Naming

Distributed Systems [5]

Naming Entities

- Names, identifiers, and addresses
- Name resolution
- Name space implementation

Naming Entities

- 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.

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

Three Kinds of Naming

- Human-friendly name: f1, cs.nju.edu.cn
  - may refer to several identities
- Address: name of an access point, entity's address
  202.119.32.6
  - May be moved;
  - An entity can offer more than one access points.
- Identifier: a name has properties:
  - an identifier refers to at most one entity
  - each entity is referred to by at most one identifier
  - an identifier is never reused

    Ethernet address?

Requirements on Name Services

- Usage of unique naming conventions
  - Enables sharing
  - It is often not easily predictable which services will eventually share resources
- Scalability
  - Naming directories tend to grow very fast, in particular in Internet
- Consistency
  - Short and mid-term inconsistencies tolerable
  - In the long term, system should converge towards a consistent state
- Performance and availability
  - Speed and availability of lookup operations
  - Name services are at the heart of many distributed applications
- Adaptability to change
  - Organizations frequently change structure during lifetime
- Fault isolation
  - System should tolerate failure of some of its servers

Identifiers

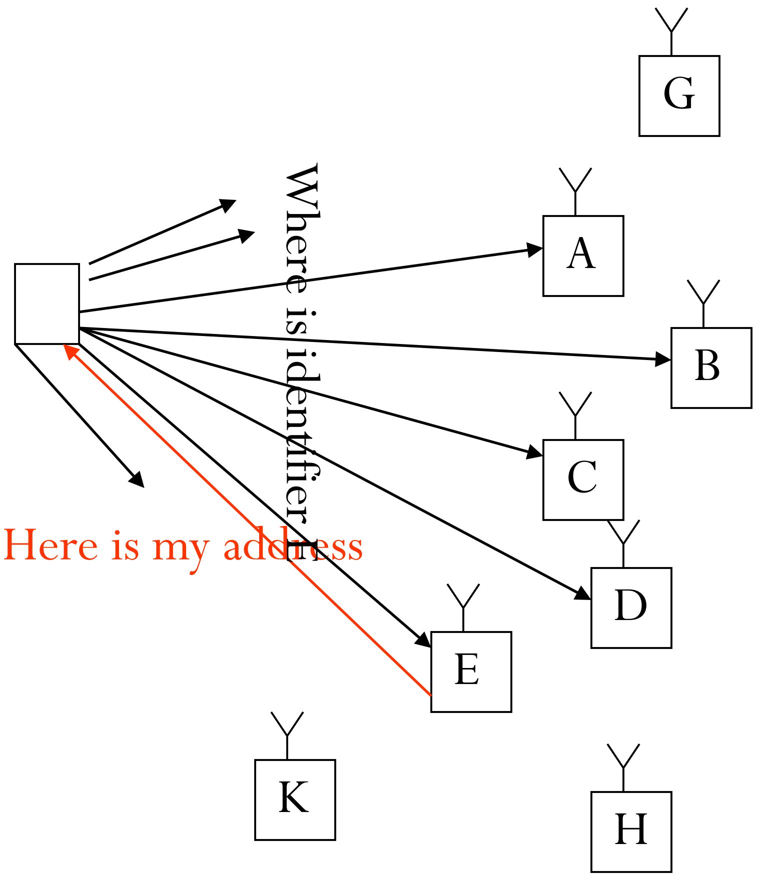
- A name that has no meaning at all; it is just a random string. Pure names can be used for comparison only.
- A name having the following properties:
  - P1: Each identifier refers to at most one entity
  - P2: Each entity is referred to by at most one identifier
  - P3: An identifier always refers to the same entity (prohibits reusing an identifier)
- An identifier need not necessarily be a pure name, i.e., it may have content.

Flat naming

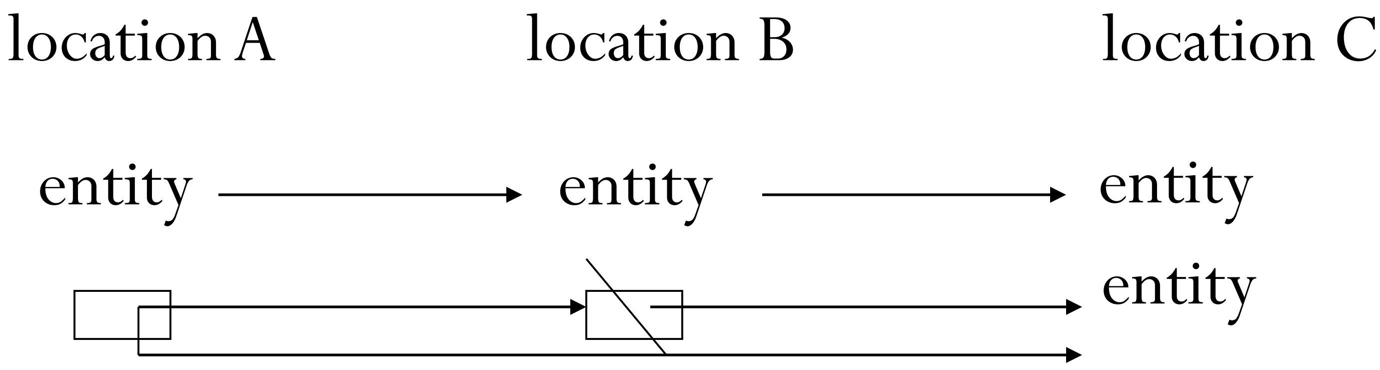
- Given an essentially unstructured name (e.g., an identifier), how can we locate its associated access point?
  - Simple solutions
    - Broadcasting
    - Forwarding pointers
    - Home-based approaches
  - Distributed Hash Tables (structured P2P)
  - Hierarchical location service

Simple Solutions (Broadcasting)

- Broadcast the ID, requesting the entity to return its current address.
  - Can never scale beyond local-area networks
  - Requires all processes to listen to incoming location requests

