A. Sharding
- B. State Channels
- C. Sidechains
- D. Rollups

Q19: If you are the CTO of an IT organisation, which of the following use cases would you focus for an immediate production?

- A. Authenticated Provenance
- B. Decentralized web
- C. Non-Fungible tokens
- D. Blockchain managed services

Q20: L2 technologies sound too good to be true. What can be their drawbacks?

- A. They are still new and need more research.
- B. They need additional configuration, and hence, are more complex to handle.
- C. Both of the above
- D. None of these

Answers

1. D 2. C 3. C 4. C 5. B 6. D 7. B 8. C 9. C 10. D 11. D 12. D 13. B
14. C 15. A 16. B 17. C 18. B 19. C 20. C

PHASE I – PUBLIC BLOCKCHAINS

Even a decade after the birth of Bitcoin, it's still the talk of the town, and hence, the other cryptocurrencies are mostly backed by public Blockchains. The biggest benefit of public Blockchains is that they are public in nature, and hence, do not need any further infrastructure to invest in for production. In the following chapters, let's explore some of the most popular public Blockchains and discuss how they have managed to overcome the shortcomings of the Bitcoin Blockchain.

CHAPTER 2

Ethereum

“In order to have a decentralised database, you need to have security. In order to have security, you need to have incentives”

— Vitalik Buterin

Founded: 2015

Native token: ETH

Market Capitalization: \$355.8B

In this chapter, we will cover Ethereum, the first programmable Blockchain platform that inspired the whole concept of Altcoins and many more Blockchains and DLTs of today.

Introduced in a whitepaper in 2009, Bitcoin started trading in 2010 and immediately attracted global attention. It's worthy to note that, back in 2009, when Bitcoin was introduced, the word Blockchain itself was not coined. It took some time for the industry to comprehend the power of the underlying technology in order to transform it to a programmable platform. In 2015, a young programmer, Vitalik Buterin launched Ethereum as the first programmable permissionless Blockchain that runs on a virtual machine called “Ethereum Virtual Machine” or EVM. The whole EVM network is considered as a single, canonical computer whose state is accepted by all the nodes on the Ethereum network.

2.1 Consensus Model

When launched in 2015, Ethereum too had an underlying proof-of-work consensus model, just like Bitcoin; however, it was a bit faster than its predecessor. Ethereum blocks were validated and committed to the Blockchain network in nearly every 12 seconds, whereas a Bitcoin approximately took 10 minutes. Also, unlike Bitcoin that has a

fixed supply of 21,000,000 coins, Ethereum had no upper cap in supply.

In year 2021, Ethereum started its process of moving from Proof of Work to Proof of Stake with the help of sharding. This transition would occur in many stages and ultimately would lead to a very high scalability and low transaction fees. Altair, Ethereum 2.0's first hard fork, is expected to come into effect by the end of 2021, reducing Ethereum's power consumption by 99.9%

2.2 Transaction fees

The Ethereum network runs on its native cryptocurrency called Ether that is available in most of the crypto exchanges in the world. The current average transaction fee on the Ethereum network is close to 0.0008 Ethers per transaction (i.e., almost \$3 USD) which is pretty high. Most traders and investors communicate in terms of Ethers, whereas when it comes to the Ethereum network, the consumption happens in terms of gas that determines how expensive a transaction is from its computation requirements.

2.3 L2 Solutions

Since its launch in mid 2015, the number of people participating in the Ethereum network has grown exponentially. This crazy rush led to an extremely slow network and high price for the transaction fees on the Ethereum network. Hence, in order to match the scaling needs and lowering transaction fees on Ethereum, different L2 solutions such as sharding, side chain, plasma, state channels etc., are being tried on Ethereum's L1 network.

2.4 Scalability and Performance

We already know that the current scalability of Ethereum is limited to almost 12 TPS.

2.5 Development

If you wish to write smart contracts on the Ethereum platform, then you can do so by using its primary language called Solidity. Solidity was initially proposed in August 2014 by Gavin Wood who is a British computer programmer and co-founder of Ethereum. Later on, Solidity was further developed by the Ethereum project's Solidity team, led by Christian Reitwiessner. Today, Solidity is not only used as the primary language of Ethereum, but also for many other Ethereum Virtual Machine or EVM compatible Blockchains and DLT platforms such as Quorum, Monax, Hyperledger Burrow, and Hedera Hashgraph.

2.5.1 Why Learn Solidity?

Now, one may wonder why there was even a need to create a new language when we already had so many present in the market by 2014? Yes, by the time Solidity was invented, we already had quite a few JVM languages such as **Java**, **Scala**, **Ruby** etc., with the support of rich object-oriented features. Then, why Solidity?

Well, the answer is that Solidity is a special purpose language meant to be run on the Ethereum Virtual Machine or EVM. An EVM is architected in a very different way where most of the transactions are associated with some **monetary value**. This cost calculation was not part of any of the JVMs. Hence, Solidity filled the gap as an object-oriented language that can run contracts on EVM.

2.5.2 Solidity vs. Who?

Please note that Solidity is not the only language in Ethereum for writing Smart contracts. There are others such as **LLL** which is the original low-level language for Ethereum and is deprecated now. We also have **Serpent** which is also deprecated today. We can also write smart contracts on Ethereum using **Vyper** which was released in 2017 and currently is the latest contract language based on Python.

Please note that we can also write smart contracts in **assembly language** or low level language or opcodes which can give us more fine-grained access to the functionalities which are more suitable for

writing libraries. They also come with enhanced features which are not possible in high level languages. The gas cost or cost of operation is low in the assembly and low-level languages. However, they too have their drawbacks. Many important safety checks are bypassed in the assembly languages. The maintenance of the low level languages would always be an issue.

Hence, Solidity is the **most widely used language** for writing Smart Contracts on Ethereum. You would find a lot of examples and existing codes running in production on Solidity. Also, there are frameworks on Solidity such as Truffle, Embark, and OpenZeppelin to save the developer's time. Hence, Solidity seems to be the safest bet so far.

2.5.3 Solidity Features

What are the features that made Solidity to be widely used in the Blockchain initiatives? Let's look at the following features:

- Solidity is a **statically typed** programming language. In Solidity, the variables have to be declared before they can be assigned values which is very similar to what we observe in Java, C, C++, FORTRAN, Pascal, and Scala.
- Solidity is partly designed after **ECMAScript**, and hence, its syntaxes are pretty much similar to JavaScript. Hence, if a programmer had already coded in **JavaScript**, Solidity would be familiar.
- However, unlike JavaScript, Solidity is an **object-oriented** language and supports all the related features such as encapsulation, inheritance, polymorphism, abstraction, function overloading, etc.
- Once compiled, Solidity is translated to bytecodes and can run on the Ethereum Virtual Machine.

2.5.4 REMIX browser

The best integrated development environment (IDE) to write a Solidity program for beginners is REMIX which is an online browser

that is openly available on <https://remix.ethereum.org/>. Just make sure that you have a connection to the Internet with a high bandwidth and any standard browser available. I have tested Solidity samples on Chrome.

Now, we have to click on the Solidity button, visit the “FILE EXPLORERS” section, and then click on the “New File link” link under the Contracts folder to create a new Solidity file. The Solidity file comes with an extension of the .sol file. We can assign a name to our file and see it being added to the File explorer section on the left hand side.

2.5.5 Write a Small Program

Now, let me write a small program, compile it using the REMIX browser, deploy it, and run it with the simplest possible approach. Do not worry about the programming details which I will cover later on with details. Let’s just concentrate on the environment.

Now, I have written a very simple code, which is as follows:

```
//SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;
contract FirstContract {
    constructor() {
    }

    function retrieveOne() public pure returns(uint){
        return 1;
    }
}
```

Please note that all Solidity files have a .sol extension.

2.5.5.1 Compiling

Compiling the Solidity program is pretty straight forward on REMIX. Click on the “SOLIDITY COMPILER” section. Please make sure that the right compiler is selected on the left hand side, which is a default activity.

Under the “COMPILER CONFIGURATION” sub-section, there is an option for “Auto Compile”; if you set it on, it’s really helpful while writing complex algorithms. If something goes wrong in the code, it can indicate you immediately. You can also prefer to set it off and compile it directly by clicking on the button. If there is an error, it shows the line number and marks it in red. All the errors must be fixed before moving on to the next stage.

Now, here you may find a couple of warnings marked in yellow if you delete the first line, but that is fine. Unless we have errors, we can ignore these warnings for the time being and go ahead with deployment and running.

2.5.5.2 Deployment

Deploying a Solidity contract on REMIX is simple. We can just visit the “DEPLOY AND RUNTRANSACTION” section and choose the compiled contract to run.

Now, we have three different environments for deploying and testing the contracts, which are as follows:

- JavaScript VM (default)
- Injected Web3
- and Web3 Provider

Let’s keep the default one, i.e., JavaScript VM. Why?

It will run an isolated Ethereum node in the browser. It is very useful when you want to test a contract. There are already a few Ethereum accounts that are created and prepopulated with 100 Ethers each. Now, we deploy.

You can find a Deployed Contracts section just below the Deploy button. You can keep clicking on the deploy button again and again; each time you will find a new version of the deployed contract. You can also expand this section and delete the previous versions of the deployed contracts to avoid confusion. Then, deploy again.

Now, you can expand the deployed contract and find a button in the same name as the function that we wrote. You can click on the button and find the desired output now.

The Solidity codes can further undergo unit testing and integrated testing from the test networks, and finally be deployed on the Ethereum mainnet or on production. We can visit <https://etherscan.io/> to find many transactions that are being executed on the Ethereum network every now and then.

2.5.6 Application Binary Interface

Every contract comes with a binary interface for integration with the front-end code as well as the other contracts that can interact with it. The binary interface is way similar to API or Application Programming Interface that the high level languages use to expose their services to the external world. ABI has the information of all the function names, input and output parameters, and Event names and their parameters. On the REMIX browser, once we compile the code, we can find a “Compilation Details” button, clicking on which, we can view the details of the ABI.

2.5.7 Layout of Solidity source code

Now, let's find out the internal details of a Solidity file. First of all, a solidity file has a .sol extension.

2.5.7.1 SPDX License Number

On the first line in the Solidity file, we can have an optional comment within which we have to specify the license associated with the code. This is useful if you wish to add any copyright related information, as follows:

```
// SPDX-License-Identifier: Some Identifier
```

If you do not wish to write any SPDX identifier, just write the following:

```
// SPDX-License-Identifier: Some Identifier
```

However, not writing it might throw a warning message.

2.5.7.2 Version

Next to the information on licensing, the code has a mandatory “pragma solidity version number”, where we set the version for the compiler.

Please note that the latest version, as of December 2021, is “Version 0.8.10”. However, you will find many different Solidity sample codes on the Internet with different version numbers. Please note that the new versions of Solidity keep getting released from time to time, which come with many new features as well as bug fixes. While it’s advisable to work with the latest versions, it’s also true that some of the previous versions might not be compatible with the future compiler changes.

Hence, if we write “pragma solidity ^0.5.11;”, then the code would not compile with the earlier versions of the compiler. At the same time, it would also not compile with any version starting from 0.6.0 and onwards. The “^” symbol ensures that the code is compiled with the right compiler.

We can also write something like “pragma solidity >0.4.23 <0.5.0;” which means that the code can be compiled by any version of Solidity that falls under these two versions.

We can also write “pragma solidity pragma solidity >=0.4.0;”.

2.5.7.3 Import File

We can have an optional import statement where we can import the other Solidity file within the current code, as follows:

```
import "filename";  
  
We can also write  
  
import "filename" as symbolName;
```

While importing the other Solidity files, it’s very important to specify the path to those files correctly. If the other file is in the same location, then we can write the name directly. Or else, if it’s in some other folder, then we can write something like the following:

```
"import /folder1/folder2/filename.sol"
```

2.5.7.4 Contract

We also have one or more contracts in the body.

Constructor: In each contract, there can be an optional constructor. Also, each contract can have different types of variables, events, and functions within them.

Comments: We can add single line comments, like the following:

```
// This is a single-line comment
```

We can also add multiline comments, as the following:

```
/*  
  
This is a multiline comment  
I can describe the entire business logic of the function here  
  
*/
```

We can also write a double asterisk block before every contract and every function to add additional data on the input parameters, and return, as shown as follows:

```
/**  
 * @title Calculator  
 * @dev Implements addition of two numbers  
 */
```

2.5.7.5 Libraries

A library is a utility or reusable piece of code that can be called from a contract many times as per requirement. A library can have variables or functions just like a Contract and can be called from a Contract as a utility.

2.5.7.6 Interfaces

An interface in Solidity is an abstract type that specifies the behaviour of the contract. The interface declares the function signatures which are external in nature, that an inheriting contract has to implement. It can also not declare the constructor or state variables.

2.5.7.7 Events

Events are an abstraction on top of the Ethereum Virtual Machine's logging functionalities. They are especially useful for integration with the front end code. The front end code can work as an observer with the call back functions waiting for a particular event to have occurred. In the decentralized applications, events play a major role in knowing the status of a transaction.

2.5.7.8 Order of Elements in Solidity File

While writing the Solidity code, we must make sure that all its elements are in a particular order. When you're writing the Solidity code, you should first make sure to use a proper layout. The contracts should be at the end of it. The following is the proper order of elements:

1. The pragma statement – which handles the associated compiler version.
2. Import statements – that imports the other files to the current.
3. Interfaces – which are the abstract types for a contract.
4. Libraries – mostly utilities or common codes.
5. Contracts – every Solidity file can have one or more contracts within them.

2.5.7.9 Order of Elements in a Contract

Libraries, interfaces, and contracts have their own elements as well.

The inner elements of each contract should have the following order:

1. Type declarations
2. State variables

- 3. Events
- 4. Functions

2.5.8 Functions in Solidity

Functions are reusable piece of code where we often write the business logic and can call it again and again anywhere in the program.

A function can have four visibility types, which are as follows:

- External
- Public
- Internal
- Private

The visibility must be assigned to a function, otherwise it would throw an error.

External: A contract can be called from another contract, provided that the external contract already knows the calling contract by using the external keyword as a part of the contract name.

Public: The keywords are used mostly in getters and setters as well as functions that you wish to invoke on the contract directly. They can also be used against the state variables.

Internal: The visibility is only from within the current contract or the contracts derived from it.

Private: The visibility is only for the contract that they are defined in and not in the derived contracts.

Hence, all the visibility in the order of decreasing permissiveness is as follows:

Public > External > Internal > Private

2.5.8.1 Constructor

Constructor is a function with the same name as a contract. Every contract can optionally have a maximum of one constructor in it. However, unlike Java, the constructor can't be overloaded. The

constructor is invoked only once when the contract is created. Hence, we can keep any kind of initialization code within it, as shown as follows:

```
// SPDX-License-Identifier: Some Identifier
pragma solidity ^0.8.10;

contract ConstructorSample {
    uint a;
    constructor() {
        a = 7;
    }
    function someFunction() public view returns(uint){
        return a;
    }
}
```

In this example, you can find that the value of the variable is initiated in the constructor.

If there is no constructor, the contract will assume the default constructor, which is equivalent to a constructor with an empty body `constructor() {}`.

Prior to this version, we had to specify the visibility of the constructors as either internal or public, but now it's no longer required.

2.5.8.2 Setter() and Getter()

These are special types of functions where we can allocate and retrieve the values in the variables.

In the Setter() functions, which we actually named as `setName()`, `setAge(uint age)` etc., we set or allocate the values to the State variables. In the Getters, which are usually named after what we are retrieving such as `getName()`, `getAge()`, we retrieve the values of the state variable. These functions are usually not used for the implementation of any other business logic.

Refer to the following code:

```
// SPDX-License-Identifier: Some Identifier
pragma solidity ^0.8.10;

// A simple contract with getter and setter functions
contract GetterSetterContract {
    string value = "Some Value";

    function getValue() public view returns(string memory) {
        return value;
    }

    function setValue(string memory newValue) public {
        value = newValue;
    }
}
```

In a simple Solidity program, you can find how the value is allocated with the `setValue` function and then retrieved later using a `getValue` function.

Please do not worry about the keyword “memory”. We will cover that later in this chapter.

2.5.8.3 Return Values

A function can have one or more optional return statements as the last line. We already know how the getter function returns a value, whereas a setter does not. Now, let's improve the program further to retrieve multiple values in the same function. Refer to the following code:

```
// SPDX-License-Identifier: Some Identifier
pragma solidity ^0.8.10;

// A simple contract with getter and setter functions and
multiple values in return statement
contract GetterSetterContract {
    string value1 = "Some Value 1";
    string value2 = "Some Value 2";
```

```
function getValues() public view returns(string memory,  
string memory) {  
    return (value1, value2);  
}  
  
function setValue(string memory newValue1, string memory  
newValue2) public {  
    value1 = newValue1;  
    value2 = newValue2;  
}  
}
```

When a function has multiple return types, the statement `return (v0, v1, ..., vn)` can be used to return multiple values. The number of components must be the same as the number of return variables and their types have to match.

2.5.8.4 Order of Functions

In a Solidity file, the functions should be ordered as per their visibility, as follows:

- constructor
- fallback function (if exists)
- external
- public
- internal
- private

2.5.9 Variables

In terms of accessibility and storage locations, the variables in Solidity can be divided into three types, which are as follows:

1. State Variables
2. Local Variables
3. Global Variables

2.5.9.1 State variable

The State variables store the value permanently in the contract storage. They can be accessed from anywhere in the Contract. Refer to the following code:

```
// SPDX-License-Identifier: Some Identifier
pragma solidity ^0.8.10;

contract GetterSetterContract {
    string value = "Some Value";
    function getValue() public view returns(string memory) {
        return value;
    }

    function setValue(string memory newValue) public {
        value = newValue;
    }
}
```

2.5.9.2 Local Variable

A local variable can be accessed only within the context of a function, and it cannot be accessed outside. Usually, these variables are used to hold temporary values. Their main purpose is to help with processing or calculating something and returning the final value in the function's response. Refer to the following code:

```
// SPDX-License-Identifier: Some Identifier
pragma solidity ^0.8.10;

contract SumContract {
    function sum() public pure returns(uint) {
        uint var1 = 10;
        uint var2 = 20;
        return var1 + var2;
    }
}
```

“var1” and “var2” in the upper screen is a local variable, which we cannot use outside of the “sum” function.

Furthermore, there is a difference in the storage location between the state and the local variables, that we’ll cover later in this chapter.

2.5.9.3 Global variables

There are quite a few globally available predefined variables in Solidity that we often use in our code. Some of them are wei, gwei, blockhash, msg.data, msg.sender, now etc.

Refer to the following code:

```
// SPDX-License-Identifier: Some Identifier
pragma solidity ^0.8.10;

contract SenderDetails {
    address owner;
    constructor() {
        owner = msg.sender;
    }

    function getOwner() external view returns (address) {
        return owner;
    }
}
```

If you compile, deploy, and run the preceding program, then you will find a value like “0xd031A3AAC160f17a87D8D445c4307eFd250c3aed” which is the address of the Ethereum account holder who is running the contract. Hence, “msg.sender” is a global variable.

While coding, the programmer must not use these reserved names for the global variables as a local or state level variable.

2.5.10 Types

Earlier, it was already mentioned that Solidity is a statically typed language, which means that the variable types are to be declared

before using them in the program. Some of the variable types are Integers, fixed number, Boolean, byte, enum, address, array, struct, and string.

However, all these data types can be again divided into two types, which are as follows:

- Value type
- Reference type

Please note that even if Solidity is partly designed after ECMAScript, its syntaxes are pretty much similar to JavaScript. However, the concept of the “undefined” or “null” values does not exist in Solidity, but the declared variables, even if not explicitly initialized, always have a default value dependent on its type.

2.5.10.1 Value Type

The value type variables are those that are passed by value after being copied independently for wherever they are used and not by their reference in some real location where they are stored. Now, let's explore the types that fall under the value type.

2.5.10.1.1 Integers

Integers in Solidity can be either of signed (i.e., int) or unsigned (i.e., uint) type. I hope we all know the difference between signed and unsigned. The unsigned integers are 0 and positive numbers, whereas the signed integer can be negative, 0, and positive integers.

By default, the size of the integer in Solidity is 256 bits, which can also be represented as int256 for the signed and uint256 bits for the unsigned integer. We can have other sizes as int8/uint8, int16/uint16, int32/uint32 up to int256/uint256. Please refer to the maximum and minimum allowed values in each of these integer types while using them in the program.

2.5.10.1.2 Fixed Numbers

A number can be of a constant or a fixed value by using a fixed keyword in its front.

2.5.10.1.3 Boolean

In Solidity, `bool` stands for a Boolean value that is either true or false. Booleans are pretty useful in the functions where we write the business logic and compare the different values with the help of logical operations like “or”, “and”, “equal to”, “not equal to”, “greater than”, “smaller than” etc., to get a Boolean value.

2.5.10.1.4 Byte

Byte is a fixed size array of value type. It comes with a special functionality called “`.length`” that returns the length of the byte array. Please note that a byte can be a dynamically sized byte or string as well; however, that would fall under the value by reference as the size might grow bigger than the maximum size for the pass by value type of variables.

2.5.10.1.5 String Literals

A string can be represented by any group of characters within the double or single apostrophes, i.e., “” or “. Unlike byte, the string does not come with special functionality called “`.length`”. Hence, depending upon the requirement, a byte or string has to be chosen by the programmer as a byte can also work as an array of characters which is ultimately similar to a string. If required, bytes can be converted to a string as well. Refer to the following code:

```
// SPDX-License-Identifier: Some Identifier
pragma solidity ^0.8.10;

contract BytesToStringContract {
    function bytesToString() public pure returns(string memory)
    {
        bytes memory byteString = "Sample data";
        string memory message = string(byteString);
        return message;
    }
}
```

The preceding code is an example of the process of converting a byte into a string.

2.5.10.1.6 Enum

Enum represents a fixed number of predefined constant values, and has one as the default. Enums are greatly helpful when we need to have a set of fixed and related data, for example, the days of a week or months in a year or holidays in a year etc. The following is an example, i.e., FirstEnumContract.sol:

```
// SPDX-License-Identifier: Some Identifier
pragma solidity ^0.8.10;

contract FirstEnumContract {
    enum SomeData {DEFAULT,ONE,TWO}
    SomeData someData;

    function FirstEnum() public{
        someData = SomeData.DEFAULT;
    }

    function setValues(uint value) public {
        require(uint(SomeData.TWO) >= value);
        someData = SomeData(value);
    }

    function getValue() public view returns (uint){
        return uint(someData);
    }
}
```

2.5.10.1.7 Address

Address is a type that saves an account address on the Ethereum node. Addresses can be very useful if we have to transfer the Ethers as payments from one Ethereum account to another. Using the address's balance and transfer functions, we can copy the amount from one to another address on the Ethereum nodes. Addresses can be of the following two types:

1. address: A static address with just the functionality of holding a 20-byte value (size of an Ethereum address).
2. address payable: It's an address with additional members as transfer and send with the support for the functionalities of transferring and sending the Ethers.

2.5.10.2 Reference Type

Complex data types as arrays, structs, etc., that are far bigger than 256 bits in size needs to be handled with more care than the types passed by value. Such types, known as reference types, are also saved with special keywords such as storage (persistent store) or memory (non-persisting store).

Now, let's explore the types that fall under the reference type.

2.5.10.2.1 Array

An Array is a data type which carries a group of data of the same value, such as type byte, string, int, uint etc. An array can be created with a fixed length that we assign at the time of declaration or the variable length type. Arrays can also be dynamic where no size is allocated and it can grow gradually with the addition of more data. Refer to the following example code:

```
uint[5] marks = [10, 20, 30, 40, 50];
```

If it's a state variable, then we have to declare with a storage keyword, so that it can be saved in a permanent location, as shown as follows:

```
// SPDX-License-Identifier: Some Identifier
pragma solidity ^0.8.10;

contract Types {
    uint[5] score;

    function getScore() public returns (uint[5]memory){
        score = [10, 20, 30, 40, 50];
        return score;
    }
}
```

```
}
```

However, if we need to create a local variable of type array, we must store it using the memory keyword to save it in the temporary storage, as shown as follows:

```
// SPDX-License-Identifier: Some Identifier
pragma solidity ^0.8.10;

contract Types {
    function getScores() public pure returns (uint8[5] memory){
        uint8[5] memory scores = [10, 20, 30, 40, 50];
        return scores;
    }
}
```

2.5.10.2.2 Struct

I find Struct to be the most interesting complex data type that can further be comprised of many other simple data types, arrays, and mappings.

2.5.10.2.3 Mappings

Mappings are always the state variables and are similar to the hash tables in java, where the key-value pair data is stored.

Please note that Integers, fixed number, Boolean, byte, enum, and address can be copied by value, whereas Array, Struct, and String are passed by reference. Please note that passing by reference means that you are pointing to a memory location of the variable.

There are mainly two types of variables available in Solidity – State variables and Local variables. We consider them as the Global variables and Local variables, just as any other language.

2.5.10.3 Constant and Immutable State Variables

A State variable can be declared as either “constant” or “immutable” if it needs to have a fixed value. However, in case of the constant, the value has to be set in the compilation time, whereas for the

immutable variables, the value can be set at runtime. Refer to the following code:

```
// SPDX-License-Identifier: Some Identifier
pragma solidity ^0.8.10;

contract FixedValue {
    string constant name = "some name";
    uint immutable maxValue;

    constructor(uint value) {
        maxValue = value;
    }
}
```

2.5.11 Operators in Solidity

In most of the languages, we often have to use different operators to do the calculations, processing, and implementation of different business logics as per the requirement. Solidity supports the following types of operators:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Conditional Operators

Let's have a look at all the operators one by one.

2.5.11.1 Arithmetic Operator

The Arithmetic operators are perhaps the most widely used operators in any programming language and are the basic requirement of any mathematical calculation.

Solidity supports the following arithmetic operators, as shown in [Table 2.1](#), as follows:

OPERATOR	DENOTATION	DESCRIPTION
----------	------------	-------------

Addition	+	Used to add two operands
Subtraction	−	Used to subtract the second operand from the first
Multiplication	*	Used to multiply both operands
Division	/	Used to divide the numerator by the denominator
Modulus	%	Gives the remainder after the integer division
Increment	++	Increases the integer value by one
Decrement	—	Decreases the integer value by one

Table 2.1: Arithmetic Operators

Refer to the following code:

```
// SPDX-License-Identifier: Some Identifier
pragma solidity ^0.8.10;

contract ArithmeticContract {
    uint a = 10;
    uint b = 2;

    function addition() public view returns(uint) {
        return a + b;
    }

    function subtraction() public view returns(uint) {
        return a - b;
    }

    function multiplication() public view returns(uint) {
        return a * b;
    }

    function division() public view returns(uint) {
        return a / b;
    }

    function modulus() public view returns(uint) {
```

```

    return a % b;
}

function increment() public returns(uint) {
    return ++a;
}

function decrement() public returns(uint) {
    return --a;
}

function retrieveValues() public view returns(uint, uint) {
    return (a, b);
}
}

```

Please refer to the preceding simple Solidity contract to know how they can be used in the code.

2.5.11.2 Relational Operators

These operators are used to compare the relationship between two values. [Table 2.2](#) shows the list supported by Solidity, as follows:

OPERATOR	DENOTATION	DESCRIPTION
Equal	==	Checks if two values are equal or not, returns true if equal and vice-versa
Not Equal	!=	Checks if two values are equal or not, returns true if not equal and vice-versa
Greater than	>	Checks if the left value is greater than the right or not, returns true if greater and vice-versa
Less than	<	Checks if the left value is less than the right or not, returns true if less and vice-versa
Greater than or Equal to	>=	Checks if the left value is greater or equal than the right or not, returns true if greater or equal and vice-versa
Less than or Equal to	<=	Checks if the left value is less or equal than the right or not, returns true if less or equal and

Table 2.2: Relational Operators

Refer to the following code:

```
// SPDX-License-Identifier: Some Identifier
pragma solidity ^0.8.10;

contract RelationalContract {
    uint a = 10;
    uint b = 2;

    function isEqual() public view returns(bool) {
        return a == b;
    }

    function isNotEqual() public view returns(bool) {
        return a != b;
    }

    function isGreaterThan() public view returns(bool) {
        return a > b;
    }

    function isSmallerThan() public view returns(bool) {
        return a < b;
    }

    function isGreaterThanOrEqualTo() public view returns(bool) {
        return a >= b;
    }

    function isSmallerThanOrEqualTo() public view returns(bool) {
        return a <=b;
    }
}
```

The preceding example shows a sample code for each. The return value for each of the preceding function is in bool or Boolean.

2.5.11.3 Logical Operators

Table 2.3 shows the logical operators which also give Boolean as the output, as follows:

OPERATOR	DENOTATION	DESCRIPTION
Logical AND	&&	Returns true if both conditions are true, and false if one or both conditions are false
Logical OR		Returns true if one or both conditions are true, and false when both are false
Logical NOT	!	Returns true if the condition is not satisfied, else false

Table 2.3: Logical Operators

The following is a sample program:

```
// SPDX-License-Identifier: Some Identifier
pragma solidity ^0.8.10;

contract LogicalContract {
    uint a = 10;
    uint b = 2;
    function logicalAnd() public view returns(bool) {
        return a >= b && a > b;
    }
    function logicalOr() public view returns(bool) {
        return a == b || a > b;
    }
    function logicalNot() public pure returns(bool) {
        bool someBool = true;
        return !someBool;
    }
}
```

2.5.11.4 Bitwise Operator

These operators work at a bit level and are used to perform bit-level operations. Solidity supports the following arithmetic operators, as shown in [Table 2.4](#), as follows:

OPERATOR	DENOTATION	DESCRIPTION
Bitwise AND	&	Performs boolean AND operation on each bit of integer argument
BitWise OR		Performs boolean OR operation on each bit of integer argument
Bitwise XOR	^	Performs boolean exclusive OR operation on each bit of integer argument
Bitwise Not	~	Performs boolean NOT operation on each bit of integer argument
Left Shift	<<	Moves all bits of the first operand to the left by the number of places specified by the second operand
Right Shift	>>	Moves all bits of the first operand to the right by the number of places specified by the second operand

Table 2.4: Bitwise Operators

Refer to the following code:

```
// SPDX-License-Identifier: Some Identifier
pragma solidity ^0.8.10;

contract BitwiseContract {
    uint a = 10;
    uint b = 2;

    function bitwiseAnd() public view returns(uint) {
        return a & b;
    }

    function bitwiseOr() public view returns(uint) {
        return a | b;
    }
}
```

```

function bitwiseNot() public view returns(uint) {
    return ~a;
}

function bitwiseXOR() public view returns(uint) {
    return a ^ b;
}

function bitwiseLeftShift() public view returns(uint) {
    return a << b;
}

function bitwiseRightShift() public view returns(uint) {
    return a >> b;
}
}

```

2.5.11.5 Assignment Operators

These operators are for the assignment of a value to a variable. The operand on the left side is the variable, while operand on the right side is the value. Solidity supports the following arithmetic operators, as shown in [Table 2.5](#), as follows:

OPERATOR	DENOTATION	DESCRIPTION
Simple Assignment	=	Simply assigns the value on the right side to the operand on the left side
Add Assignment	+=	Adds the operand on the right side to the operand on the left side and assigns the value to the left operand
Subtract Assignment	-=	Subtracts the operand on the right side from the operand on the left side and assigns the value to the left operand
Multiply Assignment	*=	Multiplies both the operands and assigns the value to the left operand
Divide Assignment	/=	Divides the operand on the left side by the operand on the right side and assigns the value to the left operand

Modulus Assignment	%=	Divides the operand on the left side by the operand on the right side and assigns the remainder to the left operand
--------------------	----	---

Table 2.5: Assignment Operators

The following is a sample program:

```
// SPDX-License-Identifier: Some Identifier
pragma solidity ^0.8.10;

contract AssignmentContract {
    uint a = 100;

    function addAssignment() public returns(uint) {
        return a += 10;
    }

    function subtractAssignment() public returns(uint) {
        return a -= 10;
    }

    function multiplyAssignment() public returns(uint) {
        return a *= 10;
    }

    function divideAssignment() public returns(uint) {
        return a /= 10;
    }

    function modulusAssignment() public returns(uint) {
        return a %= 10;
    }
}
```

2.5.11.6 Conditional (or ternary) Operators

It is a ternary operator where we first evaluate the condition on the left, and depending upon true or false, either of the two values on the right is chosen and returned.

Refer to the following example:

Syntax: if condition true ? then A: else B

The following is a sample program:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract ConditionalContract {

    function condition(uint a, uint b) public pure returns(uint)
    {
        return a > b ? 11 : 22;
    }
}
```

The right use of the operators is essential in Solidity programming.

[2.5.12 Gas and Operation Cost of Ethereum](#)

Just like running a vehicle would need fuel, the execution of any Solidity operation on the Ethereum network would need gas. We all know that Ethereum is a public Blockchain that runs on the Proof of work consensus model. Similar to Bitcoin, on the Ethereum network, we need to allure and engage the miners to validate the transactions by paying a fee called gas. The other advantage of the gas is that it keeps the spammers at bay, as with each transaction, they are spending the money from their pocket in the form of gas. Hence, it's well understood that more complex the transaction is, more would be the gas price that the requester of the smart contract has to pay. Similarly, the sooner the transaction has to be executed, the more would be the price of the gas to be paid and vice versa. So, what would happen if too less an amount of gas price is associated with a transaction? Oh well, that might not allure any miner and perhaps one has to wait for a very long time before the transaction is picked up. Hence, as per the need of the business, the gas can be set by the requester.

[2.5.12.1 Ether](#)

Ether is a crypto currency that is associated with a value which keeps on fluctuating, depending upon the demand and supply in the market. One can always buy, retain, and sell the Ethers in a crypto exchange.

You can find all the units and subunits of Ether in [Table 2.6](#), as follows (<https://ethdocs.org/en/latest/ether.html>):

Unit	Wei Value	Wei
wei	1 wei	1
Kwei (babbage)	1e3 wei	1,000
Mwei (lovelace)	1e6 wei	1,000,000
Gwei (shannon)	1e9 wei	1,000,000,000
microether (szabo)	1e12 wei	1,000,000,000,000
milliether (finney)	1e15 wei	1,000,000,000,000,000
ether	1e18 wei	1,000,000,000,000,000,000

Table 2.6: Units of Ether

2.5.12.2 Gas vs. Ether

Please note that Gas and Ether are two different things. Unlike Ether, gas is associated with the cost of the operation for running the smart contract. However, internally, the gas is converted to Ether, though not directly. But how? Let's see.

The equation of the Transaction Fee associated with any operation on the Ethereum network is as follows:

$$\text{Tx Fees} = \text{Gas Limit} * \text{Gas Price}$$

2.5.12.3 Gas limit

It's the maximum value that the user is ready to pay to run an operation. The minimum price of the Gas limit is 21,000. If the entire value is not consumed, then the rest is returned to the user's account after the execution of the transaction. The Gas limit helps

the user to safeguard his money that he is paying on the transaction with a maximum limit.

2.5.12.4 Gas price

It's the per unit price, which is calculated in Gwei. This is an ever-fluctuating value which can be found at the <https://ethgasstation.info/> website. The website gives an idea on the gas price that one needs to spend for executing a transaction at a fast rate, i.e., <2m, or standard rate, i.e., <5m, or low pace, i.e., <30 mins. You can use its calculator by clicking on the Tx Calculator link in LHS to get a full idea of how much money you are spending here.

This final transaction fee is calculated in Ether.

Gas is a very small fraction of Ether (the crypto currency associated with Ethereum).

If we visit <https://etherscan.io/txs>, we can view the Gas Limit, Gas Used by Transaction, Gas Price, and Transaction Fee in Ether.

2.5.12.5 Future of Gas

With Ethereum 2.0 working in full strength, the consensus model would switch from the "Proof of Work" to "Proof of Stake" model. Hence, perhaps the role of the miners would be minimal. And hence, the gas cost might be minor or non-existing. It would be a good news for all of us as the cost of the execution of contracts would be far less, and hence may attract many more users in the market.

2.5.13 Storing Data in Solidity

Now that we have a good amount of knowledge on all the different elements of a Solidity contract, it's also time to figure out how to write Smart Contracts following best practices, so that the code is optimized for a minimum usage of gas and, at the same time, is not affecting any of the functionalities adversely.

We know that the Smart contracts are compiled, deployed, and run on the Ethereum Virtual Machine.

Can Solidity smart contracts store a huge amount of data in the contracts? No, because the storage capacity is limited and also the storing of data on a contract needs a lot of cost in terms of the consumption of gas. Hence, we must write the code judiciously, and hence, we need to have a good idea on the data storing mechanism in EVM.

The EVM uses four different types of memory locations to save the associated data in a smart contract. They are Storage, Memory, Calldata, and Stack.

2.5.13.1 Storage

Storage is the area where all the state variables are stored.

It's the permanent memory associated with each account where the data is stored in the key value pair. Every Ethereum account has the access only to its own Storage, where it can store and retrieve data.

Refer to the following code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract StorageContract {
    uint varStorage1;
    uint varStorage2;

    function saveInStorage() public {
        varStorage1 = 10;
        varStorage2 = 20;
    }
}
```

The cost of gas is the maximum when associated with the data in Storage. Hence, we need to write the code in such a way that the number of state variables are minimum to save the associated gas cost.

2.5.13.2 Memory

Memory is a temporary storage area that you can use within the functions to save the data by applying a special keyword “memory” in front of a variable. It can’t be used to retain the data in between the calls, and hence, the gas cost in storing the data in memory is far less than saving the same in storage.

Refer to the following code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract StorageMemory {

    function saveInMemory() public pure returns(string memory,
string memory){
        string memory varStorage1 = "";
        string memory varStorage2 = "";
        return (varStorage1, varStorage2);
    }
}
```

Please also note that the memory type of storage is applicable only to the variables that pass by reference, i.e., Array, Struct, String, Mappings etc.

2.5.13.3 Calldata

Every variable of the type reference can be stored in either of the three data locations, i.e., Storage, Memory, or Calldata. Calldata is a non-modifiable, non-persistent area with almost a memory type of a temporary storage, and therefore, only stores the function arguments.

2.5.13.4 Stack

If the Solidity program needs to store small amount of local data inside the functions to make sure of low usage of gas, then we have to choose Stack. Stack is the data area for performing the calculations and it can store a maximum of up to 1024 elements with a maximum value of 256 bits.

Refer to the following code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract StackContract {
    function saveInStack() public pure returns(uint, uint){
        uint varFirst = 10;
        uint varSecond = 20;
        return (varFirst, varSecond);
    }
}
```

In Stack storage, we do not need to specify any keywords to save the data to Stack as it's the default choice inside a function to save the data. Among Storage, Memory, and Stack, the Stack uses the least gas cost for saving the data, and yet can hold a limited amount of data. Please note that the variables that pass by reference, i.e., Array, Struct, String, Mappings etc., can be stored either in storage or in memory and not in stack, which is only meant for storing the data in small amounts. Also, it's worth mentioning that while writing error free code is a mandate in any programming language, it's always advisable to write with best practices and attend to all the warnings for adhering to the coding standards and deliver quality.

2.5.14 Control of Flow in Solidity

Writing some complex algorithm in a smart contract also needs to execute many different types of loops. Some of the loops that Solidity supports are the following:

- if, else,
- while, do,
- for,
- break,
- continue,
- return,

- ? :

However, unlike the other languages, Solidity does not support Switch and Goto statements.

Flows have to be used with precaution with enough testing as they run many times and may lead to undesirable use and wastage or resources or even infinite loops.

2.5.14.1 if-else Condition

Refer to the following code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract IfElseContract {
    function getResult(uint a, uint b, uint c) public pure
    returns(uint) {
        uint result;

        // if else statement
        if(a > b && a > c) {
            result = a;
        } else if(b > a && b > c){
            result = b;
        } else {
            result = c;
        }
        return result;
    }
}
```

The preceding example is an if-then-else statement. We can write the if-else statement when the number of conditions are limited to two.

2.5.14.2 while-do Loop

Refer to the following code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract WhileDoContract {

    function getResult(uint number) public pure returns(uint) {
        // while do statement
        while (number < 10) {
            number+= 25;
        }
        return number;
    }
}
```

2.5.14.3 do-while Loop

Refer to the following code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract DoWhileContract {

    function getResult(uint number) public pure returns(uint) {
        // do while statement
        do {
            number+= 25;
        } while (number < 10);
        return number;
    }
}
```

2.5.14.4 for Loop

Refer to the following code:

```
/// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract ForLoopContract {
```

```
function getResult(uint number, uint times) public pure
returns(uint) {
    uint result;

    // for loop
    for (uint i = 0; i< times; i++) {
        result += number;
    }
    return result;
}
```

2.5.14.5 break Statement

Refer to the following code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract ForLoopContract {
    function getResult(uint number, uint times) public pure
    returns(uint) {
        uint result;

        // break statement
        for (uint i = 0; i< times; i++) {

            if (result > 30)
            {
                break;
            }
            result += number;
        }
        return result;
    }
}
```

2.5.14.6 Continue Statement

Unlike the break statements, the continue statements continue the loop and yet do not let the next line get executed in the current loop. Run the preceding program by replacing break with continue and observe the output.

2.5.14.7 Conditional (or ternary) Operators

We already discussed a ternary operator where we first evaluate the condition on the left and, depending upon true or false, either of the two values on the right is chosen and returned.

Refer to the following example:

Syntax: if condition true ? then Result1: else Result 2

The following is the sample code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract ConditionalContract {
    function condition(uint a, uint b) public pure returns(uint)
    {
        return a > b ? 11 : 22;
    }
}
```

You can observe how simple and concise the code is in comparison to the if-else statement. Ternary operators are used when the number of conditions are limited to only two. However, if they are more, then we have to use the if-then-else statement.

2.5.15 View Function

In a function marked with a view keyword, we are not supposed to modify any of the state variables.

The following statements are considered to be modifying the state:

1. Writing to state variables
2. Emitting events
3. Creating other contracts

4. Using selfdestruct
5. Sending Ether via calls
6. Calling any function not marked as view or pure
7. Using low-level calls

Refer to the following code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract ViewContract {
    uint a = 10;
    uint b = 2;
    function addition() public view returns(uint) {
        return a + b;
    }
}
```

2.5.16 Pure Function

In a function marked with a pure keyword, we do not update the state variables, and don't even read them.

Refer to the following code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract ViewContract {
    function addition() public pure returns(uint) {
        uint a = 10;
        uint b = 2;
        return a + b;
    }
}
```

Please note that running the view and pure functions would not involve any gas cost. However, if they are called from another function, then the cost would be there.

Refer to the following code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract PureFunctionTest {
    uint c;
    function updateLocalVariables(uint a, uint b) public pure
    returns (uint) {
        return a + b ;
    }
    function updateStateVariable(uint x, uint y) public {
        c = updateLocalVariables(x, y);
    }
}
```

We should use these two keywords, pure or view, as much as possible in the code for the optimal usage of gas.

2.5.17 Fallback Function

A contract can optionally have one fallback function of a particular specification. The fallback function must have no arguments and no return statement, and must have external visibility. It can be marked as payable to receive the Ethers.

Refer to the following code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract CalledContract {
    fallback() external payable {}
}

contract CallerContract {
    function callSink(CalledContract calledContract) public
    returns (bool) {
        address payable calledContractPayable =
        address(calledContract);
    }
}
```

```
        return (calledContractPayable.send(2 ether));
    }
}
```

2.5.18 Error Handling

Error handling refers to the routines in a program that respond to the abnormal input or conditions. Similar to the high level languages such as Java, Scala, JavaScript etc., Solidity has a feature to revert the state changes in case of any issues occurring while running the code. There are quite a few different ways to handle the exceptions. The following is the list:

- require
- revert
- assert
- try/catch

2.5.18.1 Require

Require might be useful for validating the inputs to a function or the output from an external contract or for running to check the parameters before running a condition.

Refer to the following code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract RequireFunctionTest {
    uint someValue;

    function testRequireWithoutString(uint localValue) public
    {
        require(localValue >= 100);
        someValue = localValue;
    }

    function testRequireWithString(uint localValue) public {
```

```
require(localValue >= 100, "Value should be either 100 or
more");
someValue = localValue;
}
function retrieveValue() public view returns (uint) {
    return someValue;
}
}
```

2.5.18.2 Revert

The `revert` function is another way to trigger the exceptions from within the other code blocks to flag an error and revert the current call. The function aborts the execution and reverts any changes done to the state. It's especially useful before running any complex logical flow.

The function takes an optional string message containing the details about the error that is passed back to the caller.

The following example shows how to use an error string together with `revert`:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract RevertFunctionTest {
    uint someValue;

    function testRevert(uint localValue) public {
        if (localValue >= 100) {
            revert("Value too high");
        }
        someValue = localValue;
    }

    function retrieveValue() public view returns (uint) {
        return someValue;
    }
}
```

If we assign any value to the testRevert function that is higher or equal to 100, then the function throws an exception and the value is not assigned to the state variable.

2.5.18.3 Assert

The `assert` function should only be used to test for internal errors, and to check the invariants, for example, the overflow and underflow conditions. `assert()` is just there to prevent anything really bad from happening. The following is an example:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract AssertFunctionTest {
    uint someValue;

    function testAssert(uint localValue) public {
        assert(localValue < 100);
        someValue = localValue;
    }

    function retrieveValue() public view returns (uint) {
        return someValue;
    }
}
```

2.5.18.4 Try/Catch

Try/Catch can only be used in calling the external functions or when creating a contract. Please check the following code where a failure in an external is caught using a try/catch statement:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract ExchangeRates{
    constructor(uint currencyInINR) {
        currencyInINR/75;
    }
}
```

```

}

contract TryCatchTest {
    ExchangeRates exchangeRates;
    function convertINT2USD(uint INRvalue) public returns (bool)
    {
        try new ExchangeRates(INRvalue) {
            return (true);
        } catch Error(string memory) {
            return (false);
        } catch (bytes memory) {
            return (false);
        }
    }
}

```

2.5.19 Events

Events are an abstraction on top of the Ethereum Virtual Machine's logging functionalities. They are especially useful for integration with the frontend code. The front end code can work as an observer with the call back functions waiting for a particular event to have occurred. In the decentralized applications, events play a major role in knowing the status of a transaction. For executing an event, we must first declare an event, and then emit it with some useful data that would be logged through them.

Refer to the following code:

```

//Declare an Event
eventEventName(address fromAddress,uintvalue);

//Emit an event
emit EventName(msg.sender,indexNumber);

```

The Application Binary Interface or ABI is a JSON artifact produced by the Solidity compiler when compiling a smart contract. The ABI defines the interface of a smart contract that may contain the following:

- All the functions names that can be called from outside the smart contract
- Function signatures (function names, argument types, and return types)
- Events in the contract

However, the ABI does not contain the details of the function or the event's implementation.

The ABI is used outside the smart contract by the Ethereum client libraries like web3 to interact with the smart contract. Hence, it would need only the details of the functions of the public and external types. Only the public and external types of functions are part of the ABI interface, which means the private and internal types are not represented. The reason being that ABI is used for integration with the front-end code.

For an example, please compile the following code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract Covid19Testing {
    uint numberOfPositivePatients;
    event NewPositiveFound(address fromAddress, uint number); //
    Event

    function testForCovidPositivePublic() public {
        // if a new positive is found
        emit NewPositiveFound(msg.sender,
            ++numberOfPositivePatients); // Triggering event
    }

    function testForCovidPositivePublicInternal() internal {
        emit NewPositiveFound(msg.sender,
            ++numberOfPositivePatients); // Triggering event
    }

    function testForCovidPositivePublicExternal() external {
        emit NewPositiveFound(msg.sender,
            ++numberOfPositivePatients); // Triggering event
    }
}
```

```

    }

    function testForCovidPositivePublicPrivate() private {
        emit NewPositiveFound(msg.sender,
            ++numberOfPositivePatients); // Triggering event
    }
}

```

Once you compile, you would find the following as the output:

```

[
  {
    "anonymous": false,
    "inputs": [
      {
        "indexed": false,
        "internalType": "address",
        "name": "fromAddress",
        "type": "address"
      },
      {
        "indexed": false,
        "internalType": "uint256",
        "name": "number",
        "type": "uint256"
      }
    ],
    "name": "NewPositiveFound",
    "type": "event"
  },
  {
    "inputs": [],
    "name": "testForCovidPositivePublic",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  },
  {

```

```
    "inputs": [],
    "name": "testForCovidPositivePublicExternal",
    "outputs": [],
    "stateMutability": "nonpayable",
    "type": "function"
  }
]
```

Only the public and external functions are represented on ABI, whereas the other two are not.

[2.5.19.1 Event Logs in Solidity](#)

We all know that Ethereum is a public blockchain where the transactions are continuously stored in Blocks. Every transaction can be associated with logs by invoking the Events where the custom data can be stored in the Block and remain associated with the contract's address permanently. However, there are some limitations for accessing the event data.

The logs associated with the events help in the integration with the frontend code and even in the integration with the other contracts at times.

