DISTRIBUTED SYSTEMS
Principles and Paradigms
Second Edition
ANDREW S. TANENBAUM
MAARTEN VAN STEEN

Chapter 1
Introduction

*** Modified by Prof. Rivalino Matias, Jr.**

History

- From 1945 to approx. 1985, computers were big and very expensive to buy, maintain and operate.
- Few organizations had computers, which worked independently.
- In mid of 1980’s, two technology advances changed this scenario:
  - Development of high-capacity microprocessors (8, 16, 32, and 64 bits)
  - Invention of computer networks (LAN, MAN, WAN, …)
- As a result, it became possible to connect multiple computers and build networked systems (or distributed systems).

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.

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.

Definition of a Distributed System (2)

This definition has very important aspects:

- A DS is composed of autonomous computers.
- The users (people or programs) don't know they are dealing with multiple computer systems.
  - This means that the autonomous systems must cooperate!
  - How to establish this cooperation is a key aspect of developing distributed systems.
- Also, there is no assumption on the computer types of the autonomous systems (e.g., mainframes to sensor nodes)
- Likewise, there is no assumption on the way the autonomous computers are interconnected.

Definition of a Distributed System (3)

Important characteristics of DS:

- The differences between the computers and the way they are interconnected are, mostly, hidden from users.
  - The same applies to the internal organization of DS.
- Users (people and programs) can interact with the DS in a consistent and uniform way, regardless where these interactions occur.
- Increase the whole system scalability must be easy, as a result of using independent computers.
- Availability is also an expected property, although some parts may be temporarily offline.
  - The faulty and replaced parts should not be perceived by users.

Definition of a Distributed System (4)

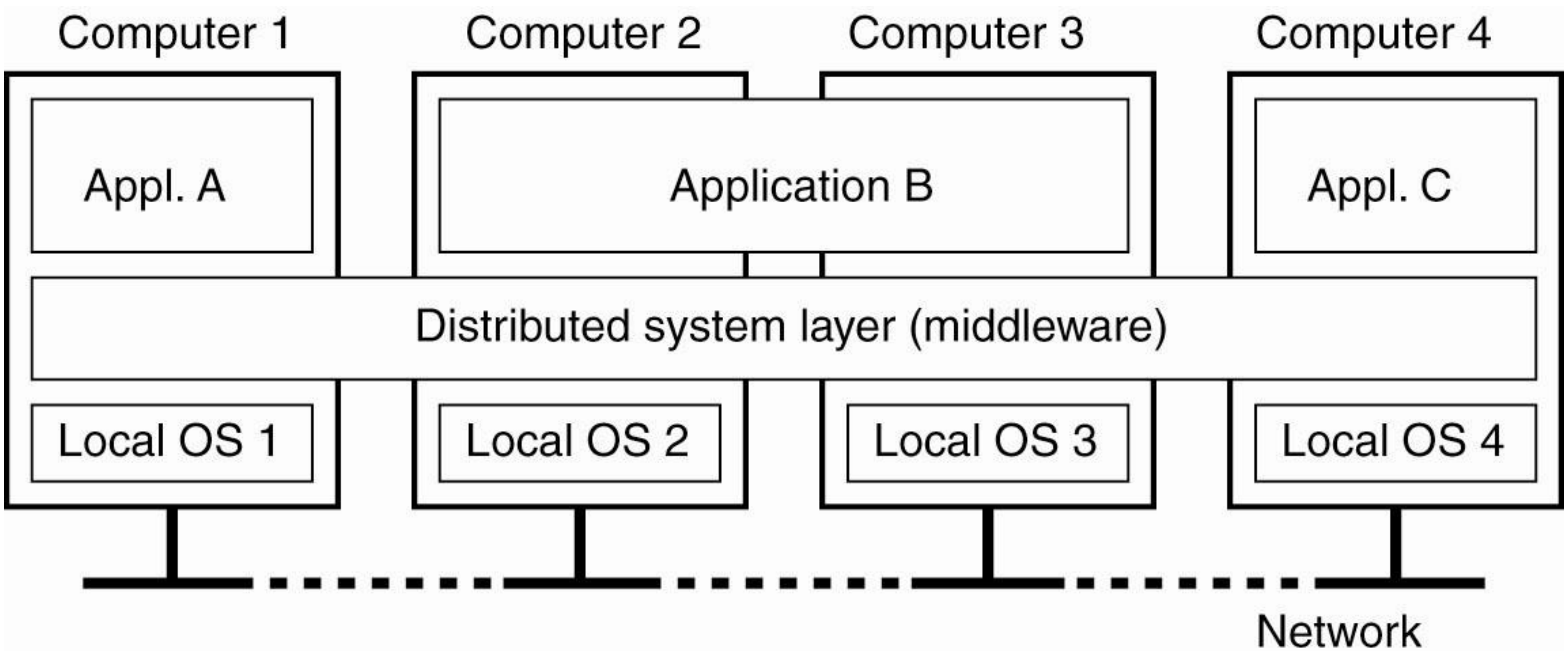

Figure 1-1. A distributed system organized as middleware. The middleware layer extends over multiple machines, and offers each application the same interface.

Goals

- Making resources available.
- Distribution transparency.
- Openness.
- Scalability.

Goals

- **Making resources available.**
  - This is the main goal of a distributed systems.
  - Among the many reasons, economy is the most obvious.
  - To connect users and resources makes information sharing and collaboration easier. The Internet is an example.
  - As connectivity and data/information sharing increase, security becomes even more important.

Goals

- Making resources available.
- **Distribution transparency**.
  - A distributed system that can present itself to users (people and programs) as a single computer system is called *transparent*.
  - There are many types of transparency in distributed systems (see next slide).

