Chapter 5

Naming

October, 22nd

Introduction

Motivation

- What if we had to memorize IP addresses to reach websites?
  - Do you know Google's IP address?
  - From USI: `172.217.18.110`
- IPv4: 32 bit address. IPv6: 128 bit address
  - Do you know Google's IPv6 address?
  - From USI: `2a00:1450:400a:804:200e`

Types of naming

- Flat naming
- Structured naming
- Attribute-based naming

Names, identifiers, and addresses

- **Name** refers to the *entity*
  - typically human-friendly
  - not necessarily unique or identifying (e.g. person's names)
- **Address** is a name for an *access point* of an entity
  - low-level
  - not human-friendly
- **Identifier** is a name *uniquely* identifying an entity
  - refers to at most one entity
  - each entity is referred to by at most one identifier
  - an identifier always refers to the same entity

Flat Naming

Broadcasting

Broadcast the ID, requesting the entity to return its current address

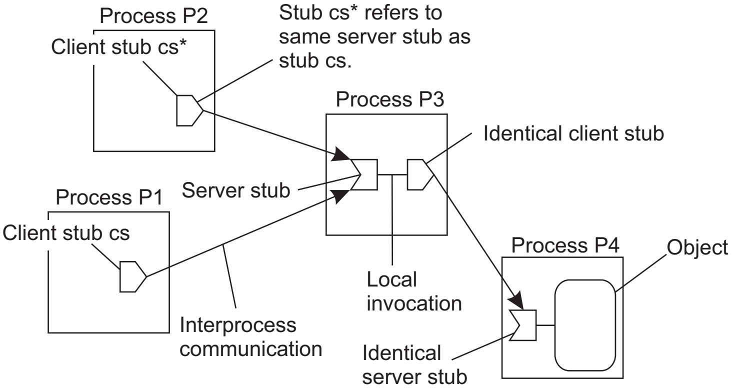
- can never scale beyond local area networks
- requires all processes to listen to incoming location requests

Example: Address Resolution Protocol (ARP)

- translates IP addresses to MAC addresses
- broadcast ”who has this IP address?”

Forwarding pointers

When an entity moves from A to B, it leaves a reference in A pointing to B. Requests are forwarded transparently.



The principle of forwarding pointers using (client stub, server stub) pairs.

Forwarding pointers

Drawbacks:

- ✗ A fast moving entity leaves a **long chain** of references
- ✗ Each intermediate location needs to allocate **resources** for the chain information and request forwarding
- ✗ Any missing step breaks the chain and makes the **entity unreachable**

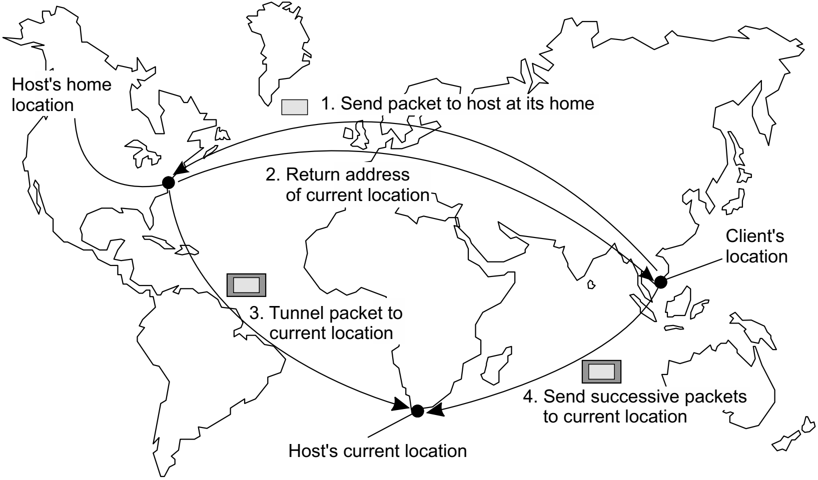
Home-based approaches

A home location keeps track of the current location

1. In IPv6, a mobile device requests a local, temporary address
2. That address is registered with the home agent
3. Home agent forwards packets to the mobile device
4. Home agent tells sender the new temporary location

IPv6 address becomes an identifier

Home-based approaches



The principle of Mobile IP.

Home-based approaches

Drawbacks:

- ✗ Increased latency through indirection (if client is close to entity)
- ✗ Home agent must always be available
- ✗ How to handle permanent relocations?

