- **What a distributed system is:**
  - its basic characteristics and ingredients
- **Why we are interested in their design:**
  - motivation, benefits
  - design goals
- **What types we do distinguish:**
  - high-level classification
  - application domains

Material

Distributed Systems, 3rd edition, version 01

Maarten van Steen, Andrew

What is a distributed system

Leslie Lamport:

“*A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.*”

What is a distributed system

Coulouris, et al.

“A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages”

What is a distributed system

van Steen, Tanenbaum

“A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system”

Characteristics

Definitions may vary, but there is consensus on their main characteristics

- No global notion of time
- No global notion of state
- Heterogeneous resources
- Network communication
- Independent failures

Basic ingredients

- **Processing elements (nodes)**
  - autonomous (often geographically separated)
    - no global clock and no global state
  - full-fledged computers
    - desktop, laptop, smart phones, plug computers, ...
  - or a process, virtual machine
  - heterogeneous

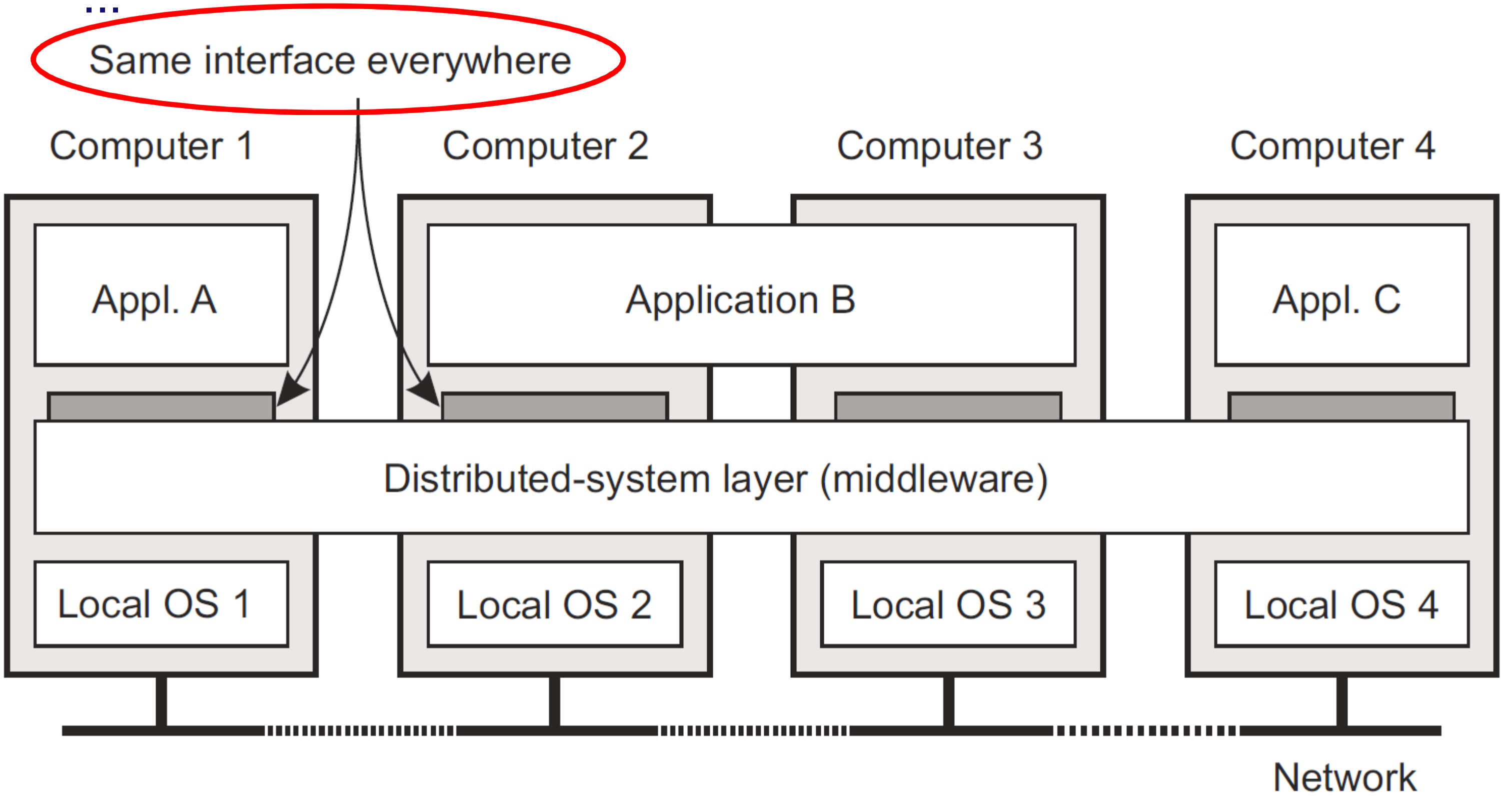
Heterogeneous resources in distributed systems refer to components or entities that have different characteristics, capabilities, or configurations. These resources may vary in terms of hardware, software, operating systems, network protocols, or any other attributes that distinguish them from one another.

