UNIT 2 NAMING

NAMES:

- In a distributed system, a name is used to refer to an entity (e.g., computer, service, remote object, file, user), Share resources, Uniquely identify entities, To refer locations

- An address is a name that refers to an access point of an entity – E.g. a server's address consists of an IP address and a port number

- An entity may have multiple access points and addresses – A person has several phone numbers (e.g. work, home, mobile)

- An entity may change its access points – E.g., a service is moved to a different host, a person changes its email address after changing his job

- Identifiers are used to uniquely identify an entity. An identifier is a name with the following properties:
  – An identifier refers to at most one entity
  – Each entity is referred to by at most one identifier
  – An identifier always refers to the same entity (i.e., it is never reused).

Another important type of name is that which is tailored to be used by humans, also referred to as human-friendly names. In contrast to addresses and identifiers, a human-friendly name is generally represented as a character string. These names appear in many different forms. For example, files in UNIX systems have character-string names that can be as long as 255 characters, and which are defined entirely by the user. Similarly, DNS names are represented as relatively simple case-insensitive character strings.

Addresses and identifiers are two important types of names that are each used for very different purposes. In many computer systems, addresses and identifiers are represented in machine-readable form only, that is, in the form of bit strings. For example, an Ethernet address is essentially a random string of 48 bits. Likewise, memory addresses are typically represented as 32-bit or 64-bit strings.

A name is decomposed into several parts such as ftp.cs.vu.nl and that name resolution takes place through a recursive look-up of those parts. For example, a client needing to know the address of the FTP server named by ftp.cs.vu.nl would first resolve nl to find the server NS(nl) responsible for names that end with nl, after which the rest of the name is passed to server NS(nl). This server may then resolve the name vu to the server NS(vu.nl) responsible for names that end with vu.nl who can further handle the remaining name ftp.cs. Eventually, this leads to routing the name resolution request as:

NS(.) NS(nl) NS(vu.nl) address of ftp.cs.vu.nl

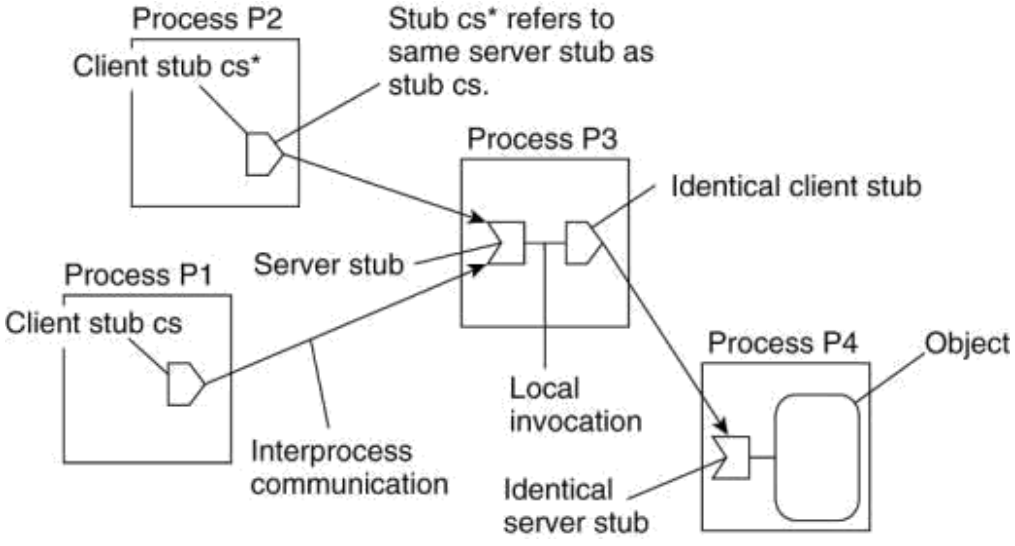
where NS(.) denotes the server that can return the address of NS(nl), also known as the root server. NS(vu.nl) will return the actual address of the FTP server. It is interesting to note that the boundaries between name resolution and message routing are starting to blur.

FLAT NAMING

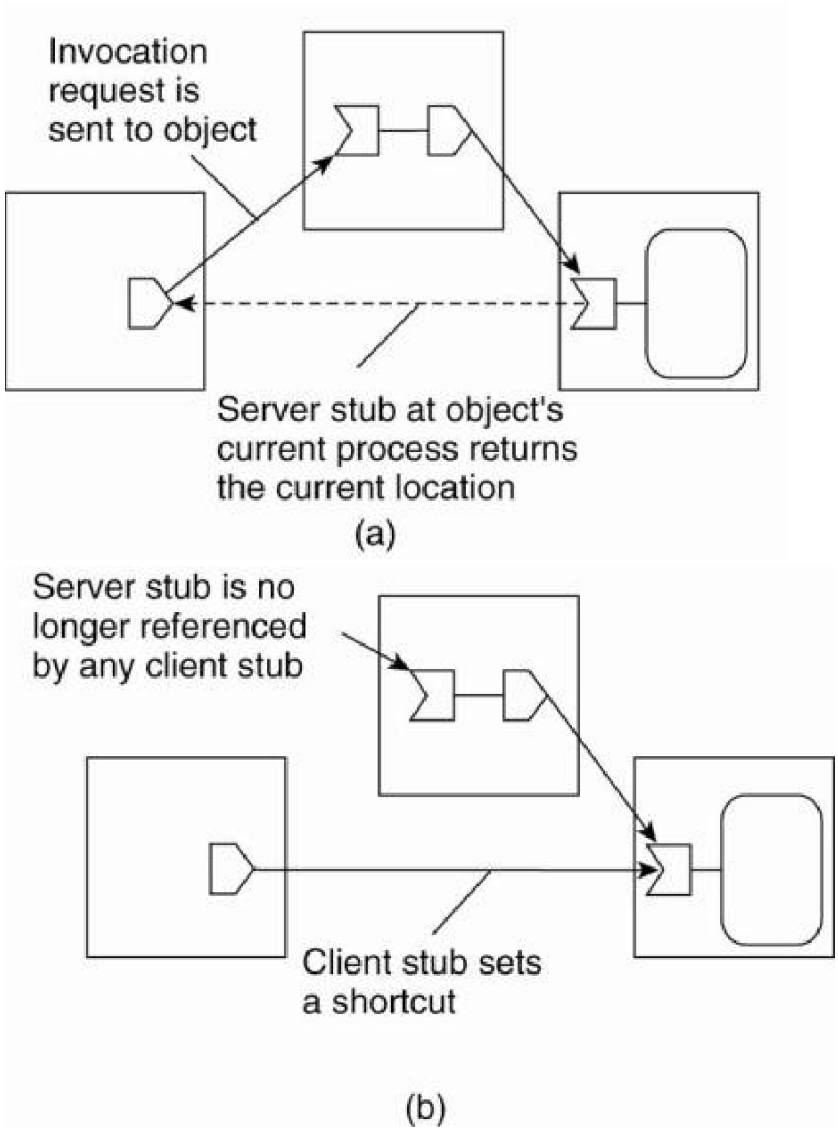
A naming system maintains a name-to- address binding to resolve names to addresses – In a distributed system, the implementation of a naming system is often distributed across multiple machines. **. In the following, we will take a look at how flat names can be resolved, or, equivalently, how we can locate an entity**
when given only its identifier.
We first consider two simple solutions for locating an entity. Both solutions are applicable only to local-area networks. Nevertheless, in that environment, they often do the job well, making their simplicity particularly attractive. Simple solutions: Broadcasting and multicasting
A location service accepts an identifier as input and returns the current address of the identified entity.
Simple solutions exist to work in local area network. Address Resolution Protocol (ARP) to map the IP address of a machine to its data-link address, which uses broadcasting.
Multicasting can be used to locate entities in point-to- point networks (such as the Internet).
Each multicasting address can be associated with multiple replicated entities.

