Distributed Systems Lecture 5

Professor Louise E. Moser

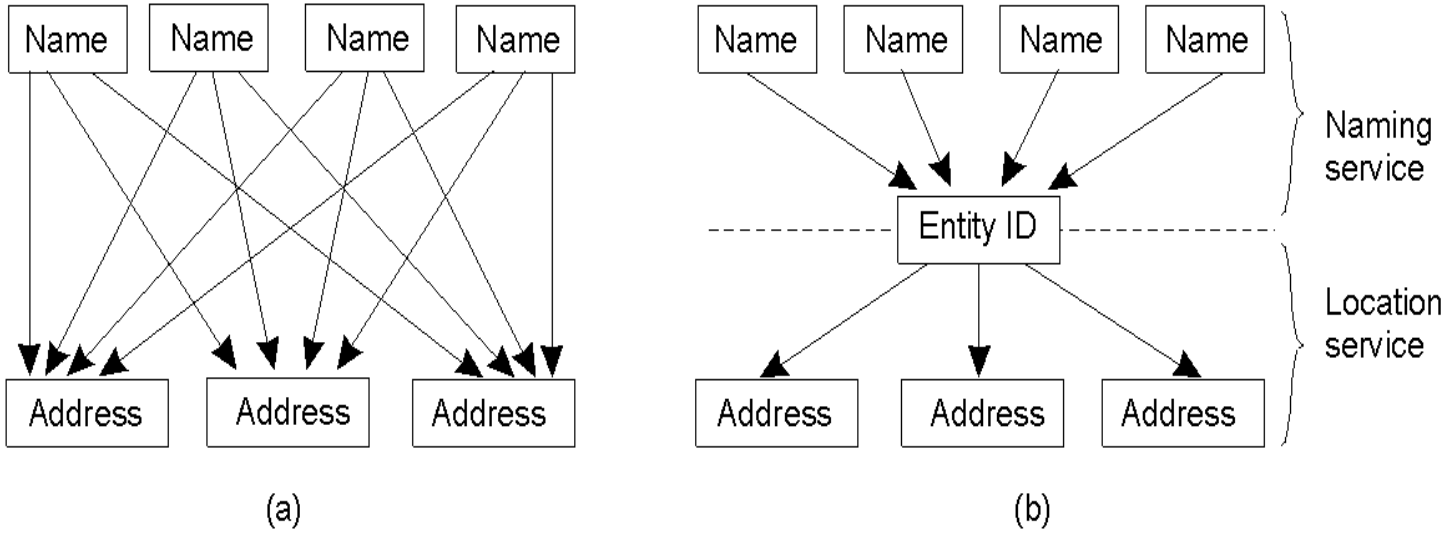
Spring 2014

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; 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 of the access point of the entity
- An **identifier** is a unique, persistent name of an entity
  - **An identifier refers to at most one entity; an entity is referred to by at most one identifier**
  - **An identifier always refers to the same entity; it cannot be reused**
- An identifier need not be a pure name; it may have content

Naming vs. Locating Entities

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



(a) Direct mapping between names and addresses

(b) Indirect mapping using intermediate entity identifiers

Naming vs. Locating Entities

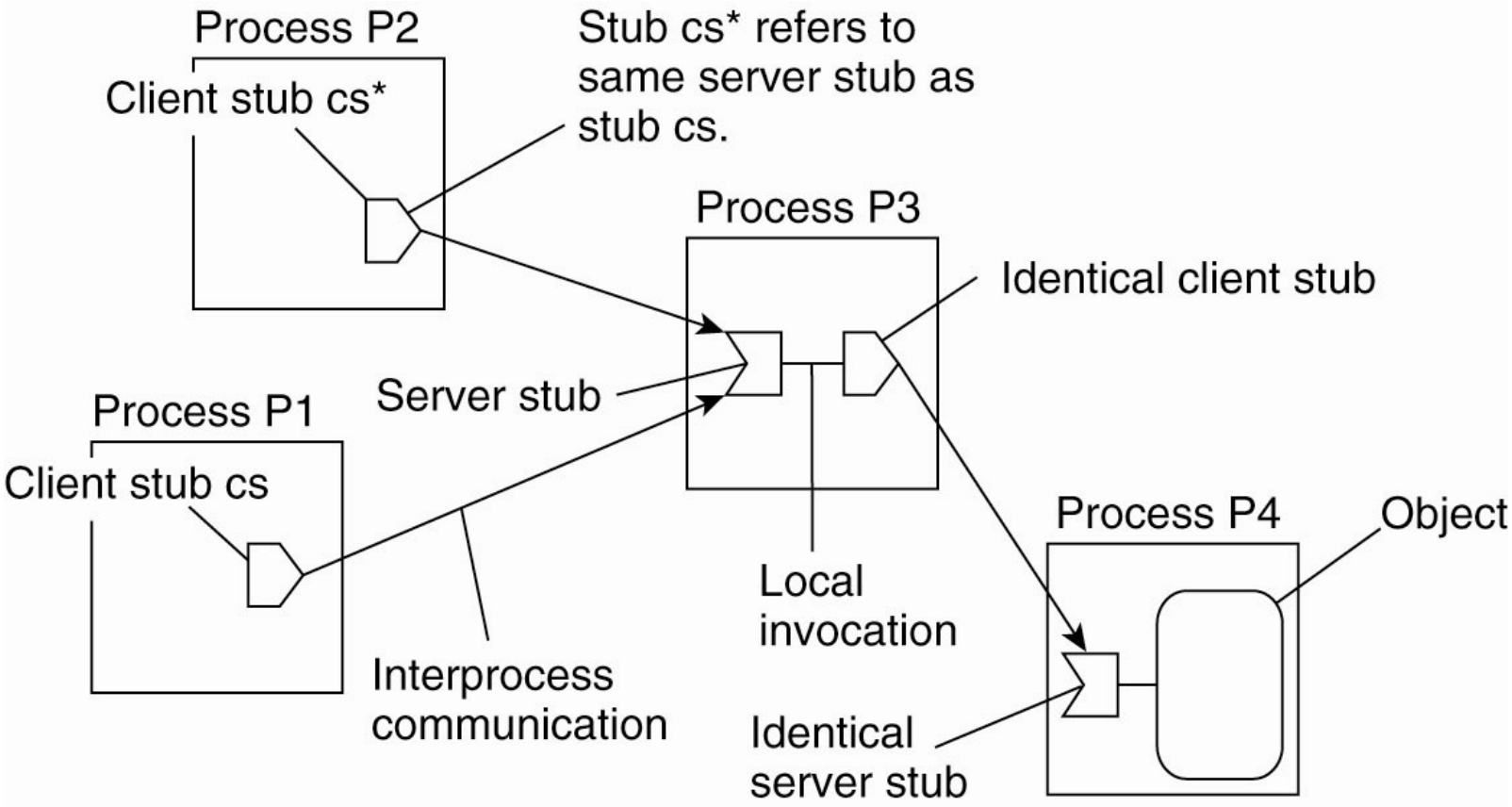
- **Naming service:** Provides the ***contents*** of a node in a name space, given the node's name
  - **Node contents are (attribute, value) pairs**
  - **Assumes that node contents at the *global* and *administrational* levels (see Slide 24) are relatively stable, for scalability reasons**
- **Location service:** Provides the addresses of the ***current locations*** of entities
  - **Assumes that entities can be mobile and that their locations can change frequently**
- **Note:** If a Naming service is also used to ***locate*** entities, also assume that the node contents at the ***managerial*** level (see Slide 24) are stable, because names are used as identifiers

