Naming

Distributed Systems

FACULTY OF INFORMATION TECHNOLOGY

Naming

Essence

Names are used to denote entities in a distributed system. To operate on an entity, we need to access it at an access point. Access points are entities that are named by means of an address.

Note

A location-independent name for an entity E, is independent from the addresses of the access points offered by E.

Identifiers

Pure name

A name that has no meaning at all; it is just a random string. Pure names can be used for comparison only.

Identifier: A name having some specific properties

1. An identifier refers to at most one entity.
2. Each entity is referred to by at most one identifier.
3. An identifier always refers to the same entity (i.e., it is never reused).

Observation

An identifier need not necessarily be a pure name, i.e., it may have content.

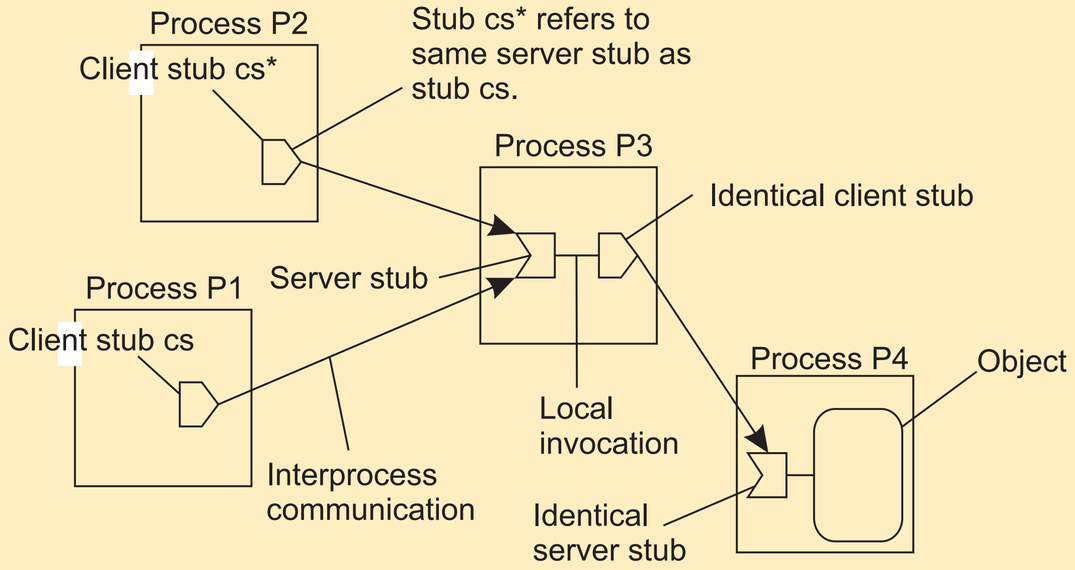
Forwarding pointers

When an entity moves, it leaves behind a pointer to its next location

- Dereferencing can be made entirely transparent to clients by simply following the chain of pointers
- Update a client's reference when present location is found
- Geographical scalability problems (for which separate chain reduction mechanisms are needed):
  - Long chains are not fault tolerant
  - Increased network latency at dereferencing

