HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

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

Fall 2012

Sini Ruohomaa

(Slides joint work with Jussi Kangasharju et al.
Figures from course material)

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.

Where Does the Definition Leave Us?

- Which of the following are distributed systems?

Multi-core processor

Internet

Computing cluster

My workroom electronics

Multi-processor computer

Corporate intranet

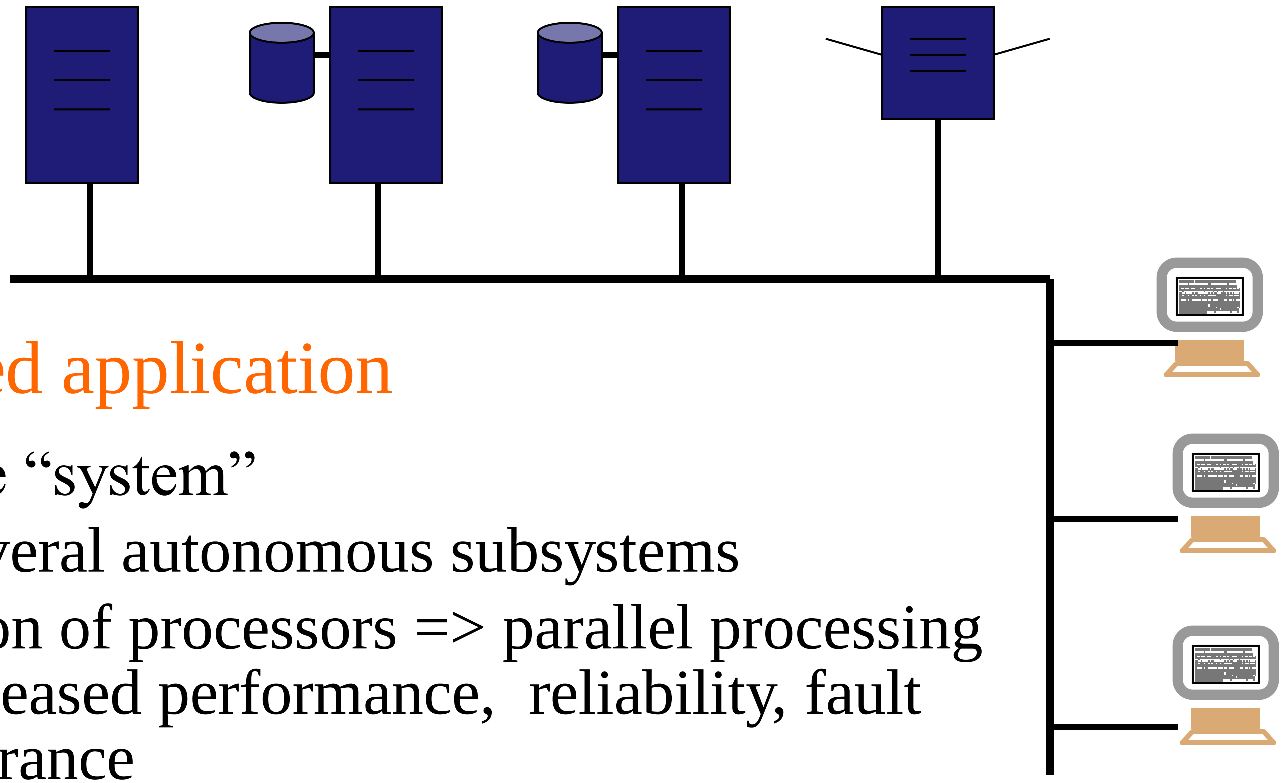
One data center

Local Area Network

Web

Network of data centers

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

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Goals and challenges for distributed systems

Getting a feel of the playing field

Goals

- Making resources accessible
- Openness
- Scalability
- Security
- Fitting the given concrete environment
- Fulfilling system design requirements
- Distribution transparency

- What could go wrong?

Challenges for Making Resources Accessible

- Goal: should be easy for users to access/share resources
- What it takes to achieve this:
  - Naming
  - Access control
  - Security
  - Availability
  - Performance
  - Mutual exclusion of users, fairness
  - Consistency in some cases

Challenges for Openness

- Goal: follow standard rules, allow different players on field
  - Interoperability: allow different solutions to coexist
  - Portability: solution executable as is in different systems
  - Extensibility: simple to add new components, or
  - Possible to reimplement (by independent providers)
- Supported by
  - Public, well-specified interfaces
  - Standardized communication protocols
  - Separation of policy (rules of use) from mechanism (functionality available for use): allows change of policy

Challenges for Scalability (1/2)

Scalability:

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

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

Challenges for Scalability (2/2)

- Controlling the cost of physical resources
- Controlling performance loss
- Preventing software resources running out
- Avoiding performance bottlenecks
- Scaling solutions
  - asynchronous 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., the DNS, handling domain names on the Internet)

Challenges for Distribution Transparency (1+)

- Goal: Collection of independent, autonomous actors appear to user as single unified system
  - Hide the distribution

- Different categories of transparency:

Transparencies (RM-ODP standard, 1998)

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 (∗) (the resource does not notice)
Relocation	Hide that a resource may be moved to another location (∗) while in use (the others don’t notice)
Replication	Hide that a resource is replicated
Transaction	Hide that multiple competing users perform concurrent actions on the resource
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

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

Challenges for Distribution Transparency (2)

- Concurrency
  - Many things happening at the same time
- Replications and migration cause additional requirements:
  - Ensure consistency between different replicas and
  - Support distributed decision-making
- Heterogeneity
  - All the differences in hardware, software, etc to account for
- Failure models
  - Things can go wrong in different ways

Handling Concurrency

- Concurrency:
  - Several simultaneous users => integrity of data
    - mutual exclusion
    - synchronization
    - ext: transaction processing in databases
  - 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

Handling 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?

Failure handling: what can go wrong? 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.

What can go wrong? 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

Challenges for Security

- Mostly similar to normal challenges in wide-area networks
  - Sometimes easier, with closed, dedicated systems
- Solution techniques
  - cryptography
  - authentication
  - access control techniques
- Policies
  - access control models
  - information flow models
- Leads to: secure channels, secure processes, controlled access, controlled flows

Challenges from the Environment

- 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 1 millisecond to weeks
  - in space: from 1 mm to thousands of kilometers
  - in dependability: link always there or completely unreliable
- Autonomous independent actors (=> independent failures, too!)

No global clock

Global state information not possible

Challenges from Design Requirements

- Performance requirements
  - responsiveness
  - throughput
  - load sharing, load balancing
  - issue: abstract algorithm vs. actual system behavior
- Quality of service requirements
  - correctness (in nondeterministic environments)
  - reliability, availability, fault tolerance
  - security (again with the security!)
  - performance
  - adaptability

False assumptions everyone makes when developing their first distributed application:

- 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

- By Peter Deutsch (creator of Ghostscript)

