ECE151 – Lecture 5

Chapter 4

Naming

Naming

Names, identifiers, and addresses

Name resolution

Name space implementation

Essence: Names are used to denote entities in a distributed system. To operate on an entity, we need to access it at an **access point**. Access points are entities that are named by means of an **address**.

Note: A **location-independent** name for an entity *E*, is independent from the addresses of the access points offered by *E*.

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:

P1 Each identifier refers to at most one entity

P2 Each entity is referred to by at most one identifier

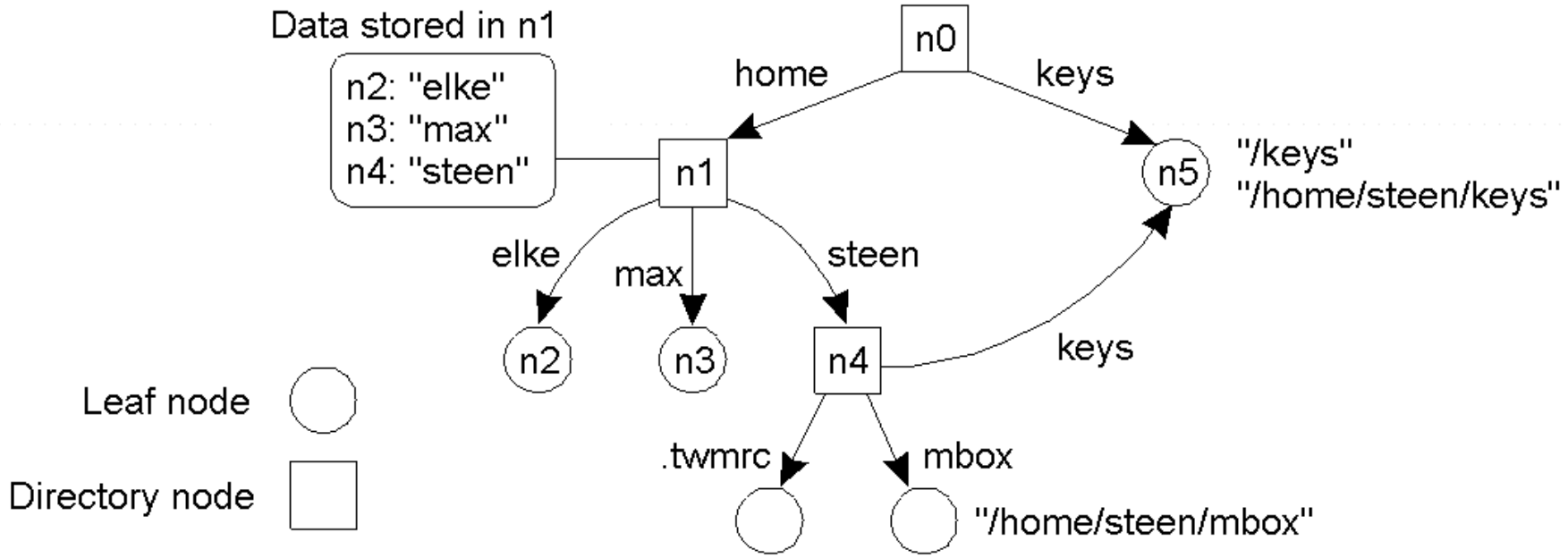
P3 An identifier always refers to the same entity (prohibits reusing an identifier)

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

Question: Can the content of an identifier ever change?

Name Spaces

Essence: a graph in which a **leaf node** represents a (named) entity. A **directory node** is an entity that refers to other nodes. A directory node contains a (directory) table of *(edge label, node identifier)* pairs.



Name Spaces

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

...

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

Name Resolution

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

Closure mechanism: The mechanism to select the implicit context from which to start name resolution:

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: IP address of the VU's Web server

Question: Why are closure mechanisms always *implicit*?

Observation: A closure mechanism may also determine how name resolution should proceed

Name Linking

Hard link: What we have described so far as a **path name**: a name that is resolved by following a specific path in a naming graph from one node to another.

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*.

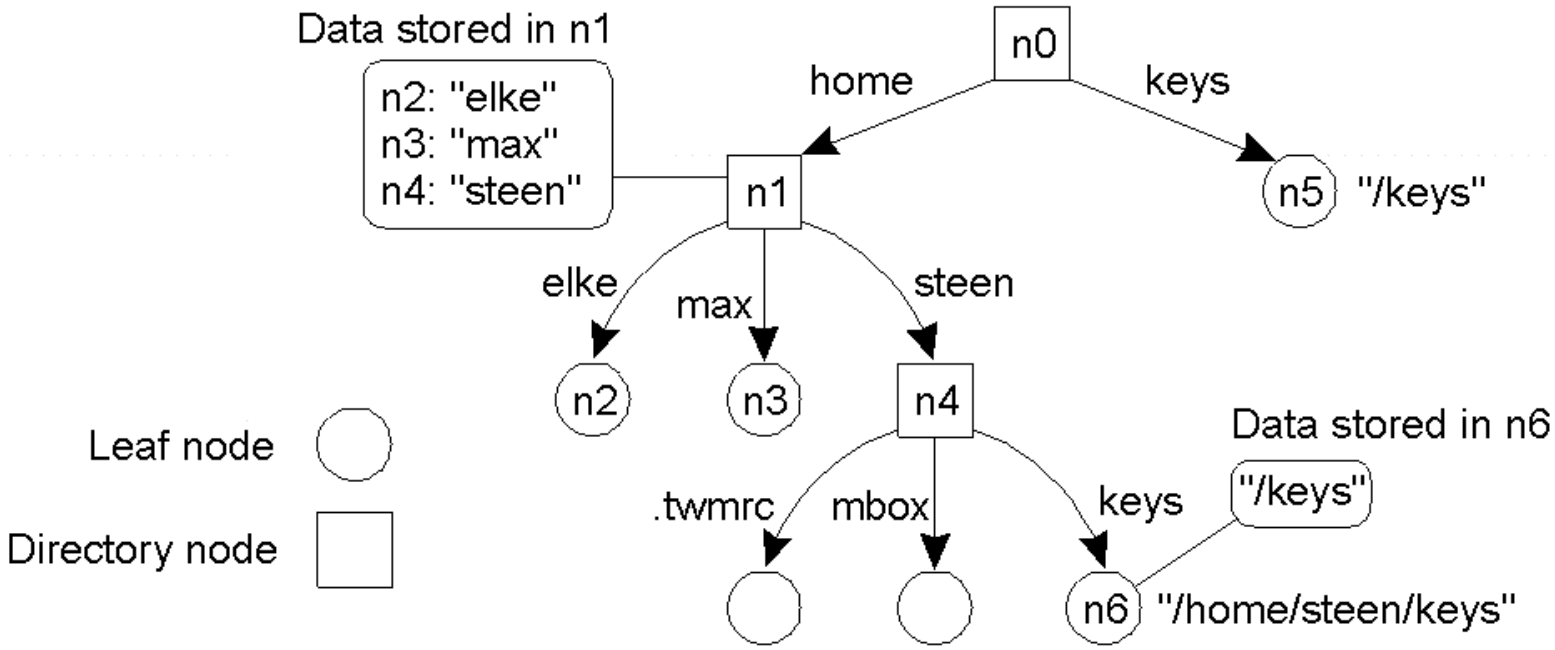
Name resolution continues with *name*.

