Introduction Distributed Systems - Naming

Today

- Names, identifiers and addresses
- Name resolution

Names, identifiers and addresses

- Names are used to denote entities in a distributed system
  – Hosts, printers, files, processes, users ….
- To operate on an entity, e.g. print a file, we need to access it at an access point
  – An entity can offer more than one access points (think of telephone numbers)
- Access points are entities that are named by means of an address (telephone numbers)
- A location-independent name for an entity *E, is independent* from the addresses of the access points offered by *E*

Name, identifiers and addresses

- Identifier – a name having the following properties
  - Each 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 (no reusing)
- Human-friendly names – unlike identifiers and addresses, normally a character string
- Now, here's the question:
  How do we resolve names & identifiers to addresses?
  - Naming system

Flat naming

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

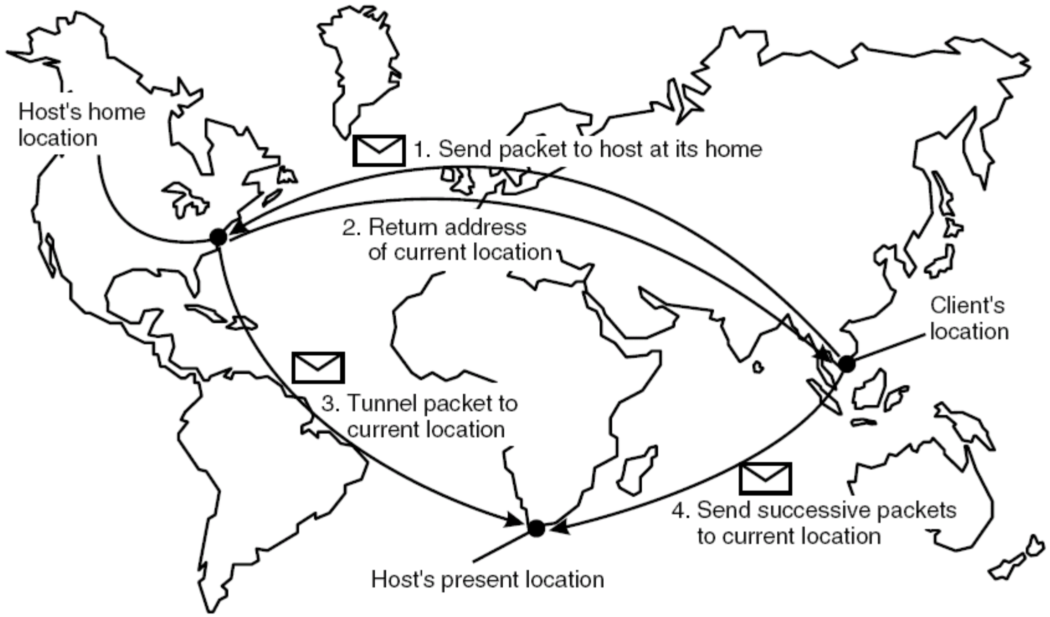
Simple solutions

- Broadcasting – simply 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
- Forwarding pointers – each time an entity moves, it leaves behind a pointer telling where it has gone to.
  - Dereferencing can be made entirely transparent to clients by simply following the chain of pointers
  - Update a client’s reference as soon as present location has been found
  - Geographical scalability problems:
    - Long chains are not fault tolerant
    - Increased network latency at dereferencing

Essential to have separate chain reduction mechanisms

Home-based approaches

- Another approach to support mobile entities – let a home keep track of where the entity is:
  - An entity's home address is registered at a naming service
  - The home registers the foreign address of the entity
  - Clients always contact the home first, and then continues with the foreign location



