18CSE356T
Distributed Operating Systems

Unit I

Course Outcome

The purpose of learning this unit is to:

- ***CLR-1 : To recognize the essential concepts of distributed system***

At the end of this unit, learners will be able to:

- ***CLO-1 : Characterize the fundamental hardware and software concepts of distributed systems*.**

Topics Covered

- Introduction- Distributed Systems
- Goals of Distributed Systems
- Hardware Concepts- Bus-based Multiprocessors
- Switched Multiprocessors
- Bus-based Multicomputers
- Switched Multicomputers

INTRODUCTION- DISTRIBUTED SYSTEMS
GOALS OF DISTRIBUTED SYSTEMS

Operating System (OS)

- An **Operating System** (OS) is an **interface** between a **computer user** and **computer hardware**.
- An **operating system** is a software which performs all the basic tasks like **file management, memory management, process management, handling input and output**, and **controlling peripheral devices** such as disk drives and printers.

Definition of a Distributed System (1)

A distributed system is:

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

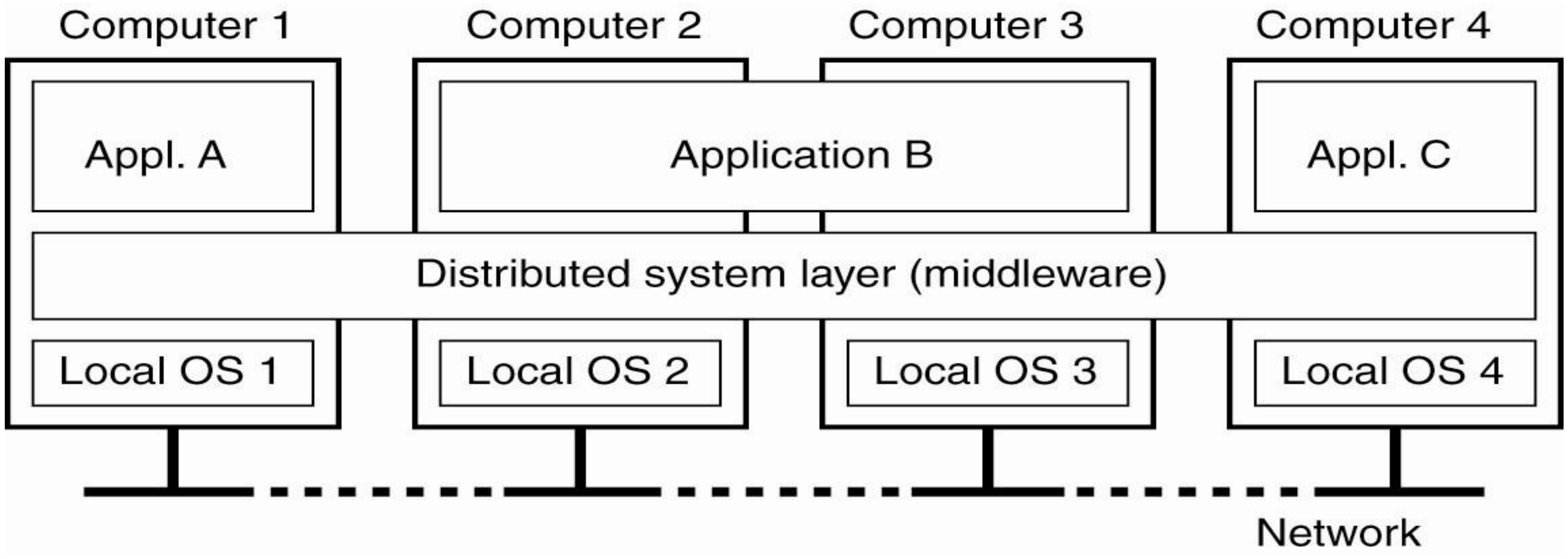
Ex:

1. Pool of processors - assigned dynamically

2. Defective Robot in n/w – replace robot

3. Bank Computer Communication

Definition of a Distributed System (2)

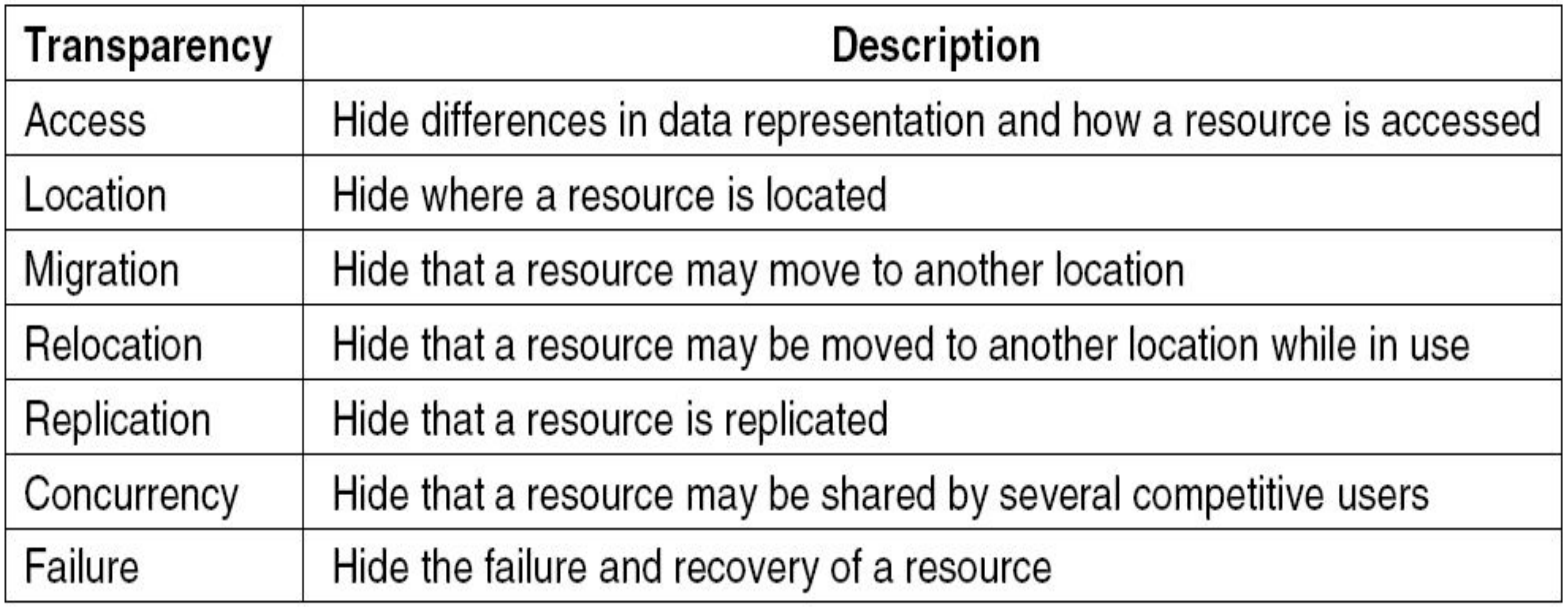


A distributed system organized as middleware.
The middleware layer extends over **multiple machines**, and offers each application the **same interface**

Goals of Distributed Systems

- Making Resources Accessible - Easily Connect Users/Resources
- Exhibit Distribution Transparency
- Support Openness
- Be Scalable:
  - in size
  - Geographically (the users and resources may lie far apart)
  - administratively

Transparency in a Distributed System

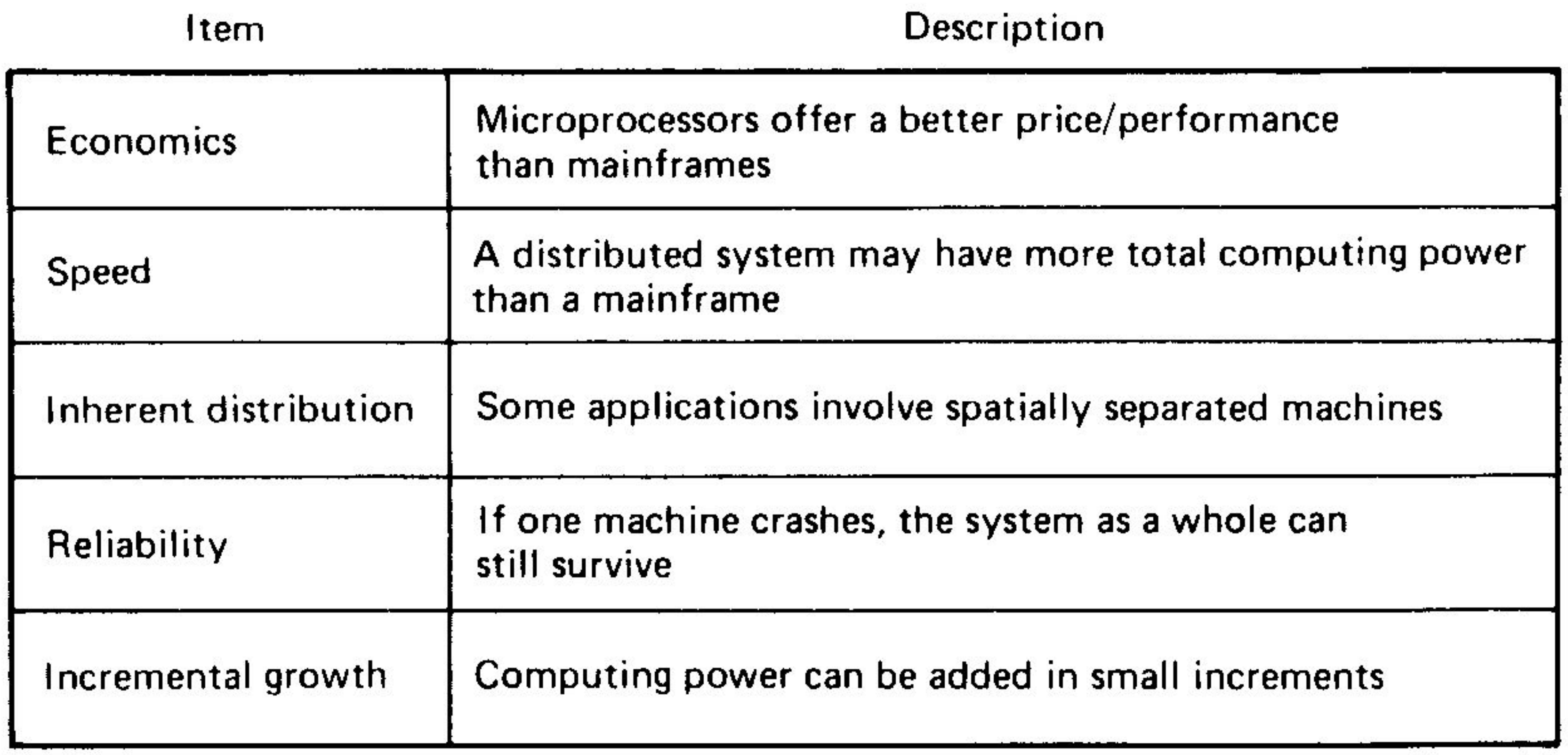


Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that 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

Different forms of transparency in a distributed system (ISO, 1995)

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

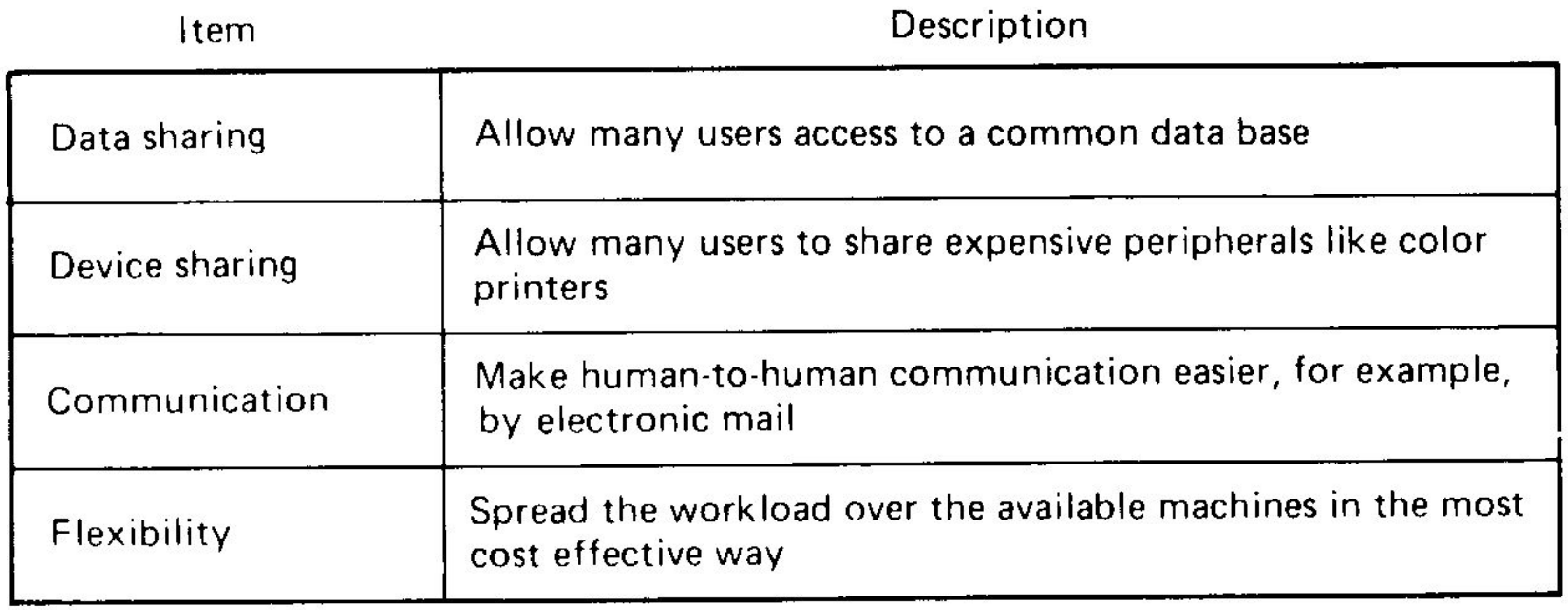
Advantages of Distributed System over Centralized System



Item	Description
Economics	Microprocessors offer a better price/performance than mainframes
Speed	A distributed system may have more total computing power than a mainframe
Inherent distribution	Some applications involve spatially separated machines
Reliability	If one machine crashes, the system as a whole can still survive
Incremental growth	Computing power can be added in small increments

Fig. 1-1. Advantages of distributed systems over centralized systems.

