HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

# Chapter 1: Distributed Systems: What is a distributed system?

Fall 2009

Jussi Kangasharju

# Course Goals and Content

- Distributed systems and their:
  - Basic concepts
  - Main issues, problems, and solutions
  - Structured and functionality
- Content:
  - Distributed systems (Tanenbaum, Ch. 1)
    - Architectures, goal, challenges
    - Where our solutions are applicable
  - Synchronization: Time, coordination, decision making (Ch. 5)
  - Replicas and consistency (Ch. 6)
  - Fault tolerance (Ch. 7)
- Chapters refer to Tanenbaum book

## Course Material

- Tanenbaum, van Steen: Distributed Systems, Principles and Paradigms; Prentice Hall 2007
  - 2002 edition also ok
- Coulouris, Dollimore, Kindberg: Distributed Systems, Concepts and Design; Addison-Wesley 2005
- Lecture slides on course website
  - NOT sufficient by themselves
  - Help to see what parts in book are most relevant

## Course Exams

- Normal way (recommended)
  - Exercises, home exercises, course exam
- Grading:
  - Exam 48 points
  - Exercises 12 points (~ 20 exercises, scaled to 0—12)
  - Home exercises 6 points (3 exercises)
  - Grading based on 60 point maximum
  - Need 30 points to pass with minimum 16 points in exam
  - 50 points will give a 5
- Possible to take as separate exam

# Exercises

- Weekly exercises:
  - Smaller assignments

- Home exercises
  - 1 study diary, 2 design exercises
  - Due dates will be announced later
  - Study diary individual work
  - Design exercises can be done in groups of up to 3

# People

- Jussi Kangasharju
  - Lectures: Tue 10-12 and Thu 10-12 in D122
  - Office hour: Thu 12-13 or ask for appointment by email
- Mikko Pervilä
  - Exercise groups:
    - 1. Mikko Pervilä Tue 14-16 in B119
    - 2. Mikko Pervilä Tue 16-18 in B119 (in English)
  - Home exercises
  - Office hour: During exercises or ask appointment by email

# Questions?

## Chapter Outline

- Defining distributed system
- Examples of distributed systems
- Why distribution?

- Goals and challenges of distributed systems

- Where is the borderline between a computer and a distributed system?
- Examples of distributed architectures

# Definition of a Distributed System

A distributed system is

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

... or ...
as a single system.

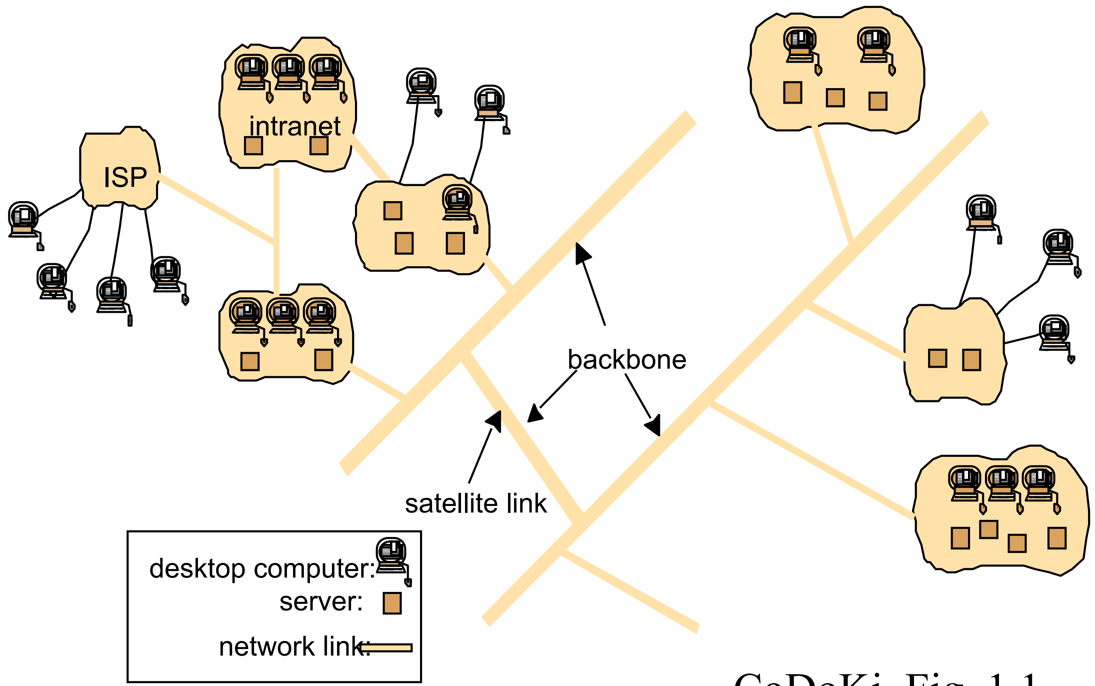
# Examples of Distributed Systems

The Internet: net of nets

global access to “everybody”

(data, service, other actor; open ended)

- enormous size (open ended)
- no single authority
- communication types
  - interrogation, announcement, stream
  - data, audio, video




CoDoKi, Fig. 1.1

**Figure 1.1 A typical portion of the Internet**

# Examples of Distributed Systems

Intranets ( CoDoKi, Fig. 1.2)

- a single authority
- protected access
  - a firewall
  - total isolation
- may be worldwide
- typical services:
  - infrastructure services: file service, name service
  - application services

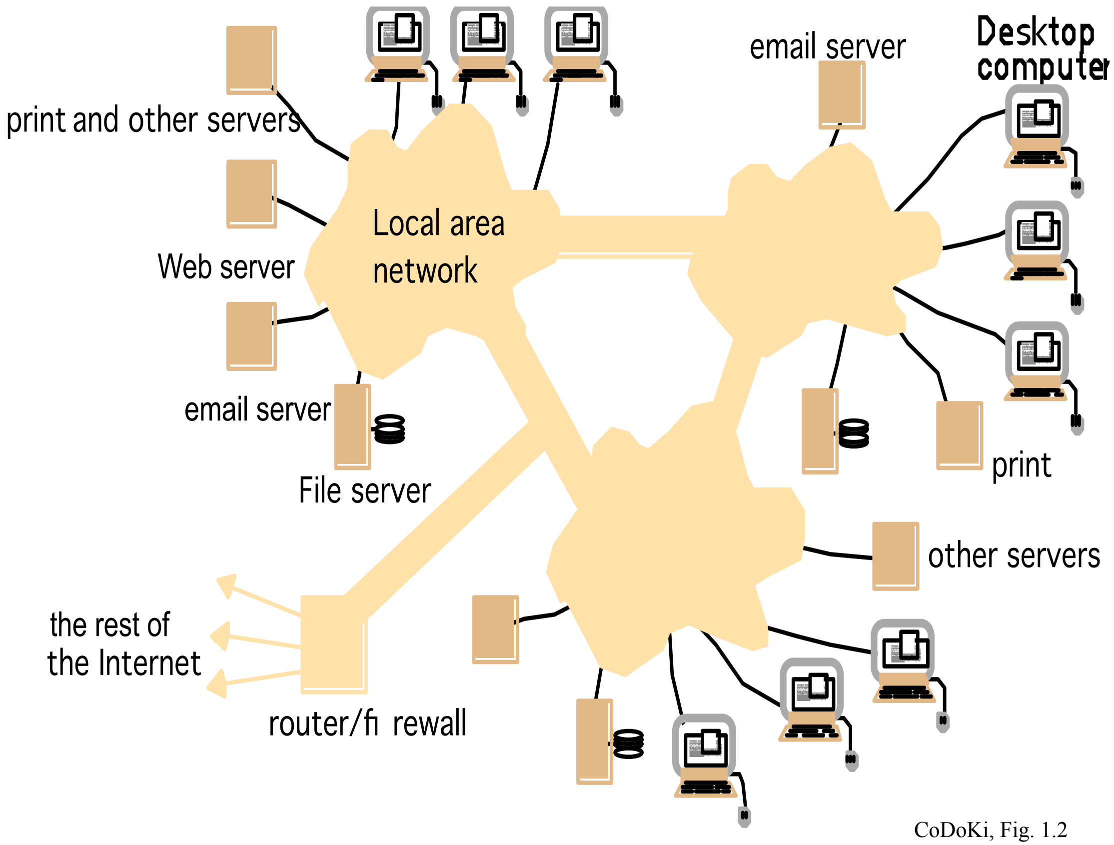



**Figure 1.2 A typical intranet**

# Examples of Distributed Systems

Mobile and ubiquitous computing ( CoDoKi Fig 1.3 )

- Portable devices
  - laptops
  - handheld devices
  - wearable devices
  - devices embedded in appliances
- Mobile computing
- Location-aware computing
- Ubiquitous computing, pervasive computing

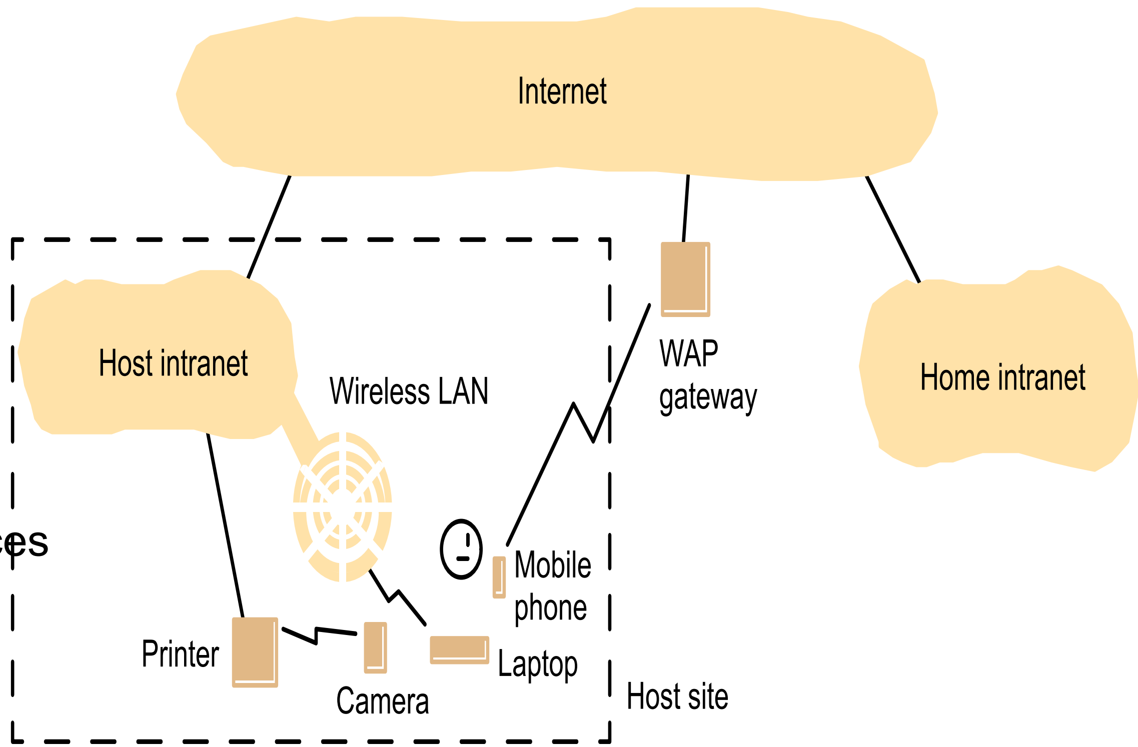
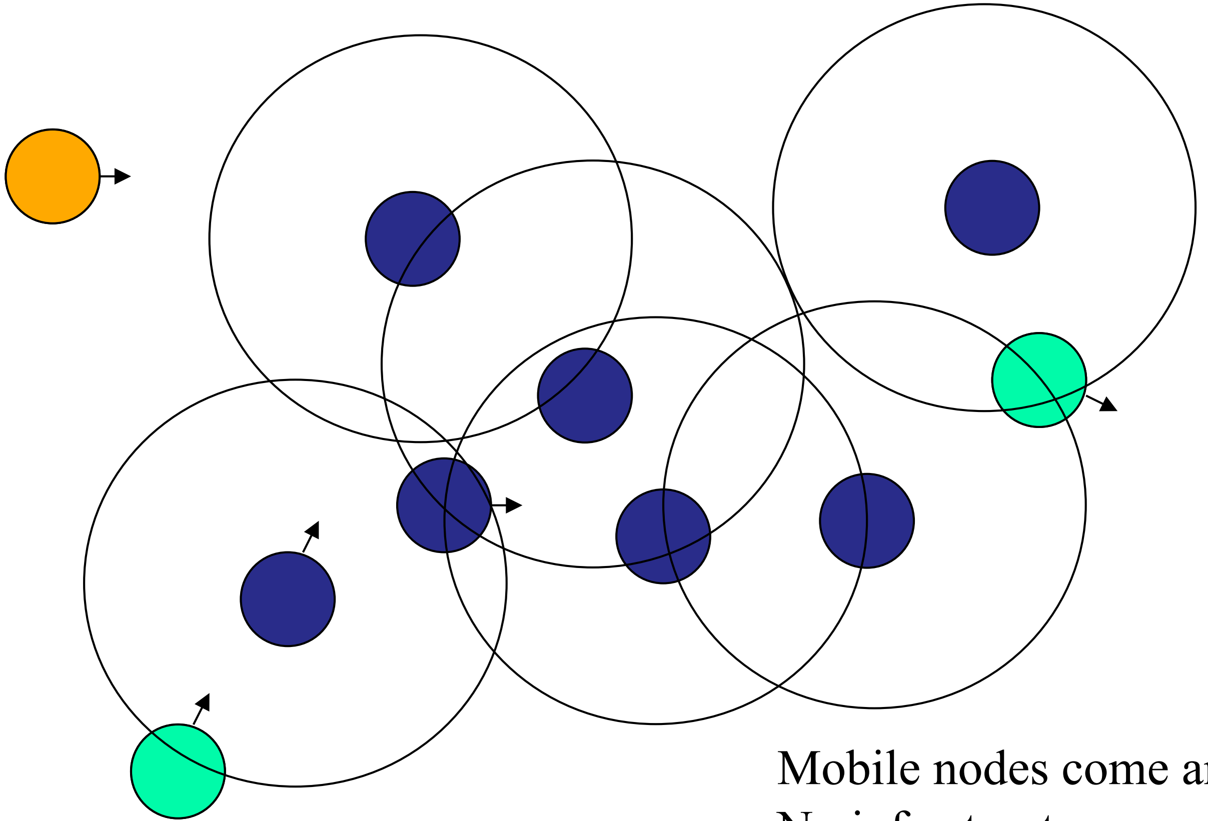



