Naming

Amir H. Payberah
amir@sics.se

Amirkabir University of Technology
(Tehran Polytechnic)

Based on slides by Maarten Van Steen

Naming

- To operate on an entity, we need to access it at an access point.
- Names are used to denote entities in a distributed system.

Naming

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

Naming

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

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 the following properties:
  - P1: each identifier refers to at most one entity.
  - P2: each entity is referred to by at most one identifier.
  - P3: an identifier always refers to the same entity.

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 the following properties:
  - P1: each identifier refers to at most one entity.
  - P2: each entity is referred to by at most one identifier.
  - P3: an identifier always refers to the same entity.
- Observation: an identifier need not necessarily be a pure name, i.e., it may have content.

Different Class of Naming

- Flat naming
- Structured naming
- Attribute-based naming

Flat Naming

Flat Naming

- Problem: given an unstructured name (e.g., an identifier), how can we locate its associated access point?

Flat Naming

- Problem: given an unstructured name (e.g., an identifier), how can we locate its associated access point?
  - Simple solutions (broadcasting)
  - Home-based approaches
  - Distributed Hash Tables (structured P2P)
  - Hierarchical location service

Simple Solution

Simple Solution (1/2)

- Broadcasting: broadcast the ID, requesting the entity to return its current address.

Simple Solution (1/2)

- 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.

Simple Solution (2/2)

- Forwarding pointers: when an entity moves, it leaves behind a pointer to its next location.

Simple Solution (2/2)

- 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.

Simple Solution (2/2)

- 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.
- Geographical scalability problems:
  - Long chains are not fault tolerant.
  - Increased network latency at dereferencing.

Home-Based Approaches

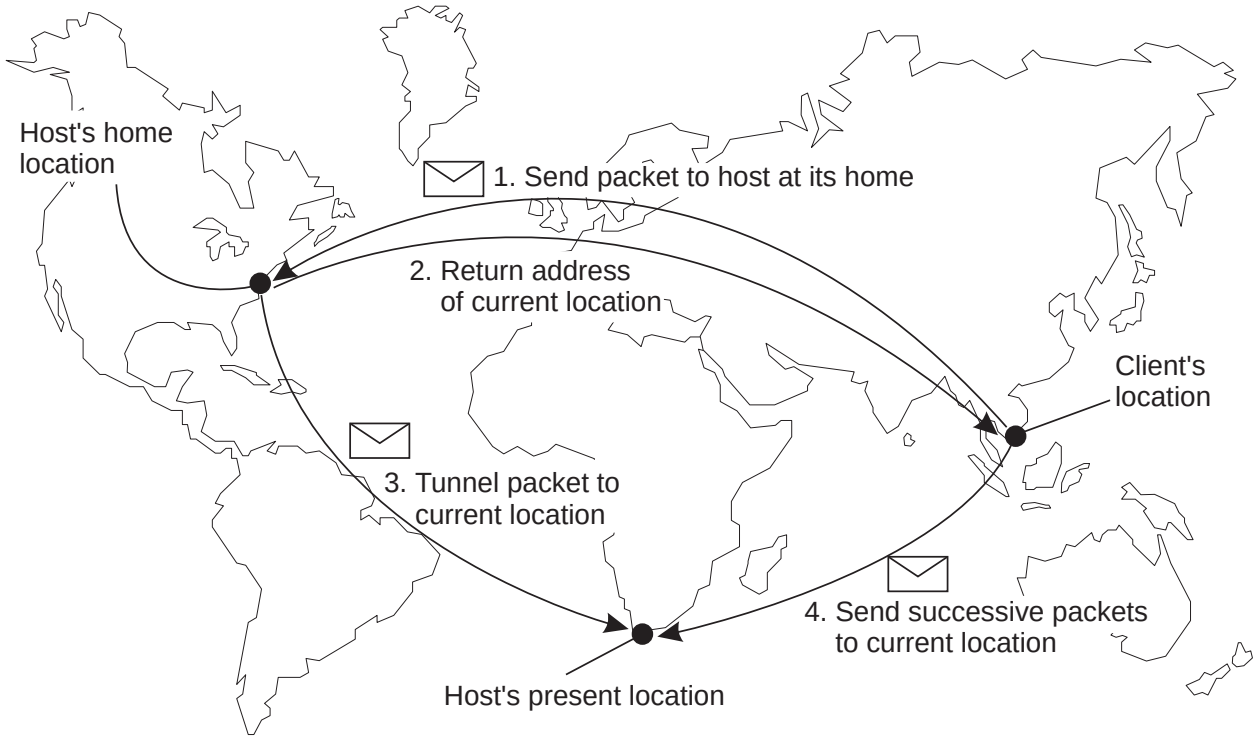
Home-base Approaches (1/3)

- Single-tiered scheme: let a home keep track of where the entity is:

Home-base Approaches (1/3)

- Single-tiered scheme: let a home keep track of where the entity is:
  - Entity's home address registered at a naming service.
  - The home registers the foreign address of the entity.
  - Client contacts the home first, and then continues with foreign location.

Home-base Approaches (2/3)



Home-base Approaches (3/3)

- Two-tiered scheme: keep track of visiting entities:
  - Check local visitor register first.
  - Fall back to home location if local lookup fails.

Home-base Approaches (3/3)

- Two-tiered scheme: keep track of visiting entities:
  - Check local visitor register first.
  - Fall back to home location if local lookup fails.
- Problems with home-based approaches:
  - Home address has to be supported for entity's lifetime.
  - Home address is fixed $\Rightarrow$ unnecessary burden when the entity permanently moves.
  - Poor geographical scalability (entity may be next to client).

Home-base Approaches (3/3)

- Two-tiered scheme: keep track of visiting entities:
  - Check local visitor register first.
  - Fall back to home location if local lookup fails.
- Problems with home-based approaches:
  - Home address has to be supported for entity's lifetime.
  - Home address is fixed $\Rightarrow$ unnecessary burden when the entity permanently moves.
  - Poor geographical scalability (entity may be next to client).
- How can we solve the permanent move problem?
  - Permanent moves may be tackled with another level of naming (DNS).

Distributed Hash Table (DHT)

Distributed Hash Table (1/4)

- Chord: organize the nodes into a logical ring.

Distributed Hash Table (1/4)

- Chord: organize the nodes into a logical ring.
  - Each node is assigned a random m-bit identifier.

Distributed Hash Table (1/4)

- Chord: organize the nodes into a logical ring.
  - Each node is assigned a random m-bit identifier.
  - Every entity is assigned a unique m-bit key.

Distributed Hash Table (1/4)

- Chord: organize the nodes into a logical ring.
  - Each node is assigned a random m-bit identifier.
  - Every entity is assigned a unique m-bit key.
  - Entity with key k falls under jurisdiction of node with smallest $id \geq k$ (called its successor).

Distributed Hash Table (1/4)

- Chord: organize the nodes into a logical ring.
  - Each node is assigned a random m-bit identifier.
  - Every entity is assigned a unique m-bit key.
  - Entity with key k falls under jurisdiction of node with smallest $id \geq k$ (called its successor).
- Let node id keep track of $succ(id)$ and start linear search along the ring.

