Naming Technologies within Distributed Systems

Names, Identifiers and Addresses

Naming

Naming systems play an important role in all computer systems, and especially within a distributed environment.

The three main areas of study:

1. The organisation and implementation of human-friendly naming systems.
2. Naming as it relates to mobile entities.
3. Garbage collection – what to do when a name is no longer needed.

Some Definitions

- *Name* – a string (often human-friendly) that refers to an entity.
- *Entity* – just about any resource.
- *Address* – "access-point" of an entity.

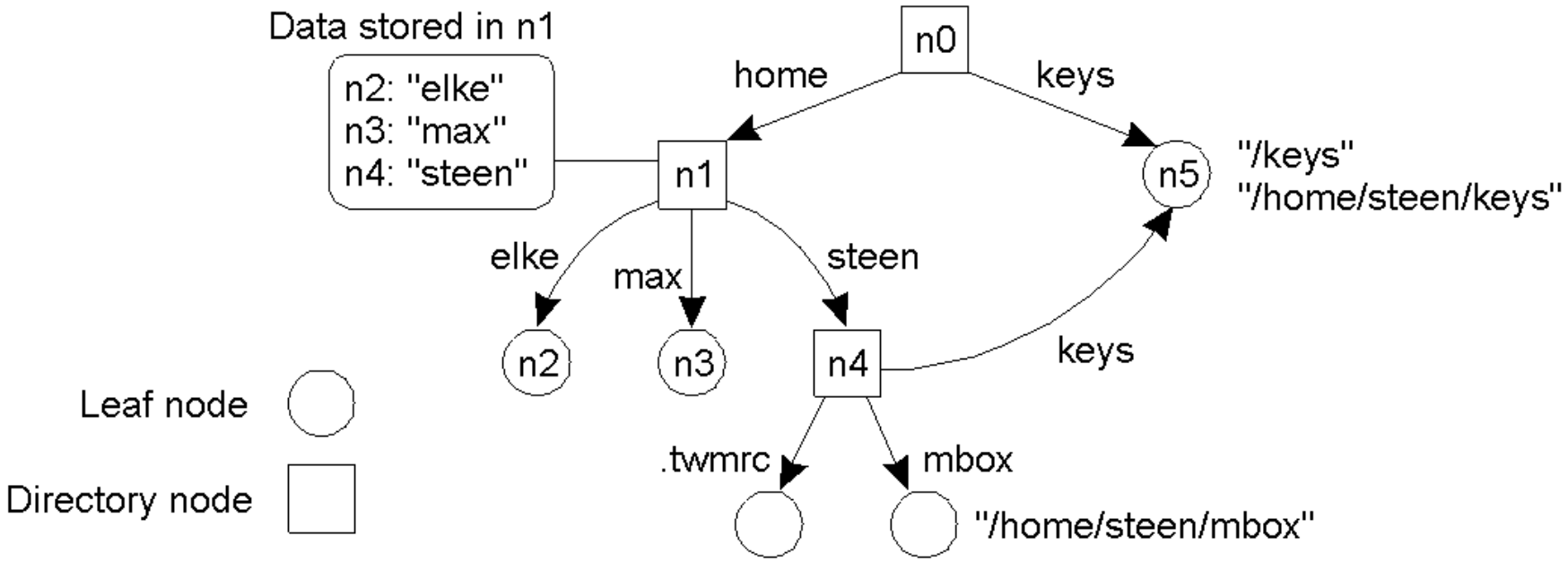
A **name** for an **entity** that is independent of an **address** is referred to as "location independent".

- *Identifier* – a reference to an entity that is often unique and never reused.

Namespaces

- Names are often organised into **namespaces**.
  - Within distributed systems, a namespace is represented by a labelled, directed graph with two types of nodes:
    - *leaf nodes*: information on an entity.
    - *directory nodes*: a collection of named outgoing edges (which can lead to any other type of node).
  - Each namespace has *at least* one **root** node.
  - Nodes can be referred to by path names (with absolute or relative).
  - File systems are a classic example…

Name Spaces and Graphs



A general **naming graph** with a single root node, showing relative and absolute path names.

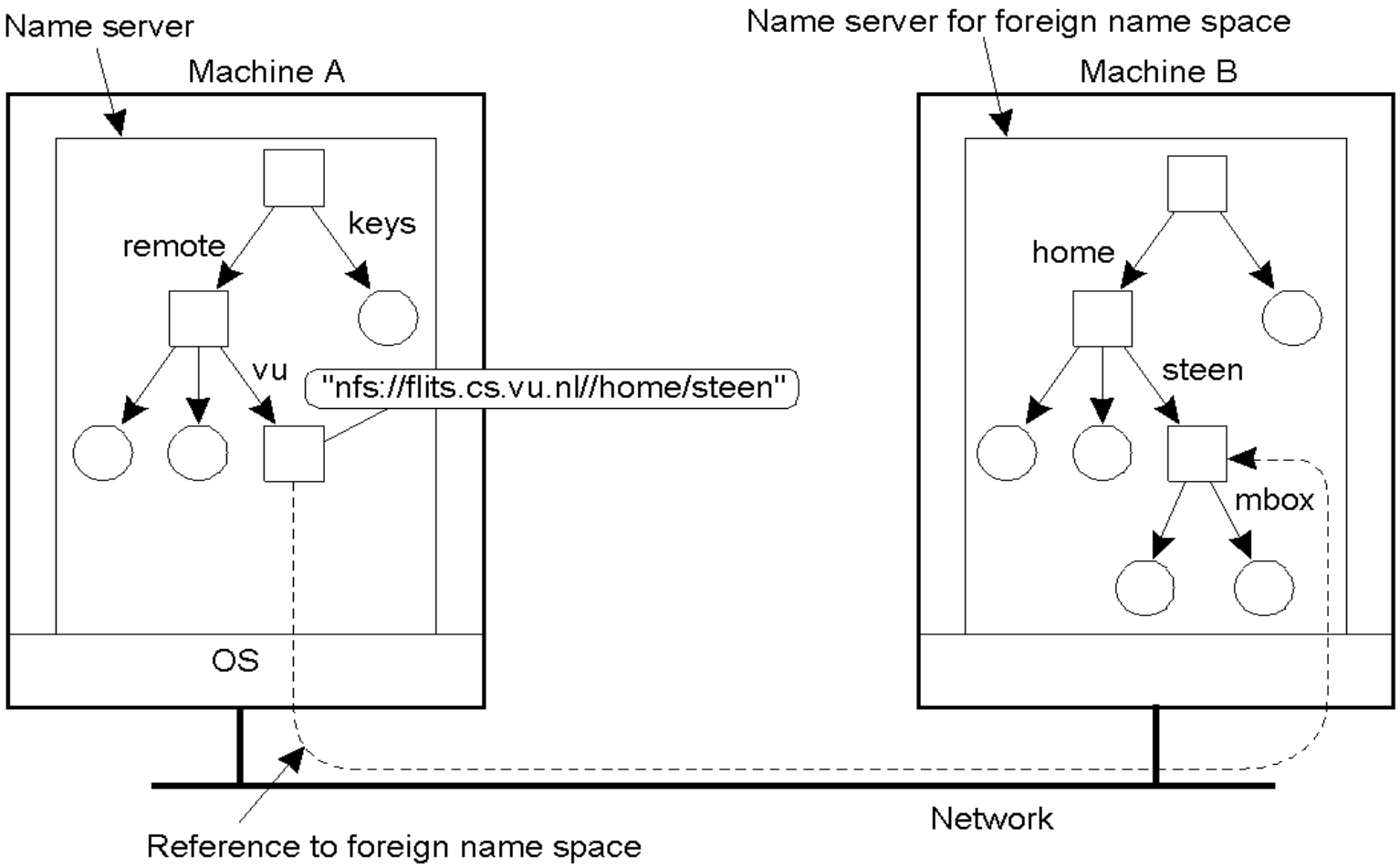
Other Name Space Examples

- UNIX file system implementation
  - With NFS enhancements to support “remote mounting” of remote file systems.
- SNMP MIB-II
  - “sub-namespace” within a much larger namespace maintained by the ISO.
- DNS (more on this later).

