Distributed Systems

Tanenbaum

Chapter 1

Outline

- Definition of a Distributed System
- Goals of a Distributed System
- Types of Distributed Systems

What Is A Distributed System?

- A collection of independent computers that appears to its users as a single coherent system.
- Features:
  - No shared memory – message-based communication
  - Each runs its own local OS
  - Heterogeneity
- Ideal: to present a single-system image:
  - The distributed system “looks like” a single computer rather than a collection of separate computers.

Distributed System Characteristics

- To present a single-system image:
  - Hide internal organization, communication details
  - Provide uniform interface
- Easily expandable
  - Adding new computers is hidden from users
- Continuous availability
  - Failures in one component can be covered by other components
- Supported by middleware

Definition of a Distributed System

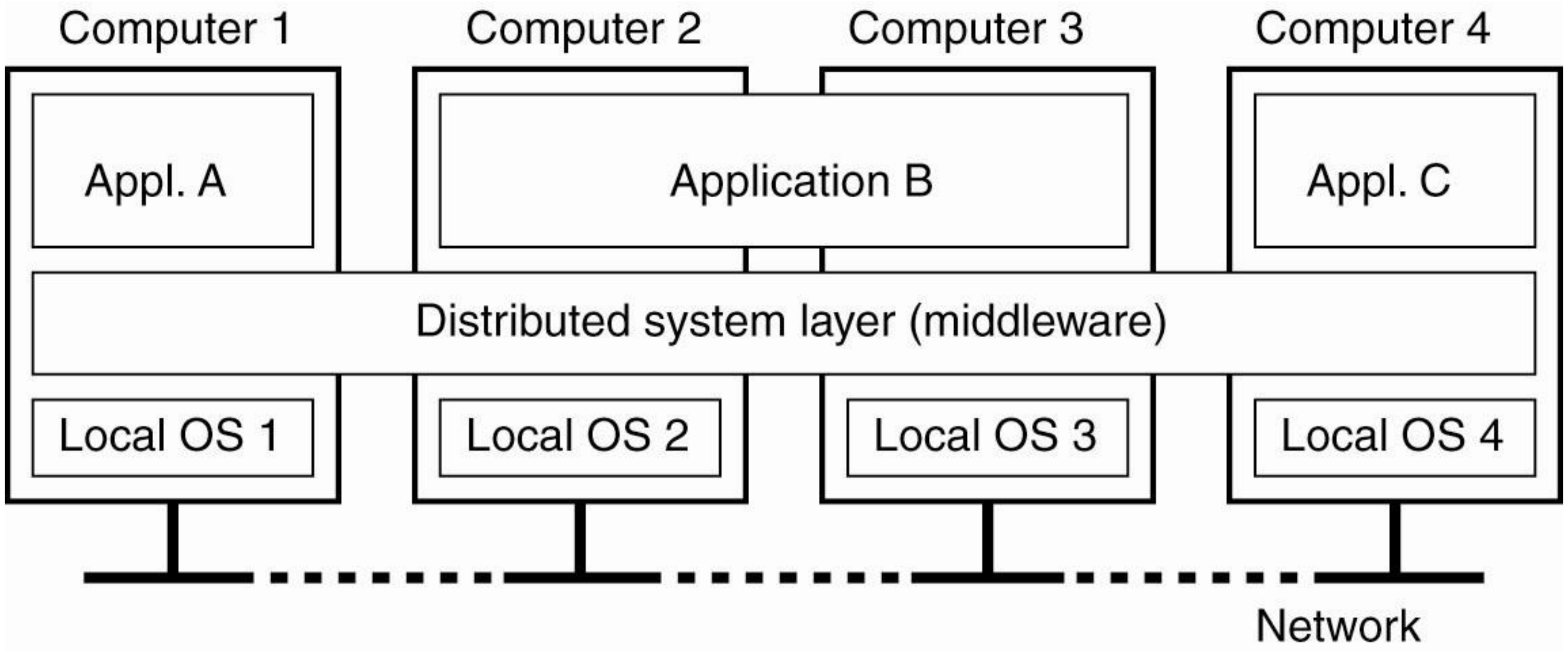

Figure 1-1. A distributed system organized as middleware. The middleware layer runs on all machines, and offers a uniform interface to the system

Role of Middleware (MW)

- In some early research systems: MW tried to provide the illusion that a collection of separate machines was a single computer.
  - Clustering software allows independent computers to work together closely

Role of Middleware (MW)

- MWs support communication across a network:
  - They provide protocols that allow a program running on one kind of computer, using one kind of operating system, to call a program running on another computer with a different operating system
  - The communicating programs must be running the *same* middleware.

Distributed System Goals

- Resource Accessibility
- Distribution Transparency
- Openness
- Scalability

Goal 1 – Resource Availability

- Support user access to remote resources (printers, data files, web pages, CPU cycles) and the fair sharing of the resources
- Sharing expensive resources
- Performance enhancement
- Access to remote services
- Resource sharing introduces security problems.

Goal 2 – Distribution Transparency

- Software hides some of the details of the distribution of system resources.
  - Makes the system more user friendly.
- A distributed system that appears to its users & applications to be a single computer system is said to be *transparent*.
  - Users & apps should be able to access remote resources in the same way they access local resources.
- Transparency has several dimensions.