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

Figure 1-2. Different forms of transparency in a distributed system (ISO, 1995).

Goals

- Making resources available.
- Distribution transparency.
- **Openness**.
  - An open DS is a system that offers its services according to standard rules (syntax and semantics).
  - Usually, the services are specified by means of interfaces, which are commonly described in an IDL (interface description language).
  - An adequate interface specification is complete and neutral.
  - Interoperability and portability are also important.
    - The former indicates the extent to which two systems or components implementations, from different providers, may coexist and work.
    - The latter characterizes the extent an application developed for a DS *A* can be executed, w/o modification, on a DS *B* that implements the same interface implemented by *A*.

Goals

- Making resources available.
- Distribution transparency.
- Openness.
- **Scalability**.
  - This is a very important property of a DS.
  - The scalability can be measured considering three dimensions: size, geographic, and administrative.
  - Sometimes, meeting two or more of these dimensions imply in loss of performance.
  - Growing a system for each one of these dimensions require to solve different scalability problems (see next slide).

Scalability Problems

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

Figure 1-3. Examples of scalability limitations.

Scalability Problems

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.

Geographic scalability

- Many systems were developed to execute in a LAN, and thus are based on *synchronous* communication.
  - Building interactive applications to be executed over WANs requires a different and careful design.
- Communication over WANs is inherited not reliable, and most of times peer-to-peer.
  - e.g., broadcast as used in LANs is not an effective option!
- Sharing resources over WANs is most of time not feasible.
- Different administrative domains create important challenges (rules, policies, security, …).

Scaling Techniques (1)

- Mostly, scalability problems appear as performance issues caused by limited capacity of servers and networks.
- There are three techniques to scale systems:
  - Hiding communication latency, distribution, and replication.
- **Hiding latency**: avoid as much as possible to wait, synchronously, for remote replies.
  - Adopting asynchronous communication is an option,
  - however, it is an issue for interactive applications; which can be mitigated moving part of the server-side processing to the client side.

Scaling Techniques (1)

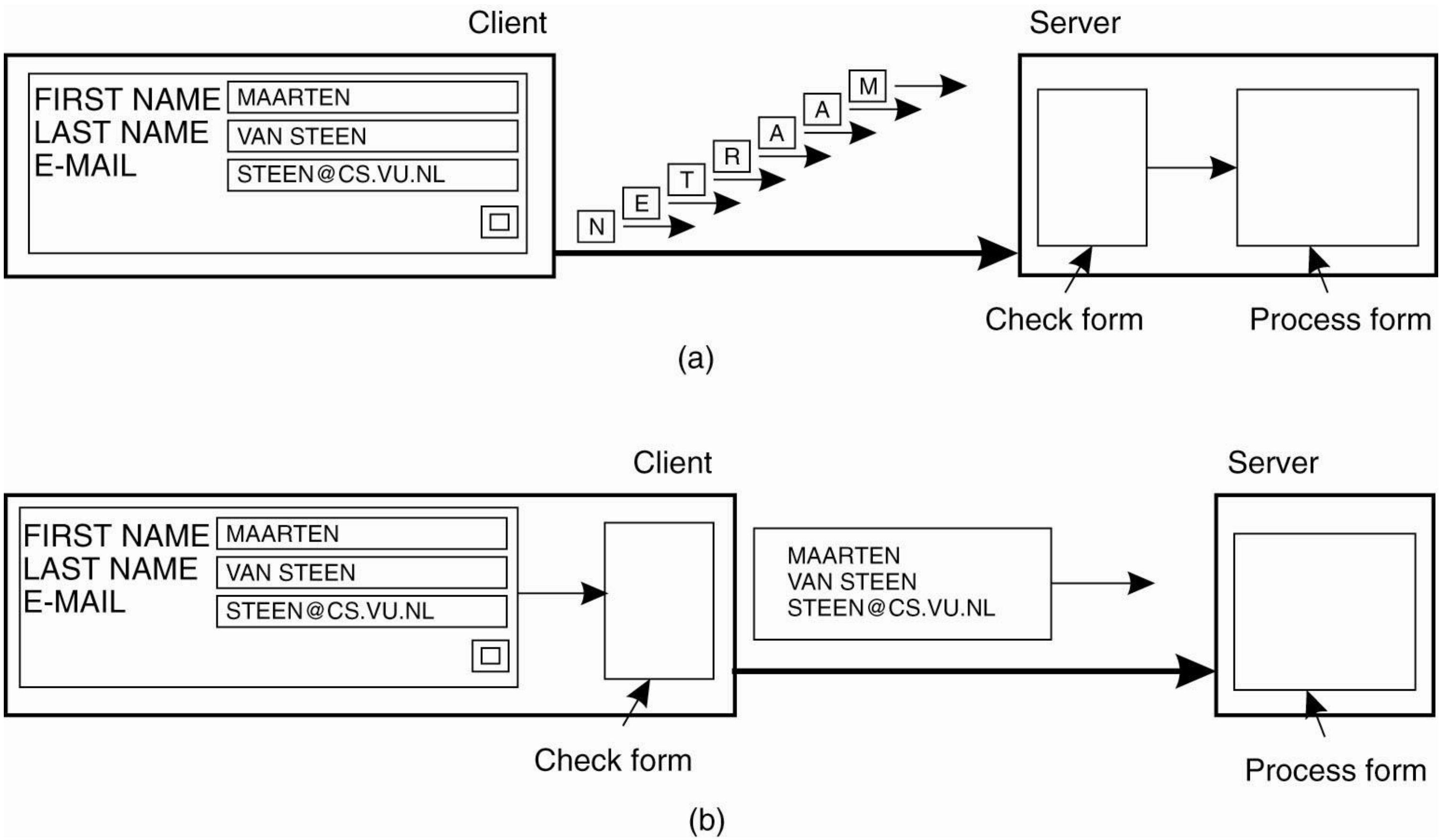

Figure 1-4. The difference between letting (a) a server or (b) a client check forms as they are being filled.

