Distributed Systems

Lesson 1
Introduction

University of New York in Tirana
Master of Science in Computer Science
Prof. Dr. Marenglen Biba

Lesson 1

01: Introduction
02: Architectures
03: Processes
04: Communication
05: Naming
06: Synchronization
07: Consistency & Replication
08: Fault Tolerance
09: Distributed Object-Based Systems
10: Distributed File Systems
11: Distributed Web-Based Systems
12: Distributed Coordination-Based Systems
13: Amazon Web and Cloud Services

Introduction

1.1 DEFINITION OF A DISTRIBUTED SYSTEM

1.2 GOALS

- 1.2.1 Making Resources Accessible
- 1.2.2 Distribution Transparency
- 1.2.3 Openness
- 1.2.4 Scalability
- 1.2.5 Pitfalls

1.3 TYPES OF DISTRIBUTED SYSTEMS

- 1.3.1 Distributed Computing Systems
- 1.3.2 Distributed Information Systems
- 1.3.3 Distributed Pervasive Systems

Distributed Systems

- A distributed system is a piece of software that ensures that:
  - a collection of independent computers appears to its users as a single coherent system
- Two main aspects:
  - independent computers
  - single system

Distributed Systems

- One important characteristic of DSs is that differences between the various computers and the ways in which they communicate are mostly hidden from users.
  - The same holds for the internal organization of the distributed system.

- Another important characteristic is that users and applications can interact with a distributed system in a consistent and uniform way, regardless of where and when interaction takes place.

Distributed Systems

- In principle, distributed systems should also be relatively easy to expand or scale.
  - This characteristic is a direct consequence of having independent computers, but at the same time, hiding how these computers actually take part in the system as a whole.

- A distributed system will normally be continuously available, although perhaps some parts may be temporarily out of order.

- Users and applications should not notice that parts are being replaced or fixed, or that new parts are added to serve more users or applications.

Middleware

- In order to support heterogeneous computers and networks while offering a single-system view, distributed systems are often organized by means of a layer of software - that is, logically placed between:
  - a higher-level layer consisting of users and applications,
  - and a layer underneath consisting of operating systems and basic communication facilities,

- Accordingly, such a distributed system is sometimes called middleware.

Middleware

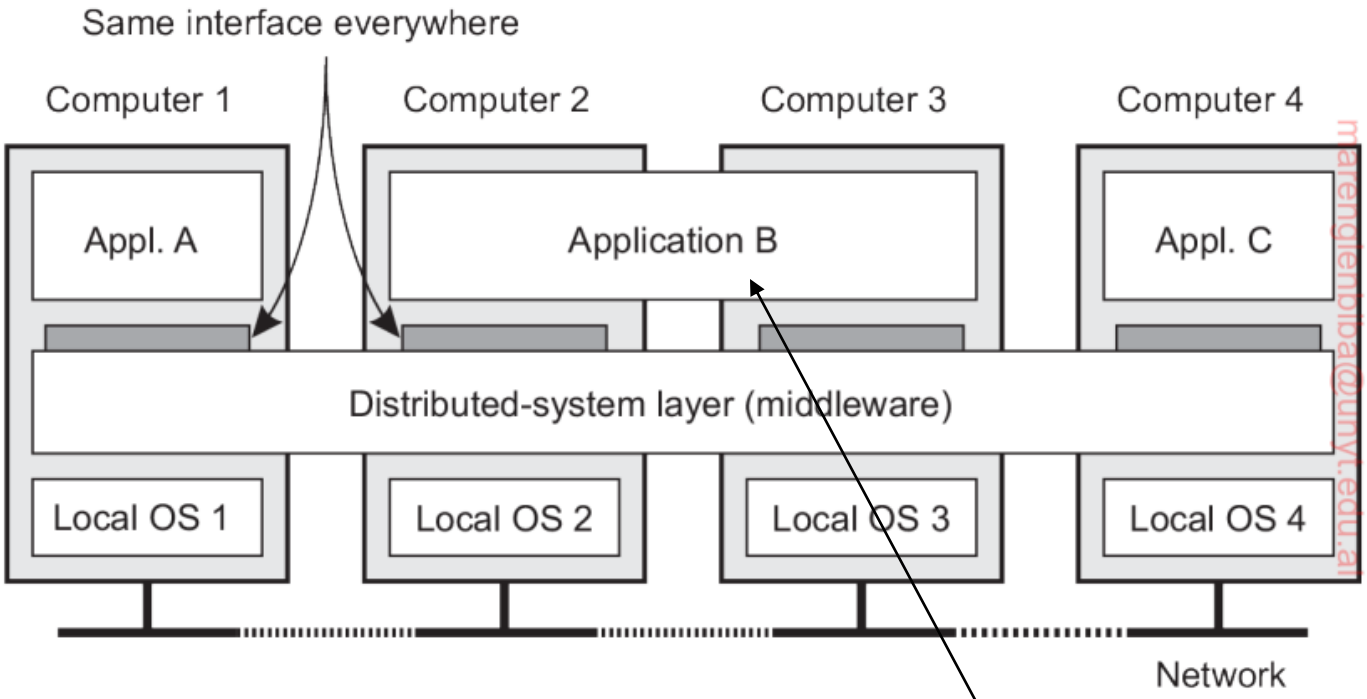


Figure 1.1: A distributed system organized in a middleware layer, which extends over multiple machines, offering each application the same interface.

Application B runs on two different Computers with different OSs

Introduction

1.1 DEFINITION OF A DISTRIBUTED SYSTEM

1.2 GOALS

- 1.2.1 Making Resources Accessible
- 1.2.2 Distribution Transparency
- 1.2.3 Openness
- 1.2.4 Scalability
- 1.2.5 Pitfalls

1.3 TYPES OF DISTRIBUTED SYSTEMS

- 1.3.1 Distributed Computing Systems
- 1.3.2 Distributed Information Systems
- 1.3.3 Distributed Pervasive Systems

Making resources available

- The main goal of a distributed system is to make it easy for the users (and applications) to access remote resources, and to share them in a controlled and efficient way.

- Resources can be just about anything, but typical examples include things like:
  - printers, computers, storage facilities, data, files, Web pages, and networks, to name just a few.

Making resources available

- There are many reasons for wanting to share resources.
- One obvious reason is that of economics. For example, 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, image setters, and other expensive peripherals.

Making resources available

- Connecting users and resources also makes it easier to collaborate and exchange information, as is clearly illustrated by the success of the Internet with its simple protocols for exchanging files, mail. documents, audio, and video.

- The connectivity of the Internet is now leading to numerous virtual organizations in which geographically widely-dispersed groups of people work together by means of groupware, that is, software for collaborative editing, teleconferencing, and so on.

- Likewise, the Internet connectivity has enabled electronic commerce allowing us to buy and sell all kinds of goods without actually having to go to a store or even leave home.

Introduction

1.1 DEFINITION OF A DISTRIBUTED SYSTEM

1.2 GOALS

- 1.2.1 Making Resources Accessible
- 1.2.2 Distribution Transparency
- 1.2.3 Openness
- 1.2.4 Scalability
- 1.2.5 Pitfalls

1.3 TYPES OF DISTRIBUTED SYSTEMS

- 1.3.1 Distributed Computing Systems
- 1.3.2 Distributed Information Systems
- 1.3.3 Distributed Pervasive Systems

Distribution transparency

- An important goal of a distributed system is to hide the fact that its processes and resources are physically distributed across multiple computers.

- A distributed system that is able to present itself to users and applications as if it were only a single computer system is said to be transparent.

- There are different kinds of transparency in distributed systems.

Distribution transparency

Transp.	Description
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 from a client the ability of a system to change the location of an object to which the client is bound
Replication	Hides the fact that an object or its state may be replicated and that replicas reside at different locations
Concurrency	Hides the coordination of activities between objects to achieve consistency at a higher level
Failure	Hides failure and possible recovery of objects

Introduction

1.1 DEFINITION OF A DISTRIBUTED SYSTEM

1.2 GOALS

- 1.2.1 Making Resources Accessible
- 1.2.2 Distribution Transparency
- 1.2.3 Openness
- 1.2.4 Scalability
- 1.2.5 Pitfalls

1.3 TYPES OF DISTRIBUTED SYSTEMS

- 1.3.1 Distributed Computing Systems
- 1.3.2 Distributed Information Systems
- 1.3.3 Distributed Pervasive Systems

Openness

- An open distributed system is a 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).

Interface Definition Language (IDL)

- Interface definitions written in an IDL nearly always capture only the syntax of services.
- In other words, they specify precisely the names of the functions that are available together with types of the parameters, return values, possible exceptions that can be raised, and so on.
- The hard part is specifying precisely what those services do, that is, the semantics of interfaces.
- In practice, such specifications are always given in an informal way by means of natural language.

Introduction

1.1 DEFINITION OF A DISTRIBUTED SYSTEM

1.2 GOALS

- 1.2.1 Making Resources Accessible
- 1.2.2 Distribution Transparency
- 1.2.3 Openness
- 1.2.4 Scalability
- 1.2.5 Pitfalls

1.3 TYPES OF DISTRIBUTED SYSTEMS

- 1.3.1 Distributed Computing Systems
- 1.3.2 Distributed Information Systems
- 1.3.3 Distributed Pervasive Systems

Scalability

- Scalability of a system can be measured along at least three different dimensions.
  - First, a system can be scalable with respect to its size, meaning that we can easily add more users and resources to the system.
  - Second, a geographically scalable system is one in which the users and resources may lie far apart.
  - Third, a system can be administratively scalable, meaning that it can still be easy to manage even if it spans many independent administrative organizations.

- Unfortunately, a system that is scalable in one or more of these dimensions often exhibits some loss of performance as the system scales up.

Challenges of scalability

- At least three components:
  - Number of users and/or processes (size scalability)
  - Maximum distance between nodes (geographical scalability)
  - Number of administrative domains (administrative scalability)

Observation

- Most systems account only, to a certain extent, for size scalability.
  - The (non)solution: powerful servers.
- Today, the challenge lies in geographical and administrative scalability.

Centralized Services

- Many services are centralized in the sense that they are implemented by means of only a single server running on a specific machine in the distributed system.

- The problem with this scheme is obvious: the server can become a bottleneck as the number of users and applications grows.

- Even if we have virtually unlimited processing and storage capacity, communication with that server will eventually prohibit further growth.

Centralized Services

- Unfortunately. using only a single server is sometimes unavoidable.
- Imagine that we have a service for managing highly confidential information such as medical records, bank accounts. and so on.
  - In such cases, it may be best to implement that service by means of a single server in a highly secured separate room, and protected from other parts of the distributed system through special network components.
  - Copying the server to several locations to enhance performance maybe out of the question as it would make the service less secure.

- How should we keep track of the telephone numbers and addresses of 50 million people?
- Imagine how the Internet would work if its Domain Name System (DNS) was still implemented as a single table.

Decentralized Services

- Only decentralized algorithms should be used!
- These algorithms generally have the following characteristics, which distinguish them from centralized algorithms:
  1. No machine has complete information about the system state.
  2. Machines make decisions based only on local information,
  3. Failure of one machine does not ruin the algorithm.
  4. There is no implicit assumption that a global clock exists. (we will explain the “clock issue” during the course)

Scaling Techniques

- In most cases, scalability problems in distributed systems appear as performance problems caused by limited capacity of servers and network.