- **Communication subsystem or network**
  - modeled as a graph
  - subject to dynamic changes
- **Software**
  - operating system(s), runtime systems
  - generic system services (middleware)
  - application/domain specific services (purpose of the system)

Middleware

A layer on top of the OS in every node providing generic services that make the distributed system appear as a single machine to the applications such as:

- communication, transactions, orchestration , reliability, security, accounting, ...



Middleware: software that provides services beyond those provided by the operating system to enable the various components of a distributed system to communicate and manage data e.g web servers, application servers, messaging and similar tools that support application development and delivery

Distributed systems (why)

- **Unavoidable**
  - Inherent distributed environment/application
  - Multiple data/resources/users at separated physical location

- **Associated benefits**
  - Reduced development/maintenance cost through modularity
  - Reduced operational cost through resource sharing
  - Improved performance and scalability through replication
  - Improved dependability through redundancy

- **Design goals**
  - In general, realizing these benefits

Design goals

Since distributed systems are complex, expensive, exist for a long time, and span a large geographical range, they must be designed to be

- **Efficient and effective**
  - resource combination and sharing
- **Transparent**
  - hiding their (internal) complexity
  - which makes them easier to understand and use
- **Scalable**
  - coping with growth,
- **Open**
  - allowing usage by, extension with, integration into and built from 3th party components and systems.

Distribution transparency

To obtain *distribution transparency*, a number of more specific transparencies all have to be achieved. In practice, especially failure transparency is extremely hard, if not impossible, to achieve (cf. Lamport definition).

Transparency	Description
Access	Hide differences in data representation and how objects are 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

Transparency means that any form of distributed system should hide its distributed nature from its users, appearing and functioning as a normal centralized system.

