COMP2121: Principles of Distributed Systems and Networks

Naming

Unit coordinator Dr. Vincent Gramoli
School of Information Technologies

THE UNIVERSITY OF SYDNEY

Introduction

- Previous lecture:
  - RPC/RMI gives the impression that entities can be accessed locally

- Today's lecture:
  - How do we actually name these entities to retrieve and access them later on?
  - How to retrieve and bring remote resources (e.g., file, webpage) locally?

Outline

- Name
- Locating an Entity
- Distributed File System
- Caching
- Domain Name System
- Distributed Hash Table

Name

THE UNIVERSITY OF
SYDNEY

Name

Name: used to denote an entity of a distributed system

- Names are used to refer to a variety of resources such as
  - Computers
  - Services
  - Remote objects
  - Files
  - Users
- What is a name used for?
  - to act on a particular resource out of many (e.g., URL: Uniform Resource Locator)
  - to share a resource with other processors (e.g., printer address)
  - to allow user communication (e.g., email address)
- Name service: provides a client with data about named objects in distributed systems
- Directory service: provides data about objects that satisfy a given description

Name type

- A *human-friendly name* represents an entity in some form that presents a meaning for human, like:
  - File name: /etc/password
  - Internet domain name: usyd.edu.au
- A *program-interpreted identifier* does not need to be human-friendly but can simply be a sequence of numbers, like:
  - Remote object references
  - NFS file handles
- A name is *resolved* when it is translated into data about the named resource of object, often to invoke an action upon it
- Given an object, one of its *attribute* is a value of a property associated with it
- A *binding* is the association between a name and an (attribute of an) object

Name format

Representations can help locate object or simply name a resource

- Uniform Resource Identifiers (URI) identify uniquely a resource
  - in a *coherent* way, such that the resource can be identified by many software,
  - and *uniformly*, such that new resources can be added.

- There are two kinds of URIs:
  - Uniform Resource Locator (URL): used to locate and access a resource
    Example: http://rp-www.cs.usyd.edu.au/~gramoli/php/ identifies the resource as a web page at the given path /~gramoli/php on the host rp-www.cs.usyd.edu.au and specifies that the resource should be accessed using HTTP
  - Uniform Resource Name (IRN): used to name rather than locate a resource
    Example: International Standard Book Number (ISBN) uniquely identifies books, e.g., ISBN:0470936908

Locating an Entity

THE UNIVERSITY OF
SYDNEY

Locating an entity

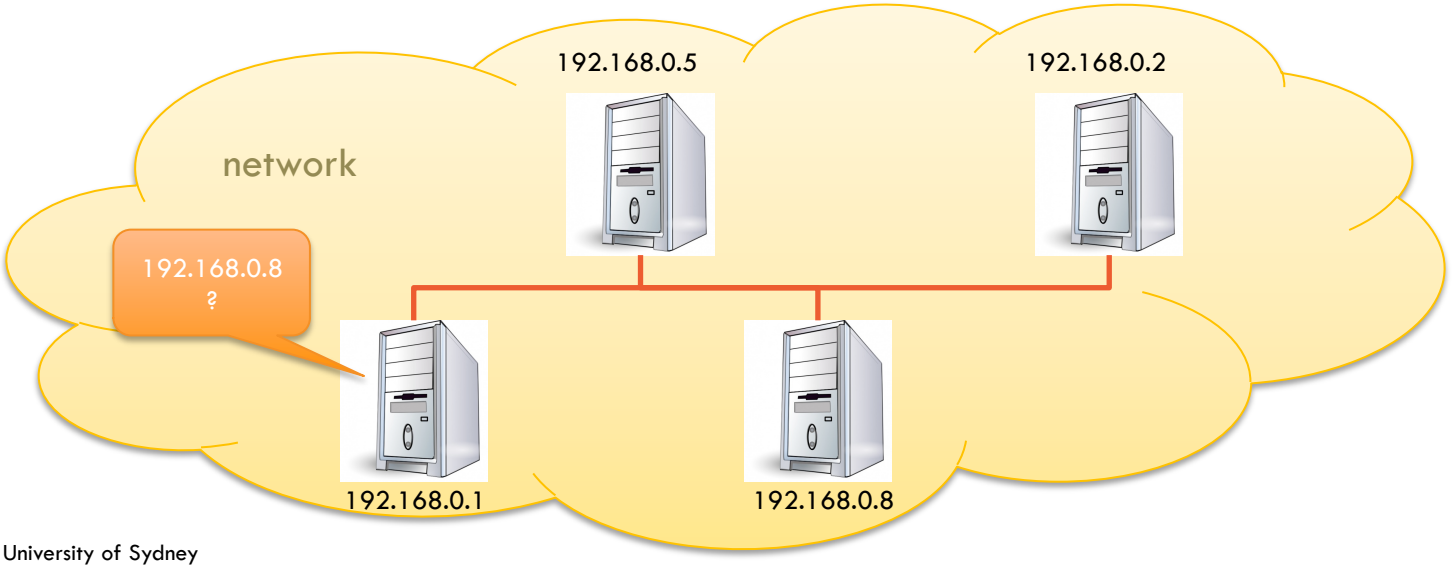
Broadcast: sending messages to all

- Physical address (@): identifies network interfaces
  - 48-bits that represent a unique number, set-up by the manufacturer (e.g., 40:6c:8f:17:52:0f)
  - A single machine may have multiple interfaces and physical addresses
  - Ethernet, Medium-Access Control (MAC) or Wireless addresses
  - The physical address of the receiver must be part of the Ethernet packet (msg)

Locating an entity

Broadcast: sending messages to all

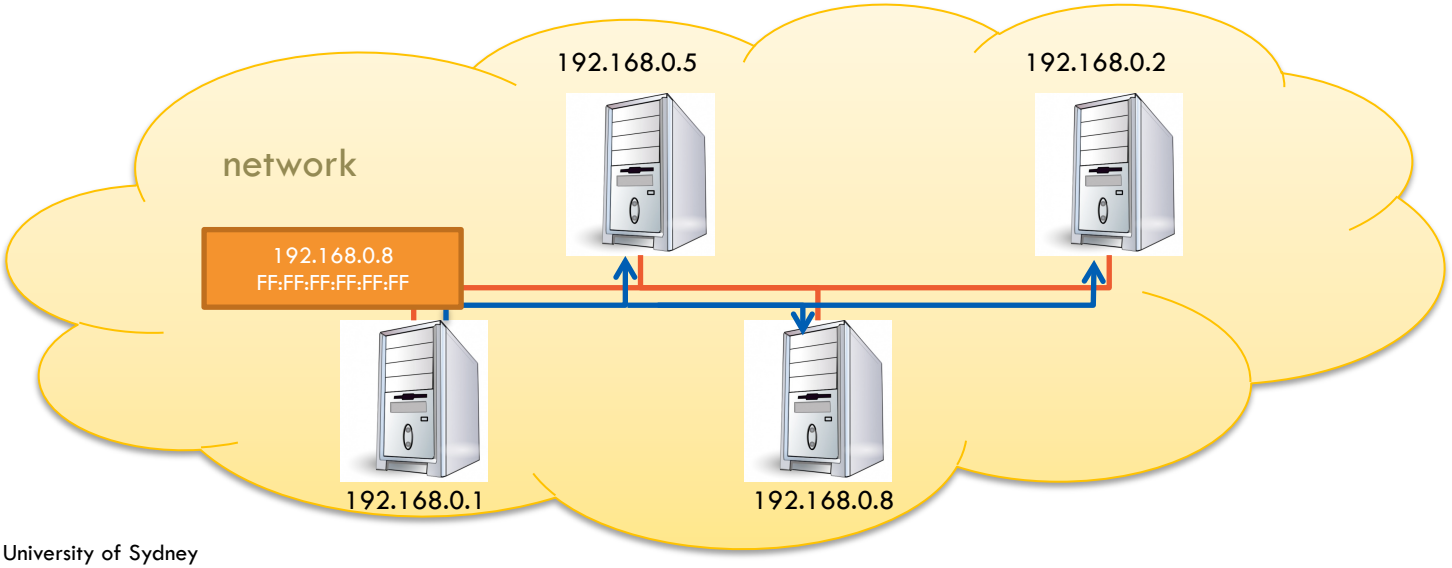
- Physical address (@): identifies network interfaces
  - 48-bits that represent a unique number, set-up by the manufacturer (e.g., 40:6c:8f:17:52:0f)
  - A single machine may have multiple interfaces and physical addresses
  - Ethernet, Medium-Access Control (MAC) or Wireless addresses
  - The physical address of the receiver must be part of the Ethernet packet (msg)
- Address Resolution Protocol (ARP) returns the physical @ given an IP @ in a network



Locating an entity

Broadcast: sending messages to all

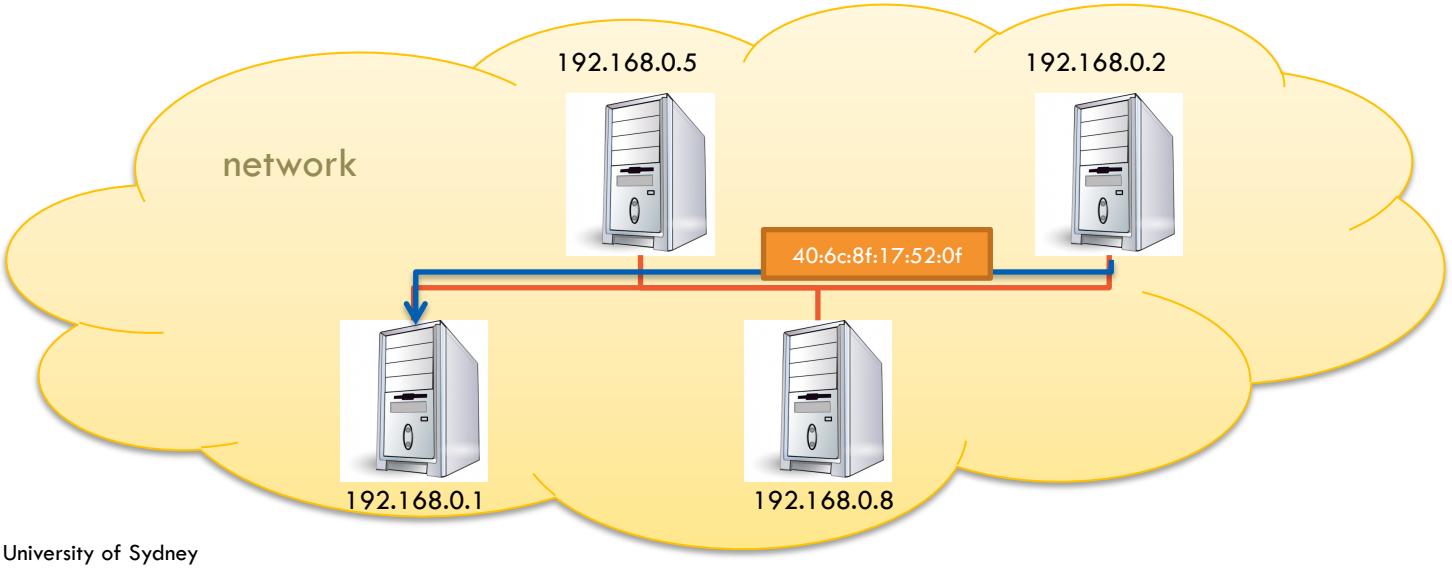
- Physical address (@): identifies network interfaces
  - 48-bits that represent a unique number, set-up by the manufacturer (e.g., 40:6c:8f:17:52:0f)
  - A single machine may have multiple interfaces and physical addresses
  - Ethernet, Medium-Access Control (MAC) or Wireless addresses
  - The physical address of the receiver must be part of the Ethernet packet (msg)
- Address Resolution Protocol (ARP) returns the physical @ given an IP @ in a network



Locating an entity

Broadcast: sending messages to all

- Physical address (@): identifies network interfaces
  - 48-bits that represent a unique number, set-up by the manufacturer (e.g., 40:6c:8f:17:52:0f)
  - A single machine may have multiple interfaces and physical addresses
  - Ethernet, Medium-Access Control (MAC) or Wireless addresses
  - The physical address of the receiver must be part of the Ethernet packet (msg)
- Address Resolution Protocol (ARP) returns the physical @ given an IP @in a network



