System Models

Architectural System Model

- An architectural model of a distributed system is concerned with the placement of its parts and the relationships between them.
- **Examples**
  - Client-server
  - Peer-to-peer
- **Interaction Model**
  - Deals with performance and the difficulty to set time limits (e.g., in message delivery).
- **Failure Model**
  - Gives a precise specification of the faults of the processes and the links.
  - Defines reliable communication and correct processes.

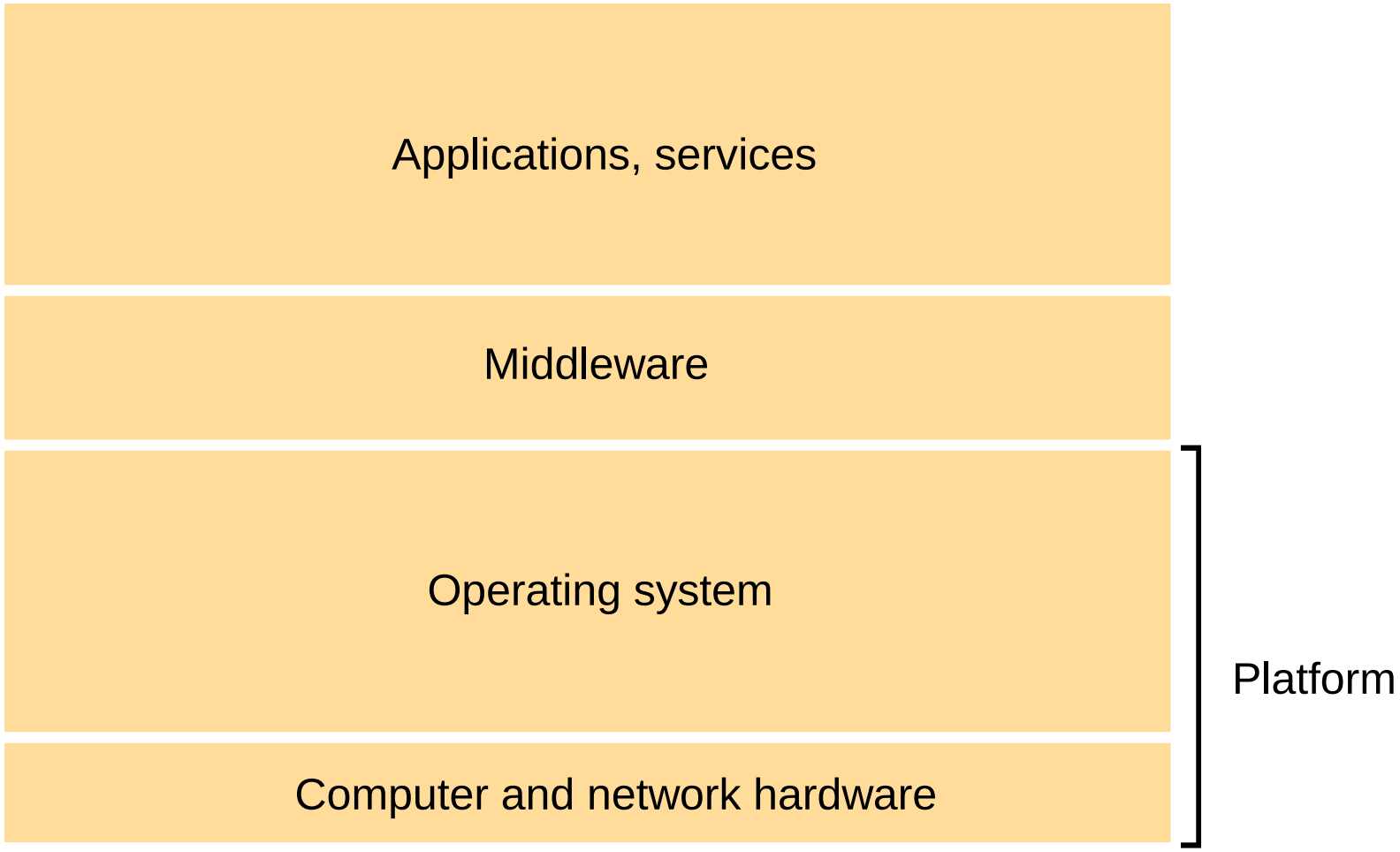
Architectural Models

- The architecture abstracts the functions of the individual components of the distributed system.
  - ensure that the structure will meet present and likely future demands
  - make the system reliable, manageable, adaptable, and cost-effective
- Classification of processes
  - Servers, clients, peers
  - Identifies responsibilities, helps to assess their workloads, determines the impact of failures in each of them.

