

---

# **Distributed Systems**

## 分布式系统

System Models

系统模型

---

# Our topic today

---

## 2.1 Physical model （物理模型）

## 2.1 Architectural model （体系结构模型）

- Architectural elements
  - Communication paradigms
  - Roles and responsibilities
  - Placement
- Architectural patterns

## 2.3 Fundamental model （基础模型）

- Interaction model （交互模型）
- Failure model （故障模型）
- Security model （安全模型）

# Descriptive models for distributed system design

---

## Physical model

- Most explicit description of a system.
- Capture **hardware composition** in terms of computers and their interconnecting networks.

## Architectural model

- Describes a systems in terms of computational and communication task performed by **computational elements**.

## Fundamental model

- **Abstract** perspective in order to study the individual aspects of a system.
- Three models are introduced: interaction model, failure model, and the security model.

# Difficulties for and threats to distributed systems

---

## Widely varying mode of use

- Component parts of the system are subject to wide variations in workload, e.g. some web pages are accessed several million times a day.
- Some parts of the systems might be disconnected or poorly connected, e.g. mobile computers.
- Some applications have special requirements such as high communication bandwidth and low latency, e.g. multimedia applications.

## Wide range of system environments

- Distributed systems accommodate heterogeneous hardware, operating systems, networks.
- Networks may differ widely in performance (wireless networks vs. LAN)

## Internal threats

## External threats

---

# Physical Model

# Introduction to physical model

---

A physical model is a representation of the underlying hardware elements of a distributed system that abstracts from specific details of the computer and networking technologies employed.

## Baseline physical model (基线物理模型)

- Hardware and software components located at networked computers communicate and coordinate their actions by passing messages.
- Very simple physical model of a distributed system.

# Three generations of distributed systems

---

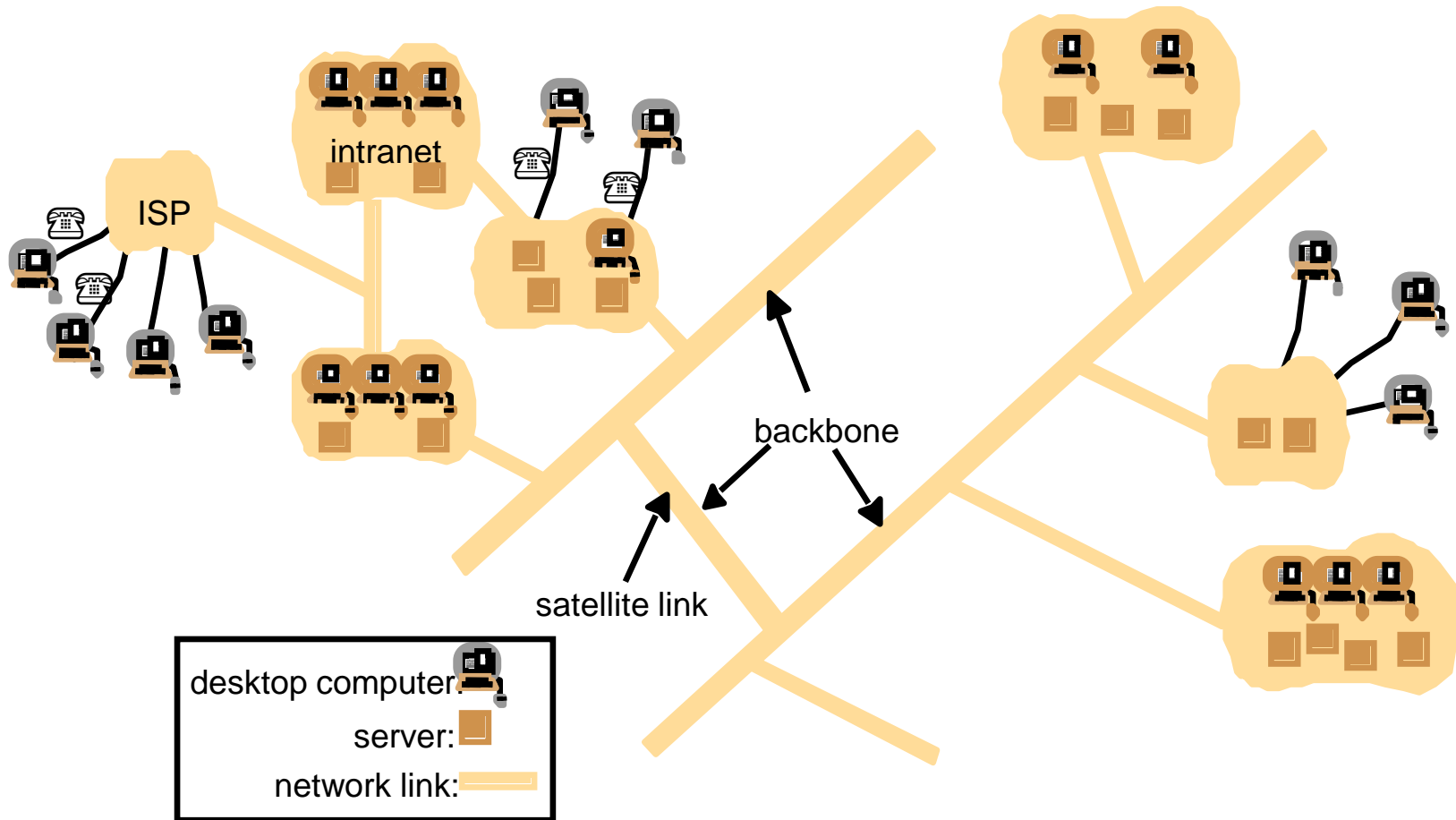
## **Early distributed systems**

- Emerged in the late 1970s and early 1980s because of the usage of local area networking technologies.
- System typically consisted of 10 to 100 nodes connected by a LAN, with limited Internet connectivity and supported services (e.g., shared local printer, file servers).

## **Internet-scale distributed systems**

- Emerged in the 1990s because of the growth of the Internet.