Forwarding Pointers
Another popular approach to locating mobile entities is to make use of forwarding pointers (Fowler, 1985). The principle is simple: when an entity moves from A to B, it leaves behind in A a reference to its new location at B. The main advantage of this approach is its simplicity: as soon as an entity has been located, for example by using a traditional naming service, a client can look up the current address by following the chain of forwarding pointers.

There are also a number of important drawbacks. First, if no special measures are taken, a chain for a highly mobile entity can become so long that locating that entity is prohibitively expensive. Second, all intermediate locations in a chain will have to maintain their part of the chain of forwarding pointers as long as needed. A third (and related) drawback is the vulnerability to broken links. As soon as any forwarding pointer is lost (for whatever reason) the entity can no longer be reached.



Whenever an object moves from address space A to B, it leaves behind a client stub in its place in A and installs a server stub that refers to it in B. An interesting aspect of this approach is that migration is completely transparent to a client. The only thing the client sees of an object is a client stub. How, and to which location that client stub forwards its invocations, are hidden from the client. Also note that this use of forwarding pointers is not like looking up an address. Instead, a client's request is forwarded along the chain to the actual object

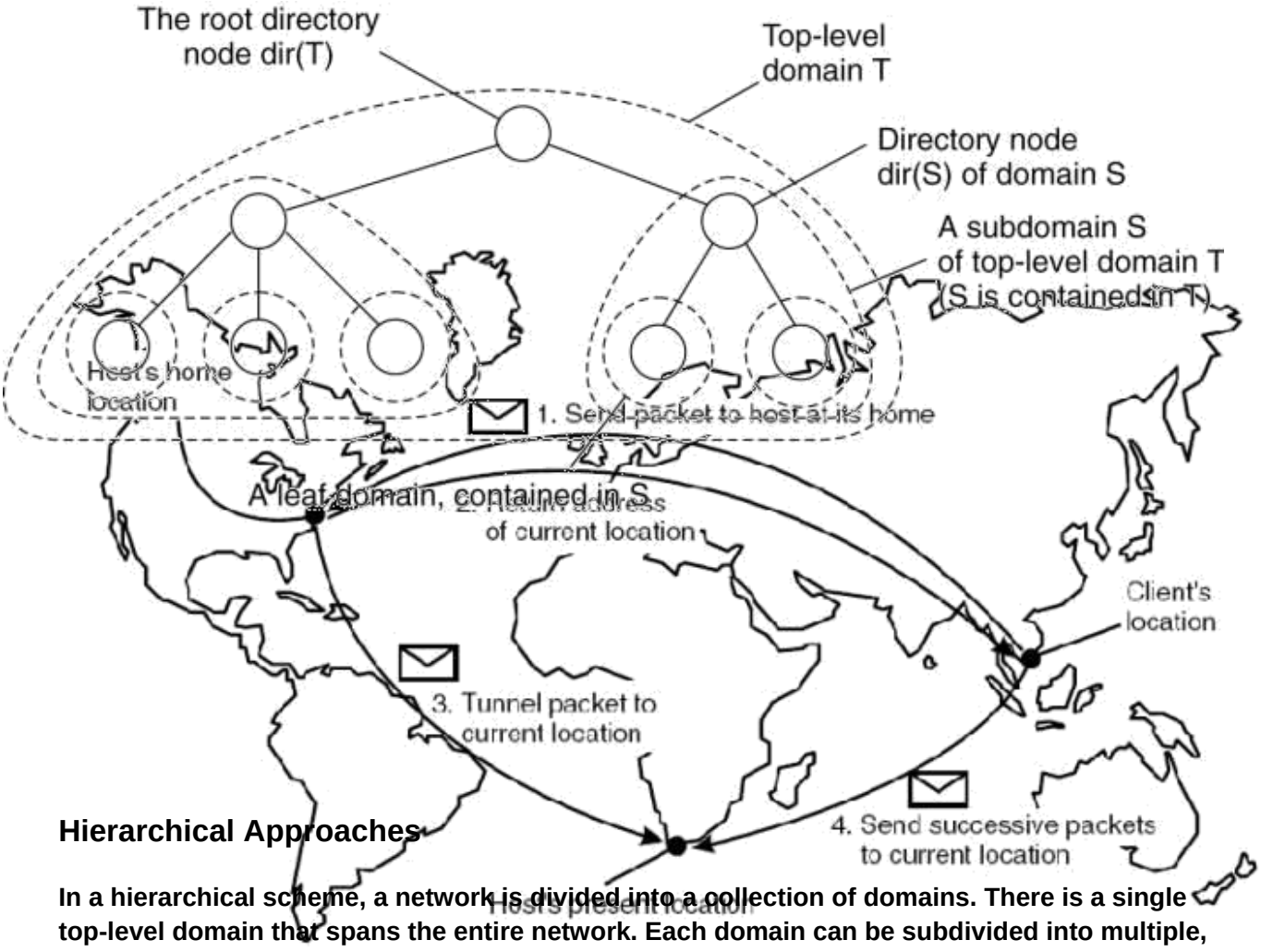


Redirecting a forwarding pointer, by storing a shortcut in a proxy

Home-Based Approaches

The use of broadcasting and forwarding pointers imposes scalability problems. Broadcasting or multicasting is difficult to implement efficiently in largescale networks whereas long chains of forwarding pointers introduce performance problems and are susceptible to broken links.