Figure 1.3 Portable and handheld devices in a distributed system

CoDoKi, Fig. 1.3

# Mobile Ad Hoc -Networks



Mobile nodes come and go

No infrastructure

- wireless data communication
- multihop networking
- long, nondeterministic dc delays

Problems, e.g.:

- reliable multicast
- group management

# Resource Sharing and the Web

- Hardware resources (reduce costs)
- Data resources (shared usage of information)
- Service resources
  - search engines
  - computer-supported cooperative working
- Service vs. server (node or ***process*** )

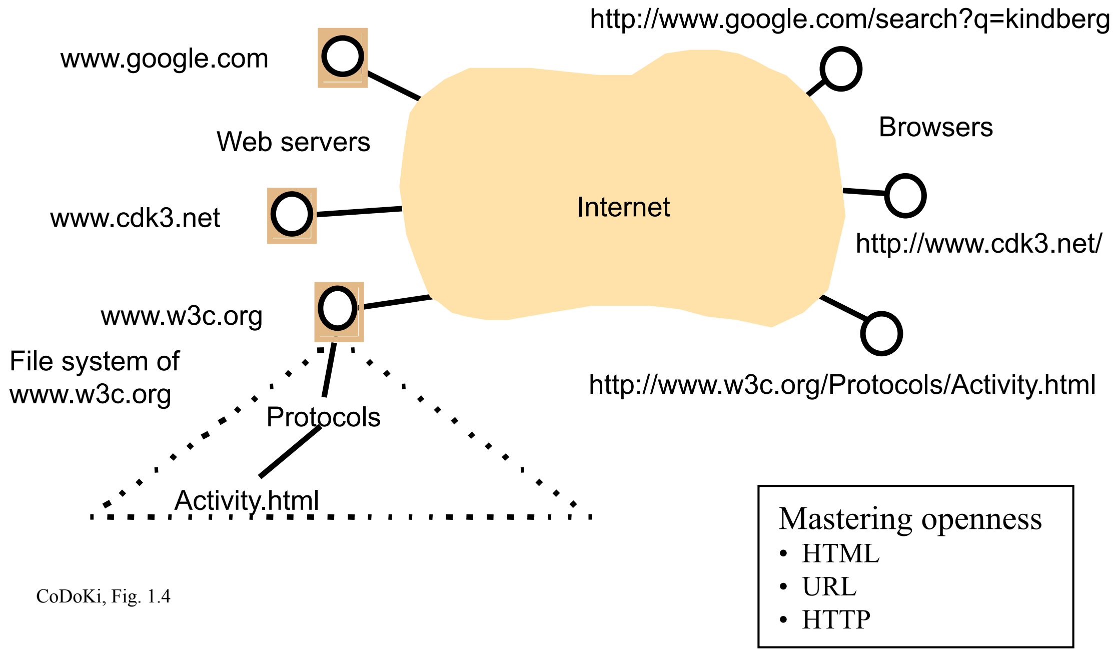
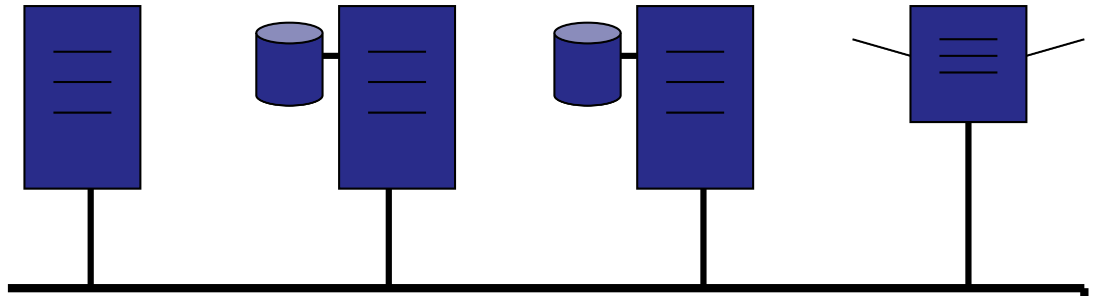
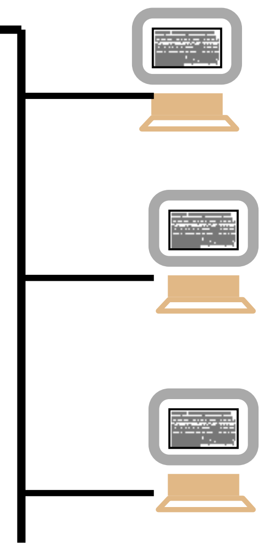



**Figure 1.4 Web servers and web browsers**

# Examples of Distributed Systems



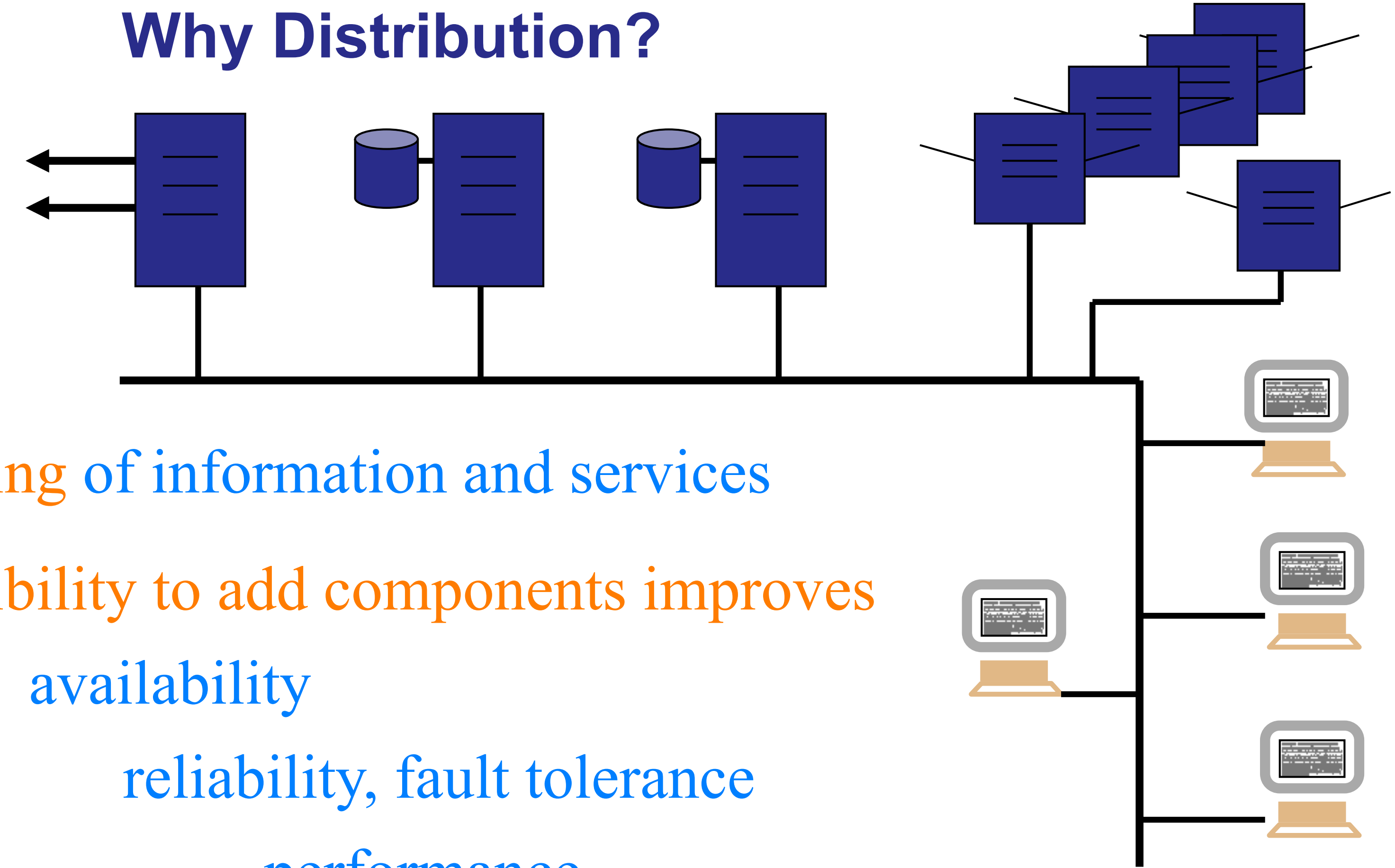


## Distributed application

- one single “system”
- one or several autonomous subsystems
- a collection of processors => parallel processing
  => increased performance, reliability, fault tolerance
- partitioned or replicated data
  => increased performance, reliability, fault tolerance

Dependable systems, grid systems, enterprise systems

# Why Distribution?



Sharing of information and services

Possibility to add components improves

availability

reliability, fault tolerance

performance

scalability

Facts of life: history, geography, organization

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

# Goals and challenges for distributed systems

# Goals

- Making resources accessible
- Distribution transparency
- Openness
- Scalability
- Security
- System design requirements

# Challenges for Making Resources Accessible

- Naming
- Access control
- Security
- Availability
- Performance
- Mutual exclusion of users, fairness
- Consistency in some cases

# Challenges for Transparency

- The fundamental idea: a collection of independent, autonomous actors
- Transparency:
  - Concealment of distribution
  - => user’s point of view: a single unified system

# Transparencies

| Transparency | Description |
|---|---|
| Access | Hide differences in data representation and how a resource is accessed |
| Location | Hide where a resource is located (∗) |
| Migration | Hide that a resource may move to another location (∗)<br>(the resource does not notice) |
| Relocation | Hide that a resource may be moved to another location (∗)<br>while in use (the others don’t notice) |
| Replication | Hide that a resource is replicated |
| 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 |

(∗) Notice the various meanings of ”location” : network address (several layers) ; geographical address

## Challenges for Transparencies

- replications and migration cause need for ensuring consistency and distributed decision-making
- failure modes
- concurrency
- heterogeneity

# Figure 2.10
# 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 *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 2.11
# 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. |

# Failure Handling

- More components => increased fault rate
- Increased possibilities
  - more redundancy => more possibilities for fault tolerance
  - no centralized control => no fatal failure
- Issues
  - Detecting failures
  - Masking failures
  - Recovery from failures
  - Tolerating failures
  - Redundancy
- New: partial failures

# Concurrency

- Concurrency:
  - Several simultaneous users => integrity of data
    - mutual exclusion
    - synchronization
    - ext: transaction processing in data bases
  - Replicated data: consistency of information?
  - Partitioned data: how to determine the state of the system?
  - Order of messages?

- There is no global clock!

# Consistency Maintenance

- Update ...
- Replication ...
- Cache ...
- Failure ...
- Clock ...
- User interface ....

} ... consistency

# Heterogeneity

- Heterogeneity of
  - networks
  - computer hardware
  - operating systems
  - programming languages
  - implementations of different developers
- Portability, interoperability
- Mobile code, adaptability (applets, agents)
- Middleware (CORBA etc)
- Degree of transparency? Latency? Location-based services?

# Challenges for Openness

- Openness facilitates
  - interoperability, portability, extensibility, adaptivity
- Activities addresses
  - extensions: new components
  - re-implementations (by independent providers)
- Supported by
  - public interfaces
  - standardized communication protocols

## Challenges for Scalability

Scalability:

- The system will remain effective when there is a significant increase in:
  - number of resources
  - number of users

- The architecture and the implementation must allow it
- The algorithms must be efficient under the circumstances to be expected
  - Example: the Internet

# Challenges: Scalability (cont.)

- Controlling the cost of physical resources
- Controlling performance loss
- Preventing software resources running out
- Avoiding performance bottlenecks
- Mechanisms (implement functions) & Policies (how to use the mechanisms)
- Scaling solutions
  - asyncronous communication, decreased messaging
  - caching (all sorts of hierarchical memories: data is closer to the user → no wait / assumes rather stable data!)
  - distribution i.e. partitioning of tasks or information (domains) (e.g., DNS)

# Challenges for Security

- Security: confidentiality, integrity, availability
- Vulnerable components (Fig. 2.14)
  - channels (links <–> end-to-end paths)
  - processes (clients, servers, outsiders)
- Threats
  - information leakage
  - integrity violation
  - denial of service
  - illegitimate usage

CoDoKi, Fig. 2.14

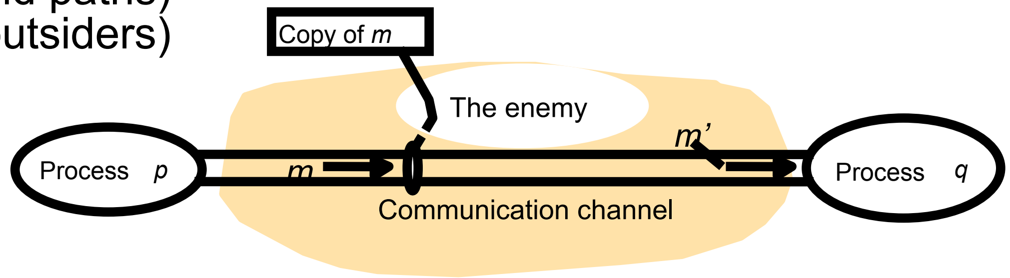



**Figure 2.14 The enemy**

Current issues:

denial-of-service attacks, security of mobile code, information flow;

open wireless ad-hoc environments

# Threats

- Threats to channels (Fig. 2.14)
  - eavesdropping (data, traffic)
  - tampering, replaying
  - masquerading
  - denial of service
- Threats to processes (Fig. 2.13)
  - server: client's identity; client: server's identity
  - unauthorized  access (insecure access model)
  - unauthorized information flow (insecure flow model)

