CHAPTER 2
SYSTEM MODELS

UNIVERSITÉ DE NEUCHÂTEL

SYSTEM MODELS

Introduction

Physical models

Architectural models

Fundamental models

INTRODUCTION

- Architecture paradigms for distributed systems
  - Layers
  - Client – server
  - Peer – to – Peer
- Fundamental characteristics
  - No global time
  - Communication through exchange of messages
- Models to characterize distributed system architectures
  - **Interaction** model
  - **Failure** model
  - **Security** model

INTRODUCTION

- Physical models
  - hardware composition
- Architectural models
  - tasks
- Fundamental models
  - interactions, failures, security

- Difficulties and threats
  - Widely varying modes of use
  - Wide range of system environments
  - Internal problems
  - External threats

PHYSICAL MODELS

- Early distributed systems
  - client-server
- Internet-scale distributed systems
  - heterogeneity, standards, middleware
- Contemporary distributed systems
  - mobile, ubiquitous, cloud
- Distributed systems of systems
  - sensors networks, Internet-of-Things

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

Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5

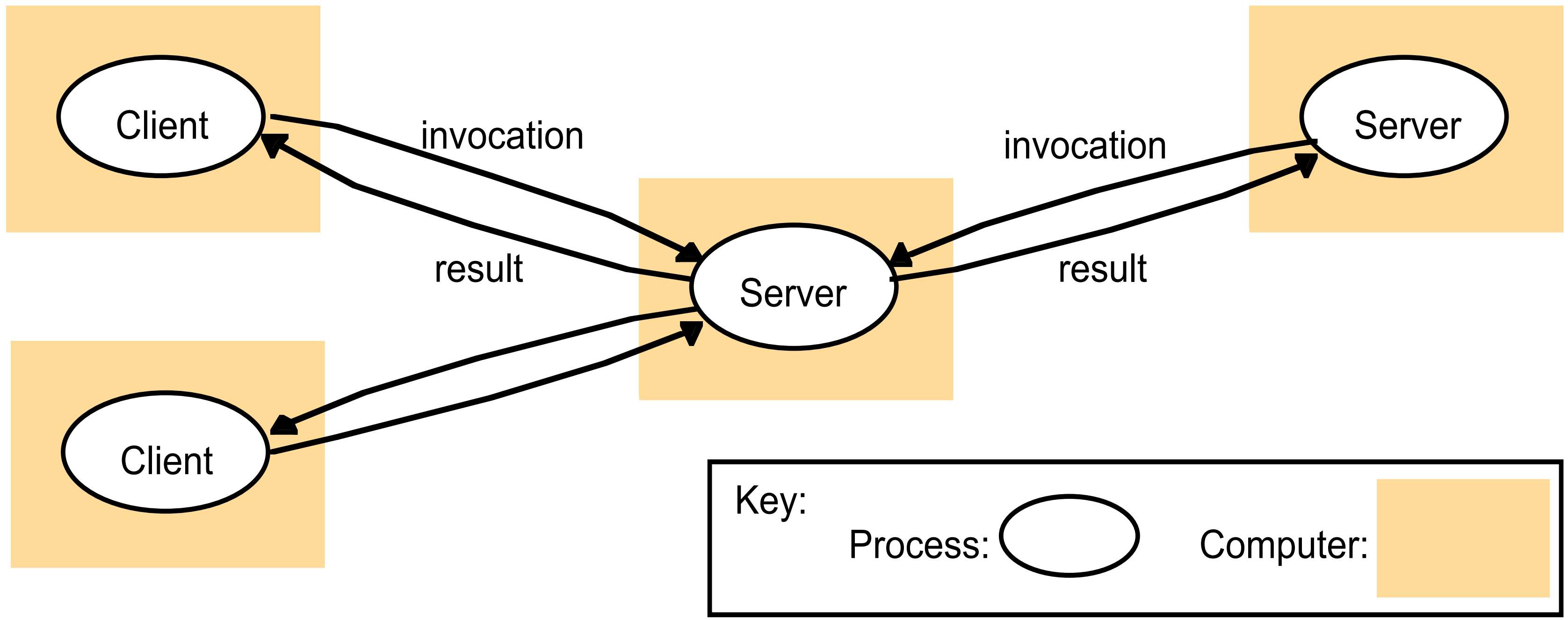
ARCHITECTURAL MODELS

- Elements
  - Communicating entities
  - Communication paradigms
  - Roles & responsibilities
  - Placement

