COMP4057
Distributed and Cloud Computing
COMP7940
Cloud Computing

Chapter 02

System Models

Learning Outcomes

- Be able to describe a distributed system from the following three different but complementary models:
  - Physical models
  - Architecture models
  - Fundamental models
- Understand three important fundamental models
  - Interaction models
  - Failure models
  - Security models

Outline

- Motivation of developing different models
- Physical Models
  - Different generations of DSs
- Architectural Models
  - Architectural elements
  - Architectural patterns
  - Middleware solutions
- Fundamental Models
  - Interaction models
  - Failure models
  - Security models

Motivation

- Real world systems should be designed to function correctly in ALL circumstances.

- Distributed systems of different types share important underlying properties and give rise to common design problems.
  - Example: how is Google Map similar to and different from online stock quotation?

- Distributed system models help in
  - classifying and understanding different implementations;
  - identifying their weaknesses and their strengths;
  - crafting new systems outs of pre-validated building blocks.

Difficulties and Threats for DSs

- Widely varying models of use
  - High variation of workload (e.g., online shopping, Netflix (online video streaming), Baidu search engine), partial disconnection of components (e.g., WiFi), or poor connection.
- Wide range of system environments
  - Heterogeneous hardware, operating systems, network, and performance.
- Internal problems
  - Non synchronized clocks, conflicting data updates, various hardware and software failures.
- External threats
  - Attacks on data integrity, secrecy, and denial of service.

Dealing with Challenges

- Observations
  - —Widely varying models of use
    - The structure and the organization of systems allow for distribution of workloads, redundant services, and high availability.
  - —Wide range of system environments
    - A flexible and modular structure allows for implementing different solutions for different hardware, OS, and networks.
      - E.g., through layered architecture
  - —Internal problems
    - The relationship between components and the patterns of interaction can resolve concurrency issues, while structure and organization of component can support failover mechanisms.
  - —External threats
    - Security has to be built into the infrastructure and it is fundamental for shaping the relationship between components.

I. Physical Models

- A physical model is a representation of the underlying **hardware** elements of a DS that abstracts away from specific details of the computer and networking technologies.

- Baseline physical model
  - An extensible set of computer nodes interconnected by a computer network for the required passing of messages

Three Generations of DSs

- Early distributed systems [late 70-80s]
  - LAN-based
  - To shared printer, file servers, etc.
  - Email, file transfer (FTP)

- Internet-scale distributed systems [early 90-2005]
  - Emphasis on open standards, web services
  - Web, clusters, grids, peer-to-peer
  - Desktop computers or notebooks, relatively static for extended periods

- Contemporary distributed systems
  - mobile computing: dynamic nodes like mobile-based services (nodes are very dynamic not static like other models).
    - Need for added capabilities: service discovery, support for spontaneous interoperation.
  - cloud computing
  - Internet of things
    - washing machines, refrigerators, TVs, coffee makers, …

Three Generations of DSs

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

REF1 Fig. 2.1

II. Architectural Models

- The architecture of a system is its structure in terms of separately specified components and their interrelationships.
  - Its goal is to meet present and likely future demands.
  - Major concerns are to make the system reliable, manageable, adaptable, and cost-effective.

- To understand architectural model:
  1. Discuss underlying architectural elements;
  2. Examine typical composite architectural patterns.
  3. Consider middleware platforms to support the various styles of programming that emerge from these architectural styles.

1. Architectural Elements

- To understand the fundamental building blocks of a DS, four key questions need to be considered
  - What are the entities that are communicating in the DS?
  - How do they communicate, or, what communication paradigm is used?
  - What roles and responsibilities do they have in the overall architecture?
  - How are they mapped on to the physical distributed infrastructure, or, what is their placement?

1.1 Communication Entities

- From a system perspective, the entities that communicate in a DS are typically processes
  - In some primitive environments (e.g., sensor networks), the OS may not support process abstractions and the entities are nodes
  - In most environments, processes are supplemented by threads.
- From a programming perspective, more problem-oriented abstractions have been proposed.
  - Objects: to enable and encourage object-oriented approaches. A computation consists of a number of interacting and distributed objects.
  - Components: offer problem-oriented abstractions for building DSs; specify not only the interfaces but also the assumptions (in terms of other components/interfaces that must be present, i.e., dependencies) explicitly.
  - Web services:
    - Still based on encapsulation of behavior and accessed through interfaces.
    - Use web standards to represent and discover services.
    - Supports direct interactions with other software agents using XML-based message exchanges via Internet-based protocols, e.g., Simple Object Access Protocol (SOAP).

1.2 Communication Paradigms

- There are three typical communication paradigms
- (1) Inter-process communication: a relatively low-level support for communication between processes in DSs, such as socket programming, MPI (Message Passing Interface), and multicast communication.
- (2) Remote invocation: most common, based on two-way exchange between communicating entities
  - Request-reply protocols, pairwise exchange of messages, such as HTTP
  - RPC (Remote procedure calls): procedures in processes on remote computers can be called as if they are local
  - RMI (Remote method invocation): similar to RPC, but for distributed objects. A calling object can invoke a method in a remote object.

1.2 Communication Paradigms (Cont.)

- (3) Indirect Communication: the communication is indirect, e.g., via a third entity
  - Space uncoupling: senders do not need to know who they are sending to
  - Time uncoupling: senders and receivers do not need to exist at the same time
  - Key techniques include:
    - Group communication: deliver messages to a group of recipients represented by a group identifier. The sender does not need to know all recipients.
    - Publish-subscribe systems: a (large) number of publishers distribute information of interest to a large number of subscribers, e.g., financial trading.
      - Also called distributed event-based systems
      - An intermediary service efficiently ensures information generated by producers is routed to the consumers.
    - Message queues: producer processes can send messages to a specified queue, and consumer processes can receive messages from the queue
    - Distributed shared memory (DSM): DSM systems provide an abstraction for sharing data between processes that do not share physical memory. Programmers are presented with a familiar abstraction of reading or writing (shared) data structures as if they were in their own local address space.

