Organizational Communications and Distributed Object Technologies

Lecture 3: Models and Architectures

System Models

- **Architectural Models**

  Placement of parts and the relationships between them

  Examples: client-server model

  peer process model

- **Fundamental Models**

  Formal description of properties that are common in the architectural models focusing on the dependability characteristics (correctness, reliability, and security)

Main Architectural Models of DS

- Client processes
- Server processes
- Peer processes – cooperate and communicate in a symmetrical manner to perform a task

Software and hardware service layers in distributed systems

Applications, services

Middleware

Services such as communications, data sharing, naming,security, transactions, persistent storage and event notification

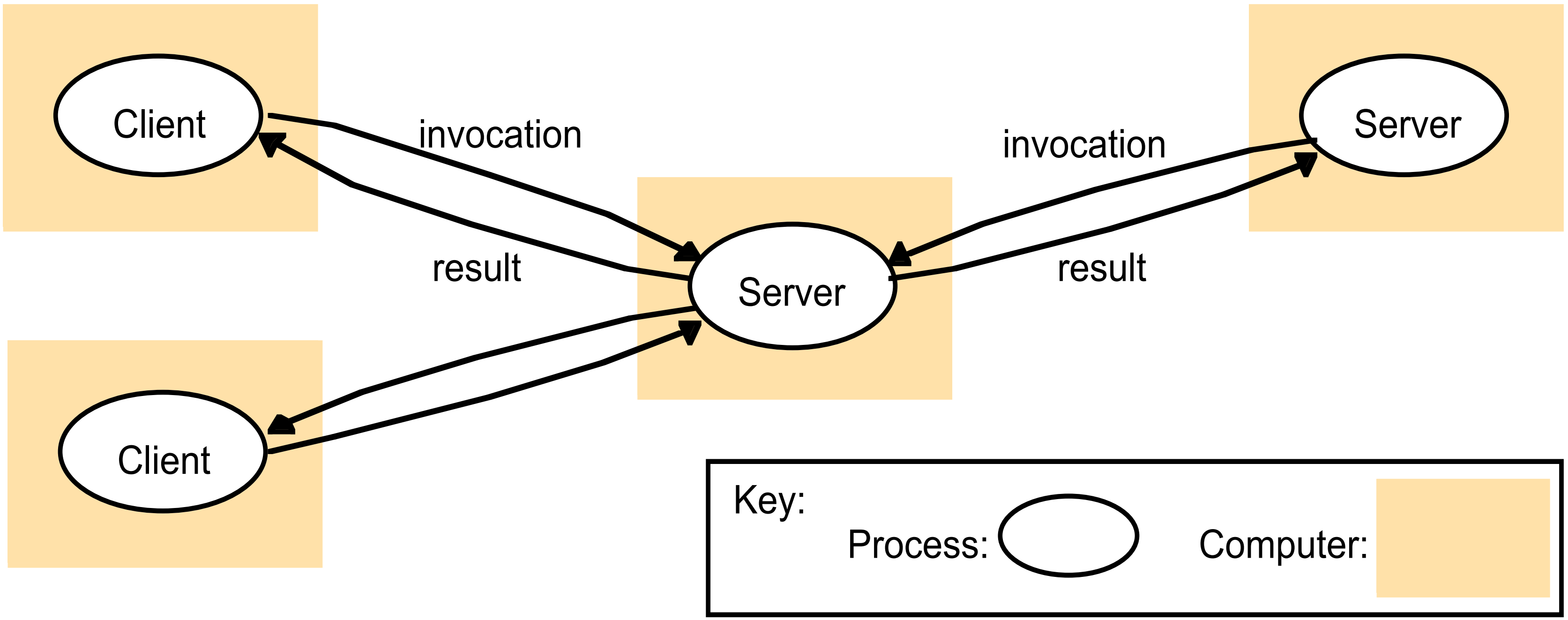
Mask heterogeneity and provides a convenient programming model to applications