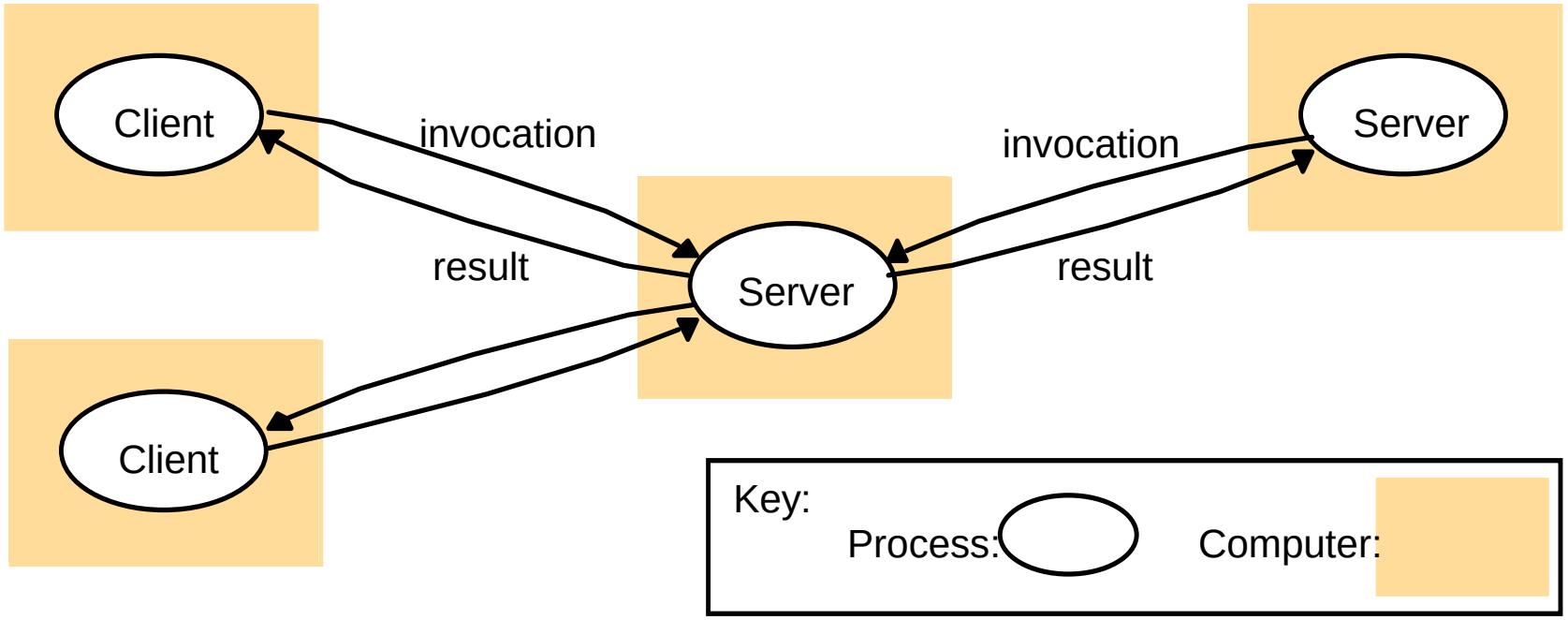
Software Layers

- **Software architecture** refers to services offered and requested between processes located in the same or different computers.
  - structuring of software as layers or modules
  - Service layers
- Distributed service
  - One or more server processes
  - Client processes
- Platform -> the lowest level hardware and software layers
  - Examples: Intel x86/Windows, Intel x86/Solaris, PowerPC/MAC OS, Intel x86/Linux
- Middleware
  - masks heterogeneity & provides a convenient programming model
  - Provides useful building blocks:
    - Remote method invocation, communication between a group of processes, notification of events, partitioning, placement and retrieval of data or objects, replication, transmission of multimedia data in real time

