DISTRIBUTED SYSTEMS
Principles and Paradigms
Second Edition
ANDREW S. TANENBAUM
MAARTEN VAN STEEN

# Chapter 5
# Naming

# The Naming of Cats

*T. S. Eliot*

The naming of Cats is a difficult matter,
It isn't just one of your holiday games;
You may think at first I'm mad as a hatter
When I tell you, a cat must have THREE DIFFERENT NAMES
. . .

"What's in a name? That which we call a rose
By any other name would ~~smell~~ serve as sweet a method

modified from *Romeo and Juliet (II, ii, 1-2)*

## Overview

Names are used to refer to entities, locations, resources and more. We need to resolve a name to the entity it refers to. The naming system may itself be implemented in a distributed fashion.

- How to organize and implement human friendly names. E.g. files systems, World Wide Web
- Locating entities in a way that is independent of current location from their names
- Resolving names by means of entity attributes

## Names, Identifiers and Addresses (1)

Properties of a true identifier:

- An 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

# Names, Identifiers and Addresses (2)

- A *name* in a distributed system is a string of bits or characters used to refer to an *entity*. Entity is something that is operated on using some *access point*. The name of the access point is called an *address*.
- *Entities*: hosts, printers, disks, files, processes, users, mailboxes, web pages, graphical windows, messages, network connections, etc.
- An entity may change its access point over course of time. Thus the address cannot be treated as the name of the entity. Moreover an entity may have more than one access point. *Location independent name* is separate from the address of the access point.
- An *identifier* is an unambiguous reference to an entity.
- We need a universally unique identifier. How to generate a *Universally Unique Identifier*?

## Names, Identifiers and Addresses (3)

- Check out java.util.UUID for a class that generates a Universally Unique Identifier
- The RFC 4122 (*A Universally Unique IDentifier (UUID) URN Namespace)* proposes several ways of creating UUIDs.

## Types of Naming Systems

- Flat naming: The identifier is simply a random bit string. It does not contain any information whatsoever on how to locate an access point of its associated entity. Good for machines.
- Structured naming: Composed of simple human-readable names. Examples are file system naming and host naming on the Internet.
- Attribute-based naming: Allows an entity to be described by (*attribute*, *value*) pairs. This allows a user to search more effectively by constraining some of the attributes.

## Flat Naming Systems

- Broadcasting and multicasting
- Forwarding pointers
- Home-based approaches
- Distributed hash tables
- Hierarchical approaches

# Broadcasting/Multicasting

- LANs offer an efficient broadcasting facility, We can use that for the mapping.
  - A message containing the identity of the entity is broadcast to each machine
  - Only the machines that can offer access to that entity send a reply containing the address of the access point.
- Address Resolution Protocol (ARP). Used on a LAN to find the data-link (MAC) address of a machine given only the IP address.
- Check `man arp` on Linux or run the `arp` command in the lab!
- A more efficient approach for larger networks is *multicasting*, by which only a restricted group of machines receive the request.

# Forwarding Pointers (1)

- 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 using a traditional naming service, a client can look up the current address by following the chain of forwarding pointers.
- It is important to keep the forwarding chains relatively short and to ensure that the forwarding pointers are robust.
- *Example*: Remote objects that can move from host to host.
  - 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 A to B, it leaves behind a client stub in its place on A and installs a server stub that refers to it in B. This makes the migration completely transparent to a client.

# Forwarding Pointers (2)

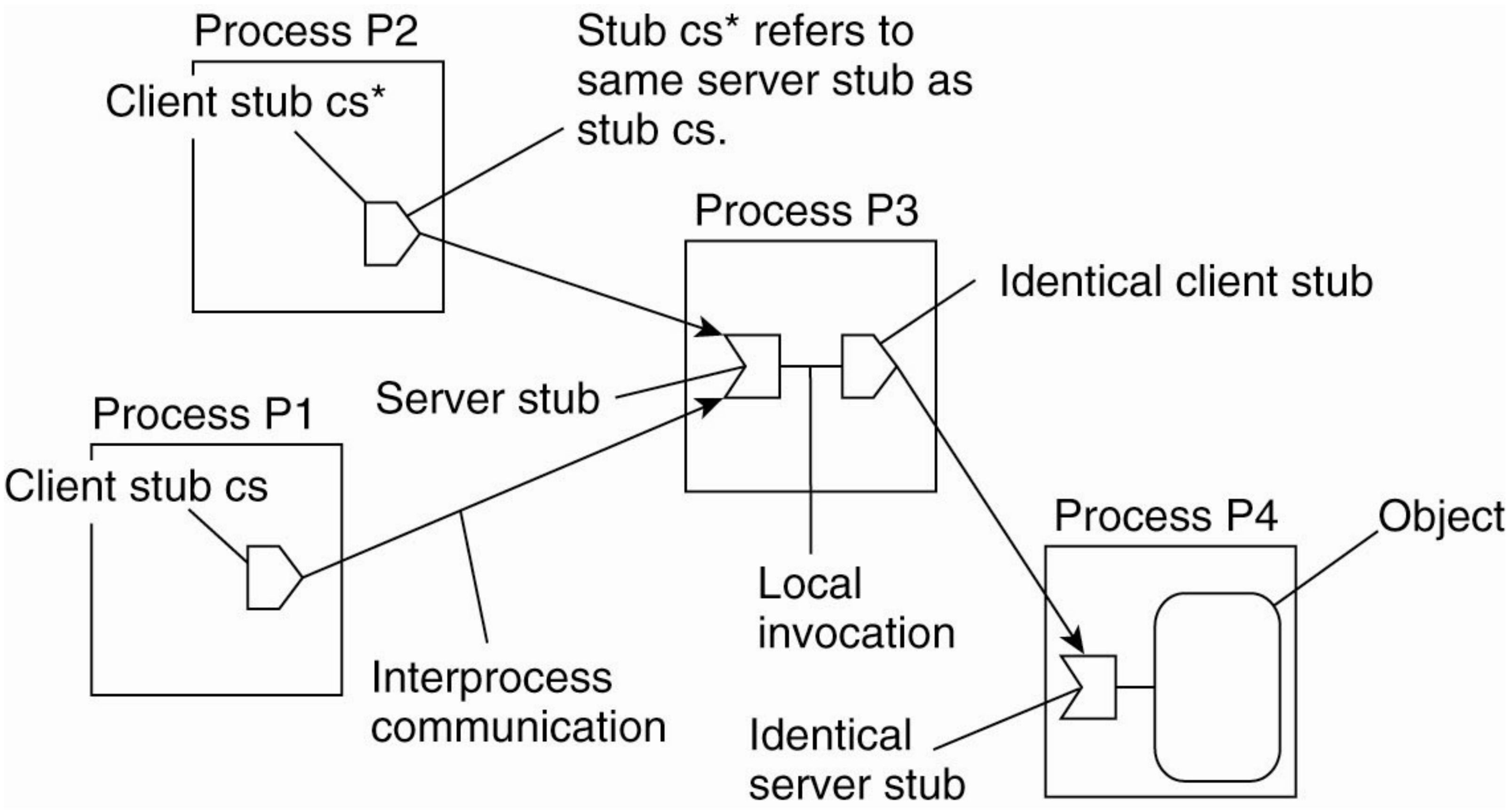


Figure 5-1. The principle of forwarding pointers using (client stub, server stub) pairs.

# Forwarding Pointers (3)

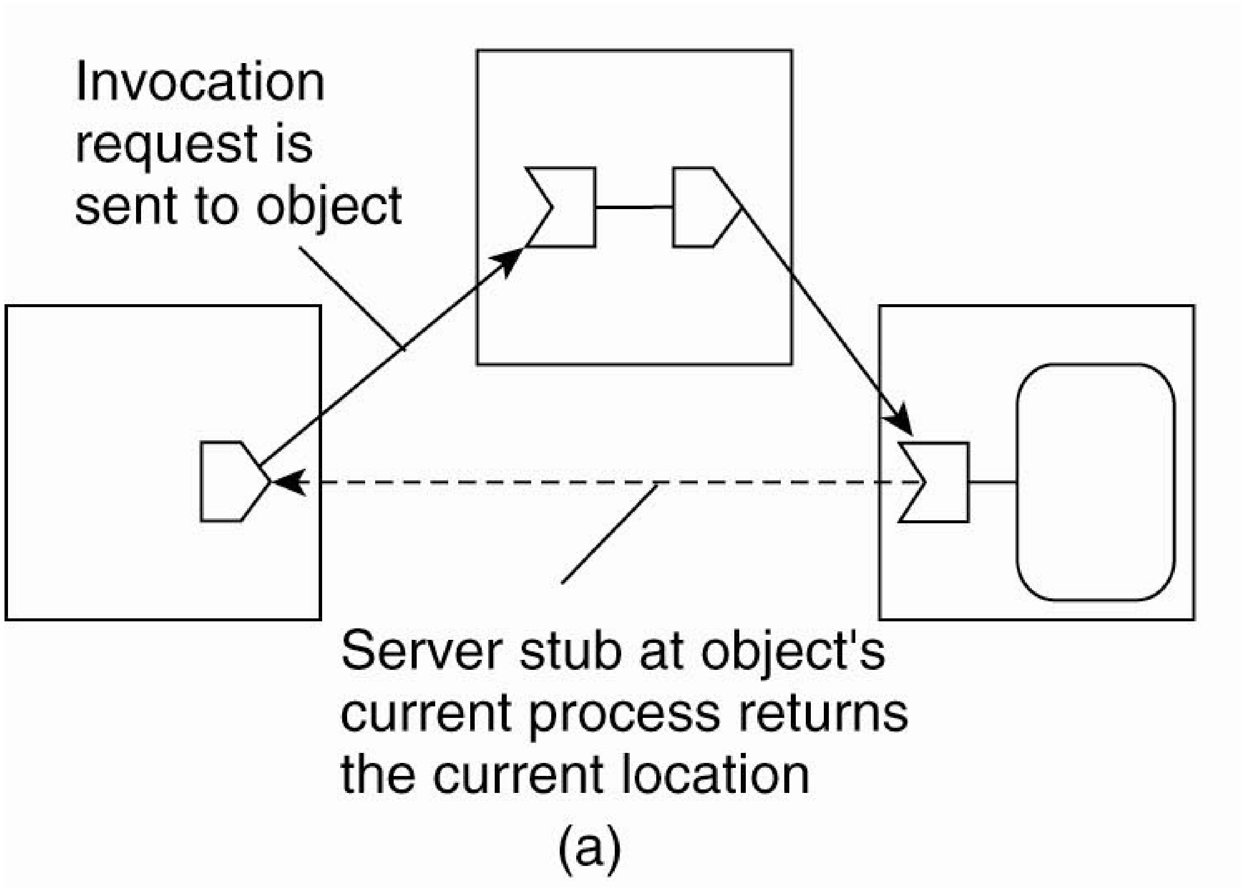

Invocation request is sent to object
Server stub at object's current process returns the current location
(a)

Figure 5-2. Redirecting a forwarding pointer by storing a shortcut in a client stub.

# Forwarding Pointers (4)

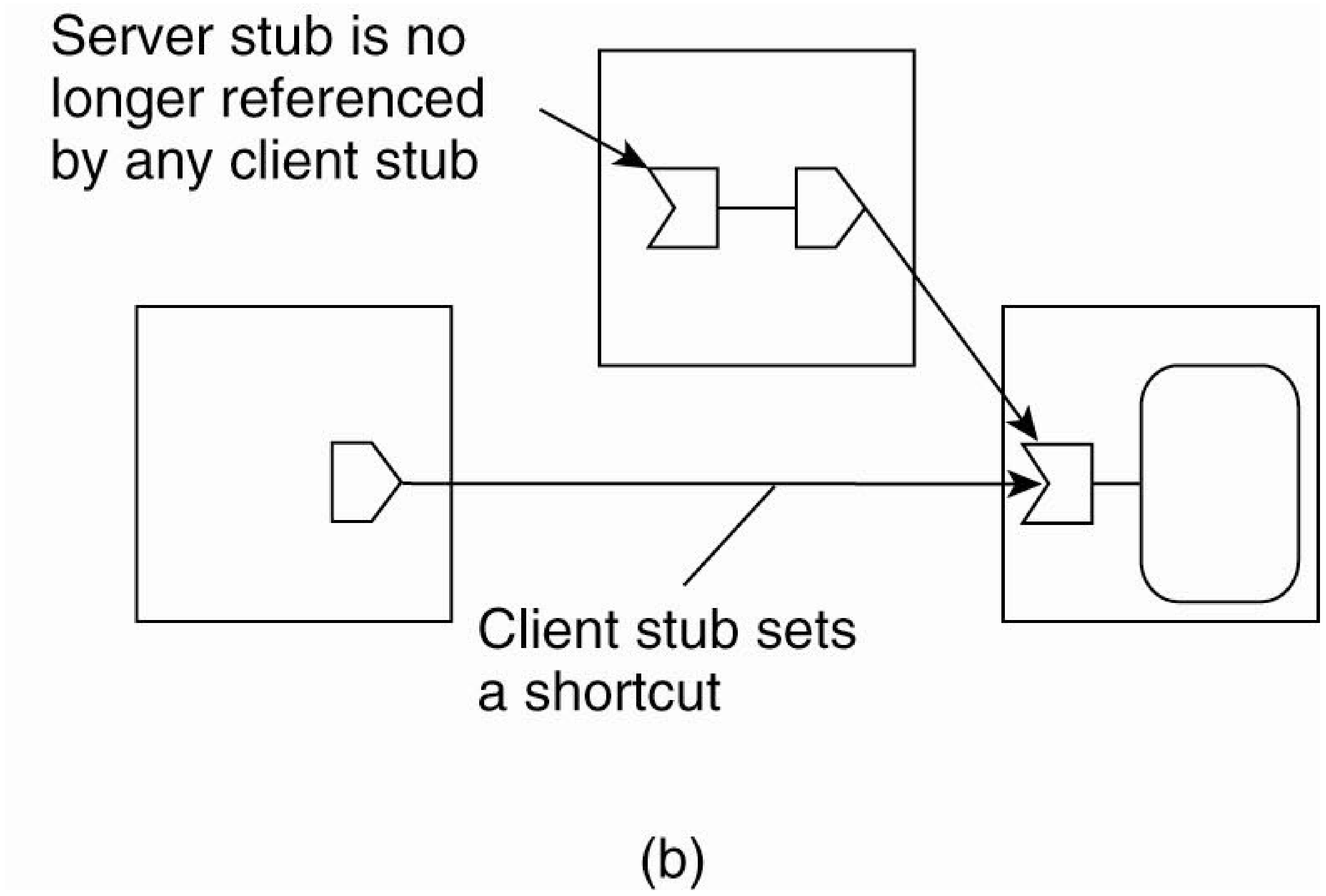

Server stub is no longer referenced by any client stub
Client stub sets a shortcut
(b)

Figure 5-2. Redirecting a forwarding pointer by storing a shortcut in a client stub.

