

CHAPTER 2

ARCHITECTURES

Distributed Systems Architecture

- Distributed Systems are composed of various hardware and software (collectively called **components**) that communicate with each other only by transfer of messages. These components are placed inside a single network.
- So, when we think about the architecture styles for distributed computing, the main task would be to ensure that *how these components are organized and communicate with each other over a communication network?*

Conti...

- Software Architecture

— The logical organization of distributed systems into software components, also referred to as software Architecture.

- System Architecture

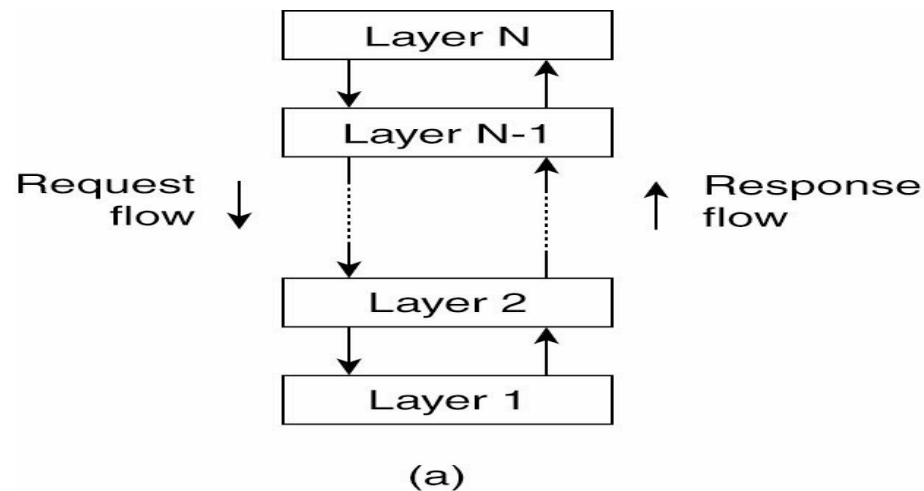
- How to place software components on real machines.

Architectural Styles

- So, Architectural style describe
 - Component/element types
 - Connections
 - Data exchange
- Important styles of architecture for distributed systems are:
 - Layered architectures
 - Object-based architectures
 - Data-centered architectures
 - Event-based architectures

Layered Architectural Style

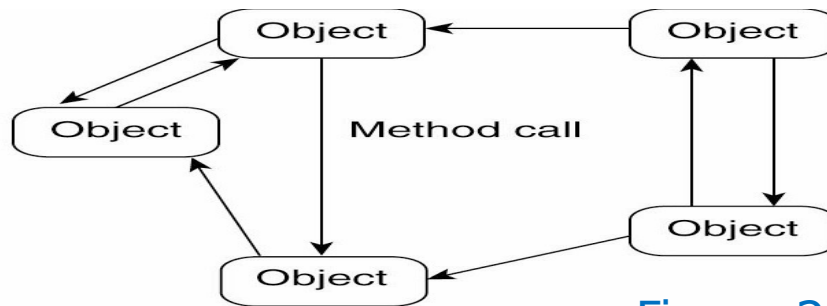
- The basic idea for the layered style is simple: components are organized in a layered fashion, where a component at layer L_i is allowed to call components at the underlying layer L_{i+1} , but not the other way around, as shown in Fig. 2-1(a).
- This model has been widely adopted by the networking community.
- An key observation is that control generally flows from layer to layer: requests go down the hierarchy whereas the results flow upward.



- Figure 2-1. The (a) layered architectural style

Object-based Architecture

- A far **looser organization** is followed in object-based architectures, which are illustrated in Fig. 2-1(b).
- In essence, each **object corresponds to what we have** defined as a component, and these components are connected through a remote procedure call (RPC) mechanism.
- This software architecture **matches the client-server system architecture.**



(b)

Figure 2-1. (b) The object-based architecture

The Layered and Object Based Architectures still form the most important styles for large software systems.

Event-based Architecture

- In event-based architectures, **processes essentially communicate through the propagation of events**, which optionally also carry data, as shown in Fig. 2-2(a).
- The basic idea is that processes **publish** events, after which the middleware ensures that only those processes that subscribed to those events will receive them.
- The main advantage of event-based systems is that **processes are loosely coupled**. In principle, they need not explicitly refer to each other. This is also referred to as being decoupled in space, or referentially decoupled.