Distributed Hash Table (2/4)

- Finger table: each node p maintains a finger table $FT_p[]$ with at most m entries: $FT_p[i] = succ(p + 2^{i-1})$.

Distributed Hash Table (2/4)

- Finger table: each node p maintains a finger table $FT_p[]$ with at most m entries: $FT_p[i] = succ(p + 2^{i-1})$.
- $FT_p[i]$ points to the first node succeeding p by at least 2^{i-1}.

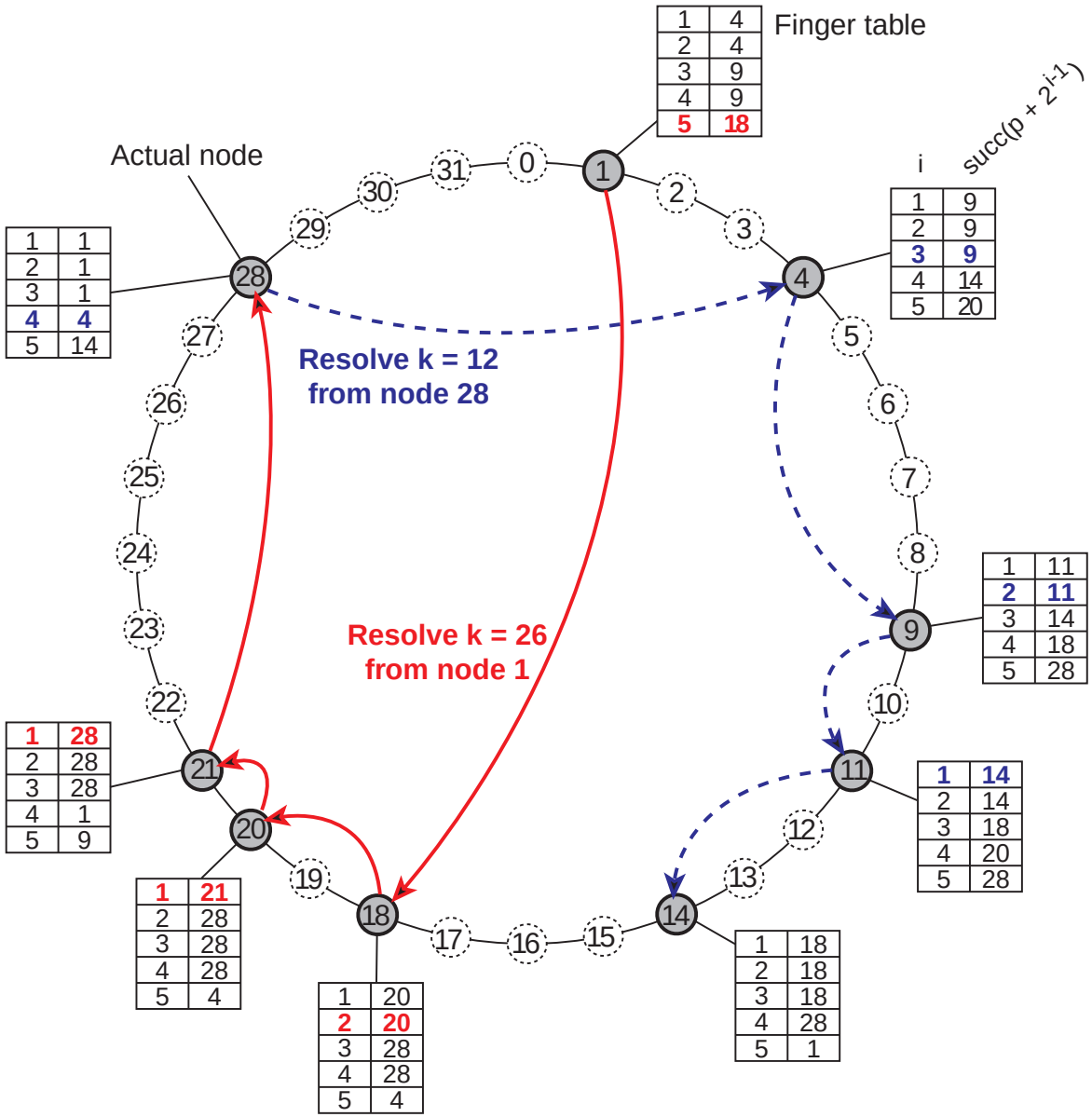
Distributed Hash Table (2/4)

- Finger table: each node p maintains a finger table $FT_p[]$ with at most m entries: $FT_p[i] = succ(p + 2^{i-1})$.
- $FT_p[i]$ points to the first node succeeding p by at least 2^{i-1}.
- To look up a key k, node p forwards the request to node with index j satisfying $q = FT_p[j] \leq k < FT_p[j+1]$.

Distributed Hash Table (2/4)

- Finger table: each node p maintains a finger table $FT_p[]$ with at most m entries: $FT_p[i] = succ(p + 2^{i-1})$.
- $FT_p[i]$ points to the first node succeeding p by at least 2^{i-1}.
- To look up a key k, node p forwards the request to node with index j satisfying $q = FT_p[j] \leq k < FT_p[j+1]$.
- If $p < k < FT_p[1]$, the request is also forwarded to $FT_p[1]$.

Distributed Hash Table (3/4)



Distributed Hash Table (4/4)

- Problem: the logical organization of nodes in the overlay may lead to erratic message transfers in the underlying Internet: node k and node $succ(k+1)$ may be very far apart.

Distributed Hash Table (4/4)

- Problem: the logical organization of nodes in the overlay may lead to erratic message transfers in the underlying Internet: node k and node $succ(k+1)$ may be very far apart.
- Topology-aware node assignment: when assigning an ID to a node, make sure that nodes close in the ID space are also close in the network.

Distributed Hash Table (4/4)

- Problem: the logical organization of nodes in the overlay may lead to erratic message transfers in the underlying Internet: node k and node $succ(k+1)$ may be very far apart.
- Topology-aware node assignment: when assigning an ID to a node, make sure that nodes close in the ID space are also close in the network.
- Proximity routing: maintain more than one possible successor, and forward to the closest.

