Distributed Systems

Assistant Prof. Spyros Voulgaris, Ph.D.

Course material based on:
- Lecture notes by Prof. Abraham Bernstein
- Tanenbaum & van Steen, 2007
- Coulouris et al, 2005

Dynamic and Distributed Information Systems

Today's Agenda

- Introduction
- Challenges in Distributed Systems
- Hardware Architectures
- Software Architectures
- Types of Network Interactions
- Course Outline

Introduction

Evolution of Networks

- In the early times, computers were standalone devices
- In the late 80's / early 90's, computer networks started spreading
  - The Internet brought a revolution to the way of life on the planet!
- Today networking has become a fundamental part of computers
  - …if not the most important one!
  - We cannot even think of computers are isolated units
  - E.g., even the extremely low budget $100 laptop (OLPC --- One Laptop Per Child project) is designed to be networked
- Shift from standalone computers, to the paradigm of computers communicating, interacting, collaborating with each other.
- Managing Distributed Systems at such a large and complex scale is not a trivial task

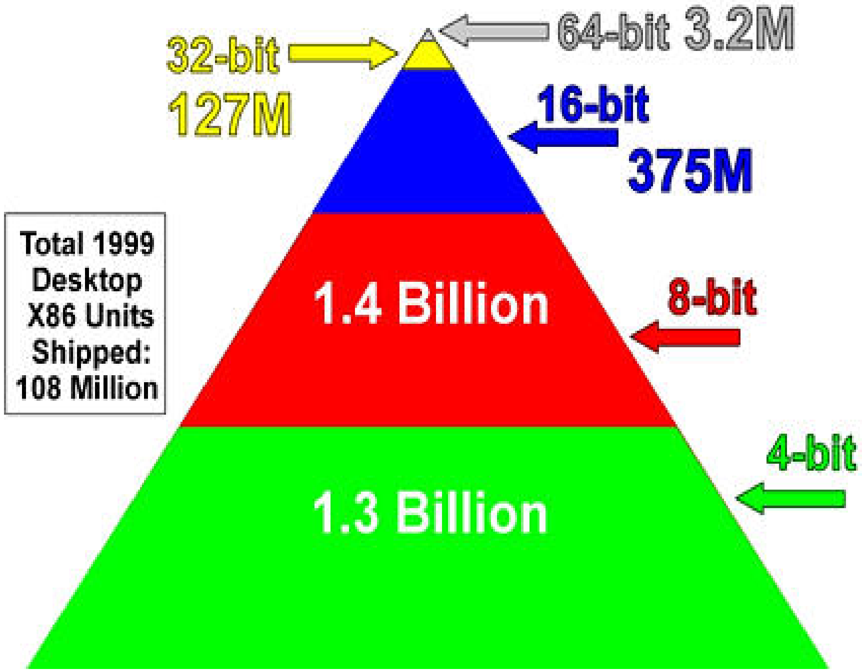
The scale is very large…

Date	*Computers*	*Web servers*	*Percentage*
1993, July	1,776,000	130	0.008
1995, July	6,642,000	23,500	0.4
1997, July	19,540,000	1,203,096	6
1999, July	56,218,000	6,598,697	12
2001, July	125,888,197	31,299,592	25
2003, July		42,298,371	

- Increasing number of computers
- Increasing number of distributed applications
- Needs are increasingly
  - *complex*
  - *larger scale*
  - *application-specific*

Figure 1.6

But the real scale is even larger!



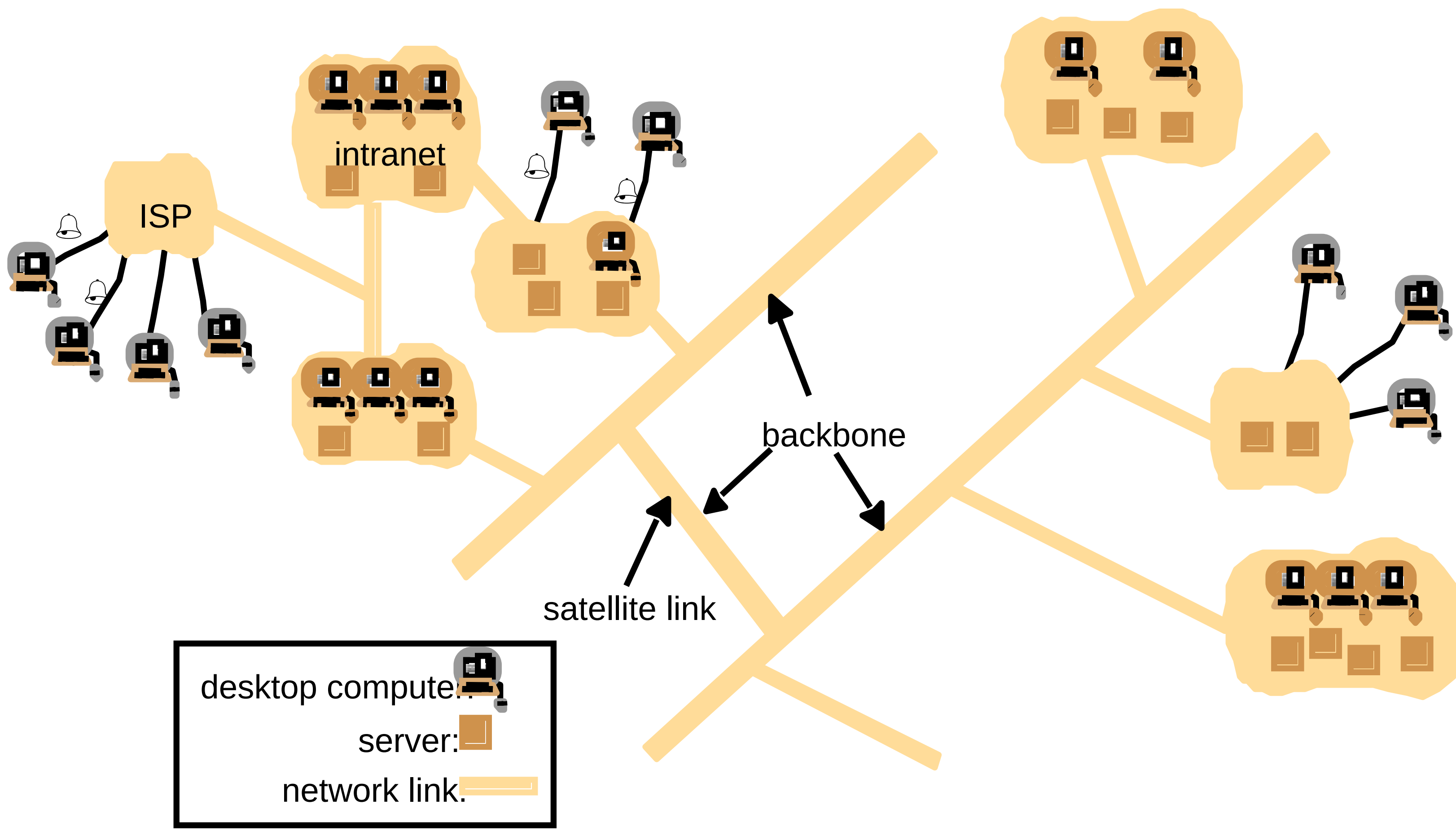
Quellen: http://www.linuxdevices.com/cgi-bin/printerfriendly.cgi?id=AT9656887918
http://www.embedded.com/1999/9905/9905turley.htm

Definition ???

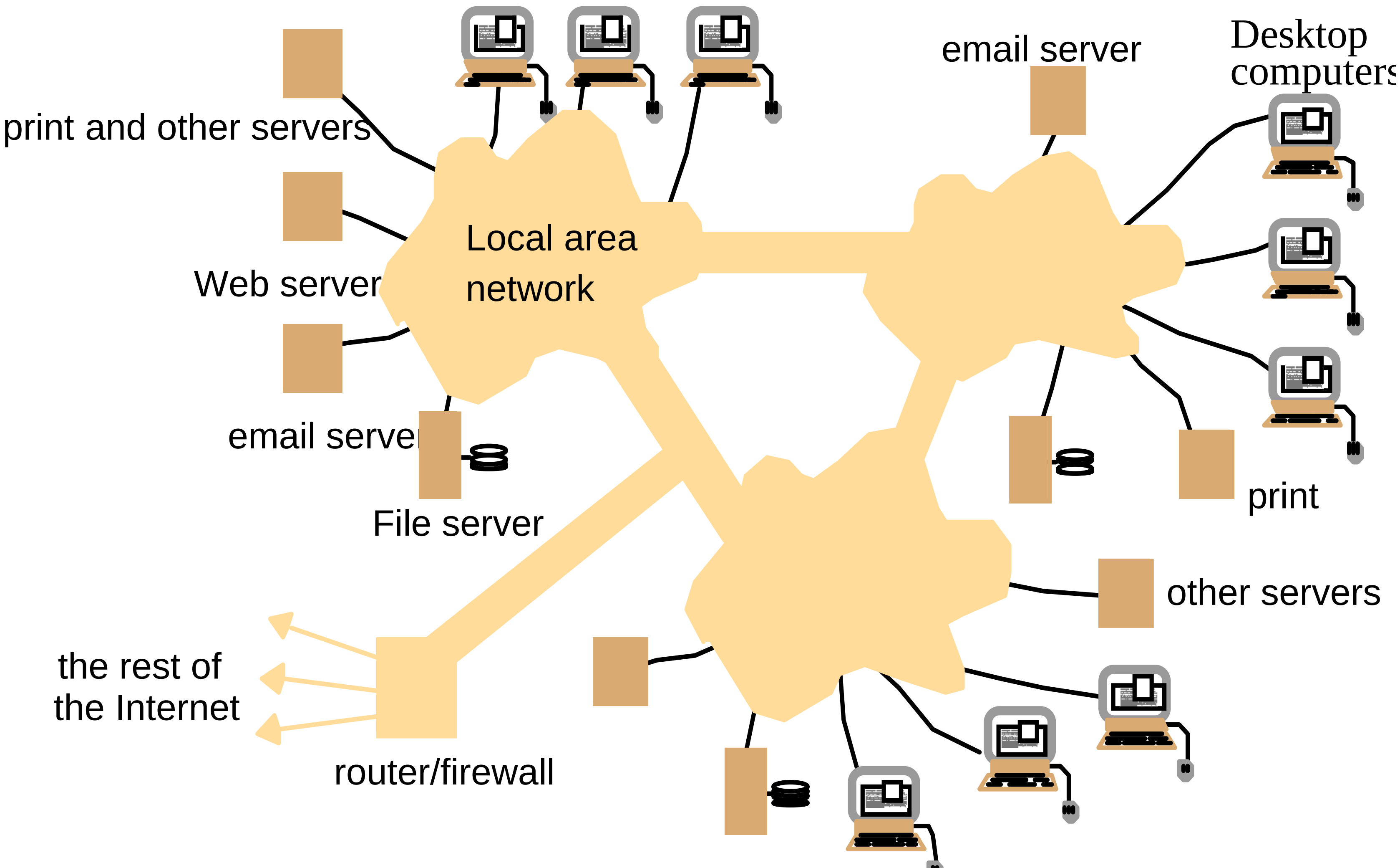
Not trivial…

…Distributed Systems come in really different flavors !

