CS6223 Distributed Systems

Lecturer: *Dr. X. Lin*

Office: ACAD Y6423

Telephone: 2788 9723

Email: *csxlin@cityu.edu.hk*

http://www.cs.cityu.edu.hk/~cs6223

18:20-20:20 pm, Monday

(20:30-21:20 pm - tutorial)

B5-310

CS6223 Distributed Systems

Textbooks:

Distributed Systems, Principles and Paradigms, by A. Tanenbaum and M. V. Steen, Prentice Hall, 2002.

ISDN: 0-13-088893-1

Reference Book:

Distributed Systems: Concepts and Design, *by Coulouris, Dollimore* and *Kindberg.*Edition 3, © Addison-Wesley 2001

ISDN 0201-61918-0

Course Outline

- Introduction to Distributed Systems (DS)
- Process & Inter-Process Communication in DS
- Naming
- Synchronization in DS
- Consistency & Replication
- Fault Tolerance
- Security in DS
- Distributed System Paradigms
- New Developments in Distributed Systems

Course Work and Assessment

- **Course Work: 30%:**

- One middle-term test: 15%
- One assignment: 15%

An assignment involving the development of a distributed application using various technologies related to the course.

- **Final Examination (two-hour): 70%**

For a student to pass the course, at least 30% of the maximum mark for the examination must be obtained.

1. Introduction to DS

- The definition of distributed systems (DS) and characteristics
- Major goals in building DS
- Hardware and software concepts in DS
- Middleware in DS
- Client-server model

Learning Objectives

- To understand the (informal) definition of distributed systems and its implications;
- To examine the major goals in building DS: connecting users and resource, transparency, openness and scalability;
- To understand the basic architectures of the underlying hardware in DS, focusing on multiprocessor and multicomputer systems;
- To examine the major features of distributed operating system (DOS) and network operating system (NOS);
- To study how middleware is developed on the top of the NOS to build a DS;
- To gain a good understanding of client-server model in DS.

What Is A Distributed System?

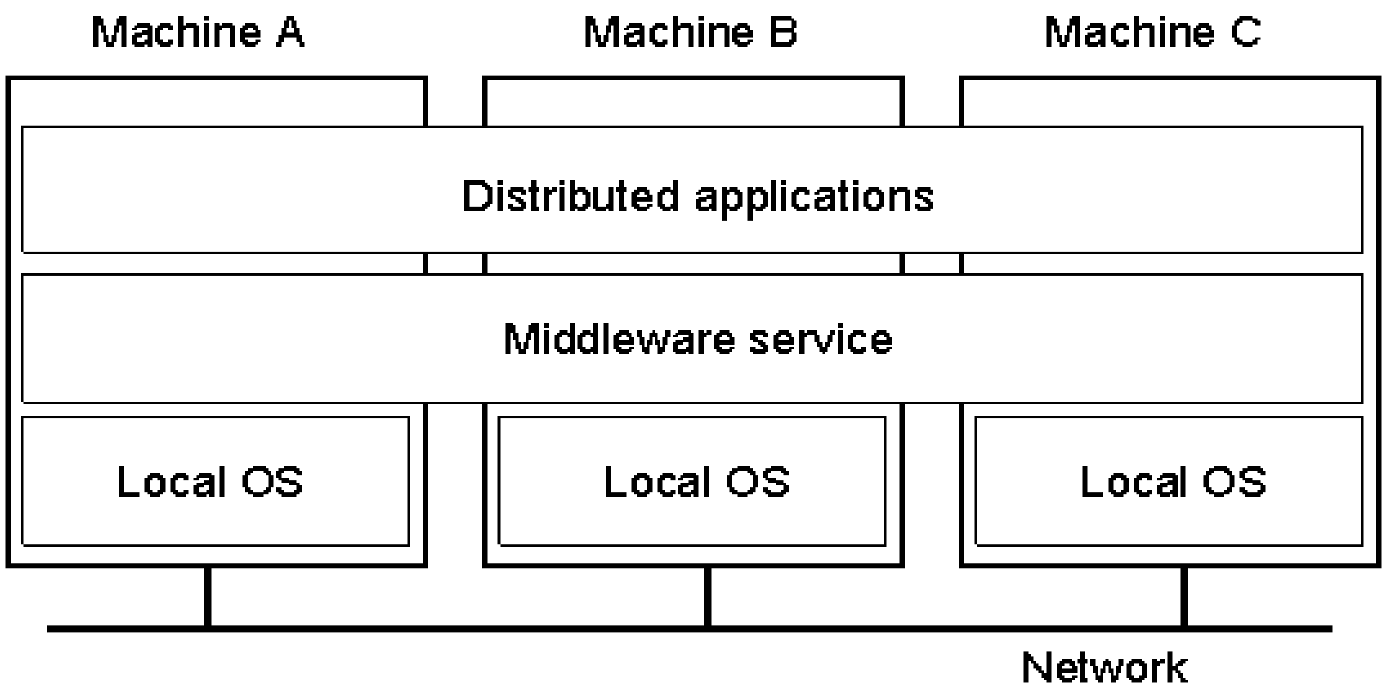
- *A distributed system is a collection of independent computers that appears to its users as a single coherent system.*

  - A. S. Tanenbaum et al. 2002

- *A distributed system is defined as one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages.*

  - G. Coulouris et al. 2001

Distributed System



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

Some implications from the definition of DS

- **Concurrency**: In a DS, concurrent program execution is the norm, sharing resources such as web pages or files when necessary;
- **Independent failures**: Each component of the system can fail independently, leaving the others still running, which may not immediately made known to the other components.
- **No global clock**: There is no single global notion of the correct time. There are limits to the accuracy with which the computers in a network can synchronize their clocks.

WHY Distributed System?

- The sharing of resources is a main motivation for constructing distributed systems.
- The DS may provide distribution-transparency platforms for easy programming for distributed applications.
- Some applications are inherently distributed (e.g. banking and supermarket chain etc).
- Other good features of DS include high performance/cost, reliability, scalability, and flexibility etc.

Goal 1: Connecting users and resources

- Computer resources include all hardware resources (e.g., disks, printers), software resources (e.g., compiler) and data resources (e.g., files) etc.
- The main goal of a DS is to make it easy for users to access remote resources, and share them with other users in a controlled way.
- Connecting users and resources also makes it easier to collaborate and exchange information.
- Security becomes a big issue as connectivity and sharing increase.

Goal 2: Distribution Transparency

- *Transparency* is defined as the concealment from the user and the application programmer of the separation of components in a DS.
- Achieving distribution transparency makes everyone into thinking that the collection of machines is simply an old-fashioned time-sharing system, instead of a collection of independent components.
- The transparency is generally preferable for any DS. It also should be considered together with other issues such as performance.

Transparency in a Distributed System

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

Different forms of transparency in a distributed system.

Goal 3: Openness

- *Openness* is the characteristic that determines whether the system can be extended in various ways.
- Open DS: offers services according to standard rules that describe the syntax and semantics of those services.
- In DSs, services are generally specified through *interfaces*, which are often described in an *Interface Definition Language* (IDL).
- The interface definitions written in IDL usually only capture the syntax of the services (e.g., function name, parameter types, return value, exception etc) The semantics of the interfaces often is given in an informal way (e.g., using natural language).

Openness: Summary

- Open systems are characterized by the fact that their key interfaces are published.
- Open DSs are based on the provision of uniform communication mechanism and published interfaces for access to shared resources.
- Open DSs can be built from heterogeneous hardware and software, possibly from different vendors. The conformance of each component to the published standard must be carefully tested and verified to ensure the system works correctly.

Goal 4: Scalability

- The number of computers in the world is unlimited. Therefore, accessing shared resources should be nearly independent of the size of the network.
- A system is described as *scalable* if it will remain effective when there is a significant increase (or decrease) in the number of resources and the number of users.
- Scalability presents a number of challenges such as how to control the cost of physical resources and the performance loss, and preventing software resources running out, etc.