# Forwarding Pointers (5)

- Chains can be short-cut by sending a response directly to the initiating client stub or along the reverse path of forwarding pointers. What are the pros and cons of the two approaches?
- Makes it easy to pass remote object references among processes. Why?
- To overcome crash of a process in the chain, we always keep a fault-tolerant reference to the current location of an object on the machine it was created. Then we need a way for the home location to change. This is usually done via traditional naming services.

# Locating Mobile Entities

Traditional naming systems are primarily used for naming entities that have a fixed location. They are not well-suited for supporting name-to-address mappings that change regularly. Suppose an entity moves to a new address. How do we handle this?

- Record the new address in the original DNS database.
  Problem: If entity moves again, the update becomes a remote operation, possibly taking long time to complete

- Record the new name, creating a symbolic link.
  Problem: Each lookup now takes two operations. Also each move of the entity adds another level of lookup.
  *Solution*: Separate naming from locating entities by introducing identifiers. An entity has a unique identifier that never changes.

# Home-Based Approach

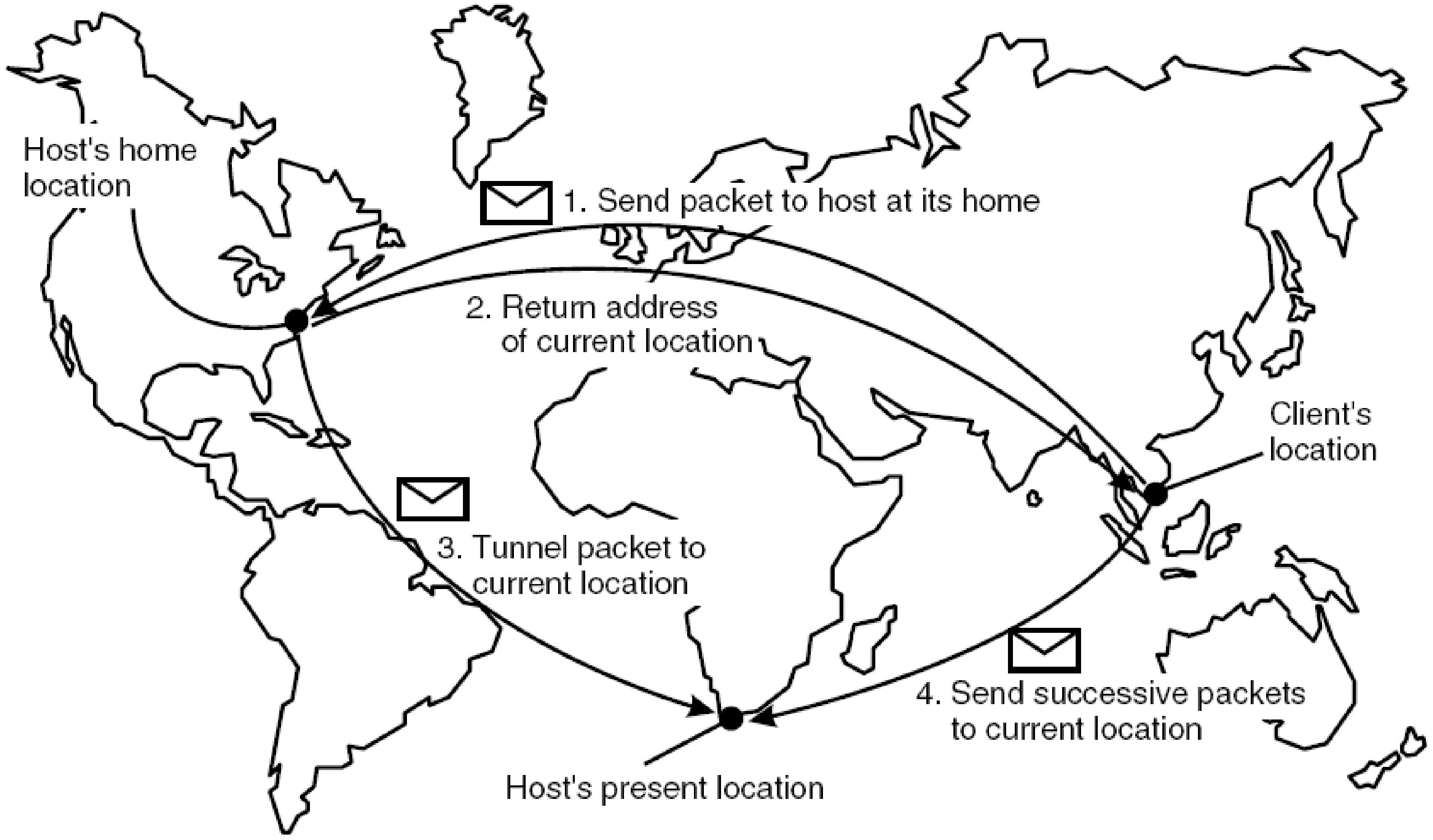

Host's home location
1. Send packet to host at its home
2. Return address of current location
Client's location
3. Tunnel packet to current location
4. Send successive packets to current location
Host's present location

Figure 5-3. The principle of Mobile IP.

## Distributed Hash Tables (1)

- A distributed technique on resolving an identifier to the address of an associated entity.
- Selected applications
  - BTDigg: BitTorrent DHT search engine
  - Oracle Coherence: An in-memory data grid built on a Java DHT implementation
  - WebSphere eXtreme Scale: proprietary DHT implementation by IBM, used for object caching
  - Coral Content Distribution Network
  - YaCy: Java-based distributed search engine

# Distributed Hash Tables (1)

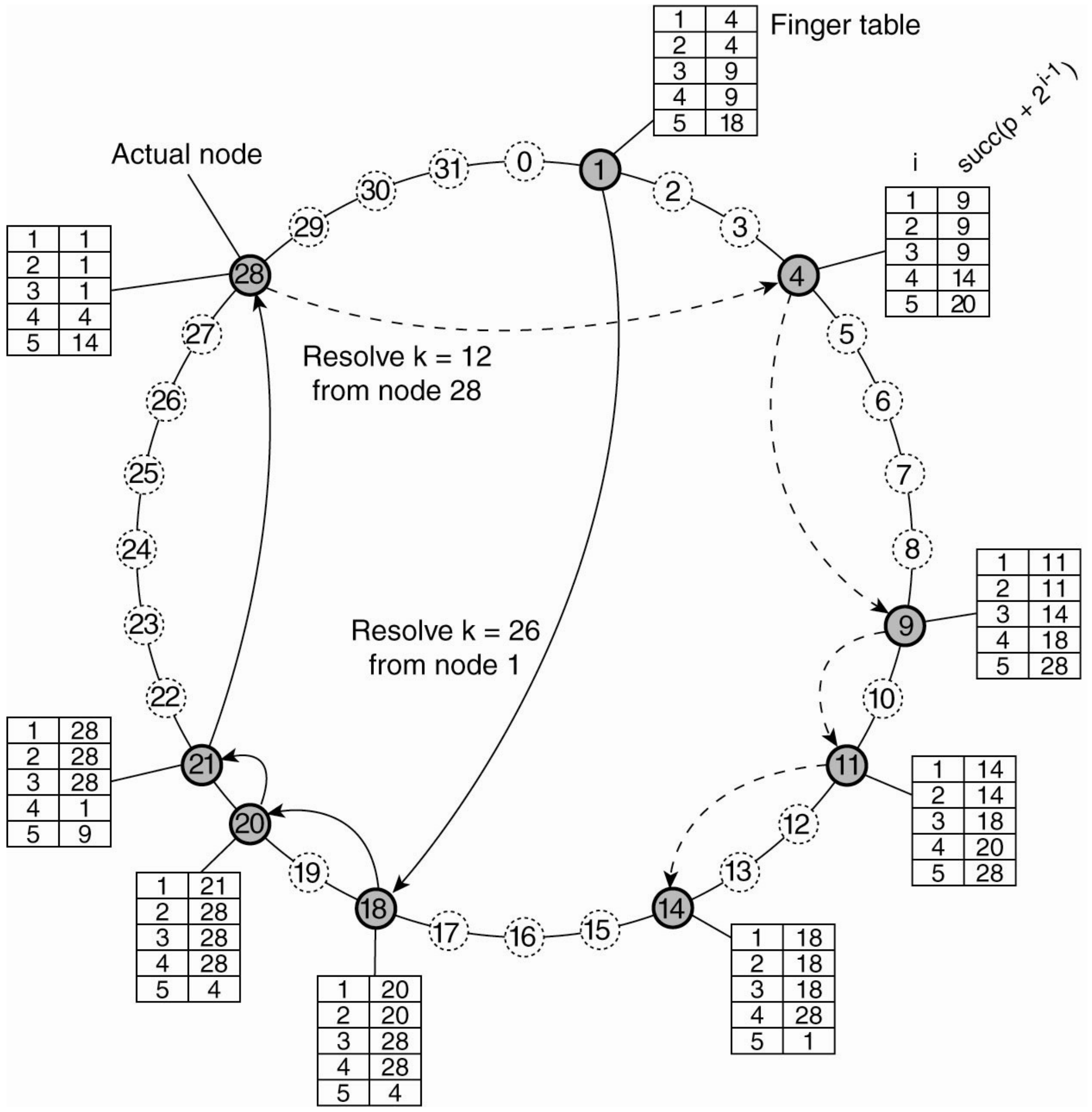
- Chord system: An *m-bit* (usually 128 or 160 bits) identifier space to assign randomly chosen identifiers to nodes as well as specific entities.
- An entity with key *k* falls under the jurisdiction of the node with the smallest identifier $id \geq k$. This node is referred to as the *successor* of *k* and denoted as *succ(k)*.
- How to resolve a key *k* to the address of *succ(k)*?
  - Linear search: Each node keeps track the successor *succ(p+1)* and predecessor *pred(p)*. A node can then pass along a request to resolve to one of its two neighbors unless *pred)p*) $< k \leq p$ in which case it returns its own address. This would be slow....

# Distributed Hash Tables (2)

- How to efficiently resolve a key *k* to the address of *succ*(*k*)?
  - *Unbounded binary search*: Each node maintains a *finger table* of at most *m* entries
    $$FT_p[i] = succ(p + 2^{i-1})$$
  - The *i*th entry points to the first node succeeding p by at least $2^{i-1}$. These are exponentially increasing short-cuts in the identifier space
  - To look up a key *k*, a node *p* will then forward the request to node *q* with index *j* in *p*'s finger table where:
    $$q = FT_p[j] \leq k < FT_p[j+1]$$
  - Uses modulo arithmetic

# Distributed Hash Tables (3)

## General Mechanism



Node 1:

| i | succ |
|---|---|
| 1 | 4 |
| 2 | 4 |
| 3 | 9 |
| 4 | 9 |
| 5 | 18 |

Node 4:

| i | succ |
|---|---|
| 1 | 9 |
| 2 | 9 |
| 3 | 9 |
| 4 | 14 |
| 5 | 20 |

Node 9:

| i | succ |
|---|---|
| 1 | 11 |
| 2 | 11 |
| 3 | 14 |
| 4 | 18 |
| 5 | 28 |

Node 11:

| i | succ |
|---|---|
| 1 | 14 |
| 2 | 14 |
| 3 | 18 |
| 4 | 20 |
| 5 | 28 |

Node 14:

| i | succ |
|---|---|
| 1 | 18 |
| 2 | 18 |
| 3 | 18 |
| 4 | 28 |
| 5 | 1 |

Node 18:

| i | succ |
|---|---|
| 1 | 20 |
| 2 | 20 |
| 3 | 28 |
| 4 | 28 |
| 5 | 4 |

Node 20:

| i | succ |
|---|---|
| 1 | 21 |
| 2 | 28 |
| 3 | 28 |
| 4 | 28 |
| 5 | 4 |

Node 21:

| i | succ |
|---|---|
| 1 | 28 |
| 2 | 28 |
| 3 | 28 |
| 4 | 1 |
| 5 | 9 |

Node 28:

| i | succ |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 1 |
| 4 | 4 |
| 5 | 14 |

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

# Distributed Hash Tables (4)

- *How to add/remove nodes?* Suppose node *p* wants to join. Contact an arbitrary node and request a lookup for *succ*(*p*+1). Removal is also easy if each node also keeps track of their predecessor.
- Keeping finger tables up-to-date is more complex. Each node *q* regularly runs a simple procedure that contacts *succ*(*q*+1) and requests it to return *pred*(*succ*(*q*+1))
- Have a background thread/process check each entry in the table.
- Have each node periodically check whether its predecessor is alive. If there is no response, it marks it's predecessor to be "unknown." This gets set when a node is updating its link to the next known node and finds that node's predecessor to be unknown.