Home-based approaches

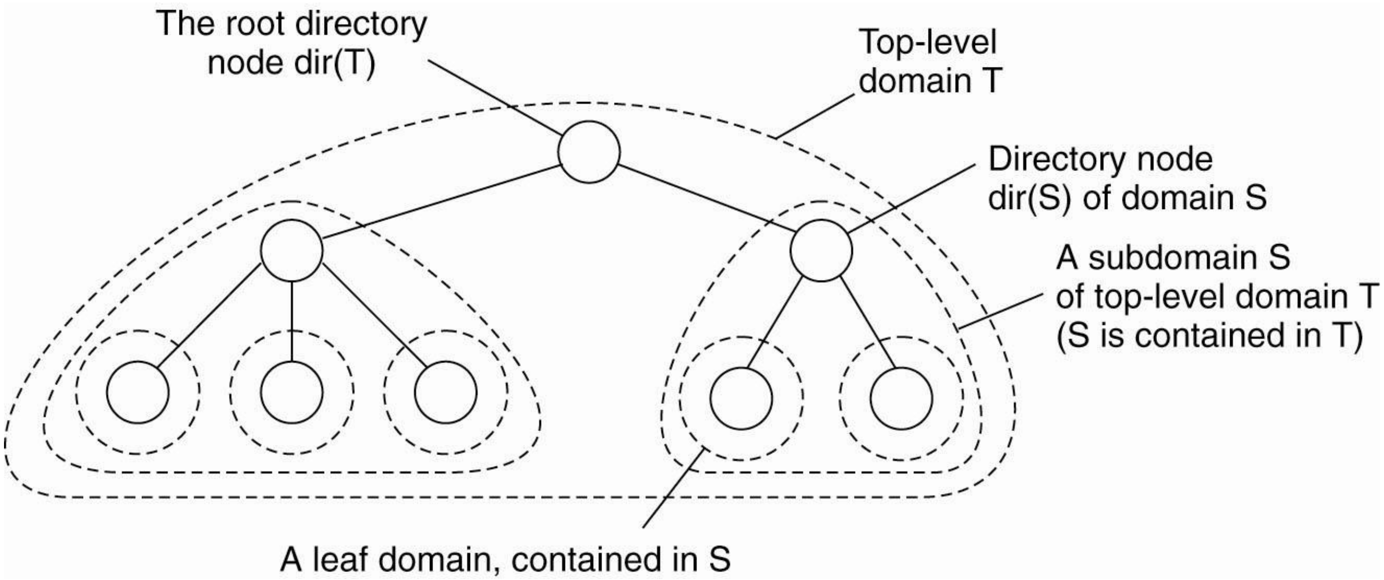
- Problems with home-based approaches
  - Home address has to be supported as long as the entity lives
  - Home address is fixed, which means an unnecessary burden when the entity permanently moves to another location
  - Poor geographical scalability (entity may be next to the client)

Distributed Hash Tables (DHT)

- Consider the organization of nodes into a logical ring (Chord)
  - 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 ≥ k (called its successor)*
- Non-solution: Let node *id* keep track of *succ(id)* (and pred) and do a linear search along the ring
- DHTs – alternative ways to find shortcuts

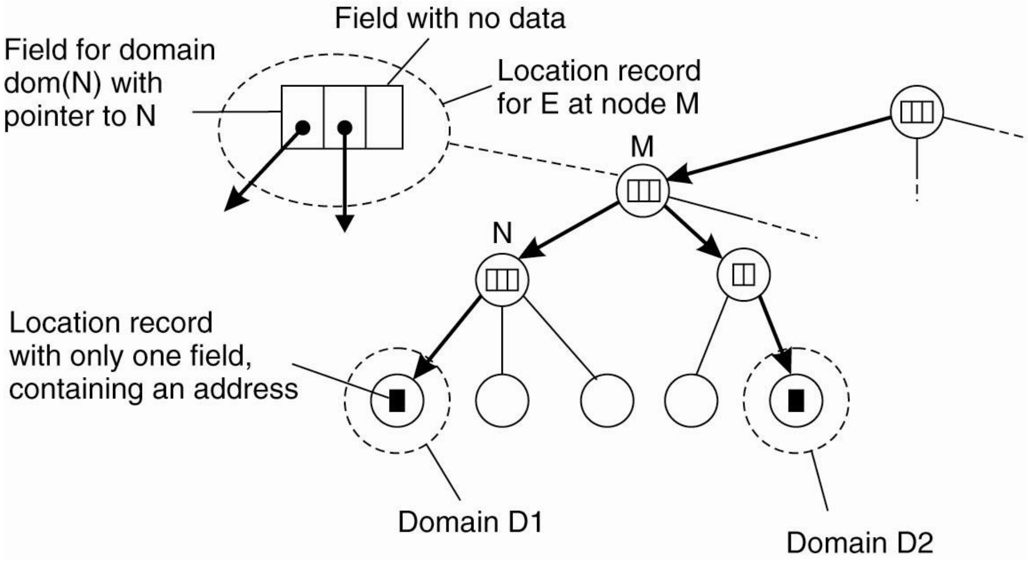
Hierarchical location system

- 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.