A typical portion of the Internet



A typical intranet



Portable and handheld devices in a DS

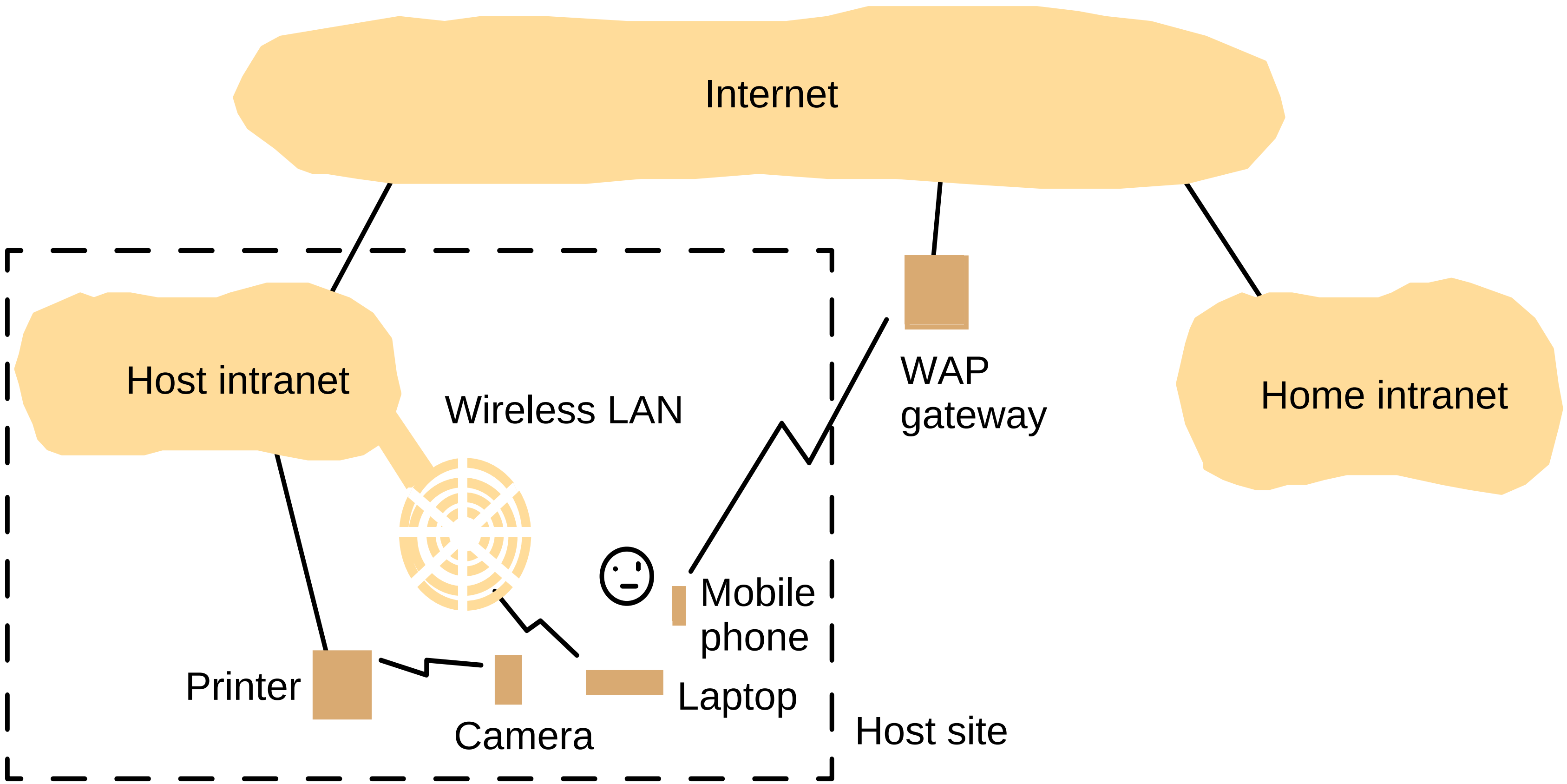


Figure 1.3

Definition of a Distrib. System (1/2)

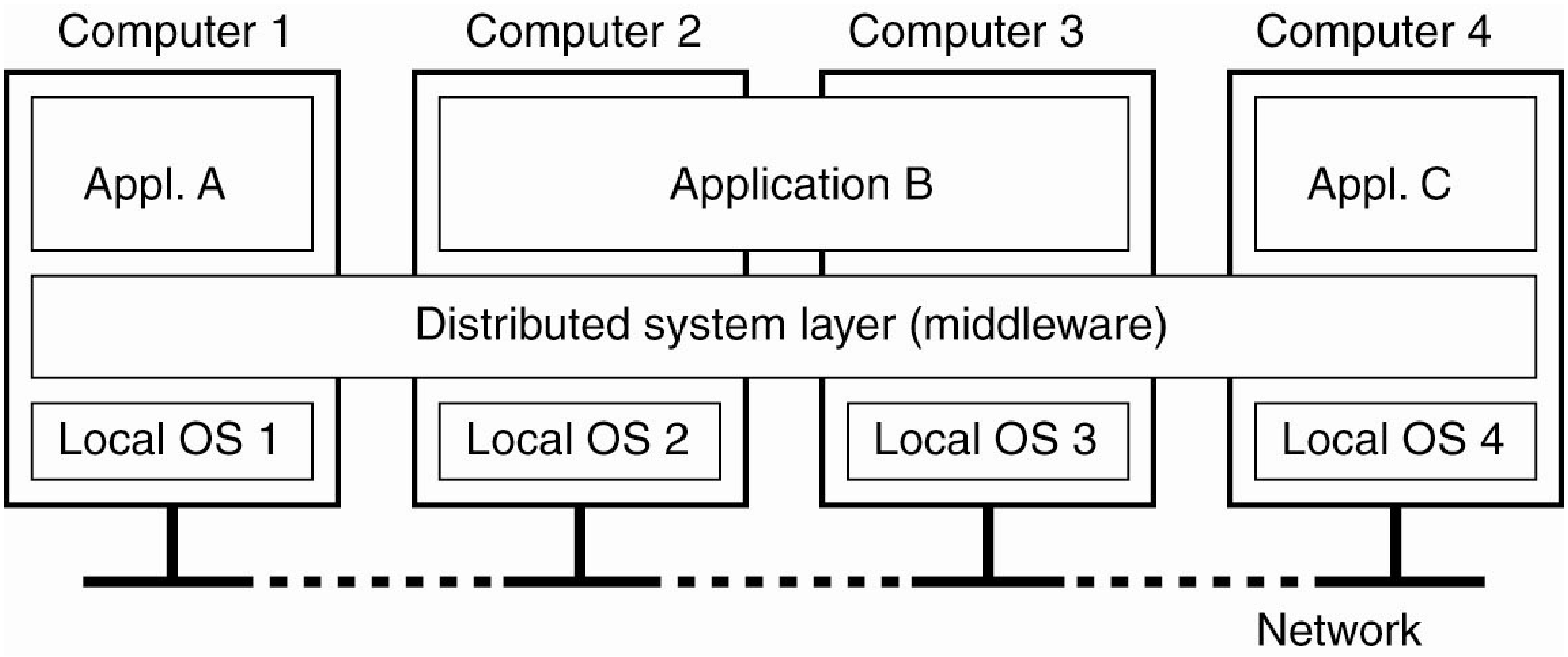
A distributed system is:

"A collection of independent computers that appears to its users as a single coherent system."

Two notions to be noted in this definition:

- HARDWARE: The machines are autonomous
- SOFTWARE: The users think they deal with a single system

Definition of a Distrib. System (2/2)



A distributed system organized as middleware.
Note that the middleware layer extends over multiple machines.

Challenges in Distributed Systems

Challenges in Distributed Systems

- Transparency
  - Single view of the system
  - Hide numerous details
- Heterogeneity
  - Networks
  - Computers (HW)
  - Operating systems
  - Programming languages
  - Developers
- Failure Handling
  - Detecting
  - Masking
  - Tolerating
  - Recovery
  - Redundancy
- Openness
  - Extensibility
  - Publication of interfaces
- Scalability
  - Controlling the cost of resources
  - Controlling the performance
  - Preventing resources from running out
  - Avoiding performance bottlenecks
- Security
  - Secrecy
  - Authentication
  - Authorization
  - Non-repudiation
  - Mobile code
  - Denial of service

Challenges: **Transparency**

"A collection of independent computers that appears to its users as a single coherent system."

- Make a set of computers appear as a single computer to the applications
- Provide abstractions, to facilitate application development
- Hide the details, and deal with them transparently
- Transparency comes in *many* different flavors…

Challenges: **Transparency**

(by Tanenbaum & van Steen)

Transparency type	*Description*
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located Resources accessed without knowledge of their location (e.g., IP addr)
Migration	Hide that a resource may be moved to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that there may exist multiple replicas of a given resource
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

Challenges: **Transparency**

- Access transparency
  - Different data representations
    - e.g., Sun SPARC uses little endian representation, and Intel big endian
  - Different conventions
    - e.g., Linux filesystem is case-sensitive, Windows is not
- Location / Migration / Relocation transparency
  - Instant Messaging: no knowledge where your buddies are located
  - Transparent handover when changing wireless access point
- Replication transparency
  - Hotmail / Facebook / CNN / your bank / [you_name_it] maintain their data *transparently replicated* for increased availability and improved access time

Challenges in Distributed Systems

- Transparency
  - Single view of the system
  - Hide numerous details
- Heterogeneity
  - Networks
  - Computers (HW)
  - Operating systems
  - Programming languages
  - Developers
- Failure Handling
  - Detecting
  - Masking
  - Tolerating
  - Recovery
  - Redundancy
- Openness
  - Extensibility
  - Publication of interfaces
- Scalability
  - Controlling the cost of resources
  - Controlling the performance
  - Preventing resources from running out (IP addr)
  - Avoiding performance bottlenecks
- Security
  - Secrecy
  - Authentication
  - Authorization
  - Non-repudiation
  - Mobile code
  - Denial of service

Challenges: **Heterogeneity**

- *Access Transparency* is dealing with one part of heterogeneity
- Additionally, transparently address the differences in
  - performance
  - capabilities
  - network connectivity
  - etc.
- For instance, in a Personal Area Network (PAN), connecting your desktop, laptop, PDA, and mobile phone, transparently avoid assigning intensive tasks to the mobile phone!

Challenges in Distributed Systems

- Transparency
  - Single view of the system
  - Hide numerous details
- Heterogeneity
  - Networks
  - Computers (HW)
  - Operating systems
  - Programming languages
  - Developers
- Failure Handling
  - Detecting
  - Masking
  - Tolerating
  - Recovery
  - Redundancy
- Openness
  - Extensibility
  - Publication of interfaces
- Scalability
  - Controlling the cost of resources
  - Controlling the performance
  - Preventing resources from running out (IP addr)
  - Avoiding performance bottlenecks
- Security
  - Secrecy
  - Authentication
  - Authorization
  - Non-repudiation
  - Mobile code
  - Denial of service

Challenges: **Failure Handling**

- Leslie Lamport's definition of a DS:

  "You know you have one when the crash of a computer you've never heard of stops you from getting any work done."

