

Josep Ll. L rida

# Distributed Computing

## Chapter 1. Introduction

# Distributed Computing

---

## Chapter 1 – Introduction

### 1.1. Computing Evolution

### 1.2. What is distributed Computing?

### 1.3. Distributed System Architectures

### 1.4. Applications of Distributed Systems

# Computing Evolution

## Technology Hallmarks

<b>First Generation</b>	(____-____) .....	Electrical component ... .. architecture
<b>Second Generation</b>	(____-____) .....	Electrical component
<b>Third Generation</b>	(____-____) .....	
<b>Fourth Generation</b>	(____-____) .....	
<b>Fifth Generation</b>	(____-____) .....	

# Computing Evolution

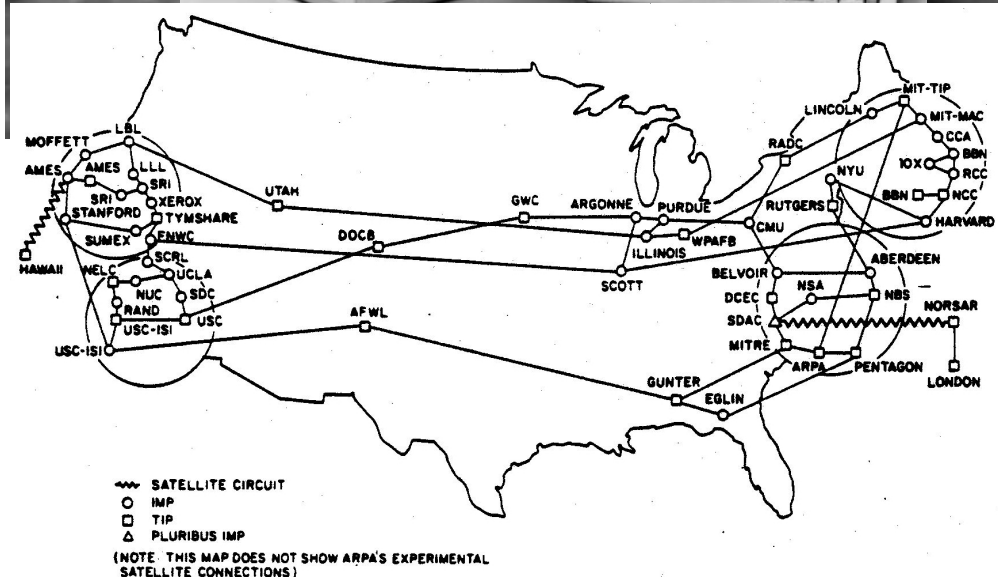


The **ENIAC** (1946), is considered to be the first general-purpose electronic computer.

**Mainframes** (1950s-1970s)

**Minicomputers** (1960s-1980s)

**Microcomputers** (1980s)

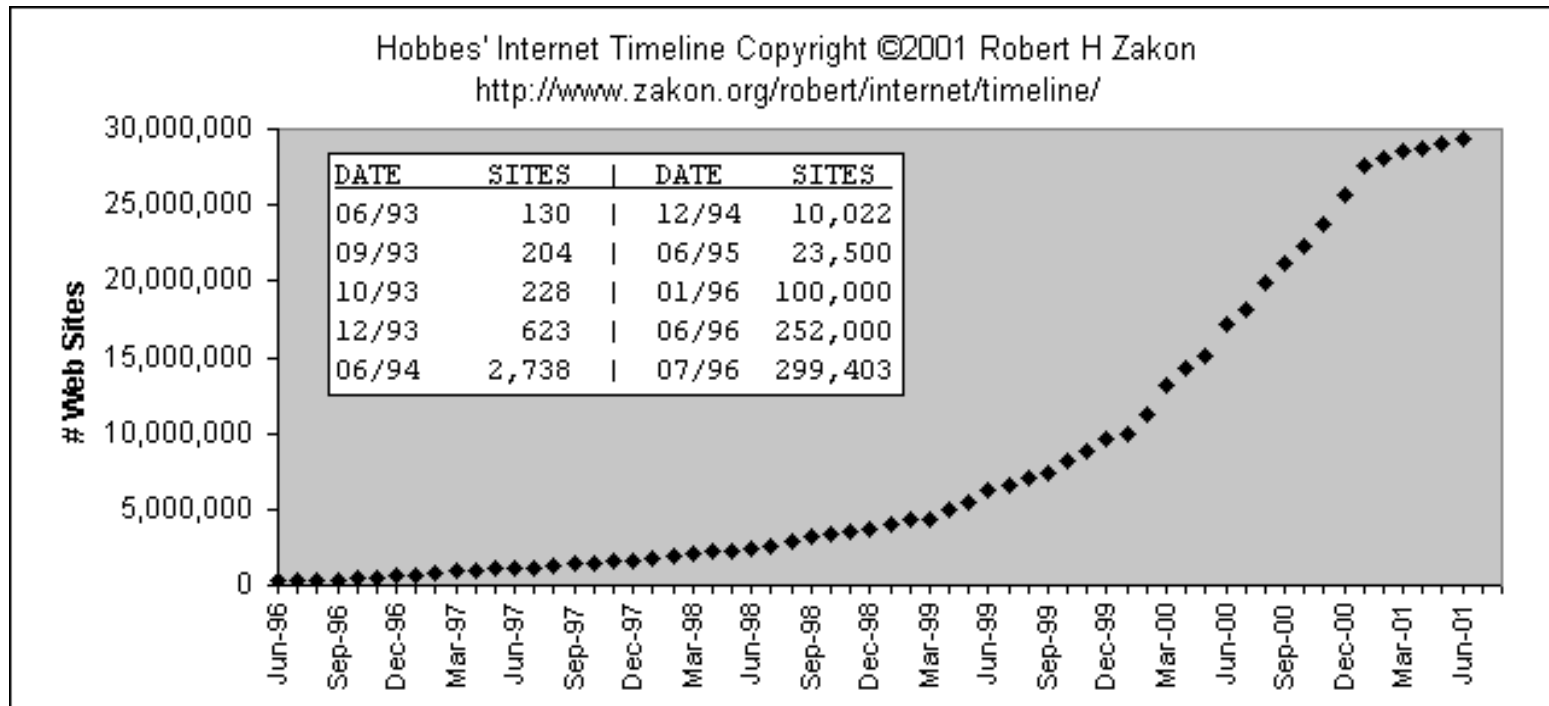


**ARPANET** (1969-1990)  
The Birth of the **INTERNET**

# Computing Evolution

## Internet evolution

Since the original proposal, the growth of the World-Wide Web has been extraordinary, and has expanded far beyond the research and academic community into all sectors world-wide, including commerce and private homes.



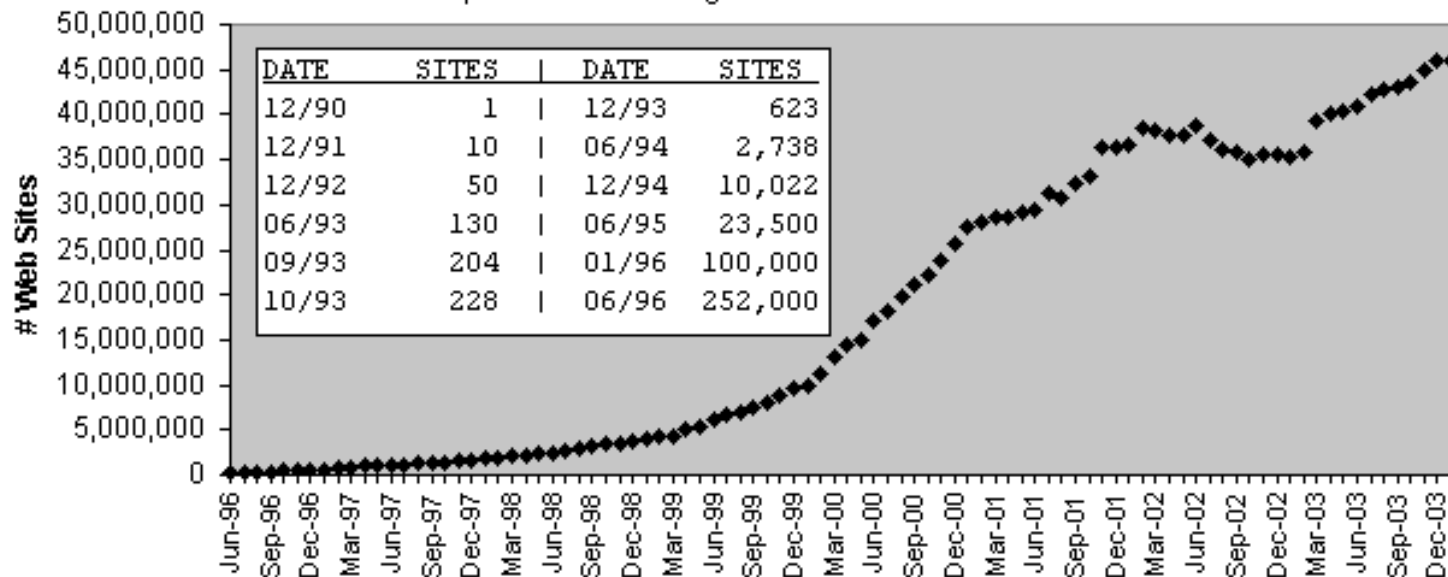
# Computing Evolution

## Internet evolution

Since the original proposal, the growth of the World-Wide Web has been extraordinary, and has expanded far beyond the research and academic community into all sectors world-wide, including commerce and private homes.

Hobbes' Internet Timeline Copyright ©2001 Robert H Zakon

Hobbes' Internet Timeline Copyright ©2004 Robert H Zakon  
<http://www.zakon.org/robert/internet/timeline/>



# Computing Evolution

## Internet evolution

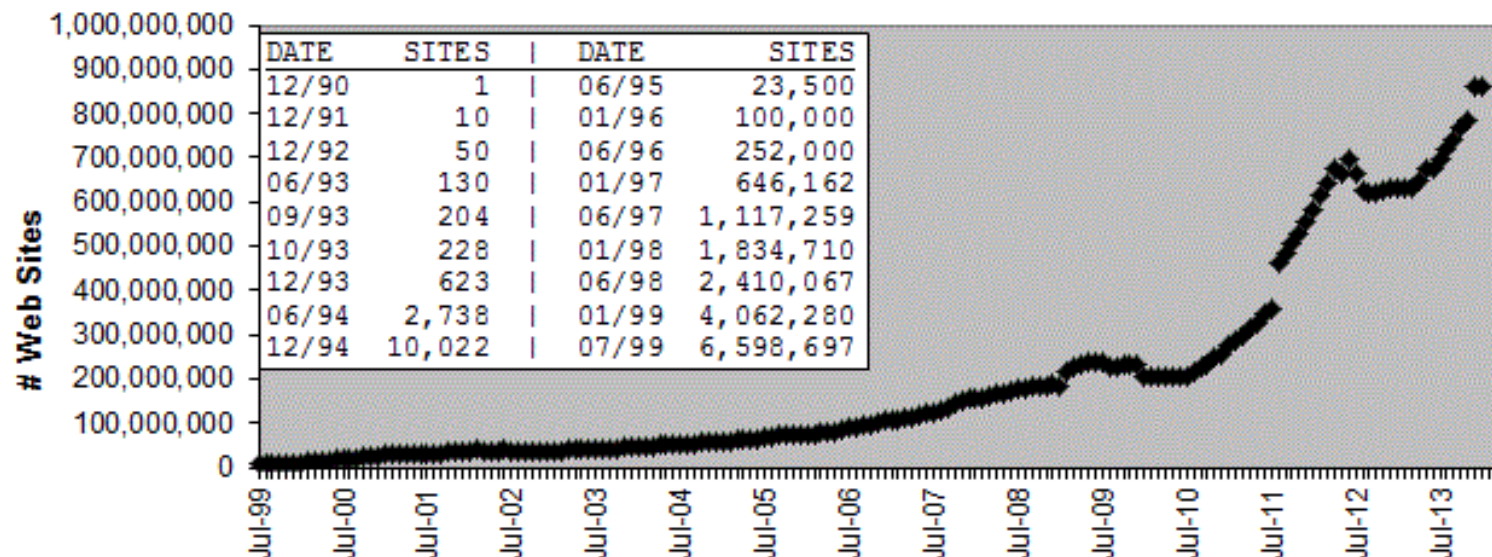
Since the original proposal, the growth of the World-Wide Web has been extraordinary, and has expanded far beyond the research and academic community into all sectors world-wide, including commerce and private homes.

Hobbes' Internet Timeline Copyright ©2001 Robert H Zakon

Hobbes' Internet Timeline Copyright ©2004 Robert H Zakon

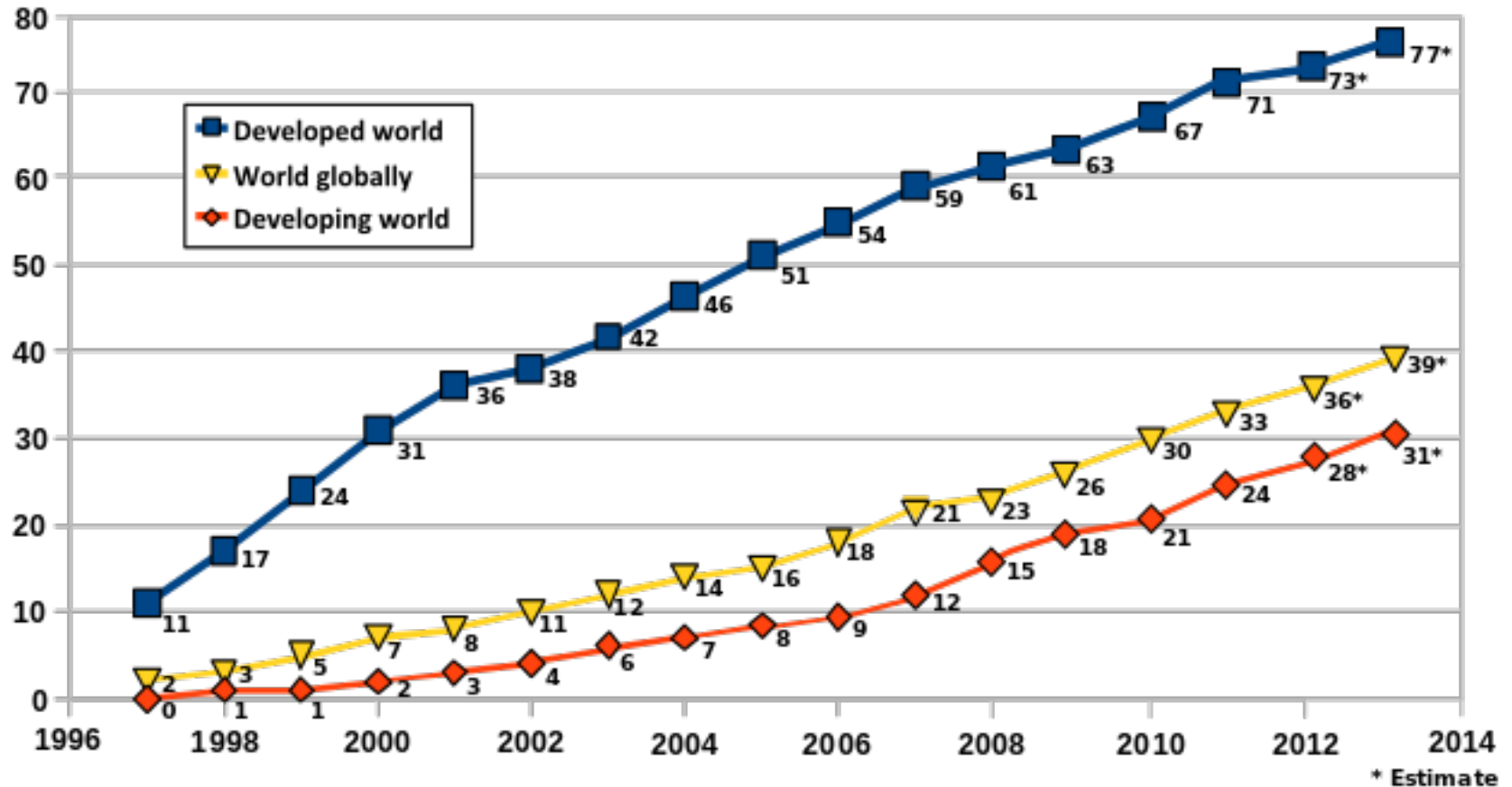
Hobbes' Internet Timeline Copyright ©2014 Robert H Zakon