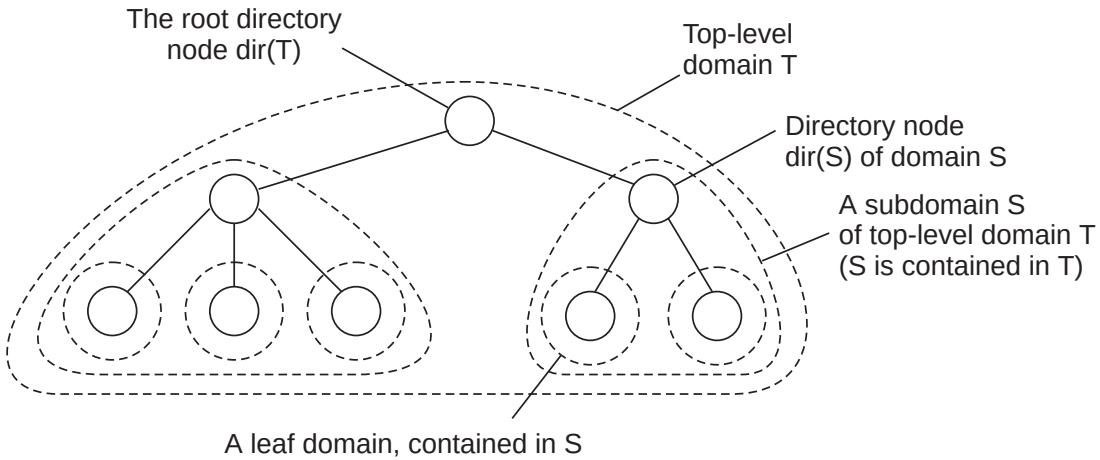
Distributed Hash Table (4/4)

- Problem: the logical organization of nodes in the overlay may lead to erratic message transfers in the underlying Internet: node k and node $succ(k+1)$ may be very far apart.
- Topology-aware node assignment: when assigning an ID to a node, make sure that nodes close in the ID space are also close in the network.
- Proximity routing: maintain more than one possible successor, and forward to the closest.
- Proximity neighbor selection: when there is a choice of selecting who your neighbor will be (not in Chord), pick the closest one.

Hierarchical Location Services (HLS)

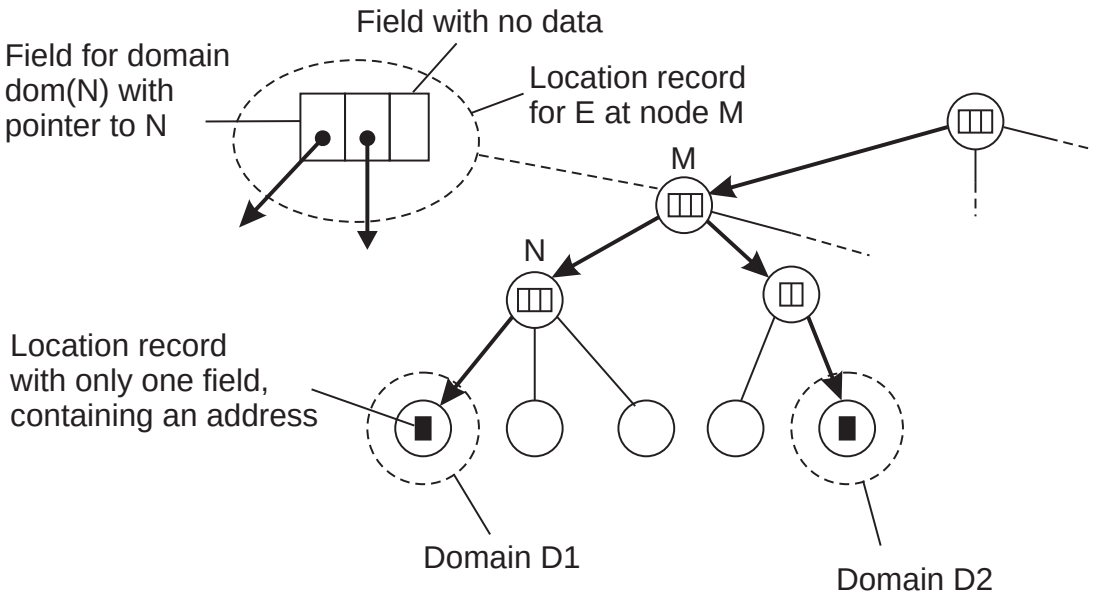
Hierarchical Location Services (1/4)

- 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.



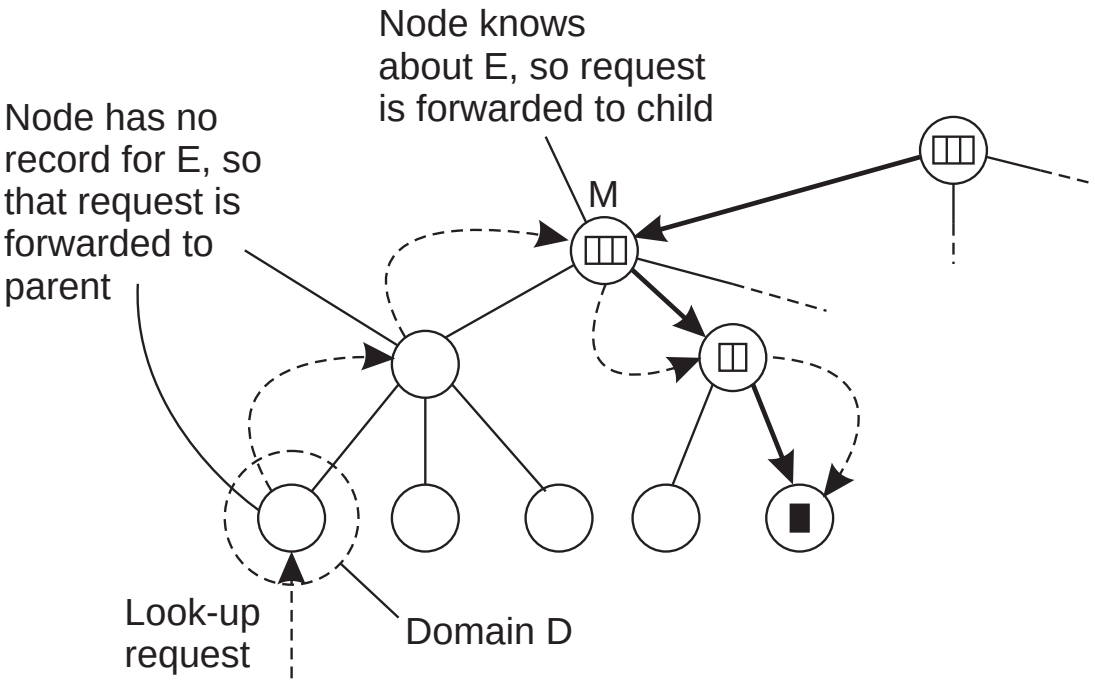
Hierarchical Location Services (2/4)

- Tree organization: address of entity E is stored in a leaf or intermediate node.
- Intermediate nodes contain a pointer to a child iff the subtree rooted at the child stores an address of the entity.
- The root knows about all entities.



