# COMP28112 Lecture 3

- **Architectures of distributed systems**
- **Fundamental Models**

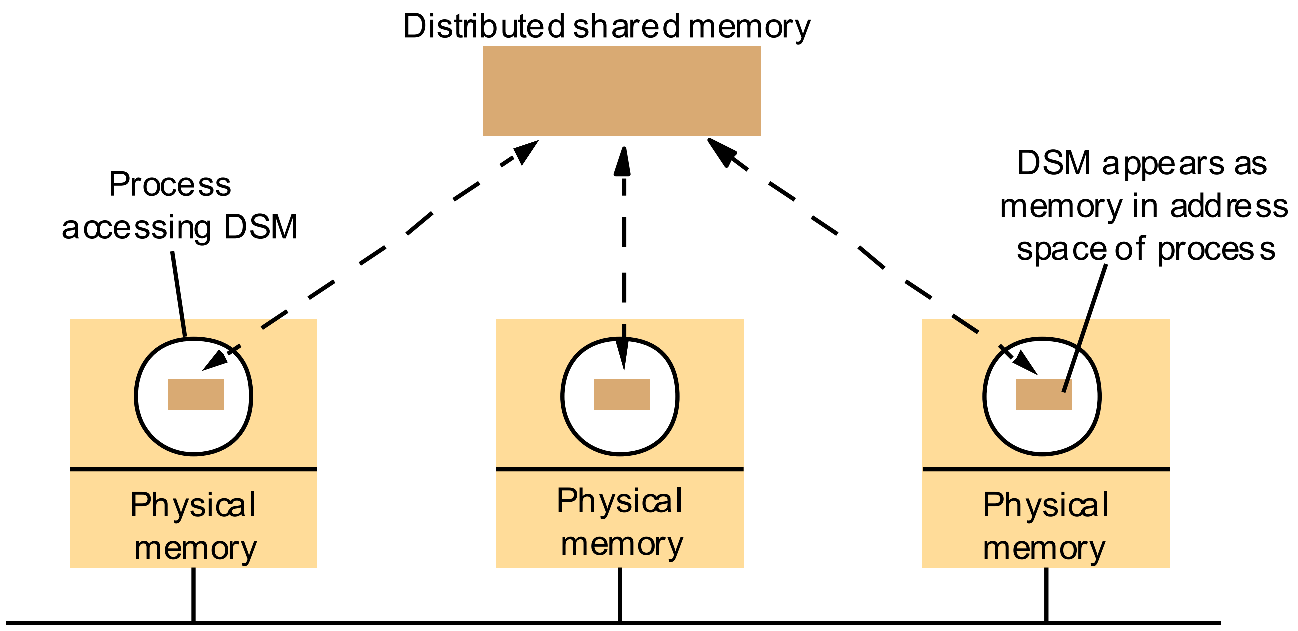
# Architectures of Distributed Systems

- **Tightly coupled**
  - Highly integrated machines that may look as a single computer.
- **Loosely coupled** (share nothing)
  - Client-Server
  - Peer-to-Peer
  - A Key differentiation based on the Programming Interface:
    - Distributed Objects
    - Web Services
    - Read "Web Services are not Distributed Objects", Internet Computing, Nov-Dec. 2003, http://www.allthingsdistributed.com/historical/archives/000343.html

# Tightly coupled systems

- **<u>Distributed shared memory</u>** (DSM)provides the illusion of a single shared memory: it spares the programmer the concerns of message passing.

Figure 18.1
The distributed shared memory abstraction


Distributed shared memory
Process accessing DSM
DSM appears as memory in address space of process
Physical memory
Physical memory
Physical memory

## Issues with Distributed Shared Memory

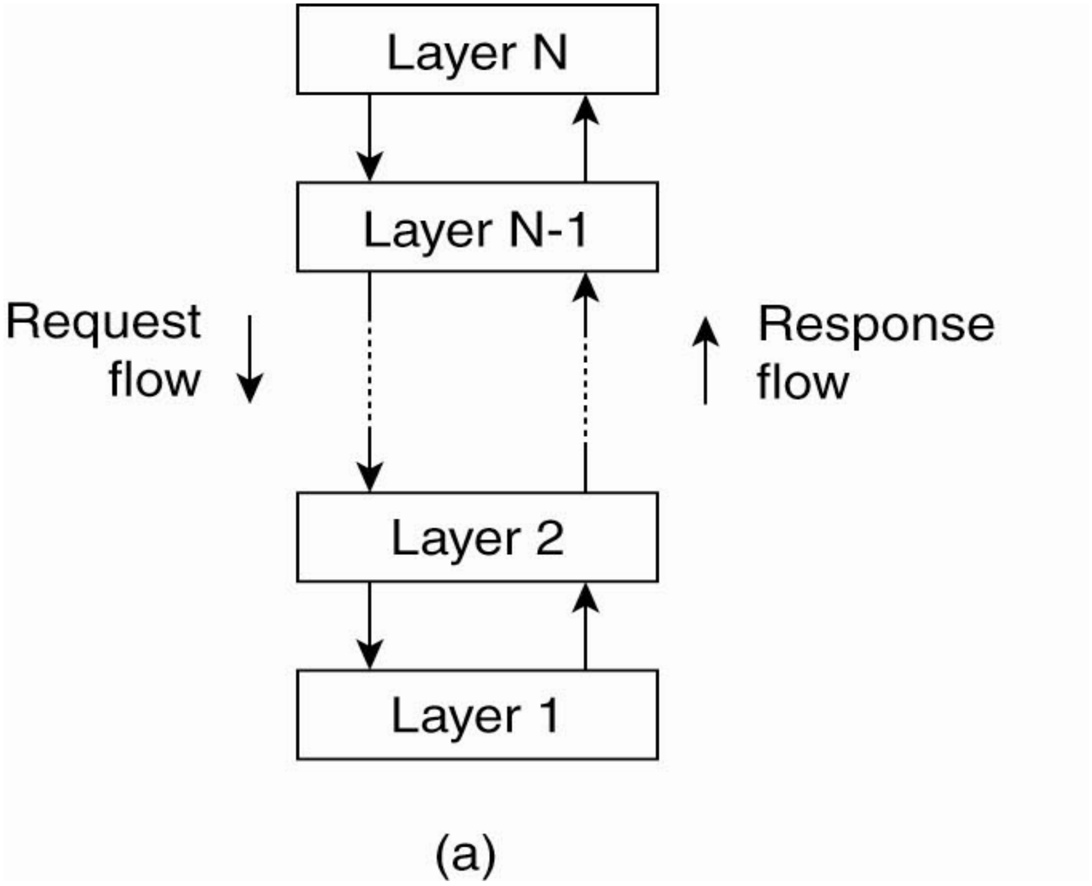
- Machines are still connected by a network:
  - Minimize network traffic
  - Reduce the latency between request and completion
- How to keep track of the location of shared data
- How to overcome delays when accessing remote data
- How to make shared data concurrently accessible
- Replicate shared data on multiple machines:
  - Need memory coherence
- Lots of research in the context of building parallel computers – see Chapter 18 in Coulouris *et al* book.