Locating Entities

- **Broadcasting:** Simply broadcast the entity's identifier, requesting the entity to return its current address
  - **Doesn't scale beyond local-area networks (think of ARP)**
  - **Requires all processes to listen to incoming address 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
  - **Dereferencing increases network latency**
  - **Long chains are not fault-tolerant**
- Therefore, it is necessary to have chain reduction mechanisms

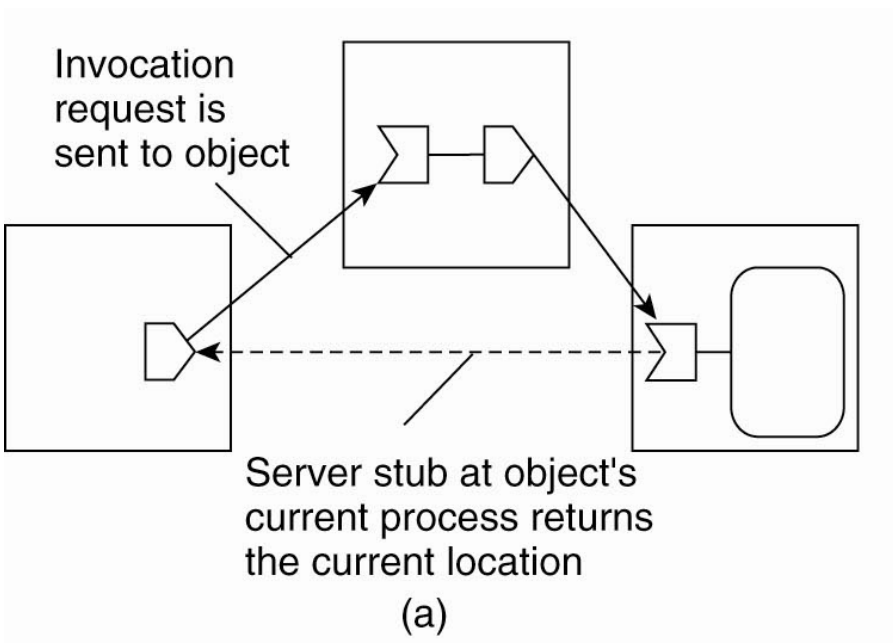
Flat Naming

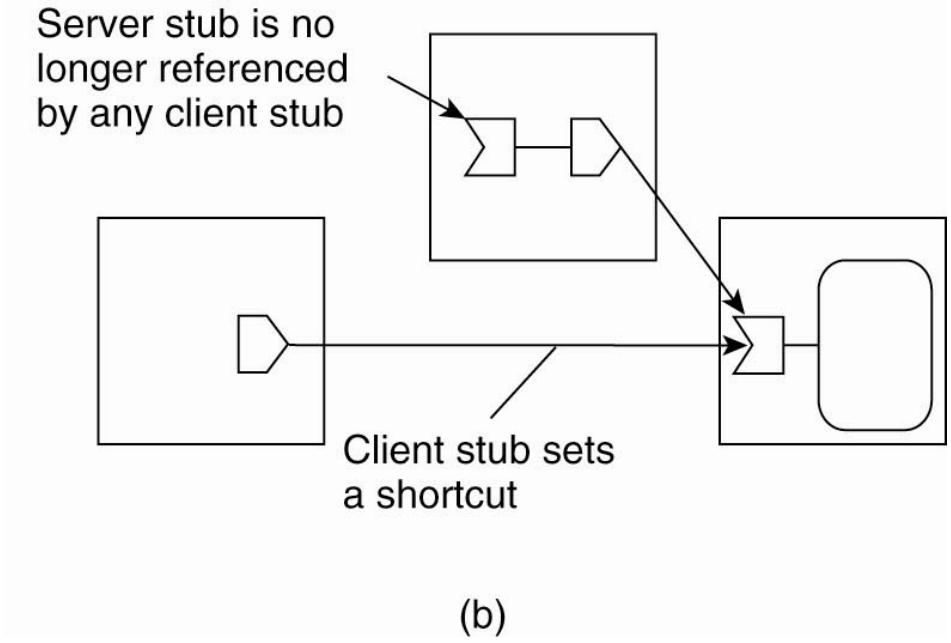
- **Forwarding pointers** using *(client stub, server stub)* pairs



