# Distributed Systems

## Naming

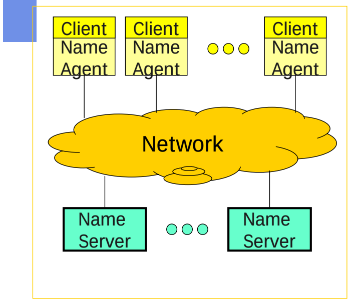
Lecture 8

June 7 2004

## Schedule of Today

- Motivation, Notions, Objectives
- Naming Entities
  - Names, Identifiers, and Addresses
  - Name Resolution
  - Implementation of a Name Space
  - Examples: DNS and X.500
- Locating Mobile Entities
  - Naming versus Locating Entities
  - Simple Resolution
  - Home-based –Hierarchical Approaches
- Removing Unreferenced Entities
  - Unreferenced Objects
  - Reference Counting and Listing
  - Identifying Unreachable Entities

## Components of Name Service Architecture


Client Name Agent
Client Name Agent
Client Name Agent
Network
Name Server
Name Server

Name agent:
- Supplies interface between name service and client
- Cooperates with name servers in order to *generate, resolve, …* names
- Caches the results

Name server:
- Manages context information and implements mapping function
- Sometimes cooperate with other name servers
- Supplies an interface for the name agent

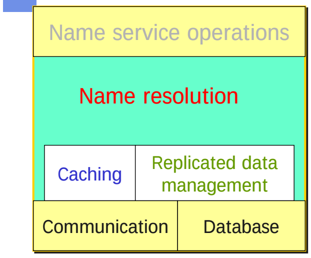
## Name Service Operations

Manipulation of context information:
- ADD: Create a reference: Name → Object
- DELETE: Delete a reference
- MODIFY: Change a reference

Queries:
- READ: Resolve a name into an object
- SEARCH: Find a name or object based on some attributes
- LIST: List all names

Administration:
- Assignment of access privileges
- Authentication
- Extension of the name space

## Name Server Architecture


Name service operations
Name resolution
Caching
Replicated data management
Communication
Database

Name Resolution
- mapping of names to objects

Caching
- of remote context information for efficient access

Replicated data management
- management of replicated context information

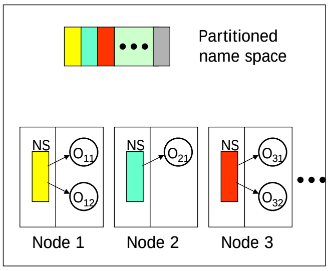
Communication
- between agents and name servers
- between name servers for cooperation purposes

Database
- management of local context information

## Resolution of Structured Names

- Resolution mechanism depends on type of name
- Location-dependent names
  - Name contains the location of the object it identifies
- Authority-dependent names
  - Name contains location of name server being responsible for name-resolution (the *authority*)
- Location-independent names
  - Name contains no location information

# Location-Dependent Names (1)



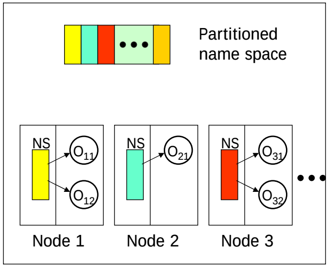
Structure of name space:

- Each node has own name server
- Name server only manages the names of local objects
- Name structure:
  `NodeID.ObjectID`

Names are generated:

- by the local name server
- ... through concatenation of NodeID and local ObjectID

# Location-Dependent Names (2)



Name resolution (of `N.O`):

- Name agent sends query to node N.
- Name server of N maps O to a local object

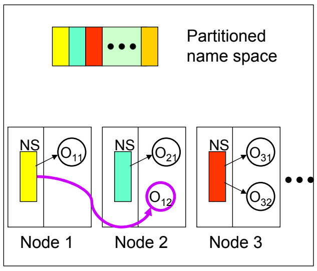
Properties:

+ Nodes are highly autonomous

+ Name authority can be found in a simple and efficient way

- Objects can not migrate

# Authority-Dependent Names (1)



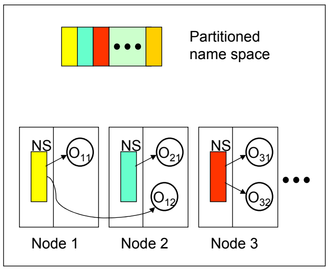
Structure of the name space:

- Every node has a name server
- Name server manages the names of all locally-created objects
- Objects may migrate
- Name server must be aware of the locations of all migrated objets
- Name structure: NodeID.ObjectID

Names are generated:

- ... by local name server
- ... through concatenation of the NodeID and local unique ObjectID

# Authority-Dependent Names (2)



Name resolution (of N.O):

- Name agent sends query to node N.
- Name server of N maps O to a local object

Properties:

+ Name generation is simple

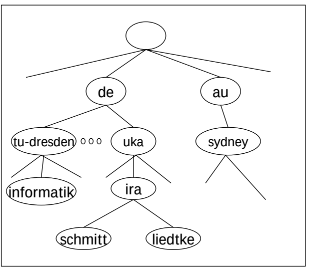
+ Name authority can be found in a simple and efficient way

+ Objects can migrate

- Additional overhead for the monitoring of migrated objects

- Only one (fixed) authority per object

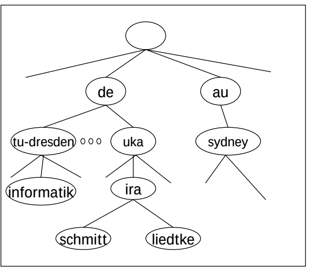
# Location-Independent Names (1)



Structure of the name space:

- Hierarchically structured NS
- Hierarchy can resemble:
  - organizational structure
  - system topology
  - geographical distribution
  - etc.