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

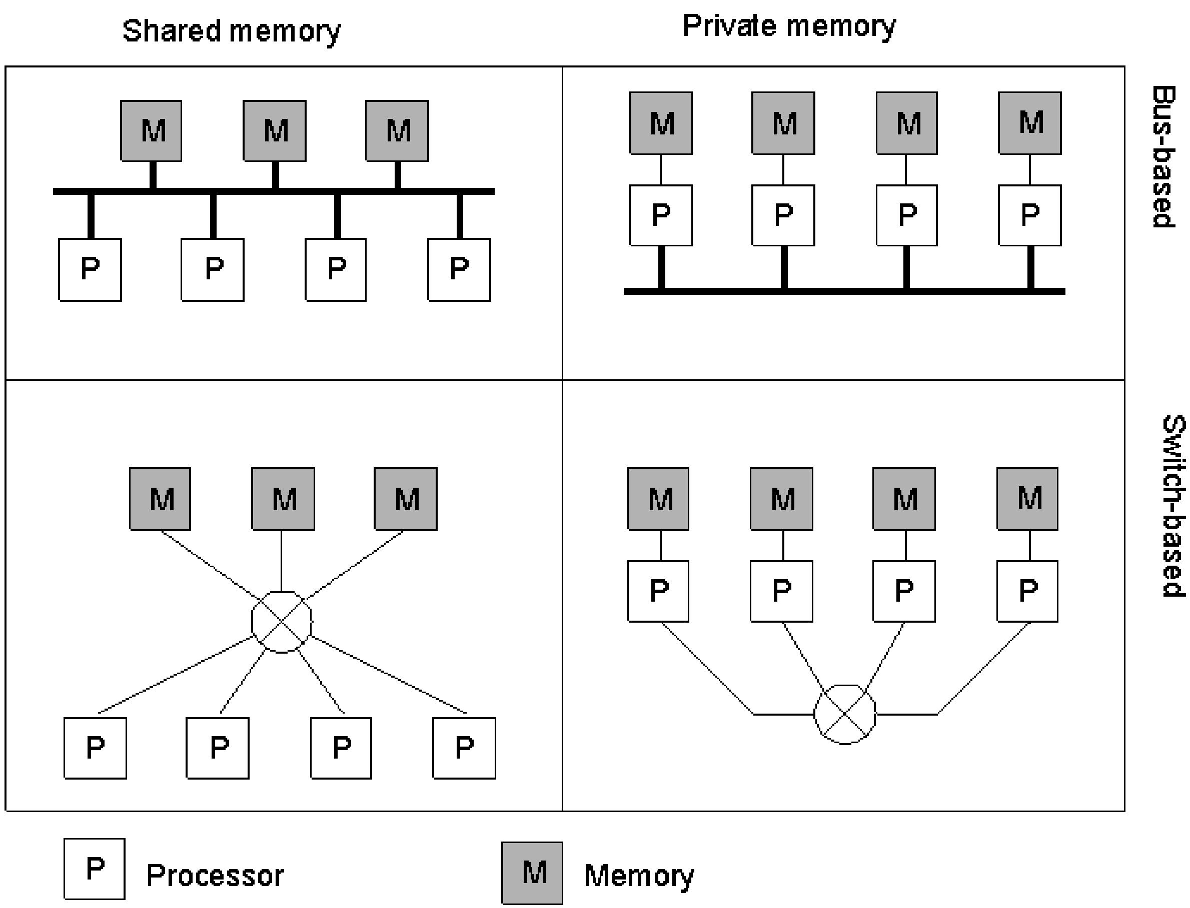
Systems, Architectures and System Architectures

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

Hardware: The Bottom Layer

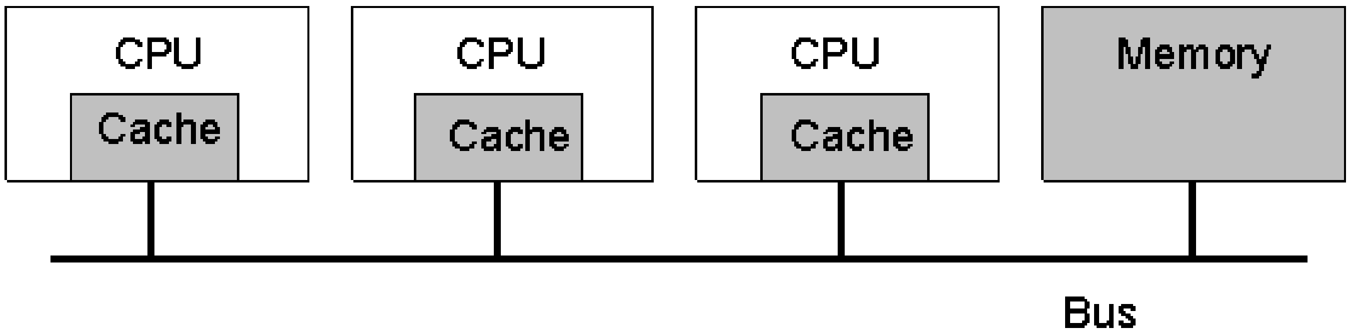
- The behavior of software systems is affected by:
- The platform ....
  - the individual nodes (”computer” / ”processor”)
  - communication between two nodes
  - organization of the system (network of nodes)
- ... and its characteristics
  - capacity of nodes
  - capacity (throughput, delay) of communication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

Basic Organizations of a Node



Different basic organizations and memories in distributed computer systems

A Look at Hardware Level: Multiprocessors



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”)

General Multicomputer Systems

- Hardware setup may be very heterogeneous
- Loosely connected systems
  - Nodes are autonomous
  - Communication is slow and vulnerable
  - => Cooperation at ”service level”
- Application architectures
  - Multiprocessor systems do parallel computation
  - Multicomputer systems form distributed systems

Some concepts for the coming history tour

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

Brief history of distributed systems (1/3)

- 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 – form ”a 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

Brief history of distributed systems (2/3)

- 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
  - Examples: Distributed Computing Environment (DCE) '90s, Microsoft's COM and later .NET framework, OMG's CORBA
- present middlewares for
  - multimedia, realtime computing, telecom
  - ecommerce, adaptive / ubiquitous systems

Brief history of distributed systems (3/3)

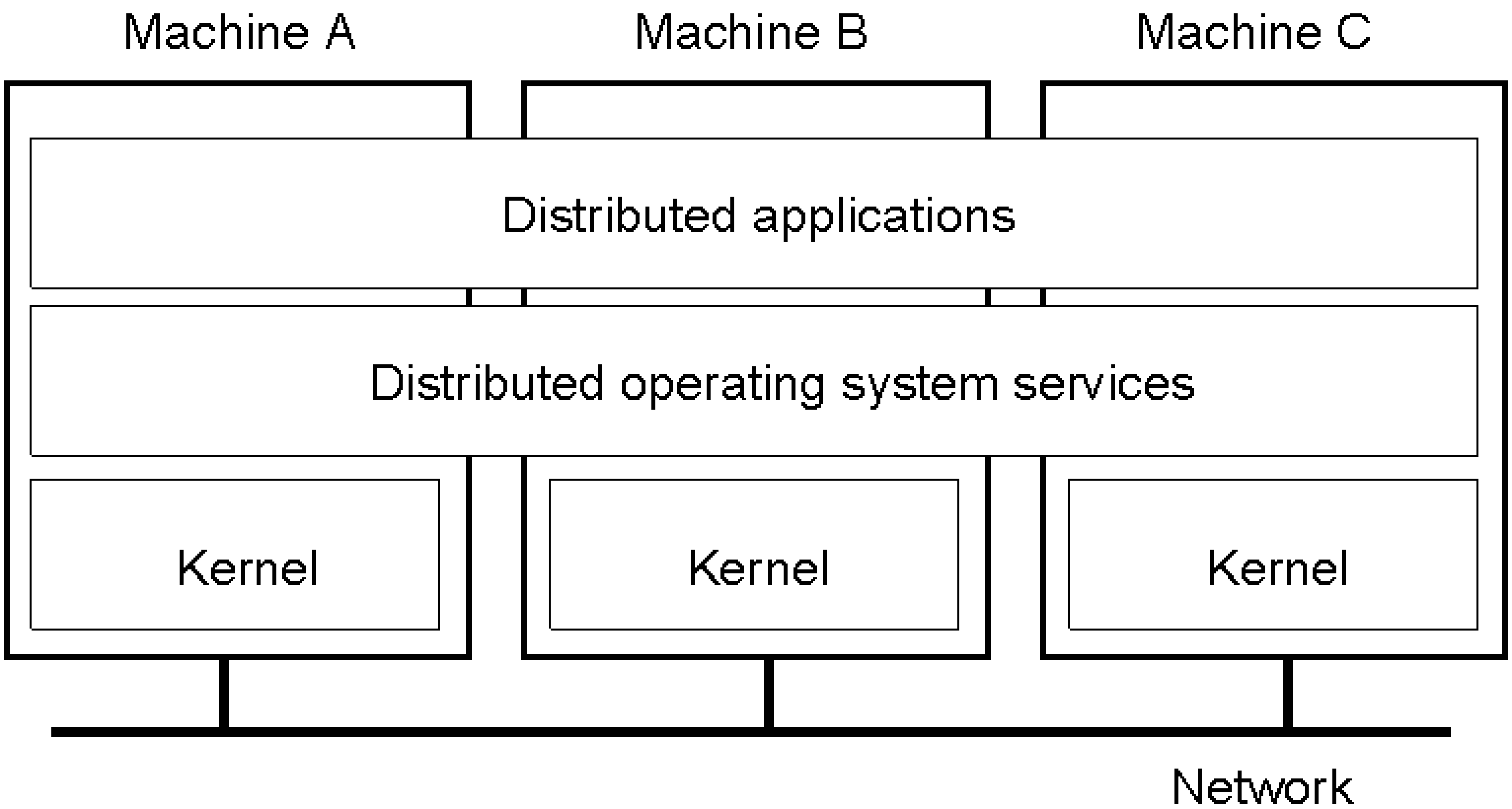
- Late 1990's: distribution middleware well-known
  - Generic, with distributed services
  - Support standard transport protocols and provide standard API
  - Available for multiple hardware, protocol stacks, operating systems
  - Examples: Distributed Computing Environment (DCE) '90s, Microsoft's COM and later .NET framework, OMG's CORBA
- Present middlewares exist for
  - multimedia, realtime computing, telecommunications
  - eCommerce, adaptive / ubiquitous systems