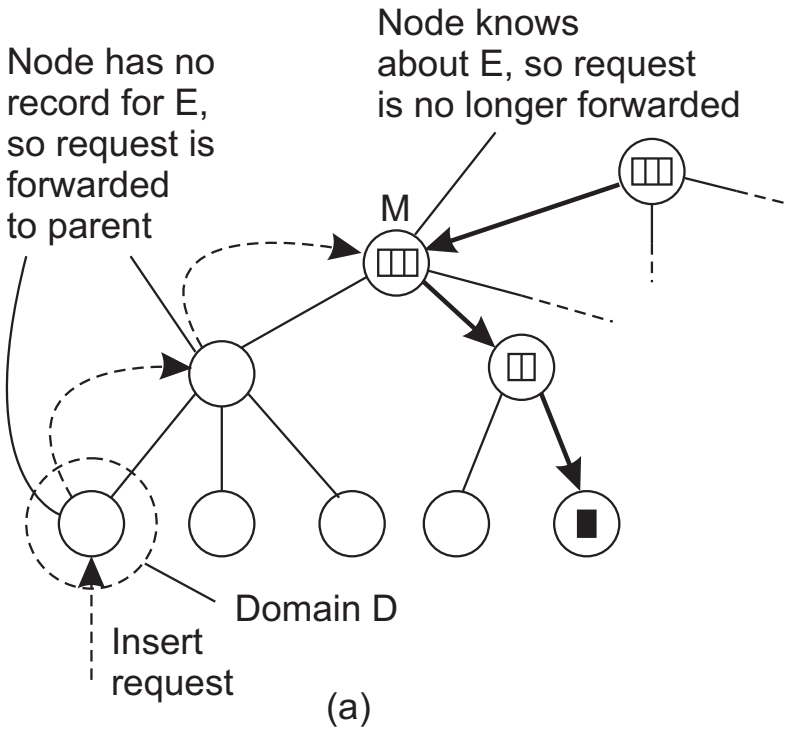
Hierarchical Location Services (3/4)

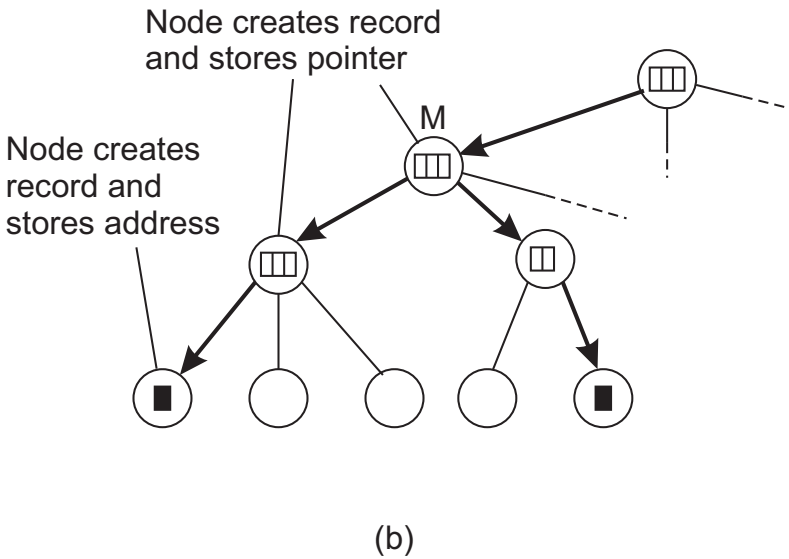
- Lookup operation: start lookup at local leaf node.
- Node knows about $E \Rightarrow$ follow downward pointer, else go up.
- Upward lookup always stops at root.



Hierarchical Location Services (4/4)

- Insert operation

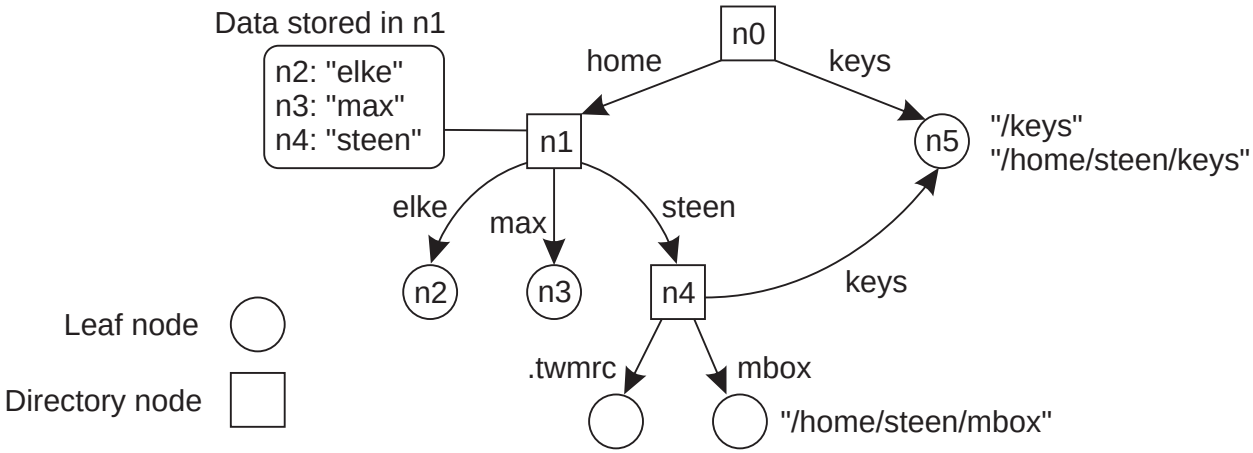




Structured Naming

Structured Naming

- Name space: a graph in which a leaf node represents an entity.
- A directory node is an entity that refers to other nodes.
- A directory node contains a table of `(edge label, node id)` pairs.



Name Space

- Directory nodes can also have attributes, besides just storing a table with `(edge label, node id)` pairs.

- We can easily store all kinds of attributes in a node, describing aspects of the entity the node represents:
  - Type of the entity
  - An identifier for that entity
  - Address of the entity's location
  - Nicknames
  - ...

Name Resolution

- To resolve a name we need a directory node. How do we actually find that (initial) node?

Name Resolution

- To resolve a name we need a directory node. How do we actually find that (initial) node?

- Closure mechanism: the mechanism to select the implicit context from which to start name resolution:
  - `www.cs.vu.nl`: start at a DNS name server
  - `/home/steen/mbox`: start at the local NFS file server
  - `0031204447784`: dial a phone number
  - `130.37.24.8`: route to the VU's Web server

Name Resolution

- To resolve a name we need a directory node. How do we actually find that (initial) node?

- Closure mechanism: the mechanism to select the implicit context from which to start name resolution:
  - `www.cs.vu.nl`: start at a DNS name server
  - `/home/steen/mbox`: start at the local NFS file server
  - `0031204447784`: dial a phone number
  - `130.37.24.8`: route to the VU's Web server

- Name linking: hard link vs. soft link

Name Linking (1/2)

- Hard link: what we have described so far as a path name.
  - A name that is resolved by following a specific path in a naming graph from one node to another.

Name Linking (2/2)

- Soft link: allow a node O to contain a name of another node:

Name Linking (2/2)

- Soft link: allow a node O to contain a name of another node:
  - First resolve O's name (leading to O).
  - Read the content of O, yielding name.
  - Name resolution continues with name.

Name Linking (2/2)

- Soft link: allow a node O to contain a name of another node:
  - First resolve O's name (leading to O).
  - Read the content of O, yielding name.
  - Name resolution continues with name.
- The name resolution process determines that we read the content of a node, in particular, the name in the other node that we need to go to.
- One way or the other, we know where and how to start name resolution given name.