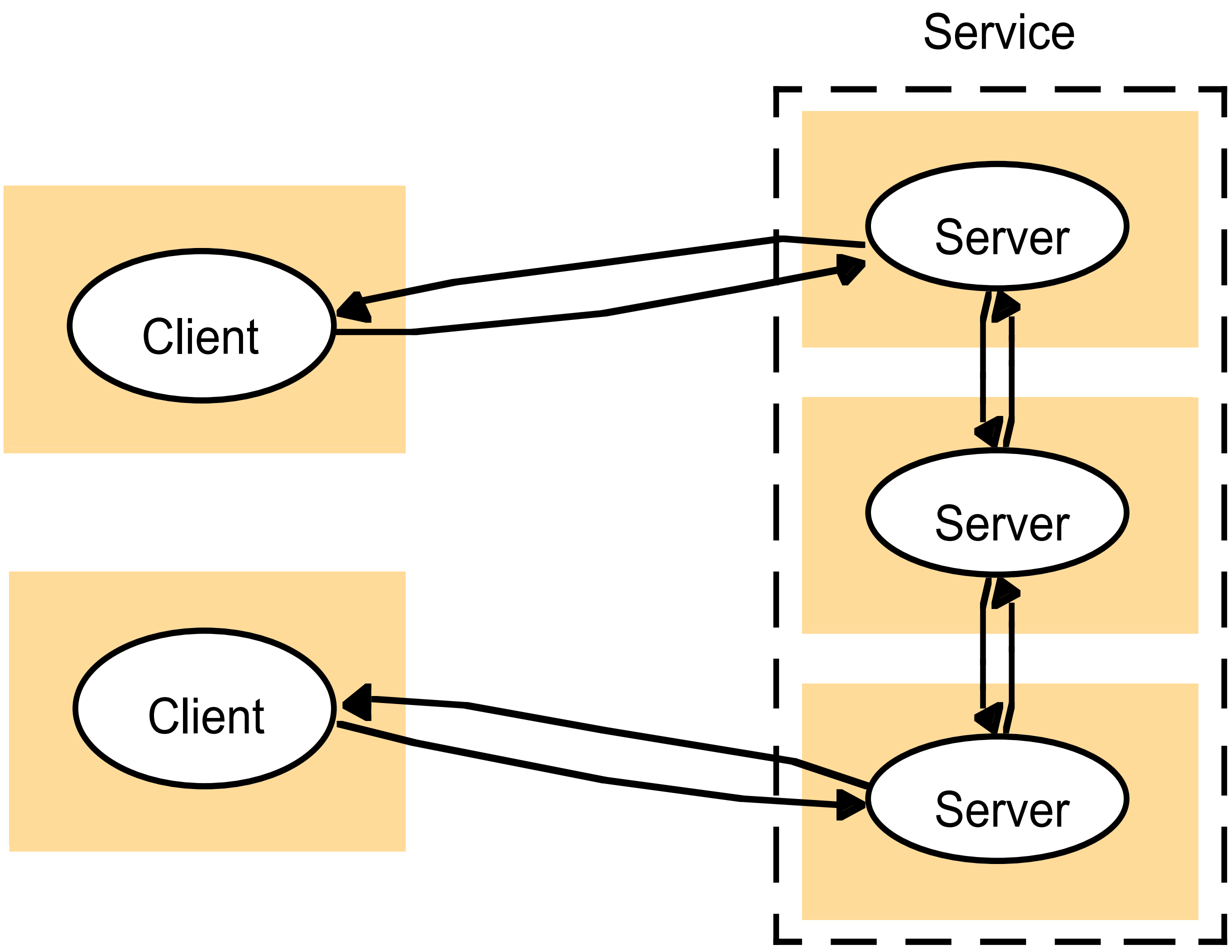
COMMUNICATION ENTITIES AND PARADIGMS

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

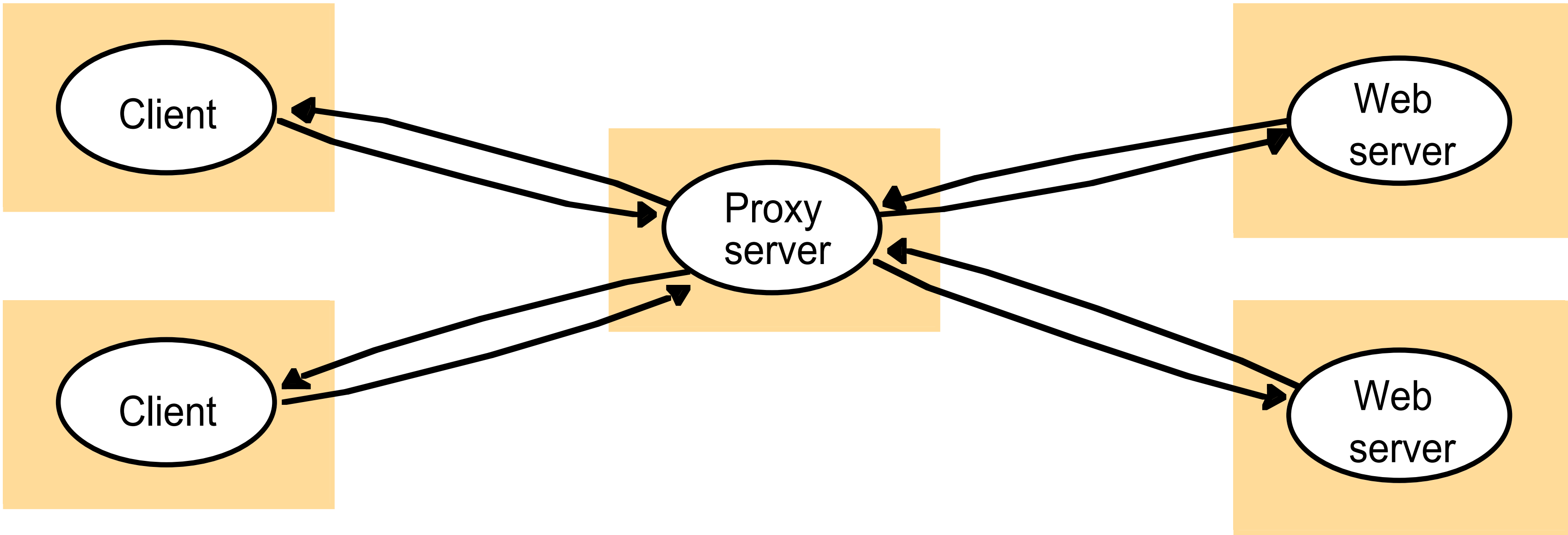
CLIENT-SERVER



PLACEMENT : MULTIPLE SERVERS

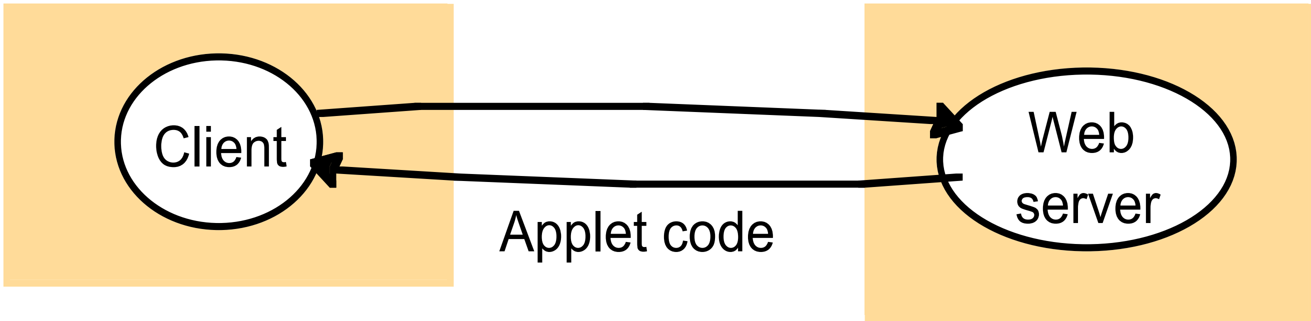


PLACEMENT : PROXY SERVER



PLACEMENT : APPLETS

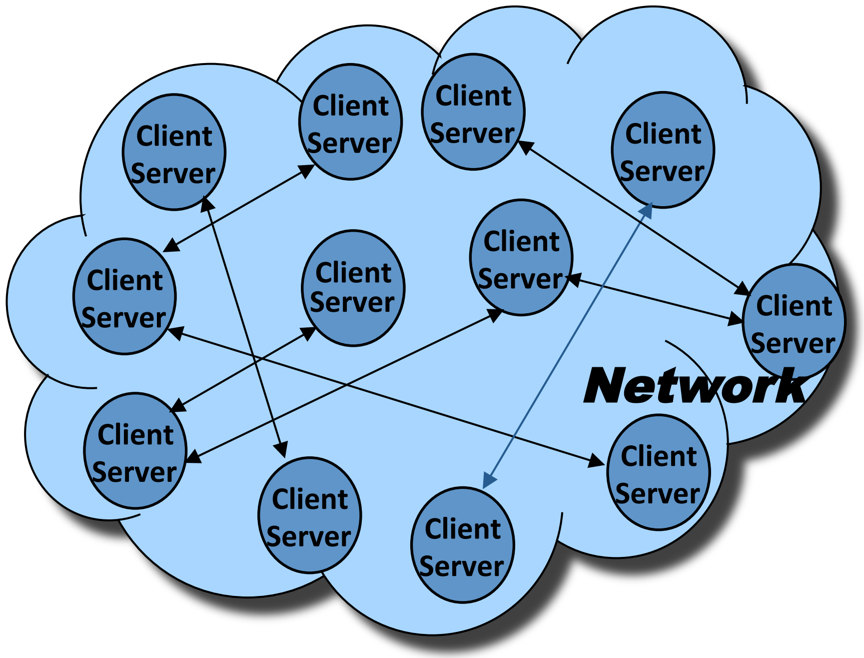
a) client request results in the downloading of applet code



