Distributed Systems
ECEG-6504

Naming

Surafel Lemma Abebe (Ph. D.)

Topics

- Names, Addresses, Name Resolution
- Internet DNS

Names, Addresses, Name Resolution

- Name
  - Used to denote entities in a DS
  - Examples of entities
    - Hosts, printers, disks, files, users, …
  - To operate on an entity, we need to access it at an access point
  - Access point
    - Is a special kind of entity named by means of an address
      - E.g., a host running a specific server, with its address formed by the combination of IP address and port number
    - Address is a special kind of name
    - An entity could have many access points
    - An entity may change its access points
  - Address is almost never used as a regular name for an entity

Names, Addresses, Name Resolution…

- Name…
  - Reasons as to why addresses are not good to use as a name
    - The instant the access point changes or is reassigned to another entity, address would be invalid reference
    - An entity could have more than one access point, making difficult which address should be chosen as the best one

  => A name for an entity that is independent from address of its access point is much easier and flexible to use
    - Location independent name

  - Identifiers
    - Are random bit strings which are referred to as unstructured, or flat names that uniquely identify an entity
    - Properties of a true identifier that uniquely identifies an entity
      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., its never reused)

Names, Addresses, Name Resolution…

- Name…
  - Human-friendly names
    - Name that is tailored to be used by humans
    - In contrast to addresses and identifiers, a human-friendly name is generally represented as a character string
  - How do we resolve names and identifiers to addresses?

Names, Addresses, Name Resolution…

- Flat name
  - Unstructured name such as identifier does not contain any information on how to locate an access point of its associated entity
  - Problem
    - How can we locate the associated access point of an unstructured name?
  - Solutions
    - Simple solutions (Broadcasting and forwarding pointers)
    - Home based approaches
    - Distributed hash tables
    - Hierarchical location services

Names, Addresses, Name Resolution…

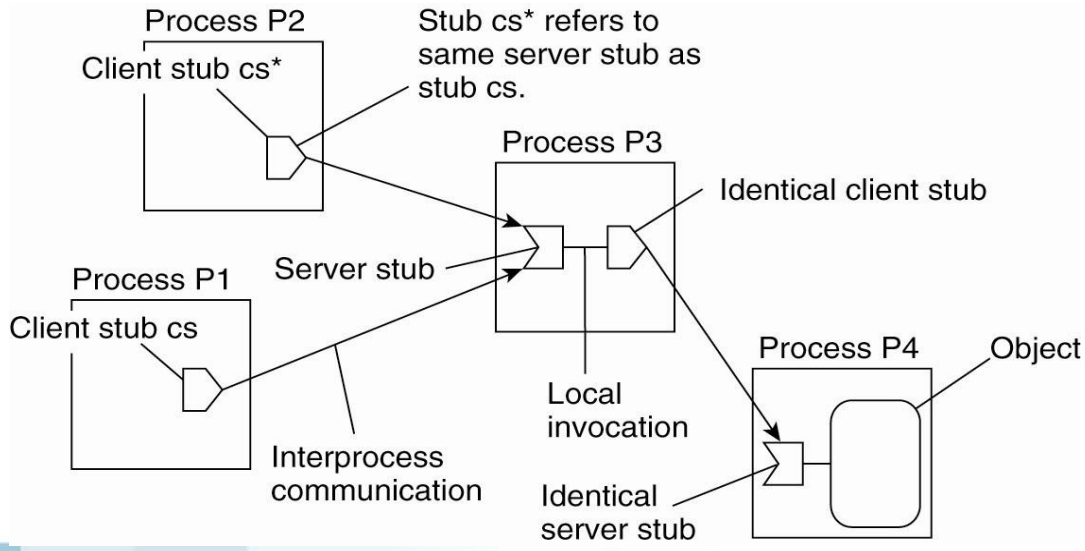
- Flat name…
  - Broadcasting and multicasting
    - Broadcasting
      - Simple but applicable to LAN
      - Locating an entity
        - » Broadcast a message containing the identifier of the entity, requesting the entity to return its current address
      - Principle is used in different protocols
        - » E.g., Address resolution protocol (ARP)
      - Limitation
        - » Becomes inefficient when the network grows (wastes bandwidth)
        - » Requires all processes to listen to incoming location requests
          - Hosts are interrupted by requests they cannot answer
      - Possible solution
        - » Multicast

Names, Addresses, Name Resolution…

- Flat name…
  - Forward pointers
    - When a (mobile) 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

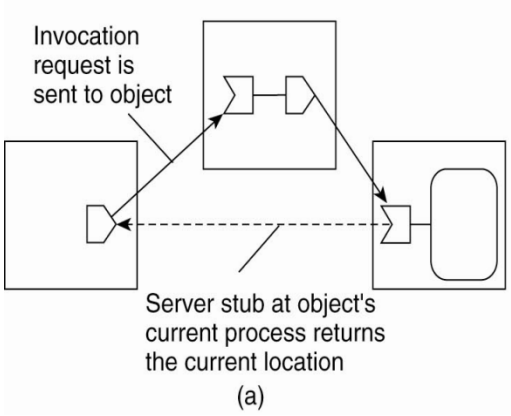
Names, Addresses, Name Resolution…

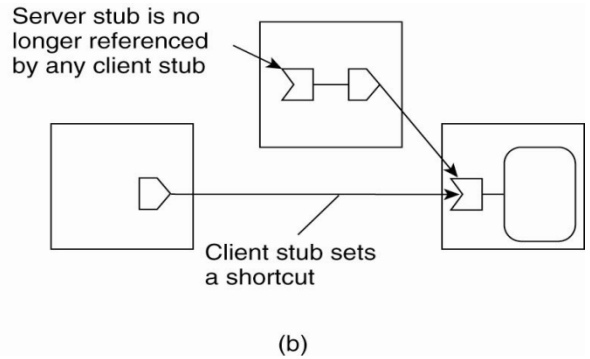
- Flat name…
  - Forwarding pointers…
    - Example of use with respect of remote objects
      - Forward pointer is implemented as (client stub, server stub) pair
      - Whenever an object moves from address space A to B, it leaves behind a client stub in its place in A and installs a server stub that refers to it in B.