Introducing Name Resolution

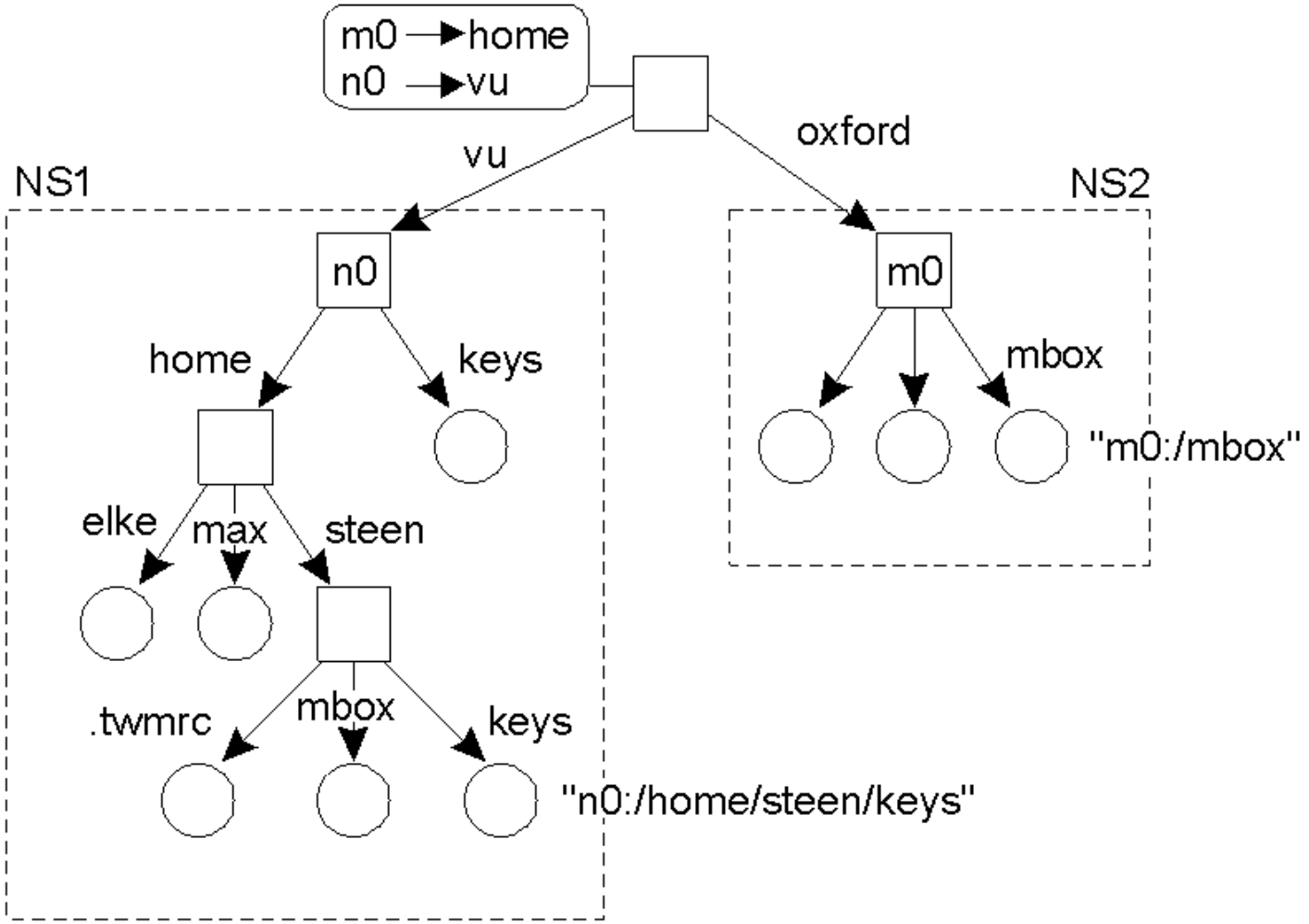
- The process of looking up information stored in the node given just the path name.
  - Assuming that you know where to start (bootstrap lookup address of naming service)…
- This can be complicated by techniques that have been devised to combine namespaces
  - such as Sun's NFS mounting and DEC's GNS…

Linking and Mounting (1)



Mounting remote name spaces through a specific process protocol (e.g. Sun's Network File System protocol - NFS).

Linking and Mounting (2)



Add a new root node and makes existing root nodes its children (e.g. organization of the DEC “Global Name Service”).

Implementing Namespaces

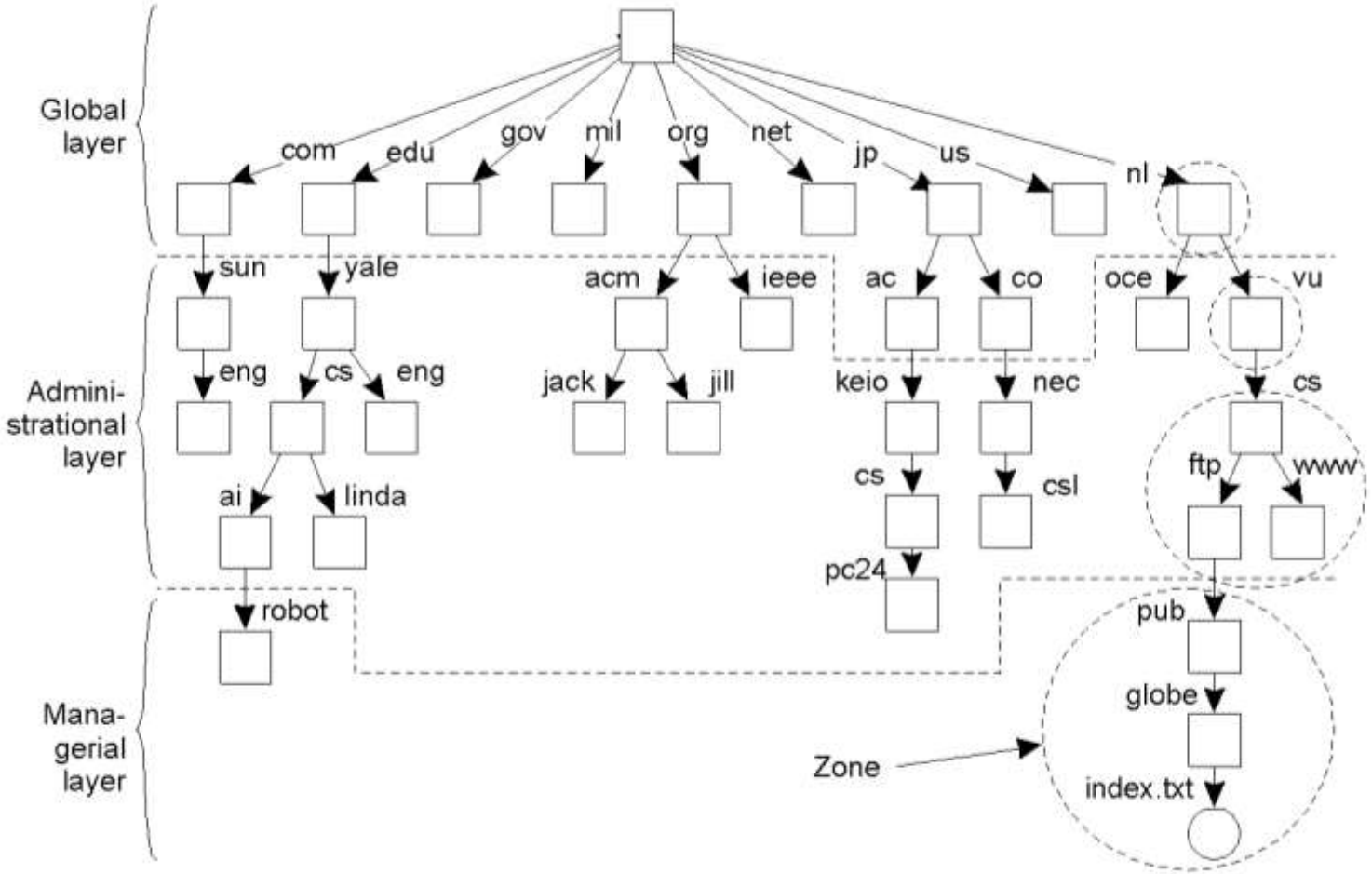
- ***Name Service*** allows users and processes to add, remove and lookup names.
  - Name services are implemented by *Name Servers*.
  - LAN's – a single server usually suffices (e.g. 'local' DNS).
  - WAN's – a distributed solution is often more practical (e.g. 'global' DNS).