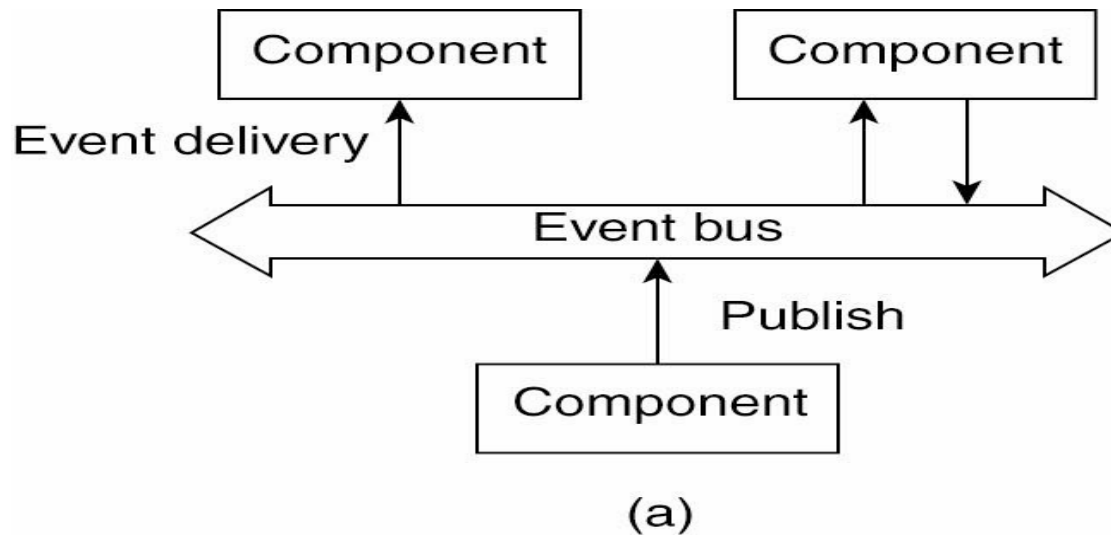
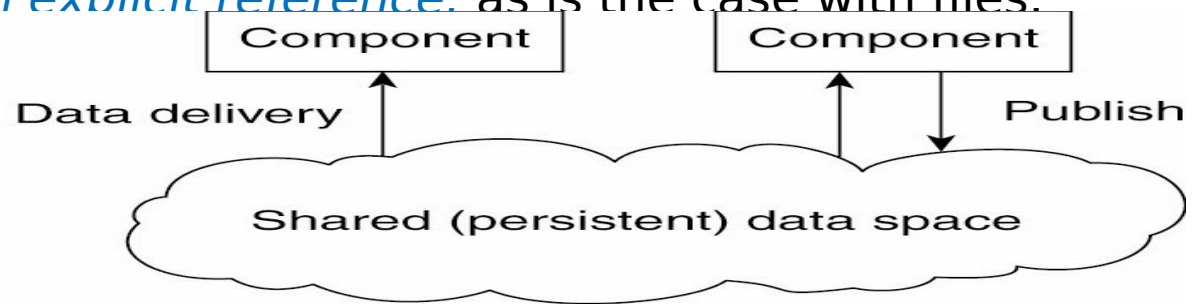


Figure 2-2. (a) The event-based architectural style

Data-centered Architectures

- Data-centered architectures evolve around the idea that **processes communicate through a common (passive or active) repository**.
- Event-based architectures can be combined with data-centered architectures, that is also known as **shared data spaces**.
- The essence of shared data spaces is that **processes are now also decoupled in time**: they need not both be active when communication takes place.
- Furthermore, many shared data spaces use a SQL-like interface for shared repository, where ***data can be accessed using a description, rather than an explicit reference***, as is the case with files.



(b)