- Fundamental points in distributed systems:
  - Reliability
  - High Availability

- Distributed Systems should be failure transparent
  - A failure on some components, should not be fatal (or, ideally even detectable) to the applications.
  - E.g., a failure in a bank server should not prevent you from withdrawing money

Challenges in Distributed Systems

- Transparency
  - Single view of the system
  - Hide numerous details
- Heterogeneity
  - Networks
  - Computers (HW)
  - Operating systems
  - Programming languages
  - Developers
- Failure Handling
  - Detecting
  - Masking
  - Tolerating
  - Recovery
  - Redundancy
- Openness
  - Extensibility
  - Publication of interfaces
- Scalability
  - Controlling the cost of resources
  - Controlling the performance
  - Preventing resources from running out (IP addr)
  - Avoiding performance bottlenecks
- Security
  - Secrecy
  - Authentication
  - Authorization
  - Non-repudiation
  - Mobile code
  - Denial of service

Challenges: Openness

- Well defined interfaces (APIs)
  - Well designed
  - Clearly described (publicly)
- Use of Interface Definition Language (IDL)
- Boost interoperability, portability, extensibility
- Boost modularity
  - The upgrade of one module should not affect the rest

Challenges in Distributed Systems

- Transparency
  - Single view of the system
  - Hide numerous details
- Heterogeneity
  - Networks
  - Computers (HW)
  - Operating systems
  - Programming languages
  - Developers
- Failure Handling
  - Detecting
  - Masking
  - Tolerating
  - Recovery
  - Redundancy
- Openness
  - Extensibility
  - Publication of interfaces
- Scalability
  - Controlling the cost of resources
  - Controlling the performance
  - Preventing resources from running out (IP addr)
  - Avoiding performance bottlenecks
- Security
  - Secrecy
  - Authentication
  - Authorization
  - Non-repudiation
  - Mobile code
  - Denial of service

Challenges: **Scalability**

Date	*Computers*	*Web servers*	*Percentage*
1993, July	1,776,000	130	0.008
1995, July	6,642,000	23,500	0.4
1997, July	19,540,000	1,203,096	6
1999, July	56,218,000	6,598,697	12
2001, July	125,888,197	31,299,592	25
2003, July		42,298,371	

Figure 1.6

Challenges: **Scalability**

- Three notions of scalability:
  - Size scalability
  - Geographic scalability
  - Administrative scalability
- Usually, most systems experience some loss of performance as they scale up in some of these dimensions

Challenges: **Scalability**

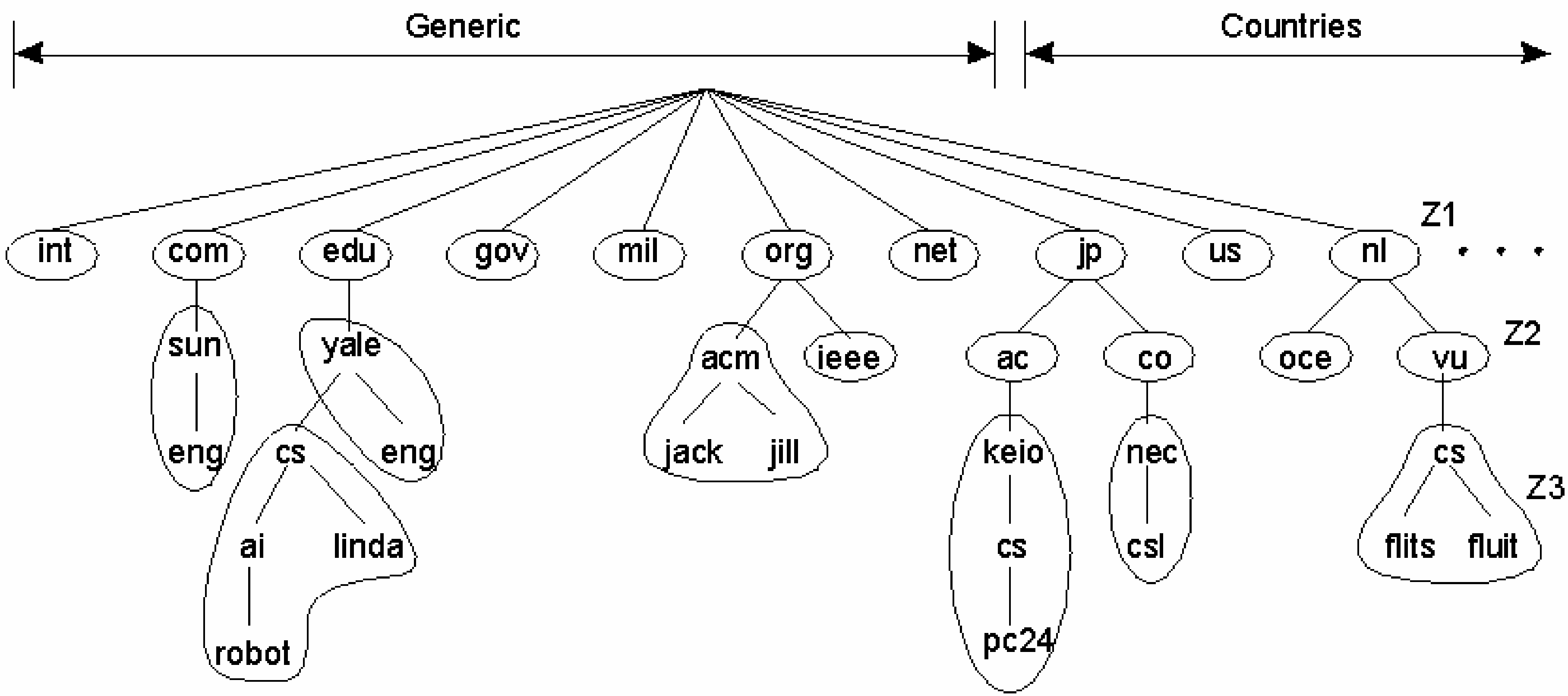
Examples of scalability limitations

Centralized services	A single server for all users
Centralized data	A single on-line telephone book
Centralized algorithms	Doing routing based on complete information

Challenges: **Scalability**

- Techniques to fight scalability problems:
  - Distribution of responsibilities (computation, storage, etc.)
  - Hide communication latencies
  - Apply replication techniques

Distribution of responsibilities

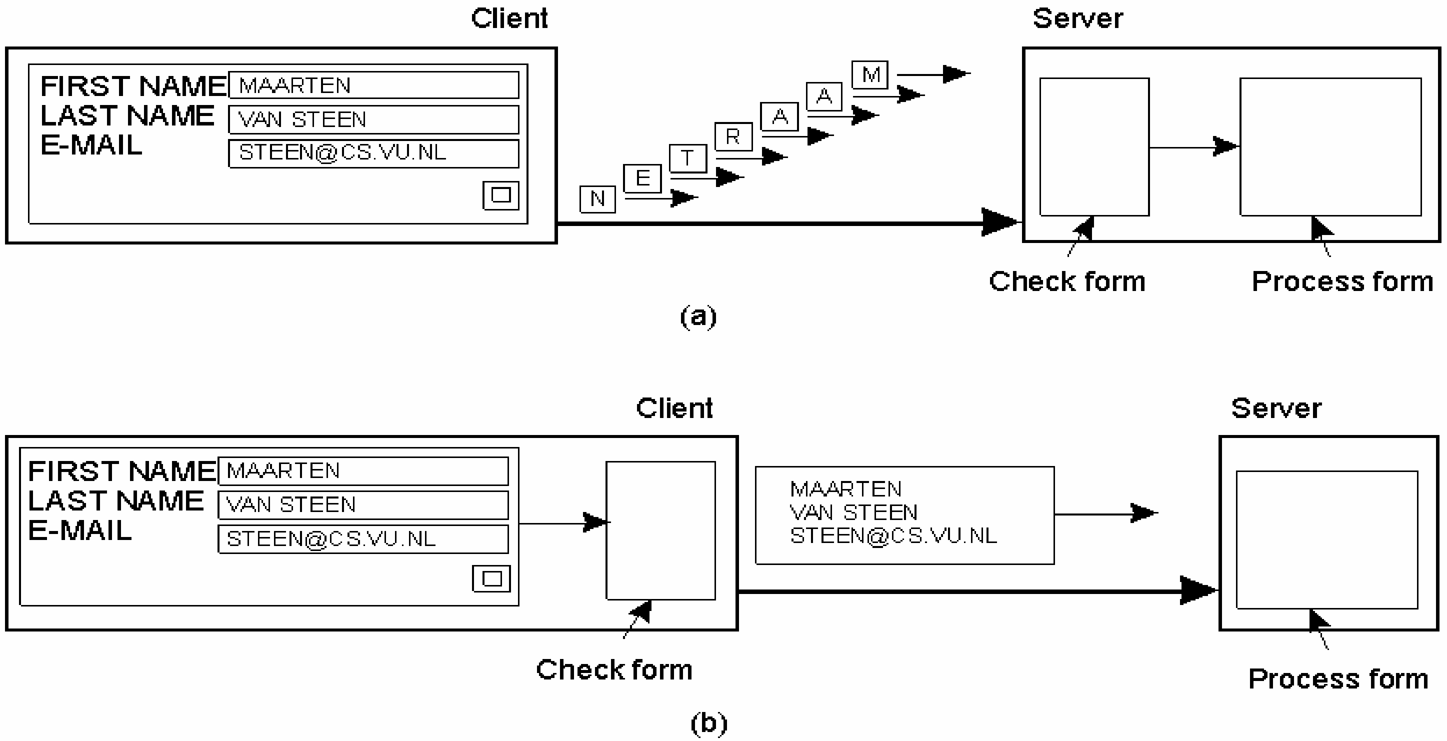


An example of dividing the DNS name space into zones.

Hiding communication latencies

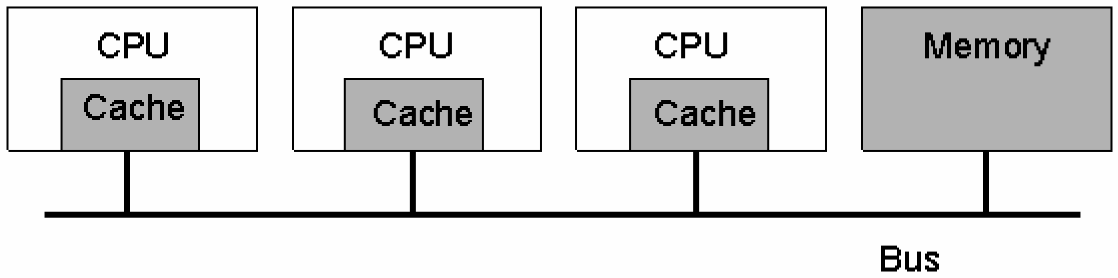
The difference between letting:

- a server or
- a client check forms as they are being filled



Replication techniques

- Caching is a typical example



Challenges in Distributed Systems

- Transparency
  - Single view of the system
  - Hide numerous details
- Heterogeneity
  - Networks
  - Computers (HW)
  - Operating systems
  - Programming languages
  - Developers
- Failure Handling
  - Detecting
  - Masking
  - Tolerating
  - Recovery
  - Redundancy
- Openness
  - Extensibility
  - Publication of interfaces
- Scalability
  - Controlling the cost of resources
  - Controlling the performance
  - Preventing resources from running out (IP addr)
  - Avoiding performance bottlenecks
- Security
  - Secrecy
  - Authentication
  - Authorization
  - Non-repudiation
  - Mobile code
  - Denial of service

Challenges: **Security**

- When sensitive information (passwords, credit cards, medical records) pass through a number of systems, securing their access is a rather complex task.
- Secure systems against attacks (viruses, worms, Denial of Service)
- Security is the "weakest link" in distributed systems

