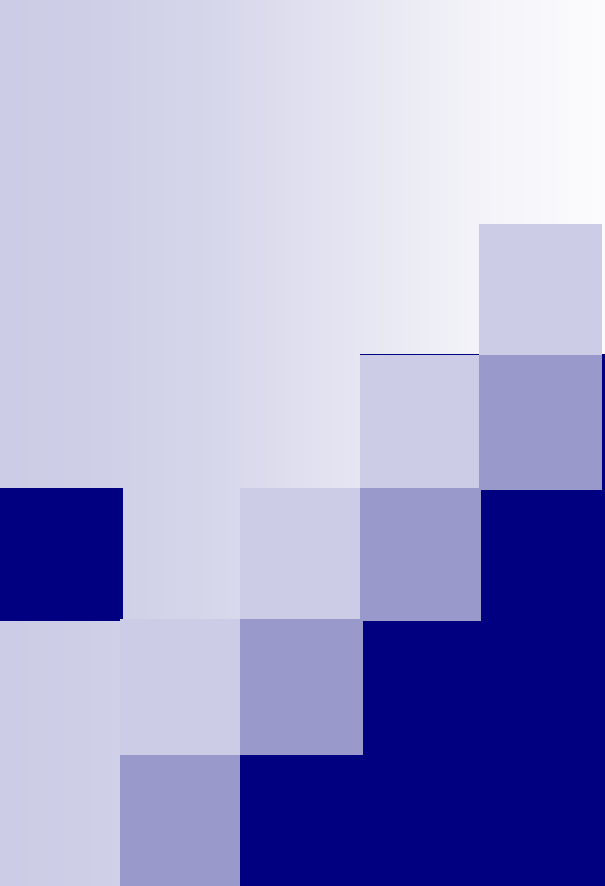




Distributed Systems

Lecture 5

Professor Louise E. Moser

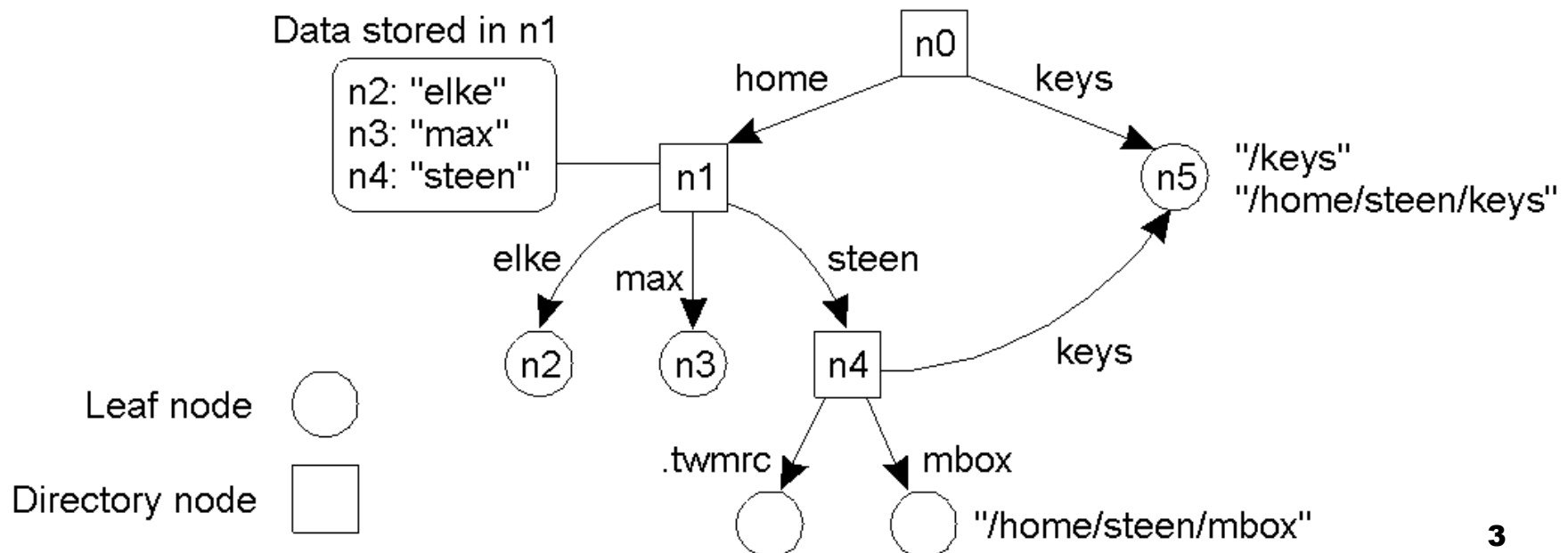
Spring 2013

Names, Addresses, Identifiers

- A **name** is used to refer to an **entity** in a distributed system
 - An **entity** is a computer, printer, user, Web page, service, etc.
- A **pure name** is a name that has no meaning at all
 - A pure name is just a random string, and it can be used only for string comparison
- To access an entity, we need an **access point**
- An **address** is the name of an access point
- A **location-independent name** is a name that does not depend on the address(es) of the access point(s) of the entity
- An **identifier** is a unique, persistent name of an entity
 - An identifier refers to at most one entity, and an entity is referred to by at most one identifier
 - An identifier always refers to the same entity, i.e., it cannot be reused
- An identifier need not be a pure name, i.e., it may have content

Name Spaces

- A **name space** can be represented by a **naming graph** with a single root node
- A **leaf node** in the graph represents a named entity
- A **directory node** in the graph contains a **directory table** that refers to other nodes as *(node identifier, edge label)* pairs



Name Spaces

- Various kinds of **attributes** that describe aspects of an entity can be stored in the node that represents the entity
 - Type of the entity
 - Identifier for the entity
 - Address of the entity's location
 - ...
- Both leaf nodes and directory nodes can have attributes

