CSF661 – Distributed Systems

分散式系統

Chapter 2
System Models

吳俊興

國立高雄大學 資訊工程學系

Chapter 2 System Models

2.1 Introduction

2.2 Physical models

2.3 Architectural models

2.4 Fundamental models

2.5 Summary

2.1 Introduction

- A physical model considers
  - the types of computers and devices that constitute a system
  - their interconnectivity
- An architectural model defines
  - the way in which the components of systems interact with one another and
  - the way in which they are mapped onto an underlying network of computers
- Fundamental models take an abstract perspective
  - in order to describe solutions to individual issues faced by most distributed systems
- Why model?
  - Each model is intended to provide an abstract, simplified but consistent description of a relevant aspect of distributed system design

2.2 Physical Models

- Baseline physical model: one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages
- Three generations
  - Early distributed systems: late 1970s and early 1980s
    - typical 10 and 100 nodes of LAN
  - Internet-scale distributed systems: 1990s
    - network of networks
  - Contemporary distributed systems: 2000s (cloud)
    - mobile devices and wireless networking
- Next-generation: distributed systems of systems?
  - 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

Figure 2.1
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

2.3 Architectural Models

- 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
  - Goal: simplifies and abstracts the functions of individual components
  - A three-stage approach
    - **Architectural elements**: looking at the core underlying architectural elements
    - **Architectural patterns**: examining composite architectural patterns that can be used in isolation or in combination in developing more sophisticated distributed systems solutions
    - **Associated middleware solutions**: considering middleware platforms that are available to support the various styles of programming

2.3.1 Architectural Elements

- Four key questions
  1. What are the ***entities*** that are communicating in the distributed system?
  2. How do they communicate or what ***communication*** paradigm is used?
  3. What ***roles and responsibilities*** do they have in the overall architecture?
  4. How are they ***mapped*** on to the physical distributed infrastructure? (What is their placement?)

Communication Entities and Paradigms

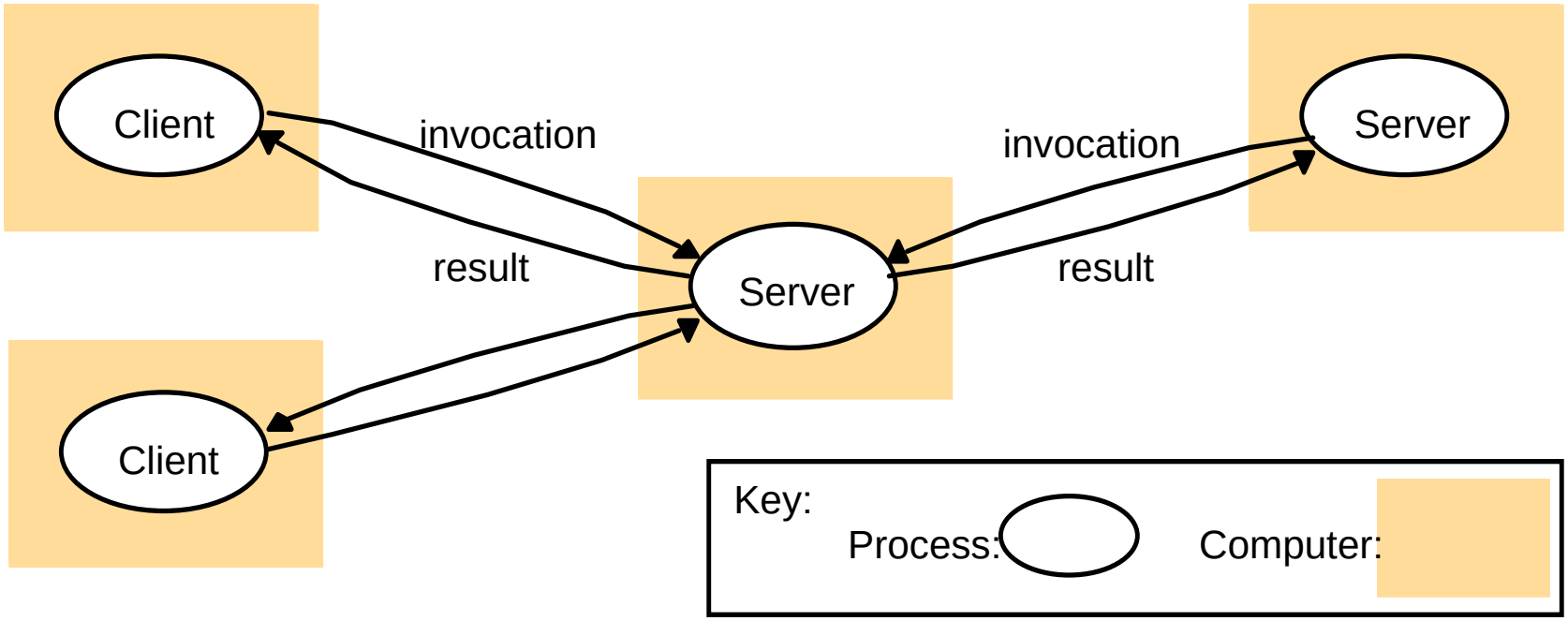
- Communicating entities
  - System-oriented perspective: processes
  - Problem-oriented perspective: objects, components, Web services

- Communication paradigms
  - Interprocess communication: RPC, request-reply protocols
  - Remote invocation: RMI
  - Indirect communication: group communication, publish-subscribe systems, message queues, tuple spaces, distributed shared memory

