Distributed Operating Systems
Introduction

Ewa Niewiadomska-Szynkiewicz and Adam Kozakiewicz

ens@ia.pw.edu.pl, akozakie@ia.pw.edu.pl

Institute of Control and Computation Engineering
Warsaw University of Technology

Lecture (1)

Introduction:

- √ Definition of a Distributed System (DS)
- √ Goals and Architecture of Distributed System
- √ Hardware Concepts
- √ Software Concepts
- √ Operating Systems
- √ Modern Architectures

Distributed System – Definition

Distributed System (DS)

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

Essential aspects:

hardware - the machines are autonomous
software - the user think they are dealing with the single system

The important characteristics:

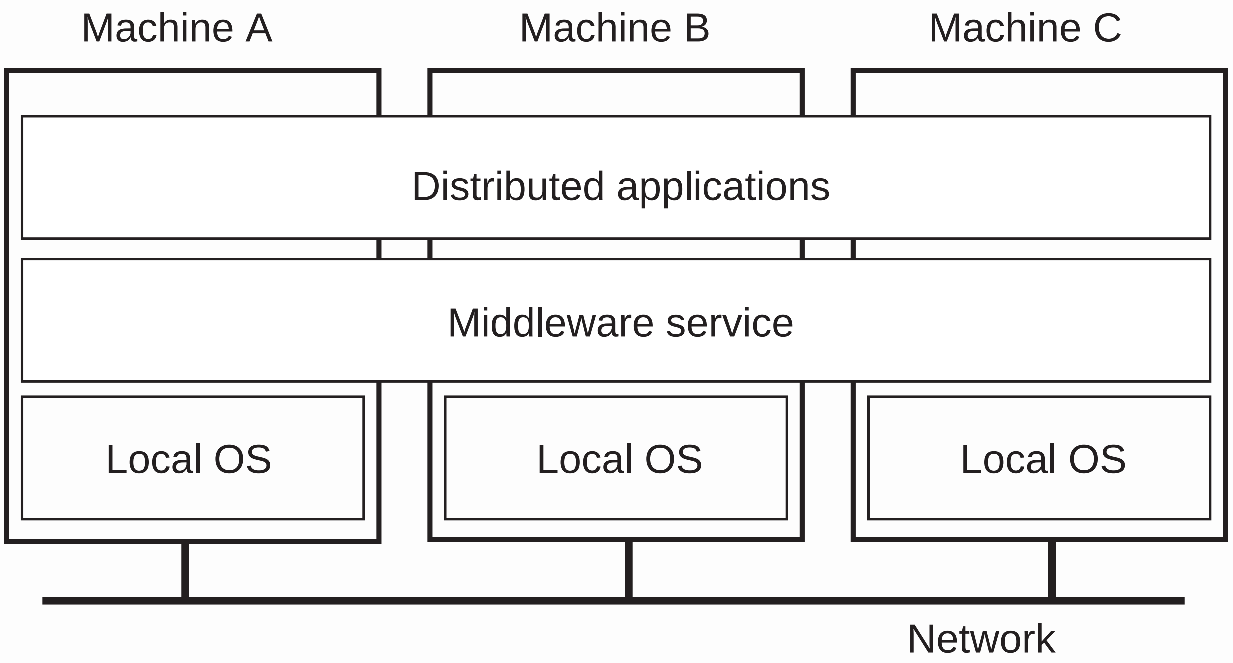
- √ the differences between the various computers and the ways in which they communicate are hidden from users,
- √ users and applications can interact with DS in a consistent and uniform way, regardless of where and when interaction takes place.

Examples: network of workstations in a university, workflow information system

that supports the automatic processing of orders, World Wide Web, etc.

Distributed System – Layered Architecture

To support heterogenous computers and networks while offering a single system view, DS are often organized by means of a layer of software that is logically placed between higher-level (users and applications) and lower level (local OS).



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

Distributed System – Goals

A distributed system should easily connect users to resources (hardware and software), hide the fact that resources are distributed across a network, be open and scalable.

The goals that should be met building DS worth and effort:

- √ connecting users and resources,
- √ concurrency,
- √ transparency,
- √ openness,
- √ scalability,
- √ fault tolerance.

Resource sharing

Allowing multiple users and applications to share resources (local and remote):

- √ **hardware**: CPU, memories, peripheral devices, the network, ...
- √ **software**: data bases, data files, programs, ...

Resource manager:
application managing the set of resources of a given type.

Problems: Security is very important. In current practice, systems provide little protection against intrusion on communication.

Concurrency

Ability to processing multiple tasks at the same time

- √ Each user does not notice that the other is making use of the same resource.
- √ Concurrent access to the shared resource leaves it in a consistent state (consistency can be achieved through locking mechanisms).

Transparency in a Distributed System

An important goal of DS is to hide the fact that its processes and resources are physically distributed across multiple computers.

Transparent system

A distributed system that is able to present itself to users and applications as if it were only a single computer system.

Different forms of Transparency in DS

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
Relocation	Hide that a resource may be moved to another location while in use
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

Access: differences in data representation, naming conventions, files manipulation should be hidden from users and applications.
Location: naming plays an important role.
Replication: resources replicated to increase availability or improve performance.
Failure: the main difficulty lies in inability to distinguish between a dead resource and a painfully slow resource.

Degree of Transparency

- √ Some attempts to blindly hide all distribution aspects is not always a good idea.
- √ A trade-off between a high degree of transparency and the performance.

The goal not to be achieved: **parallelism transparency**.

Parallelism transparency

Transparency level with which a distributed system is supposed to appear to the users as a traditional uniprocessor timesharing system.

Openness

Openness

An open distributed system is a system that offers services according to standard rules that describe the syntax and semantics of those services. Completeness and neutrality of specifications as important factors for interoperability and portability of distributed solutions.

- √ **completeness** - all necessary to make an implementation as it has been specified,
- √ **neutrality** - specification do not prescribe what an implementation should look like.

Interoperability

The extent by which two implementations of systems from different manufactures can cooperate.

Portability

To what extent an application developed for A can be executed without modification on some B which implements the same interfaces as A.

Scalability

Scalability of a system can be measured along at least three dimensions:

- √ scalable with respect to its **size** (more users and resources can be easily added to the system),
- √ **geographically** scalable systems (users and resources may lie apart),
- √ system **administratively** scalable (it can still be easy to manage even if it spans many independent administrative organizations).