Scalability issues

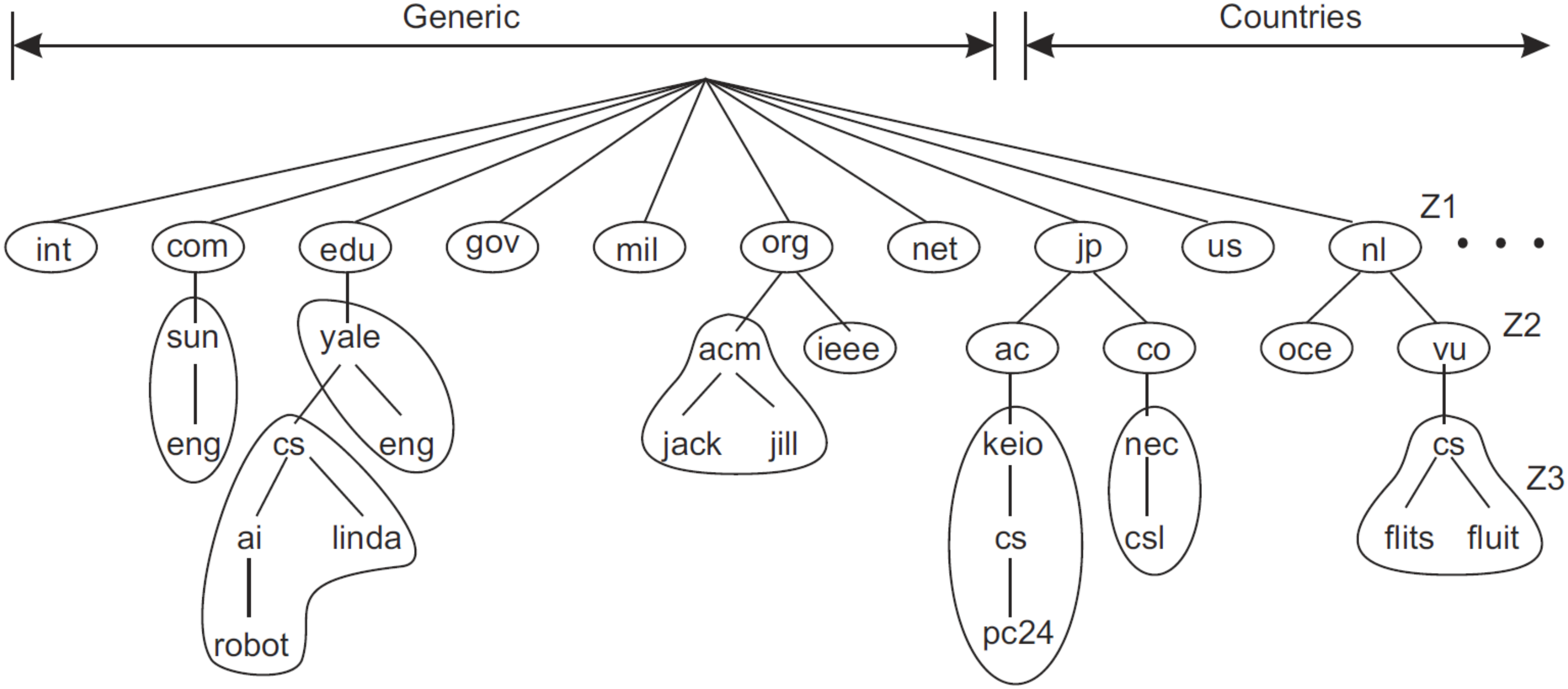
The ability to cope with growth

- **multiple dimensions:**
  - size, geographical spread, administrative domains
- **shift from a few dedicated high-performance resources to massive commodity resources:**
  - vertical scaling versus horizontal scaling
- **realized by a multitude of techniques:**
  - hiding communication latency
  - partitioning and distribution
  - replication and load balancing

DNS scaling example: partition and distribute

DNS resolution (map domain name `www.tue.nl` to IP-address 131.155.3.3)

- name space partitioned into zones
- administration differs per zone
- one authoritative server per zone
- system spans the entire globe



Openess issues

Allowing usage by, extension with, and integration into 3th party components and systems.

- **interoperability, composability, extensibility**
  - services with well-defined interfaces and service level agreements (SLAs)
    - specifications neutral and complete
  - tool support for generating boiler-plate code
- **separating policy from mechanism**
  - rich facilities for (run-time) configuration
    - i.e., policy specification
  - self-configuration to assist users

Pitfalls

The 8 fallacies of distributed computing
Peter Deutsch (7) , Brian Gosling (1)

1. The network is reliable.
2. The network is secure.
3. The network is homogeneous (G).
4. Topology doesn't change.
5. Latency is zero.
6. Bandwidth is infinite.
7. Transport cost is zero.
8. There is one administrator.

A good architect determines:

- the importance of each of these properties for the system at hand
- provides mechanisms to deal with the absence of such a property

Types of distributed systems

- **Systems for high performance computing**
  - cluster computing
  - grid computing
  - cloud computing
- **Distributed information systems**
  - distributed transactions
  - enterprise application integration
- **Pervasive systems**
  - ubiquitous computing systems
  - mobile computing systems
  - sensor networks