Hardware Architectures

Dynamic and Distributed
Information Systems

Software and Hardware service layers

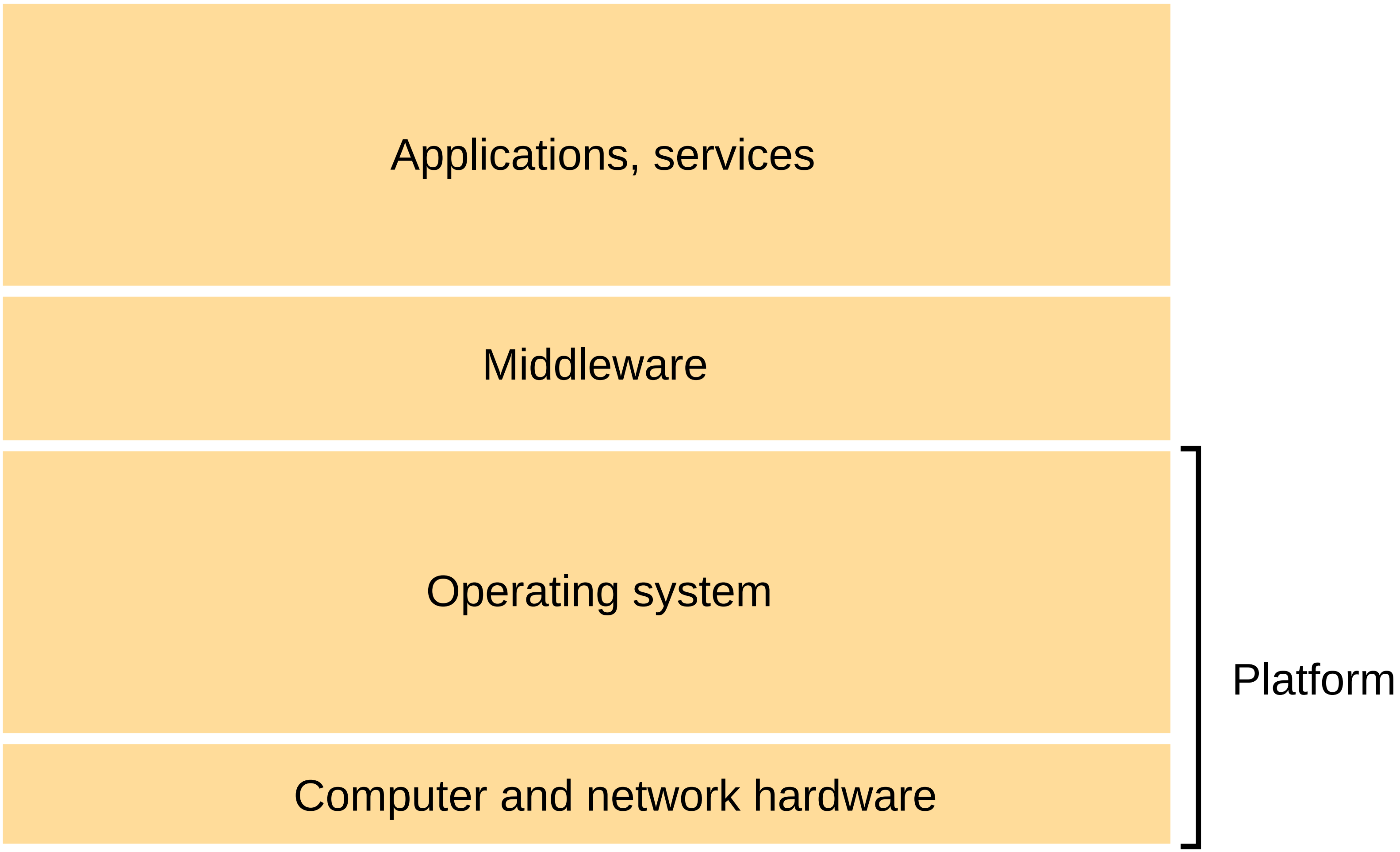
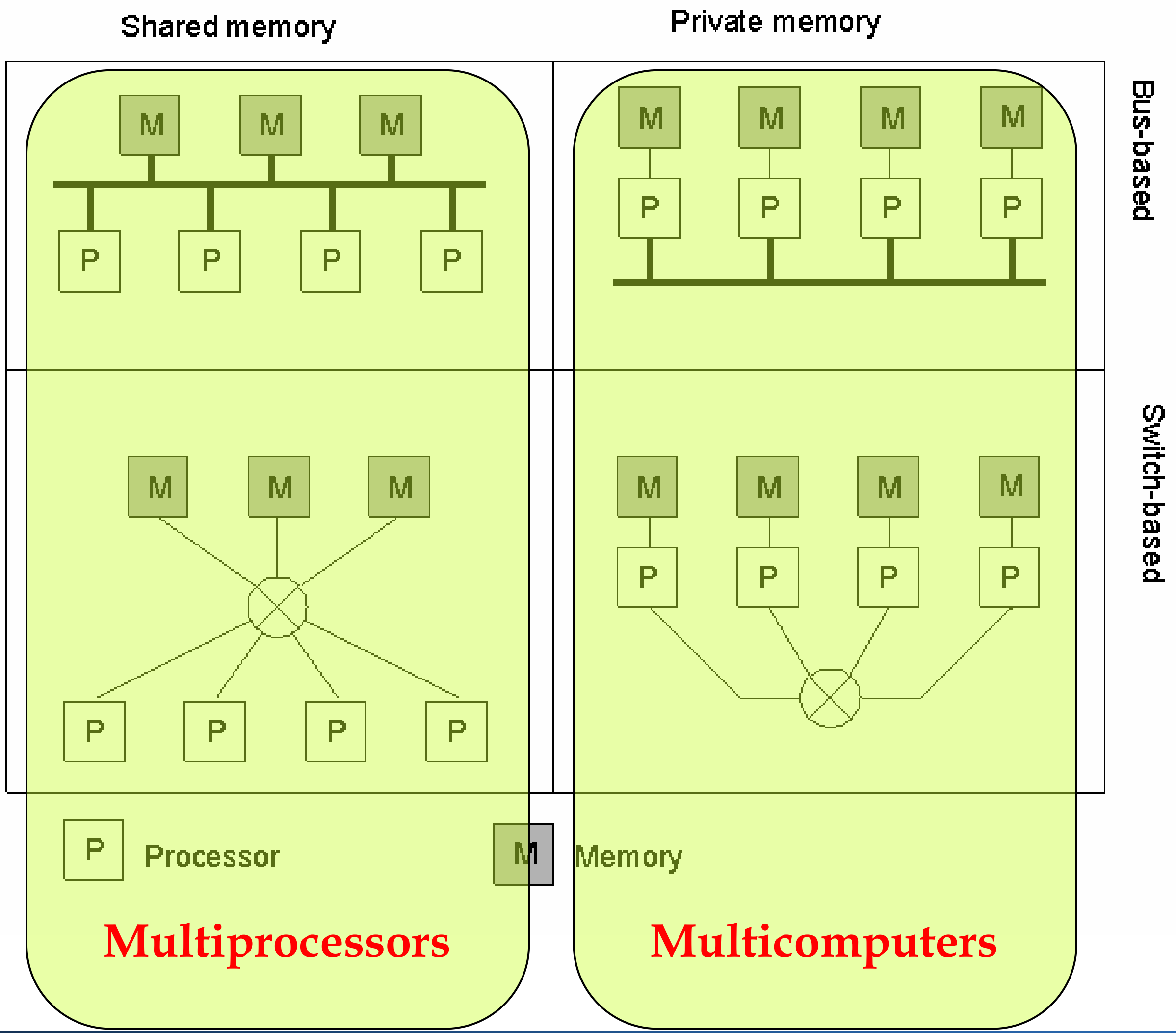


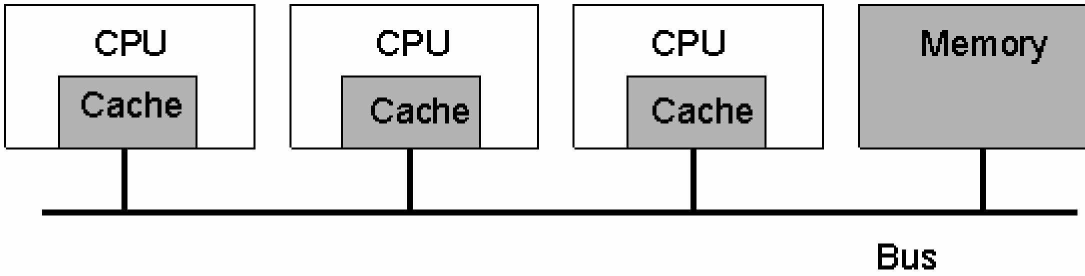
Figure 2.1

Hardware Concepts

Different basic organizations and memories in distributed computer systems



Multiprocessors, bus-based

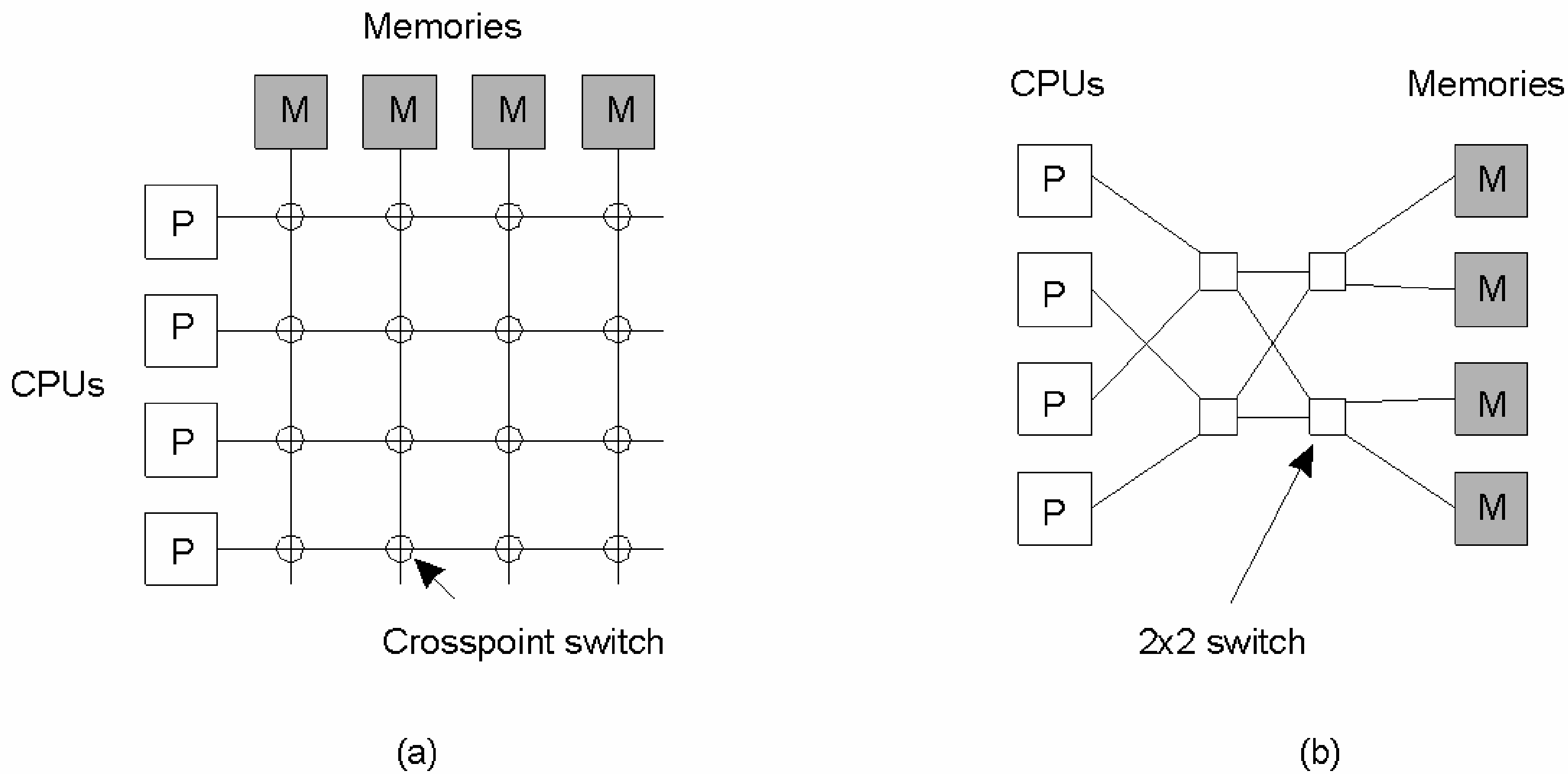


- All processors share the same bus to access the memory

- **PLUS:** simpler / cheaper construction
- **MINUS:** Not scalable: with more than a few processors, the bus is saturated

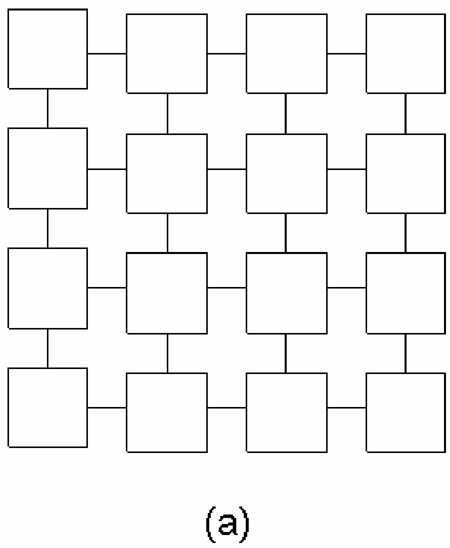
- Cache memories are introduced to prevent bus saturation
  - Consistency problems
  - Need to invalidate other caches after every write

