

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

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

Fall 2013





Course Goals and Content

- Distributed systems and their:
 - Basic concepts
 - Main issues, problems, and solutions
 - Structured and functionality
- Content:
 - Distributed systems
 - Architectures, goal, challenges
 - Where our solutions are applicable
 - Synchronization: Time, coordination, decision making
 - Replicas and consistency
 - Fault tolerance
 - Large-scale distributed systems in real world



Course Material

- Tanenbaum, van Steen: Distributed Systems, Principles and Paradigms; Pearson Prentice Hall 2007
 - 2002 edition also ok
 - Coulouris, Dollimore, Kindberg: Distributed Systems, Concepts and Design; Addison-Wesley 2005
 - Selected content from Barroso L. A. and Hölzle U.: The Datacenter as a Computer (online, see course website)
- Lecture slides on course website
 - NOT sufficient by themselves
 - Help to see what parts in book are most relevant
 - On some topics, slides cover more material than the book



Course Exams

- Normal way (recommended)
 - Exercises, home exercises, course exam
- Grading:
 - Exam 42 points, date December 11, 2013
 - Exercises 12 points (6 exercises, scaled to 0—12)
 - Home exercises 12 points (4 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

- Bi-weekly exercises:
 - Smaller assignments
 - Exception to schedule: 29.11. has exercise, 6.12. does not
 - Complete beforehand, present your work at exercise sessions to gain points
 - Must make an effort to get points!

- Home exercises
 - 4 larger exercises
 - 2 writing exercises
 - 2 programming exercises
 - Due dates will be announced later, deadlines fixed!



People

■ Jussi Kangasharju

■ Lectures:

- Period I: Mon 10-12 and Thu 10-12 in C222
- Period II: Mon 10-12 and Thu 10-12 in D122
- 12 lectures, usually on Mondays
- Thursday sessions workshops/discussion/help/...

■ Exercise group: Fri 12-14 in B222 (bi-weekly)

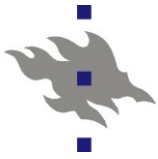
■ Office hour: Mon 13-14 or ask for appointment by email

■ Ossi Karkulahti

■ Exercise group: Fri 12-14 in B222 (bi-weekly)

■ Home exercises

■ Office hour: During exercises or ask appointment by email



Lectures vs. Workshops

- Lectures on Mondays present new material
 - There can and should be discussion as well 😊

- Workshops on Thursdays are for discussion and help
 - **First three workshops** mainly tutorials for new students
 - Later workshops discussion about exercises
 - Every week has some scheduled program
 - Program posted on website in advance



Questions?



Chapter Outline

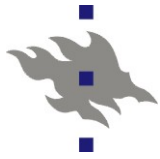
- Defining distributed system
- Examples of distributed systems
- Why distribution?
- Goals and challenges of distributed systems
- Where is the borderline between a computer and a distributed system?
- Examples of distributed architectures



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.



Where Does the Definition Leave Us?

■ Which of the following are distributed systems?

Multi-core processor

Parallel systems

Multi-processor computer

Internet

Computing cluster

Corporate intranet

One data center

Local Area Network

Web

Network of
data centers



What About the Following?