Operating systems and middleware

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

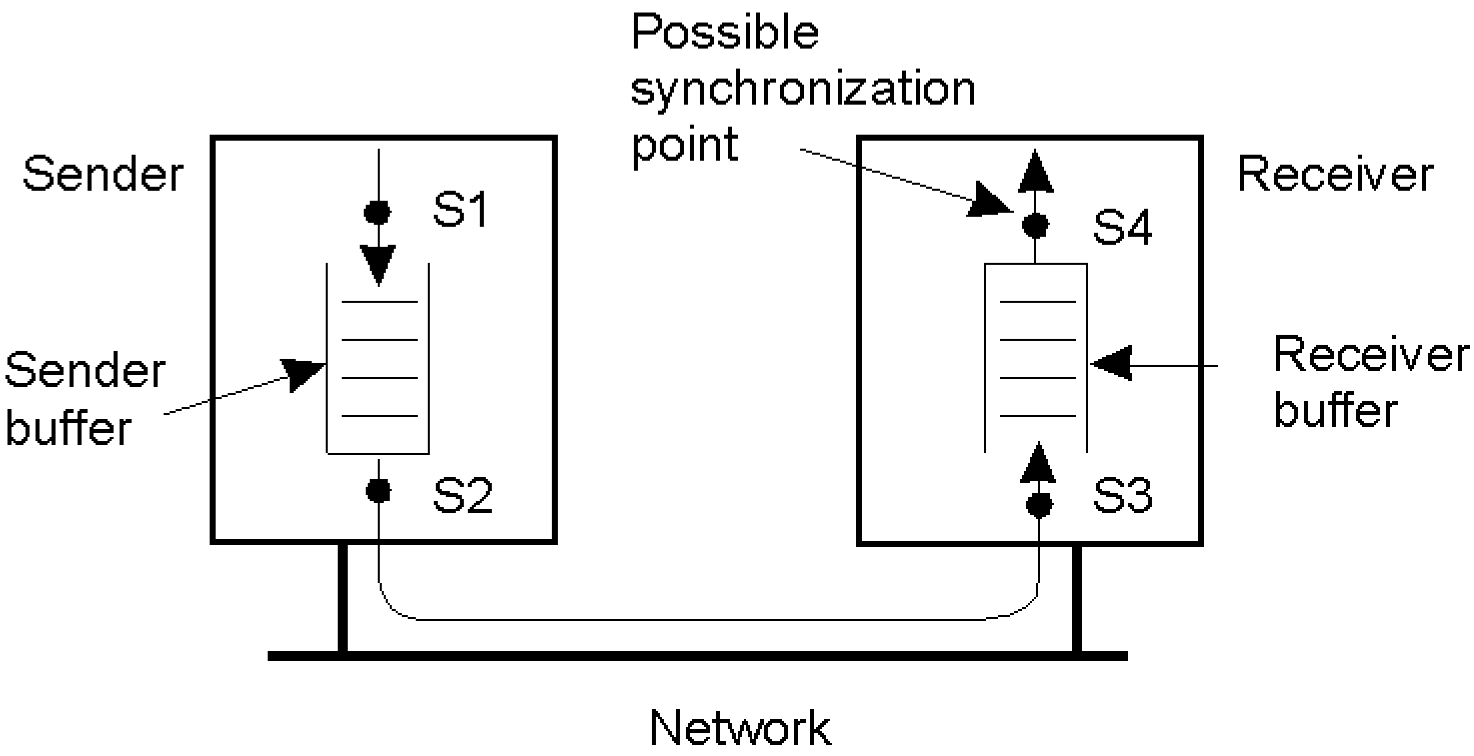
DOS: Distributed OS; NOS: Network OS

Multicomputer Operating Systems (1)



General structure of a multicomputer operating system

Multicomputer Operating Systems (2)



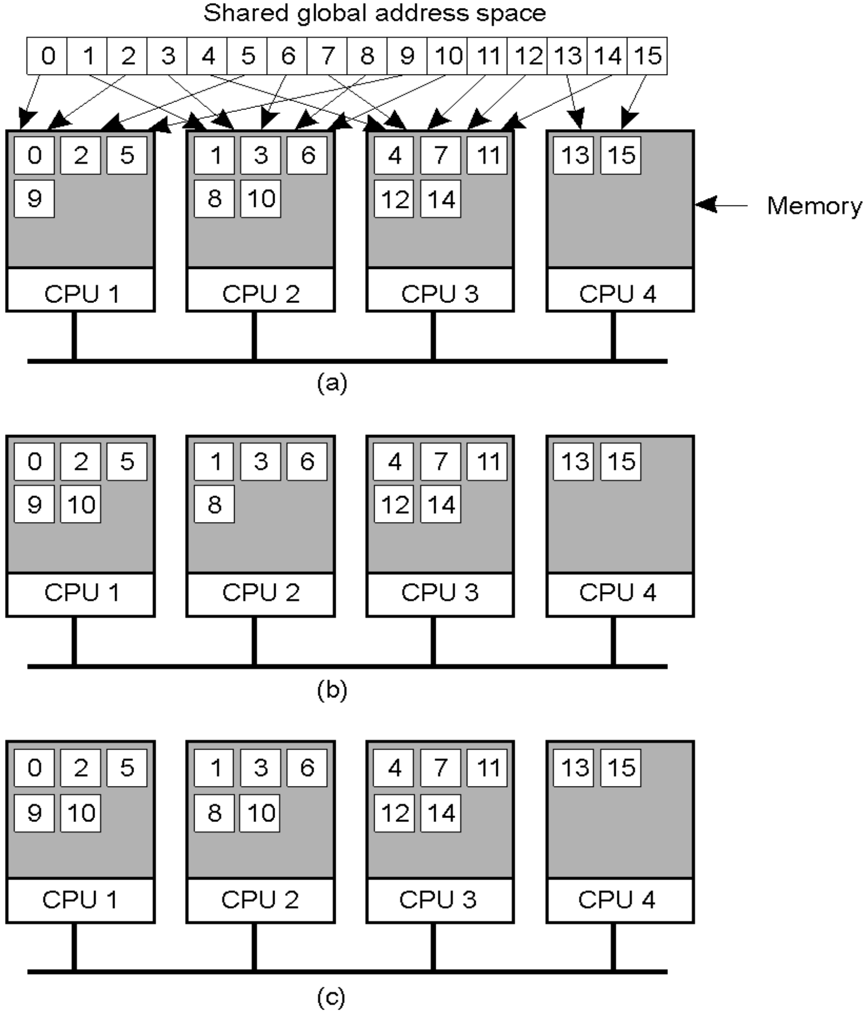
Alternatives for blocking and buffering in message passing.