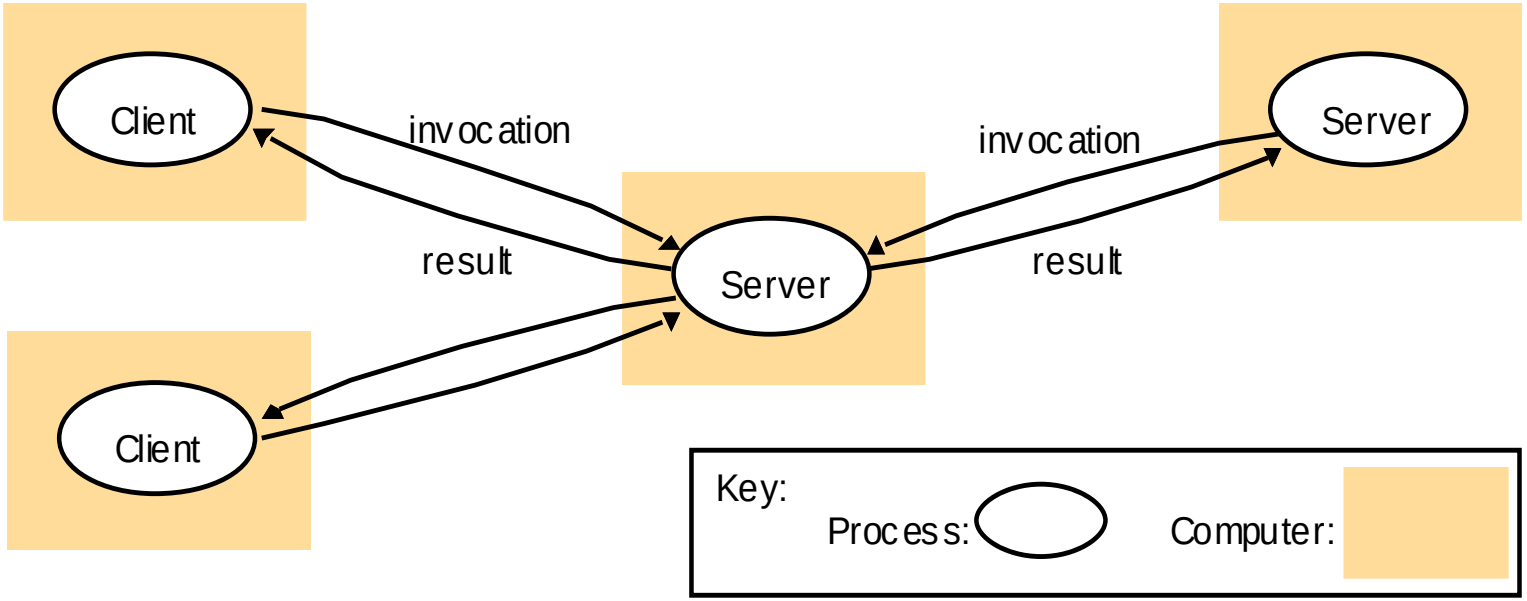
1.3 Roles and Responsibilities

- In a distributed system, processes / objects / components / services interact with each other and take different roles and responsibilities.

- Two popular architectural styles
  - Client-server
  - Peer-to-peer

Client-Server Model

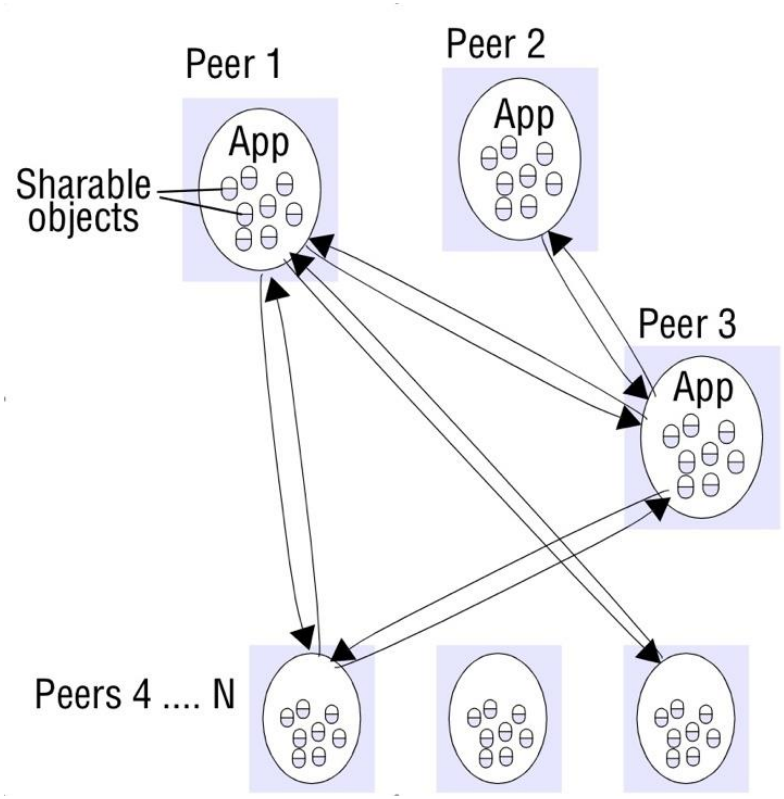
REF1 Fig 2.3: Clients invoke individual servers



- Client processes 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.
- Examples:
  - A Web Server is often a client of file server.
  - Browser → search engine → web crawlers → other web servers.
    - A web crawler runs in the background at a search engine site, using HTTP requests to access web servers throughout the Internet.

Peer-to-peer Model

- Network and computing resources owned by the users of a service could be used to support that service.
  - Good scalability
  - No single-point-of-failure
- All processes play similar roles, interacting cooperatively as peers without any distinction between "client" and "server".
- A large number of data objects are shared.
- An individual computer holds only a small part of the application database, and the storage, processing and communication loads are distributed across many computers and network links.
- Each object is replicated in several computers.



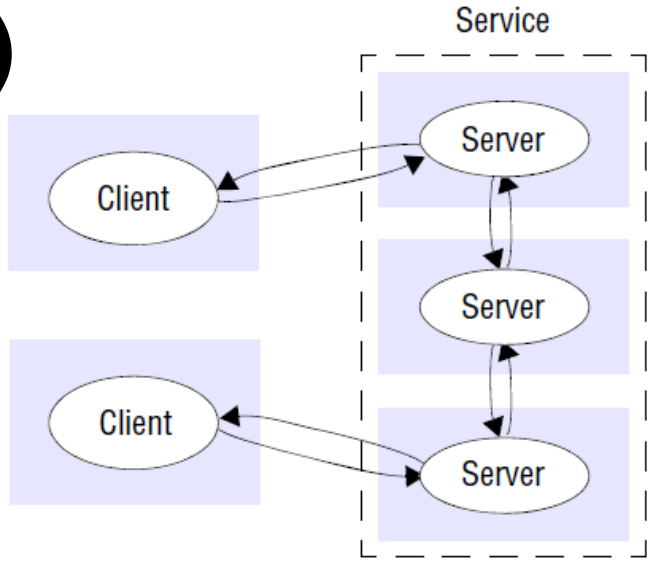
- Examples:
  - Media streaming applications such as PPLive, PPStream, QQLive
  - File sharing application such as BitTorrent

1.4 Placement

- How to map entities onto the underlying physical distributed infrastructure (i.e., many machines interconnected by networks)?
- Placement must be determined with strong application knowledge; few universal guidelines
- Placement design needs to consider the following factors:
  - Patterns of communication between entities
  - Quality of communication between machines
  - Reliability, capability, and loading of given machines
  - Economical concerns

Figure 2.4b A service provided by multiple servers

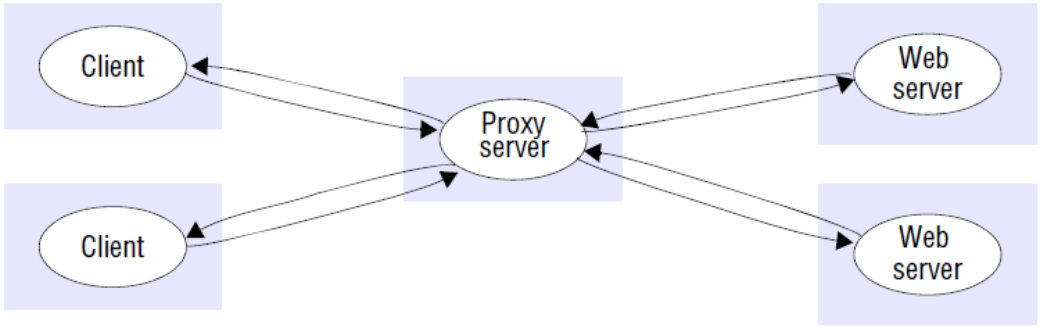
1.4 Placement (cont.)



