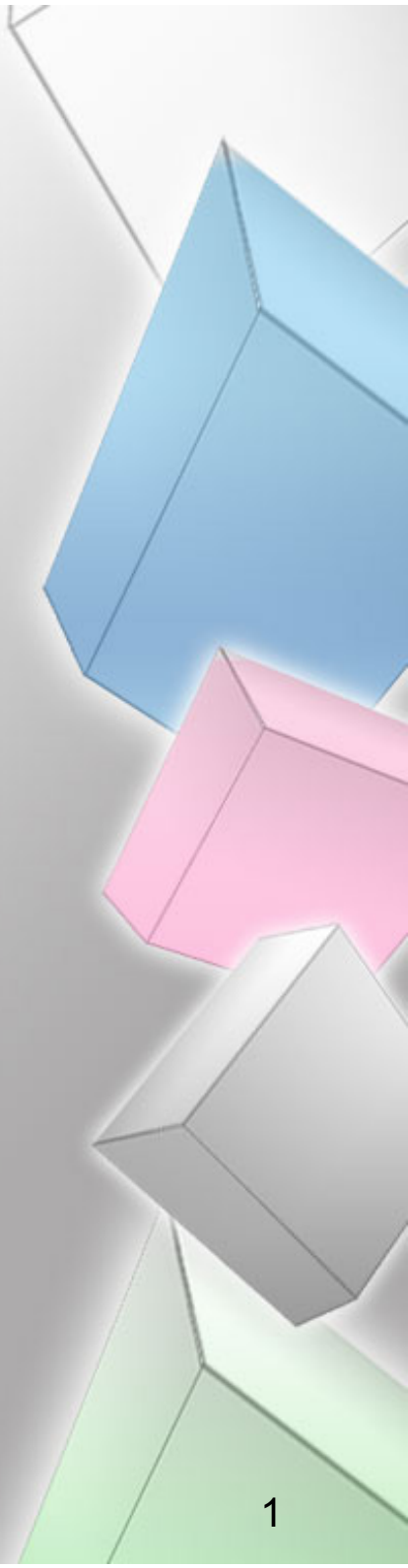
Advanced Topics in Operating Systems

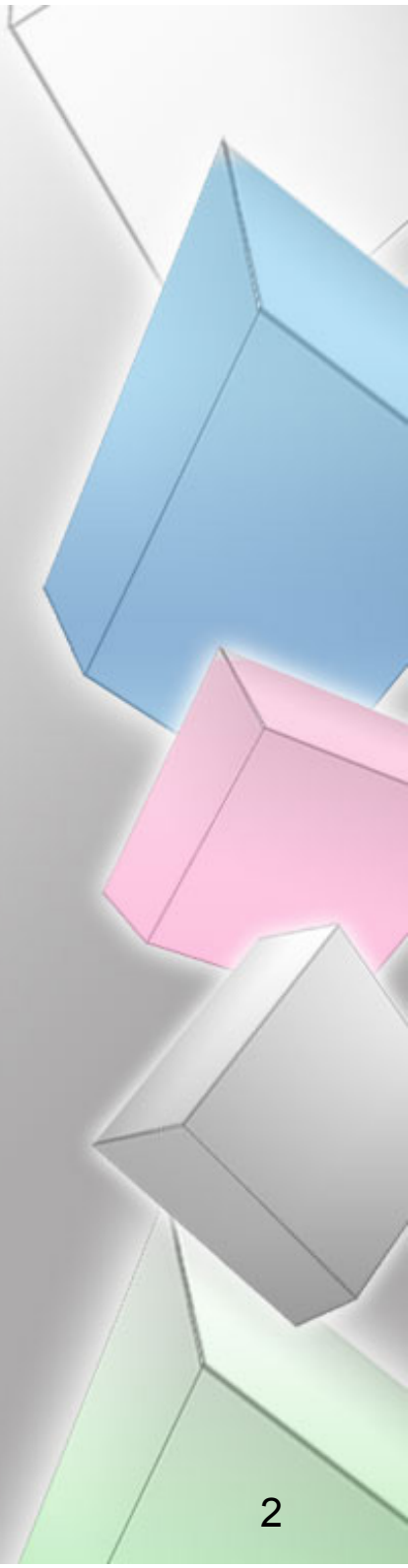
MSc in Computer Science
UNYT-UoG

Dr. Marenglen Biba
18-19-20 December 2009



Lesson 2

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

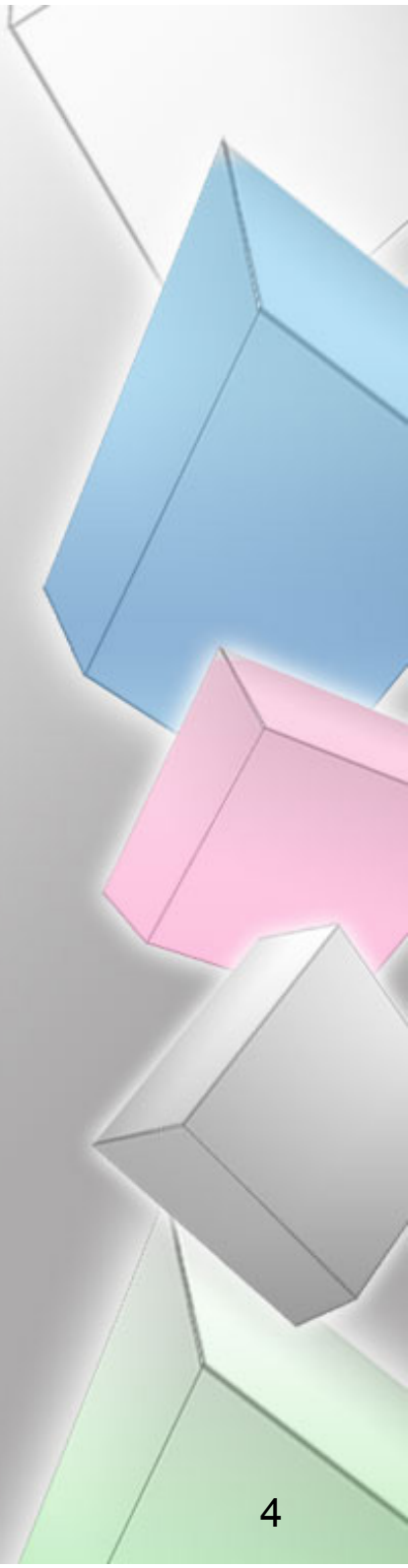


Names

- 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.
- The difference between naming in distributed systems and non distributed systems lies in the way naming systems are implemented.

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.
- Typical examples include resources such as hosts, printers, disks, and files. Other well-known examples of entities that are often explicitly named are processes, users, mailboxes, newsgroups, Web pages, graphical windows, messages, network connections, and so on.
- 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
  - The address of an access point of an entity is also simply called an address of that entity

Access Points

- An entity can offer more than one access point.
  - As a comparison, a telephone can be viewed as an access point of a person, whereas the telephone number corresponds to an address. Indeed, many people nowadays have several telephone numbers, each number corresponding to a point where they can be reached.

- 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.
  - Likewise, when a person moves to another city or country, it is often necessary to change telephone numbers as well.
  - In a similar fashion, changing jobs or Internet Service Providers, means changing your e-mail address.

Location Independent

- If an entity offers more than one access point, it is not clear which address to use as a reference.
  - For instance, many organizations distribute their Web service across several servers.
- If we would use the addresses of those servers as a reference for the Web service, it is not obvious which address should be chosen as the best one.

- Again, a much better solution is to have a single name for the Web service independent from the addresses of the different Web servers.
  - This means that a name for an entity that is independent from its addresses is often much easier and more flexible to use.
- 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)

Observation

- An identifier need not necessarily be a pure name, i.e., it may have content.

Addresses and Identifiers

- Addresses and identifiers are two important types of names that are each used for very different purposes.
  - In many computer systems, addresses and identifiers are represented in machine-readable form only, that is, in the form of bit strings.
  - For example, an Ethernet address is essentially a random string of 48 bits.
  - Likewise memory addresses are typically represented as 32-bit or 64-bit strings.
- Another important type of name is that which is tailored to be used by humans, also referred to as human-friendly names.
- In contrast to addresses and identifiers, a human-friendly name is generally represented as a character string.
  - These names appear in many different forms. For example, files in UNIX systems have character-string names that can be as long as 255 characters, and which are defined entirely by the user.
  - Similarly, DNS names are represented as relatively simple case-insensitive character strings.

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

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.

Broadcasting

- Broadcasting is used in the Internet Address Resolution Protocol (ARP) to find the data-link address of a machine when given only an IP address.
  - In essence, a machine broadcasts a packet on the local network asking who is the owner of a given IP address.
  - When the message arrives at a machine, the receiver checks whether it should listen to the requested IP address. If so, it sends a reply packet containing, for example, its Ethernet address.
- Broadcasting becomes inefficient when the network grows.
  - Not only is network bandwidth wasted by request messages, but, more seriously, too many hosts maybe interrupted by requests they cannot answer.
- One possible solution is to switch to multicasting, by which only a restricted group of hosts receives the request.
  - For example, Ethernet networks support data-link level multicasting directly in hardware.

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 provides a best-effort service to deliver that message to all group members.

Multicasting

- A multicast address can be used as a general location service for multiple entities.
  - For example, consider an organization where each employee has his or her own mobile computer.
- When such a computer connects to the locally available network:
  - it is dynamically assigned an IP address.
  - In addition, it joins a specific multicast group.
- When a process wants to locate computer *A*, it sends a "where is *A?*" request to the multicast group.
  - If *A* is connected, it responds with its current IP address.

Nearest Replica

- Another way to use a multicast address is to associate it with a replicated entity, and to use multicasting to locate the *nearest* replica.
- When sending a request to the multicast address, each replica responds with its current (normal) IP address.
- A crude way to select the nearest replica is to choose the one whose reply comes in first.
- However, selecting a nearest replica is generally not that easy!!! ☹

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.

Forwarding Pointers

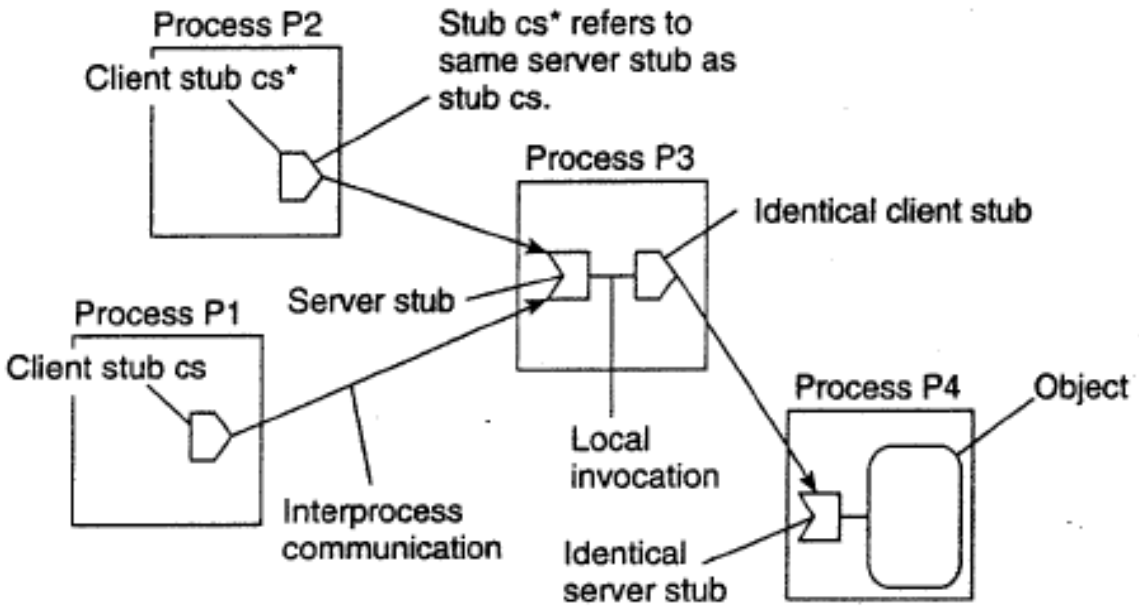


Figure 5-1. The principle of forwarding pointers using *(client stub, server stub)* pairs.

- A server stub contains either a local reference to the actual object or a local reference to a remote client stub for that object.
- Whenever an object moves from address space *A* to *B*, it leaves behind a client stub in its place in *A* and installs a server stub that refers to it in *B*.
- An interesting aspect of this approach is that migration is completely transparent to a client.
- The only thing the client sees of an object is a client stub.

