Distributed System Models

Some ideas from Chapter 1

Presentation Outline

- *Introduction*
- *Physical Models:*
  - *Three Generations of DS: Early, Internet-Scale, Contemporary*
- *Architectural Models*
  - *Software Layers*
  - *System Architectures*
    - *Client-Server*
      - *Clients and a Single Server, Multiple Servers, Proxy Servers with Caches, 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*

Introduction

- *Real world systems should be designed to function correctly in ALL circumstances/scenarios.*
- *Distributed system models helps in…*
  - *..classifying and understanding different implementations*
  - *..identifying their weaknesses and their strengths*
  - *..crafting new systems outs of pre-validated building blocks*
- *We will study distributed system models from different perspectives*
  - *Structure, organization, and placement of components*
  - *Interactions*
  - *Fundamental properties of systems*

Characterization

- *The structure and the organization of systems and the relationship among their components should be designed with the following goals in mind:*
  - *To cover the widest possible range of circumstances.*
  - *To face the possible difficulties and threats.*
  - *To meet the current and possibly the future demands.*

Characterization: Challenges (Difficulties and Threats)

- *Widely varying models of use*
  - *High variation of workload, partial disconnection of components, or poor connection.*
- *Wide range of system environments*
  - *Heterogeneous hardware, operating systems, network, and performance.*
- *Internal problems*
  - *Non synchronized clocks, conflicting updates, various hardware and software failures.*
- *External threats*
  - *Attacks on data integrity, secrecy, and denial of service.*

Distributed System Design

- *In this chapter it will be showed that how the properties and design issues of distributed systems can be captured and discussed through the use of descriptive models. Each type of model is intended to provide an abstract, simplified but consistent description of a relevant aspect of distributed system design*

Distributed System Design

- *Physical Model*
  - *Physical models are the most explicit way in which to describe a system; they capture the hardware composition of a system in terms of the computers and their interconnecting networks*
- *Architectural models*
  - *An Architectural model of a distributed system is concerned with the placement of its parts and relationship between them*
- *Fundamental Model*
  - *Fundamental Models are concerned with a formal description of the properties that are common in all of the architectural models*

Physical Models

- *A representation of the underlying h/w elements of a DS that abstracts away specific details of the computer/networking technologies.*
- *Baseline physical model*
- *3 Generations of DS:*
  - *Early distributed systems [late 70-80s]: LAN-based*
  - *Internet-scale distributed systems [early 90-2005]: Clusters, grids, P2P, Clouds*
  - *Contemporary distributed systems: dynamic nodes like mobile-based services (nodes are very dynamic not static like other models).*

Physical Models

- ***Distributed systems of systems***
  - *A recent report discusses the emergence of ultralarge-scale (ULS) distributed systems*
  - *systems of systems*
    - *A system of systems can be defined as a complex system consisting of a series of subsystems that are systems in their own right and that come together to perform a particular task or tasks.*

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

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 making 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 elements

- *To understand the fundamental building blocks of a distributed system, it is necessary to consider four key questions*
  - *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 to the physical distributed infrastructure (what is their placement)?*

Communicating entities

- *to address the first question tow perspectives are exist*
  - *System-oriented perspective*
    - *From a system perspective, the entities that communicate in a distributed system are typically processes*
      - *However, In some environments the entities that communicate are nodes*
  - *Problem-oriented perspective (programming perspective)*
    - *Objects*
    - *Components*
    - *Web Services*

Communication paradigms

- *three types of communication paradigm*
  - *interprocess communication*
    - *communication between processes in distributed systems, including message-passing primitives, direct access to the API offered by Internet protocols*
  - *remote invocation*
  - *indirect communication*

Communicating entities and communication paradigms

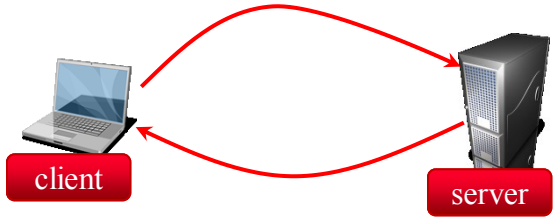
Communicating entities (what is communicating)		*Communication paradigms (how they communicate)*		
System-oriented entities	*Problem-oriented entities*	*Interprocess communication*	*Remote invocation*	*Indirect communication*
Nodes	Objects	Message passing	Request-reply	Group communication
Processes	Components	Sockets	RPC	Publish-subscribe
	Web services	Multicast	RMI	Message queues
				Tuple spaces
				DSM

Roles and responsibilities

- *two architectural styles stemming from the role of individual processes*
  - *client-server*
  - *peer-to-peer.*

System Architectures

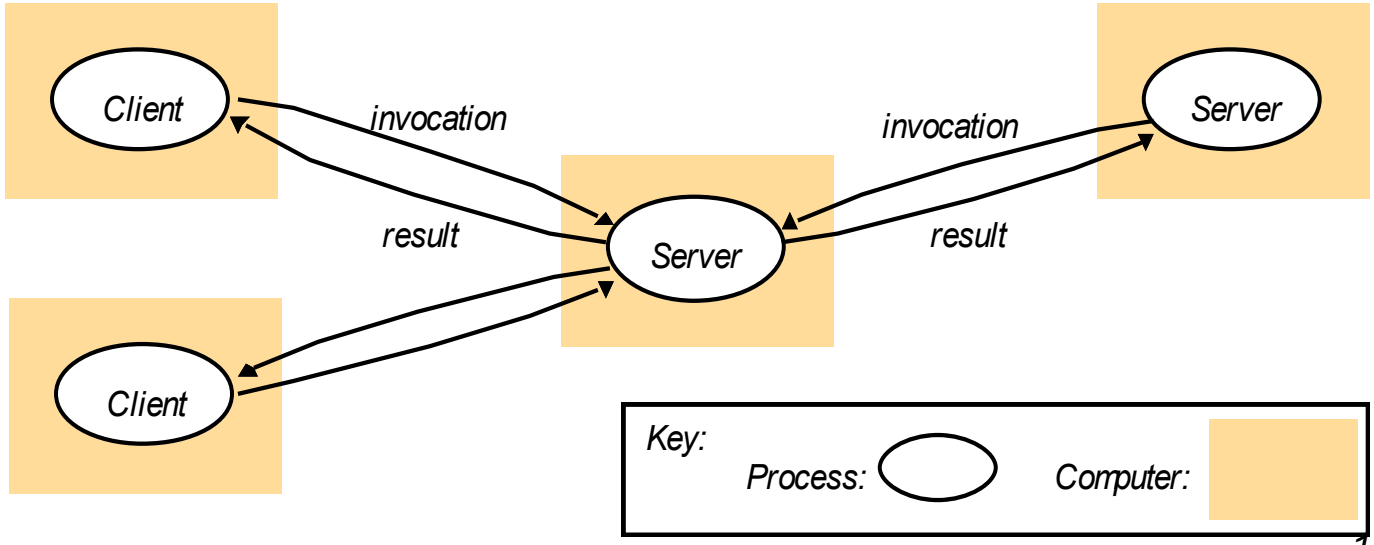
- *Client-Server model*
  - *Most often architecture for distributed systems.*
  - *Client process interact with individual server processes in a separate host computers in order to access the shared resources*