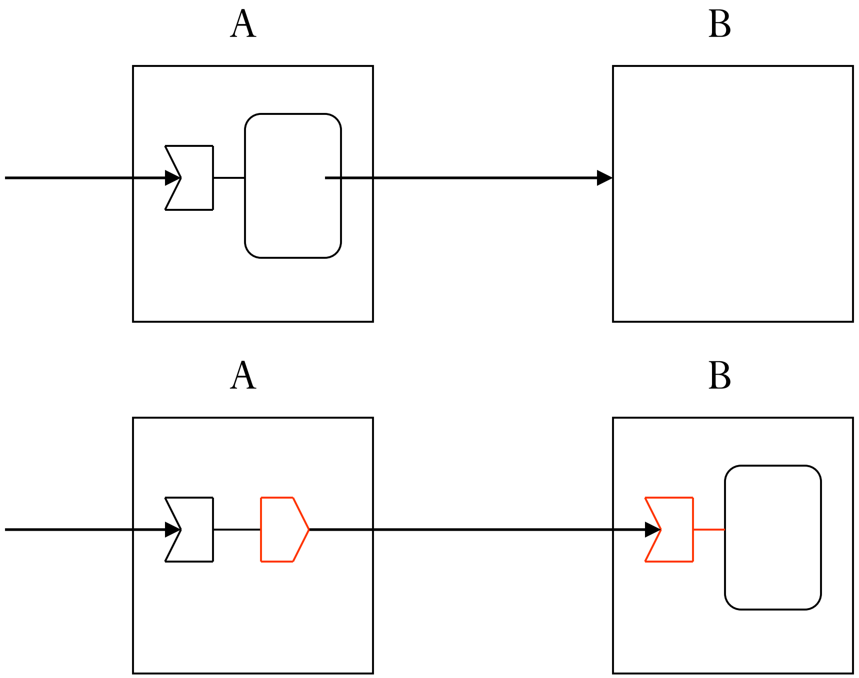


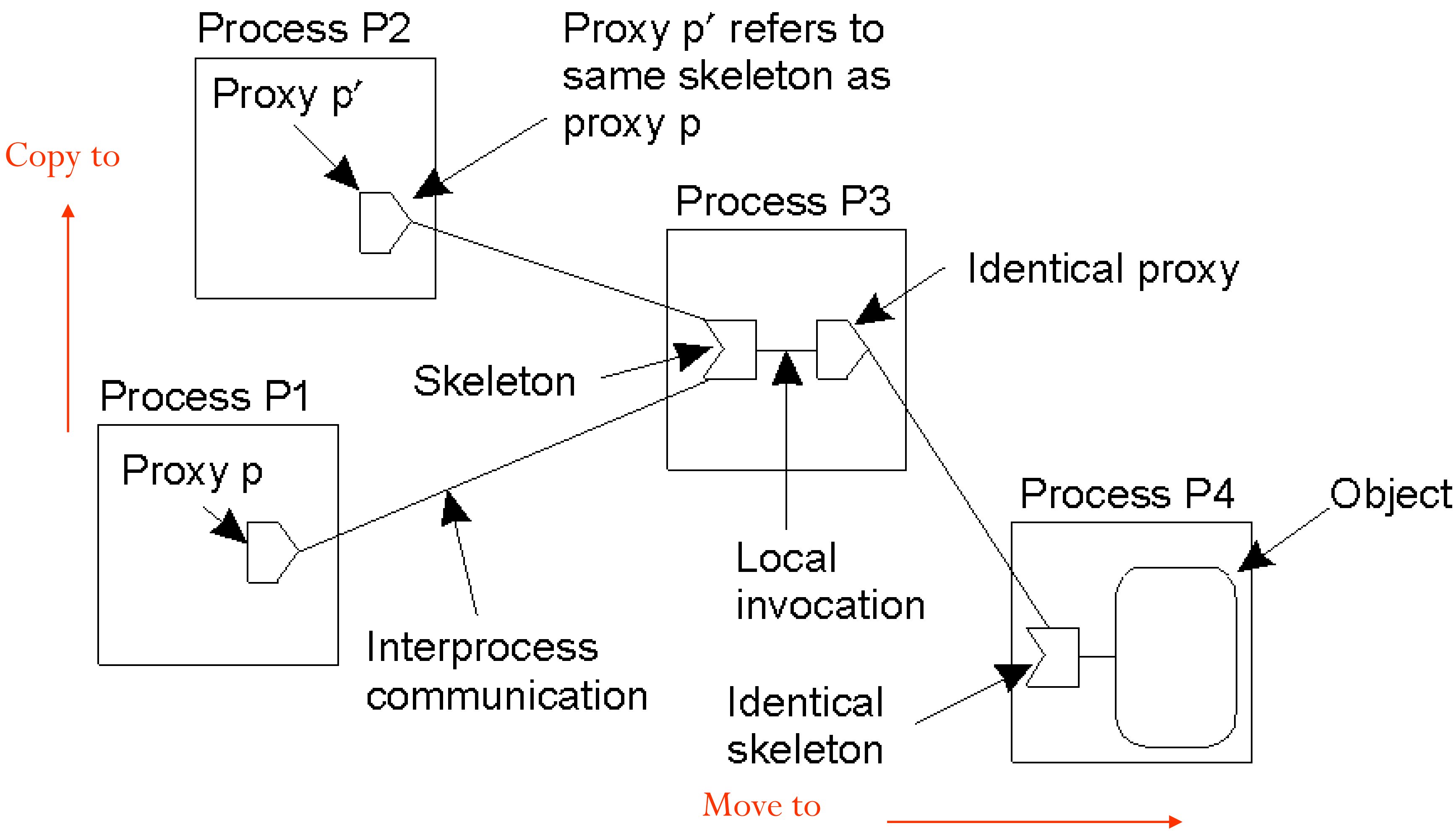
Forwarding Pointers



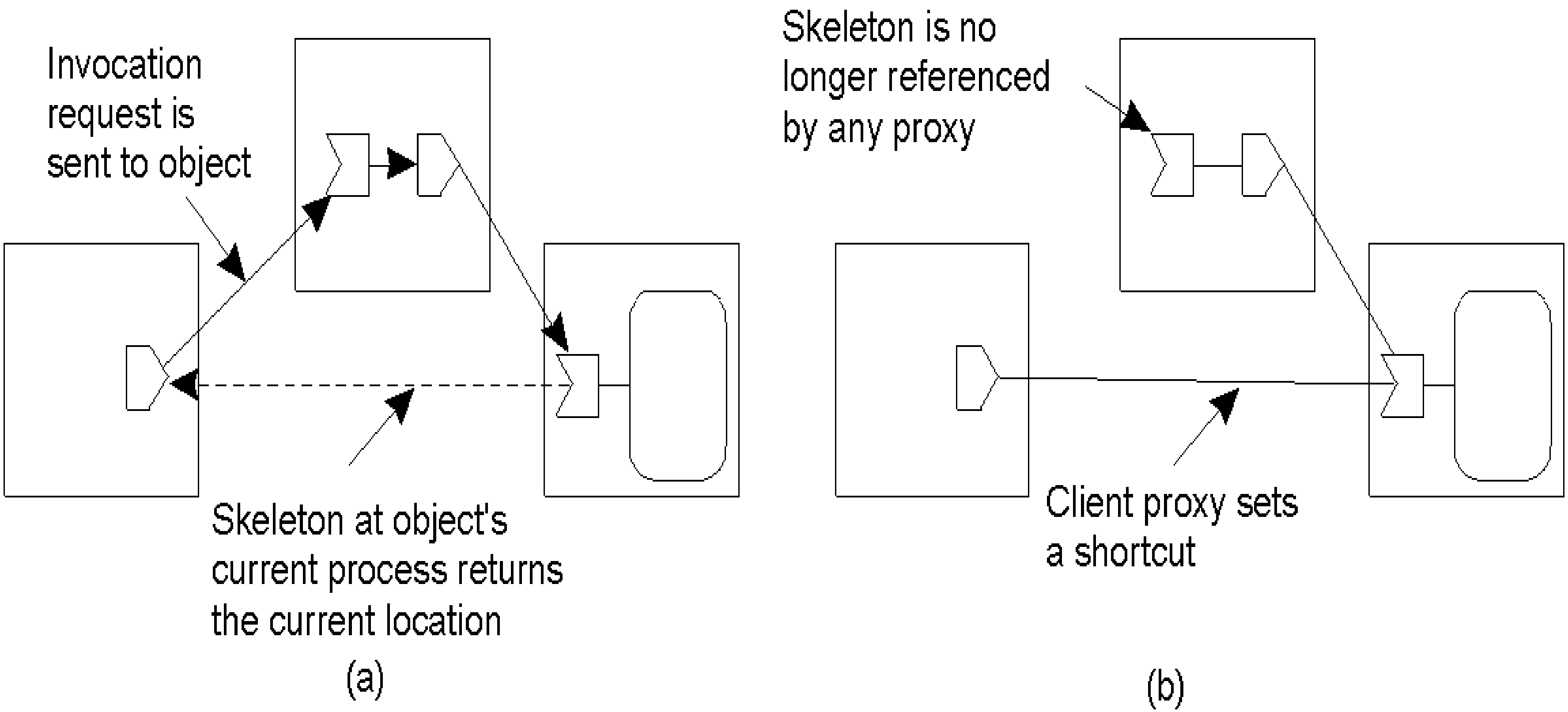
When an entity move from A to B, it leaves behind a reference to its new location at B

With respect to distributed objects, when an object move from address space A to B, it leaves behind a proxy in A and install a skeleton that refers to it in B.





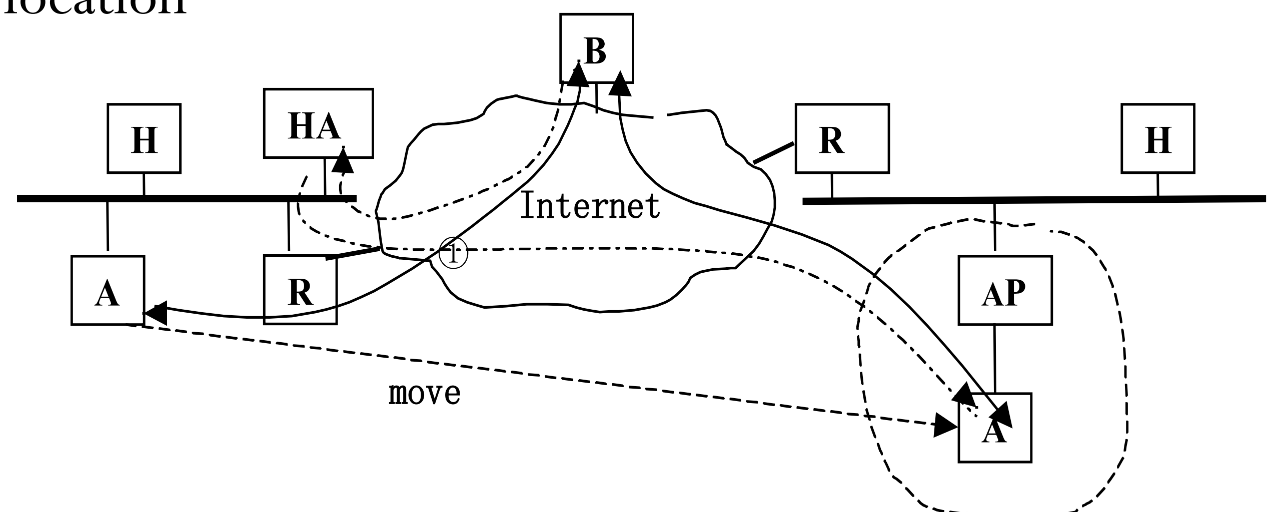
The principle of forwarding pointers using (*proxy, skeleton*) pairs.