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

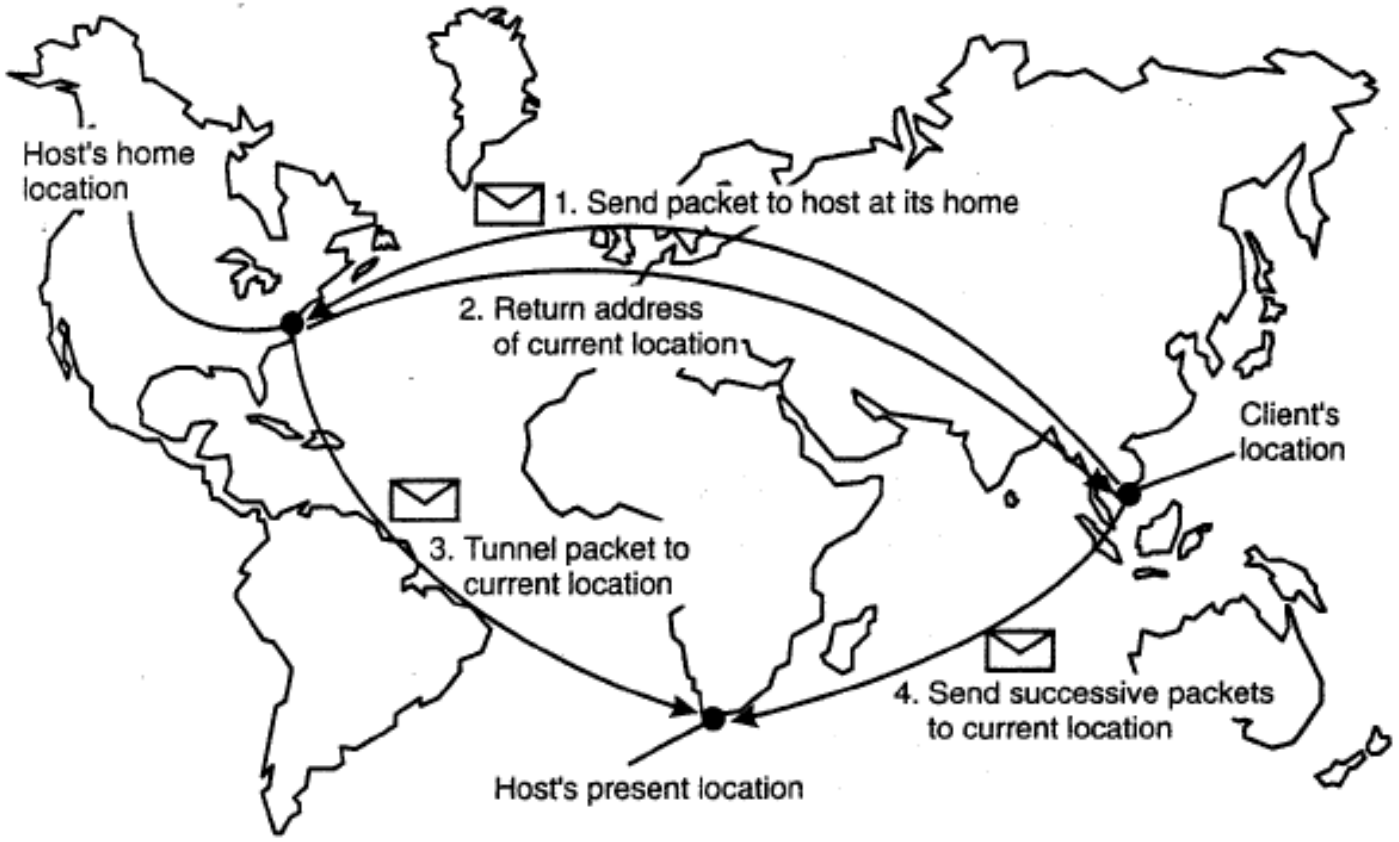


Figure 5-3. The principle of Mobile IP.

Home-based approaches

Two-tiered scheme

Keep track of visiting entities:

- Check local visitor register first
- Fall back to home location if local lookup fails

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)

Chord

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).

Nonsolution

- Let node id keep track of succ(id) and start linear search along the ring.

DHTs: Finger Tables

Principle

- Each node *p* maintains a finger table $FT_p[]$ with at most m entries:
  - $FT_p[i] = succ(p+2^{i-1})$
- Note: $FT_p[i]$ points to the first node succeeding *p* by at least 2^{i-1}.
- To look up a key *k*, node *p* forwards the request to node q with index *j* satisfying

$$q = FT_p[j] \leq k < FT_p[j+1]$$

- If $p < k < FT_p[1]$, the request is also forwarded to $FT_p[1]$.

DHTs: Finger Tables

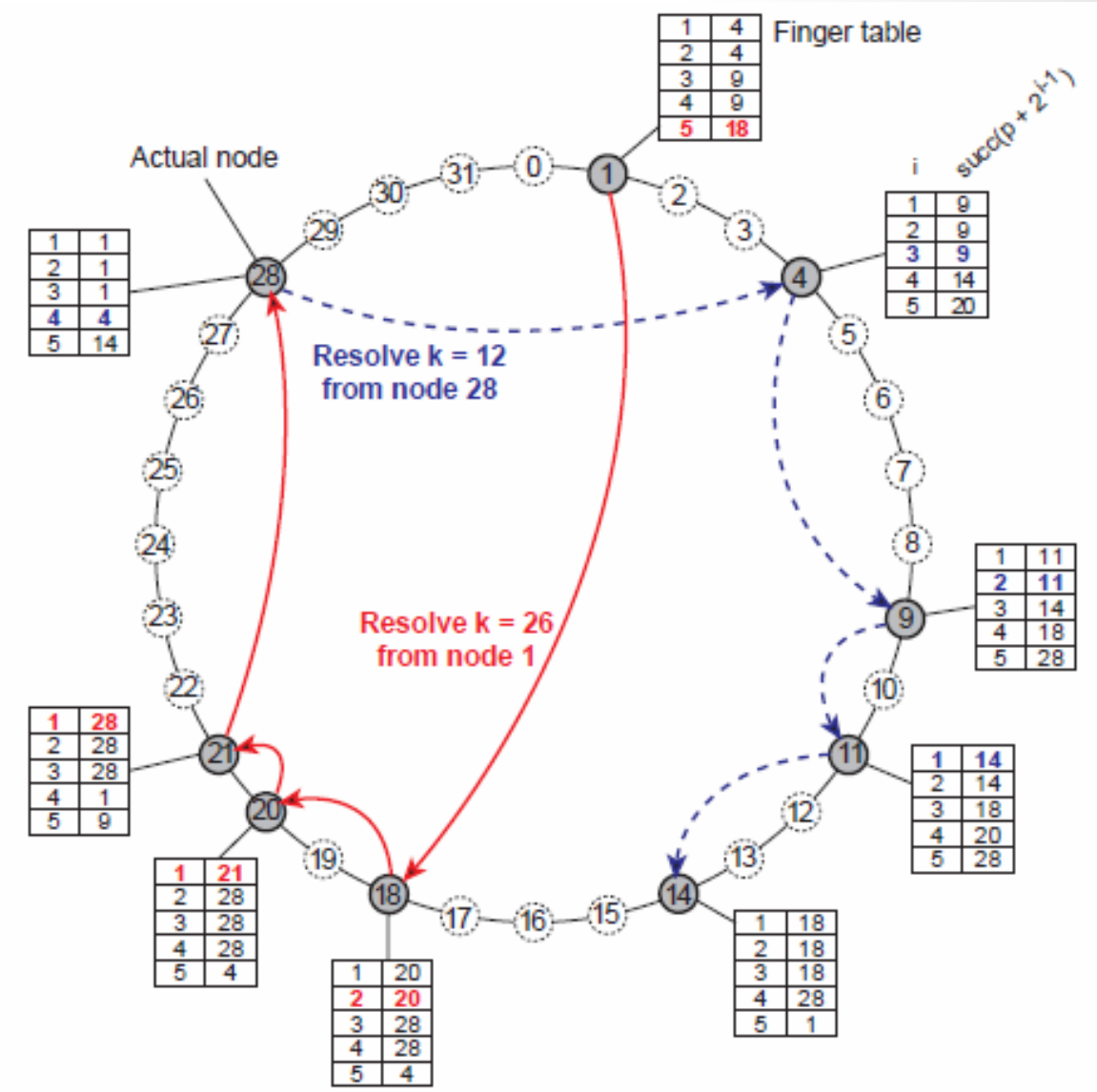


Figure 5-4. Resolving key 26 from node 1 and key 12 from node 28 in a Chord system.

Exploiting network proximity

Problem

- The logical organization of nodes in the overlay may lead to erratic message transfers in the underlying Internet: node k and node succ(k +1) may be very far apart.

- For example, assume that node 1 is placed in Amsterdam, The Netherlands; node 18 in San Diego, California; node 20 in Amsterdam again; and node 21 in San Diego.
  - The result of resolving key 26 will then incur three wide-area message transfers which arguably could have been reduced to at most one. To minimize these pathological cases, designing a DHT-based system requires taking the underlying network into account.

Awareness of the network

- Topology-aware node assignment: When assigning an ID to a node, make sure that nodes close in the ID space are also close in the network. Can be very difficult!!!

- Proximity routing: Maintain more than one possible successor, and forward to the closest.
  - Example: in Chord $FT_p[i]$ points to first node in INT = $[p+2^{i-1};p+2^{i-1}]$. Node *p* can also store pointers to other nodes in INT.

- Proximity neighbor selection: When there is a choice of selecting who your neighbor will be (not in Chord), pick the closest one.

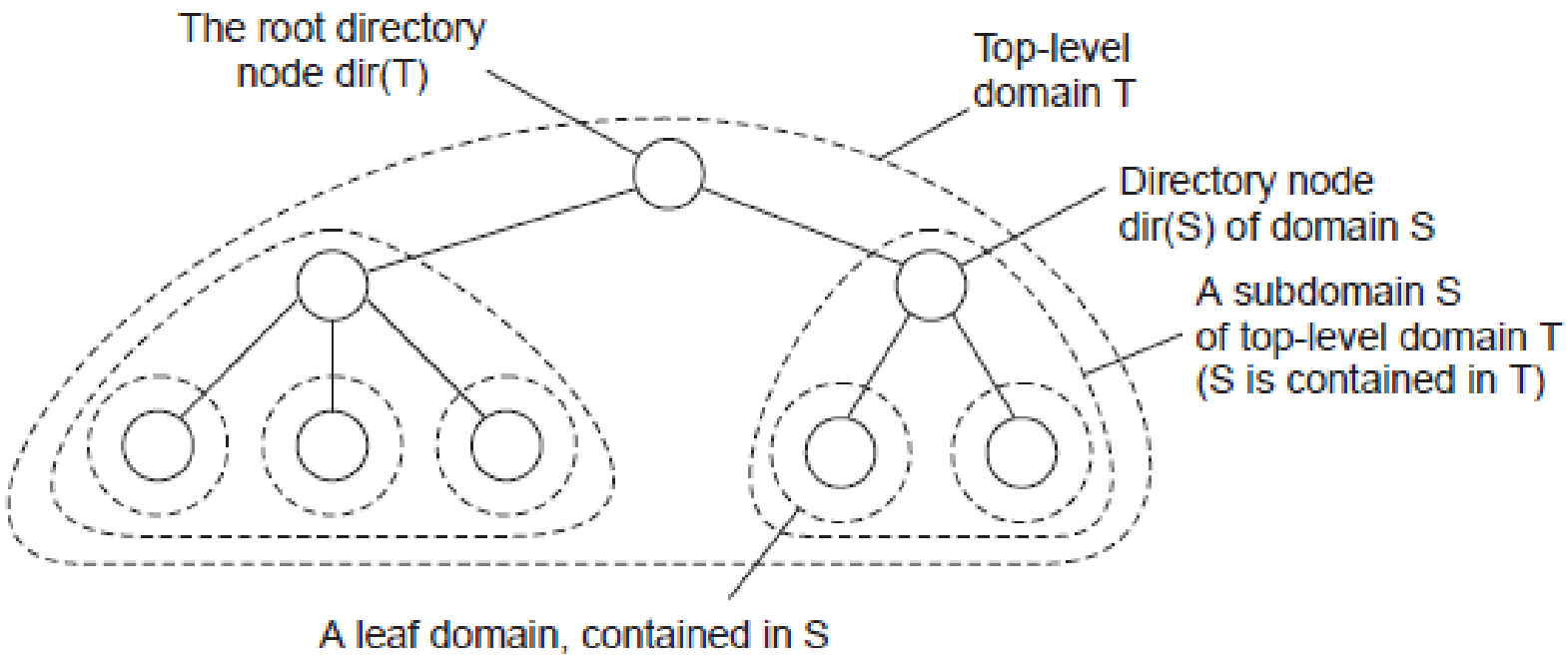
Iterative and Recursive Lookup

- Finally, we also note that a distinction can be made between iterative and recursive lookups.
- In iterative mode, a node that is requested to look up a key will return the network address of the next node found to the requesting process.
- The process will then request that next node to take another step in resolving the key.
- An alternative, and essentially the way that we have explained it so far, is to let a node forward a lookup request to the next node.
- Both approaches have their advantages and disadvantages, which we explore later in this chapter.

