

Architecture Styles

6. Space-Based Architecture Style

- With increased users, **scaling the web server**, then **scaling app server** and finally scaling the DB server can minimize the performance degrade.
- This is relatively easy and inexpensive, However, in most cases of high user load, scaling out the web-server layer just moves the **bottleneck down to the application server**.
- **Scaling application servers** further moves the bottleneck down to the **database server**, which is even more difficult and expensive to scale creating **triangular shaped topology**, with the widest part of the triangle being the web servers (easiest to scale).

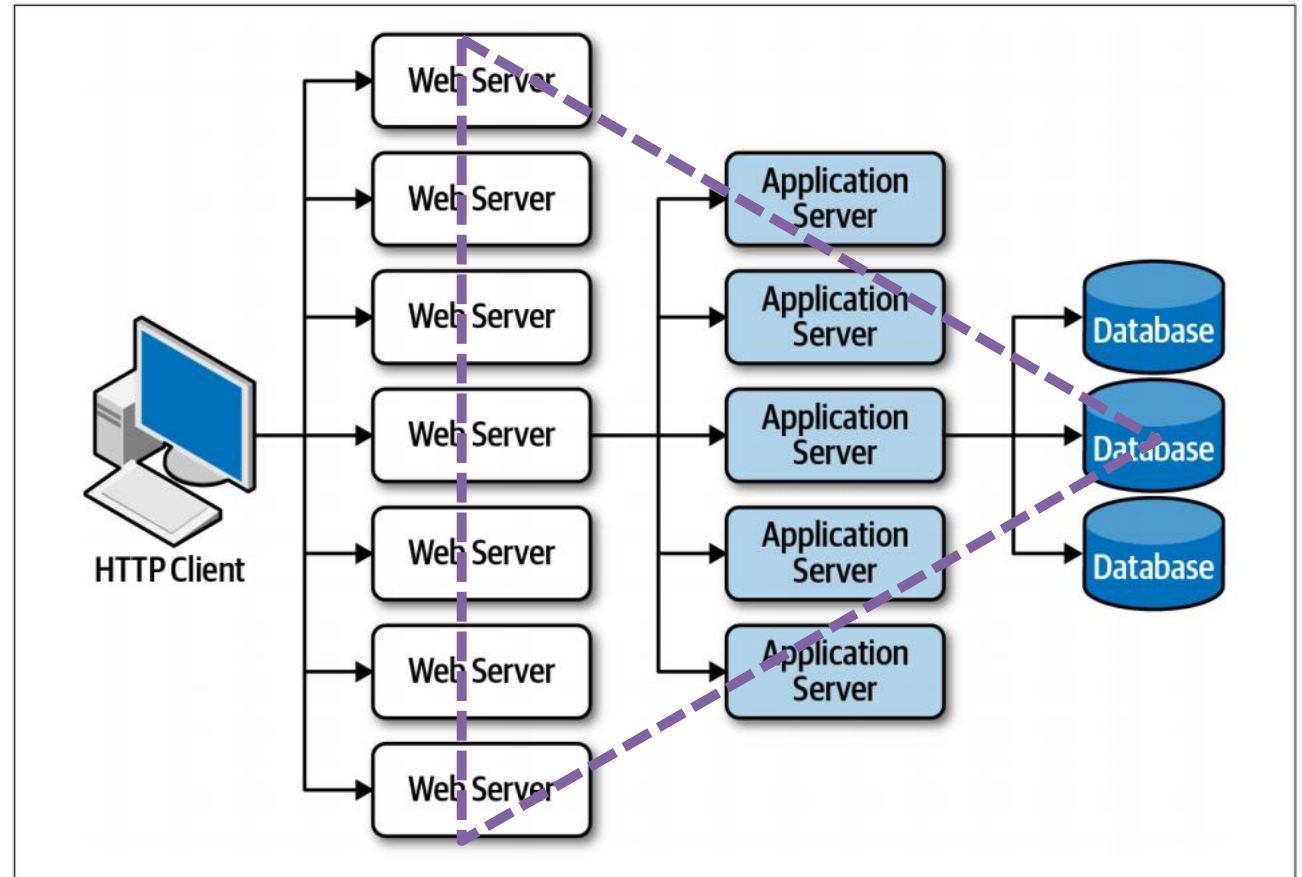
6. Space-Based Architecture Style

- Any application with a **large concurrent user** load, **concurrent transactions** limits database access speed.
- Caching and database scaling products help to address these issues, the fact remains that scaling out a normal application for extreme loads is a very difficult.
- **Space-based architecture style** is specifically designed to address problems involving high **scalability, elasticity, and high concurrency** issues.
- The **space-based architecture style** is specifically designed to address **high scalability, elasticity, and high concurrency** issues. Its also useful for applications that have variable and **unpredictable concurrent user** volumes.

6. Space-Based Architecture Style

Even if we can scale the database, what we eventually end up with is a **triangle shaped topology**,

- with the widest part of the triangle being the web servers (easiest to scale) and
- the smallest part being the **database** (hardest to scale),



Scalability limits within a traditional web-based topology

6. Space-Based Architecture Style

General Topology

- Space-based architecture leverage the technique of using **multiple parallel processors communicating through shared memory**
- High scalability, high elasticity, and high performance are achieved by removing the **central database** as a synchronous constraint in the system and instead **leveraging replicated in-memory data grids**.
- **Application data is kept in-memory** and replicated among all the active processing units. When a processing unit updates data, it asynchronously sends that data to the database, usually via **messaging with persistent queues**.
- Processing units start up and shut down dynamically as user load increases and decreases, thereby addressing variable scalability.
- There is no central database involved in the application, the database bottleneck is removed, thus providing near-infinite scalability within the application.

