DISTRIBUTED SYSTEMS
Principles and Paradigms

First Edition

ANDREW S. TANENBAUM
MAARTEN VAN STEEN

Chapter 1
Introduction

By: Hatem Moharram

Definition of a Distributed System (1)

A distributed system is:

A collection of independent computers that appears to its users as a single coherent system.

1- **hardware**: **the machines are autonomous.**

2- **software**: **the users thinks they are dealing with single system.**

important characteristics:

- Differences between the various computers and the way in which they communicate are hidden from users.

- Users and applications can interact with a distributed system in a consistent and uniform way, regardless of where and when interaction take places.

-A Distributed system is easy to expand or scale.

- A Distributed system is continuously available, users and applications should not notice the parts that being replaced or fixed.

Why Distributed Systems?

- Resource sharing
- Availability
- Extensibility
- Improved performance
- Structure of modern organizations

The Competitive Business Environment and The emerging Digital Firm

البيئة التنافسية في الأعمال وظهور المؤسسة الرقمية

- Globalization
- Transformation of Industrial Economy
- Transformation of the Enterprise
- Digital Firm

• Globalization

Emergence of the Global Economy

Management and control in a global marketplace.

Competition in world markets.

Global work groups.

Global delivery Systems.

• Transformation of Industrial Economy

Knowledge- and information-based economies.

Turbulent environment .

productivity.

New products and services.

time based competition.

Short product life.

• Transformation of the Enterprise

.Flattering

.Decentralization

.Flexibility

.Location independence

.Low transaction and coordination costs

.Collaborative work and teamwork

• Digital Firm

- **Digitally enable relationships.**
- **Core business processes accomplished via digital networks.**
- **Digital management.**
- **Rapid sensing and responding to environmental changes.**

To support heterogeneous computers and networks while offering single-system view,

distributed systems are organized by means of layer of software

middleware

The middleware is logically placed between a higher-level layer consisting of users and applications, and a layer underneath consisting of operating system.

Definition of a Distributed System (2)