- There are now basically only three techniques for scaling:
  - hiding communication latencies
  - distribution
  - replication

Hiding communication latencies

- Avoid waiting for responses; do something else:
  - Make use of asynchronous communication
  - Have separate handler for incoming response
  - Problem: not every application fits this model
    - In interactive applications when a user sends a request he will generally have nothing better to do than to wait for the answer.
    - In such cases, a much better solution is to reduce the overall communication,
      - for example, by moving part of the computation that is normally done at the server to the client process requesting the service.

Reducing overall communication

- A typical case where this approach works is accessing databases using forms.
  - Filling in forms can be done by sending a separate message for each field, and waiting for an acknowledgment from the server.
  - For example, the server may check for syntactic errors before accepting an entry.

- A much better solution is to ship the code for filling in the form, and possibly checking the entries, to the client, and have the client return a completed form.
  - This approach of shipping code is now widely supported by the Web in the form of Java applets and Javascript.

Shipping code to clients

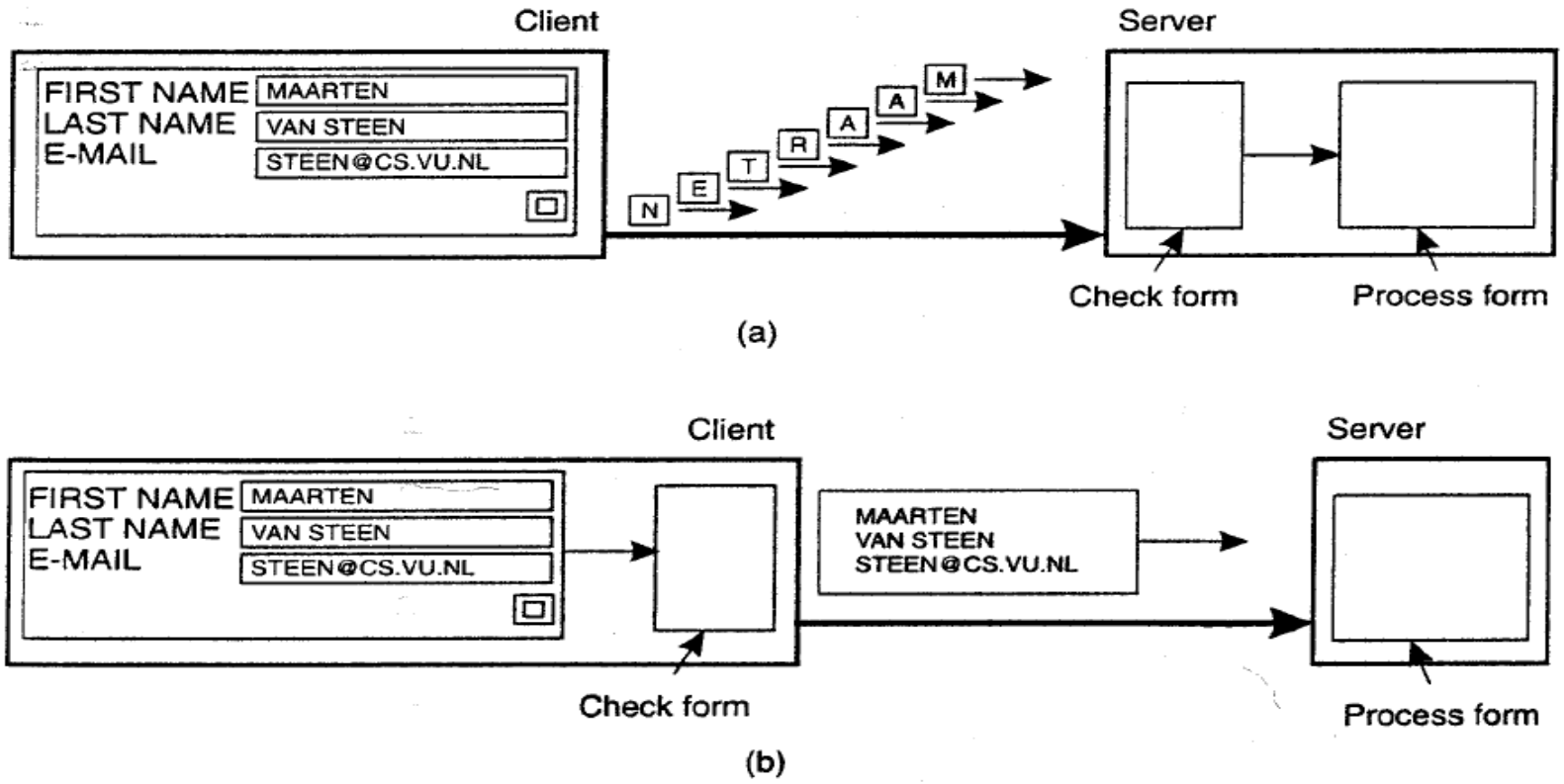


Figure 1-4. The difference between letting (a) a server or (b) a client check forms as they are being filled.

Distribution

- Another important scaling technique is distribution.
- Distribution involves taking a component, splitting it into smaller parts, an subsequently spreading those parts across the system.
- An excellent example of distribution is the Internet Domain Name System (DNS).
  - The DNS name space is hierarchically organized into a tree of **domains,** which are divided into non overlapping **zones.**
  - The names in each zone are handled by a single name server.
  - One can think of each path name, being the name of a host in the Internet, and thus associated with a network address of that host.
  - Basically, resolving a name means returning the network address of the associated host.

Domain Name System

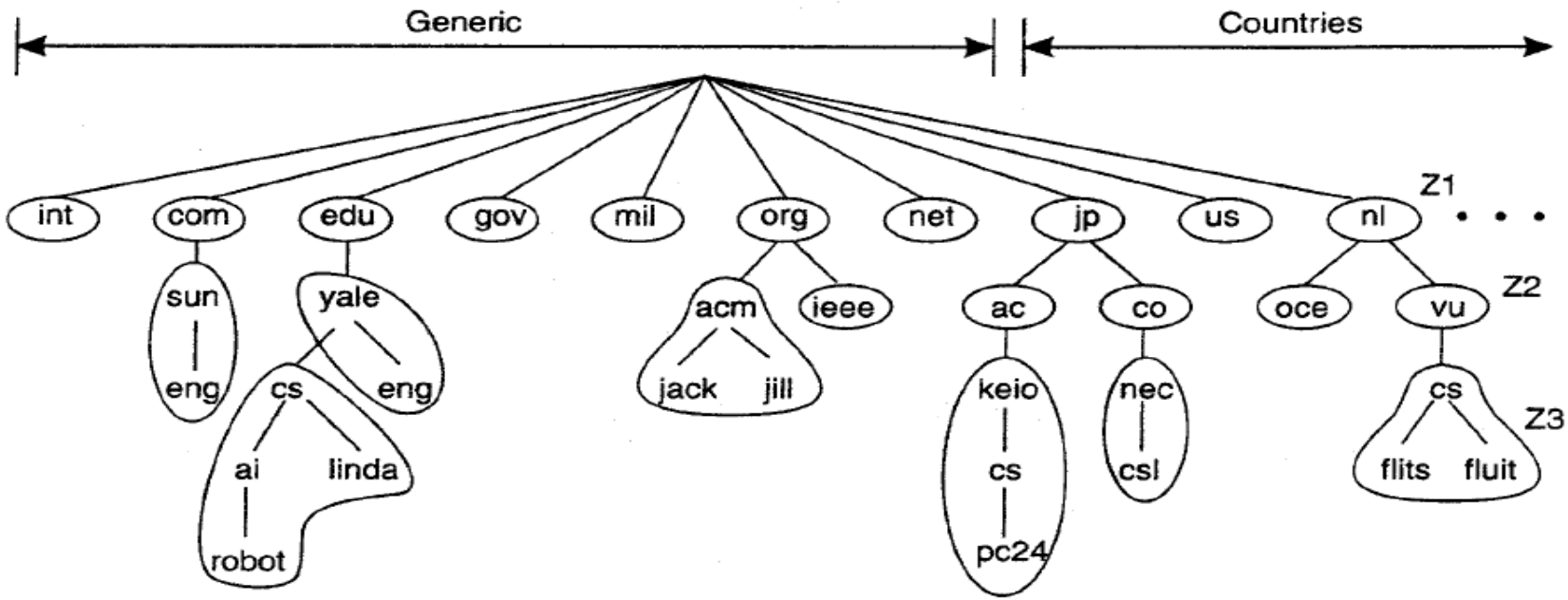


Figure 1-5. An example of dividing the DNS name space into zones.

World Wide Web

- As another example, consider the World Wide Web. To most users, the Web appears to be an enormous document-based information system in which each document has its own unique name in the form of a URL.
  - Conceptually, it may even appear as if there is only a single server.
  - However, the Web is physically distributed across a large number of servers, each handling a number of Web documents.
  - The name of the server handling a document is encoded into that document's URL.
  - It is only because of this distribution of documents that the Web has been capable of scaling to its current size.

Replication

- Considering that scalability problems often appear in the form of performance degradation, it is generally a good idea to actually replicate components across a distributed system.
- Replication not only increases availability, but also helps to balance the load between components leading to better performance.
- Also, in geographically widely-dispersed systems, having a copy nearby can hide much of the communication latency problems.
- Make copies of data available at different machines:
  - Replicated file servers and databases
  - Mirrored Web sites
  - Web caches (in browsers and proxies)
  - File caching (at server and client)

Caching and replication

- Caching is a special form of replication, although the distinction between the two is often hard to make or even artificial.
- As in the case of replication, caching results in making a copy of a resource, generally in the proximity of the client accessing that resource.
- However, in contrast to replication, caching is a decision made by the client of a resource, and not by the owner of a resource.
- Also, caching happens on demand whereas replication is often planned in advance.

Replication, copies may be different

- There is one serious drawback to caching and replication that may adversely affect scalability.
- Because we now have multiple copies of a resource, modifying one copy makes that copy different from the others.
- Consequently, caching and replication leads to consistency problems.

Inconsistency

- To what extent inconsistencies can be tolerated depends highly on the usage of a resource.
  - For example, many Web users find it acceptable that their browser returns a cached document of which the validity has not been checked for the last few minutes.
  - However, there are also many cases in which strong consistency guarantees need to be met, such as in the case of electronic stock exchanges and auctions.

Inconsistency

- The problem with strong consistency is that an update must be immediately propagated to all other copies.
  - **Moreover, if two updates happen concurrently, it is often also required that each copy is updated in the same order!!!**
- Situations such as these generally require some global synchronization mechanism.
- Unfortunately, such mechanisms are extremely hard or even impossible to implement in a scalable way.

Is Scaling really feasible?

- When considering these scaling techniques, one could argue that size scalability is the least problematic from a technical point of view.
  - In many cases, simply increasing the capacity of a machine will save the day (at least temporarily and perhaps at significant costs).
- Geographical scalability is a much tougher problem as Mother Nature is getting in our way.
- Nevertheless, practice shows that combining distribution, replication, and caching techniques with different forms of consistency will often prove sufficient in many cases.
- Finally, administrative scalability seems to be the most difficult problem to solve, partly because we need to deal with nontechnical issues, such as politics of organizations and human collaboration.

Introduction

1.1 DEFINITION OF A DISTRIBUTED SYSTEM

1.2 GOALS

- 1.2.1 Making Resources Accessible
- 1.2.2 Distribution Transparency
- 1.2.3 Openness
- 1.2.4 Scalability
- 1.2.5 Pitfalls

1.3 TYPES OF DISTRIBUTED SYSTEMS

- 1.3.1 Distributed Computing Systems
- 1.3.2 Distributed Information Systems
- 1.3.3 Distributed Pervasive Systems

Pitfalls

- Developing distributed systems can be a formidable task.
- As we will see many times throughout this course, there are so many issues to consider at the same time that it seems that only complexity can be the result.
- However, by following a number of design principles, distributed systems can be developed that strongly adhere to the goals we set out here.