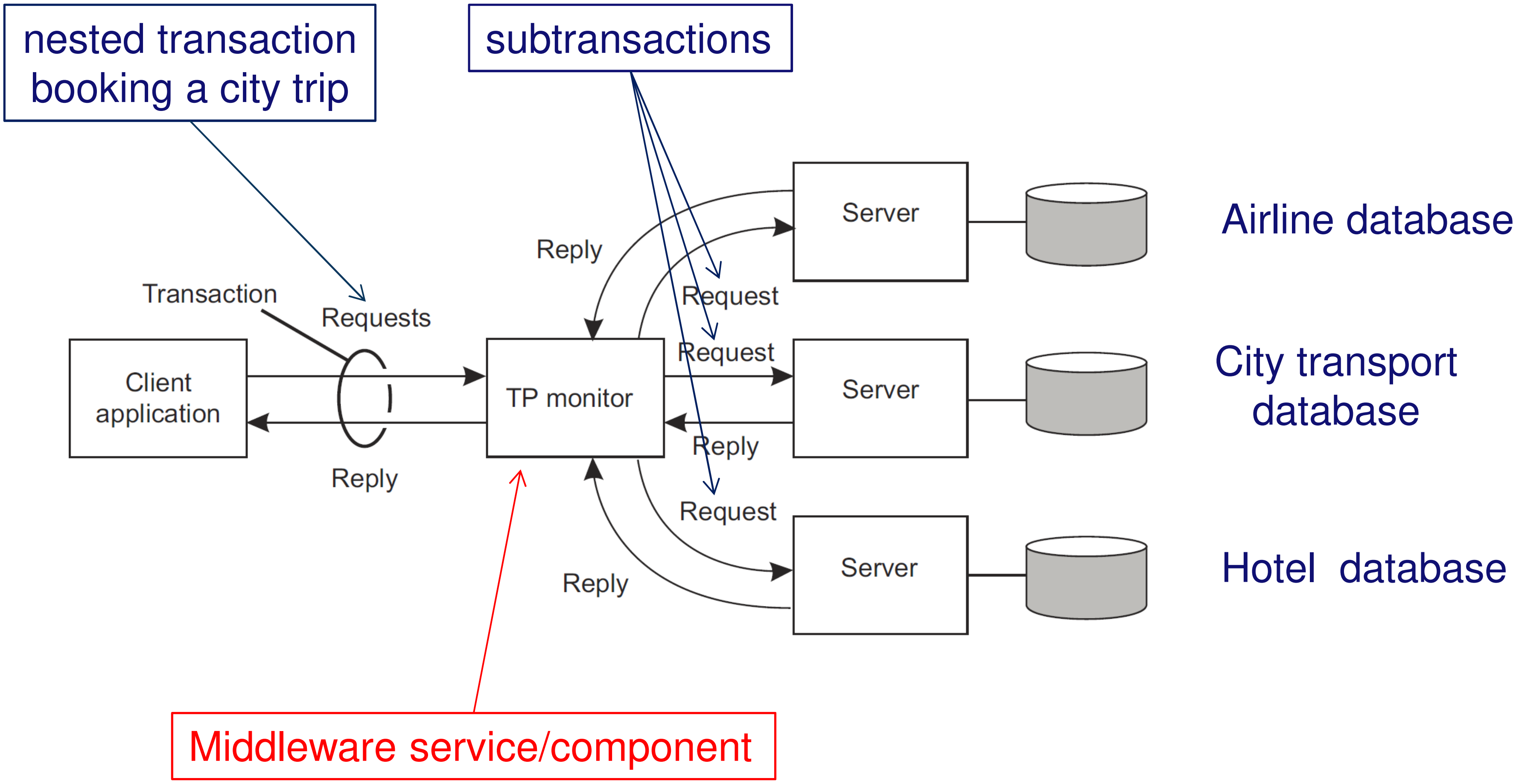
Systems for high performance computing

- **Cluster**
  - nodes run same OS and are connected by a high-speed network
  - master-worker nodes or
  - “fully” symmetric
    - process migration to achieve single-system image
- **Grid**
  - federation of systems; resources in multiple administrative domains
    - security issues,
    - virtual organizations through layered architecture
- **Cloud**
  - virtualized resources
    - pay per resource, guarantees through SLAs**,**
  - organized in service layers: IaaS, PaaS, SaaS

Distributed information systems

- **Distributed transaction processing**
  - nested transactions complicate maintaining transaction semantics
    - the ACID properties
  - coordination support by middleware
    - e.g.TP-monitor, special component for distributed commit
- **Enterprise application integration**
  - resource/service sharing between various applications in an enterprise
    - service-oriented approach (SOA)
  - message-oriented middleware (MOM) support
    - enterprise service bus (ESB)

