POLITECNICO
MILANO 1863

Distributed Systems

Naming

Gianpaolo Cugola
Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy
cugola@elet.polimi.it
http://home.dei.polimi.it/cugola

Contents

- **Basics**
  - **Entities, names, addresses, identifiers**
  - **Name resolution**
- Flat naming
  - Simple solutions
  - Home-based approaches
  - Distributed hash tables
  - Hierarchical approaches
- Structured naming
  - Name spaces and name servers
- Attribute based naming
  - Directory services, LDAP
- Removing unreferenced entities
  - Reference counting, reference listing, distributed mark-and-sweep

Naming concepts

- *Names* are used to refer to *entities*
  - Hosts, users, files, services, …
- Entities are usually accessed through an *access point*
  - A special entity characterized by an *address*
- An *address* is just a special case of a *name*
- Observation:
  - The same entity can be accessed through several access points at the same time...
    - When I am in my office I can be reached through e-mail, cell phone, office phone
  - ...and it can change its access points during its lifetime
    - My notebook changes its network (and consequently its IP address) while I move from office to home
- Consequence:
  - It is not convenient to use the address of its access point as a name for an entity...
  - ...better using location-independent names
    - Totally opaque: They do not depend on the access point used to access the entity

The same entity can be accessed through multiple access point and some of them can be active or not.
(es: first Wi-FI network, then wired network. The address in this case changes in time)

Naming concepts

es: file's name, it's valid only in the directory where the file is located

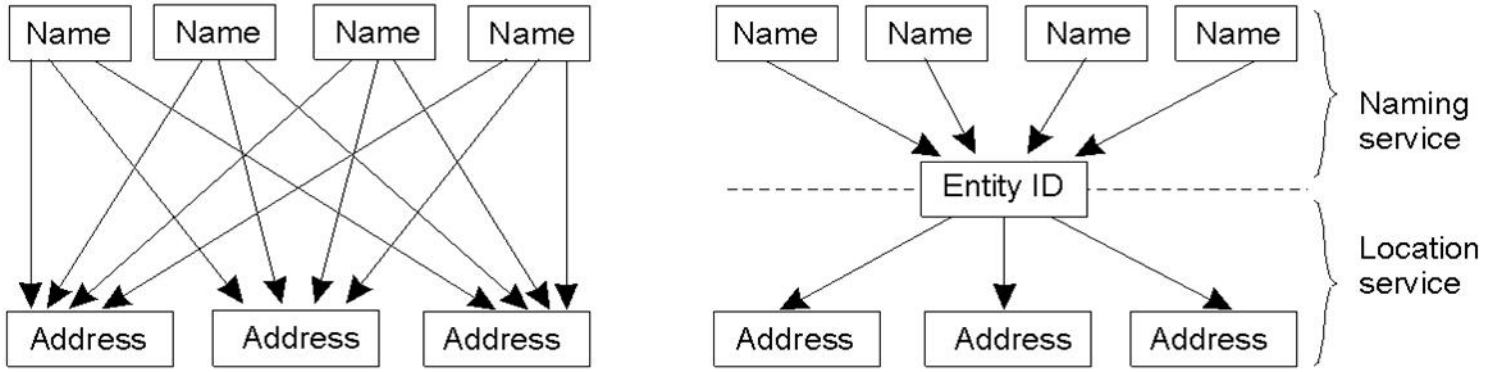
- Global vs. local names

es: entire path to the file

  - A global names denotes the same entity, no matter where the name is used
  - Interpretation of a local name depends on where it is being used
- Human-friendly names vs. machine-friendly names
  - 00:15:2A:2E:E1:97 vs “My Mobile”

Identifiers

- Resolving a name directly into an address does not work with mobility
- *Identifiers* are names such that:
  - They never change (during the lifetime of the entity)
  - Each entity has exactly one identifier
  - An identifier for an entity is never assigned to another entity
- Using identifiers enables to split the problem of mapping a name to an entity and the problem of locating the entity



Name resolution

- It is the process of obtaining the address of a valid access point of an entity having its name
- Examples:
  - DNS maps domain names to hosts
  - X500 and LDAP maps a person's name to email, telephone number, …
  - The RMI registry maps the name of a Java remote object to its remote reference
  - UDDI maps the description of a Web service to the service metadata (e.g., a Web server) needed to access it
- The way name resolution is performed depends on the nature of the naming schema employed
  - *Flat naming*
  - *Structured naming*
  - *Attribute-based naming*

Contents

- Basics
  - Entities, names, addresses, identifiers
  - Name resolution
- **Flat naming**
  - **Simple solutions**
  - **Home-based approaches**
  - **Distributed hash tables**
  - **Hierarchical approaches**
- Structured naming
  - Name spaces and name servers
- Attribute based naming
  - Directory services, LDAP
- Removing unreferenced entities
  - Reference counting, reference listing, distributed mark-and-sweep

Flat naming

- In a flat naming schema names are “flat”
  - They are simple strings with neither structure nor content
- The name resolution process
  - Simple solutions
    - Broadcast or multicast-based, forwarding pointers
  - Home-based approaches
  - Distributed hash tables
  - Hierarchical approaches

Simple solutions

- Designed for small-scale (e.g., LAN) environments
- Broadcast
  (Address Resolution Protocol)
  - Similar to ARP, send “find” message on broadcast channel: Only the interested host replies
  - Drawback: All hosts must process all “find” messages
    - Traffic and computational overhead

-who's andrea rossi?
-me

- Multicast
  - Same as broadcast, but send to a multicast address to reduce the scope of the search

sent to MAC address (address available in a certain local area)

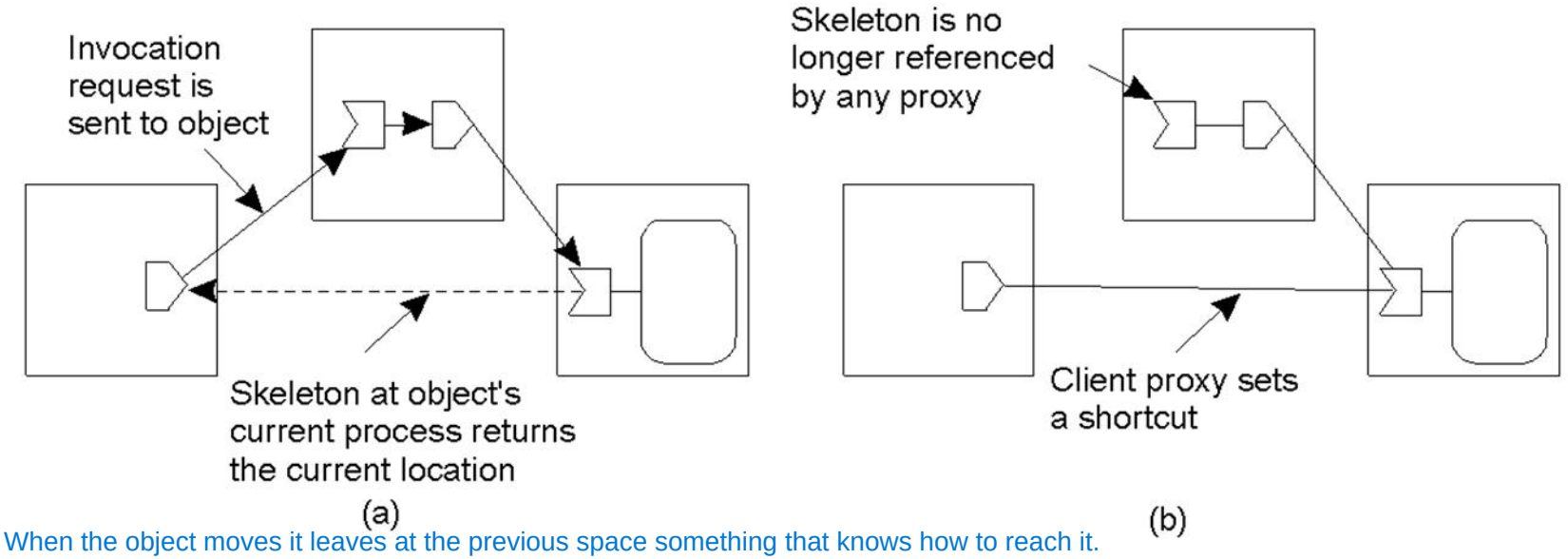
- Forwarding pointers (for mobile nodes)

was used in mobile phone communication

  - Leave reference to the next location at the previous location
  - Chains can become very long
    - Broken links cause chain to break
    - Increased network latency at dereferencing
  - Chain reduction (clean-up) mechanisms can be complex
    - When to clean-up?
    - How?

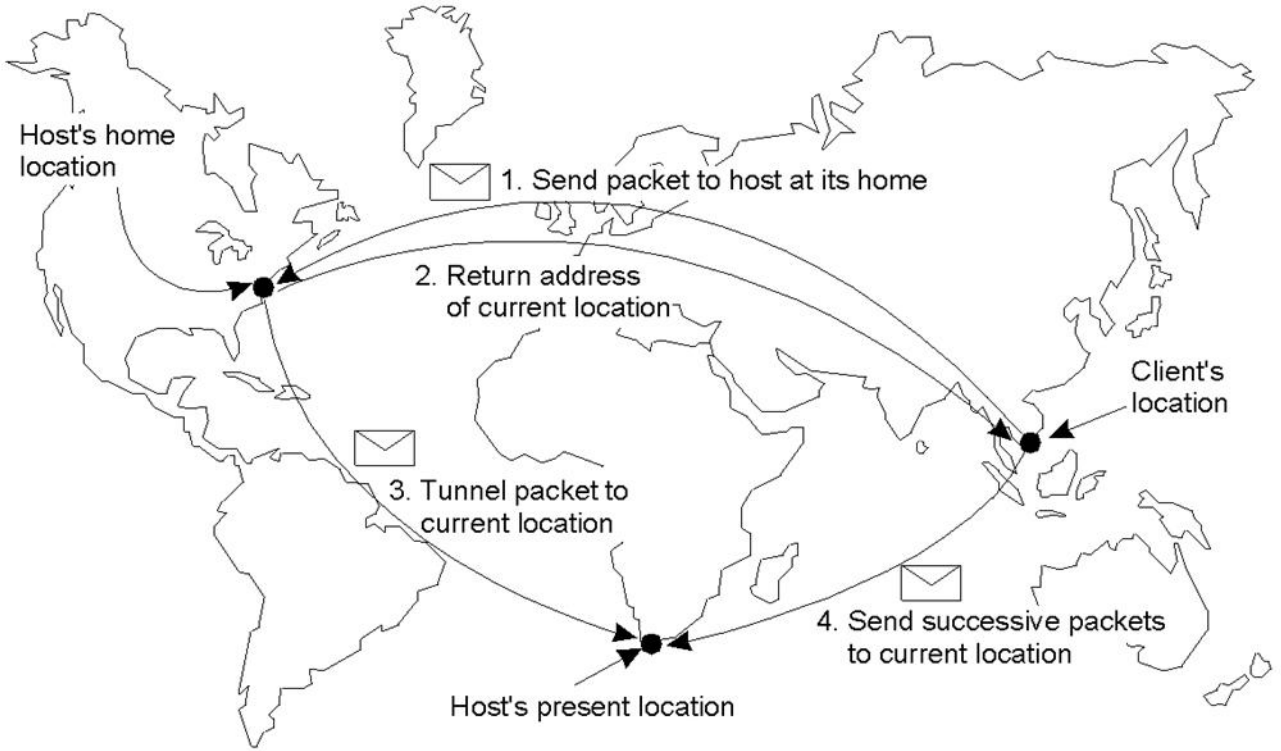
Forwarding pointers and distributed objects

- Proxies forward requests to the real object instance
  - This scheme works fine also w.r.t. parameter passing
- Keep chain short by adjusting proxies
  - Send adjustment along path or direct to original proxy?
  - Skeletons know when they are no longer referenced by a proxy and can be deleted



When the object moves it leaves at the previous space something that knows how to reach it.
In practicse, it leaves a proxy that knows how to reach the object.
After the object has been reached, it's possible that will send information back to be contacted directly without passing by the proxy again