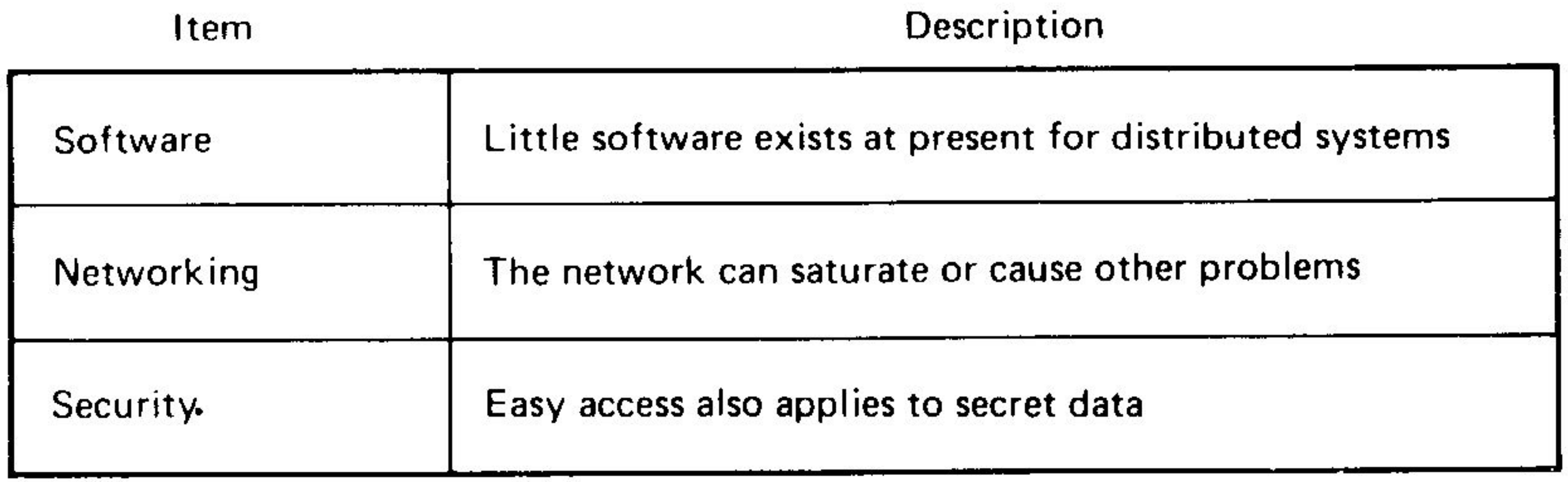
Advantages of Distributed System over isolated PCs



Item	Description
Data sharing	Allow many users access to a common data base
Device sharing	Allow many users to share expensive peripherals like color printers
Communication	Make human-to-human communication easier, for example, by electronic mail
Flexibility	Spread the workload over the available machines in the most cost effective way

Fig. 1-2. Advantages of distributed systems over isolated (personal) computers.

Disadvantages of Distributed System



Item	Description
Software	Little software exists at present for distributed systems
Networking	The network can saturate or cause other problems
Security.	Easy access also applies to secret data

Fig. 1-3. Disadvantages of distributed systems.

Hardware Concepts

- Bus-based Multiprocessors
- Switched Multiprocessors
- Bus-based Multicomputers
- Switched Multicomputers

Computer - Design Concepts

- Flynn's Taxonomy
  - Number of Instructions Stream
    - Number of Data streams
- Classification based on Flynn's Taxonomy
  - SISD (ex. Uniprocessor)
  - SIMD (ex. Super computers)
  - MISD (no such available)
  - MIMD (ex. Multiprocessor computers)

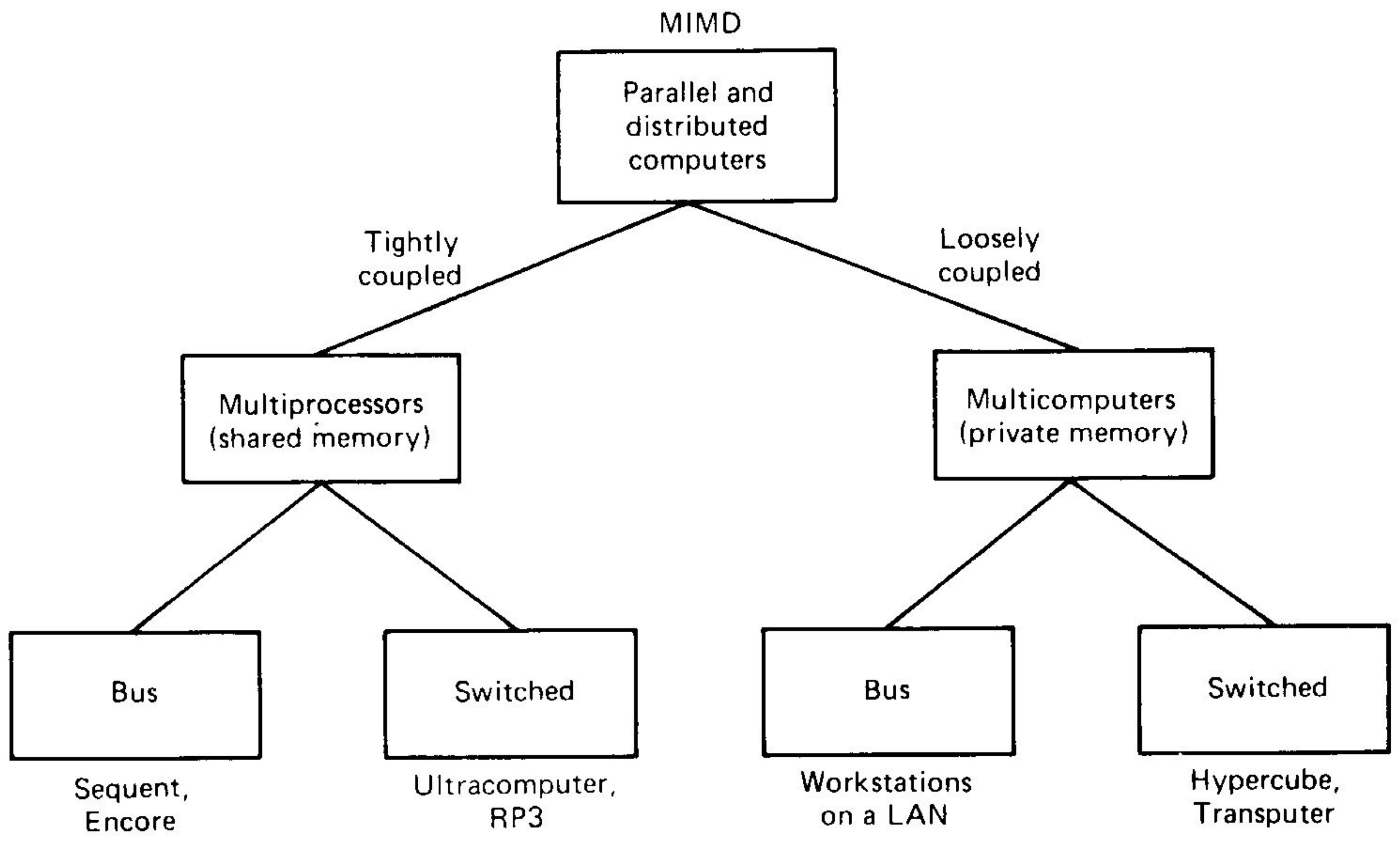
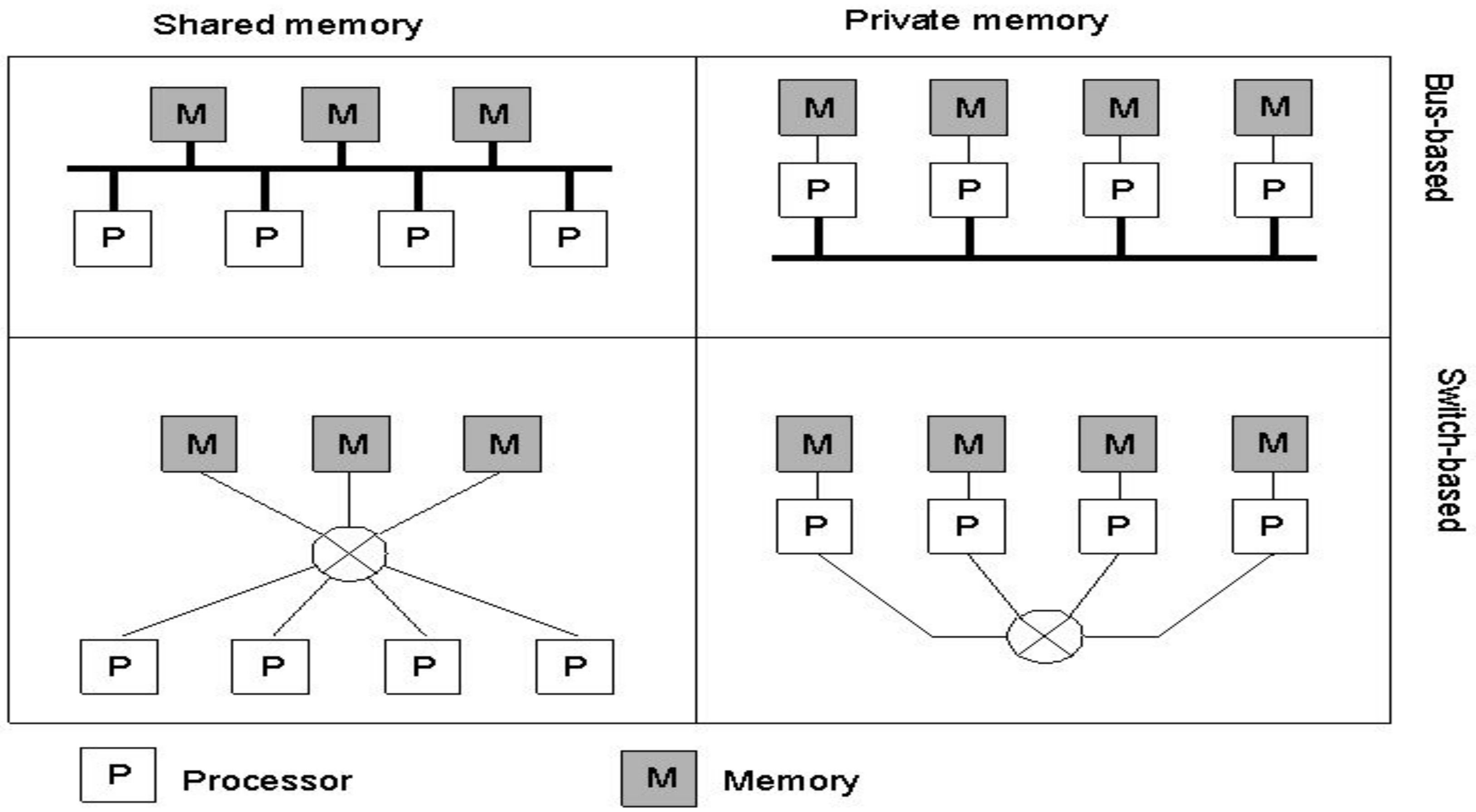


Fig. 1-4. A taxonomy of parallel and distributed computer systems.

CLASSIFICATION

- General Classification:
  - Multiprocessor – a single address space among the processors
  - Multicomputer – each machine has its own private memory.
- OS can be developed for either type of environment.

Hardware Concepts
Multiprocessor vs. Multicomputer



Basic organizations and memories in distributed computer systems

Tightly Vs Loosely Coupled

- Tightly – Delay Experienced when a message is sent from one computer to another is short and Data rate is high.
- Mostly used as parallel systems.
  - Eg: 2 CPU chips on the same Printed circuit board connected by wires.
- Loosely – Opposite to previous case. Intermachine message delay is large and data rate is low.
- Mostly used as distributed systems.
  - Eg: 2 computers connected by 2400 bits/sec modem over the telephone system.

MULTIPROCESSORS
(BUS AND SWITCH BASED)

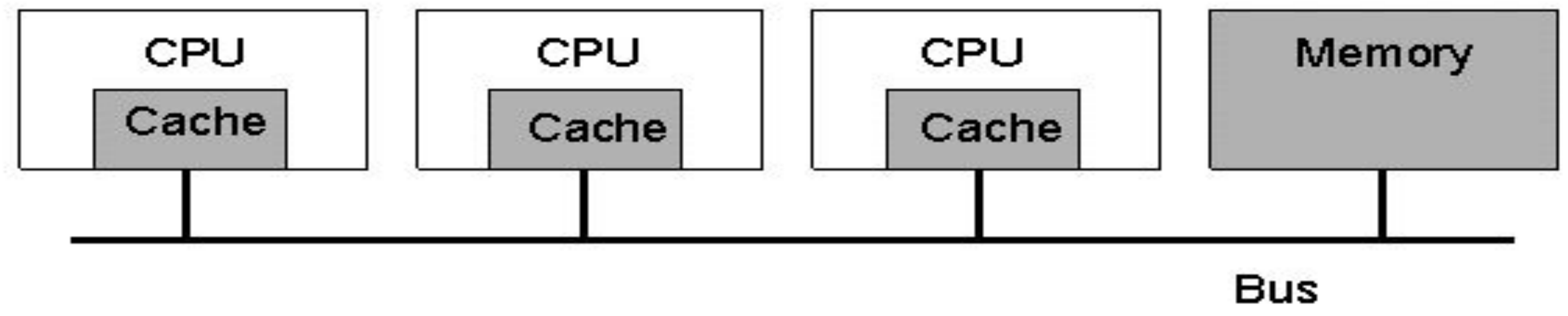
Bus Based Multiprocessors

- Bus-based Multiprocessor – has multiple CPUs all connected via a common bus
- A bus has – 32/64 bit address line, 32/64 bit data line and 32 or more bit control lines
- Memory management is **coherent.(CPU B gets CPU A's written data immediately)**
- **Coherent – Plays an important role in DOS**
- If no. of CPU increases then performance will be degrade, use **cache memory**
- To improve performance, **cache memory** is used b/w bus and CPU
- Cache memory management