# Location-Independent Names (2)



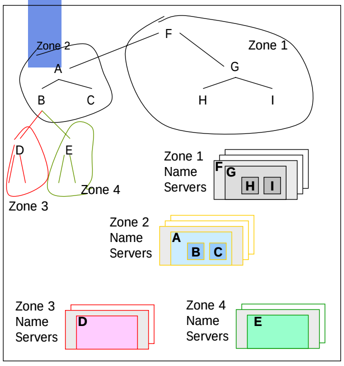
Relative name of an object:

- Identifies a name uniquely in context of its parent object
- Example:
  uka/ira/schmitt (parent = de)

Absolute name of an object:

- Globally unique identifier for each object
- Concatenation of the relative names of all predecessors
- Example:
  /de/uka/ira/liedtke

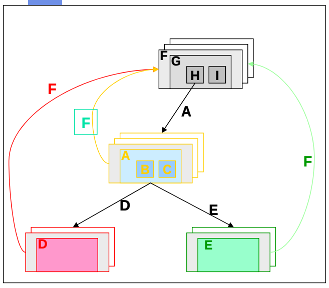
# Location-Independent Names (3)



Partitioning of name space:

- Hierarchical name space is split up into zones
- Each zone has at least one name server
- Name server manages name space covered by the zone
- Replication achieved by having multiple name servers per zone

# Location-Independent Names (4)



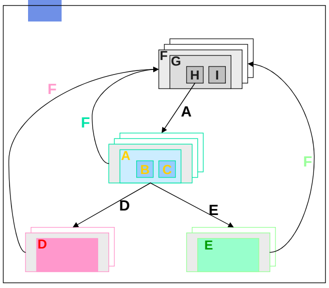
Every name server knows

- ... at least one server of root zone
- ... the root zone of every child zone and at least one name server of it
- Name server manages the name

Name resolution:

- Agent asks arbitrary name server N
- If N knows the name
  ⇒ local resolution
- If N does not know the name
  ⇒ Resolution starts with a root server
  ⇒ ... and works its way down the server hierarchy until the name has been completely resolved

# Location-Independent Names (5)



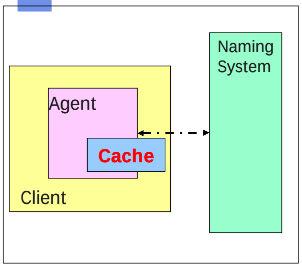
Name resolution

- ... is done by the authority responsible for the object

Properties:

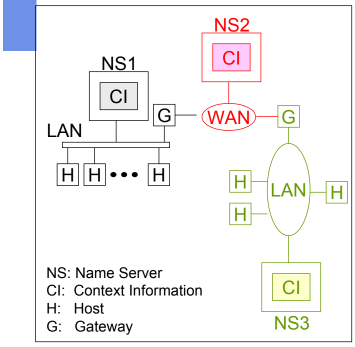
+ Objects can migrate
+ Authority for an object may change
+ Multiple authorities possible (replication)
- Resolution may be expensive (Efficiency improved by caching)
- Name generation may require communication

# Caching



- Cache contains mapping information
- On receipt of a request for resolution, agent checks if required information is already or still in the cache
  - Yes: Resolution is done locally
  - No: Agent sends a query to the naming system; its result is stored in the cache
- A cache entry is invalidated when after a resolution it is found to be stale

# Replication (1)



NS: Name Server
CI: Context Information
H: Host
G: Gateway

Goal:

- High availability, fast access

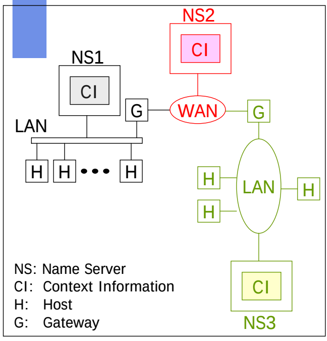
Pessimistic methods:

- One-copy serializability
- Limited availability if the network is partitioned

Optimistic methods:

- Weak consistency
- But: Available even if the network is partitioned

# Replication (2)



NS: Name Server
CI: Context Information
H: Host
G: Gateway

Weak consistency is sufficient:

If naming information is changed *infrequently*

Even if stale information is used, in many cases no harm will be done

If use of the information will reveal that it is stale

## Some Examples

- DNS: The Internet Domain Name System
- Jini: Network Technology
- GNS: Global Navigation System
- NIS - Network Information Service
- X.500
- LDAP - Lightweight Directory Protocol

## Domain Name System*

DNS = distributed database used by TCP/IP applications
to map between hostnames and IP addresses +
to provide electronic mail routing information