Scalability: Scaling Techniques

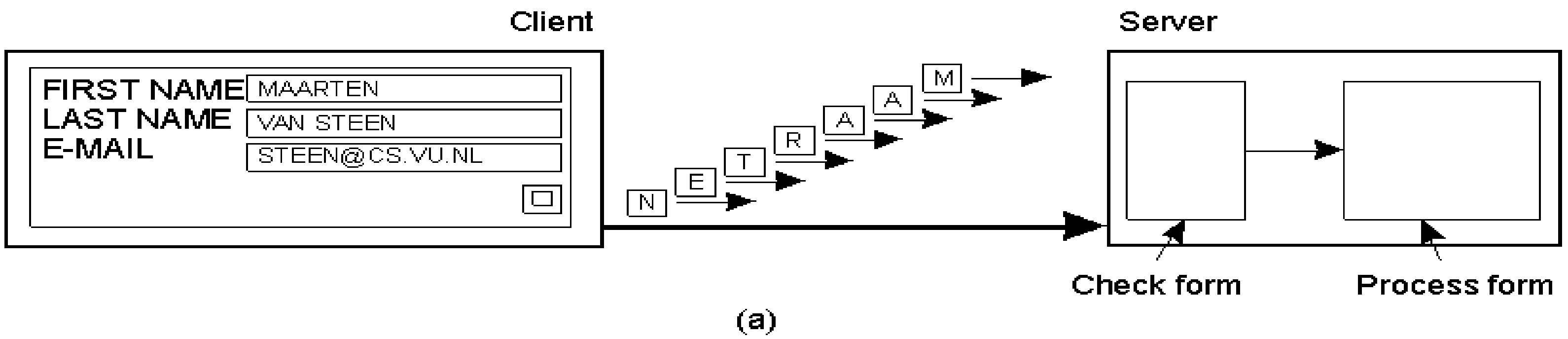
- The scalability problems in DSs appear as performance problems caused by limited capacity of server and network. There are basically three techniques for scaling:

  * *hiding communication latencies*: try to avoid waiting for response to remote services as much as possible (using asynchronous communication); or reduce the overall communication, as the example shown in the next slide.

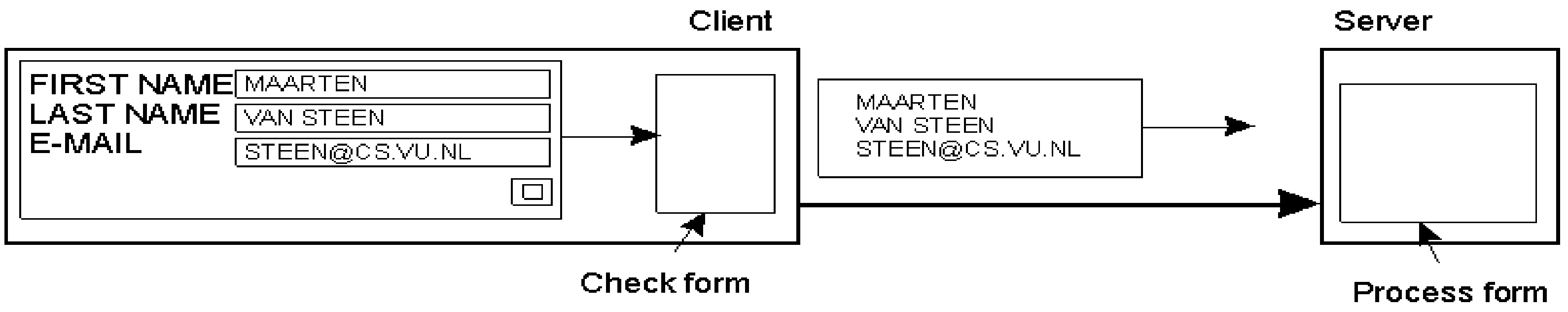
  * *distribution*: taking a component, splitting it into smaller parts, and subsequently spreading those parts across the system. Example: Internet Domain Name System (DNS).

  * *replication*: replicating components across a DS to increase availability and balance the load between components for better performance.

Scaling Techniques: reducing the overall communication



(a)



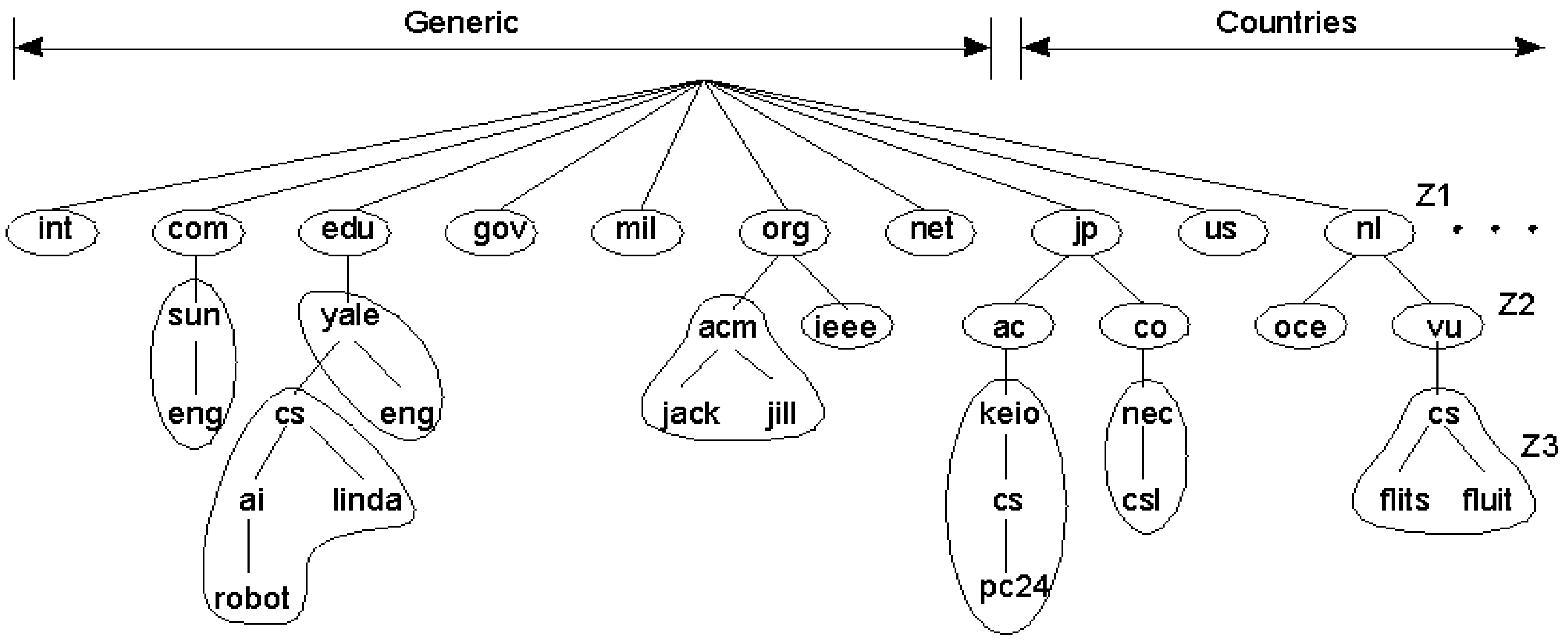
(b)

The difference between letting:

a) a server or

b) a client check forms as they are being filled

Scaling Techniques: distribution



An example of dividing the DNS name space into zones.

