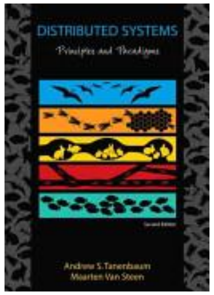
Chapter 1: Introduction

Grand tour of the Distributed Systems



Thanks to the authors of the textbook [**TS**] for providing the base slides. I made several changes/additions.
These slides may incorporate materials kindly provided by Prof. Dakai Zhu.
So I would like to thank him, too.

Turgay Korkmaz
korkmaz@cs.utsa.edu

Chapter 1: Introduction

- DEFINITION OF A DISTRIBUTED SYSTEM
- GOALS
  - Making Resources Accessible
  - Distribution Transparency
  - Openness
  - Scalability
  - Pitfalls
- TYPES OF DISTRIBUTED SYSTEMS
  - Distributed Computing Systems
  - Distributed Information Systems
  - Distributed Pervasive Systems

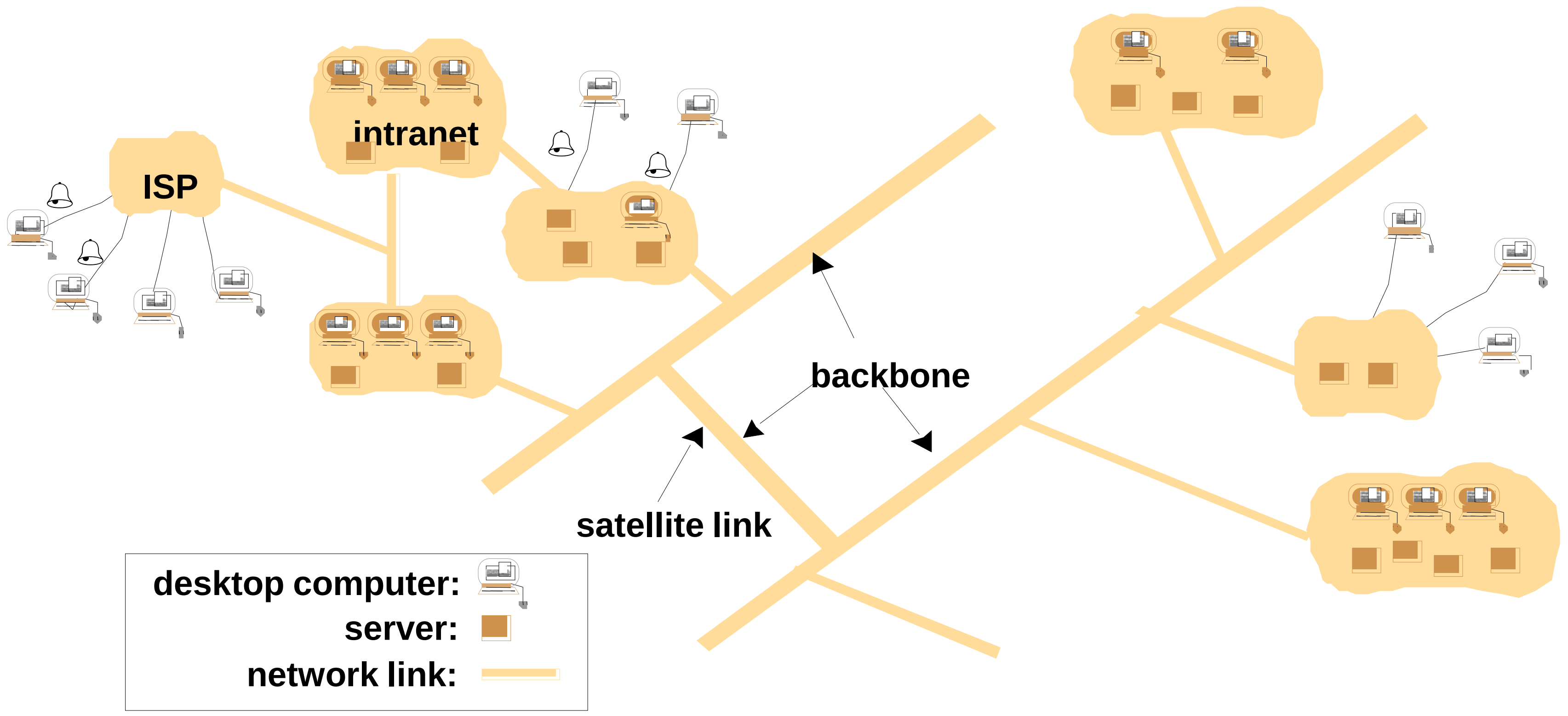
Objectives

- To provide a **grand tour** of the key issues in distributed systems

Computer System Revolution

- Computers
  - large/expensive → small/cheap
- Networks:
  - LAN → WAN,
  - bps → Kbps → Gbps
- Now, it is easy to put together many computers, and people to:
  - Solve problems
  - Share resources
  - Increase collaboration
- **Centralized sys→ Distributed sys** → Cloud

What are *Distributed* Systems?