Airplane

Car

My home theater setup

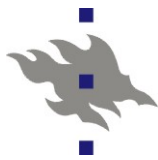
Ticket reservation system

Nuclear power plant

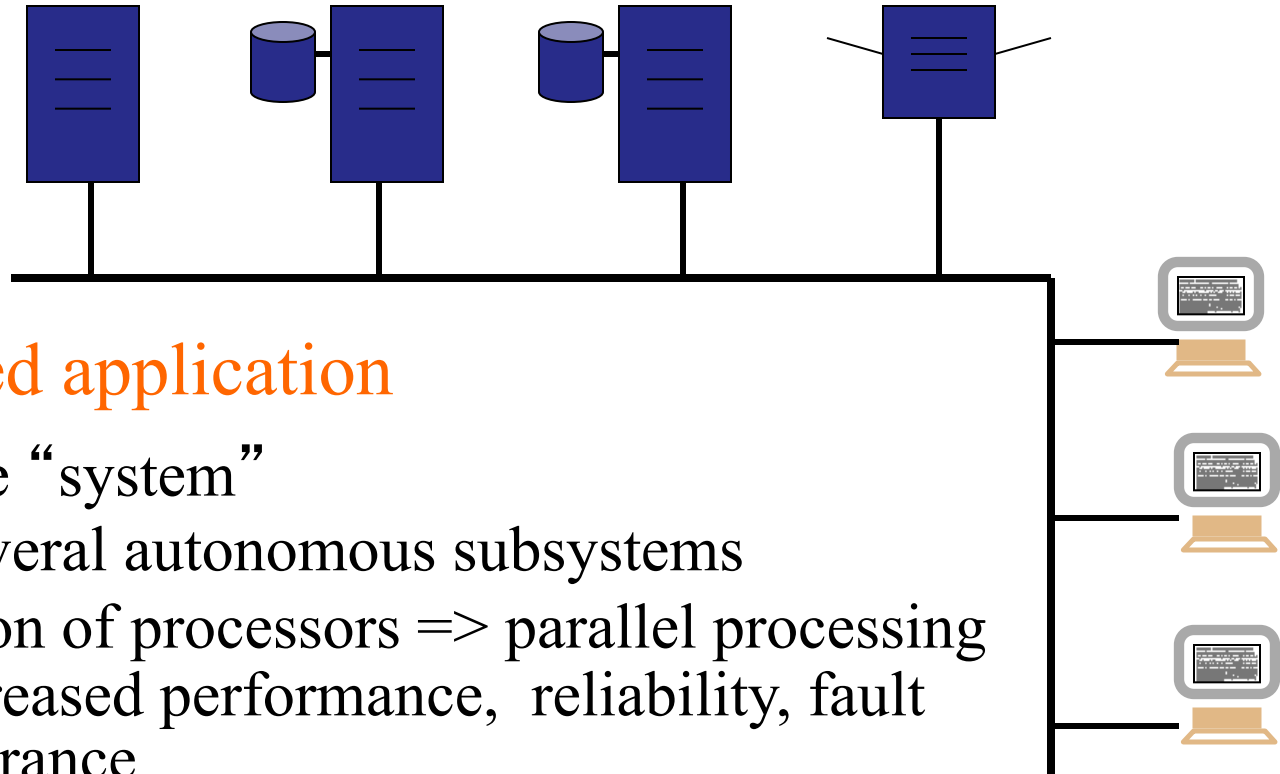
Mobile phone

Factory

My office



Examples of Distributed Systems

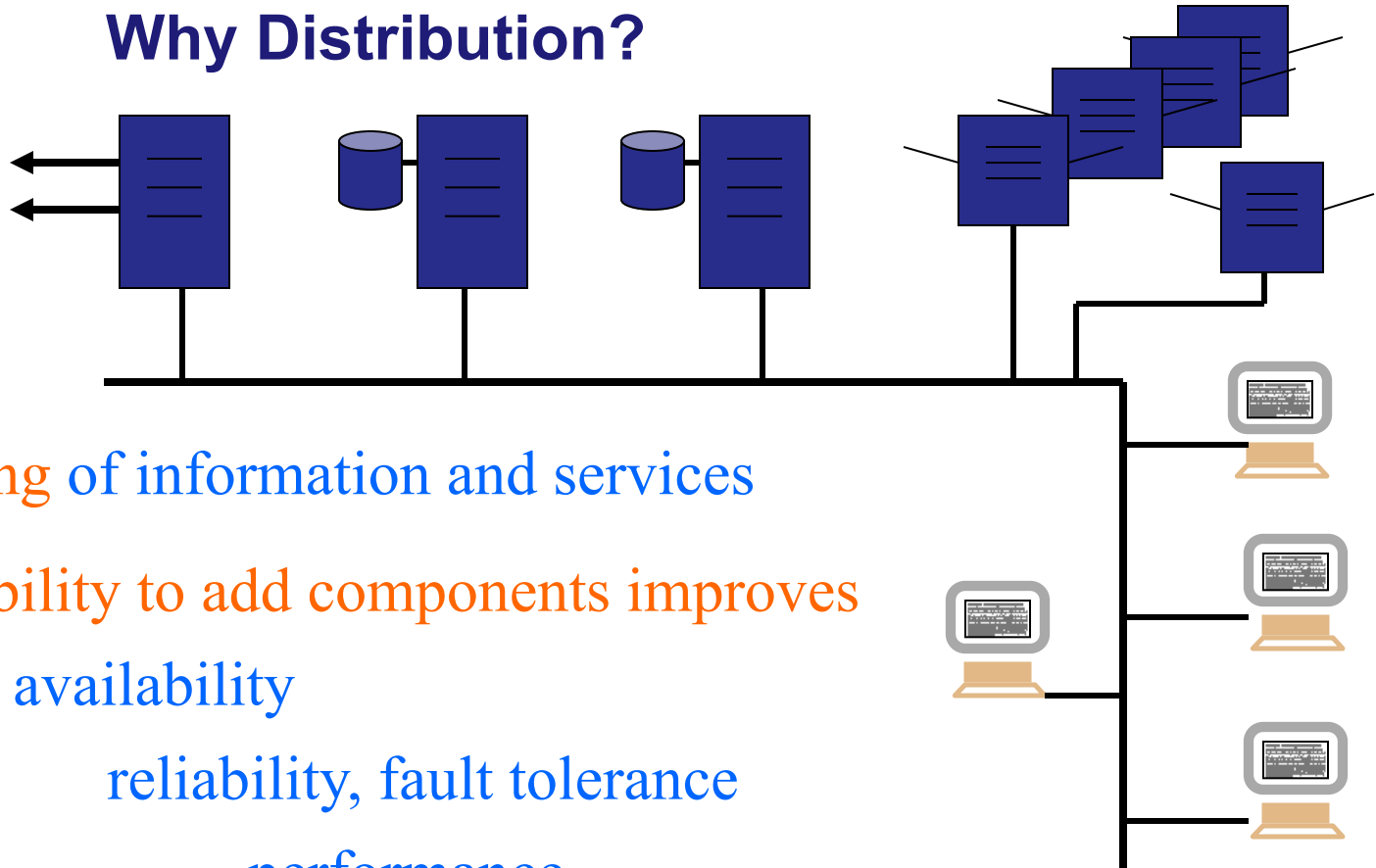


Distributed application

- 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



Why Distribution?



Sharing of information and services

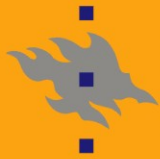
Possibility to add components improves
availability

reliability, fault tolerance

performance

Leads to scalability

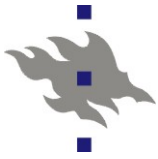
And a large set of gotchas...



HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

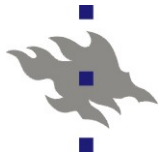
Goals and challenges for distributed systems





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

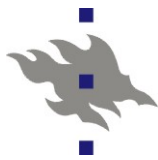
Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located (*)
Migration	Hide that a resource may move to another location (*) (the resource does not notice)
Relocation	Hide that a resource may be moved to another location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

(*) Note 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



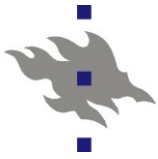
Omission and arbitrary failures

<i>Class of failure</i>	<i>Affects</i>	<i>Description</i>
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 <i>send</i> , 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.



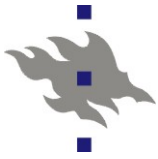
Timing failures

<i>Class of Failure</i>	<i>Affects</i>	<i>Description</i>
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
- New: warehouse-scale computing (we return to this)



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



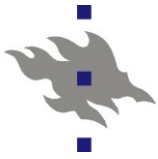
Challenges: Scalability (cont.)

- 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
 - 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., DNS)



Challenges for Security

- Mostly similar to normal challenges in wide-area networks
 - Sometimes easier (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

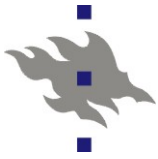


Environment challenges

- 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 msec's 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



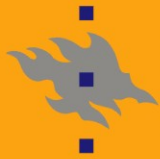
Challenges: Design Requirements

■ 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

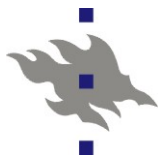
**Where is the borderline between a
computer and distributed system?**



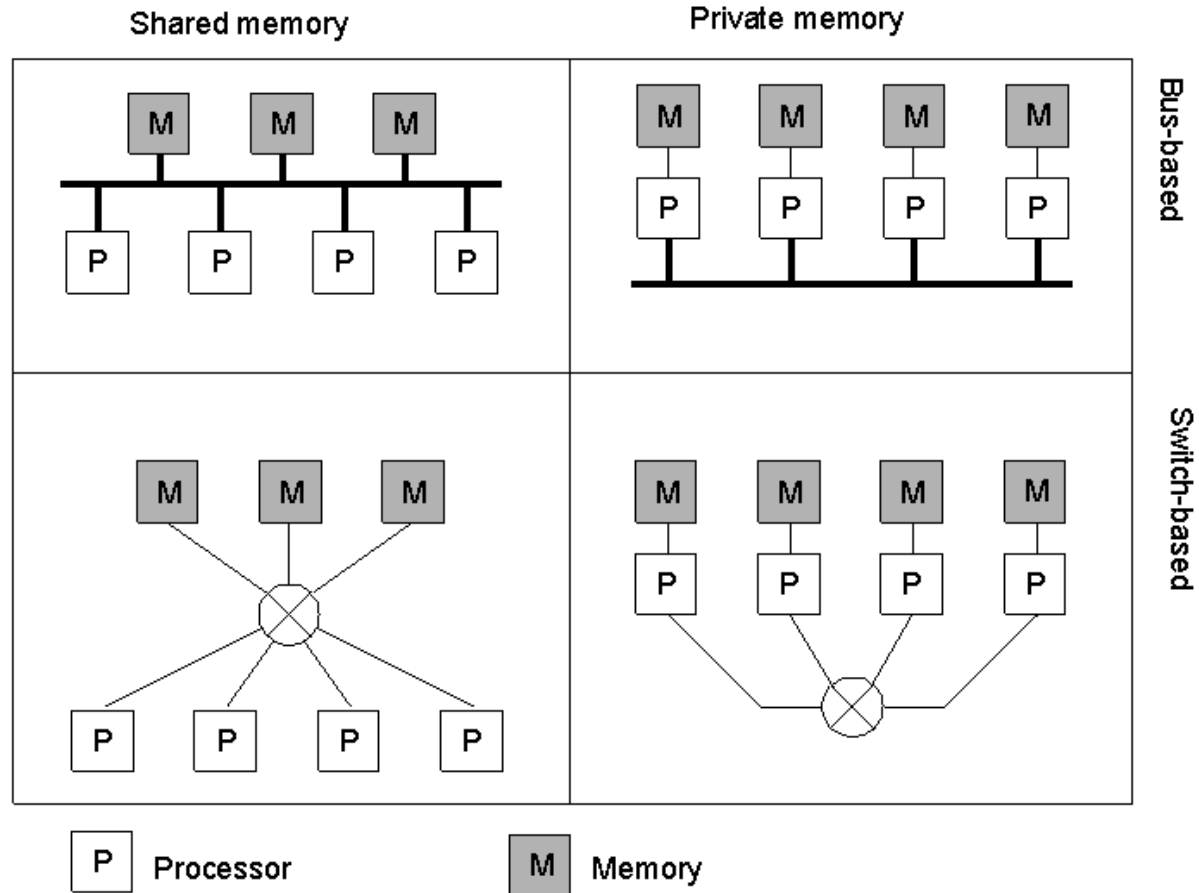


Hardware Concepts

- 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



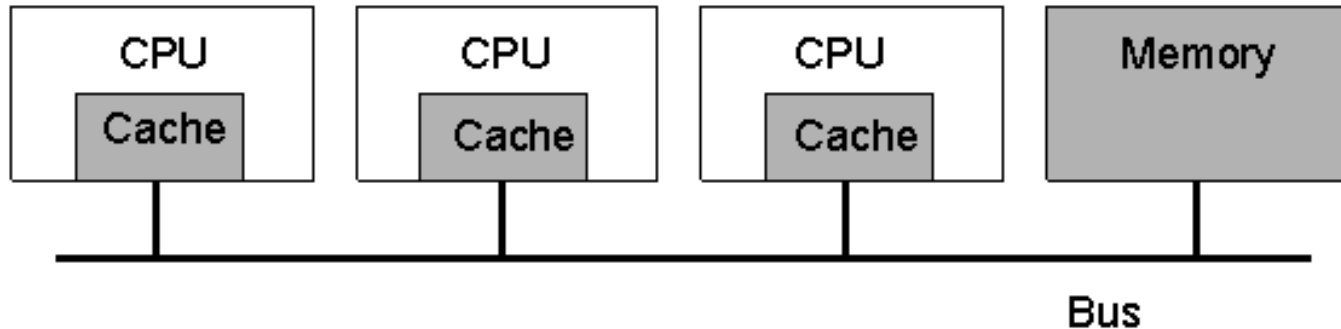
Basic Organizations of a Node



Different basic organizations and memories in distributed computer systems



Multiprocessors (1)



A bus-based multiprocessor.