18

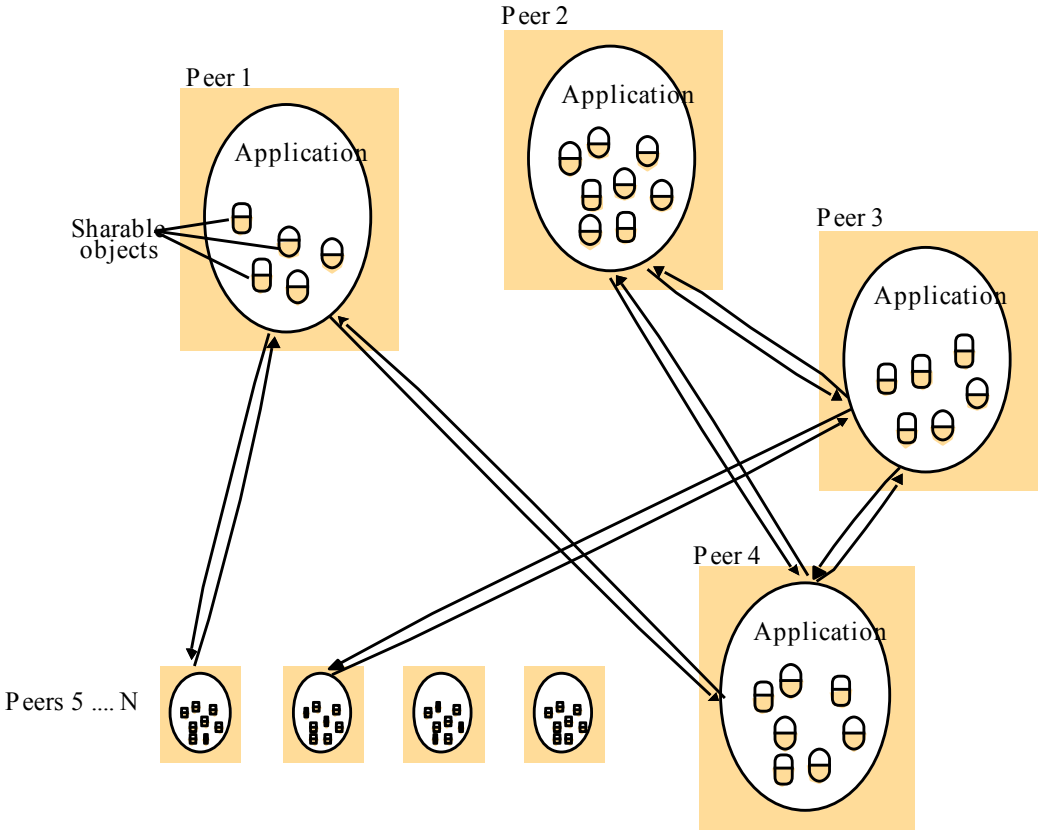
System Architectures

- Servers may in turn be clients of other servers.
  - E.g. a web server is often a client of a local file server that manages the files in which the web pages are stored.



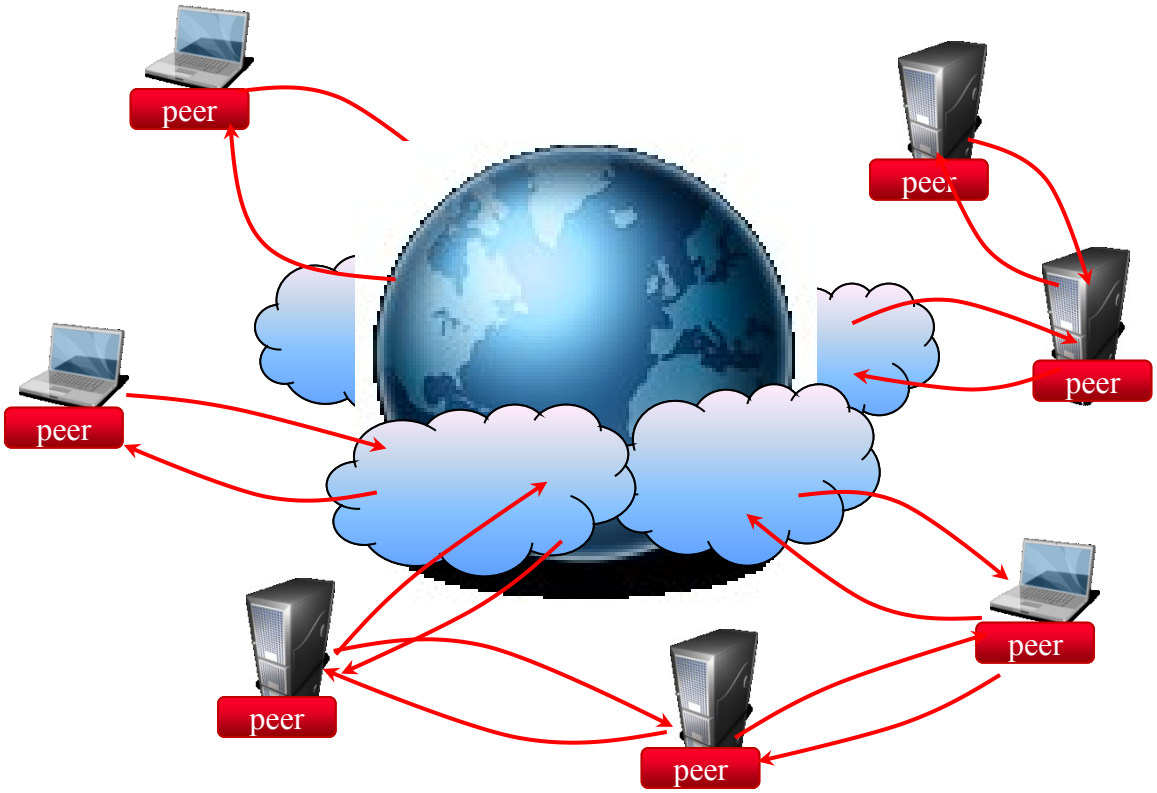
19

Peer Processes: A distributed application based on peer processes



- *All of the processes play similar roles, interacting cooperatively as peers to perform 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.*