CoDoKi, Fig. 2.14

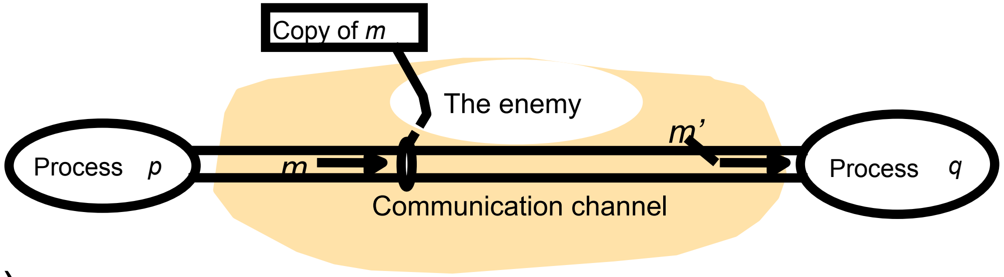
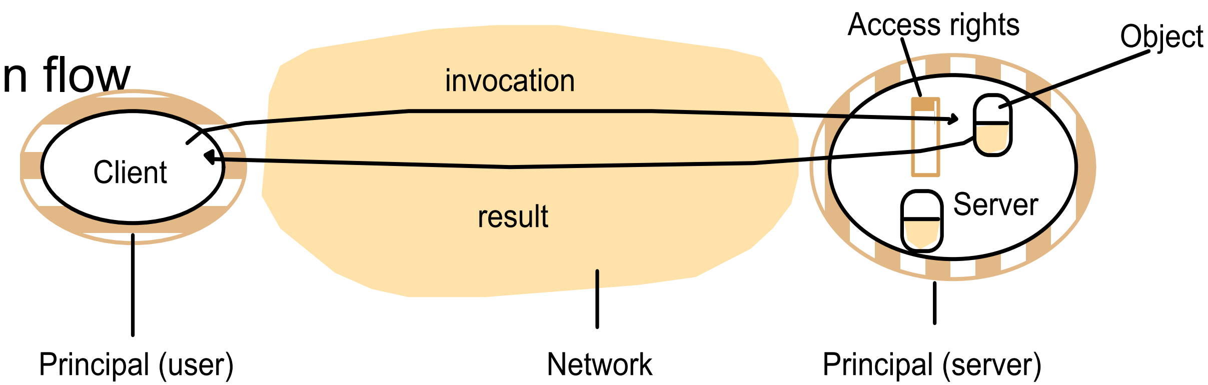



Figure 2.14 The enemy




CoDoKi, Fig. 2.13

Figure 2.13 Objects and principals

# Defeating Security Threats

- Techniques
  - cryptography
  - authentication
  - access control techniques
    - intranet: firewalls
    - services, objects: access control lists, capabilities
- Policies
  - access control models
  - lattice models
  - information flow models
- Leads to: secure channels, secure processes, controlled access, controlled flows

# Environment challenges

- A distributed system:
  - HW / SW components in different nodes
  - components communicate (using messages)
  - components coordinate actions (using messages)
- Distances between nodes vary
  - in time: from msecs to weeks
  - in space: from mm's to Mm's
  - in dependability
- Autonomous independent actors (=> even independent failures!)

No global clock

Global state information not possible

# Challenges: Design Requirements

- Performance issues
  - responsiveness
  - throughput
  - load sharing, load balancing
  - issue: algorithm vs. behavior
- Quality of service
  - correctness (in nondeterministic environments)
  - reliability, availability, fault tolerance
  - security
  - performance
  - adaptability

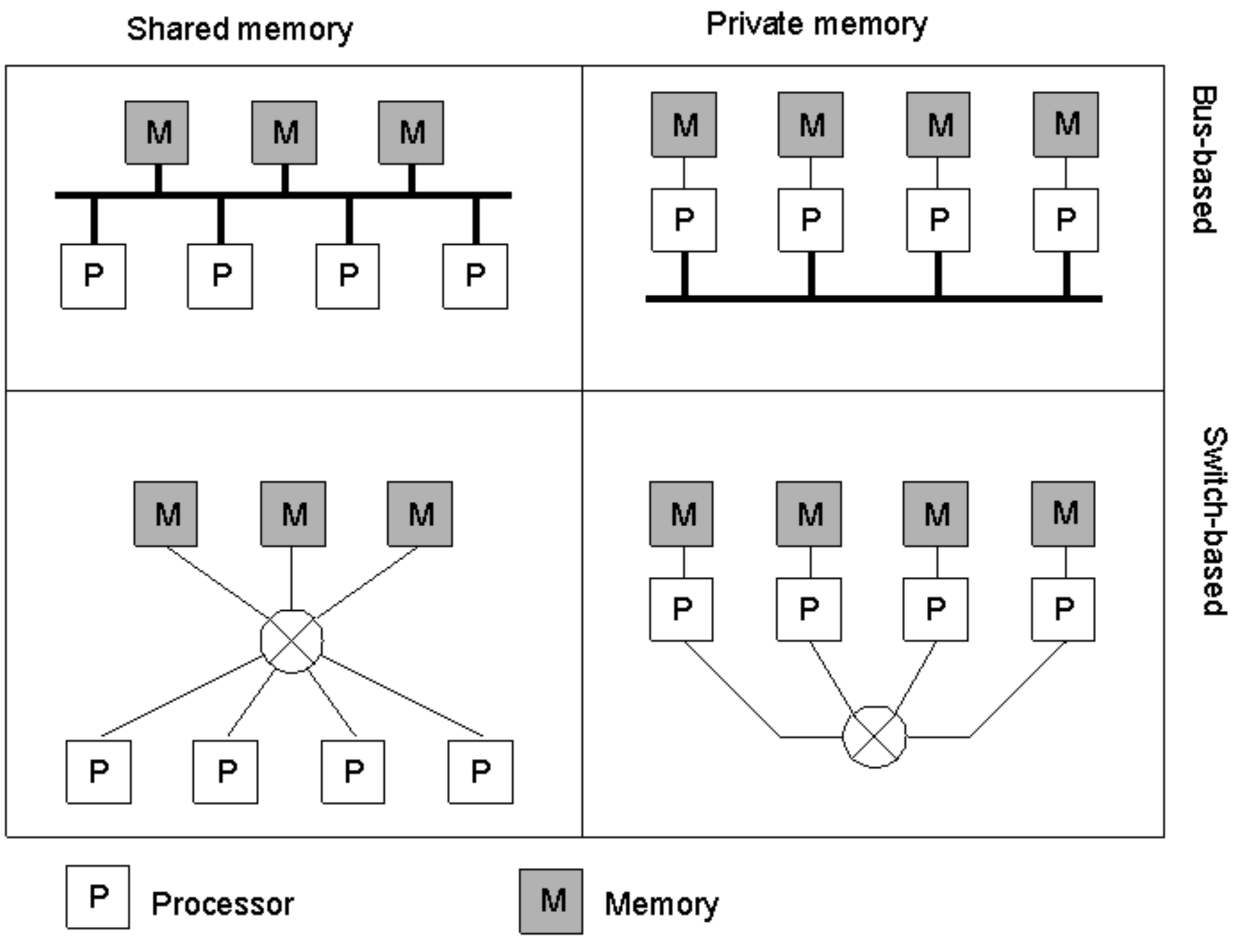
HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

# Where is the borderline between a computer and distributed system?

# Hardware Concepts

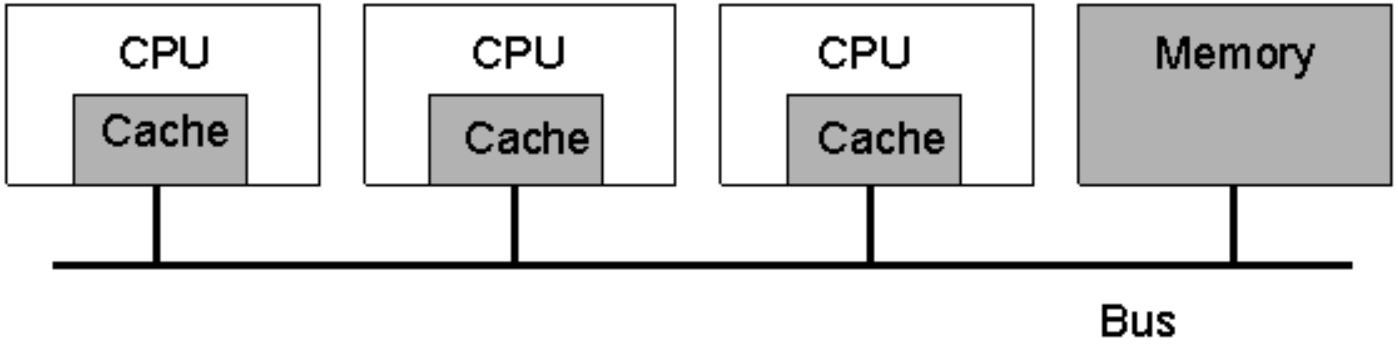
- Characteristics which affect the behavior of software systems
- The platform ....
  - the individual nodes (”computer”, ”processor”)
  - communication between two nodes
  - organization of the system (network of nodes)
- ... and its characteristics
  - capacity of nodes
  - capacity (throughput, delay) of communication links
  - reliability of communication (and of the nodes)
- → Which ways to distribute an application are feasible

# Basic Organizations of a Node




1.6 Different basic organizations and memories in distributed computer systems

# Multiprocessors (1)

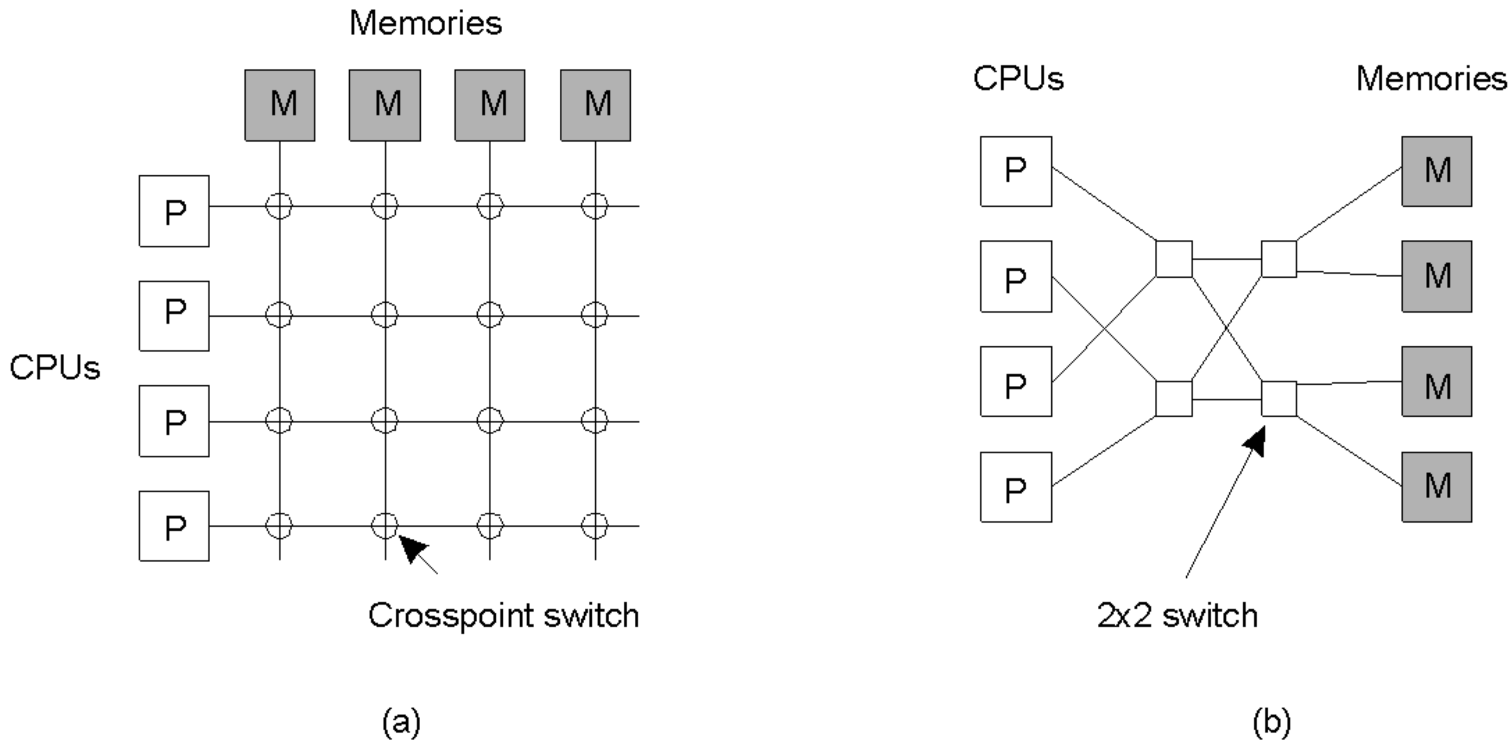



1.7 A bus-based multiprocessor.

Essential characteristics for software design

- fast and reliable communication (shared memory)
  => cooperation at "instruction level" possible
- bottleneck: memory (especially the "hot spots")

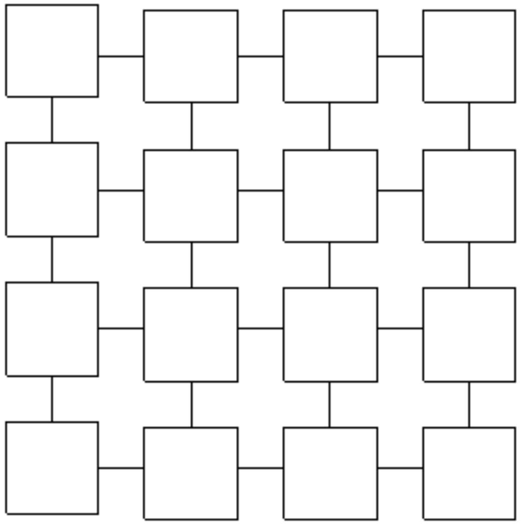
# Multiprocessors (2)



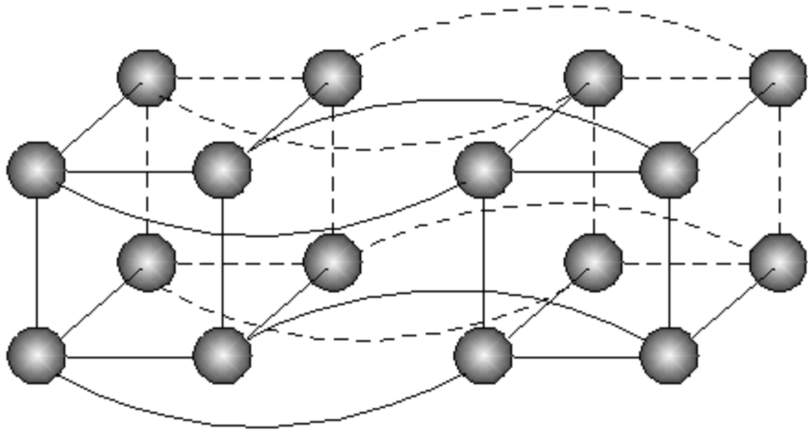

1.8 a) A crossbar switch b) An omega switching network

A possible bottleneck: the switch

# Homogeneous Multicomputer Systems



(a)



(b)

1-9 a) Grid b) Hypercube

A new design aspect: locality at the network level

# General Multicomputer Systems

- Hardware: see Ch1 (internet etc.)
- Loosely connected systems
  - nodes: autonomous
  - communication: slow and vulnerable
  - => cooperation at ”service level”
- Application architectures
  - multiprocessor systems: parallel computation
  - multicomputer systems: distributed systems
  - ( how are parallel, concurrent, and distributed systems different?)