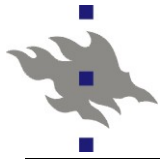
Essential characteristics for software design

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



General Multicomputer Systems

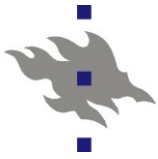
- Hardware: Possibly very heterogeneous
- Loosely connected systems
 - nodes: autonomous
 - communication: slow and vulnerable
 - => cooperation at "service level"
- Application architectures
 - multiprocessor systems: parallel computation
 - multicomputer systems: distributed systems
 - (how are parallel, concurrent, and distributed systems different?)



Software Concepts

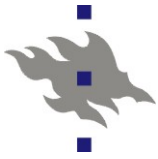
System	Description	Main Goal
DOS	Tightly-coupled operating system for multiprocessors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middle-ware	Additional layer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



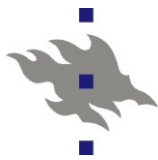
History of distributed systems

- 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

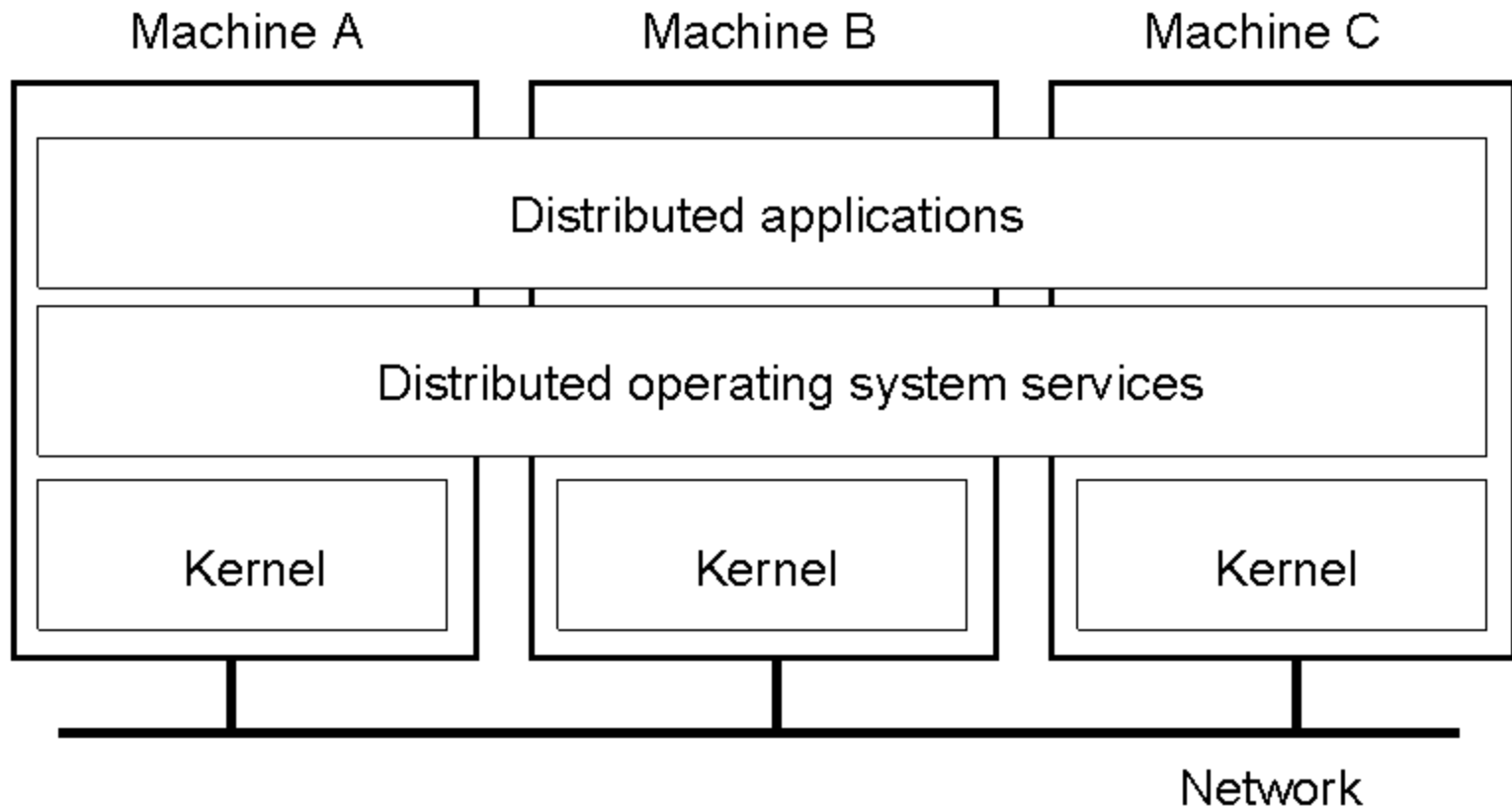


Misconceptions tackled

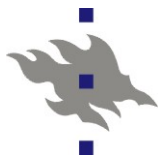
- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero
- Bandwidth is infinite
- Transport cost is zero
- There is one administrator
- There is inherent, shared knowledge



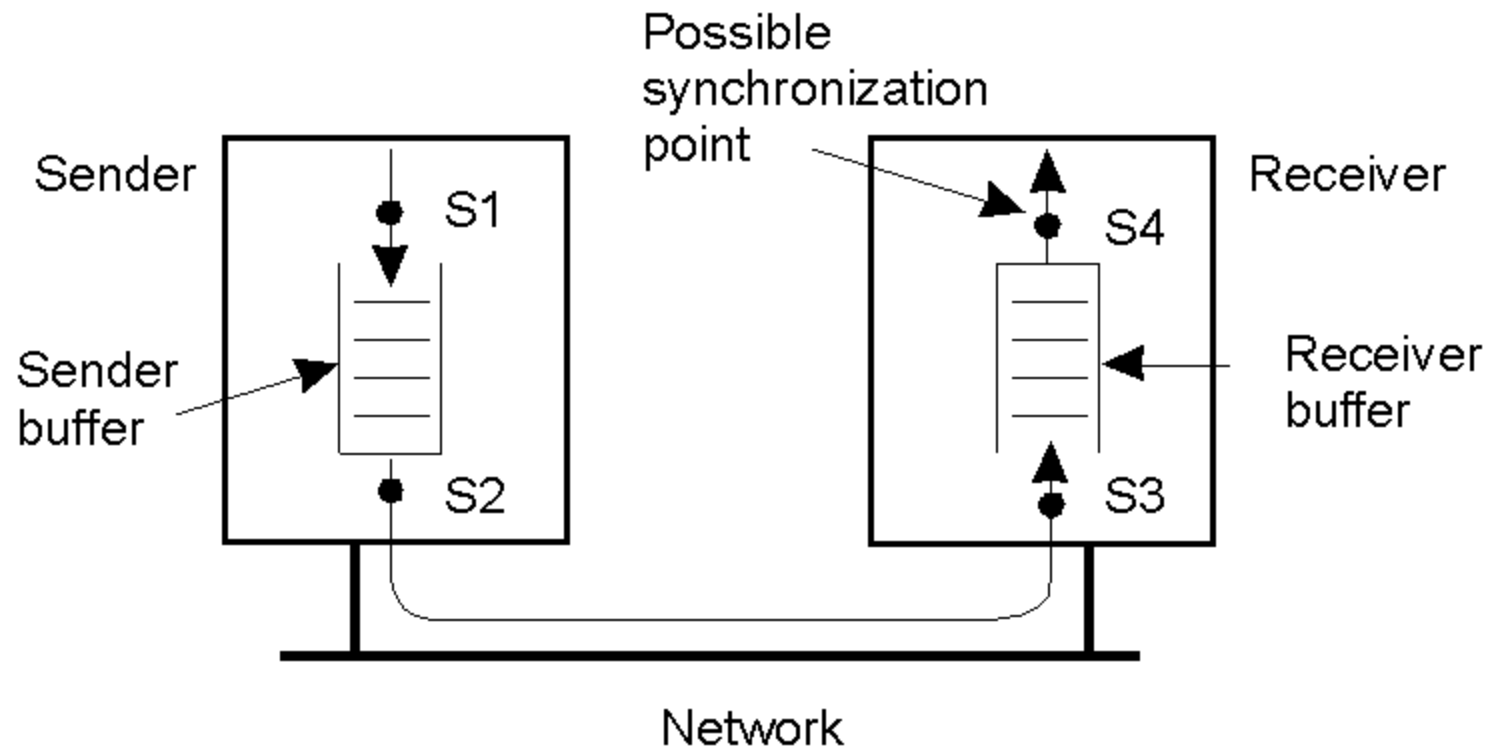
Multicomputer Operating Systems (1)