Principles: hold them tight

- Many principles follow the basic rules of decent software engineering and will not be repeated here.
- However, distributed systems differ from traditional software because components are dispersed across a network.
- Not taking this dispersion into account during design time is what makes so many systems needlessly complex and results in mistakes that need to be patched later on.

False assumptions

- Peter Deutsch, then at Sun Microsystems, formulated these mistakes as the following 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.

Short bio

- L Peter Deutsch or Peter Deutsch (born Laurence Peter Deutsch) is the founder of Aladdin Enterprises and creator of Ghostscript, a free software PostScript and Pdf interpreter.
- Deutsch's other work includes the definitive Smalltalk implementation that, among other innovations, inspired Java just-in-time technology 15 or-so years later.
- He also wrote the PDP-1 Lisp 1.5 implementation, Basic PDP-1 LISP, "while still in short pants" between the age of 12-15 years old.
- He is also the author of a number of RFCs, and the The Eight Fallacies of Distributed Computing.
- Deutsch received a Ph.D. in Computer Science from the University of California, Berkeley in 1973. He has done stints at Xerox PARC and Sun Microsystems. In 1994 he was inducted as a Fellow of the Association for Computing Machinery.

Introduction

1.1 DEFINITION OF A DISTRIBUTED SYSTEM

1.2 GOALS

- 1.2.1 Making Resources Accessible
- 1.2.2 Distribution Transparency
- 1.2.3 Openness
- 1.2.4 Scalability
- 1.2.5 Pitfalls

1.3 TYPES OF DISTRIBUTED SYSTEMS

- 1.3.1 Distributed Computing Systems
- 1.3.2 Distributed Information Systems
- 1.3.3 Distributed Pervasive Systems

Distributed Computing Systems

- An important class of distributed systems is the one used for high-performance computing tasks.

- In cluster computing the underlying hardware consists of a collection of similar workstations or PCs, closely connected by means of a high-speed local-area network.
  - In addition, each node runs the same operating system.

- The situation becomes quite different in the case of grid computing.
- Grids consist of distributed systems that are often constructed as a federation of computer systems, where each system may fall under a different administrative domain, and may be very different when it comes to hardware, software, and deployed network technology.

Cluster Computing

- Cluster computing systems became popular when the price/performance ratio of personal computers and workstations improved.

- At a certain point, it became financially and technically attractive to build a supercomputer using off-the-shelf technology by simply hooking up a collection of relatively simple computers in a high-speed network.

- In virtually all cases, cluster computing is used for parallel programming in which a single (compute intensive) program is run in parallel on multiple machines.

Cluster Computing: Linux-based Beowulf clusters

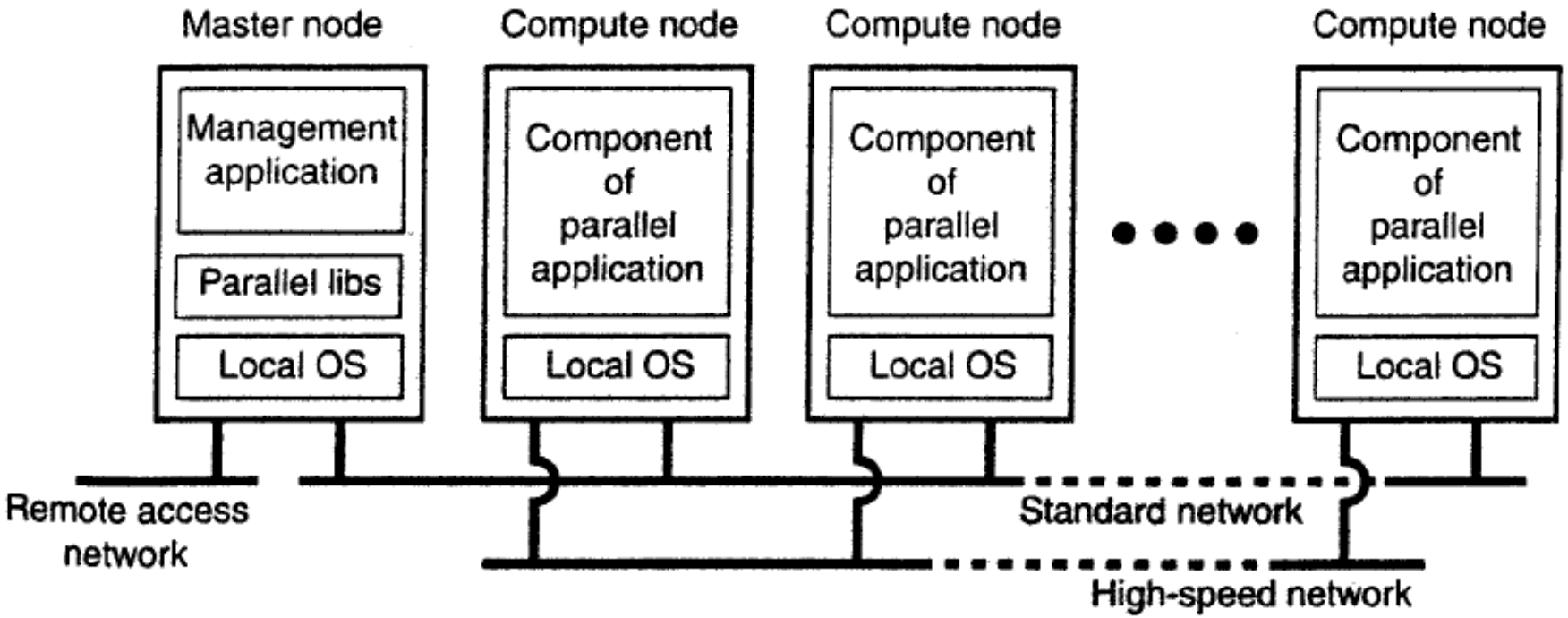
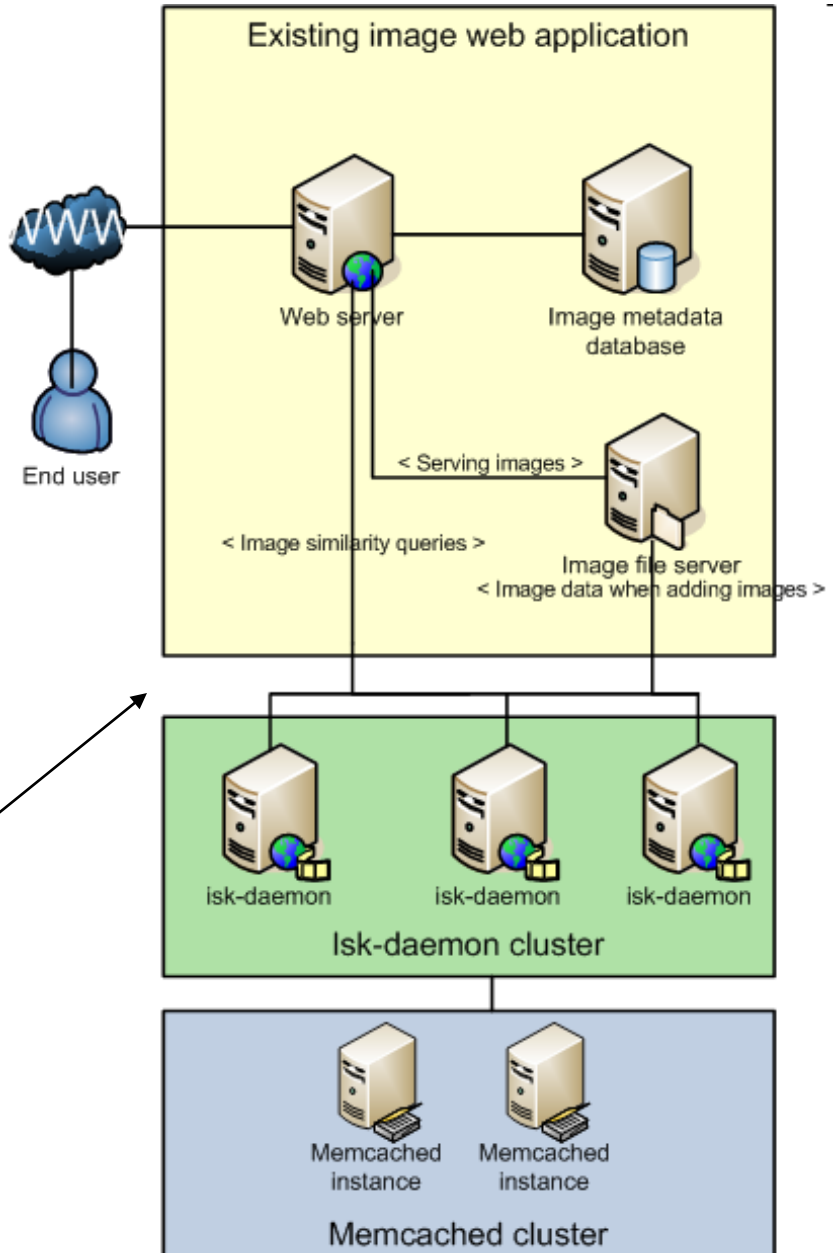


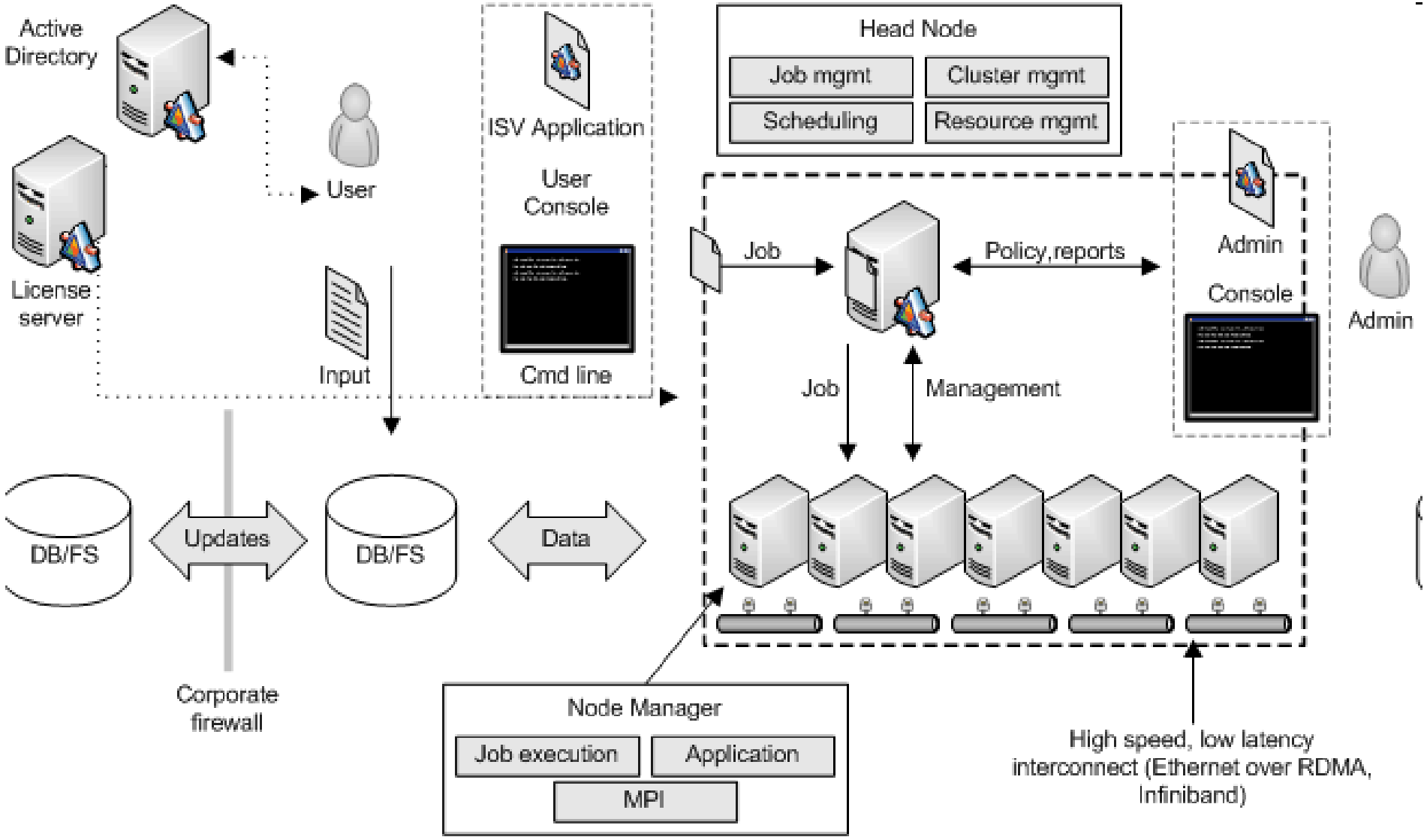
Figure 1-6. An example of a cluster computing system.