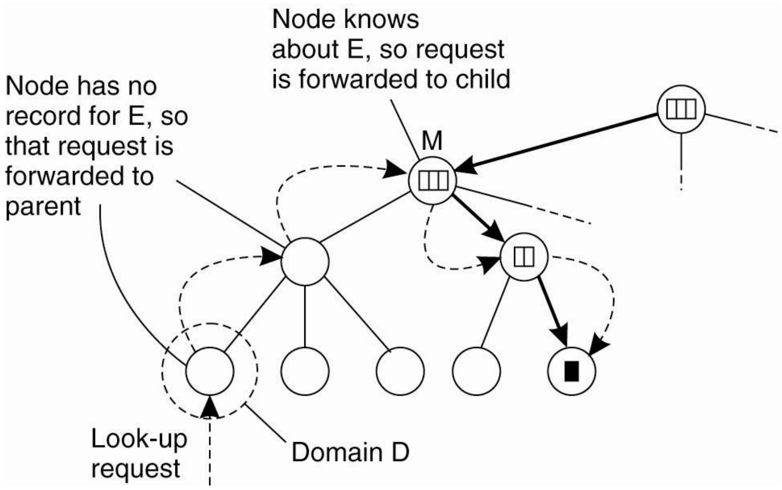
HLS – Tree organization

- The address of an entity is stored in a leaf node, or in an 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

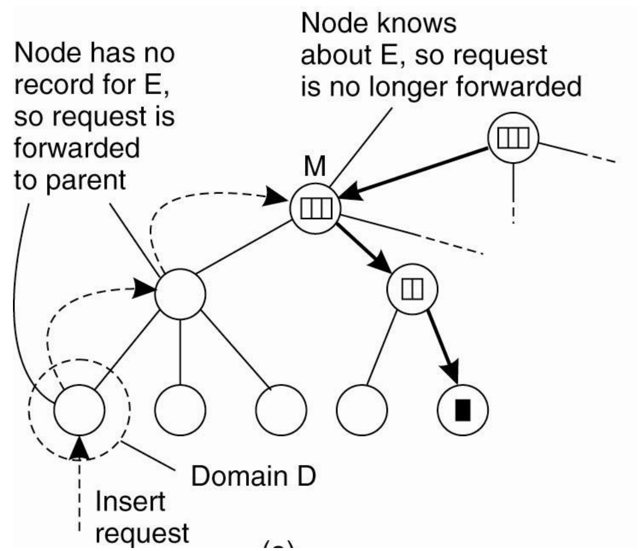


HLS lookups and inserts

- Start lookup at local leaf node
- If node knows it, follow downward
pointer, otherwise go one up
- Upward lookup always stops at root

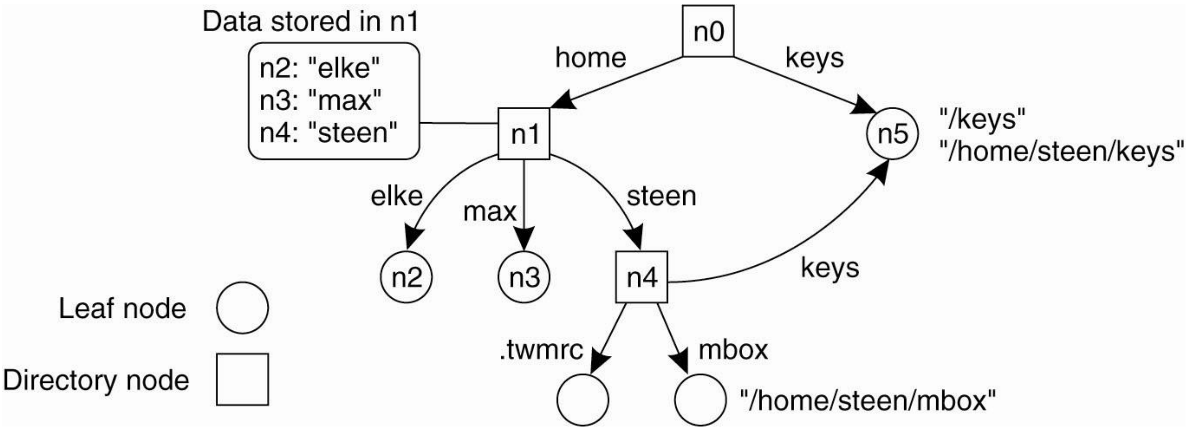


- Insertion of a replica for E initiated in leaf domain D
- This forwards to parent, … until it reaches directory node M
- Request is push down with each node creating a location record



