DISTRIBUTED SYSTEMS

INTRODUCTION

&

ARCHITECTURES

INTRODUCTION - DISTRIBUTION

- What is Distributed Computing?
- Why Distribute a System?
- Types of Systems
  - ❑ Centralised
  - ❑ Decentralised
  - ❑ Distributed
- Other Computing Paradigms?

- **A system with multiple components**
- **Located on Different Machines**
- **Communicate and Coordinate actions**
- **End User unaware of the Distribution**

DISTRIBUTED SYSTEM: DEFINITION

- ***A Distributed System Is a Collection Of Independent Computers That Appears to its Users as a Single Coherent System***

- **TWO ASPECTS OF THE ABOVE DEFINITION:**
- (1) HARDWARE - Collection of autonomous Computers (**Loosely Coupled**)

  - No global clock

    - Communication is through message passing (**no shared memory**)

- (2) SOFTWARE – Users Think They Are Dealing With A Single System

  For example, it requires that an end user would not be able to tell exactly on which computer a process is currently executing, or even perhaps that part of a task has been spawned off to another process executing somewhere else. Likewise, where data is stored should be of no concern, and neither should it matter that the system may be replicating data to enhance performance. This is called **distribution transparency**

GOALS OF DISTRIBUTED SYSTEMS

- **ALLOW USERS TO ACCESS AND SHARE RESOURCES**
- **TRANSPARENCY**
  - - To hide the fact that its processes and resources are physically distributed across multiple computers
- **OPENNESS**
  - To offer services according to standard rules that describe the syntax and semantics of those services
  - E.G., Specify interfaces using an interface definition language (IDL)
- **SCALABILITY**

GOAL 1: SUPPORTING RESOURCE SHARING

- An important goal of a distributed system is to make it easy for users (and applications) to access and share remote resources. Resources can be virtually anything, but typical examples include peripherals, storage facilities, data, files, services, and networks
- **Reasons to share resources:**
  - One obvious reason is that of **economics**. For example, it is cheaper to have a single high-end reliable storage facility be shared then having to buy and maintain storage for each user separately.
  - Connecting users and resources also makes it **easier to collaborate** and exchange information
- However, resource sharing in distributed systems is perhaps best illustrated by the success of file-sharing peer-to-peer networks like **BitTorrent**. These distributed systems make it extremely simple for users to share files across the internet.

GOAL 2: TRANSPARENCY

- An important goal of a distributed system is to hide the fact that its processes and resources are physically distributed across multiple computers, possibly separated by large distances.
- It tries to make the distribution of processes and resources **transparent**, that is, invisible, to end users and applications.
- Name the different transparencies.
- What do you mean by concurrency transparency
- Why is it difficult to mask a failure
- Difference between Migration transparency and Relocation transparency

TRANSPARENCY

Transparency	Description
Access	Hides differences in data representation and invocation mechanisms
Location	Hides where an object resides
Migration	Hides from an object the ability of a system to change that object's location
Relocation	Hides from a client the ability of a system to change the location of an object to which the client is bound
Replication	Hides the fact that an object or its state may be replicated and that replicas reside at different locations
Concurrency	Hides the coordination of activities between objects to achieve consistency at a higher level
Failure	Hides failure and possible recovery of objects
Persistence	Hides the fact that an object may be (partly) passivated by the system

EXTRA: DEGREE OF TRANSPARENCY

- **OBSERVATION:** Aiming at full transparency may be too much:
- Users may be located in different continents; distribution is apparent and not something you want to hide
- Completely hiding failures of networks and nodes is (theoretically and practically) impossible
  - You cannot distinguish a slow computer from a failing one (
  - You can never be sure that a server actually performed an operation before a crash
- Full transparency will cost performance, exposing distribution of the system
  - Keeping web caches exactly up-to-date with the master copy
  - Immediately flushing write operations to disk for fault tolerance

GOAL 3: OPENNESS

- **Open distributed system:** An **open distributed system** Is essentially a system that offers components that can easily be used by, or Integrated into other systems.
- *To be open means that components should **adhere to standard rules that describe the Syntax and semantics** of what those components have to offer* (i.e., which service they provide)
- Able to interact with services from other open systems, irrespective of the underlying environment requires the following characteristics:
  - Systems should conform to well-defined **interfaces**
  - Systems should support **portability** of applications
  - Systems should easily **interoperate**

OPENNESS OF DISTRIBUTED SYSTEMS

- Define services through **interfaces** using an **interface definition language** (**IDL**).
- If properly specified, an interface definition allows an arbitrary process that needs a certain interface, to talk to another process that provides that interface. It also allows two independent parties to build completely different implementations of those interfaces, leading to two separate components that operate in exactly the same way.
- **Interoperability** characterizes the extent by which two implementations of systems or components from different manufacturers can co-exist and work together by merely relying on each other's services as specified by a common standard.
- **Portability** characterizes to what extent an application developed for a distributed system A can be executed, without modification, on a different distributed system B that implements the same interfaces as A.
- **Achieving openness:** at least make the distributed system independent from **heterogeneity** of the underlying environment: Hardware Platforms Languages

POLICIES VERSUS MECHANISMS EXTRA

- **Implementing openness:** Requires support for different **policies** specified by applications and users:
  - What level of consistency do we require for client cached data?
  - Which operations do we allow downloaded code to perform?
  - Which QoS requirements do we adjust in the face of varying bandwidth?
  - What level of secrecy do we require for communication?
- **Implementing openness:** Ideally, a distributed system provides only **mechanisms**:
  - Allow (dynamic) setting of caching policies, preferably per cachable item
  - Support different levels of trust for mobile code
  - Provide adjustable QoS parameters per data stream
  - Offer different encryption algorithms

GOAL 4: SCALABILITY

- **Observation:** many developers of modern distributed systems easily use the adjective "scalable" without making clear *why/how* their system actually scales.
- **Scalability Dimensions:** Scalability of a system can be measured along at least three different dimensions :
  - Number of users and/or processes **(size scalability)**
    - *Size scalability* A system can be scalable with respect to its size, meaning that we can easily add more users and resources to the system without any noticeable loss of performance.
  - Maximum distance between nodes **(geographical scalability)**

    A geographically scalable system is one in which the users and resources may lie far apart, but the fact that communication delays may be significant is hardly noticed.
  - Number of administrative domains **(administrative scalability)**

    An administratively scalable system is one that can still be easily managed even if it spans many independent administrative organizations.

SCALABILITY LIMITATIONS (SIZE SCALABILITY)

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

- The problem with the centralized scheme is obvious: the server, or group of servers, can simply become **a bottleneck** when it needs to process an increasing number of requests.
- There are essentially three root causes for becoming a bottleneck:
  - The computational capacity, limited by the CPUs
  - The storage capacity, including the transfer rate between CPUs and disks
  - The network between the user and the centralized service
- Another problem is that it could be the service being a **single point of failure**

GEOGRAPHICAL SCALABILITY - LIMITATIONS

- GEOGRAPHICAL SCALABILITY HAS ITS OWN PROBLEMS:

- **Synchronous communication**: One of the main reasons why it is still difficult to scale existing distributed systems that were designed for local-area networks is that many of them are based on **synchronous communication**. In this form of communication, a party requesting service, generally referred to as a **client**, blocks until a reply is sent back from the **server** implementing the service.
- Another problem that hinders geographical scalability is that communication in wide-area networks is **inherently much less reliable** than in local-area networks
- Yet another issue that pops up when components lie far apart is the fact that wide area systems generally have only very limited facilities for multipoint communication. In contrast, local-area networks often support **efficient broadcasting mechanisms.** For Ex: Group Communication can be implemented effortlessly in a LAN using broadcasting, the concept which may not work in a WAN