Operating system

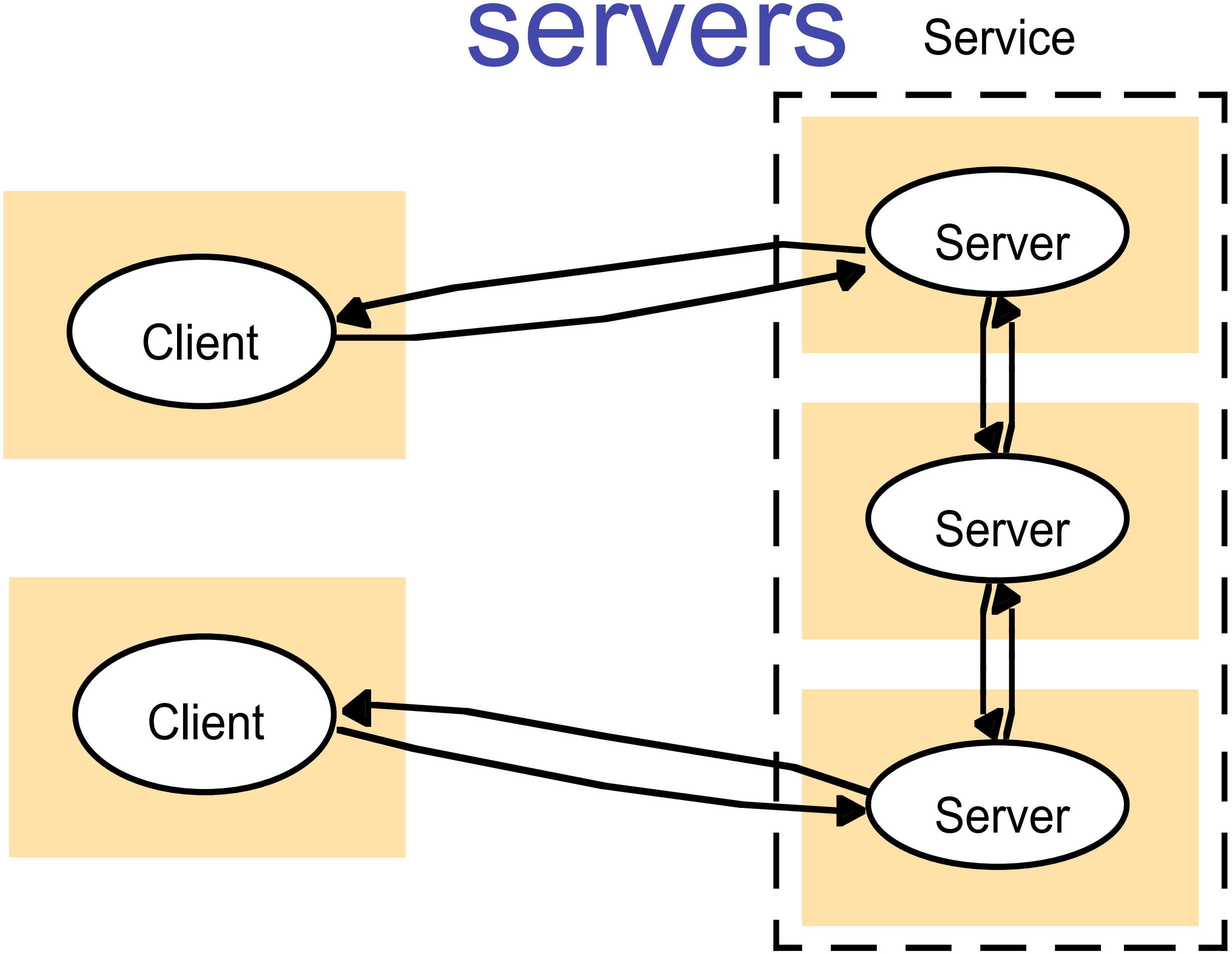
Computer and network hardware

Platform

Client-Server Model

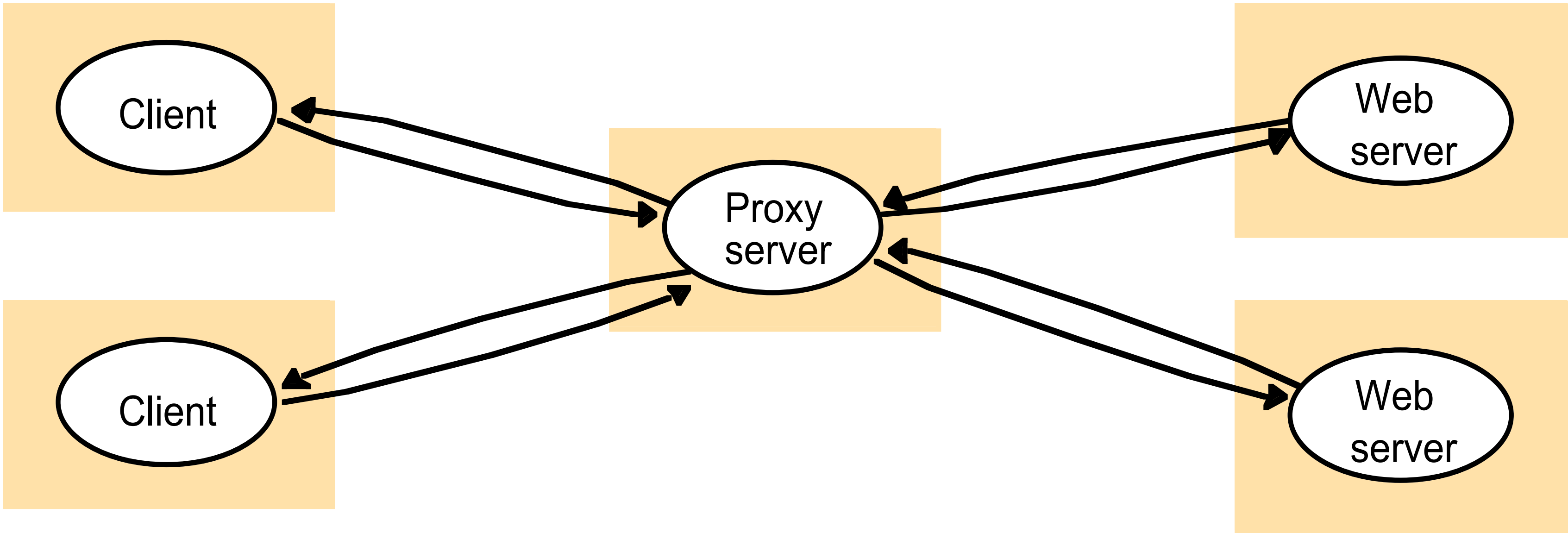


A service provided by multiple servers



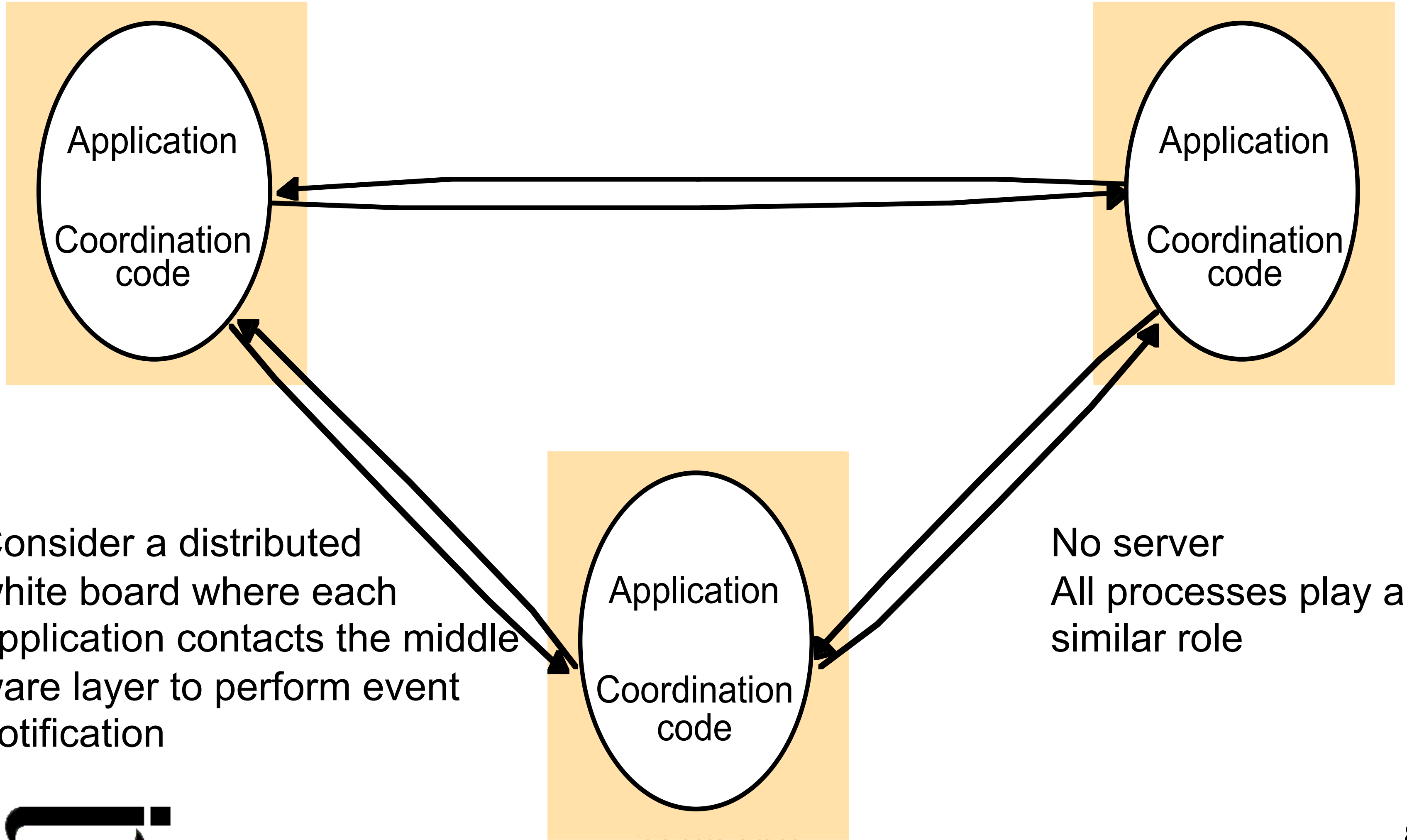
Replication is used to increase performance and availability and to improve fault tolerance

Proxy server and caches



A cache is a store of recently used data objects that is closer than the objects themselves. Caches may be collocated with each client or may be located in a proxy server that can be shared by several clients. This approach may increase availability and performance.