# Physical model of the internet-scale distributed system





# Three generations of distributed systems

---

## Early distributed systems

- Emerged in the late 1970s and early 1980s because of the usage of local area networking technologies.
- System typically consisted of 10 to 100 nodes connected by a LAN, with limited Internet connectivity and supported services (e.g., shared local printer, file servers).

## Internet-scale distributed systems

- Emerged in the 1990s because of the growth of the Internet.
- Infrastructure became global.

## Contemporary distributed systems

- Emergence of **mobile computing** leads to nodes that are location-independent
- Need to added capabilities such as service discovery and support for **spontaneous interoperation**
- Emergence of **cloud** computing and ubiquitous computing

# Distributed system of systems

---

Emergence of ultra-large-scale (ULS) distributed systems

Complex systems consisting of a series of subsystems that are systems in their own right and that come together to perform particular task or tasks

Example: environmental management system for flood prediction

- Consists of sensor networks deployed to monitor the state of various environmental parameters
- Coupled with systems that predict the likelihood for floods (running complex simulations)
- Additionally early warning systems to key stakeholders via mobile phones

# Generations of distributed systems

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

---

# Architectural Model

# Architectural model

---

An architectural model of a distributed system simplifies and abstracts the functions of the individual components of a distributed system and

- Organization of components across the network of computers.
- Their interrelationship, i.e., communicate with each other.

# Architectural elements

---

What are the entities that are communicating in the distributed system?

How do they communicate, or, more specifically, what communication paradigm is used?

What (potentially changing) roles and responsibilities do they have in the overall architecture?

How are they mapped on the physical distributed infrastructure (what is their placement)?

---

Architectural elements

**Communicating entities**

# System-oriented perspective

---

In distributed systems the entities that communicate are typically **processes**.

Exceptions:

- In primitive environments such as **sensor networks**, operating systems does not provide any abstractions, therefore nodes communicate.
- In most environments processes are supplemented by **threads**, so threads are more the endpoints of communications.



# Problem-oriented perspective

---

## Objects

- Computation consists of a number of interacting objects representing units of decomposition for the problem domain.
- Objects are accessed via interfaces.

## Components

- Resemble objects in that they offer problem-oriented abstractions, also accessed via interfaces.
- Specify not only their interfaces but also the assumptions they make in terms of other components/interfaces that must be present for a component to fulfil its function.

## Web services

- Software application which is identified via URI
- Supports direct interactions with other software agents

---

Architectural elements

# Communicating paradigms

# Types of communication paradigms

---

Interprocess communication

Remote invocation

Indirect communication

# Interprocess communication

---

- Low-level support for communication **between processes** in distributed systems including message parsing-primitives.
- Direct access to the API offered by Internet protocols (socket programming) and support for multicast communication.

# Remote invocation

---

Covering a range of techniques based on a two-way exchange between communicating entities.

Resulting in the calling of a remote operation, procedure or method

- Request-reply protocols: more a pattern imposed on an underlying message-parsing service to support client-server computing
- Remote procedure calls: procedures in processes on remote computers can be called as if they are procedures in the local address space
- Remote method invocation: a calling object can invoke a method in a remote object

# Indirect communication

---

## Group communication

- Delivery of messages to a set of recipients.
- Abstraction of a group which is represented in the system by a group identifier
- Recipients elect to receive message send to a group.

## Publish-subscribe-systems

- A large number of producers (publisher) distribute information items of interest (events) to a similarly large number of consumers (subscribers)

## Message queues

- Message queues offer a point-to-point service whereby producer processes can send messages to a specified queue and consumer processes can receive messages from the queue or being notified.

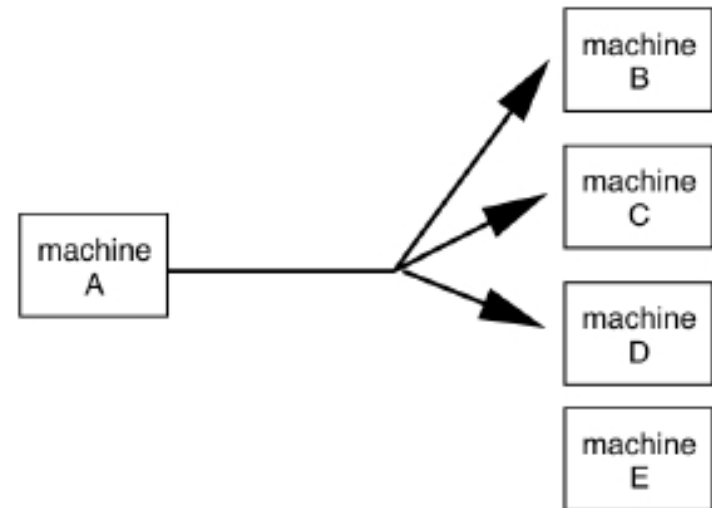
# Group communication

Two kinds of group communication:

- **Broadcast** (message sent to everyone)
- **Multicast** (message sent to specific group)

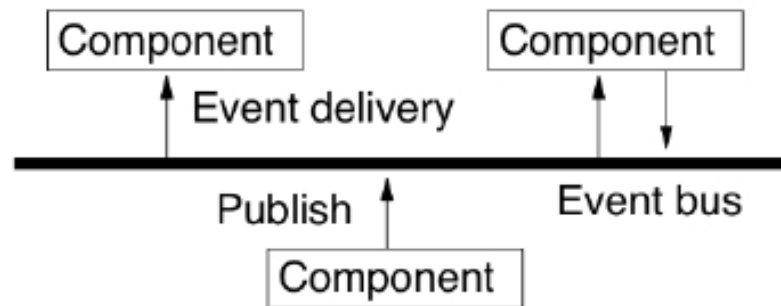
Used for:

- Replication of services
- Replication of data
- Service discovery
- Event notification



# Publish-subscribe-systems (or event-based communication)

- Communication through propagation of events
- Generally associated with publish/subscribe systems
- Sender process **publishes** events
- Receiver process **subscribes** to events and receives only the ones it is interested in