Locating an entity

Broadcast: sending messages to all

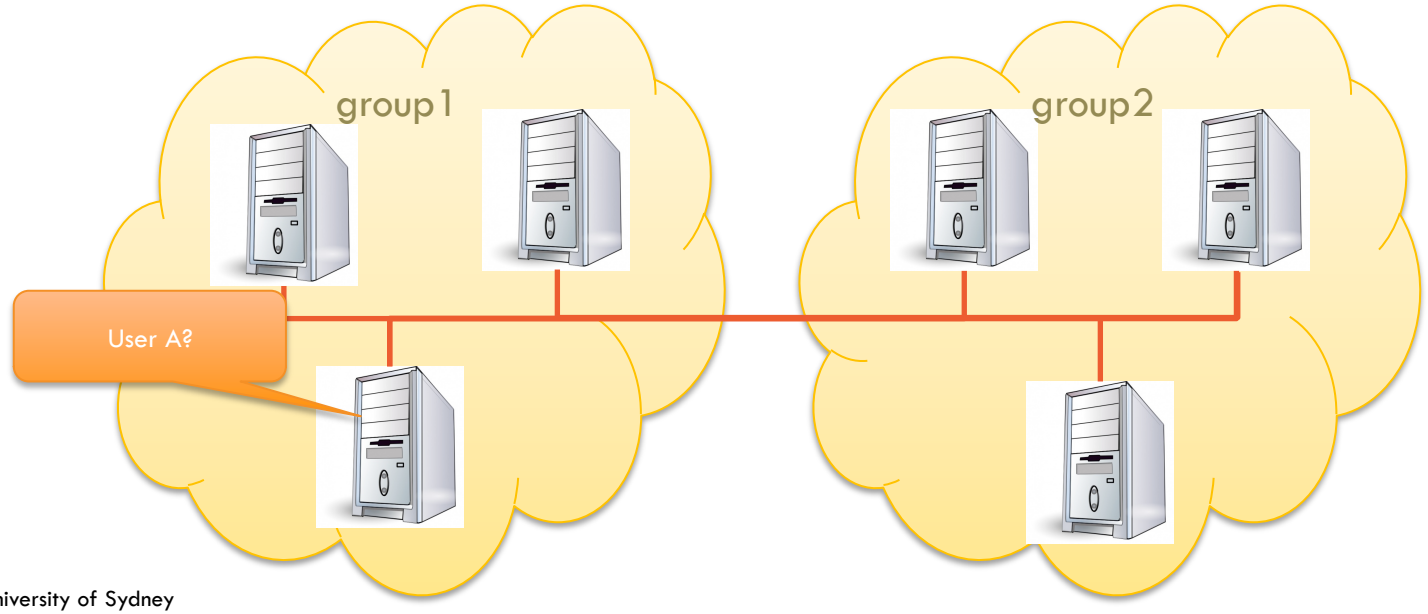
- Physical address (@): identifies network interfaces
  - 48-bits that represent a unique number, set-up by the manufacturer (e.g., 40:6c:8f:17:52:0f)
  - A single machine may have multiple interfaces and physical addresses
  - Ethernet, Medium-Access Control (MAC) or Wireless addresses
  - The physical address of the receiver must be part of the Ethernet packet (msg)
- Address Resolution Protocol (ARP) returns the physical @ given an IP @in a network

Broadcasting does not scale with network size as it becomes inefficient

Locating an entity

Multicast: sending messages to many

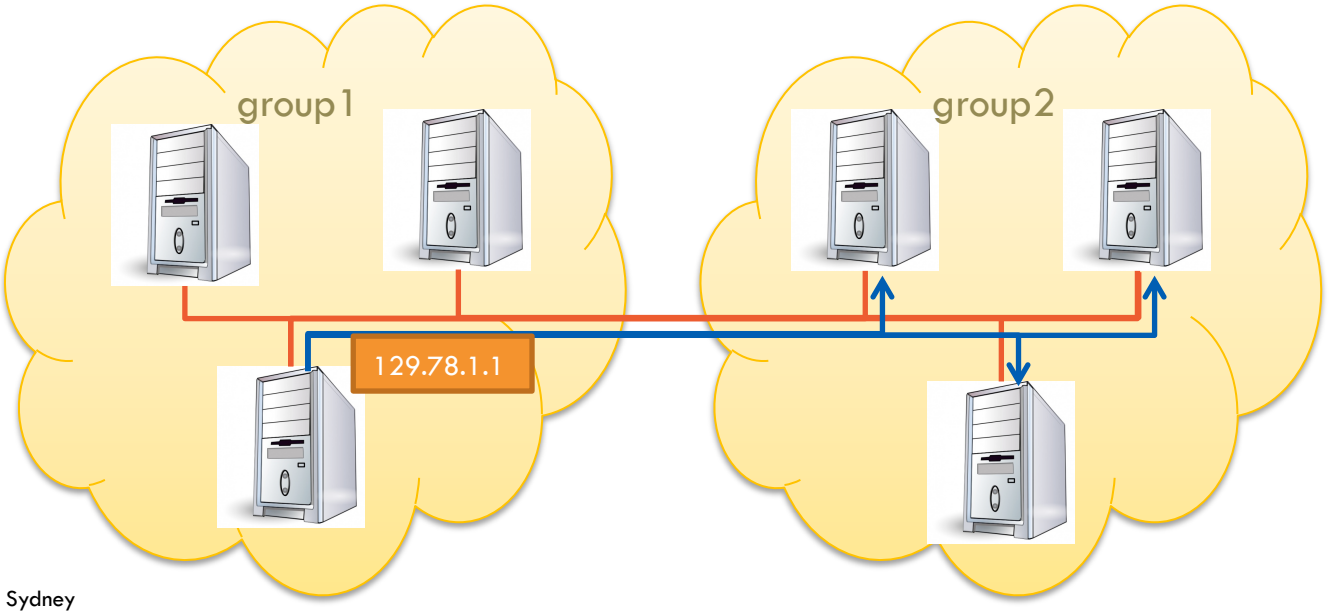
- Network-level multicasting: to obtain the particular IP address of a user
  - Supported in internet
  - Hosts join a specific multicast group
  - Each group is identified by a multicast address
  - Message to this address are delivered to all group members
  - The IP address must be part of the IP message



Locating an entity

Multicast: sending messages to many

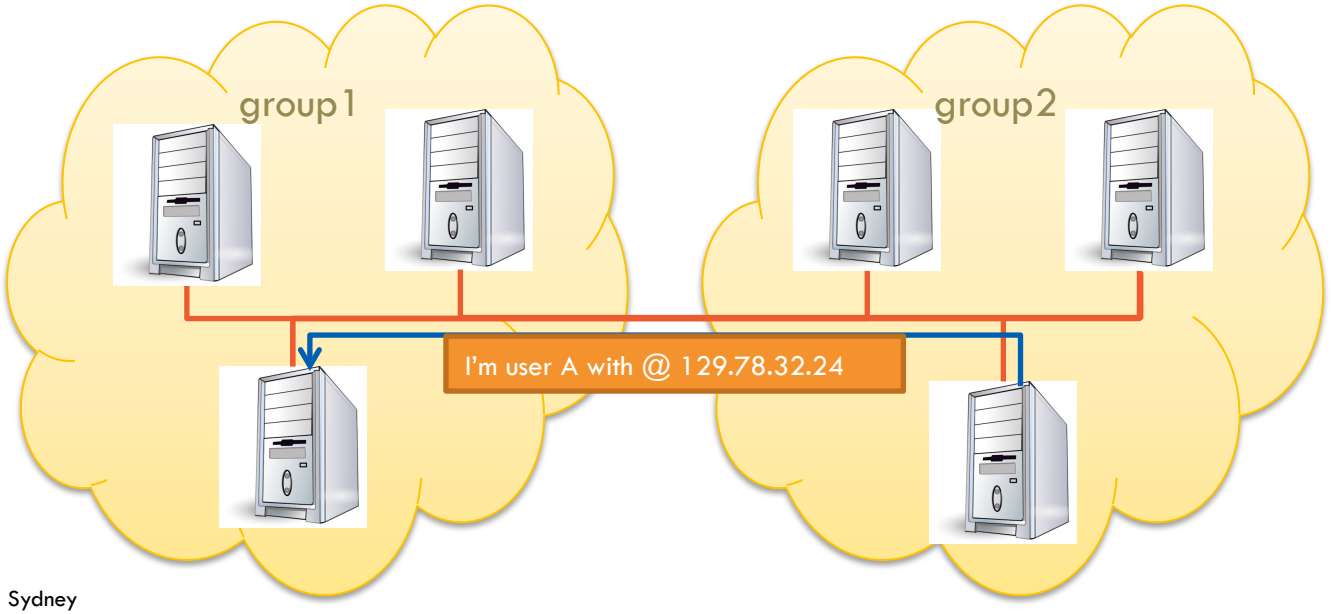
- Network-level multicasting: to obtain the particular IP address of a user
  - Supported in internet
  - Hosts join a specific multicast group
  - Each group is identified by a multicast address
  - Message to this address are delivered to all group members
  - The IP address must be part of the IP message



Locating an entity

Multicast: sending messages to many

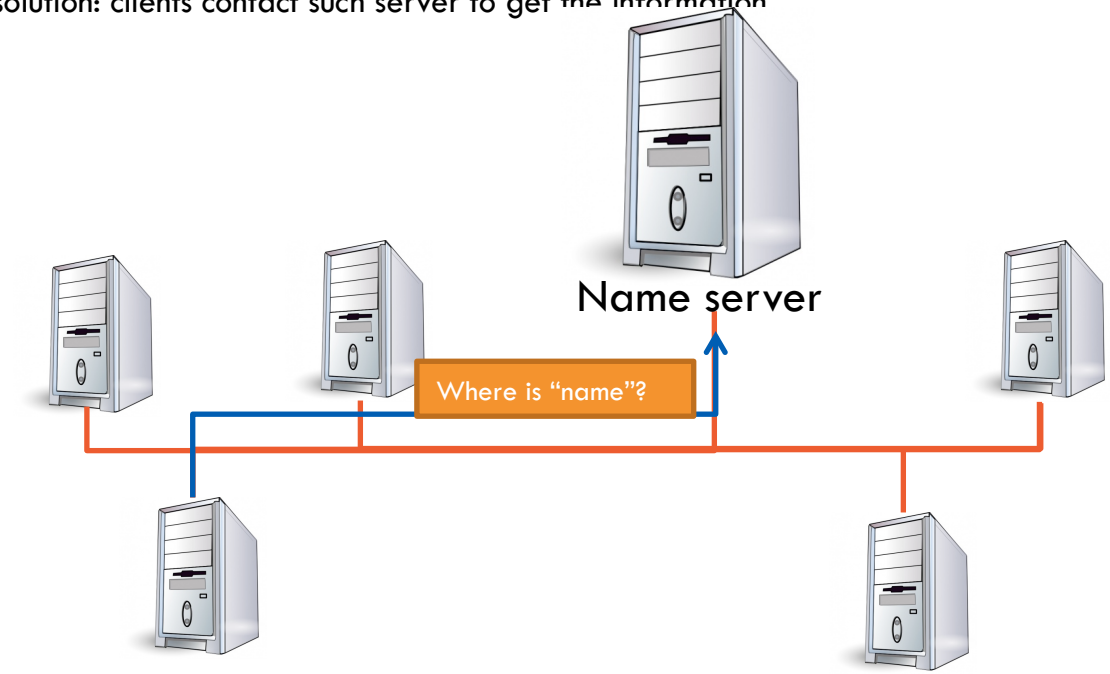
- Network-level multicasting: to obtain the particular IP address of a user
  - Supported in internet
  - Hosts join a specific multicast group
  - Each group is identified by a multicast address
  - Message to this address are delivered to all group members
  - The IP address must be part of the IP message



Locating an entity

Alternatives

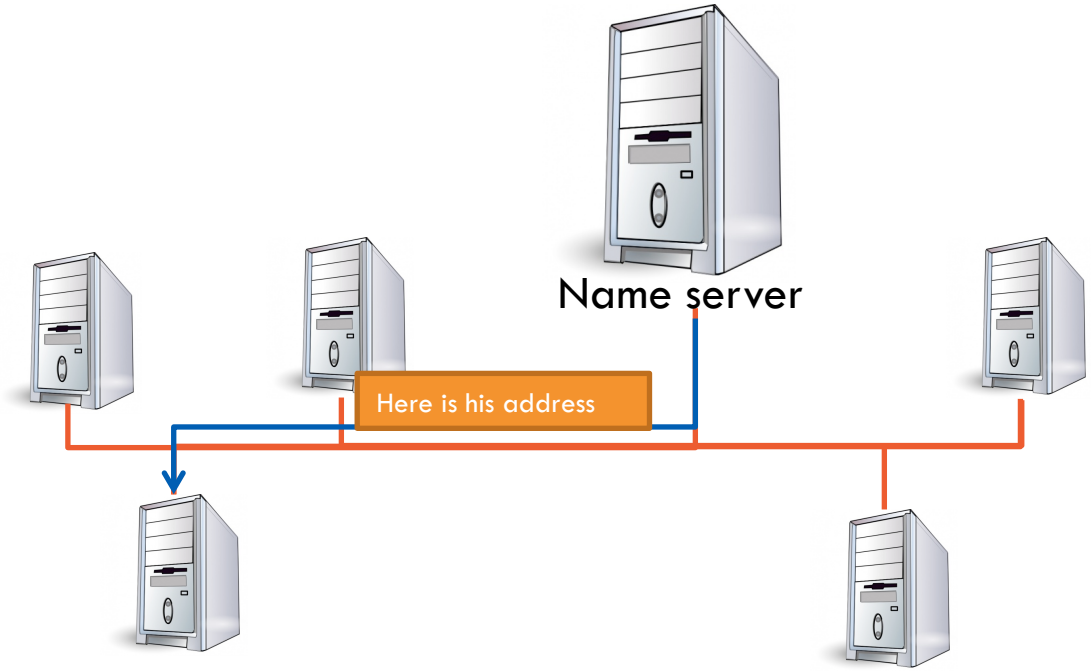
- Multicast/broadcast are costly:
  - lots of messages are wasted: the receiver simply ignore them
- Solution
  - Name servers: make servers responsible of maintaining the name-address mapping
  - Name resolution: clients contact such server to get the information



