Architectural Models

From **Coulouris, Dollimore, Kindberg and Blair**

Distributed Systems:

Concepts and Design

Presenter: Dr Hamed Vahdat-Nejad

Physical model

- **Determining the types of computers and devices (e.g. smartphone) that constitute a system and their interconnecting networks.**
- **The most explicit way to describe a system**

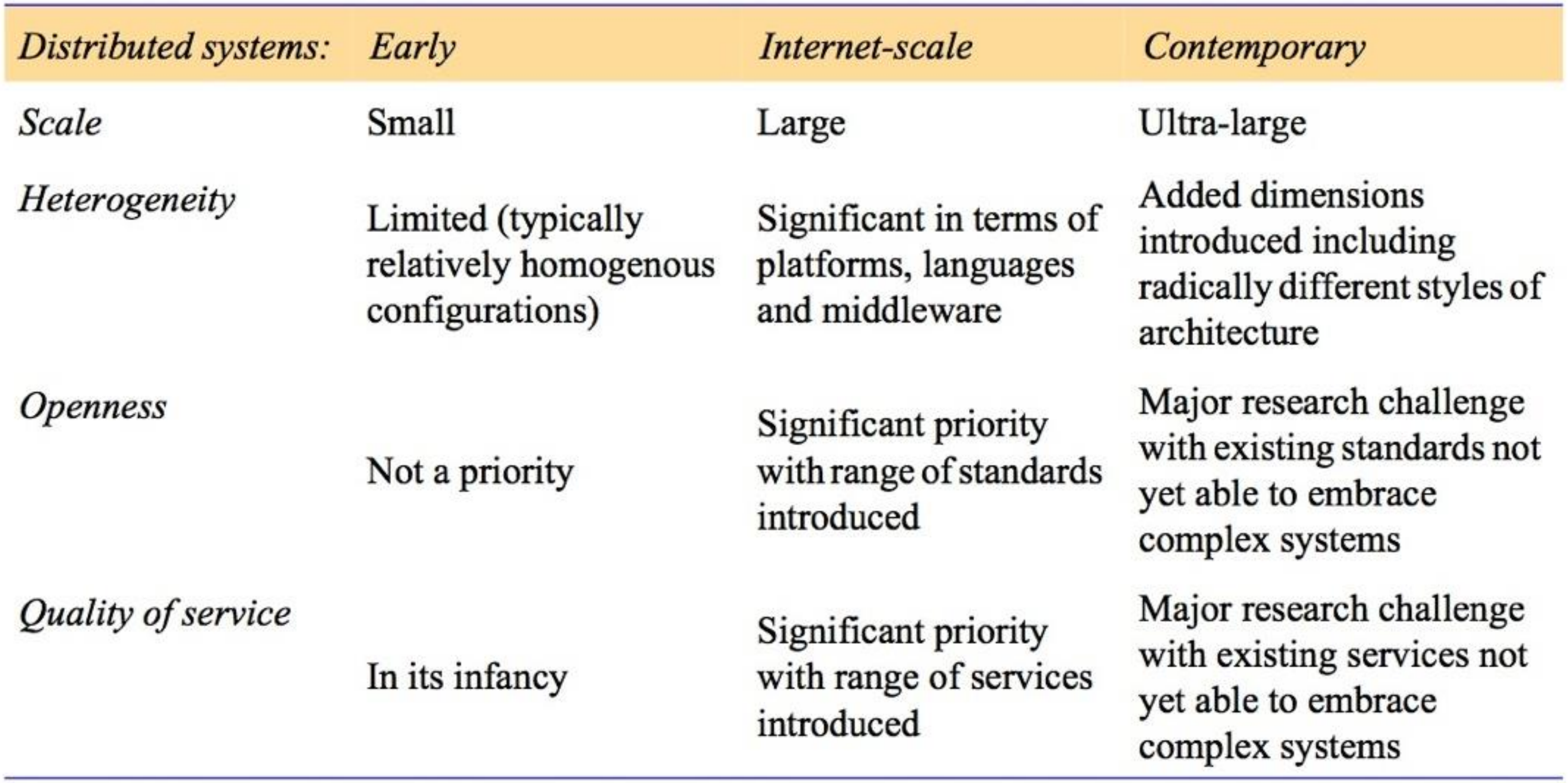
Architectural model

- **Describing a system in terms of computational and communication tasks performed by its computational elements.**

Fundamental model

- **Taking an abstract perspective to determine individual aspects of the system including**
  - interaction model, Considers structure and sequencing of the communication between elements of the system
  - failure model, Considers the ways in which a system may fail to operate correctly.
  - Security model, Considers how the system is protected against attacks.

Physical model
Generations of distributed systems



Distributed systems:	*Early*	*Internet-scale*	*Contemporary*
Scale	Small	Large	Ultra-large
Heterogeneity	Limited (typically relatively homogenous configurations)	Significant in terms of platforms, languages and middleware	Added dimensions introduced including radically different styles of architecture
Openness	Not a priority	Significant priority with range of standards introduced	Major research challenge with existing standards not yet able to embrace complex systems
Quality of service	In its infancy	Significant priority with range of services introduced	Major research challenge with existing services not yet able to embrace complex systems

Contemporary systems: System of systems

An environmental management system for flood prediction includes

- Sensor networks for monitoring
- Systems responsible for flood prediction by running simulations on clusters
- Early warning systems to stakeholders via smartphones.

Architectural models

Architecture: Structure in terms of separately specified components and their interrelationships

Architectural elements:

- What are the entities that are communicating?
- How do they communicate or What communication paradigm is used?
- What roles and responsibilities do they have?
- How are they mapped on to the physical distributed infrastructure?

Communicating entities

From a programming perspective:

- Objects,
- Components (In contrast to objects, they also require other components/interfaces that must be present to fulfill their function)
- Web services:
- A software application identified by a URI, whose interface is defined, described and discovered as XML items.
- Objects and components are often used within an organization to develop tightly coupled applications, while web services often cross organizational boundaries .

Communication paradigm

Inter-process communication: Low level communication between processes

- Message passing primitives
- API offered by Internet Protocols e.g. Socket programming
- Multicast communication

Remote invocation: The most common paradigm in distributed systems in which a remote procedure, function or method is called.

Indirect communication: The communication is performed through a third party, which decouples sender from receiver.

- Publish-subscribe: A large number of publishers distribute information to a large number of subscribers. Publish-subscribe systems provide an intermediary service that efficiently routes the information from producers to consumers.
- Distributed shared memory: Provides transparency of sharing data between processes that do not share physical memory. Programmers are presented a familiar abstraction for reading or writing data as if they were in their local memory.

Roles and responsibilities

Processes (objects, components or services)take on given roles to perform a useful activity.