## Architectural Styles
(talking about loosely-coupled systems)

Based on the logical organization of the components of distributed systems:

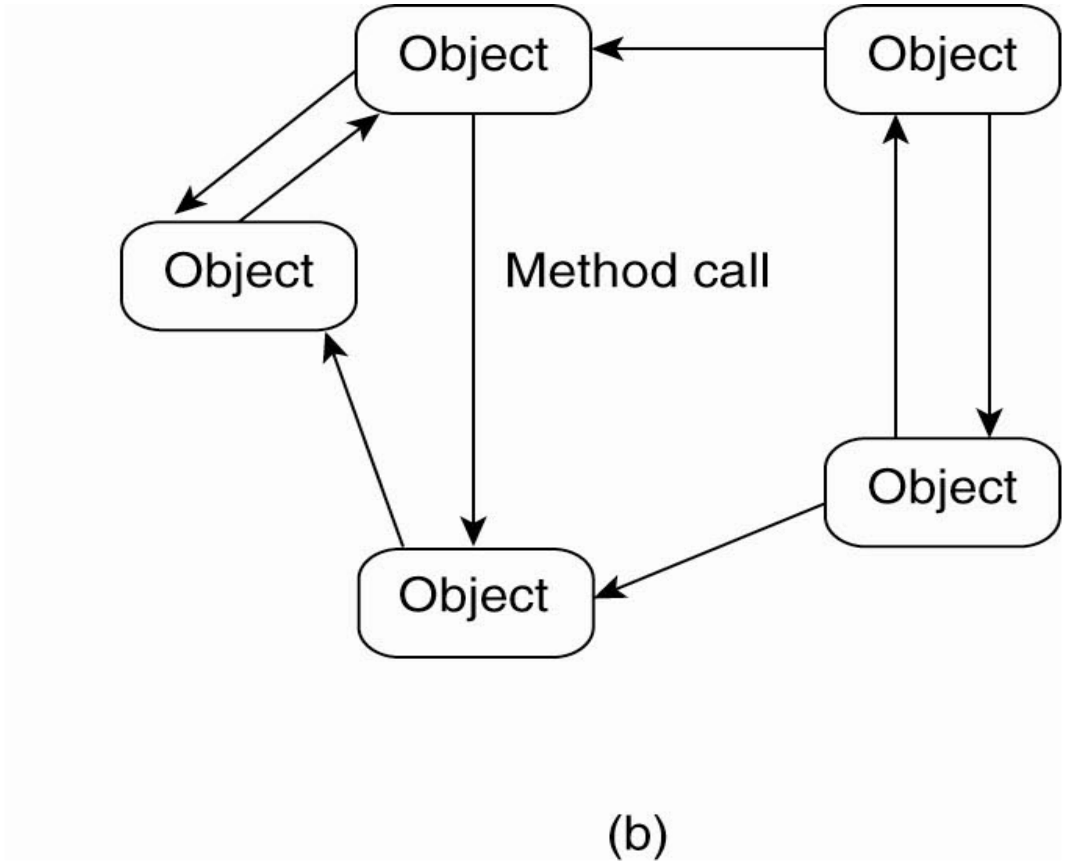
- Layered architectures
- Object-based architectures
- Data-centered architectures
- Event-based architectures