- Example design: Mapping of services to multiple servers
  - Several server processes in separate computers interact as necessary to provide a service to client processes.
  - The servers may partition the set of objects on which the service is based and distribute these objects among themselves.
    - E.g., common Web falls into this category. Each web server manages its own resources.
  - The servers may also maintain replicated copies of the objects.
    - E.g., the Sun Network Information Service (NIS) enables all computers on a LAN to access the same user authentication data.
    - Each NIS server has its own replica of a common password file, containing login names and encrypted passwords.
    - E.g., the Windows Update website has load balancers to direct traffic to many file servers.

1.4 Placement (cont.)

- Example design: caching
  - A cache is a store of recently used data objects.
  - When a new object is received from a server, it is added to the local 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-date copy is fetched.
  - Caches may locate within each client, or in a proxy server shared by a few clients.
  - For security reasons, a company may locate a proxy server as a frontend for other hidden servers. Attacks are directed to the proxy server only.



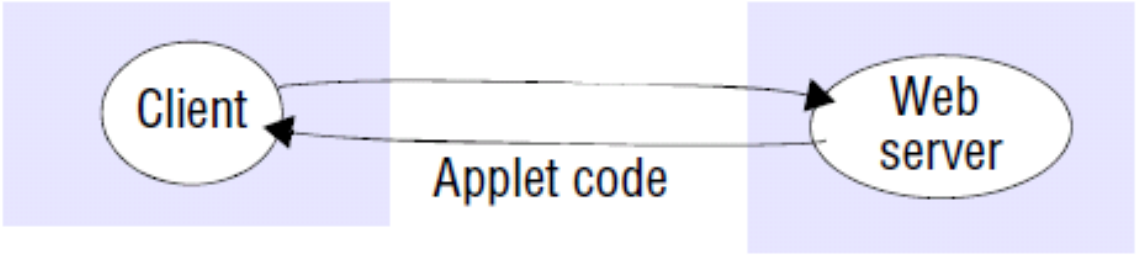
1.4 Placement (cont.)

- Example design: mobile code
  - Java applets are widely used mobile code.
  - A user downloads an applet via a link and runs the applet within the browser.
  - Advantage: good interactive response, no network delay.
  - An applet may communicate with the server.
  - Applications such as stock price quotations may need to keep information up-to-date.
    - This cannot be achieved by asking the user to continue to initiate the interactions by clicking a button.
    - Solution: use an applet so that the server initiates the interactions by continue sending updated prices to the applet – often called the **push** model.
  - Mobile code is a potential security threat.

1.4 Placement (cont.)

Web applets

a) client request results in the downloading of applet code



b) client interacts with the applet

2. Architectural Patterns

- Architectural patterns build on the primitive architectural elements and provide composite recurring structures.

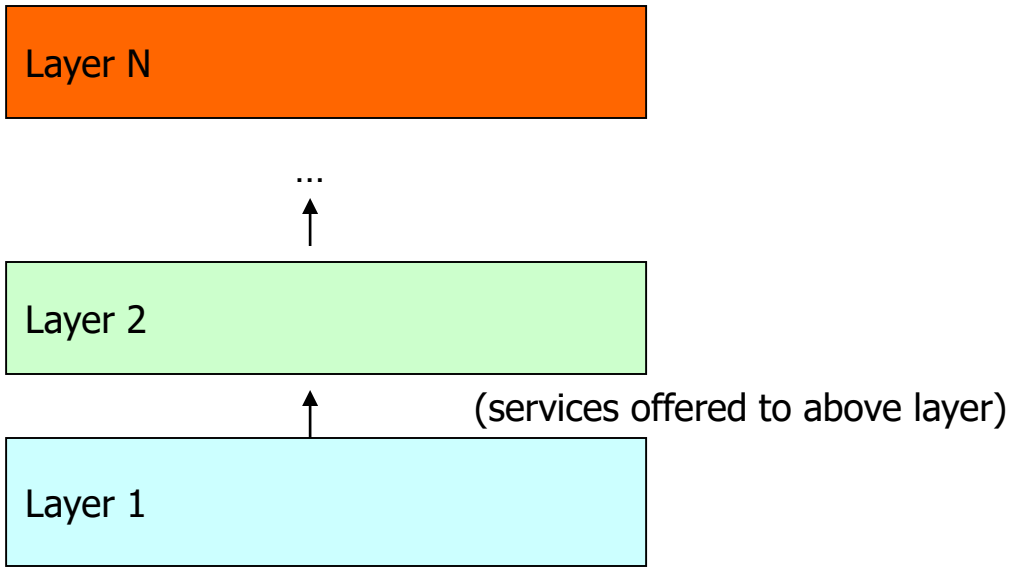
- We will study two key architectural patterns

