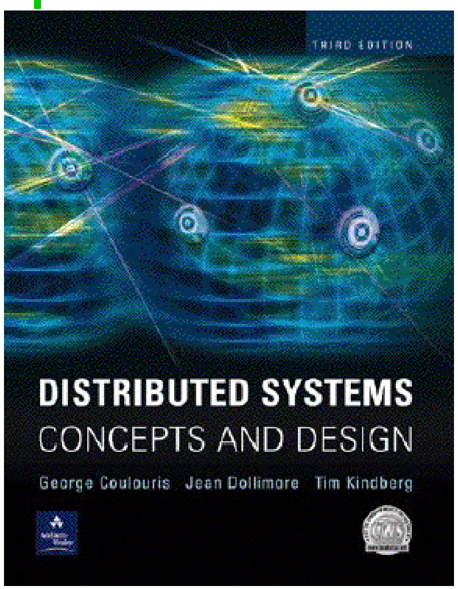
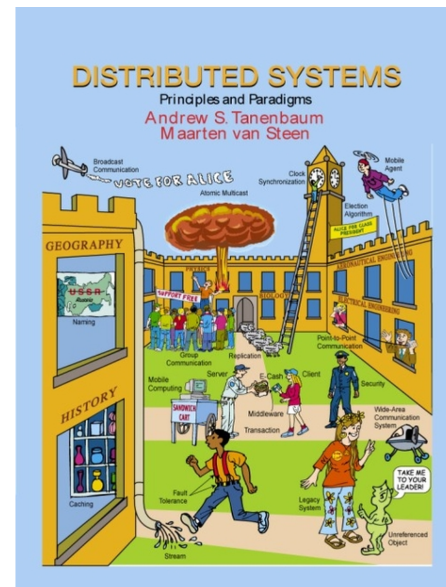
Architectural Models

Dr. Gannouni Sofien



Most concepts are drawn from Chapter 2
© Pearson Education



Some ideas from Chapter 1
© Pearson Education

Presentation Outline

- Introduction
- Architectural Models
  - Software Layers
  - System Architectures
    - Client-Server
      - Clients and a Single Sever, Multiple Servers, Proxy Servers with Cachers, Peer Model
    - Alternative Client-Sever models driven by:
      - Mobile code, mobile agents, network computers, thin clients, mobile devices and spontaneous networking
    - Design Challenges/Requirements
- Fundamental Models – formal description
  - Interaction, failure, and Security models.
- Summary

Lecture Overview (I)

- An Architectural model of a distributed system is concerned with the placement of its parts and relationship between them. Examples:
  - Client-Server (CS) and peer process models.
  - CS can be modified by:
    - The partitioning of data/replication at cooperative servers
    - The caching of data by proxy servers or clients
    - The use of mobile code and mobile agents
    - The requirements to add or remove mobile devices.

Lecture Overview (II)

- Fundamental Models are concerned with a formal description of the properties that are common in all of the architectural models.
- Models addressing time synchronization, message delays, failures, security issues are addressed are:
  - Interaction Model – deals with performance and the difficulty of setting of time limits in a distributed system.
  - Failure Model – specification of the faults that can be exhibited by processes
  - Secure Model – discusses possible threats to processes and communication channels.

Architectural Models

Software Layers
System Architectures
Interfaces and Objects
Design Requirements

Architectural Models – Intro [1]

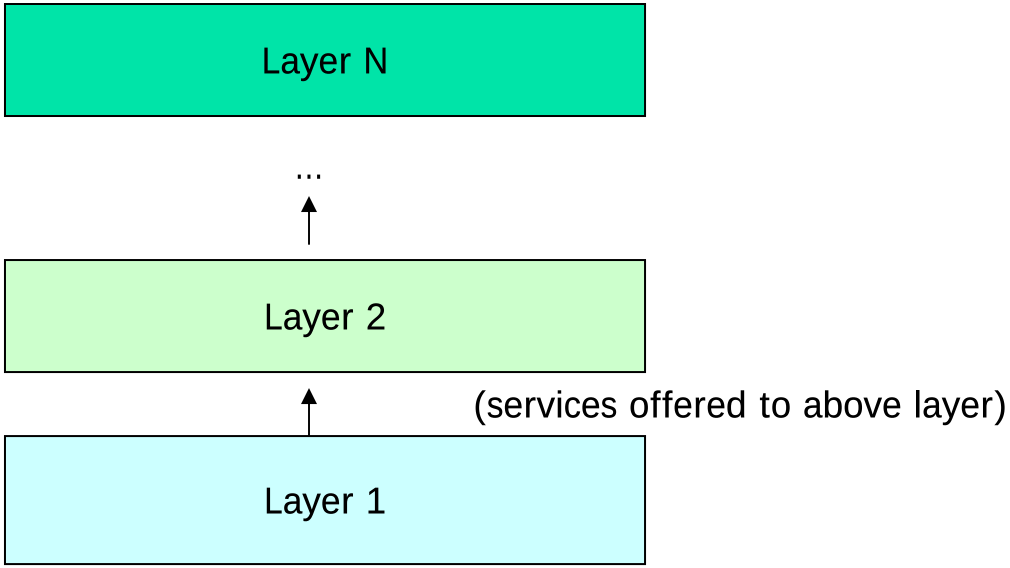
- The architecture of a system is its structure in terms of separately specified components.
  - Its goal is to meet present and likely future demands.
  - Major concerns are make the system reliable, manageable, adaptable, and cost-effective.
- Architectural Model:
  - Simplifies and abstracts the functions of individual components
  - The placement of the components across a network of computers – define patterns for the distribution of data and workloads
  - The interrelationship between the components – ie., functional roles and the patterns of communication between them.

Architectural Models – Intro [2]

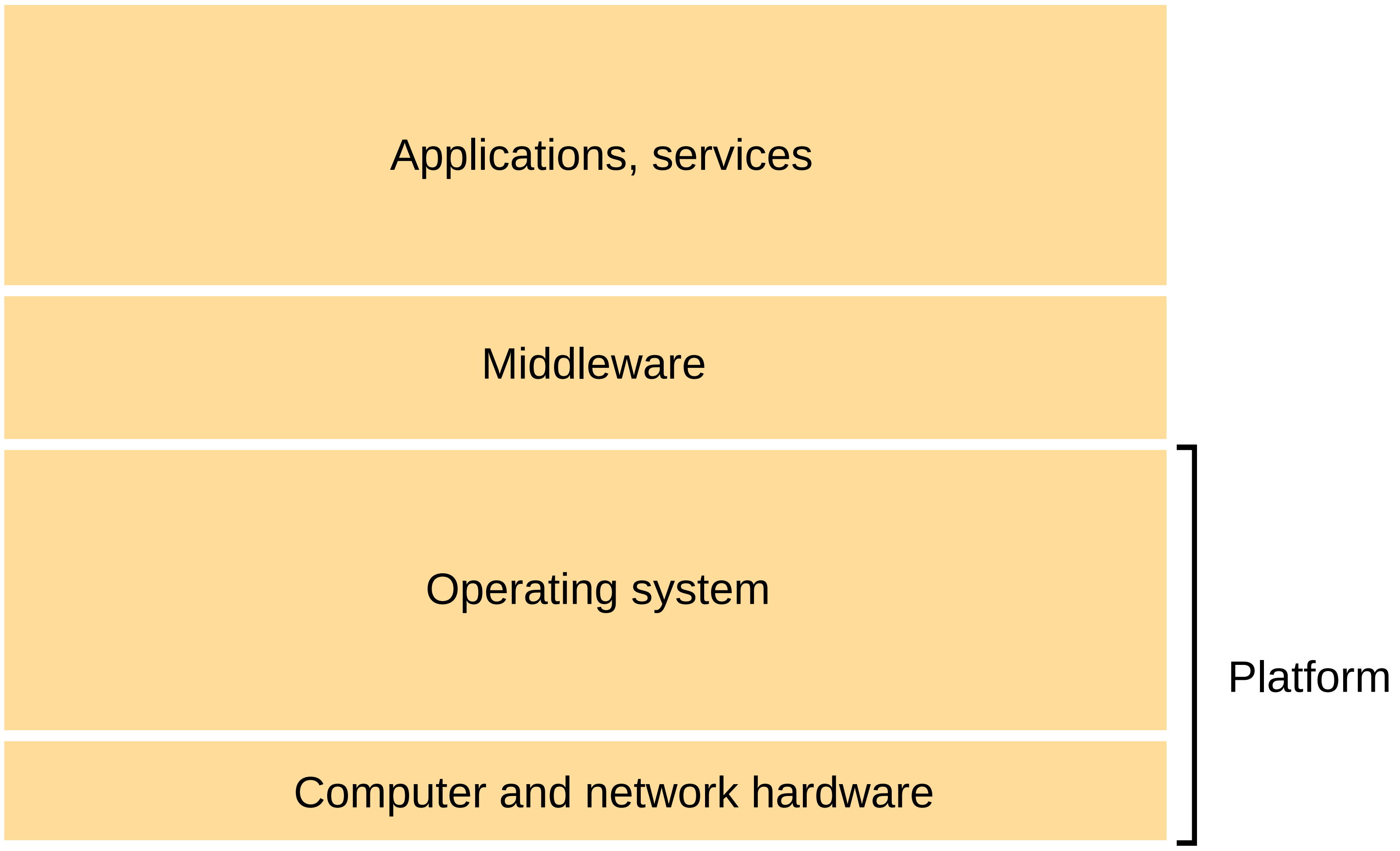
- Architectural Model - simplifies and abstracts the functions of individual components:
  - An initial simplification is achieved by classifying processes as:
    - Server processes
    - Client processes
    - Peer processes
      - Cooperate and communicate in a symmetric manner to perform a task.

Software Architecture and Layers

- The term *software architecture* refereed:
  - Originally to the structure of software as *layers* or modules in a single computer.
  - More recently in terms of *services* offered and requested between processes in the same or different computers.
- Breaking up the complexity of systems by designing them through layers and services
  - Layer: a group of related functional components
  - Service: functionality provided to the next layer.



Software and hardware service layers in distributed systems



Platform

- The lowest hardware and software layers are often referred to as a platform for distributed systems and applications.
- These low-level layers provide services to the layers above them, which are implemented independently in each computer.
- Major Examples
    - Intel x86/Windows
    - Intel x86/Linux
    - Intel x86/Solaris
    - SPARC/SunOS
    - PowePC/MacOS

Middleware

- A layer of software whose purpose is to mask heterogeneity present in distributed systems and to provide a convenient programming model to application developers.
- Major Examples:
    - Sun RPC (Remote Procedure Calls)
    - OMG CORBA (Common Request Broker Architecture)
    - Microsoft D-COM (Distributed Components Object Model)
    - Sun Java RMI
    - Modern Middleware:
        - Melbourne Gridbus – for Grid computing
        - ANL Globus – a Grid toolkit
        - IBM WebSphere
        - Microsoft .NET
        - Sun J2EE

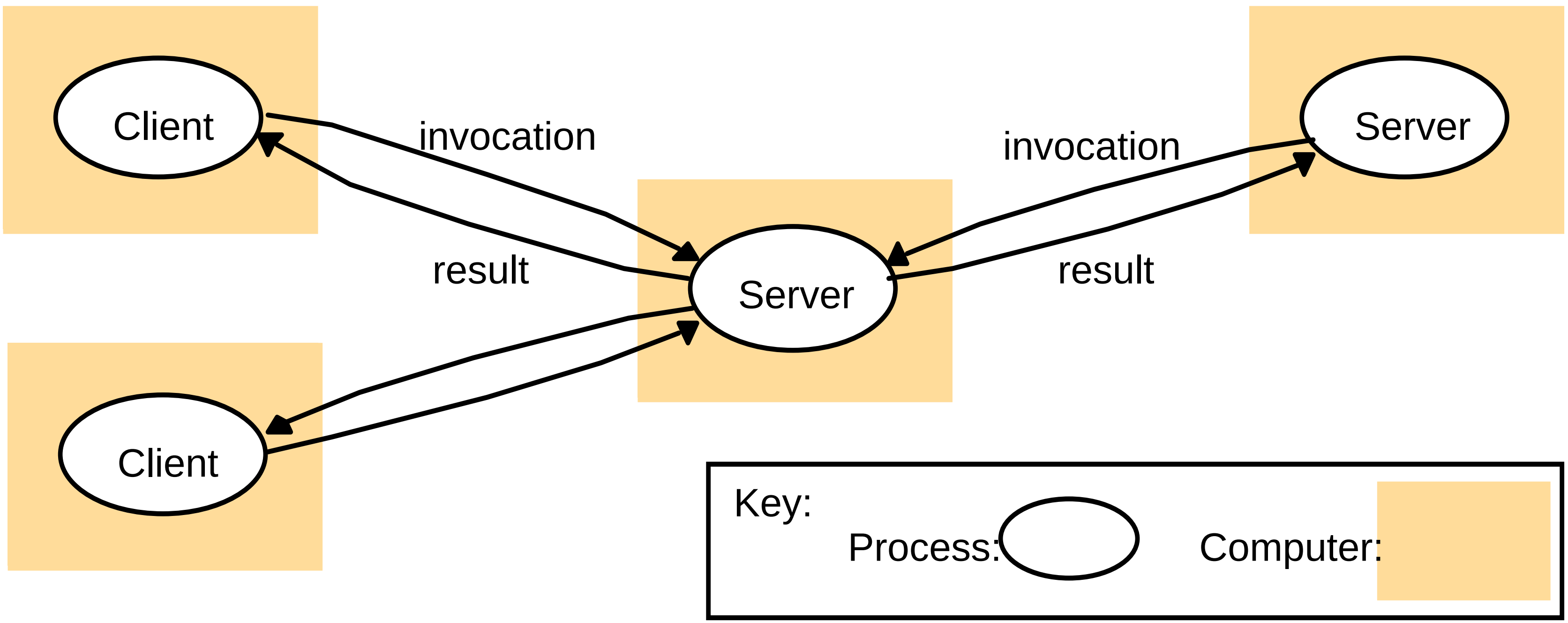
System Architecture

- The most evident aspect of DS design is the division of responsibilities between system components (applications, servers, and other processes) and the placement of the components on computers in the network.
- It has major implication for:
  - Performance, reliability, and security of the resulting system.