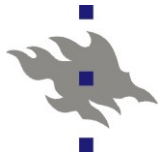
General structure of a multicomputer operating system



Multicomputer Operating Systems (2)

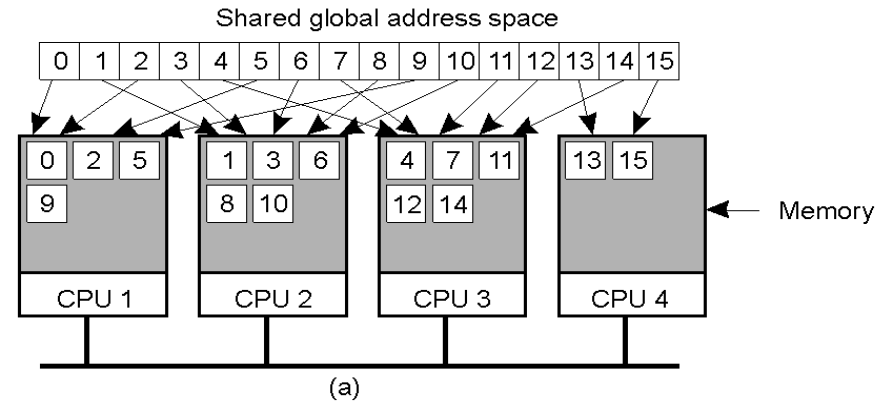


Alternatives for blocking and buffering in message passing.

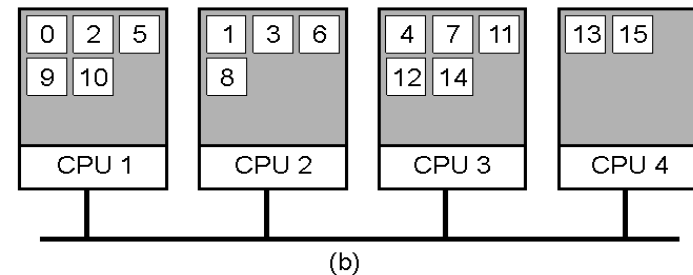


Distributed Shared Memory Systems (1)

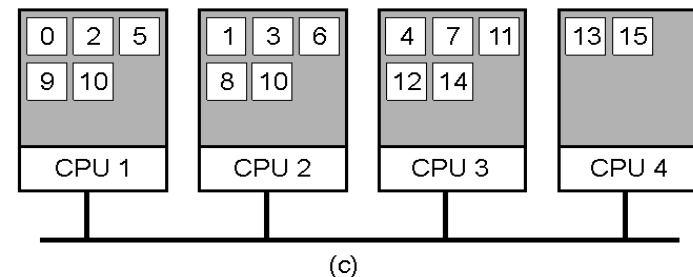
a) Pages of address space distributed among four machines

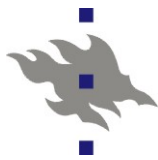


b) Situation after CPU 1 references page 10

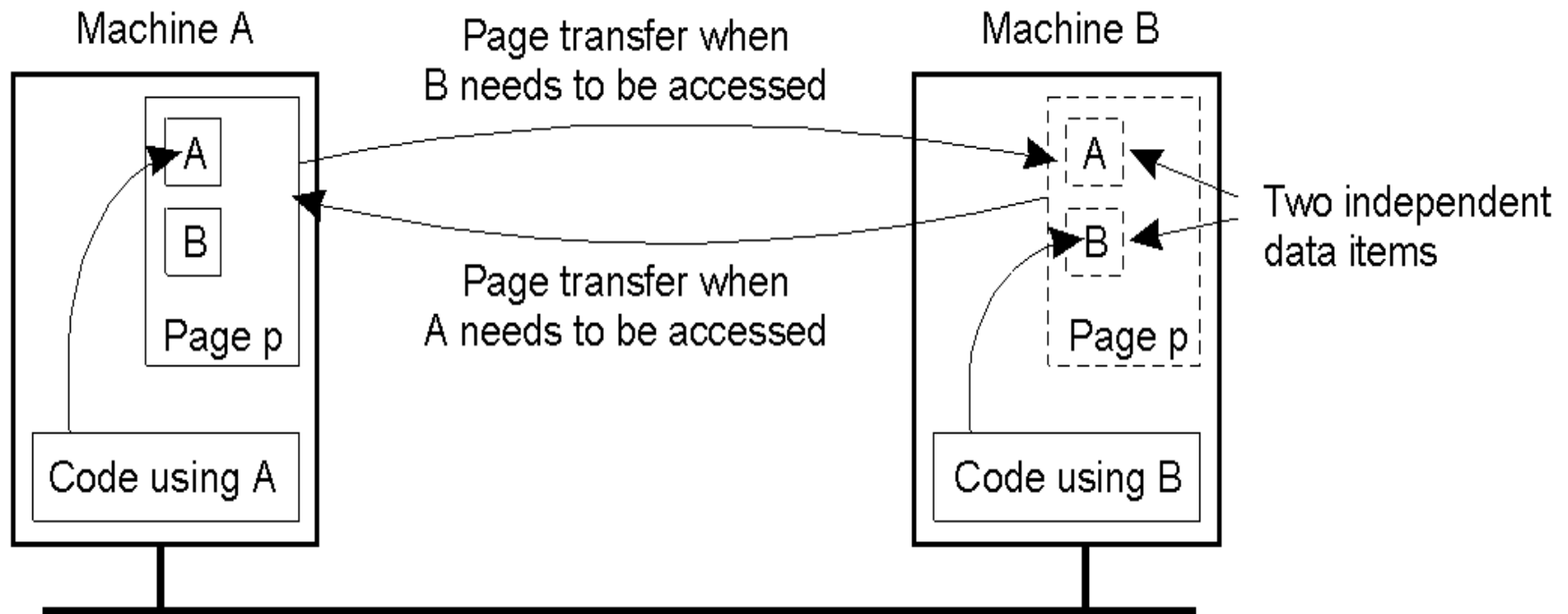


c) Situation if page 10 is read only and replication is used

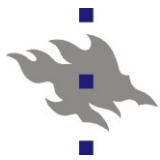




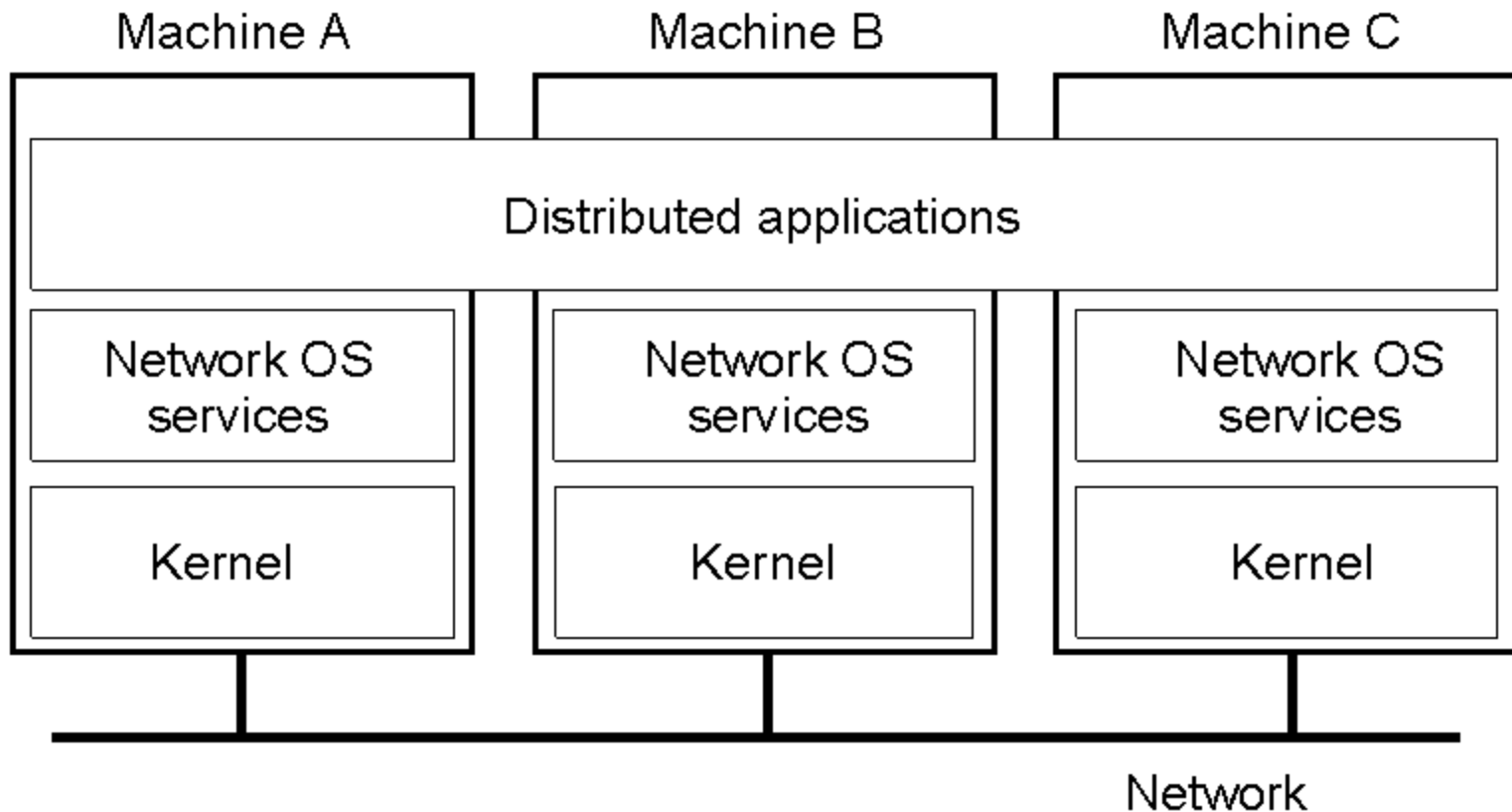
Distributed Shared Memory Systems (2)