Hardware Concepts

- There are several ways to organize hardware in multiple-CPU systems, especially in teams of how they are interconnected and how they communicate.
- Base on *memory organization*:

  Multiprocessor: using shared memory

  communication: through shared variables

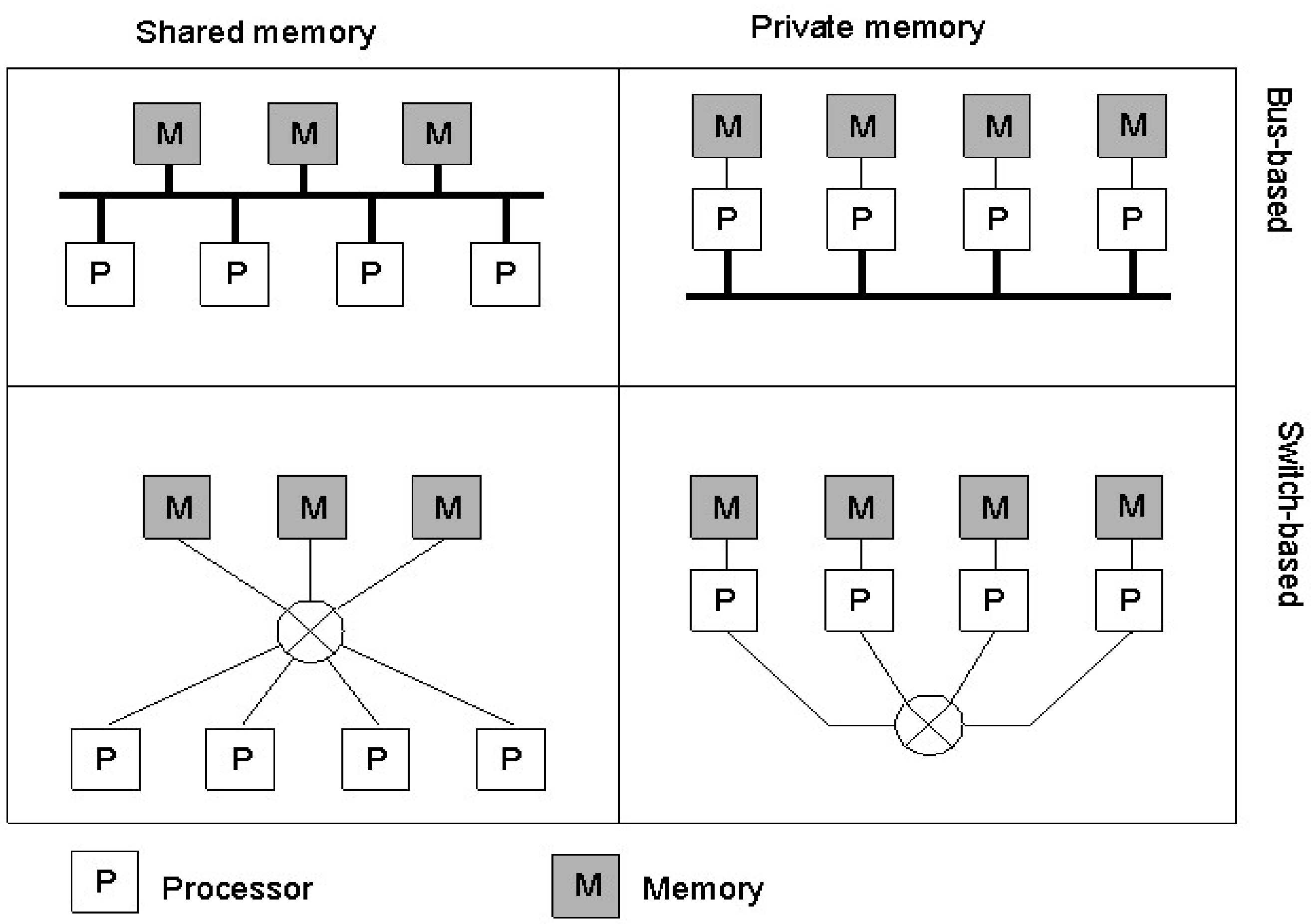
  Multicomputer: no shared memory, using private memory

  communication: message-passing
- Based on the *architecture of interconnection network*:

  Bus-based systems (multiprocessor or multicomputer)

  Switch-based systems (multiprocessor or multicomputer)

Hardware Concepts

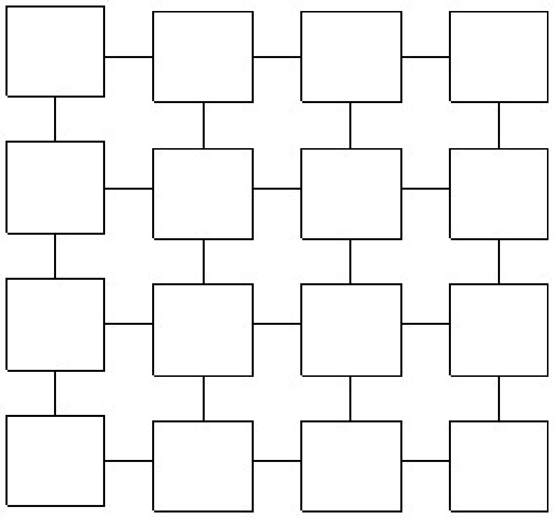


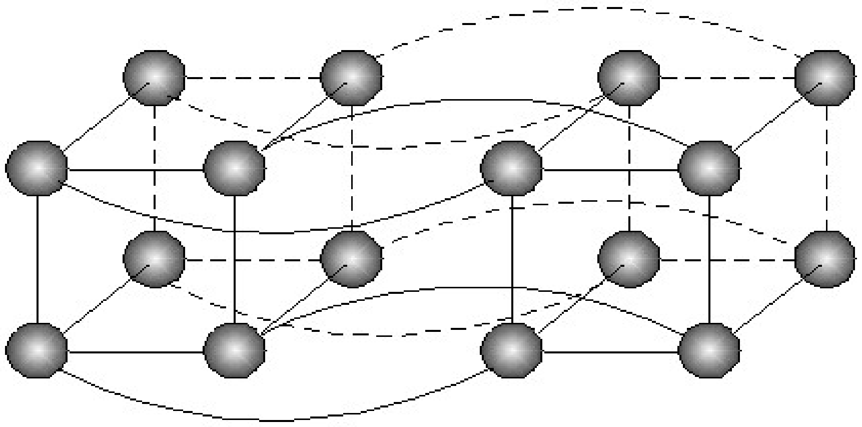
Different basic organizations and memories in multiple-CPU computer systems

Hardware Concepts: Multicomputers

- *Homogeneous* multicomputers (usually used in parallel systems): a single interconnection network, all processors are the same and generally have access to the same amount of private memory.
- *Heterogeneous* multicomputers (usually used in distributed systems): a variety of different, independent computers connected through different networks.
- Due to the large scale, inherent heterogeneity, and lack of global system view in heterogeneous multicomputer, sophisticated software is needed to build applications, developing a distributed system (DS). Thus DSs usually have a software layer (middleware) to provide transparency.

Homogeneous Multicomputer Systems


(a)


(b)

a) 2D-Mesh (Grid)

b) Hypercube

Software Concepts: Distributed OS

- Distributed systems act as *resource manager* (like traditional OS), and more importantly, attempt to hide the heterogeneous nature to provide a virtual single system on which applications can be easily executed.
- *Distributed operating systems* (DOS): managing multiprocessor and homogeneous multicomputers. DOS aims to support high performance through multiple processors.
- In multiprocessor, DOS supports for multiple processors having access to a shared memory and protects data against simultaneous access the same shared memory locations.
- In multicomputer, DOS offers the message-passing facilities to applications.
- The main goal of DOS is to hide the intricacies of managing the underlying hardware such that it can be shared by multiple processes.