Observations:

The name resolution process determines that we read the *content* of a node, in particular, the name in the other node that we need to go to.

One way or the other, we know where and how to start name resolution given *name*.

Linking and Mounting



The concept of a symbolic link explained in a naming graph.

Merging Name Spaces

Problem: We have different name spaces that we wish to access from any given name space.

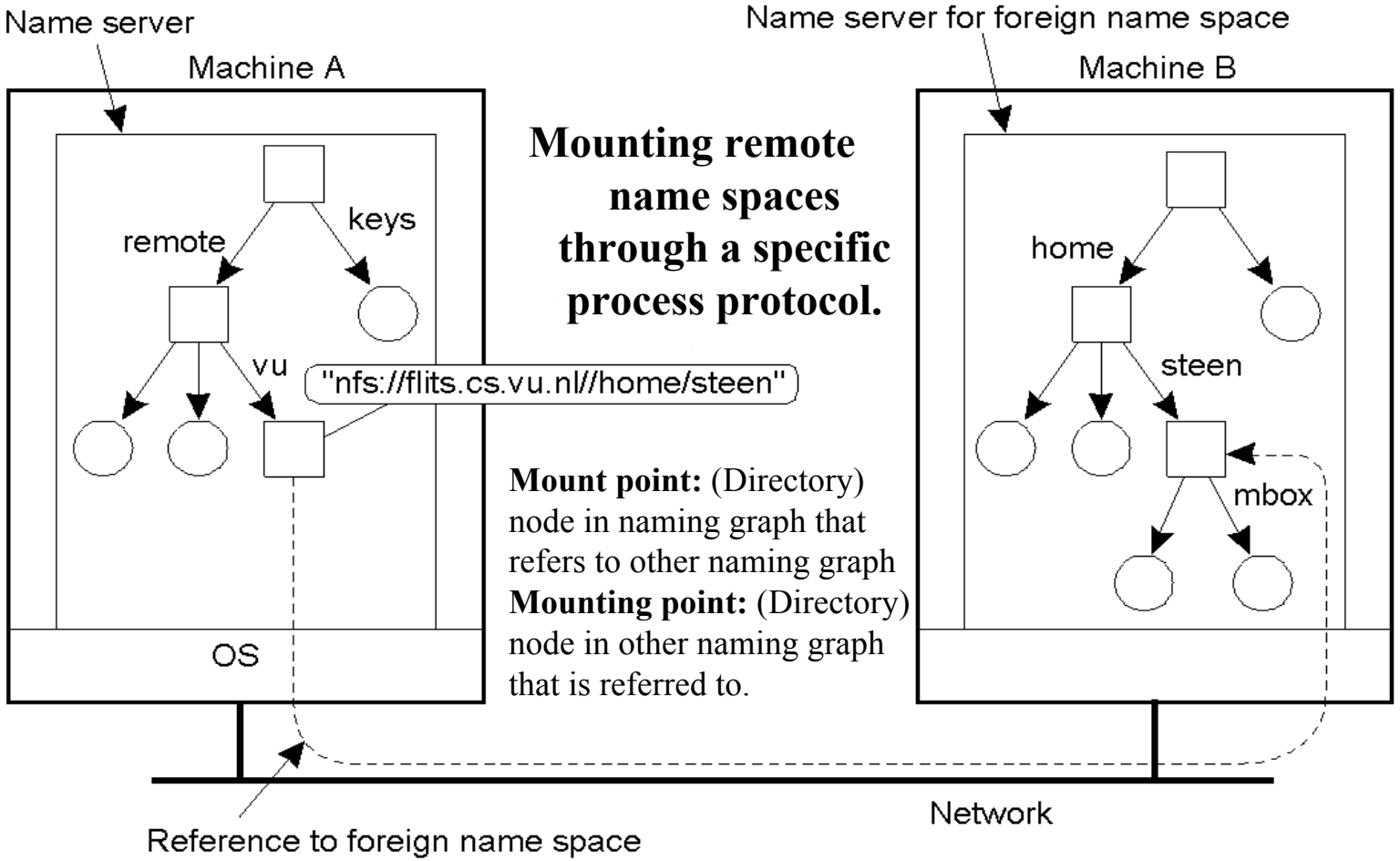
Solution 1: Introduce a naming scheme by which pathnames of different name spaces are simply concatenated (URLs).

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

ftp	Name of 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) node in the name space rooted at the context node mapped to the FTP server