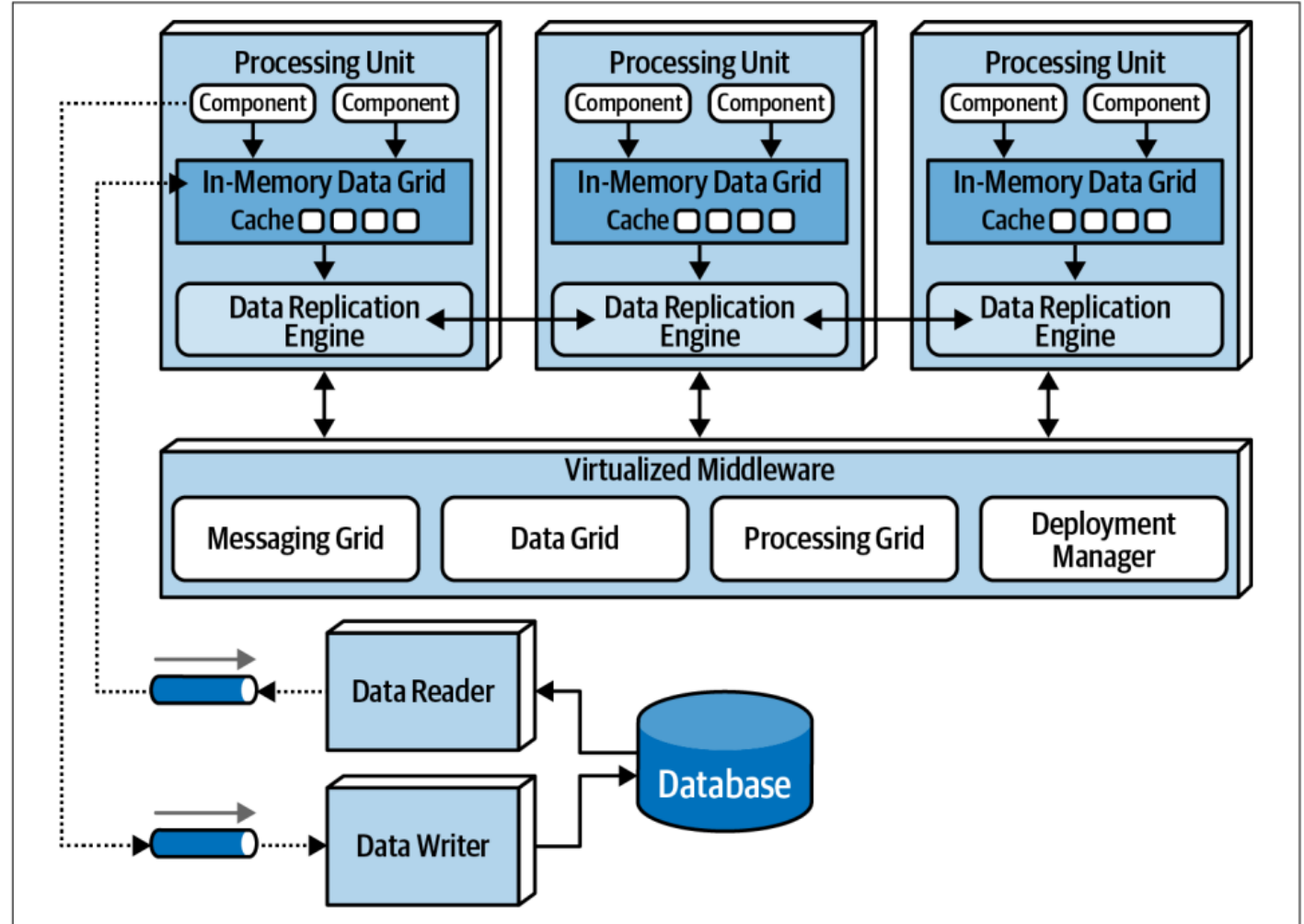
6. Space-Based Architecture Style

- There are several architecture components that make up a space-based architecture:
 - a processing unit containing the **application code**,
 - **virtualized middleware** used to manage and coordinate the processing units,
 - **data pumps** to asynchronously send updated data to the database,
 - **data writers** that perform the updates from the data pumps, and
 - **data readers** that read database data and deliver it to processing units upon startup.

6. Space-Based Architecture Style

Processing Unit

- The processing unit contains the application logic (web based component and back end business logic)
- Depending on the size of the functionality processing units may deploy all or part of the business logic.



Space-based architecture basic topology

6. Space-Based Architecture Style

Virtualized Middleware

- The **virtualized middleware** handles the **infrastructure concerns** within the architecture that control various aspects of data synchronization and request handling.
- The components in virtualized middleware include
 - **a messaging grid**— manages input request and session state.
 - **Data grid** — maintains the data across processing caches ,
 - **processing grid** — manages orchestrated request processing and
 - **deployment manager** — manages the dynamic startup and shutdown of processing unit instances based on load conditions.