System Architecture Models

- Client / Server Architecture Models
- Meta-Computing model
- Global Computing Model
- Grid Computing
- P2P Model

Client Server Basic Model: Clients invoke individual servers



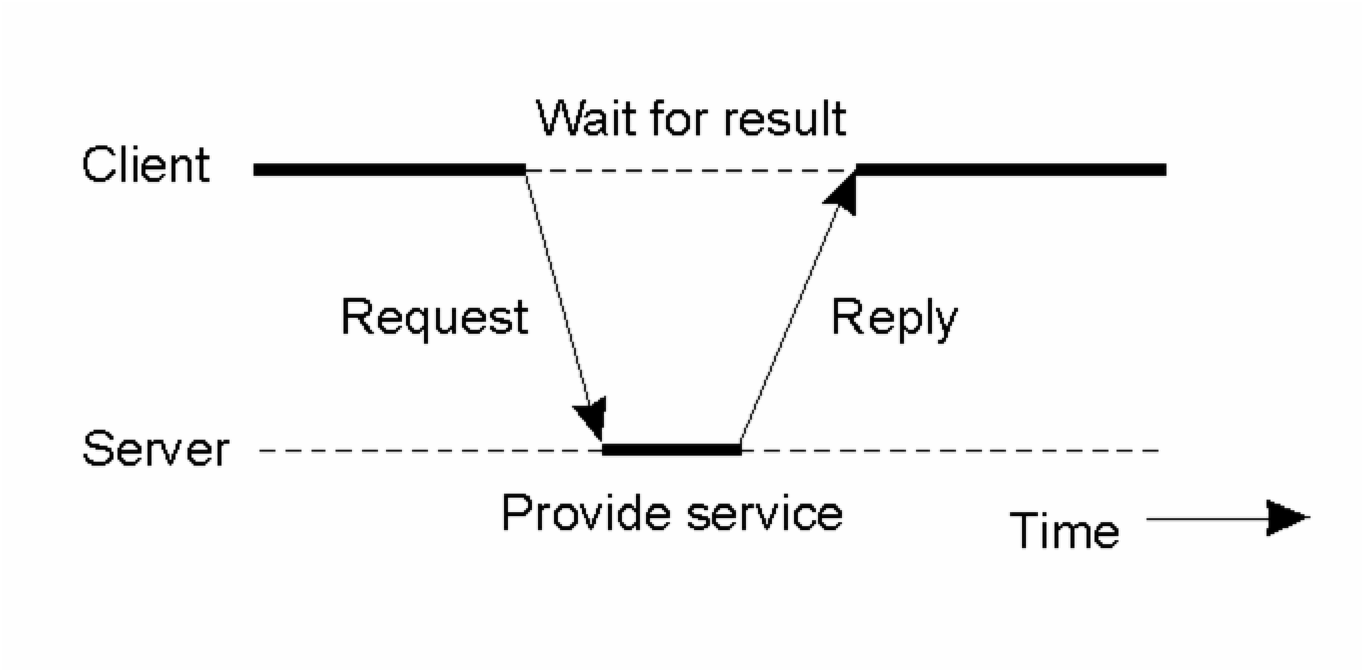
- Client process interact with individual server processes in a separate computer in order to access data or resource. The server in turn may use services of other servers.
- Example:
  - A Web Server is often a client of file server.
  - Browser → search engine -> crawlers → other web servers.

Clients and Servers

- The Server is a process that perform tasks **on demand** by **exchanging messages** with the clients.
- The Server is a resource manager and as such they provide:
  - ***Encapsulation***:
    - Provide a **useful service interface** (a set of operations) to the resources **that meet the client's needs**
  - ***Concurrent processing***:
    - Achieving **concurrency transparency**.
  - ***Protection***:
    - Resources require protection **from illegitimate accesses**.
    - Read, Write and Execution **Permissions**.

Clients and Servers

- General interaction between a client and a server.



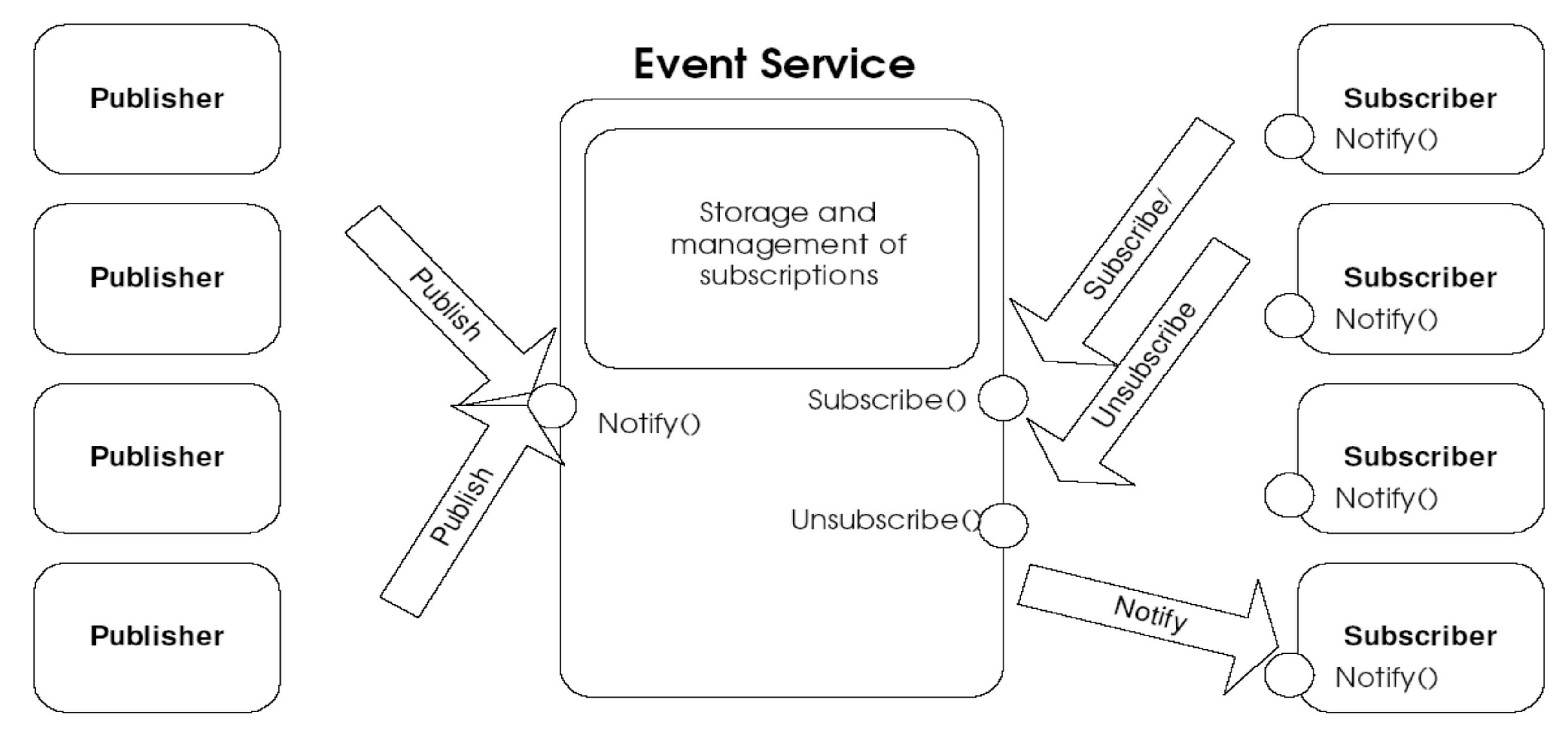
Client/Server Communications: Interaction schemes

- A client/server system relies on communication, generally involving:
  - Publisher/Subscriber
  - Message passing
  - RPC/RMI
  - Shared spaces
  - Message queuing

Publish/Subscribe Paradigm

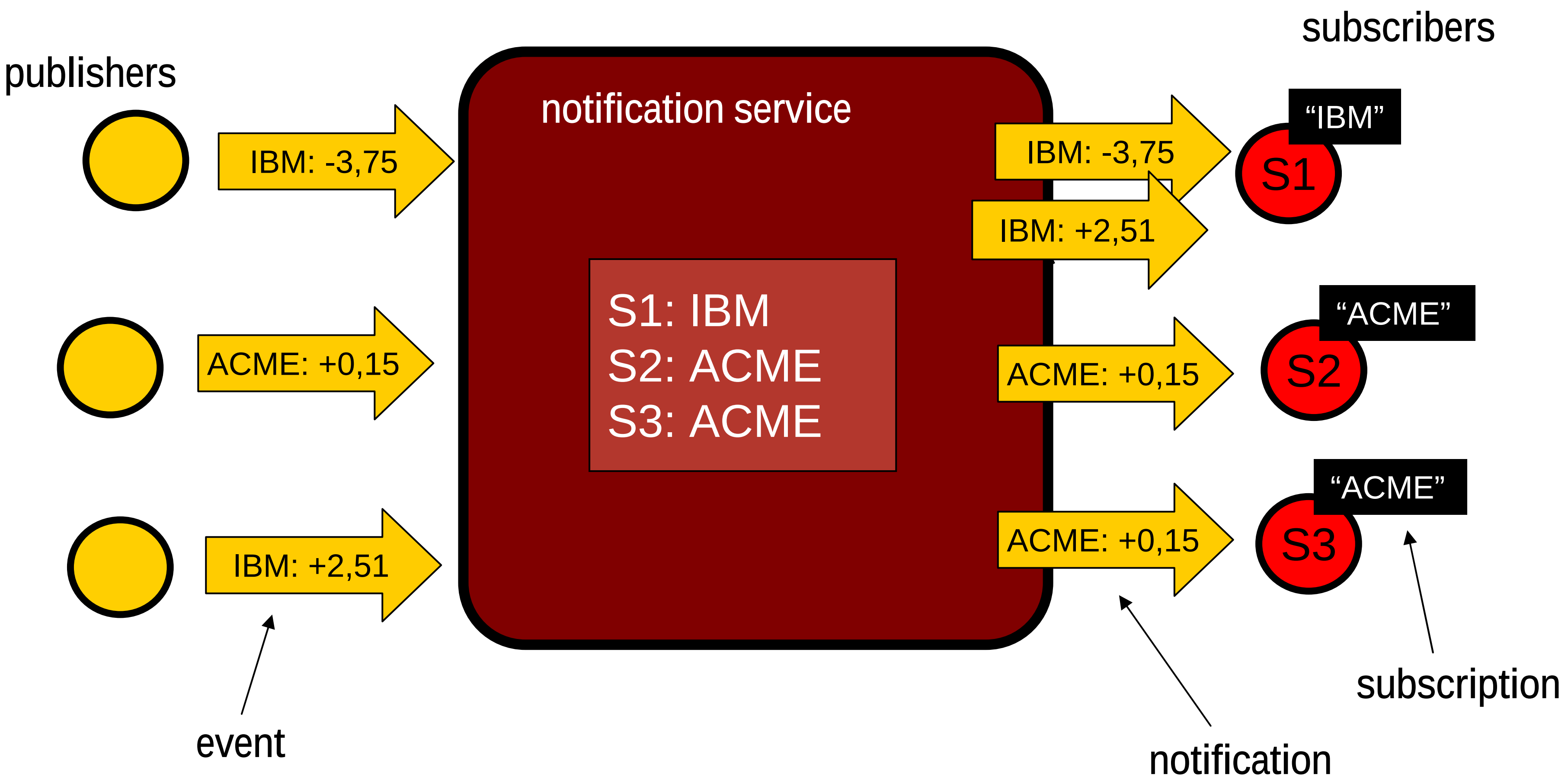
- Publish/Subscribe (pub/sub): a powerful abstraction for building distributed applications
  - Message-based, anonymous communication
  - Participants are decoupled
- Good solution for highly dynamic, decentralized systems (e.g., wired environments with huge numbers of publishers and subscribers, Mobile networks, P2P etc)
- Many research issues, involving several research areas (e.g., systems, software eng., databases etc)

Publish/Subscribe Interactions



- Eugster, P.T., et al., **The Many Faces of Publish/Subscribe**. ACM Computing Surveys, 2003. 35(2).

