# Naming

Chapter 4

## Naming (1)

- Name resolution allows a process to access a named entity.
- A naming system is necessary.
- In a distributed system the naming system is distributed.

## Naming (2)

- In a distributed system
- A **name** is a string.
- An **entity** is a generic resource.

- An **access point** is a special entity.
- A name of an access point is called **address**.

- Ex: Phone - number,    channel - frequency

## Naming (3)

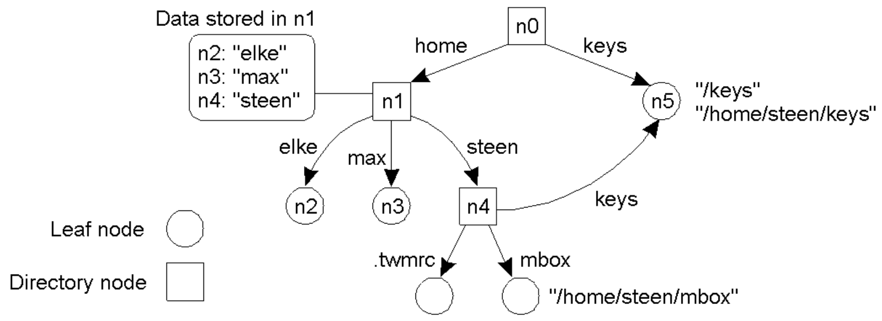
- An entity may have more than one access point.

- An entity name can be **location independent**.

- When
  - a name refers to at most one entity,
  - each entity is referred to by at most one name,
  - a name always refers to the same entity (no reuse)

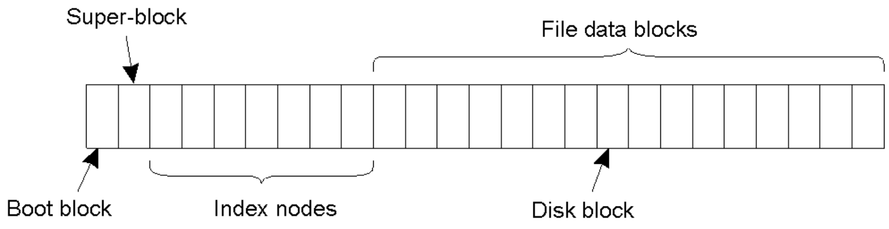
  the name is called an **identifier**.

## Name Spaces (1)



A general naming graph with a single root node.

## Name Spaces (2)



The general organization of the UNIX file system implementation on a logical disk of contiguous disk blocks.

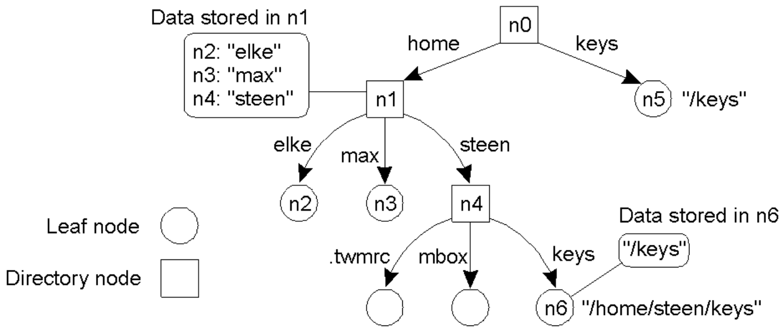
## Name resolution

- Distributed name resolution:

  *N:<l1, l2, …, ln>*

- Closure mechanism: knowing from where to start name resolution.
- Aliases:
  – hard links
  – symbolic links.

