DISTRIBUTED SYSTEMS
1. Introduction
2. System Architectures

1. Introduction

Definition of a Distributed System

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

Definition of a Distributed System

Several important aspects:

1. Consists of components that are autonomous.
2. Users think they are dealing with a single system.
3. No assumptions made regarding the types of computers or the way they are interconnected.

Definition of a Distributed System

Distributed systems are often organized by means of a layer of software -

- logically placed between a higher layer consisting of users and applications, and a lower layer consisting of operating systems and basic communication facilities
- **middleware.**

Definition of a Distributed System

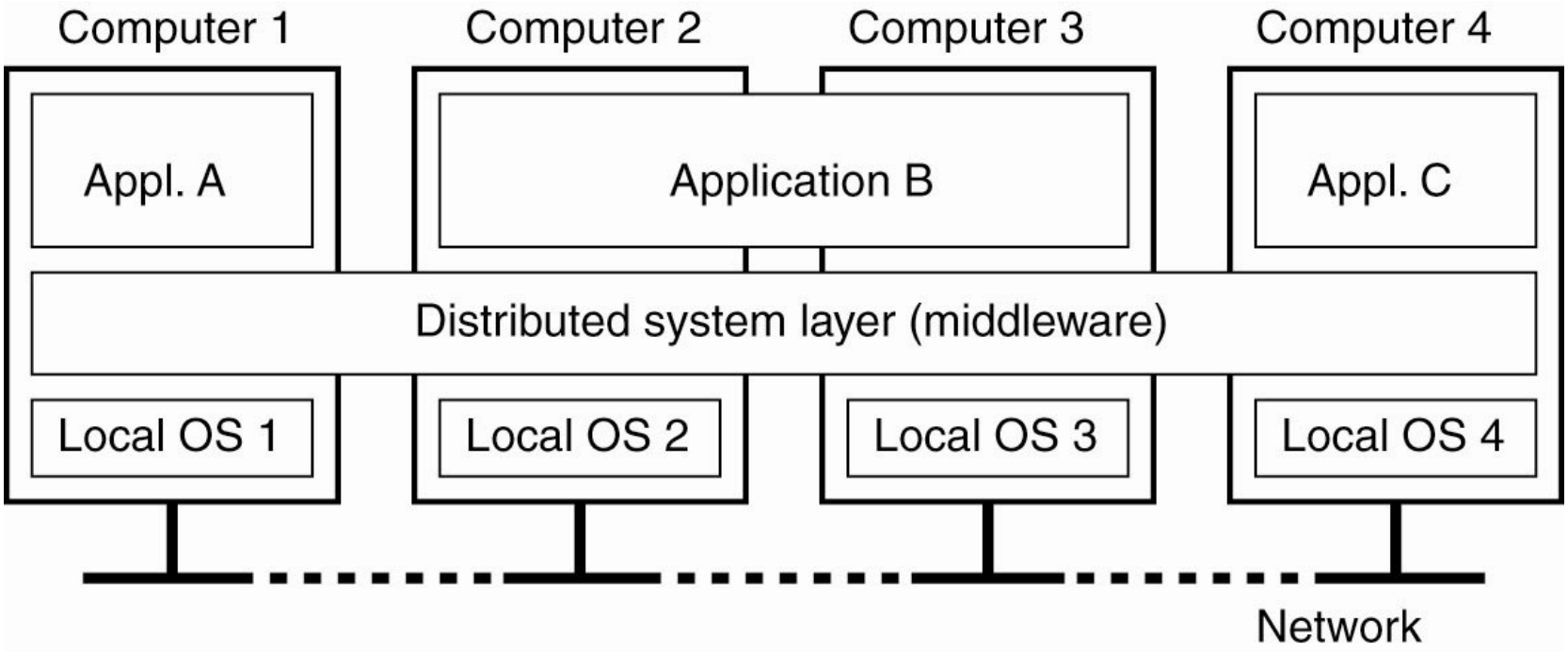

Figure 1-1. A distributed system organized as middleware. The *middleware* layer extends over multiple machines, and offers each application the same interface.

