Distributed Systems

Lesson 6
Processes, Communication, Naming

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

Lesson 6

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

PART I - Processes

- Threads
- Virtualization
- Clients
- Servers
- Migration

Processes

- The concept of a process is fundamental in the field of operating systems where it is generally defined as a program in execution.

- From an operating-system perspective, the management and scheduling of processes are perhaps the most important issues to deal with.

- However, when it comes to distributed systems, other issues turn out to be equally or more important.

Concurrency transparency

- An important issue is that the operating system takes great care to ensure that:

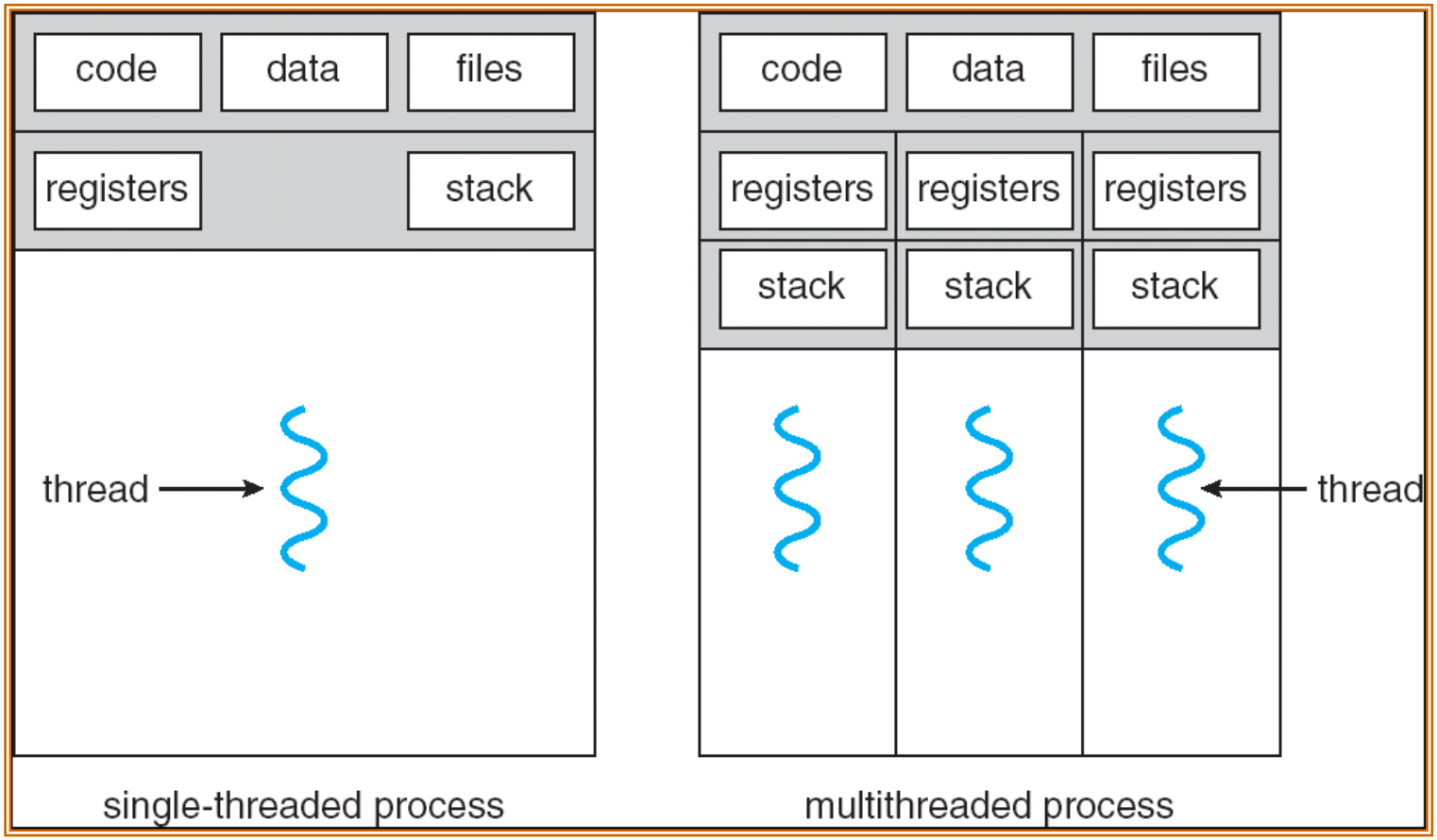
  independent processes cannot maliciously or inadvertently affect the correctness of each other's behavior.

- In other words, the fact that multiple processes may be concurrently sharing the same CPU and other hardware resources is made transparent.

Processes and threads

- We build virtual processors in software, on top of physical processors:
  - Processor: Provides a set of instructions along with the capability of automatically executing a series of those instructions.
  - Thread: A minimal software processor in whose context a series of instructions can be executed.
  - Process: A software processor in whose context one or more threads may be executed. Executing a thread, means executing a series of instructions in the context of that thread.

Single and Multithreaded Processes



Multithreading for large applications

- Multithreading is also useful in the context of large applications.
- Such applications are often developed as a collection of cooperating programs, each to be executed by a separate process.
- Cooperation is typical for a UNIX environment.
  - Cooperation between programs is implemented by means of interprocess communication (IPC) mechanisms.
  - The major drawback of all IPC mechanisms is that communication often requires extensive context switching.

Threads and Distributed Systems: blocking system calls!

- An important property of threads is that they can provide a convenient means of allowing blocking system calls without blocking the entire process in which the thread is running.

- This property makes threads particularly attractive to use in distributed systems as it makes it much easier to express communication in the form of maintaining multiple logical connections at the same time.

- We illustrate this point by taking a closer look at multithreaded clients and servers.

Threads and Distributed Systems

Multithreaded Web client

- Hiding network latencies:
  - Web browser scans an incoming HTML page, and finds that more files are to be fetched.
    - Each file is fetched by a separate thread, each doing a (blocking) HTTP request.
    - As files come in, the browser displays them.

Multithreaded Web browsers

- If several connections are set up to the same server and the server is heavily loaded, or just plain slow, no real performance improvements will be noticed compared to pulling in the files that make up the page strictly one after the other.
- However, in many cases, Web servers have been replicated across multiple machines, where each server provides exactly the same set of Web documents.
- When using a multithreaded client, connections may be set up to different replicas, allowing data to be transferred in parallel,
  - web document are fully displayed in a much shorter time than with a nonreplicated server.

Multithreaded clients

Multiple request-response calls to other machines (RPC)

- A client does several calls at the same time, each one by a different thread.
- It then waits until all results have been returned.
- Note: if calls are to different servers, we may have a linear speed-up.

- This approach is possible only if the client can handle truly parallel streams of incoming data.
- Threads are ideal for this purpose.

Multithreaded Servers

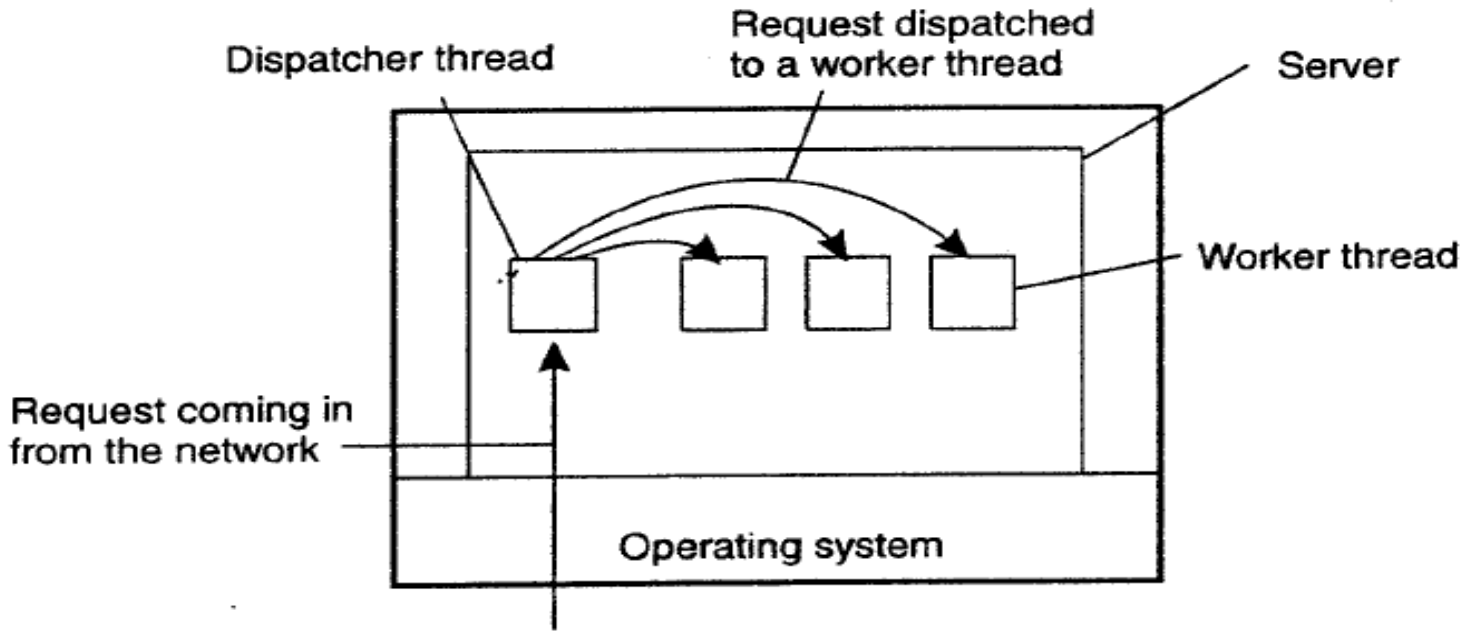


Figure 3-3. A multithreaded server organized in a dispatcher/worker model.

PART I - Processes

- Threads
- Virtualization
- Clients
- Servers
- Migration

Resource Virtualization

- Threads and processes can be seen as a way to do more things at the same time.
- In effect, they allow us to build programs that appear to be executed simultaneously.
- On a single-processor computer, this simultaneous execution is, of course, an illusion.
  - As there is only a single CPU, only an instruction from a single thread or process will be executed at a time.

Resource Virtualization

- By rapidly switching between threads and processes, the illusion of parallelism is created.

- This separation between having a single CPU and being able to pretend there are more can be extended to other resources as well, leading to what is known as resource virtualization.

Virtualization

- One of the most important reasons for introducing virtualization in the 1970s, was to allow legacy software to run on expensive mainframe hardware.
  - The software not only included various applications, but in fact also the operating systems they were developed for.

- This approach toward supporting legacy software has been successfully applied on the IBM 370 mainframes (and their successors) that offered a virtual machine to which different operating systems had been ported.

Virtualization: Why?

- Virtualization is becoming increasingly important:
  - Hardware changes faster than software
  - Ease of portability and code migration
  - Isolation of failing or attacked components

Virtualization

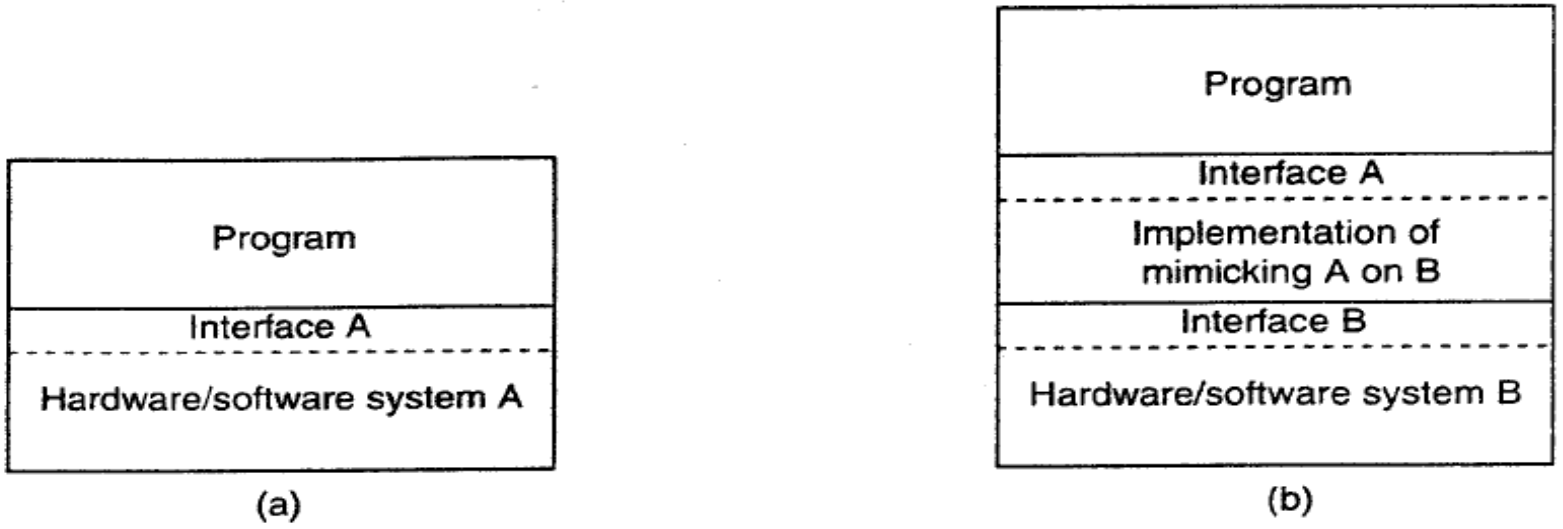
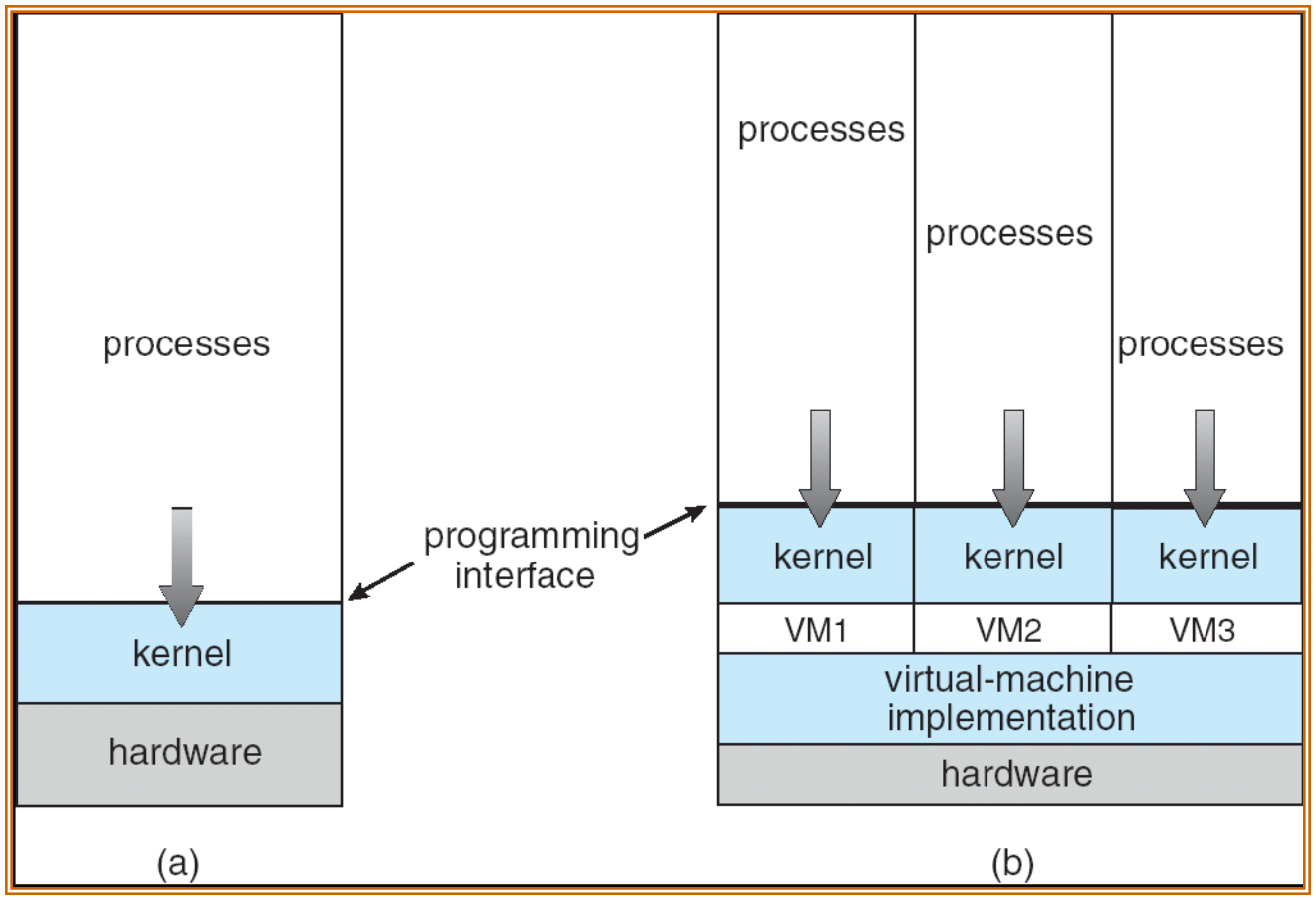


Figure 3-5. (a) General organization between a program, interface, and system. (b) General organization of virtualizing system A on top of system B.

Virtual Machines



Virtual Machines

- A Virtual Machine Monitor (or System virtual machine) provides a complete system platform which supports the execution of a complete operating system (OS).
  - VMware
  - Virtual PC (Microsoft)

- A process virtual machine is designed to run a single program, which means that it supports a single process.
  - This type of VM has become popular with the Java programming language, which is implemented using the Java virtual machine.
  - Another example is the .NET Framework, which runs on a VM called the Common Language Runtime.