# The layered architectural style



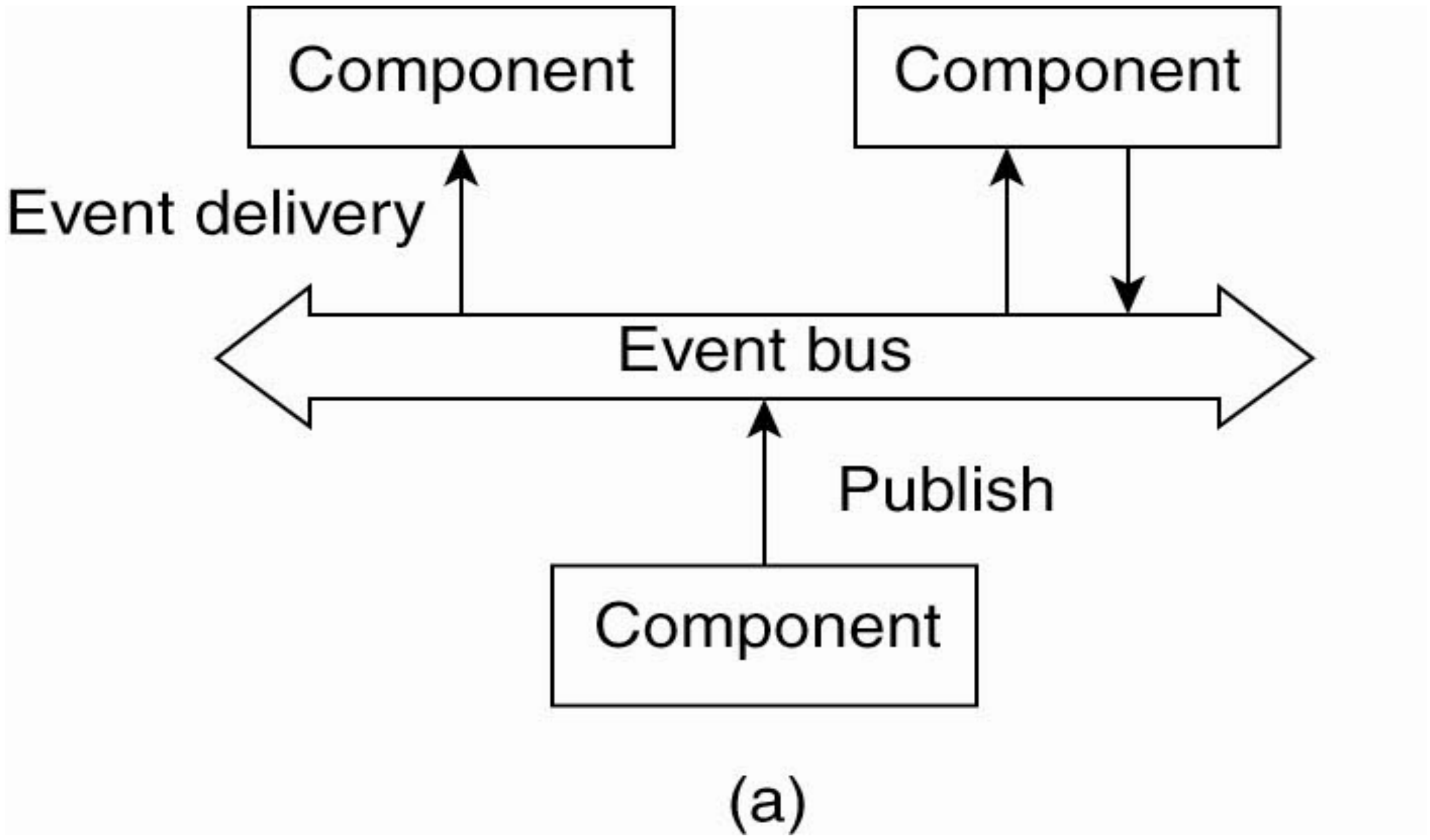
See Figure 2-1 in Tanenbaum and Van Steen book

# The object-based architectural style



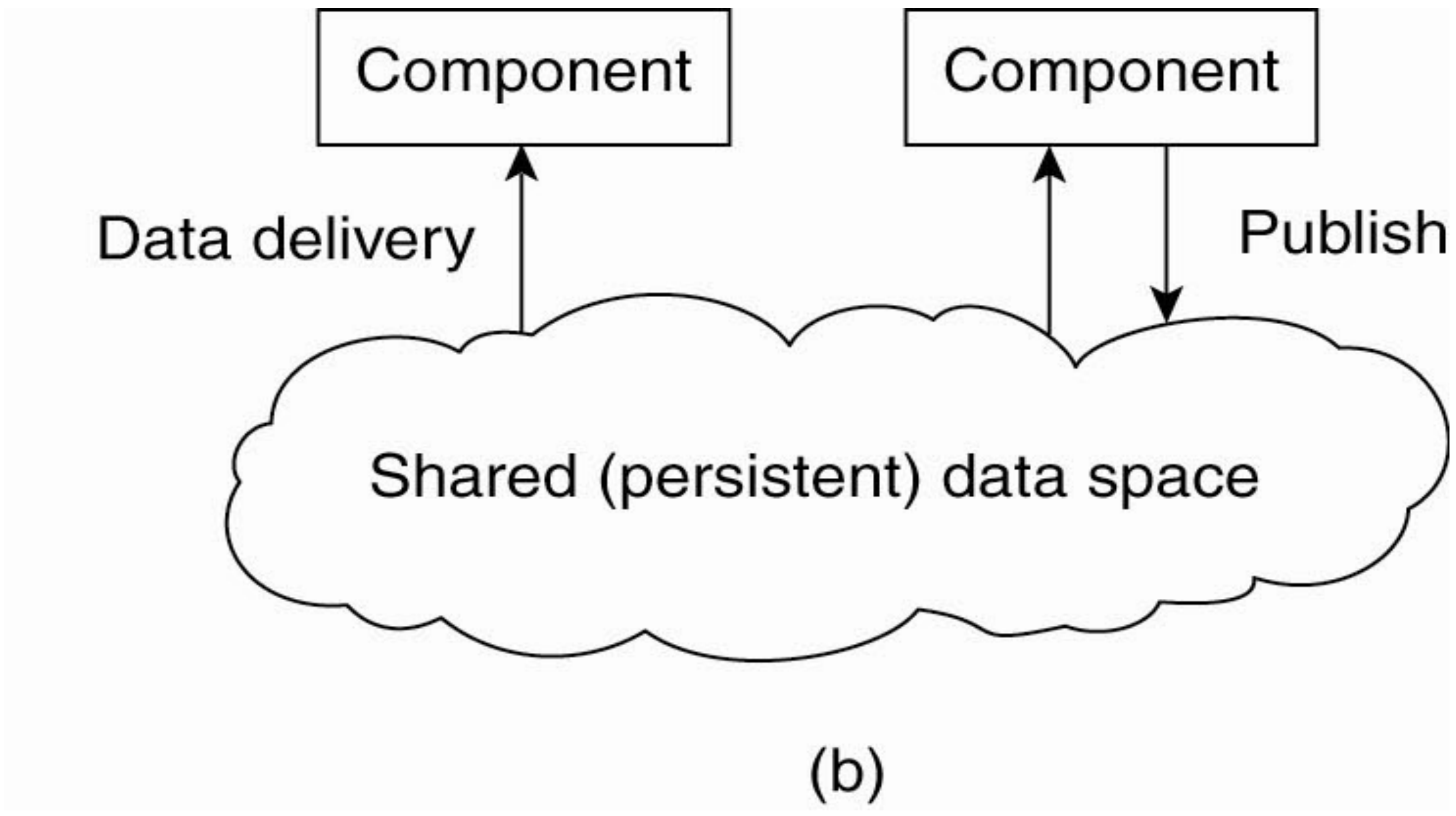
See Figure 2-1 in Tanenbaum and Van Steen book