Hierarchical Location Services (HLS)

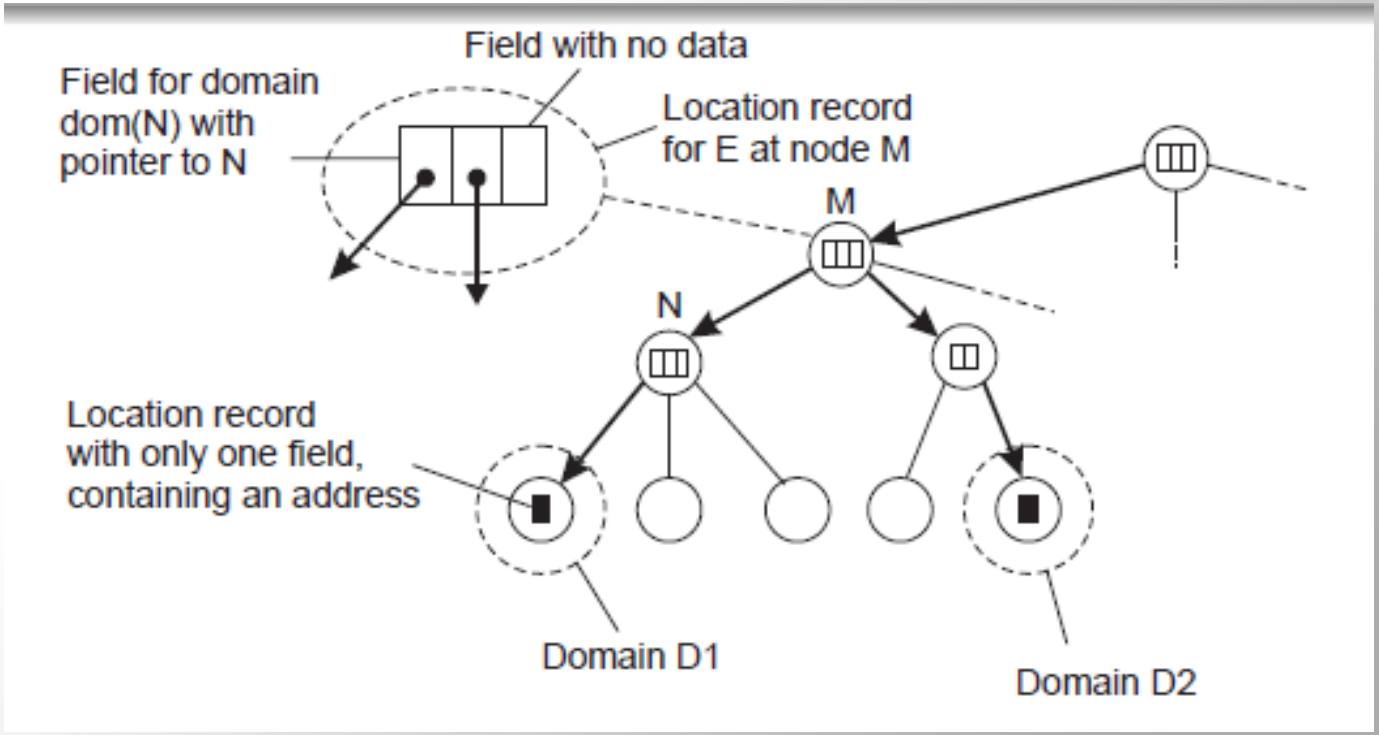
- In a hierarchical scheme, a network is divided into a collection of domains.
  - There is a single top-level domain that spans the entire network.
  - Each domain can be subdivided into multiple, smaller subdomains.
- A lowest-level domain, called a leaf domain, typically corresponds to a local-area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

HLS


The root directory
node dir(T)
Top-level
domain T
Directory node
dir(S) of domain S
A subdomain S
of top-level domain T
(S is contained in T)
A leaf domain, contained in S

HLS as a Tree

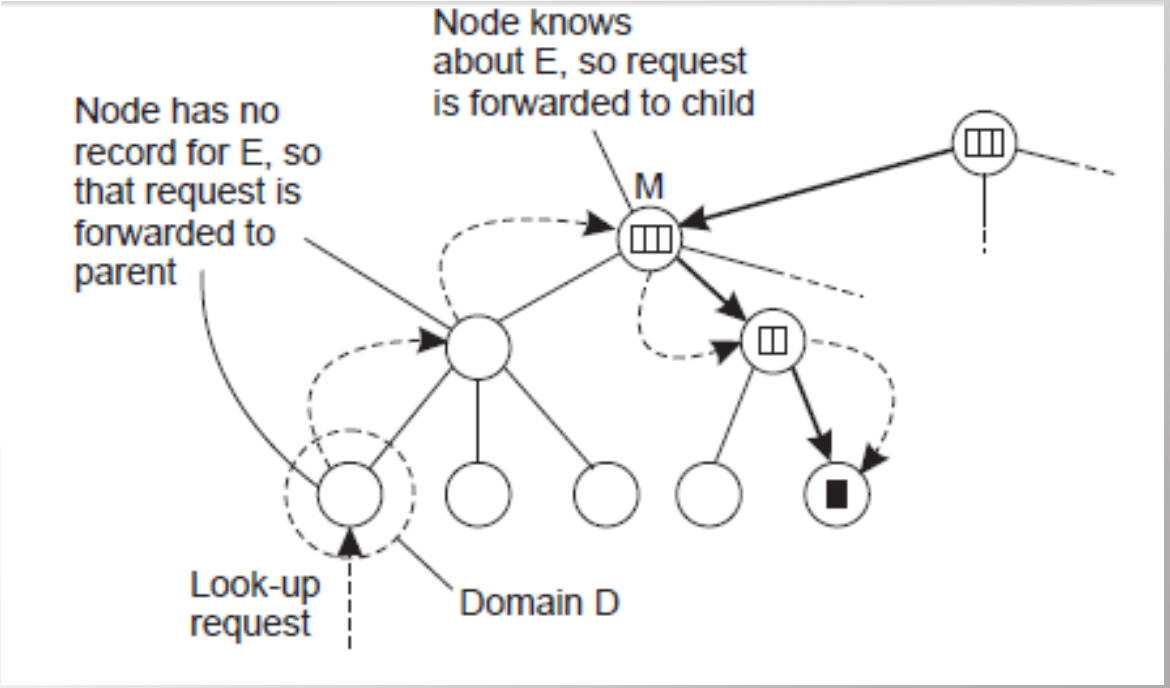
- Address of entity E is stored in a leaf or intermediate node
- Intermediate nodes contain a pointer to a child iff the subtree rooted at the child stores an address of the entity
- The root knows about all entities



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



HLS: Insert operation

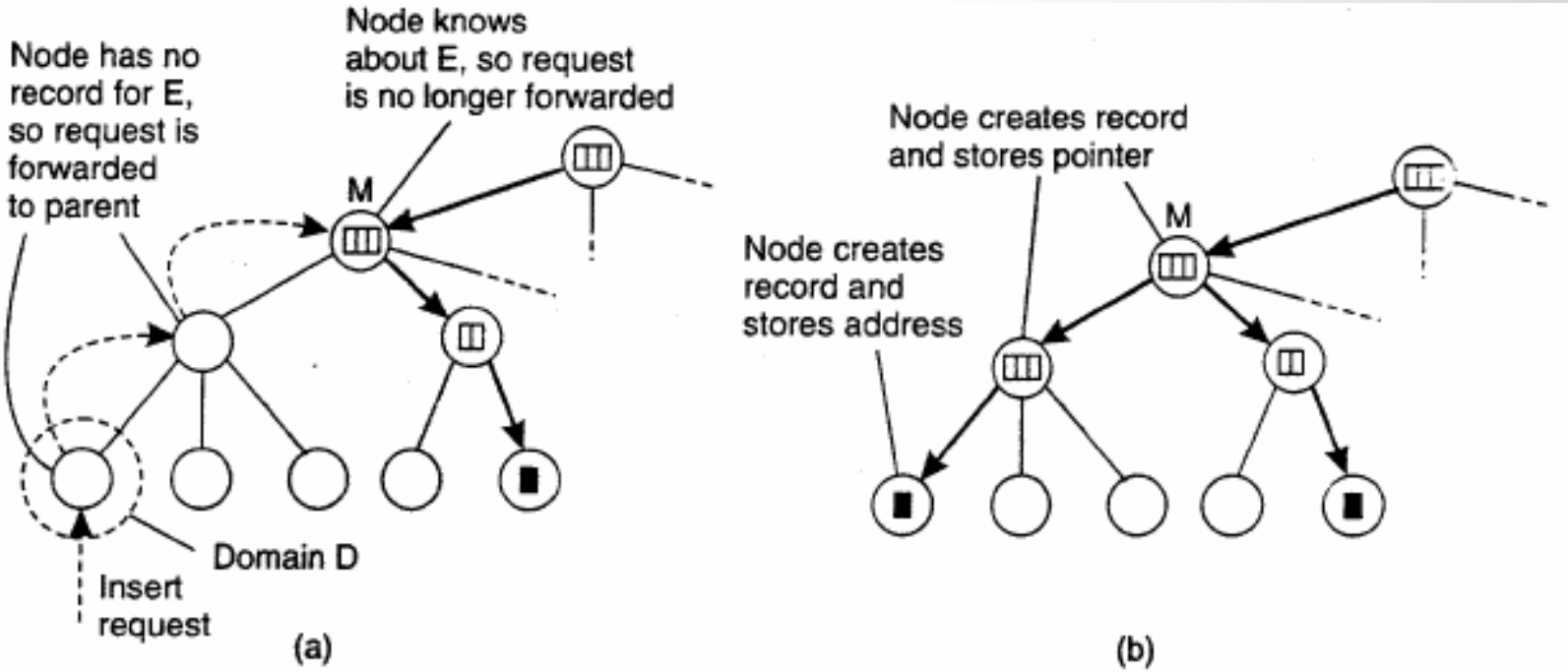
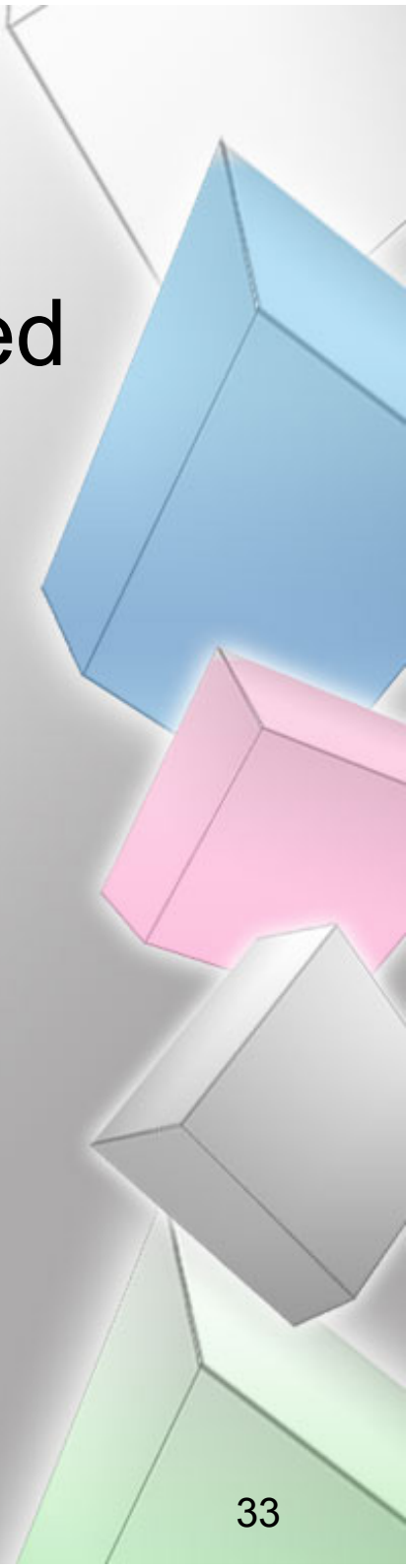


Figure 5-8. (a) An insert request is forwarded to the first node that knows about entity *E*. (b) A chain of forwarding pointers to the leaf node is created.