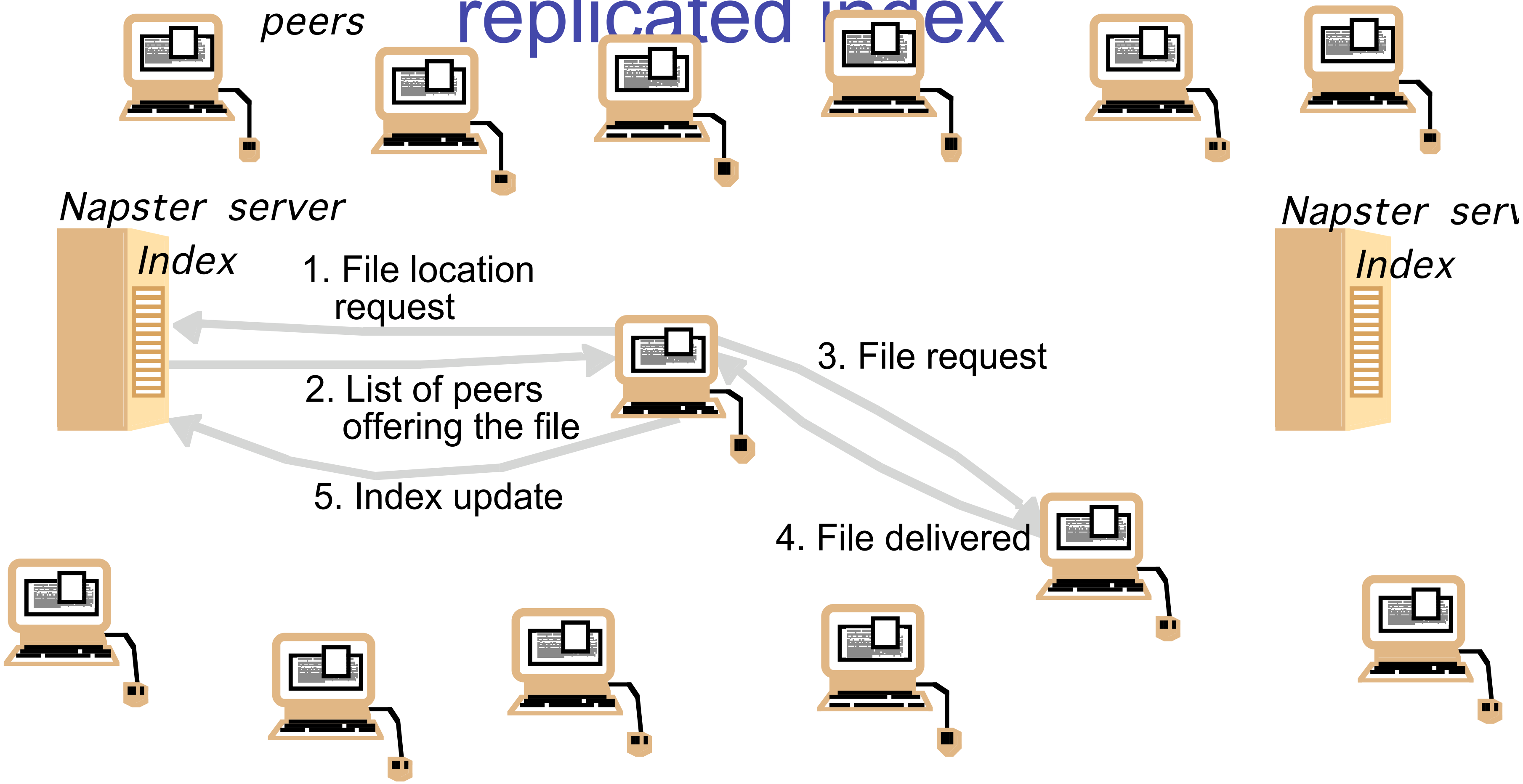
A distributed application based on peer processes



Consider a distributed white board where each application contacts the middle ware layer to perform event notification

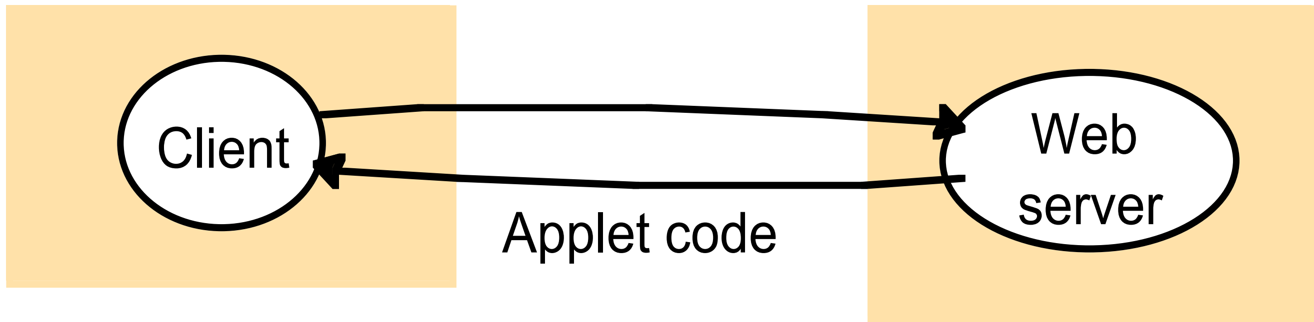
No server
All processes play a similar role