ADMINISTRATIVE SCALABILITY - LIMITATIONS

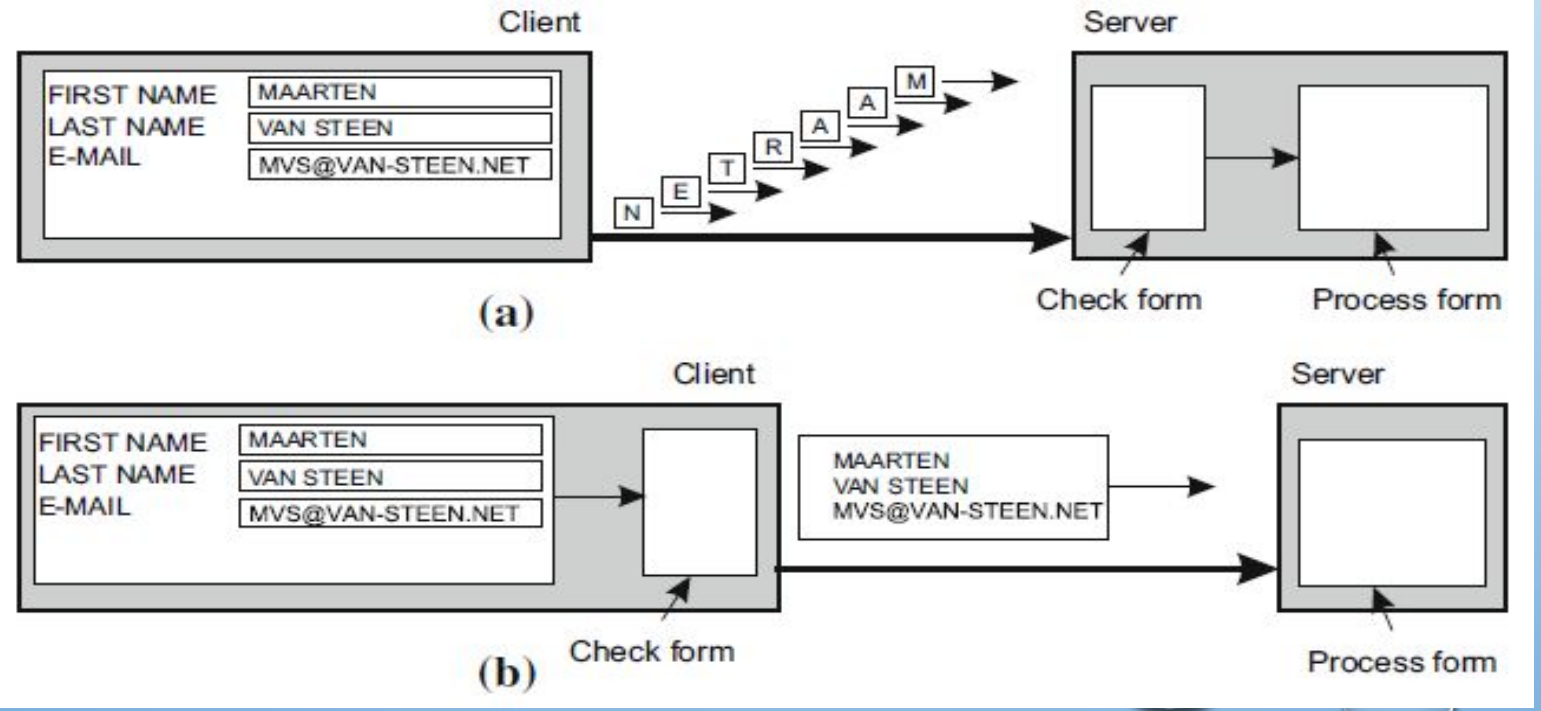
- Question is how to scale a distributed system across multiple, independent administrative domains
- A major problem that needs to be solved is that of **conflicting policies** with respect to resource usage (and payment), management, and security.
- Many components of a distributed system that reside within a single domain can often be **trusted by users** that operate within that same domain. System administration may have tested and certified applications, and may have taken special measures to ensure that such components cannot be tampered with. In essence, the users trust their system administrators. However, this trust does not expand naturally across domain boundaries.
- If a distributed system expands to another domain, **two types of security measures need to be taken**. First, the distributed system has to **protect itself against malicious attacks from the new domain**. For example, users from the new domain may have only read access to the file system in its original domain or expensive imagesetters or high-performance computers may not be made available to unauthorized users. Second, the **new domain has to protect itself against malicious attacks** from the distributed system.

TECHNIQUES FOR SCALING

- Scalability problems in distributed systems appear as performance problems caused by limited capacity of servers and network. Simply improving their capacity (e.g., By increasing memory, upgrading CPUs, or replacing network modules) is often a solution, referred to as **scaling up**. When it comes to **scaling out**, that is, expanding the distributed system by essentially deploying more machines, there are basically only three techniques we can apply: hiding communication latencies, distribution of work, and replication.
- **Hiding Communication Latencies. Ex:** Asynchronous Communication
- Hiding communication latencies is applicable in the case of geographical scalability. The basic idea is simple: try to avoid waiting for responses to remote-service requests as much as possible. This means constructing the requesting application in such a way that it uses only **asynchronous communication**.
- Asynchronous communication can often be used in batch-processing systems and parallel applications
- OR a new thread of control can be started to perform the request. Although it blocks waiting for the reply, other threads in the process can continue.

Hiding Communication Latencies

- However, there are many applications that **cannot make effective use of asynchronous communication**. For example, in interactive applications when a user sends a request and have nothing better to do than to wait for the answer.
- A much better solution is to reduce the overall communication, for example, by moving part of the computation that is normally done at the server to the client process requesting the service.



Distribution: Scaling Technique

Partition data and computations across multiple machines:

- Move computations to clients (java applets)
- Decentralized naming services (DNS)
- Decentralized information systems (WWW)

- A good example of partition and distribution is the internet domain name system (DNS). The DNS name space is hierarchically organized into a tree of **domains**, which are divided into nonoverlapping **zones**