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 - for example, its address - so that a client can access it.
    - Alternatively, it can store the state of that entity, such as in the case of file systems 'in which a leaf node actually contains the complete file 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

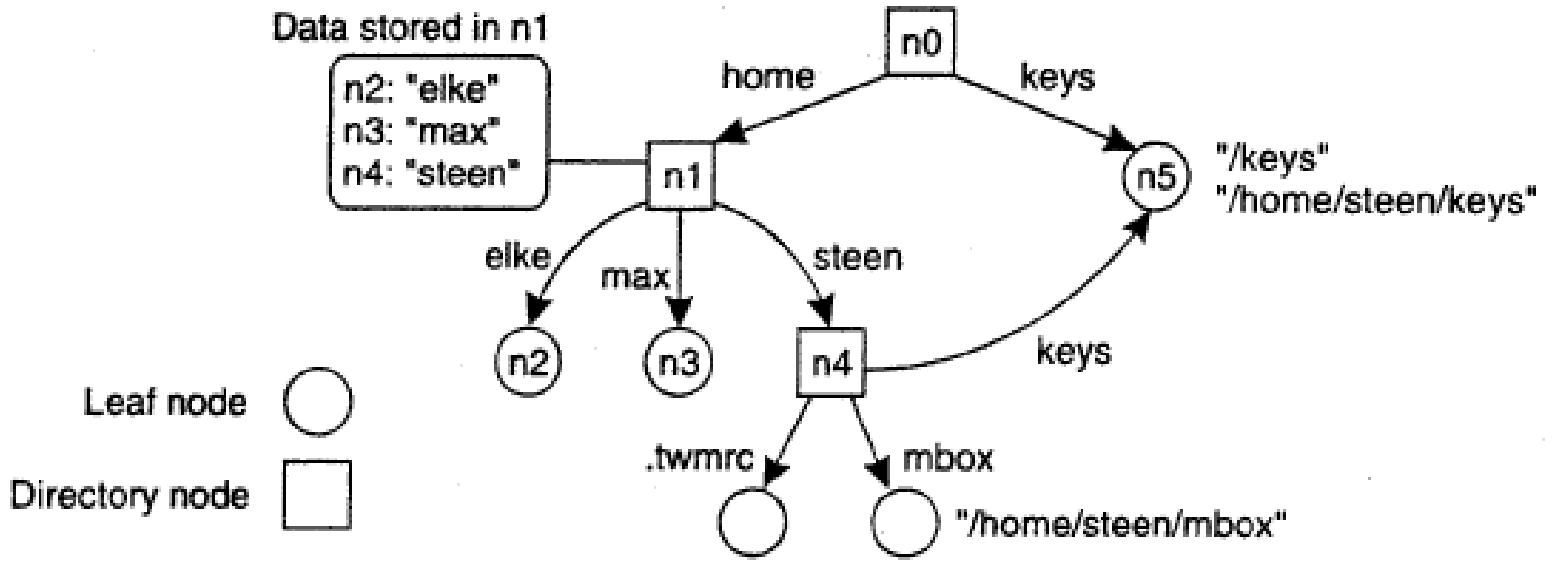


Figure 5-9. A general naming graph with a single root node.

- Each path in a naming graph can be referred to by the sequence of labels corresponding to the edges in that path, such as:

N: <label-I, label-2, ..., label-n>

where *N* refers to the first node in the path. Such a sequence is called a path name. If the first node in a path name is the root of the naming graph, it is called an absolute path name. Otherwise, it is called a relative path name.

Local and Global Names

- It is important to realize that names are always organized in a name space.
- As a consequence, a name is always defined relative only to a directory node. In this sense, the term "absolute name" is somewhat misleading.
- Likewise, the difference between global and local names can often be confusing. A global name is a name that denotes the same entity, no matter where that name is used in a system.
  - In other words, a global name is always interpreted with respect to the same directory node. In contrast, a local name is a name whose interpretation depends on where that name is being used. Put differently, a local name is essentially a relative name whose directory in which it is contained is (implicitly) known.

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.

UNIX file system implementation

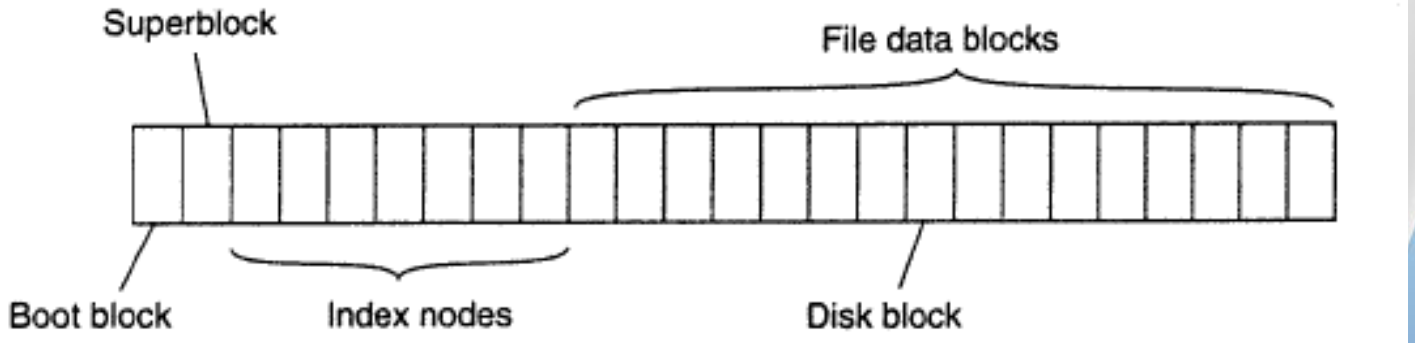


Figure 5·10. The general organization of the UNIX file system implementation on a logical disk of contiguous disk blocks.

- The superblock contains information on the entire file system. such as its size, which blocks on disk are not yet allocated, which inodes are not yet used, and so on.
- Inodes are referred to by an index number. Each inode contains information on where the data of its associated file can be found on disk. In addition, an inode contains information on its owner, time of creation and last modification, protection and others.

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)
  - 0031204447784: dial a phone number
  - 130.37.24.8: route to the VU's Web server

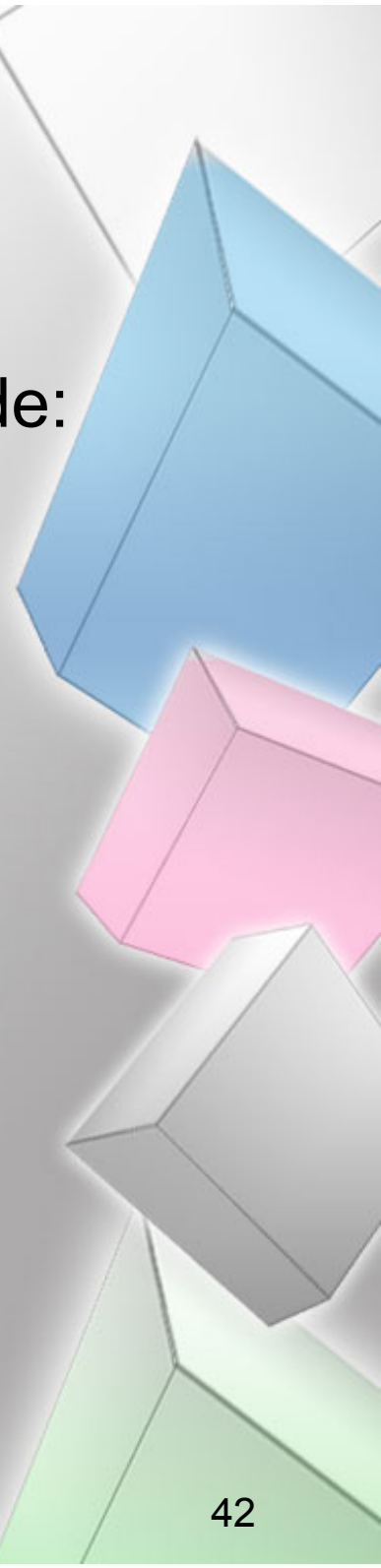
Linking

- Strongly related to name resolution is the use of aliases.
  - An alias is another name for the same entity.
- An environment variable is an example of an alias. In terms of naming graphs, there are basically two different ways to implement an alias:
- The first approach is to simply allow multiple absolute paths names to refer to the same node in a naming graph.
- This approach is illustrated in Slide 36, in which node n_5 can be referred to by two different path names.
- In UNIX terminology, both path names */keys* and */homelsteen/keys* are called **hard links** to node n_5 .

Name linking

Soft link

- Allow a node O to contain a name of another node:
  - First resolve O's name (leading to O)
  - Read the content of O, yielding name
  - Name resolution continues with name



Name Linking

- The second approach is to represent an entity by a leaf node, say *N,* but instead of storing the address or state of that entity, the node stores an absolute path name.
- When first resolving an absolute path name that leads to *N,* name resolution will return the path name stored in *N,* at which point it can continue with resolving that new path name. This principle corresponds to the use of **symbolic links** in UNIX file systems.

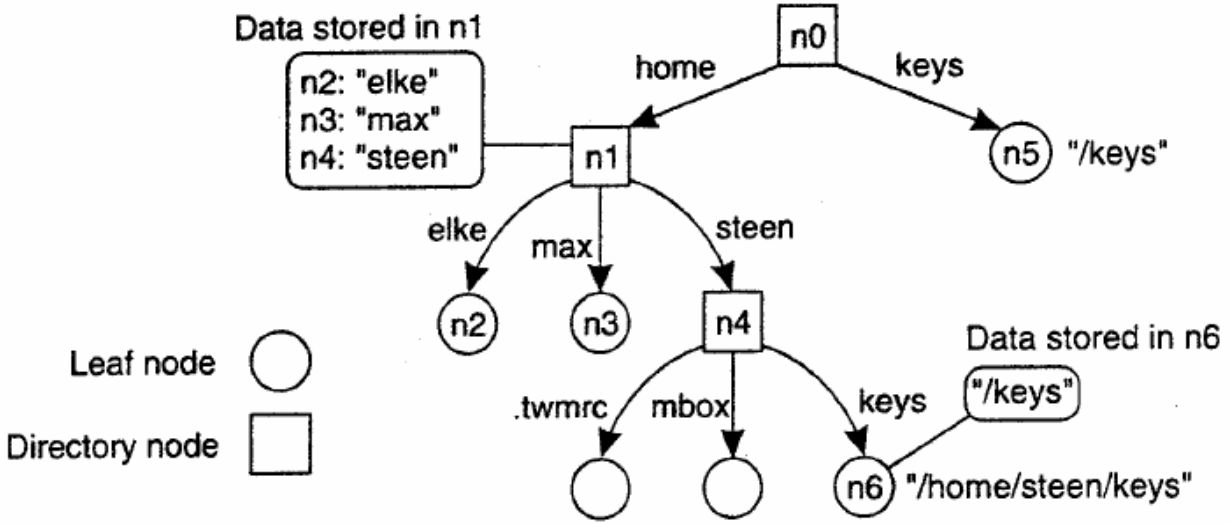


Figure 5-11. The concept of a symbolic link explained in a naming graph.

In this example, the path name */home/steen/keys,* which refers to a node containing the absolute path name */keys,* is a symbolic link to node *n*5 .

Mount point

- Name resolution as described so far takes place completely within a single name space.
- However, name resolution can also be used to merge different name spaces in a transparent way.
- Mounted file system.
- In terms of our naming model, a mounted file system corresponds to letting a directory node store the identifier of a directory node from a *different* name space, which we refer to as a foreign name space.
- The directory node storing the node identifier is called a mount point.
- Accordingly, the directory node in the foreign name space is called a mounting point.
- Normally, the mounting point is the root of a name space.
- During name resolution, the mounting point is looked up and resolution proceeds by accessing its directory table.

Mounting foreign name spaces

Information required to mount a foreign name space in a distributed system

- The name of an access protocol.
- The name of the server.
- The name of the mounting point in the foreign name space.

Remote mounting

- A user with a laptop computer wants to access files that are stored on a remote file server.
- The client machine and the file server are both configured with Sun's Network File System (NFS).
- NFS is a distributed file system that comes with a protocol that describes precisely how a client can access a file stored on a (remote) NFS file server.
- In particular, to allow NFS to work across the Internet, a client can specify exactly which file it wants to access by means of an NFS URL, for example, *nfs:l/flits.cs. vu.nl//homelsteen.*
- This URL names a file (which happens to be a directory) called */home/steen* on an NFS file server*flits.cs. vu.nl,* which can be accessed by a client by means of the NFS protocol