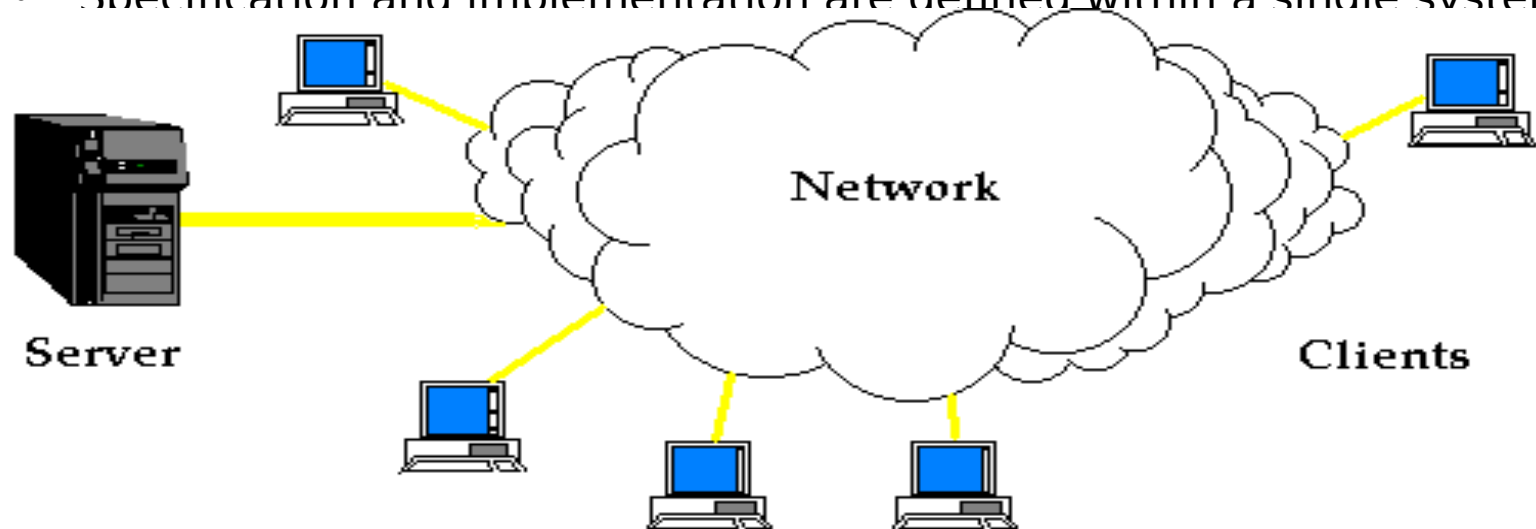
Figure 2-2. (b) The shared data-space architectural style

System Architectures

- How many distributed systems are actually organized, is decided by, [where software components are placed](#).
- Deciding on software components, their interaction, and their placement leads type of a software architecture, also called a [system architecture](#), that may be classified as either:
 - Centralized,
 - Decentralized or Distributed, or
 - Hybrid

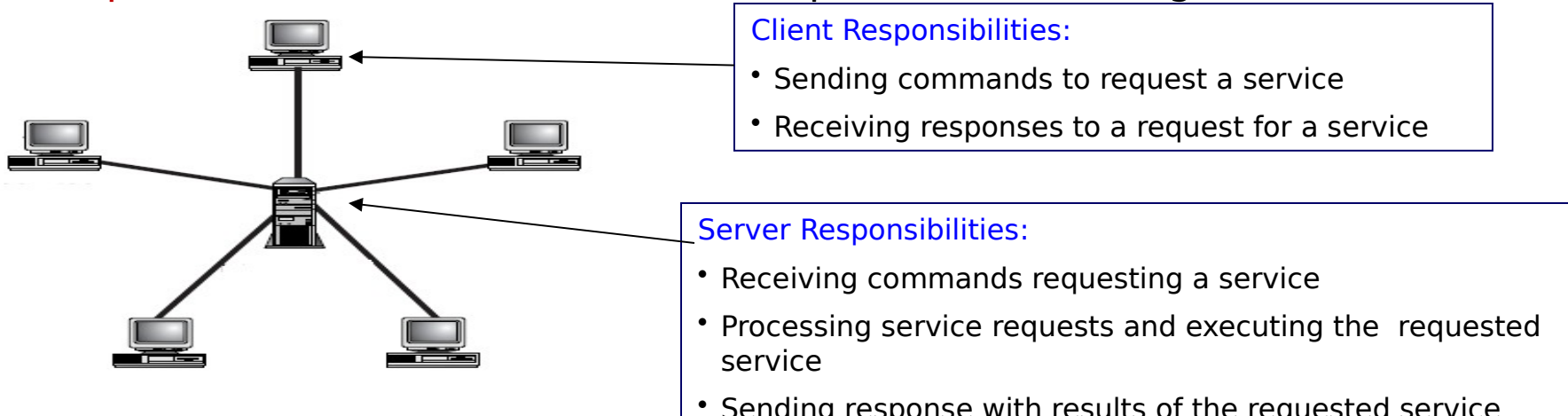
Centralized System Architecture

- **In Centralized systems,** Data, Process and Interface components of an information system are central.
- Users interact with the system via terminals or terminal emulators.
- A single computer with one or more CPUs processes all incoming request.
- Example: [Client-Server Architecture](#).
- Disadvantage :
 - Problems with reliability
 - Specification and implementation are defined within a single system