Nested transaction processing



nested transaction booking a city trip

subtransactions

Airline database

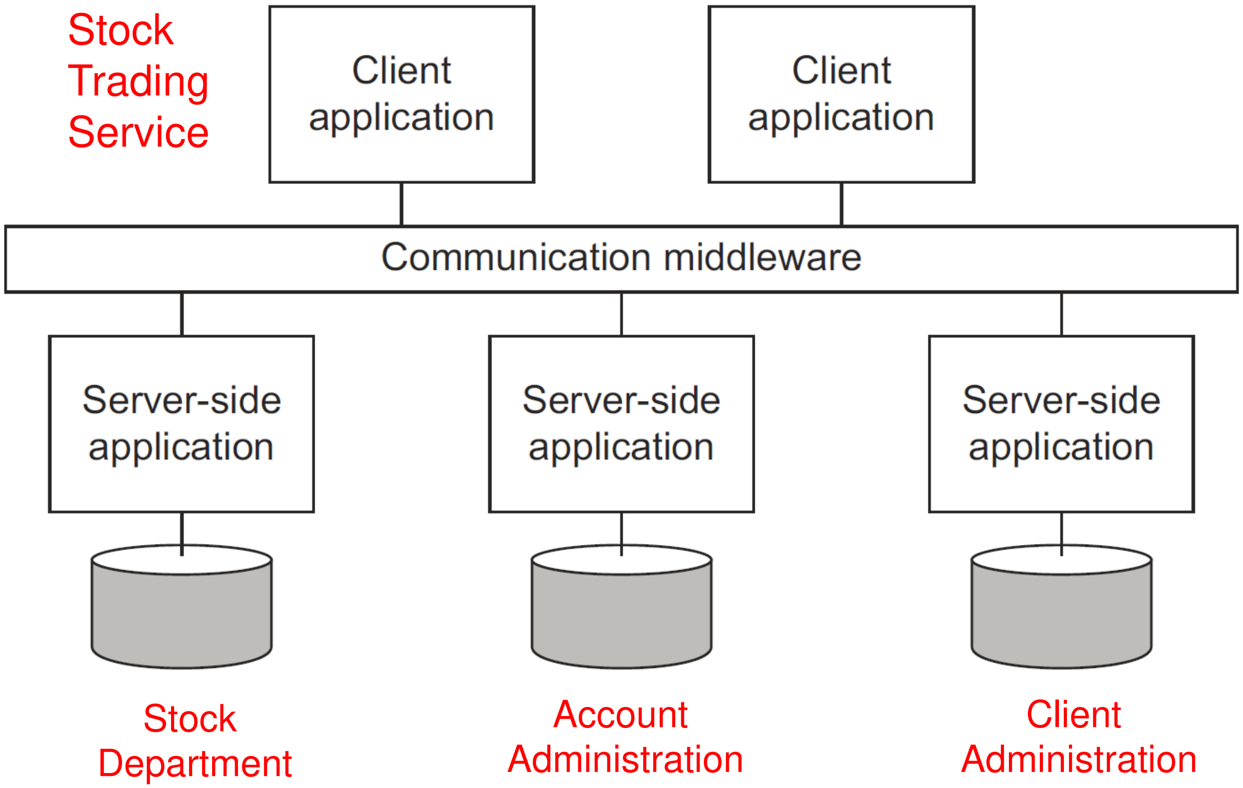
City transport database

Hotel database

Middleware service/component

Enterprise integration

Multiple client applications built by combining services (server-side apps) offered at various branches of an enterprise