Locating an entity

Alternatives

- Costly: lots of messages are wasted: the receiver simply ignore them
- Solution
  - Name servers: make servers responsible of maintaining the name-address mapping
  - Name resolution: clients contact such server to get the information

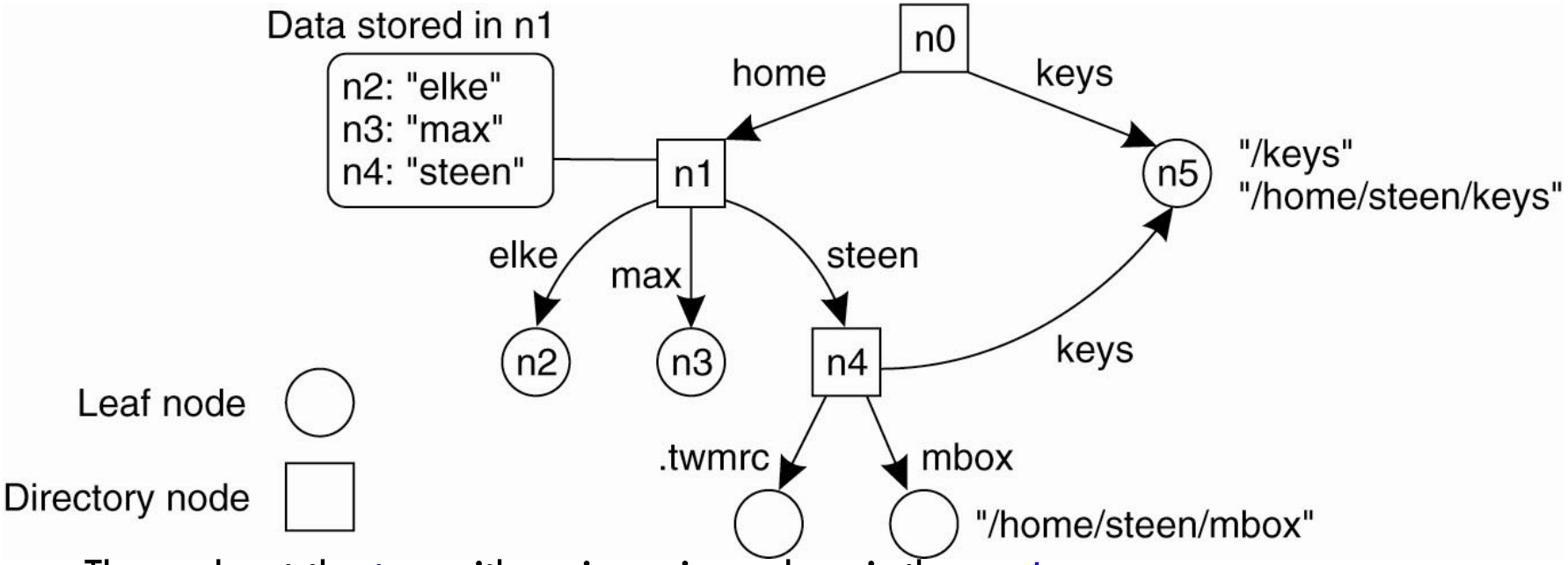


Distributed File System

THE UNIVERSITY OF SYDNEY

Distributed file system

Name space: organization of all the names

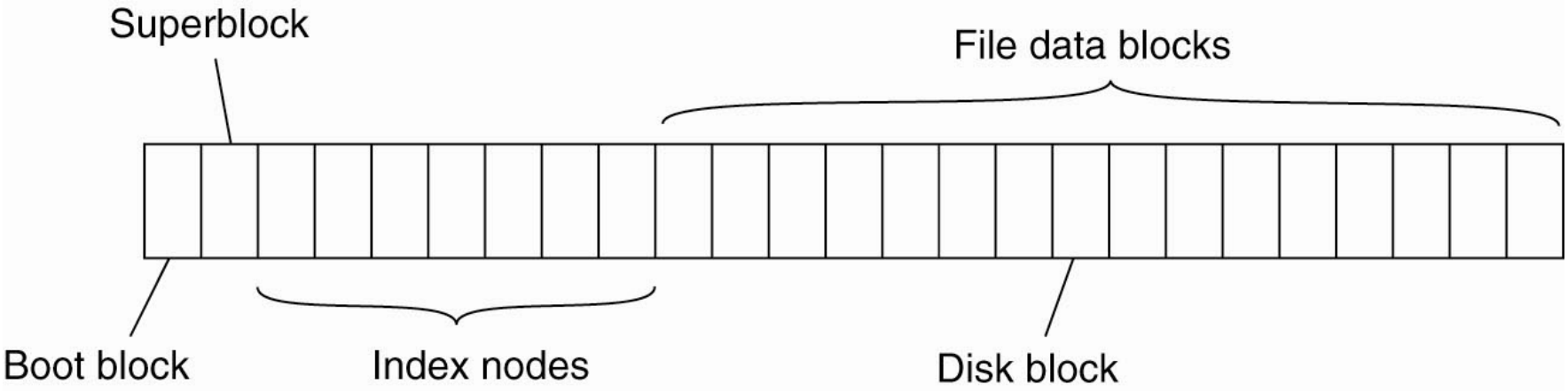


- The node at the top with no incoming edges is the *root*
- Each leaf is a file and each inner (non-leaf) node is a directory
- Each directory maintains a directory table with entries <label: "node identifier">
- The concatenated labels of a graph path is a *path name*
- A path name starting from the root is an *absolute path name* (as opposed to *relative* ones)

Distributed file system

Name space: organization of all the names

- The general organization of the UNIX file system implementation on a logical disk of contiguous disk blocks.

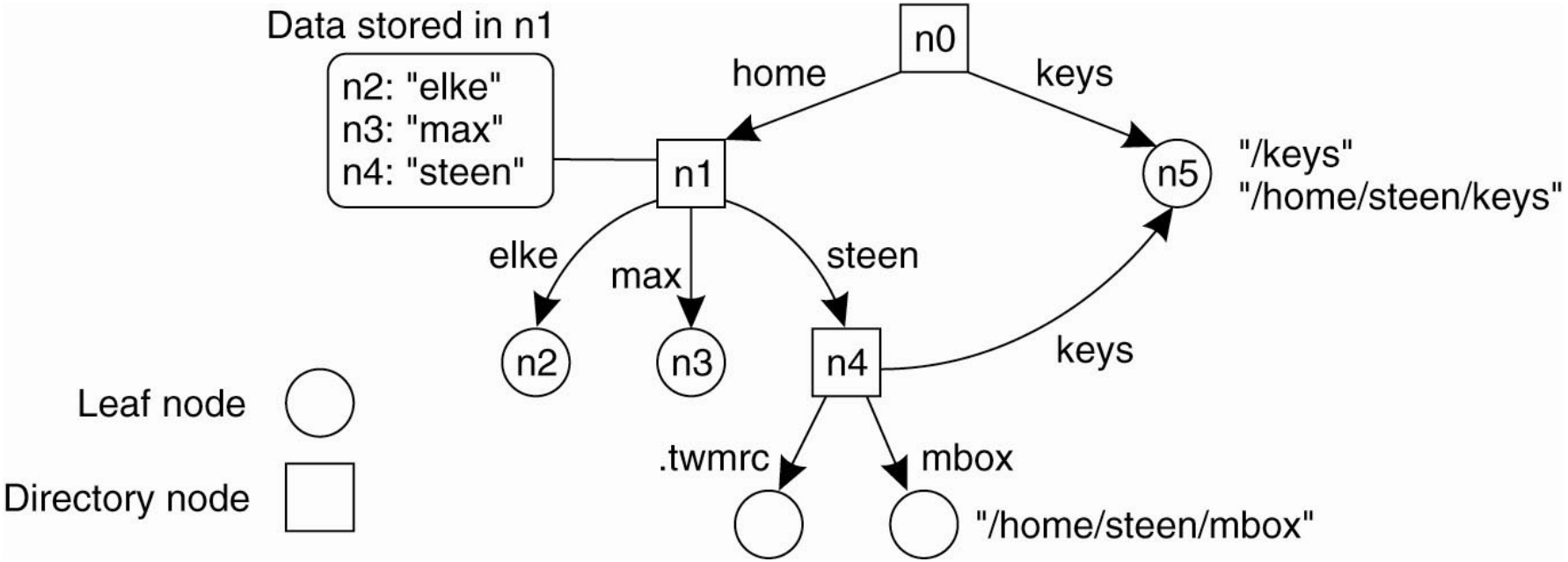


- The boot block is loaded into main memory at system boot time
- The superblock has info of the FS (size, unallocated blocks, unused inodes)
- *Inodes* (i.e., index nodes) have indexes starting at 0 (for the root) and have info about where to find their file data on disk, ownership.
- Each directory is represented as a file as well

Distributed file system

Alias: another name for the same entity (e.g., environment variable)

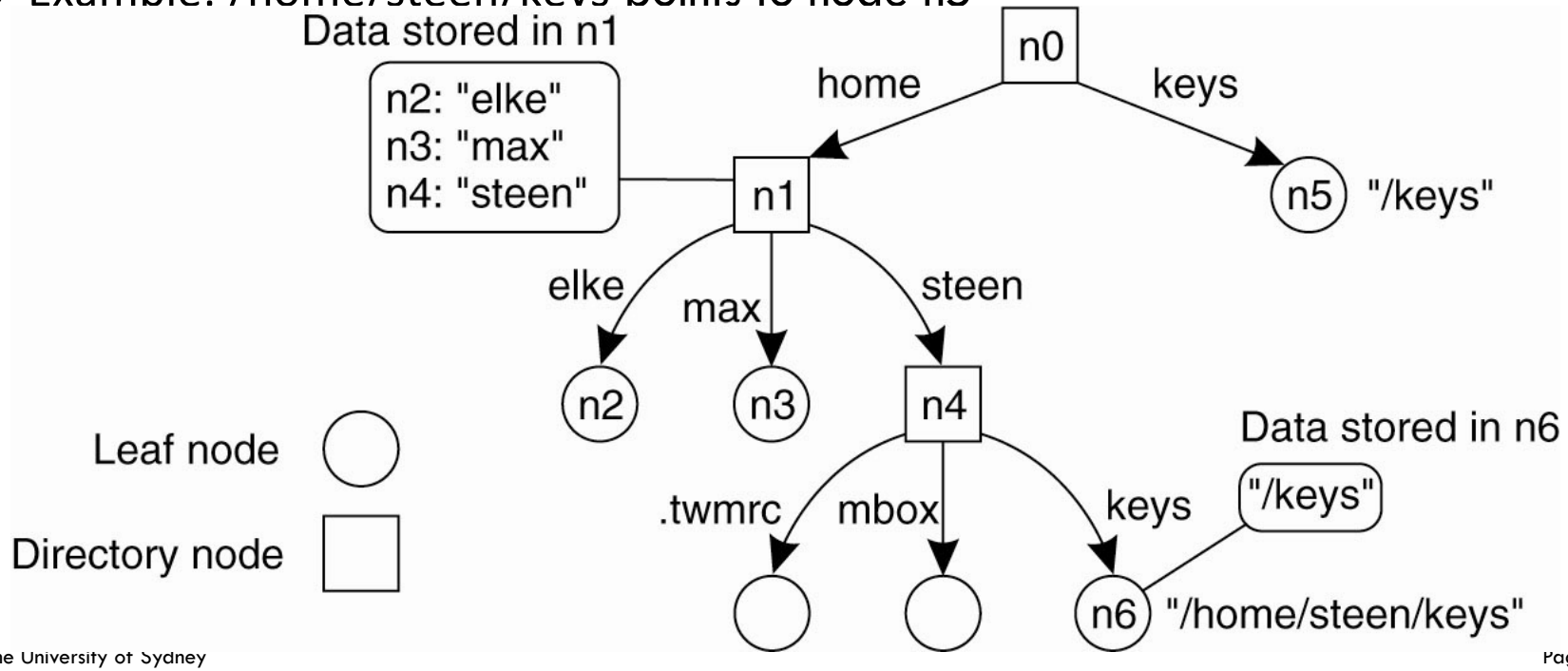
- Hard links:
  - Multiple absolute paths names refer to the same node in a naming graph
  - Example: /keys and /home/steen/keys point to node n5



Distributed file system

Alias: another name for the same entity (e.g., environment variable)