Name Space Implementation (1/5)

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

Name Space Implementation (1/5)

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

Name Space Implementation (1/5)

- Distribute the name resolution process, as well as the name space management across multiple machines, by distributing nodes of the naming graph.
- Distinguish three levels:
  - Global level

Name Space Implementation (1/5)

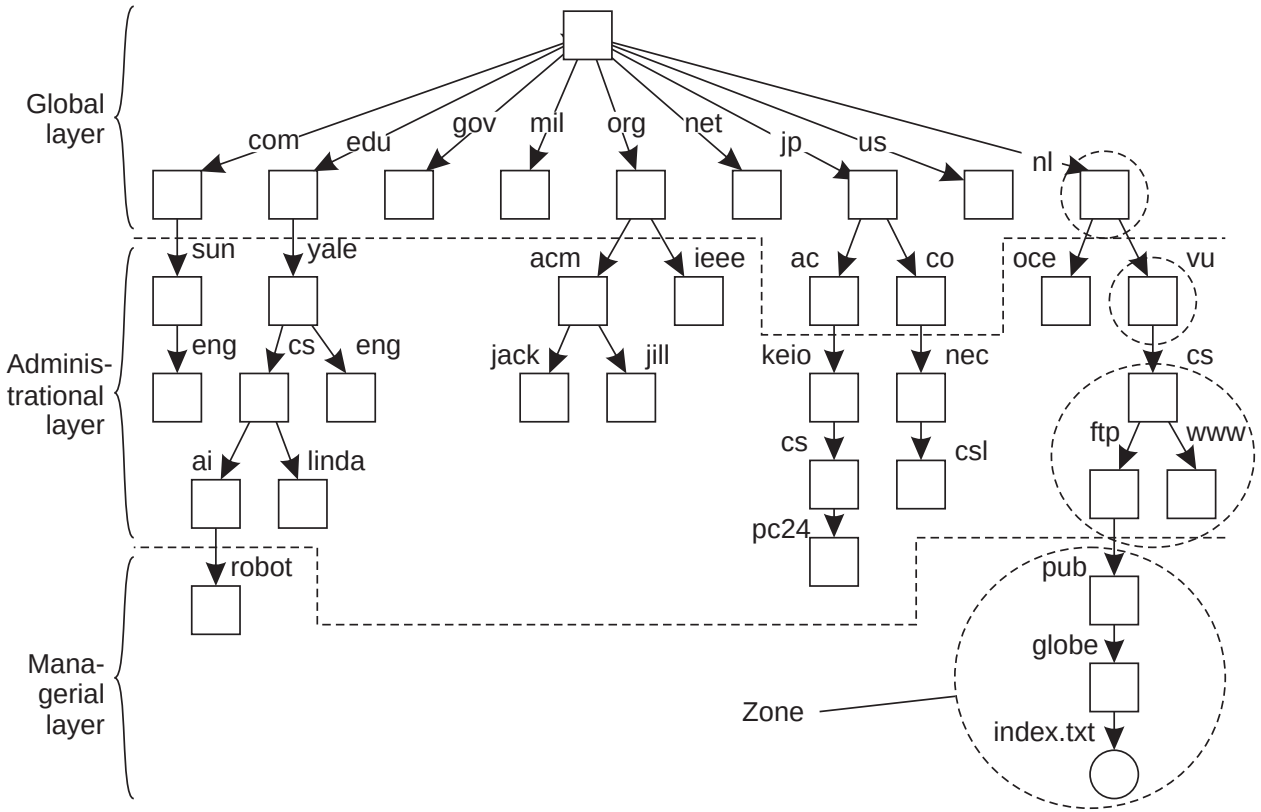
- Distribute the name resolution process, as well as the name space management across multiple machines, by distributing nodes of the naming graph.
- Distinguish three levels:
  - Global level
  - Administrational level

Name Space Implementation (1/5)

- Distribute the name resolution process, as well as the name space management across multiple machines, by distributing nodes of the naming graph.
- Distinguish three levels:
  - Global level
  - Administrational level
  - Managerial level

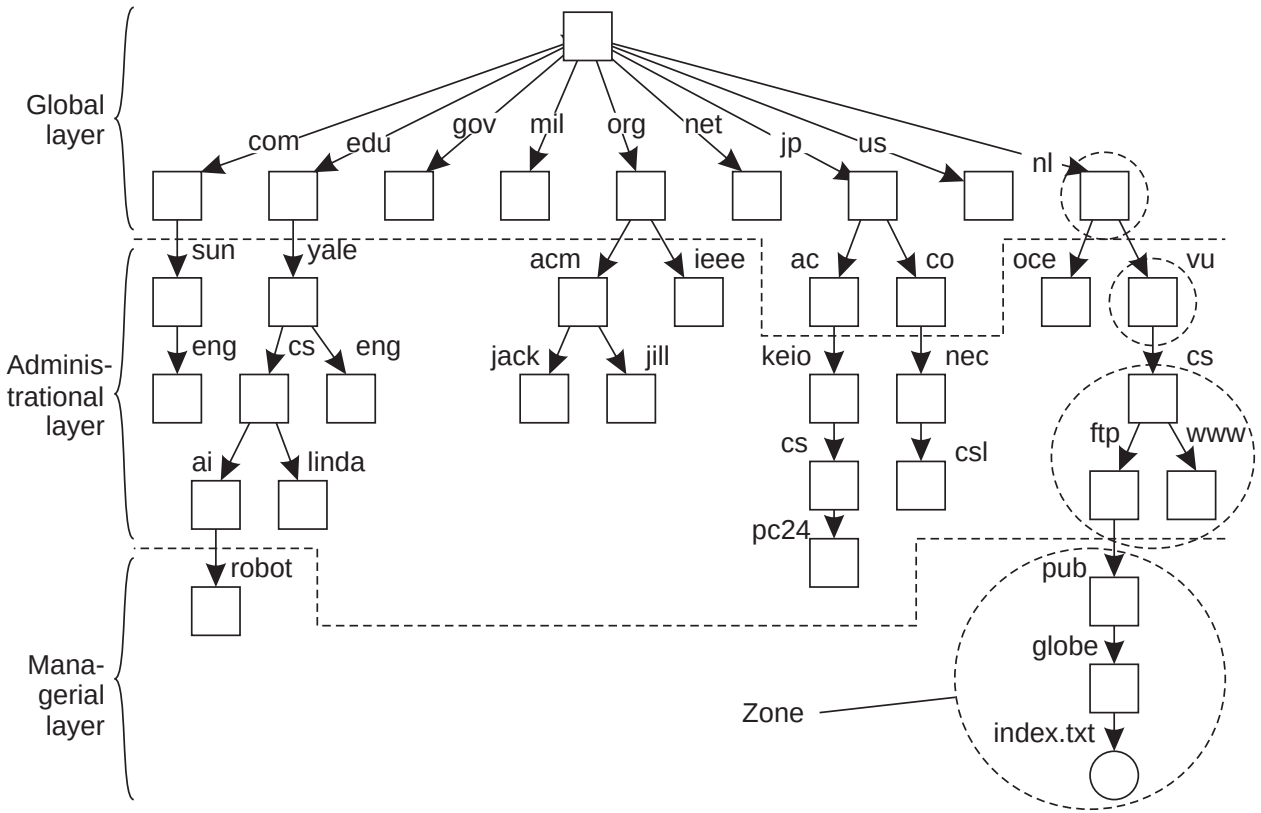
Name Space Implementation (2/5)

- Global level: consists of the high-level directory nodes. Main aspect is that these directory nodes have to be jointly managed by different administrations.