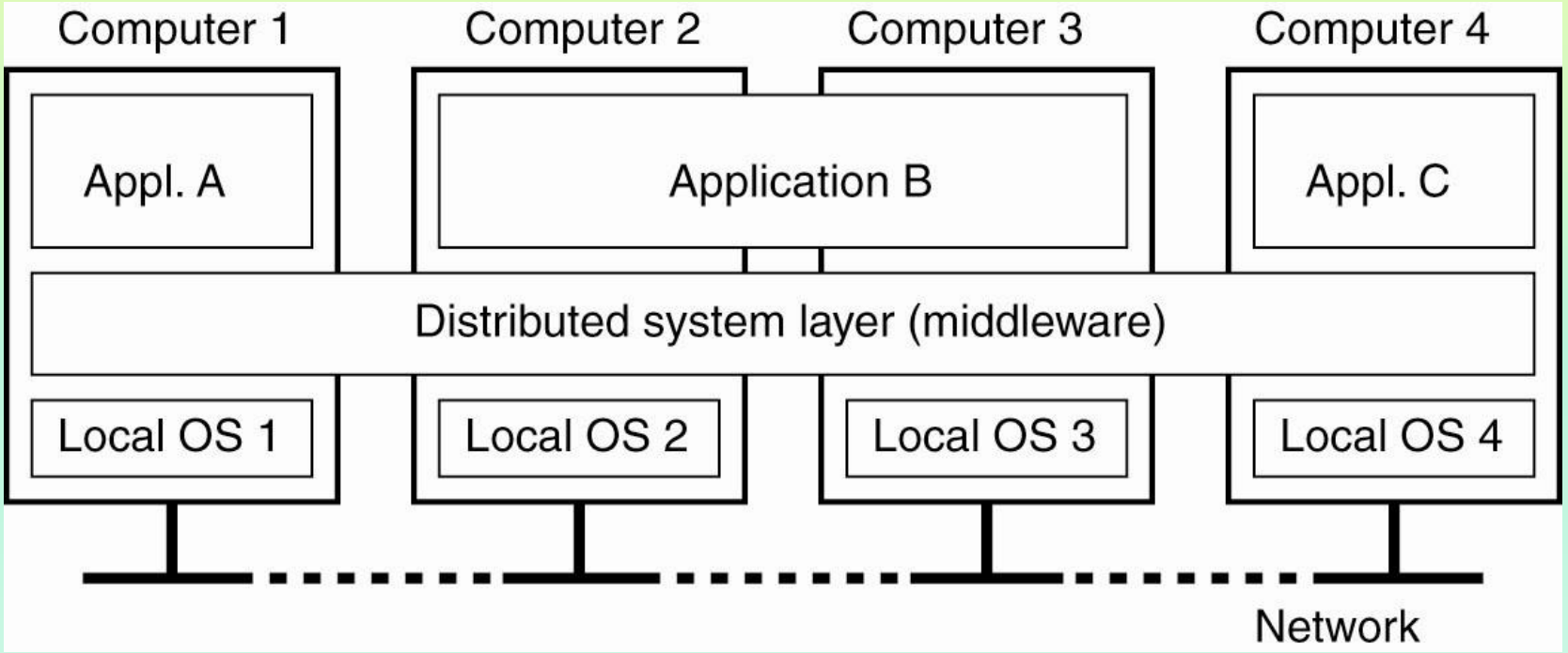


Figure 1-1. A distributed system organized as middleware. The middleware layer extends over multiple machines, and offers each application the same interface.

Ex. (1): network of workstations in a university.

In addition to each user's personal workstation, there might be a pool of processors in the machine room that are not assigned to specific users.

Such a system may have a single file system, with all file accessible from all machines in the same way and using the same path name.

When user types a command, the system could look for the best place to execute that command. If the system as a whole looks and acts like a classical single-processor timesharing system (i.e., multi-user), it qualifies as a distributed system.

Ex. (2): A workflow information system that supports the automatic processing of orders.

used by people from several departments at different locations, orders are placed by means of laptop computers that are connected to the system through the telephone network, possibly using cellular phones.

Incoming orders are automatically forwarded to the planning department, resulting in new internal shipping orders sent to the stock department, as well as billing orders to be handled by the accounting department.

Users are totally unaware of how orders physically flow through the system, to them it appears as if they are all operating on a centralized database.

Ex. (3): The World Wide Web.

The Web offers a simple consistent, and uniform model of distributed documents. To see a document, a user need activate a reference, and the document appears on the screen. In theory (not in practice) there is no need to know from which server the document was fetched.

Publishing a document is simply by giving it a unique name in the form of a URL that refers to a local file containing the document's content.

If the WWW would appear to its users as a huge centralized document system, it would qualify as a distributed system. We have not reached that point.

1.2 Goals

There a four important goals that should be met to make building a distributed system worth the effort.

-Easily connect users to resources.

-Hide the fact that resources are distributed across a network (Transparency).

- Open. a system that offers services according to standard rules: Interoperability, Portability, flexibility

- Scalable.

size scalable, geographically scalable, administratively scalable

1.2.1 Connecting users and resources

Resources can be anything, printers ,computers, storage facilities, data, files, web pages, and networks, etc.,

There are many reasons for sharing resources:

1- economics.

2- easier to collaborate and exchange information.

As connectivity and sharing increase, security is becoming more and more important.

1.2.2 Transparency

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.

Type of transparency:

1- access transparency
2- location transparency
3- migration transparency
4- relocation transparency
5- replication transparency
6- concurrency transparency
7- failure transparency

Transparency in a Distributed System

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

Figure 1-2. Different forms of transparency in a distributed system (ISO, 1995).

1.2.2 Transparency

1- access transparency

Hide the differences in data representation and the way that resources can be accessed by users.

Ex. 1: sending an integer differ from a computer system to another.

Ex. 2: each operating system have its own file-naming conventions.

Users of the Unix network file system (NFS) can use the same commands and parameters for file system operations, such as copying or deleting files, regardless of whether the accessed files are on a local or a remote disk. Likewise, application programmers use the same library function calls to manipulate local and remote files on an NFS.

2- location transparency

Users cannot tell where a resource is physically located in the system.

Can be achieved by assigning only logical names to resources, that is, names in which the location of a resource is not secretly encoded.

Ex.: http://www.prenhall.com/index.html

No clue (sign) about the location of Prentice Hall's main Web server. No clue about the location of index.html.

The users of a video-on-demand system do not have to know the name or the IP address from which their client component downloads a video. Similarly, the application programmer who wrote the client component should not have to know the physical location of the video-on-demand server that provides the video.

