

Middleware and Distributed Systems

System Models

Dr. Martin v. Löwis

System Models (Coulouris et al.)

- Architectural models of distributed systems
 - placement of parts and relationships between them
 - e.g. client-server, peer-to-peer
- Fundamental models
 - formal description of properties common to all architectural models
 - addresses correctness, reliability, and security
- Selected drawings taken from Coulouris, Dollimore and Kindberg
Distributed Systems: Concepts and Design, Edition 4, © Pearson Education
2005

Architectural Models

- placement of components across a network of computers
 - define useful patterns for the distribution of data and workload
- interrelationships between components
 - functional roles, patterns of communication
- abstraction: server processes, client processes, peer processes
- variation of models, e.g. for client-server architecture
 - define mobile code to have some part of the application run on the client
 - support mobile clients to allow matching of clients and servers dynamically

Software Layers

© Pearson Education 2005

Application Services

Middleware

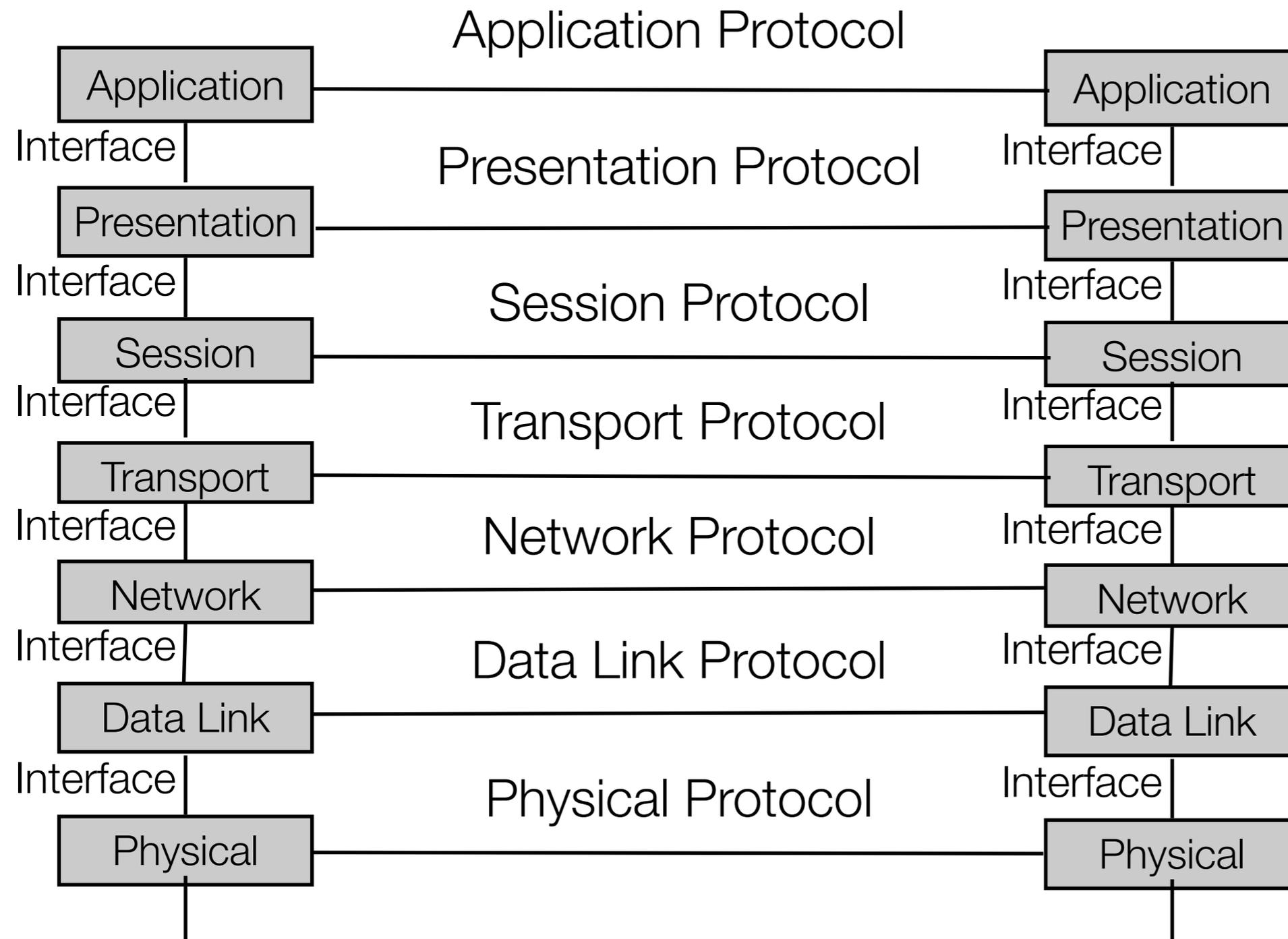
Operating System

Computer and network hardware



Platform

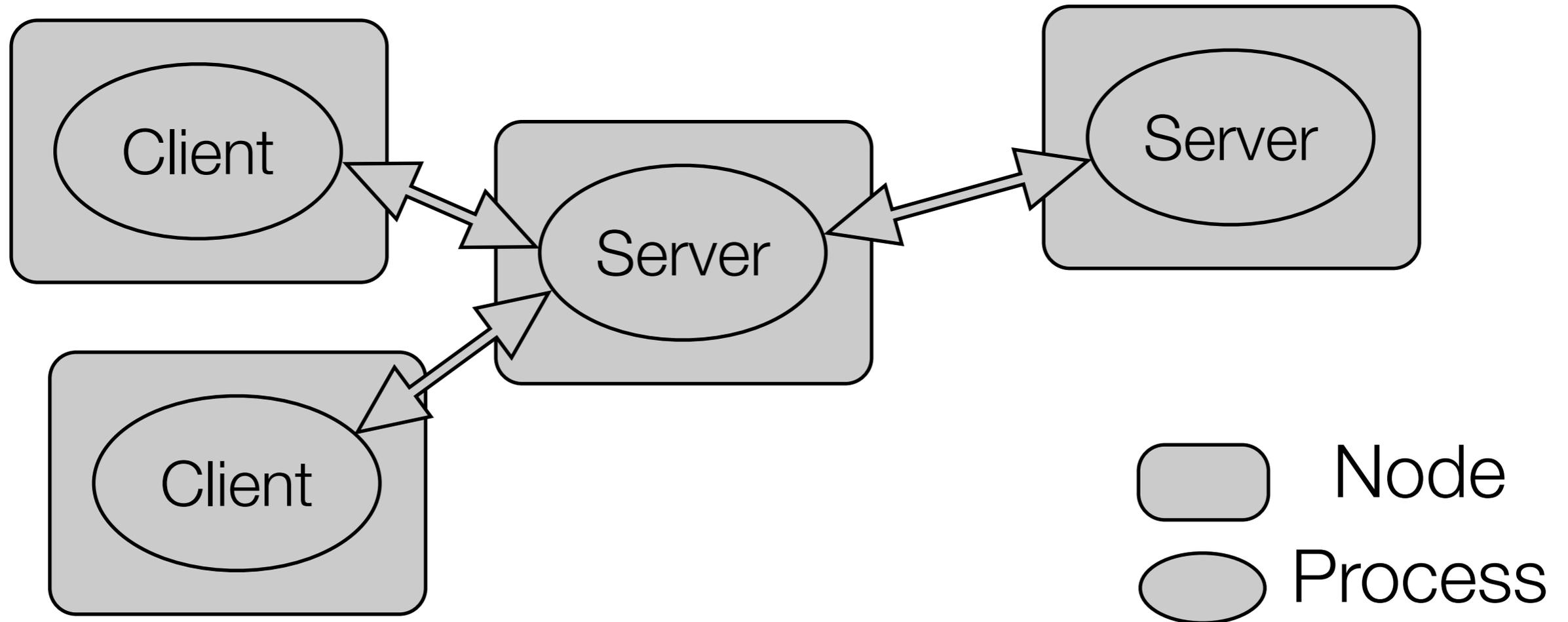
Protocol Layers: OSI-RM



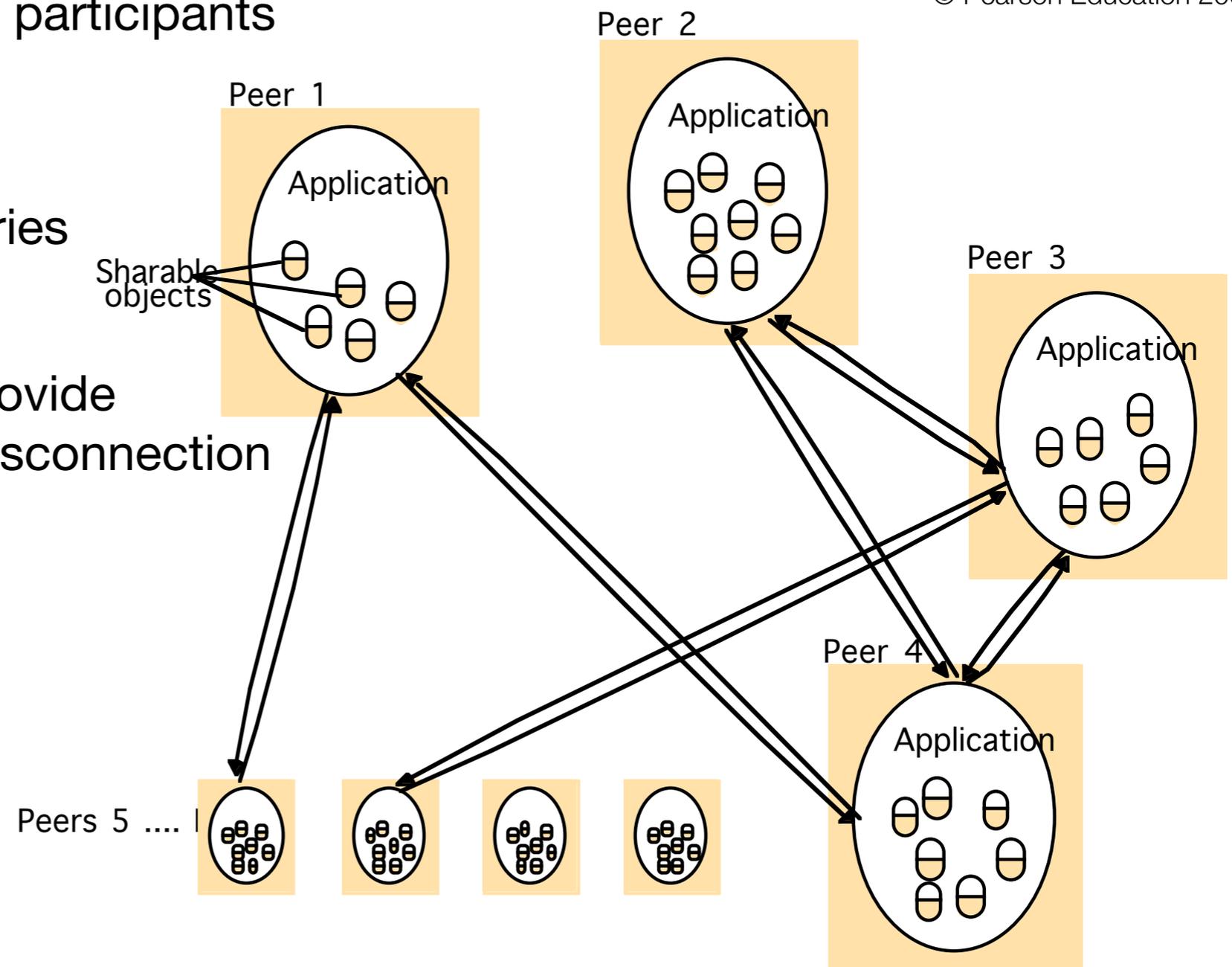
Client - Server Model

- Client: consumer process, uses remote services / information
- Server: provider process, offers a service / information
 - may in turn be client of another server