Mounting remote name spaces

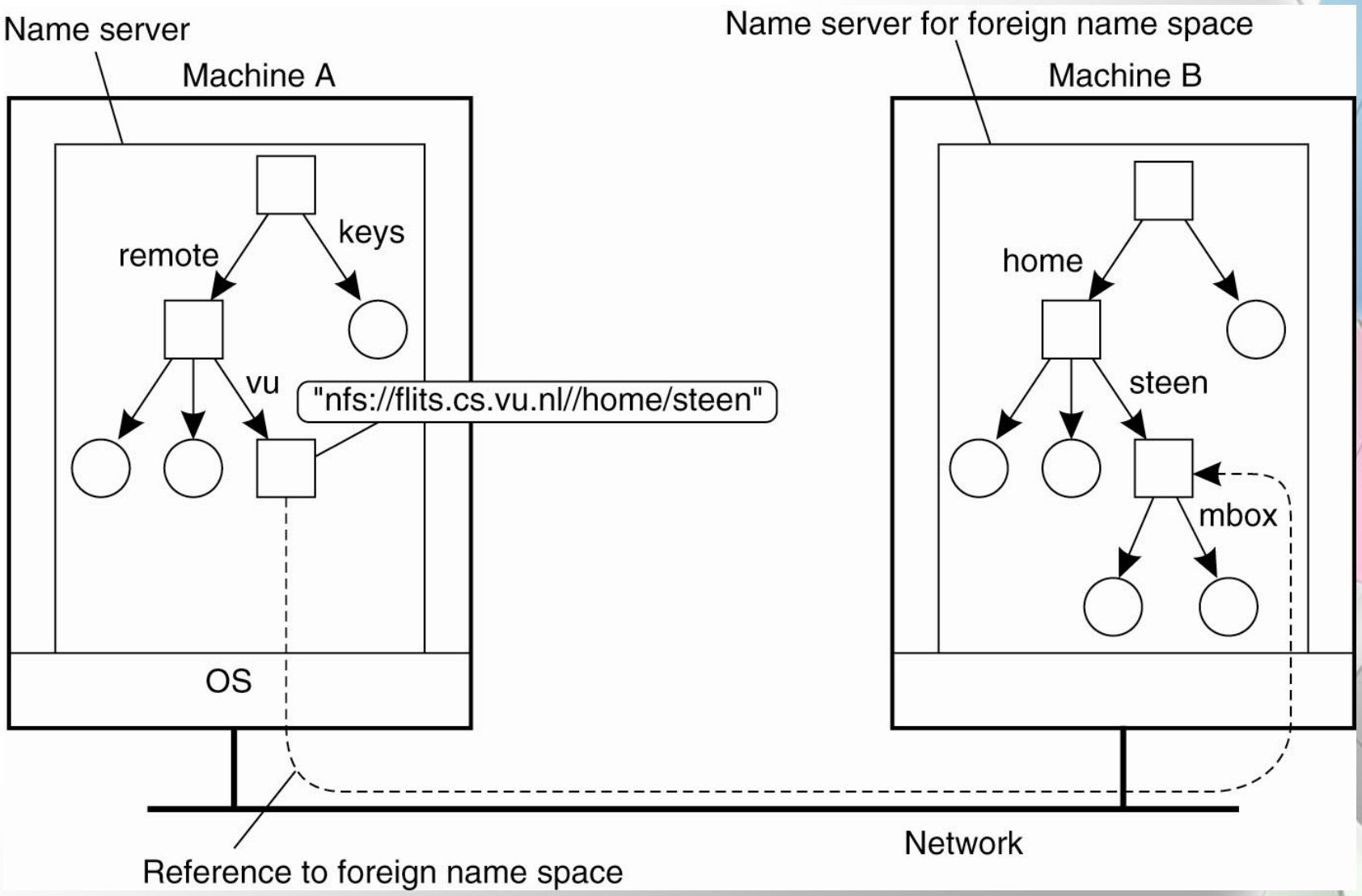


Figure 5-12. Mounting remote name spaces through a specific access protocol.

Name-space implementation

- A name space forms the heart of a naming service, that is, a service that allows users and processes to add, remove, and look up names.
  - A naming service is implemented by name servers.
- If a distributed system is restricted to a local area network, it is often feasible to implement a naming service by means of only a single name server.
- However, in large-scale distributed systems with many entities, possibly spread across a large geographical area, it is necessary to distribute the implementation of a name space over multiple name servers.

Name Space Distribution

- 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

Global Layer

- The global layer is formed by highest-level nodes, that is, the root node and other directory nodes logically close to the root, namely its children.

- Nodes in the global layer are often characterized by their stability, in the sense that directory tables are rarely changed.

- Such nodes may represent organizations or groups of organizations, for which names are stored in the name space.

Administrational layer

- The administrational layer is formed by directory nodes that together are managed within a single organization.
- A characteristic feature of the directory nodes in the administrational layer is that they represent groups of entities that belong to the same organization or administrational unit.
  - For example, there may be a directory node for each' department in an organization, or a directory node from which all hosts can be found.
- Another directory node may be used as the starting point for naming all users, and so forth. The nodes in the administrational layer are relatively stable, although changes generally occur more frequently than to nodes in the global layer.

Managerial Layer

- Finally, the managerial layer consists of nodes that may typically change regularly.
- For example, nodes representing hosts in the local network belong to this layer.
- For the same reason, the layer includes nodes representing shared files such as those for libraries or binaries.
- Another important class of nodes includes those that represent user-defined directories and files. In contrast to the global and administrational layer, the nodes in the managerial layer are maintained not only by system administrators, but also by individual end users of a distributed system.

Name-space implementation

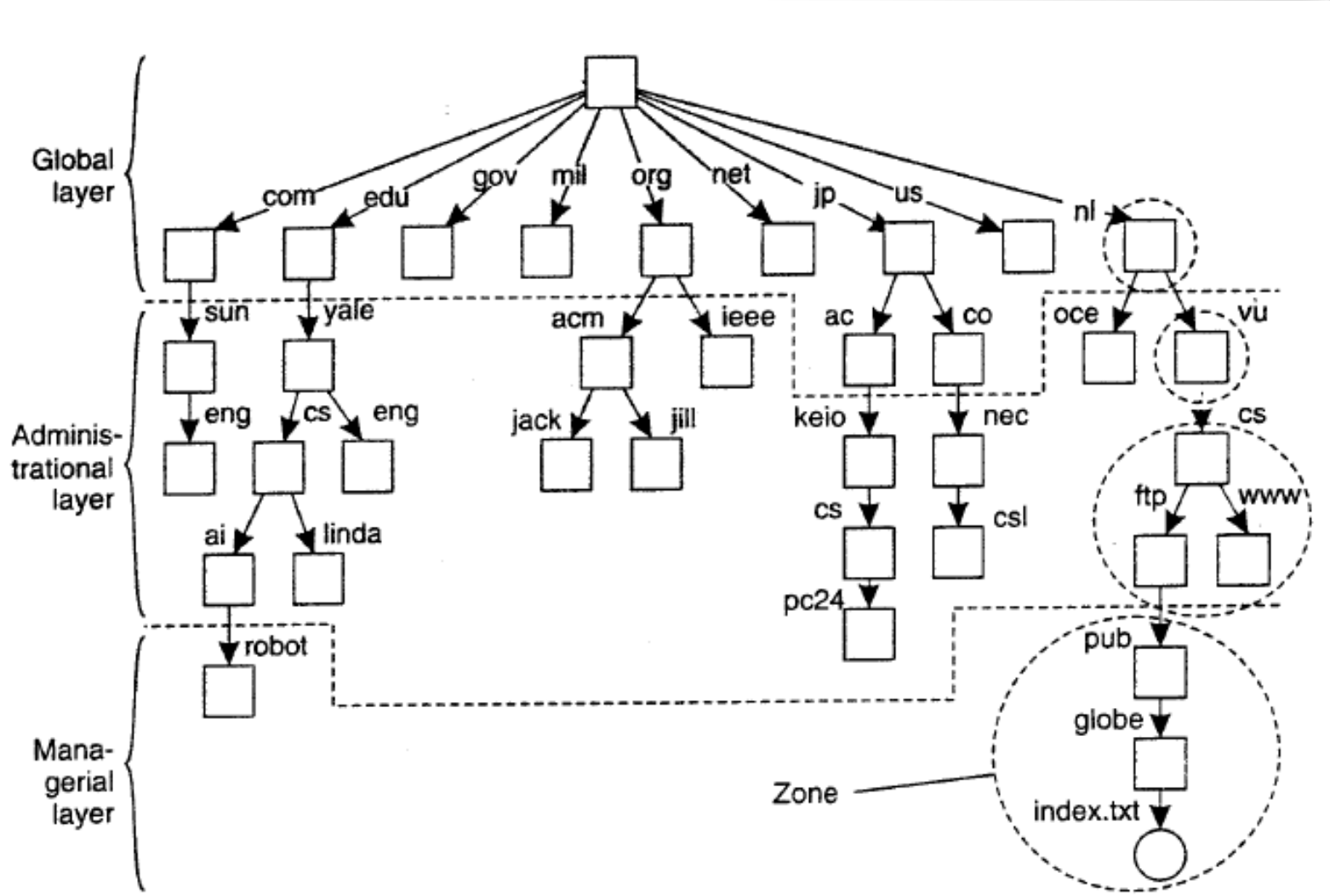


Figure 5-13. An example partitioning of the DNS name space, including Internet-accessible files, into three layers.

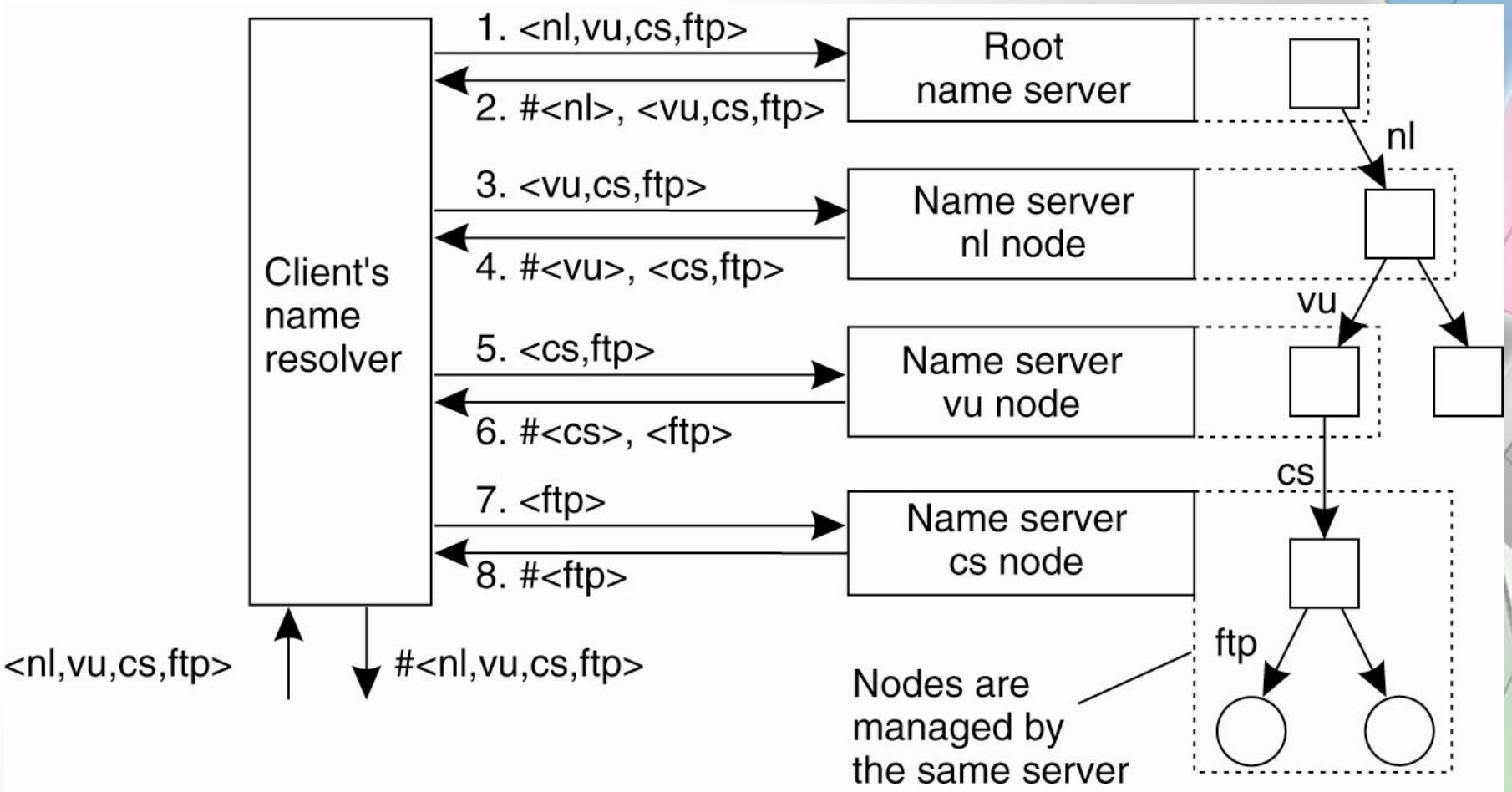
Name-space implementation

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

1. resolve(dir,[name1,...,nameK]) sent to Server0 responsible for dir
2. Server0 resolves resolve(dir,name1) ! dir1, returning the identification (address) of Server1, which stores dir1.
3. Client sends resolve(dir1,[name2,...,nameK]) to Server1, etc.