# Distributed Hash Tables (5)

- Exploiting network proximity:
  - *Topology-based assignment of node identifiers.*
  - *Proximity routing*: Nodes maintain a list of alternatives to forward a request to.
  - *Proximity neighbor selection*: Optimize routing tables such that the nearest node is selected as neighbor. This implies that there are more nodes to choose from.
  - *Iterative lookup*: The node returns the network address of the next node found to the requesting process, which then iterates the lookup with the next node,
  - *Recursive lookup*: Let a node forward the lookup request to the next node.

# Hierarchical Approaches (1)

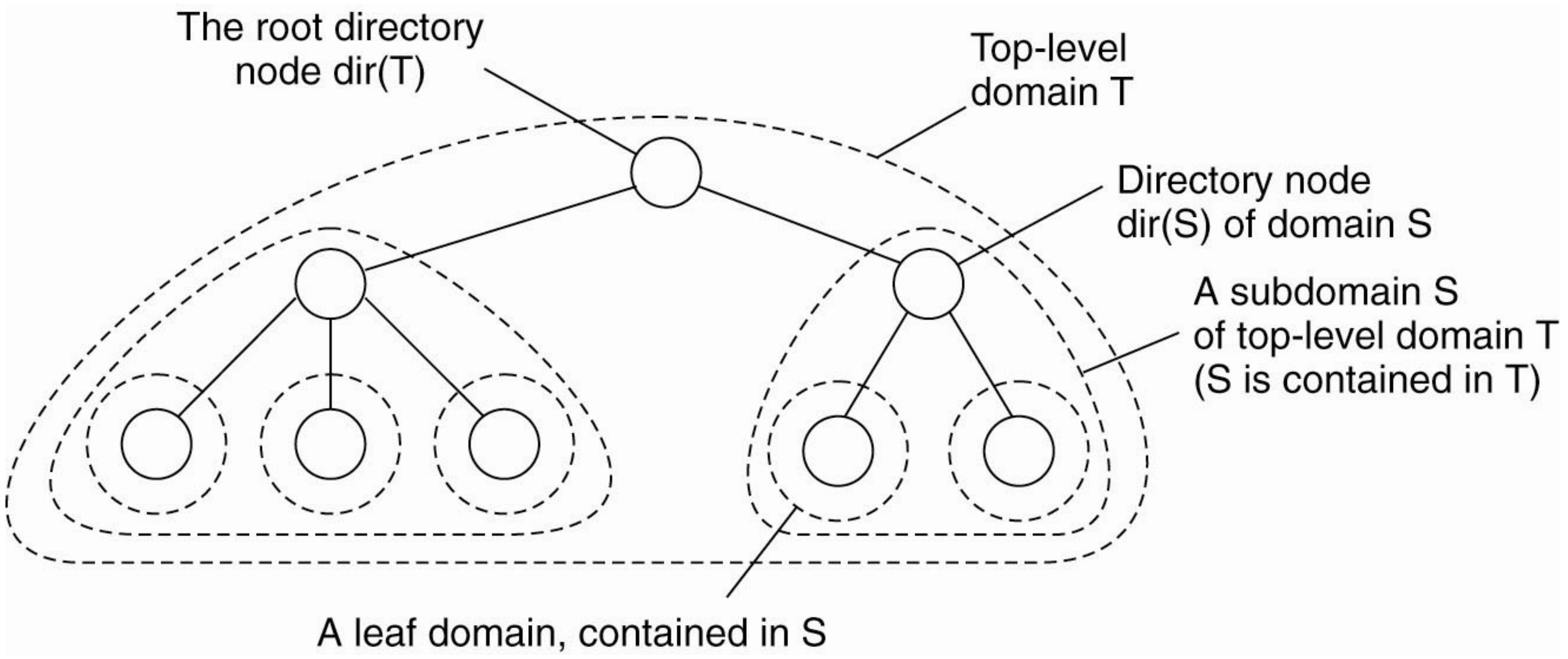

The root directory node dir(T)
Top-level domain T
Directory node dir(S) of domain S
A subdomain S of top-level domain T (S is contained in T)
A leaf domain, contained in S

Figure 5-5. Hierarchical organization of a location service into domains, each having an associated directory node.

# Hierarchical Approaches (2)

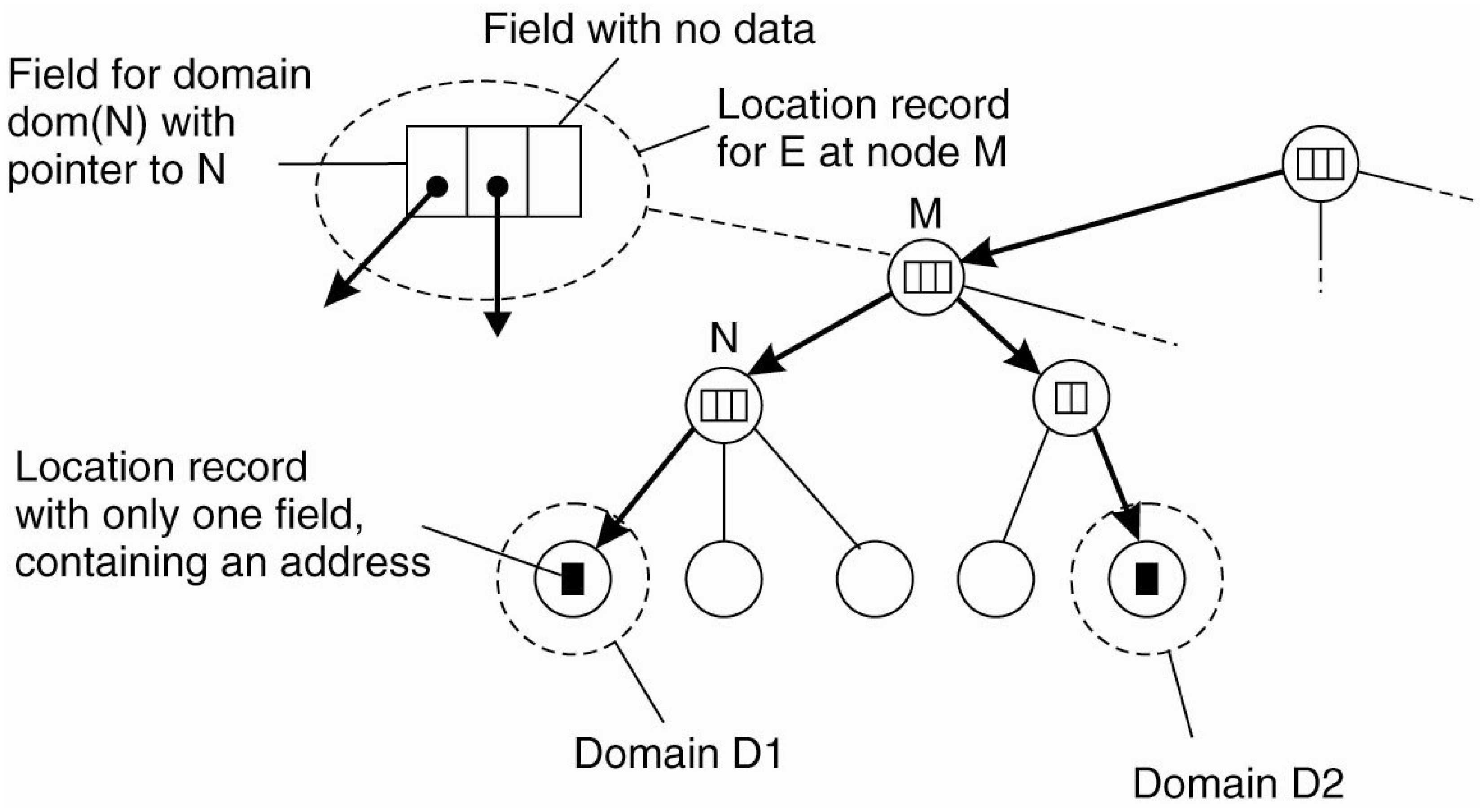

Field with no data
Field for domain dom(N) with pointer to N
Location record for E at node M
M
N
Location record with only one field, containing an address
Domain D1
Domain D2

Figure 5-6. An example of storing information of an entity having two addresses in different leaf domains.

# Hierarchical Approaches (3)

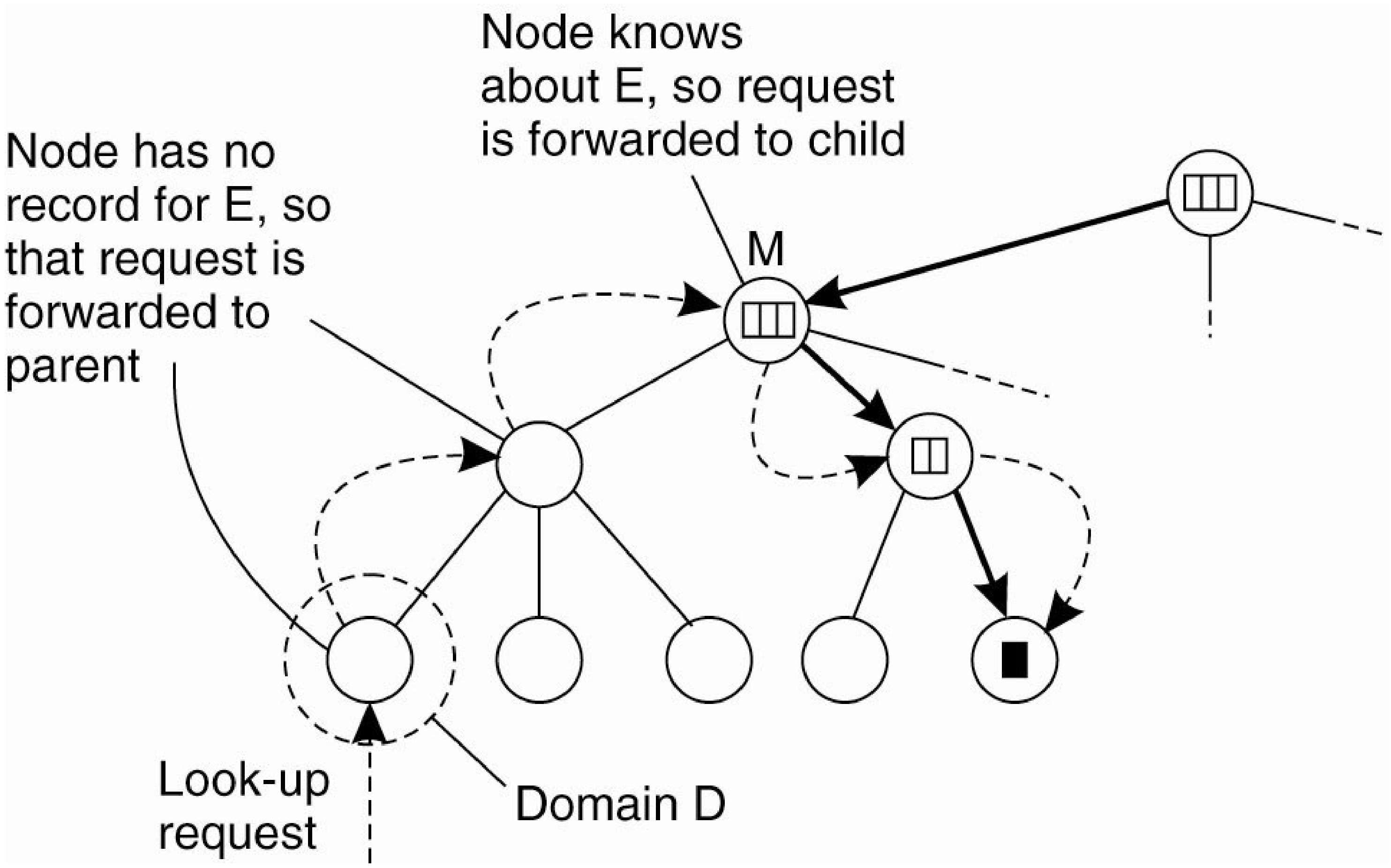

Node knows about E, so request is forwarded to child
Node has no record for E, so that request is forwarded to parent
M
Look-up request
Domain D

Figure 5-7. Looking up a location in a hierarchically organized location service.