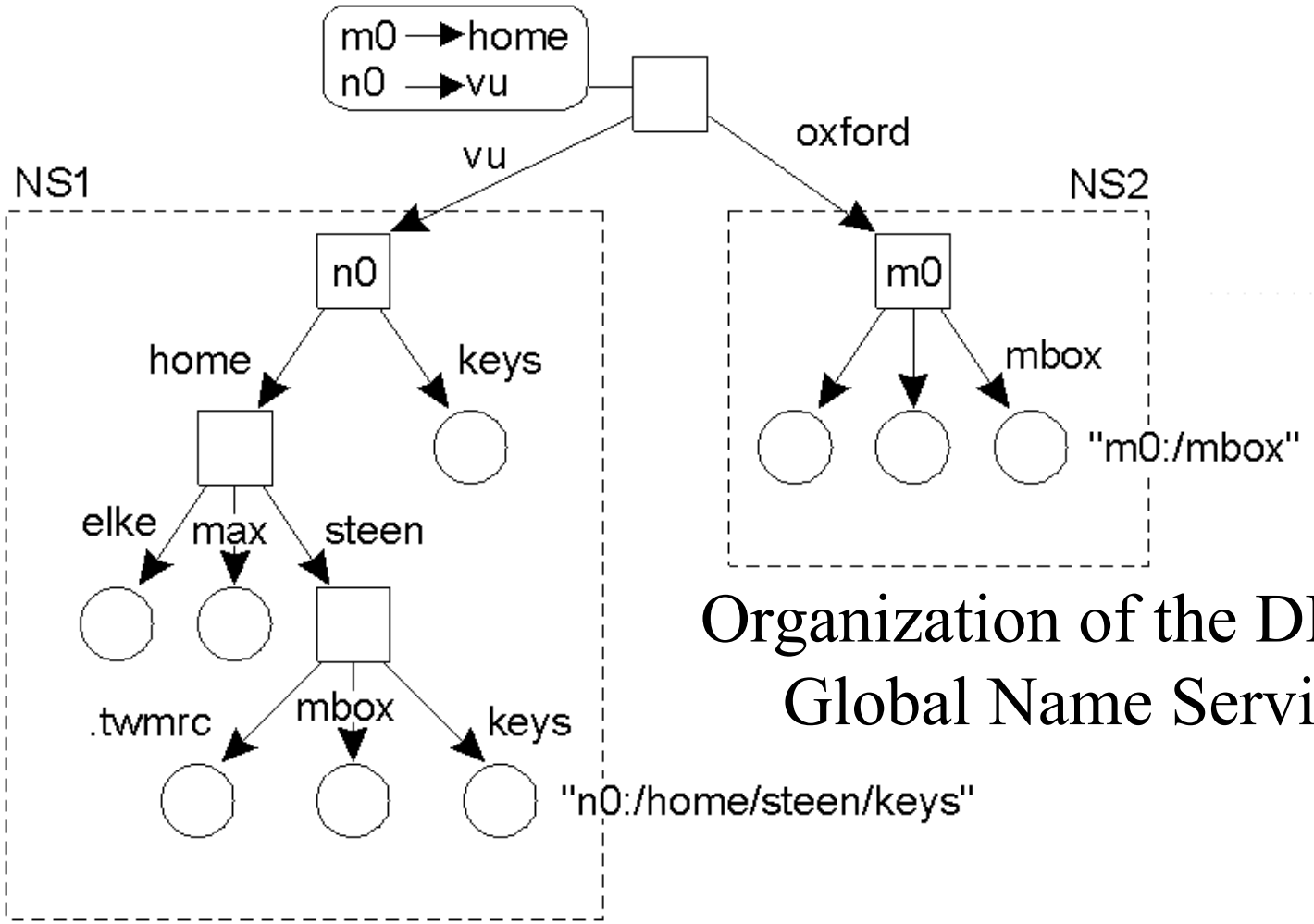
Merging Name Spaces

Introduce nodes that contain the name of a node in a "foreign" name space, along with the information how to select the initial context in that foreign name space (Jade).



Merging Name Spaces

Use only *full pathnames*, in which the starting context is explicitly identified, and merge by adding a new root node (DEC's Global Name Space).



Organization of the DEC Global Name Service

Name Space Implementation

Basic issue: Distribute the name resolution process as well as name space management across multiple machines, by distributing nodes of the naming graph.

Consider a hierarchical naming graph and distinguish three levels:

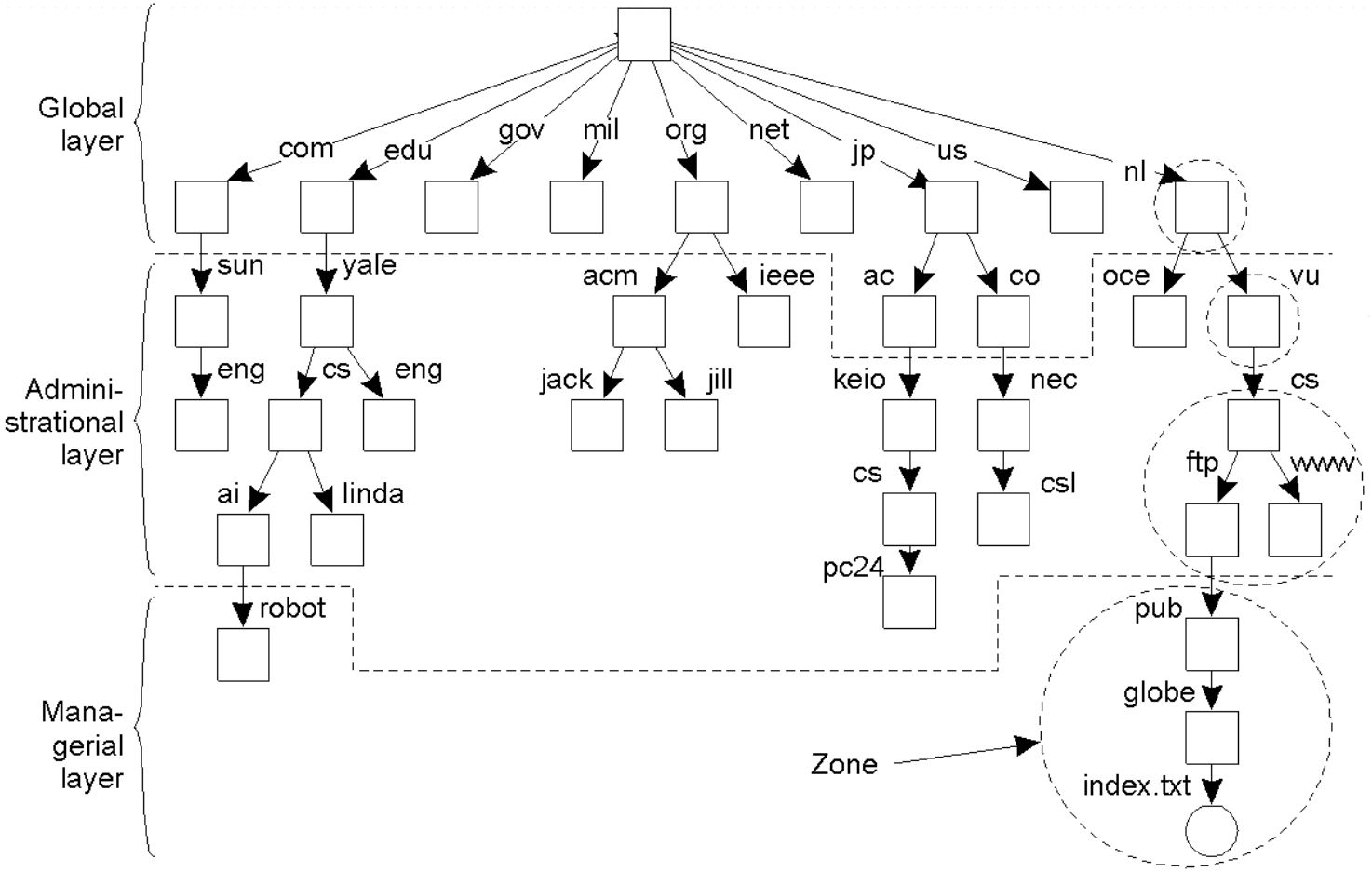
Global level: Consists of the high-level directory nodes. Main aspect is that these directory nodes have to be jointly managed by different administrations