Primary task:

- Mapping from a symbolic name to the 32 bit Internet(IP) address, e.g.
  `mcculler.uni-karlsruhe.de` → **129……**
- General functionality:
- Mapping from hierarchical names to objects

*Paul Mockapetris (1984, standard in the Internet since 1987)

## Domain Name System

- Name space: tree, edges labeled
- Name of incoming edge identifies a node
- Each subtree is called a *domain*
- Sequence of edge labels to the root is a *domain name*
- Each node contains a *resource record*

## Domain Name System

Name space:
manages names for: computer, email-server, …

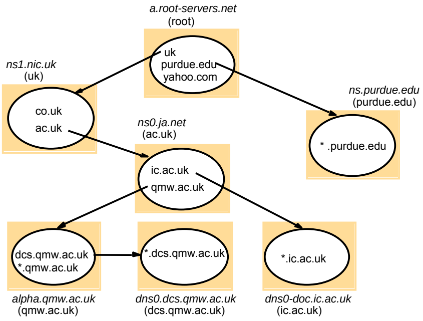
Name structuring:

- hierarchical name space
- pathnames start at the leave and end at the root, e.g. i30www.ira.uka.de
- Top level domain has fixed names (can be changed only by an official Internet-office) below top level domain you can have as many levels as you need

## DNS Top-Level Domain

| Domain Name | Meaning |
|---|---|
| com | Commercial bussiness |
| edu | Universities (colleges) in USA |
| gov | Government departments(USA) |
| mil | Military institutions |
| net | Netprovider |
| org | All other business |
| arpa | Temporal ARPA-domain |
| int | International organisations |
| Zip code of Country(e.g. de) | Abbreviations of all countries |

## DNS Name Servers



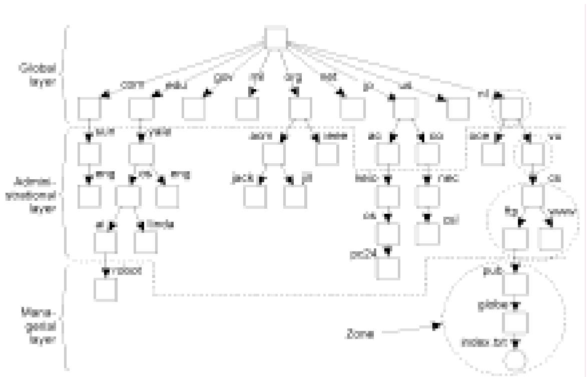
*Note*: Name server names are in italics, and the corresponding domains are in parentheses.

# DNS 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 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 (OS + HW-type) this node represents |
| TXT | Any kind | Contains any entity-specific information considered useful |

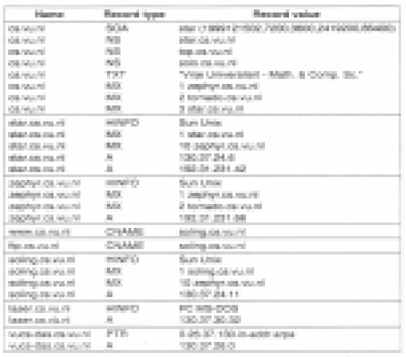
- Most important types of resource records forming the contents of nodes in the Internet's DNS name space.

# DNS Name Space



# DNS Implementation

- 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

| 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.

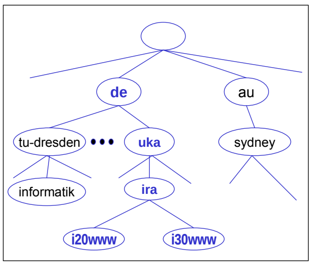
# Replication and Caching in DNS

- Replication
  - for every root server there are at least 2 replicas
    - primary/backup principle
    - backup servers update their state periodically via primary server
- Caching
  - each name server implements caching

Further reading:
F. Halsall: "Data Communications, Computer Networks and Open System",
Addison-Wesley 1992

D. Comer: "Computernetzwerke und Internets", Prentice Hall 1997

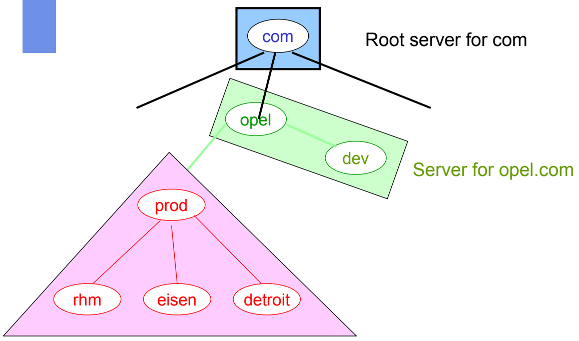
# DNS Name Space



- Hierarchical, location-independent names
- Name space is split up into domains being ordered in a *tree-like fashion*
- Each domain receives a domain name being *unique* among its sibling nodes
- Absolute name of a domain is obtained by concatenating the relative names on the path from this domain to the root of the tree