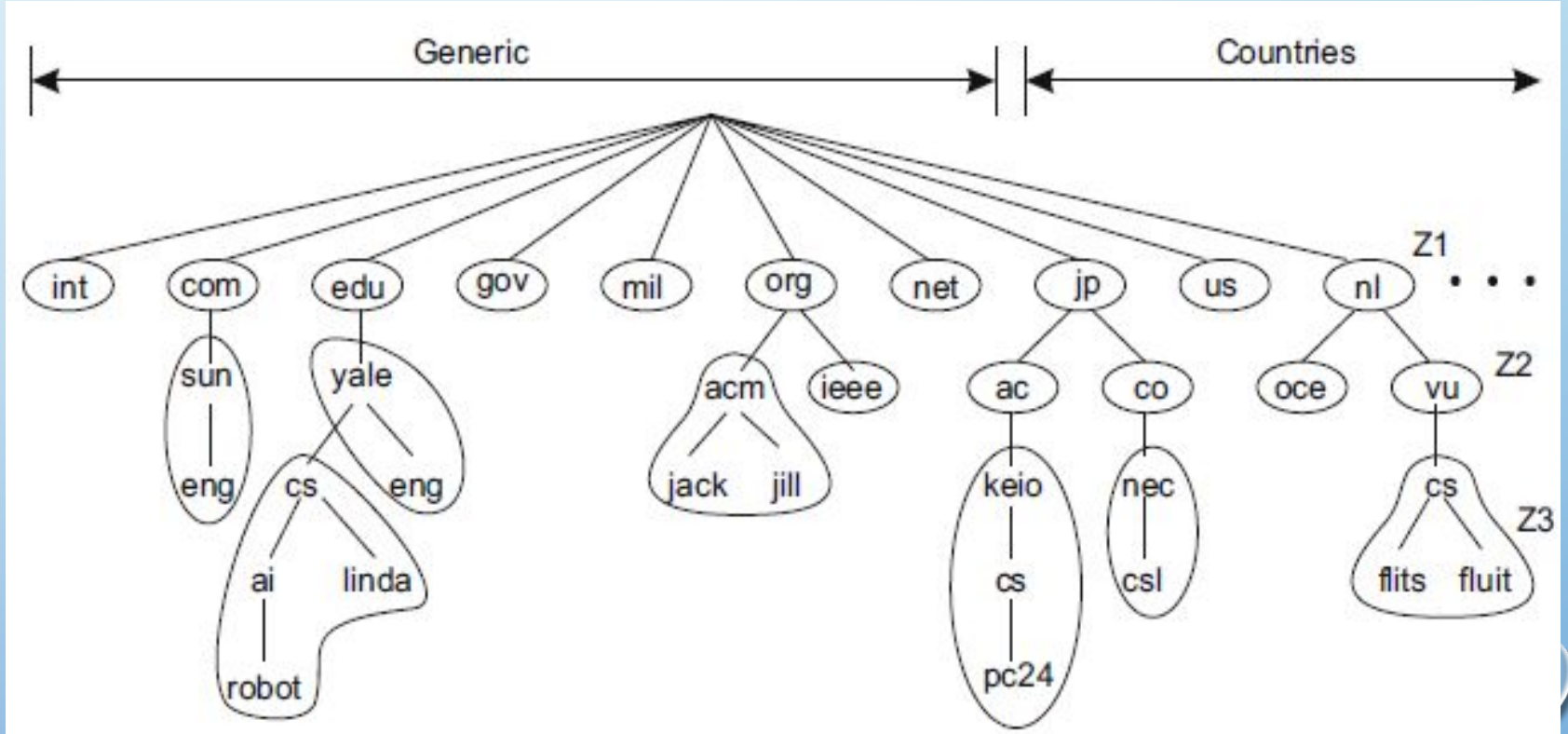


Fig. 4 An example of dividing the (original) DNS name space into zones

Replication & Caching

- Scalability problems often appear in the form of performance degradation, it is generally a good idea to actually replicate components across a distributed system. Replication not only **increases availability and reliability**, but also helps to **balance the load between components** leading to **better performance**. Also, having a copy nearby can hide much of the communication latency problems.
- Make copies of data available at different machines:
  - Replicated file servers (mainly for fault tolerance)
  - Replicated databases
  - Mirrored web sites
  - Large-scale distributed shared memory systems
- **Caching is a special form of replication, caching results in making a copy of a resource, generally in the proximity of the client accessing that resource.**
- Allow client processes to access local copies:
  - Web caches (browser/web proxy)
  - File caching (at server and client)
- There is one serious drawback to caching and replication that may adversely affect scalability. Because we now have multiple copies of a resource, modifying one copy makes that copy different from the others. Consequently, caching and replication leads to consistency problems.

MODELING DISTRIBUTED SYSTEMS

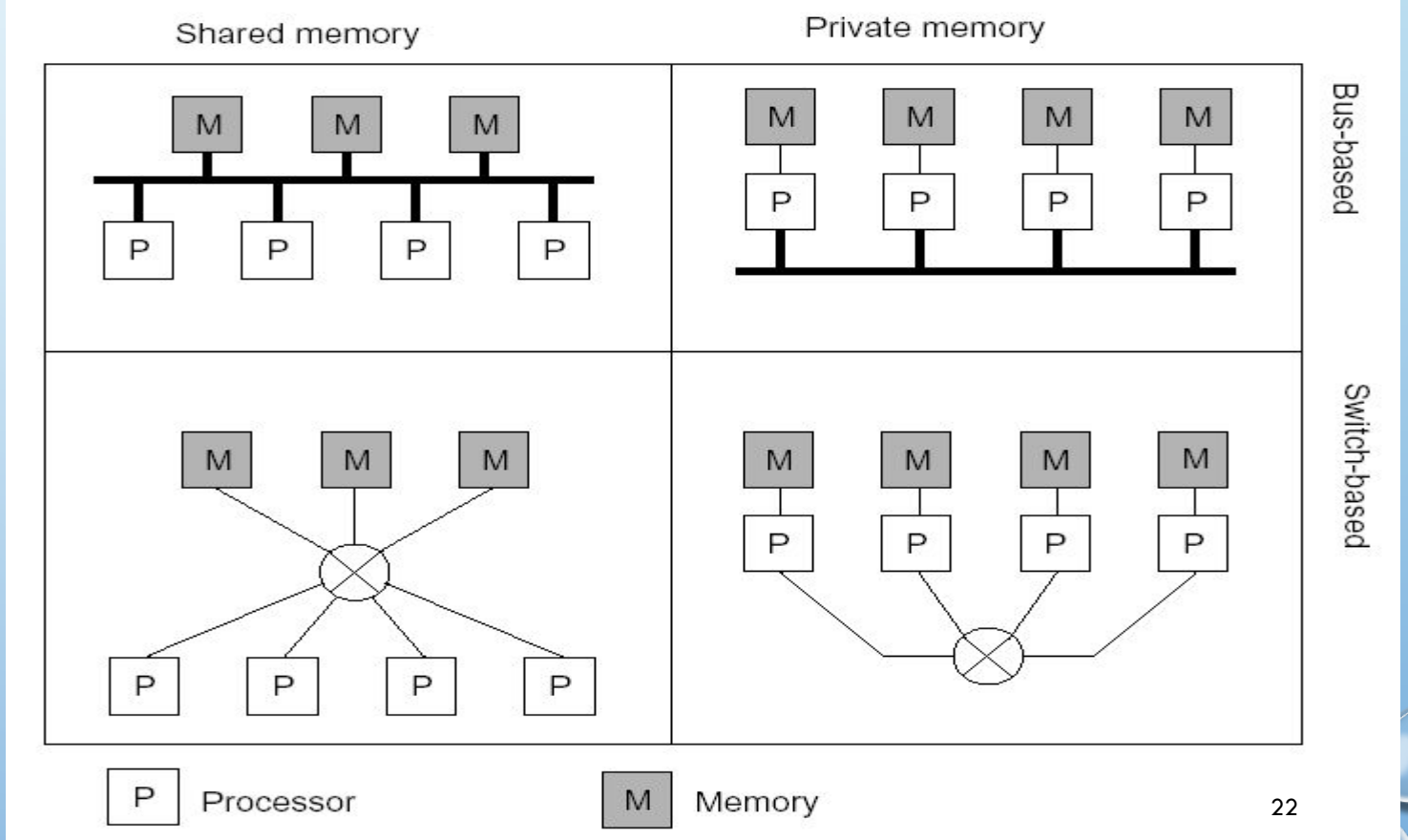
- When building distributed applications, system builders have often looked to the non-distributed systems world for models to follow
- Consequently, distributed systems tend to exhibit certain characteristics that are already familiar to us.
- This applies equally to hardware concepts as it does to software concepts.