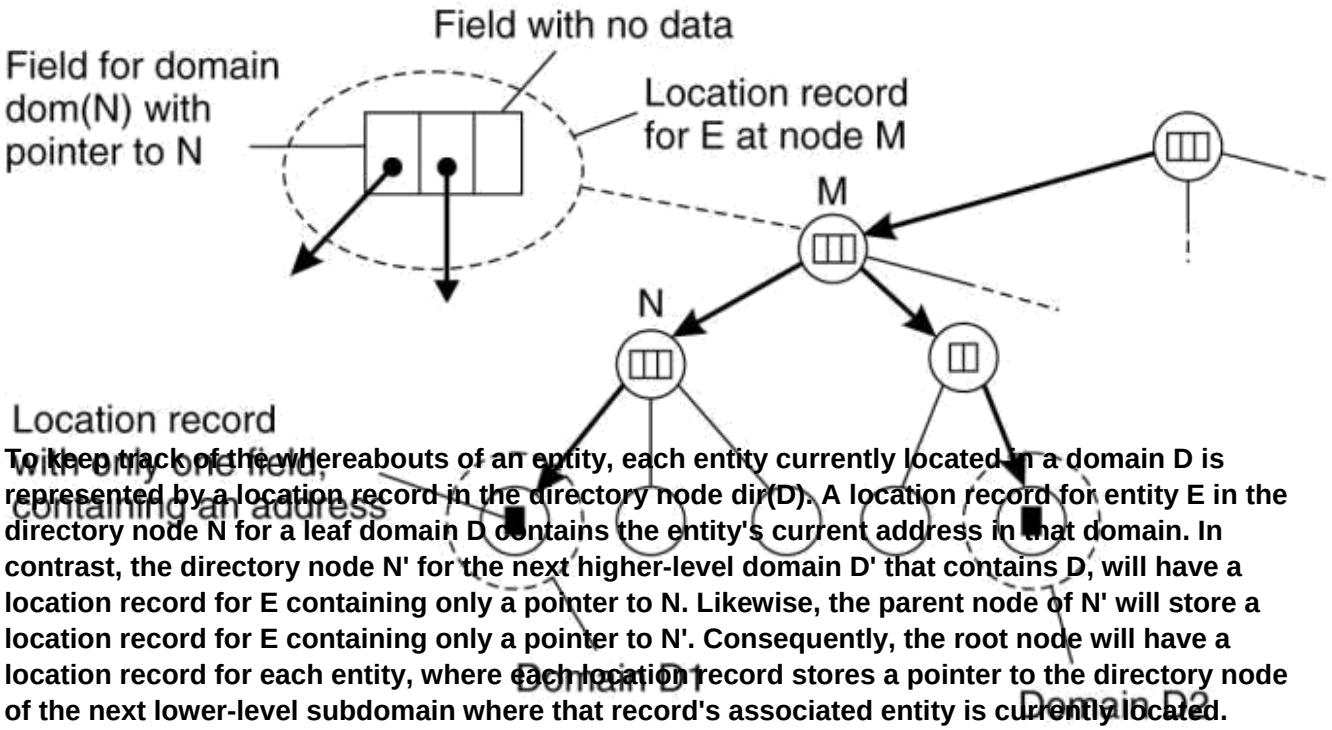
A popular approach to supporting mobile entities in large-scale networks is to introduce a home location, which keeps track of the current location of an entity. Special techniques may be applied to safeguard against network or process failures. In practice, the home location is often chosen to be the place where an entity was created.



Hierarchical Approaches

In a hierarchical scheme, a network is divided into a collection of domains. There is a single top-level domain that spans the entire network. Each domain can be subdivided into multiple, smaller subdomains. A lowest-level domain, called a leaf domain, typically corresponds to a local-area network in a computer network or a cell in a mobile telephone network.

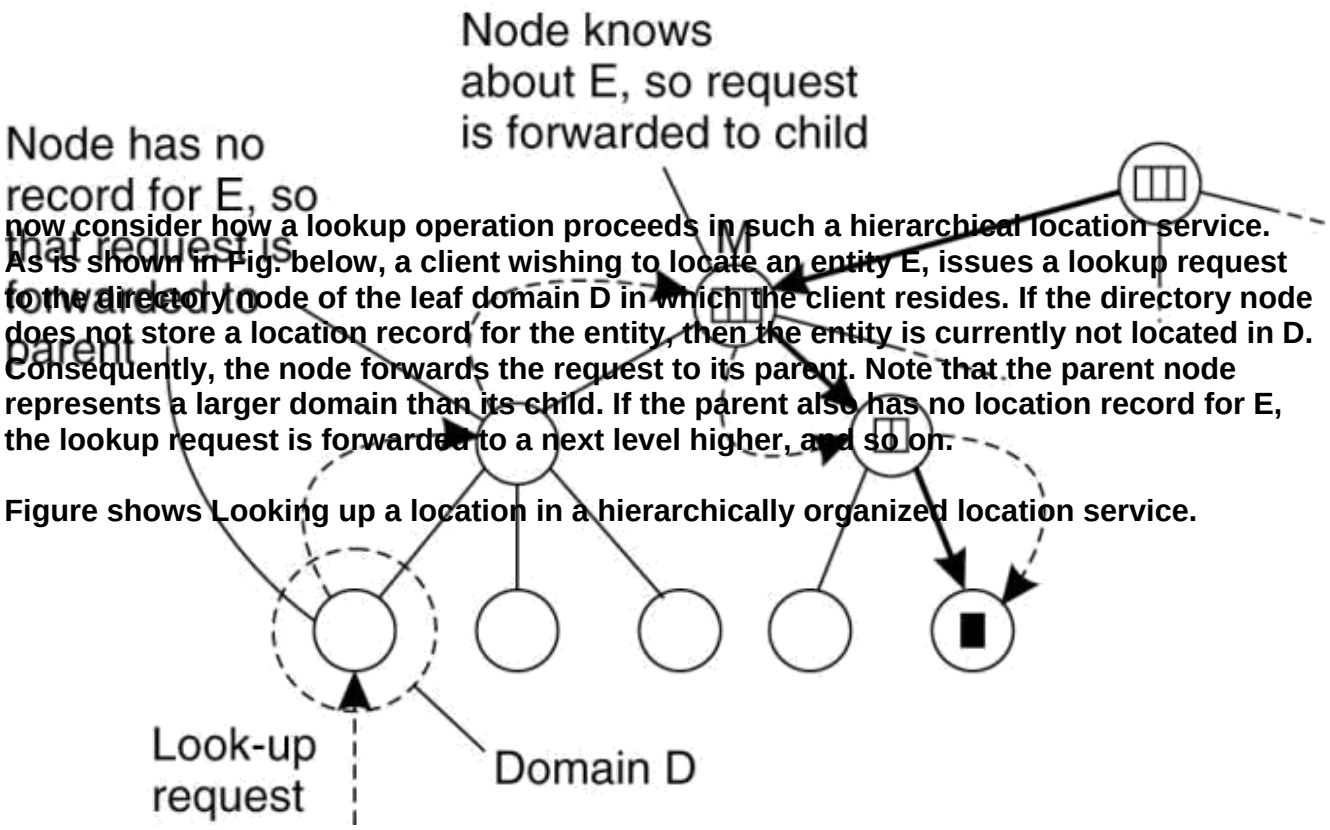
Each domain D has an associated directory node dir(D) that keeps track of the entities in that domain. This leads to a tree of directory nodes. The directory node of the top-level domain, called the root (directory) node, knows about all entities. This general organization of a network into domains and directory nodes is illustrated below
Figure Hierarchical organization of a location service into domains, each having an associated directory node.