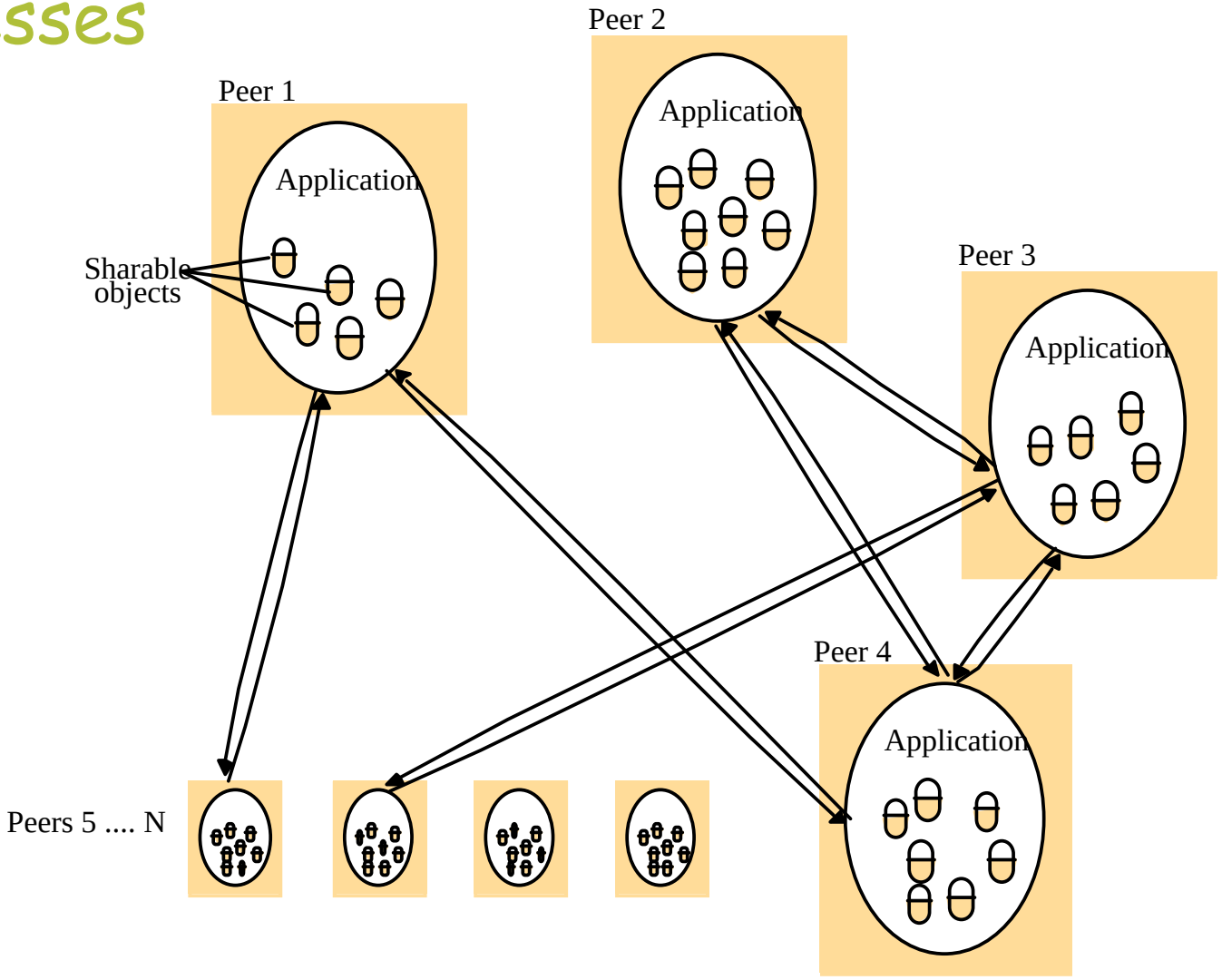
Software and hardware service layers in distributed systems