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

Examples of scalability limitations

Scaling Techniques (1)

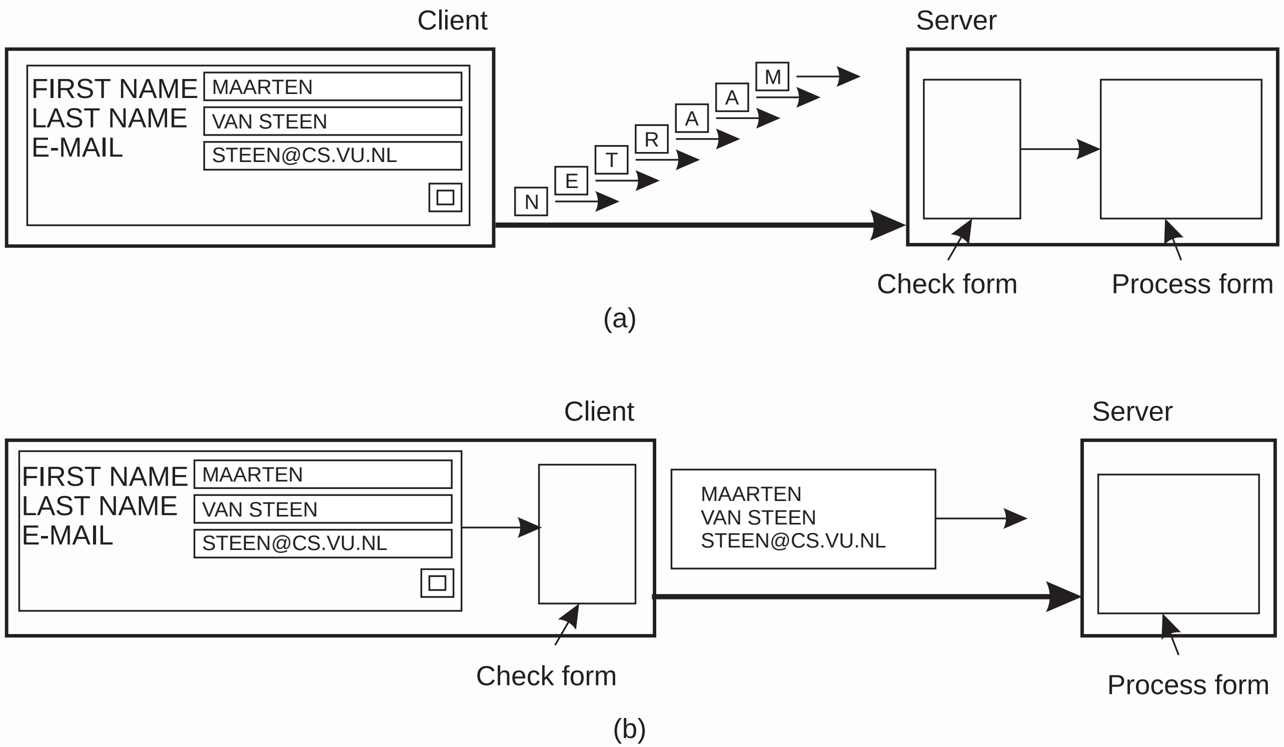
Scalability problems: problems caused by limited capacity of servers and network.
Techniques for scaling:

- √ asynchronous communication (to hide communication latencies),
- √ distribution (splitting into smaller parts and spreading),
- √ replication (to increase availability and to balance the load),
- √ caching (as a special form of replication).

Decentralized Algorithms

1. No machine has complete information about the system state.
2. Machines make decisions based only on local information.
3. Failure of one machine does not ruin the algorithm.
4. There is no implicit assumption that a global clock exists.

Scaling Techniques (2)



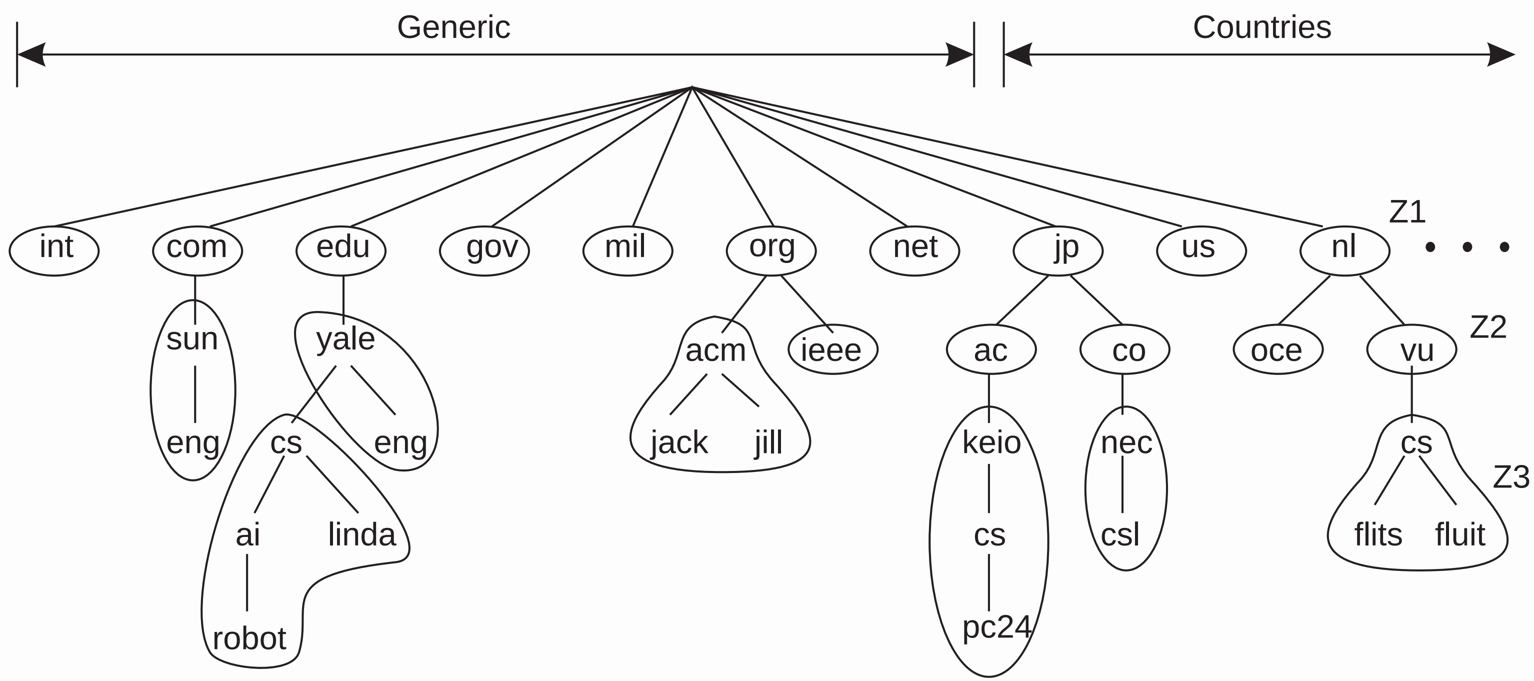
A difference between letting:

a a server

b a client

check forms as they are being filled.

Scaling Techniques (3)



An example of dividing the DNS name space into zones.