- **Namespaces** (and services) are oftenly organised into one of three layers.

Three Name Space Layers

- **Global Layer**:
  - highest level nodes (root); stable; entries change very infrequently.
- **Administrational Layer**:
  - directory nodes managed by a single organisation; relatively stable; although changes can occur more frequently.
- **Managerial Layer**:
  - nodes change frequently; nodes maintained by users as well as administrators; nodes are the ‘leaf entities’, and can often change.

Name Space Distribution (1)



e.g. partitioning of the DNS name space, including Internet-accessible files, into the three name space layers. A "**zone**" in DNS is a non-overlapping part of the namespace that is implemented by a separate name server.

Name Space Distribution (2)

- Comparing features/characteristics of name servers that implement nodes within a large-scale name space (partitioned into global, administrational and managerial layers).
- **Availability** and **performance** requirements are met by replication and caching at each of the various layers (more on caching later).

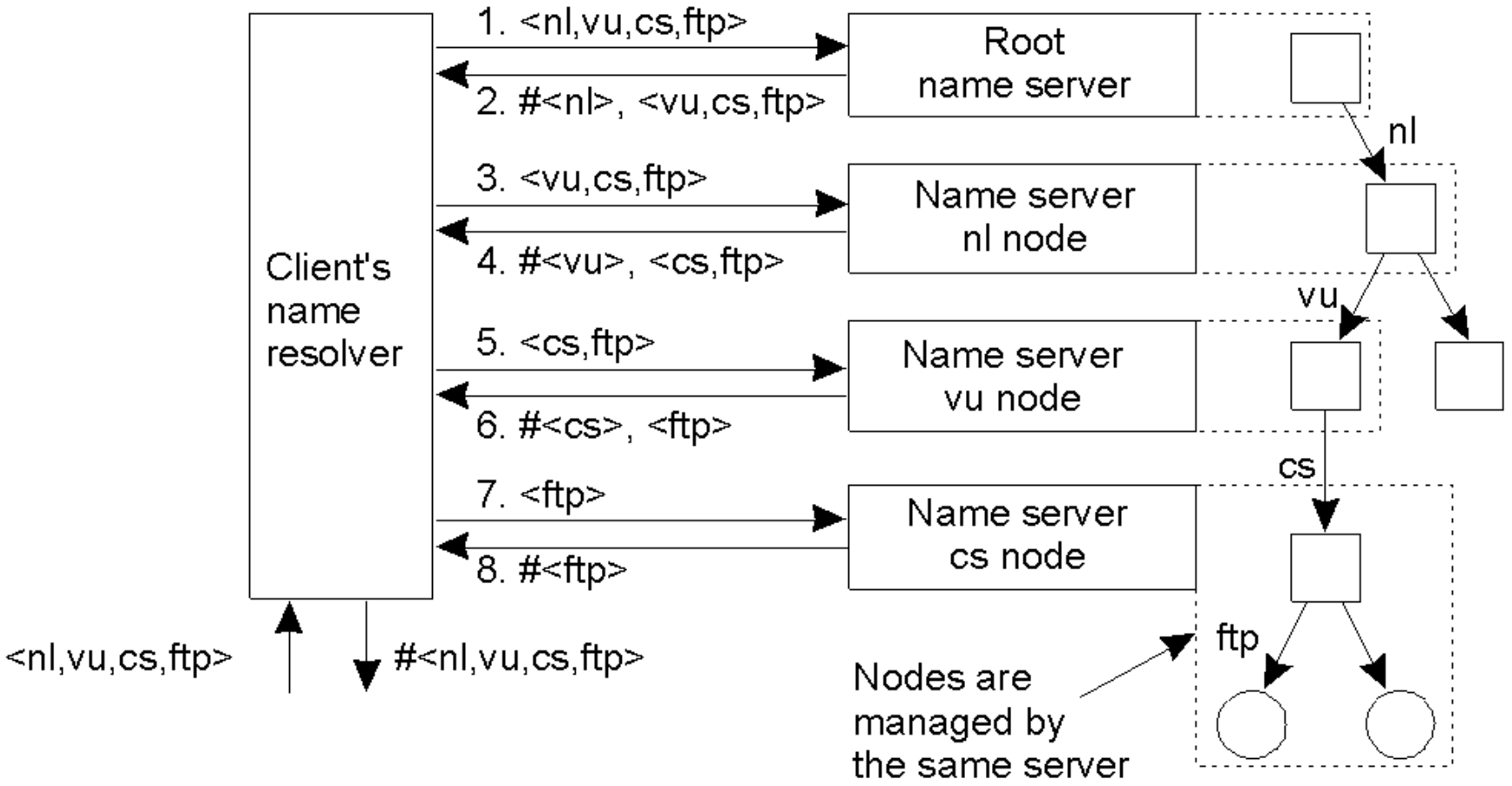
Feature/Characteristic	Global	Administrational	Managerial
Geographical **scale** of network	Worldwide	Organization	Department
Total **number** of nodes	Few	Many	Vast numbers
Responsiveness to lookups	Seconds	Milliseconds	Immediate
Update propagation	Lazy	Immediate	Immediate
Number of **replicas**	Many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

More on Name Resolution

- A “name resolver” provides a local name resolution service to clients
  - responsible for ensuring that the name resolution process is carried out.

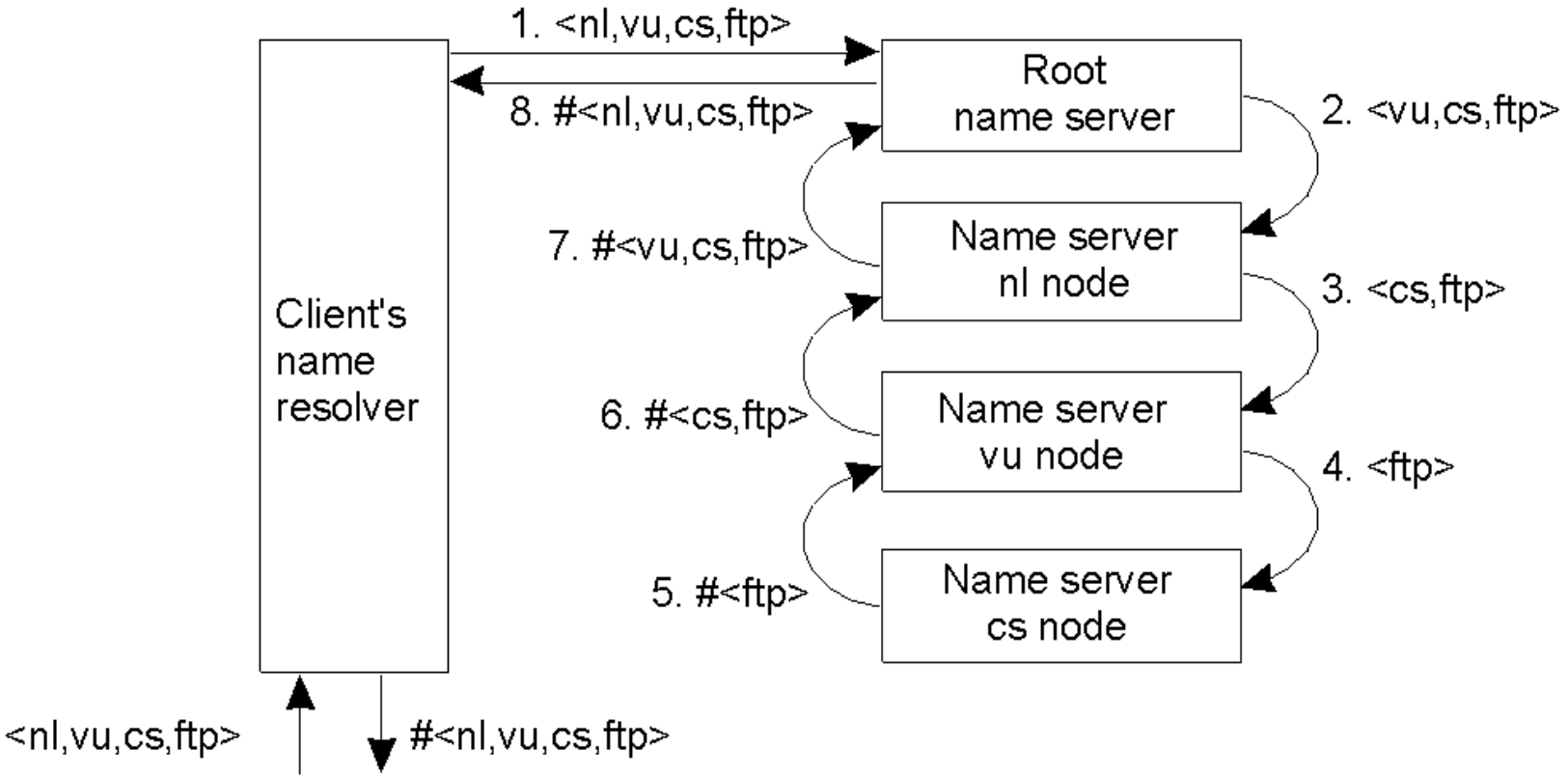
- Two Common Approaches:
  - 1. ***Iterative*** Name Resolution.
  - 2. ***Recursive*** Name Resolution.

