Distributed Systems (3 Cr.)

Course Code: CACS352

Year/Semester: II/VI

Class Load: 4Hrs./Week

(Theory: 3Hrs. Tutorials: 1Hr)

By Prashant Gautam

Course Descriptions

→Give an insight on how modern distributed systems operate.

Objectives

→To make familiar with different aspect of distributed system, middleware, system level support and different issues in designing algorithms and finally systems.

Unit 1: Introduction (4 Hrs.)

Background

1.1 Characteristics

1.2 Design Goals

1.3 Types of Distributed Systems

1.4 Case Study: WWW

Background

- The pace at which computer systems change was, is, and continues to be overwhelming.
- 1945→ when the modern computer era began.
- 1945-1985→computers were large and expensive.
- Moreover, for lack of a way to connect them, these computers operated independently from one another.
- mid-1980s → Development of powerful microprocessors
  → invention of high-speed computer networks

Development of powerful microprocessors

- Initially, these were 8-bit machines, but soon 16-, 32-, and 64-bit CPUs became common.

Invention of high-speed computer networks

- **Local-area networks or LANs** allow thousands of machines within a building or campus to be connected in such a way that small amounts of information can be transferred in a few microseconds or so.
- Larger amounts of data can be moved between machines at rates of billions of bits per second (bps).
- **Wide-area networks or WANs** allow hundreds of millions of machines all over the earth to be connected at speeds varying from tens of thousands to hundreds of millions bps, and sometimes even faster.

Smartphone as the most impressive outcome

- Packed with sensors, lots of memory, and a powerful CPU, these devices are nothing less than full-fledged computers.
- Of course, they also have networking capabilities.

plug computers and nano computers

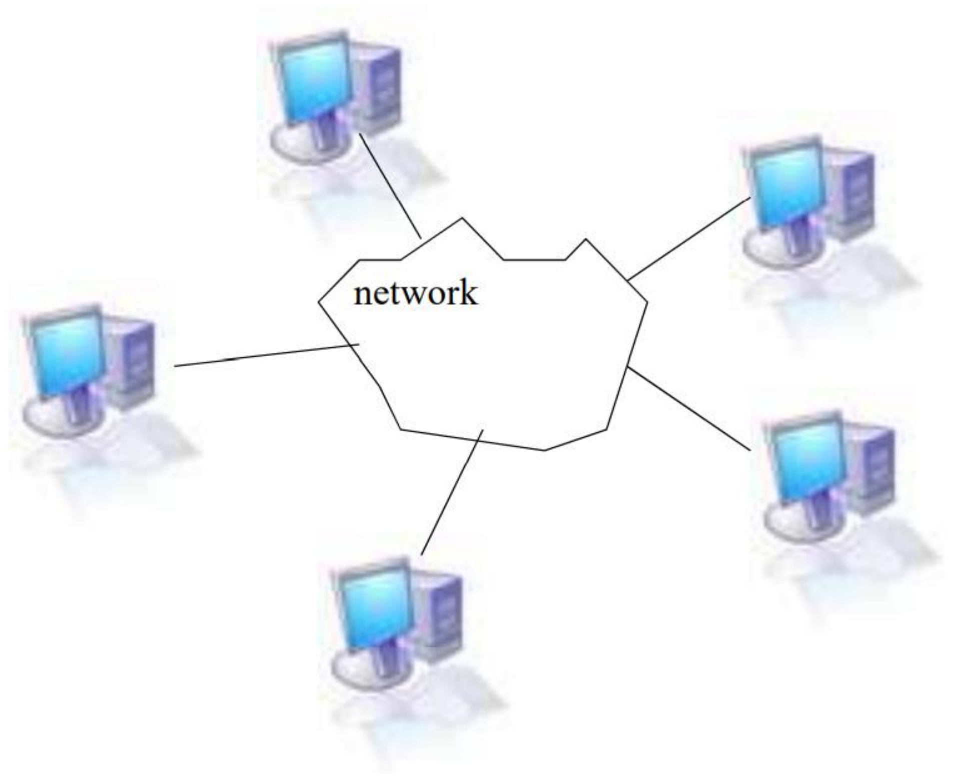
- These small computers, often the size of a power adapter, can often be plugged directly into an outlet and offer near-desktop performance.