b) client interacts with the applet

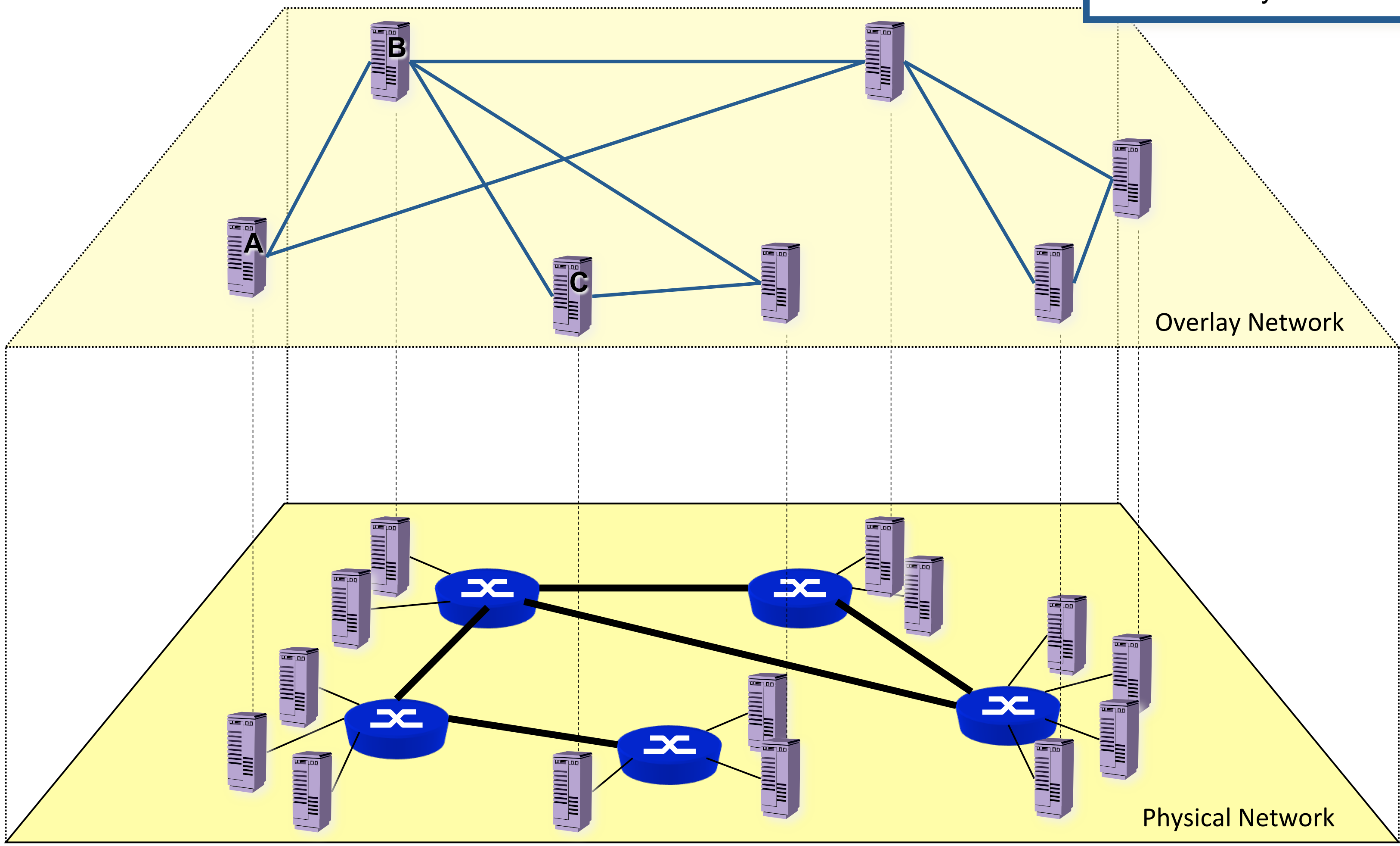
PEER-TO-PEER SYSTEMS

- P2P is nothing new – see Arpanet (Internet)
- Every participating node acts as a client and a server at the same time («servent»)
- **Decentralized** system architecture :
  - No central control/coordination or DB
  - No global state : global behavior emerges from local interactions
  - All data and services are accessible from any peer
  - Peers are numerous and autonomous; frequent join and leave is the norm
  - Peers and interconnections are unreliable
  - Fault-tolerant, self-organizing
  - Symmetric communication
- «Business model»: Every node contributes to the system by providing access to some of its resources: incentive to participate.
- Online communities