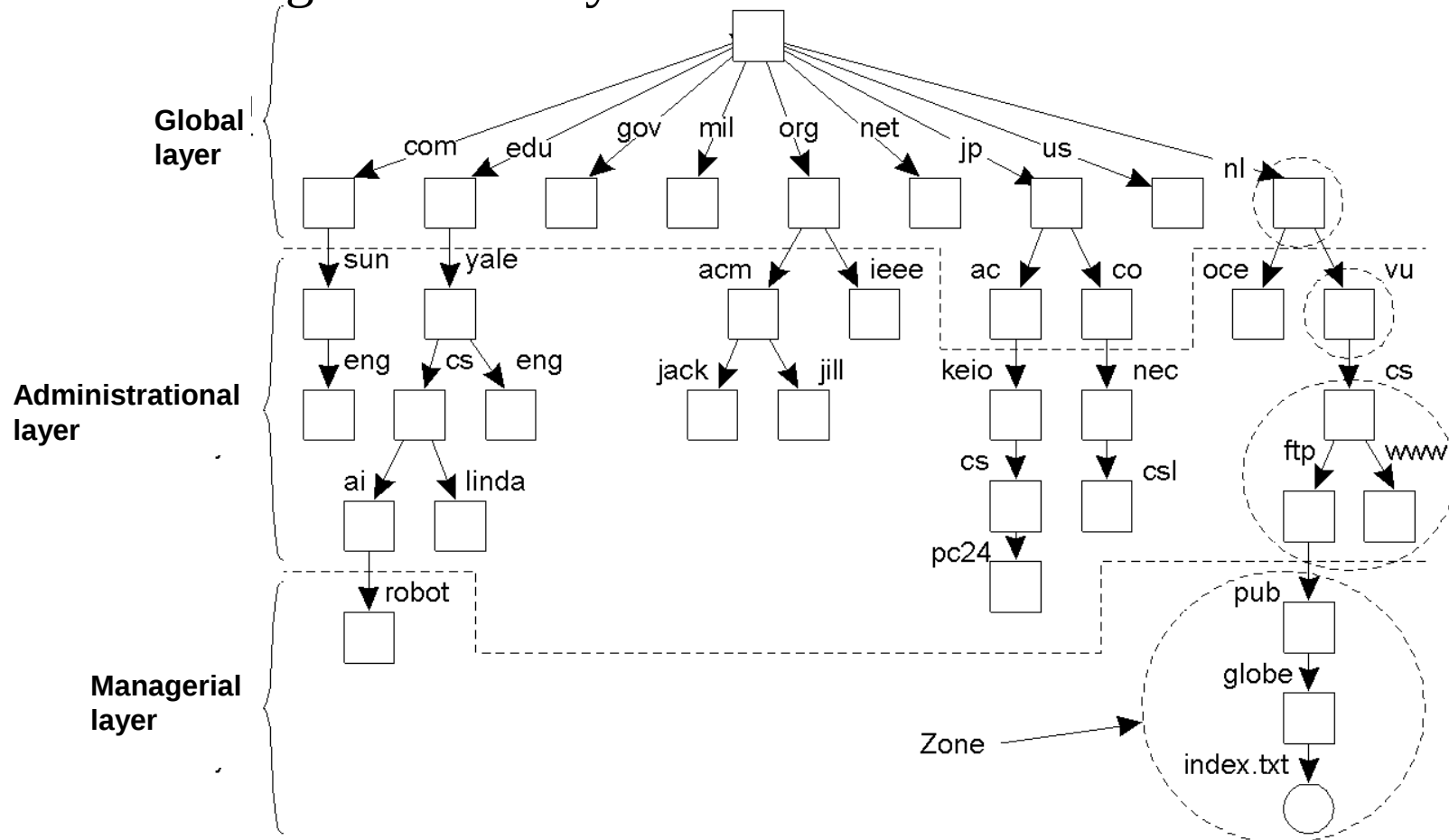


Name Space Distribution

- **Basic idea:** Distribute name space management across multiple machines, by distributing nodes of the naming graph
- Distinguish three layers in a hierarchical naming graph
 - **Global layer:** High-level directory nodes. Main issue is management of these nodes by different administrations
 - **Administrational layer:** Mid-level directory nodes. Main issue is grouping them together so that each group can be assigned to a separate administration
 - **Managerial layer:** Low-level directory nodes within a single administration. Main issue is effectively mapping these nodes to local name servers

Name Space Distribution

- **Example:** The name space of the Domain Name System (DNS), showing the three layers



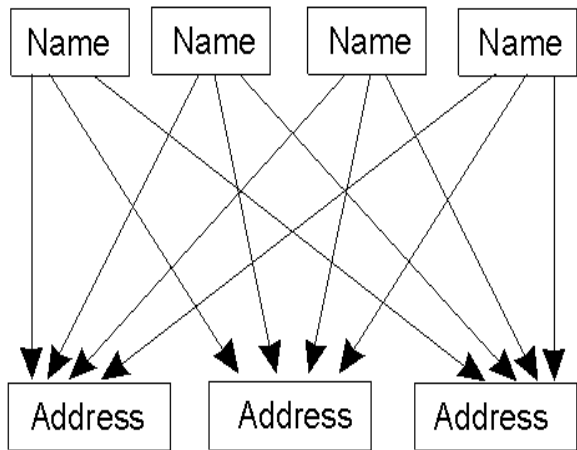
Name Space Distribution

- A comparison of DNS name servers using a large name space, with the three layers

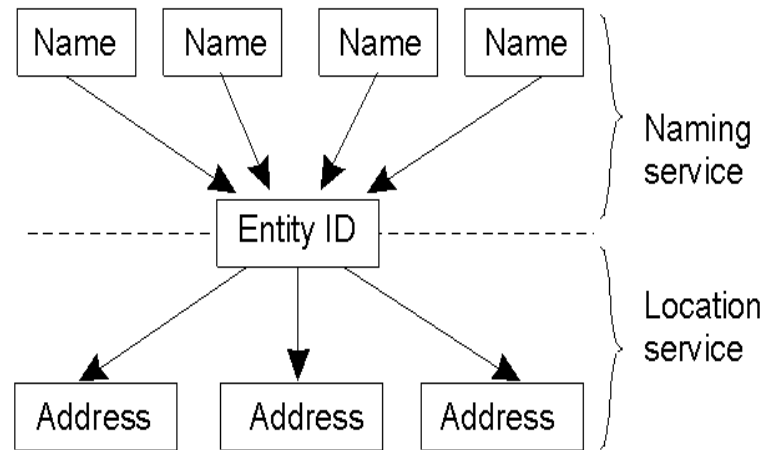
Item	Global	Administrational	Managerial
Geographical scale of network	Worldwide	Organization	Department
Total number of nodes	Few	Many	Vast numbers
Responsiveness to lookups	Seconds	Milliseconds	Immediate
Update propagation	Lazy	Immediate	Immediate
Number of replicas	Many	None or few	None
Is client-side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

Naming vs. Locating Entities

- **Problem:** It is not realistic to assume stable node contents down to the local naming level
- **Solution:** Decouple naming from locating entities, to provide greater flexibility and ease-of-use



(a)



(b)

- (a) Direct mapping between names and addresses
- (b) Indirect mapping using intermediate entity identifiers

Naming vs. Locating Entities

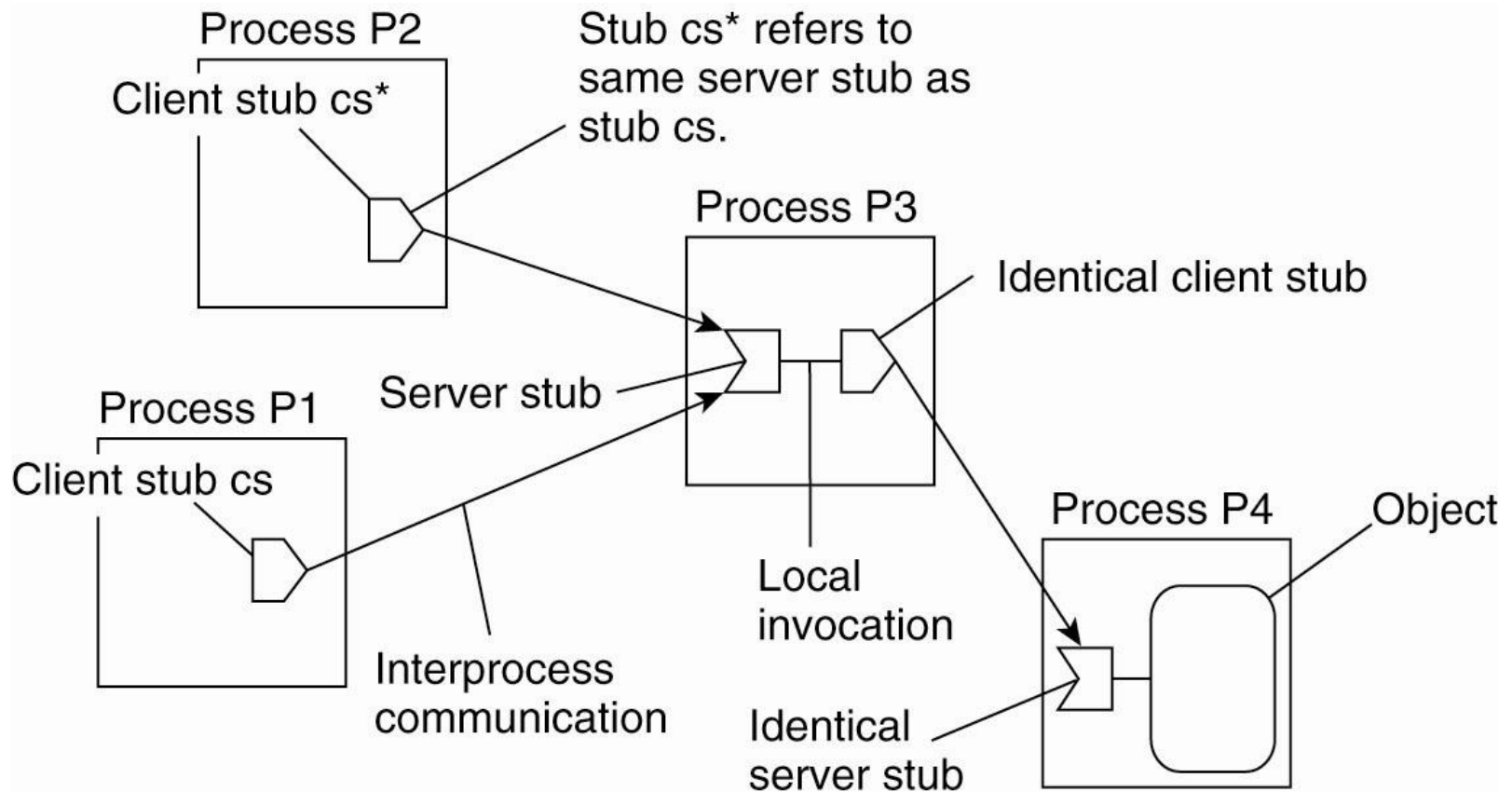
- **Location service:** Aims to provide the addresses of the *current locations* of entities
 - Assumes that entities can be mobile and that their current addresses can change frequently
- **Naming service:** Aims to provide the *contents* of a node in a name space, given the node's name
 - Node contents consist of (attribute, value) pairs
 - Assumes that node contents at the *global* and *administrational* levels are relatively stable, for scalability reasons
- **Note:** If a traditional Naming service is used to *locate* entities, we must also assume that the node contents at the *managerial* level are stable, because we can use only names as identifiers (think of Web pages)