## Linking and Mounting (1)



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

## Linking and Mounting (2)

For mounting a remote name space in a distributed system it is necessary to resolve:

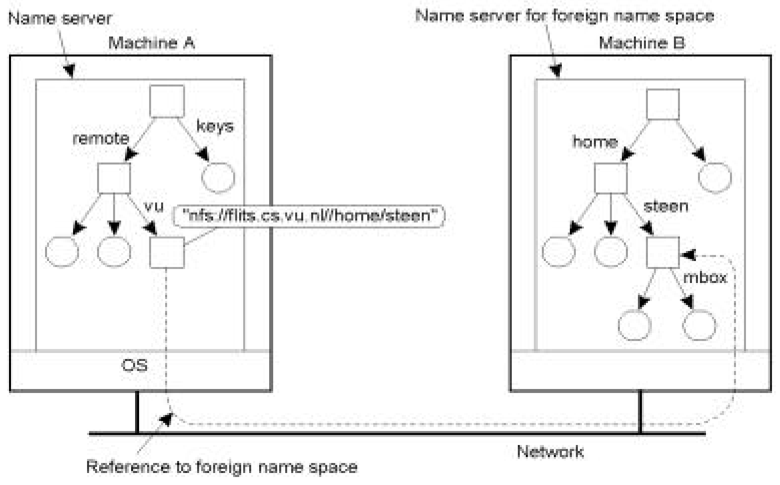
•.the name of the access protocol,

• the name of the server,

• the name of the mounting point in the remote name space.

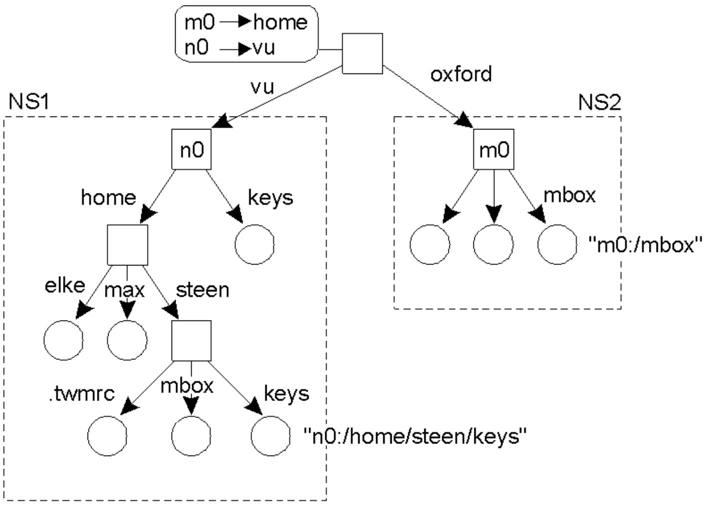
Ex: nfs://flits.cs.vu.nl/home/steen

## 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

Mounting remote name spaces through a specific process protocol.

# Linking and Mounting (3)



Organization of the DEC Global Name Service

# Distributed Name Space

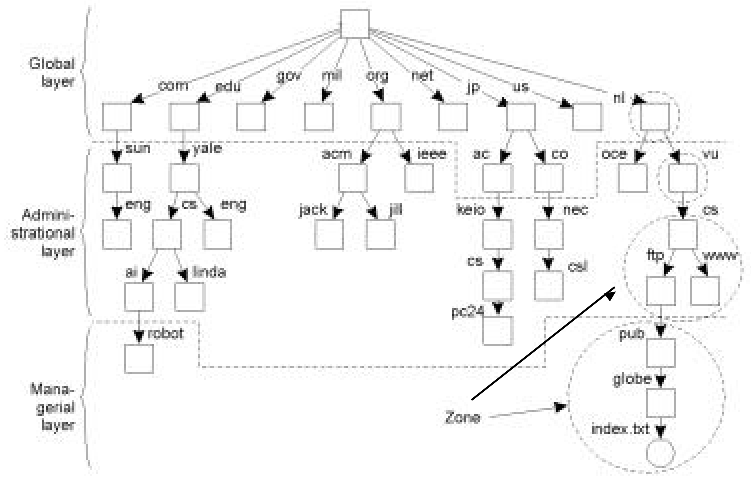
Large distributed systems use hierarchical name servers.

Name server replication may be helpful.

Name spaces can be partitioned in logical layers:

- global layer,
- administration layer,
- managerial layer.