Virtual Machines in Action

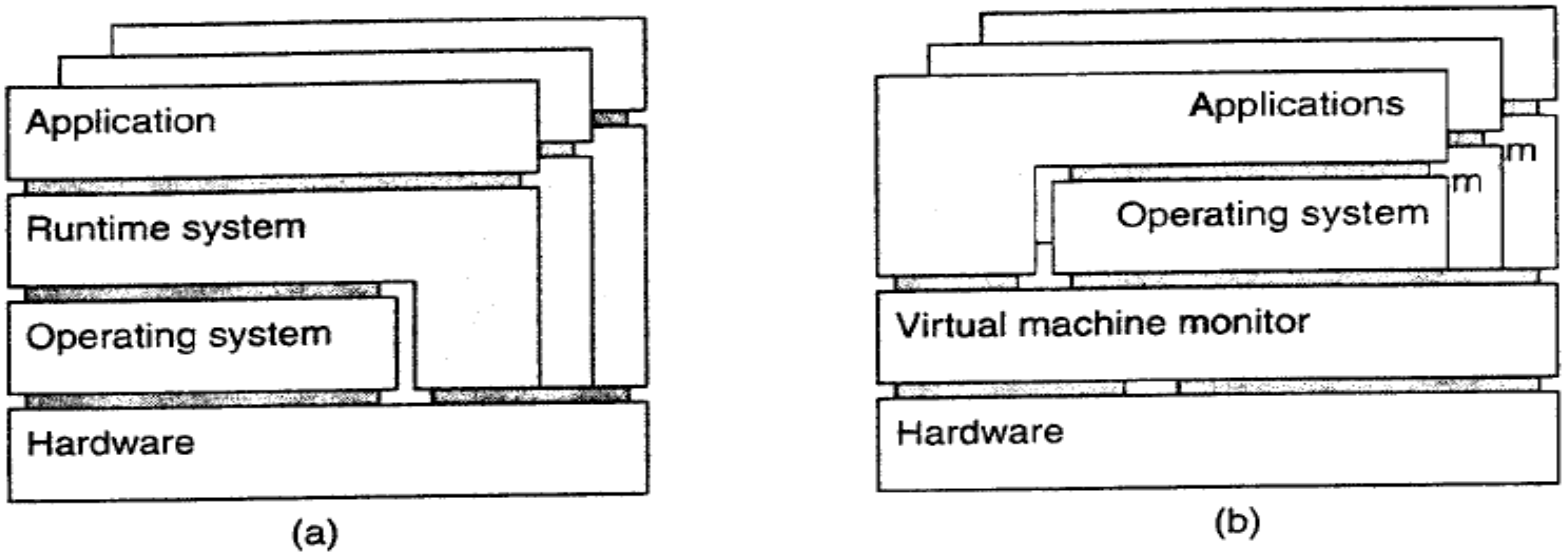
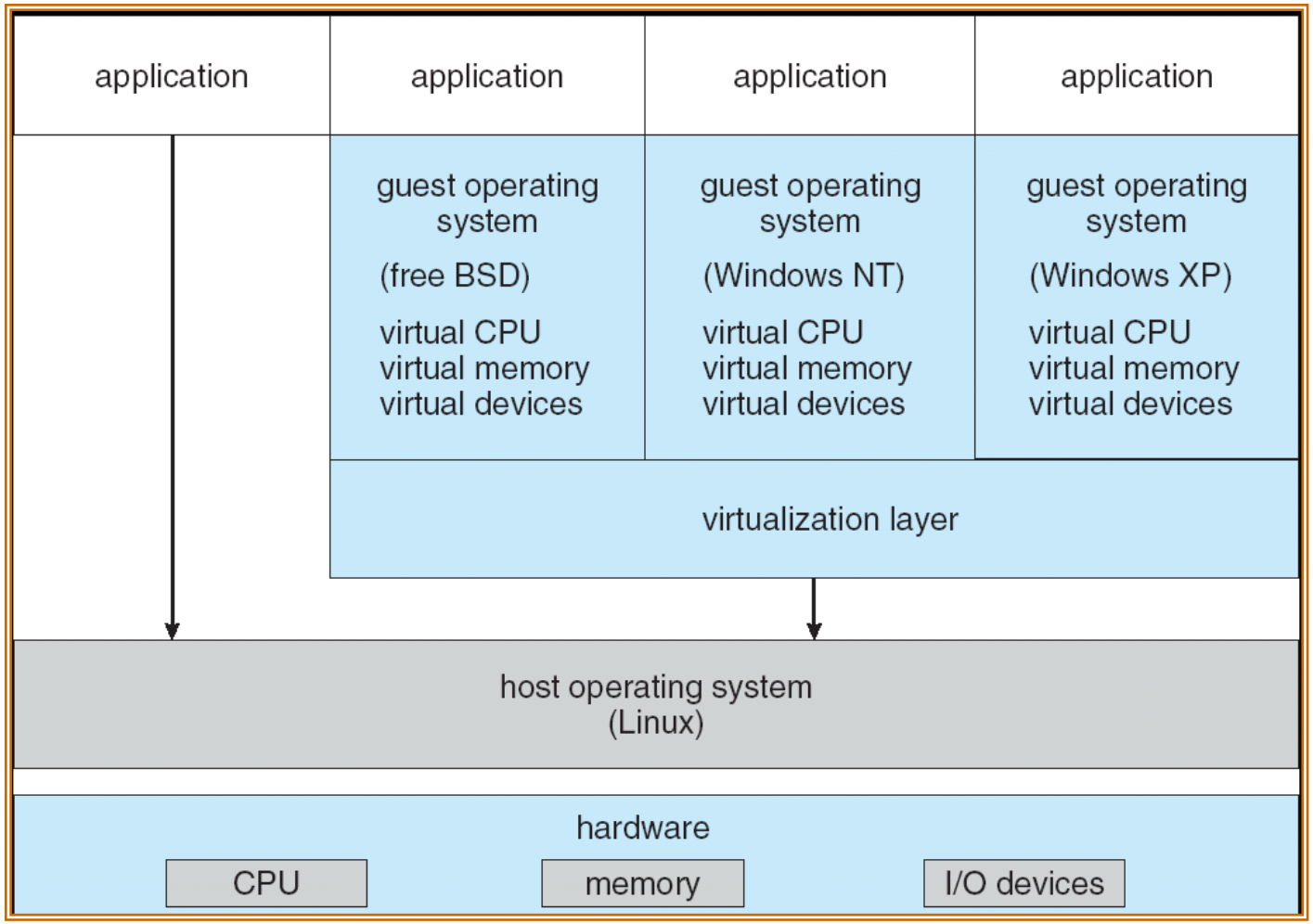
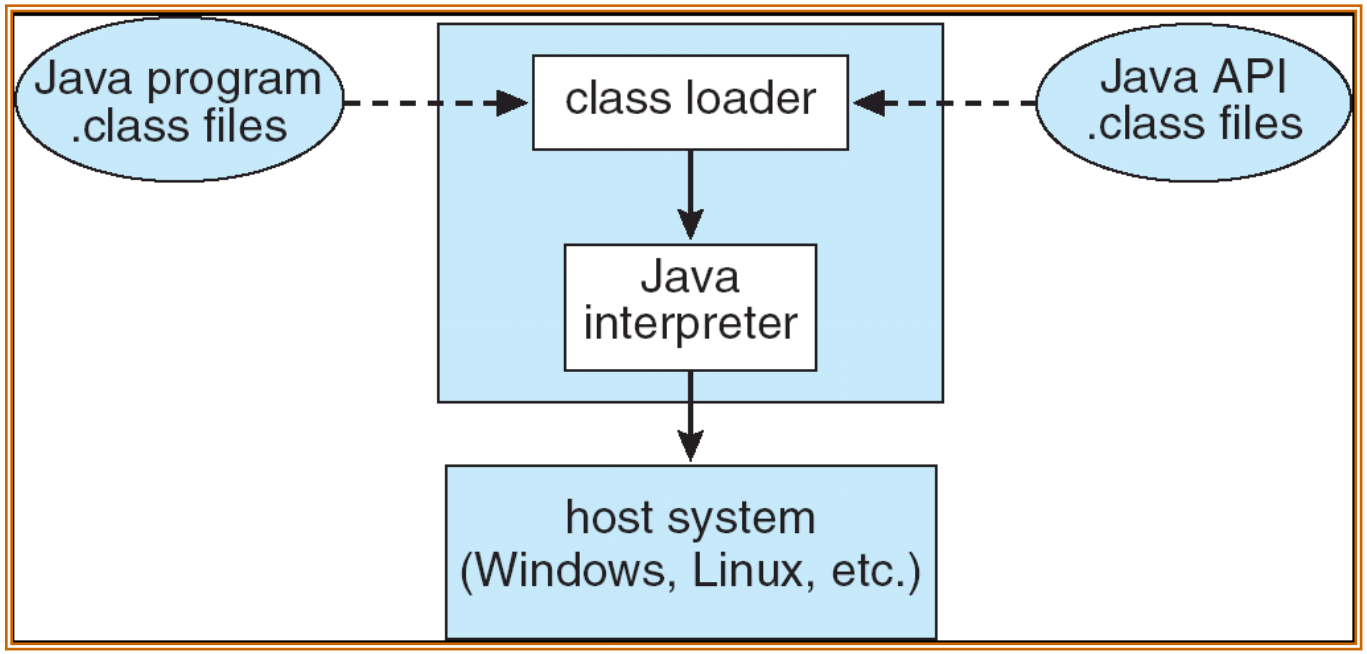


Figure 3-7. (a) A process virtual machine, with multiple instances of (application, runtime) combinations. (b) A virtual machine monitor, with multiple instances of (applications, operating system) combinations.

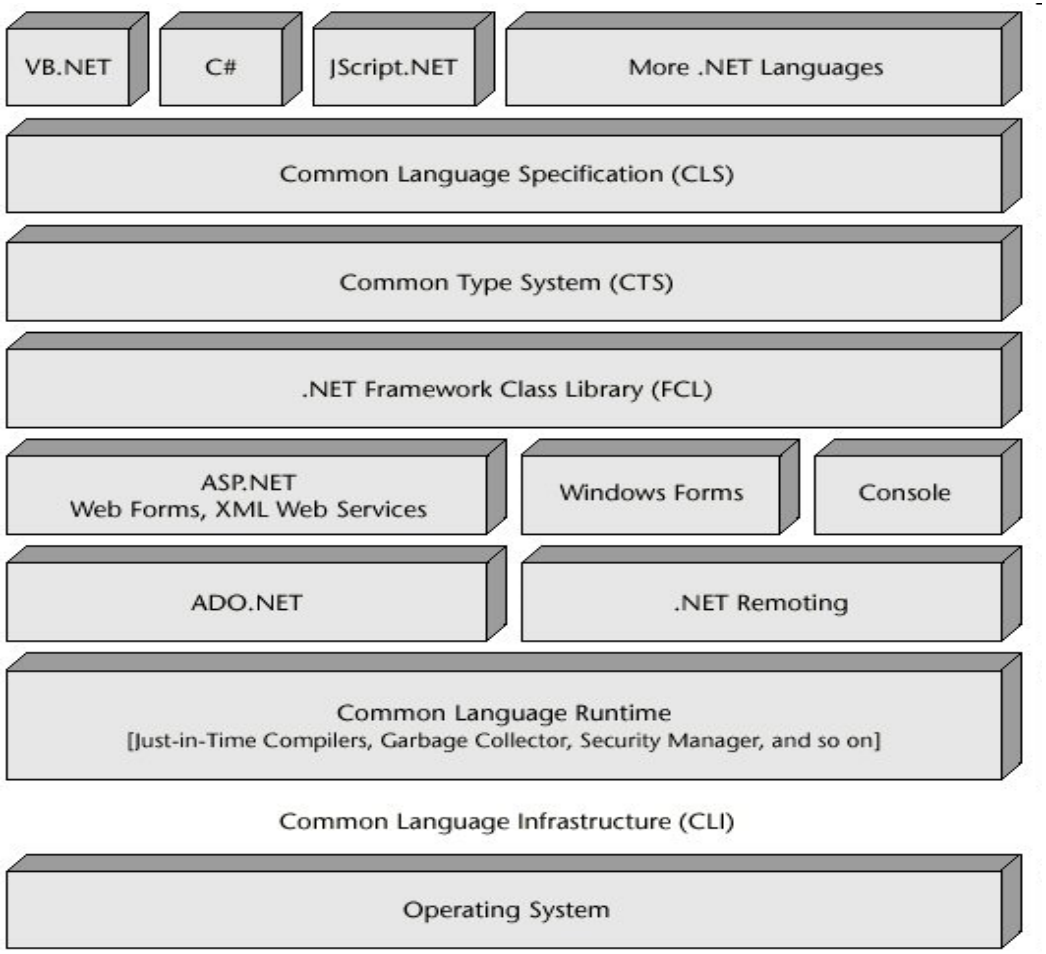
VMware Architecture



The Java Virtual Machine



.NET Architecture



Importance of VMMs

- VMMs will become increasingly important in the context of reliability and security for (distributed) systems.
- As they allow for the isolation of a complete application and its environment, a failure caused by an error or security attack need no longer affect a complete machine.
- In addition, as we also mentioned before, portability is greatly improved as VMMs provide a further decoupling between hardware and software, allowing a complete environment to be moved from one machine to another.

PART I - Processes

- Threads
- Virtualization
- Clients
- Servers
- Migration

Thin clients

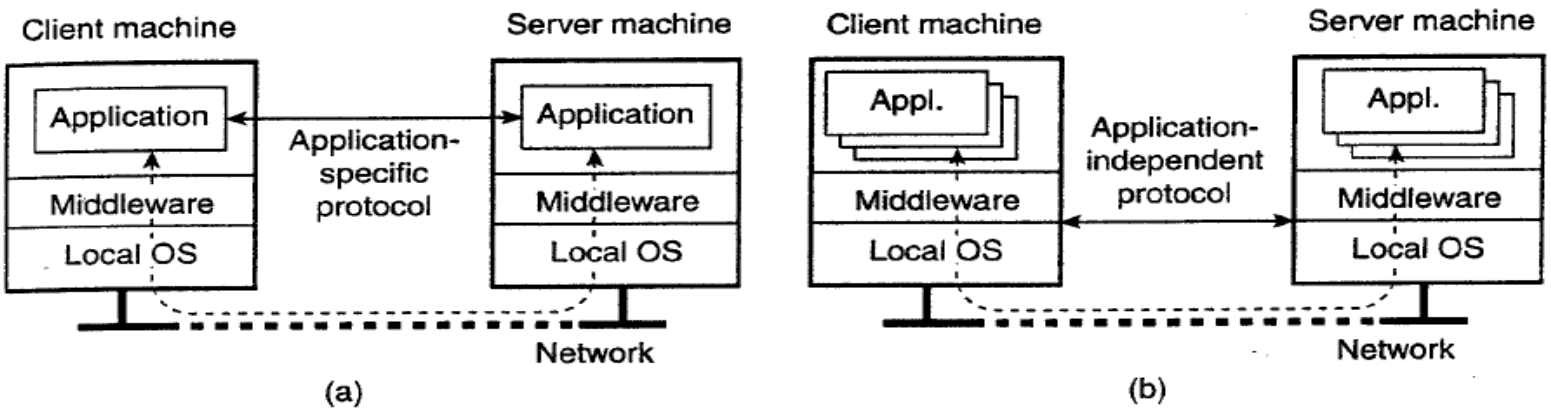


Figure 3-8. (a) A networked application with its own protocol.. (b) A general solution to allow access to remote applications.

In b) the client machine is used only as a terminal with no need for local storage, leading to an application neutral solution.

In the case of networked user interfaces, everything is processed and stored at the server.

This thin-client approach is receiving more attention as Internet connectivity increases, and hand-held devices are becoming more sophisticated.

PART I - Processes

- Threads
- Virtualization
- Clients
- Servers
- Migration

Servers anatomy

- There are several ways to organize servers.
- In the case of an iterative server, the server itself handles the request and, if necessary, returns a response to the requesting client.
- A concurrent server does not handle the request itself, but passes it to a separate thread or another process, after which it immediately waits for the next incoming request.
- A multithreaded server is an example of a concurrent server.
  - An alternative implementation of a concurrent server is to fork a new process for each new incoming request.

Where clients contact a server

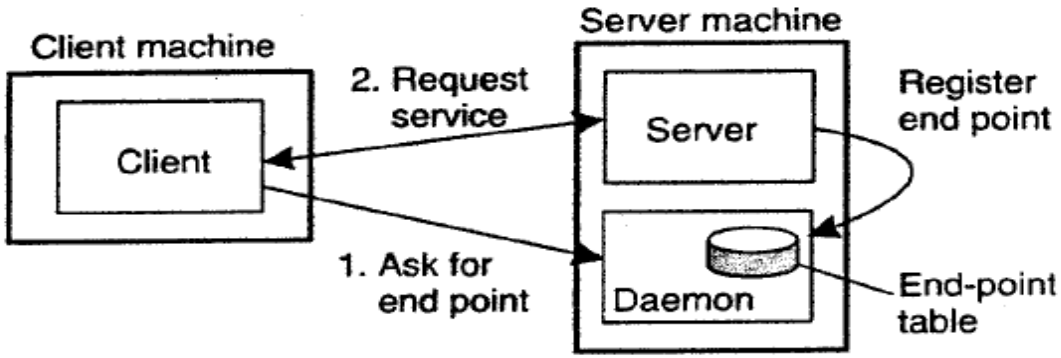
- Another issue is where clients contact a server.
- In all cases, clients send requests to an end point, also called a port, at the machine where the server is running. Each server listens to a specific end point.
- How do clients know the end point of a service? One approach is to globally assign end points for well-known services.
- For example, servers that handle Internet FTP requests always listen to TCP port 21. Likewise, an HTTP server for the World Wide Web will always listen to TCP port 80.
- End points have been assigned by the Internet Assigned Numbers Authority (IAN).
- With assigned end points, the client only needs to find the network address of the machine where the server is running.
  - As we will explain after, name services can be used for that purpose.

Ports for services

ftp-data	20	File Transfer [Default Data]
ftp	21	File Transfer [Control]
telnet	23	Telnet
	24	any private mail system
smtp	25	Simple Mail Transfer
login	49	Login Host Protocol
sunrpc	111	SUN RPC (portmapper)
courier	530	Xerox RPC

Daemons

- There are many services that do not require a preassigned end point. For example, a time-of-day server may use an end point that is dynamically assigned to it by its local operating system.
- In that case, a client will first have to look up the end point.
- One solution is to have a special daemon running on each machine that runs servers.
- The daemon keeps track of the current end point of each service implemented by a co-located server. The daemon itself listens to a well-known end point. A client will first contact the daemon, request the end point, and then contact the specific server.



(a)

Superserver

- It is common to associate an end point with a specific service.
- However, actually implementing each service by means of a separate server may be a waste of resources.
  - For example, in a typical UNIX system, it is common to have lots of servers running simultaneously, with most of them passively waiting until a client request comes in.
- Instead of having to keep track of so many passive processes, it is often more efficient to have a single superserver listening to each end point associated with a specific service.

Superserver

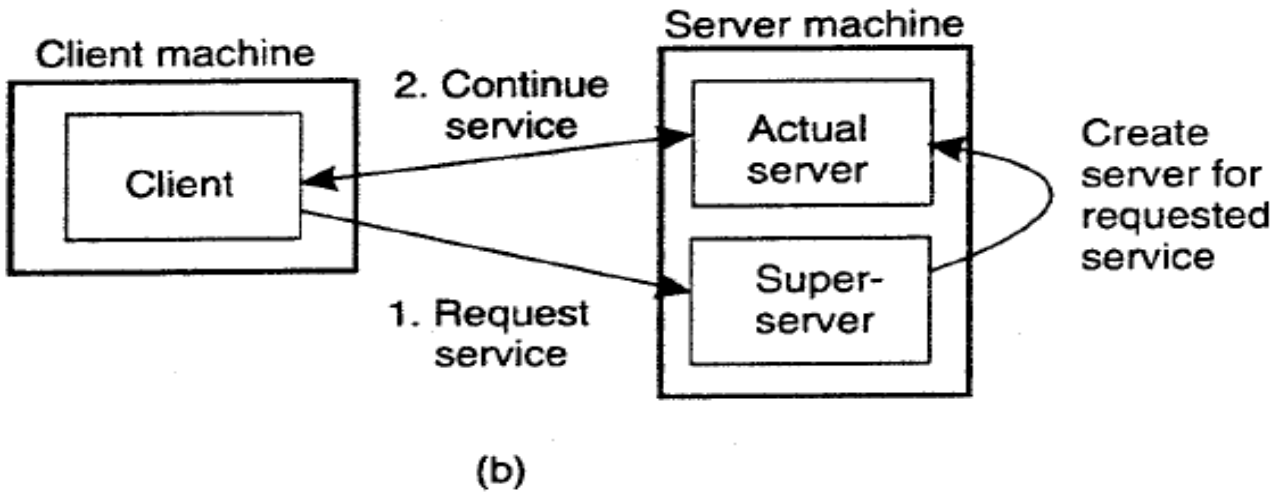


Figure 3-11. (a) Client-to-server binding using a daemon. (b) Client-to-server binding using a superserver..