1.2.2 Transparency

3- migration transparency

Resources can be moved without affecting how that resource can be accessed.

This mechanism allows for the load balancing of any particular client, which might be overloaded. The systems that implement this transparency are NFS and Web pages.

Migration transparency depends on access and location transparency.

1.2.2 Transparency

4- relocation transparency

Resources can be relocated while they are being accessed without the user or application noticing anything.

Ex.: mobile users can continue to use their wireless laptop while moving from place to place without ever being disconnected.

1.2.2 Transparency

5- Replication transparency

Hiding the fact that several copies of resources exist.

Resources may be replicated to:

1- increase availability or

2- improve performance by placing a copy close to the place where it is accessed.

To hide replication from users, it is necessary that all replicas have the same name.

A system that supports replication transparency should support access and location transparency as well.

In the video-on-demand service, we could imagine that it would be beneficial to replicate the video database server. In that way, the video-on-demand system scales to accommodate many concurrent users. Replication transparency means that customers who watch a video are not aware that the video is downloaded from a replica server rather than the original

1.2.2 Transparency

6- concurrency transparency

Each user does not notice that other users is making use of the same resource.

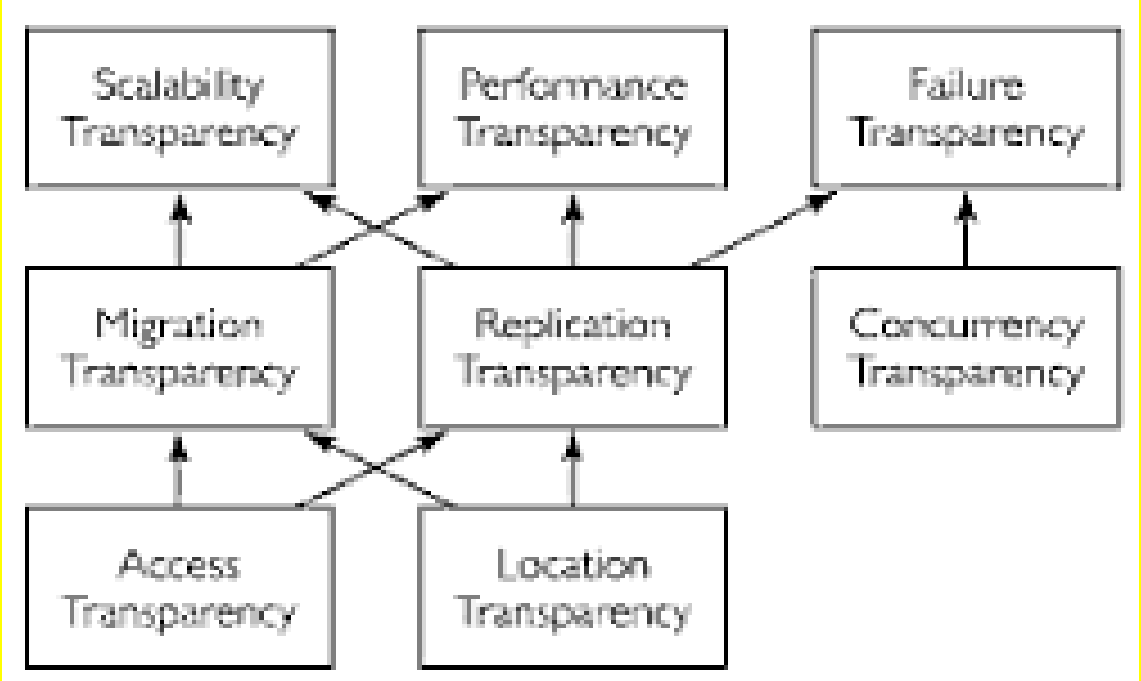
Concurrent access to a shared resource must leaves that resource in a consistent state, this can be achieved through locking mechanisms.

1.2.2 Transparency

7- failure transparency

A user does not notice that a resource (he has possibly never heard of) fails to work properly, and that the system subsequently recovers from that failure.

Masking failures is one of the hardest issues in distributed systems.



1.2.2 Transparency

8- persistence transparency

which deals with masking whether a resource is in volatile memory or perhaps somewhere on a disk.

Ex. : many object oriented databases provide facilities for directly invoking methods on stored objects. What happens behind the scenes, is that the database server first copies the object's state from disk to main memory, performs the operation, and perhaps writes that state back to secondary storage. The user, however, is unaware that the server is moving state between primary and secondary memory.