## Name Space Distribution (1)



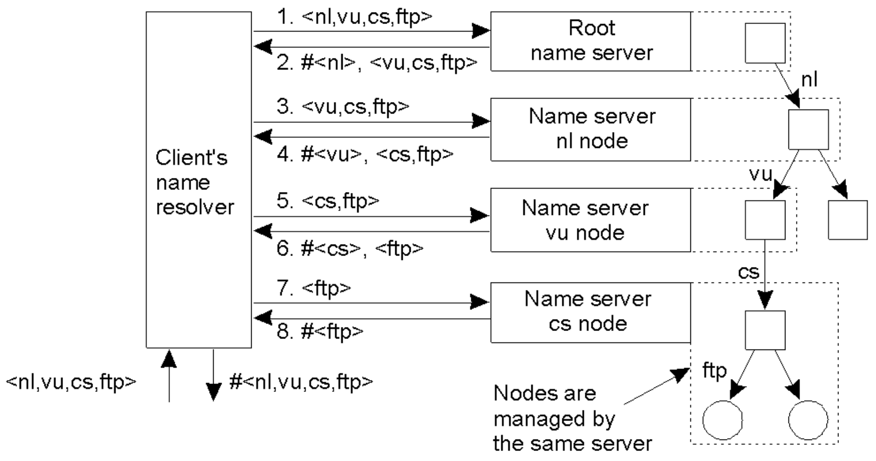
An example partitioning of the DNS name space, including Internet-accessible files, into three layers.

## 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 |

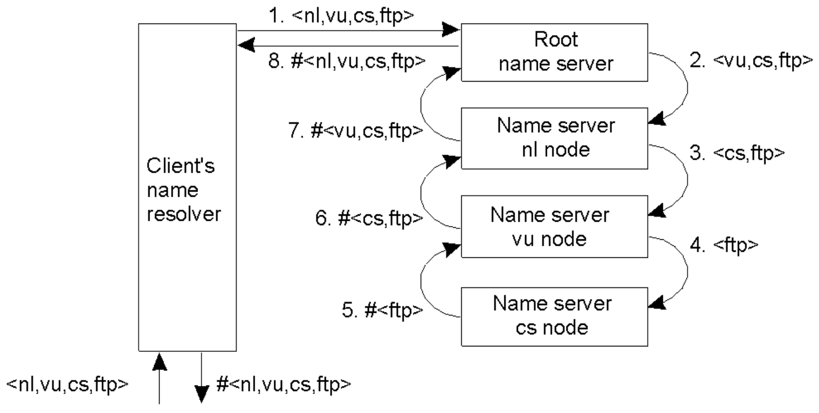
A comparison between name servers for implementing nodes from a large-scale name space partitioned into a global layer, as an administrational layer, and a managerial layer.

## Iterative Name Resolution



The principle of iterative name resolution.

## Recursive Name Resolution



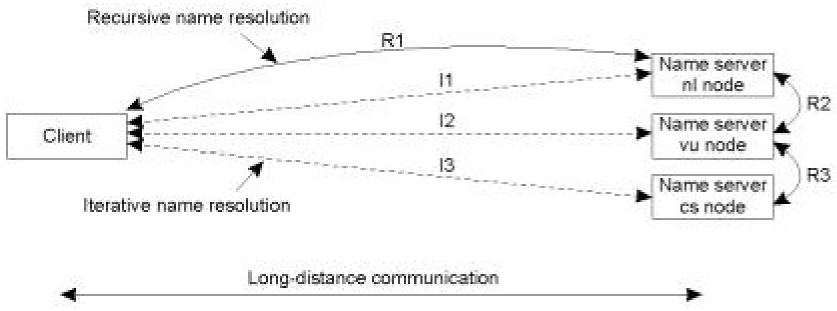
The principle of recursive name resolution.