  —Layering

  —Tiered architecture

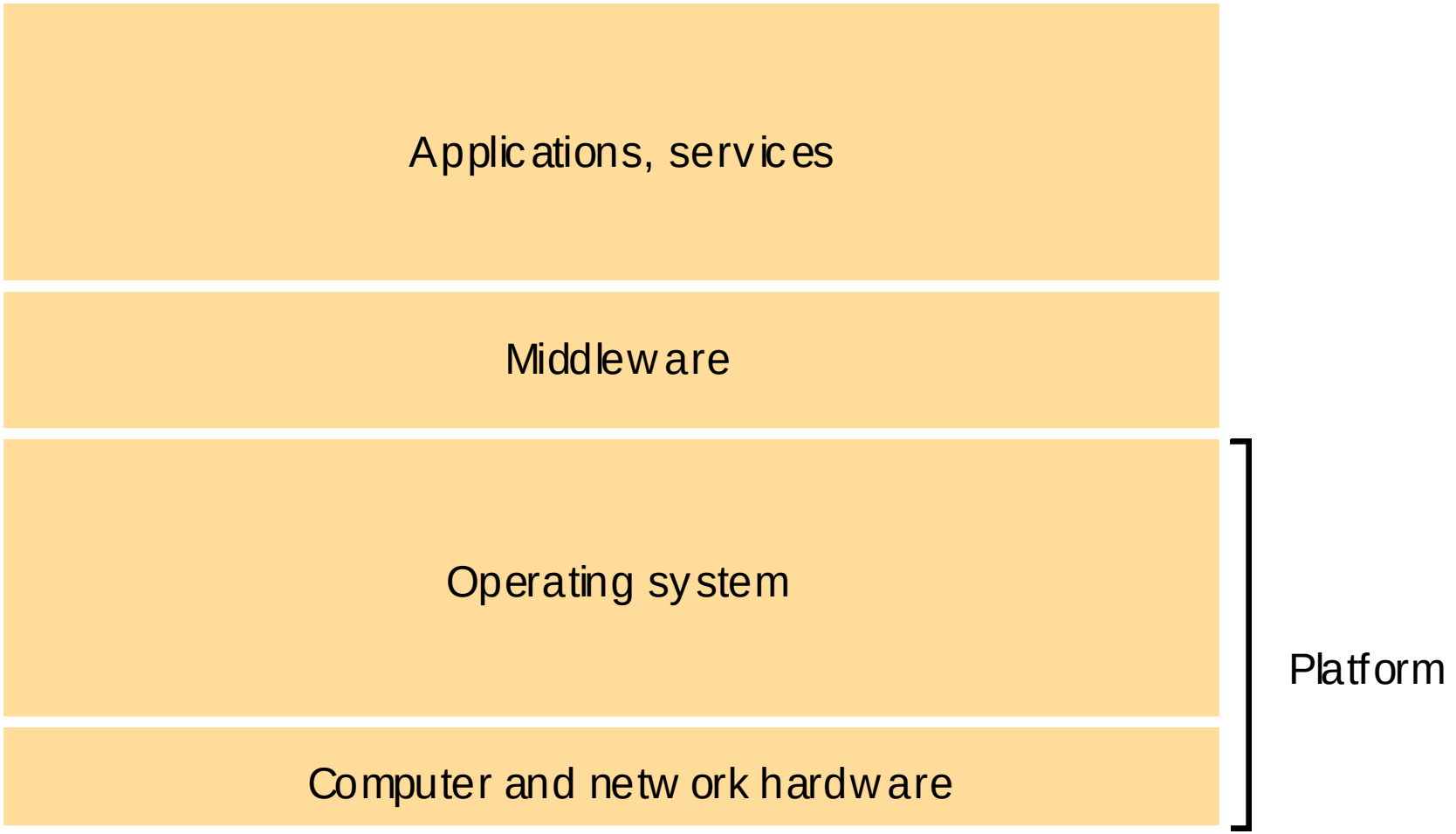
2.1 Layering Architecture

- 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.
    - Example: a network time service is implemented based on the Network Time Protocol (NTP) by server processes running on hosts throughout the Internet.
    - The server processes supply the current time to any client that requests it, and adjust their version of the current time by interactions with one another.



2.1 Software and hardware service layers

REF1 Fig. 2.7



2.1 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
  - Intel x86 / Mac OS
  - ARM / iOS

2.1 Middleware

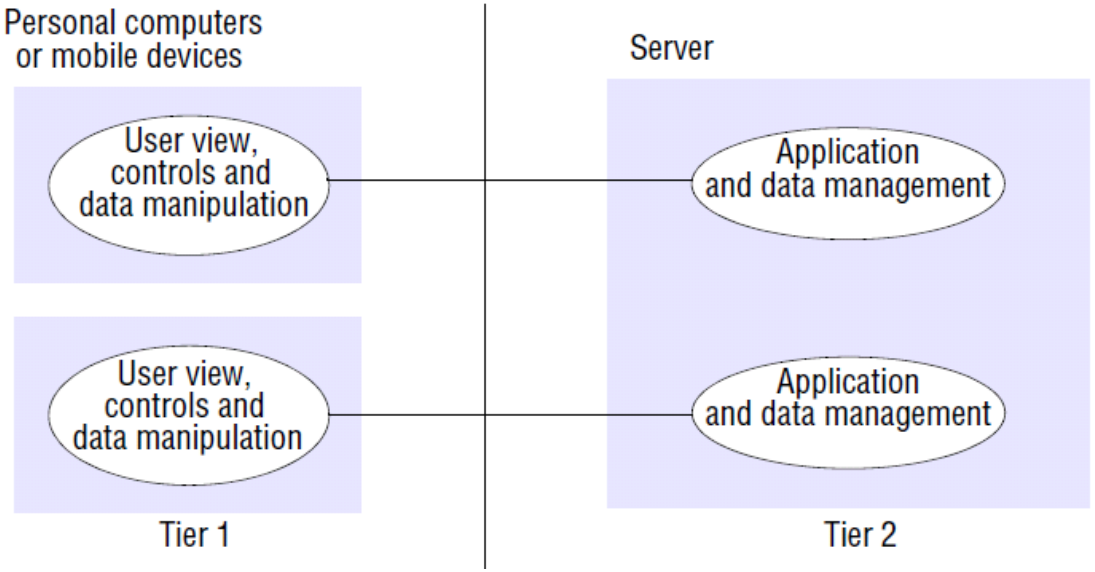
- A layer of software whose purpose is to mask heterogeneity present in distributed systems and to provide a convenient programming model to application developers.
- Middleware provides useful building blocks for the construction of software components (e.g., communication, resource-sharing support) that can work with one another in a DS.
- Major Examples:
  - Sun RPC (Remote Procedure Calls)
  - CORBA (Common Object Request Broker Architecture)
  - Microsoft D-COM (Distributed Components Object Model)
  - Sun Java RMI and other Java libraries
  - Modern Middleware:
    - IBM WebSphere
    - Microsoft .NET
    - Sun J2EE
    - Google App Engine

2.2 Tiered Architecture

- Tiering is complementary to layering.
  - It organizes functionality of a given layer and places this functionality into appropriate servers, and on to physical nodes.
  - Most commonly associated with the applications and services layer, but other layers are also possible.

- Consider the functional decomposition of a given application:
  - Presentation logic: handling user interaction and updating the view of the application as presented to the user;
  - Application logic: concerned with the detailed application-specific processing;
  - Data logic: concerned with the persistent storage of the application, such as a database management system
  - Examples: MS Word and Excel