Distributed Shared Memory Systems (1)

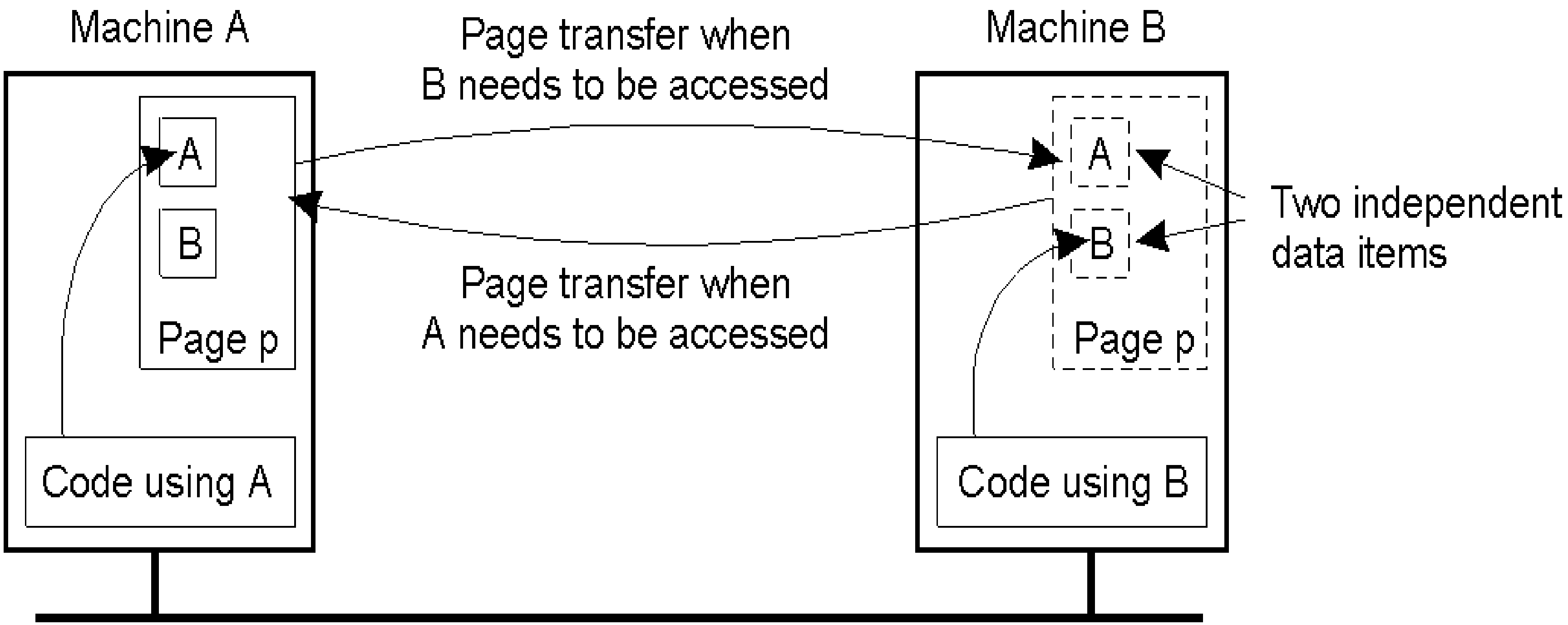
a) Pages of address space distributed among four machines

b) Situation after CPU 1 references page 10

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

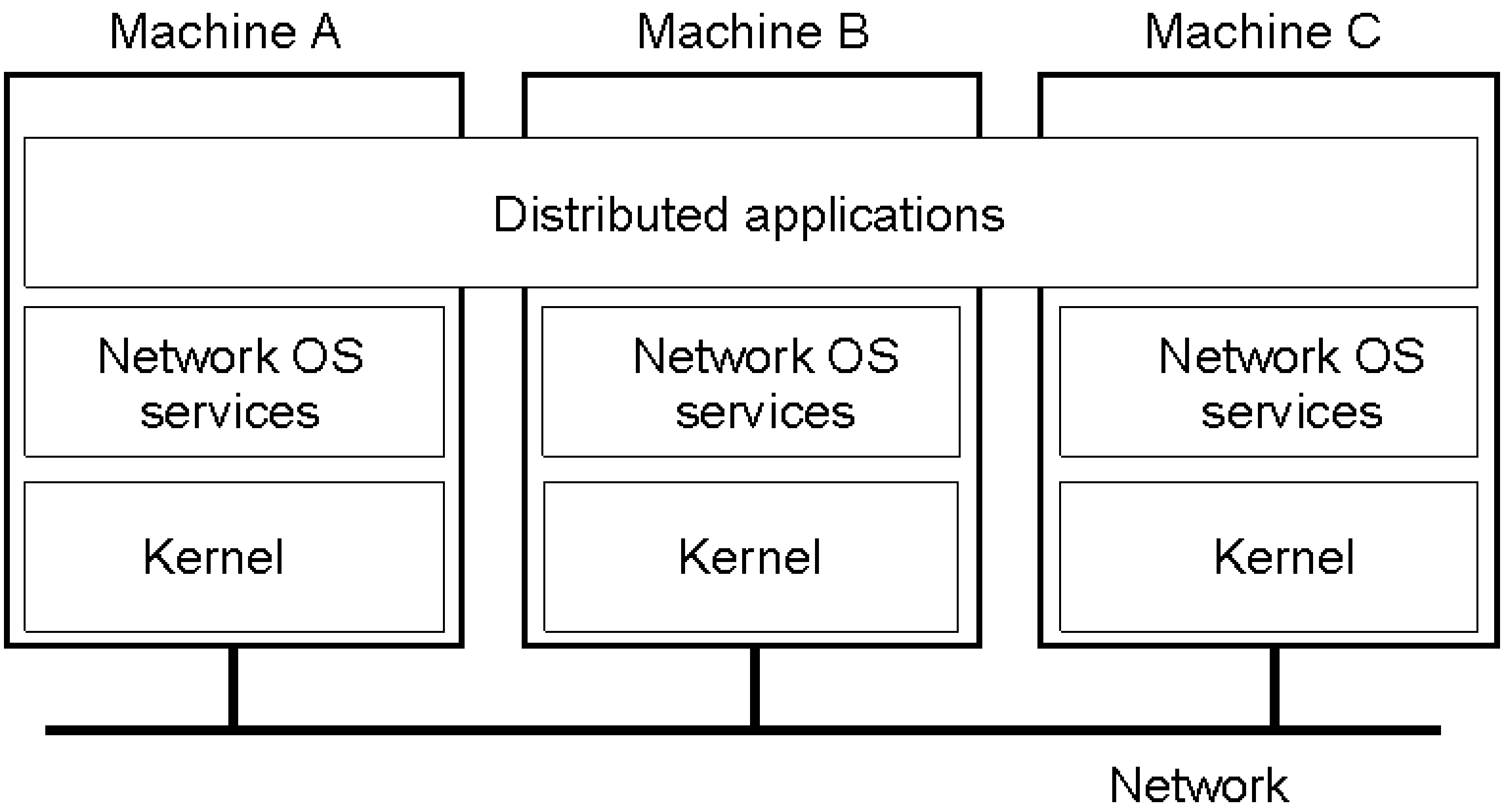


Distributed Shared Memory Systems (2)