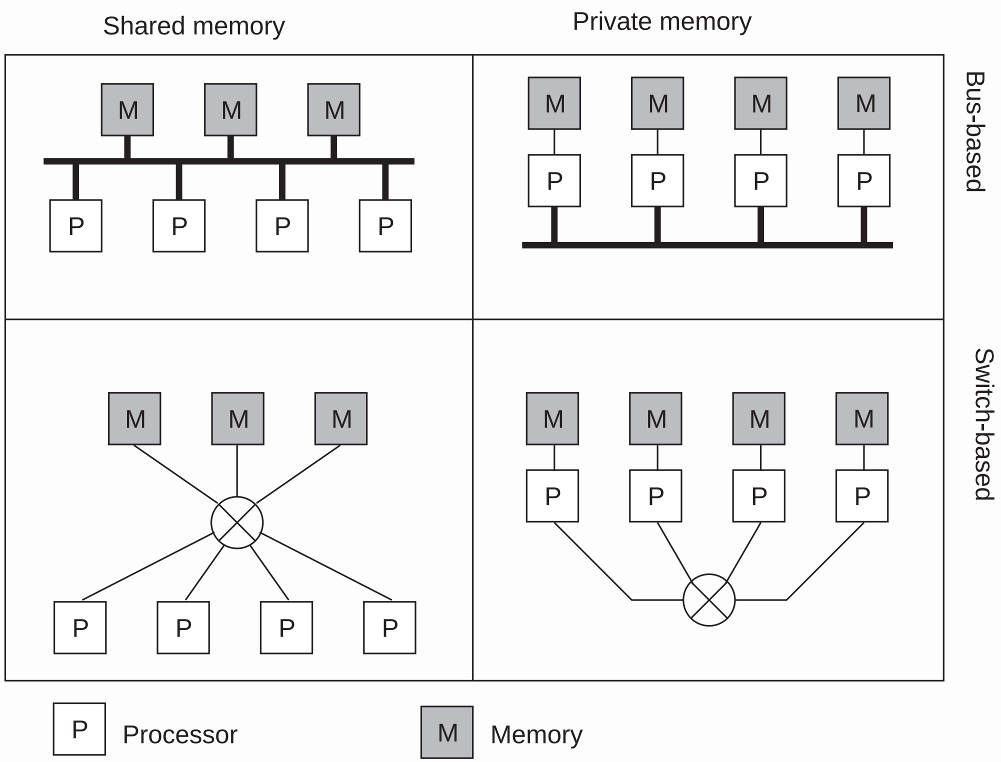
Fault Tolerance

A system can provide its service even in the presence of faults (a user does not notice that a resource fails to work properly and that the system subsequently recovers from that failure)

Solutions for building the fault-tolerant systems:

- √ physical redundancy (hardware and software) - extra equipment or processes are added to make the system as a whole to tolerate the loss or malfunctioning of some components,
- √ reliable group communication,
- √ applications that recover the data and remove the faults.

Hardware Concepts



Different basic organizations and memories in distributed computer systems.

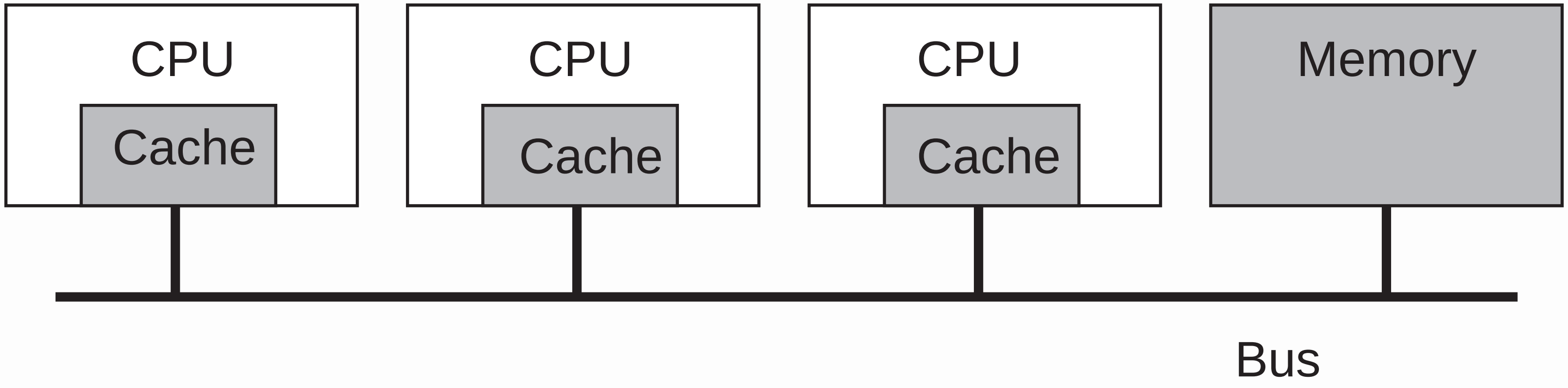
Multiprocessors - systems built of a collection of independent computers that have shared memory.
Multicomputers - systems built of a collection of independent computers that have their local memory (not shared).

Interconnection Network – Architectures

We distinguish two categories based on the architecture of the interconnection network:

- √ **bus** – There is a single network, bus, cable or other medium that connects all the machines (e.g. cable television).
- √ **switch** – Switched systems do not have a single backbone. There are individual wires from machine to machine with many different wiring patterns in use. messages move along the wires, with an explicit switching decision made at each step to route the message along one of the outgoing wires (e.g. worldwide public telephone system).

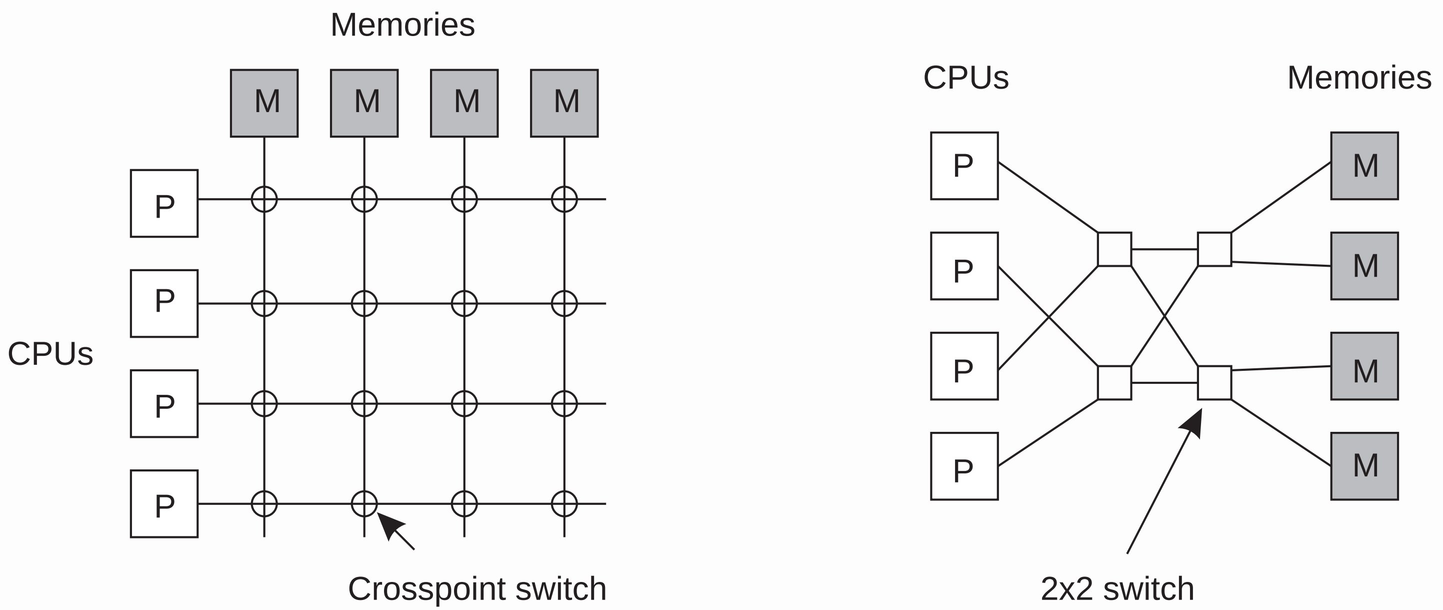
Multiprocessors (1)



A bus-based multiprocessor.

Problem: a few CPUs - bus is overloaded and performance drops drastically. The solution: add a high-speed *cache memory* between the CPU and the bus (the cache holds the most recently accessed words). **Problem**: incoherent memory.