There are two main architectural styles steming from the role of processes:

- **Client-server**: Processes take on the roles of being client or server. A server may be client of another server e.g. a web server is client of the DNS server as well as the local file server that manages the files of webpages. A search engine runs crawlers that act as clients of other web servers.
- **Peer to peer**: All processes play similar roles by running the same program and offering the same interfaces to each other. The scalability of the service is much higher than client-service architecture, e.g. BitTorrent file sharing system. It uses millions of computers resources.

Figure 2.3
Processes take on the role of being Clients or servers

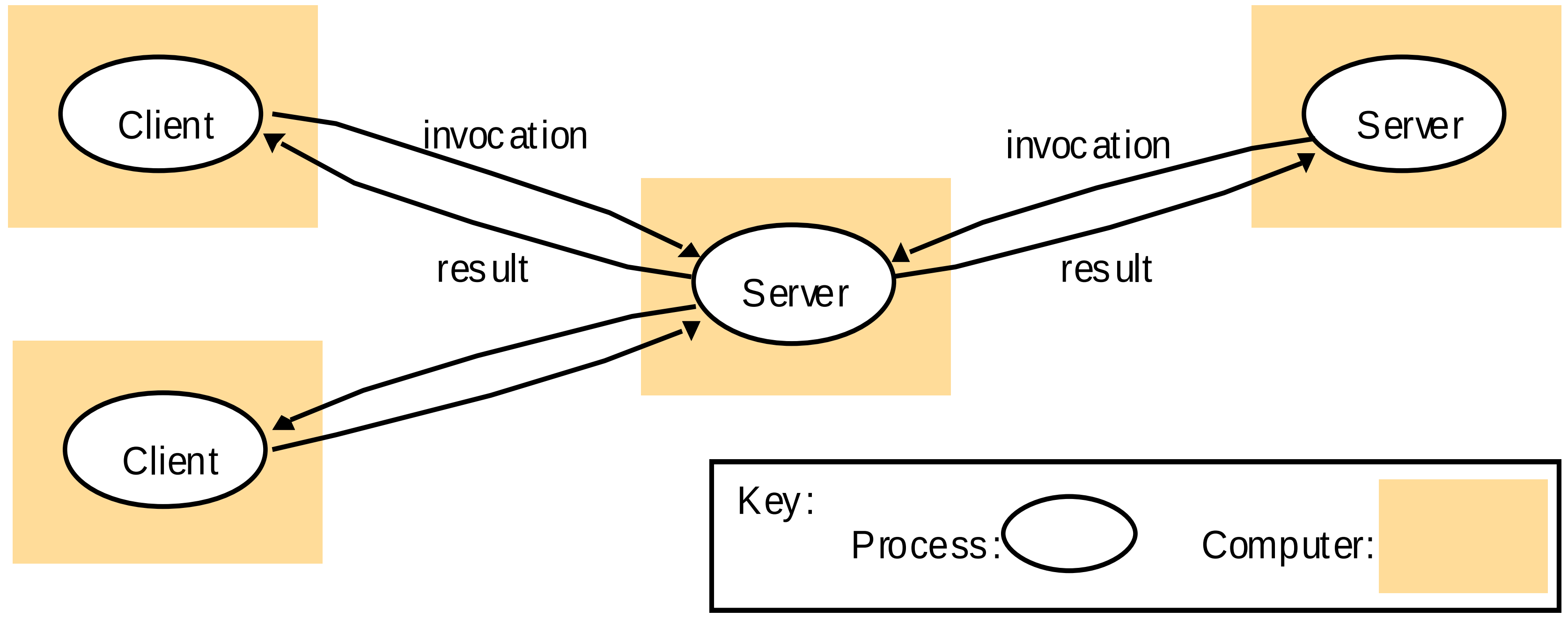
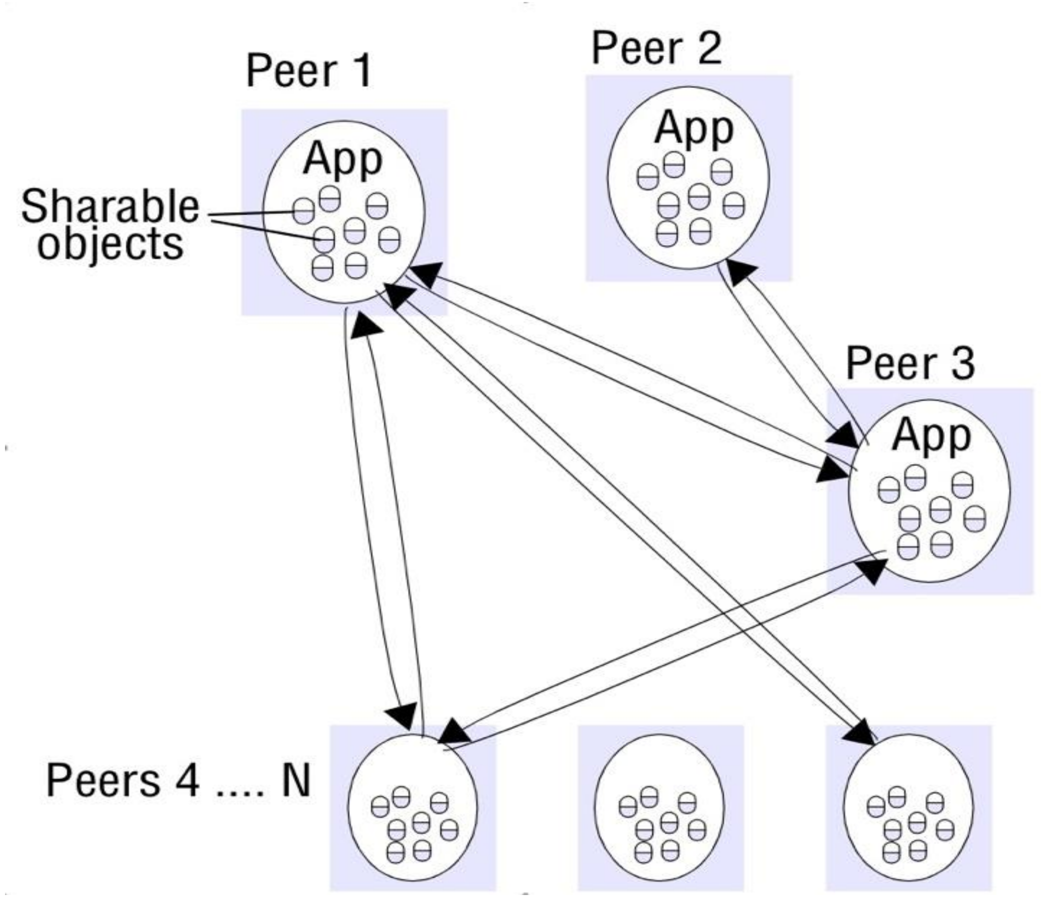


Figure 2.4a
Peer-to-peer architecture: Applications are composed of a large numbers of peer processes running on separate computers. A large number of data objects are shared. The need to place objects and retrieve them and to maintain replicas amongst many computers makes this architecture much more complex than client-server



Placement: How entities such as objects or services map on the underlying physical infrastructure?