Scaling Techniques (2)

- **Distribution**: Divide a component in smaller parts and distribute them.
  - This can solve problems of limited processing and storage capacity.

Scaling Techniques (2)

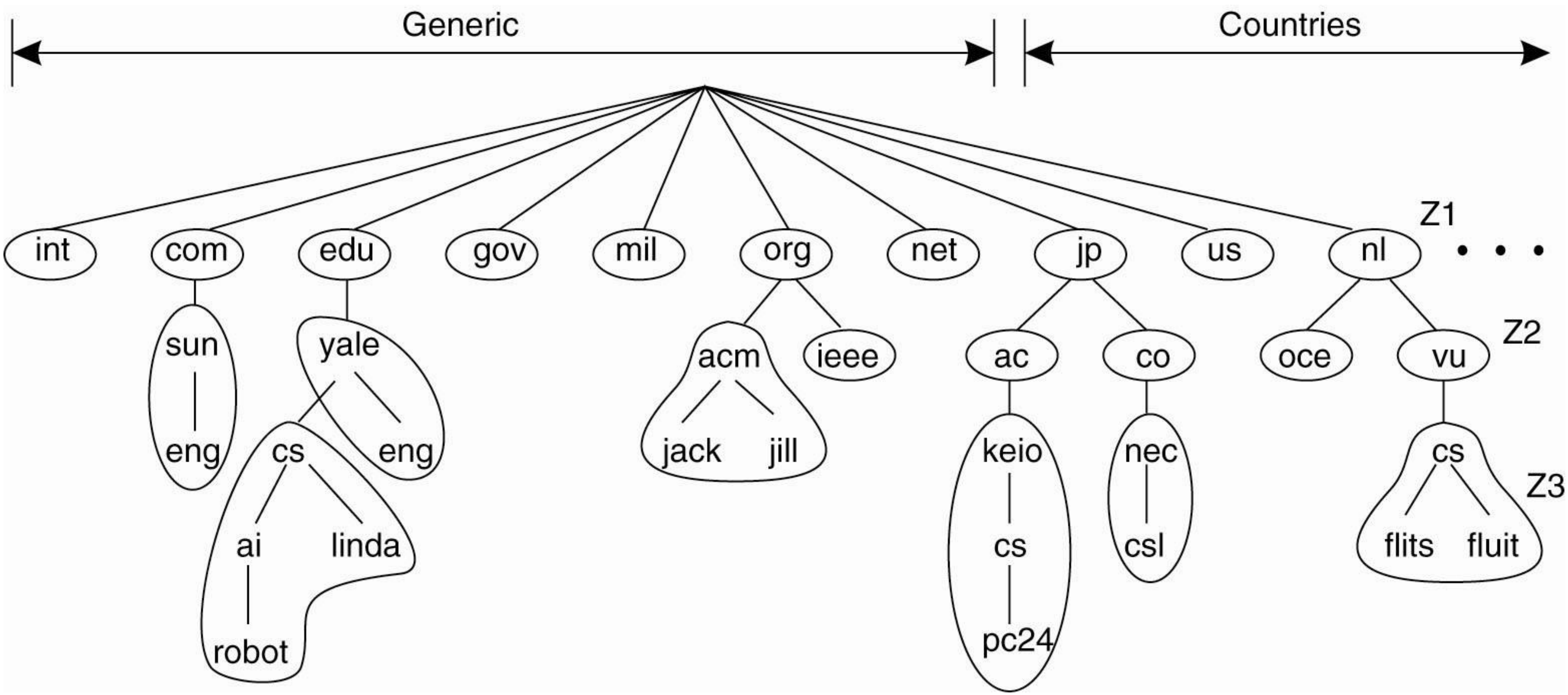

Figure 1-5. An example of dividing the DNS name space into zones.

Scaling Techniques (3)

- **Replication**: distribute multiple instances of the same component.
  - Help to increase the availability and to balance the system load.
  - For systems distributed over a large geographic area, having a copy of service/data closer helps to hide latency.
  - *Cache* is a special form of replication.
  - Consistency is an issue in replication of certain data/components, and keep all replicas synchronized can be a hard problem with large number of replicas connected through WANs.

Pitfalls when Developing Distributed Systems

False assumptions made by first time developer:

- 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

Some types of DS are:

- Distributed computing systems.
  - e.g.: Clusters and Grids
- Distributed information systems.
  - e.g., Transaction processing systems (TPS), Enterprise application integration (EAI).
- Distributed pervasive systems.
  - e.g., Household systems, Health Care Systems, Sensor networks.