Degree of transparency

there are situations in which attempting to blindly hide all distribution aspects from users is not always a good idea.

Ex.: many Internet applications repeatedly try to contact a server before finally giving up. Attempting to mask a transient server failure before trying another one may slow down the system as a whole. In such a case, it may have been better to give up earlier, or at least let the user cancel the attempts to make contact.

Aiming for distribution transparency is a nice goal when designing and implementing distributed systems, but other issues must be considered such as performance.

1.2.3 openness

An open distributed system is:

a system that offers services according to standard rules that describe the syntax and the semantics of those services.

Ex.: in computer networks, standard rules govern the format, contents, and meaning of messages sent and received. Such rules are formalized in protocols.

1.2.3 openness

In distributed systems, services are generally specified through interfaces, which are often described in an Interface Definition Language (IDL).

Interface definitions written in a IDL nearly always capture only the syntax of services.

The semantics of the interface are given in a an informal way by means of natural language.

1.2.3 openness

Interoperability characterizes the extent by which two implementations of systems or components from different manufactures can co-exist and work together by merely relying on each other's services as specified by common standard.

operating systems that run on different CPUs, and Microsoft Word's ability to read files created using Excel

Portability characterizes to what extent an application developed for distributed system A can be executed, without modification, on a different distributed system B that implements the same interfaces as A.

Unix operating systems are widely used in servers, workstations, and mobile devices. Unix was designed to be portable, multi-tasking and multi-user in a time-sharing configuration.

1.2.3 openness

flexibility, meaning that it should be easy to configure the system out of different components possibly from different developers.

Also, it should be easy to add new components or replace existing ones without affecting those components that stay in place.

1.2.4 Scalability

Scalability of a system can be measured along at least three different dimensions :

1- a size scalable system is one in which we can easily add more users and resources to the system.

2- a geographically scalable system is one in which the users and resources may lie far apart.

3- an administratively scalable system is one in which it is easy to manage even if it spans many independent administrative organizations.

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

1.2.4 Scalability

A- scaling with respect to size:

If more users or resources need to be supported, we are often faced with the limitations of centralized:

- services
- data,
- algorithms

1.2.4 Scalability

Scalability Problems

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

1.2.4 Scalability

A- scaling with respect to size

A-1 centralized services

Ex.: only a single server running on a specific machine in the distributed system. the server can simply become a bottleneck as the number of users grows. Even if we have virtually unlimited processing and storage capacity, communication with that server will eventually prohibit further growth.

using only a single server is sometimes unavoidable.

Ex.: service for managing highly confidential information such as medical records, bank accounts, personal loans.

A-2 centralized data

Ex. 1: How should we keep track of the telephone numbers and addresses of 50 million people? A single 2.5-gigabyte disk would provide enough storage. But, having a single database would undoubtedly saturate all the communication lines into and out of it.