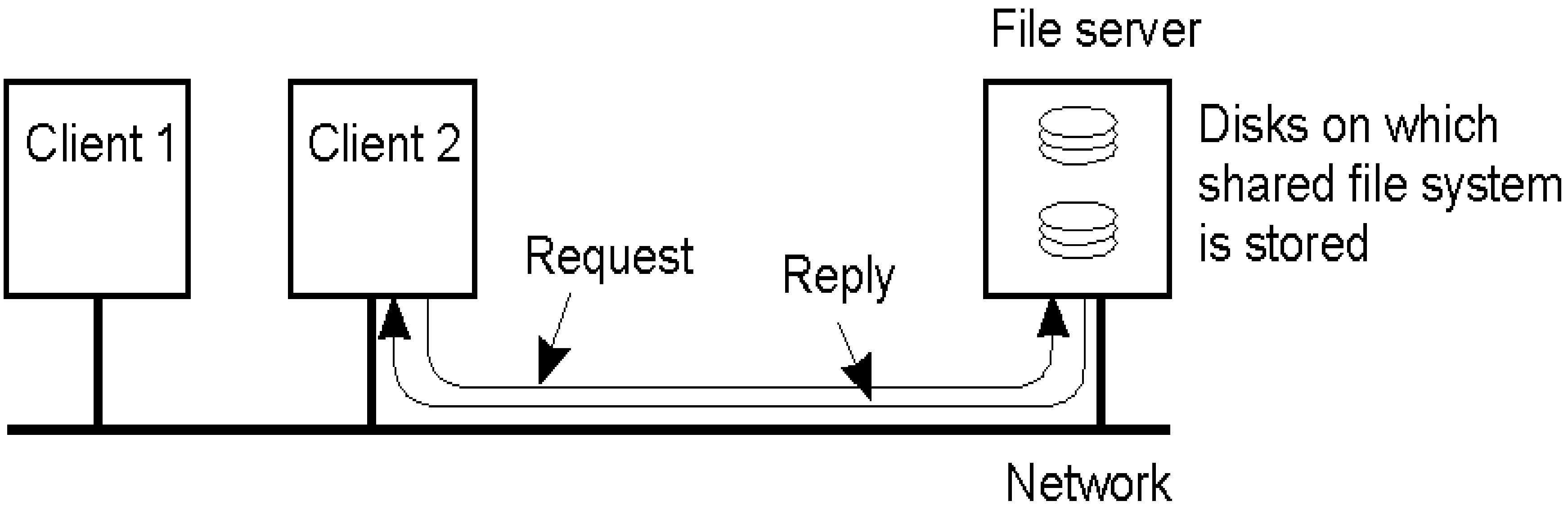
False sharing of a page between two independent processes.

Network Operating System (1)



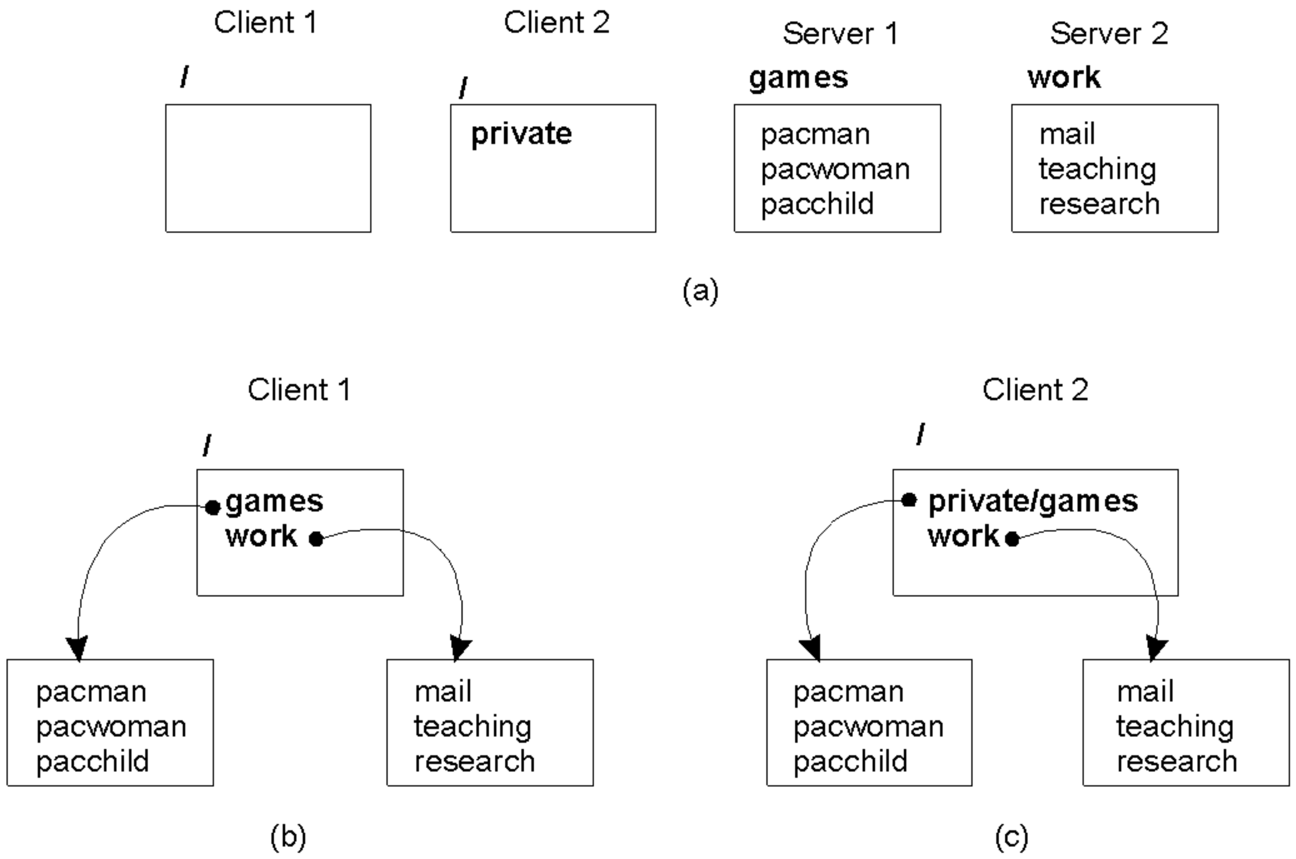
General structure of a network operating system.

Network Operating System (2)



Two clients and a server in a network operating system.

Network Operating System (3)

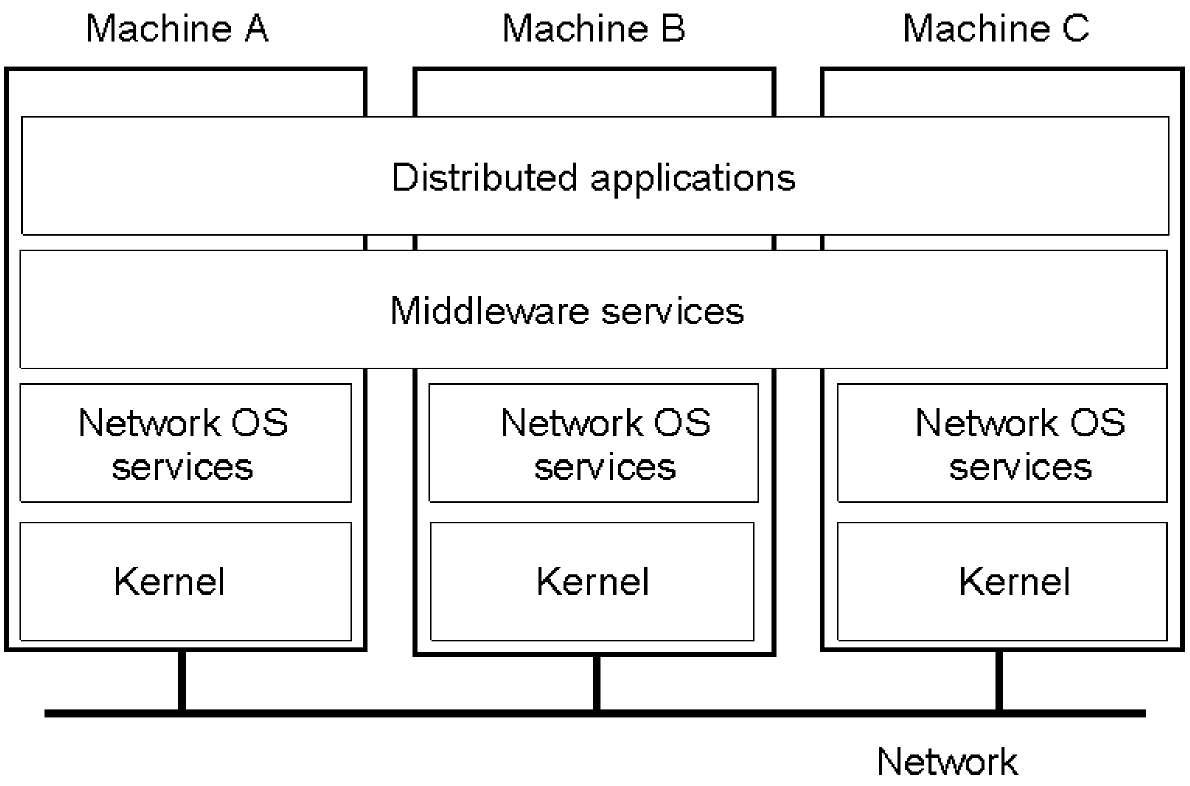


Different clients may mount the servers in different places.