OVERLAYS

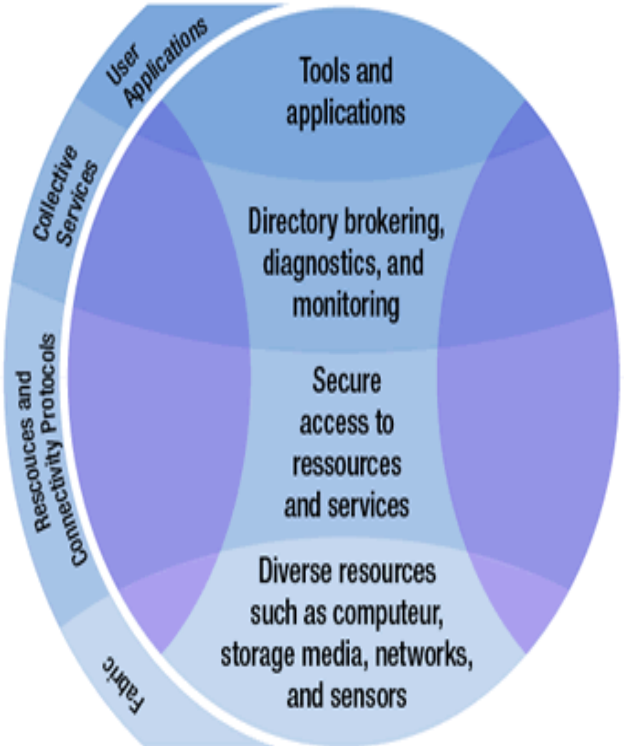
Focus on the application layer

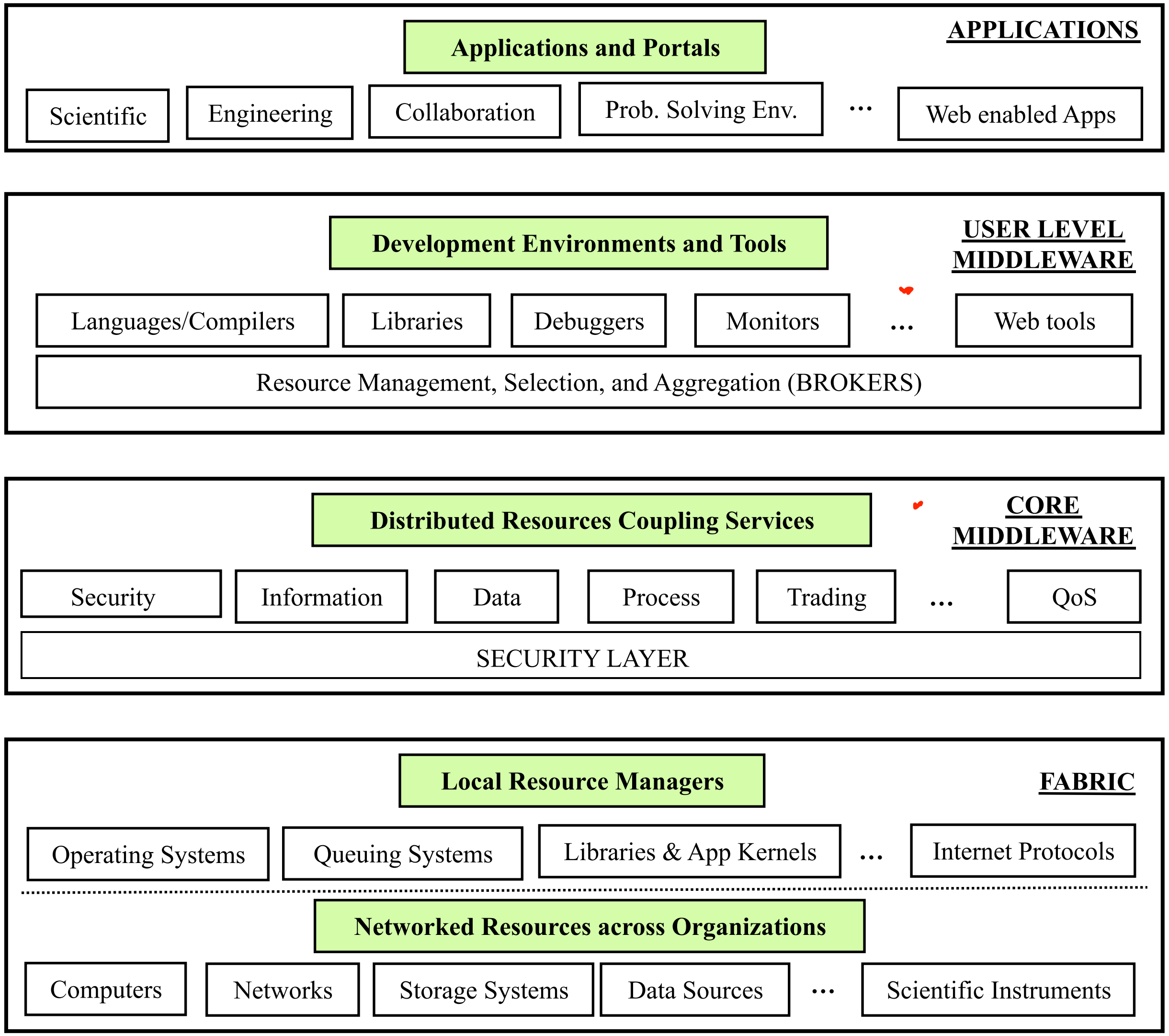


AGENTS

UNIVERSITÉ DE NEUCHÂTEL

- Agent characteristics
  - *Autonomy*: capability to pursue its goals without interactions or commands from the environment
  - *Social ability*: capability to interact with the environment; context/situation awareness
  - *Reactivity*: capability of reacting appropriately to influences or information from its environment
  - *Proactivity*: capability to take the initiative under specific circumstances
  - *Mobility*: the ability to move around in an electronic network (Security? Trust? Ontology?)
  - *Collaboration*: the ability to collaborate and coordinate actions and behavior
  - *Learning*: the ability to learn from past situations and actions


User Applications
Collective Services
Rescouces and Connectivity Protocols
Fabric
Tools and applications
Directory brokering, diagnostics, and monitoring
Secure access to ressources and services
Diverse resources such as computeur, storage media, networks, and sensors

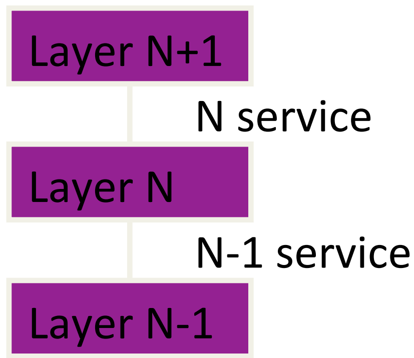

APPLICATIONS
Applications and Portals
Scientific
Engineering
Collaboration
Prob. Solving Env.
…
Web enabled Apps
USER LEVEL MIDDLEWARE
Development Environments and Tools
Languages/Compilers
Libraries
Debuggers
Monitors
…
Web tools
Resource Management, Selection, and Aggregation (BROKERS)
CORE MIDDLEWARE
Distributed Resources Coupling Services
Security
Information
Data
Process
Trading
…
QoS
SECURITY LAYER
FABRIC
Local Resource Managers
Operating Systems
Queuing Systems
Libraries & App Kernels
…
Internet Protocols
Networked Resources across Organizations
Computers
Networks
Storage Systems
Data Sources
…
Scientific Instruments

ARCHITECTURAL PATTERNS

- Layers
- Tiered architecture
- Thin clients
- Brokering

LAYERS

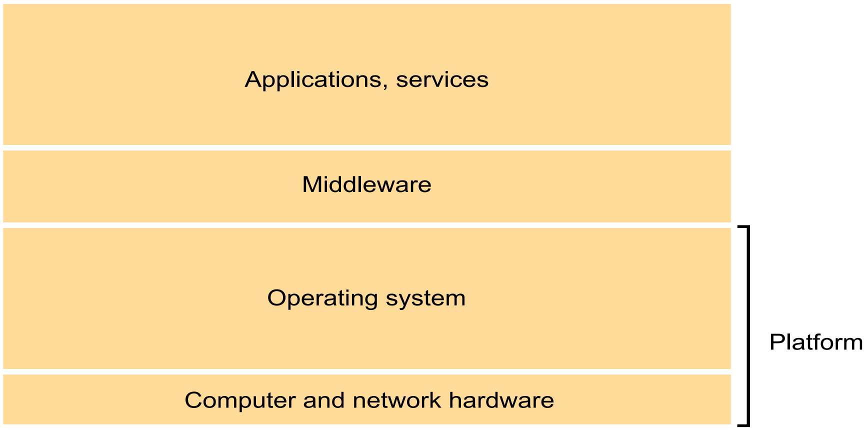
- **Idea:** breaking up the complexity of systems into smaller parts : ***layers*** and ***services***
  - **Layer :** group of closely related and coherent functions
  - **Service :** functions provided to next higher layer



  - Examples of layered architecture
    - Operating systems (kernel, I/O system, file system etc.)
    - Network protocol architectures (TCP/IP, IEEE 802.x etc.)