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

Home-based approaches

- Let a home keep track of where the entity is:
  - Entity's home address registered at a naming service
  - The home registers the foreign address of the entity
  - Client contacts the home first, and then continues with foreign location

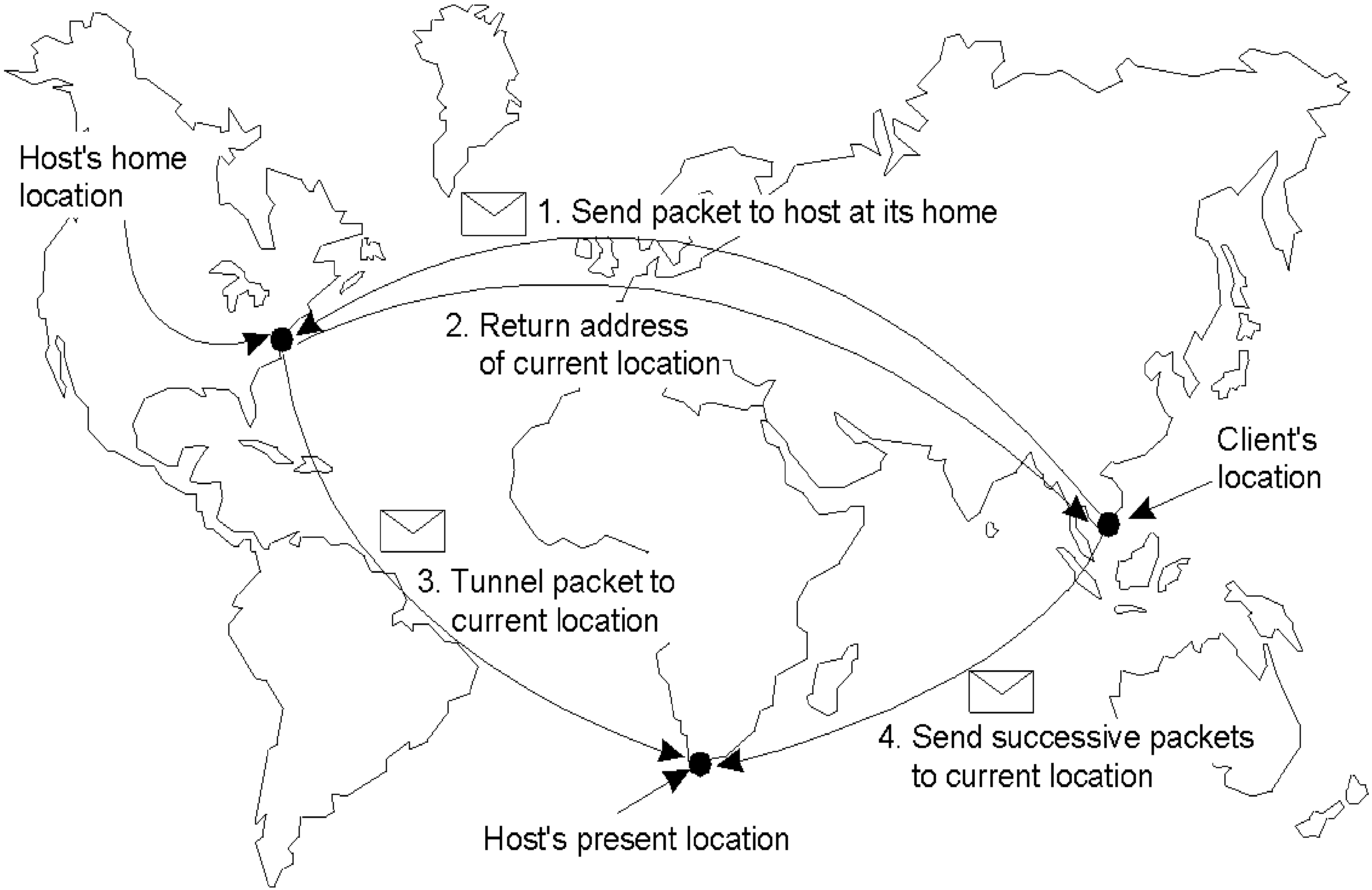


HA—home agent
R— router

A—mobile host
H—fixed host

Principle of mobile IPv6

Home-based approaches



Home-based approaches

- Keep track of visiting entities:
  - Check local visitor register first
  - Fall back to home location if local lookup fails
- Problems with home-based approaches
  - Home address has to be supported for entity's lifetime
  - Home address is fixed ⇒ unnecessary burden when the entity permanently moves
  - Poor geographical scalability (entity may be next to client)
- Question: How can we solve the "permanent move" problem?

Distributed Hash Tables (DHT)

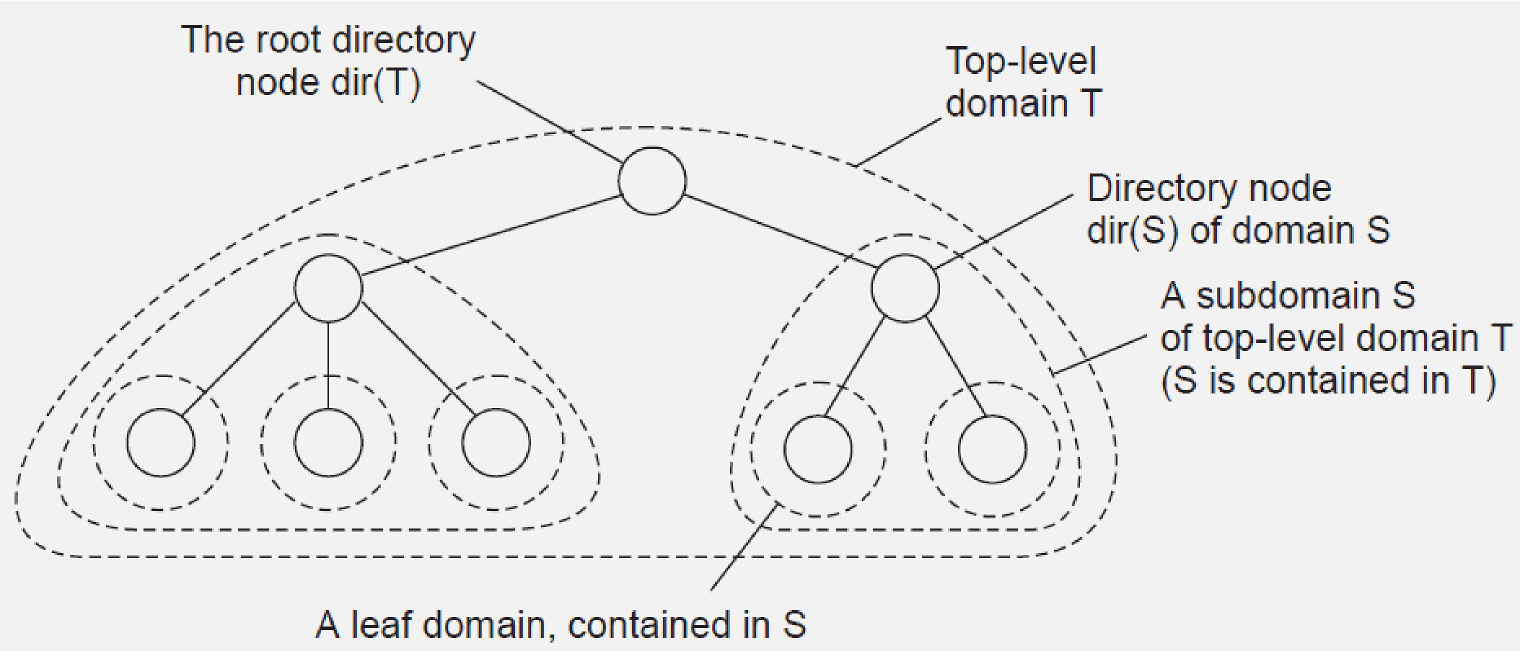
- Consider the organization of many nodes into a logical ring
  - Each node is assigned a random m-bit identifier.
  - Every entity is assigned a unique m-bit key.
  - Entity with key k falls under jurisdiction of node with smallest $id \geq k$ (called its successor).
- Let node id keep track of succ(id) and start linear search along the ring.

