Advanced Operating Systems

Chapter 4: **NAMING**

Contents

Names, Identifiers, and Addresses

- **Name**

A name in a distributed system is a string of bits or characters used to refer to an entity.

- **Entities:**

An entity in a distributed system can be practically anything

For example: hosts, printers, disks, files, processes, users, mailboxes, web pages, graphical windows, messages, network connections, etc.

Names, Identifiers, and Addresses

Entities can be operated on. To operate on an entity, it is necessary to access it, for which we need an **access point** (address).

An entity can offer more than one access point.

As a comparison, a telephone can be viewed as an access point of a person, whereas the telephone number corresponds to an address.

In a distributed system, a typical example of an access point is a host running a specific server, with its address formed by the combination of, for example, an IP address and port number

An address is just a special kind of name

Use the address as a name:

An entity may change its access points in the course of time.

For example when a mobile computer moves to another location, it is often assigned a different IP address than the one it had before.

So, **If an address is used to refer to an entity**, we will have an invalid reference the instant the access point changes or is reassigned to another entity.

if **an entity offers more than one access point**, it is not clear which address to use as a reference.

A name for an entity that is independent from its addresses is often much easier and more flexible to use.

Names, Identifiers, and Addresses

Identifier

A true identifier is a name that is used to uniquely identify an entity

Properties of a true identifier:

- 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)

By using identifiers, it becomes much easier to unambiguously refer to an entity.

Types of Naming Systems

Flat naming: The identifier is simply a random bit string.

It does not contain any information whatsoever on how to locate an access point of its associated entity.

Structured naming: Composed of simple human-readable names.

Examples are file system naming and host naming on the Internet.

Attribute-based naming: Allows an entity to be described by (attribute, value) pairs. This allows a user to search more effectively by constraining some of the attributes.

Flat Naming Systems

- Broadcasting and multicasting
- Forwarding pointers
- Home-based approaches
- Distributed hash tables
- Hierarchical approaches

Broadcasting/Multicasting

- Consider a LAN that offers efficient ***broadcasting*** facility.

A message containing the identity of the entity is broadcast to each machine.

Only the machines that can offer access to that entity send a reply containing the address of the access point. For example: ARP (Address Resolution Protocol)

Problem:
- Broadcasting is not suitable for larger networks
- Bandwidth is wasted

- A more efficient approach for larger networks is ***multicasting***, by which only a restricted group of machines receive the request.

For example: data-link level in Ethernet networks

Forwarding Pointers

- 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 using a traditional naming service, a client can look up the current address by following the chain of forwarding pointers.

Example: Remote objects that can move from host to host.

Forwarding Pointers

- A server stub contains either a local reference to the actual object or a local reference to a remote client stub for that object.
- Whenever an object moves from address A to B, it leaves behind a client stub in its place on A and installs a server stub that refers to it in B.

This makes the migration completely transparent to a client.

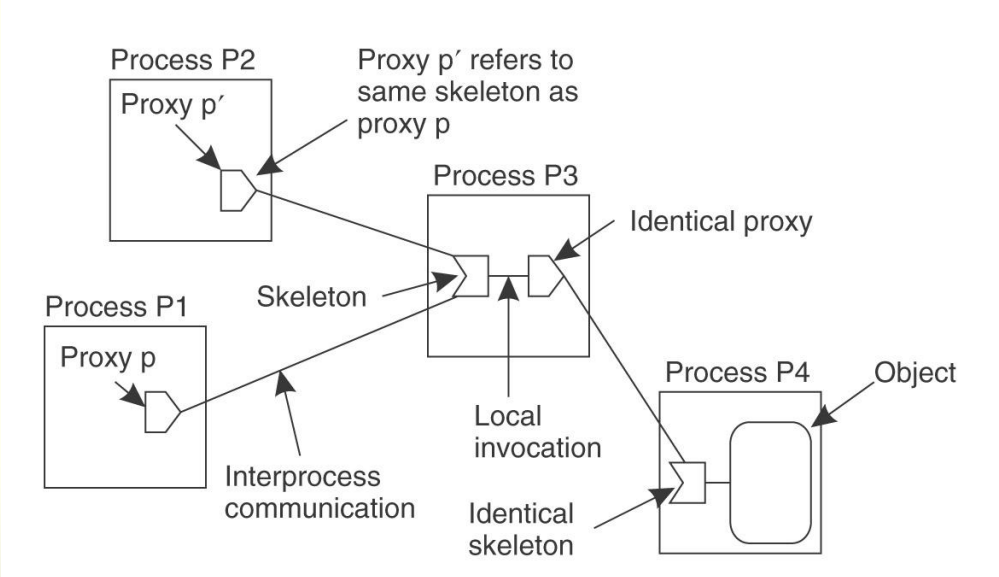

Fig 18: The principle of forwarding pointers using (client stub, server stub) pairs

Forwarding Pointers (important drawbacks)

Problems:

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.

It is important to keep the forwarding chains relatively short and to ensure that the forwarding pointers are robust.

Home-Based Approach

A popular approach to supporting mobile entities in large-scale networks is to introduce a **home location**, which keeps track of the current location an entity.