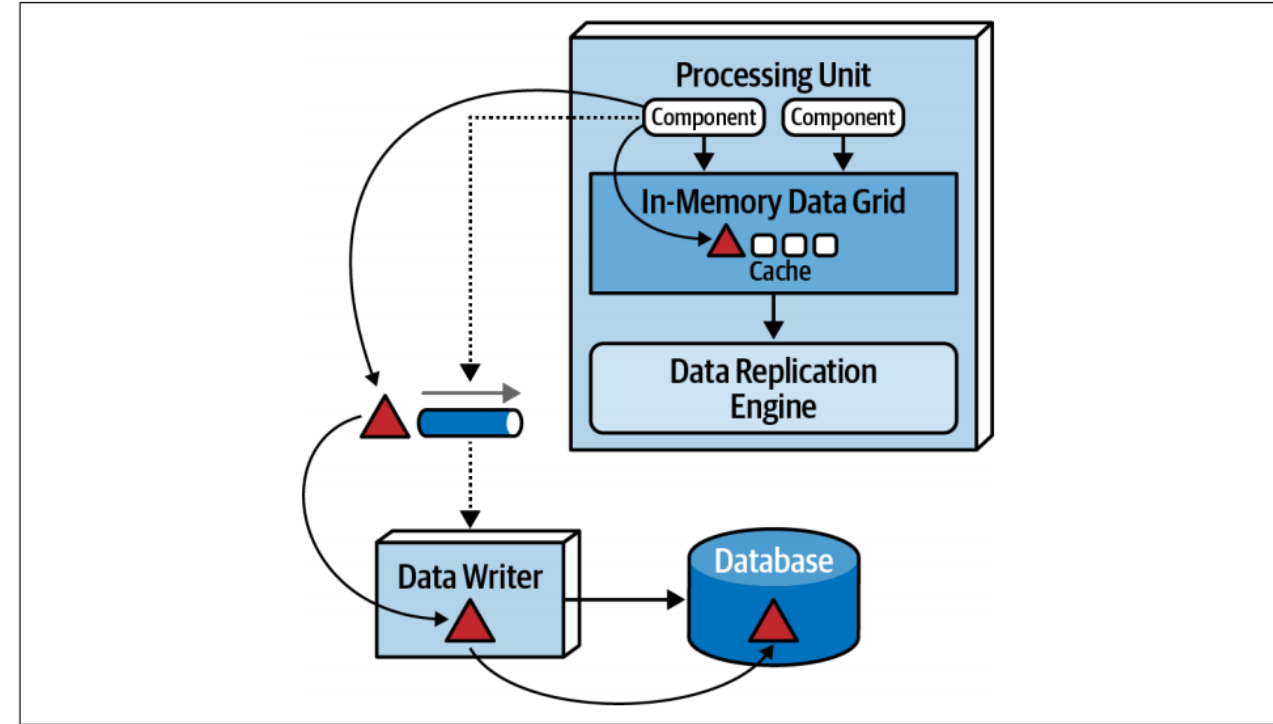
Data Pumps

- Data pumps **manages data** as processing units do not directly read from and write to a database.
- Data pumps are always asynchronous, providing **eventual consistency** with the in memory cache and the database.

6. Space-Based Architecture Style

Data Pumps

- Data pumps are usually implemented using messaging supporting asynchronous communication, guaranteed delivery, and preserving message order via FIFO.
- Messaging decouples processing units and data writer, if the data writer is not available, uninterrupted processing can still take place within the processing units.