DISTRIBUTED SYSTEMS: HARDWARE CONCEPTS

- Multiprocessors
- Multicomputers
- Networks of computers

MULTIPROCESSORS AND MULTICOMPUTERS

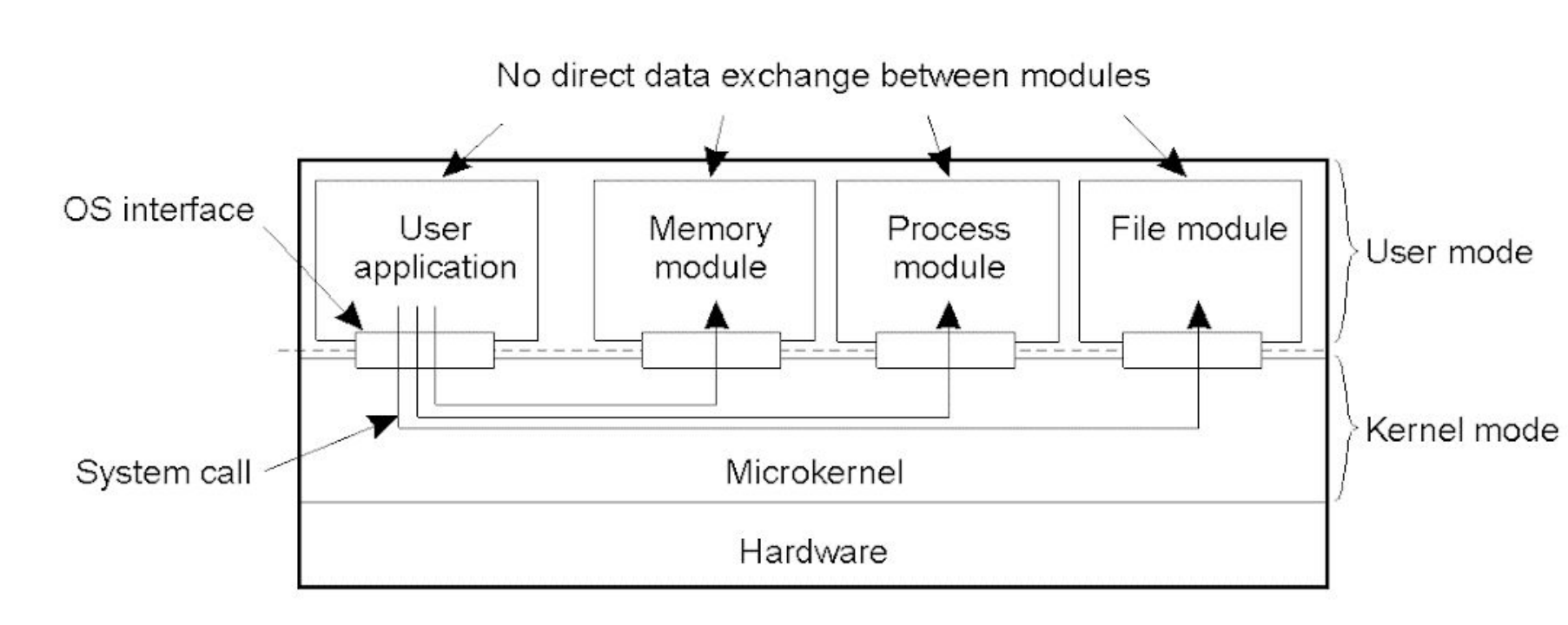
- **Distinguishing features:**
- Private versus shared memory
- Bus versus switched interconnection



NETWORKS OF COMPUTERS

- **High degree of node heterogeneity:**
  - High-performance parallel systems (multiprocessors as well as multicomputers)
  - High-end pcs and workstations (servers)
  - Simple network computers (offer users only network access)
  - Mobile computers (palmtops, laptops)
- **High degree of network heterogeneity:**
  - Local-area gigabit networks
  - Wireless connections
  - Long-haul, high-latency POTS connections
  - Wide-area switched megabit connections
- **Ideally, a distributed system hides these differences**

UNIPROCESSOR OPERATING SYSTEMS



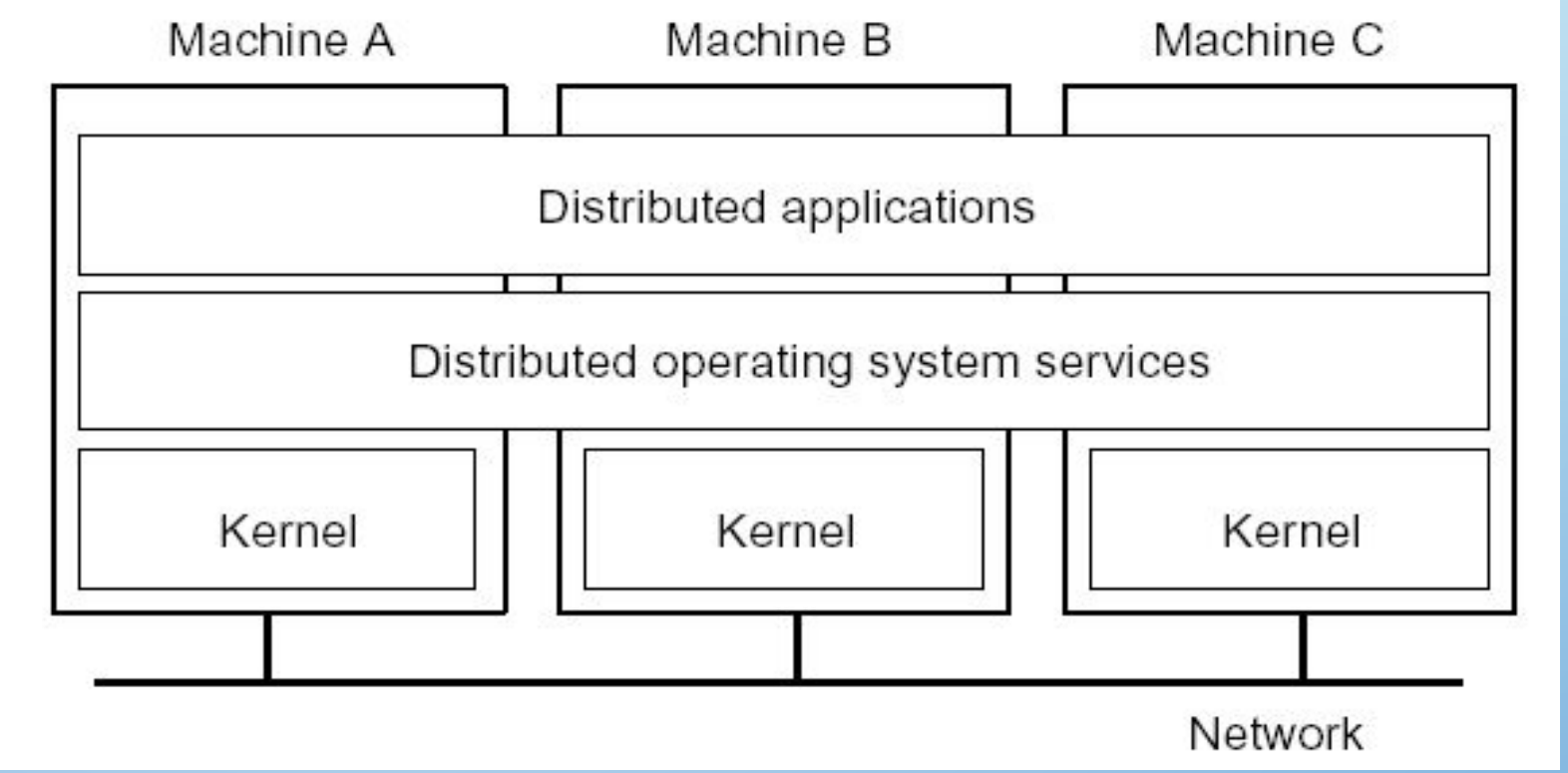
Separating applications from operating system code through a "microkernel" – can provide a good base upon which to build a distributed OS (DOS)

DISTRIBUTED SYSTEMS: SOFTWARE CONCEPTS

- Distributed operating system
- Network operating system
- Middleware

DISTRIBUTED OPERATING SYSTEM

- **Some characteristics:**
- OS on each computer knows about the other computers
- OS on different computers generally the same
- Services are generally (transparently) distributed across computers