Multiprocessors (2)



(a) (b)

(a) a crossbar switch
(b) an omega switching network (2^k inputs and a like outputs; $\log_2 N$ stages, each having $N/2$ exchange elements at each stage)

NUMA - *NonUniform Memory Access* – hierarchical systems (each CPU – fast access to the local memory and slower to the memories of other CPUs).

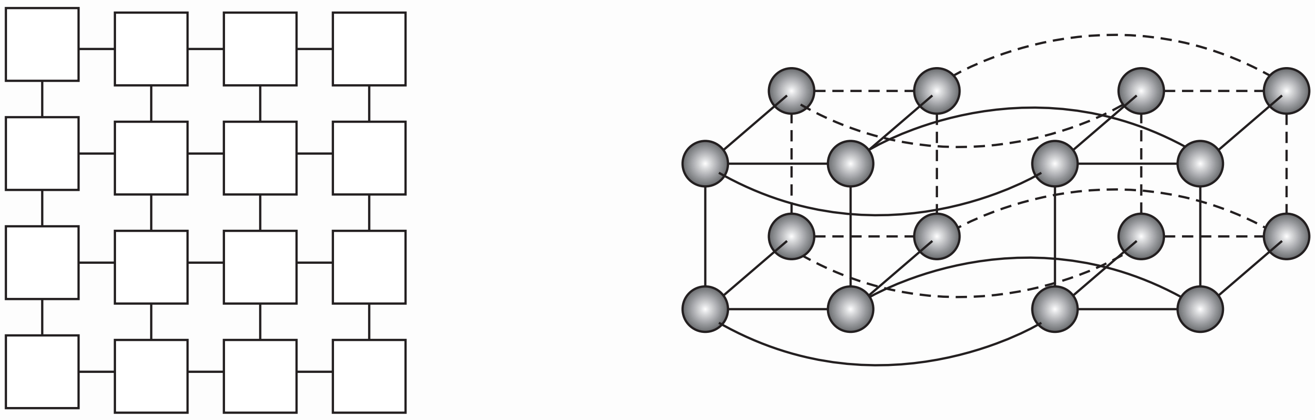
Homogeneous Multicomputer Systems (1)

Homogeneous Multicomputer Systems (System Area Networks)

- √ homogenous machines connected through a single, often high-performance interconnection network that uses the same technology everywhere,
- √ all processors are the same and each CPU has a direct connection to its local memory,
- √ architecture of the interconnection network: switch or bus,
- √ connection topologies: meshes, grids, hipercubes.

Homogeneous Multicomputer Systems (2)

Connection topologies:



(a) (b)

a. grid
b. hypercube

Examples: Massively Parallel Processors (MPPs), Clusters of Workstations

Heterogeneous Multicomputer Systems

- √ multicomputer system may contain a variety of different, independent computers,
- √ the computers in turn are connected through different networks.

Examples:

- √ collection of different local-area computer networks interconnected through an FDD or ATM-switched backbone,
- √ grid systems.

Multiprocessors and Multicomputers - comparison

Multiprocessors:

- √ bus systems - problems with congestion and scalability (single network)
- √ switched systems - scalable but complex, slow and expensive solution.
- √ simple software: shared memory access, synchronization mechanisms provided as software applications.

Multicomputers:

- √ easy to build
- √ scalable solution
- √ complex software: message based communication, problems with buffering, synchronization, lost packets, etc.