<http://www.zakon.org/robert/internet/timeline/>



# Computing Evolution

## Internet evolution

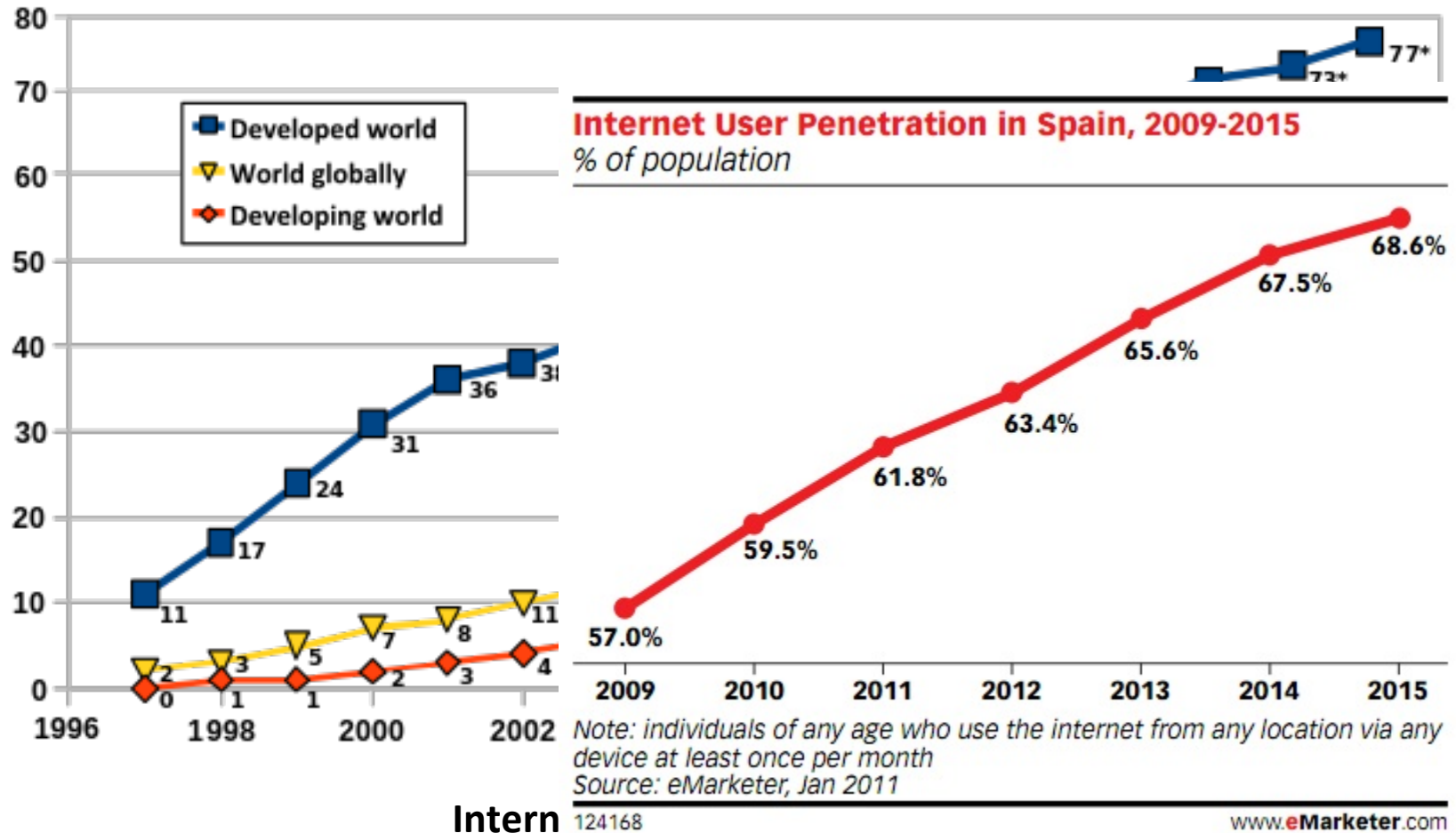


Internet user per 100 inhabitants



# Computing Evolution

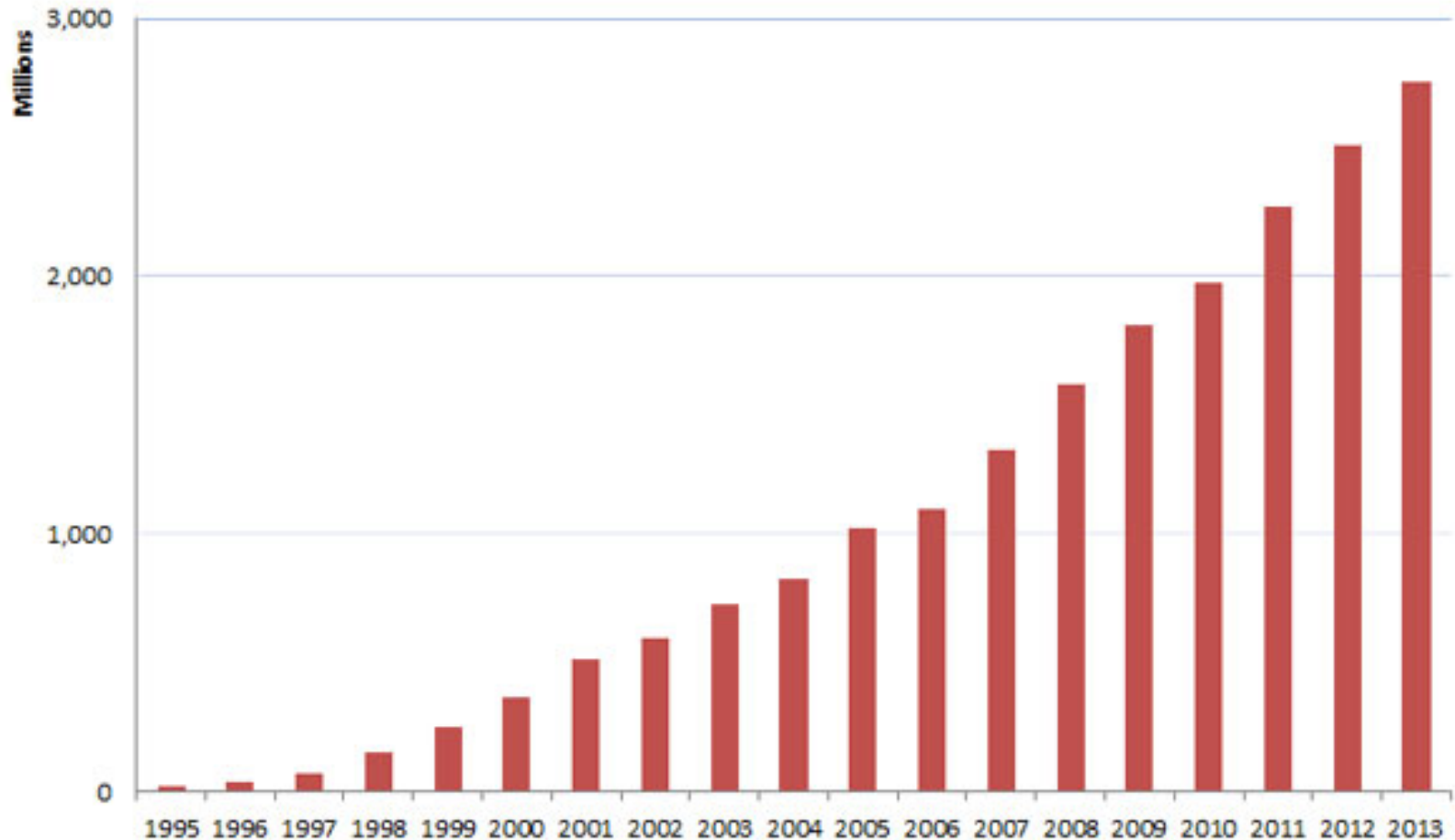
## Internet evolution



# Computing Evolution

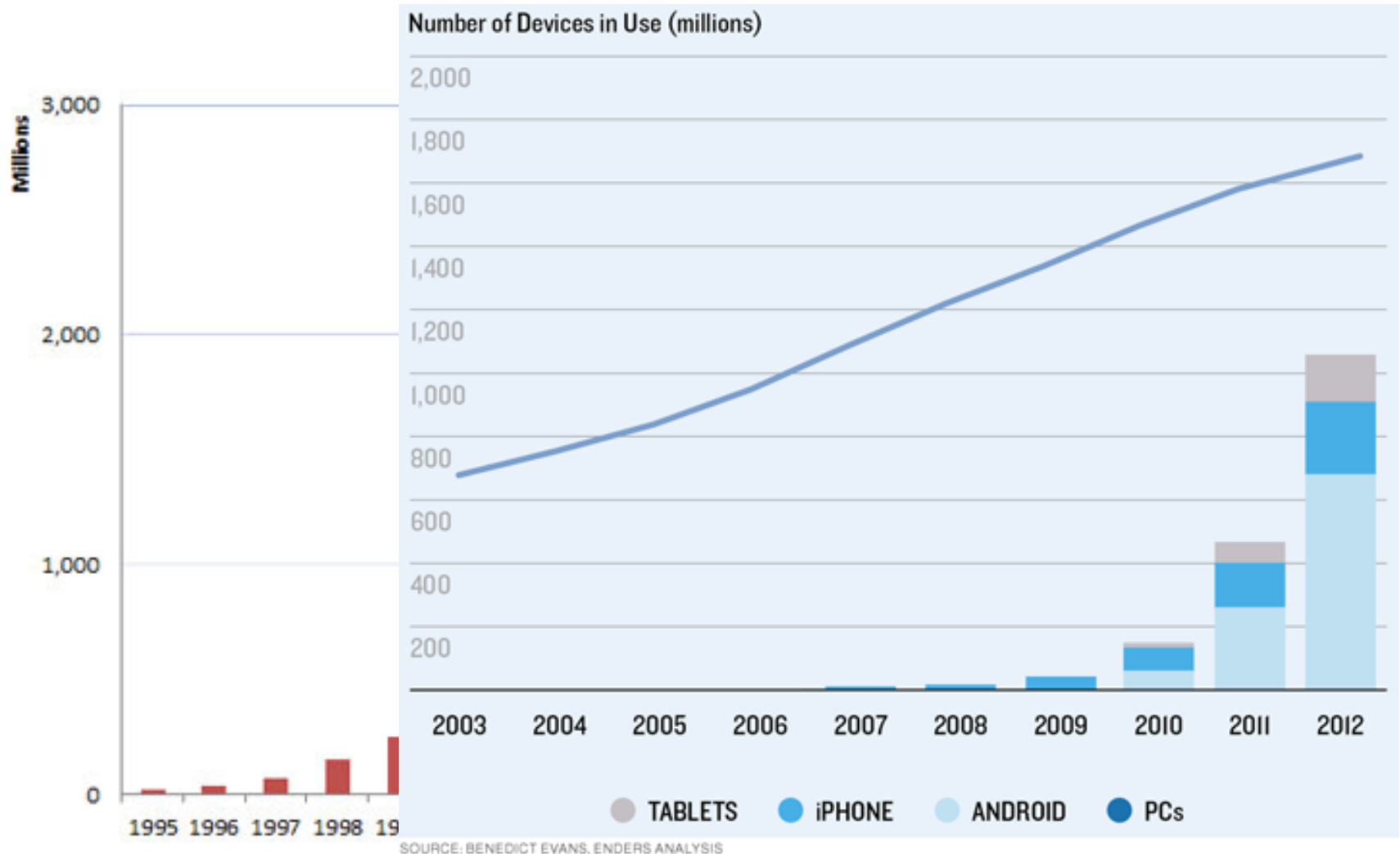
## Internet evolution

Internet users worldwide



# Computing Evolution

## Internet evolution



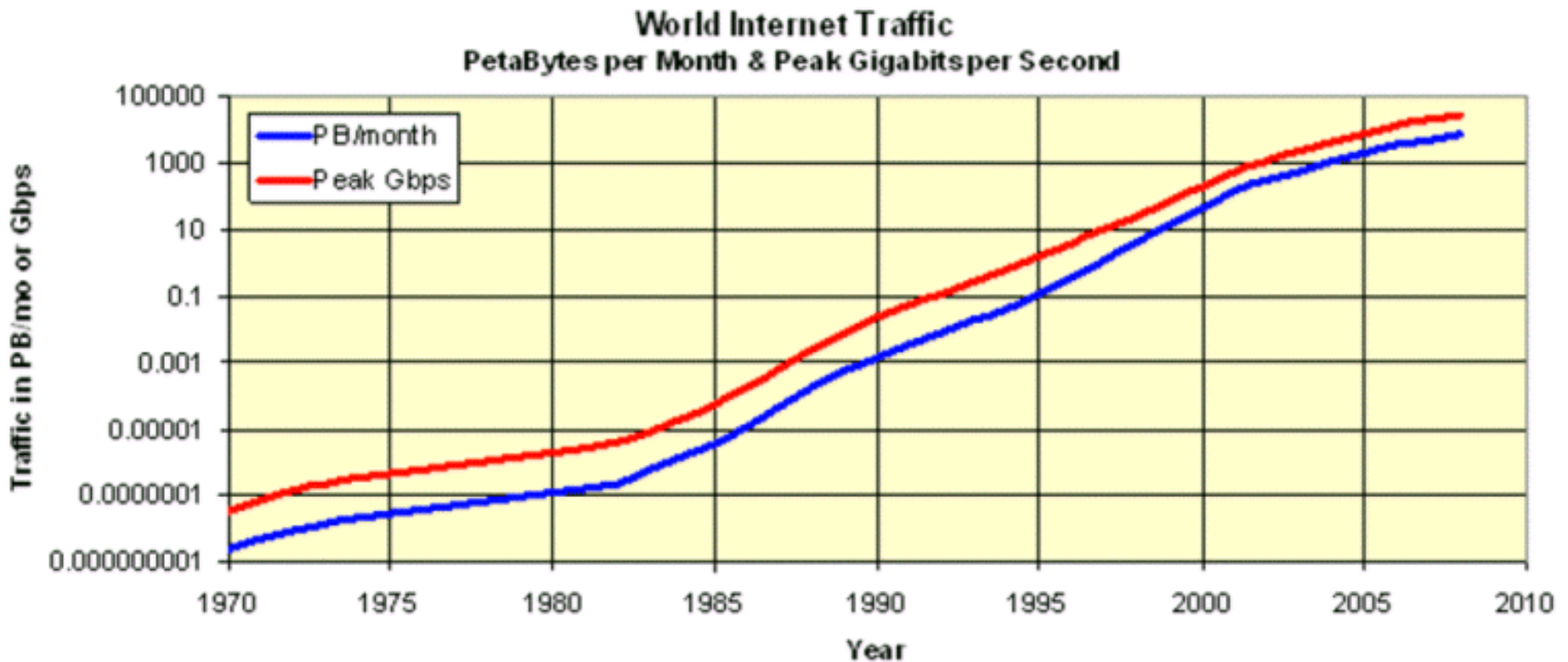
# Computing Evolution

## Internet evolution

Number of Devices in Use (millions)

2,000

1,800



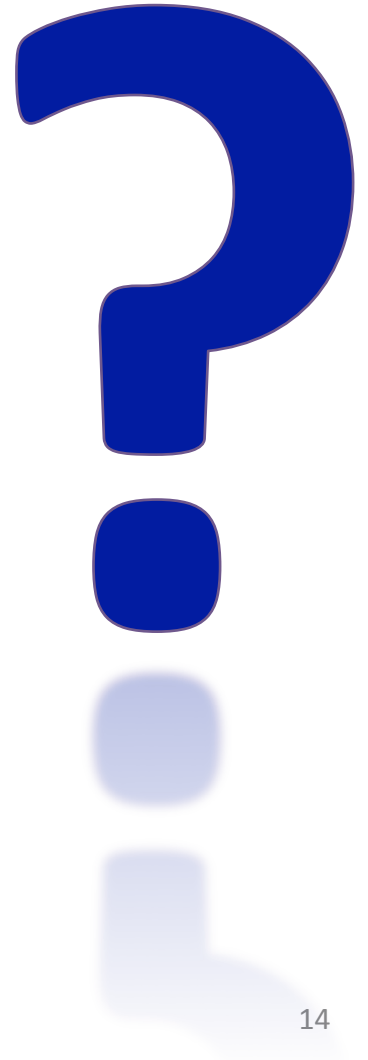
SOURCE: BENEDICT EVANS, ENDERS ANALYSIS

# Computing Evolution

---

## Recent Advances and New Challenges

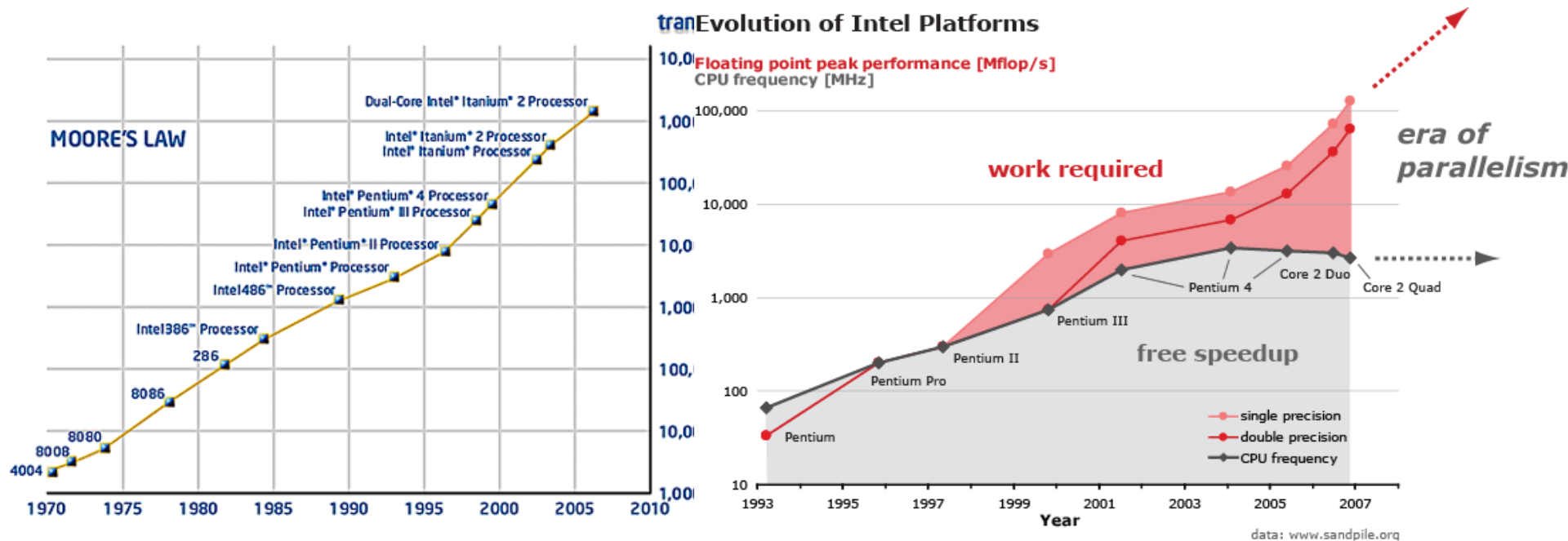
- Hardware and Computer structure
- Architecture
- Software
- Communications