Cluster computing for web resources



Cluster to serve web resources

Cluster Architecture



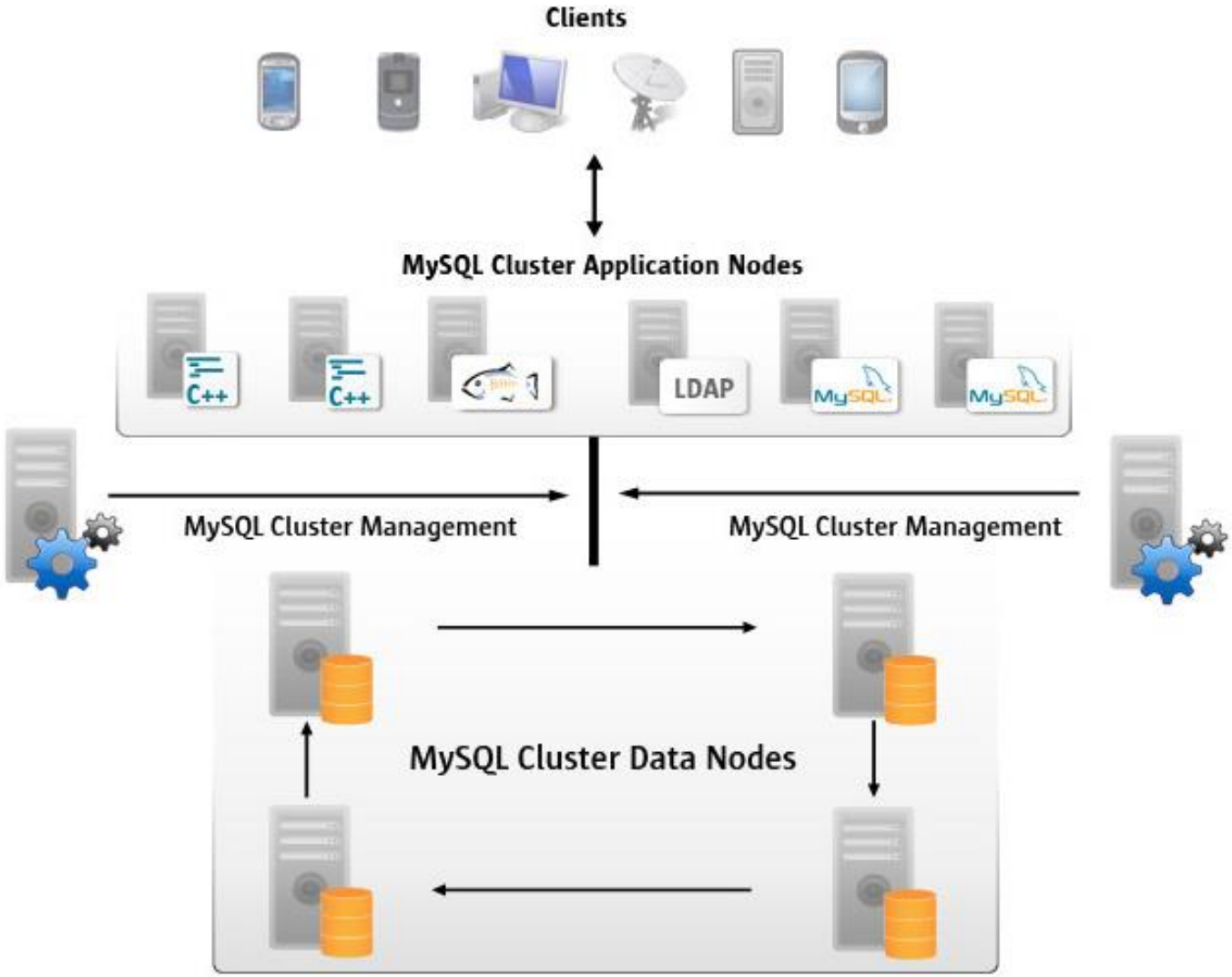
Database clusters: MySQL

- MySQL Cluster is a high availability database which leverages a shared-nothing data storage architecture.

- The system consists of multiple nodes which can be distributed across hosts to ensure continuous availability in the event of a data node, hardware or network failure.

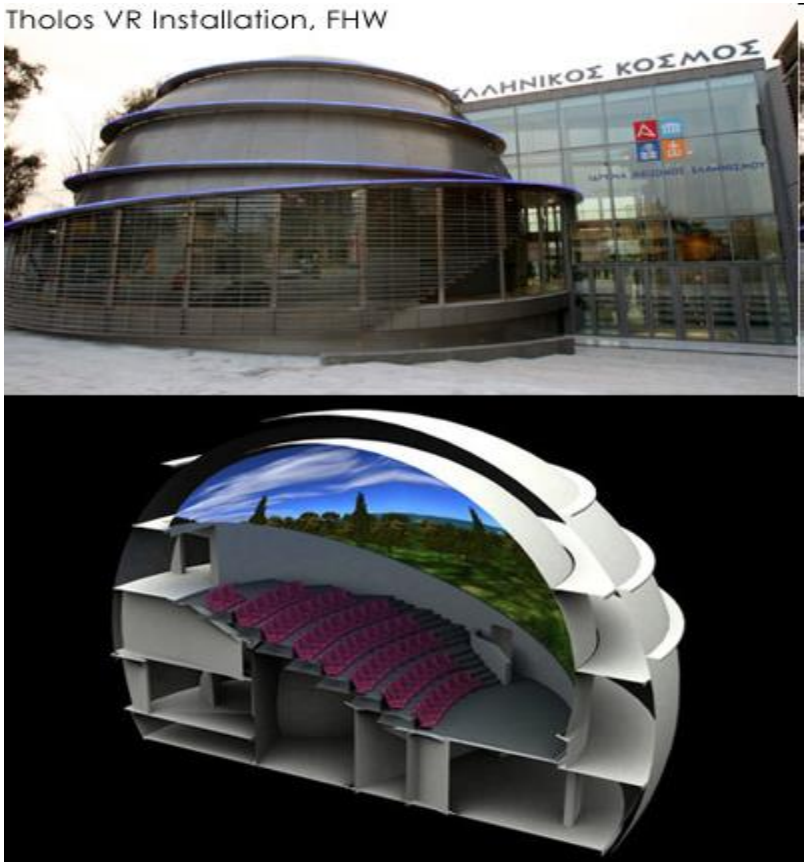
- MySQL Cluster Carrier Grade Edition uses a storage engine, consisting of a set of data nodes to store data, which is accessed through a native C++ API, Java, LDAP or standard SQL interface.