Iterative Name Resolution



The **name resolver** queries each name server (at each layer) in an **iterative** fashion (NB: the client is doing all the work here and generating a lot of traffic).

Recursive Name Resolution



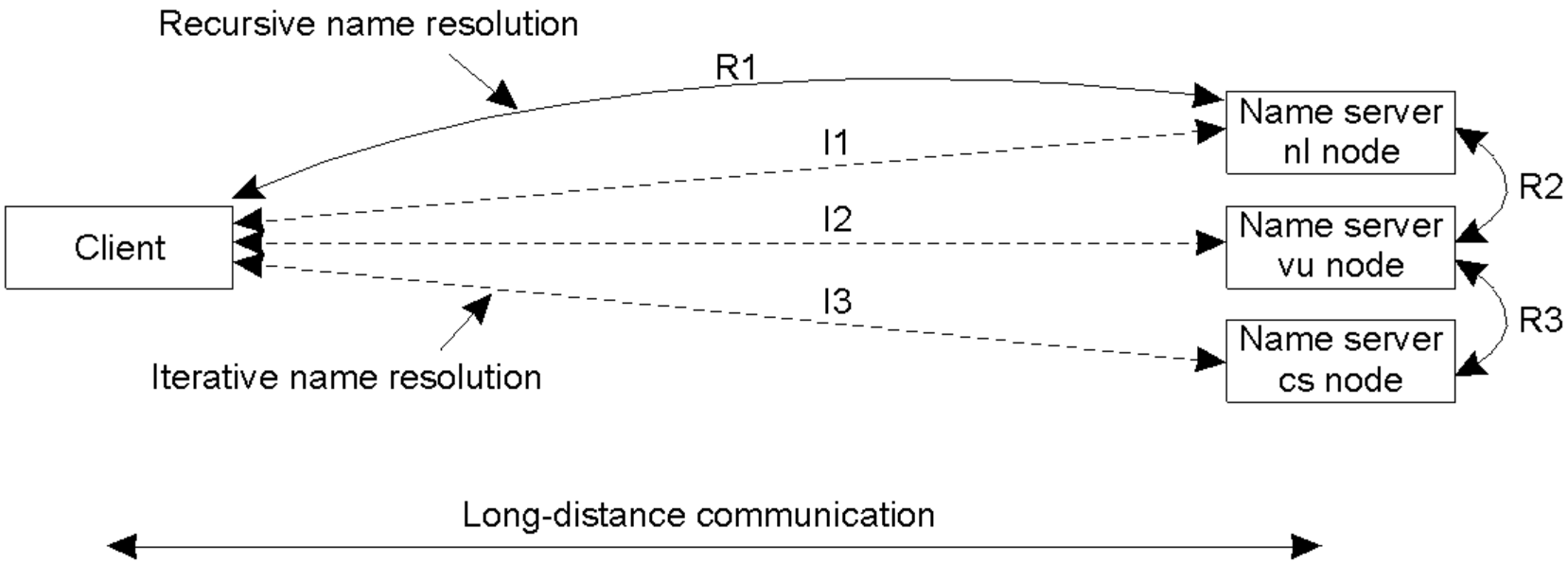
The **name resolver** starts the process, then each server **temporarily becomes a client** of the next name server until the resolution is satisfied (results are then returned to the client).

Caching and Recursive 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. This is seen as a *key advantage* to the recursive name resolution approach, even though the workload has been moved from the client to the servers. Nevertheless, think about subsequent lookups…

Iterative vs. Recursive Resolution



Comparison between recursive and iterative name resolution with respect to **communication costs**. Again, the recursive technology is generally regarded to have an advantage in this situation (especially over longer, more expensive WAN links).

Two Naming Examples

The Domain Name Service (DNS)

The X.500 Directory Service

Example: DNS

- One of the largest distributed naming services in use today.
  - DNS is a classic “rooted tree” naming system.
  - Each label (bit between ‘.’) must be < 64 chars.
  - Each path (the whole name) must be < 256 chars.

- The root is given the name ‘.’ (although, in practice, the dot is rarely shown nor required).

DNS Names

- A **subtree** within DNS is referred to as "**domain**".
- A **path name** is referred to as a "domain name".
  - These can be *relative* or *absolute*.
- A DNS server operates at each node (except those at the bottom).
  - Here, the information is organised into "resource records".

DNS – Types of Resource Record

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.

The most important types of resource records forming the contents of nodes (and maintained by servers) in the DNS name space.

DNS Implementation

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

The "database" is a small collection of **files** maintained within each DNS "zone".

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

Example: X.500 Naming Service

- A traditional naming service (like DNS) operates very much like the Telephone Directory.
  - *Find 'B', then find 'Barry', then find 'Paul', then get the number.*

- With a **directory service** (like X.500), the client can look for an entity based on a description of its properties instead of its full name (more like the Yellow Pages).
  - *Find 'Perl Consultants', obtain the list, search the list, find 'Paul Barry', then get the number.*

More on X.500

- Directory entries in X.500 are roughly equivalent to domain names in DNS.
  - The entries are organised as a series of "Attribute/Value Pairings"

- A collection of directory entries is referred to as a Directory Information Base (DIB).