P2P with a Centralized Index Server (e.g. Napster Architecture)

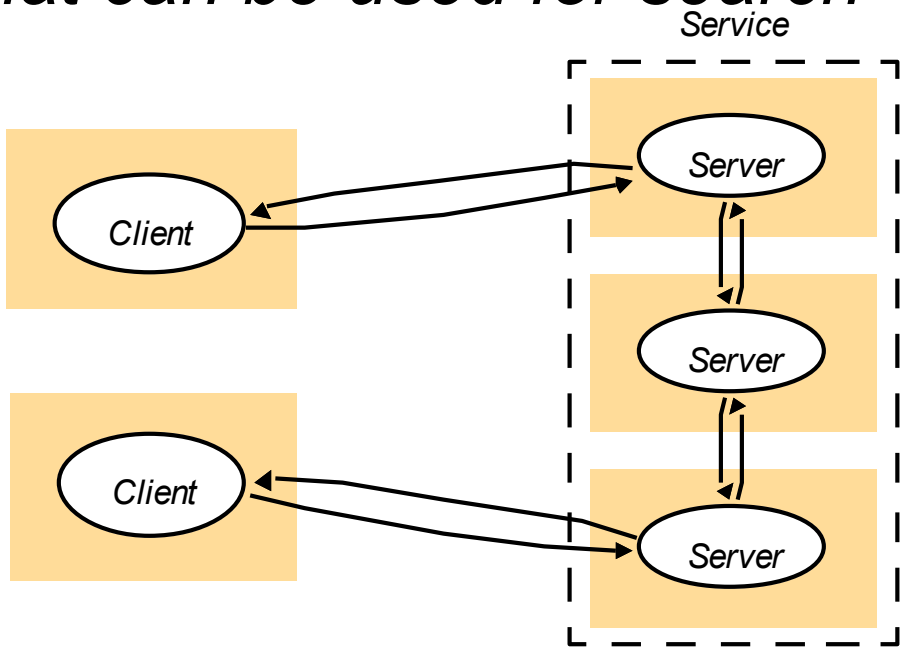


Placement

- *Placement is crucial in terms of determining the properties of the distributed system, most obviously with regard to performance but also to other aspects, such as reliability and security.*
  - *mapping of services to multiple servers;*
  - *caching;*
  - *mobile code;*
  - *mobile agents.*

Placement (2)

- *Services provided by multiple servers*
  - *Services may be implemented as several server processes in separate host computers interacting as necessary to provide a service to client processes.*
  - *E.g. cluster that can be used for search engines.*



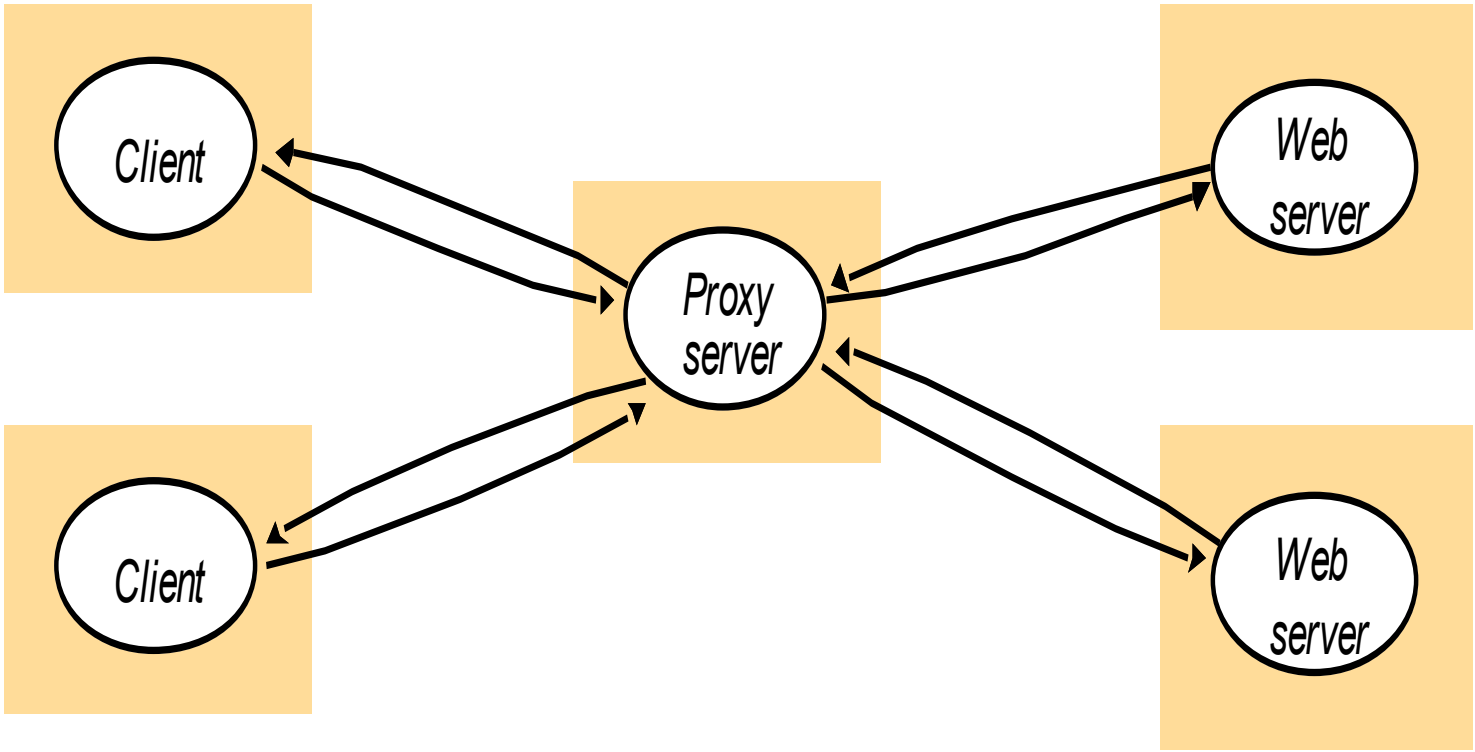
23

Placement(3)