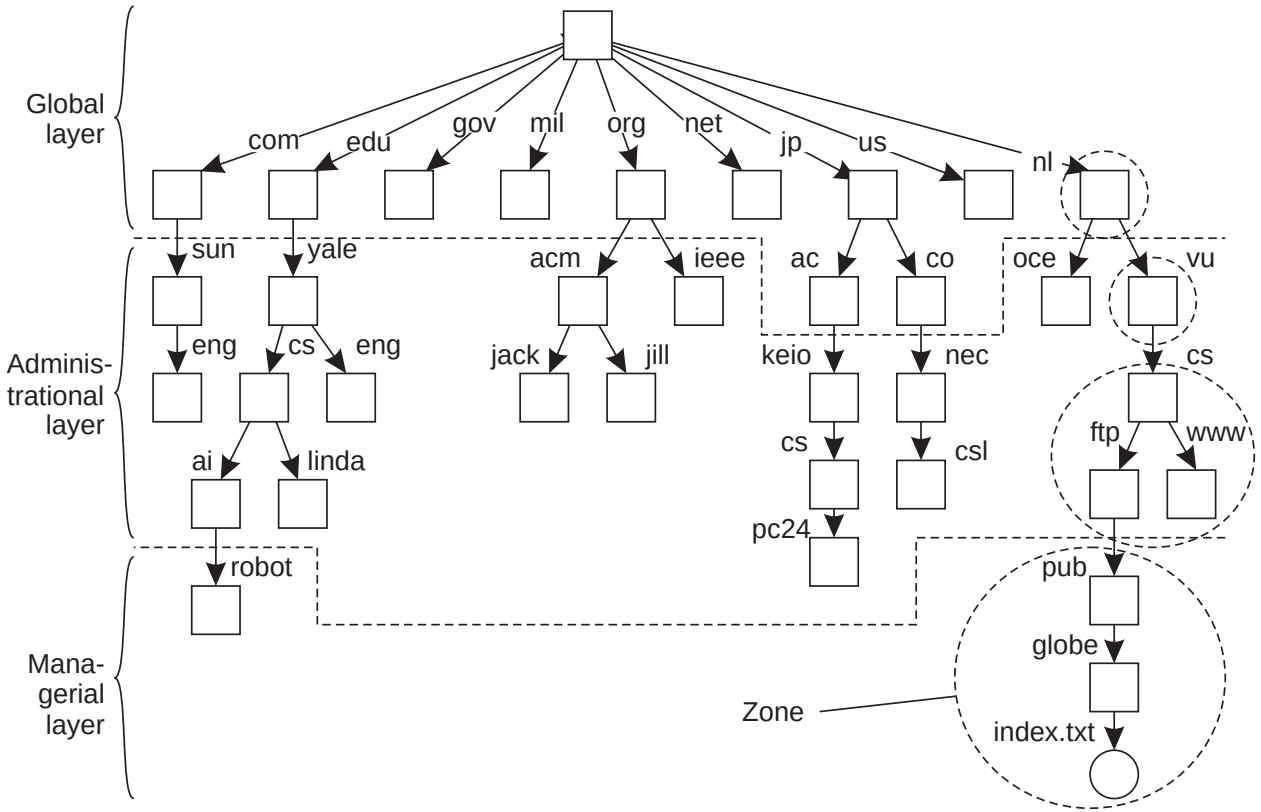
Name Space Implementation (3/5)

- Administrational level: contains mid-level directory nodes that can be grouped in such a way that each group can be assigned to a separate administrational.



Name Space Implementation (4/5)

- Managerial level: consists of low-level directory nodes within a single administrational. Main issue is effectively mapping directory nodes to local name servers.



Name Space Implementation (5/5)

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

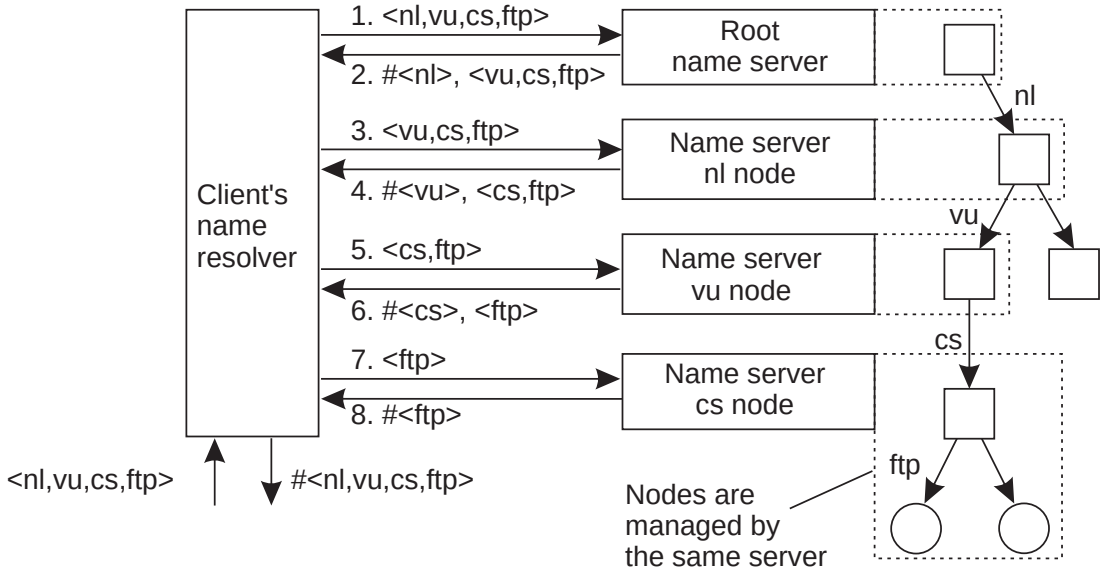
1: Geographical scale	4: Update propagation
2: # Nodes	5: # Replicas
3: Responsiveness	6: Client-side caching?