Software Concepts: Network OS

- *Network operating system* (NOS): used for heterogeneous multicomputer systems. In additional to managing the underlying hardware and other resources, it makes the local services available to remote clients. NOS does not provide a single system image (transparency) to the users.
- *Example*: some (non-transparency) services NOS may provide (in UNIX)

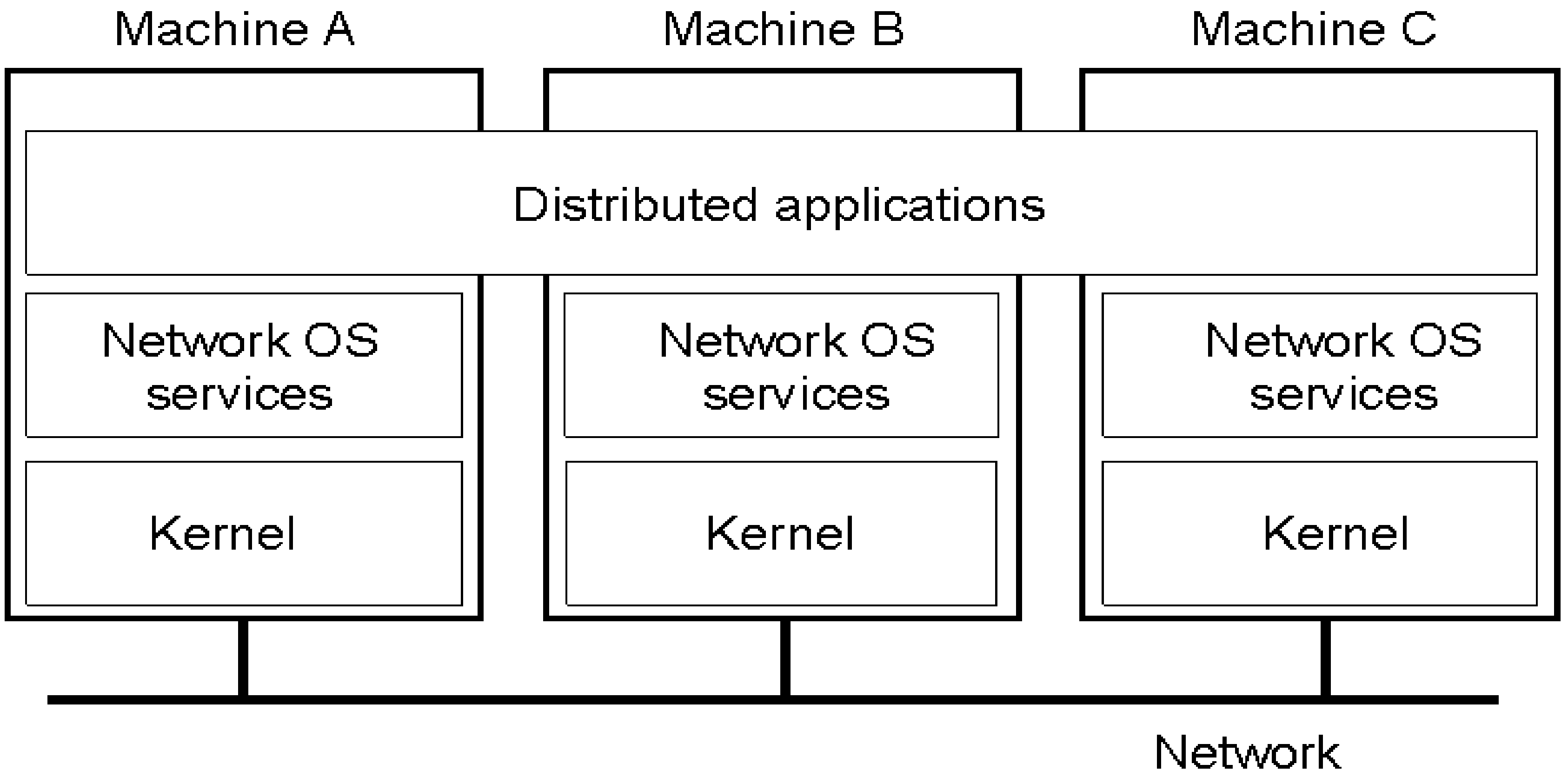
```
rlogin machine

rcp machine1:file2 machine2:file2
```

- The lack of transparency in NOS has some drawbacks, such as they are harder to use and manage, and introducing some security problem.

Network Operating System

- General structure of a network operating system.



Software Concepts: An Overview

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 heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