- *Chaching*
  - *A cache is a store of recently used data objects.*
  - *When a new object is received at a computer it is added to the cache store, replacing some existing objects if necessary.*
  - *When an object is needed by a client process the caching service first checks the cache and supplies the object from there if an up-to-date copy is available.*
  - *If not, an up-to-data copy is fetched*

Placement(4)

- *Caches may be collected with each client or they may be located in a proxy server that can be shared by several clients.*

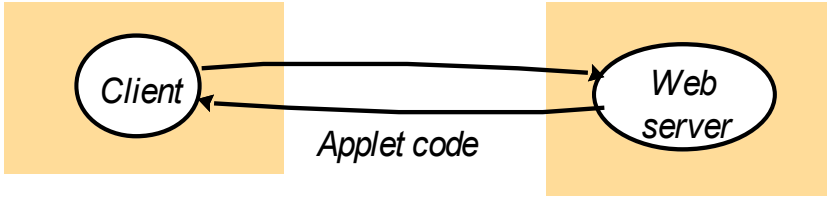


25

Placement(5)

- *Mobile code*
  - *Applets are a well-known and widely used example of mobile code.*
  - *Applets downloaded to clients give good interactive response*
  - *Mobile codes such as Applets are a potential security threat to the local resources in the destination computer.*

a) client request results in the downloading of applet code



b) client interacts with the applet

26

Placement(5)

- *Mobile Agent*
  - *A mobile agent is a running program (including both code and data) that travels from one computer to another in a network carrying out a task on someone's behalf, such as collecting information, and eventually returning with the results.*

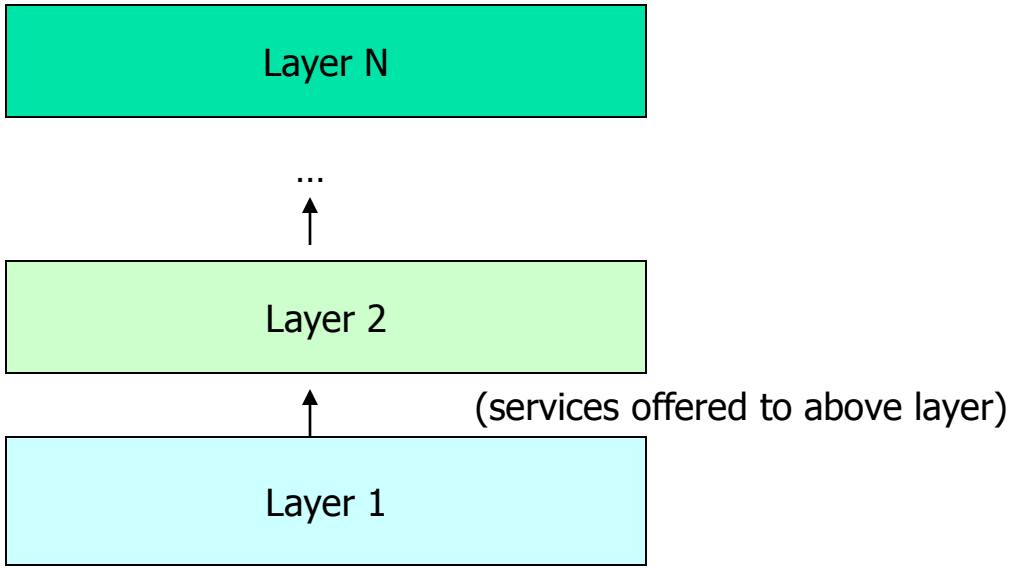
27

Architectural patterns

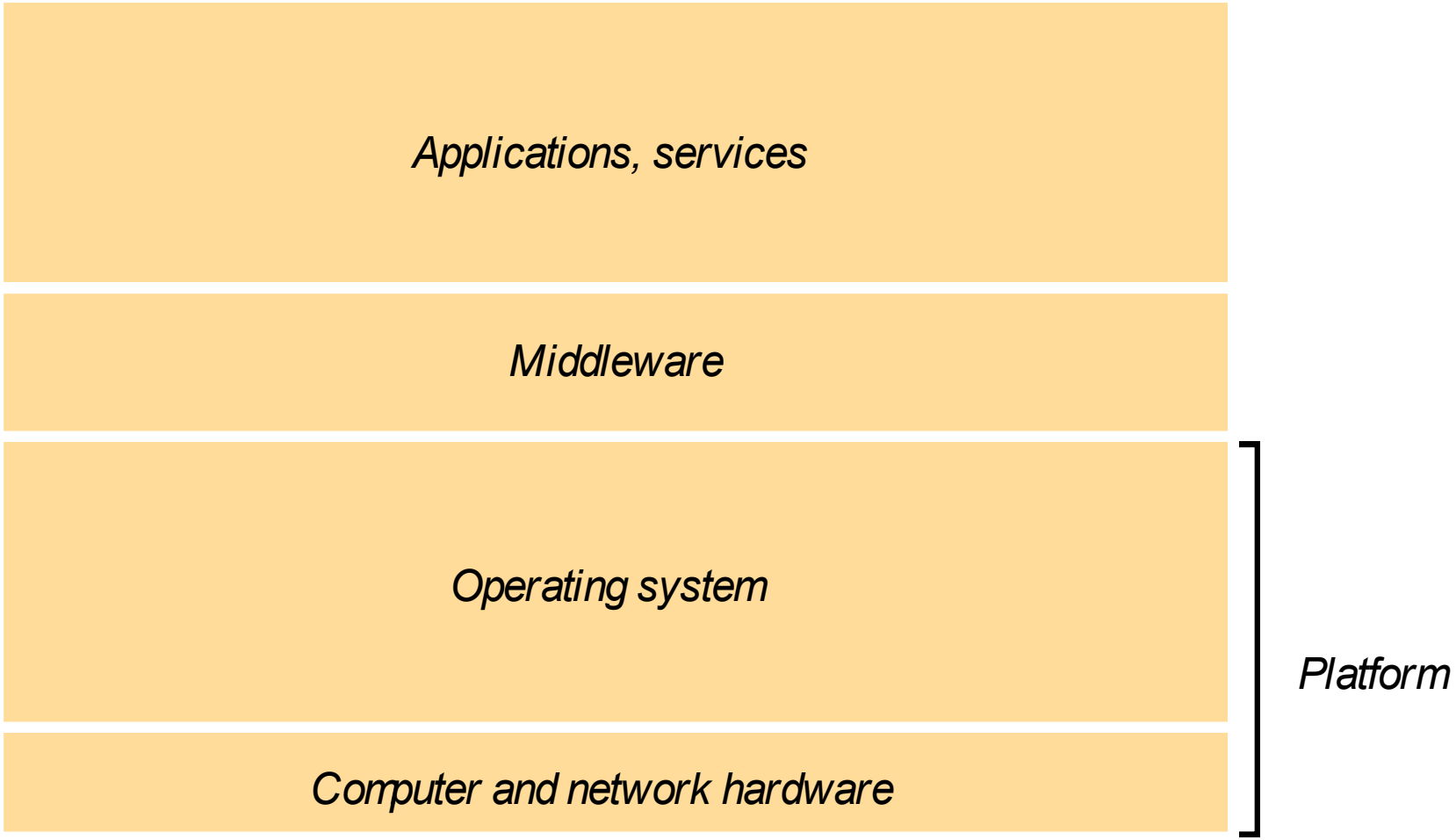
- **Layering**
- **Tiered architecture**
- **Thin clients**