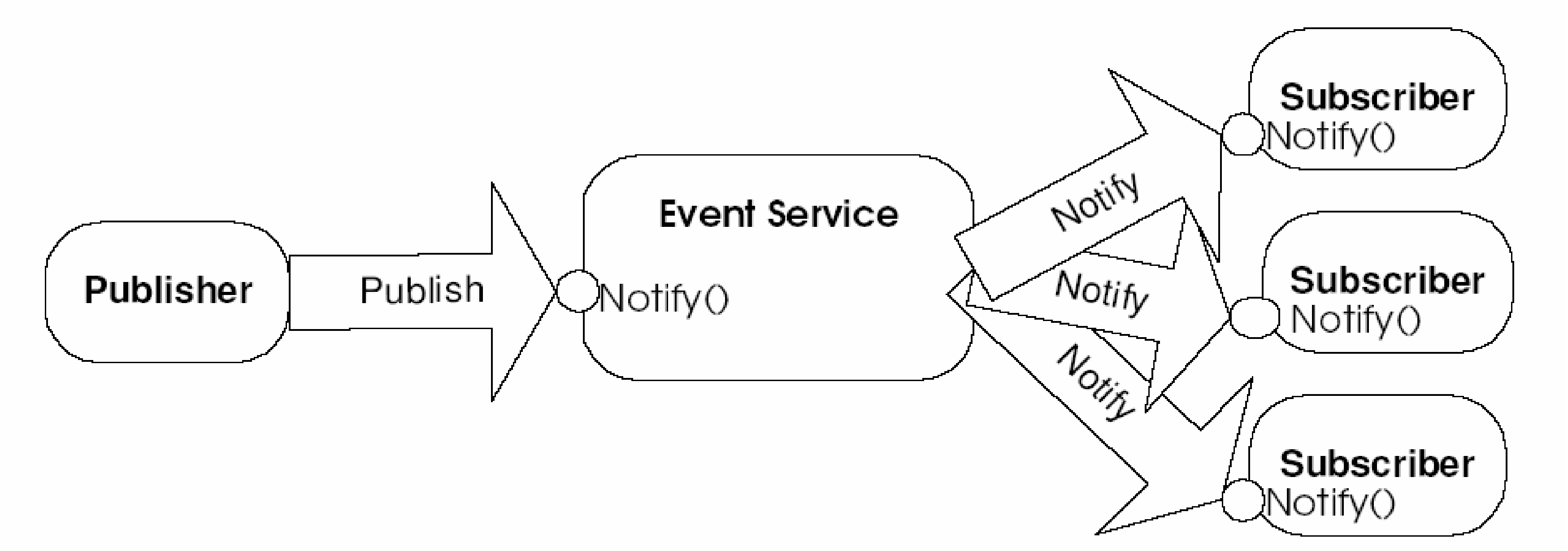
Basic Interaction Model



Why decoupling?

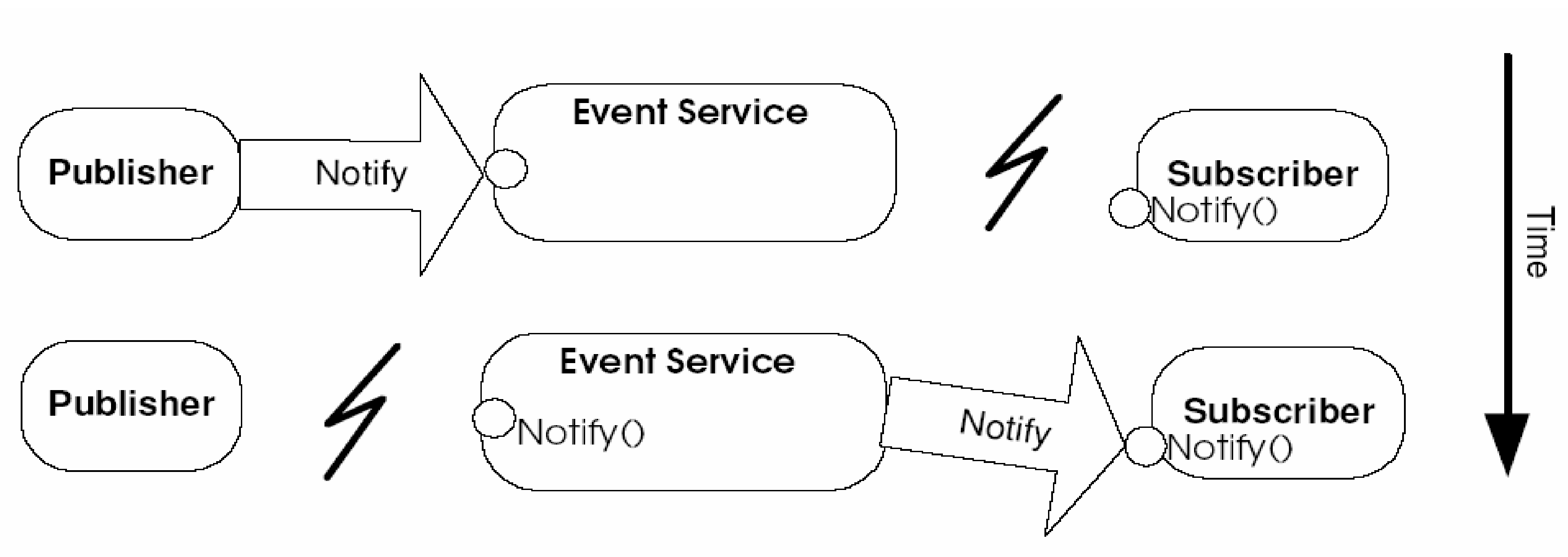
- Decouple producers and consumers
  - Remove explicit dependencies
  - Reduced coordination & synchronization between different entities
- Increase scalability of distributed systems
- Create highly dynamic, decentralized systems
- Decouple in three dimensions
  - Space
  - Time
  - Synchronization (flow)

Space decoupling



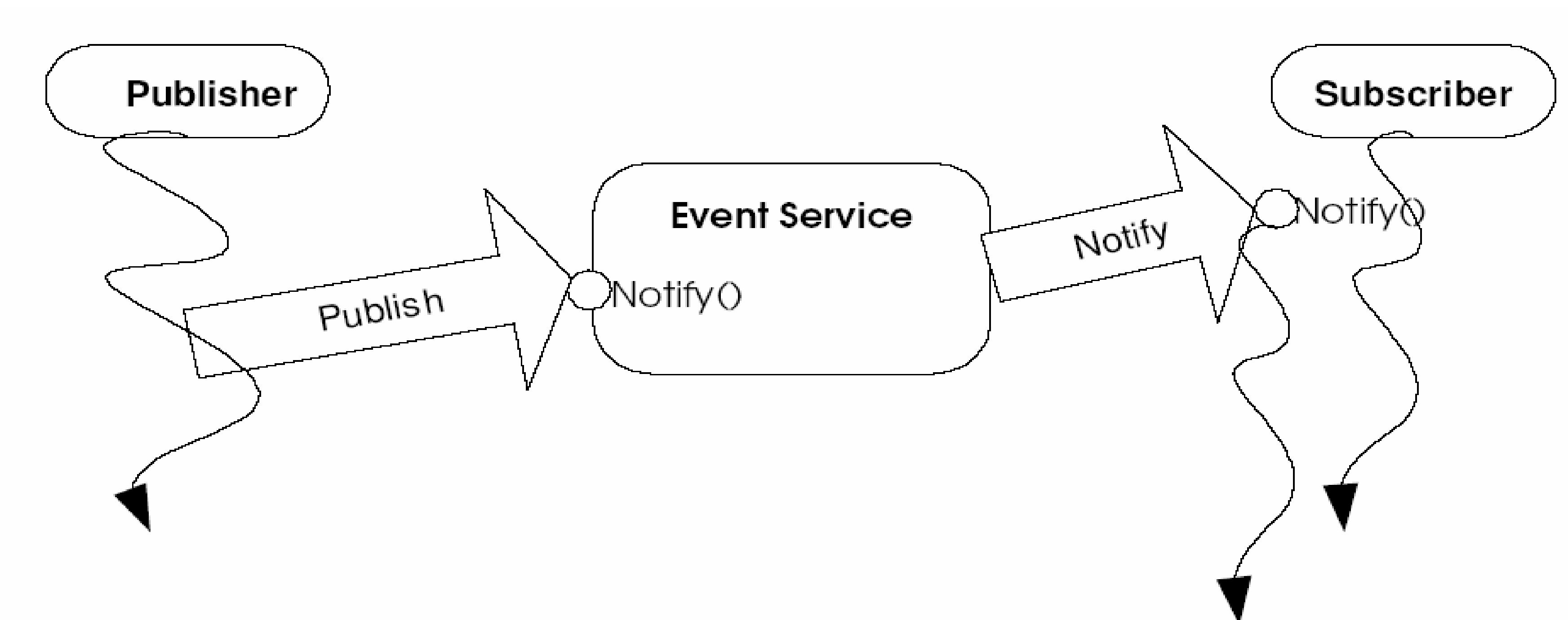
- no need to hold references or even know each other

Time decoupling



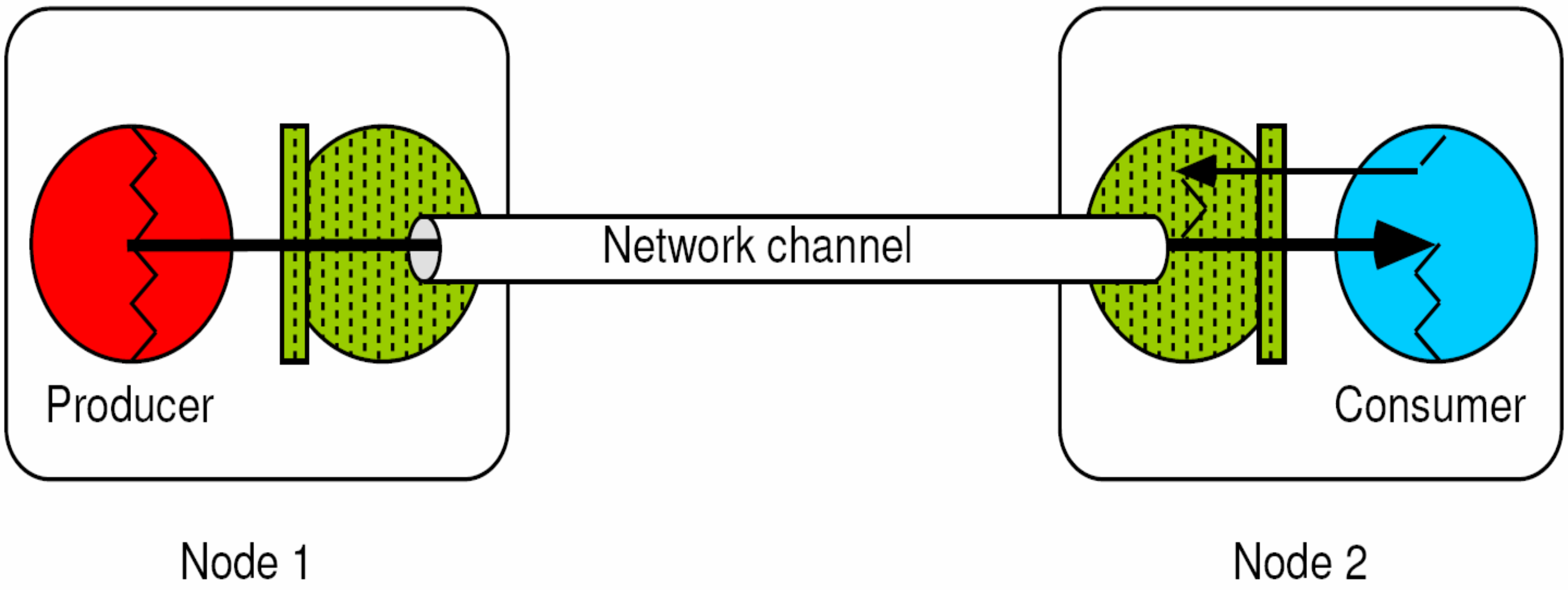
- no need to be available at the same time