A collection of

networked *independent* computers

that appears to its users as a

single coherent system

Distributed Systems: Definition

- A distributed system (DS) is a piece of software that ensures that

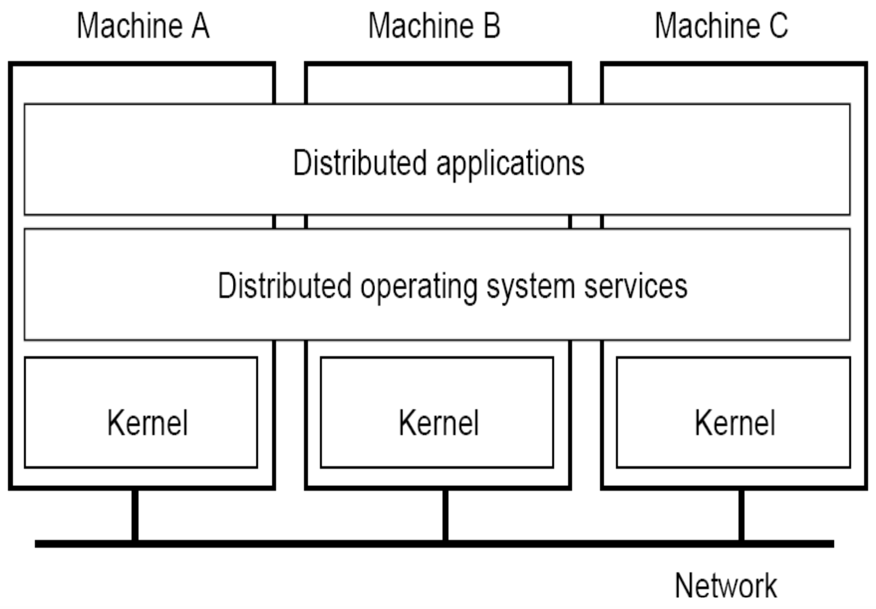
  a collection of independent computers

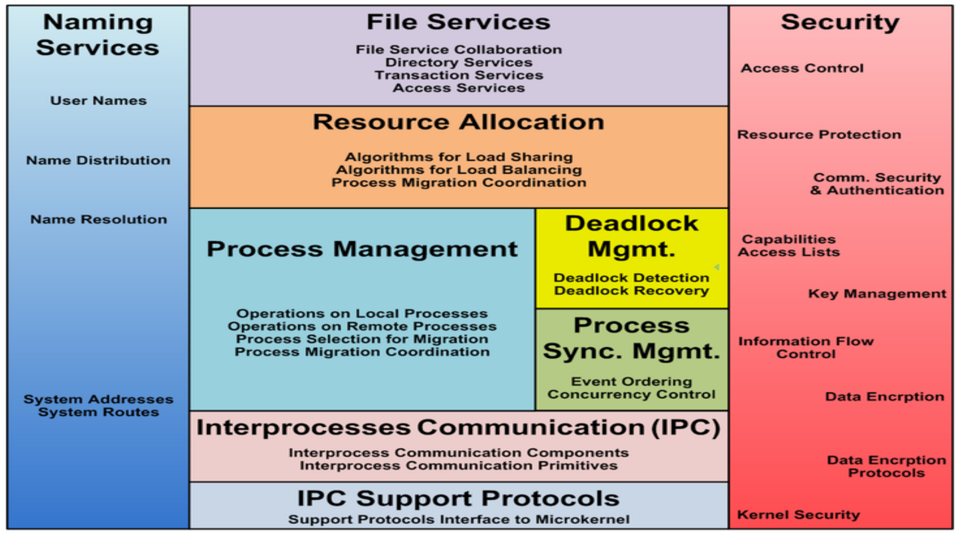
  appears to its users as a single coherent system

- But HOW can we
  - hide the differences between independent computers &
  - provide a single system view?
- Solutions for distributed systems
  - Distributed OS
  - Network OS
  - **Middleware**

Distributed Operating Systems

- OS essentially tries to maintain a **single, global view of the resources** it manages (Tightly-coupled OS).
- ***Full* transparency**: users feel a big system and are not aware of multiple different machines
- Access to *remote* services similar to local resources

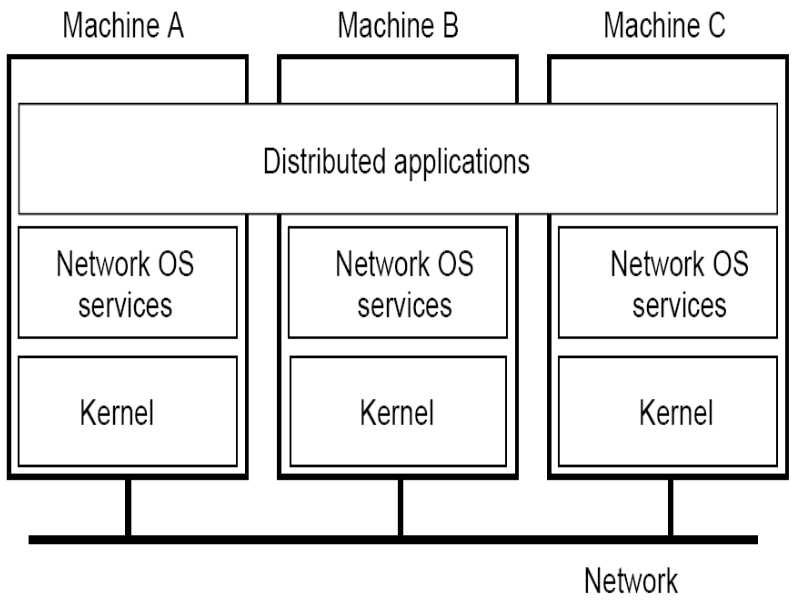




[From Wikipedia]

Network Operating Systems

- Collection of independent OS augmented by **network services** (Loosely-coupled OS)
- **No transparency**: users are aware of the multiplicity of the machines
  - explicitly log on into remote machines, or copy files from other machines
- Apps use ***network services*** to access resources