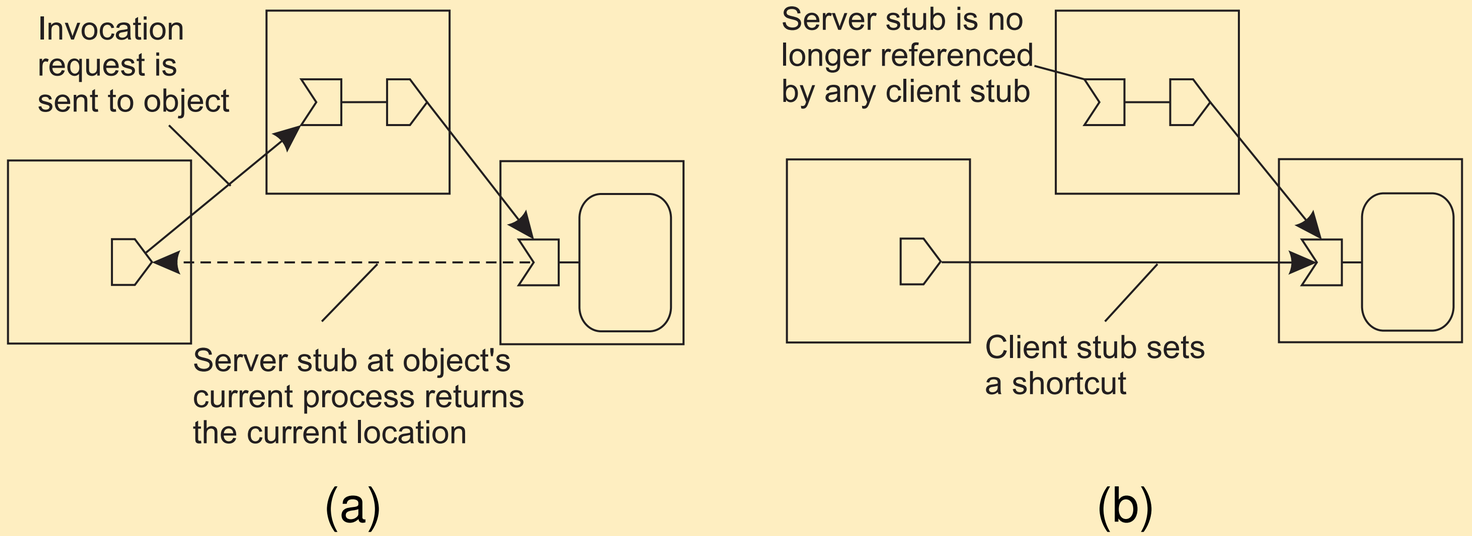
Example: SSP chains

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

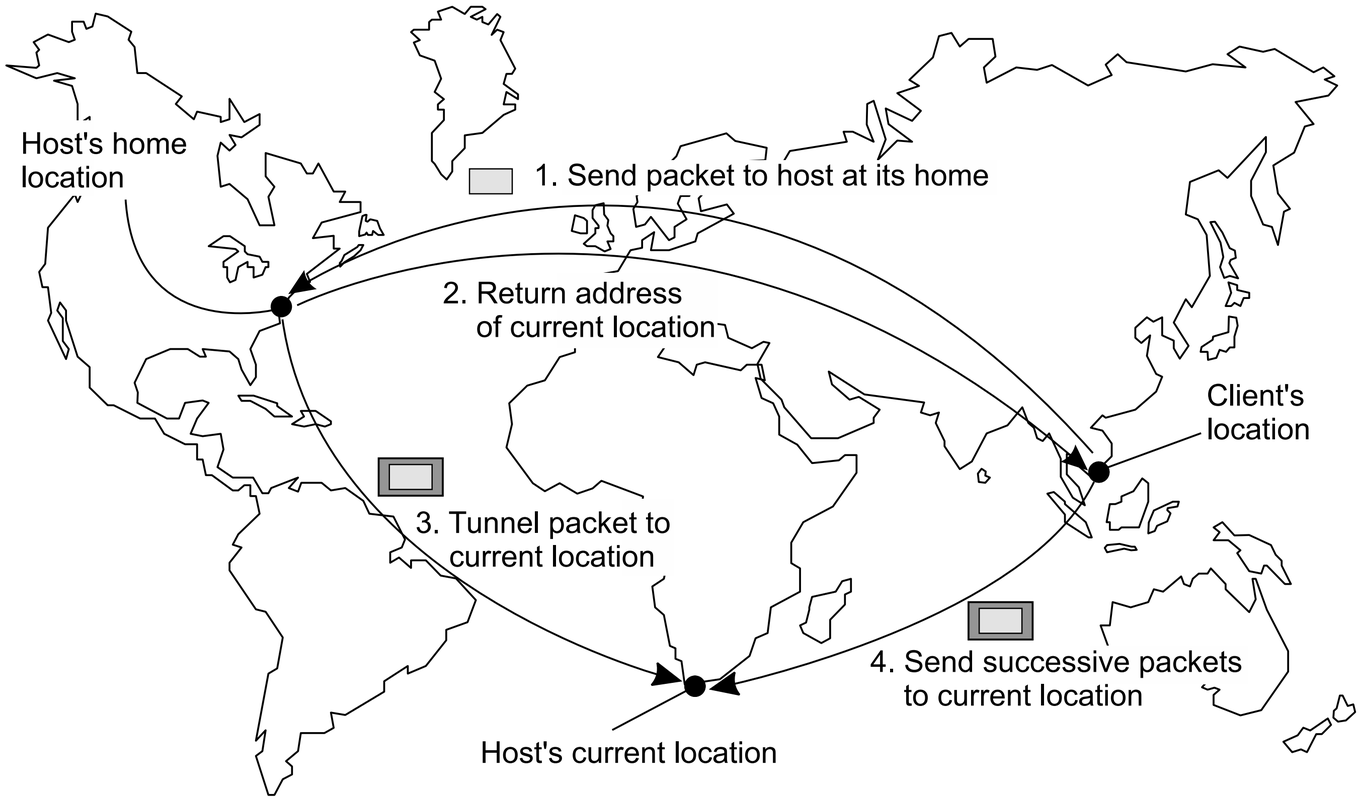


Example: SSP chains

Redirecting a forwarding pointer by storing a shortcut in a client stub