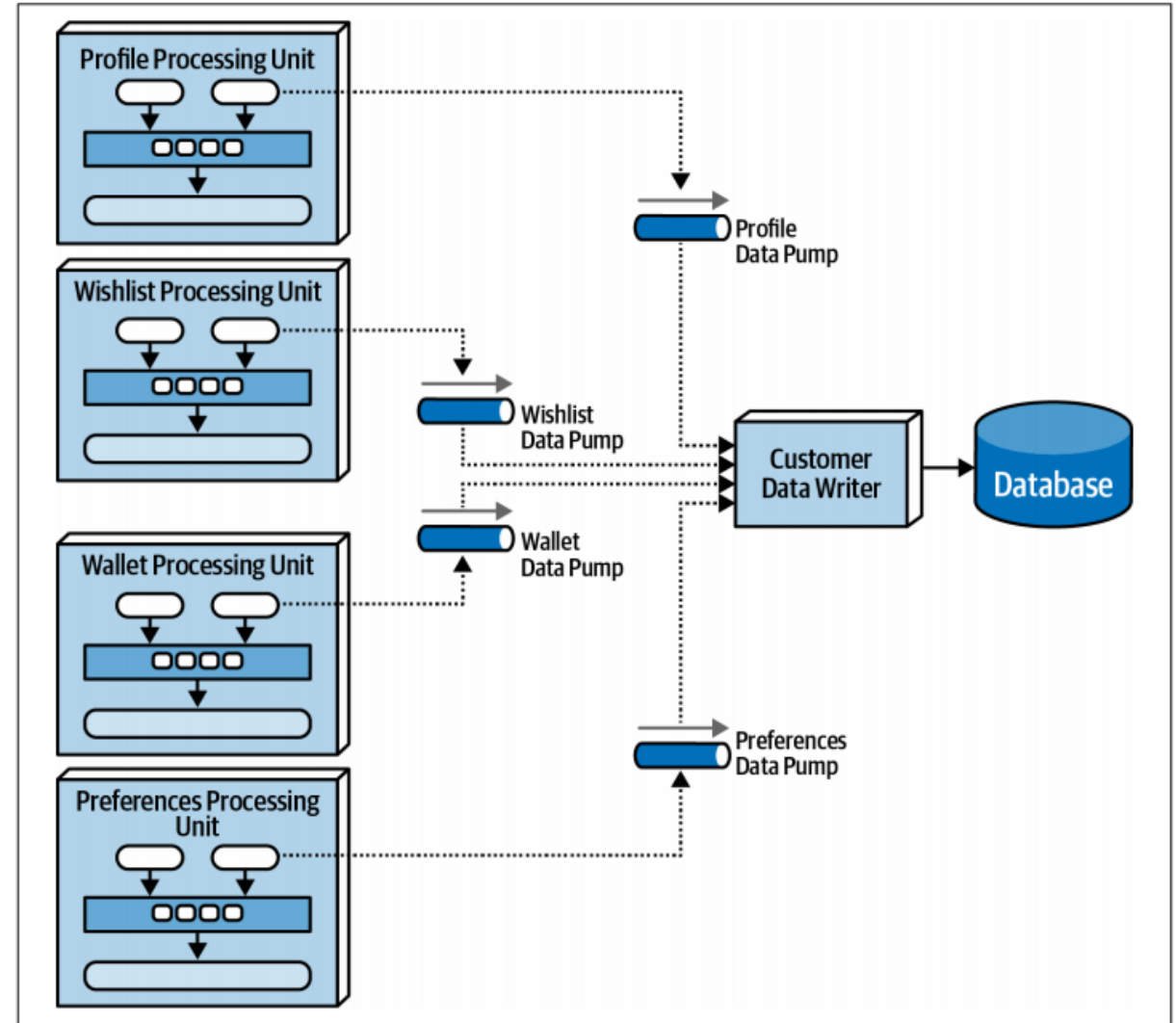


Data pump used to send data to a database

6. Space-Based Architecture Style

Data Writers

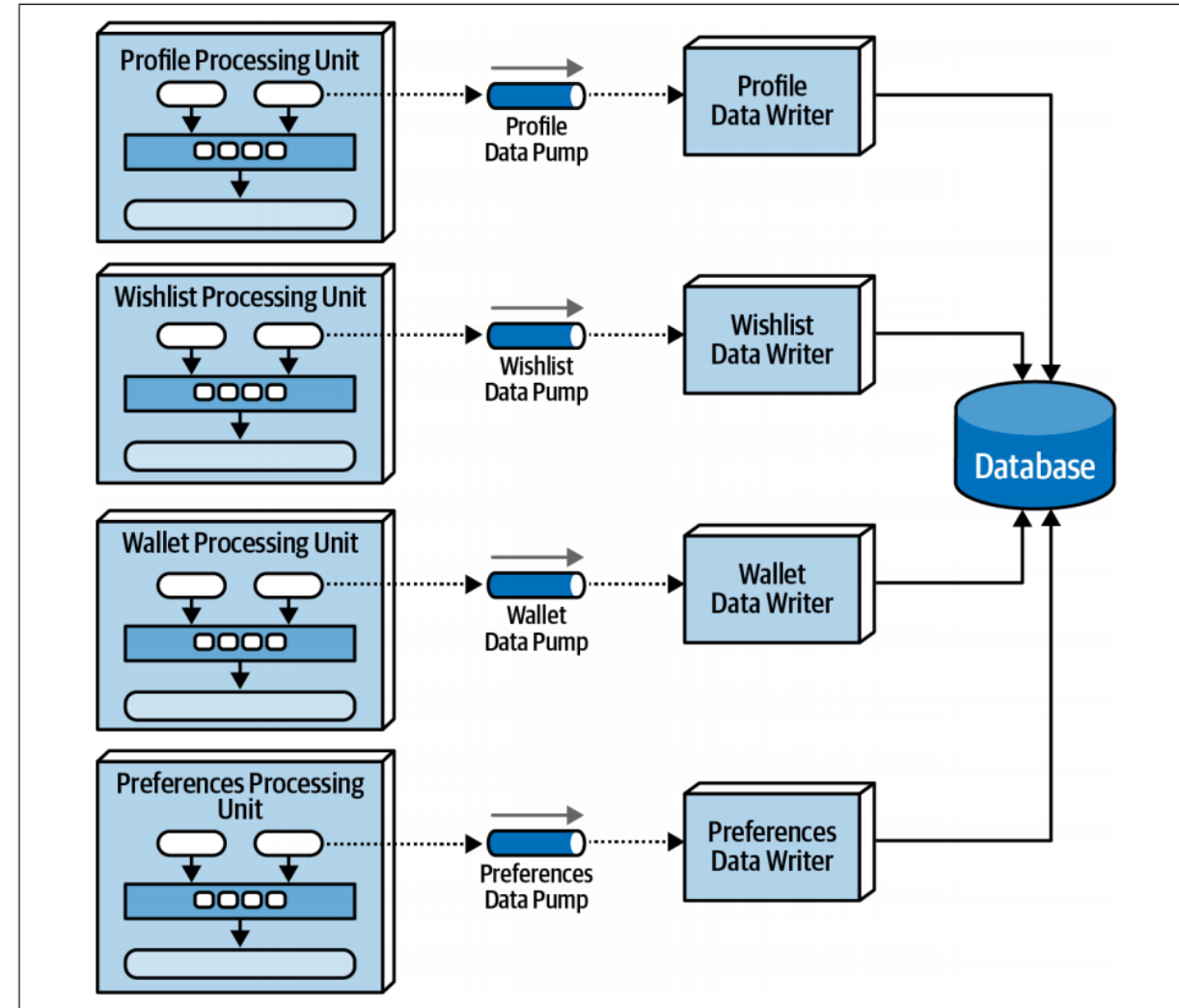
- The data writer component accepts messages from a data pump and **updates the database**.
- Data writers can be implemented as **services, applications, or data hubs** (such as Ab Initio).
- **Notice there are four different processing units and data pumps representing the customer domain but only one data writer.**
- **The single data writer listens to all four data pumps.**



6. Space-Based Architecture Style

Data Writers

- Alternatively, each class of processing unit can have its own dedicated data writer component.
- In this model the data writer is dedicated to each corresponding data pump.
- While this model tends to produce too many data writer components, it does provide better scalability and agility due to the alignment of processing unit, data pump, and data writer.