- No single view of the distributed system

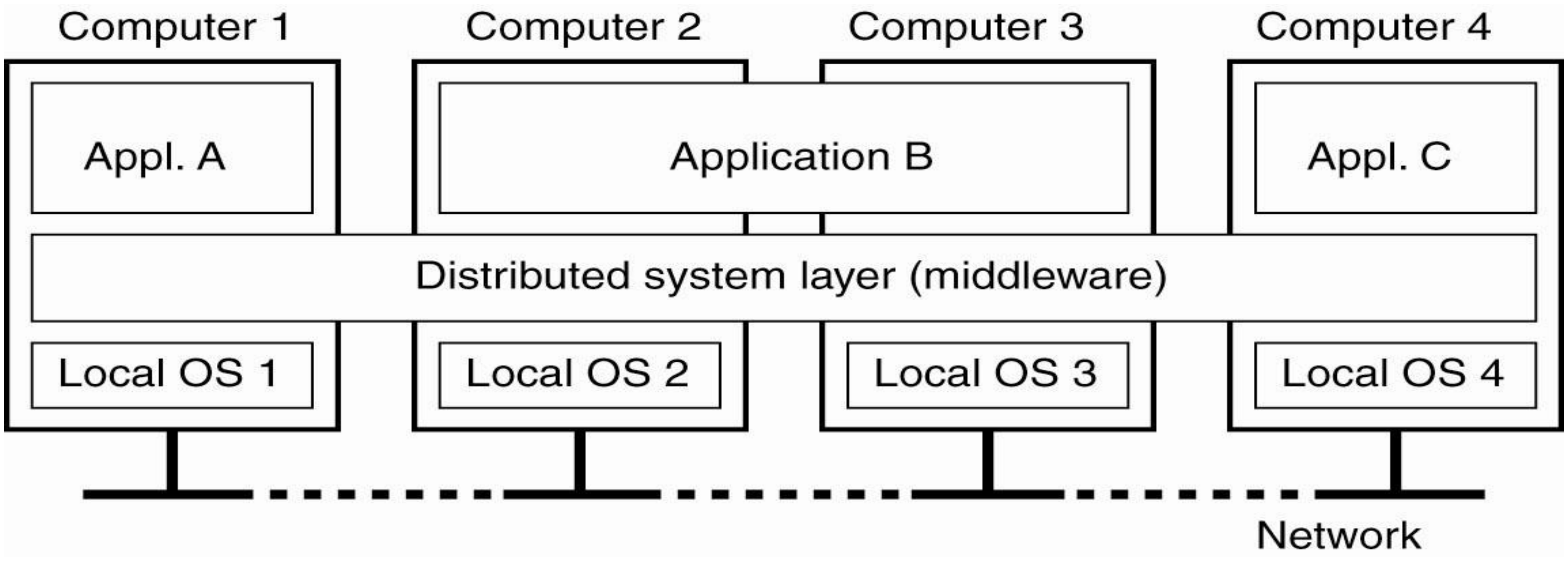
- Need multiple passwords, multiple access permissions.

- Only means of communication is message passing

+ Adding or removing a machine is relatively simple.

Middleware-Based Distributed Systems

- Most modern distributed systems are designed to provide a **level of transparency** through a software layer on top of local OSes
- This software layer is called **Middleware**



Middleware-Based DS (cont.)

- ***Middleware***
  - A higher level of programming abstraction
    - Examples: RPC, RMI
  - It hides the differences between various computers and the ways in which they communicate
  - It provides a single-system view
  - …..
- As a result, ***middleware*** facilitates the **integration** and **interaction** of various networked applications in a **consistent** and **uniform** manner.

Make resources available/accessible

Distribution transparency

Openness

Scalability

GOALS OF DISTRIBUTED SYSTEMS

Make resources available/accessible

A^7: Anytime Anywhere Affordable Access to Anything by Anyone Authorized **(**Jeannette M. Wing, 2008**)**

+ share resources (economics)

+ increase collaboration

- security and privacy

- unwanted traffic

Distribution Transparency

-- users and applications see the DS as a single coherent system --

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 that a resource may be moved to another location while in use
Replication	Hides the fact that an object or its state may be replicated at different locations
Concurrency	Hides coordination of activities between objects to achieve consistency at a higher level
Failure	Hides failure and possible recovery of Objects

Distribution transparency is a nice a goal, but achieving it is a different story.

Degree of Transparency

- Aiming at full distribution transparency is good, but too much of it might hurt (like food :)
- Full transparency will cost performance
  - Keeping Web caches exactly up-to-date with the master
  - Immediately flushing write operations to disk for fault tolerance
- Completely hiding failures of networks and nodes is (theoretically and practically) impossible
  - Can we distinguish a slow computer from a failing one?
  - Can we be sure that a server actually performed an operation before a crash?
- Moreover, some things cannot be hidden (e.g., propagation delay)
- Solution: Expose distribution of the system
  - Let user be aware of distribution at various levels of transparency
  - For example, would you prefer a busy printer in CS or the idle one in ECE?

Openness of Distributed Systems

- Offer services according to standard rules that describe the *syntax* and *semantics* of those services
- So that different open systems would be able to interact and use services from each other
- **How to achieve openness**
  - Conform to well-defined **interfaces** (often described using IDL)
    - Easy to define syntax. But semantic is hard so in practice it is defined in a natural language
  - Support **portability**
    - The same implementation (source code) should work on different machines
  - Easily **interoperate**
    - Two different implementations should work together irrespective of their environments
- Distributed system should be independent from **heterogeneity** of the underlying environment

  Hardware, Software Platforms, and Languages

Implementing Openness:
Separate Policies and Mechanisms

- What are they? Take web-caching as an example:
- Policies:
  - What level of consistency do we require
  - Which operations in a downloaded code do we allow
  - Which QoS requirements do we adjust
  - What level of secrecy do we require for communication?
- Mechanisms:
  - Allow (dynamic) setting of caching policies
  - Support different levels of trust for mobile code
  - Provide adjustable QoS parameters per data stream
  - Offer different encryption algorithms
- Separate them for flexibility and efficiency
- For this, design system as a collection of **small components** instead of a monolithic large prog