Figure 2.2 Communicating entities and communication paradigms

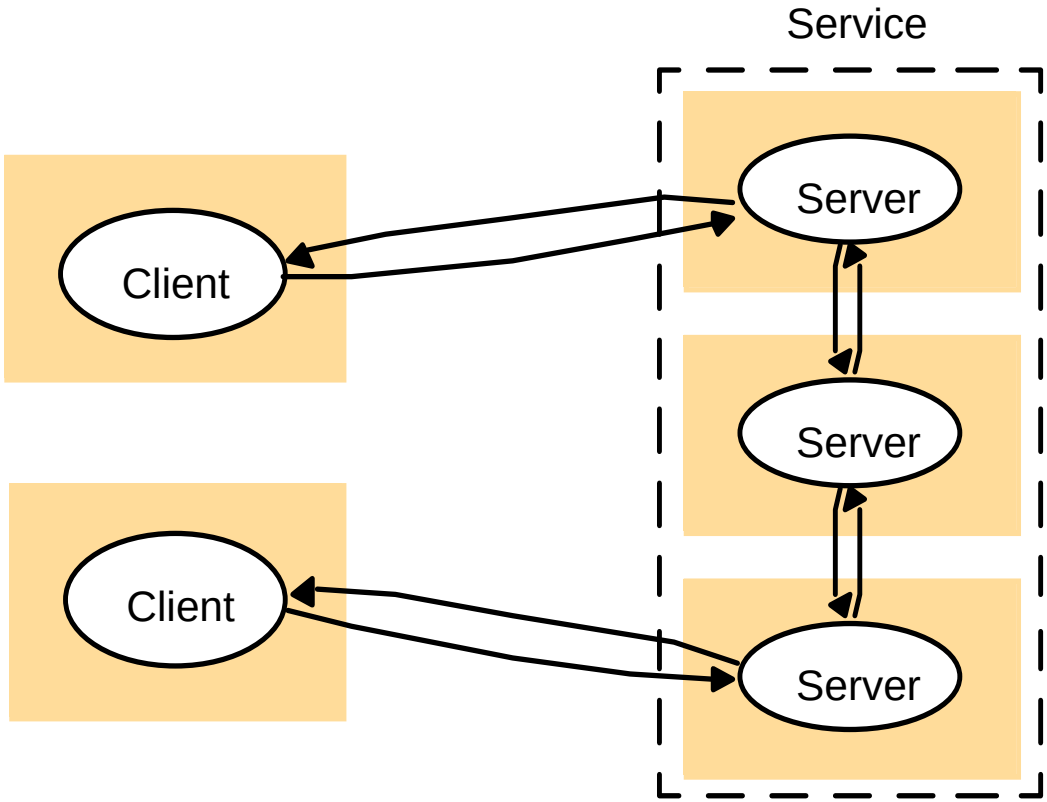
Communicating entities (what is communicating)		*Communication paradigms (how they communicate)*		
System-oriented entities	*Problem-oriented entities*	*Interprocess communication*	*Remote invocation*	*Indirect communication*
Nodes	Objects	Message passing	Request-reply	Group communication
Processes	Components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

Figure 2.3
Clients invoke individual servers



- Servers provide services
- Clients access services

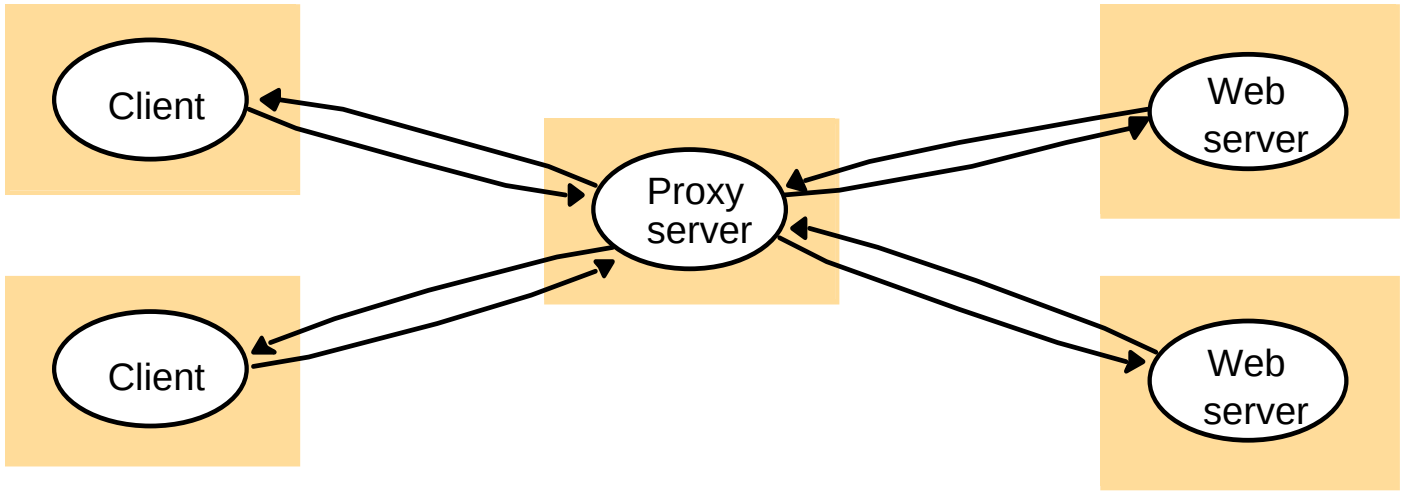
Figure 2.4 A service provided by multiple servers



- Objects are partitioned/replicated
  - Web: each server manages its objects
  - NIS: replicated login/password info
  - Cluster: closely coupled, scalable (search engines)

Figure 2.5 Web proxy server

- Cache: local copies of remote objects for faster access
- Browser cache
- Proxy server
  - Additional roles: filtering, firewall