In practice, the home location is often chosen to be the place where an entity was created.

The home-based approach is used as a fall-back mechanism for location services based on forwarding pointers.

Home-Based Approach

An example where the home-based approach is followed is in Mobile IP.

Each mobile host uses a fixed IP address.
All communication to that IP address is initially directed to the mobile host's home agent.

This home agent is located on the local-area network corresponding to the network address contained in the mobile host's IP address.

Whenever the mobile host moves to another network, it requests a temporary address that it can use for communication.

This care-of address is registered at the home agent.

When the home agent receives a packet for the mobile host, it looks up the host's current location.

Home-Based Approach

If the host is on the current local network, the packet is simply forwarded.

Otherwise, it is tunneled to the host's current location, that is, wrapped as data in an IP packet and sent to the care-of address.

At the same time, the sender of the packet is informed of the host's current location.

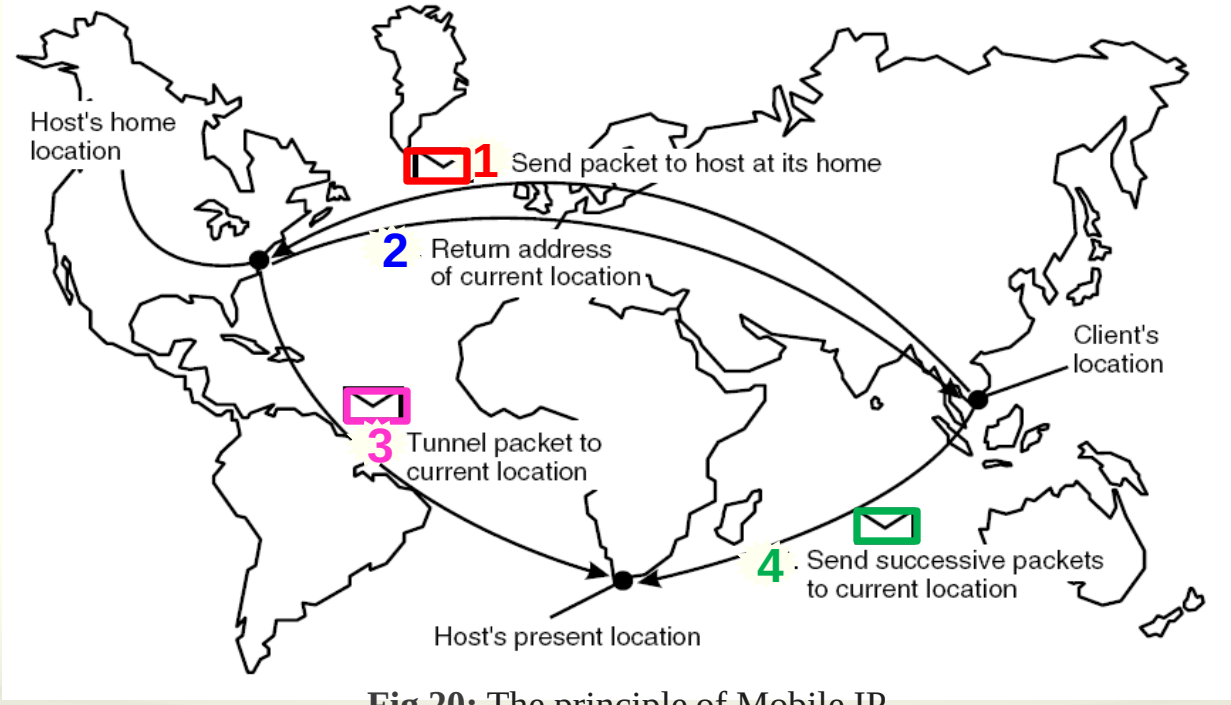

Fig 20: The principle of Mobile IP.

Home-Based Approach (important drawbacks)

To communicate with a mobile entity, a client first has to contact the home, which may be at a completely different location than the entity itself.

The result is an increase in communication latency.

A drawback of the home-based approach is the use of a fixed home location.

For one thing, it must be ensured that the home location always exists. Otherwise, contacting the entity will become impossible.

Distributed Hash Tables

➢A distributed technique on resolving an identifier to the address of an associated entity.

➢Selected applications

- BTDigg: BitTorrent DHT search engine
- Oracle Coherence: An in-memory data grid built on a Java DHT Implementation
- WebSphere eXtreme Scale: proprietary DHT implementation by IBM, used for object caching
- YaCy: Java-based distributed search engine

Distributed Hash Tables

How to efficiently resolve a key k to the address of succ(k)?