Scalability in Distributed Systems

- Many developers of modern distributed system easily use the adjective "scalable" without making clear **why their system actually scales.**
- Three aspects of scalability

● **Size**	Number of users and/or processes
● **Geographical**	Maximum distance between nodes
● **Administrative**	Number of administrative domains

- Most systems account for **size** scalability: powerful servers (supercomputer)
- Challenges: geographical and administrative scalability

Problems with Size Scalability

- What happens when more users/resources added?
- Limitations of centralized systems
  - Service (e.g., single server) overloaded servers
  - Data (e.g., single phone book) saturated communication links
  - Algorithm (e.g., routing based on global info) too much traffic
- Use distributed service, database, and algorithm
  - No machine has complete info
  - Make decision based on local info
  - Failure of one node does affect others (not always)
  - No global clock (it can be done on LANs but trick in WANs)

Problems with Geographical Scalability

- Suppose we have an interactive application working on a LAN, can we use it over a WAN?
- Delay
  - Blocking read/write might be OK on LAN but not on WAN
- Reliability
  - Longer the distance higher the chance of loosing messages
- Bandwidth
  - Locating a service by broadcasting is OK on LAN (e.g., ARP) but not on WAN

Problems with Administrative Scalability

- In a single domain:
  - We can try to optimize resource usage because each entity belongs to the same domain and can be trusted
- In case of multiple and independent administrative domains:
  - We do not own all resources and cannot trust others
  - So, we try to get things done based on some policies and agreements rather than optimization (e.g., BGP vs. OSPF)
  - But there are several problems
    - Conflicting policies (who uses what and pays how much)
    - Management
    - Security (access rights and trust management)

Techniques for Scalability
to solve performance problems

- Use **asynchronous** communication
  - Separate handler for incoming response and do something while waiting
  - + hide communication latencies
  - - what if there is nothing else to do
- Partition data and computations into smaller parts and **distribute** them across multiple machines
  - Decentralized naming services (DNS)
  - Decentralized data, information systems (WWW)
  - Decentralized algorithm (Distance Vector)
- Move computations to clients (Java applets)
- Minimize packet format and protocol overheads
- Use forward error coding instead of re-transmission

Techniques for Scalability (cont'd)

to solve performance problems

- Use **Replication/caching** that makes multiple copies of the same services or data available at different machines
  - Mirrored Web sites
  - Replicated file servers and databases
  - Web caches (in browsers and proxies)
  - File caching (at server and client)

  - + increase availability
  - + improve load balance and performance
  - + hide communication latency

  - **- Inconsistencies** *when one copy is modified*
  - **- Global synchronization** is needed for keeping copies consistent but it precludes large-scale solutions
  - - Tolerance to inconsistencies depends on application

Techniques for Scalability (cont'd)

- All the techniques discussed so far deal with performance problems due to size and geographical scalability
- How about administrative scalability?
  - The most challenging one (why?)
  - The problems are often non-technical (Politics!)

Developing Distributed Systems: Pitfalls

- A complex task: Sound SW Eng Principles help
- But a lot of mistakes are made because
  - the dispersion of many components are not taken in to account during design,
- Mistakes are often due to **false assumptions**:
  - The same **global time**
  - **Perfect** network/communication
    - Latency is zero
    - Bandwidth is infinite
    - The network is reliable
    - The network is secure
    - The network is homogeneous
  - The **topology** does not change
  - There is one **administrator**

Distributed **Computing** Systems (DCS)

Cluster computing, Grid computing, Cloud computing

Distributed **Information** Systems (DIS)

Web servers, Distributed database applications

Distributed **Pervasive** Systems (DPS)

Smart home systems, Electronic health systems, Sensor networks: surveillance systems