Solution:

- ✓ Home locations are registered with DNS

Distributed hash tables

Example: Chord

- Nodes are assigned an m-bit identifier space
- Main issue: efficiently resolve successor of k, $\mathrm{succ}(k)$
    - Linear key lookup does not scale well
- Solution: a finger table such that $\mathrm{FT}_p[i] = \mathrm{succ}(p + 2^{i-1})$
    - I.e., ith entry points to the first node succeeding p by at least 2^{i-1}
    - Look up of key k at node p causes a forward of the request to node q with index j in p's finger table, where

$$q = \mathrm{FT}_p[j] \leq k < \mathrm{FT}_p[j+1]$$

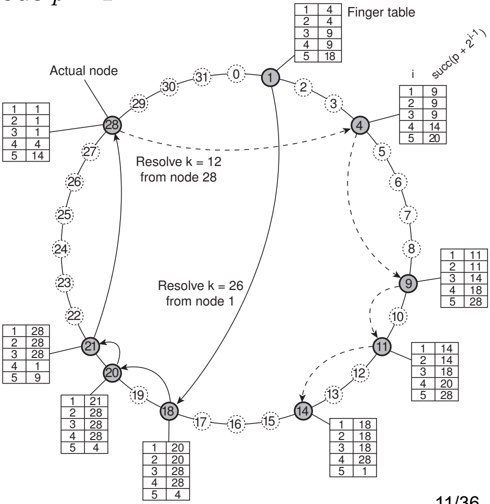
    - Complexity: $O(\log(N))$, where N is the number of nodes

Distributed hash tables

Example: Chord

Example: “Resolve $k = 26$ from node $p = 1$”

1. Node 1
   - $26 \geq \mathrm{FT}_1[2]$
   - forward to $\mathrm{FT}_{18}[2] = 18$
2. Node 18
   - $26 \geq \mathrm{FT}_{18}[2]$
   - forward to $\mathrm{FT}_1[5] = 18$
3. $\ldots$
4. Node 21
   - $26 < \mathrm{FT}_{21}[1]$
   - return “Node 28”



Distributed hash tables

Drawbacks:

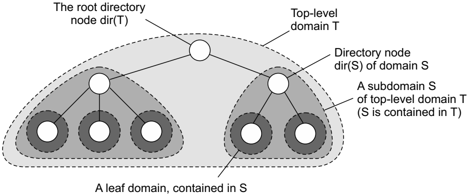
- ✗ Nodes joining and leaving (voluntarily or crashing) forces a finger table update
- ✗ Network proximity

Solution:

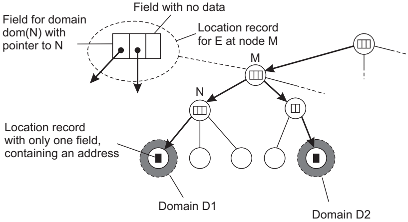
- ✓ Use $\mathrm{succ}(k)$ to update their finger tables in the background
- ✓ Proximity routing / proximity neighbor selection (not in Chord)

Hierarchical approaches

- Divide network nodes into *hierarchy of domains*
- Each domain has a *directory node*
- Each directory node records only next-level directory node locations and only for names in domain
- Leaf domains contain the location of entities



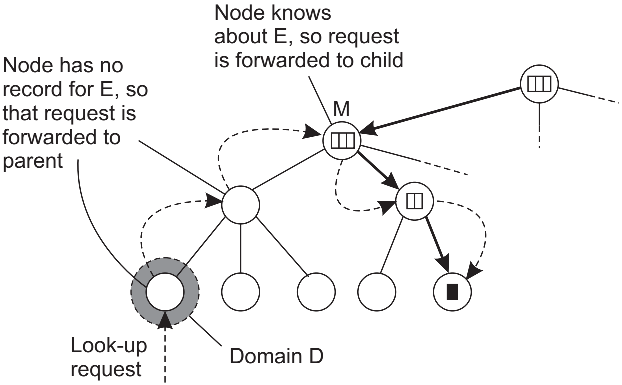
(a) Hierarchical organization of a location service into domains.



(b) An example of storing information of an entity having two addresses in different leaf domains.

Hierarchical approaches

- Lookup requests are first forwarded amongst children of the directory node, and then up the hierarchy $\Rightarrow$ *leverage locality*.