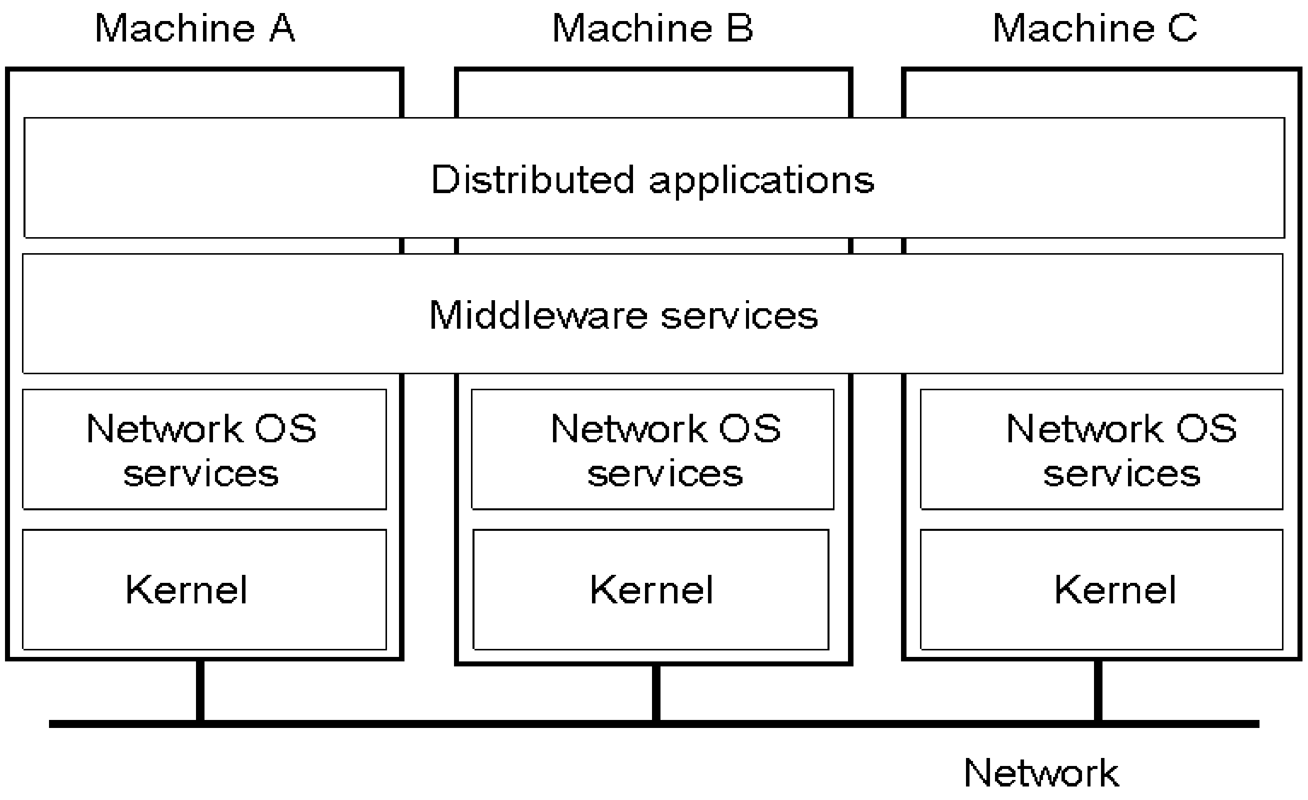
An overview between

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

Software Concepts: Middleware

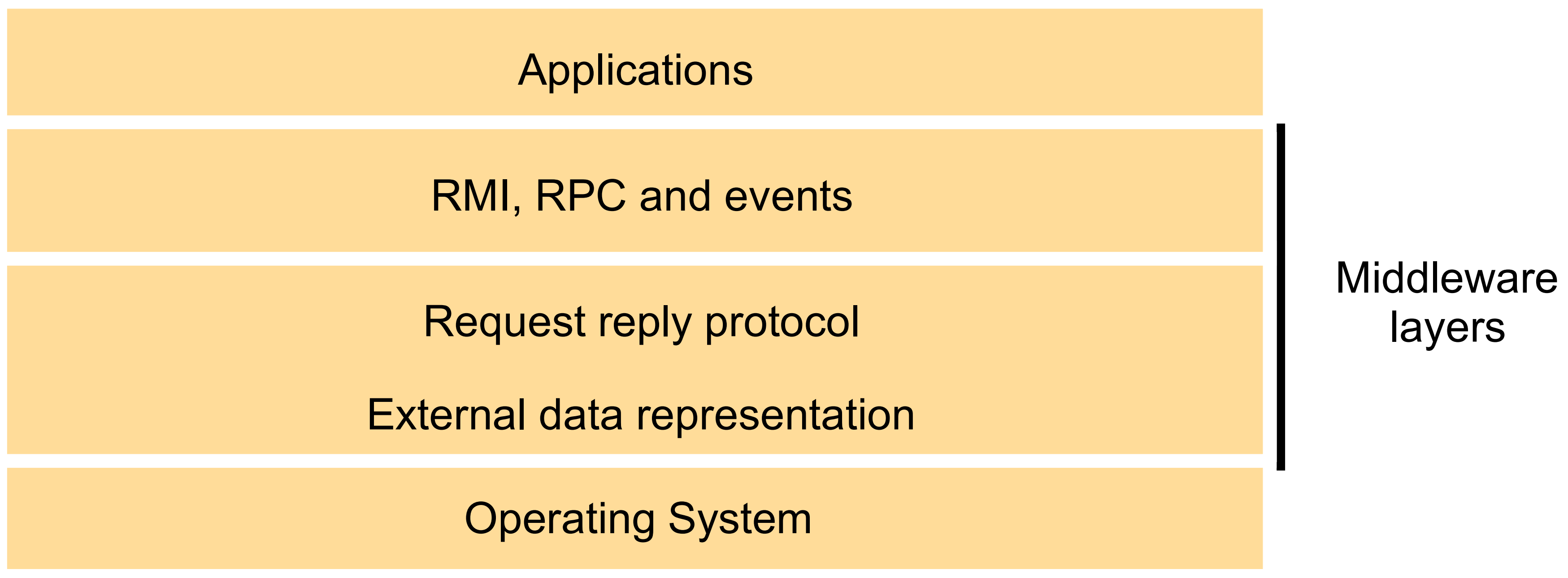
- Middleware: a software layer between applications and the NOS provides a higher level abstraction as well as masking the heterogeneity of the underlying components. Modern DSs are generally built in this way.
- In such a DS, the local OS in each computer manages its resources while the middleware offer a more-or-less complete collection of services used by the applications.
- Most middleware is based on some model (or paradigm) for describing distribution and communication, such as distributed file system, remote procedure call (RPC), distributed object, distributed document etc.

Positioning Middleware



- General structure of a distributed system as middleware.

Middleware layers based on RMI/RPC



Middleware: Services

- Some services common to many middleware systems:

  * high-level communication facilities: to hide the low-level message passing through computer networks, and implement access transparency;

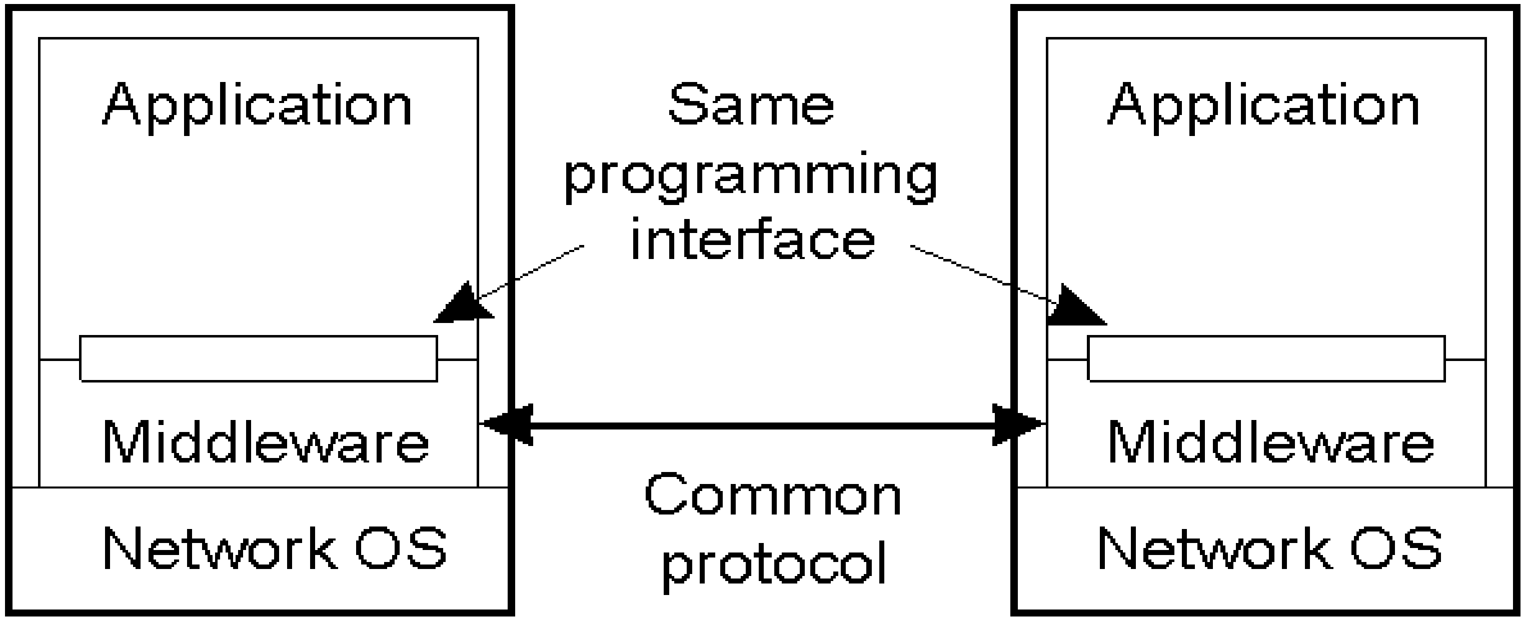
  * naming services: allow entities to be shared and looked up. Scalability could be a big issue;

  * facilities for storage (persistence): It could be offered through a distributed file system, and integrated database into their system in more advanced middleware;

  * facilities for distributed transactions;

  * facilities for security.

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

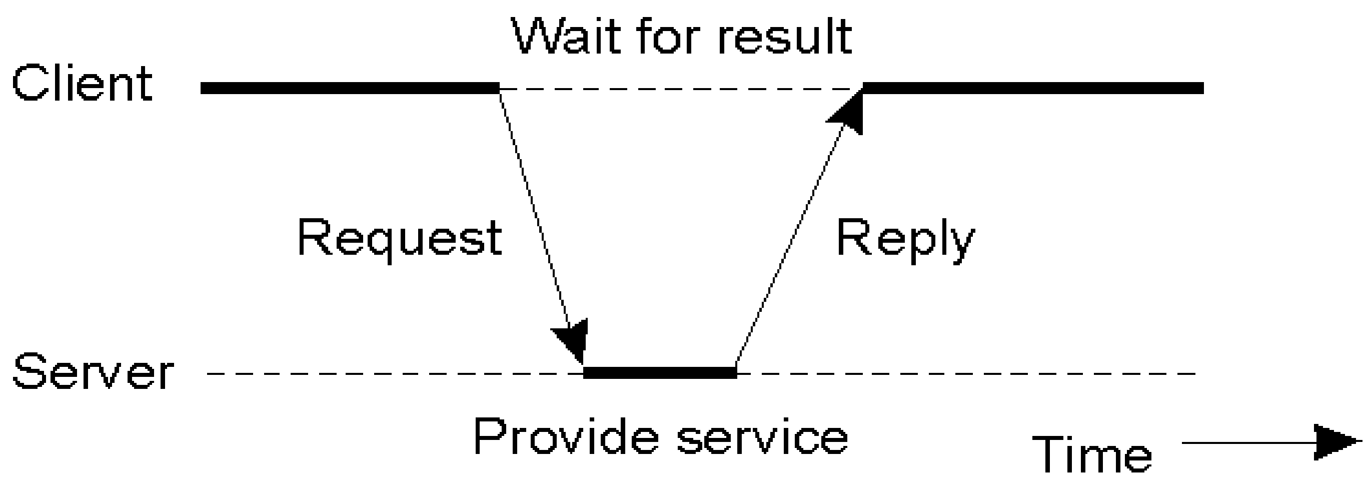
Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global,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