Placement strategies:

- **Mapping of services to multiple servers**: services may be implemented as several server processes in separate host computers.

1. The servers may partition the set of objects on which the service is based and distribute those objects between themselves.
2. The servers may maintain replicated copies of the objects on several hosts

- **Caching**: is the storage of recently used data object that is closer to the client than the objects themselves. Cache may be co-located with each client or be located in a proxy server that is shared by clients. Proxy server aims to increase the availability and performance of the service by reducing the load on network and web servers.

Figure 2.4b
A service provided by multiple servers

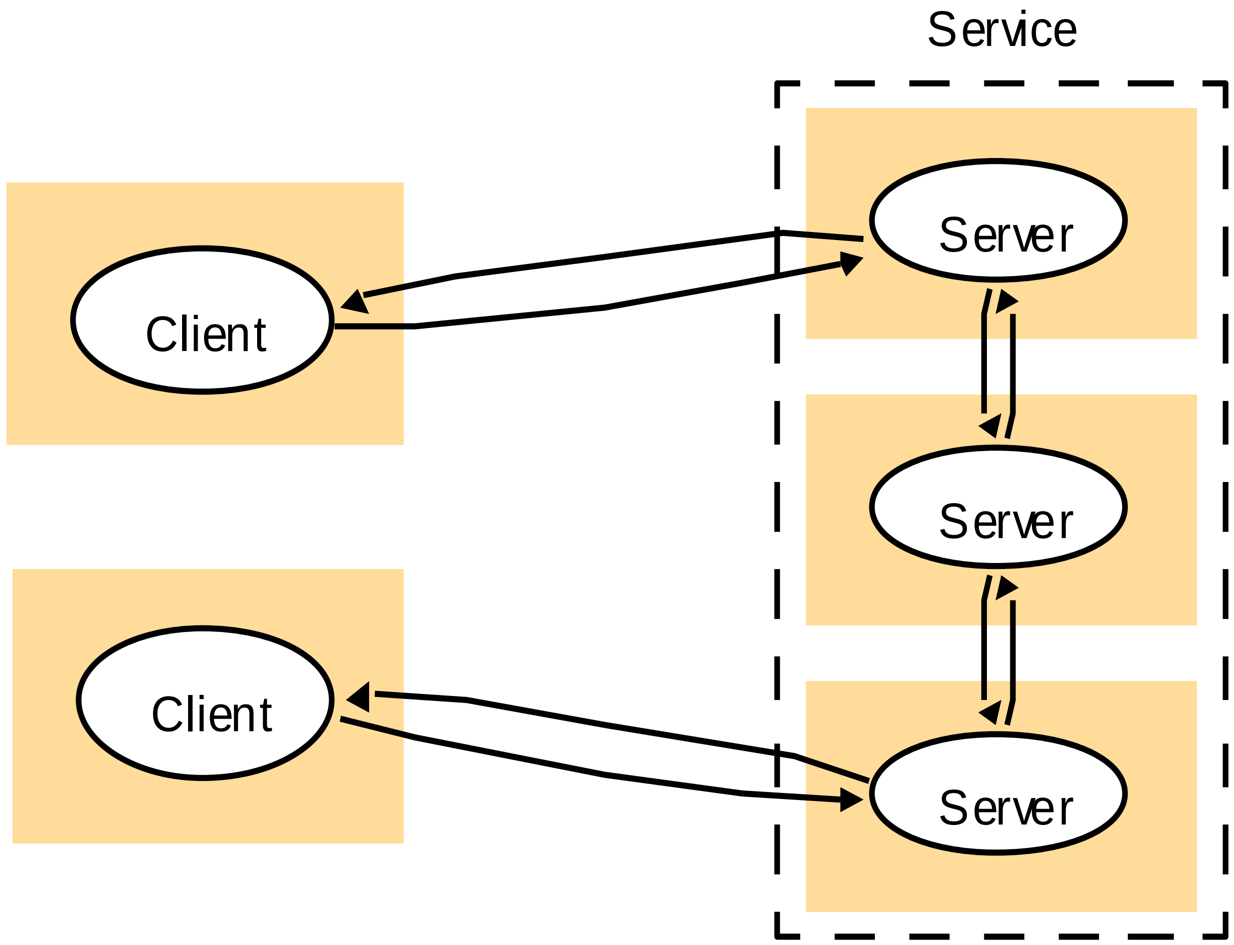
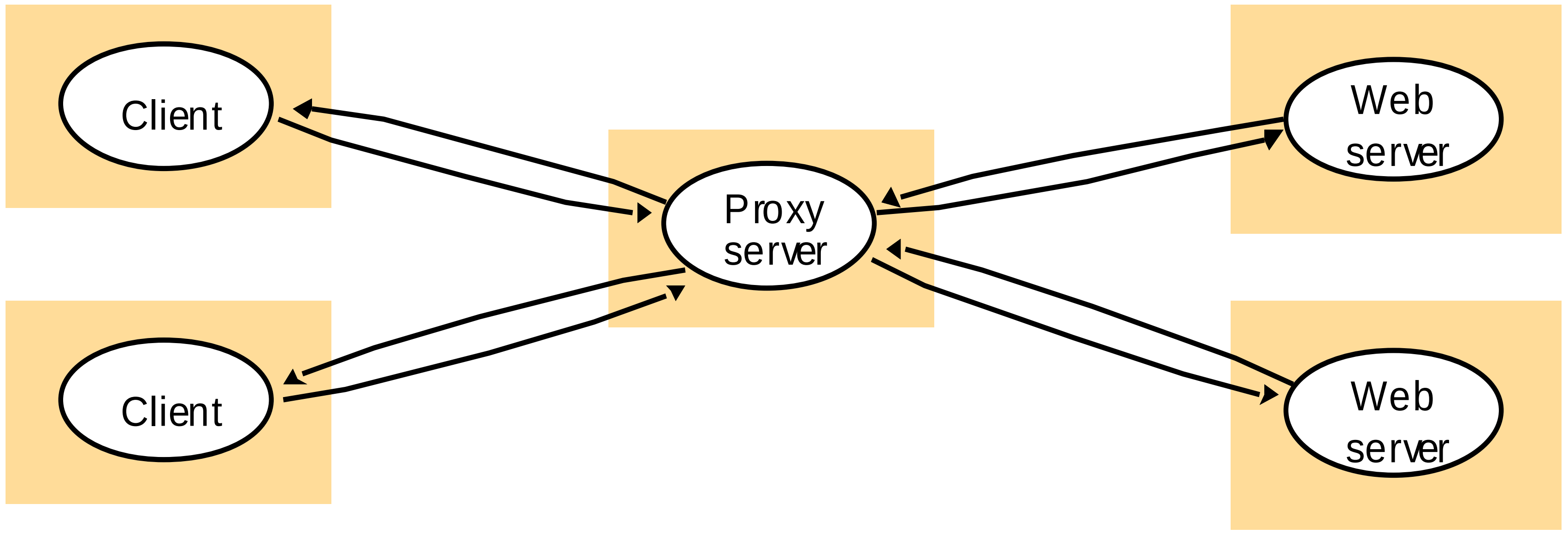