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

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

## Implementation of Name Resolution


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

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

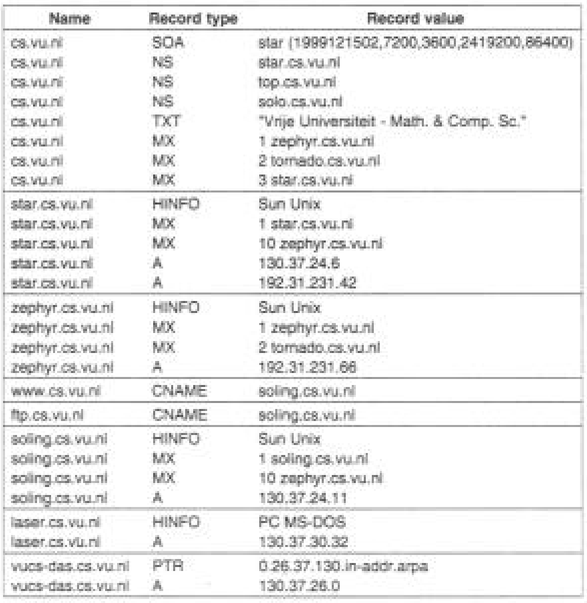
## The DNS Name Space

| 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 |

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

## 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 |

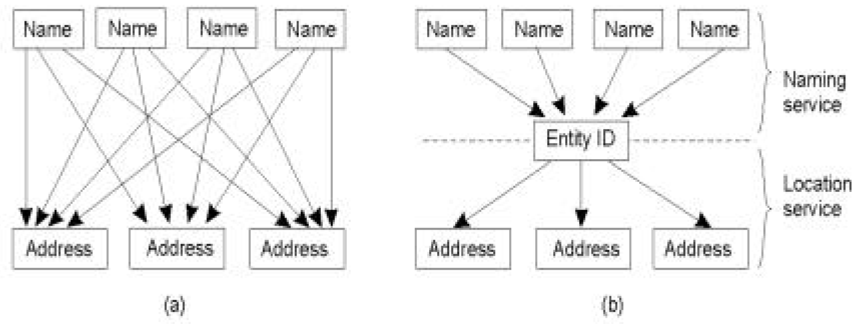
Part of the description for the *vu.nl* domain which contains the *cs.vu.nl* domain.

## Naming versus Locating Entities

a) How to handle the moving of servers in different domains ?
   a) Record the address of the new machine in the DNS entr of the old machine.
   b) Record the name of the new machine in the DNS entr of the old machine.

b) A multi-step look up is needed.

## Naming versus Locating Entities

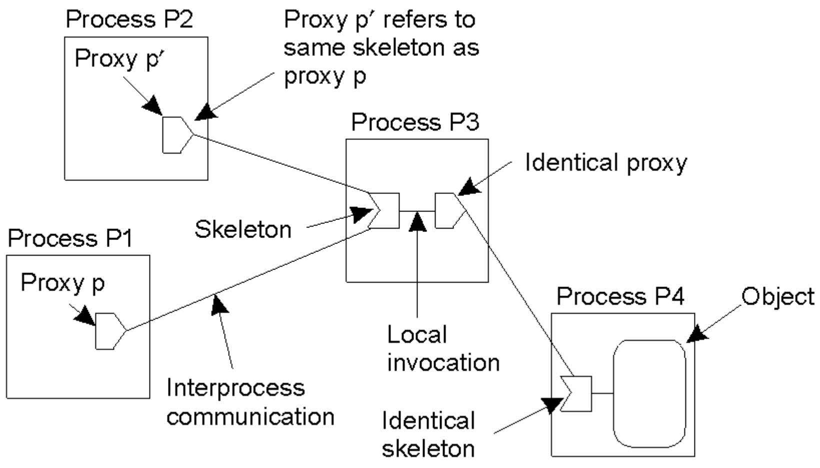


- Direct, single level mapping between names and addresses.
- Two-level mapping using identities.

## Broadcasting and Multicasting

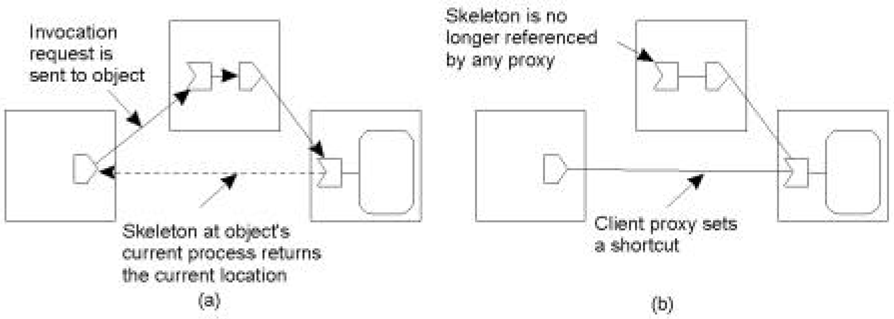
- In a LAN with a few nodes broadcasting can be used.
  - An entity id is sent to each machine asking it to check for the entity owner.
- When the number of nodes if large multicasting can be used.

