

Dr. Srikanth Prabhu

Dept. of CSE MIT Manipal

DISTRIBUTED SYSTEMS

Module-1

INTRODUCTION

Advances in computer networks

- Local-area networks (LANs)
- Wide-area networks (WANs)
- Computing systems are grouped using LANs and WANs.
- The computers are geographically distributed and they form a distributed system.

What is a Distributed System ?

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

Computing element – Node which can be either a hardware device or a software process.

Characteristic 1: Collection of autonomous computing elements

- ➔ Nodes can act independently from each other.
- ➔ They are programmed to achieve common goals by exchanging messages with each other.
- ➔ Nodes will have their own notion of time and there is no **global clock** which leads to synchronization and coordination issue
- ➔ There is need to provide **group membership**.

Open group: Any node can join and communicate with other nodes.

Closed group: Only members of the group can communicate.

➔ Distributed system is often organized as an **overlay network** in which a node is typically a software process equipped with a list of other processes it can directly communicate.

Two types of overlay networks

1) Structured overlay: Each node has a well-defined set of neighbors with whom it can communicate. (Example: Tree)

2) Unstructured overlay: Each node has a number of references to randomly selected other nodes.

Characteristic 2: Single coherent system

There should be a **single-system view**.

The end users should not even notice that the processes, data, and control are dispersed across the network.

The distributed system provides the means for components of a single distributed application to communicate with each other, but also to let different applications communicate.