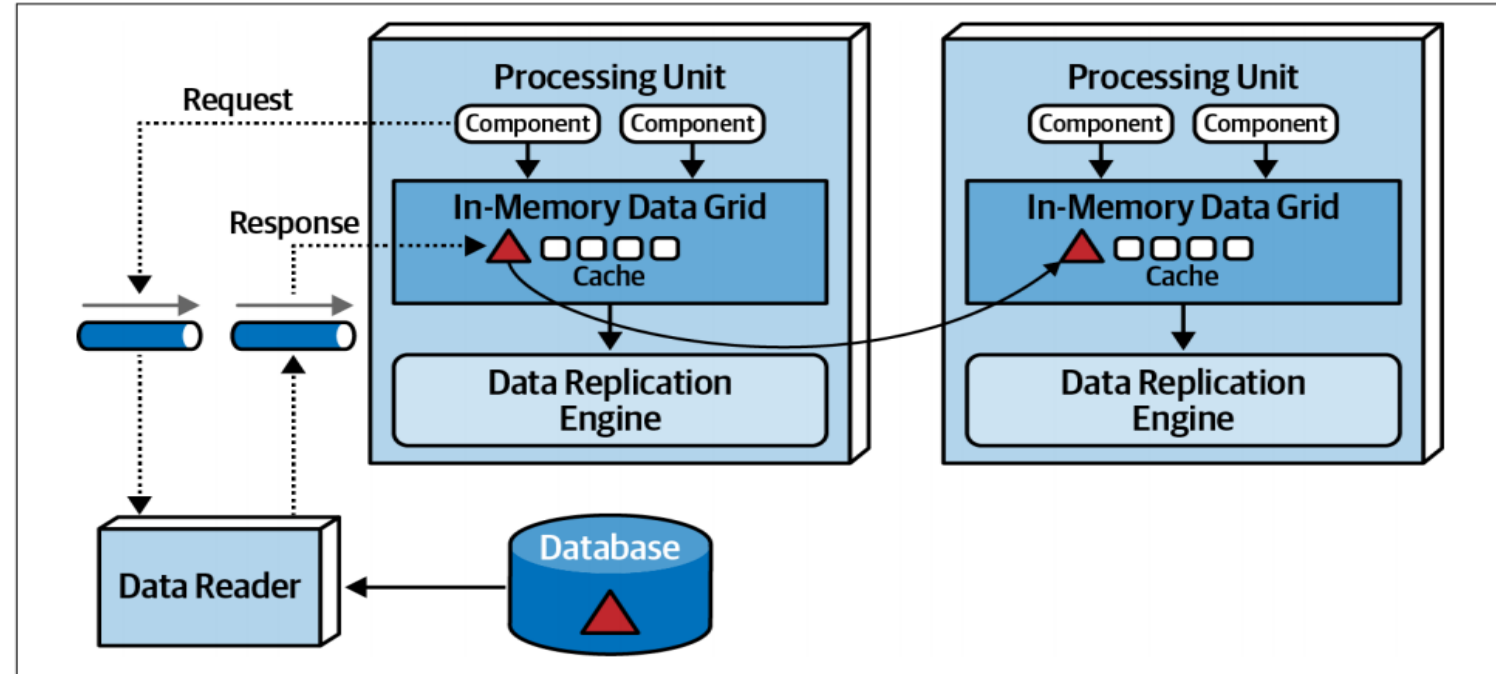


Dedicated data writers for each data pump

6. Space-Based Architecture Style

Data Readers

- Data readers take on the responsibility for reading data from the database and sending it to the processing units via a reverse data pump.



- Data readers are only invoked under one of three situations:
 - A crash of all processing unit instances of the same named cache
 - A redeployment of all processing units within the same named cache, or
 - Retrieving archive data not contained in the replicated cache.

6. Space-Based Architecture Style

Implementation Examples

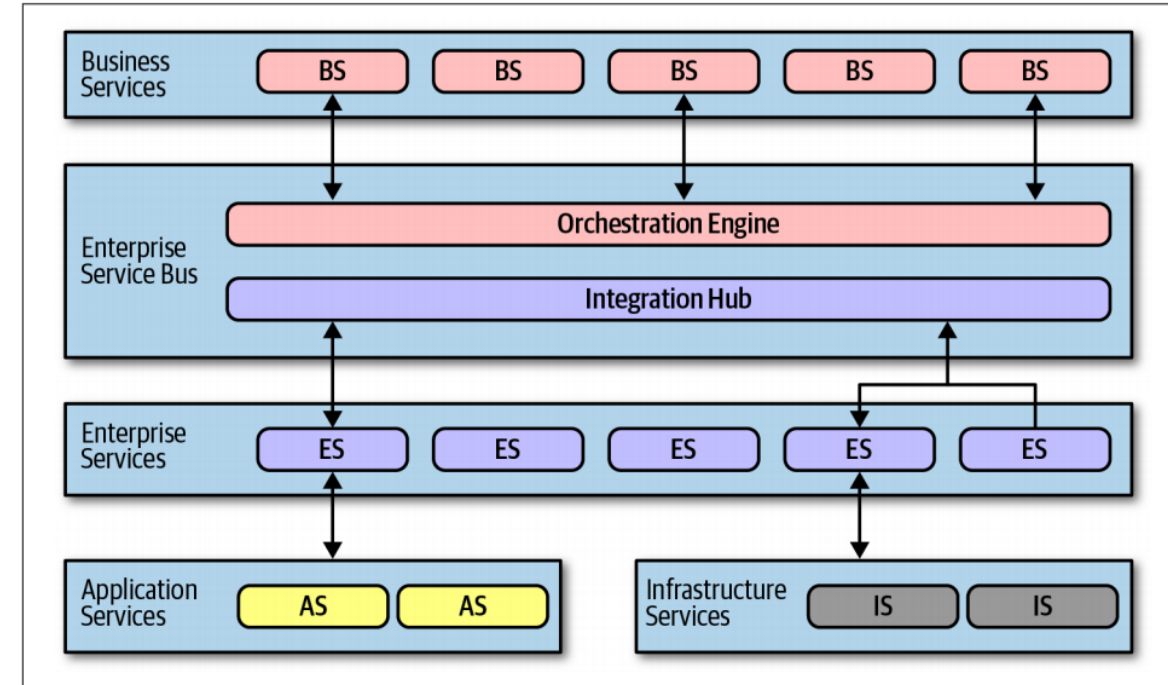
- Space-based architecture is well suited for applications that experience high spikes in user or request volume and applications that have throughput in excess of 10,000 concurrent users.
- Examples of space-based architecture include applications like
 - online concert ticketing systems and
 - online auction systems.
- Both of these examples require high performance, high scalability, and high levels of elasticity.

Architecture Styles

7. Orchestration-Driven Service-Oriented Architecture

- Service-Oriented Architecture (SOA) is an architectural style.
- Applications built using the SOA style deliver **functionality as services** that can be used or reused when building applications
- SOA uses open standards to integrate software assets as services
- It provides a standard form of interactions of services
- SOA services become the building blocks that form business flows
- Services can be reused by other applications
- A service provides a discrete business function that operates on data.

Topology



7. Orchestration-Driven Service-Oriented Architecture

- Characteristics of a Service
 - Supports open standards for integration: Although proprietary integration mechanisms may be offered by the SOA infrastructure, SOA's should be based on open standards.
 - Open standards ensure the broadest integration compatibility opportunities
 - Loose coupling: provides well defined interfaces to clients
 - Stateless: The service does not maintain state between invocations. If a transaction is involved, the transaction is committed and the data is saved to the database.
 - Location agnostic: Users of the service do not need to worry about the implementation details for accessing the service. The SOA infrastructure will provide standardized access mechanisms with service-level agreements.

7. Orchestration-Driven Service-Oriented Architecture

- A service is a reusable component that can be used as a building block to form larger, more complex business-application functionality.
- A service may be as simple as “get me some person data,” or as complex as “process a disbursement.”

Business Services

- Sit at the top of this architecture and provide the **entry point**.
- Example, services like **ExecuteTrade** or **PlaceOrder** represent domain behavior.
- These service definitions contained no code—just input, output, and sometimes schema information.