Looking up a location in a hierarchically organized location service.

Structured Naming

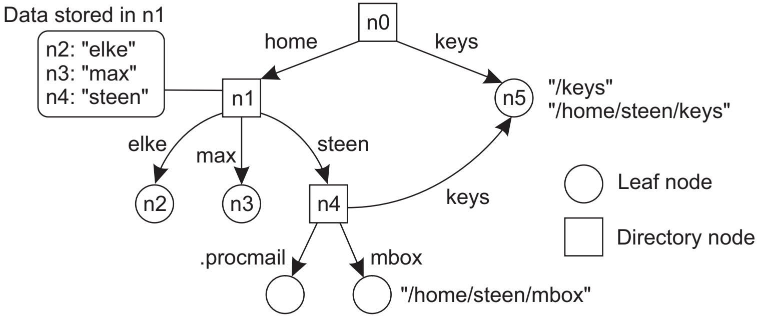
Name spaces

- Name spaces can be represented as a labeled, directed graph with directory and leaf nodes
  - The root node is a directory node with only outgoing edges
- A path name is the sequence of labels corresponding to the edges in that path
  - Starting at the root: absolute path name
  - Otherwise: relative path name
- A global name can be used anywhere in the system, a local name can only be interpreted locally

Name spaces

File systems are name spaces

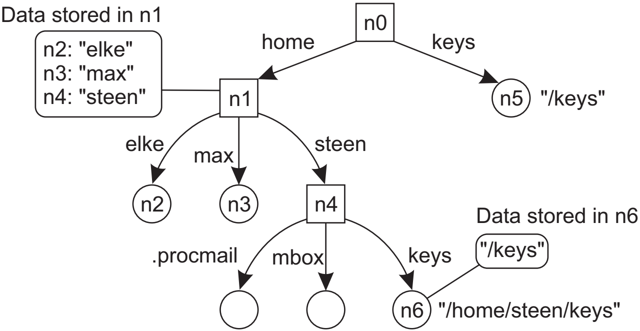
- Usual path name notation: names separated by "/"
- Example: $\langle$**home**, **steen**, **mbox**$\rangle$ = **/home/steen/mbox**



A general naming graph with a single root node.

Linking and mounting

- In UNIX file systems, edges are called *hard links*
- If a node contains an absolute path instead of a file, it is called a *symbolic link*



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

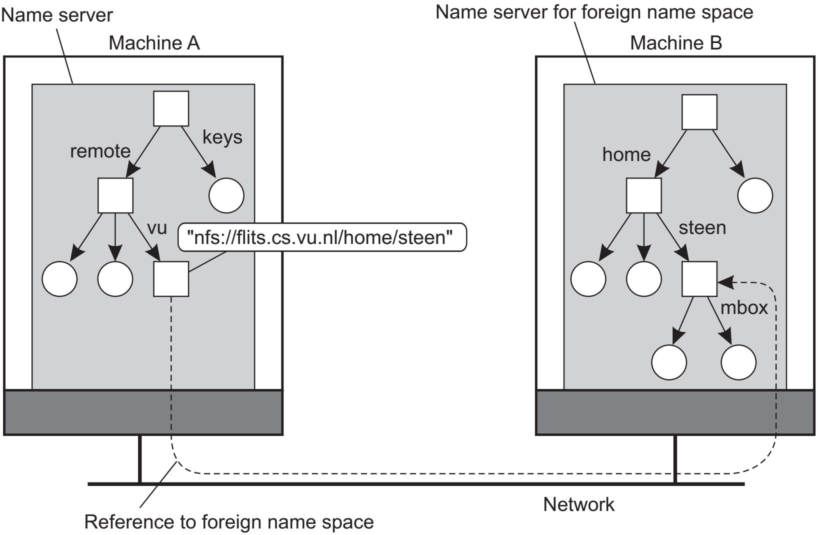
Linking and mounting

- Name resolution can also be used to merge different address spaces transparently through *mounting*
    - Associating node identifier of another name space with node in current name space
    - Type of symbolic link
- Terminology
    - **Foreign name space:** name space to be accessed/integrated
    - **Mount point:** node in current name space containing the node identifier of the foreign name space
    - **Mounting point:** node in foreign name space where to continue name resolution