Home-based approaches



- Used in Mobile IP and cellphone networks
- Rely on one home node that knows the location of the mobile unit
  - Home is assumed to be stable, and can be replicated for robustness
  - Original IP of the host is effectively used as an identifier

Problems with home-based approaches

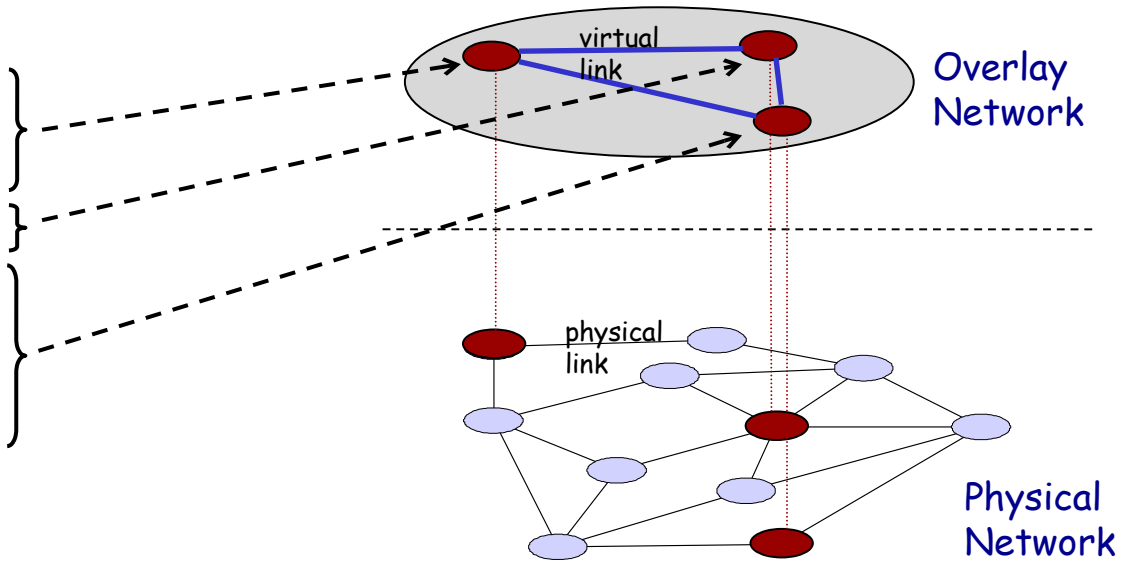
very long time before being able to communicate

- The extra step towards the home increases latency
- 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)

DHT: Overview

- *Abstraction*: A distributed "hash-table" (DHT) data structure:
  - put(id, item);
  - item = get(id);
- *Implementation*: Nodes organized in a structured overlay network
  - Can be Ring, Tree, Hypercube, Skip List, Butterfly Network, ...
- DHT and name resolution
  - id ⇔ name/identifier item ⇔ identifier/address

Key	Value
1	Algorithms
9	Routing
11	DS
12	Peer-to-Peer
21	Networks
22	Grids

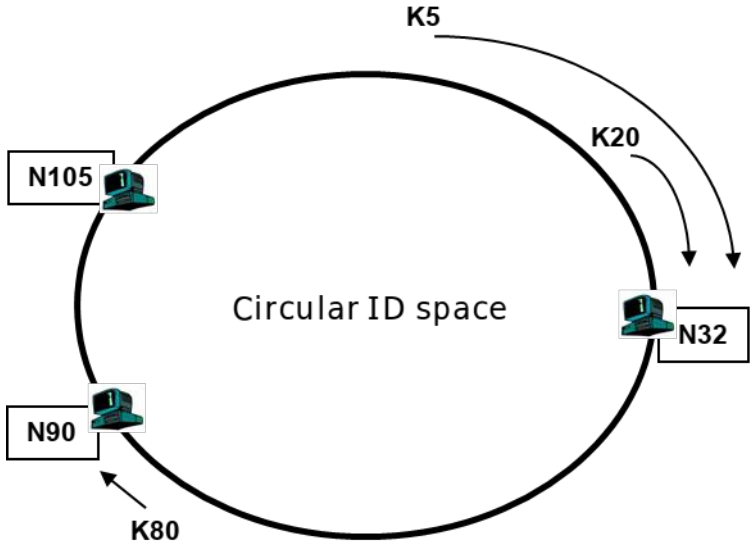


DHT example: Chord

- Nodes and keys are organized in a *logical ring*
  - Each node is assigned a unique *m*-bit identifier
    - Usually the hash of the IP address
  - Every item is assigned a unique *m*-bit key
    - Usually the hash of the item

The problem: find the node where the value associated to the key is located.

  - The item with key *k* is managed (e.g., stored) by the node with the smallest $id \geq k$ (the *successor*)

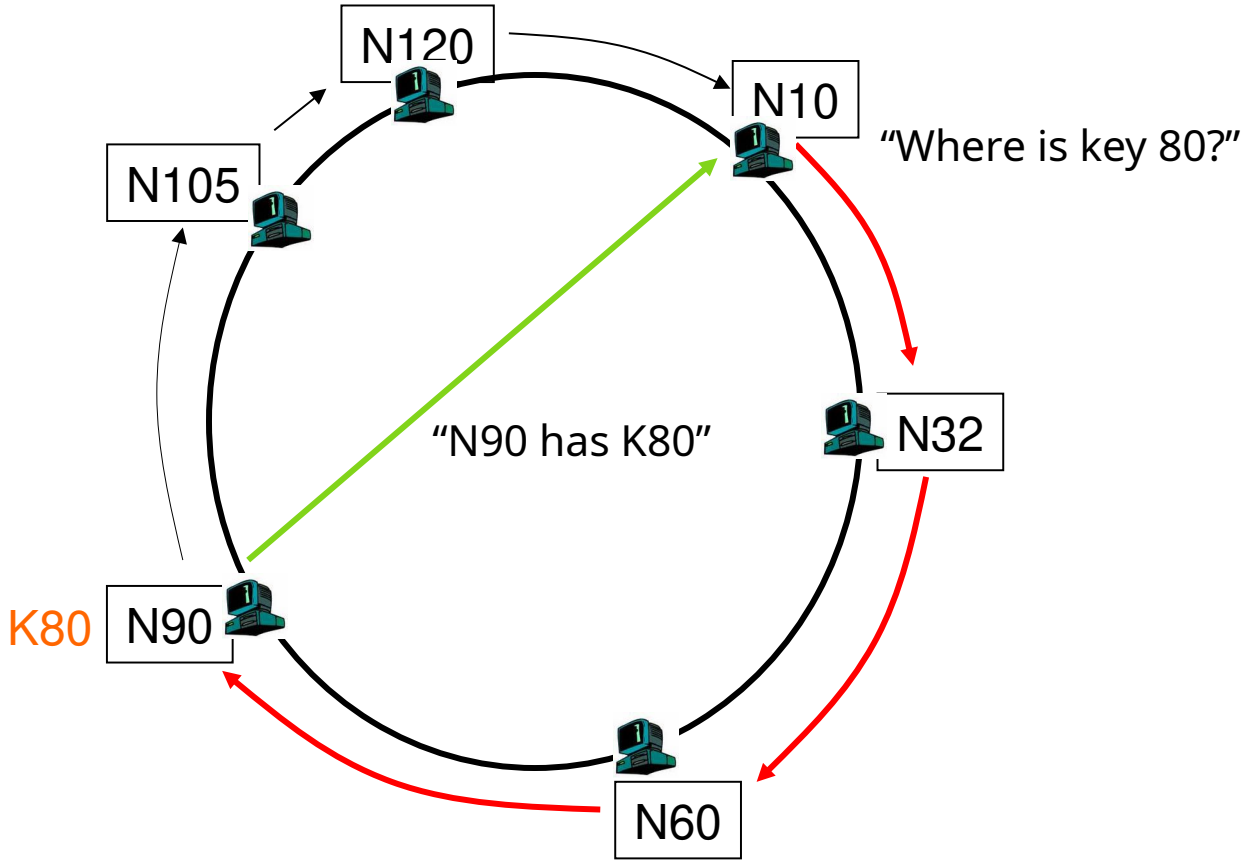


Ring --> reasoning in module

Chord: Basic lookup

- Each node keeps track of its successor
- Search is performed "linearly"

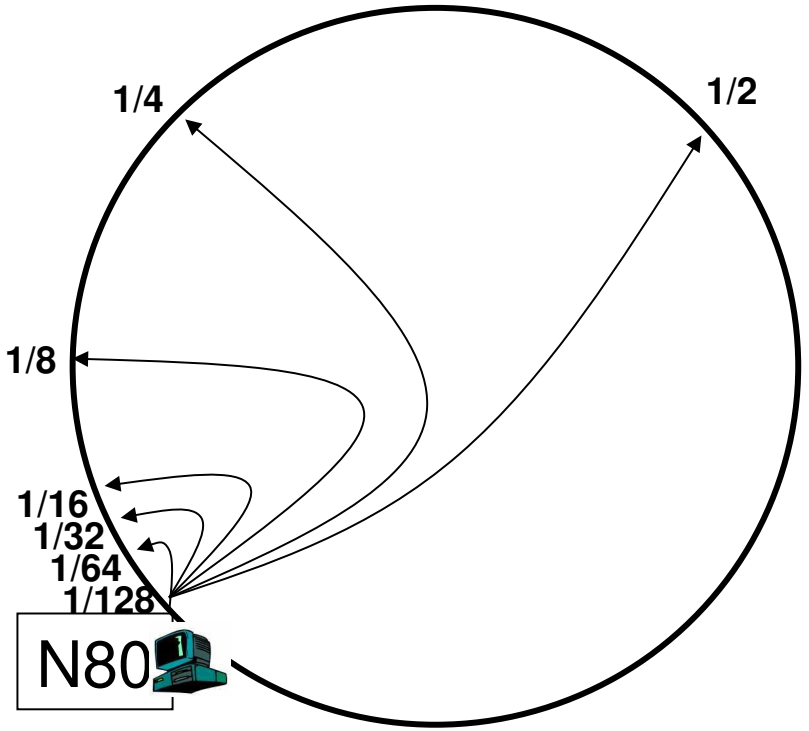
Works but is inefficient



Chord “finger table”

- Each node maintains a finger table with *m* entries
- Entry *i* in the finger table of node *n* is the first node whose id is higher or equal than $n + 2^i$ ($i = 0...m-1$)
  - In other words, the i^{th} finger points $1/2^{m-i}$ way around the ring

Distance seen as fraction of the circonference:

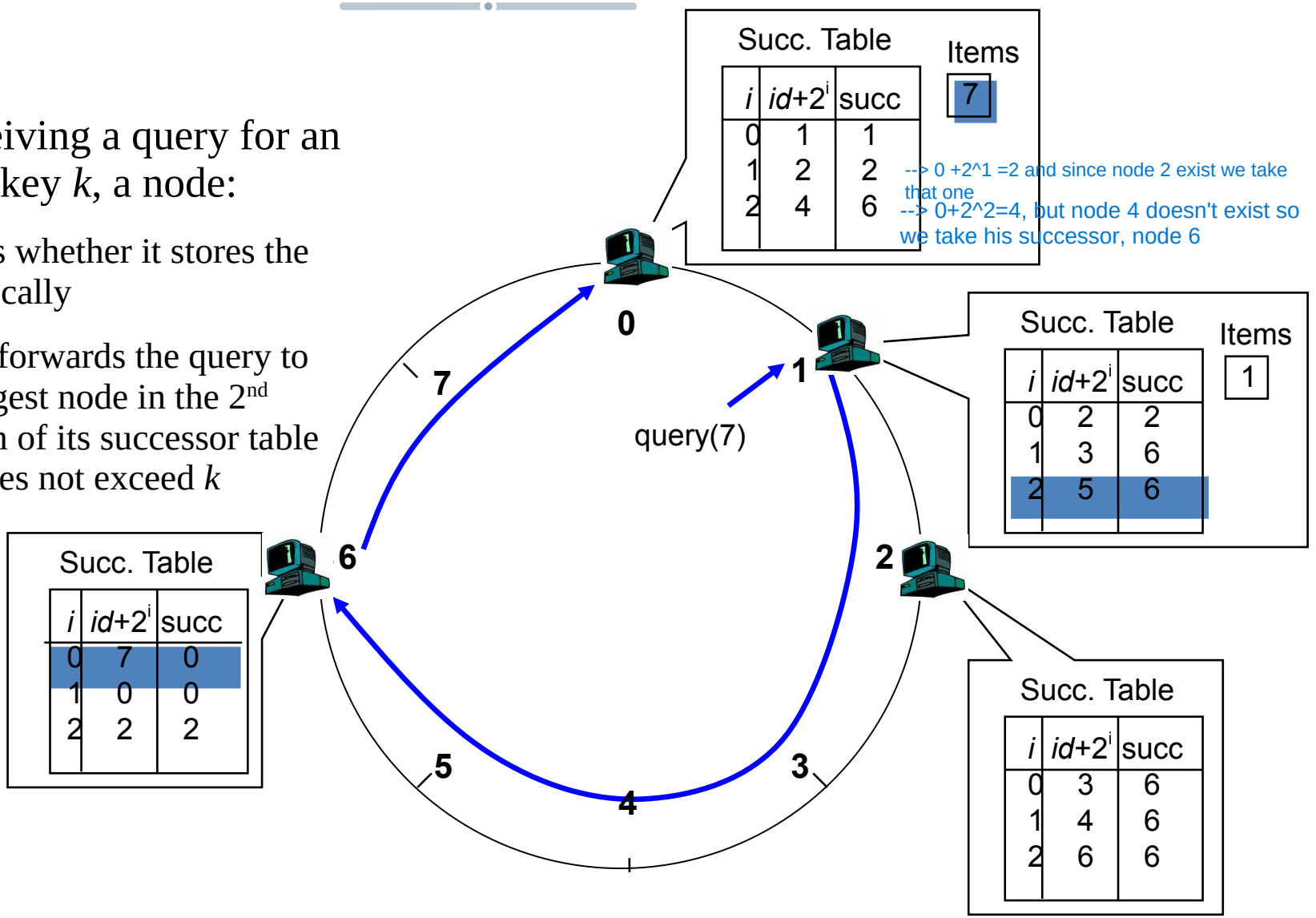


Exaple

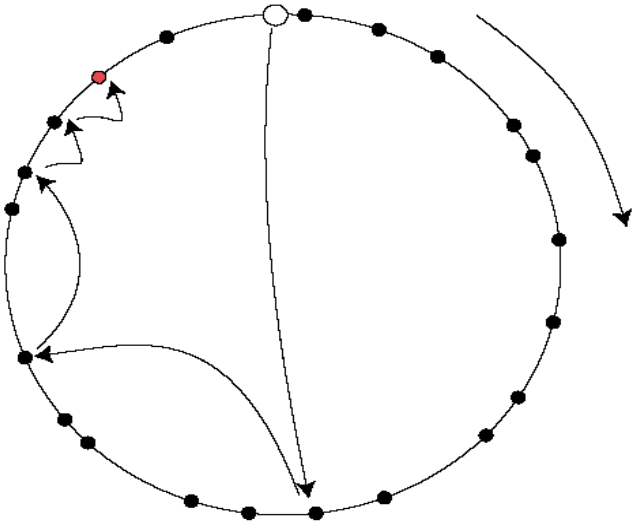
Chord routing

Number of rows per finger tables = number of bits for k = number of bits for node id = 3 in this case

- Upon receiving a query for an item with key *k*, a node:
  - Checks whether it stores the item locally
  - If not, forwards the query to the largest node in the 2nd column of its successor table that does not exceed *k*



Chord summary

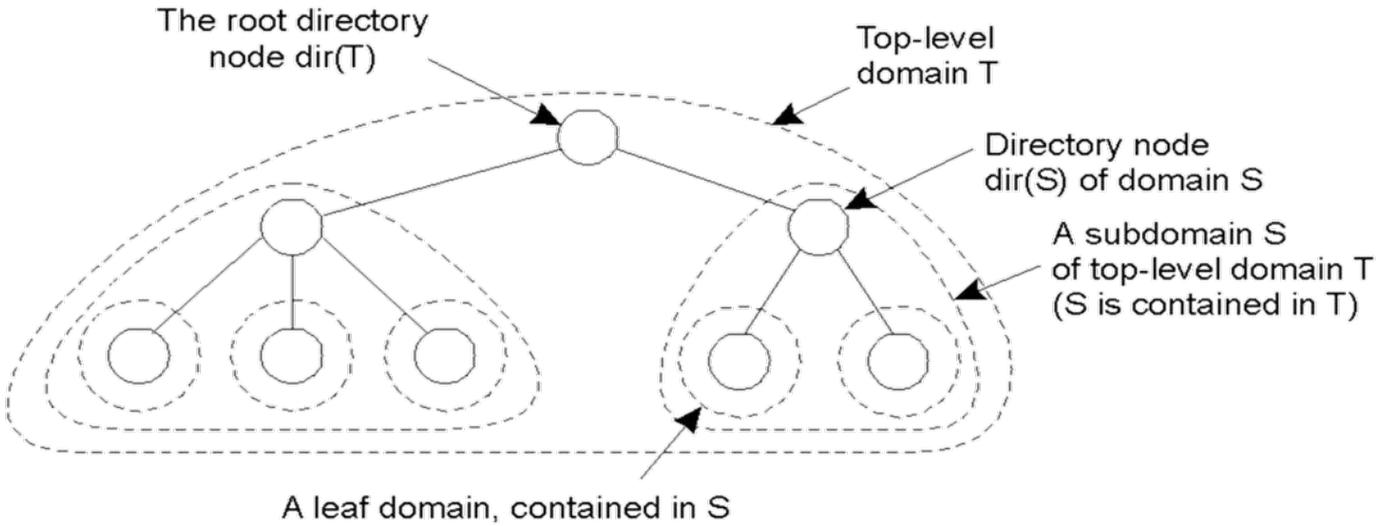


- Routing table size (with $N=2^m$ nodes)?
  - Log N fingers
- Routing time?
  - Each hop expects to 1/2 the distance to the desired id => expect O(log N) hops

The ring cannot reflect the fisical network, so we cannot optimize in that way

Hierarchical approaches

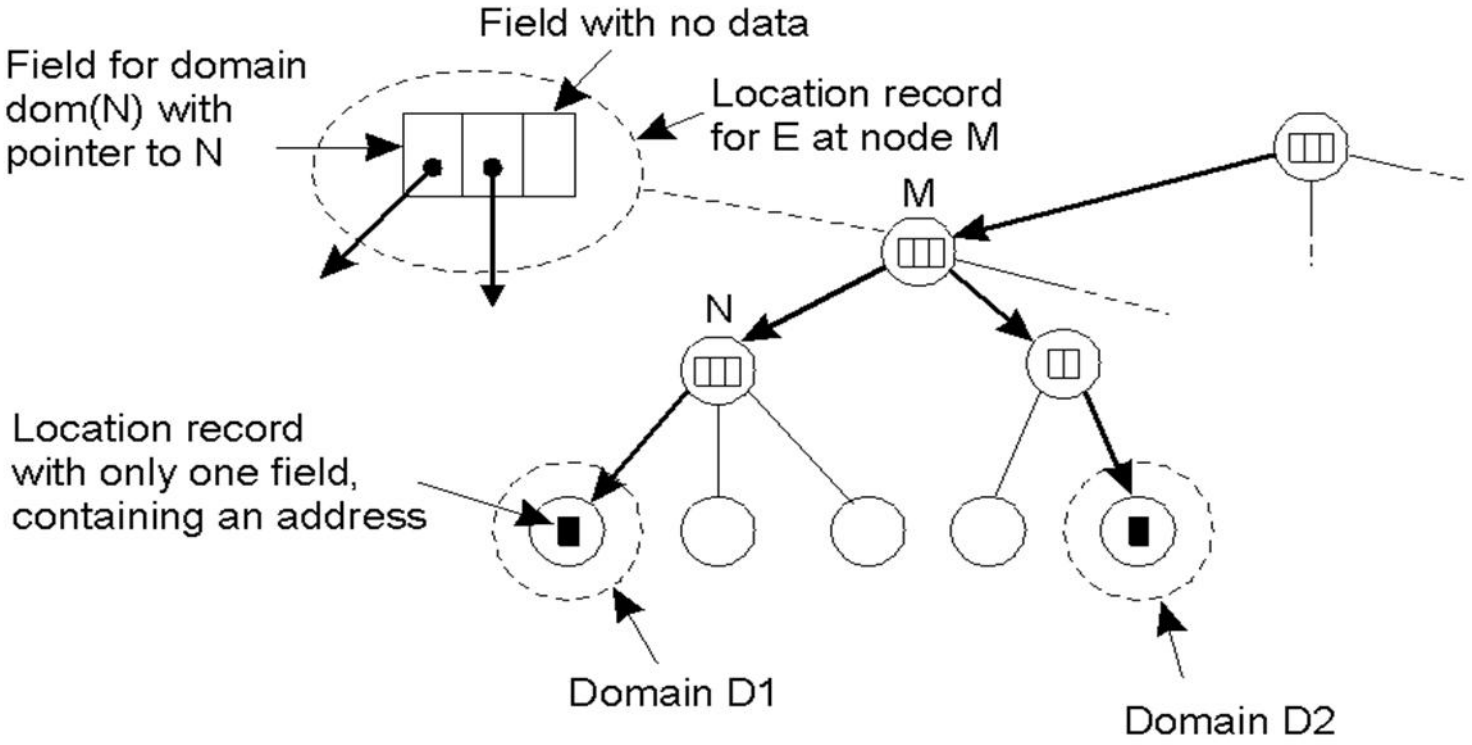
- Two-tier home-based approaches:
  - First look in a local registry to see if the entity is around
  - Otherwise, contact the home
- Hierarchical approaches generalize the above
  - The network is divided into domains
  - The root domain spans the entire network, while leaf domains are typically a LAN or a (mobile phone) cell
  - Each domain has an associated directory node that keeps track of the entities in that domain



The more you go up and the more your query becomes slow cause nodes have more data so there's the advantage that if there's locality it's not necessary to go very up (es: domain of politecnico, domain of milano, domain of lombardy), so if the structure of the network mirrors the fisical structure, the performances are improved. Another way to improve performances is with caching, meaning that intermediate nodes can save information about nodes after a lookup)

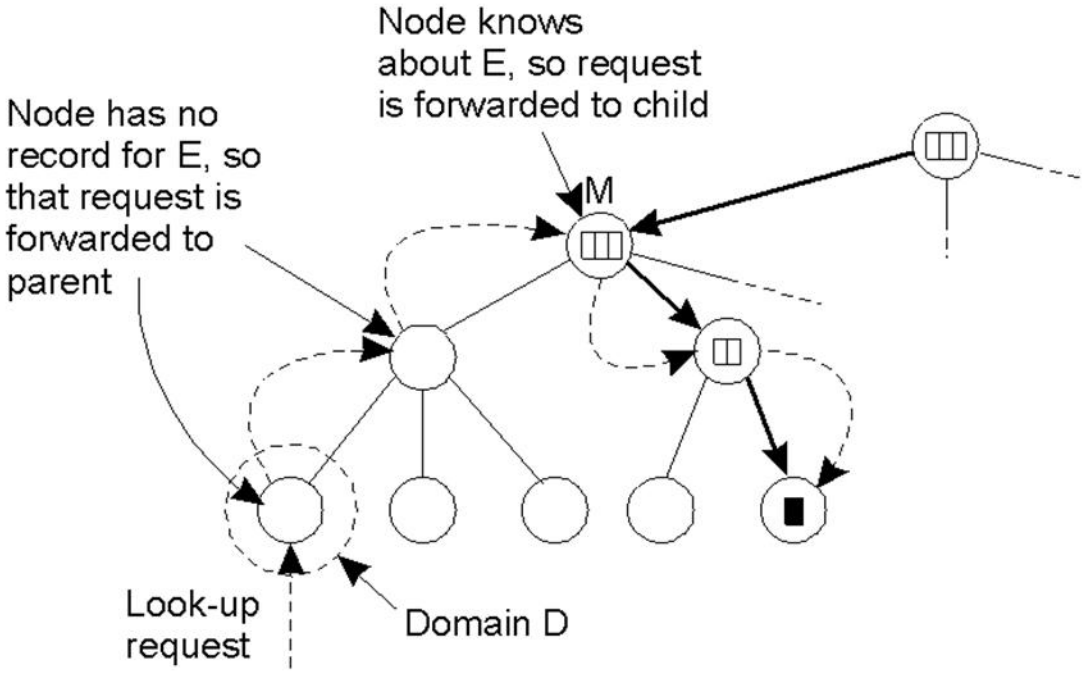
Hierarchical distribution of information