Goals of Distributed Systems

- Making resources accessible
- Distribution transparency
- Openness
- Scalability

Making Resources Accessible

- Making it easy for users and applications to access remote resources
- Share remote resources in a controlled and efficient manner

Making Resources Accessible

- Benefits of sharing remote resources
  - Better economics by sharing expensive resources
  - Easier to collaborate and exchange information
  - Connectivity of the Internet has lead to numerous virtual organizations where geographically dispersed people can work together using ***groupware***
  - Connectivity has enabled *electronic commerce*
  - However, as connectivity and sharing increase …
- Security problems
  - Eavesdropping or intrusion on communication
  - Tracking of communication to build up a preference profile of a specific user

Distribution Transparency

An important goal - hide the fact that the processes and resources are physically distributed across multiple computers.

Transparent - A distributed system that presents itself to users and applications as if it were only a single computer system.

Types of Transparency

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).

Types of Transparency

Access transparency - hide differences in data representation and the way the resources are accessed

Ex: a distributed system may have computer systems that run different operating systems, each having their own file-naming conventions.

Types of Transparency

Location transparency - users cannot tell where a resource is physically located in the system. Achieved by assigning only logical names to resources.

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

Migration transparency - resources can be moved without affecting how those resources can be accessed.

Types of Transparency

Relocation transparency - resources can be relocated *while* they are being accessed without the user or application noticing anything.

Ex: when mobile users can continue to use their wireless laptops while moving from place to place.

Replication transparency - hide the fact that several copies of a resource exist.

Types of Transparency

Sharing of resources can also be done in a competitive way.

Ex: 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.

This phenomenon is called concurrency transparency.

Failure transparency - a user does not notice that a resource fails to work properly, and that the system subsequently recovers from that failure.

Degree of Transparency

Complete hiding the distribution aspects from users is not always a good idea.

- Attempting to mask a server failure before trying another one may slow down the system
- Requiring several replicas to be always consistent means a single update operation may take seconds to complete
- For mobile and embedded devices, it may be better to *expose* distribution rather than trying to hide it
- Signal transmission is limited by the speed of light as well as the speed of intermediate switches.

Openness

An *open* distributed system offers services according to standard rules that describe the syntax and semantics of those services.

- Services are generally specified through *interfaces*, which are often described in an *Interface Definition Language (IDL).*

Openness

- An interface definition - allows an arbitrary process that needs a certain interface to talk to another process that provides that interface.
  allows two independent parties to build completely different implementations of those interfaces.
- Proper specifications are complete and neutral.
- Completeness and neutrality are important for interoperability and portability.

Openness

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

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.*

Openness

Extensibility

- It should be easy to configure the system out of different components
- It should be easy to add new components or replace existing ones.

Scalability

Scalability can be measured against three dimensions.

- *Size*: be able to easily add more users and resources to a system
- *Geography*: be able to handle users and resources that are far apart
- *Administrative*: be easy to manage even if it spans many independent administrative organizations

Scalability Problems

Consider scaling w.r.t. size - we are often confronted with the limitations of centralized services, data and algorithms.

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

Figure 1-3. Examples of scalability limitations.

Scalability Problems

Only decentralized algorithms should be used.

Characteristics of *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 global clock exists

Scalability Problems

LANs use synchronous communication. Designing WANs using synchronous communication is much more difficult

Communication in WANs is inherently unreliable, and virtually always point-to-point. LANs use broadcasting.
Ex: this makes it very easy to locate a service.

Scaling across multiple, independent administrative domains leads to conflicting policies w.r.t. resource usage, payment, management and security.

Scalability Problems

Security issues:

Many components of a distributed system that resides within a single domain, may not be trusted by users in other domains.

Scaling Techniques

How can the scalability problems be solved?

Three techniques for scaling:

- Hiding communication latencies
- Distribution
- Replication

Scaling Techniques

Hiding communication latencies:

Basic idea: Try to avoid waiting for responses to remote service requests as much as possible.

In applications that cannot make effective use of asynchronous communication, a better solution is to reduce the overall communication.