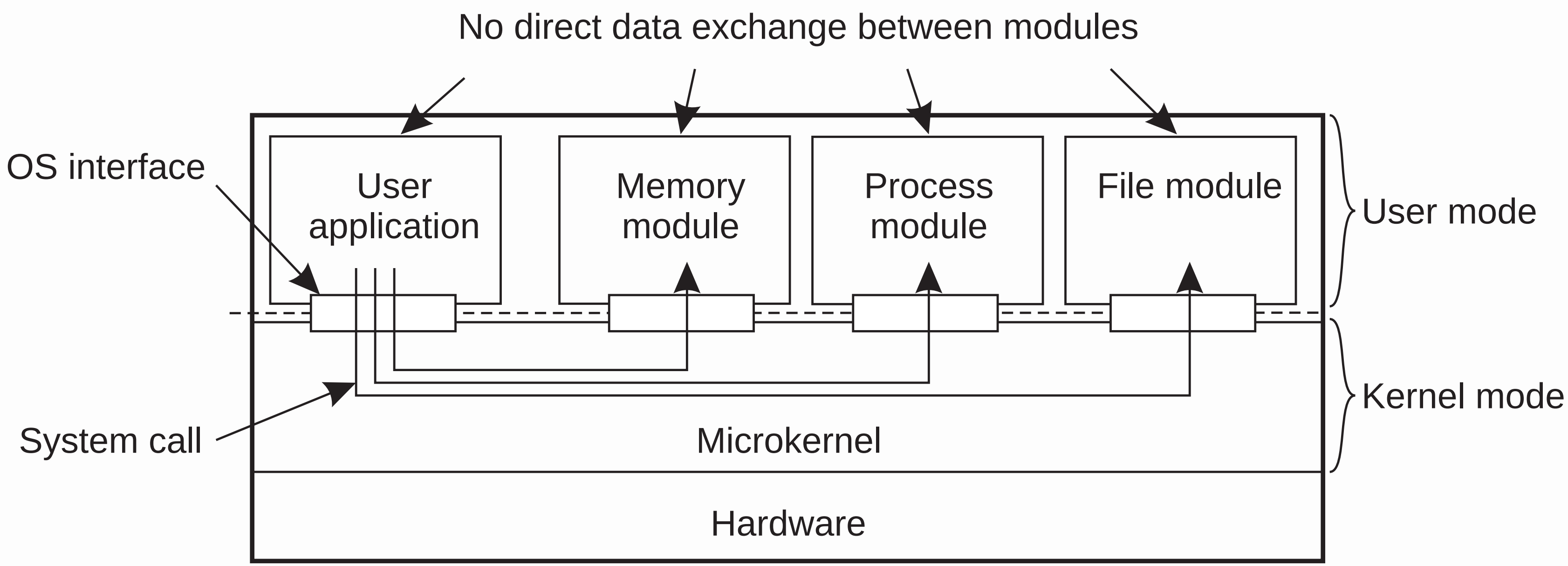
Software Concepts

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

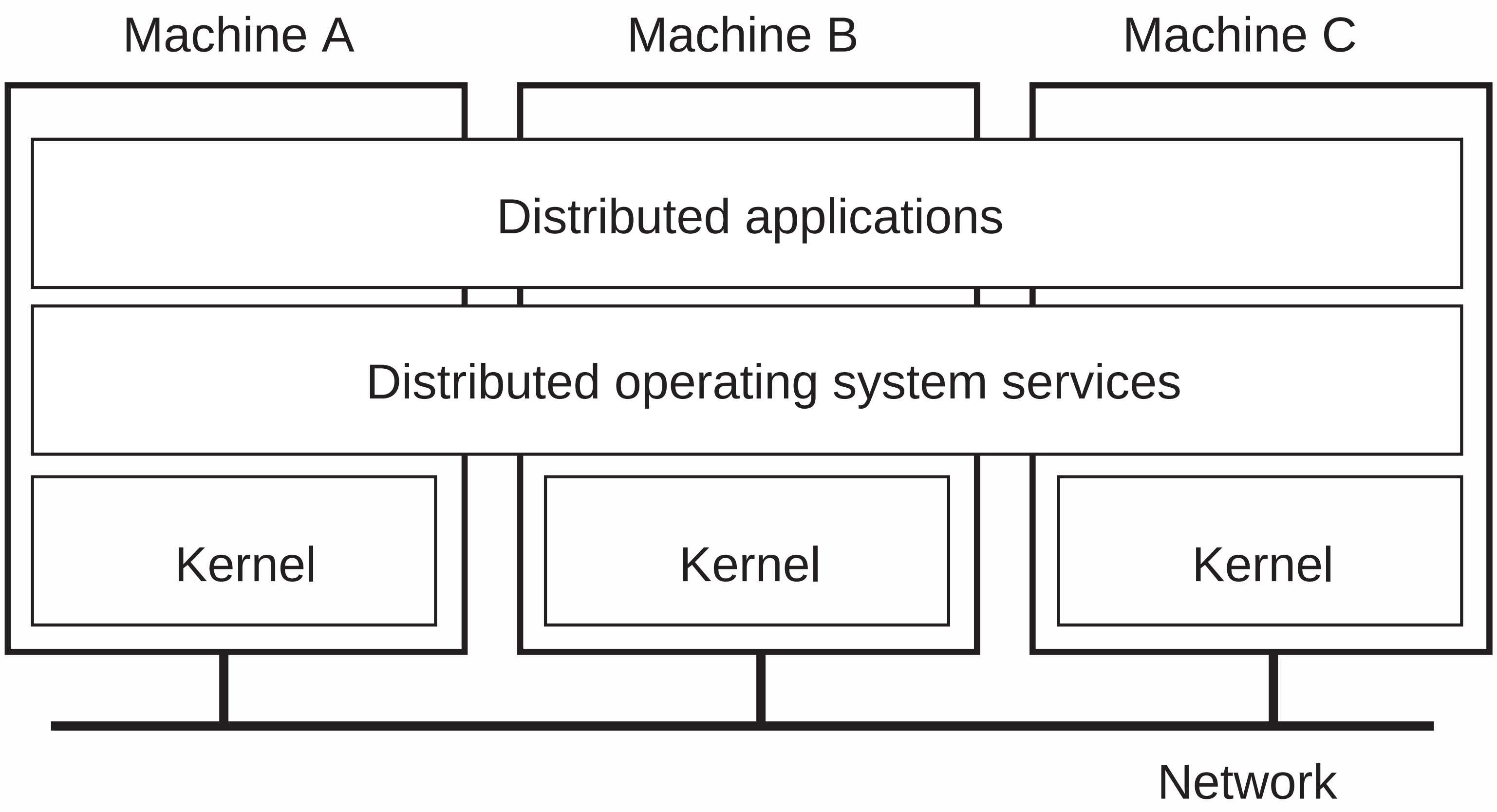
DOS distributed operating system.

NOS network operating system.

Uniprocessor Operating Systems

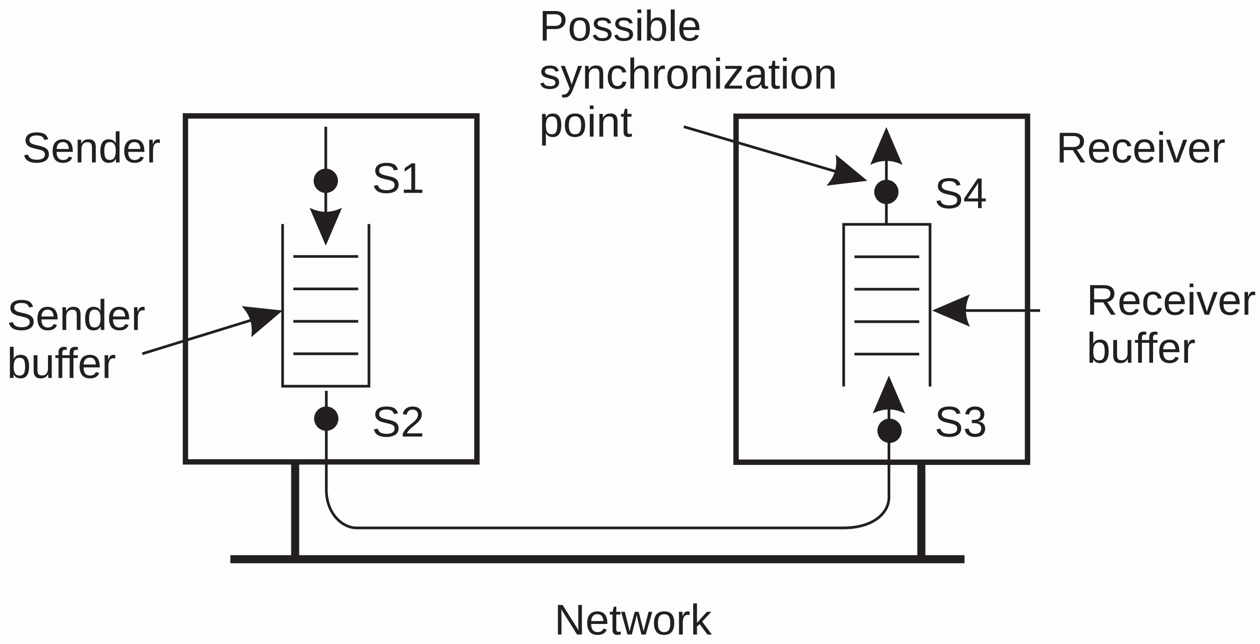


Multicomputer Operating Systems (1)



General structure of a multicomputer operating system.

Multicomputer Operating Systems (2)



Alternatives for blocking and buffering in message passing.

S1: sender blocked when the buffer is full (buffer at sender's side)
S2, S3, S4: there are no sender buffer: S2: the message has been sent, S3: the message has arrived to receiver

S4: the message has been delivered to receiver

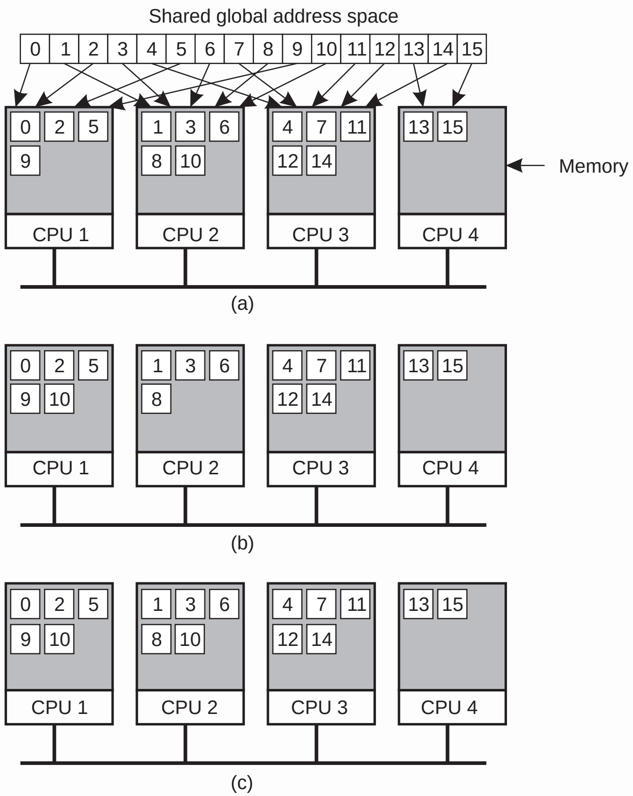
Multicomputer Operating Systems (3)

Synchronization point	Send buffer	Reliable comm. guaranteed?
Block sender until buffer not full	Yes	Not necessary
Block sender until message sent	No	Not necessary
Block sender until message received	No	Necessary
Block sender until message delivered	No	Necessary

Relation between blocking, buffering, and reliable communications.