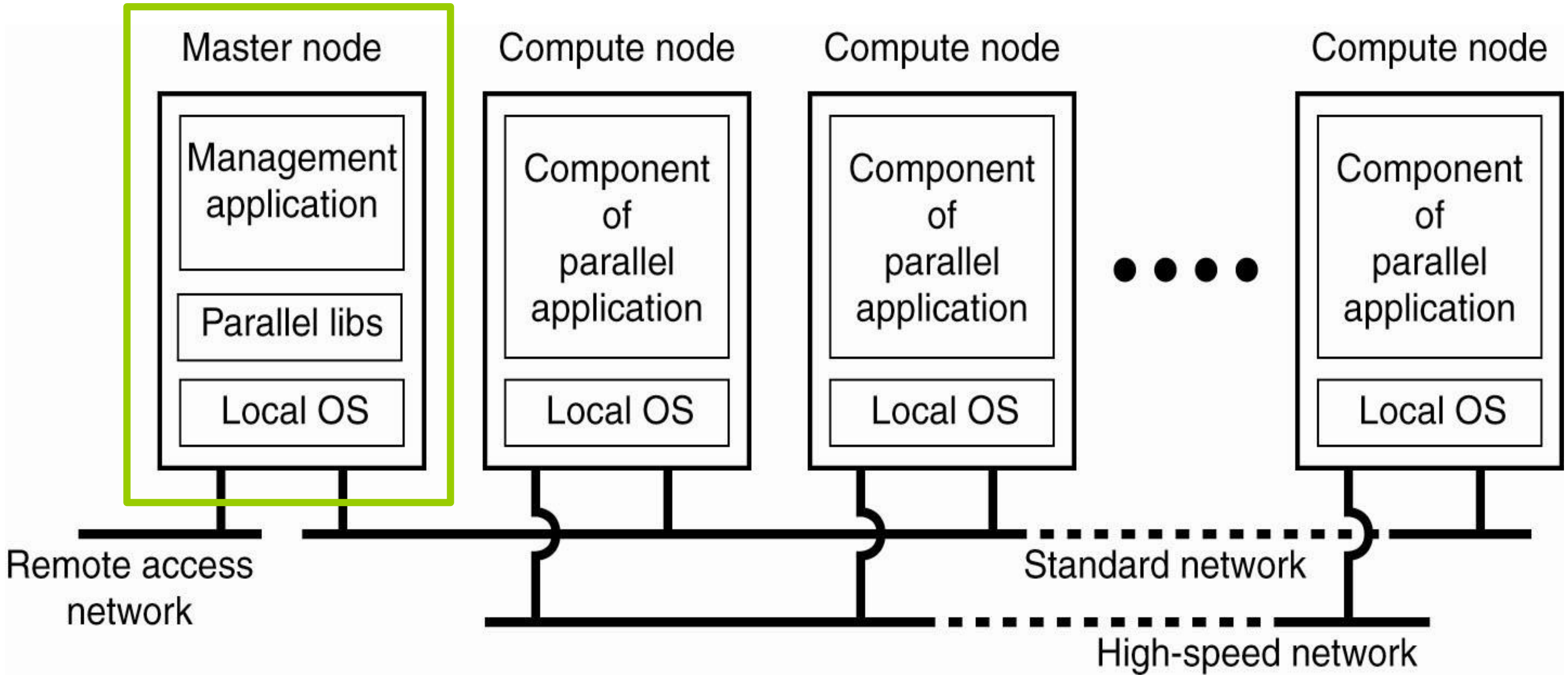
DIFFERENT TYPES OF DISTRIBUTED SYSTEMS

DCS: Distributed Computing Systems

Cluster Computing Systems

A group of high-end systems connected through a LAN

- **Homogeneous**: same OS, near-identical hardware
- **Single managing node**



DCS: Distributed Computing Systems

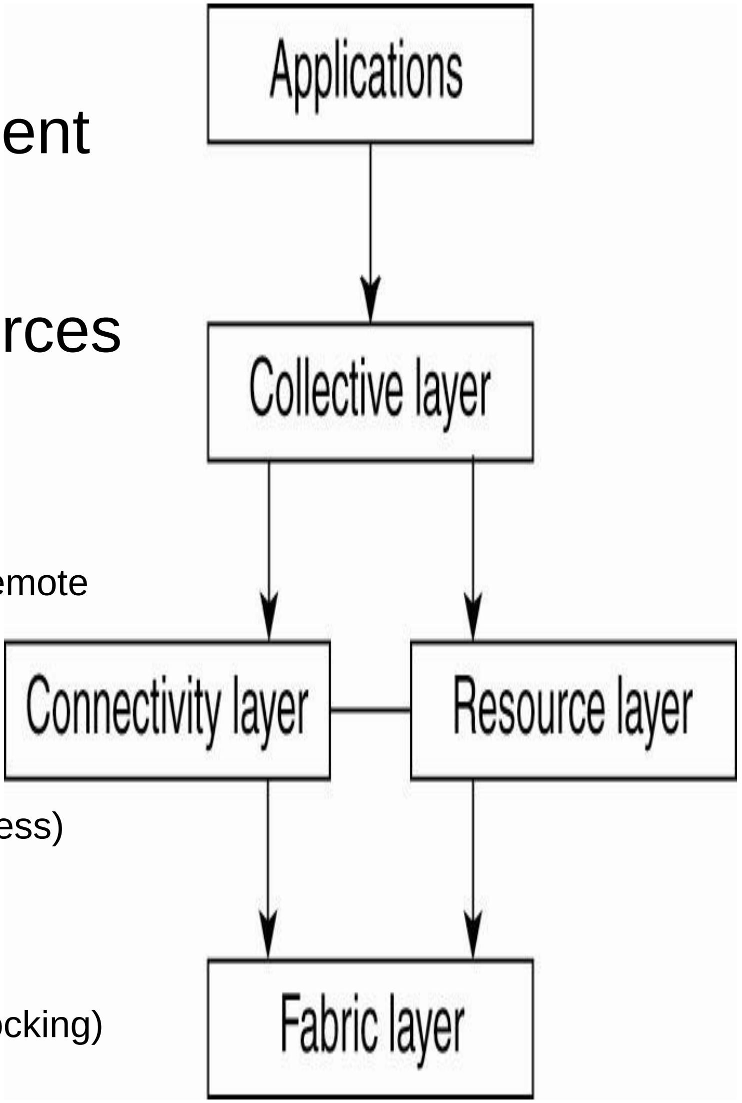
Grid Computing Systems

- Lots of nodes from everywhere share resources and collaborate
  - **Heterogeneous**
  - Dispersed across several organizations
  - Can easily span a wide-area network
- To allow for collaborations, grids generally use **virtual organizations**.
  - In essence, this is a grouping of users (or better: their IDs) that have the same access rights
  - The key questions are
    - how to authorize users from different administrative domains and
    - how to provide these authorized users with the access to resources

DCS: Distributed Computing Systems

Grid Computing Systems (cont'd)

- **Application**:
  - Use the grid computing environment
- **Collectivity** layer:
  - Handles access to multiple resources (resource discovery)
- **Connectivity** layer:
  - Communication protocols (access a remote resource, security)
- **Resource** layer:
  - Manage single resource (create a process)
- **Fabric** layer:
  - Interface to local resources (query, locking)



DCS: Distributed Computing Systems

Cloud Computing Systems

- “Cloud computing has become another buzzword after Web 2.0.”
- “We won’t compute on local computers, but on centralized facilities operated by third-party compute and storage utilities”
- “There are dozens of different definitions for cloud computing and there seems to be no consensus on what a cloud is.” *Here is one definition:*

  “A large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet.”
- “Cloud computing is not a completely new concept; it has intricate connection to the relatively new but thirteen-year established grid computing paradigm, and other relevant technologies such as utility computing, cluster computing, and distributed systems in general.”