  - Write-through cache

  - Snoopy cache

Bus Based Multiprocessors



A bus-based multiprocessor

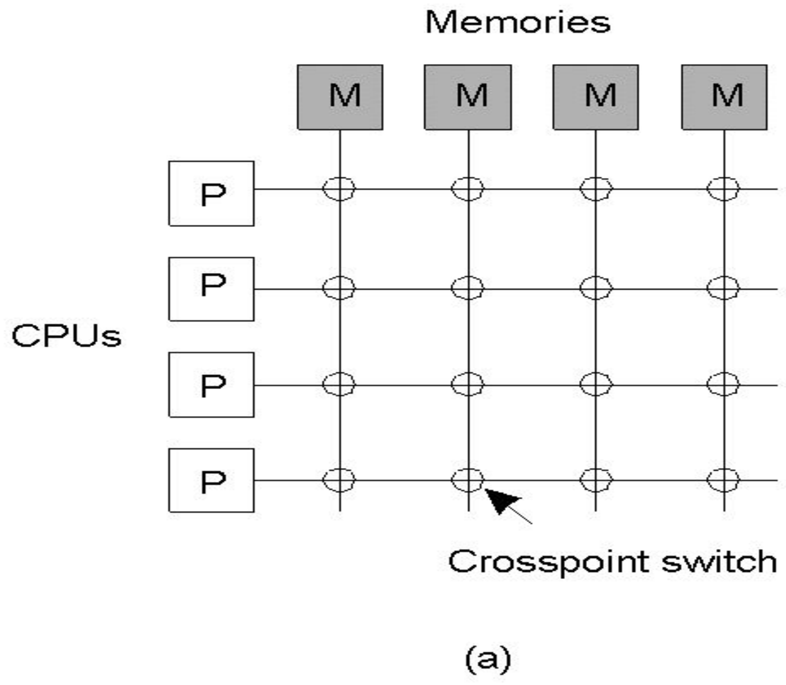
SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

Bus Based Multiprocessors

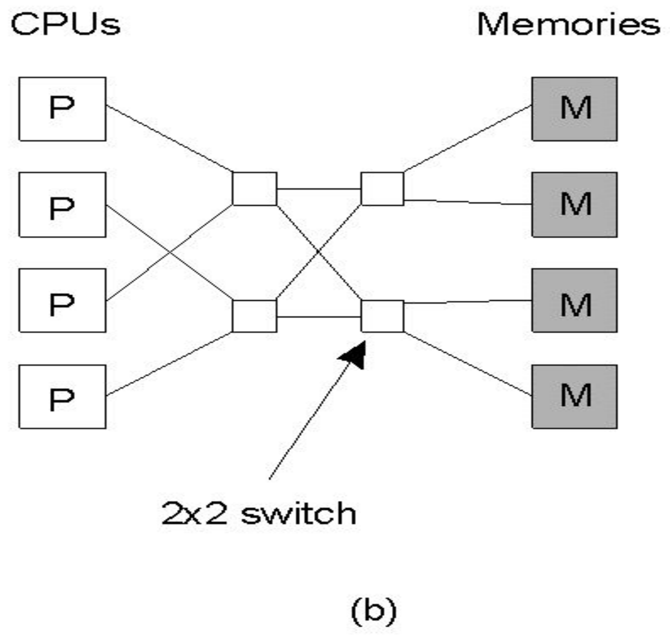
- *Uniform Memory Access [UMA]*
  - *Caching is vital for reasonable performance (e.g., caches on a shared memory multiprocessor).*
  - *Want to maintain cache coherency*
    - ***Write-through cache** : Any changes to cache are written through to memory*
    - ***Snoopy Cache**: Any changes to cache are either removed or updated with new value.*

A design consisting of snoopy write- through caches is COHERENT and INVISIBLE to the programmer.

Switched Multiprocessors



A crossbar switch



An Omega Switching network

Crossbar Switch

- Useful to connect multiple CPUs with many memory modules
- Crosspoint switch is used to connect CPU with memory module based on requirement.
- To connect n-CPUs and n-Memory modules, n^2 crosspoint switches are required.
- For larger n, the interconnection is complex.
- When CPU want to access memory corresponding crosspoint closed and its allow to access
- If 2 CPU try to access same memory then 1 should be wait

An omega switching network

- Contains 4 (2x2) switches having 2 input and 2 output ports, switching between them.
- Less complex compared to crossbar switch as this omega switching requires only n/2 switches to connect n-CPUs with n-memory modules with $\log_2 n$ switching stages.
- Limitations: for larger values of n, the switching time is larger and increases delay in instruction execution. Moreover its expensive

Switched Multiprocessors-Delay and cost

- ***Non-Uniform Memory Access** [NUMA]*
  - *Attempts to reduce delay – incurs more cost.*
  - *Attempts to reduce cost gave rise to - A hierarchy where CPUs have their own memory (not the same as a cache)*
  - *Access costs to memory is non-uniform.*

- Conclusion about MULTIPROCESSORS (BUS and SWITCH BASED)

"***Building a large, tightly – coupled, shared memory multiprocessor is possible, but is difficult and expensive***"

MULTICOMPUTERS

Multicomputer Systems

The network traffic is much lower than multiprocessor systems

CPU-to-CPU v.s. CPU-to-Memory

<u>Bus-based multicomputer</u> Example: fast-Ethernet cluster

Problems: scalability, 25-100 nodes at most

<u>Switch-based multicomputer</u> –various intercon. n/ws

- Grid – easy to understand, pbms that have 2 dimen nature(Graph Theory or vision)
- Hypercubes: n-dimension cube,4dimen.,vertices & edges

Vertex – CPU, edge-connection b/w cpu's, each CPU n connections to other CPU

✔ Hypercubes with 1024 CPUs have been available

✔ 16,384 CPUs are starting to become available

Bus-based Multicomputer

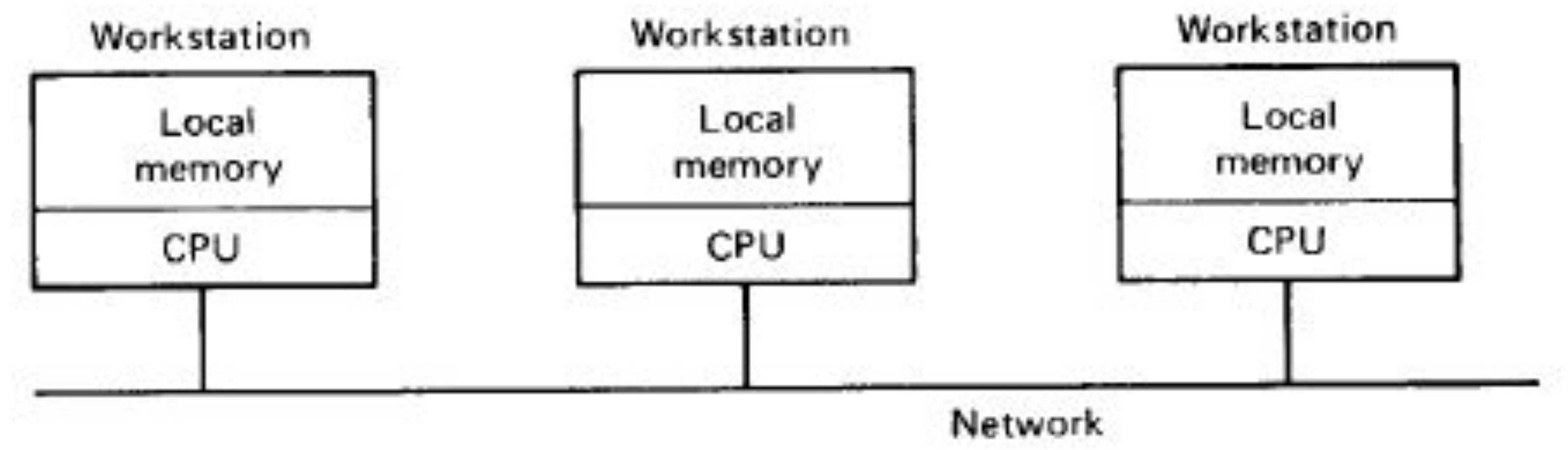
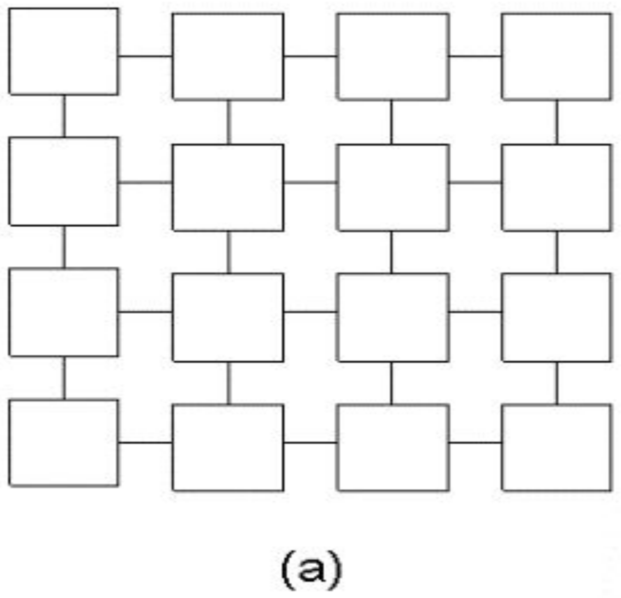
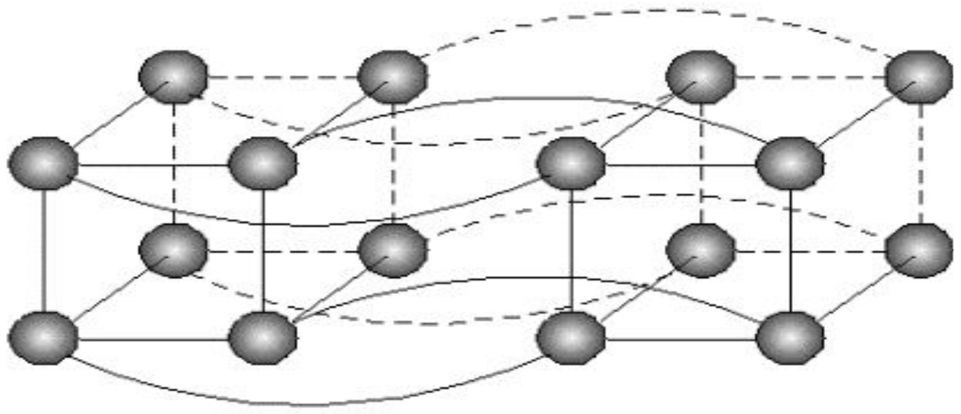


Fig. 1-7. A multicomputer consisting of workstations on a LAN.

Switched-based Multicomputer


(a)


(b)

Grid   Hypercube

Software Concepts
Design Issues

Software Concepts

- Focuses on various types of Operating Systems for the multiprocessors and multicomputers.
- Also discusses on which type of software goes with which kind of hardware.
- Two types of Operating systems
  - Loosely – Allows users and machines to be fundamentally independent of one another – interact wherever necessary.
  - Tightly – Allows a multiprocessor dedicate to a specific purpose.

NOS – File Server

- Provides shared, global file system accessible from all the workstations.
- The file system is supported by one or more machines called ***file servers.***
- Accept requests from user's programs running on the other (nonserver) machines called ***CLIENTS*** – to read and write files.
- Each request – examined and executed – reply sent back.
- Maintains Hierarchial file systems – root directories, subdirectories and files.
- Workstations can import or mount these file systems , augmenting their local file systems with those located on server.

Network Operating System

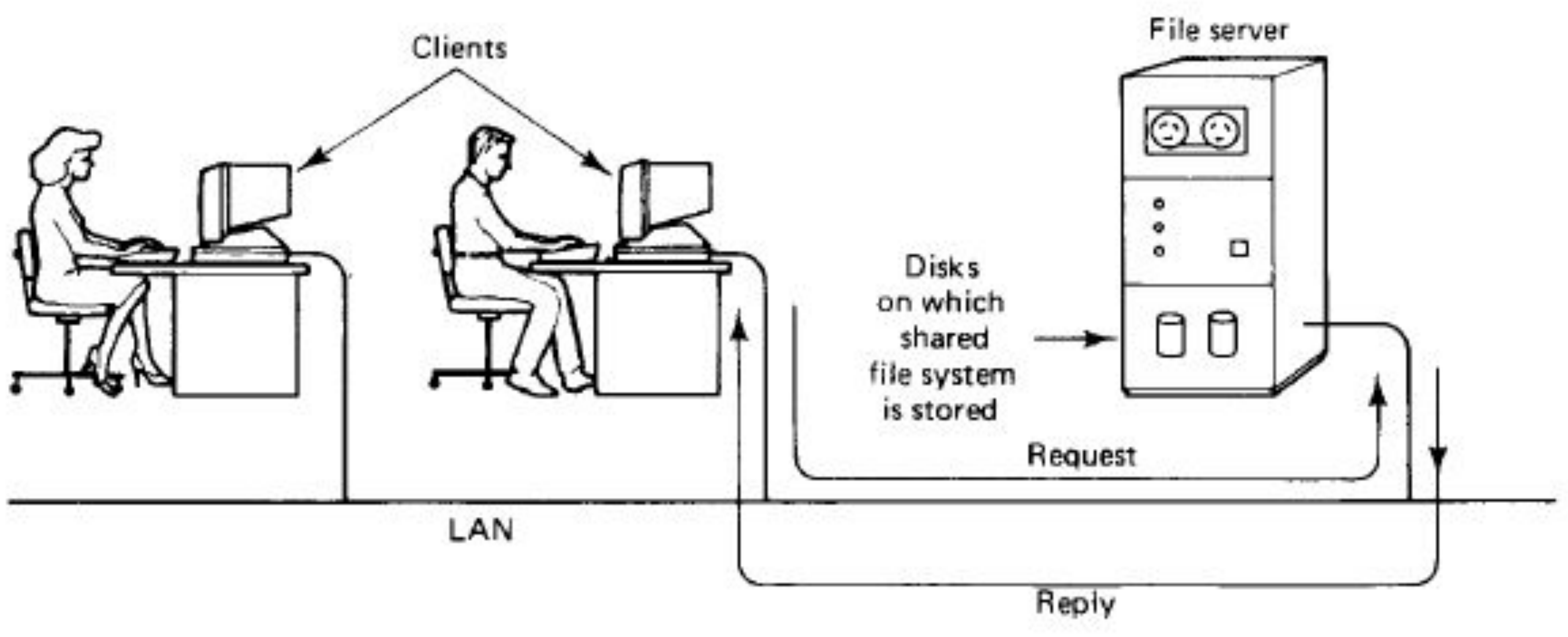


Fig. 1-9. Two clients and a server in a network operating system.