Layering

- *The term software architecture referred:*
  - *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*
  - *PowerPC/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 Object Request Broker Architecture)*
  - *Microsoft D-COM (Distributed Components Object Model)*
  - *Sun Java RMI*
  - *Modern Middleware:*
    - *Manjrasoft Aneka– for Cloud computing*
    - *IBM WebSphere*
    - *Microsoft .NET*
    - *Sun J2EE*
    - *Google AppEngine*

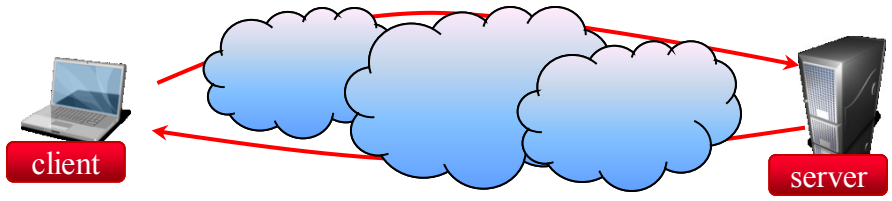
Tiered Architecture

- *Tiered architectures are complementary to layering*

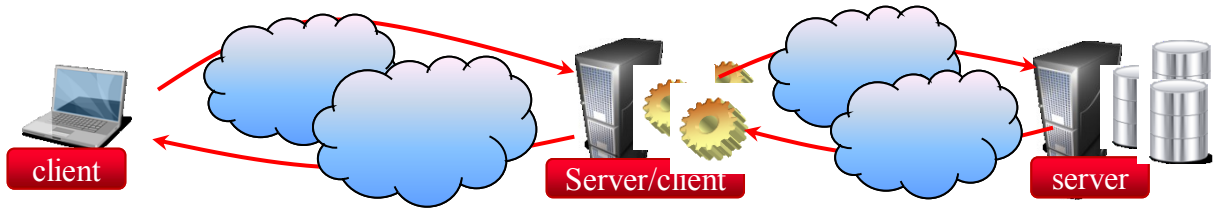
- *three-tiered architecture*
  - *the presentation logic*
  - *the application logic*
  - *the data logic*

Client-Server Architecture Types

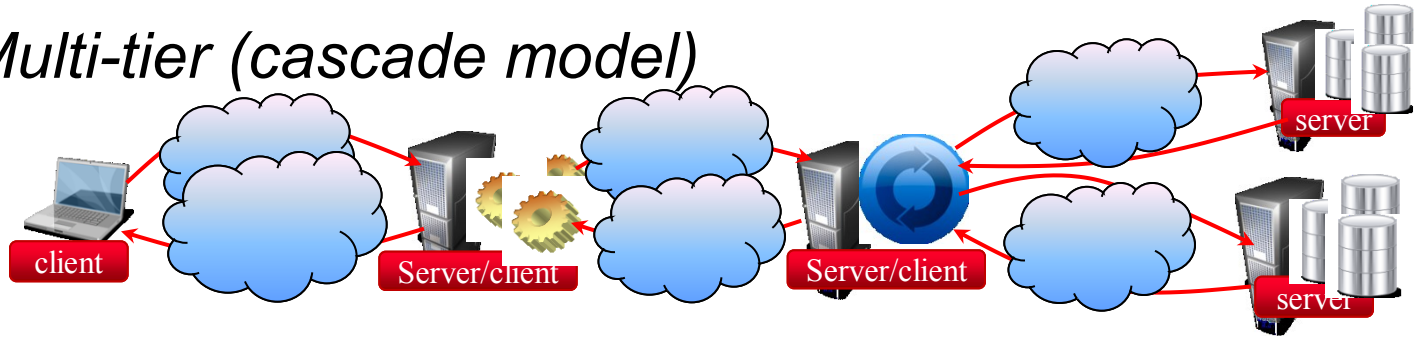
- *Two-tier model (classic)*



- *Three-tier (when the server, becomes a client)*



- *Multi-tier (cascade model)*



The Role of AJAX

- *AJAX is as an extension to the standard client-server style of interaction used in the World Wide Web*
- *The constraint of Standard HTTP*
  - *Once the browser has issued an HTTP request for a new web page, the user is unable to interact with the page until the new HTML content is received and presented by the browser*
  - *In order to update even a small part of the current page with additional data from the server, an entire new page must be requested and displayed*
  - *The contents of a page displayed at a client cannot be updated in response to changes in the application data held at the server*
- *AJAX enables Javascript front-end programs to request new data directly from server programs.*