7. Orchestration-Driven Service-Oriented Architecture

Enterprise Services

- Contain **fine-grained, shared implementations**. Typically, a team of developers is tasked with building atomic behavior around particular business domains.
- These services are the building blocks that make up the coarse-grained business services, tied together via the orchestration engine.
- This separation of responsibility flows from the reuse goal in this architecture.
- If developers can build fine-grained enterprise services at just the correct level of granularity, the business won't have to rewrite that part of the business workflow again.

Application Services

- Application services are single-implementation services.

Infrastructure Services

- Infrastructure services supply the operational concerns, such as monitoring, logging, authentication, and authorization.

7. Orchestration-Driven Service-Oriented Architecture

Orchestration Engine

- integration hub.
- The orchestration engine defines the relationship between the **business** and **enterprise services**, **how they map together**, and **where transaction boundaries lie**.
- It acts as an **integration hub**, allowing architects to integrate custom code with package and legacy software systems.

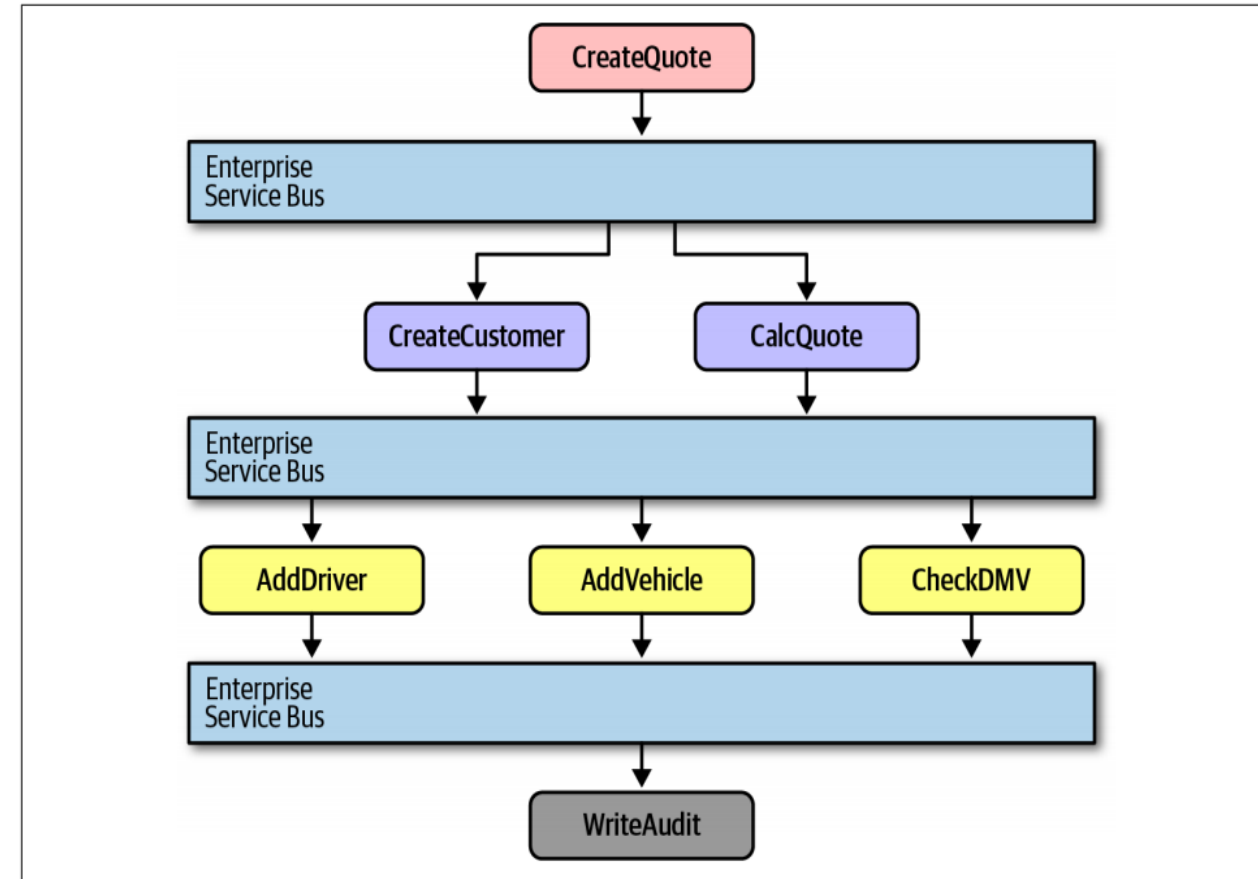
Message Flow

- Handle requests, Calls, and responses.
- All requests go through the orchestration engine—it is the location within this architecture where logic resides. Thus, message flow goes through the engine even for internal calls.

7. Orchestration-Driven Service-Oriented Architecture

Message Flow

- The **CreateQuote** business-level service calls the service bus, which defines the workflow that consists of calls to **CreateCustomer** and **CalculateQuote**, each of which also has calls to application services.
- The service bus acts as the inter-mediary for all calls within this architecture, serving as both an integration hub and orchestration engine.