Figure 2.5
Web proxy server



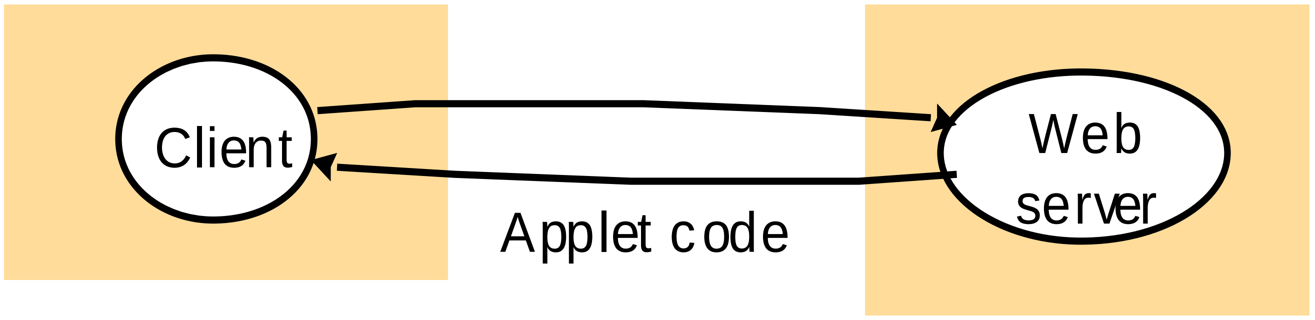
Placement strategies (Continue)

- **Mobile code:** The code is downloaded to the client's browser and runs there, e.g. applets
- It gives good interactive response
- It is a potential security threat to the local resources of client; therefore, browsers give applets limited access to local resources.

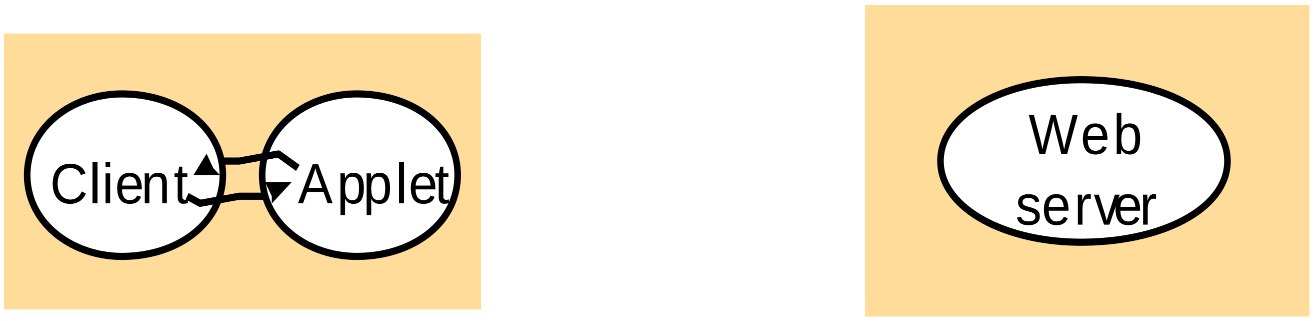
- **Mobile agents**: A running program (code and data) that travels from one computer to another, carrying a task on someone's behalf (e.g. collecting information), and eventually returning with results.
- It may make many invocations to local resources e.g. accessing database entries.
- They are used to install and maintain software on the computers within an organization.
- They are used to compare the prices of products from a number of vendors by visiting their sites.
- They are a potential security threat to the local resources. The environment receiveing an agent should decide which of the local resources it should be allowed to use.
- They may not complete their task if they are refused access to the information they need.

Figure 2.6
Web applets

a) client request results in the downloading of applet code



b) client interacts with the applet

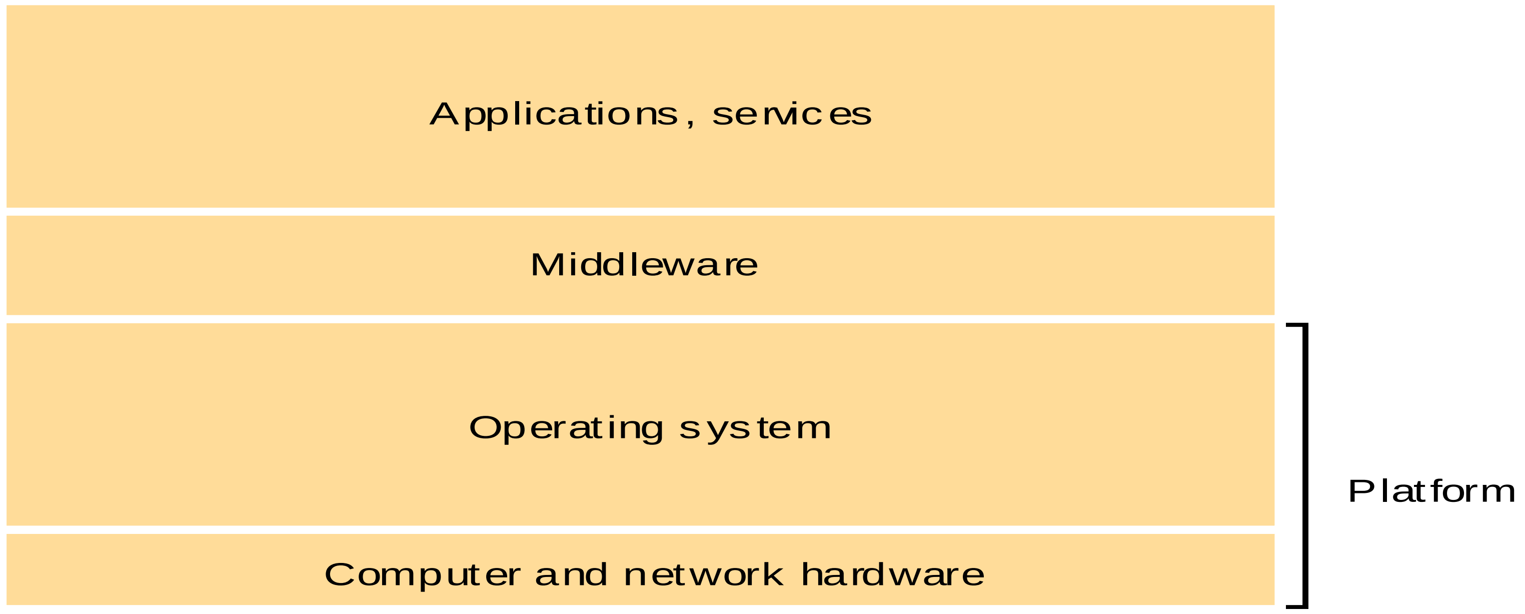


Architectural patterns

Layering: A complex system is partitioned into some layers.