- The root has entries for every entity
- Entries point to the next sub-domain
- A leaf domain contains the address of an entity (in that domain)
- Entities may have multiple addresses (like entity E in figure above) in different leaf domains (replication)

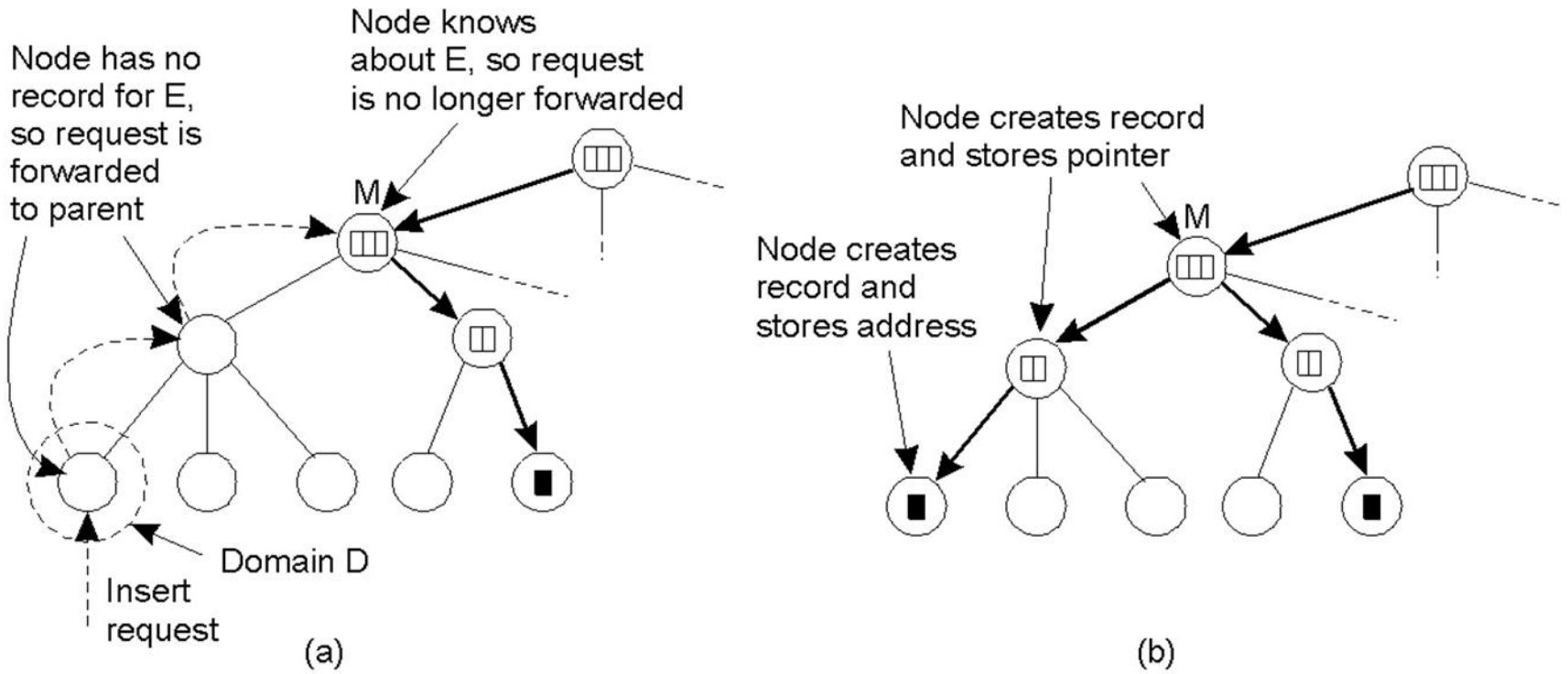


Hierarchical lookup

- Lookup may start anywhere, where client resides
- Look locally first: if not found, forward lookup to parent
- If entry found, forward lookup to child until a leaf holding the concrete entry is found
- Locality of queries is achieved (good)
  - Expanding ring searches
- Root has information on all entries (bad)

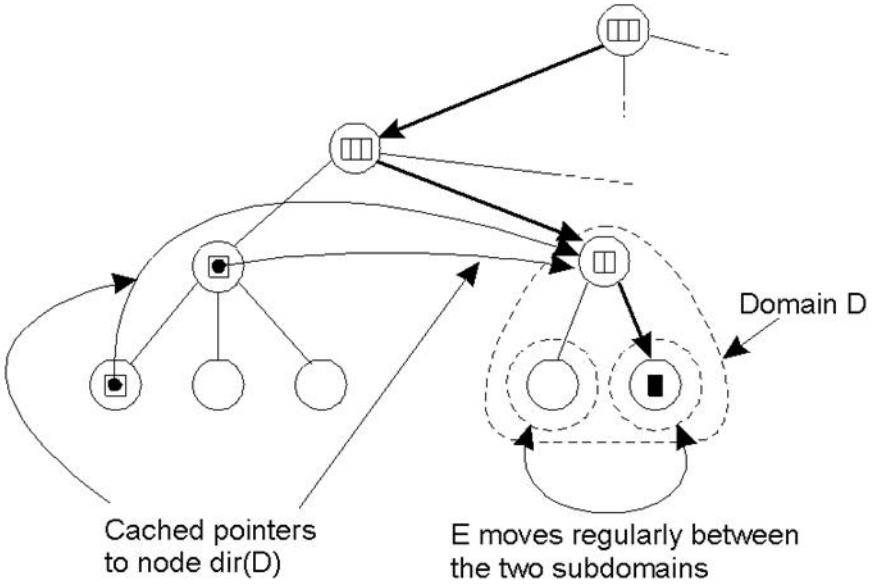


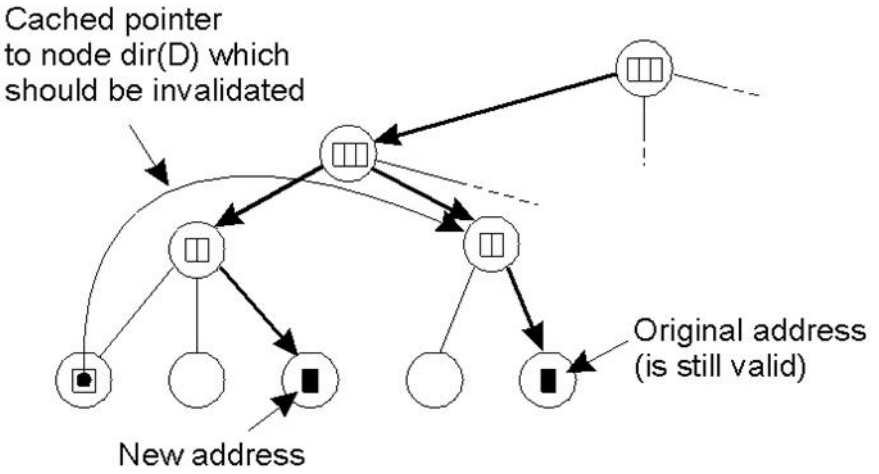
Hierarchical updates



- Updates start with insert request from new location
- Location records created top down or bottom up
  - Bottom up allows immediate queries
- Deleting proceeds from old node up, stopping when a node with multiple children nodes is reached

Optimizations and open issues





- Caching
  - Caching addresses directly is generally inefficient
  - It is possible to shortcut search if information about mobility patterns is available (see above)
- Scalability
  - Root node expected to hold data for all entities
  - Records are small, but lookups through root create bottleneck
    - The root can be distributed, but then allocation of entities to roots becomes tricky (e.g., if user is in the US, assigning it to the portion of the root maintained in Italy is inefficient)

Contents

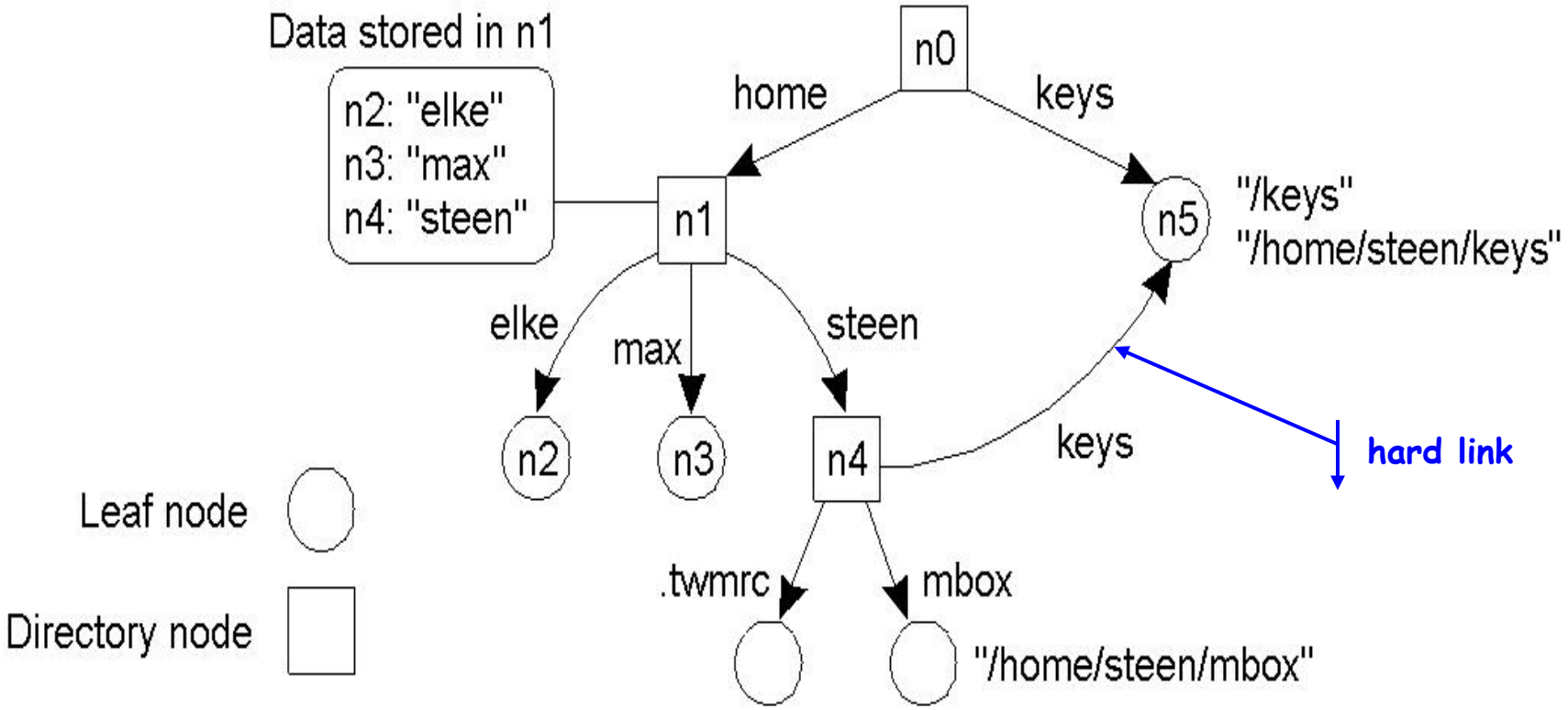
- Basics
  - Entities, names, addresses, identifiers
  - Name resolution
- Flat naming
  - Simple solutions
  - Home-based approaches
  - Distributed hash tables
  - Hierarchical approaches
- **Structured naming**
  - **Name spaces and name servers**
- Attribute based naming
  - Directory services, LDAP
- Removing unreferenced entities
  - Reference counting, reference listing, distributed mark-and-sweep