Names, Addresses, Name Resolution…

- Flat name…
  - Forwarding pointers…
    - Short-cut a chain of (client stub, server stub) pairs
      - When the invocation reaches the object at its current location, a response is sent back to the client stub where the invocation was initiated (often without going back up the chain)
      - Trade-off between sending the response directly to the initiating client stub vs along the reverse path of forwarding pointers





Names, Addresses, Name Resolution…

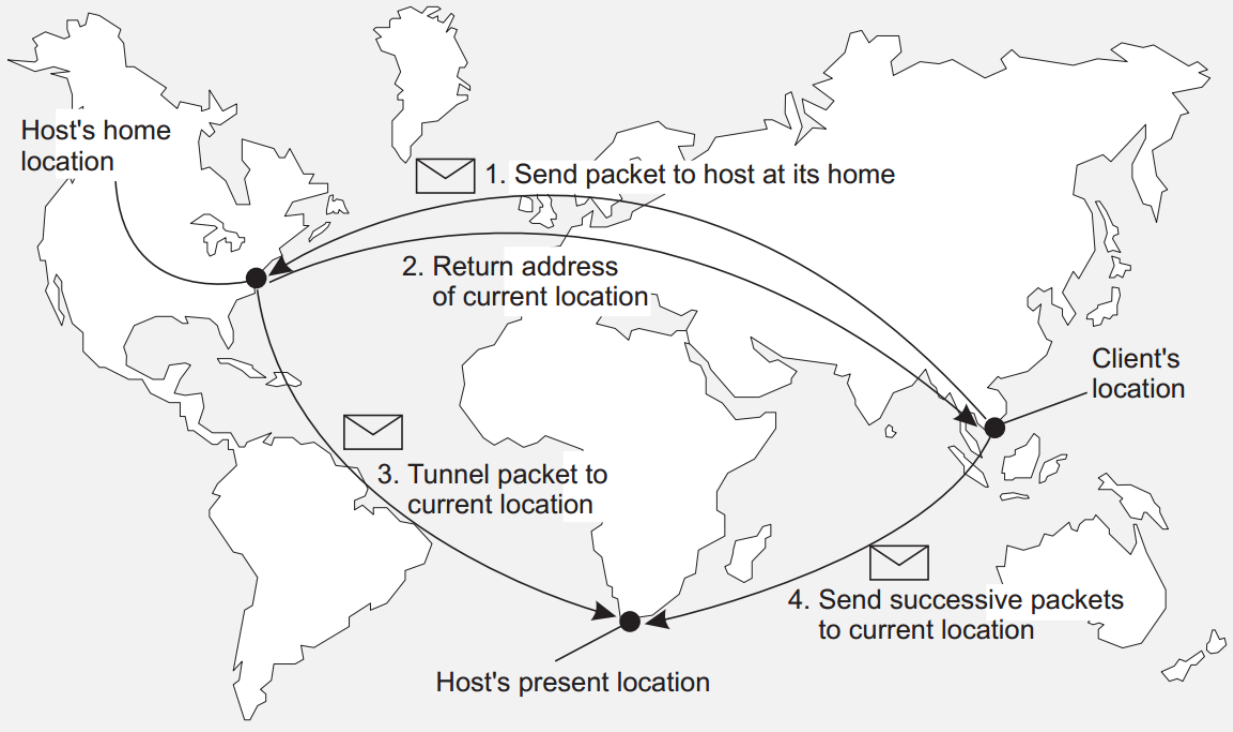
- Flat name…
  - Forwarding pointers…
    - Limitations
      - Long chain makes locating entity prohibitively expensive
        - » Increased network latency at dereferencing
      - Intermediate locations in a chain will have to maintain their part of the chain of forwarding pointers
      - Vulnerability to broken links

Names, Addresses, Name Resolution…

- Flat name…
  - Home-based approaches
    - Home location
      - Popular approach to supporting mobile entities in large-scale networks
      - Keeps track of the current location of an entity
      - Is often chosen to be the place where an entity was created
      - Used
        - » As a fall-back mechanism for location services based on forwarding pointers
        - » In mobile IP

Names, Addresses, Name Resolution…

- Flat name…
  - Home-based approaches…



Names, Addresses, Name Resolution…

- Flat name…
  - Home-based approaches…
    - Limitations
      - Increased communication latency
        - » When the home location is different than the location of the entity
        - » Even when entity is next to the client
      - Fixed home location
        - » It always has to exist
        - » Unnecessary burden when the entity moves permanently
      - Home address has to be supported for entity's lifetime

Names, Addresses, Name Resolution…

- Flat name…
  - Distributed hash table (DHT)
    - Chord
      - A protocol and algorithm for a peer-to-peer DHT
      - Considers the organization of many 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 nodes with smallest id ≥ k
          - The node is called successor of k and is denoted as succ(k)
      - Goal: to efficiently resolve a key k to the address of succ(k)
        - » Non-scalable approach
          - To let each node, p, keep track of the successor succ(p+ 1) as well as its predecessor pred(p)
          - Performs a linear search along the ring to lookup a key, k