- A layer uses the services offered by the beneath layer. A layer offers a software abstraction to the above layer.
- Higher layers are unaware of the implementation details of the below layer and also of any other layer beneath it.

Figure 2.7
Common view of a layered architecture

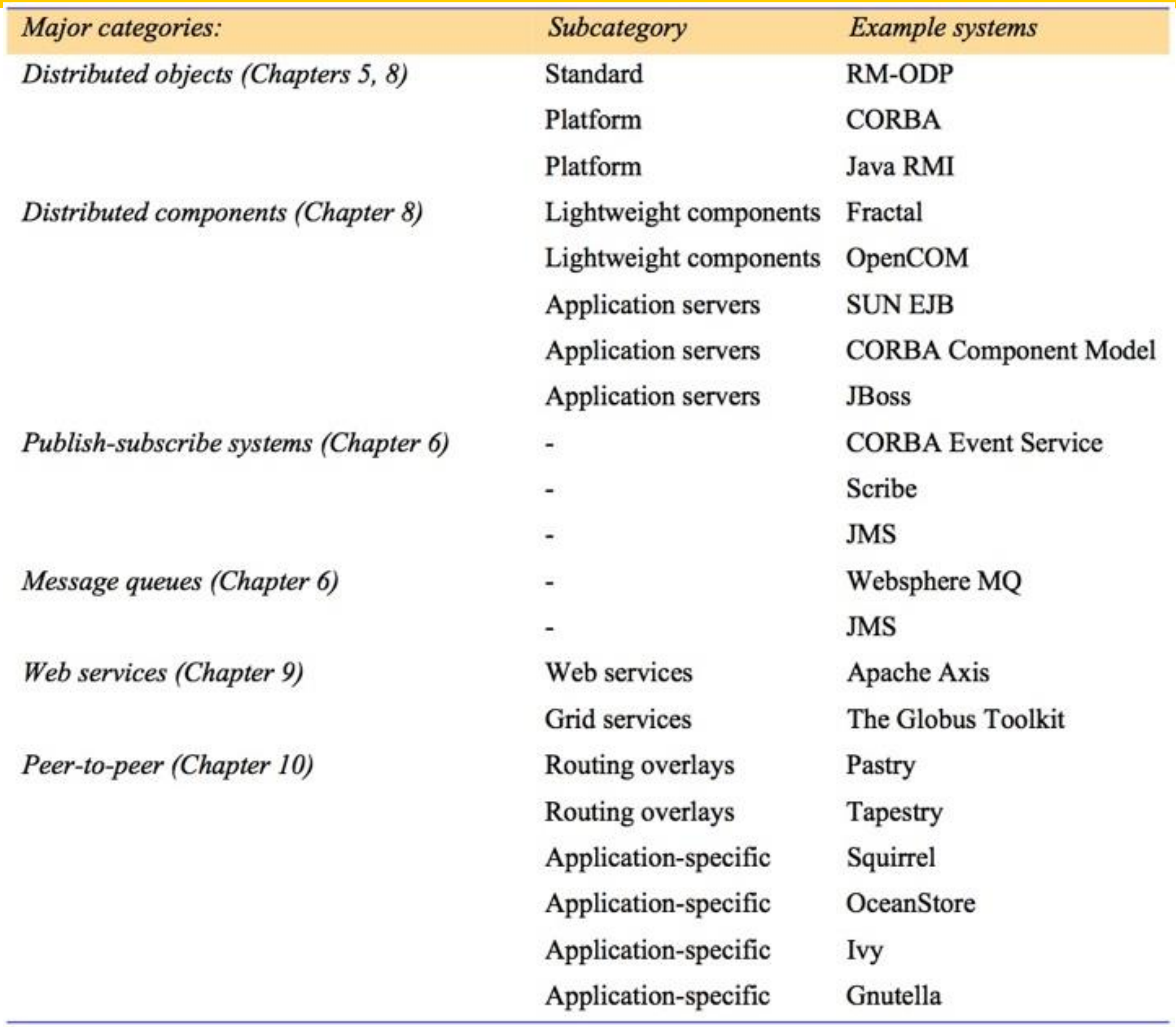


- Middleware is represented by processes or objects in a set of computers that interact with each other to implement communication and resource-sharing for distributed applications.
- It raises the level of communication activities of application programs through the support of communication abstractions such as remote method invocation; communication between a group of processes; notification of events; the partitioning, placement and retrieval of shared data objects, etc.
- Remote procedure calling packages such as Sun RPC and group communication systems such as ISIS were among the earliest instances of middleware

Categories based on communicating entities and paradigms:

- Distributed objects
- Distributed components
- Publish-subscribe systems
- Message queues
- Web services.