LAYERS

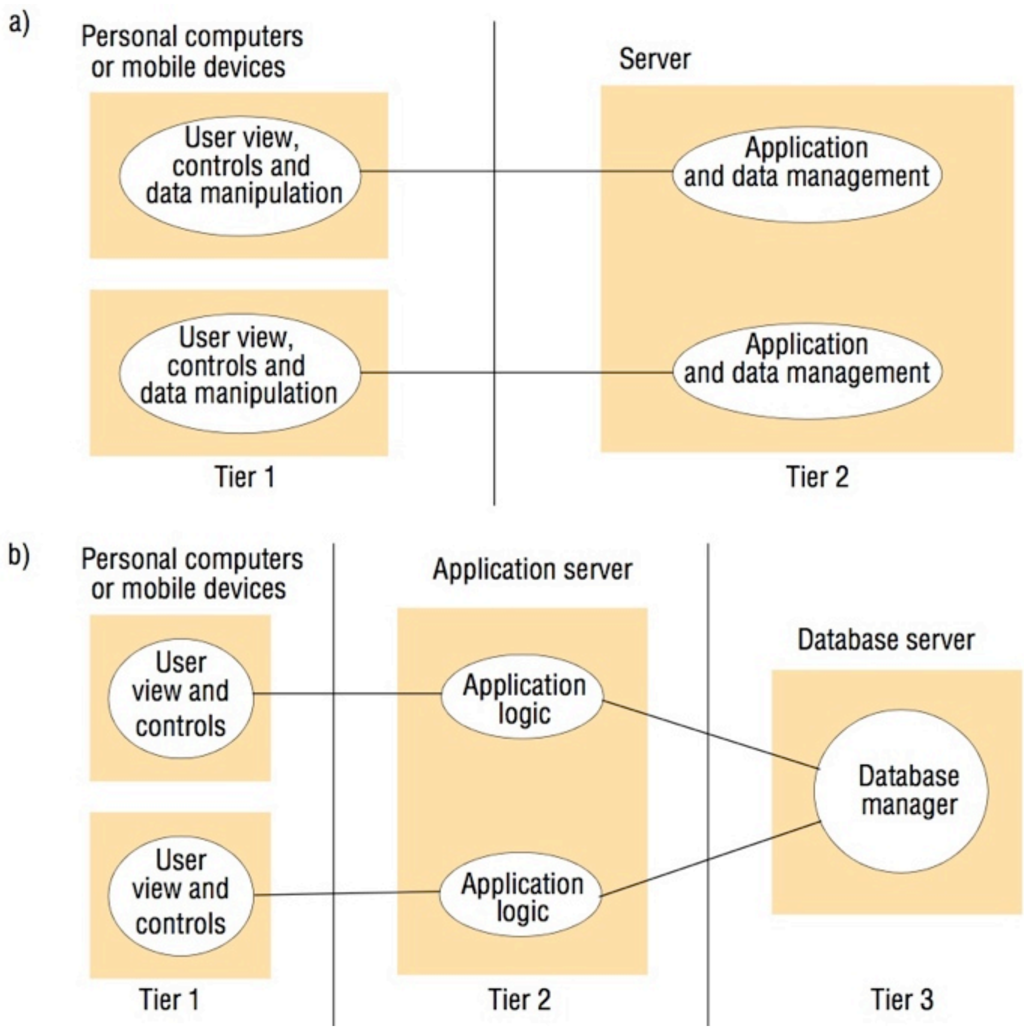
- Layers in Distributed Systems



© Pearson Education 2012

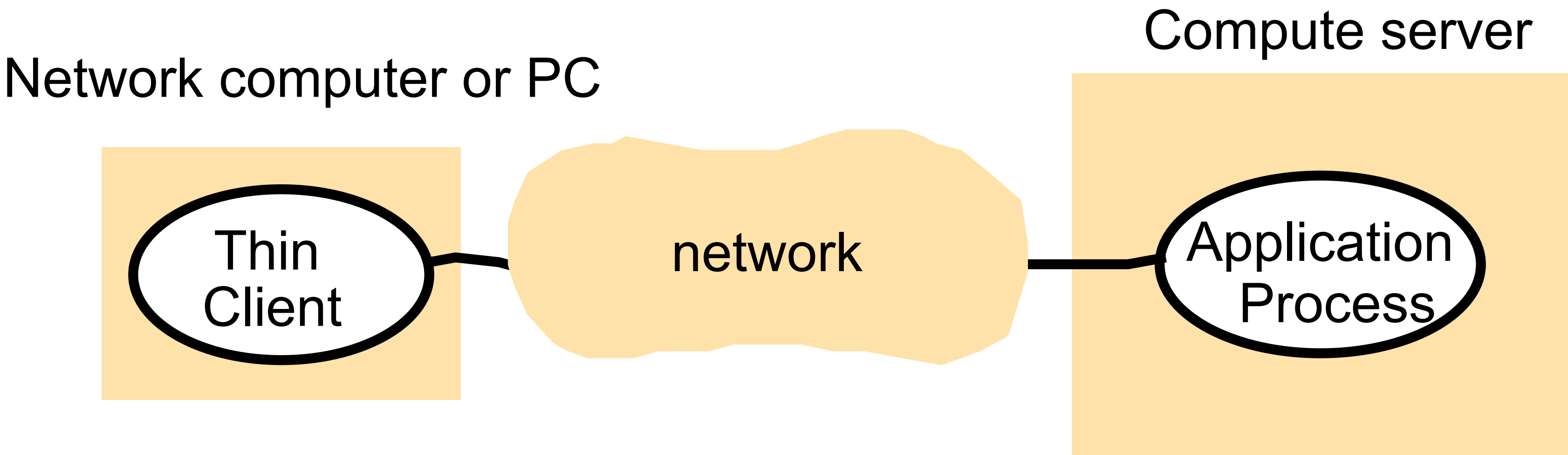
- Platform: computer and network hardware; operating system
  - Intel/AMD processor, Ethernet; Windows x/MacOS/IOS
- Middleware: transparency across heterogeneous platforms
  - Communication and resource sharing
  - Examples
    - Java Remote Method Invocation (RMI), SUN
    - Common Object Request Broker Architecture (CORBA), OMG
    - Distributed Component Object Model (DCOM), Microsoft
    - Open Distributed Processing (ODP), ITU-T/ISO