Network Operating System

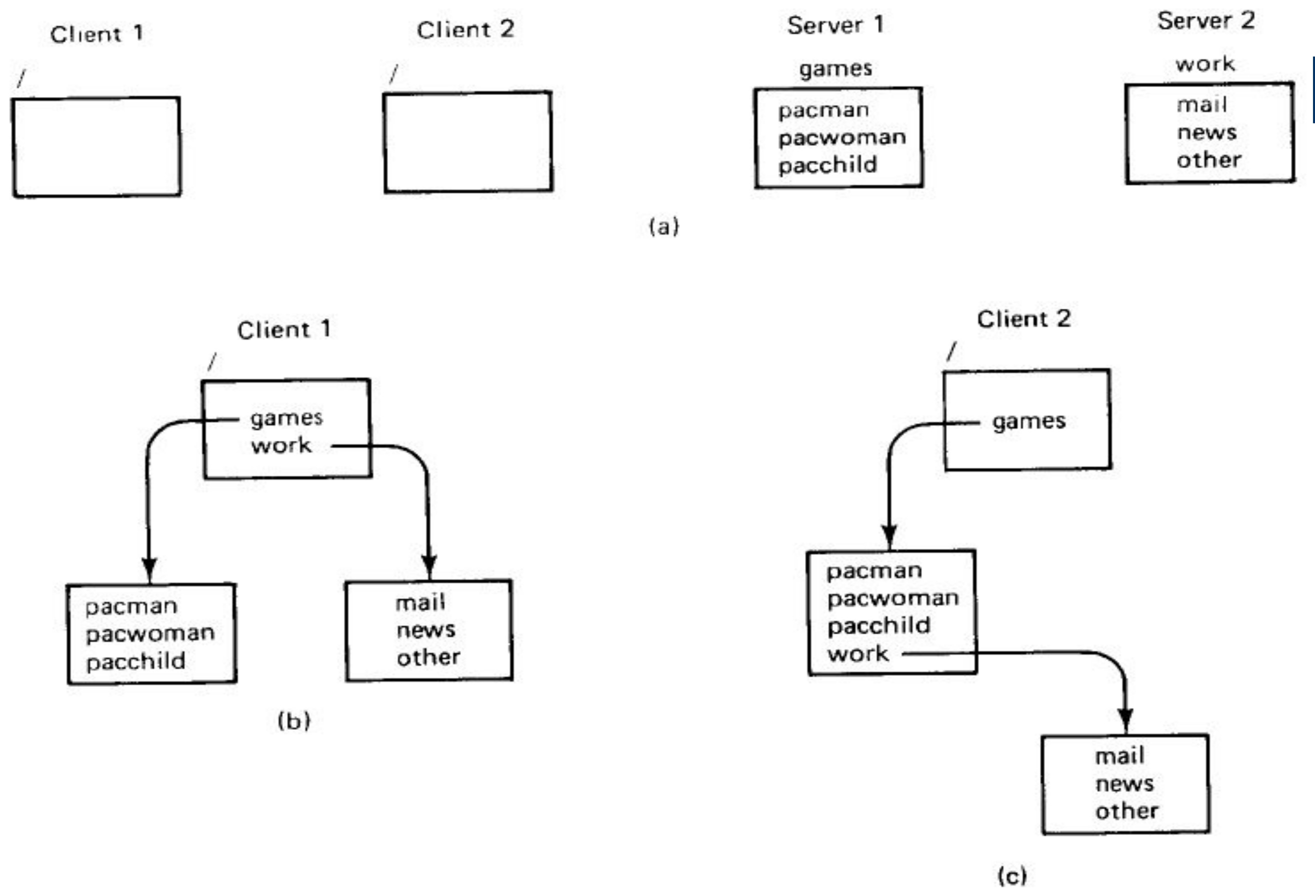


Fig. 1-10. Different clients may mount the servers in different places.

- Different clients may mount the servers in different places
- Inconsistencies in view make NOS's harder, in general for users than DOS's.
  - But easier to scale by adding computers

Merits and Demerits of NOS

- The advantages of network operating systems are as follows –
  - Centralized servers are highly stable.
  - Security is server managed.
  - Upgrades to new technologies and hardware can be easily integrated into the system.
  - Remote access to servers is possible from different locations and types of systems.
- The disadvantages of network operating systems are as follows –
  - High cost of buying and running a server.
  - Dependency on a central location for most operations.
  - Regular maintenance and updates are required.

True Distributed Systems

- Tightly-coupled Software on Loosely-Coupled Hardware.
- GOAL: To give an illusion as single time-sharing system – called as ***SINGLE-SYSTEM IMAGE or VIRTUAL UNIPROCESSOR.***
- ***Characteristics of a true DS:***
  - *Single*
  - *Global Inter process communication mechanism*
  - *Global Protection Scheme*
- ***Not enough***
  - ***Process management*** *– must be same*
  - ***File System*** *- must be same*

Multiprocessor Timesharing System

- Tightly-coupled software on tightly-coupled hardware.

Eg: Multiprocessor that are operated as UNIX time sharing system.

32 30-MIPS CPUs □ 960-MIPS CPU

- <u>**Key Characteristic**</u>: Existence of a single-run queue: a list of all the processes in the system that are logically unblocked and ready to run.
- Organization of file system differs from previous NOS and True DS.

Single Run Queue

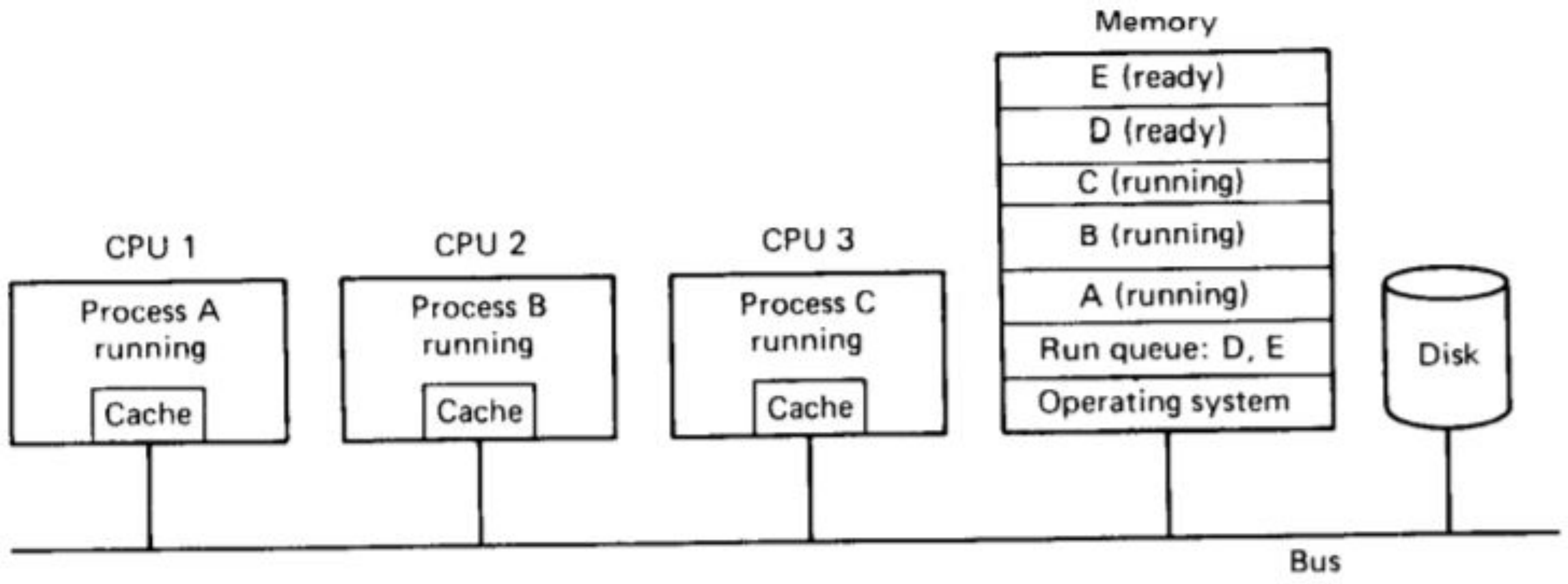
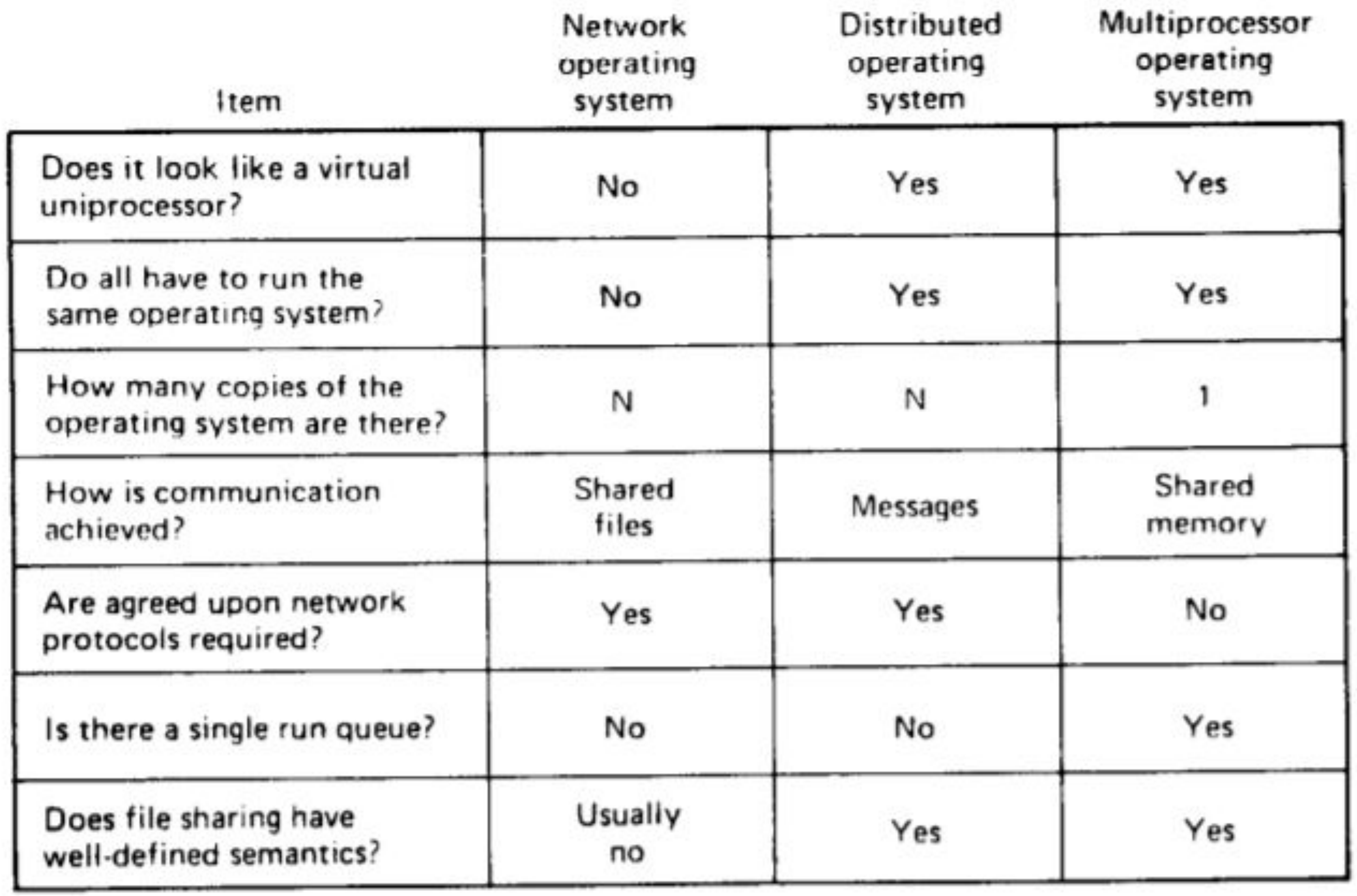


Fig. 1-11. A multiprocessor with a single run queue.



Item	Network operating system	Distributed operating system	Multiprocessor operating system
Does it look like a virtual uniprocessor?	No	Yes	Yes
Do all have to run the same operating system?	No	Yes	Yes
How many copies of the operating system are there?	N	N	1
How is communication achieved?	Shared files	Messages	Shared memory
Are agreed upon network protocols required?	Yes	Yes	No
Is there a single run queue?	No	No	Yes
Does file sharing have well-defined semantics?	Usually no	Yes	Yes

Fig. 1-12. Comparison of three different ways of organizing *n* CPUs.

Software Concepts

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

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

DESIGN ISSUES

- Transparency
- Flexibility
- Reliability
- Performance
- Scalability

Transparency

- Should be perceived as the single entity by the users.
- User should be unaware of where the services are located and the transferring from a local machine to a remote one should be transparent.
- Achieved in easy way:
  - To hide the distribution from users.

    Eg: UNIX – make command

❑ Hiding is harder , but also possible through the System call interface.

Eg: By issuing System call to read files.

Transparency

- The concept of transparency can be applied to several aspects of a distributed system.

a) **Location transparency:** The users cannot tell where resources (h/w and s/w) are located

b) **Migration transparency:** Resources can free to move without changing their names

c) **Replication transparency:** The users cannot tell how many copies exist.

d) **Concurrency transparency:** Multiple users can share resources automatically.

e) **Parallelism transparency:** Activities can happen in parallel without users knowing.

Flexibility

- Microkernel - user services and kernel, services are kept in separate address space.
- Monolithic kernel,- both user services and kernel services are kept in the same address space.