Figure 2.12
Categories of middleware



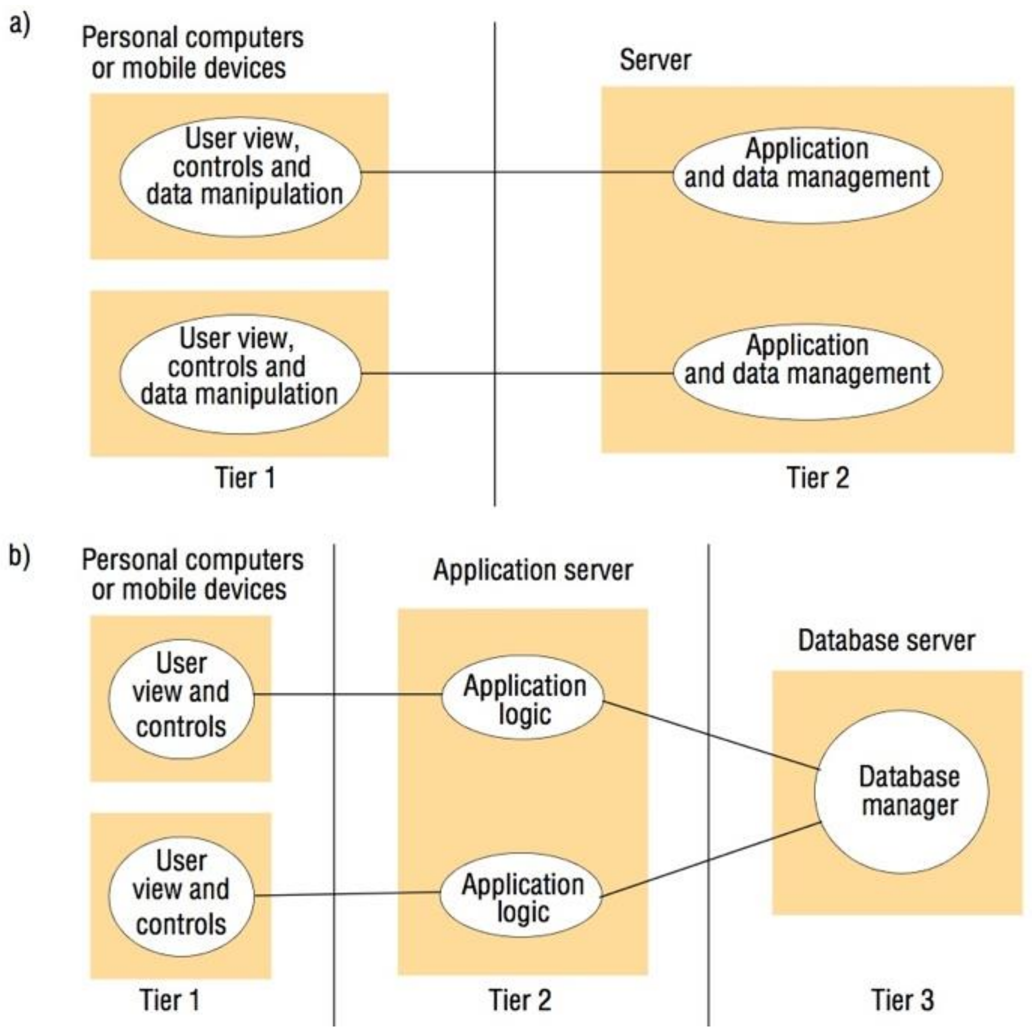
Major categories:	*Subcategory*	*Example systems*
Distributed objects (Chapters 5, 8)	Standard	RM-ODP
	Platform	CORBA
	Platform	Java RMI
Distributed components (Chapter 8)	Lightweight components	Fractal
	Lightweight components	OpenCOM
	Application servers	SUN EJB
	Application servers	CORBA Component Model
	Application servers	JBoss
Publish-subscribe systems (Chapter 6)	-	CORBA Event Service
	-	Scribe
	-	JMS
Message queues (Chapter 6)	-	Websphere MQ
	-	JMS
Web services (Chapter 9)	Web services	Apache Axis
	Grid services	The Globus Toolkit
Peer-to-peer (Chapter 10)	Routing overlays	Pastry
	Routing overlays	Tapestry
	Application-specific	Squirrel
	Application-specific	OceanStore
	Application-specific	Ivy
	Application-specific	Gnutella

Architectural patterns

Tiered architecture: Tiered architectures are complementary to layering.

- Layering deals with the vertical organization of services into layers of abstraction, tiering is a technique to organize functionality into appropriate servers on to physical nodes.
- The typical functional decomposition of an application:
- **Presentation logic**: is concerned with the handling user interaction and the view of the application presented to the user.
- **Application logic**: is the detailed application-specific processing
- **Data logic**: Persistent storage of the application typically in a database management system.

Figure 2.8
Two-tier and three-tier architectures

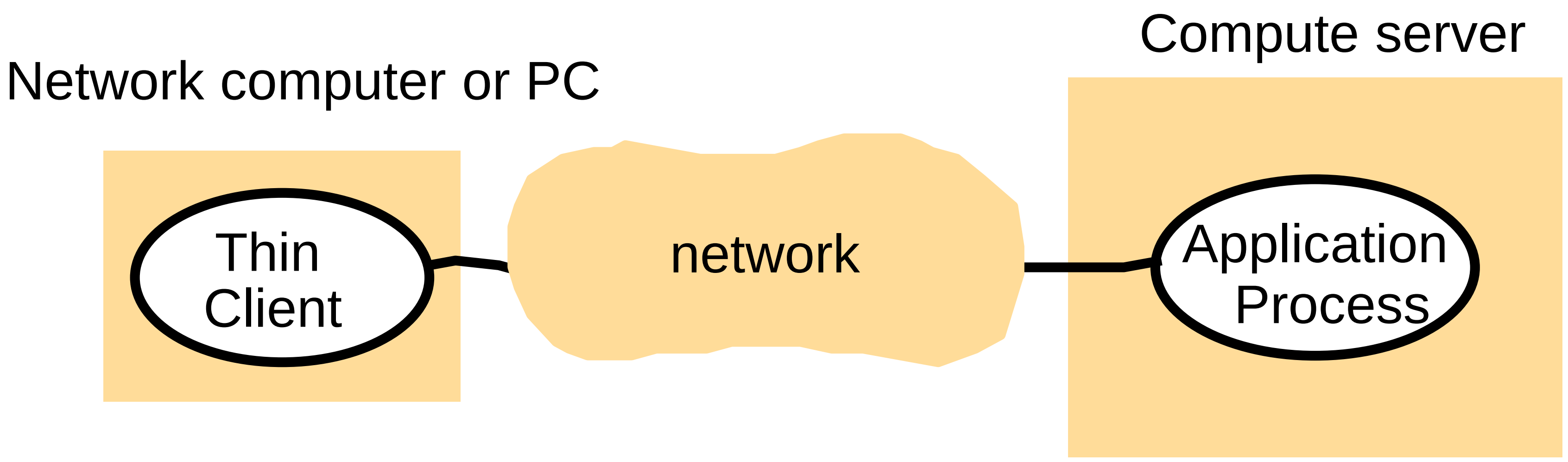

a)
Personal computers or mobile devices
Server
User view, controls and data manipulation
Application and data management
User view, controls and data manipulation
Application and data management
Tier 1
Tier 2
b)
Personal computers or mobile devices
Application server
Database server
User view and controls
Application logic
Database manager
User view and controls
Application logic
Tier 1
Tier 2
Tier 3