Names, Addresses, Name Resolution…

- Flat name…
  - Distributed hash table…
    - Finger table
      - Each node with identifier *p* maintains a finger table $FT_p[]$ with at most m entries

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

Note: $FT_p[i]$ points to the first node succeeding identifier *p* by at least 2^{i-1}

      - To lookup a key, k, node with identifier p forwards the request to node with index j satisfying

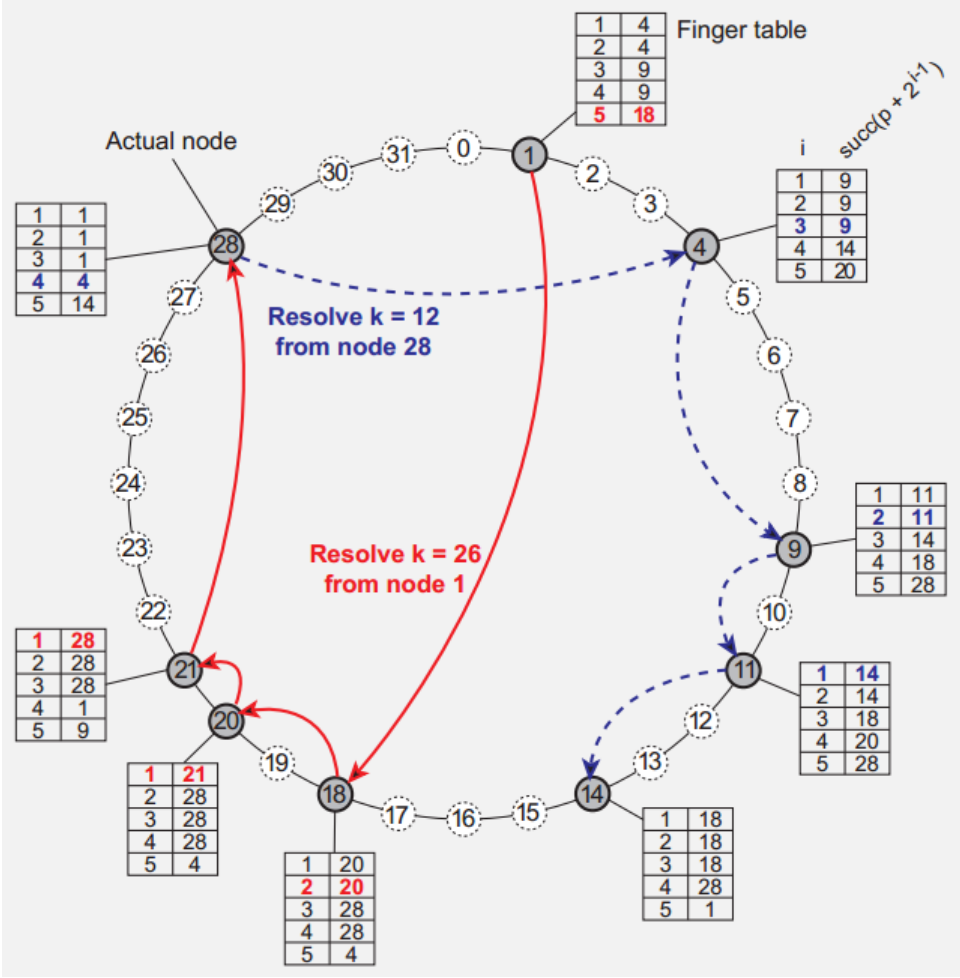
$$q= FT_p[j] \leq k < FT_p[j+1]$$

=> *Try to get closer to k, but don't overshoot*

      - If $p<k<FT_p[1]$, the request is also forwarded to $FT_p[1]$

Names, Addresses, Name Resolution…

- Flat name…
  - DHT…
    - Chord…
    - Example: m=5
      - Finger table
        - » $FT_p[i] = succ(p+2^{i-1})$
      - Lookup
        - » $q = FT_p[j] \le k < FT_p[j+1]$
        - » If $p<k<FT_p[1]$, the request is also forwarded to $FT_p[1]$
      - Resolve k=26 and 12



Names, Addresses, Name Resolution…

- Flat name…
  - Distributed hash table…
    - Exploiting network proximity
      - Problem
        - » The logical organization of nodes in the overlay may lead to erratic message transfers in the underlying Internet
        - » E.g., successive nodes in different continents
      - Solution
        - » Take the underlying network into account while designing DHT-based systems
        - » 3 approaches proposed…
      - Topology-based assignment
        - » When assigning an ID to a node, make sure that nodes close in the ID space are also close in the network