False sharing of a page between two independent processes.



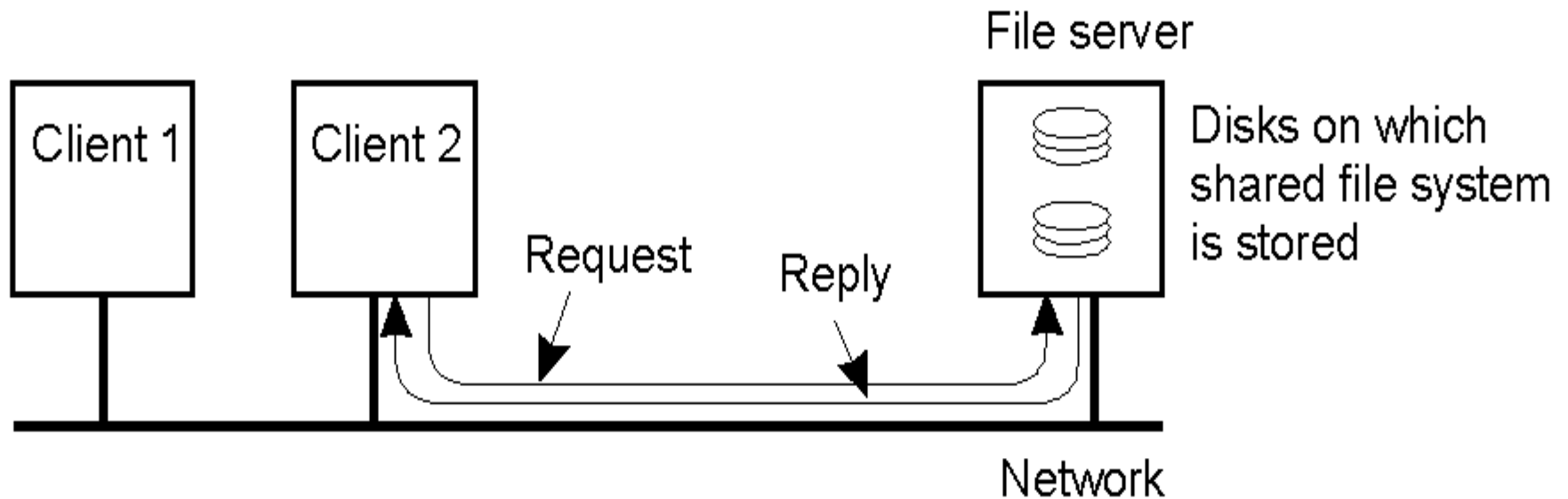
Network Operating System (1)



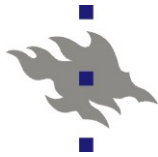
General structure of a network operating system.



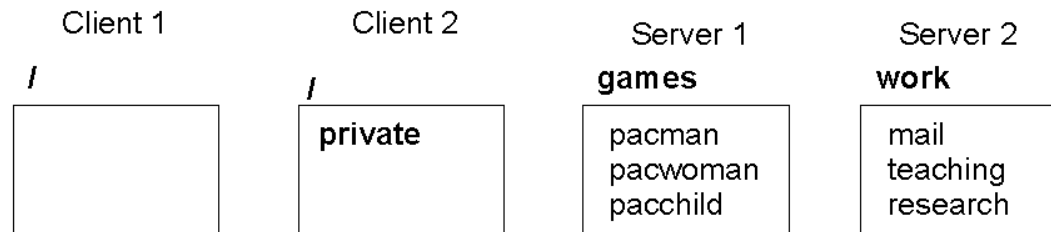
Network Operating System (2)



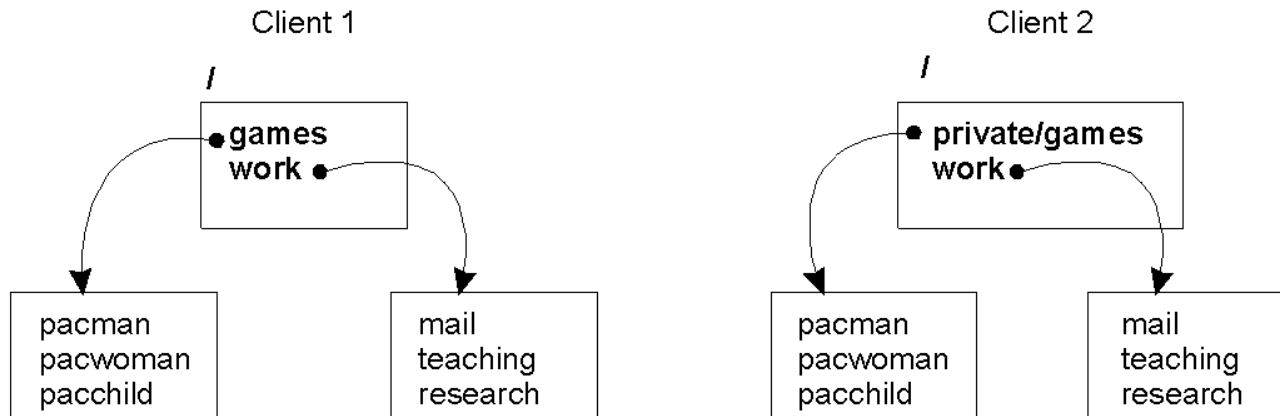
Two clients and a server in a network operating system.



Network Operating System (3)



(a)



(b)

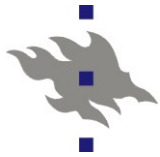
(c)

Different clients may mount the servers in different places.