Server Clusters

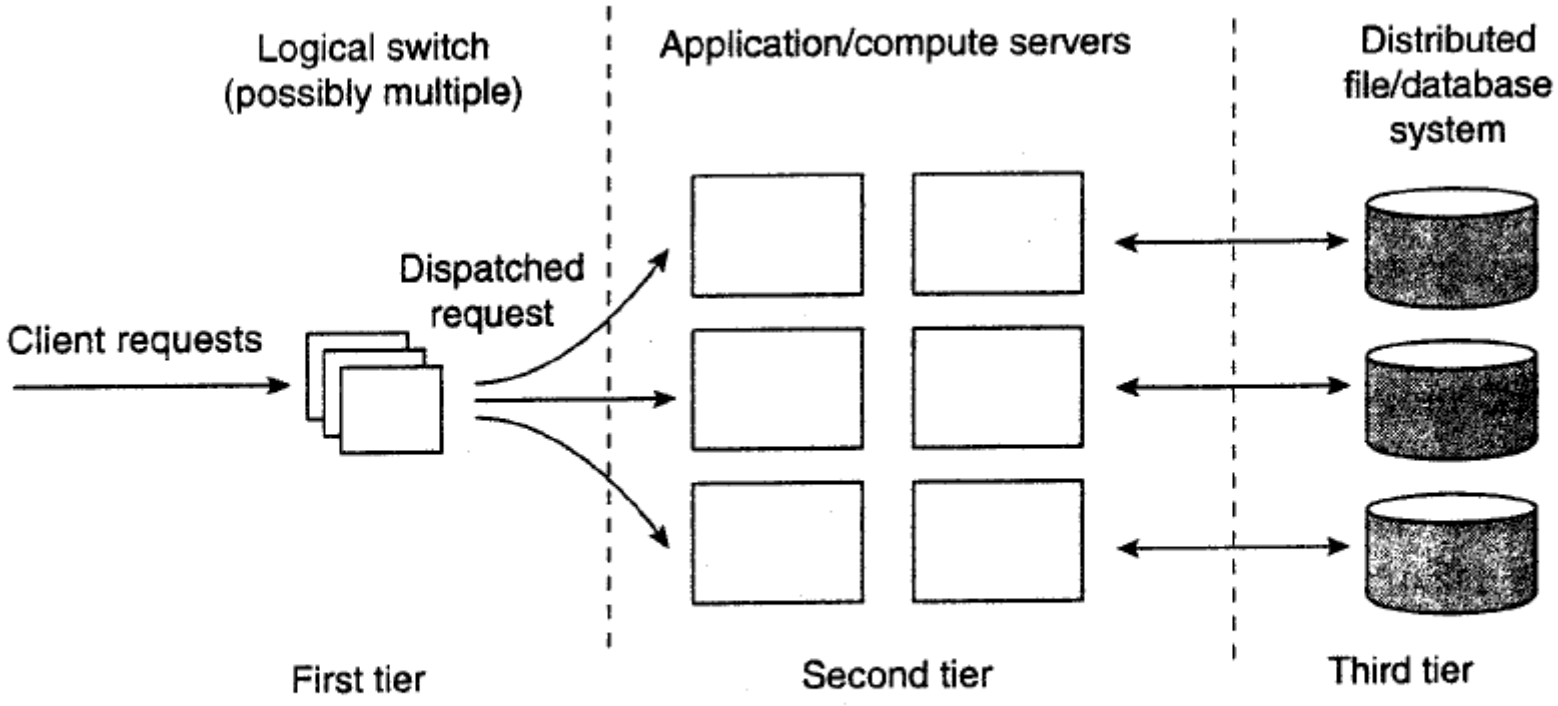


Figure 3-12. The general organization of a three-tiered server cluster.

Critical element

The first tier is generally responsible for passing requests to an appropriate server.

Server clusters for streaming media

- Of course, not all server clusters will follow this strict separation in 3 parts.

- It is frequently the case that each machine is equipped with its own local storage, often integrating application and data processing in a single server leading to a two tiered architecture.

- For example, when dealing with streaming media by means of a server cluster, it is common to deploy a two-tiered system architecture, where each machine acts as a dedicated media server.

PART I - Processes

- Threads
- Virtualization
- Clients
- Servers
- Migration

Code Migration

- Many second-tier machines run only a single application.
  - This limitation comes from dependencies on available software and hardware, but also that different applications are often managed by different administrators.

- As a consequence, we may find that certain machines are temporarily idle, while others are receiving an overload of requests.
  - What would be useful is to temporarily migrate services to idle machines.
  - A solution is to use virtual machines allowing a relative easy migration of code to real machines.

Distributed Servers

- The server clusters discussed so far are generally rather statically configured.
  - A separate administration machine keeps track of available servers, and passes this information to other machines such as the switch.

- Most server clusters offer a single access point. When that point fails, the cluster becomes unavailable.

- To eliminate this potential problem, several access points can be provided, of which the addresses are made publicly available.

Route optimization: different access points

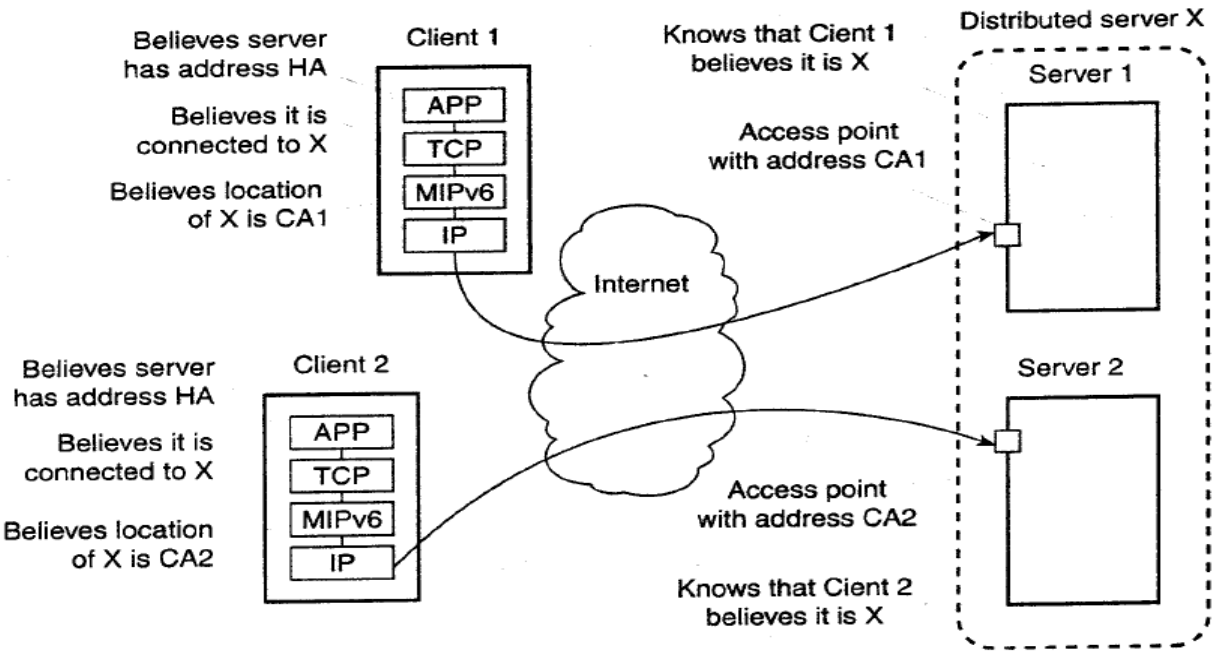


Figure 3-14. Route optimization in a distributed server.

Route optimization can be used to make different clients believe they are communicating with a single server, where, in fact, each client is communicating with a different member node of the distributed server

Code Migration

- Until now we have been mainly concerned with distributed systems in which communication is limited to passing data.

- However, there are situations in which passing programs, sometimes even while they are being executed, simplifies the design of a distributed system.

Heterogeneity: moving processes on Process VMs

- About 25 years later, code migration in heterogeneous systems is being attacked by scripting languages and highly portable languages such as Java.

- All such solutions have in common that they rely on a (process) virtual machine that either directly interprets source code (as in the case of scripting languages), or otherwise interprets intermediate code generated by a compiler (as in Java).

Heterogeneity: moving entire computing environment

- Recent developments have started to weaken the dependency on programming languages.

- In particular, solutions have been proposed not only to migrate processes, but to migrate entire computing environments.

Migrating entire environments

- There are several reasons for wanting to migrate entire environments, but perhaps the most important one is that it allows continuation of operation while a machine needs to be shutdown.
- For example, in a server cluster, the systems administrator may decide to shut down or replace a machine, but will not have to stop all its running processes.
  - Instead, it can temporarily freeze an environment, move it to another machine (where it sits next to other, existing environments), and simply unfreeze it again.
- Clearly, this is an extremely powerful way to manage long-running compute environments and their processes.

Migrating Operating Systems ☺

- The overall effect of migrating environments, instead of migrating processes, is that now we actually see that an entire operating system can be moved between machines.

End of PART I

- Readings
  - Distributed Systems, Chapter 3

PART II - Communication

- Fundamentals
- Remote Procedure Call
- Message-Oriented Middleware (MOM)
- Data streaming

Interprocess communication

- Interprocess communication is at the heart of all distributed systems.
- Communication in distributed systems is always based on low-level message passing as offered by the underlying network.
  - Expressing communication through message passing is harder than using primitives based on shared memory, as available for nondistributed platforms.
- Modern distributed systems often consist of thousands or even millions of processes scattered across a network with unreliable communication such as the Internet.

Protocols and Models

- The rules that communicating processes must adhere to are known as protocols,
  - We concentrate on structuring those protocols in the form of layers.
    - Low-level layers
    - Transport layer
    - Application layer
    - Middleware layer

- Widely-used models for communication:
  - Remote Procedure Call (RPC)
  - Message-Oriented Middleware (MOM)
  - Data streaming

Absence of shared memory

- Due to the absence of shared memory, all communication in distributed systems is based on sending and receiving (low level) messages.
- When process *A* wants to communicate with process *B,* it first builds a message in its own address space.
- Then it executes a system call that causes the operating system to send the message over the network to *B.*
- *A* and *B* have to agree on the meaning of the bits being sent!!!
  - If *A* sends a brilliant new novel written in French and encoded in IBM's EBCDIC character code, and *B* expects the inventory of a supermarket written in English and encoded in ASCII, communication will be less than optimal. ☺

How do process agree?

- Many different agreements are needed.
  - How many volts should be used to signal a 0-bit, and how many volts for a 1-bit?
  - How does the receiver know which is the last bit of the message?
  - How can it detect if a message has been damaged or lost, and what should it do if it finds out?
  - How long are numbers, strings, and other data items, and how are they represented?

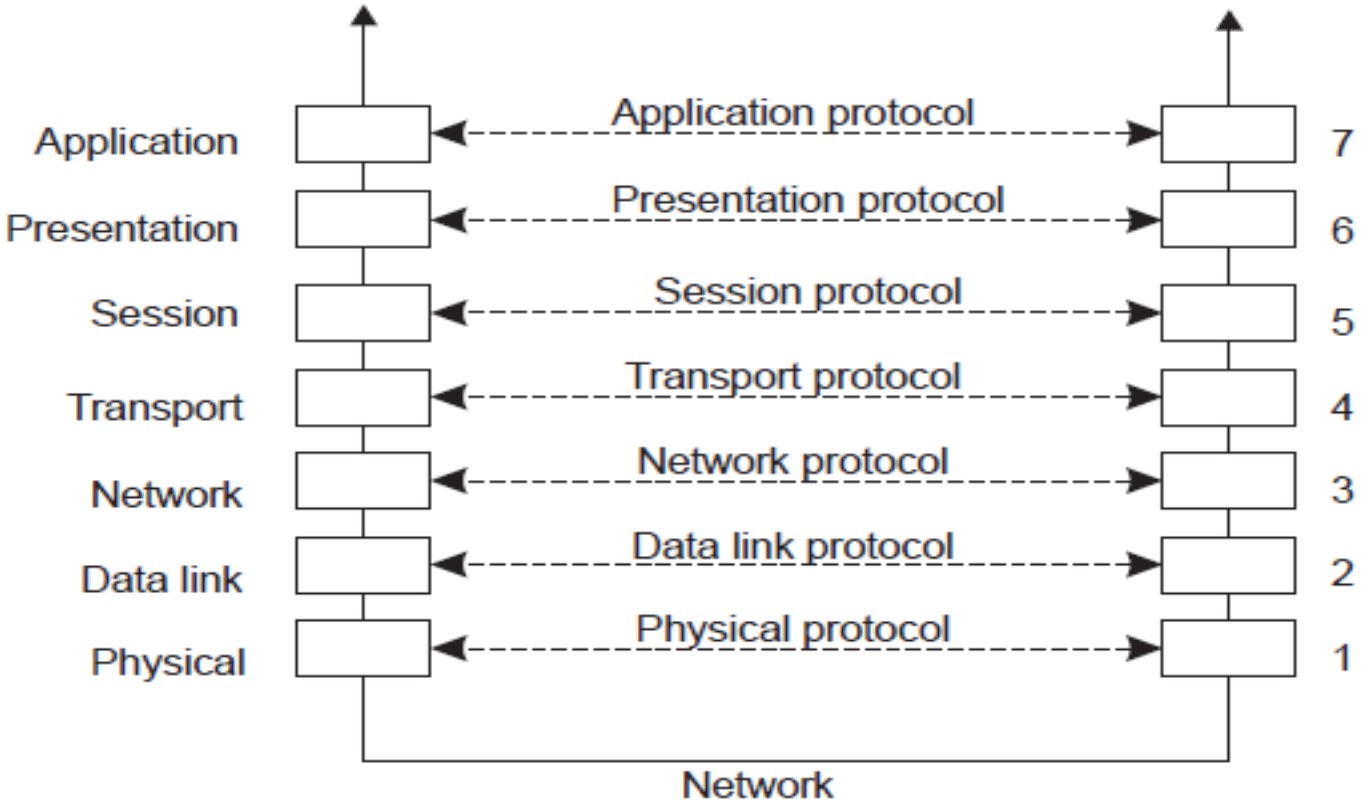
- In short, agreements are needed at a variety of levels, varying from the low-level details of bit transmission to the high-level details of how information is to be expressed.

The ISO/OSI reference model

- To make it easier to deal with the numerous levels and issues involved in communication, the International Standards Organization (ISO) developed a reference model that clearly identifies the various levels involved, gives them standard names, and points out which level should do which job.

- This model is called the Open Systems Interconnection Reference Model usually abbreviated as ISO OSI or sometimes just the OSI model.

The OSI model



Building messages

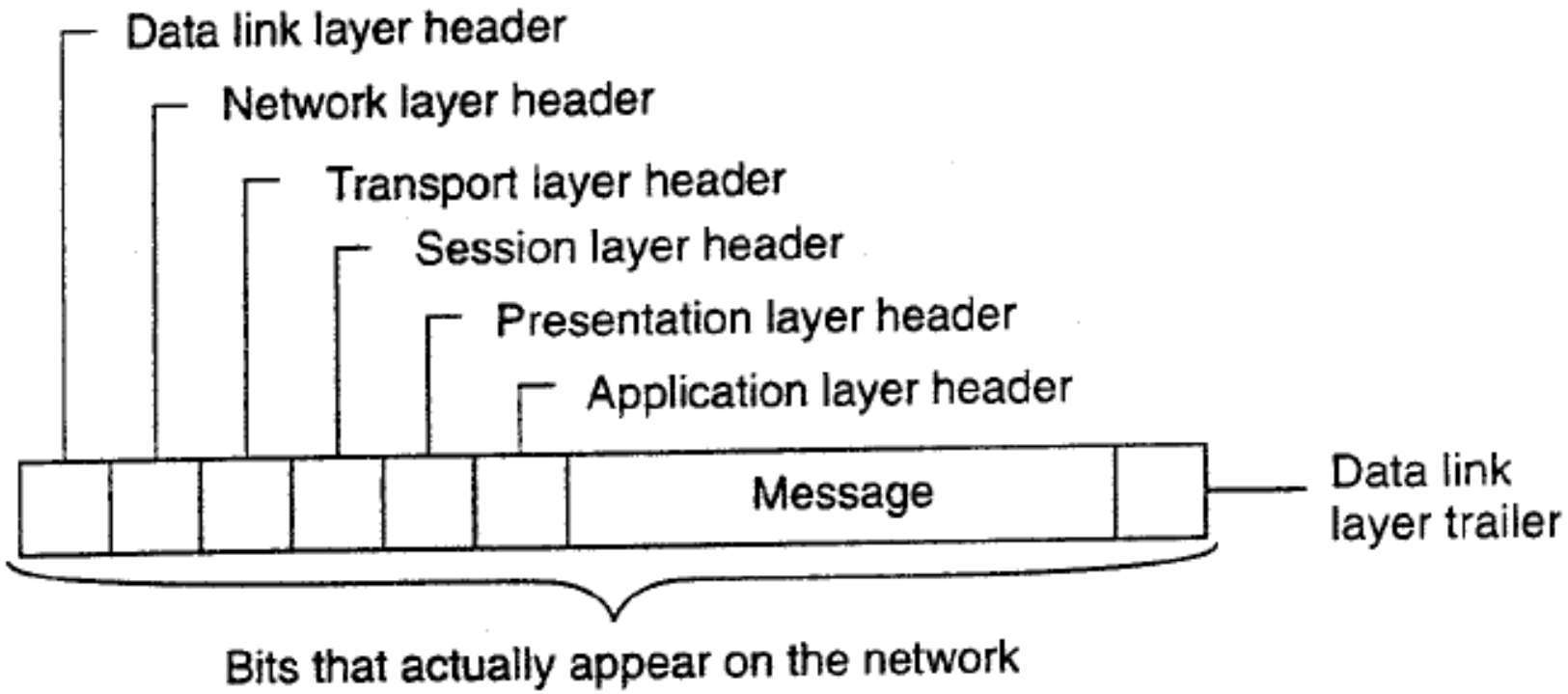


Figure 4-2. A typical message as it appears on the network.

Transport Protocol

- The job of the transport layer is to provide a reliable connection.
  - The idea is that the application layer should be able to deliver a message to the transport layer with the expectation that it will be delivered without loss.
  - Upon receiving a message from the application layer, the transport layer breaks it into pieces small enough for transmission, assigns each one a sequence number, and then sends them all.

- The discussion in the transport layer header concerns:
  - which packets have been sent
  - which have been received
  - how many more the receiver has room to accept
  - which should be retransmitted
  - and similar topics.

Problems of OSI

- What is missing in OSI is:

  a clear distinction between applications, application-specific protocols, and general-purpose protocols.

- Internet File Transfer Protocol (FTP) defines a protocol for transferring files between a client and server machine.
  - The protocol should not be confused with the *ftp* program, which is an end-user application for transferring files and which also (not entirely by coincidence) happens to implement the Internet FTP.

Problems of OSI

- A typical application-specific protocol is the HyperText Transfer Protocol (HTTP), designed to remotely manage the transfer of Web pages. Implemented by Web browsers and Web servers.
  - However, HTTP is now also used by systems that are not intrinsically tied to the Web.
  - For example, Java's object-invocation mechanism uses HTTP to request the invocation of remote objects that are protected by a firewall (Sun Microsystems, 2004b).