Structured naming

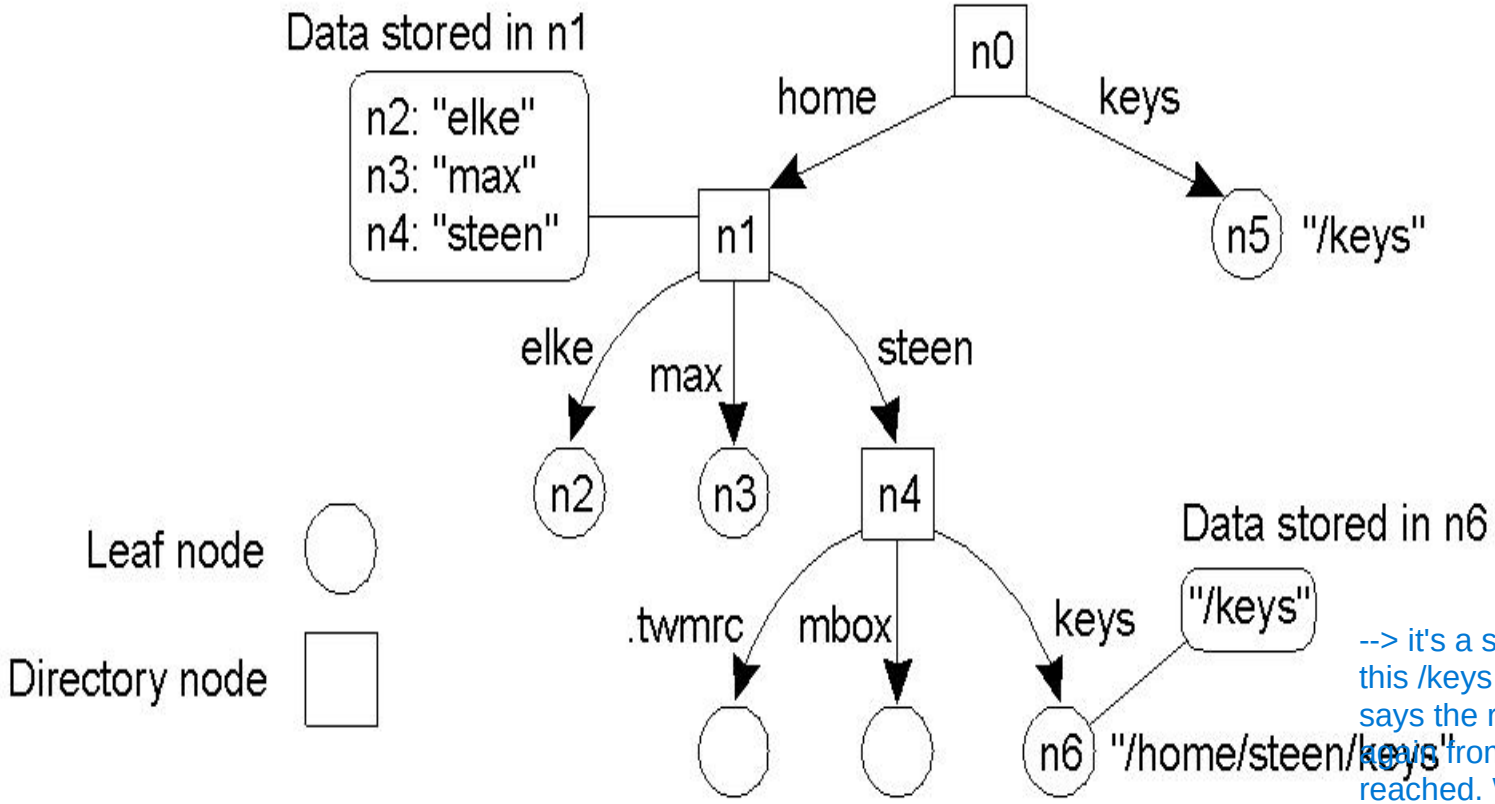
- In a structured naming system names are organized in a *name space*
- A name space is a labeled graph composed of *leaf nodes* and *directory nodes*
- A leaf node represents a *named entity*
  - It stores information about the entity it refers to. They include, at least, its identifier/address
- A directory node has a number of labeled outgoing edges, each pointing to a different node
  - A node is a special case of an entity. It has an identifier/address
- Resources are referred through *path names*
  - E.g., <*alpha*, *beta*, *gamma*> or simply "*/alpha/beta/gamma*"
- Path names can be *absolute* or *relative*
- Multiple path names may refer to the same entity (*hard linking*) or...
- ...leaf nodes may store absolute path names of the entity they refer to instead of their identifier/address (*symbolic linking*)

or soft link

Example: File systems as name spaces



Example: File systems as name spaces



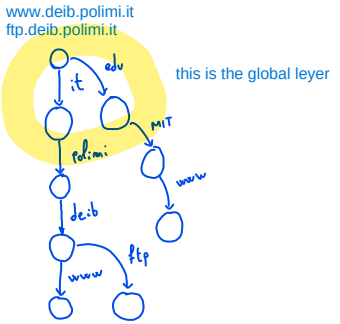
symbolic link

--> it's a soft link
this /keys is not the real destination, it only says the right thing to check. So after checking again from n0, the real destination can be reached. Windows uses those soft links, also DNS does.

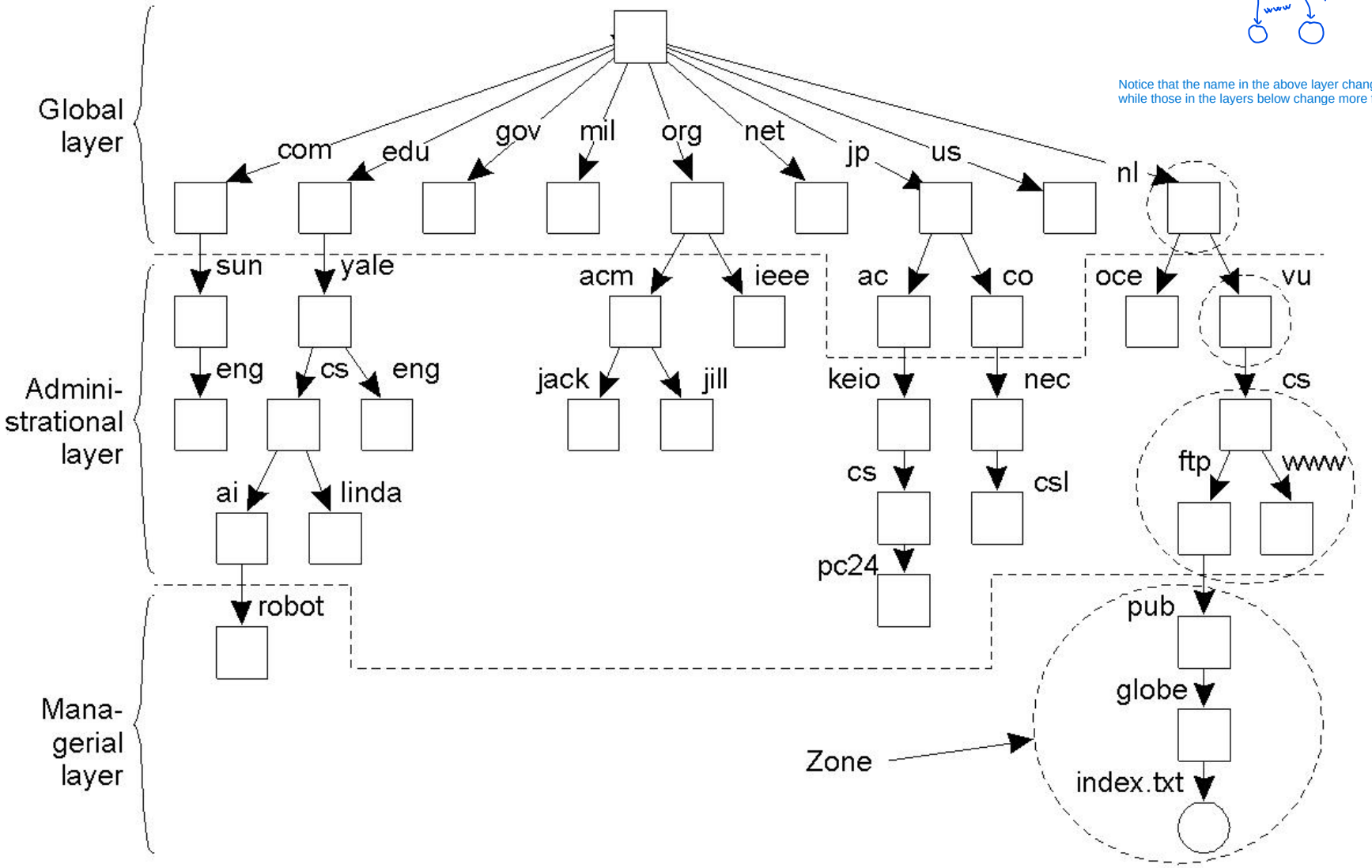
Name space distribution

- Name spaces for a large scale, possibly worldwide, distributed system are often distributed among different *name servers*, usually organized hierarchically Each of the serve take a piece of the name space. Also machines are organized in a tree
- The name space is partitioned into *layers*...
  - *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
- ...and each node of the name space is assigned to a name server

An example: The DNS



Notice that the name in the above layer change very rarely, while those in the layers below change more frequently



Comparison among 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

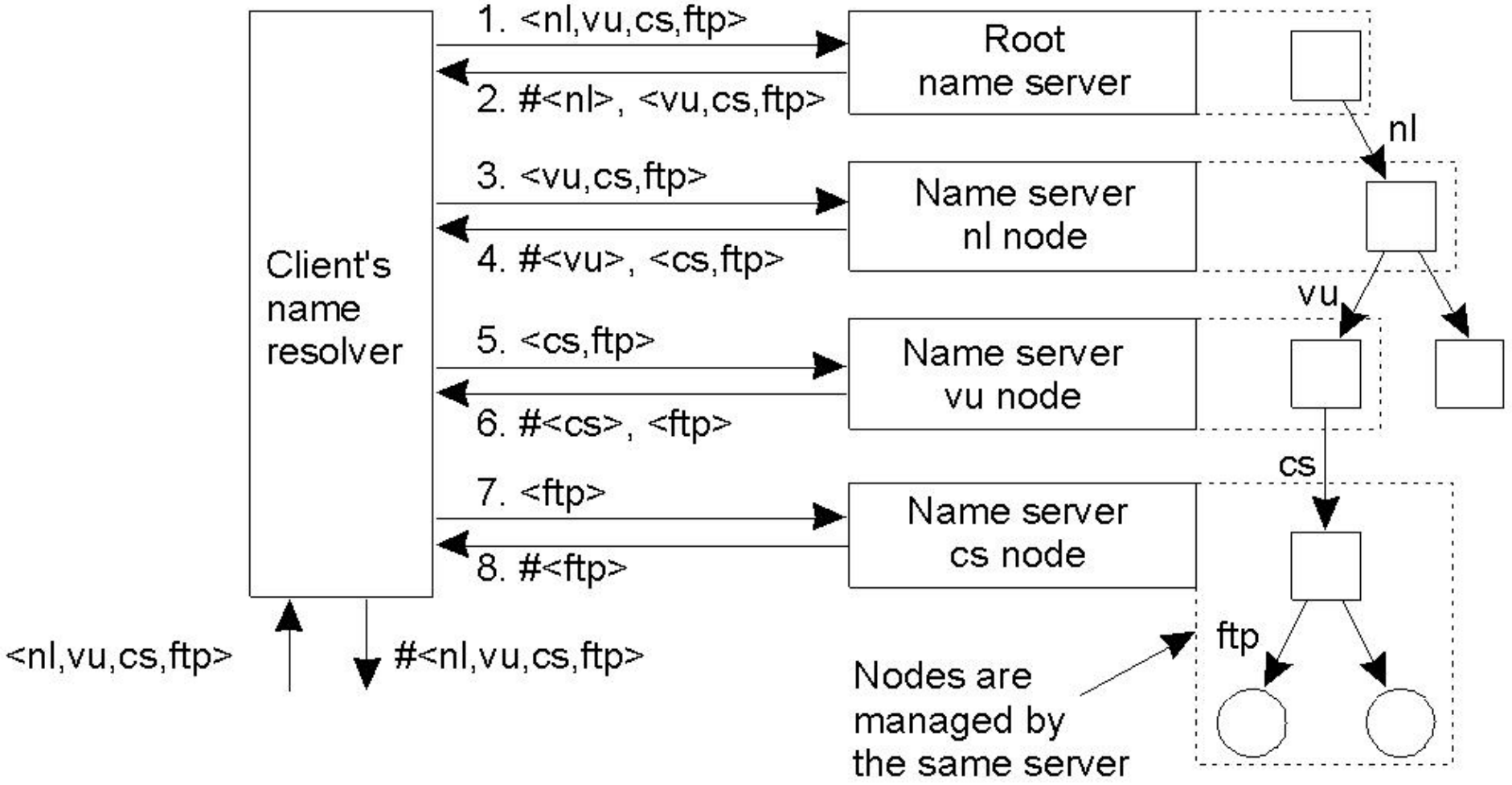
why so much?
few names, very stable --> you resolve global names one and then cache in your machine, because they probably won't change

How changes (updates) are propagated?