Clients invoke individual servers



A distributed application based on peer processes

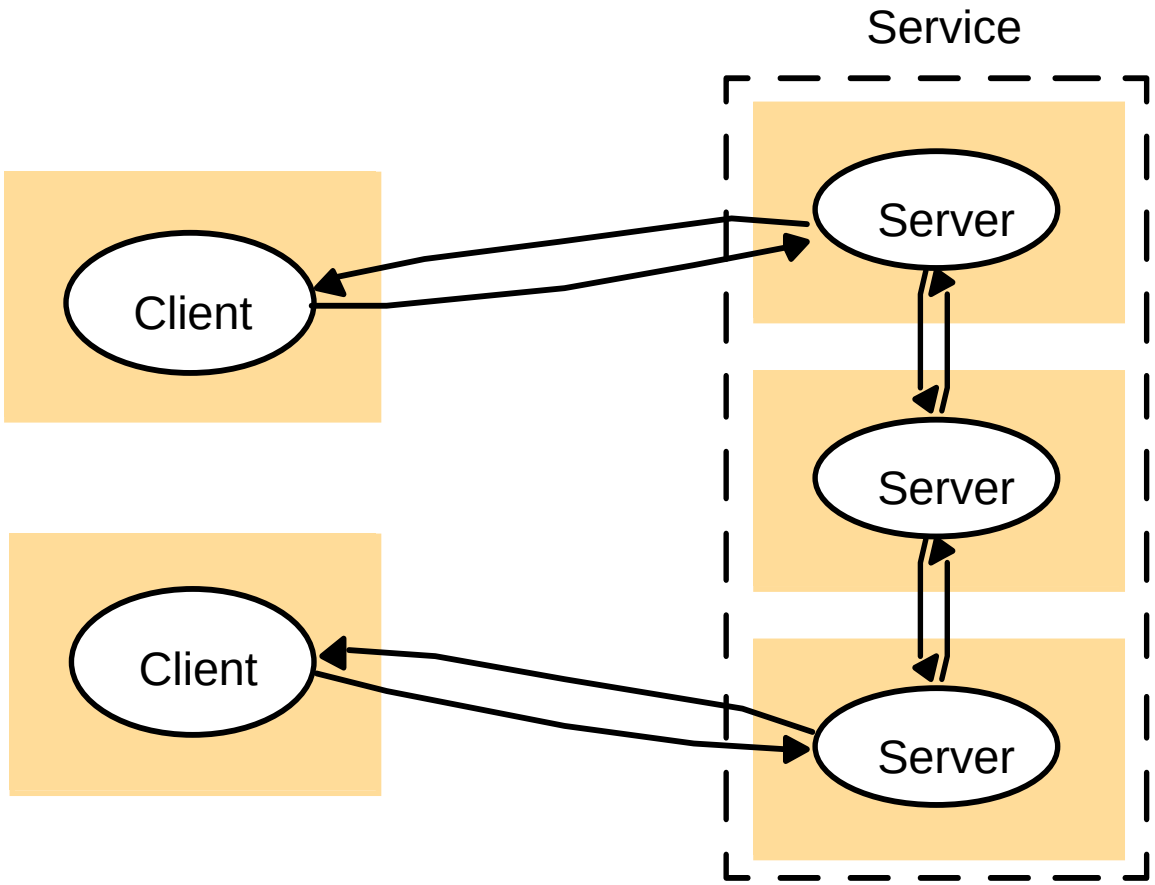


Variations

Derived from the consideration of the following factors:

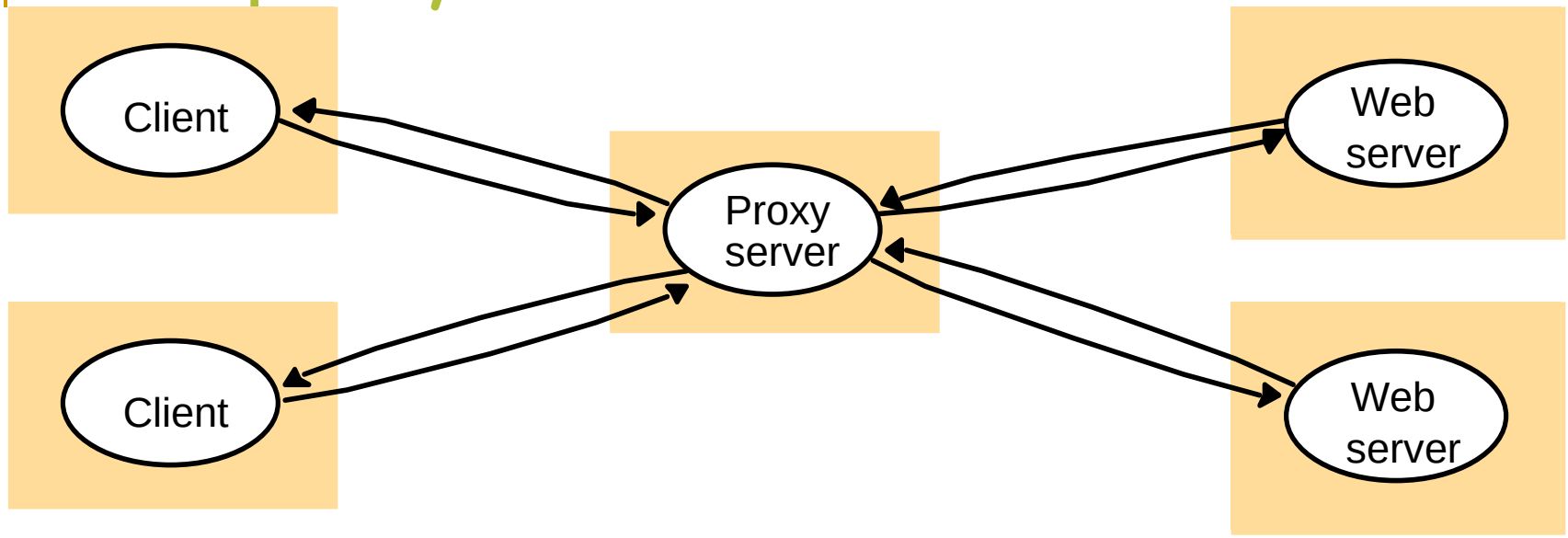
- Use of multiple servers and caches to increase performance and resilience
- Use of mobile code and mobile agents
- User's need for low-cost computers with limited hardware resources that are simple to manage
- Requirement to add and remove mobile devices in a convenient manner