Flat Naming

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



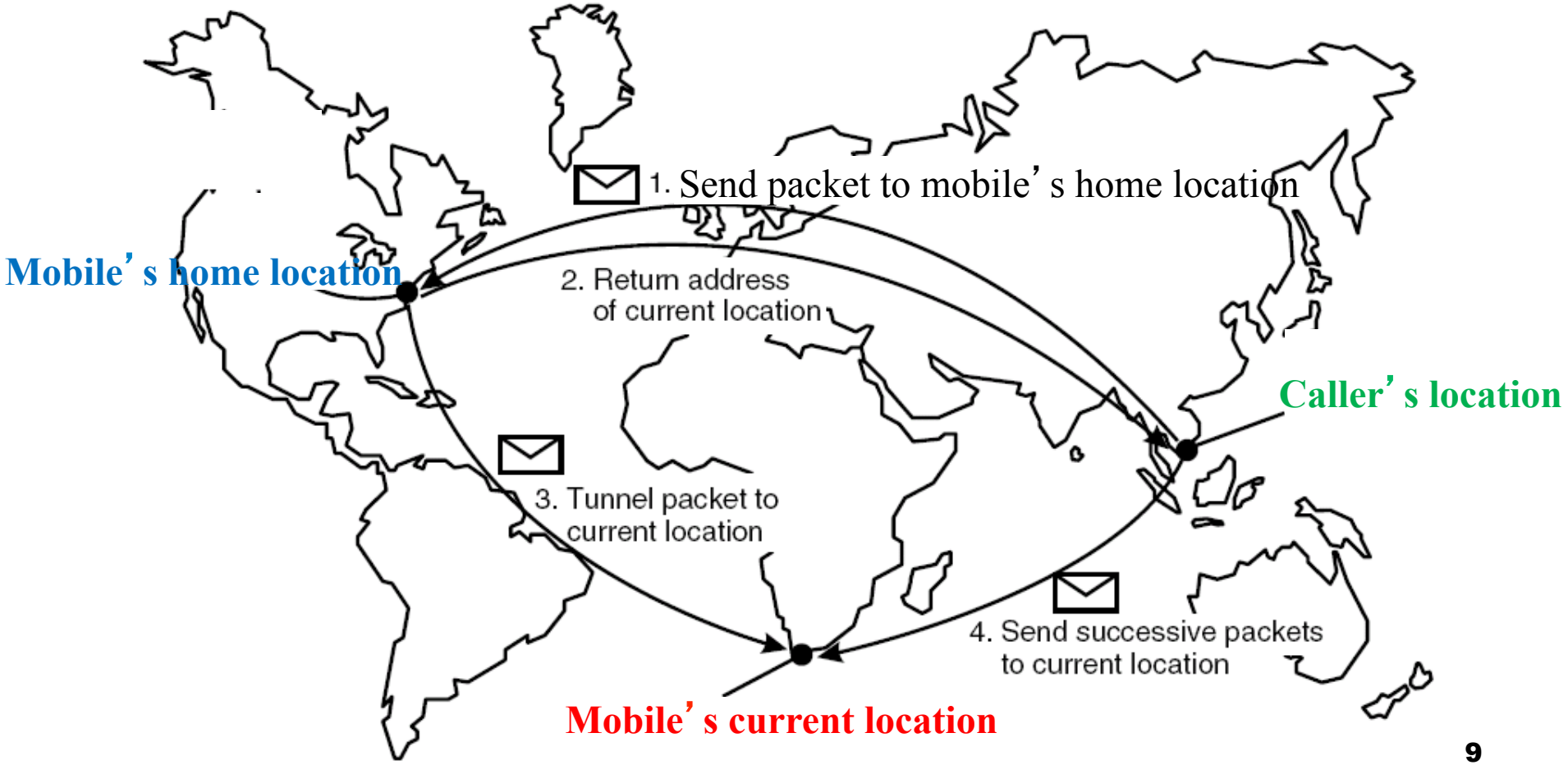


Home-Based Approach

- **Single-tier scheme:** The **home** of an entity keeps track of the location of the entity (e.g., mobile phone)
  - **Home address is registered with a Location Service**
  - **Home maintains the foreign address of the entity**
  - **Caller first contacts the entity's home, and then continues with the foreign address**
- **Two-tier scheme:** Each node keeps track of **visiting** entities
  - **Check local visitor registry first**
  - **Fall back to home 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 caller)**
- **Question:** How can we solve the "permanent move" problem?

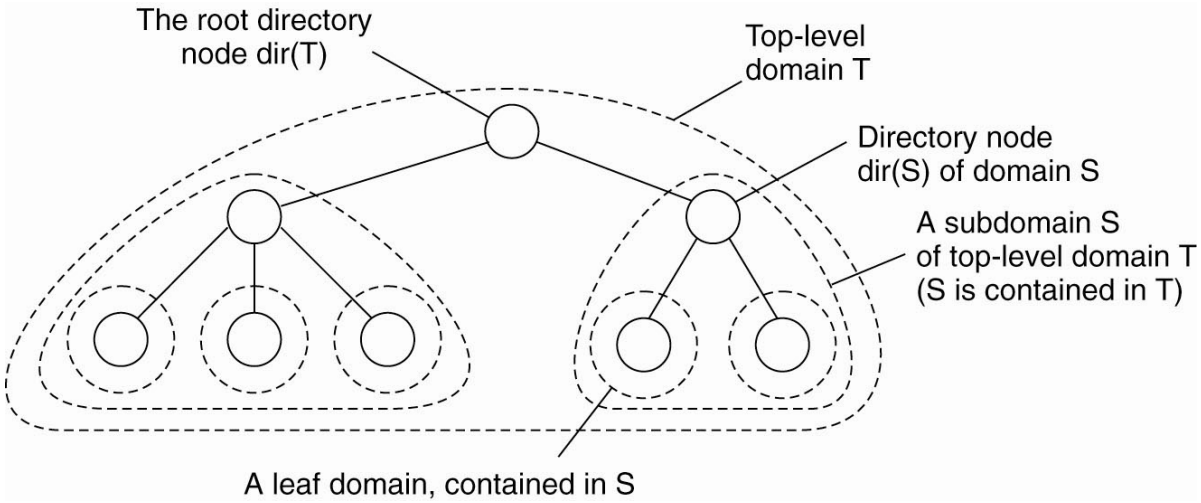
Home-Based Approach

- **Example: Mobile phones**



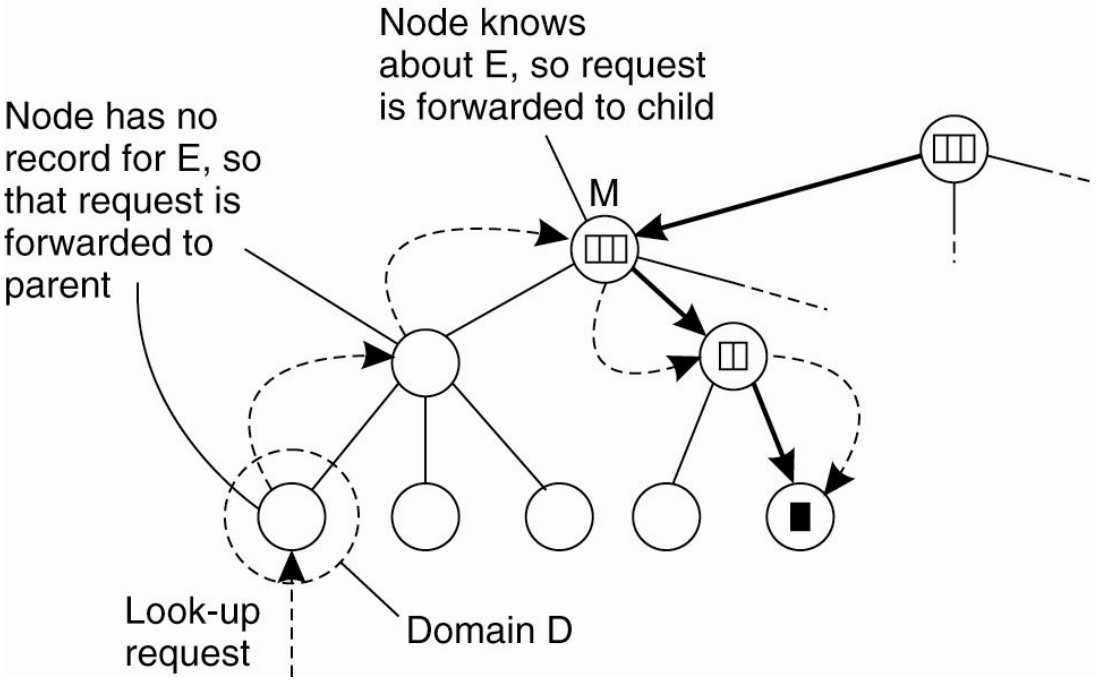
Hierarchical Approach

- **Basic Idea: 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 the directory node N contains the entity's current address in the domain D**
  - **For the next higher domain D', the location record in the directory node N' contains a pointer to the node 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 E, go up to the parent**
    - **Upward lookup stops at the root node**