- Unbounded binary search: Each node maintains a finger table of at most m entries

$$FT_p[i] = succ(p + 2^{i-1})$$

- The ith entry points to the first node succeeding p by at least 2^{i-1} these are exponentially increasing short-cuts in the identifier space
- To look up a key k, a node p will then forward the request to node q with index j in p's finger table where:

$$q = FT_p[j] \leq k < FT_p[j+1]$$

- Uses modulo arithmetic

Distributed Hash Tables

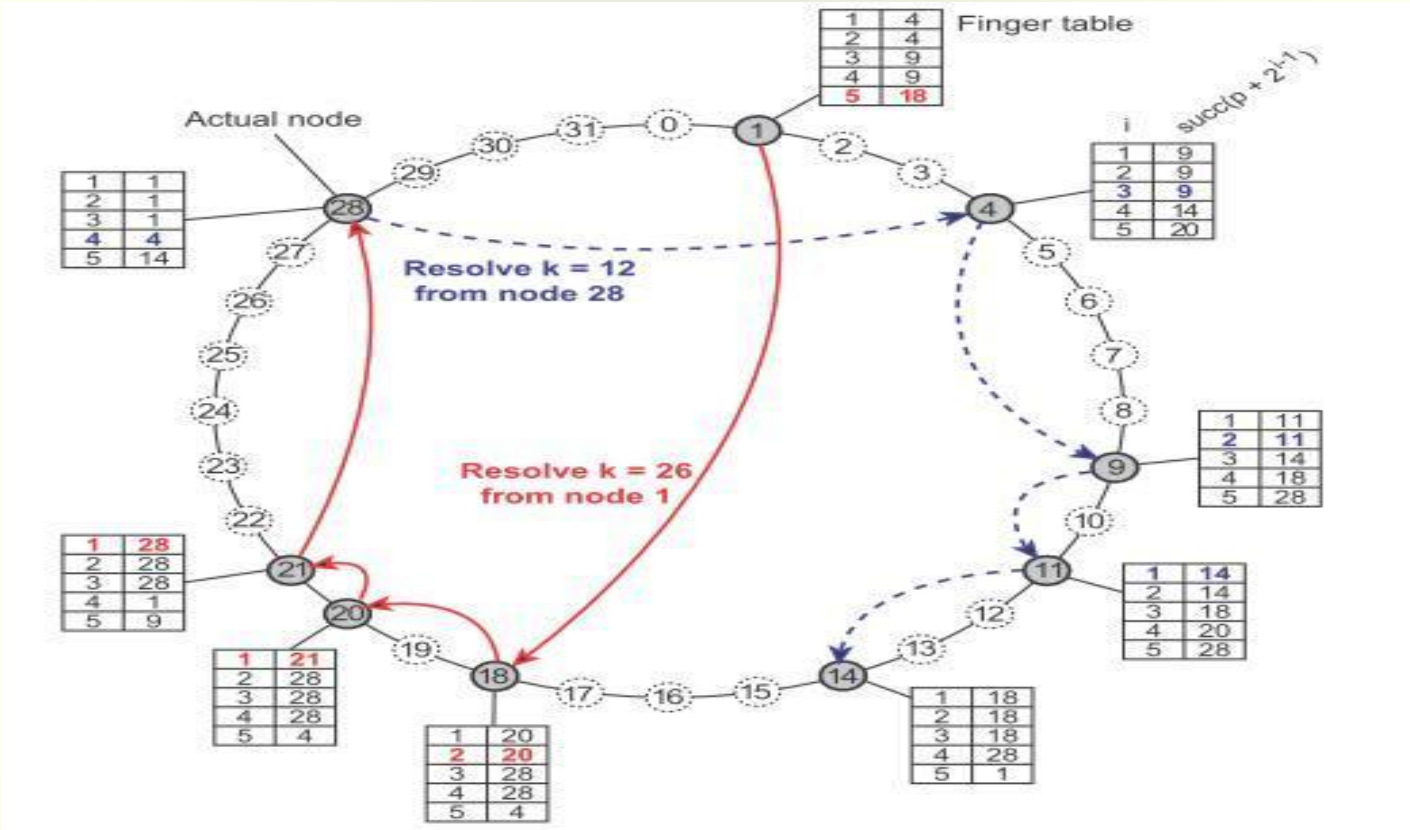

Fig : Resolving key 26 from node 1 and key 12 from node 28 in a Chord system

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 sub domains.

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.

Hierarchical Approaches

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 sub domain where that record's associated entity is currently located.