A service provided by multiple servers



- Partition the set of objects and distribute them between themselves
- maintain replicated copies
- Web
- SUN Network Information Service

Web proxy server



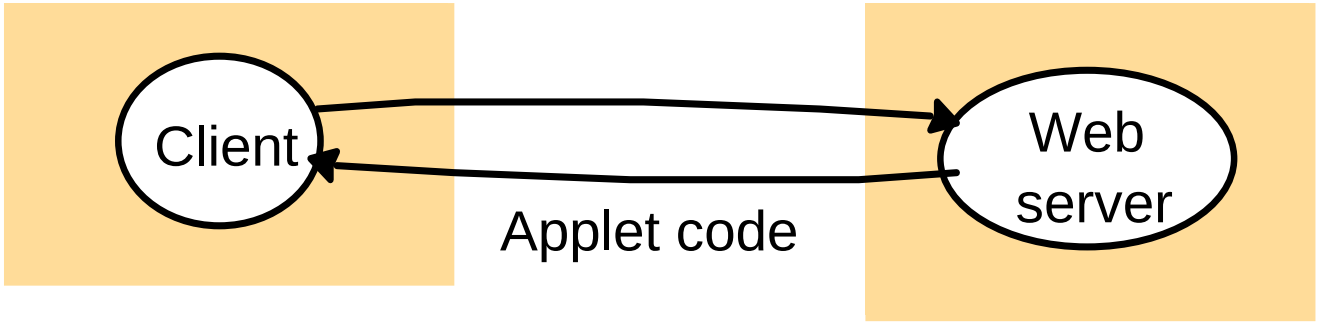
A cache is a storage facility of recently used data objects that is closer to the objects themselves.

Web proxy servers provide a shared cache of web resources for the client machines at a site or across several sites.

- Increase availability
- Increase performance
- Reduce the load on the wide-area network and web server

Mobile Code - Web applets

a) client request results in the downloading of applet code



b) client interacts with the applet

- Good interactive response

Mobile Agents

- A mobile agent is a running program that travels from one computer to another in a network carrying out a task on someone's behalf.
- A mobile agent may make many invocations to local resources at each site it visits (e.g., access to individual database entries).

Might be used to:

- install software on the computers of an organization
- compare the prices of products of vendors

Negative aspects

- mobile agents are a potential security threat to the resources
- they can be themselves vulnerable (they may not be able to complete their tasks if they are refused access to the information needed)

Design Requirements for Distributed Architectures

Performance Issues

- Responsiveness
  - The speed of a remote invocation depends on:
    - The load and performance of the server and network
    - Delays in all the software components (client and server operation systems and middleware, code of the process that implements the service)
  - Transfer of data is slow
- Throughput
  - Rate at which computational work is done
  - Fairness
- Balancing of Computational Loads
  - Applets remove load from the server
  - Use several computers to host a single service

Design Requirements for Distributed Architectures

- **Quality of Service**
  - Reliability (the ability of a system to perform and maintain its function in every circumstance).
  - Performance
  - Adaptability (the ability of a system to adapt itself efficiently and fast to changed circumstances)
  - Resource Availability

- Some applications handle **time-critical data** – streams of data that are required to be processed or transferred from one process to another at a fixed rate.

Design Requirements for Distributed Architectures

- **Caching and Replication**
  - Cached copies of resources should be kept up-to-date when the resource at a server is updated.
  - A variety of cache-coherency protocols are used to suit different applications.
- **Web Caching Protocol**
  - Web browsers and proxy servers cache responses to client requests from web servers
  - The cache-consistency protocol can provide browsers with fresh copies of the resources held by the web server, but for performance reasons the freshness condition can be relaxed.
  - A browser or proxy can validate a datum with the server.
  - Web servers assign approximate expiry times to their resources
  - The expiry time of the resource and the current time at the server are attached to the response.