Middleware and distributed systems

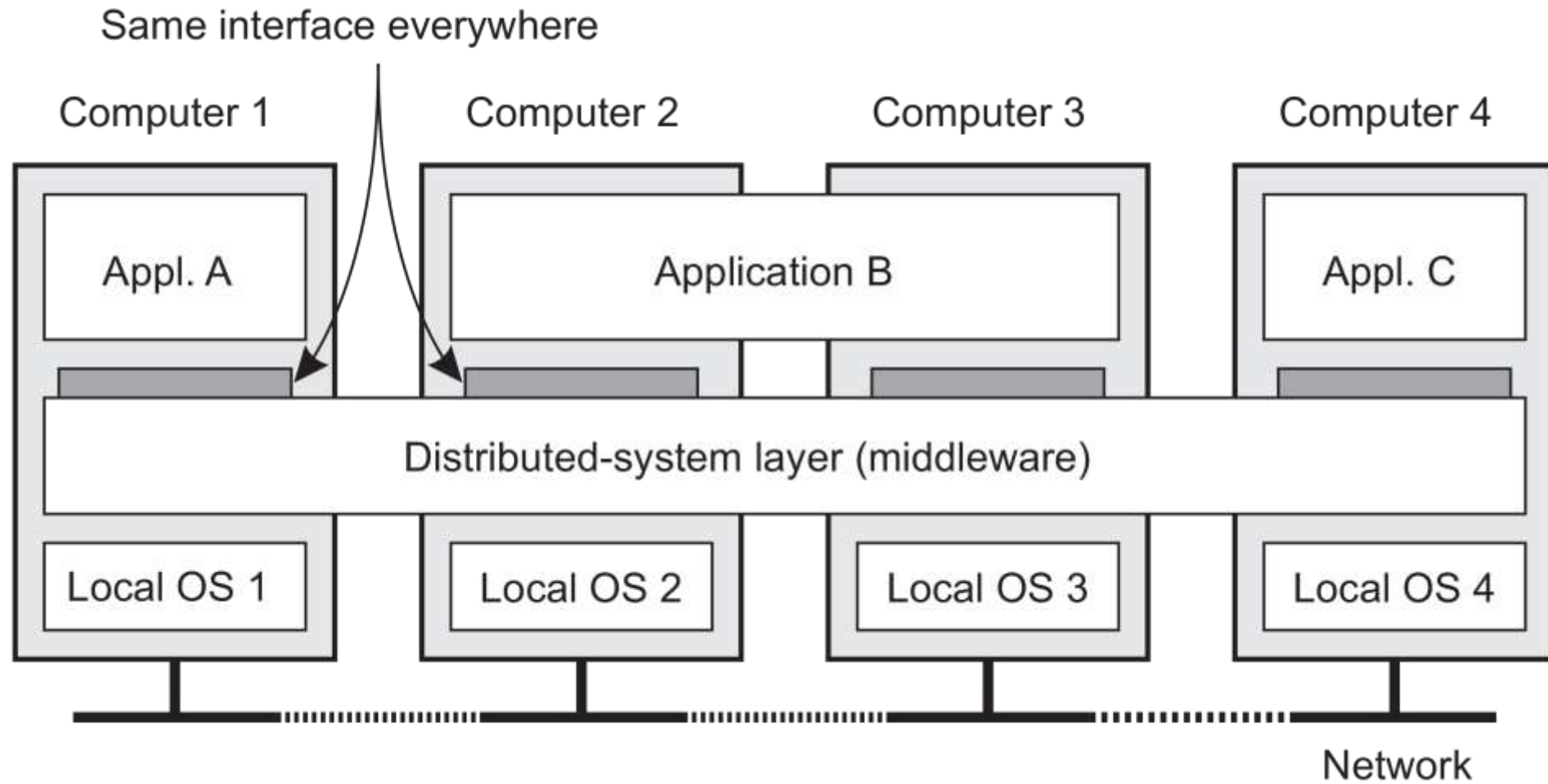


Figure 1-1. A distributed system organized as middleware. The middleware layer extends over multiple machines, and offers each application the same interface.

Middleware is the same to a distributed system as what an operating system is to a computer: a manager of resources offering its applications to efficiently share and deploy those resources across a network.

Middleware ↔ Distributed System Operating System ↔ Computer

Other services provided by middleware

- a) Facilities for interapplication communication.
- b) Security services.
- c) Accounting services.
- d) Masking of and recovery from failures.


Typical middleware services

Communication: Remote Procedure Call (RPC) allows an application to invoke a function present on a remote computer as if it was locally available.