Name space

- A graph in which a leaf node represents a (named) entity. A directory node is an entity that refers to other nodes
- A directory node contains a (directory) table of *(edge label, node identifier) pairs.*
- We can easily store all kinds of attributes in a node, describing aspects of the entity the node represents:

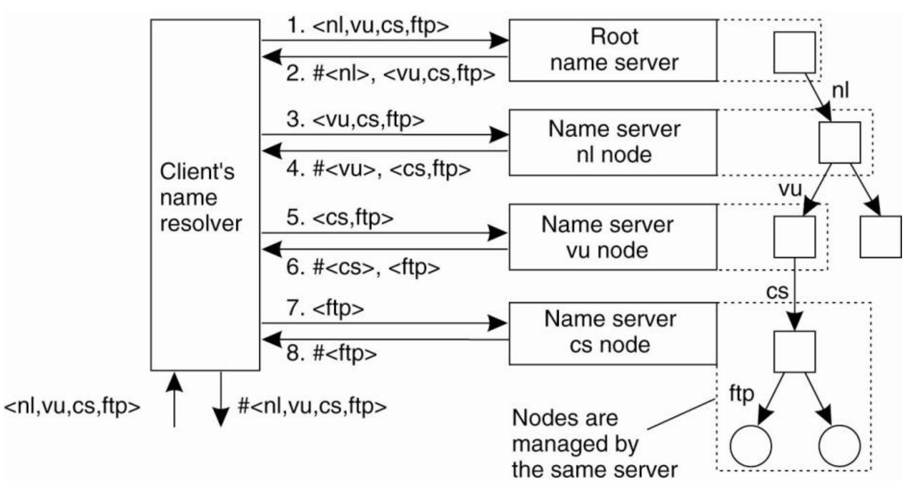


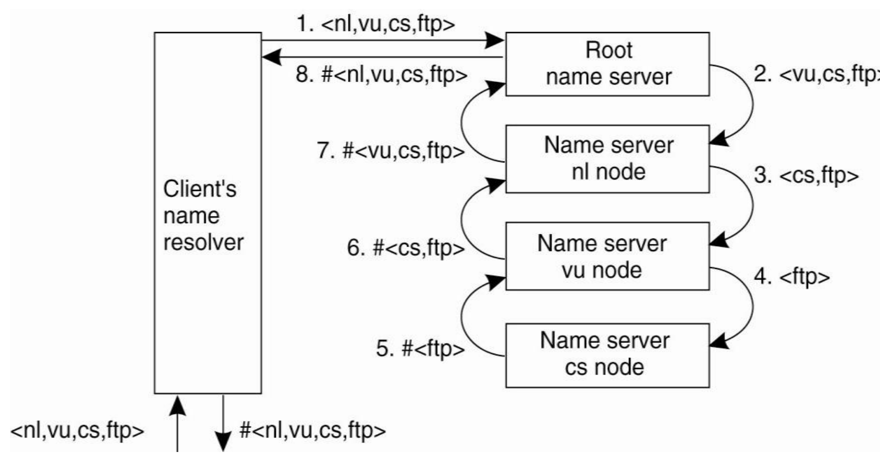
Name space implementation

- Basic issue – distribute name resolution process and name space management across multiple machines, by distributing nodes of the naming graph
- Consider a hierarchical naming graph, three key levels
  – Global level – high-level directory nodes; jointly managed by different administrations
  – Administrational level – mid-level directory nodes grouped so that each group can be assigned to a separate administration
  – Managerial level – low-level directory nodes within a single administration; main issue is effectively mapping directory nodes to local name servers
- At high levels, content of nodes hardly ever changes – leverage replication & start name resolution at nearest server

Interactive and recursive resolution

- Interactive – client drives the resolution
  – Caching by clients
  – Potentially costly communication
- Recursive – the server does
  – Higher performance demand on servers
  – More effective caching
  – Reduced communication costs





Attribute-based naming

- In many cases, it is much more convenient to name, and look up entities by means of their attributes
- Lookup operations can be extremely expensive, as they require to match requested attribute values, against actual attribute values
- Solutions:
  - Implement basic directory service as database, and combine with traditional structured naming system – LDAP
  - Entities' descriptions are translated into attribute-value trees which are encoded into a set of unique hash ids for a DHT – INS/Twine, SWORD, Mercury

Question 1

- *How and were do you start name resolution? How do you select the initial node in a name space?*