TWO-TIER AND THREE-TIER ARCHITECTURES

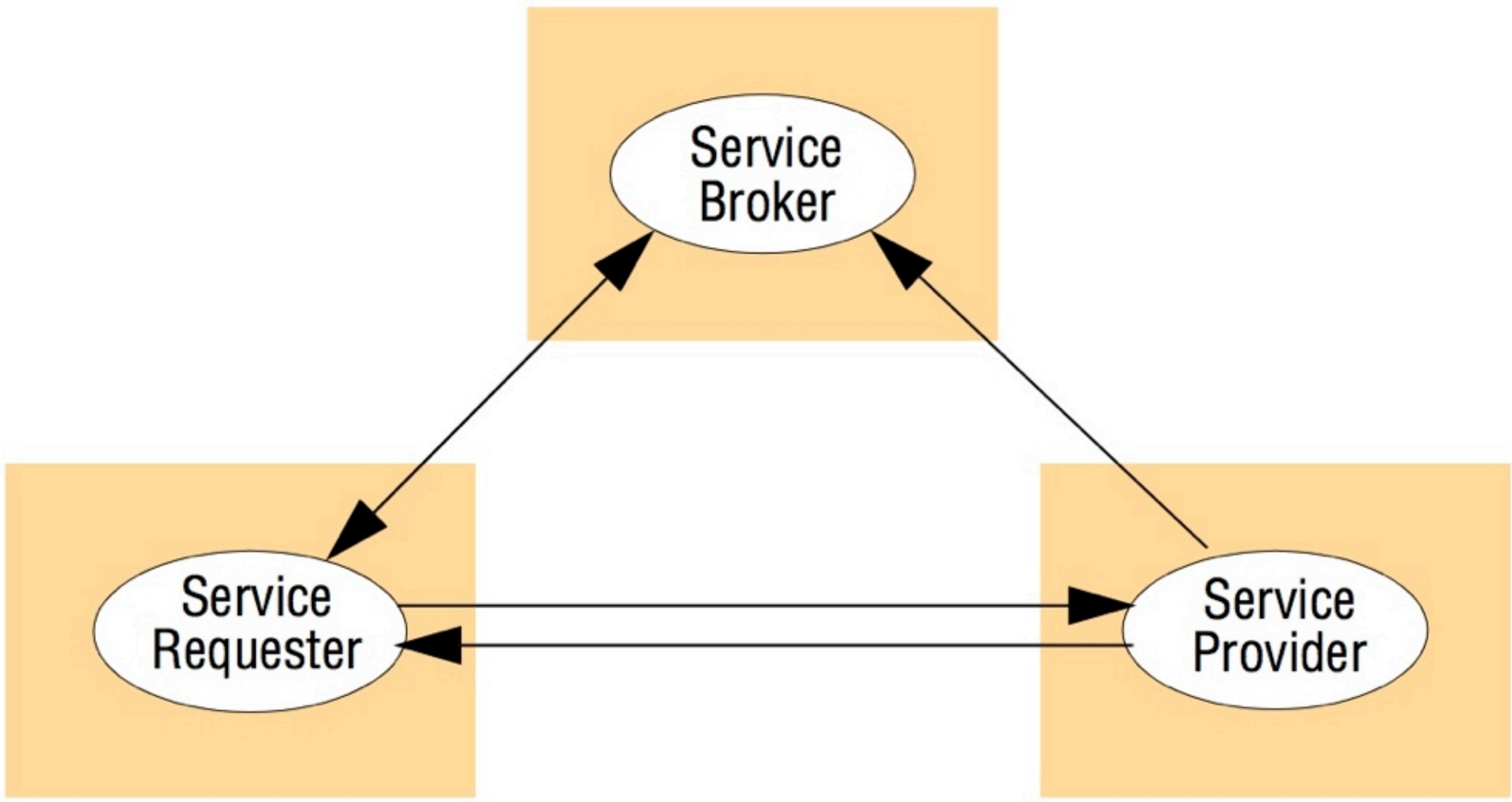


Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5

THIN CLIENTS AND COMPUTE SERVERS



THE WEB SERVICE ARCHITECTURAL PATTERN



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

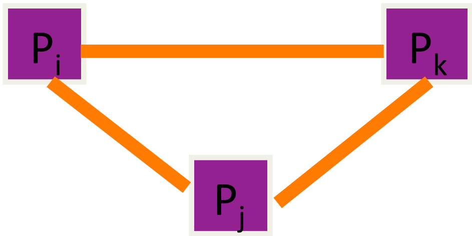
Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5

FUNDAMENTAL MODELS

- Aspects
  - **Interaction**
  - **Failure**
  - **Security**

INTERACTION MODEL

- Distributed Systems
  - Multiple processes
  - Connected by communication channels
- Interaction between processes is needed since processes must
  - Exchange data
  - Cooperate, coordinate and synchronize
  - Compete for resources
- Distributed Algorithms
  - Steps to be executed by each process
  - Communication between processes
    - Synchronisation
    - Information flow/exchange



INTERACTION

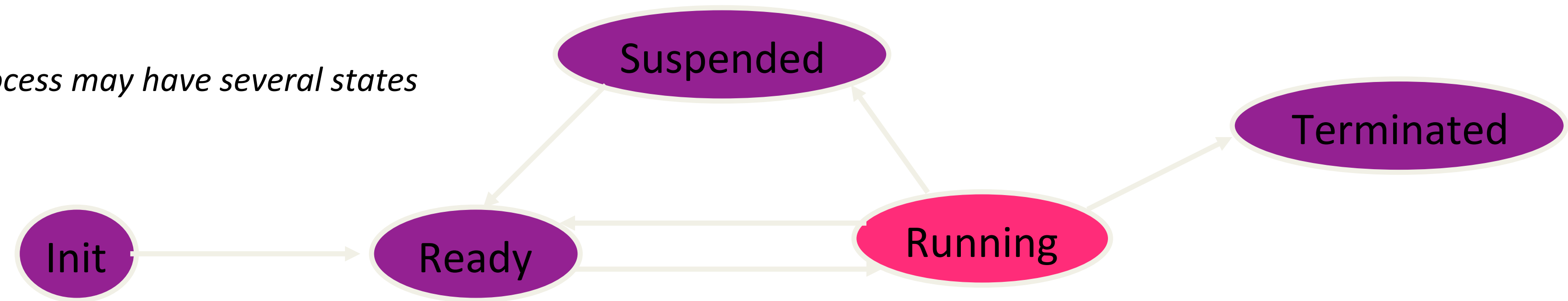
- Communication
  - Latency
  - Bandwidth
  - Jitter

- Clocks
  - Clock drift

PROCESSES AND THREADS

- Different definitions and kinds of use exist in literature
- Processes and tasks are mostly understood as UNIX-processes; these are *heavyweight processes*, since they have a separate address space
- Most systems offer *leightweight processes (or **threads**)* running in a shared address space
- Most UNIX systems offer standard UNIX tasks, which may consist of several threads
- In such a way, a process becomes an execution environment for its threads

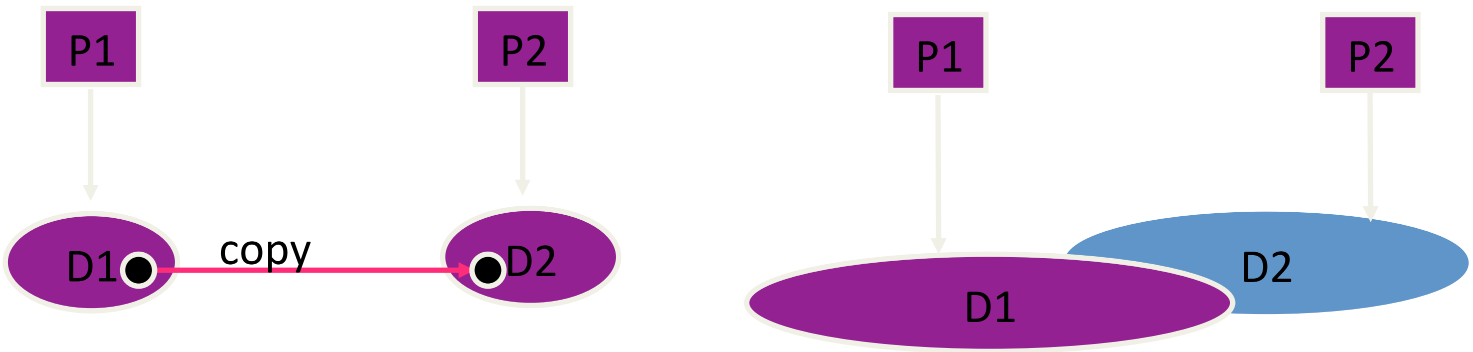
a process may have several states



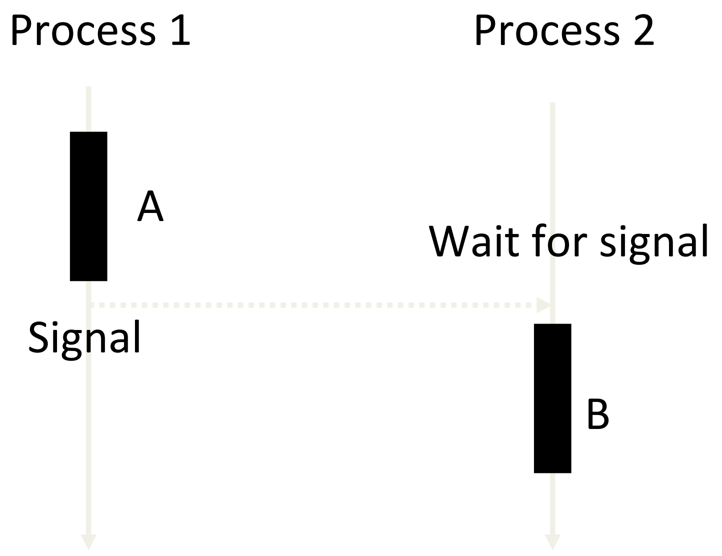
INTER-PROCESS COMMUNICATION (IPC)

- Interactions between processes are needed, since processes must
    - exchange data
    - generate new processes
    - wait for the end of another process
    - collaborate
    - compete for resources
- IPC has a functional and a time-dependent aspect:
    - functional: exchange of information
    - time dependent: sequence control (coordination and synchronization)
    - Two kinds of IPC are possible

  Communication or *Cooperation*



COORDINATION (FLOW CONTROL)



- Sequence A in Process 1 shall be executed before B in Process 2
- The system call *wait* stops Process 2 until the signal is given
- The execution of wait resets the signal (consumes the signal)