Above the Operating System: 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



General structure of a distributed system as middleware.

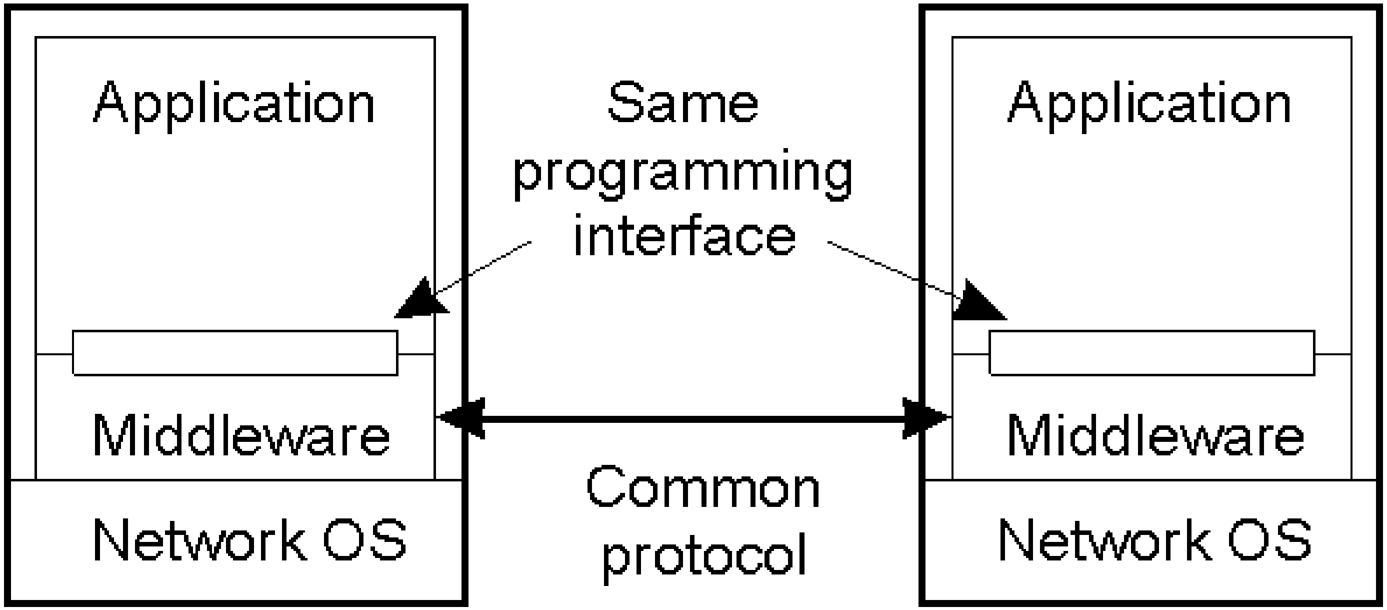
Middleware

- Operations offered by middleware
  - Remote Method Invocation (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 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.