# Computing Evolution

## Recent Advances and New Challenges

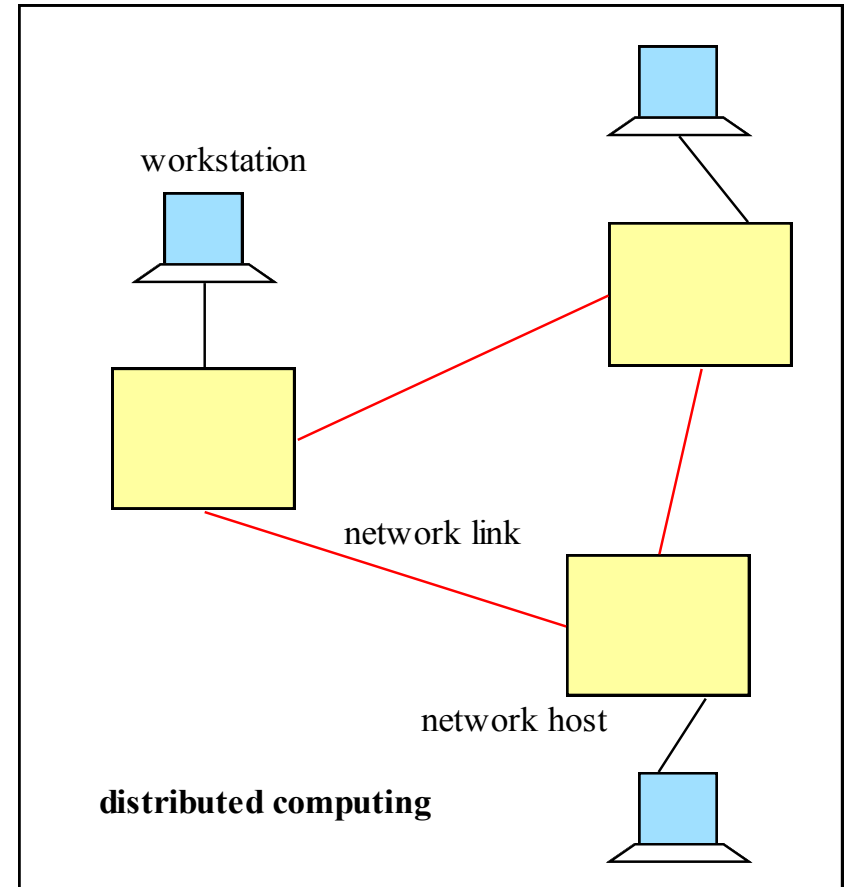
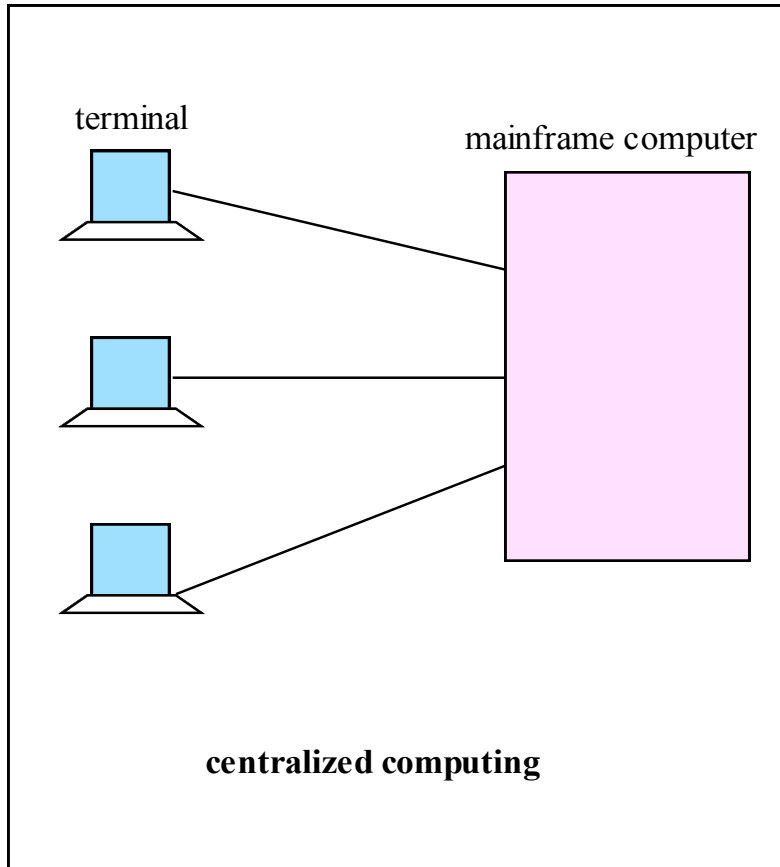
One major difference compared to previous decades is the end of the era of exponentially increasing processor speed.



The current trend is towards 4 (4)-core and many-4 (4) architectures as the main platforms for scalable data processing.

# Computing Evolution

## Centralized vs. Decentralized architectures



# Computing Evolution

## Activity

From \_\_\_\_\_(8) **purpose** to \_\_\_\_\_(7) **purpose** machines

From \_\_\_\_\_(9) to **personal computer** machines

From **isolated** to \_\_\_\_\_(9) networked machines

From the \_\_\_\_\_zed(11) to the \_\_\_\_\_ted(11) **computing**

From **client-server** to \_\_\_\_\_(13) applications

From **proprietary** to \_\_\_\_\_(11) protocols



# Computing Evolution

---

## Internet evolution (**killer apps**)

- 70s: FTP, Telnet, Usenet, e-mail
- 90s: Web browsers, online shopping, search engines
- Late 90s: Instant Messaging
- Early 2000: Napster, Grid computing
- Recently: social \_\_\_\_\_, \_\_\_\_\_ computing

# Computing Evolution

---

## Summary

- **Hardware and Computer structure**  
multi- and many- core machines, GPUs
- **Architecture**  
Cluster, Grid, P2P, Cloud (Virtualization)
- **Software**  
Search Engines, Business Intelligence,  
Language programming abstractions,  
Big Data processing
- **Communications**  
Network Infrastructures, Sensor of Networks, Internet of Things

# Distributed Computing

---

## Chapter 1 – Introduction

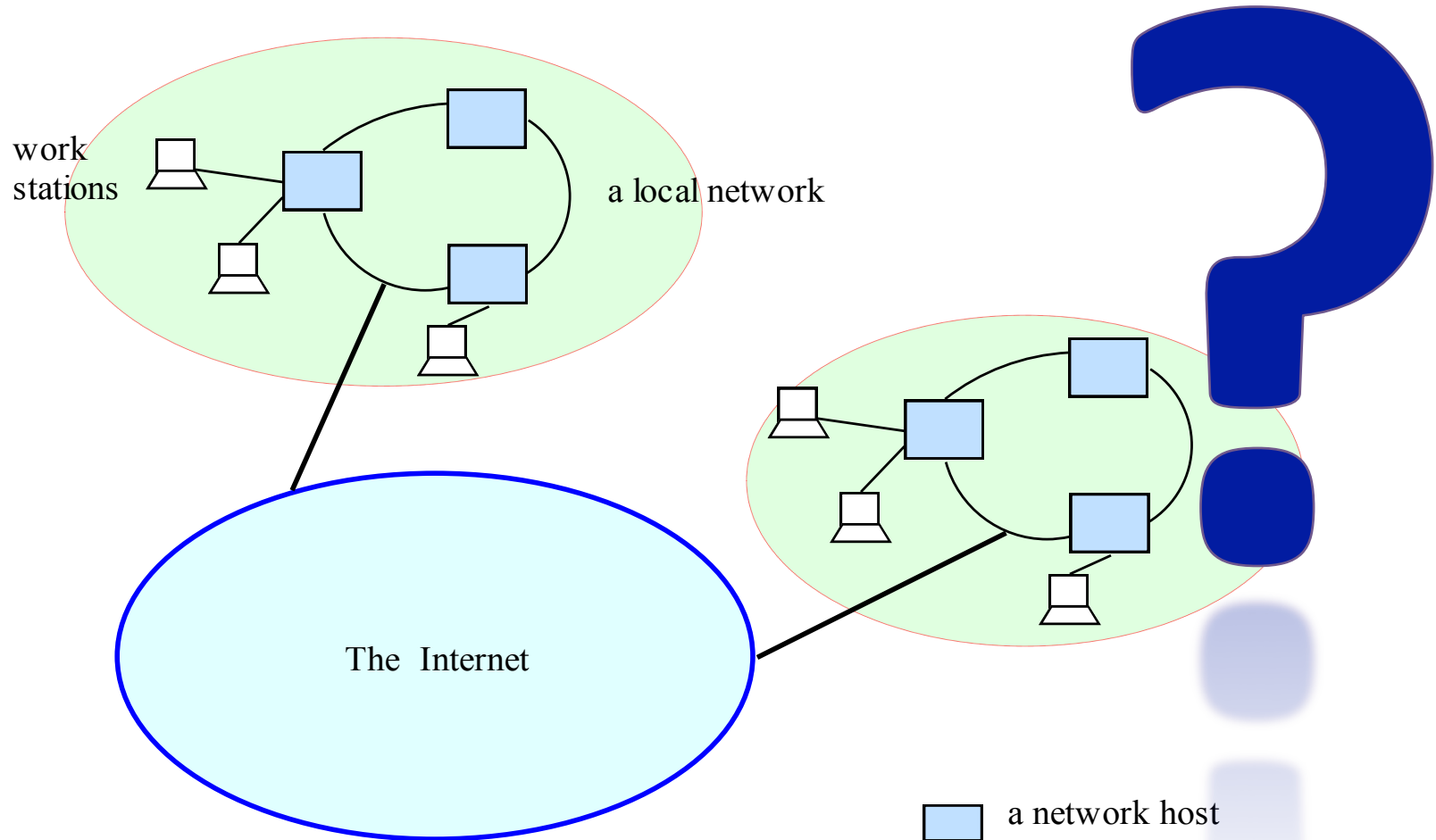
### 1.1. Computing Evolution

### 1.2. What is distributed Computing?

### 1.3. Distributed System Architectures

### 1.4. Applications of Distributed Systems

# What is Distributed Computing?



# What is Distributed Computing?

## Some Definitions in Literature

- Network-based computing
- a collection of **independent computers**, interconnected via a network, able to **collaborate** in order to perform a task.
- Originally referred to computer networks where **individual computers** were **physically distributed** within some geographical area.
- a collection of **independent computers** that appears to its users as a **single coherent system**
- Autonomous processes that run on the same physical computer and **interact with each other by message passing**

**You know when you have one ...**

**... when the failure of a computer you've never heard of stops you from getting any work done  
(L.Lamport)**

# What is Distributed Computing?

## Definition

a collection of **independent computers**, interconnected via a network, able to **collaborate** in order to perform a task, appearing to its users as a **single coherent system**.







## Significant Properties

- Lack of world clock
- Independent failure of components
- Autonomous entities with its own memory
- Communication by message passing
- The structure is unknown in advance
- Heterogeneity

# What is Distributed Computing?

## Activity

Considering distributed computing as defined in this chapter. For each of the following activities, determine and explain whether it is an example of distributed computing:

-  • Using Excel on a personal computer.
-  • Browse the web.
-  • Using instant messaging.
-  • Compiling and testing a program written in Cobol on a departmental machine connected to thin clients.
-  • Using email on the computer of a department to send a message to yourself.
-  • Using a software to download music.

# What is Distributed Computing?

## Parallel vs. Distributed Computing



- The processors in a typical distributed system run concurrently in parallel.
- Parallel computing can be particular \_\_\_\_\_(7) coupled form of distributed computing
- As a rule of thumb, High-performance parallel computation in a \_\_\_\_\_(6)-memory multiprocessor uses \_\_\_\_\_(8) algorithms while the coordination of a large-scale \_\_\_\_\_(11) system uses distributed algorithms.



# What is Distributed Computing?

## Activity

### Concepts and Context.

The distributed computing area involves a lot of new concepts that describe the new artifacts necessary to cover this topic, and also terminology with specific meaning in this area. We need to manage this concepts and use correctly in the corresponding context, and particularly in English.

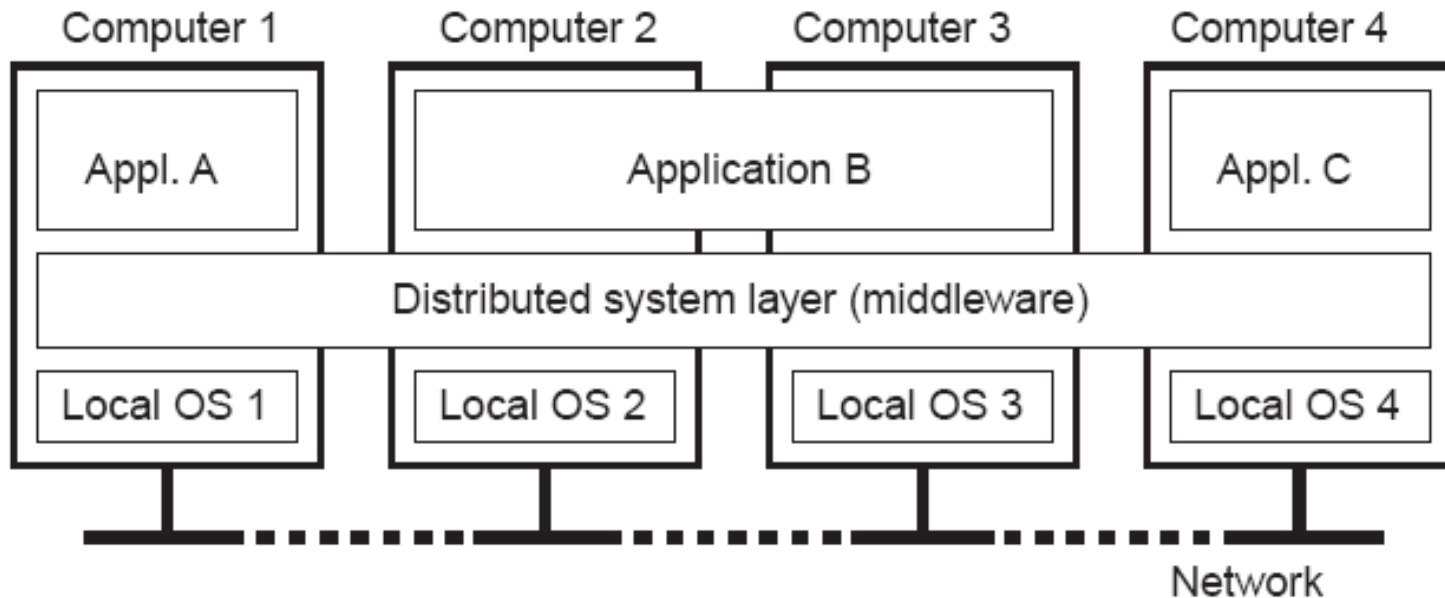
Define the following concepts and use them within the context of distributed computing. For each concept try to put a sentence that use it correctly in the distributed computing context.