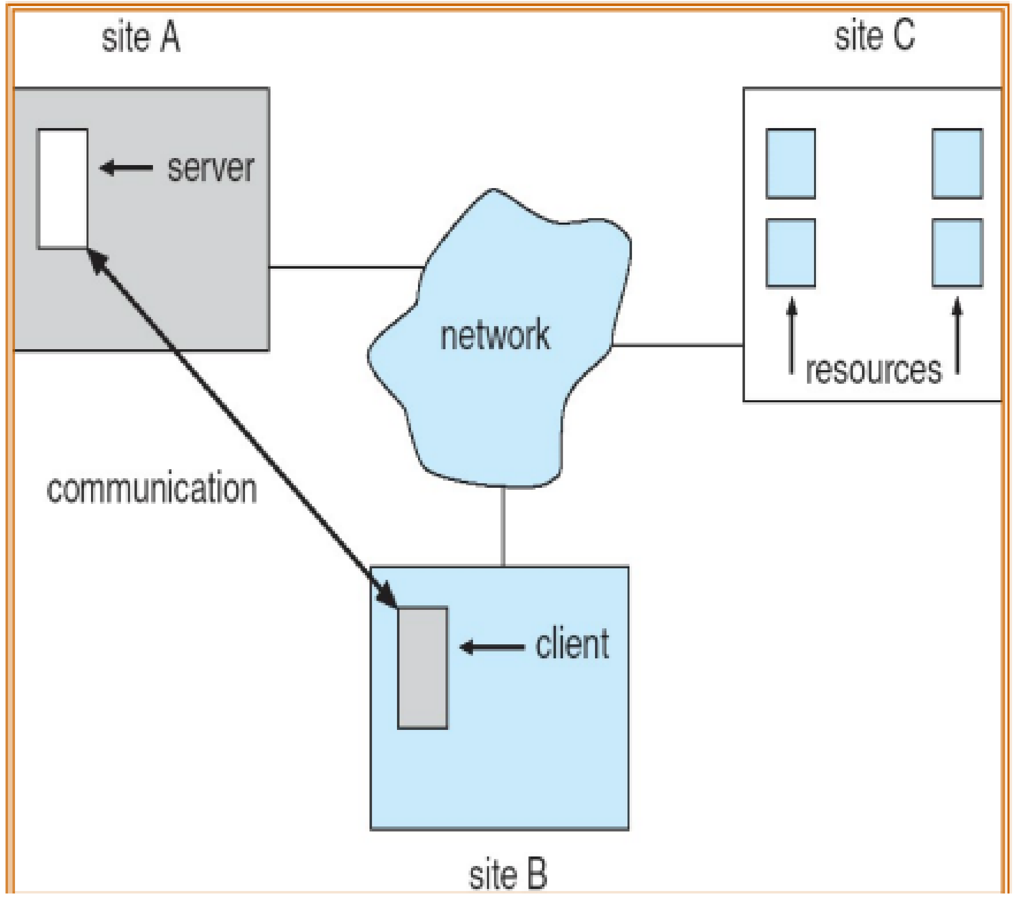
Distributed System

- The size of a distributed system may vary from a handful of devices, to millions of computers.
- The interconnection network may be wired, wireless, or a combination of both.
- Moreover, distributed systems are often highly dynamic, in the sense that computers can join and leave, with the topology and performance of the underlying network almost continuously changing.

What is a distributed system?

- A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system.
    - a collection of computing elements(Node: H/w device or S/w Process) each being able to behave independently of each other.
    - users (people or applications) believe they are dealing with a single system.





1. Group of autonomous hosts, each host executes components and operates distribution applications.
2. Hosts are Geographically dispersed/separated over (LAN, WAN,……)
3. Hosts Connected via a network
4. The network is used to: transfer messages and mail, and execute applications: airline reservation, stock control, ….)

1.1. Characteristic 1: collection of autonomous computing elements

- In a DS there are multiple components that may be decomposed further.
- These components are autonomous, i.e. they possess full control over their parts at all times.

1.1 Characteristic 2: single coherent system

- In a single coherent system the collection of nodes as a whole operates the same, no matter where, when, and how interaction between a user and the system takes place.

Middleware and distributed systems

- To assist the development of distributed applications, 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 shown

in Fig. 1, leading to what is known as middleware.

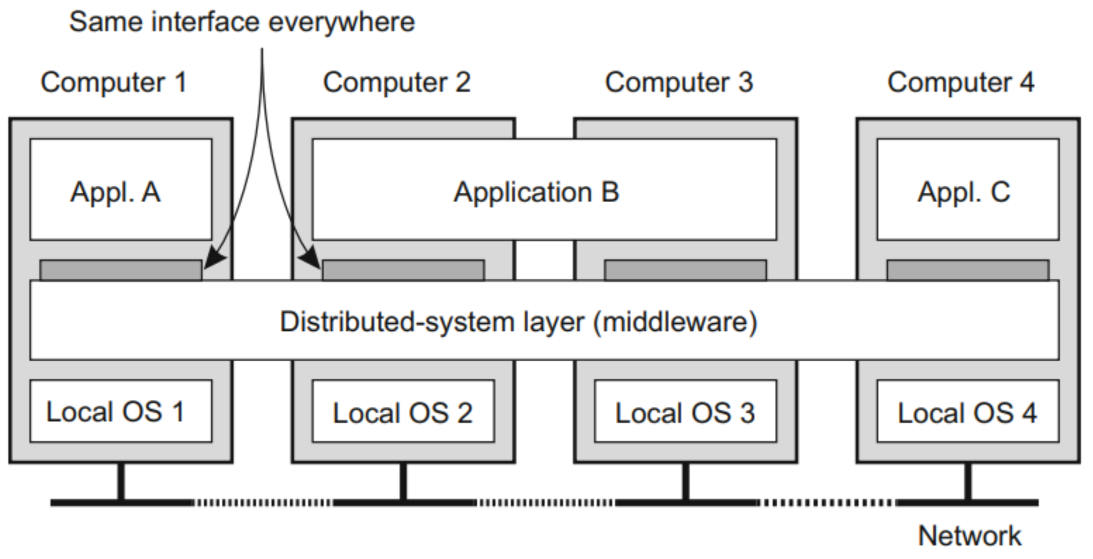


Fig. 1. A distributed system organized as middleware. The middleware layer extends over multipl
machines, and offers each application the same interface

1.2 Design Goals

- Making Resources Accessible
- Distribution Transparency
- Openness
- Scalability

1.2.1 Making Resources Accessible

- Goal → to make it easy for the users (and applications) to access remote resources, and to share them in a controlled and efficient way.
- Resources→ printers, computers, storage facilities, data, files, Web pages, and networks, etc.

Why Resources Sharing ?

- Economic →

It is cheaper to let a printer be shared by several users in a small office than having to buy and maintain a separate printer for each user.

Likewise, it makes economic sense to share costly resources such as supercomputers, high-performance storage systems, imagesetters, and other expensive peripherals.