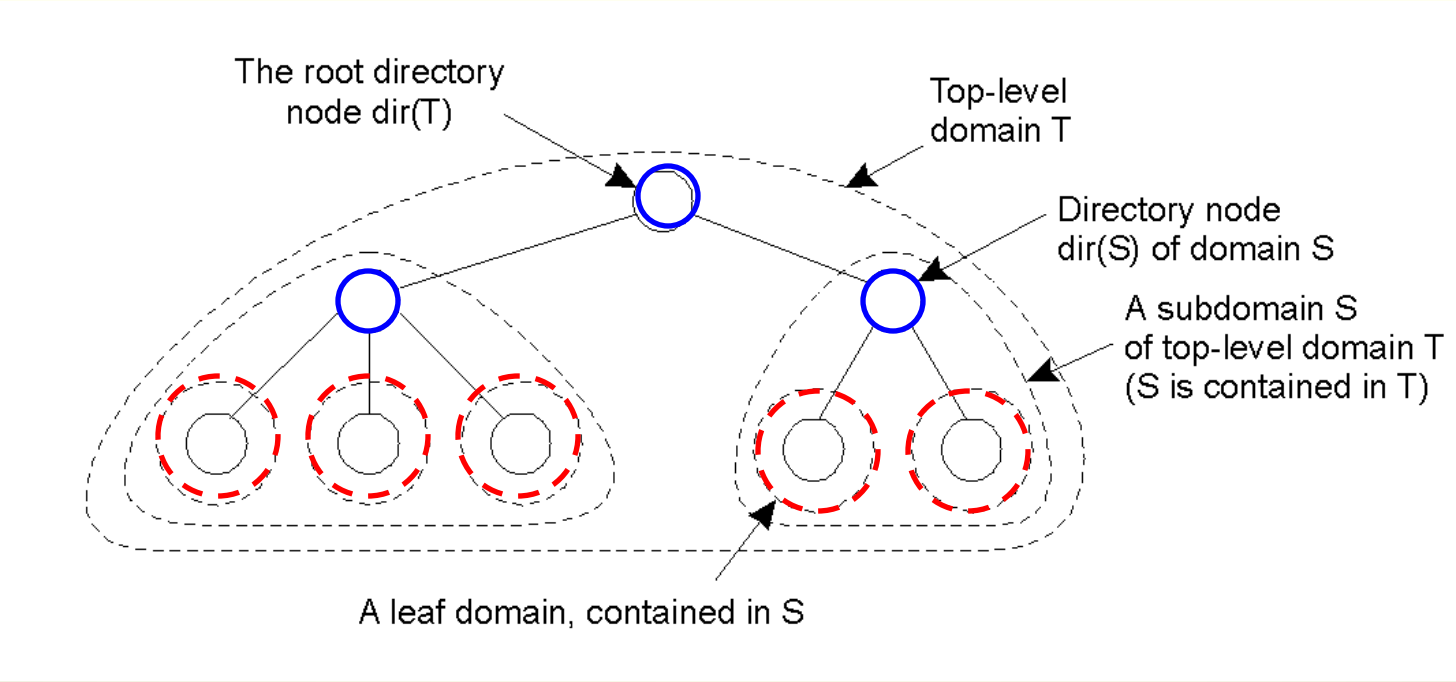

Fig21:Hierarchical organization of a location service into domains, each having an associated directory node.

Hierarchical Approaches

An entity may have multiple addresses, for example if it is replicated.
If an entity has an address in leaf domain *D 1 and D2 respectively, then the directory* node of the smallest domain containing both *D 1 and D 2, will have two pointers* one for each sub domain containing an address.

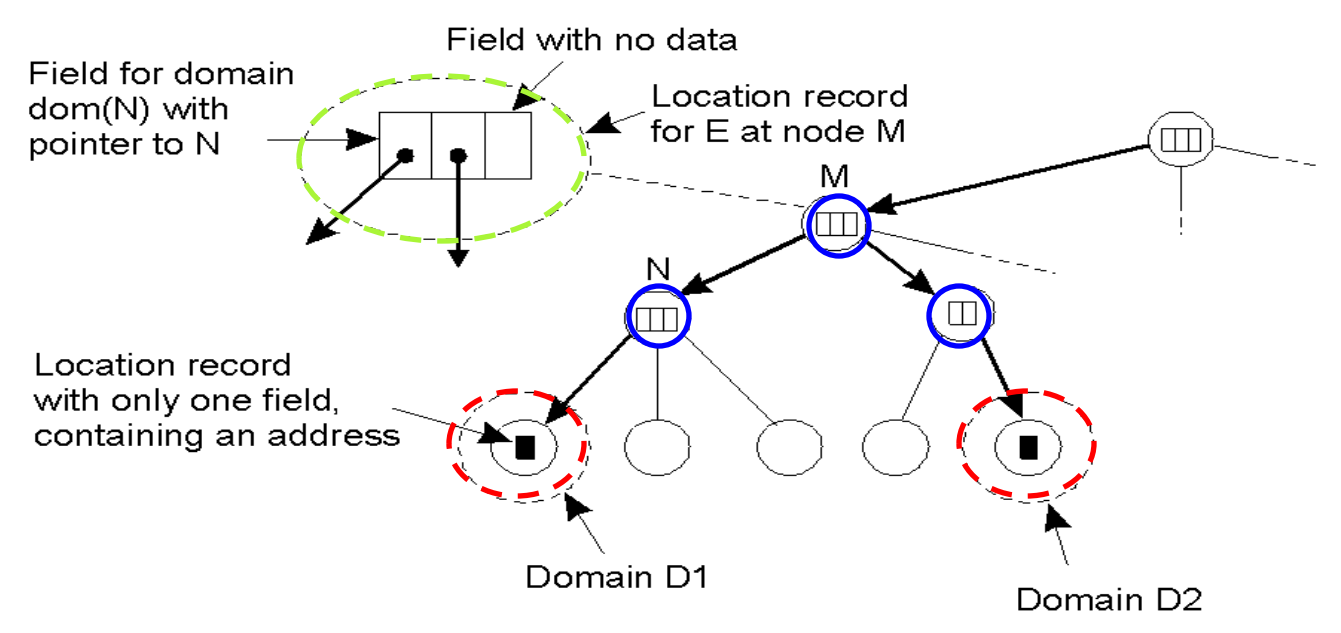

Fig21:An example of storing information of an entity having two addresses in different leaf domains.