Synchronization decoupling



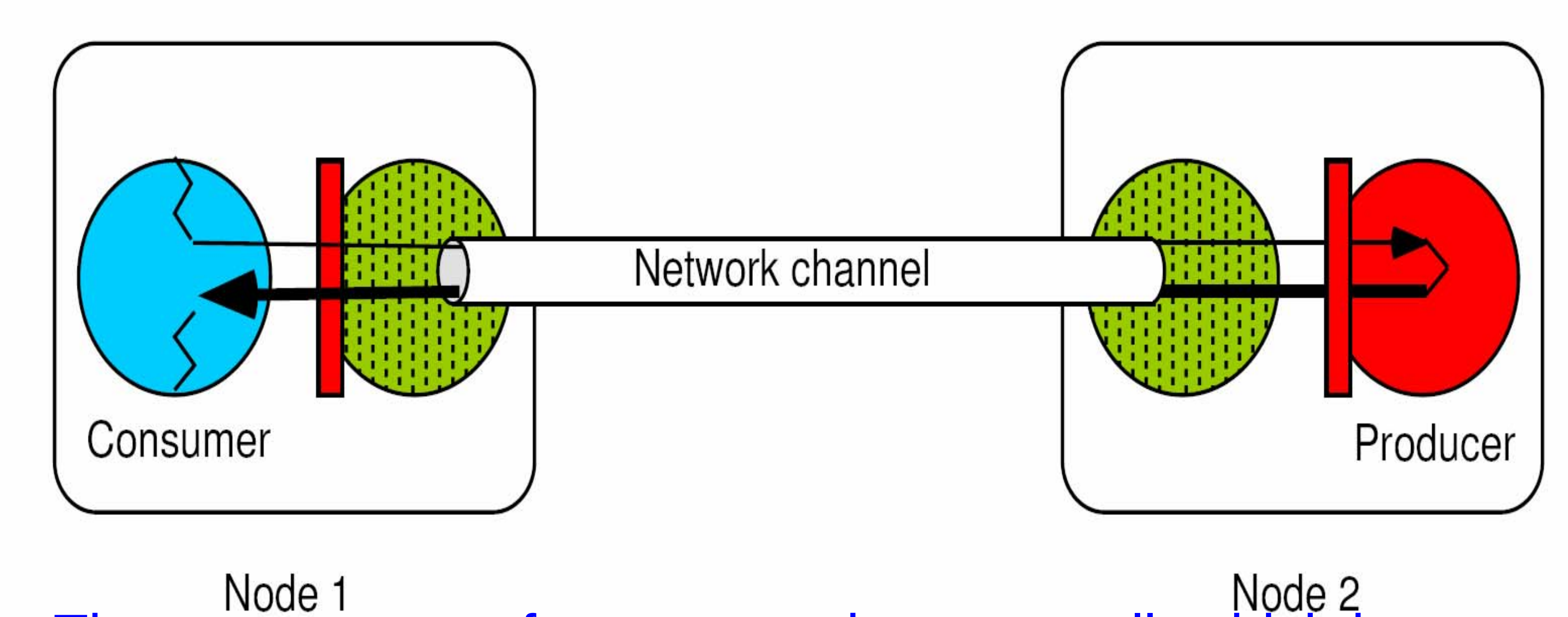
- Control flow is not blocked by the interaction

Message Passing



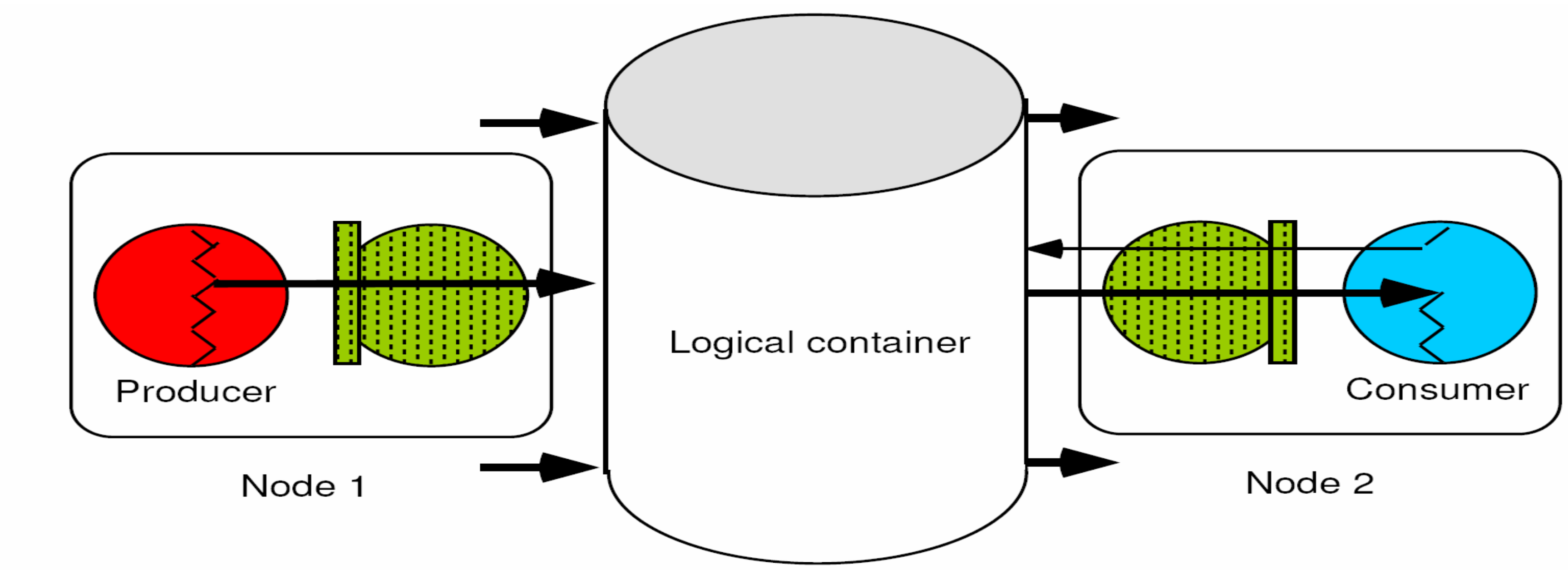
- The producer sends messages asynchronously through a communication channel (previously set up for that purpose).
- The consumer receives messages by listening synchronously on that channel.
- Coupled in space, time and sync (consumer side)

RPC/RMI



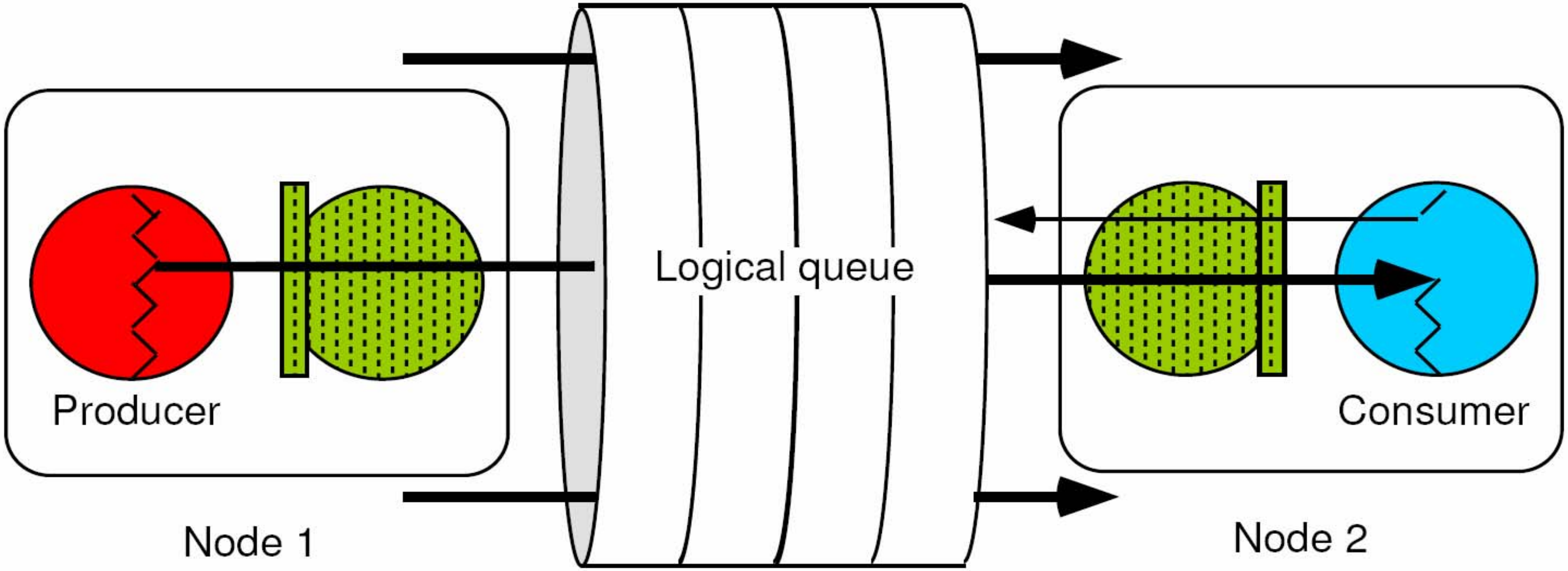
- The consumer performs a synchronous call, which is processed asynchronously by the producer.
- Coupled in space, time and sync (consumer side)

Shared Spaces



- producers insert data asynchronously into the shared space, while consumers read data synchronously.
- Coupled in sync (consumer side)

Message Queuing (Pull)



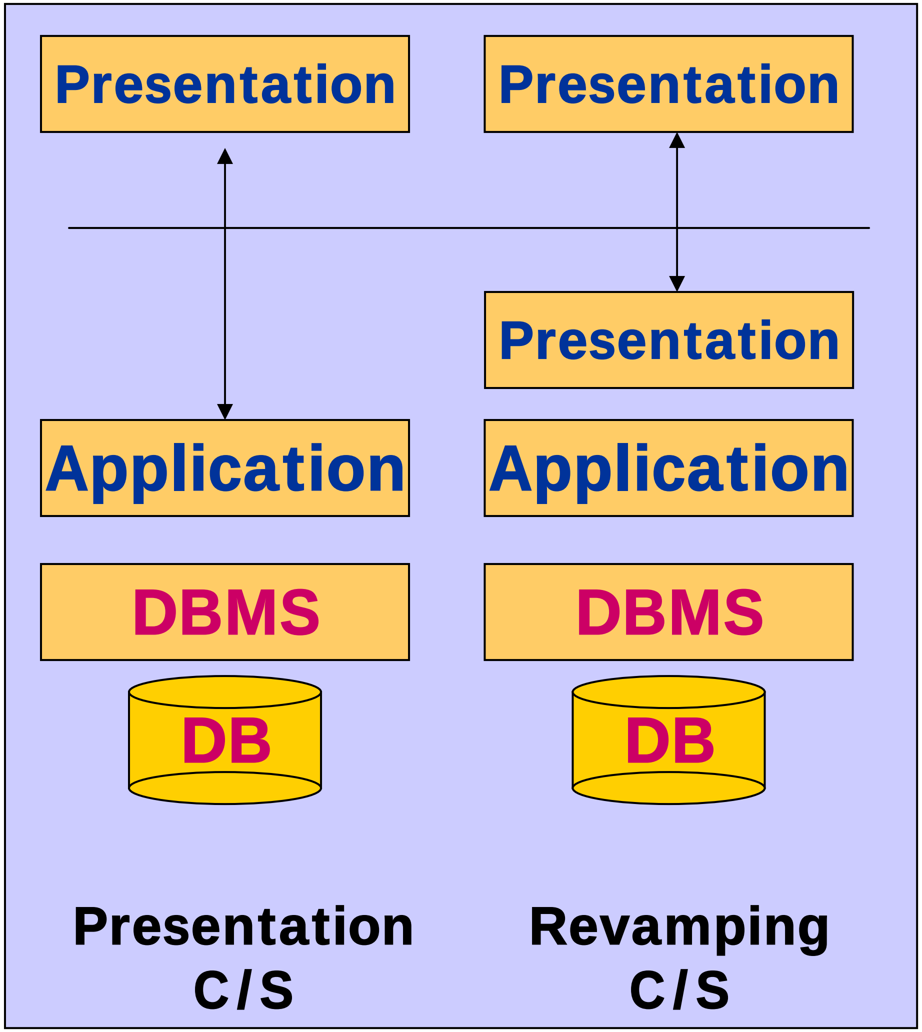
- Messages are stored in a FIFO queue.
- Producers append messages asynchronously at the end of the queue,
- Consumers dequeue them synchronously at the front of the queue.
- Coupled in sync (consumer side)

Generations Of C/S

- First Generation: frontal processing
- Second Generation: collaborative processing
- Third Generation: distributed data & processing
- Internet: to a universal C/S