An event can be declared using the event keyword. Refer to the following example:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract Covid19Testing {
    uint numberOfPositivePatients;
    event NewPositiveFound(address fromAddress, uint number); //
    Event

    function testForCovidPositive() public payable {
        // if a new positive is found
        emit NewPositiveFound(msg.sender,
            ++numberOfPositivePatients); // Triggering event
    }
}
```



```
}
```

Now, the ABI related to this code can be found after compiling it, as shown in the following example:

```
[
  {
    "anonymous": false,
    "inputs": [
      {
        "indexed": false,
        "internalType": "address",
        "name": "fromAddress",
        "type": "address"
      },
      {
        "indexed": false,
        "internalType": "uint256",
        "name": "number",
        "type": "uint256"
      }
    ],
    "name": "NewPositiveFound",
    "type": "event"
  },
  {
    "inputs": [],
    "name": "testForCovidPositive",
    "outputs": [],
    "stateMutability": "payable",
    "type": "function"
  }
]
```

Here, we can find the interface details of an event `NewPositiveFound` and secondly the interface details of a function named `testForCovidPositive`. In case of an Event, we can find the

inputs, whereas in the function, we can find both the input and the output types.

If we delete all the code from the contract and make it blank, then we can see that the ABI becomes [], i.e., blank too.

Now, access the contract's event in the JavaScript code. Refer to the following code:

```
var abi = /* abi as generated using compiler */;
var ClientReceipt = web3.eth.contract(abi);
var clientReceiptContract = ClientReceipt.at("0x1234...ab67" /*
address */);

var event = clientReceiptContract.Deposit(function(error,
result) {
    if(!error) console.log(result);
});
```

It should print the details similar to the output given in the following section.

2.5.19.2 Output

Refer to the following output:

```
{
  "returnValues": {
    "_from": "0x1111...FFFFC"CC",
    " "id": "0x50...sd5ad"20",
    " "_va"ue": "0x420"42"
  },
  " "aw": {
    " d"ta": "0x7f...91"85",
    " "top"cs"" ["0xfd4...b4e"d7", "0x7f...1a91"85"]
  }
}
```

2.5.20 Object Oriented Approach of Solidity

People who have some background in any Object-oriented programming, must be aware that OOP has four basic concepts, which are as follows:

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

Now, let's explore these features one by one.

2.5.20.1 Encapsulation

Encapsulation or data hiding is a process where the data and the function that gives access to the data work as a single unit. Data in Solidity, by default, is of private visibility and the functions that give access to the data are public.

Refer to the following code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

// A simple contract with getter and setter functions
contract GetterSetterContract {
    string value = "Some Value";

    function getValue() public view returns(string memory) {
        return value;
    }

    function setValue(string memory newValue) public {
        value = newValue;
    }
}
```

2.5.20.2 Abstraction

Abstraction is a process where the implementation details are hidden, and only the essential information are shown. In Solidity,

Abstraction can be achieved with either abstract classes or interfaces.

2.5.20.3 Inheritance

Inheritance is a feature in which one object acquires all the properties and behaviors of a parent object. Inheritance promotes the reusability of the code as we can have one contract at the top level with generic features and many child contracts inheriting those as well as more features of their own.

In the following example, we have two contracts. The first one is the parent contract and the second one is a child contract.

This behavior of the same function working differently in the parent and the child contract is called function overriding, and it uses the **virtual** keyword in the parent contract and the **override** keyword in the child contract.

The following is an example of function overriding:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract Mammal {
    function mammalFunction() public pure virtual returns(string
memory) {
        return "Mammals produce offspring by directly giving birth
and females have mammary glands";
    }
}

contract SpecialMammal is Mammal {

    function mammalFunction() public pure override
returns(string memory) {
        return "Humans are special mammals who can speak and walk
with two legs";
    }
}
```

We can also prefer to keep the parent function intact in the child contract, as shown as follows:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract Mammal {
    function mammalFunction() public pure returns(string memory)
    {
        return "Mammals produce offsprings by directly giving
        birth and females have mammary glands";
    }
}

contract SpecialAnimal is Mammal {
    function specialFunction() public view returns(string
memory) {
        this.mammalFunction();
        return "Special mammals can walk with two legs ";
    }
}
```

Solidity also supports multiple inheritance, as shown in the following code example:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract Mammal {
    function mammalFunction() public pure returns(string memory)
    {
        return "Mammals produce offsprings by directly giving
        birth and females have mammary glands";
    }
}

contract Bird {
    function birdFunction() public pure returns(string memory) {
```

```

        return "Birds produce offsprings by laying eggs and
        hatching them";
    }
}

contract SpecialAnimal is Mammal, Bird {
    function specialFunction() public view returns(string
    memory) {
        this.mammalFunction();
        this.birdFunction();
        return "Special Animal can Birds produce offsprings by
        laying eggs and hatching them. Theie females have mammary
        glands ";
    }
}

```

2.5.20.4 Polymorphism

Polymorphism is a feature, by which we can perform a single action in different ways. In Solidity, we can perform polymorphism by **function overloading**, by using the same **virtual** keyword in the parent contract and the **override** keyword in the child contract.

Here, when we run the Mammal contract and then run mammalFunction, the output would be "Mammals produce offspring by directly giving birth and females have mammary glands"; however, if we deploy and run the SpecialMammal contract, the output of the function mammalFunction would be "Humans are special mammals who can speak and walk with two legs".

The following is an example of function overloading when a function with the same name can be used multiple times and each has a different set of input parameters:

```

// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract Calculator {
    function calculate(uint a, uint b) public pure returns(uint)
    {

```

```
    return a*b;
}

function calculate(uint a, uint b, uint c) public pure
returns(uint) {
    return a*b/c;
}
}
```

2.5.21 Abstract Contracts and Interfaces

In larger projects, we need to handle multiple Solidity files and a huge number of contracts interacting with each other. Also, in order to make the code modular, we might need to exploit the object-oriented features of Solidity. To facilitate this, we usually use two of the following ways:

- Abstract contracts
- Interfaces

2.5.21.1 Abstract Contract

We can transform any contract to an abstract contract just by adding a keyword `abstract` before the contract. The abstract contract optionally can have one or more functions that are only defined and yet not implemented. Such functions have to be marked as “virtual”. Refer to the following code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;
abstract contract AbstractContract {
    function someFunction() public pure virtual returns(string
memory);
}
```

Please note that an abstract contract might have all the functions implemented and yet just adding the `abstract` keyword makes the contract abstract. Refer to the following code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;
abstract contract AbstractContract {
    function someFunction() public pure returns(string memory) {
        return "some message";
    }
}
```

Now, after compiling, try to deploy and run any of the preceding two contracts and you will find the error message “This contract may be abstract, and does not implement an abstract parent’s methods completely or invoke an inherited contract’s constructor correctly.”

Now, let’s add another contract to the same file and implement the same. Refer to the following code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;
abstract contract AbstractContract {
    function someFunction() public pure virtual returns(string
memory);
}
contract ImplementorContract is AbstractContract {
    function someFunction() public override pure returns(string
memory) {
        return "some message";
    }
}
```

You can find an “override” keyword that marks that the function in the ImplementorContract is overriding a function in the AbstractContract.

Now, again we can compile it and deploy the second contract by choosing the one from the REMIX browser’s list of contracts, and finally run this without any issue.

[2.5.21.2 Interface](#)

Interfaces are another type of facility in contracts to promote **abstraction**. In interface, we can only define the functions and not implement them. Also, all the functions have to be marked as “external”. Apart from functions, interfaces can also declare enums, structs, and events. Refer to the following code:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

interface SomeInterface {
    function someFunction() external pure returns(string
memory);
}
```

However, just like the abstract contract, an interface can't be deployed and would receive the following error:

“This contract may be abstract, and does not implement an abstract parent's functions completely or invoke an inherited contract's constructor correctly.”

Now, if we implement the following interface, we can find that it's working:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;
interface SomeInterface {
    function someFunction() external pure returns(string
memory);
}
contract ImplementorContract is SomeInterface {
    function someFunction() public override pure returns(string
memory) {
        return "some message";
    }
}
```

In Solidity, we can also implement multiple interfaces at one time, as shown as follows:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

interface SomeInterface1 {
    function someFunction1() external pure returns(string
memory);
}
interface SomeInterface2 {
    function someFunction2() external pure returns(string
memory);
}

contract ImplementorContract is SomeInterface1,
SomeInterface2 {
    function someFunction1() public override pure returns(string
memory) {
        return "some message 1";
    }

    function someFunction2() public override pure returns(string
memory) {
        return "some message 2";
    }
}
```

2.5.22 Creating and Destroying Contracts

We can instantiate a contract from another contract just by using the “new” keyword. Using the new keyword, we instantiate the new instance of the contract and use that newly created contract instance. Let’s do this with an example.

Deploy only the ClientContract and run invokeCallMe and watch how the other contract works, even if not deployed, as shown in the following example:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract ServerContract {
```

```

    function callMe() public pure returns(uint) {
        return 5;
    }
}

contract ClientContract {

    function invokeCallMe() public returns(uint) {
        ServerContract s = new ServerContract();
        return s.callMe();
    }
}

```

2.5.22.1 Destruction of a Contract

We can deactivate or self-destroy a contract when the contract is no longer needed. Deactivation of a contract frees up the space on the blockchain by clearing all of the contract's data.

But why would we need to do so? There can be many reasons.

A Smart Contract can be terminated for any of the following reasons:

- By breach of contract
- By agreement between the parties
- By performance of the party's contractual obligations
- By frustration
- By inability to perform
- By fraud

We can self-destroy a contract by invoking the selfdestruct function, as shown in the following example:

```

// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract SelfDestructContract {
    address payable owner;

    constructor() {

```

```

    owner = msg.sender;
}

function calculate(uint a, uint b) public pure
returns(uint) {
    return a*b;
}

function destroyContract() public {
    if (msg.sender == owner) {
        selfdestruct(owner);
    }
}
}

```

After running the `destroyContract` function once, the calculation would not work again.

2.5.23 Lab 1

Now that we have learnt most of the features of Solidity, the following is a sample code that you can run to know how the students can enroll to a program with a Solidity smart contract:

```

// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract StudentEnrollmentContract {
    address private owner;
    Student[] private students;
    uint private latestEnrollmentNumber;

    struct Student {
        string name; // short name
        string dob; // date of birth in ddMMyyyy format
        string grade; //score assigned to student
        uint enrollmentNumber; // enrollmentNumber that keeps on
        increasing with each new student
    }
}

```

```

constructor() {
    owner = msg.sender;
}

function addNewStudent(string calldata newStudentName,
string calldata newStudentDob, string calldata
newStudentGrade) public returns(uint){
    if (msg.sender != owner) {
        return 0;
    }

    uint newEnrollmentNumber = ++latestEnrollmentNumber;
    students.push(Student({
        name: newStudentName,
        dob: newStudentDob,
        grade: newStudentGrade,
        enrollmentNumber: newEnrollmentNumber
    }));

    return newEnrollmentNumber;
}

function retrieveGrade(uint studentEnrollmentNumber)
public view returns(string memory){
    for (uint i=0; i < students.length; i++) {
        if (students[i].enrollmentNumber ==
studentEnrollmentNumber) {
            return students[i].grade;
        }
    }
}
}

```

2.5.24 Lab 2

Now, let's rewrite the contract of Lab1 as Lab2 using more than one contracts and instantiating one within the other:

```
// SPDX-License-Identifier: SOME IDENTIFIER
```

```

pragma solidity ^0.8.10;

contract StudentDetails {
    string name; // short name
    string dob; // date of birth in ddMMyyyy format
    string grade; //score assigned to student
    uint enrollmentNumber; // enrollmentNumber that keeps on
    increasing with each new student

    constructor(string memory _name, string memory _dob,
    string memory _grade, uint _enrollmentNumber) {
        name = _name;
        dob = _dob;
        grade = _grade;
        enrollmentNumber = _enrollmentNumber;
    }

    function getName() public view returns(string memory
    _name) {
        return name;
    }

    function getDob() public view returns(string memory _dob)
    {
        return dob;
    }

    function getGrade() public view returns(string memory
    _grade) {
        return grade;
    }

    function getEnrollmentNumber() public view
returns(uint _enrollmentNumber) {
        return enrollmentNumber;
    }
}

contract StudentEnrollmentContract {
    address private owner;
    StudentDetails[] private students;

```

```

uint private latestEnrollmentNumber;

constructor() {
    owner = msg.sender;
}

function addNewStudent(string calldata newStudentName,
string calldata newStudentDob, string calldata
newStudentGrade) public returns(uint){
    if (msg.sender != owner) {
        return 0;
    }

    uint newEnrollmentNumber = ++latestEnrollmentNumber;
    StudentDetails newStudent = new
    StudentDetails(newStudentName, newStudentDob,
    newStudentGrade, newEnrollmentNumber);
    students.push(newStudent);
    return newEnrollmentNumber;
}

function retrieveGrade(uint studentEnrollmentNumber)
public view returns(string memory){
    for (uint i=0; i < students.length; i++) {
        if (students[i].getEnrollmentNumber() ==
        studentEnrollmentNumber) {
            return students[i].getGrade();
        }
    }
}
}

```

2.5.25 Ethereum Tokens and ERC Standards

Ethereum tokens are simply digital assets that can be represented and traded on the Ethereum blockchain. The advantage of such tokens is that they can be built really quickly from scratch and deployed on Ethereum in almost no time. Also, Ethereum being a public network, there is no extra infrastructure work that is needed.

So, how can we write a smart contract for trading tokens?

Now that we have learnt the basics of Solidity, we can write many smart contracts and deploy them on the public Ethereum network as per the business need. However, with the rising popularity of Solidity, and many contracts deployed in production, the programmers realized certain loopholes, and as a result, certain best practices were created and followed as a standard.

2.5.25.1 Ethereum Request for Comments

An 'Ethereum Request for Comments' (ERC) is a document that the smart contract programmers follow for using the Ethereum blockchain platform. They describe the rules in these documents that the Ethereum-based tokens must comply with.

2.5.25.2 ERC20

ERC20 is the most popular token standard and many tokens created using the ERC20 standards are running on the Ethereum platform today. ERC20 is fungible in nature, i.e., Fungibility is the ability of a good or an asset to be interchanged with the other individual goods or assets of the same type. The Fungible tokens are like digital cash and are widely used in creating crypto currencies. Hence, if you wish to create an ERC20 fungible token, you have to follow the standard by implementing the following functions and events:

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns
        (uint256);
    function transfer(address recipient, uint256 amount)
        external returns (bool);
    function allowance(address owner, address spender) external
        view returns (uint256);
```



```
function approve(address spender, uint256 amount) external
returns (bool);
function transferFrom(address sender, address recipient,
uint256 amount) external returns (bool);
event Transfer(address indexed from, address indexed to,
uint256 value);
event Approval(address indexed owner, address indexed
spender, uint256 value);
}
```

ERC20 standard code provides the following six functions:

- `totalSupply` function determines the total number of tokens
- `balanceOf` shows the balance of the account specified by the `address_owner` parameter, where `_owner` is the address
- `transfer` implements the transfer of tokens from the primary address to the address of an individual user
- `transferFrom` is used to transfer tokens from one user to another
- `approve` checks whether the tokens remain in the smart contract and provides a withdrawal of funds from the account up to the maximum allowable amount, which is specified as a parameter of the function
- `allowance` guarantees that there are enough tokens at the sender's wallet to transfer them to the recipient

Also, ERC-20 provides for the following two types of events:

- `transfer` – transfer event of tokens between accounts
- `approval` – the event that becomes active upon the successful execution of the `approve` function

It's worth mentioning that the ERC20 token standard has a bug with its `transfer` function that we just covered. This bug has burnt the tokens worth millions of US dollars. With the `transfer` function, you can only send the tokens to the recipient's account. If you use the "transfer" function, you will see a successful transaction, but the receiver's address will never receive the tokens. This burns the

tokens forever, and it can't be retrieved. By using the wrong function, several users have lost their tokens for good.

Hence, post ERC20, there are quite a few other standards created for creating different types of tokens. A few are quoted as follows:

1. ERC 721 - Standard for non-fungible tokens. A non-fungible token, also known as a nifty, is a special type of cryptographic token which represents something unique; non-fungible tokens are thus not mutually interchangeable by their individual specification.
2. ERC 165
3. ERC 223
4. ERC 621
5. ERC 777
6. ERC 827
7. ERC 884
8. ERC 865

2.5.26 Truffle and Ganache

Truffle Suite is one of the most widely used development environment based on the Ethereum Blockchain, where the developers can develop, compile, and deploy the Solidity smart contracts and also integrate them with front-end for testing. Truffle Suite comes with the Truffle and Ganache tools for a faster development.

2.5.26.1 Truffle

It is a Development Environment, Testing Framework, and Asset pipeline for the Ethereum Blockchain. It's extremely useful for any Solidity developer, whosoever wishes to dive into the Ethereum development and needs a framework, so that they can better organize their DApp development assets and not worry about manually setting up a test environment. First, let's learn how to install Truffle and Ganache. In order to install Truffle, please make sure that

you have the latest versions of the node installed in your machine. Check their versions with the following commands:

```
node -v  
npm -v
```

Requirements: NodeJS v8.9.4 or later, Windows, Linux or Mac OS X
You can install Truffle by using the following command where -g stands for a global installation:

```
npm install -g truffle
```

2.5.26.2 Ganache

Ganache is a personal Ethereum Blockchain used to test the smart contracts where you can deploy the contracts, develop the applications, run the tests, and perform other tasks without any cost. Similarly, download Ganache from the following link:

<https://github.com/trufflesuite/ganache/releases>

Now, let me run the Ganache that I have already installed on my machine.

Once you run Ganache, you should see a screen as shown in [Figure 2.1](#), as follows:

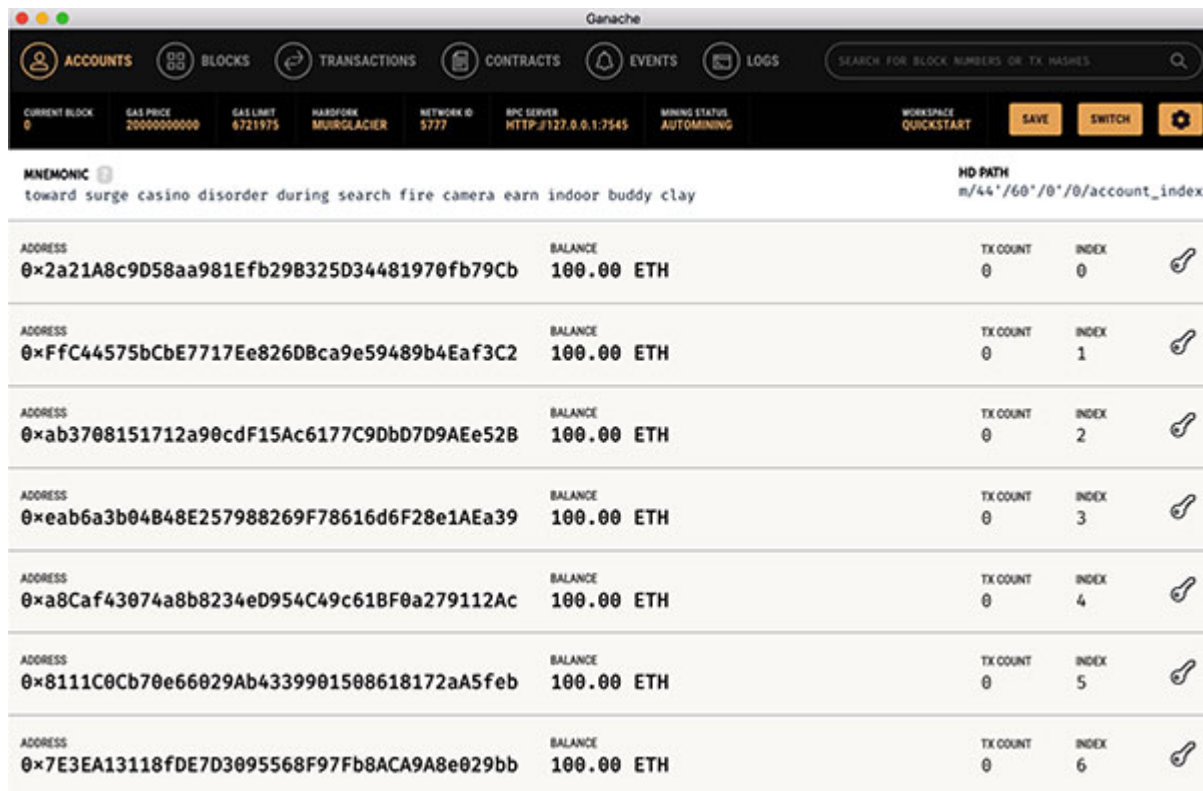


Figure 2.1: Ganache Screen

Ganache is an improved version of a virtual blockchain, earlier known as “Testrpc”. Ganache sets up 10 default Ethereum addresses, complete with the private keys and all, and each pre-loaded with 100 Ethers. The beauty of Ganache is that there is no requirement of “mining”, and it immediately confirms any transaction coming its way. This fast-tracks the development. You can write the unit tests for your code which executes on this simulated blockchain, deploys the smart contracts, plays around, and calls the functions. Every time you restart Ganache, it’s freshly ready for further simulation or new tests, returning all the addresses to their initial state of 100 Ether.

Ganache comes in two forms – UI based, that we have already run, and CLI (command line interface). The UI version is recommended because of its simplicity. When we run Ganache, it will run on a particular default port and IP address, which is **localhost:7545** or **127.0.0.1:7545**.

Please note that Ganache should only be used for development. It can't mimic the exact behaviour of the main network as there are a few problems here, which are noted as follows:

1. There are no miners on Ganache. Because of this, you cannot accurately reproduce the miner actions on the main network.
2. The gasLimit on the main network is a moving target and it can be altered by the miners from time to time. If, for some reason, you are depending on an exact number (as you can set in Ganache), you may find that you will run into issues.

Truffle comes with a set of boilerplates to help the developers quickly build the distributed applications on the Ethereum blockchain. These boilerplates are called Truffle Box. These boxes help fast track the learning process, as through them, Truffle comes with certain existing projects that we simply need to unbox, compile, deploy, and run.

Please note that the reason why Solidity is the only smart contract language that has been covered at length in this book is because, not only Ethereum, but a number of other public Blockchains use Solidity as their official language for smart contracts. We will learn about this in the later chapters.

[2.6 Wallet](#)

Ethereum comes with many different types of wallets for iOS, android, and desktop. Some of the top picks are Metamask, Exodus, Binance, Coinbase, Kraken, Gemini etc.

[2.7 Live Projects](#)

Being the first in the market and the most popular, Ethereum has the maximum number of Blockchain projects running in production. While the earlier Ethereum was mostly used in the ICO (Initial Coin Offering for crowd funding) projects, nowadays, they are used in many different ways. Some of them are listed as follows:

- **MetaMask:** Web browser plug-in that connects your device to the Ethereum network. MetaMask also enables peer-to-peer sharing and token swapping.
- **Rarible:** The first community owned NFT marketplace.
- **Civic:** Secure identity and data management on the Blockchain.
- **Audius:** A music streaming and sharing platform that puts the power back into the content creators' hands.
- **Compound Finance:** Opensource protocol for algorithmic, efficient money markets on the Ethereum network.
- **Zapper:** Tracks and visualizes your DeFi assets and liabilities on a simple dashboard.

2.8 Altcoins

With Ethereum, started an age of “Altcoins” or the arrival of an array of cryptocurrencies, other than Bitcoin. Most Altcoins or alternative cryptocurrencies, also known as “Ethereum Killers” were introduced to the market by “Initial Coin Offering” or ICO which are fundraising events just like IPO (Initial Public Offering) and are listed in the crypto exchanges. Depending upon their inherent architecture, features, associated team, funders, co-founders, revenue model etc., they are to be evaluated in the market for growth.

As per CoinMarketCap, in early 2021, “Crypto Market Cap Hits \$1 Trillion — Some Think It Will Double in Six Months”.

But as per the Economics Times in November 2021, “Over the last few years, the crypto-asset space has grown and today more than 13,000 Altcoins collectively command a market cap of USD1.4 trillion.”

Just like Ethereum, many of these Altcoins were programmable and yet different from Bitcoin or Ethereum in one way or the other. Be it their consensus mechanism, L2 architecture etc., they worked on the weak points of Ethereum, gradually leading the market to “Money 2.0” or a new kind of cash that is programmable, and works on the Internet in a fast and reliable manner. In the next few chapters, we

will cover some of the leading Altcoins in the market that has taken the market by storm.

2.9 Summary

In this chapter, we covered the following topics:

- Consensus model of Ethereum with some of Ethereum I's bottlenecks.
- Introduction to Ethereum's most widely used Smart Contract language Solidity.
- Covering functions, variables, operators, gas, storage of data, Control flow with condition stamens, read-only functions etc.
- Most popular projects on Ethereum
- Introduction to Altcoins, ICOs, and NextGen Blockchains in the make.

References

- Intro to Ethereum - <https://ethereum.org/en/developers/docs/intro-to-ethereum/>
- 100+ Ethereum Apps You Can Use Right Now [2021 Update] - <https://consensys.net/blog/news/90-ethereum-apps-you-can-use-right-now/>

Quiz

Q1: Choose the correct answer from the following options.

- A. Solidity is an object-oriented language
- B. Solidity is a contract language
- C. Solidity handles the monetary values associated with the transactions on Ethereum
- D. All of the above

Q2: Solidity is preferred over Vyper because of which of the following?

- A. More resources are available on Solidity
- B. Most Ethereum projects in productions run on Solidity
- C. Solidity has many available frameworks to make the developers life easier
- D. All of the above

Q3: Why are JVM languages not suitable for Ethereum?

- A. Because JVM languages are difficult to learn
- B. Because JVM languages are object oriented
- C. Because EVM needed a language that can handle the monetary part of the transactions
- D. None of the above

Q4: When to use Assembly language for writing contracts?

- A. When we need fine grained control on logic
- B. When writing libraries for Solidity
- C. When we need to pay less per transaction
- D. All of the above

Q5: The syntax of Solidity is close to which of the following?

- A. Java
- B. Scala
- C. Ruby
- D. JavaScript

Q6: On what DLT platforms can Solidity be used?

- A. R3 Corda
- B. Hyperledger Fabric
- C. Hedera Hashgraph
- D. Quorum

Q7: Why is REMIX the most widely used IDE for developing the code in Solidity?

- A. No need to install libraries and set up environment
- B. Quick development and testing
- C. Runs on most of the web-based browsers
- D. All of the above

Q8: Why should we save the Solidity files offline even if we can save it on the REMIX browser?

- A. REMIX is not a good option as an IDE
- B. It's a good idea to save a duplicate code offline
- C. REMIX would not work properly when the file grows bigger
- D. The Solidity file on REMIX gets saved to the browser cache, and hence, the deletion of history would lead to the deletion of the Solidity file

Q9: Why should "Auto Compile" be set as On while writing your code?

- A. Auto compile improves the coding standard
- B. Auto compile is mandatory when the file size is bigger than 20 lines
- C. Auto compile makes the developer's life easier by indicating the errors and warnings as soon as we write the code
- D. None of these

Q10: You deployed and run the FirstSample contract in the REMIX browser and found that its retrieveOne function is working perfectly. Then, you added a new function, retrieveTwo to the code and found that the Solidity file is no longer compiling successfully. However, you can still run it. How? Refer to the following code:

```
//SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;
contract FirstSampleContract {
    constructor() {
```

```
function retrieveOne() public pure returns(uint){  
    return 1;  
}  
  
function retrieveTwo() public pure returns(){  
    return 1;  
}  
}
```

- A. It's simply not possible. Smart contract can never run without compiling the file successfully
- B. Because compiling is a different environment from deployment and running in Solidity
- C. You are running the contract from the previous versions of the already deployed contracts, as she has not cleared the "Deployed Contracts" section yet
- D. None of these

Q11: While writing the smart contracts in Solidity because JavaScript VM is the preferred environment over Injected web3 vs Web3 Provider, which of the following is true?

- A. It will run an isolated Ethereum node in the browser, hence very useful to test a contract
- B. It will be 100% compatible with most versions of Solidity, hence preferable.
- C. JavaScript VM comes free of cost while the other two charge real Ethers
- D. JavaScript VM is not the default browser, all of them are equally preferable for development and testing

Q12: When we deploy the following smart contract Solidity file on the REMIX browser on JavaScript VM and run the store function a number of times, what happens to the Ethers associated with the account?

```
//SPDX-License-Identifier: SOME IDENTIFIER  
pragma solidity ^0.8.10;
```

```
contract SumContract {
    constructor() {
    }

    function add() public pure returns(uint){
        uint var1 = 10;
        uint var2 = 20;
        uint sum = var1 + var2;
        return sum;
    }
}
```

- A. The number of Ether remains the same, i.e., 100 Ethers as it's a demo account
- B. The number of Ethers decrease every time
- C. The number of Ethers would decrease only on the first transaction and then would remain the same
- D. The transaction can run maximum 5 times which is a restriction in REMIX

Q13: What happens if the version is not mentioned in the following Solidity file and it's compiled, deployed, and run on the REMIX browser in the JavaScript VM environment?

```
Contract NoVersionSample {
    constructor() {
    }
    function sum() public pure returns(uint){
        uint a = 1;
        uint b = 2;
        uint result = a + b;
        return result;
    }
}
```

- A. The code does not compile
- B. The code compiles but creates problems during deployment

- C. The code compiles, gets deployed, and runs without any issue
- D. The code compiles with warnings as the version being unavailable, but gets deployed and run without issue

Q14: What does ^0.4.0 mean in the following code?

```
pragma solidity ^0.4.0;  
contract VersionCheckSample {  
}
```

- A. The code would compile only with the compiler version 0.4.0
- B. The code might not compile with any compiler version less than 0.4.0
- C. The code might not compile with any compiler version less than 0.4.0 and nor with any version equal to or greater than 0.5.*
- D. None of these

Q15: What happens when inside a Solidity file, the elements are not in the right order?

- A. The code might compile, get deployed, and run with warnings
- B. The code would definitely not compile
- C. The code would compile but definitely not get deployed
- D. The code would get deployed but would definitely break while running

Q16: What is incorrect regarding the interfaces in Solidity?

- A. An interface can declare a public function
- B. An interface can inherit another interface
- C. An interface can declare state variables
- D. All of the above

Q17: What is true regarding the contract/contracts in a Solidity file?

- A. A Solidity file might have just one Contract
- B. A Solidity file might have many contracts

- C. A Solidity file can have libraries
- D. All of the above

Q18: Why are events useful?

- A. They work like functions where we can write business logic
- B. They function like libraries that we can use like utilities
- C. They can log messages that are useful for debugging
- D. They can trigger call back functions in frontend conveying the status of the transactions

Q19: If we write some code in the constructor, when would it be executed?

- A. As soon as the contract is created
- B. When the contract is destroyed to release the resources
- C. Every time and just before any of the function is called
- D. Every time and just after any of the function is called

Q20: Which of the following is correct?

- A. A constructor in the Solidity file is optional
- B. A Solidity file can have many constructors by overloading
- C. A constructor can be private
- D. None of the above

Q21: Which of the return syntaxes are correct regarding a return statement in a Solidity function?

- A. return var;
- B. return (7, true, 2);
- C. A function might have no return statement
- D. All of the above

Q22: What is the issue with the following code?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;
```

```
contract FunctionOverloadingContract {  
    int value;  
  
    constructor(int newValue) {  
        value = newValue;  
    }  
}
```

- A. No issue at all. Constructor can have parameters in Solidity
- B. Would throw a compilation error as only constructors with no parameters are allowed in Solidity
- C. Would cause issue during the run as only the constructors with no parameters are allowed in Solidity
- D. None of these

Q23: What is the issue with the following code?

```
// SPDX-License-Identifier: SOME IDENTIFIER  
pragma solidity ^0.8.10;  
  
contract FunctionOverloadingContract {  
    int value;  
  
    constructor(){  
        value = 1;  
    }  
  
    constructor(int newValue) {  
        value = newValue;  
    }  
}
```

- A. No issue at all. Constructor overloading is allowed in Solidity
- B. Would throw a compilation error as the variable is needed to be declared private
- C. Would cause issue during the run as the variable is needed to be declared private

D. Would throw a compilation error as constructor overloading is not allowed in Solidity

Q24: What is the issue with the following code?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract FunctionOverloadingContract {
    string value;

    function someFunction(string calldata newValue) public {
        value = newValue;
    }

    function someFunction(int someValue) public pure
    returns(int){
        return someValue + 10;
    }
}
```

- A. No issue at all. Function overloading is allowed in Solidity
- B. Would throw a compilation error as the variable is needed to be declared private
- C. Would cause issue during the run as the variable is needed to be declared private
- D. Would throw a compilation error as constructor overloading is not allowed in Solidity

Q25: What is true regarding the state variables?

- A. State variables are stored in a permanent storage associated with the account on which the smart contract runs
- B. The life of the State variables is associated with the life of the contract
- C. The state variables can be accessed from anywhere in the contract
- D. All of the above

Q26: What is/are the difference(s) between a state variable and a local variable?

- A. State variables can be accessed from anywhere in the contract, whereas the scope of the local variables are limited to the function
- B. State variables are stored permanent storage, whereas the local variables are stored in temporary storage
- C. State variables are more expensive than the local variables
- D. All of the above

Q27: Which of the visibility types is not applicable for the state variables that is left blank in the following code?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

// A simple contract with getter and setter functions
contract GetterSetterContract {
    string __ value1 = "Some Value 1";
    string value2 = "Some Value 2";

    function getValues() public view returns(string memory,
string memory) {
        return (value1, value2);
    }

    function setValue(string calldata newValue1, string calldata
newValue2) public {
        value1 = newValue1;
        value2 = newValue2;
    }
}
```

- A. public
- B. internal
- C. private
- D. external

Q28: What is the default visibility of the state variables?

- A. public
- B. internal
- C. private
- D. external

Q29: Out of the following code samples, which one would compile without errors?

- A.

```
contract EnumContract {
    enum Choices{Expensive, Regular, economic}
}
```
- B.

```
contract EnumContract {
    enum Choices{1, 2, 3}
}
```
- C.

```
contract EnumContract {
    enum Choices{true, false, true}
}
```
- D.

```
contract EnumContract {
    enum Choices{Very Expensive, Quite Regular, Almost
    Economic}
}
```

Q30: What would be the issue with the following code when compiled and run with values in setValue set to first time 127 and second time 128?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract MaxValueContract {
    int8 veryBigNumber;

    function getValue() public view returns(int8) {
        return veryBigNumber;
    }

    function setValue(int8 newValue) public {
```

```
veryBigNumber = newValue;  
}  
}
```

- A. The code would not compile at all
- B. The code would compile but throw an error while running in both the cases
- C. The code would compile and run successfully in both cases
- D. The code would work fine for value 127 and throw an error for 128

Q31: While the following is written as the first line of the Solidity code, what do you comprehend?

`pragma solidity >=0.4.20<0.4.25;`

- A. The code can be compiled by any compiler with the version between 0.4.20 and 0.4.25
- B. The code can be compiled by any compiler with the version between 0.4.20 and 0.4.24
- C. The code can be compiled by any compiler with the version greater than 0.4.20 but might not work with 0.4.20
- D. The code can be compiled by any compiler with the version greater than 0.4.20 but might not work with 0.4.25

Q32: You need to implement a very complex mathematical algorithm in your Solidity code. What kind of operator(s) would be mandated to be used?

- A. Arithmetic operator
- B. Assignment operator
- C. Bitwise operator
- D. Both A and B

Q33: Which operator would be used to write a compact code in logical comparisons?

- A. Arithmetic operator
- B. Conditional operator

C. Logical operator

D. Relational operator

Q34: What would be the output of function1 and function2?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract LogicalContract {

    function function1() public pure returns(bool) {
        return true && true || false;
    }

    function function2() public pure returns(bool) {
        return false && true || false;
    }
}
```

A. true, true

B. true, false

C. false, true

D. false, false

Q35: What would be the output of function1 and function2?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract AnotherLogicalContract {
    uint a = 10;
    uint b = 2;

    function function1() public returns(bool) {
        return (++a)*b > a*(b++);
    }

    function function2() public returns(bool) {
        return a*(++b) > (++a)*b;
    }
}
```

```
}
```

- A. true, true
- B. true, false
- C. false, true
- D. false, false

Q36: Hope you remember the BODMAS rules for the arithmetic operations. What would be the output of the following function?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract LogicalContract {
    uint a = 50;
    uint b = 10;
    uint c = 3;
    uint d = 45;
    uint e = 20;

    function checkBODMAS1() public view returns(bool) {
        return a/b*c+d-e == 40;
    }

    function checkBODMAS2() public view returns(bool) {
        return a%b*c+d-e == 40;
    }
}
```

- A. true, true
- B. true, false
- C. false, true
- D. false, false

Q37: Which of the following operations would consume gas?

- A. Deploying the smart contract
- B. Updating the value of state variable

- C. Reading value of a constant in a pure function
- D. Both A and B

Q38: Pick the correct answer from the following options.

- A. The price of Ether is determined by a software program
- B. Always keeps on rising uniformly
- C. Dependent upon the demand and supply of Ether in the market
- D. None of these

Q39: What is true regarding the relationship between Ether and gas?

- A. They do not have any relationship
- B. Gas is a subunit of an Ether
- C. The amount of Gas is used in the calculation of the transaction cost in Ether
- D. None of these

Q40: The requester has set a gasLimit of 30000 and runs a Solidity smart contract. After a while, the transaction is executed; however, the user does not get any Ether back in his account. Why?

- A. Because the transaction might be complex, and all the gas has been consumed
- B. Some Ethers would definitely come back to the user's account, he has to be patient
- C. Because the transaction might have failed
- D. None of these

Q41: Which of the following are used in finding the cost of the transaction?

- A. Gas Limit
- B. Gas Price
- C. Ether converted to gas
- D. Both A and B

Q42: What does the gas usage in a transaction depend on?

- A. The number of state variables
- B. The amount of storage
- C. The complexity of algorithms
- D. All of above

Q43: Which among the following is the default storage for the state variables?

- A. Storage
- B. Memory
- C. Stack
- D. Calldata

Q44: Which among the following is the default storage for the local variables in a function?

- A. Storage
- B. Memory
- C. Stack
- D. Calldata

Q45: What is the cause of error in the following code?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract StorageMemory {
    function saveInMemory() public {
        uint memory varStorage1 = 0;
        uint memory varStorage2 = 0;
    }
}
```

- A. No issues at all. The code will compile, get deployed, and run
- B. The code would run but would consume a lot of gas

- C. The code would not compile as the version does not support memory
- D. The code would not compile as the memory storage is not supported for the integer type of variables

Q46: You have to handle a large amount of data in a function of type Array. Which type of data saving technique would you use?

- A. Storage
- B. Stack
- C. Memory
- D. None

Q47: The variables of type reference can be stored in which of the following?

- A. Storage
- B. Stack
- C. Memory
- D. Both A and C

Q48: Which of the following statement(s) is/are correct?

- A. The default location for storing local variables of struct, array, or mapping type reference is storage
- B. The default location for storing local variables of value type uint, int, string, bool etc., are stack
- C. The default location of storing the state variables is storage
- D. All of the above

Q49: In a particular algorithm in a Solidity file, we need to continue a repetitive calculation till a particular value is reached. After the value is reached, the calculation should not continue. Which control statement would you choose?

- A. If-else
- B. Switch
- C. do-while

D. while-do

Q50: In a particular algorithm in a Solidity file, we need to continue a repetitive calculation till a particular value is reached. After the value is reached, the calculation should continue for one last time. Which control statement would you choose?

- A. If-else
- B. Switch
- C. do-while
- D. while-do

Q51: As Solidity does not support the Switch statement, what else can you do to replace the same behavior?

- A. if-else
- B. if-else if-else
- C. for loop
- D. do-while loop

Q52: What are the advantages of the ternary operator over the if-else statement?

- A. Ternary Operator is more readable than the If conditions
- B. Ternary Operator is suitable for executing a function or several statements
- C. Ternary Operator runs faster than the If-Else statement
- D. All of the above

Q53: Break statements are useful for which of the following?

- A. Control breaking a loop and going ahead with the execution of the next line
- B. Control breaking an If-Else statement and going ahead with the execution of the next line
- C. Control breaking a ternary operator and going ahead with the execution of the next line
- D. None of these

Q54: Try to run the following program with and without the continue statement. What would be the result?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract ForLoopContract {

    function getResult() public pure returns(uint) {
        uint result;

        // continue statement
        for (uint i = 0; i < 10; i++) {
            if (result > 30)
            {
                continue;
            }
            result += 10;
        }
        return result;
    }
}
```

- A. 100, 100
- B. 40, 40
- C. 100, 40
- D. 40, 100

Q55: What is the default value of uint in Solidity?

- A. null
- B. undefined
- C. 0
- D. We have to initialize uint, otherwise it won't compile

Q56: Which among the following is the smallest subunit of Ether?

- A. Wei
- B. Kwei

- C. Mwei
- D. Gwei

Q57: Which of the following statements are true regarding ABI?

- A. ABI is a JSON file
- B. ABI has many different elements as functions, inputs, outputs, constructors etc.
- C. ABI is formed only after the successful deployment of the contract
- D. Both A and B

Q58: What is the benefit of using enum instead of a simple variable?

- A. Enums restrict a variable to have one of only a few predefined values
- B. Enum can have a default choice, so that even if no value is allocated, the default value is already prepopulated
- C. In case of any value outside the listed ones in enum, there would be a runtime exception
- D. All of the above

Q59: How can we optimize the code?

- A. By minimizing the use of the state variables
- B. By using better algorithms to minimize the code complexity
- C. By saving more variables in the storage location
- D. Both A and B

Q60: What would you do if you find a warning in the code?

- A. Ignore it. The code can run even with many warnings.
- B. Configure the REMIX browser to hide the warning and fasten the development
- C. All warnings must be handled before moving for testing, there might be another issue buried beneath it
- D. Warnings lead to faster consumption of gas, hence must be handled carefully

Q61: What happens if the execution of a Smart Contract consumes more than the specified gas?

- A. The contract still gets executed
- B. The contract does not get executed
- C. The user gets a refund
- D. None of the above

Q62: In the Etherscan website, the transaction fee of each transaction is quoted in which of the following?

- A. Ether
- B. Wei
- C. USD
- D. Both A and C

Q63: What is the maximum value that you can set in the following code?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract MaxValueContract {
    uint8 veryBigNumber;

    function getValue() public view returns(uint8) {
        return veryBigNumber;
    }

    function setValue(uint8 newValue) public {
        veryBigNumber = newValue;
    }
}
```

- A. The maximum value for uint8 is 99999999
- B. The maximum value for uint8 is 99999998
- C. The maximum value for uint8 is 255
- D. The maximum value for uint8 is too big to be accommodated in one line

Q64: What issue would the following code cause?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract ValueContract {
    bytes byteValue = "Sample Data";
    string stringValue = "Sample Data";

    function getBytesLength() public view returns (uint){
        return byteValue.length;
    }

    function getStringLength() public view returns (uint){
        return stringValue.length;
    }
}
```

- A. No issue at all. The code would compile, run, and return 11 for both functions
- B. No issue at all. The code would compile, run, and return 11 for getBytesLength and 10 for getStringLength function
- C. The code would compile but would throw an error while running it
- D. The code would not compile at all

Q65: What would be the output of the following contract in the retrieveDefaultValue function?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract EnumSample {
    enum Day
    {
        Monday,
        Tuesday,
        Wednesday,
        Thursday,
    }
}
```

```
    Friday
}

Day constant default_value = Day.Friday;
function retrieveDefaultDate() public pure returns(Day) {
    return default_value;
}
}
```

- A. Thursday
- B. Friday
- C. 4
- D. 5

Q66: What is/are true regarding the following Contract?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract Keccak256Contract {
    function callKeccak256() public pure returns(bytes32 result)
    {
        return keccak256("Sample Contract");
    }
}
```

- A. The output would compute the Ethereum-SHA-3 (Keccak-256) hash of the input value "Sample Contract" string
- B. The output would be "0: bytes32: result 0xe95e5acfb5f6d7b6773cf501d3a978d6be609d54a203215cd195bebe4158ef17"
- C. The output would never change, no matter how many times we deploy and at what time interval we run it
- D. All of the above

Q67: What would be the issue in the following contract?

```
// SPDX-License-Identifier: SOME IDENTIFIER
```

```

pragma solidity ^0.8.10;

contract ArrayContract {
    uint[2] fixedSizeArray;
    uint[] dynamicSizeArray;

    constructor() public {
        fixedSizeArray[0] = 10;
        fixedSizeArray[1] = 20;
        fixedSizeArray[2] = 30;

        dynamicSizeArray[0] = 10;
        dynamicSizeArray[1] = 20;
        dynamicSizeArray[2] = 30;
    }
}

```

- A. No issue at all, the code would run and populate both arrays
- B. The code would throw a compilation error as Solidity does not support dynamic array
- C. The code would throw a compilation error as fixed array of size 2 can't be assigned a third value
- D. The code would compile but would throw a runtime error as fixed array of size 2 can't be assigned a third value

Q68: What are the values that we can set in the following contract in the `setScheduleMeeting` function?

```

// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract MeetingSchedule {
    enum Day
    {
        Monday,
        Tuesday,
        Wednesday,
        Thursday,
        Friday
    }
}

```

```

    }

    struct MeetingDay {
        Day day;
    }

    Day public someday;
    MeetingDay public meetingDay;

    function scheduleMeeting(Day newDay) public returns (Day) {
        meetingDay.day = newDay;
        return meetingDay.day;
    }

    function retrieveNextMeetingScheduled() public view
    returns (Day) {
        return meetingDay.day;
    }
}

```

- A. Monday to Friday
- B. Monday to Sunday
- C. Any number between 0-100
- D. Any number between 0-4

Q69: You are writing a function that involves modification of the state variable. You can mark the function with which of the following?

- A. view
- B. pure
- C. either view or pure would do
- D. None of these

Q70: You are writing a function that returns the value of a global variable and has no use of any state variable. You can mark the function with which of the following?

```

// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

```

```
contract ViewContract {  
    function someFunction() public returns(address) {  
        return msg.sender;  
    }  
}
```

- A. view
- B. pure
- C. either view or pure would do
- D. None of these

Q71: What issue would the following code have?

```
// SPDX-License-Identifier: SOME IDENTIFIER  
pragma solidity ^0.8.10;  
  
contract ViewContract {  
    function someFunction(uint a, uint b) public returns(uint) {  
        return a+b;  
    }  
}
```

- A. No issue at all, the code would compile and run perfectly
- B. The code would throw warnings as it's not marked properly
- C. The code is incomplete as there is no need of initializing the state variables
- D. None of these

Q72: What issue would the following code have?

```
// SPDX-License-Identifier: SOME IDENTIFIER  
pragma solidity ^0.8.10;  
  
contract ViewContract {  
    function someFunction(uint a, uint b) public view  
    returns(uint) {  
        return a+b;  
    }  
}
```



```
}
```

- A. No issue at all, the code would compile and run perfectly
- B. The code would throw warnings as it's not marked properly
- C. The code is incomplete as there is no need of initializing the state variables
- D. None of these

Q73: What issue would the following code have?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract ViewContract {
    uint a = 10;
    uint b = 2;
    function someFunction() public view returns(uint) {
        return a++;
    }
}
```

- A. No issue at all, the code would compile and run perfectly
- B. The code is incomplete as there is no need of initializing the state variables
- C. The code would not compile as there is a transaction in a function marked as view
- D. None of these

Q74: If you need to send a few Ethers to a contract without knowing the name of any function, which function in that contract would be invoked?

- A. Pure Function
- B. View Function
- C. Fallback Function
- D. None of these

Q75: From your contract code, you need to call a function in another contract. Which error handling mechanism would you use?

- A. revert
- B. assert
- C. require
- D. try/catch

Q76: In order to validate the user inputs, which among the following would you use in the Solidity code?

- A. revert
- B. assert
- C. require
- D. try/catch

Q77: For checking the internal errors, for example, the overflow and underflow conditions, which among the following would you use in the Solidity code?

- A. revert
- B. assert
- C. require
- D. try/catch

Q78: If an assert statement fails, what would this denote?

- A. The code has some bug to be resolved
- B. The input to the function has some issue
- C. The response from some external contract has some issue
- D. None of these

Q79: The ABI does not contain which of the following?

- A. Name of functions
- B. Implementation details of functions
- C. Inputs, Outputs, and Return types of functions
- D. Events

Q80: Which of the following types of functions are represented on ABI?

- A. public
- B. private
- C. external
- D. Both A and C

Q81: For the integration with smart contract, the front-end code needs which of the following?

- A. Address on which Solidity Contract is running
- B. ABI
- C. User Id and Password
- D. Both A and B

Q82: Which of the following variables participate in Encapsulation?

- A. State variables
- B. Local variables
- C. Global variable
- D. All of these

Q83: Through which feature does Solidity adhere to the abstraction property of the object-oriented programming?

- A. Encapsulation
- B. Inheritance
- C. Abstraction
- D. Polymorphism

Q84: What is the difference between function overriding and overloading in Solidity?

- A. Function overriding uses keywords as virtual and override, whereas overloading does not
- B. Function overriding is an implementation of Inheritance, whereas function overloading is an implementation of

Polymorphism

- C. Function overriding uses the same input parameters, whereas overloading does not
- D. All of the above

Q85: What would be the issue with the following code?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract Calculator {
    function calculate(uint a, uint b) public pure returns(uint)
    {
        return a*b;
    }

    function calculate(uint a, uint b) public pure returns(uint,
uint) {
        return (a*b, b);
    }
}
```

- A. No issues, the code would compile, get deployed, and run
- B. The code would throw a compilation error as both functions have the same name and same input parameters
- C. The code would throw a runtime error as both functions have the same name and same input parameters
- D. None of these

Q86: What would be the issue with the following code?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract Mammal {
    function mammalFunction() public pure returns(string memory)
    {
```

```

    return "Mammals produce offsprings by directly giving
    birth and females have mammary glands";
}
}

contract SpecialMammal is Mammal {

    function mammalFunction() public pure override
    returns(string memory) {
        return "Humans are special mammals who can speak and walk
        with two legs";
    }
}

```

- A. No issues, the code would compile, get deployed, and run
- B. The code would throw a runtime error as the function in the parent contract is not declared virtual
- C. The code would throw a compilation error as the function in the parent contract is not declared virtual
- D. The code would throw a runtime error for improper use of the override keyword in the function of the child contract

Q87: What would be the issue with the following code?