Cluster Computing Systems

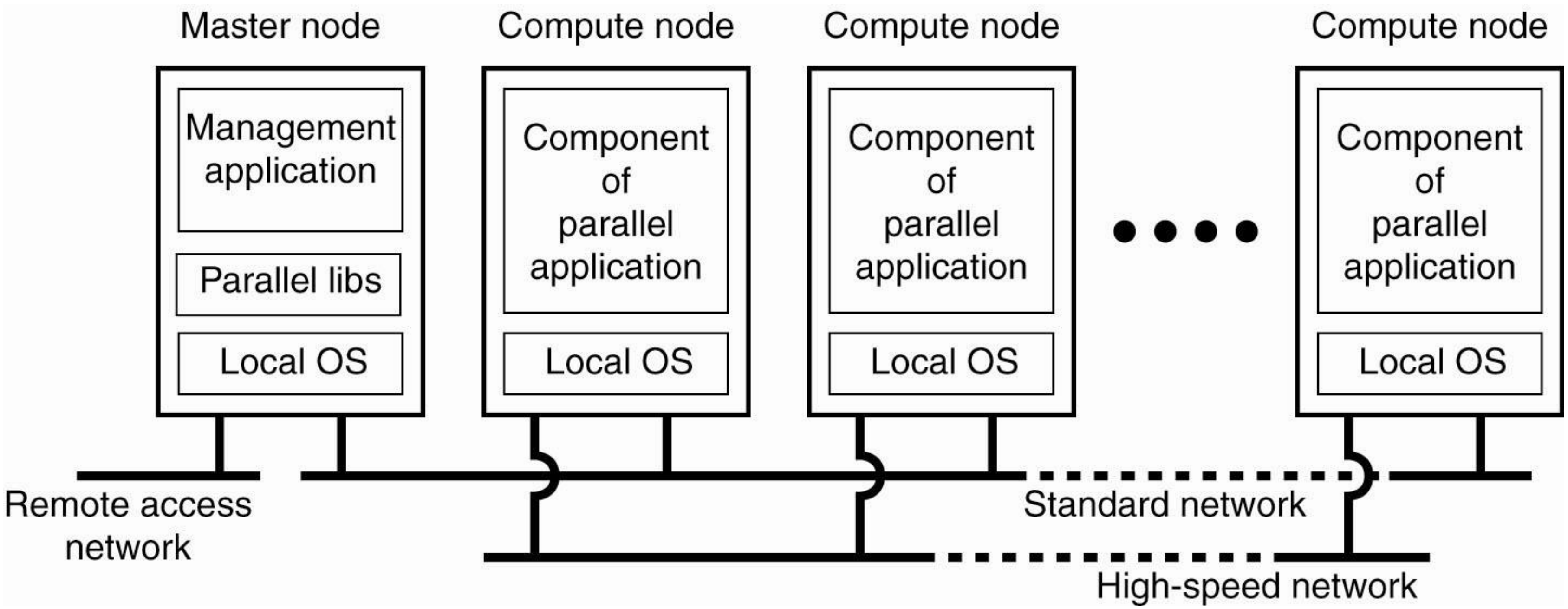

Figure 1-6. An example of a cluster computing system.

Grid Computing Systems

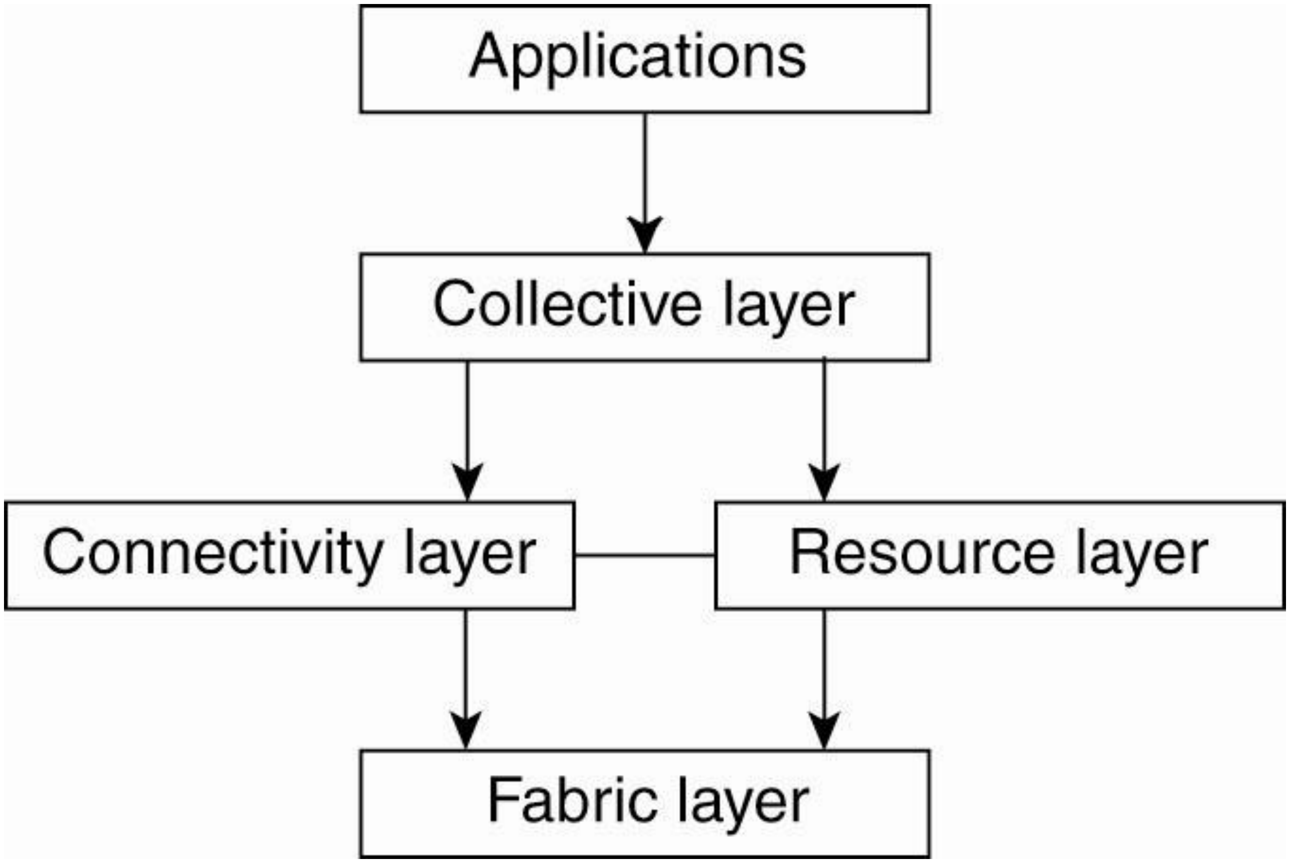

Figure 1-7. A layered architecture for grid computing systems.