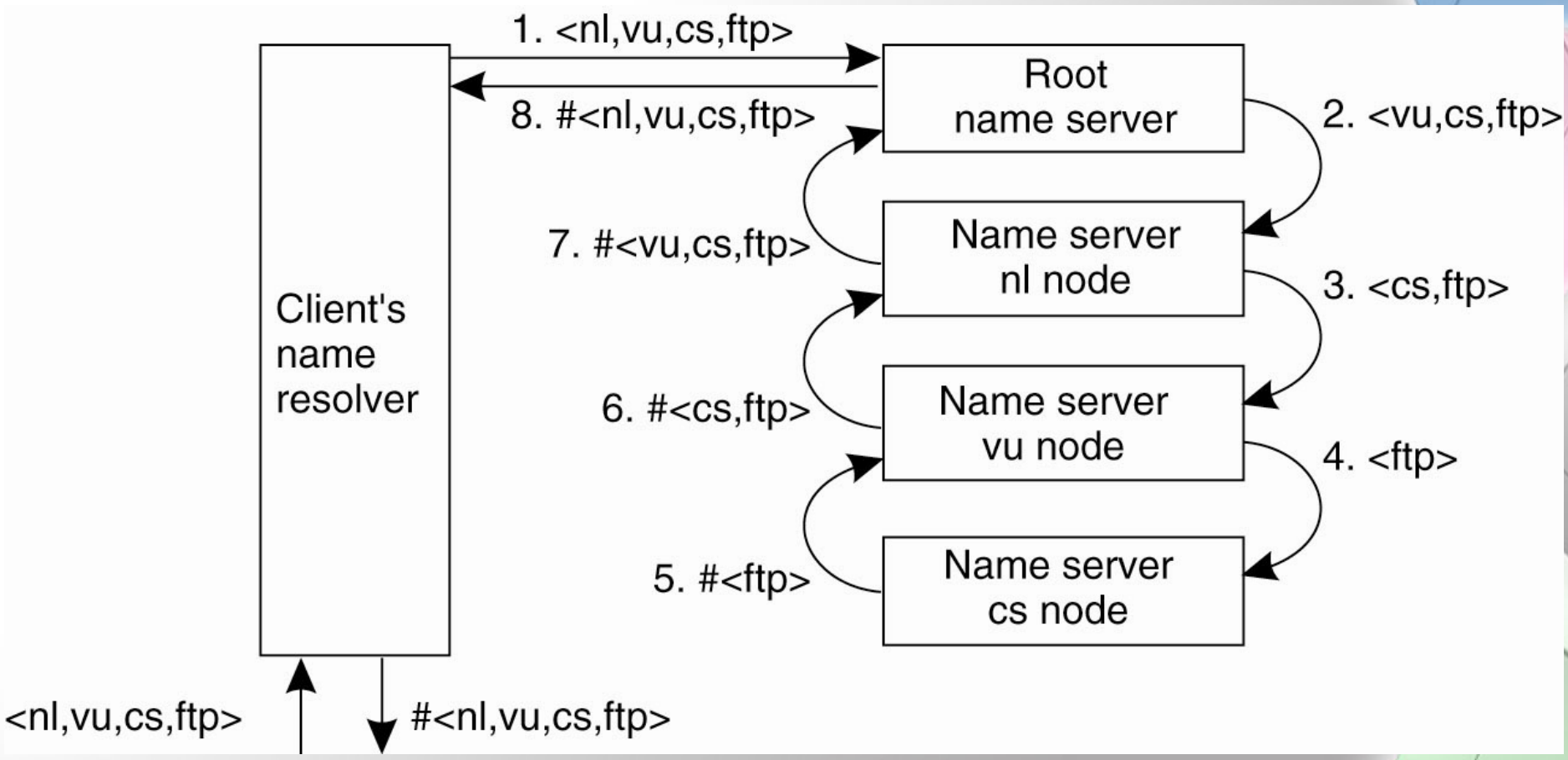


Recursive name resolution

1 resolve(dir,[name1,...,nameK]) sent to Server0 responsible for dir

2 Server0 resolves resolve(dir,name1) ! dir1, and sends resolve(dir1,[name2,...,nameK]) to Server1, which stores dir1.

3 Server0 waits for result from Server1, and returns it to client.

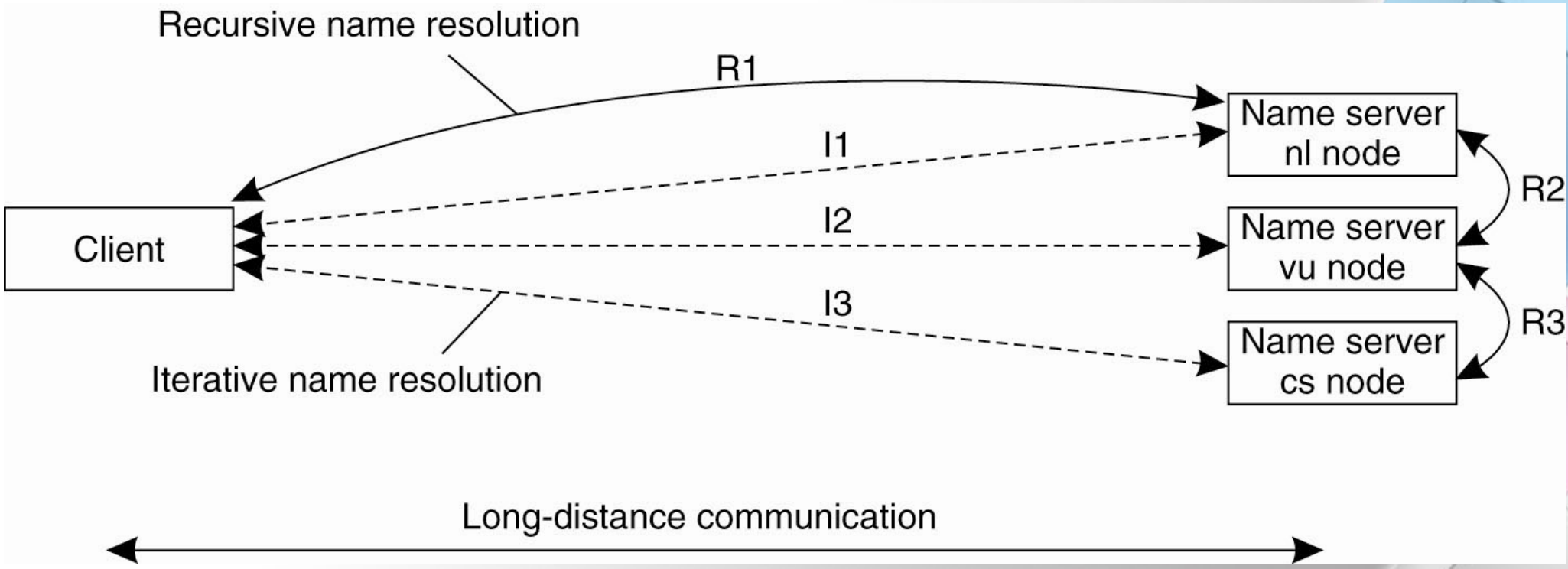


Implementation of Name Resolution (3)

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.

Example: The Domain Name System



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

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.
- Because each node in the DNS name space has exactly one incoming edge (with the exception of the root node, which has no incoming edges), the label attached to a node's incoming edge is also used as the name for that node.
- A subtree is called a domain; a path name to its root node is called a domain name.
- Note that, just like a path name, a domain name can be either absolute or relative.
- 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

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.
- The LDAP directory service has been derived from OS1's X.500 directory service.
- As with many OSI services, the quality of their associated implementations hindered widespread use, and simplifications were needed to make it useful.

LDAP

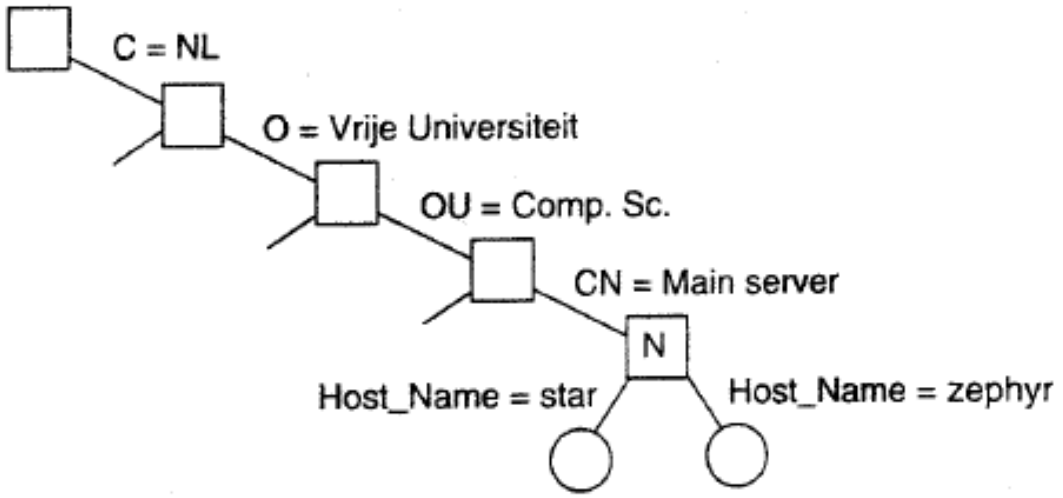
- Conceptually, an LDAP directory service consists of a number of records, usually referred to as directory entries.
- A directory entry is comparable to a resource record in DNS.
- Each record is made up of a collection of *(attribute. value)* pairs, where each attribute has an associated type.
- A distinction is made between single-valued attributes and multiple-valued attributes.
- The latter typically represent arrays and lists.

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

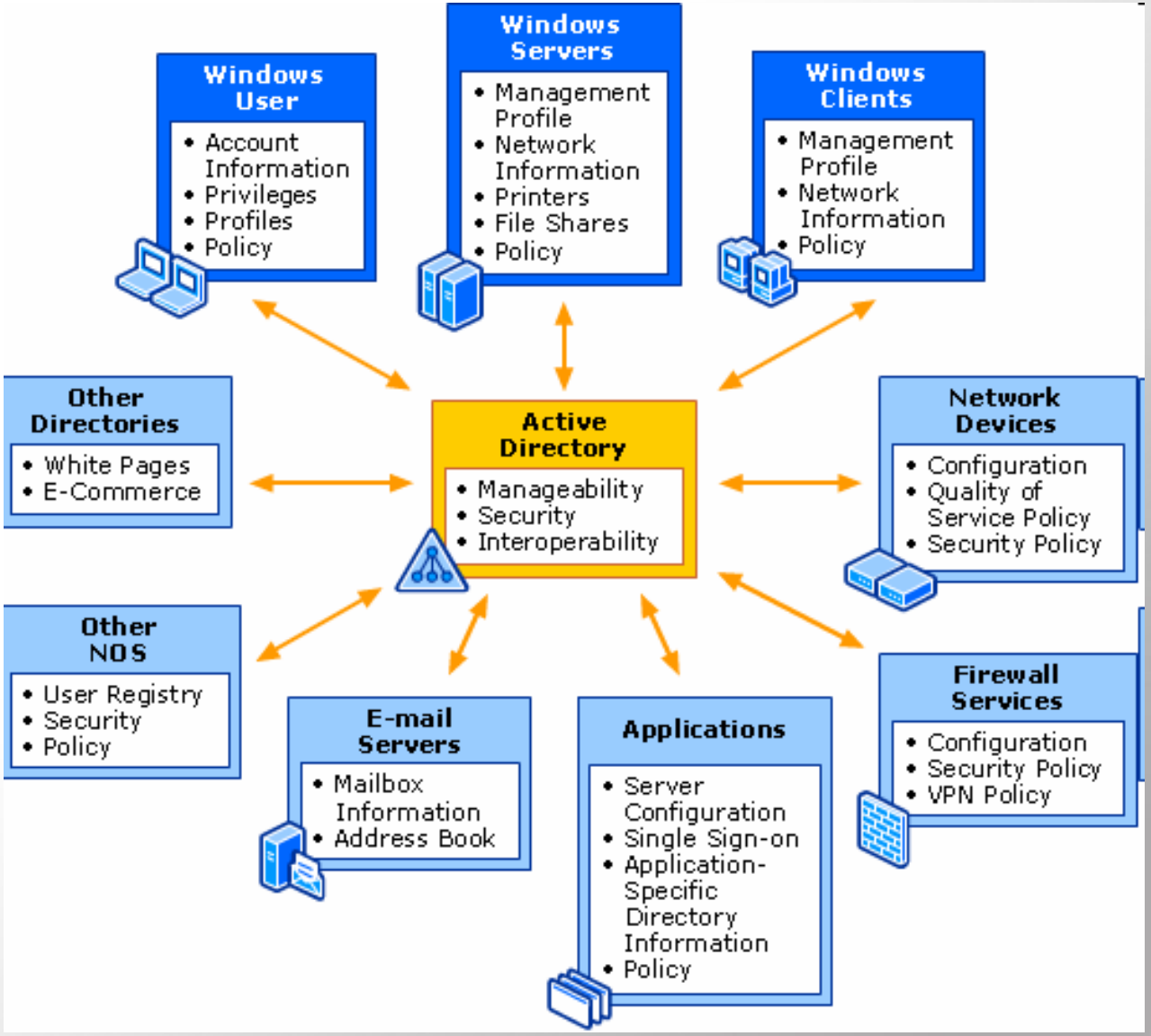
(b)