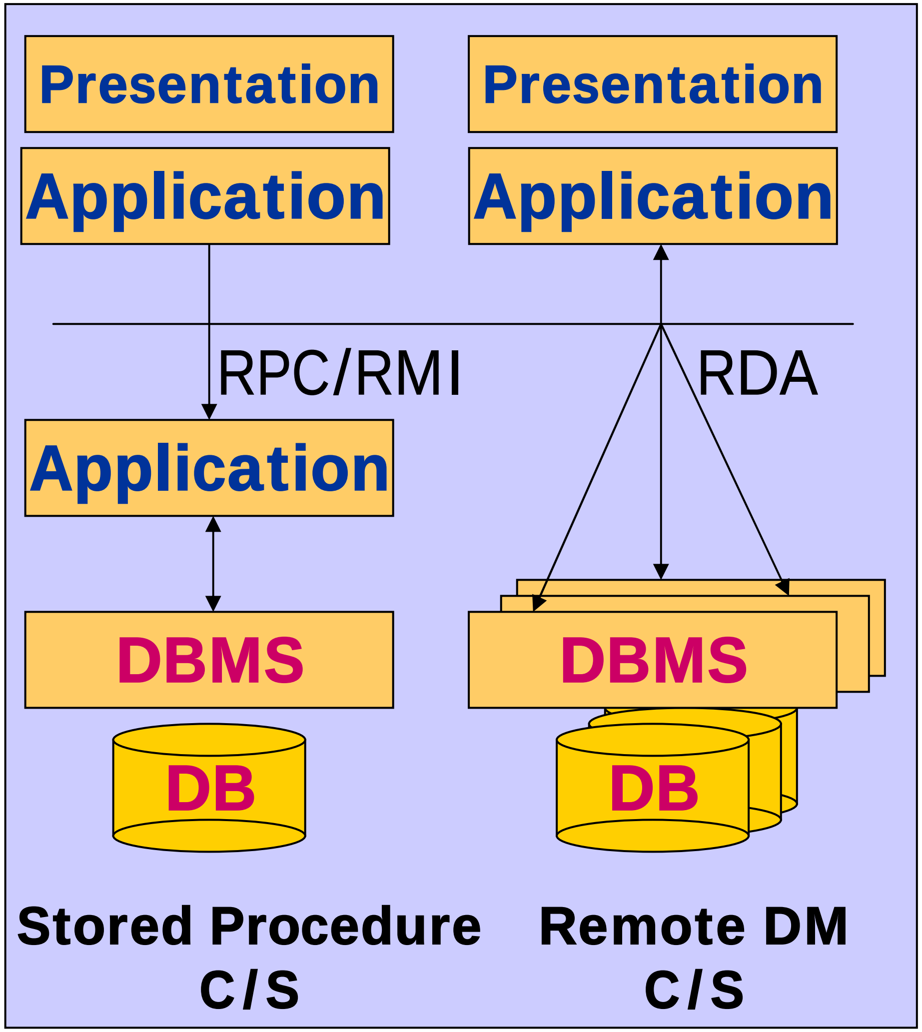
Frontal Processing

- Presentation C/S
  - The Logic of the user's interface is implemented on the client
- Revamping



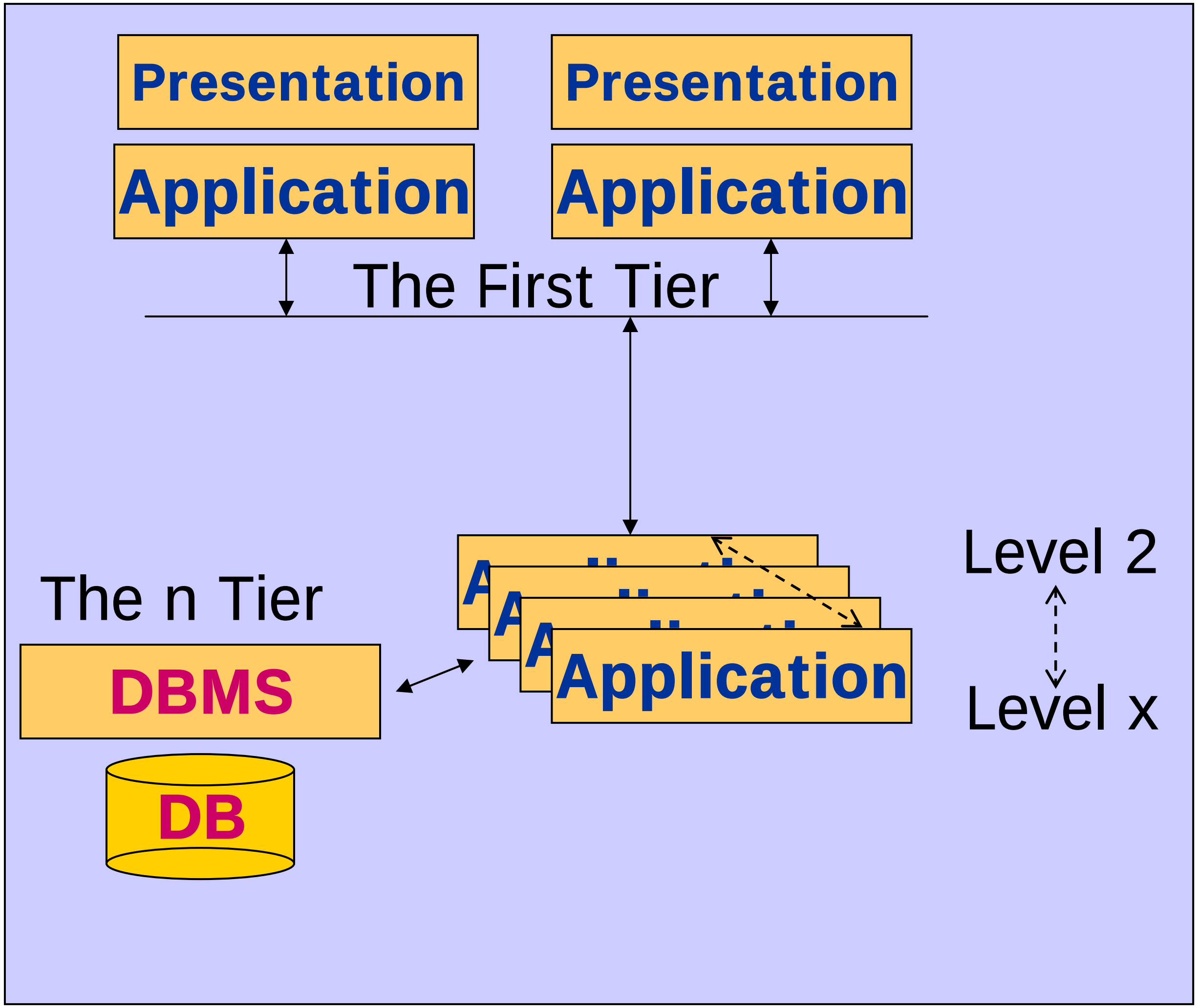
Collaborative Processing

- Stored Procedure
  - The application is spread over the client and the server.
- Remote Data Management

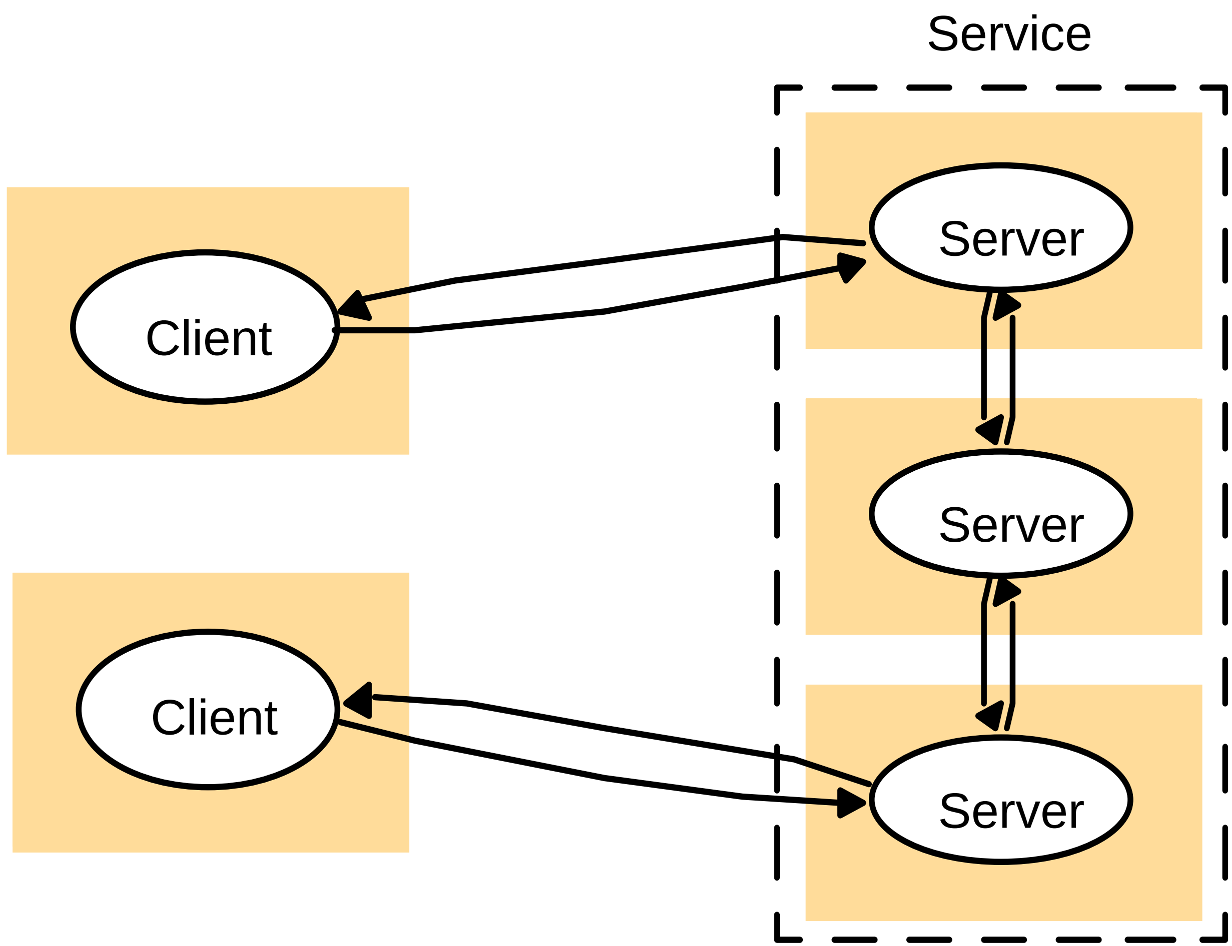


Multilevel collaborative processing

- Each server performs a special task.
- A server may request a specific task from an other server.