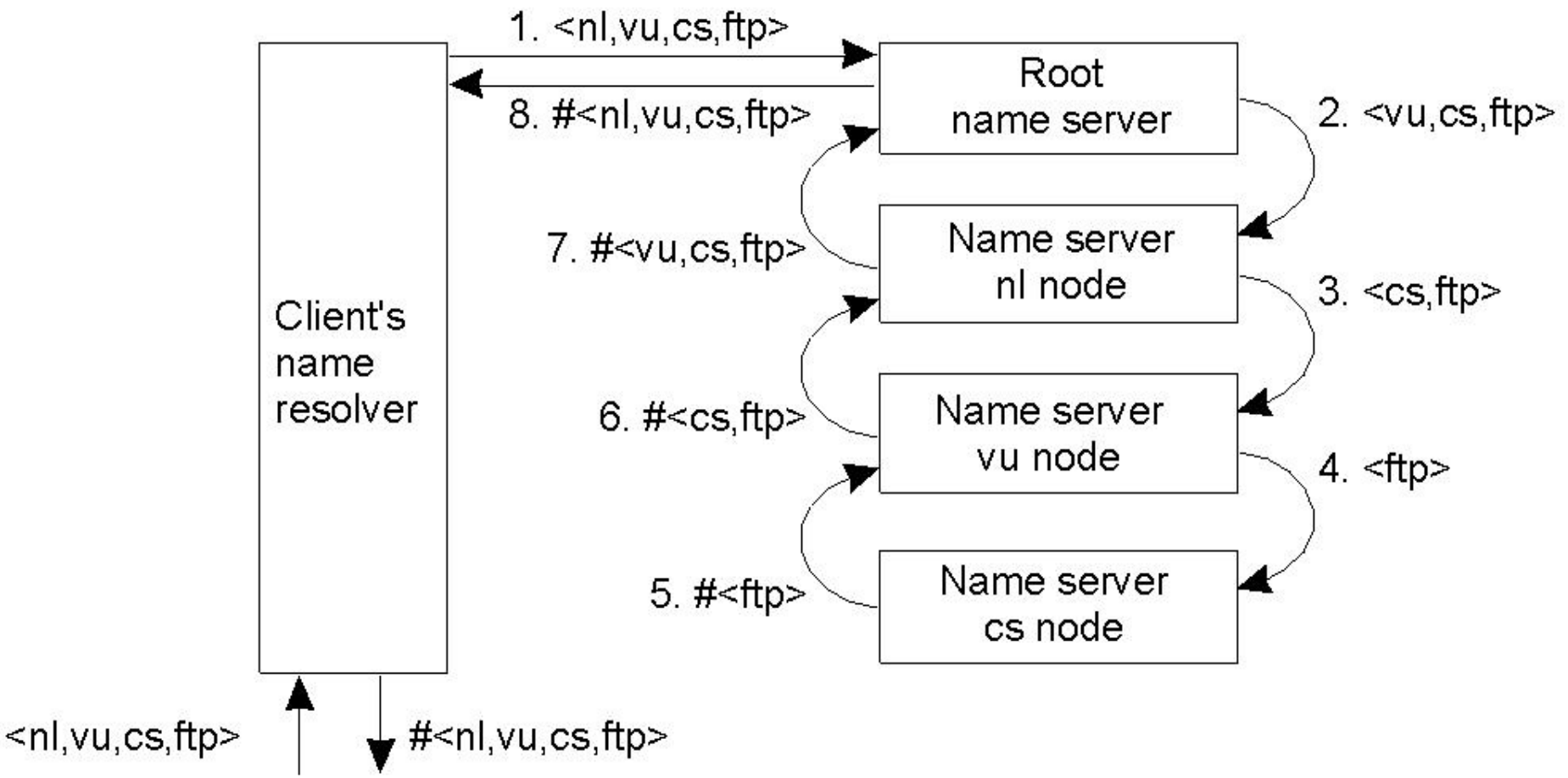
Take very long (hours) to update some names, because it's a worldwide change

How many times can I replicate the server that holds the names?

Iterative name resolution

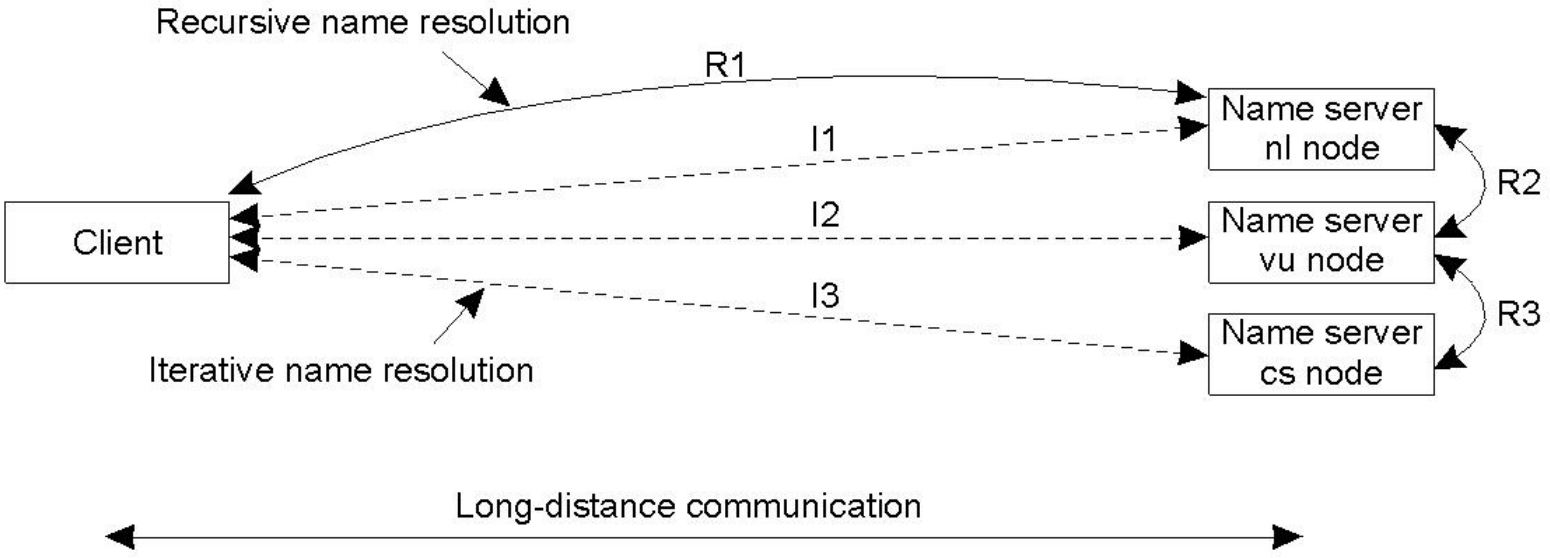


Recursive name resolution



Notice that while using flat names we can optimize performances by using locality (so in many cases will not be the necessity to go up to the root), in structured names the name space is distributed without any type of locality, so it will be always
necessary to go up to the root. However, structured names are in any case more efficient (thanks to cachinng)

Tradeoffs



- Pros of recursive vs. iterative resolution
  - Communication costs may be reduced
  - Caching is more effective because it can be performed along the whole resolution chain, leading to faster lookups
- Cons:
  - Higher demand (suspended threads) on each name server

DNS structure in practice

- Name space hierarchically organized as a rooted tree (no hard links)
  - Each subtree is named “domain”, and belongs to a separate authority
  - Each name server is responsible for a zone
- Name is case-insensitive, up to 255 characters, max 63 characters per “label”
- Each node may represent several entities through a set of resource records:

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 (symbolic link) to a mail server to handle mail addressed to this node
SRV	Domain	Refers (symbolic link) to a server handling a specific service, e.g., http
NS	Zone	Refers (symbolic link) 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, for reverse lookups
HINFO	Host	Holds information on the host this node represents
TXT	Any kind	Contains any entity-specific information considered useful

Sample DNS records

Name Servers

Mail Servers (prioritized)

2 Network Interfaces (redundancy)

Mail Server Info (with backup)

Web and FTP (same machine)

Network Printer

For Inverse Mapping

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

for mail

hardware info

canonical name

address

if i want to send a mail to mario.rossi@cs.vu.nl I need to reach one of the 3 servers (with a priority): zephyr ecc.
Notice that those are soft links

DNS resolution in practice

- Clients can request the resolution mode, but servers are not obliged to comply
- In practice, a mixture of the two is used
  - Global name servers typically support only iterative resolution
- Global servers are mirrored, and *IP anycast* is used to route queries among them
- Caching and replication are massively used
  - Secondary DNS servers are periodically (e.g., twice per day) brought up-to-date by the primary ones
  - A TTL attribute is associated to information, determining its persistence in the cache
- Transient (days at the global level) inconsistencies are allowed
- Only host information is stored, but in principle other information could be stored