Napster: peer-to-peer file sharing with a centralized, replicated index



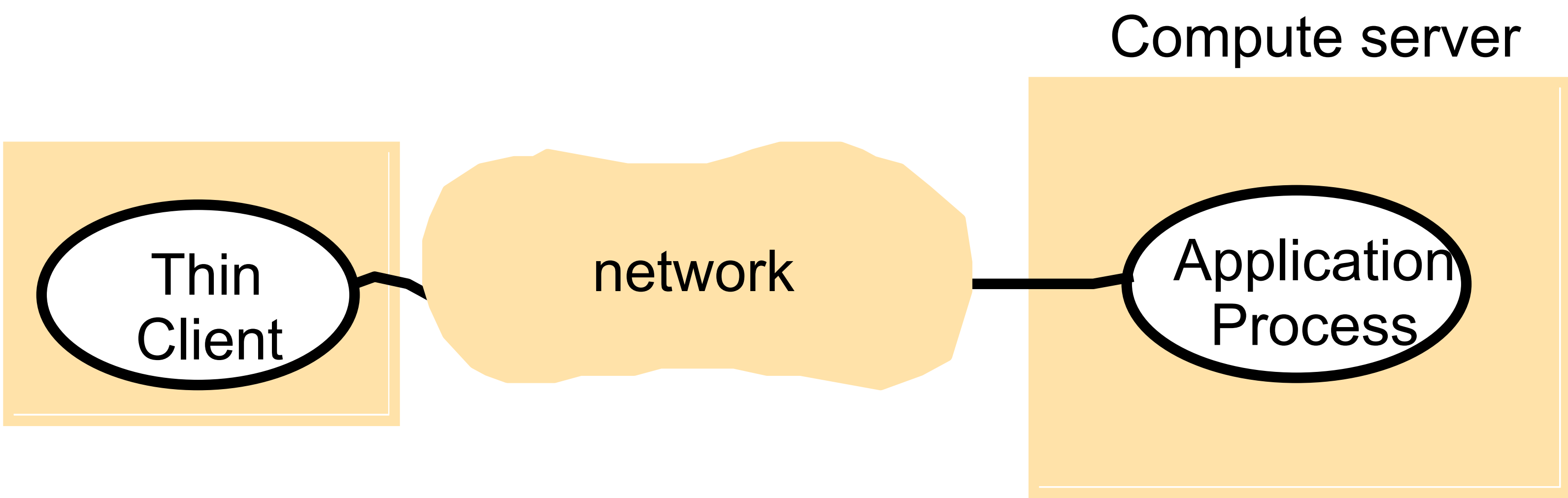
Variations on client-server (mobile code)

a) client request results in the downloading of applet code



b) client interacts with the applet

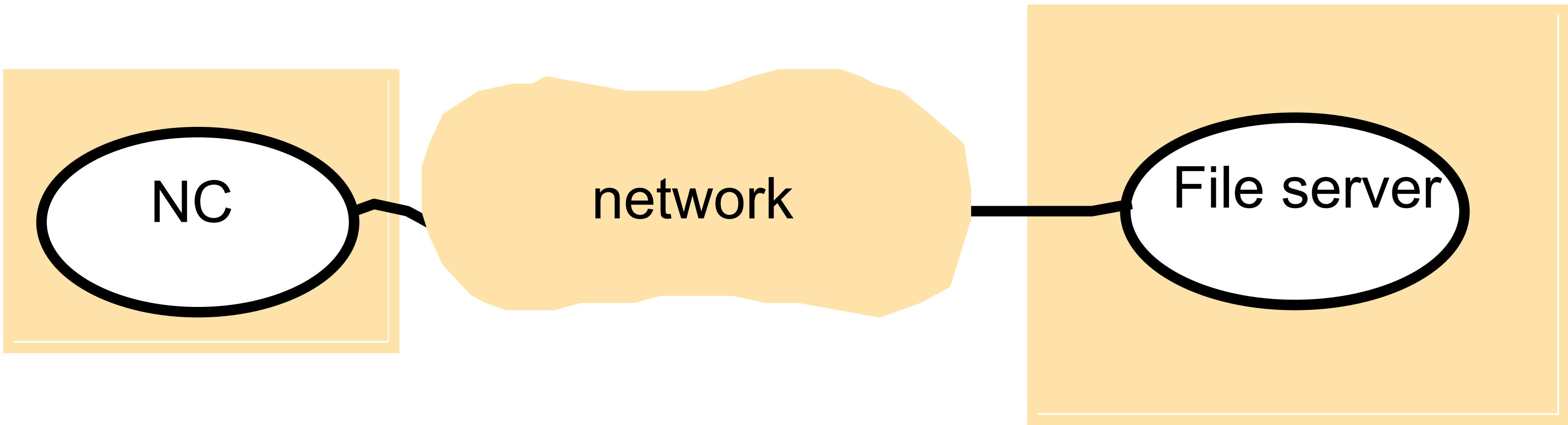
Thin Clients



The client does graphics only.
Think “Old Text terminal” with a GUI.
Applications run on the sever.

Examples include:
Unix X-11 display protocol
Citrix and Microsoft put
Microsoft apps
in a mainframe model

Network Computer



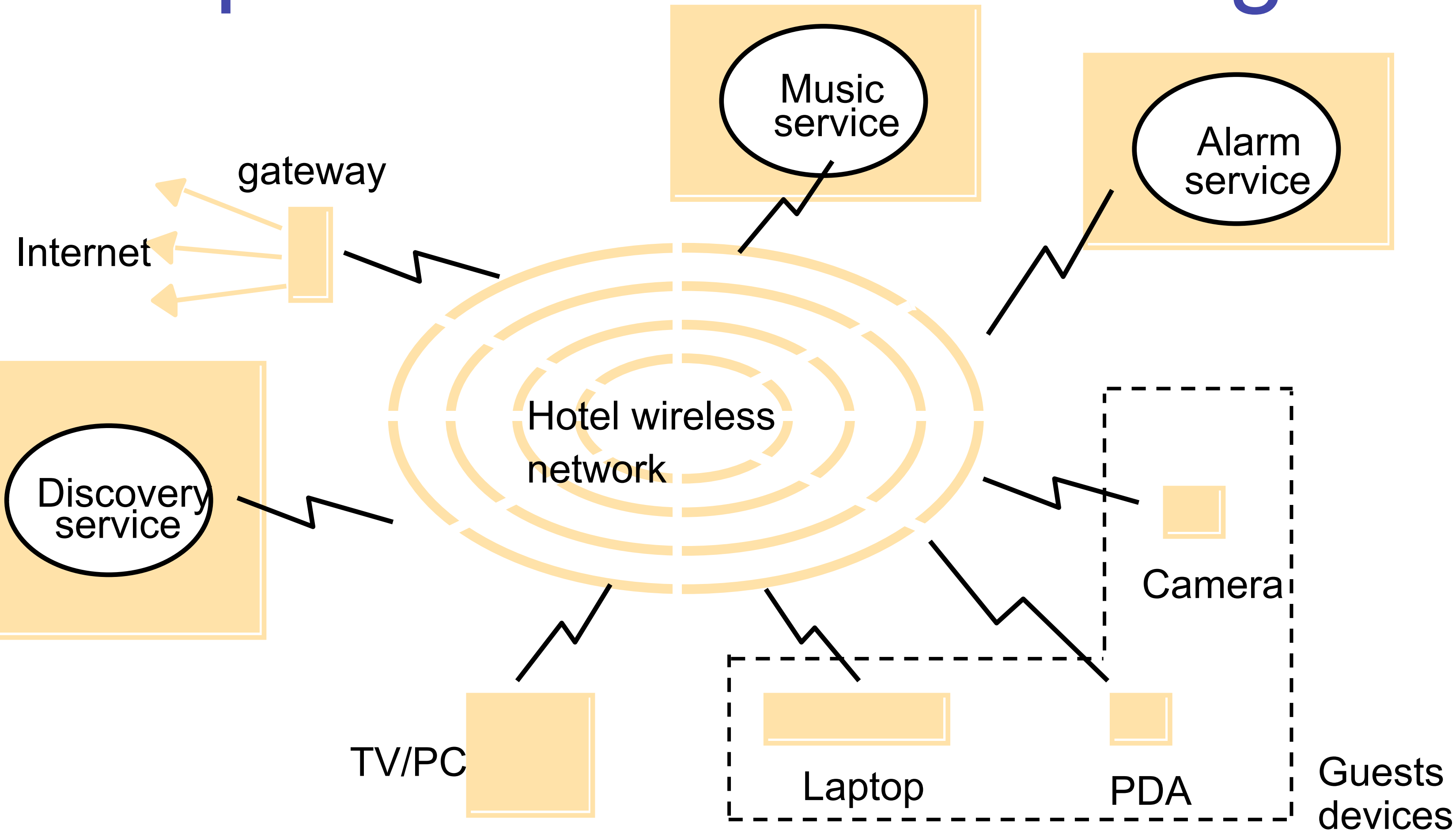
The OS and Applications
are downloaded to the client.
No disk (or very little) is included.
A disk would act as a cache.