X.500 Attribute/Value Pairings

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

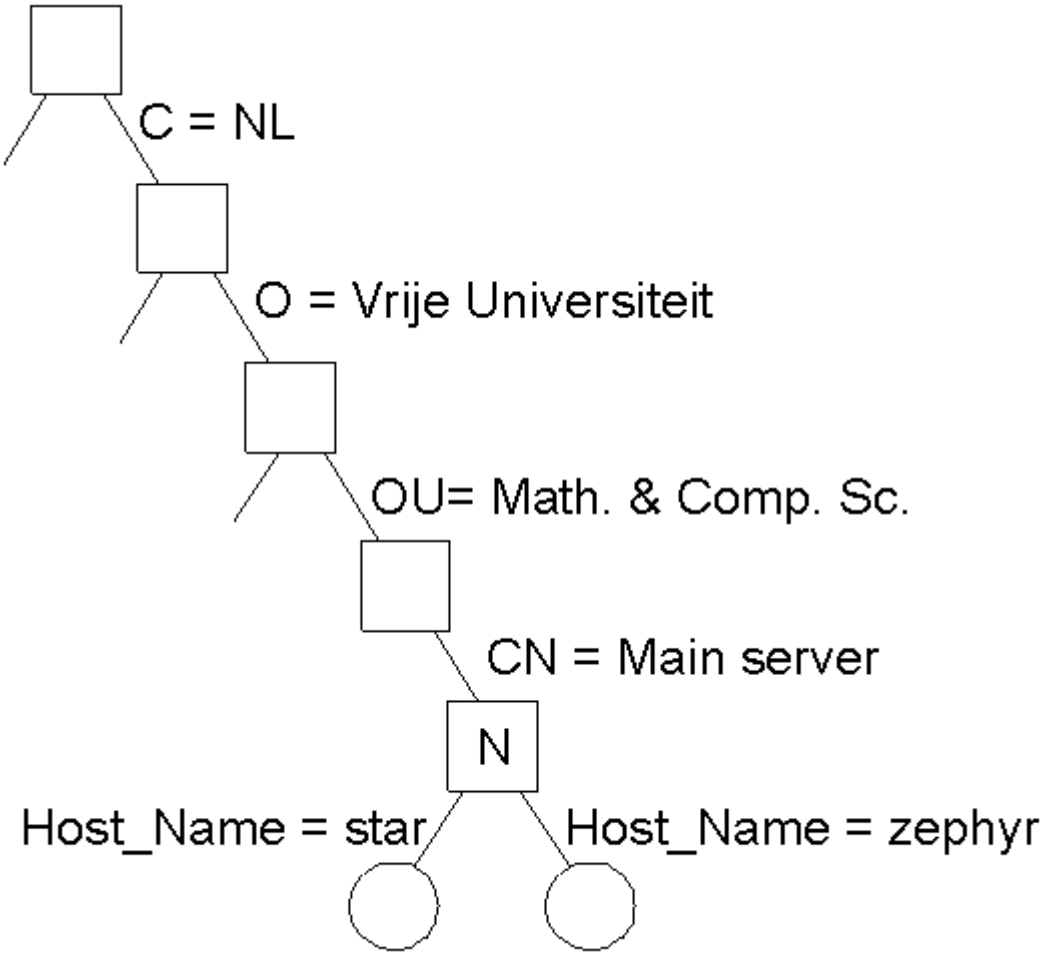
A simple example of a X.500 directory entry using X.500 naming conventions (NB: both Microsoft and Novell have based their name space technology on the X.500 standard).

X.500 RDN's and DIT's

- A collection of naming attributes is called a Relative Distinguished Name (RDN).
  - RDN's can be arranged in sequence into a Directory Information Tree (DIT).
  - The DIT is usually partitioned and distributed across several servers (called *Directory Service Agents – DSA*).
- Clients are known as *Directory User Agents – DUA*.

The X.500 DIT

Part of the X.500 Directory Information Tree (DIT)



X.500 Commentary

- Searching the DIT is an expensive task.
- Implementing X.500 is not trivial (as is the case with so many ISO standards).
- On the Internet, a similar service is provided by the simpler **Lightweight Directory Access Protocol** (**LDAP**)
  - regarded as a useful and implementable subset of the X.500 standards.

Locating Mobile Entities

Tricky …

- Traditional naming services (DNS, X.500) are not suited for environments where entities change location (i.e. move).
- The assumption is that moves occur rarely at the Global and Administrative layers, and when moves occur at the Managerial layer, the entity stays within the same domain.

But, what happens is `ftp.cs.vu.nl` moves to `ftp.cs.unisa.edu.au`?

Possible Solutions

1. A record of the new address of the entity is stored in the cs.vu.nl name server.
2. A record of the name of the new entity is stored in the cs.vu.nl name server (i.e. a *symbolic link* is created).

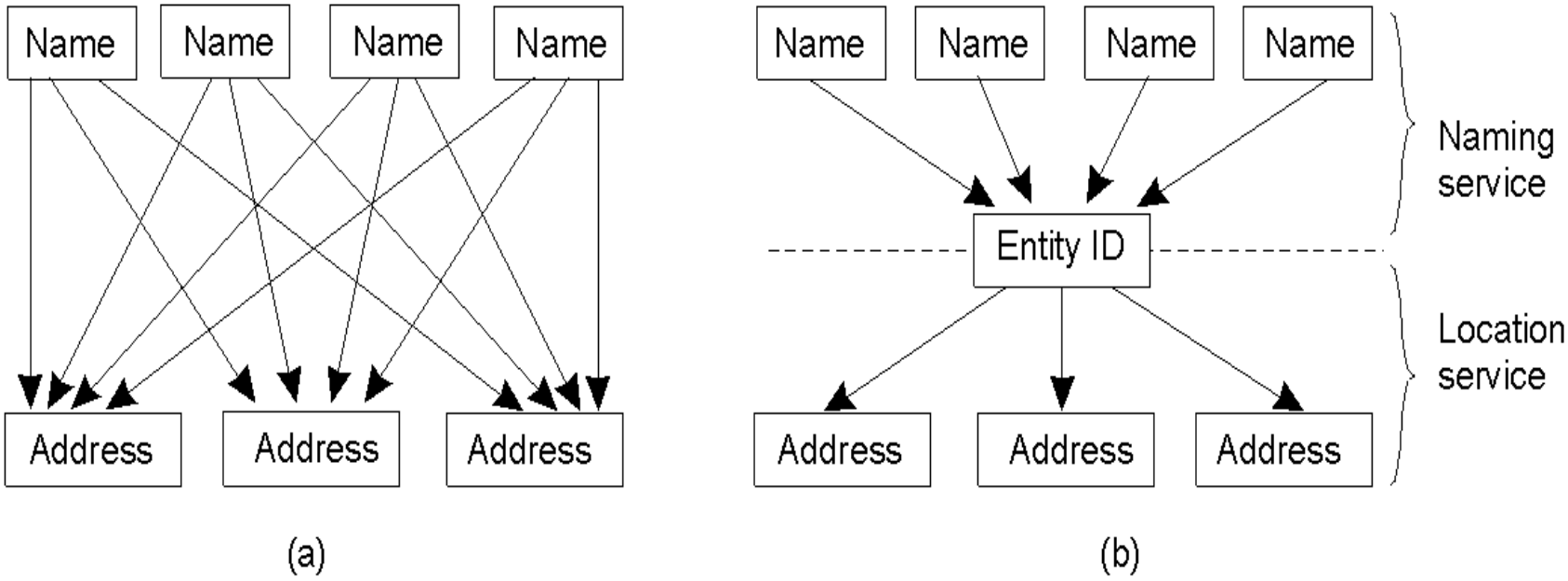
- Both “solutions” seem OK, until you consider what happens when the entity moves again, then again, then again…
- Consequently, both “solutions” can be shown to be ***inefficient*** and ***unscalable***.

More Location Problems

- Even non-mobile entities that change their name often cause name space problems – consider the DNS within a DHCP environment (currently incompatible).
  - So… a different solution is needed.

- What's required is a "Location Service" (or middle-man technology).