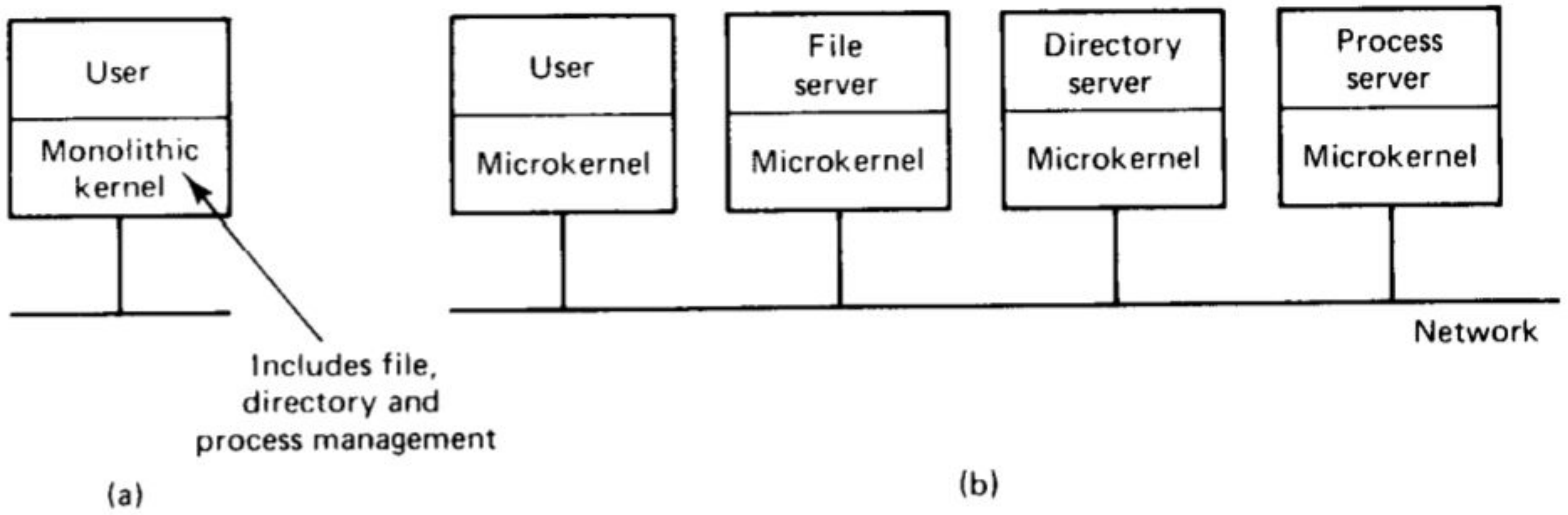
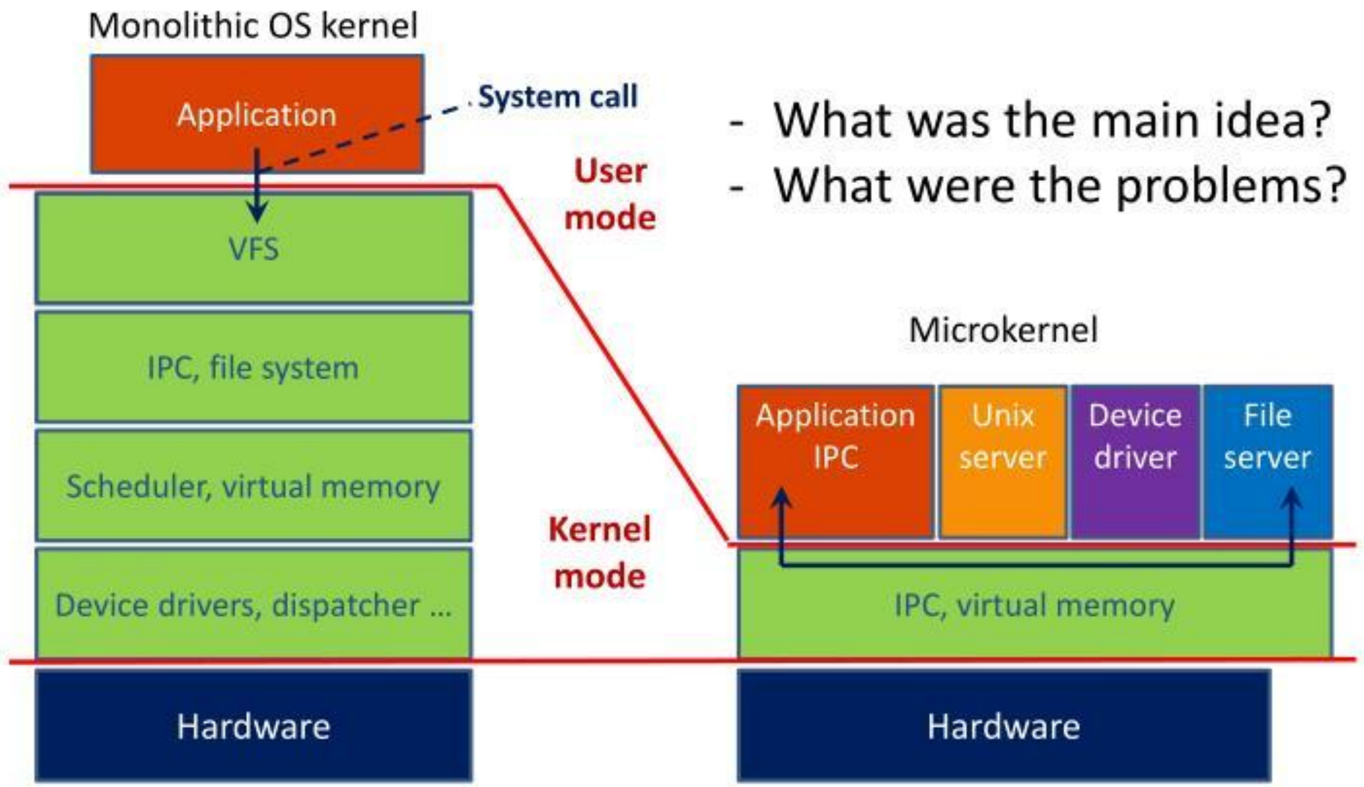


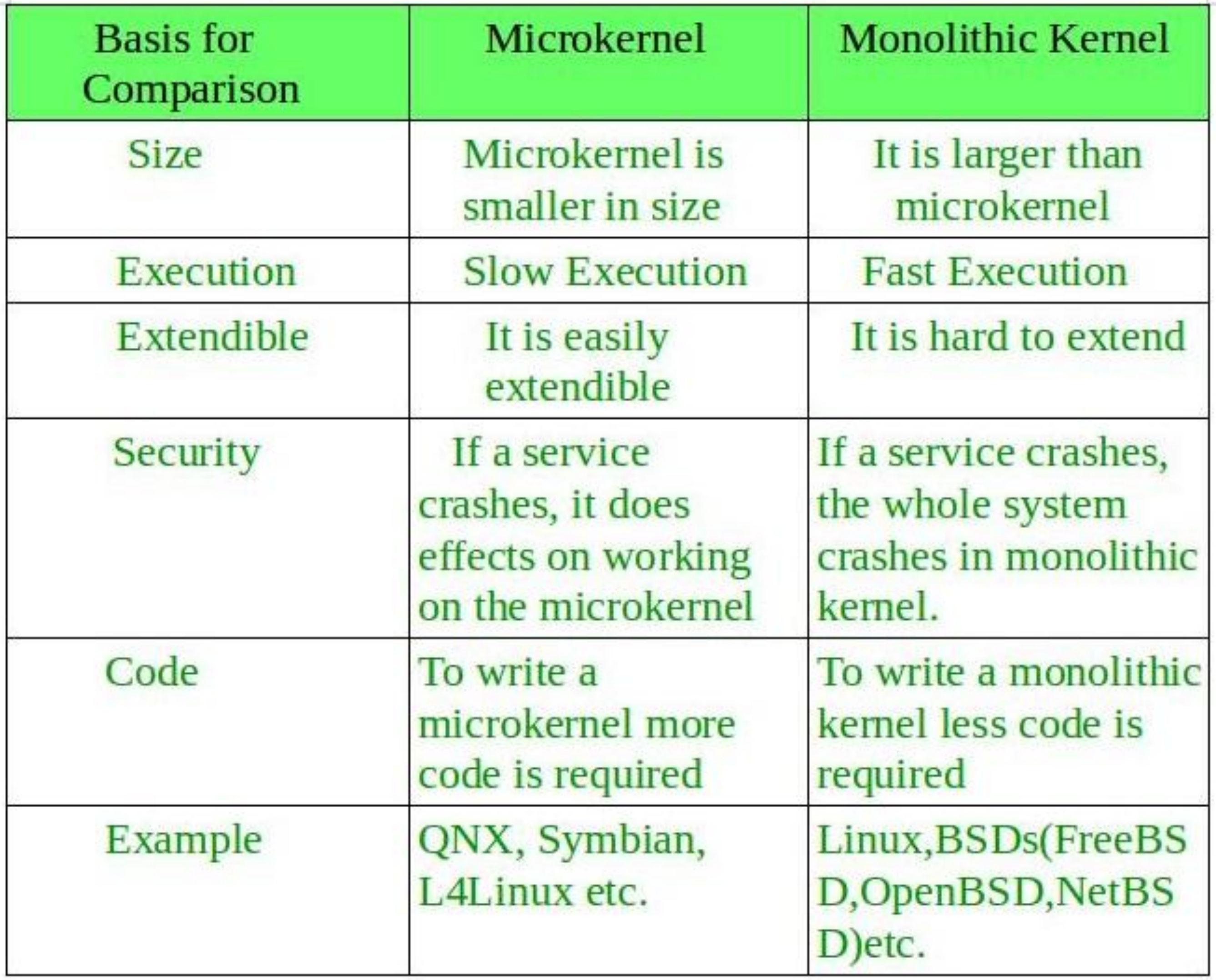
Fig. 1-14. (a) Monolithic kernel. (b) Microkernel.

Monolithic kernel vs Microkernel



- What was the main idea?
- What were the problems?

Micro Vs Monolithic



Basis for Comparison	Microkernel	Monolithic Kernel
Size	Microkernel is smaller in size	It is larger than microkernel
Execution	Slow Execution	Fast Execution
Extendible	It is easily extendible	It is hard to extend
Security	If a service crashes, it does effects on working on the microkernel	If a service crashes, the whole system crashes in monolithic kernel.
Code	To write a microkernel more code is required	To write a monolithic kernel less code is required
Example	QNX, Symbian, L4Linux etc.	Linux,BSDs(FreeBSD,OpenBSD,NetBSD)etc.

Reliability

- Goals of building distributed systems - To make them more reliable than single-processor systems.
- The idea is that if a machine goes down, some other machine takes over the job.
- **Three important aspects : 1) Availability 2) Security 3) Fault Tolerance**
- **1) Availability:**

A highly reliable system must be highly **available**, but that is not enough.

- Data entrusted to the system must not be lost or garbled in any way, and if files are stored redundantly on multiple servers, all the copies must be kept consistent.
- In general, the more copies that are kept, the better the availability, but the greater the chance that they will be inconsistent, especially if updates are frequent.

Reliability (1)

- **Security**
  - In a DS, its hard to achieve.
  - No name or identification field in the incoming message.
- **Fault Tolerance**
  - DS can be designed to mask failures (i.e) to hide them from users.

Performance

- Always the hidden data in the background is the issue of performance.
- Building a transparent, flexible, reliable distributed system, more important lies in its performance.
- In particular, when running a particular application on a distributed system, it should not be appreciably worse than running the same application on a single processor.

Performance

- Performance metrics used:
  - Throughput
  - Response time
  - System Utilization
  - Amount of network capacity consumed
- Performance problem is often due to - **Communication**
- Solution – To reduce the number of messages communicated as far as possible.

Performance

- Parallelism – Based on the grain size of all computations
  - Fine-grained
  - Coarse-grained
- Fine-grained: Jobs involving large number of small computations, High interaction rates, More data.
- Coarse-grained: Jobs involving large computations, Less interaction rates, little data. [BETTER FIT]

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

Examples of scalability limitations

CENTRALIZED SERVICES

- Many services are centralized in the sense that they are implemented by means of only a single server running on a specific machine in the distributed system.
- Problem with this scheme is obvious: the server can become a bottleneck as the number of users and applications grows.
- Even if we have virtually unlimited processing and storage capacity, communication with that server will eventually prohibit further growth

CENTRALIZED DATA

- Having a single database would undoubtedly saturate all the communication lines into and out of it.
- Domain Name System (DNS)
  - DNS maintains information on millions of computers worldwide and forms an essential service for locating Web servers.
  - If each request to resolve a URL has to be forwarded to that **one and only DNS server**, no one would be using the Web.