MySQL Cluster

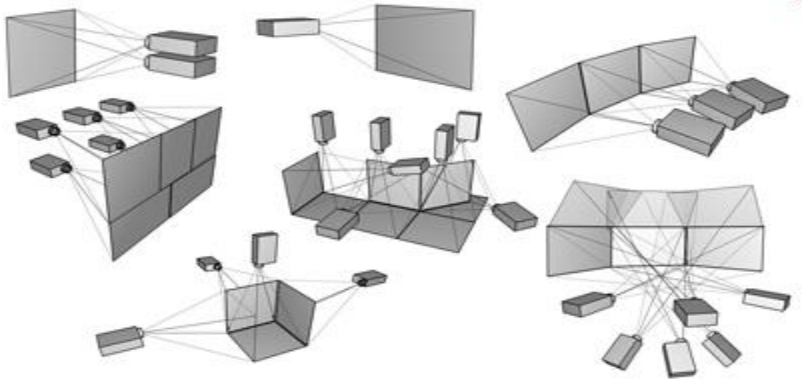


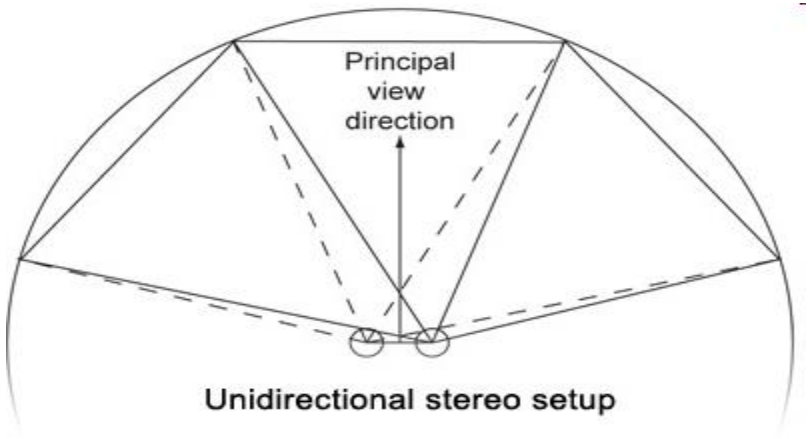
Clusters at NASA

Cluster Example: **The "Tholos" Virtual Reality Dome Theater**



Example: **The "Tholos" Virtual Reality Dome Theater**





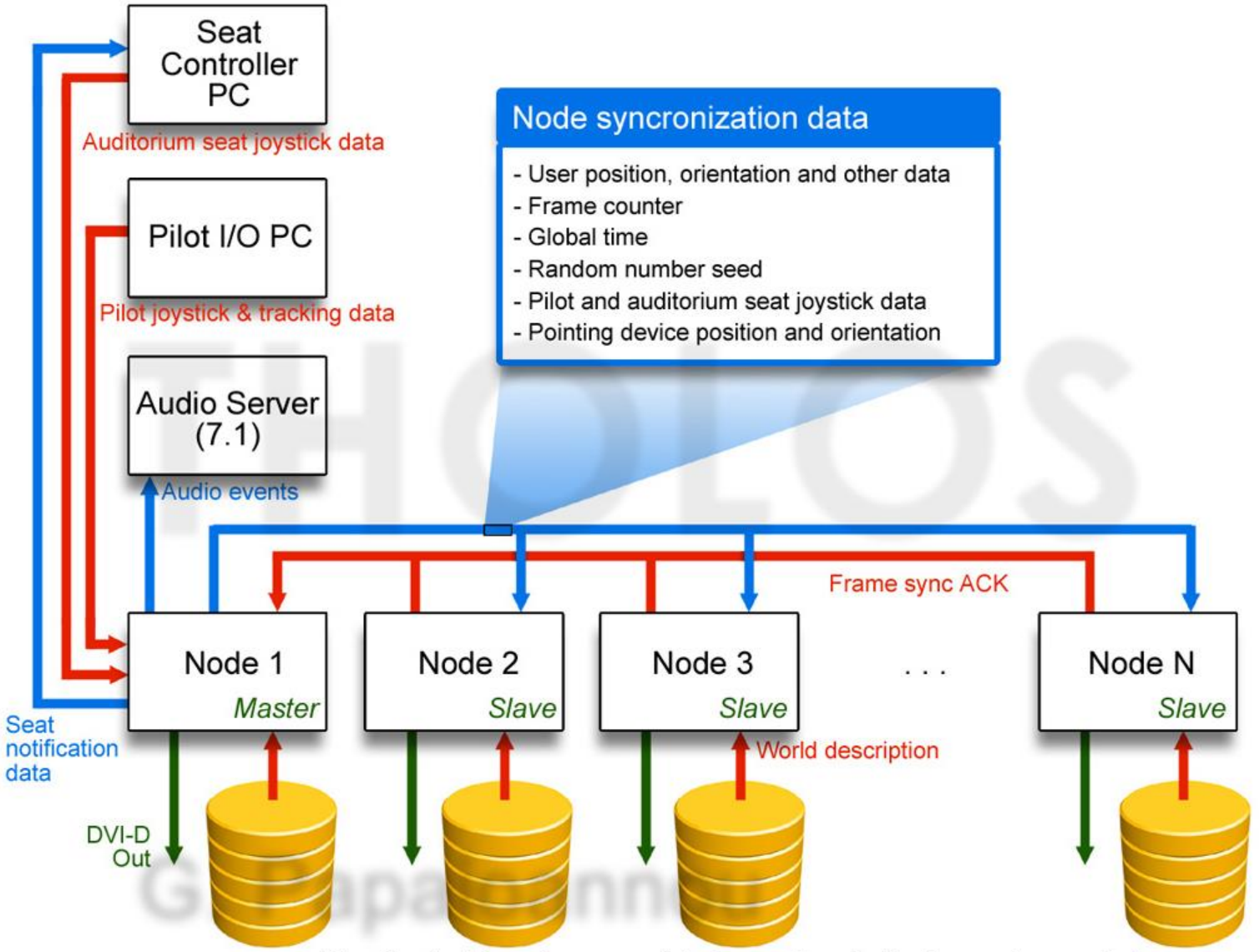
Tholos computing system

- In order to drive a multi-display environment, multiple graphics outputs need to be provided and synchronized to generate partial views of the same panorama (12 in our case).
- Due to the high amount of rendered and simulation data, the corresponding processes that drive each display output need to run in parallel.
- The obvious viable solution for satisfying the rendering demands of such a display system is a virtual reality cluster.
- For the Tholos VR system, an asymmetric master/slave cluster configuration was designed and implemented, which provides a highly parallel execution and has almost zero scaling overhead (frame lag) when adding new nodes.

Tholos computing system

Multi-headed display cluster stress test at the AUEB computer labs (32 nodes simultaneously deployed).

Example: **The "Tholos" Virtual Reality Dome Theater**



Grid Computing Systems

- A characteristic feature of cluster computing is its homogeneity.

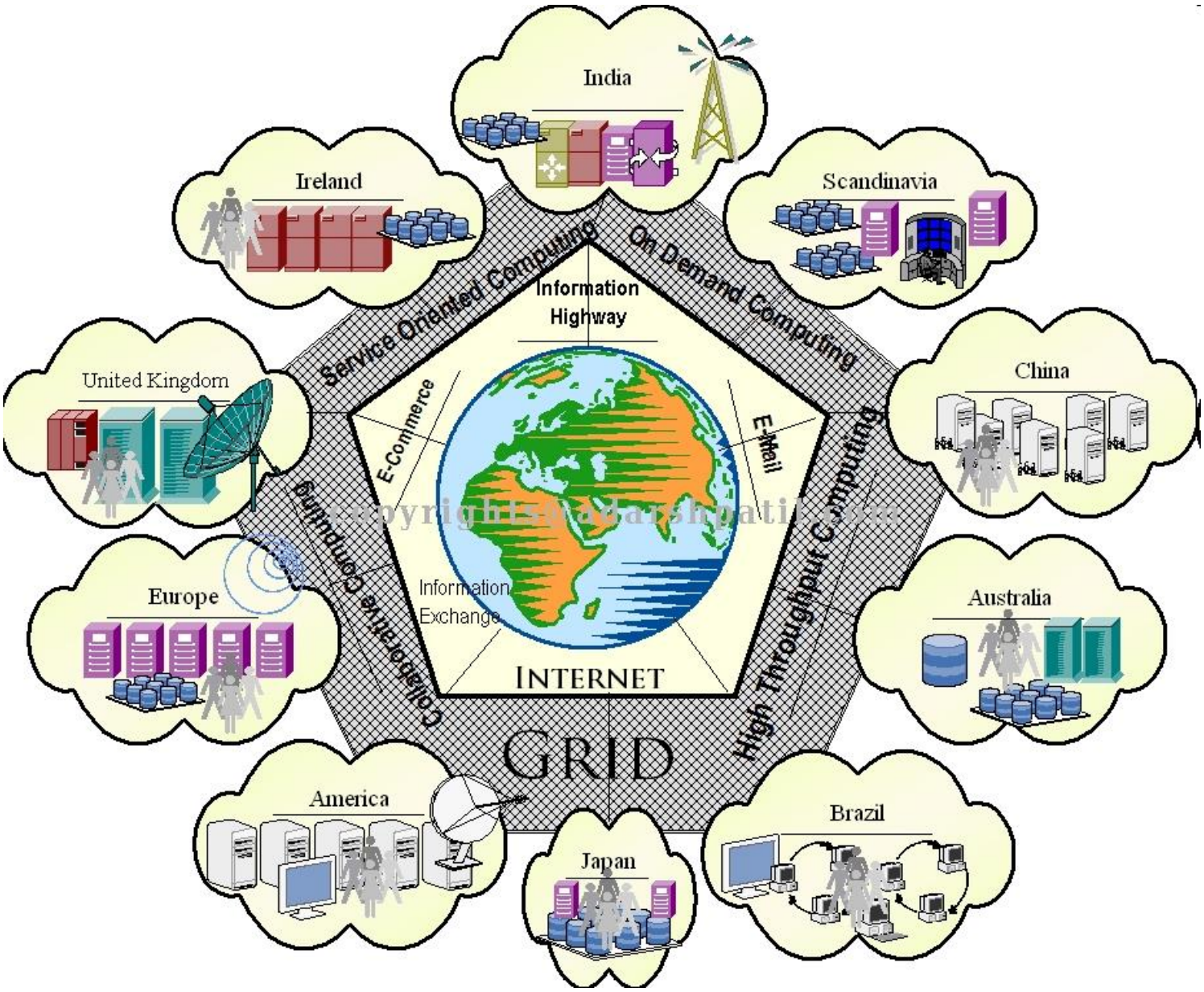
- In most cases, the computers in a cluster are largely the same, they all have the same operating system, and are all connected through the same network.

- In contrast, grid computing systems have a high degree of heterogeneity: no assumptions are made concerning hardware, operating systems, networks, administrative domains, security policies, etc.

Grid computing

- A key issue in a grid computing system is that resources from different organizations are brought together to allow the collaboration of a group of people or institutions.
- Such a collaboration is realized in the form of a virtual organization.
- The people belonging to the same virtual organization have access rights to the resources that are provided to that organization.
- Typically, resources consist of compute servers (including supercomputers, possibly implemented as cluster computers), storage facilities, and databases.

Grid Computing*



*From Adarsh Patil

Middleware for grid computing

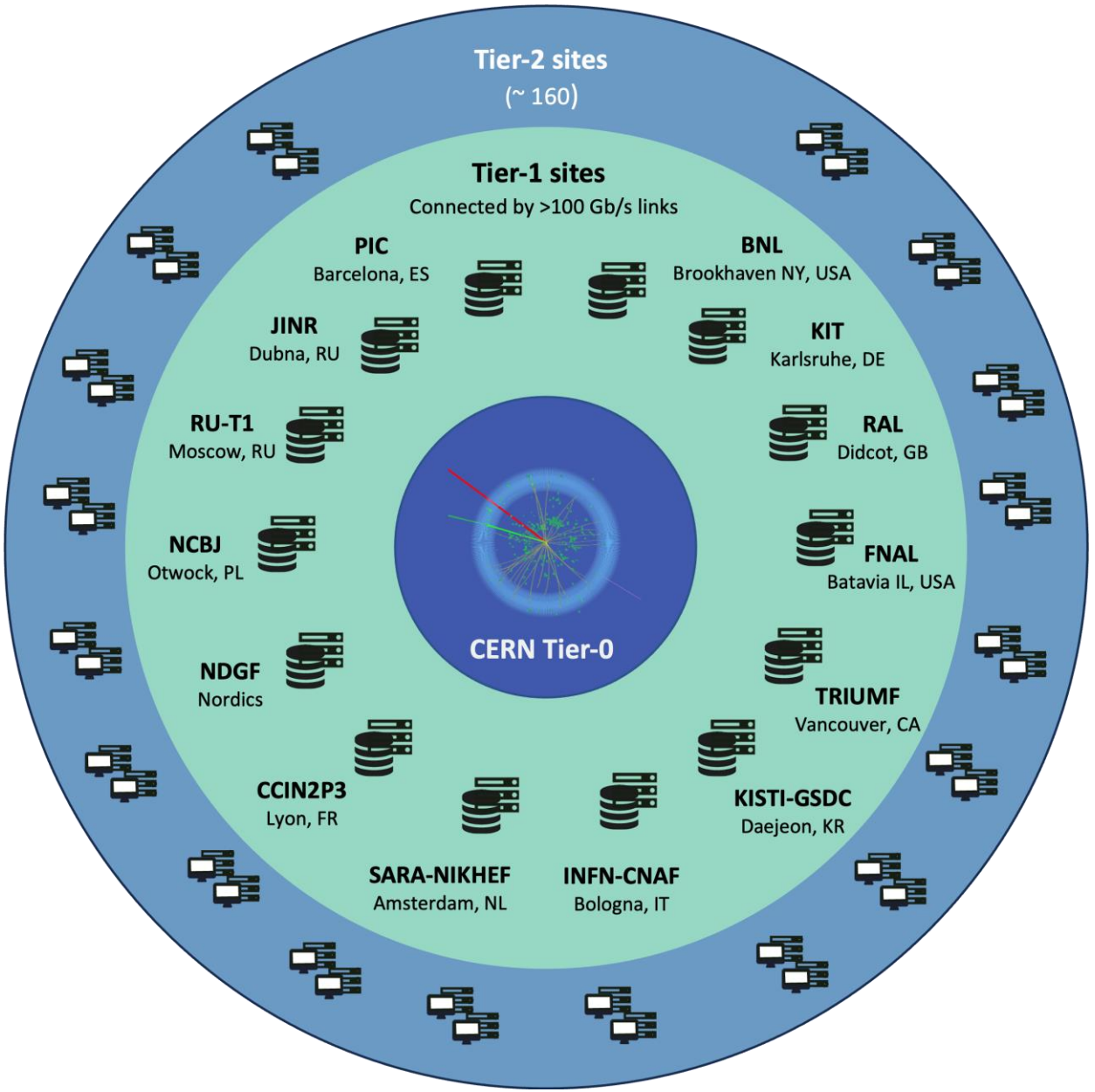
- At the core of Grid computing is the middleware: it consists of a series of software components that realize the interface between the distributed resources on one side, and the applications on the other side.