## The Event-Based architectural style



- See Figure 2-2 in Tanenbaum and Van Steen book

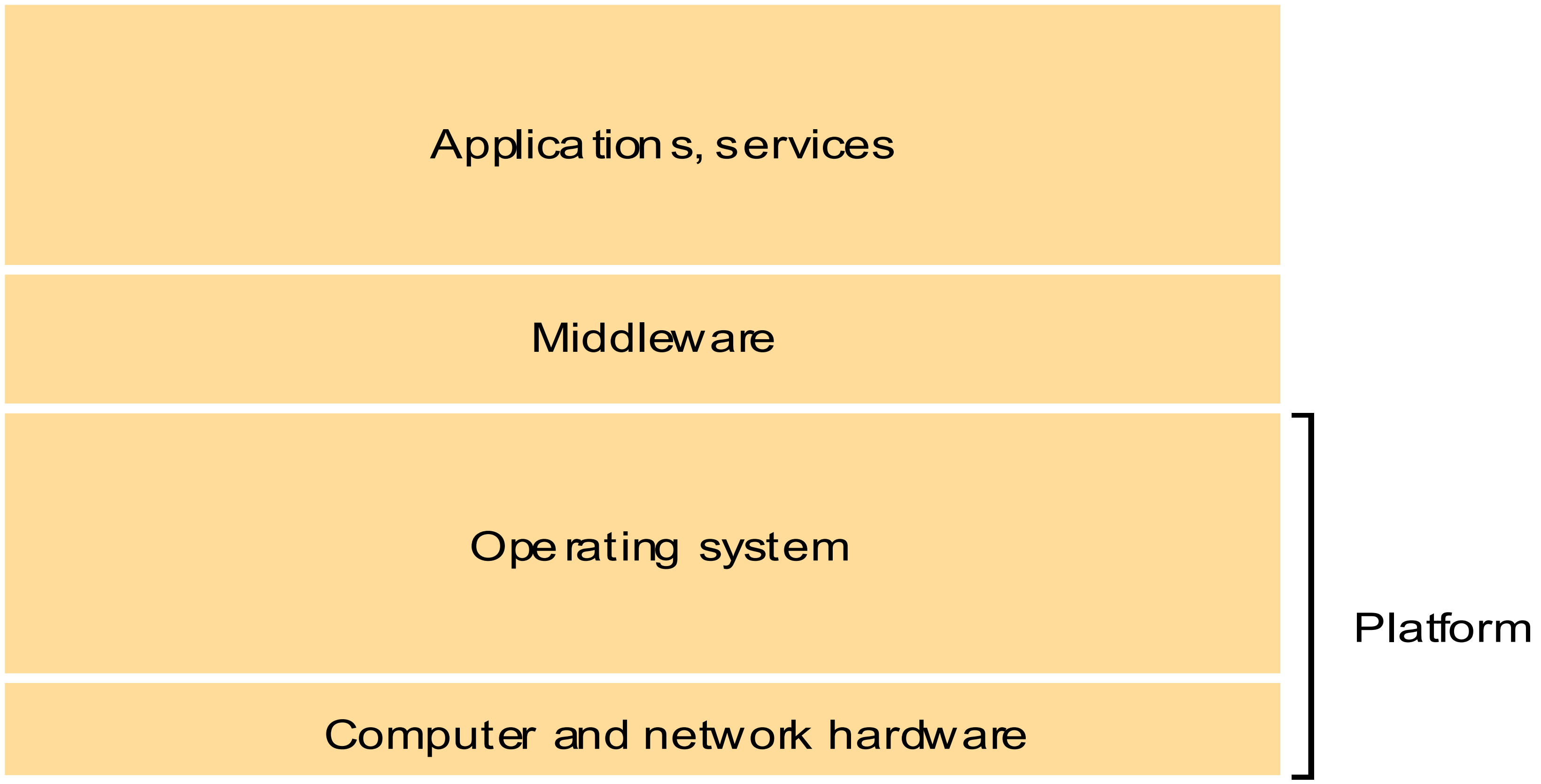
## Shared-Data Space Architectural Style



- See Figure 2-2 in Tanenbaum and Van Steen book

# System Architectures of Distributed Systems

Figure 2.1
Software and hardware service layers in distributed systems



## Middleware

- **Middleware**: a software layer that provides a programming abstraction to mask the heterogeneity of the underlying platforms (networks, languages, hardware, …)
  - E.g., CORBA, Java RMI
- But… the end-to-end argument implies that some aspects of communication support cannot always be abstracted away from applications.