---

Architectural elements

# Roles and responsibilities

# Architectural styles

---

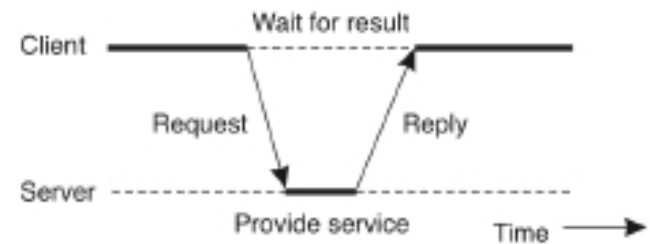
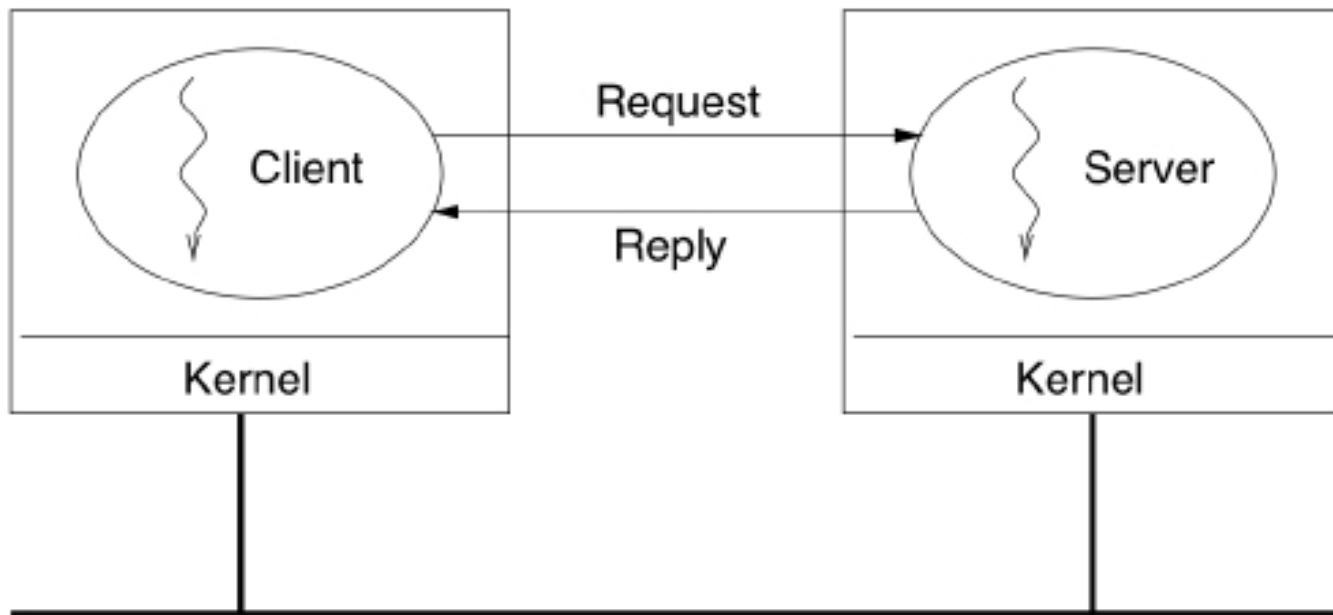
- Client-Server
- Peer-to-peer

---

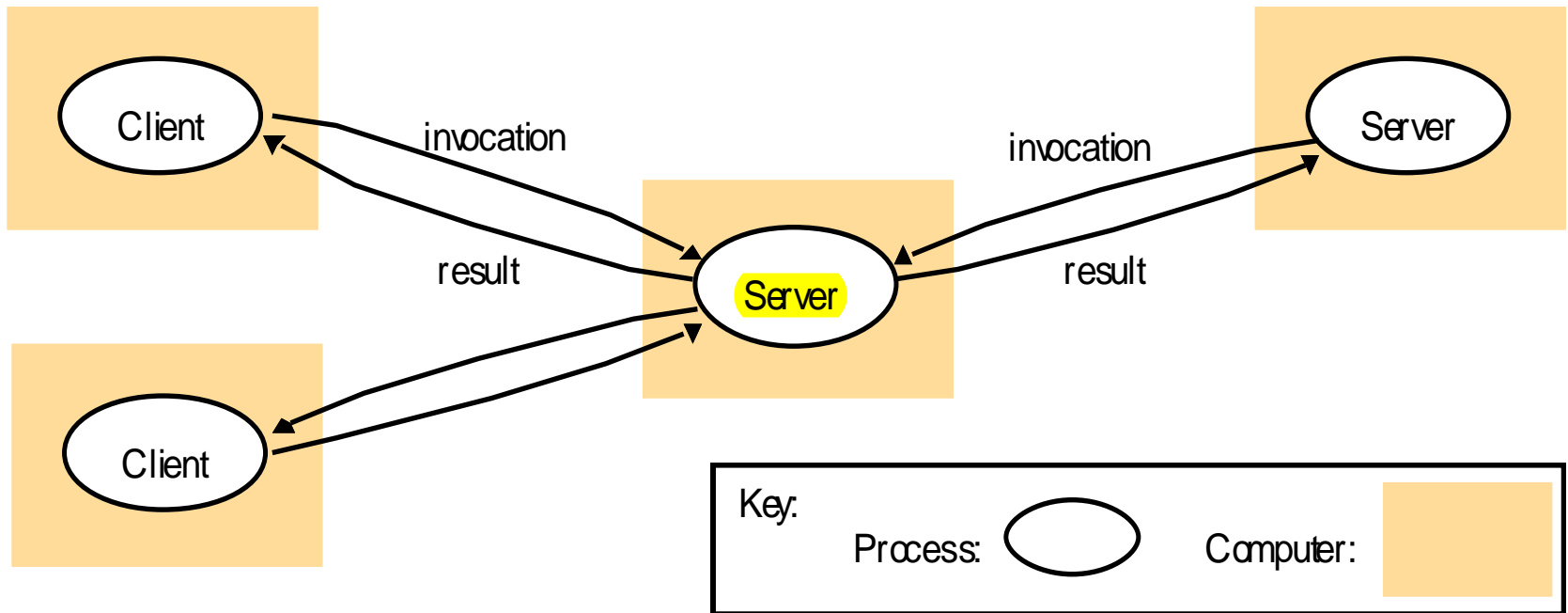
Roles and responsibilities

# Client-server

# Client-server



# Clients invoke individual servers



# Fundamental issue with client-server

---

Client-server offers a direct, relatively simple approach to the sharing of data and other resources