Names, Addresses, Name Resolution…

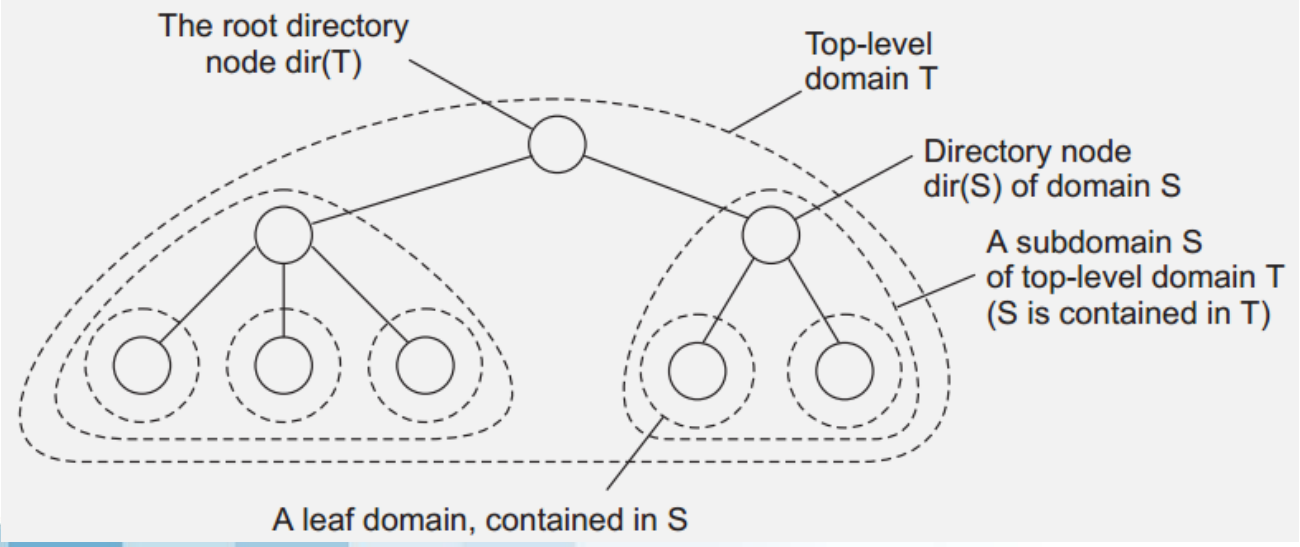
- Flat name…
  - Distributed hash table…
    - Exploiting network proximity…
      - Proximity routing
        - » Nodes maintain a list of alternative successors to forward a request to
          - Example: in Chord $FT_p[i]$ points to first node in INTERVAL = $[p + 2^{i-1}, p + 2^i - 1]$
          - Node p can also store pointers to other nodes in INTERVAL
        - » Advantage of having multiple successors for every table entry
          - Forwards a request to the node closest to itself
          - Node failures need not immediately lead to failures of lookups

Names, Addresses, Name Resolution…

- Flat name…
  - Distributed hash table…
    - Exploiting network proximity…
      - Proximity neighbor
        - » Idea: Optimize routing tables such that the nearest node is selected as neighbor
        - » When there is a choice of selecting who your neighbor will be (not in Chord), pick the closest one
    - Iterative vs recursive lookup
      - Iterative: process sends a request to a node, gets the address of the next node, sends a request to the next node …
      - Recursive: the requested node forwards a lookup to the next node

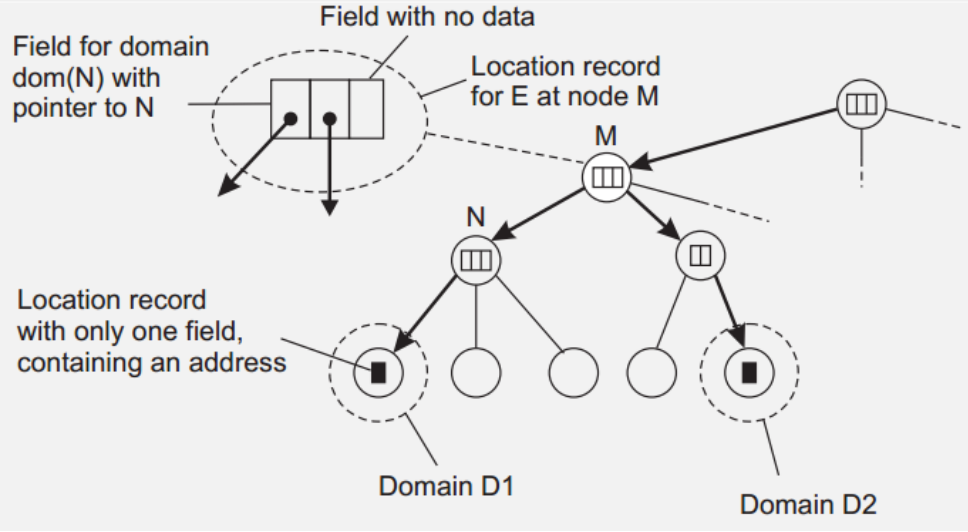
Names, Addresses, Name Resolution…

- Flat name…
  - Hierarchical approach
    - A network is divided into a collection of domains
      - Single top-level (root) domain spans the entire network
      - Each domain can be sub-divided into multiple, smaller sub-domains
      - Leaf domain 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



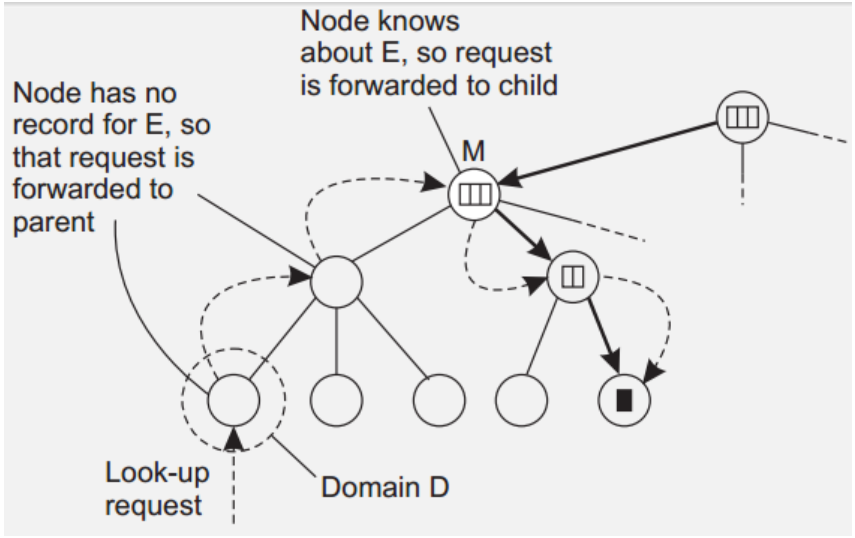
Names, Addresses, Name Resolution…

- Flat name…
  - Hierarchical approach…
    - Each entity currently located in a domain D is represented by a location record in the directory node, dir(D)
      - Address of entity, E, is stored in a leaf or intermediate node
    - 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
    - The root knows about all entities