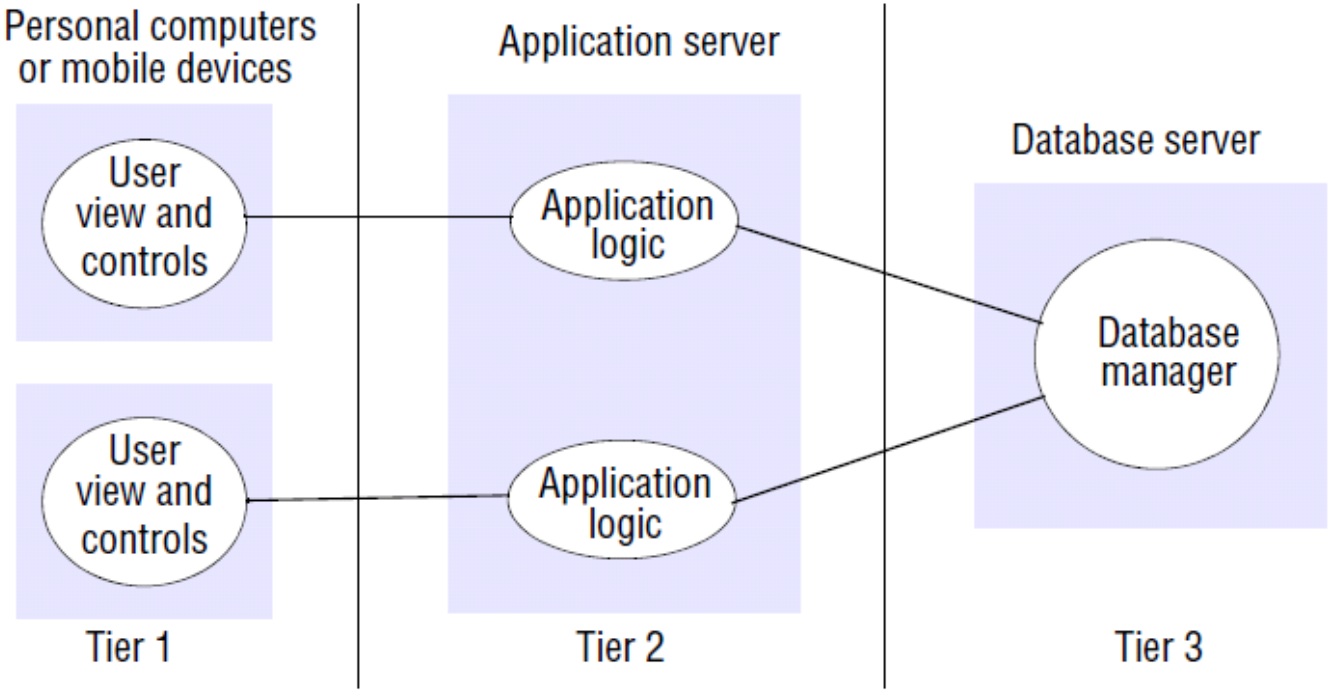
2.2 Two-tier Architecture



In the two-tier solution, the three aspects must be partitioned into two processes, the client and the server.

- The application logic is split between the client and server.
- Advantage: low interaction latency – one exchange of messages to invoke an operation
- Disadvantage: splitting of application logic across a process boundary, with the consequent restriction on which parts of the logic can be directly invoked from which other part.
- Example: generation of invoice in online shopping

2.2 Three-tier Architecture



In the three-tier solution, there is a one-to-one mapping from logical elements to physical elements.

- Advantage: each tier has a well-defined role.
- The first tier can be a simple user interface, allowing for a thin client.
- Disadvantage: added complexity of managing 3 servers; added network traffic and hence latency
- Example: online movie streaming

Case Study: WWW and AJAX

- World Wide Web (WWW) is an evolving system for publishing and accessing resources and services across the Internet.

- WWW is based on three main standard technological components:
  - HTML: HyperText Markup Language for specifying the contents and layout of pages
  - URL/URI: Uniform Resource Locator/Uniform Resource Identifier, for identifying documents and other resources stored as part of the Web
  - HTTP: HyperText Transfer Protocol, for defining interaction rules between browsers and Web servers

Case Study: WWW and AJAX (Cont.)

- Standard HTTP interaction has the following constraints:
  - Once the browser sends an HTTP request for a new Web page, the user is unable to interact with the page until the new HTML content is received. This time interval is indeterminate.
  - 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. This results in a delayed response, redundant network traffic, and additional processing at both client & server sides.
    - E.g., updating stock price within a page
  - The contents of a page displayed at a client cannot be updated in response to changes in the application data held at the server.

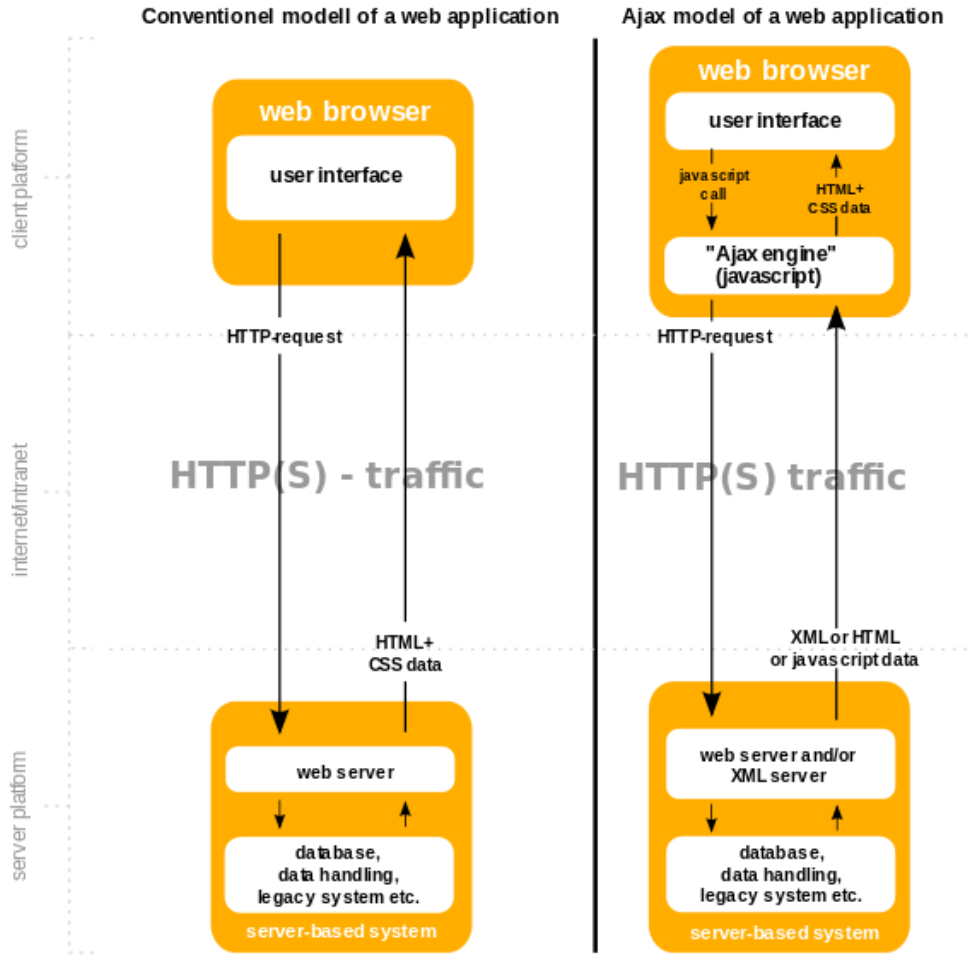
Case Study: WWW and AJAX (Cont.)

- Dynamic Pages by server side CGI program
  - Web users can fill out a Web form and submit the form to the Web server through HTTP.
  - The Web server runs a Common Gateway Interface (CGI) program to process the user's input, generate an HTML based on the user's input, and return it to the user through HTTP.