Exploiting network proximity

- The logical organization of nodes in the overlay may lead to erratic message transfers in the underlying Internet: node k and node succ(k +1) may be very far apart.
- Topology-aware node assignment: When assigning an ID to a node, make sure that nodes close in the ID space are also close in the network. Can be very difficult.
- Proximity routing: Maintain more than one possible successor, and forward to the closest.
  - Example: in Chord $FT_p[i]$ points to first node in $\mathrm{INT} = [p + 2^{i-1}, p + 2^i - 1]$. Node p can also store pointers to other nodes in INT .
- Proximity neighbor selection: When there is a choice of selecting who your neighbor will be (not in Chord), pick the closest one.

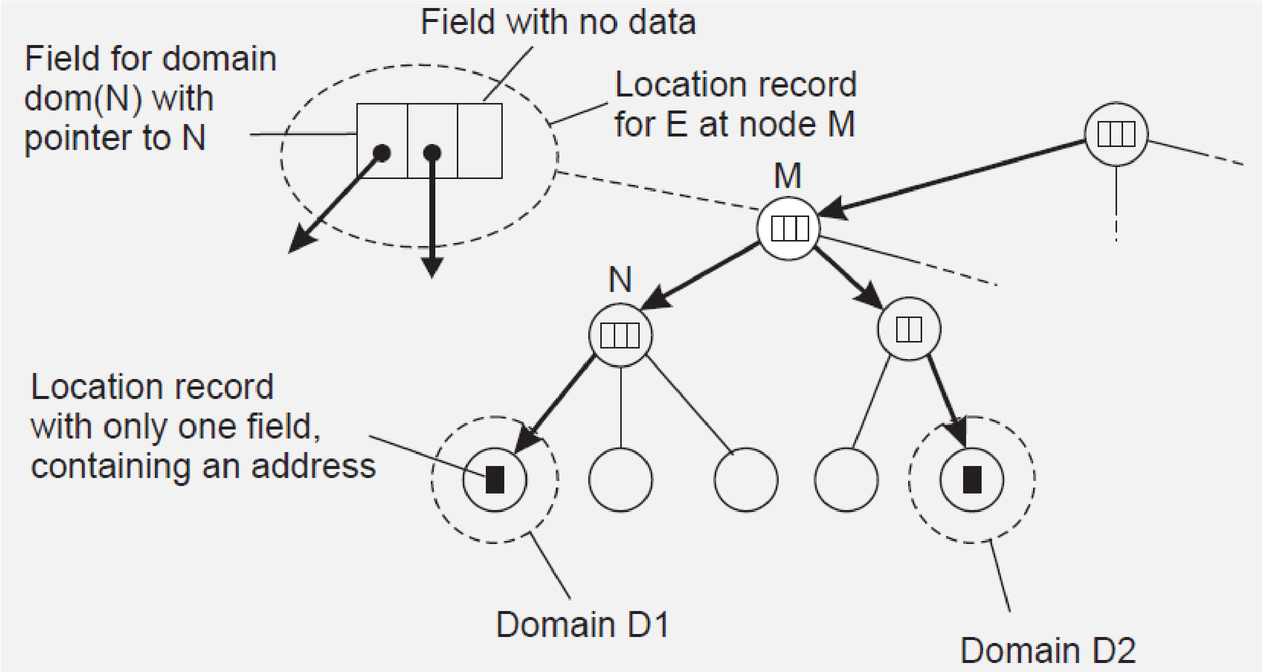
Hierarchical Location Services (HLS)

- 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.



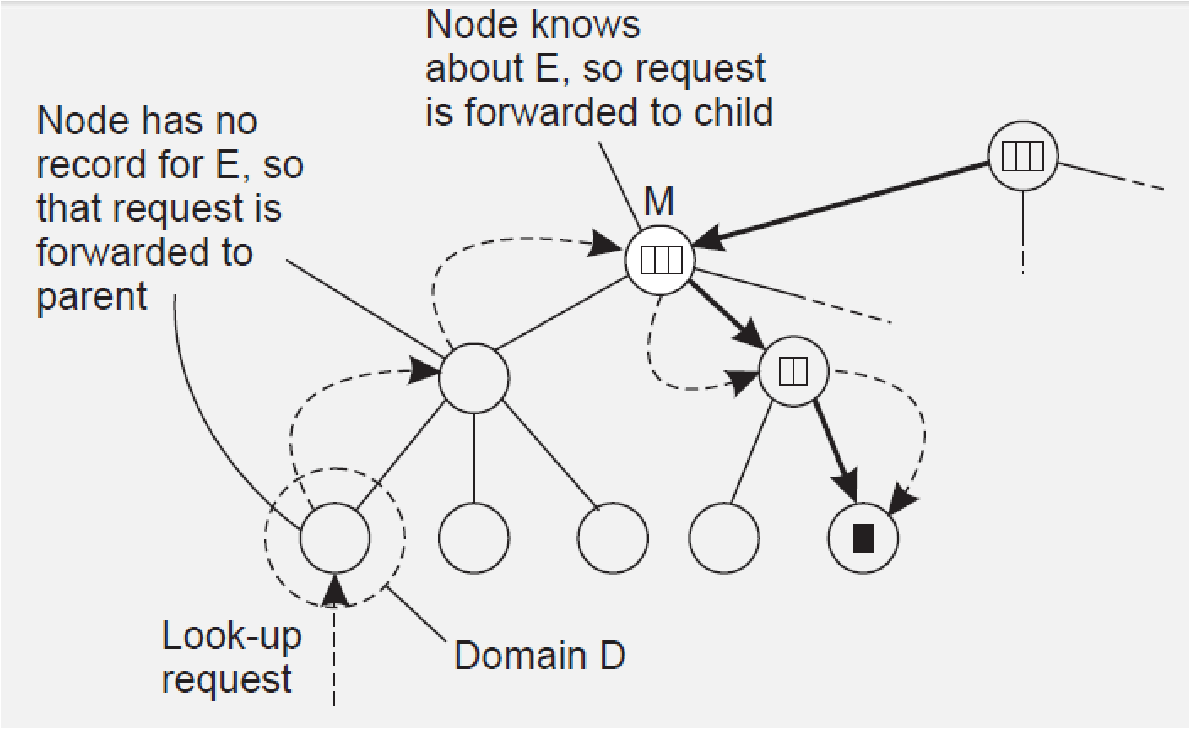
HLS: Tree organization

- Address of entity E is stored in a leaf or intermediate node
- Intermediate nodes contain a pointer to a child iff the subtree rooted at the child stores an address of the entity
- The root knows about all entities



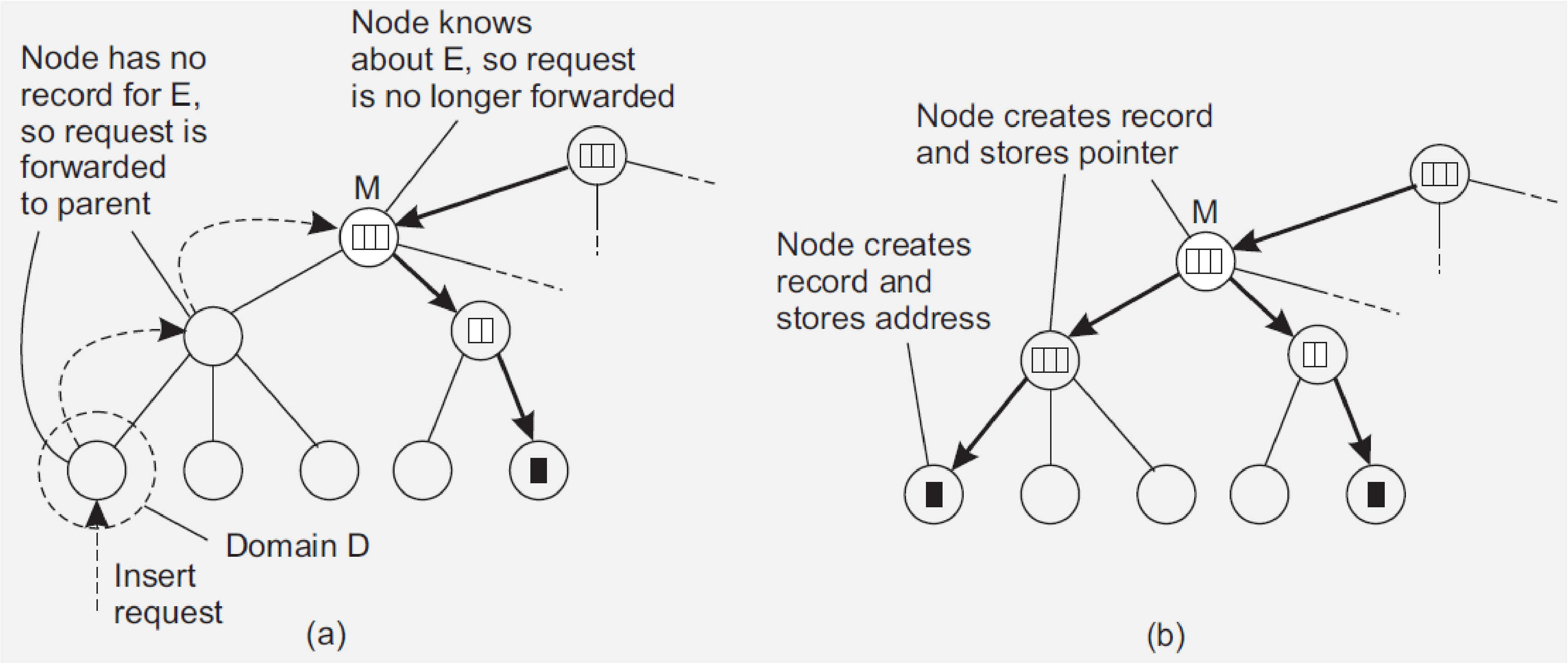
HLS: Lookup operation

- Start lookup at local leaf node
- Node knows about E ⇒ follow downward pointer, else go up
- Upward lookup always stops at root