Example: i30www.ira.uka.de

# DNS Name Server Hierarchy



com

Root server for com

opel

dev

Server for opel.com

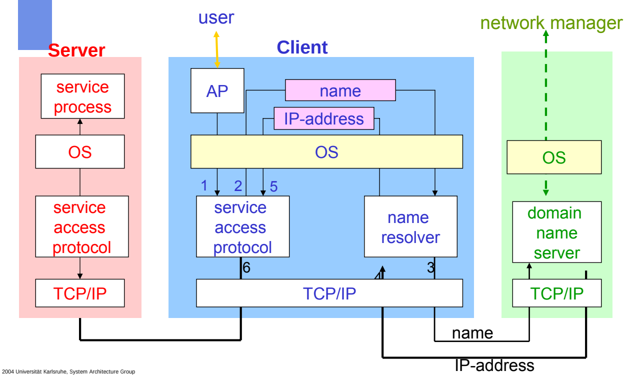
prod

rhm

eisen

detroit

# DNS System Architecture



user

network manager

Server

Client

service process

OS

service access protocol

TCP/IP

AP

name

IP-address

OS

1 2 5

service access protocol

name resolver

6 4 3

TCP/IP

OS

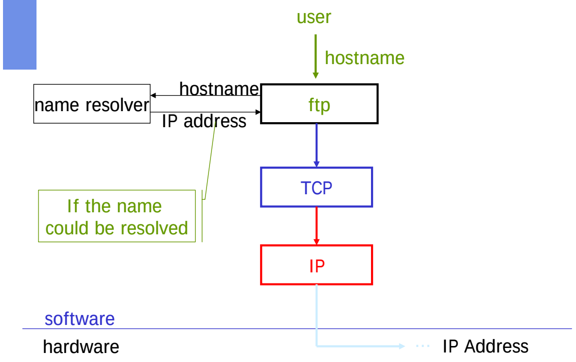
domain name server

TCP/IP

name

IP-address

# Example



user

hostname

name resolver

hostname

IP address

ftp

TCP

If the name could be resolved

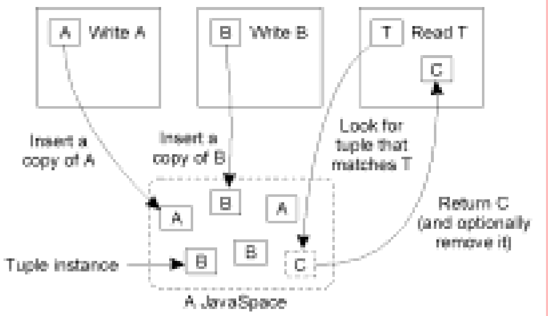
IP

software

hardware

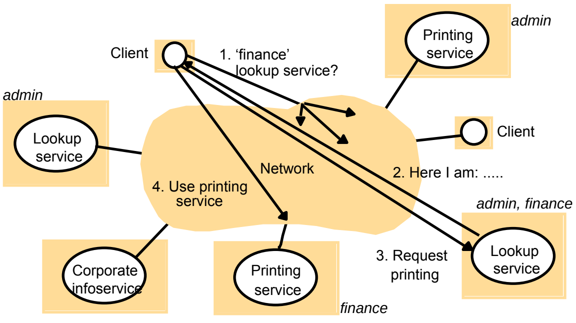
IP Address

# Overview on Jini



- The general organization of a JavaSpace in Jini.

# Service Discovery in Jini



admin

Printing service

Client

1. 'finance' lookup service?

admin

Lookup service

Client

Network

2. Here I am: .....

4. Use printing service

admin, finance

3. Request printing

Lookup service

Corporate infoservice

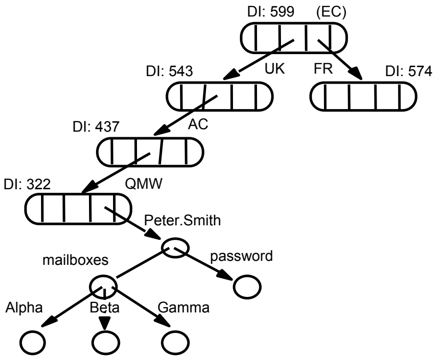
Printing service

finance

# Books on Jini

- "Jini Technology: An Overview" by Ilango Kumaran Prentice Hall PTR (November 2001), "For all managers, architects and consultants seeking to evaluate Jini from a technological and architectural perspective, and compare it with other work."
- " Developing Jini Applications Using Java 2 " by Hinkmond Wong Addison Wesley Longman (September 2001), "This book is your key to understanding and avoiding common traps and pitfalls that await developers approaching Jini and J2ME technology for the first time. "

# Global Name Service (GNS)



# Directory Service: X 500

CCITT and ISO standard (1988):
Names

- List of tuples <attribute = value)
- Attributes

| | |
|---|---|
| - country | “c” |
| - organization | “o” |
| - organizational unit | “ou” |
| - surname | “sn” |
| … | |
| - telephone number | “telephone” |