Administrational level: Contains mid-level directory nodes that can be grouped in such a way that each group can be assigned to a separate administration.

Managerial level: Consists of low-level directory nodes within a single administration. Main issue is effectively mapping directory nodes to local name servers.

Name Space Implementation

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

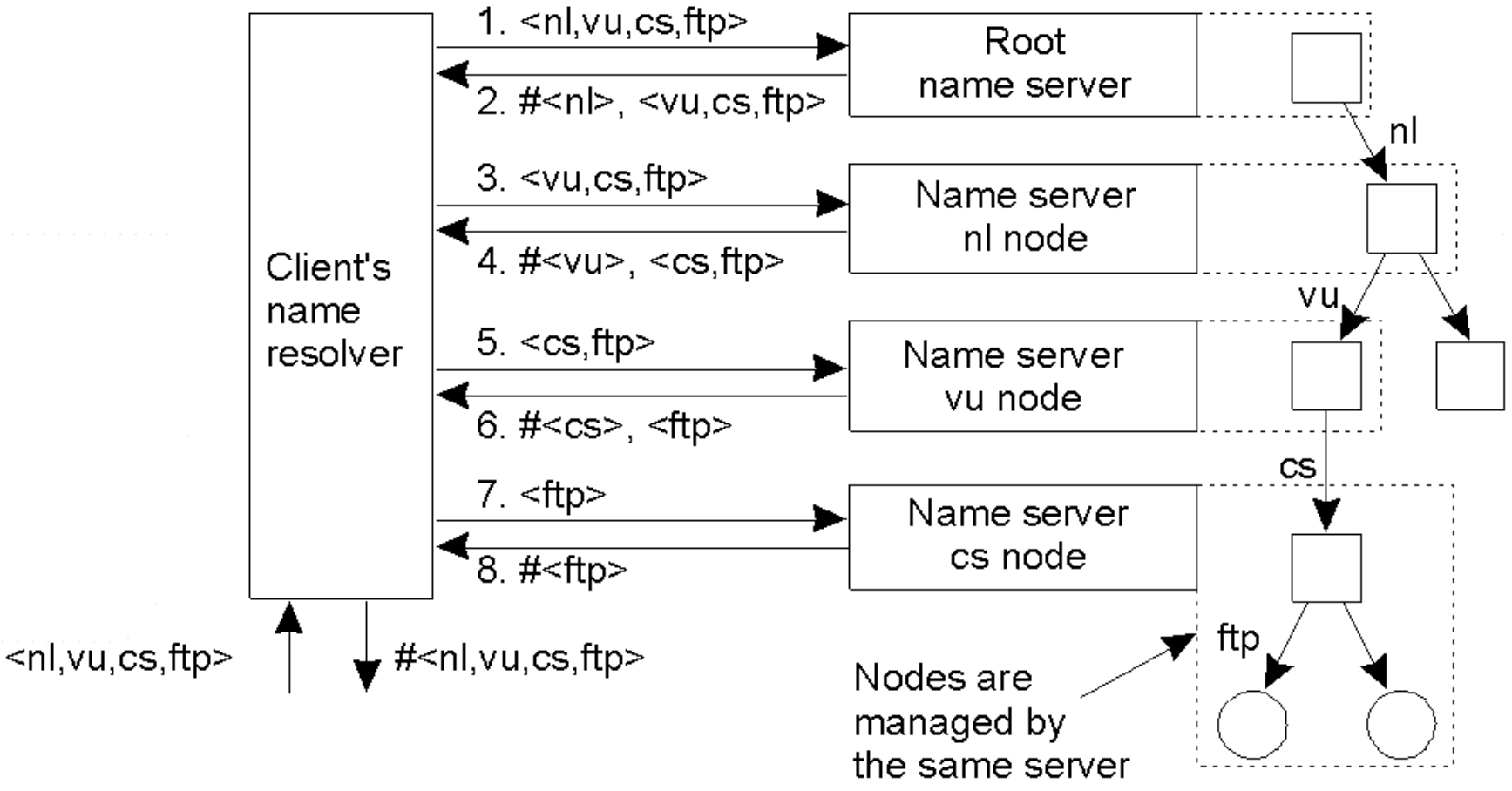


Name Space Implementation

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

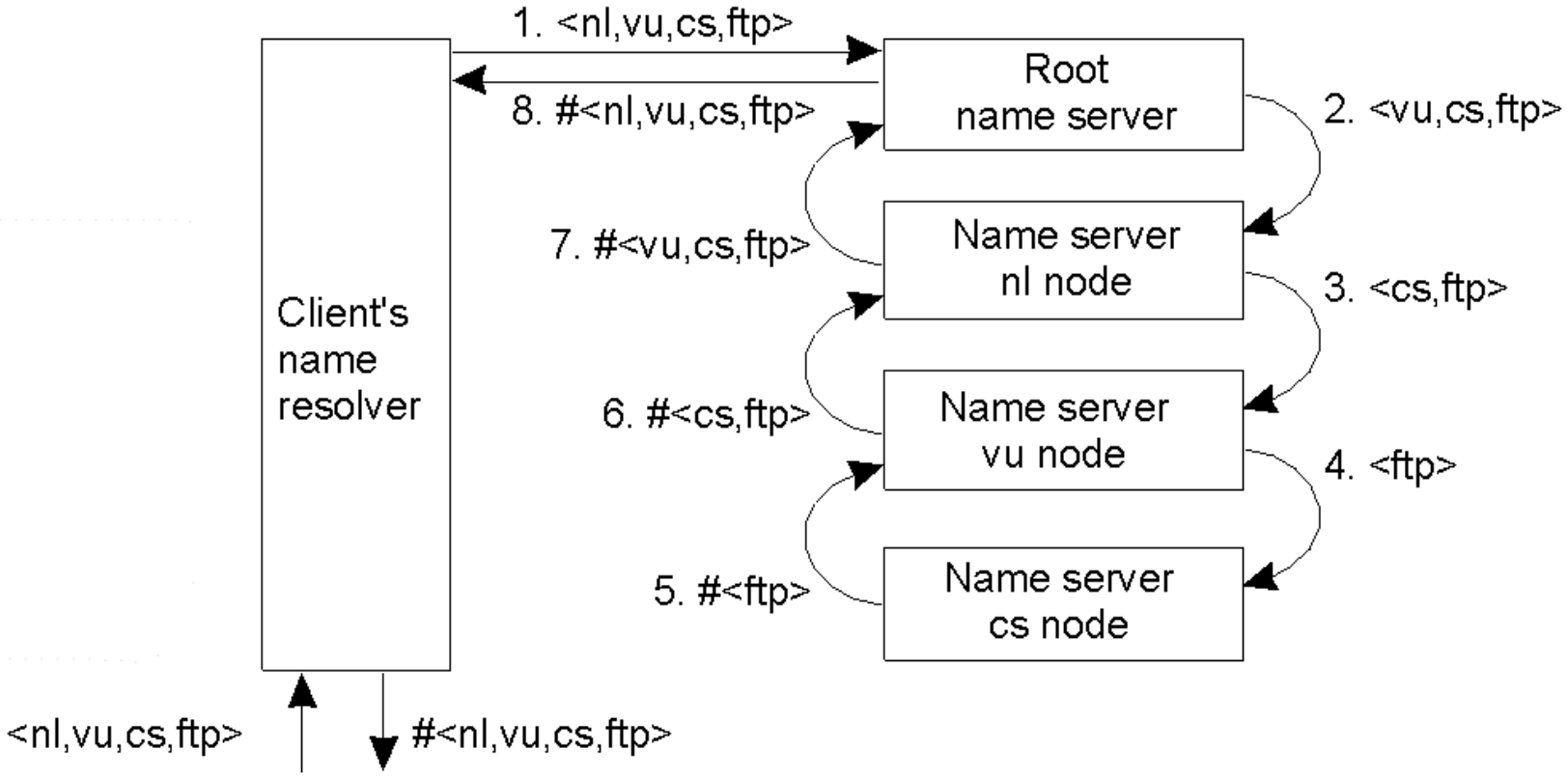
A comparison between name servers for implementing nodes from a large-scale name space partitioned into a global layer, as an administrational layer, and a managerial layer.

Implementation of Name Resolution



Implementation of Name Resolution

The principle of recursive name resolution.



Implementation of Name Resolution

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>