### Concepts:

- Reliability, abstraction, affordability, fault tolerance, openness, replication, loosely coupled, resource sharing, interoperability, scalability, relocation, Middleware, security, failure, availability, transparency, portability, tightly coupled, latency, complexity, portability

# What is Distributed Computing?

A major goal is to appear as a \_\_\_\_\_(6) system.



- Independent hardware installations
- Uniform software layer (**middleware**)

**Note:** the middleware layer extends over multiple machines

# What is Distributed Computing?

**Middleware** is a software layer that provides a \_\_\_\_\_(11) **abstraction**, masking the heterogeneity of the underlying \_\_\_\_\_(8), \_\_\_\_\_(8), operating \_\_\_\_\_(7) and \_\_\_\_\_(11) languages. (It supports transparent access to remote services)

The middleware provides a uniform computational model for use by programmers servers and distributed applications.

Services that can be regarded as middleware include:

- CORBA (Common Object Request Broker Architecture)
- DCOM (Distributed Component Object Management)
- RPC (Remote Procedure Call)
- RMI (Remote Method Invocation)
- SOAP (Simple Object Access Protocol)

# What is Distributed Computing?

---

## Strenghts and Weaknesses

**What about?**

**Complexity, availability, failures,  
resource sharing, security,  
bandwidth, scalability, latency,  
reliability, affordability**



# What is Distributed Computing?

---

## Strenghts and Weaknesses

Some of the reasons for the popularity of distributed computing:

- **Affordability** of computers and **Availability** of network access
- **Resource sharing and replication**
- **Scalability**
- **Reliability**

# What is Distributed Computing?

## Strenghts and Weaknesses

The disadvantages of distributed computing:

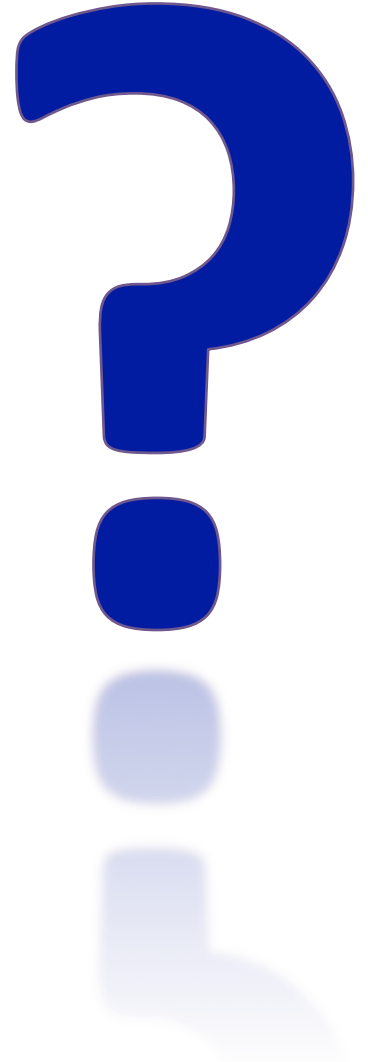
- **Software complexity:** Use of complex middlewares, programming paradigms, etc.
- **Multiple Points of Failures:** the failure of one or more participating computers, or one or more network links, can spell trouble.
- **Network Concerns:** saturation, latency, etc.
- **Security Concerns:** In a distributed system, there are more opportunities for unauthorized attacks.

# What is Distributed Computing?

---

## Goals and Challenges

- Accessibility
- Transparency
- Openness
- Scalability



# What is Distributed Computing?

## Accessability

- To access and share resources in a controlled and efficient way.
  - Printers, computers, storage facilities, data, files, Web pages, and networks, ...
- Connecting users and resources also makes it easier to **collaborate** and **exchange** information.
  - Internet for exchanging files, mail, documents, audio, and video
- **Security** is becoming increasingly important
  - Little protection against eavesdropping or intrusion on communication
  - Tracking communication to build up a preference profile of a specific user



# What is Distributed Computing?

## Transparency

Transparency	Description
Access	
Location	
Migration	
Reallocation	
Replication	
Concurrency	
Failure	

**Note:** Distribution transparency may be set as a goal, but achieving it is very difficult.

# What is Distributed Computing?

## Transparency

Transparency	Description
<b>Access</b>	Hide differences in data representation and how a resource is accessed
<b>Location</b>	Hide where a resource is located
<b>Migration</b>	Hide that a resource may move to another location
<b>Reallocation</b>	Hide that a resource may be moved to another location while in use
<b>Replication</b>	Hide that a resource is replicated
<b>Concurrency</b>	Hide that a resource may be shared by several competitive users
<b>Failure</b>	Hide the failure and recovery of a resource

**Note:** Distribution transparency may be set as a goal, but achieving it is very difficult.

# What is Distributed Computing?

## Openness

**Goal:** Open distributed system -- able to interact with services from other open systems, irrespective of the underlying environment:

- System should conform to well-defined interfaces
- Systems should support portability of applications
- Systems should easily interoperate
- Achieving openness: At least make the distributed system independent from heterogeneity of the underlying environment:
  - Hardware
  - Platforms
  - Languages

# What is Distributed Computing?

## Scalability

Many developers of modern distributed system easily use the adjective “scalable” without making clear why their system actually scales.

**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.

# What is Distributed Computing?

## Scalability Problems @ size

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

### 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 **world clock** exists

# What is Distributed Computing?

## Scalability Problems @ geography

- **Synchronous** communication
  - The **client** blocks until a reply is sent back.
- WANs is unreliable and **Point-to-point**
- LANs provide reliable communication facilities based on **broadcasting**.
- Geographical scalability is strongly related to the problems of centralized solutions that hinder size scalability.
  - In addition, centralized components now lead to a waste of network resources.

# What is Distributed Computing?

## Scalability Problems @ administration

- It is a difficult and in many cases open question
  - Conflicting policies with respect to **resource usage** ( and payment), **management**, and **security** must be considered.

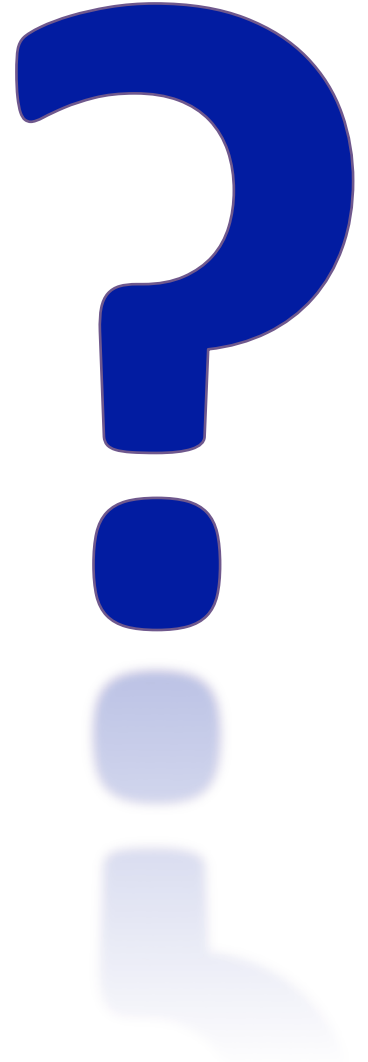
**Note:** the components of a distributed system that reside within a single domain can be trusted by users that operate within that same domain. However, this trust does not expand naturally across domain boundaries.

- users from the new domain may have only read access to the file system in its original domain.
- The new domain does not know which behavior to expect from the foreign code, and may therefore decide to severely limit the access rights for such code.

# What is Distributed Computing?

---

## Some Scalability Solutions

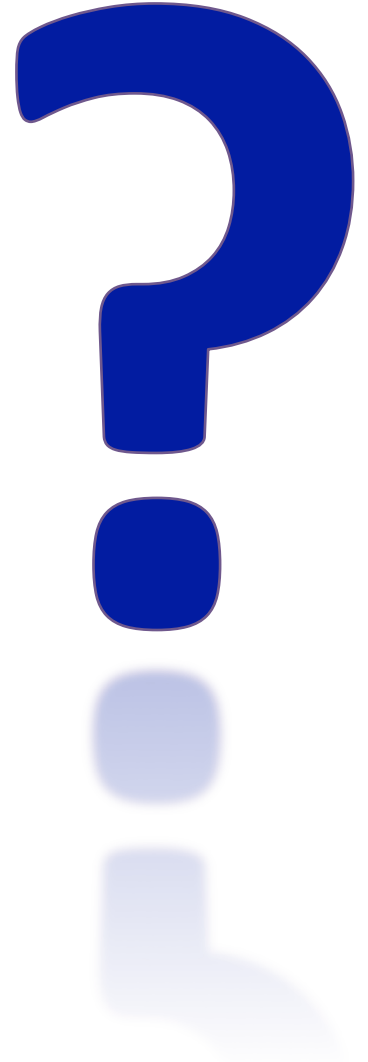




# What is Distributed Computing?

## Some Scalability Solutions

- **Hide communication \_\_\_\_\_(9)**
- **\_\_\_\_\_(4) Distribution**
- **\_\_\_\_cation/caching**



# What is Distributed Computing?

## Some Scalability Solutions

- **Hide communication latencies:** Avoid waiting for responses;
  - Make use of asynchronous communication
  - Have separate handler for incoming response
  - Problem: not all applications fit into this model
- **Load Distribution:** Partition data and computations across multiple machines:
  - Move computations to clients (Java applets)
  - Decentralized naming services (DNS)
  - Decentralized information systems (WWW)
- **Replication/caching:** 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)

# What is Distributed Computing?

## Summary

We discussed the main following topics:

- What is meant by **distributed computing**
- The meaning of **Middleware**
- Distributed computing vs. **parallel computing**
- The **Strengths** and **Weaknesses**
- **Goals** and **Challenges**

# What is Distributed Computing?

## Summary

- Distributed systems consist of autonomous computers that work together to give the appearance of a single coherent system.
  - One important advantage is that they make it easier to **integrate** different applications running on different computers into a single system.
  - Another advantage is that when properly designed, distributed systems **scale well** with respect to the size of the underlying network.
- Distributed systems often aim at hiding many of the intricacies related to the distribution of processes, data, and control.
  - However, this distribution **transparency** not only comes at a price-performance price, but in practical situations it can never be fully achieved.
  - The fact that **trade-offs** need to be made between achieving various forms of distribution transparency and can easily complicate their understanding.

# What is Distributed Computing?

## Activity

Analyse the operation and functionalities of an application or internet protocol. Determine how this application meet the objectives of a distributed application from the point of view of:

- Accessibility
- Openness
- Heterogeneity
- Transparency
- Scalability
- Fault tolerance
- Security



# Distributed Computing

---

## Chapter 1 – Introduction

### 1.1. Computing Evolution

### 1.2. What is distributed Computing?

### 1.3. Distributed System Architectures

### 1.4. Applications of Distributed Systems

# Distributed System Architectures

---

Deciding upon software components, their interaction, and their placement leads to an **instance of system architecture**.

The most common classification of the distributed systems architectures distinguishes two types of basic architectures:

- Centralised architectures
- Decentralised architectures
  
- Hybrid architectures

# Distributed System Architectures

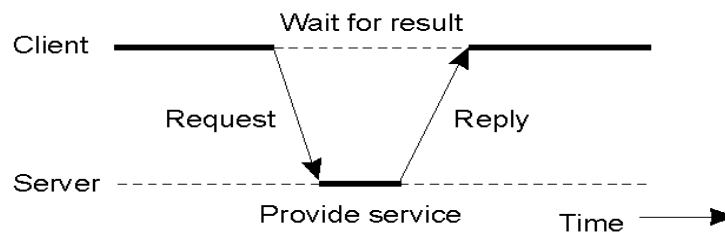
## Centralised

### Basic Client-Server Model:

- **Client**: makes a request to another program and waits for the reply.  
E.g., a file system service or a database service
- **Server**: a process which fulfills the requests from Clients.
- Clients and servers can be distributed across different machines
- This client-server interaction, also known as **request-reply behaviour**

### Considerations

Main problem to deal with: \_\_\_\_\_(4)liable communications  
often both roles simultaneously for different services





# Distributed System Architectures

## **Centralised** - Unreliable communication

- How can a client tell that a request message was lost?
  - [The Request Time Out Message](#) is one approach.
- How can a client detect the difference between a request message that was lost, and a reply message that was lost?
- Does using a connection-oriented protocol like TCP help?

# Distributed System Architectures

## **Centralised** - Unreliable communication

- What guarantees does TCP provide?
  - reliable, byte-sequence.
- TCP provides guarantees only in the absence of errors.
  - Packets can be lost, but this can be considered as a “normal” operation.
- If you want to make sure that the data actually got there and got processed, you need to wait for an application-level acknowledgement from the receiver.

# Distributed System Architectures

## **Centralised** - Unreliable communication

- Can you classify these into two main categories?
  - Read my account balance.
  - Transfer \$100 from savings to checking account.
  - Change block 100 of file A to "abcdef...".
  - Move block 100 of file A to block 200.
- When an operation can be repeated multiple times without harm, it is said to be **idempotent**.
- Since some requests are idempotent and others are not
  - it should be clear that there is **no single solution** for dealing with lost messages.

# Distributed System Architectures

## Centralised - Application Layering

Traditional three-layered view:

- User-\_\_\_\_\_(9) or P\_\_\_\_\_(11) layer
- L\_\_\_\_\_(6) or B\_\_\_\_\_(7) layer
- \_\_\_\_\_(4) layer

**Observation:** This layering is found in many distributed information systems,

- using traditional database technology and accompanying applications.

# Distributed System Architectures

## Centralised - Application Layering

### Traditional three-layered view:

- **User-interface or Presentation** layer contains units for an application's user interface
- **Logical or Business** layer contains the functions of an application, i.e. without specific data
- **Data** layer contains the data that a client wants to manipulate through the application components

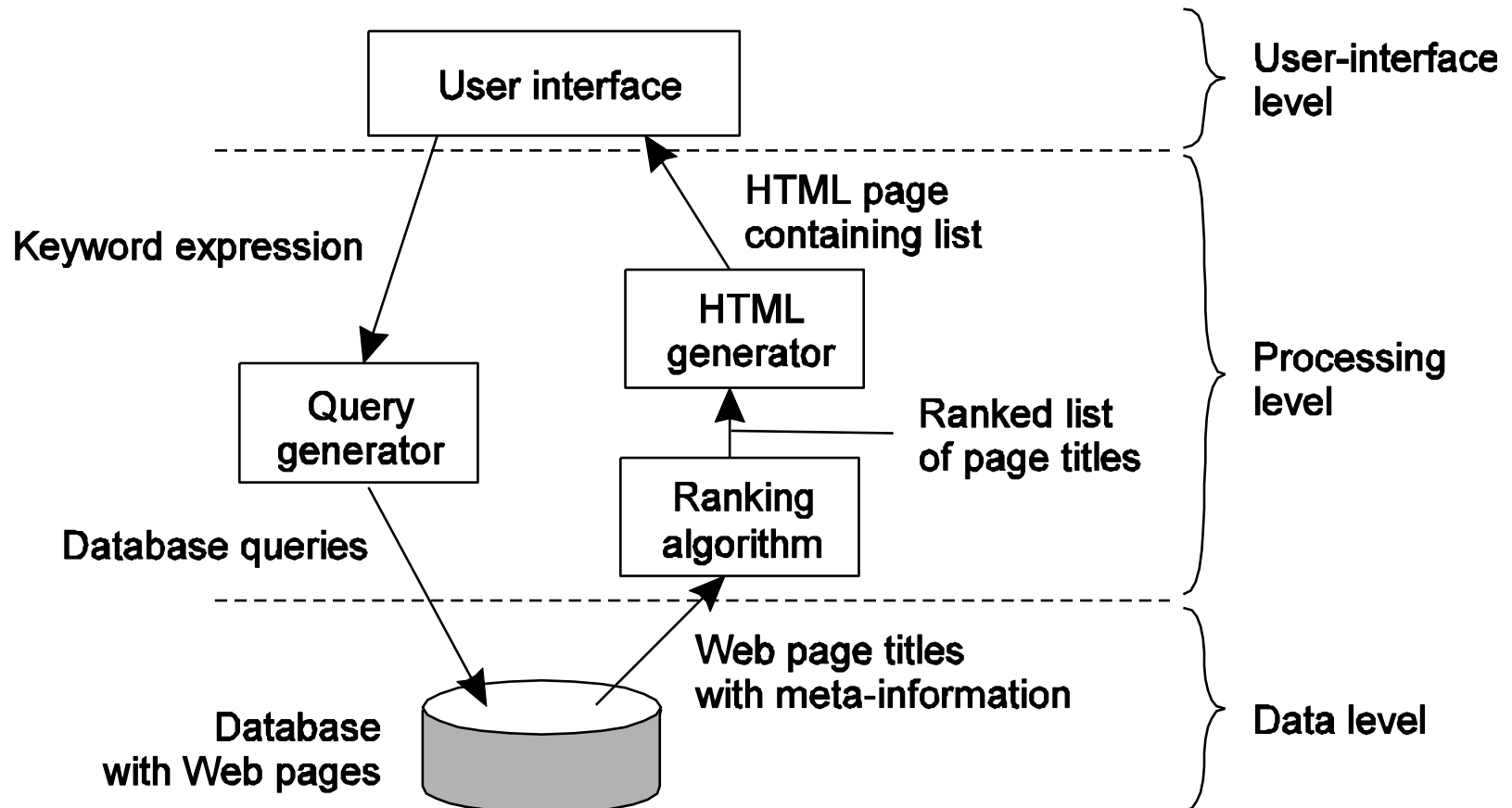
**Observation:** This layering is found in many distributed information systems,

- using traditional database technology and accompanying applications.