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

The Client-Server Model

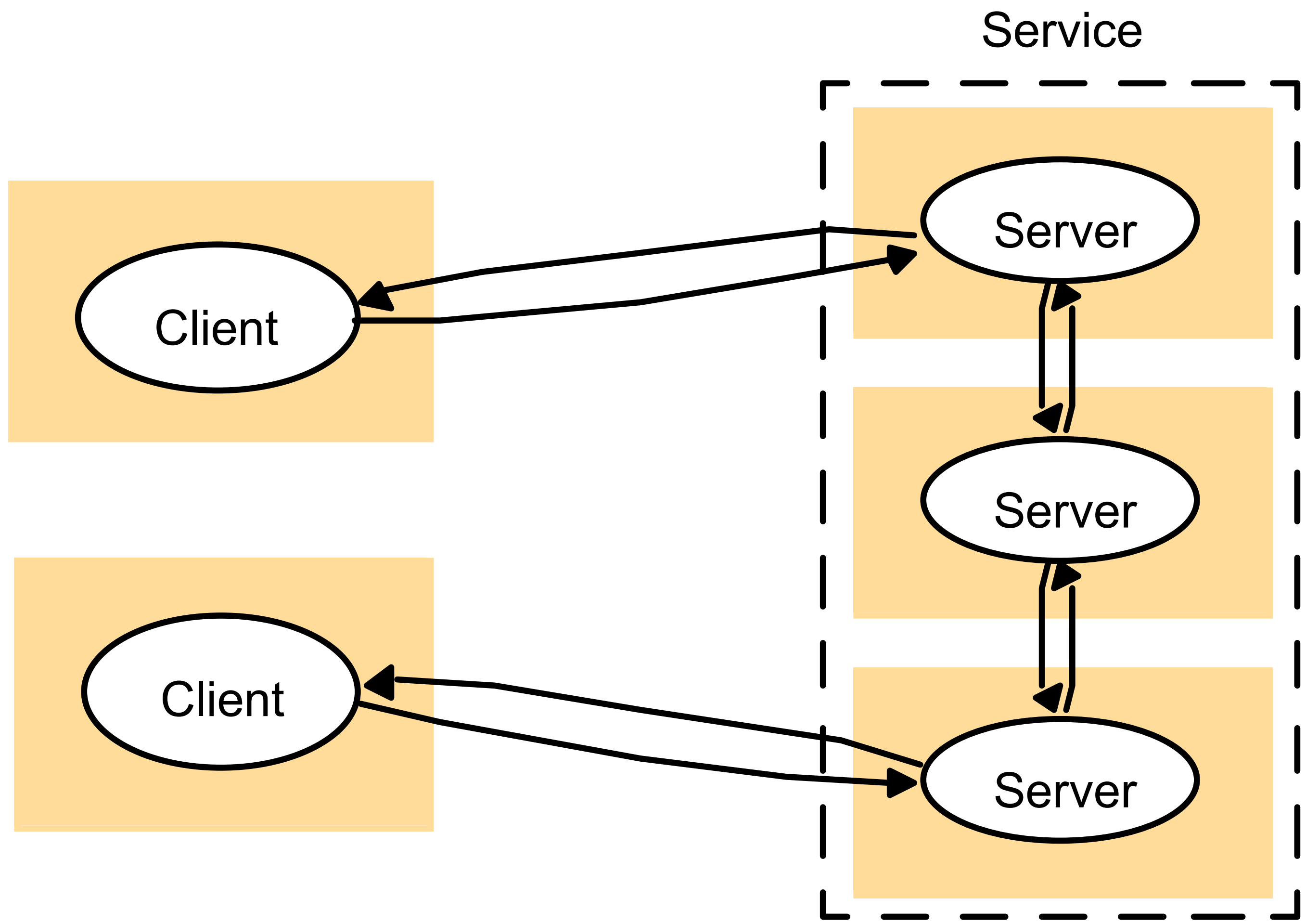
- The shared resources can be accessed through the *resources service*. Services may restrict resource access to a well-defined set of operations.
- Each resource must be managed by program that provides a communication interface enabling the resource to be accessed and updated reliably and consistently.
- *Server* is a running program (a process) in a DS that accepts requests from programs (*clients*) running on other computers to perform a service and responds accordingly. The requests and replies are send in messages.
- Client-server is the major model in DS.

Clients and Servers



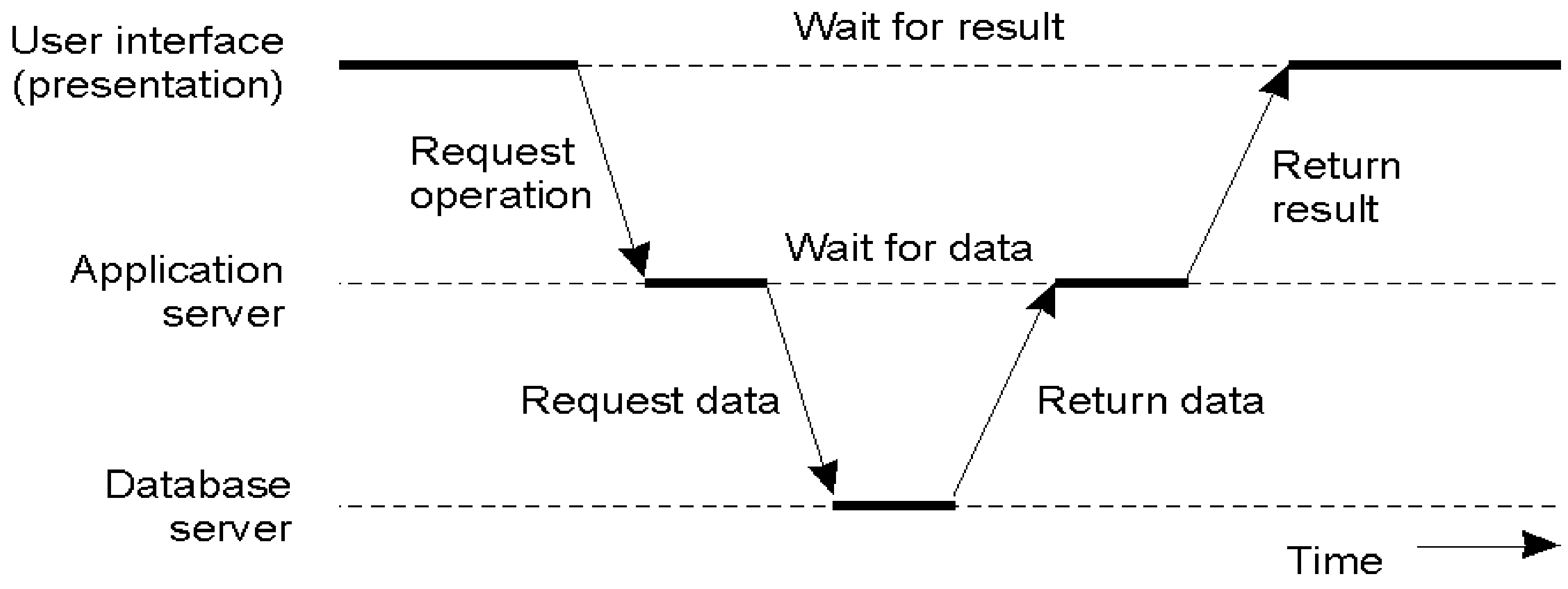
- General interaction between a client and a server.

A service provided by multiple servers



Multitiered Architectures

- An example of a server acting as a client (vertical distribution).