Multiple Web proxy servers

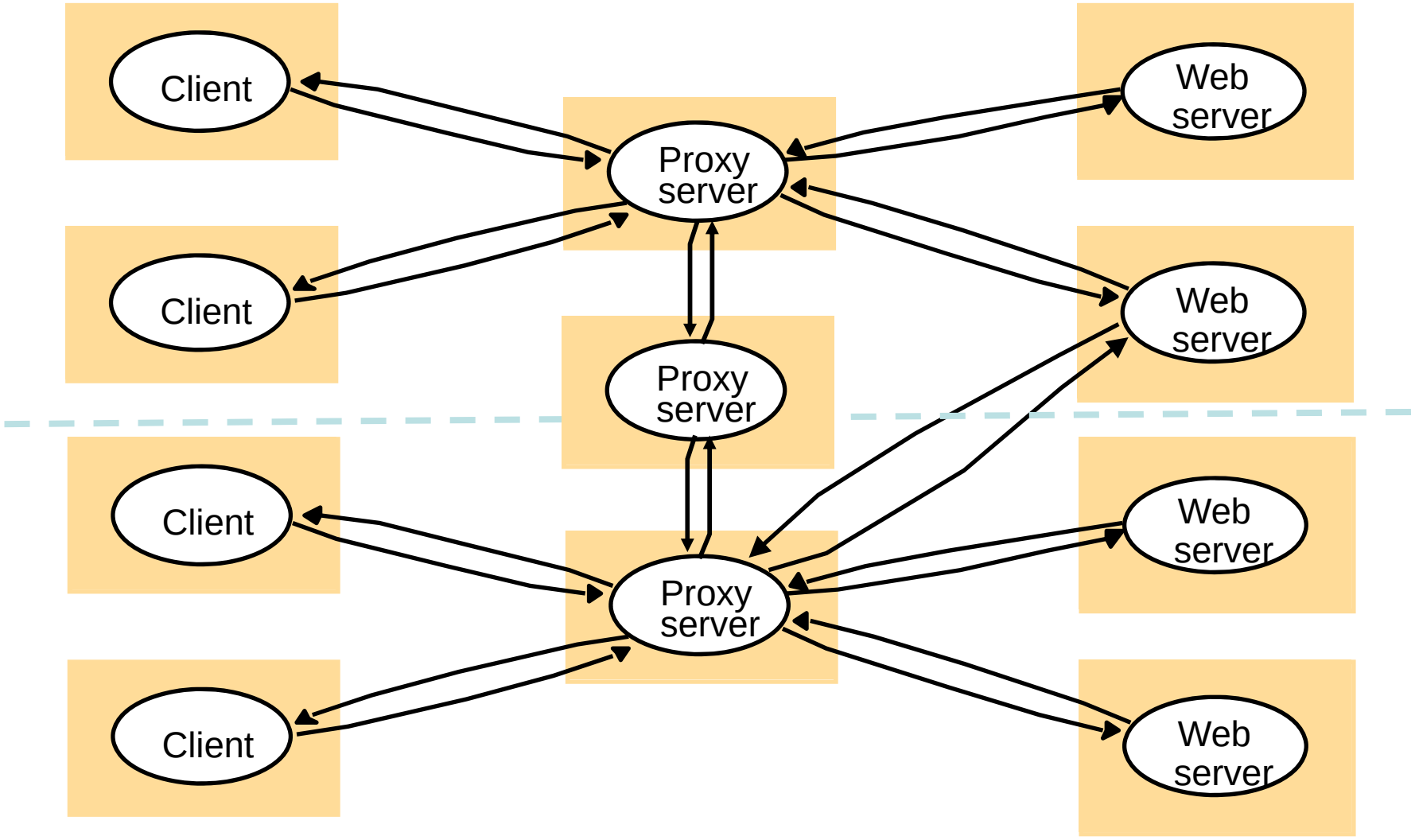
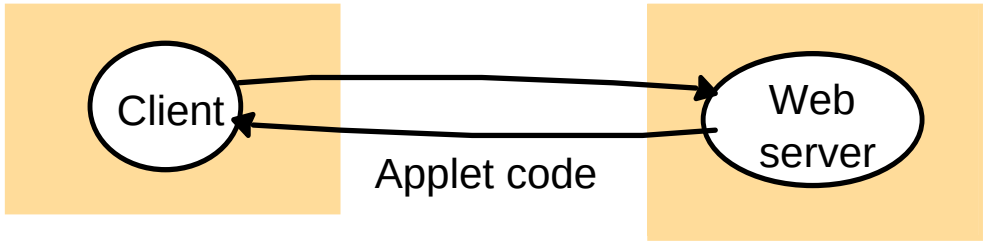


Figure 2.6 Web applets

- Running code locally vs. remotely
  - Network bandwidth
  - What examples have you seen?
  - Security issues?