# Distributed System Architectures

## Centralised - Application Layering

- The Internet Search Engine example

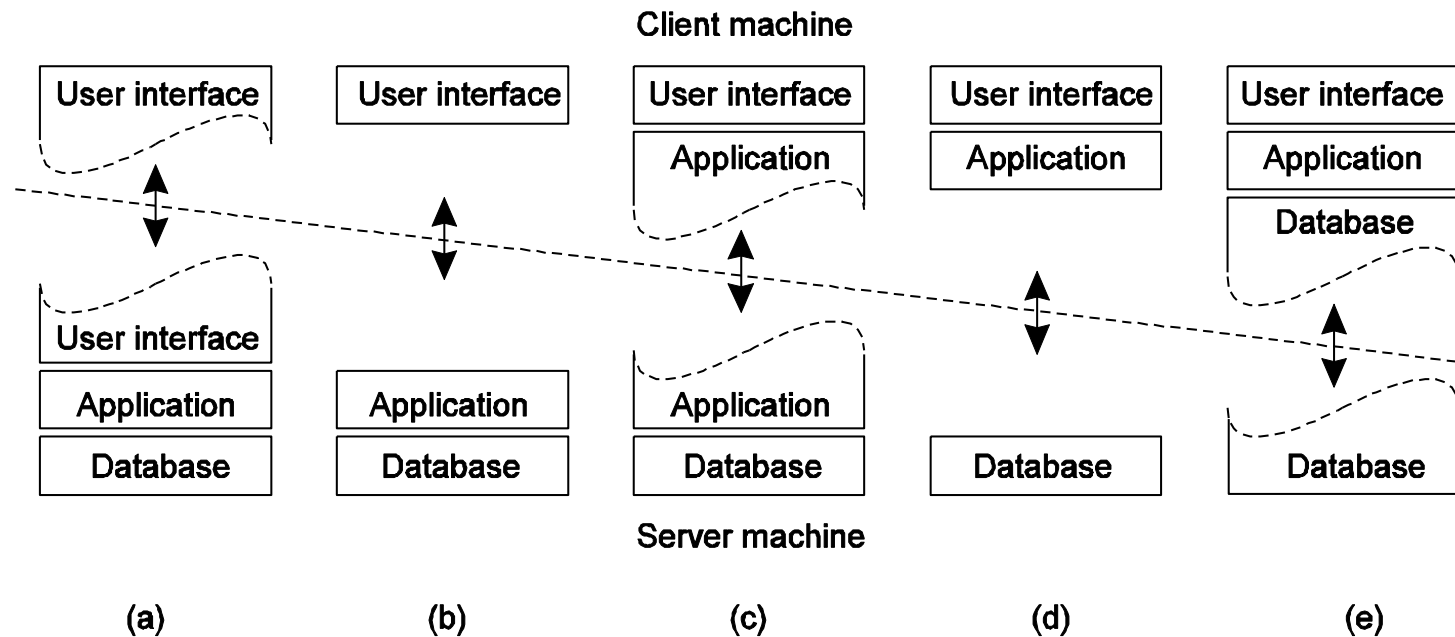


# Distributed System Architectures

## Centralised – Application Layering

- **Single-tiered**: dumb terminal/mainframe configuration
- **Two-tiered**: client/single server configuration
- **Three-tiered**: each layer on separate machine

Traditional two-tiered configurations:

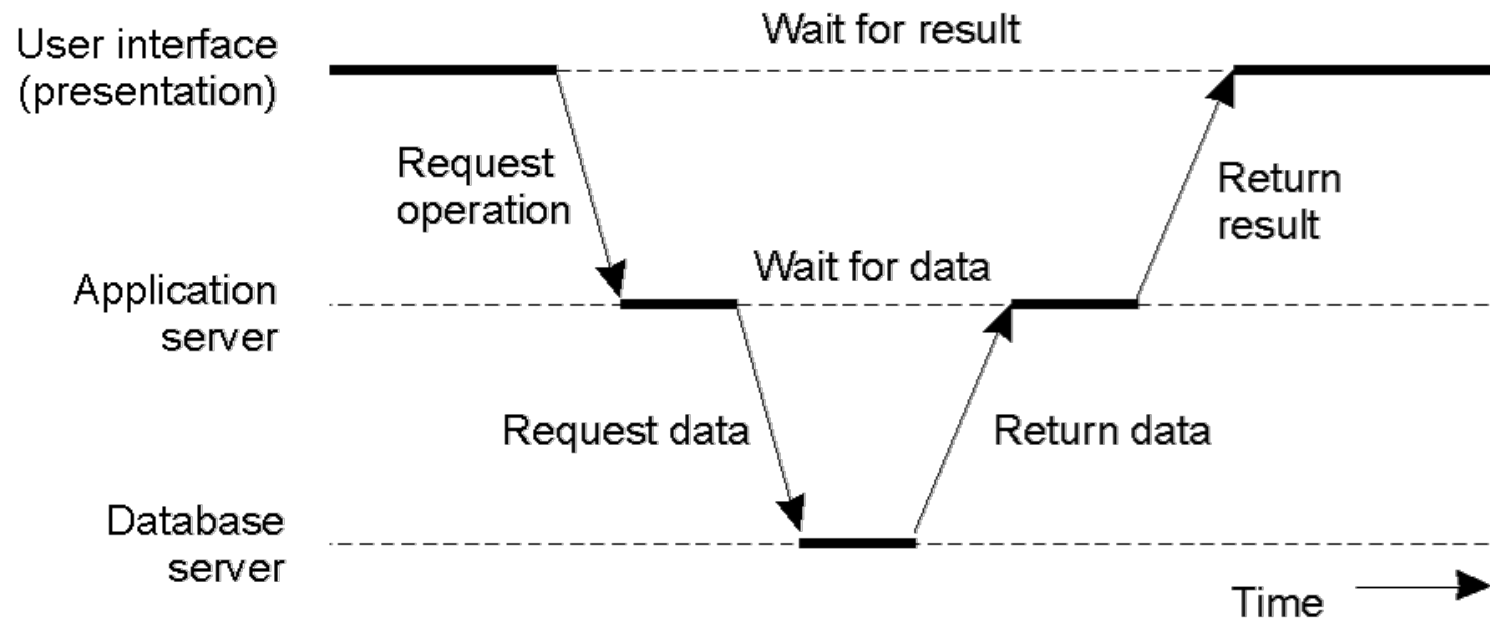


# Distributed System Architectures

## Centralised - Application Layering

- Physical 3-Tiered architecture

Server-side solutions are becoming increasingly more distributed as a single server is being replaced by multiple servers running on different machines. A server may sometimes need to act as a client

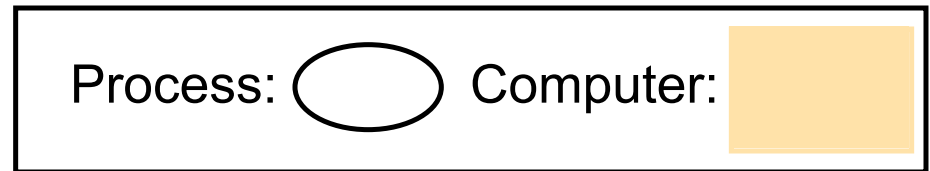
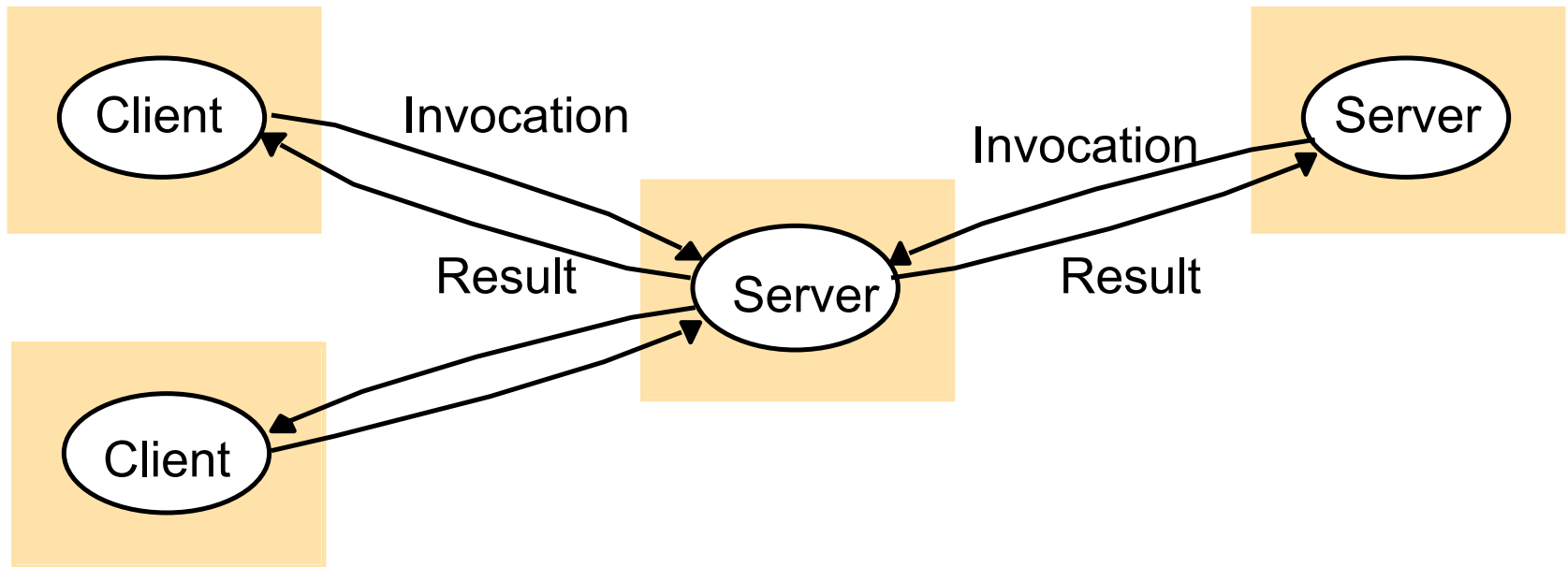




# Distributed System Architectures

## Centralised - Application Layering

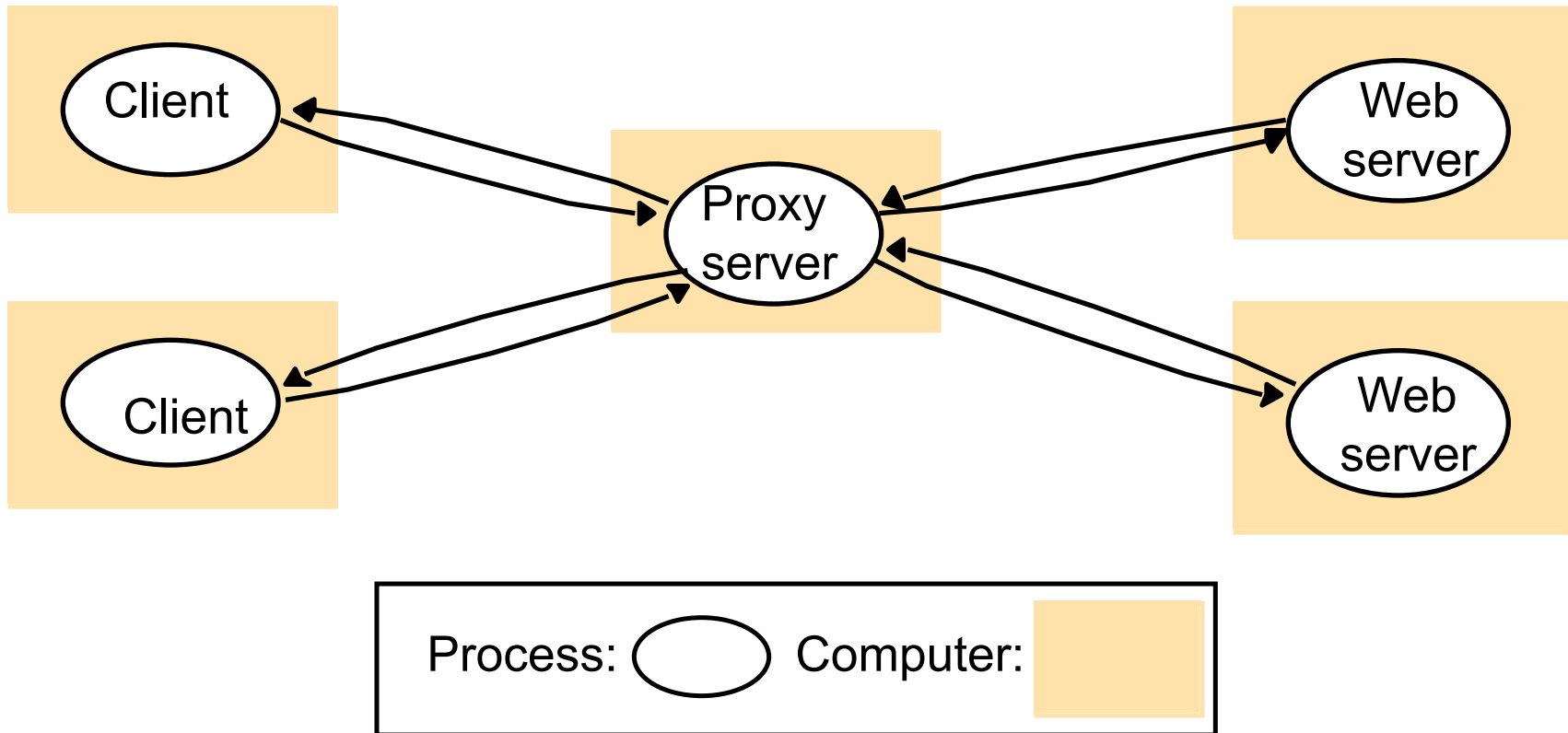
- **3-Tier Example:** Clients Invoke Individual Servers



# Distributed System Architectures

## Centralised - Application Layering

- **3-Tier Example:** Web Proxy Server



# Distributed System Architectures

## Centralised - Application Layering

- **3-Tier Example:** Web Proxy Server
- The **Proxy Server** can be shared for one or more clients
- The main objective is to increase the **availability** and **performance** reducing the load in the network and servers
- Usually the proxies incorporate a **cache** memory storage for sharing web resources
- Another important role for the proxy server is the Security, acting as a **firewall** for the web servers

# Distributed System Architectures

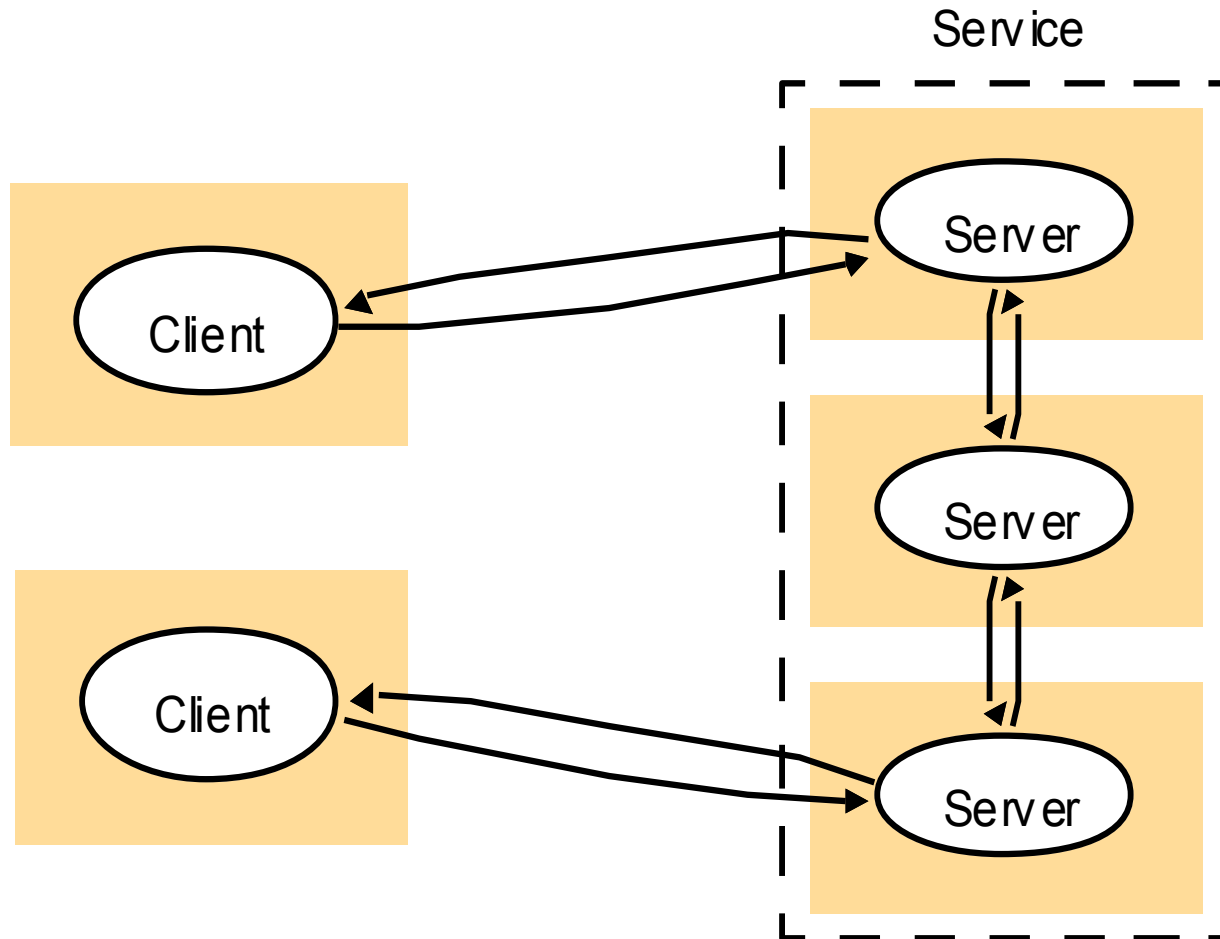
## Decentralised architectures

- Previously, we have looked at what is known as **vertical distribution**.
  - The different tiers correspond directly with the **logical** organization of applications.
    - Multitiered client-server architectures are a direct consequence of dividing applications into a user-interface, processing components, and a data level.
- We can also have **horizontal distribution**, what is that?
  - A client or server may be physically split up into logically equivalent parts,
  - but each part is operating on its own share of the complete data set, thus balancing the load.
  - Operations like **replication**, **mirroring**, **reallocation**, etc., are essential to manage **data and request flow** in this architectures.
  - A class of modern architectures that support horizontal distribution, known as **peer-to-peer**