- There are also many general-purpose protocols that are useful to many applications, but which cannot be qualified as transport protocols.
- In many cases, such protocols fall into the category of middleware protocols

Middleware layer

- Middleware is invented to provide common services and protocols that can be used by many different applications.

- Middleware is an application that logically lives (mostly) in the application layer, but which contains many general-purpose protocols that warrant their own layers, independent of other, more specific applications.

- A distinction can be made between high-level communication protocols and protocols for establishing various middleware services.

Middleware services

- There are numerous protocols to support a variety of middleware services.
- Many protocols tend to have a general application-independent nature
- For example:
  - Authentication protocols are not closely tied to any specific application, but instead, can be integrated into a middleware system as a general service.
  - Authorization protocols by which authenticated users and processes are granted access only to those resources for which they have authorization.

Middleware services

- Commit Protocols
  - Commit protocols establish that in a group of processes either all processes carry out a particular operation, or that the operation is not carried out at all.
    - This phenomenon is also referred to as atomicity and is widely applied in transactions.

- Distributed locking protocol
  - By which a resource can be protected against simultaneous access by a collection of processes that are distributed across multiple machines.

Middleware communication services

- Middleware communication protocols support high-level communication services.

- For example, there are protocols that allow a process to call a procedure or invoke an object on a remote machine in a highly transparent way: Remote Method Invocation

- Likewise, there are high-level communication services:
  - for setting and synchronizing streams
  - for transferring real-time data, such as needed for multimedia applications.

- Finally some middleware systems offer reliable multicast services that scale to thousands of receivers spread across a wide area network.

Plugging the Middleware

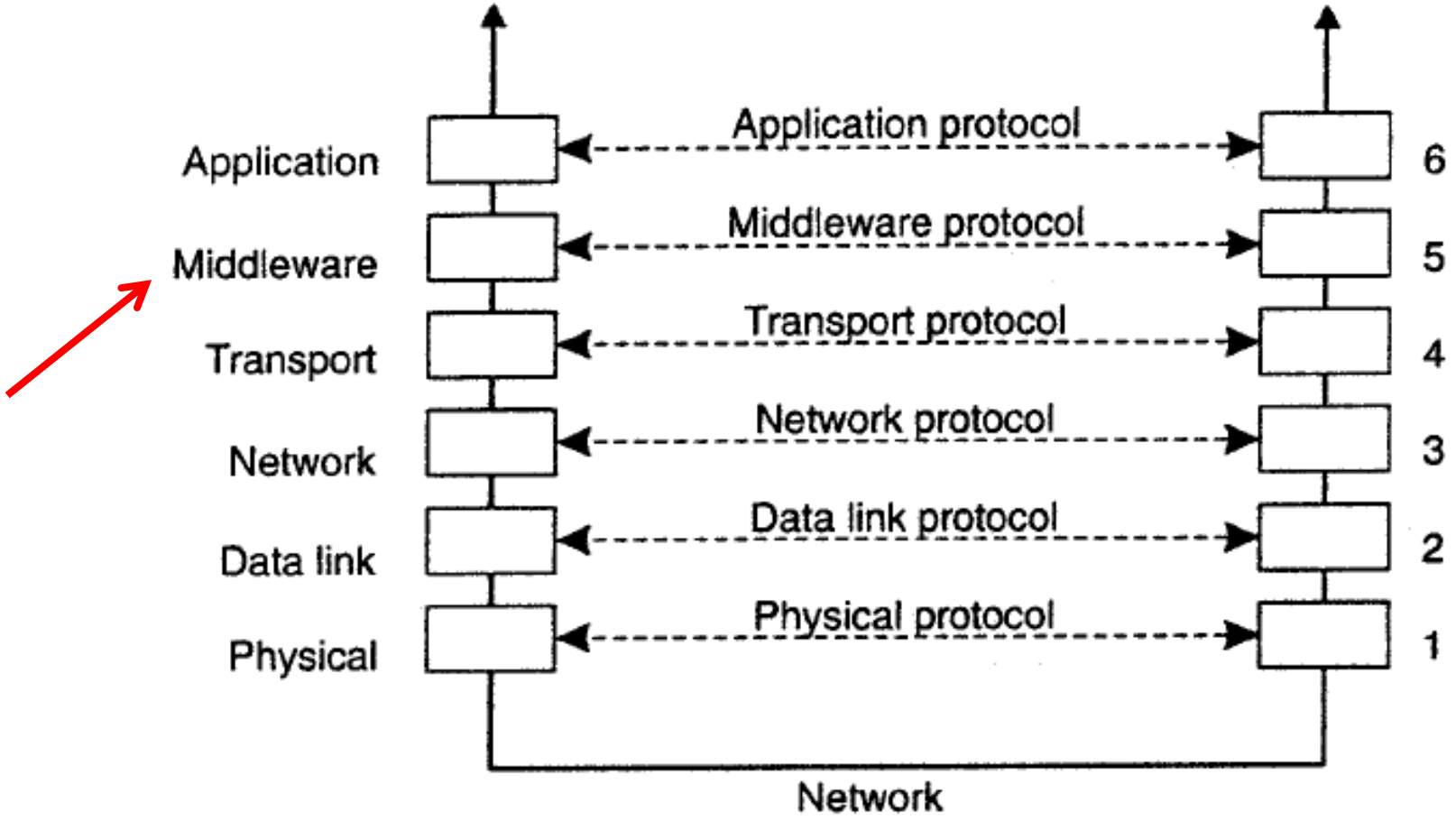


Figure 4-3. An adapted reference model for networked communication.

PART II - Communication

- Fundamentals
- Remote Procedure Call
- Message-Oriented Middleware (MOM)
- Data streaming

The RPC basic idea

- All application developers are familiar with simple procedure model
  - Procedures operate in isolation (black box)
  - There is no fundamental reason not to execute procedures on separate machine
- A paper by Birrell and Nelson (1984) introduced a completely different way of handling communication
  - Allow programs to call procedures located on other machines.
  - When a process on machine *A* calls' a procedure on machine *B,* the calling process on *A* is suspended, and execution of the called procedure takes place on *B.*
  - Information can be transported from the caller to the callee in the parameters and can come back in the procedure result.
- No message passing at all is visible to the programmer.
- This method is known as **Remote Procedure Call,** or often just RPC.

Client-Server with RPC

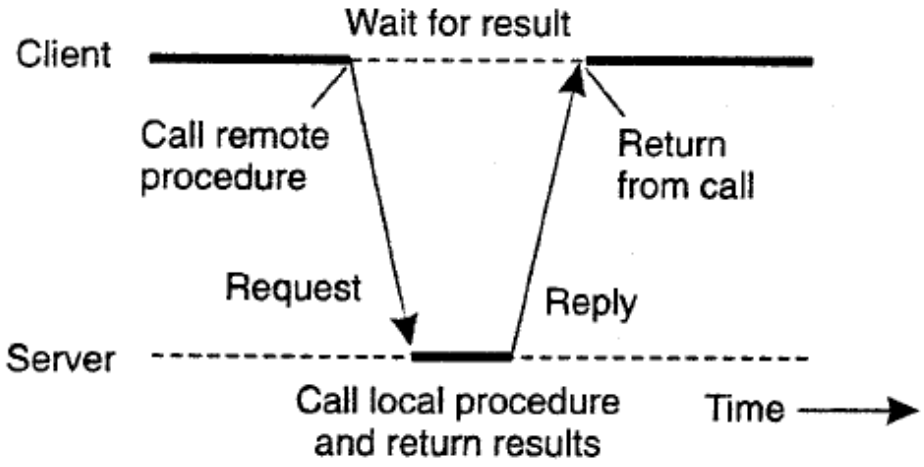
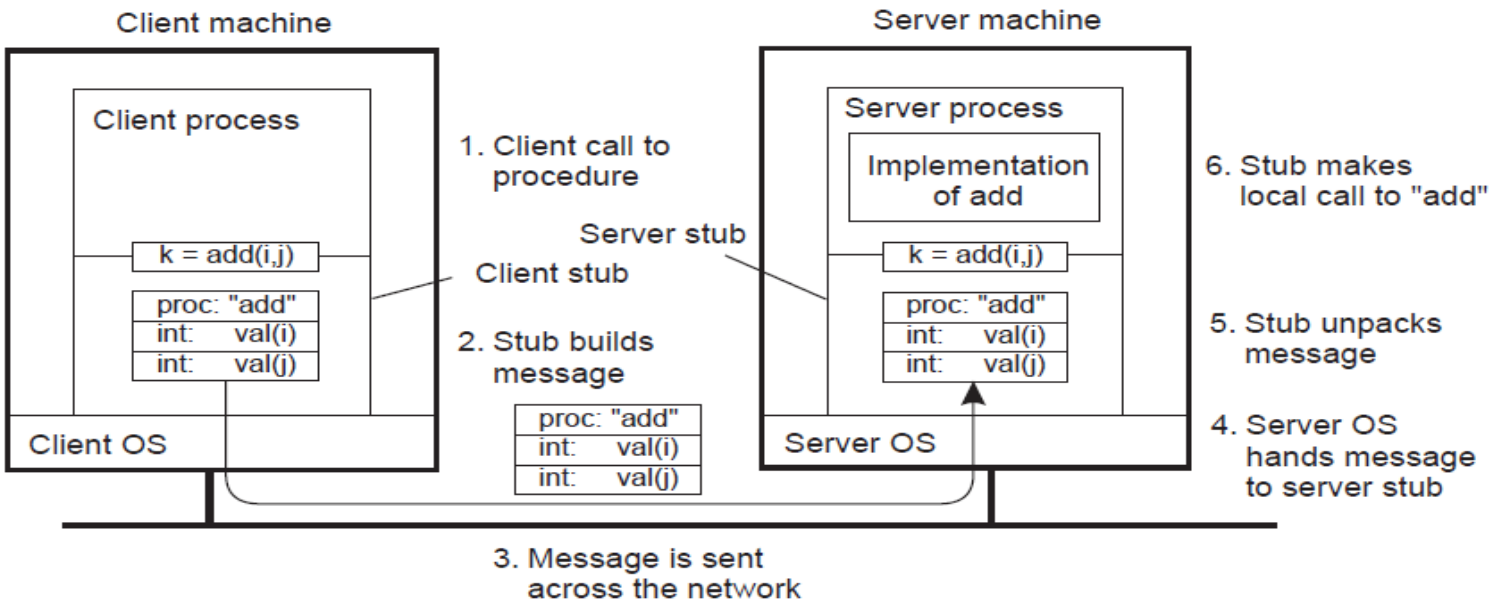


Figure 4-6. Principle of RPC between a client and server program.

RPC Step-by-Step



1. Client procedure calls client stub.
2. Stub builds message; calls local OS.
3. OS sends message to remote OS.
4. Remote OS gives message to stub.
5. Stub unpacks parameters and calls server.
6. Server returns result to stub.
7. Stub builds message; calls OS.
8. OS sends message to client's OS.
9. Client's OS gives message to stub.
10. Client stub unpacks result and returns to the client.

RPC: Parameter Passing

- The function of the client stub is to take its parameters, pack them into a message, and send them to the server stub.
- Packing parameters into a message is called **parameter marshaling**.
- Problems:
  - Client and server machines may have different data representations (for example byte ordering)
  - Wrapping a parameter means transforming a value into a sequence of bytes
  - Client and server have to agree on the same encoding:
    - How are basic data values represented (integers, floats, characters)
    - How are complex data values represented (arrays, unions)
  - Client and server need to properly interpret messages, transforming them into machine-dependent representations.

Types of RPCs: synchronous

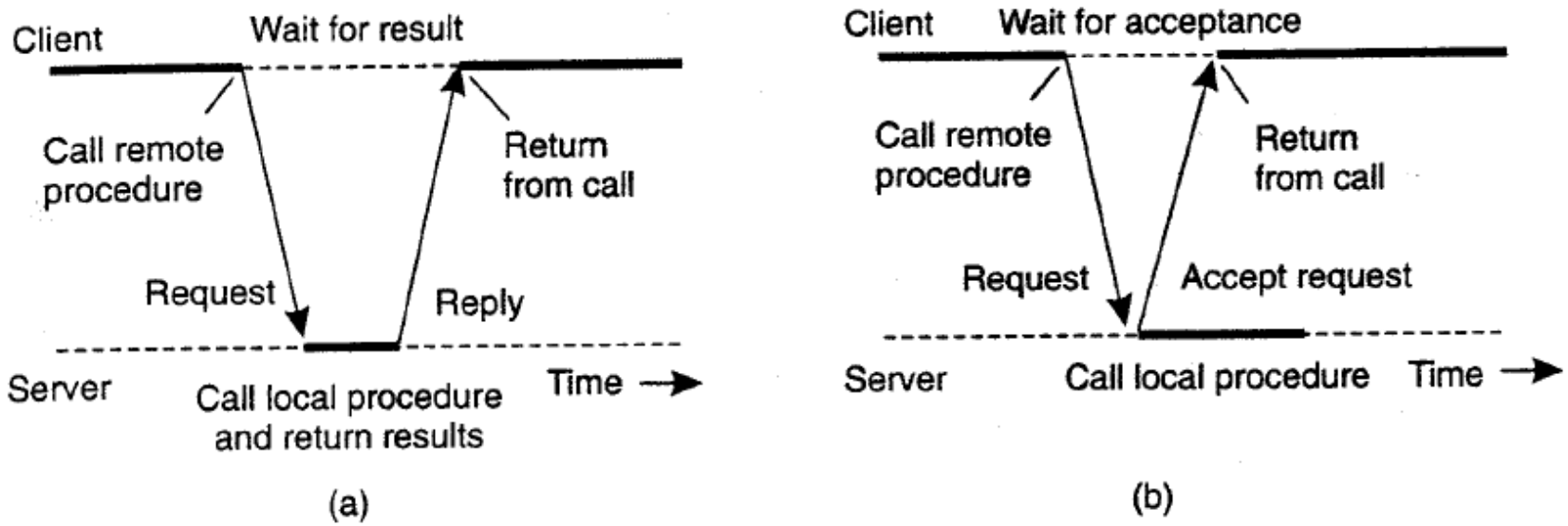


Figure 4-10. (a) The interaction between client and server in a traditional RPC. (b) The interaction using asynchronous RPC.

Can we try to get rid of the strict request-reply behavior? Let the client continue without waiting for an answer from the server. => next slide

Asynchronous RPCs

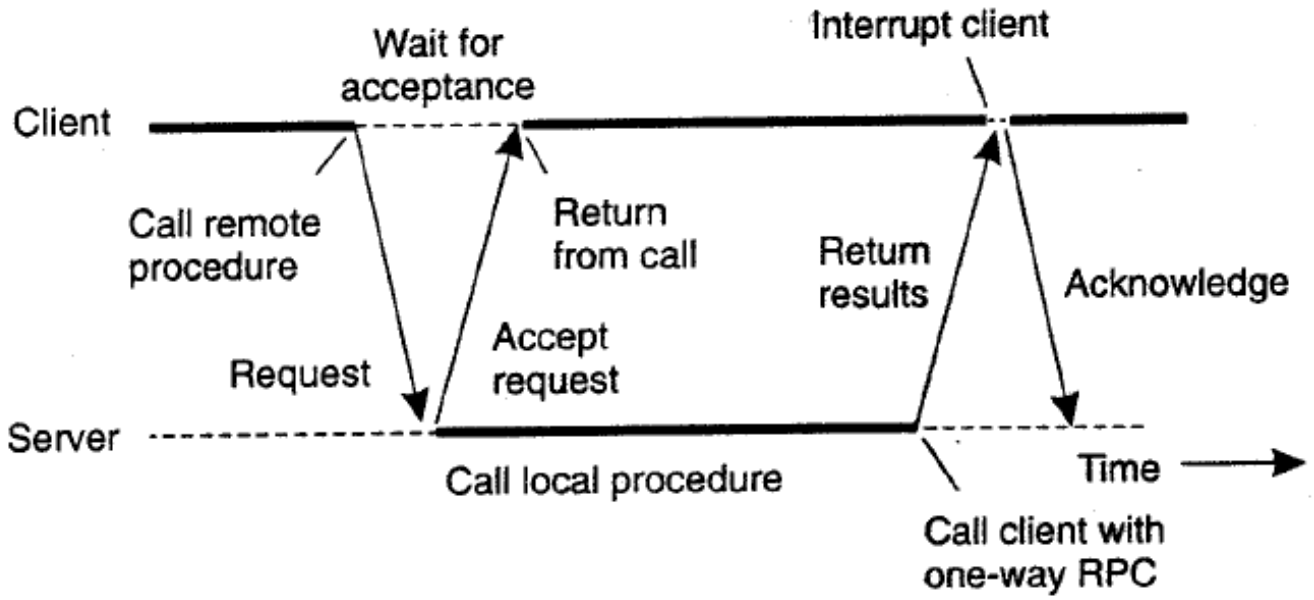


Figure 4-11. A client and server interacting through two asynchronous RPCs.

PART II - Communication

- Fundamentals
- Remote Procedure Call
- Message-Oriented Middleware (MOM)
- Data streaming

Message-Oriented Communication

- Transient Messaging
- Message-Queuing System
- Message Brokers
- Example: IBM Websphere

RPC: not always solves

- Remote procedure calls and remote object invocations contribute to hiding communication in distributed systems, that is, they enhance access transparency.
- Unfortunately, neither mechanism is always appropriate.
- When it cannot be assumed that the receiving side is executing at the time a request is issued, alternative communication services are needed.
- The inherent synchronous nature of RPCs, by which a client is blocked until its request has been processed, sometimes needs to be replaced by something else.

Transient messaging: Berkeley sockets

- Conceptually, a socket is a communication end point to which an application can write data that are to be sent out over the underlying network, and from which incoming data can be read.

- *Definition: A socket forms an abstraction over the actual communication end point that is used by the local operating system for a specific transport protocol.*

- (Dict. transient: Lasting only for a short time; impermanent)

Socket Primitives for TCP/IP

Primitive	Meaning
Socket	Create a new communication end point
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

Figure 4-14. The socket primitives for TCPIIP.

Transient messaging: sockets

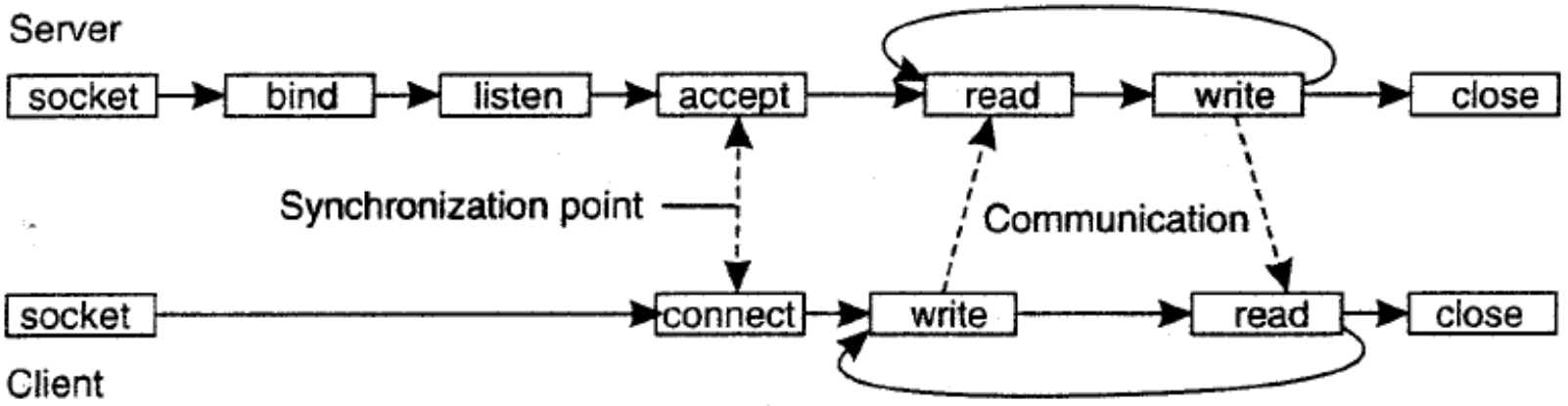


Figure 4-15. Connection-oriented communication pattern using sockets.