```

// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract Mammal {
    function mammalFunction() public pure virtual returns(string
    memory) {
        return "Mammals produce offsprings by directly giving
        birth and females have mammary glands";
    }
}

contract SpecialMammal is Mammal {

    function mammalFunction() public pure returns(string memory)
    {

```

```
    return "Humans are special mammals who can speak and walk  
    with two legs";  
}  
}
```

- A. No issues, the code would compile, get deployed, and run
- B. The code would throw a compilation error as the function in the child contract is not declared override
- C. The code would throw a runtime error as the function in the child contract is not declared override
- D. The code would throw a runtime error for improper use of the virtual keyword in the function of the child contract

Q88: In what situation can we use an abstract contract with no virtual function?

- A. We can't have any such abstract contract
- B. When we do not want the contract to get instantiated
- C. We can always do so but it would not be of any use
- D. None of these

Q89: Under what condition are the abstract contracts used?

- A. When we have to implement a group of behavior in contracts with some common functionalities
- B. To promote clean coding
- C. To prohibit code duplication
- D. Both A and C

Q90: When should we use interface instead of abstract contract?

- A. When we are building large, complex, decentralized applications
- B. When we are designing larger scale decentralized applications prior to their comprehensive implementations
- C. When we need to facilitate extensibility in your dapps without introducing added complexity

D. All of the above

Q91: What would be the issue with the following code?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

interface SomeInterface1 {
    function someFunction() external pure returns(string
memory);
}

interface SomeInterface2 {
    function someFunction() external pure returns(string
memory);
}

contract ImplementorContract is SomeInterface1,
SomeInterface2 {
    function someFunction() public override pure returns(string
memory) {
        return "some message";
    }
}
```

- A. No issue at all. It would get compiled and would run to give "some message" in response
- B. It would throw a compilation error as ImplementorContract can't implement two interfaces at a time
- C. It would throw a compilation error as both interfaces have a function of the same name and ImplementorContract can't take a decision on which one to implement
- D. None of these

Q92: What would be the issue with the following code?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;
```

```

interface SomeInterface1 {
    function someFunction(string memory someString) external
    pure returns(string memory);
}

interface SomeInterface2 {
    function someFunction() external pure returns(string
    memory);
}

contract ImplementorContract is SomeInterface1,
SomeInterface2 {
    function someFunction() public override pure returns(string
    memory) {
        return "some message";
    }
}

```

- A. No issue at all. It would get compiled and would run to give "some message" in response
- B. It would throw a compilation error as ImplementorContract can't implement two interfaces at a time
- C. It would throw a compilation error as both interfaces have a function of the same name and ImplementorContract can't take a decision on which one to implement
- D. It would throw a compilation error as ImplementorContract does not implement the function in SomeInterface1

Q93: What would be the issue with the following code?

```

// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

interface SomeInterface1 {
    function someFunction(string memory someString) external
    pure returns(string memory);
}

interface SomeInterface2 {

```



```

function someFunction() external pure returns(string
memory);
}

contract ImplementorContract is SomeInterface1,
SomeInterface2 {
    function someFunction(string memory someString) public
    override pure returns(string memory) {
        return someString;
    }
    function someFunction() public override pure returns(string
memory) {
        return "some message";
    }
}

```

- A. No issue at all. It would get compiled and the second someFunction function would run to give "some message" in response
- B. It would throw a compilation error as ImplementorContract can't implement two interfaces at a time
- C. It would throw a compilation error as both interfaces have a function of the same name and ImplementorContract can't take a decision on which one to implement
- D. It would throw a compilation error as ImplementorContract does not implement SomeInterface1

Q94: What is true for a contract that is destroyed?

- A. Transactions will fail
- B. Any funds sent to the contract will be lost
- C. By default, the contract can be destroyed only by the creator
- D. Both A and B

Q95: There are two contracts named A and B where B instantiates A. Choose the correct answer.

- A. It's a mandate to deploy both A and B before running them

- B. It's a mandate to deploy B before running them
- C. It's a mandate to deploy A before running them
- D. All of the above

Q96: How can you instantiate a contract which has no constructor?

- A. We must write a constructor in a contract in order to get it instantiated
- B. Default constructor would be used
- C. Both A and B
- D. None of these

Q97: Which among the following is the most well documented and most widely used ERC standard?

- A. ERC20
- B. ERC721
- C. ERC165
- D. ERC223

Q98: Fungible tokens are used in which of the following?

- A. Crypto currencies
- B. Auction of gold
- C. Transferring of Ethers from one account to another
- D. Both A and C

Q99: Which of the following functions of ERC20 should we not use?

- A. totalSupply
- B. balanceOf
- C. transfer
- D. allowance

Q100: Why were ERC standards introduced?

- A. To set the guidelines to write efficient smart contracts
- B. So that all contracts would abide by a common standard

- C. So that different types of contracts would have different types of predefined specifications
- D. All of the above

Q101: An interface can't declare which of the following?

- A. Events
- B. Functions
- C. Constants
- D. Structs

Q102: How does Solidity adhere to the encapsulation of a fundamental feature of the object-oriented programming?

- A. By not disclosing any information of smart contract
- B. By making state variables private by default
- C. By advising to use public functions to access the private state variables
- D. Both B and C

Q103: Why do we need to write Events in a Solidity contract?

- A. To integrate with the front-end code
- B. To integrate with the other contracts
- C. Both A and B
- D. None of these

Q104: Which of the following would not lead to the consumption of gas?

- A. Deploying a contract on the Ethereum network
- B. Invoking a function marked with view or pure
- C. Invoking a function calling another function marked with view or pure
- D. All of the above would lead to the consumption of gas

Q105: You have to write a smart contract to sell random items in an auction. Which ERC interface would you implement?

- A. ERC20
- B. ERC721
- C. ERC165
- D. None of the above

Q106: A Solidity contract can be destroyed for which of the following reasons?

- A. By mutual agreement between the parties
- B. By breach of contract
- C. By fraud
- D. All of the above

Q107: What would be the issue in the following code?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract SomeContract {

    function someOtherFunction() public pure returns (uint){
        uint8 y = 10;
        uint16 z = 12;
        uint32 x = y + z;
        return x;
    }
}
```

- A. No issues at all. The code would compile and run successfully
- B. Compilation error as multiple as the types of x, y, and z are different
- C. Code would have issues in getting deployed
- D. None of these

Q108: What would be the issue in the following code?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;
```

```

contract SomeContract {

    function someOtherFunction() public pure returns (uint){
        uint32 y = 10;
        uint16 z = 12;
        uint8 x = y + z;
        return x;
    }
}

```

- A. No issues at all. The code would compile and run successfully
- B. Compilation error as type uint32 is not implicitly convertible to uint8
- C. Code would have issues in getting deployed
- D. None of these

Q109: What would be the issue in the following code?

```

// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

interface SomeInterface {
    function someFunction() external pure returns(string
memory);
}

contract ImplementorContract is SomeInterface {
    function someFunction() public override pure returns(string
memory) {
        return "some message";
    }

    function someOtherFunction() public returns (string memory){
        SomeInterface a = new SomeInterface();
        return "Some other message";
    }
}

```

- A. No issues at all. The code would compile and run successfully

- B. Compilation error as multiple as the interface can't be instantiated in Solidity
- C. Code would have issues in getting deployed
- D. None of these

Q110: What would be the issue in the following code?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

abstract contract AbstractContract1 {
    function someFunction1() public pure virtual returns(string
memory);
}

abstract contract AbstractContract2 {
    function someFunction2() public pure virtual returns(string
memory);
}

contract ImplementorContract is AbstractContract1,
AbstractContract2 {
    function someFunction1() public override pure returns(string
memory) {
        return "some message 1";
    }

    function someFunction2() public override pure returns(string
memory) {
        return "some message 2";
    }
}
```

- A. No issues at all. The code would compile and run successfully
- B. Compilation error as multiple inheritance is not supported in Solidity
- C. Code would have issues in getting deployed
- D. None of these

Q111: What would be the issue with the following code?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract ServerContract {

    function callMe() public pure returns(uint) {
        return 5;
    }
}

contract ClientContract {

    function invokeCallMe() public returns(bool) {
        ServerContract a = new ServerContract();
        ServerContract b = new ServerContract();
        return a == b;
    }
}
```

- A. No issues at all. The code would compile and run.
- B. Compilation error as there is an issue with the way we compare
- C. Code can't be deployed as the first contract can't be instantiated twice
- D. None of these

Q112: After running the destroyContract function, if you run the calculate function, what would be the output?

```
// SPDX-License-Identifier: SOME IDENTIFIER
pragma solidity ^0.8.10;

contract SelfDestructContract {
    address payable owner;

    constructor() {
        owner = msg.sender;
    }
}
```

```
function calculate(uint a, uint b) public pure returns(uint)
{
    return a*b;
}