CENTRALIZED ALGORITHMS

- Enormous number of messages have to be routed over many lines.
- Optimal way to do this
  - Collect complete information about the load on all machines and lines
  - Run an algorithm to compute all the optimal routes.
  - Spread this information around the system to improve the routing.
- These messages would overload part of the network
- SOLUTION : TO USE DECENTALIZED ALGORITHMS

:Characteristics of decentralized algorithms

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

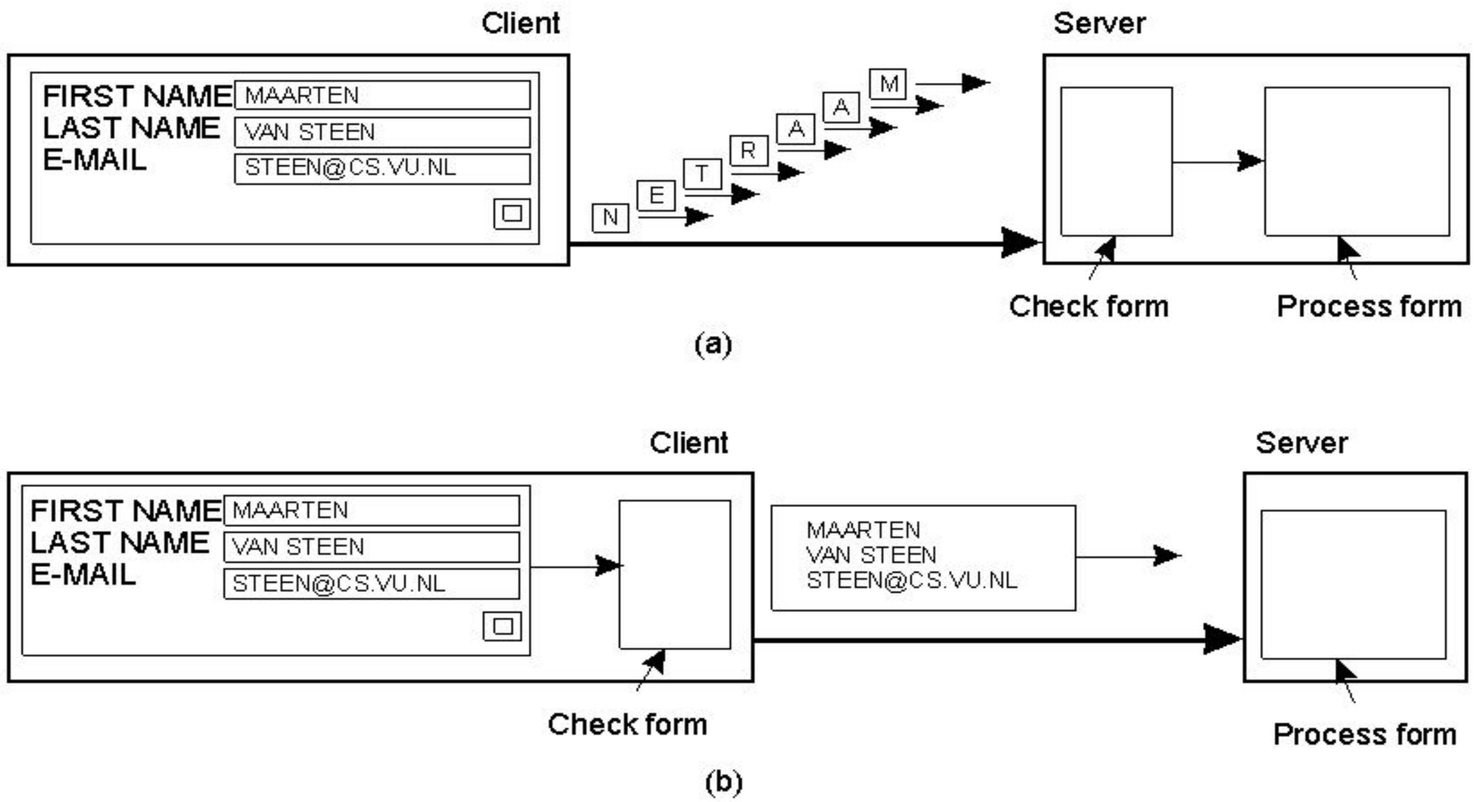
Scaling Techniques

- Hiding communication latencies
- Distribution
- Replication

Hiding communication latencies

- Is important to achieve geographical scalability.
  - Try to avoid waiting for responses to remote (and potentially distant) service requests as much as possible.
- Use only asynchronous communication – DO SOME USEFUL WORK WHILE WAITING
- Many applications not making effective use of asynchronous communication
  - SOLUTION: 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

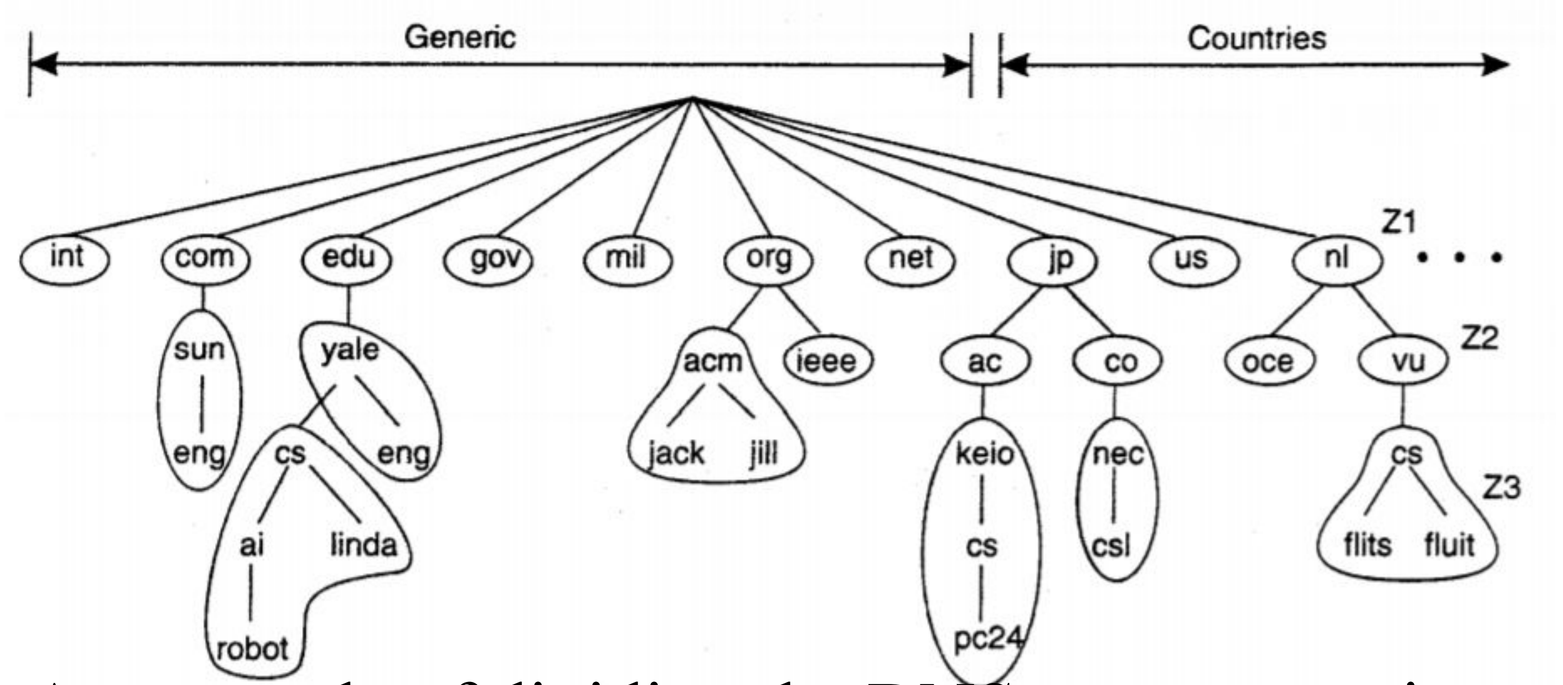
Hiding communication latencies (1)



The difference between letting (a) a server or (b) a client check forms as they are being filled

Distribution

- Taking a component, splitting it into smaller parts, and subsequently spreading those parts across the system.
  - Eg :1 DOMIAN NAME SERVICE (DNS)
  - Eg: 2 WORLD WIDE WEB



An example of dividing the DNS name space into zones

Replication

- Replicate components across a distributed system.
- Not only increases availability, but also balances the load between components leading to better performance.
- A special form of replication – ***Caching***- results in making a copy of a resource, generally in the proximity of the client accessing that resource.
- **Caching Vs Replication**:
  - Caching - a decision made by the client of a resource, and not by the owner of a resource.
  - Caching happens on demand whereas replication is often planned in advance.

Additional Contents on Distributed Systems

Examples of Distributed Systems

- An "ATM machine"
- A remote file access mechanism
- A database
- A chat room
- A computing "grid" (like SETI)

Pitfalls when Developing Distributed Systems

- Peter Deutsh (SUN MICROSYSTEMS)
  - 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

Types of Distributed Systems

- Distributed Computing Systems
  - High Performance Computing (HPC)
- Distributed Information Systems
  - Transaction Processing Systems (TPS)
  - Enterprise Application Integration (EAI)
- Distributed Pervasive Systems
  - Ubiquitous Systems

DISTRIBUTED COMPUTING SYSTEMS

SRM INSTITUTE OF SCIENCE & TECHNOLOGY
Deemed to be University u/s 3 of UGC Act, 1956

High Performance Computing (HPC)

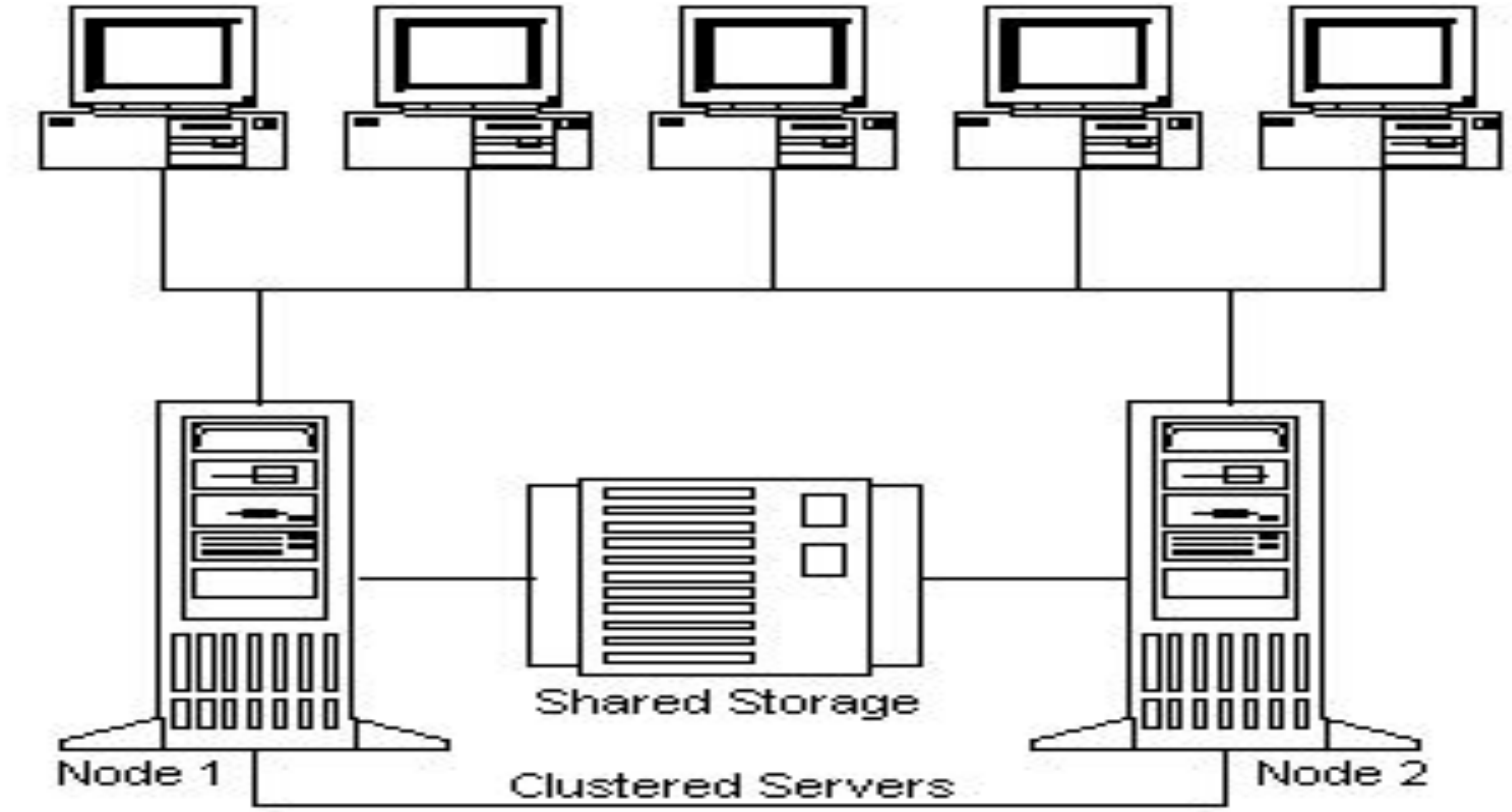
- TWO SUBGROUPS

CLUSTER COMPUTING	GRID COMPUTING
Underlying hardware consists of (1 a collection of similar workstations or PCs, closely connected by means of a high speed local-area network	Consists of distributed systems (1 that are often constructed as a federation of computer systems, where each system may fall under .a different administrative domain
Each node runs the same (2 .operating system	Very different when it comes to (2 hardware, software, and deployed .network technology

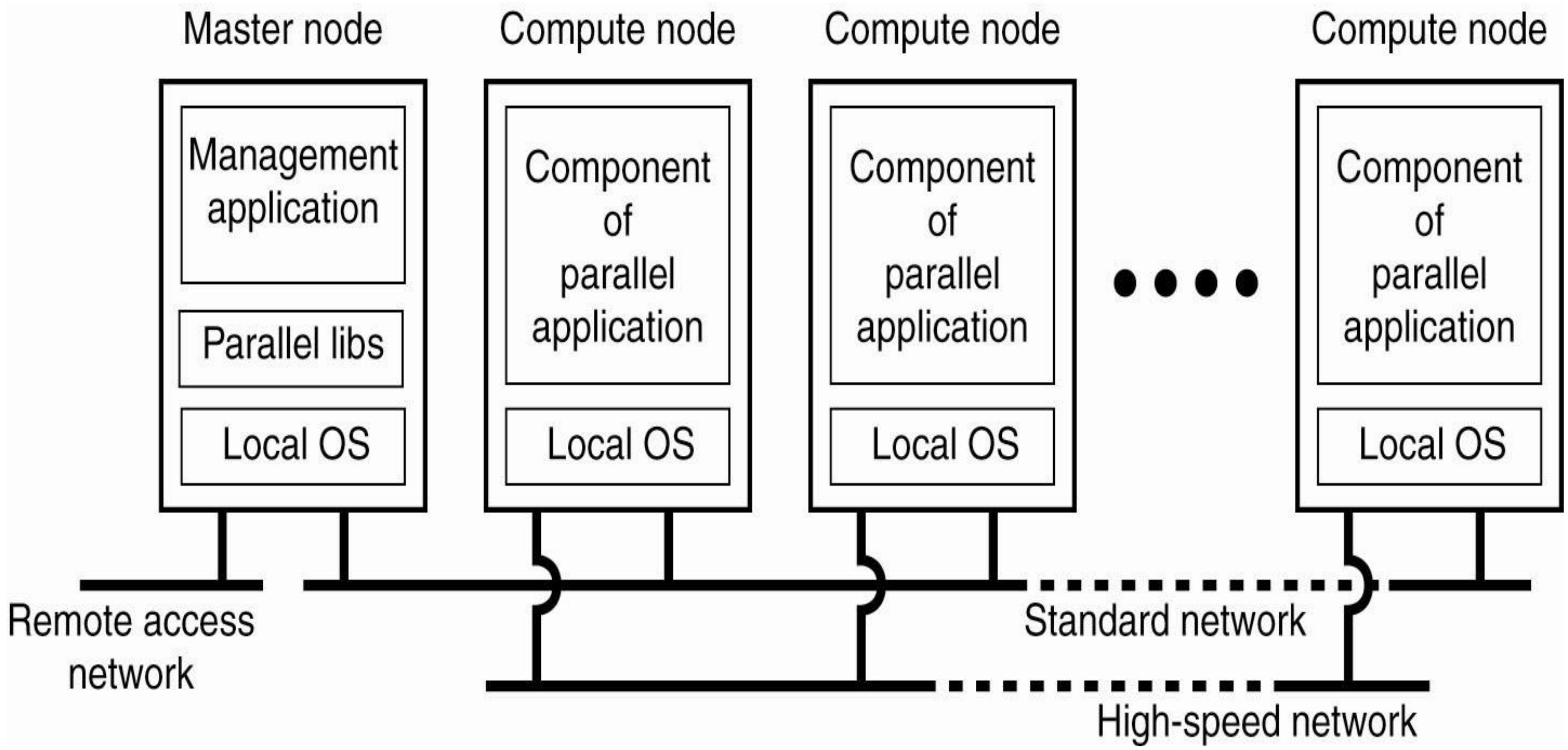
Cluster Computing Systems

- Collection of similar workstations/PCs, closely connected by means of a high-speed LAN:
  - Each node runs the same OS.
  - Homogeneous environment
  - Can serve as a supercomputer
  - Excellent for parallel programming
- Examples: Linux-based Beowulf clusters, MOSIX (from Hebrew University).