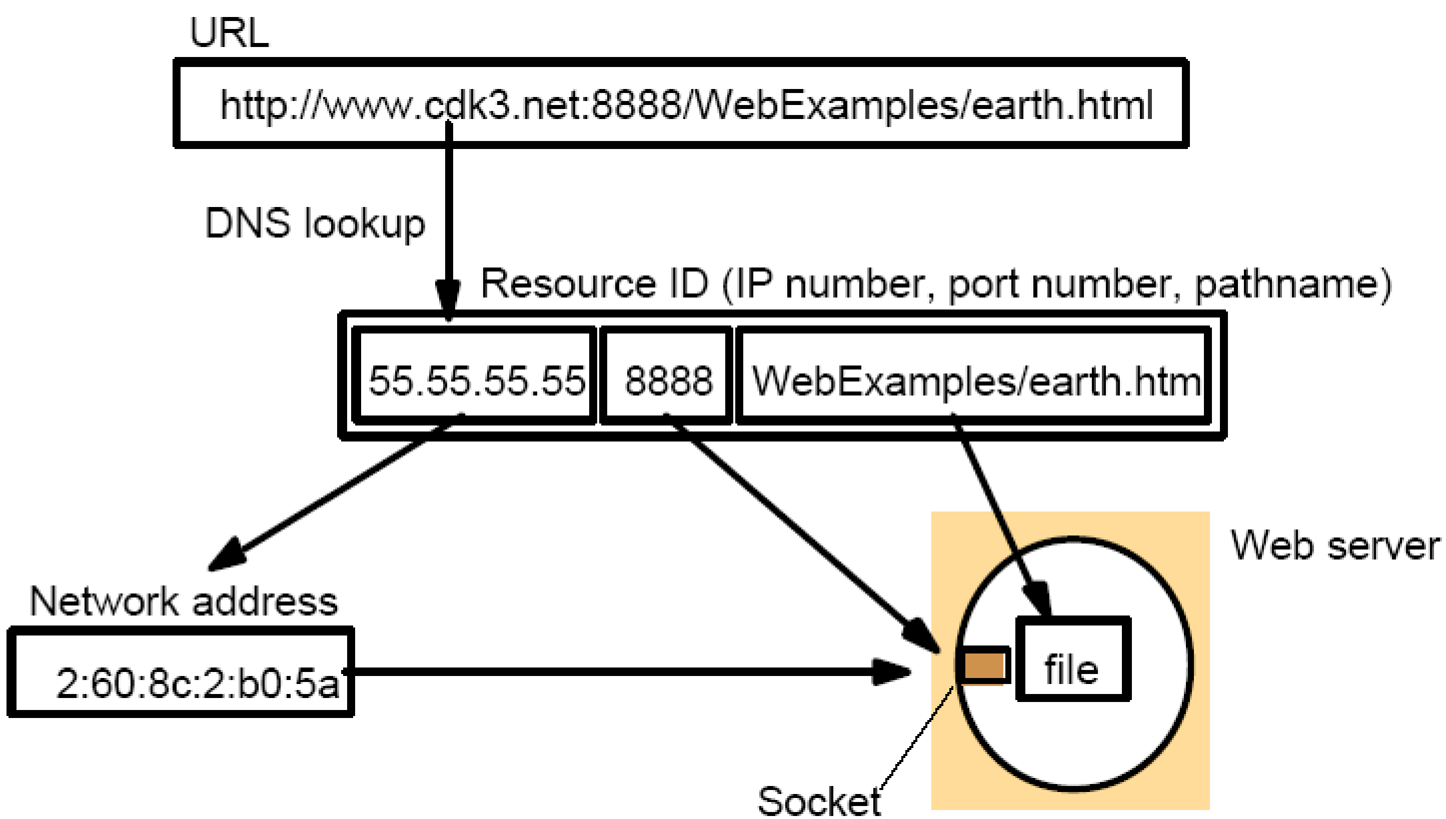


HLS: Insert operation

- An insert request is forwarded to the first node that knows about entity *E*.
- A chain of forwarding pointers to the leaf node is created.