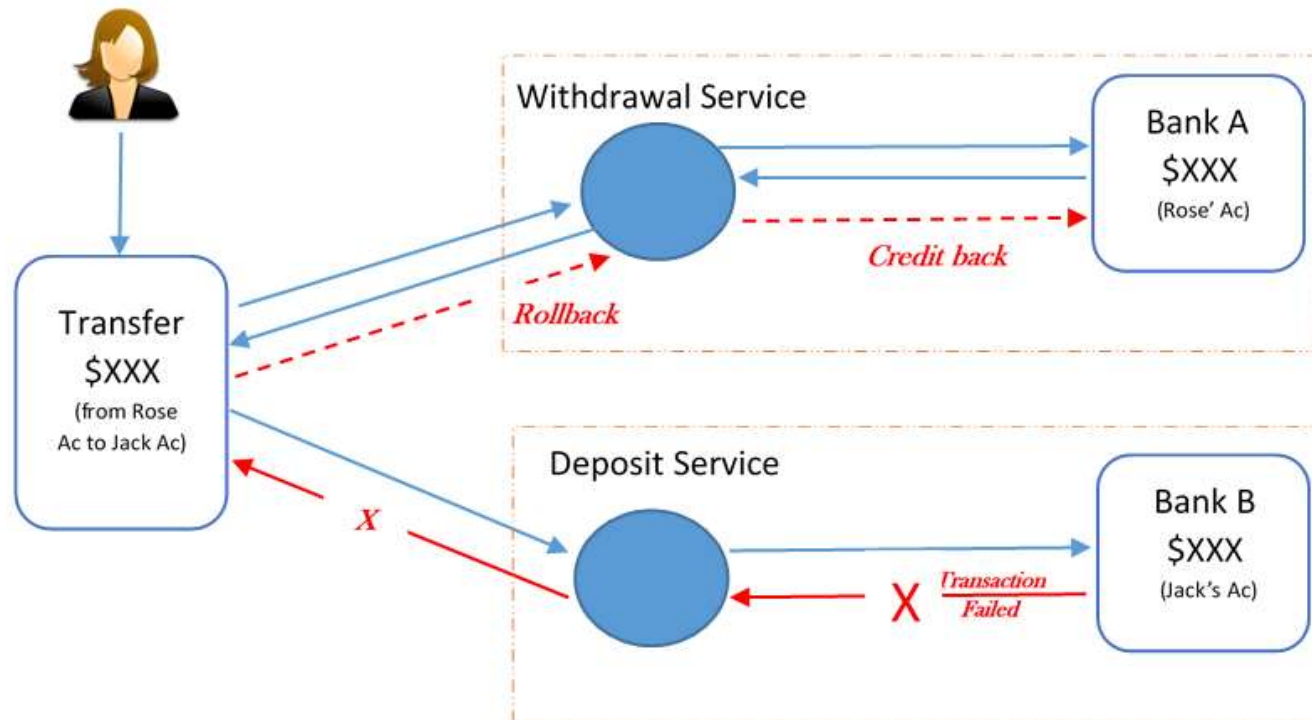
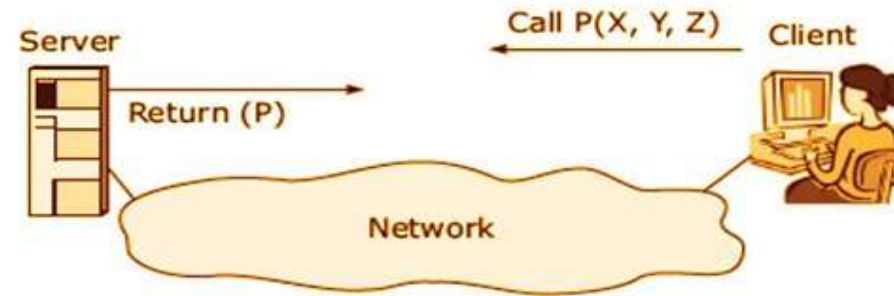
Transactions: Applications use multiple services that are distributed among several computers. Middleware offers special support for executing such services in an all-or-nothing fashion, commonly referred to as an **atomic transaction**.

Service composition: Developing new applications by taking existing programs and gluing them together. Example: **Mashups**-Web pages that combine and aggregate data from different sources.

Reliability: Providing functions for building reliable distributed applications.

Example: Sender  Received by all or no other process

Remote Procedure Call (RPC)



Design goals

There are 4 important design goals that should be met while building a distributed system.

1) Supporting resource sharing

It should be easy for users (and applications) to access and share remote resources, such as storage facilities, data, files, services and networks.

Example: It is cheaper to have a single high-end reliable storage facility be shared than having to buy and maintain storage for each user separately.

2) Making distribution transparent

An important goal of a distributed system is to **hide the fact that its processes and resources are physically distributed across multiple computers** possibly separated by large distances. In other words, it tries to make the distribution of processes and resources transparent, that is, invisible, to end users and applications.

Types of distribution transparency

The concept of transparency can be applied to several aspects of a distributed system as listed in Figure 1.2. Here object is either a process or a resource.

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

Figure 1.2: Different forms of transparency in a distributed system (see ISO [1995]). An object can be a resource or a process.

Access transparency: The computers in a distributed systems may run different operating systems, each having their own file-naming conventions.

Differences in naming conventions, differences in file operations, or differences in how low-level communication with other processes is to take place, are examples of access issues that should preferably be hidden from users and applications.

Location transparency: Refers to the fact that users cannot tell where an object is physically located in the system.

Naming can be used to achieve location transparency.

Logical names are assigned to resources in which the location of a resource is not secretly encoded

Example of a logical name:

Uniform Resource Locator (URL) <http://www.prenhall.com/index.html>, which gives no clue about the **actual location** of Prentice Hall's main Web server.