COOPERATION (PROBLEMS)

- Problems may appear, if more than one process accesses the same data
- Inconsistent data may be generated

1. Disjoint processes without common data

Process 1	*Process 2*
x1 = max (a1, b1)	x2 = max (a2, b2)
y1 = max (c1, d1)	y2 = max (c2, d2)
z1 = x1 + y1	z2 = x2 - y2

2. Disjoint processes with common data that are *read only*

Process 1	*Process 2*
x1 = max (a1, b2)	x2 = max (a1, b1)
y1 = max (a2, b1)	y2 = max (a2, b2)
z1 = x1 + y1	z2 = x2 - y2

COOPERATION (PROBLEMS, II)

3. Processes with common data (*read/write*)

a := 10

Process 1	*Process 2*
1.1 a := a * a	2.1 a := a/2
1.2 print a	2.2 print a

Results depending on the execution sequence:

1.1-1.2-2.1-2.2	=>	100	50
1.1-2.1-1.2-2.2	=>	50	50
1-1-2.1-2.2-1.2	=>	50	50
2.1-2.2-1.1-1.2	=>	25	5
2.1-1.1-1.2-2.2	=>	25	25
2.1-1.1-2.2-1.2	=>	25	25

COOPERATION (PROBLEMS, III)

Race Conditions

Generated by time critical execution of instructions

For instance:

Process 1	*Process 2*
(Observer)	(Reporter)
repeat	repeat
observe(event);	print(counter);
counter ++;	counter:=0;
until (END)	until(END);

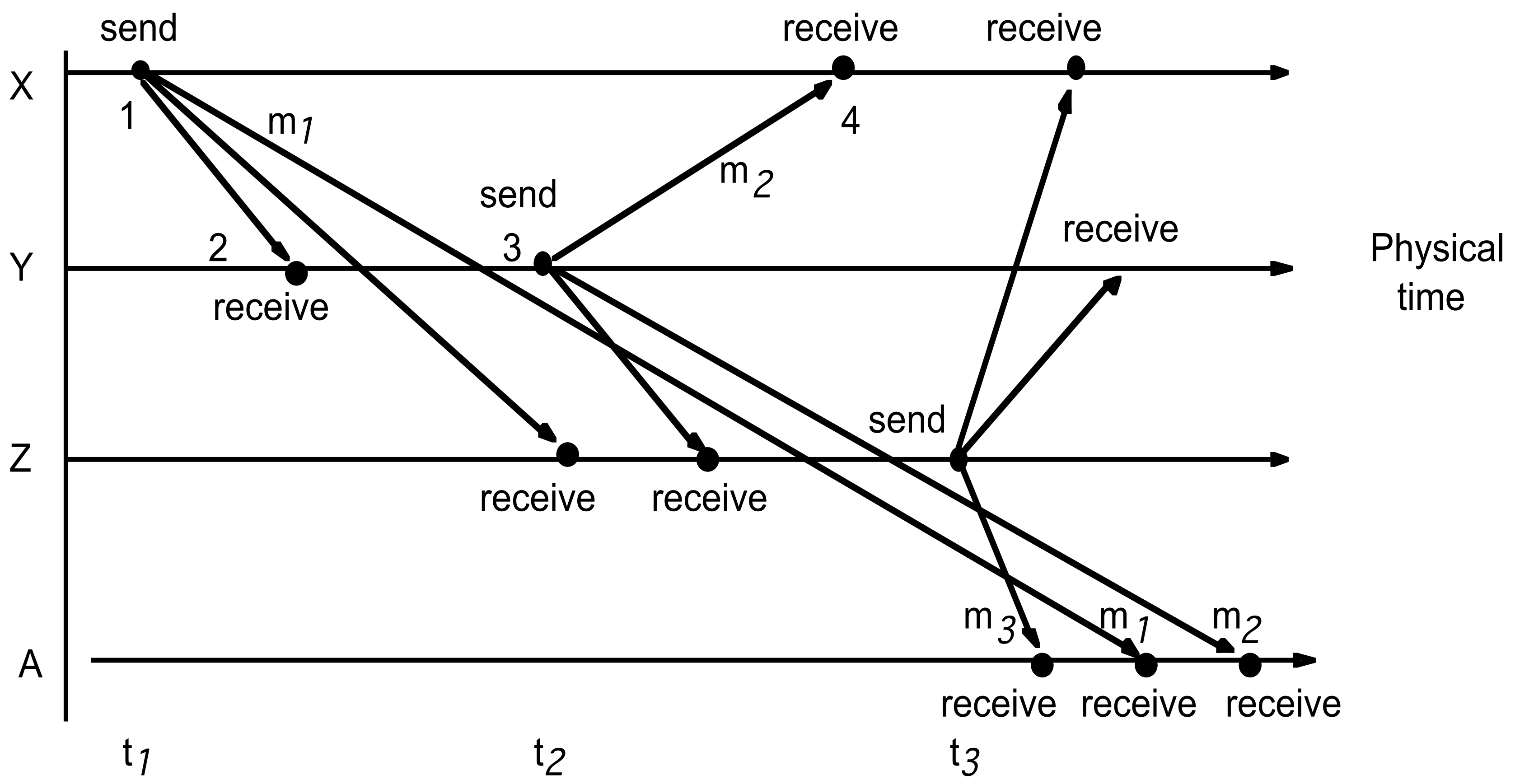
INTERACTION MODEL

- **Synchronous** distributed systems
  - Time to execute each step of computation within a process has lower and upper bounds known in advance
  - Message delivery times are bound to a value known in advance
  - Each process has a clock whose drift rate from the real time is bound by a known value
- **Asynchronous** distributed systems
  - **No bounds on :**
    - Process execution time
    - Message delivery times
    - Clock drift rate

TIME

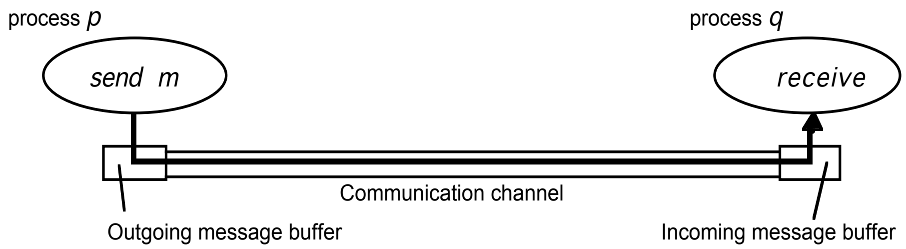
- We need to measure time accurately:
  - to know the time an event occurred at a computer
  - to do this we need to synchronize its clock with an authoritative external clock (e.g. UTC)
- Algorithms for clock synchronization useful for
  - concurrency control based on timestamp ordering
  - authenticity of requests e.g. in Kerberos
- There is no global clock in a distributed system
- Logical time is an alternative
  - It gives an ordering of events - also useful for consistency of replicated data

REAL-TIME ORDERING OF EVENTS



FAILURES

- Omission Failures
  - **Process** omission failures: process crashes
    - Failure detection with timeouts
    - A crash is *fail-stop* if another process can detect with certainty that the process has crashed
  - **Communication** omission failure: message is not delivered (dropped)
    - Possible causes are
      - Network transmission error
      - Receiver buffer overflow
- Arbitrary Failures
  - Process: omit intended processing steps or carry out unwanted ones
  - Communication channel: non-delivery, corrupted or duplicated data
- Timing Failures