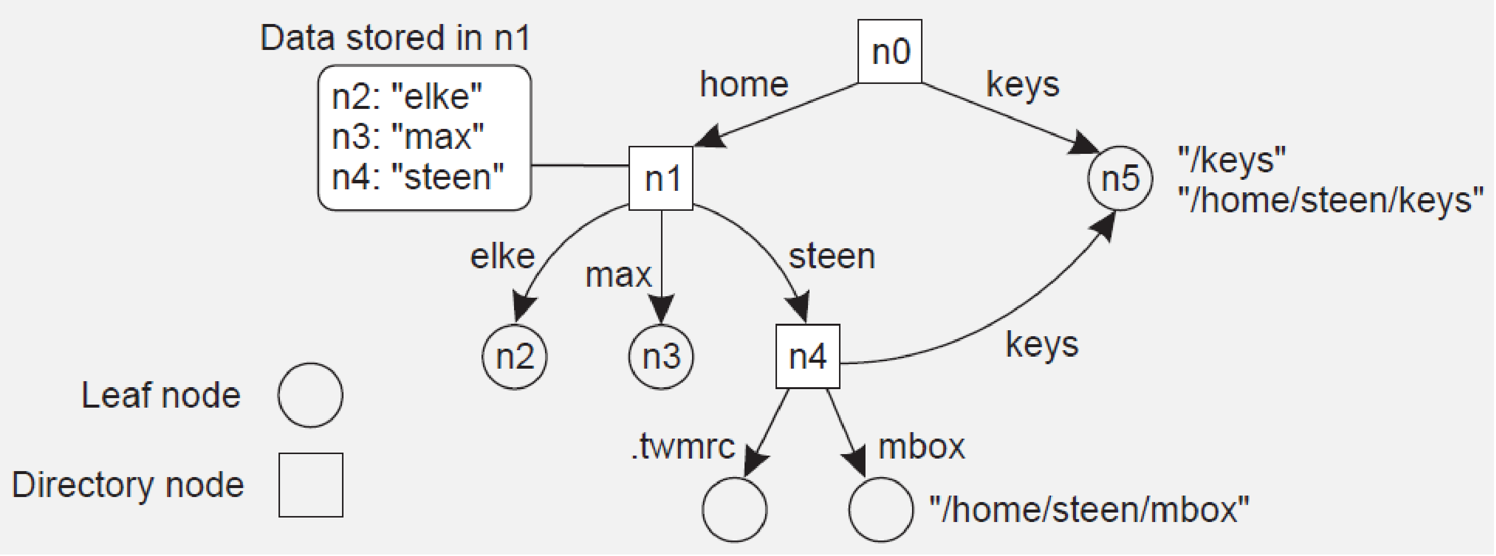


Composed Naming Domains



Name Spaces

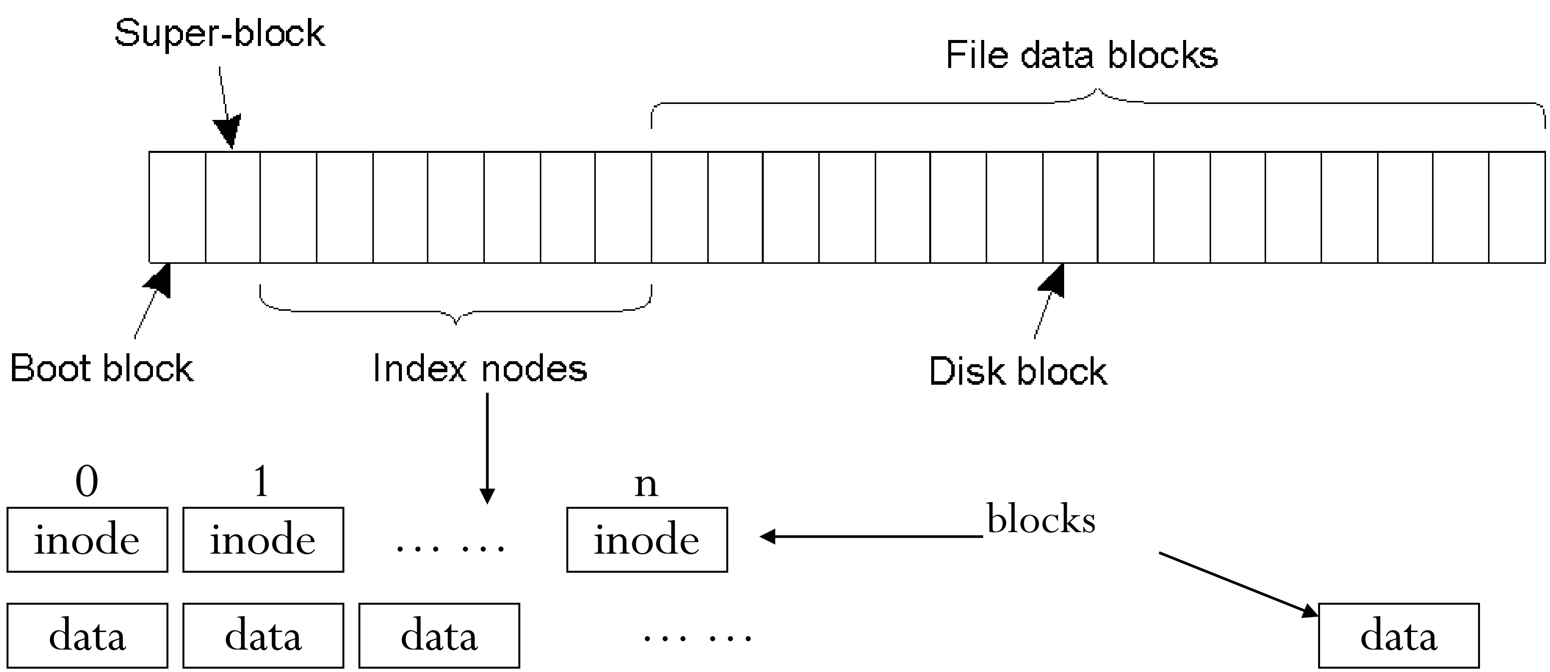
- Leaf node
  - represent named entity
  - store information (address or state) on the entity
- Directory node
  - represent a collection of entities
  - store a directory table of (edge label, node identifier).
- Path name
  - N:<label-1, label-2, …, label-n> no:<home, steen, mbox>



Name Space

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

Example of Unix file system



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

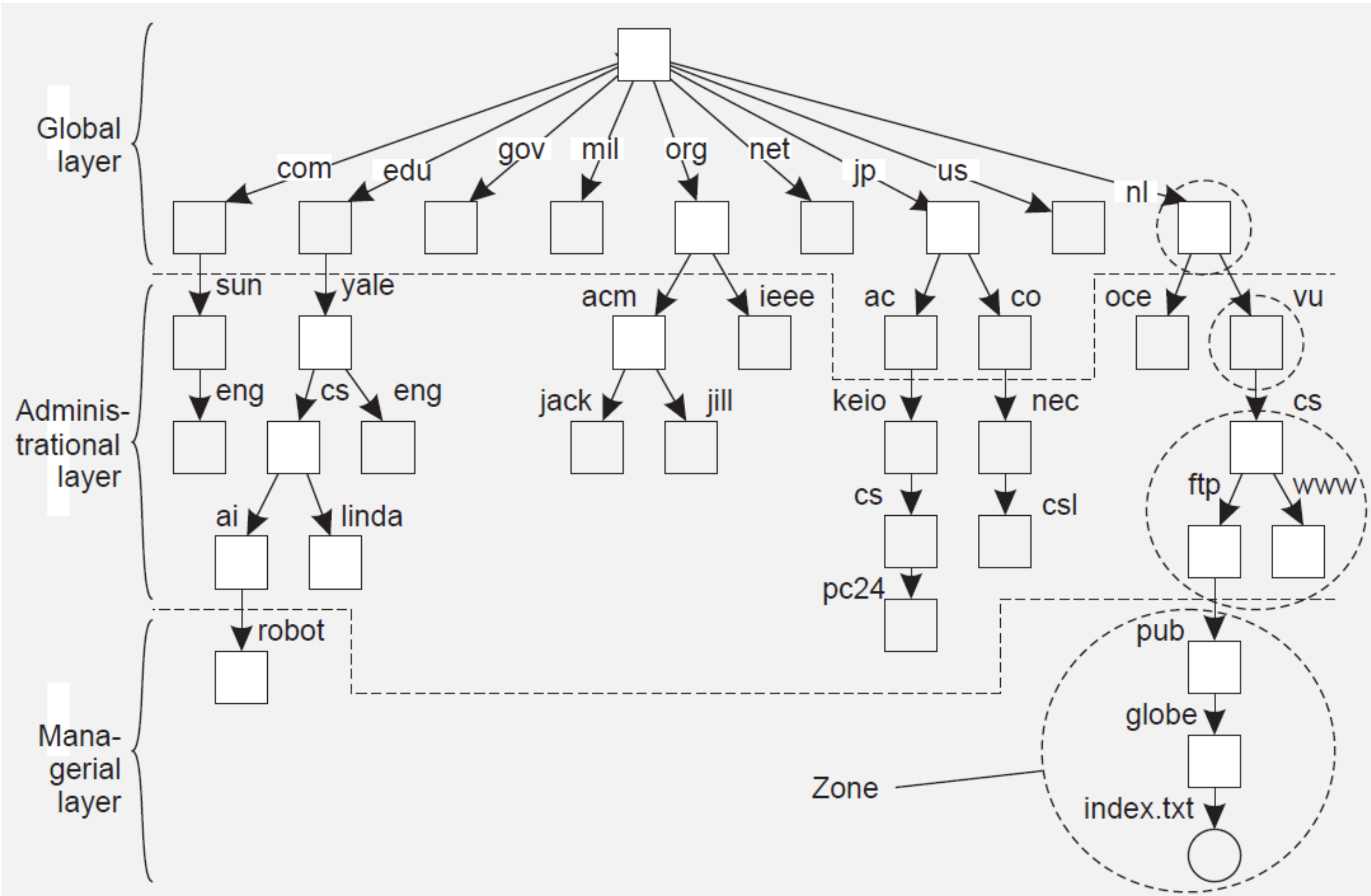
Name resolution

- To resolve a name we need a directory node. How do we actually find that (initial) node?
- 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)
  - 130.37.24.8: route to the VU's Web server