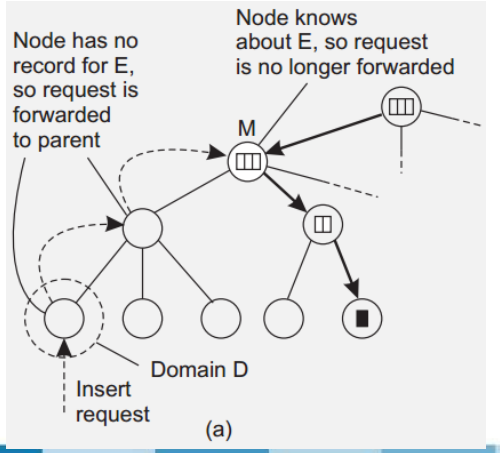
Names, Addresses, Name Resolution…

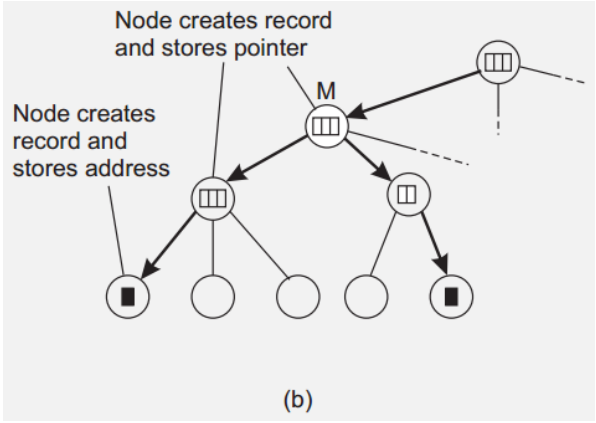
- Flat name…
  - Hierarchical approach…
    - Lookup operation
      - Start lookup for entity, E, at local leaf node
      - Node knows about E, then follow down ward pointer, else go up
      - Upward lookup always stops at root
      - Observation
        - » Lookup operation exploits locality



Names, Addresses, Name Resolution…

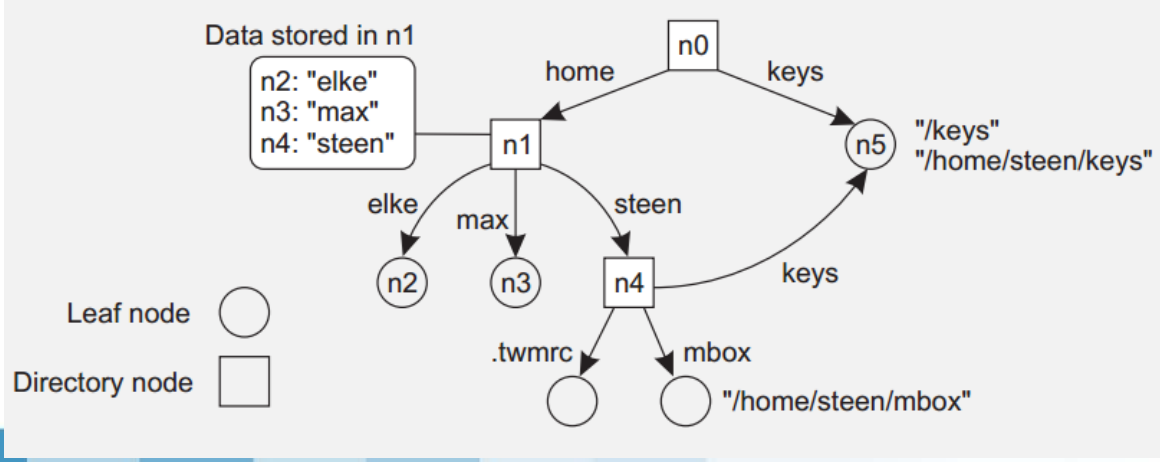
- Flat name…
  - Hierarchical approach…
    - Insert operation
      - Example: Inserting address of an entity E's replica in leaf domain D
        - » The insertion is initiated at the leaf node dir(D) of D which forwards the insert request to its parent
        - » Request is forwarded until it reaches a directory node M that already stores a location record for E
        - » Observation
          - Locality





Names, Addresses, Name Resolution…

- Structured naming
  - Names that are composed from simple, human-readable names
  - Name spaces
    - Used to organize names
    - Represented as a labeled, directed graph with two types of nodes
      - Leaf node
        - » Represents a named entity
        - » Stores information (e.g., address, state) on the entity
      - Directory node
        - » Has a number of outgoing edges, each labeled with a name
        - » Has a directory table
          - Stores a table representing an outgoing edge as a pair (edge label, node identifier)



Names, Addresses, Name Resolution…

- Structured naming…
  - Name spaces…
    - Each path in a naming graph can be referred to by the sequence of labels corresponding to the edges in the path
    - Global name
      - Denotes the same entity no matter where that name is used in the system
      - Always interpreted with respect to the same directory node
    - Local name
      - Interpretation depends on where the name is being used
      - Same as relative name whose directory in which it is contained is (implicitly) known