Security Concern

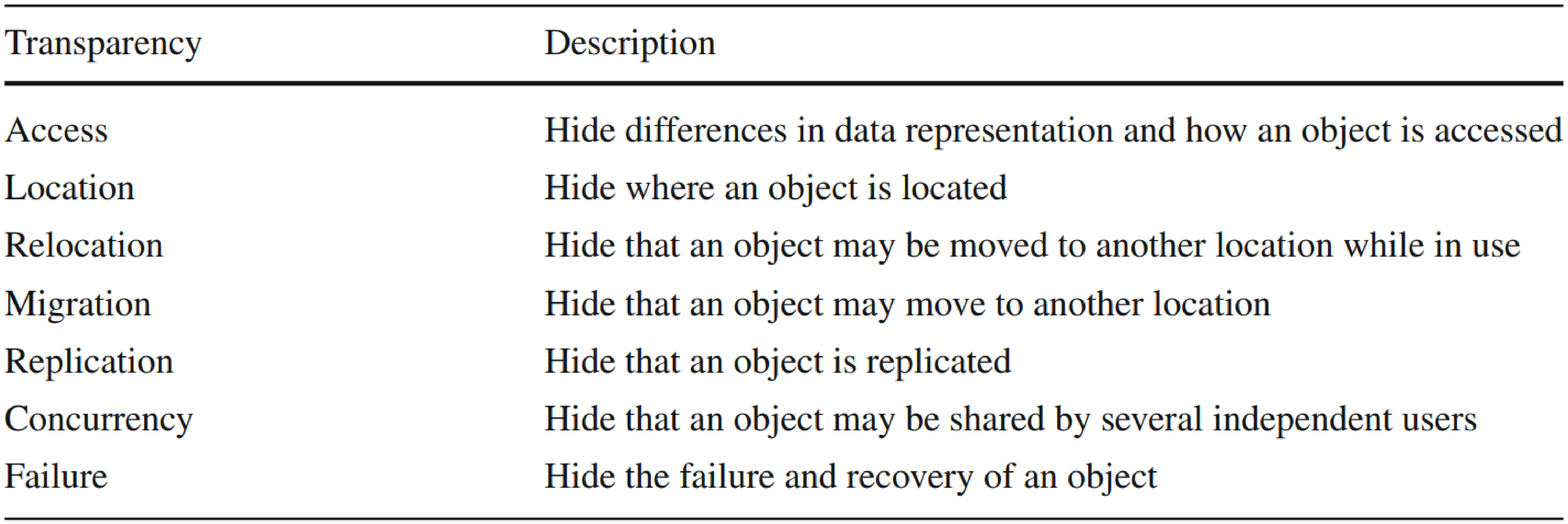
- As connectivity and sharing increase, security is becoming increasingly important.
- There is much room for improvement.
- Technique such as cryptographic encryption can be used while sharing contents.

1.2.2 Distribution Transparency

- Goal → to hide the fact that its processes and resources are physically distributed across multiple computers.
- In other words, it tries to make the distribution of processes and resources transparent, that is, invisible, to end users and applications.

Types of Transparency

- The concept of transparency can be applied to several aspects of a distributed system, the most important ones shown below:



Transparency	Description
Access	Hide differences in data representation and how an object is accessed
Location	Hide where an object is located
Relocation	Hide that an object may be moved to another location while in use
Migration	Hide that an object may move to another location
Replication	Hide that an object is replicated
Concurrency	Hide that an object may be shared by several independent users
Failure	Hide the failure and recovery of an object

Access Transparency

- For example, a distributed system may have computer systems that run different operating systems, each having their own file-naming conventions. Differences in naming conventions, differences in file operations, or differences in how low-level communication with other processes is to take place, are examples of access issues that should preferably be hidden from users and applications.

Location Transparency

- An example of a such a name is the uniform resource locator (URL) http://www.distributed-systems.net/index.php, which gives no clue about the actual location of the site’s Web server. The URL also gives no clue as to whether the file index.php has always been at its current location or was recently moved there. F

Relocation Transparency

- For example, the entire site may have been moved from one (part of a) data center to another to make more efficient use of disk space, yet users should not notice.

Migration Transparency

- A typical example is communication between mobile phones: regardless whether two people are actually moving, mobile phones will allow them to continue their conversation.

Replication transparency

- For example, resources may be replicated to increase availability or to improve performance by placing a copy close to the place where it is accessed.

concurrency transparency

- For example, two independent users may each have stored their files on the same file server or may be accessing the same tables in a shared database. In such cases, it is important that each user does not notice that the other is making use of the same resource.

Failure Transparency

- For example, when contacting a busy Web server, a browser will eventually time out and report that the Web page is unavailable. At that point, the user cannot tell whether the server is actually down or that the network is badly congested.

1.2.3 Openness

- Open Distributed System →system that offers services according to standard rules that describe the syntax and semantics of those services.
- For example, in computer networks, standard rules govern the format, contents, and meaning of messages sent and received. Such rules are formalized in protocols. In distributed systems, services are generally specified through interfaces, which are often described in an Interface Definition Language (IDL).

1.2.4 Scalability

- Scalability of a system can be measured along at least three different dimensions:
  - **A system can be scalable with respect to its size**

    add more users and resources to the system
  - **A geographically scalable system**

    add more users and resources to the system
  - **A system can be administratively scalable**

    it can still be easy to manage even if it spans many independent administrative organizations.

1.2.5 Pitfalls when developing DS

- False assumptions that everyone makes when developing a distributed application for the first time:
    1. The network is reliable.
    2. The network is secure.
    3. The network is homogeneous.
    4. The topology does not change.
    5. Latency is zero.
    6. Bandwidth is infinite.
    7. Transport cost is zero.
    8. There is one administrator