MPI: Message Passing Interface

- The need to be hardware and platform independent eventually has led to the definition of a standard for message passing, simply called the Message-Passing Interface or MPI.

- MPI is designed for parallel applications and as such is tailored to transient communication.

- MPI makes direct use of the underlying network.

MPI Primitives

Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and wait until receipt starts
MPI_recv	Receive a message; block if there is none
MPI_irecv	Check if there is an incoming message, but do not block

Figure 4-16. Some of the most intuitive message-passing primitives of MPI.

Message-Oriented Persistent Communication

- Asynchronous persistent communication through support of middleware-level queues.
  - Queues correspond to buffers at communication servers.

- Message-queuing systems
  - An important aspect of message-queuing systems is that a sender is generally given only the guarantees that its message will eventually be inserted in the recipient's queue.
  - No guarantees are given about when, or even if the message will actually be read, which is completely determined by the behavior of the recipient.

Architecture of a Message-Queuing System

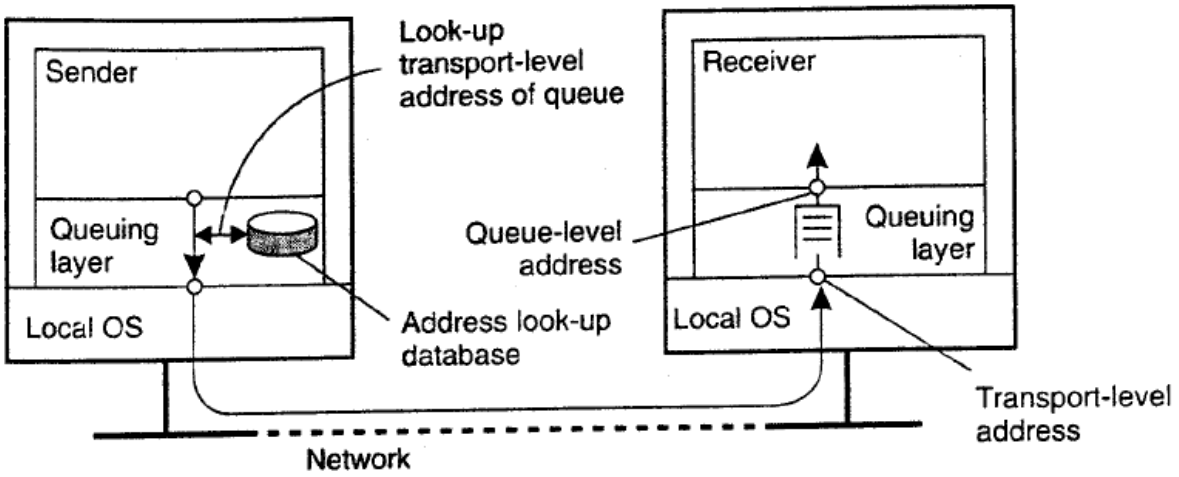


Figure 4-19. The relationship between queue-level addressing and network-level addressing.

- The collection of queues is distributed across multiple machines.
- Consequently, for a message-queuing system to transfer messages, it should maintain a mapping of queues to network locations.
- In practice, this means that it should maintain a (possibly distributed) database of queue names to network locations.
- Such a mapping is analogous to the Domain Name System (DNS) for e-mail in the Internet.

Message broker

Observation

- Message queuing systems assume a common messaging protocol: all applications agree on message format (i.e., structure and data representation)

Message broker

Centralized component that takes care of application heterogeneity in an MQ system:

- Transforms incoming messages to target format
- Very often acts as an application gateway
- May provide subject-based routing capabilities => Enterprise Application Integration

Message Broker

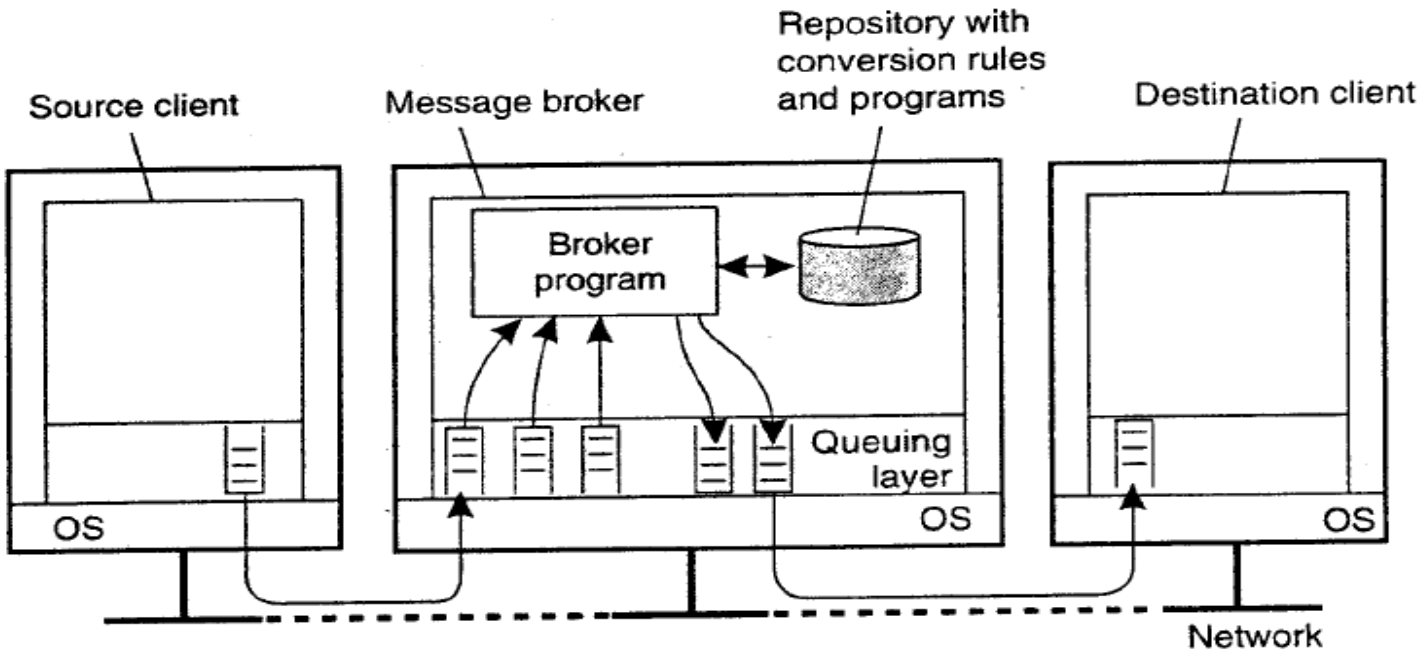


Figure 4-21. The general organization of a message broker in a message-queuing system.

Message Broker in Enterprise Information Systems

- More common is the use of a message broker for advanced enterprise application integration (EAI).
- In this case, rather than (only) converting messages, a broker is responsible for matching applications based on the messages that are being exchanged.
- In such a model, called publish/subscribe, applications send messages in the form of *publishing.*
  - In particular, they may publish a message on topic X, which is then sent to the broker.
  - Applications that have stated their interest in messages on topic X, that is, who have *subscribed* to those messages, will then receive these messages from the broker.

E-Mail systems

- E-mail systems are generally implemented through a collection of mail servers that store and forward messages on behalf of the users on hosts directly connected to the server.
  - For example, in the mail protocol for the Internet, SMTP (Postel, 1982), a message is transferred by setting up a direct TCP connection to the destination mail server.

General message-queuing systems

- General message-queuing systems are not aimed at supporting only end users.
  - They are set up to enable persistent communication between processes, regardless of whether a process is running a user application, handling access to a database or performing computations.

IBM Websphere

Message transfer

- Messages are transferred between queues
- Message transfer between queues at different processes, requires a channel
- At each endpoint of channel is a message channel agent (MCA)
- Message channel agents are responsible for:
  - Setting up channels using lower-level network communication facilities (e.g., TCP/IP)
  - (Un)wrapping messages from/in transport-level packets
  - Sending/receiving packets

Websphere

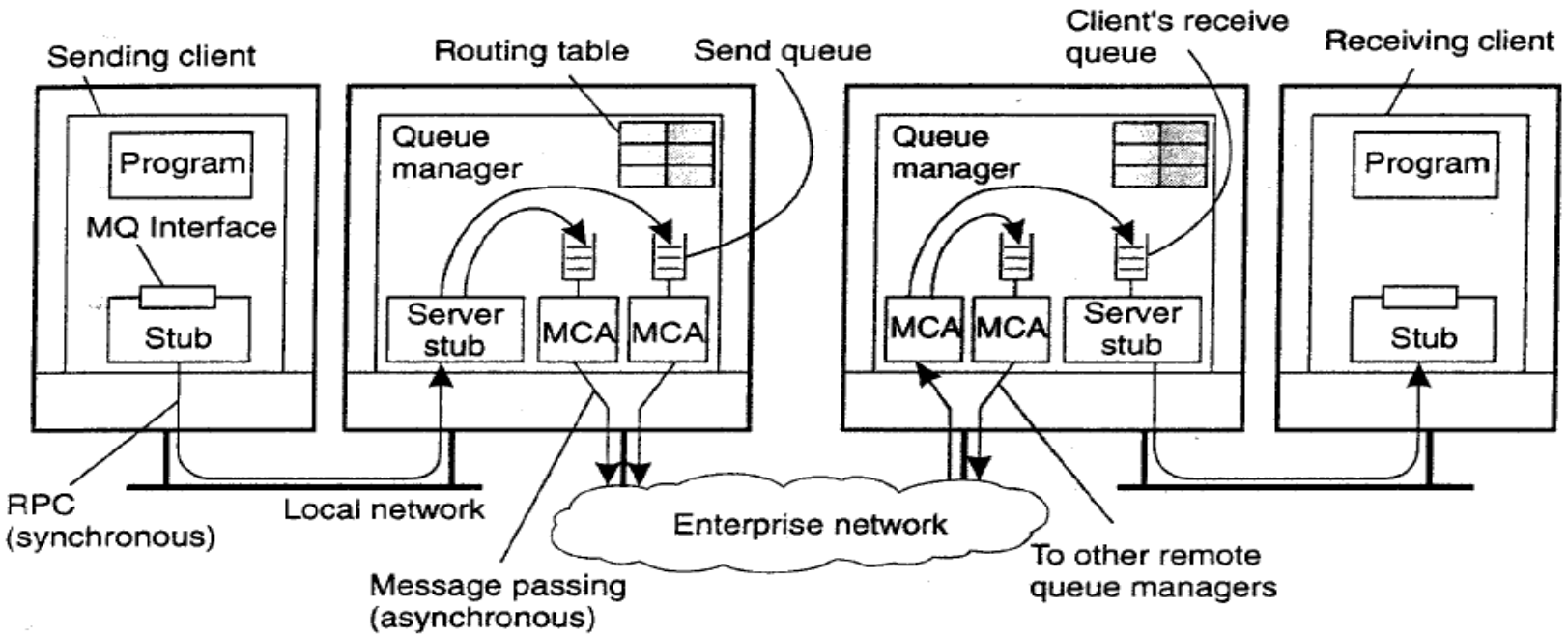


Figure 4-22. General organization of IBM's message-queuing system.

PART II - Communication

- Fundamentals
- Remote Procedure Call
- Message-Oriented Middleware (MOM)
- Data streaming

Stream-oriented communication

- Support for continuous media
- Streams in distributed systems
- Stream management

Continuous flow of data

Observation

- All communication facilities discussed so far are essentially based on a discrete, that is time-independent exchange of information
- **Continuous media**
  - Characterized by the fact that values are time dependent:
    - Audio
    - Video
    - Animations
    - Sensor data (temperature, pressure, etc.)

Continuous media

Transmission modes

Different timing guarantees with respect to data transfer:

- Asynchronous: no restrictions with respect to when data is to be delivered
- Synchronous: define a maximum end-to-end delay for individual data packets
- Isochronous: define a maximum and minimum end-to-end delay

Stream

Definition

A (continuous) data stream is a connection-oriented communication facility that supports isochronous data transmission.

Some common stream characteristics

- Streams are unidirectional
- There is generally a single source, and one or more destinations

- Simple stream: a single flow of data, e.g., audio or video
- Complex stream: multiple data flows, e.g., stereo audio or combination audio/video

Streaming

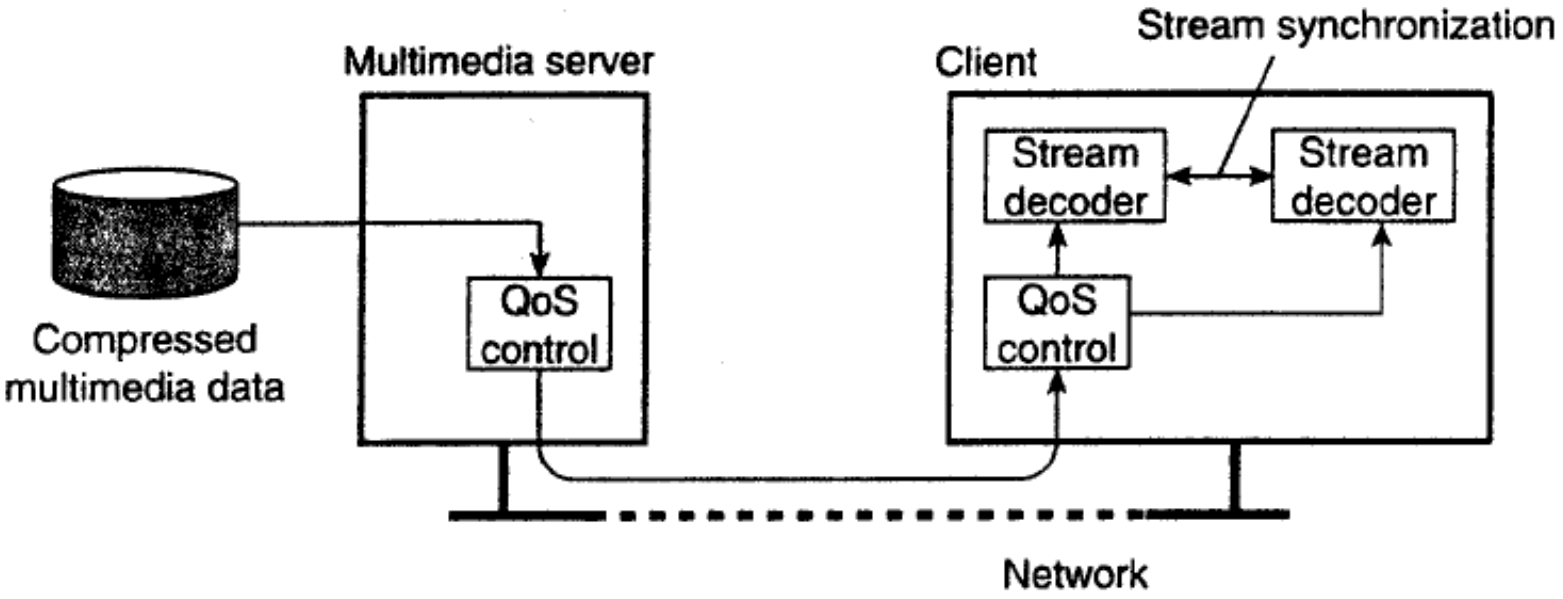


Figure 4-26. A general architecture for streaming stored multimedia data over a network.

Streams and QoS

Essence

- Streams are all about timely delivery of data. How do you specify this Quality of Service (QoS)?
- Basics:
  - The required bit rate at which data should be transported.
  - The maximum delay until a session has been set up (i.e., when an application can start sending data).
  - The maximum end-to-end delay (i.e., how long it will take until a data unit makes it to a recipient).
  - The maximum delay variance, or jitter.
  - The maximum round-trip delay.

Stream synchronization

Problem

- Given a complex stream, how do you keep the different substreams in synch?

Example

- Think of playing out two channels, that together form stereo sound. Difference should be less than 20–30 msec!

Stream explicit synchronization at level data units

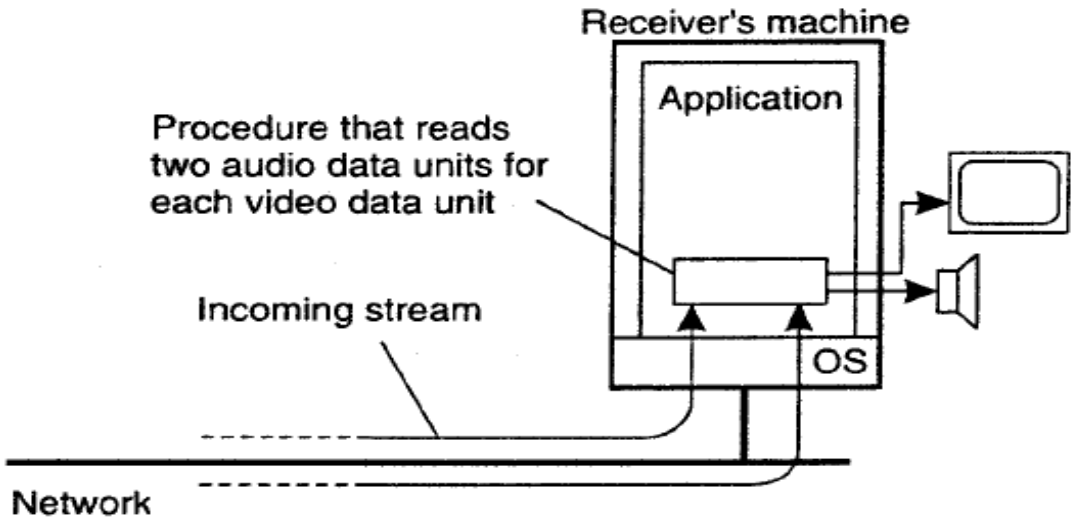


Figure 4-29. The principle of explicit synchronization on the level data units.