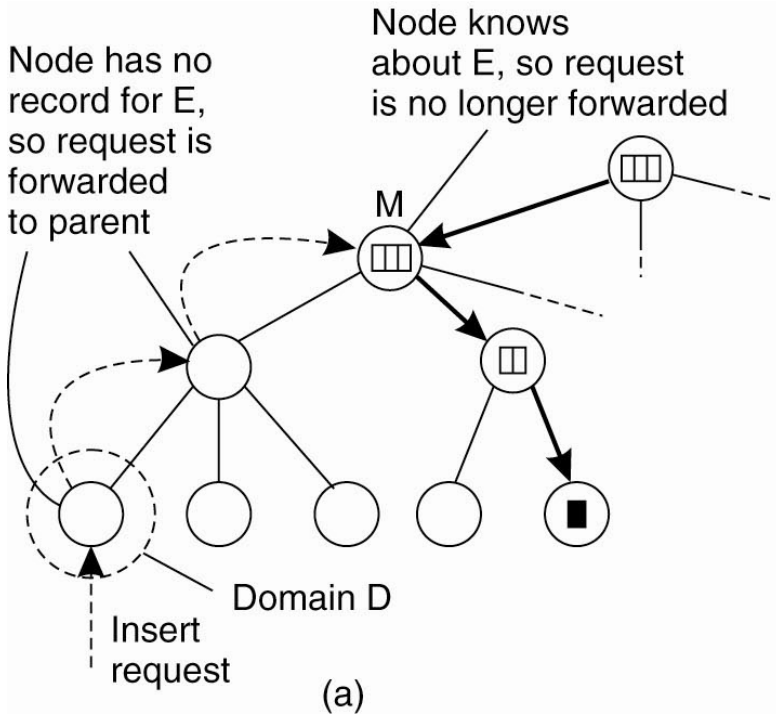


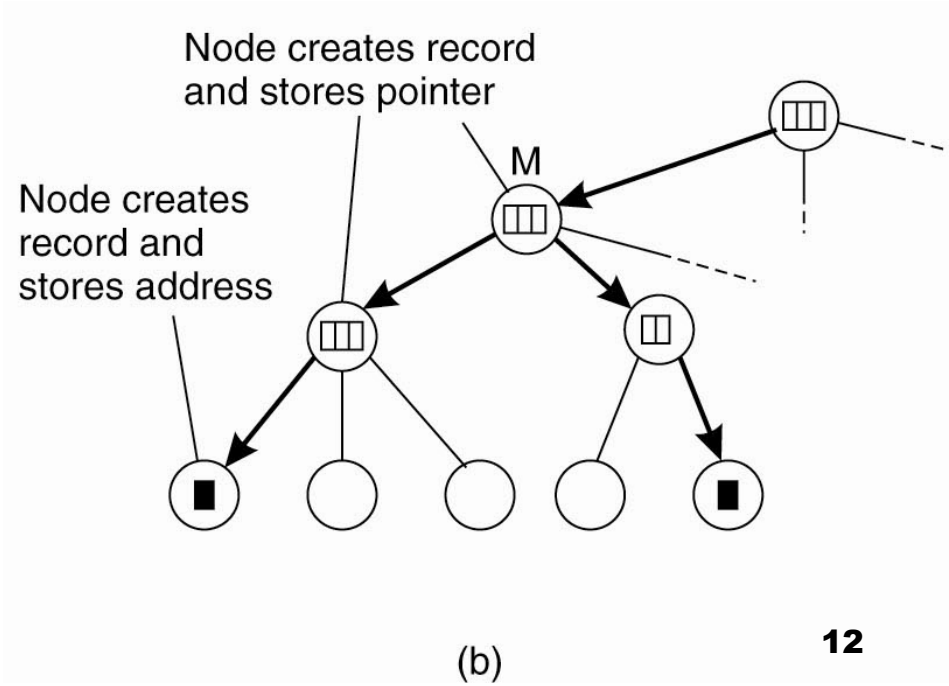
Hierarchical Approach

- 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 the entity E is found

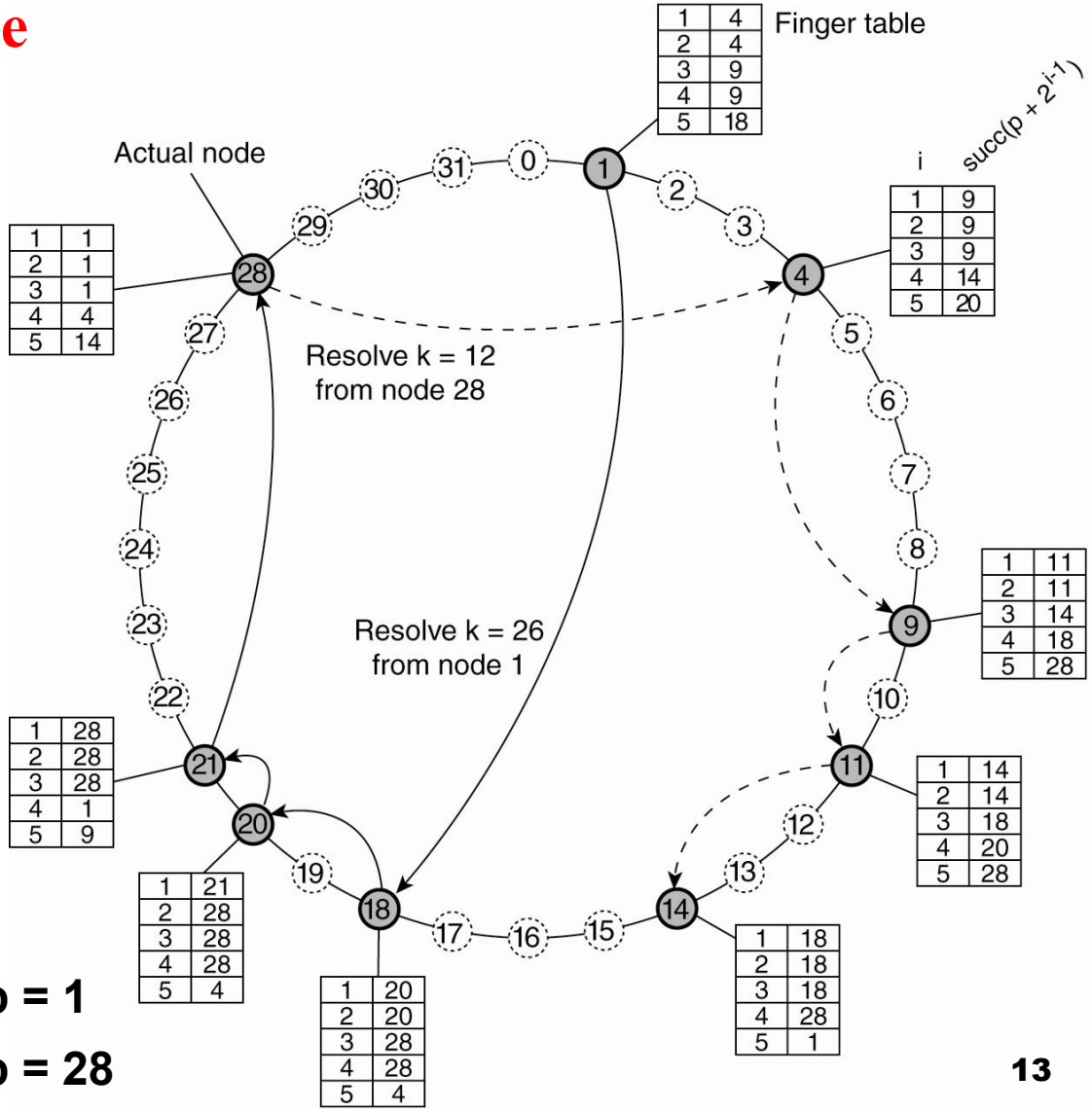
(b) Create a chain of pointers down to the leaf node for the entity E