a) client request results in the downloading of applet code



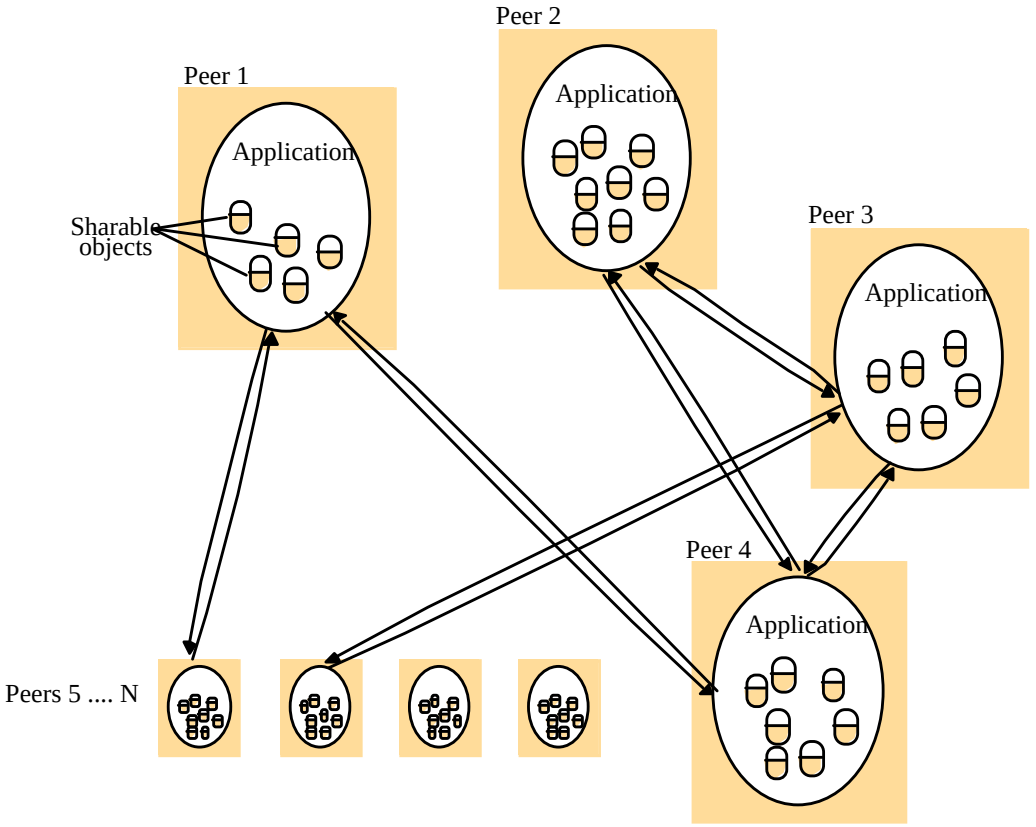
b) client interacts with the applet

Mobile agents

- Running program that travels from one computer to another
  - Perform tasks on users' behalf
  - Security issues
- How does it compare to one client interacting with multiple servers?
  - What if the program needs to access a lot of remote data?
- How does it compare with applets?
- Agents can provide functionality a remote web site does not provide, but allows access to data.

Peer to Peer

- Peers play similar roles
- No distinction of responsibilities



Variations

- Variations on the client-server and P2P models
  - the user of multiple servers
  - the use of caches to increase performance and resilience
  - the use of mobile code and mobile agents
  - users' need for low-cost computers with limited hardware resources that are simple to manage
  - the requirement to add and remove mobile devices in a convenient manner

Mapping and Placement

- Key issues of mapping to physical infrastructure
  - mapping of services to multiple servers
  - caching
  - mobile code
  - mobile agents
- Considerations
  - ***The placement of the components*** across a network of computers – define patterns for the distribution of data and workloads
  - ***The interrelationships between the components*** – that is, their functional roles and the patterns of communication between them
- 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

2.3.2 Architectural patterns

- Layering
  - –Platform: hardware, OS
  - –Middleware
    - Definition: a software layer that provides a programming abstraction to mask the heterogeneity of the underlying networks, hardware, operating systems and programming languages
    - Examples:
      - – Sun RPC, Java RMI
      - – CORBA, Microsoft .NET, Java J2EE
    - reliable services in the middle layer, still need application-specific reliability
  - –Applications, services

Figure 2.7 Software and hardware service layers in distributed systems

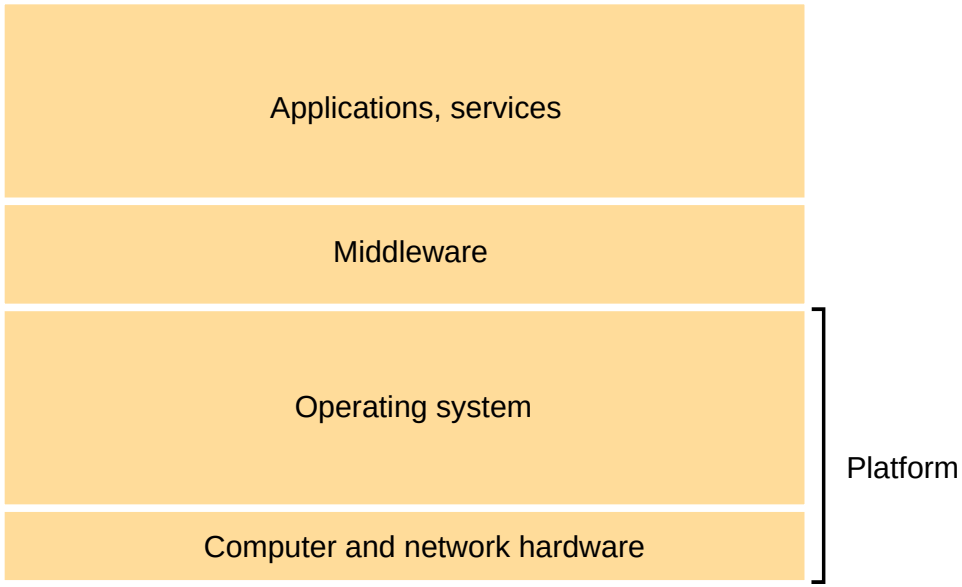


Figure 2.8 Two-tier and three-tier architectures

- Tiered architecture
  - Organize functionality of a given layer and place them into appropriate servers or physical nodes
  - Complementary to layering which deals with the vertical organization of services