- Downloaded code (such as JavaScript and Java Applet)
  - For better-quality interaction, a Web browser can run a code downloaded from the Web server
  - E.g., a form with JavaScript can perform error checking on the browser side and improve the response time. Checking errors by the Web server incurs communication overhead.
    - Any experience in buying plane tickets?

Case Study: WWW and AJAX (Cont.)

- AJAX (Asynchronous JavaScript And XML) is a further innovative step to enable interactive Web applications.
  - It decouples data interchange layer from the presentation layer.
- It enables JavaScript front-end programs to request new data directly from back-end server programs.
  - The browser changes the display dynamically without reloading the entire page.



ACK: https://commons.wikimedia.org/wiki/File%3AAjax-vergleich.svg

Case Study: WWW and AJAX (Cont.)

AJAX example: soccer score updates

```
new Ajax.Request('scores.php?game=Arsenal:Liverpool',
      {onSuccess: updateScore});

function updateScore(request) {
.....
      ( request contains the state of the Ajax request including the returned result.
            The result is parsed to obtain some text giving the score, which is used
            to update the relevant portion of the current page.)
.....
  }
```

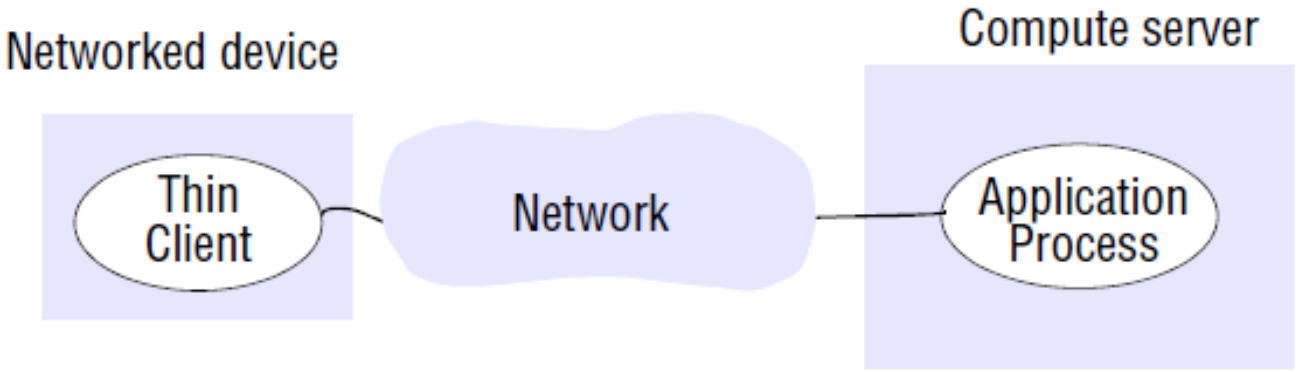
- An excerpt from a web application that displays a page listing up-to-date scores for soccer matches.
- Ajax.Request is an object that sends an HTTP request to a scores.php program located at the server.
- The Ajax.Request object then returns control, allowing the browser to continue to respond to other user actions. (This is asynchronous communication.)
- When the scores.php program obtains the latest score, it returns the score in an HTTP response.
- The Ajax.Request object is reactivated and invokes the updateScore() function to process and display the new score.

Case Study: WWW and AJAX (Cont.)

- The Google Maps application is another AJAX example.
- Maps are displayed as an array of 256 X 256 pixel images called tiles.
- When the map is moved, the visible tiles are repositioned by Javascript code in the browser.
- Additional tiles needed to fill the visible area are requested with an AJAX call to a Google server.
- They are displayed as soon as they are received.

Thin Client

- The concept of thin client:
  - —The trend in distributed computing is towards moving complexity away from the end-user devices.
    - notebooks, tablets, mobile phones, etc., with window-based user interfaces
- Application programs are executed remotely, e.g., in a cloud.



III. Fundamental Models

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

1. Interaction Model

- Computation occurs within processes;
- Multiple server processes may cooperate to perform a service, e.g.,
  - Data in the Domain Name System are partitioned and replicated throughout the Internet.
  - A video conferencing system distributes streams of audio data through multiple servers.
- The processes interact by passing messages, resulting in:
  - Communication (information flow)
  - Coordination (synchronization and ordering of activities) between processes.
- Each process has its own state (values of variables, data, registers, memory space, resources allocated, etc.) The state belonging to each process is completely private.
- 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.

Performance of Communication Channel

- The communication channel in our model is realized 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 reception by another.
    - Latency includes
      - Time taken for the first of a string of bits transmitted through a network to reach its destination.
      - The delay in accessing the network, which increases when the network is heavily loaded.
      - Time taken by operating systems at both sending and receiving computers.
  - Bandwidth of network:
    - 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.

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 timestamps with their events.
- However, even if two processes read their clocks at the same time, their local clocks may supply different time values.
  - 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 of about 1 microsecond.

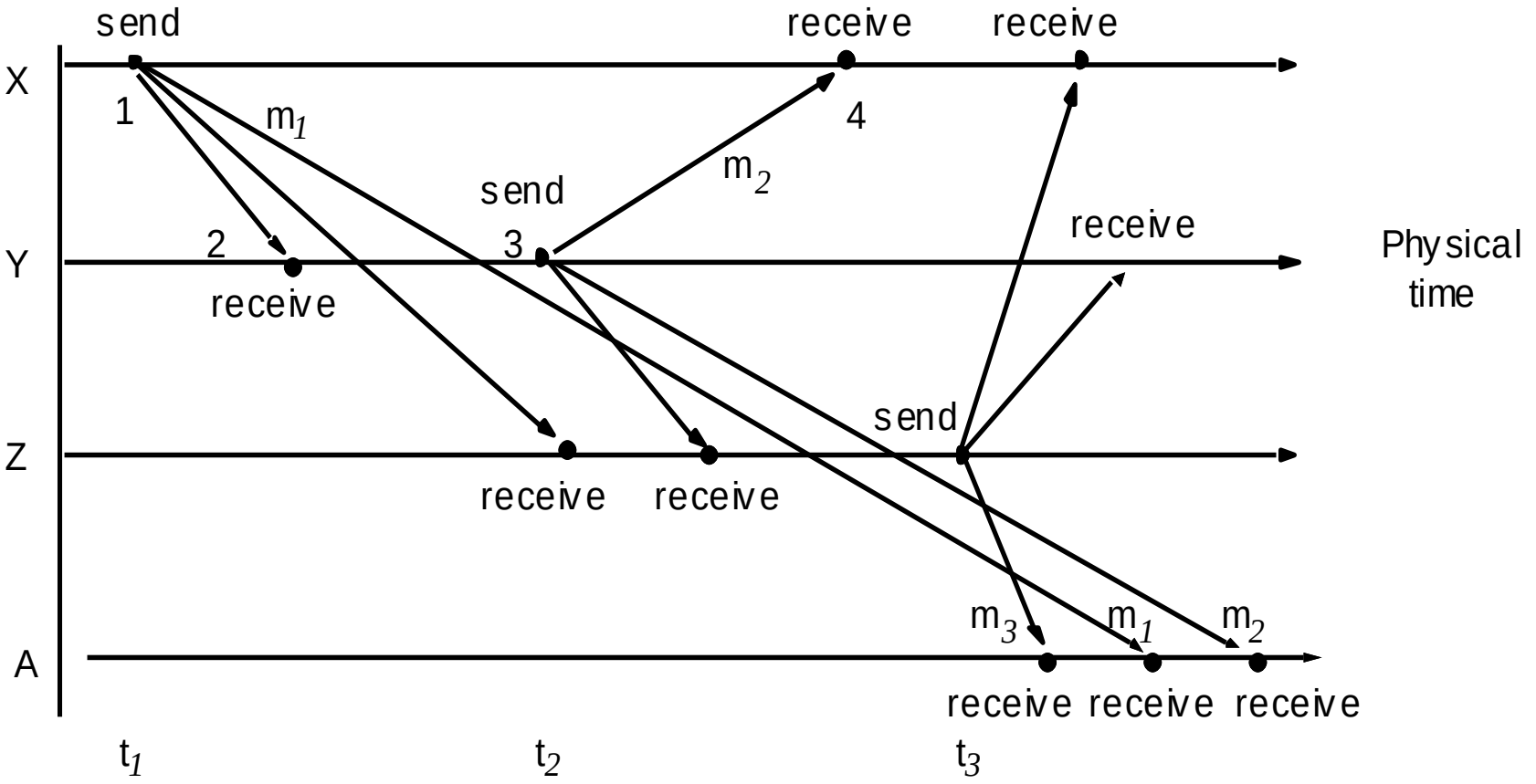
Synchronous and asynchronous Distributed Systems