Oracle and Sun were once
behind this approach.
Remember the Javastation?

Mobile Agents

- Includes both code and data.
- Copies itself from one machine to another.
- Collects information and returns.
- The Xerox PARC worm carried out useful work on idle processors. 1982
- Big security concerns but doable.
- The disk with the Robert Morris Internet worm is displayed at the Boston Museum of Science (under glass!)

Mobile devices and Spontaneous networking



Fundamental Models

All of the above systems share some fundamental properties:

Interaction Model

communication takes place with
delays and without a universal clock

Failure Model

correct operation of DS is threatened by
various types of faults

Security Model

the security model defines and classifies forms
of attack

Interaction Model

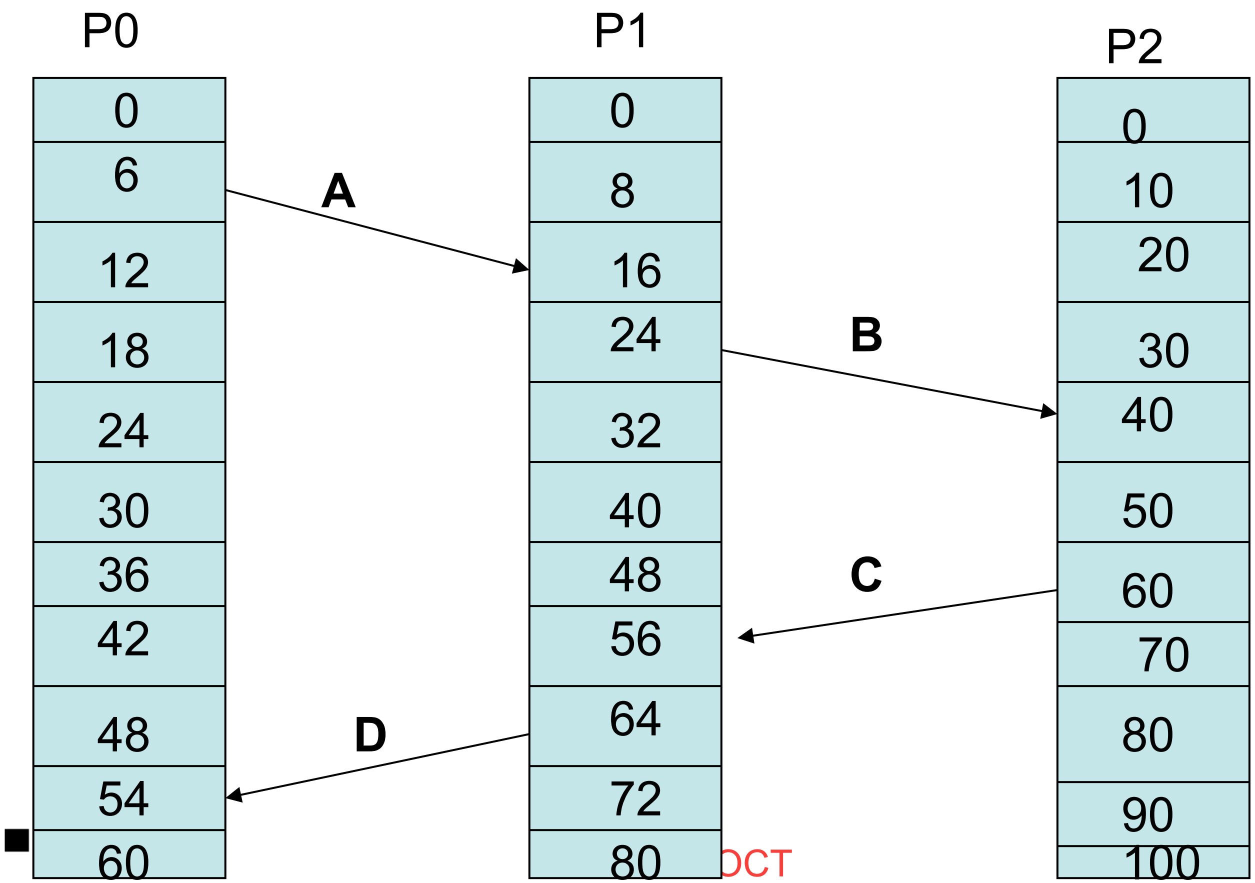
- Two Variants:

  Synchronous Distributed System
  The model assumes we can place bounds on time needed for process execution, communication, and clock drift.

  Asynchronous Distributed Systems
  assumes we can set no bounds on any of the above (some design problems can be solved even with these assumptions)
  The internet fits this model well.

Lamport Clocks (1)

Three processes, each with its own clock.



Lamport Clocks (2)

Lamport defined the “happens-before” relation. The expression a->b is read “a happens before b”.

If a and b are two events within the same process and a occurs before b then a -> b is true.

If a is the sending of a message by one process and b is the reception of that message by another then a -> b is true.

“Happens-before” is transitive.
If a->b and b->c then a->c.

Lamport Clocks (3)

We need a way of assigning time values so that all processes (P) agree that:

If a->b then C(a) < C(b) with C always increasing never decreasing.