Example:
/c=de/o=uni-karlsruhe/ou=rz/sn=zoller/telephone=+49 721 608 405

# Directory Service: X.500

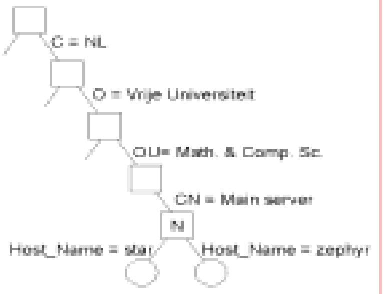
- A *Directory Service* supports lookup based on a set of attribute values (*yellow pages*)
- Directory entries contain <attrib, value> pairs
- Set of entries forms Directory Information Base (DIB).
- *Naming attributes* of an entry jointly identify an entry uniquely.
- Canonical sequences of naming attributes form the Directory Information Tree (DIT)
    - Edges are labeled with <attrib, value> pairs
- Each name attribute is a so called RDN (relative distinguished name)

# The X.500 Directory Entries

| Attribute | Abbr. | Value |
|---|---|---|
| Country | C | NL |
| Locality | L | Amsterdam |
| Organization | O | 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.

# The X.500 DIB



- Part of the directory information tree

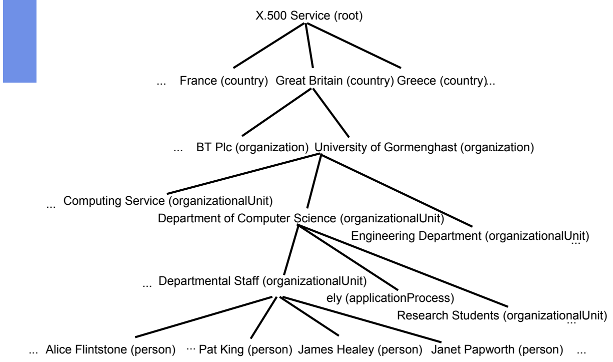
# The X.500 Name Space

| 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** |

- Two directory entries having *Host_Name* as RDN

# X.500 Directory Information Tree



X.500 Service (root)

... France (country) Great Britain (country) Greece (country)..

... BT Plc (organization) University of Gormenghast (organization)

... Computing Service (organizationalUnit)

Department of Computer Science (organizationalUnit)

Engineering Department (organizationalUnit) ...

... Departmental Staff (organizationalUnit)

ely (applicationProcess)

Research Students (organizationalUnit) ...

... Alice Flintstone (person) ... Pat King (person) James Healey (person) Janet Papworth (person) ...

# X.500 Lookup

**Name lookup**

```
list(/C=NL/O=Vrije Universiteit/
   OU=Math.&Comp.Sci./CN=Main server)
```

returns corresponding names

```
star zephyr
```

**Directory lookup**

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

returns all entries with matching attributes

# Locating Mobile Entities

- Up to now we always had a mapping from:
  User friendly names → node addresses

- Mobile systems often change either their names or their addresses in a DS, we need a better solution for looking up a specific node:
  - Name → identifier
  - Identifier → address

# Migration of Service

Example:

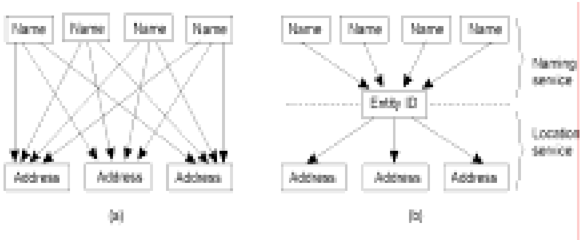
Assume we move ftp.cs.vu.nl to a new machine in a very far away domain, suppose: ftp.cs.unisa.edu.au

However, we would like to use ftp.cs.vu.nl , because many applications might contains program fragments using this name a as symbolic link, i.e. this name will be used as an ID, thus hanging this ID, all links to it will be invalid.

# Two Principle Solutions

1. Insert the address of the new machin in the DND data base for the entry cs.vu.nl
   - No implications for look ups
   - However, if this ftp-service will migrate again to some other place, also its entry in the DNS data base has to be updated
2. Insert the name of the new machione instead of its address (thus ftp.cs.vu.nl becomes a symbolic name link).
   - Loop ups are less efficient due to its 2 steps
     1. Look for the name of the new machine
     2. Look for the address related to this new name

# Naming versus Locating Entities

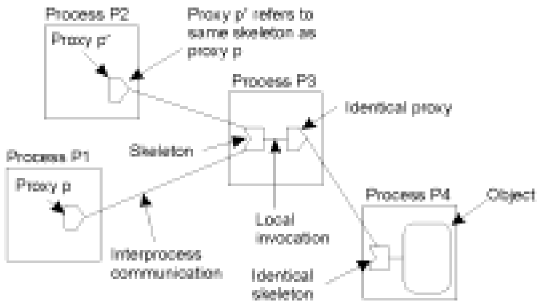


a) Single level mapping between names and addresses
b) T-level mapping using identities.