## Forwarding Pointers (1)



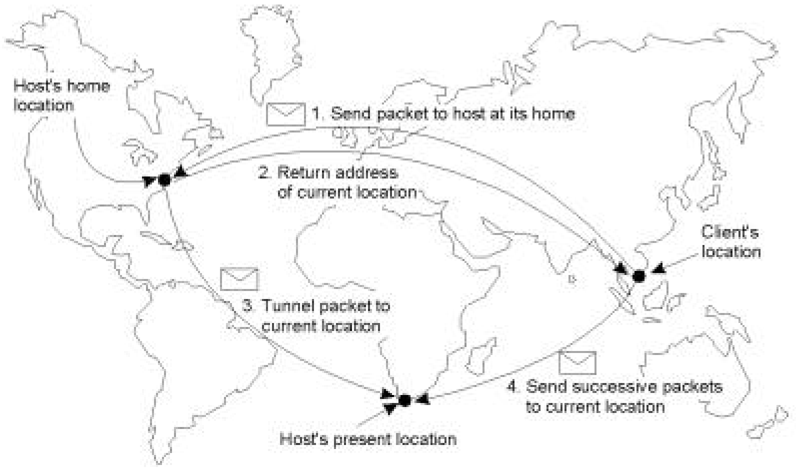
The principle of forwarding pointers using (*proxy*, *skeleton*) pairs.

## Forwarding Pointers (2)


Invocation request is sent to object
Skeleton at object's current process returns the current location
(a)
Skeleton is no longer referenced by any proxy
Client proxy sets a shortcut
(b)

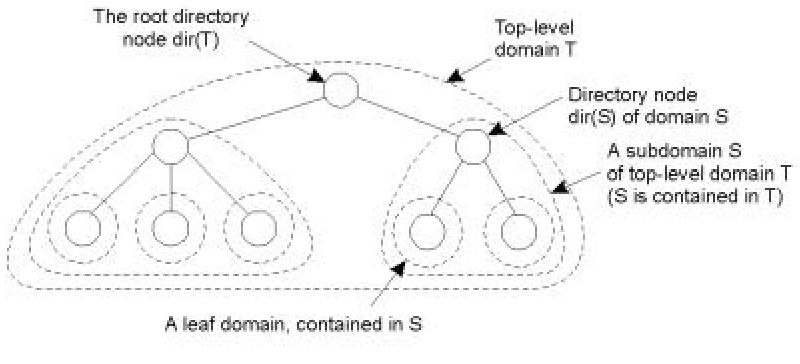
Redirecting a forwarding pointer, by storing a shortcut in a proxy.

## Home-Based Approaches


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

The principle of Mobile IP.

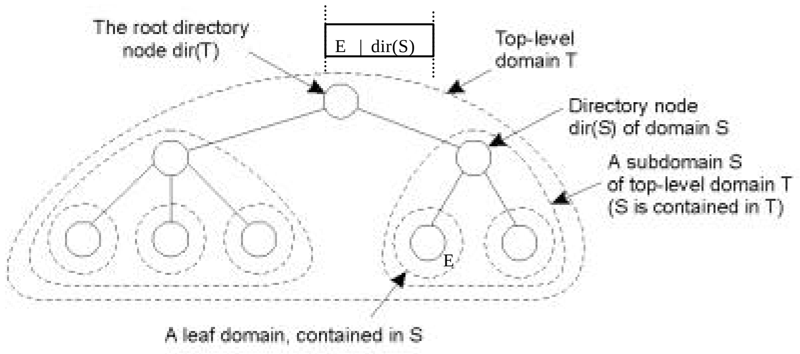
## Hierarchical Approaches (1)