Names, Addresses, Name Resolution…

- Structured naming…
  - Name resolution
    - Is the process of looking up a name
    - Given a path name, it should be possible to look up any information stored in the node referred to by that name
    - To resolve a name, we need to know the node from which a name resolution resolves
    - Problem
      - How do we find the (initial directory) node?
    - Solution
      - Know before hand how and where to start => closure mechanism

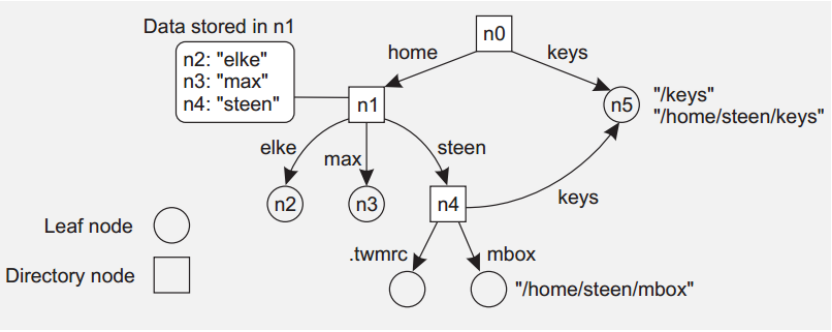
Names, Addresses, Name Resolution…

- Structured naming…
  - Name resolution
    - Closure Mechanism
      - Is the separation of the name space from how to start
      - Deals with selecting the initial node in a name space from which name resolution is to start
      - Requires implementation of some mechanism by which the resolution process can start

      - Why are closure mechanisms implicit?
        - » You have to have a naming outside the name space from which you start the name resolution
        - » Example
          - www.aait.edu.et : start at a DNS name server
          - /home/steen/mbox : start at the local NFS fileserver
          - 00251911101010 :
            - dial a phone number

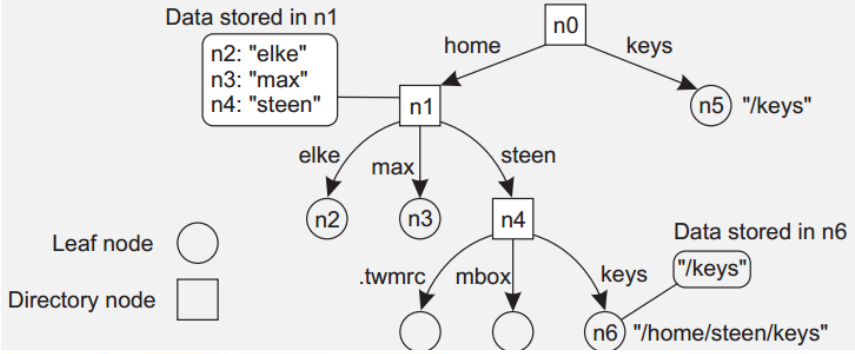
Names, Addresses, Name Resolution…

- Structured naming…
  - Linking
    - Alias
      - Is another name for the same entity
        - » E.g., environmental variables
      - Two ways to implement alias in naming graphs
      - Hard link
        - » Multiple absolute path names refer to the same node in a naming graph
      - Soft link
        - » Leaf node stores an absolute path name
        - » Corresponds to the use of symbolic links in UNIX file system



Names, Addresses, Name Resolution…

- Structured naming…
  - Linking…
    - Alias…
      - Soft link name resolution
        - » First resolve the leaf node containing the absolute path
        - » Read the content of the leaf node, yielding “name”
        - » Name resolution continues with “name”



Names, Addresses, Name Resolution…

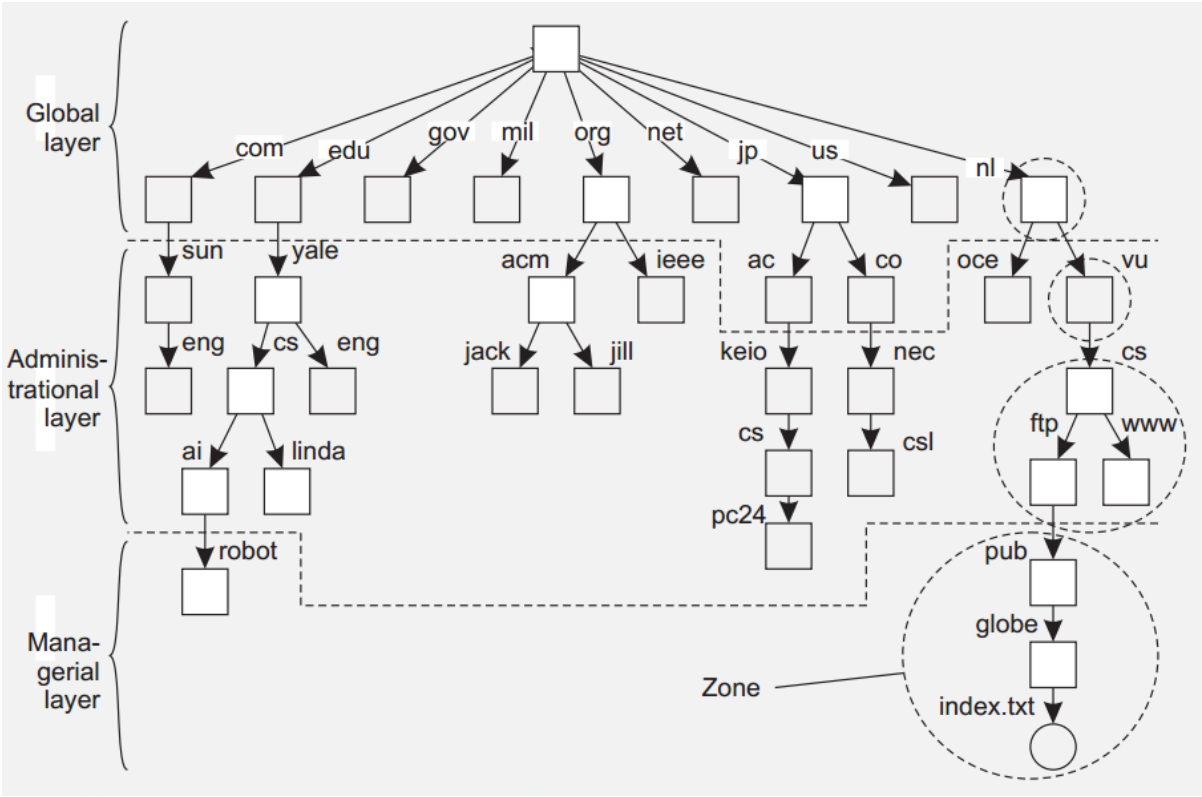
- Implementation of name space
  - Name space is the heart of a naming service
  - Name service is implemented by name servers
    - LAN – single name server is enough
    - Large scale DS – Multiple name servers
  - Name spaces are partitioned into logical layers and organized into hierarchy to implement them effectively