Hierarchical Approaches/Looking up a location in a hierarchically organized location service

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.

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 figure, M is shown to store a location record containing a pointer to one of its sub domains.

Hierarchical Approaches/Looking up a location in a hierarchically organized location service

The lookup request is then forwarded to the directory node of that sub domain, 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.

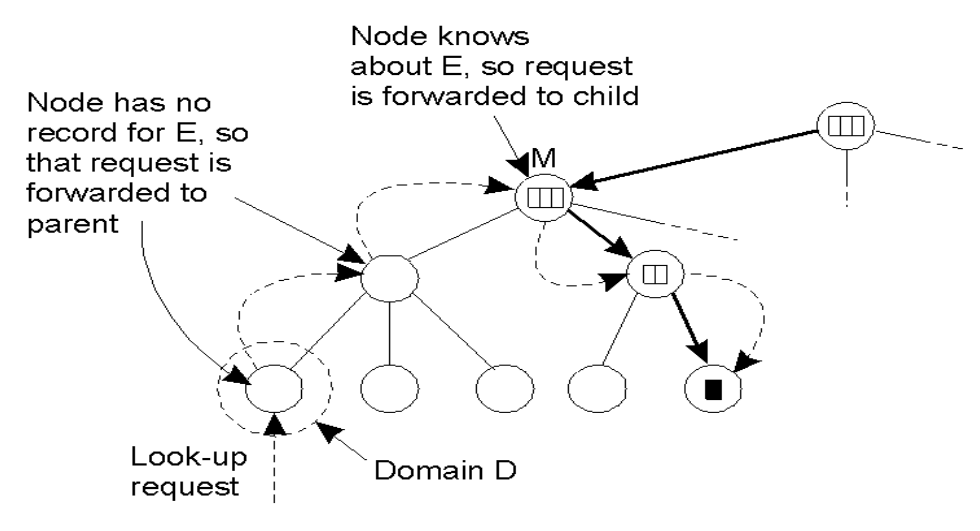

Fig21:Looking up a location in a hierarchically organized location service.

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 in a distributed system are organized into what is commonly referred to as a **name space.**

A name space 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.

A **directory node** has a number of outgoing edges , each labeled with a name.

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.

name spaces

Path name N:<label-1, label-2,…, label-n>

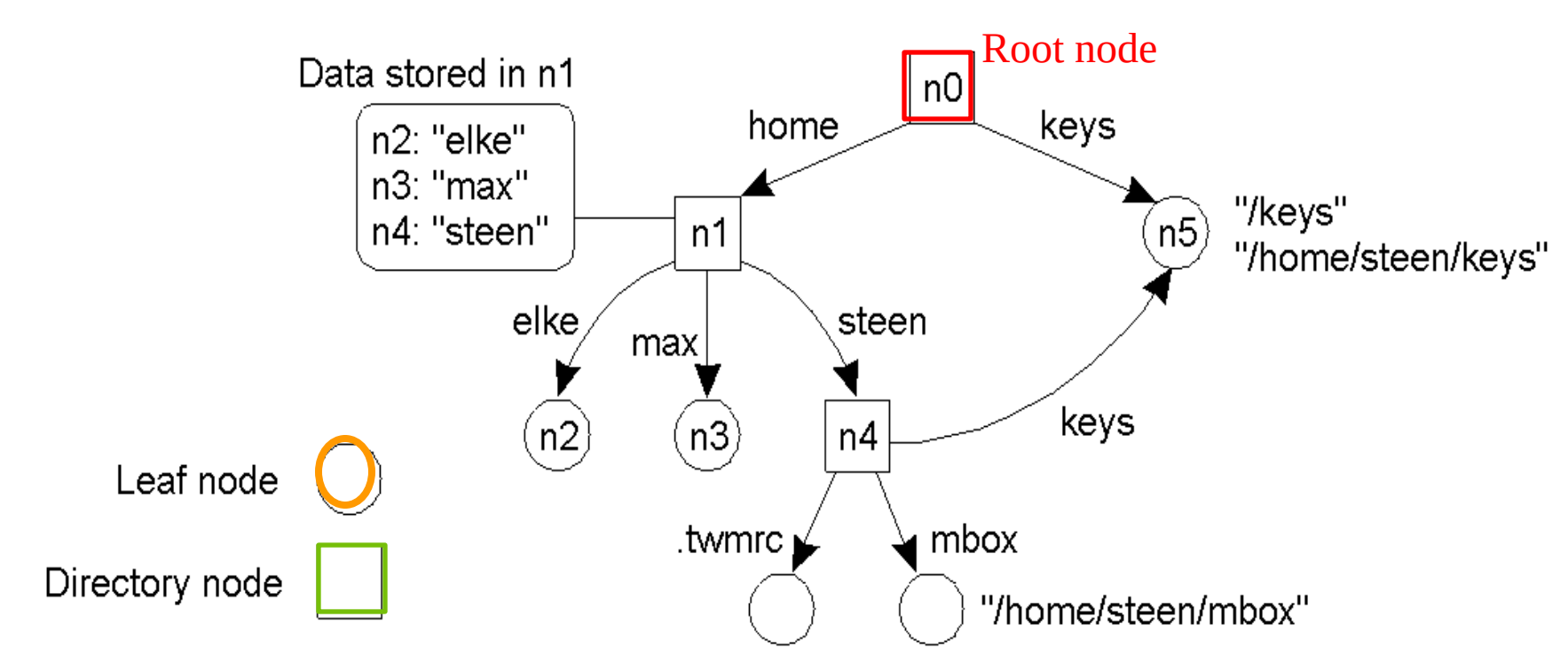

Fig 1: A general naming graph with a single root node

Name Resolution & Closure mechanism

- 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**.