ni	<vu,cs,ftp>	#<vu>	<cs,ftp>	#<cs> #<cs,ftp>	#<vu> #<vu,cs> #<vu,cs,ftp>
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>

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

Scalability Issues

Size scalability: We need to ensure that servers can handle a large number of requests per time unit, i.e. high-level servers are in big trouble.

Solution: Assume (at least at global and administrational level) that content of nodes hardly ever changes.

In that case, we can apply extensive replication by mapping nodes to multiple servers, and start name resolution at the nearest server.

Observation: An important attribute of many nodes is the **address** where the represented entity can be contacted. Replicating nodes makes large-scale traditional name servers unsuitable for locating mobile entities.

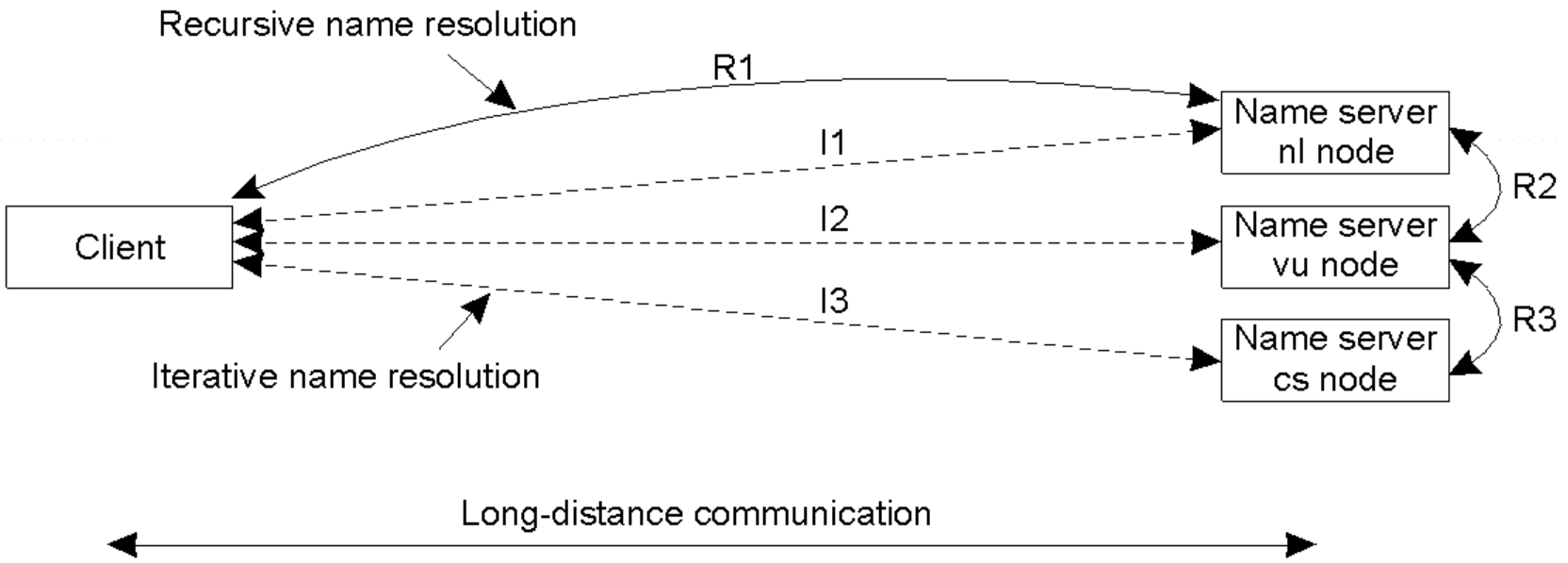
Scalability Issues

Geographical scalability: We need to ensure that the name resolution process scales across large geographical distances.