Architectural patterns

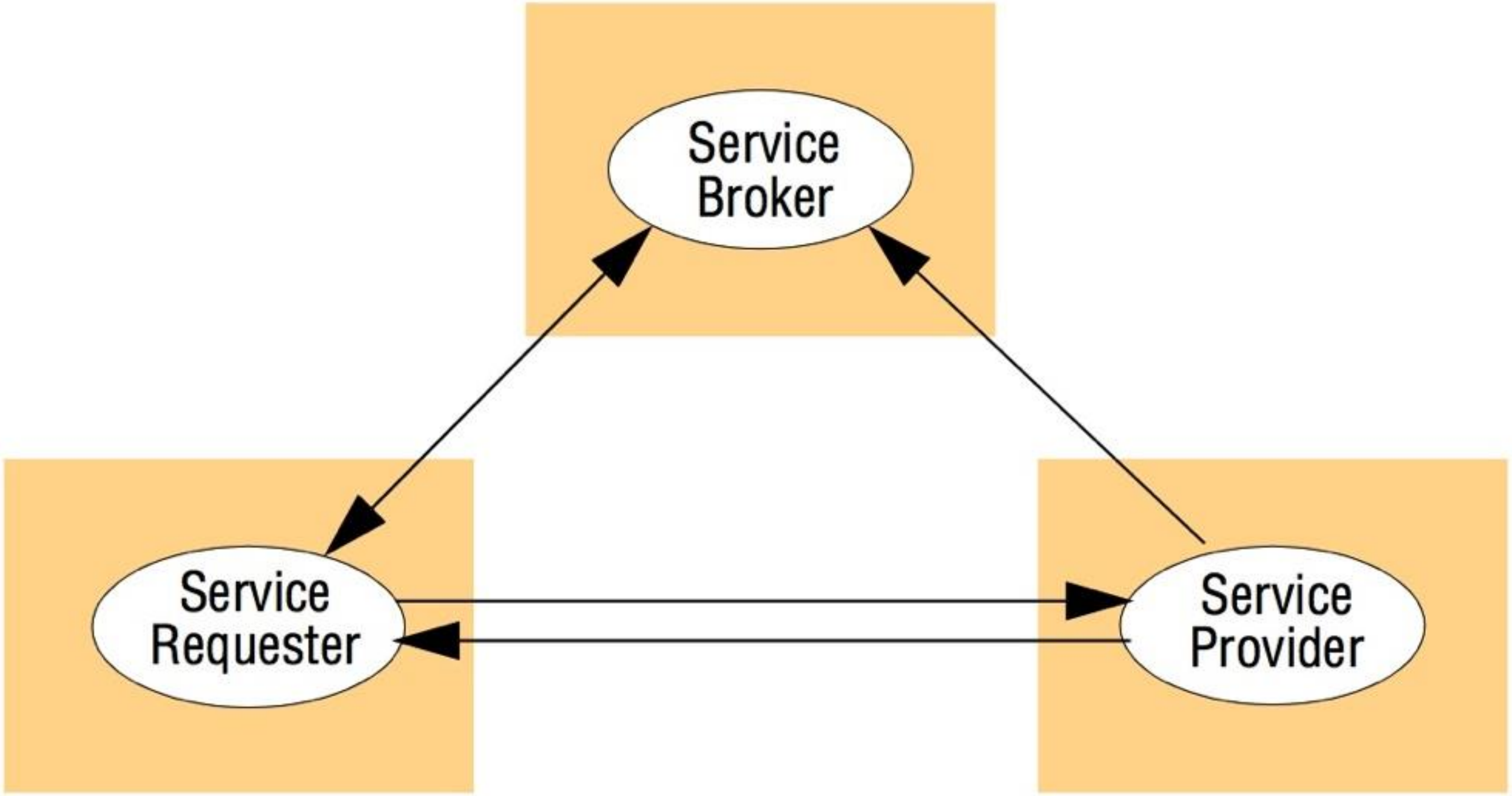
Thin clients: Moving complexity away from the user's device towards services in the internet and cloud.

- Thin client is a software layer that supports a local window-based interface; while executing application or service programs on a remote computer.
- The main drawback is the delay imposed in highly interactive graphical activities that is originated from transferring high-volume of data (image).
- Virtual Network Computing (VNC) is based on thin client. It provides remote access to graphical user interface. A VNC client (viewer) interacts with a VNC server through a low-level VNC protocol, which transfers frames and works with any OS or application.

Figure 2.10
Thin clients and compute servers



Architectural pattern
Web service



Fundamental models

Fundamental models are based on fundamental properties that allow us to be more specific about the system characteristics and the failure and security risks it might exhibit.

The aspects of distributed system that we capture are:

- Interaction
- Failure
- Security

Interaction aspect

Processes interact by passing messages resulting in communication (information flow) and coordination (synchronization and ordering of activities) between processes.

Interaction model:

- Multiple server processes might cooperate with one another to provide a service e.g. Domain Name System as well as Sun's network information service, which keeps replicated copies of password files at several servers in a LAN.
- A set of peer processes may cooperate with one another to achieve a common goal

Interaction aspect
Performance of communication channels

Communication over a network has the following performance characteristic:

Latency: The delay between the start of a message transmission from one process and the beginning of its receipt by another. Latency includes:

- The time taken for the first bit to reach its destination.
- The delay in accessing the network, which increases when the network is heavily loaded e.g. for Ethernet, the sending station should wait for the network to be free.
- The time taken by the OS communication services at both the sending and receiving processes , which depends on the current OS load.

Bandwidth: The total amount of information that can be transmitted in a given time.

Jitter: Variation in the time taken to deliver a serious of messages. It is mainly related to multimedia data e.g. if consecutive samples of audio data are played with different time intervals, the sound will be distorted.

2 variants of the interaction model

Synchronous distributed systems:

- The time to execute each step of a process has known lower and upper bounds.
- Each message transmitted over a channel is received within a known bounded time.
- Each process has a local clock whose drift rate from real time has a known bound.