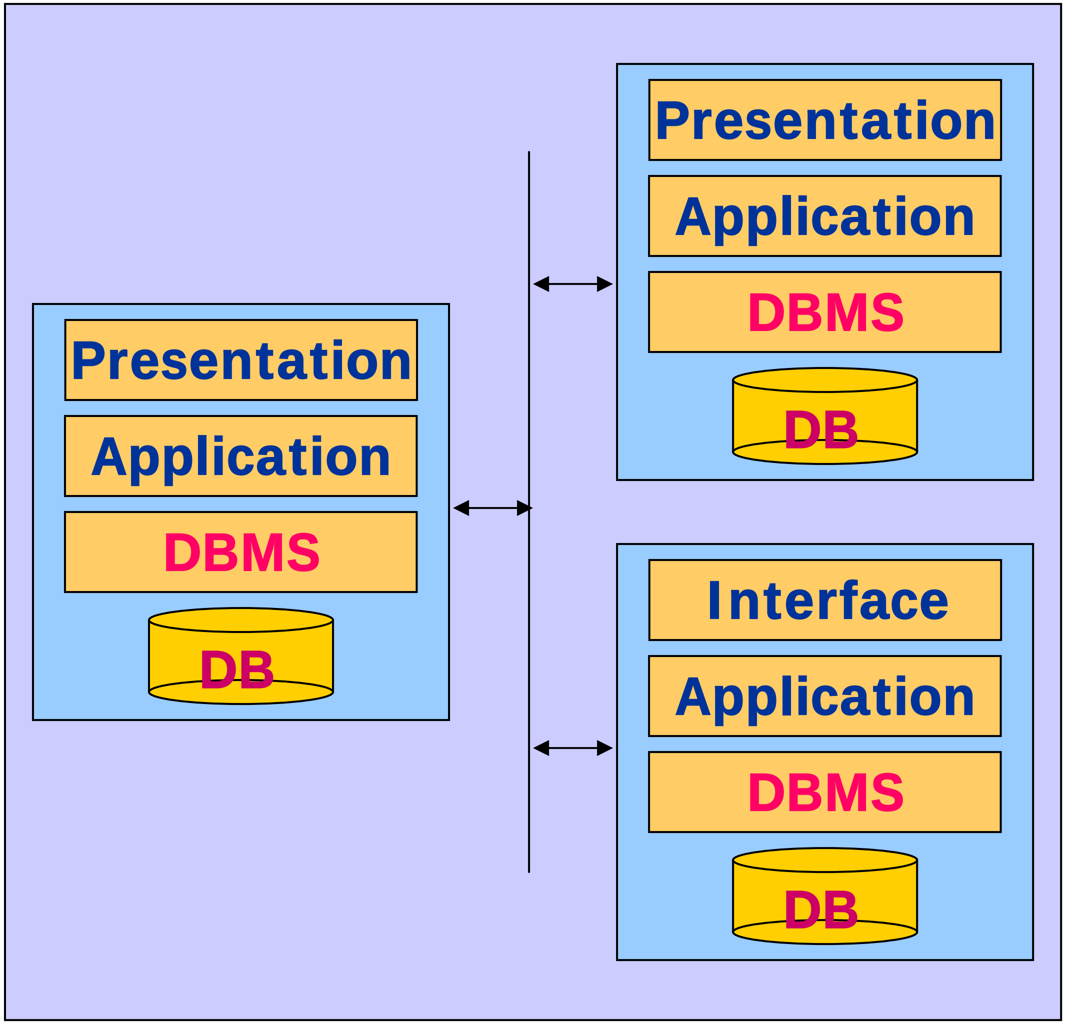
A service provided by multiple servers



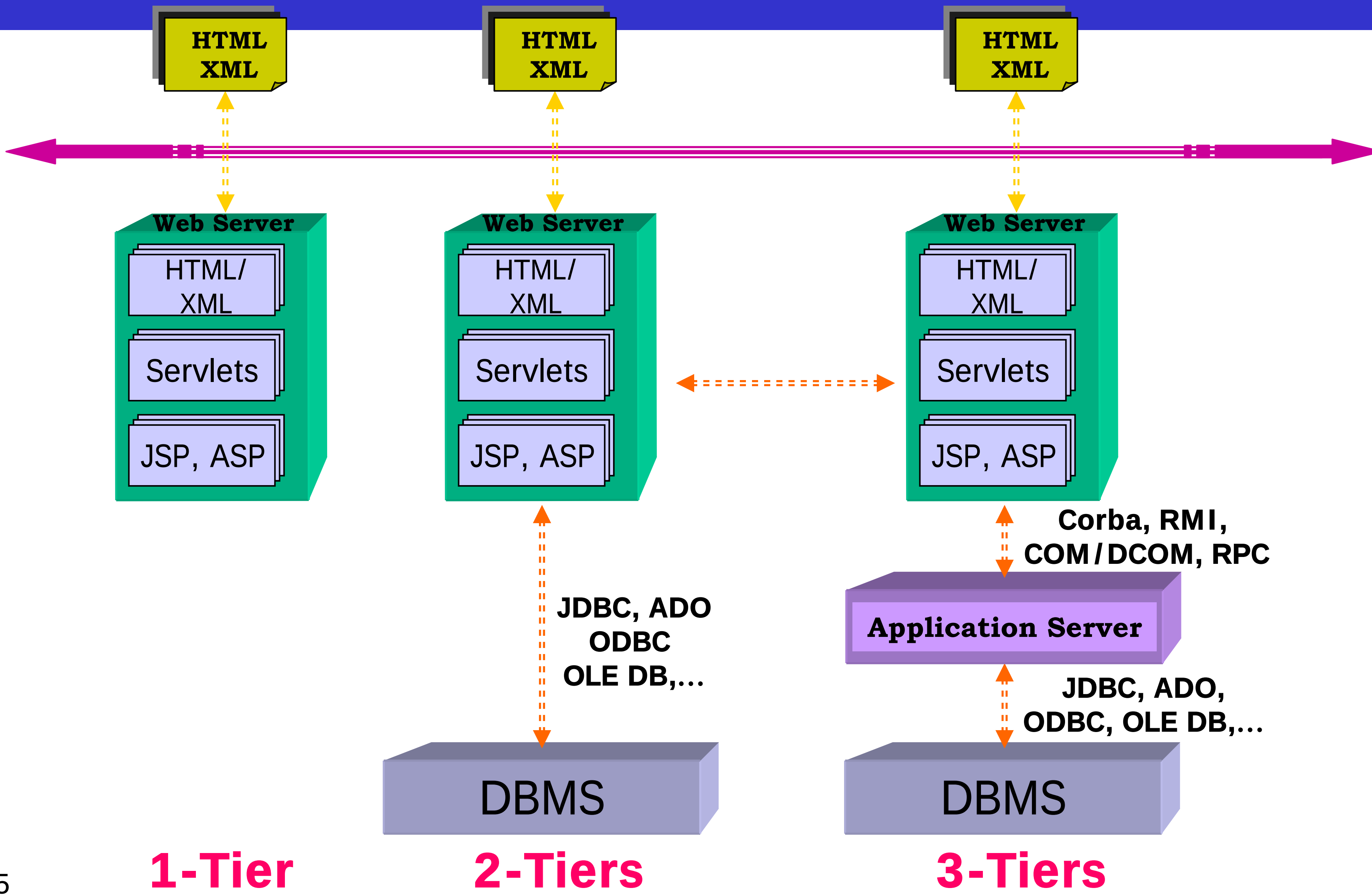
- Services may be implemented as several server processes in separate host computers.
- Example: Cluster based Web servers and apps such as Google, parallel databases Oracle

Distributed Data & Processing

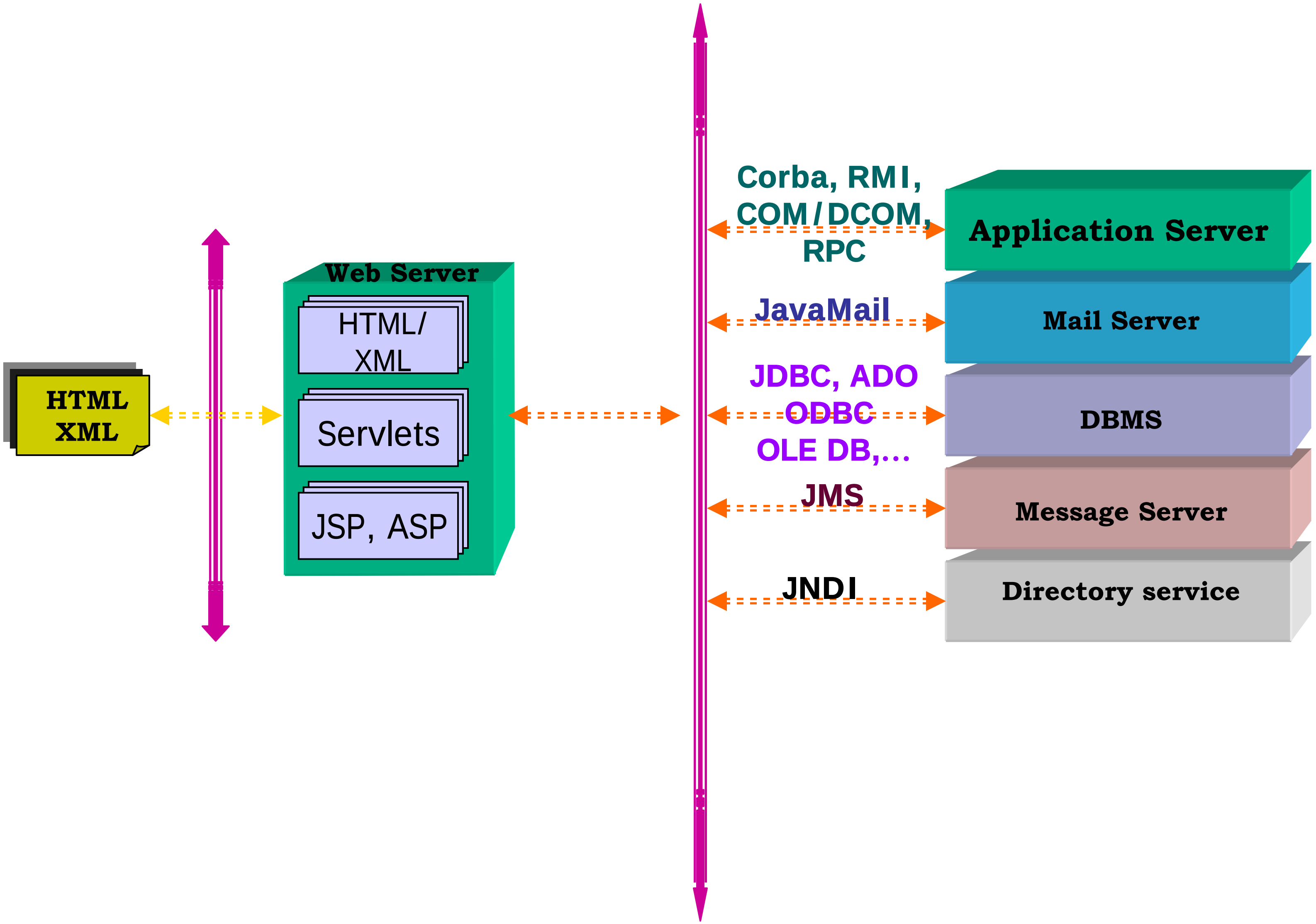
- No machine has complete information on the state of the system
- Failure of one machine does not ruin the system.
- Machines make decisions based only on local available information.
- No assumption of the existence of a global clock.
- Symmetric processing



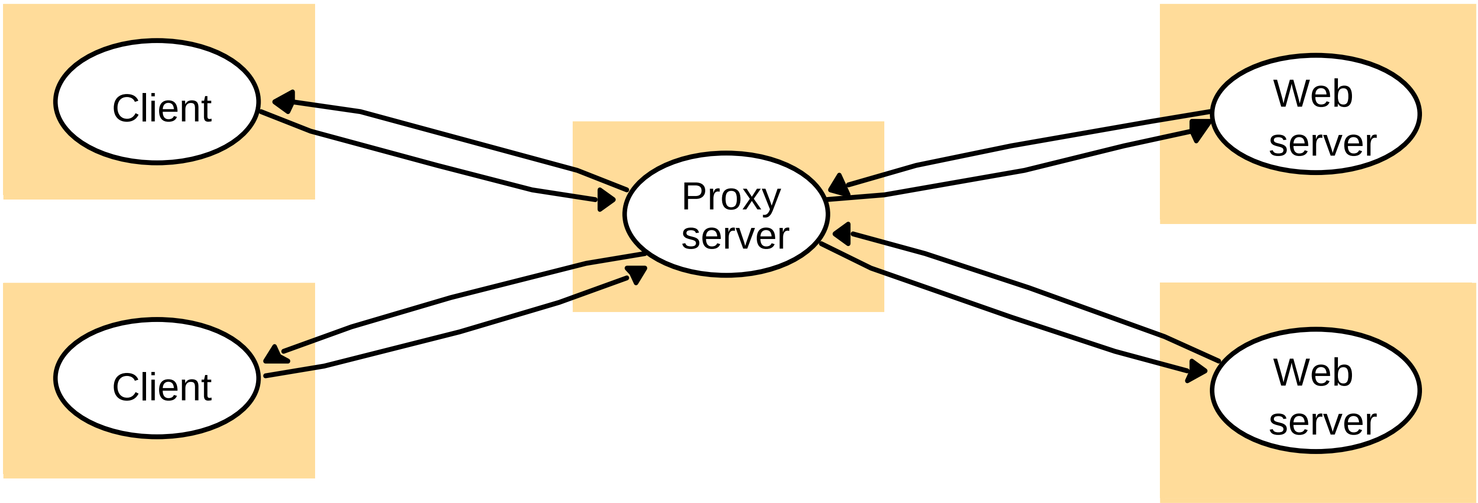
Internet & New C/S Architectures



Internet: Universal C/S



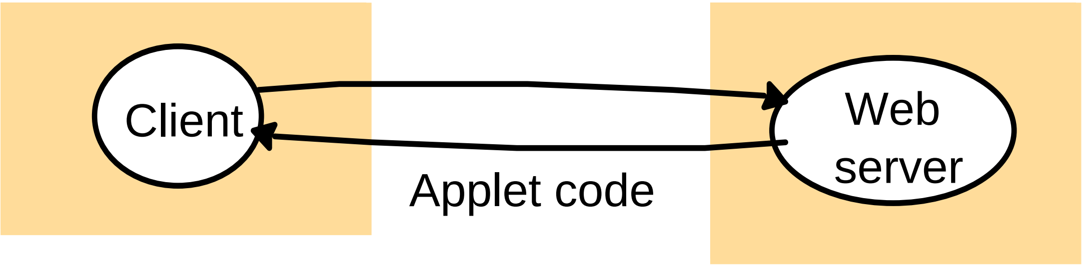
Proxy servers (replication transparency) and caches: Web proxy server



- A cache is a store of recently used data.

Variants of Client Sever Model: Mobile code and Web applets

a) client request results in the downloading of applet code



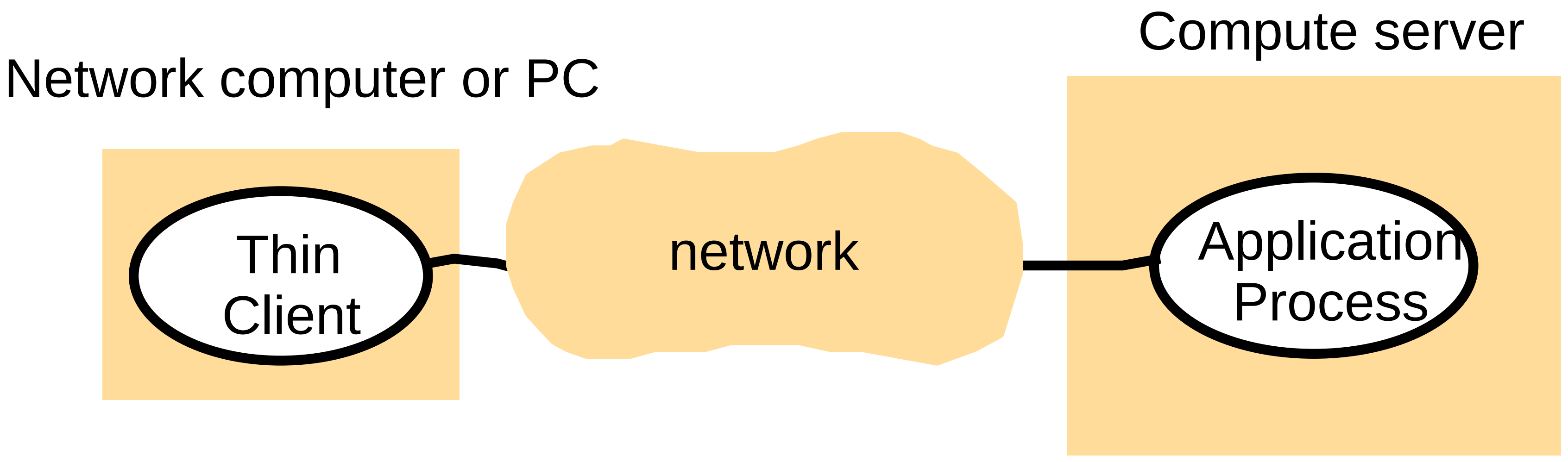
b) client interacts with the applet

- Applets downloaded to clients give good interactive response
- Mobile codes such as Applets are potential security threat, so the browser gives applets limited access to local resources (e.g. NO access to local/user file system).