Relocation transparency: The URL also gives no clue as to whether the file index.html has always been at its current location or was recently moved there.

For example, the entire site may have been moved from one data center to another, yet users should not notice.

Migration transparency: Offered by a distributed system when it supports the mobility of processes and resources initiated by users, without affecting ongoing communication and operations.

Example:

- 1) Communication between mobile phones - regardless whether two people are actually moving, mobile phones will allow them to continue their conversation.
- 2) Online tracking and tracing of goods as they are being transported from one place to another.

Replication transparency:

Why replication ? : Resources may be replicated to increase availability or to improve performance by placing a copy close to the place where it is accessed

Replication transparency deals with hiding the fact that several copies of a resource exist.

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

Concurrency transparency: Two independent users may each have stored their files on the same file server or may be accessing the same tables in a shared database. In such cases, it is important that each user does not notice that the other is making use of the same resource.

Issue: Concurrent access to a shared resource should leave that resource in a consistent state.

Consistency can be achieved through locking mechanisms.

Failure transparency: A user or application does not notice that some piece of the system fails to work properly, and that the system subsequently (and automatically) recovers from that failure.

Masking and transparently recovering from failures is **difficult** as its difficult to **distinguish between** a **dead process** and a **slowly responding process**.

Example:

While contacting a busy web server unavailability of a web page can be due to server failure or network congestion.

Degree of distribution transparency:

Distribution transparency is essential but may not good at all the times.

Example:

- a) Requesting for a e-news paper from different time zone.
- b) Replicas in different continents must be consistent all the time.

Change in a copy should be propagated to all copies before other operation.

This update operation may take seconds which cannot be hidden from users.

3) Being open

An open distributed system is essentially a system that offers components that can easily be used by, or integrated into other systems.

Interoperability, composability, and extensibility

To be open means that components should adhere to standard rules that describe the syntax and semantics of what those components have to offer (i.e., which service they provide).

Interoperability: Characterizes the extent by which two implementations of systems or components from different manufacturers can co-exist and work together.

Composability: Portability

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

Extensibility: It should be easy to add new components or replace existing ones.

4) Being scalable

For a distributed system addition and removal of nodes should be an easy task.

Scalability is measured along three different dimensions:

Size scalability: A system can be scalable with respect to its size, meaning that we can easily add more users and resources to the system without any noticeable loss of performance.

Geographical scalability: A geographically scalable system is one in which the users and resources may lie far apart.

Administrative scalability: An administratively scalable system is one that can still be easily managed even if it spans many independent administrative organizations.

Size scalability: (Why it is difficult to achieve ?)

When a system needs to scale different types of problems need to be solved.

If more users or resources need to be supported, we are often confronted with the limitations of centralized services.

For example: If many services are implemented on a single server running on a specific machine in the distributed system, the server can simply become a bottleneck when it needs to process an increasing number of requests.

Three root causes for becoming a bottleneck:

- 1) The computational capacity, limited by the CPUs
- 2) The storage capacity, including the I/O transfer rate
- 3) The network between the user and the centralized service

Geographical scalability: (Why it is difficult to achieve ?)

One of the main reasons why it is still difficult to scale existing distributed systems that were designed for LANs is that many of them are based on **synchronous communication**.

A party requesting service, generally referred to as a client, **blocks** until a reply is sent back from the server implementing the service.

In a wide area system it is difficult to achieve **synchronous communication** across the nodes.

Communication in WANs is less reliable than in LANs.

Administrative scalability: (Why it is difficult to achieve ?)

To be able to scale a Distributed System (DS) across multiple, independent administrative domains, we need to solve the **conflicting policies** with respect to **resource usage (and payment), management, and security**.

If a DS expands to another domain, **two types of security measures** need to be taken.

- 1) DS has to protect itself against malicious attacks from the new domain.
- 2) The new domain has to protect itself against malicious attacks from the DS.

Scaling techniques (To solve some of the scalability problems)

Generally the performance problems of servers can be solved by increasing their memory and upgrading CPUs. This is usually termed as **scaling up**.