Name Resolution

- Iterative
- Recursive

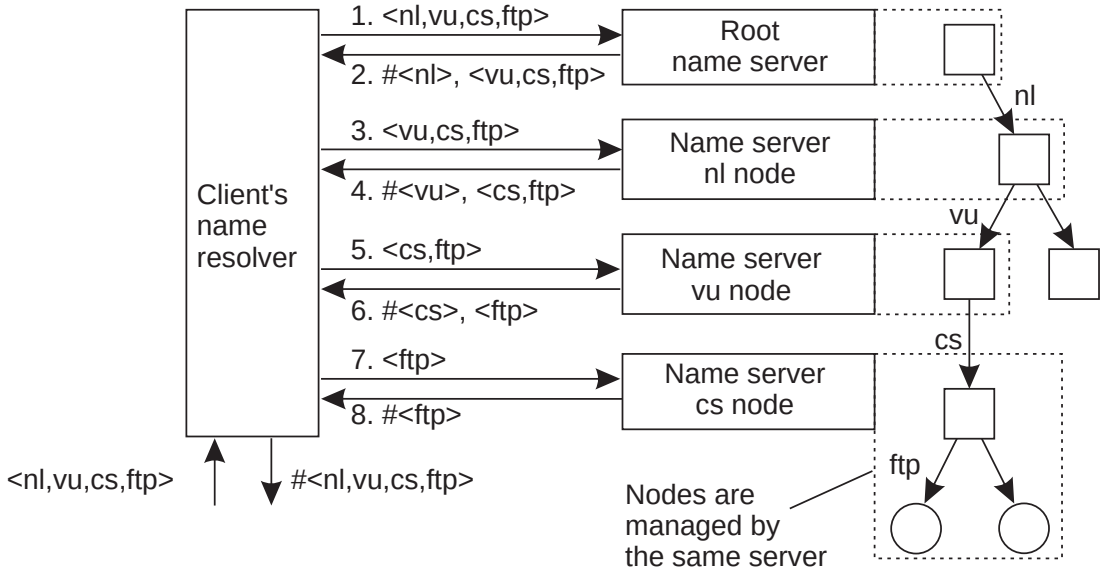
Iterative Name Resolution

1. `resolve(dir, [name1, ..., nameK])` sent to `Server0` responsible for `dir`.



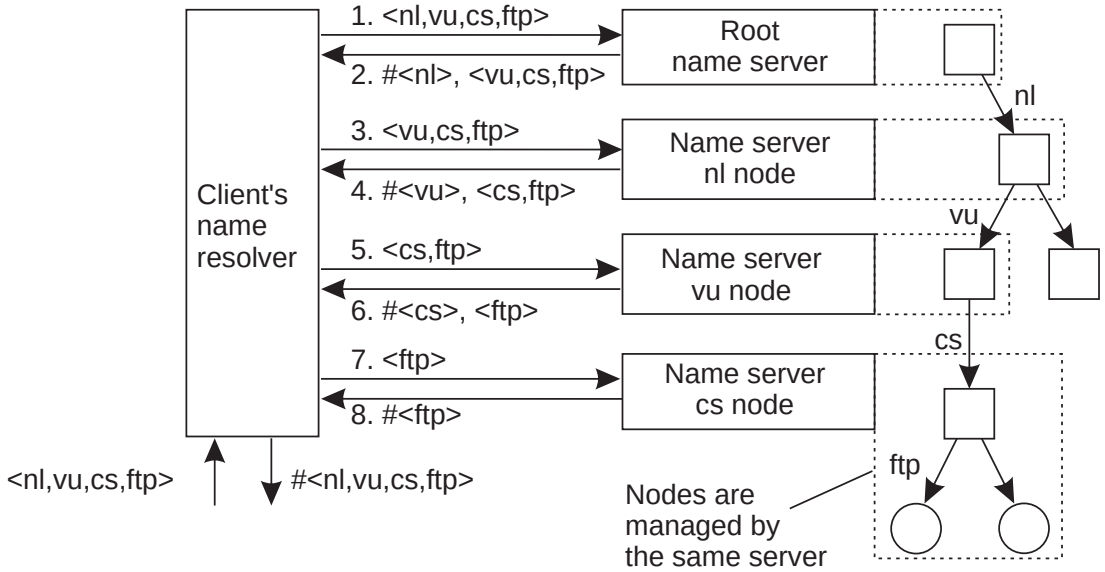
Iterative Name Resolution

1. `resolve(dir, [name1, ..., nameK])` sent to `Server0` responsible for `dir`.
2. `Server0` resolves `resolve(dir, name1)` $\rightarrow$ `dir1`, returning the identification (address) of `Server1`, which stores `dir1`.



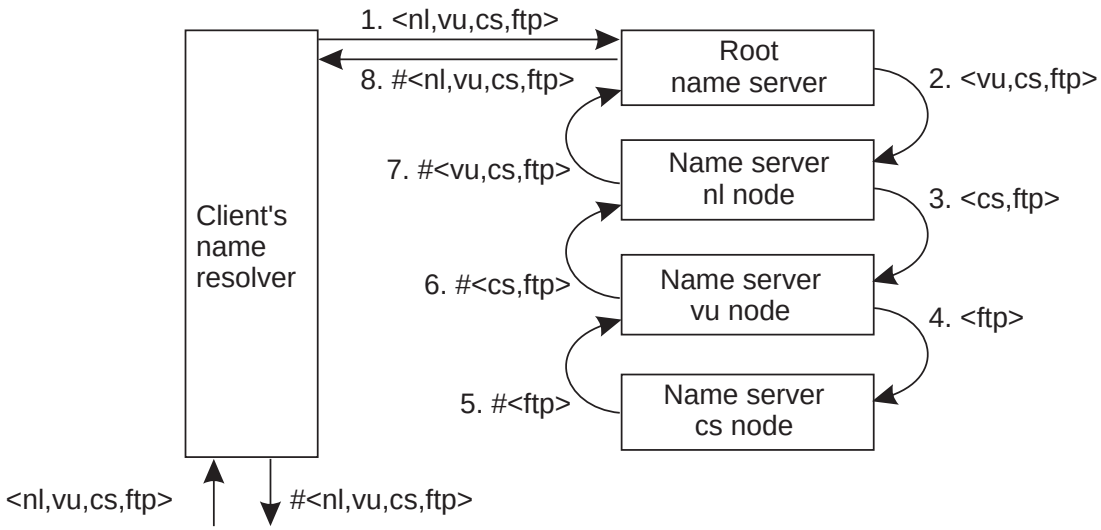
Iterative Name Resolution

1. `resolve(dir, [name1, ..., nameK])` sent to `Server0` responsible for `dir`.
2. `Server0` resolves `resolve(dir, name1)` $\rightarrow$ `dir1`, returning the identification (address) of `Server1`, which stores `dir1`.
3. Client sends `resolve(dir1, [name2, ..., nameK])` to `Server1`, etc.