Multiprocessors, switch-based

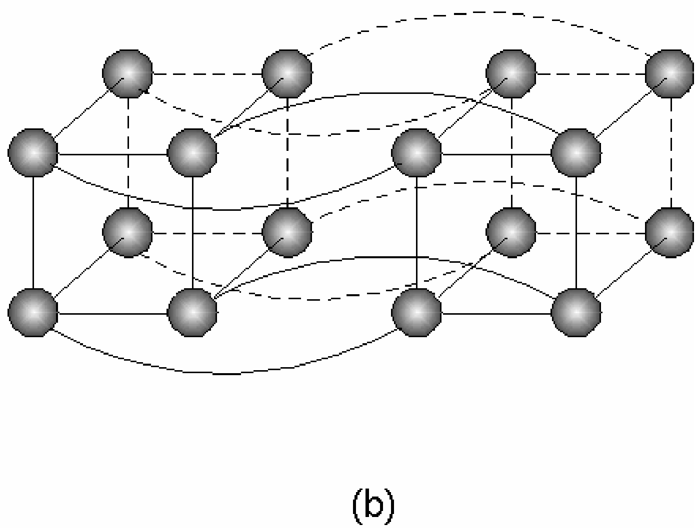


- Concurrent memory access by multiple processors
  - although not all combinations
- **PLUS:** Increased concurrency → gives speed
- **MINUS:** Delay due to many switches, expensive linkage & fast crosspoint switches

Multicomputers, switch-based



Grid topology



Hypercube topology

Software Architectures

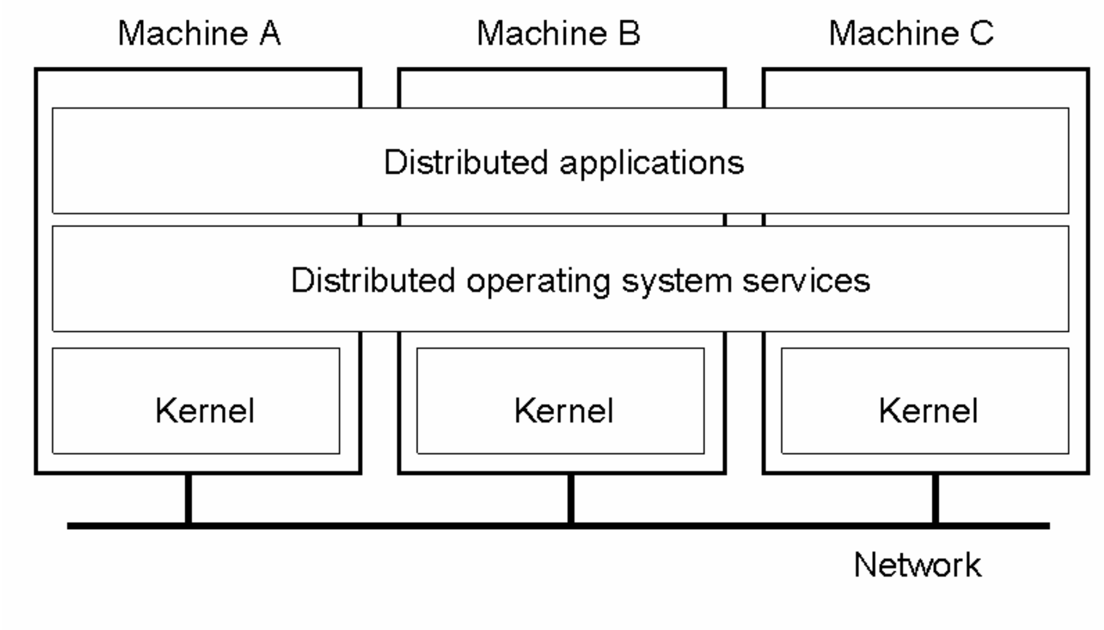
Dynamic and Distributed
Information Systems

Software Concepts

- Distributed Operating Systems (DOS)
- Network Operating Systems (NOS)
- Middleware

System	*Description*	*Main Goal*
Distributed OS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
Network OS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

Distributed OS



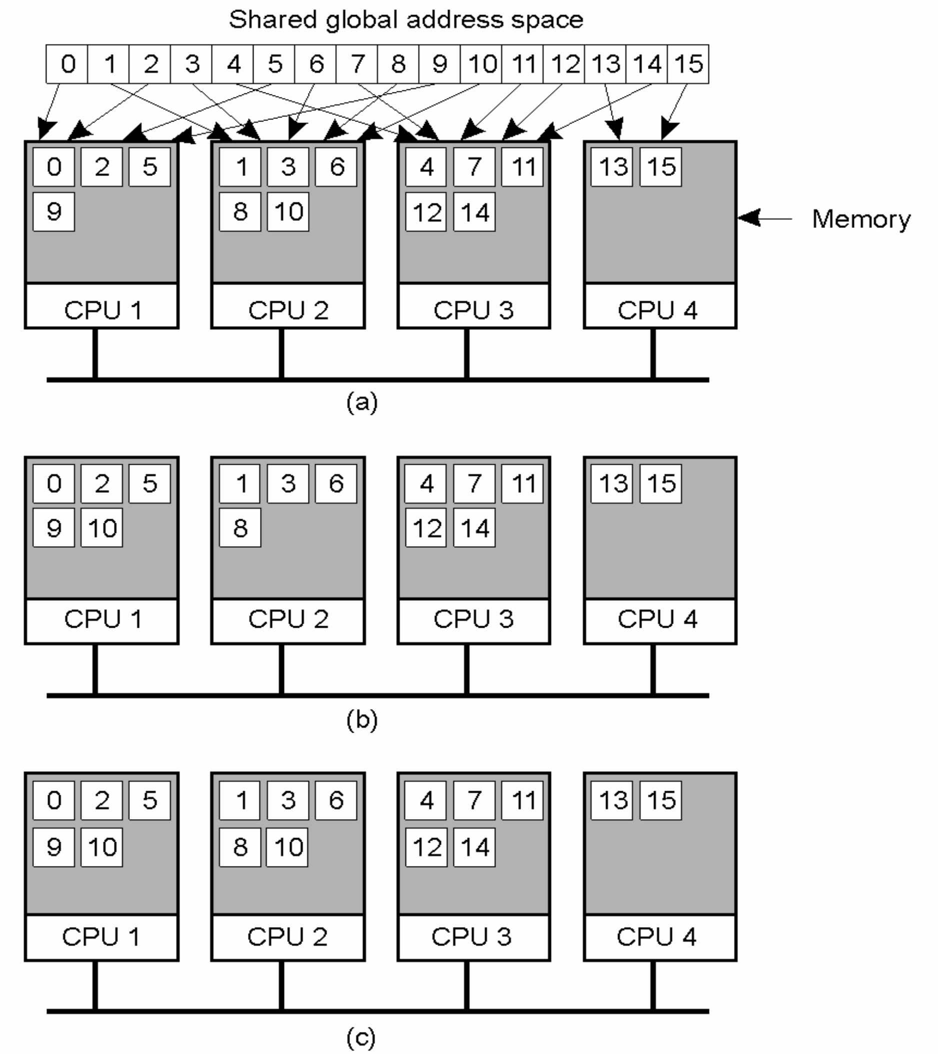
- Tightly coupled, detailed control
- Takes care of:
  - Transparent task allocation to a processor
  - Transparent memory access (Distributed Shared Memory)
  - Transparent storage, etc.
- Provides complete transparency and single view of the system
- Requires multiprocessors or *homogeneous* multicomputers

Example: Distributed Shared Memory

Pages of address space distributed among four machines

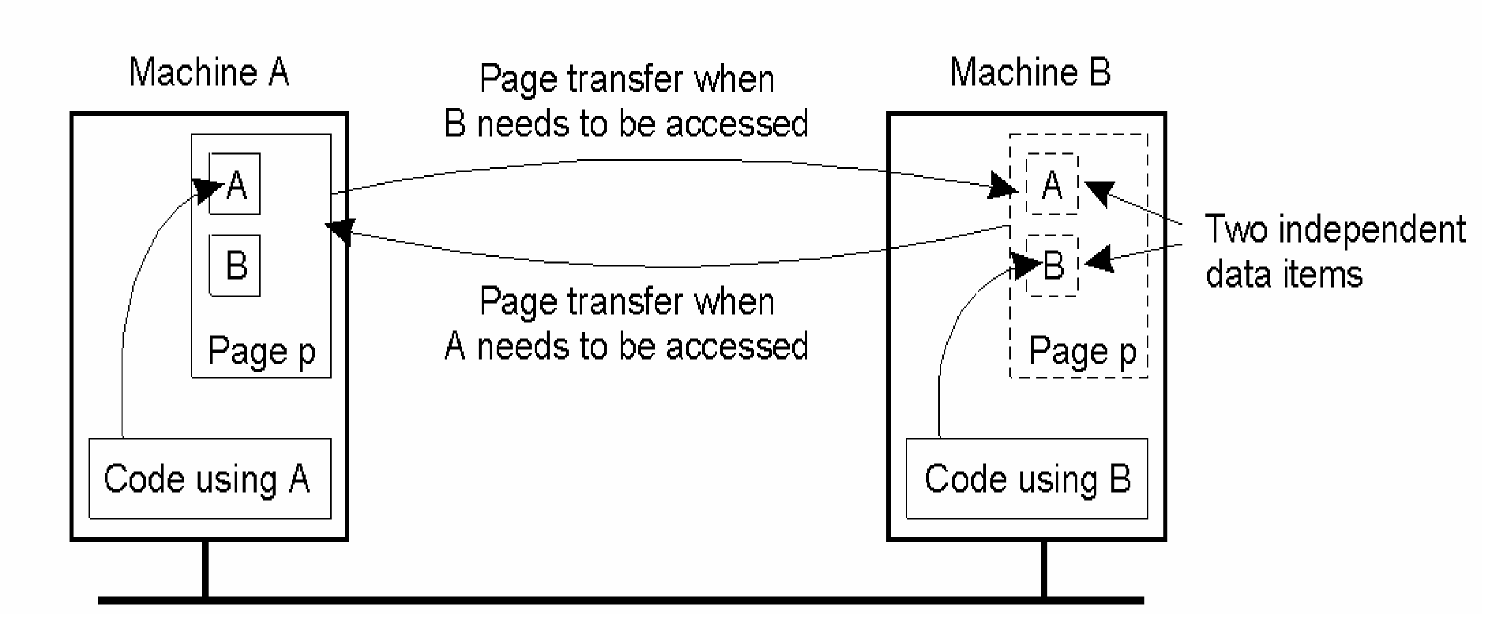
Situation after CPU 1 references page 10