Scaling Techniques

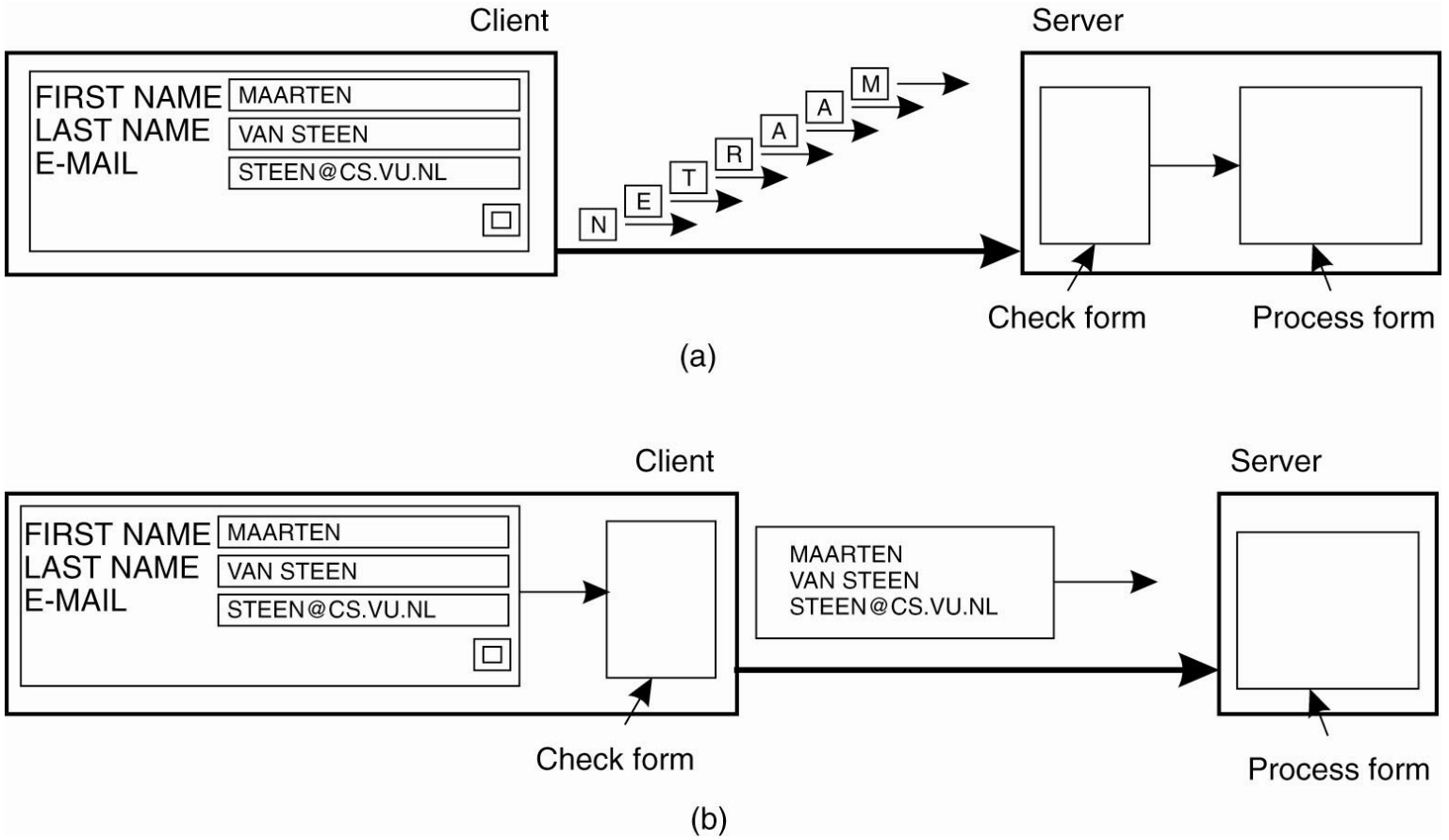

Figure 1-4. The difference between letting (a) a server or (b) a client check forms as they are being filled.

Scaling Techniques

- *Distribution*: Taking a component, splitting into smaller parts, and subsequently spreading them across the system.
  Ex: the Internet Domain Name System (DNS).
  - The DNS namespace is hierarchically organized into a tree of **domains,** which are divided into nonoverlapping **zones**.