Linking and mounting

- Mounting a foreign name space in a distributed system requires at least:
    - The name of an access protocol
    - The name of the server
    - The name of the mounting point in the foreign name space
- Example: NFS
  NFS URLs: `nfs://server.usi.ch/home/ricardo`
    - `nfs://` — access protocol
    - `server.usi.ch` — server name
    - `/home/ricardo` — name of the mounting point (`ricardo` is mounted)

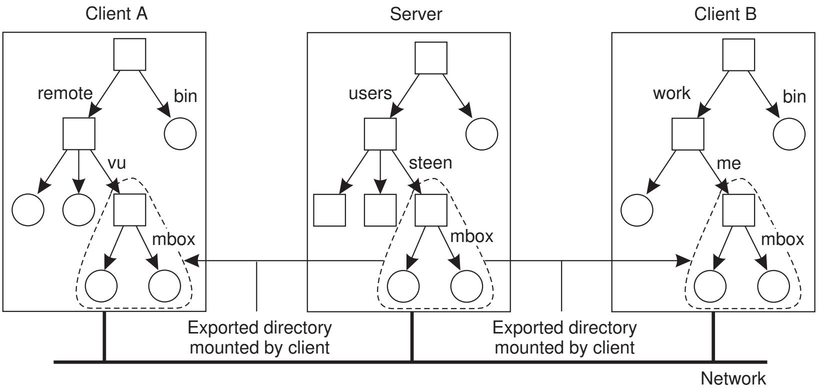
Linking and mounting



Mounting remote name spaces through a specific protocol.

Linking and mounting

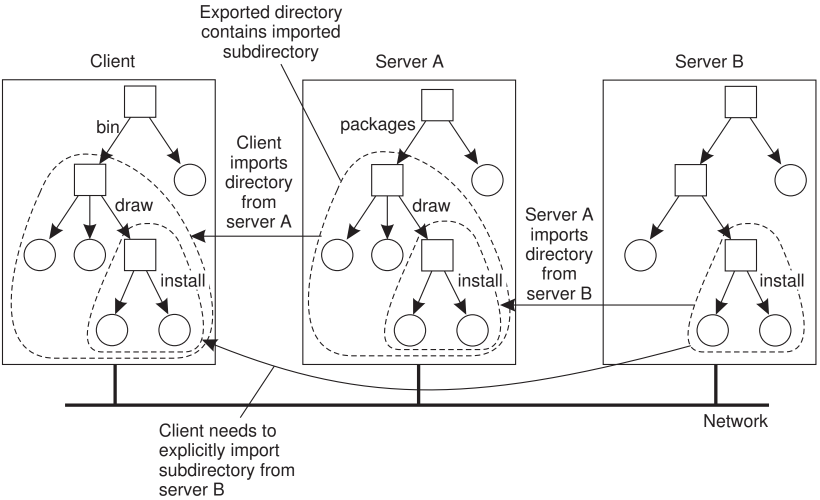
Different clients can have different namespaces.



Mounting (part of) a remote file system in NFS.

Linking and mounting

Nested mounts are possible.

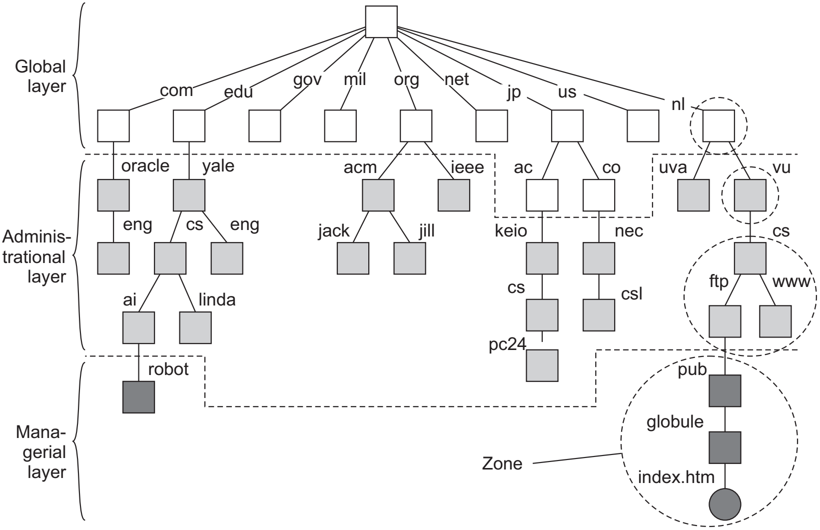


Mounting nested directories from multiple servers in NFS.