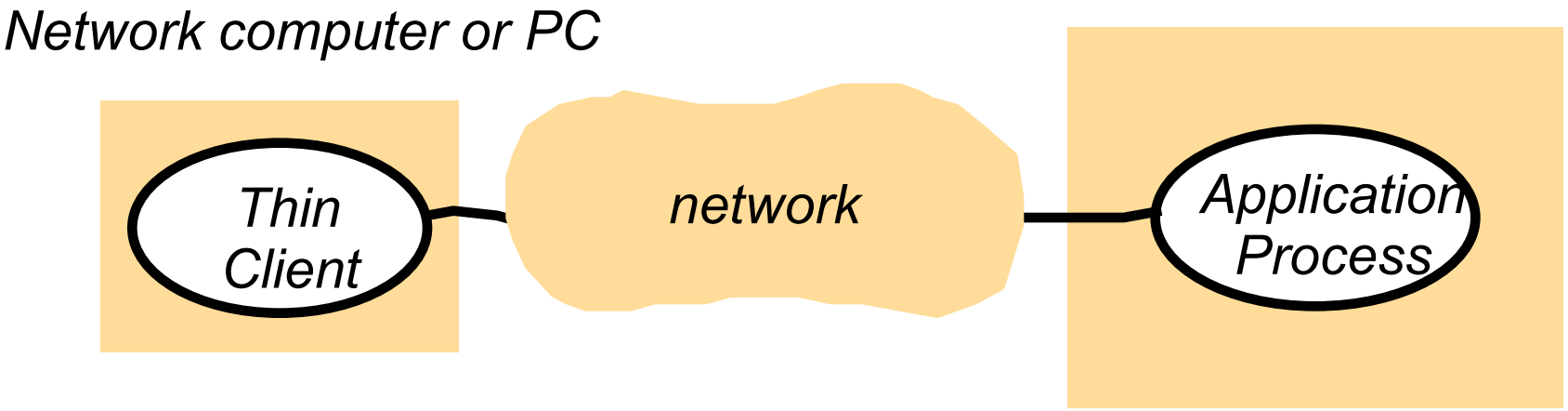
Thin Clients

- *Thin clients*
  - *It is a software layer that supports a window-based user interface on a computer that is local to the user while executing application programs on a remote computer.*
  - *This architecture has the same low management and hardware costs as the network computer scheme.*
  - *Instead of downloading the code of applications into the user's computer, it runs them on a compute server.*

36

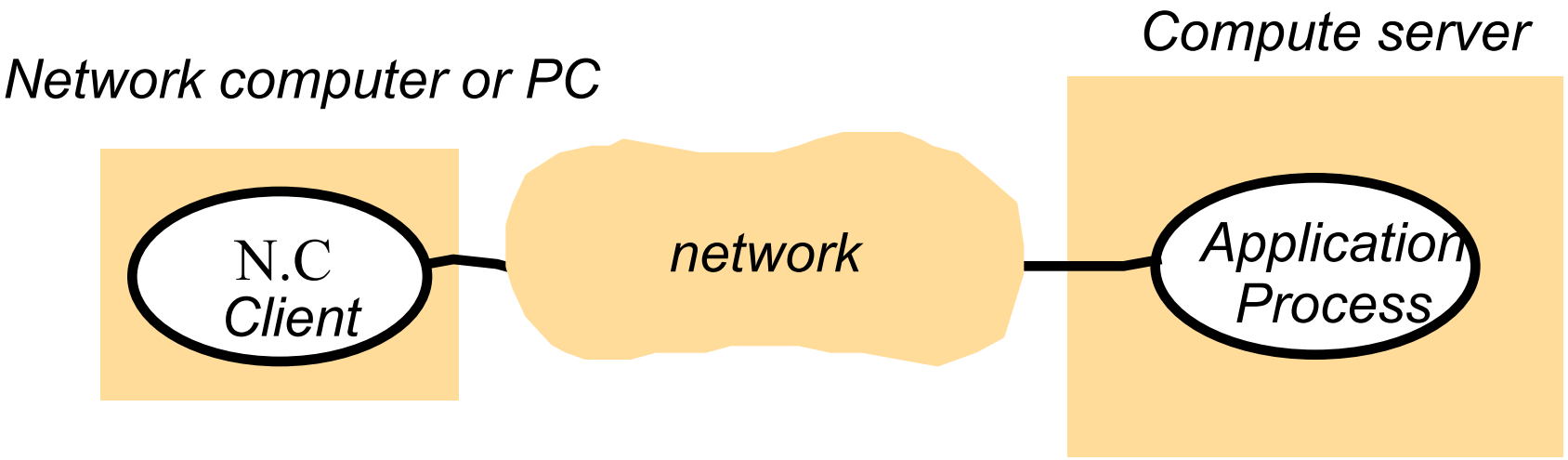
Thin Clients

- *Compute server is a powerful computer that has the capacity to run large numbers of application simultaneously.*
- *The compute server will be a multiprocessor or cluster computer running a multiprocessor version of an operation system such as UNIX or Windows.*



37

Network Computer vs. Thin clients



- *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.*

Presentation Outline

- *Introduction*
- *Architectural Models*
  - *Software Layers*
  - *System Architectures*
    - *Client-Server*
      - *Clients and a Single Sever, Multiple Servers, Proxy Servers with Caches, 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*

Fundamental Model

- *All architectural models are composed of processes that communicate with each other by sending messages over a computer networks.*
- *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:*
  - *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*
  - *Security 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

- *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 reception 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 bandwidth.*
  - *Jitter*
    - *The variation in the time taken to deliver a series of messages. It is very 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 can associate timestamp with their events.*
- *However, even if two processes read their clocks at the same time, their local clocks may supply different time.*
  - *This is because computer clock drifts 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 correct 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.*