# Software Concepts

| System | Description | Main Goal |
|---|---|---|
| DOS | Tightly-coupled operating system for multiprocessors and homogeneous multicomputers | Hide and manage hardware resources |
| NOS | Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN) | Offer local services to remote clients |
| Middle-ware | Additional layer atop of NOS implementing general-purpose services | Provide distribution transparency |

DOS: Distributed OS; NOS: Network OS

# History of distributed systems

- RPC by Birel &Nelson -84
- network operating systems, distributed operating systems, distributed computing environments in mid-1990; middleware referred to relational databases
- Distributed operating systems – ”single computer”
  - Distributed process management
    - process lifecycle, inter-process communication, RPC, messaging
  - Distributed resource management
    - resource reservation and locking, deadlock detection
  - Distributed services
    - distributed file systems, distributed memory, hierarchical global naming

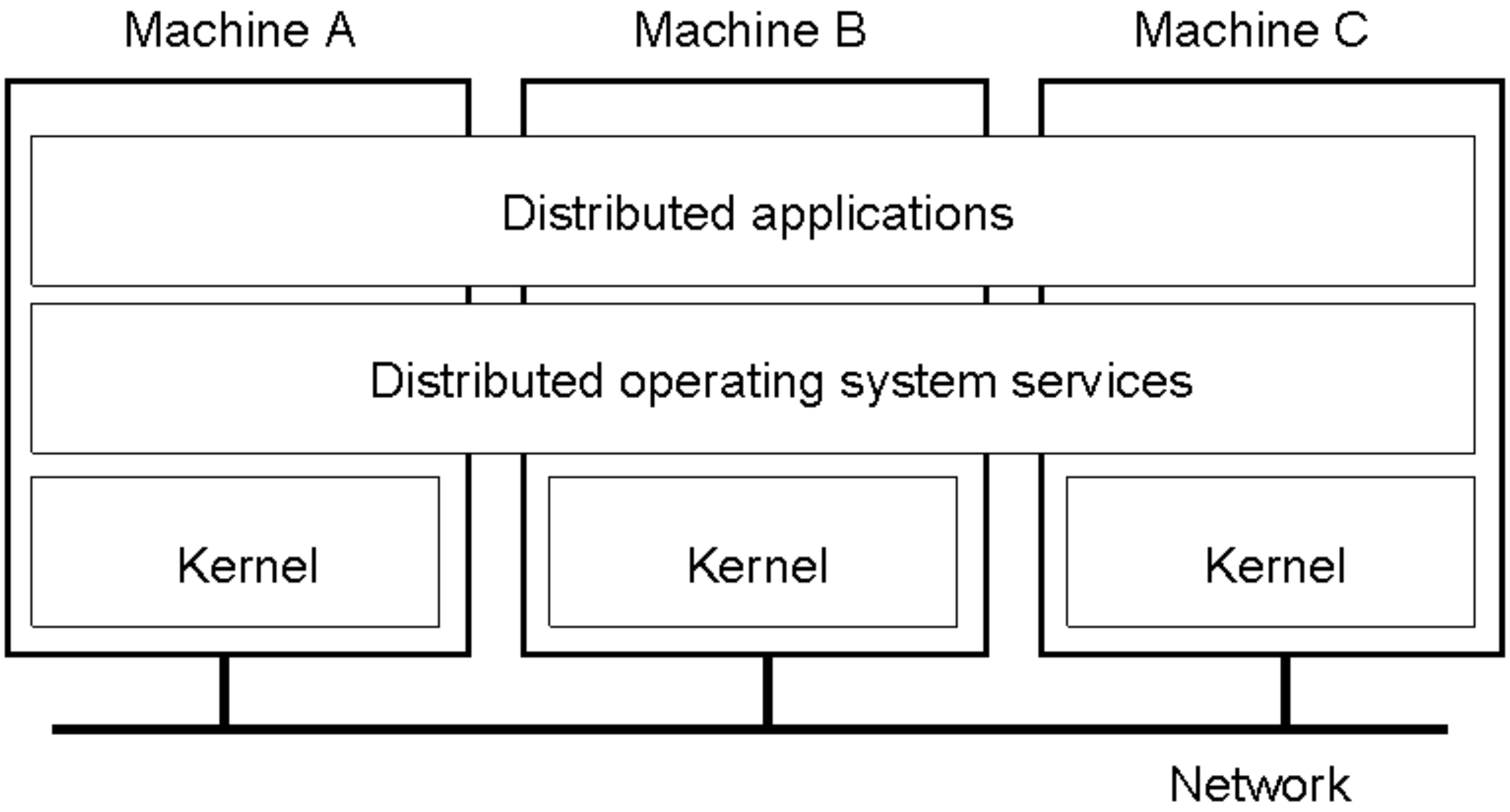
# History of distributed systems

- late 1990's distribution middleware well-known
  - generic, with distributed services
  - supports standard transport protocols and provides standard API
  - available for multiple hardware, protocol stacks, operating systems
  - e.g., DCE, COM, CORBA
- present middlewares for
  - multimedia, realtime computing, telecom
  - ecommerce, adaptive / ubiquitous systems

# Misconceptions tackled

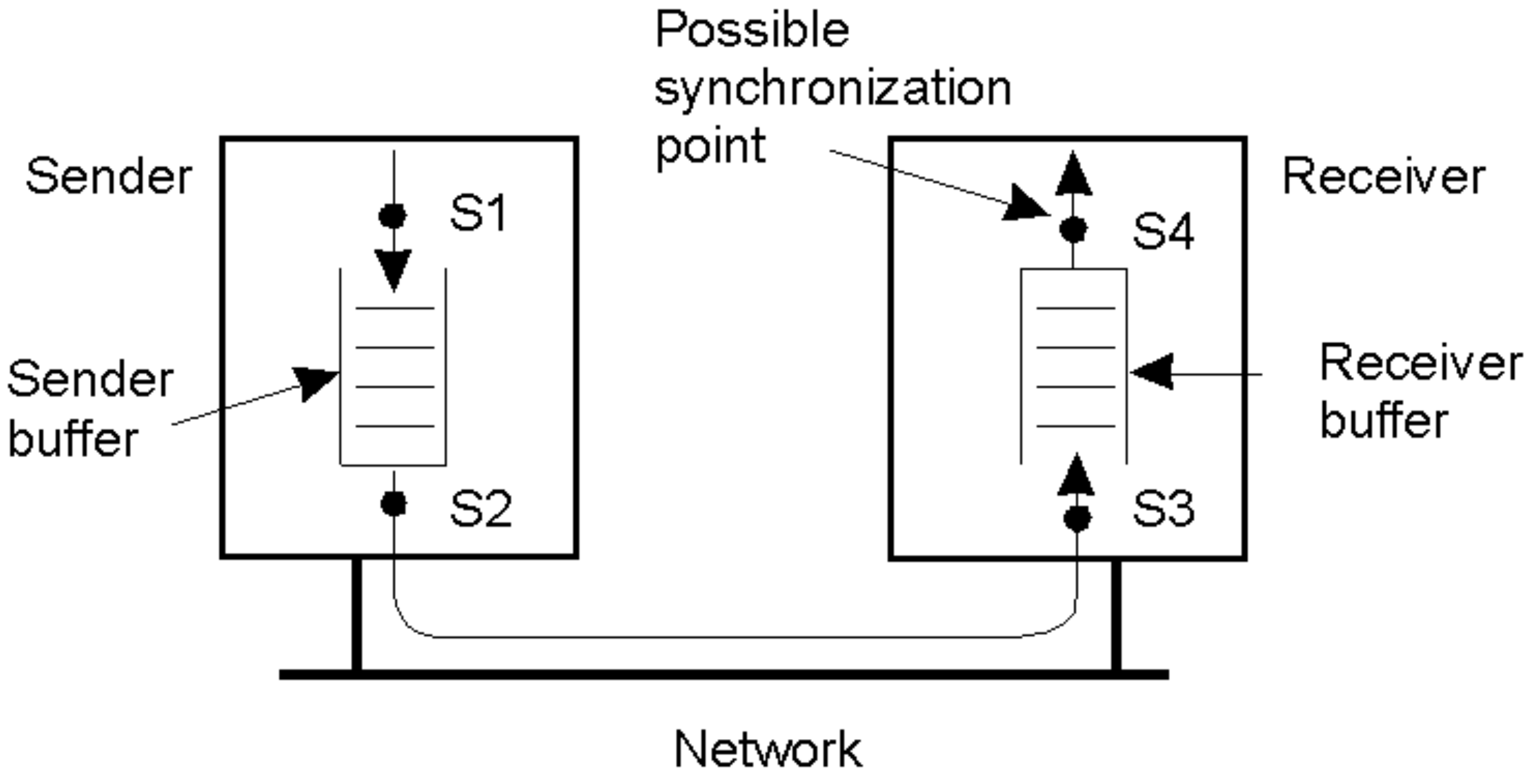
- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero
- Bandwith is infinite
- Transport cost is zero
- There is one administrator
- There is inherent, shared knowledge

# Multicomputer Operating Systems (1)




1.14 General structure of a multicomputer operating system

# Multicomputer Operating Systems (2)



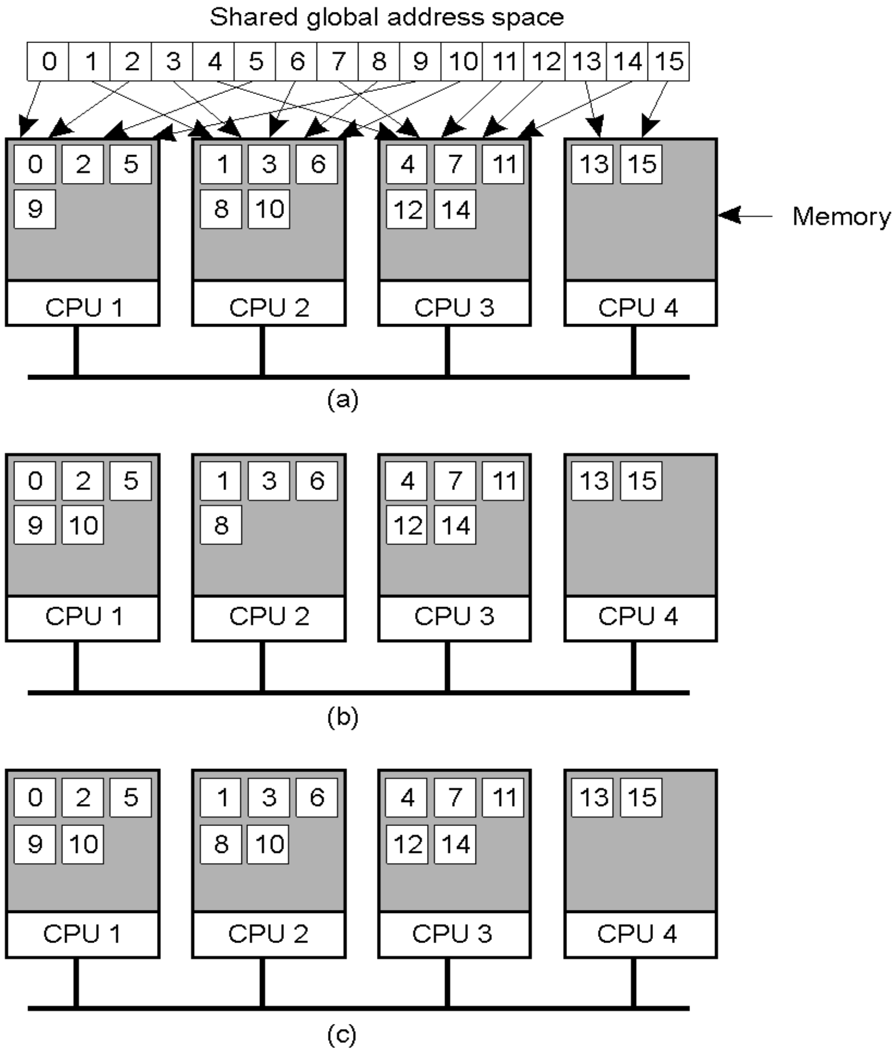

1.15 Alternatives for blocking and buffering in message passing.

# Distributed Shared Memory Systems (1)

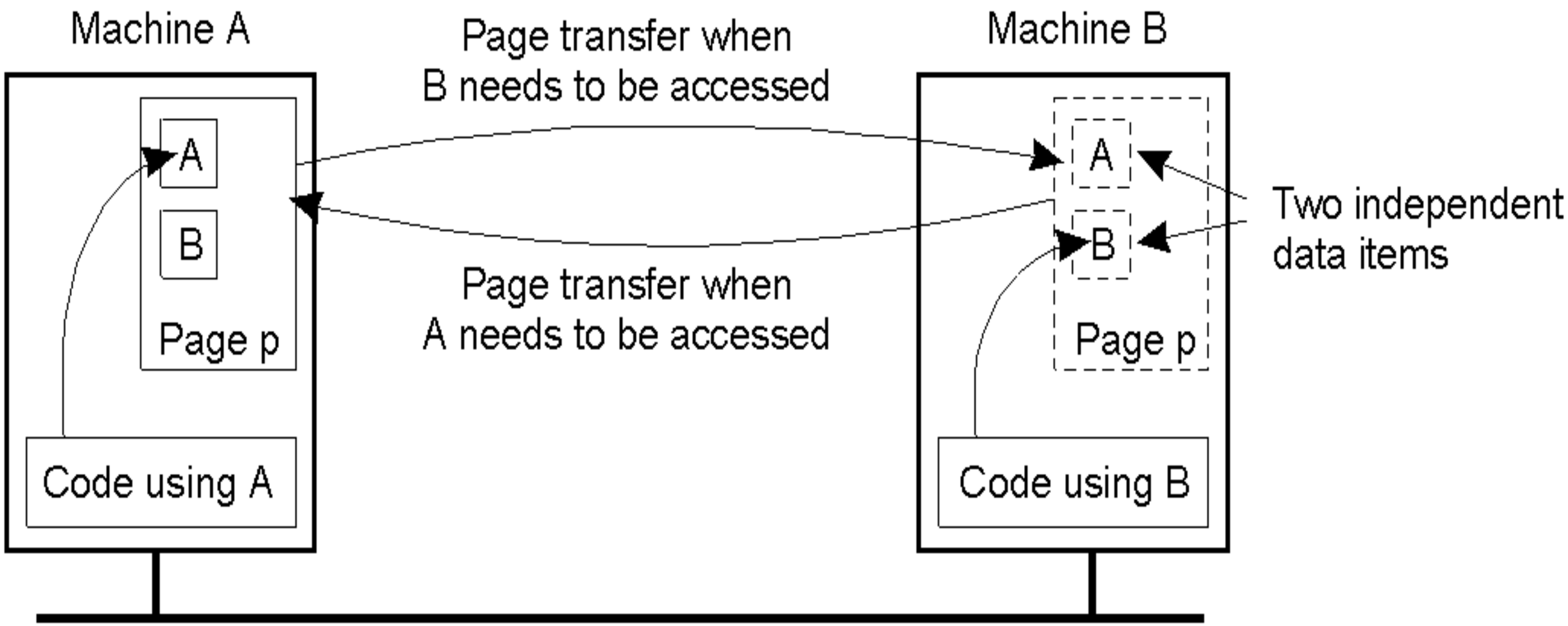
a) Pages of address space distributed among four machines