# Location Services

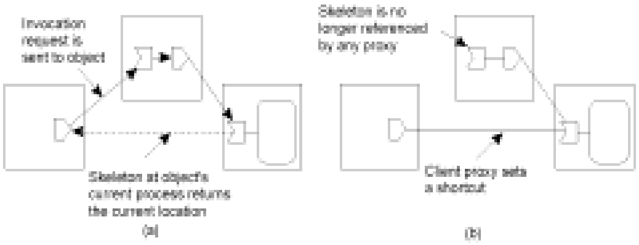
- Simple approaches (in general, LAN only)
- Broadcast id, corresponding node responds with its current address (ex.: ARP)
- Multicast id to a well-known group, all mobile nodes subscribe to that group

# Forwarding Pointers (1)



- Principle of forwarding pointers using (*proxy, skeleton*) pairs

# Forwarding Pointers (2)

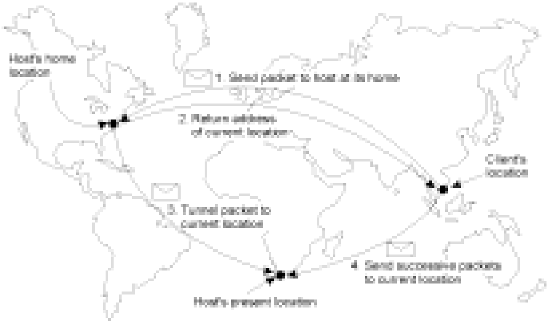

Invocation request is sent to object
Skeleton at object's current process returns the current location
(a)
Skeleton is no longer referenced by any proxy
Client proxy sets a shortcut
(b)

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

# Location Home-Based Approach

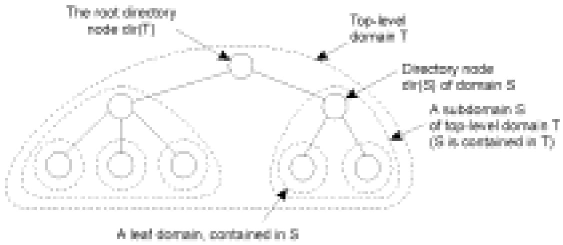
- Can be used as a fall-back mechanism with forwarding pointers
- A dedicated *home node* always keeps an up-to-date pointer to the mobile object
- Home node is typically node where an object originates
- Make home node *fault-tolerant* to ensure object can always be located

# 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

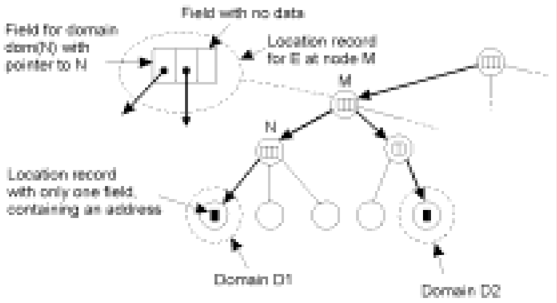
- The principle of Mobile IP

# 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

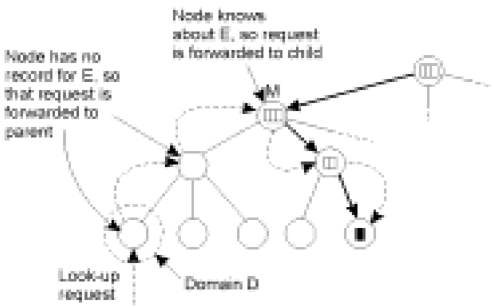
- Hierarchical organization of a location service into domains, each having an associated directory node.

## Hierarchical Approaches (2)


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

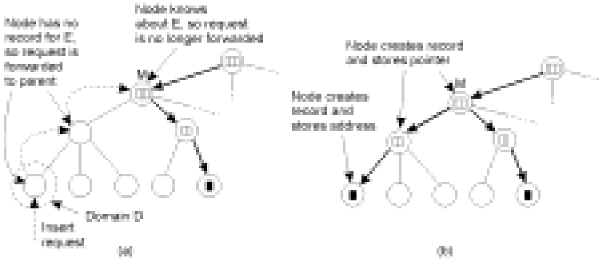
- An example of storing information of an entity having two addresses in different leaf domains.

## Hierarchical Approaches (3)


Node knows about E, so request is forwarded to child
Node has no record for E, so that request is forwarded to parent
M
Look-up request
Domain D

- Looking up a location in a hierarchically organized location service

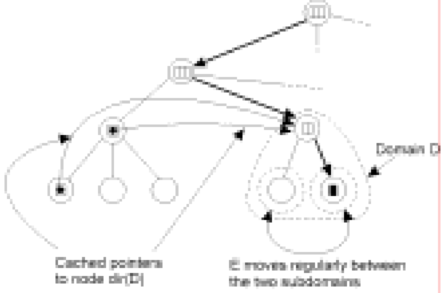
## Hierarchical Approaches (4)


Node has no record for E, so request is forwarded to parent
Node knows about E, so request is no longer forwarded
M
Domain D
Insert request
(a)
Node creates record and stores pointer
Node creates record and stores address
M
(b)

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

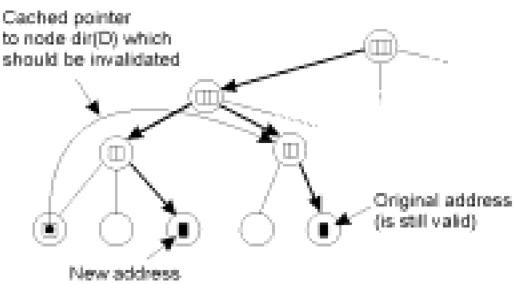
b) A chain of forwarding pointers to the leaf node is created