Design Requirements for Distributed Architectures

- Dependability Issues
  - Dependable applications should continue to function correctly in the presence of faults in hardware, software and networks.
- Attributes
  - Availability – readiness for correct service
    - The ratio of the total time a functional unit is capable of being used during a given interval to the length of the interval
  - Reliability
    - the probability that a device will perform its intended function during a specified period of time under stated conditions
    - Replication
      - Multiple computers – multiple communication paths
      - Several replicas of a data item
      - Retransmission of messages, etc.
  - Safety – absence of incorrect behavior
  - Integrity – absence of improper system alteration
  - Maintainability – ability to undergo modifications and repairs
    - correct defects, meet new requirements, make future maintenance easier, cope with a changed environment)

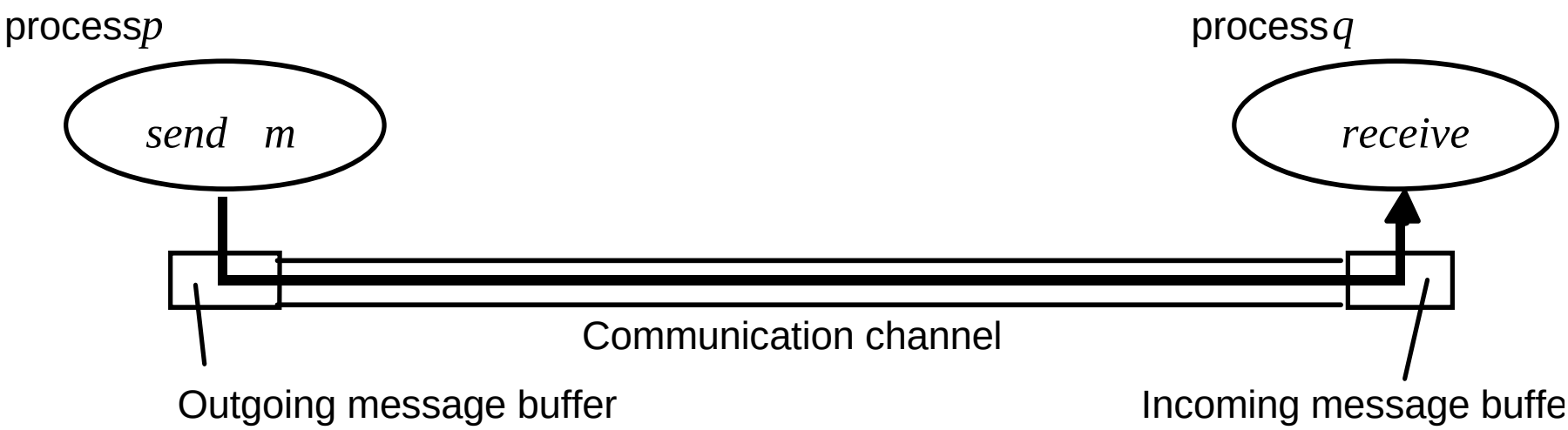
Fundamental Models

- A model contains only the essential ingredients needed to understand and reason about some aspects of a system's behavior.
- A system model has to address the following:
  - What are the main entities in the system?
  - How do they interact?
  - What are the characteristics that affect their individual and collective behavior?
- Purpose
  - Make explicit all the relevant assumptions about the system we are modeling
  - Make generalizations concerning what is possible or impossible, given those assumptions.
    - General purpose algorithms
    - Desirable properties
- Interaction
  - Communication takes place with delays
  - Maintaining the same notion of time across all nodes of a distributed system is difficilt.
- Failure

Interaction Model

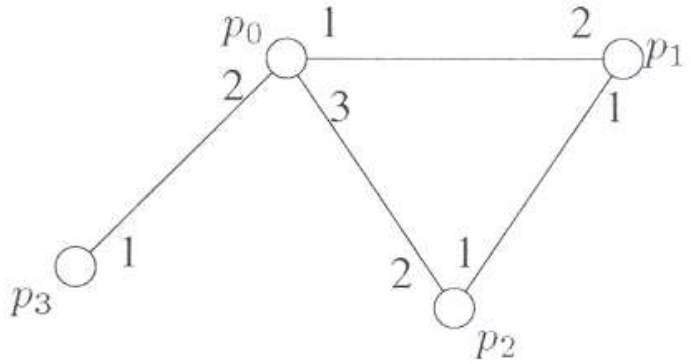
- The behavior and state of distributed systems can be described by a distributed algorithm
  - A definition of the steps to be taken by each of the processes, including the transmission of messages between them.
- Messages are transmitted to transfer information between processes and to coordinate their activity.
- The computing rates of processes and the timing of the transmission of messages cannot in general be predicted.
- Each process has its own state, consisting of the set of data that it can access and update (i.e., its local variables).

Processes and Channels



Formal Model of Message-Passing Systems

- There are n processes in the system: p_0, .., p_{n-1}
- Each process is modeled as a state machine.



- The state of each process is comprised by its local variables and a set of arrays. For instance, for p_0, the state includes six arrays:
  - $inbuf_0[1]$, …, $inbuf_0[3]$: contain messages that have been sent to p_0 by p_1, p_2 and p_3, respectively, but p_0 has not yet processed.
  - $outbuf_0[1]$, …, $outbuf_0[3]$: messages that have been sent by p_0 to p_1, p_2, and p_3, respectively, but have not yet been delivered to them.