Names, Addresses, Name Resolution…

- Implementation of name space…
  - Layers
    - Global layer
      - Consist of highest-level nodes that are stable
      - Represents a (group of) organizations which manage it jointly
    - Administrational layer
      - Consists of mid-level directory nodes that are managed within a single organization
      - Represent groups of entities that belong to the same organization or administration unit
      - Relatively stable
    - Managerial layer
      - Consists nodes that change regularly
      - Example:
        - » hosts in a network
      - Managed by system administrators and end users of a DS

Names, Addresses, Name Resolution…

- Implementation of name space…
  - Zone
    - A non-overlapping division of a name space
    - Implemented by a separate name server



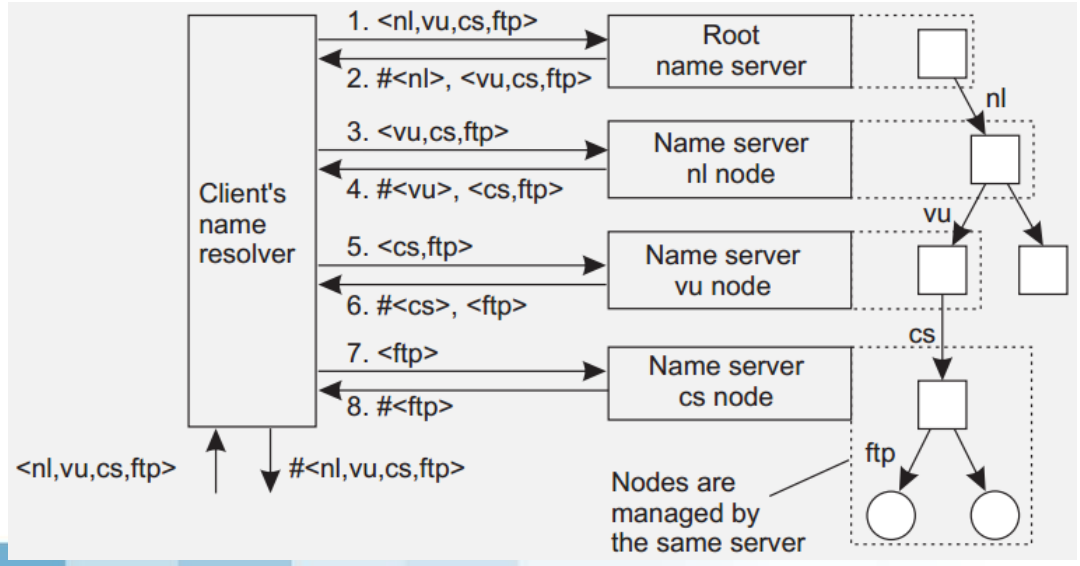
Names, Addresses, Name Resolution…

- Implementation of name space…

Item	Global	Administrational	Managerial
Geographical scale of network	Worldwide	Organizational	Departmental
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
Client-side caching	Yes	Yes	Sometimes