Stream synchronization: high-level interface

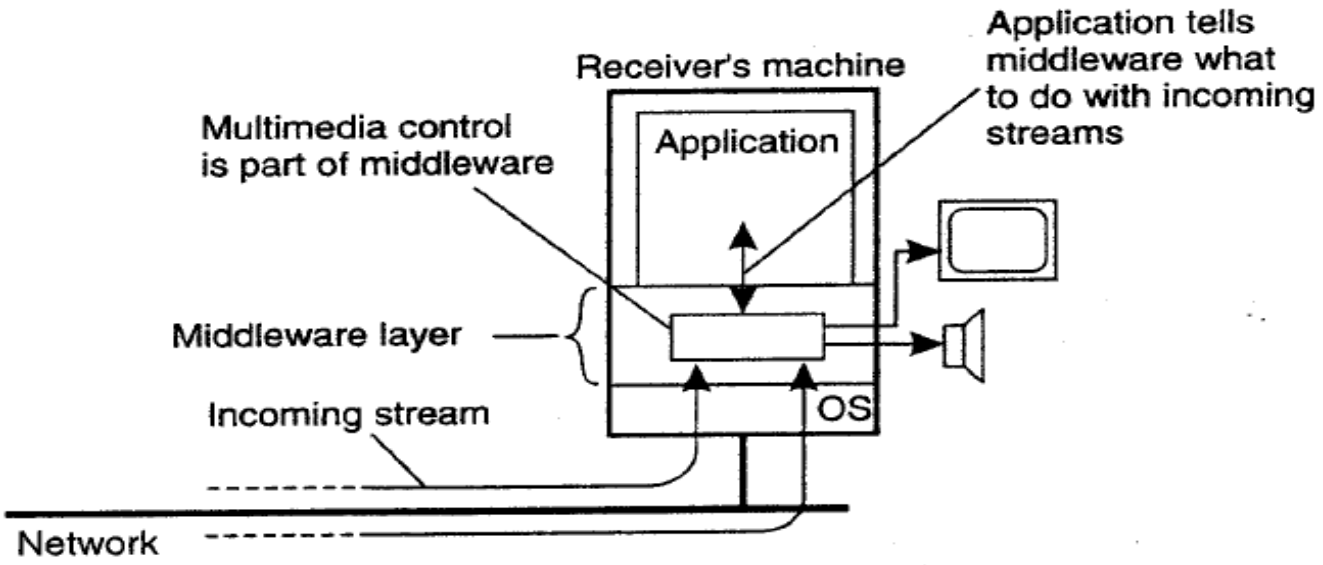


Figure 4·30. The principle of synchronization as supported by high-level interfaces.

- Multiplex all substreams into a single stream, and demultiplex at the receiver.
- Synchronization is handled at multiplexing/demultiplexing point (this approach is followed for MPEG streams).

End of PART II

- Readings
  - Distributed Systems, Chapter 4.

PART III - Naming

- Names, Identifiers, and addresses
- Flat Naming
- Structured Naming
- Attributed-based Naming

Names

- Names are used to share resources, to uniquely identify entities, to refer to locations, and more.
- An important issue with naming is that a name can be resolved to the entity it refers to.
- Name resolution thus allows a process to access the named entity.
- To resolve names, it is necessary to implement a naming system.

Naming

- Names, identifiers, and addresses
- Name resolution
- Name space implementation

Names and Access Points

- A name in a distributed system is a string of bits or characters that is used to refer to an entity.
  - An entity in a distributed system can be practically anything.

- To operate on an entity, it is necessary to access it, for which we need an access point.
  - An access point is yet another, but special, kind of entity in a distributed system
  - The name of an access point is called an address

Access Points

- An entity can offer more than one access point.
- In a distributed system, a typical example of an access point is a host running a specific server, with its address formed by the combination of, for example, an IP address and port number (i.e., the server's transport-level address).
- An entity may change its access points in the course of time.
  - For example. when a mobile computer moves to another location, it is often assigned a different IP address than the one it had before.

Location Independent

- If an entity offers more than one access point, it is not clear which address to use as a reference.
- A much better solution is to have a single name for the Web service independent from the addresses of the different Web servers.
- Such a name is called location independent.

Identifiers

Pure name

A name that has no meaning at all; it is just a random string. Pure names can be used for comparison only.

Identifier

A name having the following properties:

- Each identifier refers to at most one entity
- Each entity is referred to by at most one identifier
- An identifier always refers to the same entity (prohibits reusing an identifier)

Name-to-address binding

- Having names, identifiers, and addresses brings us to the central theme of naming: how do we resolve names and identifiers to addresses?
  - It is important to realize that there is often a close relationship between name resolution in distributed systems and message routing.

- In principle, a naming system maintains a name-to-address binding which in its simplest form is just a table of *(name, address)* pairs.

- In distributed systems that span large networks and for which many resources need to be named, a centralized table is not going to work.

PART III - Naming

- Names, Identifiers, and addresses
- Flat Naming
- Structured Naming
- Attributed-based Naming

Flat Naming

Problem

Given an essentially unstructured name (e.g., an identifier), how can we locate its associated access point?

- Simple solutions
  - Broadcasting and Multicasting
- Home-based approaches
- Distributed Hash Tables (structured P2P)
- Hierarchical location service

Broadcasting

- Consider a distributed system built on a computer network that offers efficient broadcasting facilities.
  - Typically, such facilities are offered by local-area networks in which all machines are connected to a single cable or the logical equivalent thereof. Also, local-area wireless networks fall into this category.
- Locating an entity in such an environment is simple: a message containing the identifier of the entity is broadcast to each machine and each machine is requested to check whether it has that entity
  - Only the machines that can offer an access point for the entity send a reply message containing the address of that access point.

Multicasting

- Multicasting can also be used to locate entities in point-to-point networks.
  - For example, the Internet supports network-level multicasting by allowing hosts to join a specific multicast group.
- Such groups are identified by a multicast address.
- When a host sends a message to a multicast address, the network layer tries to deliver that message to all group members.

Forwarding Pointers

- Another popular approach to locating mobile entities is to make use of forwarding pointers.
- The principle is simple: *when an entity moves from A to B, it leaves behind in A a reference to its new location at B.*
  - The main advantage of this approach is its simplicity: as soon as an entity has been located, for example by using a traditional naming service, a client can look up the current address by following the chain of forwarding pointers.

Home-based approaches

Single-tiered scheme

Let a home keep track of where the entity is:

- Entity's home address registered at a naming service
- The home registers the foreign address of the entity
- Client contacts the home first, and then continues with foreign location

Home-based approaches: Mobile IP

- Each mobile host uses a fixed IP address.
- All communication to that IP address is initially directed to the mobile host's home agent.
- This home agent is located on the local-area network corresponding to the network address contained in the mobile host's IP address.
- In the case of IPy6, it is realized as a network-layer component.
- Whenever the mobile host moves to another network, it requests a temporary address that it can use for communication.
- This care-of address is registered at the home agent.

Home-based approaches: Mobile IP

- When the home agent receives a packet for the mobile host, it looks up the host's current location.
  - If the host is on the current local network, the packet is simply forwarded.
  - Otherwise, it is tunneled to the host's current location, that is, wrapped as data in an IP packet and sent to the care-of address.
- At the same time, the sender of the packet is informed of the host's current location.

Home-based approaches: Mobile IP

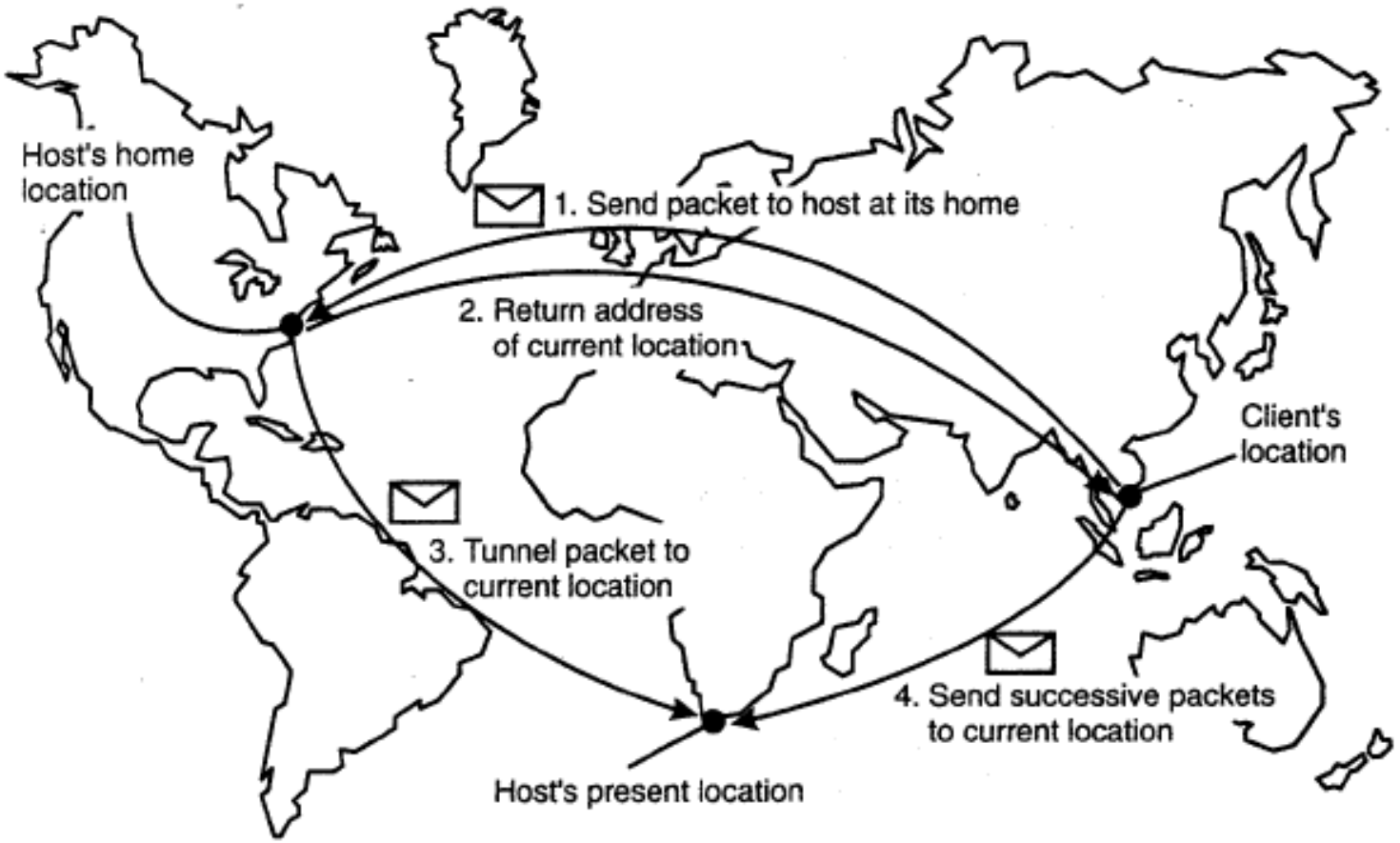


Figure 5-3. The principle of Mobile IP.

Home-based approaches

Problems with home-based approaches

- Home address has to be supported for entity's lifetime
- Home address is fixed => unnecessary burden when the entity permanently moves
- Poor geographical scalability (entity may be next to client)

Question

- How can we solve the "permanent move" problem?

Distributed Hash Tables (DHT)

Consider the organization of many nodes into a logical ring

- Each node is assigned a random m-bit identifier.
- Every entity is assigned a unique m-bit key.
- Entity with key k falls under jurisdiction of node with smallest id ≥ k (called its successor).

- DHT-based systems provide several name resolution services

Hierarchical Location Services (HLS)

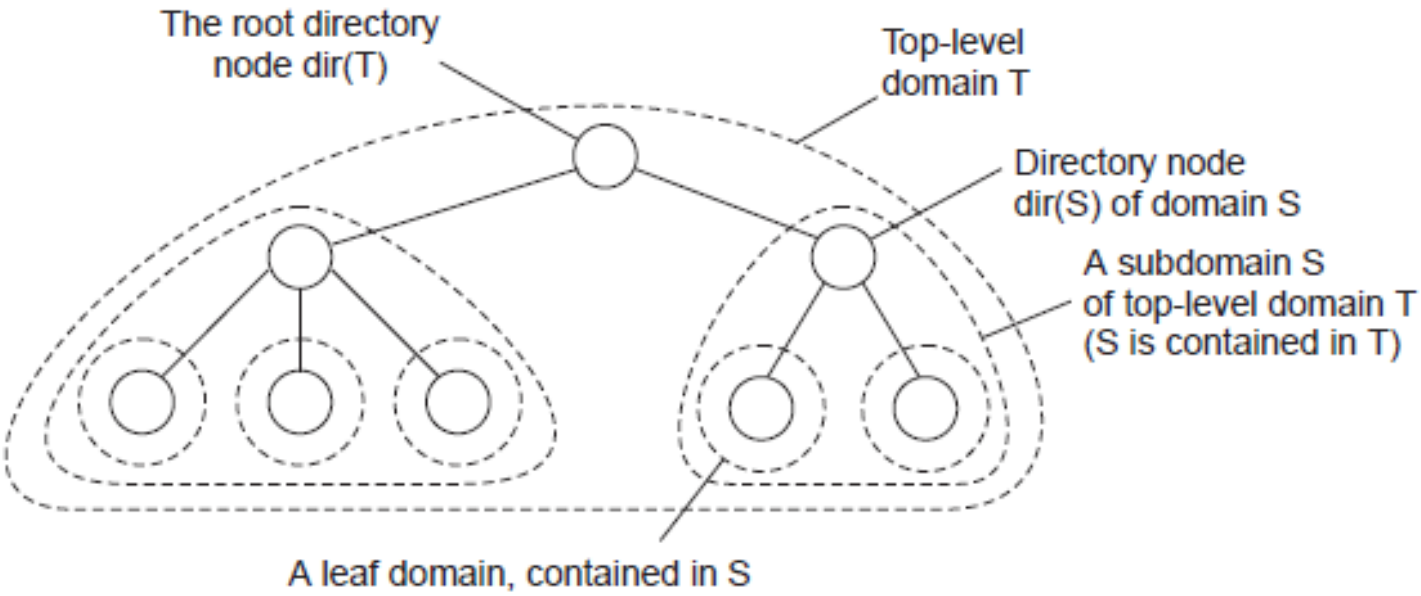
- In a hierarchical scheme, a network is divided into a collection of domains.
  - There is a single top-level domain that spans the entire network.
  - Each domain can be subdivided into multiple, smaller subdomains.
- A lowest-level domain, called a leaf domain, typically corresponds to a local-area network in a computer network or a cell in a mobile telephone network.

Directory nodes

- Each domain *D* has an associated directory node *dir(D)* that keeps track of the entities in that domain.

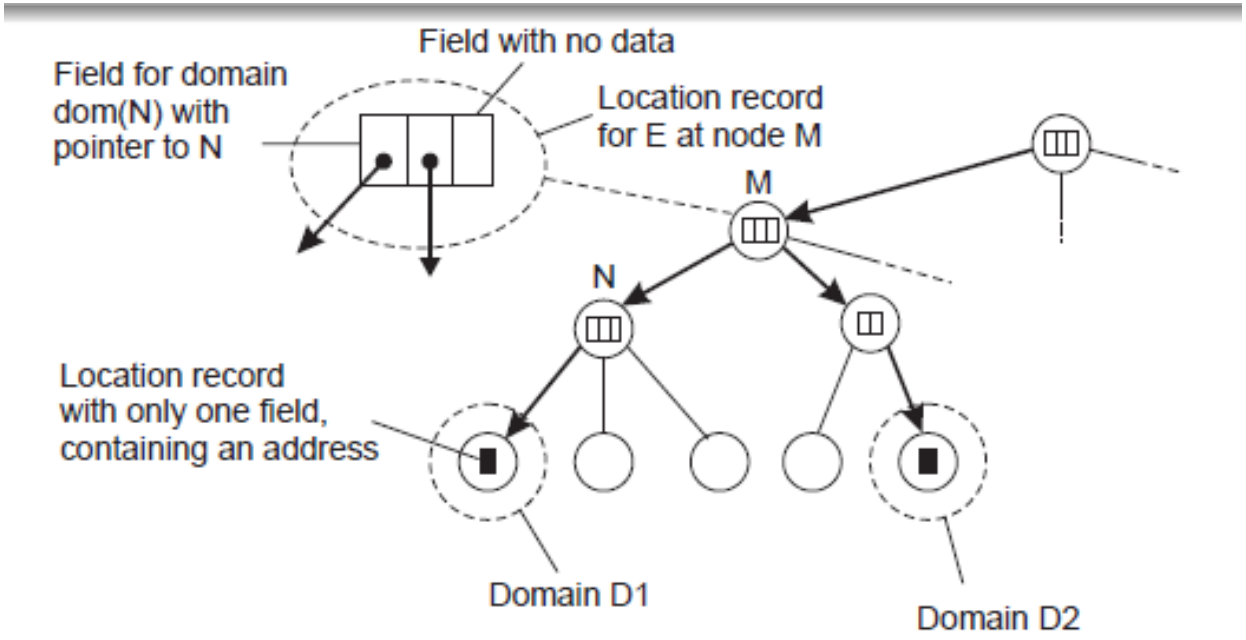
- This leads to a tree of directory nodes
  - The directory node of the top-level domain, called the root (directory) node, knows about all entities keeping for each of them a location record.

Directory nodes



HLS as a Tree

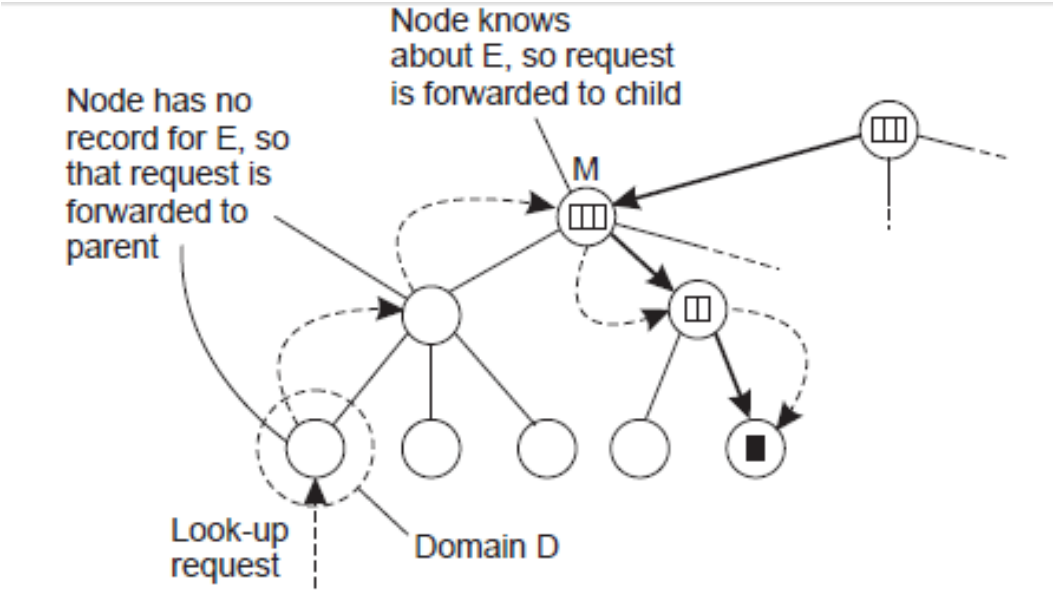
- Address of entity E is stored in a leaf or intermediate node
- Intermediate nodes contain a pointer to a child iff the subtree rooted at the child stores an address of the entity
- The root knows about all entities



HLS: Look-up

Basic principles

- Start lookup at local leaf node
- Node knows about E => follow downward pointer, else go up
- Upward lookup always stops at root



PART III - Naming

- Names, Identifiers, and addresses
- Flat Naming
- Structured Naming
- Attributed-based Naming

Structured Naming

- From Flat Naming towards Structured Naming

Structured Naming

- Flat names are good for machines, but are generally not very convenient for humans to use.
- As an alternative, naming systems generally support structured names that are composed from simple, human-readable names.
- Not only file naming, but also host naming on the Internet follow this approach.
- In this section, we concentrate on structured names and the way that these names are resolved to addresses.