Formal Model of Message-Passing Systems

- The state of process p_i excluding the $outbuf_i[l]$ components, comprises the accessible state of p_i.
- Each process has an initial state in which all inbuf arrays are empty.
- At each step of a process, all messages stored in the inbuf arrays of the process are processed, the state of the process changes and a message to each other neighboring process can be sent.
- A configuration is a vector $C = (q_0, .., q_{n-1})$ where q_i represents the state of p_i.
  - The states of the outbuf variables in a configuration represent the messages that are in transit on the communication channels.
  - In an initial configuration all processes are in initial states.

Formal Model of Message-Passing Systems

- **Computation event, comp(i)**
  - Represents a computation step of process p_i in which p_i's transition function is applied to its current accessible state.
- **Delivery Event, del(i,j,m)**
  - Represents the delivery of message m from processor p_i to processor p_j (i.e., message m is placed in one of the inbuf buffers of p_j)
- The behavior of a system over time is modeled as an **execution**, which is a sequence of configurations alternating with events.
- This sequence must satisfy a variety of conditions.
  - Safety condition
    - Holds in every finite prefix of the execution (it states that nothing bad has happened yet)
  - Liveness condition
    - Holds a certain number of times (it states that eventually something good must happen)

Performance of Communication Channel

- **Latency**

  The delay between the start of a message's transmission from one process and the beginning of its receipt by another.

- **Bandwidth**

  Total amount of information that can be transmitted over it in a given time. The latency includes:

  - Time taken for the first of a string of bits transmitted through the network to reach its destination + delay in accessing the network + time taken by the OS communication services at both sender and receiver

- **Jitter**

  The variation in the time taken to deliver a series of messages.

Computer Clocks and Timing Events

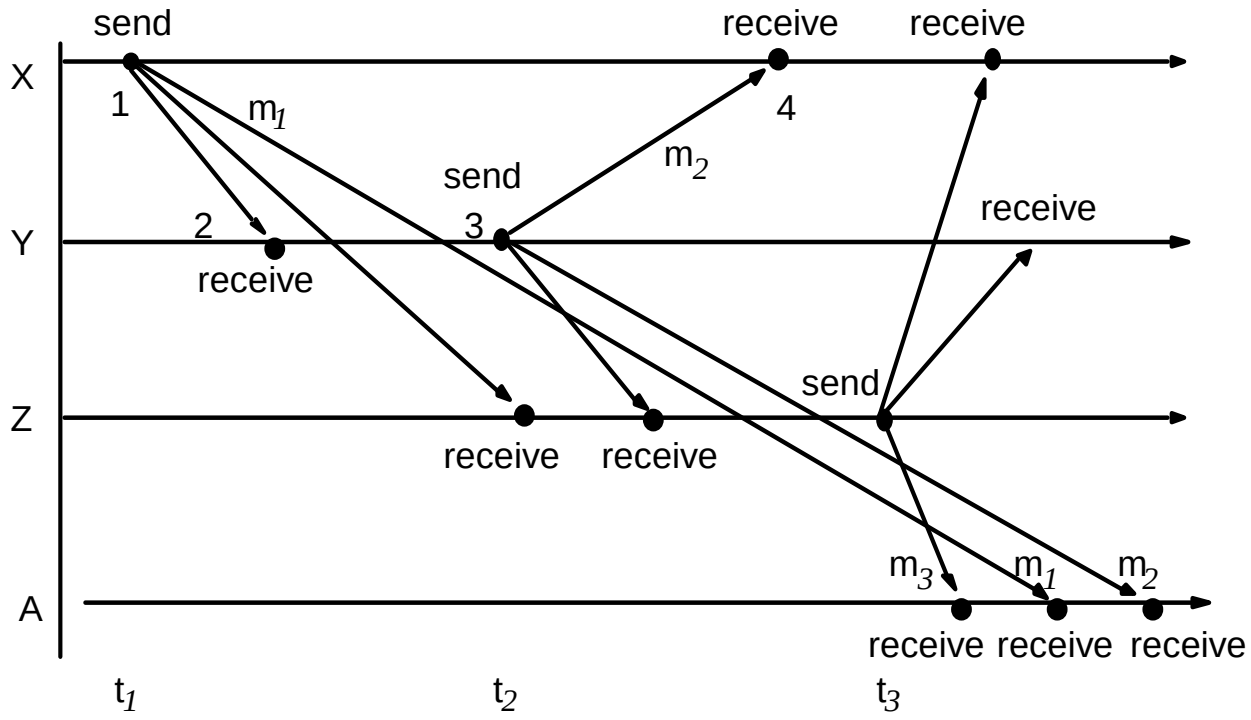
- Local clocks
- Clock drifts from perfect time and their drift rates differ from one another
- Clock Drift Rate: the relative amount that a computer clock differs from a perfect reference clock.
- Even if clocks are set at the same time initially, they would eventually vary unless corrections are applied periodically.