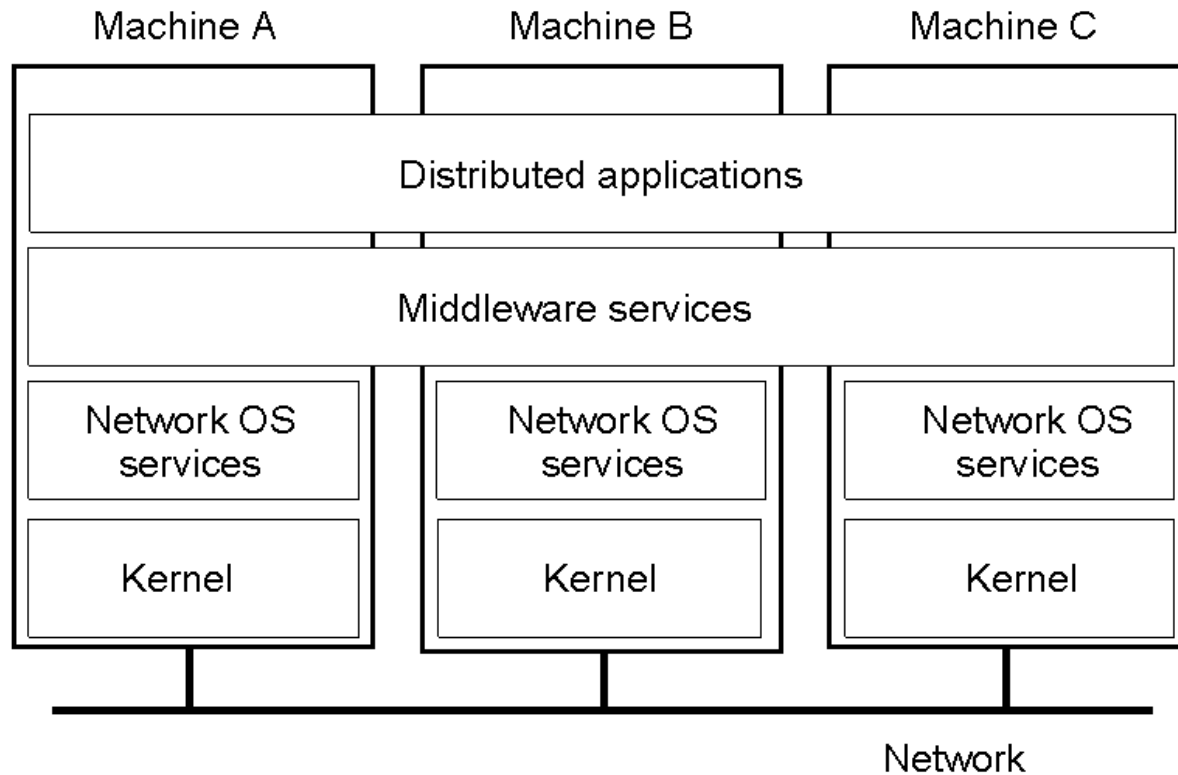


Software Layers

- 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



General structure of a distributed system as middleware.

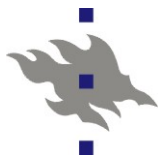


Middleware

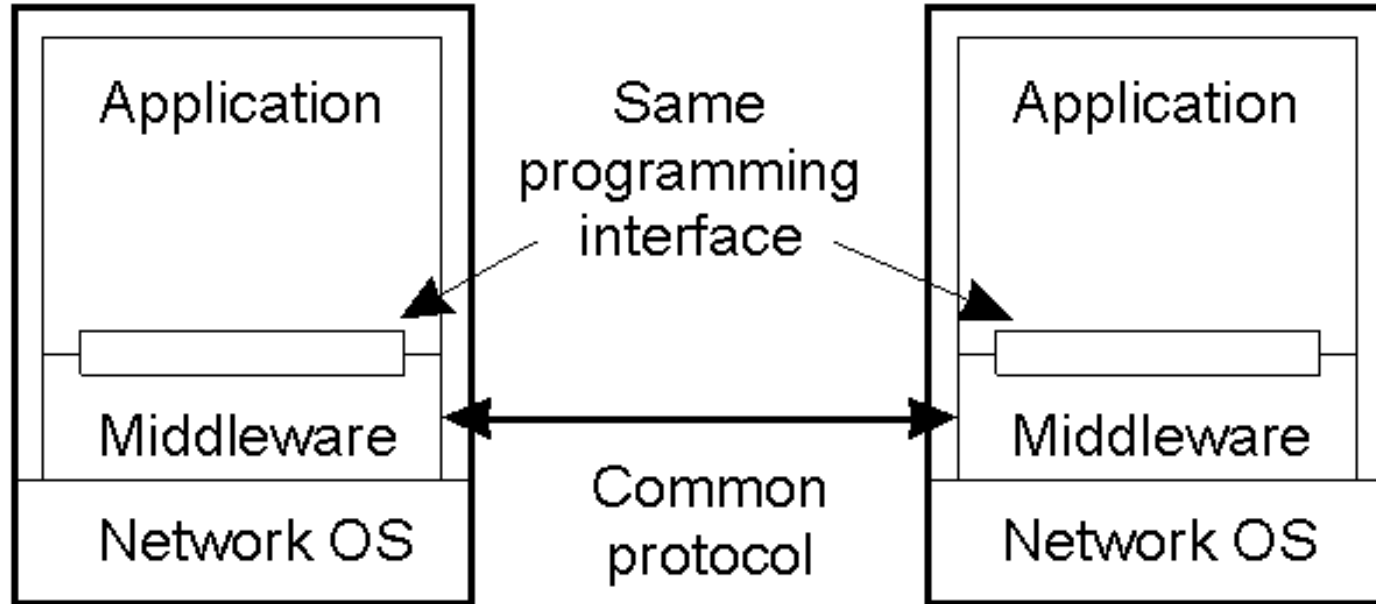
- Operations offered by middleware
 - RMI, group communication, notification, replication, ... (Sun RPC, CORBA, Java RMI, Microsoft DCOM, ...)
- Services offered by middleware
 - naming, security, transactions, persistent storage, ...
- Limitations
 - ignorance of special application-level requirements

End-to-end argument:

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



Middleware and Openness

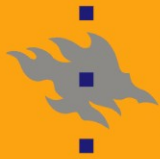


In an open middleware-based distributed system, the protocols used by each middleware layer should be the same, as well as the interfaces they offer to applications.



Comparison between Systems

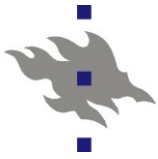
Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global,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





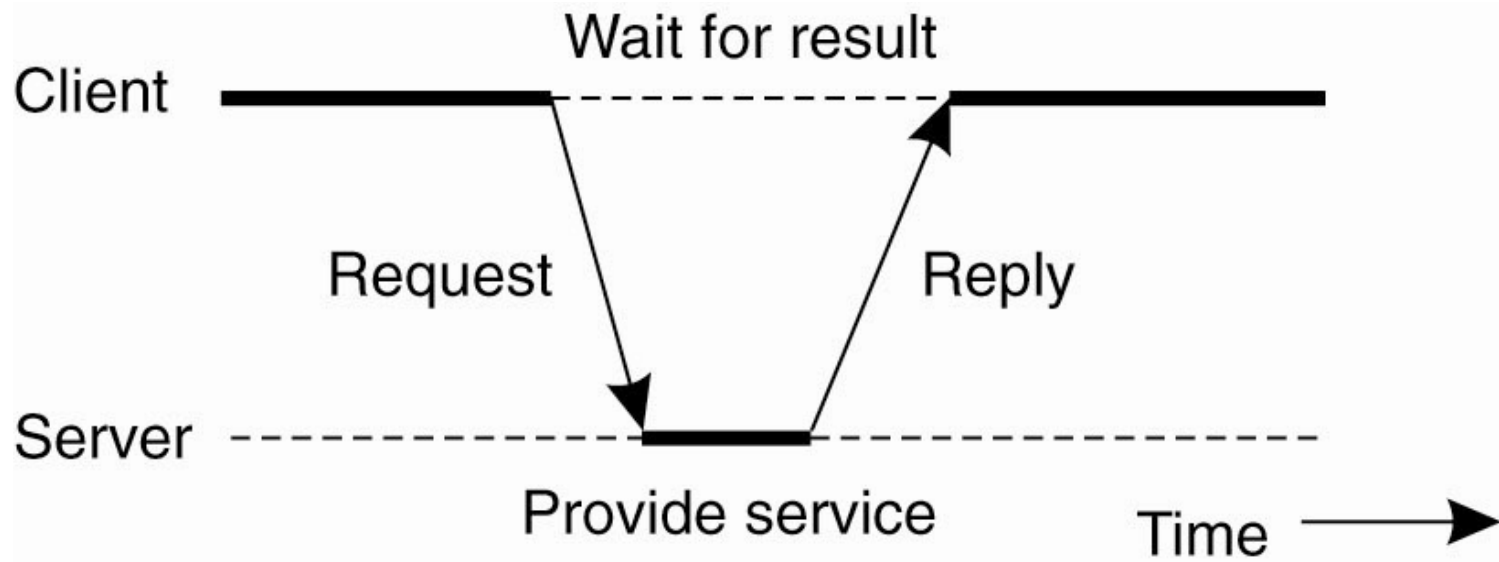
Architectural Models

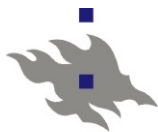
- Architectural models provide a high-level view of the distribution of functionality between system components and the interaction relationships between them
- Architectural models define
 - components (logical components deployed at physical nodes)
 - communication
- Criteria
 - performance
 - reliability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