Types of Transparency

Transparency	Description
Access	Hide differences in data representation & resource access
Location	Hide location of resource (can use resource without knowing its location)
Migration	Hide possibility that a system may change location of resource
Replication	Hide the possibility that multiple copies of the resource exist
Concurrency	Hide the possibility that the resource may be shared concurrently
Failure	Hide failure and recovery of the resource. How does one differentiate betw. slow and failed?
Relocation	Hide that resource may be moved during use

Figure 1-2. Different forms of transparency in a distributed system

Trade-off: transparency vs other factors

- Reduced performance: multiple attempts to contact a remote server can slow down the system
- Convenience: direct the print request to my local printer, not one on the next floor
- Too much emphasis on transparency may prevent the user from understanding system behavior.

Goal 3 - Openness

- An **Open Distributed System**:
  - offers services according to standard rules that describe the syntax and semantics of those services.
  - In other words, the interfaces to the system are clearly specified and freely available.
- **Interface Definition/Description Languages (IDL):** used to describe the interfaces between software components
  - Definitions are language & machine independent
  - Support communication between systems using different OS/programming languages; e.g. a C++ program running on Windows communicates with a Java program running on UNIX

Open Systems Support

- **Interoperability**: the ability of two different systems or applications to work together
  - A process that needs a service should be able to talk to any process that provides the service.
  - Multiple implementations of the same service may be provided.

- **Portability**: an application designed to run on one distributed system can run on another system which implements the same interface.

- **Extensibility**: Easy to add new components, features

Goal 4 - Scalability

- Dimensions that may scale:
  - With respect to **size**
  - With respect to **geographical distribution**
  - With respect to **the number of administrative organizations spanned**
- A scalable system still performs well as it scales up along any of the three dimensions.

Size Scalability

- Scalability is negatively affected when the system is based on
  - Centralized server: one for all users
  - Centralized data: a single data base for all users
  - Centralized algorithms: one site collects all information, processes it, distributes the results to all sites.

Decentralized Algorithms

- No machine has complete information about the system state
- Machines make decisions based only on local information
- Failure of a single machine doesn't ruin the algorithm
- There is no assumption that a global clock exists.

Geographic Scalability

- Early distributed systems ran on LANs, relied on **synchronous communication.**
  - May be too slow for wide-area networks
  - Wide-area communication is unreliable, point-to-point;
  - Unpredictable time delays may even affect correctness
- LAN communication is based on broadcast.
  - Consider how this affects an attempt to locate a particular kind of service
- Centralized components + wide-area communication: waste of network bandwidth

Scalability - Administrative

- Different domains may have different policies about resource usage, management, security, etc.

- Trust often stops at administrative boundaries
  - Requires protection from malicious attacks

Scaling Techniques

- Scalability affects performance more than anything else.

- Three techniques to improve scalability:
  - Hiding communication latencies
  - Distribution
  - Replication

Hiding Communication Delays

- use **asynchronous communication**
- While waiting for one answer, do something else
  - *e.g.*, create one thread to wait for the reply and let other threads continue to process or schedule another task

Hiding Communication Delays

- Download part of the computation to the requesting platform to speed up processing
  - Filling in forms to access a DB: send a separate message for each field, or download form/code and submit finished version.