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

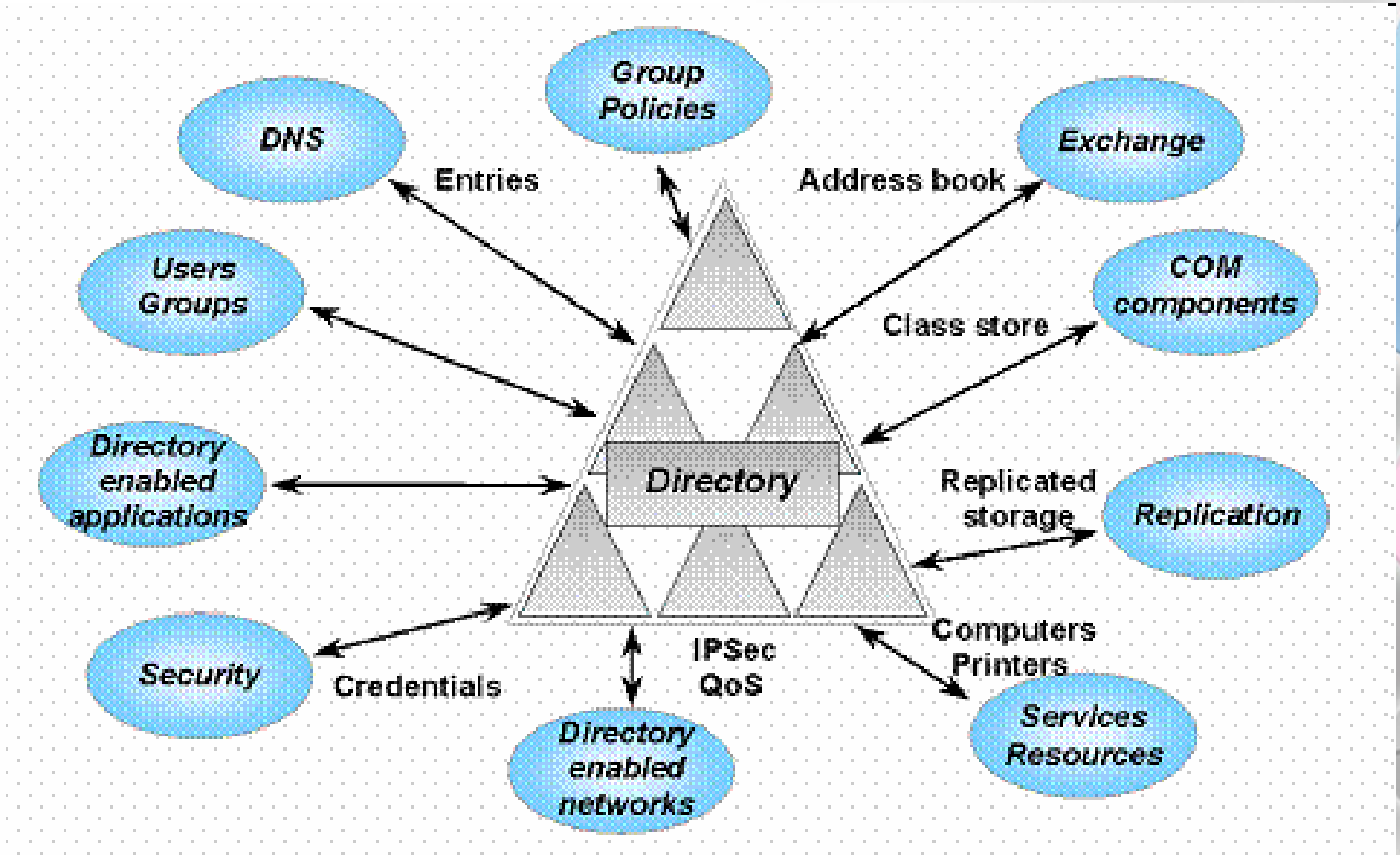
Active Directory

- 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

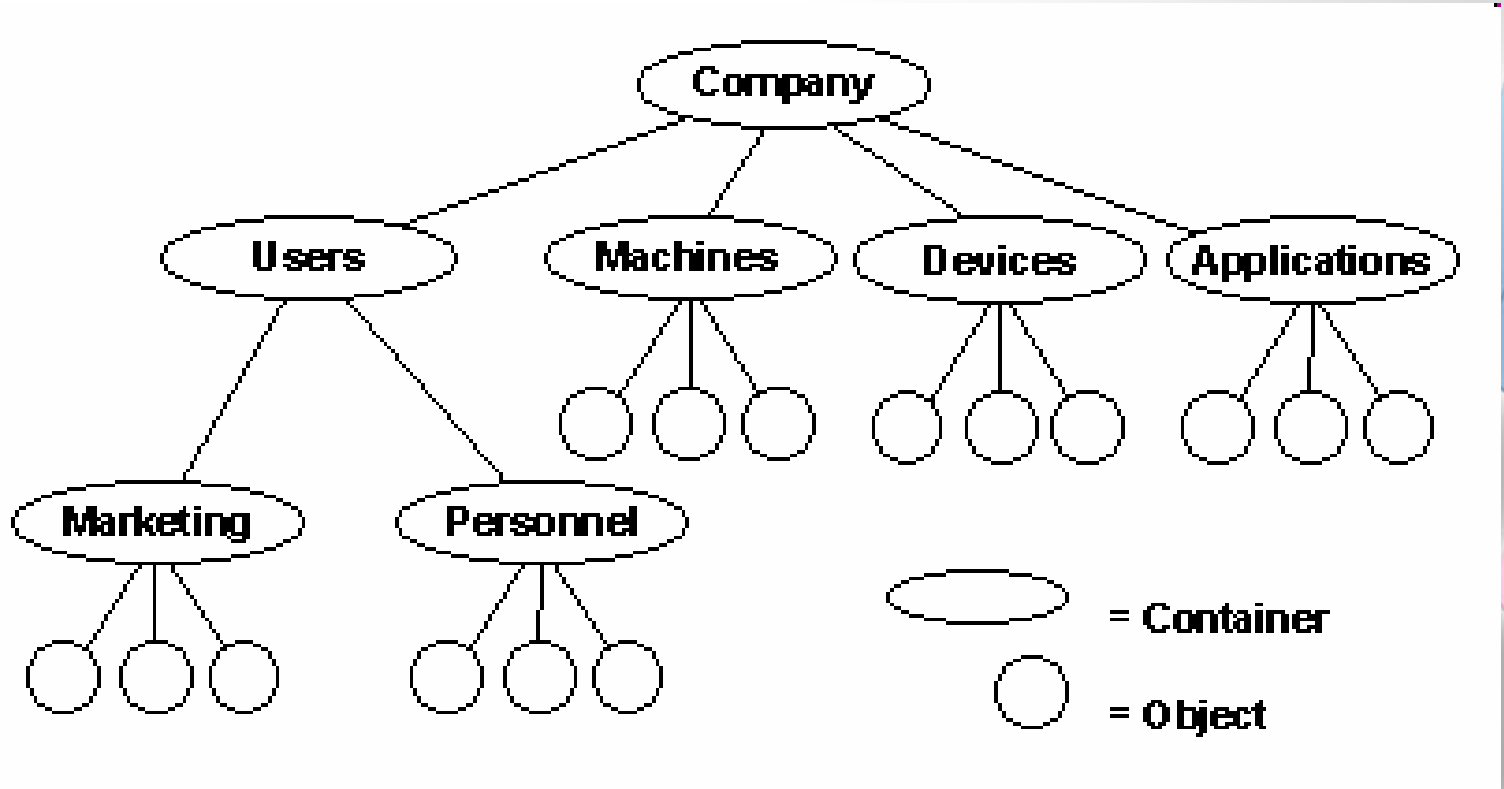
Active Directory



Organization of AD

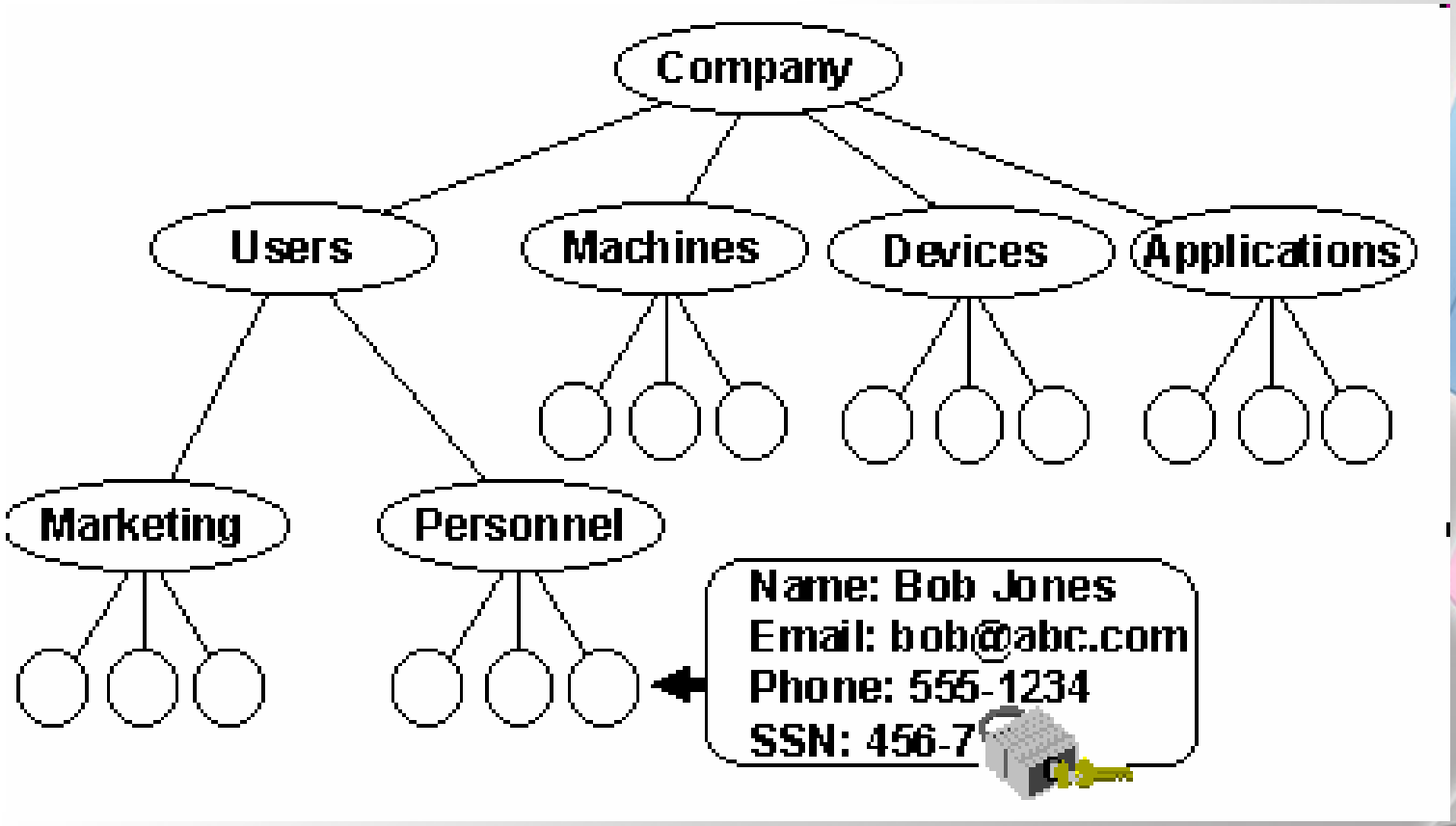


Hierarchical organization of AD



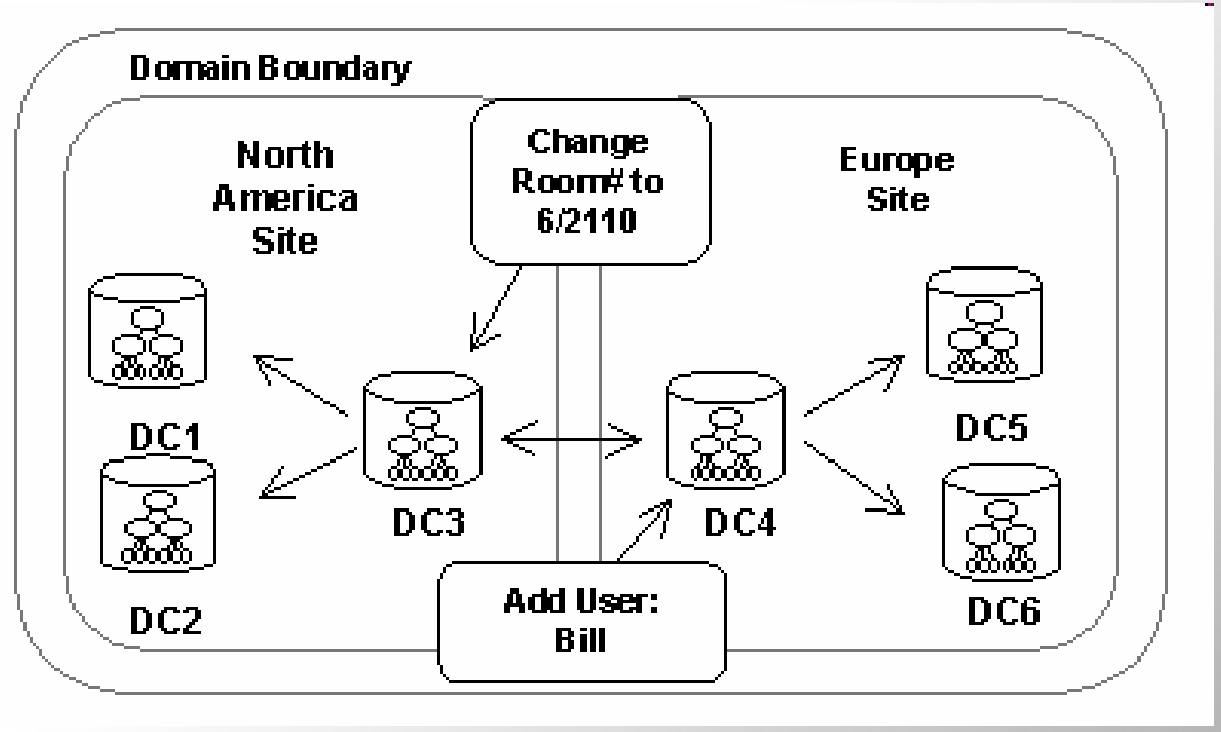
Active Directory organizes information hierarchically to ease network use and management