- These components include:
  - resource discovery,
  - job scheduling,
  - authentication and
  - authorization,
  - data logging,
  - data transfer and
  - replication etc.

Worldwide LHC Computing Grid (WLCG)

- Provides global computing resources for the storage, distribution and analysis of the data generated by the LHC (Large Hadron Collider).
- WLCG combines about 1.4 million computer cores and 1.5 exabytes of storage from over 170 sites in 42 countries.
- This massive distributed computing infrastructure provides more than 12000 physicists around the world with near real-time access to LHC data, and the power to process it.
- It runs over 2 million tasks per day and, at the end of the LHC's LS2, global transfer rates exceeded 260 GB/s.
- These numbers will increase as time goes on and as computing resources and new technologies become ever more available across the world.
- CERN provides about 20% of the resources of WLCG.

Structure of WLCG – 4 TIERS



WLCG Tiers

- Tier-0
  - This is the CERN Data Centre, which is located in Geneva, Switzerland. All data from the LHC passes through the central CERN hub, but CERN only provides around 20% of the total compute capacity.
- Tier 1
  - These are fourteen large computer centres with sufficient storage capacity and with round-the-clock support for the Grid. They are responsible for the safe-keeping of a proportional share of raw and reconstructed data, large-scale reprocessing and safe-keeping of corresponding output, distribution of data to Tier 2s and safe-keeping of a share of simulated data produced at these Tier 2s.
- Tier 2
  - The Tier 2s are typically universities and other scientific institutes, which can store sufficient data and provide adequate computing power for specific analysis tasks. Around 160 such centers.
- Tier 3
  - Individual scientists will access these facilities through local (also sometimes referred to as Tier 3) computing resources, which can consist of local clusters in a University Department or even just an individual PC.

Middleware products for WLCG

- EGI (European Grid Infrastructure) and partners rely on the Unified Middleware Distribution (UMD) available here: http://repository.egi.eu/download/
- NDGF (Nordic Data Grid Facility) relies on Advanced Resource Connector (ARC) middleware that is available through the EGI UMD and the NorduGrid site: http://www.nordugrid.org/
- OSG (Open Science Grid) relies on the OSG software distribution: https://opensciencegrid.org/docs/
- For WLCG as a whole there are further requirements and procedures: https://twiki.cern.ch/twiki/bin/view/LCG/WLCGOperationsWeb

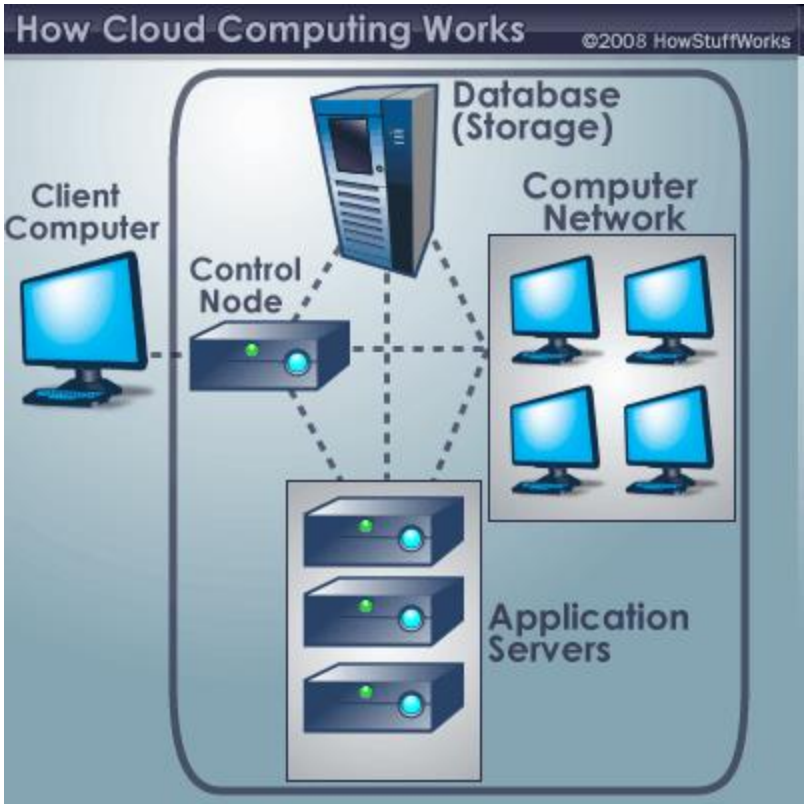
Cloud Computing

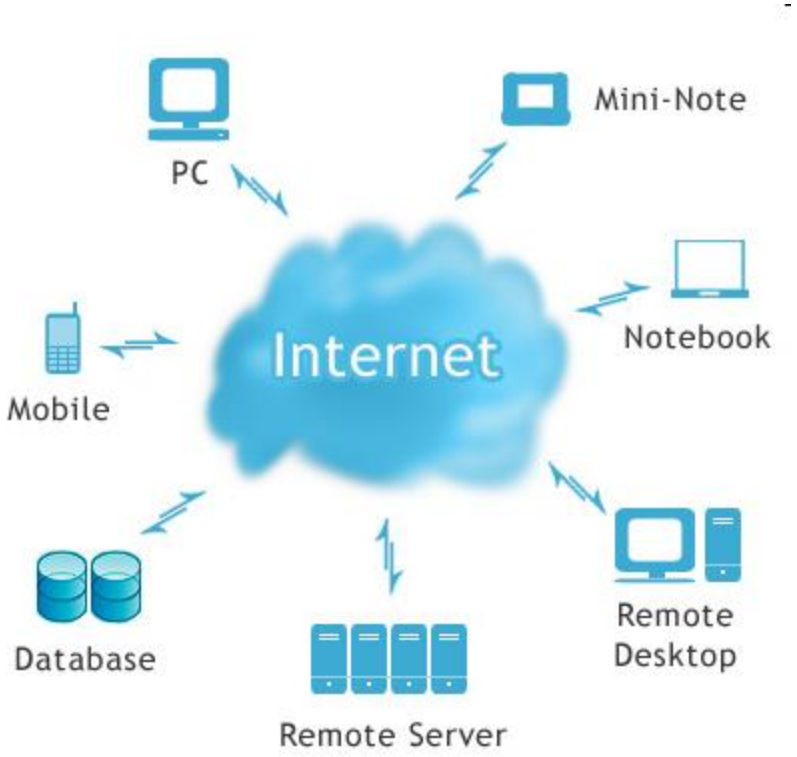
- Cloud computing is Internet- ("cloud-") based development and use of computer technology ("computing").

- In concept, it is a paradigm shift whereby details are abstracted from the users who no longer need knowledge of, expertise in, or control over the technology infrastructure "in the cloud" that supports them.

- *It typically involves the provision of dynamically scalable and often virtualized resources as a service over the Internet.*
  - *AWS, Microsoft Azure Cloud, Google Cloud Platform.*

Cloud computing





Cloud Computing

- Typical cloud computing providers deliver common business applications online which are accessed from a web browser, while the software and data are stored on the servers.
- These applications are broadly divided into the following categories:
  - Software as a Service (SaaS)
  - Utility Computing
  - Web Services
  - Platform as a Service (PaaS)
  - Managed Service Providers (MSP)
  - Service Commerce
  - Internet Integration

- The name cloud computing was inspired by the cloud symbol that is often used to represent the Internet in flow charts and diagrams.

Cloud computing: applications

- A cloud application leverages cloud computing in software architecture, often eliminating the need to install and run the application on the customer's own computer, thus alleviating the burden of software maintenance, ongoing operation, and support.
- For example:

- Peer-to-peer / volunteer computing (BOINC, Skype)
- Web applications (Webmail, Facebook, Twitter, YouTube)
- Security as a service (MessageLabs, Purewire, ScanSafe, Zscaler)
- Software as a service (A2Zapps.com, Google Apps, Salesforce,Learn.com, Zoho, BigGyan.com)
- Software plus services (Microsoft Online Services)
- Storage [Distributed]
- Content distribution (BitTorrent, Amazon CloudFront)
- Synchronisation (Dropbox, Live Mesh, SpiderOak, ZumoDrive)

Introduction

1.1 DEFINITION OF A DISTRIBUTED SYSTEM

1.2 GOALS

- 1.2.1 Making Resources Accessible
- 1.2.2 Distribution Transparency
- 1.2.3 Openness
- 1.2.4 Scalability
- 1.2.5 Pitfalls

1.3 TYPES OF DISTRIBUTED SYSTEMS

- 1.3.1 Distributed Computing Systems
- 1.3.2 Distributed Information Systems
- 1.3.3 Distributed Pervasive Systems

Distributed Information Systems

- Another important class of distributed systems is found in organizations that were confronted with a wealth of networked applications, but for which interoperability turned out to be a painful experience.

- Many of the existing middleware solutions are the result of working with an infrastructure in which it was easier to integrate applications into an enterprise-wide information system.

- We can distinguish several levels at which integration took place. In many cases, a networked application simply consisted of a server running that application (often including a database) and making it available to remote programs, called clients.

Distributed Information Systems

- As applications became more sophisticated and were gradually separated into independent components (notably distinguishing database components from processing components), it became clear that integration should also take place by letting applications communicate directly with each other.
- This has now led to a huge industry that concentrates on enterprise application integration (EAI).
- The vast amount of distributed systems in use today are forms of traditional information systems, that now integrate legacy systems.

Transaction Processing Systems

- A transaction is a collection of operations on the state of an object (database, object composition, etc.) that satisfies the following properties (ACID):

- Atomicity: All operations either succeed, or all of them fail. When the transaction fails, the state of the object will remain unaffected by the transaction.

- Consistency: A transaction establishes a valid state transition.

- Isolation: Concurrent transactions do not interfere with each other. It appears to each transaction T that other transactions occur either before T, or after T, but never both.

- Durability: After the execution of a transaction, its effects are made permanent.

Nested Transaction

- Nested transactions are important in distributed systems, for they provide a natural way of distributing a transaction across multiple machines.

- They follow a *logical* division of the work of the original transaction.
  - For example, a transaction for planning a trip by which three different flights need to be reserved can be logically split up into three subtransactions.
  - Each of these subtransactions can be managed separately and independent of the other two.