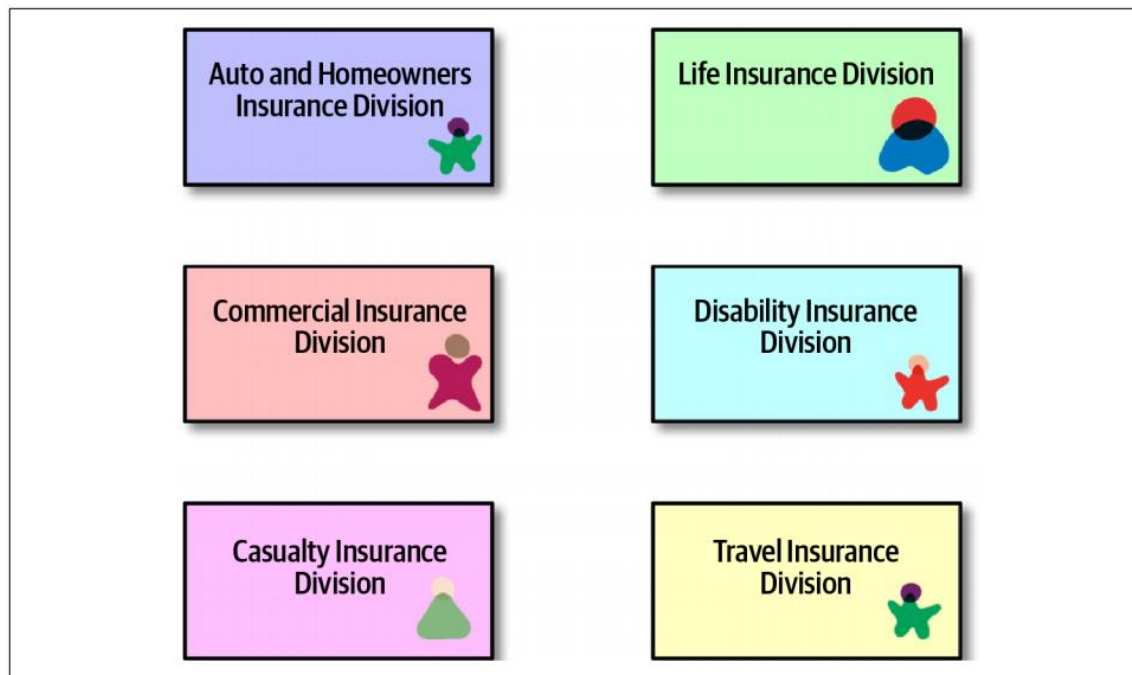
Message flow with service-oriented architecture

Architecture Styles

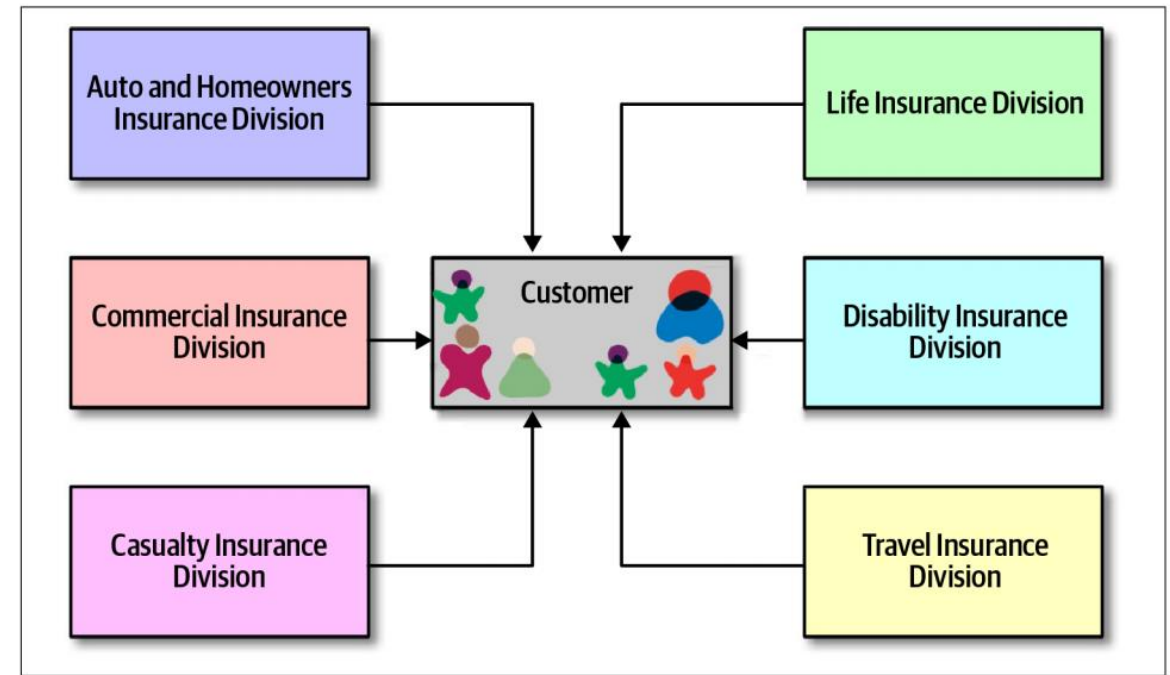
7. Orchestration-Driven Service-Oriented Architecture

Reuse...and Coupling

- A major goal of this architecture is reuse at the service level—the ability to gradually build business behavior that can be incrementally reused over time.



Seeking reuse opportunities in service-oriented architecture



Building canonical representations in service-oriented architecture

7. Orchestration-Driven Service-Oriented Architecture

Reuse...and Coupling with Example

- The proper strategy for service oriented architecture entails extracting the customer parts into a reusable services and allowing the original services to reference the canonical Customer service.
- By isolating all customer behavior into a single Customer service, leads achieving obvious reuse goals.

Challenges in this Example

- The reuse, they also incur a huge amount of coupling between components in such a way that a change to the Customer service ripples out to all the other services.
- The Customer service will have to include details about driver's licenses that the disability insurance division cares nothing about.