- A synchronous distributed system defines these bounds:
  - 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 a known bound.
- Synchronous DS's are difficult to be built, but are used to model algorithms.
- An asynchronous distributed system has no bounds on:
  - Process execution speeds
  - Message transmission delays
  - Clock drift rates
- The internet is an asynchronous DS.

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.
- However, clocks cannot be synchronized perfectly across a distributed system – no guarantee that t1<t2<t3.

2. Failure Model

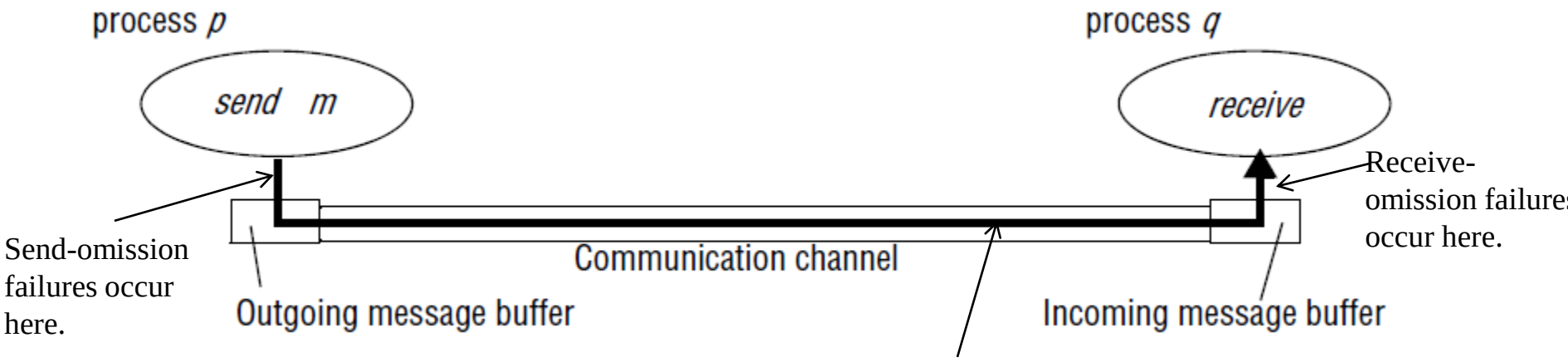
- In a DS, both processes and communication channels may fail – i.e., they may depart from what is considered to be correct or desirable behavior.
- Types of failures:
  - Omission Failure
  - Arbitrary Failure
  - Timing Failure

Omission Failure

- Omission failure occurs when a process or communication channel fails to perform actions that it is supposed to do.
  - Process omission failure
    - A process has halted and will not execute any further steps of its program.
    - In asynchronous DS's, this process failure is called a crash – other processes cannot detect the failure.
      - The other processes can only indicate that the suspicious process is not responding.
    - In synchronous DS's, this process failure is called a fail-stop – other processes can detect the failure by using timeouts.

Omission Failure (cont.)

- —Communication omission failures
  - A process p performs a *send* by inserting the message m in its outgoing message buffer.
  - The communication channel transports m to the incoming message buffer of process q.
  - Process q performs a *receive* by taking m from its incoming message buffer and delivering it.
  - The two buffers are typically provided by the OS.



Omission Failure (cont.)

- —Communication omission failures (cont.)
  - A 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 an intervening gateway, or by a network transmission error.
  - The loss of messages between the sending process and the outgoing message buffer is called send-omission failure.
  - The loss of messages between the incoming message buffer and the receiving process is called receive-omission failure.

Arbitrary Failure

- The term arbitrary or Byzantine failure is used to describe the worst possible failure semantics – any type of error may occur.
- A process
  - —may set wrong values in its data items;
  - —may return a wrong value to an invocation;
  - —may omit some processing steps or take unintended steps;
- A communication channel may transmit corrupted or nonexistent messages, or real messages may be delivered more than once.
  - —Usually handled by checksum or sequence numbers.

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

- Timing failures apply to synchronous DS's only.
- Real-time OS's are designed to provide timing guarantees, but they are complex to design.
- Timing is particularly relevant to multimedia computers with audio and video channels.

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.