Example: Hierarchical Approach

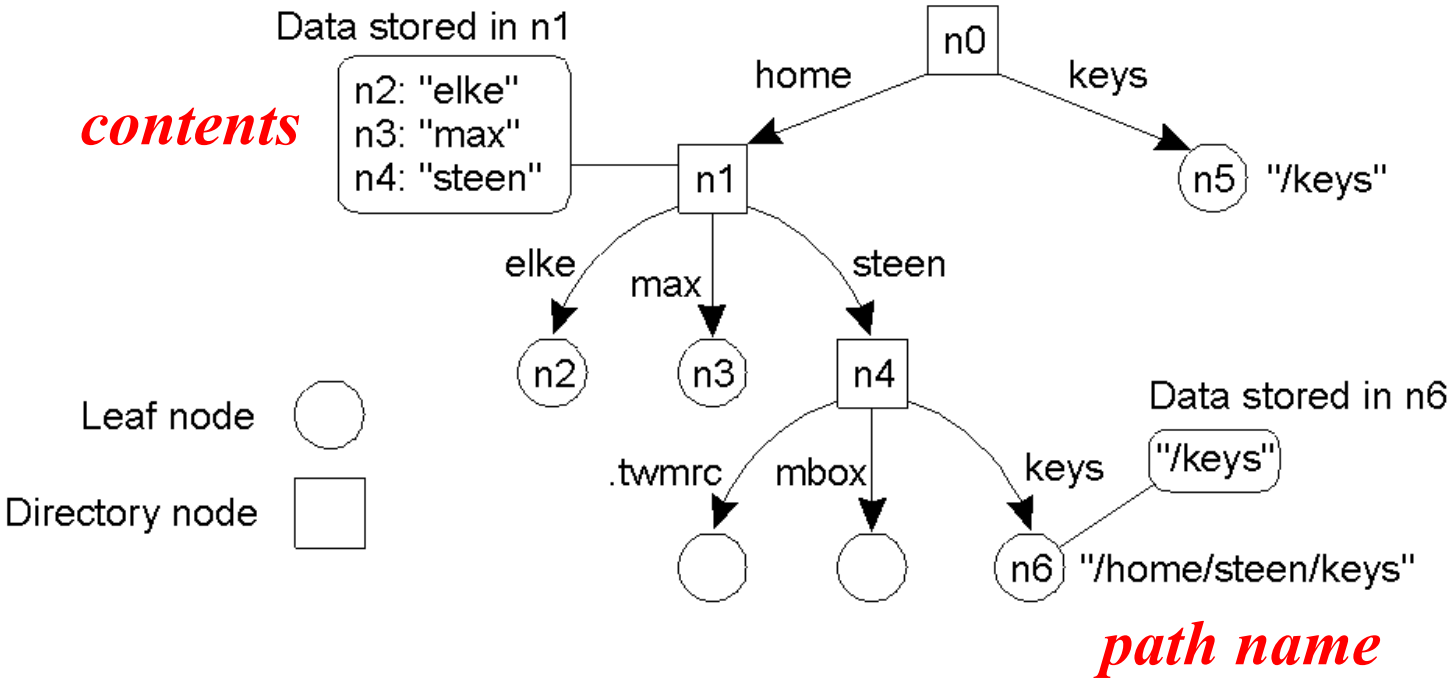
- **Distributed Hash Table (DHT) for the Chord system**
- Finger table
  - **# entries <= log(size ring)**
  - **p node, i index**
  - **n <= succ(n)**
- Key resolution
  - **If m < k <= n, then node n holds the data with key k**
- Examples:
  - **Resolving k = 26 from p = 1**
  - **Resolving k = 12 from p = 28**



Structured Naming

- A **name space** can be represented by a **naming graph** with a single root node (see Slide 15)
- 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
- 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**
  - **...**
- Both leaf nodes and directory nodes can have attributes

Name Space – Naming Graph



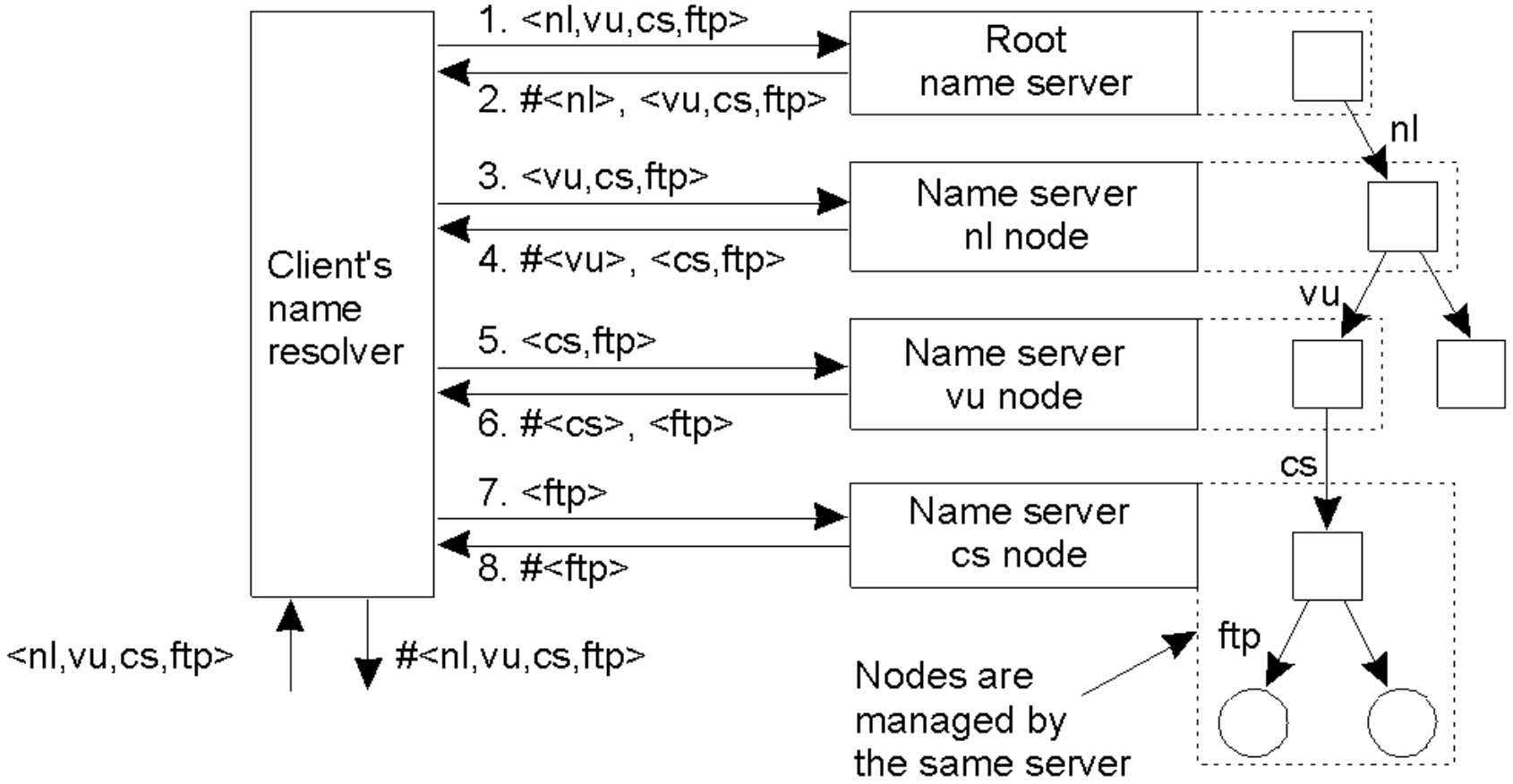
- **Hard link:** Follow a ***path name*** in a naming graph from one node to another
- **Soft link:** Read the ***contents*** of a node, i.e., the ***name*** of another node to go to