Object-oriented Storage in AD



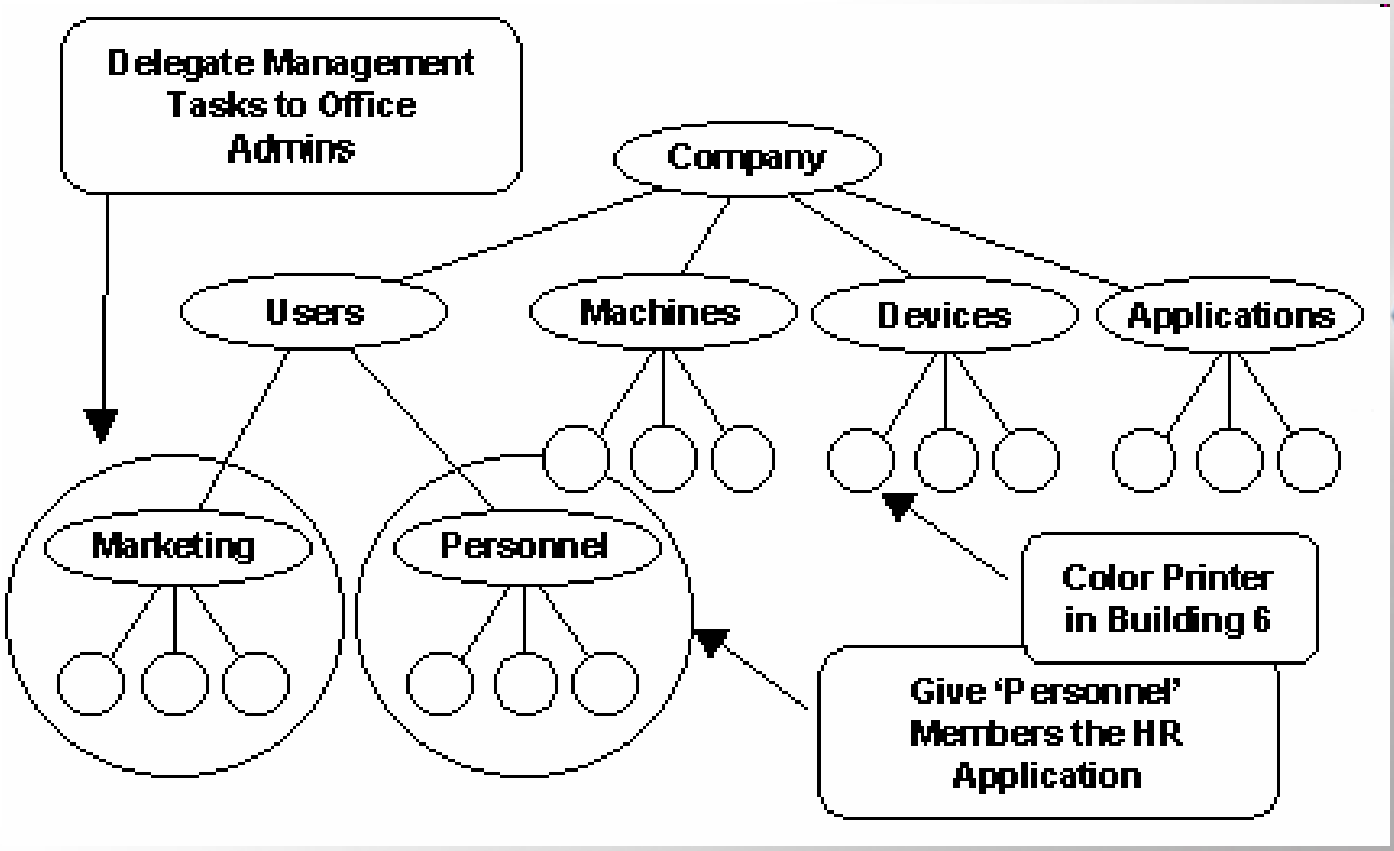
Active Directory objects and attributes are protected by access control lists.

Multi-Master Replication



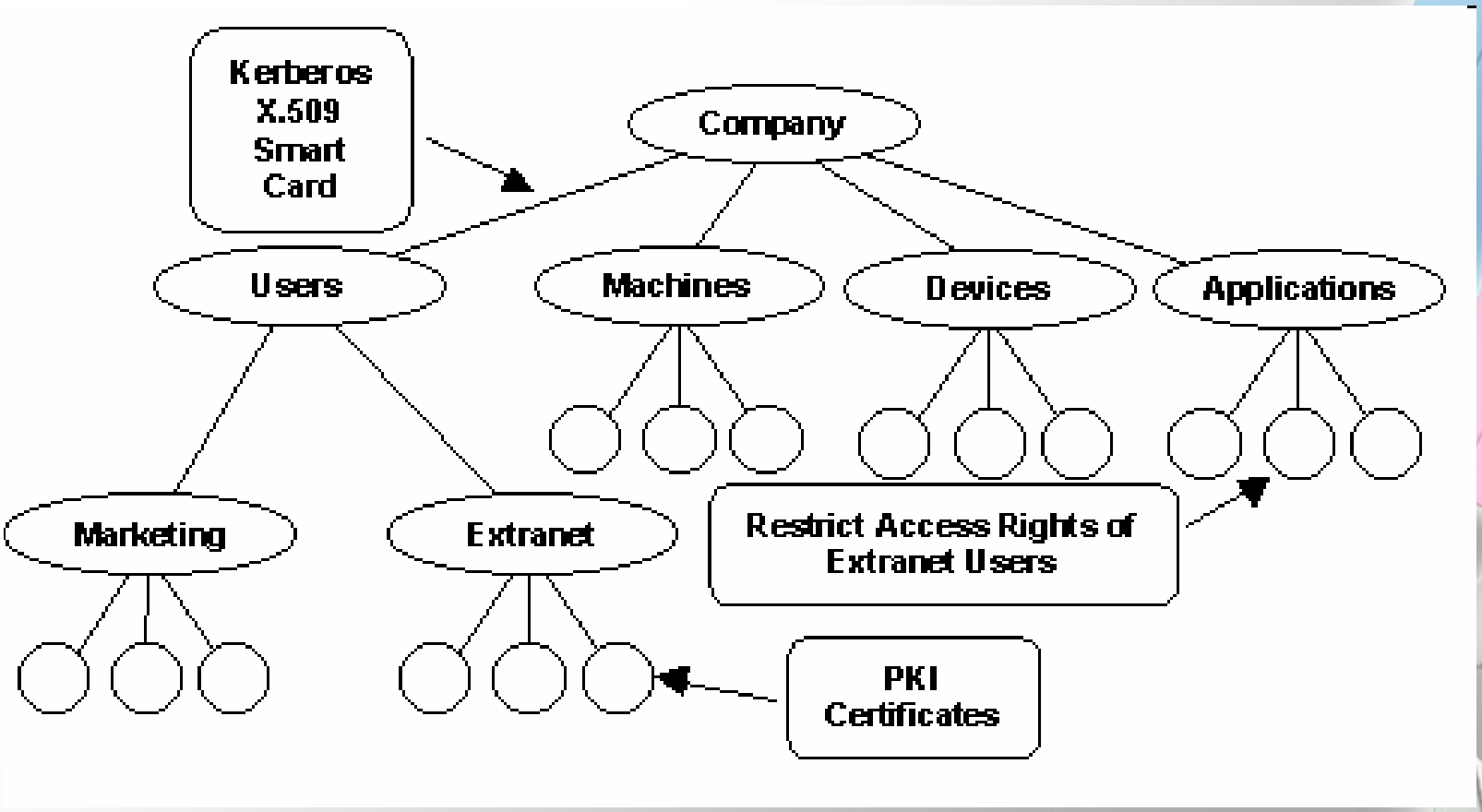
Active Directory supports multi-master replication for flexibility, high-availability, and performance.

Management of network resources

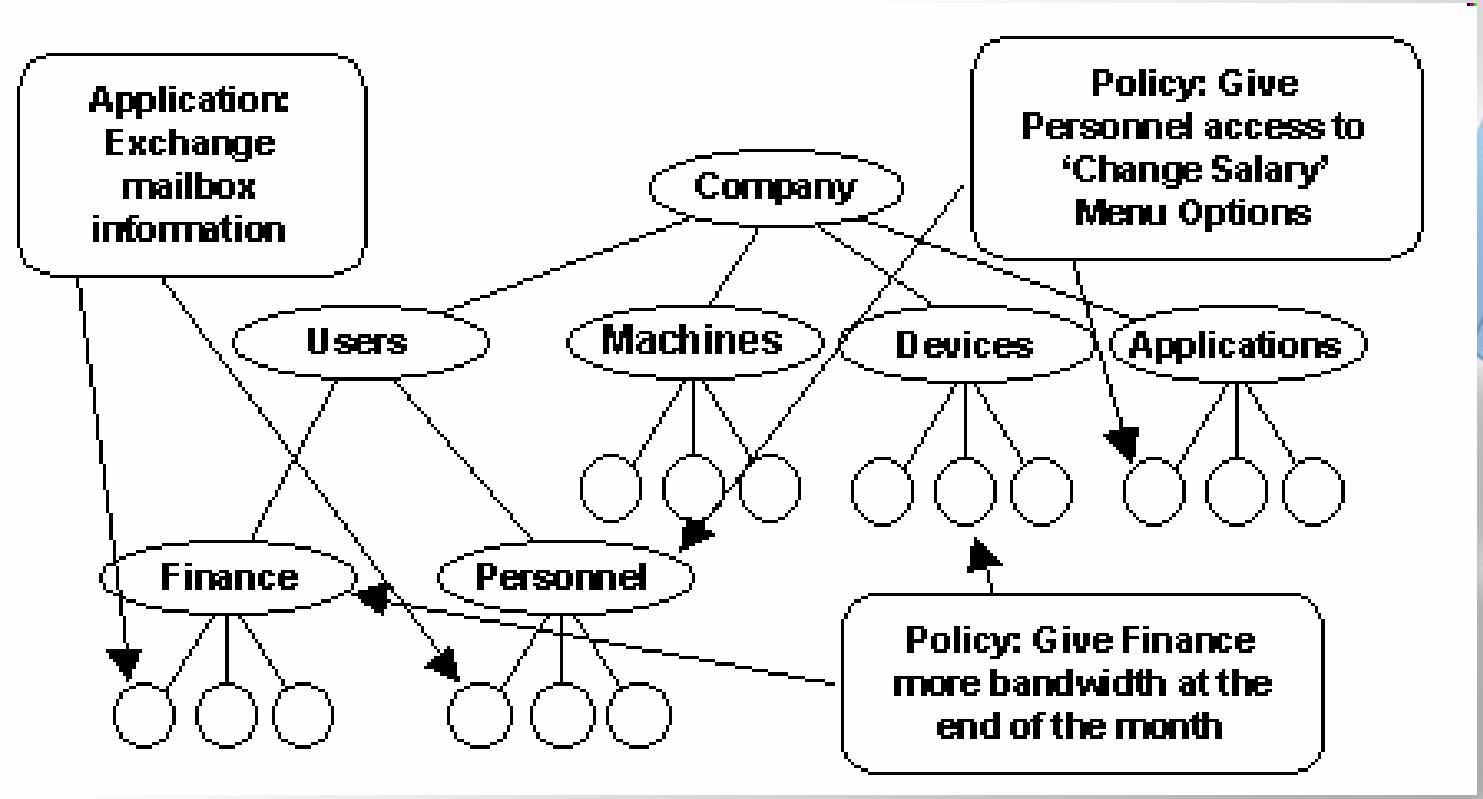


Active Directory simplifies management of network resources.

Active Directory Strengthens Security



Active Directory Extends Interoperability



End of Lesson 5

- Readings
  - Distributed Systems, Chapter 5.