b) Situation after CPU 1 references page 10

c) Situation if page 10 is read only and replication is used

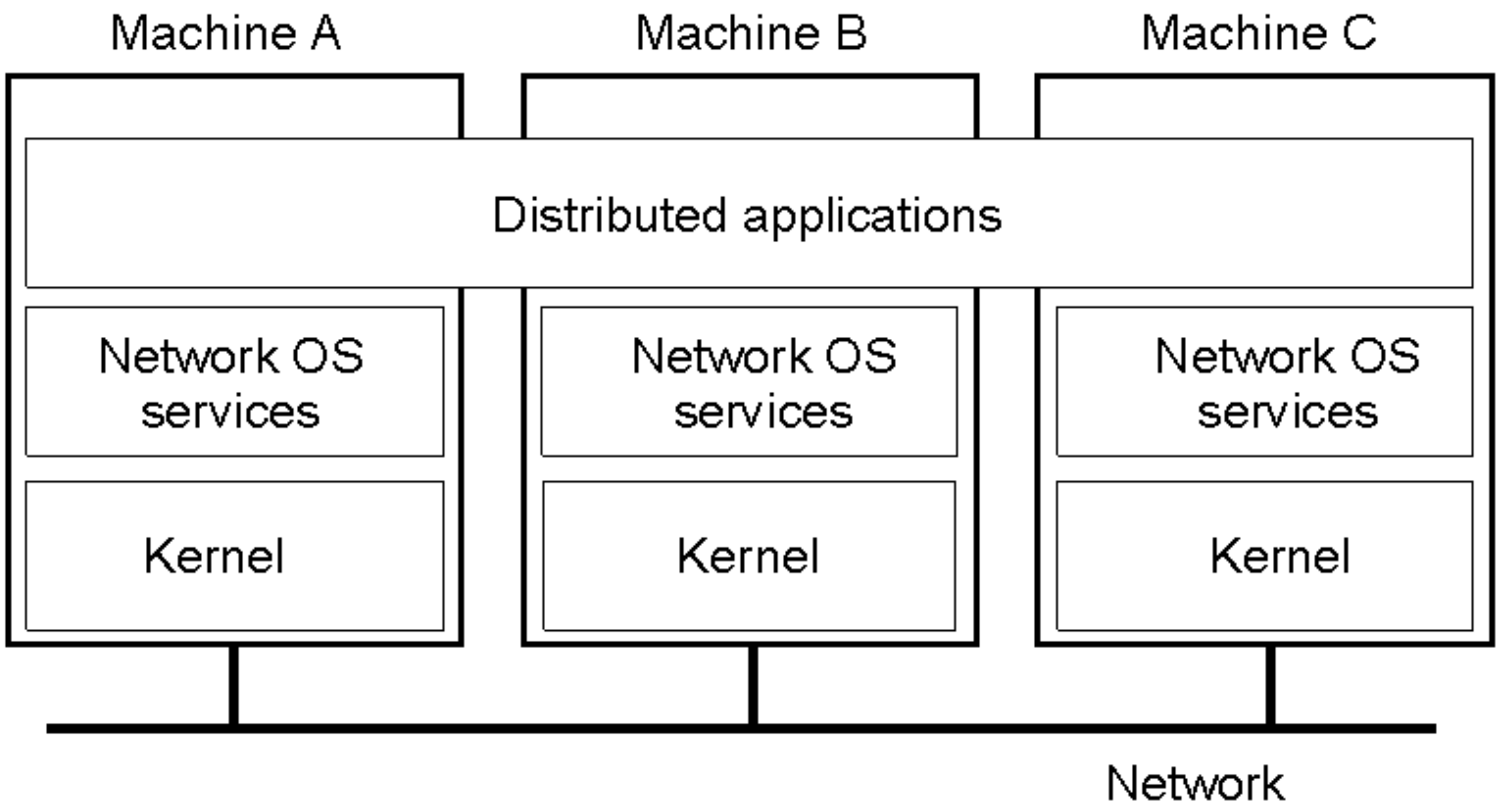


# Distributed Shared Memory Systems (2)



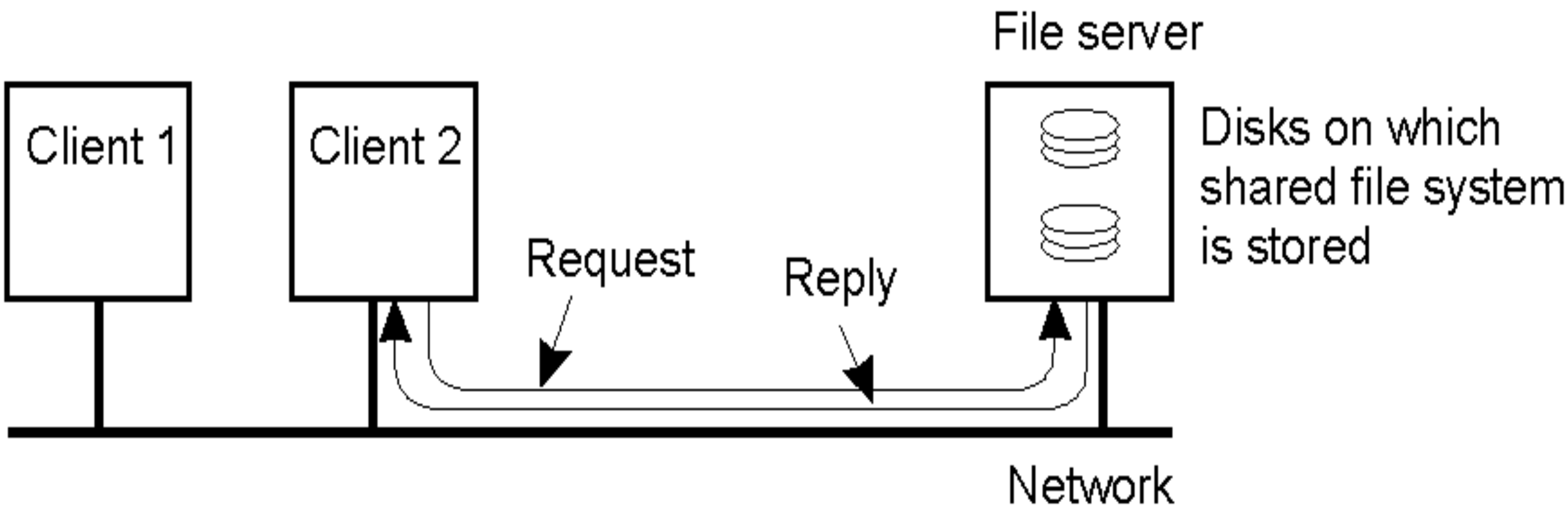

1.18 False sharing of a page between two independent processes.

# Network Operating System (1)




1-19 General structure of a network operating system.

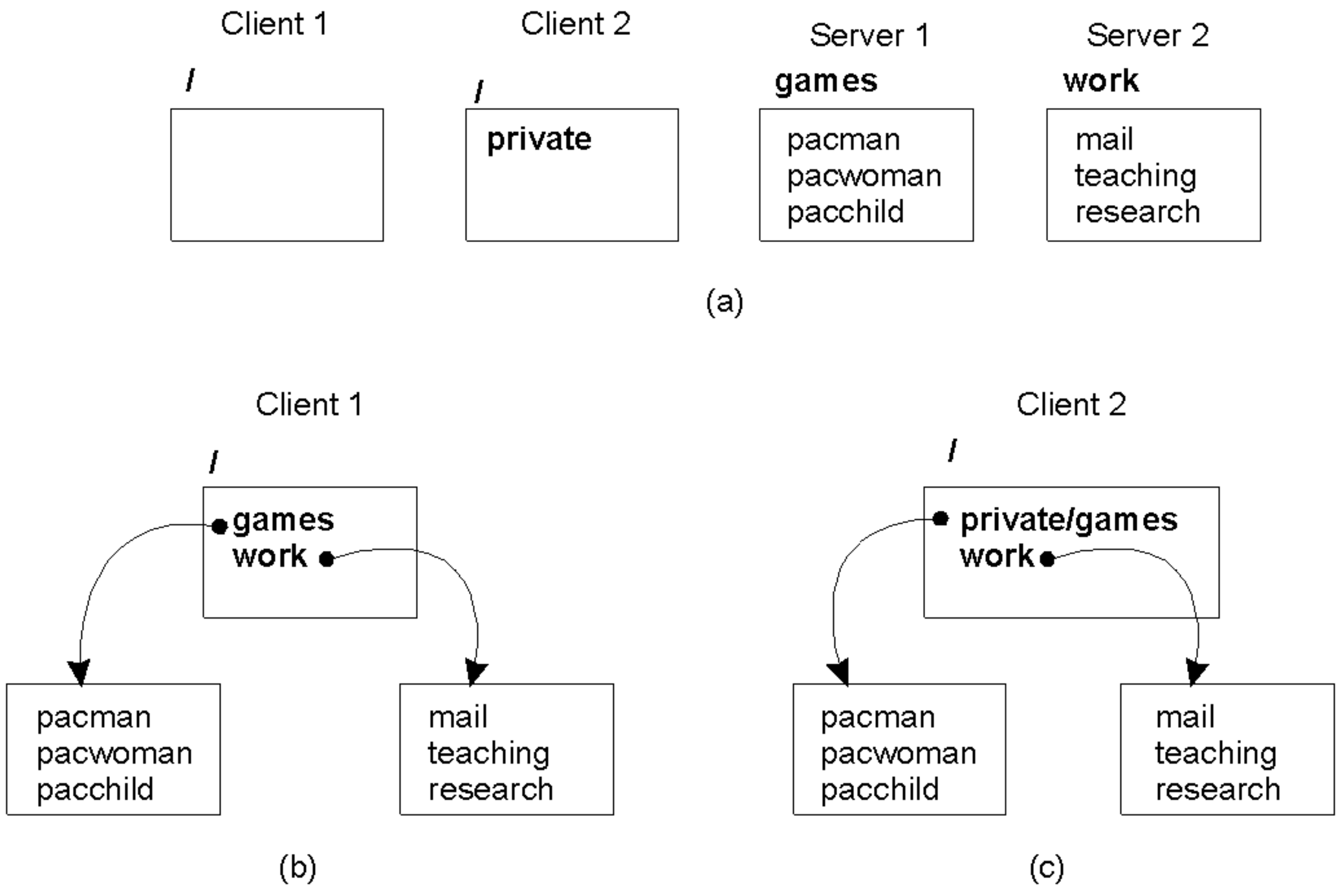
# Network Operating System (2)




1-20

Two clients and a server in a network operating system.

# Network Operating System (3)

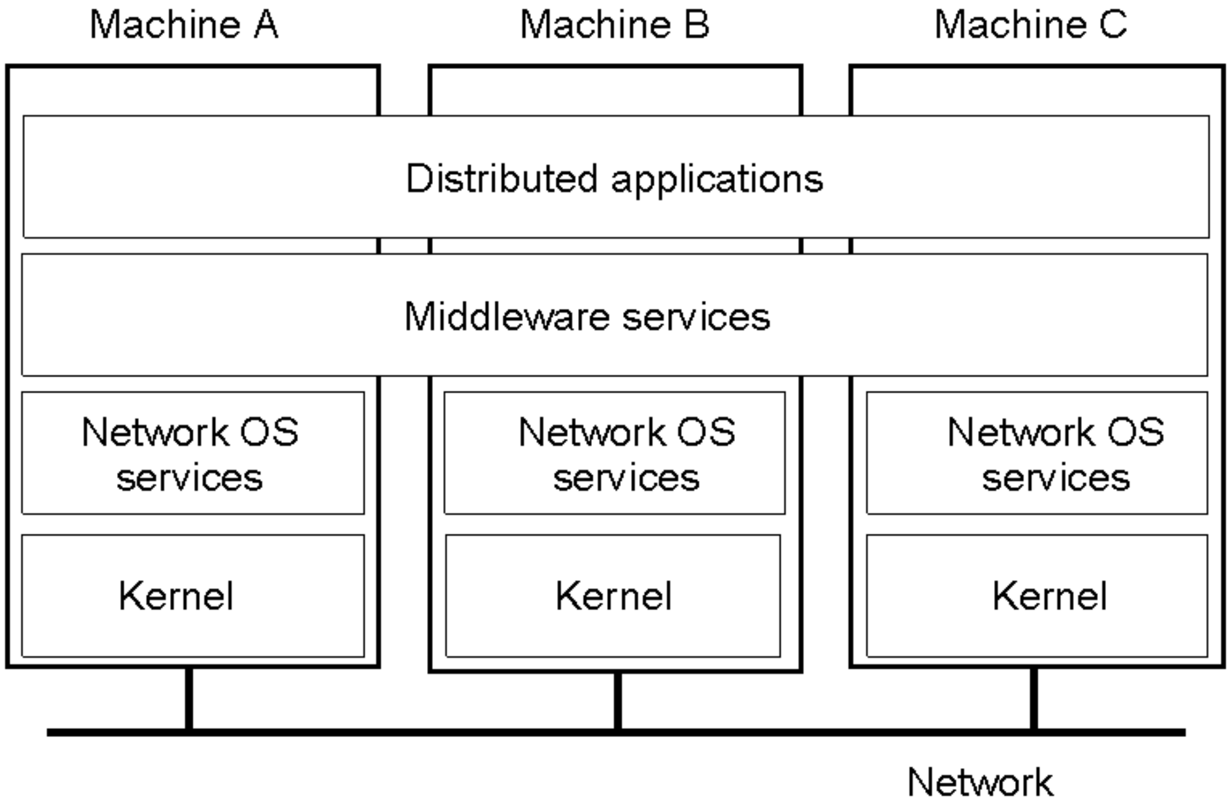



1.21 Different clients may mount the servers in different places.

## Software Layers

- Platform: computer & operating system & ..
- Middleware:
  - mask heterogeneity of lower levels
  - (at least: provide a homogeneous “platform”)
  - mask separation of platform components
    - implement communication
    - implement sharing of resources
- Applications: e-mail, www-browsers, …

# Positioning Middleware




1-22 General structure of a distributed system as middleware.