Name Resolution

- **Problem:** To **resolve** a name, i.e., find an address corresponding to a name
- We need a directory node that maintains a name-to-address binding
- **Question:** How do we find such a directory node?
- **Closure mechanism:** Selects the implicit context from which to start the name resolution, e.g.,
  - **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:** The closure mechanism might also determine how name resolution should proceed

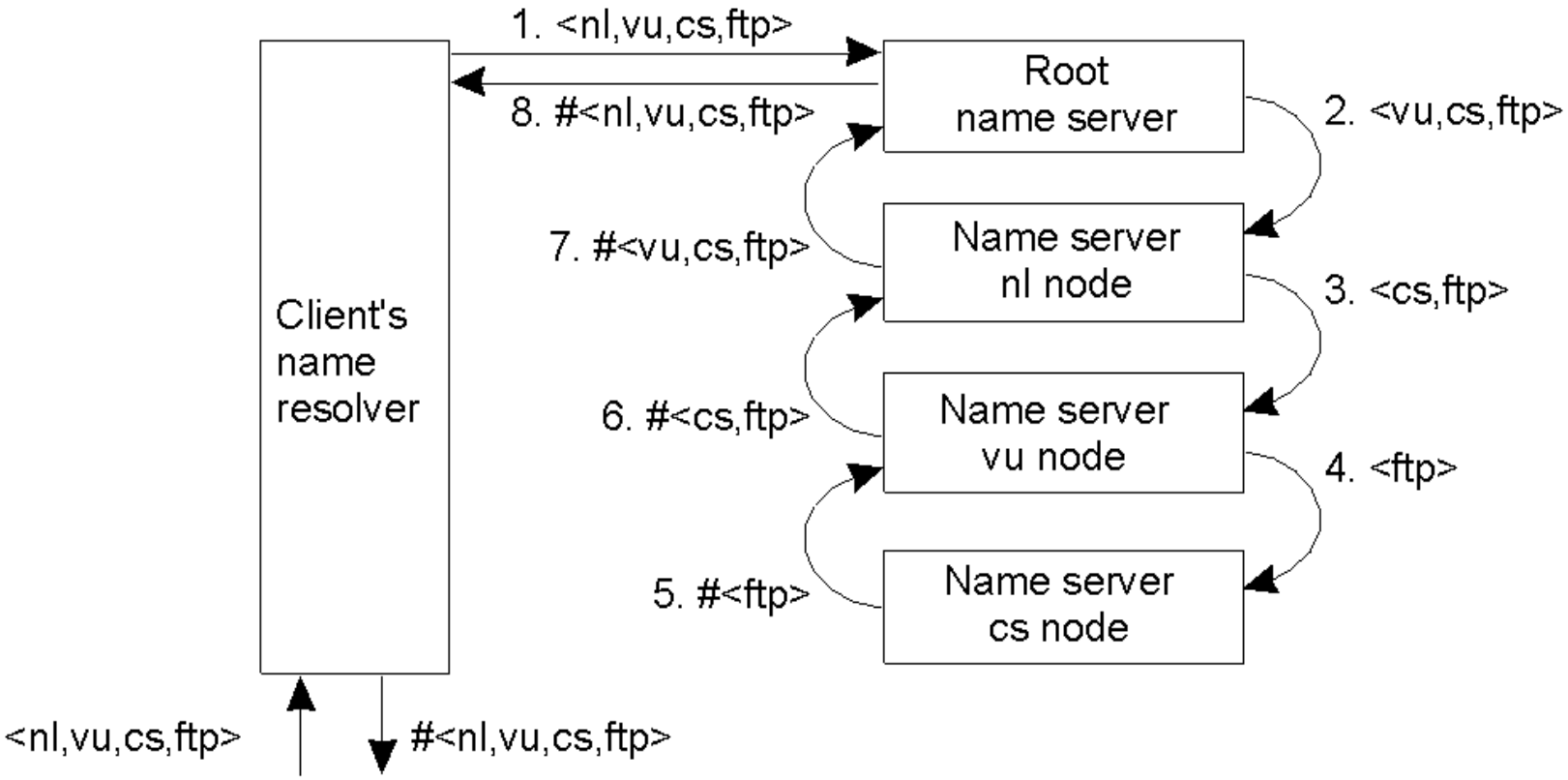
Name Resolution

- **Iterative name resolution**



Name Resolution

- **Recursive name resolution**



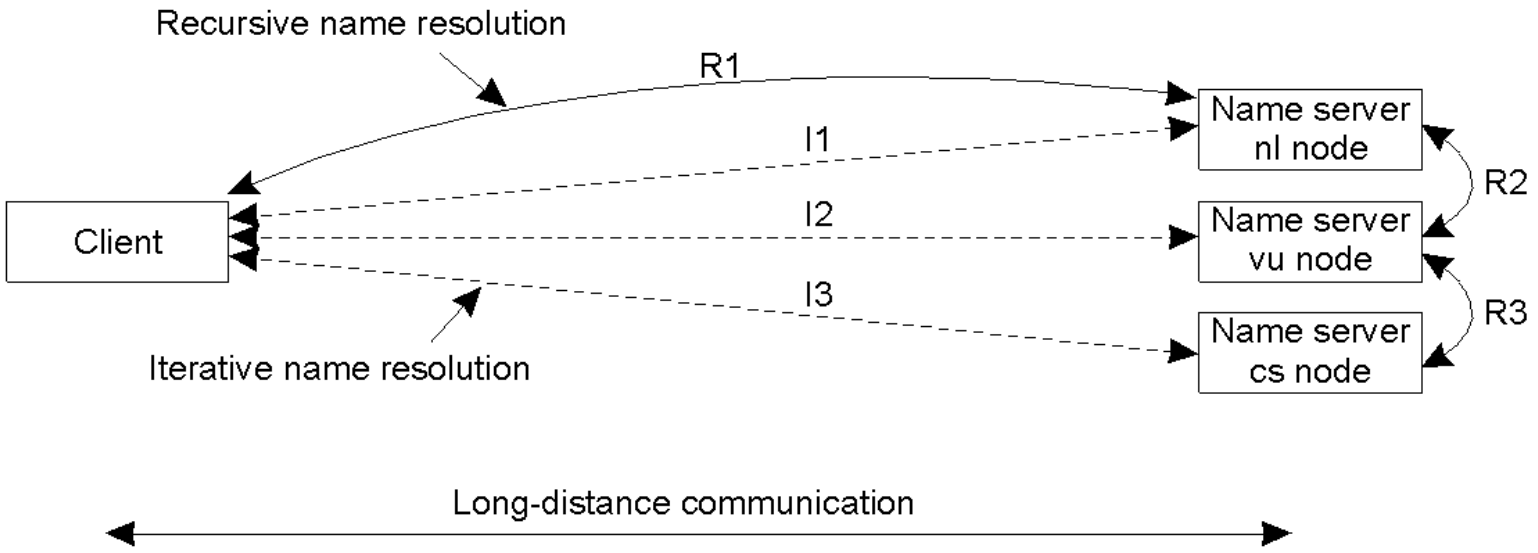
Name Resolution

- **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	<ni,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>