## Pointer Caches (1)


Domain D
Cached pointers to node dir(D)
E moves regularly between the two subdomains

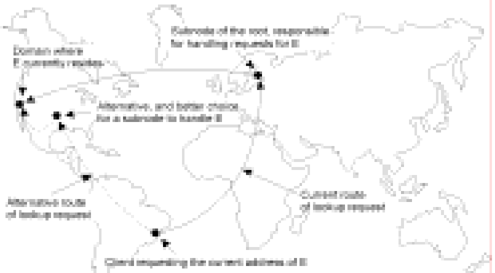
- Caching a reference to a directory node of the lowest-level domain in which an entity will reside most of the time

## Pointer Caches (2)


Cached pointer to node dir(D) which should be invalidated
Original address (is still valid)
New address

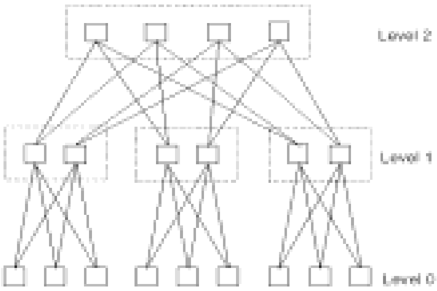
- Cache entry that needs to be invalidated because it returns a nonlocal address, while such an address is available.

## Scalability Issues


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

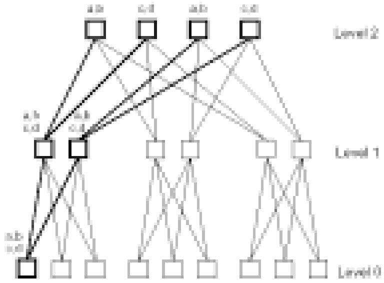
Scalability issues related to uniformly placing subnodes of a partitioned root node across the network covered by a location service.

## Scalability Solution



Use of a *fat tree* to increase the number of servers at upper levels of the tree

## Scalability Solution



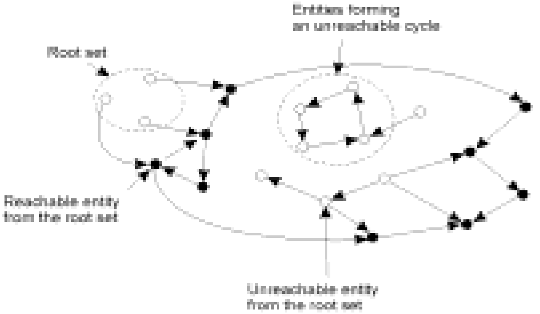
- Replicating and partitioning binding information among the multiple parents in the fat tree

## Deleting of Objects

- *How to manage, that an object can be deleted without any dangling reference?*
- However, we want to able to delete objects whether there might be any valid refernces to that object
- *What to do with objects without any current reference ?* (it's wasting resource, may be never reused again)

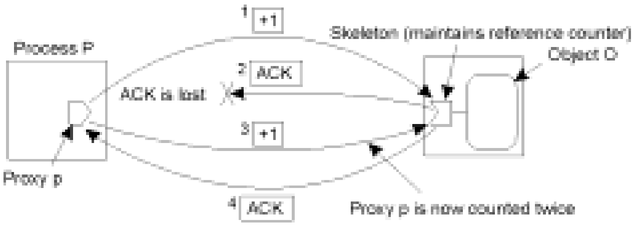
⇒ we need a distributed garbage collection

## Problem of Unreferenced Objects



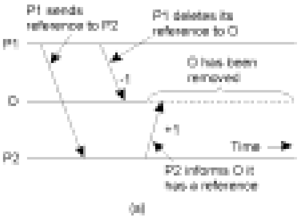
- An example of a graph representing objects containing references to each other.

## 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

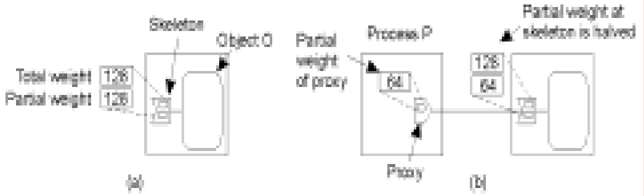
## Reference Counting (2)



a) Copying a reference to another process and incrementing the counter too late

b) A solution.

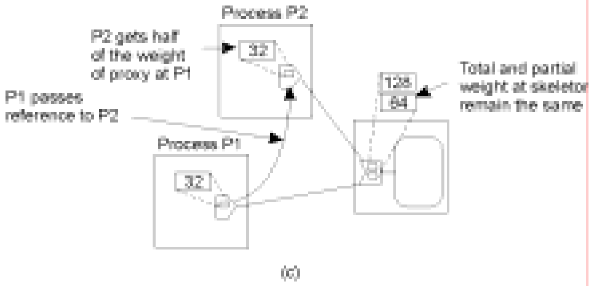
## Advanced Referencing Counting (1)



a) The initial assignment of weights in weighted reference counting

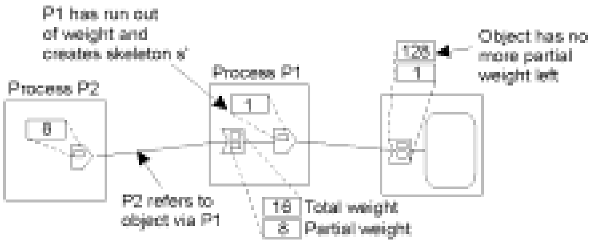
b) Weight assignment when creating a new reference.