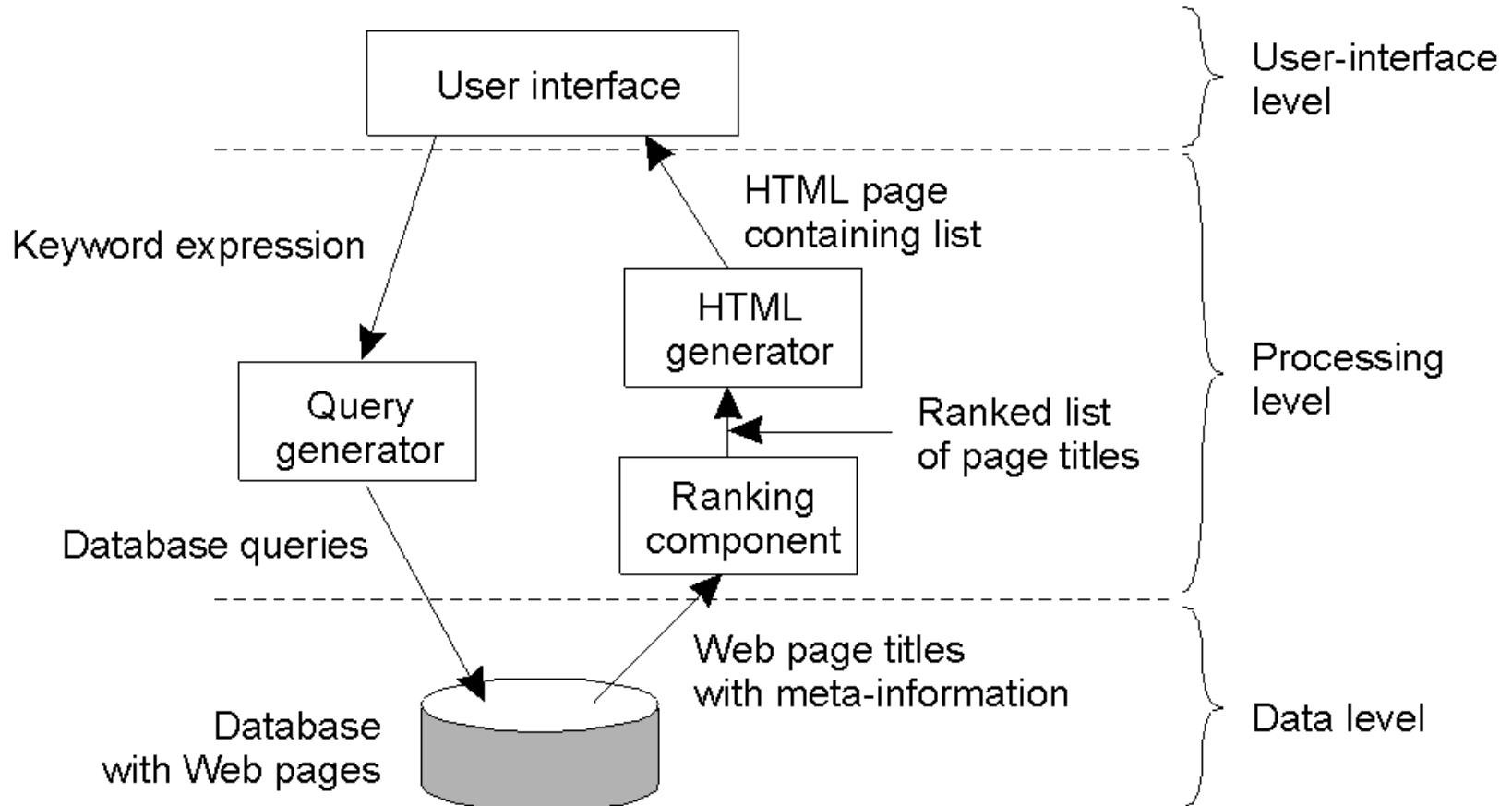
Client-Server Architectures

- General interaction between a client and a server.

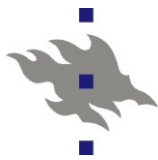




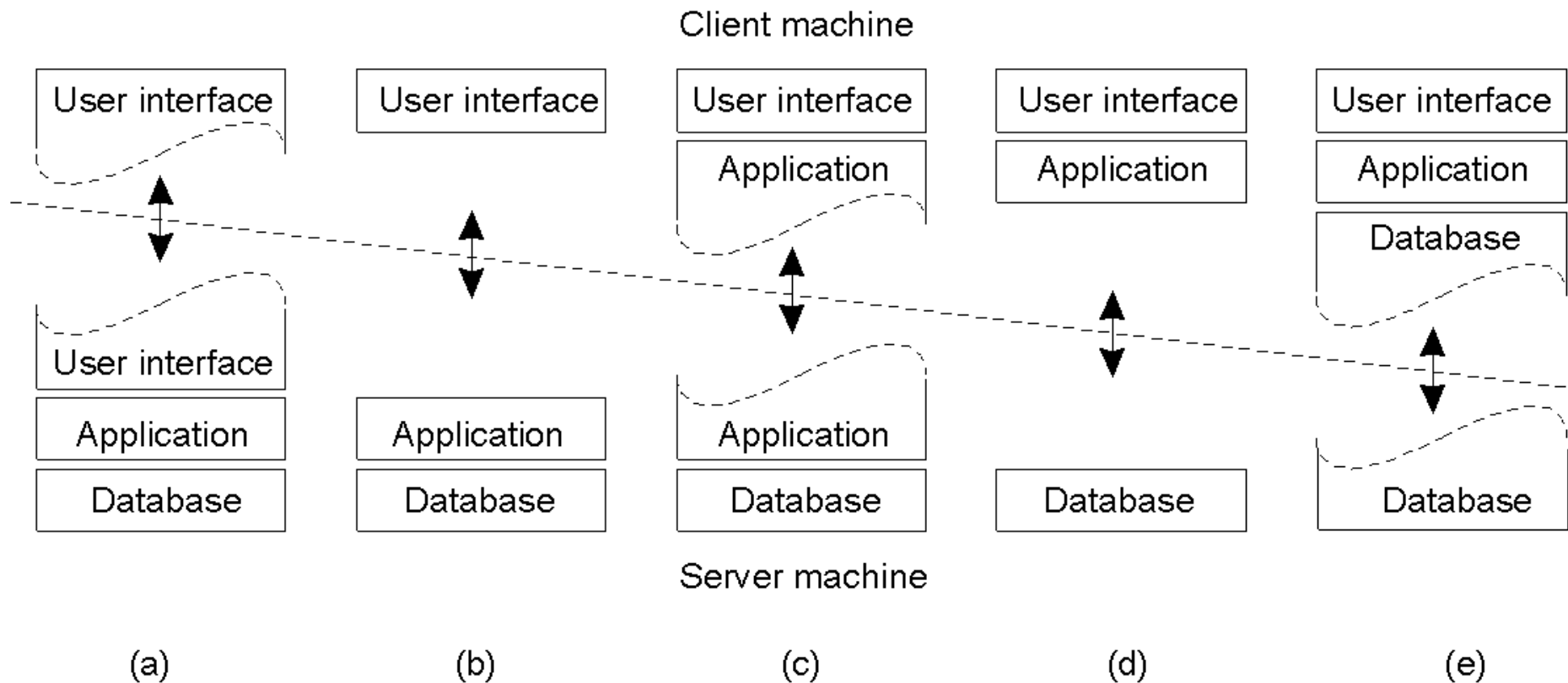
Processing Level



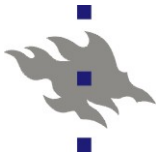
The general organization of an Internet search engine into three different layers



Multitiered Architectures (1)

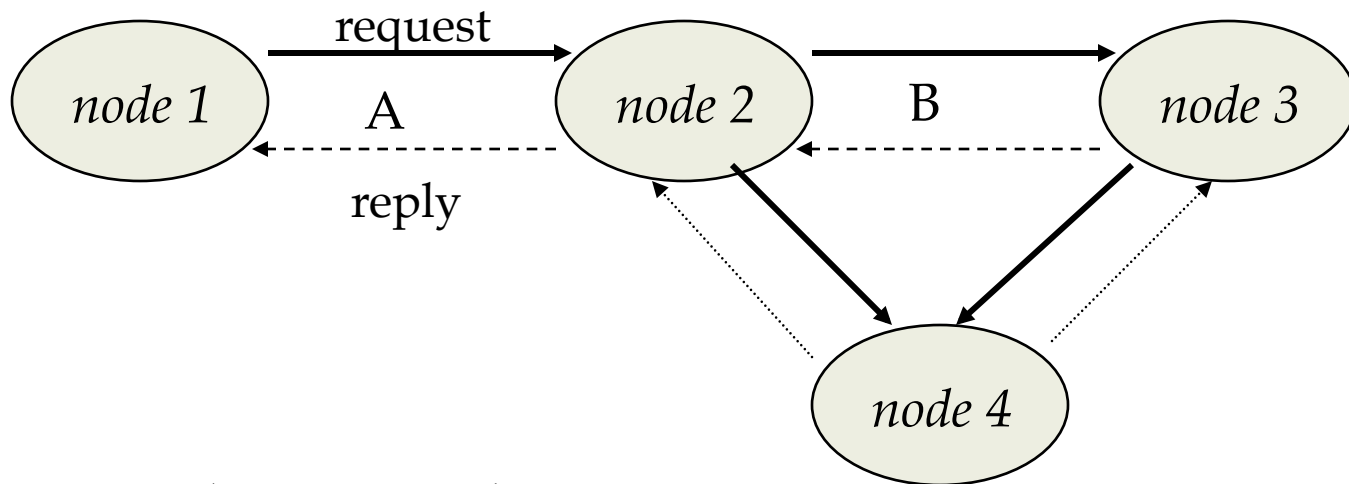


Alternative client-server organizations.