The Client-Server Model: Application Layering

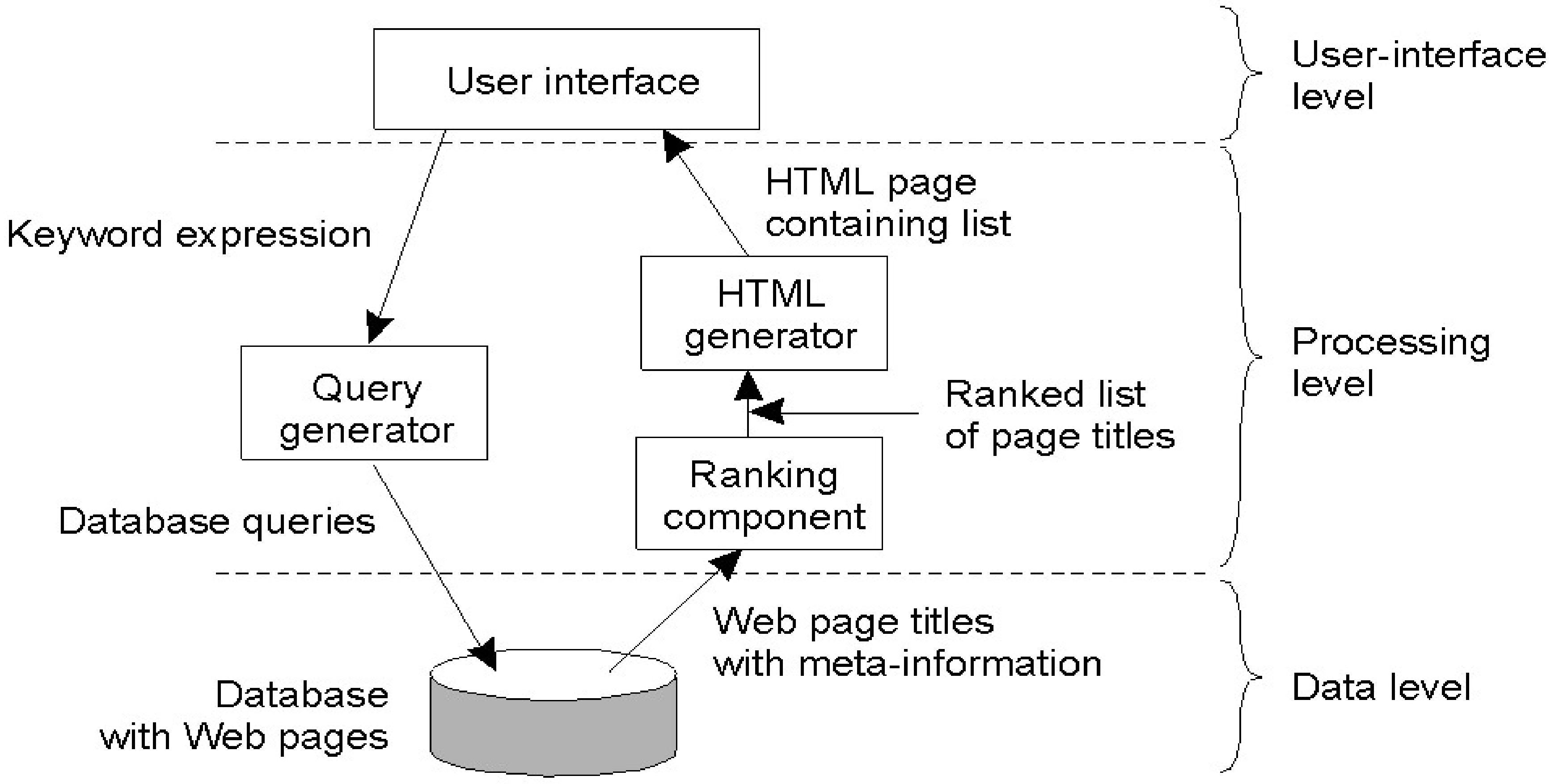
- There is often no clear distinction between a client and a server.
- Many client-server applications are targeted toward supporting user access to database, a distinction can be drawn between the following three levels:

  * user-interface level: contains all that is necessary to directly interface with the user, such as display management;

  * process level: typically contains the applications;

  * data level: contains the actual data being acted on.

Application Layering (search engine in Internet)

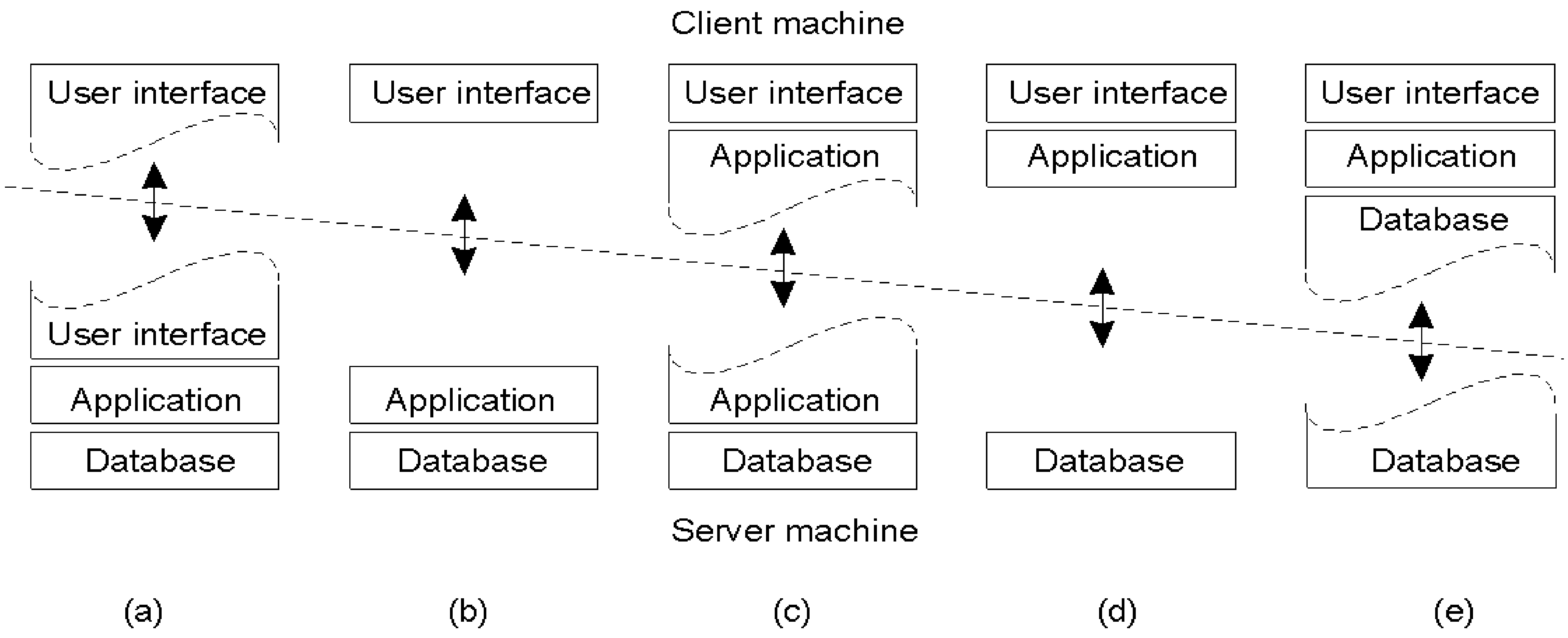


The general organization of an Internet search engine into three different layers