Nested Transaction

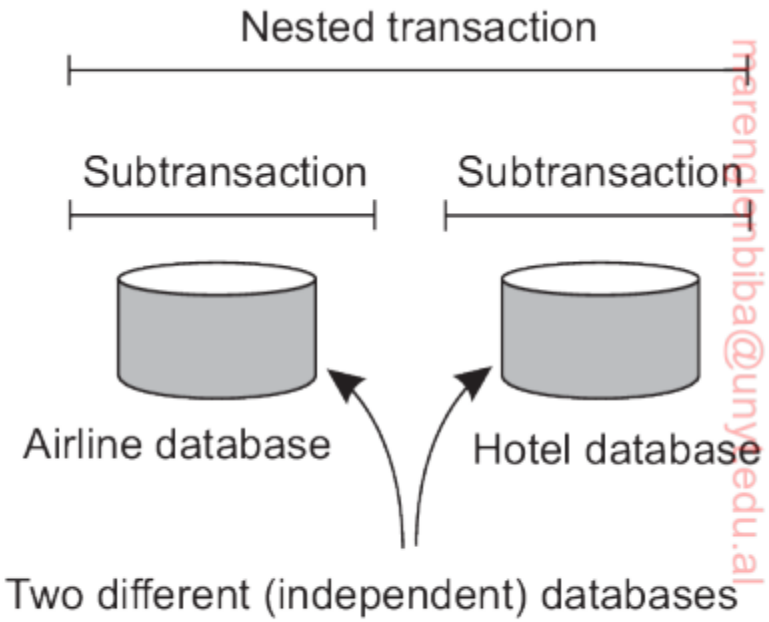


Figure 1.11: A nested transaction.

Transaction Processing Monitor

- In the early days of enterprise middleware systems, the component that handled distributed (or nested) transactions formed the core for integrating applications at the server or database level.

- This component was called a transaction processing monitor or TP monitor for short.
  - Its main task was to allow an application to access multiple server/databases by offering it a transactional programming model

Transaction Processing Monitor

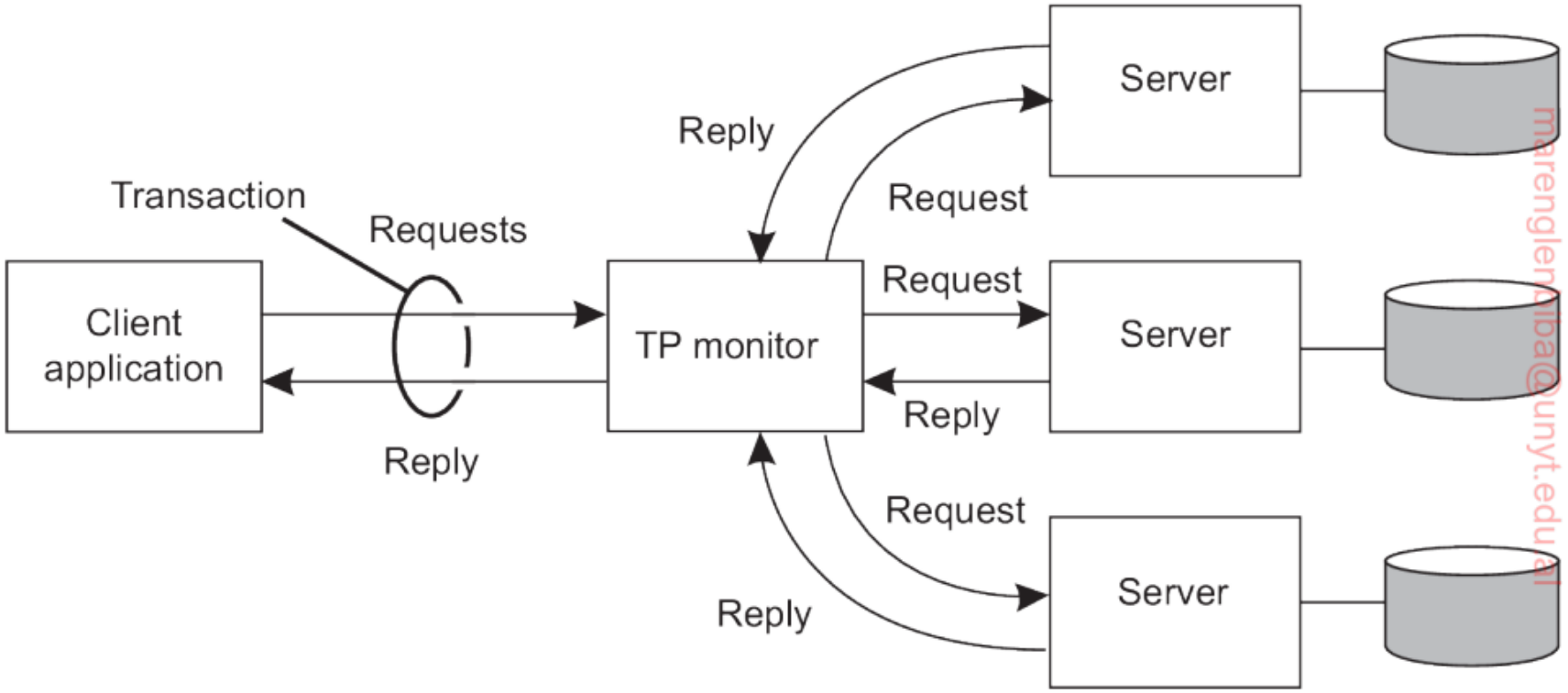


Figure 1.12: The role of a TP monitor in distributed systems.

Enterprise Application Integration

- The more applications became decoupled from the databases they were built upon, the more evident it became that facilities were needed to **integrate applications independent from their databases.**

- In particular, application components should be able to **communicate directly with each other** and not merely by means of the request/reply behavior that was supported by transaction processing systems.

- This need for inter-application communication led to many different **communication models**, which we will discuss in detail in this course (and for which reason we shall keep it brief for now).

- **The main idea was that existing applications could directly exchange information.**

Enterprise Application Integration

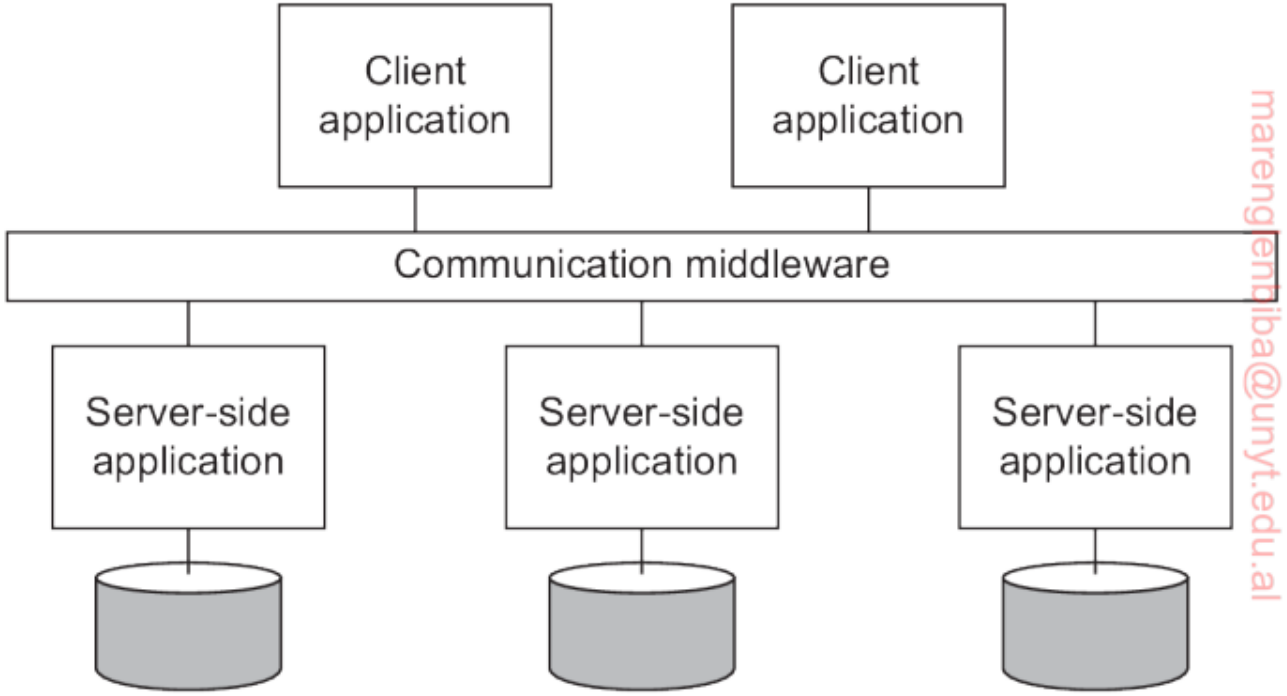


Figure 1.13: Middleware as a communication facilitator in enterprise application integration.

Communication Middleware

- Several types of communication middleware exist.
- With remote procedure calls (RPC), an application component can effectively send a request to another application component by doing a local procedure call, which results in the request being packaged as a message and sent to the callee.
  - Likewise, the result will be sent back and returned to the application as the result of the procedure call.

Communication Middleware

- As the popularity of object technology increased, techniques were developed to allow calls to remote objects, leading to what is known as remote method invocations (RMI).
  - An RMI is essentially the same as an RPC, except that it operates on objects instead of applications.

Introduction

1.1 DEFINITION OF A DISTRIBUTED SYSTEM

1.2 GOALS

- 1.2.1 Making Resources Accessible
- 1.2.2 Distribution Transparency
- 1.2.3 Openness
- 1.2.4 Scalability
- 1.2.5 Pitfalls

1.3 TYPES OF DISTRIBUTED SYSTEMS

- 1.3.1 Distributed Computing Systems
- 1.3.2 Distributed Information Systems
- 1.3.3 Distributed Pervasive Systems

Distributed Pervasive Systems

- The distributed systems we have been discussing so far are largely characterized by their stability: nodes are fixed and have a more or less permanent and high-quality connection to a network.

- To a certain extent, this stability has been realized through the various techniques that are discussed in this course and which aim at achieving distribution transparency.

- However, matters have become very different with the introduction of mobile and embedded computing devices. We are now confronted with distributed systems in which instability is the default behavior.

- The devices in these, what we refer to as distributed pervasive systems, are often characterized by being small, battery-powered, mobile, and having only a wireless connection, although not all these characteristics apply to all devices.

Distributed Pervasive Systems

Some requirements

- Contextual change: The system is part of an environment in which changes should be immediately accounted for.
- Ad hoc composition: Each node may be used in very different ways by different users.
- Sharing is the default: Nodes come and go, providing sharable services and information.

Home Systems

- An increasingly popular type of pervasive system, but which may perhaps be the least constrained, are systems built around home networks.

- These systems generally consist of one or more personal computers, but more importantly integrate typical consumer electronics such as TVs, audio and video equipment. Gaming devices, (smart) phones, PDAs, and other personal wearables into a single system.

- In addition, we can expect that all kinds of devices such as kitchen appliances, surveillance cameras, clocks, controllers for lighting, and so on, will all be hooked up into a single distributed system. Welcome to the future!

Electronic health systems

- Devices are physically close to a person:
  - Where and how should monitored data be stored?
  - How can we prevent loss of crucial data?
  - What is needed to generate and propagate alerts?
  - How can security be enforced?
  - How can physicians provide online feedback?