Scaling Techniques

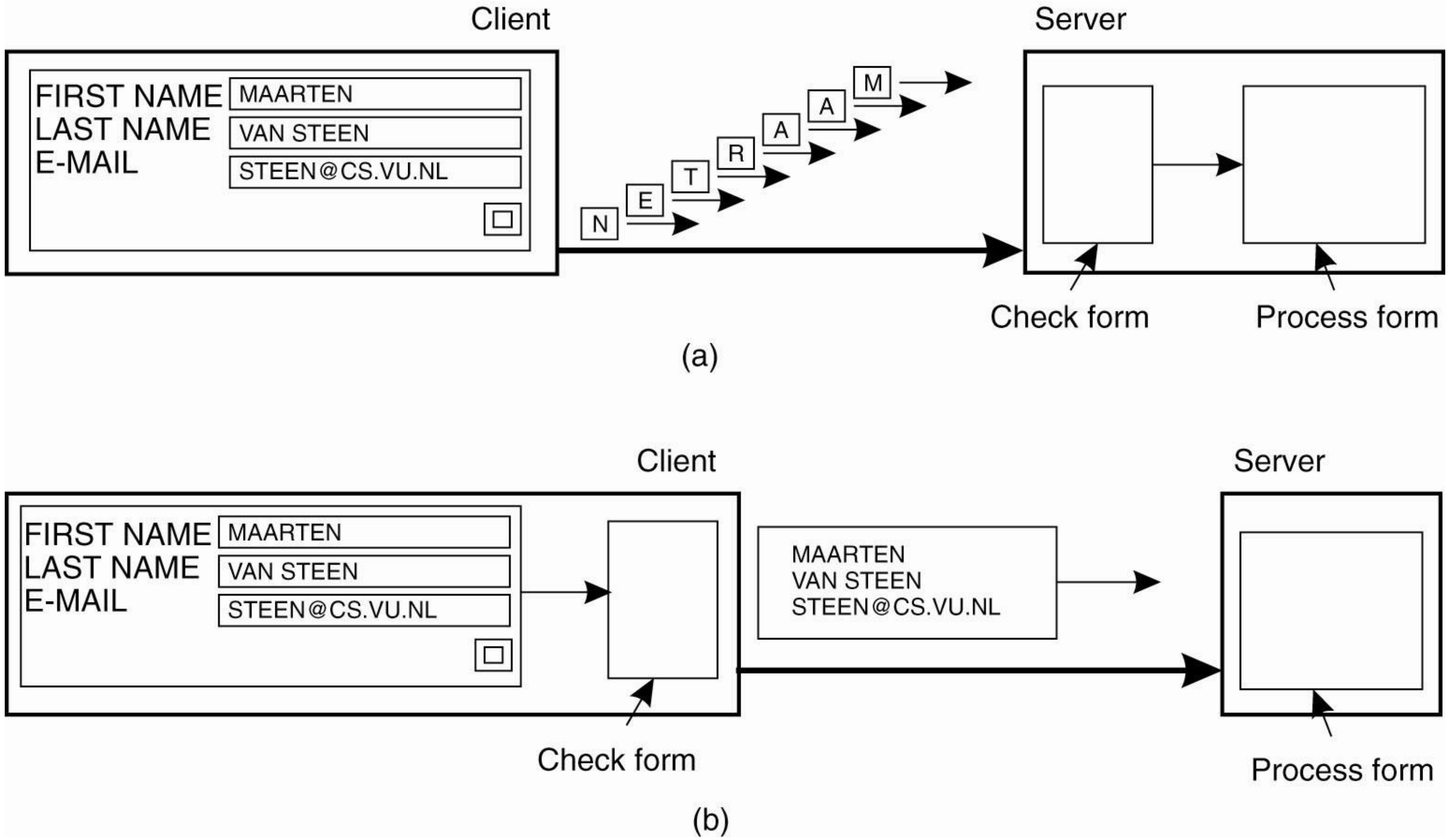

Figure 1-4. The difference between letting (a) a server or (b) a client check forms as they are being filled.

Distribution

- Instead of one centralized service, divide into parts and distribute geographically
- Each part handles one aspect of the job
  - Example: DNS namespace is organized as a tree of domains; each domain is divided into zones; names in each zone are handled by a different name server
  - WWW consists of many (millions?) of servers

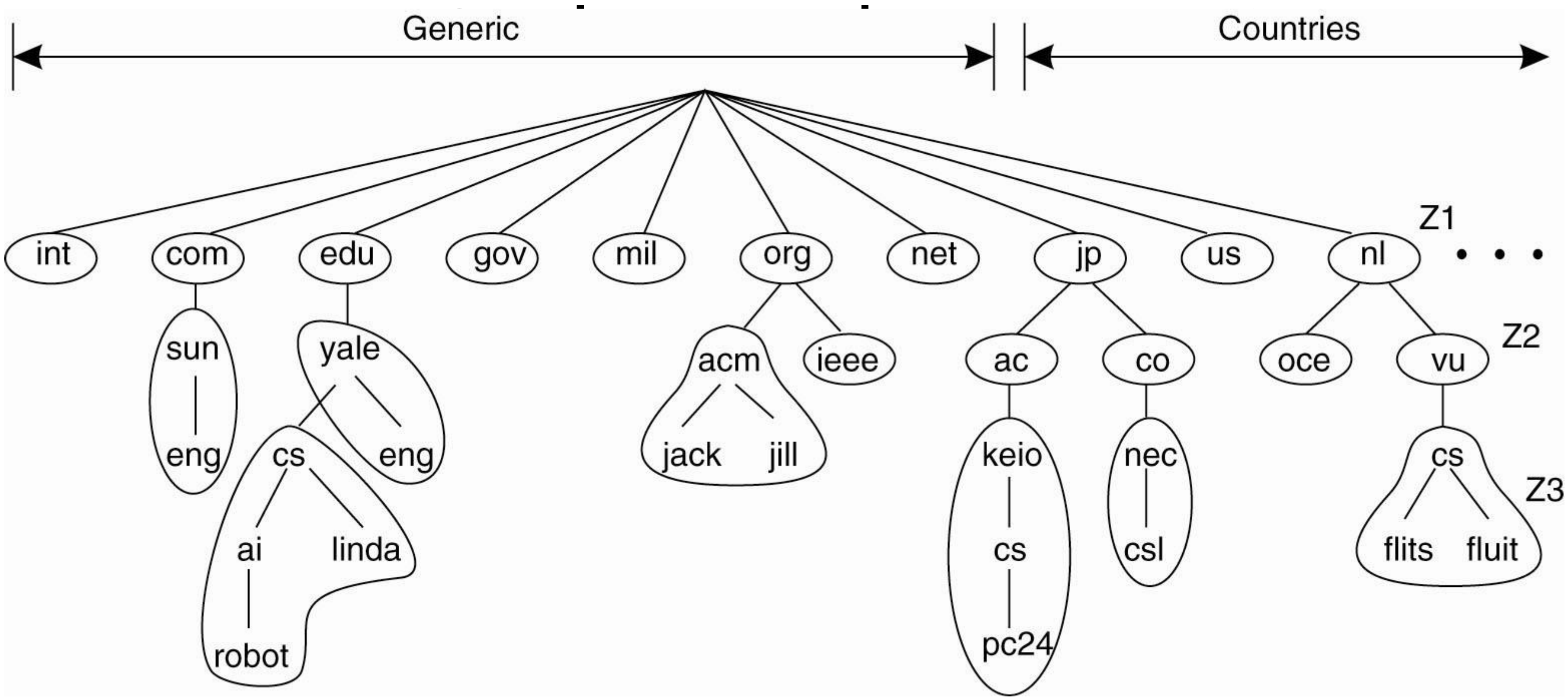

Figure 1-5. An example of dividing the DNS name space into zones.