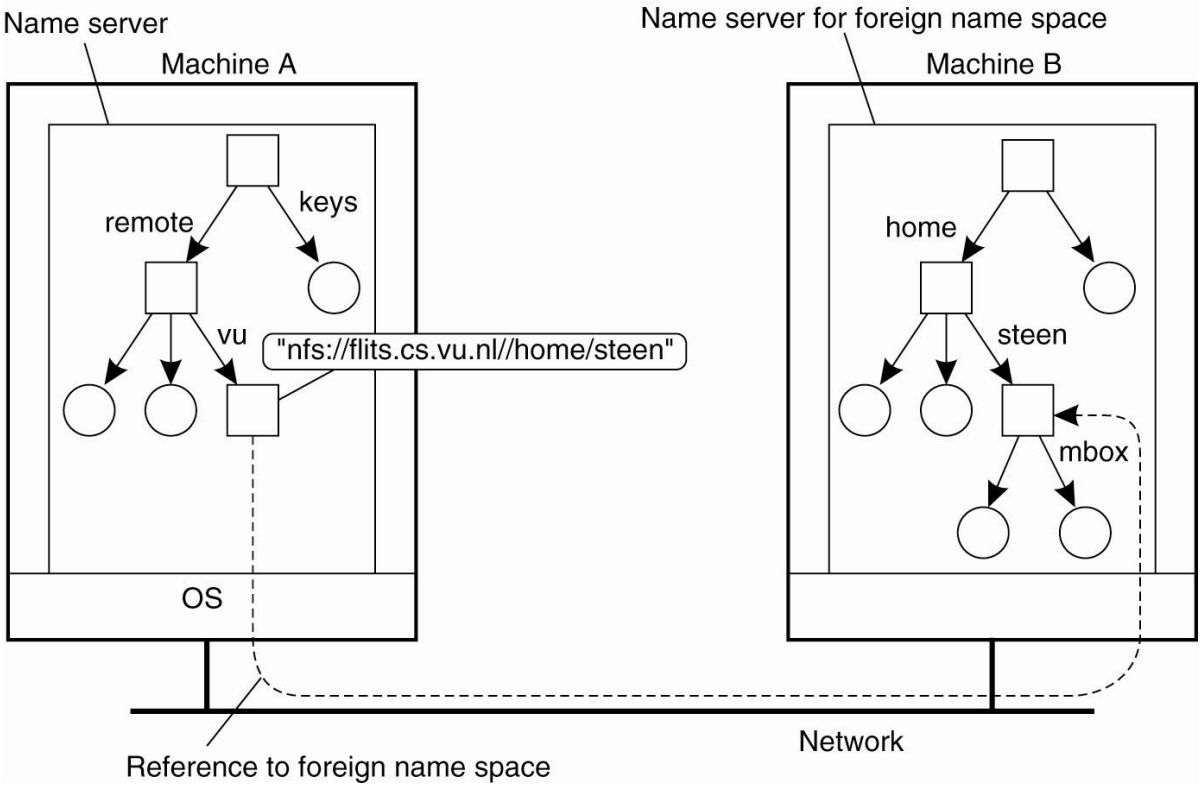
- Symbolic links:
  - a leaf node stores only an absolute path name; resolving a path name leading to this node returns another path name, that is resolved in turn
  - Example: /home/steen/keys points to node n5



Distributed file system

Mounting

- A *mounted file system* lets a directory node store the identifier of a directory node from a different name space. The former directory node is called the *mount point* while the latter is called the *mounting point*.
- To mount a remote foreign name space, one needs to specify the FS protocol name (e.g., nfs), the server name (e.g., flits.cs.vu.nl) and the mounting point (e.g., /home/steen)

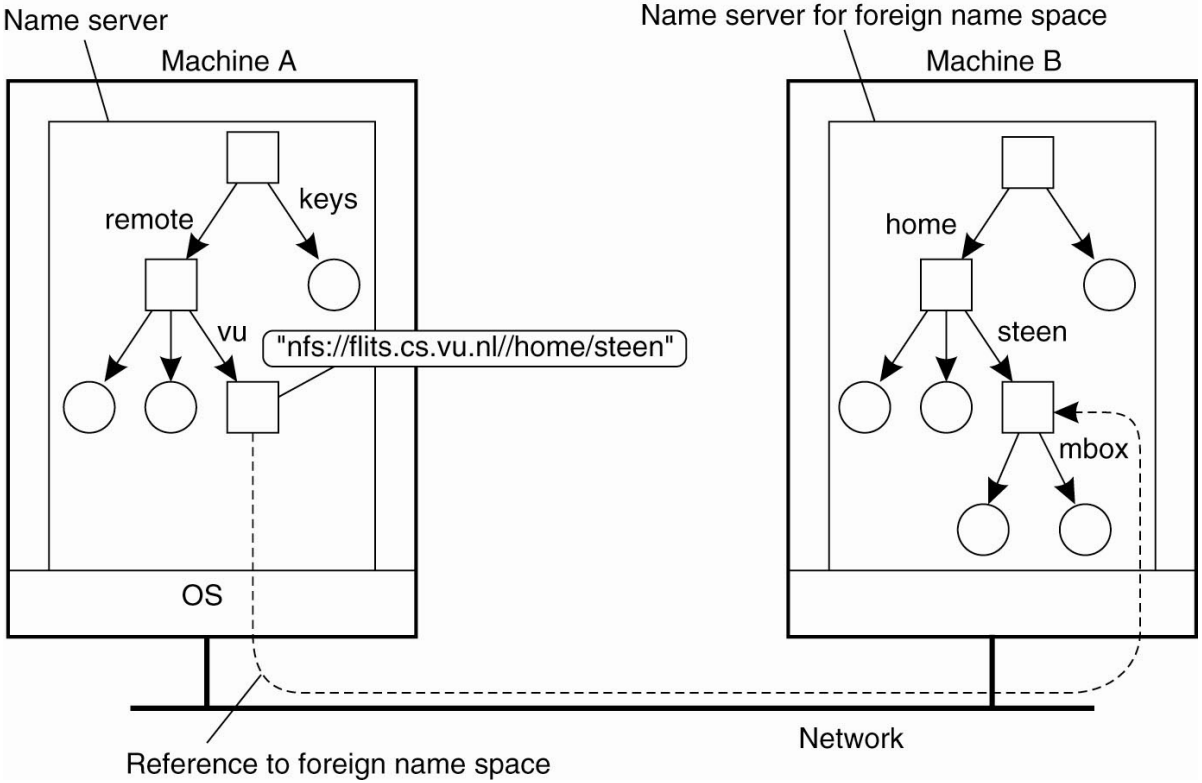


Distributed file system

Mounting (con't)

> cat /remote/vu/mbox

- The name /remote/vu/mbox is resolved by
  1. Starting in the root directory on the client's machine 'A' and continuing until node /remote/vu is reached
  2. The URL nfs://flits.cs.vu.nl//home/steen is returned, leading the client to contact file machine 'B' using the NFS protocol
  3. Then the client accesses the directory /home/steen in which file mbox is read, and the resolution process stops



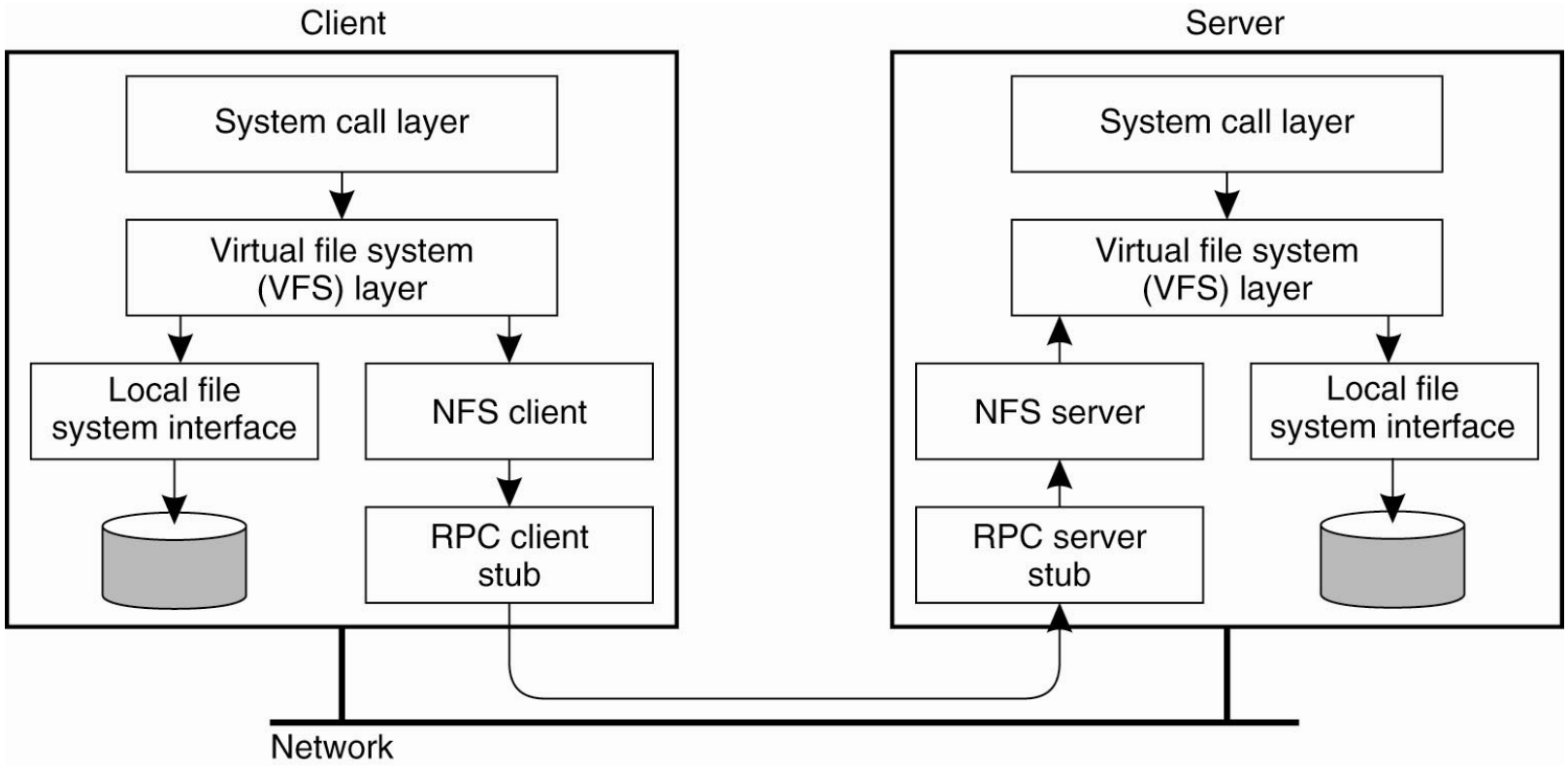
Distributed file system

Network File System (NFS)

- Virtual file system (VFS) layer provides mechanisms which give same system call interface for different types of file system
- NFS three layers
  - UNIX file-system interface: open, read, write, close calls + file descriptors
  - VFS layer: distinguishes local from remote files, calls NFS
  - NFS service layer: bottom layer, implements the NFS protocol
- NFS protocol: RPC for file operation on server
  - Reading/searching a directory
  - Manipulating links and directories
  - Accessing file attributes/reading and writing files
- NFS servers are stateless (up to v3)
  - Each request includes all the information necessary for execution
- Modifications were committed to the server's disk before returning to the client
  - Very slow
- Fault-tolerance can be achieved through replicated file servers with same directories

Distributed file system

NFS (con't)



Caching

THE UNIVERSITY OF SYDNEY

Caching

Exploiting locality by caching data in memory

- Accessing a remote entity can take time
  - Network access
  - Disk access
- Caching consists in keeping a copy locally (in memory)
  - Accessing the remote entity creates a copy in cache
  - Re-accessing the same entity accesses the cached copy instead of remote entity
  - *Cache-hit*: if the copy is in the cache, the access returns successfully
  - *Cache-miss*: if the copy is not in the cache (or has been invalidated), the remote entity must be accessed
- Policy
  - Tells which part should be overridden by new copies when the cache gets full

Caching

Distributed file system caching

- Name translations: mapping from paths → inodes
  - Only the first time it is accessed, the directory is parsed
  - Later on, the paths is known to map to a given inode

- Disk blocks: mapping from block address → disk content
  - If two processes on the same machine access the same disk block, the second one will find it in the memory

- Least recently used (LRU) policy
  - When the cache is full, and another access occurs, the least-recently used block or path entry is overridden by the new access entry.
  - Well suited for name translation, and for disk blocks as long as |cache| > |file|

Caching

Distributed file system caching (con't)

- NFSv3: weak consistency
  - Clients check the server if data has not been checked in last 3-30 seconds (tunable)
  - Upon client modification, server is notified but other clients are not until timeout
  - A client can observe a partially updated file

- Andrew File System: stronger consistency (as well as NFSv4)
  - Clients register callbacks
  - Server is stateful, it keeps track of what clients are accessing
  - The server can inform the client as soon as a modification occurs using callbacks
  - When a client opens a file, it gets a snapshot either before or after a concurrent modification
  - No polling, better scalability with the number of clients (no update ⇒ no communication)

- The data is up-to-date (within few seconds)