Situation if page 10 is read only and replication is used



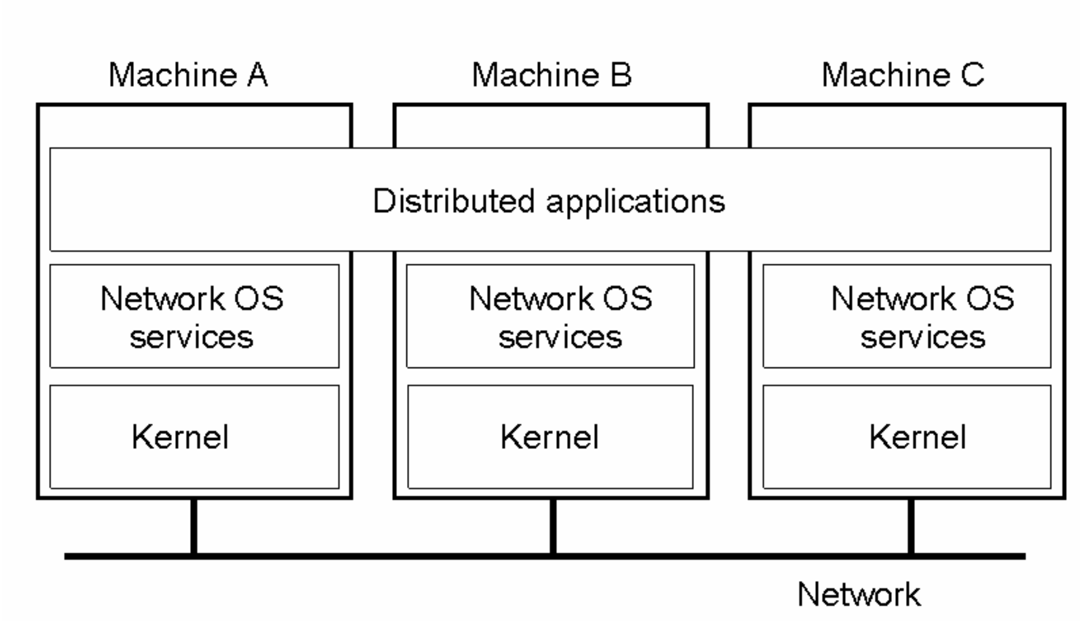
Distributed Shared Memory

- Increased complexity



- E.g., false sharing of a page between two independent processes.
  → There is a tradeoff between page size and # of transfers

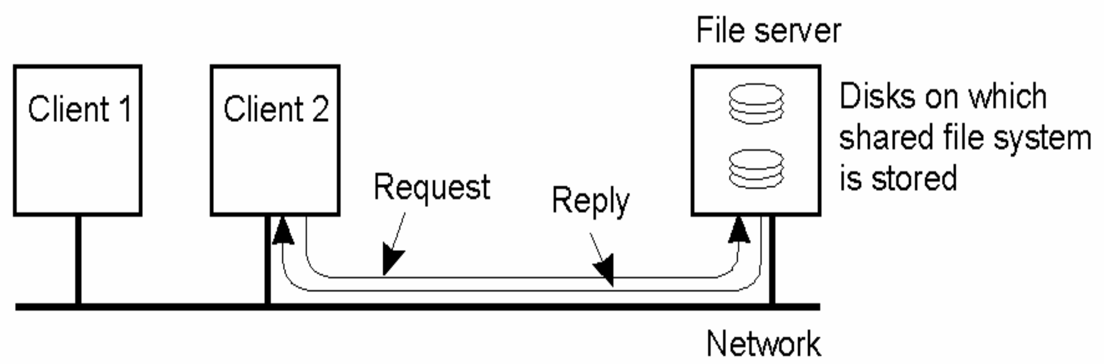
Network OS



- Loosely coupled, less control
- Provides services, such as:
  - rlogin
  - ftp, scp
  - NFS, etc.
- Not transparent, no single view of the system
- Very flexible w.r.t. heterogeneity and participation

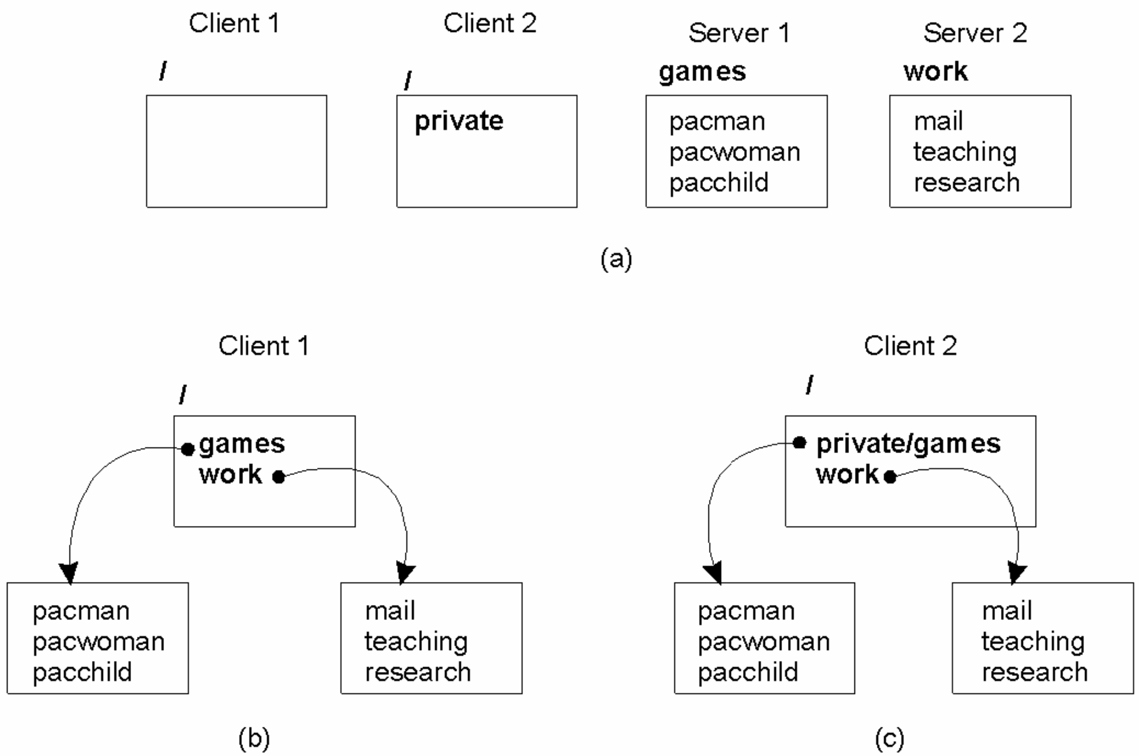
Network OS

Two clients and a server in a network operating system.

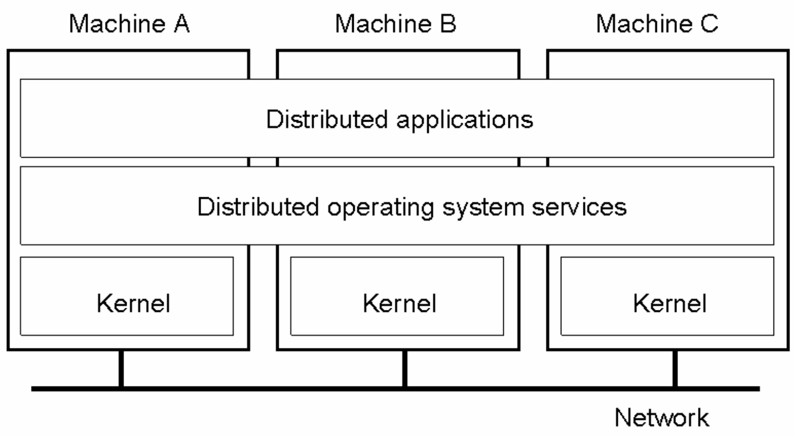


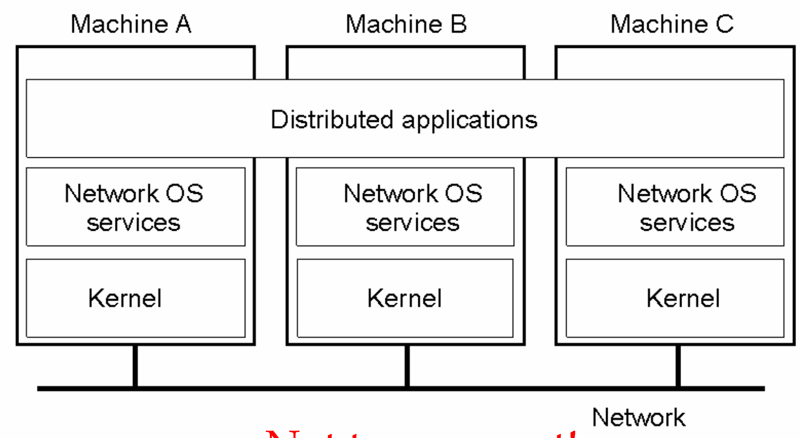
Network OS

Different clients may mount the servers in different places.
No transparency!!



Distributed OS --*vs.*-- Network OS



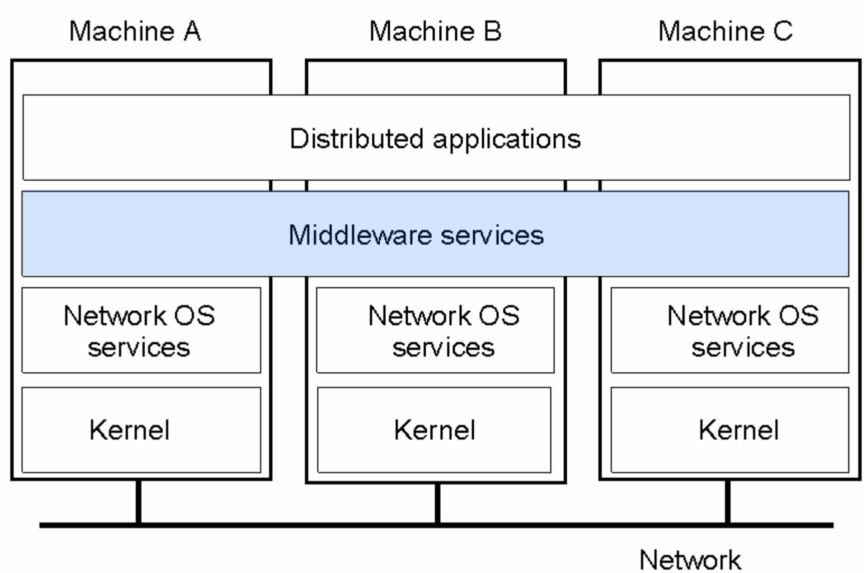


Transparent

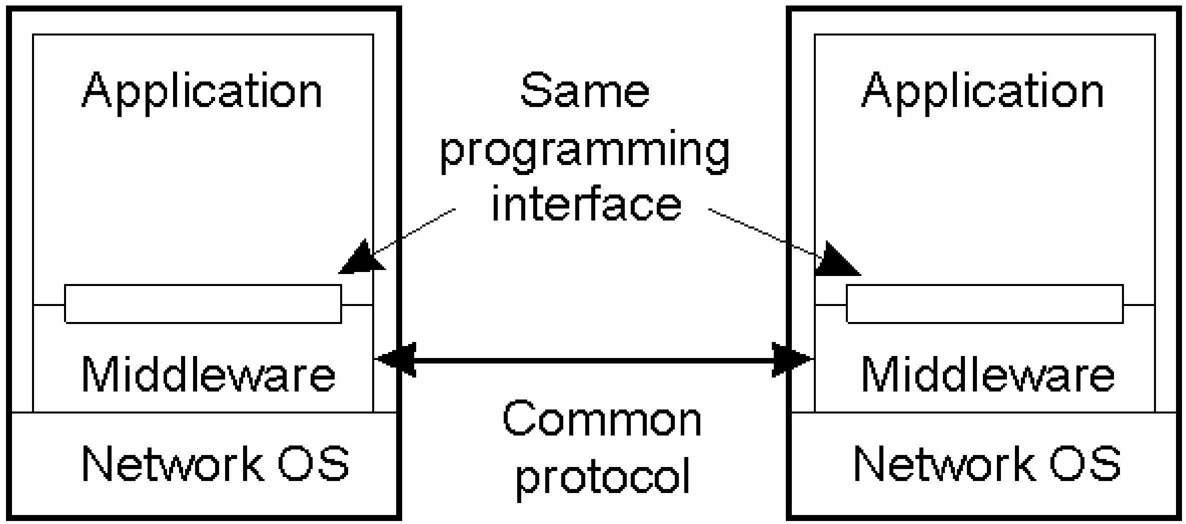
Not autonomous computers!

Not transparent!