Clustered Systems Architecture



Architecture for Cluster Computing System

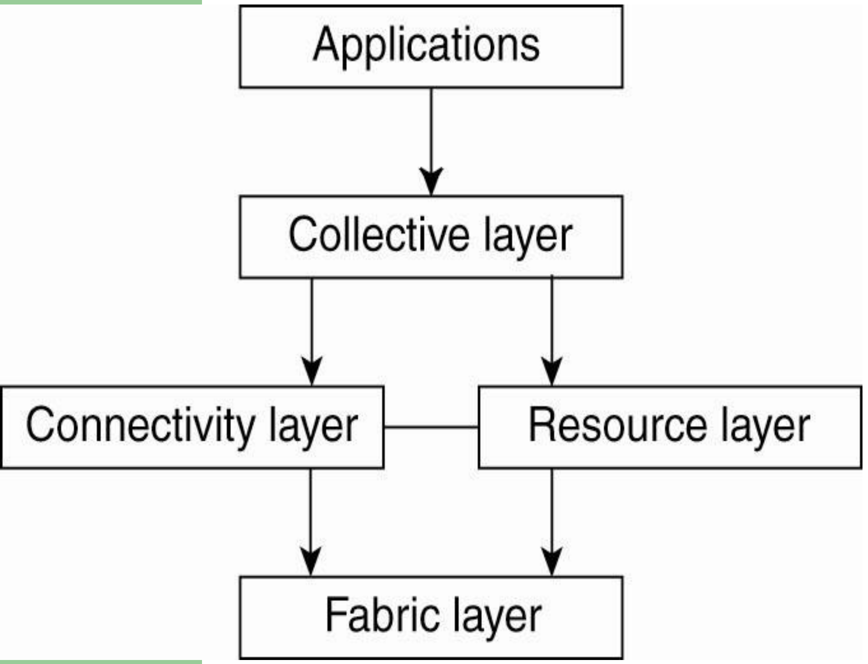


Grid Computing Systems

- Collection of computer resources, usually owned by multiple parties and in multiple locations, connected together such that users can share access to their combined power:
  - Can easily span a wide-area network
  - Heterogeneous environment
  - Crosses administrative/geographic boundaries
  - Supports Virtual Organizations (VOs)

– Examples: EGEE - Enabling Grids for E-SciencE (Europe), Open Science Grid (USA).

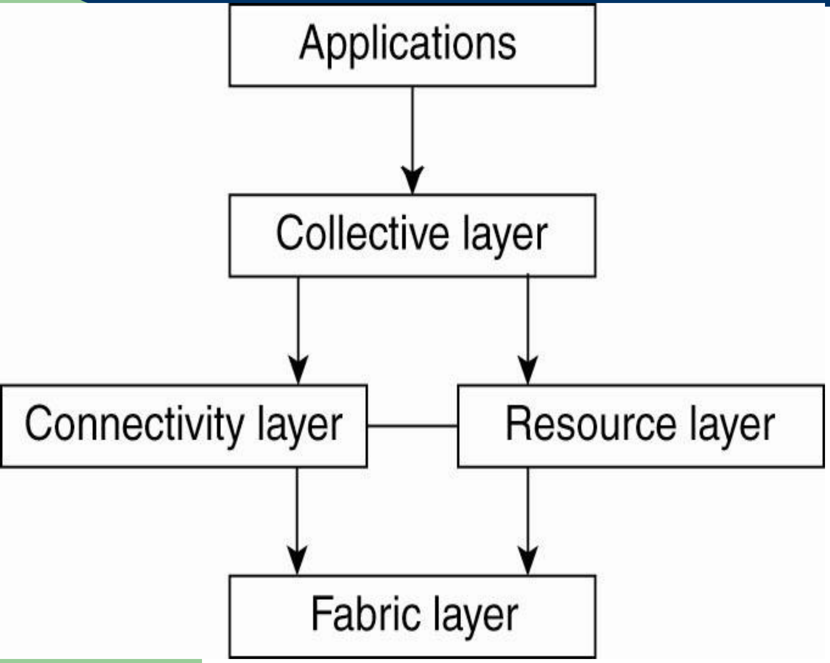
Architecture for Grid Computing Systems



- **Fabric layer** - Provides **interfaces** to local resources at a **specific site.**
  Eg: sharing of resources within a virtual organization.
- **Connectivity layer** –
  - Consists of **communication protocols** for supporting grid transactions that span the usage of multiple resources.
  -Contain **security protocols** to authenticate users and resources.
- **Resource layer**
  - For **managing a single resource.**
  - Responsible for **access control**

Hence will rely on the **authentication** performed as part of the connectivity layer.

Architecture for Grid Computing Systems (2)



- **Collective Layer**

  - Deals with **handling access to multiple resources** and typically consists of services for **resource discovery, allocation and scheduling** of tasks onto multiple resources, data replication, and so on.

- **Application layer**

  - Consists of the applications that **operate within a virtual organization** and which make use of the grid computing environment.

Typically the **collective, connectivity, and resource layer** form the **heart** of what could be called a grid middleware layer.

Cloud Computing Systems (1)

- Collection of computer resources, usually owned by a **single entity**, connected together such that **users can lease access to a share of their combined power:**
  - **Location independence**: the user can access the desired service from anywhere in the world, using any device with any (supported) system.
  - **Cost-effectiveness**: the whole infrastructure is owned by the provider and requires no capital outlay by the user.
  - **Reliability**: enhanced by way of multiple redundant sites, though outages can occur, leaving users unable to remedy the situation.

Cloud Computing Systems (2)

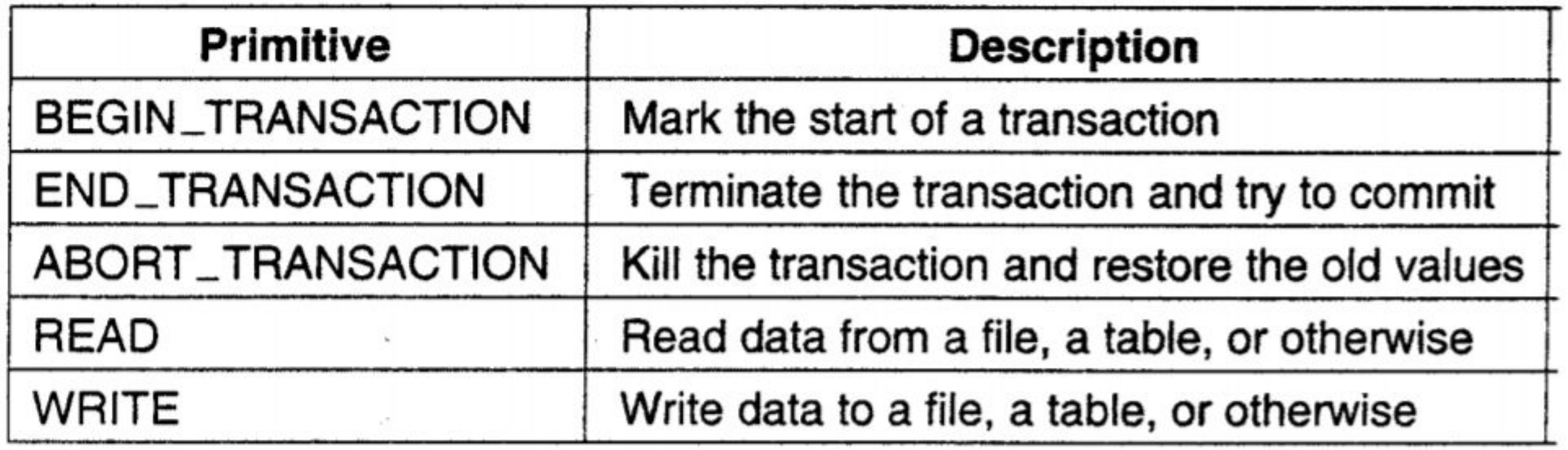
- **Scalability**: user needs can be tailored to available resources as demand dictates – cost benefit is obvious.
- **Security**: low risk of data loss thanks to centralization, though problems with control over sensitive data need to be solved.
- **Readily consumable**: the user usually does not need to do much deployment or customization, as the provided services are easy to adopt and ready-to-use.

- Examples: Amazon EC2 (Elastic Compute Cloud), Google App Engine, IBM Enterprise Data Center, MS Windows Azure, SUN Cloud Computing.

DISTRIBUTED INFORMATION SYSTEMS

Transaction Processing Systems (TPS)

- Operations on a **database** are usually carried out in the form of **transactions.**
- Programming using transactions requires **special primitives** that must either be supplied by the underlying distributed system or by the language runtime system.



Primitive	Description
BEGIN_TRANSACTION	Mark the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

ACID Properties

- The characteristic feature of a transaction is either all of these operations are executed or none are executed.
- This all-or-nothing property of transactions is one of the four characteristic properties that transactions have.
  - 1. Atomic: To the outside world, the transaction happens indivisibly.
  - 2. Consistent: The transaction does not violate system invariants.
  - 3. Isolated: Concurrent transactions do not interfere with each other.
  - 4. Durable: Once a transaction commits, the changes are permanent.

NESTED TRANSACTION

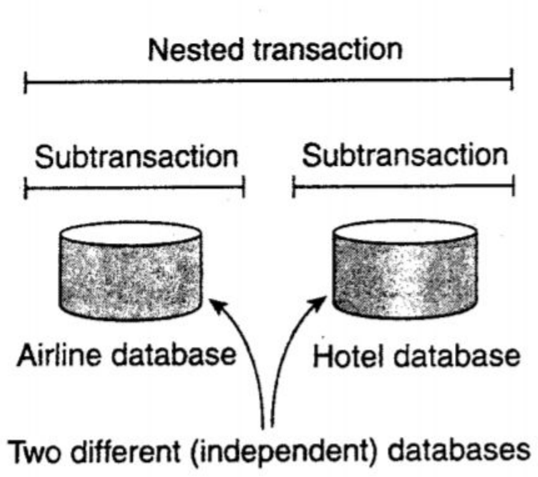
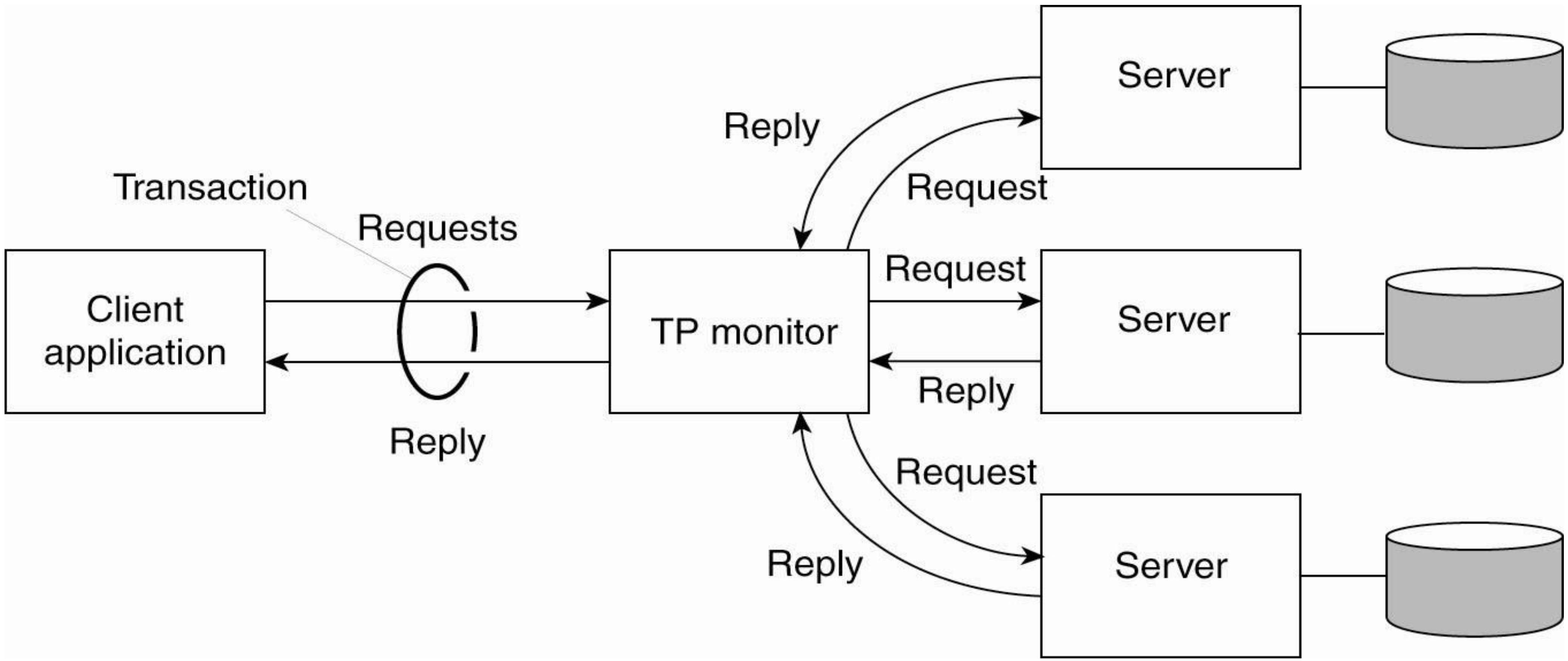


Figure 1-9. A nested transaction.