- Knowing how and where to start name resolution is generally referred to as a **closure mechanism.**

Essentially, a closure mechanism deals with selecting the initial node in a name space which name resolution is to start.

Linking and Mounting

Name resolution can be used to merge **different name spaces** in a transparent way.

Let us first consider a mounted file system.
In terms of our naming model, a mounted file system corresponds to letting a directory node store the identifier of a directory node from a different name space, which we refer to as a foreign name space.

- *The directory node storing the node identifier is called a **mount point.***
- *Accordingly, the directory node in the foreign name space is called a **mounting point.***

During name resolution, the mounting point is looked up and resolution proceeds by accessing its directory table.

Linking and Mounting

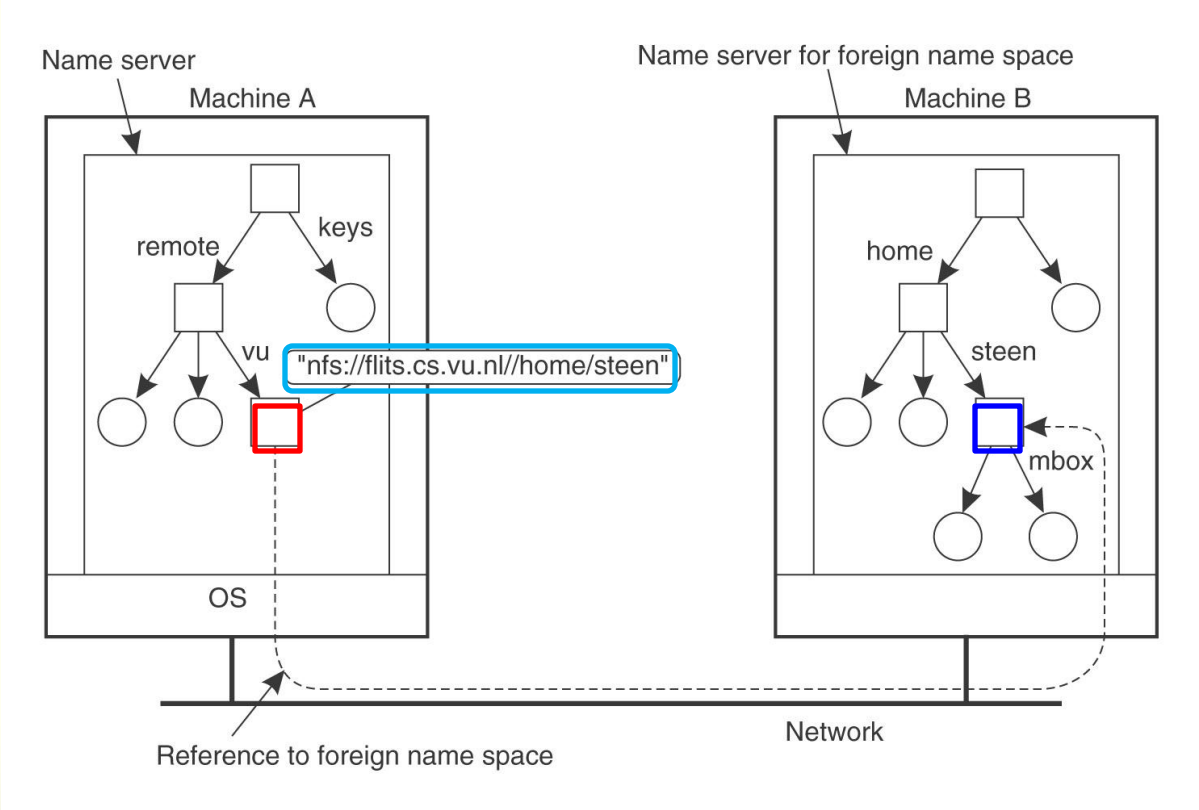

Fig 4: Mounting remote name spaces through a specific access protocol

Linking and Mounting

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 NS 2 into a name space NS 1, it may be necessary to communicate over a network with the server of NS 2, 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.