Name Resolution

- Iterative name resolution has a higher communication cost than recursive name resolution



Mounting Name Spaces

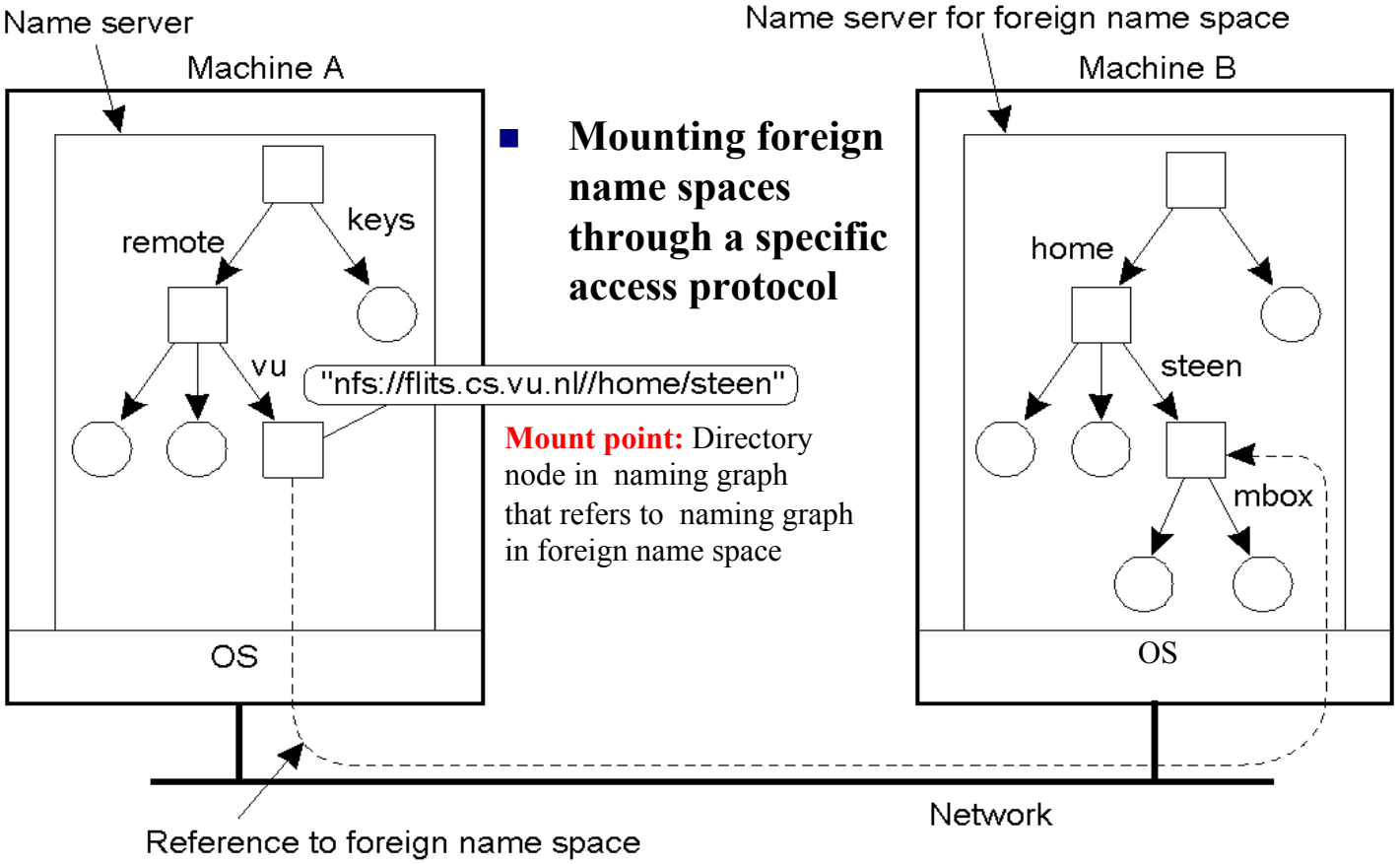
- **Problem:** We have other name spaces that we wish to access from a given name space
- **Solution:** Introduce a naming scheme by which path names of name spaces are concatenated (e.g., URLs)
- **Example:**

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



- **Mounting foreign name spaces through a specific access protocol**

Mount point: Directory node in naming graph that refers to naming graph in foreign name space

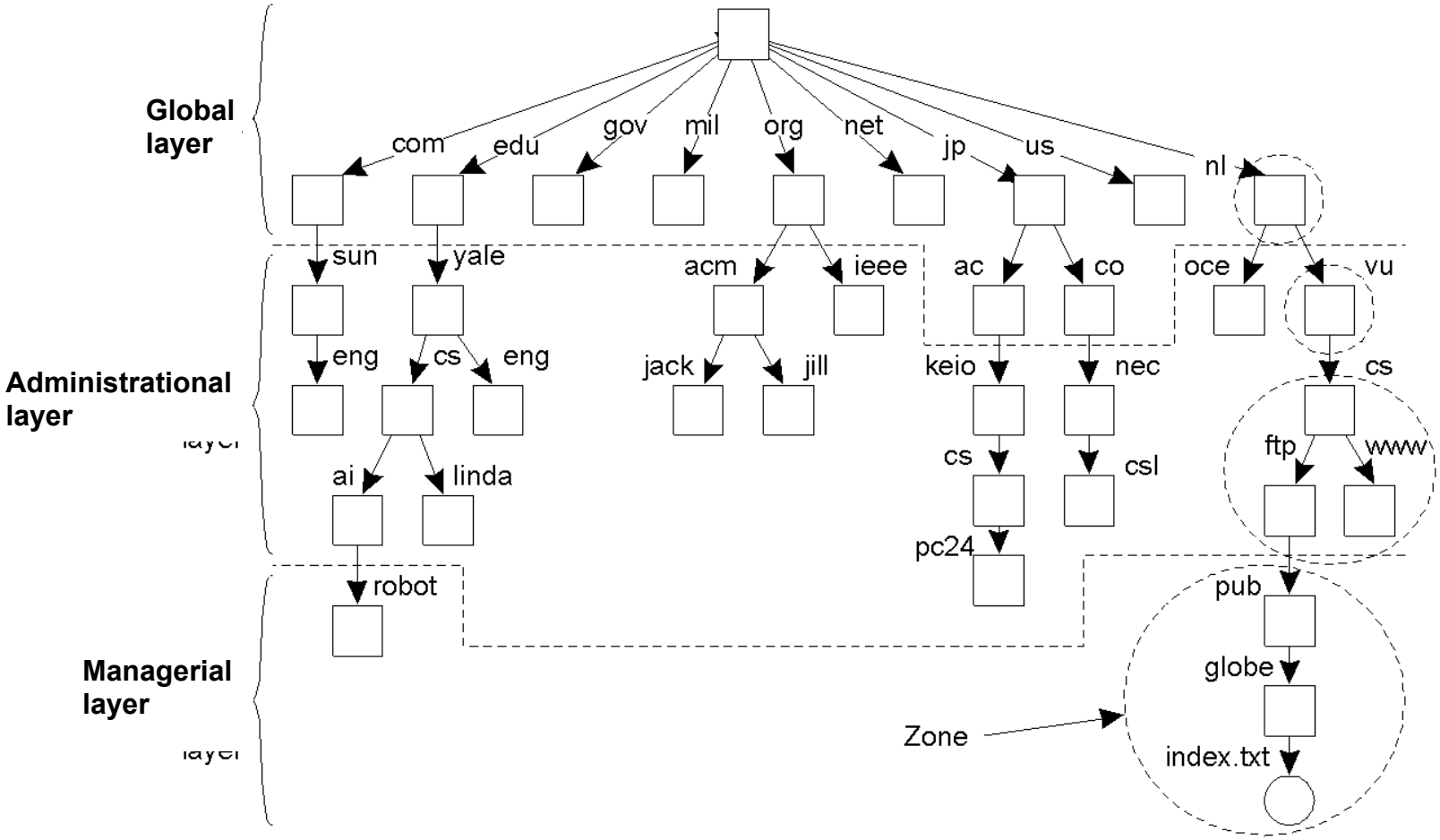
Mounting point: Directory node in naming graph in foreign name space that is referred to