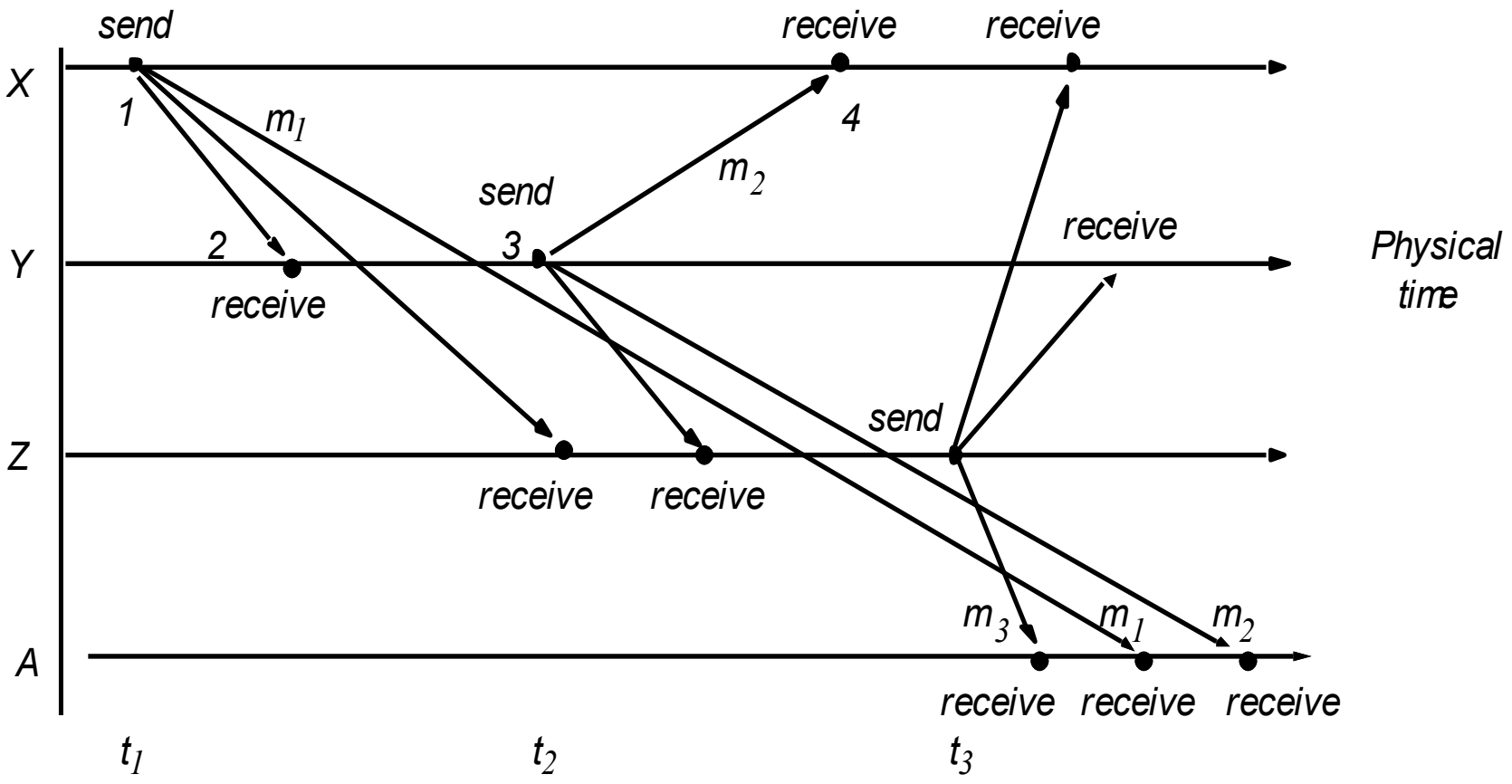
Interaction Model: Two variants of the interaction model

- *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 taken to execute a 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 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 other 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:

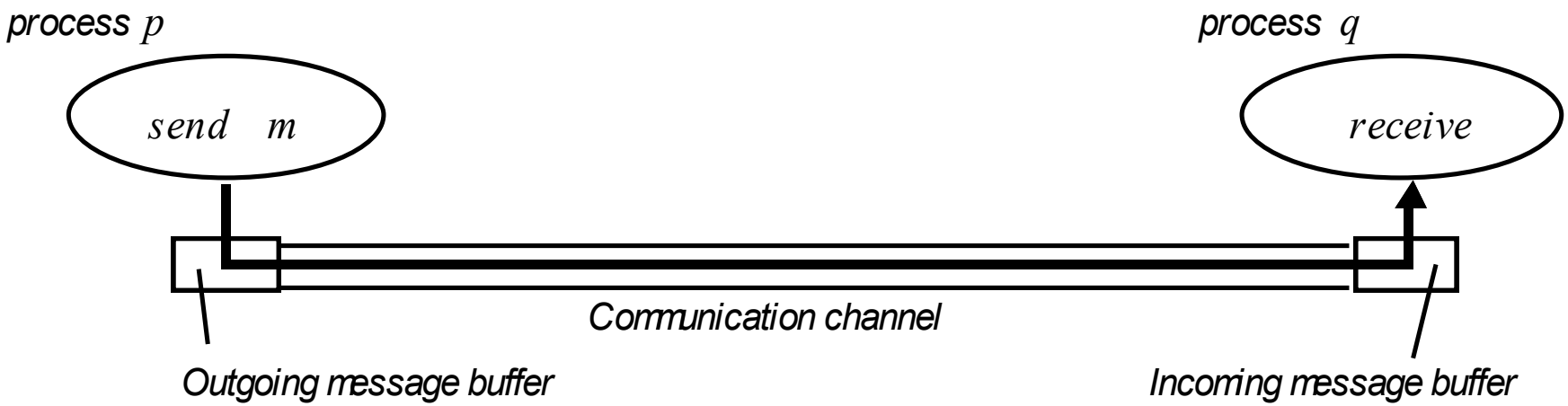
Item	*From*	*Subject*
23	*Z*	*Re: Meeting*
24	*X*	*Meeting*
26	*Y*	*Re: Meeting*

- *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 produces 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 a 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.