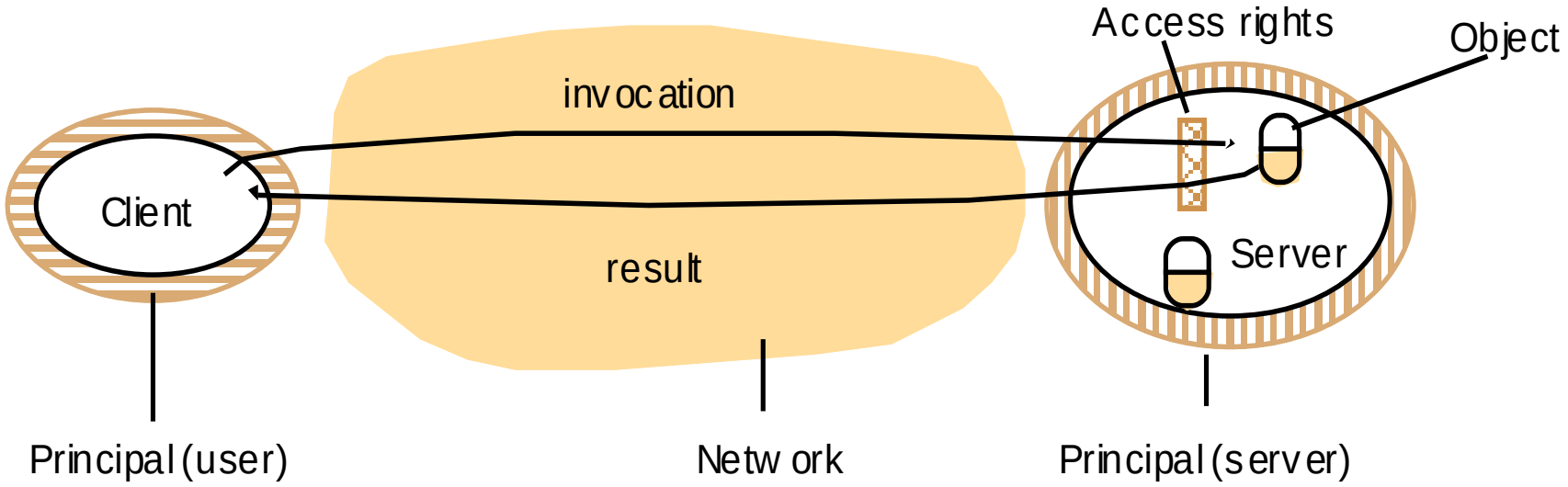
Masking Failures

- 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, converting an arbitrary failure into an omission failure.
  - A communication omission failure can be masked by retransmission of messages.
  - A process crash can be masked by replacing the process and restoring its memory state from information saved on disk at the last check point.

3. Security Model

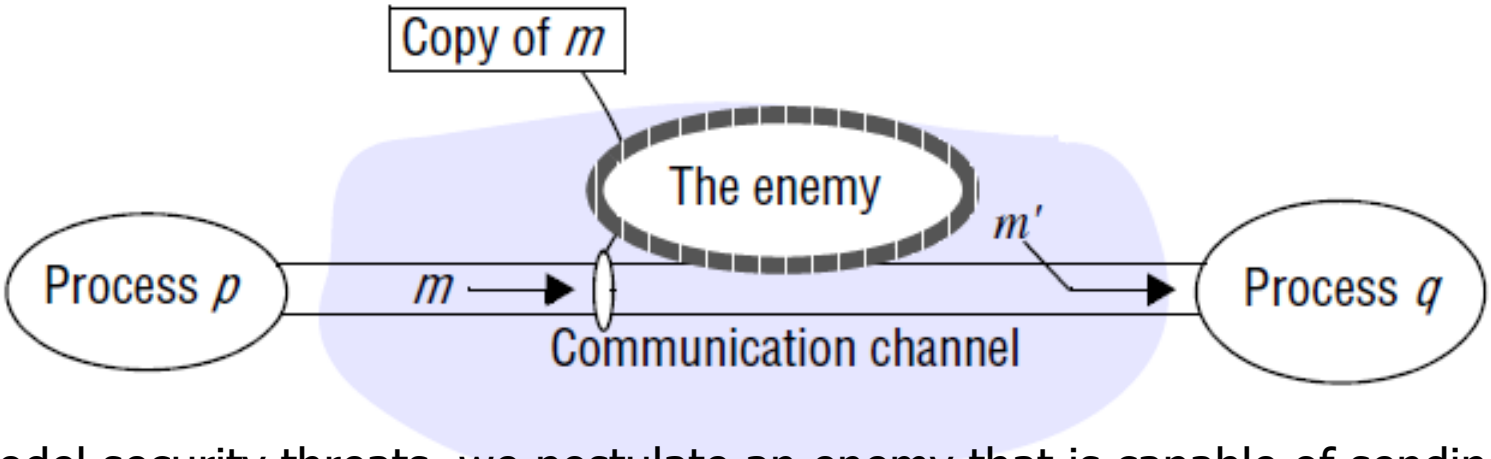
- The security of a DS can be achieved by securing the processes and the channels used in their interactions and by protecting the objects that they encapsulate against unauthorized access.

Protecting Objects: Objects and principals



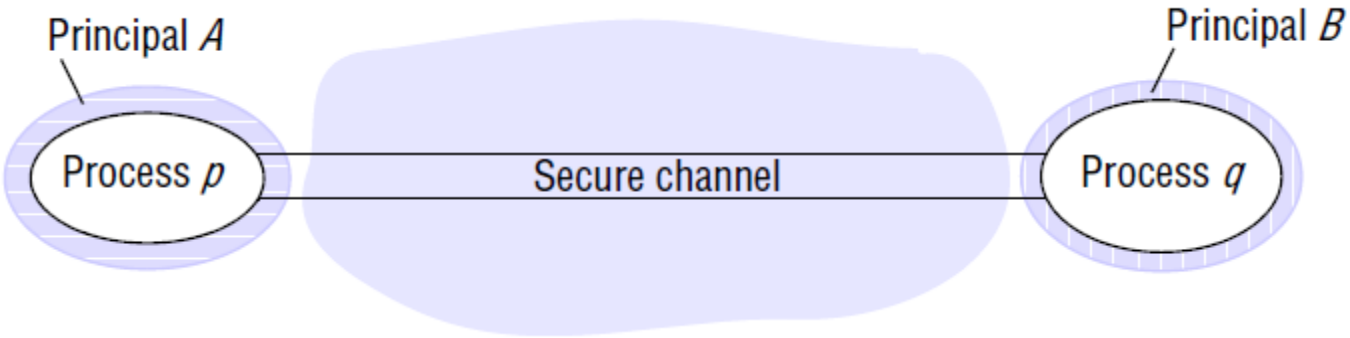
- Use "access rights" that define who is allowed to perform operation on a object.
- The server should verify the identity of the principal behind each operation and check that they have sufficient access rights to perform the requested operation on the particular object, rejecting those who do not.
  - A principal may be a user or a process.
- The client may check the identity of the principal behind the server.

The Enemy



- To model security threats, we postulate an enemy that is capable of sending any message to any process or reading/copying messages between a pair of processes
- Threats form a potential enemy:
  - Threats to processes
    - A server may receive a message from an enemy with a forged source address.
    - A client cannot tell whether the source of the result message is from the intended server or an enemy.
  - Threats to communication channels
    - An enemy can copy, alter, or inject messages as they travel across the network.
    - An enemy can save copies of messages and replay them later, e.g., resending a message requesting money transfer from a bank account.
  - Denial of service
    - Bombarding a server with many, possibly duplicated, requests.

Defeating Security Threats: Secure Channels



- Encryption and authentication are use to build secure channels.
- Each of the processes knows reliably the identity of the principal on whose behalf the other process is executing and can check their access rights before performing an operation.
- The client is sure that it is receiving results from a bona fide server.
- A secure channel ensures the privacy and integrity (protection against tampering) of the data being transmitted.
- Each message includes a physical or logical timestamp to prevent messages from being replayed or reordered.