Name Space Distribution

- **Basic idea:** Distribute management of name spaces 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 such nodes by different administrations**
  - **Administrational layer:** **Mid-level directory nodes. Main issue is grouping together such nodes 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 such nodes to local name servers**

Name Space Distribution

- **Example:** The name space of the Domain Name System (DNS) (see Slides 30-32), 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

Scalability Issues

- **Size scalability:** Ensure that name servers can handle a large number of requests per time unit; overloading of high-level name servers is the problem
  - **Solution is to partition a node into a number of sub-nodes and evenly assign entities to sub-nodes**
  - **Naive partitioning can introduce node management problems, because a sub-node might need to know how its parent and children are partitioned**
- **Geographical scalability:** 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**

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**
- A 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)**
- Listing RDNs in sequence yields a hierarchy of Directory Entries, referred to as the **Directory Information Tree (DIT)**

LDAP Directory Service

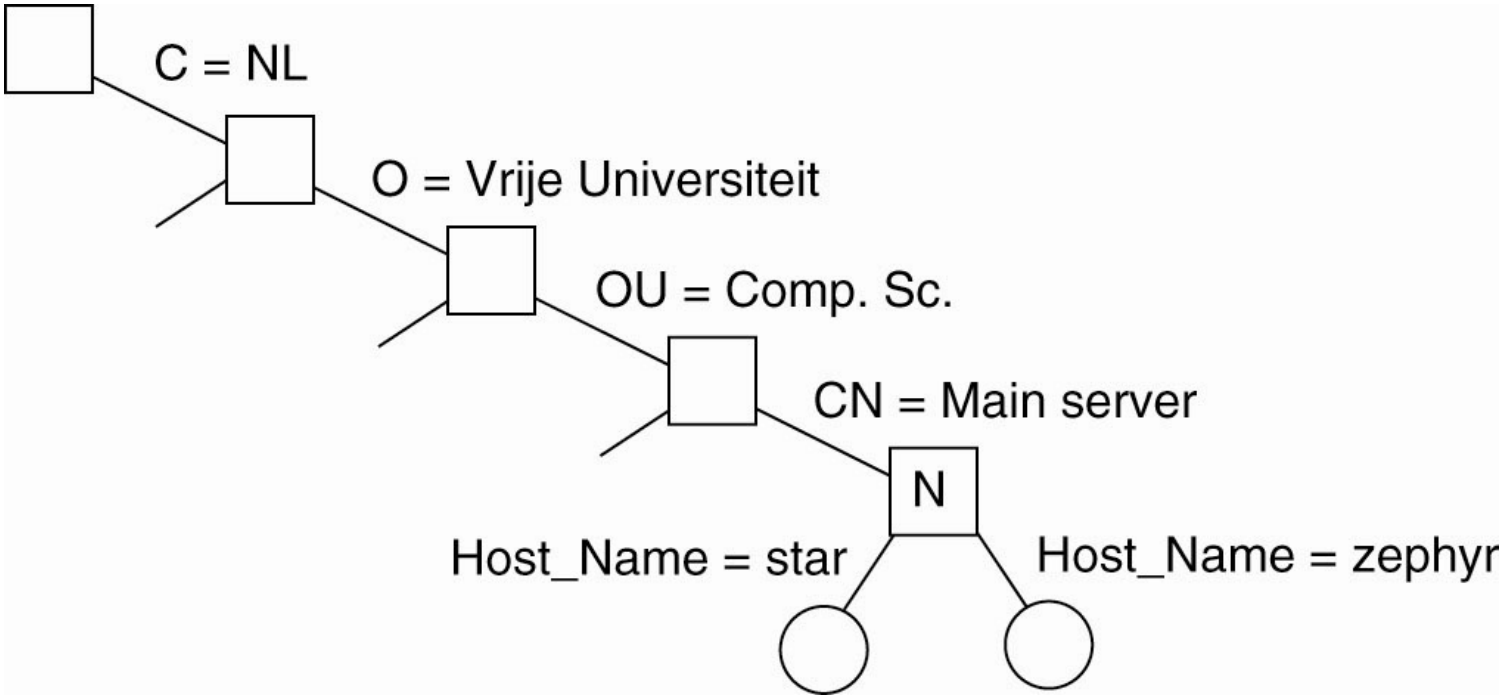
- Two example LDAP Directory Entries, illustrating:
  - **RDNs, i.e., (attribute, value) pairs**
  - **LDAP naming conventions for attributes**

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

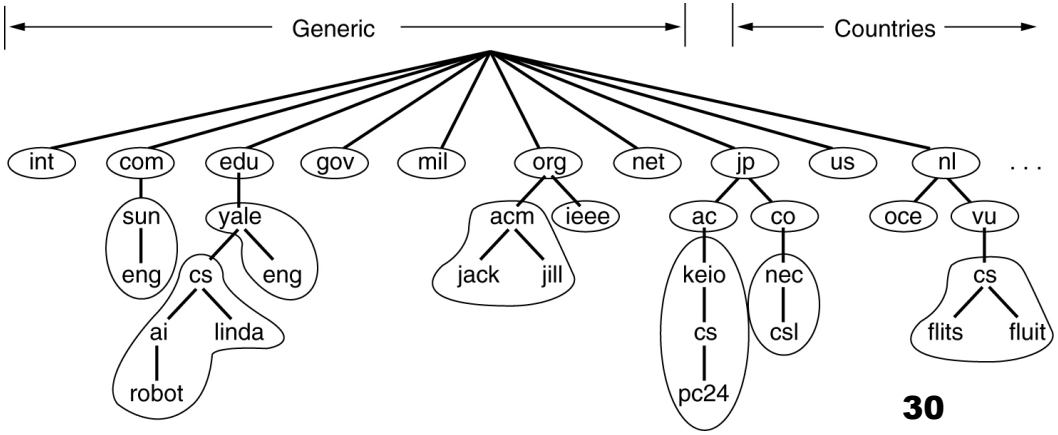
LDAP Directory Service

- Part of the LDAP Directory Information Tree (DIT)



Domain Name System

- The **Domain Name System (DNS)** is the hierarchical Naming/Location 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"