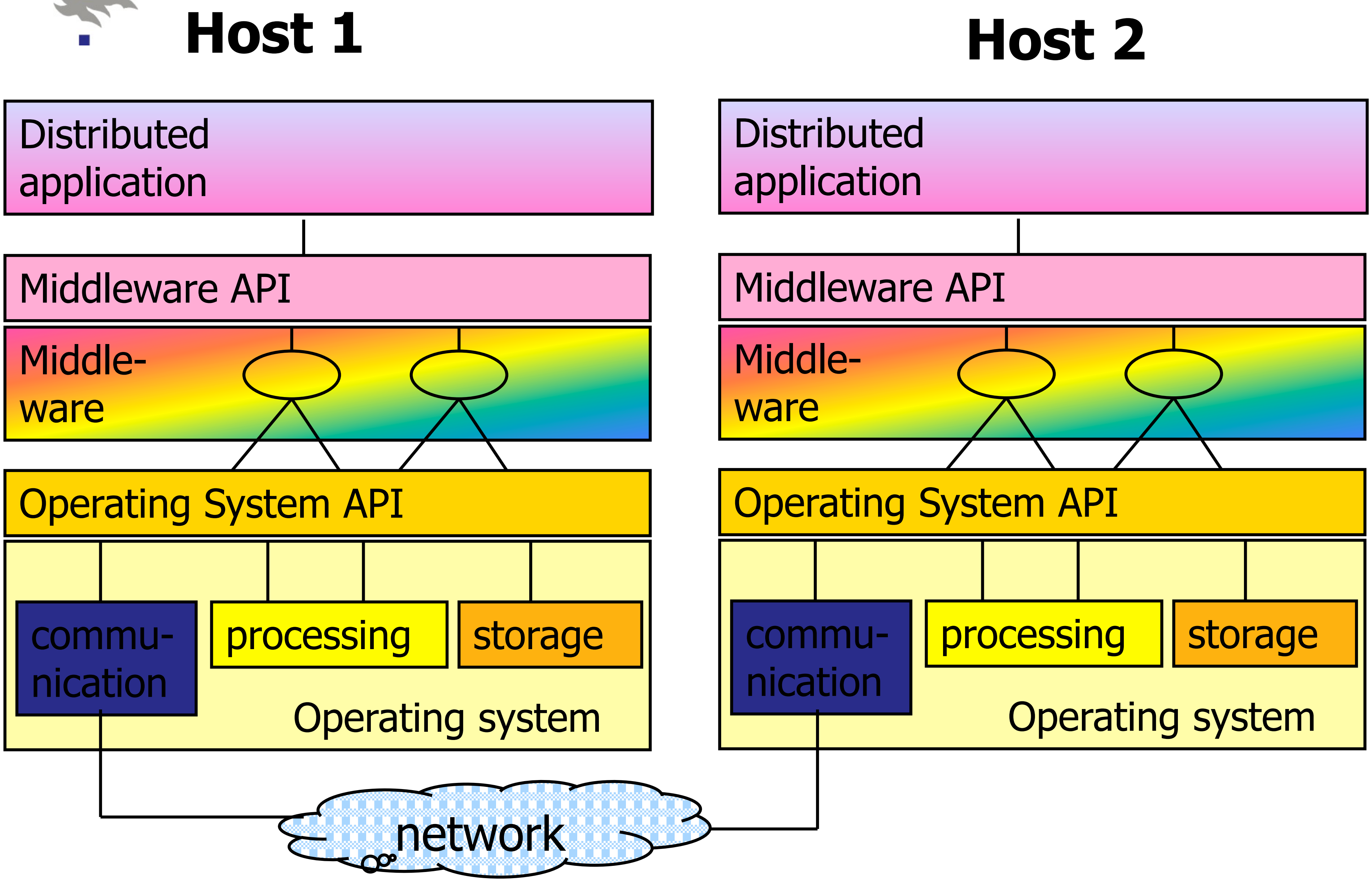
# Middleware

- Operations offered by middleware
  - RMI, group communication, notification, replication, … (Sun RPC, CORBA, Java RMI, Microsoft DCOM, ...)
- Services offered by middleware
  - naming, security, transactions, persistent storage, …
- Limitations
  - ignorance of special application-level requirements

End-to-end argument:

- Communication of application-level peers at both ends is required for reliability

# Middleware



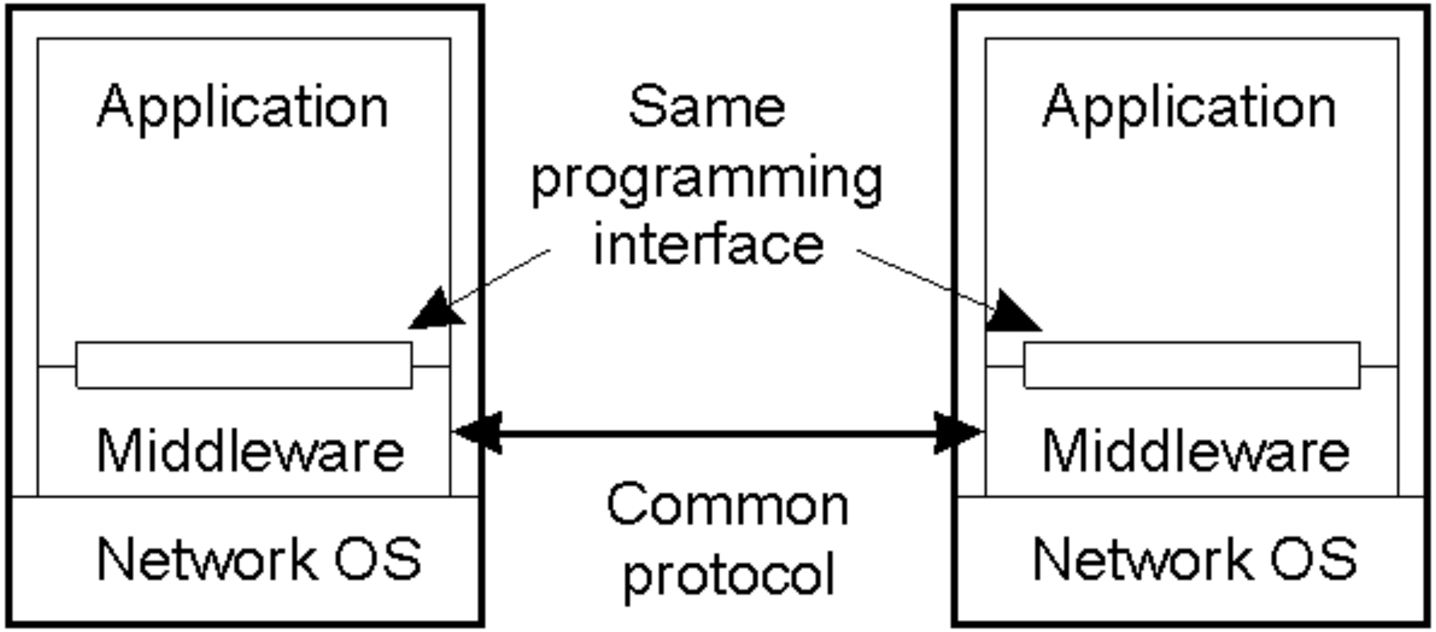
Host 1
Host 2
Distributed application
Middleware API
Middle-ware
Operating System API
commu-nication
processing
storage
Operating system
Distributed application
Middleware API
Middle-ware
Operating System API
commu-nication
processing
storage
Operating system
network

# Middleware

Middleware is a class of software technologies designed to help manage the complexity and heterogeneity inherent in distributed systems. It is defined as a layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system.

Bakken 2001: Encyclopedia entry

# Middleware and Openness




1.23 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.

# Comparison between Systems

| Item | Distributed OS | | Network OS | Middleware-based OS |
|---|---|---|---|---|
| | Multiproc. | Multicomp. | | |
| Degree of transparency | Very High | High | Low | High |
| Same OS on all nodes | Yes | Yes | No | No |
| Number of copies of OS | 1 | N | N | N |
| Basis for communication | Shared memory | Messages | Files | Model specific |
| Resource management | Global, central | Global, distributed | Per node | Per node |
| Scalability | No | Moderately | Yes | Varies |
| Openness | Closed | Closed | Open | Open |

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

# More examples on distributed software architectures

# Architectural Models

- Architectural models provide a high-level view of the distribution of functionality between system components and the interaction relationships between them
- Architectural models define
  - components (logical components deployed at physical nodes)
  - communication
- Criteria
  - performance
  - reliability
  - scalability, ..

# Client-Server

- Client-server model: CoDoKi, Fig. 2.2
- Service provided by multiple servers: Fig. 2.3
- Needed:
  - name service
  - trading/broker service
  - browsing service
- Proxy servers and caches, Fig. 2.4

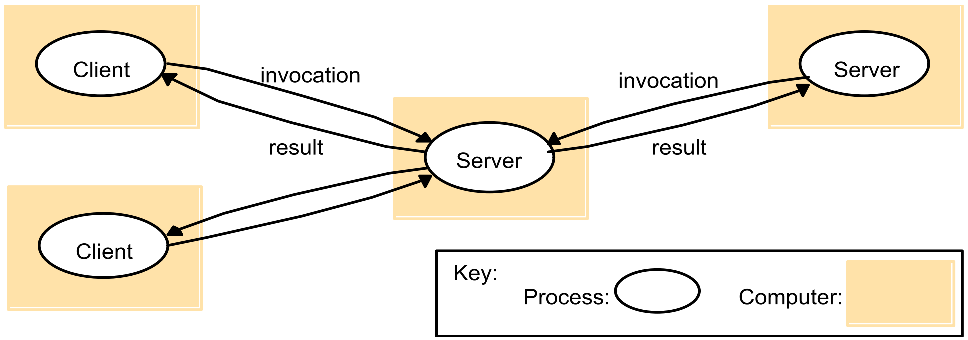
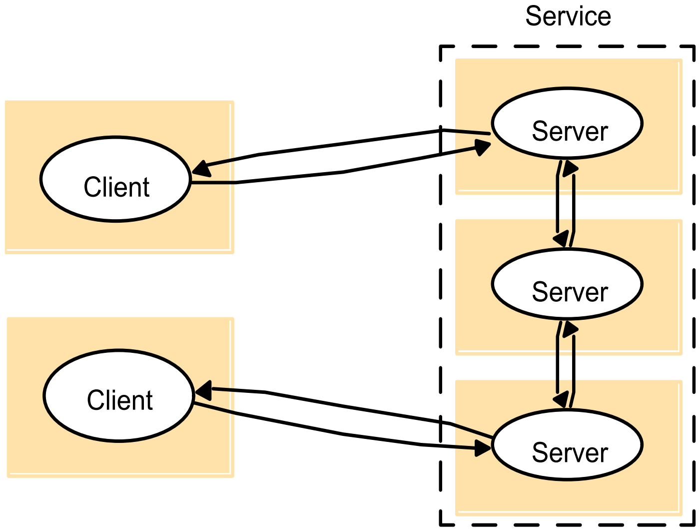



Figure 2.2 Clients invoke individual servers

CoDoKi, Fig. 2.2




CoDoKi, Fig. 2.3

Figure 2.3 A service provided by multiple servers

CoDoKi, Fig. 2.4

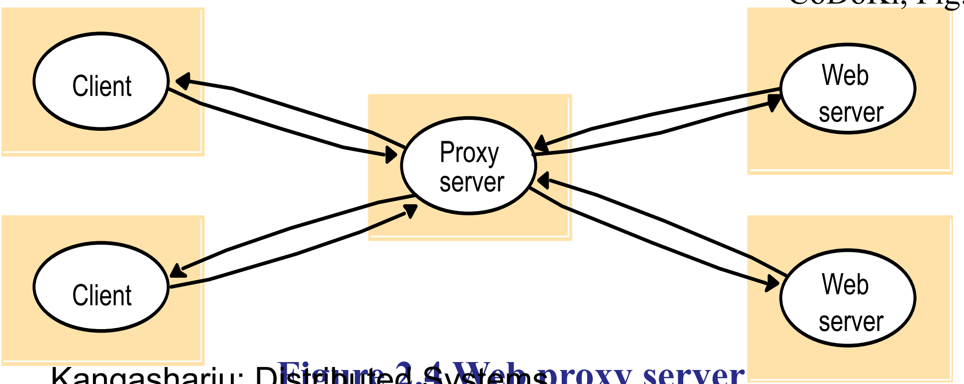



Figure 2.4 Web proxy server

# An Example Client and Server (1)

```
/* Definitions needed by clients and servers.                */
#define TRUE              1
#define MAX_PATH          255   /* maximum length of file name          */
#define BUF_SIZE          1024  /* how much data to transfer at once    */
#define FILE_SERVER       243   /* file server's network address        */

/* Definitions of the allowed operations */
#define CREATE            1     /* create a new file                    */
#define READ              2     /* read data from a file and return it  */
#define WRITE             3     /* write data to a file                 */
#define DELETE            4     /* delete an existing file              */

/* Error codes. */
#define OK                0     /* operation performed correctly        */
#define E_BAD_OPCODE      -1    /* unknown operation requested          */
#define E_BAD_PARAM       -2    /* error in a parameter                 */
#define E_IO              -3    /* disk error or other I/O error        */

/* Definition of the message format. */
struct message {
    long source;                /* sender's identity                    */
    long dest;                  /* receiver's identity                  */
    long opcode;                /* requested operation                  */
    long count;                 /* number of bytes to transfer          */
    long offset;                /* position in file to start I/O        */
    long result;                /* result of the operation              */
    char name[MAX_PATH];        /* name of file being operated on       */
    char data[BUF_SIZE];        /* data to be read or written           */
};
```

The *header.h* file used by the client and server.