But it scales poorly

The centralization of service provision and management implied by placing a service at a single address does not scale well beyond the capacity of the computer that hosts the service and the bandwidth of its connections

Even though, there are several variations of the client-server architecture to respond to this problem but none of them really solve it.

There is a need to distribute shared resources much more widely in order to share the computing and communication loads amongst a much larger number of computers and network links.

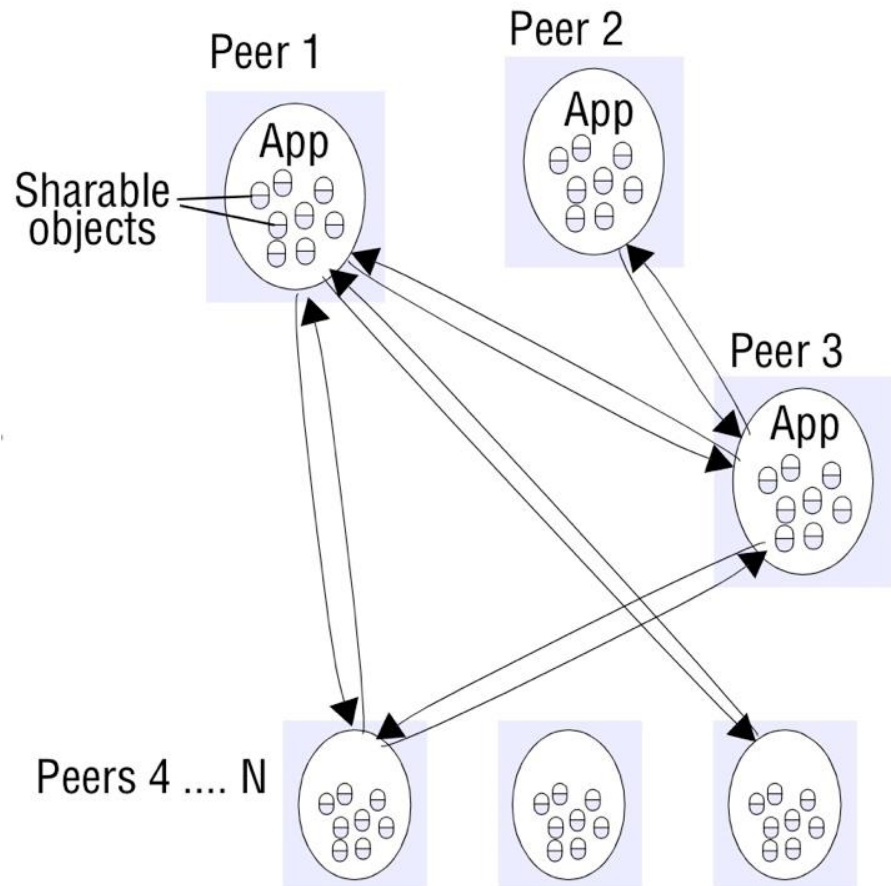
---

Roles and responsibilities

Peer-to-peer

# Peer-to-peer application

- Is composed of a large number of peer processes running on separate computers.
- All processes have client and server roles.
- Patterns of communication between them depends entirely on application requirements.
- Storage, processing and communication loads for accessing objects are distributed across computers and network links.
- Each object is replicated in several computers to further distribute the load and to provide resilience in the event of disconnection of individual computers.
- Need to place and retrieve individual computers is more complex than in client-server architecture.





---

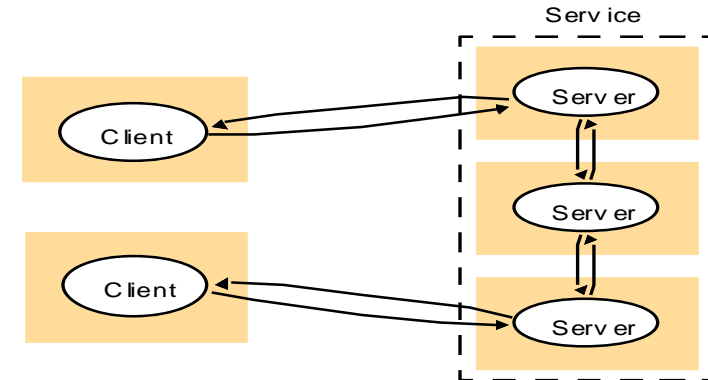
Architectural elements

# Placement

# Services provided by multiple servers

## Option 1

- Servers partition a set of objects in which the service is based and distribute them between themselves.
- Example
  - In the Web in which each web server manages its own set of resources.
  - User can employ a browser to access a resource at any one of the servers.



## Option 2

- Servers maintain replicated copies of them on several hosts.
- Example:
  - NIS (Network Information Service) used by computers on a LAN.