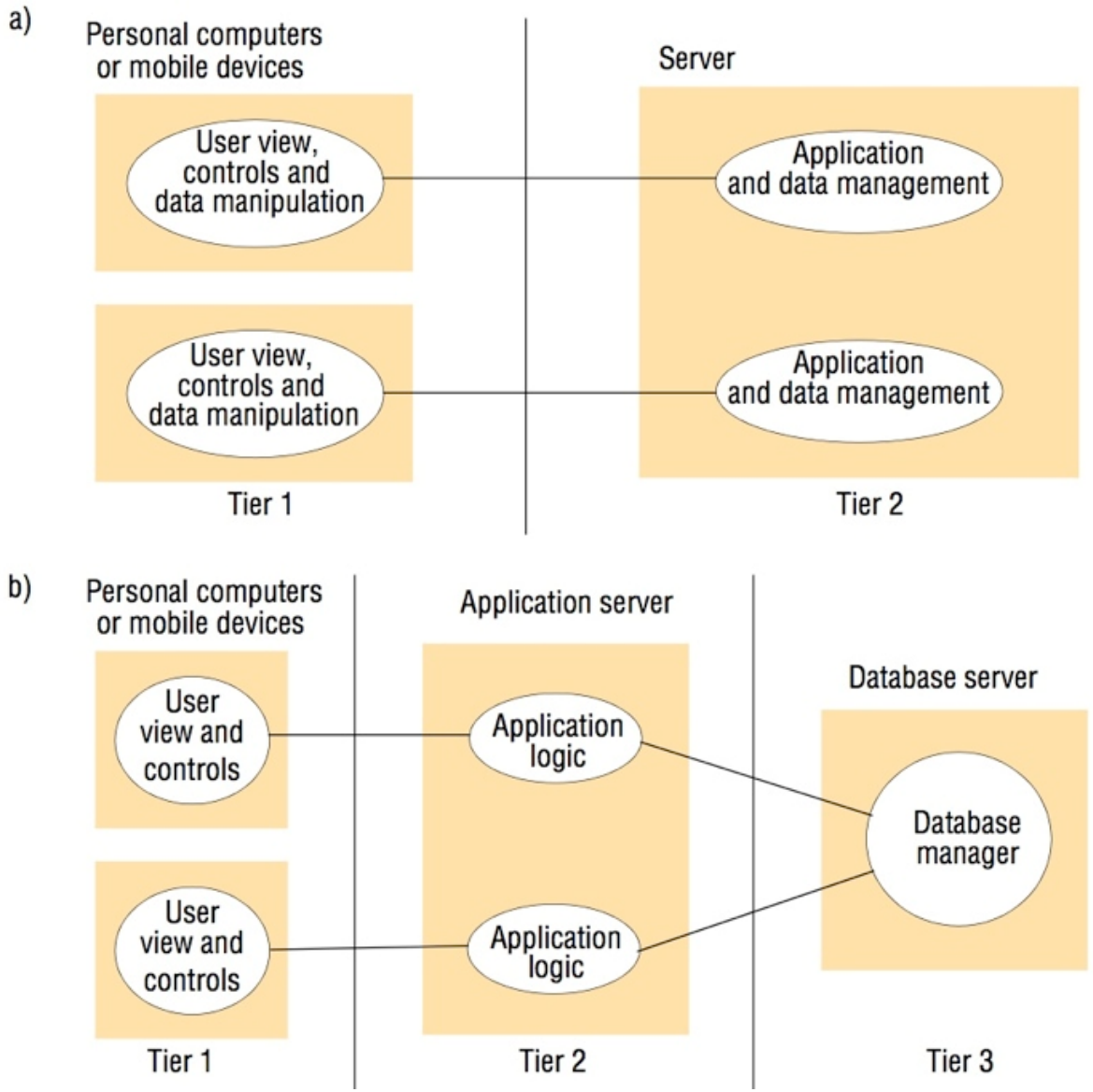


Figure 2.10 Thin clients and compute servers (network computers)

- Thin client
  - Basically a display with keyboard (“dumb terminal”)
  - Remote computation, storage
- How does it compare with the
  - PC model or
  - distributed workstation model?

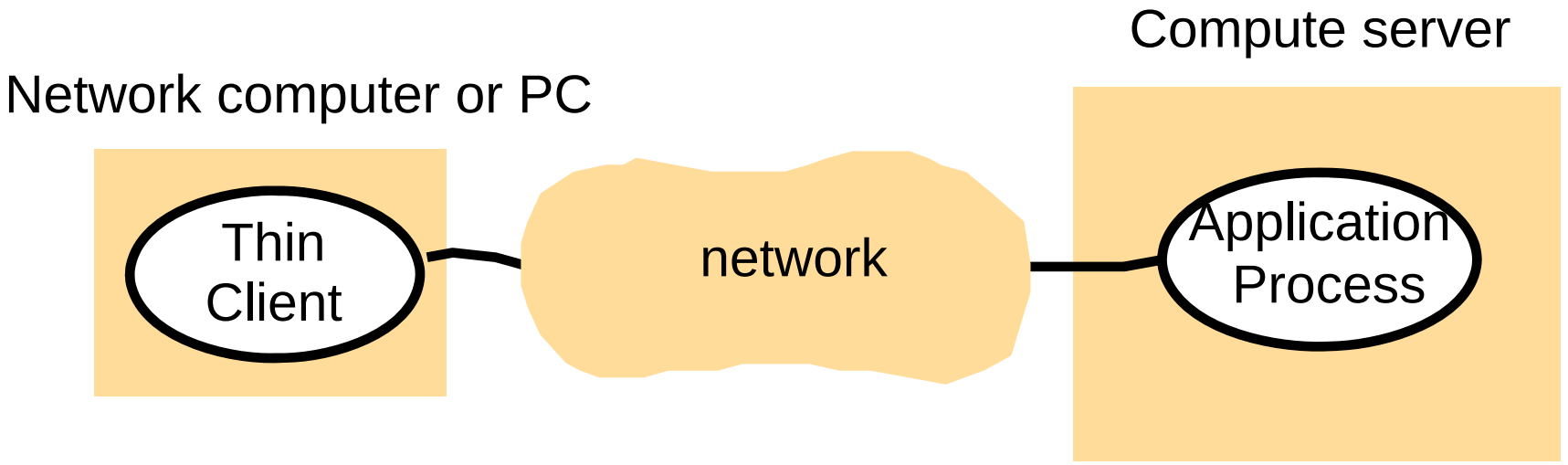
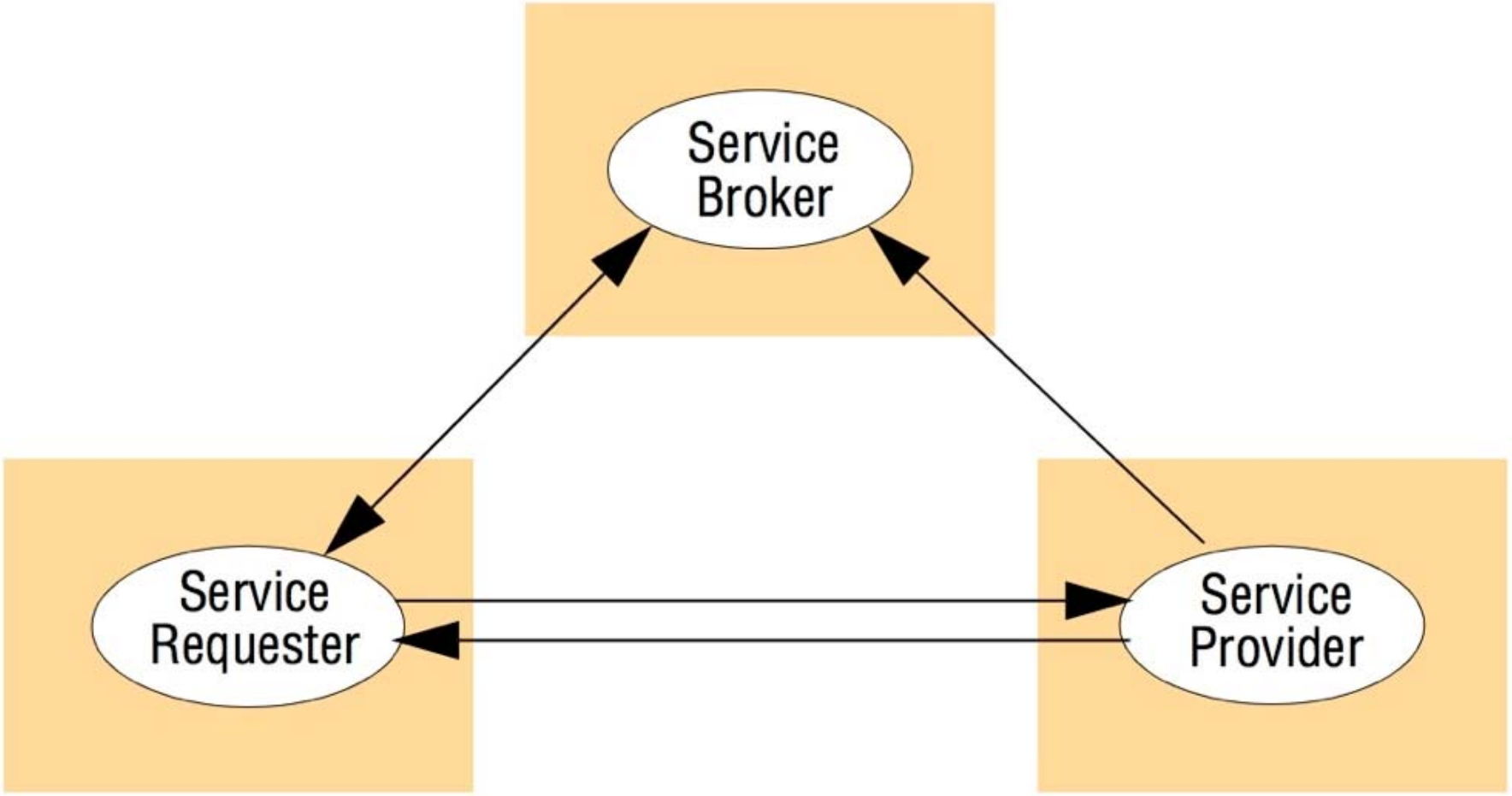