Naming vs. Location Services



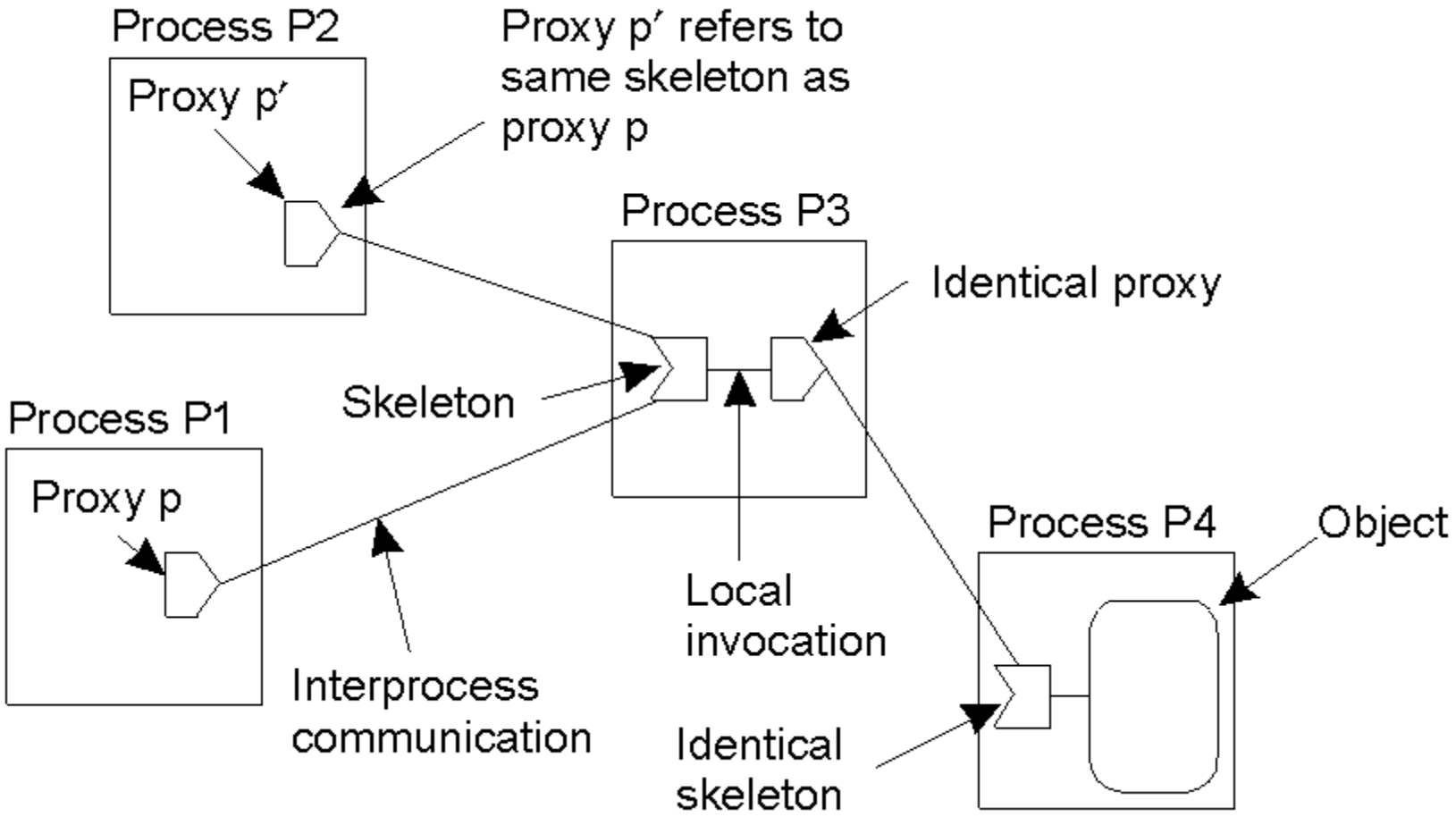
a) Direct, single-level mapping between names and addresses.

b) Two-level mapping using a "location service".

Simple Solution #1

- **Broadcasting** and **Multicasting** technologies.
  - Sending out “where are you?” packets …
  - Address Resolution Protocol (ARP) used by the TCP/IP suite for resolving IP names to underlying networking technology addresses.
  - Works well (on LAN’s and other broadcast technologies) but doesn’t scale well.

Simple Solution #2: Forwarding Pointers



Principle of **forwarding pointers** using (*proxy*, *skeleton*) pairs: after each relocation, the process leaves a pointer to where it moved to next. Simple to implement, but has a number of disadvantages.

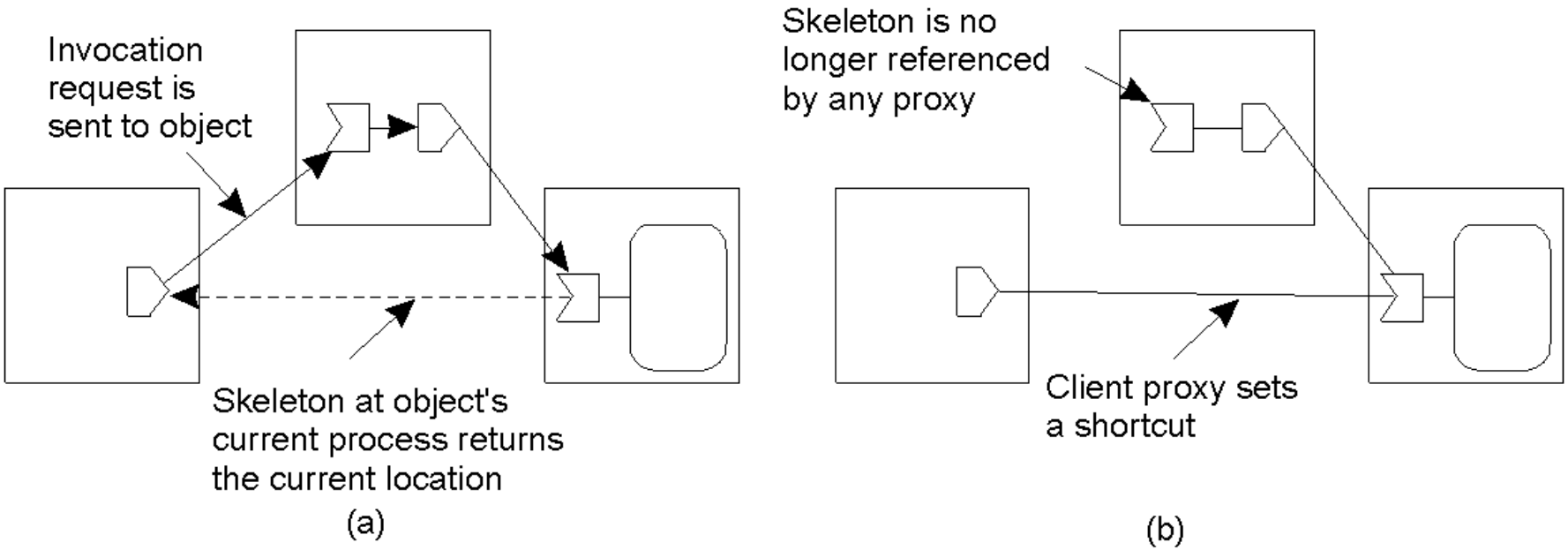
Disadvantages of Forwarding Pointers

1. A chain can become very long, and the "lookup" eventually becomes prohibitively expensive.

2. All the "intermediate locations" must maintain their chains for "as long as needed" (however long that is).

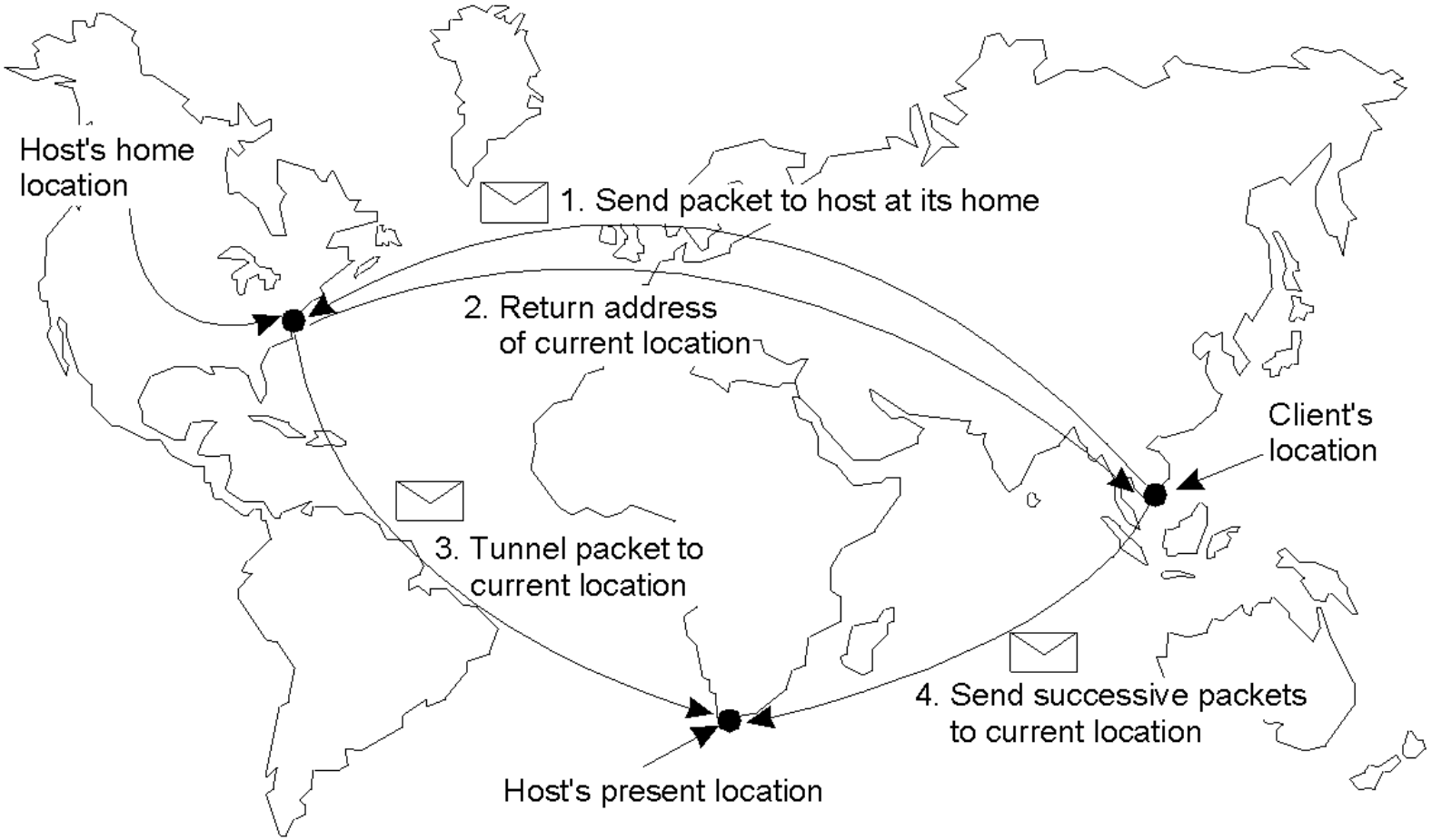
3. Big vulnerability: *broken links (*break a link and a forwarded entity is lost… oh, dear).