To keep track of the whereabouts of an entity, each entity currently located in a domain D is represented by a location record in the directory node dir(D). A location record for entity E in the directory node N for a leaf domain D contains the entity's current address in that domain. In contrast, the directory node N' for the next higher-level domain D' that contains D, will have a location record for E containing only a pointer to N. Likewise, the parent node of N' will store a location record for E containing only a pointer to N'. Consequently, the root node will have a location record for each entity, where each location record stores a pointer to the directory node of the next lower-level subdomain where that record's associated entity is currently located.

An entity may have multiple addresses, for example if it is replicated. If an entity has an address in leaf domain D1 and D2 respectively, then the directory node of the smallest domain containing both D1 and D2, will have two pointers, one for each subdomain containing an address. This leads to the general organization of the tree as shown in Fig below

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

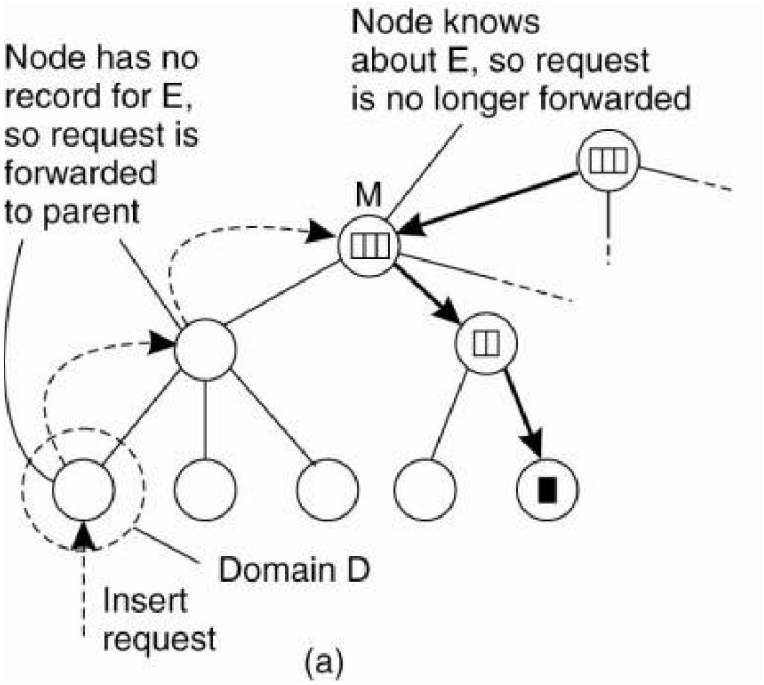


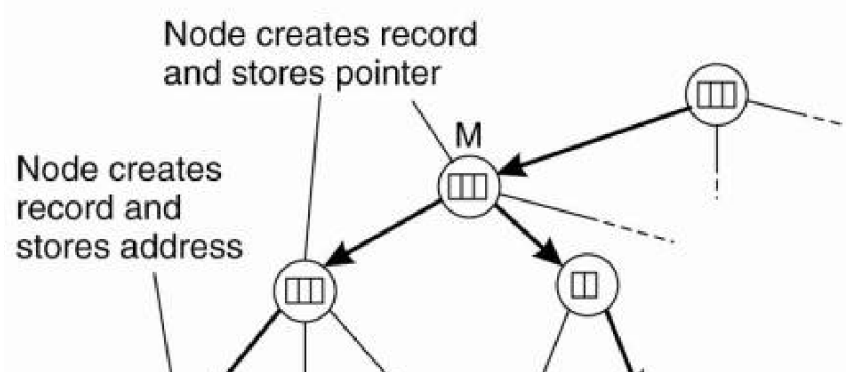
now consider how a lookup operation proceeds in such a hierarchical location service. As is shown in Fig. below, a client wishing to locate an entity E, issues a lookup request to the directory node of the leaf domain D in which the client resides. If the directory node does not store a location record for the entity, then the entity is currently not located in D. Consequently, the node forwards the request to its parent. Note that the parent node represents a larger domain than its child. If the parent also has no location record for E, the lookup request is forwarded to a next level higher, and so on.

Figure shows Looking up a location in a hierarchically organized location service.

As soon as the request reaches a directory node M that stores a location record for entity E, we know that E is somewhere in the domain dom(M) represented by node M. In Fig. above, M is shown to store a location record containing a pointer to one of its subdomains. The lookup request is then forwarded to the directory node of that subdomain, which in turn forwards it further down the tree, until the request finally reaches a leaf node. The location record stored in the leaf node will contain the address of E in that leaf domain. This address can then be returned to the client that initially requested the lookup to take place.

Figure 5-8. (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.





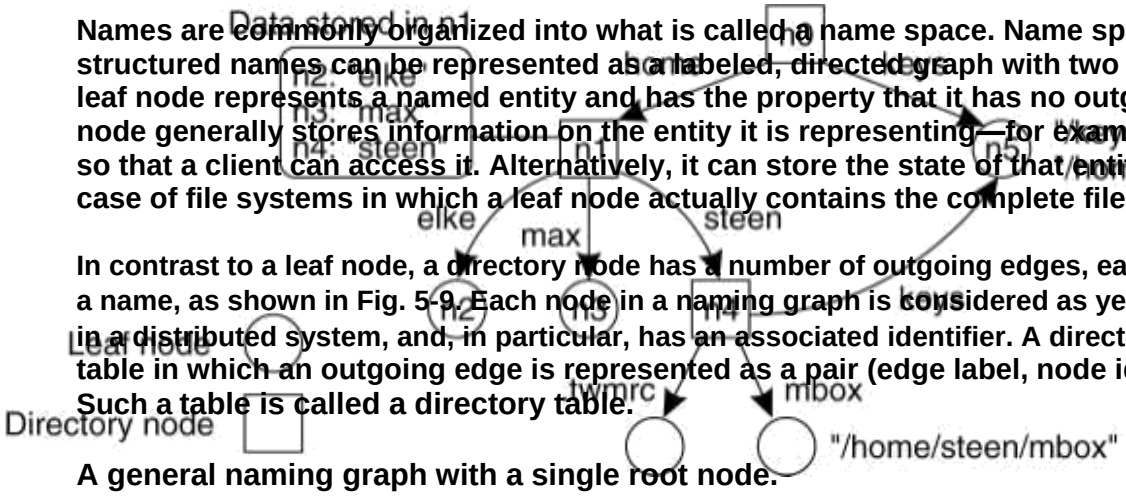
Node M will then store a pointer in the location record for E, referring to the child node from where the insert request was forwarded. At that point, the child node creates a location record for E, containing a pointer to the next lower-level node from where the request came. This process continues until we reach the leaf node from which the insert was initiated. The leaf node, finally, creates a record with the entity's address in the associated leaf domain.

5.3. Structured Naming

Flat names are good for machines, but are generally not very convenient for humans to use. As an alternative, naming systems generally support structured names that are composed from simple, human-readable names. Not only file naming, but also host naming on the Internet follow this approach

Name Spaces

Names are commonly organized into what is called a name space. Name spaces for structured names can be represented as a labeled, directed graph with two types of nodes. A leaf node represents a named entity and has the property that it has no outgoing edges. A leaf node generally stores information on the entity it is representing—for example, its address—so that a client can access it. Alternatively, it can store the state of that entity, such as in the case of file systems in which a leaf node actually contains the complete file it is representing.