Scaling out: Expanding the DS by essentially deploying more machines.

For performing scaling out, we can apply only 3 basic techniques:

- 1) Hiding communication latencies
- 2) Distribution of work
- 3) Replication

1) Hiding communication latencies (Latency – Total delay experienced)

This is applicable in the case of geographical scalability.

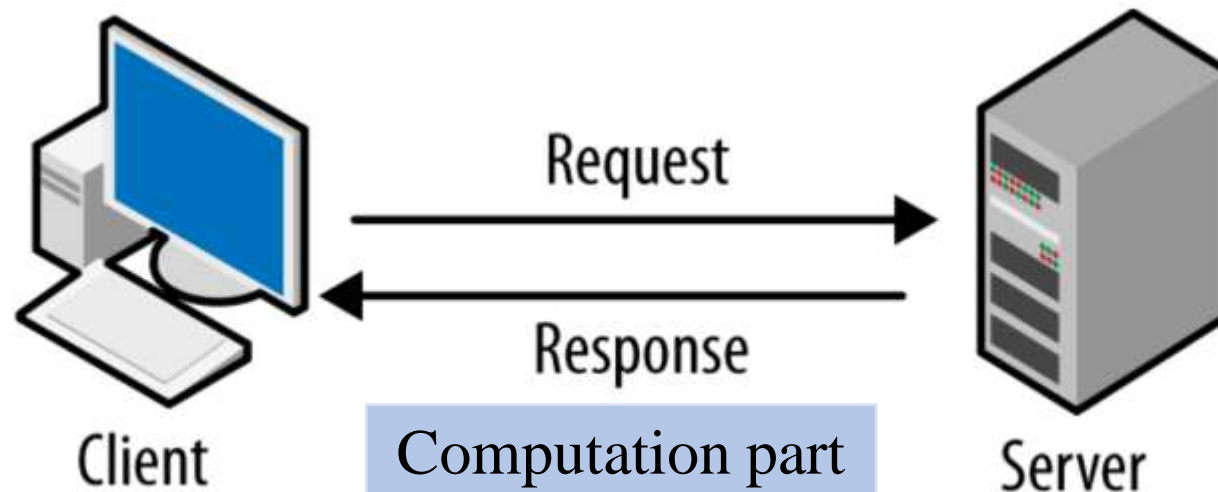
Basic idea: Avoid waiting for responses to remote-service requests.

Example: When a service has been requested at a remote machine, an alternative to waiting for a reply from the server is to do other useful work at the client side. Essentially, this means that we should construct the client in such a way that it uses only **asynchronous communication**.

However some applications cannot make use of asynchronous communication.

Example: Interactive applications – Client has no other work.

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.



A typical case where this approach works is accessing **databases using forms**.