The principle of mobile IP

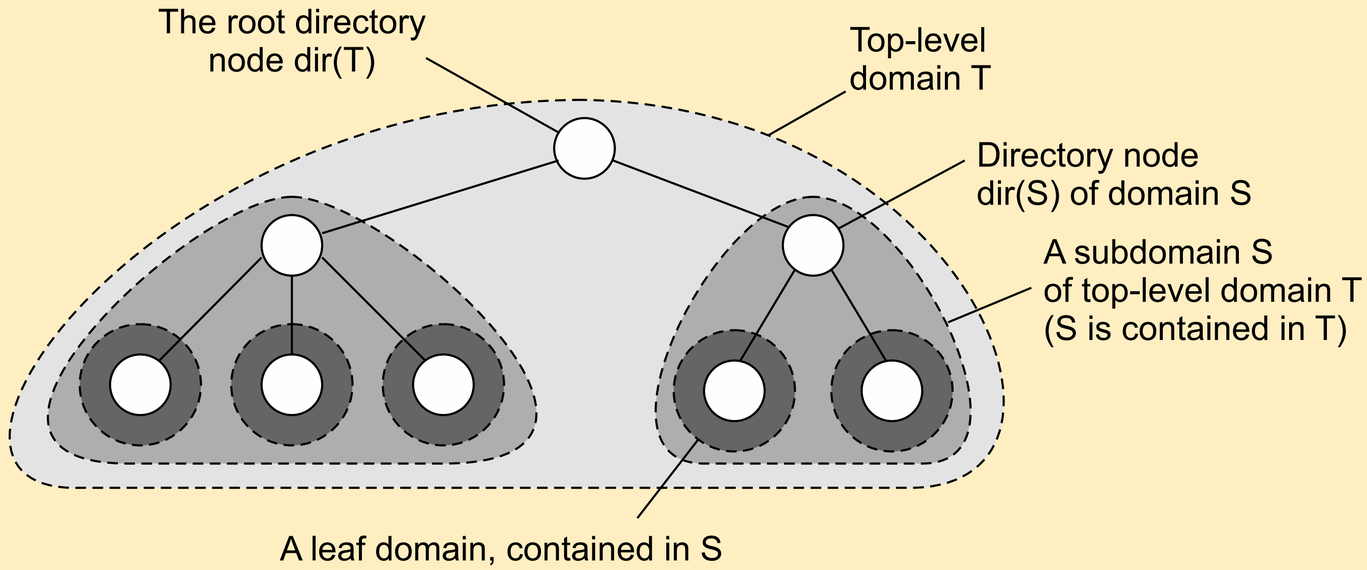


Hierarchical Location Services (HLS)

Basic idea

Build a large-scale search tree for which the underlying network is divided into hierarchical domains. Each domain is represented by a separate directory node.