1.2.4 Scalability

A- scaling with respect to size

A-2 centralized data

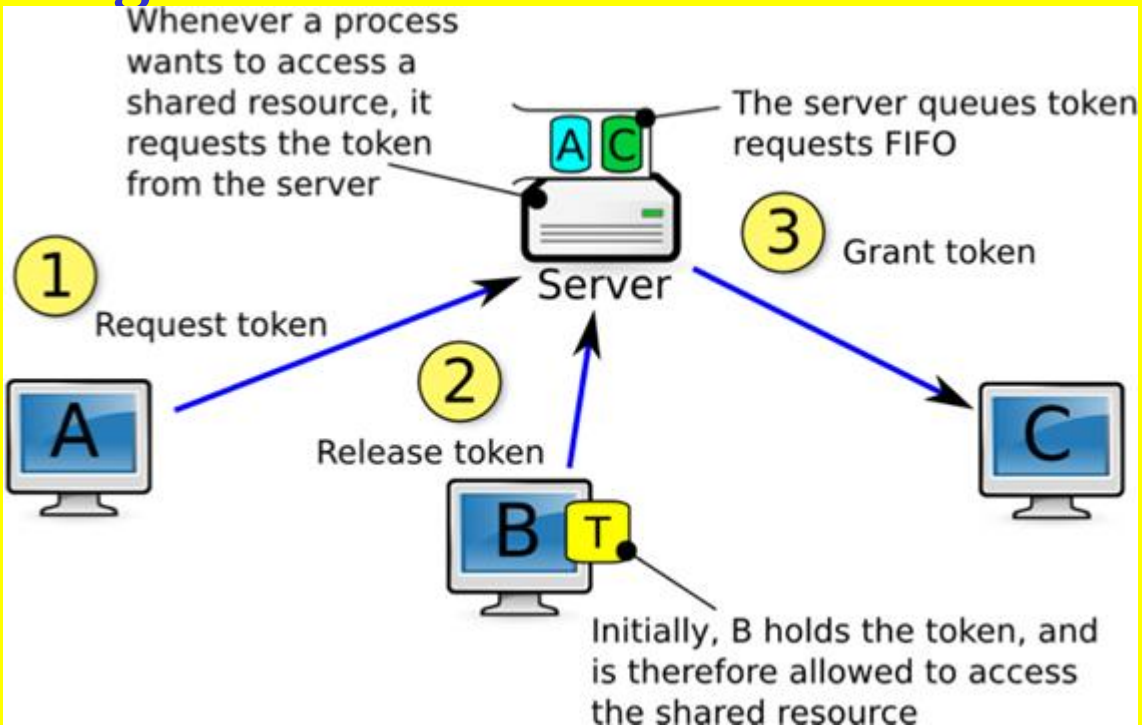
Ex. 2: how the Internet would work if its Domain Name System (DNS) was still implemented as a single table. If each request to resolve a URL had to be forwarded to that one and only DNS server, it is clear that no one would be using the Web (which, by the way, would probably solve the problem again).

1.2.4 Scalability

A- scaling with respect to size

A-3 centralized algorithms

any algorithm that operates by collecting information from all sites, sends it to a single machine for processing, and then distributes the results must be avoided. Only decentralized algorithms should be used.



1.2.4 Scalability

A- scaling with respect to size

A-3 centralized algorithms

characteristics of decentralized algorithms:

l. 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.

B- Geographical scalability

B-1 synchronous communication

A party requesting service, generally referred to as a client, blocks until a reply is sent back. Synchronous is real time, and asynchronous is time delayed.

Synchronous communication takes place when people are connected at the same time (real time communication) whereas asynchronous communication does not require people to be online at the same time.

Ex.:

Synchronous: phone calls, skype, text chat, face-to-face talk.
Asynchronous: e-mails, blogs, forums, video blogs

local-area networks are based on synchronous communication. communication between two machines is generally at worst a few hundred microseconds.

Wide-area system, interprocess communication may be hundreds of milliseconds, three orders of magnitude slower. Building interactive applications using synchronous communication in wide-area systems requires a great deal of care.

B-2 communication reliability

A reliable communication is acommunication where messages are guaranteed to reach their destination complete and uncorrupted and in the order they were sent.

communication in wide-area networks is unreliable, and virtually always *point-to-point*.

local-area networks generally provide highly reliable communication facilities based on *broadcasting*, making it much easier to develop distributed systems.

1.2.4 Scalability

B- Geographical scalability

Ex.: consider the problem of locating a service. In a local-area system, a process can simply broadcast a message to every machine, asking if it is running the service it needs. Only those machines that have that service respond, each providing its network address. Such a location scheme is unthinkable in a wide-area system.

B-3 centralized solutions

If we have a system with many centralized components, it is clear that geographical scalability will be limited due to the performance and reliability problems resulting from wide-area communication.

In addition, centralized Components now lead to a waste of network resources.

1.2.4 Scalability

B- Geographical scalability

Ex.: a single mail server an entire country. This would mean that sending an e-mail to your neighbor would first have to go to the central mail server, which may be hundreds of miles away. Clearly, this is not the way to go.

C- administrative scalability

how to scale a distributed system across multiple, independent administrative domains?

A major problem that needs to be solved is that of conflicting policies with respect to resource usage (and payment), management, and security.

Ex.: the users trust their system administrators. However, this trust does not expand naturally across domain boundaries.

1.2.4 Scalability

two types of security measures need to be taken:

1- the distributed system protection against malicious attacks from the new domain.

2- the new domain has to protect itself against malicious attacks from the distributed system.

1.2.4 Scalability

Scaling Techniques

how those problems can generally be solved?

there are basically only three techniques for solving scalability problems:

A- hiding communication latencies,

B- distribution, and

C- replication

A- hiding communication latencies

Latency is a measure of time delay experienced in a system.

avoid waiting for responses to remote service requests as much as possible.