Locating Entities

- **Broadcasting:** Simply broadcast the entity's identifier, requesting it to return its current address
 - Doesn't scale beyond local-area networks (think of ARP)
 - Requires all processes to listen to incoming location requests
- **Forwarding pointers:** When an entity moves, it leaves behind a pointer telling where it went
 - Dereferencing can be made transparent to clients by simply following the chain of pointers
 - Update a client's reference as soon as the entity's current location is found
- Geographical scalability problems
 - Long chains are not fault-tolerant
 - Dereferencing increases network latency
- Essential to have chain reduction mechanisms

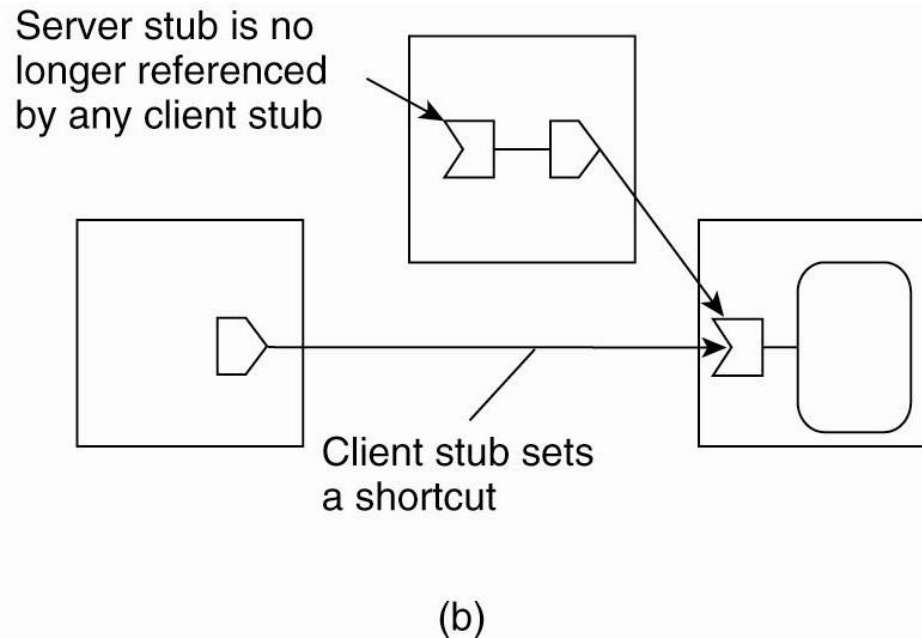
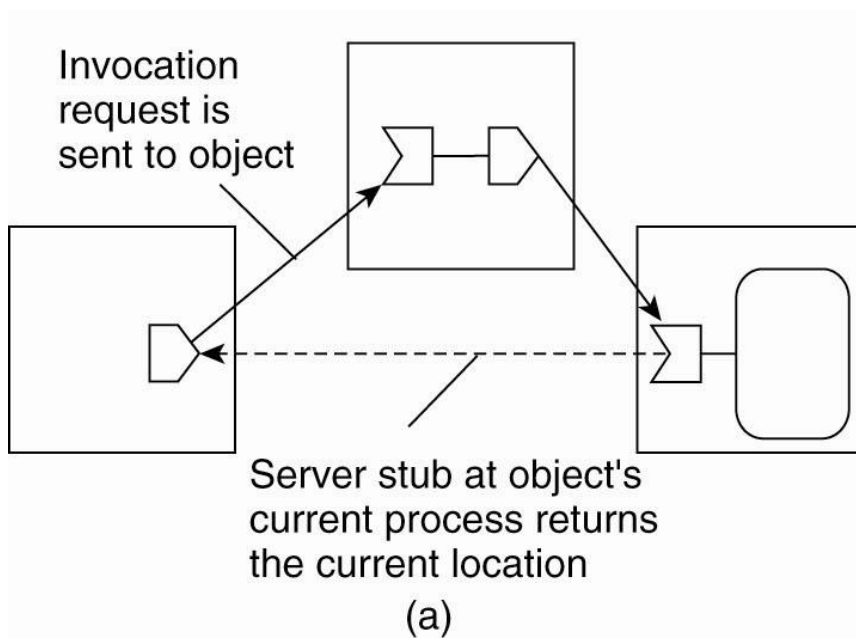
Flat Naming

- **Forwarding pointers** using (*client stub*, *server stub*) pairs



Flat Naming

- **Chain reduction mechanism:** Redirecting a forwarding pointer by storing a shortcut in a client stub

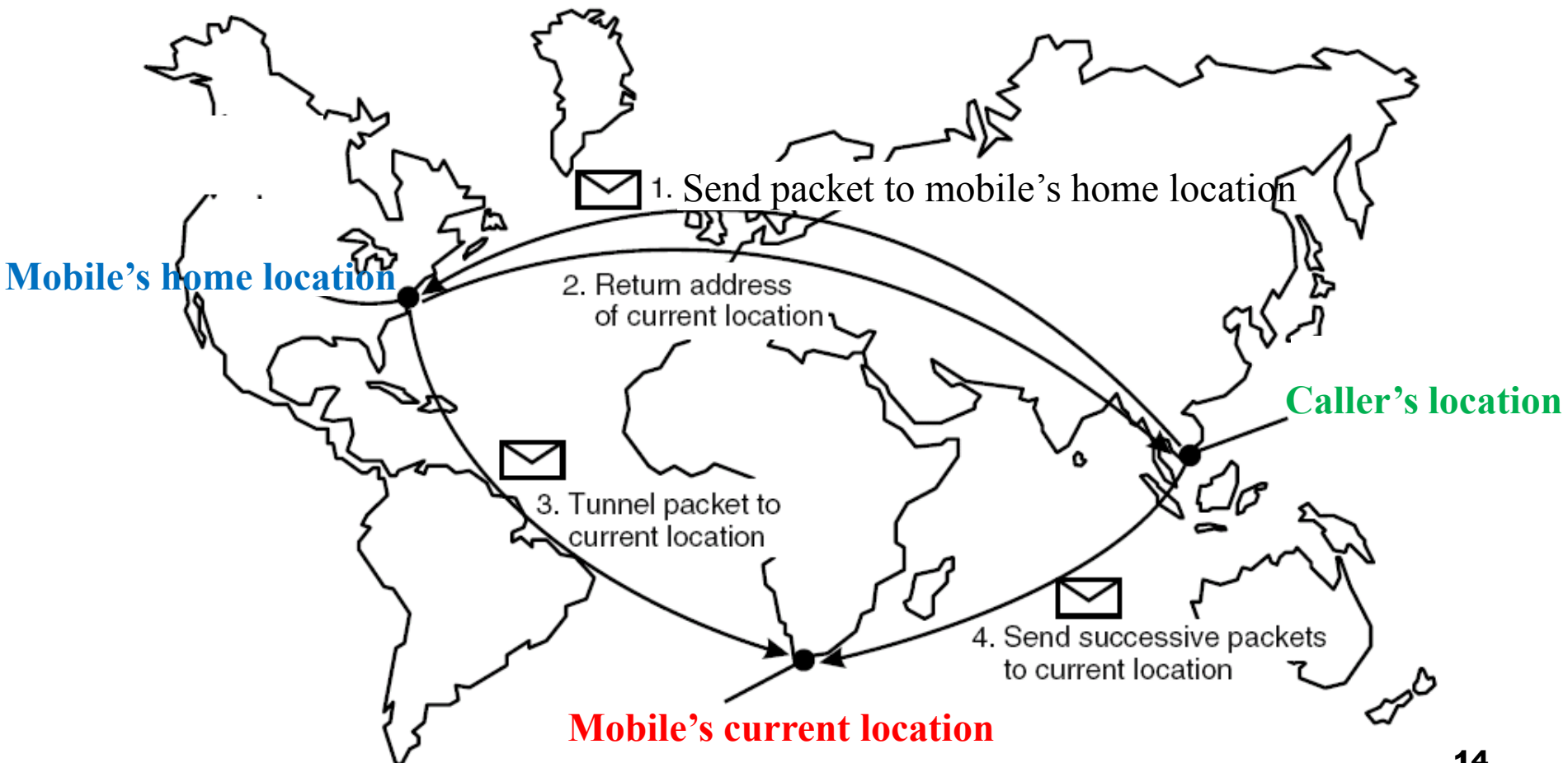


Home-Based Approach

- **Single-tier scheme:** **Home** keeps track of the location of an entity (mobile phone)
 - Home address is registered in a Location/Naming service
 - Home registers the foreign address of the entity
 - Client (caller) first contacts the entity's home, and then continues with the foreign location
- **Two-tier scheme:** Node keeps track of **visiting** entities
 - Check local visitor register first
 - Fall back to home location if local lookup fails
- **Problems with the home-based approach**
 - Home address must be supported as long as the entity lives
 - Home address is fixed, but the entity can permanently move to another location
 - Geographical scalability is poor (entity might be near the client)
- **Question:** How can we solve the “permanent move” problem? ¹³