Problem: By mapping nodes to servers that may, in principle, be located anywhere, we introduce an implicit location dependency in our naming scheme.

Solution: No general one available yet.

Implementation of Name Resolution



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

Locating Mobile Entities

Naming versus locating objects

Simple solutions

Home-based approaches

Hierarchical approaches

Naming & Locating Objects

Location service: Solely aimed at providing the addresses of the *current* locations of entities.

Assumption: Entities are mobile, so that their current address may change frequently.

Naming service: Aimed at providing the content of nodes in a name space, given a (compound) name.
Content consists of different (attribute,value) pairs.

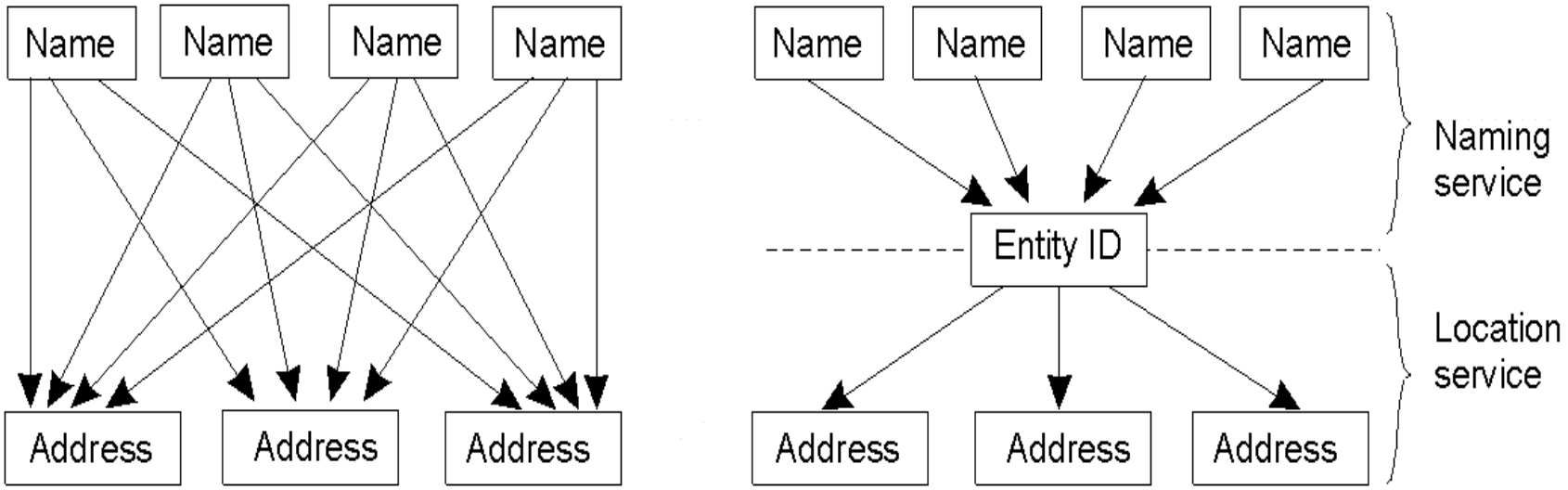
Assumption: Node contents at global and administrational level is relatively stable for scalability reasons.

Observation: If a traditional naming service is used to locate entities, we also have to assume that node contents at the managerial level is stable, as we can use only names as identifiers (think of Web pages).

Naming versus Locating Entities

Problem: It is not realistic to assume stable node contents down to the local naming level

Solution: Decouple naming from locating entities



a) Direct, single level mapping between names and addresses.

b) T-level mapping using identities.

Naming versus Locating Entities

Name: Any name in a traditional naming space

Entity ID: A true identifier

Address: Provides *all* information necessary to contact an entity

Observation: An entity's name is now completely independent from its location.

Question: What may be a typical address?

Simple Solutions for Locating Entities

Broadcasting: Simply broadcast the ID, requesting the entity to return its current address.

Can never scale beyond local-area networks (think of ARP/RARP)

Requires all processes to listen to incoming location requests

Forwarding pointers: Each time an entity moves, it leaves behind a pointer telling where it has gone to.

Dereferencing can be made entirely transparent to clients by simply following the chain of pointers

Update a client's reference as soon as present location has been found

Geographical scalability problems:

- Long chains are not fault tolerant
- Increased network latency at dereferencing

Essential to have separate chain reduction mechanisms

Home-Based Approaches

Single-tiered scheme: Let a **home** keep track of where the entity is:

An entity's **home address** is registered at a naming service

The home registers the **foreign address** of the entity

Clients always contact the home first, and then continues with the foreign location

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:

- The home address has to be supported as long as the entity lives.
- The home address is fixed, which means an unnecessary burden when the entity permanently moves to another location
- Poor geographical scalability (the entity may be next to the client)

Question: How can we solve the "permanent move" problem?

Hierarchical Location Services

Basic idea: Build a large-scale search tree for which the underlying network is divided into hierarchical domains.

Each domain is represented by a separate directory node.

The address of an entity is stored in a leaf node, or in an intermediate node

Intermediate nodes contain a pointer to a child if and only if the subtree rooted at the child stores an address of the entity

The root knows about all entities

Start lookup at local leaf node

If node knows about the entity, follow downward pointer, otherwise go one level up

Upward lookup always stops at root

HLS: Scalability Issues

Size scalability: Again, we have a problem of overloading higher-level nodes:

Only solution is to partition a node into a number of subnodes and evenly assign entities to subnodes