function destroyContract() public {
    if (msg.sender == owner) {
        selfdestruct(owner);
    }
}
}
```

- A. No issues at all. The code would run and give the desired output.
- B. The code would run but always would give 0 as output
- C. The code would throw an error as the contract is destroyed already
- D. None of these

Q113: Which of the following use cases would work best on Ethereum?

- A. e-Voting
- B. P2P Lending
- C. e-Auction
- D. All of the above

Q114: Why is Ethereum the default choice for Decentralized finance?

- A. High Scalability
- B. High Performance
- C. No intermediaries and cost effective
- D. Low learning curve

Q115: What is the main issue with Ethereum to build an e-Voting DApps with PoW consensus?

- A. Scalability
- B. Performance
- C. No cost of transaction
- D. All of the above

Q116: Which functional or non-functional requirements can Ethereum not adhere to?

- A. Performance
- B. Cost effective transaction
- C. Accuracy of transaction
- D. Peer to peer of transaction

Q117: What is the chief advantage of the PoS model of Ethereum?

- A. Scalability
- B. Performance
- C. No cost of transaction
- D. All of the above

Answers

1. D 2. D 3. C 4. D 5. D 6. C 7. D 8. D 9. C 10. C 11. A 12. B 13. D
 14. C 15. A 16. D 17. D 18. D 19. A 20. A 21. D 22. A 23. D 24. A
 25. D 26. D 27. D 28. C 29. A 30. D 31. B 32. D 33. B 34. B 35. A
 36. B 37. D 38. C 39. C 40. A 41. D 42. D 43. A 44. C 45. D 46. C
 47. D 48. D 49. D 50. C 51. B 52. A 53. A 54. C 55. C 56. A 57. D
 58. D 59. D 60. C 61. B 62. D 63. C 64. D 65. C 66. D 67. C 68. D
 69. D 70. A 71. B 72. B 73. C 74. C 75. D 76. C 77. B 78. A 79. B
 80. D 81. D 82. A 83. C 84. D 85. B 86. C 87. B 88. B 89. D 90. D
 91. C 92. D 93. A 94. D 95. B 96. B 97. A 98. D 99. C 100. D 101.
 C 102. D 103. C 104. B 105. B 106. D 107. A 108. B 109. B 110.
 C 111. A 112. A 113. D 114. C 115. D 116. A 117. C

CHAPTER 3

Hedera Hashgraph

“PoW is certainly not Asynchronous Byzantine, as ABFT means that it will work even if the Internet is screwed up in various ways. Even if an attacker can control the Internet itself, as long as you let enough messages through, the tree to make any progress at all—the hashgraph’s Gossip protocol will still work.” - Leemon Baird

Founded: 2019

Native token: HBAR

Market Capitalization: \$3.29B

Smart Contract languages: Solidity

In this chapter, we will explore the 2019 launched Hedera Hashgraph that is built on a Direct Acyclic Graph platform rather than the Blockchain and yet has surfaced as a prime competitor of Ethereum for its stability and high scalability.

In April 2021, Hedera crossed 1 billion transactions on its mainnet. While Ethereum took many years for the same and Bitcoin has not achieved it yet, Hedera achieved this milestone in just a little over six months. Needless to mention, today, Hedera Hashgraph is one of the most used, enterprise-grade public DLT network for the decentralised economy.

3.1 Consensus Model – Proof of Stake

Hedera’s data structure and consensus algorithm is called Hashgraph, which is ideal for a distributed environment. It’s underlying asynchronous Byzantine Fault Tolerant (aBFT) algorithm makes sure that no single node (or small group of nodes) can prevent the network from reaching a consensus.

Hashgraph works on a Proof of Stake consensus model where the transactions are validated by a group of close to 40 members

together known as the “Governing Council”. The council comprises of the world’s leading organizations as well as the leaders from many different business verticals. Some of these organizations are Avery Dennison, Boeing, Chainlink Labs, Dentons, Deutsche Telekom, DLA Piper, EDF (Électricité de France), eftpos, FIS (WorldPay), Google, IBM, the Indian Institute of Technology (IIT), LG Electronics, the London School of Economics and Political Science (LSE), Magalu, Nomura Holdings, Shinhan Bank, Standard Bank Group, Swirlds, Tata Communications, University College London (UCL), Wipro, and Zain Group.

Unlike Bitcoin and Ethereum, Hedera Hashgraph is not composed of blocks, and therefore it’s not a Blockchain. Rather, Hedera Hashgraph is a special graph-like structure that uses the events to share the information with each other.

Hedera uses two different concepts, i.e. “gossip about gossip” and “virtual voting” to create a similar level of transparency and immutability features as the Blockchain has with a far greater efficiency, scalability, and throughput, as shown in [Figure 3.1](#) as follows:

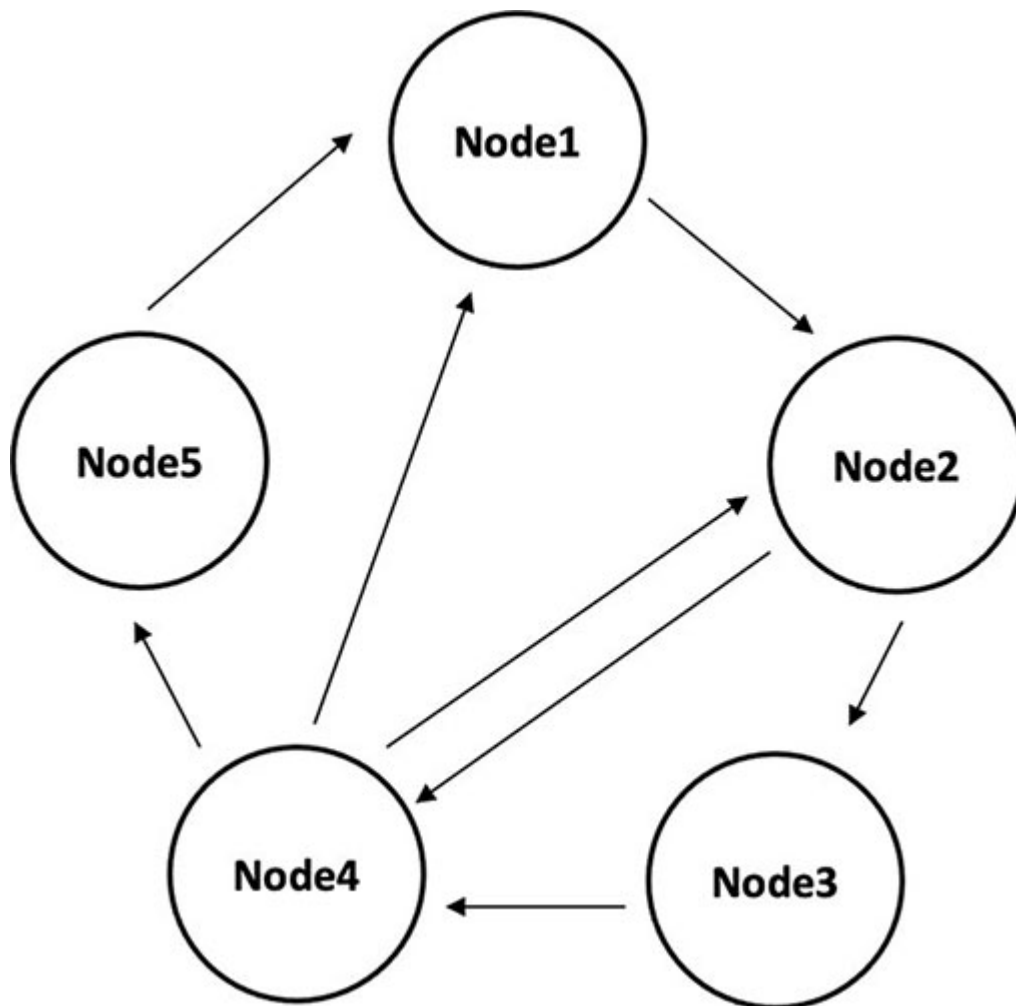


Figure 3.1: Hedera Hashgraph's Gossip Protocol

Gossip about gossip: Information is transmitted from node to node through a gossip network. When a node receives the transactions, it transfers or gossips this information to the next node which is called an “Event”. While doing so, it also shares the identifiers of the previous event as it received and also the previous event it sent. In [Figure 3.2](#), we can find a sample on what an event looks like, i.e., signature, timestamp, current transaction, and two hashes of the other two transactions.

Virtual voting: As each participating node receives the events from the others, they can independently analyse that shared history in order to determine a consensus timestamp for all the transactions. This consensus mechanism that Hedera's participants use is called virtual voting.

Refer to [Figure 3.2](#) as follows:

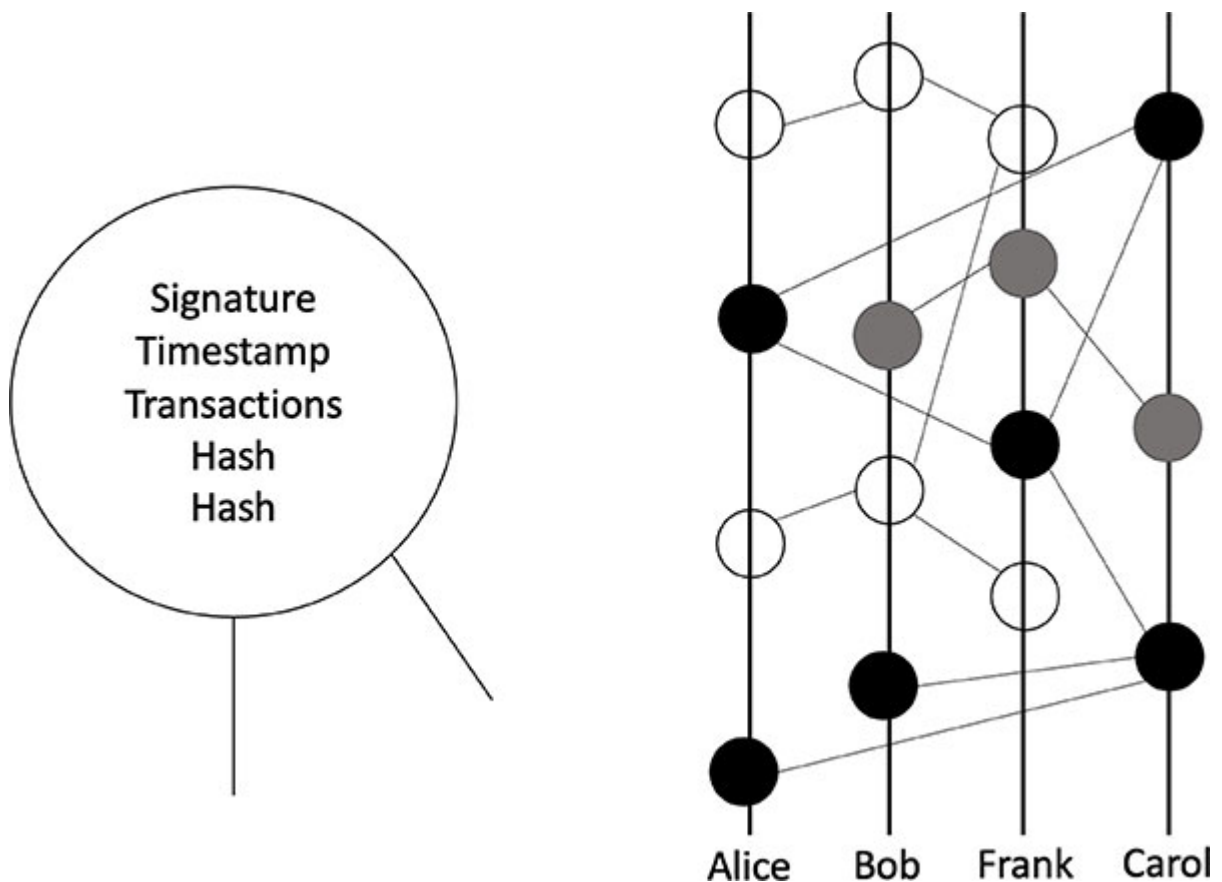


Figure 3.2: Hedera Hashgraph Gossip about Gossip

This new form of consensus model eventually became so popular that many later public Blockchain platforms used it in one way or the other, and also with more variations. Even the CEO of Hedera Hashgraph accused the Facebook Libra currency to copy some of the ideas of Hashgraph for consensus.

3.2 Transaction fees

Just like Ethereum, Hedera too has its native, energy-efficient token called HBAR. The average fees of the transactions on Hedera Hashgraph are around \$0.0001, however a detailed chart can be found on the following Hedera's website: <https://docs.hedera.com/guides/mainnet/fees#transaction-and-query-fees>

We all know that, in public Blockchains like Bitcoin, the price of the native tokens fluctuates as per demand and supply. However, the transaction fees in Hashgraph are relatively stable. Why?

The transaction fees in Hedera Hashgraph are relatively stable as its stringently regulated by its governing council. Also, the fees are set in fiat USD but paid in HBAR.

3.3 L2 Solutions

Hedera Hashgraph is already fast and scalable, and hence, does not need an L2 solution. However, the framework supports it. In May 2021, Hedera tweeted to announce AllianceBlock's L2 validator network 'AllianceBridge' that enables cross-chain interoperability using Hedera Consensus Service & Scheduled Transactions whose case study is available on the following website:

<https://hedera.com/users/allianceblock>

3.4 Scalability and Performance

Hedera Hashgraph has been tested till 10,000 TPS and 6.5 million transactions per day with a finality of 3-5 seconds.

3.5 Development

Hedera's smart contracts can be written in Solidity, hence it would be easy, especially for the Ethereum Solidity developers to cross-train themselves in Hedera Hashgraph. Hedera SDK is used to write a layer in order to interact with the smart contract. Hedera SDK is available in Go, Java, and JavaScript for easy integration. Developers can start from the website <https://docs.hedera.com/guides/docs/sdks> that has loads of examples to start with.

3.6 Wallet

Hedera Hashgraph has the support from many different types of wallets, for example, D'CENT which is a hardware wallet, Guarda,

that can run on Desktop, Mobile Browser Extension or Web wallet, Atomic and Exodus that has support to be run on Desktop and mobile etc.

3.7 Live Projects

Now, let's explore some of the most popular projects on Hedera Hashgraph.

EarthId: Award winning Decentralized identity platform, EarthId is using Hedera Hashgraph as the underlying DLT platform to build trust against the user's credentials.

CuLedger: Credit union-owned CUSO (credit union service organization) CuLedger is using Hedera Hashgraph to build a cross border payments platform.

Mingo: A multi-channel messenger for the exchange of messages on well-known social media platforms as Facebook, Twitter, Steam, Slack, Skype, IRC Cloud, and Discord.

Carbon: Algorithm based Stablecoin that is fast, scalable, secure, and closely correlates with the US Dollar.

Red Swan: Secure real estate tokenization platform that performs large scale and high value transactions.

Artbit: Micropayments platform for artists.

3.8 Summary

In this chapter, we covered the following topics:

- Innovative concepts of Hedera Hashgraph, i.e., Direct Acyclic Graph, gossip about gossip, and virtual voting.
- High scalability and relatively stable price of Hedera tokens.
- Award winning projects on Hedera Hashgraph.

References

- Hello Future, Hedera - <https://hedera.com/>

- Libra: Did Facebook Libra copy Hedera Hashgraph ideas? - <https://en.cryptonomist.ch/2019/06/24/libra-facebook-hedera-hashgraph-2/>
- Hedera on Twitter - <https://twitter.com/hedera/status/1398013943734366209>

CHAPTER 4

Tezos

Founded: June 2018

Native token: XTZ

Market Capitalization: \$5.11B

Smart Contract languages: Python

In this chapter, we will cover the Tezos Blockchain that is very popular among the Blockchain developers for its unique liquid model. Proposed in 2014 through a whitepaper and launched in 2018, Tezos is another Proof of Stake based Blockchain. However, what makes it different from the others is its “self-amending Blockchain protocol”. Let’s explore at length.

4.1 Consensus Model – Liquid Proof of Stake

Tezos uses a different version of the Proof of Stake consensus called “Liquid Proof of Stake”, where the validators validate the transaction after putting their stakes on the network. As already discussed, the Tezos Blockchain does not need to fork when the platform has to be upgraded. Rather, the platform would allow all the owners of its native cryptocurrency XTZ to vote on the possible changes to its rules. Once voted and agreed, the software would automatically update to the new version. The voting process which is known as “Baking” needs the users to stake their XTZ tokens for being the participants of the process. The code that updates itself based on voting is called “**Shell**”.

4.2 Transaction fees

The transaction fees on Tezos is dependent on the gas consumption as well as the storage usage.

4.3 L2 Solutions

Tezos is pretty slow in comparison to many 3rd generation public Blockchains, and hence it is in a process of trying its hands on the L2 technologies for improvement. Recently, the Cornwall University submitted a paper for using sharding technologies for improving scalability; similarly another organization called Marigold too is working on Plasma technologies for improving the Tezos throughput, scalability, and finality.

4.4 Scalability and Performance

At 40 transactions per second, Tezos might seem better than Bitcoin or Ethereum; however, it has much space to improve.

4.5 Development

The Tezos protocol was originally coded with OCaml which is a 1996 born object oriented language. Tezos support to write its smart contract using Python.

Tezos development portal <https://tezos.com/developer-portal/> has good documentation for smart contracts, access to Oracle services, sandbox environment, and deployment of public node on AWS.

4.6 Live Projects

There are more than 100 projects implemented on Tezos which are mostly on decentralized finance and DApps. The following are a few examples:

Societe General: As per news in September 2020, France's central bank Banque de France

Societe Generale – FORGE has selected the Tezos Blockchain for Central Bank Digital Currency (CBDC) experiments.

StableTZ: Platform for Stablecoins on Tezos.

Kalamint: Non fungible tokens on Tezos

4.7 Summary

In this chapter, we covered the following topics:

- The Liquid Proof of Stake consensus model of Tezos, the self-amending Blockchain.
- Improvement area for Tezos and live projects

References

- Tezos selected by Societe General – FORGE for its central banking digital currency experiment - <https://www.sgforge.com/tezos-selected-by-societe-generale-forge-for-its-central-banking-digital-currency-experiment/>
- Tezos projects - <https://tezosprojects.com/>

CHAPTER 5

Cardano

“It [Cardano] actually does the things that we’ve always wanted to do in cryptocurrencies, which is build a financial operating system for people who don’t have one, one that actually can compete with a global financial system.” - Charles Hoskinson

Founded: 2017

Native token: ADA

Market Capitalization: \$71.65B

Smart Contract languages: Plutus, Marlowe, or Rosetta but internally uses Haskell

Launched in 2017, Cardano has consistently occupied the top 5-6 positions in the Coin Marketcap website <https://coinmarketcap.com/> in terms of market capitalization. In this chapter, we will cover Cardano and explore how it came up with a Proof of Stake model that was secure and highly scalable, ending up being the top competitor to Ethereum in the crypto market.

Cardano was founded in 2015 by Charles Hoskinson who had earlier co-founded Ethereum. The name of the parent company is Input Output Hong Kong or simply IOHK, a Blockchain-engineering company, whose primary business is the development of Cardano, along with the Cardano Foundation. The platform is named after the Italian polymath Gerolamo Cardano, while the cryptocurrency itself is named after the English mathematician and the first computer programmer Ada Lovelace.

5.1 Consensus Model – Proof of Stake

So, what made Cardano so popular among the crypto lovers?

Well, first of all, Cardano is one of the first public Blockchains to implement and go live with the Proof of Stake consensus

successfully, whereas Ethereum is still struggling to do so till now.

Cardano's Proof of Stake protocol is called "Ouroboros" that relies on the stake pools for processing the transactions and blocks and saving them to the ledger. Cardano consists of many such stake pools or reliable server nodes that are responsible for the maintenance of the combined stake of many different stakeholders in a single entity. Cardano has been able to generate huge amount of trust in the market by maintaining over 2,000 distributed stake pools handled by the community. Hence, it's a huge success for Cardano where all its transactions in Blockchains are validated by the users without any dependency on a centralized authority. Of course, the maintenance of Ouroboros would involve some cost, but that is covered by the transaction fees deposited by the users. Though initially introduced only as a public Blockchain, Cardano got the smart contract support in 2021.

5.2 Transaction fees

Cardano's transaction fees are dependent on two constants, which are protocol parameters and the size of the transaction in bytes. The values of the protocol parameters can be adjusted in order to adapt to the changes in the transaction volume, hardware prices, and the underlying crypto's valuation. Cardano's inherent crypto is called ADA. The current transaction fees on Cardano is close to 0.16 ADA per transaction, which is \$0.4 USD.

5.3 L2 Solutions

Have you heard of Hydra, the serpentine water monster in Greek and Roman mythology that has many heads? If any of the monster's head was chopped, a new head would regenerate.

The second layer solution on top of the Cardano is called "Hydra", which is an extended UTxO model that allows sharding of the stake space without the need to shard ledger itself. Every pool can create a new Hydra's head and each head can process around 1000 TPS.

Hydra also comes with low latency and minimal storage features as well as the support for smart contracts, using which, DApps can be

developed. Hydra follows the “State Channel” L2 architecture for achieving this scalability, as shown in [Figure 5.1](#), as follows:

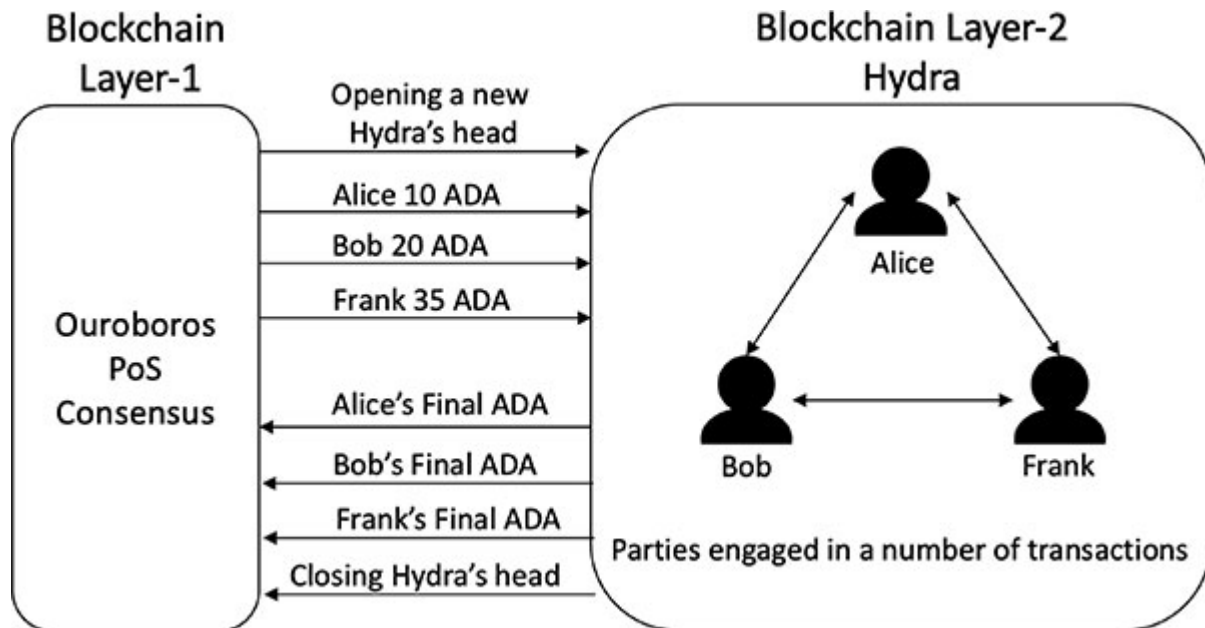


Figure 5.1: Cardano's L2 solution with “Hydra”

Now, let's say there are a group of participants who wish to do some business transactions among each other. Even if they join the ecosystem in the first layer, the individual transactions are sent to a Hydra head that can complete the verifications, and finally send a cumulative transaction to the first layer or the main ledger to get updated.

5.4 Scalability and Performance

As per Cardano, "Each Hydra head can process around 1000 TPS and there is room for further optimization. So, with 1000 pools, Cardano could theoretically be able to scale up to 1 million TPS and the finality of the transactions will be very fast. Hydra enables horizontal scaling."

Ouroboros Hydra breaks new ground in PoS scalability. With Hydra, Cardano can really become the alternative to the current fiat money.

5.5 Development

The Cardano developers need to learn three different languages in order to write smart contracts on Cardano, which are as follows:

- Plutus
- Rosetta
- Marlowe

However, internally, Cardano is written in Haskell.

“Haskell is a general-purpose, statically typed, purely functional programming language with type inference and lazy evaluation” as per Wiki.

Cardano functions can be written independently and tested in complete isolation. Hence, Haskell enables Cardano to write powerful methods and better testability. Now, let's explore the three languages of Cardano.

5.5.1 Plutus

It's the Turing complete smart contract language for Cardano, originally written in Haskell. Hence, again, it's a purely functional programming language with full-stack programming environment.

5.5.2 Marlowe

Marlowe, which is a domain specific language (DSL), was introduced later to Cardano and it was specially crafted to handle writing and executing financial contracts conveniently. The learning curve in Marlowe is quite low, and hence, it's highly recommended for building Decentralized Finance (DeFi) applications. Marlowe also comes with a playground or a sandbox environment using which the developer can easily develop, simulate, and test smart contracts.

5.5.3 Rosetta

Rosetta provides a set of tools to integrate the other external layers with Blockchains.

5.6 Wallet

Cardano comes with quite a few official wallets, even though there are wallet options available outside as well for its huge popularity in the crypto world. Some of the well-known wallets are as follows:

Daedalus: It's a desktop wallet with which the user downloads a complete copy of the Cardano Blockchain and can get involved in validating the transactions. Being a full node wallet, Daedalus provides maximum security without the need of any centrally-hosted third party servers.

Yoroi: It's a hardware wallet that again works with a full node.

5.7 Live Projects

The following are some of the well-known projects on Cardano:

- **Sundaeswap:** A decentralized exchange that allows the users to exchange their native tokens paying a small fee
- **Ardana:** An asset-backed Stablecoin based decentralized exchange
- **MELD:** A decentralized and trust-less lending protocol governed by the MELD token.
- **Drunken Dragon Games:** NFT game
- **Empowa:** The first 'RealFi' platform in the world to provide affordable housing on the African continent.
- **THEOS:** Instant liquidity protocol for non-fungible tokens
- **Cardax:** Decentralized exchange for participants to exchange their ADA for any Cardano Native Token.

5.8 Summary

In this chapter, we covered the following topics:

- Reasons for Cardano to occupy the position of top 5 crypto consistently.
- Hydra, Cardano's innovative L2 technology capable of offering linear scalability.
- Different development options on the Cardano platform.

References

- Get Started, Cardano Development Portal -
<https://developers.cardano.org/docs/get-started/>
- A beginner's guide to Cardano -
<https://levelup.gitconnected.com/a-beginners-guide-to-cardano-a3fbbb3be243>
- Cardano Smart Contracts Explained: What Are Smart Contracts? -
<https://www.gfinityesports.com/cryptocurrency/cardano-smart-contracts-explained-meaning-language-examples-ada/>
- Top 5 projects on the Cardano ecosystem –
<https://azcoinnews.com/top-5-projects-on-cardano-ecosystem-you-should-watch-out-according-to-coin-bureau.html>
- Top 5 Cardano Projects After Alonzo Upgrade (Part 2) –
<https://www.altcoinbuzz.io/bitcoin-and-crypto-guide/top-5-cardano-projects-after-alonzo-upgrade-part-2/>
- Hydra: Cardano scalability solution –
<https://cardanojournal.com/hydra-cardano-scalability-solution-46>

CHAPTER 6

Algorand

“Algorand developed a public Blockchain that runs on a version of Proof of Stake, which drives electricity consumption to almost zero. I care about the planet.” – Silvio Micali

Founded: June 2019

Native token: ALGO

Market Capitalization: \$9.7B

Smart Contract languages: TEAL

In this chapter, we will cover Algorand, one of the most widely used Blockchains today in the financial domain for its highly innovative concepts in cryptography.

As per a news reported by CNBC in September 2021, the cryptocurrency of Algorand has steadily progressed to 500% in just last 12 months, whereas Yahoo reported it to have rallied 600% this year. Algorand that hit the road in mid 2019, had become extremely popular not only among the crypto lovers and DApps developers, but also among the Governments to build Stablecoins as USDC and Tether. One main reason perhaps is its co-founder Silvio Micali who is a respectable name in the computer science industry. Earlier, the same Silvio had co-invented many other algorithms and tools in cryptography such as probabilistic encryption, Zero-Knowledge Proofs, Verifiable Random Functions etc.

6.1 Consensus Model – Pure Proof of Stake

Algorand uses a special version of the Proof of Stake consensus which is known as “Pure Proof of Stake” model. It is a public permissionless Blockchain network that allows every user to participate in the consensus process by just having some ALGOs in their account (starting from 1 ALGO). At any point of time, the

proposers of the blocks are selected at random based on their online stakes. This means, the more the number of ALGOs in the participant's account, the more is the probability for being selected as a proposer.

Algorand is the first public Blockchain protocol that uses the "Pure Proof of Stake" consensus for which it is far more secure, reliable, and un-forkable than its original PoS version. Algorand has two different types of nodes called Relay nodes and Participating nodes. While the Relay nodes work as the network hubs relaying the protocol messages, the participating nodes take the responsibility of block creation and validation. Algorand's PPoS consensus works in three stages, i.e., block proposal, soft vote, and certify vote.

6.1.1 Block Proposal

In the proposal phase, all the nodes check whether the ALGO is available in the accounts and, just like lottery, randomly select the proposers, as shown in [*Figure 6.1*](#), as follows:

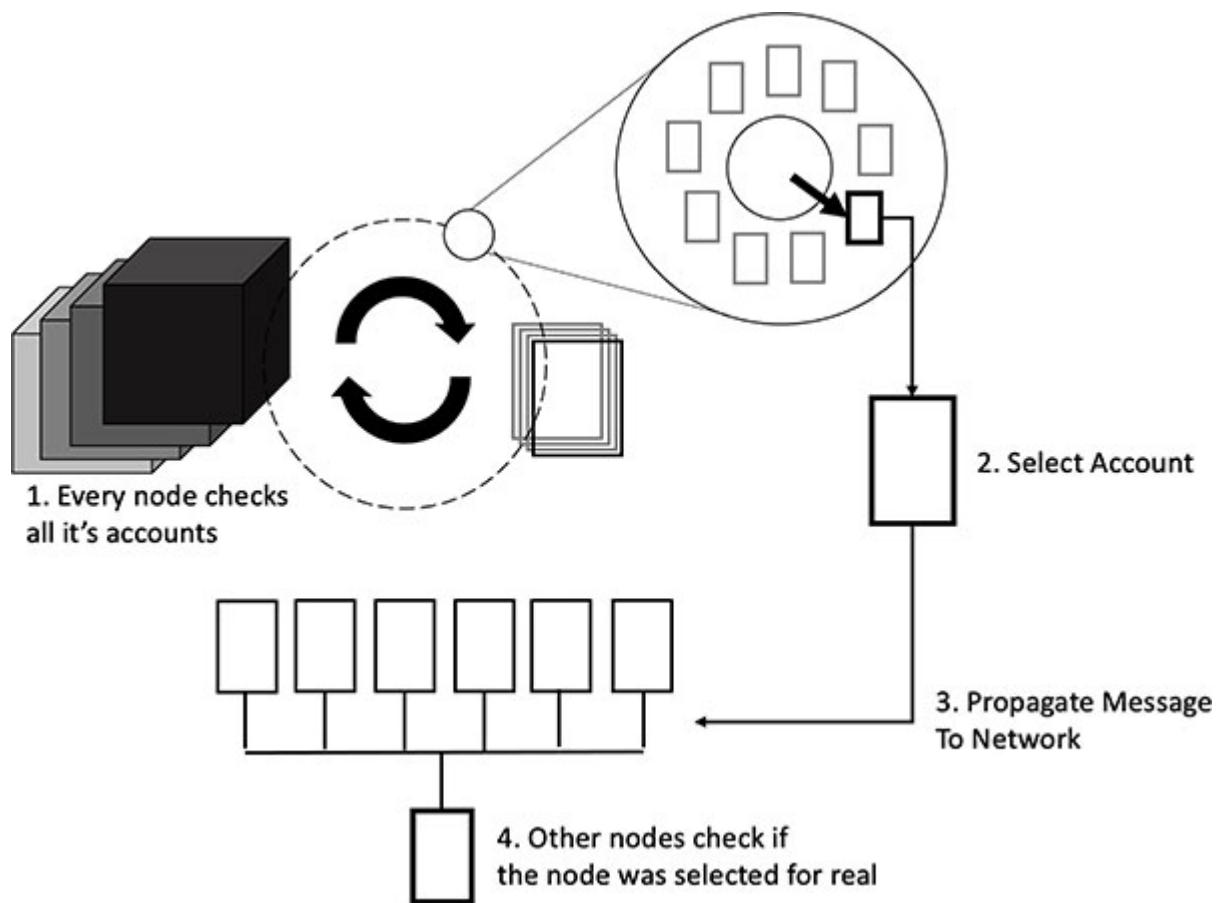


Figure 6.1: Block Proposal in Consensus process of Algorand

Only the selected accounts can propose new blocks to the network.

6.1.2 Soft Vote

Verifiable random functions (VRFs) invented by Silvio Micali along with two more scientists Michael Rabin and Salil Vadhan, are cryptographic functions that can create verifiable pseudonymous outputs on the basis of inputs and a secret key. These outputs, created using VRFs along with its public key proofs, can be verified without having access to the private key. After the block proposal, the nodes can execute VRF on the participating accounts to figure out if the accounts are selected for the voting process.

6.1.3 Certify Vote

After the voting, the other checks such as double spending, overspending etc., are done through a committee for certification.

6.2 Transaction fees

The transaction fees at Algorand Blockchain are currently set at 0.001 Algo per transaction. Currently, the USD to ALGO ratio is close to 1:1.5 which makes the fee quite low. However, in the future, if the price of the ALGO goes higher, just like Bitcoin, then it could be a threat to the business transactions.

6.3 L2 Solutions

Algorand's L1 is fast and capable enough to execute a high number of transactions with efficiency. Algorand supports the L2 frameworks wherever there is a need of implementing much larger and complex smart contracts that need more customization.

6.4 Scalability and Performance

Algorand is already fast and scalable; however, as per the co-founder Silvio, Algorand's 2021 performance would improve significantly. While the block finalization time would be between 2.5-4.5 seconds, the throughput can rise from 1000 to 46000 TPS depending upon the configuration.

6.5 Development

Just like any traditional web server, Algorand supports two different types of smart contracts, i.e., stateless for managing authentication, authorization etc., and stateful smart contracts for handling transactions. The smart contracts can be written using a bytecode-based stack language called Transaction Execution Approval Language (TEAL). Algorand's development portal <https://developer.algorand.org/> is self-explanatory and well documented.

6.6 Wallet

Algorand has the support from many different wallet providers; for example, Ledger hardware wallet, MyAlgo web wallet, Algorand core's mobile wallet, desktop and mobile wallets from Atomic and Coinomi, Guarda Wallet that works on web, desktop, mobile and chrome extension, Trust mobile wallet from Binance etc.

6.7 Live Projects

The following are some of the important projects that are already live on Algorand:

Stablecoins: First and foremost, the two most popular Stablecoins, Tether and USD Coin (USDC), were launched on the Algorand network.

Yielding: The Cross-chain platform between Ethereum and Algorand using which the users can exchange value.

Algobot: One of the first intelligent chatbots to handle the network security policy management.

Opulous: Algorand based NFT for handling the copyrights in the music industry. It can also be used to provide loans to artists guaranteed against the artist's past streaming revenues with the copyrights they own.

Società Italiana degli Autori ed Editori (SIAE): Italian major copyright collecting agency SIAE founded in 1882 is using Algorand for a similar cause.

Republic: Profit sharing NFT for the facilitation of crowdfunding campaigns for start-ups and small-to-medium businesses (SMBs).

Planetwatch: A spin-off of CERN called Planetwatch is working on a global network of air quality sensors, where the sensors record measurement data onto the Algorand Blockchain. CERN is a European Organization for Nuclear Research and one of the largest in the world.

6.8 Summary

In this chapter, we covered the following topics:

- Pure Proof of Stake consensus model, Algorand's differentiator.
- High adoption of Algorand for the development of DeFi.
- Live projects including the most popular Stablecoins.

References

- Ethereum fees are skyrocketing — But traders have alternatives - <https://cointelegraph.com/news/ethereum-fees-are-skyrocketing-but-traders-have-alternatives>
- Why You Should Issue NFTs on a Trustless, Permissionless Blockchain with Layer-1 Capabilities - <https://www.algorand.com/resources/blog/issue-nfts-on-trustless-permissionless-blockchain>
- Algorand 2021 Performance - <https://www.algorand.com/resources/algorand-announcements/algorand-2021-performance>
- Algorand 2021 Performance - <https://www.algorand.com/resources/algorand-announcements/algorand-2021-performance>
- Best Algorand Wallets: 5 Secure ALGO Staking Options - <https://www.coinbureau.com/analysis/algorand-wallets-algo/>

CHAPTER 7

Solana

“Literally the goal of Solana is to carry transactions as fast as news travels around the world — so speed of light through fiber.” - Anatoly Yakovenko, Founder & CEO Solana

Launched: April 2020

Native token: SOL

Market Capitalization: \$41.33B

Smart Contract languages: Rust, C, C++

In this chapter, we will cover the Solana Blockchain, the hottest name in the public Blockchain market today.

Solana published a whitepaper in 2017 with many unique and innovative concepts for a new public Blockchain. In September 2021, Solana claimed to be the fastest Blockchain in the world, successfully handling 50,000 transactions per second and 400 ms block time latency with over 200 nodes on the current Testnet iterations. This achievement makes Solana the world's first web-scale decentralized network. In order to reach this massive scalability, Solana has discovered eight different innovative technologies, which are as follows:

- **Proof of history** consensus to be discussed next
- **Tower BFT** algorithm optimized to work with the PoH consensus
- **Turbine** for faster message propagation achieved by breaking down the data to smaller units.
- **Gulf-Stream** protocol for caching and forwarding the transactions quickly reducing the confirmation time
- **Sealevel** helps in running thousands of smart contracts in parallel

- **Pipeline** transaction processing unit that helps the transaction execution faster by reducing the load on hardware
- **Cloudbreak** protocol for reading and writing data across the network concurrently.
- **Achievers** or data storage for DLT

7.1 Consensus Model – Proof of History & Proof of Stake

As shown in [Figure 7.1](#), the Solana network is run by a group of validator nodes who have similar hardware configuration. At any point in time, one of these nodes is selected as a leader node through “Proof of Stake” based election mechanism. Refer to [Figure 7.1](#) as follows:

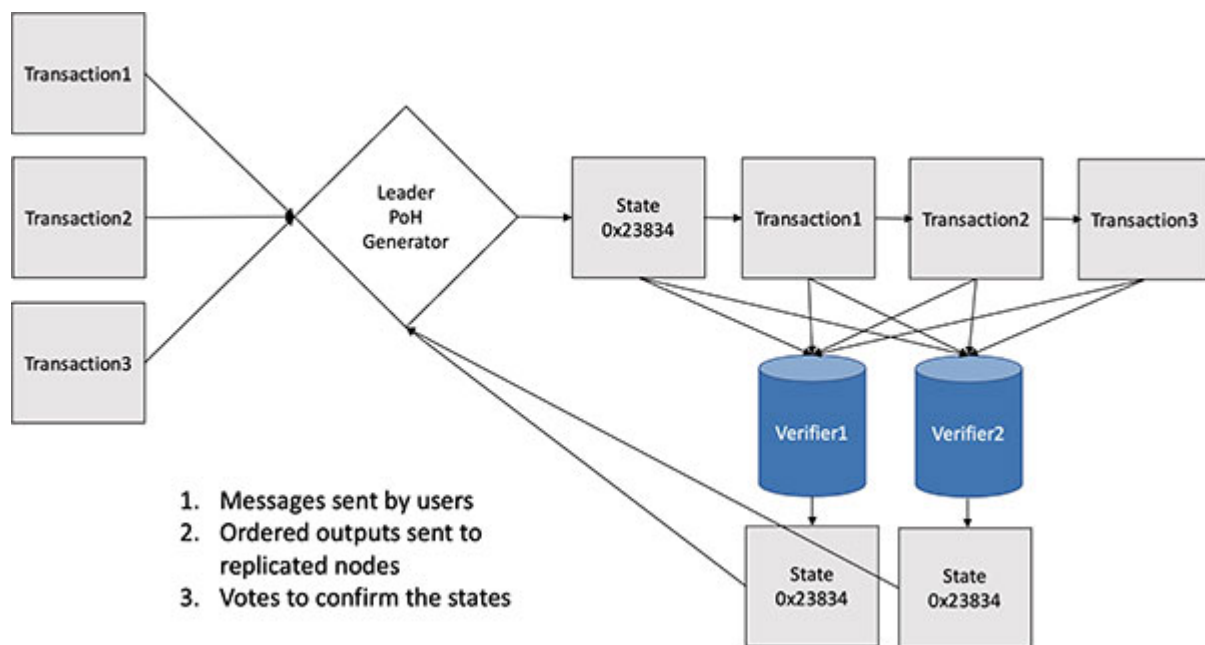


Figure 7.1: Transaction Flow through the network

The validation part in Solana works on a new consensus model called “Proof of History” where different leader nodes are selected from time to time. All users send their transactions to the leader node that orders the transactions and keeps on broadcasting it to the other validator nodes from time to time. The validator nodes then

independently verify and vote to confirm the state. Refer to [Figure 7.2](#) that illustrates the verification through Solana nodes, as follows:

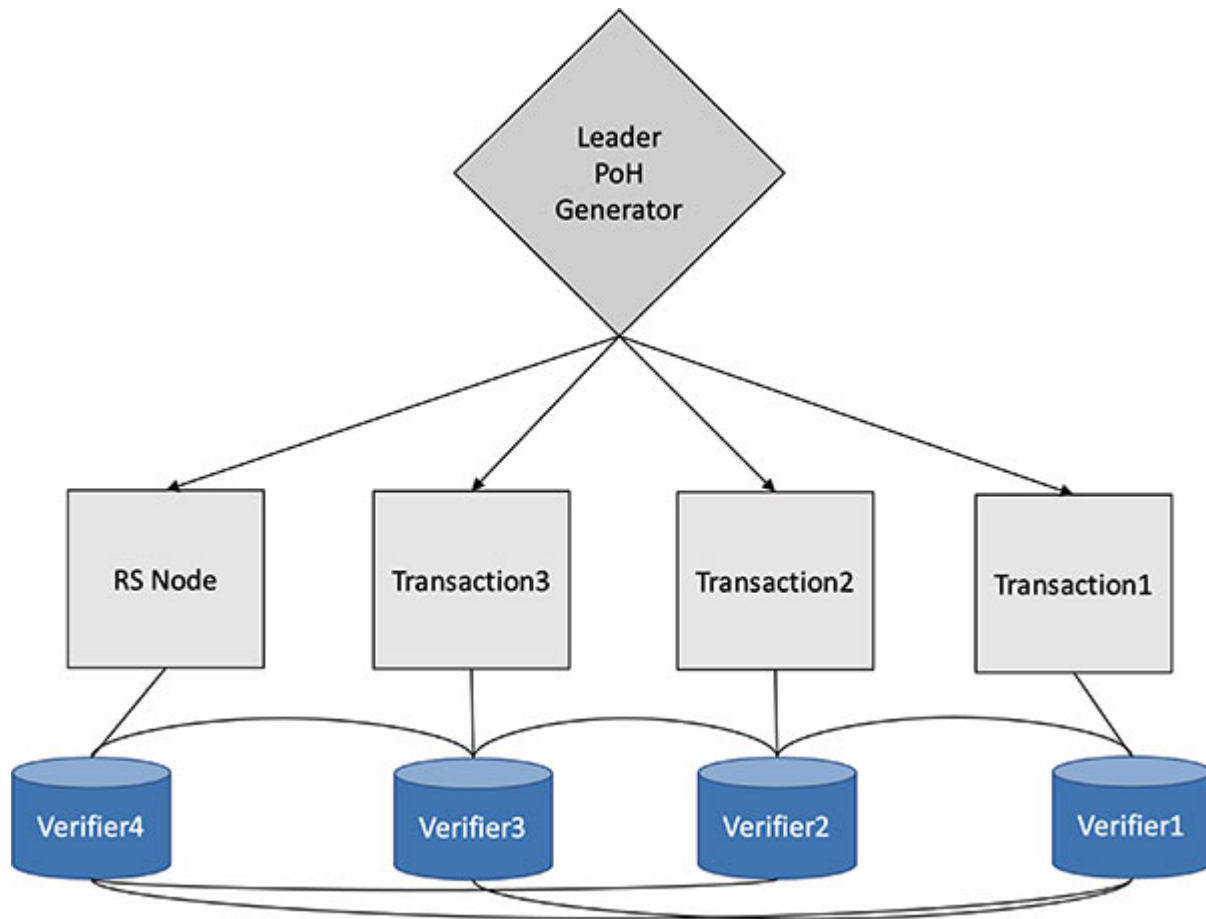


Figure 7.2: Verification through Solana nodes

7.2 Transaction fees

Solana's consensus model ensures that the price per transaction is lower than \$0.01 for the developers and users. At the moment, Solana charges around \$0.00025 per transaction which is one of the lowest in the entire group of public Blockchain networks.

7.3 L2 Solutions

Solana is already designed so well, with so much of innovation that its already fast and massively scalable. Hence, there is no need for an L2 layer, at least for the foreseeable future.

7.4 Scalability and Performance

As per Solana's whitepaper, the Blockchain network is capable of achieving a throughput up to 710K transactions per second with modern hardware, if the transactions, on an average, are not more than 176 bytes. So far, it's tested up to 50,000 TPS and, with adequate configuration, can scale even higher.

7.5 Development

Solana smart contracts are known as programs that can be written either in the C, C++, or Rust programming language. Once written, the program can be deployed on Blockchain.

For transactions and queries, we can interact with a Solana node from a JavaScript client application, and use the solana-web3.js library, that can connect through the RPC methods.

Solana can be tested on Devnet and Testnet before deployment to Mainnet.

7.6 Wallet

Solana comes with many different types of wallet options as hardware, web, or mobile. Trust is a Solana wallet that works on iOS and Android. Exodus is another multicurrency wallet that not only works on iOS and Android but also on Mac, Linux, and Windows. Sollet is a wallet completely confined to the web. The options are many more.

7.7 Live Projects

Some of the important projects on Solana are as follows:

- **Raydium:** Automated market maker to enable lightning-fast trades
- **Serum:** Orderbook based exchange
- **Saber:** Stablecoin exchange
- **Orca:** Cryptocurrency exchange focussed on user-friendliness
- **Oxygen:** DeFi brokerage protocol for trading assets

- **Phantom Wallet:** Digital wallet for DeFi and NFT

So far, the Solana Blockchain platform already has 400+ projects under its kitty that work in different spaces as DeFi, NFTs, Web3 etc. As per Solana's co-founder Raj Gokal, "Over 1,000 projects have taken over Solana since its inception. Every day, 4-5 projects are being announced on Twitter. More than 2,000 transactions are being processed every second on the Solana network; an average transaction costs \$0.00025 per transaction."

7.8 Summary

In this chapter, we covered the following topics:

- Solana's massive scalability with Proof of History & Proof of Stake consensus.
- Wide adoption and live projects on Solana.

References

- White paper - <https://solana.com/solana-whitepaper.pdf>
- Solana's network architecture - <https://medium.com/solana-labs/solanas-network-architecture-8e913e1d5a40>
- Solana SOL price prediction - <https://coinmarketcap.com/headlines/news/solana-sol-price-prediction/>
- Solana (SOL) Explained: Your Guide To The Fastest Cryptocurrency - <https://solanaguide.github.io/>

CHAPTER 8

Avalanche

“The measure of speed for a blockchain is *latency*. The time from submission to completion, also known as time to finality.. Latency, a measure of speed, is measured in seconds, not TPS. High capacity doesn’t necessarily imply low latency. If someone says they are the *fastest*, and offers numbers measured in TPS, they are either trying to pull a fast one on you or they don’t know what they are talking about.” - Emin Gün Sire, Founder & CEO Ava Labs.

Launched: September 2020

Native token: AVAX

Market Capitalization: \$14.66B

Smart Contract languages: Solidity

In this chapter, we will explore Avalanche, one of the hottest names among the Layer 1 Blockchains, and especially DeFi market today.

Avalanche claims to be capable of processing 10,000 TPS with 4000 validators. In terms of time, to reach the finality, Avalanche Blockchain claims to be the fastest among all contemporaries as it takes just 1-2 seconds for finality.

8.1 Consensus Model – Proof of Stake

Internally, Avalanche is configured with three different Blockchains, namely Exchange Chain (X-Chain), Platform Chain (P-Chain), and Contract Chain (C-Chain), as shown in [Figure 8.1](#), as follows:

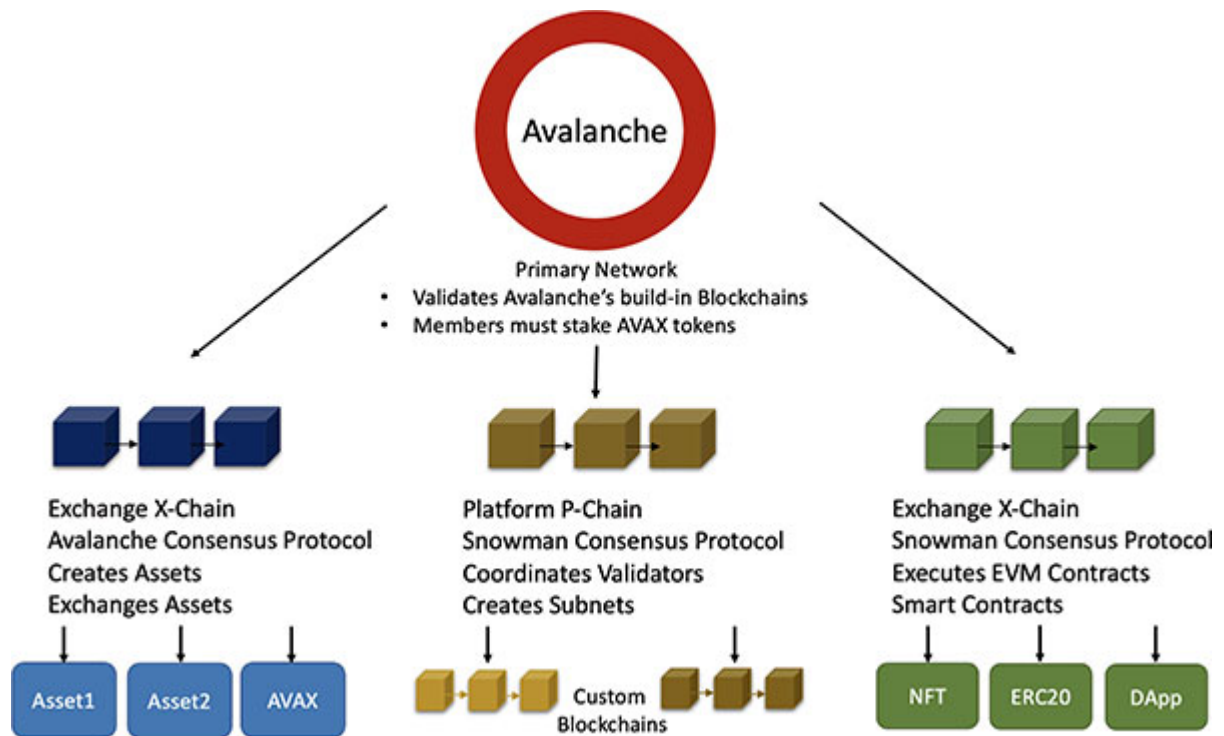


Figure 8.1: Avalanche Platform overview (Source: <https://docs.avax.network/learn/platform-overview>)

The X-Chain creates and manages assets, P-Chain handles the validators as well as creation of custom blockchains, and C-Chain gets the smart contracts executed. Also, please note that while X-Chain, also called as “Avalanche Virtual Machine” or AVM, works on Avalanche consensus, P-Chain and C-Chain adhere to another type of consensus known as the Snowman consensus protocol.

The validation of the transactions in Avalanche happens through repeated voting. A validator node doesn't finalize any transaction just by self-analysis but selects a subset of other validators every time and collects their confidences on the validity of the transaction until a threshold is reached, as shown in [Figure 8.2](#) as follows:

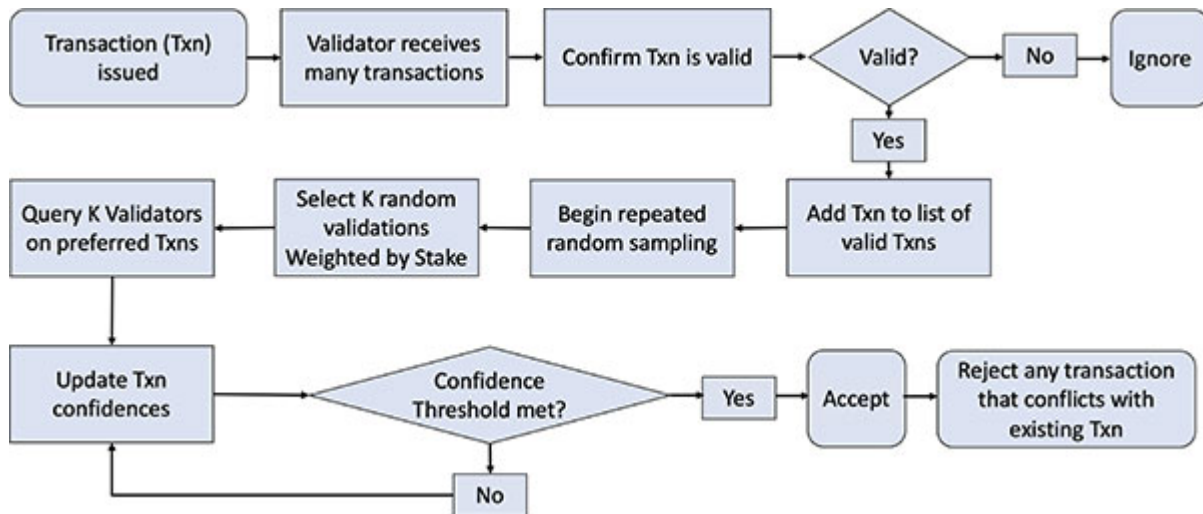


Figure 8.2: How Avalanche Consensus Works Source: <https://docs.avax.network/learn/platform-overview> and <https://research.thetie.io/avalanche-ecosystem/>

Avalanche is fully compatible with EVM, and hence, many developers are moving their smart contracts from Ethereum to Avalanche Blockchain. Avalanche works on a special architecture that works on subnets. A subnet consists of a dynamic set of validators that handle the consensus model of one or more associated Blockchain(s). But, each Blockchain is associated with just one subnet. The primary network in Avalanche is a special subset on which the custom subnets can be built. One can be a member of these custom subnets by staking 2000 AVAX.

The Subnet model enables Avalanche to support the private Blockchains that can be built with selected validators, which is kind of Avalanche's forte.

8.2 Transaction fees

Avalanche transactions need fees to get executed on the network. This is also a way to keep the spammers and hackers at bay. Different types of transactions require different types of fees which are paid in Avalanche's native token AVAX. The transaction fees on Avalanche can be found on the table that is available at the following link:

<https://docs.avax.network/learn/platform-overview/transaction-fees>

As the price of AVAX coin keeps varying, so does the fees. In fact, this pricing model is similar to that of Ethereum's. Many organizations seek a Blockchain model with a fixed price per transaction and this model can be a deterrent to its wider adoption in the market.