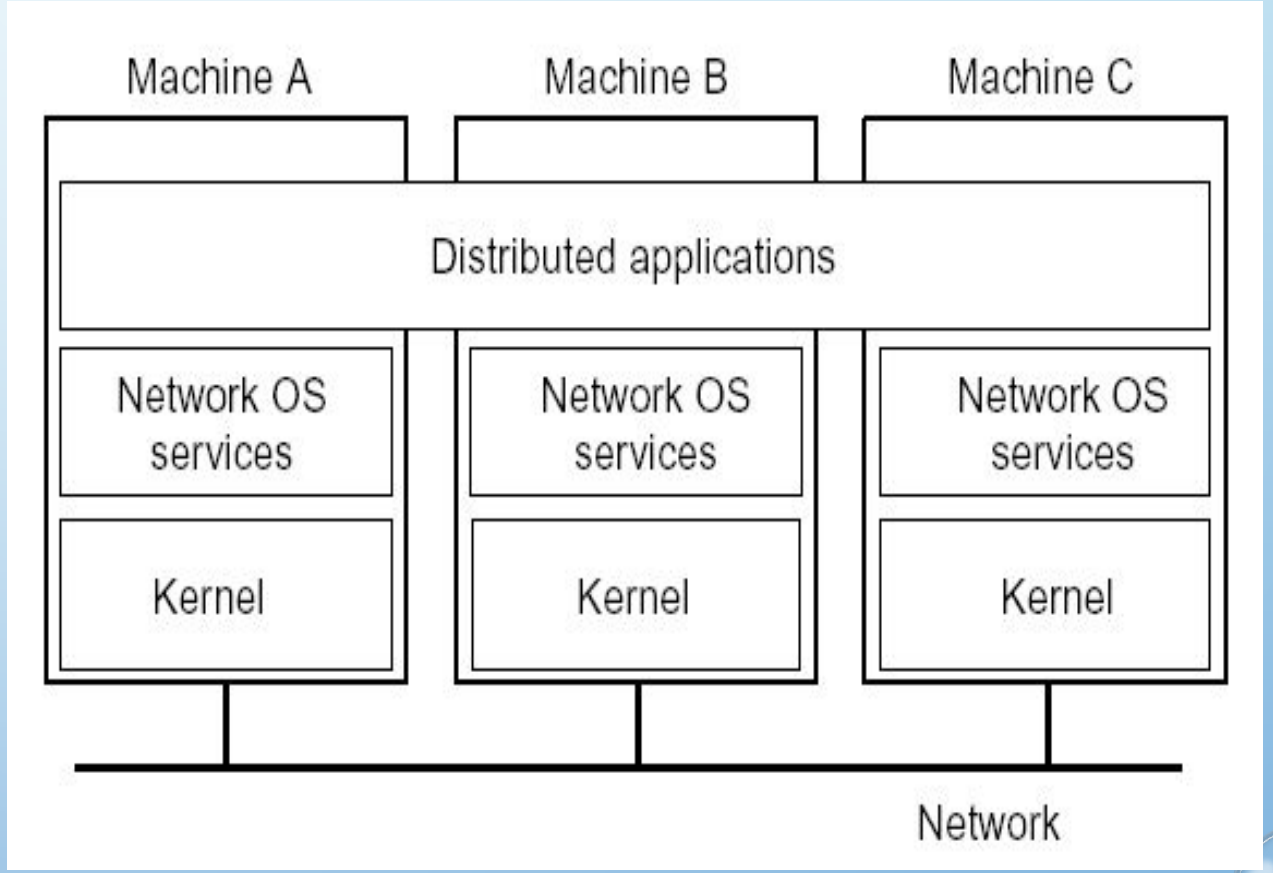
MULTICOMPUTER OPERATING SYSTEM

Harder than traditional (multiprocessor) OS: because memory is not shared, emphasis shifts to processor communication by message passing:

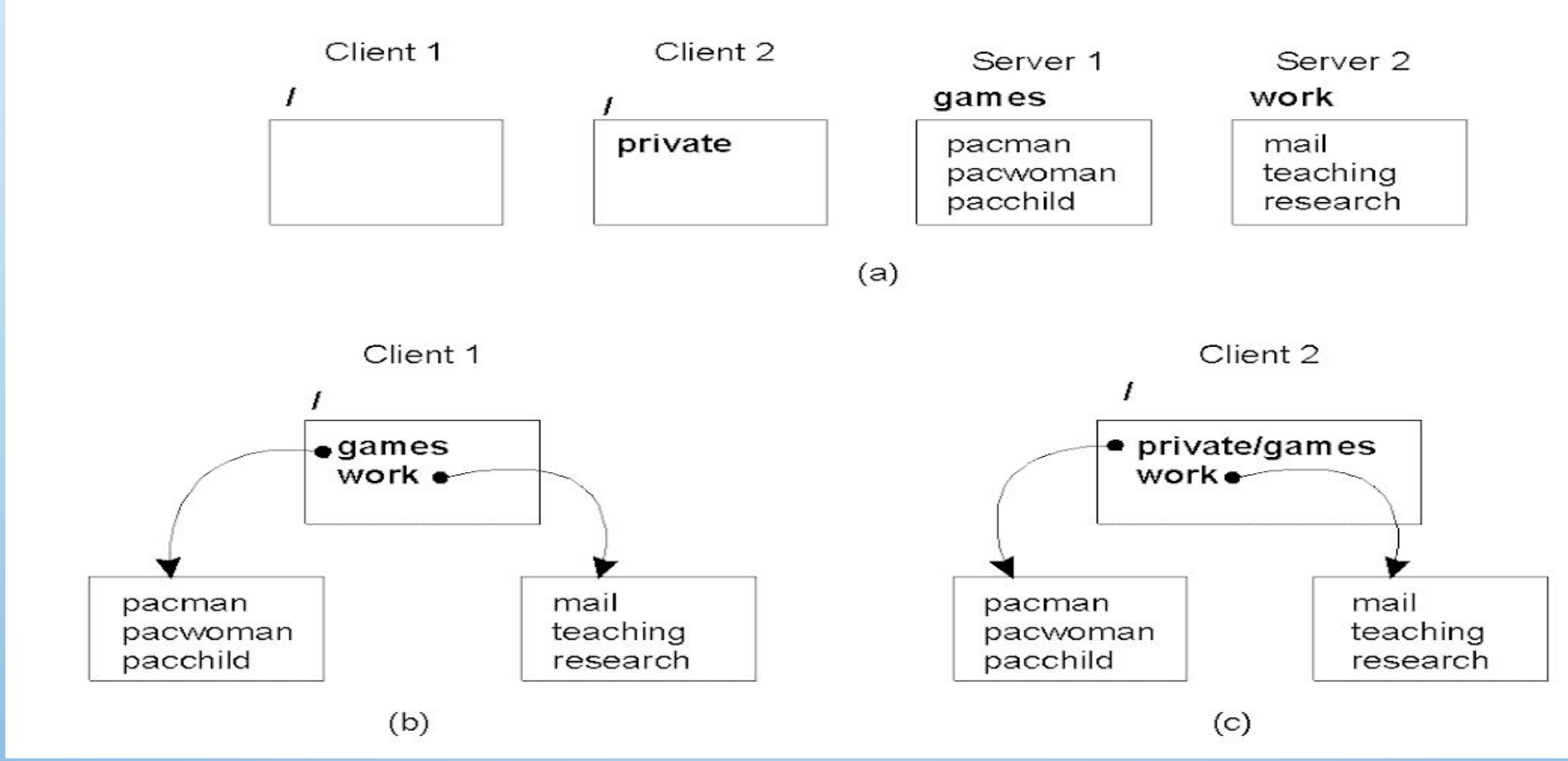
- Often no simple global communication:
  - Only bus-based multicomputers provide hardware broadcasting
  - Efficient broadcasting may require network interface programming techniques
- No simple system-wide synchronization mechanisms
- Virtual (distributed) shared memory requires OS to maintain global memory map in software
- Inherent distributed resource management: no central point where allocation decisions can be made
- **Practice:** only very few truly multicomputer operating systems exist (example: amoeba)