# Hierarchical Approaches (4)

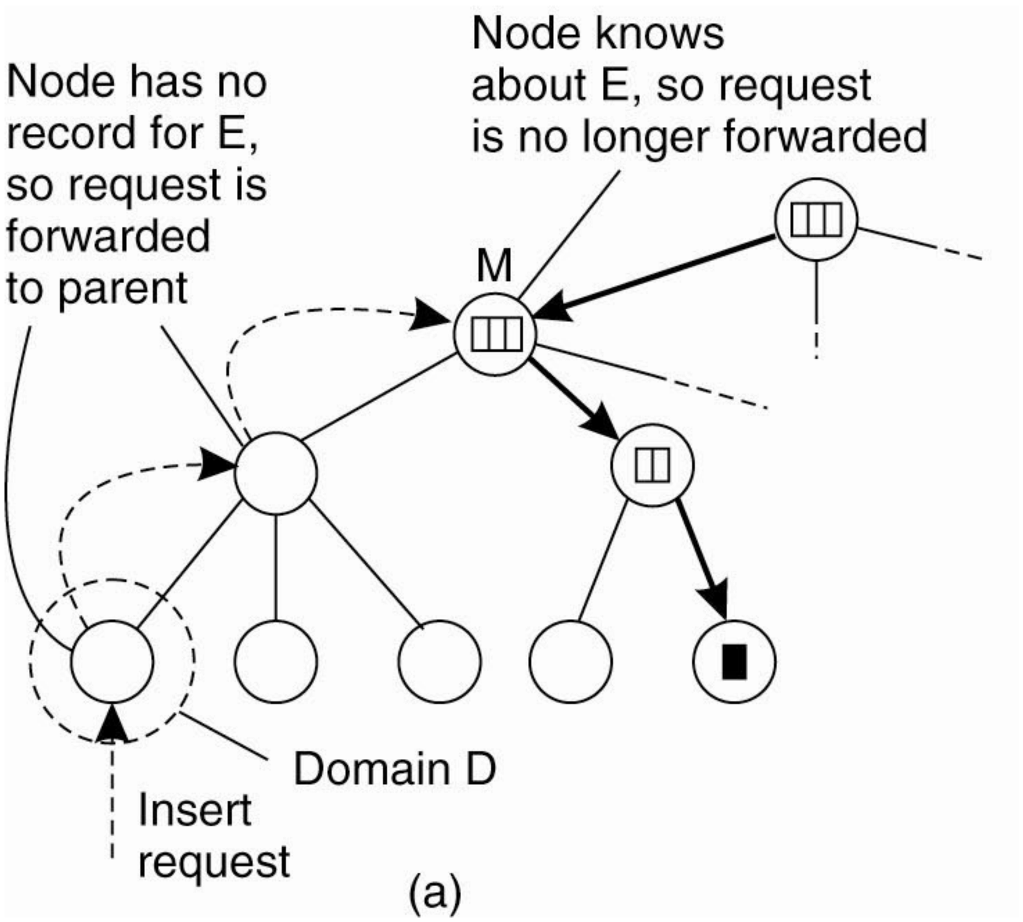

Node has no record for E, so request is forwarded to parent
Node knows about E, so request is no longer forwarded
M
Domain D
Insert request
(a)

Figure 5-8. (a) An insert request is forwarded to the first node that knows about entity E.

# Hierarchical Approaches (5)

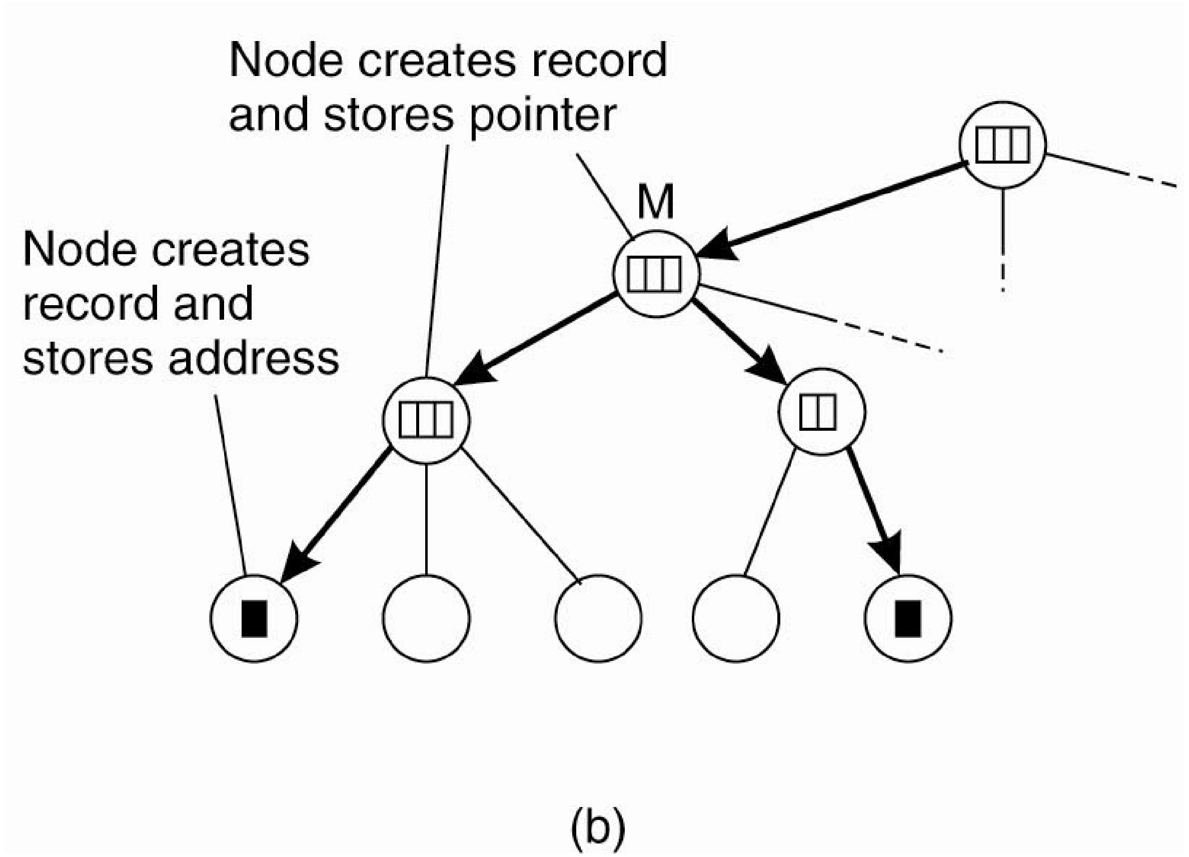

Node creates record and stores pointer
M
Node creates record and stores address
(b)

Figure 5-8. (b) A chain of forwarding pointers to the leaf node is created.

# Structured Naming

- Name spaces
- Name resolution
- Name space implementation
- Example: *Domain Name System* (DNS)

# Name Spaces (1)

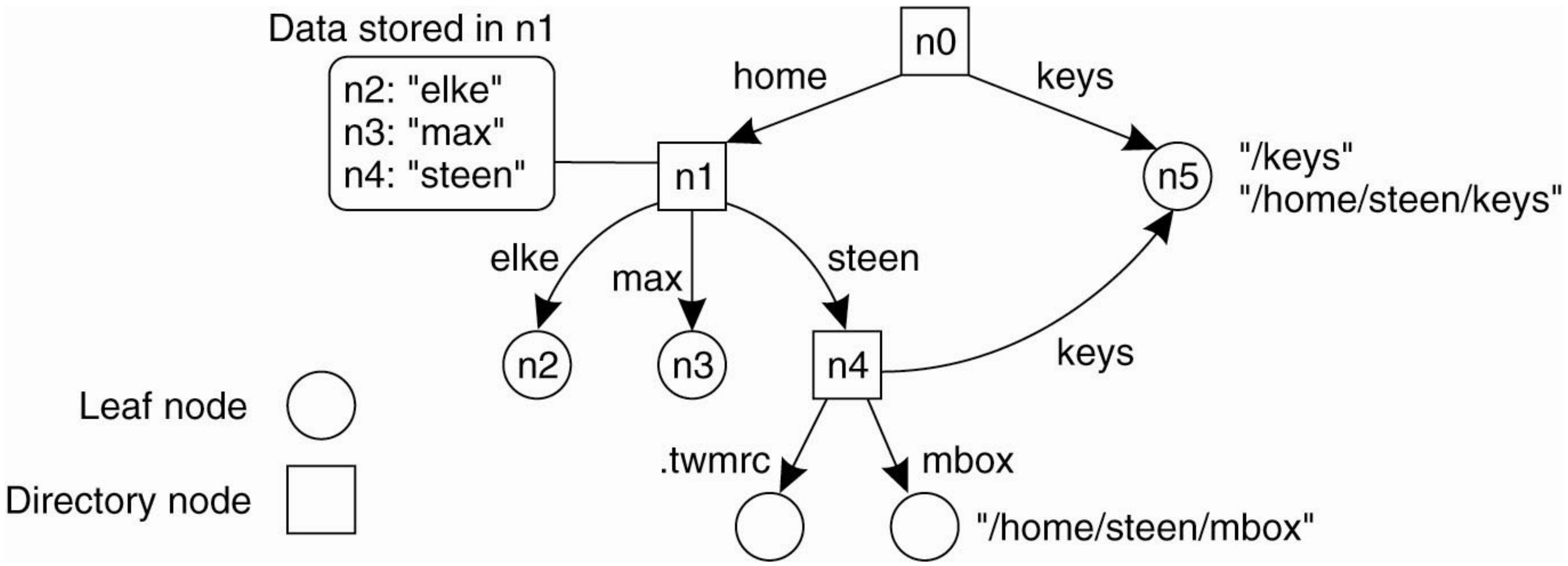


Figure 5-9. A general naming graph with a single root node.

## Name Resolution

- The process of looking up a name is called *name resolution*. Given a path name, it should be possible to look up any information stored in the node referred to by that name

- Knowing how and where to start name resolution is generally referred to as *closure mechanism*

- Use of *aliases*: hard links and symbolic links

# Linking and Mounting (1)

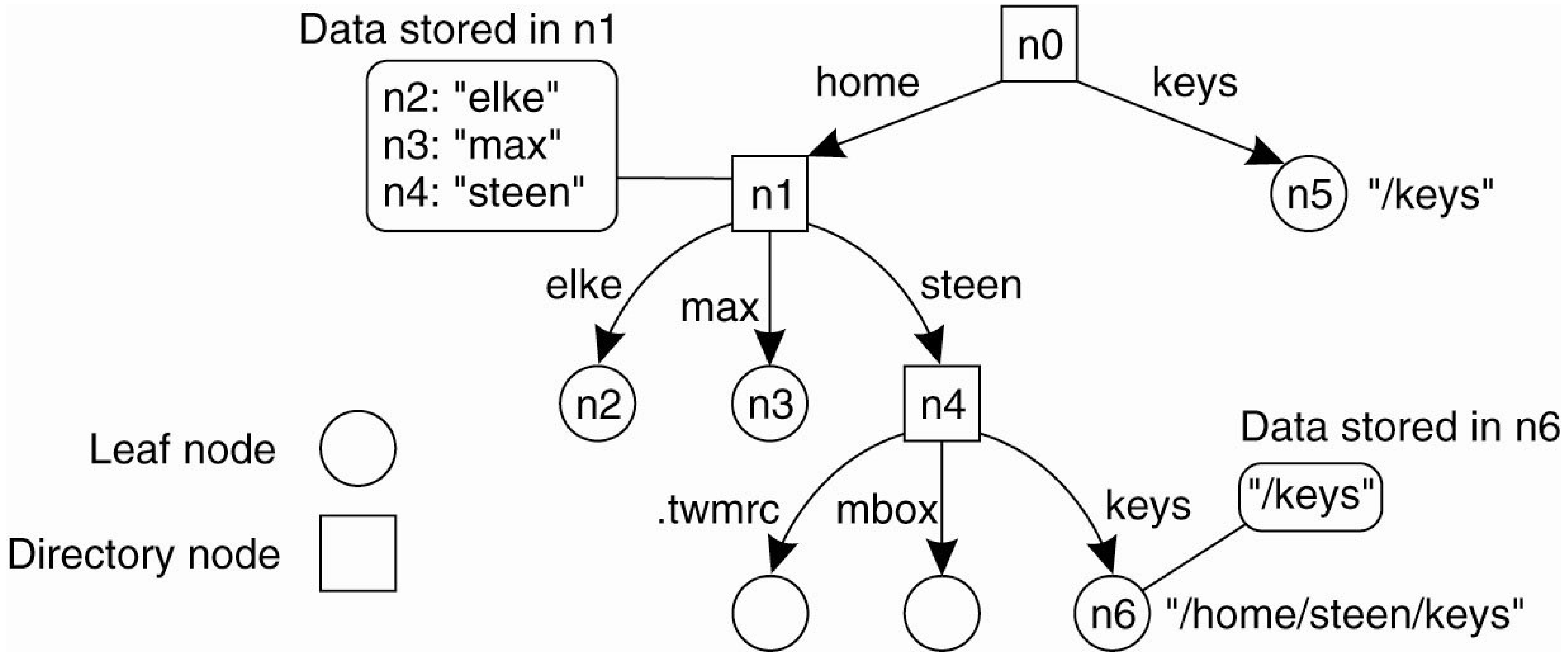


Figure 5-11. The concept of a symbolic link explained in a naming graph.