I. Foster, Cloud Computing and Grid Computing 360-Degree Compared, Grid Computing Environments Workshop, 2008. GCE '08

DIS: Distributed Information Systems

- Organizations have legacy networked applications, but it is hard to make them interoperate
- Middleware can help
- Integration can take place at several levels
  - Client-servers wrap a number of request into one and have it executed as a **Distributed Transaction** (all or none of requests would be executed)
  - Applications can be detached from their databases or divided into several components, these applications need to directly communicate instead of req/reply: **Enterprise Application Integration** (EAI)

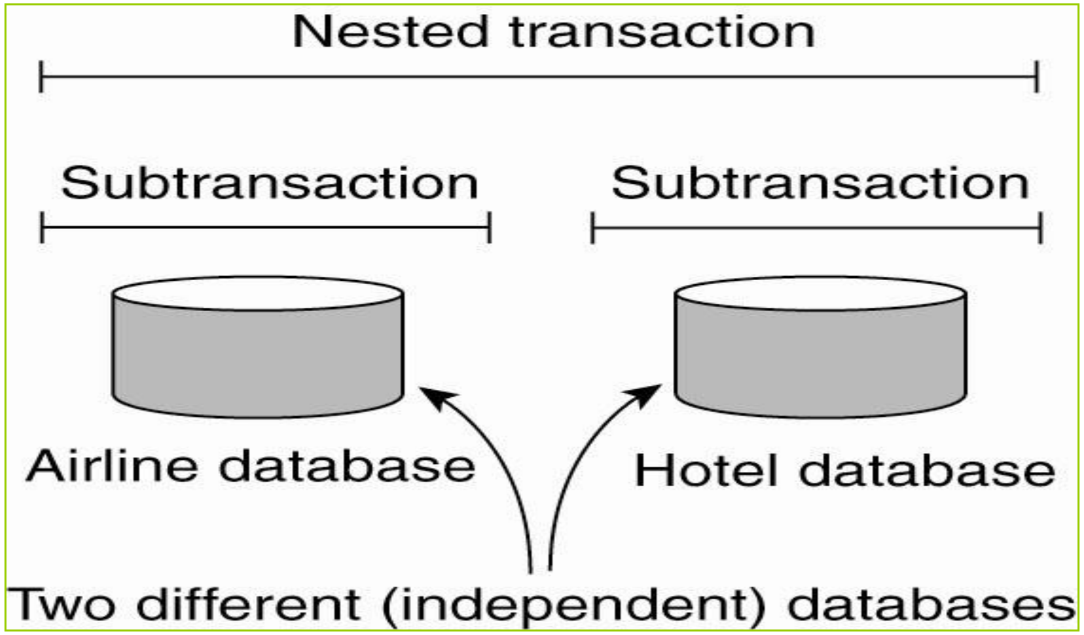
Characteristic properties of **transactions** (ACID)

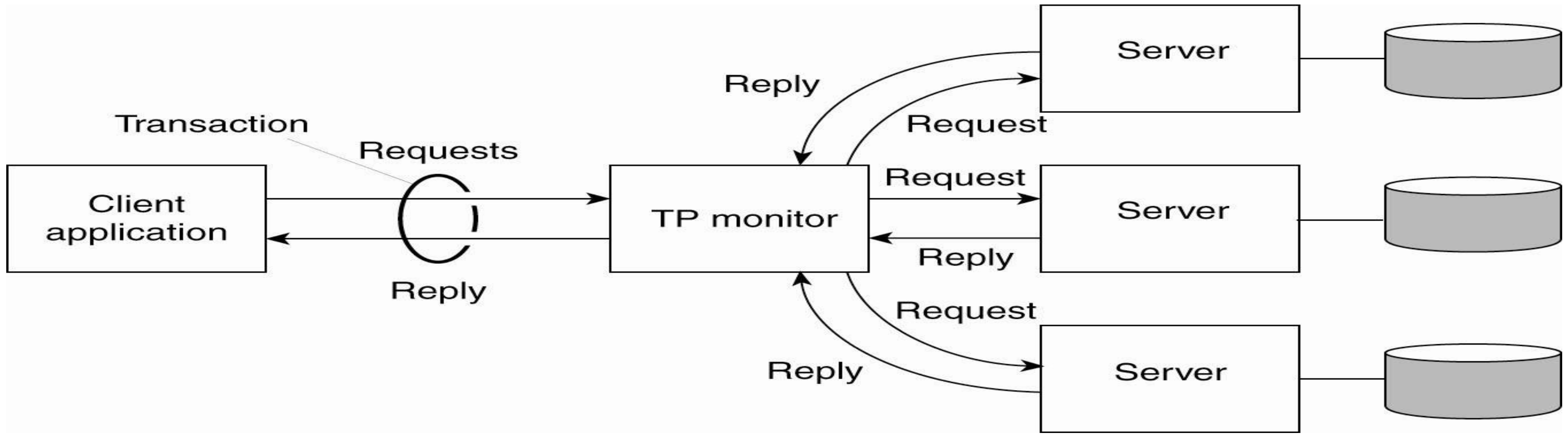
- **A**tomic: To the outside world, the transaction happens indivisibly;
  - All operations either succeed, or all of them fail;
- **C**onsistent: The transaction does not violate system invariants;
  - Not exclude the possibility of invalid, ***intermediate*** states
- **I**solated: Concurrent transactions do not interfere with each other
- **D**urable: Once a transaction commits, the changes are permanent