Client / Server Architecture

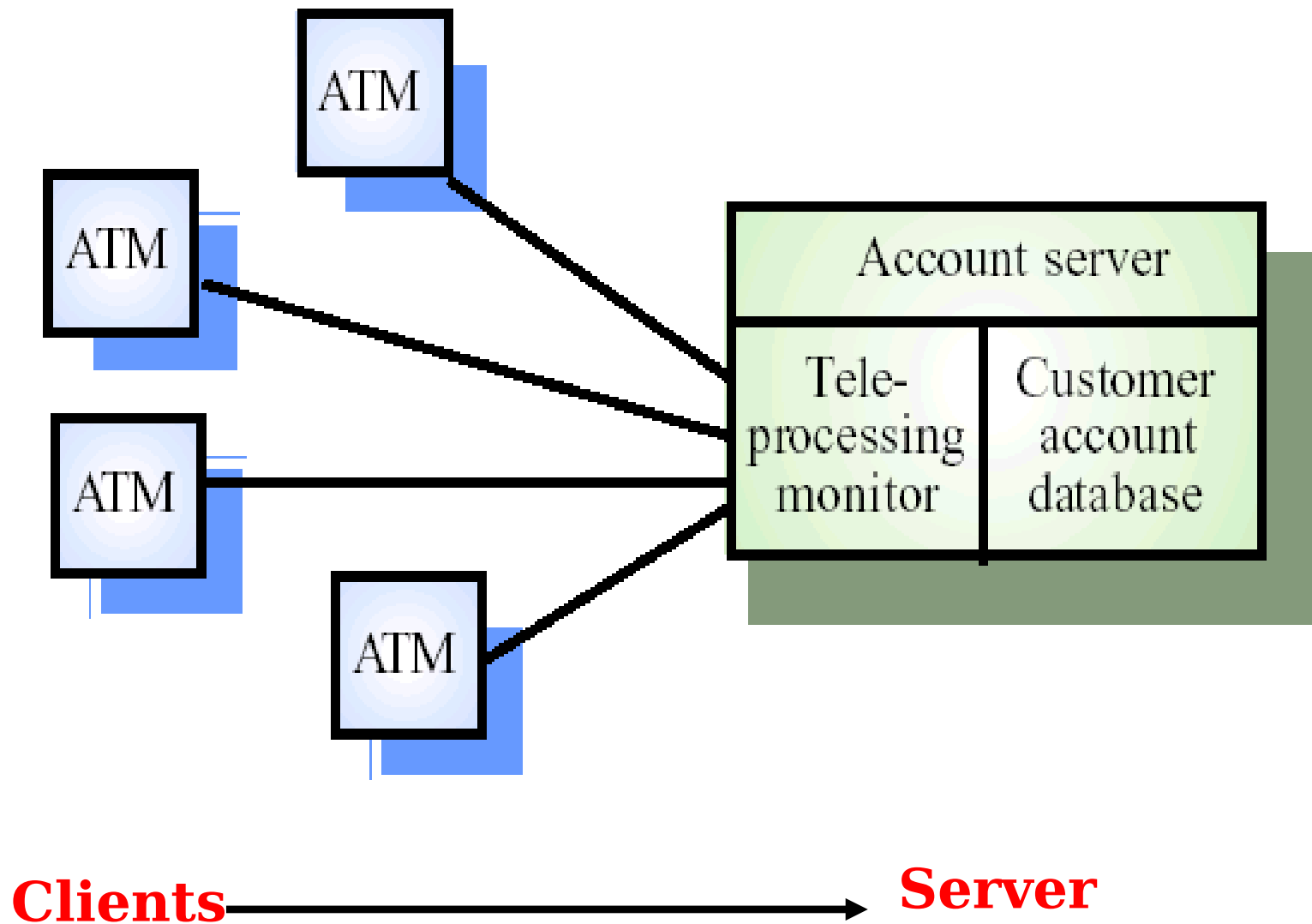
- Client/server is a distributed computing model, in which *client applications request services from server processes*. Clients and servers typically run on different computers interconnected by a computer network.
- **Client** application is a process or program that sends messages to a server via the network. Those messages request the server to perform a specific task, such as looking up a customer record in a database or returning a portion of a file on the server's hard disk.
- **Server** process or program listens for client requests that are transmitted via the network. Servers receive those requests and perform actions such as database queries and reading files.