Interaction (Timing) Models

- **(Partially-Fully) Synchronous Systems**
  - There is a fixed upper bound Δ on the time for messages to be delivered (communication is synchronous).
  - There is a fixed upper bound Φ on the rate at which one processor's clock can run faster than another's (processors are synchronous).
  - If $\Phi=1$ and $\Delta=1$, we talk about a **fully synchronous** system. Then,
    - The sequence of alternating configurations and events can be partitioned into disjoint rounds.
    - A round consists of a deliver event for every message in an outbuf variable, until all outbuf variables are empty, followed by one computation event for every processor.
- **Asynchronous Systems**

  There is no fixed upper bound on how long it takes for a message to be delivered or how much time elapses between consecutive steps of a processor. Clock drift rates may be also arbitrary.

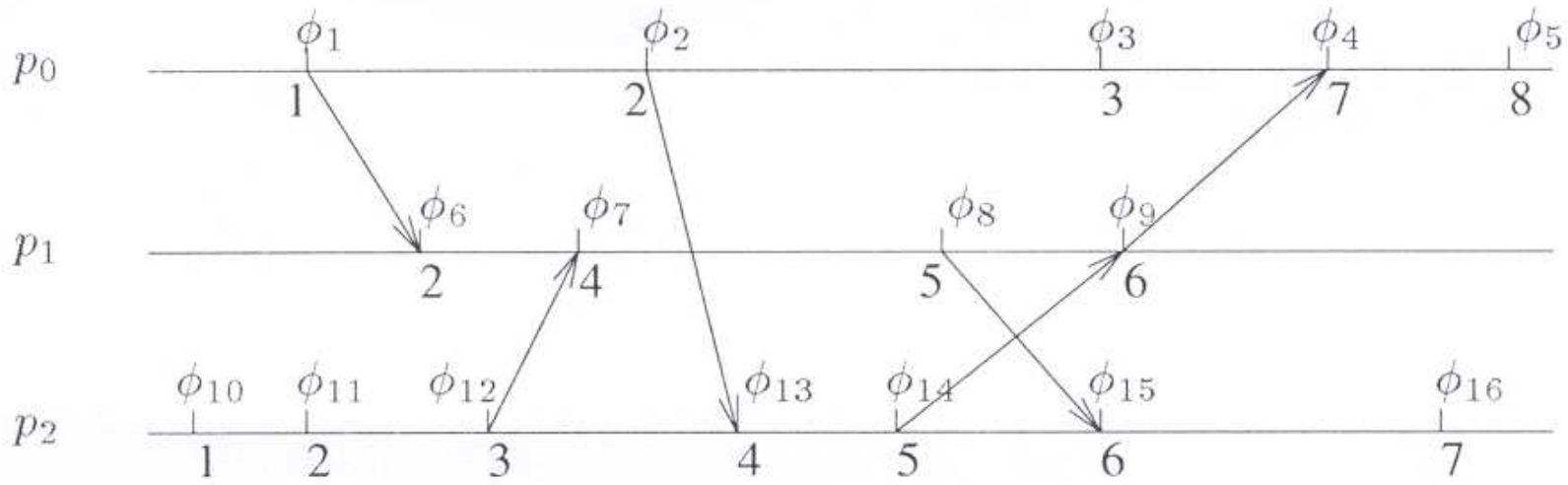
Real-Time Ordering of Events



A receives the messages in the wrong order!

- Relative order in which events take place in a system: there are relationships between events in distributed systems (causality)
- Logical clock, logical timestamp

Logical Clocks



Omission and arbitrary failures

Class of failure	*Affects*	*Description*
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*send*, 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.

Timing failures

Class of Failure	*Affects*	*Description*
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.

Masking Failures – Reliability of one-to-one Communication

- A service masks a failure, either by hiding it all together or by converting it into a more acceptable type of failure.
  - Checksums are used to mask corrupting messages -> a corrupted message is handled as a missing message
  - Message omission failures can be hidden by re-transmitting messages.
- The term reliable communication is defined in terms of validity and integrity as follows:
  - Validity: any message in the outgoing buffer is eventually delivered to the incoming message buffer
  - Integrity: the message received is identical to one sent, and no messages are delivered twice.