# Distributed System Architectures

## Decentralised architectures

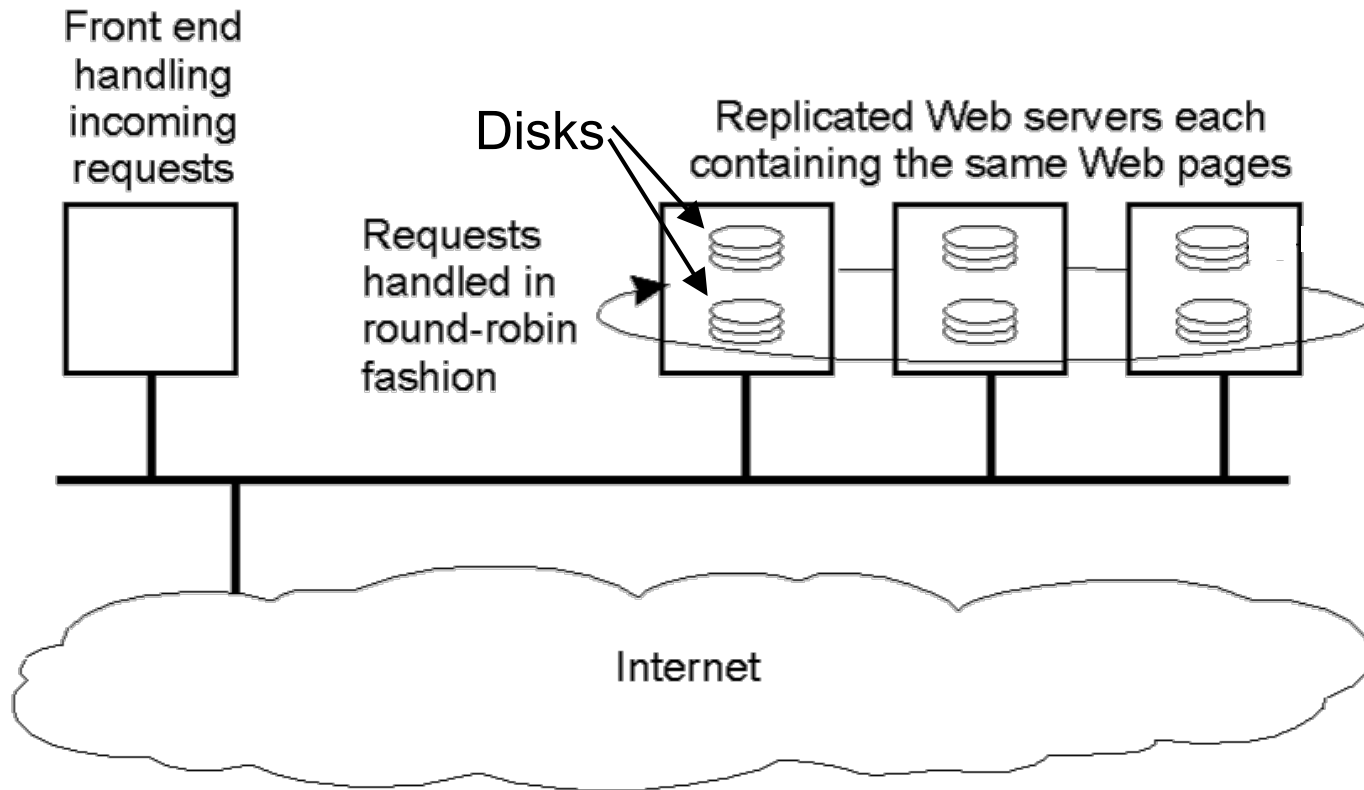
Horizontally distributed servers may talk to each other.



# Distributed System Architectures

## Decentralised architectures

An example of **horizontal distribution** of a Web service.



# Distributed System Architectures

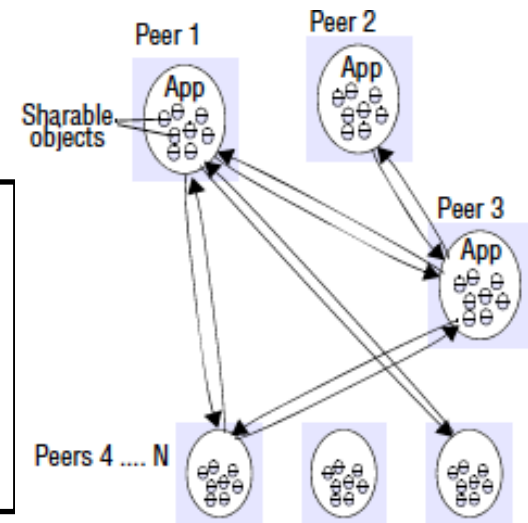
## Decentralised architectures – Peer to Peer

A **peer-to-peer** (P2P) network is a type of decentralized and distributed network architecture in which individual nodes (called "peers") **act as both *suppliers and consumers*** of resources.

- Generally, implement an **overlay network**

What means an overlay network?

**Overlay network**, is a network in which the nodes are formed by processes and the links represent the possible communication channels (which are usually realized as TCP connections). In general, a process cannot communicate directly with an arbitrary other process, but is required to send messages through the available communication channels.



An **overlay** network is **a logical or virtual network**.

- Are neighbors in the overlay network connected by a real link?
- Are nodes that are close in the overlay network close in the physical network too?

# Distributed System Architectures

## Decentralised architectures – Peer to Peer

**Observation:** In the last couple of years we have been seeing a tremendous growth in **peer-to-peer systems**:

- Structured P2P
- Unstructured P2P
- Hybrid P2P

**Note:** Practically, in all cases, we are dealing with **overlay networks**: data is routed over connections set up between the nodes (application-level multicasting).



# Distributed System Architectures

## Decentralised architectures – Peer to Peer

**Observation:** In the last couple of years we have been seeing a tremendous growth in **peer-to-peer systems**:

- **Structured P2P**: nodes are organized following a specific distributed data structure
- **Unstructured P2P**
- **Hybrid P2P**

**Note:** Practically, in all cases, we are dealing with **overlay networks**: data is routed over connections set up between the nodes (application-level multicasting).

# Distributed System Architectures

## Decentralised architectures – Peer to Peer

**Observation:** In the last couple of years we have been seeing a tremendous growth in **peer-to-peer systems**:

- **Structured P2P**: nodes are organized following a specific distributed data structure
- **Unstructured P2P**: nodes have randomly selected neighbors
- **Hybrid P2P**

**Note:** Practically, in all cases, we are dealing with **overlay networks**: data is routed over connections set up between the nodes (application-level multicasting).

# Distributed System Architectures

## Decentralised architectures – Peer to Peer

**Observation:** In the last couple of years we have been seeing a tremendous growth in **peer-to-peer systems**:

- **Structured P2P**: nodes are organized following a specific distributed data structure
- **Unstructured P2P**: nodes have randomly selected neighbors
- **Hybrid P2P**: some nodes are appointed special functions in a well-organized fashion

**Note:** Practically, in all cases, we are dealing with **overlay networks**: data is routed over connections set up between the nodes (application-level multicasting).

# Distributed System Architectures

## Decentralised - **Structured P2P**

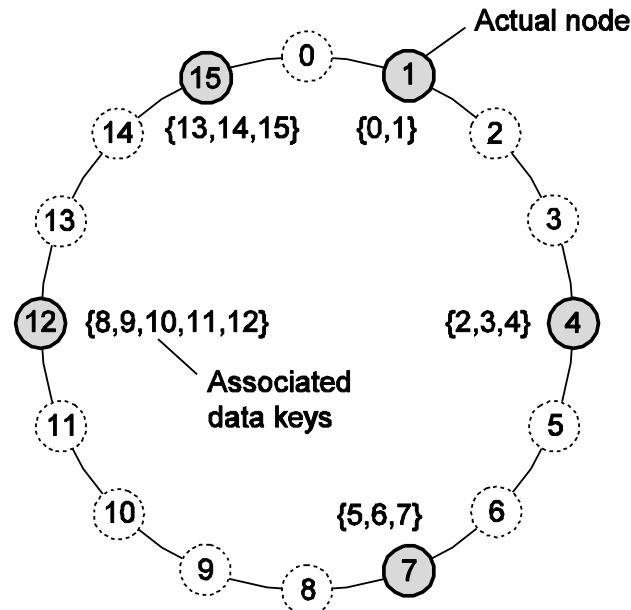
**Basic idea:** Organise the nodes in a structured **overlay network** such as a logical ring, hypercube, tree, etc. Assume that for each data object, there is an associated **key** that is an integer. **Map** the keys in the nodes in a deterministic way.

- By far the most-used procedure is to organize the processes through a **DHT** (*distributed hash table*).

# Distributed System Architectures

## Decentralised - **Structured P2P**

- In a **DHT**-based system, data items are assigned a random key from a large identifier space, such as a 128-bit or 160-bit identifier.
- The nodes are logically organized in a ring such that a data item with key  $k$  is mapped to the node with the smallest identifier  $id \geq k$ . This node is referred to as the **successor** of key  $k$  and denoted as **succ( $k$ )**.



**Note:** The system provides an operation **LOOK UP(key)** that will efficiently **route** the look up request to the associated node.

# Distributed System Architectures

## Decentralised - **Structured P2P**

**Basic idea:** Organize the nodes in a structured overlay network such as a logical ring, hypercube, tree, etc. Assume that for each data object, there is an associated key which is an integer. Map the keys in the nodes in a deterministic way.

- By far the most used procedure is to organize the processes through a DHT (*distributed hash table*)
- **DHT** systems guarantee the location of any object in  $O(\log N)$  steps,  $N$  being the number of peers.
- The path between two nodes in the physical network can be very different from the way in the overlay network. This can cause a quite large search **latency**.

Some example? **eMule KAD, OpenDHT**

# Distributed System Architectures

## Decentralised **Unstructured P2P Systems**

**Observation:** Many unstructured P2P systems attempt to maintain a **random graph**

**Basic principle:** Each node is required to be able to contact other random selected node:

- Each peer maintains a **partial view** of the network, consisting of  $c$  other nodes
- Each node  $P$  periodically selects a node  $Q$  from its partial view
- $P$  and  $Q$  exchange information and exchange members from their respective partial views

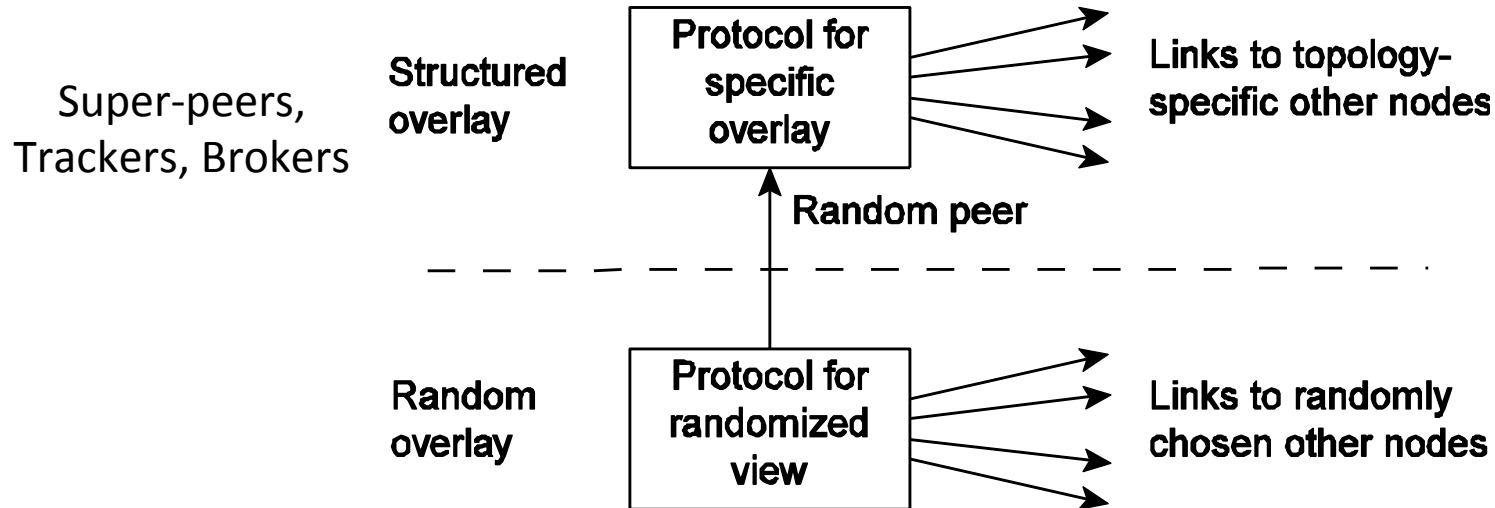
**Observation:** Unstructured systems are useful for locating **highly replicated** (popular) objects and are **reliable** to the connections and disconnections of the nodes. The problems lie in the management of a high rate of requests or the sudden growth of system size.

Some example? **Gnutella**

# Distributed System Architectures

## Hybrid Approaches

**Basic idea:** Distinguish two layers: (1) maintain random partial views in lowest layer; (2) be selective on which roles you keep in higher-layer partial view.



**Note:** lower layer **feeds** upper layer with random nodes; upper layer is **selective** when it comes to keeping references.