© Pearson Education 2005

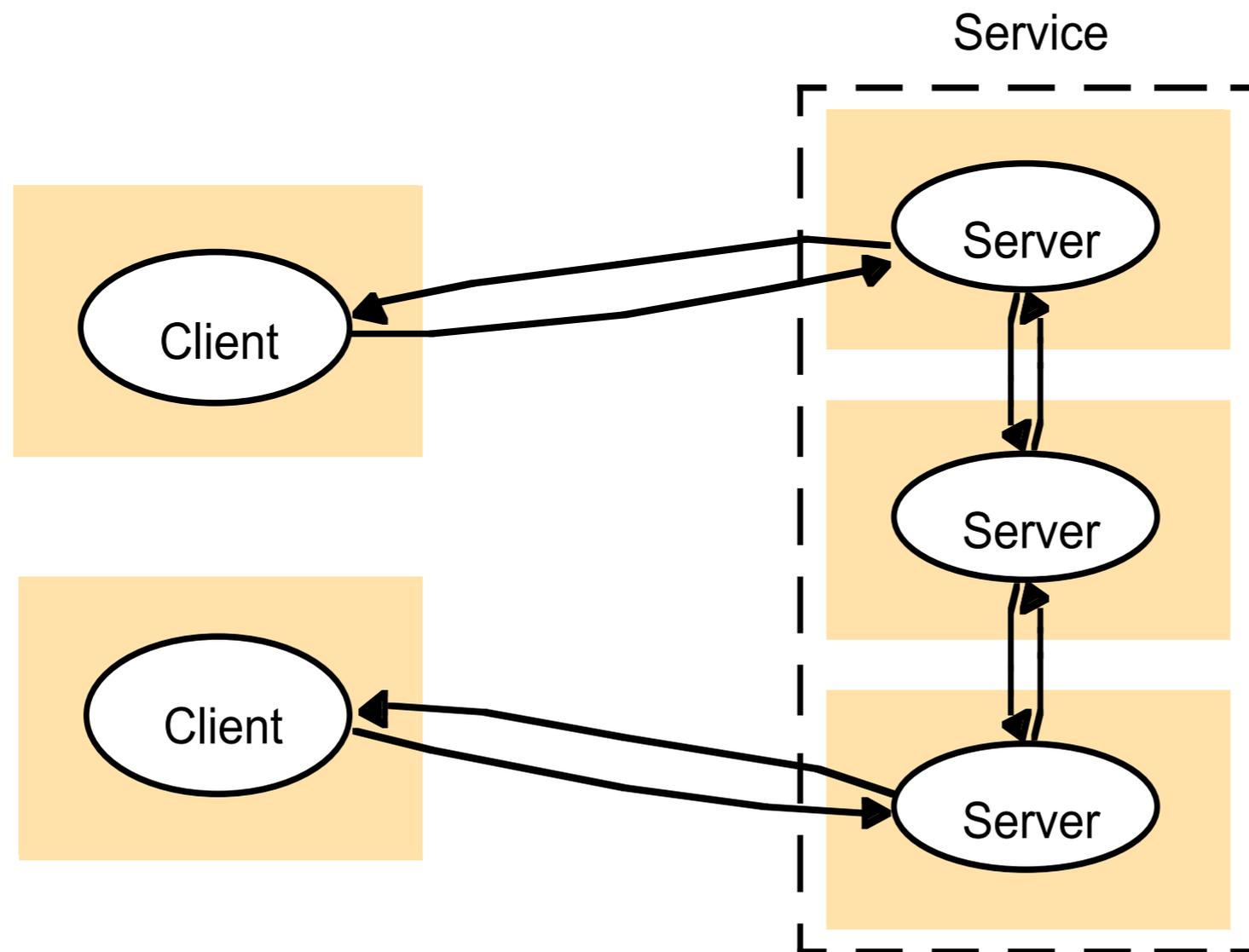


- potentially large number of participants
 - often, home users
- communication pattern varies over time
- replication necessary to provide resilience in the event of disconnection