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

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

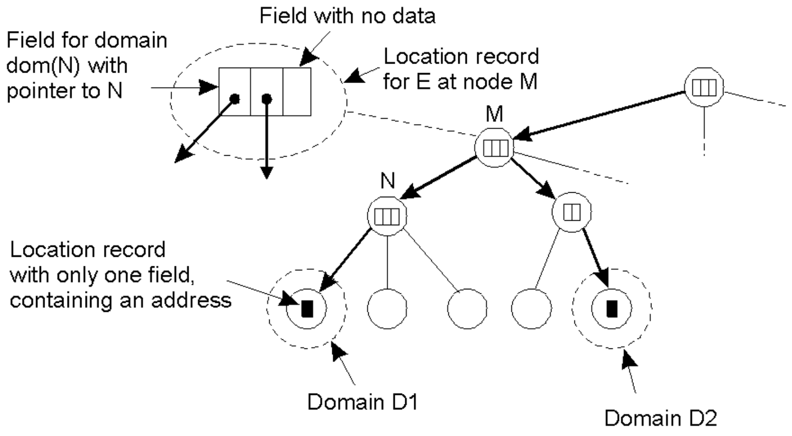
An entity in **D** is identified by a location record in **dir(D)**.

## Hierarchical Approaches (2)


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

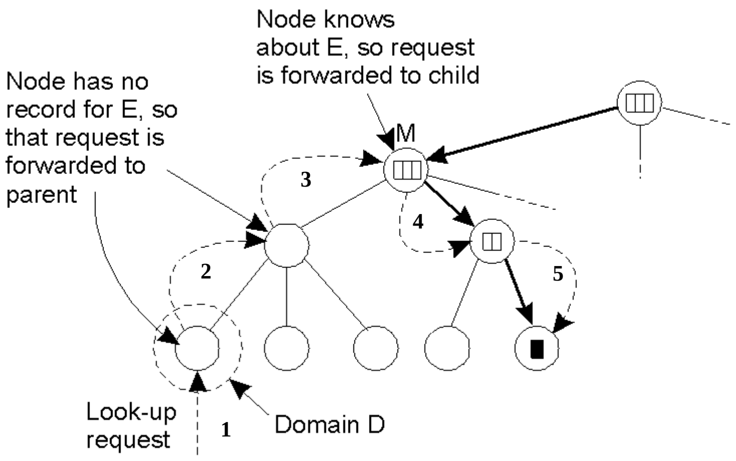
A sub-tree root node contains an entry for each entity. The location record contains a pointer to the directory node of the next lower-level sub-domain.

## Hierarchical Approaches (3)


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

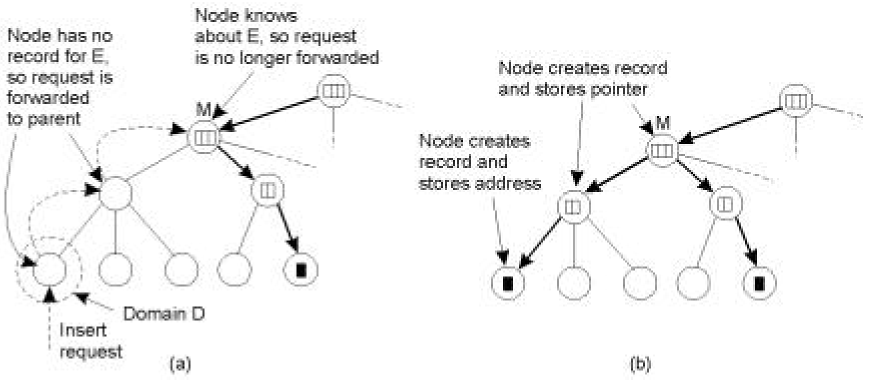
An example of storing information of a replicated entity having two addresses in different leaf domains.

## Hierarchical Approaches (4)



Looking up a location in a hierarchically organized location service.

## Hierarchical Approaches (5)



a) An **insert** request is forwarded to the first node that knows about entity *E*.

b) A chain of forwarding pointers to the leaf node is created.