- The top-level transaction may fork off children that run in parallel with one another, on different machines, to gain performance or simplify programming.
- Each of these children may also execute one or more sub transactions, or fork off its own children.
- Provide a natural way of distributing a transaction across multiple machines.
- Follow a logical division of the work of the original transaction.

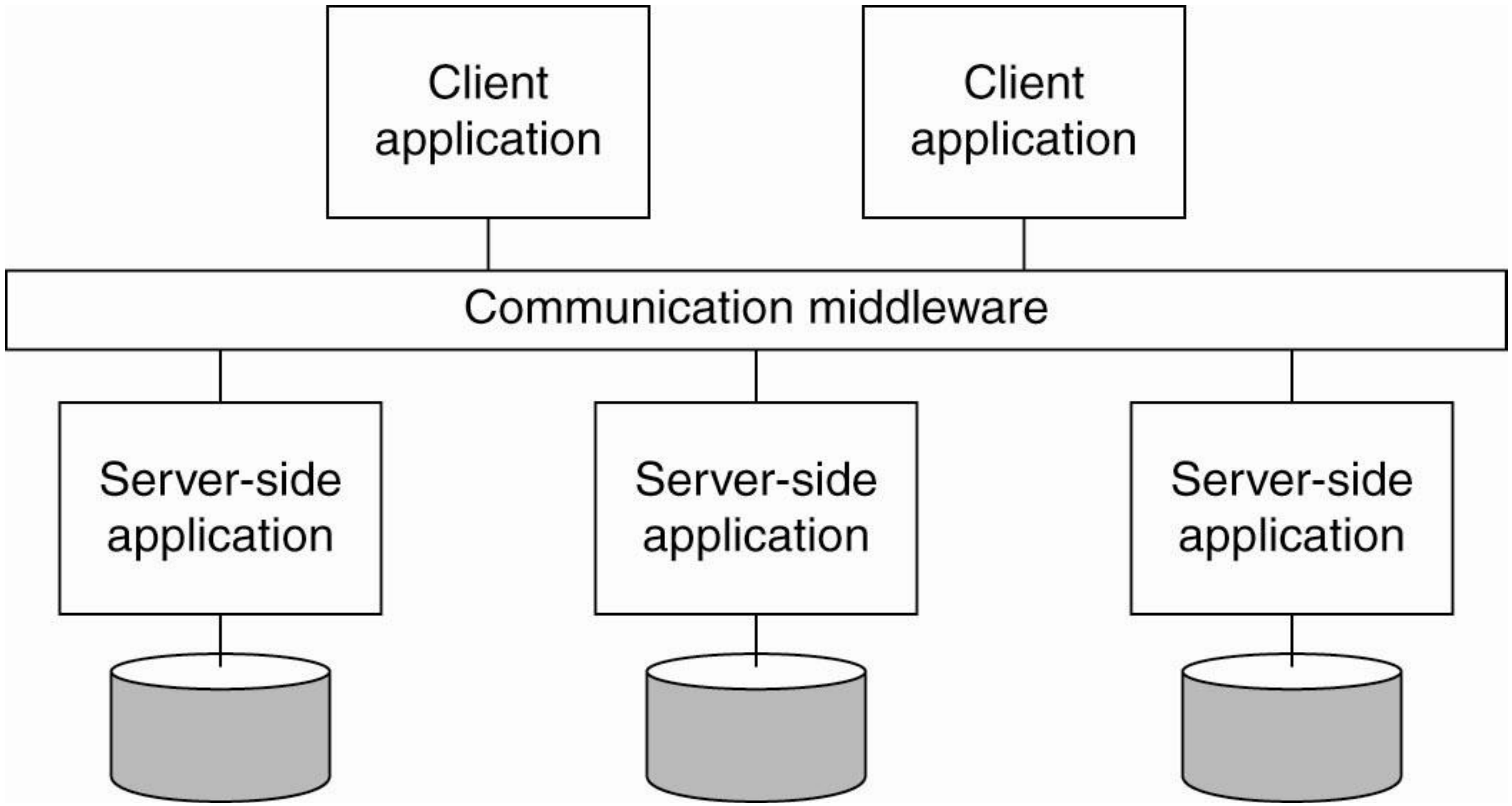
For example, a transaction for planning a trip by which three different flights need to be reserved can be logically split up into three sub transactions. Each of these sub transactions can be managed separately and independent of the other two.

Transaction Processing Systems (TPS)



The role of a TP monitor in distributed systems

Enterprise Application Integration



Distributed Pervasive Systems

- Characterized by being small, battery-powered, mobile, and having only a wireless connection.
- Devices generally join the system in order to access (and possibly provide) information.
- A sensor network typically consists of tens to hundreds or thousands of relatively small nodes, each equipped with a sensing device. Most sensor networks use wireless communication, and the nodes are often battery powered.

Distributed Pervasive Systems

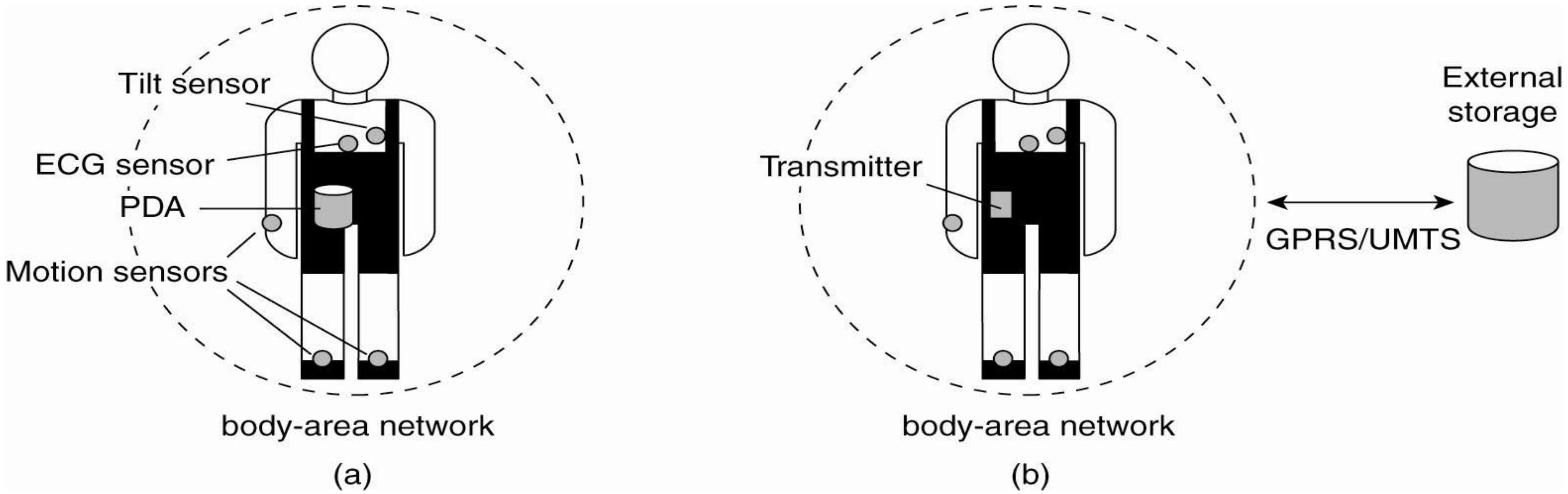
Used in

- Home Systems
- Electronic Health Care Systems
- Sensor Systems

Requirements for pervasive systems:

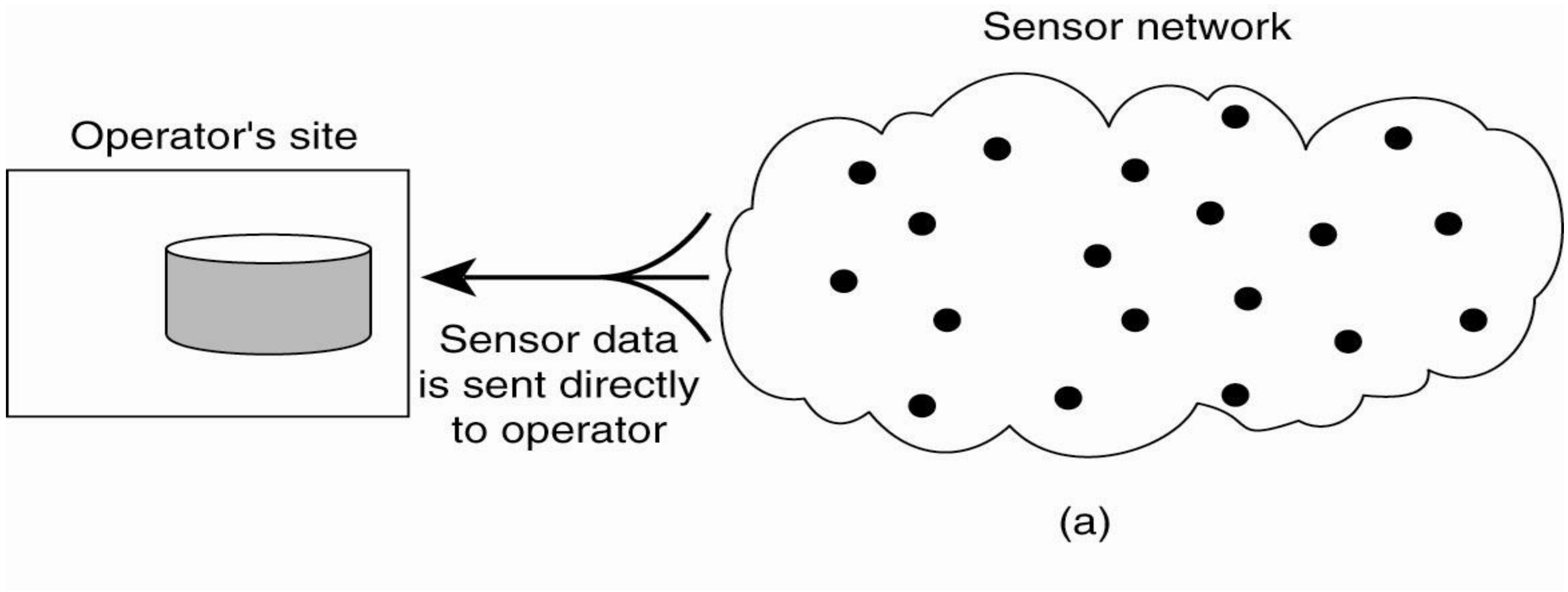
- Embrace contextual changes
- Encourage ad hoc composition
- Recognize sharing as the default
- Support distribution transparency

Electronic Health Care Systems (2)



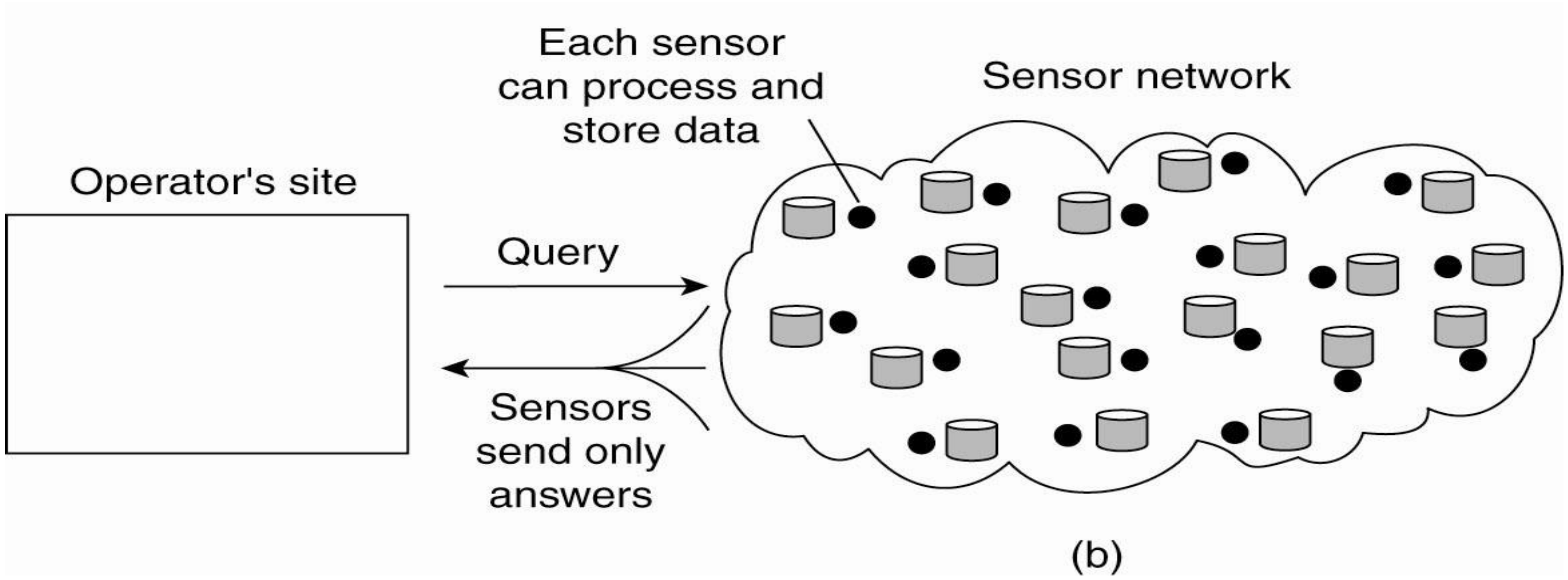
Monitoring a person in a pervasive electronic health care system, using (a) a local hub
or (b) a continuous wireless connection

Sensor Networks (2)



Organizing a sensor network database, while storing and processing data (a) only at the operator's site or …

Sensor Networks (3)


Each sensor can process and store data
Sensor network
Operator's site
Query
Sensors send only answers
(b)

Organizing a sensor network database, while storing and processing data … or (b) only at the sensors