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

Client-server model generalized,

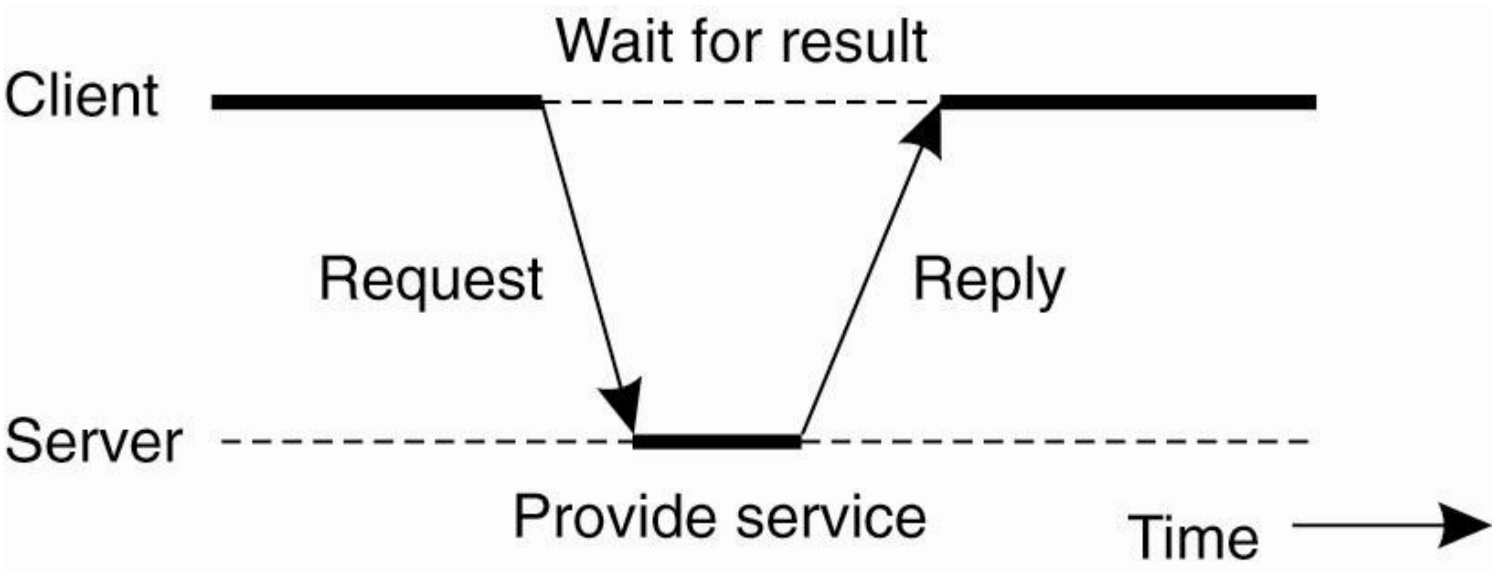
Peek at architectural styles

Architectural Models

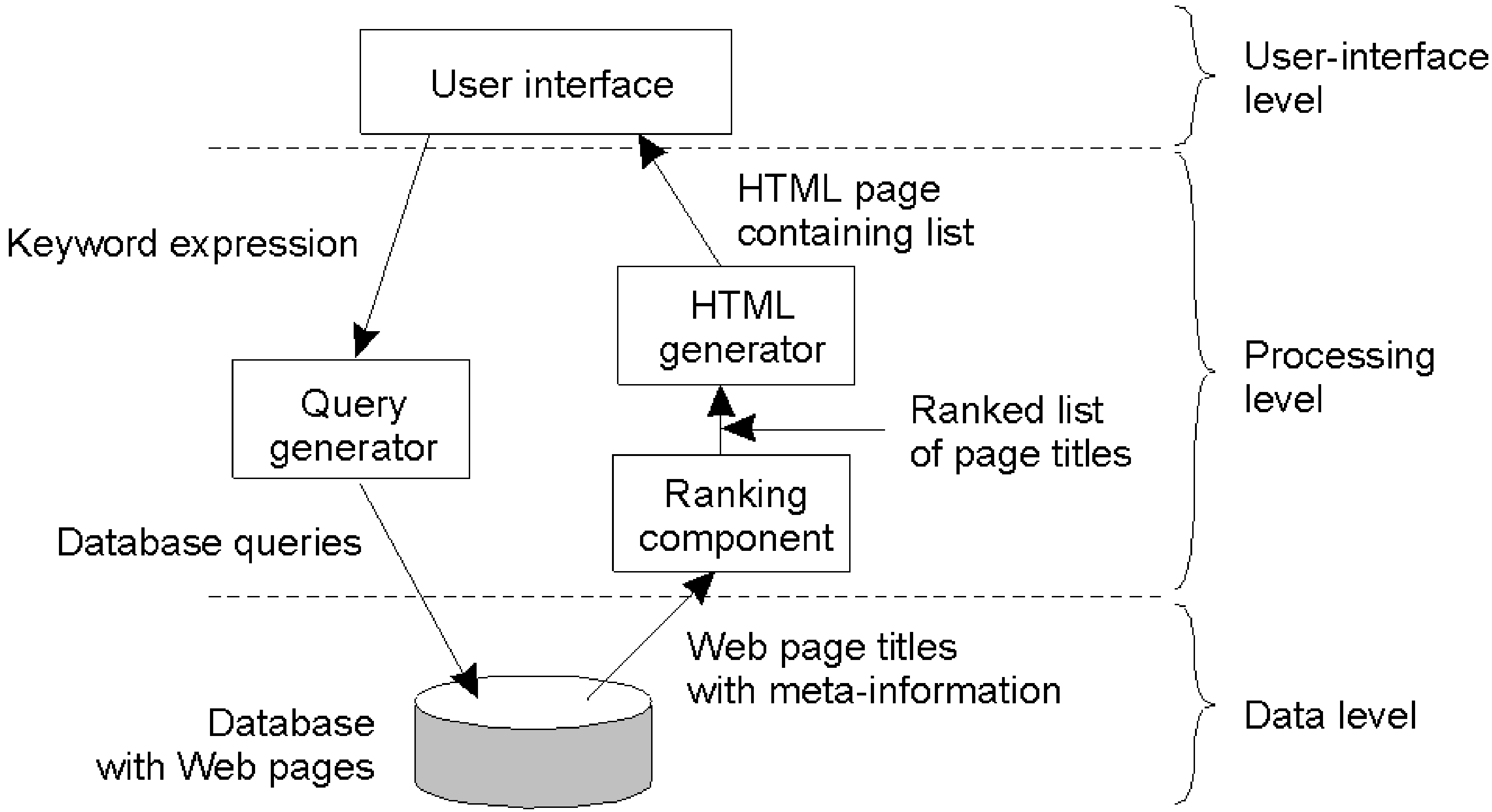
- Architectural models provide a high-level view of the distribution of functionality between system components and the interaction relationships between them
- Architectural models define
  - components
  - communication
- Criteria for architecture design:
  - performance
  - reliability
  - scalability, …

Client-Server Architectures

- General interaction between a client and a server.

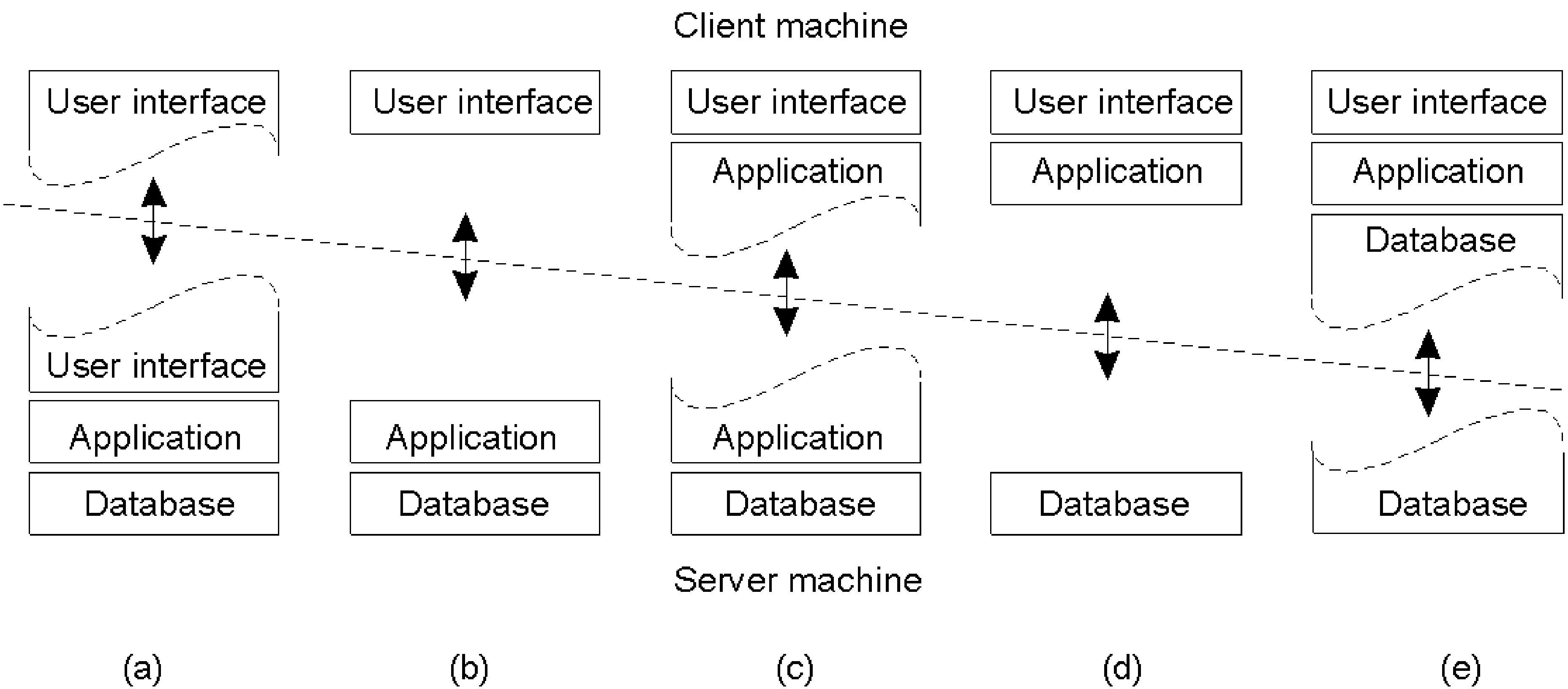


Layered architecture



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

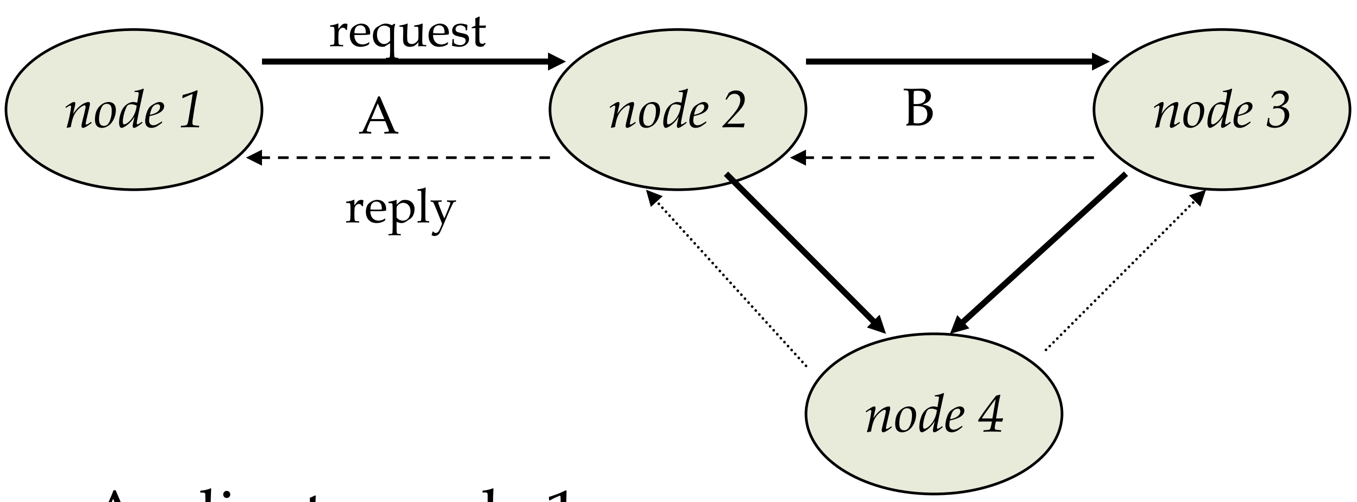
Multitiered Architectures (1)



Alternative client-server organizations.