DIS: Distributed Information Systems

Distributed Databases: Transaction Process

- **TP Monitor:** coordinate the execution of a transaction (sub-transactions) when data is distributed across several servers

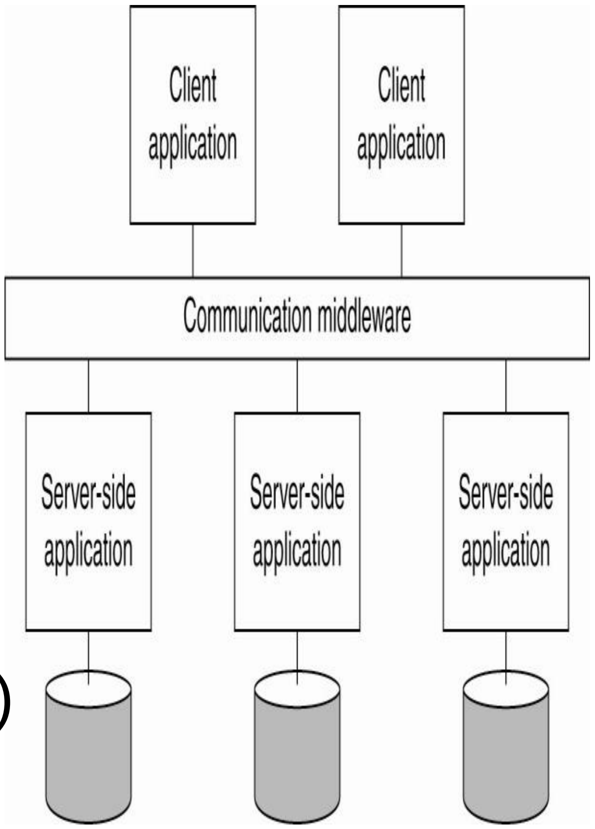




DIS: Distributed Information Systems

Enterprise Application Integration

- A TP monitor doesn’t separate apps from their databases.
- But we can do that and allow these applications to directly communicate
  - Use RPC or RMI
    - both applications must be up and running
    - know exactly how to refer to each other
  - Message-Oriented Middleware (MOM)
    - send data to a logical contact
    - publish/subscribe



DPS: Distributed Pervasive Systems

- So far we considered stable distributed systems (fixed nodes good connections)
- But this is not the case for the emerging next-generation of distributed systems in which mobile and embedded devices are used
- Some requirements
  - Computing **anywhere** and **anytime**
  - **Contextual change:** environment changes should be immediately accounted for.
  - **Ad hoc composition:** Each node may be used in a very different ways by different users. Requires ease-of-configuration.
  - **Sharing is the default:** Nodes come and go, providing sharable services and information. Calls again for simplicity.

Expose distribution instead of hiding it!