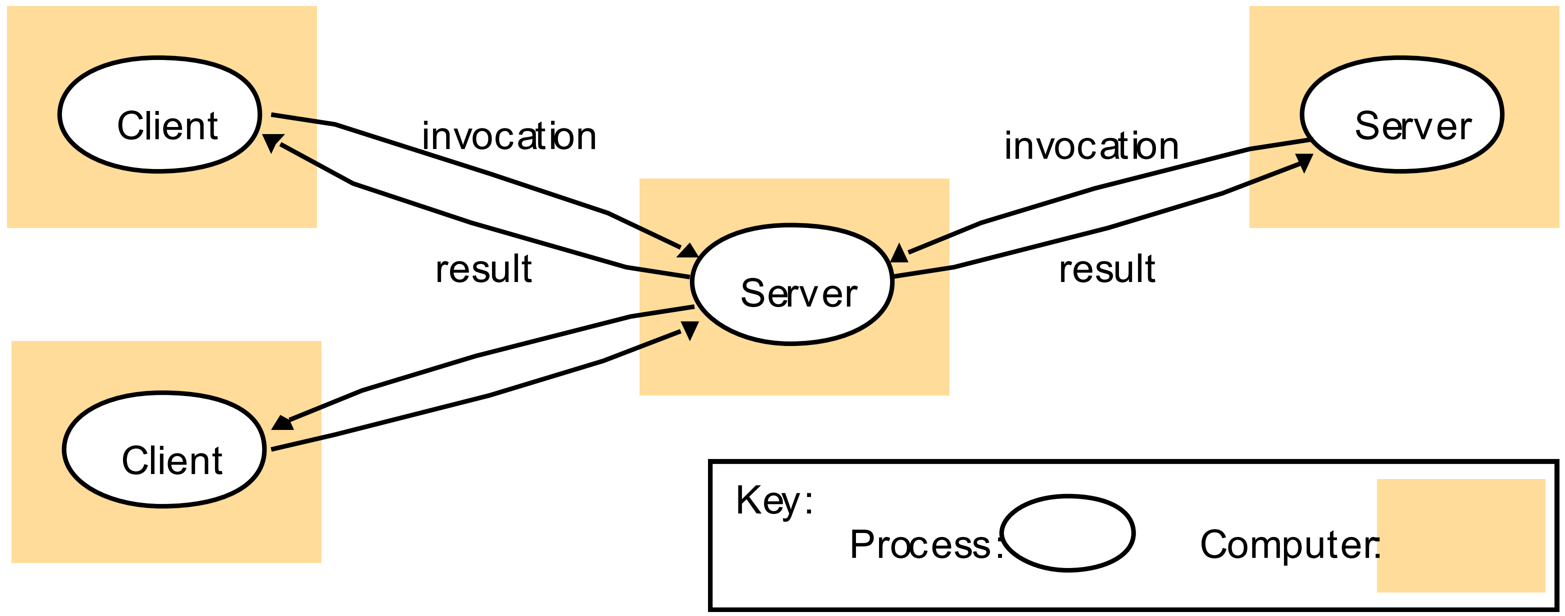
## The end-to-end argument

*“The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)”*

- See Coulouris pages 33-34 and http://www.reed.com/dpr/locus/Papers/EndtoEnd.html