Variation: Services provided by multiple servers

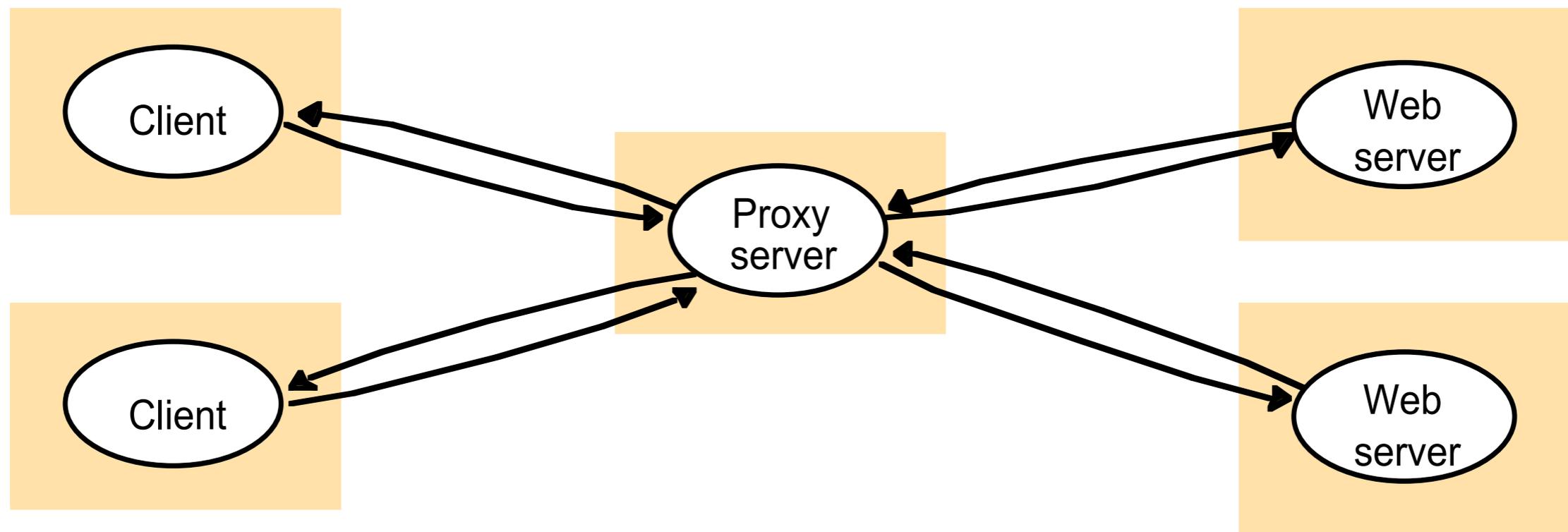
- Services provided by multiple servers



© Pearson Education 2005

Variation: Proxy Servers and Caches

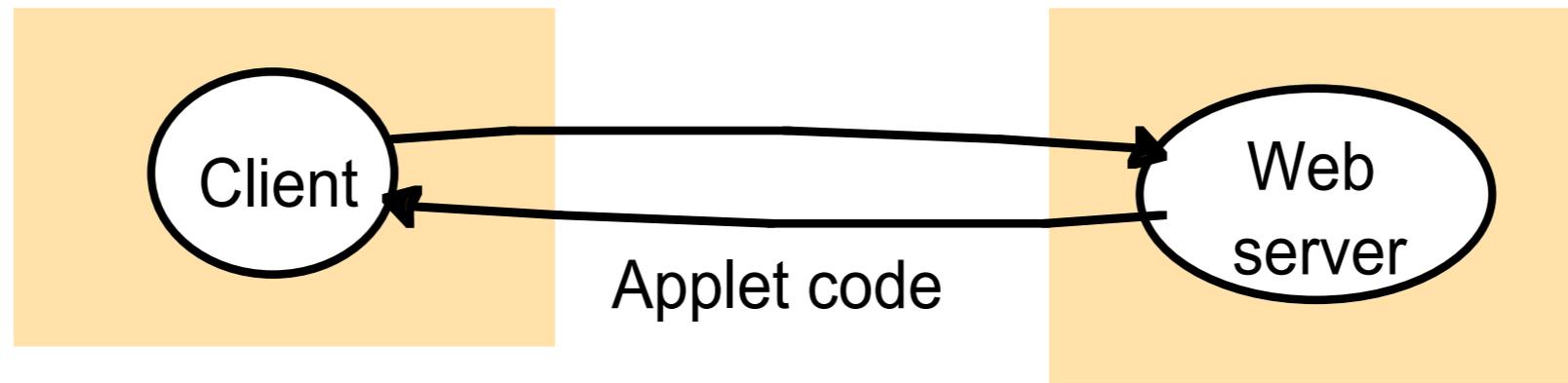
© Pearson Education 2005



Variation: Mobile Code

© Pearson Education 2005

a) client request results in the downloading of applet code



b) client interacts with the applet