(applicable in the case of geographical scalability)

1- asynchronous communication:

an alternative to waiting for a reply from the server is to do other useful work at the requester's side.

this means constructing the requesting application in such a way that it uses only asynchronous communication.

A- hiding communication latencies

there are many applications that cannot make effective use of asynchronous communication
(in interactive applications).

better solution is:

2- reduce the overall communication,

Ex: by moving part of the computation that is normally done at the server to the client process requesting the service. A typical case where this approach works is accessing databases using forms.

Scaling Techniques (1)

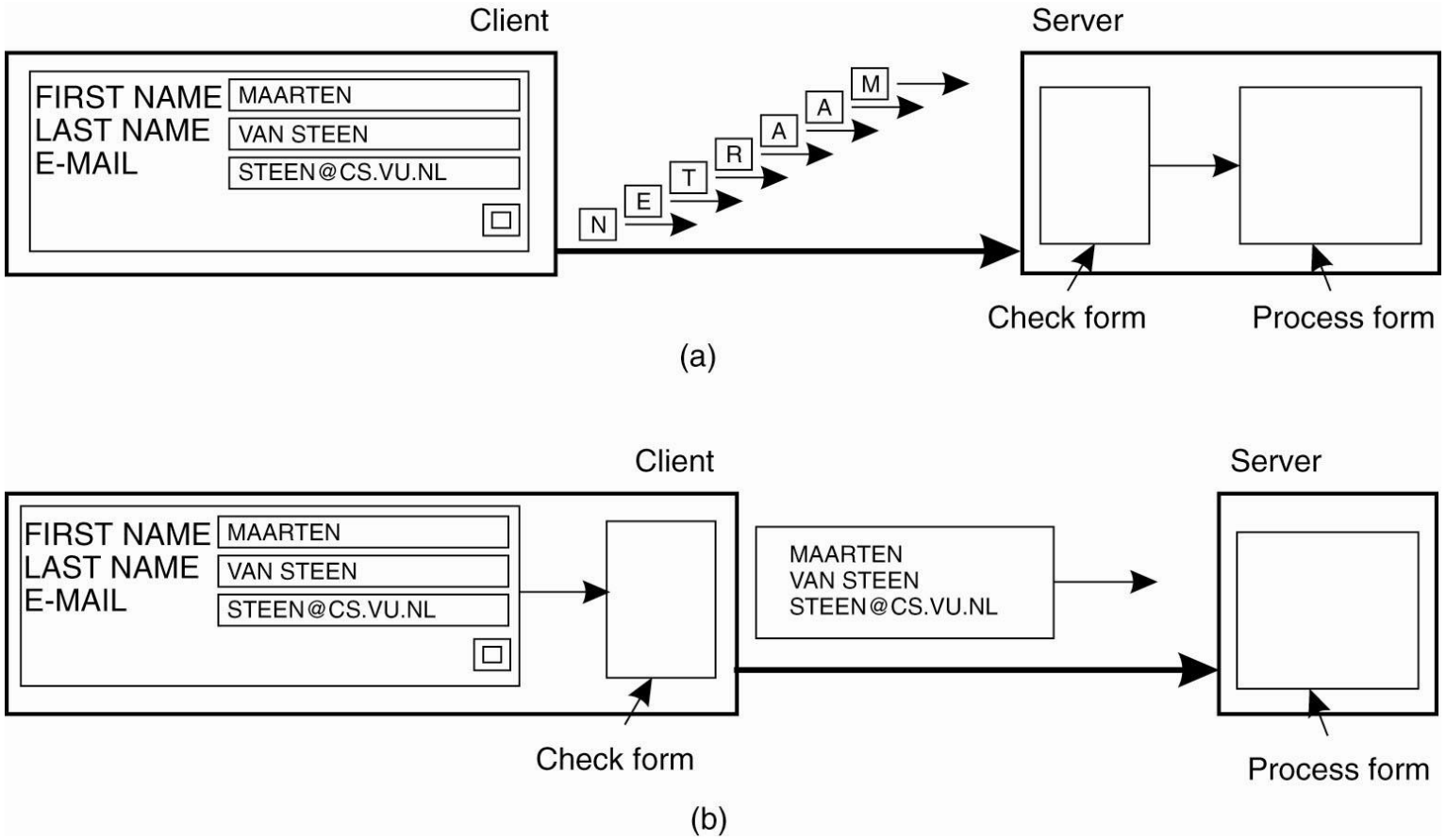


Figure 1-4. The difference between letting (a) a server or (b) a client check forms as they are being filled.