## Scalability Issues

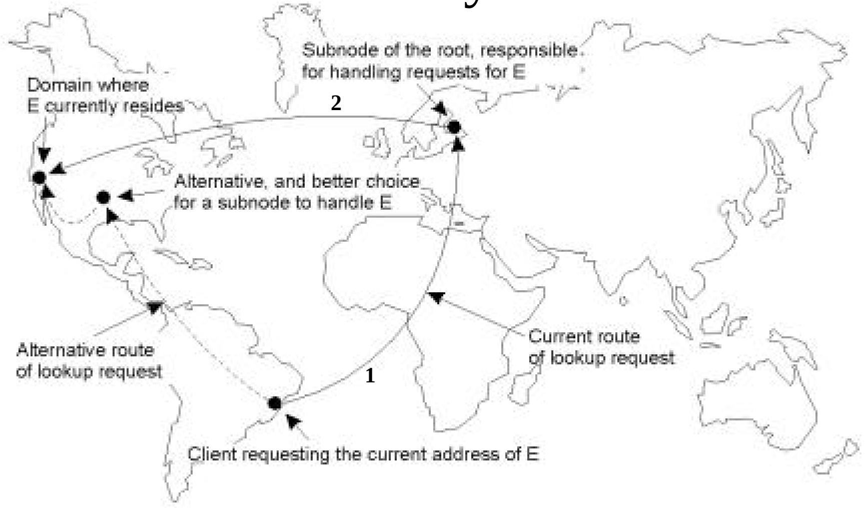
In hierarchical location services the root node must store entries for all of the entities.

The root node can be the system bottleneck.

It can be partitioned in a set of nodes that handle a subset of entities.

Finding the best way to locate the nodes is a challenging issue.

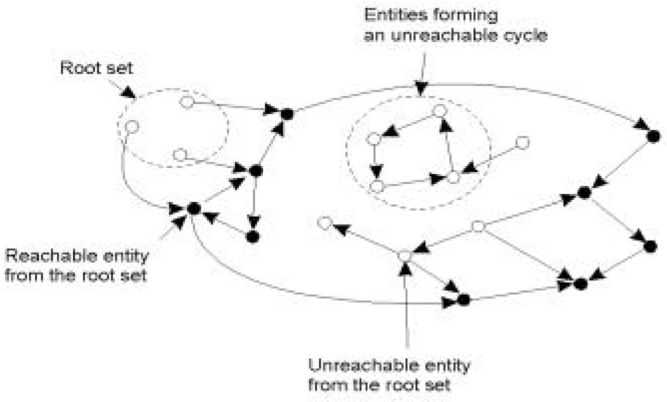
## Scalability Issues


Subnode of the root, responsible for handling requests for E
Domain where E currently resides
2
Alternative, and better choice for a subnode to handle E
Alternative route of lookup request
Current route of lookup request
1
Client requesting the current address of E

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

## The Problem of Unreferenced Objects

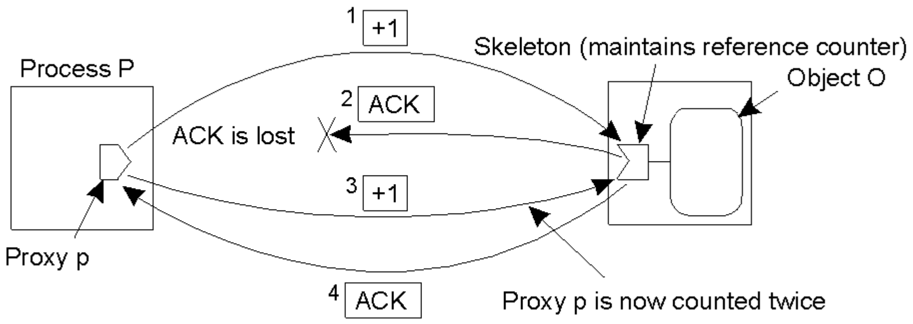
Solution: distributed garbage collector.



Entities forming an unreachable cycle
Root set
Reachable entity from the root set
Unreachable entity from the root set

An example of a graph representing objects containing references to each other. White nodes should be removed.

## Reference Counting (1)



1 +1
Skeleton (maintains reference counter)
Process P
Object O
2 ACK
ACK is lost
3 +1
Proxy p
4 ACK
Proxy p is now counted twice

The problem of maintaining a proper reference count in the presence of unreliable communication: *detect duplicate messages*.