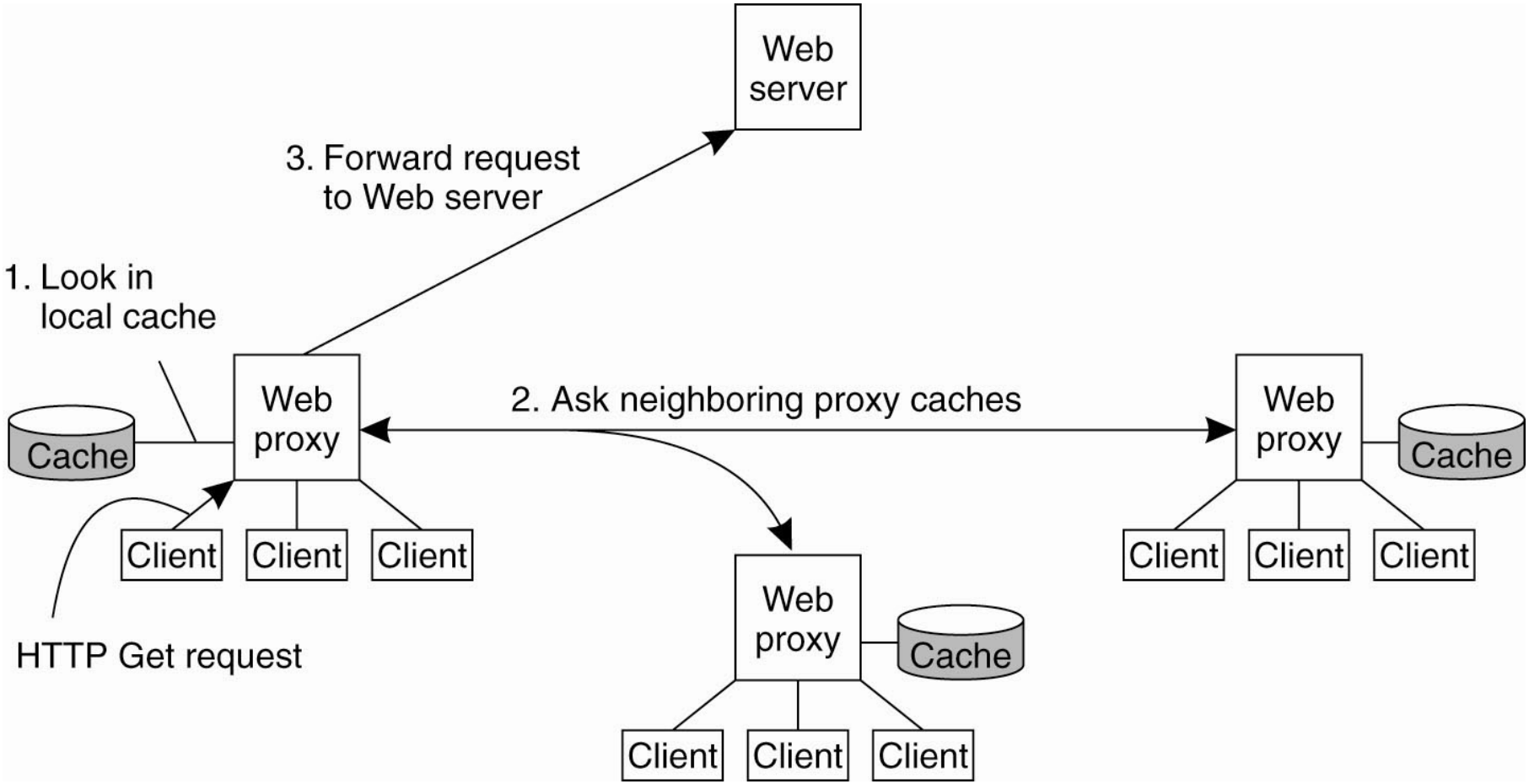
Caching

Web caching

- World wide web:
  - Originally a graphical front-end to RPC protocol
  - Initially: no caching
  - Generally does not have to provide up-to-date information (within seconds): daily accuracy for a news website could be sufficient
- Browser provides client-side cache
  - The browser caches webpage for faster re-accesses
  - The update might not be apparent instantaneously (upon page reload)
  - The browser cache can generally be flushed or disabled in the browser settings
  - Time-to-live (TTL) is given by the HTTP "expires" header from server
  - Server refresh – HTTP "meta refresh tag" causes client periodic polling
- Proxy provides network cache
  - Decreases the load on the server
  - Hierarchical cache: can have multiple network caches between client and web server
- Challenges
  - Efficient for static traffic (~40% of the traffic)
  - Inefficient for dynamic traffic (e.g., multimedia files)

Caching

Web caching (con't)

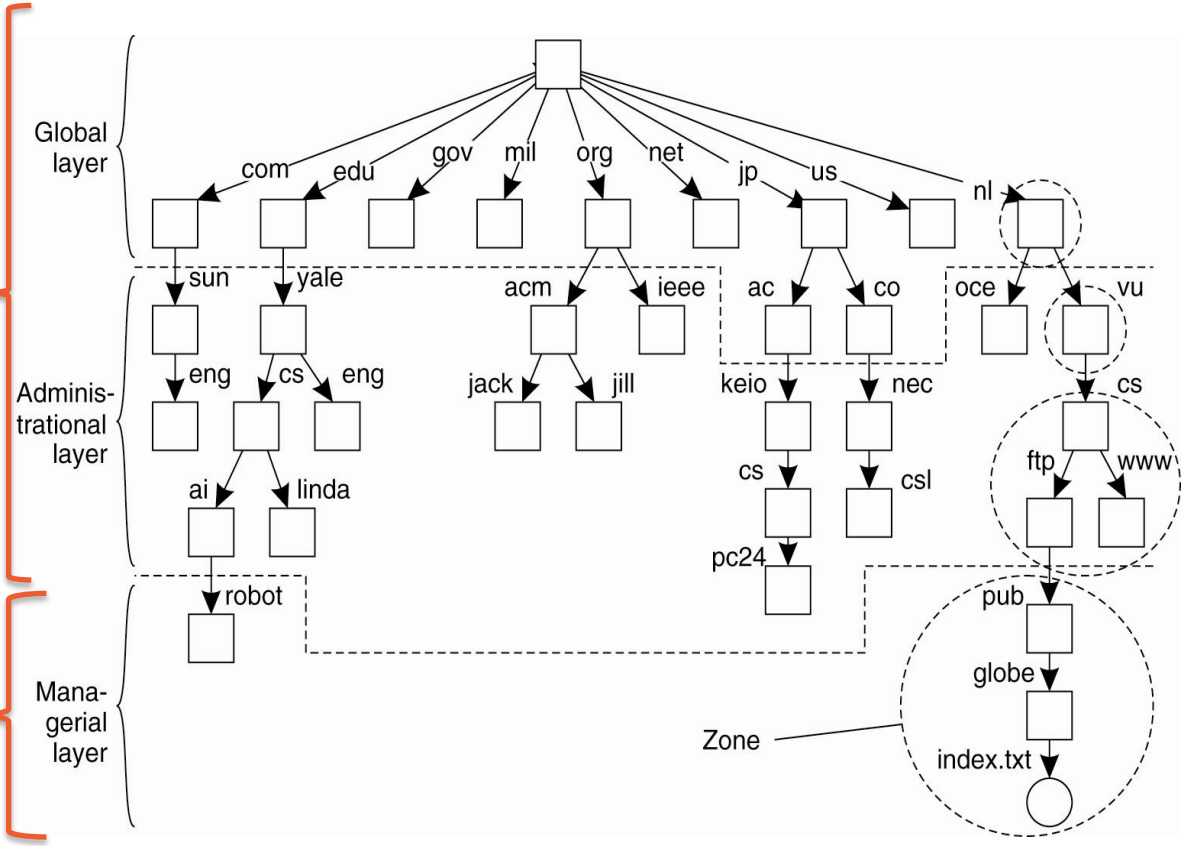


Domain Name System

THE UNIVERSITY OF SYDNEY

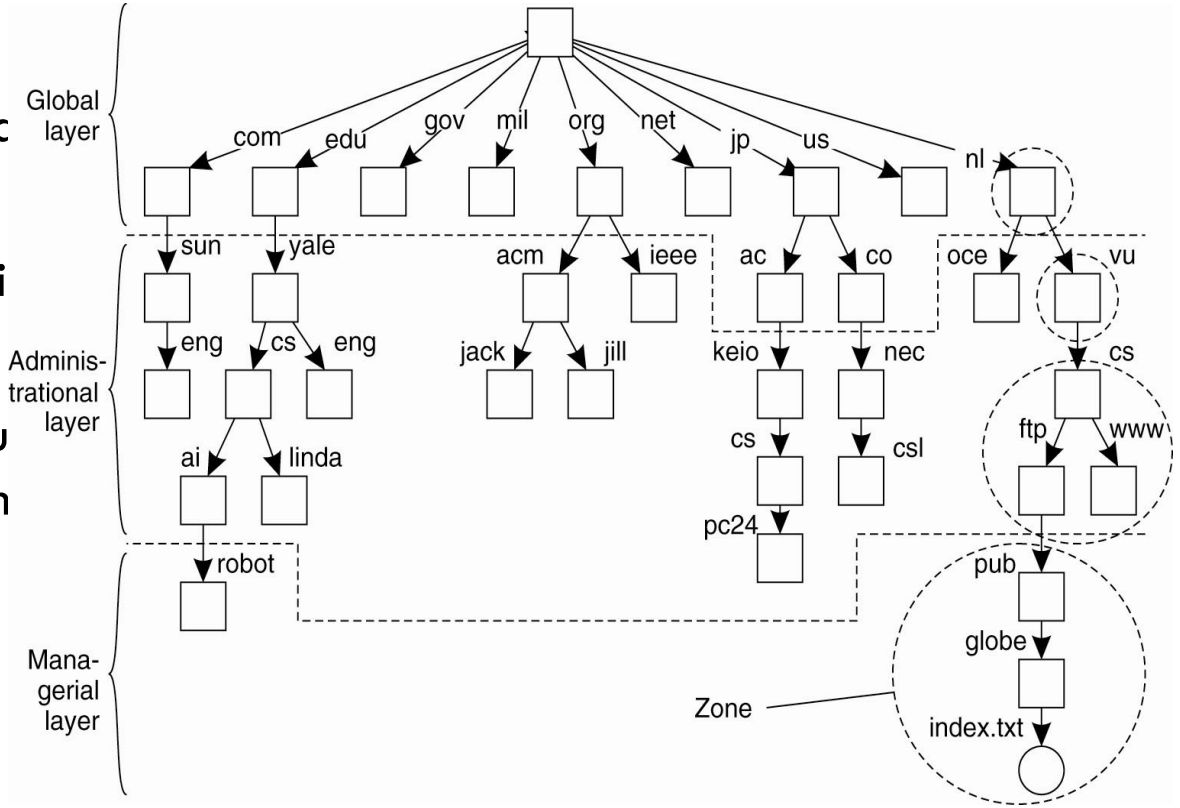
Domain name system

- An example partitioning of the DNS name space, including internet-accessible files, into three layers of directory nodes
- Global layer: highest level nodes, root and its children, nodes do not change often (server response slow high throughput)
- Administrational layer: organizations, nodes change sometimes (server more responsive and non-faulty)
- Managerial layer: hosts of the local network, change often (highly efficiency but failure-prone)



Domain name system

- The domain space is represented as a tree
  - The root node has no incoming edge
  - A subtree is called a *domain* with incoming ec of same name
  - A path to its root node i *domain name*
  - A node contains a resou records with information email of administrator, host IP address, zone information…