Asynchronous distributed systems: (such as Internet)
There are no bounds on

- Process execution speeds
- Message transmission delays
- Clock drift rates

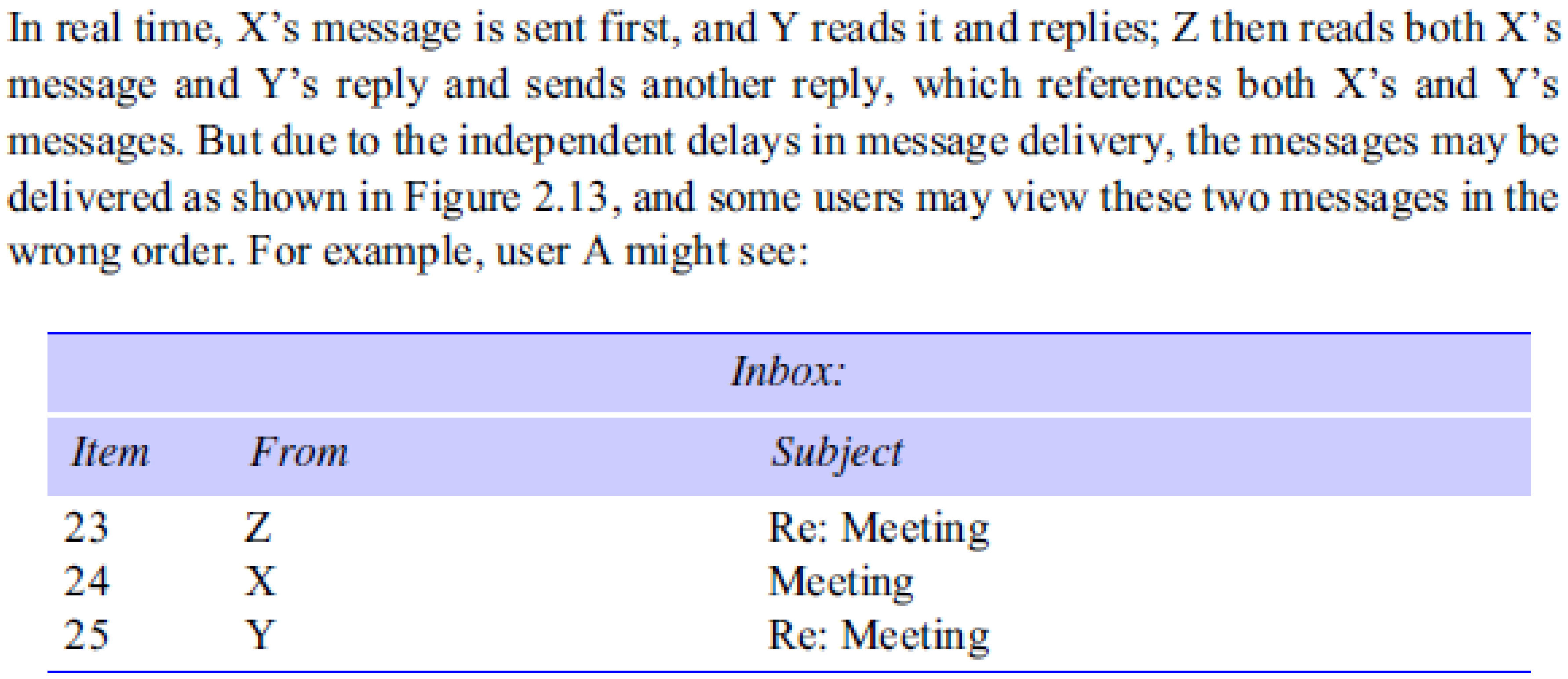
Event ordering

The execution of a system can be described in terms of events and their ordering despite the lack of accurate clocks.

For example consider the following scenario between a group of email users:

- User X broadcasts a message with subject Meeting
- Users Y and Z consecutively reply by sending the message Re:Meeting

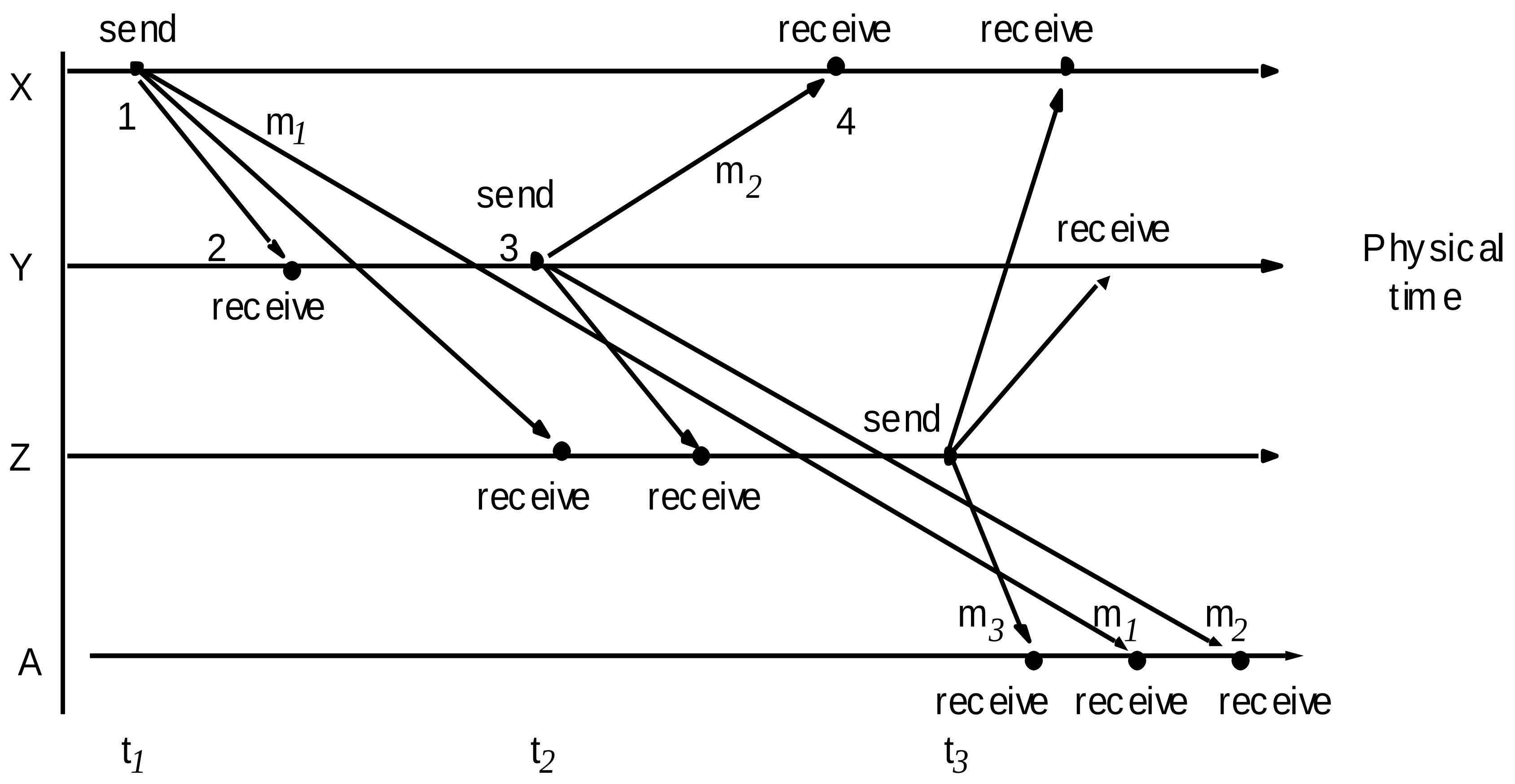
Scenario-



In real time, X's message is sent first, and Y reads it and replies; Z then reads both X's message and Y's reply and sends another reply, which references both X's and Y's messages. But due to the independent delays in message delivery, the messages may be delivered as shown in Figure 2.13, and some users may view these two messages in the wrong order. For example, user A might see:

Inbox:		
Item	*From*	*Subject*
23	Z	Re: Meeting
24	X	Meeting
25	Y	Re: Meeting

Figure 2.13
Real-time ordering of events



Logical time

Clocks cannot be synchronized perfectly across a distributed system.

Lamport [1978] proposed logical time that provides an ordering among the events.

- Logically, a message is received after it was sent.
- Replies are sent after receiving messages.
- Logical time assign a number to each event corresponding to its logical ordering, so that later events have higher numbers than earlier ones.