Name Spaces

- Names are commonly organized into what is called a name space.

- Name spaces for structured names can be represented as a labeled directed graph with two types of nodes.
  - A **leaf node** represents a named entity and has the property that it has no outgoing edges.
    - A leaf node generally stores information on the entity it is representing
  - **A directory node** has a number of outgoing edges, each labeled with a name
    - Stores a table in which an outgoing edge is represented as a pair *(edge label, node identifier).* This is called directory table.

Naming Graph

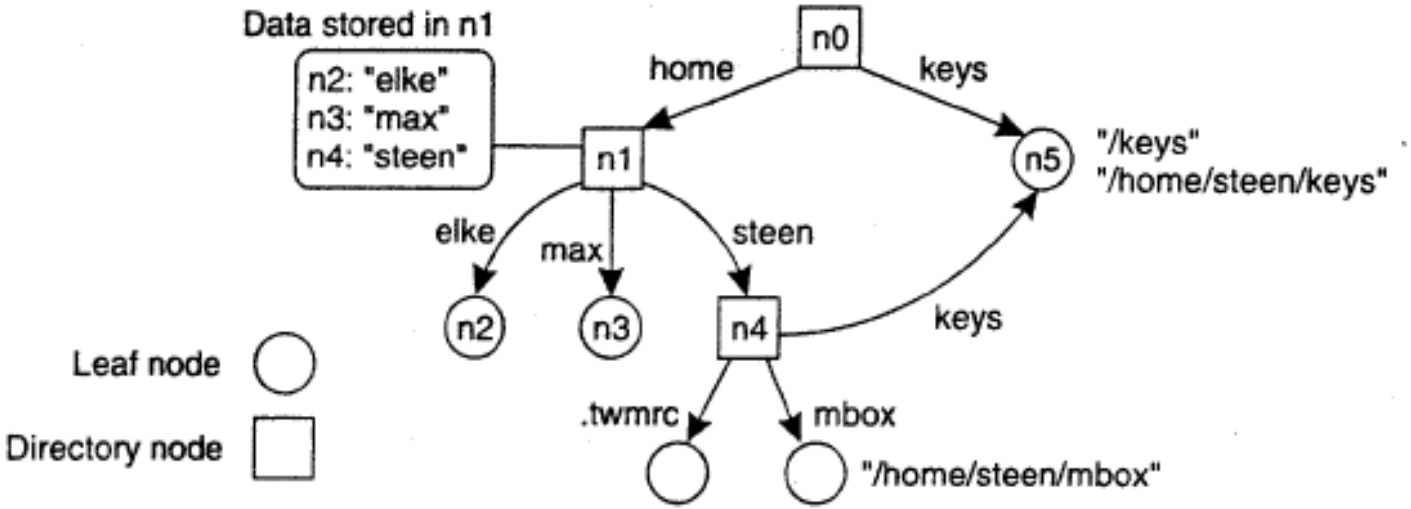


Figure 5-9. A general naming graph with a single root node.

Name Space

Observation

- We can easily store all kinds of attributes in a node, describing aspects of the entity the node represents:
  - Type of the entity
  - An identifier for that entity
  - Address of the entity's location
  - Nicknames
  - ...

Note

- Directory nodes can also have attributes, besides just storing a directory table with (edge label, node identifier) pairs.

Name resolution

Problem

To resolve a name we need a directory node. How do we actually find that (initial) node?

Closure mechanism

- Name resolution can take place only if we know how and where to start.
- Knowing how and where to start name resolution is generally referred to as a closure mechanism.
  - www.cs.vu.nl: start at a DNS name server
  - /home/steen/mbox: start at the local NFS file server (possible recursive search)

Mounting remote name spaces

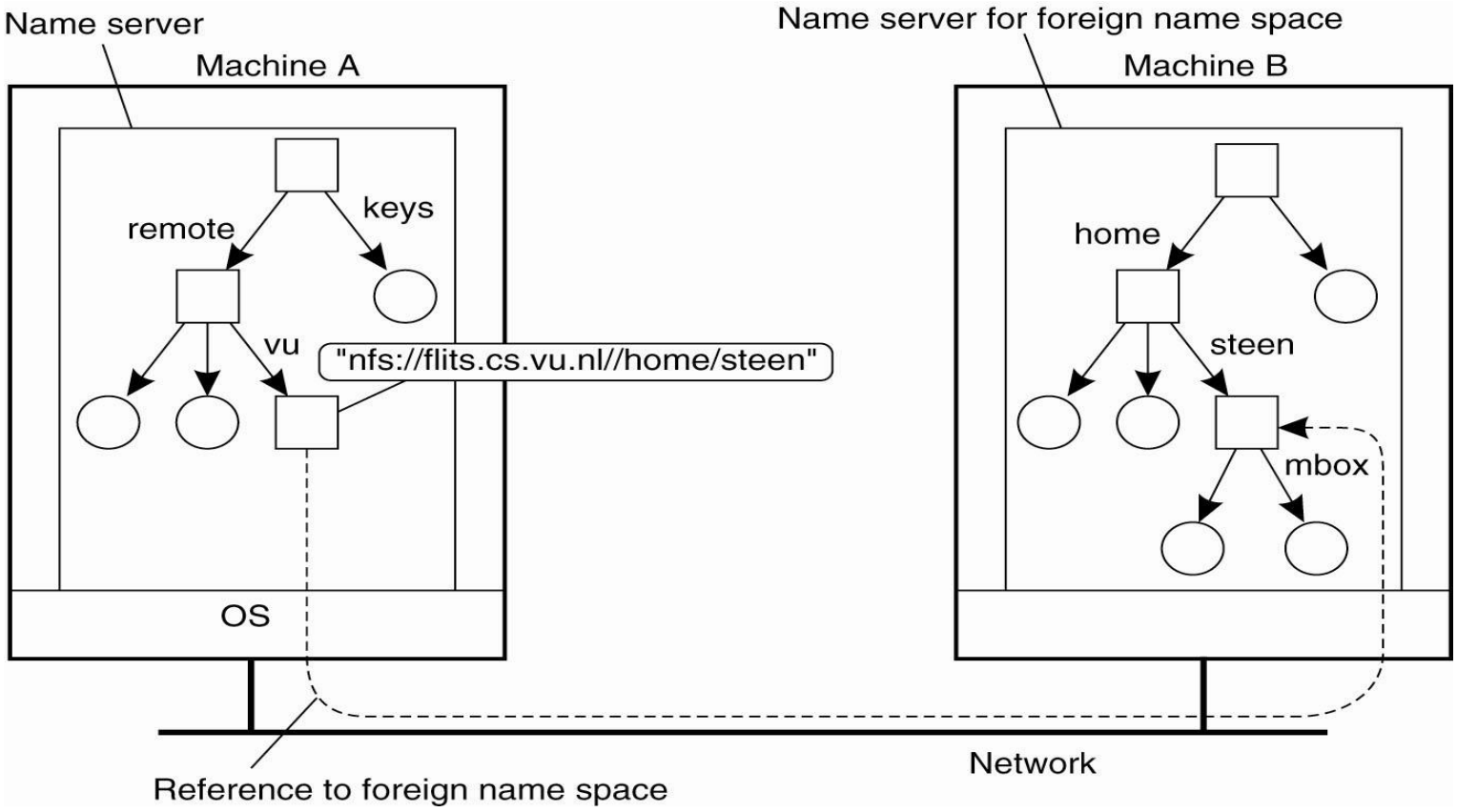


Figure 5-12. Mounting remote name spaces through a specific access protocol.

Name Space Distribution

- *Concept: distribute the name resolution process as well as name space management across multiple machines, by distributing nodes of the naming graph.*

- As before, assume such a name space has only a single root node.
- To effectively implement such a name space, it is convenient to partition it into logical layers.
  - Global level
  - Administrational level
  - Managerial level

Name-space implementation

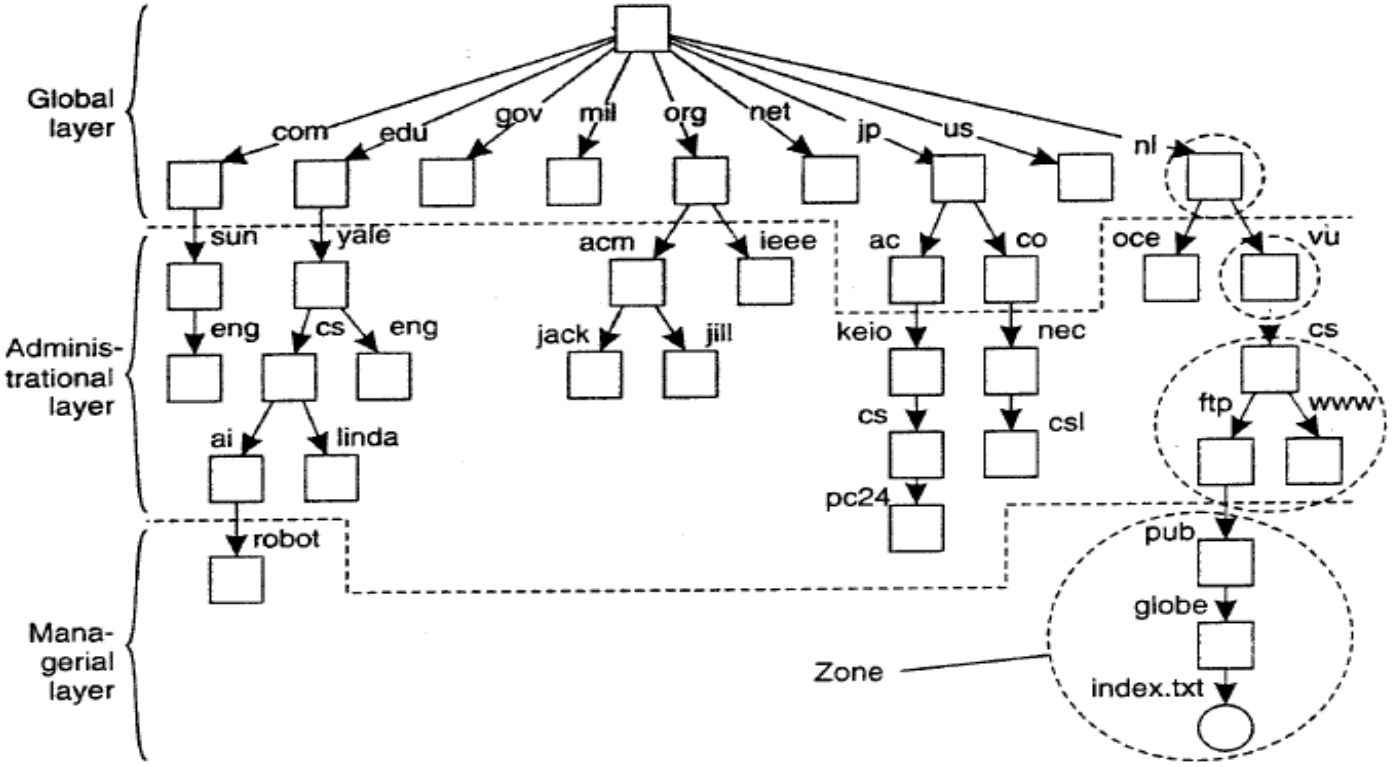


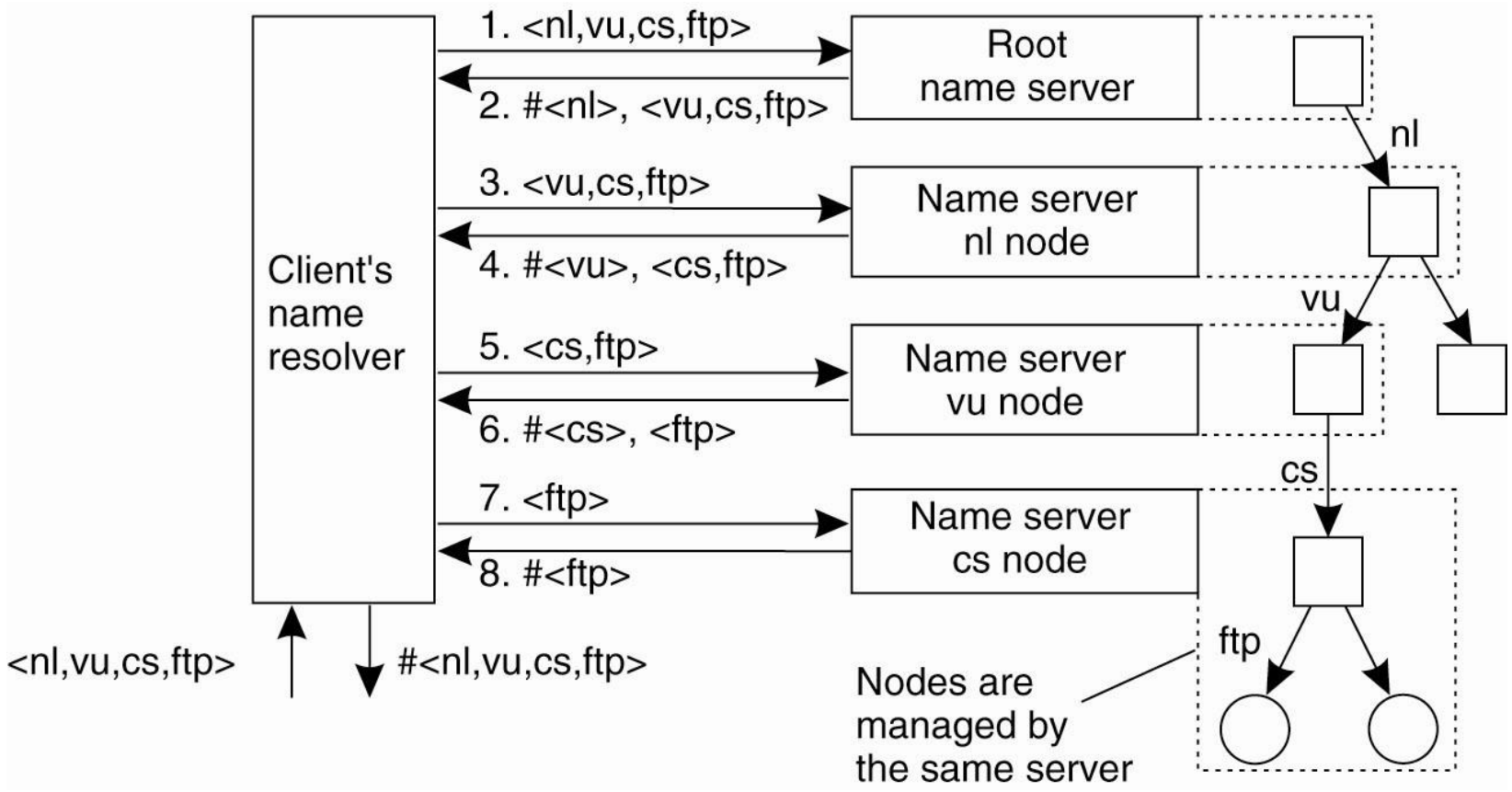
Figure 5-13. An example partitioning of the DNS name space, including Internet-accessible files, into three layers.

Name-space implementation

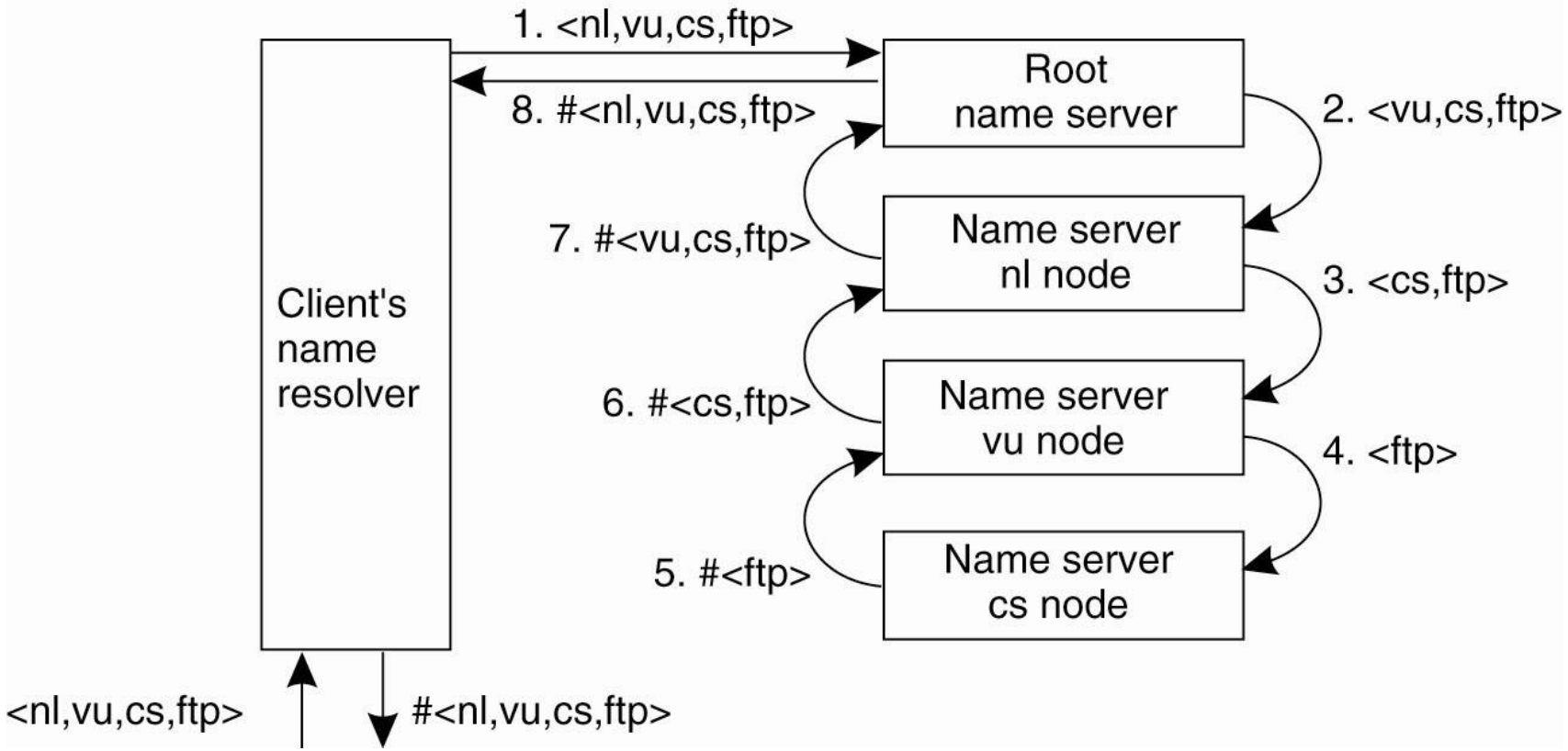
Item	Global	Administrational	Managerial
Geographical scale of network	Worldwide	Organization	Department
Total number of nodes	Few	Many	Vast numbers
Responsiveness to lookups	Seconds	Milliseconds	Immediate
Update propagation	Lazy	Immediate	Immediate
Number of replicas	Many	None or few	None
Is client-side caching applied?	Yes	Yes	Sometimes

A comparison between name servers for implementing nodes from a large-scale name space partitioned into a global layer, an administrational layer, and a managerial layer.