## Linking and Mounting (2)

- Distributed name spaces can be merged transparently using the concept of mount points. To mount a remote name space in a distributed system requires at least the following information:
  - Name of an access protocol
  - Name of the server
  - Name of the mounting point in the foreign name space
- *NFS* (Network File System) 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) server

# Linking and Mounting (3)

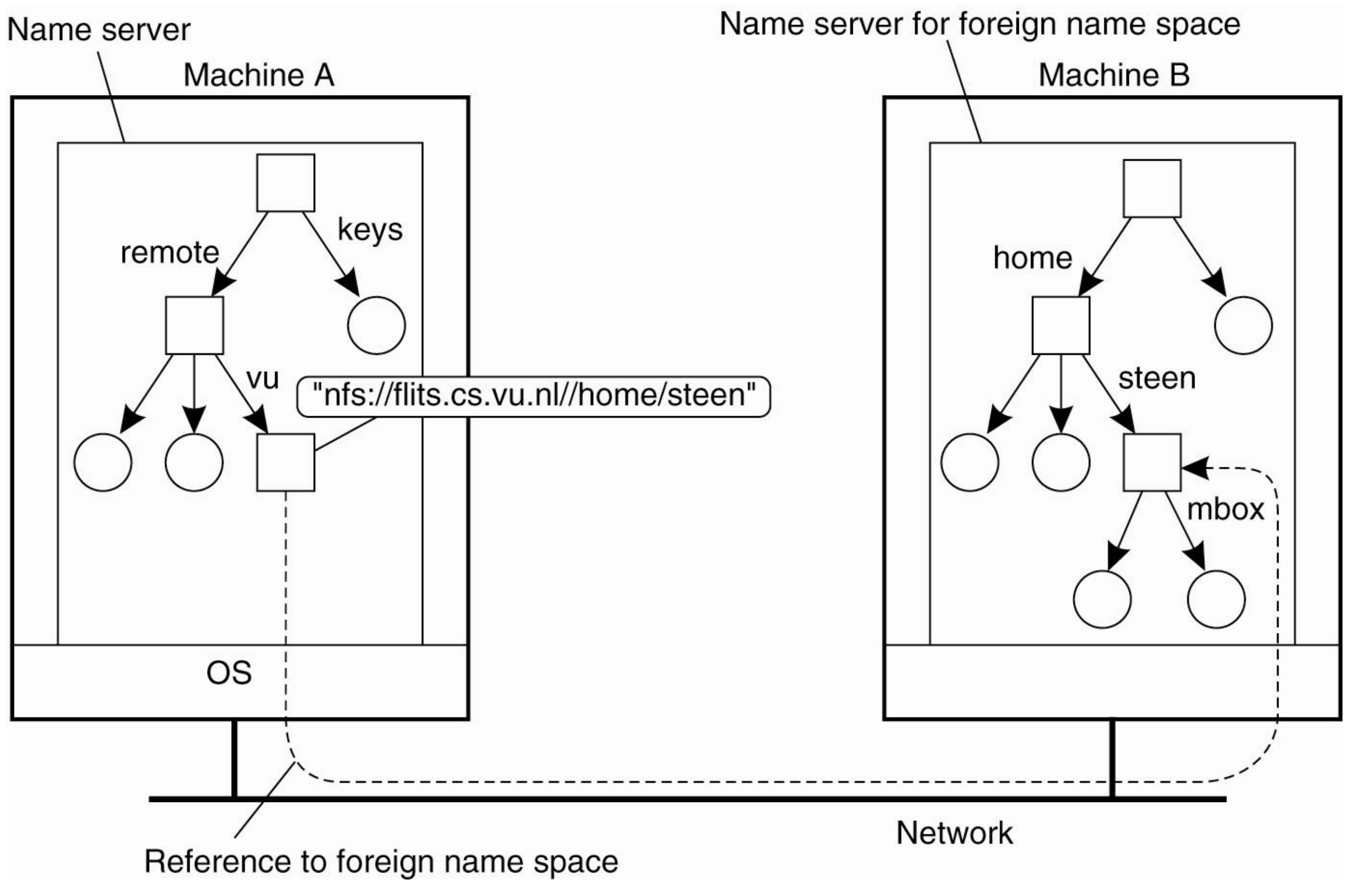

Name server
Machine A
remote
keys
vu
"nfs://flits.cs.vu.nl//home/steen"
OS
Name server for foreign name space
Machine B
home
steen
mbox
Network
Reference to foreign name space

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

## Name Space Implementation

- A name space is implemented by a naming service that allows users to add, remove and look up names. It is implemented by name server(s)
- For a distributed system limited to a LAN, it may be feasible to implement the name service with a single name server
- However, in a large-scale distributed system spread across a large geographical area, it is necessary to distribute the implementation over many name servers

# Name Space Distribution (1)

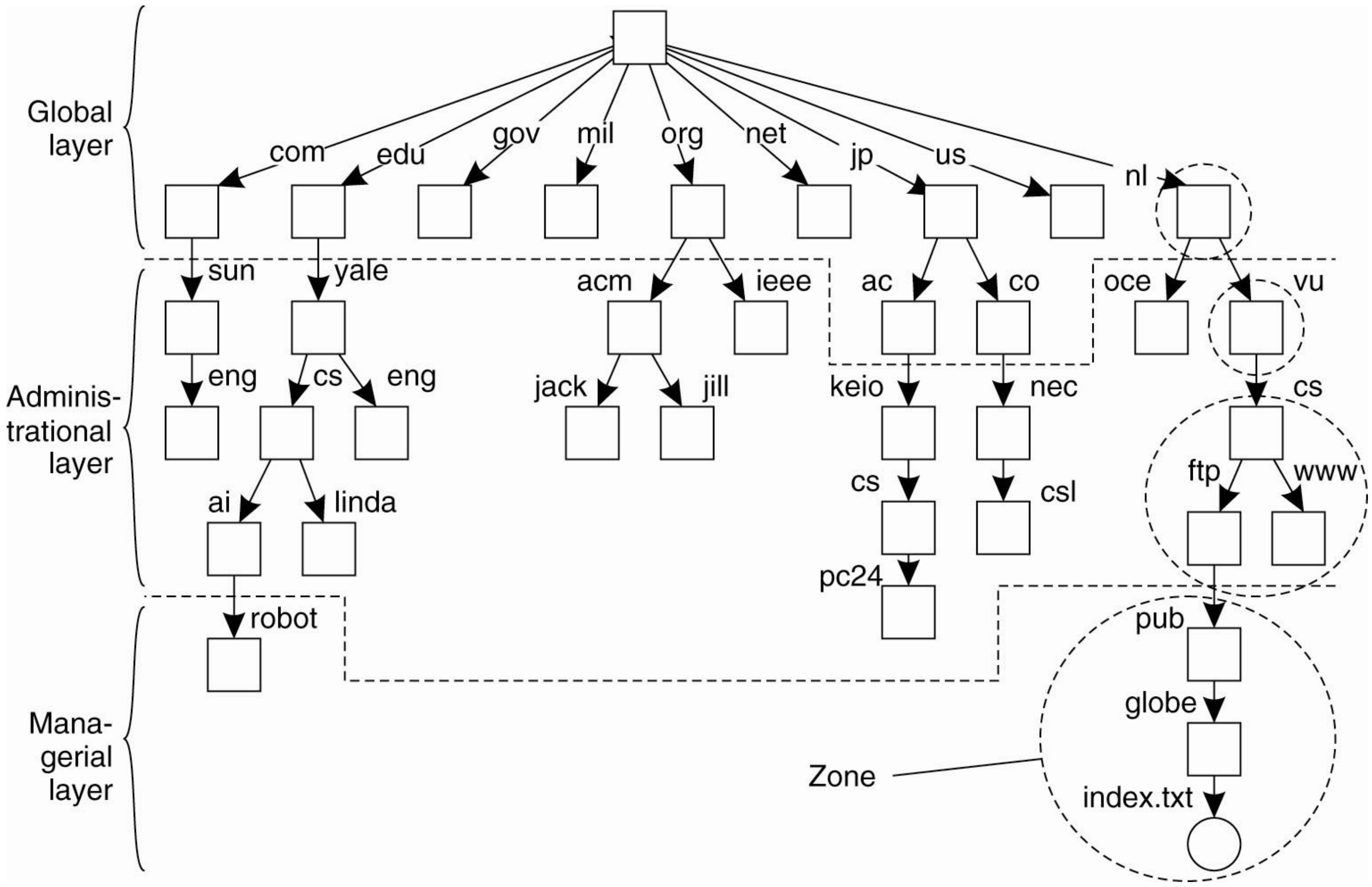
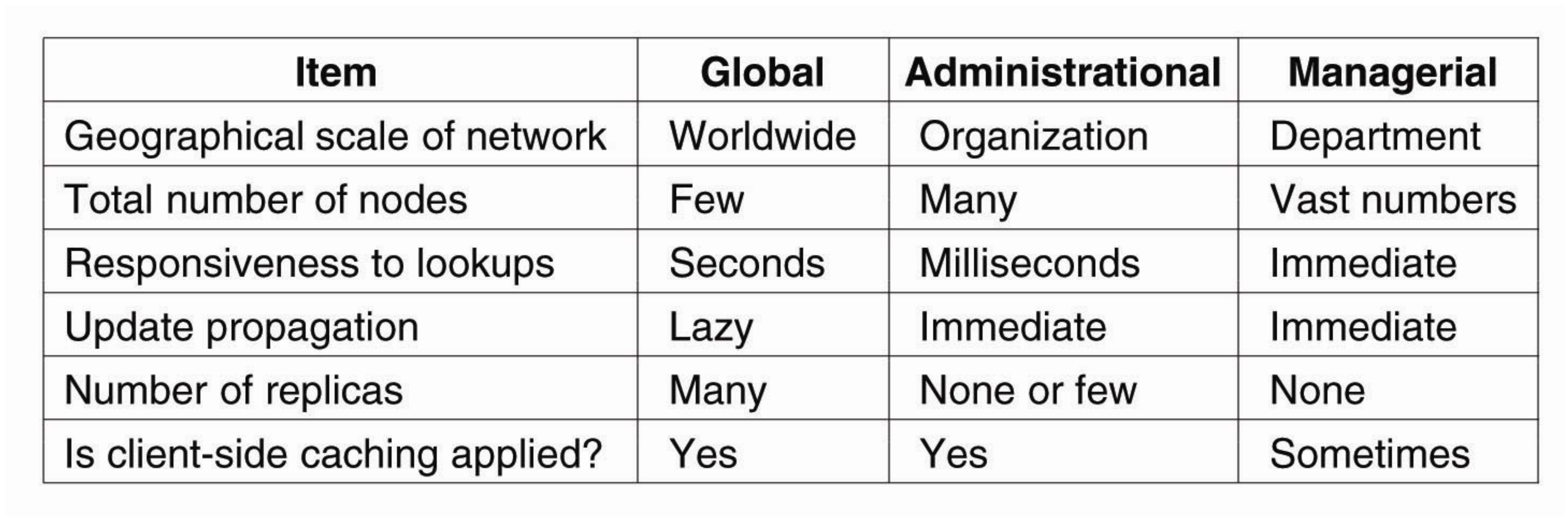


Figure 5-13. An example partitioning of the DNS name space, including Internet-accessible files, into three layers.

# Name Space Distribution (2)



| 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 |

Figure 5-14. 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.

# Implementation of Name Resolution (1)

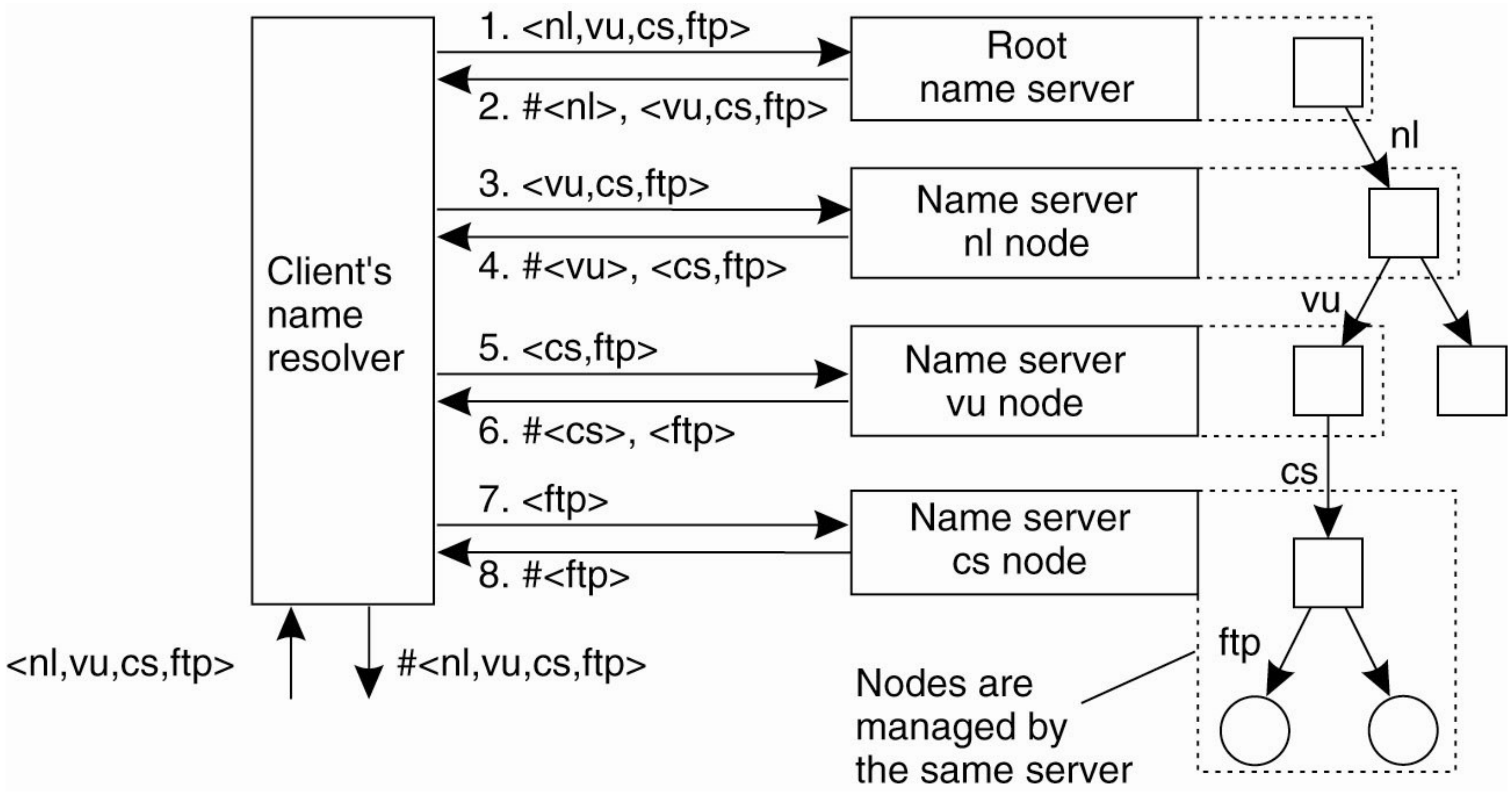


Figure 5-15. The principle of *iterative* name resolution.