1.3 Types of Distributed Systems

1- Distributed Computing Systems : Focus on computation

--Cluster Computing Systems
--Grid Computing Systems

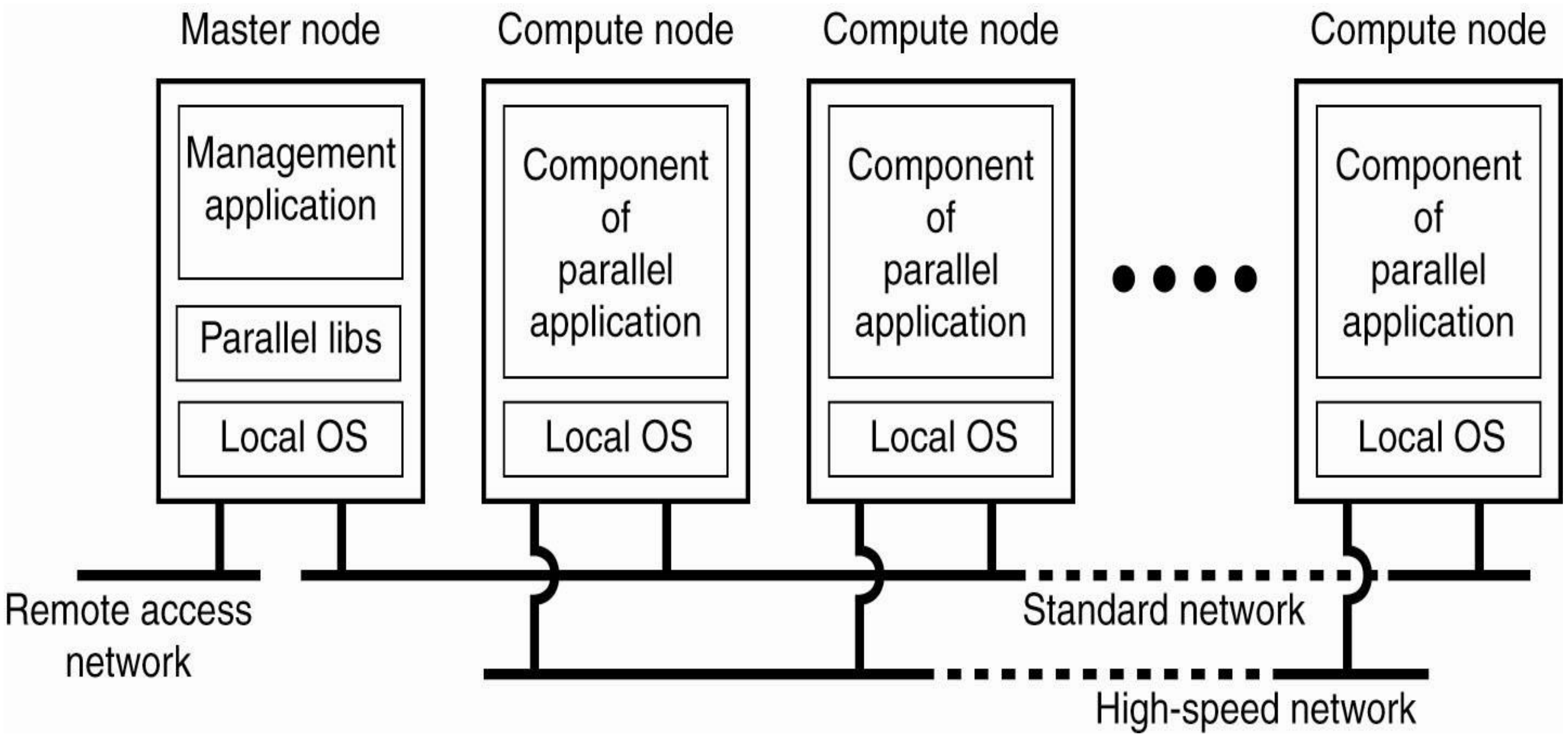
2- Distributed Information Systems: Focus on interoperability

--Transaction Processing Systems
--Enterprise Application Integration
(Exchange info via RPC or RMI)

3- Distributed Pervasive Systems (usually small, battery-powered systems, Mobile & wireless): Focus on mobile, embedded, communicating

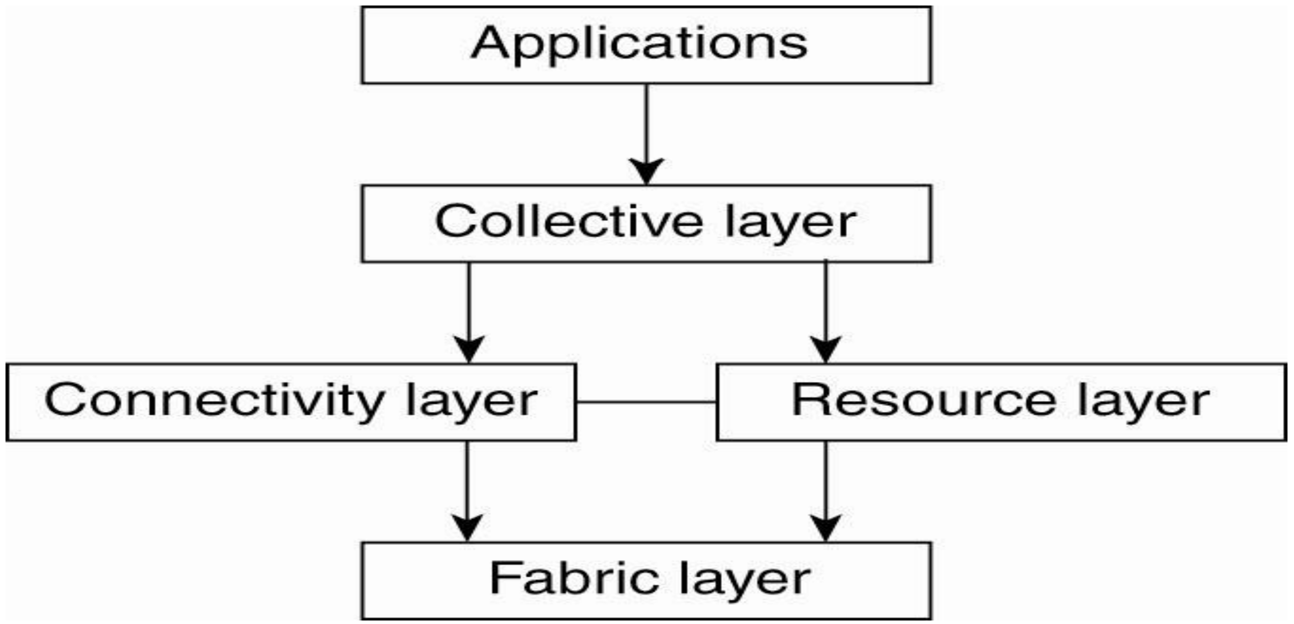
--Home Systems (e.g. Smart phones, PDAs)
--Electronic Health care systems (Heart monitors, BAN: Body Area Networks)
--Sensor Networks (distributed Databases connected wirelessly)