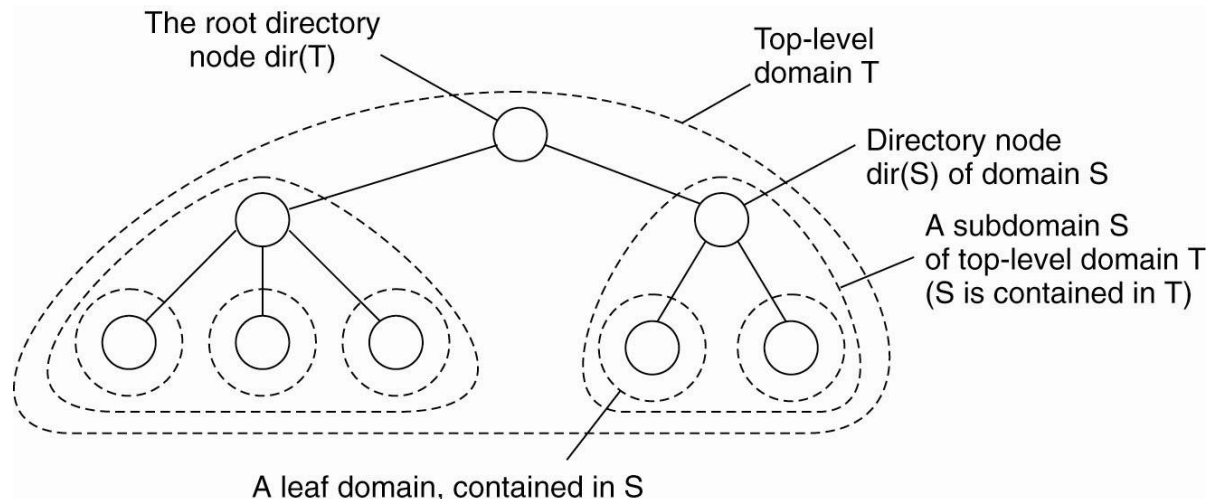
Home-Based Approach

■ Example: Mobile phones



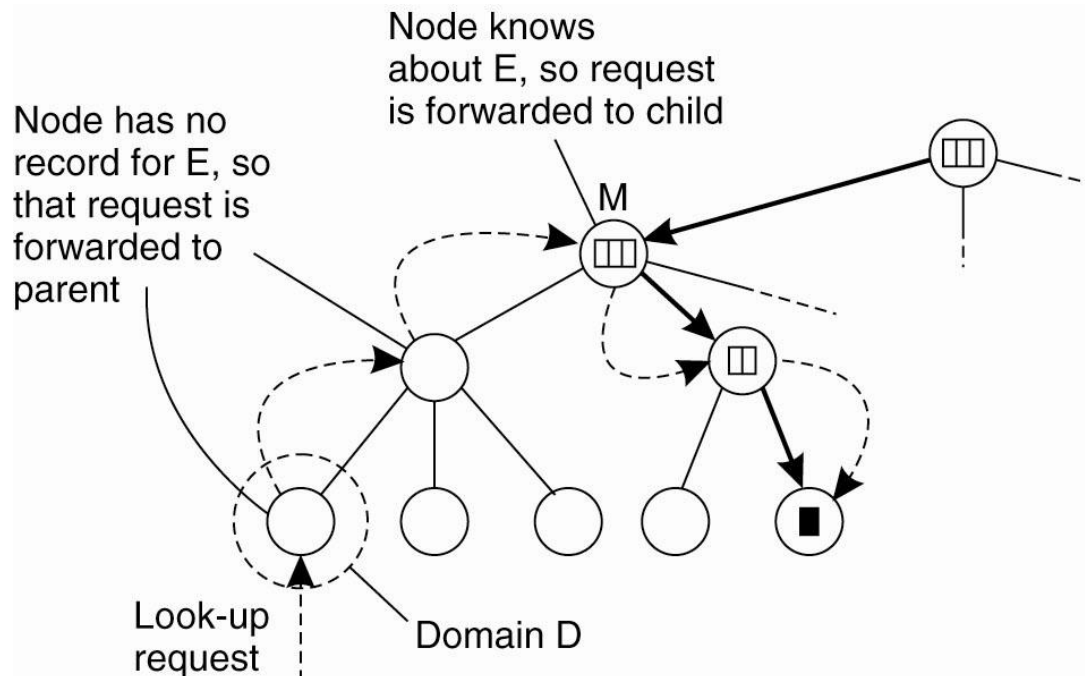
Hierarchical Approach

- **Basic Idea:** A **Location Service** is hierarchically organized into **domains**, each having an associated **directory node**
- Each **entity** in a domain is represented by a **location record** stored in the directory node
 - For a leaf domain D , the location record in its directory node N contains the entity's current address in D
 - For the next higher domain D' , the location record in its directory node N' contains a pointer to N



Hierarchical Approach

- Looking up the location of an entity E
 - Start the lookup at the local node
 - If the node knows about the entity E, go down to the child
 - If the node does not know about the entity, go up to the parent
 - Upward lookup stops at the root node

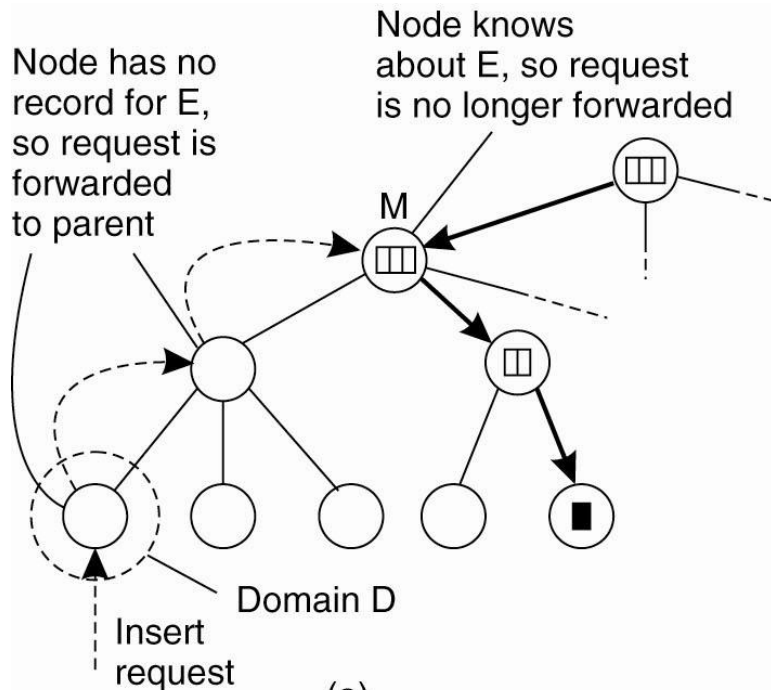


Hierarchical Approach

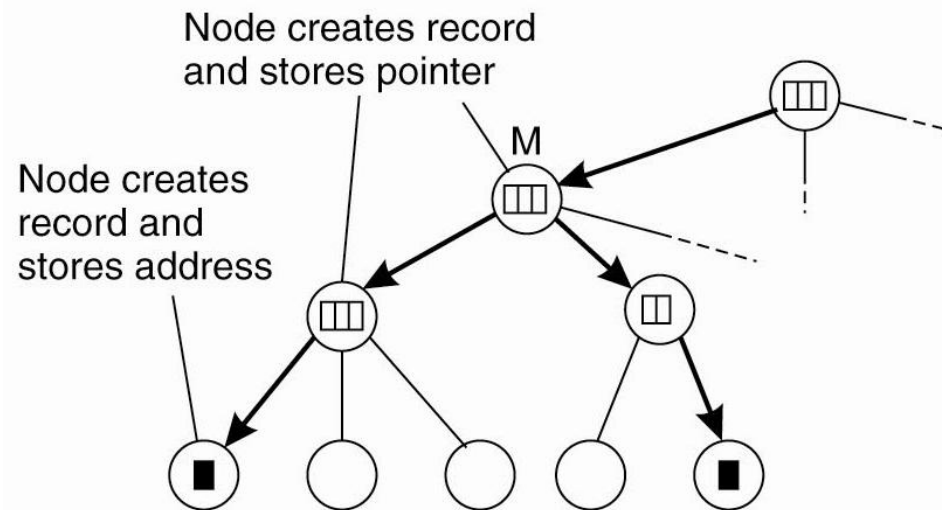
■ Inserting a location record for an entity E at a leaf node