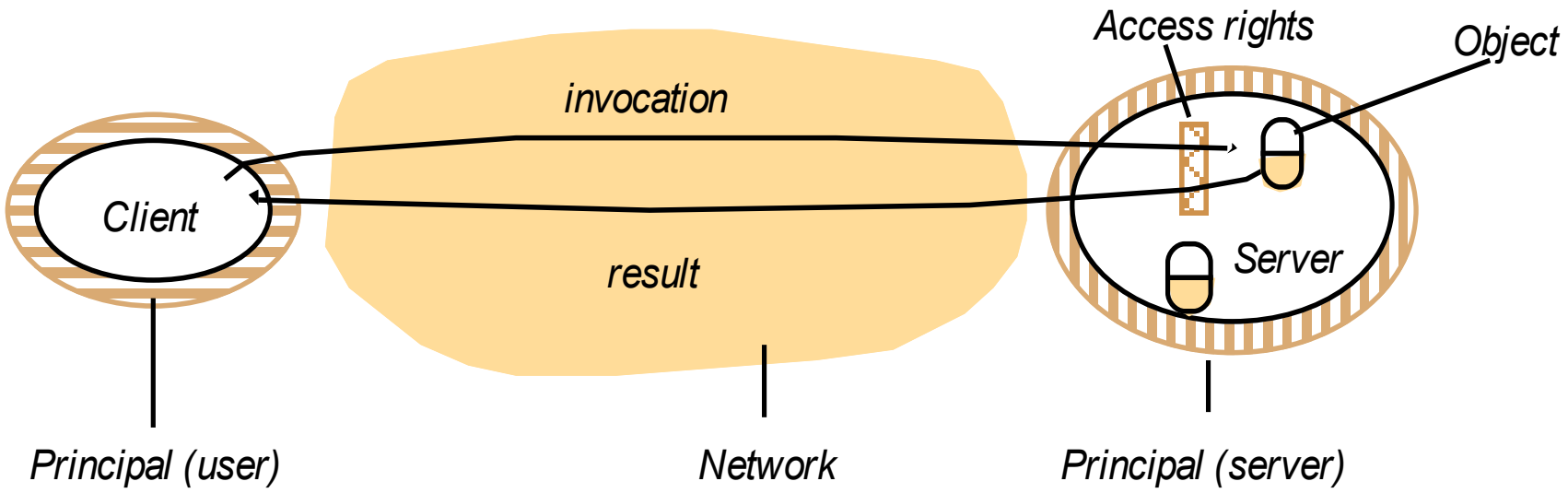
Masking Failures

- *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

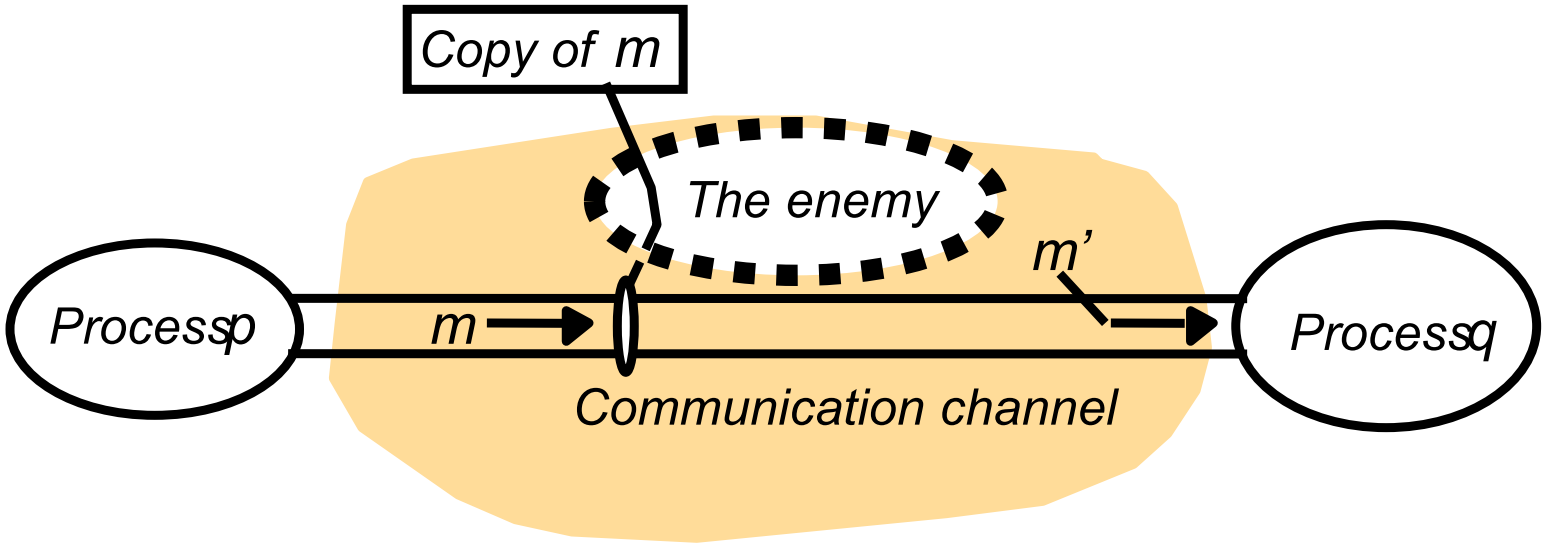
- *The security of a DS can be achieved by securing the processes and the channels used in their interactions and by protecting the objects that they encapsulate against unauthorized access.*

Protecting Objects: Objects and principals



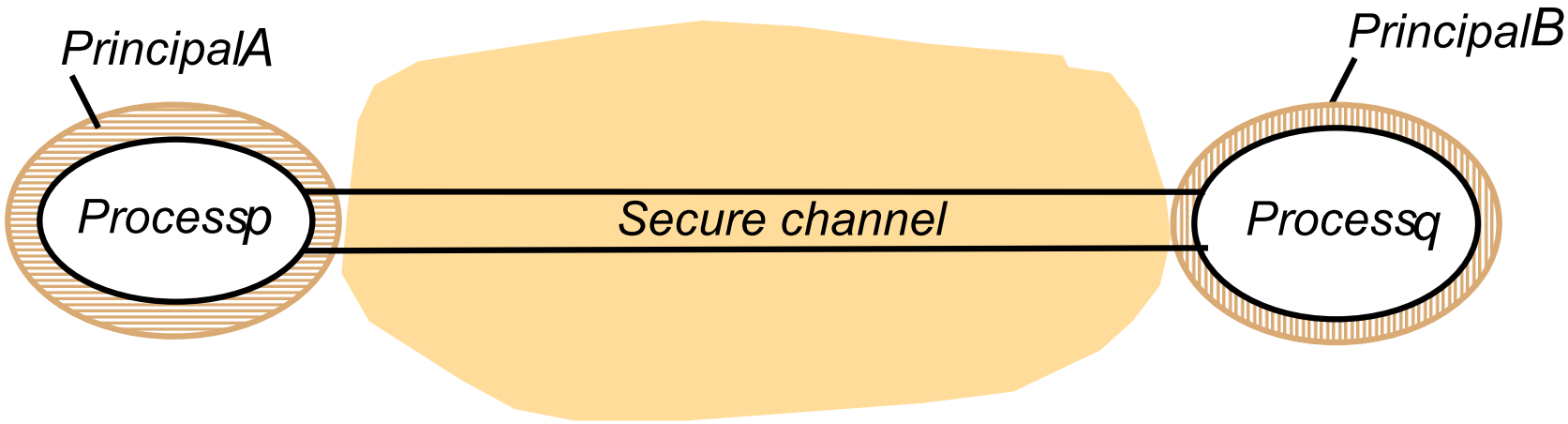
- *Use "access rights" that define who is allowed to perform operation on a object.*
- *The server should verify the identity of the principal (user) behind each operation and checking that they have sufficient access rights to perform the requested operation on the particular object, rejecting those 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*