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

Figure 1-8. Example primitives for transactions.

Transaction Processing Systems (2)

Characteristic properties of transactions:

- **A**tomic: To the outside world, the transaction happens indivisibly.
- **C**onsistent: The transaction does not violate system invariants.
- **I**solated: Concurrent transactions do not interfere with each other.
- **D**urable: Once a transaction commits, the changes are permanent.

Transaction Processing Systems (3)

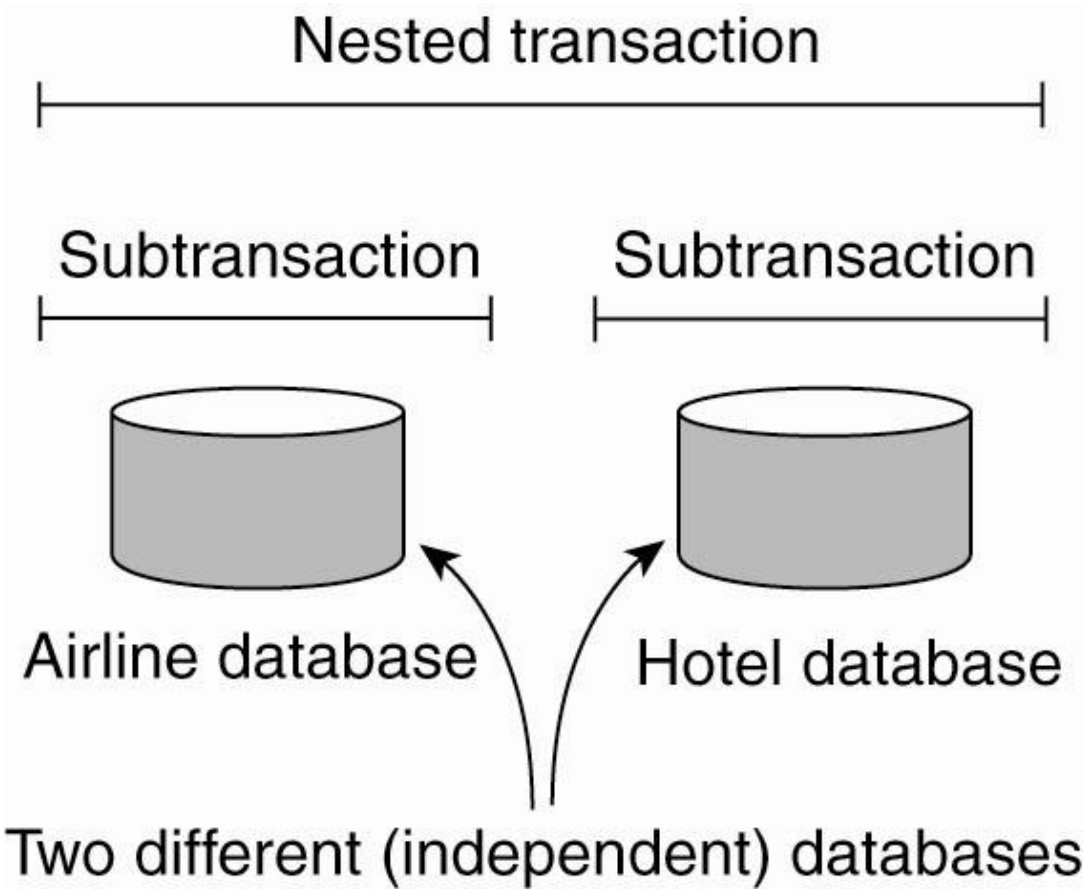

Figure 1-9. A nested transaction.

Transaction Processing Systems (4)