Autonomous computers



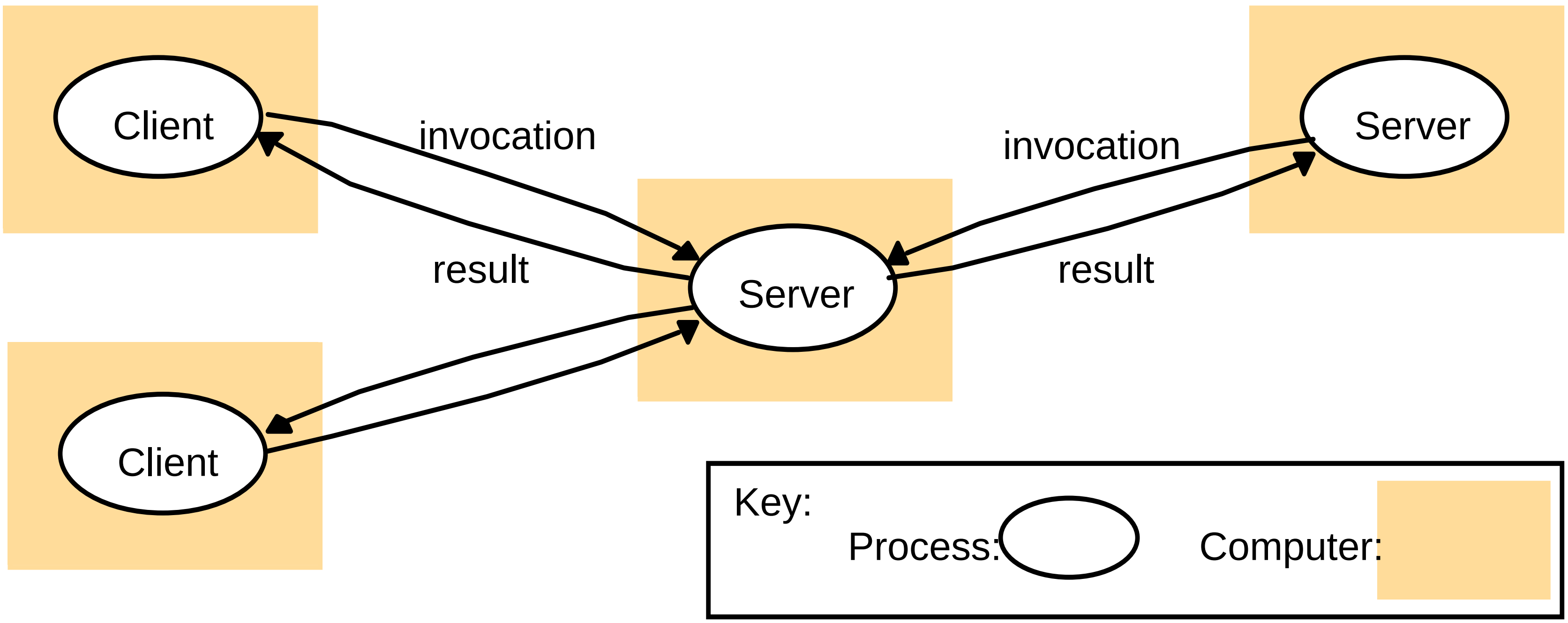
Middleware and Openness



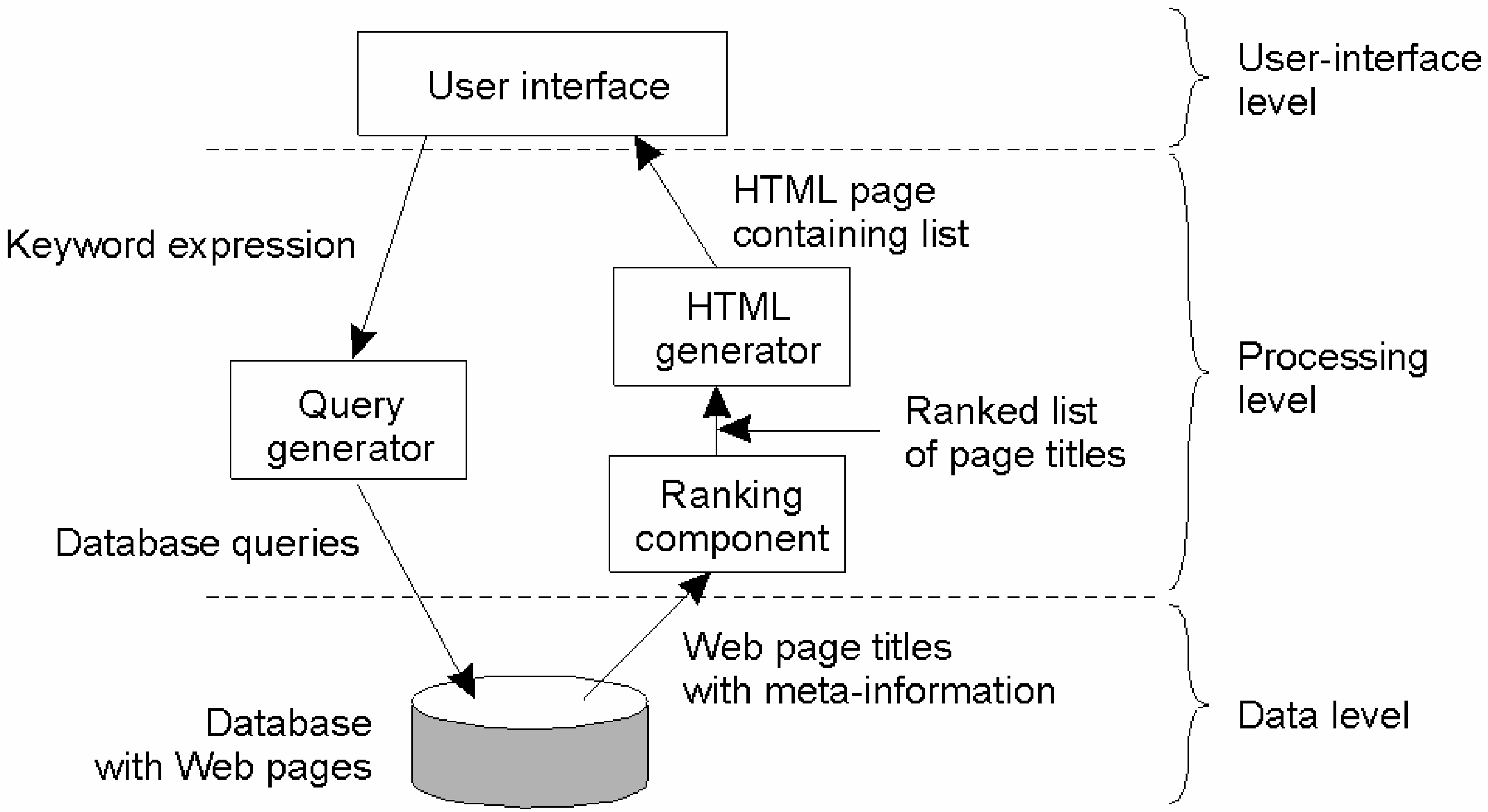
In an open middleware-based distributed system, the protocols used by each middleware layer should be the same, as well as the interfaces they offer to applications.