NETWORK OPERATING SYSTEM

- **SOME CHARACTERISTICS:**
- Each computer has its own operating system with networking facilities
- Computers work independently (i.E., They may even have different operating systems)
- Services are tied to individual nodes (ftp, telnet,www)
- Highly file oriented (basically, processors share *only* files)



NETWORK OPERATING SYSTEM (2)



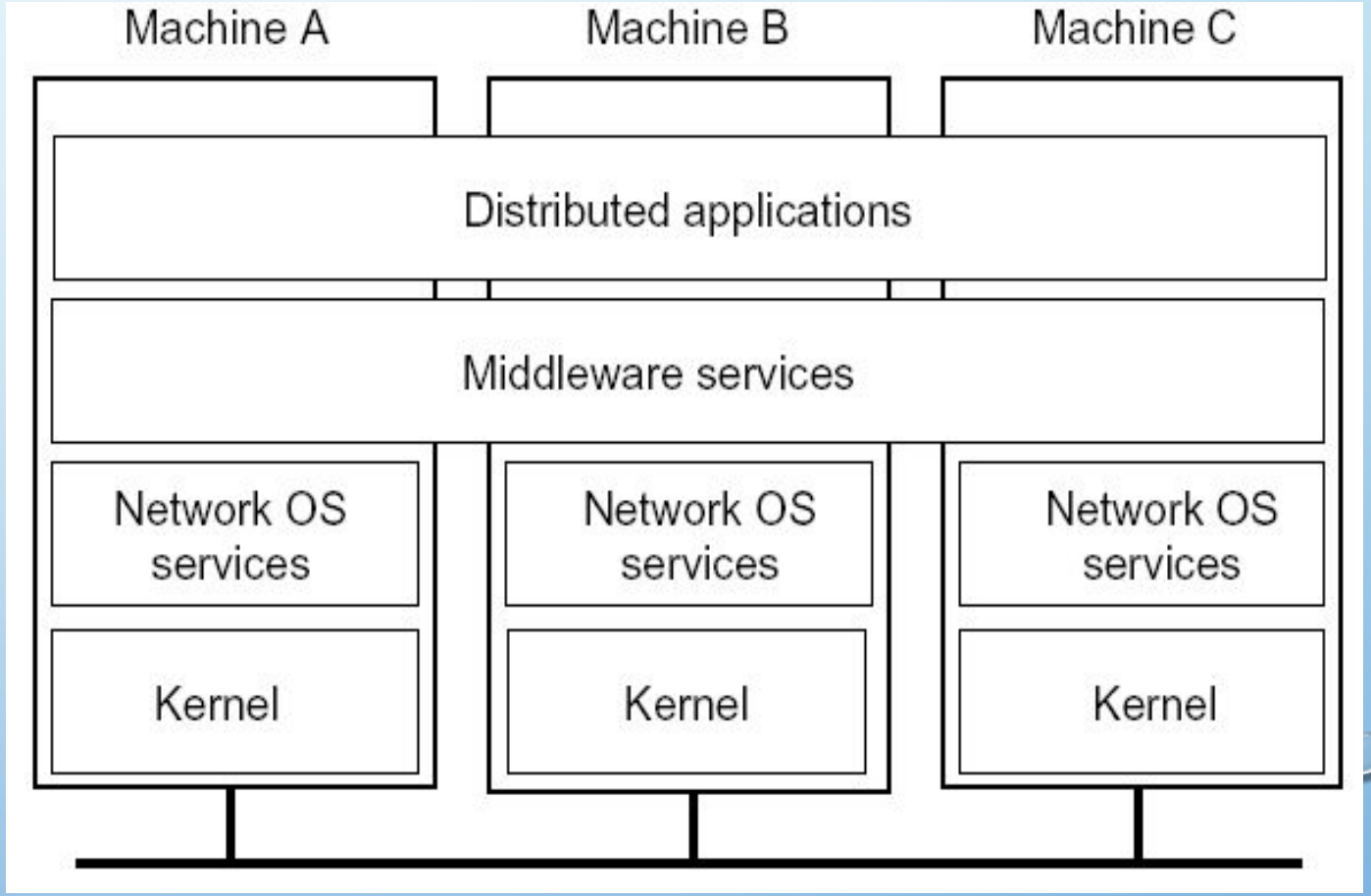
Different clients may mount the servers in different places – difficult to maintain a consistent "view" of the system

THE BEST OF BOTH WORLDS - MIDDLEWARE

- DOS: Too Inflexible (All Systems Of The Same Type).
- Nos: Too Primitive (Lowest Common Denominator – Too Much Diversity).
- “Middleware” – Best Possible Compromise
- Middleware = Nos + Additional Software Layer.

MIDDLEWARE

- **SOME CHARACTERISTICS:**
- OS on each computer need not know about the other computers
- OS on different computers need not generally be the same
- Services are generally (transparently) distributed across computers



DISTRIBUTED SYSTEMS: SOFTWARE CONCEPTS

- Distributed operating system
- Network operating system
- Middleware

System	Description	Main goal
DOS	Tightly-coupled OS for multiprocessors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled OS 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

B.E 2022
32

NEED FOR MIDDLEWARE

- **Motivation: T**oo many networked applications were difficult to integrate:
  - Departments are running different NOSs
  - Integration and interoperability only at level of primitive NOS services
  - Need for federated information systems:
    - – Combining different databases, but providing a single view to applications
    - – Setting up enterprise-wide internet services, making use of existing information systems
    - – Allow transactions across different databases
    - – Allow extensibility for future services (e.G., Mobility, teleworking, collaborative applications)
  - Constraint: use the existing operating systems, and treat them as the underlying environment

MIDDLEWARE AND DISTRIBUTED SYSTEMS

- Distributed systems are often organized to have a separate layer of software that is logically placed on top of the respective operating systems of the computers that are part of the system. This organization is known as **middleware.**
- Middleware is the same to a distributed system as what an operating system is to a computer: a ***manager of resources*** offering its applications to efficiently share and deploy those resources across a network
- Next to resource management, it offers services including:
  - ***Facilities for inter-application communication.***
  - ***Security services.***
  - ***Accounting services.***
  - ***Masking of and recovery from failures.***

MIDDLEWARE SERVICES (1/3)

- **COMMUNICATION SERVICES:** Abandon primitive socket-based message passing in favor of:
  - Procedure calls across networks **RPC**
  - Remote-object method invocation **RMI**
  - Message-queuing systems **MOM**
  - Advanced communication through **streams**
  - Event notification service
- **INFORMATION SYSTEM SERVICES:** Services that help manage data in a distributed system:
  - Large-scale, system-wide **naming services**
  - Advanced **directory services** (search engines)
  - Location services for **tracking mobile objects**
  - **Persistent storage** facilities
  - Data **caching and replication**

MIDDLEWARE SERVICES (2/3)

- **CONTROL SERVICES:** Services giving applications control over when, where, and how they access data:
  - Middleware generally offers special support for executing services in multiple machines in an all-or-nothing fashion, commonly referred to as an **atomic transaction**.
  - Distributed transaction processing
  - Code migration