# Implementation of Name Resolution (2)

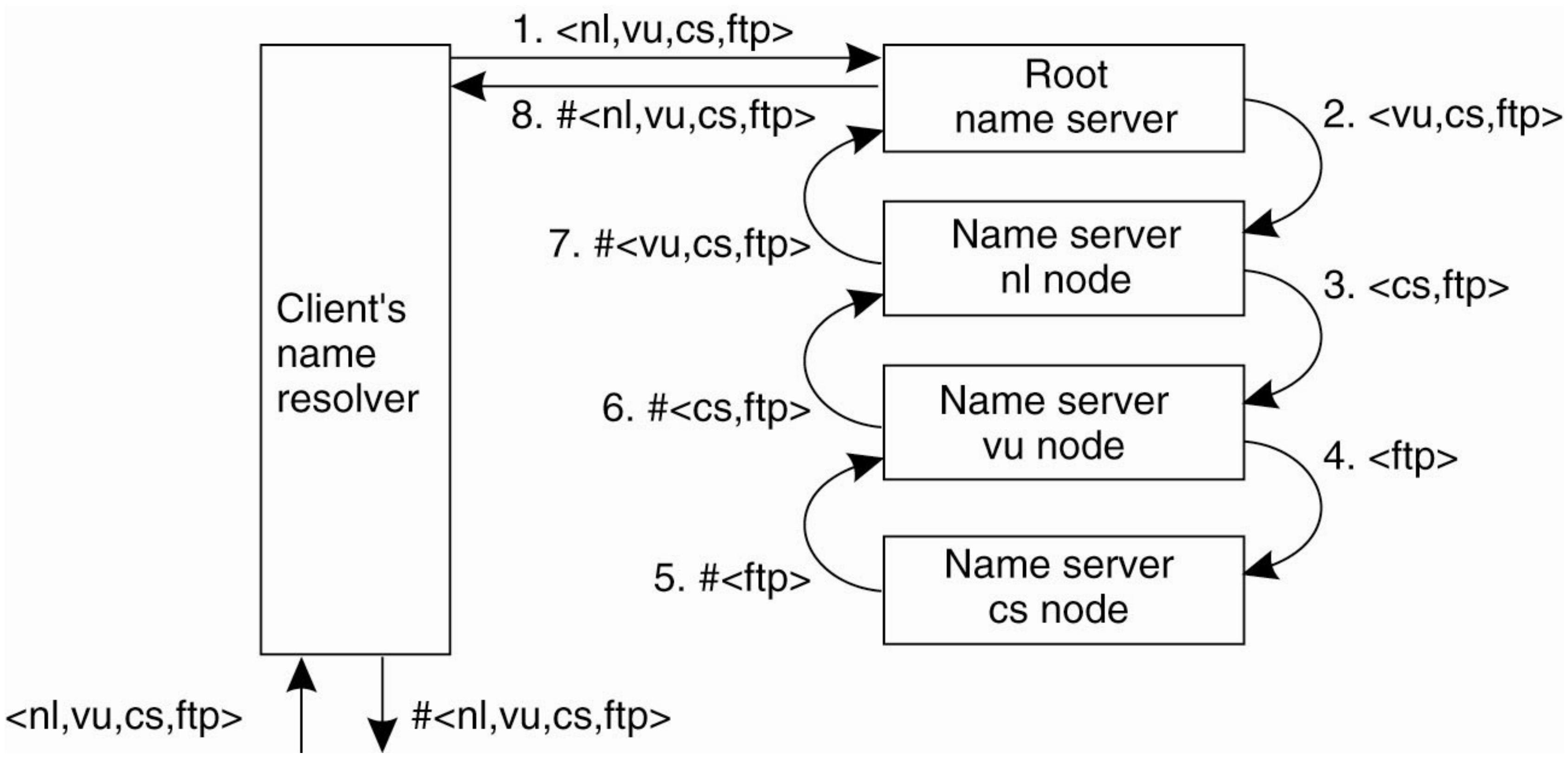
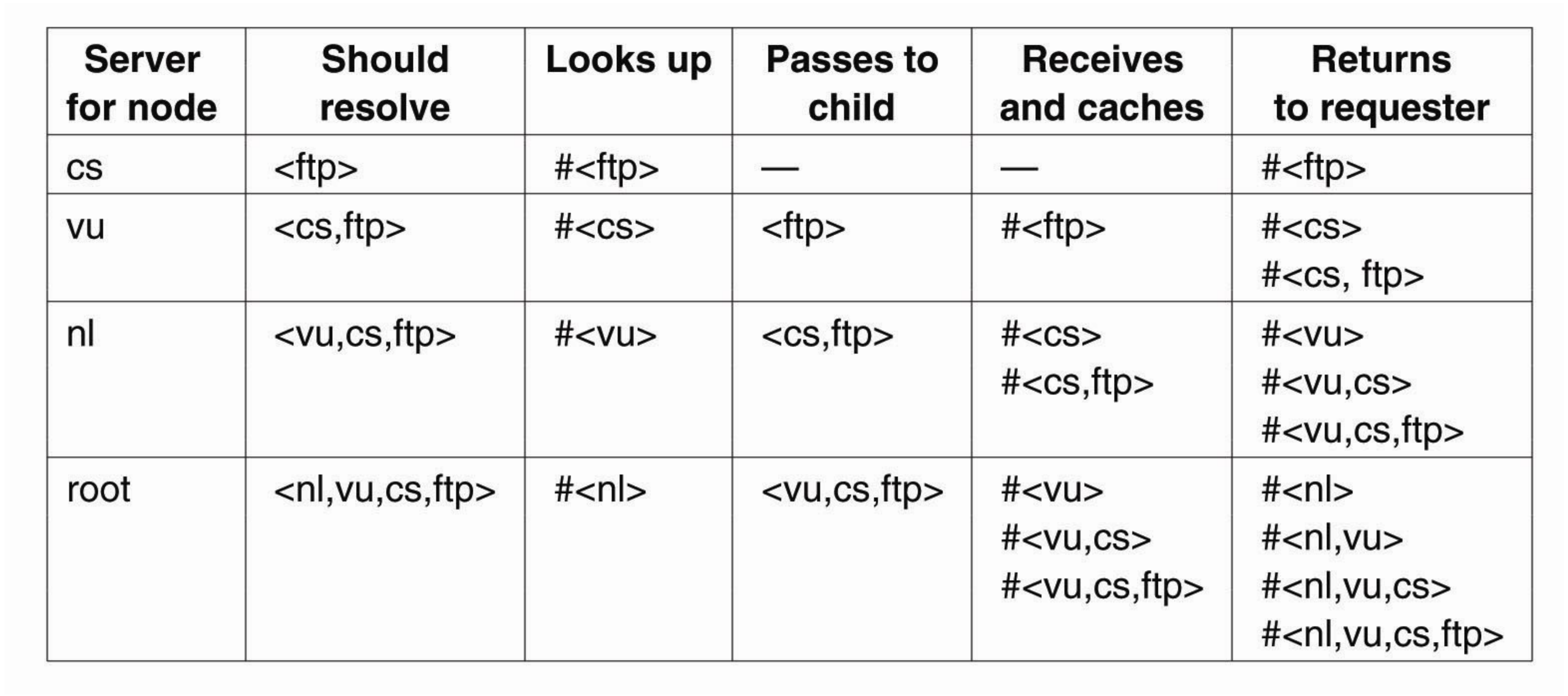


Figure 5-16. The principle of *recursive* name resolution.

# Implementation of Name Resolution (3)

- Recursive name resolution allows each name server to gradually learn the address of each name server responsible for lower-level nodes. As a result, caching can be effectively used to enhance performance.
- Iterative name resolution, caching is necessarily restricted to the client's name resolver. To improve performance, many organizations use a local intermediate name server that is shared by their client machines
- Recursive name resolution is often cheaper with respect to communication
- Recursive name resolution puts a higher performance demand on each name server. Thus name servers at the global layer only support iterative resolution.

# Implementation of Name Resolution (4)



| 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><br>#<cs, ftp> |
| nl | <vu,cs,ftp> | #<vu> | <cs,ftp> | #<cs><br>#<cs,ftp> | #<vu><br>#<vu,cs><br>#<vu,cs,ftp> |
| root | <nl,vu,cs,ftp> | #<nl> | <vu,cs,ftp> | #<vu><br>#<vu,cs><br>#<vu,cs,ftp> | #<nl><br>#<nl,vu><br>#<nl,vu,cs><br>#<nl,vu,cs,ftp> |

Figure 5-17. Recursive name resolution of <*nl, vu, cs, ftp*>. Name servers cache intermediate results for subsequent lookups.

# Implementation of Name Resolution (5)

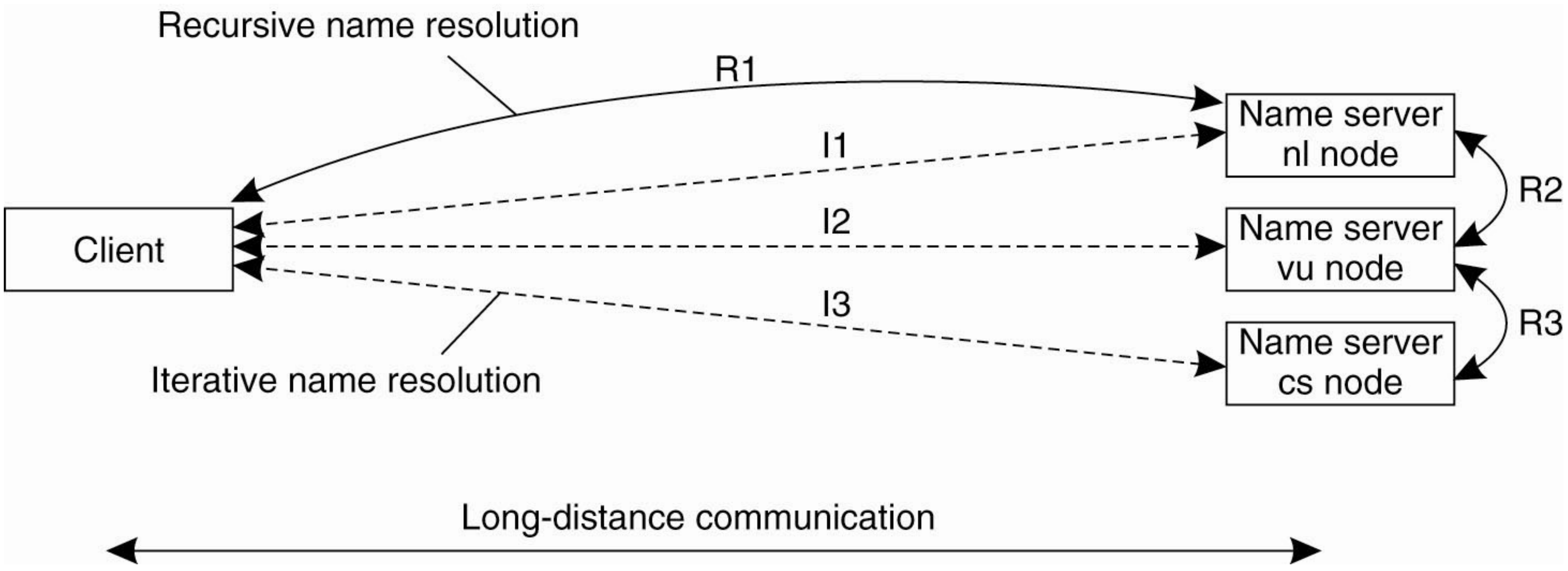

Recursive name resolution
R1
I1
I2
I3
R2
R3
Client
Name server nl node
Name server vu node
Name server cs node
Iterative name resolution
Long-distance communication

Figure 5-18. The comparison between recursive and iterative name resolution with respect to communication costs.

# DNS: Domain Name Service (1)

- One of the largest distributed system in use today
- Used for mapping hostnames to IP addresses on the Internet
- Uses an iterative lookup by default, with recursive lookup as an option
  - The three major components are: *Domain Name Space* and *Resource Records*
  - *Name Servers* are server programs which hold information about the domain tree's structure and set information
  - *Resolvers* are programs that extract information from name servers in response to client requests. Resolvers must be able to access at least one name server and use that name server's information to answer a query directly, or pursue the query using referrals to other name servers

Look up RFC 1034 and RFC 1035 for more information.