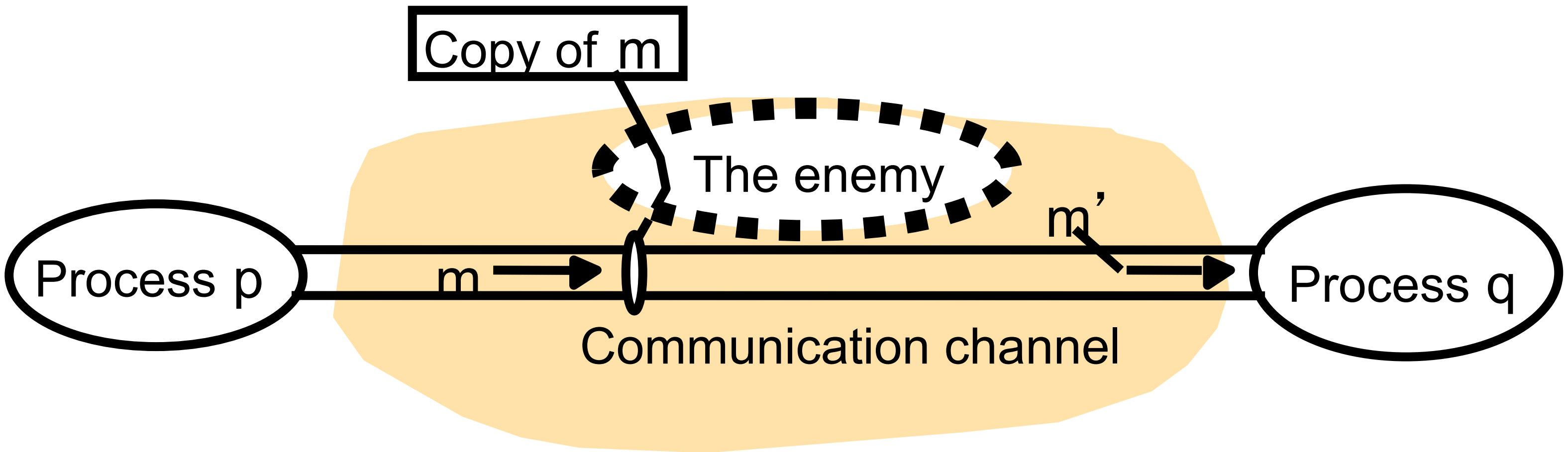
Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never 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

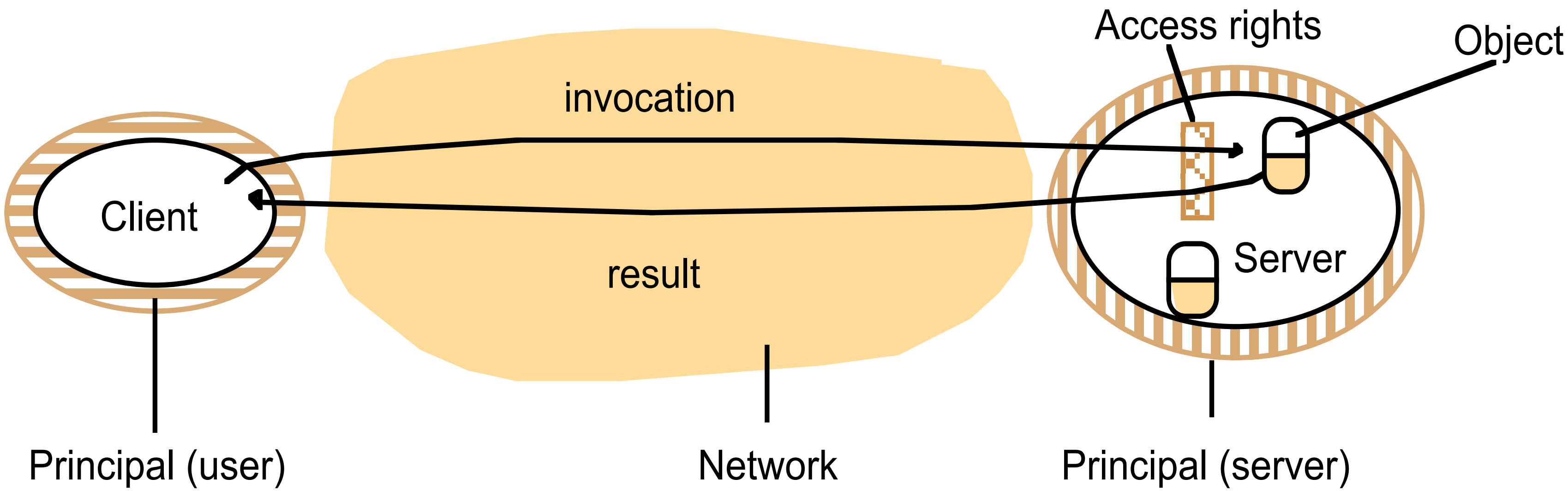
Class of Failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

SECURITY

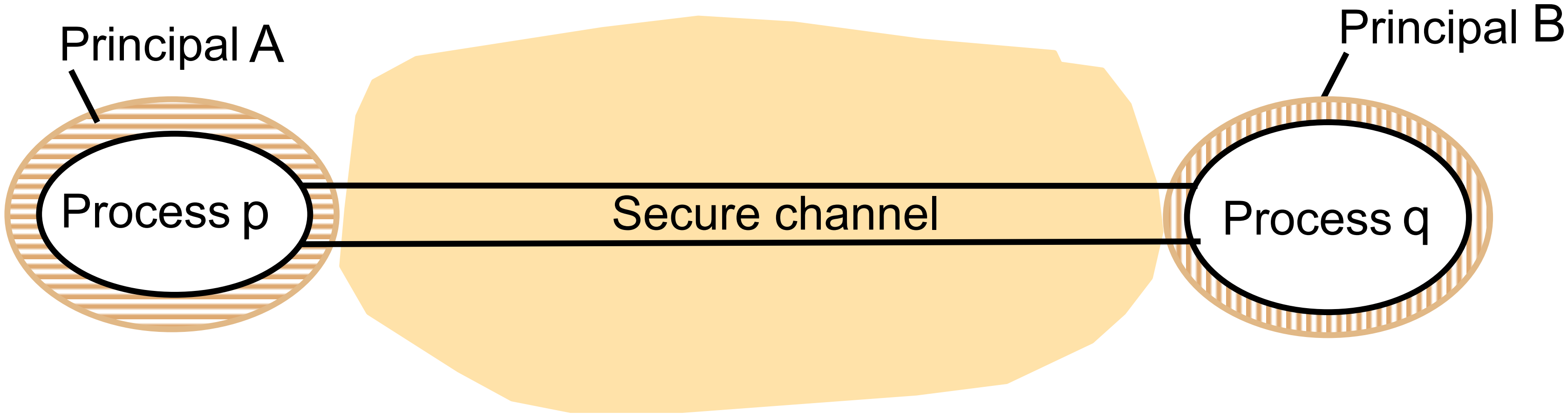
- Protecting access to objects
  - Access rights
  - User (client) authentication
- Protecting processes and interactions
  - Threats to processes: unauthenticated requests/replies
  - Threats to communication channels: unauthorised and unfriendly copying, altering or injecting of messages transient on the network. Use of secure channels based on *cryptographic* methods
- Denial of Service
  - e.g. *Pings* to selected Web sites or other servers
  - Generating deliberately network or server load to make network services unavailable
- Mobile Code
  - Malicious code (e.g. worms)



SECURITY MODEL: OBJECTS AND PRINCIPALS



SECURE CHANNELS



Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5