8.3 L2 Solutions

Avalanche currently does not handle any L2 solutions, as it's already fast enough with its L1 architecture with three different Blockchains assigned to handle different types of tasks.

8.4 Scalability and Performance

Though in terms of transaction throughput, Avalanche is at 4500 TPS, yet each of those transactions can reach finality within 1-2 seconds. As per Avalanche, it can reach 10,000 TPS with a specific set up of the validator nodes.

8.5 Development

As Avalanche works on Solidity, it would not need much extra work in terms of development. The slow running DApps, deployed on Ethereum, can be considered to move to Avalanche for its scalability and significantly faster speed execution.

8.6 Wallet

First of all, Avalanche has its own non-custodial type web wallet called Avalanche wallet. Ledger Nano X + Avalanche is a hardware wallet for Avalanche. Similarly, MetaMask has a browser extension and mobile wallet support for Avalanche. Coin98 also has support for Avalanche through its mobile and browser extension wallet.

8.7 Live Projects

Some of the most popular Avalanche projects running in production are as follows:

BENQI – Money market project where people can liquidate their wealth, convert them to tokens, lend, borrow, and earn interest rates.

Penguin Finance – NFT platform

Snowball: DeFi protocol that gives the users rewards upon participation and helps in re-investing those rewards.

8.8 Summary

In this chapter, we covered the following topics:

- Unique consensus model of Avalanche that leads to the fastest scalability among all Blockchain networks.
- Live projects on Avalanche.

References

- Scalable and Probabilistic Leaderless BFT Consensus through Metastability (Avalanche Whitepaper) - https://assets-global.website-files.com/5d80307810123f5ffbb34d6e/6009805681b416f34dcae012_Avalanche%20Consensus%20Whitepaper.pdf

PHASE II – INTEROPERABLE BLOCKCHAINS

Blockchain is there in the market for over a decade and there are thousands of DApps that are already running in production or in UAT waiting for their turn to be online. Though it all started with Ethereum, gradually, the market is invaded by many different types of public Blockchains, and there is an increasing need for DApps installed on different Blockchains to interact with each other seamlessly. Hence, it is high time that Blockchain should also focus on interoperability.

So why is interoperability so important?

Interoperability is the ability of the Blockchain networks to connect to other such networks and share the data seamlessly. It's important because, most Blockchains today are implemented in silos and they are not compatible with each other to execute cross-chain data sharing.

Now, let's discuss some of the most popular interoperable Blockchains of today, followed by a comparison in the end.

CHAPTER 9

Polygon

“Decentralisation is not an option, it’s THE very purpose this all started for!” - Sandeep Nailwal, Co-Founder Polygon

Launched: October 2017

Native token: MATIC

Market Capitalization: \$7.57B

Smart Contract languages: Solidity & Vyper

In this chapter, we will cover Polygon, one of the most talked about interoperable Blockchains of today. Launched in October 2017, under the name of “Matic Network”, and co-founded by Jaynti Kanani, Sandeep Nailwal, Anurag Arjun, and Mihailo Bjelic, this Blockchain initially tried to solve the perpetual drawback of Ethereum, i.e., scalability. Originally, Matic used an L2 technology called Plasma with which it could create a child Blockchain that could take some load of the transaction processing from layer 1.

The company was rebranded as Polygon only in 2021 with superb interoperability features keeping the name of the native token the same as “MATIC”. Being fully compatible with Ethereum, many of the existing Ethereum based DApps preferred to move to Polygon for higher scalability, lesser transaction fee, and future-proofing with interoperability. No wonder, today Polygon has 3000+ projects under its kitty, hundreds of partners, and a backing from crypto exchanges like Coinbase and Binance.

9.1 Consensus Model – Proof of Stake

In [Figure 9.1](#), we can see how Polygon can integrate with the other public blockchains or other public interoperable blockchains seamlessly, as shown as follows:

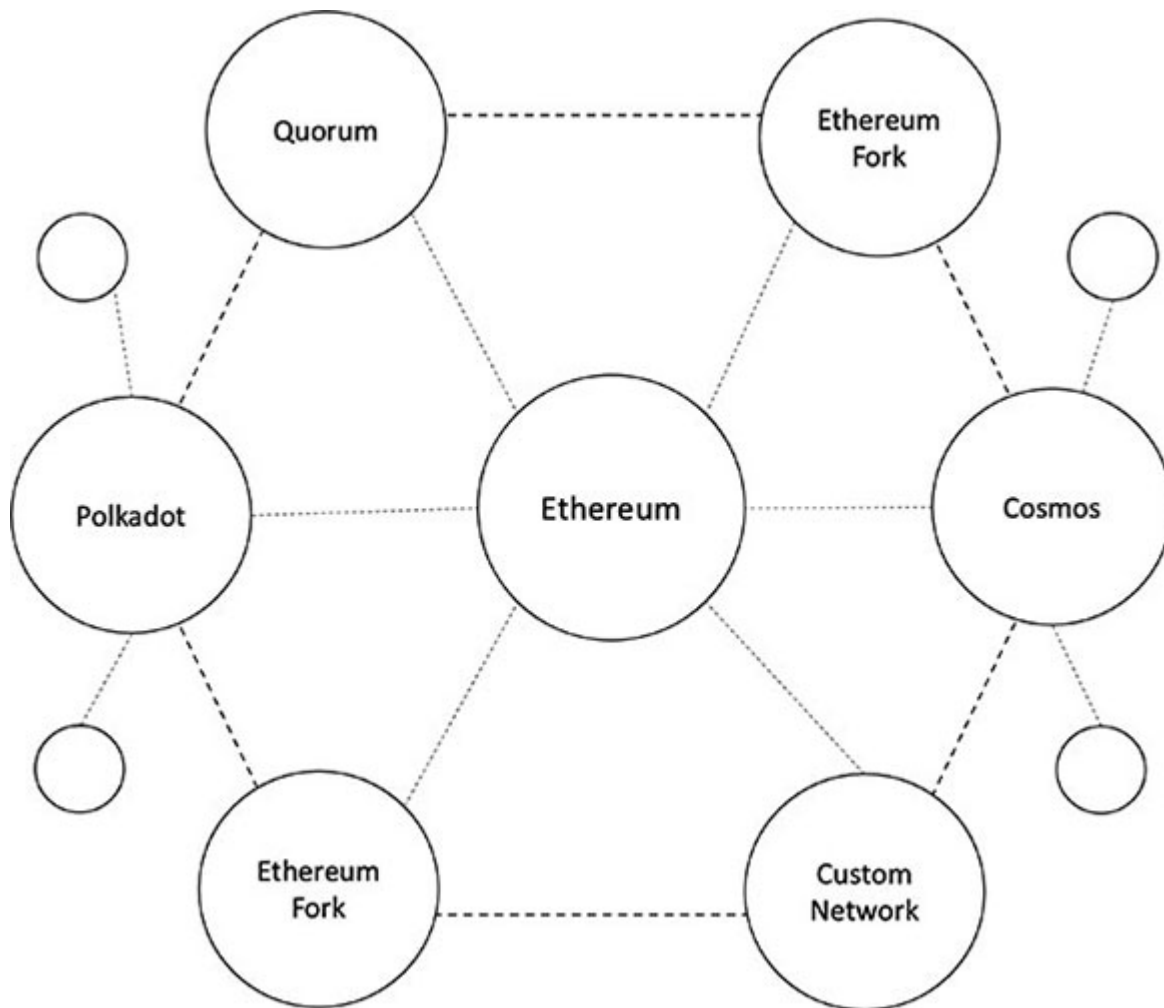


Figure 9.1: Polygon Basic Architecture

Polygon works on the Proof of Stake consensus, where validators invest their stakes to validate different transactions on the side chains. When it comes to the MATIC tokens, the users have to lock their funds on the parent Ethereum network on smart contracts and then they can be minted in MATIC in the Polygon network. In order to redeem the tokens, first they have to be burnt on the Polygon, and the proof has to be sent to the Ethereum layer from where they can be refunded again.

9.2 Transaction fees

The transaction fees in Polygon can be pretty low because of the involvement of side chains. Currently, it's about \$0.0001, which is one of the lowest in the entire Blockchain market.

9.3 L2 Solutions

Polygon uses two L2 scaling solutions, called ZK-Rollups and Optimistic rollups, that works on the concepts of zero-knowledge and fraud proof respectively.

Please note that in the current architecture, Polygon has only Ethereum as the base layer; however, as it matures, it has plans to use the other Blockchains to be configured as the base, that would ultimately make Polygon a cross-chain L2 platform.

9.4 Scalability and Performance

As per the Polygon promoters, this interoperable Blockchain can achieve a scalability of up to 65,000 TPS with a block time of just 2 seconds (<https://cryptobriefing.com/behind-polygon-mission-become-aws-ethereum/>). However, theoretically, Polygon can achieve a far greater scalability of millions of transactions per second by adding more side chains.

9.5 Development

The step-by-step instruction for the development and testing of DApps can be found at the Polygon's website <https://sdk-docs.polygon.technology/docs/overview/>.

The development engagement is pretty high in the Polygon Blockchain. In fact, many of the original developers of Solidity on Ethereum, reskilled themselves to work on Polygon, as the underlying language is the same, i.e. Solidity, and only the configuration part needs some extra attention.

9.6 Wallet

Polygon is supported by a huge number of wallets of different types. Ethereum's existing Metamask browser wallet, WalletConnect mobile wallet, Coinbase wallet, Math and Onto cross-chain wallets, SafePal hardware wallet etc., are a few of them.

9.7 Live Projects

Among the interoperable Blockchains, Polygon has observed maximum participation. A few of them are listed as follows:

QuickSwap is an automated market maker (AMM) product which was earlier deployed on Ethereum under the name of Uniswap and used to be one of the most popular DApp. Using this product, the users can easily swap between the different types of ERC-20 crypto tokens and generate liquidity. QuickSwap uses its own cryptocurrency called QUICK.

PlotX is a non-custodial prediction product, using which, the users can earn unlimited rewards on the high-yield prediction markets.

Zapper is another product, using which, the users can interact with more than 50 different decentralized finance protocols through one interface.

Aave is a money market product, using which, the users can trade and also earn interest. Aave supports a feature called “flash loans”, using which, the users can get instant loans without collateral; however, they have to repay the loan within the same block.

***Note:** As of now, Polygon is the most sought after interoperable Blockchain with many advantages. One of the major advantages of Polygon is its underlying Ethereum layer. The only concern though is that, with the introduction of Ethereum 2.0 in its full-fledged version, there can be some impact on Polygon adoption.*

9.8 Summary

In this chapter, we covered the following topics:

- Polygon’s high adoption among all interoperable blockchain networks.
- ZK-Rollups and Optimistic Rollups, Polygon’s unique L2 technology.

References

- Polygon (MATIC): A Blockchain with Indian founders and a great potential -

<https://www.thenewsminute.com/article/polygon-matic-blockchain-indian-founders-and-great-potential-156293>

- Pros and Cons of investing in Polygon - <https://trading-education.com/pros-and-cons-of-investing-in-polygon-matic>
- Polygon Overview: Everything You Need to Know About MATIC - <https://www.cryptovantage.com/buying-crypto/polygon/>
- Polygon (Matic) – Ethereum's Internet of Blockchains - <https://finematics.com/polygon-matic-explained/>
- What is Polygon (MATIC)? A Guide on the Ethereum Layer Two Solution - <https://coincentral.com/what-is-polygon-matic-a-guide-on-the-ethereum-layer-two-solution/>

CHAPTER 10

Polkadot

“Governance is Critical for Crypto Ecosystem. The world, in some sense, belongs to coders.” – Gavin Wood, Co-founder Ethereum, Creator of Polkadot and Kusama.

Launched: 2020

Native token: DOT

Market Capitalization: \$27.69B

Smart Contract languages: Rust and JavaScript

In this chapter, we will learn about Polkadot, a May 2020 launched multi-chain architecture that was co-founded by Gavin Wood who had earlier co-founded Ethereum. Polkadot is a third generation public Blockchain that handles many issues that the earlier public Blockchains lacked. Developed by the Web3 foundation and implemented by Parity technologies, Polkadot comes with some of its own differentiators that very few other Blockchain protocols have solved so far, i.e., Interoperability, Scalability, and Security.

As per Polkadot, Blockchain not communicating is like having many different non-interacting Internets all over the world. Hence, Interoperability is the core strength of Polkadot.

At the heart, Polkadot is not just another Blockchain, but a framework for Blockchains where many other Blockchains can be created.

10.1 Consensus Model – Nominated Proof of Stake

Polkadot is an innovative solution that comes with many different layers of Blockchain, as shown in [Figure 10.1](#) as follows:

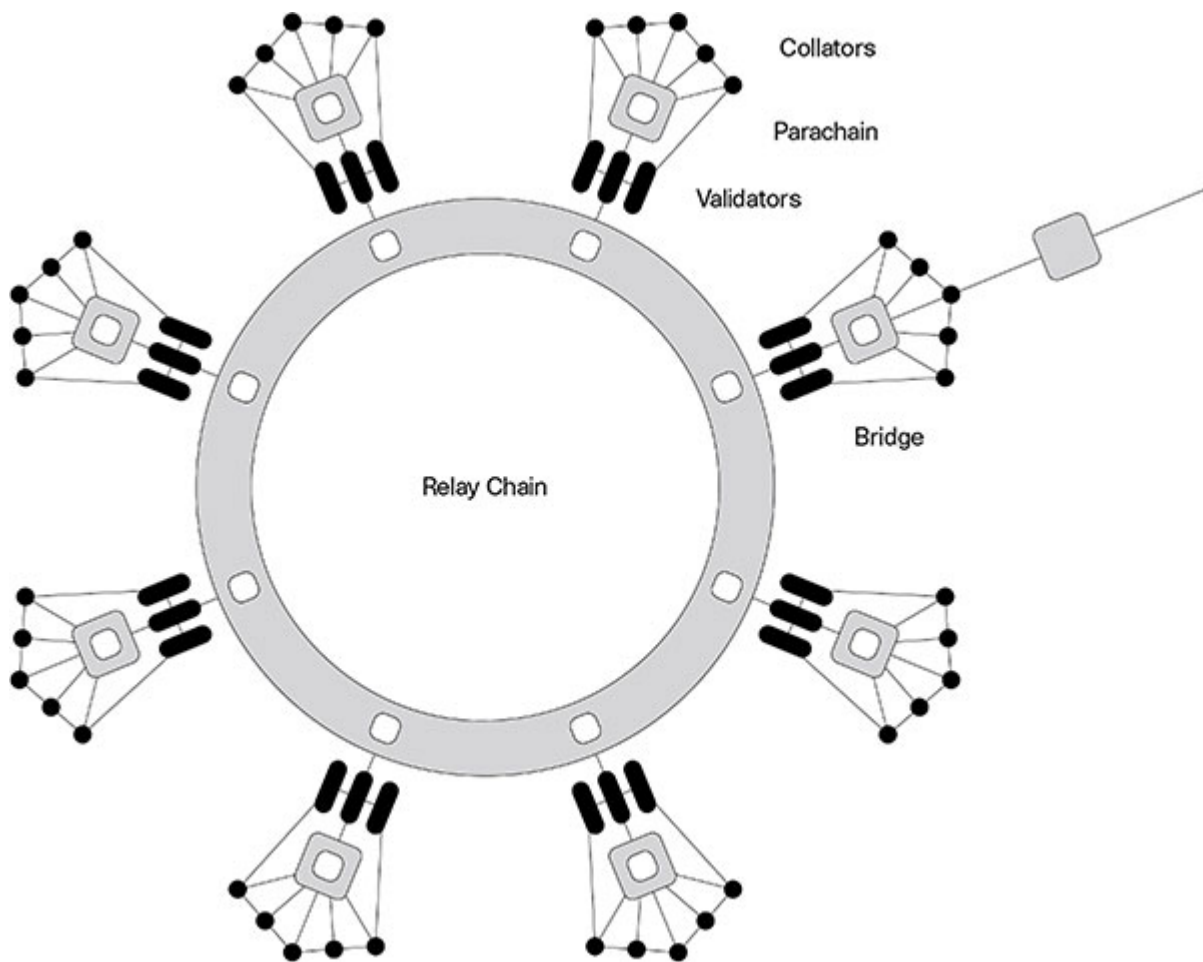


Figure 10.1: Architecture of Polkadot
(Source: <https://wiki.polkadot.network/docs/getting-started>)

Relaychain

It's the main Blockchain that plays a central hub in Polkadot's ecosystem. The Relaychain has all the validator nodes staked on its network. The Relaychain's main task is to handle and coordinate the entire network of Blockchains together. It does not support smart contracts.

Parachains

These are different parallel Blockchains that can work independently on top of the main relaychain. The Relaychain has many different slots for Parachains to occupy and stay connected with the Relaychain for a particular time period.

Bridge Chains

These chains help connect the different Parachains with the main Blockchain or the Relay Chain, as shown in [Figure 10.2](#), as follows:

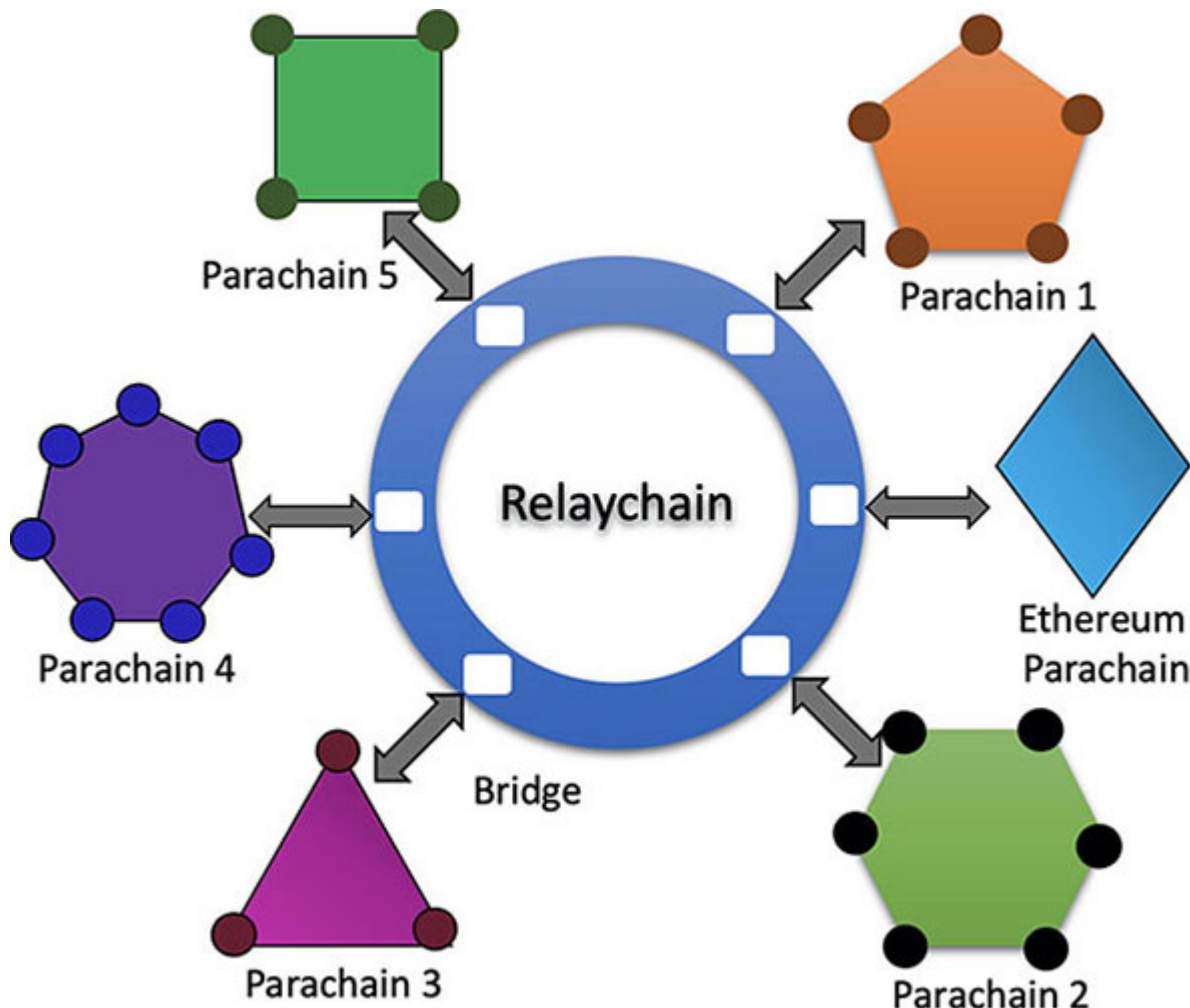


Figure 10.2: Layers of Blockchains in Polkadot

Polkadot makes sure that the Relaychain or the mainchain and the Parachains or the sidechains share the same state with each other.

Polkadot works on the principle of “Nominated Proof of Stake” consensus where there can be three different types of members, i.e., validator, nominator, and collator, which are described as follows:

- Parties can be Validators by staking Polkadot’s native token “DOT”. **Validators**, if selected as among the validator set, can validate and publish blocks on the Relaychain and earn DOT tokens in the process. They also collect proofs from the

collators to make sure that the Parachain transactions are trustworthy.

- **Nominators** nominate and vote for the validators by bonding their stakes to the validator. The selected validators can validate blocks, and in return, the validator pays the nominators with rewards.
- **Collators** are the nodes that collect the transactions on the Parachains, produce proofs, and pass on to the Relay chain.

10.1.1 Parachain Slots

The maintenance of Parachains need cost and effort. Hence, to start with, Polkadot has a limited number of Parachain slots that can grow with time. These slots can be reserved by the Parachains through auctions.

The existing Blockchains such as Ethereum, Bitcoin, Cardano, Tezos, Solana, ZCash etc., can become Parachains to the Polkadot Blockchain's relay chain or main chain.

10.2 Transaction fees

In Polkadot, the transaction fees are only for the Relay Chain or the main Blockchain network. It is calculated with an algorithm that depends on the Weight fee or time required to validate a transaction and the tips or an optional fee that the users may add, just to run the transaction with a higher priority.

10.3 L2 Solutions

Polkadot cannot be termed as a simple L2 solution; rather, it's a multichain architecture that can work on a parallel technology with a combination of parallel chains as well as sharding. It has gathered more success in recent times.

10.4 Scalability and Performance

Polkadot's sharding protocols help in easy cross-chain message transfer between Parachains. The architecture enables to process a

huge number of transactions simultaneously removing the bottlenecks. Currently, Polkadot has a TPS of around 166,666.

While, in the other Blockchains, we write smart contracts to trigger transactions, in Polkadot, it's more about the integration of different Blockchain protocols with its Relay Chain. Polkadot also uses Oracles heavily for easy integration of the external APIs with the distributed ledger. Hence, choose Polkadot when you have a special need for the interoperability of Blockchains along with the other crucial features such as security, scalability, throughput etc.

[10.5 Development](#)

Polkadot is 100% opensource and comes with fantastic documentation. In Polkadot, Parachains or custom Blockchains, to suit the needs of particular use cases, can be created from scratch using special templates called **Substrates**. The details can be found in the following website for a step-by-step development:

<https://substrate.io/>

Smart contracts can be written and executed on Parachains. There are three different types of networks for deployment, i.e., three different Testnets (Westend, Canvas, Rococo), Kusama Canary network, and Polkadot Mainnet.

Polkadot is opensource and anyone can contribute to this community work.

[10.6 Wallet](#)

Parity technologies give support for many different wallets that one can use on iOS, Android, Windows, Mac, Linux, or the browser extensions.

[10.7 Live Projects](#)

Launched only in mid 2020, Polkadot has 511 projects, many of which are running successfully in production and still counting. Refer to the following link: <https://polkaproject.com/#/>

10.8 Summary

In this chapter, we covered the following topics:

- High scalability with Unique Parachain based L2 concept of Polkadot.
- Substrates for building custom Blockchains as per the requirement of individual use cases.

References

- Polkadot Wiki Architecture - <https://wiki.polkadot.network/docs/learn-architecture>
- What is Polkadot? | A Polkadot for Beginner's Guide and Intro to Blockchain - <https://www.youtube.com/watch?v=kW8eu2VadFA>
- Polkadot Is the Only Blockchain Protocol Fitting the Revolutionary Bill: Interview With Parity - <https://cryptopotato.com/polkadot-is-the-only-blockchain-protocol-fitting-the-revolutionary-bill-interview-with-parity/>

CHAPTER 11

Cosmos

“Transaction fees on OSMOSIS and COSMOS are going to be so much cheaper than PoW, I’m betting that trading bots and their market volumes will quickly move over to the Cosmos ecosystem, once they figure it out.” - Jae Kwon, Founder and CEO of Tendermint, co-creator of Cosmos

Launched: March 2019

Native token: ATOM

Market Capitalization: \$7.7B

Smart Contract languages: Rust

In this chapter, we will explore the 2019 launched interoperable Blockchain network Cosmos that has become a very popular option for DApps. The goal of Cosmos is to build an “Internet of Blockchains” platform, using which, different Blockchain protocols can communicate with each other independently.

11.1 Consensus Model – Proof of Stake Tendermint

The Cosmos network is composed of three different layers, namely application, network, and consensus, which are explained as follows:

1. **Application layer** is the topmost layer that handles the transactions and keeps the information updated on the network.
2. **Networking** connects the transactions to all the Blockchains.
3. **Consensus** helps all the nodes to agree on a common state of the system.

Refer to [*Figure 11.1*](#), that illustrates the Cosmos network, as follows:

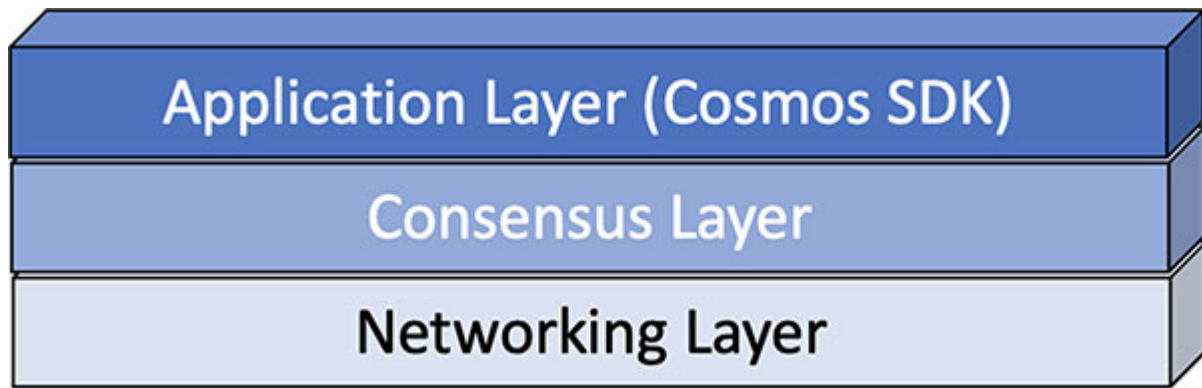


Figure 11.1: Cosmos Network

Cosmos uses open-source tools to tie all the layers together and enable the developers to build the Blockchain applications.

The following are the major differentiators of the Cosmos Blockchain network:

Tendermint: The consensus model that Cosmos follows is called “Tendermint Core Byzantine Fault Tolerance” or simply “Tendermint”. The Tendermint BFT engine helps in forming the layered design of Cosmos by allowing the developers to create the Blockchains without having to code from scratch.

IBC: The “Inter Blockchain Communication” protocol or IBC is responsible for the communication between the layers of the Blockchain network. IBC enables the developers to build a wide range of cross-chain DApps that can do token transfers, run smart contracts and data, and code sharding of different kinds.

Cosmos SDK: Opensource framework to build multiple custom Blockchains from scratch that can communicate with each other.

In order to connect all the different Blockchains, Cosmos follows a “**Hub and Spoke**” architecture, where a hub or the centre can connect to multiple spokes, each bearing a different Blockchain, as shown in [Figure 11.2](#). Each Blockchain has its own set of validators that can work independently. Refer to [Figure 11.2](#) as follows:

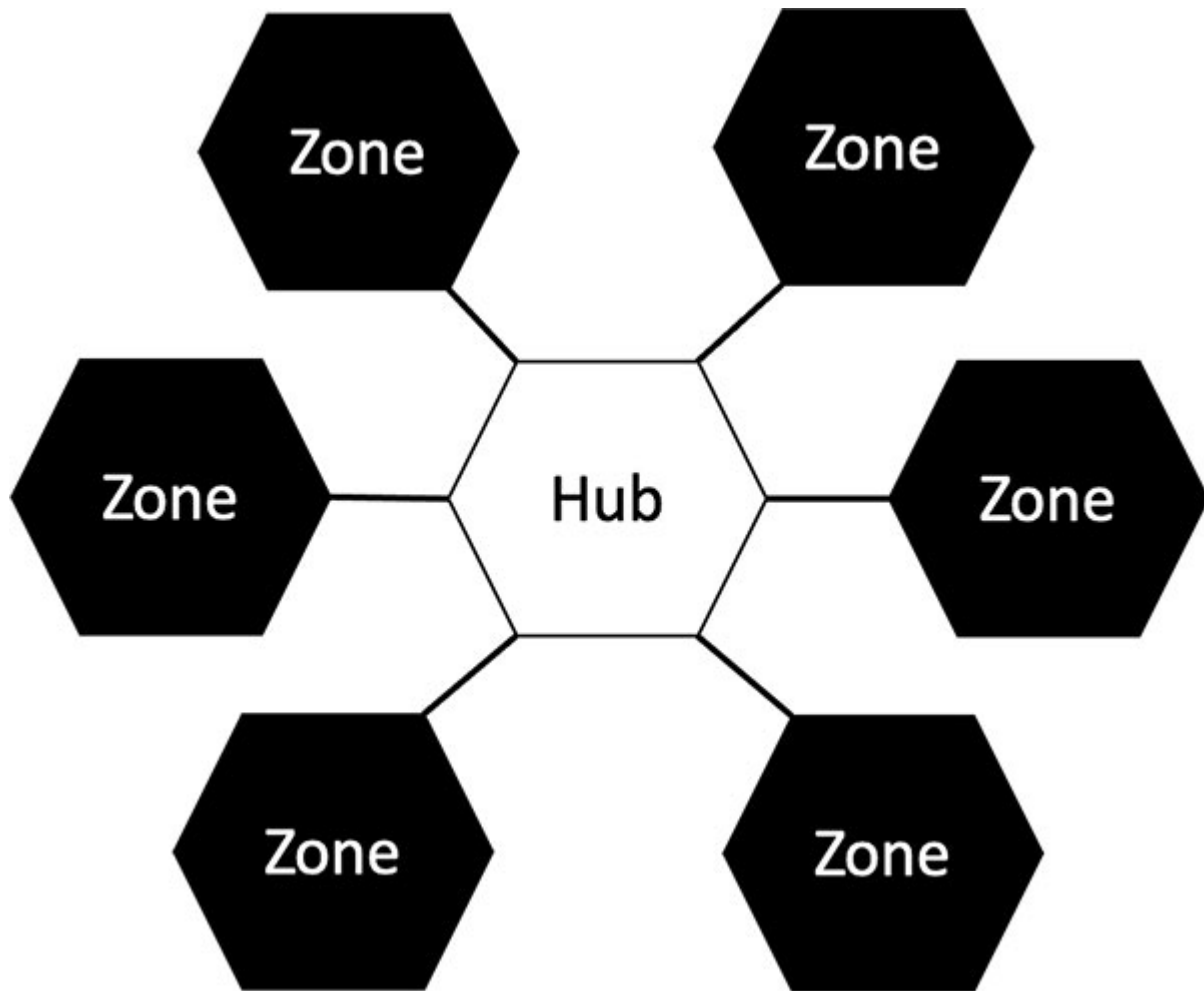


Figure 11.2: Hub and Zones Architecture of Cosmos

Also, [Figure 11.3](#) demonstrates that there can be multiple hubs that can connect to the other hubs. Hence, this architecture can scale infinitely. Refer to [Figure 11.3](#), as follows:

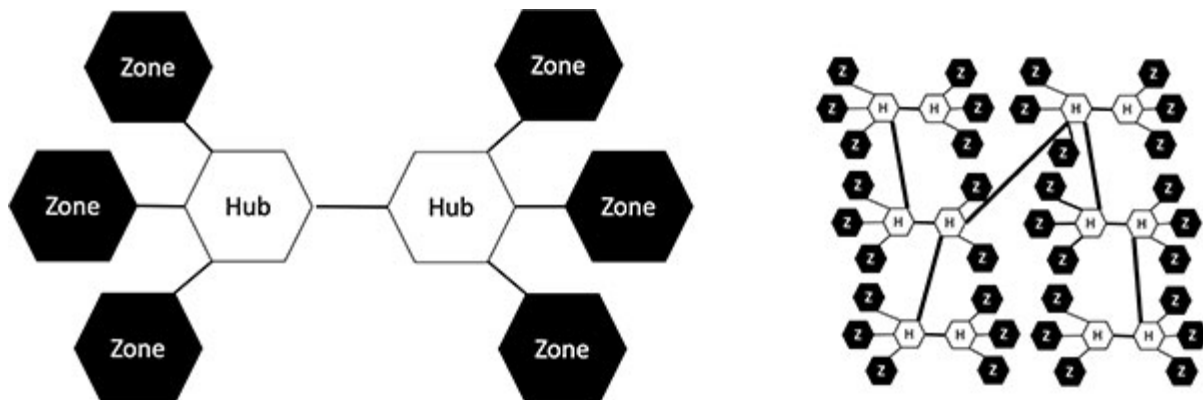


Figure 11.3: Hub to Hub communication in Cosmos

The zones or Blockchains share the information with each other through a hub with the help of the IBC protocol, as shown in [Figure 11.4](#). The hub keeps a check on the “double spending” by the zones. It also keeps track of all the zones and their headers and provides the proof to the other zones during the transactions. Refer to [Figure 11.4](#), as follows:

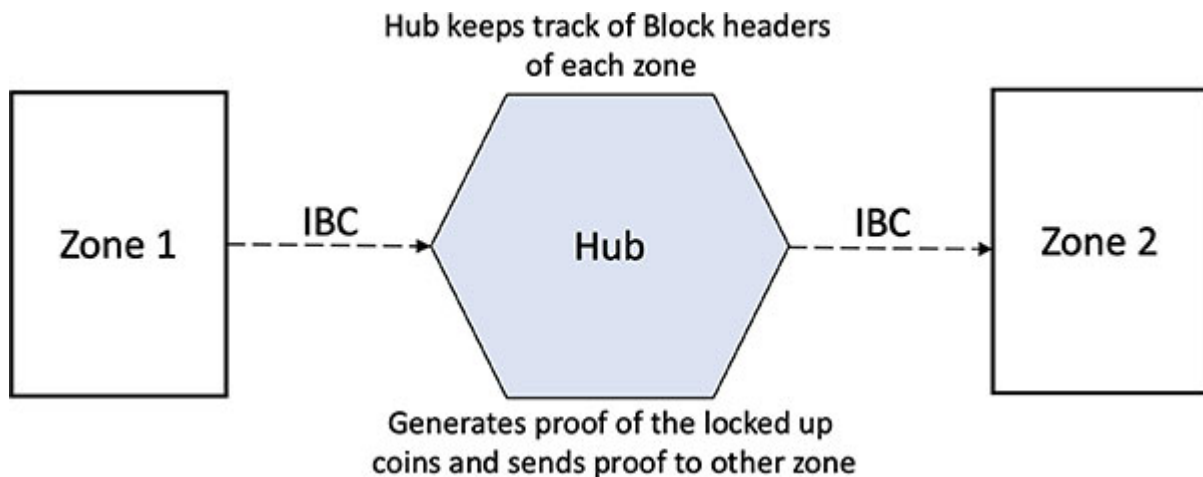


Figure 11.4: Hub generates trust in Cosmos

Hence, the zones do not need to trust each other. They just have to trust the hub.

These types of interoperability works for the Blockchain zones that can abide by the Tendermint consensus. However, when it comes to interactions between a Tendermint and a non-Tendermint Blockchain, such as PoW based Bitcoin or Ethereum-I, then we have to introduce a “Peg Zone” or a Blockchain that tracks the state of another Blockchain, as represented in [Figure 11.5](#) as follows:

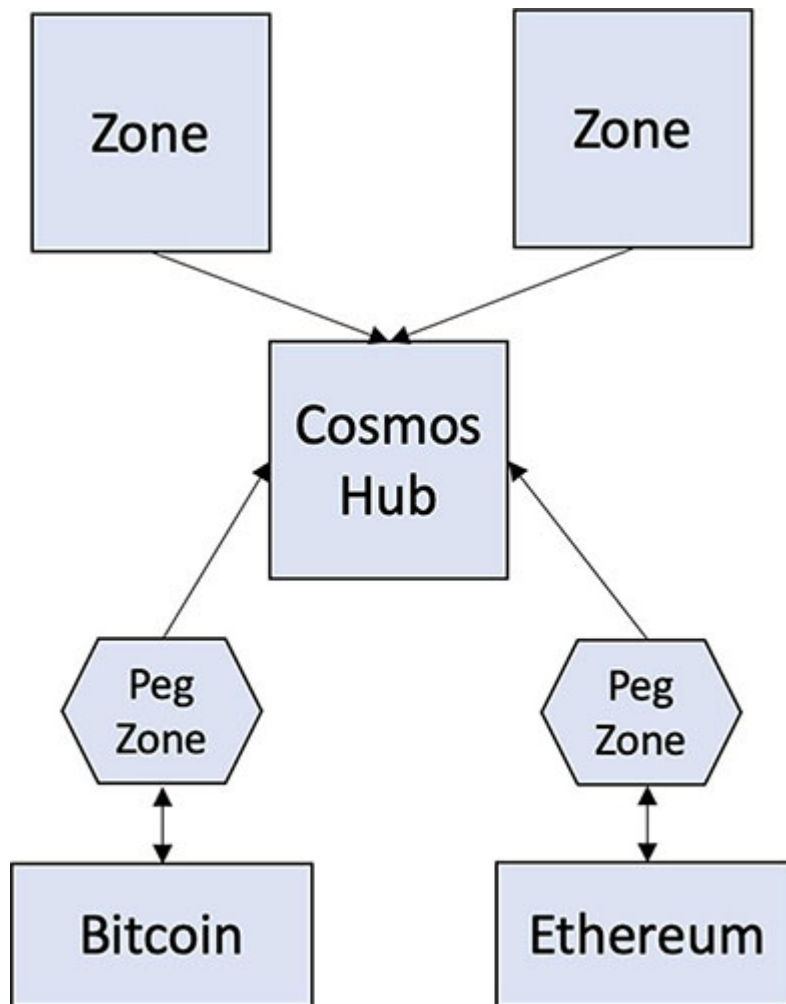


Figure 11.5: Peg Zone in Cosmos for interoperability

11.2 Transaction fees

The transaction fees are again dependent upon the gas and the prices of gas. The native currency of COSMOS is ATOM. However, the cost of the transactions as per price at the moment is close to \$0.01.

11.3 L2 Solutions

When it comes to layer2 solutioning, Cosmos itself is nothing but a sidechain framework on which independent Blockchains can be configured.

11.4 Scalability and Performance

Cosmos's underlying consensus, Tendermint BFT can handle thousands of transactions per second and have a block time on the order of one second. Instant finality means that forks are never created until more than one-third of the validators are honest (byzantine).

As per an article on Coingraph in April 2021, Cosmos can scale up to 100,000 TPS which makes it the fastest and most scalable platform on the market. Refer to the following link for more information:

<https://cointelegraph.com/news/this-blockchain-can-scale-to-100-000-tps-here-s-what-that-would-mean-for-you>

11.5 Development

Cosmos website, <https://v1.cosmos.network/sdk> has good documentation, using which, a developer can easily build a custom Blockchain.

11.6 Wallet

Cosmos has a group of wallets that support the Blockchain on different platforms like iOS, Android, browsers, MacOS, Windows, Linux etc.

11.7 Live Projects

As per the website of Cosmos, there are 259+ apps and services running on Cosmos, the list for which can be found on the following website:

<https://cosmos.network/ecosystem/apps>

11.8 Summary

In this chapter, we covered the following topics:

- Unique consensus model Tendermint that helps in layered design of Cosmos.

- IBC protocol responsible for communication between layers of Blockchain network.

References

- Cosmos — an early in-depth analysis at the ecosystem of connected Blockchains — Part Two - <https://cryptoseq.medium.com/cosmos-an-early-in-depth-analysis-at-the-ecosystem-of-connected-blockchains-part-two-2d5a9886166>
- How does Cosmos work - <https://www.preethikasireddy.com/post/how-does-cosmos-work-how-does-it-compare-to-bitcoin-and-ethereum-part-2>

CHAPTER 12

Comparison of Blockchains

While most Blockchain websites would speak on their functionalities, while choosing one over another, we have to explore their features and select the one that matches our requirements and are also well accepted in the industry. In this chapter, the reader can find a comparative study that can work as an accelerator in Blockchain adoption. Please note that some of these parameters are highly volatile and may look very different by the time the book is launched.

12.1 Comparison of Public Blockchains – Ethereum, Hashgraph, Cardano, Algorand, Avalanche, and Solana

Now, let's compare all the public Blockchains that we studied so far in [Table 12.1](#), as follows:

Blockchain /DLT	DLT Type	Consensus	Supply	Scalability & Finality	L2 Technology	Market Adoption	Tx Fee	Smart Contract Language	Market Cap
Ethereum	Blockchain	PoW -> PoS	Unlimited	Can scale with Ethereum 2.0	Yes with Ethereum 2.0	Very high	Very high	Solidity	\$355.8B
Hedera Hashgraph	DAG	PoS	Limited	10K TPS	No	Very high	Very low and steady	Solidity	\$3.29B
Tezos	Blockchain	Liquid PoS	Unlimited	40 TPS	Researching	High	Low but considerable	OCaml	\$5.11B
Cardano	Blockchain	PoS Ouroboros	Limited	Linear scalability with Hydra	Yes, Hydra	High	Low but considerable	Plutus, Marlowe, or Glow	\$67.67B
Algorand	Blockchain	Pure PoS	Limited	46K TPS	No but supports	Very high	Very low	TEAL	\$9.7B
Avalanche	Blockchain	PoS	Limited	4500 TPS	No	Very high	Low	Solidity	\$14.66B
Solana	Blockchain	Proof of history & PoS	Unlimited	65K TPS	No	Very high	Very low	Rust, C, C++	\$41.33B

Table 12.1: Comparison of public Blockchains

The comparison in the preceding table is done on the basis of the following properties that are vital for any organization to select one Blockchain over other, which are the type of DLT, consensus mechanism, scalability and finality, L2 technology if any, market adoption/number of projects, transaction fees, smart contract language, and market capitalization.

Considering all the aspects. Solana and Algorand seem to be ahead of the others.

12.2 Comparison of Interoperable Blockchains – Polkadot vs Cosmos vs Polygon

The interoperable Blockchains are compared on the basis of the time of the launch, language for the smart contract (for considering development or learning efforts), limitations, number of live projects (as of November 2021), quality of documentation, transaction fees (as of November 2021), market capitalization, and the price of one cryptocurrency against USD (as of November 2021). Please find their comparison in [Table 12.2](#), as follows:

Note: All the three interoperable blockchains, i.e., Polygon, Cosmos, and Polkadot are open source protocols.

Blockchain	Launched	Supply	Smart Contract Language	Limitation	No of live projects	Documentation	Transaction Fee (Variable)	Market Cap	Crypto/ USD
Polkadot	May 2020	Unlimited	Rust	Limited slots for hubs, may increase with time. One Relaychain	511	Best	~\$0.011	\$27.69B	\$52.72
Cosmos	March 2019	Unlimited	Rust	Infinitely scalable. Future-proof with many parallel hubs	259	Good	~\$0.01	\$7.7B	\$35.81
Polygon	October 2017	Limited	Solidity & Vyper	Association with Ethereum may have impact in future if PoS is great success. The Mainchain now is only Ethereum, it might change in future.	3000	Need to improve	~\$0.0001	\$10B	\$1.86

Table 12.2: Comparison of public interoperable Blockchains

Among all interoperable Blockchains, Polygon (earlier Matic network) seems to win the race.

12.3 Use Cases on Interoperable Blockchains

While the use cases on public Blockchains are widely discussed, they can achieve a new level by the adoption of interoperable Blockchains. With interoperable Blockchain's infinite scalability and access to the global network, some of the use cases that would be most benefitted are as follows:

- **NFT** or Non-Fungible tokens
- Decentralized Finance or **DeFi**

12.4 Why the organizations stay away from public Blockchains?

In the last five years, the enterprises are more focused on the private permissioned DLT, rather than researching on public Blockchains. Why?

It's because the business model and architecture of the traditional organizations can never align with the philosophies of a public Blockchain. Traditional organizations work on profit and loss models and public Blockchains need a long term vision and it works on token economics.

They can't think of giving the controls in the hands of the public. The name "public chains" explains itself. It can be like the network will handle billions of dollars. Frauds in the cryptocurrency markets are pretty well known. The hackers always target the loopholes in the technology and steal in open daylight by just transferring it to another account, and no one on the earth will be able to do anything about it.

The following are some of the biggest scams in the cryptocurrency market:

In 2015, cryptocurrencies worth US\$5 million were stolen from a Luxembourg based crypto exchange, Bitstamp. Similarly, in 2019, cryptocurrencies worth US\$40 million were stolen from Binance and \$60 million worth of cryptocurrency from NiceHash in 2017. US\$60

million in Bitcoin, Bitcoin Cash, and Monacoin were stolen in September 2018 from Zaif.

And the list goes on.

Hence, one needs to be extra careful while choosing the Blockchain platform for the business, keeping all the risks and loopholes under consideration. In the next chapter, we will explore private and permissioned DLT, the favourite platform for the organizations and the differential features that they offer, other than a public Blockchain.

Quiz

Q1: You need a public Blockchain which would not fork with the changes in the protocol. Which among the following would you choose?

- A. Algorand
- B. Avalanche
- C. Tezos
- D. Solana

Q2: You need a superfast public Blockchain which scales well without the need to set up any L2 technology. Which among the following would you choose?

- A. Algorand
- B. Avalanche
- C. Tezos
- D. Solana

Q3: The co-founder of which of the following public Blockchains earlier co-invented probabilistic encryption, Zero-Knowledge Proofs, and Verifiable Random Functions?

- A. Algorand
- B. Hashgraph
- C. Tezos

D. Solana

Q4: The price of a cryptocurrency is dependent upon which of the following?

- A. Utility
- B. Whether it's limited or unlimited in supply
- C. Hoarding
- D. All of the above

Q5: Which among the following is not a Blockchain?

- A. Ethereum
- B. Hedera Hashgraph
- C. Algorand
- D. Solana

Q6: Which of the following Blockchains have an L2 technology called Hydra?

- A. Ethereum
- B. Cardano
- C. Algorand
- D. Solana

Q7: Which among the following interoperable Blockchains can set up multiple hubs where each hub can have multiple slots?

- A. Cosmos
- B. Polygon
- C. Polkadot
- D. None of these

Q8: Which among the following interoperable Blockchains runs auction to select the owner of the slots for the Blockchains?

- A. Cosmos
- B. Polygon

- C. Polkadot
- D. None of these

Q9: Which among the following interoperable Blockchains have the maximum number of projects on it?

- A. Cosmos
- B. Polygon
- C. Polkadot
- D. All of them have similar number of projects

Q10: Which among the following interoperable Blockchains uses ZK-Rollups and Optimistic rollups for L2 scaling?

- A. Cosmos
- B. Polygon
- C. Polkadot
- D. None of these

Q11: Stablecoins Tether and USDC are built on which Blockchain?

- A. Ethereum
- B. Cardano
- C. Algorand
- D. Solana

Q12: You have a Blockchain product deployed on Ethereum which is facing scalability and interoperability issues. Which interoperable Blockchain would you prefer?

- A. Cosmos
- B. Polygon
- C. Polkadot
- D. None of these

Q13: Smart contracts among which of the following Blockchains can be written using Solidity?

- A. Ethereum

- B. Hedera Hashgraph
- C. Avalanche
- D. All of the above

Q14: Which among the following interoperable Blockchains provides substrates or templates to quickly build custom Blockchains as per the business use case?

- A. Cosmos
- B. Polygon
- C. Polkadot
- D. None of these

Q15: Which among the following interoperable Blockchains uses ZK-Rollups and Optimistic rollups for the implementation of L2 technologies?

- A. Cosmos
- B. Polygon
- C. Polkadot
- D. None of these

Answers

1. C 2. D 3. A 4. D 5. B 6. B 7. A 8. C 9. B 10. B 11. C 12. B 13. D
14. C 15. B

PHASE III – PRIVATE PERMISSIONED DLT

A private permissioned Blockchain is often termed as a “Distributed ledger technology”, or DLT, rather than Blockchain. As the name indicates, such an ecosystem is built with a private network where only limited and trusted parties can build their nodes and share the data with the privacy and security factors intact. But, why was there a need of such an ecosystem, as we already have public Blockchains like Ethereum running in the production? Before jumping to different private permissioned DLTs, let’s first understand why we need such networks, and also, what are the similarities and differences with public Blockchains. Please refer to *Table Phase III* for a comparison of the limitations of public blockchains and the advantages of private permissioned DLT over them for the enterprise use cases, as follows:

	Limitations of Public Blockchains	Advantages of Private Permissioned DLTs
Privacy of Data	Public Blockchains would be ideal for certain use cases such as crypto trading, e-Auction, ICOs etc., where the data shared on such networks are visible to all and, at the same time, the identity of the participants does not matter. However, many use cases in banking, FinTech, Insurance, Healthcare, Aviation, e-Governance etc., would opt for more privacy where they can share the data only with selected nodes not opting to make it global.	Unlike public Blockchains, a private permissioned DLT allows only selected parties to participate in the sharing of data with each other. Security Zones: Many private permissioned DLTs also use security zones (i.e., a mini DLT within a DLT) or private peer-to-peer data sharing with messaging techniques, so that the parties can share the data with a few among them with even additional privacy and security.

KYC	In certain use cases, the identity verification becomes a major requirement, and hence, provision should be there for all the nodes to be onboarded only after proper KYC verification.	Most private permissioned DLTs have an in-built KYC process which adds parties only after stringent background verification which is ideal for banks, fintech, and many other verticals.
Transaction Fees	Most public Blockchains are dependent on their inherent cryptocurrencies associated with the fees for each transaction. Because the cryptocurrencies are too volatile and their prices keep on fluctuating too much, many organizations find it risky to use such infrastructure to host their business.	Parties do not need cryptocurrencies to validate a transaction as they trust each other and collectively can validate the transactions. DLT networks would not need a voting process with 51% consensus or staking mechanism for the validation of transactions. As the nodes trust each other, they can jointly take a decision on whether or not to validate the transactions. Hence, the consensus mechanism is often far too simple.
Crypto Economics	As public Blockchains have their respective cryptocurrencies part, the maintenance of the crypto is always an overhead.	Private permissioned DLTs do not need cryptocurrencies for mining purpose, and hence, have no such maintenance overhead.
Performance and Scalability	Because of the nature of the consensus processes, most public Blockchains are slow and also face scalability challenges.	DLTs are much faster than public Blockchains and also can scale well.

Table Phase III

Though the private permissioned DLTs are not open to all, they use certain common technologies with Blockchain such as asymmetric Cryptography, hashing, multiple nodes with respective data stores, decentralized consensus etc. There might be quite a few DLTs in the

market; however, the top three remain Hyperledger Fabric, R3 Corda, and Consensus Quorum. They come with certain common features, which are as follows:

- Private permissioned DLT
- Open source (Corda has a closed source Enterprise version too)
- Highly scalable, high in performance, and secure
- Support uses cases across the industry

In the next few chapters, let's explore these most popular DLTs in the market and analyse their internal working models that would help us select one over the other while selecting a DLT product for a specific use case.

CHAPTER 13

Hyperledger Fabric

Version 1.0 Launched: February 2018

Founder: Linux Foundation

Current Version: 2.3

Hyperledger Fabric was an open source private permissioned DLT, introduced by Linux foundation whose main contribution came from IBM. Initially, Fabric was built with an objective to handle the issues of the present in the public Blockchain such as better performance and scalability, and at the same time, enjoy the same goodness of Blockchain such as transparency, immutability, neutrality etc. Fabric comes with just one version, both for the opensource and the enterprise requirements, which means that the same Fabric can be configured in many different ways, either to be used in the opensource mode during development, unit and functional testing, UAT etc., or in the enterprise mode for production requirements.

In this chapter, we will learn the key concepts of Fabric and some of these configurations that would help us understand how to use Fabric in real life projects.

13.1 Key Concepts

Hyperledger Fabric is the most widely used private permissioned DLT used in the industry. However, does it meet the requirements of all sorts for every business vertical? What features make Fabric popular and where does it lack? Now, let's explore the different components of Fabric and learn it ourselves.

13.1.1 The Ledger

Just like any other Blockchain, Fabric too has a ledger that is often referred to as a "world state" and stores the data in key-value pair.

Along with the world state, the ledger also has a Blockchain that stores all the historical data, and once stored, the data becomes immutable. As shown in [Figure 13.1](#), the Blockchain provides the world state as the latest or current state of Blockchain, as follows:

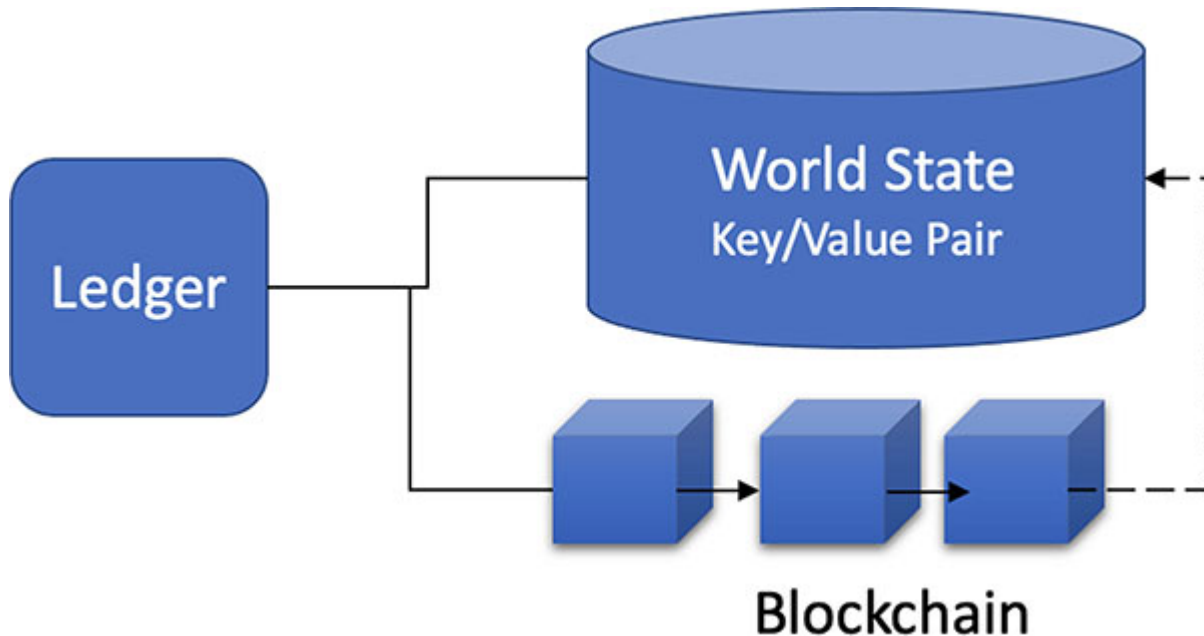


Figure 13.1: Hyperledger Fabric Ledger

The world state database is a pluggable solution that can be configured either with the default LevelDB option or CouchDB. The latter comes with more features as the storage as JSON, rich query options etc., that is preferable for production.

13.1.2 Identity

Identity is paramount in any Blockchain network. Fabric has many different actors such as orderers, peers, and administrators, and each of them have separate roles and access rights associated with their identities. In order to achieve this, Fabric uses the same public key infrastructure, public and private keys, digital signature, certificate authorities, and revocation of certificates, some of which, are already described in [Chapter 1](#).

13.1.3 Membership Service Provider (MSP)