Name space distribution

- Large-scale name services require distribution of workload
  - Large number of clients
  - Large area of coverage
- Large-scale usually means hierarchical organization
- Organizational layers:
  - **Global layer:** root node and its children (very stable)
  - **Administrational layer:** nodes managed by a single organization (stable)
  - **Managerial layer:** managed locally (may change frequently)

Name space distribution



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

Name space distribution

Issue	Global	Administrational	Managerial
Geographical scale	Worldwide	Organization	Department
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
Client-side caching	Yes	Yes	Sometimes

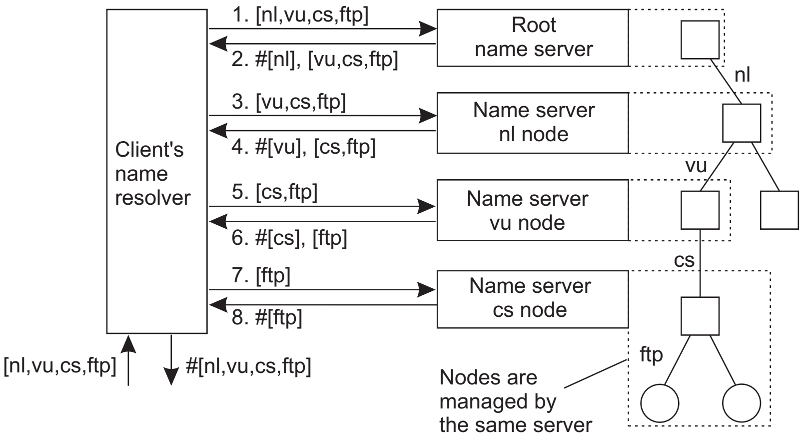
A comparison between name servers for implementing nodes from a large-scale name space partitioned into three layers.

Implementation of name resolution

- Clients have access to a *name resolver*
- Name resolution can be implemented:
  - **Iteratively:** resolver performs each step of the resolution
  - **Recursively:** resolver delegates the resolution to the root server

Implementation of name resolution

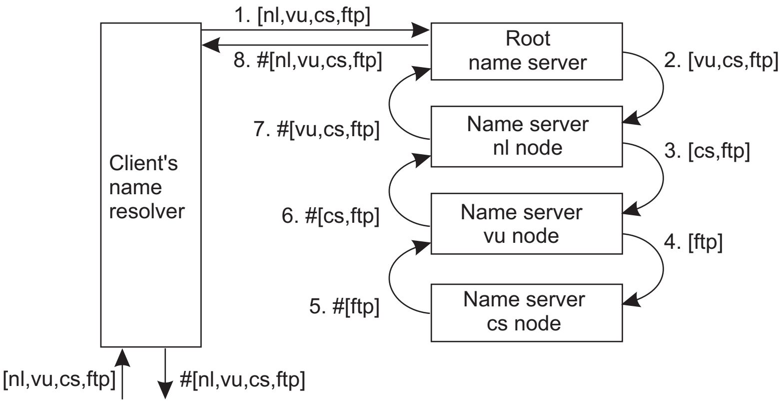
Iterative name resolution



The principle of iterative name resolution.

Implementation of name resolution

Recursive name resolution

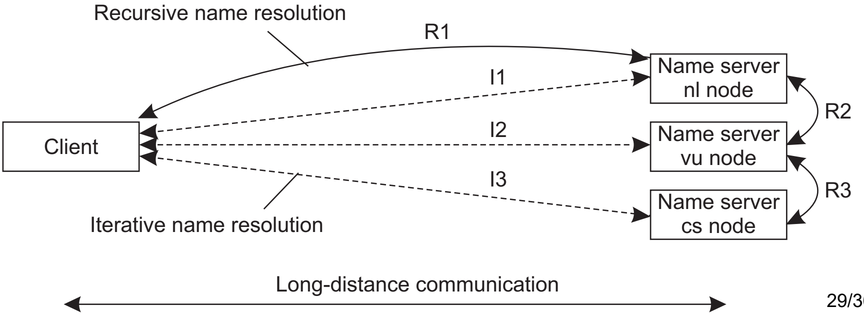


The principle of recursive name resolution.