Name-space implementation

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



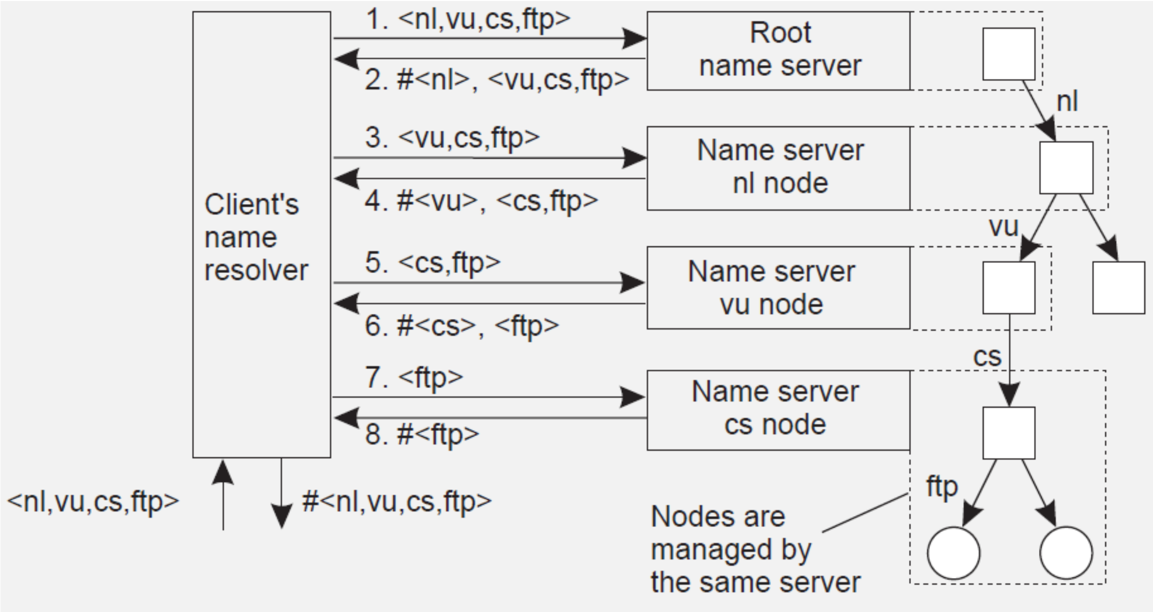
Name-space implementation

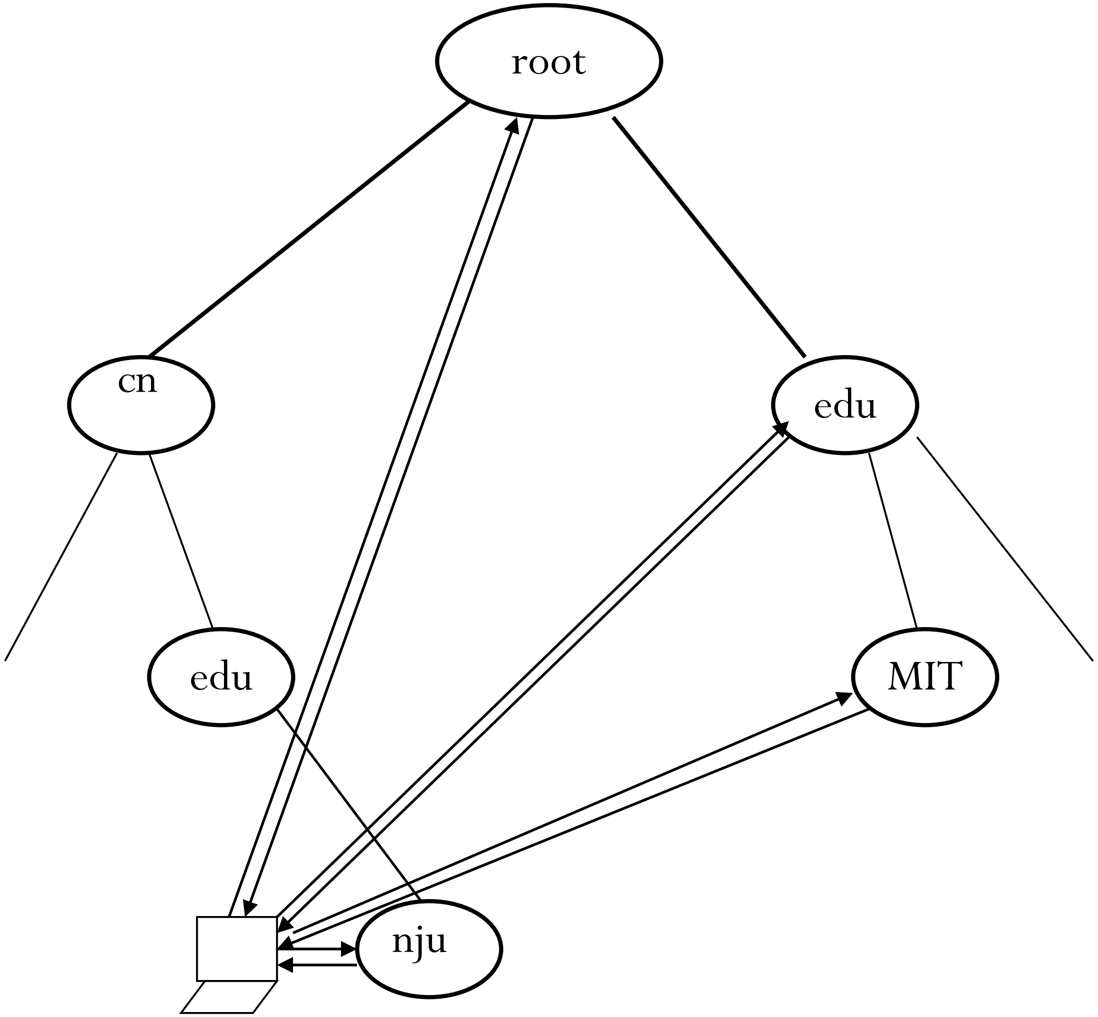
Item	Global	Administrational	Managerial
1	Worldwide	Organization	Department
2	Few	Many	Vast numbers
3	Seconds	Milliseconds	Immediate
4	Lazy	Immediate	Immediate
5	Many	None or few	None
6	Yes	Yes	Sometimes

1: Geographical scale 2: # Nodes 3: Responsiveness	4: Update propagation 5: # Replicas 6: Client-side caching?

Iterative name resolution

1. Resolve $(dir, [name1, \ldots, nameK])$ sent to Server0 responsible for dir
2. Server0 resolves resolve $(dir, name1) \rightarrow dir1$, returning the identification (address) of Server1, which stores dir1.
3. Client sends resolve $(dir, [name2, \ldots, nameK])$ to Server1, etc.