Multitiered Architectures (2)

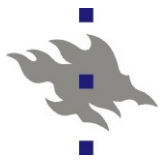
Client - server: generalizations



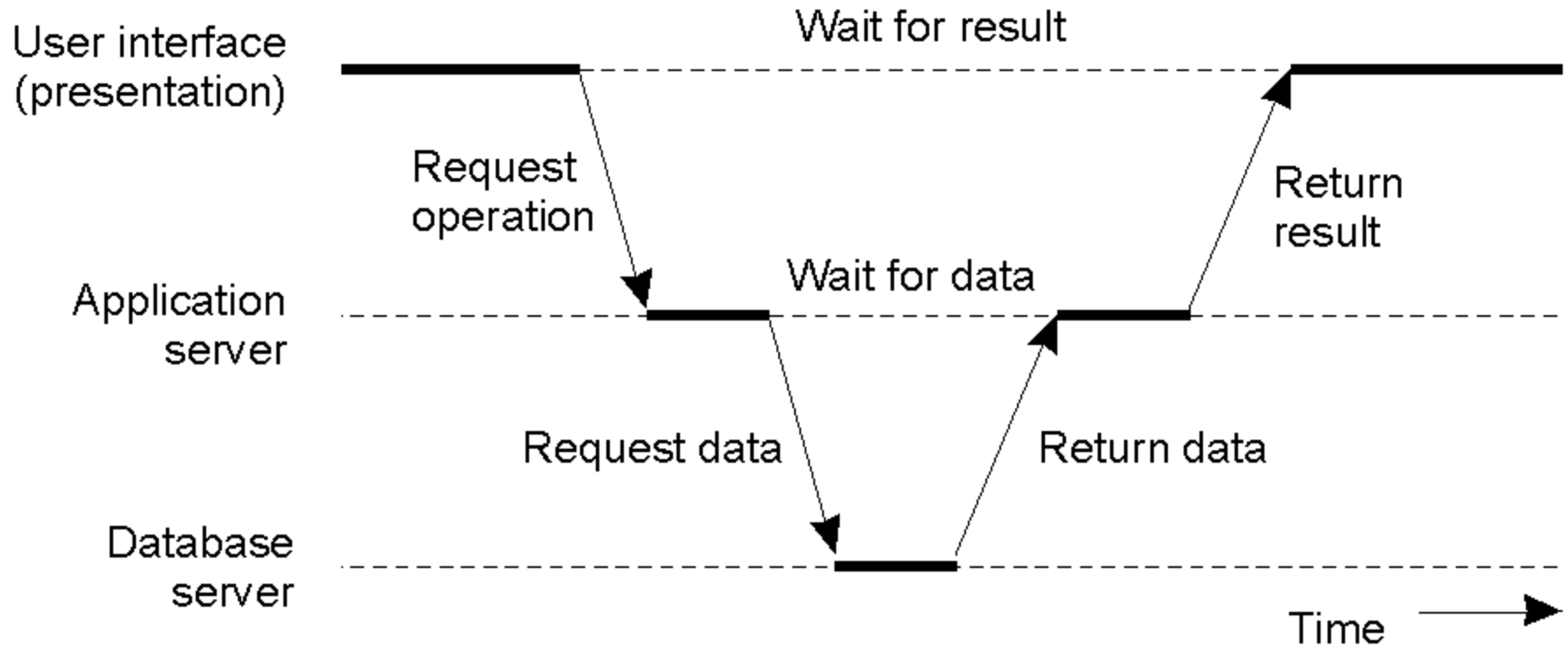
A client: node 1
server: node 2

B client: node 2
server: node 3

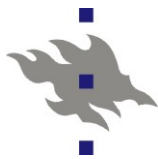
the concept is related
to **communication**
not to nodes



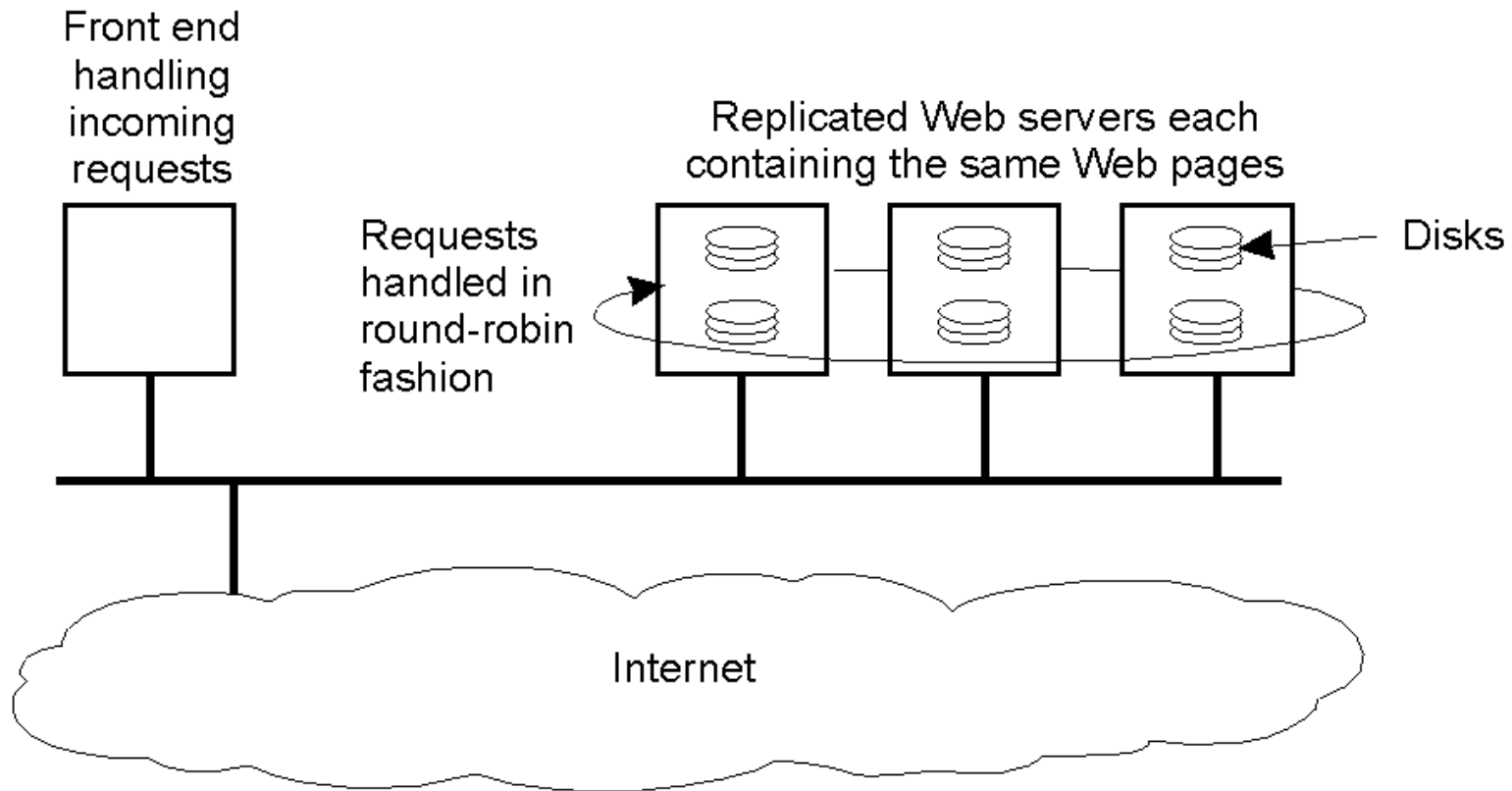
Multitiered Architectures (3)



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