13 Root Servers

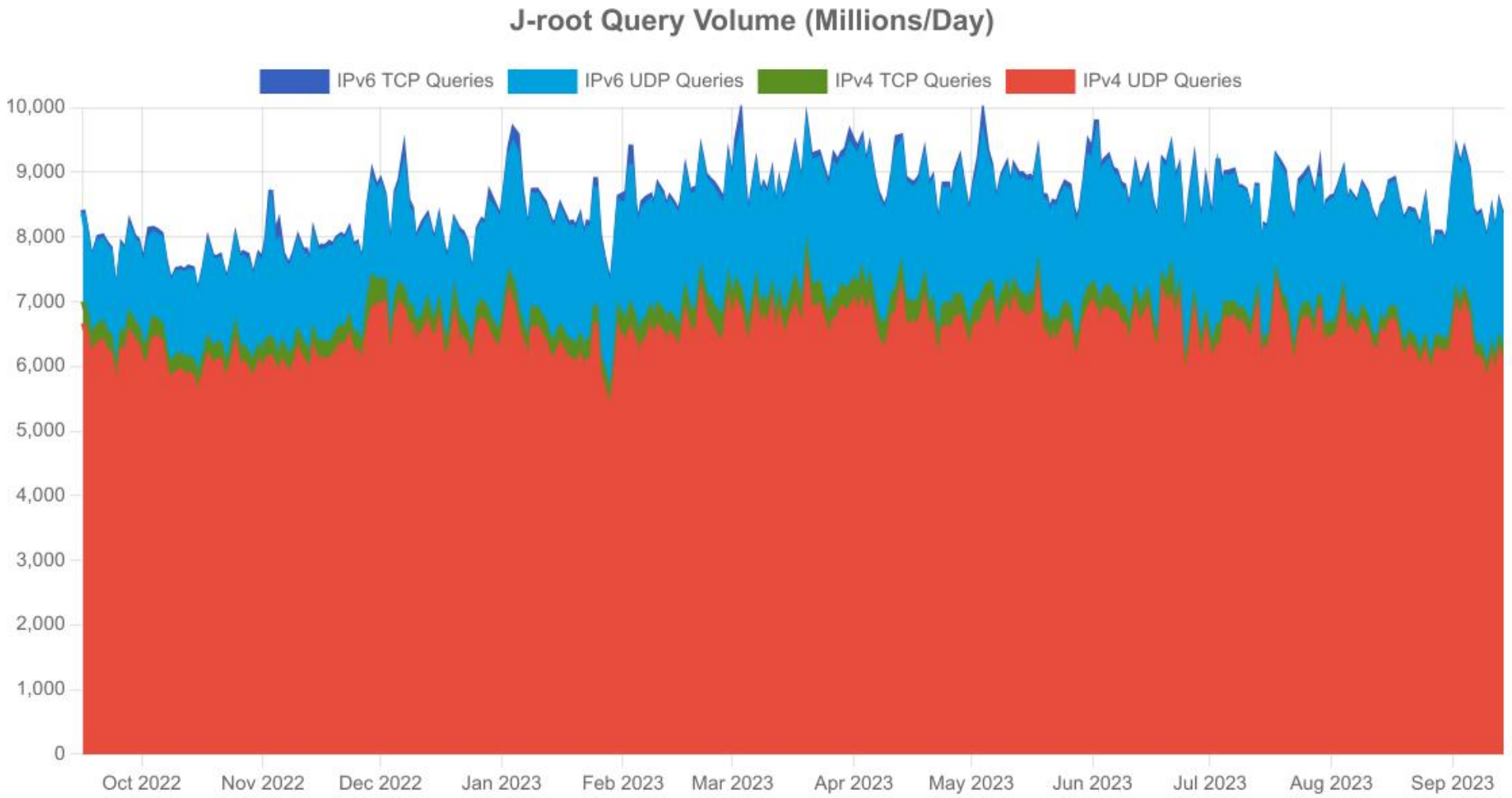
As of 2023 the root server system consists of 1751 instances operated by the 12 independent root server operators.

see www.root-servers.org

13 Root Servers

HOSTNAME	IP ADDRESSES	OPERATOR
a.root-servers.net	198.41.0.4, 2001:503:ba3e::2:30	Verisign, Inc.
b.root-servers.net	199.9.14.201, 2001:500:200::b	University of Southern California, Information Sciences Institute
c.root-servers.net	192.33.4.12, 2001:500:2::c	Cogent Communications
d.root-servers.net	199.7.91.13, 2001:500:2d::d	University of Maryland
e.root-servers.net	192.203.230.10, 2001:500:a8::e	NASA (Ames Research Center)
f.root-servers.net	192.5.5.241, 2001:500:2f::f	Internet Systems Consortium, Inc.
g.root-servers.net	192.112.36.4, 2001:500:12::d0d	US Department of Defense (NIC)
h.root-servers.net	198.97.190.53, 2001:500:1::53	US Army (Research Lab)
i.root-servers.net	192.36.148.17, 2001:7fe::53	Netnod
j.root-servers.net	192.58.128.30, 2001:503:c27::2:30	Verisign, Inc.
k.root-servers.net	193.0.14.129, 2001:7fd::1	RIPE NCC
l.root-servers.net	199.7.83.42, 2001:500:9f::42	ICANN
m.root-servers.net	202.12.27.33, 2001:dc3::35	WIDE Project

Statistics for j.root-servers.org



https://j.root-servers.org/metrics

DNS and mobile entities

- DNS works well based on the assumptions that :
  - Content of global/administrational layers is quite stable
  - Content of managerial layer changes often, but requests are served by name servers in the same zone, therefore updates are efficient
- What if a host is allowed to “move”?
- Not a problem if it stays within the original domain: Just update the database of the domain name servers
  - E.g., moving the FTP server *ftp.elet.polimi.it* to a different machine affects only the DNS servers for *elet.polimi.it*
- If a host (e.g., *ftp.elet.polimi.it*) moves to an entirely different domain (e.g., *cs.wustl.edu*)
  - Better keep its name because applications and users are relying on it
  - DNS servers of *elet.polimi.it* could provide the IP address of the new location
    - Lookups not affected
    - Further updates no longer “localized” (managerial layer no longer efficient)
  - DNS servers of *elet.polimi.it* could provide the name of the new location (essentially a symbolic link). Similar to the forwarding pointers approach
    - Lookups less efficient: essentially two distributed lookups are required
    - Further updates not affected
- Problem exacerbated for truly mobile entities with continuously moving entities DNS would not scale

Contents

- Basics
  - Entities, names, addresses, identifiers
  - Name resolution
- Flat naming
  - Simple solutions
  - Home-based approaches
  - Distributed hash tables
  - Hierarchical approaches
- Structured naming
  - Name spaces and name servers
- **Attribute based naming**
  - **Directory services, LDAP**
- Removing unreferenced entities
  - Reference counting, reference listing, distributed mark-and-sweep

Attribute based naming

- Problem: As more information is made available, it becomes important to effectively search for items
- Solution: Refer to entities not with their name but with a set of attributes, which code their properties
- Each entity has a set of associated attributes
- The name system can be queried by searching for entities given the values of (some) of their attributes
  - More entities can be returned
- Attribute based naming systems are usually called *directory services*
- They are usually implemented by using DBMS technology

Structured, attribute based naming: The LDAP case

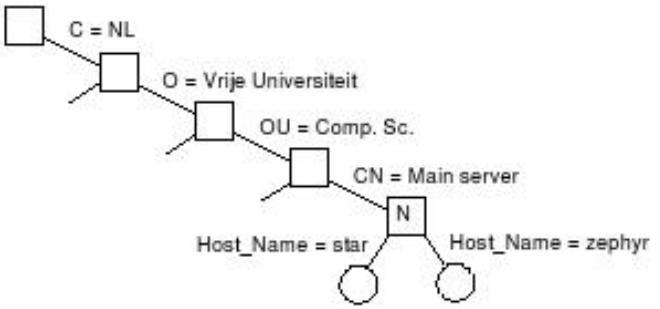
- A common approach to implement distributed directory services is to combine structured with attribute based naming
- The LDAP (lightweight directory access protocol) directory service is becoming the de-facto standard in this field
  - Derived from the OSI X.500 directory service
- An LDAP directory consist of a number of records (*directory entries*)
  - Each is made as a collection of <*attribute*, *value*> pairs
  - Each attribute has a type
  - Both single-valued and multiple-valued attributes exist
- Some attributes are part of the standard

The Directory Information Base

- The collection of all records in a LDAP directory service is called *Directory Information Base – DIB*
- Each record has a unique name
  - Defined as a sequence of naming attributes (aka *relative distinguished name – RDN*)
  - E.g., /C=NL/O=Vrije Universiteit/OU=Comp. Sc./CN=Main server
- This approach leads to build a *directory information tree* - *DIT*
  - A node in a LDAP naming graph can thus simultaneously represent a directory in a traditional sense (i.e., in a hierarchical name space)

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



More on attributed-based naming

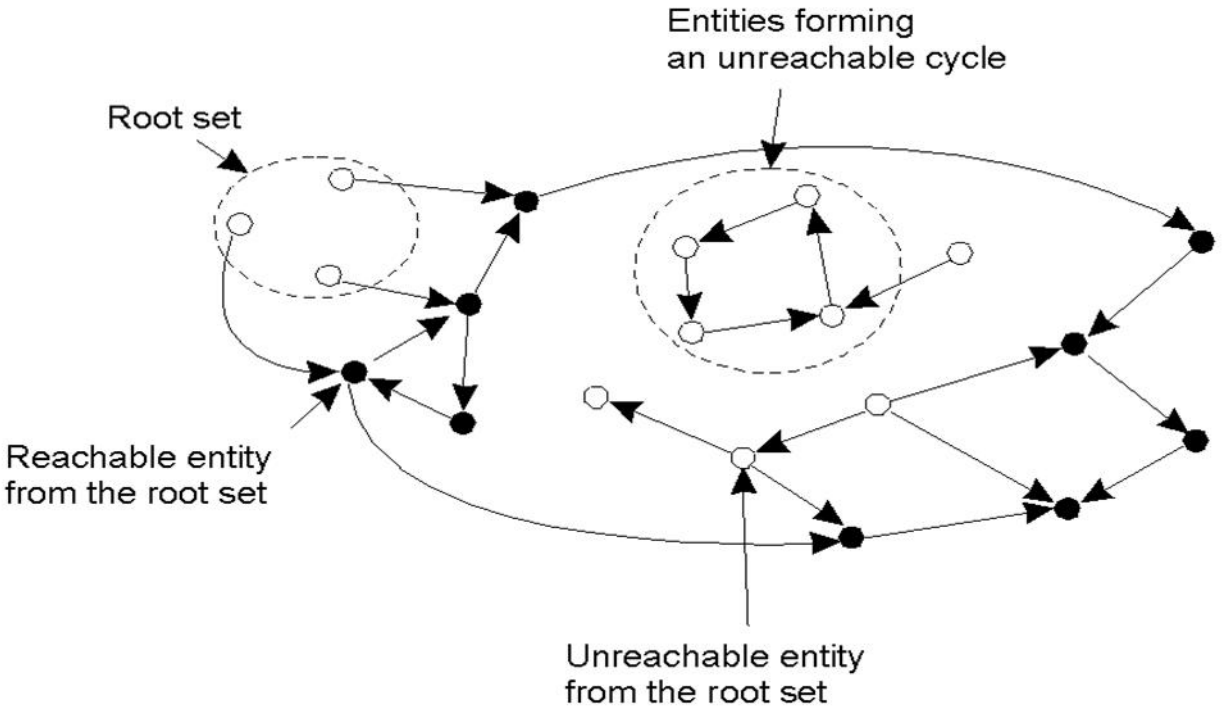
- When dealing with large scale directories, the DIB is usually partitioned according to the DIT structure (as in DNS)
  - Each server is known as a *Directory Service Agent – DSA*
  - The client is known as *Directory User Agent – DUA*
- What LDAP adds to a standard hierarchical naming schema is its searching ability
  - E.g. search("&(C=NL)(O=Vrije Universiteit)(OU=*)(CN=Main server)")
  - In general several DSA have to be accessed to resolve a query
- Microsoft's Active Directory is based on LDAP together with several other technologies
- Another directory service that is becoming a standard has been developed in the context of grid computing and web services and is know as *Universal Directory and Discovery Integration - UDDI*

Contents

- Basics
  - Entities, names, addresses, identifiers
  - Name resolution
- Flat naming
  - Simple solutions
  - Home-based approaches
  - Distributed hash tables
  - Hierarchical approaches
- Structured naming
  - Name spaces and name servers
- Attribute based naming
  - Directory services, LDAP
- **Removing unreferenced entities**
  - **Reference counting, reference listing, distributed mark-and-sweep**

Removing unreferenced entities

- Naming provide a global referencing service
- Entities accessed through stale bindings should be removed
- Automatic garbage collection is common in conventional systems
- Distribution greatly complicates matters, due to lack of global knowledge about who's using what, and to network failures
- Practical importance in distributed object platforms (e.g., RMI)



Reference counting

- The object (e.g., in its skeleton) keeps track of how many other objects have been given references --> so it's easy (just keep a number)