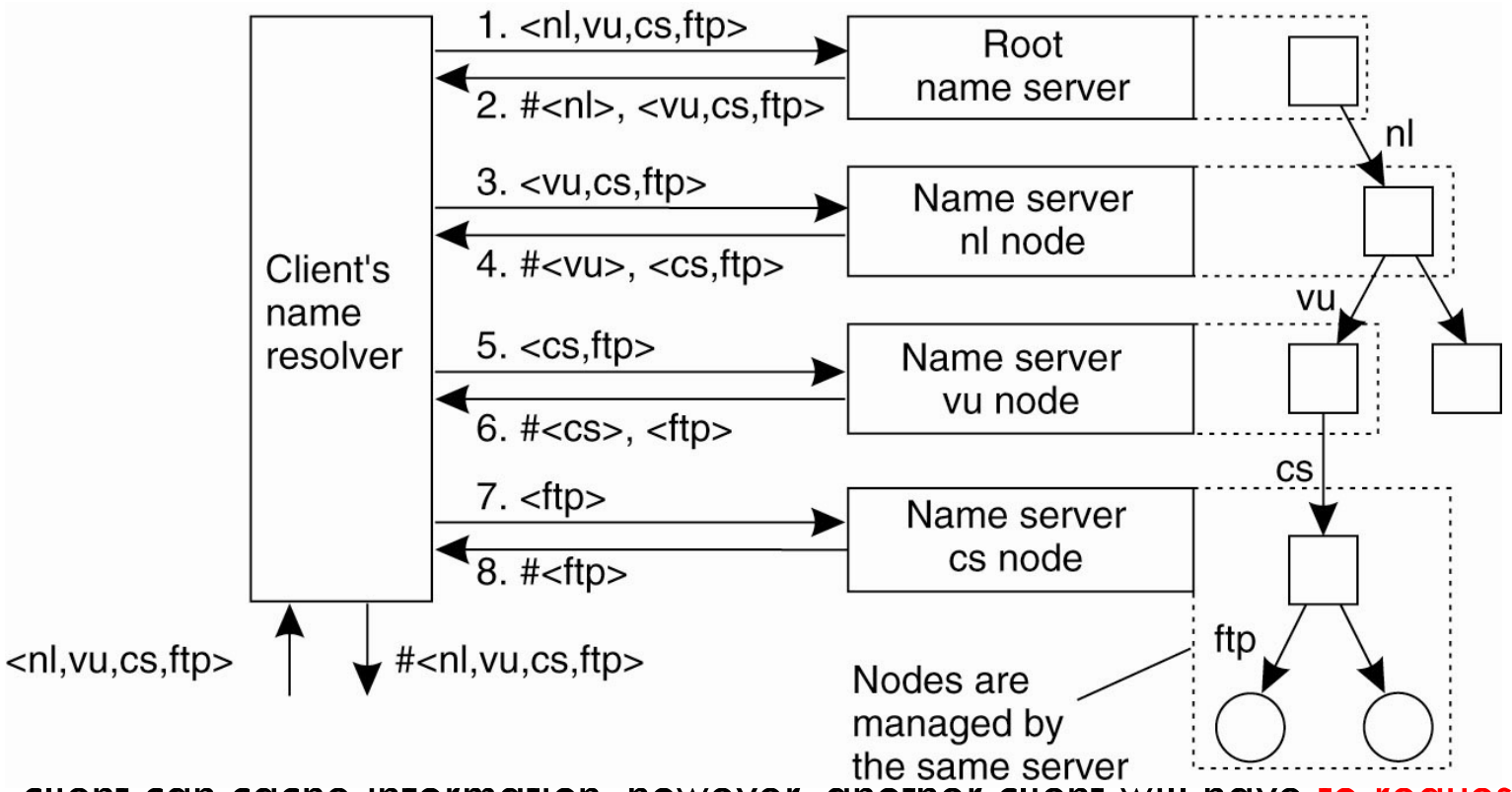
Domain name system

- 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

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

Domain name system

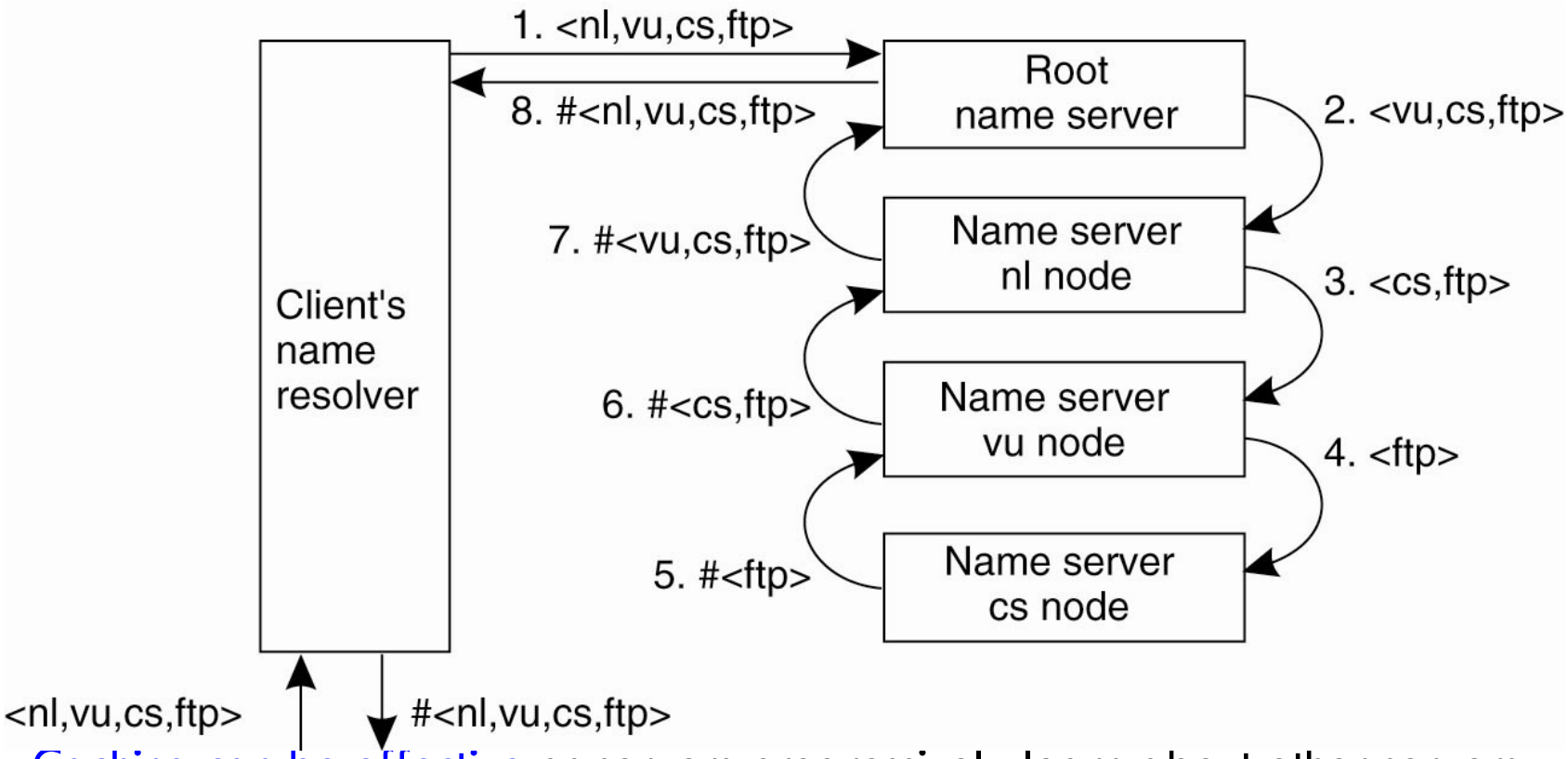
- The principle of iterative name resolution



- A client can cache information, however, another client will have to request again

Domain name system

- The principle of recursive name resolution



- Caching can be effective as servers progressively learn about other servers

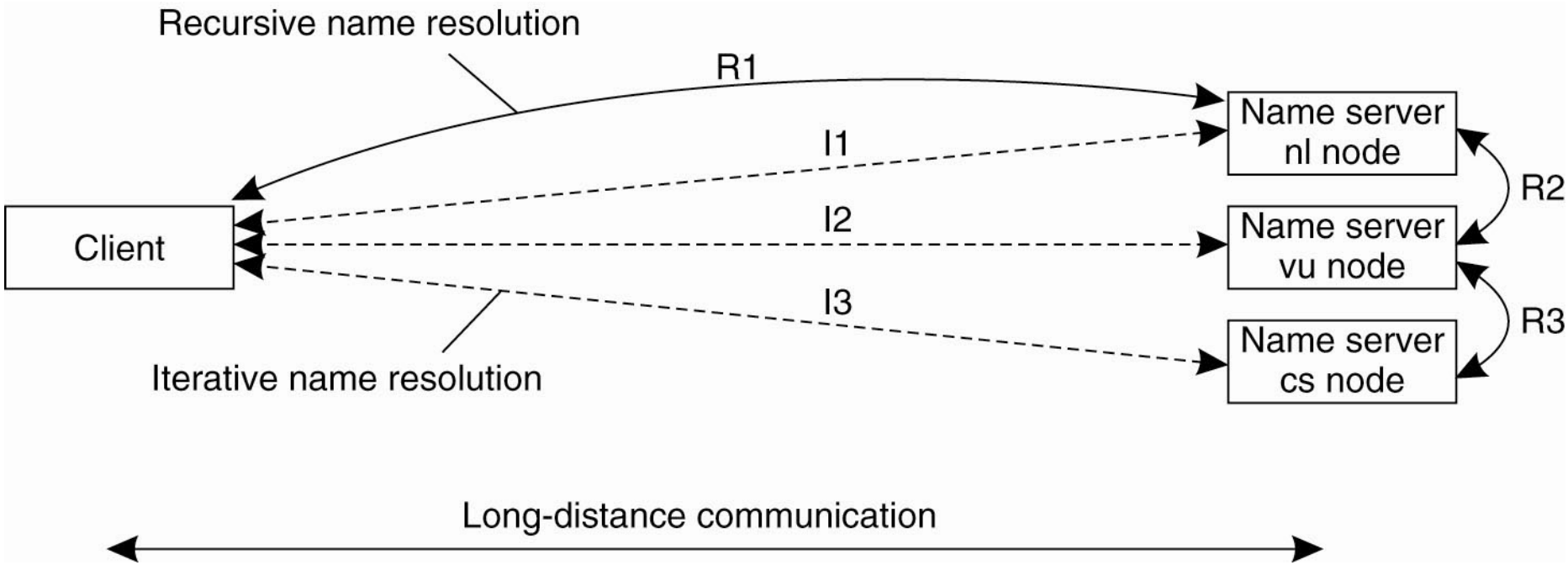
Domain name system

- Recursive name resolution of *<nl, vu, cs, ftp>*
- Name servers cache intermediate results for subsequent lookups

Server for node	Should resolve	Looks up	Passes to child	Receives and caches	Returns to requester
cs	<ftp>	#<ftp>	—	—	#<ftp>
vu	<cs,ftp>	#<cs>	<ftp>	#<ftp>	#<cs> #<cs, ftp>
nl	<vu,cs,ftp>	#<vu>	<cs,ftp>	#<cs> #<cs,ftp>	#<vu> #<vu,cs> #<vu,cs,ftp>
root	<nl,vu,cs,ftp>	#<nl>	<vu,cs,ftp>	#<vu> #<vu,cs> #<vu,cs,ftp>	#<nl> #<nl,vu> #<nl,vu,cs> #<nl,vu,cs,ftp>

Domain name system

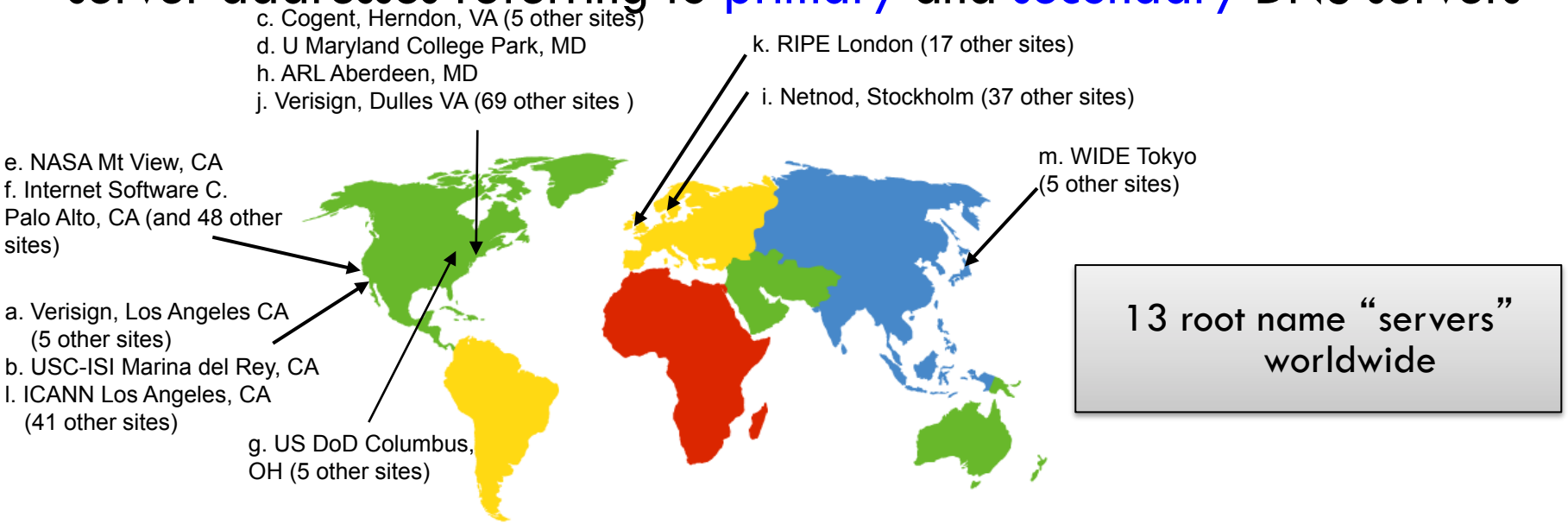
- The comparison between recursive and iterative name resolution with respect to communication costs



Domain name system

Fault-tolerance: replication of DNS server

- Each zone is implemented by a name server replicated for availability
- The network preference of your machine often takes two DNS server addresses referring to primary and secondary DNS servers

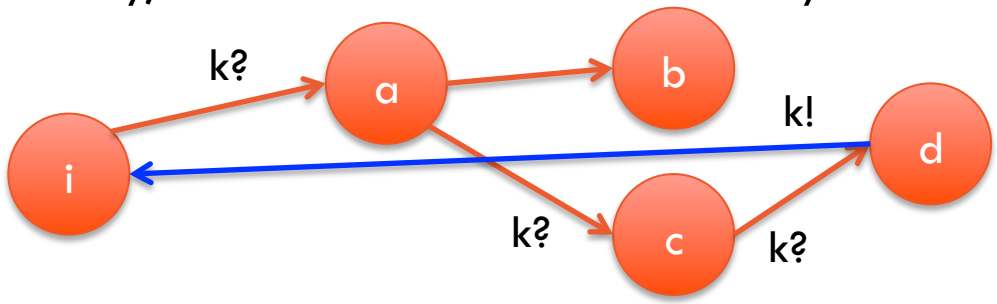


Distributed Hash Table

Distributed hash table

Goal: to provide a distributed lookup service returning the host of a resource

- Given a key, find the resource:
  - keys are assigned to resources similar to names bound to objects
  - Locating a key is similar to resolving a name
  - Actually, the resource is hashed into a key and the DHT returns the host of that key