Cluster Computing Systems



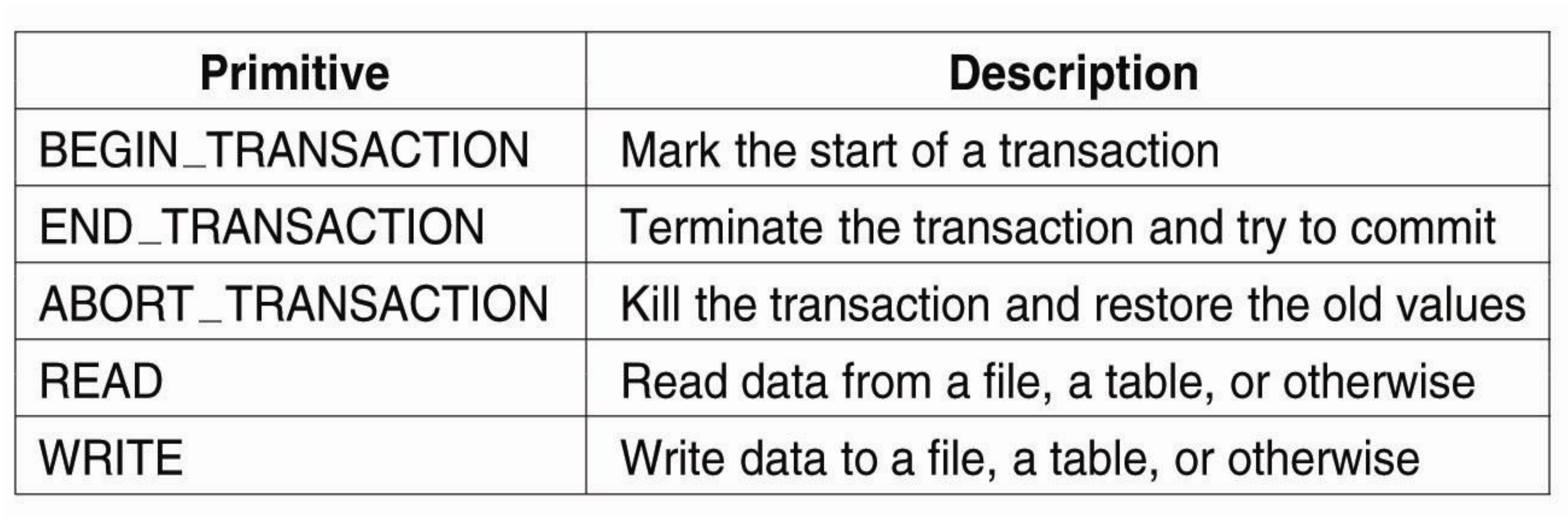
- **Hooking up a collection of simple computers via high-speed networks to build a supercomputing environment**
- **Mostly homogenous**
- **Example: server clusters at Banks**

Grid Computing Systems



- have a high degree of heterogeneity
-Users and resources from different organizations are brought together to allow collaboration (i.e. a V.O. = Virtual Organization)
- Members belonging to the same V.O. have access rights to a common set of resources (e.g. Police and some local agencies may form a computing grid)

Transaction Processing Systems (1)



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

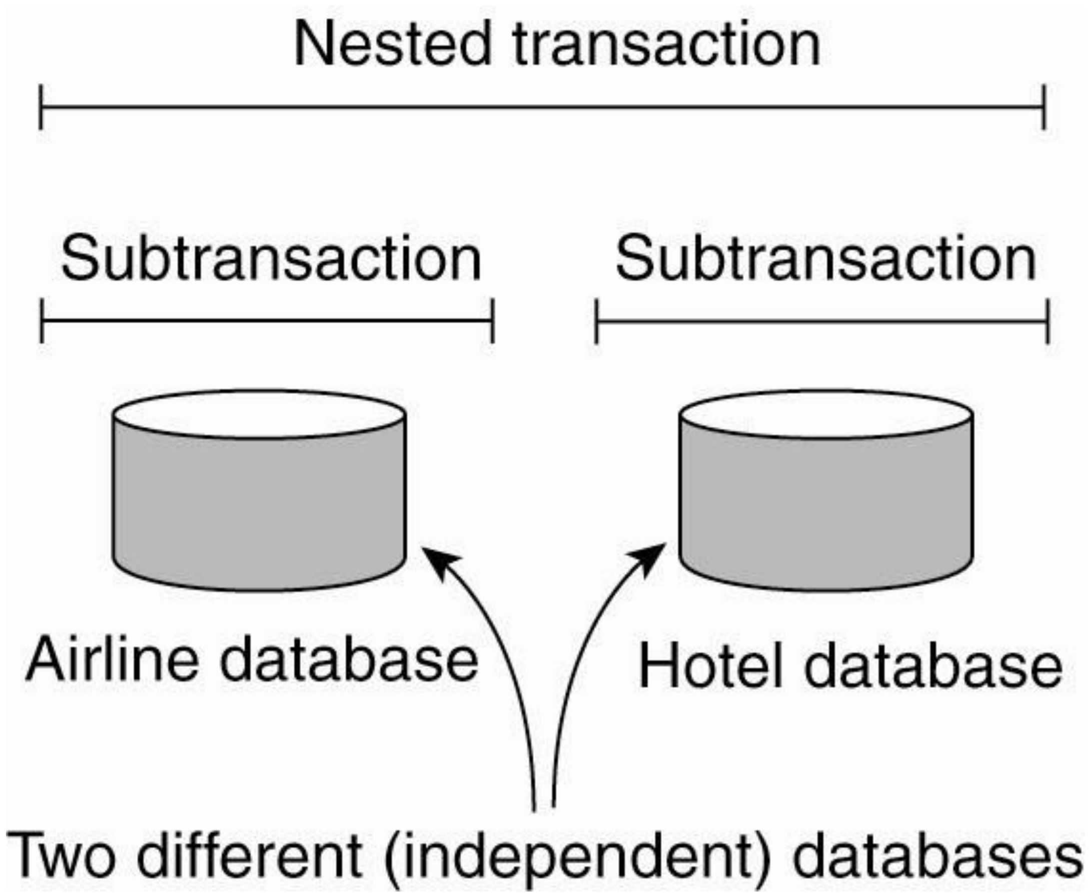
```
BEGIN_TRANSACTION
salary1 = doctor1.getSalary()
doctor1.setSalary(salary1 + bonus)
salary2 = doctor2.getSalary()
doctor2.setSalary(salary2 - bonus)
END_TRANSACTION
```

Transaction Processing Systems (2)