Principle

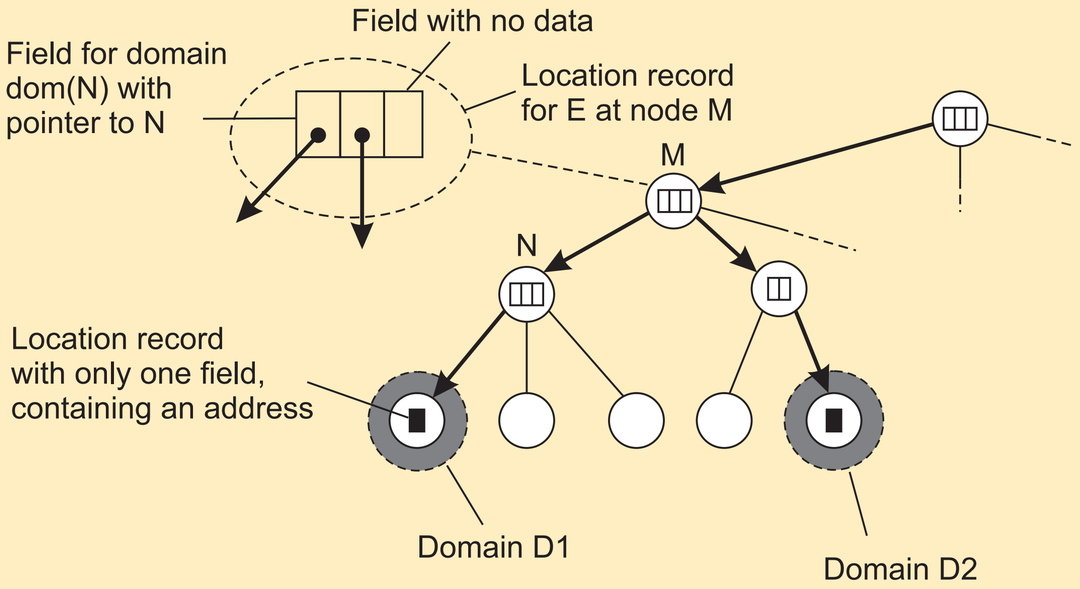


HLS: Tree organization

Invariants

- Address of entity *E* is stored in a leaf or intermediate node
- Intermediate nodes contain a pointer to a child if and only if the subtree rooted at the child stores an address of the entity
- The root knows about all entities

Storing information of an entity having two addresses in different leaf domains

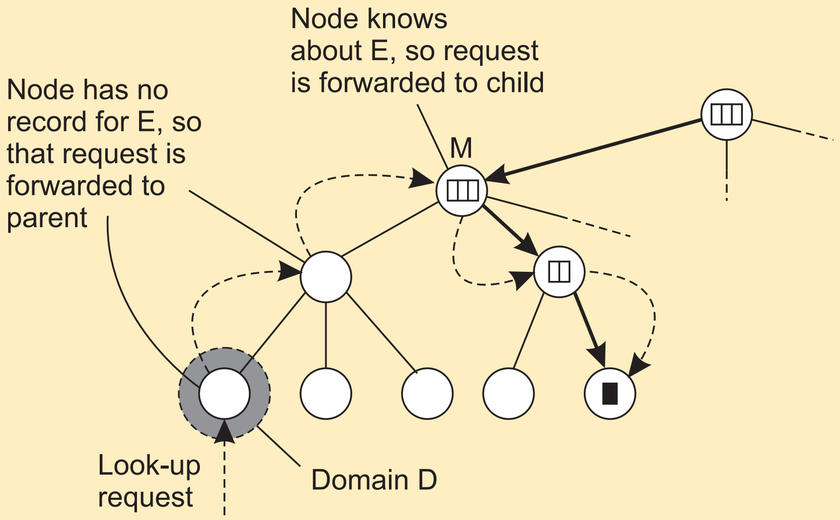


HLS: Lookup operation

Basic principles

- Start lookup at local leaf node
- Node knows about $E \Rightarrow$ follow downward pointer, else go up
- Upward lookup always stops at root