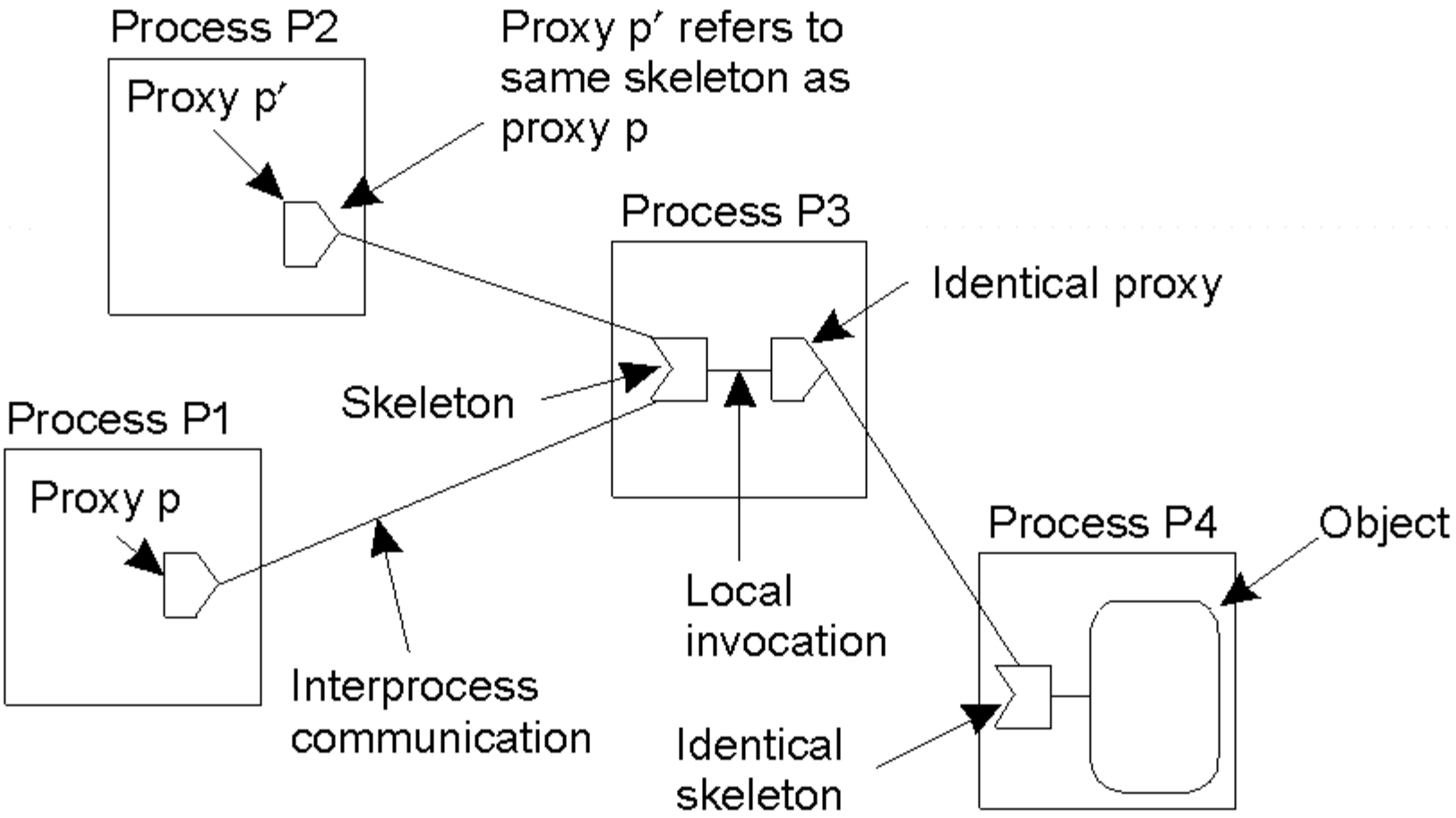
Naive partitioning may introduce a node management problem, as a subnode may have to know how its parent and children are partitioned.

Geographical scalability: We have to ensure that lookup operations generally proceed monotonically in the direction of where we'll find an address:

If entity *E* generally resides in California, we should not let a root subnode located in France store *E*'s contact record.

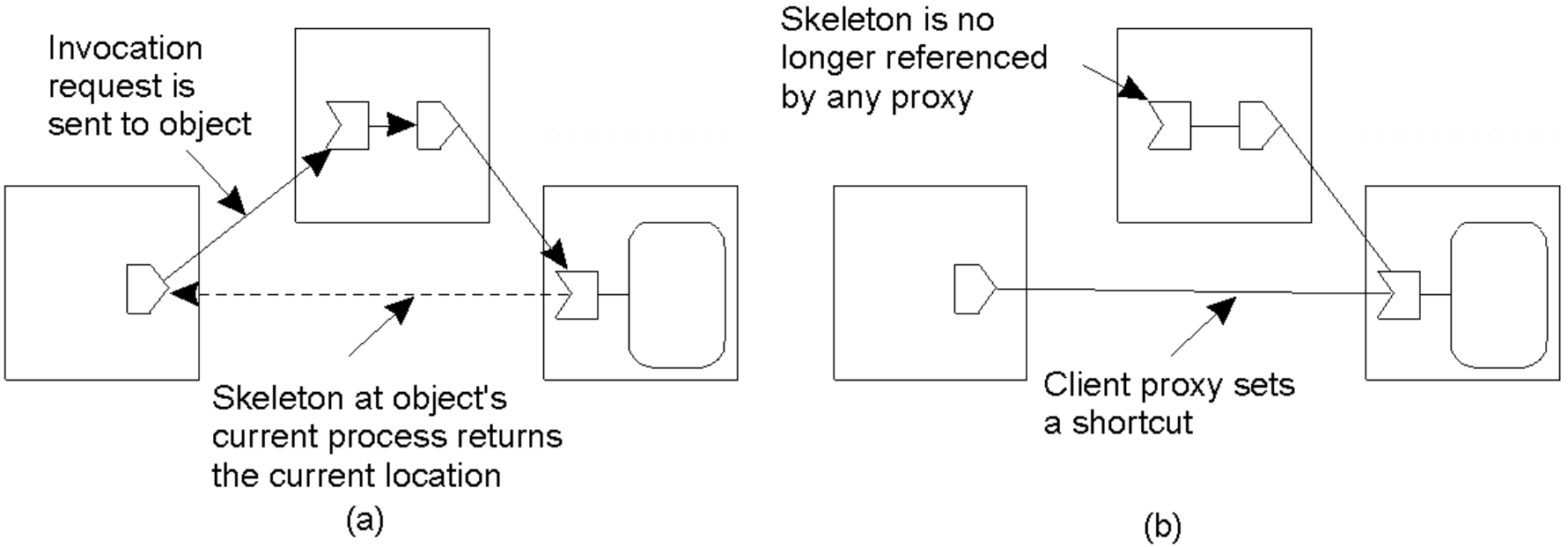
Unfortunately, subnode placement is not that easy, and only a few tentative solutions are known

Forwarding Pointers (1)



The principle of forwarding pointers using (*proxy, skeleton*) pairs.