# Client-Server Model

Figure 2.2
Clients invoke individual servers

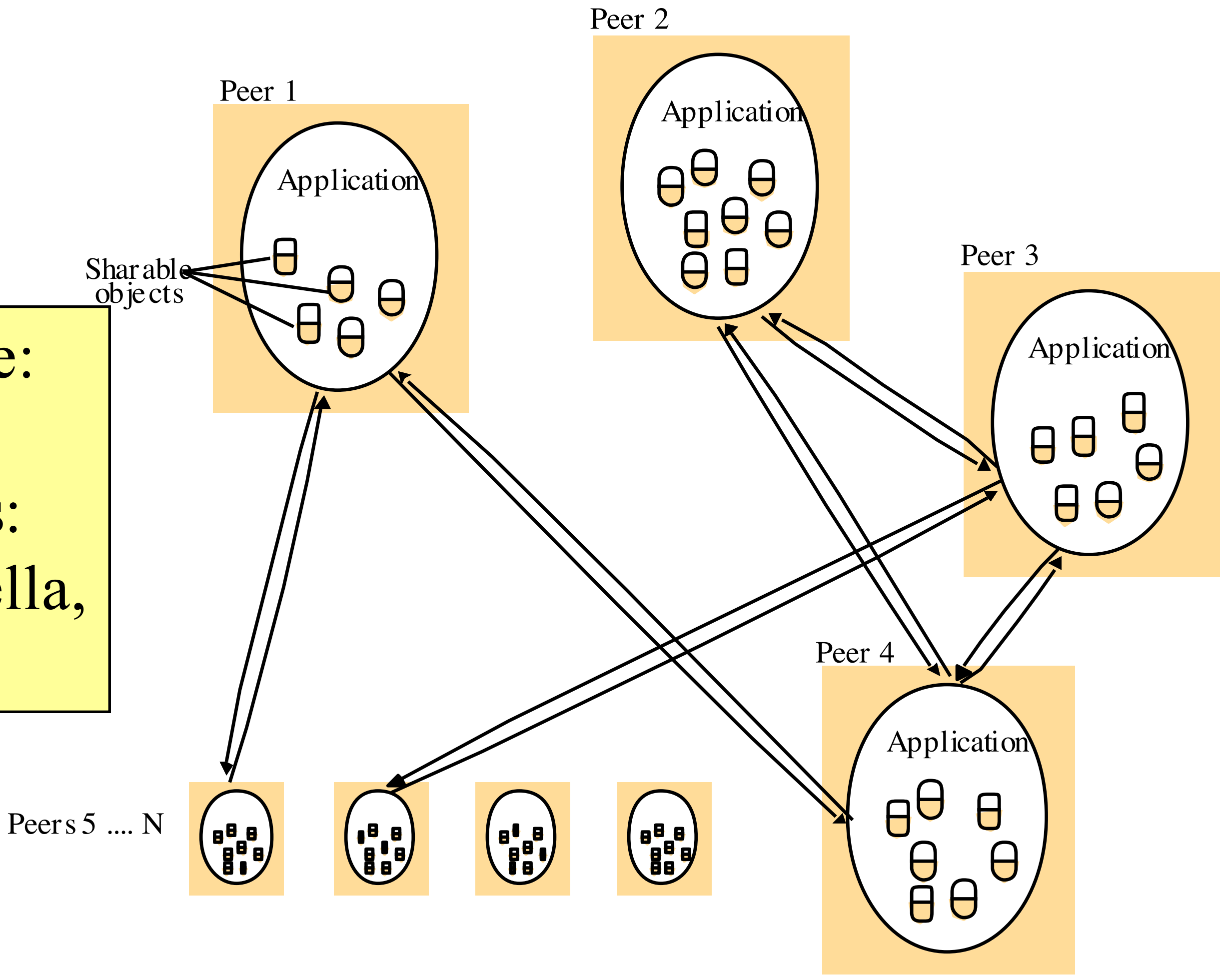


## Client-Server Model

- Characteristics of a server:
  - Passive (slave)
  - Waits for requests
  - Upon receipts of requests, it processes them and sends replies
  - Can be stateless (does not keep any information between requests) or stateful (remembers information between requests)
- Characteristics of a client:
  - Active (master)
  - Sends requests
  - Waits for and receives server replies

# Peer-to-Peer (P2P)

Figure 2.3
A distributed application based on peer processes



Earlier Example:
- Usenet News

Other examples:
- Napster, Gnutella, Freenet, …

# Client-Server vs P2P

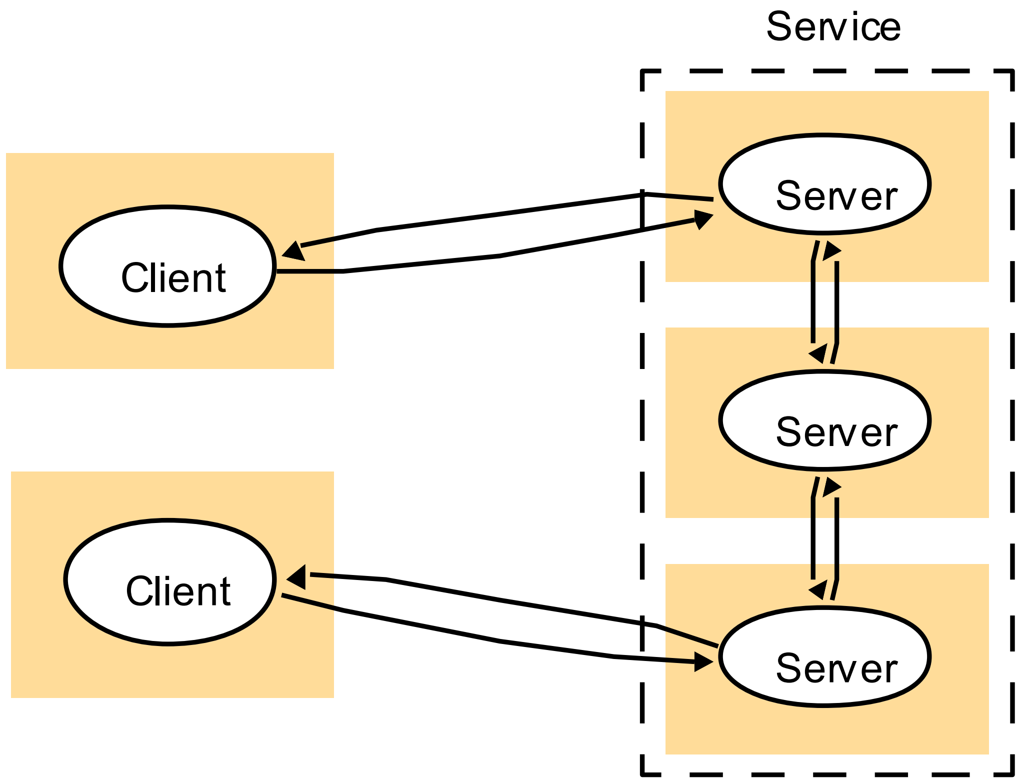
- Client-Server
  - Widely Used
  - Functional Specialisation
  - Asymmetrical
  - Tends to be centralised
  - Tends to scale poorly
- P2P
  - Symmetrical, computers have same "rights"
  - Truly Distributed
  - Share / exploit resources with a large number of participants
  - Resource discovery is a challenge

## Variations on a theme…

- Sometimes we need to consider:
  - The use of multiple servers to increase performance and resilience
  - The use of mobile code
  - Users' need for low-cost computers
  - Requirements to add and remove mobile devices