Looking up a location

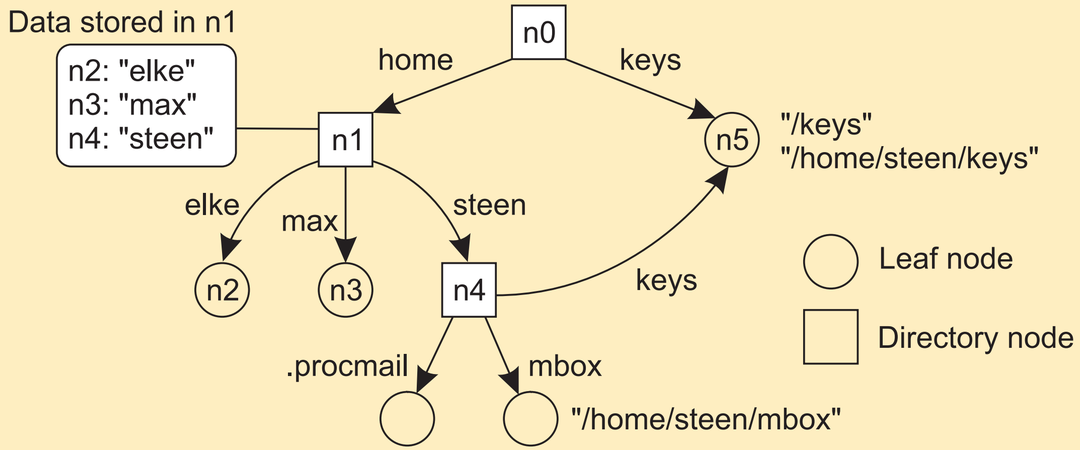


Name space

Naming graph

A graph in which a leaf node represents a (named) entity. A directory node is an entity that refers to other nodes.

A general naming graph with a single root node

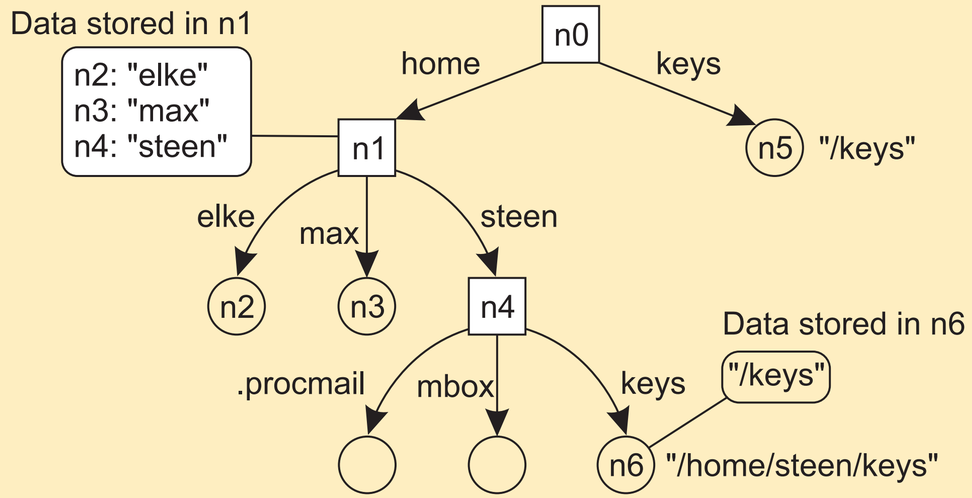


Note

A directory node contains a table of *(node identifier, edge label)* pairs.