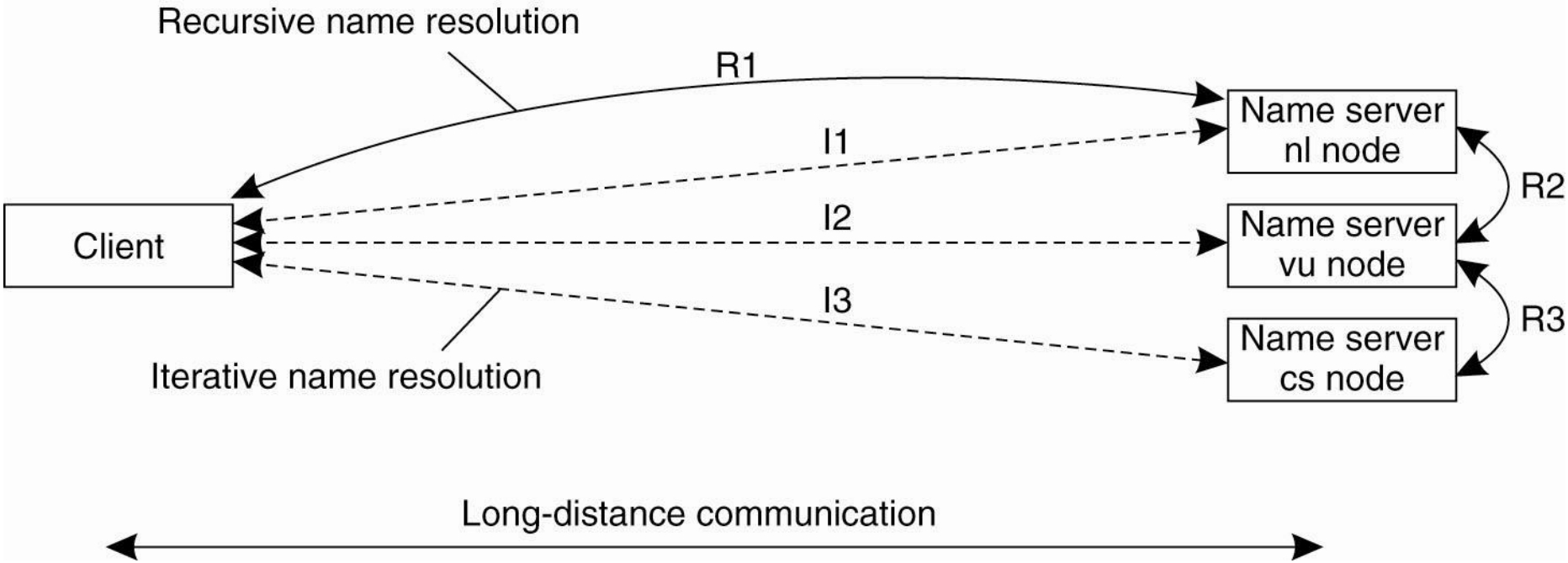
Iterative name resolution



Recursive name resolution



Example: The Domain Name System



The comparison between recursive and iterative name resolution with respect to communication costs.

Caching in name servers

Server for node	Should resolve	Looks up	Passes to child	Receives and caches	Returns to requester
cs	<ftp>	#<ftp>	—	—	#<ftp>
vu	<cs,ftp>	#<cs>	<ftp>	#<ftp>	#<cs> #<cs, ftp>
nl	<vu,cs,ftp>	#<vu>	<cs,ftp>	#<cs> #<cs,ftp>	#<vu> #<vu,cs> #<vu,cs,ftp>
root	<nl,vu,cs,ftp>	#<nl>	<vu,cs,ftp>	#<vu> #<vu,cs> #<vu,cs,ftp>	#<nl> #<nl,vu> #<nl,vu,cs> #<nl,vu,cs,ftp>

Recursive name resolution of <*nl, vu, cs, ftp*>. Name servers cache intermediate results for subsequent lookups.

DNS

- One of the largest distributed naming services in use today is the Internet Domain Name System (DNS).
- The DNS name space is hierarchically organized as a rooted tree. A label is a case-insensitive string made up of alphanumeric characters.
  - A label has a maximum length of 63 characters; the length of a complete path name is restricted to 255 characters.
- A subtree is called a domain
- The contents of a node is formed by a collection of resource records.

DNS: Resource Records

Type of record	Associated entity	Description
SOA	Zone	Holds information on the represented zone
A	Host	Contains an IP address of the host this node represents
MX	Domain	Refers to a mail server to handle mail addressed to this node
SRV	Domain	Refers to a server handling a specific service
NS	Zone	Refers to a name server that implements the represented zone
CNAME	Node	Symbolic link with the primary name of the represented node
PTR	Host	Contains the canonical name of a host
HINFO	Host	Holds information on the host this node represents
TXT	Any kind	Contains any entity-specific information considered useful

DNS Implementation (1)

Name	Record type	Record value
cs.vu.nl.	SOA	star.cs.vu.nl. hostmaster.cs.vu.nl. 2005092900 7200 3600 2419200 3600
cs.vu.nl.	TXT	"Vrije Universiteit - Math. & Comp. Sc."
cs.vu.nl.	MX	1 mail.few.vu.nl.
cs.vu.nl.	NS	ns.vu.nl.
cs.vu.nl.	NS	top.cs.vu.nl.
cs.vu.nl.	NS	solo.cs.vu.nl.
cs.vu.nl.	NS	star.cs.vu.nl.
star.cs.vu.nl.	A	130.37.24.6
star.cs.vu.nl.	A	192.31.231.42
star.cs.vu.nl.	MX	1 star.cs.vu.nl.
star.cs.vu.nl.	MX	666 zephyr.cs.vu.nl.
star.cs.vu.nl.	HINFO	"Sun" "Unix"
zephyr.cs.vu.nl.	A	130.37.20.10
zephyr.cs.vu.nl.	MX	1 zephyr.cs.vu.nl.
zephyr.cs.vu.nl.	MX	2 tornado.cs.vu.nl.
zephyr.cs.vu.nl.	HINFO	"Sun" "Unix"

An excerpt from the DNS database for the zone *cs.vu.nl*.

DNS Implementation (2)

ftp.cs.vu.nl.	CNAME	soling.cs.vu.nl.
www.cs.vu.nl.	CNAME	soling.cs.vu.nl.
soling.cs.vu.nl.	A	130.37.20.20
soling.cs.vu.nl.	MX	1 soling.cs.vu.nl.
soling.cs.vu.nl.	MX	666 zephyr.cs.vu.nl.
soling.cs.vu.nl.	HINFO	"Sun" "Unix"
vucs-das1.cs.vu.nl.	PTR	0.198.37.130.in-addr.arpa.
vucs-das1.cs.vu.nl.	A	130.37.198.0
inkt.cs.vu.nl.	HINFO	"OCE" "Proprietary"
inkt.cs.vu.nl.	A	192.168.4.3
pen.cs.vu.nl.	HINFO	"OCE" "Proprietary"
pen.cs.vu.nl.	A	192.168.4.2
localhost.cs.vu.nl.	A	127.0.0.1

PART III - Naming

- Names, Identifiers, and addresses
- Flat Naming
- Structured Naming
- Attributed-based Naming

ATTRIBUTE-BASED NAMING

Observation

- In many cases, it is much more convenient to name, and look up entities by means of their attributes => traditional directory services (ex. yellow pages).

Problem

- Lookup operations can be extremely expensive, as they require to match requested attribute values, against actual attribute values => inspect all entities (in principle).

Solution

- Implement basic directory service as database, and combine with traditional structured naming system.

LDAP

- A common approach to tackling distributed directory services is to combine structured naming with attribute-based naming.

- This approach has been widely adopted, for example, in Microsoft's Active Directory service and other systems.

- Many of these systems use, or rely on the lightweight directory access protocol commonly referred simply as LDAP.

LDAP

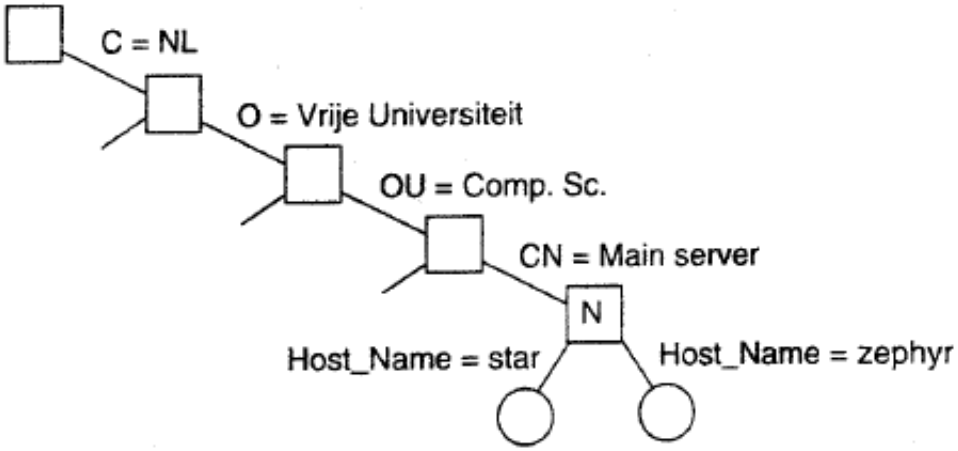
- Conceptually, an LDAP directory service consists of a number of records, usually referred to as directory entries.
- A directory entry is comparable to a resource record in DNS.
- Each record is made up of a collection of *(attribute. value)* pairs, where each attribute has an associated type.

LDAP Directory

Attribute	Abbr.	Value
Country	C	NL
Locality	L	Amsterdam
Organization	O	Vrije Universiteit
OrganizationalUnit	OU	Comp. Sc.
CommonName	CN	Main server
Mail_Servers	—	137.37.20.3, 130.37.24.6, 137.37.20.10
FTP_Server	—	130.37.20.20
WWW_Server	—	130.37.20.20

Figure 5-22. A simple example of an LDAP directory entry using LDAP naming conventions.

Example: LDAP



(a)

Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	star
Host_Address	192.31.231.42

Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	zephyr
Host_Address	137.37.20.10

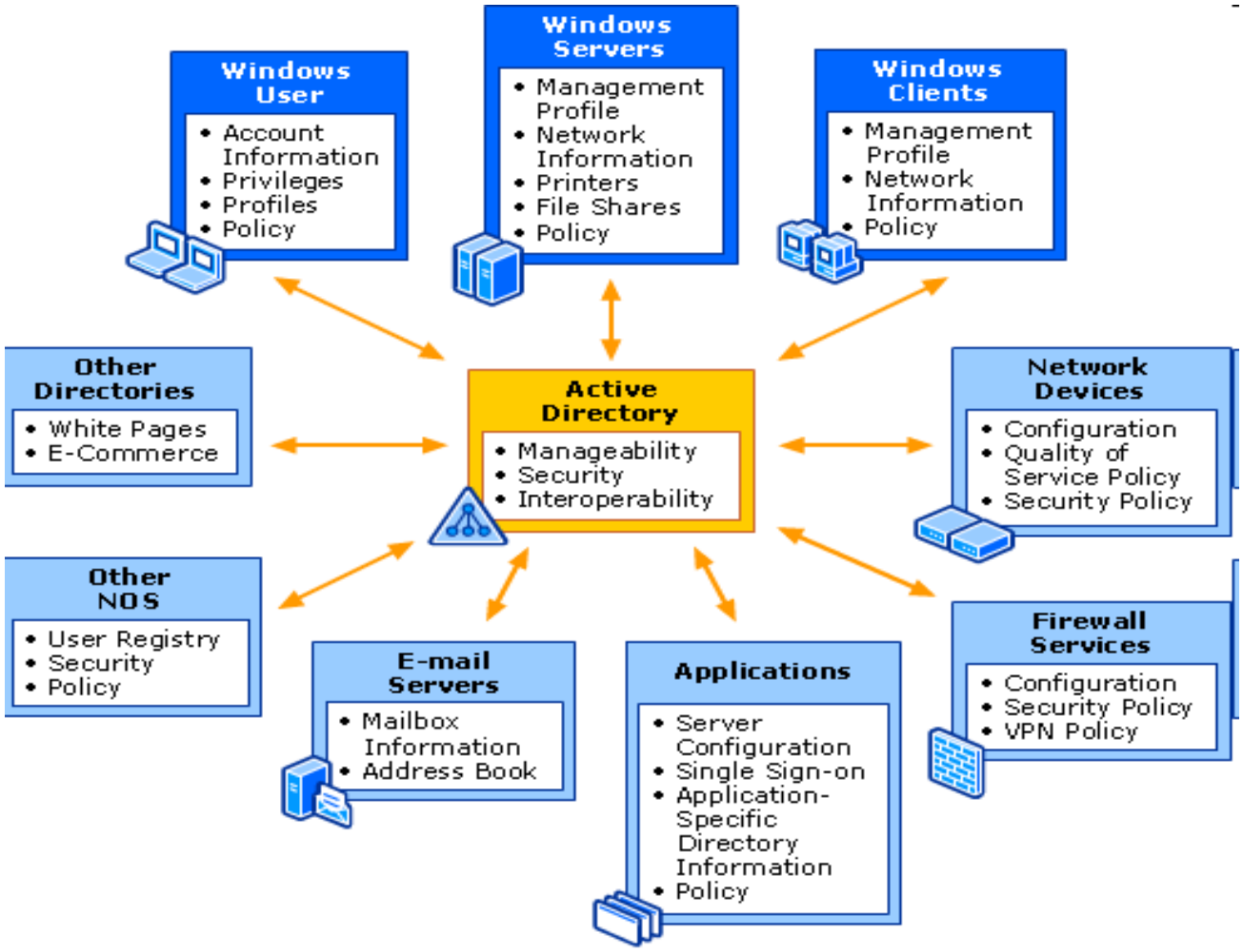
(b)

Figure 5-23. (a) Part of a directory information tree. (b) Two directory entries having *Host_Name* as RDN.

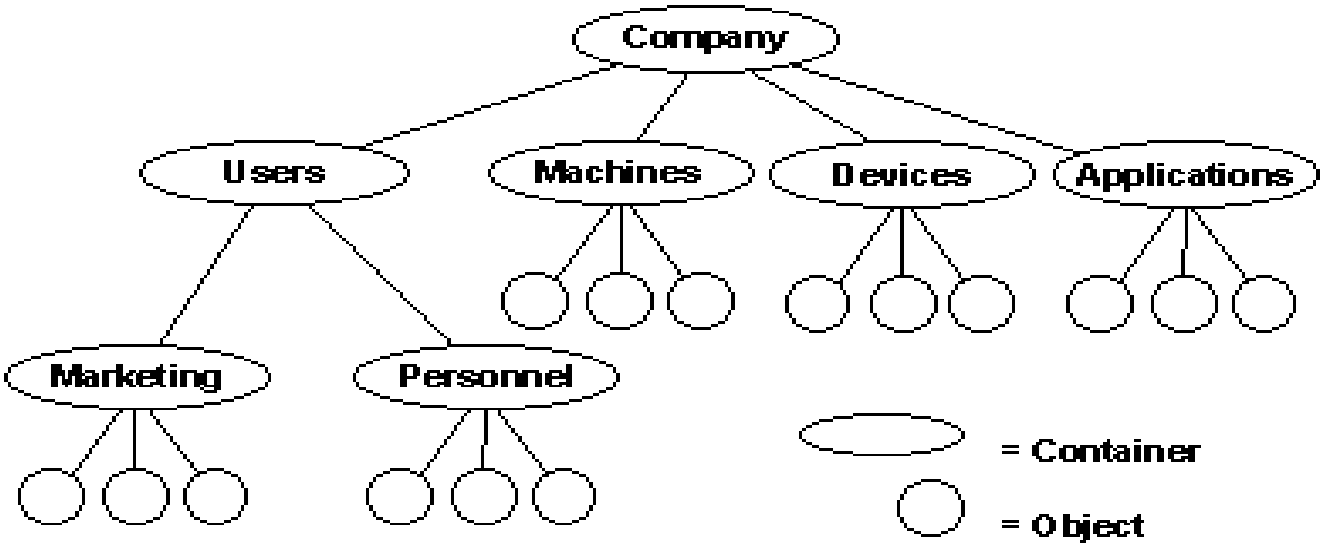
Active Directory

- Active Directory is a technology created by Microsoft that provides a variety of network services, including:
  - LDAP-like directory services
  - Kerberos-based authentication
  - DNS-based naming and other network information
  - Central location for network administration and delegation of authority
  - Information security and single sign-on for user access to networked based resources
  - The ability to scale up or down easily
  - Central storage location for application data
  - Synchronization of directory updates amongst several servers

Active Directory

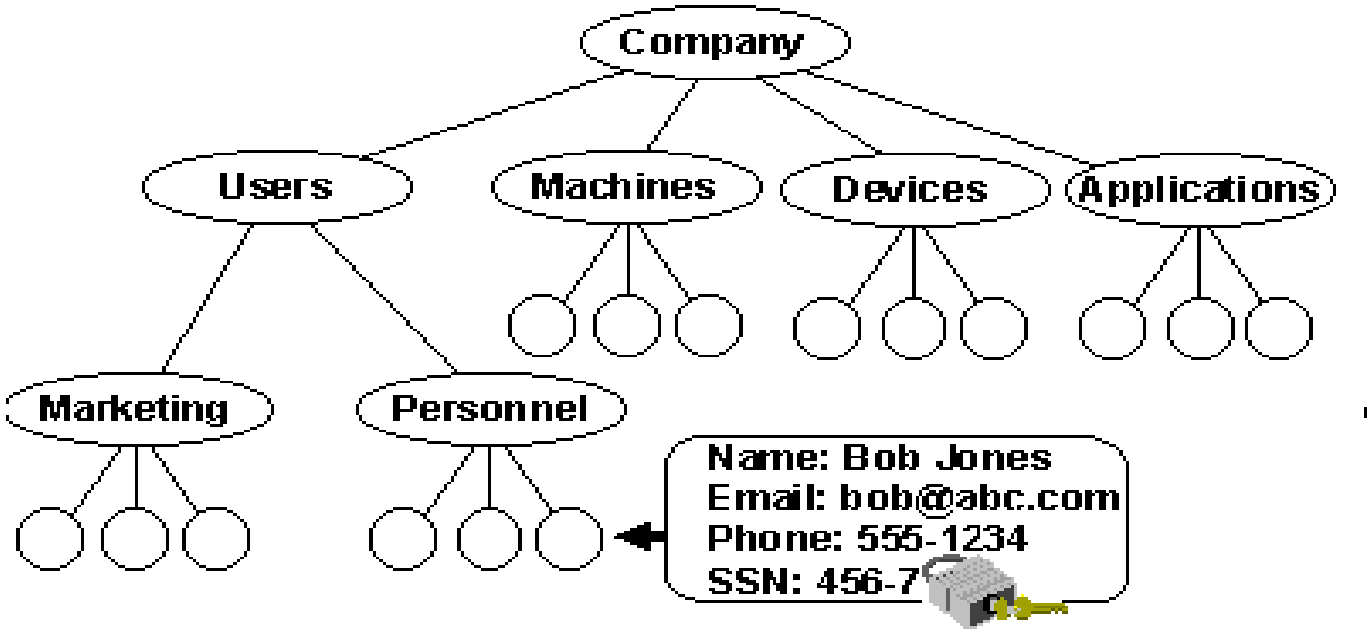


Hierarchical organization of AD



Active Directory organizes information hierarchically to ease network use and management

Object-oriented storage in AD



Active Directory objects and attributes are protected by access control lists.

End of PART III

- Readings
  - Distributed Systems, Chapter 5.