Lamport Clocks (4)

Lamport's Algorithm

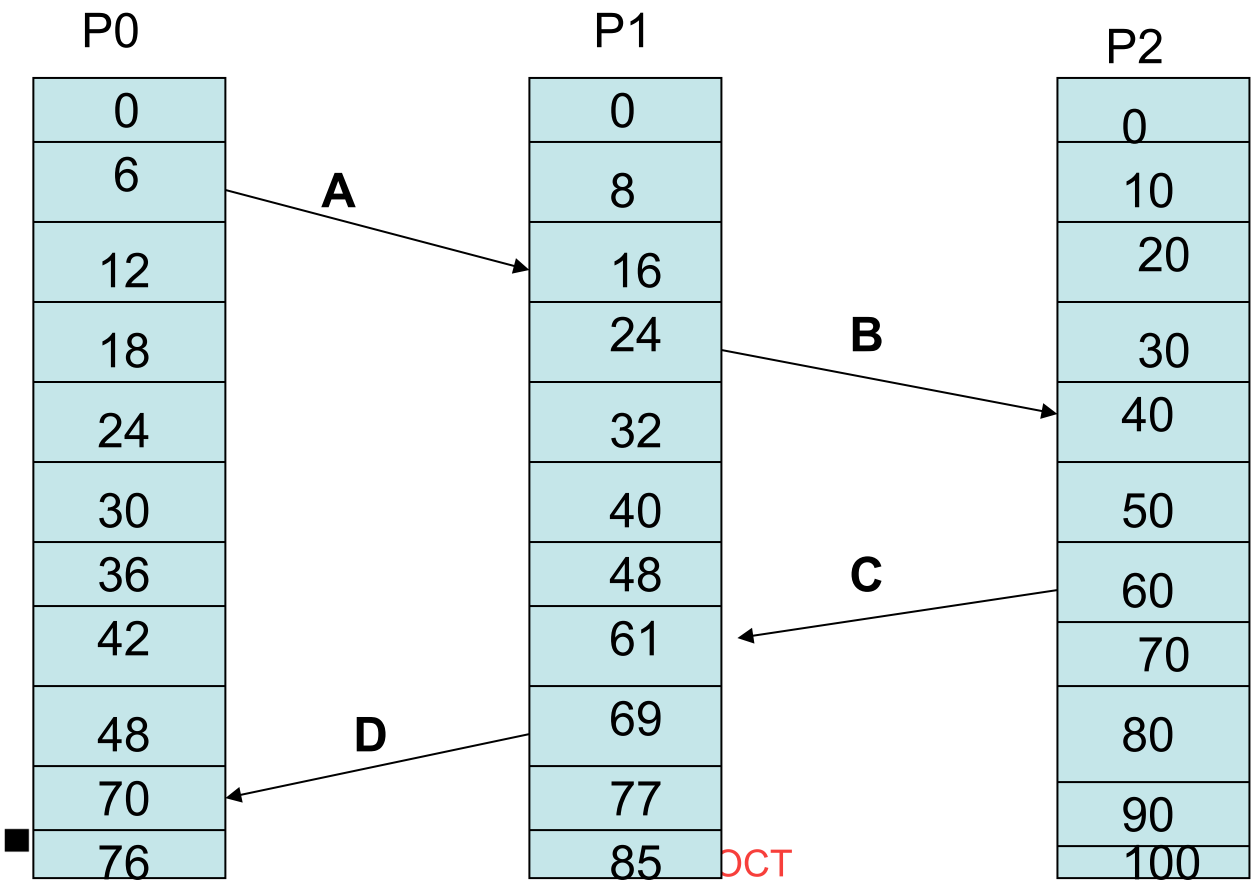
LC1: C(i) is incremented before each
event happens at P(i)

LC2: (a) When a process P(i) send a message
m, it piggybacks on m the value t = C(i)
(b) On receiving (m,t), a process P(j)
computes C(j) = max(C(j), t) and then applies
LC1 before timestamping the event receive(m)

So, if the "happens-before" relation holds between a and b then C(a) < C(b)

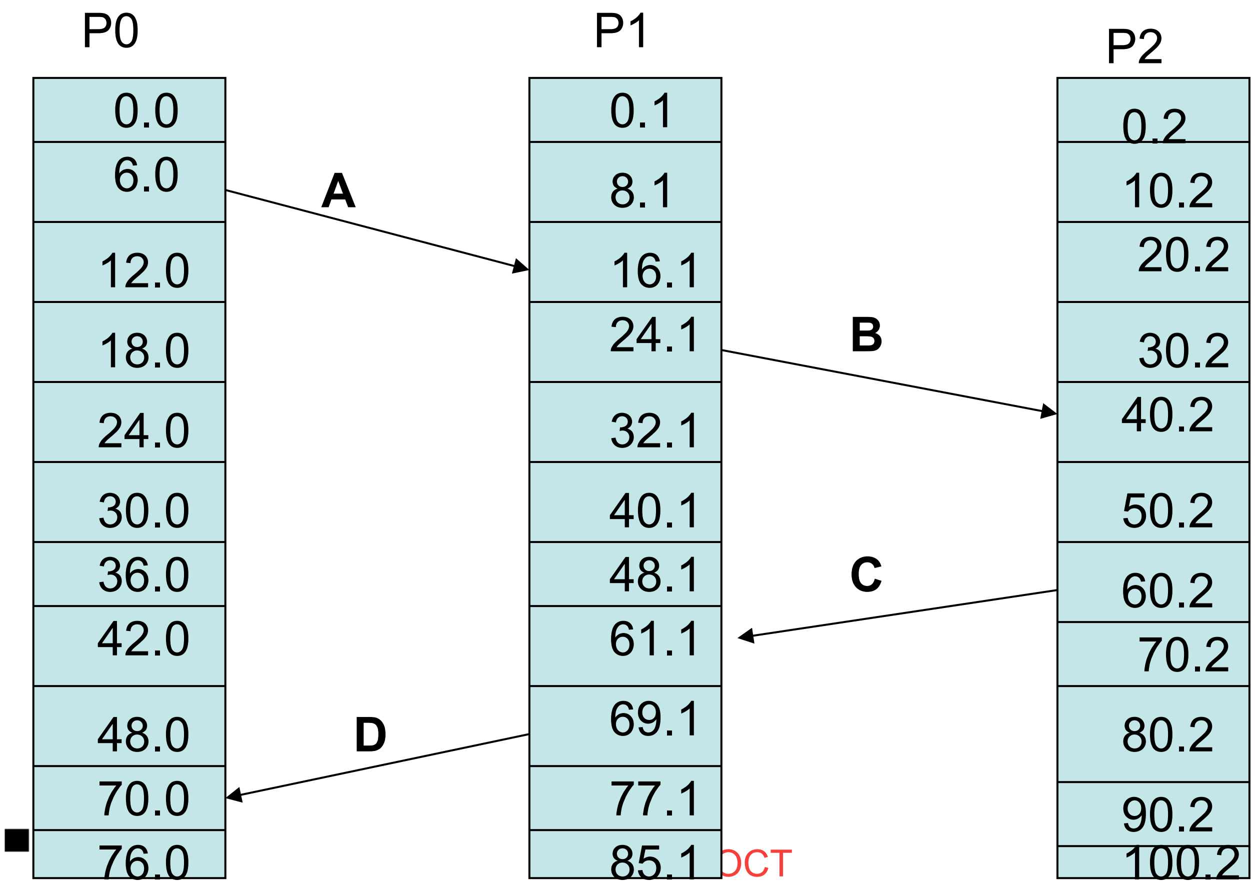
Lamport Clocks (5)

Three processes, using Lamport's Algorithm.