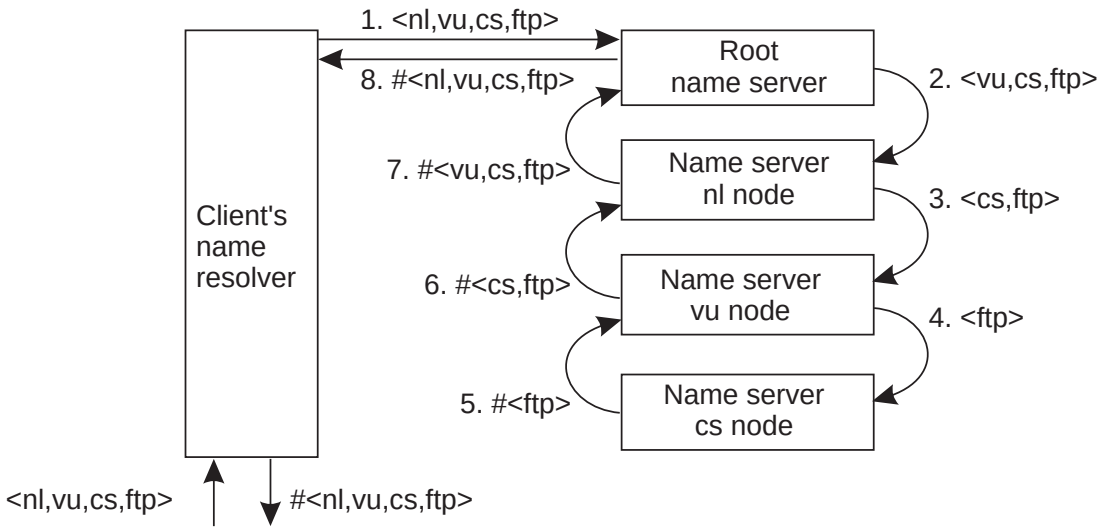
Recursive Name Resolution

1. `resolve(dir, [name1, ..., nameK])` sent to `Server0` responsible for `dir`.



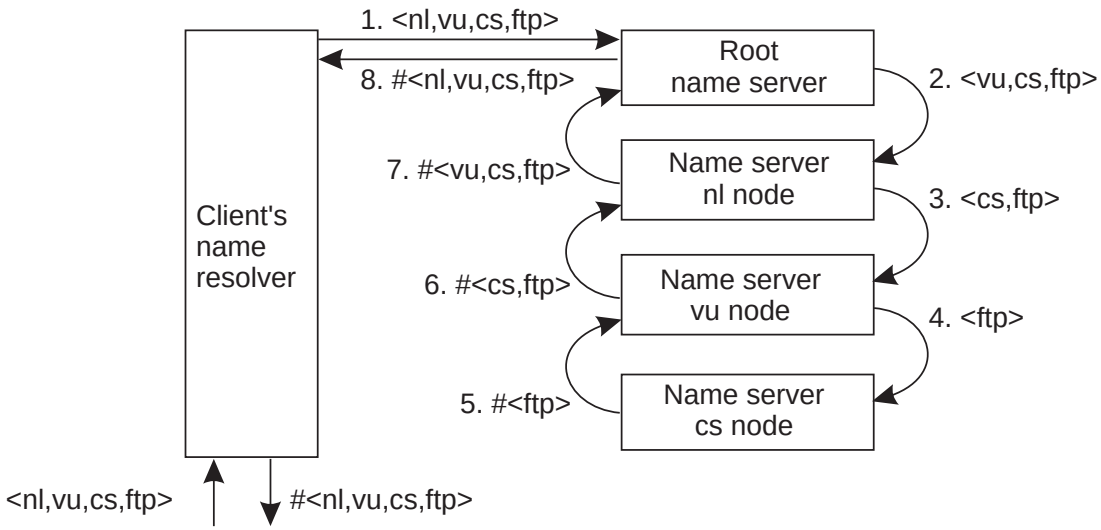
Recursive Name Resolution

1. `resolve(dir, [name1, ..., nameK])` sent to `Server0` responsible for `dir`.
2. `Server0` resolves `resolve(dir, name1)` $\rightarrow$ `dir1`, and sends `resolve(dir1, [name2, ..., nameK])` to `Server1`, which stores `dir1`.



Recursive Name Resolution

1. `resolve(dir, [name1, ..., nameK])` sent to `Server0` responsible for `dir`.
2. `Server0` resolves `resolve(dir, name1)` $\rightarrow$ `dir1`, and sends `resolve(dir1, [name2, ..., nameK])` to `Server1`, which stores `dir1`.
3. `Server0` waits for result from `Server1`, and returns it to client.



Scalability issues

- Size scalability: we need to ensure that servers can handle a large number of requests per time unit $\Rightarrow$ high-level servers are in big trouble.

Scalability issues

- Size scalability: we need to ensure that servers can handle a large number of requests per time unit $\Rightarrow$ high-level servers are in big trouble.

- Solution:
  - Assume (at least at global and administration level) that content of nodes hardly ever changes.
  - We can then apply extensive replication by mapping nodes to multiple servers, and start name resolution at the nearest server.

Attribute-based Naming

Attribute-based Naming

- In many cases, it is much more convenient to name, and look up entities by means of their attributes $\Rightarrow$ traditional directory services.

Attribute-based Naming

- In many cases, it is much more convenient to name, and look up entities by means of their attributes $\Rightarrow$ traditional directory services.

- Problem: lookup operations can be extremely expensive, as they require to match requested attribute values, against actual attribute values $\Rightarrow$ inspect all entities (in principle).

Attribute-based Naming

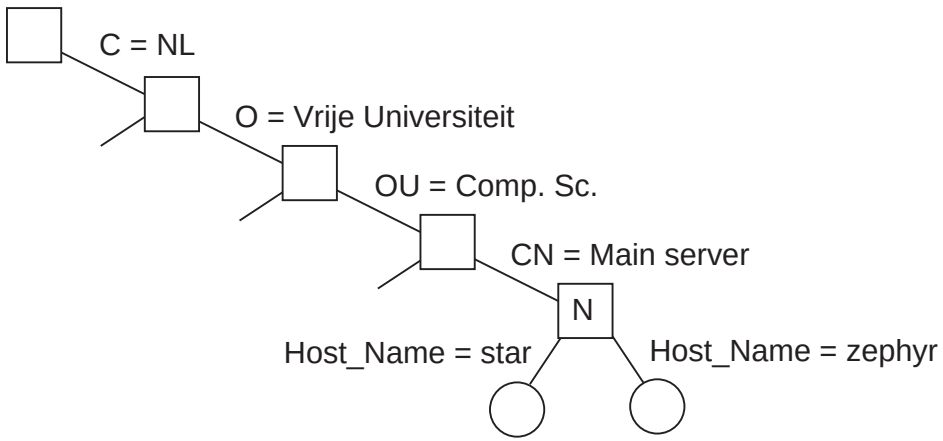
- In many cases, it is much more convenient to name, and look up entities by means of their attributes $\Rightarrow$ traditional directory services.
- Problem: lookup operations can be extremely expensive, as they require to match requested attribute values, against actual attribute values $\Rightarrow$ inspect all entities (in principle).
- Solution: implement basic directory service as database, and combine with traditional structured naming system.

Example: LDAP

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

- answer = search("&(C = NL) (O = Vrije Universiteit) (OU = *) (CN = Main server)")



Summary

Summary

- Naming
- Class of naming: flat, structured, attribute-based
- Flat naming: broadcast, home-based, DHT, hierarchical
- Structured naming: global level, administration level, managerial level
- Attribute-based: LDAP

Reading

- Chapter 5 of the Distributed Systems: Principles and Paradigms.

Questions?