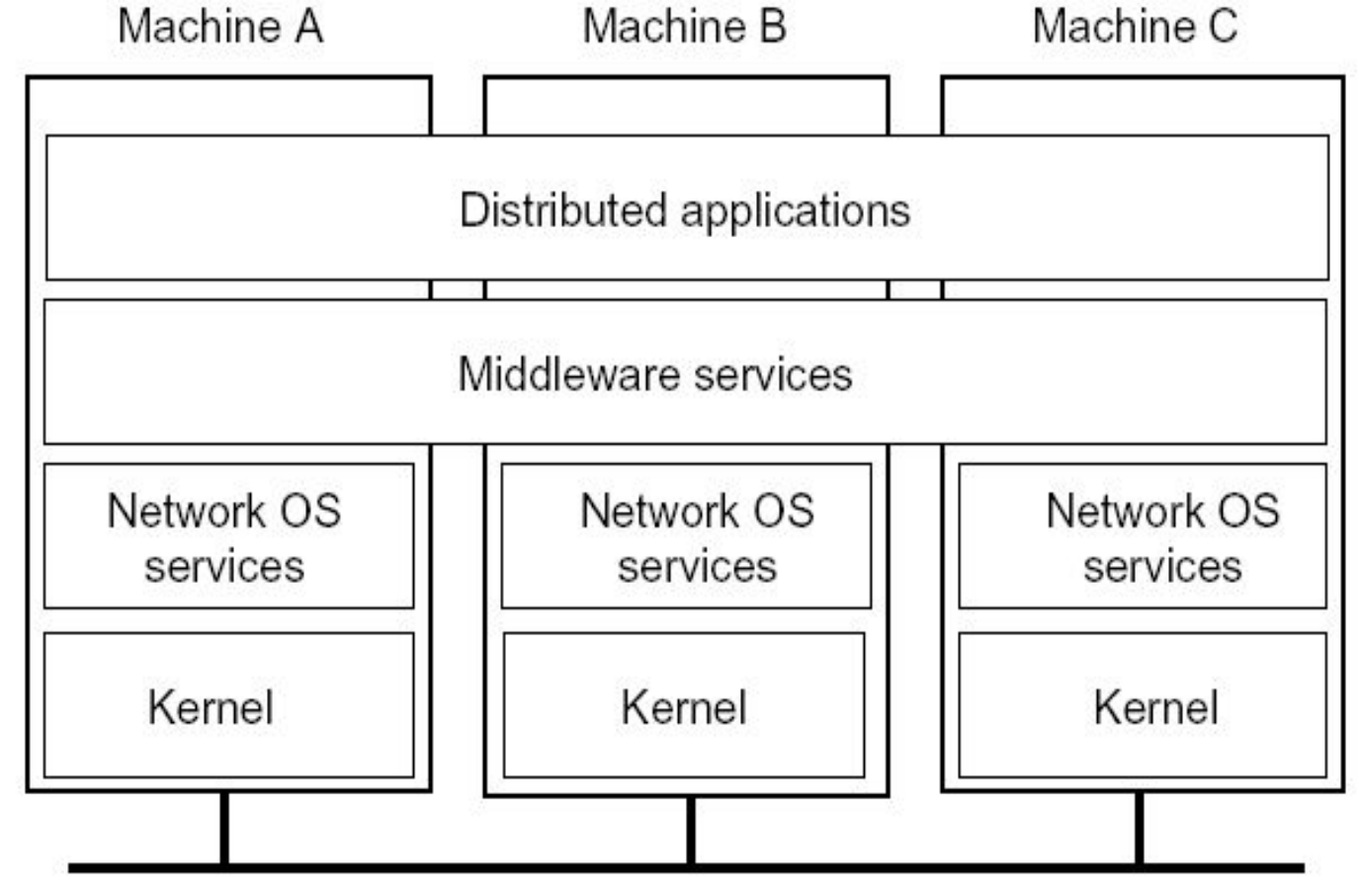
- **SECURITY SERVICES:** Services for secure processing and communication:
  - Authentication and authorization services
  - Simple encryption services
  - Auditing service

MIDDLEWARE SERVICES (3/3)

- ***SERVICE COMPOSITION:*** It is common to develop new applications by taking existing programs and gluing them together. This is notably the case for many web-based applications known as **web services**. Web-based middleware can help by standardizing the way web services are accessed and providing the means to generate their functions in a specific order. An example of how service composition is deployed is by **mashups**: web pages that combine and aggregate data from different sources. Well-known mashups are those based on google maps in which maps are enhanced with extra information such as trip planners or real-time weather forecasts.

- ***RELIABILITY*** **-**Providing enhanced functions for building reliable distributed applications. It allows a developer to build an application as a group of processes such that any message sent by one process is **guaranteed to be received by all** or no other process. Such guarantees can greatly simplify developing distributed applications

MIDDLEWARE


Machine A
Machine B
Machine C
Distributed applications
Middleware services
Network OS services
Network OS services
Network OS services
Kernel
Kernel
Kernel

COMPARISON OF DOS, NOS, AND MIDDLEWARE

Comparison Of DOS, NOS, And Middleware

1: Degree of transparency
2: Same operating system on each node?
3: Number of copies of the operating system
4: Basis for communication
5: How are resources managed?
6: Is the system easy to scale?
7: How open is the system?

Item	Distributed OS		Network OS	Middle-ware DS
	multiproc.	multicomp.		
1	Very High	High	Low	High
2	Yes	Yes	No	No
3	1	N	N	N
4	Shared memory	Messages	Files	Model specific
5	Global, central	Global, distributed	Per node	Per node
6	No	Moderately	Yes	Varies
7	Closed	Closed	Open	Open

Client–server Models

- **Basic Model**
- **Application Layering**
- **Client–server Architectures**

Servers: Generally provide services related to a *shared resource*:

- Servers for file systems, databases, implementation repositories, etc.
- Servers for shared, linked documents (Web, VoD)
- Servers for shared applications
- Servers for shared distributed objects

Clients: Allow remote service access:

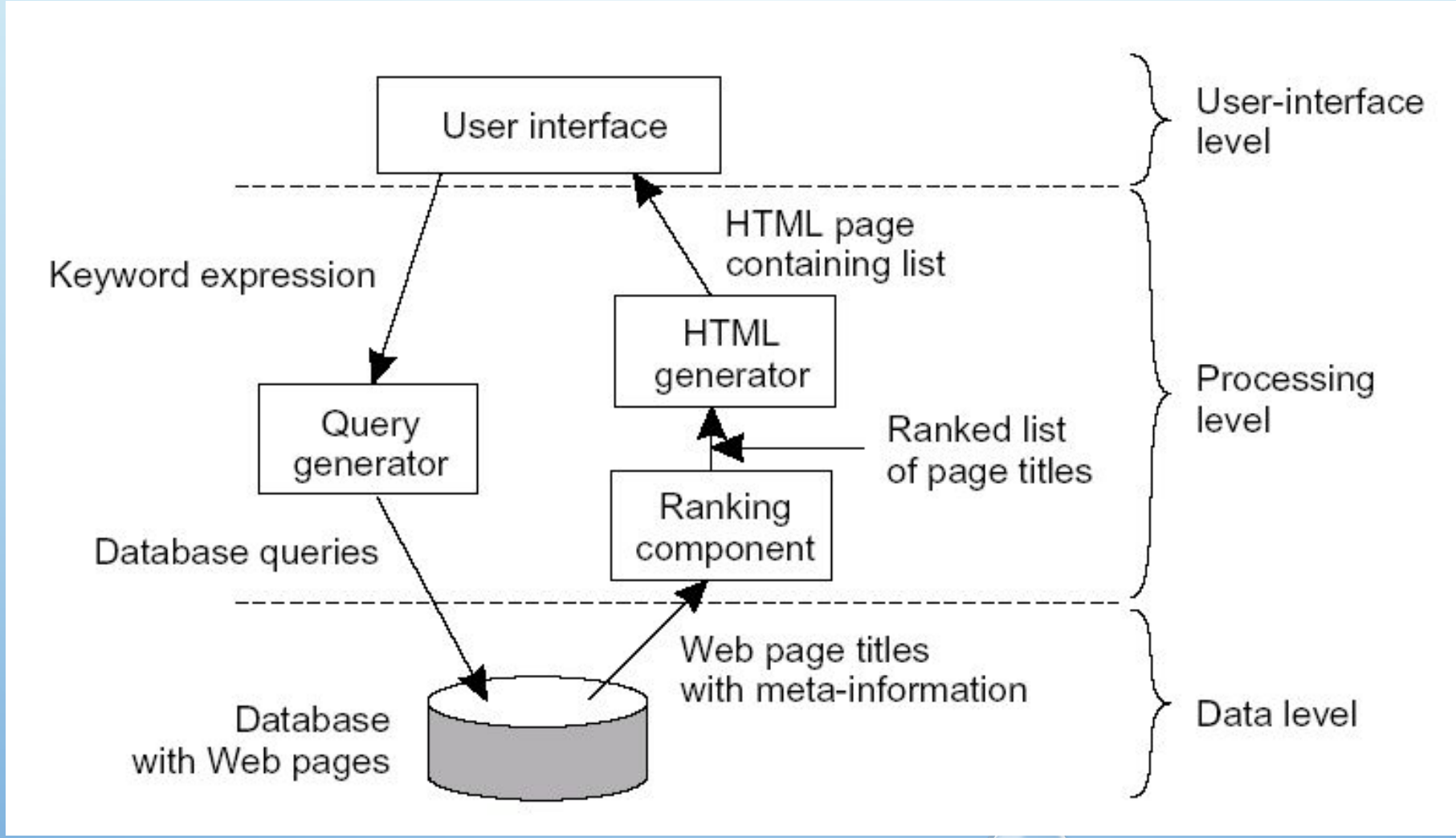
- Programming interface transforming client's local service calls to request/reply messages
- Devices with (relatively simple) digital components (barcode readers, teller machines, hand-held phones)
- Computers providing independent user interfaces for specific services
- Computers providing an integrated user interface for related services (compound documents)

APPLICATION LAYERING (1/2)

- **Traditional Three-layered View:**
- User-interface layer contains units for an application's user interface
- Processing layer contains the functions of an application, i.E. Without specific data
- Data layer contains the data that a client wants to manipulate through the application components

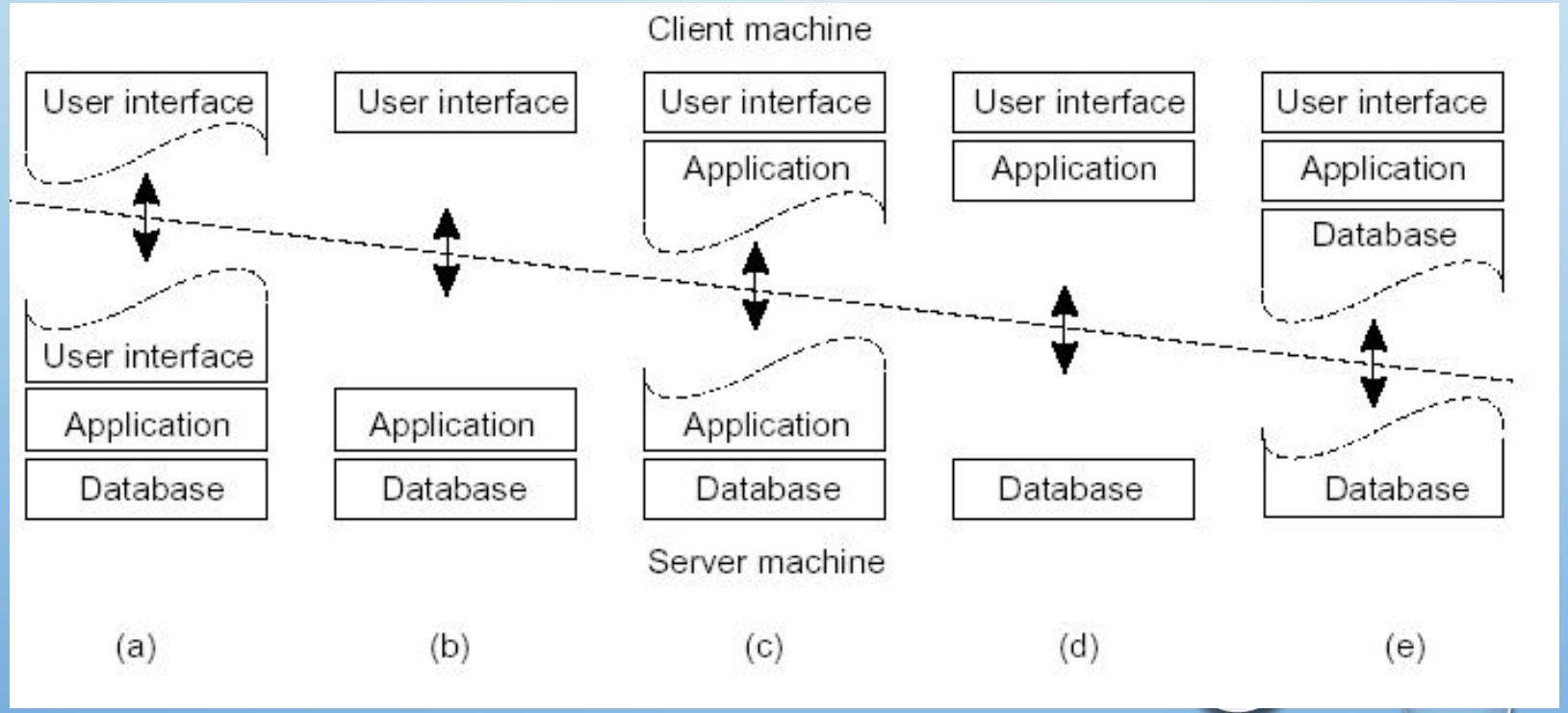
- **Observation:** this layering is found in many distributed information systems, using traditional database technology and accompanying applications.

APPLICATION LAYERING (2/2)



Client-server Architectures

- **Single-tiered:** Dumb Terminal/Mainframe Configuration
- **Two-tiered:** Client/Single Server Configuration
- **Three-tiered:** Each Layer On Separate Machine
- **TRADITIONAL TWO-TIERED CONFIGURATIONS:**



Alternative C/S Architectures

- Multi-tiered Architectures Seem To Constitute Buzzwords That Fail To Capture Many Modern Client–server Systems.
- **Cooperating Servers:** Service Is Physically Distributed Across A Collection Of Servers:
  - Traditional Multi-tiered Architectures
  - Replicated File Systems
  - Network News Services
  - Large-scale Naming Systems (DNS, X.500)
  - Workflow Systems
  - Financial Brokerage Systems
- **Cooperating Clients:** Distributed Application Exists By Virtue Of Client Collaboration:
  - Teleconferencing Where Each Client Owns A (Multimedia) Workstation
  - Publish/Subscribe Architectures In Which Role Of Client And Server Is Blurred
  - Peer-to-peer (P2P) Applications

BOOK

- **Distributed systems: Principles and Paradigms** by **Tanenbaum**

QUESTIONS

- Goals of distributed systems
- Problems with scalability and solutions (techniques for scalability)
- What is an open distributed system
- Compare distributed systems with NOS, DOS, and middleware with neat diagrams
- Types of client server architectures
- An experimental file server is up 75% of the time and down for 25% of the time due to bugs. How many times does this file server have to be replicated to give an availability of at least 99% ?