In contrast to a leaf node, a directory node has a number of outgoing edges, each labeled with a name, as shown in Fig. 5-9. Each node in a naming graph is considered as yet another entity in a distributed system, and, in particular, has an associated identifier. A directory node stores a table in which an outgoing edge is represented as a pair (edge label, node identifier). Such a table is called a directory table.

A general naming graph with a single root node.

The naming graph shown in Fig. above has one node, namely n0, which has only outgoing and no incoming edges. Such a node is called the root (node) of the naming graph. Although it is possible for a naming graph to have several root nodes, for simplicity, many naming systems have only one. Each path in a naming graph can be referred to by the sequence of labels corresponding to the edges in that path, such as

N:<label-1, label-2, ..., label-n>

where N refers to the first node in the path. Such a sequence is called a path name. If the first node in a path name is the root of the naming graph, it is called an absolute path name. Otherwise, it is called a relative path name.

Name Resolution

Name spaces offer a convenient mechanism for storing and retrieving information about entities by means of names. More generally, given a path name, it should be possible to look up any information stored in the node referred to by that name. The process of looking up a name is called name resolution.

Name Resolution

- **The process of looking up a name**
- **Closure Mechanism -> Knowing how and where to start name resolution**
- **Mounting -> transparent way for name resolution with different name spaces**
- **Mounted File System -> letting a directory node store the identifier of a directory node from a different name space (foreign name space)**
- **Mount point -> directory node storing the node identifier**
- **Mounting point -> directory node in the foreign name space**

Linking and Mounting

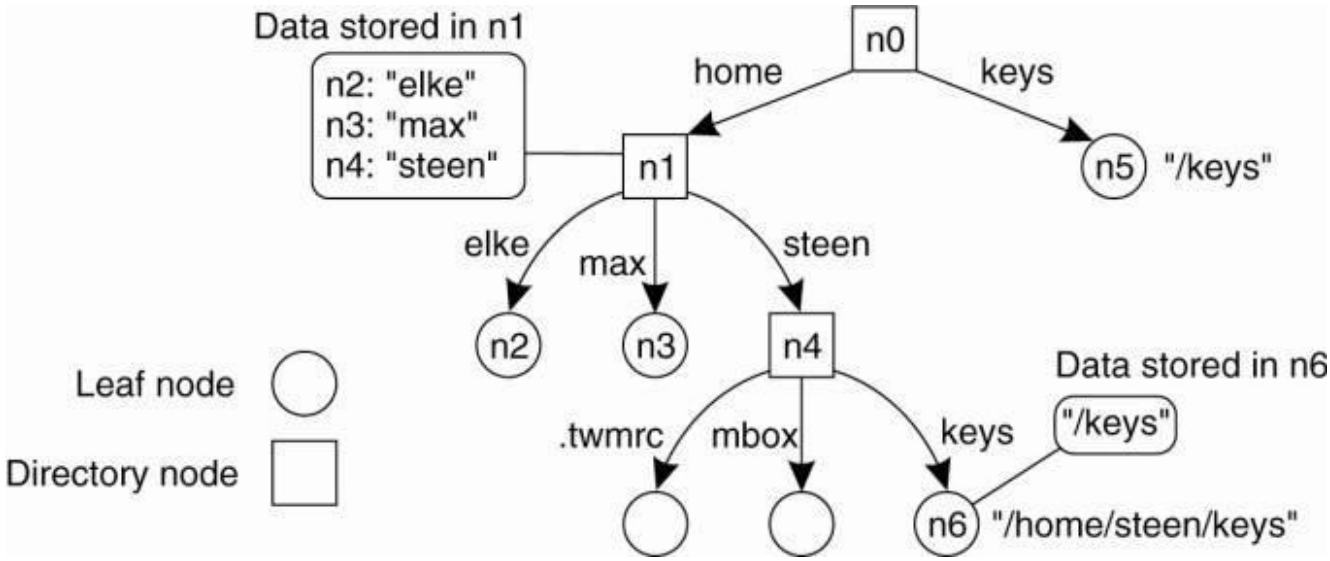
Strongly related to name resolution is the use of aliases. An alias is another name for the same entity. An environment variable is an example of an alias. In terms of naming graphs, there are basically two different ways to implement an alias. The first approach is to simply allow multiple absolute paths names to refer to the same node in a naming graph.

This approach is illustrated in Fig.above, in which node n5 can be referred to by two different path names. In UNIX terminology, both path names /keys and /home/steen/keys in Fig.above are called hard links to node n5.

The second approach is to represent an entity by a leaf node, say N, but instead of storing the address or state of that entity, the node stores an absolute path name. When first resolving an absolute path name that leads to N, name resolution will return the path name stored in N, at which point it can continue with resolving that new path name. This principle corresponds to the use of symbolic links in UNIX file systems, and is illustrated in Fig. 5-11. In this example, the path name /home/steen/keys, which refers to a node containing the absolute path name /keys,

is a symbolic link to node n5.

Figure below shows The concept of a symbolic link explained in a naming graph.



The principle of mounting can be generalized to other name spaces as well. In particular, what is needed is a directory node that acts as a mount point and stores all the necessary information for identifying and accessing the mounting point in the foreign name space. This approach is followed in many distributed file systems.

Consider a collection of name spaces that is distributed across different machines. In particular, each name space is implemented by a different server, each possibly running on a separate machine. Consequently, if we want to mount a foreign name space NS2 into a name space NS1, it may be necessary to communicate over a network with the server of NS2, as that server may be running on a different machine than the server for NS1. To mount a foreign name space in a distributed system requires at least the following information:

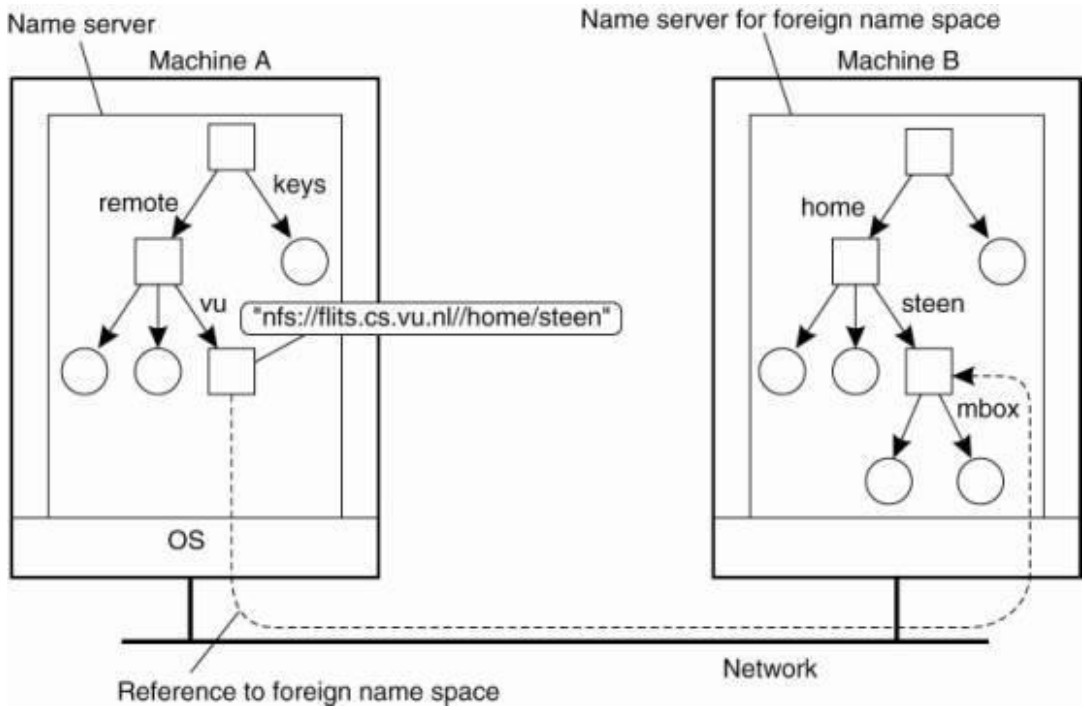
1. **The name of an access protocol.**
2. **The name of the server.**
3. **The name of the mounting point in the foreign name space.**

Note that each of these names needs to be resolved. The name of an access protocol needs to be resolved to the implementation of a protocol by which communication with the server of the foreign name space can take place. The name of the server needs to be resolved to an

address where that server can be reached. As the last part in name resolution, the name of the mounting point needs to be resolved to a node identifier in the foreign name space.

Mounted File System

- During resolution, mounting point is looked up & resolution proceeds by accessing its directory table
- Mounting requires at least
  - Name of an access protocol (for communication)
  - Name of the server (resolved to address)
- Name of mounting point in foreign name space (resolved to node identifier in foreign NS)
- Each of these names needs to be resolved
- Three names can be represented as URL nfs://oslab.khu.ac.kr/home/oufastupdates



The Implementation of a Name Space

A name space forms the heart of a naming service, that is, a service that allows users and processes to add, remove, and look up names. A naming service is implemented by name servers. If a distributed system is restricted to a local area network, it is often feasible to implement a naming service by means of only a single name server. However, in large-scale distributed systems with many entities, possibly spread across a large geographical area, it is necessary to distribute the implementation of a name space over multiple name servers.

Name Space Distribution

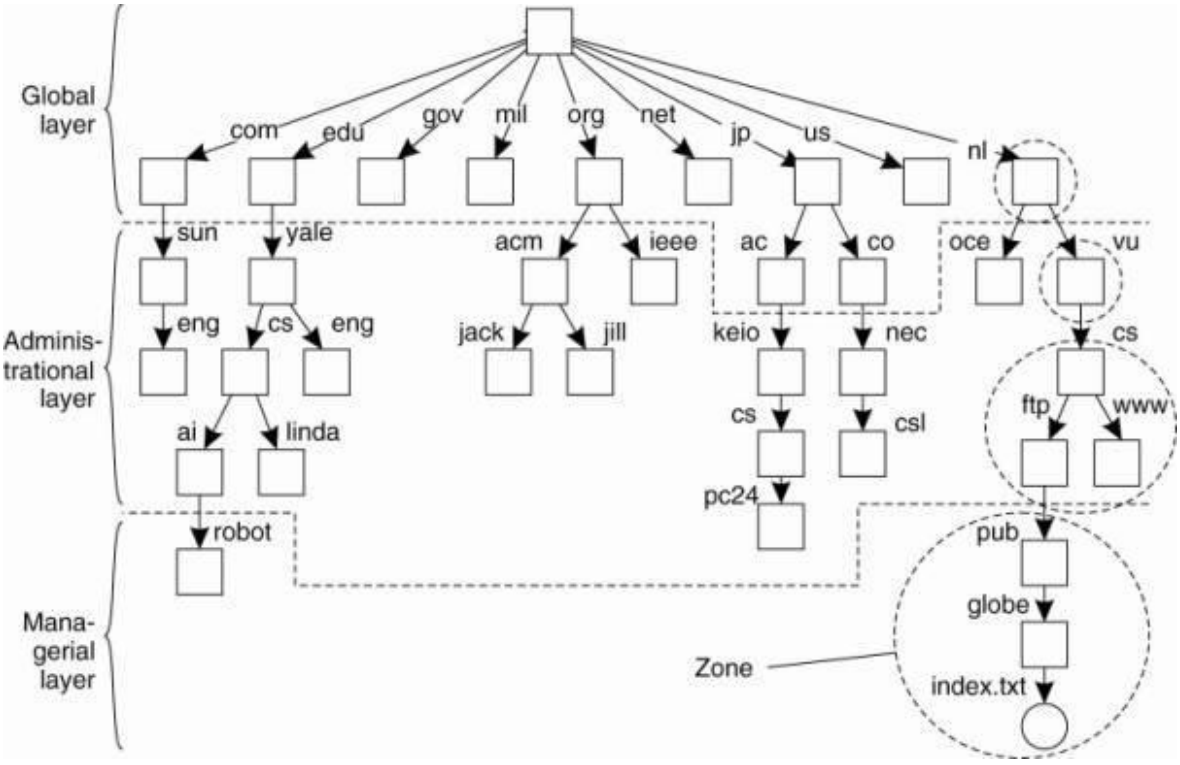
To effectively implement a name space, it is convenient to partition it into logical layers.

The global layer is formed by highest-level nodes, that is, the root node and other directory nodes logically close to the root, namely its children. Nodes in the global layer are often characterized by their stability, in the sense that directory tables are rarely changed. Such nodes may represent organizations, or groups of organizations, for which names are stored in the name space.

The administrational layer is formed by directory nodes that together are managed within a single organization. A characteristic feature of the directory nodes in the administrational layer is that they represent groups of entities that belong to the same organization or administrational unit.
Finally, the managerial layer consists of nodes that may typically change regularly. For example, nodes representing hosts in the local network belong to this layer. For the same reason, the layer includes nodes representing shared files such as those for libraries or binaries. Another important class of nodes includes those that represent user-defined directories and files. In contrast to the global and administrational layer, the nodes in the managerial layer are maintained not only by system administrators, but also by individual end users of a distributed system.

Fig. below shows an example of the partitioning of part of the DNS name space, including the names of files within an organization that can be accessed through the Internet, for example, Web pages and transferable files. The name space is divided into non overlapping parts, called zones in DNS . A zone is a part of the name space that is implemented by a separate name server. Some of these zones are illustrated in Fig.below.