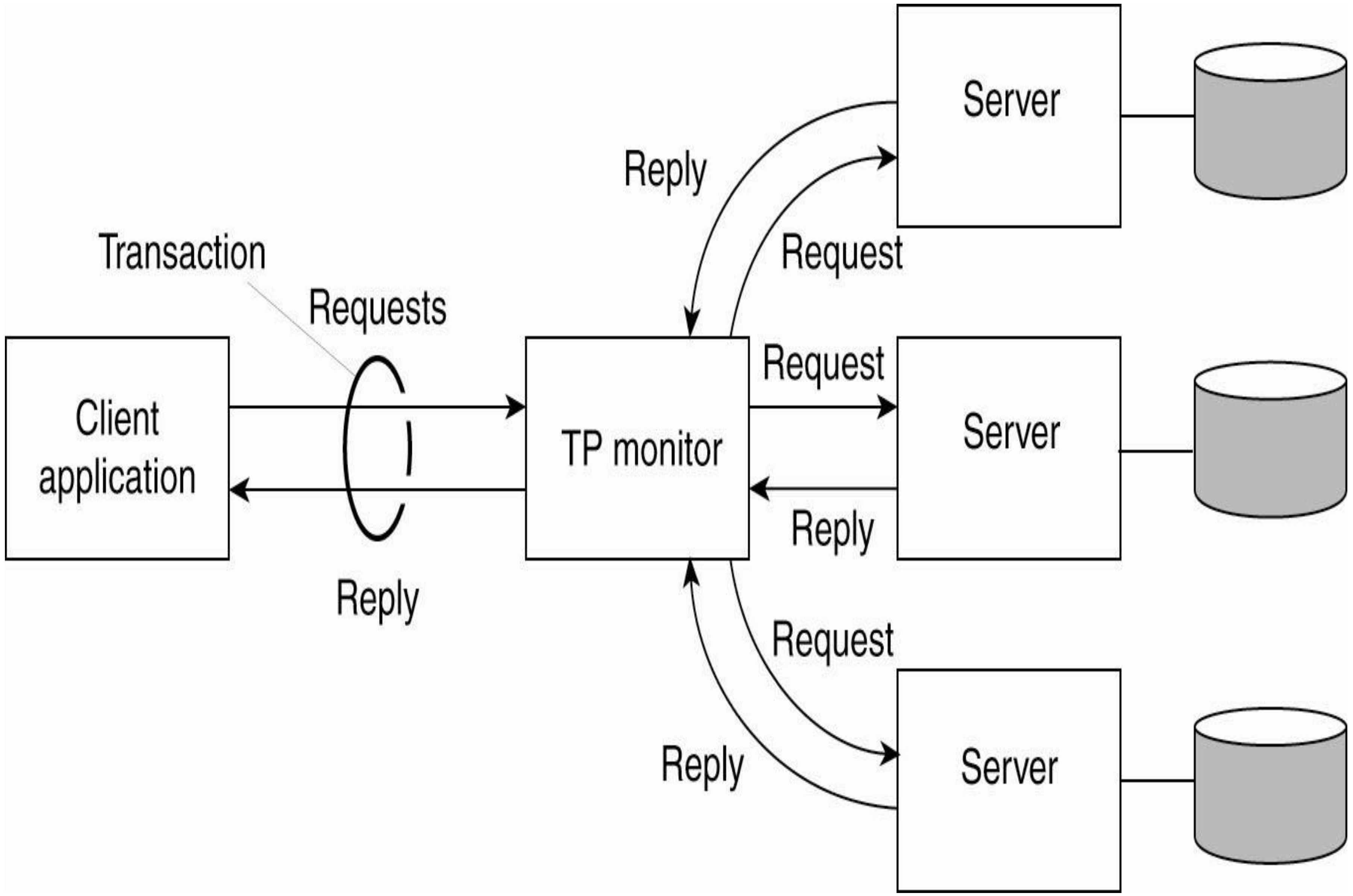
Characteristic:

- Atomic: The transaction happens indivisibly.
- Consistent: The transaction does not violate system invariants.
- Isolated: Concurrent transactions do not interfere with each other.
- Durable: Once a transaction commits, the changes are permanent.

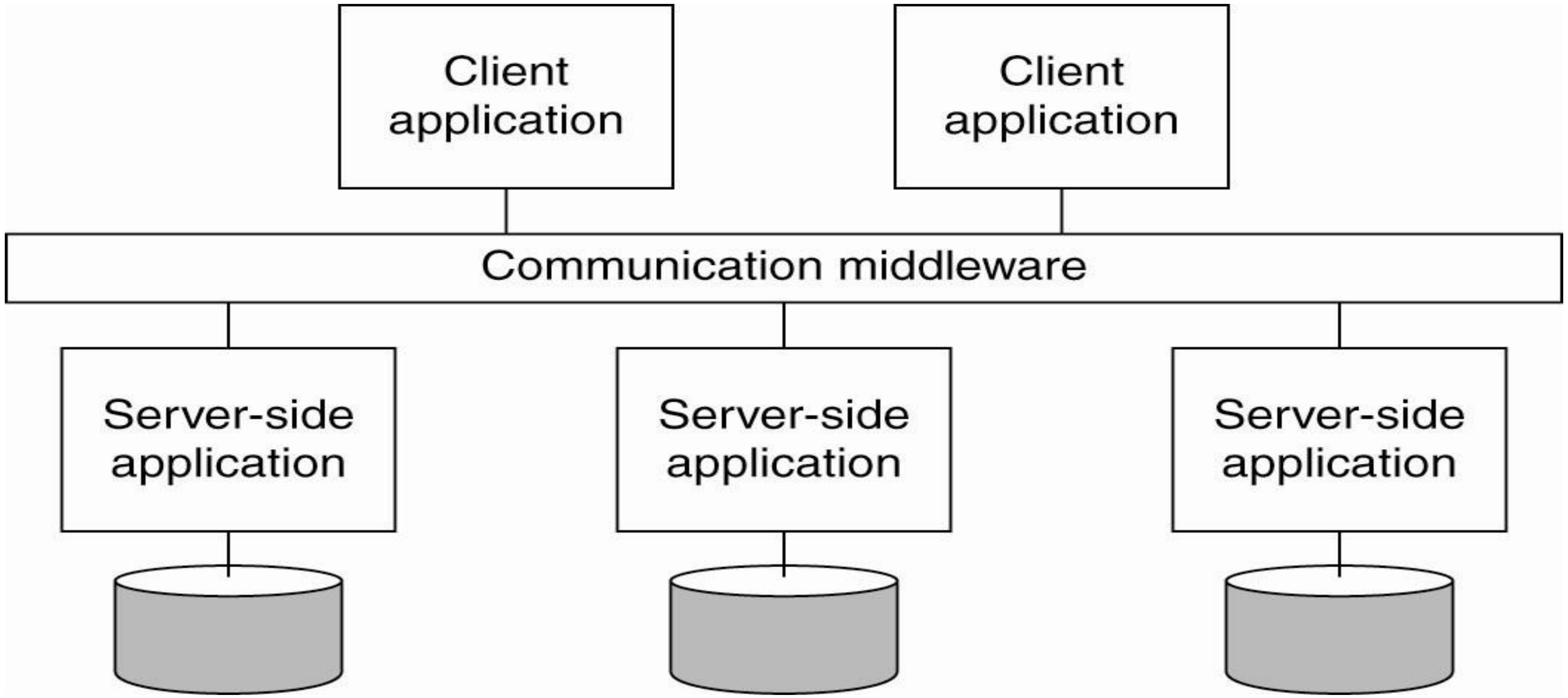
Transaction Processing Systems (3)