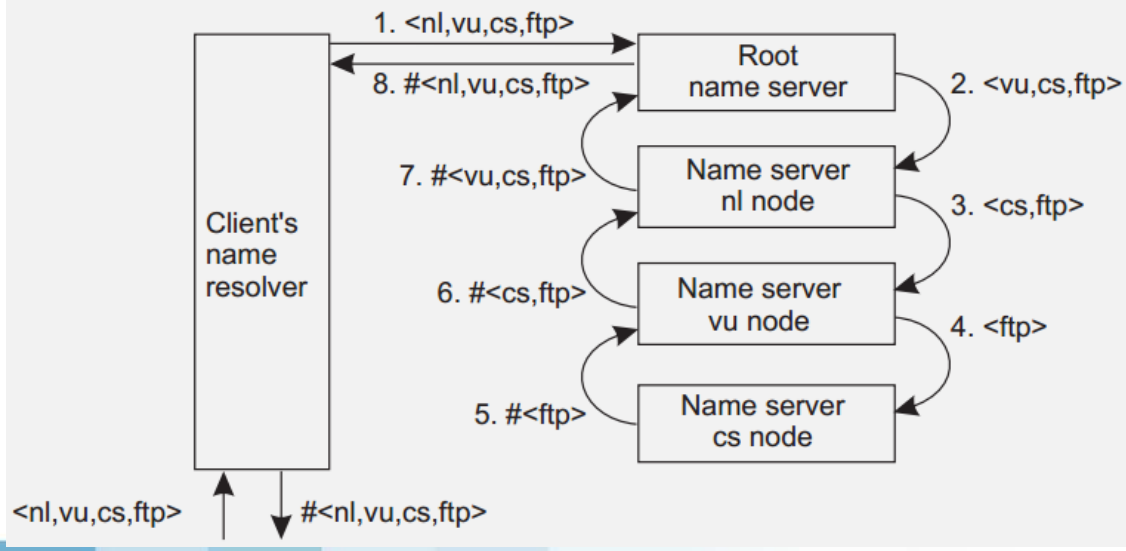
Names, Addresses, Name Resolution…

- Implementation of name resolution
  - Iterative name resolution
    1. *resolve(dir, [name$_1$, …, name$_k$])* sent to *Server$_0$* responsible for *dir*
    2. *Server$_0$* resolves *name$_1$* using *dir*, and returns the address of *server$_1$* which stores *dir$_1$*
    3. Client sends *resolve(dir$_1$, [name$_2$, …, name$_k$])* to *Server$_1$*
    4. …



Names, Addresses, Name Resolution…

- Implementation of name resolution…
  - Recursive name resolution
    1. *resolve(dir, [name$_1$, …, name$_k$])* sent to *Server$_0$* responsible for *dir*
    2. *Server$_0$* resolves *name$_1$* using *dir*, 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 result to client

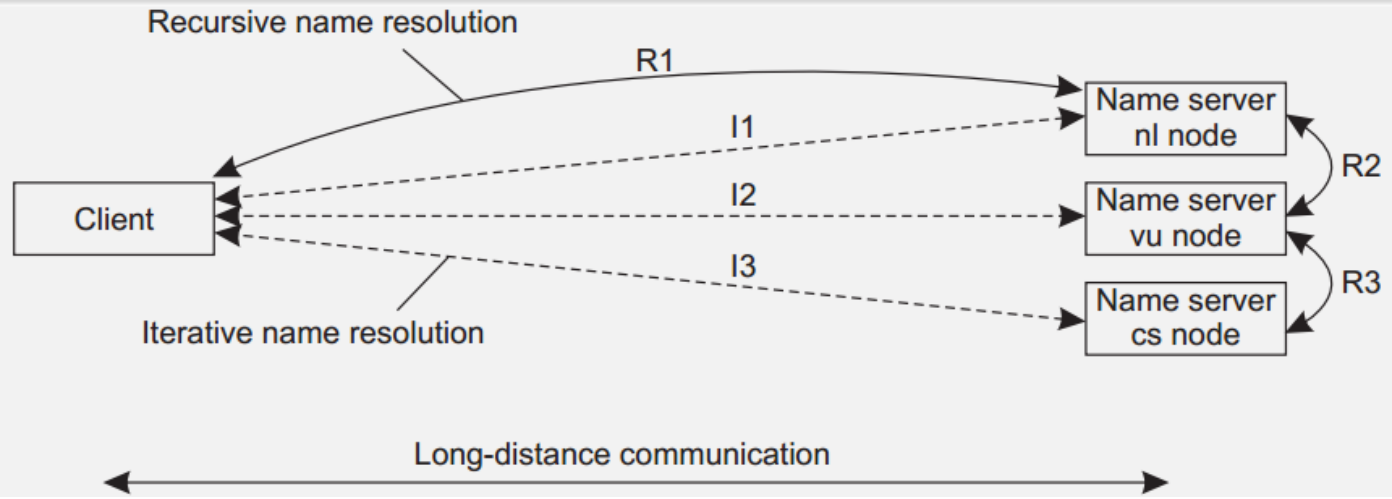


Names, Addresses, Name Resolution…

- Implementation of name resolution…
  - Recursive name resolution…
    - Drawbacks
      - Puts higher performance demand on each name server
    - Possible solution
      - Replication at the global and administration level
    - Advantages
      - Caching results is more effective
        - » *(for iterative name resolution, caching is restricted to the client's name resolver)*

Names, Addresses, Name Resolution…

- Implementation of name resolution…
  - Recursive name resolution…
    - Advantage…
      - Reduced communication costs



Internet DNS

- Used to lookup IP addresses of hosts and mail servers
- DNS name space is hierarchically organized as a rooted tree
- Label
  - Is case-insensitive string made up of alphanumeric characters
  - Has a maximum length of 63 characters
- Path name
  - Has a length of 255 characters
  - Composed of labels separated by a dot (.)
  - Starts by a dot(.) from the rightmost
    - The rightmost dot also represents the root (e.g., ftp.cs.vu.nl.)
- Domain
  - Represents a sub-tree
  - The path name to the domain is called a domain name

Internet DNS…

- Contents of a node is formed by a collection of resource records

SOA	Zone	Holds info on the represented zone
A	Host	IP addr. of host this node represents
MX	Domain	Mail server to handle mail for this node
SRV	Domain	Server handling a specific service
NS	Zone	Name server for the represented zone
CNAME	Node	Symbolic link
PTR	Host	Canonical name of a host
HINFO	Host	Info on this host
TXT	Any kind	Any info considered useful

Internet DNS…

- DNS Implementation
  - DNS name space is divided into a global layer and an administration layer
    - Managerial layer is generally formed by local file system and is not part of DNS
  - Each zone is implemented by a name server
  - Implementation of a DNS database uses small files that contain all the resource records for all the nodes in a particular zone

Internet DNS…

- Decentralized DNS implementation
  - Higher-level nodes receive many requests than lower-level nodes
  - Avoids scalability problems
    - Compute the hash of a DNS name, and take that hash as a key value to be looked up in a distributed-hash table
    - Drawback
      - Loses the structure of the original name