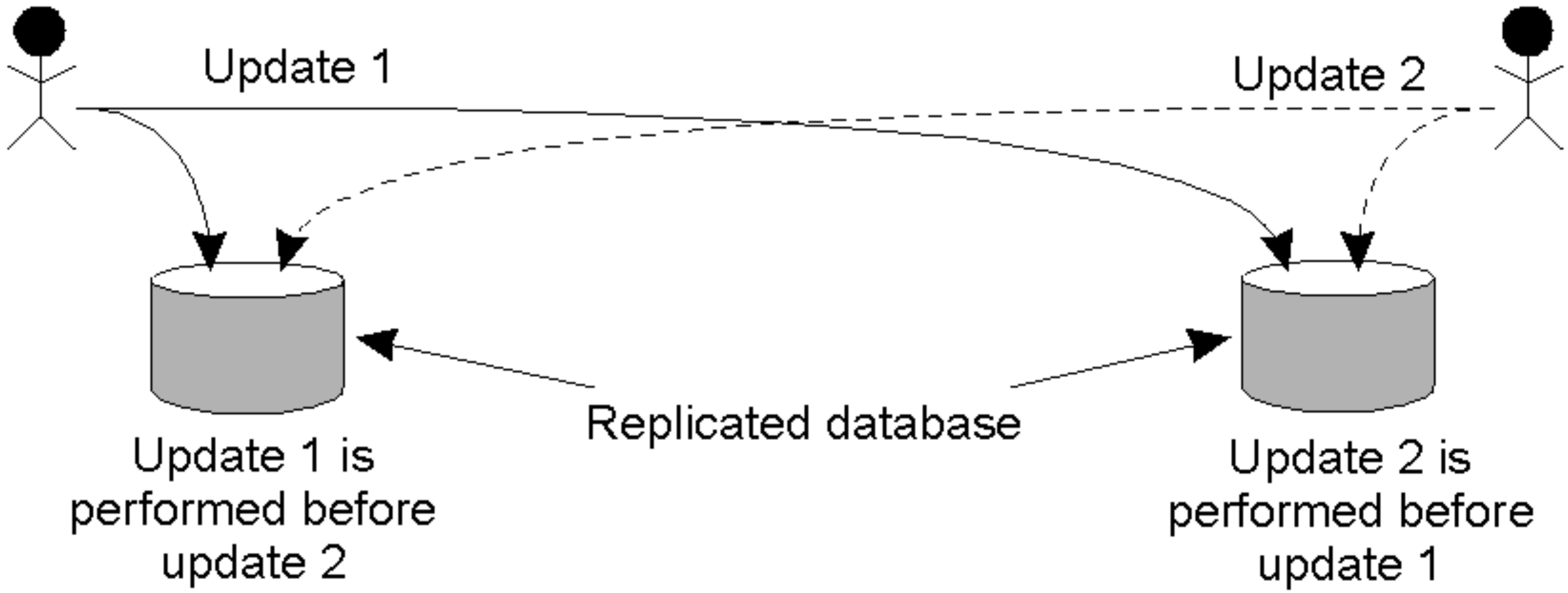


Lamport Clocks (6)

Adding the process number to events



Lamport Timestamps(7)



Update 1 : San Fransisco wants to add $100 to account containing $1000.
Update 2: New York wants to add 1% interest to the same account.
Result: San Fransisco Data Base holds $1,111
             New York Data Base holds $1,110

Totally-ordered multicast

- Assume that messages from the same sender are received in the order that they were sent. Assume all messages arrive and are muticast to <u>every</u> process.
- When a process receives a message it puts it into a local queue ordered according to the timestamp in the message (every message has a unique timestamp)
- The receiver <u>multicasts</u> an acknowledgement.
- Note that the timestamp on the received message will always be lower than the timestamp on the acknowledgement.

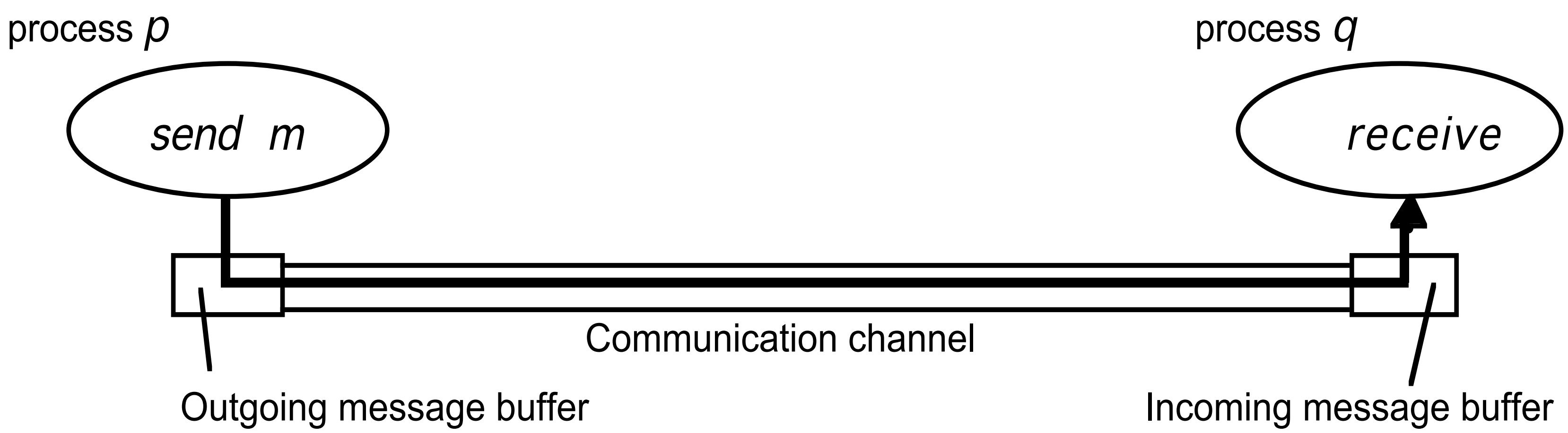
Totally-ordered multicast

- All processes will eventually have the same copy of the local queue.
- A process can deliver a message to the application it is running only when that message is at the head of the queue and has been acknowledged by each other process. At that point the message and its acknowledgements are removed
- All messages are delivered in the same order everywhere

Failure Model

- Processes may fail
- Communication channels may fail
- The failure model describes various types of failures

Processes and channels



Process omission failures
Communication omission failures
Arbitrary failures
Timing failures in synchronized systems

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.