MSP is the trusted authority that has the power to issue, govern, or revoke the certificates of all the actors in the Fabric Blockchain network. By default, it uses the X.509 certificates for identity. Because Fabric is a permissioned DLT, the importance of such a service is paramount. In Fabric, the MSPs can be configured in two different domains, i.e., local MSP and channel MSP. While the local MSP sets the MSP the right levels of peers and orderers; the channel MSP does the same for the channels.

13.1.4 Policies

Policies are a set of rules that determine which organization has access to what resources. Policies can be implemented in the system channel (primary channel) level or application level or smart contract and access control level. Policies make sure of the access by matching the signature associates with each identity in the transactions.

13.1.5 Peers

The Fabric Blockchain constitutes of a number of peer nodes which are generally referred to as peers, as illustrated in [Figure 13.2](#). Each organization can have one or more peers that can host the ledgers and smart contracts. Refer to [Figure 13.2](#), as follows:

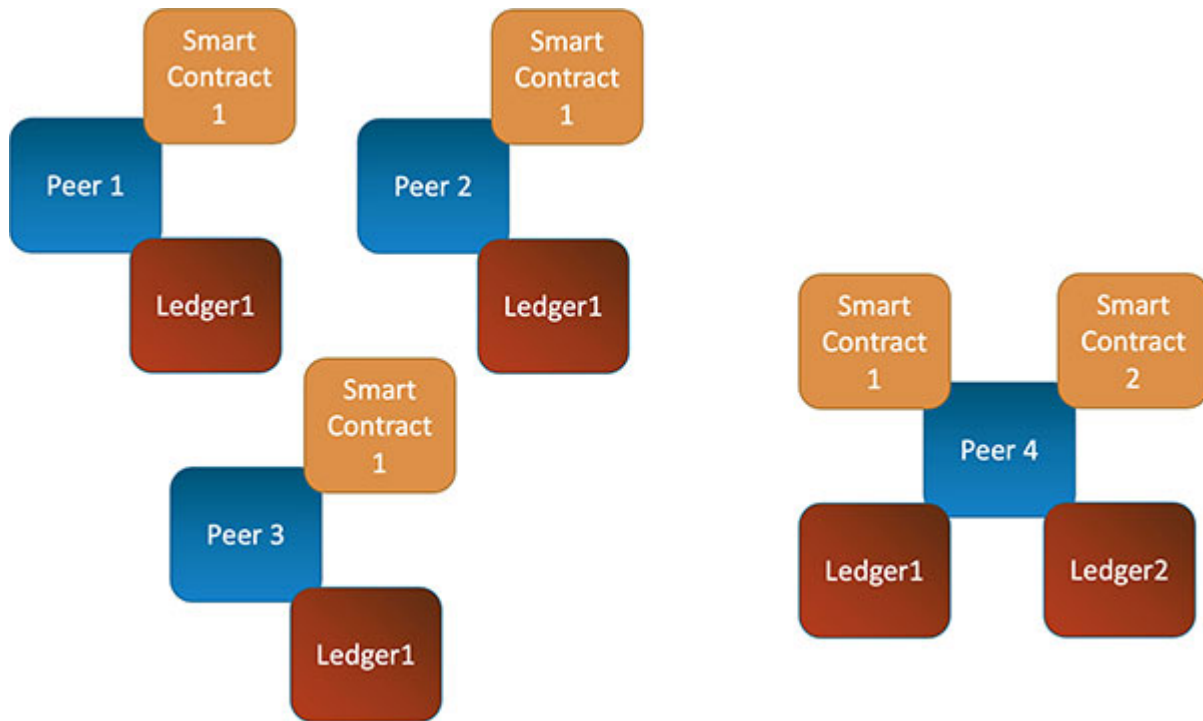


Figure 13.2: Hyperledger Fabric Peers

A peer can also host more than one ledger and smart contracts depending upon the requirement.

13.1.6 Channel

Hyperledger Fabric introduced a concept of security zones with a new feature called channel that enables us to create kinds of Blockchains within a Blockchain. A channel is a zone within which, only the organizations affiliated to that channel can share the information. Of course, there is a base channel on which all the organizations in the network are registered, but then, on the basis of the business requirements, the channels facilitate the organizations to create different channels with a smaller group of selected organizations to share the information within that group.

13.1.7 Orderer

Orderers are nodes responsible for creating an order of transactions after they are endorsed by endorsing the peers. The group of orderers collectively create the ordering service. Each orderer is

bootstrapped with an initial channel known as the system channel. Orderers maintain a list of organizations or the consortium that can create the channels. The orderer service can configure the orderer nodes in three different ways, i.e. Solo, Kafka, and Raft.

13.1.7.1 Solo

Solo was the default configuration with just one orderer node, ideal for development and testing, but it was deprecated in version 2.x.

13.1.7.2 Apache Kafka

Kafka was introduced with the Fabric version 1.0 and was supported till version 2.x, post which, it was deprecated as well for its associated difficulties and need for high expertise in administration.

13.1.7.3 Raft

Raft is the latest addition that follows a leader and follower model to choose a leader node per channel to take the decisions that would be replicated to the other nodes. It's much simpler and easier to work with and needs less expertise than Kafka.

13.1.8 Smart Contract (Chaincode)

Just like any other Blockchain or DLT, Fabric too has its own smart contract that defines the rules against which the transactions are validated before getting endorsed and committed to a ledger. Hence, every smart contract is associated with an endorsement policy that finds out the organization that must approve the transactions before they are considered valid.

Once the smart contract is packaged, it is called Chaincode that has to be deployed to the Blockchain network.

13.2 Consensus and Fabric End-To-End Transaction Flow

The consensus process of Hyperledger Fabric consists of the following components:

- Client SDK
- Endorsing Peers
- Committing Peers
- Orderer(s)

Now, let's discuss the consensus process in Fabric that enables the peers to collectively agree on the finality of a transaction. The process can be divided into three different phases, i.e., proposal and endorsement, ordering, and validation and commit that occurs in a chronological order, as shown in [Figure 13.3](#), as follows:

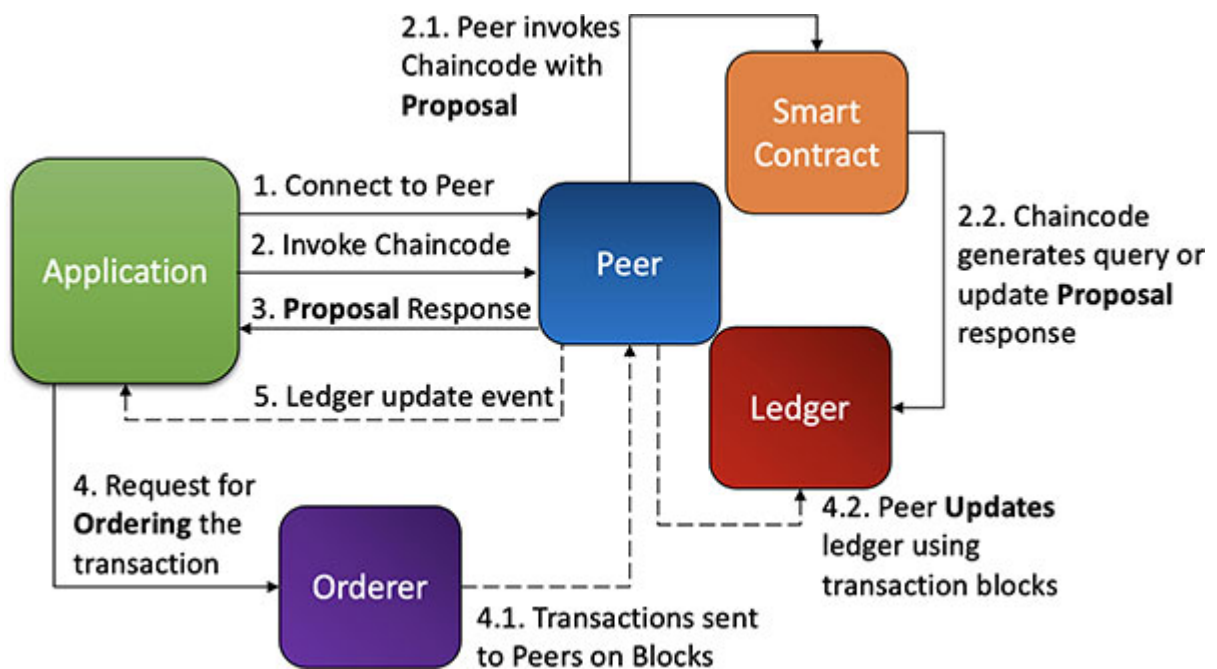


Figure 13.3: Consensus Process in Hyperledger Fabric

13.2.1 Stage 1 Proposal and Endorsement

Once a client submits a transaction proposal, it's sent to one or more endorsing peers who check and sign the transaction with their own digital signature. These endorsed transactions are now sent back to the client.

13.2.2 Stage 2 Ordering

In this phase, the endorsed transactions in stage – I are sent to the orderer(s) from many different applications running concurrently. The orderer collects all the endorsed transactions, orders them ascendingly, and packages them into blocks, where the maximum transactions per block can be adjusted through configuration. These blocks can be saved to the orderer's ledger, and later, can be distributed to all the peers associated to the channel.

13.2.3 Stage 3 Validation & Commit

In this phase, the blocks are actually relayed to all the associated peers. The peers then can validate these transactions themselves and commit these transactions to their respective ledger, and finally, they can reach a common consensus.

Ideally, an application connects to one or more endorsing peers within a channel with a transaction. The peers invoke a smart contract associated with that peer, get the related ledger updated, and send a response back to the application.

Please note that this consensus part is very important for the developers, especially to understand how to club every piece together for integration. [Figure 13.4](#) represents the entire end-to-end ecosystem of Hyperledger Fabric along with the other layers that invoke it, as shown as follows:

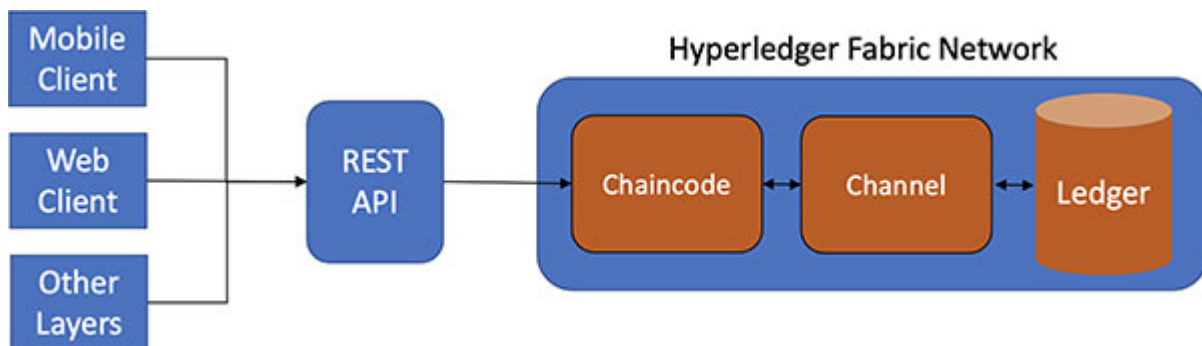


Figure 13.4: Hyperledger Fabric in integration with other layers

13.3 Private Data

Along with Channels, Fabric has another feature called “Private Data” or “Private Data Collections” that facilitates the organizations to share information on peer-to-peer basis using the gossip protocol. But then, why is there a necessity of a private data, as we already had channels, because a channel has the administration overhead of the versions, policies, and MSP. As shown in [Figure 13.5](#), the private data set up is relatively simple and is not visible to the ordering service. Here, the data resides only on the ledgers of the peers on which it is shared and only the hash of the data is sent to the ledger of the peers in that channel. The hash is a proof of the transaction happening off-ledger. Refer to [Figure 13.5](#), as follows:

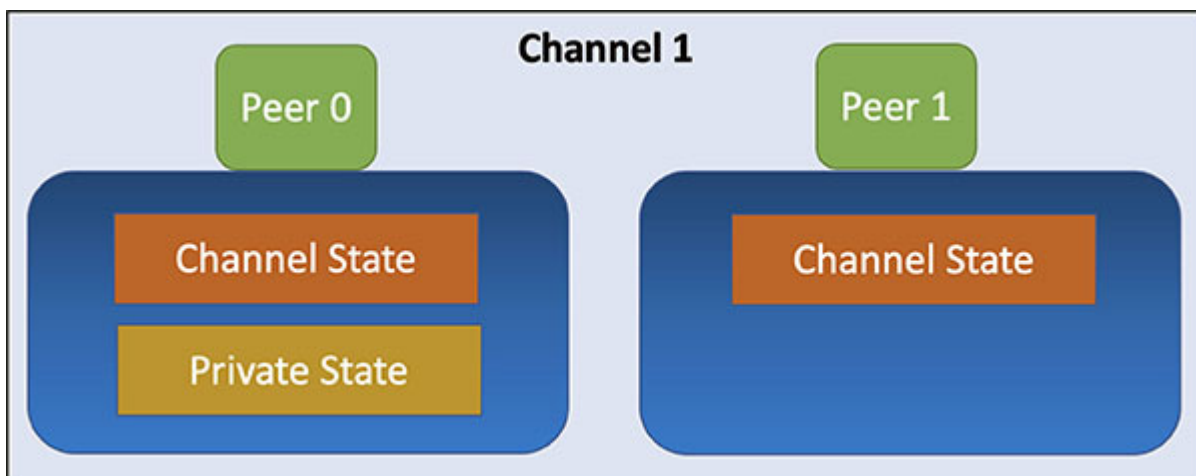


Figure 13.5: Hyperledger Fabric Private Data

Please note that, unlike the data shared on the ledger through the ordering service, the private data does not come with the maintenance of historical data.

13.4 Interoperability with Oracles

Sometimes, our smart contracts need certain volatile information from the outside world in order to execute the transactions, for example, the current exchange rate of USD/INR or the spot price of gold or a particular stock. For such information, the smart contract needs the Oracle services, as shown in [Figure 13.6](#). The latter can connect to the external API and help the smart contract with the information it seeks in a neutral way. Refer to [Figure 13.6](#) as follows:

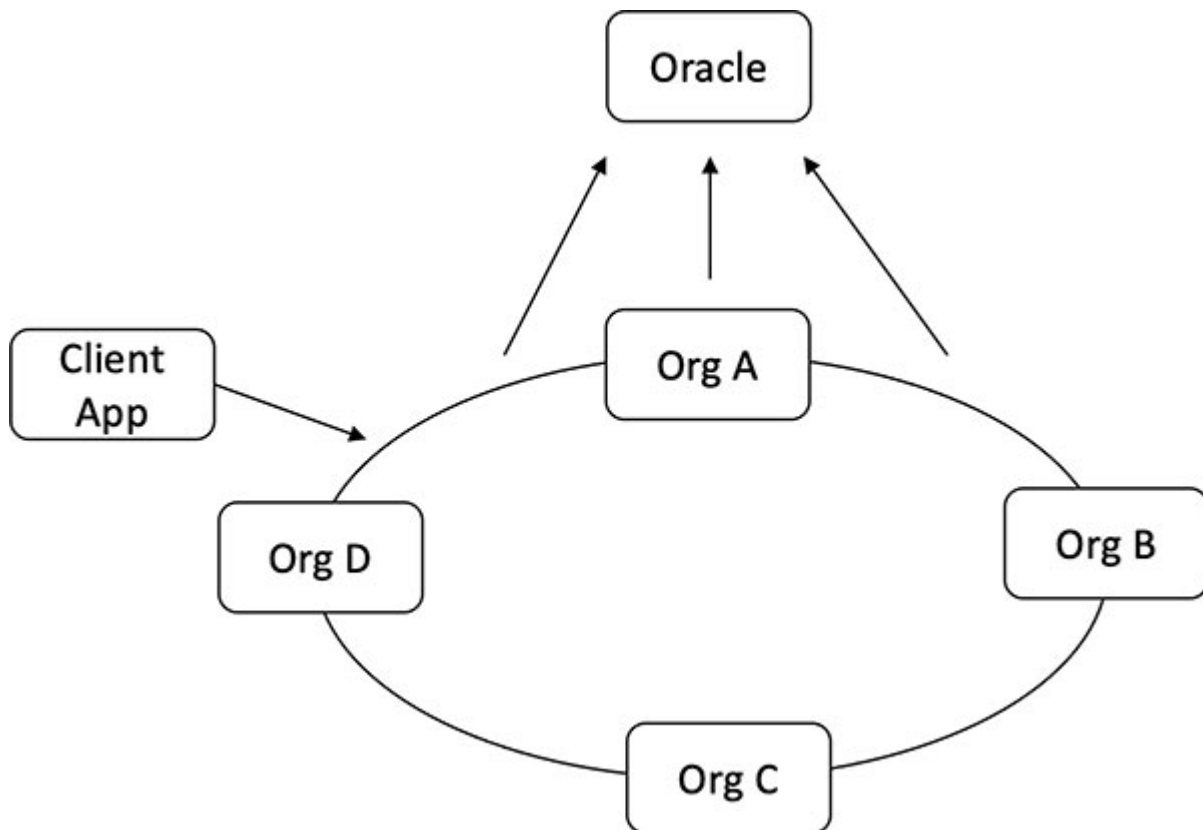


Figure 13.6: Interoperability on Hyperledger Fabric with Oracle Services

Please refer to <https://developer.ibm.com/articles/cl-extend-blockchain-smart-contracts-trusted-oracle/> for further support in the configuration.

13.5 Performance and Scalability

Hyperledger Fabric can be fast and highly scalable if configured properly. The Hyperledger family also comes with a benchmark tool called Caliper to measure the performance of many Hyperledger Blockchain protocols, including Fabric.

13.6 Industry Adoption

Hyperledger is widely used in the industry at different capacities. In fact, out of the three major players in DLT, Fabric finds the largest share. You can refer to the Hyperledger Fabric's website to go through their complete list of major projects in production,

<https://www.hyperledger.org/learn/case-studies>, which gets regularly updated.

However, because Hyperledger Fabric is completely opensource, one needs to seek partnership with organizations like IBM who are the major supporters of Hyperledger Fabric, and hence, have more experience in handling the projects in production.

13.7 Development

The Hyperledger Fabric smart contracts can be written either in Go, Node.js, or Java and some examples can be found in Python. One can refer to the Fabric's website https://hyperledger-fabric.readthedocs.io/en/release-2.2/developapps/developing_applications.html for further guidance in development and running in different environments.

13.8 Deployment on Cloud

Hyperledger Fabric, for its high popularity, has support on most of the major cloud platforms. The clients also prefer to get their solutions deployed in production to be hosted on cloud, rather than on-premise. The major players in this space are AWS, Azure, and Oracle.

13.9 Summary

In this chapter, we covered the following topics:

- Key concepts of Hyperledger Fabric with end-to-end flow of a transaction.
- Private data collection for facilitating organizations to share information on peer-to-peer basis.
- Interoperability of Hyperledger Fabric with external APIs with the help of Oracles.

References

- A Blockchain Platform for the Enterprise Hyperledger - <https://hyperledger-fabric.readthedocs.io/en/release-2.2/> and <https://hyperledger-fabric.readthedocs.io/en/release-2.3/>
- Hyperledger Architecture Volume I - https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf
- Oracles: Common architectural patterns for Hyperledger Fabric - <https://developer.ibm.com/articles/oracles-common-architectural-patterns-for-fabric/>

CHAPTER 14

R3 Corda

“Gartner’s survey saw the usage of Corda at 15% of the reported use cases in 2018... but rise to over 40% of use cases in 2020, overtaking Hyperledger Fabric and Ethereum. These growth point may be linked to the release of Corda Enterprise by R3, the owner of Corda, in 2018 and 2019. It is also clear, though, that Corda now penetrates well beyond its original financial services origin” – Gartner

Version 1.0 Launched: October 2017

Founder: R3

Current Version: 5

In 2014, R3 introduced Corda, a privacy oriented DLT that eventually became a main competitor to Hyperledger Fabric. Corda initially came only in the open source version but was soon introduced in the enterprise version with standard features as support, documentation, and certification. In this chapter, we will cover the key concepts of R3 Corda and also focus on the privacy and security features that make it suitable, especially for banking, fintech, and insurance industries, and others.

14.1 Key Concepts

Let’s first cover some of the key concepts of R3 Corda.

14.1.1 Network

In R3 Corda, each node is represented by an organization with a unique IP address. Nodes can discover each other through the network map service. They can communicate with each other on peer-to-peer basis where all the messages are encrypted using transport layer security. A node can share the data with one or more

nodes or all nodes depending on the requirement of the use case and there is no global broadcasting.

14.1.2 Ledger

In R3 Corda, each node has its separate storage known as the “**Vault**” that contains only the information relevant to that node, as shown in [Figure 14.1](#) as follows:

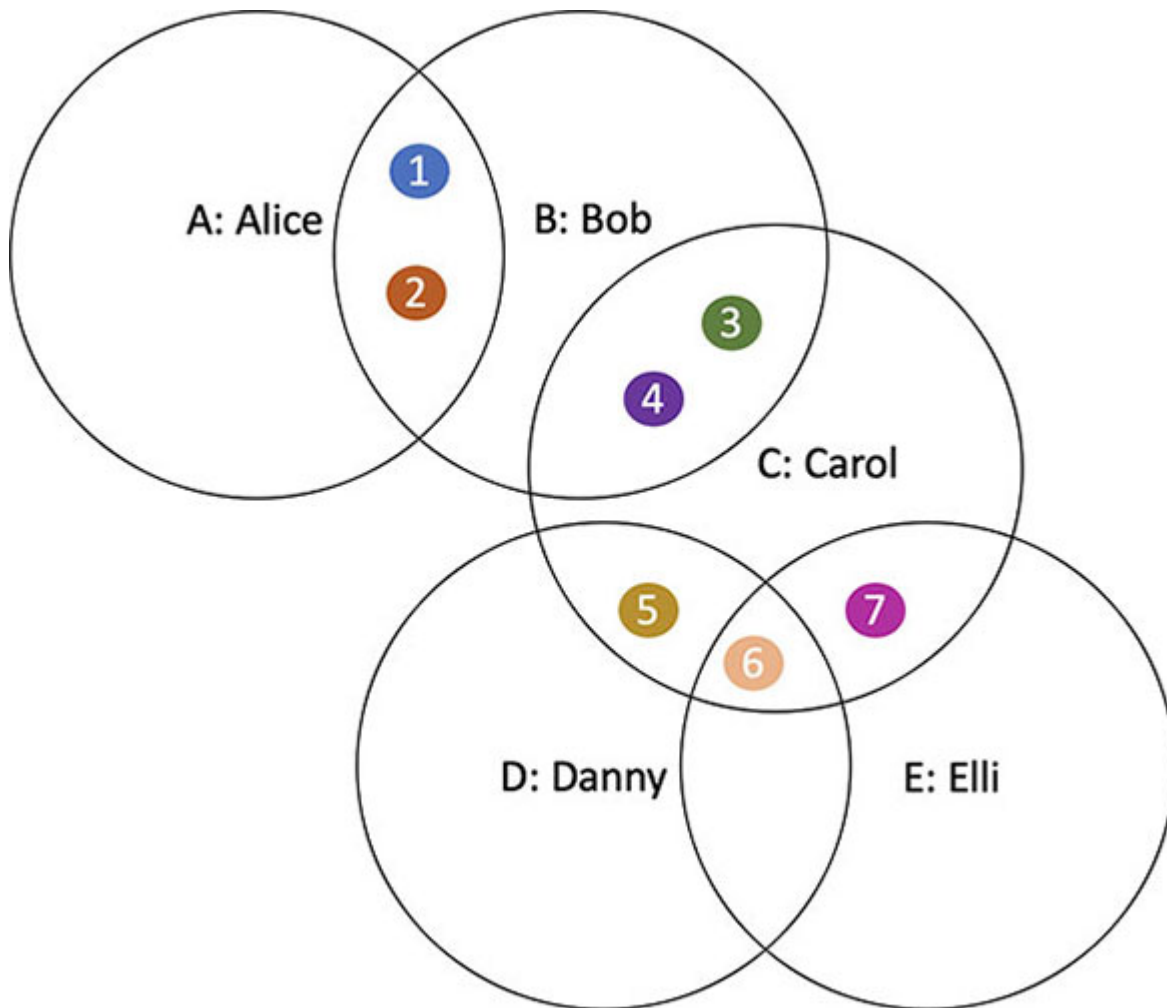


Figure 14.1: R3 Corda Vaults

For example, if A, B, C, D, and E are 5 nodes and each are engaged in different transactions with each other, then each node would have the information only on the transactions they are associated with and would have no knowledge of the ones where they are not involved. Hence, each node should have different information that are related

to only the transactions that the owner of the node is associated with, as shown in [Figure 14.2](#), as follows:

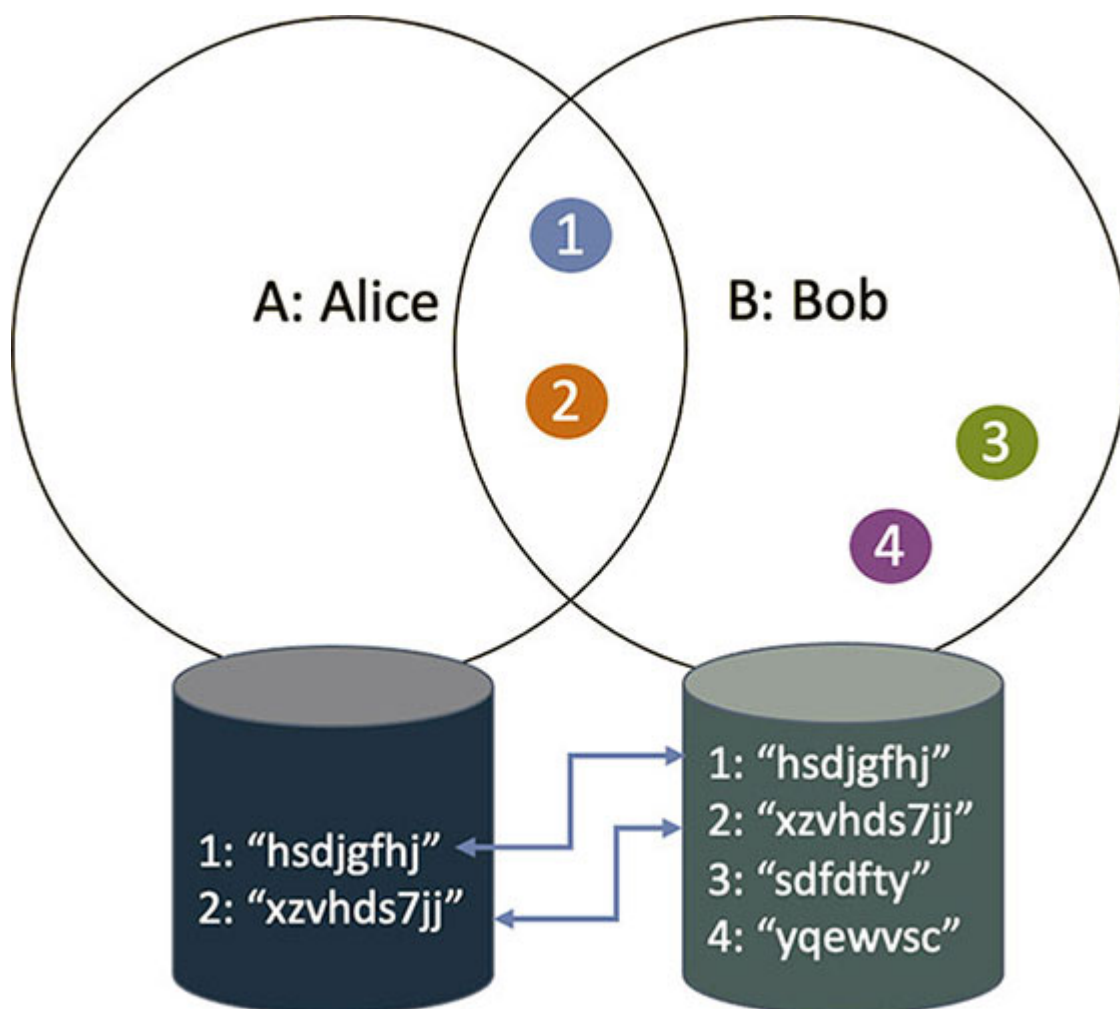


Figure 14.2: Only relevant data of node saved to vault

14.1.3 State

A state, as shown in [Figure 14.3](#), is an object that we save in the vault and whose status is immutable, as follows:

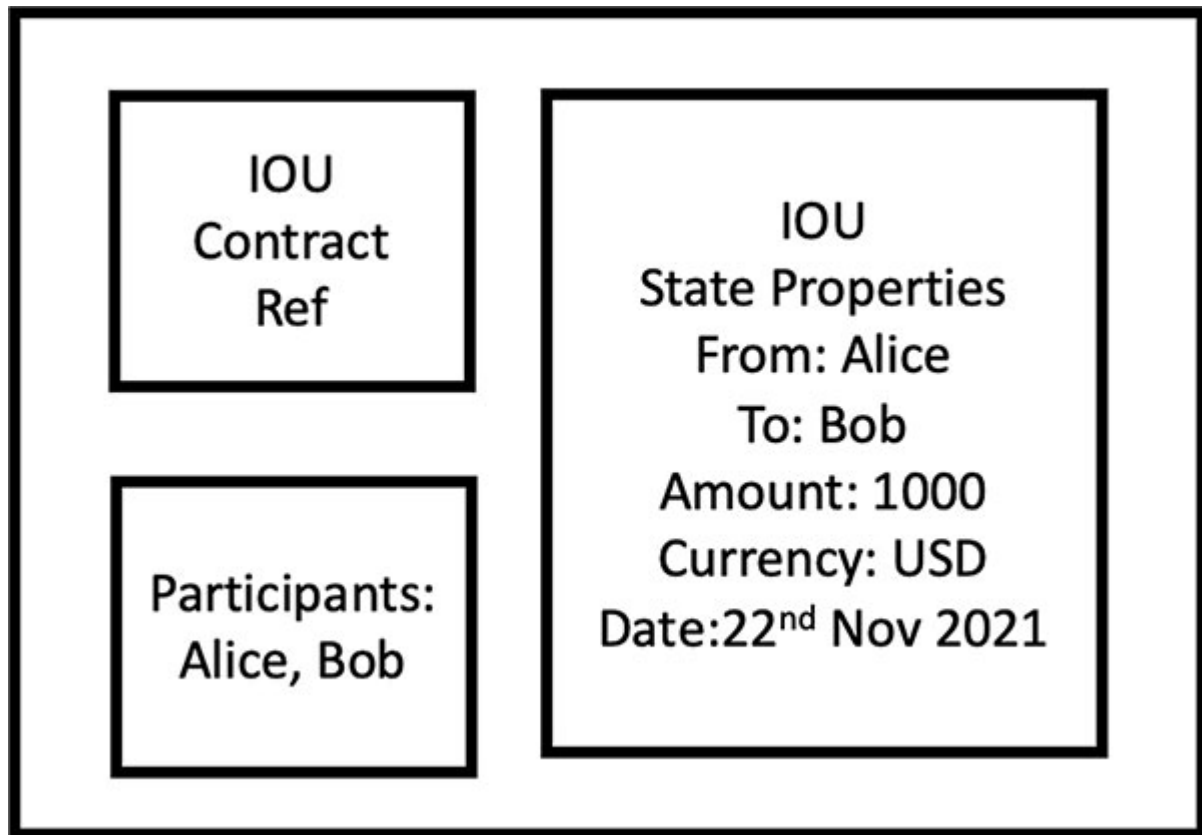


Figure 14.3: Corda State

The state is like a row in a table with a combination of data of many different types, i.e., string, integer, double, float, date etc. As the data in DLT is immutable, every time there is a need of update or even delete, a new version of the state is added to the ledger that automatically makes the previous version as the consumed historical data, as illustrated in [Figure 14.4](#) as follows:

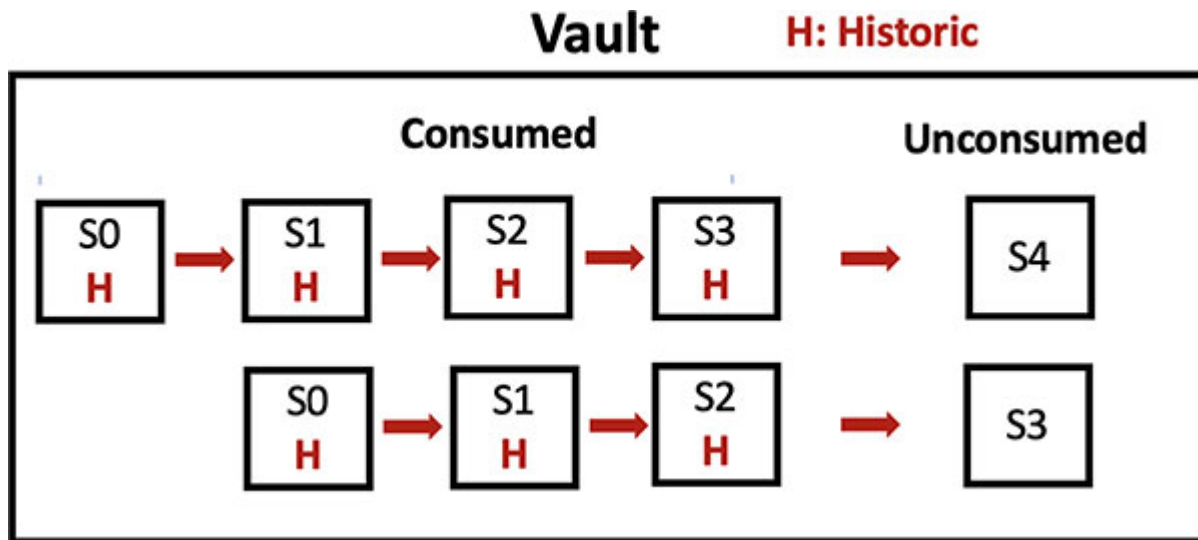


Figure 14.4: R3 Corda Historical Data

In Corda, all the consumed historical data are saved in the vault, including the current unconsumed version.

14.1.4 Transactions

Each transaction is a request to the ledger to update the status of zero or more states as historic and, at the same time, introduce one or more new unspent states as the output, as shown in [Figure 14.5](#), as follows:

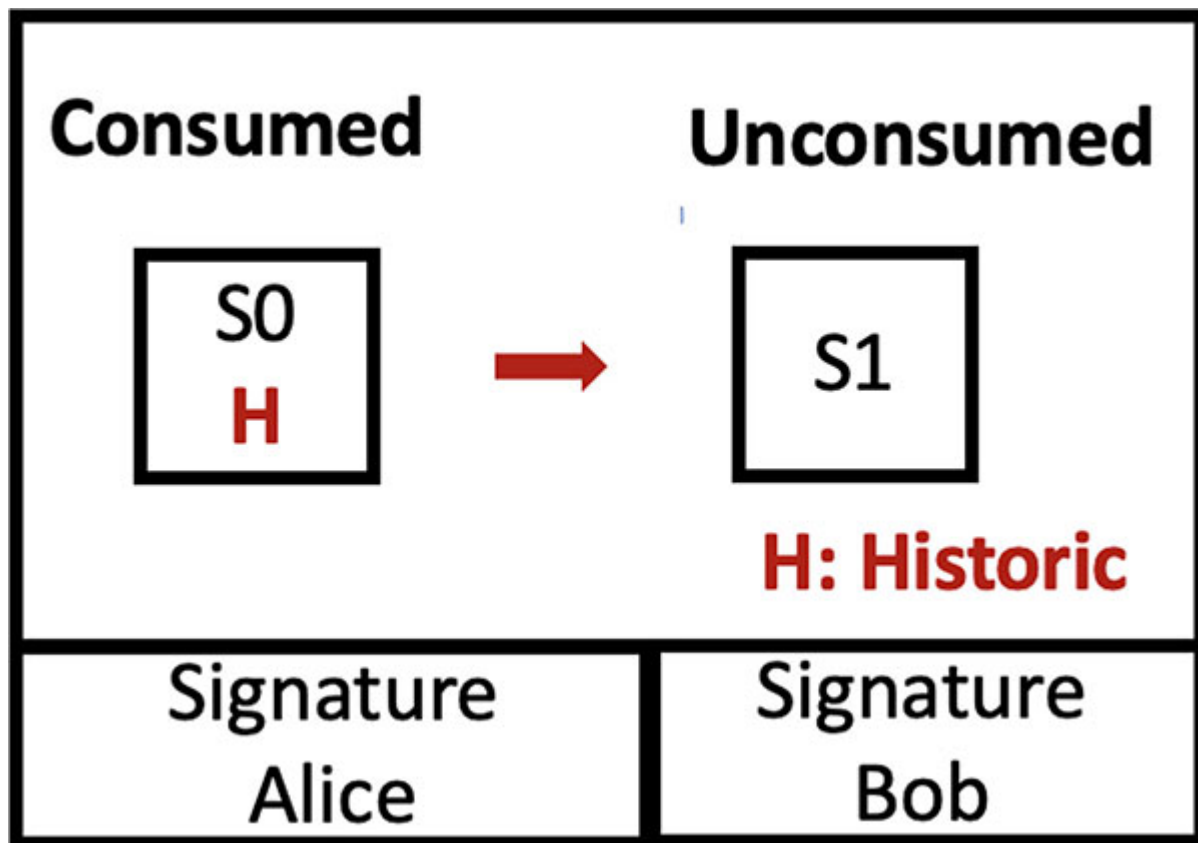


Figure 14.5: *Change of State in a Transaction in Corda*

As shown in [Figure 14.6](#), in each transaction, the digital signature of each participating party would be collected that are automatically handled by successful handling of smart contracts. Once the transaction is executed, the previous state would be saved as the historical data. Refer to [Figure 14.6](#), as follows:

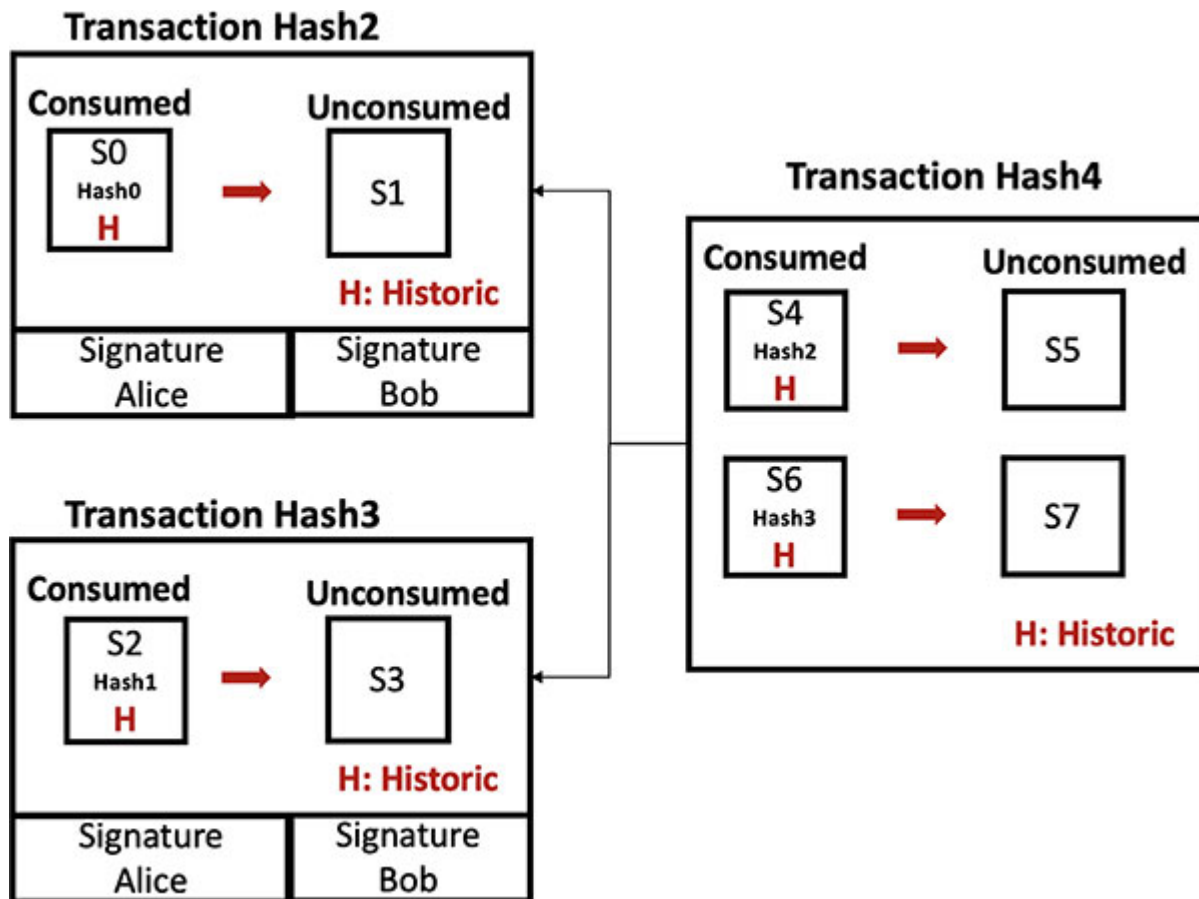


Figure 14.6: Corda Transactions

(Source: <https://docs.corda.net/docs/corda-os/4.7/key-concepts-transactions.html>)

R3 Corda has no blocks; hence, it is ideally not a Blockchain. However, it comes with all the goodness of Blockchain sans its complexity of configuration and maintenance. Also, unlike the public Blockchains, where the blocks are connected together through hashes, in Corda, the connections happen at the transaction level, i.e., the hash of all the previous transactions goes to the state of the next transaction. Thus, they are stitched together.

The transactions have many important concepts as Notary, Time Window, and Attachments.

14.1.5 Notary

Each Corda network may have one or more independent notaries which are neutral nodes used for finalizing a transaction. Each

transaction, before getting executed, gets validated by the selected notary that checks all the inputs for any possible double spends.

Many Blockchain enthusiasts argue that Notary brings centralization to Corda's architecture; however, please note that in order to avoid a single point of failure, we can introduce multiple notaries that can help us in load balancing and low latency.

Please note that a Notary can be configured to be of validating or non-validating type. In case of the validating type, the notary has to validate the entire input data of the transaction, exposing the privacy of the data to a third party. Most notary related configurations, however, are of the non-validating type where it is used just to ensure there are no double spends.

14.1.6 Time Window

With this feature, a Corda transaction can be executed with a time constraint, i.e., before or after a particular pre-determined time. It is highly helpful in time based contract execution.

14.1.7 Attachments

Corda also supports uploading attachment files like images, PDF etc., in the form of Zip or JAR. This is especially helpful when the state is associated with an official document, an image etc.

14.1.8 Contracts

In Corda, smart contracts can be written in Kotlin or Java and it can make sure that the transaction is executed after the validation checks. Each transaction can have zero or more inputs and outputs.

14.1.9 Interoperability with Oracles

In Corda, interoperability can be achieved by the use of Oracle services implemented on the Oracle nodes. Some contracts may need information from some external source before validating a transaction. For example, the exchange price of the bond should be more, equal, or less than the current price on a particular stock

exchange. This current rate is found by sending the request to an independent and neutral Oracle node and through the Oracle services. The Oracle node connects to the external API endpoint and fetches the rate for us and signs it to assure that the rate is true.

14.1.10 Consensus

Unlike in public Blockchains or even Hyperledger Fabric, Corda does not need a universal acceptance for a transaction to be validated. Only the parties participating in the transaction validate it by signing the transaction with their respective keys. Also, the notary checks and prohibits any double spends in the transactions.

14.2 CordApp

As shown in [Figure 14.7](#), Corda distributed application or CordApp is crafted to enable Corda to run on multiple nodes in parallel and reach a consensus to update their respective ledgers, as follows:

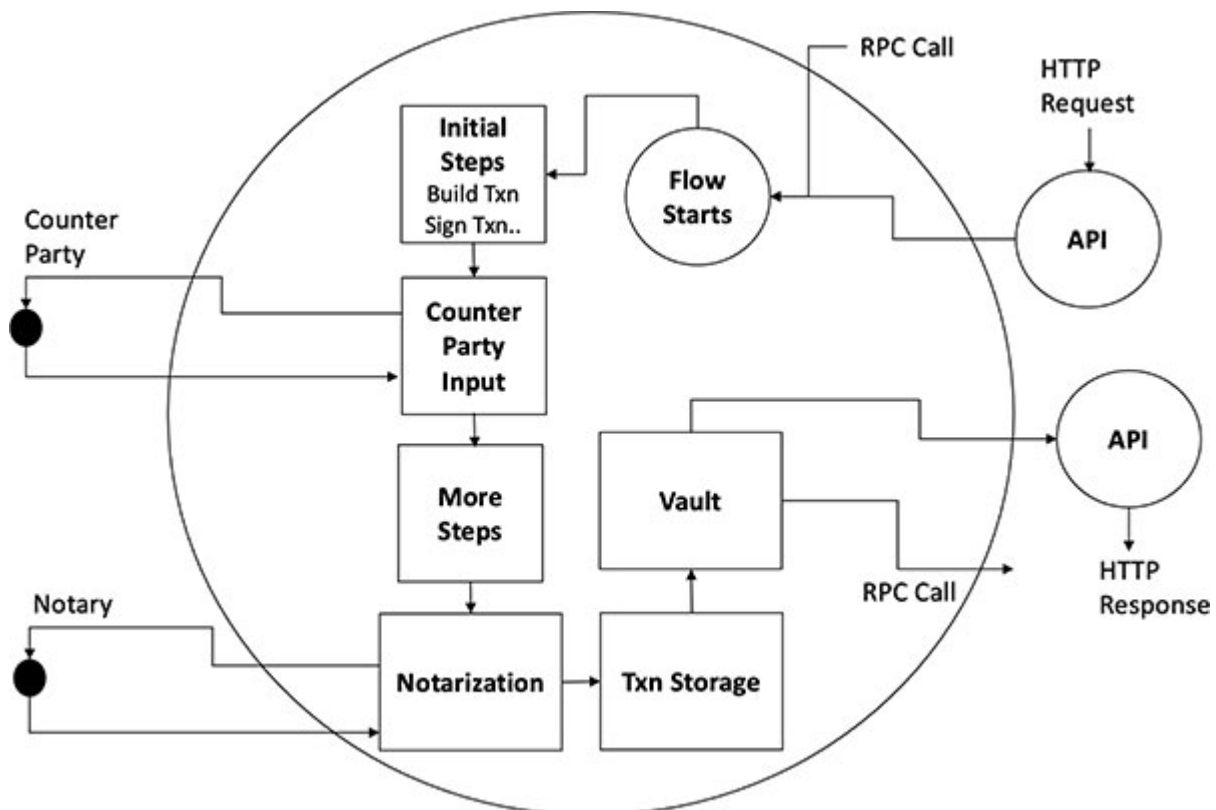


Figure 14.7: CordApp

(Source: <https://docs.corda.net/docs/corda-os/4.7/cordapp-overview.html>)

Let's find out how that can be achieved. In Corda, each node can be accessed either using an RPC call directly or by calling it through an HTTP request. Hence, each organization can run their REST endpoints on a different URL with the help of a Spring Boot server that internally invokes the corresponding node with an RPC call. Once the request reaches the node, it can validate the data in the request with smart contract, sign the transaction, share it with counter party, and get their sign off, and finally get it validated by the notary. If all goes well, the data can be stored on the respective node's vaults and the transactions get committed. The response can be sent back either through the RPC call response or as an HTTP response. Hence, it's pretty simple to get the Corda nodes tested as well as give a demo to the clients, showing them the actual changes happening at the node level.

14.3 Corda End-To-End Transaction Flow

As shown in [Figure 14.8](#), Corda flows are business processes involving multiple point-to-point communications, Transaction requests, data sharing, contract validations, notarization and ledger updates, as follows:

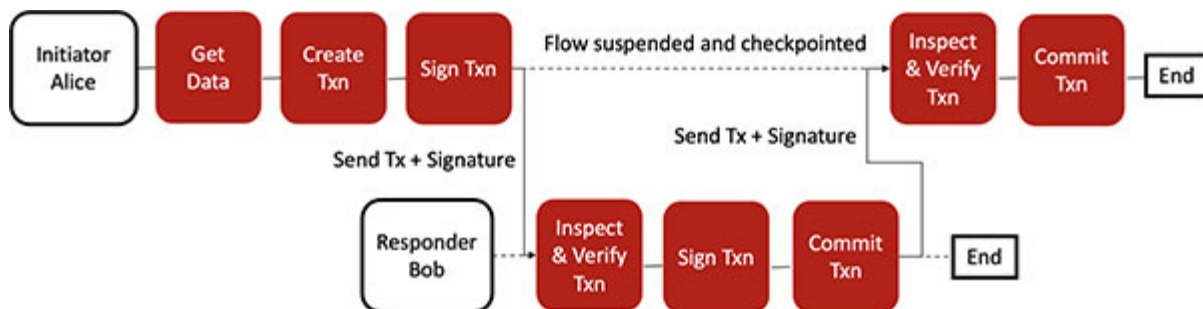


Figure 14.8: Corda Transaction flow between parties

One flow may contain zero or more subflows where each subflow is responsible for handling a particular subprocess.

Note: Unlike Fabric, there is no global broadcasting in Corda. All data sharing are point-to-point and happens only when explicitly done on need basis.

14.4 Corda Accounts Model

Corda introduced the Accounts library from version 4.3 that enables the Corda vault to be logically divided into many smaller sub-vaults, as shown in [Figure 14.9](#). This is similar to having multiple peers in Hyperledger Fabric under one organization. Refer to [Figure 14.9](#), as follows:

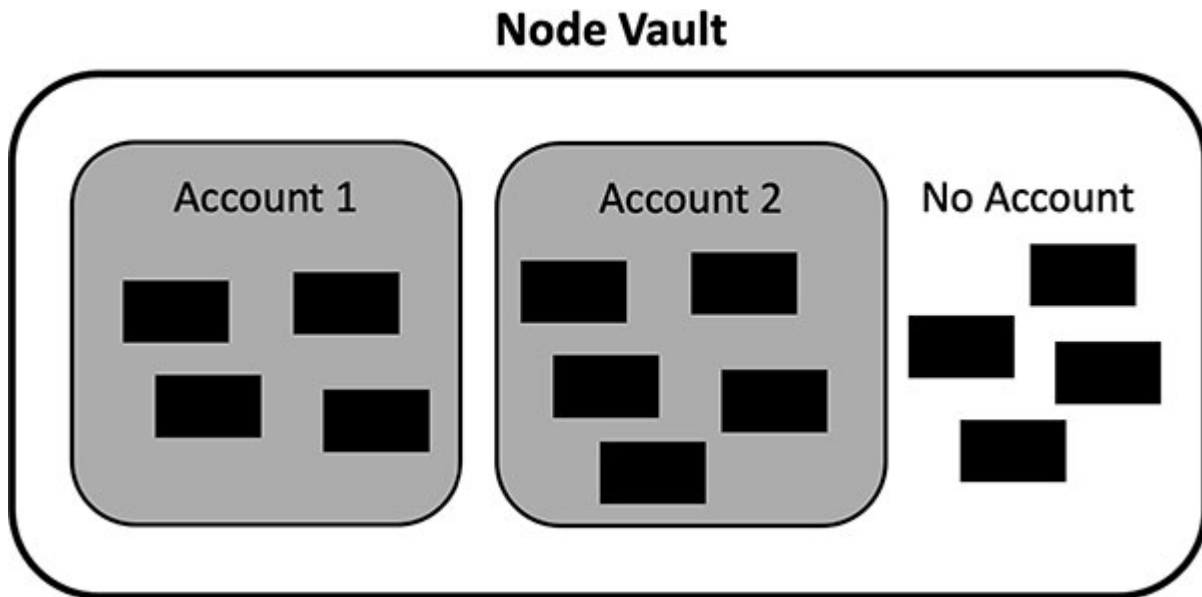


Figure 14.9: Corda Node states inside and outside accounts

As shown in [Figure 14.10](#), this new feature enables Corda states to be owned either directly by a node or aligned to any of the accounts within a node. For example, a software organization can hire many employees who are initially on bench. We can refer them to the employees of organization A. After they are allocated to different projects, we can refer them as employees of organization A within project A1, A2 or A3. Refer to [Figure 14.10](#), as follows:

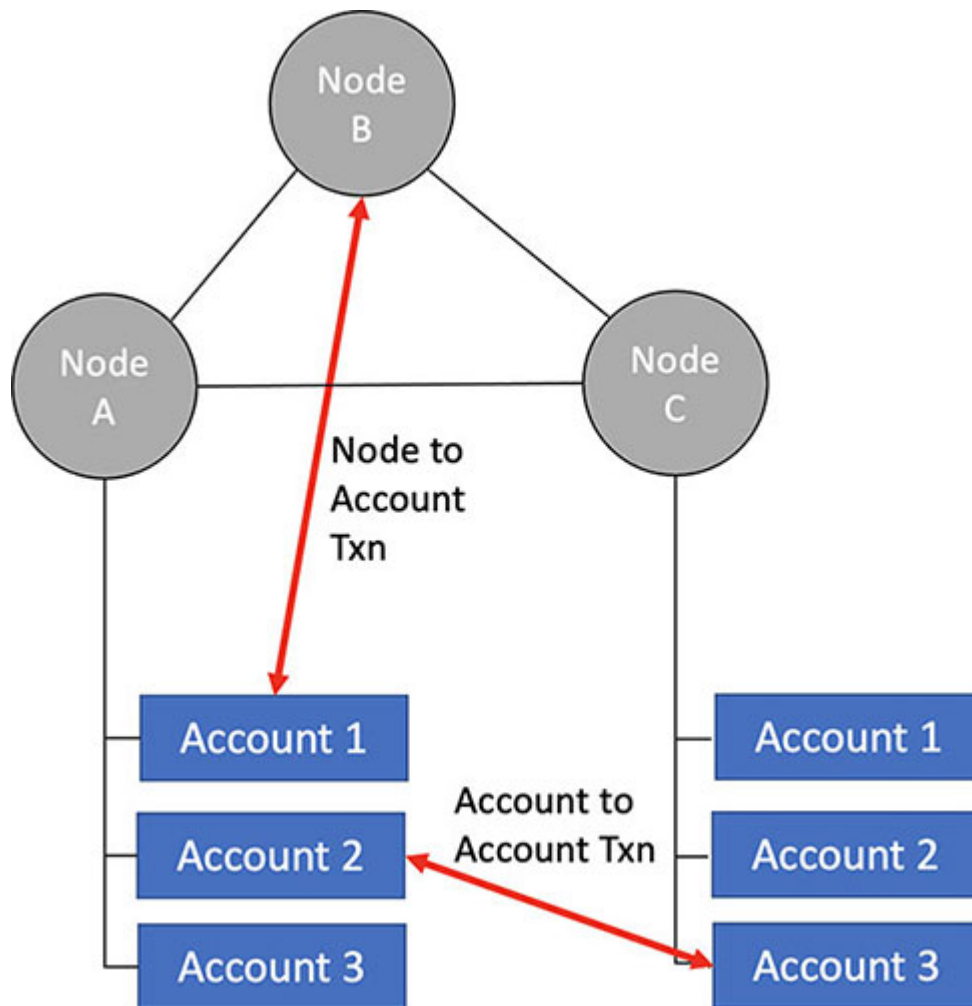


Figure 14.10: Tri-Node Business Network

Accounts model reduces the cost of hosting multiple entities related to one organization by allocating them one common node. In [Figure 14.11](#), you can see the flow is going back and forth between different parties' accounts in a Supply-Chain ecosystem, as follows:

Business Flow:

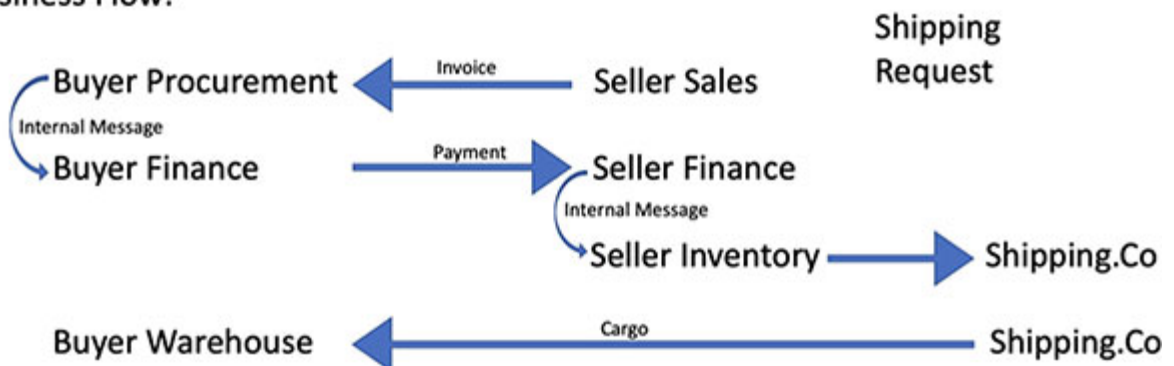


Figure 14.11: *Accounts model in Supply-Chain project*
(Source: <https://github.com/corda/accounts-demo-supplychain>)

You can refer to this Supply-Chain project on GitHub to understand how Accounts model can be effortlessly implemented for a supply-chain ecosystem.

Accounts model is also hugely helpful for many different business models in the banking and financial sectors. Especially, “Central Bank Digital Currency” or CBDC implementation can be greatly benefitted by the use of Corda’s accounts library.

14.5 Token SDK

R3 Corda has very well developed libraries for creating and maintaining both fungible and non-fungible tokens. So, the customers can represent any of their assets as non-fungible tokens and can sell it on the ledger after the exchange of money. The libraries give us an option to define our own token, issue it, update the ownership, or even redeem the token. If needed, the history of the state of the tokens can be retrieved from Corda’s vault.

Corda Token SDK can also be used in CBDC projects to create coins with money like features that are not necessarily associated with any particular account and can be independently traded with the other assets, just like cash.

14.6 Corda Network

Corda also comes with its public permissioned version running on the Internet, known as “Corda network”. This network run by not-for-profit Corda Network Foundation can onboard certified nodes of different organizations that can exchange cash, digital assets, bonds, identity etc., on the ledger in an interoperable way. With the Corda network, R3 has managed to introduce a publicly available infrastructure, using which organizations can build super-quick decentralized applications that can eliminate operational complexity. Corda network also comes with 24/7 assistance and support to onboard new organizations and help them to operate.

14.7 Corda Opensource vs. Corda Enterprise

The opensource and enterprise versions of Corda are 100% compatible and interoperable, which means that in order to save the cost, the developers can download the opensource version and start learning all the features with loads of examples available on GitHub. However, when moving to production, it's justified to adopt the enterprise version. [Table 14.1](#) is a comparison chart showing the differentiators that prompt the organizations for Corda enterprise adoption, as follows:

Features	Corda Opensource	Corda Enterprise
Database support	H2, Postgres, SQL Server	H2, Postgres, SQL Server, Oracle
Multiple nodes for high availability/disaster recovery	No	Yes
External Artemis MQ	No	Yes
Corda firewall and External Artemis MQ with multi-node sharing	No	Yes
HSM support for key storage	No	Yes
Oracle RAC connectivity and clustered notary	No	Yes
Tool for node health check, Configuration obfuscation and HA administration	No	Yes
Software maintenance and support	No	Yes

Table 14.1: Comparison of R3 Corda Opensource and Enterprise versions

14.8 Performance and Scalability

R3 Corda, being a private permissioned DLT, is highly scalable and its performance and scalability benchmarking results are regularly updated on their website, <https://docs.r3.com/en/platform/corda/4.6/enterprise/performance-testing/performance-results.html>.

14.9 Industry Adoption

Corda, for its simple and scalable DLT platform, has hundreds of clients across the globe, including governments, major banks, insurance companies, supply chain and trade finance organizations, healthcare companies etc., the list for which you can find on their website <https://www.r3.com/customers/>. Though initially R3 Corda focused only on the BFSI sector, today it is a platform of choice for many other business verticals.

14.10 Development

The best place to start the development of Corda is its website for the developers <https://docs.r3.com/>. It also comes with a huge amount of samples (<https://docs.r3.com/en/samples.html>) both in Java and Kotlin that keeps on getting regular updates from time to time.

14.11 Deployment on Cloud

Just like Hyperledger Fabric, R3 Corda too has support on most of the major cloud platforms as Azure, AWS, and Google cloud.

14.12 Summary

In this chapter, we covered the following topics:

- Key concepts of R3 Corda with end-to-end flow of a transaction
- Corda's Accounts and Token libraries
- Corda Network
- Corda open source for development and Corda enterprise for production
- Corda's industry adoption

References

- Welcome to Corda - <https://docs.corda.net/docs/corda-os/4.8.html>

- CordApp examples - <https://www.corda.net/samples/>

CHAPTER 15

Consensys Quorum

“The goal of Proof of Stake is to be the most efficient way to keep a public blockchain validated, not to maximize the rewards for a specific use case.” — Vitalik Buterin

Version 1.0 Launched: November 2016

Founder: JPMorgan

Current Version: 21.10.0

Ethereum was the first public Blockchain platform for building decentralized applications on production. When Ethereum was launched in 2015, almost in no time it had gathered momentum leading to millions of transactions in production; yet the market quickly realised the need of an enterprise version of Ethereum that would be ideal for organizations with additional needs of privacy, scalability, governance, deployment etc. Soon, certain organizations such as JP Morgan’s Quorum (modified from go-ethereum), PagaSys’s Besu etc., started working on Ethereum clients on their own to support permissioned deployment. Together in 2017, these organizations launched “Enterprise Ethereum Alliance” or EEA to collectively work on features like privacy, identity, and permission management etc., and also to build, promote, and the support adoption of technology in the market. In 2019, Besu was admitted to the Hyperledger family, and in 2020, Consenys acquired J.P. Morgan’s Quorum to further advance Ethereum’s enterprise Blockchain adoption.

In this chapter, we will cover the detailed architecture of Quorum and learn how it is different from its rivals, Hyperledger Fabric and R3 Corda.

15.1 Key Concept

Quorum is a light-weight fork of go-ethereum (geth), a public Ethereum client that has been improved to match the enterprise needs like scalability, performance, interoperability etc. As already discussed, the current form of Consensys Quorum is the culmination of the work of several different independent projects. Just like Fabric, Quorum can be architected in more than one way, which are as follows:

- Java based Besu
- Go based GoQuorum

[Figure 15.1](#) represents the culmination of Hyperledger Besu and JP Morgan Quorum into one Consensys Quorum platform that supports both the previous versions, as shown as follows:

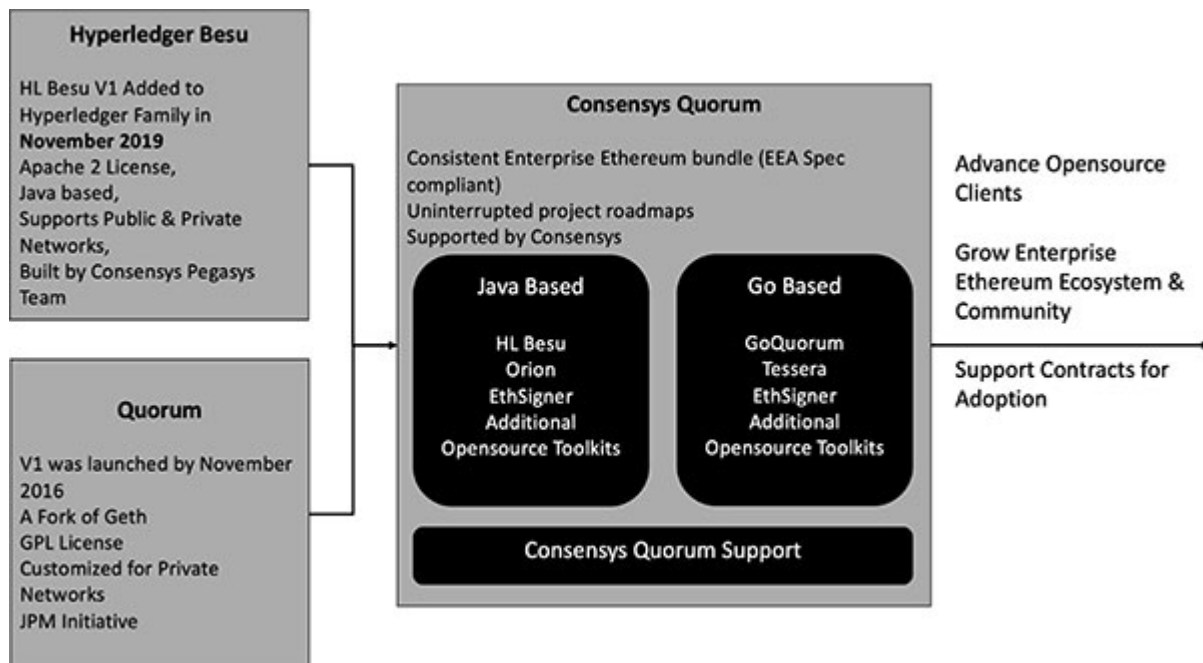


Figure 15.1: Consensys Quorum

As shown in [Figure 15.2](#), though both have slightly different sets of configuration set up, the basic architecture is pretty much alike, as follows:

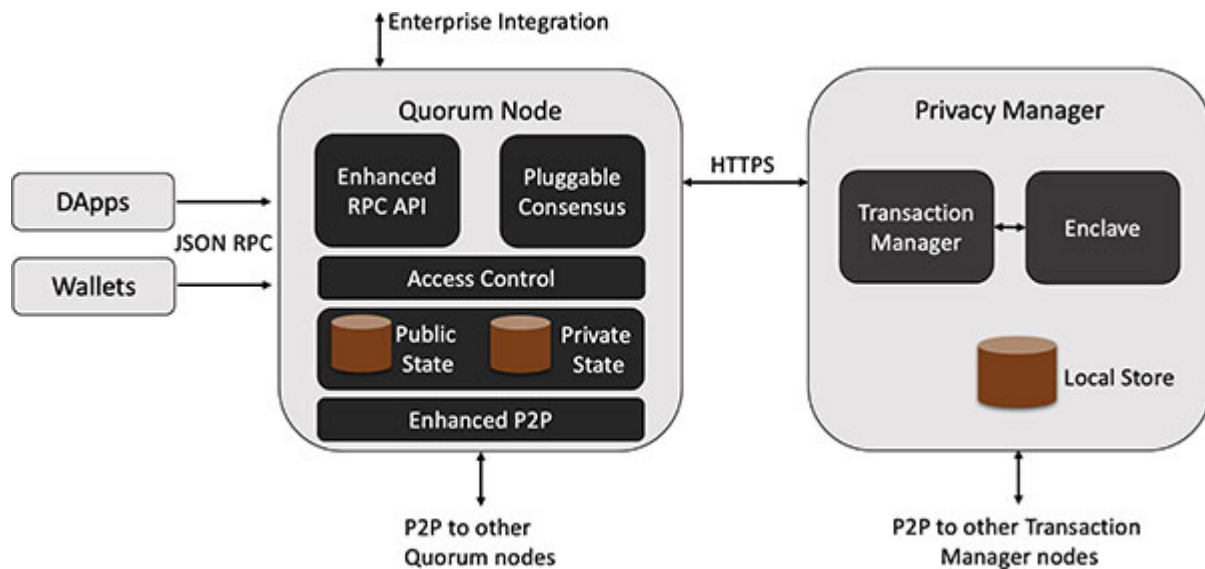


Figure 15.2: Internals of a Quorum Node

Let's explore the inner components of Quorum and understand how it's different from Hyperledger Fabric or R3 Corda.

15.2 Consensus - RAFT

Quorum uses RAFT or Istanbul BFT algorithms for reaching the consensus instead of Ethereum I's PoW or Ethereum II's PoS models.

Also, the pricing part of the gas is removed in Quorum though the gas part is there only for programming.

15.2.1 Privacy Manager

Quorum has the advantages over Fabric or Corda in a way that it can support both the public and the private transactions in the same network. Because Quorum is backed by Ethereum, it can always deal with the public transactions; at the same time, for the private transactions, it relies on an internal component called the "Privacy Manager".

15.2.2 Transaction Manager

The transaction manager is a stateless component of the privacy manager that can be invoked with the REST end points. It's the

responsibility of the transaction manager to receive the transaction payload requests, get it encrypted and stored locally, and also share it with another permissioned Quorum node. Currently, Consensys Quorum supports Java based Tessera for GoQuorum clients or Orion for Besu.

15.2.3 Enclave

Enclave is a storage to manage all the private and public keys for the encryption and requirements of the transaction manager. It also maintains a list of identities of all nodes in the private network.

15.3 Consensus Quorum End to End Transaction Flow

[Figure 15.3](#) is a description of how Private Transactions are processed in GoQuorum where we have a network of nodes, and the transaction happens only between a subset of them, as shown as follows:

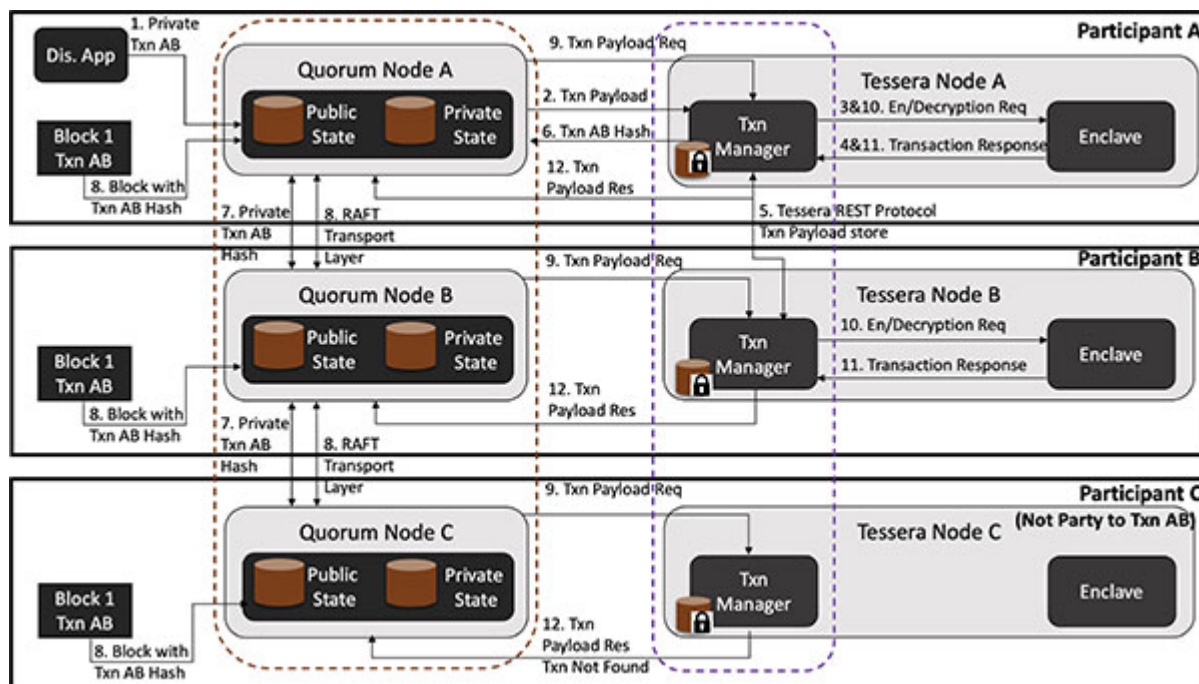


Figure 15.3: Consensys Quorum End-to-end Transaction flow among Quorum nodes
(Source:

<https://docs.goquorum.consensys.net/en/stable/Concepts/Privacy/PrivateTransactionLifecycle/>

- In this scenario, where Org A, Org B, and Org C are working together in a Quorum network that is being represented through their nodes A, B, and C respectively. While Org A wishes to execute a transaction with Org B, it doesn't wish to involve Org C in the same. Now, let's explore how the entire flow of the data would look like.
- Org A now sends a transaction to its own node with the instructions in the associated transaction payload. The payload has a property "privateFor" where it includes the public keys of all the nodes with which the transaction must be shared. In the current scenario, "privateFor" property would have the public keys of Org A and Org B and not Org C. Having its own public key, however, is optional for this property.
- Node A's transaction manager now shares the transaction with its own Enclave. The latter encrypts the transaction payload in such a way that only the rightful owner can decrypt it with its own private key. The encrypted transaction now is stored and shared back to the node A's transaction manager by the Enclave.
- Transaction manager now propagates the transaction in the entire network using P2P protocol. The transaction now packaged in a block reaches every participating node in the network.
- Only the nodes that are the rightful receiver can receive the encrypted transaction through the transaction manager and get it decrypted through the private keys stored in their Enclave.

15.4 Industry Adoption

Quorum has many implementations across business verticals which are in PoC, pilot, or in production stage. The most popular project called "JPM Coin" crafted by JP Morgan was introduced for instant settlement between the JP Morgan's clients from different parts of the globe. Project Ubin is another very popular project by Monetary

Authority of Singapore for clearing and settlement. With clients like ING group, Ant group, Microsoft, Novartis etc., Quorum has a long way to go.

15.5 Development

In their website <https://consensys.net/quorum/developers/>, Consensys has come up with good documentation in order to develop projects using Quorum. The developers have the option of choosing from Hyperledger Besu or GoQuorum as per their expertise or partnership network for support.

15.6 Deployment on Cloud

Quorum can be deployed in production using Azure VM, either with Hyperledger Besu or GoQuorum. Besu can be used with public Ethereum, and as it's completely interoperable with the latter, it can also be used as a private permissioned DLT. Both of them are open source.

15.7 Summary

In this chapter, we covered the following topics:

- Evolution of Quorum from JP Morgan and Hyperledger Besu to Consensys Quorum.
- Key concepts of Quorum with end-to-end flow of a transaction.
- Industry adoption of Quorum.

References

- Enterprise Ethereum - <https://docs.kaleido.io/kaleido-platform/protocol/ethereum/>
- ConsenSys Acquires J.P. Morgan's Quorum to Advance Enterprise Blockchain Adoption - <https://consensys.net/blog/news/consensys-acquires-jpm-quorum/>

- GoQuorum Private Transaction Life Cycle -
<https://docs.goquorum.consensys.net/en/stable/Concepts/Privacy/PrivateTransactionLifecycle/>
- Quorum for developers, Consensys -
<https://consensys.net/quorum/developers/>
- Getting started with Consensys Quorum -
<https://consensys.net/quorum/products/guides/getting-started-with-consensys-quorum/>
- Hyperledger Besu Enterprise Ethereum Client -
<https://besu.hyperledger.org/en/stable/>
- Quorum Dev Quickstart -
<https://azuremarketplace.microsoft.com/en-us/marketplace/apps/consensys.quorum-dev-quickstart?tab=overview>
- Releases Consensys Quorum GitHub -
<https://github.com/ConsenSys/quorum/releases>

CHAPTER 16

Comparison of Hyperledger Fabric, R3 Corda, and Consensus Quorum

Now that we have learnt about the three major players of enterprise DLT, it's time to compare their features in [Table 16.1](#) that would help the readers select the ideal platform for their requirements, as follows:

	Hyperledger Fabric	R3 Corda	Consensus Quorum
Opensource or Enterprise	Fabric has just one version that can be configured for both Opensource or Enterprise	Separate versions for Opensource and Enterprise	One version that can be configured for Opensource or Enterprise
Languages supported	Go, NodeJS, Python, Java for Smart Contract	Kotlin, Java for Smart Contract	Solidity for Smart Contract
Message propagation	Broadcast messages within the channel for public messages, Gossip network for sharing private messages	Point-to-point messaging in a private network for all messages	Point-to-point messaging only to permissioned nodes in a public network for all messages
Security Zones for sharing data with selected peers with privacy	Channels	Not needed as all message propagations are point-to-point	Not needed as all message propagations are point-to-point

Data storage	LevelDB, CouchDB	H2, Postgres, SQL Server, Oracle	Tessera supports H2, can support others with additional configuration
Consensus	RAFT, a CFT consensus algorithm or Istanbul Byzantine Fault Tolerance — a BFT consensus algorithm	Supported by Notary, contractual validation and agreement of all participating nodes	Pluggable with RAFT or BFT or others
Scalability	Highly scalable	Highly scalable	Highly scalable
Interoperability	Possible, however configuration is complex	Possible through Corda Settler and Oracle node. Configuration is simple	Possible through Oracles
Support	Open source but promoted and supported by IBM commercially	R3 Corda team	Consensys team
Configuration and Set up	Relatively tough	Relatively simple	Relatively tough
Documentation	Well-documented and huge number of examples on GitHub	Very well documented and huge number of examples on GitHub	Relatively less documented and lack of examples on GitHub
Resource & learning curve	Learning curve high but high availability of resources	Learning curve low and high availability of resources	Learning curve high but high availability of resources

Table 16.1: Comparison of private permissioned DLT

References

- Hyperledger Fabric, Quorum, Sawtooth, Besu, Corda, And Multichain Saw A 12-Fold Increase In Engineers In The Last Three Years - <https://www.forbes.com/sites/benjessel/2020/01/23/hyperledger-fabric-quorum-sawtooth-besu-corda-and-multichain-saw-a-12-fold-increase-in-engineers-in-the-last-three-years>

Quiz

Q1: You need to deploy a DLT solution that needs to interact with public Ethereum as well as some nodes in private. Which DLT platform would you work with?

- A. Hyperledger Fabric
- B. R3 Corda
- C. Consensus Quorum
- D. None of these

Q2: Which DLT platform is most widely used in the industry?

- A. Hyperledger Fabric
- B. R3 Corda
- C. Consensus Quorum
- D. None of these

Q3: Why is R3 Corda the platform of choice for Banks and FinTechs?

- A. For its secure architecture
- B. For its enterprise version with support
- C. For its specially crafted libraries for use cases as accounts, tokens, CBDC etc.
- D. All of the above

Q4: Which DLT has the support for standard RDBMS as Postgres, SQL Server, Oracle as underlying ledger?

- A. Hyperledger Fabric

- B. R3 Corda
- C. Consensys Quorum
- D. None of these

Q5: Which of the following DLTs do not have blocks involved in transactions?

- A. Hyperledger Fabric
- B. R3 Corda
- C. Consensys Quorum
- D. None of these

Q6: Please choose the right order of steps involved in consensus mechanism in Hyperledger Fabric.

- A. Proposal & Endorsement, Ordering, Validation, and Commit
- B. Proposal and Validation, Ordering, Endorsement, and Commit
- C. Proposal, Endorsement, Commit
- D. None of the above

Q7: Does notary make R3 Corda's DLT a centralized solution?

- A. Yes, it does as a centralized notary has to check every transaction.
- B. No, R3 Corda DLT can have multiple notaries.
- C. No, Notaries are not mandatory in R3 Corda DLT.
- D. None of the above

Q8: Quorum can be implemented through which of the following?

- A. Solidity
- B. Java based Besu
- C. Go based GoQuorum
- D. Both B and C

Q9: Which among the following is the latest type of orderer in Hyperledger Fabric?

- A. Solo
- B. Kafka
- C. Raft
- D. Notary

Q10: The Accounts library in R3 Corda would be ideal for which use case?

- A. CBDC
- B. Supply Chain
- C. Digital identity
- D. All of the above

Q11: Which languages do R3 Corda support for development?