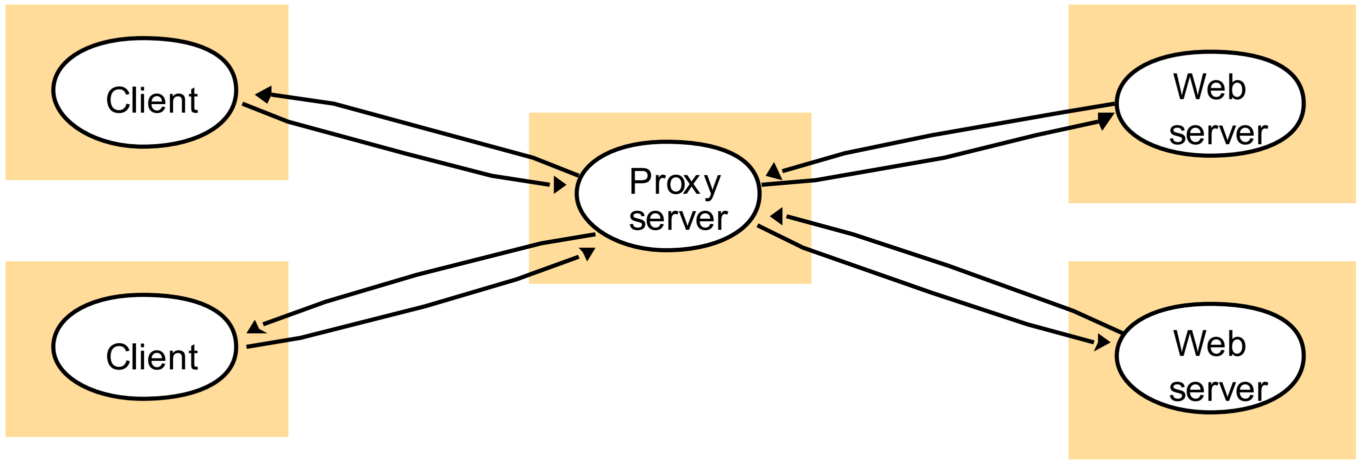
# Multiple Servers

Figure 2.4
A service provided by multiple servers



# Proxies and Caches

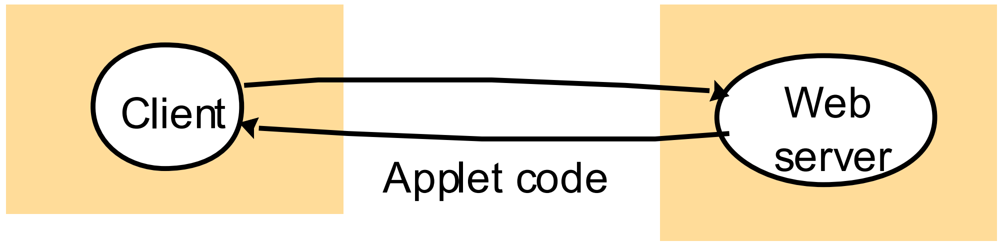
Figure 2.5
Web proxy server



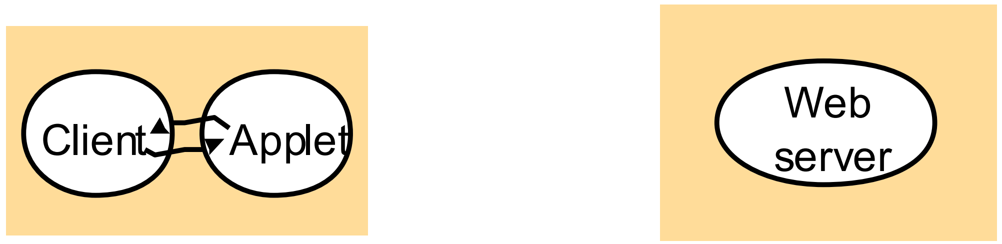
# Mobile Code…

Figure 2.6
Web applets

a) client request results in the downloading of applet code

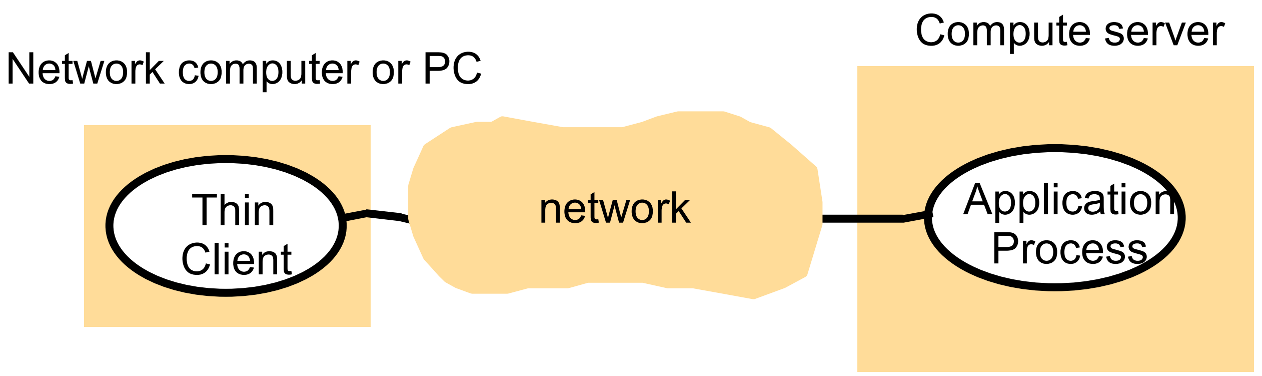


b) client interacts with the applet



# Thin Clients

Figure 2.7
Thin clients and compute servers



## Design Requirements

- Performance
  - Responsiveness
  - Throughput
  - Load balancing
- Quality of Service
- Caching and Replication
- Dependability
  - Correctness
  - Security
  - Fault-Tolerance