Simple Solution #2, cont.



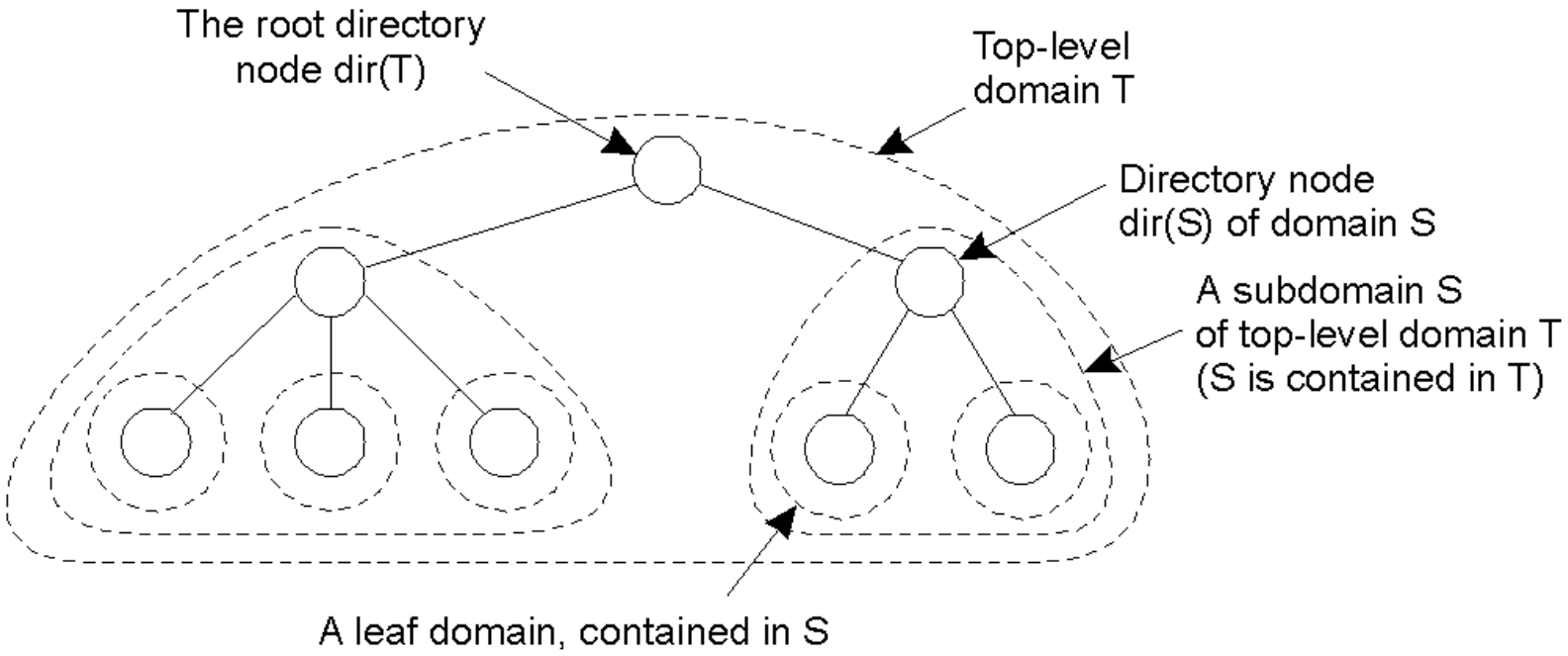
- Somewhat of an improvement: redirecting a forwarding pointer, by storing a shortcut in a proxy. However, to avoid large chains of pointers, it is important to reduce chains at regular intervals (easier said than done).
- Of course, the more pointers there are, the more latency problems there are.
- And… this solution does NOT scale well.

Solution #3: Home-Based Approaches



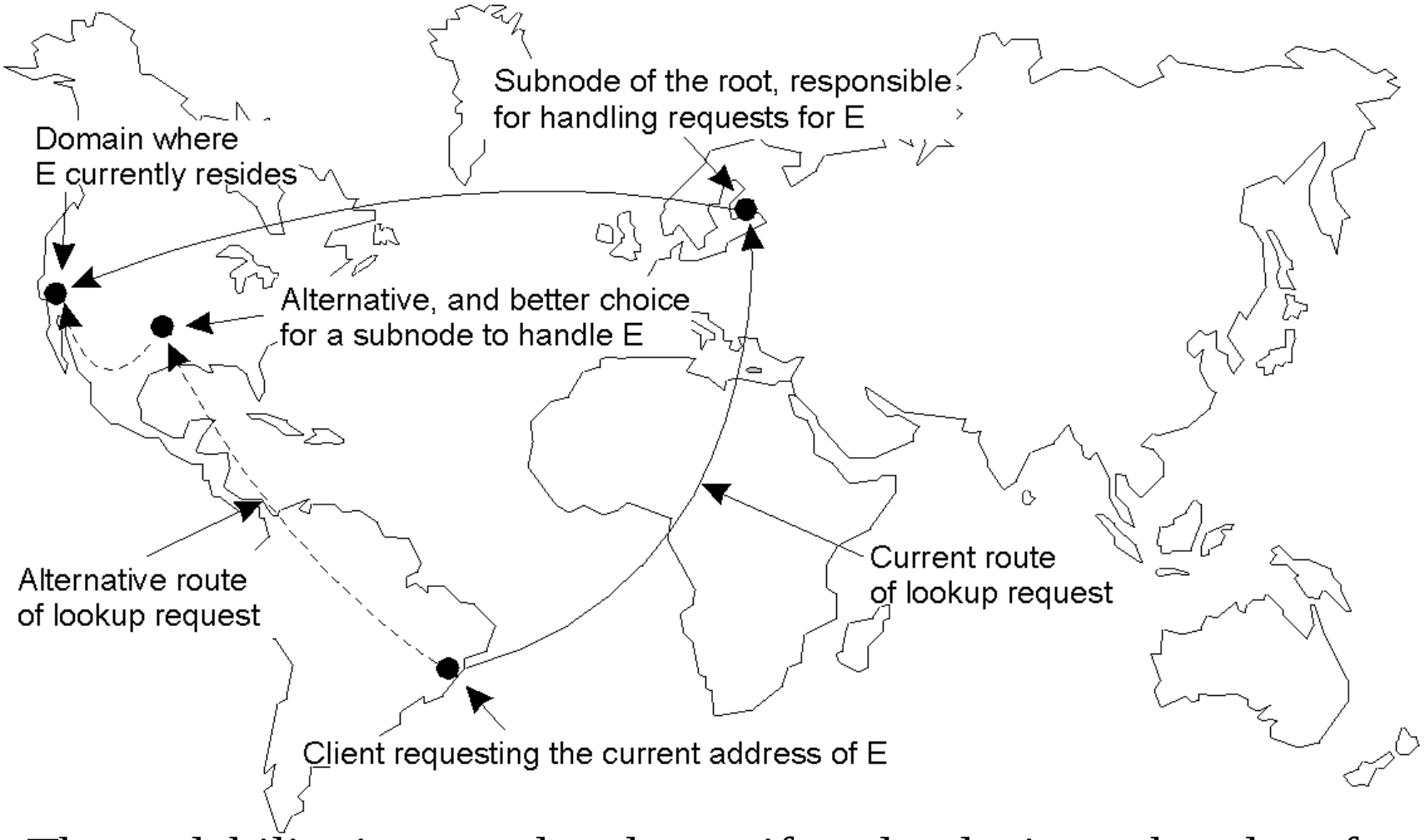
- An entity has a "**home**" which can be contacted in order to determine the mobile entities *current* location. This is the principle employed by the **Mobile IP** technologies (with its "home agents" and "care-of addresses").
- **Drawbacks**: increased latency and permanent moves.