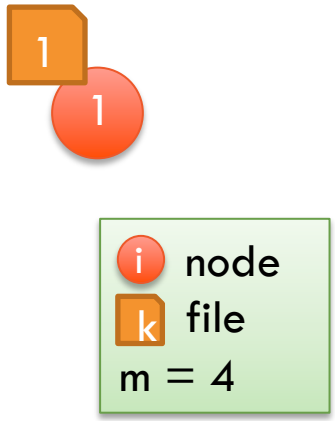


- Distributed lookup:
  - Iterative lookup: a node requested to look up a key returns to the requester the address of the next node found
  - Recursive lookup: a node requested to look up a key forward the lookup request to the next node

Distributed hash table

Chord (con't)

- Resources (files) and hosts:
  - Each node has an *id*, each file has a *key*, both coded on m bits
  - A file of key k is stored on the node whose id i = min { node.id ≥ k }

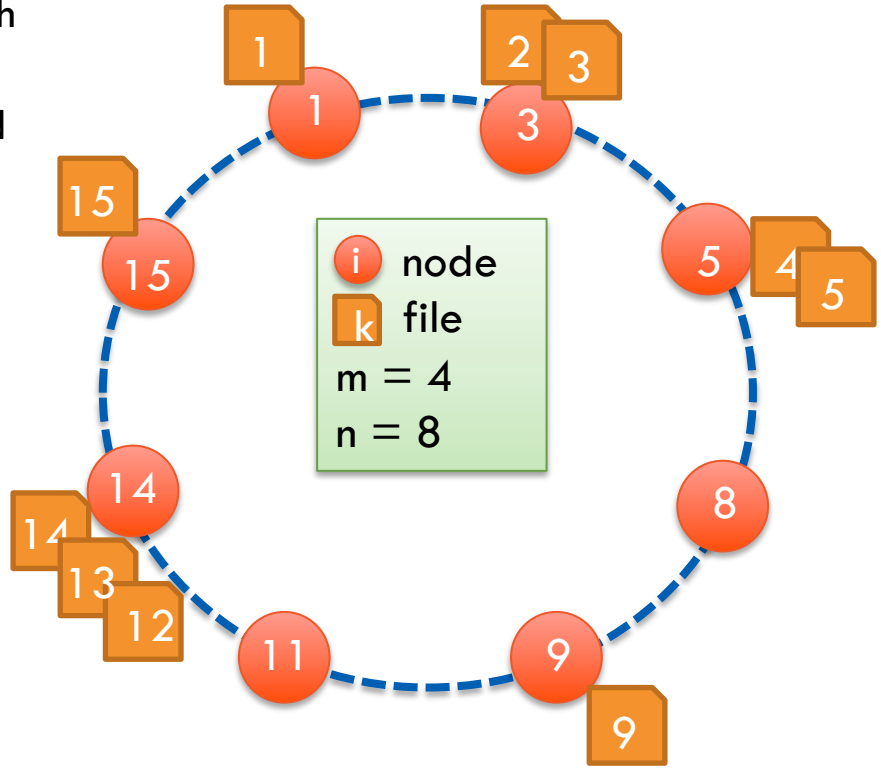


Distributed hash table

Chord (con't)

- Resources (files) and hosts:
  - Each node has an *id*, each file has a *key*, both coded on m bits
  - A file of key k is stored on the node whose id i = min { node.id ≥ k }

- Ring structure:
  - n nodes are organized into a ring of $2^m > n$ slots with ascending id in the clockwise direction

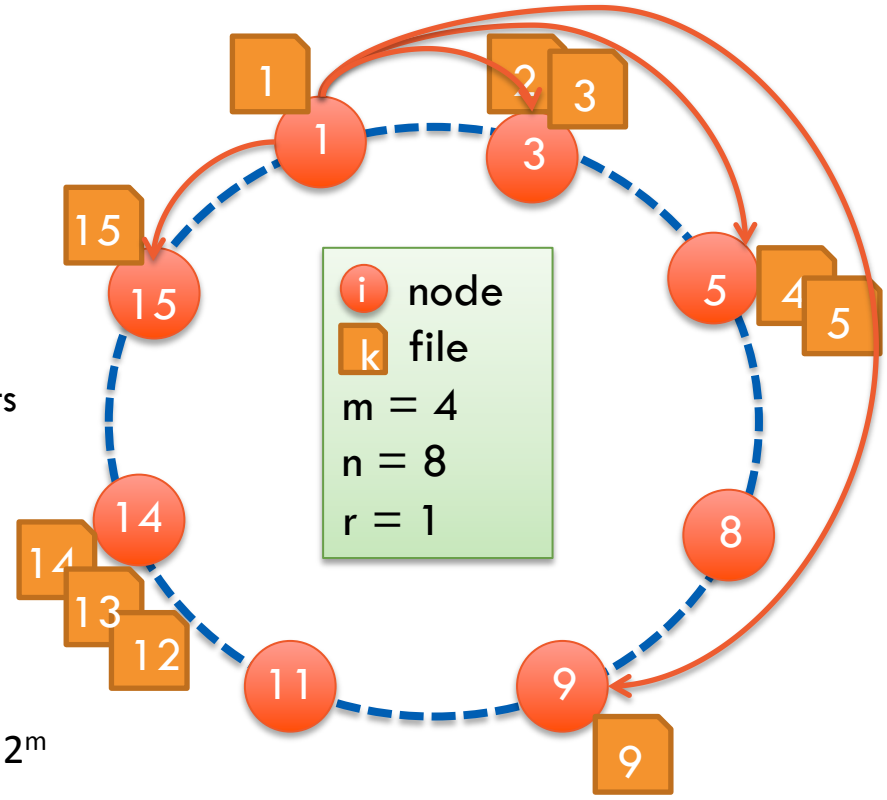


Distributed hash table

Chord (con't)

- Resources (files) and hosts:
  - Each node has an *id*, each file has a *key*, both coded on m bits
  - A file of key k is stored on the node whose id i = min { node.id ≥ k }

- Ring structure:
  - n nodes are organized into a ring of $2^m > n$ slots with ascending id in the clockwise direction
  - Each node j maintains:
    - The id of its r successors on the ring
    - The id of its r predecessors on the ring
    - A finger table of m entries: i^{th} entry with address of node responsible of $j+2^{i-1} \bmod 2^m$



Distributed hash table

Chord (con't)

› Routing: never pass the target

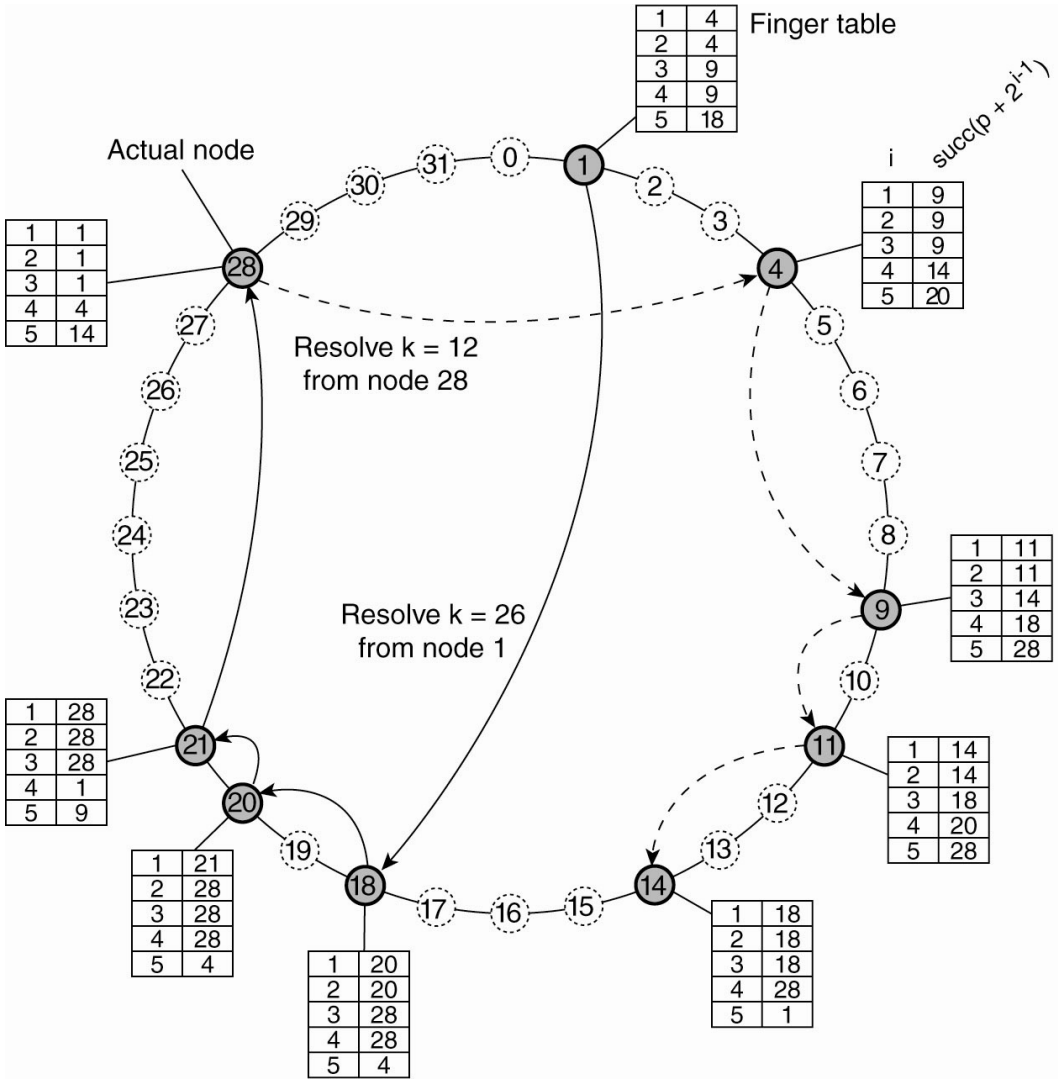
- If key between me and my succ then ask succ
- Otherwise, ask the node w/ the closest lower id than the key

– Resolving key 26 from node 1

- 1's finger table indicates to visit the closest lower id 18 to route to 26
- 18's finger table indicates to visit 20
- 20 tells to go to 21
- 21 tells to go to its succ, 28.

– Resolving key 12 from node 28

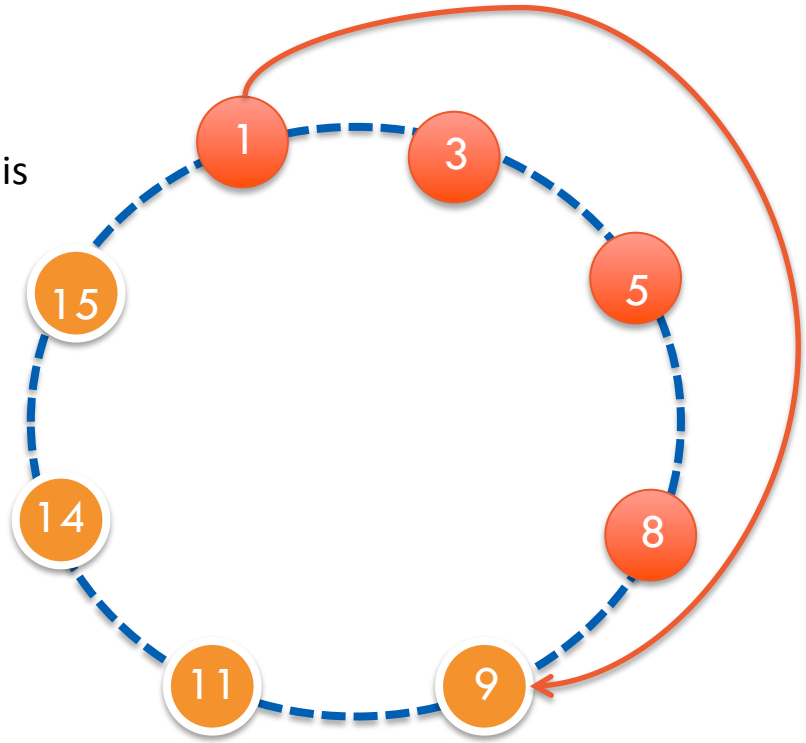
- 28's finger table indicates to visit the closest lower id 4 to route to 12
- 4 tells to go to 9, 9 to 11.
- 11 knows the succ 14 should have it



Distributed hash table

Chord (con't)