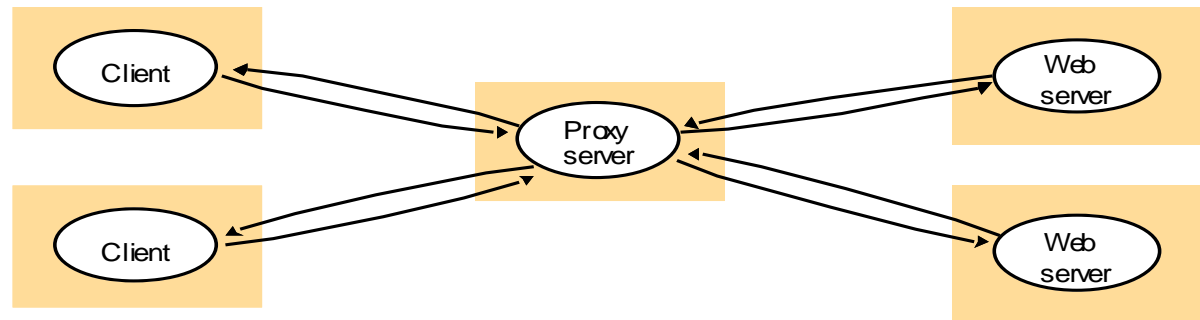
# Proxy server and caches

A cache is a **store of recently used data** objects that is **closer to the objects** themselves. Caches might be co-located with each client or may be located in a proxy server that can be shared by several clients.

## Process

- A new object is received at a computer -> it is **added to the cache** store, replacing some existing objects if necessary.
- Object is needed by the client process -> caching service **checks the cache** for an up-to-date copy.
- If copy is not available, this copy is fetched.

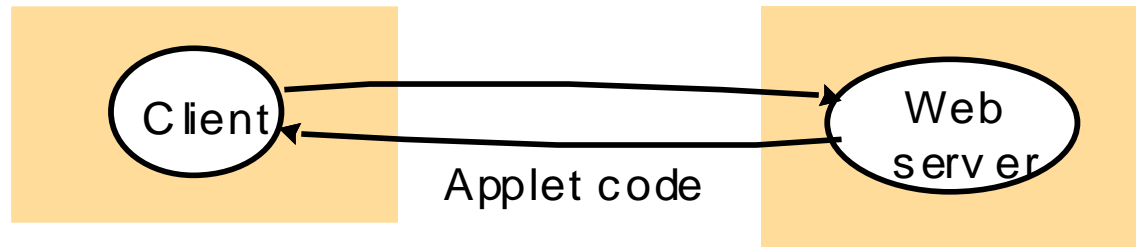
Examples?



# Mobile code

A typical well-known and widely-used example for mobile code are **applets**.

a) client request results in the downloading of applet code



b) client interacts with the applet



# Mobile agents

---

A mobile agent is a running program (both code and data) that travels from one computer to another in a network **carrying out a task** on someone's behalf, e.g. collecting information.

Benefits agents provide for creating distributed systems (Lange & Oshima, 1999)

- They reduce the network load.
- They overcome network latency.
- They encapsulate protocols.
- They execute asynchronously and autonomously.
- They adapt dynamically.
- They are naturally heterogeneous.
- They are robust and fault-tolerant.

Examples?

---

Architectural models

**Architectural patterns**

# Concept of layering

---

Vertical organization of services into a service layers.

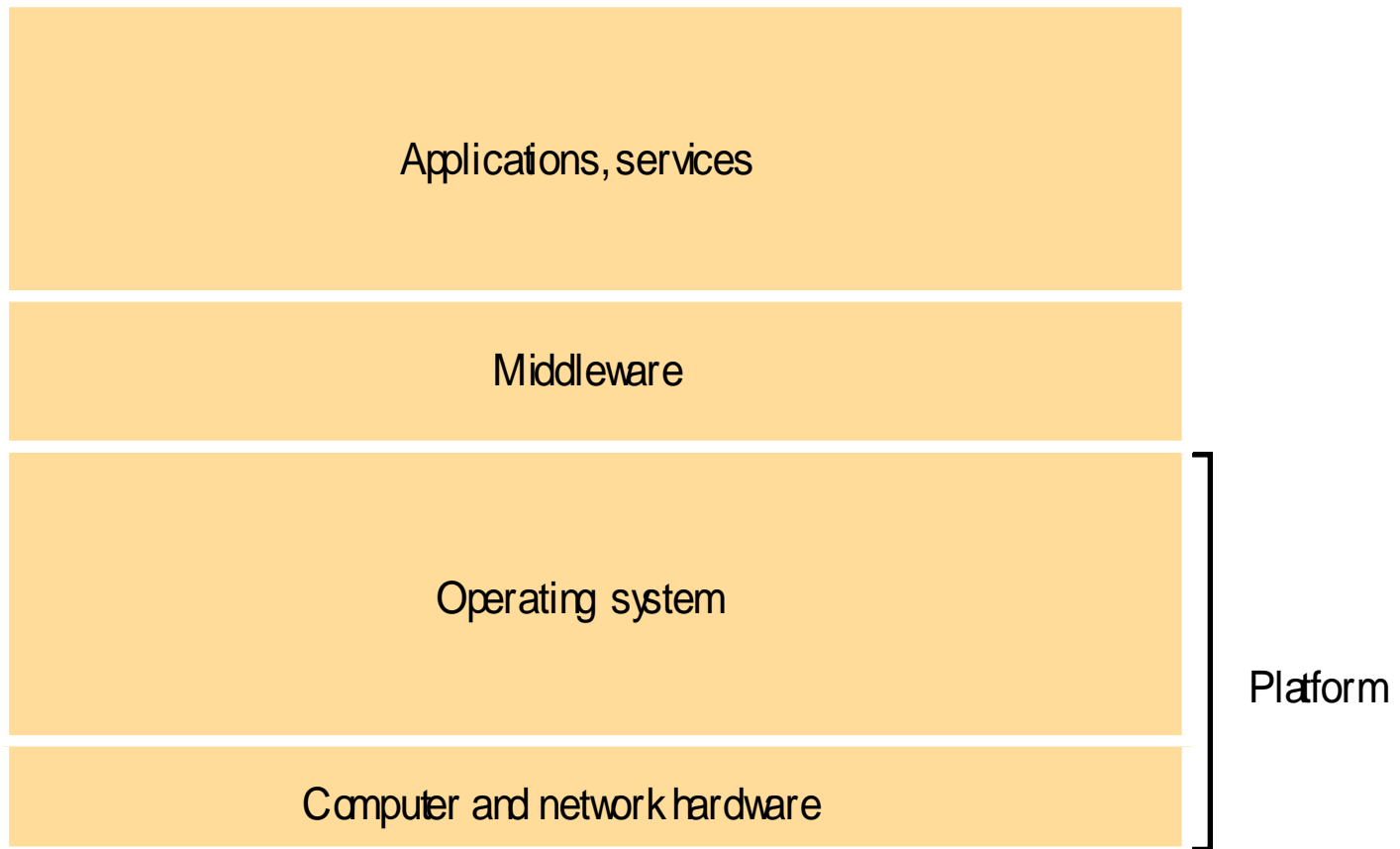
Distributed services can be provided by one or more server processes, interacting with each other and with client processes in order to maintain a consistent system-wide view of the service's resources.