Third Scaling Technique - Replication

- Replication: multiple identical copies of something
  - Replicated objects may also be distributed, but aren't necessarily.
- Replication
  - Increases availability
  - Improves performance through load balancing
  - May avoid latency by improving proximity of resource

Caching

- Caching is a form of replication
  - Normally creates a (temporary) replica of something closer to the user
- User (client system) decides to cache, server system decides to replicate
- Both lead to **consistency** problems

Summary
Goals for Distribution

- Resource accessibility
  - For sharing and enhanced performance
- Distribution transparency
  - For easier use
- Openness
  - To support interoperability, portability, extensibility
- Scalability
  - With respect to size (number of users), geographic distribution, administrative domains

Issues/Pitfalls of Distribution

- Requirement for advanced software to realize the potential benefits.
- Security and privacy concerns regarding network communication
- Replication of data and services provides fault tolerance and availability, but at a cost.
- Network reliability, security, heterogeneity, topology
- Latency and bandwidth
- Administrative domains

Distributed Systems

- Early distributed systems emphasized the single system image – often tried to make a networked set of computers look like an ordinary general purpose computer

Types of Distributed Systems

- Distributed Computing Systems
  - Clusters
  - Grids
- Distributed Information Systems
  - Transaction Processing Systems
  - Enterprise Application Integration
- Distributed Embedded Systems
  - Home systems
  - Health care systems
  - Sensor networks

Cluster Computing

- A collection of **Similar Processors** (PCs, workstations) running the **same** OS, connected by a **high-speed** LAN.

- **Parallel Computing** capabilities using inexpensive PC hardware

Cluster Types & Uses

- High Performance Clusters (HPC)
  - run large parallel programs
  - Scientific, military, engineering apps; e.g., weather modeling
- Load Balancing Clusters
  - Front end processor distributes incoming requests
  - server farms (e.g., at banks or popular web site)
- High Availability Clusters (HA)
  - Provide redundancy – back up systems
  - May be more fault tolerant than large mainframes

Clusters – Beowulf model

- Linux-based
- Master-slave paradigm - one processor is the master
  - allocates tasks to other processors
  - maintains batch queue of submitted jobs
  - handles interface to users
- Master has libraries to handle message-based communication or other features (the middleware).

Cluster Computing Systems

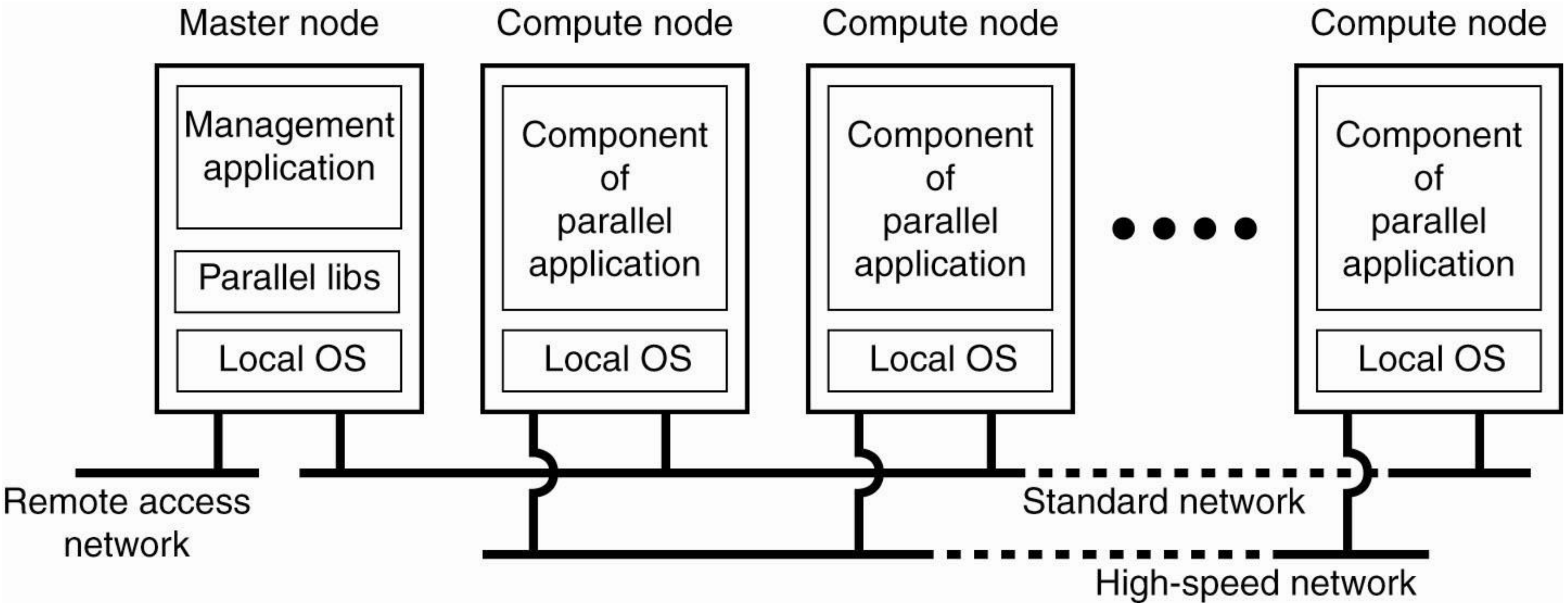

Figure 1-6. An example of a (Beowulf) cluster computing system

Grid Computing Systems

- Highly heterogeneous with respect to hardware, software, networks, security policies, etc.

- Grids support **virtual organizations**: a collaboration of users who pool resources (servers, storage, databases) and share them

Grids

- Similar to clusters but processors are more loosely coupled, tend to be heterogeneous, and are not all in a central location.
- Problems are broken up into parts and distributed across multiple computers in the grid – less communication between parts than in clusters.
- Grid software is concerned with managing sharing across administrative domains.