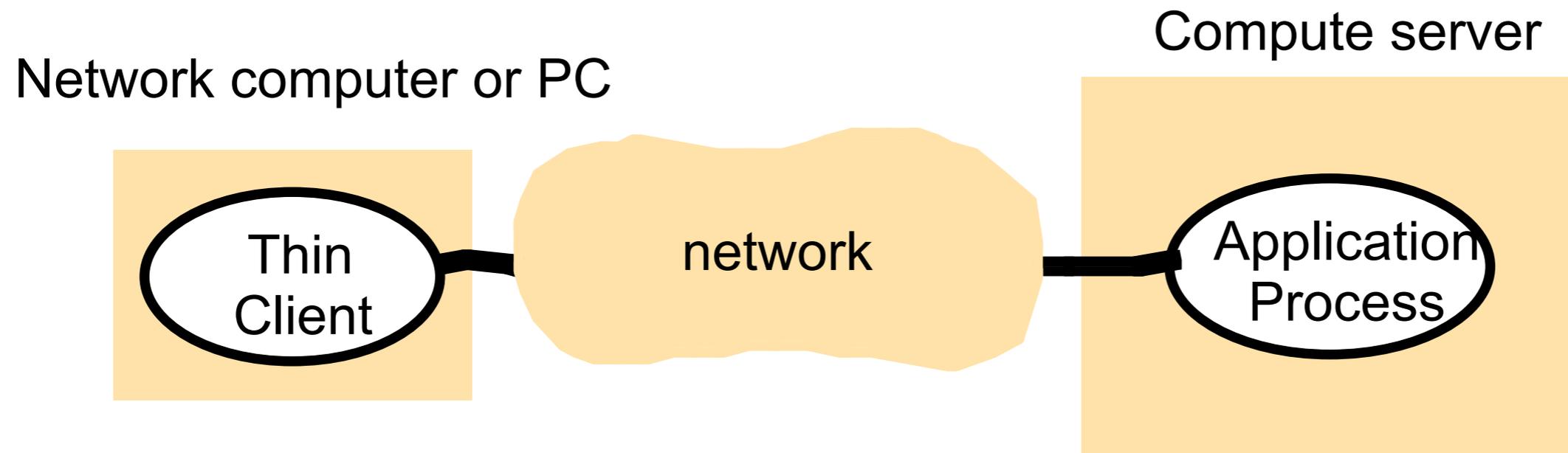
Variation: Mobile Agent

- running program is moved from node to node (with both code and data)
- security challenge to the server: client gets hold of the entire state of the agent
 - only have non-secret data in the agent
- security challenge to the client: client node runs arbitrary server-defined code
 - need to establish trust in agent code
 - need to restrict agent's access to local resources (sandboxing)