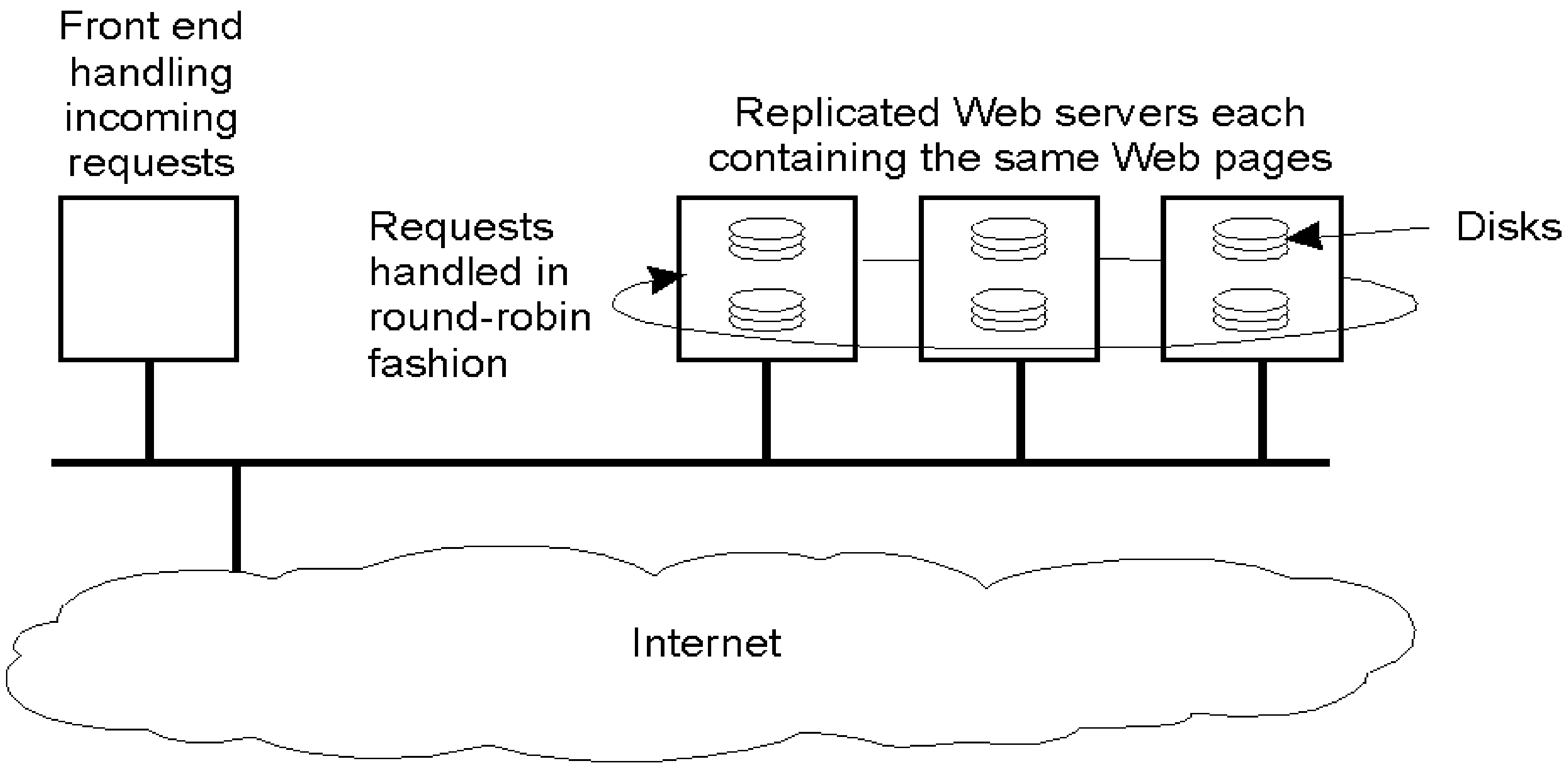
Client-Server Architecture

- The distinction into three logical levels suggests a number of possibilities for physically distributing a client-server application across several machines.
- One approach for organizing clients and servers is to distribute the programs in the application layers across different machines, as shown in the next slide.
- In modern architectures, it often uses horizontal distribution: a client or server may be physically slit up into logically equivalent parts, but each part is operating on its own share of the complete data set, thus balancing the load.
- For simple collaborative applications, there may be no server. A peer process model can be used.

Alternative client-server organizations

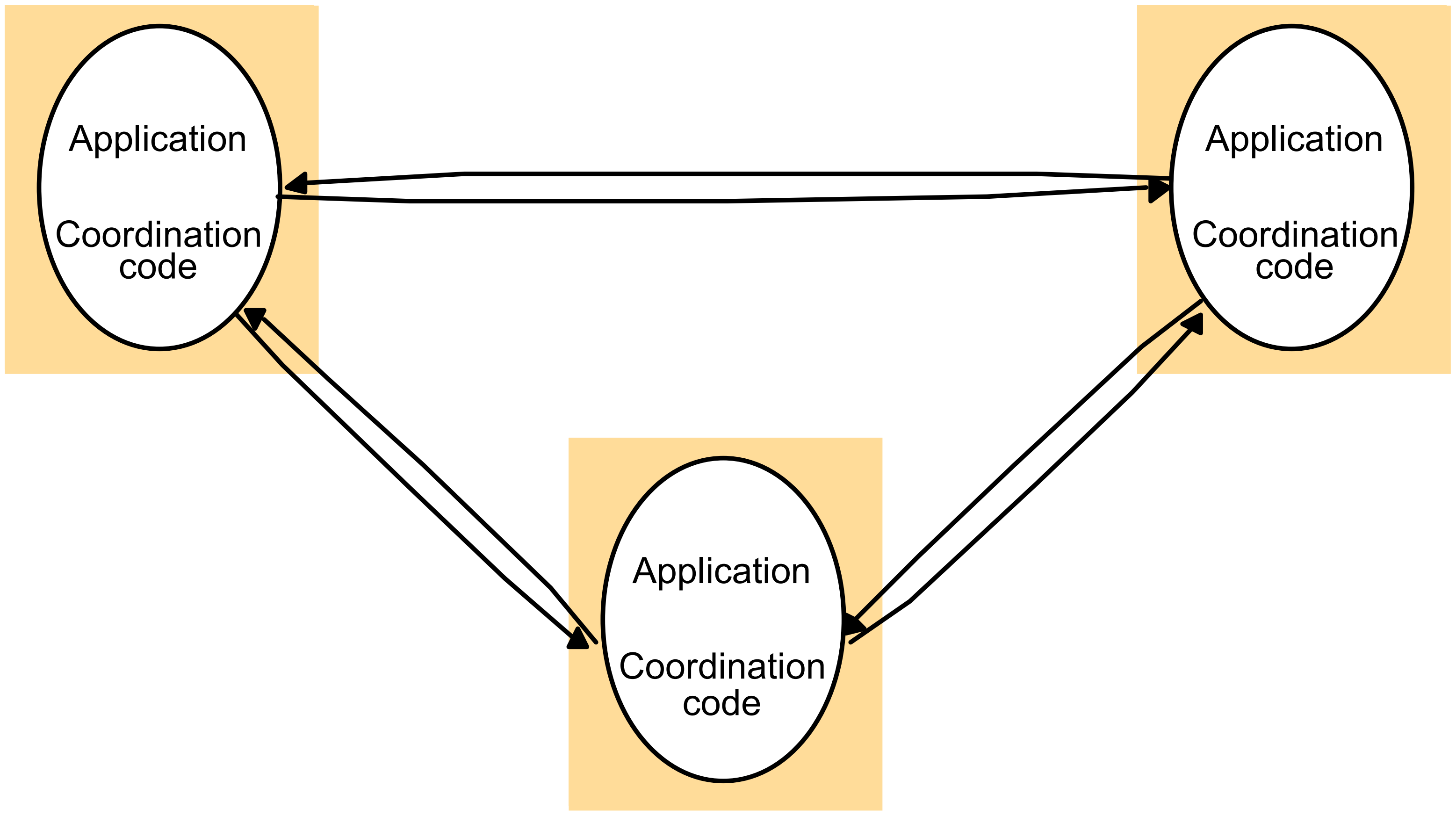


Modern Architectures


Front end handling incoming requests
Replicated Web servers each containing the same Web pages
Requests handled in round-robin fashion
Disks
Internet

- An example of horizontal distribution of a Web service.

A distributed application based on peer processes (not client-server model)



Summary-I

- DSs consist of *autonomous computers* working together to give the *appearance of single coherent system*.
- A DS should easily connect users to resources; It should hide the fact that resources are distributed across network; It should be open and scalable.
- A DOS manages the hardware of multiprocessors and homogeneous multicomputers. It does a good job at providing a single-system view, but does not really support autonomous computers.
- An NOS is good at connecting autonomous computers, so that users can make use of each node's local services, but it does not offer a single-system view.

Summary-II

- Modern DSs are generally built by adding an additional software layer, called *middleware*, atop of an NOS, to hide the heterogeneity and distributed nature of the underlying collection of computers. In such a DS, a specific model, such as distributed file system or distribute object, can be adopted.
- The internal organization of a DS is important. *Client-server model* is a widely applied model in DS. A further refinement in this model is often made by distinguishing a user-interface level, a processing level, and a data level.
- In modern DS, it is desirable to have *horizontal distribution* by which clients and servers are physically distributed and across multiple computers (example: WWW).