# DNS:Domain Name Service (2)

- From the user's point of view, the domain system is accessed through a simple library or system call to a local resolver. The domain space consists of a single tree and the user can request information from any section of the tree
- From the resolver's point of view, the domain system is composed of an unknown number of name servers. Each name server has one or more pieces of the whole domain tree's data, but the resolver views each of these databases as essentially static
- From a name server's point of view, the domain system consists of separate sets of local information called zones. The name server has local copies of some of the zones. The name server must periodically refresh its zones from master copies in local files or foreign name servers. The name server must concurrently process queries that arrive from resolvers

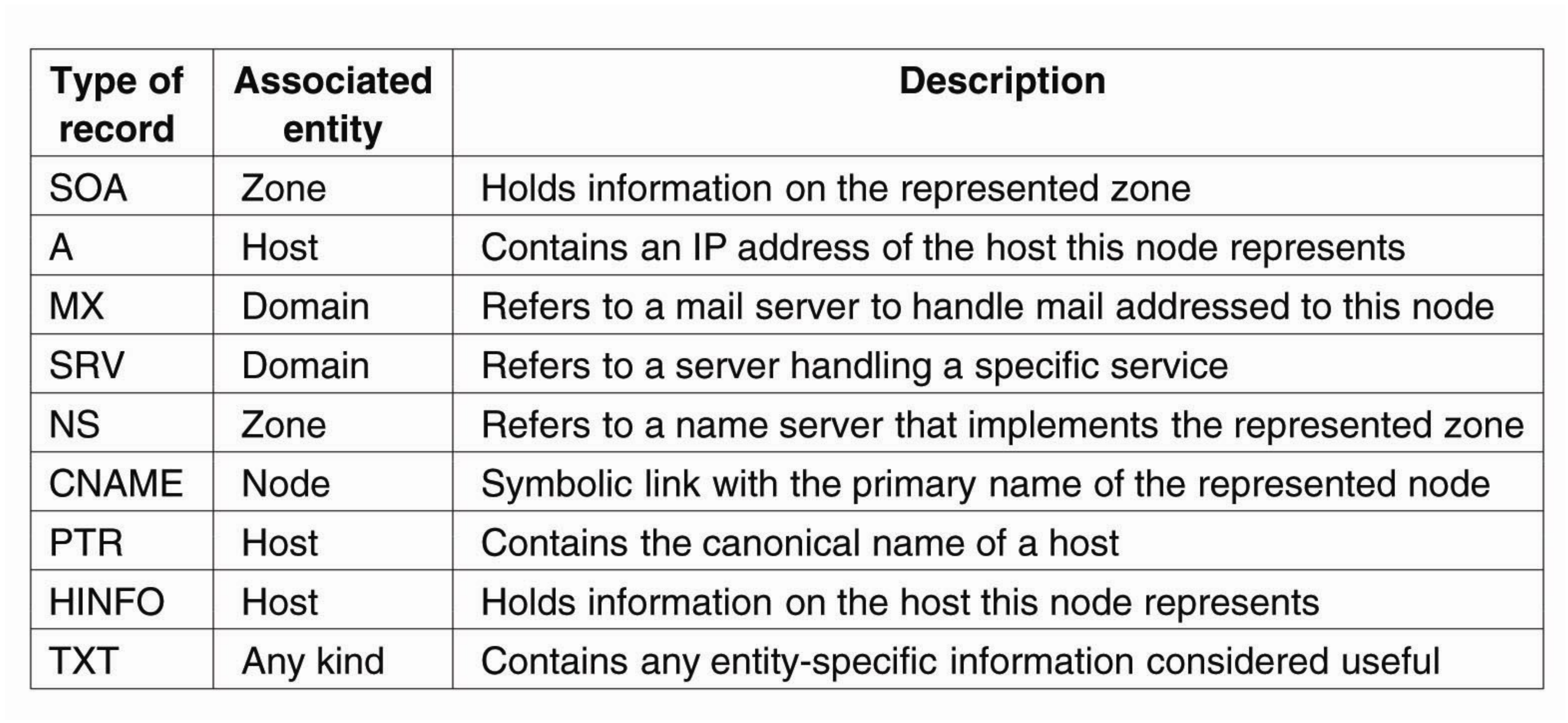
# DNS:Domain Name Service (3)

- A label has a maximum length of 63 characters. A path is limited to 255 characters. Labels are separated by dots, starting with the rightmost dot that represents the root. So the proper name for cs.boisestate.edu is cs.boisestate.edu. (the rightmost dot is usually omitted for readability)
- The name space is a tree. A subtree is called a *domain*. A path name to its root node is called a *domain name*
- The contents of a node is formed by a collection of *resource records*. There are various types of resource records.
- A domain can be implemented by several (non-overlapping) zones. An SOA (Start Of Authority) resource record contains information about the name server for that zone
- Each host has a canonical or primary name identified by a resource record labeled as A. They can have several aliases using the CNAME record

## DNS: Domain Name Service (4)

- An A (address) record represents a particular host in the Internet. If a host has several IP addresses, the node will contain an A record for each address
- An MX (mail exchange) record is a symbolic link to a node representing a mail server. There may be several MX records stored in a node
- An SRV (server) record contains the name of a server for a specific type of service. For example, _http._tcp.cs.vu.nl could be a SRV record that points to a web server.
- Nodes that represent a zone contain one or more NS (name server) records
- Inverse mapping of IP addresses to host names is maintained with PTR (pointer) records. It keeps a domain named *in-addr.arpa*, which nodes that represent Internet hosts and which are named by the IP address of the represented host

# DNS: Domain Name Service (5)



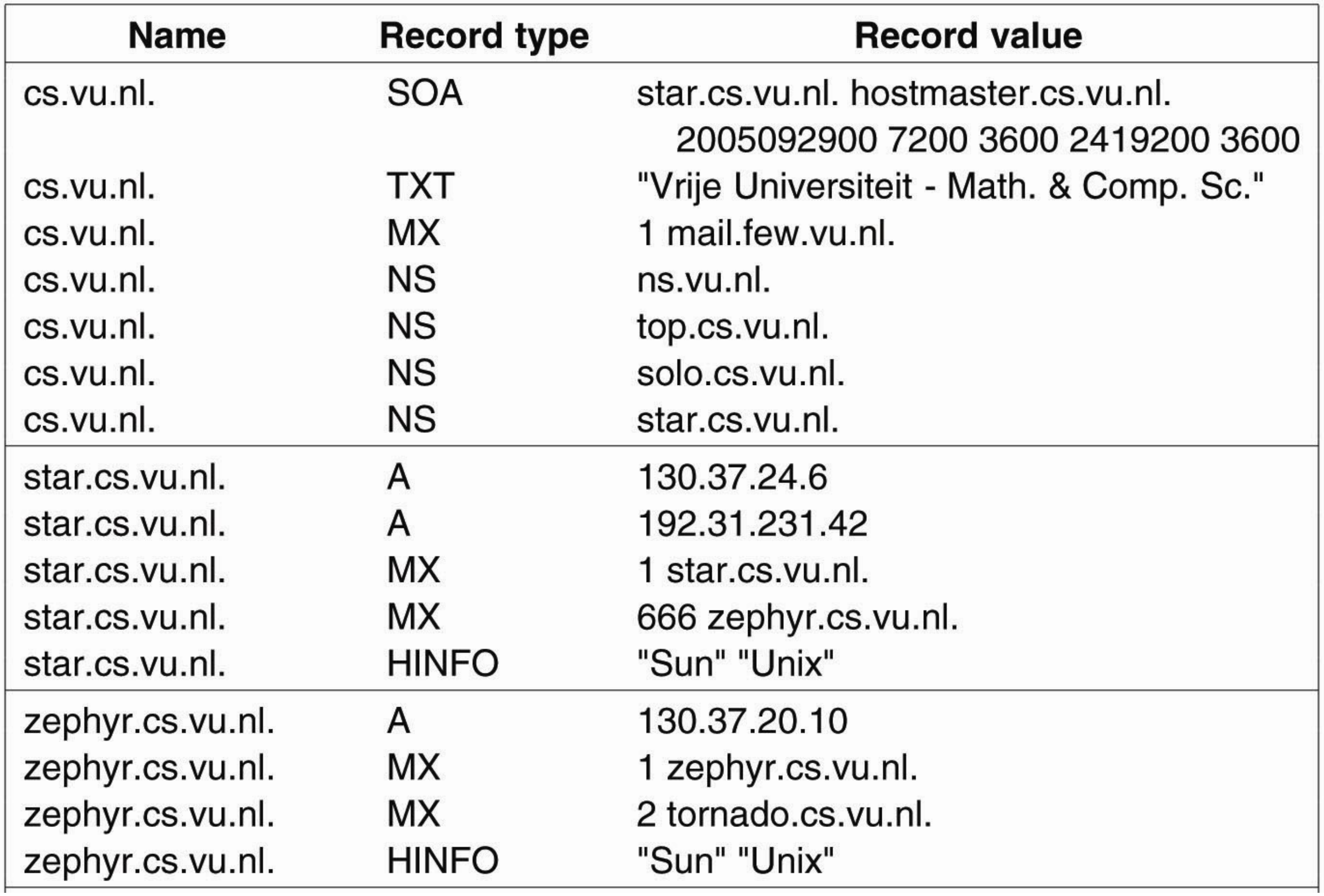
| Type of record | Associated entity | Description |
|---|---|---|
| SOA | Zone | Holds information on the represented zone |
| A | Host | Contains an IP address of the host this node represents |
| MX | Domain | Refers to a mail server to handle mail addressed to this node |
| SRV | Domain | Refers 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 |

Figure 5-19. The most important types of resource records forming the contents of nodes in the DNS name space.

## DNS: Domain Name Service (6)

- The DNS name space is divided into a global layer and an administrational layer. The managerial layer, which is generally formed by local file systems, is formally not part of the DNS
- Each zone is implemented by a name server, which is virtually always replicated for availability. Updates are handled by the primary name server by modifying DNS database local to the primary name server. Secondary name servers request the primary server to transfer its content via a *zone transfer*
- A DNS database is implemented as a collection of text files, of which the most important one contains all the nodes in a particular zone
- A SOA record also contains a serial number (that must be incremented when any resource record is changed), refresh, retry, expire and min TTL (Time To Live) for the zone

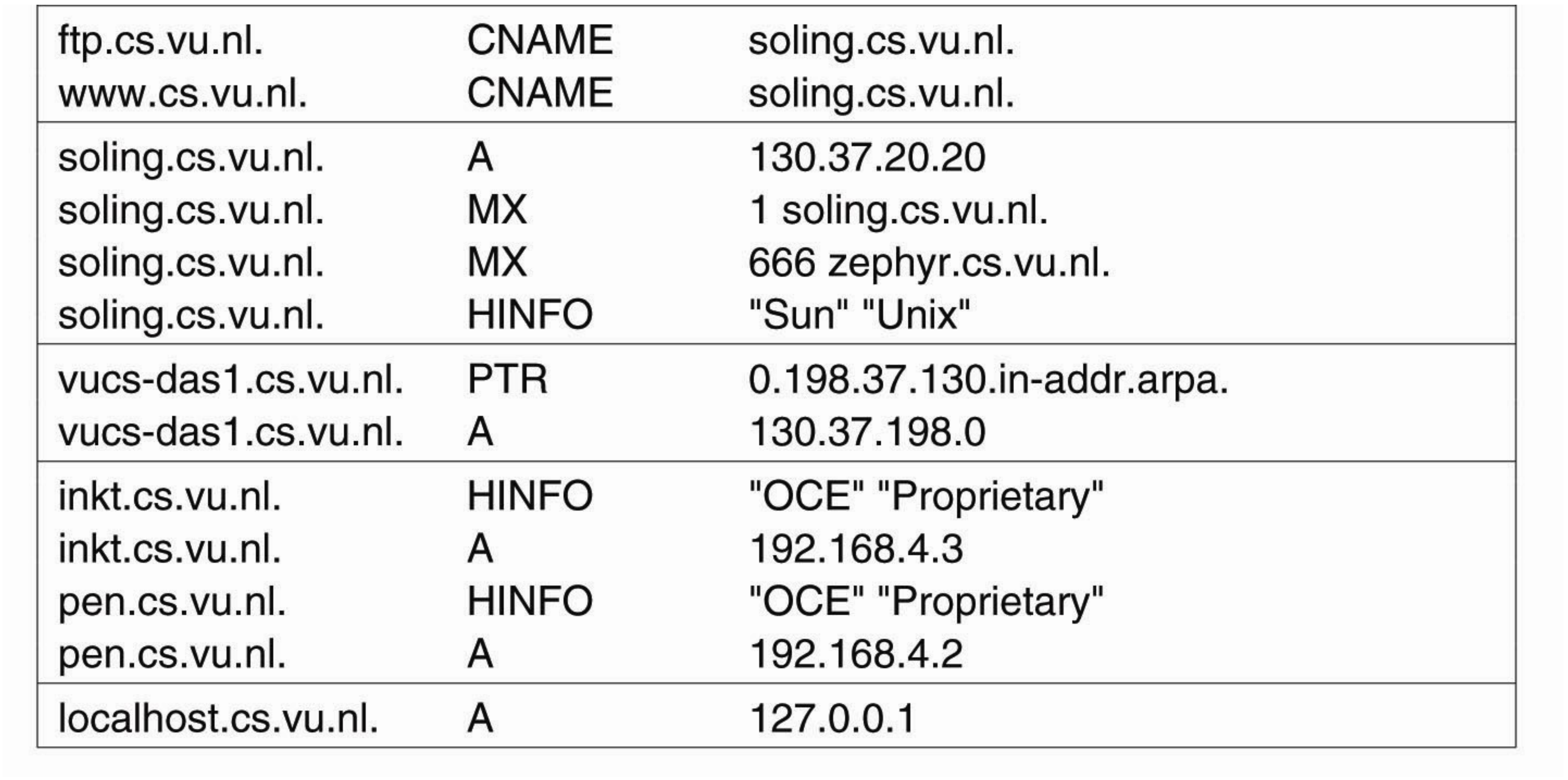
# DNS: Domain Name Service (7)



| 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" |

Figure 5-20. An excerpt from the DNS database for the zone *cs.vu.nl*.