## An Example Client and Server (2)

```
#include <header.h>
void main(void) {
      struct message ml, m2;                   /* incoming and outgoing messages       */
      int r;                                   /* result code                          */

      while(TRUE) {                            /* server runs forever                  */
            receive(FILE_SERVER, &ml);         /* block waiting for a message          */
            switch(ml.opcode) {                /* dispatch on type of request          */
                  case CREATE:      r = do_create(&ml, &m2); break;
                  case READ:        r = do_read(&ml, &m2); break;
                  case WRITE:       r = do_write(&ml, &m2); break;
                  case DELETE:      r = do_delete(&ml, &m2); break;
                  default:          r = E_BAD_OPCODE;
            }
            m2.result = r;                     /* return result to client              */
            send(ml.source, &m2);              /* send reply                           */
      }
}
```

A sample server.

# An Example Client and Server (3)

(a)

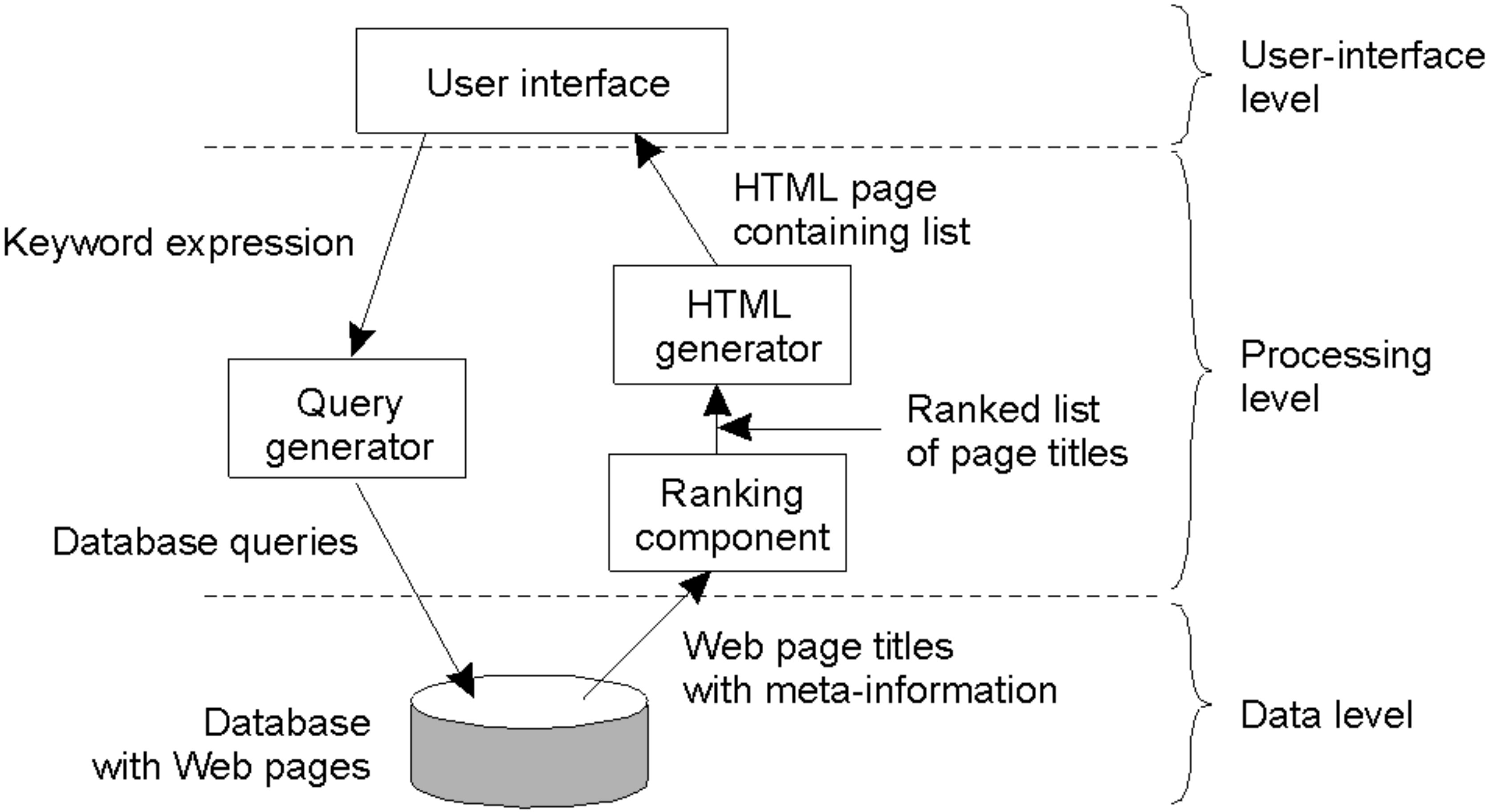
```
#include <header.h>
int copy(char *src, char *dst){                * procedure to copy file using the server      */
      struct message ml;                       ,* message buffer                               */
      long position;                           /* current file position                        */
      long client = 110;                       /* client's address                             */

      initialize( );                           /* prepare for execution                        */
      position = 0;
      do {
            ml.opcode = READ;                  /* operation is a read                          */
            ml.offset = position;              /* current position in the file                 */
            ml.count  = BUF_SIZE;                                                      /* how many bytes to read*/
            strcpy(&ml.name, src);             /* copy name of file to be read to message      */
            send(FILESERVER, &ml);             /* send the message to the file server          */
            receive(client, &ml);              /* block waiting for the reply                  */

            /* Write the data just received to the destination file.                          */
            ml.opcode = WRITE;                 /* operation is a write                         */
            ml.offset = position;              /* current position in the file                 */
            ml.count  = ml.result;             /* how many bytes to write                      */
            strcpy(&ml.name, dst);             /* copy name of file to be written to buf       */
            send(FILE_SERVER, &ml);            /* send the message to the file server          */
            receive(client, &ml);              /* block waiting for the reply                  */
            position += ml.result;             /* ml.result is number of bytes written         */
      } while( ml.result > 0 );                /* iterate until done                           */
      return(ml.result >= 0 ? OK : ml result); /* return OK or error code                      */
}
```

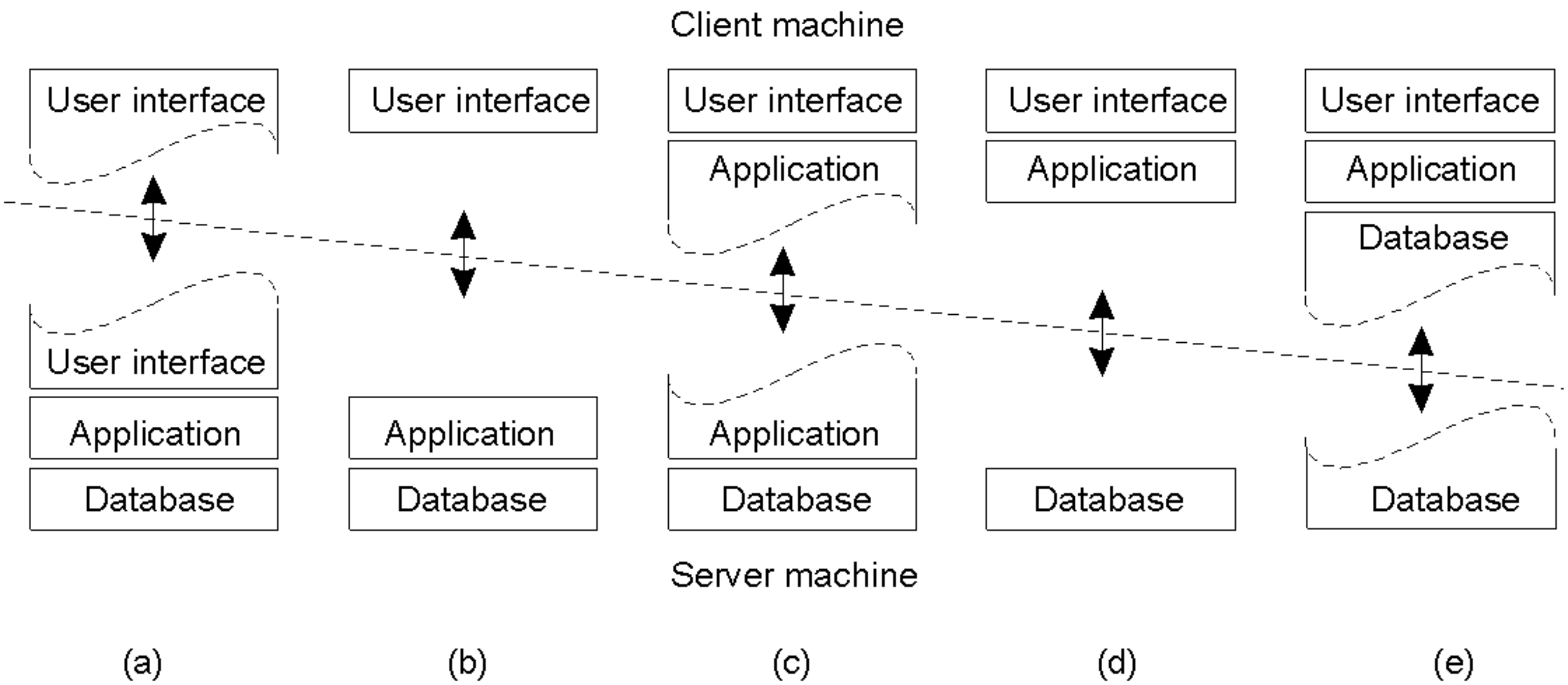
A client using the server to copy a file.

# Processing Level




1-28 The general organization of an Internet search engine into three different layers

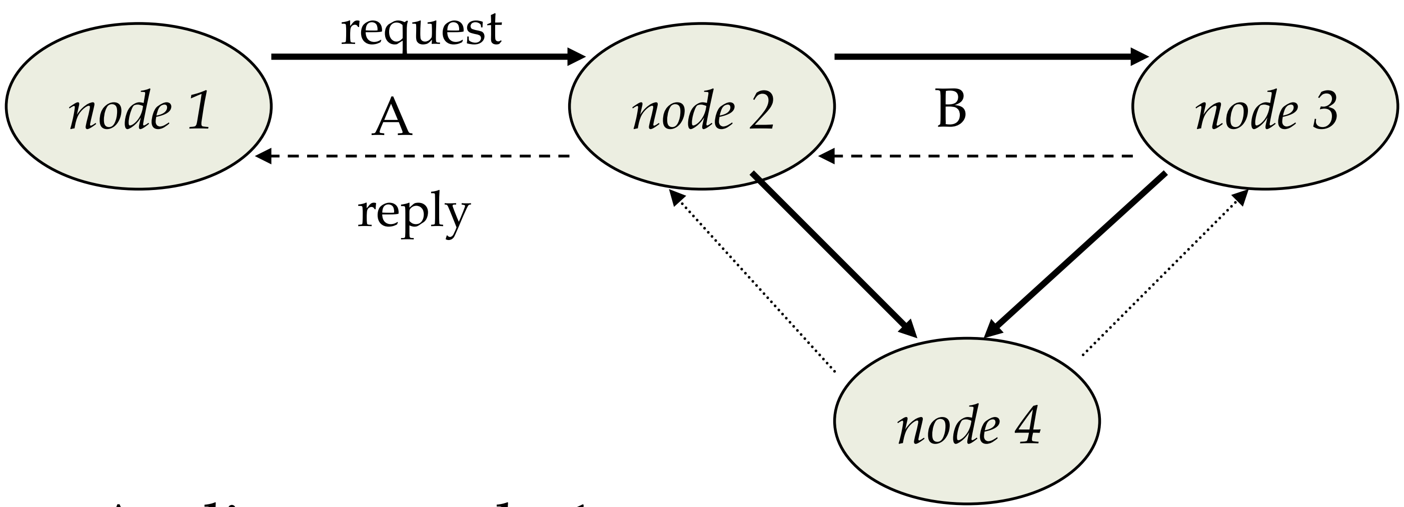
# Multitiered Architectures (1)




1-29 Alternative client-server organizations.