DPS: Distributed Pervasive Systems

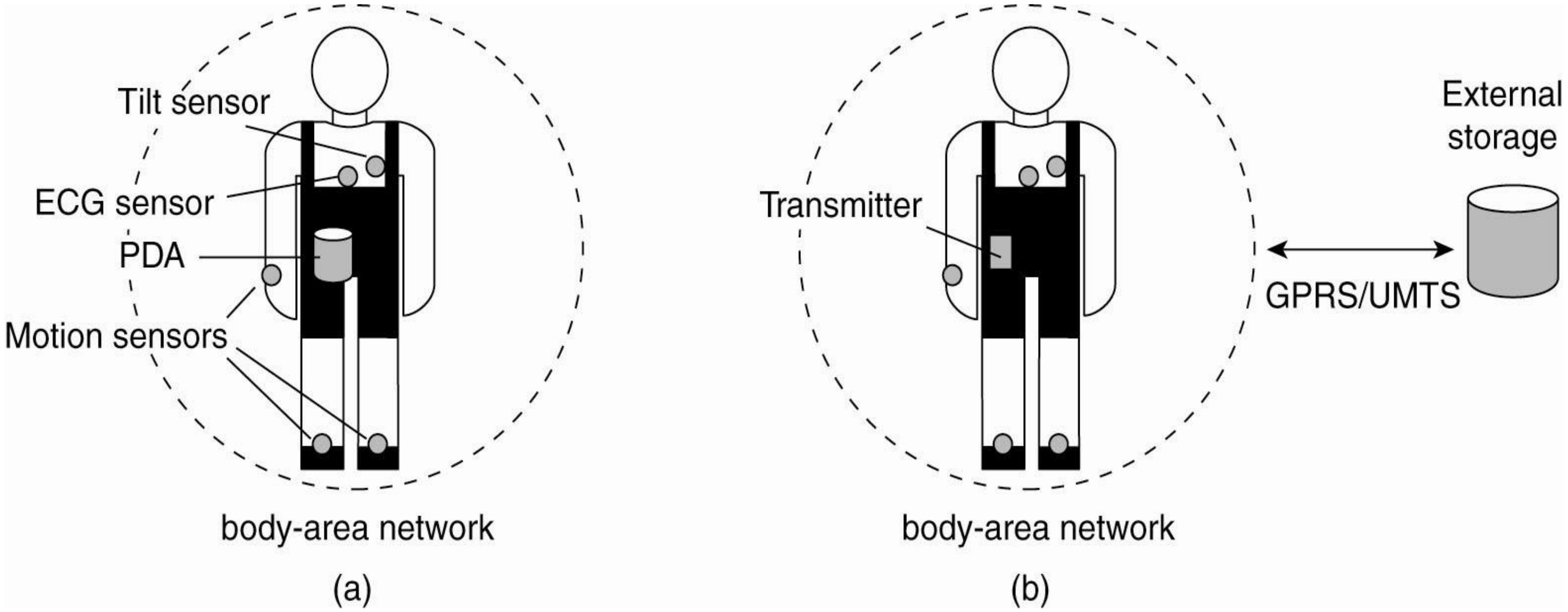
Home Systems

- Should be completely self-organizing:
- There should be no system administrator
- Provide a personal space for each of its users
- Simplest solution:
  - a centralized home box?
  - But how to access what you want?
    - Recommender programs will help…

DPS: Distributed Pervasive Systems

Electronic Health Care Systems

- Devices are physically close to a person
  - Where and how should monitored data be stored?
  - How can we prevent loss of crucial data?
  - How can **security** be enforced?
  - How can physicians provide online feedback?

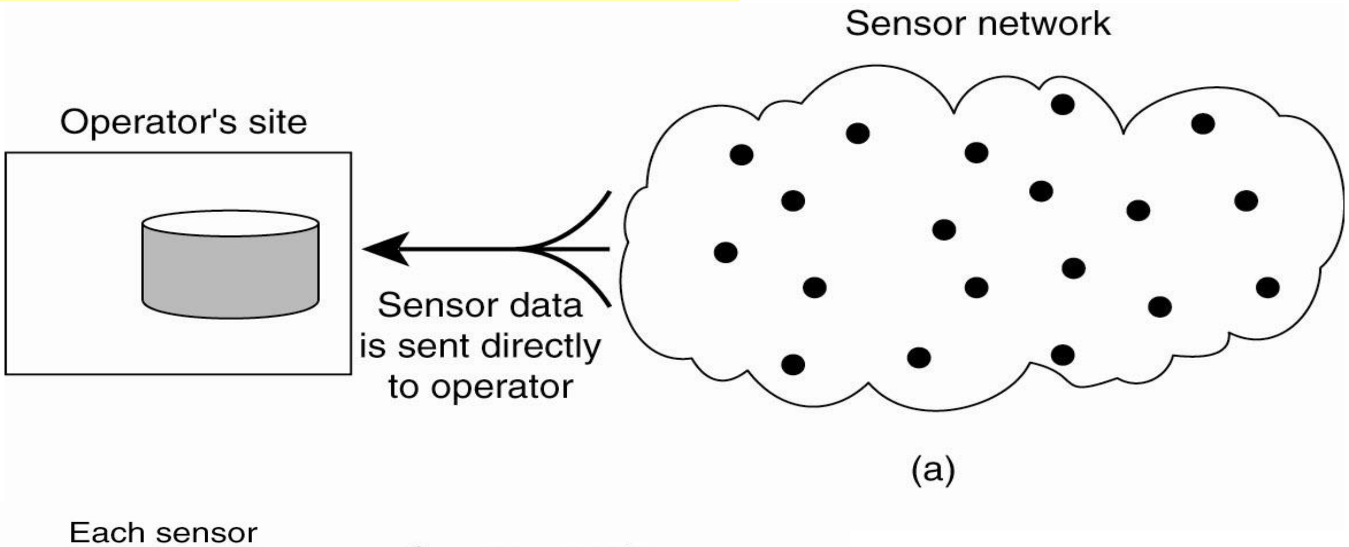


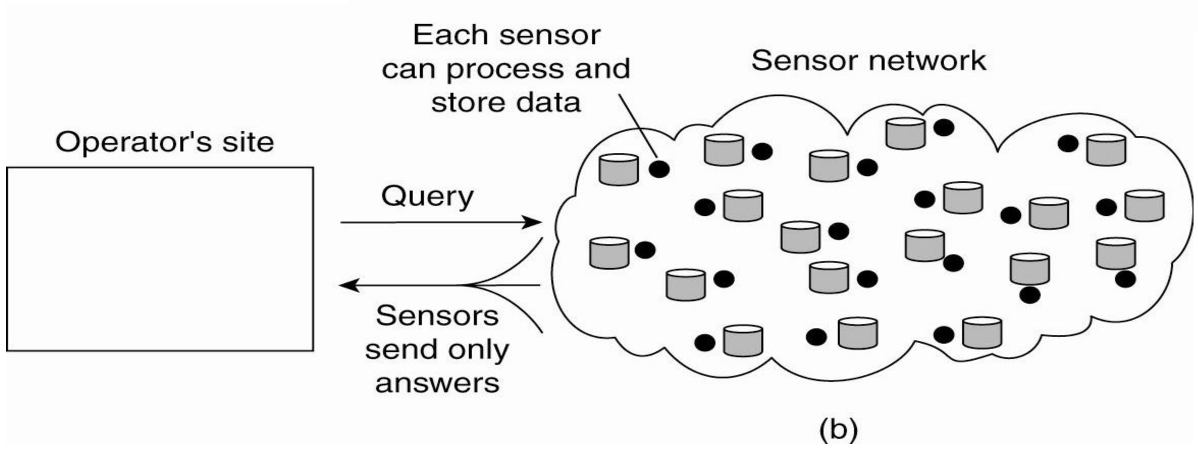
DPS: Distributed Pervasive Systems

Sensor Networks

The nodes to which sensors are attached are:

- Many (10s-1000s)
- Simple (small memory/compute/communication capacity)
- Often battery-powered (or even battery-less)




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

END