Variants of Client Sever Model: Mobile Agents

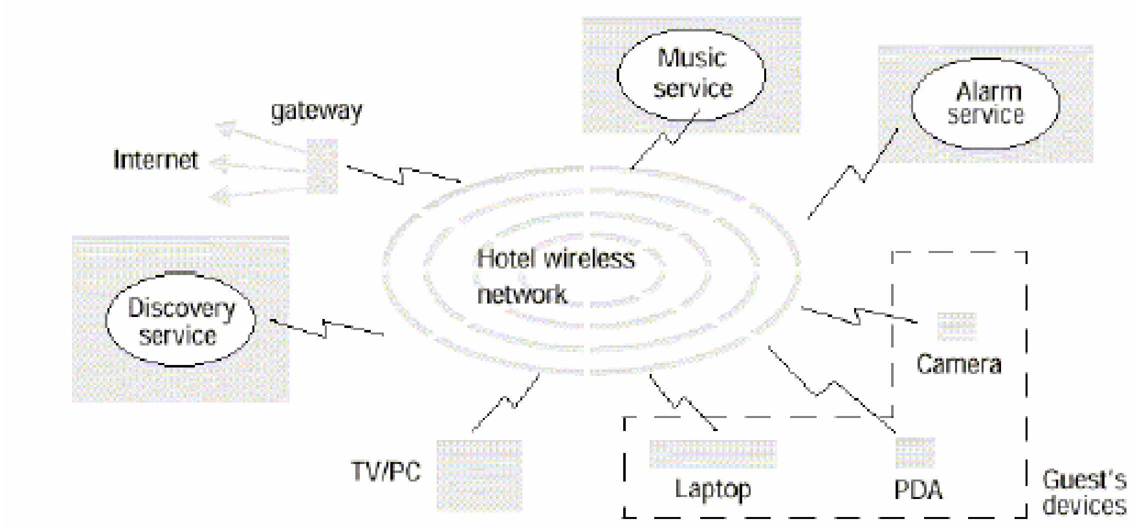
- A running program (code and data) that travels from one computer to another in a network carrying out of an autonomous task, usually on behalf of some other process
  - advantages: flexibility, savings in communications cost virtual markets, software maintain on the computers within an organization.
- Potential security threat to the resources in computers they visit. The environment receiving agent should decide which of the local resource to allow. (e.g., crawlers and web servers).
- Agents themselves can be vulnerable – they may not be able to complete task if they are refused access.

Thin clients and compute servers



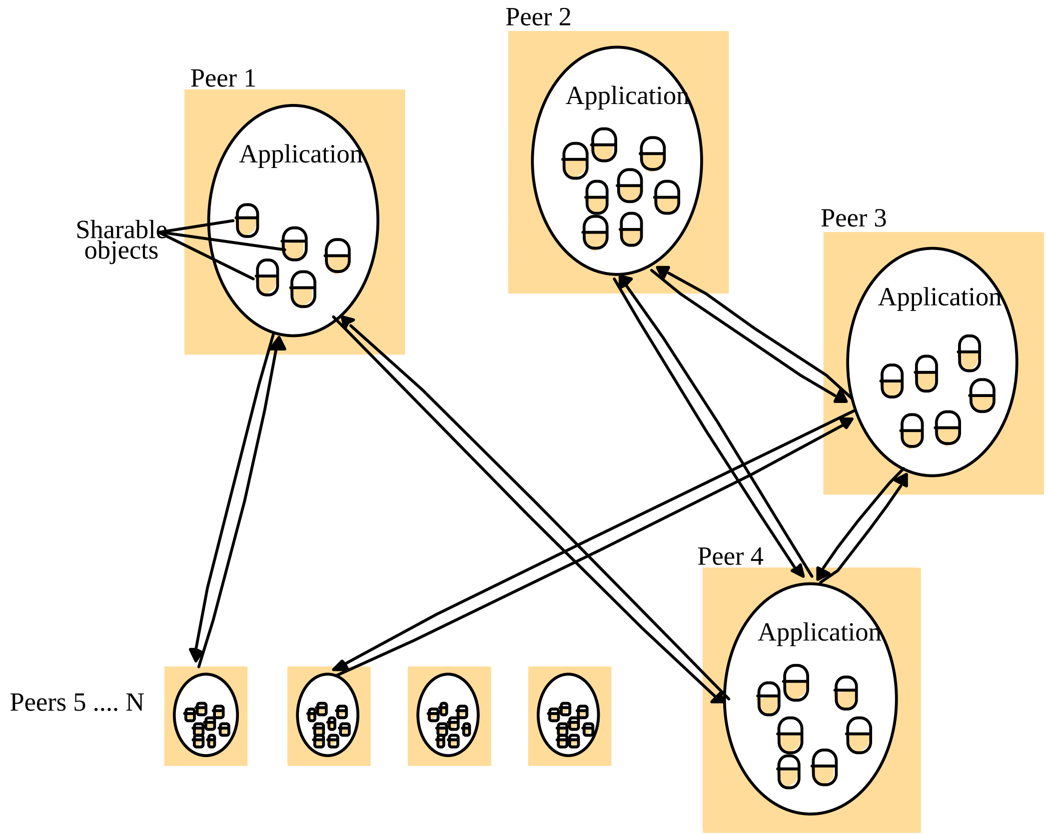
- Network computer: download OS and applications from the network and run on a desktop (solve up-gradation problem) at runtime.
- Thin clients: Windows-based UI on the user machine and application execution on a remote computer. E.g, X-11 system.

Mobile devices and spontaneous networking



- The world is increasingly populated by small and portal computing devices.
- W-LAN need to handle constantly changing heterogamous, roaming devices
- Need to provide discovery services: (1) *registration service* to enable servers to publish their services and (2) *look up service* to allow clients to discover services that meet their requirements.

Peer Processes: A distributed application based on peer processes



- All of the processes play similar roles, interacting cooperatively as peers to perform a distributed activities or computations without distinction between clients and servers. E.g., music sharing systems Gnutella, Napster, Kaza, etc.
- Distributed "**white board**" – users on several computers to view and interactively modify a picture between them.

Interfaces and Objects

- The use of CS has impact on the software architecture followed:
  - Distribution of responsibilities
  - Synchronization mechanisms between client and server
  - Admissible types of requests/responses
- Basic CS model, responsibility is statically allocated.
  - File server is responsible for file, not for web pages.
- Peer process model, responsibility is dynamically allocated:
  - In fully decentralized music file sharing system, search process may be delegated to different peers at runtime.

Design Requirements/Challenges of Distributed Systems

- Performance Issues
  - Responsiveness
    - Support interactive clients
      - Use caching and replication
  - Throughput
  - Load balancing and timeliness
- Quality of Service:
  - Reliability
  - Security
  - Adaptive performance.
- Dependability issues:
  - Correctness, security, and fault tolerance
  - Dependable applications continue to work in the presence of faults in hardware, software, and networks.

Presentation Outline

- Introduction
- Architectural Models
  - Software Layers
  - System Architectures
    - Client-Server
      - Clients and a Single Sever, Multiple Servers, Proxy Servers with Cachers, Peer Model
    - Alternative Client-Sever models driven by:
      - Mobile code, mobile agents, network computers, thin clients, mobile devices and spontaneous networking
    - Design Challenges/Requirements
- Fundamental Models – formal description
  - Interaction, Failure, and Security models.
- Summary

Lecture Overview (II)

- Fundamental Models are concerned with a formal description of the properties that are common in all of the architectural models.
- All architectural models composed of processes that communicate with each other by sending messages over a computer networks.
- Models addressing time synchronization, message delays, failures, security issues are addressed are:
  - Interaction Model – deals with performance and the difficulty of setting of time limits in a distributed system.
  - Failure Model – specification of the faults that can be exhibited by processes
  - Secure Model – discusses possible threats to processes and communication channels.

Interaction Model

- Computation occurs within processes;
- The processes interact by passing messages, resulting in:
  - Communication (information flow)
  - Coordination (synchronization and ordering of activities) between processes.
- Two significant factors affecting interacting processes in a distributed system are:
  - Communication performance is often a limiting characteristic.
  - It is impossible to maintain a single global notion of time.

Interaction Model: Performance of Communication Channel

- The communication channel in our model are realised in a variety of ways in DSs. E.g., by implementation of:
  - Streams
  - Simple message passing over a network.
- Communication over a computer network has performance characteristics:
  - Latency:
    - A delay between the start of a message's transmission from one process to the beginning of receipt by another.
  - Bandwidth:
    - the total amount of information that can be transmitted over in a given time.
    - Communication channels using the same network, have to share the available bandwith.
  - Jitter
    - The variation in the time taken to deliver a series of messages. It is vary relevant to multimedia data.

Interaction Model: Computer clocks and timing events

- Each computer in a DS has its own internal clock, which can be used by local processes to obtain the value of the current time.
- Therefore, two processes running on different computers cam associate timestamp with their events.
- However, even if two processes read their clock at the same time, their local clocks may supply different time.
  - This is because computer clock drift from perfect time and their drift rates differ from one another.
- Even if the clocks on all the computers in a DS are set to the same time initially, their clocks would eventually vary quite significantly unless corrections are applied.
  - There are several techniques to correcting time on computer clocks. For example, computers may use radio receivers to get readings from GPS (Global Positioning System) with an accuracy about 1 microsecond.

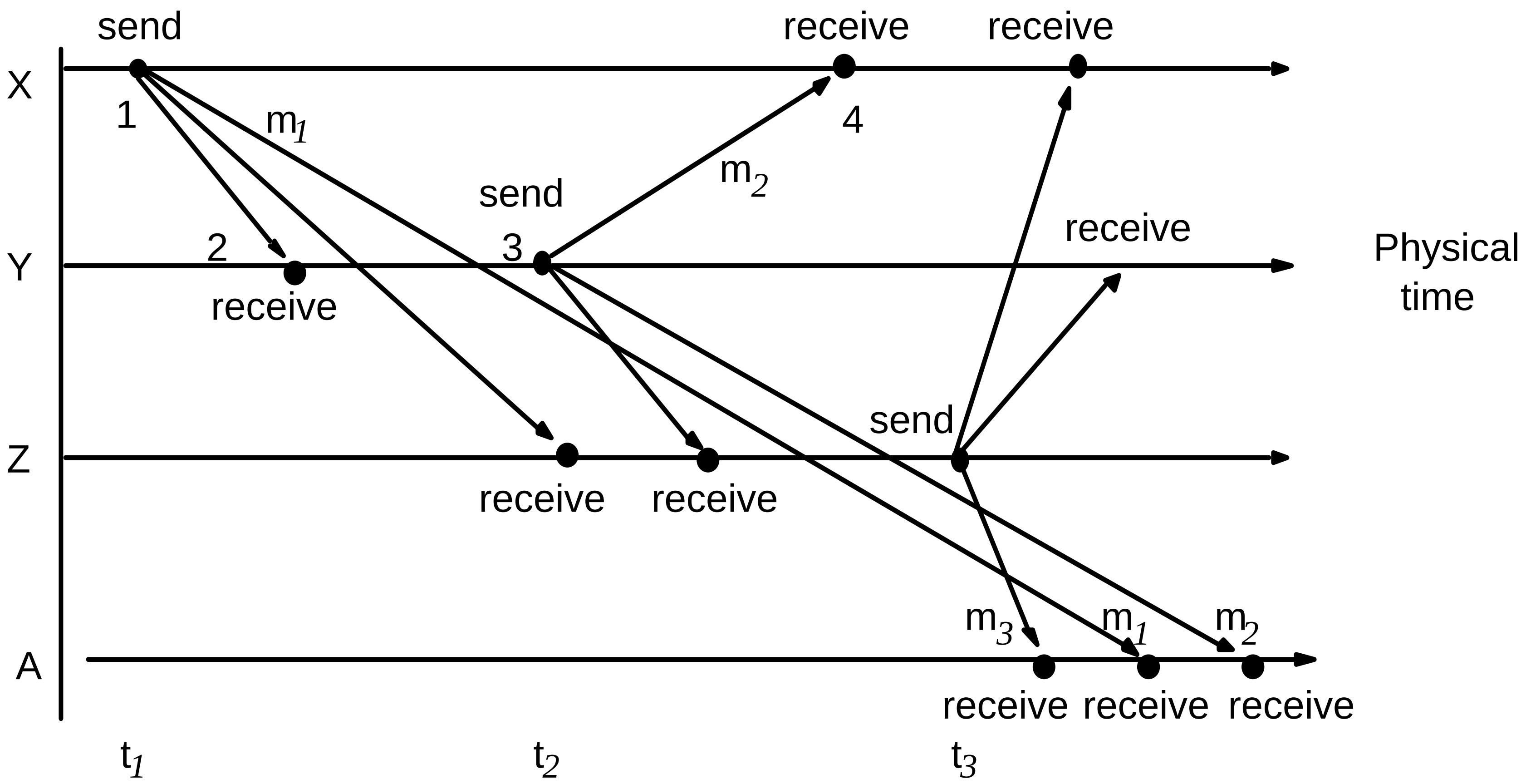
Interaction Model: Two variants of the interaction model

- In a DS it is hard to set time limits on the time taken for process execution, message delivery or clock drift.
- Synchronous DS – hard to achieve:
  - The time take to execute a step of a process has know 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 DS: There is NO bounds on:
  - Process execution speeds
  - Message transmission delays
  - Clock drift rates.