(a) Forward the insertion request up to the parent and so on until a node that knows about entity E is found

(b) Create a chain of pointers down to the leaf node for E



(a)



(b)

Distributed Hash Table

■ DHT for the Chord system

■ Finger table

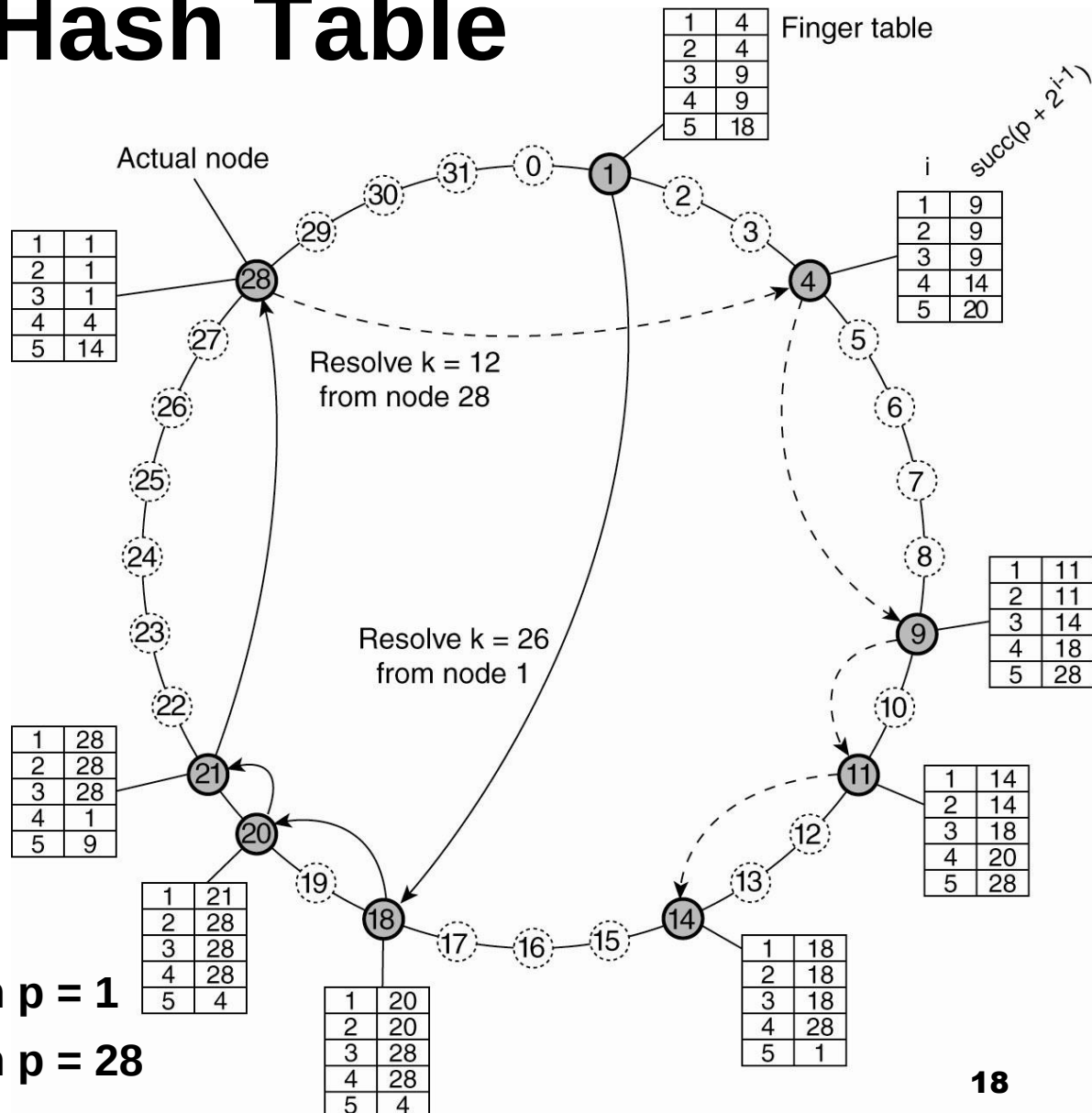
- # entries $\leq \log(\text{size ring})$
- p node, i index
- $n \leq \text{succ}(n)$

■ Key resolution

- If $m < k \leq n$, then node n holds the data with key k

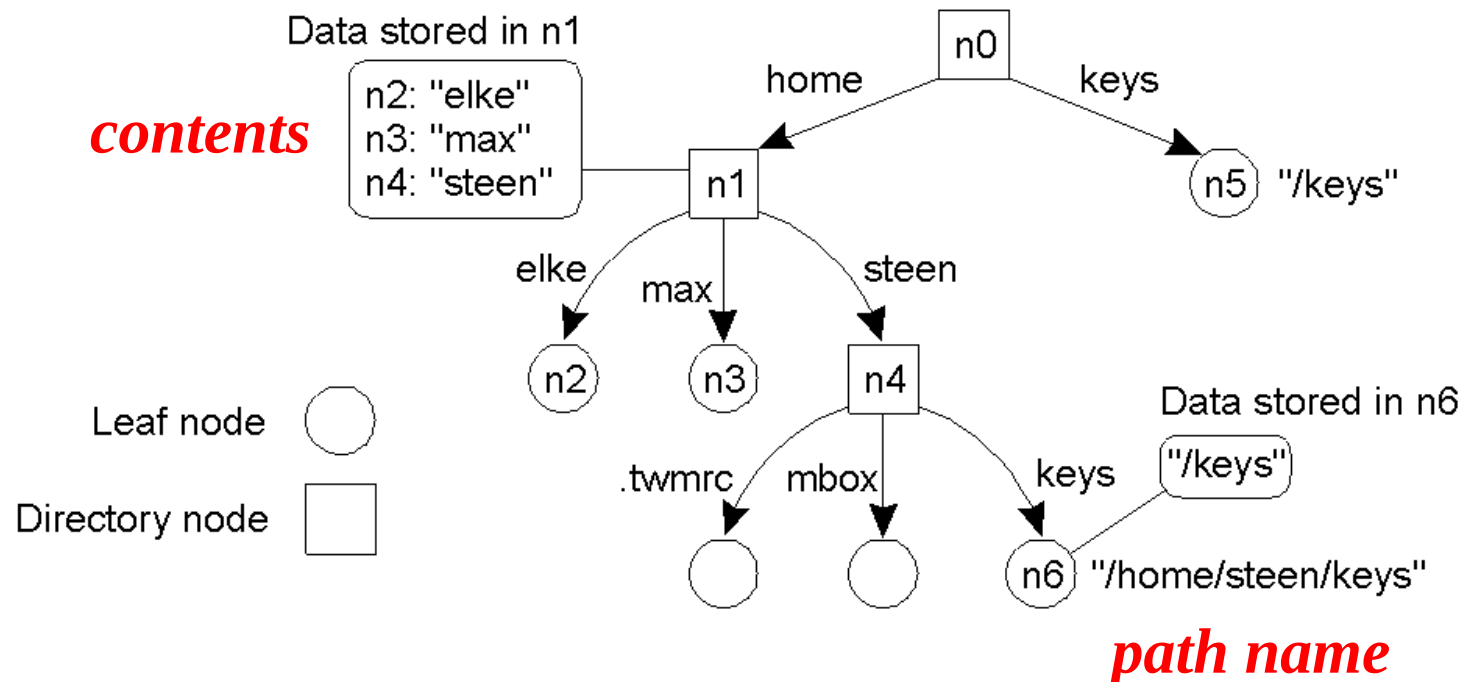
■ Examples:

- Resolving $k = 26$ from $p = 1$
- Resolving $k = 12$ from $p = 28$



Name Linking

- **Hard link:** Name resolution involves following a *path name* in a naming graph from one node to another
- **Soft link:** Name resolution involves reading the *contents* of a node, i.e., the *name* of another node where we want to go



Mounting Name Spaces

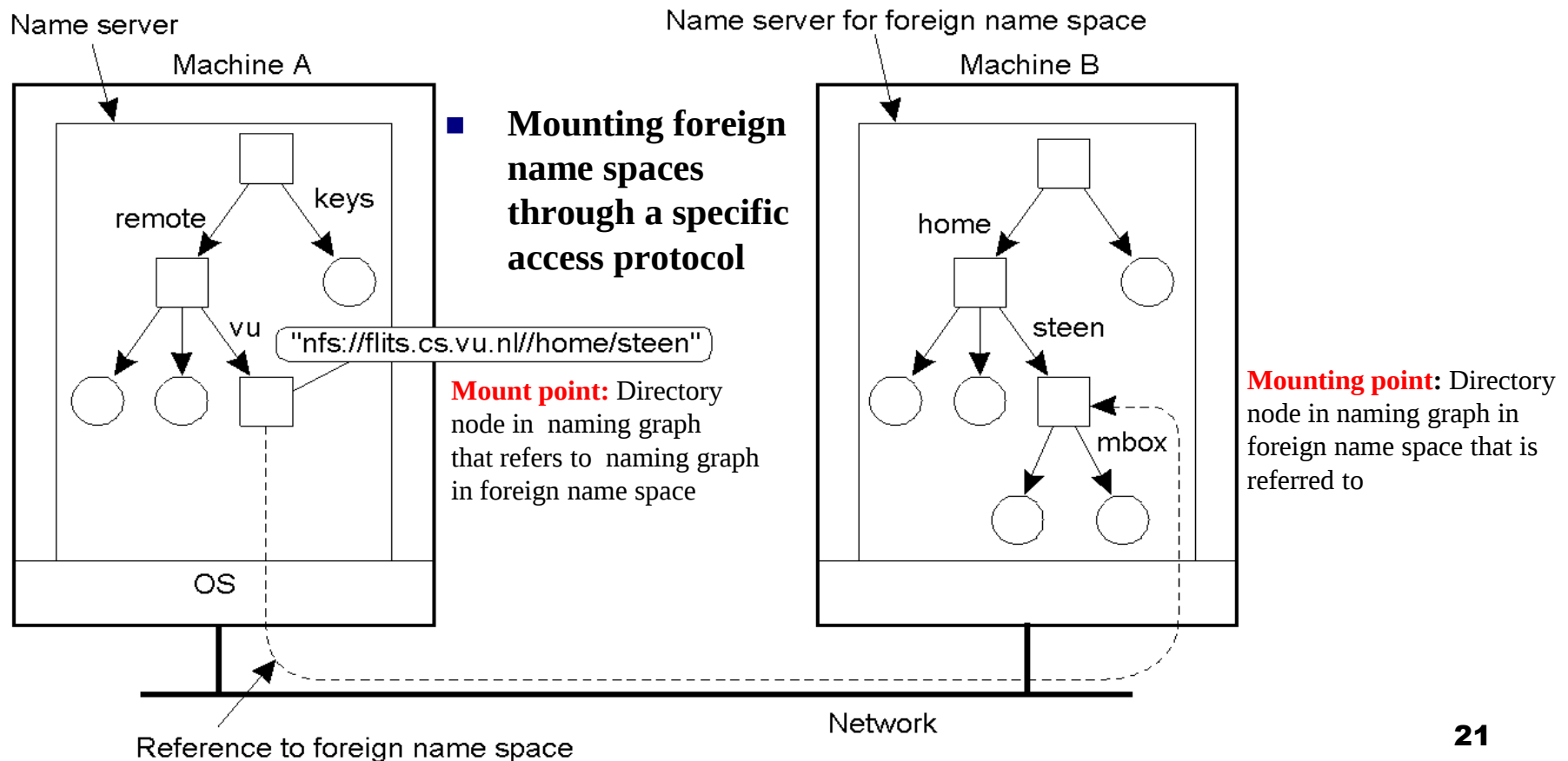
- **Problem:** We have other name spaces that we wish to access from a given name space
- **Solution:** Introduce a naming scheme by which path names of different name spaces are concatenated (e.g., URLs)

ftp://ftp.cs.vu.nl/pub/steen/

ftp	Name of access protocol used to talk with server
://	Name space delimiter
ftp.cs.vu.nl	Name of a node representing an FTP server and containing the IP address of that server
/	Name space delimiter
pub/steen/	Name of a context (subdirectory) node in the name space rooted at the context node mapped to the FTP server

Mounting Name Spaces

- Introduce a node that contains the name of a node in a “foreign” name space, along with how to select the initial context in that foreign name space

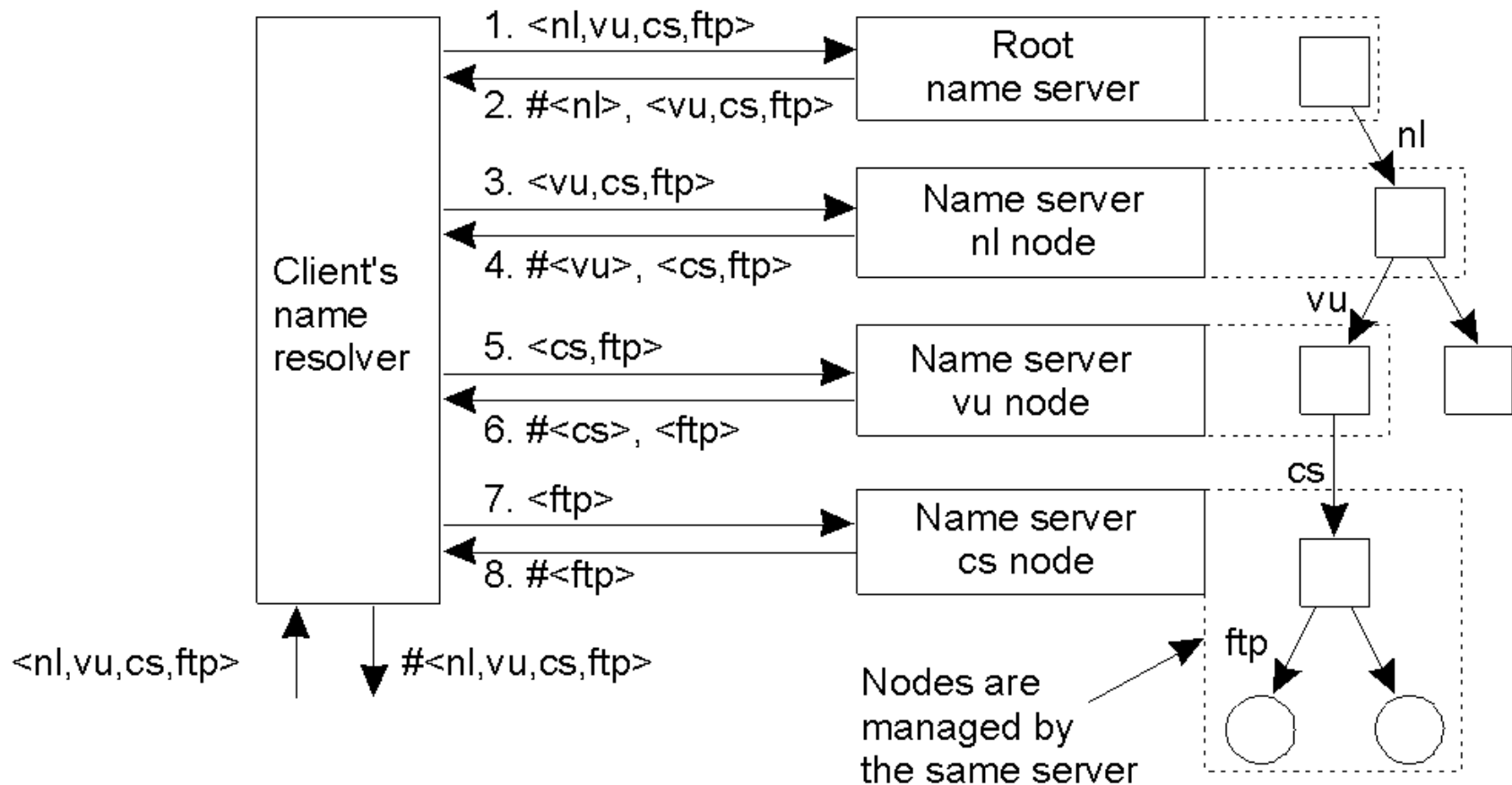


Name Resolution

- **Problem:** To **resolve** a name (i.e., find an address corresponding to the name), we need a directory node that maintains a name-to-address binding. How do we find the directory node?
- **Closure mechanism:** Selects the implicit context from which to start the name resolution:
 - **www.cs.vu.nl:** Start at a DNS name server
 - **/home/steen/mbox:** Start at the local NFS file server
 - **805-448-8249:** Dial a phone number
 - **130.37.24.8:** Use the IP address of VU's Web server
- **Note:** A closure mechanism might also determine how name resolution should proceed