Some example? **KaZaA, eDonkey, Skype, BitTorrent**

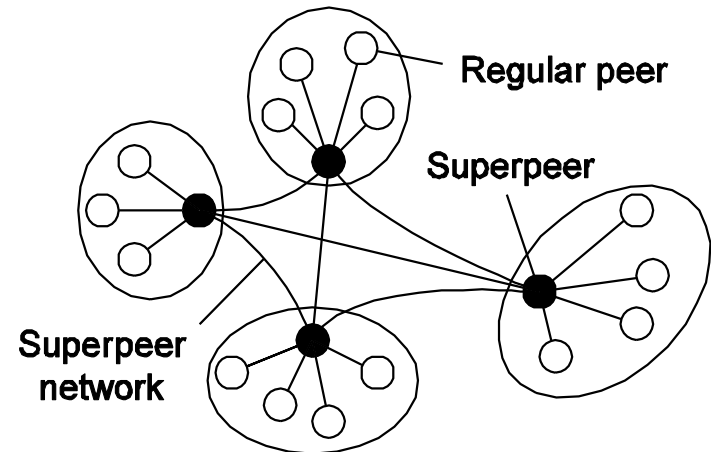


# Distributed System Architectures

## Super-peers, Trackers and Brokers

### Observations

- Some peers are selected to do specific work (**super-peers**)
- In many cases, client-server architectures are combined with peer-to-peer solutions



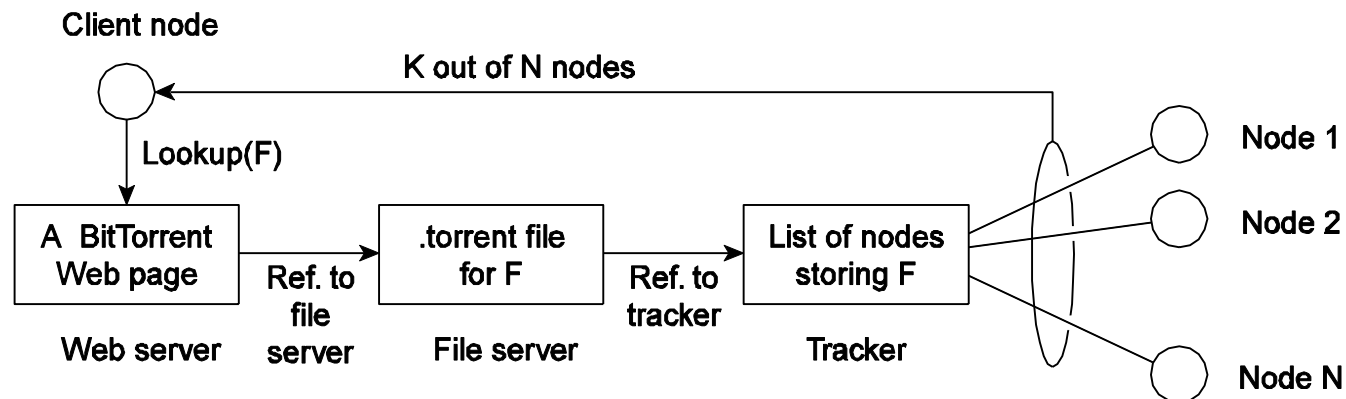
### Examples:

- Peers maintaining an index (for search)
- Peers monitoring the state of the network
- Peers being able to set up connections

# Distributed System Architectures

## Super-peers, Trackers and Brokers

**Example:** Combining a P2P download protocol with a client-server architecture for controlling the downloads: **Bittorrent**

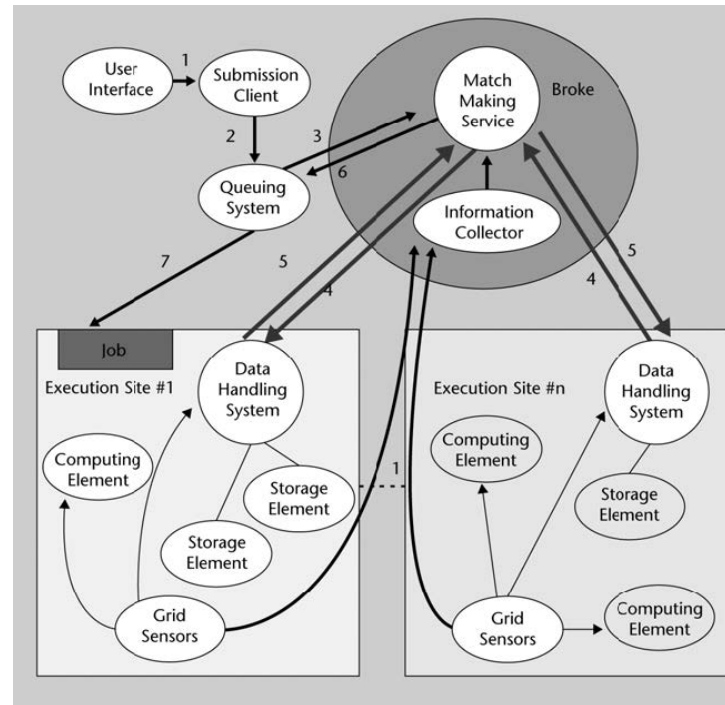


**Basic idea:** Once a node has identified where to download a file from, it joins a **swarm** of downloaders who **in parallel** get file chunks from the source, but also distribute these chunks amongst each other.

# Distributed System Architectures

## Super-peers, Trackers and Brokers

**Example:** Brokering in Grid environments: **Globus**



**Basic idea:** The brokers are responsible for resources monitoring, job scheduling, task allocation, etc.

# Distributed System Architectures

## Super-peers, Trackers and Brokers

New challenges:

- Super-peers election: can be **static**, or selected **dynamically** from the other peers. Leader election algorithms...
- **Reliability, incentives**, etc.

# Distributed System Architectures

---

## Summary

We mainly discussed about the Distributed System architectures, their characteristics, components and concerns:

- Centralized architectures:
  - The Client-Server paradigm
  - Application layering and uses
  - Proxies functionalities
- Decentralized architectures:
  - Horizontal distribution
  - Structures and unstructured P2P architecture
- Hybrid approaches:
  - super-peers, tracker and Brokers

# Distributed Computing

---

## Chapter 1 – Introduction

### 1.1. Computing Evolution

### 1.2. What is distributed Computing?

### 1.3. Distributed System Architectures

### 1.4. Applications of Distributed Systems

# Distributed Systems Applications

---

**WHAT TYPE OF APPLICATIONS**  
WHAT TYPE OF APPLICATIONS



# Distributed Systems Applications

---

- **Distributed Computing Systems**
  - Cluster Computing
  - Grid Computing
  - Collaborative Computing
- **Distributed Information Systems**
  - Information Transaction Systems
- **Distributed Pervasive Systems**
  - Home Automation Systems
  - Electronic Health Care
  - Sensor Networks



# What is Distributed Computing?

## Activity

Choose one of the topics and write a report analyzing it from the point of view of the concepts seen in class such as: Architecture (physical and/or logical), Infrastructure, Applications, Application requirements, Challenges and so on.

The main topics for study are:

- Amazon Web Services (AWS) and Azure ([+info](#)) ([+info](#))
- Collaborative applications over P2P ([+info](#)) (Napster vs. Bittorrent, Spotify, etc.)
- Volunteer computing with BOINC ([+info](#)) (Seti@home, folding@home)
- Top500 List: Architectures, operating systems, network topologies, applications, etc ([+info](#))
- Handling Failure: Fault Tolerance and Monitoring. ([+info](#))([+info](#))
- No SQL databases. What and why? ([+info](#))
- Hadoop + Spark ([+info](#))
- IoT ([+info](#)) ([+info](#))

# DS Applications

## Distributed Computing Systems

**Observation:** Many distributed systems are configured for **High-Performance (HPC)** or **High-Throughput Computing (HTC)**

**Cluster Computing:** Essentially a group of high-end systems connected through a LAN:

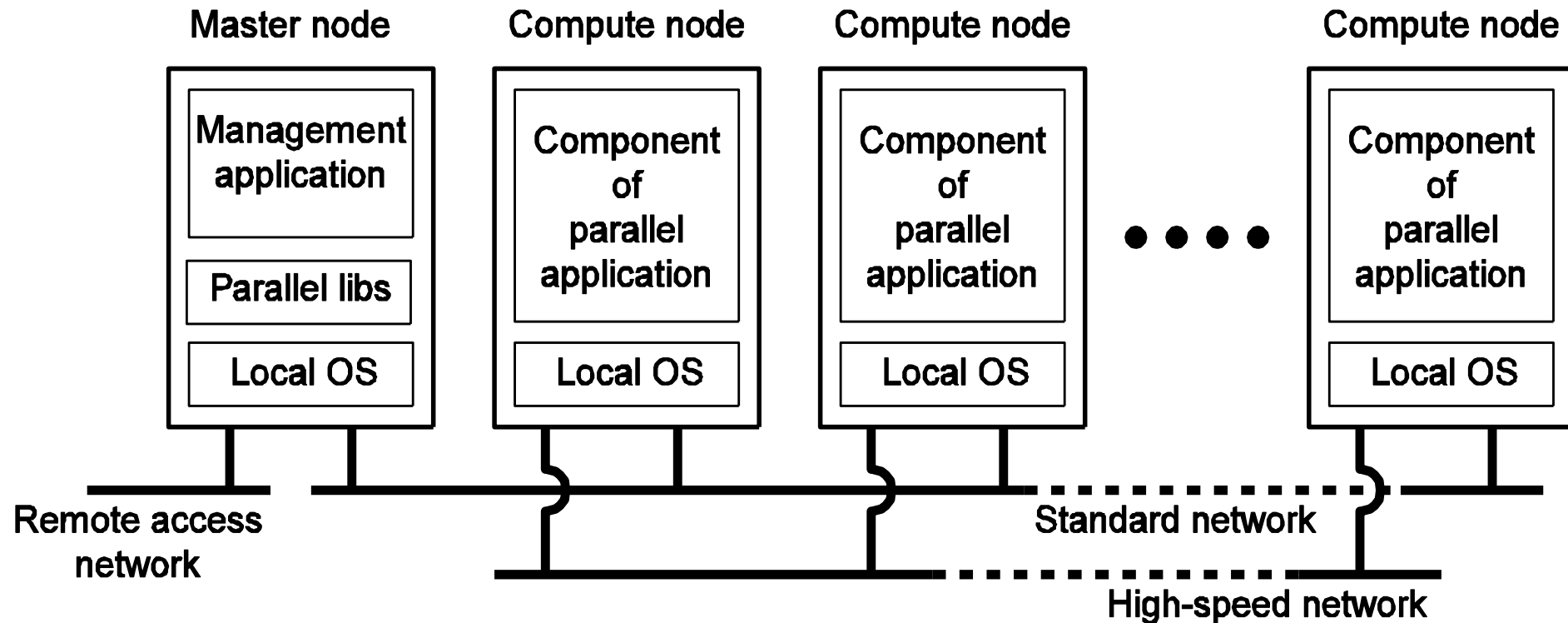
- Homogeneous: same OS, near-identical hardware
- Single managing node
- High-Performance Computing
- Dedicated

There also exists a great quantity of studies about:

- Heterogeneity
- Space & time sharing allocations
- Non-dedicated and High-Throughput systems

# DS Applications

## Cluster Computing



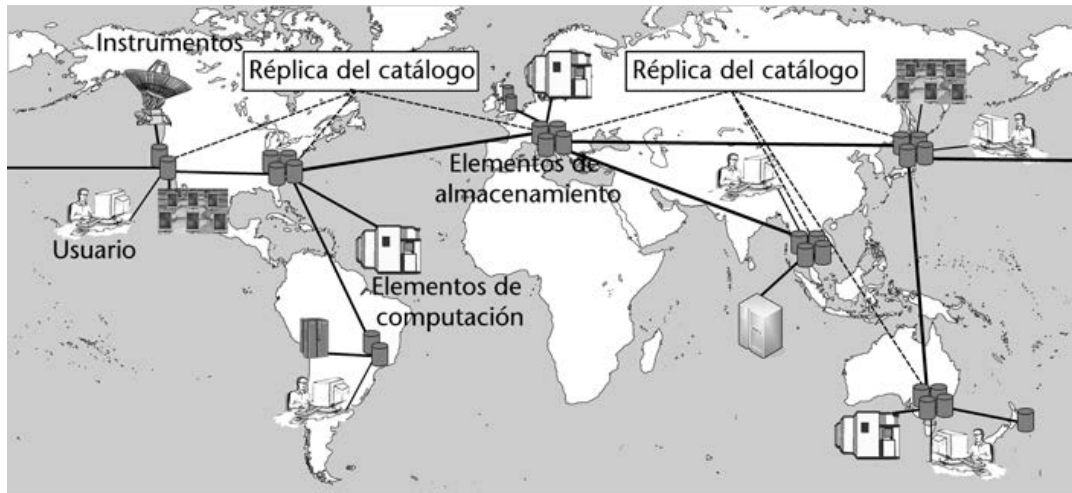
Supporting Software, e.g.: OpenMosix, Condor, PBS, Maui, SGE

# DS Applications

## Grid Computing

Lots of nodes from everywhere:

- Heterogeneous
- Dispersed across several organizations



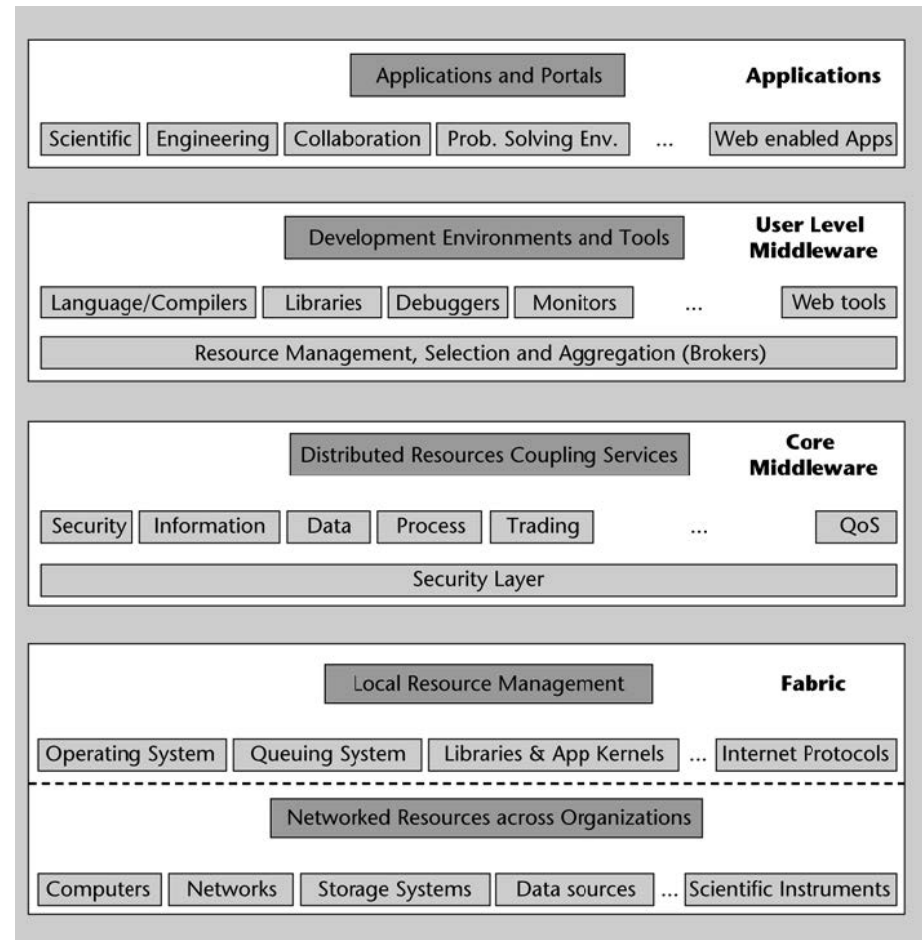
Supporting Software: **Globus Toolkit**

**Note:** To allow for collaborations, grids generally use **virtual organizations**. In essence, this is a grouping of users that will allow for authorization on resource allocation.

# DS Applications

## Grid Computing

- **Fabric layer** provides **interfaces to local resources** at a specific site and the **communication protocols** for supporting grid transactions that span the usage of multiple resources.
- **Core layer** deals with handling access to multiple resources and typically consists of services for **resource discovery**, **allocation** and **scheduling** of tasks onto multiple resources, data replication, and so on.
- **User-level Middleware** application layer consists of the **applications** that operate **within a virtual organization** and which make use of the grid computing environment.
- **Grid Applications** and **Web sites**



# DS Applications

## Cooperative distributed computing projects

Large-scale systems characterized for making use of surplus CPU cycles. These are also called, **Volunteer computing** or **Desktop Grid computing**.

- Project **BOINC** (Berkeley Open Infrastructure for Network Computing). (Year 2007: 430.000 computers, 663TFLOPS)
  - [seti@home](#) project consist in scanning data retrieved by a radio telescope to search for radio signals from another world.
  - [folding@home](#) project consist in simulating the protein folding, which is the process by which proteins reach their final three-dimensional structure, and to examines the causes of protein misfolding for disease (PMD) research purpose.

# DS Applications

## Distributed Information Systems

**Observation:** organizations that were confronted with a wealth of networked applications, but for which interoperability turned out to be a painful experience.