Transaction Processing Systems (4)



Enterprise Application Integration



-RPC (Remote Procedure Call)
-RMI (Remote Method Invocation)

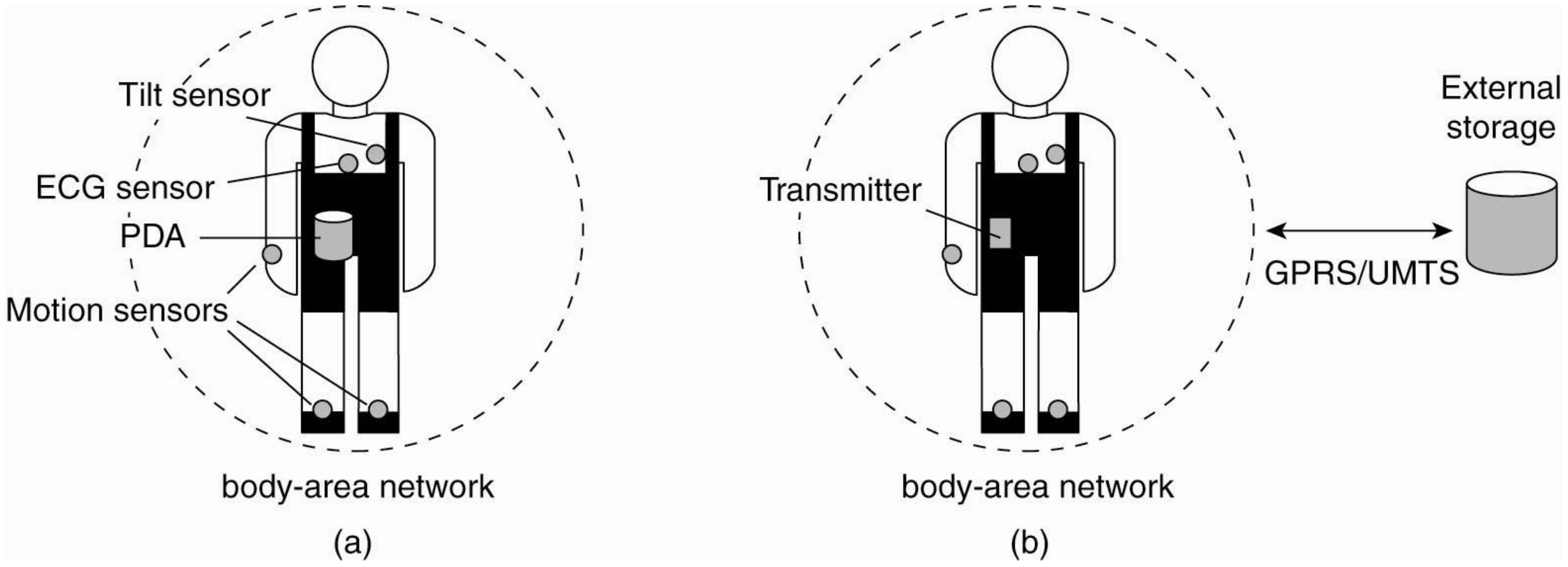
Distributed Pervasive Systems

- Embrace contextual changes
- (a phone now is a web access device. A device must continuously be aware of the fact that its environment may change)
- Encourage ad hoc composition
- (used differently by different users, e.g. PDA)
- Recognize sharing as the default
- (easily read, store, manage, and share info)

Electronic Health Care Systems (1)

- Where and how should monitored data be stored?
- How can we prevent loss of crucial data?
- What infrastructure is needed to generate and propagate alerts?
- How can physicians provide online feedback?
- How can extreme robustness of the monitoring system be realized?
- What are the security issues and how can the proper policies be enforced?

Electronic Health Care Systems (2)