## Example

- Network time service is implemented on the Internet based on the Network Time Protocol (NTP) by server processes running on hosts throughout the Internet that supply current time to any client that request it.

# Software and hardware service layers

---





# Platform

---

**Lowest level** hardware and software layers for distributed systems and applications

## Characteristics

- provide services to the layers above them
- implemented independently in each computer
- Bringing the system's programming interface up to a level that facilitates communication and coordination between processes

## Examples

x86/Windows, intel x86/Solaris, PowerPC/Mac OS X, Intel x86/Linux

# Middleware

---

Is a layer of software whose purpose is to **mask heterogeneity** and to provide a convenient programming model to application programmers.

is represented by processes or objects in a set of computers that interact with each other to implement communication and resource-sharing support.

Is concerned with providing useful building blocks for the construction of software components that can work with one another

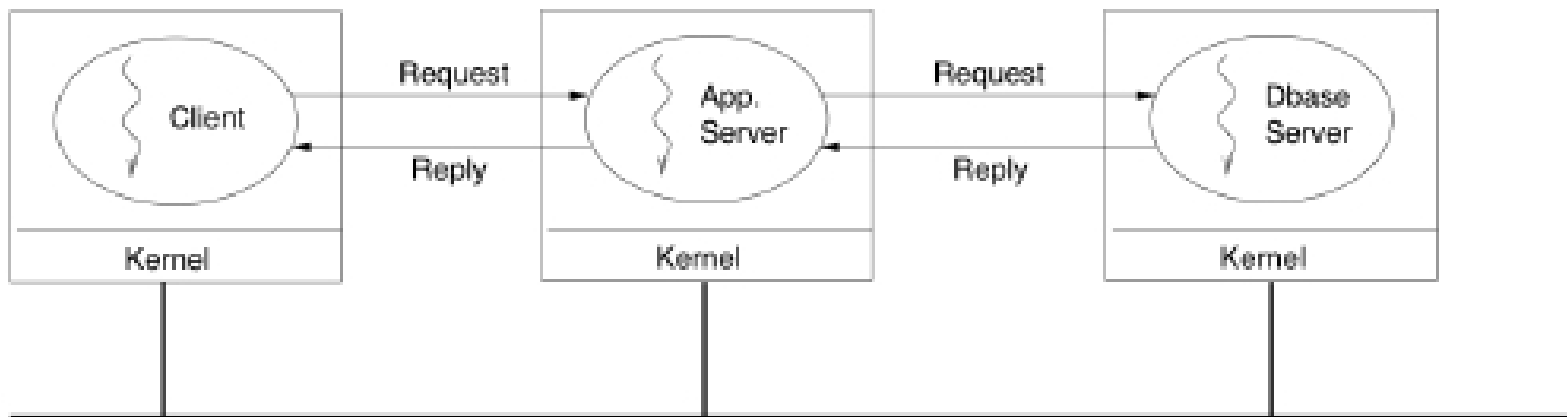
## Limitations of middleware

- Many distributed applications rely entirely on services provided by middleware to support their needs for communication and data sharing
- Example, application that is suited to the client-server model such a database of names and addresses rely on middleware that provides only remote method invocation.

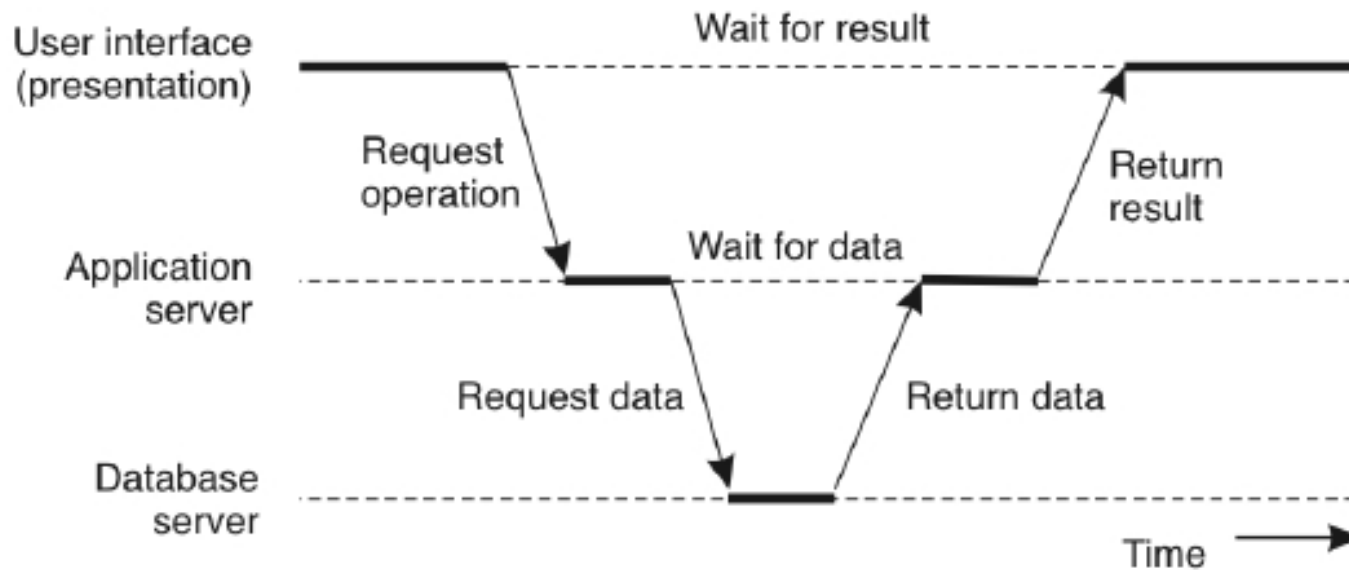
# Vertical Distribution (Multi-Tier)

An extension of the client-server architecture.