Multitiered Architectures (2)

Client - server: generalizations

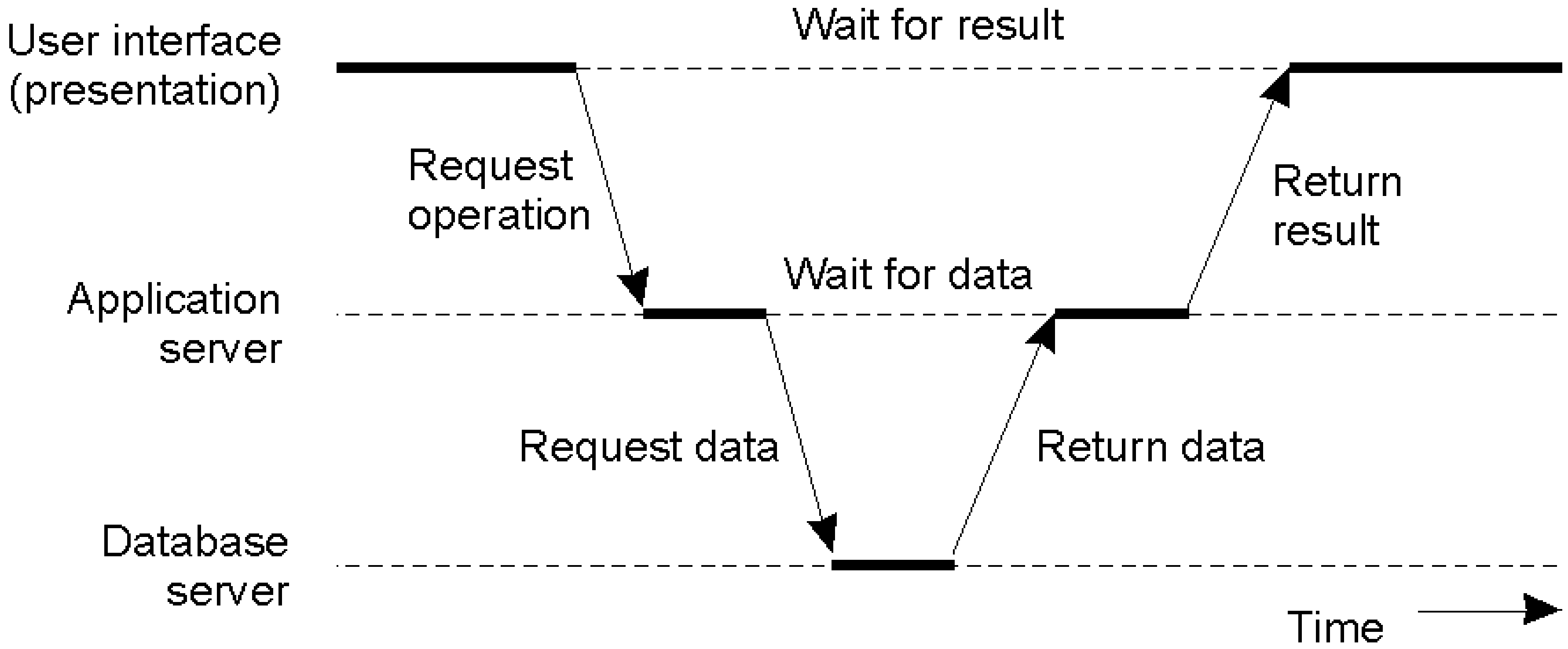


A client: node 1
server: node 2

B client: node 2
server: node 3

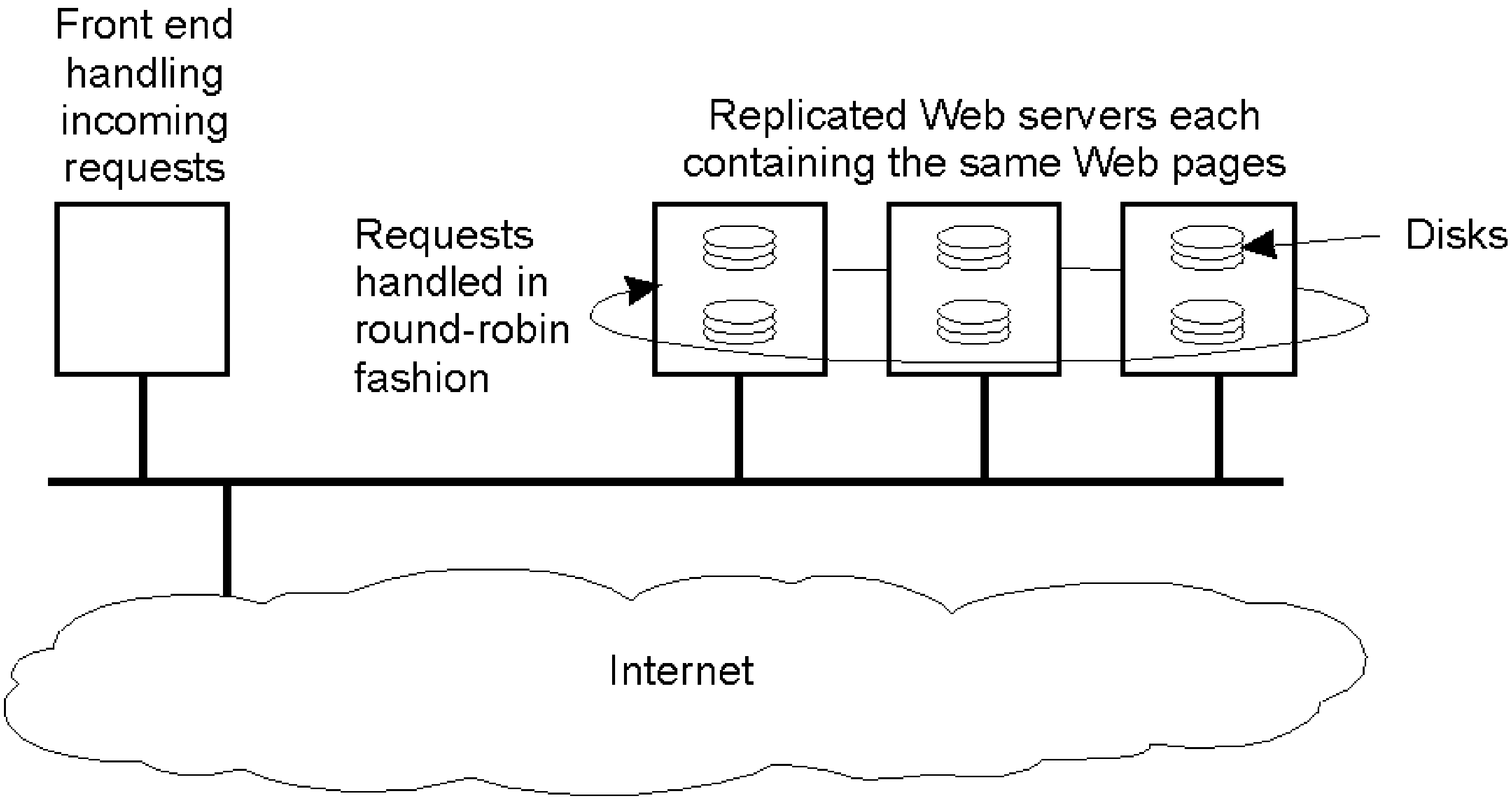
the concept is related to communication not to nodes

Multitiered Architectures (3)



An example of a server acting as a client.

Modern Architectures



An example of horizontal distribution of a Web service.

Chapter Summary

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