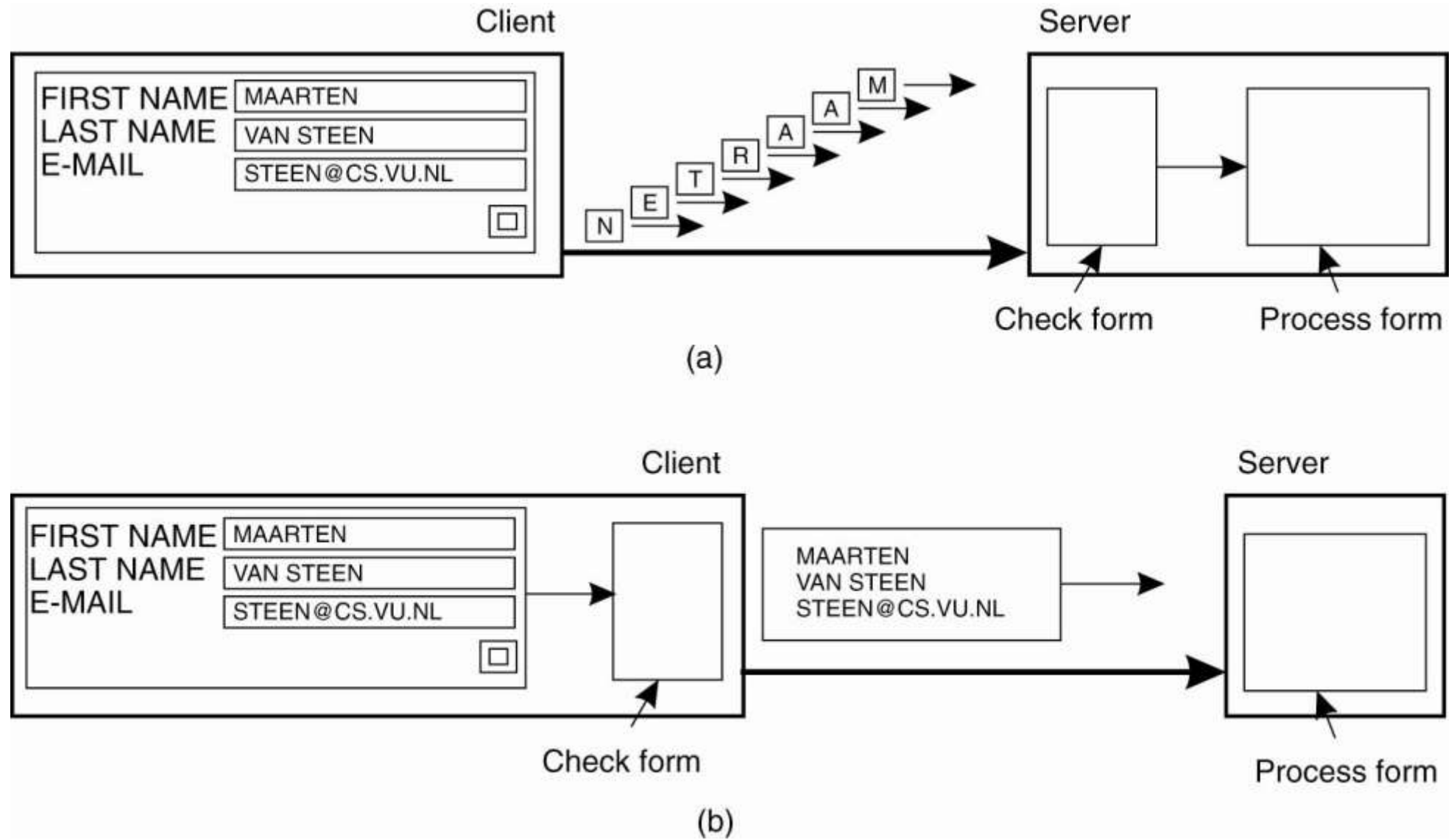


Figure 1-4. The difference between letting
(a) a server or (b) a client check forms as they are being filled.

2) Partitioning and distribution

A component is split into smaller parts and distributed across the system.

Example -1 : 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**.
- ➔ Each **path name** is a **name of the host** and **resolving a name** means getting the **network address of the host**.