Client / Server System

- There are three components/Layers of Client/Server system :
 1. The user-interface Layer,
 2. The processing Layer, and
 3. The data Layer.
- On the basis of place of implementation of these layers, Client-Server architecture is divided into two types:
 1. Two-Tier Architecture
 2. Three-Tier Architecture.
- In Two-Tier Architecture, Layer 1 is implemented on Client, while Layer2 and Layer 3 is implemented on Server.
- While, in Three-Tier Architecture, All the Layers are implemented separately at different places.

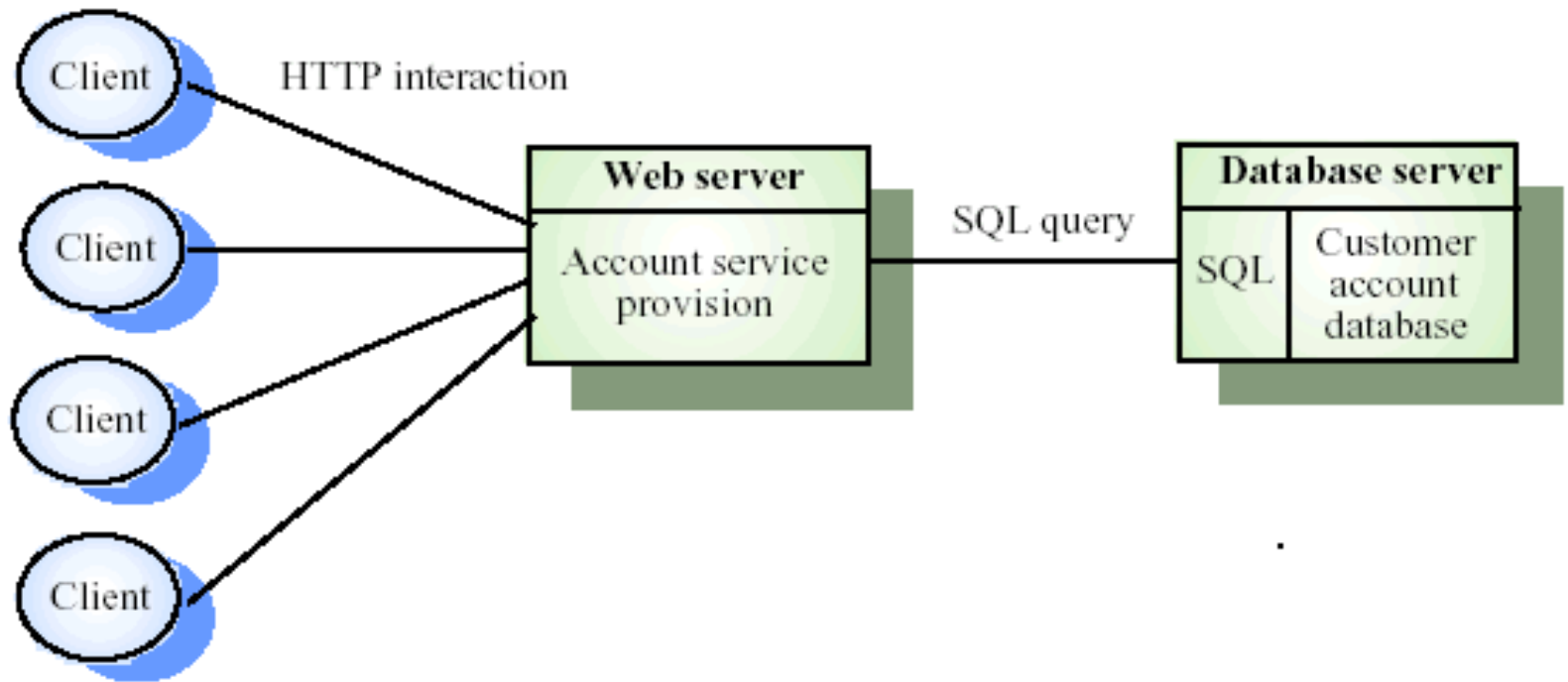
Two-Tier Client / Server Architecture



Three-tier Client Server Architectures

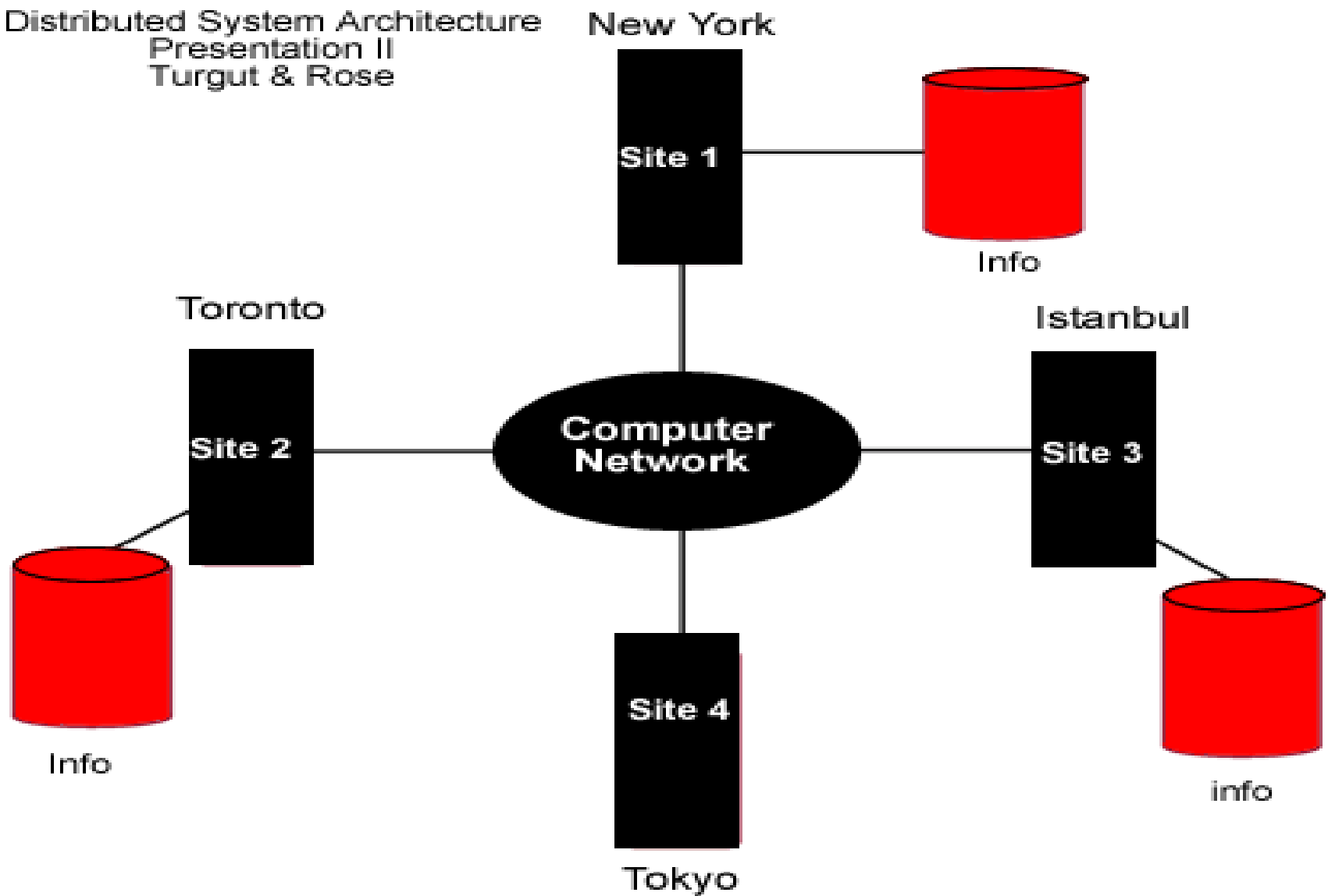
- An important design consideration for large client/server systems is *whether a client talks directly to the server, or whether an intermediary process is introduced in-between the client and the server*. The former is a two-tier architecture, the latter is a three-tier architecture.
- In the *three-tier architecture*, Intermediate process (Middle Layer) :
 - *separate* the clients and servers.
 - *cache frequently accessed server data* to ensure better performance and scalability.
 - Performance can be increased by having the intermediary process *to distribute client requests to several servers* such that requests execute in parallel.
 - The intermediary can also *act as a translation service by converting requests and replies to and from a mainframe format*, or as a security service that grants server-access only to trusted clients.