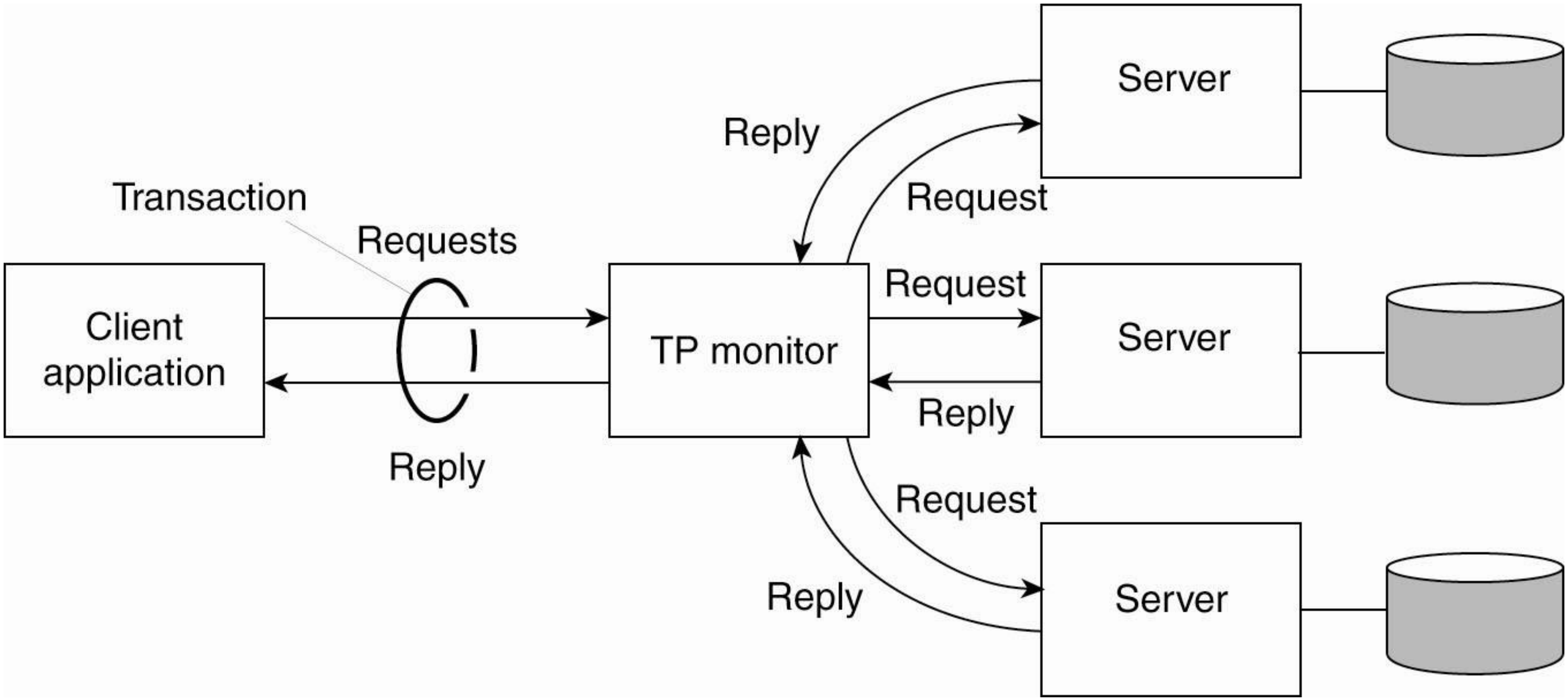

Figure 1-10. The role of a TP monitor in distributed systems.

Enterprise Application Integration

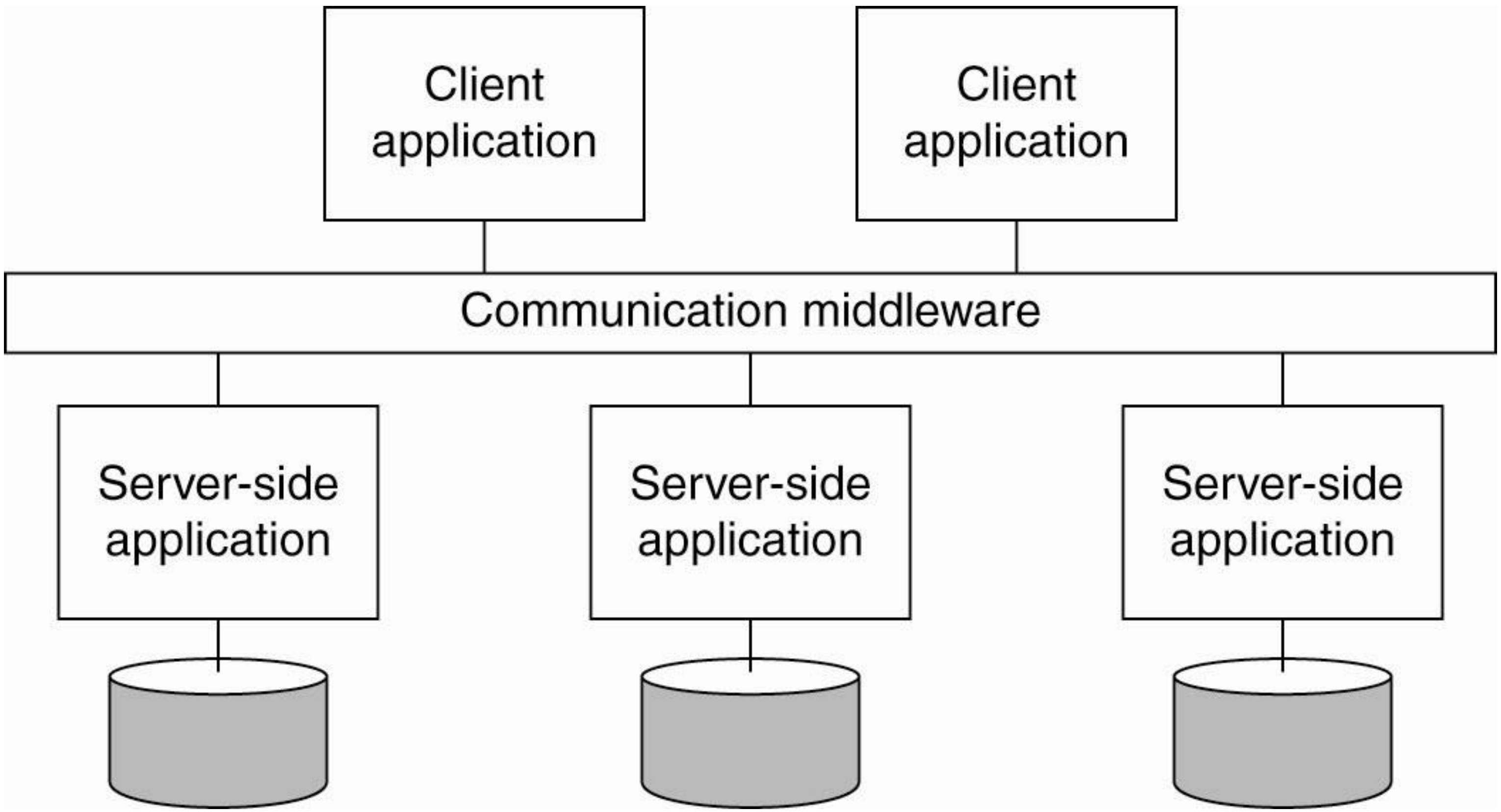

Figure 1-11. Middleware as a communication facilitator in enterprise application integration.