B- distribution

Distribution involves taking a component, splitting it into smaller parts, and subsequently spreading those parts across the system.

Ex.: the Internet Domain Name System (DNS). The DNS name space is hierarchically organized into a tree of domains, which are divided into nonoverlapping zones. The names in each zone are handled by a single name server.

Scaling Techniques (2)

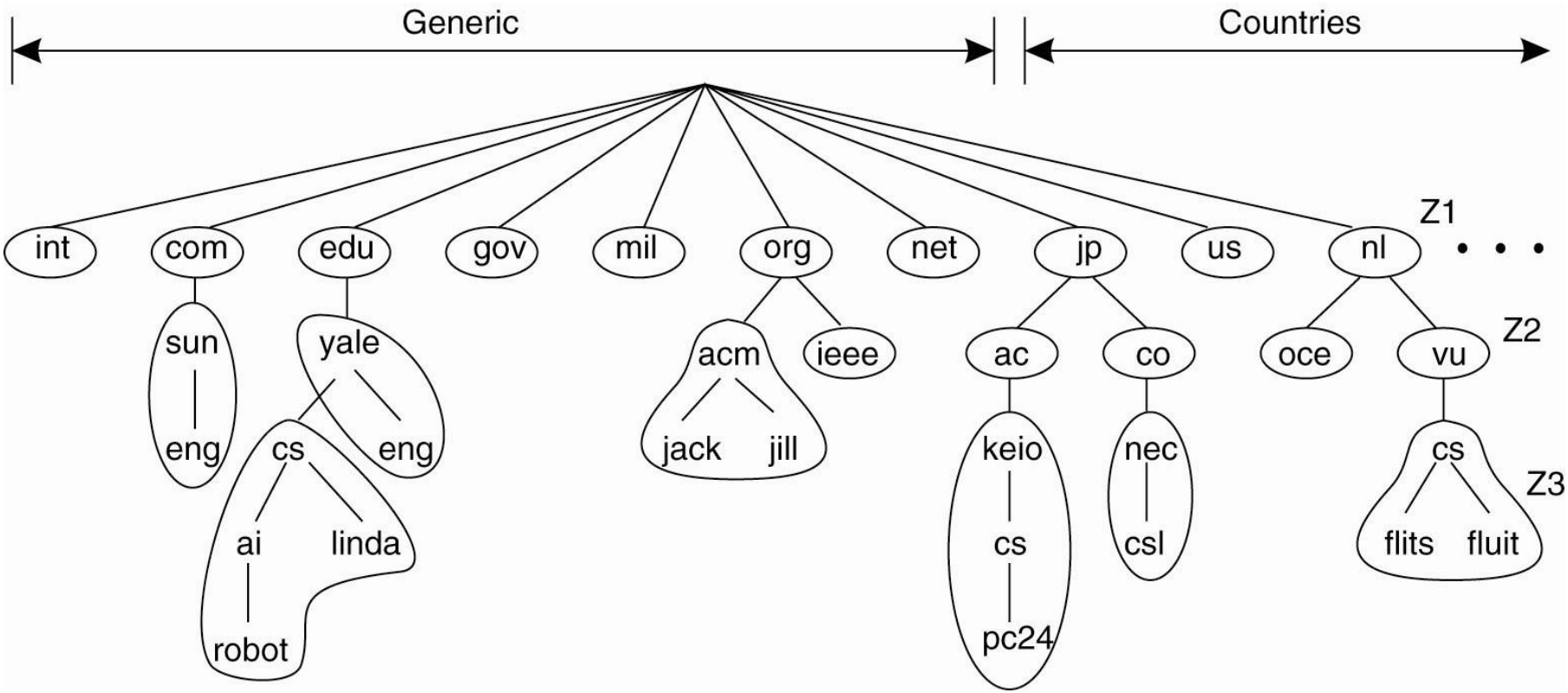
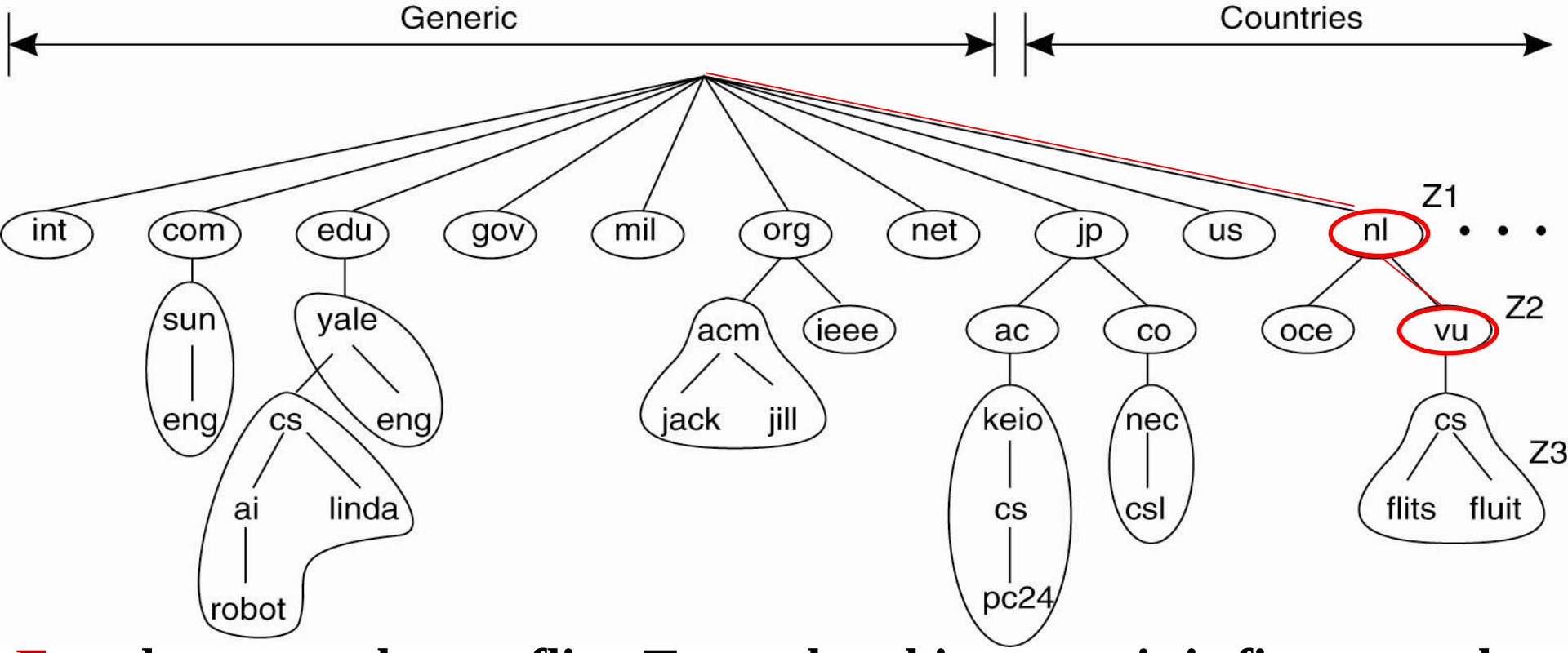


Figure 1-5. An example of dividing the DNS name space into zones.

Basically, resolving a name means returning the network address of the associated host.



Ex.: the name nl.vu.cs.flits. To resolve this name, it is first passed to the server of zone Z1 which returns the address of the server for zone Z2, to which the rest of name, vu.cs.flits, can be handed. The server for Z2 will return the address of the server for zone Z3, which is capable of handling the last part of the name and will return the address of the associated host.

Ex.: consider the World Wide Web. 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.

scalability problems often appear in the form of performance degradation,

a good idea is to replicate components across a distributed system.

C- Replication

- increases availability, and

- helps to balance the load between components

-hide much of the communication latency problems

leading to better performance.

Caching is a special form of replication,

caching is making a copy of a resource, generally in the proximity (near to) 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.

C- Replication

drawback to caching and replication:

having multiple copies of a resource, modifying one copy makes that copy different from the others. Consequently, caching and replication **leads to consistency problems**.

Data consistency: An instance of data in one resource in a distributed system is said to be "consistent" with one or more other instances of that data elsewhere in the system if it is up-to-date with respect to those instances.