- A. Java
- B. Kotlin
- C. Both of these
- D. None of these

Q12: Which among the following is a storage to manage all the private and public keys for the encryption in Quorum?

- A. Privacy manager
- B. Transaction manager
- C. Enclave
- D. None of the above

Q13: Oracle services would help in which of the following use cases?

- A. Adding the recording among multiple nodes at the same time
- B. Creation of tokens
- C. Creation of account
- D. Fetching today's BTC/USD exchange rate from a crypto exchange

Q14: What are the different SDKs provided by Hyperledger Fabric in the latest edition?

- A. Hyperledger Fabric Node SDK
- B. Hyperledger Fabric Java SDK
- C. Hyperledger Fabric Go SDK
- D. All of the above

Q15. What are the ways to set up security zones in DLTs for peer-to-peer communication between a subset of a group of nodes in a DLT network?

- A. Channels in Hyperledger Fabric
- B. Setting “privateFor” property in Consensus Quorum
- C. By default, in R3 Corda
- D. All of the above

Answers

1. C 2. A 3. D 4. B 5. B 6. A 7. B 8. D 9. C 10. D 11. C 12. C 13. D
14. D 15. D

PHASE IV – USE CASES

Today, Blockchain is adopted worldwide and across industries. Some of the use cases which are highly rated and even running in production are as follows:

Banking

- Identity management
- Remote on-boarding
- Secure payments
- Instant international money transfer
- Automated instant settlements

e-Governance

- Identity management
- Land registry

EdTech

- Identity management
- Certification programs

Manufacturing

- Identity management
- Supply chain for tracing
- Reduce Turn-around time with greater transparency
- Fraud prevention
- Compliance and Minimization of paperwork
- Efficient inventory management
- Warrant management

Travel

- Loyalty programs
- Seamless travel with Hybrid Identity management (Biometrics and Decentralized Identity)

Trade finance

- Identity management
- Reliability of suppliers
- Provenance
- Asset tracking
- Quality assurance
- Regulatory compliance
- Automated supplier payments
- Cost optimization

Healthcare

- Medical staff credential verification
- Sensitive data-sharing with consent, security and privacy
- IoT security and remote monitoring
- Automated insurance payments

Telecom

- UCC (Unsolicited Commercial Communication)
- Fraud prevention (blacklisting, device tracing, call spoofing)
- Number portability
- Dispute management in Roaming Settlements
- Identity As a Service
- Prevention of phone theft
- 5G enablement
- IoT connectivity

There can be hundreds of use cases possible in Blockchain and it would be beyond the scope of this book to cover them all. However, in the next three chapters, we will explore the use cases which would bring bigger impacts and touch our lives in one way or the other.

CHAPTER 17

Decentralized Identity

“If you ask me to quote one implementation in Blockchain that would lead to mass adoption and to be used by everyone under the sun, it is Decentralized Identity”. – Debajani Mohanty

Before discussing decentralized identity in detail, we must know how different it is from traditional digital identity, and also, why is there even a need for this technology and what all business challenges it addresses.

17.1 Digital Identity

Identifying someone in the real world is not a big deal; however, when it comes to the Internet, we use digital identity to get recognized. Using digital identity, we can identify not just humans, but organizations, devices, and even animals online. Digital Identity is necessary for authentication, authorization, access to online data, registration, and deregistration, and in fact, its role has become crucial for beginning and maintaining any kind of business relationship online. Obviously, digital identity is some information that is unique to a user that the user would not share with anyone. For example, user ids, bank accounts, credit card numbers, biometrics data etc. Digital identity is also associated with the user's personal data that may require privacy; refer to the following as examples:

- Credit card, debit card, or any form of financial details
- Health information
- Biometric data
- Criminal history
- Ethnic and racial origin
- Sexual orientation

- IP address

Unauthorised access to such information may lead to privacy issues, and hence, need to be protected with utmost care. With more and more digital awareness, there is a growing concern of data breaches and mishandling of personal data by the organizations. Recently, WEF published the following statement:

“There is a growing concern about privacy issues in the use of identity information. For example, more than 60% of people in the US are concerned about how the collected information will be used by companies and governments”.

WEF also referred to the report by PEW research centre, which is represented in [Figure 17.1](#) as follows:

Majority of Americans feel as if they have little control over data collected about them by companies and the government

% of U.S. adults who say ...

		Companies	The government
Lack of control	They have very little/no control over the data __ collect(s)	81%	84%
Risks outweigh benefits	Potential risks of __ collecting data about them outweigh the benefits	81%	66%
Concern over data use	They are very/somewhat concerned about how __ use(s) the data collected	79%	64%
Lack of understanding about data use	They have very little/no understanding about what __ do/does with the data collected	59%	78%

Note: Those who did not give an answer or who gave other responses are not shown.

Source: Survey conducted June 3-17, 2019.

“Americans and Privacy: Concerned, Confused and Feeling Lack of Control Over Their Personal Information”

PEW RESEARCH CENTER

Figure 17.1: (Source:
https://www.pewresearch.org/internet/2019/11/15/americans-and-privacy-concerned-confused-and-feeling-lack-of-control-over-their-personal-information/pi_2019-11-14_privacy_0-02-2/)

So, how has personal data associated with digital identity been handled so far? Let's find what are the different identity patterns that we have come across so far and what's there for us in the near future.

17.2 Evolution of Digital Identity

Since the Internet's inception a few decades ago, the user interaction over Internet always required user's digital representation and it is not error-free. Inadequate security practices have led to huge financial and other losses in the past that we already discussed in the previous chapter. Hence, from time-to-time, different architectures have been introduced in the market that are always trying to improve by getting rid of the vulnerabilities and adding more features in the process. As we are moving towards tremendous growth of the "Internet of Things", please expect many more models to come in the days to come. In [Figure 17.2](#), you can see the different identity models over a span of a few decades, as shown as follows:

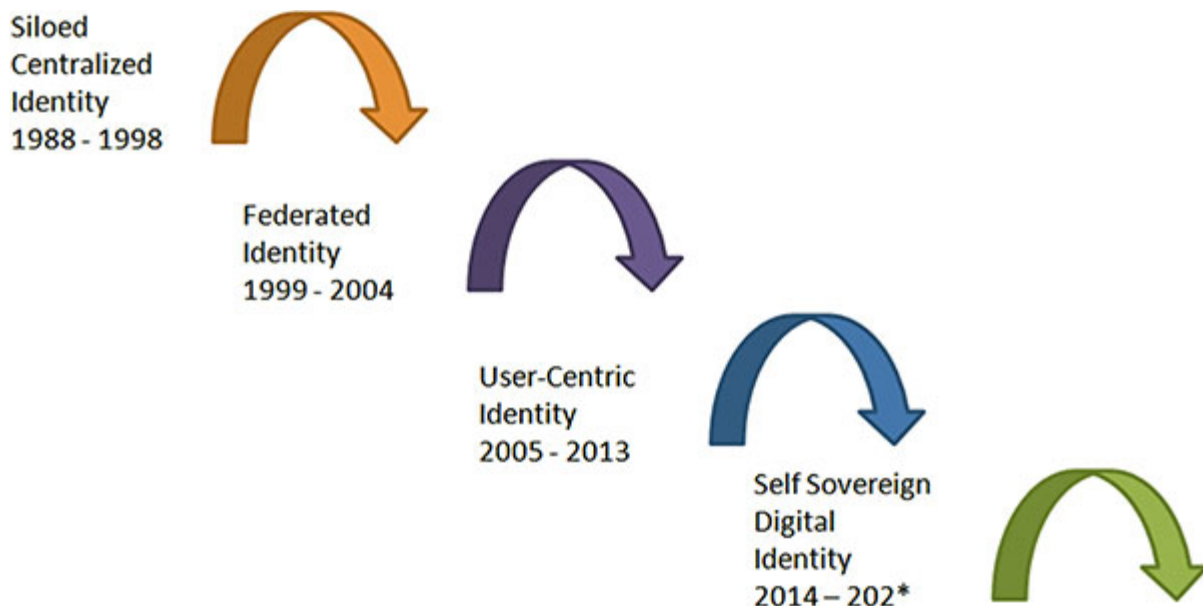


Figure 17.2: Evolution of Identity

17.2.1 Centralized Identity

The IT industry has evolved exponentially in just the last two decades. Old-timers like us, who learnt how to access the Internet in late 90s, still remember how the Internet-based applications such as banks, airlines etc., started as static websites, followed by being more and more interactive and secure, and finally reaching the state where they are today. In the early days, most websites that offered the logging in feature were based on user id and password.

As shown in [Figure 17.3](#), a user had to remember a separate set of User Id and password for each application he logs in. Also, the central repository is prone to mass attack by hackers or it simply may lead to a single point of failure.

Refer to [Figure 17.3](#), as follows:

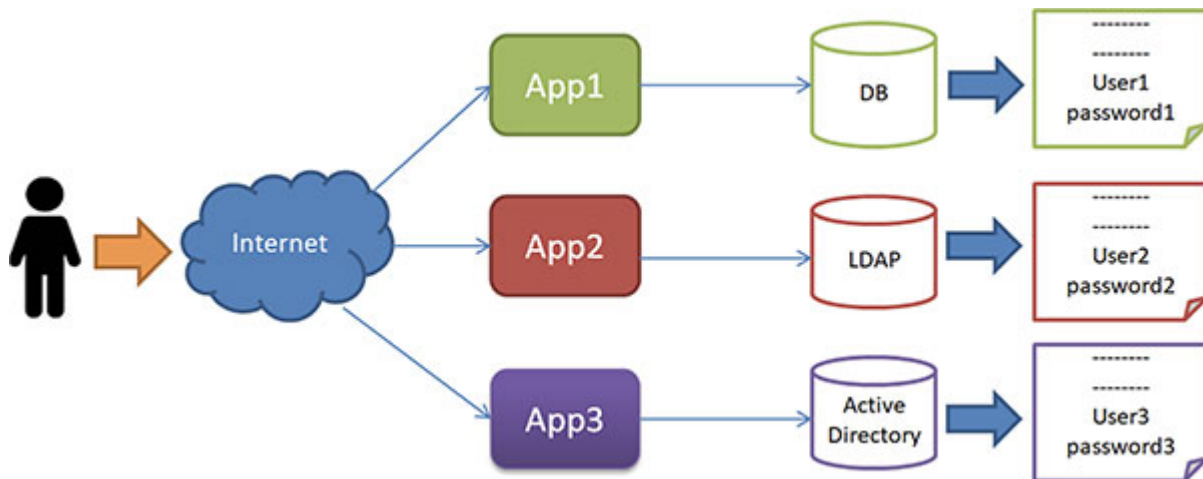


Figure 17.3: Centralized Identities

Some of the examples of centralized identity are a custom database, LDAP, Active Directory etc., which can work either in the standalone or the master-slave mode.

17.2.2 Federated Identity

In this type of identity protocol, the authentication part is separated from the application code and delegated to a trusted third party who is called an “Identity Provider” or IdP which minimizes the administrative overhead.

As shown in [Figure 17.4](#), when the user tries to log into any third party, the application in turn invokes the Identity Provider. The IdP returns a token that the application shares with the third party. Now, this third party can validate the token by directly interacting with the IdP. Single Sign On or SSO belongs to this type of identity management which can be architected in many different ways. Refer to [Figure 17.4](#), as follows:

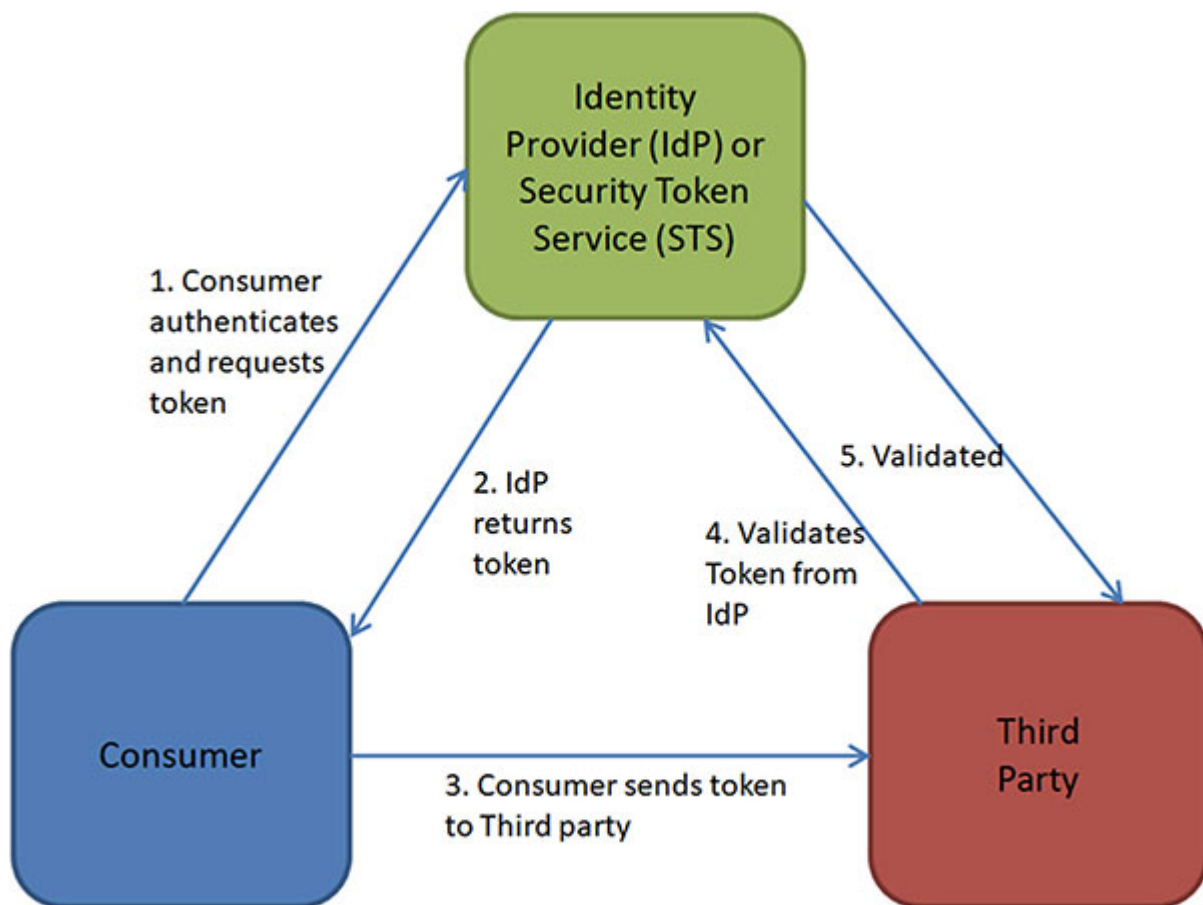


Figure 17.4: Federated Identities

SSO has proved to be extremely useful for the users as they do not have to log in again and again while traversing multiple websites retaining the log in status. They also do not need to memorise multiple passwords for each website in this process.

However, the user's personal data is still saved with one centralized server in the ownership of the Identity Provider that the other parties can check and trust. Hence, its identity management system can still lead to a single point of failure and mass attack by hackers.

17.2.3 User Centric Identity

This identity model further improves the user experiences to the next level.

As per wiki, “User-centric designs turned centralized identities into interoperable federated identities with centralized control, while also respecting some level of user consent about how to share an identity (and with whom). It was an important step toward true user control of identity, but just a step. To take the next step, required user autonomy”.

Microsoft's CardSpace falls under this category, whereas some of the most widely used user central identity protocols, perhaps today, are OpenID (2005), OpenID 2.0 (2006), OpenID Connect (2014), OAuth (2010), and FIDO (2013). The roles in this identity patterns are of four types, which are as follows:

1. User or Resource Owner
2. Client Application that accesses protected resources on behalf of the user
3. Authorization Server that issues access tokens to authenticated client applications
4. Resource Server that assigns access to a protected resource on the basis of access tokens

As shown in [Figure 17.5](#), this approach though had some benefits over previous patterns, however was not enough. The pattern needs long connectivity and can get disconnected in between.

Refer to [Figure 17.5](#), as follows:

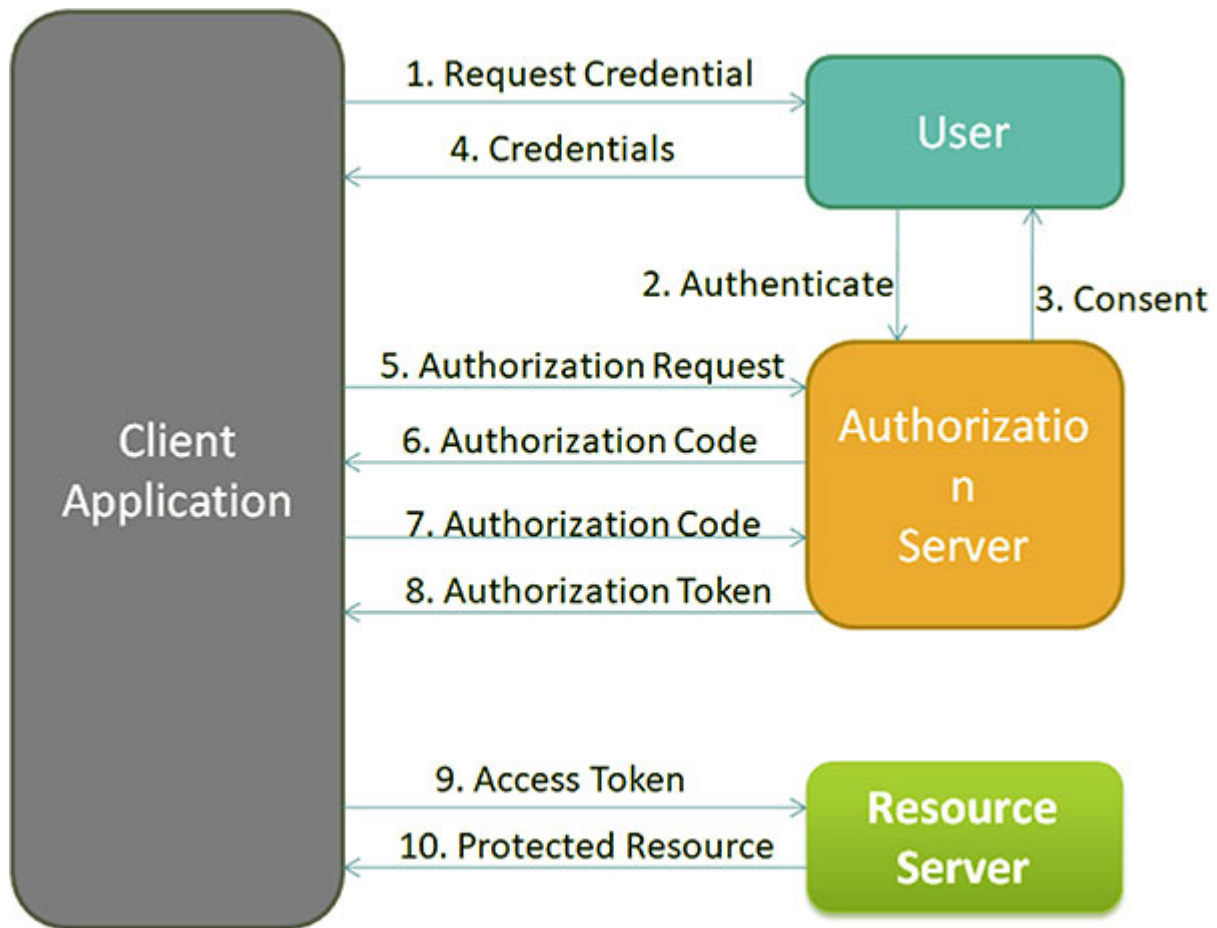


Figure 17.5: User Centric Identities

17.2.4 Decentralized Identity (DID)

In the day-to-day life, we often need to showcase our certificates to different organizations, a few examples are as follows:

- When a child joins a school, they ask for his birth certificate as well as the social security number or Aadhaar card in India.
- The bank where we apply for a loan requires our income certificate, employment status, as well as age.
- The police station where we register a road accident enquires driver's licence.

However, mostly these organizations do not have a direct way of checking the validity of these certificates. They rely on a third party that does an independent background verification for them. Also, the

user has no means of sharing the partial data with the verifier organizations.

Using Decentralized Identity (or DID) approach, we can create an ecosystem where Issuer, Holder, and Verifier can achieve their goals with utmost safety, security, and ease of use. As shown in [Figure 17.6](#), the flow can be broadly explained in the following four steps:

1. The Issuer and verifier organizations create a pair of private/public keys using Cryptography as well as a public DID on Blockchain that is visible to all. Issuer then issues a VC to the user that the latter stores in their own mobile device. The Issuer also signs the VC with their own private key and writes to a public Blockchain.
2. The user, on a later day, can share the data with a third-party Verifier organization.
3. The Verifier matches the VC against the reference data signed by the Issuer on the Blockchain.
4. If on a later date, there is a need, then the Issuer simply revokes the VC by updating the reference data Blockchain. The Verifier can again check to know that the VC is no longer valid.

Refer to [Figure 17.6](#), as follows:

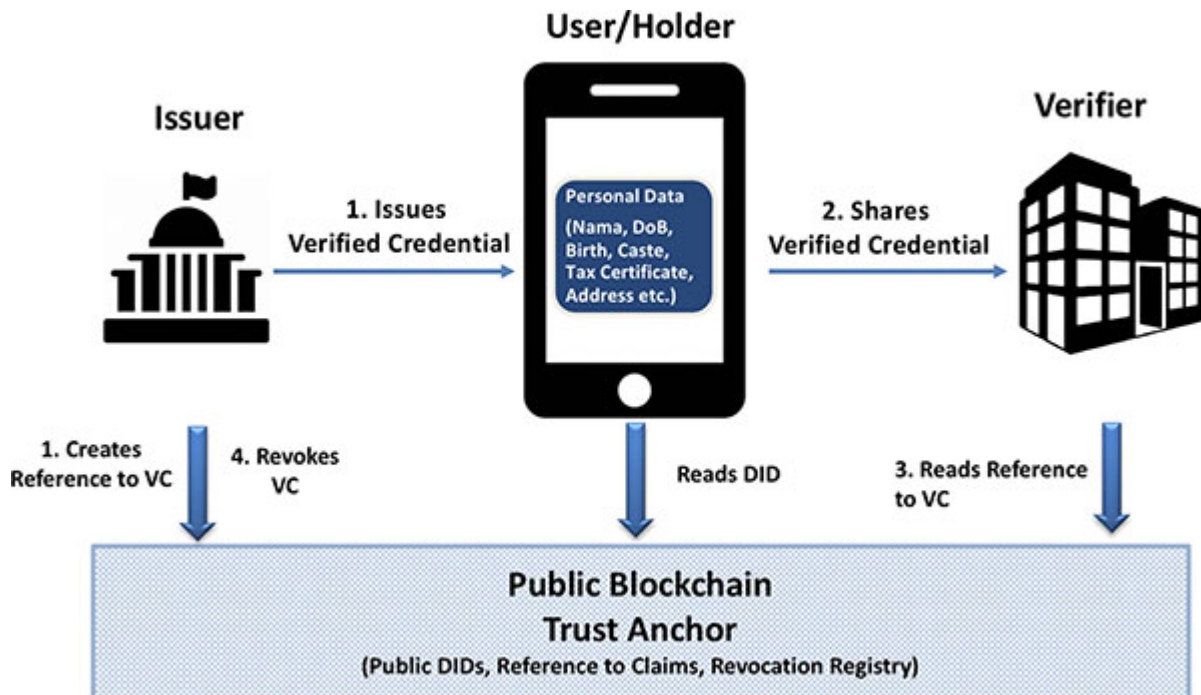


Figure 17.6: DID Architecture

This decentralized identity architecture ensures the following:

- **Ownership** of the data by the user verified by Issuer and Verifier.
- Data is shared only with the user's **consent**, and hence, ensures **privacy and security**.
- **Verified certification** by the Issuer as claimed by the user.
- **Integrity** of the data as any change of hash on the Blockchain would indicate that the data is either tampered or is no longer valid.
- The model also enables Verifier organization to check if the data is still **valid** and not revoked by Issuer.

The beauty of this model is that there is no need for the Verifier to do another background verification of the User's data as it's already verified by Issuer and the reference data is available online 24/7 through a public Blockchain. Please note that the user's personal data is not written to the Blockchain, rather only a reference hash is written on it. Hence, DID also adheres to most of the acts and laws

(GDPR, PDPA, PDPB etc.) across the world formulated on the privacy of personal data.

But, can't the Identity solutions be handled without Blockchain?

Please note that the identity solutions can be implemented using centralized storage (i.e., without Blockchain) and yet the DID implementations come with many benefits broadly categorized as security, controllability, and portability as represented in [Table 17.1](#) as follows:

Security	Controllability	Portability
The identity information must be kept secure	The user must be in control of who can see and access their data	The user must be able to use their identity data wherever they want and not to be tied to a single provider
Protection	Existence/User-centricity	Interoperability
Persistence/Longevity	Control	Transparency
Minimization	Consent	Access

Table 17.1: Features of DID solutions

17.3 World Wide Web Consortium

While developing a project in the decentralized identity that involves relatively an emerging space, it's desirable to understand the new features that DID can offer and, at the same time, not get into any loopholes in this nascent technology. Founded in 1994, the World Wide Web Consortium or W3C is the prime organization for setting the international standards for the world wide web. Its main mission is to develop the protocols and guidelines that ensure long-term growth for the web. W3C perhaps is the first consortium that came out to develop a set of protocols to define "**Decentralized Identifier**" or **DID**.

DID is a globally unique identifier and doesn't require a centralized registration authority, and unlike traditional identifiers that are created for us by third parties, a DID is self-created by the user.

Any organization that wishes to build a DID framework has to abide by all these rules for the W3C compliance.

17.4 Data sharing types

DID platforms are architected in such a way that the user has the options of sharing the data in the following three different modes:

- Traditional
- Selective disclosure
- Zero-knowledge-proof modes

In the traditional mode, all the data in the VC are shared as it is, whereas in the selective disclosure mode, only the required data are shared with the third party verifiers to promote privacy. In the zero-knowledge-proof mode, the user shares the data only in Boolean (i.e., true or false) not revealing the actual value; for example, “are you an adult, is your salary more than a particular amount” etc. In all the three different modes of data sharing, the verifier can verify the integrity, ownership, and validity of the data using cryptologic algorithms.

17.5 DID Use Cases

Today, Decentralized Identity is widely used with the other emerging technologies to build many next-generation use cases across industry verticals. A few are listed as follows:

- **National Identity** programs can adopt decentralized identity approach, so that it can be the basic identity for every citizen. Other government and private issued certificates like driving license, civic records, access to libraries, voter ids, bank accounts etc., can refer to the national identity as the first verified identity.

- **KYC and Remote digital onboarding** of new employees, students, and customers can be a completely seamless experience by using a combination of decentralized identity backed data sharing and live biometrics.
- **Next-gen authentication and authorization** is also possible with the preceding model which can work for one organization or multiple organizations in a Single-Sign-On mode.
- Decentralized digital identity is also a key requirement for remote **E-Voting** which can facilitate people to vote from any place in the world. The process would make sure that no voter can participate in voting with a fake identity, voter's identity is verified, and yet it remains hidden for privacy purpose, a single vote is cast per user etc. At the same time, the voter can check that the vote goes to the right contestant and the ecosystem is capable of handling the massive load.
- **Background verification** of new students or employees can be absolutely seamless for the education and employment use cases using decentralized identity to produce degrees, transcripts, badges, awards, statements, agreements etc., on the fly.
- **Loyalty programs** can be built using decentralized identity as a unique reference.
- **Health passports or Covid passports** can be issued to individuals for travel and stay at hotels as a basic requisite.
- Airlines can introduce **Seamless Travel** for travellers where the traveller shares verified identity, tickets, visa, and health certificates before the travelling date and time.
- **Verified credentials** can help the healthcare workers working on a rotation basis in different hospitals to be identified.
- Also, hospitals can share the verified patients' data with research institutes in selective data sharing mode for **Research in healthcare**.

The list is long, and it would be outside the scope of this book to cover them all. Let's elaborate some of the most important ones that would affect the life of a common man in the days to come. The top

three that comes to mind are remote voting, smart agriculture, and CBDC. While I am covering the first two here, CBDC would be covered in the next chapter.

17.5.1 Use Case - Remote Voting

One of the most anticipated use cases in the Blockchain space today is remote voting. People have to travel all the way to a particular place at a particular time to cast their vote that goes to a centralised system. What the ecosystem needs is a solution, using which the people can cast their vote either from their own mobile device, a website, or from a kiosk. However, unlike the other Blockchain use cases, remote voting needs additional security and privacy. Some of the special challenges in this use case are as follows:

- There can be a denial-of-service attack on a centralized server.
- The voter can come up with a fake identity.
- The voter might try to cast vote multiple times.
- The voter should be able to check if his/her vote is cast for the right candidate.
- Privacy of the user's vote should be protected.
- The ecosystem should be able to handle the massive load.

All these issues can be handled efficiently by using Blockchain. With appropriate decentralized identity solution along with live biometrics, one can vote from any place in the world. The solution should work for the tech-savvy who can cast votes from the EarthId App and also for the common man who has no access to the smart devices. Many countries in the world are researching to come up with such a solution that can also scale well.

17.5.2 Use Case – Smart Agriculture

As shown in [*Figure 17.7*](#), now the Animals too can have their own decentralized digital identity to help them send their status updates to the rightful owner from their IoT devices, as follows:

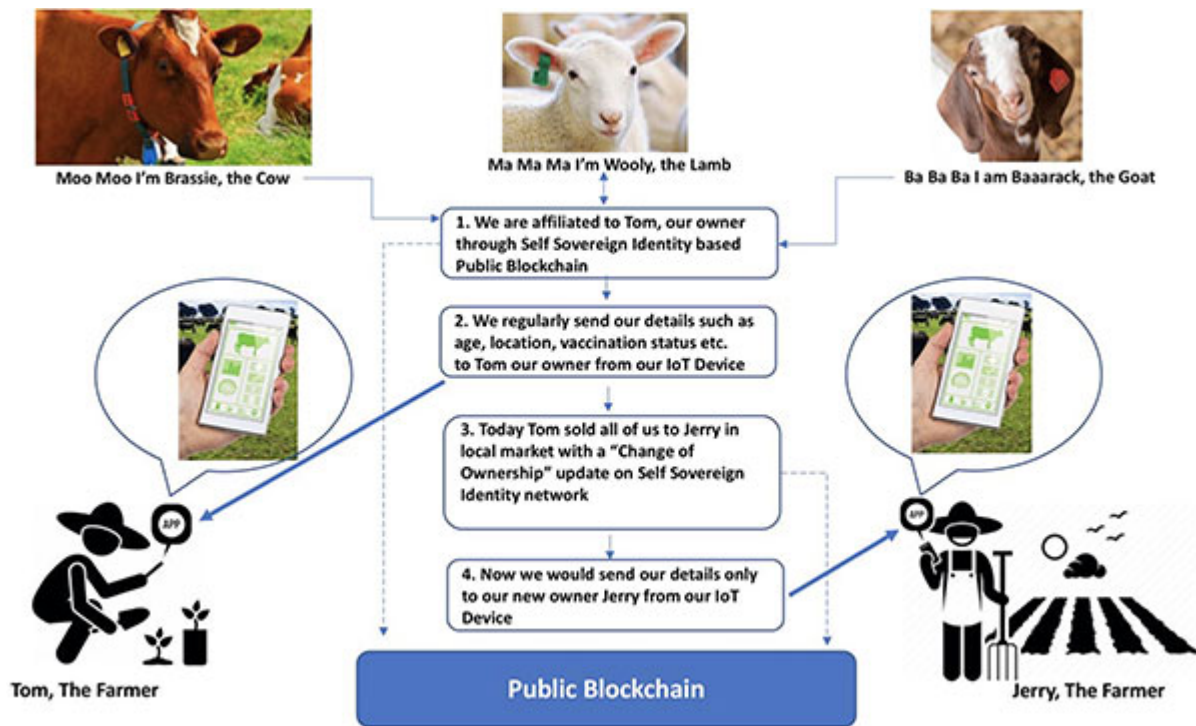


Figure 17.7: Smart Agriculture

DID for the IoT devices would bring a huge number of use cases for smart cities and smart farming in the future.

17.6 DID for Privacy Programs

Privacy is a fundamental human rights and foundation to maintaining a free society and to protect a democratic way of life. Privacy also means user's full control over personal data, which is shared only with consent.

Gartner says, by 2023, 65% of the World's Population will have its personal data covered under modern privacy regulations, up from 10% in 2020. Privacy-Tech, like Reg-Tech, is emerging.

In recent times, many countries as well as business verticals have come up with laws to protect personal data and privacy.

UNITED STATES

State Level:

- California Consumer Privacy Act (CCPA) - 2018
- Illinois Biometric Information Privacy Act (BIPA) - 2008

- Virginia Consumer Data Protection Act - March 2021

Federal Level:

- Gramm Leach Bliley Act (GLBA), Reg P - Financial
- Health Insurance Portability and Accountability Act (HIPAA) - Health
- Children's Online Privacy Protection Act (COPPA) – Education/Children
- The Electronic Communications Privacy Act of 1986 (EPCA) - Electronic
- The Telephone Consumer Protection Act of 1991 (TCPA) - Telecom

INTERNATIONAL

- EU - General Data Protection Regulations (GDPR) – May 2018
- Australia - The Privacy Act 1988 and Australian Privacy Principles (APPs)
- New Zealand: Privacy Act 2020
- China - National Standard on Personal Information Protection - May 2018
- India - Personal Data Protection Bill, 2019 (Proposed)
- Canada - Personal Information Protection and Electronic Documents Act (PIPEDA)
- Hong Kong - Personal Data (Privacy Ordinance)
- Japan – Act on the Protection of Personal Information (APPI) - Privacy Mark
- Thailand - Personal Data Protection Act - February 2019

Some of the common requirements in the business dealing with personal data would be as follows:

Obtain Consents

- Comply with the obligations to inform
- Meet Privacy and legal requirements

- Auditable consent reports

Manage Consents

- Proof of consent and preference choices
- Transparency on data collection
- Assist in data protection strategies

Optimize consents

- Automate consent management
- Scalable solution when companies grow

Hence, Decentralized Identity would be heavily used in the digital business in days to come and, with time, DID too would evolve.

17.7 DID Products

There are a couple of decentralized identity products already available in the market, which are as follows:

- Sovrin foundation backed by Hyperledger Indy
- Microsoft ION
- EarthId backed by Hedera Hashgraph
- SelfKey backed by Ethereum
- SecureKey backed by Hyperledger Fabric

Many of them also have integrated wallet solutions and can be easily integrated with the existing software applications. No need to mention that the underlying Blockchain or DLT network plays a key role in the success of these products. Hence, it's advisable to do good research before choosing a decentralized identity product for your software solution.

17.8 Summary

In this chapter, we covered the following topics:

- Evolution of Digital Identity with advantages of DID over its predecessors.

- DID standards and DID use cases.
- Privacy programs across the globe on safeguarding personal data and how DID can help.
- Popular DID products in the market with Blockchains and DLTs working underneath.

References

- A practical approach to implement a Privacy Program to meet CCPA and GDPR requirements? - <https://www.linkedin.com/pulse/practical-approach-implement-privacy-program-meet-ccpa-sangram-dash/>
- W3C DID Methods - <https://w3c.github.io/did-spec-registries/#did-methods>

Quiz

Q1: What are the causes of data breaches?

- A. Weak password
- B. Centralized server
- C. Identity co-relation
- D. All of the above

Q2: Which of the following data regulation is for Europe?

- A. GDPR
- B. HIPPA
- C. CCPA
- D. PDPA

Q3: What is the main difference between GDPR and PCI DSS data regulations?

- A. GDPR is for Europe, whereas PCI DSS is for the Americas
- B. GDPR deals with personal data, whereas PCI DSS deals with all data

- C. GDPR deals with all types of personal data, whereas PCI DSS deals with only payment card data
- D. We must abide by either of these data regulations

Q4: What can be the possible issues with a centralized voting solution?

- A. There can be a DDoS attack on the server
- B. Mass attack possible on the database
- C. Voter's privacy and anonymity can be an issue
- D. All of the above

Q5: What is the main difference between user centric identity and DID?

- A. In user centric identity, the third party decides whether to share personal data, whereas in DID, it's the user who takes the decision
- B. In DID, the data is saved in a centralized location, in user centric identity, it's not
- C. In DID, data can be of any type, i.e., image, documents etc., whereas in user centric identity, only strings can be saved
- D. DID architecture would be much more expensive to implement

Q6: Which of the following is true regarding Selective Disclosure?

- A. In this model, data is not shared
- B. In this model, only part of the data is visible
- C. In this model, only Boolean value is used to answer a question rather than exposing the real data
- D. In this model, the data is replaced with zeroes

Q7: What does Zero Knowledge Proof mean?

- A. Holder creating private DIDs for each new connection in order to make the data sharing correlation-resistant
- B. Holder answering to Verifier's queries in Boolean not revealing the actual data and still proving it

- C. Both A and B
- D. None of the above

Q8: As per Gartner Hype Cycle 2021, in how many years would DID reach “plateau of productivity”?

- A. Already reached
- B. 2-5 years
- C. 5-10 years
- D. None of these

Q9: Why would DID be the new face of blockchain?

- A. Mass adoption by billions
- B. Next-gen authentication with advanced biometrics
- C. New trust model in Covid-19 like pandemic era
- D. All of the above

Q10: As per Gartner, by 2023, what percentage of the World's Population will have its personal data covered under modern privacy regulations (which is 10% by 2020)?

- A. 25%
- B. 45%
- C. 65%
- D. 100%

Answers

1. D 2. A 3. C 4. D 5. A 6. B 7. B 8. B 9. D 10. C

CHAPTER 18

Tokenization, DeFi, NFT and CBDC

“Everything will be tokenized and connected by a Blockchain one day” – Fred Ehrsam

In this chapter, we will cover some of the Blockchain use cases that are most widely adopted in the BFSI sector.

18.1 Decentralized Finance, Tokenization and NFT

Decentralized finance or, in short DeFi, has been the talk of the town for the last few years. DeFi stands for a group of financial applications that uses payments in cryptocurrencies in one way or the other, and hence, leverages the benefits of its underlying Blockchain technology. Because of its decentralization, DeFi has introduced certain novel and very complex business scenarios which could not have been addressed with traditional centralized technologies.

Business reporter has published that Decentralized finance has grown to an \$80 billion industry in 2021 and is poised to explode 10-fold, according to veteran crypto investor Matthew Roszak.

Often, the DeFi projects are associated with tokens that are the essence of a DeFi ecosystem. In many ways, the crypto tokens are similar as well as different from the cryptocurrencies. While the cryptocurrencies are associated with only digital money, a crypto token can be a placeholder for either fiat currency, cryptocurrency, or any other physical asset of value. Both cryptocurrencies and crypto tokens are associated with the underlying Blockchain layer for cryptographical security. Each token is also linked with some digital ownership that one can prove through cryptography. Smart contracts

can be developed for instant delivery against the payments with tokens.

A crypto token can be of two different types, i.e., fungible token and non-fungible tokens.

Fungible tokens are like Bitcoin or any other cryptocurrency where every crypto has a similar value.

Non-fungible tokens or NFT are again crypto tokens, where every token is unique and has a different value or asset associated with it. Hence, unlike fungible tokens, NFTs cannot be replaced with each other. NFT gives us the power to trade unique verifiable digital assets on Blockchain.

While the NFT craze started only in 2020, by the first half of 2021, it has reached \$2.5 billion.

With a combined use of fungible tokens (or fiat cash) and NFTs, we can solve many of the frictions in the BFSI sector that has been there since ages. Most deliveries against payment use cases can be automated without the need of intermediaries resulting in a reduction in time, efforts, and expenses. Some of the novel use cases that these tokens can help in are as follows:

- Instant International Money Transfer
- Instant Settlement in Capital or Money Market
- Hedge Funds, Private Equity, Venture Capital, P2P Debt
- Loyalty programs in Travel, Insurance or retail

18.1.1 NFT Craze for Digital Contents

NFT is the latest craze in the Blockchain and DLT space. Nowadays, many celebrities are declaring NFT on their name. What does this really mean and how do such NFTs work?

These NFTs that are making the headlines are mostly used for the selling of arts in various forms and also to pass on the digital ownership. However, do we not own our digital contents already? Why do we need an NFT for that?

Let's consider the case that Alice is selling her digital contents to Bob. Now, Bob resells it to Carol and the chain continues to Harry, Frank, Grace etc. While Alice gets paid for the first time sale, she is not getting benefitted in all these reselling deals. With this new approach, Alice can create an NFT associating it with the digital content, i.e., "This content is created by Alice on such and such date and she is the original Issuer. Alice's address on Ethereum is xxxx". A fraction of money would go to her on every reselling deal where Alice is not directly involved.

If you are any kind of digital content creator, i.e., blogger, financial advisor, scientist, movie star, author etc., you can create your own NFT and sell your digital contents, even blogs, tweets etc., over the Internet with the assurance that every time it's resold, you can earn money out of it. In education, NFT can be used in Research, PhD paper presentation, Design, Architecture, Formulas etc.

18.1.2 Blockchain & DLT Platforms

NFTs can be architected in many different ways as per the business need. As shown in [Figure 18.1](#), an NFT would need a protocol where we can set different criteria for NFT regulation, whereas the NFT token can carry the information such as a unique id, Issuer, Issuance date etc., as follows:

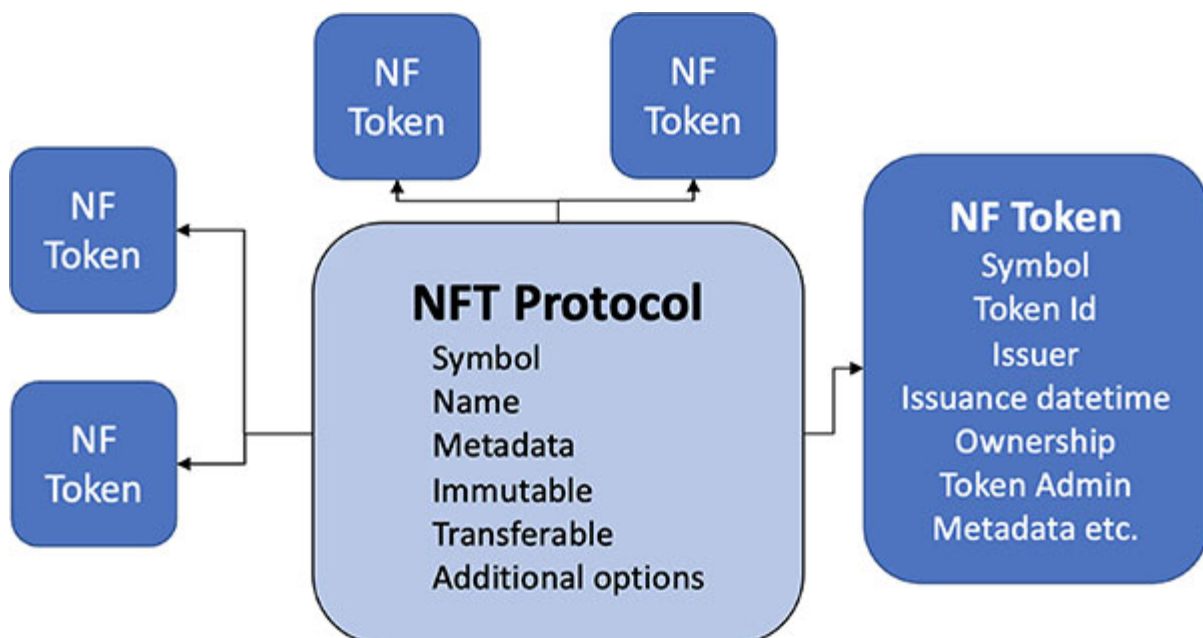


Figure 18.1: NFT Protocol

For the past few years, Ethereum has led this space of DeFi and NFTs, but lately, there are several other players introduced to this wild race. Some of the leading ones are Cardano, Solana, Polkadot, Cosmos, Avalanche, Polygon etc.

18.2 Stablecoins

One of the top reasons for most cryptocurrencies in the world including Bitcoin to always be criticized is that they are not backed by any collateral. Starting from the barter system to the digital era, currency has undergone different forms, i.e., crops, gold, silver, spices, salt etc. Currency initially has been used as a medium of exchange and always had an inherent value associated with it. With paper money, the issue of trust came into picture and banks as well as governments came forward to establish that trust.

The second issue with crypto is their highly volatile price fluctuation. Hence, in order to bring the same goodness of cryptocurrencies as transparency, immutability, trust etc., at the same time, to get rid of its price volatility and also middleman tips imposed by Banks, Stablecoins were introduced.

Stablecoins are the new type of cryptocurrencies created through the same tokenization model; however, they are backed by real world securities due to which their price remains relatively unaltered.

As shown in [Figure 18.2](#), Stablecoins can be created with many different types of collaterals. However, if the value of the collateral changes, then the price of the Stablecoin too can alter. Refer to [Figure 18.2](#), as follows:

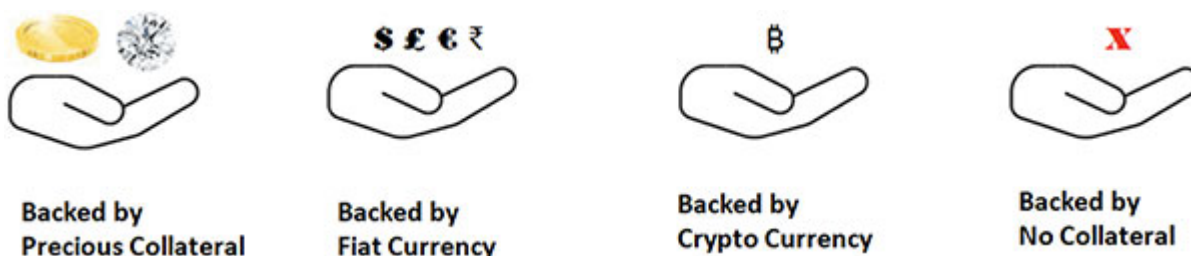


Figure 18.2: Types of Stablecoins

Please also note that even if most Stablecoins have Blockchain as the underlying technology, the associated collateral is centralized in nature. As represented in [Figure 18.3](#), all Stablecoins have their pros and cons and it's up to the organization to use one pattern over another, as follows:

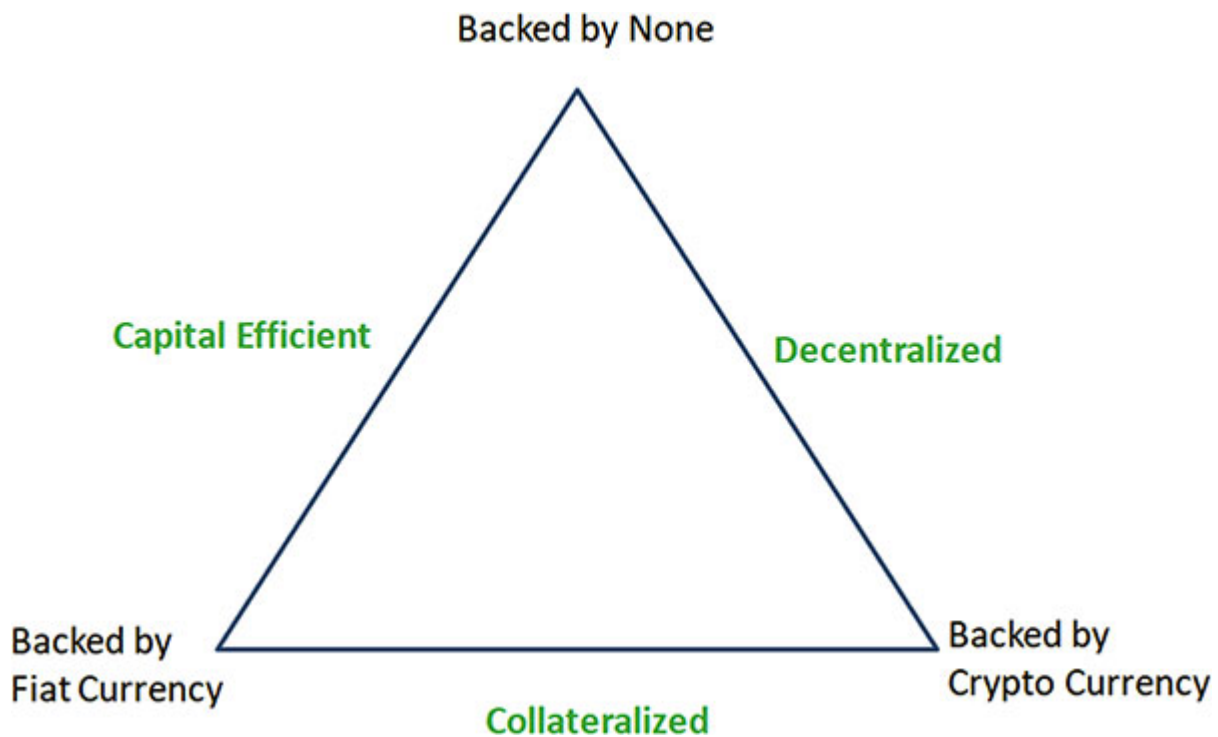


Figure 18.3: Pros and Cons of Different Types of Stablecoins

Today, the Stablecoin market capitalization has reached \$133B and still counting. As per coinmarketcap.com, Tether, USD Coin, Binance USD, DAI, TerraUSD are some of the most popular Stablecoins with maximum market capitalization backed by USD.

18.2.1 Pros and Cons of Stablecoins

The advantages of Stablecoins are as follows:

- Private money and private transactions
- Collateral gives us peace of mind
- Instant processing of international and domestic payments

The disadvantages, however, are as follows:

- Not universally accepted
- No regulation by Government

18.3 CBDC

The central bank digital currency or CBDC is a digital currency issued by the central bank of a nation that leverages some benefits of cryptocurrencies by using the distributed ledger technologies at its back. But does it not sound exactly like a Stablecoin? Perhaps, yes. However, the biggest difference between them is that, while most Stablecoins are created by private financial institutes, CBDC is issued by the central bank, and hence, it's a legal tender that can be strictly regulated by the monetary authorities of a nation. Hence, CBDC can be termed as "A Centralized Cryptocurrency" and is universally acceptable.

18.3.1 CBDC Types

CBDC can be categorized into two types depending upon its targeted users, i.e., wholesale and retail.

18.3.1.1 Wholesale CBDC

Wholesale CBDC deals with the tokenization of digital money issued by the central bank, built with the goal of resolving liquidity and counterparty risk issues. As shown in [Figure 18.4](#), they can work in scenarios where there is need of interbank instant settlements or instant international money transfer, as follows:

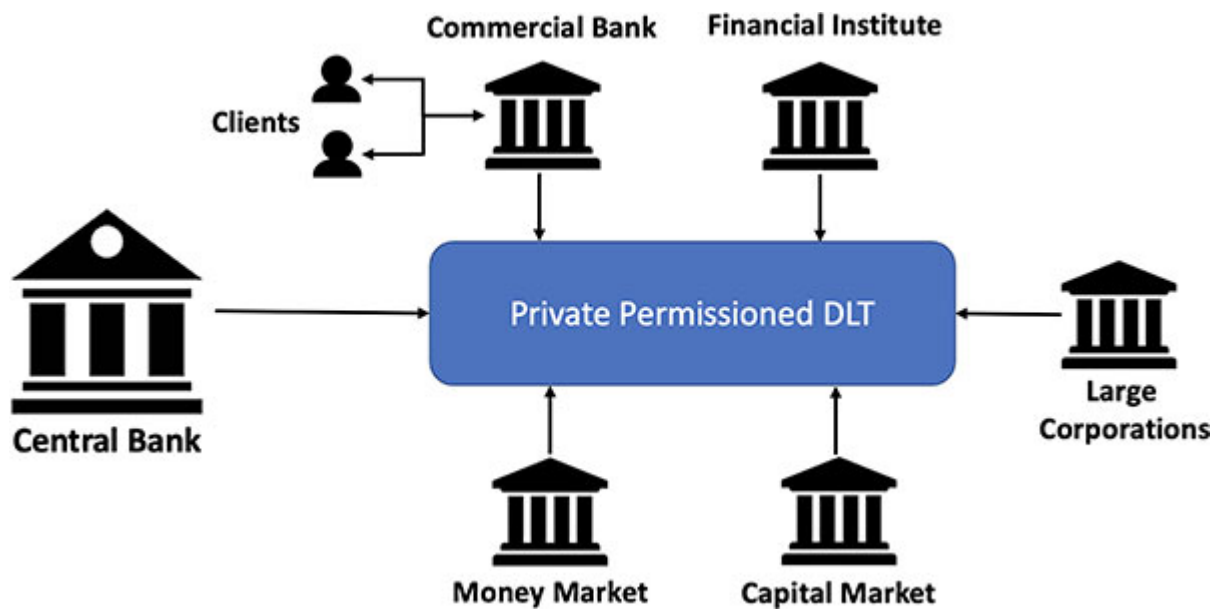


Figure 18.4: Wholesale CBDC (Source: Consensys)

Wholesale CBDCs are relatively simple to architect and handle as the number of parties involved are relatively less.

18.3.1.2 Retail CBDC

The retail CBDC can be associated with an account just like the bank accounts or it can be anonymous, similar to cash. As shown in [Figure 18.5](#), the retail CBDC can be divided into three types, i.e., direct, indirect, and hybrid.

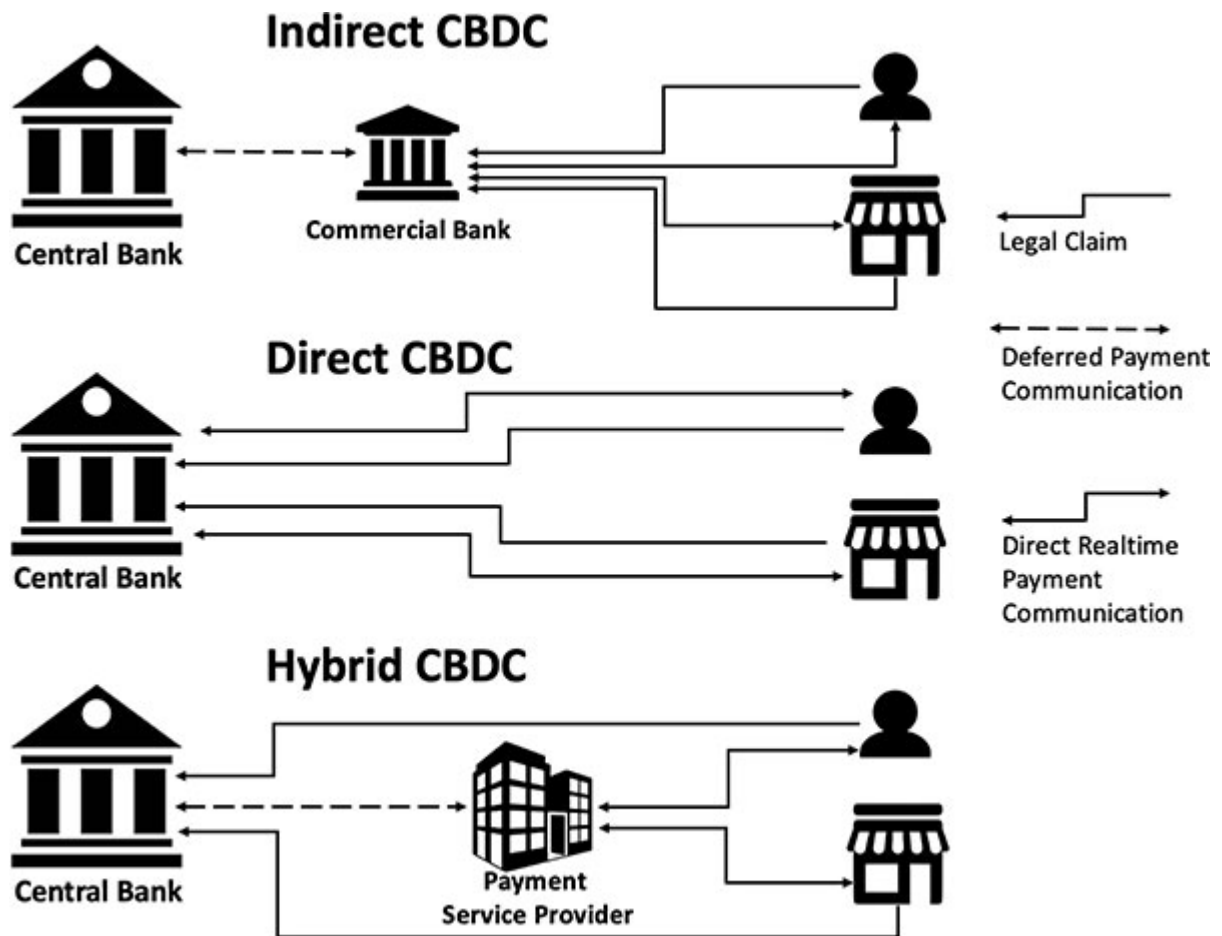


Figure 18.5: Retail CBDC Payment Communication

(Source: <https://www.bloomberg.com/news/articles/2021-03-22/here-s-how-a-central-bank-digital-currency-could-work-chart>)

18.3.1.2.1 Indirect CBDC

Indirect CBDC is indirectly issued by the central bank and intermediary banks handle the retail part. It is mostly used between financial institutions for payment.

18.3.1.2.2 Direct CBDC

Direct CBDC, as the name suggests, is directly issued by the central bank to individuals or financial institutes where the central bank keeps a record of all the money transfers in real time.

18.3.1.2.3 Hybrid CBDC

In this model, one can send a direct claim to the central bank, whereas the real-time payments are handled by intermediaries.

18.3.2 Blockchain & DLT Platforms

R3 Corda, that was initially designed to handle DLT use cases, especially in the financial industry, today has emerged as the biggest player in the CBDC market. It has set up a CBDC research center where they have built an R3 Sandbox for digital currencies, using which, CBDCs can be easily created. From the Hyperledger family, Fabric, Besu, and Iroha are also under research in this space. Hedera Hashgraph, in spite of being a public DLT, is also investing a lot on the research of CBDC with its low cost as well as higher energy efficient infrastructure.

[Figure 18.6](#) is derived from a paper from R3 Corda that shows how digital money can be transferred from the central bank to the Tier-I commercial banks and from Tier-I banks to Tier-II banks or retail banks or payment service providers. At each stage, we can use Corda's accounts model to transfer the money to individual accounts. Refer to [Figure 18.6](#), as follows:

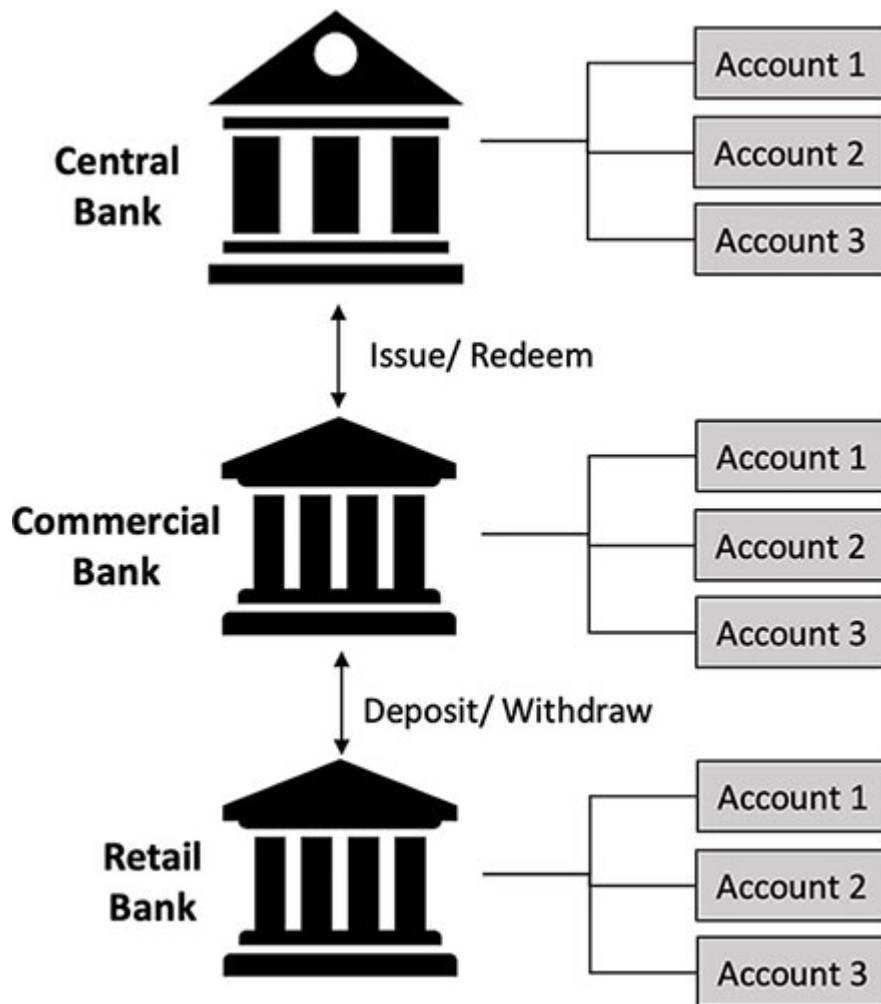


Figure 18.6: R3 Corda's CBDC Architecture

(Source: https://www.r3.com/wp-content/uploads/2020/04/r3_CBDC_report.pdf)

[Figure 18.7](#) shows the architecture of DLT for the implementation of CBDC at Sweden's Riksbank (Sweden's central bank) where they have used R3 Corda's accounts library for the same, as shown as follows:

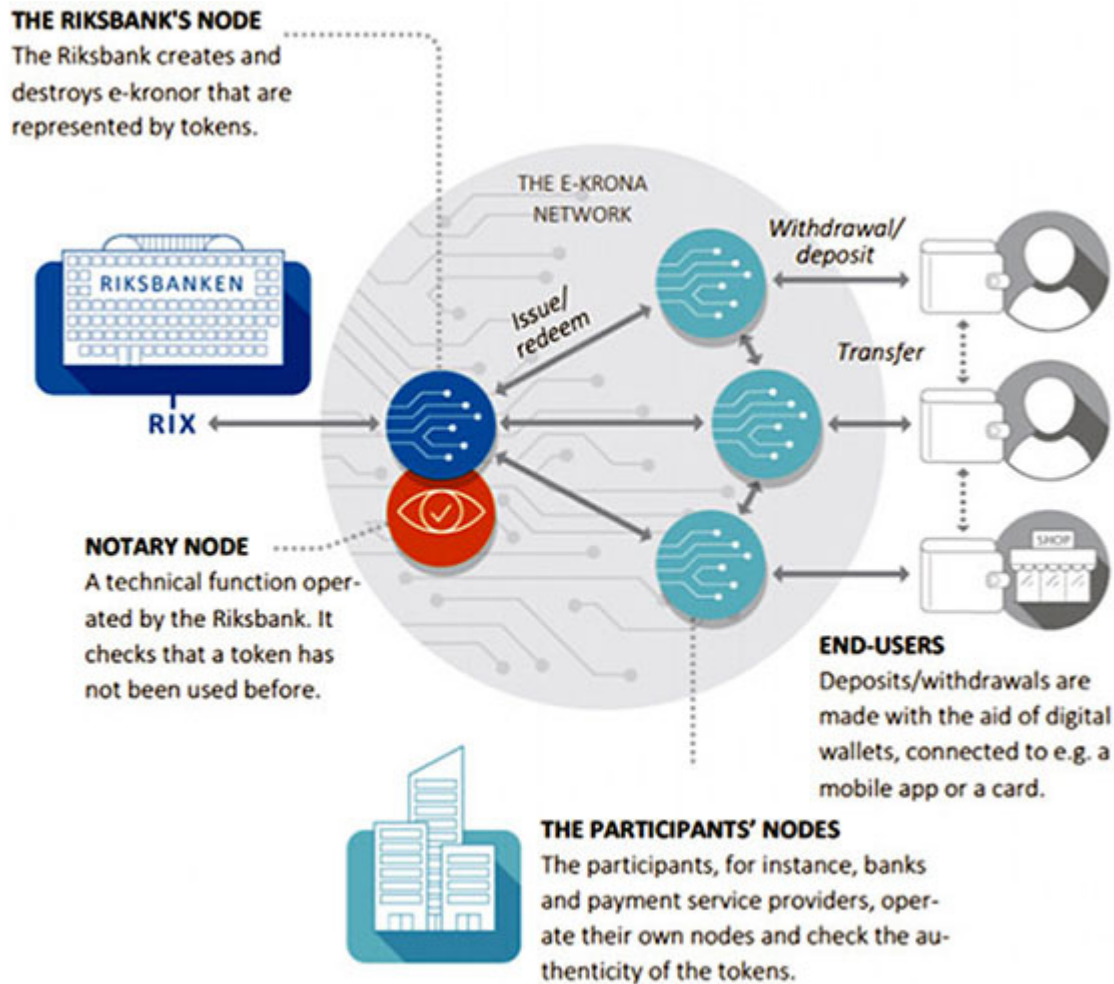


Figure 18.7: CBDC architecture for Sweden's Riksbank
(Source: Riksbank)

Please note that while designing CBDCs, we also have to think of handling the ownership of digital money, and hence, the digital identity of individual and organizations too is paramount. Decentralized digital identity is a highly researched area, especially for background verification and KYC checks.

18.3.3 Research on CBDC

CBDC is a very hot topic for many countries across the globe, and over 80% of the central banks of the world are researching on CBDC at the moment. Some of the most popular ones in this space are as follows:

- Not the Europe or USA, but the Bahamas became the first country in the world to adopt a CBDC, the Sand Dollar, in 2020.
- China's central bank, People's Bank of China (PBOC) has already created its CBDC project called "Digital Yuan" or eCNY and 140 million people had opened wallets for the digital Yuan. China's digital Yuan once live can impact the \$27 trillion payment market in the country. However, please note that the eCNY does not use any Blockchain or DLT solution as of now, although this space is currently under research.
- Nigeria's CBDC pilot called D'Cash is now the second largest behind China's digital Yuan. The new currency is to be called eNaira. It has now become the third country in the world and the first in Africa to build its own CBDC.
- Riksbank, the central bank of Sweden is one of the first in implementing CBDC. The project e-krona is already on pilot using R3's Corda platform.

The status of all CBDC projects can be tracked on the following website:

<https://cbdctracker.org/>

18.4 Summary

In this chapter, we covered the following topics:

- Concepts of decentralized finance, tokenization, and NFT, as well as its craze.
- Central bank digital currency and its advantages over crypto and Stablecoins.
- Different types of CBDC.
- The Blockchain platforms that are mostly used in building DeFi, tokens, NFTs, and CBDC.
- Some of the leading CBDC projects in the world.

References

- A step-by-step guide to NFTs for Creators - <https://creatoreconomy.so/p/guide-to-nfts-for-creators>
- Hyperledger Global Forum Highlights: CBDCs, programmable money and interoperability – Part I - <https://www.hyperledger.org/blog/2021/07/14/hyperledger-global-forum-highlights-cbdcs-programmable-money-and-interoperability-part-i>
- Riksbank begins cooperation with external participants in e-krona pilot - <https://www.enterprisetimes.co.uk/2021/06/09/riksbank-begins-cooperation-with-external-participants-in-e-krona-pilot/>
- China's CBDC Has Been Used for \$9.7B of Transactions - <https://www.coindesk.com/business/2021/11/03/chinas-cbdc-has-been-used-for-97b-of-transactions/>
- Nigeria Joins Bahamas and China In The CBDC Club - <https://tokenist.com/nigeria-follows-bahamas-and-china-with-cbdc-rollout/>
- Hedera Hashgraph integrates EMTECH CBDC infrastructure - <https://finance.yahoo.com/news/hedera-hashgraph-integrates-emtech-cbdc-103140489.html>
- Explainer: How does China's digital yuan work? - <https://www.reuters.com/article/us-china-currency-digital-explainer-idUSKBN27411T>

Quiz

Q1. Which among the following is the correct definition of CBDC?

- A. A digital currency
- B. Issued by the central bank
- C. Universally accessible
- D. All of the above

Q2. What is the advantage of CBDC over Stablecoins?

- A. Stablecoins are issued by private financial institutes, whereas CBDC is issued by the central bank.
 - B. CBDC is a legal tender, Stablecoins are not
 - C. CBDC can be strictly regulated by the monetary authorities of a nation, Stablecoins cannot be.
 - D. All of the above
- Q3.** Which among the following CBDC projects does not use a Blockchain or DLT solution?
- A. China's eCNY
 - B. Bahamas Sand Dollar
 - C. Sweden's e-krona
 - D. None of these
- Q4.** Which among the following categories of CBDC falls under retail domain?
- A. Direct CBDC
 - B. Indirect CBDC
 - C. Hybrid CBDC
 - D. All of the above
- Q5.** Which among the following is a CBDC project in production?
- A. China's eCNY
 - B. Bahamas Sand Dollar
 - C. Sweden's e-krona
 - D. None of these

Answers

1. D 2. D 3. A 4. D 5. B

CHAPTER 19

Blockchain and 5G for IoT

“New ways to improve the supply chain: Artificial Intelligence, IoT, Big Data, Augmented Reality, Blockchain, Drones, Machine Learning” – SupplyChainToday.com

In this chapter, we will explore why Blockchain adoption is paramount in NextGen IoT adoption with an example of a use case.

Just like Blockchain, IoT is another emerging technology that has observed exponential growth in the past one decade. IoT stands for “Internet of Things” that refers to a group of technologies that helps physical objects to interact with each other through the sensors attached to them. Though the sensor technology has been there for ages, due to the emergence of improved connection ecosystem and wide coverage of the Internet, this technology has reached a new level in the recent time.

19.1 How IoT Works

Before discussing about the latest innovations on IoT, it's good to know its basic functionalities. However, if you need to understand it in greater details, you can refer to any good book on IoT.

The Internet of Things (IoT) is a term coined in 1999 by a British scientist named Kevin Ashton to refer to a group of connecting objects in the physical world to the Internet. The technology that is used for this connecting network is Radio-frequency identification or RFID that uses electromagnetic fields to automatically identify and track the tags attached to the objects.

IoT is a software that connects the edge hardware, access points, and data networks to the other end, which is usually the end-user application. The IoT Platform is a multi-layer technology which is used to manage and automate the connected devices. In other

words, it is a service which helps you to bring the physical objects online. This platform will provide you with the services to connect the devices for a machine-to-machine communication. There is no specific architecture for IoT; it depends on the solution you are looking for. For any type of architecture, the following four components need to be present:

- Sensors/actuators
- Data Acquisition Systems/Internet Gateway
- Edge IT
- Data Centre/Cloud

The data that the IoT sensors collect has to pass through the following four stages, as represented in [Table 19.1](#) as follows:

Stage I – Data Collection	Stage II – Data Aggregation & Conversion	Stage III – Data Pre-Processing	Stage IV – Data Management
Data is collected through sensors and actuators	Data from sensors are collected by WIFI or wired LANs and converted into useful information	Edge IT system does more analysis of data using machine learning and visualization technologies	Data passed on to the data centre or cloud-based system

Table 19.1: States of data processing using IoT

Some of the top IoT Platforms are Google Cloud Platform, OpenRemote, IRI Voracity, Particle, ThingWorx, IBM Watson IoT, Amazon AWS IoT Core, Microsoft Azure IoT Suite, Oracle IoT, Cisco IoT Cloud Connect, Altair SmartWorks, Salesforce IoT Cloud etc.

19.2 Nextgen IoT

For any emerging technology, there is always a room for improvement.

International Data Corp. (IDC) predicts that, in 2025, 75% of the 55.7 billion devices will link to an IoT platform of some sort.

As per Forbes, “The Internet of Things (IoT) has become ubiquitous, and the number of connected devices is expected to grow to 29.3 billion by 2023.”

Obviously, for any IoT ecosystem, some of the biggest issues can be privacy, security, and scalability. For billions of devices, we need a super-fast ecosystem which would be availed by the 5G network. At the same time, the underlying DLT network can handle the issues of transparency, immutability, and trust with cryptography.

Also, it would be a great challenge for the centralized servers to manage such huge load for authentication and access management. Decentralized identity backed by Blockchain or DLT can be pretty useful to assign identity to every machine, whether big or small, and at the same time, authenticating them to any third party. This mechanism would ensure that a device can be identified by millions of other devices and servers leading to an efficient and secure machine-to-machine or M2M communication.

Hyperledger Fabric has recently come up with a paper where they have demonstrated how decentralized identity for the IoT machines can improve security, network quality assurance, customer experience, visibility, transparency, data access, and user content in the IoT devices.

[Figure 19.1](#) shows the integration of Blockchain components with various BSS/OSS systems in the Telecom domain, such as order management, CRM, billing, user and enterprise dashboards, as shown as follows:

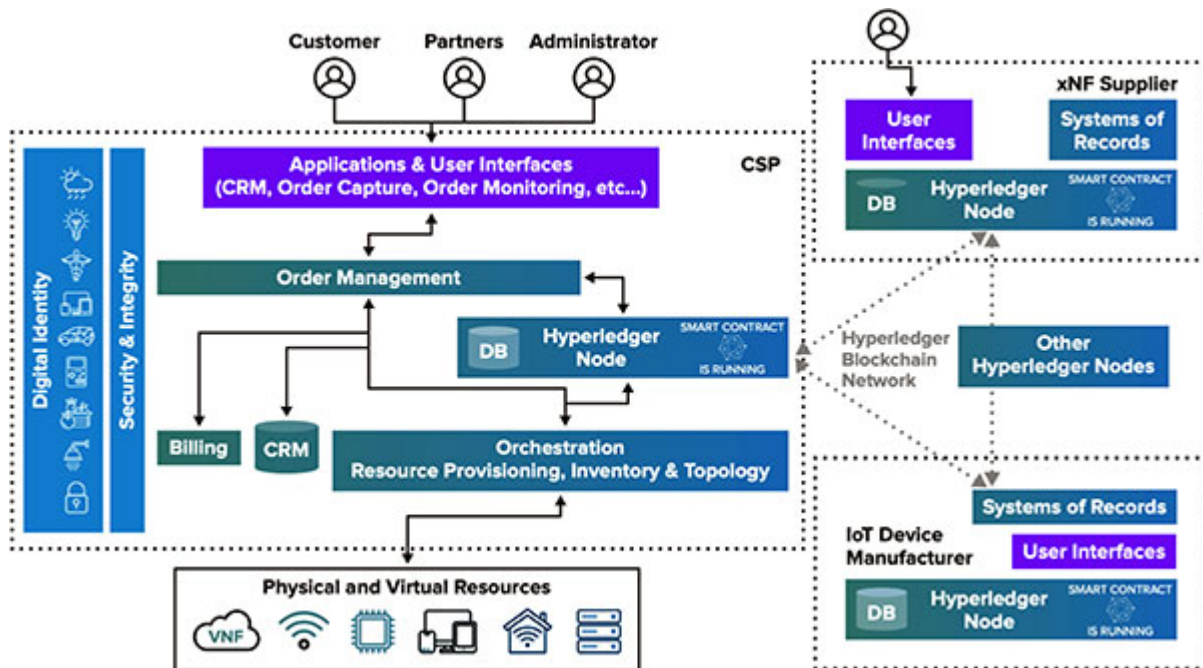


Figure 19.1: Decentralized ID and Access Management (DIAM) for IoT Networks
 (Source: https://www.hyperledger.org/wp-content/uploads/2021/02/HL_LFEdge_WhitePaper_021121_3.pdf)

This implements the end-to-end lifecycle management of the IoT devices and user/application data.

19.3 Summary

In this chapter, we covered the following topics:

- Brief introduction to IoT.
- Significance of Blockchain in IoT and 5G adoption.

References

- Distributed Secure Computing for Smart Mobile IoT Networks - <https://www.hindawi.com/journals/misy/2020/8889192/>
- 5G and Identity - <https://www.kuppingercole.com/blog/small/5g-and-identity>
- Decentralized ID and Access Management (DIAM) for IoT Networks - <https://www.lfedge.org/2021/02/25/decentralized-id-and-access-management-diam-for-iot-networks/> &

https://www.hyperledger.org/wp-content/uploads/2021/02/HL_LFEdge_WhitePaper_021121_3.pdf

- Distributed Ledger Technology's Role in the Future of IoT - <https://www.iotforall.com/distributed-ledger-technologys-role-in-the-future-of-iot>
- Decentralized Identity & Access Management in IoT || Vienna Digital Identity Conference - <https://www.youtube.com/watch?v=8pyrhUdQe1E>
- Blockchain Applications for Telecom Industry & Regulatory Authorities - <https://www.wipro.com/blockchain/blockchain-applications-for-telecom-industry-and-regulatory-authorities/>
- Market Landscape: Blockchain for Telecoms: <https://www.oracle.com/a/ocom/docs/corporate/analystrelations/ovum-blockchain-for-telecoms.pdf>

PHASE V – Blockchain Adoption and Future

When it comes to the emerging technologies, no one has a crystal ball. Forget about the next five or ten years, it's difficult even to predict how the market will shape up in the next few months. Hence, in the highly volatile field, where the customer's demands are evolving fast, it's a good idea for organizations to stay tuned to the latest and reinvent their offerings. Remember, in the emerging technologies, it's the quickest who wins the race.

CHAPTER 20

Production and Beyond

“Blockchain technology could change our world more than people imagine.” – Jack Ma

In comparison to the other emerging technologies, Blockchain has had a much quicker journey to production. However, the number of such solutions are far too less and even many fortune 500 organizations are either facing hurdles or are not sure how to find commercial benefits from this technology. In this chapter, we will try to explore the hurdles in implementing the Blockchain projects and how we can overcome them.

20.1 Challenges in Blockchain projects

Some of the lingering issues in starting a Blockchain project are as follows:

Awareness: Irrespective of the fact that the industry has spent millions on the Blockchain technology, only a handful have reached production. Many people in the industry don't understand or have even heard of the Blockchain technology, and hence, don't understand the difference between public Blockchain or private DLTs. Some people are still living in a world where they believe Blockchain and Bitcoin are synonyms.

Technical expertise: Many technical teams get bogged down and the development faces serious issues if there is lack of expertise in the team. Skill shortage is a perennial issue in any emerging technology.

Monetization: There is serious lack of clarity on the return of investment in the Blockchain projects. Most CXO level people, who are the decision makers, always hesitate to invest in Blockchain as they are not very sure as to what extent the existing processes can

improve with this emerging technology. Also, many use cases may seem catchy in the beginning; however, the monetization models are not clear. Hence, all the parties involved might not show the required interest in participation.

While training and awareness campaigns would be useful, they are not sufficient to address the preceding issues. Emerging technologies evolve pretty fast, and by the time the developers are trained, most training materials might become outdated. Hence, one needs to constantly keep an eye on the latest.

20.2 Blockchain CoE Set-Up

While there have been early adopters of the Blockchain technology, many organizations today are still struggling to set up their “Blockchain Center of Excellence” or CoE where they can form teams of Blockchain experts to evaluate the possible opportunities in this space. As the competition is tough, it’s an absolute mandate to expedite the organization’s Blockchain adoption journey, so that the solutions can reach the market quickly, cost-effectively, and successfully.

20.2.1 Accelerators in Blockchain adoption

The following accelerators can also work as a start-up kit for a Blockchain CoE:

- Experts and Advisors
- Consortiums, partnership, events & networking
- Certifications
- Evaluation of Platforms
- Use frameworks and low-code platforms (i.e., DAML)
- Create reference architecture
- Build reusable components
- Work on differentiators

20.2.2 Roles

The roles that a Blockchain CoE can possibly have are as follows:

- Blockchain architects
- Blockchain and smart contract developers
- DevOps and cloud expert
- Business analyst
- Web developer
- Quality engineer

20.2.3 Skills

Also, when it comes to development skills, apart from the Blockchain or DLT platform that one can learn, knowledge in the following areas is needed:

- Cloud
- Docker
- Kubernetes
- Cryptography
- Imagination

20.2.4 Factors to Consider for Production

Many Blockchain projects fail as the managers and architects do not consider all the aspects of the underlying protocol, especially until they enter production. The previous chapters have given the reader a good amount of guidance on different Blockchain types along with the comparison of their features. On the top of that, the following factors have to be considered before finalising a Blockchain protocol:

- Supported cloud platforms
- Cost of licenses in production

It's true that there are organizations as IBM, EY, TCS, Wipro, Accenture etc., that already have their Blockchain CoE running actively for the last five years. However, the emerging technologies

always have the edge over the traditional tech and one can always start afresh using the right strategy of CoE set-up.

20.3 Blockchain Standards

Standards play a major role in engaging the organizations in adoption with the much needed guidance. They also lead to interoperability between different parties and help business to grow together. Apart from this, the Blockchain projects also need to refer to such standards in order to handle other factors like security, governance, identity, and smart contracts. The following are a list of such standards which are set up by some of the biggest Blockchain communities in the world.

20.3.1 W3C

W3C is one of the biggest community in Blockchain. As per their website, “The mission of the Blockchain Community Group is to generate the message format standards of Blockchain based on ISO20022 and to generate guidelines for usage of storage including torrent, public Blockchain, private Blockchain, side chain and CDN”.

20.3.2 Open Standards

This is a body formed with the combination of IEEE, the Organisation for the Advancement of Structured Information Standards (OASIS) and the Internet Engineering Task Force (IETF). IEEE is working on the standards of IoT along with Blockchain.

20.3.3 European Standardization Organization

It's again a joint venture between European Telecommunications Standards Institute (ETSI, in particular the ISG PDL), the European Committee for Standardisation (CEN), European Committee for Electrotechnical Standardization (CENELEC) etc., for bringing common standards to Blockchain in Europe.

20.3.4 ISO/TC 307

It's another initiative for standards in Australia that works on building guidance on architecture and taxonomy, security and privacy, smart contracts, governance, and interoperability between Blockchains.

20.4 Quantum Computing, The Fear Factor

Just like Blockchain, Quantum Computing is another emerging technology that uses special devices called quantum computers to performs complex. Quantum computing uses quantum bits or qubits, which is also represented as a 0 or a 1 or as a mixed state called a superimposed state. With additional properties as interference, and entanglement etc., it can perform calculations millions of times faster than the traditional computers. With such high speed, there is a possibility that the quantum computers of the future would be able to break the encryption that is the key of Blockchain technology. This is especially more problematic for the public Blockchains as anyone can join and participate in the transactions, and this has raised fear among the crypto users lately. Hence, many public Blockchains are researching to upgrade to a quantum resistant version in the future.

As per Anthony Day, Blockchain leader at IBM, "Number of Blockchain and Crypto users is growing at the EXACT same speed as the Internet did".

It's astonishing that the industry started Blockchain adoption only since 2015 mostly with ICOs. Once the crazy rush of ICOs waned down, most organizations started investing in private permissioned DLTs that resulted in hundreds of such DApps running in production. As this space slowly got into saturation, new trends as NFT, CBDC, and Decentralized Identity emerged in the DLT space. Private clouds, decentralized web, integration with drones etc., are also some of the spaces that need to be researched for a possible implementation in the future. Blockchain is here to stay, perhaps in a new name and with new features.

20.5 Summary

In this chapter, we covered the following topics:

- Challenges in Blockchain adoption.
- Right strategy for Blockchain CoE set up and moving to production.
- Blockchain standards and their importance in the emerging tech space.

References

- Public Vs. Private Permissioned Ledgers and Blockchain Standards - <https://www.forbes.com/sites/forbestechcouncil/2019/06/11/public-vs-private-permissioned-ledgers-and-blockchain-standards/>
- Blockchain standards - <https://digital-strategy.ec.europa.eu/en/policies/blockchain-standards>

Glossary

Some of the widely used words in the Blockchain ecosystem are as follows:

5G: 5th generation mobile network

API: Application programming interface

BFSI: Banking, financial services, insurance sectors

BTC: Bitcoin

CBDC: Central bank digital currency

Crypto: cryptocurrency

Cryptography: A field of science that deals with secure communication using some enigmatic code that any third party can't decipher

DAG: Directed Acyclic Graph

DAML: Digital Asset Modelling Language or DAML which is an open source programming language for 10x faster development with smart contracts

DApp: Decentralized Application

DDoS: Distributed denial of service

DeFi: Decentralized Finance

DEX: Decentralized exchange or marketplace where peer-to-peer trading occurs between crypto traders

DLT: Distributed Ledger Technology

FIAT: Traditional currency (e.g. US Dollar, Euro) issued by the government

ICO: Initial Coin Offering

IoT: Internet of Things

KYC: Know Your Customer

OTP: One time password

PoS: Proof of Stake

PoW: Proof of work

SSI: Self Sovereign Identity

TPS: Transactions Per Second

Please note that some of these keywords are not used in the book; however, they are provided for your reference.

Index

A

Aave [206](#)

abstract contract [104-106](#)

abstraction [101](#)

address, types

 address payable [65](#)

 static address [65](#)

Algorand

 about [184](#)

 development [186](#)

 Layer-2 solutions [186](#)

 projects, exploring [187](#)

 Proof of Stake consensus model [184](#)

 scalability and performance [186](#)

 transaction fees [186](#)

 wallets [186](#)

Algorand PPoS consensus, stages

 block proposal [184](#), [185](#)

 certify vote [185](#)

 soft vote [185](#)

Altcoins [121](#)

Apache Kafka [237](#)

Application Binary Interface (ABI)

 about [51](#)

 interface, defining [94](#)

arithmetic operator [68-70](#)

array [65](#), [66](#)

assembly language [48](#)

assignment operator [74](#), [75](#)

asymmetric cryptography

 about [7](#)

 keys [7](#)

automated market maker (AMM) [206](#)

Avalanche

 Blockchain [196](#)

 development [198](#)

 Layer-2 solutions [198](#)

- projects, exploring [199](#)
- Proof of Stake consensus model [197](#)
- scalability and performance [198](#)
- transaction fees [198](#)
- wallets [199](#)

Avalanche Virtual Machine (AVM) [196](#)

B

Bitcoin

- about [4](#), [28](#), [29](#)
- versus Previous Generation Electronic Money [5](#)

Bitcoin Blockchain

- working [19](#), [20](#)

bitwise operator [73](#)

Blockchain

- key concepts [5](#)

Blockchain Center of Excellence (CoE)

- accelerators, in Blockchain adoption [321](#)
- factors [322](#)
- roles [321](#)
- setting up [320](#)
- skills [321](#)

Blockchain market [1](#), [2](#)

Blockchain project

- challenges [319](#), [320](#)

Blockchain projects [120](#), [121](#)

Blockchain protocols, areas

- about [29](#)
- consensus model [29](#), [30](#)
- Layer-2 solutions [31](#)
- scalability and performance [30](#)
- transaction fees [30](#)

Blockchain Standards

- about [322](#)
- European Standardization Organization [323](#)
- ISO/TC [307](#) [323](#)
- open standards [323](#)
- W3C [322](#)

Blockchain technology

- evolution [2](#)

Blockchain, use case

- about [25](#)

- private/permissioned Blockchain, need for [27](#)
- public Blockchain, need for [26](#)
- RDBMS, need for [26](#)
- Boolean [63](#)
- Bridge Chain [211](#)
- byte [63](#)

C

- Caliper tool [241](#)
- calldata [81](#)
- Cap Theorem [27](#)
- Cardano
 - about [178](#)
 - development [180](#)
 - Layer-2 solutions [179](#)
 - projects, exploring [181](#), [182](#)
 - Proof of Stake consensus model [178](#)
 - scalability and performance [180](#)
 - transaction fees [178](#)
 - wallets [181](#)
- Cardano language
 - Marlowe [180](#)
 - Plutus [180](#)
 - Rosetta [181](#)
- central bank digital currency (CBDC)
 - about [302](#)
 - Blockchain & DLT Platform [305](#)
 - Blockchain & DLT Platforms [306](#)
 - research [306](#), [307](#)
- central bank digital currency (CBDC), types
 - about [302](#)
 - retail CBDC [303](#)
 - wholesale CBDC [303](#)
- centralized identity [280](#), [281](#)
- centralized system
 - about [11](#)
 - advantages [12](#)
 - disadvantages [12](#)
- Chaincode [237](#)
- channel [236](#)
- conditional operator [76](#)
- consensus model

- about [46](#)
- Delegated Proof of Stake (DPOS) [14](#)
- Direct Acyclic Graph (DAG) [15](#)
- Practical Byzantine Fault Tolerance (PBFT) [14](#), [15](#)
- Proof of Stake (PoS) [14](#)
- Tendermint [15](#)
- with proof of work [22](#)
- consensus models, types
 - Proof of Work (PoW) [13](#)
- consensus process [13](#)
- Consensys Quorum
 - deployment, on cloud [264](#)
 - development [264](#)
 - enclave [261](#)
 - end-to-end transaction flow [261](#), [263](#)
 - industry adoption [263](#)
 - key concept [258](#)
 - privacy manager [260](#)
 - RAFT algorithm [260](#)
 - transaction manager [261](#)
 - versus Hyperledger Fabric [268](#), [269](#)
 - versus R3 Corda [268](#), [269](#)
- constant state variable [67](#)
- contract
 - creating [108](#)
 - destroying [108-110](#)
- Contract Chain (C-Chain) [196](#)
- Corda distributed application (CordApp) [249](#), [250](#)
- Cosmos
 - development [219](#)
 - Layer-2 solutions [219](#)
 - projects, exploring [220](#)
 - Proof of Stake Tendermint consensus model [216-218](#)
 - scalability and performance [219](#)
 - transaction fees [219](#)
 - wallets [219](#)
- Cosmos Blockchain network
 - Cosmos SDK [217](#)
 - Inter Blockchain Communication (IBC) [216](#)
 - Tendermint [216](#)
- Cosmos network, layer
 - application layer [216](#)
 - consensus layer [216](#)

- networking layer [216](#)
- cryptocurrency wallets (crypto wallets) [9](#)
- cryptography
 - about [6](#)
 - phases [6](#)
 - types [6](#)
- crypto price
 - about [36](#)
 - hoarding [36](#)
 - limited supply, versus unlimited supply [36](#)
 - utility [36](#)
- crypto token
 - types [298](#)
- custodial wallets [11](#)

D

- data
 - storing, in Solidity [79](#)
- data hiding [100](#)
- data sharing
 - modes [287](#)
 - types [287](#)
- data types
 - about [62](#)
 - constant state variable [67](#)
 - immutable state variable [67](#)
 - reference type [65](#)
 - value type [62](#)
- decentralization
 - about [11](#)
 - centralized system [11](#)
 - decentralized system [12](#)
 - distributed system [12](#)
- Decentralized Applications (DAPPS) [22](#)
- decentralized finance (DeFi) [297](#), [298](#)
- decentralized identity (DID)
 - about [284-286](#)
 - architecture [285](#)
 - for privacy program [290-292](#)
 - products [292](#)
 - use cases [287](#), [288](#)
- decentralized identity (DID), use cases

- remote voting [289](#)
- smart agriculture [289](#), [290](#)
- decentralized system
 - about [12](#)
 - advantages [13](#)
 - disadvantages [13](#)
- Decryption [6](#)
- Delegated Proof of Stake (DPOS) [14](#)
- desktop wallets [10](#)
- digital identity [277-279](#)
- digital identity, evolution
 - about [280](#)
 - centralized identity [280](#), [281](#)
 - decentralized identity (DID) [284](#), [286](#)
 - federated identity [281](#), [282](#)
 - user centric identity [282](#), [283](#)
- Digital Signature [7](#)
- Digital Signature Algorithm (DSA) [7](#)
- Direct Acyclic Graph (DAG) [15](#)
- distributed denial of service (DDoS) attack [24](#)
- distributed system
 - about [12](#)
 - advantages [12](#)
 - disadvantages [12](#)
- double spending [23](#)

E

- ECMAScript [48](#)
- Ed25519 [8](#)
- Edward curve Cryptography [8](#)
- Edwards-curve Digital Signature Algorithm (EdDSA) [8](#)
- elliptic curve cryptography (ECC) [8](#)
- Elliptic Curve implementation of DSA (EDSA) [8](#)
- encapsulation [100](#)
- Encryption [6](#)
- ERC20
 - about [116](#), [117](#)
 - events [117](#)
 - functions [116](#), [117](#)
 - tokens [117](#)
- ERC standards [115](#)
- error handling [89](#)

error handling, exceptions

- assert function [92](#)

- require function [90](#)

- revert function [91](#)

- try/catch function [92](#)

Ether

- about [46](#), [77](#)

- versus Gas [78](#)

Ethereum

- development [47](#)

- performance [47](#)

- scalability [47](#)

Ethereum network

- transaction fee [46](#)

Ethereum Request for Comments (ERC) [115](#)

Ethereum tokens [115](#)

Ethereum Virtual Machine (EVM) [46](#)

Etherum network

- Solidity operator, executing [77](#)

events

- about [94-97](#)

- logs, in Solidity [97-99](#)

- output [99](#)

Exchange Chain (X-Chain) [196](#)

F

fallback function [89](#)

Federal Information Processing Standard (FIPS) [9](#)

federated identity [281](#), [282](#)

function overriding [101](#)

fungible tokens [298](#)

G

Ganache tool [118-120](#)

Gartner hype cycles [2](#)

Gas

- future [79](#)

- versus Ether [78](#)

Gas limit [78](#)

Gas price [78](#)

global variable [61](#)

gossip about gossip [169](#)
Gossip Network for Message Propagation [17](#)

H

hard fork [18](#)
hardware wallets
 about [10](#)
 custodial wallets [11](#)
 non-custodial wallet [10](#)
 paper wallets [10](#)
Hashgraph [168](#)
hashing [9](#)
HBAR token [170](#)
Hedera concepts
 gossip about gossip [168](#)
 virtual voting [168](#)
Hedera Hashgraph
 about [15](#)
 development [171](#)
 Layer-2 solutions [171](#)
 projects, exploring [171](#), [172](#)
 Proof of Stake consensus model [168](#)
 scalability and performance [171](#)
 transaction fees [170](#)
 wallets [171](#)
Hub and Spoke architecture [217](#)
Hyperledger Fabric
 development [241](#)
 development, on cloud [241](#)
 industry adoption [241](#)
 interoperability, with Oracle [240](#)
 private data [239](#), [240](#)
 scalability and performance [241](#)
 versus Consensus Quorum [268](#), [269](#)
 versus R3 Corda [268](#), [269](#)
Hyperledger Fabric consensus process
 about [237](#), [238](#)
 components [237](#)
 ordering [238](#)
 proposal and endorsement [238](#)
 validation & commit [239](#)
Hyperledger Fabric, key concepts

- about [234](#)
- channel [236](#)
- identity [235](#)
- ledger [234](#)
- membership service provider (MSP) [235](#)
- orderer [236](#)
- peers [235](#), [236](#)
- policies [235](#)
- smart contract [237](#)

I

- immutable state variable [67](#)
- inheritance [101](#), [102](#)
- integers [62](#)
- interfaces [104-107](#)
- Internet of Things (IoT)
 - working [312](#)
- interoperable Blockchains
 - comparison [223](#)
 - use cases [225](#)

J

- Java [47](#)

L

- Lab1 [110](#)
- Lab2 [112](#)
- Layer-2 solutions
 - about [31](#), [46](#)
 - Parachains [32](#)
 - rollups [34](#), [35](#)
 - sharding [31](#)
 - sidechains [33](#), [34](#)
 - state channels [32](#)
- Liquid Proof of Stake [174](#)
- local variable [60](#)
- logical operator
 - about [72](#)
 - bitwise operator [73](#)
- low level language (LLL)

versus Solidity [48](#)

M

mapping [67](#)

market capitalization

about [37](#)

need for [37](#)

Marlowe [180](#), [181](#)

Merkle root [19](#)

mining [20](#), [21](#)

mobile wallets [10](#)

monetary value [47](#)

N

National Institute of Standards and Technology (NIST) [9](#)

native token [16](#), [17](#)

Nextgen IoT [313](#), [315](#)

nodes [15](#)

Nominated Proof of Stake consensus model, types

collator [212](#)

nominator [212](#)

validator [212](#)

non-custodial wallet [10](#)

non-fungible tokens (NFT)

about [297](#), [298](#)

Blockchain & DLT Platform [299](#)

Blockchain & DLT Platforms [300](#)

for digital contents [299](#)

O

object-oriented language [48](#)

object-oriented programming (OOP) [100](#)

online wallets [9](#)

OOP concepts

abstraction [101](#)

encapsulation [100](#)

inheritance [101](#), [102](#)

polymorphism [104](#)

orderer service

Apache Kafka [237](#)

Raft [237](#)
Solo [237](#)
Ouroboros protocol [178](#)
override keyword [101](#)

P

paper wallets [10](#)
Parachain [32](#), [211](#)
Parachain Slots [212](#)
Peg Zone [218](#)
permission levels [25](#)
Platform Chain (P-Chain) [196](#)
PlotX [206](#)
Plutus [180](#)
Polkadot
 about [210](#)
 development [213](#)
 Layer-2 solutions [213](#)
 Nominated Proof of Stake consensus model [210](#)
 projects, exploring [214](#)
 scalability and performance [213](#)
 transaction fees [212](#)
 wallets [214](#)
Polygon
 development [205](#)
 Layer-2 solutions [205](#)
 projects, exploring [206](#)
 Proof of Stake consensus model [204](#), [205](#)
 scalability and performance [205](#)
 transaction fees [205](#)
 wallets [206](#)
polymorphism [104](#)
Practical Byzantine Fault Tolerance (PBFT) [14](#), [15](#)
Previous Generation Electronic Money
 versus Bitcoin [5](#)
Proof of History consensus model [191](#)
Proof of Stake (PoS) [14](#)
Proof of Work (PoW)
 about [13](#), [21](#)
 using, in consensus [22](#)
public Blockchains
 comparison [221](#), [223](#)

- DDoS attack [24](#)
- double spending [23](#)
- overview [225](#), [226](#)
- sybil attack [23](#), [24](#)
- threats & challenges [23](#)
- pure function [88](#)

Q

- Quantum Computing [323](#), [324](#)
- QUICK cryptocurrency [206](#)
- QuickSwap [206](#)
- Quorum [258](#)

R

- R3 Corda
 - accounts model [251](#), [252](#)
 - attachment [248](#)
 - consensus [249](#)
 - contracts [248](#)
 - deployment, on cloud [255](#)
 - development [255](#)
 - end-to-end transaction flow [250](#)
 - industry adoption [254](#)
 - interoperability, with Oracle [248](#)
 - ledger [245](#)
 - network [253](#)
 - notary [247](#), [248](#)
 - opensource, versus enterprise [253](#), [254](#)
 - scalability and performance [254](#)
 - state [245](#), [246](#)
 - time window [248](#)
 - token SDK [253](#)
 - transaction [246](#), [247](#)
 - versus Consensus Quorum [268](#), [269](#)
 - versus Hyperledger Fabric [268](#), [269](#)
- R3 Corda, key concepts
 - about [244](#)
 - ledger [244](#)
 - network [244](#)
- Raft [237](#)
- reference type

- about [65](#)
- array [65](#), [66](#)
- mapping [67](#)
- struct [66](#)
- relational operator [70](#), [71](#)
- Relaychain [211](#)
- REMIX browser [49](#)
- retail CBDC
 - about [303](#)
 - direct CBDC [304](#)
 - hybrid CBDC [304](#)
 - indirect CBDC [304](#)
- Rivest-Shamir-Adleman (RSA) [7](#)
- rollups [34](#), [35](#)
- Rosetta [181](#)
- Ruby [47](#)

S

- Scala [47](#)
- Serpent
 - versus Solidity [48](#)
- sharding [31](#)
- Shared Replicated Ledger [16](#)
- sidechains [33](#), [34](#)
- smart contract [16](#), [237](#)
- Snowman consensus protocol [196](#)
- soft fork [18](#)
- Solana
 - development [192](#)
 - Layer-2 solutions [192](#)
 - projects, exploring [193](#)
 - Proof of History consensus model [190](#), [191](#)
 - Proof of Stake consensus model [190](#), [191](#)
 - scalability and performance [192](#)
 - technologies [190](#)
 - transaction fees [192](#)
 - wallets [193](#)
- Solidity
 - about [47](#)
 - event logs [97-99](#)
 - features [48](#)
 - need for [47](#)

- object-oriented approach [100](#)
- versus low level language (LLL) [48](#)
- versus Serpent [48](#)

Solidity data

- calldata [81](#)
- memory [80](#), [81](#)
- stack [81](#), [82](#)
- storage [79](#), [80](#)
- storing [79](#)

Solidity flow control

- about [82](#)
- break statement [85](#)
- conditional operator [86](#), [87](#)
- continue statement [86](#)
- do-while loop [84](#)
- for loop [84](#)
- if-else condition [82](#), [83](#)
- while-do loop [83](#)

Solidity function

- about [55](#)
- constructor [56](#)
- getter() function [57](#)
- order [59](#)
- setter() function [57](#)
- value, returning [58](#)

Solidity function, types

- external [55](#)
- internal [55](#)
- private [56](#)
- public [55](#)

Solidity operator

- about [68](#)
- arithmetic operator [68-70](#)
- assignment operator [74](#), [75](#)
- conditional operator [76](#)
- executing, on Ethereum network [77](#)
- logical operator [72](#)
- relational operator [70](#), [71](#)

Solidity program

- compiling [50](#)
- deploying [50](#), [51](#)
- writing [49](#)

Solidity source code

- about [51](#)
- contract [53](#)
- element order [54](#)
- element order, in contract [55](#)
- events [54](#)
- file, importing [53](#)
- libraries [54](#)
- SPDX License Number [52](#)
- version [52](#)
- Solo [237](#)
- Stablecoins
 - about [300](#), [301](#)
 - advantages [302](#)
 - disadvantages [302](#)
- stack [81](#), [82](#)
- state channels [32](#)
- state variable [59](#)
- statically typed programming language [48](#)
- string literals [63](#)
- struct [66](#)
- Substrates template [213](#)
- sybil attack [23](#)
- symmetric cryptography [6](#)

T

- Tendermint [15](#)
- Tezos
 - development [175](#)
 - Layer-2 solutions [174](#)
 - Liquid Proof of Stake consensus model [174](#)
 - projects, exploring [175](#)
 - scalability and performance [174](#)
 - transaction fees [174](#)
- The Scalability Trilemma [27](#)
- tokenization [297](#)
- Truffle Suite [118](#)
- Truffle tool [118](#)

U

- user centric identity [282](#), [283](#)

V

value type

about [62](#)

address [65](#)

Boolean [63](#)

byte [63](#)

enum [64](#)

fixed number [63](#)

integers [62](#)

string literals [63](#)

variables

about [59](#)

global variable [61](#)

local variable [60](#)

state variable [59](#)

Vault [244](#)

verifiable random functions (VRFs) [185](#)

view function [87](#)

virtual keyword [101](#)

virtual voting [169](#)

Vyper [48](#)

W

wallets [9](#), [120](#)

wallets, types

desktop wallets [10](#)

hardware wallets [10](#)

mobile wallets [10](#)

online wallets [9](#)

wholesale CBDC [303](#)

World Wide Web Consortium [286](#)

Z

Zapper [206](#)

Zero-Knowledge-Proof [34](#)

Zero-Knowledge Rollups (ZK-Rollups) [34](#)