## Advanced Referencing Counting (2)


Process P2
P2 gets half of the weight of proxy at P1
32
P1 passes reference to P2
Process P1
32
128
64
Total and partial weight at skeleton remain the same
(c)

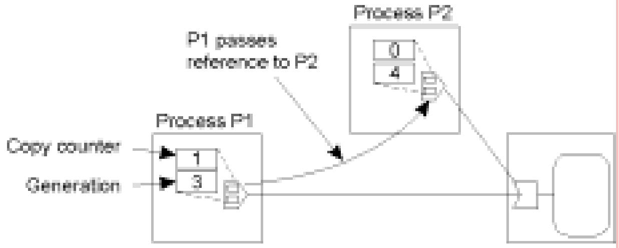
c) Weight assignment when copying a reference.

## Advanced Referencing Counting (3)


P1 has run out of weight and creates skeleton s'
Process P1
Process P2
8
1
128
1
Object has no more partial weight left
P2 refers to object via P1
16 Total weight
8 Partial weight

- Creating an indirection when the partial weight of a reference has reached 1.

## Advanced Referencing Counting (4)



- Creating and copying a remote reference in generation reference counting.

## Generation Referencing Counting

Skeleton maintains *G*

- *G[k]* is #proxies in generation *k*

When a proxy is removed

- it sends its generation *k* and copy count *n* to the skeleton
- skeleton does
  - *G[k] -= 1*
  - *G[k+1] += n*

Remaining problems:

- Node failures, need reliable communication

## Identifying Unreachable Objects

- *Mark-and-sweep tracing* (e.g., in Emerald)
- Local collectors mark objects reachable from a local root, or a marked object
- A marked proxy notifies its skeleton, thus marking it
- When all local collectors are done marking, collect unmarked objects

Problem:

- Requires execution to stop during marking
- Incremental approaches may cause excessive communication

## Reference Lists

- Adding and deleting proxies from a reference list in the skeleton of the object server are idem potent operations ⇒

  do not require *reliable communication*
- However, we will use acks to be sure that adding or deleting have been done
- Skeleton is able to control the consistency of its reference list pinging to all proxies
- Scalability is low

## Hierarchical Tracing

- Hierarchical approach to GC
- Nodes are recursively partitioned into groups
- Each group does internal GC
- References among groups are GC'ed in next higher group

## Hierarchical Tracing

Skeleton marks:

- Soft: reachable only from proxies inside group
- Hard: reachable from proxy outside group, or reachable from a root inside the group

Proxy marks:

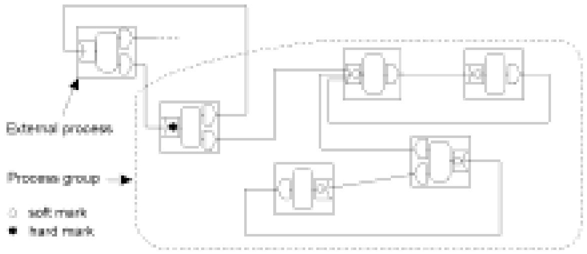
- None: unreachable
- Soft: reachable from a skeleton marked soft
- Hard: reachable from a root

## Hierarchical Tracing

GC within a group proceeds as follows:

1. Initial marking of skeletons (soft or hard)
2. Intraprocess propagation of marks from skeletons to proxies (local GC)
3. Interprocess, intragroup propagation of hard marks from proxies to skeletons
4. Stabilization (iterate steps 2 and 3)
5. Garbage reclamation (e.g., mark soft skeletons as garbage, run local GC)

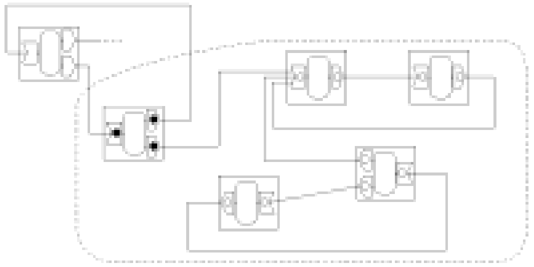
## Tracing in Groups (1)



(a)

- Initial marking of skeletons

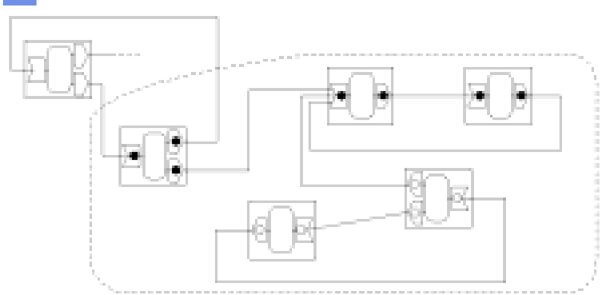
## Tracing in Groups (2)



(b)

- After local propagation in each process

# Tracing in Groups (3)



(c)

- Final marking