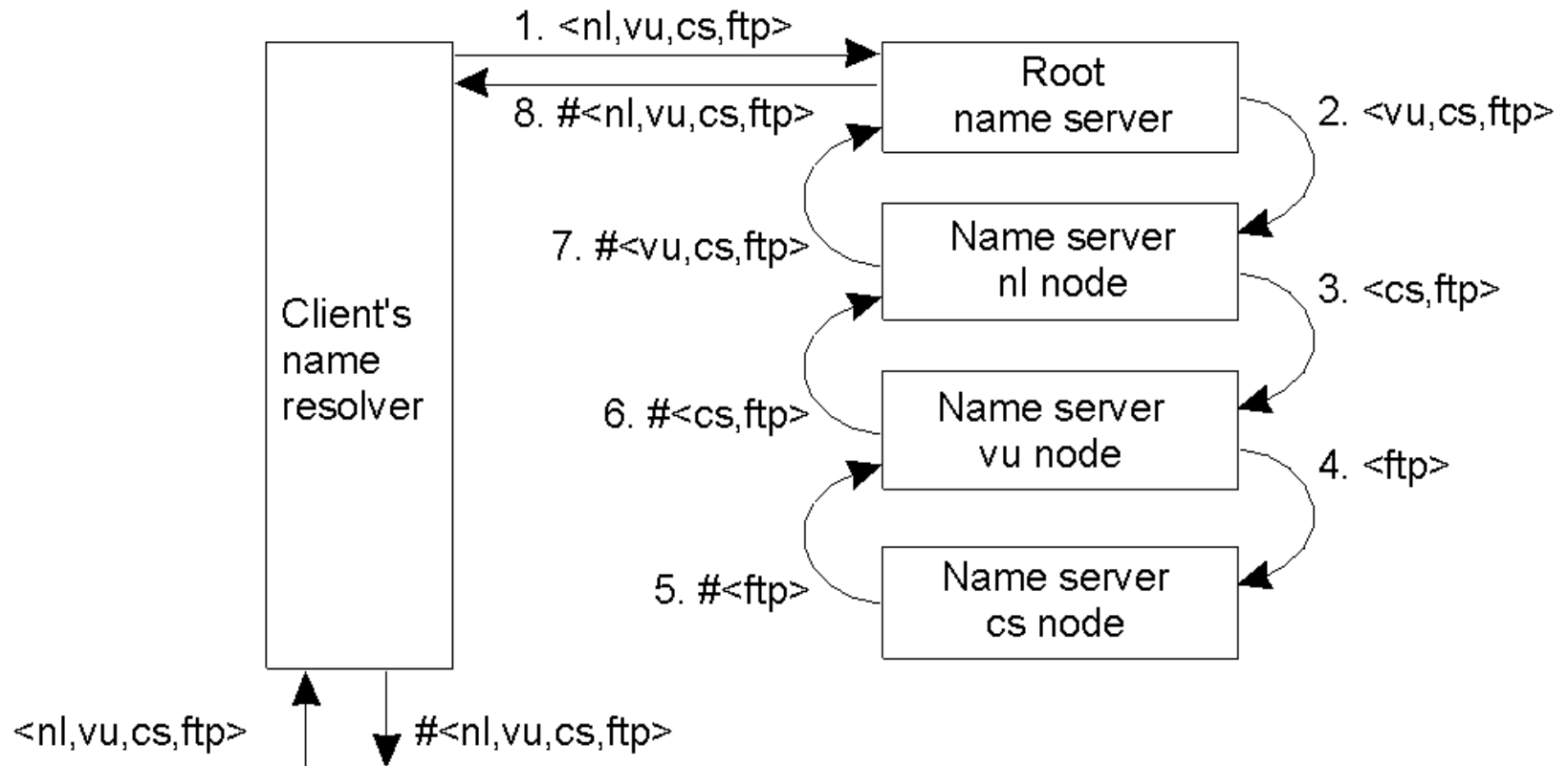
Name Resolution

■ Iterative name resolution



Name Resolution

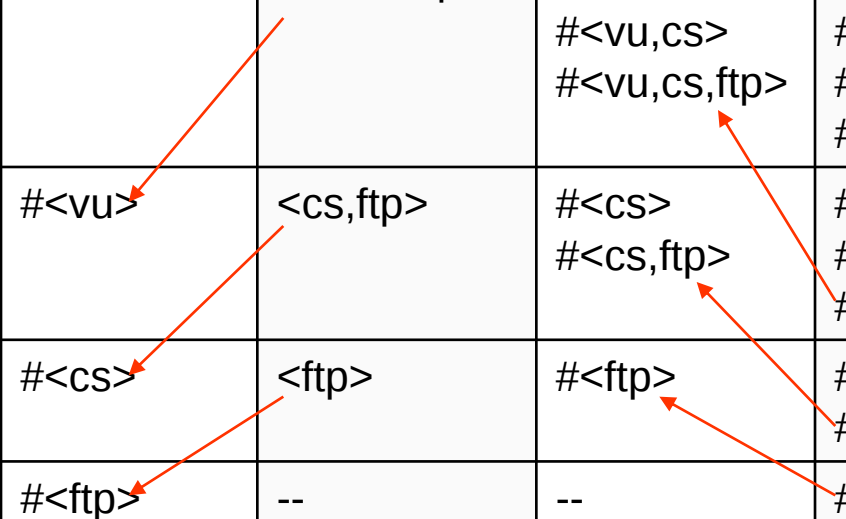
■ Recursive name resolution



Name Resolution

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

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

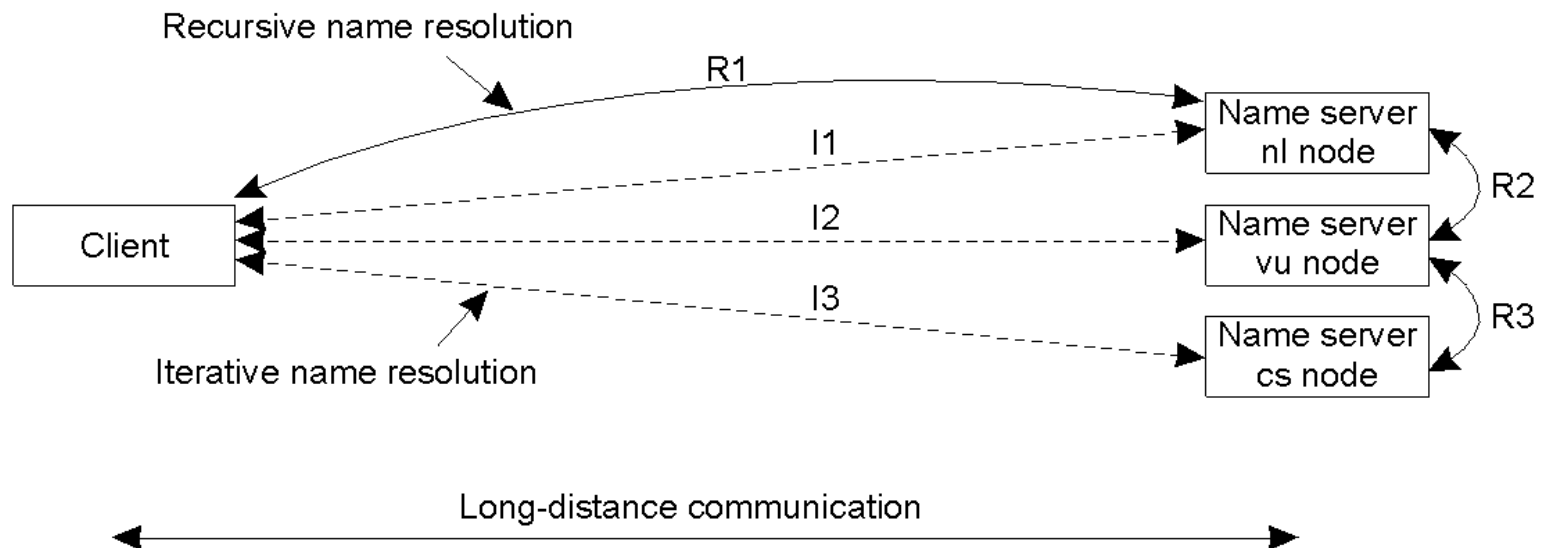


The diagram illustrates the recursive name resolution process using red arrows. The arrows show the flow of lookups and responses between the rows of the table:

- A red arrow points from the 'Looks up' column of the 'root' row to the 'Looks up' column of the 'nl' row.
- A red arrow points from the 'Looks up' column of the 'nl' row to the 'Looks up' column of the 'vu' row.
- A red arrow points from the 'Looks up' column of the 'vu' row to the 'Looks up' column of the 'cs' row.
- A red arrow points from the 'Returns to requester' column of the 'cs' row to the 'Receives and caches' column of the 'vu' row.
- A red arrow points from the 'Returns to requester' column of the 'vu' row to the 'Receives and caches' column of the 'nl' row.
- A red arrow points from the 'Returns to requester' column of the 'nl' row to the 'Receives and caches' column of the 'root' row.

Name Resolution

- Iterative vs. Recursive name resolution and their communication costs



Scalability Issues

- **Size scalability:** We need to ensure that name servers can handle a large number of requests per time unit; overloading of high-level name servers is the problem
 - Only solution is to partition a node into a number of sub-nodes and evenly assign entities to sub-nodes
 - Naive partitioning can introduce a node management problem, because a sub-node might need to know how its parent and children are partitioned
- **Geographical scalability:** We need to ensure that lookup operations generally proceed in the direction in which the address will be found
 - If an entity resides in California, we should not let a root sub-node in Europe store that entity's location record
 - Unfortunately, sub-node placement is not that easy, and only a few possible solutions have been proposed

LDAP Directory Service

- The **Lightweight Directory Access Protocol (LDAP)** is derived from the OSI X.500 Directory Service
 - **LDAP is used, e.g., in Microsoft's Active Directory Service**
- The **LDAP Directory Service** is a hierarchical directory service that consists of records, referred to as **Directory Entries**
- An LDAP directory entry consists of (*attribute, value*) pairs, referred to as **Relative Distinguished Names (RDNs)**
- The collection of all directory entries is referred to as the **Directory Information Base (DIB)**
- The use of globally unique names by listing RDNs in sequence yields a hierarchy of the collection of directory entries, referred to as a **Directory Information Tree (DIT)**

LDAP Directory Service

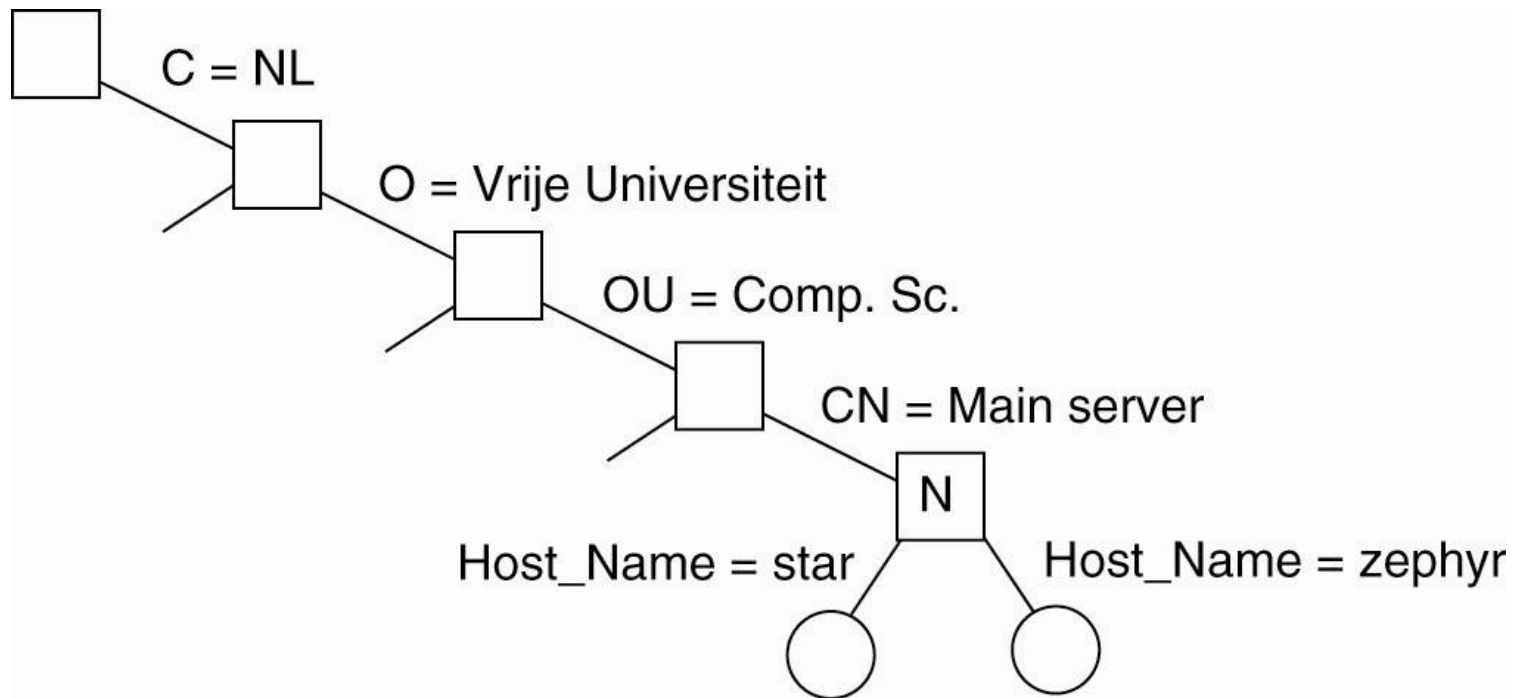
- Two example LDAP directory entries, showing the RDNs, i.e., (attribute, value) pairs and the LDAP naming conventions for attributes

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

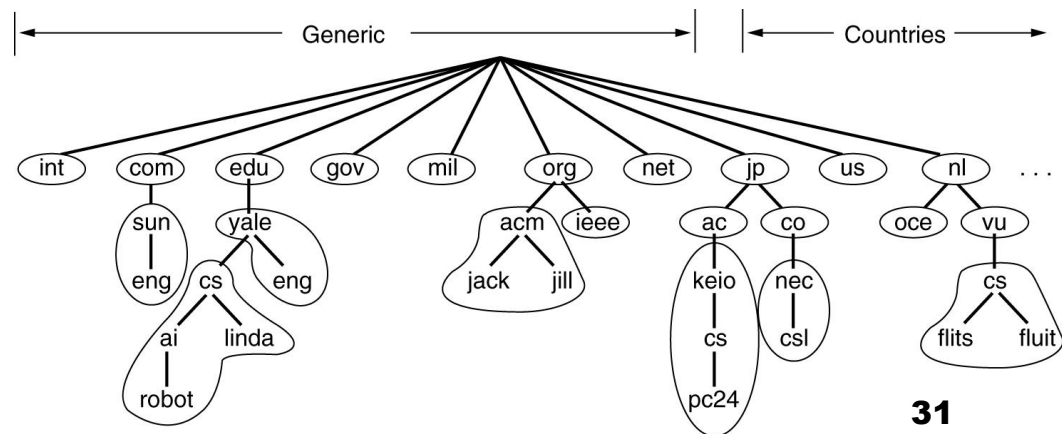
LDAP Directory Service

- Part of the LDAP Directory Information Tree



Domain Name System

- The **Domain Name System (DNS)** is the hierarchical Location/Naming service that maintains the correspondence between machine names and IP addresses in the Internet
- DNS is organized as a rooted tree
- A subtree is referred to as a **domain**
- A path name to the root of the subtree is referred to as a **domain name**
- The content of a node is a collection of **resource records**



Domain Name System

- **DNS Name Space:** Types of DNS resource records that form the contents of DNS nodes

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

Domain Name System

■ DNS database excerpt for the zone *cs.vu.nl*

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"

Domain Name System

- Part of a DNS description for the *vu.nl* domain, which contains the *cs.vu.nl* domain