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

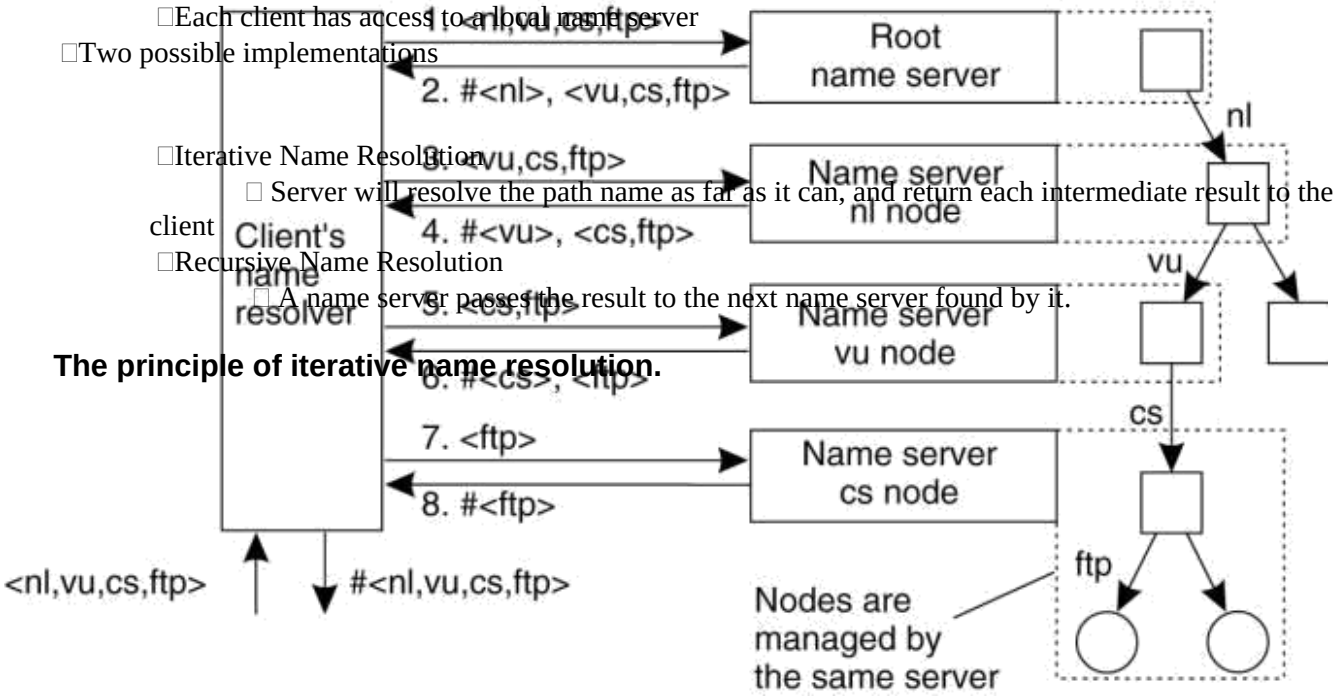


A comparison between name servers for implementing nodes from a large-scale name space partitioned into a global layer, an administrational layer, and a managerial layer.

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

Implementation of Name Resolution

- Assumptions
  - No replication of name servers
  - No client side caching
  - Each client has access to a local name server
- Two possible implementations
  - Iterative Name Resolution
    - Server will resolve the path name as far as it can, and return each intermediate result to the client
  - Recursive Name Resolution
    - A name server passes the result to the next name server found by it.



The principle of iterative name resolution.

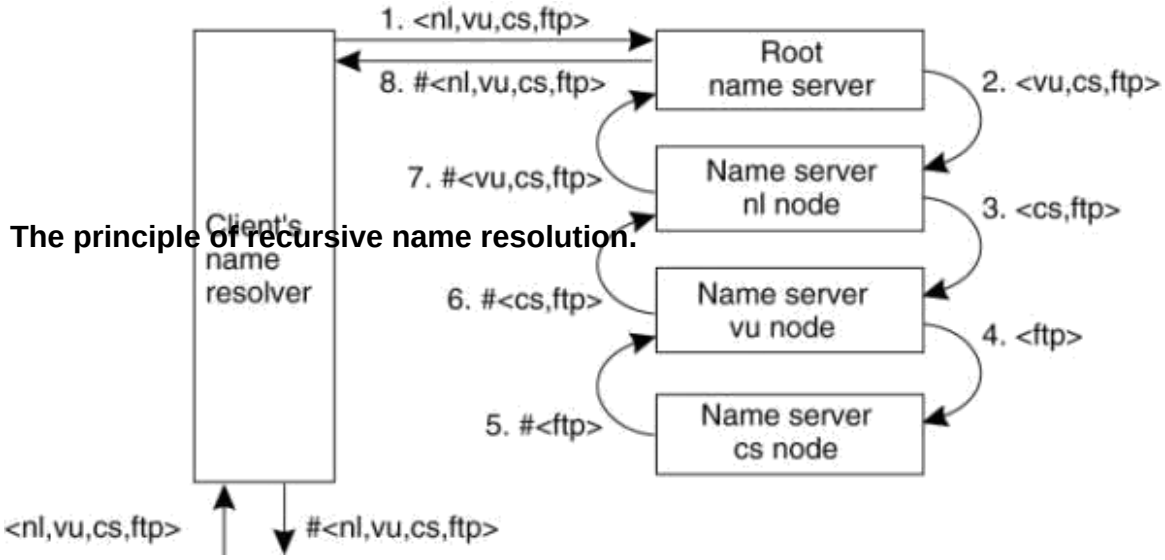
An alternative to iterative name resolution is to use recursion during name resolution. Instead of returning each intermediate result back to the client's name resolver, with recursive name resolution, a name server passes the result to the next name server it finds.

Advantages

- Less burden on name sever

Disadvantage

- More communication cost



The principle of recursive name resolution.

The main drawback of recursive name resolution is that it puts a higher performance demand on each name server. Basically, a name server is required to handle the complete resolution of a path name, although it may do so in cooperation with other name servers. This additional burden is generally so high that name servers in the global layer of a name space support only iterative name resolution.

There are two important advantages to recursive name resolution. The first advantage is that caching results is more effective compared to iterative name resolution. The second advantage is that communication costs may be reduced.

The Domain Name System

One of the largest distributed naming services in use today is the Internet Domain Name System (DNS). DNS is primarily used for looking up IP addresses of hosts and mail servers.

The DNS Name Space

The DNS name space is hierarchically organized as a rooted tree. A label is a case-insensitive string made up of alphanumeric characters. A label has a maximum length of 63 characters; the length of a complete path name is restricted to 255 characters. The string representation of a path name consists of listing its labels, starting with the rightmost one, and separating the labels by a dot ("."). The root is represented by a dot. So, for example, the path name root:<nl, vu, cs, flits>, is represented by the string

flits.cs.vu.nl., which includes the rightmost dot to indicate the root node. We generally omit this dot for readability.