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

- Global Layer
- Administration Layer
- Manegerial Layer

Name Space Distribution

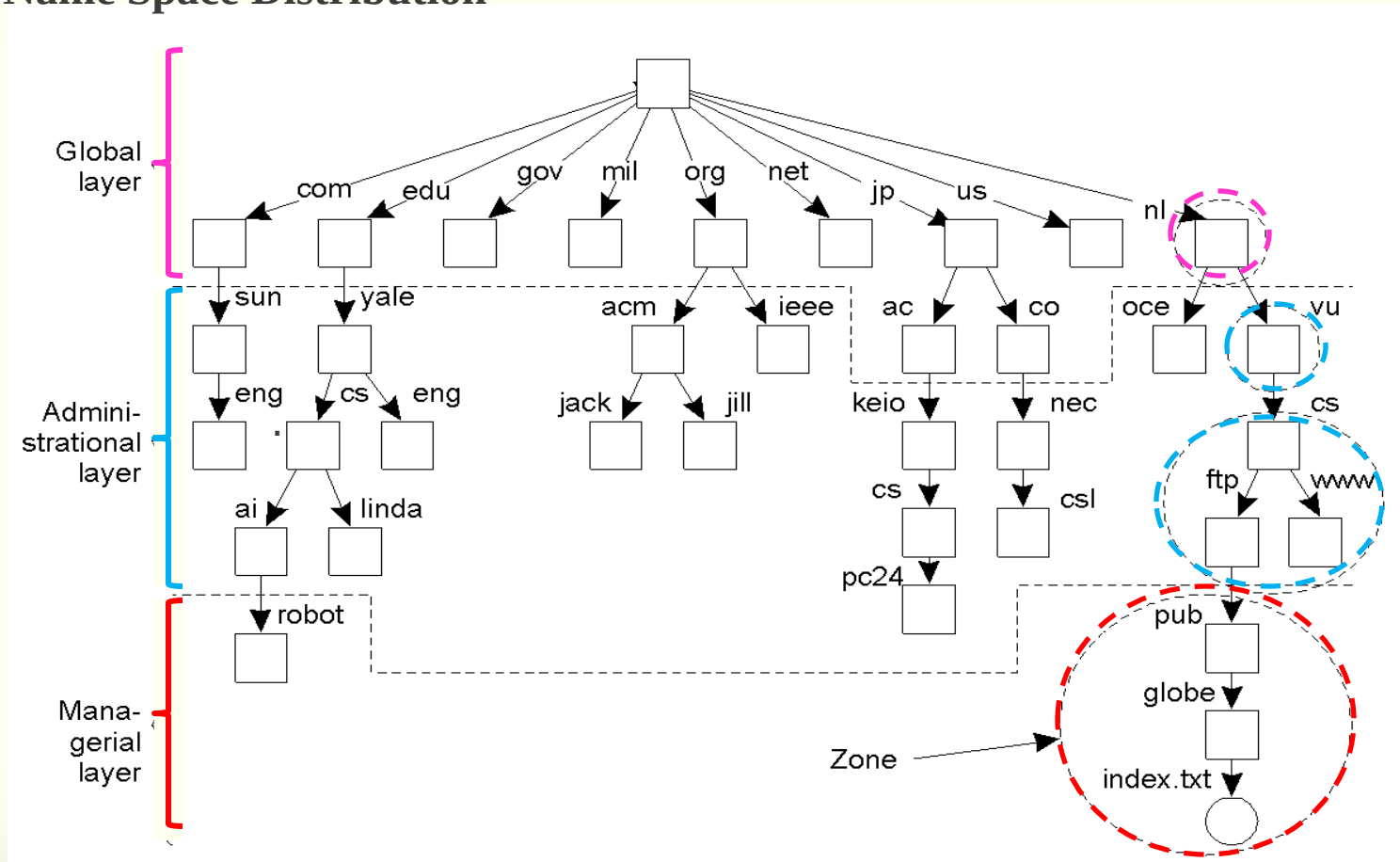

Fig 6: An example partitioning of the DNS name space, including Internet-accessible files, into three layers

Name Space Distribution/ Global layer

Global layer :

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.

Name Space Distribution/ Administrational layer

Administrational layer:

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.

For example, there may be a directory node for each‘ department in an organization, or a directory node from which all hosts can be found.

Another directory node may be used as the starting point for naming all users, and so forth.

The **nodes** in the administrational layer are **relatively stable**, although changes generally occur more frequently than to nodes in the global layer.

Name Space Distribution/ Managerial layer

Managerial layer:

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.

Name Space Distribution/ Comparison

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

Fig 7: 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.

Implementation of Name Resolution

Each client has access to a local name resolver, which is responsible for ensuring that the name resolution process is carried out.

Assume the (absolute) path name root: «nl, VU, CS, ftp, pub, globe, index.html> is to be resolved.

Using a URL notation, this path name would correspond to ftp://ftp.cs. vu.nl/pub/globe/index.html.

- ***There are now two ways to implement name resolution:***
  - **Iterative name resolution**
  - **Recursive name resolution**

Iterative name resolution

In iterative name resolution, a name resolver hands over the complete name to the root name server.
It is assumed that the address where the root server can be contacted is well known.
The root server will resolve the path name as far as it can, and return the result to the client.