Types of Network Interaction

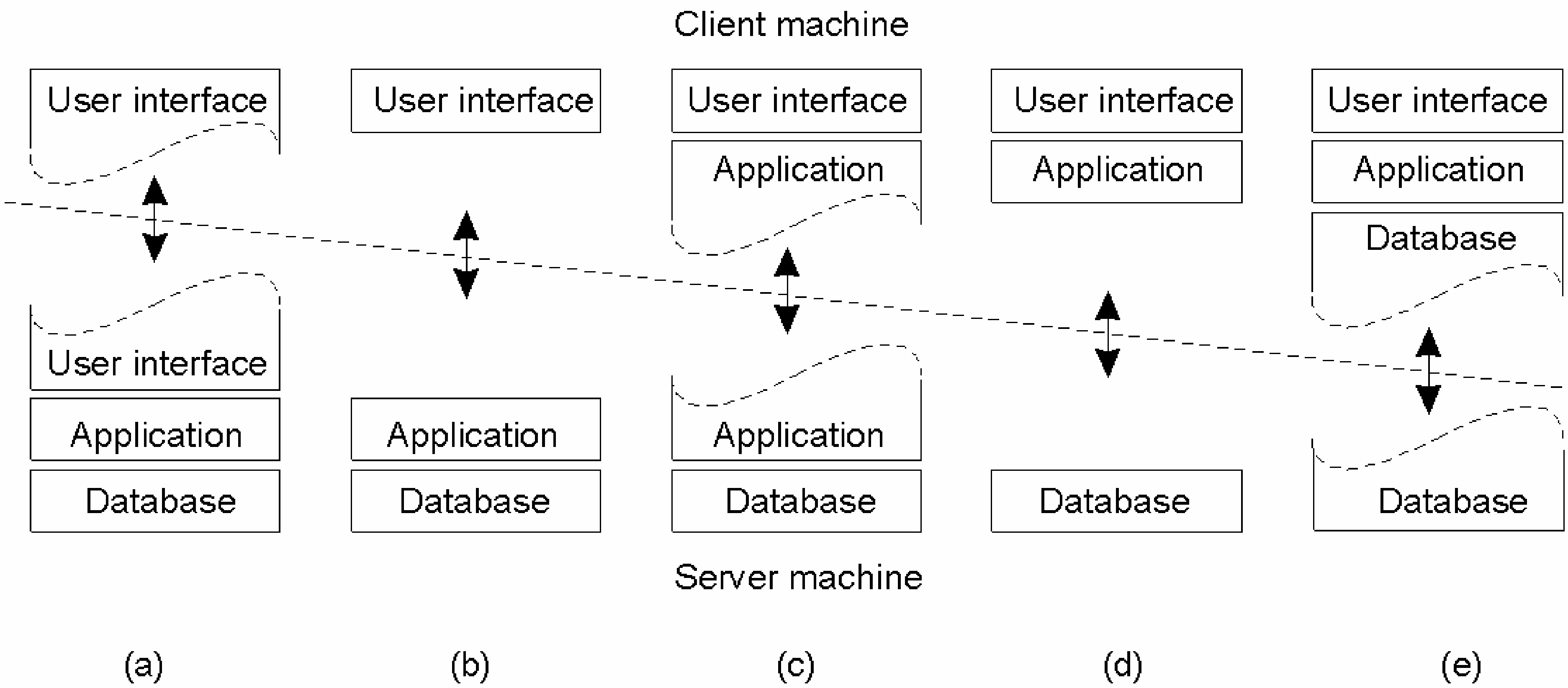
Client / Server



3-tier Network Application



Multi-tiered Architectures



Cluster of servers

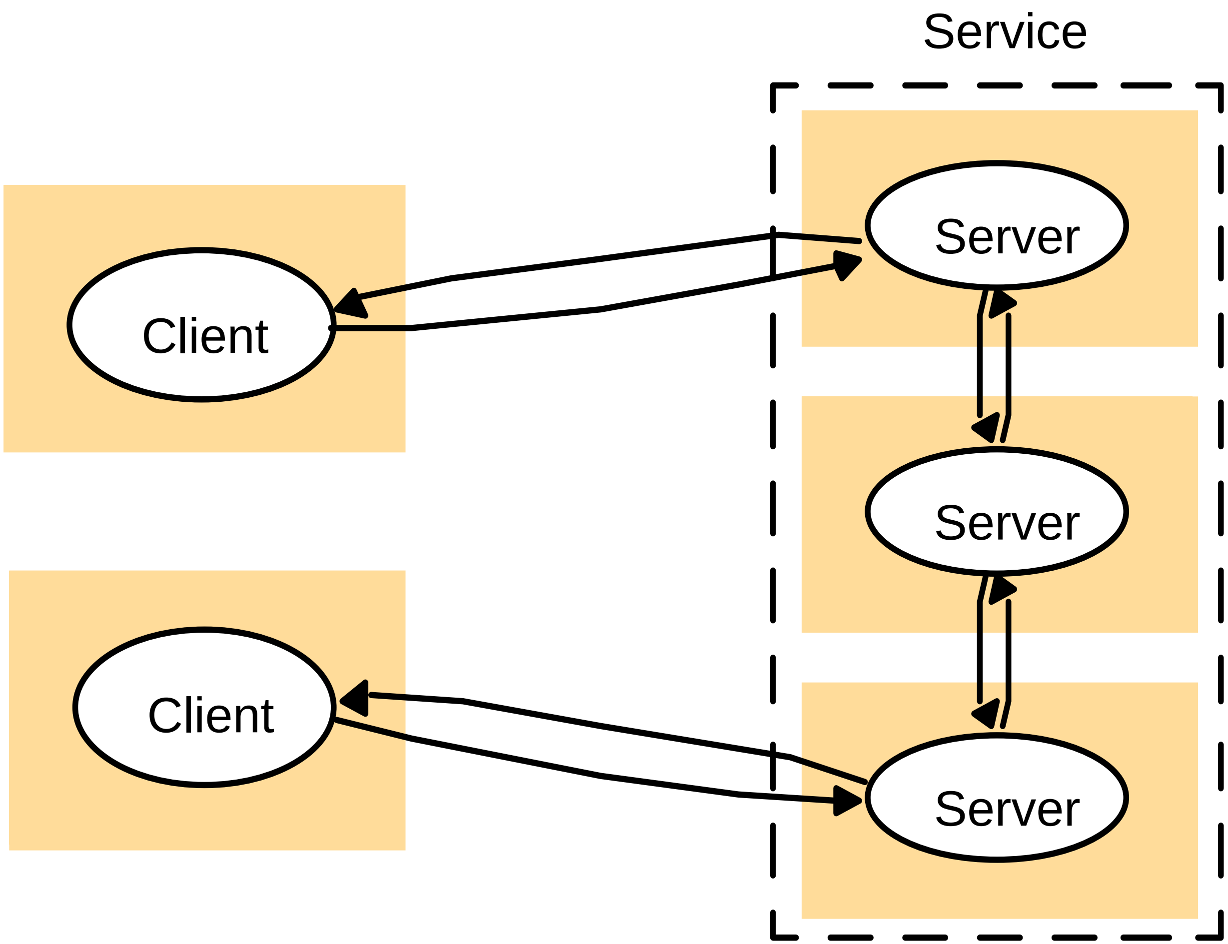


Figure 2.4

Web proxy server

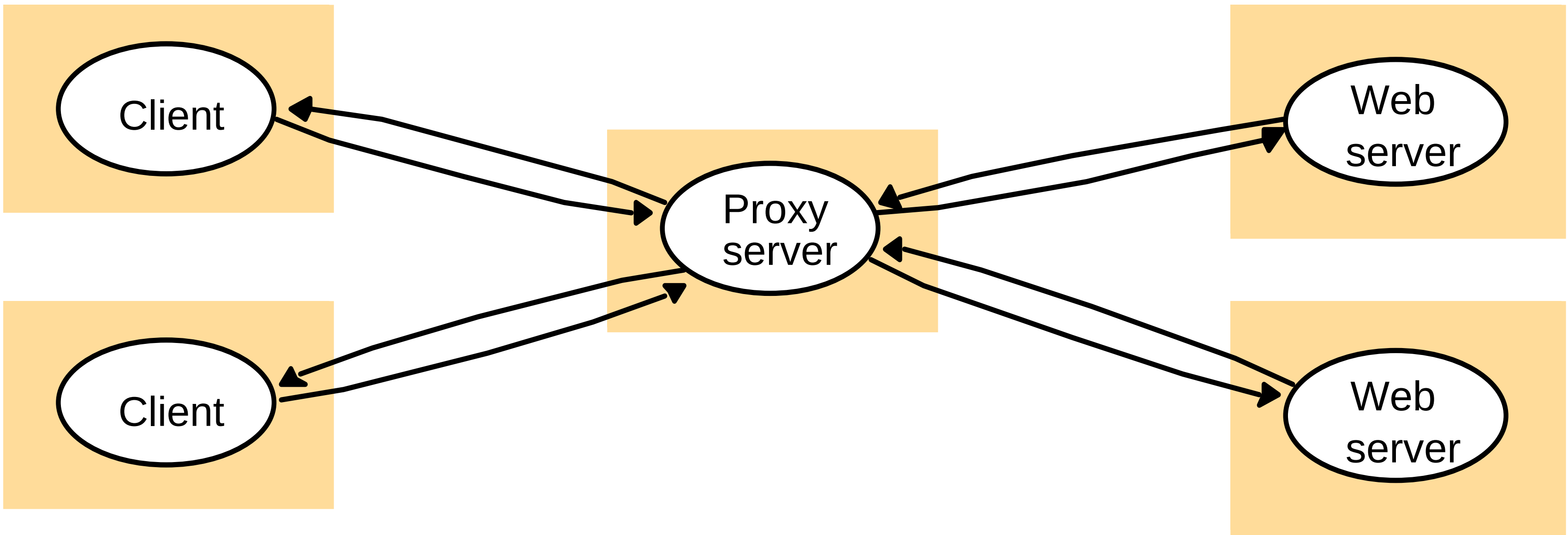
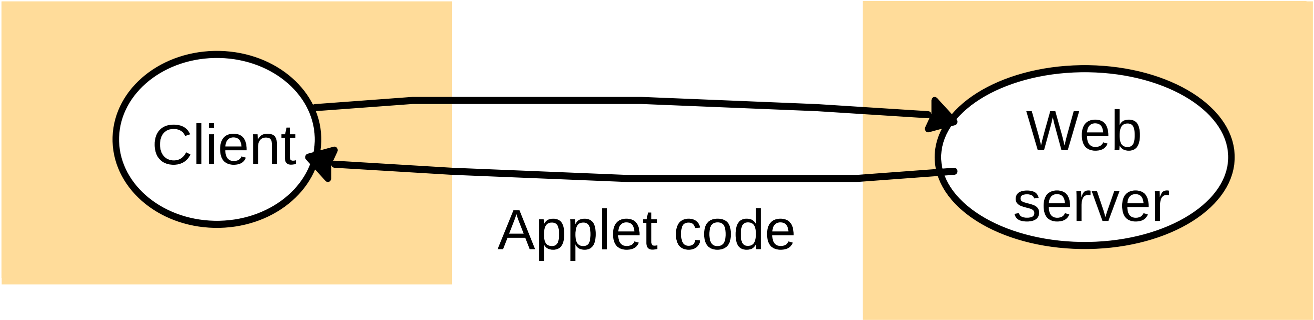


Figure 2.5

Code Mobility

a) client request results in the downloading of applet code



b) client interacts with the applet

Figure 2.6

Thin clients

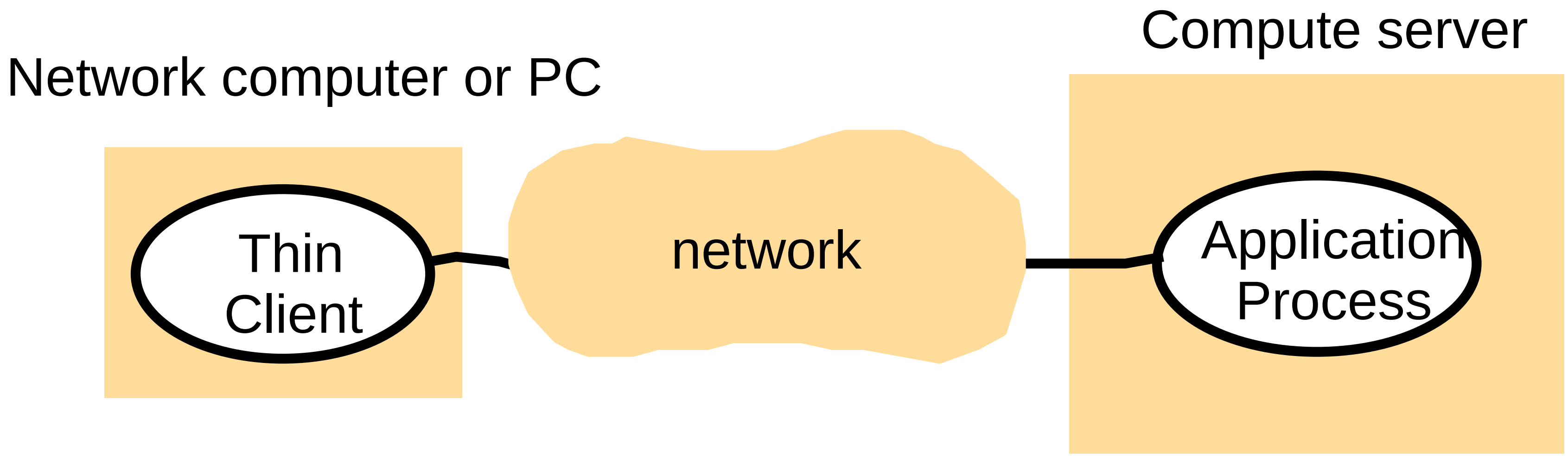


Figure 2.7

Course Outline

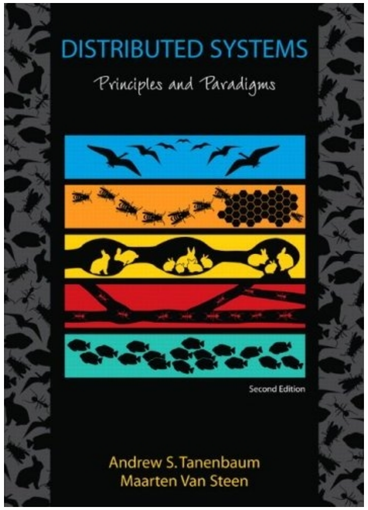
Dynamic and Distributed Information Systems

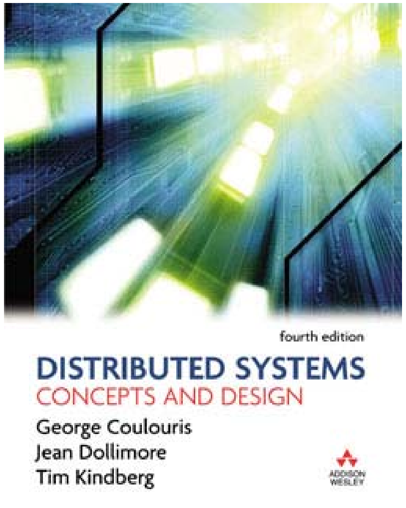
Syllabus

- Foundations
  - challenges in distributed computing
  - system models for distributed computing
  - inter-process communication
- Remote invocation and distributed objects
  - remote procedure calls (classical, marshalling)
  - web services (SOAP, WSDL, and UDDI)
  - remote object systems (RMI and CORBA)
- Coordination
  - decoupling
    - Temporal vs. Referential decoupling
  - name-space related messaging
  - tuple spaces
  - Other Coordination problems
- Other Middleware
  - distributed file systems
  - distributed shared memory
  - grids
- Peer to peer systems
  - File-sharing
  - Structured Overlays (Distributed Hash Tables)
  - Self-Organization

Course Structure – Logistics

- Class time:
  - Monday 4-6, BIN 2.A.10
- Class Style:
  - Lecture with in class assignments
- Teaching Assistant: Jiwen Li
- Grading
  - Assignments / Participation: 30%
  - Final Exam (written): 70%
- Books/Readings
  - **Distributed Systems: Principles and Paradigms (2nd edition)** Andrew S. Tanenbaum and Maarten van Steen
  - **Distributed Systems: Concepts and Design (4th edition)** George Coulouris, Jean Dollimore and Tim Kindberg.
- Communication:
  - Email: spyros (at) inf.ethz.ch
  - Please prepend subject with prefix "DS:" (including the colon)
  - Please send from your university account





Questions? Comments?