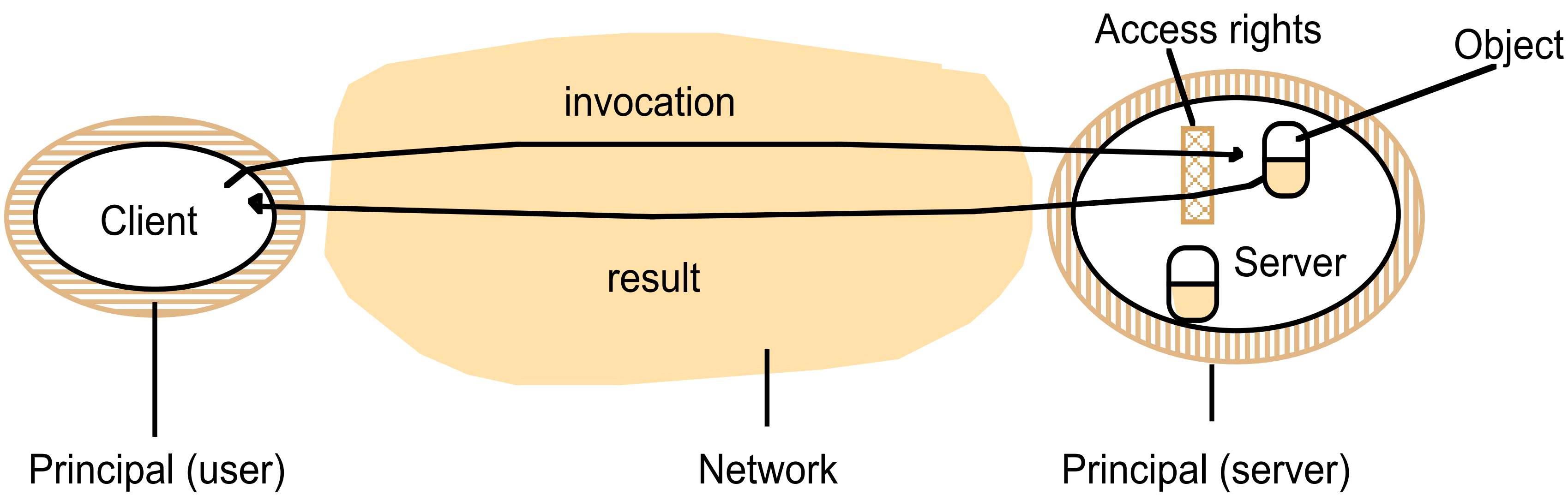
Synchronous Model Timing Failures

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.

Security Model

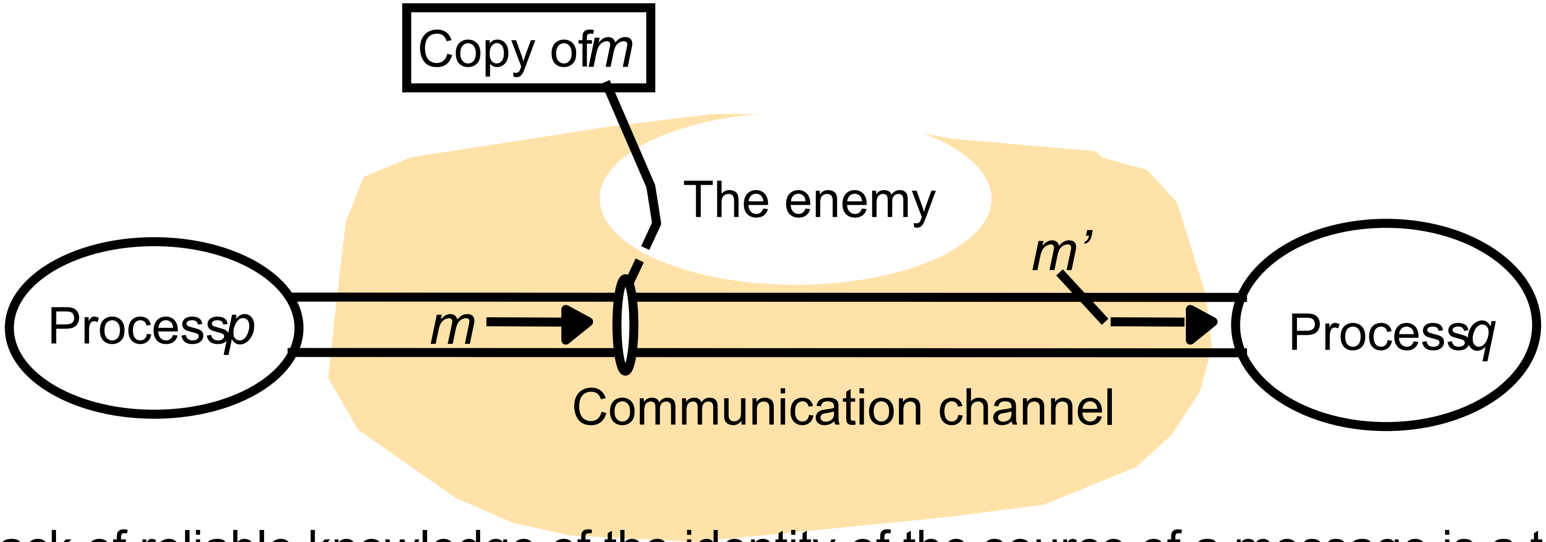
- Secure the processes
- Secure the channels
- Protect encapsulated objects from unauthorized access

Objects and principals



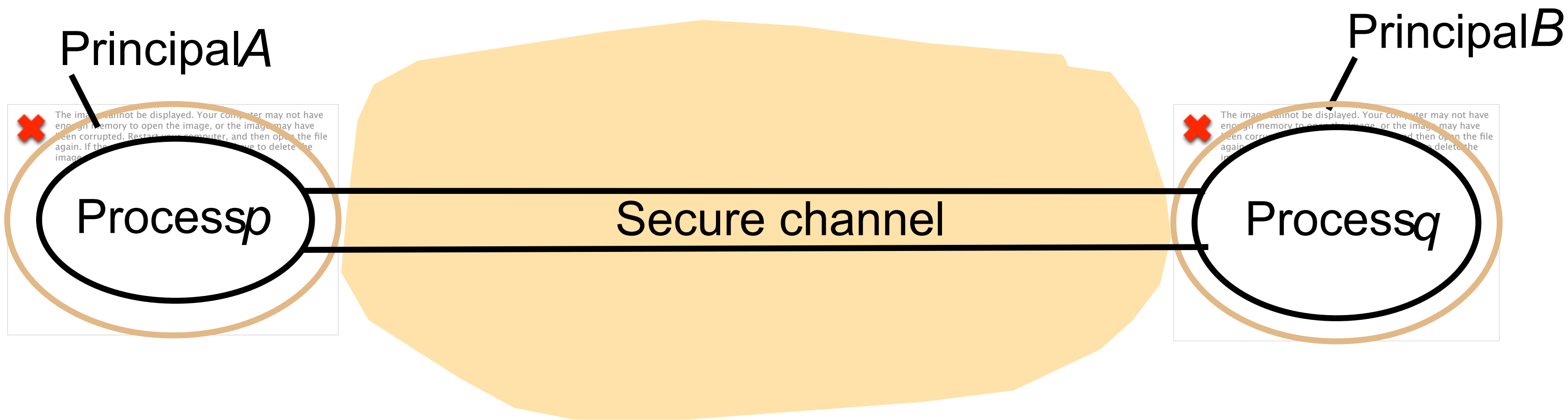
Access rights specify who is allowed to read or write the object's state.
Associated with each invocation and each result is an authority on which it is issued. Such an authority is called a *principal*. A principal may be a user or process. Both the server and the client may check the identity of the principal behind each invocation or result to determine if access rights have been granted..

The enemy



Lack of reliable knowledge of the identity of the source of a message is a threat to clients and servers.
An enemy can copy, alter or inject messages as they travel across the network.
An enemy can replay old messages.

Secure channels



Cryptography can be used for:

- Encryption
- Authentication