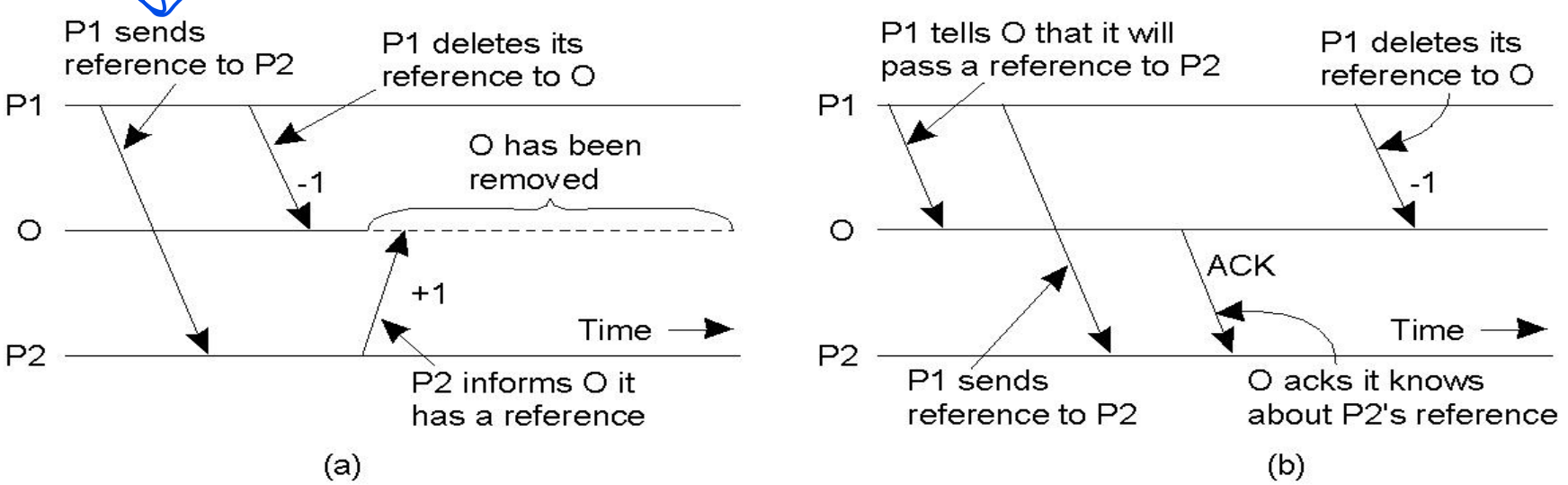
cons:

- Reliability (exactly-once message delivery) must be ensured --> sender and receiver must agree on the delivering of the message --> theoretically impossible in distributed systems, so acks and other techniques are implemented.
  - Typically, acknowledgments and duplicate detection and elimination

- Race condition when passing references among processes (not present in non-distributed systems):

With reference counting you don't solve cyclic reference problem

Race condiiton fixing: P1 tells the object what's he's gonna do



Can be that object O receive the decrementing (but not the increment by P2), so he thinks he can kill itself

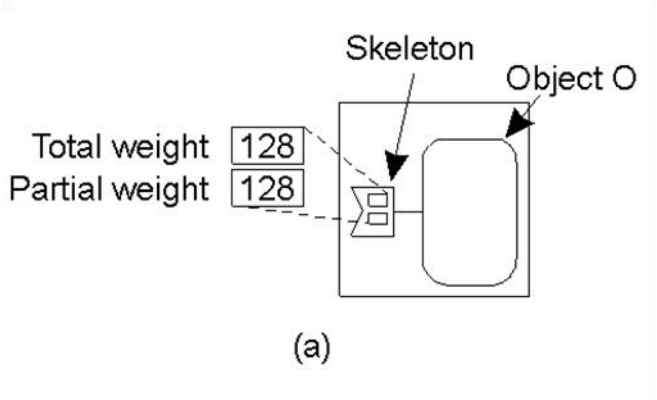
- Passing a reference requires now three messages: Potential performance issue in large-scale systems

To solve reference counting problems they invented

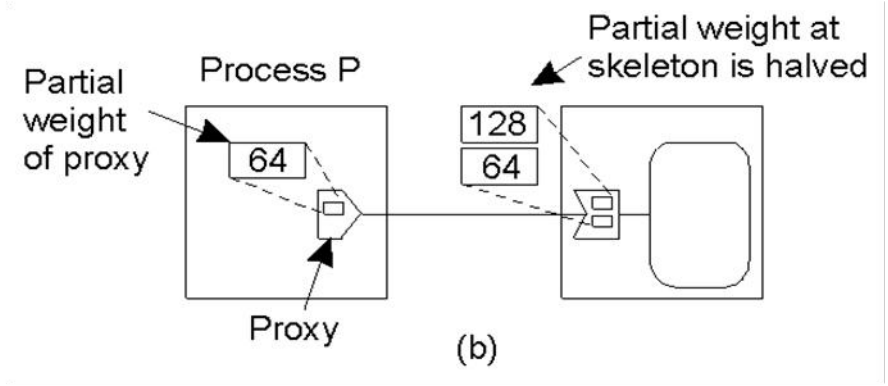
Weighted reference counting

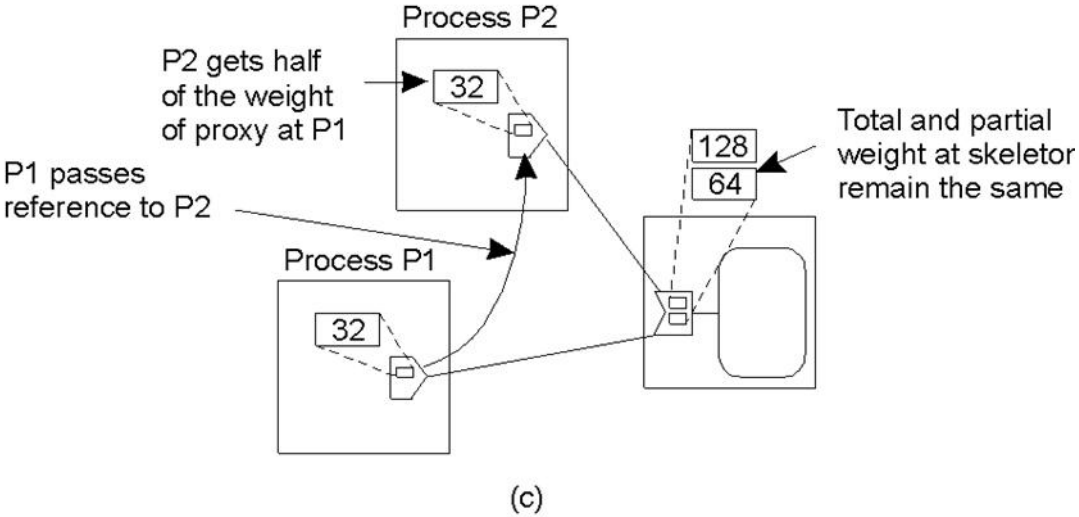
Two numbers instaed of only one

This approach solves race condition but not the exactly one message problem and the cyclic reference problem

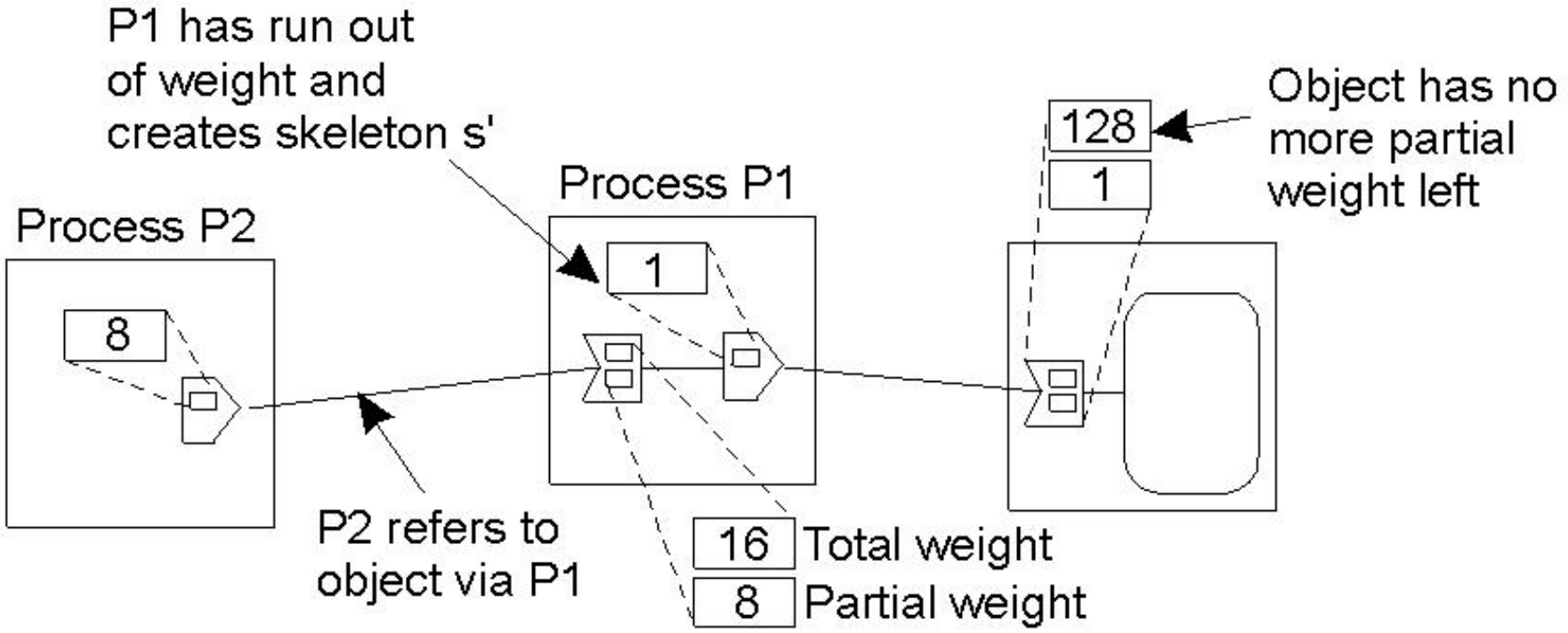


- Tries to circumvent the race condition by communicating only counter decrements
- Requires an additional counter
- Removing a reference subtracts the proxy partial counter from the total counter of the skeleton: when the total and partial weights become equal, the object can be removed
- Problem: only a fixed number of references can be created





Weighted reference counting: Using indirection



- Removes the limitation on the overall number of references, but introduces an additional hop to access the target object

Reference listing

- Instead of keeping track of the number of references, keep track of the identities of the proxies
- Advantage 1: insertion/deletion of a proxy is *idempotent*
  - Insertion and deletion of references must still be acknowledged, but requests can be issued multiple times with the same effect
  - Non-reliable communication can be used
- Advantage 2: easier to maintain the list consistent w.r.t. network failures
  - E.g., by periodically “pinging” clients (potential scalability problem)
- Still suffers from race conditions when copying references
- Used by Java RMI
  - A client creates a local proxy only after it received from the server an acknowledgment to its insertion request
  - Network failures are dealt with using leases: it is up to the client to “refresh” the reference (potential problems for long failures)
  - The lease interval is determined by run-time and non-negotiable
  - UDP is used as a transport layer for garbage collection

Identifying unreachable entities

- How to detect entities disconnected from the root set?
- Tracing-based garbage collection techniques: require knowledge about all entities, therefore they have inherent poor scalability
- *Mark-and-sweep* on centralized systems Not efficient cause it needs to stop the distributed systems
  - First phase marks accessible entities by following references
    - Initially all nodes white
    - A node is colored grey when reachable from a root (but some of its references still need to be evaluated)
    - A node is colored black after it turned grey and all its outgoing references have been marked grey
  - Second phase exhaustively examines memory to locate entities not marked and removes them
    - Garbage collects all white nodes

Distributed mark-and-sweep

- Emerald distributed object system [Jul et al., 1988]
- Garbage collectors run on each site
  - Manages proxies, skeletons, and actual objects, initially all marked white
- Marking process:
  - An object in process P, and reachable from a root in P, is marked gray together with all the proxies in it
  - When a proxy q is marked grey, a message is sent to its associated skeleton (and object)
    - An object is marked gray as soon as its skeleton is
    - Recursively, all proxies in the object are marked grey
  - An acknowledgment is expected before turning q black
- Sweep process:
  - All white objects are collected locally
  - Can start as soon as all local objects are either black or white
- Requires reachability graph to remain stable!
  - Distributed transaction blocking the entire system → scalability issues