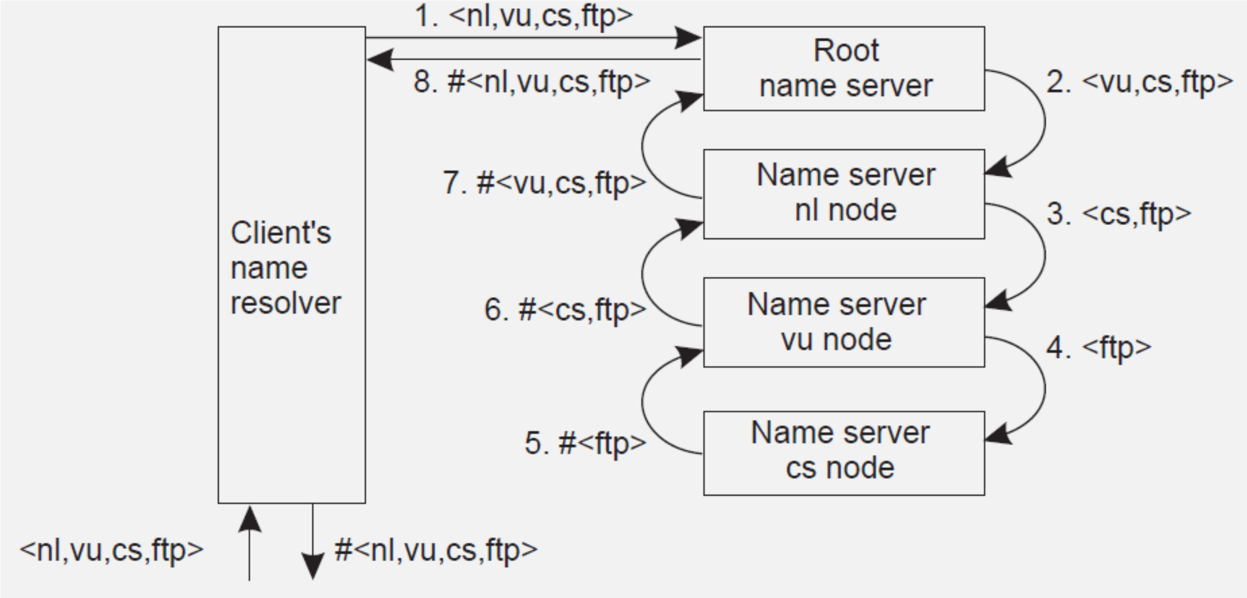
Iterative resolution

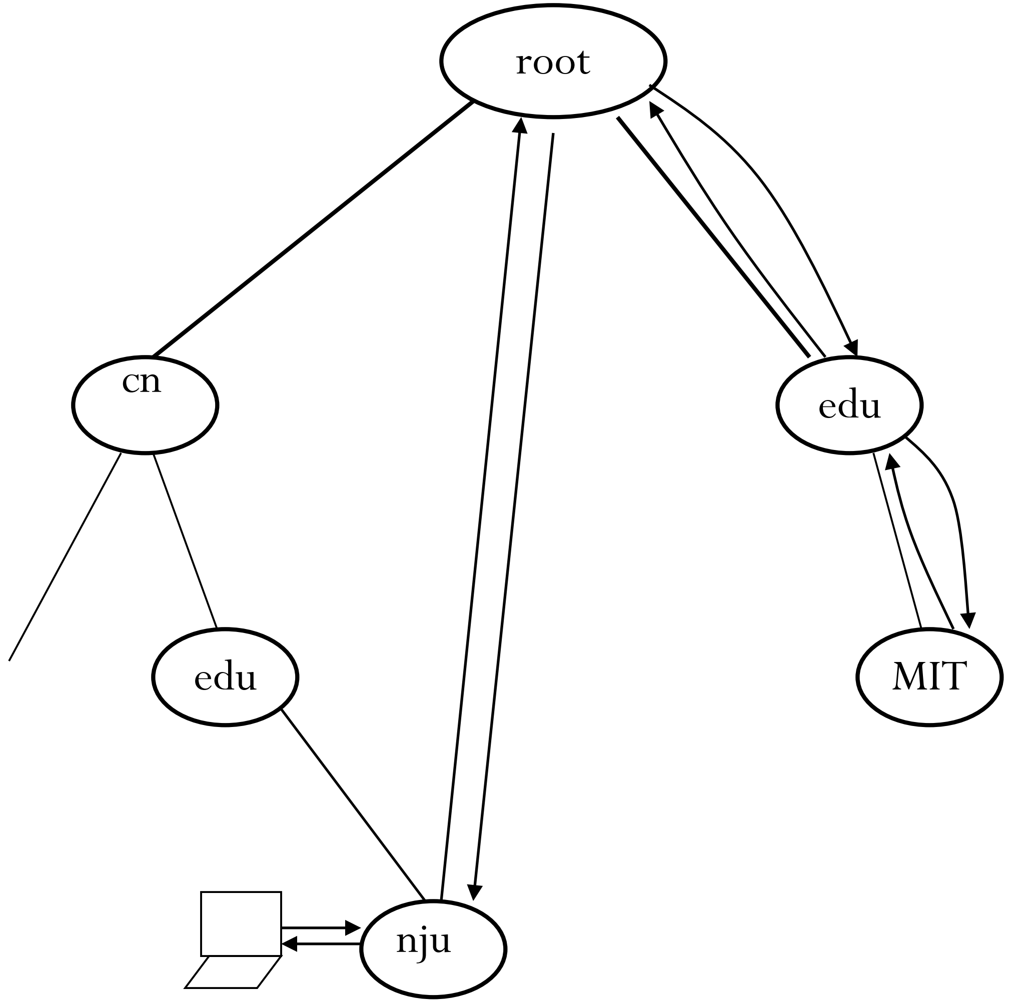
Recursive name resolution

1. Resolve $(dir, [name1, \ldots, nameK])$ sent to Server0 responsible for dir

2. Server0 resolves resolve $(dir, name1) \rightarrow dir1$, and sends resolve(dir1,[name2,…,nameK]) to Server1, which stores dir1.

3. Server0 waits for result from Server1, and returns it to client.





Recursive resolution

Caching in recursive name resolution

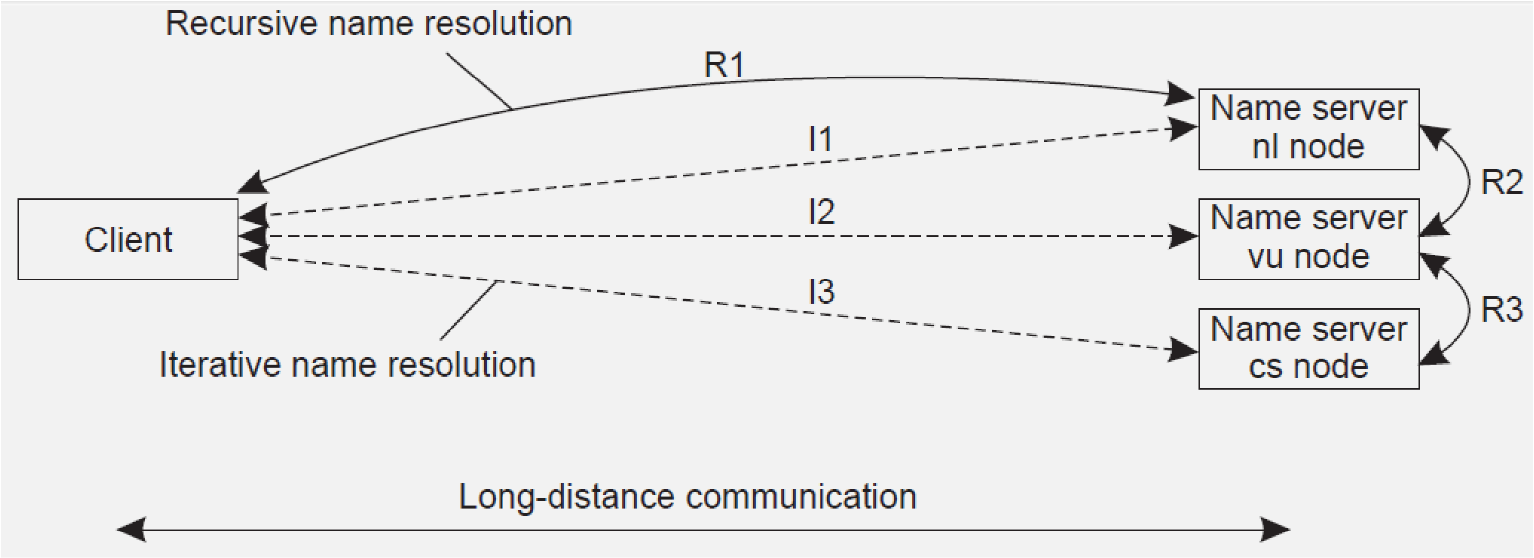
Server for node	Should resolve	Looks up	Passes to child	Receives and caches	Returns to requester
cs	<ftp>	#<ftp>	—	—	#<ftp>
vu	<cs,ftp>	#<cs>	<ftp>	#<ftp>	#<cs> #<cs, ftp>
nl	<vu,cs,ftp>	#<vu>	<cs,ftp>	#<cs> #<cs,ftp>	#<vu> #<vu,cs> #<vu,cs,ftp>
root	<nl,vu,cs,ftp>	#<nl>	<vu,cs,ftp>	#<vu> #<vu,cs> #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

Scalability issues

- We need to ensure that servers can handle a large number of requests per time unit.
  - $\Rightarrow$ high-level servers are in big trouble.
- Assume (at least at global and administrational level) that content of nodes hardly ever changes. We can then apply extensive replication by mapping nodes to multiple servers, and start name resolution at the nearest server.
- 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

- We need to ensure that the name resolution process scales across large geographical distances.



- By mapping nodes to servers that can be located anywhere, we introduce an implicit location dependency.

Example: Decentralized DNS

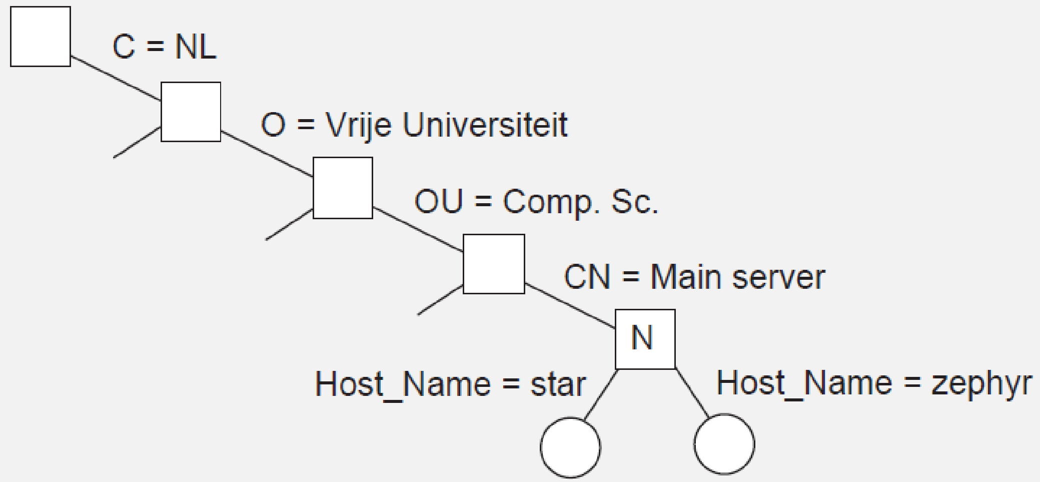
- Take a full DNS name, hash into a key k, and use a DHT-based system to allow for key lookups. Main drawback: You can't ask for all nodes in a subdomain (but very few people were doing this anyway).

SOA	Zone	Holds info on the represented zone
A	Host	IP addr. of host this node represents
MX	Domain	Mail server to handle mail for this node
SRV	Domain	Server handling a specific service
NS	Zone	Name server for the represented zone
CNAME	Node	Symbolic link
PTR	Host	Canonical name of a host
HINFO	Host	Info on this host
TXT	Any kind	Any info considered useful

Attribute-based naming

- In many cases, it is much more convenient to name, and look up entities by means of their attributes ⇒ traditional directory services.
- Lookup operations can be extremely expensive, as they require to match requested attribute values, against actual attribute values ⇒ inspect all entities (in principle).
- Implement basic directory service as database, and combine with traditional structured naming system.

Example: LDAP



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

Answer = search ("& (C = NL) (O = Vrije Universiteit) (OU = *) (CN = Main server)")