Forwarding Pointers (2)



Redirecting a forwarding pointer, by storing a shortcut in a proxy.

The DNS Name Space

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

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)

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

Name	Record type	Record value
cs.vu.nl	SOA	star (1999121502,7200,3600,2419200,86400)
cs.vu.nl	NS	star.cs.vu.nl
cs.vu.nl	NS	top.cs.vu.nl
cs.vu.nl	NS	solo.cs.vu.nl
cs.vu.nl	TXT	"Vrije Universiteit - Math. & Comp. Sc."
cs.vu.nl	MX	1 zephyr.cs.vu.nl
cs.vu.nl	MX	2 tornado.cs.vu.nl
cs.vu.nl	MX	3 star.cs.vu.nl
star.cs.vu.nl	HINFO	Sun Unix
star.cs.vu.nl	MX	1 star.cs.vu.nl
star.cs.vu.nl	MX	10 zephyr.cs.vu.nl
star.cs.vu.nl	A	130.37.24.6
star.cs.vu.nl	A	192.31.231.42
zephyr.cs.vu.nl	HINFO	Sun Unix
zephyr.cs.vu.nl	MX	1 zephyr.cs.vu.nl
zephyr.cs.vu.nl	MX	2 tornado.cs.vu.nl
zephyr.cs.vu.nl	A	192.31.231.66
www.cs.vu.nl	CNAME	soling.cs.vu.nl
ftp.cs.vu.nl	CNAME	soling.cs.vu.nl
soling.cs.vu.nl	HINFO	Sun Unix
soling.cs.vu.nl	MX	1 soling.cs.vu.nl
soling.cs.vu.nl	MX	10 zephyr.cs.vu.nl
soling.cs.vu.nl	A	130.37.24.11
laser.cs.vu.nl	HINFO	PC MS-DOS
laser.cs.vu.nl	A	130.37.30.32
vucs-das.cs.vu.nl	PTR	0.26.37.130.in-addr.arpa
vucs-das.cs.vu.nl	A	130.37.26.0

DNS Implementation (2)

Name	Record type	Record value
cs.vu.nl	NIS	solo.cs.vu.nl
solo.cs.vu.nl	A	130.37.21.1

Part of the description for the *vu.nl* domain which contains the *cs.vu.nl* domain.

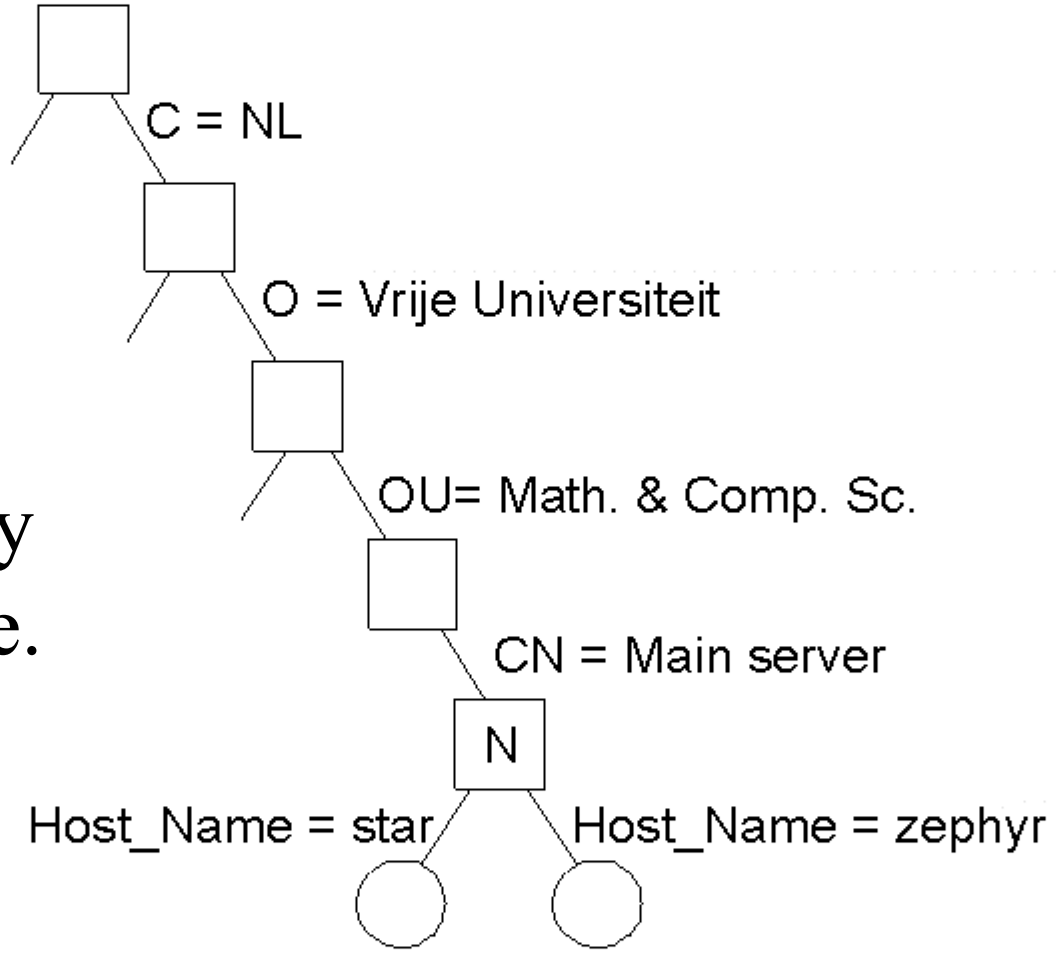
The X.500 Name Space (1)

A simple example of a X.500 directory entry using X.500 naming conventions.

Attribute	Abbr.	Value
Country	C	NL
Locality	L	Amsterdam
Organization	L	Vrije Universiteit
OrganizationalUnit	OU	Math. & Comp. Sc.
CommonName	CN	Main server
Mail_Servers	--	130.37.24.6, 192.31.231,192.31.231.66
FTP_Server	--	130.37.21.11
WWW_Server	--	130.37.21.11

The X.500 Name Space (2)

Part of the directory information tree.



The X.500 Name Space (3)

Two directory entries having *Host_Name* as RDN.

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

Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Math. & Comp. Sc.
CommonName	Main server
Host_Name	zephyr
Host_Address	192.31.231.66