Solution #4: Hierarchical Approaches



- **Hierarchical** organization of a **location service** into domains, each having an associated directory node – it can be useful to think of this as a "dynamic" name space.

Scalability Issues with the Hierarchy



- The scalability issues related to uniformly placing subnodes of a partitioned root node across the network covered by a location service.

Distributed Garbage Collection

- Removing unreferenced entities can be *tricky*.
- As soon as a entity is no longer required, it (and any copies of it and/or references/pointers to it) needs to be removed from the distributed system.
  - e.g, just look at the mess of unreferenced HTML documents ("broken links") on today's Internet
  - [As an aside: part of the XML technology hopes to fix this problem… the jury is still out on this one].

Removing Unreferenced Entities

- Managing the removal of entities in a distributed system is often difficult.
  - Consider: is every reference to an entity an intention to access it at some later date?
- It is not acceptable to *never* remove an entity – all garbage needs to be collected.
  - Consequently, a number of **Distributed Garbage Collection** mechanisms have been devised.

What's the Problem?

- Simple: an unreferenced entity is no longer needed and should be removed from the DS.
- A sick twist: a reference to an object which references another object, which in turn references another object, which references the first object (forming a "cycle") needs to be detected and removed.
- Garbage collection is well understood in uniprocessor systems and easily implemented. Things are considerable more complex when it comes to DSes.

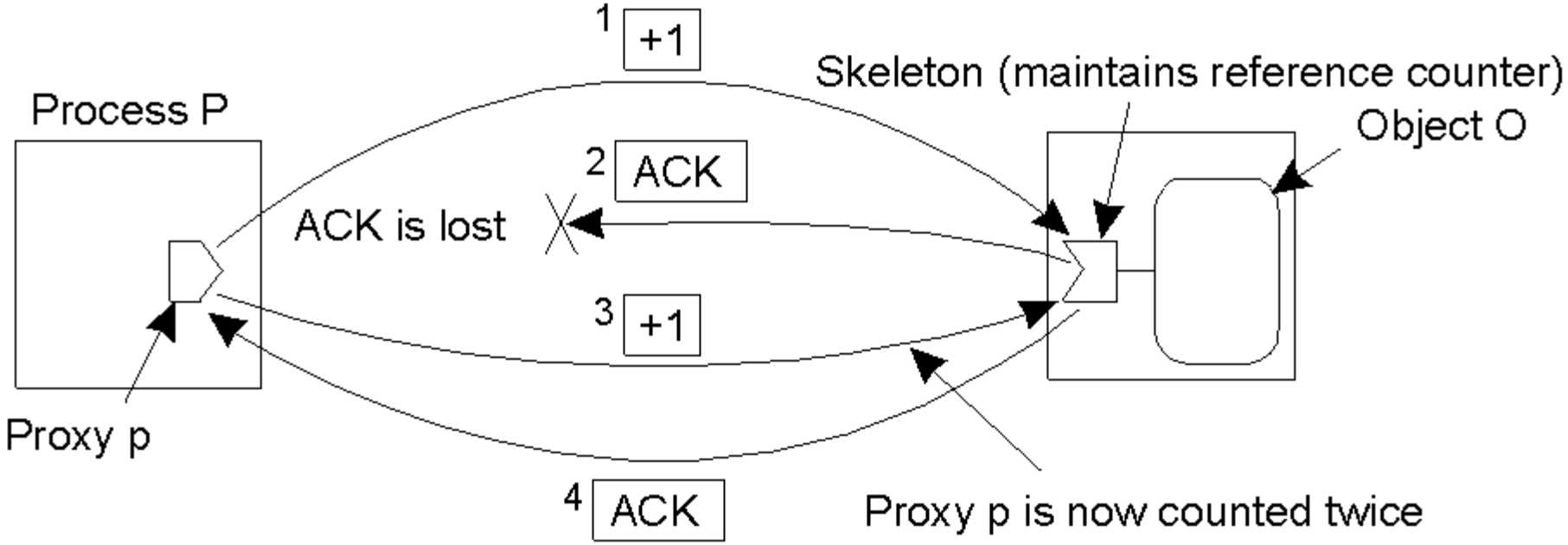
Critical Questions

- What type of communication is required to maintain references and perform distributed garbage collection?
- What happens when the communications system is subject to process failures and errors?

  - A number of solutions are proposed.
  - Unfortunately, each only solves a *part of the problem*.

Generic Solution: Reference Counting

- Increment at counter when an object is referenced.
- Decrement a counter when an object reference is no longer needed.
- Delete the object when the reference count is zero.
- Leads to a number of problems, mainly due to unreliable communications systems.



Adding Robustness

- Lost acknowledgements are easy to detect and deal with (a problem that has been solved by many other networking technologies).
- Duplicates can also be handled.
- A number of reliable enhancements to simple reference counting exist, but suffer from performance and scalability problems (they are also complex):
  - Weighted Reference Counting
  - Generation Reference Counting

Enhancements to Counting

- **Reference Listing**: a reference count is not maintained. Instead, as list of proxies that point to the object is maintained by the object.
  - The list has some important properties: if a proxy is already in the list, adding it again *does not change the list*. Also, if a proxy is not in the list, removing it from the list *does not change the list*.
  - Reference Listing is said to be "idempotent" (an operation can be repeated any number of times without affecting the end result). So a proxy can keep adding/removing itself from the list until an ACK is returned.
- **Key point**: duplicates are OK, and reliable communications is NOT required.

Think About This …

Increment and Decrement are *not* idempotent.

More on Enhancements

- Reference Listing is used by Java's RMI.
  - The object keeps track of those remote processes that current have proxies to it.
- Big disadvantage (with all Reference Listing systems): they scale poorly when there's many references to the list.
- Alternative: **Reference Tracing.**
  - Keeps track of *every* object in the distributed system.
  - A fine idea, but inherently unscalable (and a bit complex, too).

Naming: Summary

- Names refer to entities, which are organised into name-spaces.
  - *Address*: an entities access point.
  - *Identifier:* one-to-one mapping to an entity.
  - *Name*: human friendly descriptor.
- Traditional naming systems include
  - DNS and X.500.
- Neither are suited to distributed systems which must support mobile entities.

Naming: Summary, continued.

- Four approaches to finding/naming mobile entities:
  - Broadcasting/multicasting: only works on LAN's.
  - Forwarding pointers: large chains cause problems.
  - Home based systems: e.g. Mobile-IP.
  - Hierarchical, dymanic domains.
- Removal of "no longer needed" entities is important.
- Distributed systems garbage collection technologies are organised around:
  - Simple reference counting systems.
  - Reference tracing.
  - Reference Lists.
    - All have their advantages/disadvantages.

RESEARCH CONTINUES…