## Fundamental Models

- **Models include all the essential ingredients that we need to consider in order to understand and reason about some aspects of a system's behaviour.**
- **A way of abstracting from reality.**
- Aspects that we need to capture include:
  – Process Interaction
  – Failure
  – Security

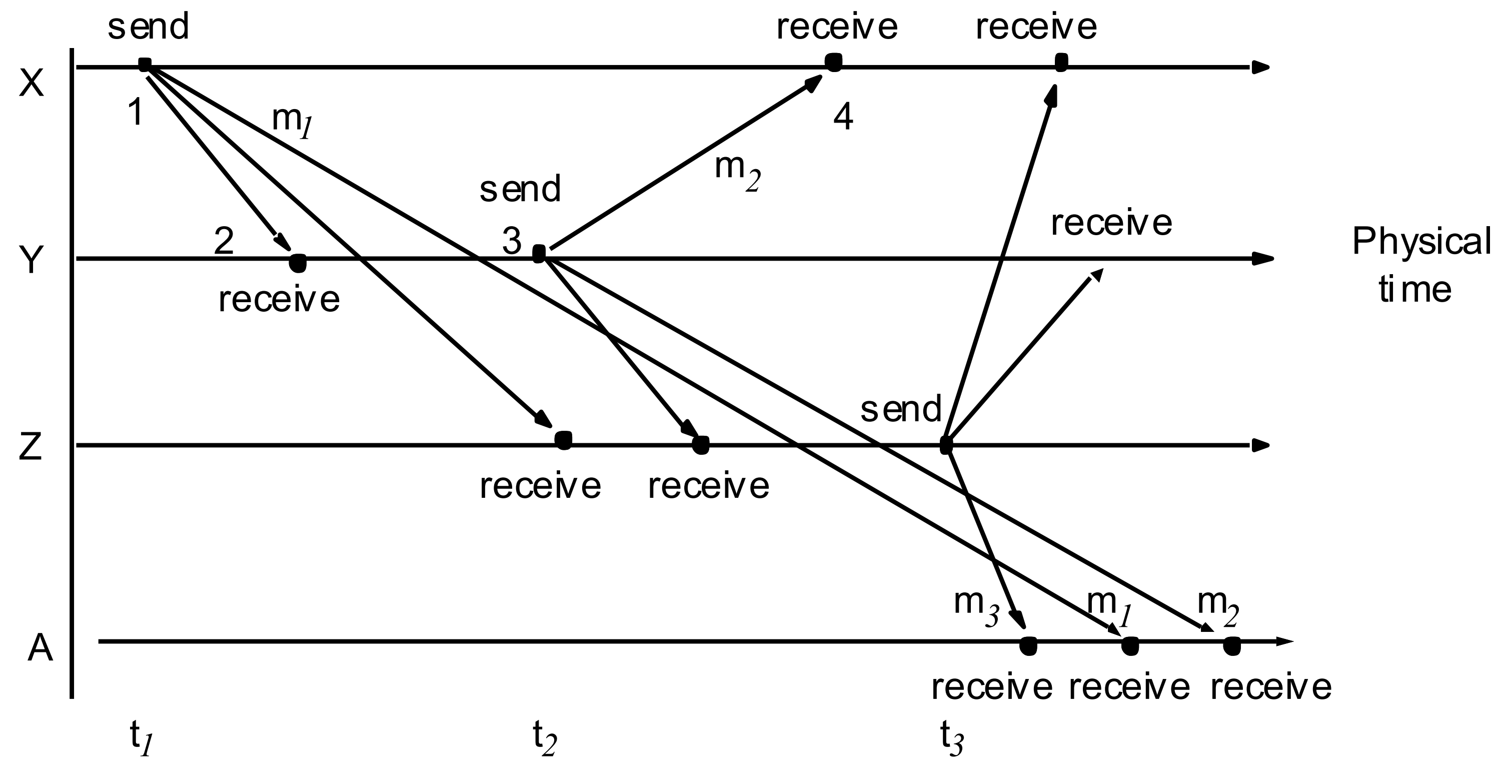
***<u>Managing concurrency and failure underlies many of the problems we face!</u>***

## Factors affecting process interaction

- Performance of communication channels
- Computer Clocks and Timing Events
- Two variants of the process interaction model:
  - Synchronous systems:
    - Process Execution Speeds: has known lower and upper bound
    - Message transmitted delay: bounded
    - Clock drift rates: drift rate from real time has a known bound
  - Asynchronous systems:
    - All the above may take an arbitrarily long time.

# Event Ordering

Figure 2.8
Real-time ordering of events



## Failures in distributed systems

- What can go wrong?
  - (see Figure 2.10 in Coulouris)
- How to mask failures?
  - Read about checksums in network protocols

# In Summary...

- Different Architectures have different solutions
- Managing **<u>Concurrency</u>**, **<u>Failures</u>**, and **<u>Security</u>** underlies many of the problems we face.
- Reading:
  - Coulouris (4th or 5th edition), Chapter 2; Tanenbaum, Sec. 2.1, 2.2 (skim through the rest of Chapter 2)
  - Read the text associated with the figures from Coulouris *et al* and Tanenbaum *et al* textbooks that are shown in this handout.
- Next time: Remote procedure calls.