Three-Tier Internet Banking System



Decentralized or Distributed System Architecture

- **In Distributed System architecture,** Data, Process, and Interface components of an information system are distributed to multiple locations in a computer network. Accordingly, the processing workload is distributed across the network.
- In this architecture, there are Set of separate computers that are capable of autonomous operation, link by a computer network.
 - Enable individual computers (different location) to share resources in the network.
 - Server implementation for the same interface located in different servers.
- Example: Peer to Peer architecture



• Peer-to-peer (P2P) architecture:

≡ Peer-to-peer (P2P) is a **communication model** in which each party has the same status/right and either party can initiate a communication session

- Peer-to-peer computing uses diverse connectivity between participants in a network, typically used for connecting nodes via largely ad-hoc connections

≡ P2P networks are useful for many purposes:

- Sharing content files, containing audio, video, data or anything in digital format
- Sharing real-time data, such as telephone traffic, road traffic, weather condition, etc.

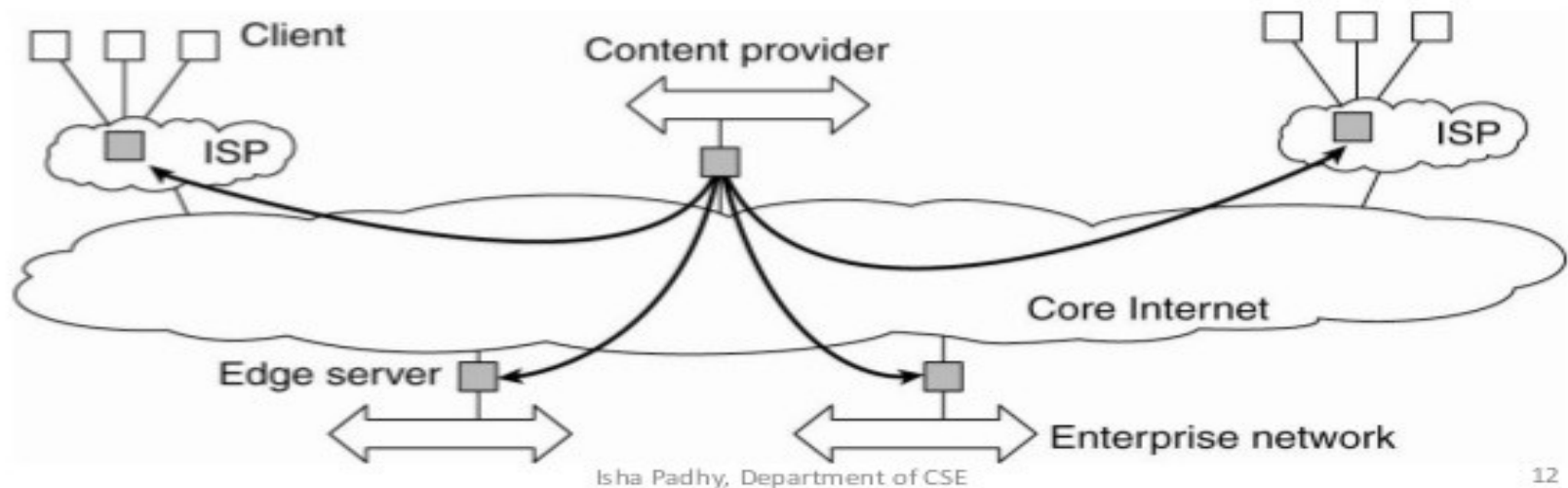
Advantages of DSA

- Companies are preferring their system **decentralized and distributed**, because Distributed System allows companies to have better customer services.
 - **Shareability:** Allows systems to use each other's resources
 - **Expandability:** Permits new systems to be added as members of the overall system
 - **Local Autonomy:** Manage local resources
 - **Improved performance:** Resource replication. Combined processing power of multiple computers provides much more processing power than a centralised system with multiple CPUs
 - **Improved reliability and availability:** Disruption would not stop the whole system from providing its services as resources spread across multiple computers
 - **Potential cost reductions**

Hybrid Architecture

- So far, we have focused on client-server architectures and a number of peer-to-peer architectures. Many distributed systems combine architectural features,
- In this section we take a look at some specific classes of distributed systems in which client-server solutions are combined with decentralized architectures.
- **Edge-Server Systems**
- These systems are deployed on the Internet where servers are placed "at the edge" of the network. This edge is formed by the boundary between enterprise networks and the actual Internet, for example, as provided by an Internet Service Provider (ISP). Likewise, where end users at home connect to the Internet through their ISP, the ISP can be considered as residing at the edge of the Internet. This leads to a general organization as shown in Fig.

Cont...