Name linking

The concept of a symbolic link explained in a naming graph



Observation

Node *n5* has only one name

Name-space implementation

Basic issue

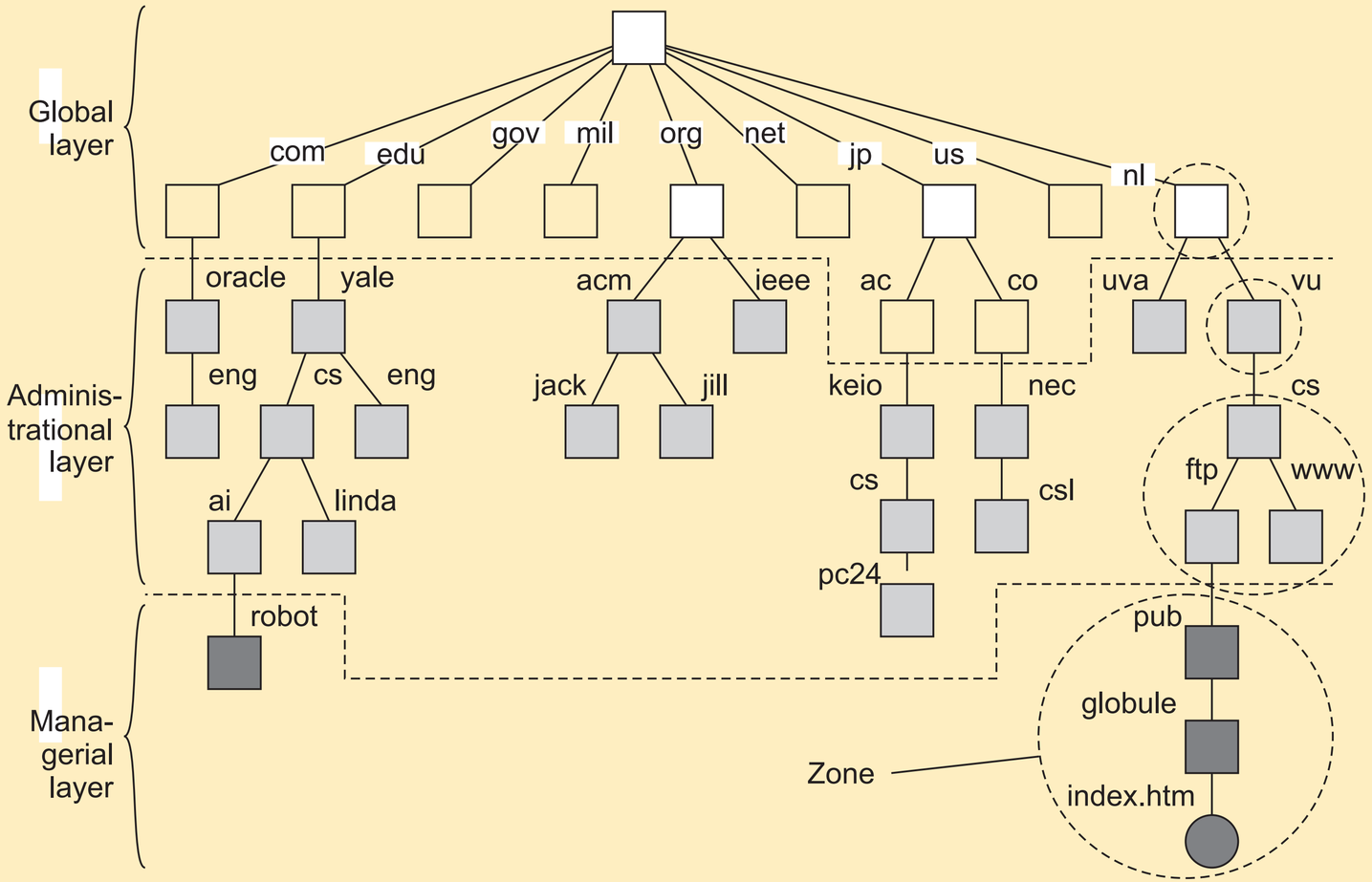
Distribute the name resolution process as well as name space management across multiple machines, by distributing nodes of the naming graph.