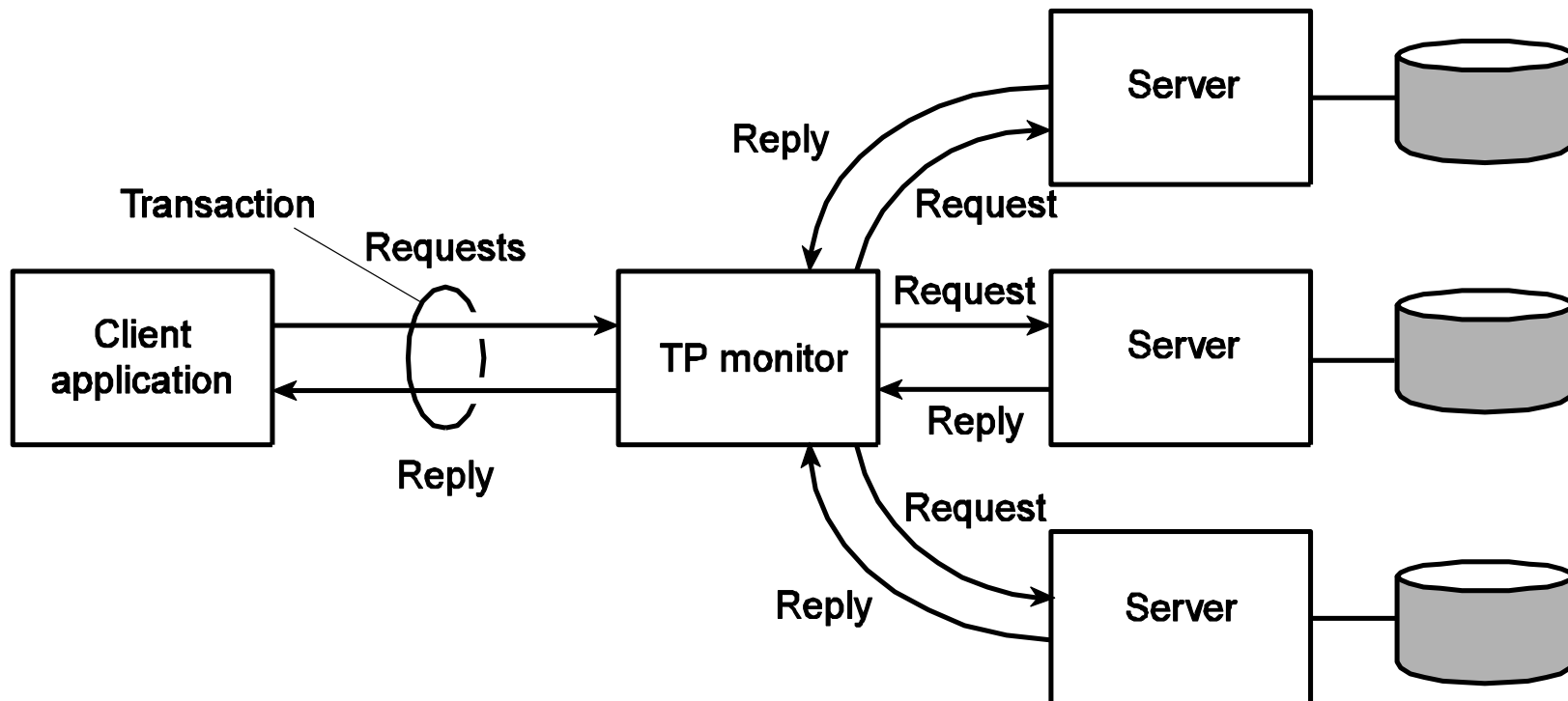
- In many cases, a networked application simply consisted of a server running that application (often including a database) and making it available to remote programs, called clients
- As applications became more sophisticated they were gradually separated into independent components (notably distinguishing database components from processing components)

The main approaches to developing systems that manage large amounts of data and support many users are **transaction-based** systems.

# DS Applications

## Transaction Processing Monitor

**Observation:** In many cases, the data involved in a transaction is distributed across several servers. A **TP Monitor** is responsible for coordinating the execution of a transaction:





# DS Applications

## Transaction Processing Monitor

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.
  - This does not exclude the possibility of invalid, intermediate states during the transaction's execution.
- **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:
  - changes to the state survive failures.

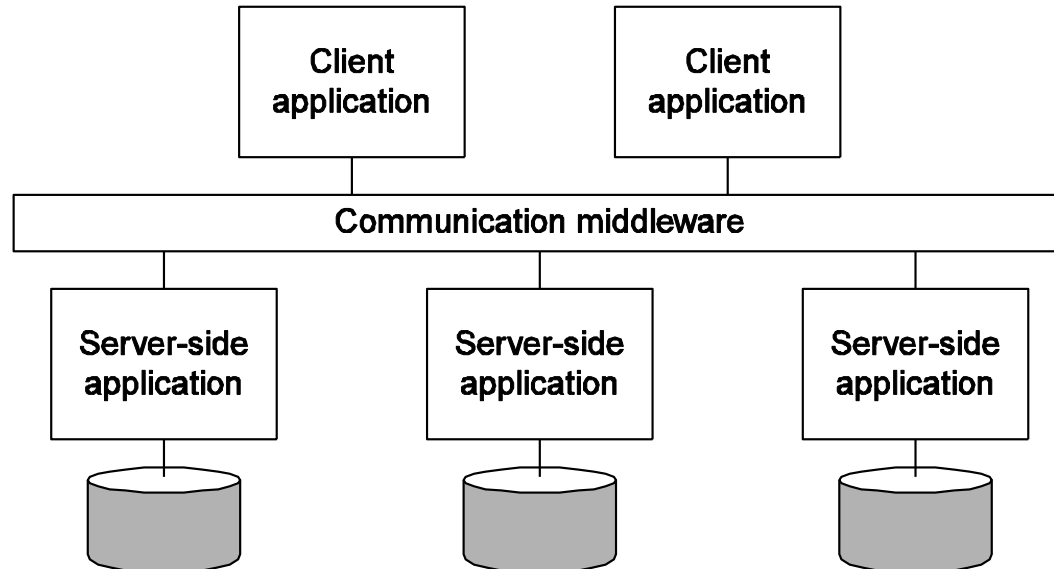
# DS Applications

## Transaction Processing Monitor

**Problem:** A TP monitor works fine for database applications, but in many cases, the apps needed to be separated from the databases they were acting on. Instead, what was needed were facilities for direct communication between applications:

Remote Procedure Call (RPC)

Message-Oriented Middleware (MOM)



# DS Applications

## Distributed Pervasive Systems

**Observation:** There is a next-generation of distributed systems emerging in which the nodes are small, mobile, and often embedded as part of a larger system. 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 a very different way by different users. It requires ease-of-configuration.

**Sharing is the default:** Nodes come and go, providing sharable services and information.

**Observation:** Pervasiveness and distribution transparency may not always form a good match.

# DS Applications

## 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**?

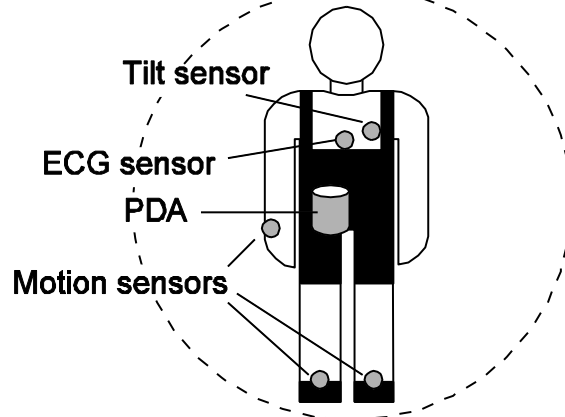
**Electronic health Care Systems:** Devices are physically close to a person.  
Set of sensors forming a Body-area Network (BAN).

Where and how should monitored data be stored?

How can we prevent loss of crucial data?

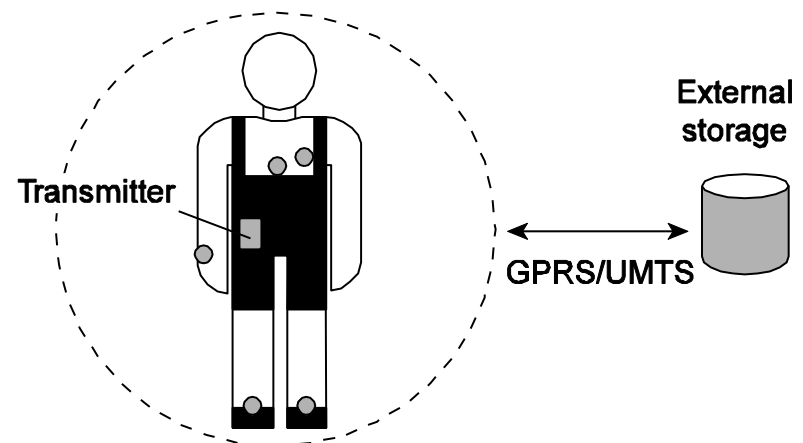
What infrastructure is needed to generate and propagate alerts?

How can security be enforced?



body-area network

(a)



body-area network

(b)

# DS Applications

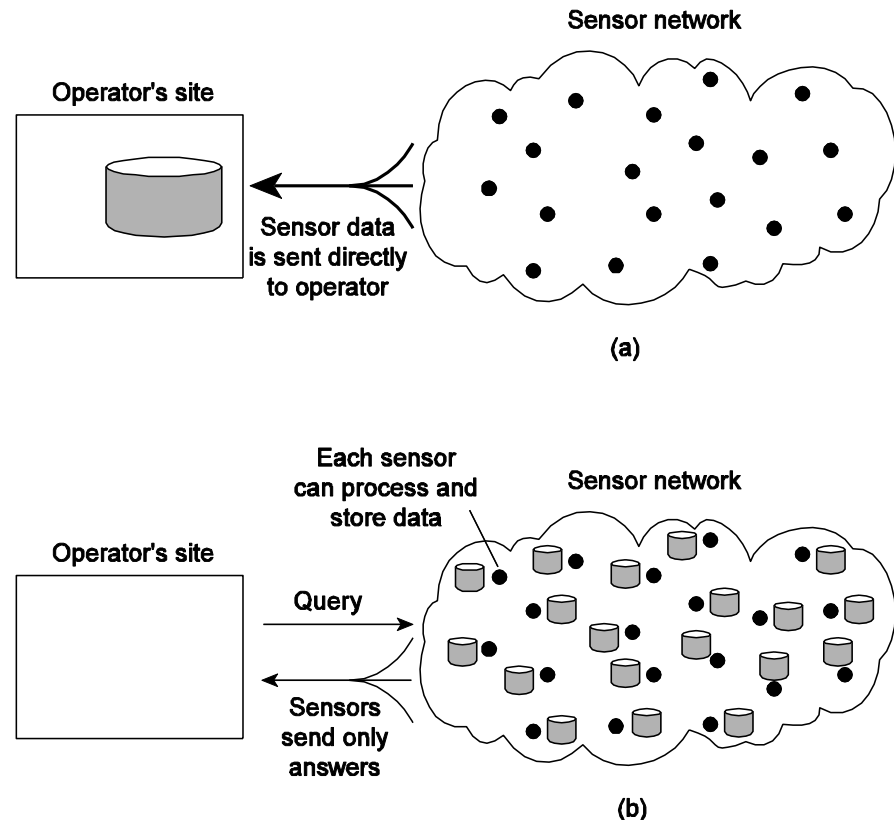
## Distributed Pervasive Systems

### Sensor Networks

The **nodes** to which sensors are attached are:

- Many (10s-1000s)
- Often battery-powered (or even battery-less)
- Simple (i.e., hardly any memory, CPU power, or communication facilities)

Sensor networks as distributed systems: consider them from a **database perspective**.



# Distributed System Architectures

## Summary

Different types of distributed systems exist which can be classified as being oriented toward supporting **computations**, **information processing**, and **pervasiveness**.

- Distributed computing systems are typically deployed for **high-performance** applications often originating from the field of parallel computing.
- A huge class of distributed can be found in traditional office environments where we see databases playing an important role. Typically, **transaction processing systems** are deployed in these environments.
- Finally, an emerging class of distributed systems is where **components are small** and the system is composed in an **ad-hoc fashion**, but most of all is **no** longer managed through a **system administrator**

# Where It Goes

EBay users rarely think about the bidding process—until the site crashes. Behind the scenes, the online auctioneer has a number of safeguards that rely increasingly on duplicated, or mirrored, technologies in case one piece of machinery or software fails. But the information must still pass through many different companies and types of equipment for everything to work properly.

**1** Bidder at home registers and submits an electronic bid from a personal computer.

**2** The bid travels from the consumer's Internet service provider, through switches and routers, to the ISP company's servers.

**3** The bid is sent through the Internet backbone.

**4** The bid travels to one of EBay's ISPs, most likely Sprint or UUNet, and through pipes to EBay.

**5** The bid passes through EBay's Cisco switches and routers.

**6** The information reaches one of about 200 front-line Compaq servers running on Windows NT. The servers are mirrored, so that if any one fails, the others pick up the slack.

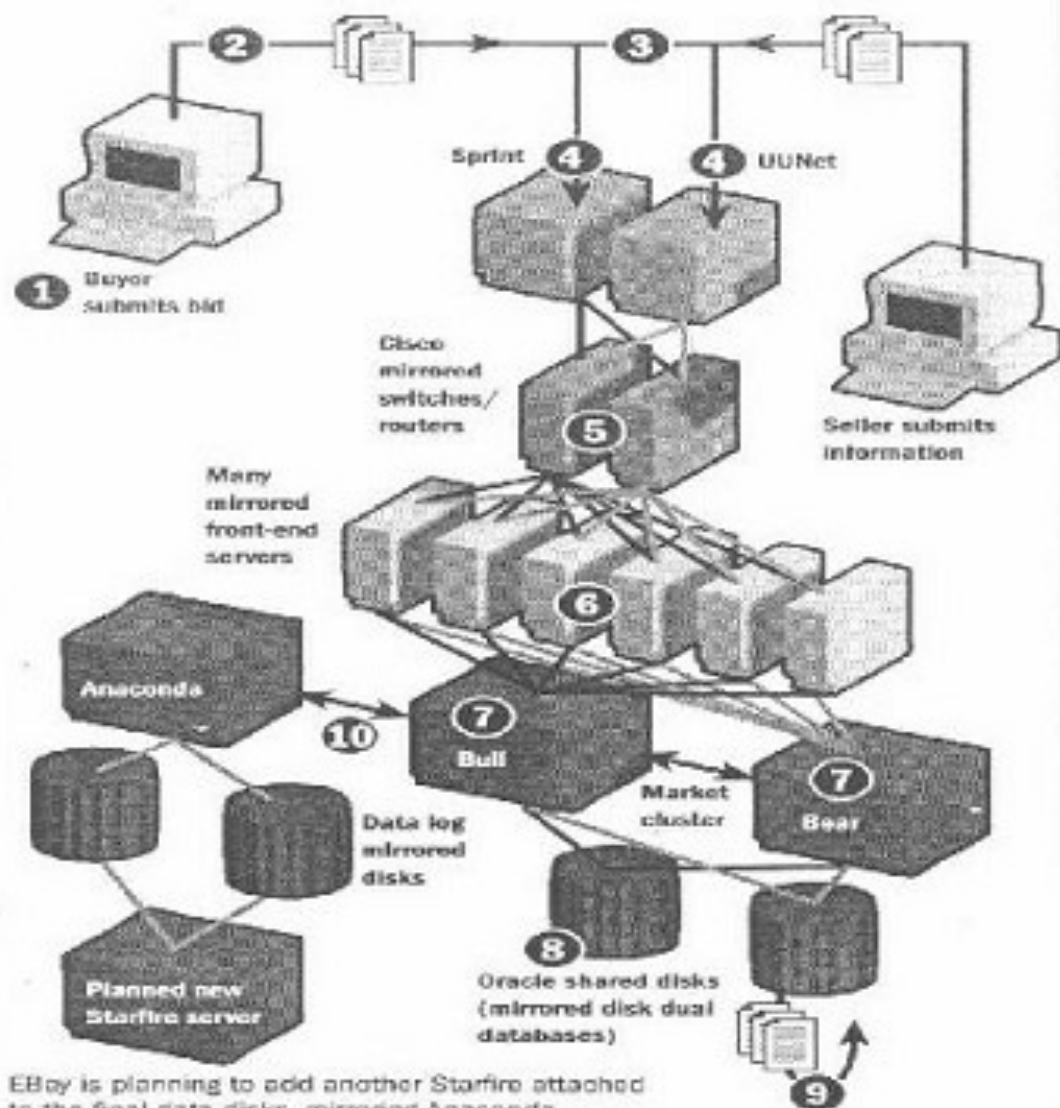
**7** The bid is passed along to one of Sun Microsystems Starfire servers, named Bull and Bear, that mirror each other.

**8** The bid is added to two information-storage databases running Oracle software, where it is matched with the seller's information.

**9** The information flow is reversed back out of EBay, into e-mails sent to both the seller and potential buyers who are outbid. Confirmation is also sent to the bidder.

**10** From Bull, the bid amount and other details are sent to another Starfire server called Anaconda, and recorded on mirrored data storage disks.

Sources: EBay staff, eBay

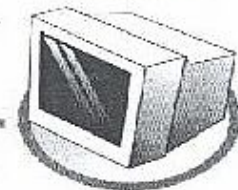
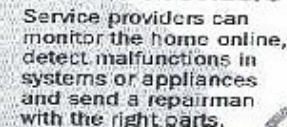
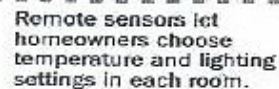
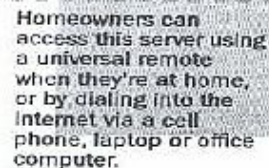
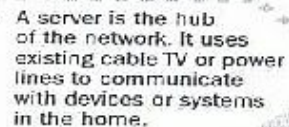




## An example small-scale distributed system (Source: Los Angeles Times.)

THURSDAY, NOVEMBER 16, 2000

conditioning settings, security systems or lighting over the Internet. How an interactive home would work:



The server can distribute video signals on existing coaxial wiring to TVs and PCs throughout the home and allow these devices to share a DVD player.

R. TORO / Los Angeles Times