A Proposed Architecture for Grid Systems

- **Fabric layer**: interfaces to local resources at a specific site
- **Connectivity layer**: protocols to support usage of *multiple resources* for a single application; e.g., access a remote resource or transfer data between resources; and protocols to provide security
- **Resource layer** manages a *single resource,* using functions supplied by the connectivity layer
- **Collective layer:** resource discovery, allocation, scheduling, etc.
- **Applications**: use the grid resources
- The collective, connectivity and resource layers together form the middleware layer for a grid

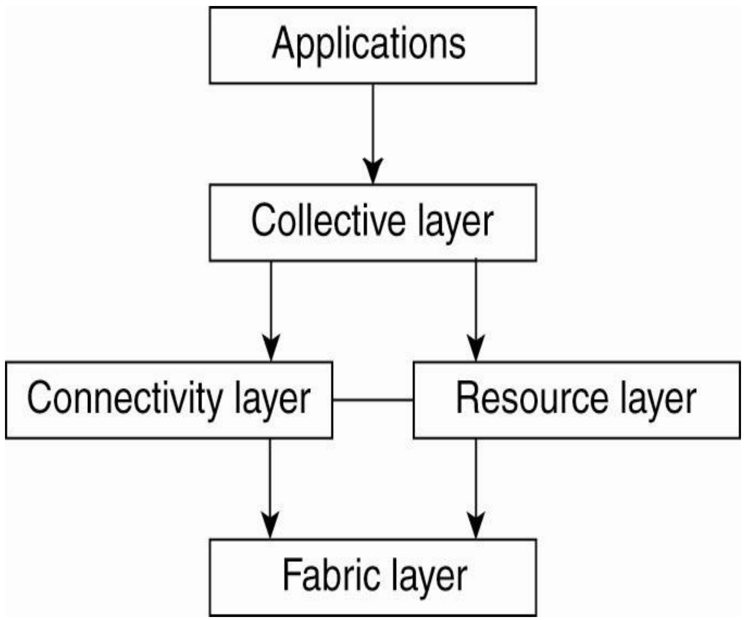

Figure 1-7. A layered architecture for grid computing systems

OGSA – Another Grid Architecture

- Open Grid Services Architecture (OGSA) is a service-oriented architecture
  - Sites that offer resources to share do so by offering specific Web services.

- The architecture of the OGSA model is more complex than the previous layered model.

Globus Toolkit

- An example of grid middleware
- Supports the combination of heterogeneous platforms into virtual organizations.
- Implements the OSGA standards, among others.

Types of Distributed Systems

- Distributed Computing Systems
  - Clusters
  - Grids
- Distributed Information Systems
- Distributed Embedded Systems

Distributed Information Systems

- Business-oriented

- Systems to make a number of separate network applications interoperable and build “enterprise-wide information systems”.

- Two types discussed here:
  - Transaction processing systems
  - Enterprise Application Integration (EAI)

Transaction Processing Systems

- Provide a highly structured client-server approach for database applications
- **Transactions** are the communication model
- Obey the ACID properties:
  - **Atomic**: all or nothing
  - **Consistent**: invariants are preserved
  - **Isolated**
  - **Durable**: committed operations can't be undone

Transaction Processing Systems

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

Nested Transactions

- A nested transaction is a transaction **within** another transaction (a sub-transaction)
  - Example: a transaction may ask for two things (e.g., airline reservation info + hotel info) which would spawn two nested transactions
- Primary transaction waits for the results.
  - While children are active parent may only abort, commit, or spawn other children