Figure 2.11 The web service architectural pattern



- Examples: Registry in Java RMI and naming service in CORBA

2.3.3 Associated middle solutions

Figure 2.12 Categories of middleware

Major categories:	*Subcategory*	*Example systems*
Distributed objects (Chapters 5, 8)	Standard	RM-ODP
	Platform	CORBA
	Platform	Java RMI
Distributed components (Chapter 8)	Lightweight components	Fractal
	Lightweight components	OpenCOM
	Application servers	SUN EJB
	Application servers	CORBA Component Model
	Application servers	JBoss
Publish-subscribe systems (Chapter 6)	-	CORBA Event Service
	-	Scribe
	-	JMS
Message queues (Chapter 6)	-	Websphere MQ
	-	JMS
Web services (Chapter 9)	Web services	Apache Axis
	Grid services	The Globus Toolkit
Peer-to-peer (Chapter 10)	Routing overlays	Pastry
	Routing overlays	Tapestry
	Application-specific	Squirrel
	Application-specific	OceanStore
	Application-specific	Ivy
	Application-specific	Gnutella

Interfaces and objects

- The set of functions available for invocation in a process (whether it is a server or a peer process) is specified by one or more interface definitions

- At the programming level
  - Need standard interfaces
    - Access/provide services/objects
    - RPC/RMI

Design requirements for distributed architectures

- Performance
  - Responsiveness, throughput, load balancing
- Quality of service (QoS)
  - time-critical applications (real-time apps)
  - guarantee certain level of quality delivered by the deadline
  - allocation of computation and communication resources
- Caching and replication
  - web caching: server provides expiration time
- Dependability
  - fault tolerance: redundancy, recovery
  - security

2.4 Fundamental Models

- Fundamental properties in processes and communication, shared among different architectures discussed previously

- The aspects of DS that we wish to capture:
  - Interaction: communication (i.e. information flow) and coordination (synchronization and ordering of activities) between processes
  - Failure: analysis of potential effects whenever a fault occurs
  - Security: analysis of threats to a system

2.4.1 Interaction model

Two significant factors affecting process interaction

- performance of communication channels
  - latency: transmission, access, os
  - bandwidth
  - jitter: variation in time taken to deliver a series of messages
- computer clocks and timing events
  - clock drift: two processes running on different computers read their clocks at the same time, their local clocks may supply different time values
  - clock drift rate: the relative amount that a computer clock differs from a perfect reference clock