Distributed Pervasive Systems

Requirements for pervasive systems

- Embrace contextual changes.
- Encourage ad hoc composition.
- Recognize sharing as the default.

Electronic Health Care Systems (1)

Questions to be addressed for health care systems:

- 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)

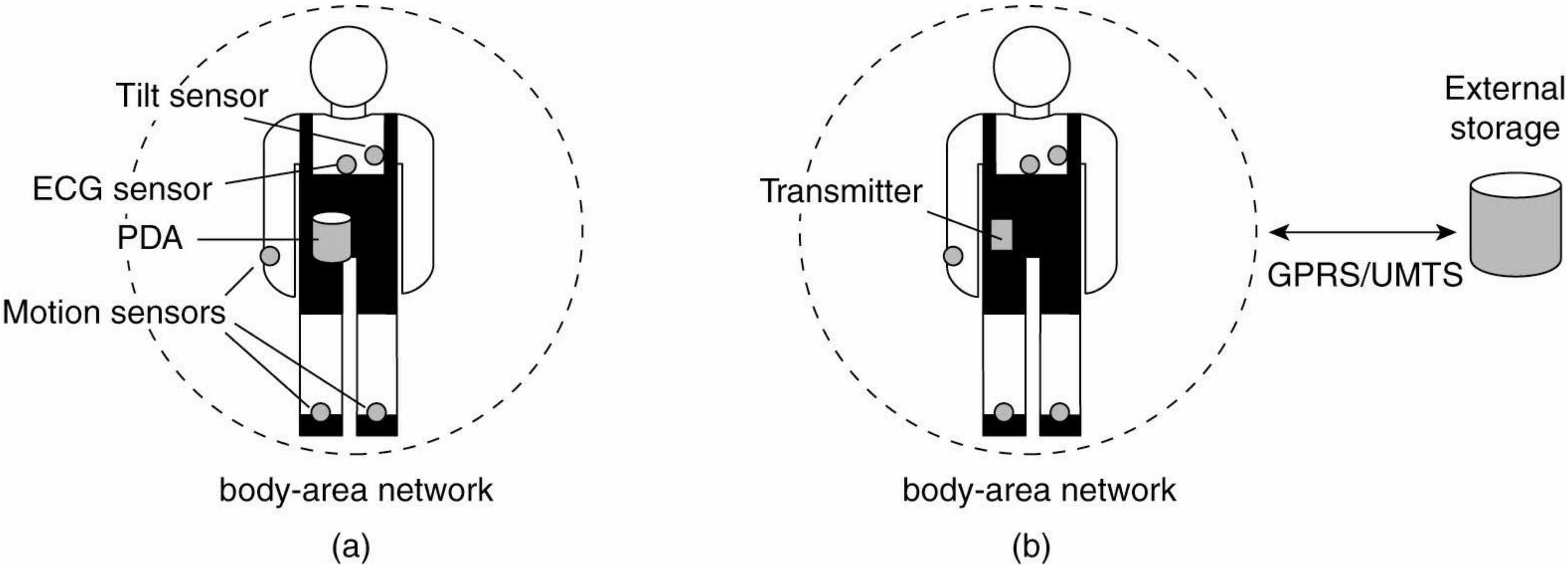

Figure 1-12. Monitoring a person in a pervasive electronic health care system, using (a) a local hub or
(b) a continuous wireless connection.

Sensor Networks (1)

Questions concerning sensor networks:

- 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)