# Multitiered Architectures (2)

Client - server: generalizations

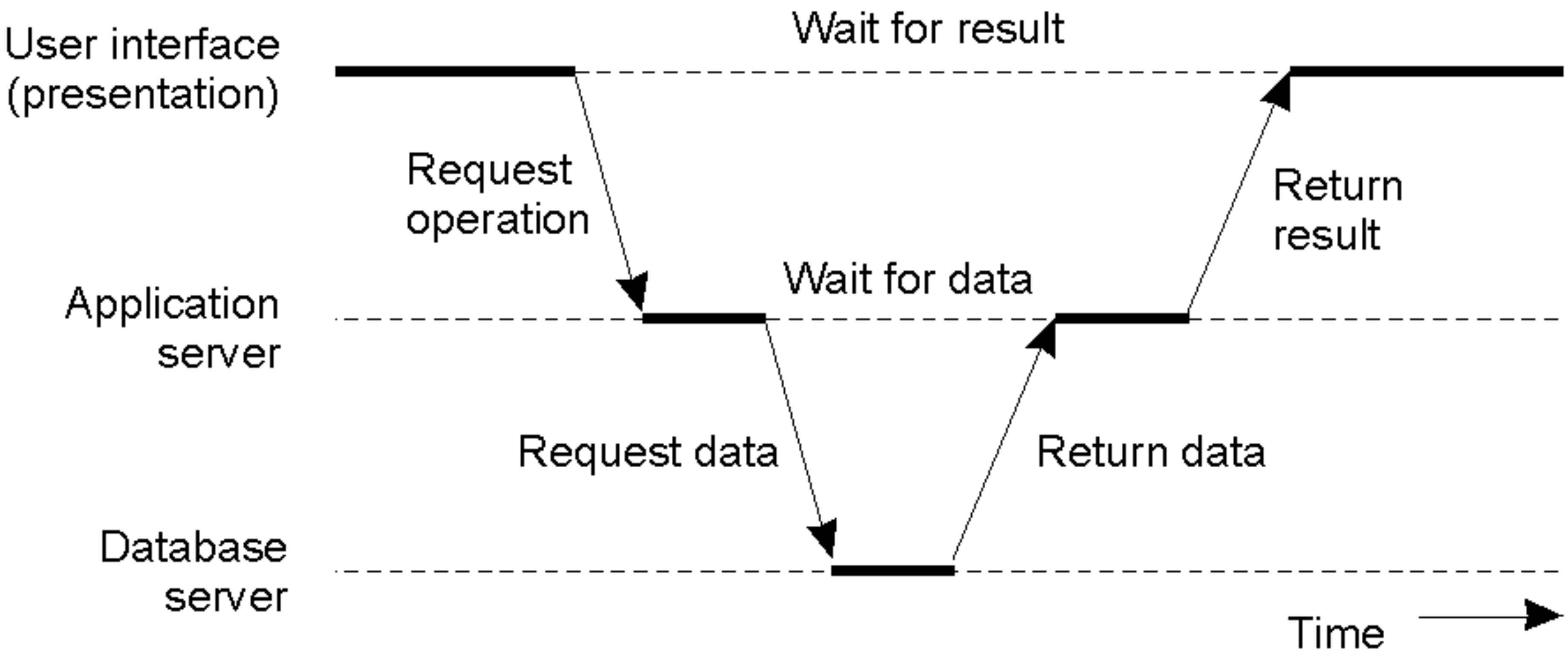



A client: node 1
server: node 2

B client: node 2
server: node 3

the concept is related to communication not to nodes

# Multitiered Architectures (3)



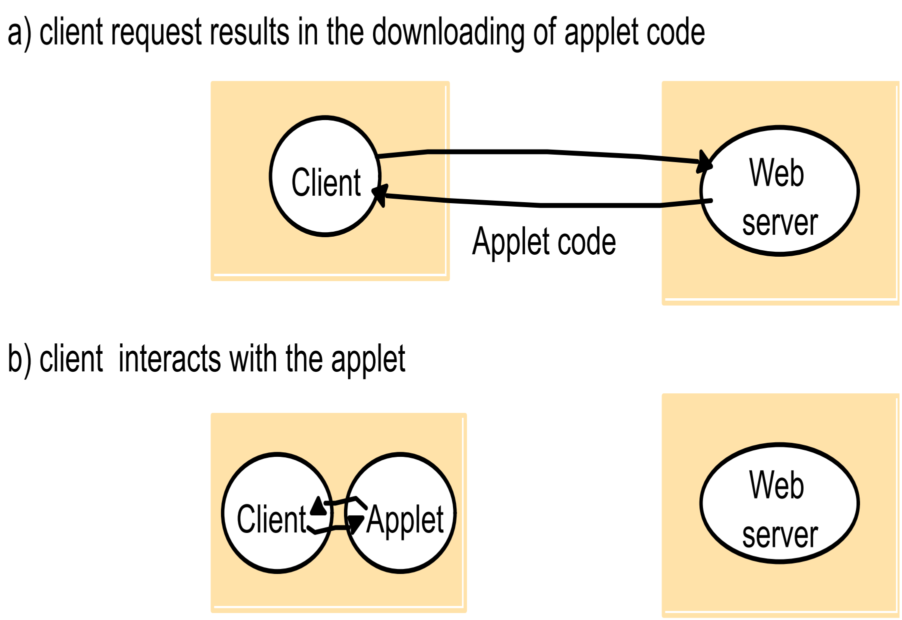

1-30 An example of a server acting as a client.

# Variations on the Client-Server model

- Mobile code

  the service is provided using a procedure

  - executed by a process in the server node
  - downloaded to the client and executed locally Fig. 2.6
  - push service: the initiator is the server
- Mobile agents
  - "a running program" (code & data) travels
  - needed: an agent platform




CoDoKi, Fig. 2.6 **Figure 2.6 Web applets**

# Variations on the Client-Server model (cont.)

- Network computers
  - “diskless workstations”
  - needed code and data downloaded for execution
- Thin clients
  - “PC”: user interface
  - server: execution of computations (Fig. 2.7)
  - example: Unix X-11 window system

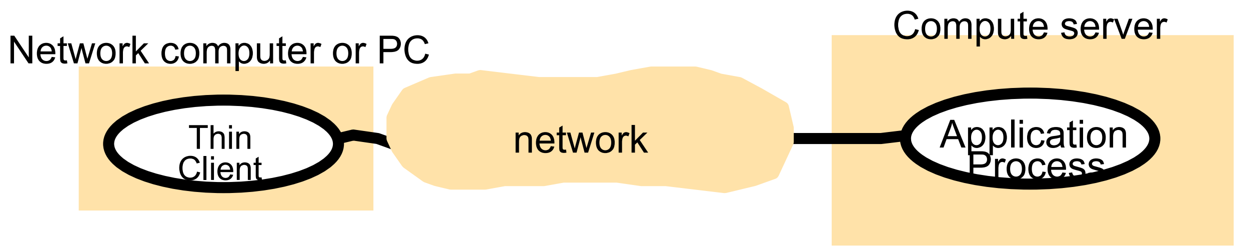



**Figure 2.7 Thin clients and compute servers**

CoDoKi, Fig. 2.7

# Variations on the Client-Server model (cont.)

- Mobile devices and spontaneous networks, ad hoc networks (Fig. 2.8)
- Needed
  - easy connection to a local network
  - easy integration with local services
- Problems
  - limited connectivity
  - security and privacy
- Discovery service

  two interfaces:
  registration, lookup

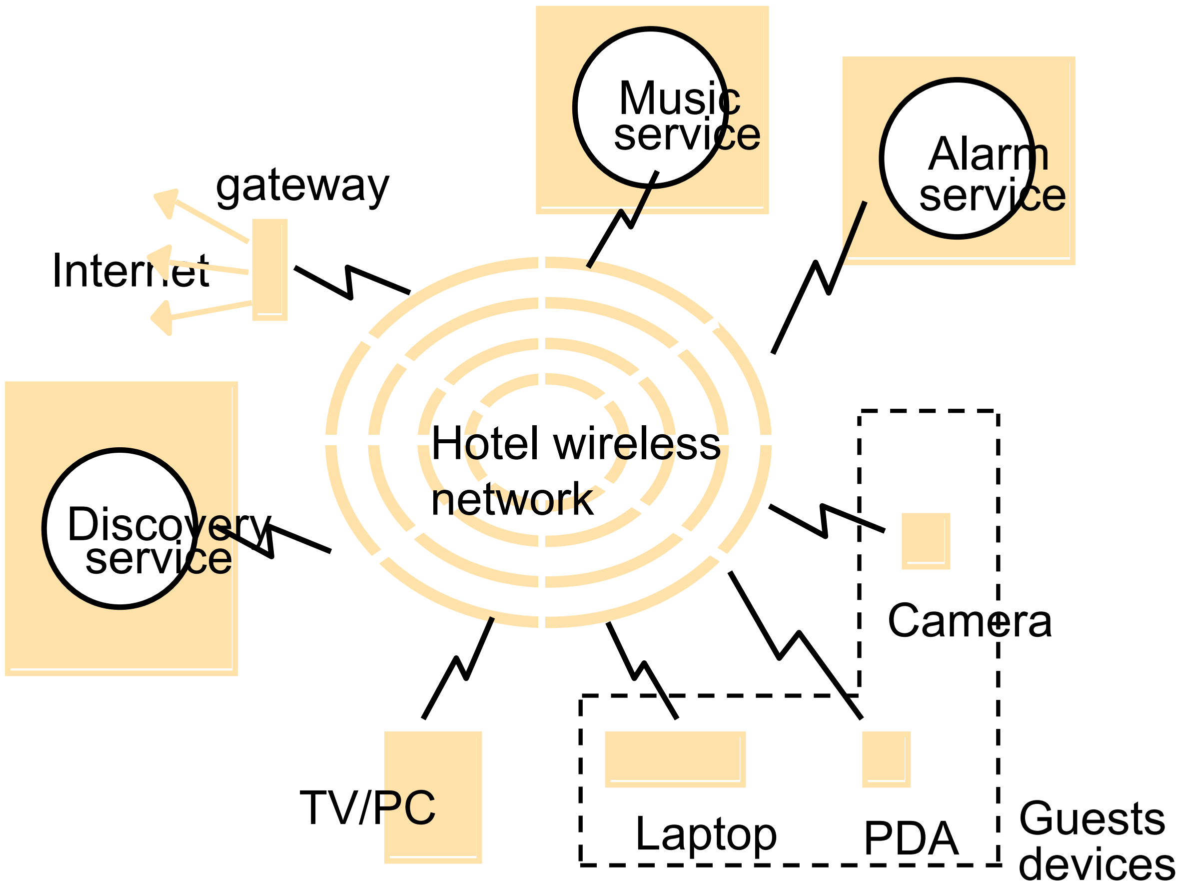
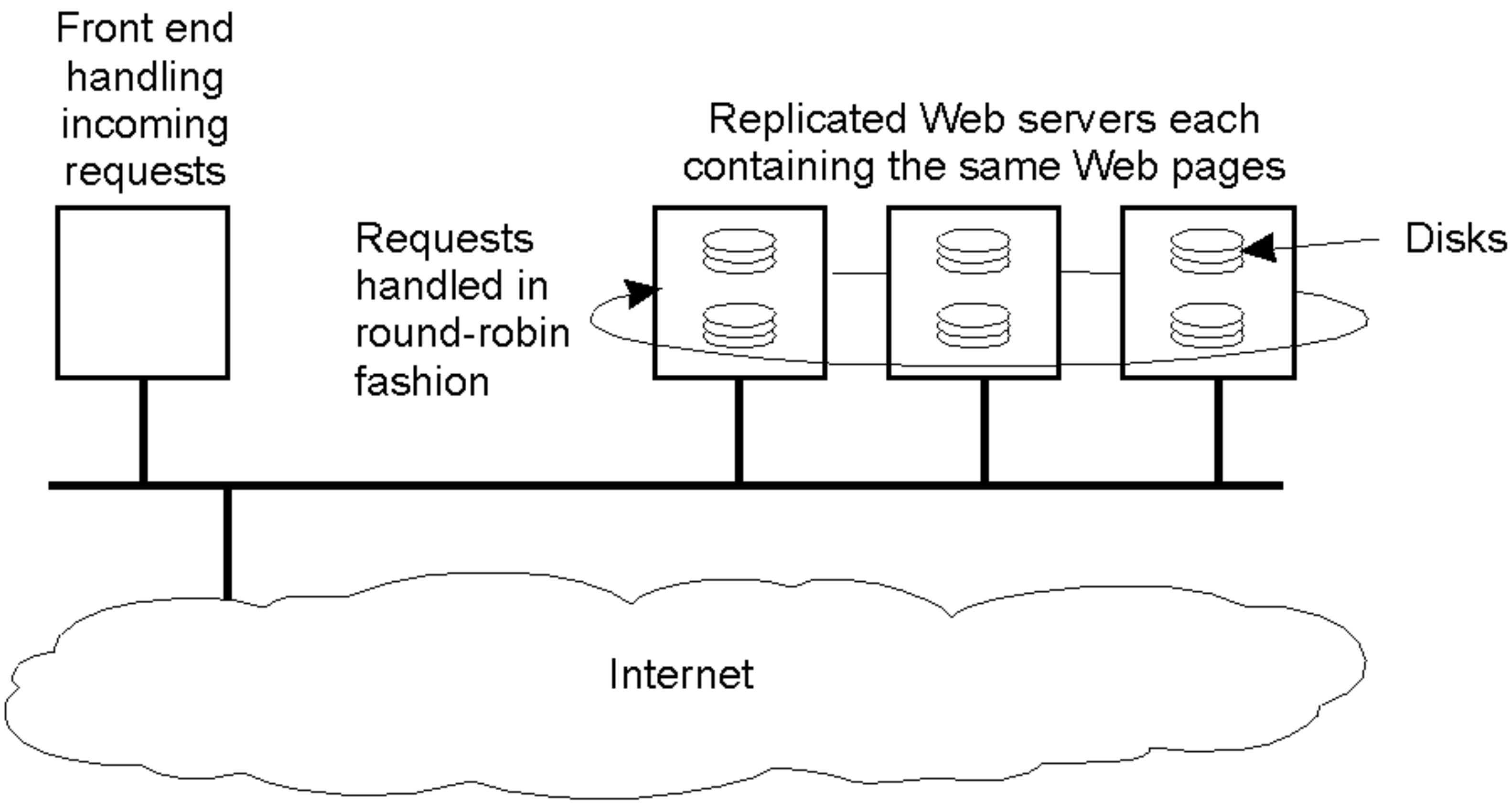



**Figure 2.8 Spontaneous networking in a hotel**

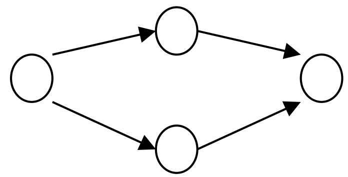
# Modern Architectures



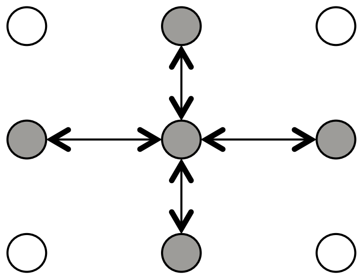

1-31 An example of horizontal distribution of a Web service.

# Other Architectures

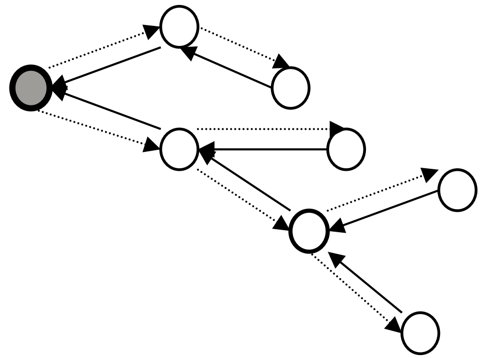
- Andrews paradigms:
  filter: a generalization of producers and consumers



heartbeat

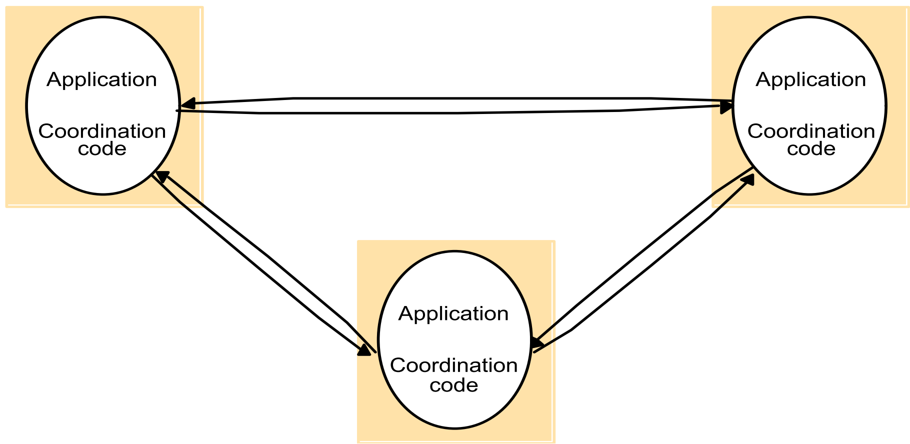


probe echo



- Peer to peer

CoDoKi, Fig. 2.5



# Chapter Summary

- Introduction into distributed systems
- Challenges and goals of distributing
- Examples of distributed systems