Scaling Techniques

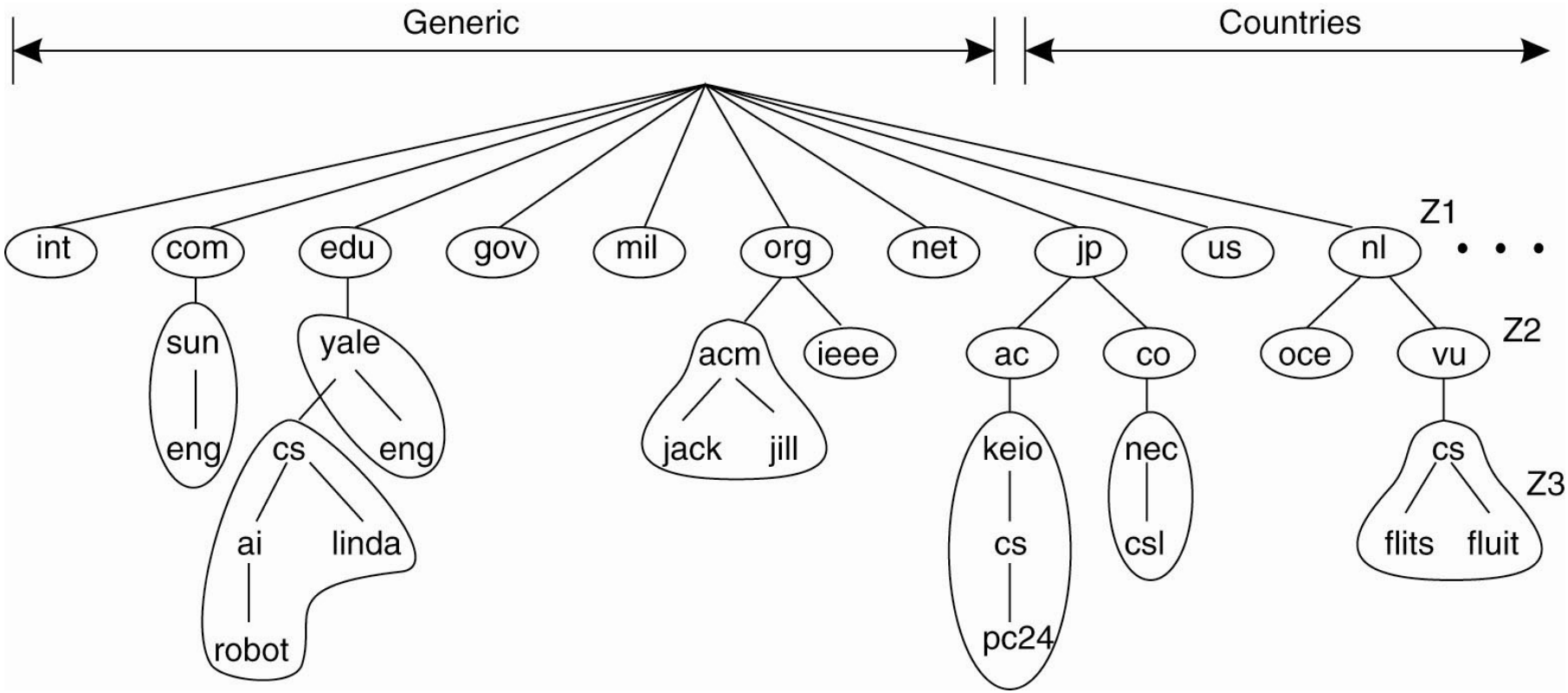

Figure 1-5. An example of dividing the DNS name space into zones.

Scaling Techniques

- *Replication*: increases availability and helps balance the load between components leading to better performance.
- *Caching:* special form of replication - making a copy of the resource, generally in the proximity of the client accessing that resource.

Scaling Techniques

One serious drawback to caching and replication - consistency problems.

Scaling Techniques

Size scalability - least problematic from a technical point of view.

Geographical scalability is a much tougher problem

Administrative scalability is the most difficult one, partly also because we need to solve nontechnical problems

Pitfalls when Developing Distributed Systems

False assumptions made by first time developer:

- 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