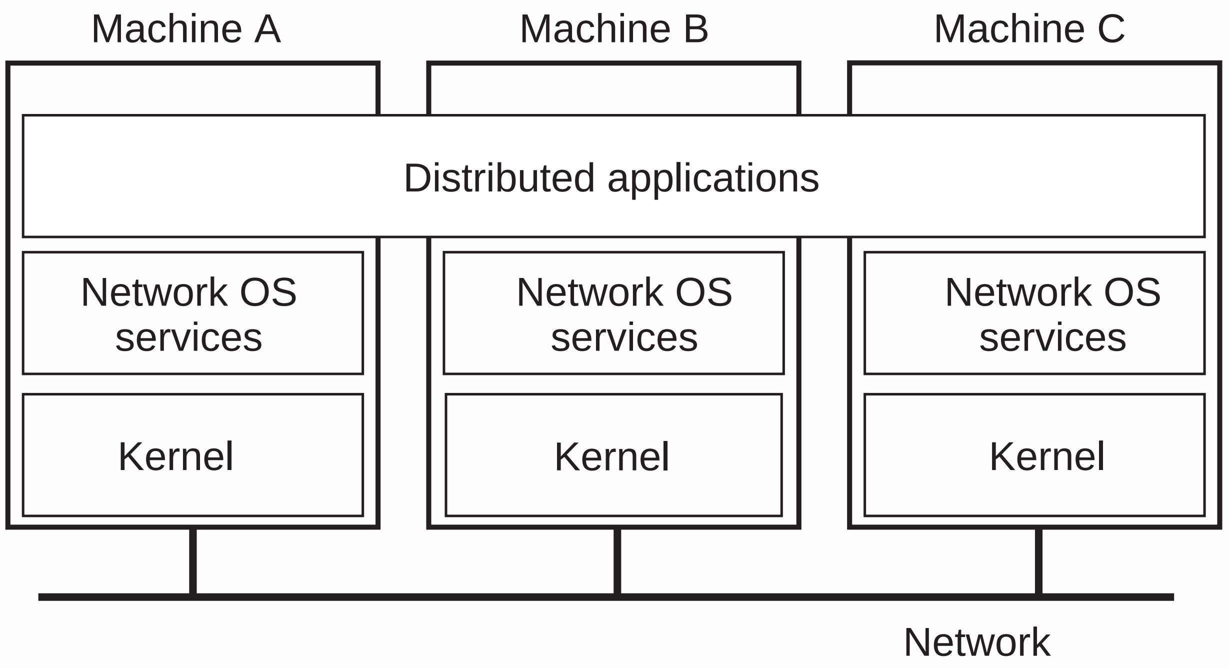
When there is a buffer at the sender's side the OS need not guarantee reliable communication.

Distributed Shared Memory (1)



- √ 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.

Network Operating Systems

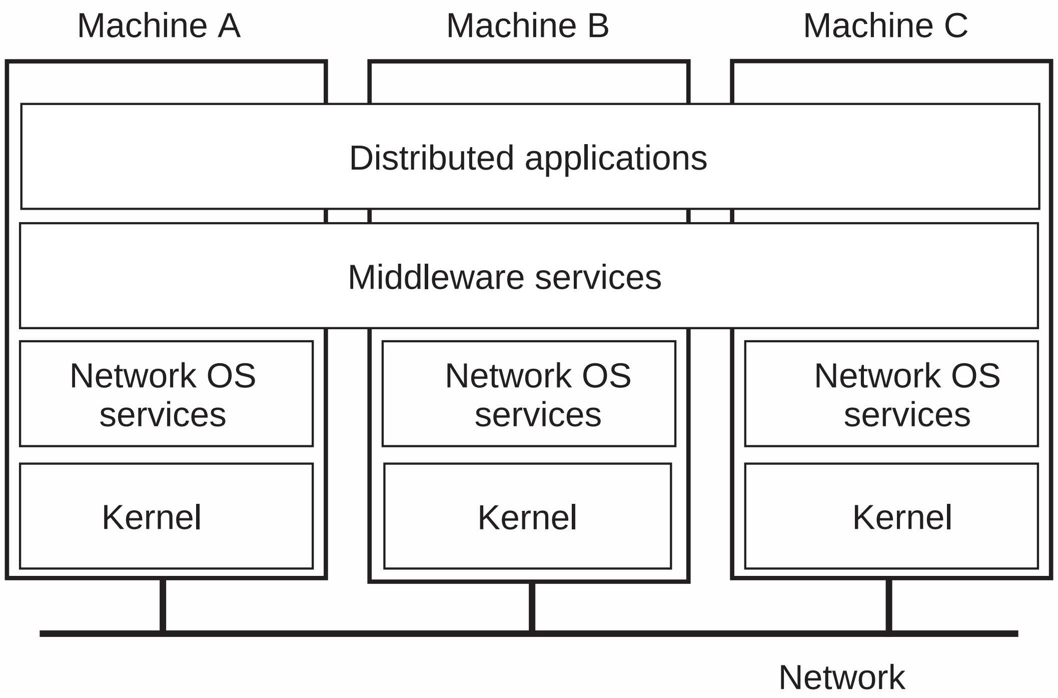


General structure of a network operating system.

NOS services

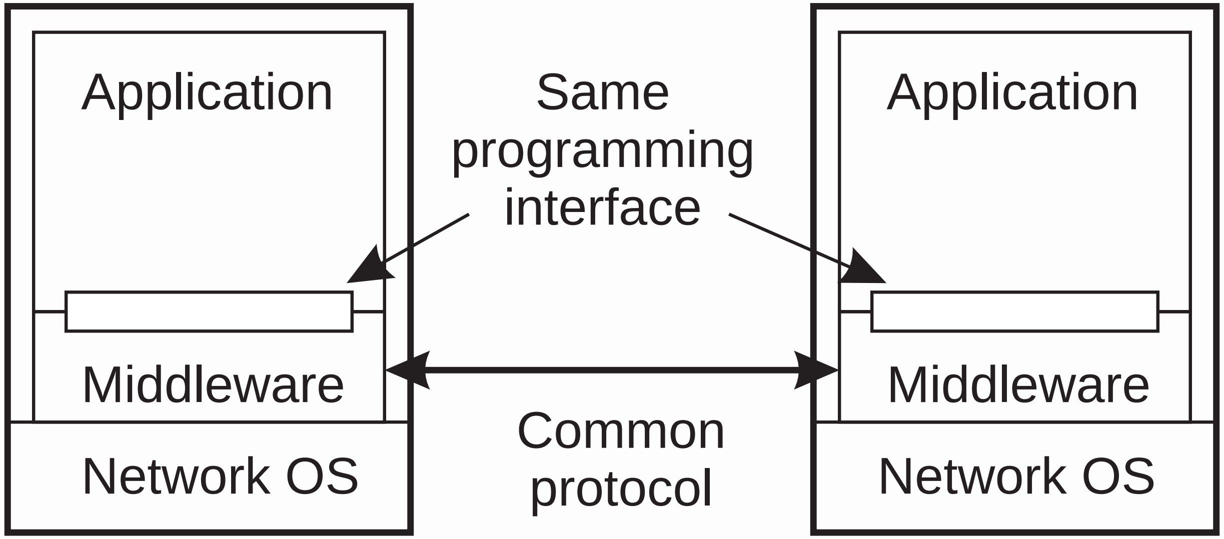
- √ Remote operation (e.g. *rlogin* in UNIX)
- √ Files copy (e.g. *rcp* in UNIX)
- √ Network File System (file server and client)

Positioning Middleware



General structure of a distributed system as middleware.
Middleware – examples: RPC (*Remote Procedure Call*), CORBA (*Common Object Request Broker Architecture*), DCOM (*Distributed Component Object Model*) RMI (*Remote Method Invocation*).