Interaction Model: Event Ordering

- In many DS applications we are interested in knowing whether an event occurred before, after, or concurrently with another event at another processes.
  - The execution of a system can be described in terms of events and their ordering despite the lack of accurate clocks.
- Consider a mailing list with users X, Y, Z, and A.

Real-time ordering of events



Inbox of User A looks like:

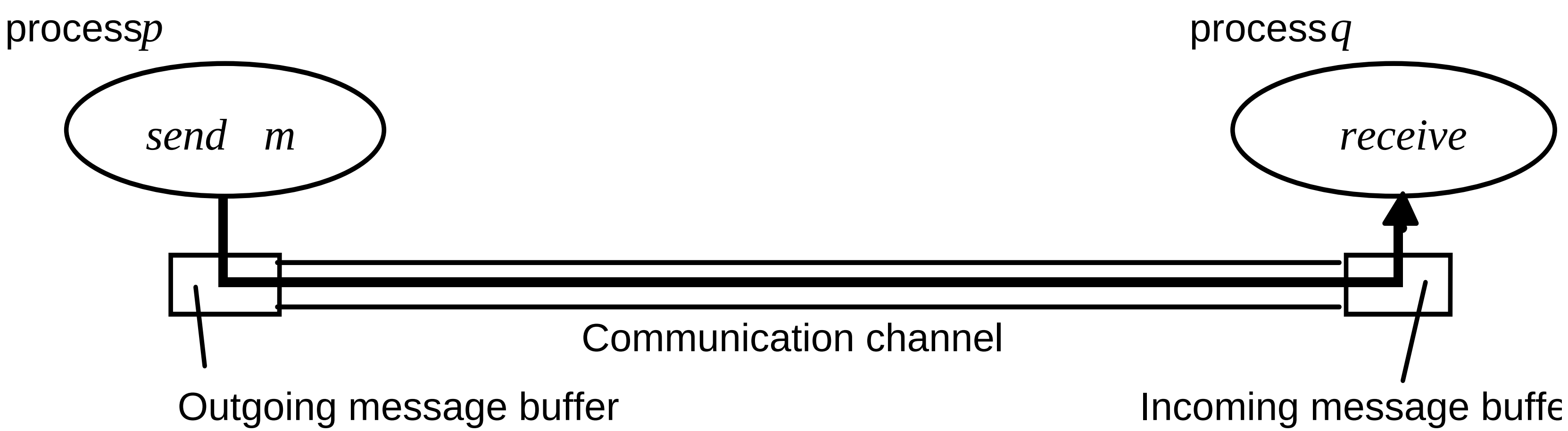
Item	*From*	*Subject*
23	Z	Re: Meeting
24	X	Meeting
26	Y	Re: Meeting

- Due to independent delivery in message delivery, message may be delivered in different order.
- If messages m1, m2, m3 carry their time t1, t2, t3, then they can be displayed to users accordingly to their time ordering.

Failure Model

- In a DS, both processes and communication channels may fail – i.e., they may depart from what is considered to be correct or desirable behavior.
- Types of failures:
  - Omission Failure
  - Arbitrary Failure
  - Timing Failure

Processes and channels



- Communication channel procedeuces an omission failure if it does not transport a message from “p”s outgoing message buffer to “q”’s incoming message buffer. This is known as “dropping messages” and is generally caused by lack of buffer space at the receiver or at gateway or by a network transmission error.

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.

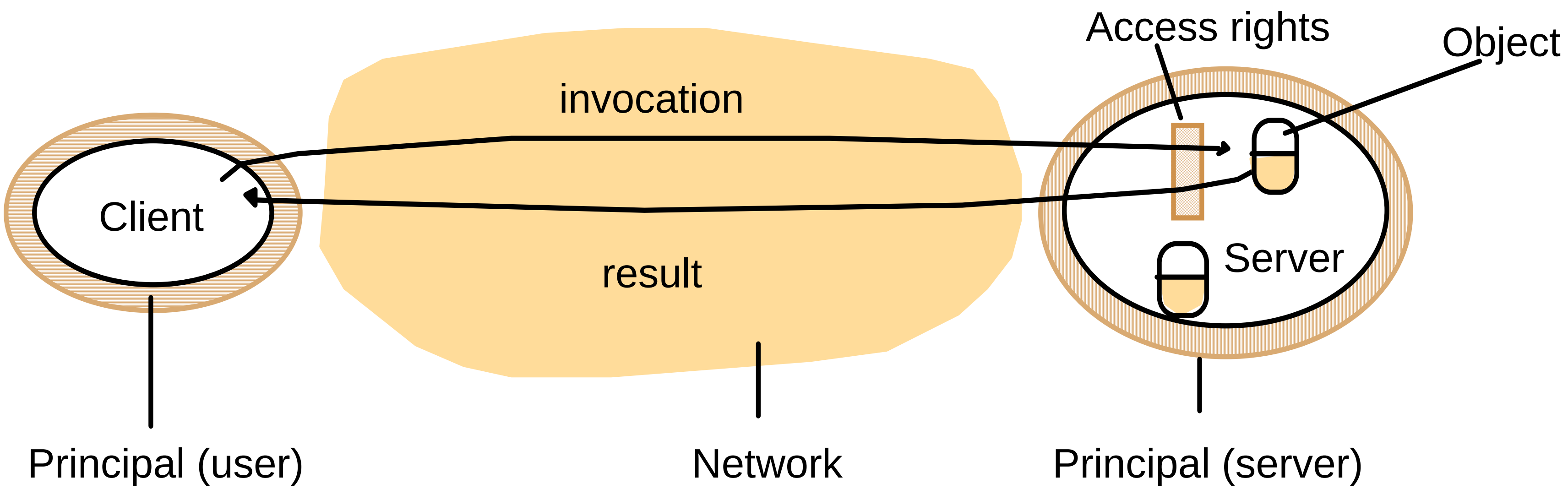
Masking Failures

- It is possible to construct reliable services from components that exhibit failures.
  - For example, multiple servers that hold replicas of data can continue to provide a service when one of them crashes.
- A knowledge of failure characteristics of a component can enable a new service to be designed to mask the failure of the components on which it depends:
  - Checksums are used to mask corrupted messages.

Security Model

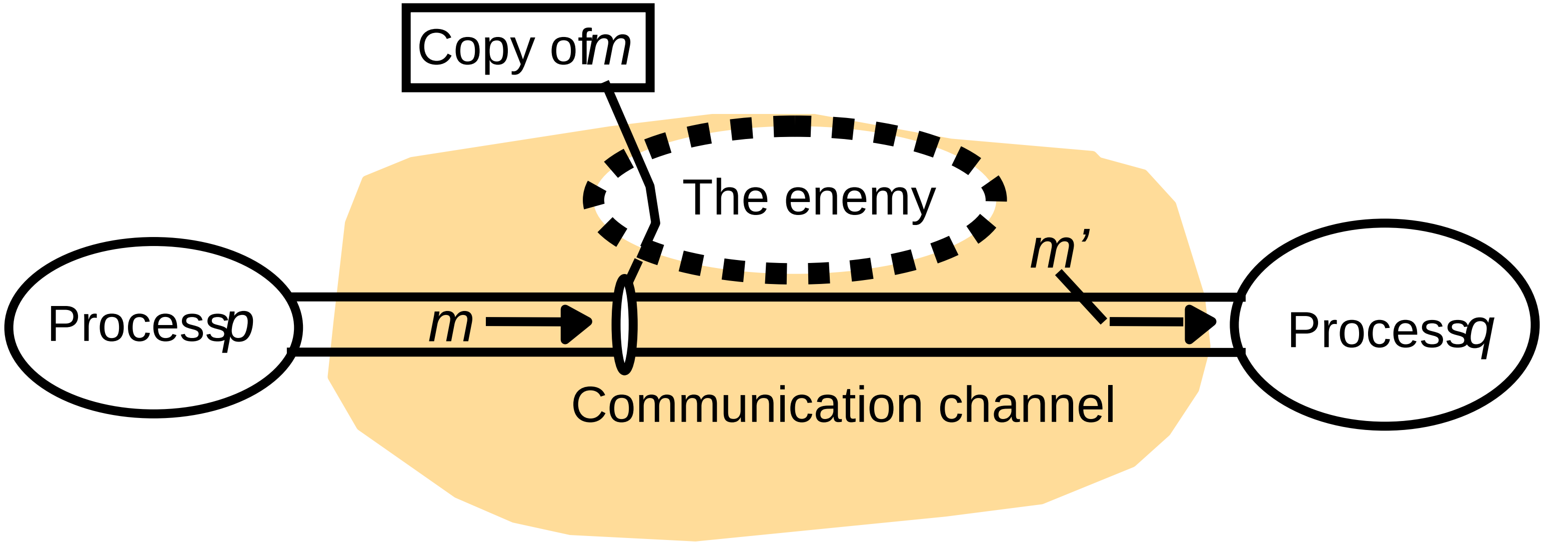
- The security of a DS can be achieved by securing the processes and the channels used in their interactions and by protecting the objects that they encapsulate against unauthorized access.

Protecting Objects: Objects and principals



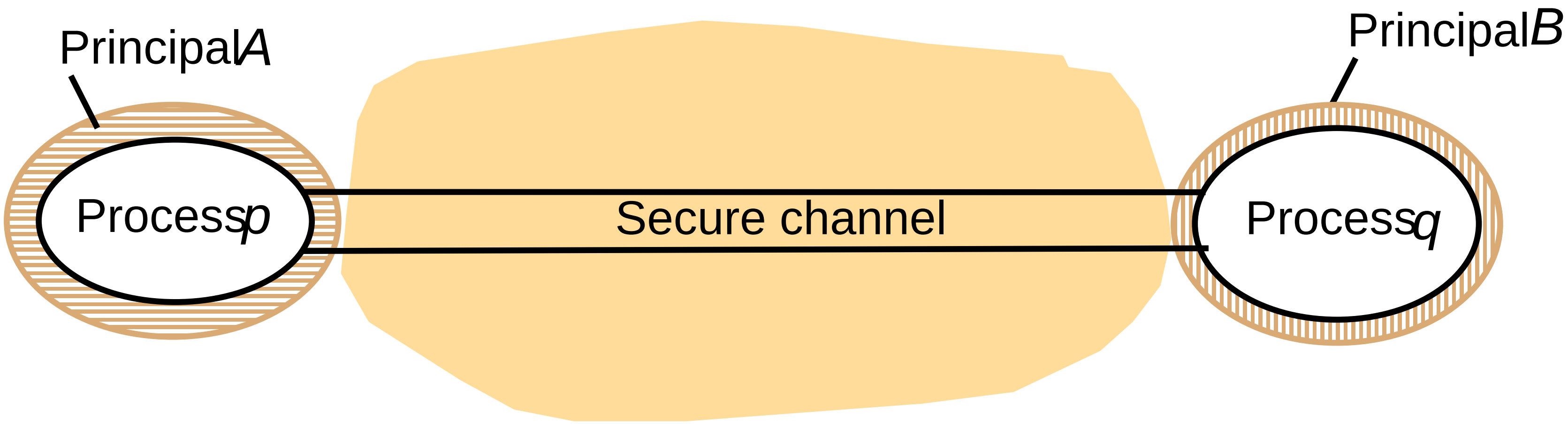
- Use “access rights” that define who is allowed to perform operation on a object.
- The sever should verify the identity of the principal (user) behind each operation and checking that they have sufficient access rights to perform the requested operation on the particular object, rejecting those who do not.

The enemy



- To model security threats, we postulate an enemy that is capable of sending any process or reading/copying message between a pair of processes
- Threats form a potential enemy: threats to processes, threats to communication channels, and denial of service.

Defeating security threats: Secure channels



- Encryption and authentication are use to build secure channels.
- Each of the processes knows the identity of the principal on whose behalf the other process is executing and can check their access rights before performing an operation.

Summary

- Most DSs are arranged accordingly to one of a variety of architectural models:
  - Client-Server
    - Clients and a Single Sever, Multiple Servers, Proxy Servers with Cache, Peer Model
  - Alternative Client-Sever models driven by:
    - Mobile code, mobile agents, network computers, thin clients, mobile devices and spontaneous networking
- Fundamental Models – formal description
  - Interaction, failure, and Security models.
- The concepts discussed in the module play an important role while architecting DS and apps.