Pervasive systems

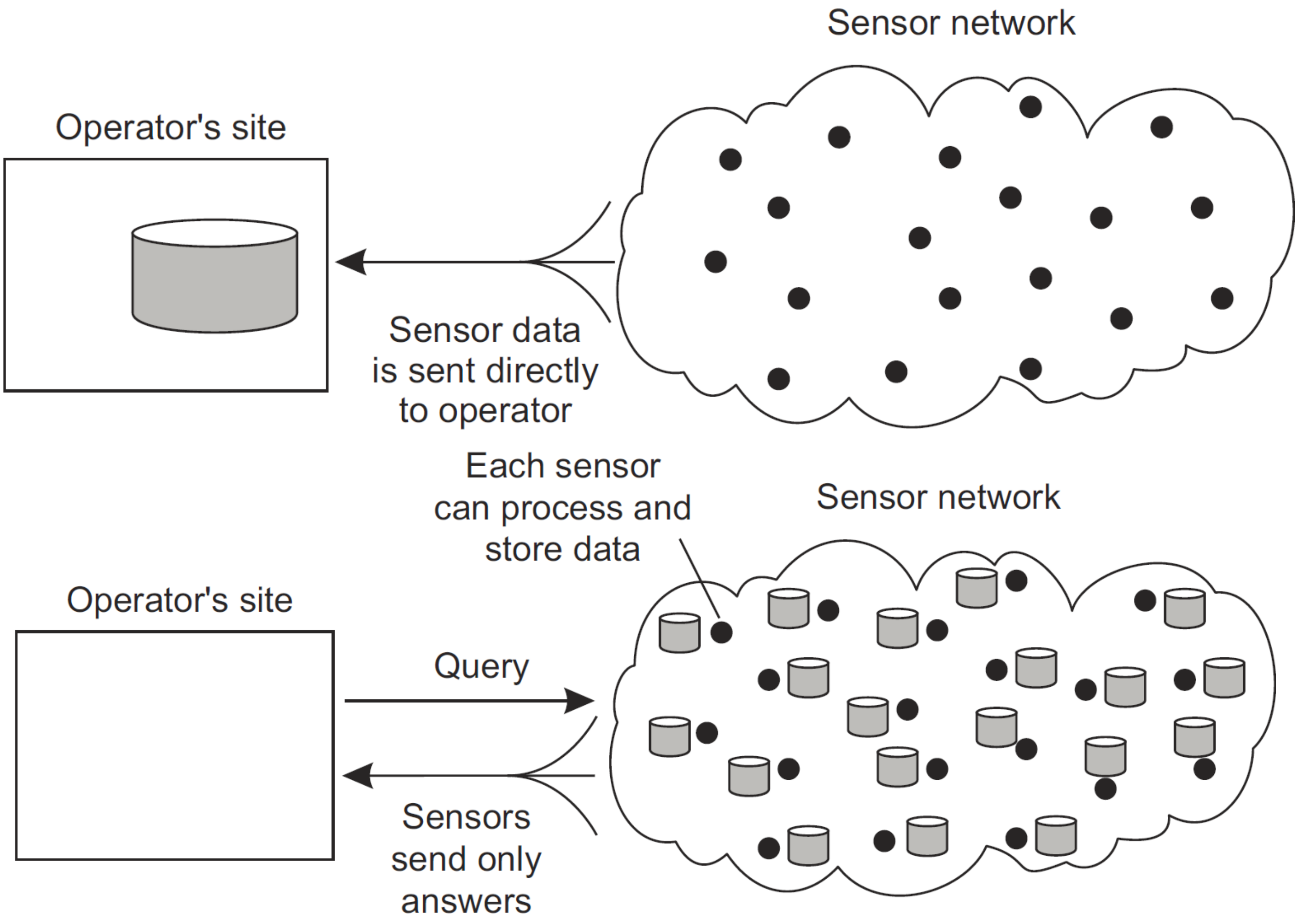
- **Ubiquitous computing systems**
  - devices are networked, context-aware, autonomous, intelligent,
  - interaction with users implicit and unobtrusive
- **Mobile computing systems**
  - wireless communication
  - changing topology and service availability
- **Sensor networks**
  - resource-constrained, specifically w.r.t. energy
  - programming support for communication
    - abstract neighborhoods, content-based addressing
  - emergent functionality through cooperation
    - aggregation, in-network data processing

Context (definition)

Dey and Abowd:

“*Any information that can be used to characterize the situation of entities (i.e., whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves.*”

In network data processing (bottom)



Sensor network

Operator's site

Sensor data
is sent directly
to operator

Each sensor
can process and
store data

Sensor network

Operator's site

Query

Sensors
send only
answers

Distributed systems (where)

Everywhere

Application domains

- Financial applications
- Manufacturing
- Reservation
- Transportation / Traffic
- Telecom
- Multimedia
- Social media
- Health care
- Surveillance
- Smart environments
- Automotive / Aviation
- Webshops
- ...