# DNS: Domain Name Service (8)



| | | |
|---|---|---|
| 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 |

Figure 5-20. An excerpt from the DNS database for the zone *cs.vu.nl*.

## DNS: Domain Name Service (9)

- DNS primarily uses UDP on port number 53 to serve requests. DNS queries/replies consist of a single UDP packet. TCP is used when the response data size exceeds 512 bytes, or for tasks such as zone transfers
- Useful exercises:
  - Setup a caching name server at your home machine. This should speed up web browsing significantly
  - Install a name server (BIND is the most common one) and setup a name server for your local network at home

## DNS: Domain Name Service (10)

- The DNS is controlled by ICANN (*Internet Corporation for Assigned Names and Numbers*) but the root zones are controlled by US Department of Commerce
- *Thirteen root-servers* that are highly distributed and resilient.
- DNS security issues
  - DNS cache poisoning
  - Domain Name System Security Extensions (DNSSEC) to have cryptographically signed transactions
  - Phishing
- Discuss bigger DNS examples

# DNS: Domain Name Service (10) Boise State DNS System

- Adonis 1000 appliances from Bluecat to manage our DNS. The boxes are Linux under the hood with a GUI client for ease of management. Two replicated name servers
- Bind version 9.7.4-P1
- 12000+ A resource records, 500+ CNAME records, 18 SRV records
- OIT Active Directory domain controllers are allowed to update DNS. TXT record is generated automatically when an active directory client computer obtains a DHCP address and dynamically registers its name with DNS
- Special SRV records automatically generated by Active Directory Domain Controllers. For example for LDAP, Kerberos, KMS (Key Management System)

# DNS: Domain Name Service (11) Boise State DNS System

- TXT record is generated automatically when an active directory client computer obtains a DHCP address and dynamically registers its name with DNS

```
Example of DHCP record in DNS
active_earth-PC A 132.178.150.240
$TTL 1800; 30 minutes TXT "31631f3e2ac0f909bef3b57120a58c8a1f”
```

- Example of special records automatically generated by Domain Controllers

```
      $ORIGIN _tcp.main-campus._sites.dc._msdcs.boisestate.edu.
      _kerberos SRV 0 100 88 drycreek1.boisestate.edu.
      SRV 0 100 88 drycreek2.boisestate.edu.
      SRV 0 100 88 DRYCREEK3.boisestate.edu.
      _ldap SRV 0 100 389 drycreek1.boisestate.edu.
   SRV 0 100 389 drycreek2.boisestate.edu.
   SRV 0 100 389 DRYCREEK3.boisestate.edu.
```

## Attribute-based Naming

- As more information becomes available, it becomes important to effectively search for entities. The user should be able to search based just on some attributes
- Each entity has certain attributes. Each attribute says something about the entity
- Users can search by constraining the attributes
- Attribute-based naming systems are also known as *directory services*
- More general model is using *resource description framework* (RDF). Resources are described as triplets consisting of a subject, predicate and an object. E.g. (person, name, Alice)

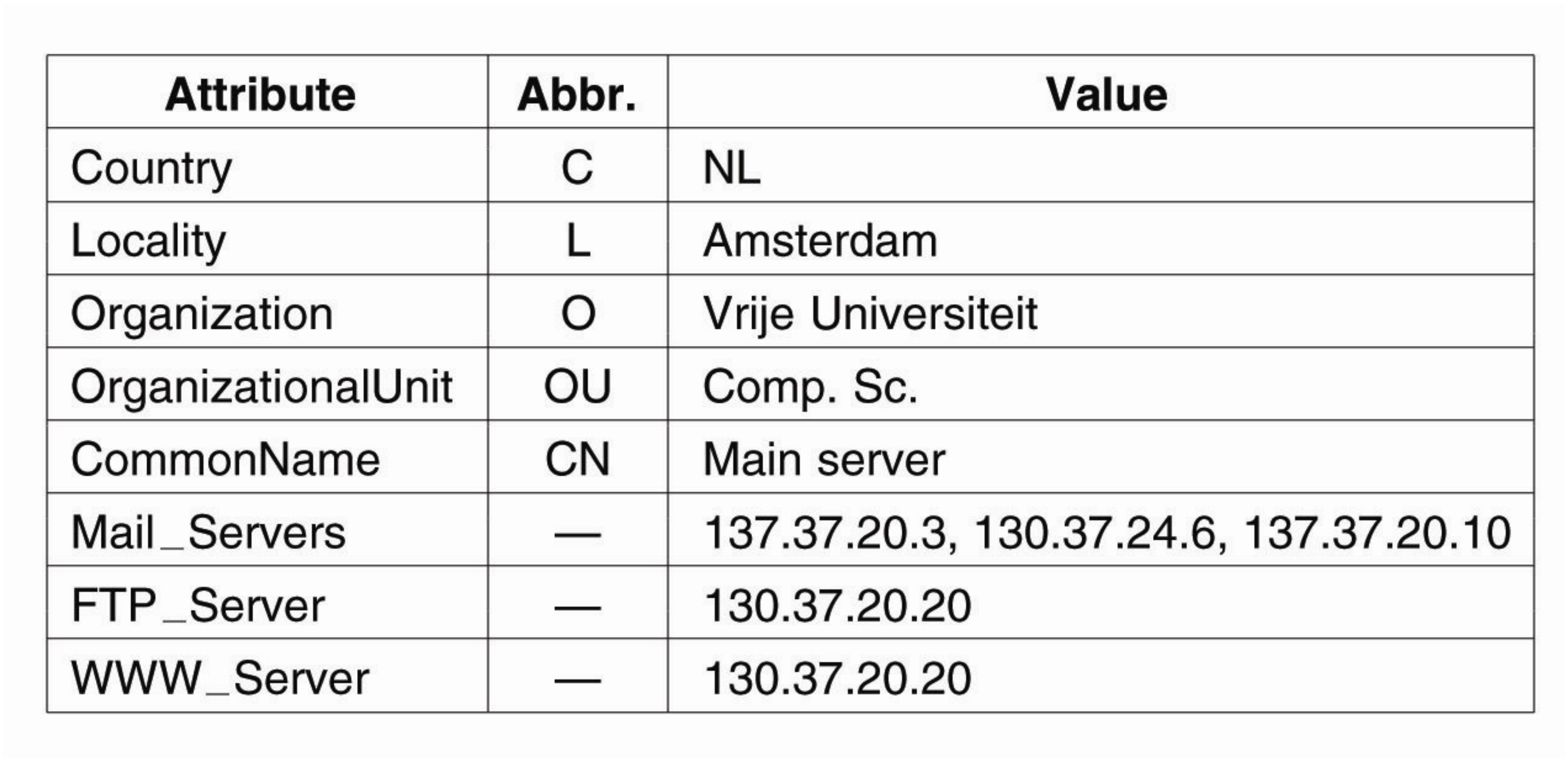
# Hierarchical Implementations: LDAP (1)

- LDAP (*Lightweight Directory Access Protocol* ). A simplified protocol implementing the X.500 directory services. LDAP is an application level protocol, implemented directly on top of TCP
- Used by email programs for contact information but can also be used to look for encryption certificates, pointers to printers or other services, sharing passwords between services etc
- LDAP defines the protocol to be used between servers and clients or servers and servers
- LDAP clients starts a LDAP session by contacting a LDAP server on default TCP port 389

# Hierarchical Implementations: LDAP (3)

- Clients can send multiple requests and server can respond in any order (with some exceptions)
- Common operations
  - *StartTLS* — use the LDAPv3 Transport Layer Security (TLS) extension for a secure connection
  - *Bind* — authenticate and specify LDAP protocol version
  - *Search* — search for and/or retrieve directory entries
  - *Compare* — test if a named entry contains a given attribute value
  - *Add/Delete/Modify* an entry
  - *Modify Distinguished Name* (DN) — move or rename an entry
  - *Abandon* — abort a previous request
  - *Extended Operation* — generic operation used to define other operations
  - *Unbind* — close the connection (not the inverse of Bind)

# Hierarchical Implementations: LDAP (4)



| 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.

## Hierarchical Implementations: LDAP (5)

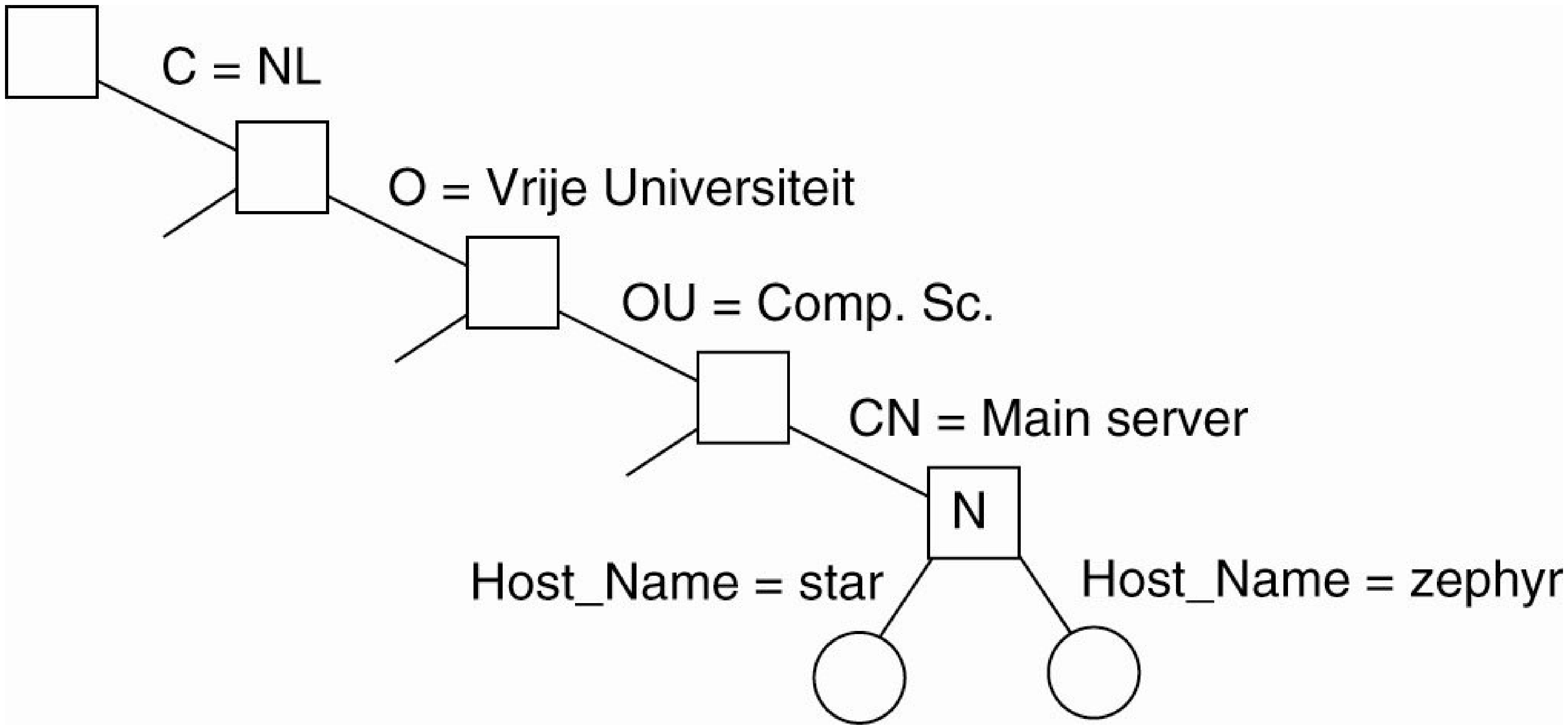


Figure 5-23. (a) Part of a directory information tree.

# Hierarchical Implementations: LDAP (6)

| 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. (b) Two directory entries having *Host_Name* as RDN.

# Hierarchical Implementations: LDAP (7)

- Collection of all directory entries in an LDAP directory service is called a *directory information base* (*DIB*). Each record is uniquely named and appears as a sequence of naming attributes. These attributes are known as *relative distinguished names* (*RDN*)
- The hierarchy of the collection of directory entries is known as the *directory information tree* (*DIT*). This is a naming graph in which each node represents a directory entry
- A large-scale DIT is usually partitioned across several *directory service agents* (*DSA*). Clients are represented by *directory user agents* (*DUA*), which are similar to a name resolver in DNS

# Hierarchical Implementations: LDAP (8)

- Microsoft's *Active Directory* allows a forest of LDAP domains! To reduce search complexity, a global index server is searched first
- Every tree in LDAP needs to be accessible at the root (domain controller in Active Directory terminology). This can be known under DNS with a SRV record
- *UDDI* (Universal Directory and Discovery Implementation) is another example of structured naming system