Distributes the traditional server functionality over multiple servers.



# Communication in a multi-tier system



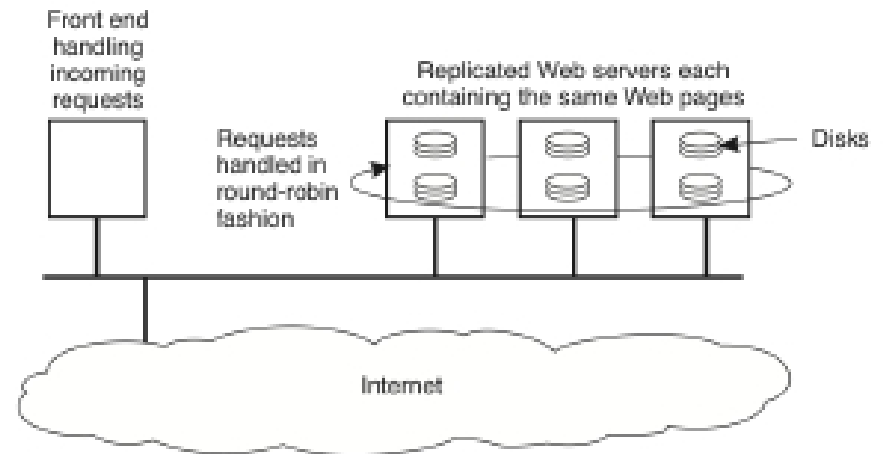
# Horizontal Distribution

Involves replicating a server's functionality over multiple computers.

Typical example: replicated Web server

- Each server machine contains a complete copy of all hosted Web pages.
- Client requests are passed on to the servers in a **round robin** fashion.

Is used to **improve scalability** (by **reducing the load** on individual servers) and reliability (by providing **redundancy**)



# Thin and fat client implementations

---

Decomposed a typical client-server application into three logical parts

- the interface part.
- the application logic part, and.
- the data part.

Thin client implementation

- Provides a minimal user interface layer, and leave everything else to the server.

Fat client implementation

- Include all of the user interface and application logic in the client.
- Rely only on the server to store and provide access to data.

Implementations in between will split up the interface or application logic parts over the clients and server in different ways.

---

# Fundamental models

# Requirements on the fundamental model

---

Questions that should be addressed by a system model

1. What are the main entities of the system?
2. How do they interact?
3. What are the characteristics that affect their individual and collective behavior?

Aspects of distributed systems that are considered are

Interaction

Failure

Security



---

Fundamental models

**Interaction models**

# Performance of communication channels

## 传播延迟和传输延迟

### Latency

- Delay between the start of a message's **transmission** from one process and the beginning of its receipt by another
- It includes:
  - Time taken for the first string of bits transmitted through a network to reach its destination.
  - Delay in accessing the network.
  - Time taken by the operating system communication services at both the sending and the receiving processes.

### Bandwidth

- total amount of information that can be transmitted over a computer network in a given time.

### Jitter

- Variation in the time taken to deliver a series of messages.

# Two variants of the interaction model

---

## *Synchronous distributed systems*

The following bounds are defined:

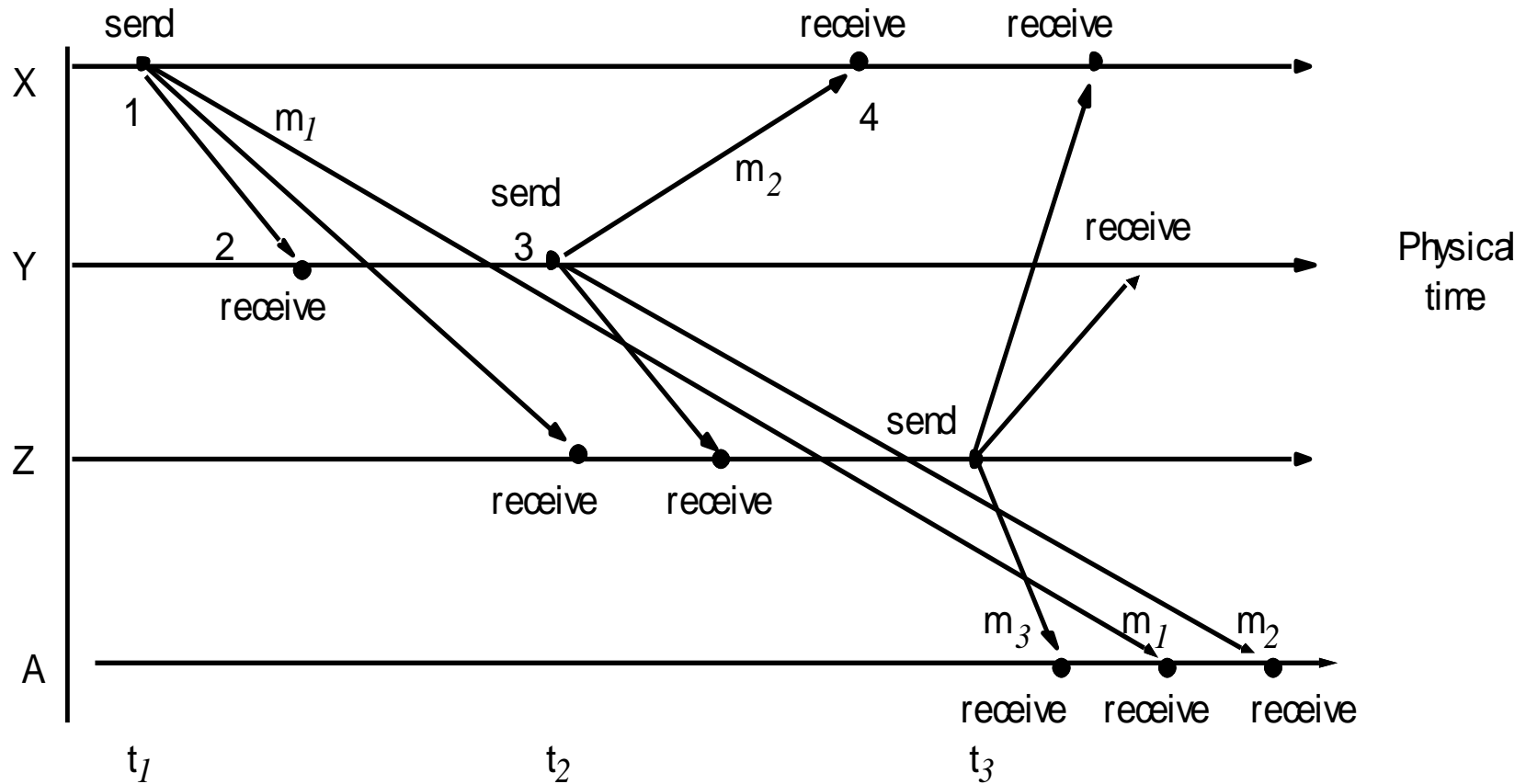
- The time to execute each step of a process has known **lower and upper bounds**
- Each message transmitted over a channel is received within a known **bounded time.**
- Each process has a **local clock** whose drift rate from real time has known bound.