Distinguish three levels

- Global level: Consists of the high-level directory nodes. Main aspect is that these directory nodes have to be jointly managed by different administrations
- Administrational level: Contains mid-level directory nodes that can be grouped in such a way that each group can be assigned to a separate administration.
- Managerial level: Consists of low-level directory nodes within a single administration. Main issue is effectively mapping directory nodes to local name servers.

Name-space implementation

An example partitioning of the DNS name space, including network files



Name-space implementation

A comparison between name servers for implementing nodes in a name space

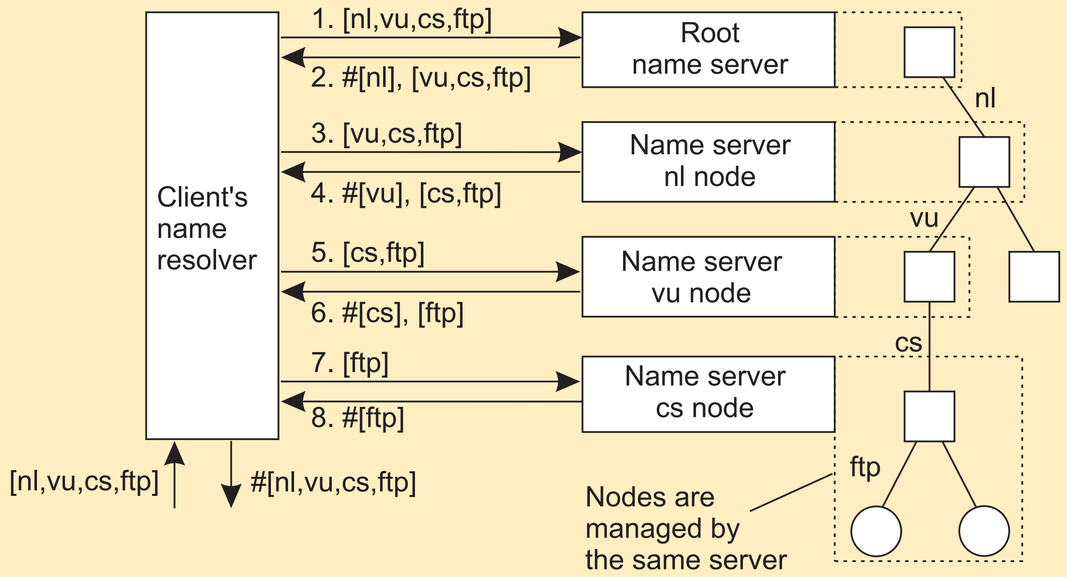
Item	Global	Administrational	Managerial
1	Worldwide	Organization	Department
2	Few	Many	Vast numbers
3	Seconds	Milliseconds	Immediate
4	Lazy	Immediate	Immediate
5	Many	None or few	None
6	Yes	Yes	Sometimes

1: Geographical scale	4: Update propagation
2: # Nodes	5: # Replicas
3: Responsiveness	6: Client-side caching?

Iterative name resolution

Principle

1. *resolve*(*dir*, [$name_1$, ..., $name_K$]) sent to $Server_0$ responsible for *dir*
2. $Server_0$ resolves *resolve*(*dir*, $name_1$) $\rightarrow$ dir_1, returning the identification (address) of $Server_1$, which stores dir_1.
3. Client sends *resolve*(dir_1, [$name_2$, ..., $name_K$]) to $Server_1$, etc.



Recursive name resolution

Principle

1. $resolve(dir, [name_1, ..., name_K])$ sent to $Server_0$ responsible for dir
2. $Server_0$ resolves $resolve(dir, name_1) \rightarrow dir_1$, and sends $resolve(dir_1, [name_2, ..., name_K])$ to $Server_1$, which stores dir_1.
3. $Server_0$ waits for result from $Server_1$, and returns it to client.