Transaction Processing Systems

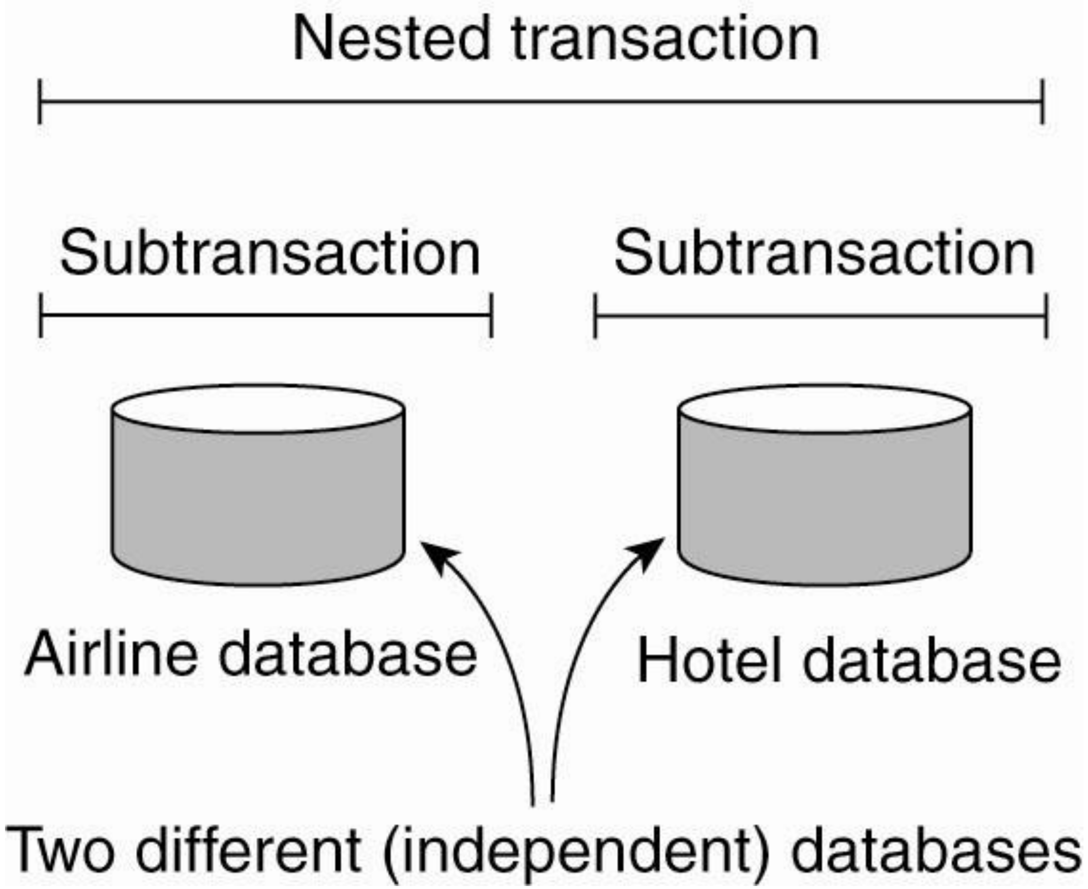

Figure 1-9. A nested transaction.

Implementing Transactions

- Conceptually, private copy of all data
- Multiple sub-transactions – commit, abort
  - Durability is a characteristic of top-level transactions only
- Nested transactions are suitable for distributed systems
  - Transaction processing monitor may interface between client and multiple data bases.

Enterprise Application Integration

- Less structured than transaction-based systems
- EA components communicate directly
  - May use different OSs, different DBs but need to interoperate sometimes.
- Communication mechanisms to support this include Remote Procedure Call (RPC) and Remote Method Invocation (RMI)

Enterprise Application Integration

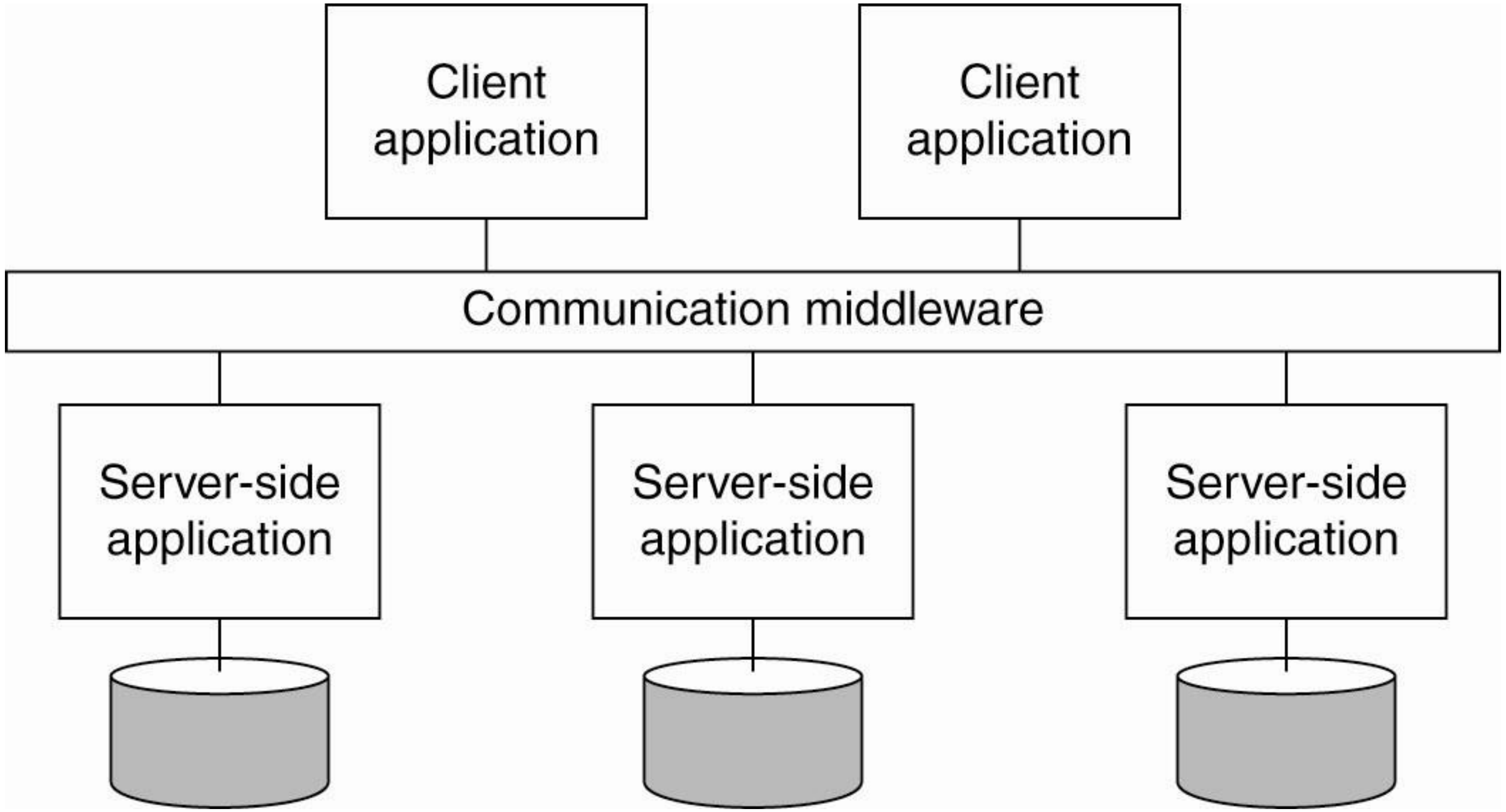

Figure 1-11. Middleware as a communication facilitator in enterprise application integration.