Middleware and Openness



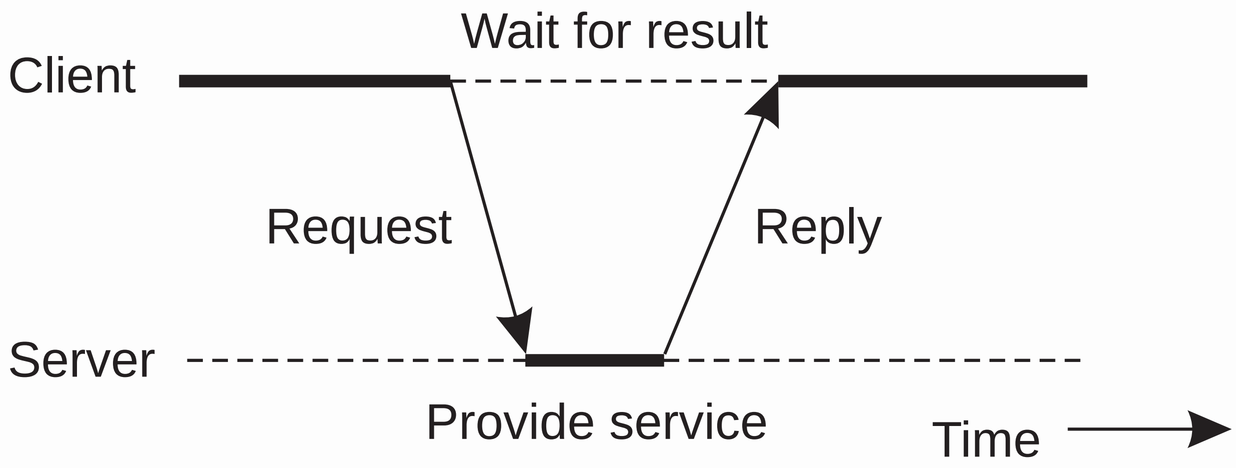
In an open middleware-based distributed system, the protocols used by each middleware layer should be the same, as well as the interfaces they offer to applications.

Comparison of Operating Systems Types

Item	Distributed OS		Network OS	Middleware-based DS
	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

A comparison between multiprocessor OS, multicomputer OS, network OS, and middleware based distributed systems.

Clients and servers

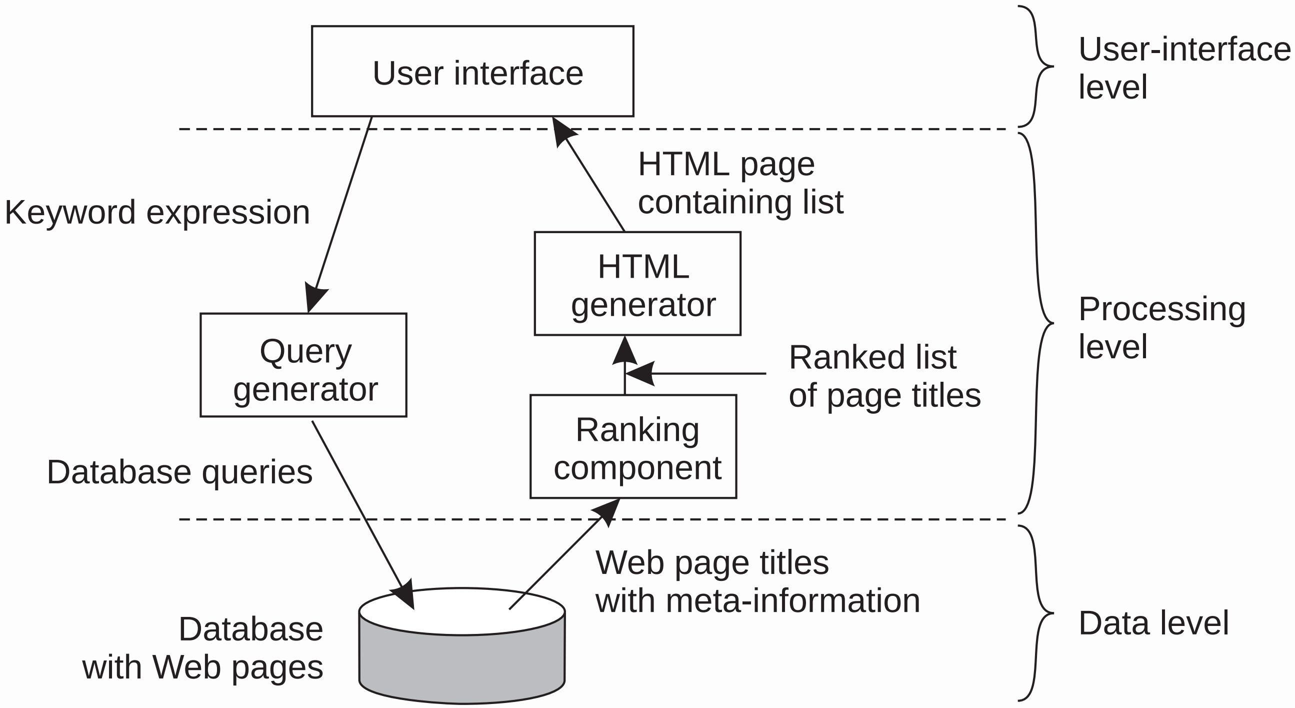


General interaction between a client and a server.

Application Layering

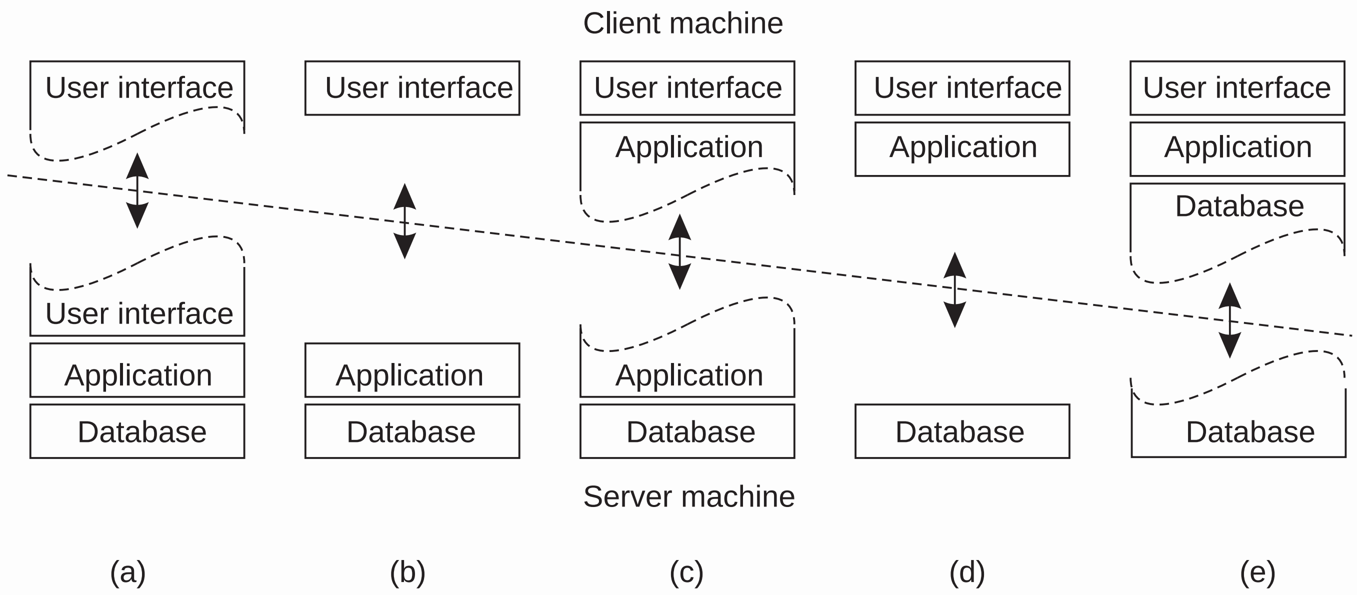
- √ **The user-interface level** – contains all that is necessary to directly interface with the user,
- √ **The processing level** – contains the application,
- √ **The data level** – contains the actual data.