Variation: Thin Clients

- Client performs just I/O, no computation

© Pearson Education 2005



Interfaces and Objects

- Set of operations offered by a process is defined by its interface
 - more precisely: set of messages it is able to send and receive
- often formally specified in interface definitions
- object-oriented middleware applications: interfaces get implemented by classes

Design Requirements for Distributed Architectures

- Various objectives for creation of distributed systems
 - sharing of computational resources (e.g. cluster computing)
 - sharing of data
 - sharing of services
- Performance issues: responsiveness, throughput, load balancing
- Quality of Service (QoS): reliability, security, performance, adaptability, time-critical data
- Dependability: correctness, security, fault tolerance, (maintainability)
- Caching and Replication

Fundamental Models

- Model: abstraction of essential properties of a natural phenomenon, for the purpose of understanding and analysis
 - make explicit all relevant assumptions
 - make generalizations concerning what is possible or impossible
- Models of distributed systems: Reasoning about
 - Interaction (e.g. communication involves delays)
 - Failure (node and network failure threatens correct operation of system)
 - Security (consider attacks by both internal and external agents)

Interaction

- Notion of distributed algorithm:
 - each node has a set of data, and runs a program
 - state of each node not accessible to any other node
 - nodes interchange messages
 - assume that all activity in the system is driven by message reception (may need to consider system boundaries specially)
- relative speed of nodes should assumed to be unknown
- performance of network: latency, bandwidth, jitter
- each node has its own internal clock
 - with specific drift rate - clock synchronization is necessary

Interaction (cntd.)

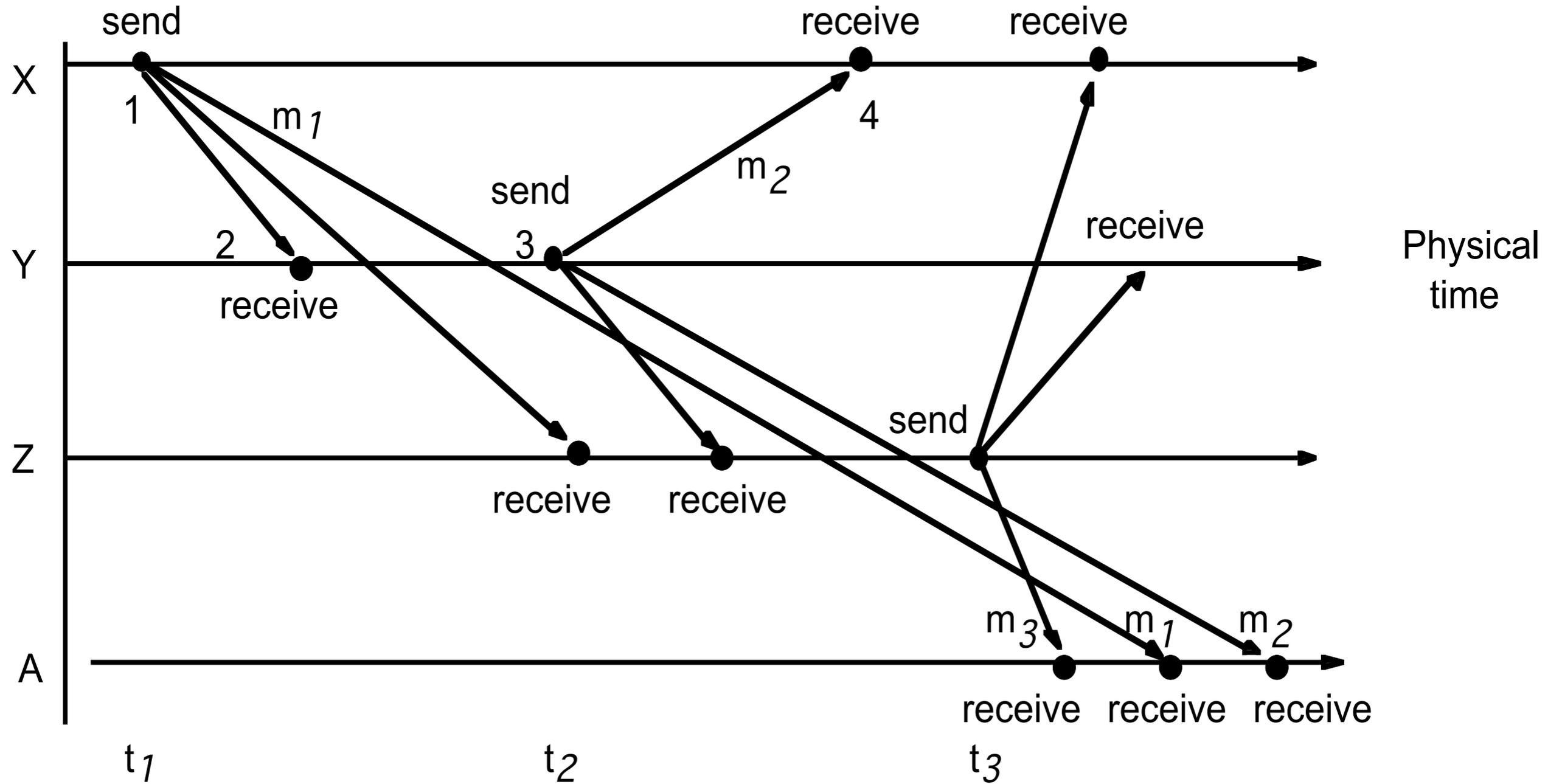
- two variants of interaction: synchronous and asynchronous
- synchronous systems:
 - known lower and upper time bound for each execution step, for each message transmission, and for the clock drift
 - consequence: can introduce a pulsed execution system
 - practically difficult to build, may help in simplifying analysis
- asynchronous systems: messages can arrive and be sent at any time
- event ordering: can usually assume no relative order of reception wrt. sending of messages
 - exception: messages sent on an order-preserving channel