Two Variants of the Interaction Model

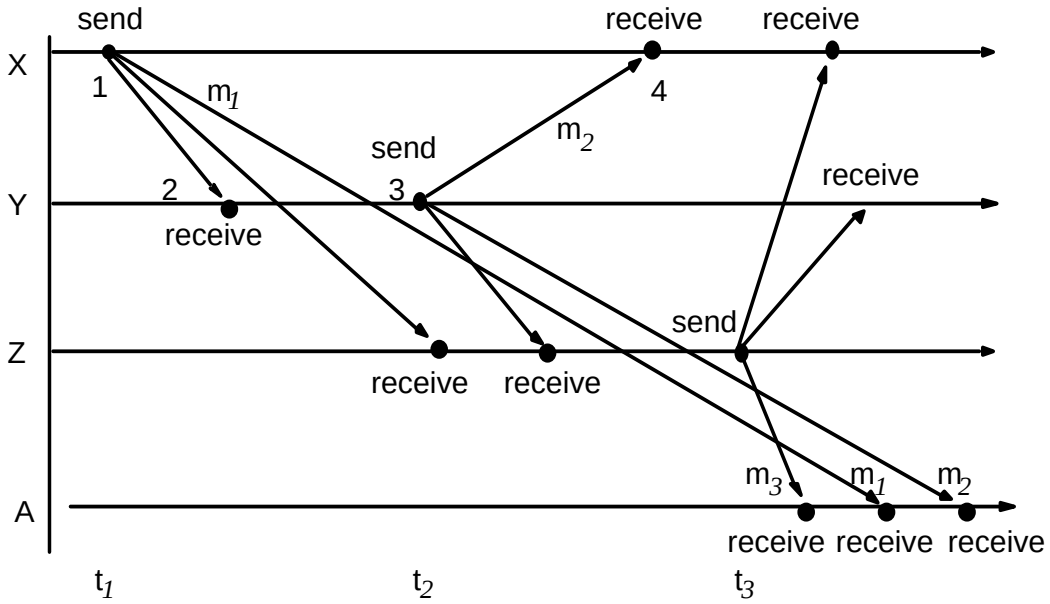
- ***Synchronous*** distributed systems
  - lower and upper bounds for execution of a step
  - message transmission in bounded time
  - clock drift rate is bounded

  - failures can be detected when bounds are exceeded
  - accomplished by allocating sufficient resources
- ***Asynchronous*** distributed systems
  - no bounds on process speed, message delay, clock drift rate

  - failures are harder to detect
  - performance can't be guaranteed

Event Ordering

- Event ordering
  - Relative ordering might be more important than exact time
  - logical clocks--ordering events without physical clocks
- Example: Consider a mailing list with users X, Y, Z, and A
  - X sends a message with the subject *Meeting*
  - Y and Z reply by sending a message with the subject *Re: Meeting*



User A:

1. From Z: Re: Meeting
2. From X: Meeting
3. From Y: Re: Meeting

A receives Z's response before X's message!

Figure 2.13 Real-time ordering of events

2.4.2 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: when a process or communication channel fails
  - Arbitrary Failure: any type of error may occur (i.e. set wrong values)
  - Timing Failure

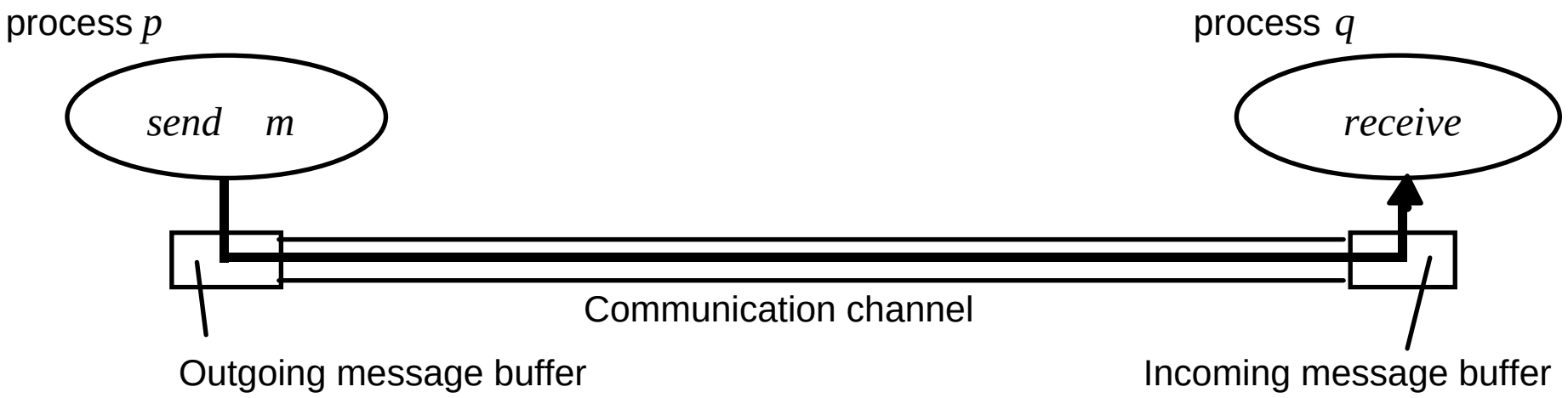


Figure 2.14 Processes and channels

Figure 2.15 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.

Figure 2.16 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.