Internet search engine (organization)



- √ The user-interface level - documents in a search engine (typically implemented at client's side),
- √ The processing level - question processing by a search engine (typically implemented at server's side),
- √ The data level index WWW pages in data base (typically implemented at server's side).

Multitiered Architectures (1)



Alternative client-server organizations.

a. terminal-depend part of the user interface on the client machine,

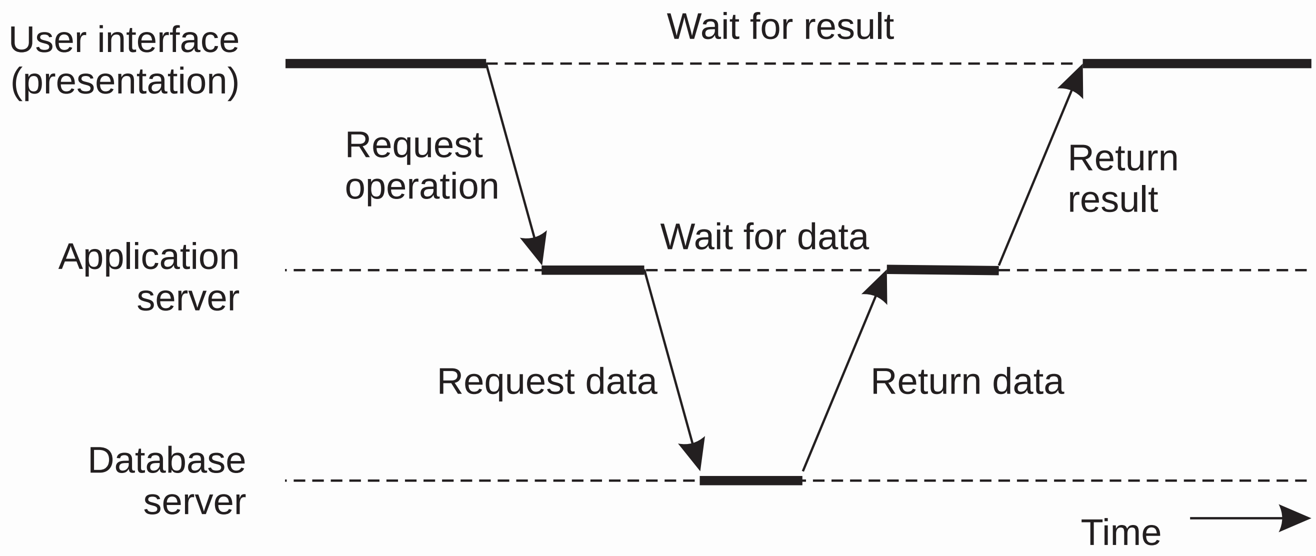
b. the entire user-interface software on the client side,

c. part of the application moved to the client machine,

d. most of the application is running on the client machine,

c. part of the data are collected on the client's local disc.

Multitiered Architectures (2)



An example of a server acting as a client.

Modern Architectures (1)

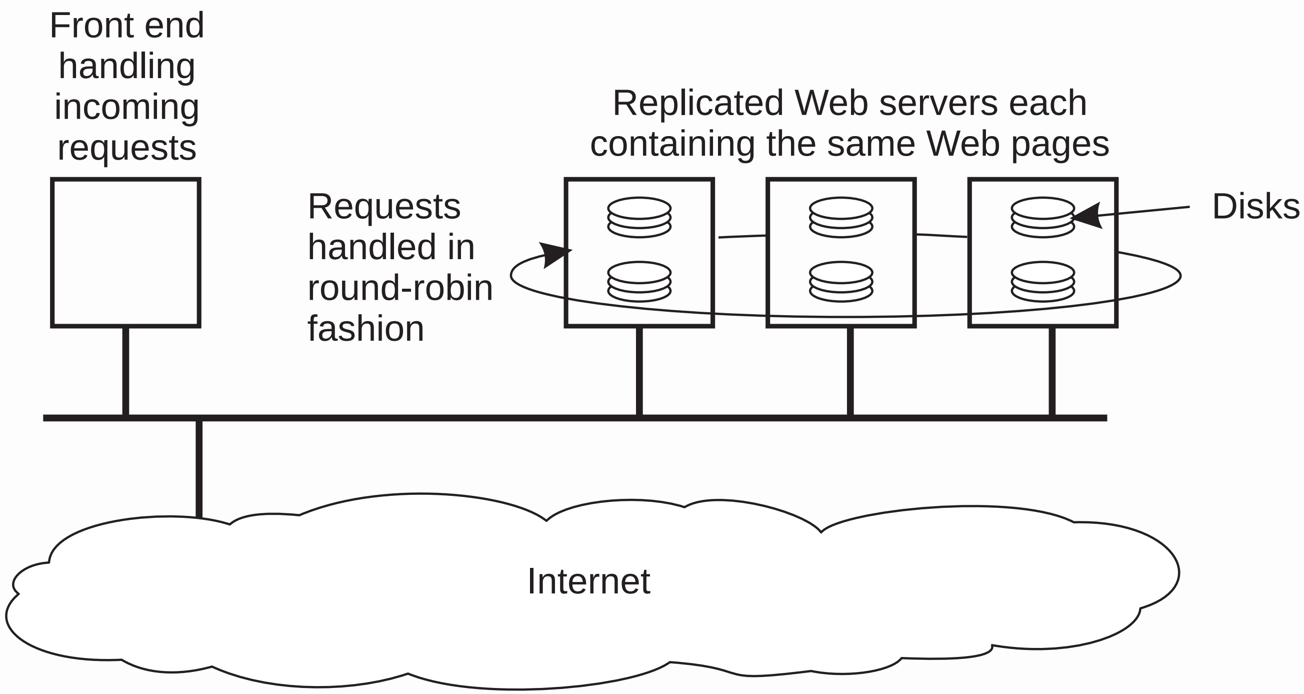
Vertical distribution

Achieved by placing logically different components on different machines (multitiered architecture).

Horizontal Distribution

Client or server may be physically split up into logically equivalent parts, but each part is operating on its own share of the complete data set, thus balancing the load.

Horizontal Distribution – Example



An example of horizontal distribution of a Web service.