Interaction: Lamport's "Logical Time" (1978)

- happens-before relation: e_1 happens before e_2 , iff
 - e_1 is executed by the same process before e_2 , or
 - e_1 is a send operation, and e_2 is the corresponding receive operation, or
 - there is an e_3 such that e_1 happens before e_3 , and e_3 happens before e_2
- logical time: Assign a number L to each event, such that $L(e_1) < L(e_2)$ if e_1 happens before e_2

Logical Time (cntd.)

© Pearson Education 2005



Failure Model

- Omission Failures: process or channels fails to perform an operation
 - process omission failure (e.g. crash failure)
 - fail-stop: other processes can detect crash (requires guaranteed delivery of messages)
 - communication omission failures (message drop)
- Arbitrary Failures (Byzantine failure): anything may happen
- Timing Failures:
 - synchronous system: activities not completed within pulse
 - real-time systems: activity not completed within promised time

Failure Model (cntd.)

- Masking failures: reconstruct reliable services on top of unreliable ones
 - through retries, error correction, ...
- Reliability of one-to-one communication:
 - validity (messages are eventually delivered to the receiver)
 - integrity (received message identical to sent one, and no message is delivered twice)

Security Model

- securing processes and channels against unauthorized access
- protecting objects: access rights given to a principal
- assumption of an enemy (aka adversary), capable of (threat model)
 - sending messages to any process
 - reading and copying any message between a pair of processes
- enemy may operate either legitimately-connected node, or illegal node

Security Model (cntd.)

- threat to processes: may receive messages sent by enemy
 - may not be able to reliably determine identity of sender
 - server: may not be able to identify principal
 - client: may fall to "spoofing"
- Threats to communication channels: enemy may
 - copy, alter, inject, or delete messages
 - gain information only intended for the communication partner
- Other threats: denial of service, trojan horses, ...
- Defeating security threats: cryptography, authentication, secure channels