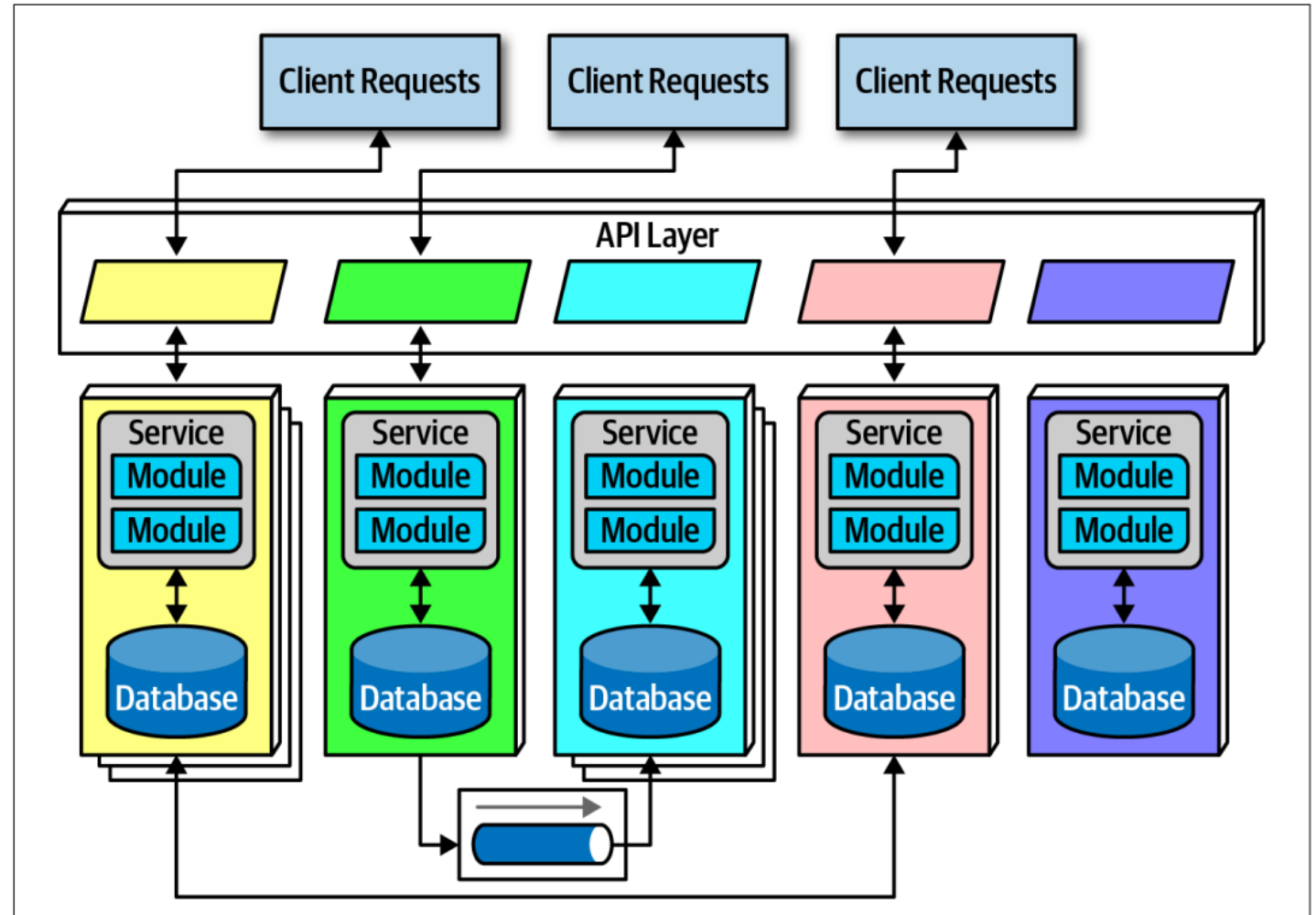
8. Micro Services Architecture

- So, what is microservice?
- Microservice is a kind of software architecture style.
- To understand it, we need to see the traditional Monolithic Architecture first.
- In Monolithic Architecture, usually all things are developed in **one application including UI components, Business Logic, and Data Access Layer**. And usually that application is deployed on **one server** node.
- However, microservice architecture **split Business Logic and Data Access Layer into several small part called Microservices**.
- Real business logic and Data access are implemented by invoking multiple different microservices distributed on different nodes.

8. Micro Services Architecture

- Decompose monolithic business systems into a number of independently deployable services;
 - Each service represents a business capability, or the smallest, “atomic” unit of service that delivers business value.
- Type equation here.Enables speed, safety, and scale via decoupling business domain into independently deployable “bounded contexts” of capabilities; and associated change cycles;
 - each service is deployed independently(a self-service paradigm);
 - offers independent, and efficient scaling of services.
 - incremental change is facilitated allowing incremental deployment of new functionality.
- The primary goal of micro-services is high decoupling, physically modeling the logical notion of bounded context.

8. Micro Services Architecture Topology



The topology of the micro services architecture style

8. Micro Services Architecture

Distributed

- Micro services form a *distributed architecture*: each service runs in its own process, which originally implied a physical computer but quickly evolved to virtual machines and containers.
- Decoupling the services to this degree allows for a simple solution to a common problem in architectures that heavily feature multitenant infrastructure for hosting applications.
- Separating each service into its own process solves all the problems brought on by sharing.
- With cloud resources and container technology, micro services reap the benefits of extreme decoupling, both at the domain and operational level.
- Network calls which take much longer than method calls, and security verification at every endpoint, and service granularity such as transaction boundaries in distributed environment can become bottleneck in micro service architectures.

8. Micro Services Architecture

Bounded Context

- The driving philosophy of micro services is the notion of bounded context: each service models a domain or workflow. Thus, each service includes everything necessary to operate within the application, including classes, other subcomponents, and database schemas.
- Instead of sharing of common classes as in monolithic architecture micro services micro services try to avoid coupling, and thus an architect building this architecture style prefers duplication to coupling.
- Micro services take the concept of a **domain-partitioned architecture** to the extreme. Each service is meant to represent a domain or subdomain; in many ways, micro service is the physical embodiment of the logical concepts in domain-driven design.

Granularity

- The purpose of service boundaries in micro services is to capture a domain or work flow.

8. Micro Services Architecture

Granularity Guidelines to help find the appropriate boundaries:

Purpose

- The most obvious boundary relies on the inspiration for the architecture style, a domain. Ideally, each **micro service should be extremely functionally cohesive**, contributing one **significant behavior** on behalf of the overall application.

Transactions

- **Bounded contexts** are business workflows, and often the entities that need to cooperate in a transaction show architects a **good service boundary**. Because **transactions cause issues in distributed architectures**, if architects can design their system to avoid them, they generate better designs.

Choreography

- If an architect builds a set of services that offer excellent domain isolation yet require extensive communication to function, the architect may consider **bundling these services back into a larger service to avoid the communication overhead**.

8. Micro Services Architecture

Data Isolation

- Many other architecture styles use a single database for persistence. However, micro services tries to avoid all kinds of coupling, including shared schemas and databases used as integration points.

API Layer

- Micro services are firmly domain partitioned, they don't need broker or mediator between the service applications.
- The API layer is often used to host service discovery, allowing a single place for user interfaces or other calling systems to **find and create services** in an elastic, consistent way.

Additional Sources

1. <https://www.ibm.com/topics/microservices>
2. <https://www.ibm.com/cloud/blog/soa-vs-microservices>

Thanks !