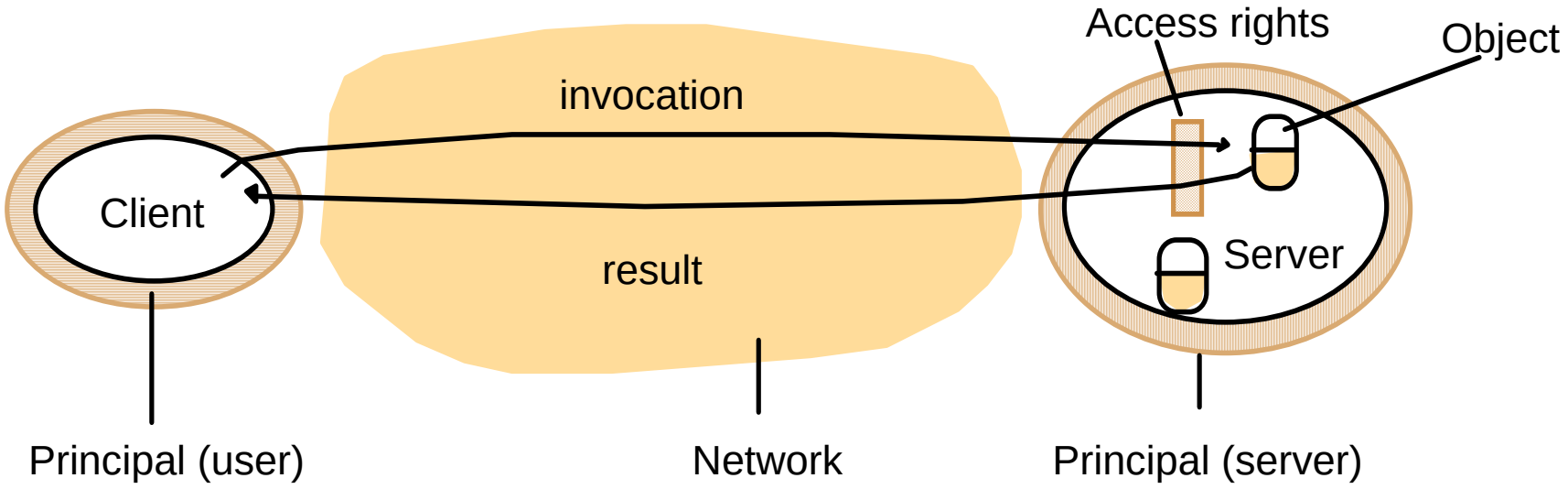
2.4.3 Security Model

- The security of a distributed system 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

Security Model – Protecting Objects

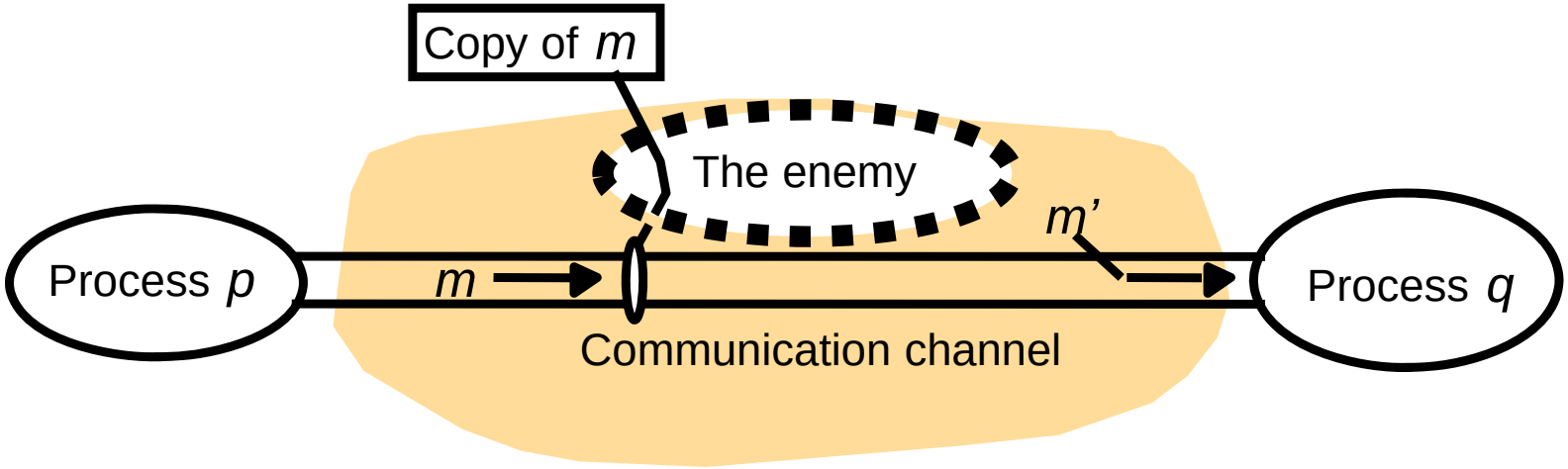
- Protecting objects:
  - authorization (access rights to principals)
  - authentication (identity of parties/principals)

Figure 2.17 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

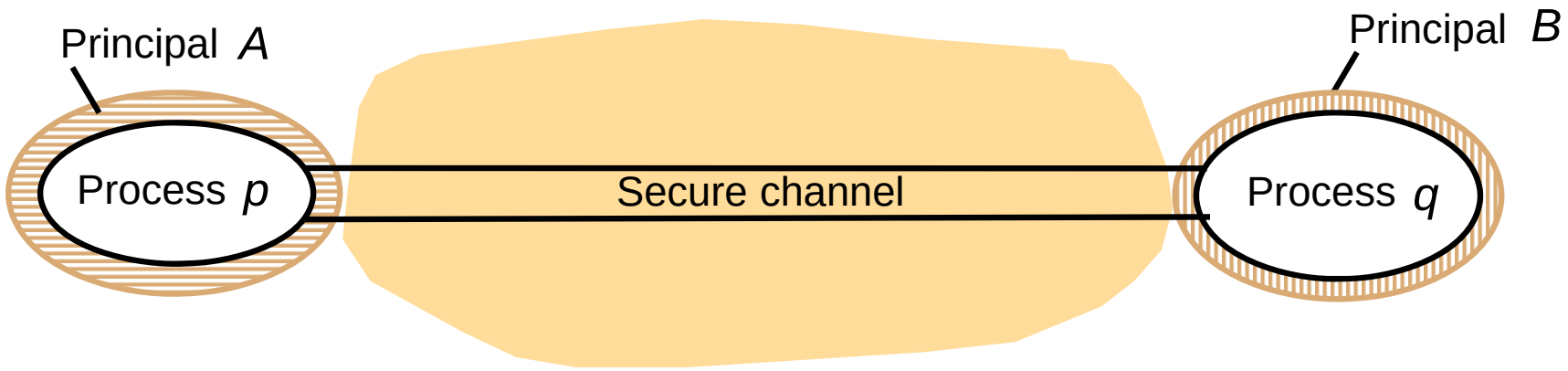
Figure 2.18 The enemy



- Threats form a potential enemy:
  - threats to processes
    - Server: an enemy might generate an invocation with a false identity
    - Client: the client could receive a result that was from an enemy
  - threats to communication channels: an enemy can copy, alter or inject messages
  - denial of service: the enemy overloads physical resources
  - mobile code: may play a Trojan horse

Security Model – Securing processes and their interactions

Figure 2.19 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

2.1 Introduction

2.2 Physical models

2.3 Architectural models

2.4 Fundamental models

2.5 Summary