Distributed Pervasive Systems

- The first two types of systems are characterized by their stability: nodes and network connections are more or less fixed
- This type of system is likely to incorporate **small**, **battery-powered**, **mobile devices**
  - Home systems
  - Electronic health care systems – patient monitoring
  - Sensor networks – data collection, surveillance

Home System

- Built around one or more PCs, but can also include other electronic devices:
  - Automatic control of lighting, alarm systems, etc.
  - Network enabled appliances
  - PDAs and smart phones, etc.

Electronic Health Care Systems

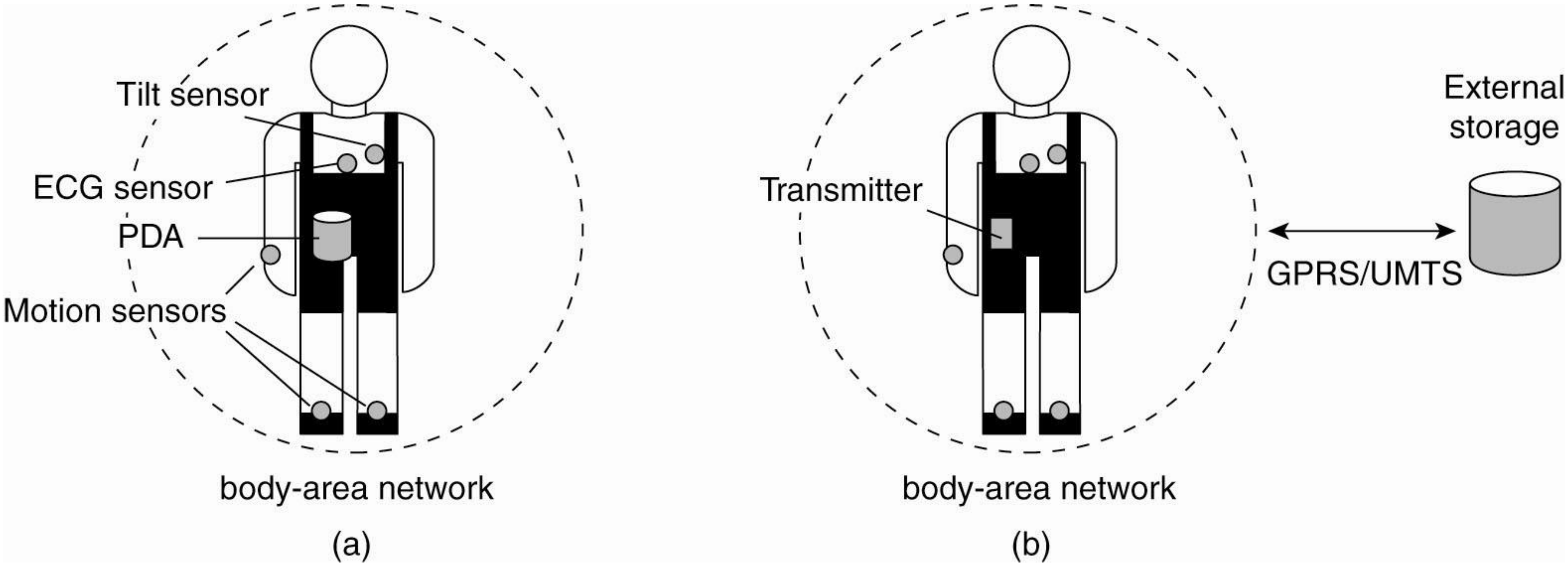

Figure 1-12. Monitoring a person in a pervasive electronic health care system, using (a) a local hub or (b) a continuous wireless connection.

Sensor Networks

- A collection of geographically distributed nodes consisting of a comm. device, a power source, some kind of sensor, a small processor…
- Purpose: to collectively monitor sensory data (temperature, sound, moisture etc.,) and transmit the data to a base station
- “smart environment” – the nodes may do some rudimentary processing of the data in addition to their communication responsibilities.