## *Asynchronous distributed system*

There are **no bounds** on:

- Process execution speed
- Message transmission delays
- Clock drift rate

# Event ordering



---

Fundamental models  
**Failure models**

# Introducing the failure model

---

The failure model defines ways in which failure may occur in order to provide an understanding of the effects of failure.

Taxonomy of failures of processes and communication channels (Hadzilacos & Toueg, 1994)

- Omission failures.

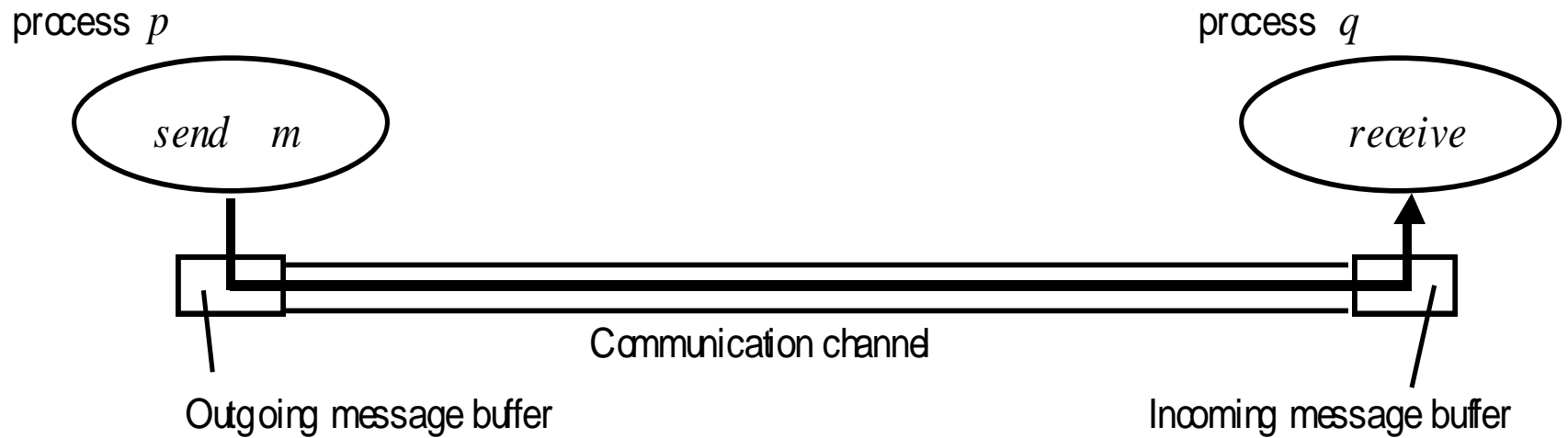


遗漏式

- Arbitrary failures.

- Timing failures.

# Processes and channels



# Omission failures

<i>Class of failure</i>	<i>Affects</i>	<i>Description</i>
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a <i>send</i> , but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.



# Arbitrary failures

---

Often called Byzantine failure.

This is the worst possible failure semantics, in which any type of error may occur.

Example of an arbitrary failure of a process.

- A process arbitrarily omits intended processes steps or takes unintended processing steps.

Example of an arbitrary failure of a communication channel

- Message content may be corrupted, nonexistent messages may be delivered or real messages may be delivered more than once.
- Solutions: *checksum* to detect corrupted messages and *message sequence numbers* to detect nonexistent and duplicated messages.

# Timing failures

---

<i>Class of Failure</i>	<i>Affects</i>	<i>Description</i>
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

Result response being unavaible to clients.

---

Fundamental models  
**Security models**

# Introducing the security model

---

The security of a distributed system can be achieved by securing the processes and the channels used for their interactions and by protecting the objects that they encapsulate against unauthorized access.

# Summary

---

- Three generations of distributed systems and the emergence of ultra-large-scale (ULS) distributed systems.
- Types of communication paradigms:
  - Interprocess communication
  - Remote invocation
  - Indirect communication
- Architectural styles: client-server and peer-to-peer
- Vertical distribution (Multi-Tier) and horizontal distribution of c/s systems
- Characteristics of synchronous distributed systems and asynchronous distributed system
- Omission failures, arbitrary failures, timing failure in distributed systems