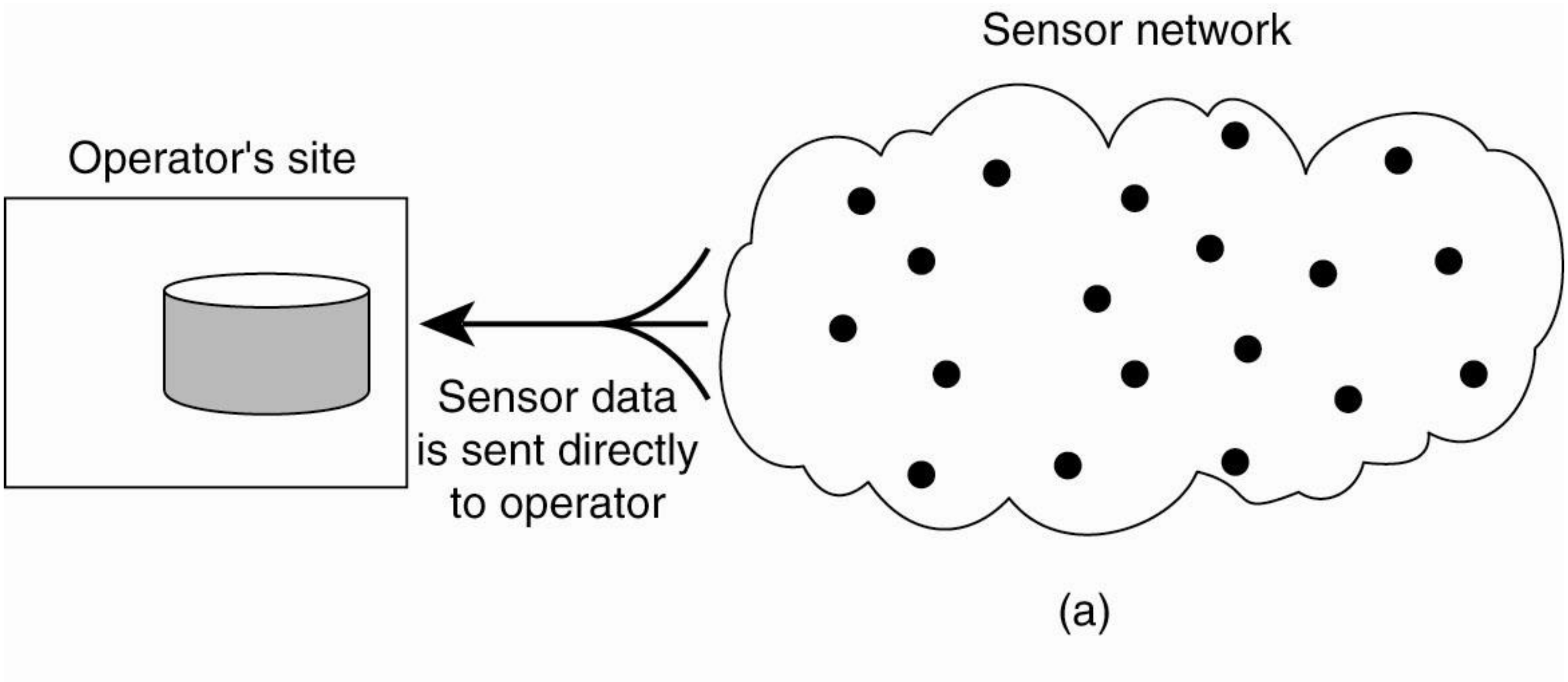

Figure 1-13. Organizing a sensor network database, while storing and processing data (a) only at the operator's site or …

Sensor Networks (3)

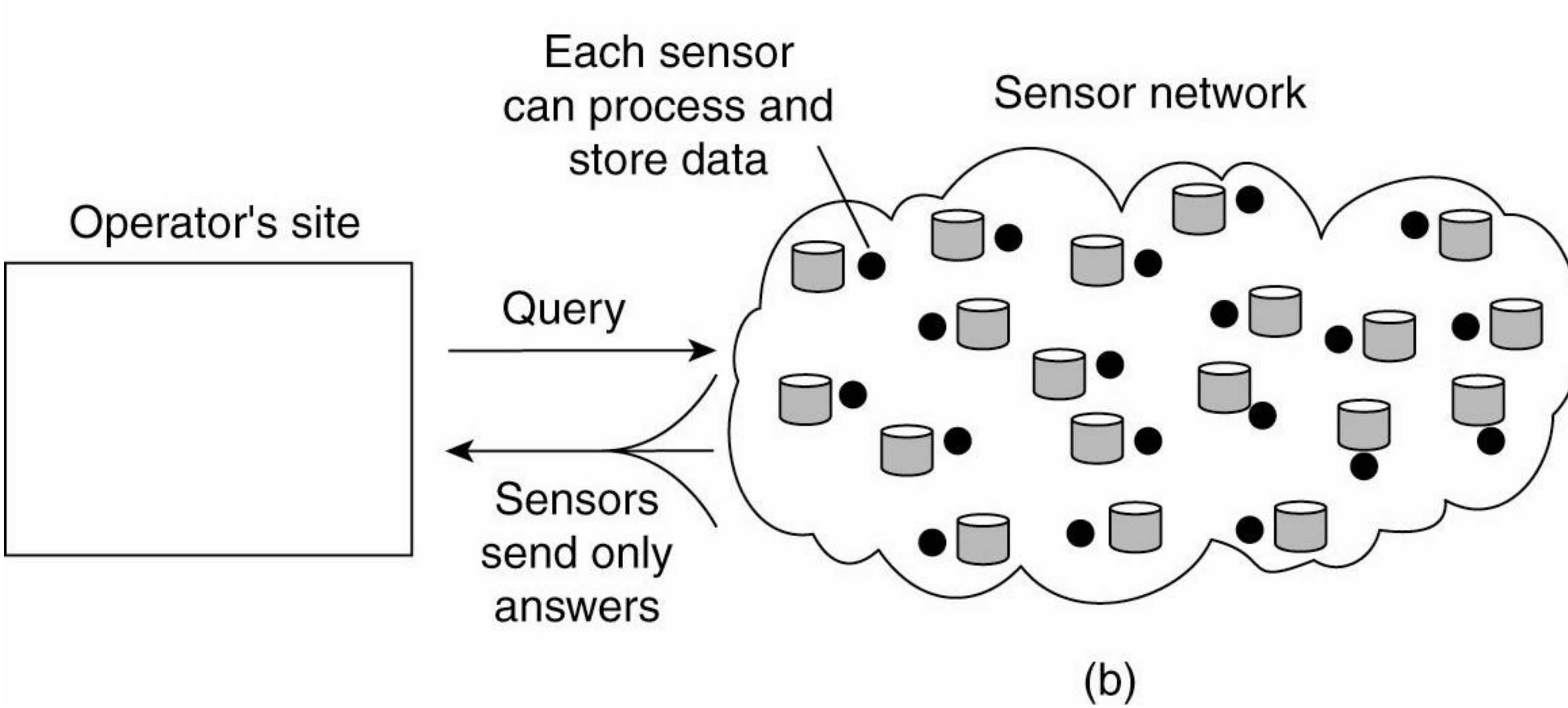

Figure 1-13. Organizing a sensor network database, while storing and processing data … or (b) only at the sensors.