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

Type of record	Associated entity	Description
SOA	Zone	Holds information on the represented zone
A	Host	Contains an IP address of the host this node Represents
MX	Domain	Refers to a mail server tohandle mail addressed to this node
SRV	Domain	Refers to a server handling a specific service
NS	Zone	Refers to a name server that implements the represented zone
CNAME	Node	Symbolic link with the primary name of the represented node
PTR	Host	Contains the canonical name of a host
HINFO	Host	Holds information on the host this node Represents
TXT	Any kind	Contains any entity-specific information considered useful

DNS Implementation

In essence, the DNS name space can be divided into a global layer and an administrational layer. The managerial layer, which is generally formed by local file systems, is formally not part of DNS and is therefore also not managed by it.

Each zone is implemented by a name server, which is virtually always replicated for availability. Updates for a zone are normally handled by the primary name server. Updates take place by modifying the DNS database local to the primary server. Secondary name servers do not access the database directly, but, instead, request the primary server to transfer its content. The latter is called a zone transfer in DNS terminology.

A DNS database is implemented as a (small) collection of files, of which the most important one contains all the resource records for all the nodes in a particular zone. This approach allows nodes to be simply identified by means of their domain name, by which the notion of a node identifier reduces to an (implicit) index into a file.

Attribute-Based Naming

Flat and structured names generally provide a unique and location-independent way of referring to entities. Moreover, structured names have been partly designed to provide a human-friendly way to name entities so that they can be conveniently accessed. In most cases, it is assumed that the name refers to only a single entity. However, location independence and human friendliness are not the only criterion for naming entities. In particular, as more information is being made available it becomes important to effectively search for entities. This approach requires that a user can provide merely a description of what he is looking for.

Directory Services

Attribute-based naming systems are also known as directory services, whereas systems that support structured naming are generally called naming systems. With directory services, entities have a set of associated attributes that can be used for searching. In some cases, the choice of attributes can be relatively simple. For example, in an e-mail system, messages can be tagged with attributes for the sender, recipient, subject, and so on.

Hierarchical Implementations: LDAP

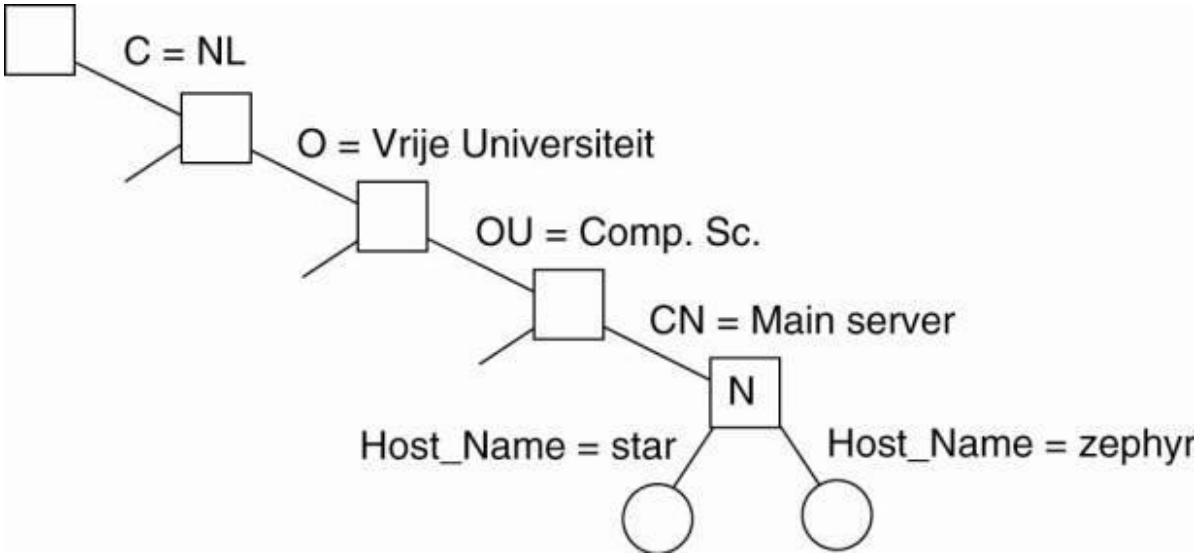
A common approach to tackling distributed directory services is to combine structured naming with attribute-based naming. This approach has been widely adopted, for example, in Microsoft's Active Directory service and other systems. Many of these systems use, or rely on the lightweight directory access protocol commonly referred simply as LDAP. The LDAP directory service has been derived from OSI's X.500 directory service.
Conceptually, an LDAP directory service consists of a number of records, usually referred to as directory entries. A directory entry is comparable to a resource record in DNS. Each record is made up of a collection of (attribute, value) pairs, where each attribute has an associated type. A distinction is made between single-valued attributes and multiple-valued attributes. The latter typically represent arrays and lists.

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

Attribute	Abbr.	Value
Country	C	NL
Locality	L	Amsterdam
Organization	O	Vrije Universiteit
OrganizationalUnit	OU	Comp. Sc.
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

The collection of all directory entries in an LDAP directory service is called a directory information base (DIB). An important aspect of a DIB is that each record is uniquely named so that it can be looked up. Such a globally unique name appears as a sequence of naming attributes in each record. Each naming attribute is called a relative distinguished name, or RDN for short.

(a) Part of a directory information tree. (b) Two directory entries having Host_Name as RDN.



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

(b)

Node N corresponds to the directory entry shown in Fig.. At the same time, this node acts as a parent to a number of other directory entries that have an additional naming attribute Host_Name that is used as an RDN. For example, such entries may be used to represent hosts as shown in (b).

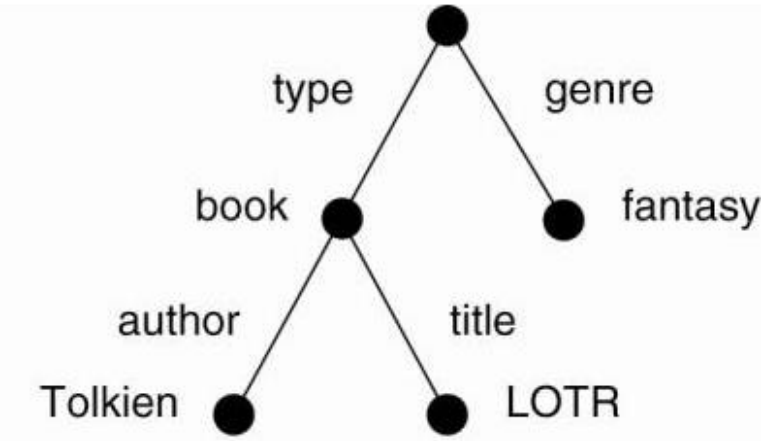
Decentralized Implementations

With the advent of peer-to-peer systems, researchers have also been looking for solutions for decentralized attribute-based naming systems. The key issue here is that (attribute, value) pairs need to be efficiently mapped so that searching can be done efficiently, that is, by avoiding an exhaustive search through the entire attribute space

Figure (a) A general description of a resource. (b) Its representation as an AVTree.

```
description {
    type = book
    description {
        author = Tolkien
        title = LOTR
    }
    genre = fantasy
}
```

(a)



(b)