Implementation of name resolution

Drawbacks:

- Iterative:
  - Communication costs (if the servers are not near the client)
  - Poor caching capabilities
- Recursive:
  - Increased load on each server on the chain
  - Increased latency for non-cached requests



Implementation of name resolution

Despite drawbacks, recursive is usually better because of very effective caching

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]

Recursive name resolution of `[nl, vu, cs, ftp]`. Name servers cache intermediate results for subsequent lookups.

The DNS name space

- DNS is a hierarchically organized name space
  - DNS root has no name
  - Each subtree is called a domain
  - Path to the subtree root node is the domain name
  - Each node contains records
  - A server is responsible for a zone
  - Domain without subdomains
- To decentralize DNS, we can map it into a DHT

DNS implementation

Type	Refers to	Description
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 u

Name	Record type	Record value
cs.vu.nl.	SOA	star.cs.vu.nl. hostmaster.cs.vu.nl. 2005092900 7200 3600 2419200 3600
cs.vu.nl.	TXT	"VU University - Computer Science"
cs.vu.nl.	MX	1 mail.few.vu.nl.
cs.vu.nl.	NS	ns.vu.nl.
cs.vu.nl.	NS	top.cs.vu.nl.
cs.vu.nl.	NS	solo.cs.vu.nl.
cs.vu.nl.	NS	star.cs.vu.nl.
star.cs.vu.nl.	A	130.37.24.6
star.cs.vu.nl.	A	192.31.231.42
star.cs.vu.nl.	MX	1 star.cs.vu.nl.
star.cs.vu.nl.	MX	666 zephyr.cs.vu.nl.
star.cs.vu.nl.	HINFO	"Sun" "Unix"
zephyr.cs.vu.nl.	A	130.37.20.10
zephyr.cs.vu.nl.	MX	1 zephyr.cs.vu.nl.
zephyr.cs.vu.nl.	MX	2 tornado.cs.vu.nl.
zephyr.cs.vu.nl.	HINFO	"Sun" "Unix"

Attribute-based Naming

Directory services

- Attribute-based naming systems are also known as directory services
- Describe an entity in terms of (attribute, value) pairs
- Entities have a set of attributes used for searching
  - Which attribute set is ideal?
  - How to describe the attributes?
- Resource Description Framework (RDF)
  - `(Person, name, Alice)` means “resource `Person` whose `name` is `Alice`”
  - In essence, RDF references work as URLs

Hierarchical implementations: LDAP

Lightweight Directory Access Protocol (LDAP)

- Each service contains a number of directory entries
- Each entry is composed of a number of (attribute, value) pairs
- Each attribute has an associated type
- Attributes can be single- or multi-valued
- The collection of all entries of a service is called a directory information base (DIB)

In a DIB, each entry has a relative distinguished name (RDN), which is globally unique

Hierarchical implementations: LDAP

Example of RDN

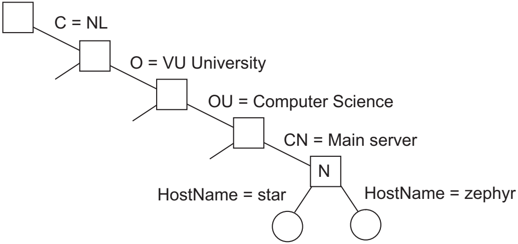
`/C=NL/O=Vrije Universiteit/OU=Comp. Sc.`

Attribute	Abbr.	Value
Country	C	NL
Locality	L	Amsterdam
Organization	O	VU University
OrganizationalUnit	OU	Computer Science
CommonName	CN	Main server
Mail_Servers	–	137.37.20.3, 130.37.24.6, 137.37.20.10
FTP_Server	–	130.37.20.20
WWW_Server	–	130.37.20.20

A simple example of an LDAP directory entry using LDAP naming conventions.

Hierarchical implementations: LDAP

- RDNs can be used as globally unique names, analogous to DNS
- RDNs establish a hierarchy of entries, called directory information tree (DIT)



Attribute	Value	Attribute	Value
Locality	Amsterdam	Locality	Amsterdam
Organization	VUUniversity	Organization	VUUniversity
OrganizationalUnit	ComputerScience	OrganizationalUnit	ComputerScience
CommonName	Mainserver	CommonName	Mainserver
HostName	star	HostName	zephyr
HostAddress	192.31.231.42	HostAddress	137.37.20.10