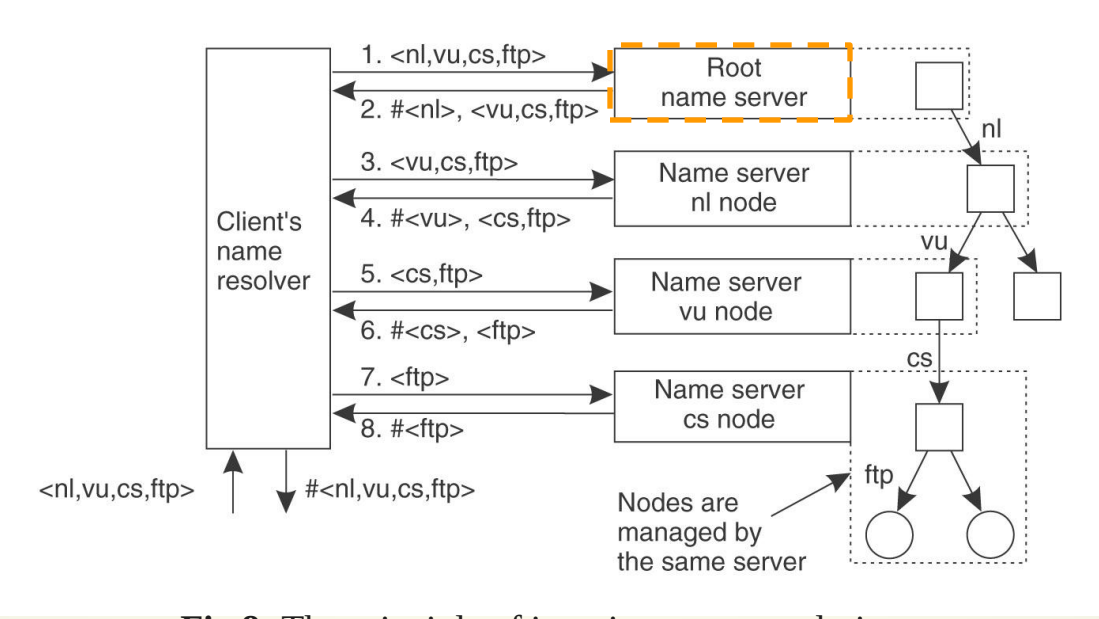

Fig 8: The principle of iterative name resolution.

Recursive name resolution

In recursive name resolution, Instead of returning each intermediate result back to the client's name resolver, a name server passes the result to the next name server it finds.

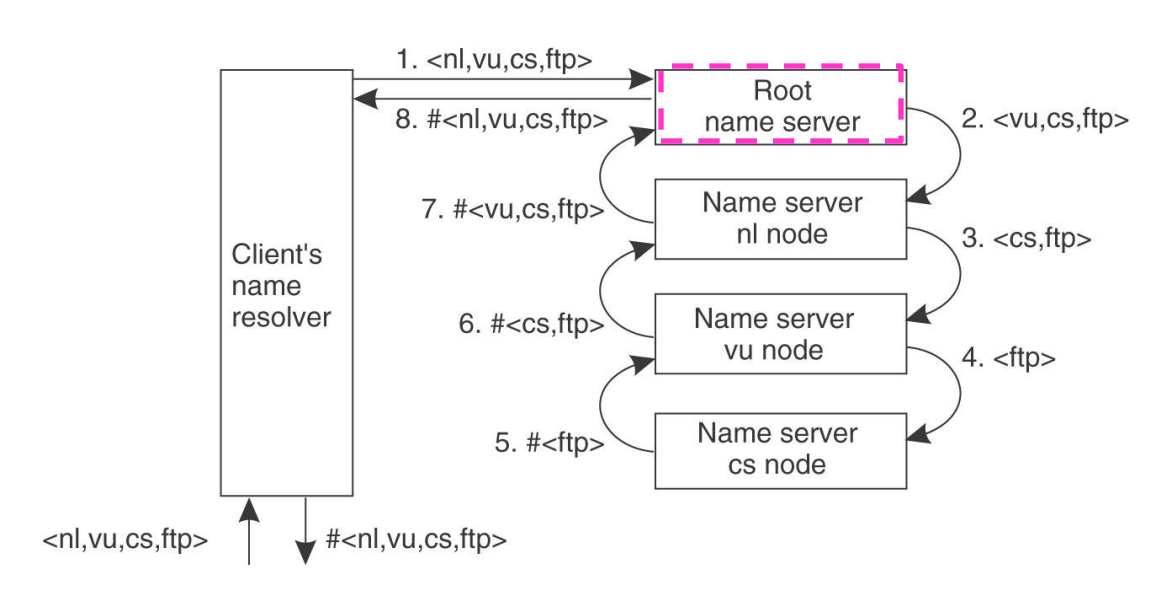

Fig 9: The principle of recursive name resolution.

Recursive name resolution

Drawback

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.

Advantages

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.

Attribute-based Naming

Flat and structured names generally provide a unique and location-independent way of referring to entities

as more information is being made available it becomes important to effectively search for entities describe an entity in terms of *(attribute, value)* pairs, generally referred to as attribute-based

Each attribute says something about entity.

By specifying which values a specific attribute should have, a user essentially constrains the set of entities that he is interested in Attribute-based naming systems are also known as directory services

With directory services, entities have a set of associated attributes that can be used for searching.

End of Chapter 4