12

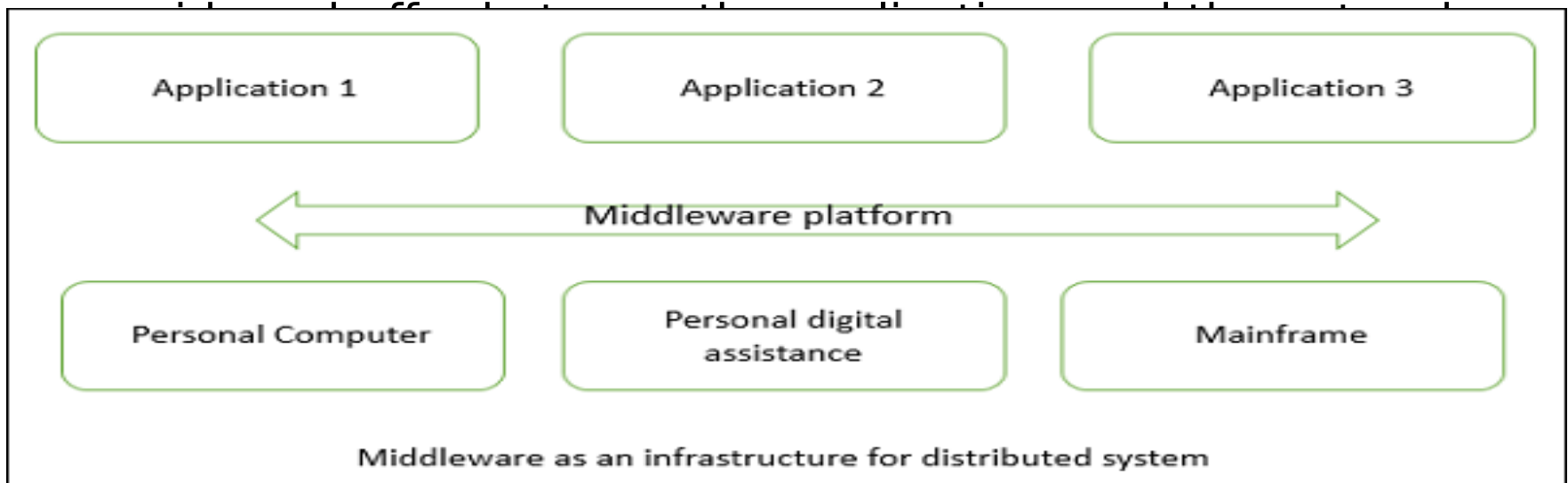
- End users, or clients in general, connect to the Internet by means of an edge server. The edge server's main purpose is to serve content, possibly after applying filtering and transcoding functions. More interesting is the fact that a collection of edge servers can be used to optimize content and application distribution.
- The basic model is that for a specific organization, one edge server acts as an origin server from which all content originates. That server can use other edge servers for

INTRODUCTION TO MIDDLEWARE

- Suitable concepts and mechanisms can be used to develop and execute applications in distributed systems. It is obvious that at which level they should be embedded? Either at Hardware level or programming level.
- Software solutions typically provide the greatest flexibility, because of their suitability for integrating existing technologies (such as operating systems and programming languages). *These conditions lead to the concept of *middleware*.*
- middleware systems actually follow a specific architectural style. For example, many middleware solutions have *adopted an object-based architectural style, such as CORBA*. Others, *like TIB (TIBCO, 2005) provide middleware that follows the event-based architectural style*.
- Middleware offers general services that support distributed execution of applications.

Conti...

- *The term middleware suggests that it is software, positioned between the operating system and the application. Viewed abstractly, middleware can be imagined as a “tablecloth” that spreads itself over a heterogeneous network, covering the complexity of the underlying technology from the application being run on it.*
- Middleware is an infrastructure that appropriately supports the development and execution of distributed applications. It



MIDDLEWARE TASKS

- **Object Model Support:** Middleware should offer mechanisms to support the concepts incorporated in the object model.
- **Operational Interaction:** Middleware should allow the operational interaction between two objects. The model used is the method invocation of an object oriented programming language.
- **Remote Interaction:** Middleware should allow the interaction between two objects located in different address spaces.
- **Distribution Transparency:** From the standpoint of the program, interaction between objects is identical for both local and remote interactions.
- **Technological Independence:** The middleware supports the integration of different technologies.
- **HETEROGENEITY OF COMPONENTS**
 - Integration enables users to access services and run applications over a heterogeneous collection of computers and networks.
 - **Heterogeneity** (that is, variety and difference) applies to all of the following: Hardware devices: computers, tablets, mobile phones, embedded devices, etc.