Sensor Networks

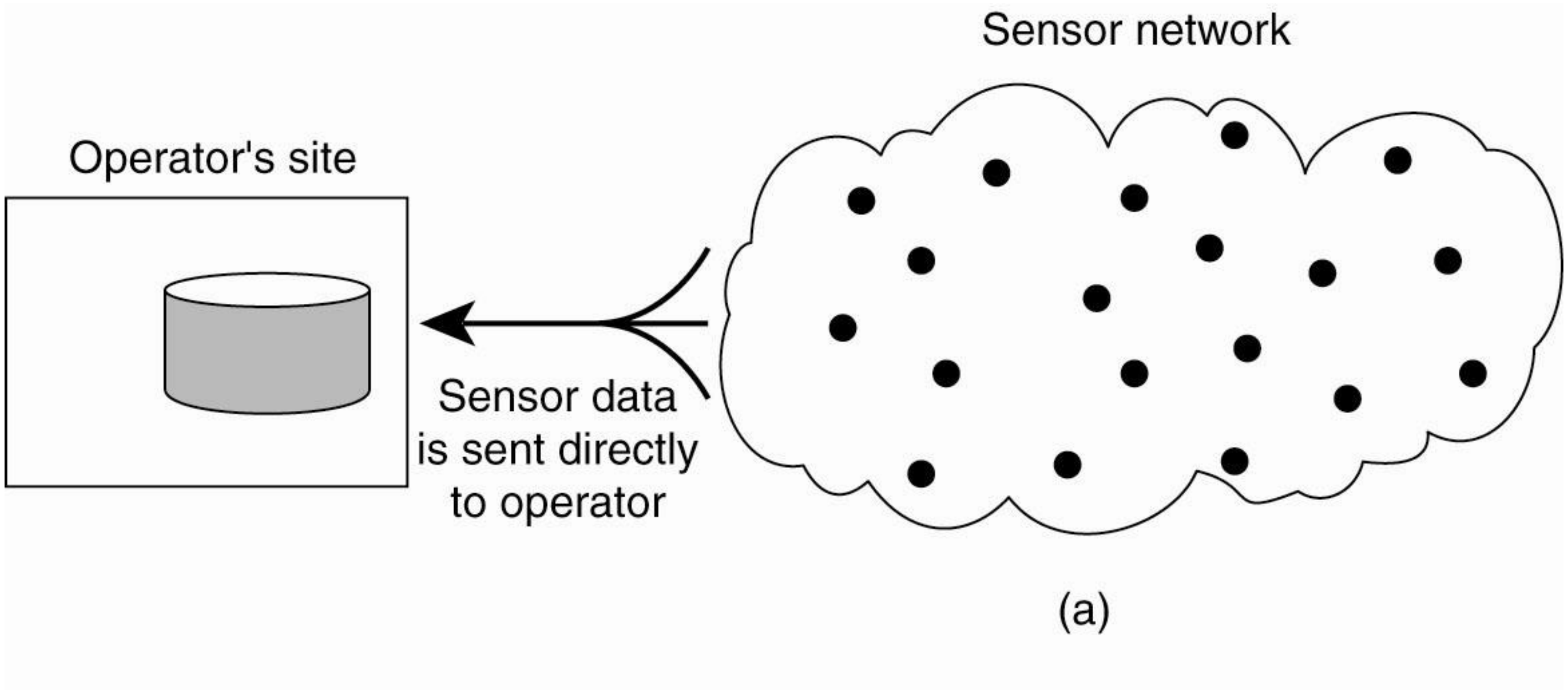

Figure 1-13. Organizing a sensor network database, while storing and processing data (a) only at the operator's site or …

Sensor Networks

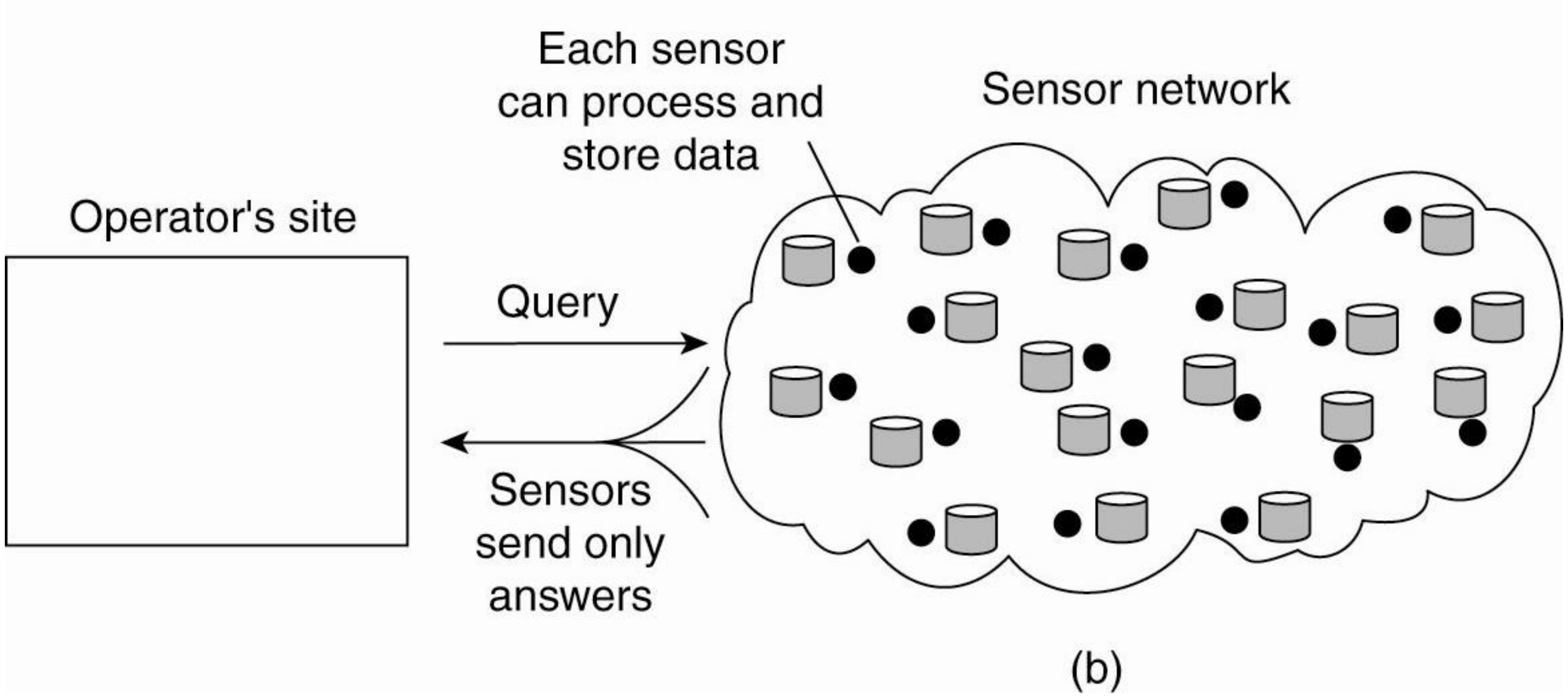

Figure 1-13. Organizing a sensor network database, while storing and processing data … or (b) only at the sensors.

Summary – Types of Systems

- Distributed computing systems – our main emphasis
- Distributed information systems – we will talk about some aspects of them
- Distributed pervasive systems – not so much

Questions?