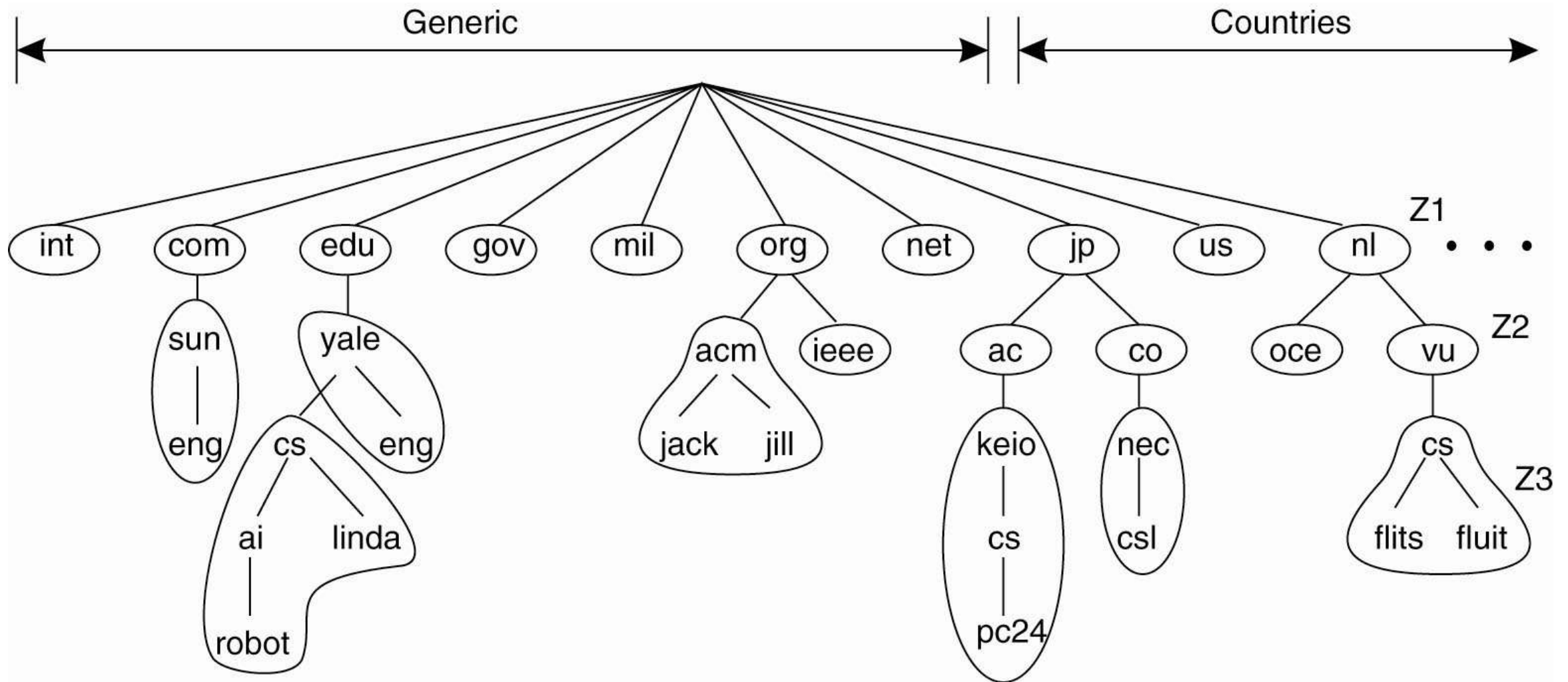


Figure 1-5. An example of dividing the (Original) DNS name space into zones.

Example: Name `flits.cs.vu.nl` will be resolved by servers of zones **Z1** → **Z2** → **Z3**

Example -2: World Wide Web (WWW)

- ➔ It appears as a collection of huge documents with URLs for each document.
- ➔ WWW may appear as a single server.
- ➔ However, the Web is physically partitioned and distributed across a few hundred million servers, each handling a number of Web documents.

3) Replication

- ➔ To address the scalability issue, it is good to **replicate the components** across a distributed system.
- ➔ Replication increases **availability** and helps to balance the load between components leading to **better performance**.
- ➔ In geographically widely dispersed systems, having a copy nearby can hide much of the communication **latency problems**
- ➔ **Caching:** A special form of replication in which a **copy of the resource** is created in the proximity of the client accessing that resource.
- ➔ **Drawback:** Multiple copies of resources may lead to **consistency problems**.

Pitfalls

False assumptions made by users while developing a DS:

The network is reliable, The network is secure, The network is homogeneous,
The topology does not change, Latency is zero, Bandwidth is infinite, Transport
cost is zero, There is one administrator.

These assumptions relate to properties that are unique to distributed systems:

reliability, security, heterogeneity, and topology of the network; latency and
bandwidth; transport costs; and finally administrative domains.