Electronic health systems

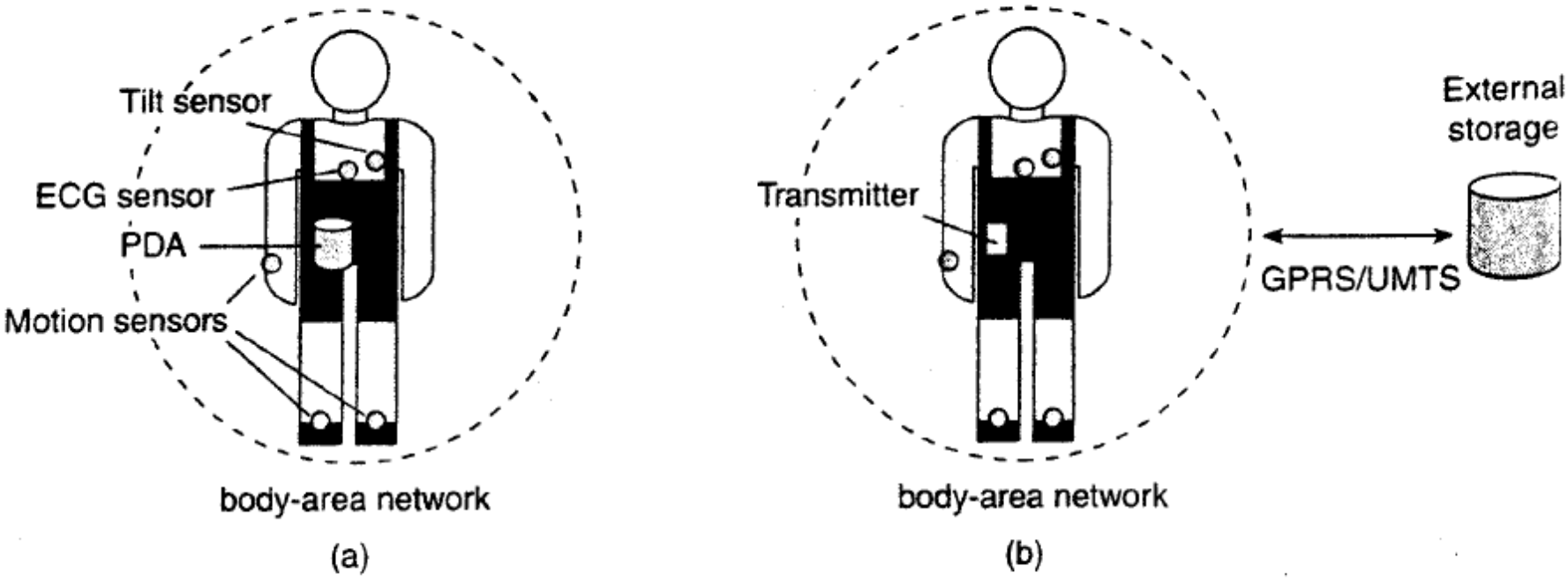


Figure 1-12. Monitoring a person in a pervasive electronic health care system, using (a) a local hub or (b) a continuous wireless connection.

Mobile computing systems

- There are several issues that set mobile computing aside to pervasive systems in general.
- First, the devices that form part of a (distributed) mobile system may vary widely. Typically, mobile computing is now done with devices such as smartphones and tablet computers. However, completely different types of devices are now using the Internet Protocol (IP) to communicate, placing mobile computing in a different perspective. Such devices include remote controls, pagers, active badges, car equipment, various GPS-enabled devices, and so on. A characteristic feature of all these devices is that they use wireless communication.
- Second, in mobile computing the location of a device is assumed to change over time. A changing location has its effects on many issues. For example, if the location of a device changes regularly, so will perhaps the services that are locally available.

MANET

- Changing locations also has a profound effect on communication. To illustrate, consider a (wireless) mobile ad hoc network, generally abbreviated as a MANET.
- Suppose that two devices in a MANET have discovered each other in the sense that they know each other's network address.
- How do we route messages between the two?
- Static routes are generally not sustainable as nodes along the routing path can easily move out of their neighbor's range, invalidating the path.
- For large MANETs, using a priori set-up paths is not a viable option.
- What we are dealing with here are so-called **disruption-tolerant networks**: networks in which connectivity between two nodes can simply not be guaranteed. Getting a message from one node to another may then be problematic, to say the least.

Sensor Networks

- Another example of pervasive systems is sensor networks.
- These networks in many cases form part of the enabling technology for pervasiveness and we see that many solutions for sensor networks return in pervasive applications.
- What makes sensor networks interesting from a distributed system's perspective is that in virtually all cases they are used for processing information.
  - In this sense, they do more than just provide communication services. which is what traditional computer networks are all about.

Sensor Networks

Characteristics

The nodes to which sensors are attached are:

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

Sensor Networks as DSs

- The relation with distributed systems can be made clear by considering sensor networks as distributed databases.

- This view is quite common and easy to understand when realizing that many sensor networks are deployed for measurement and surveillance applications.

- In these cases, an operator would like to extract information from (a part of) the network by simply issuing queries such as "What is the traffic load on a certain highway?"

Sensor Networks (2)

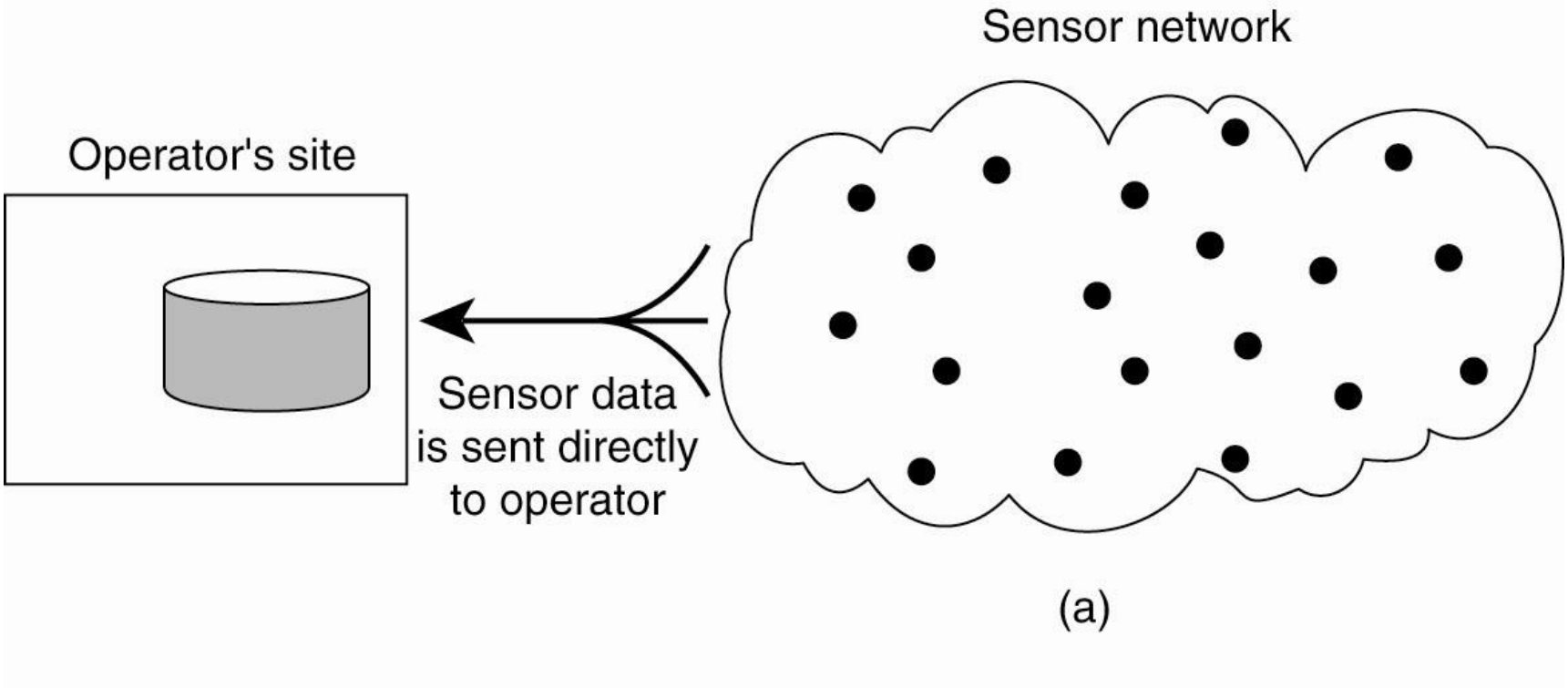


Figure 1-13. Organizing a sensor network database, while storing and processing data (a) only at the operator's site or …

Sensor Networks (3)

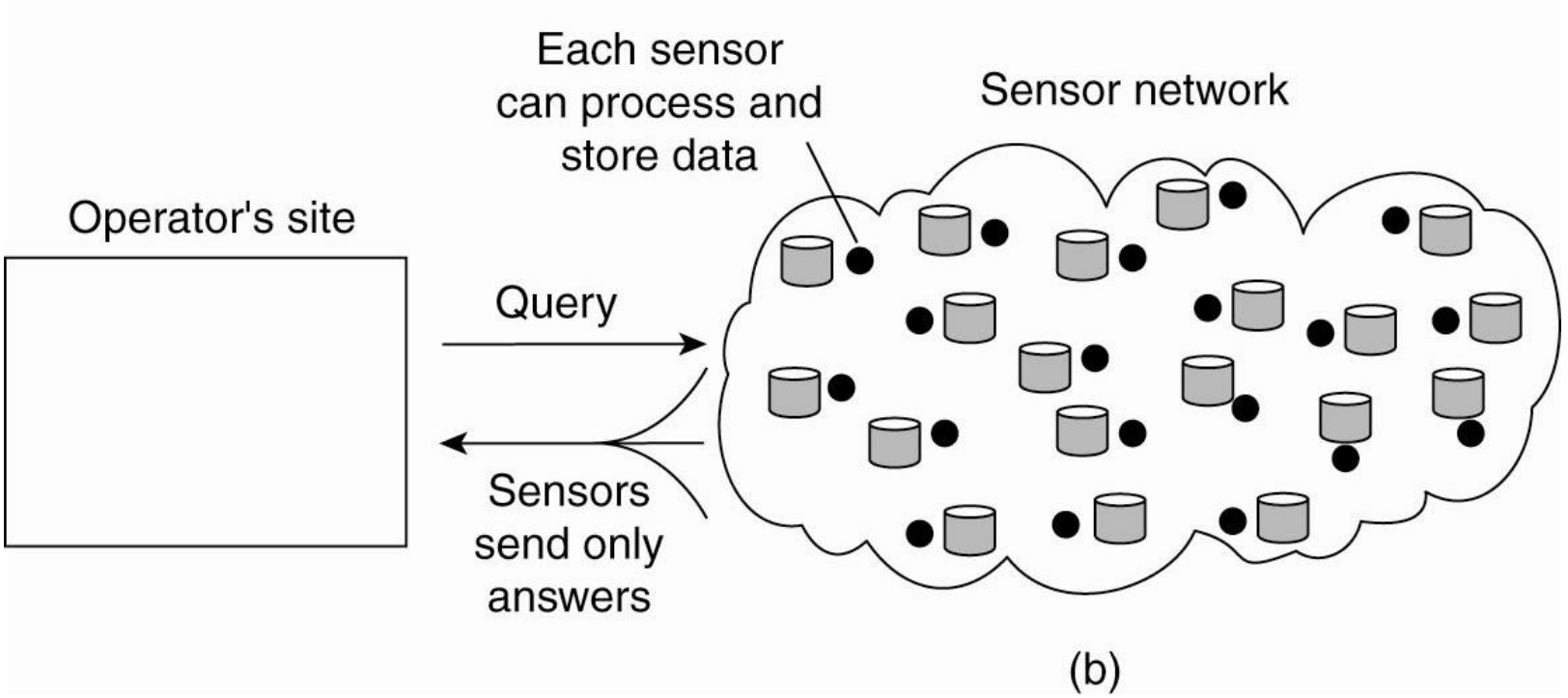


Figure 1-13. Organizing a sensor network database, while storing and processing data (b) only at the sensors.

End of Lesson 1

- Readings
  - Distributed Systems, Principles and Paradigms, Chapter 1.