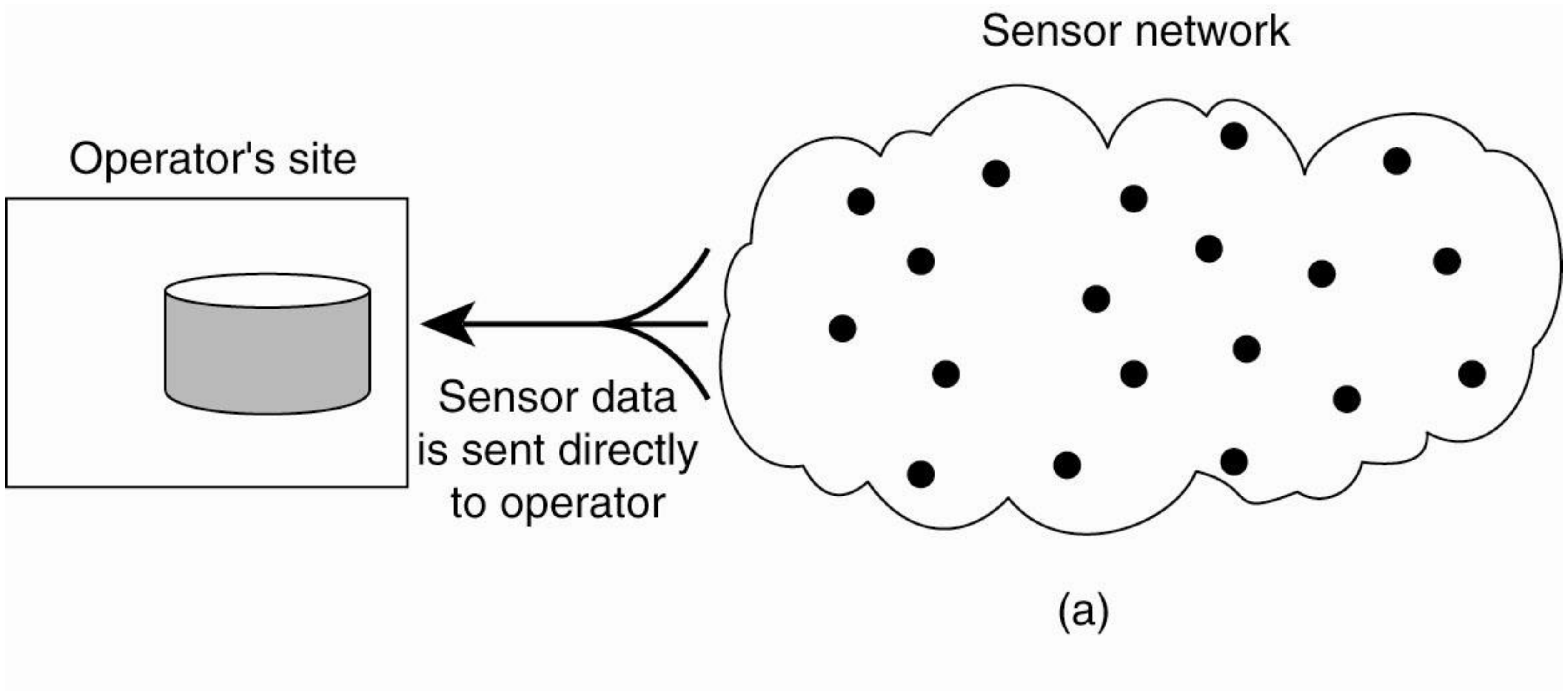


Monitoring a person in a pervasive electronic health care system, using (a) a local hub or - (b) a continuous wireless connection.

Sensor Networks (1)

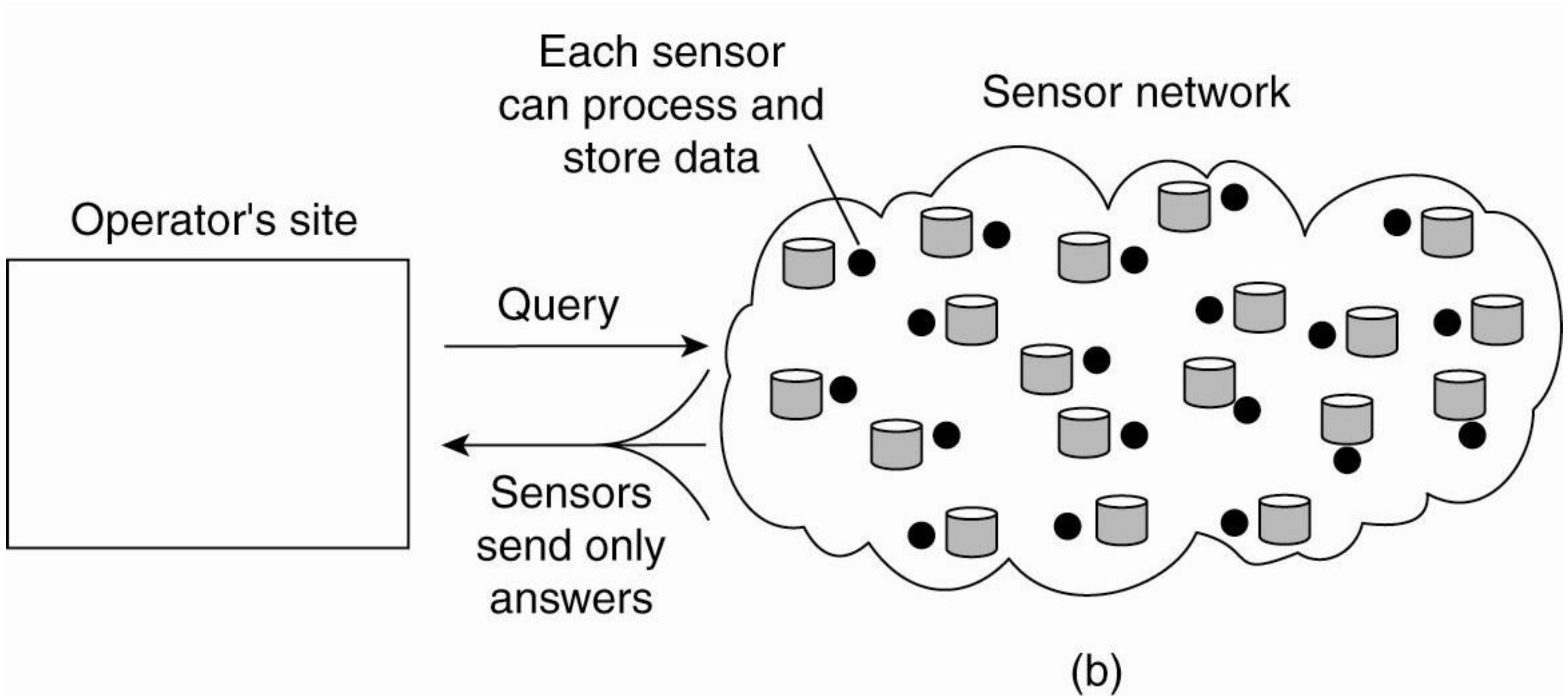
- How do we (dynamically) set up an efficient tree in a sensor network?
- How does aggregation of results take place? Can it be controlled?
- What happens when network links fail?

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.