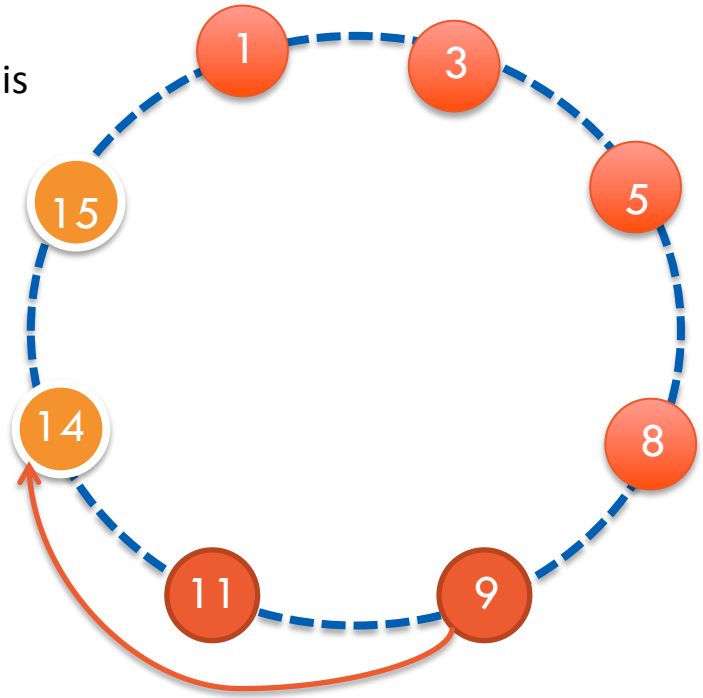
- Complexity:
  - Route of lookup:
    - Assume ids are uniformly distributed in space
    - 1st step: finds the half of the ring in which key is



Distributed hash table

Chord (con't)

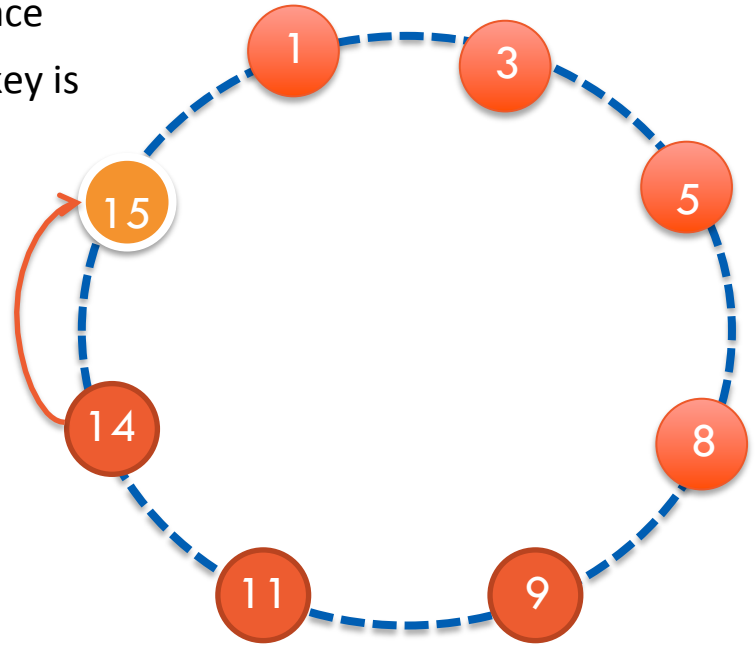
- Complexity:
  - Route of lookup:
    - Assume ids are uniformly distributed in space
    - 1st step: finds the half of the ring in which key is
    - 2nd step: finds the quarter of the ring



Distributed hash table

Chord (con't)

- Complexity:
  - Route of lookup:
    - Assume ids are uniformly distributed in space
    - 1st step: finds the half of the ring in which key is
    - 2nd step: finds the quarter of the ring
    - 3rd step: find the node that hosts the file

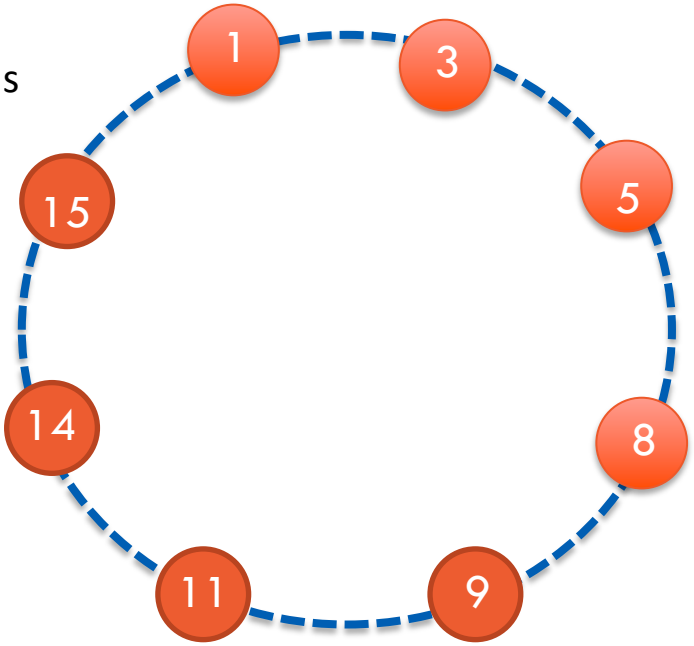


Distributed hash table

Chord (con't)

- Complexity:
  - Route of lookup:
    - Assume ids are uniformly distributed in space
    - 1st step: finds the half of the ring in which key is
    - 2nd step: finds the quarter of the ring
    - 3rd step: find the node that hosts the file
  - Generally speaking:
    - Start with n possible host nodes
    - After step i, only $n/2^i$ possible hosts remain

⇒ O(log n)



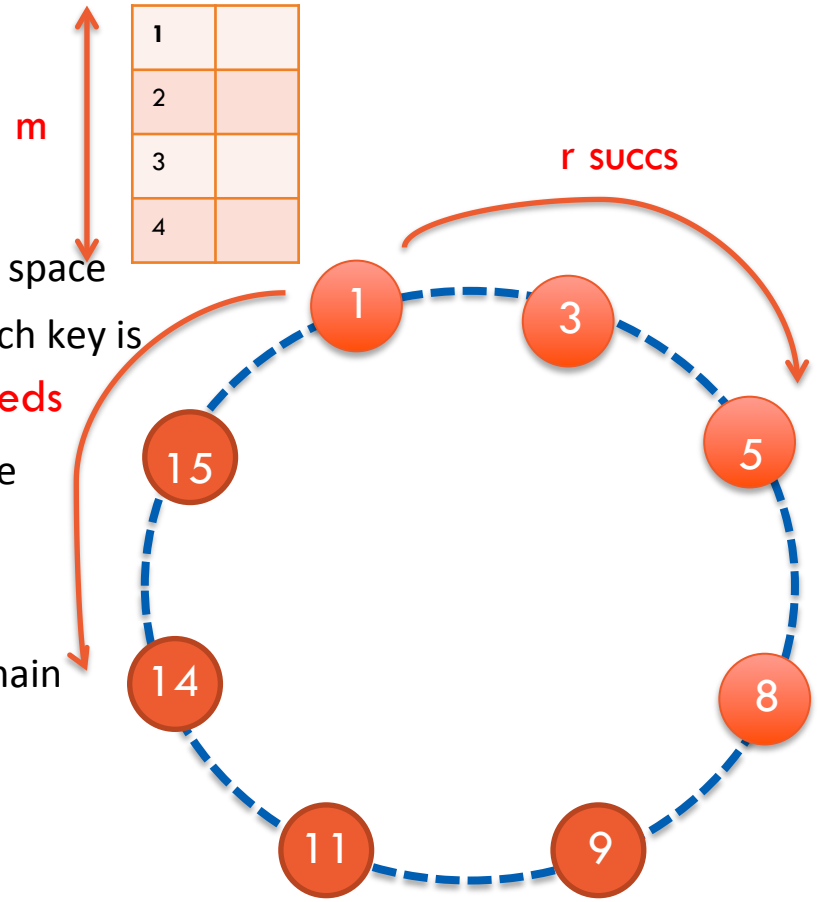
Distributed hash table

Chord (con't)

- Complexity:
  - Route of lookup:
    - Assume ids are uniformly distributed in space
    - 1st step: finds the half of the ring in which key is
    - 2nd step: finds the quarter of the ring
    - 3rd step: find the node that hosts the file
  - Generally speaking:
    - Start with n possible host nodes
    - After step i, only $n/2^i$ possible hosts remain

  ⇒ O(log n)
  - Information maintained per node:
    - size of finger table, m = O(log n) +
    - #predecessors and successors, 2r = O(1)

  ⇒ O(log n)

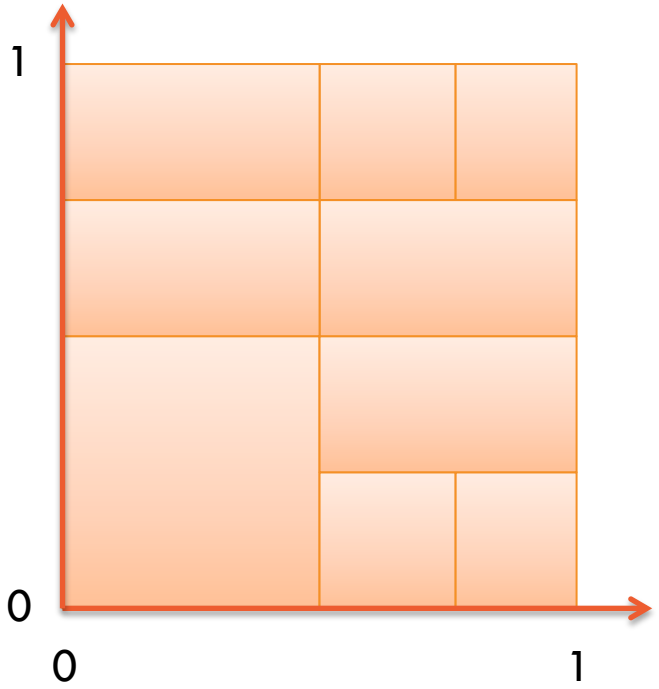


Distributed hash table

Content Addressable Network (CAN)

› d-dimensional structure:

- Each node has a zone in the (0,1)x(0,1) 2-dimensional space (d=2)
- Each node is responsible of the files whose keys fall in its region

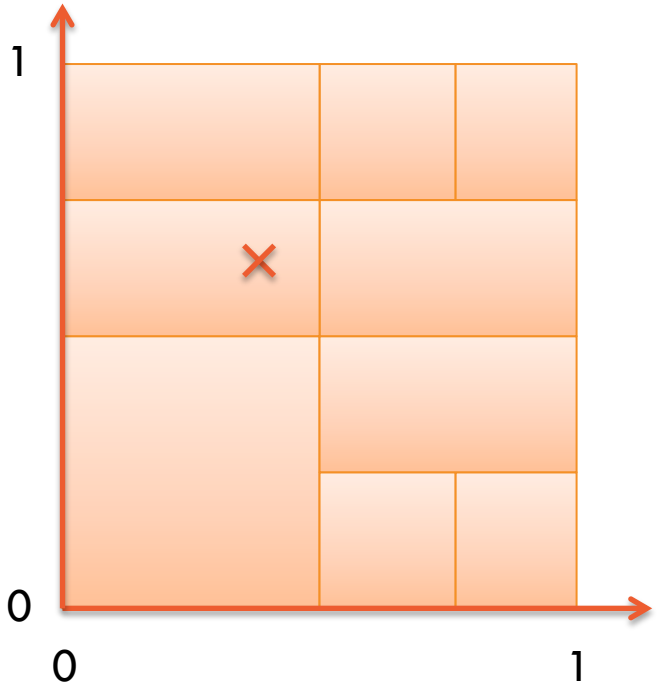


Distributed hash table

CAN (con't)

› d-dimensional structure:

- Each node has a zone in the (0,1)x(0,1) 2-dimensional space (d=2)
- Each node is responsible of the files whose keys fall in its region
- Each new node chooses a random coordinate, lookups the node responsible for this point, and becomes responsible of half of its zone

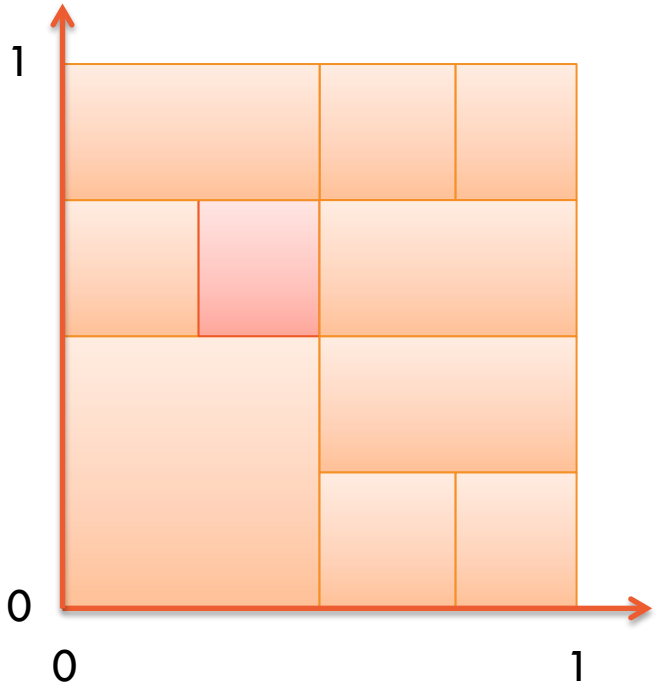


Distributed hash table

CAN (con't)

› d-dimensional structure:

- Each node has a zone in the (0,1)x(0,1) 2-dimensional space (d=2)
- Each node is responsible of the files whose keys fall in its region
- Each new node chooses a random coordinate, lookups the node responsible for this point, and becomes responsible of half of its zone

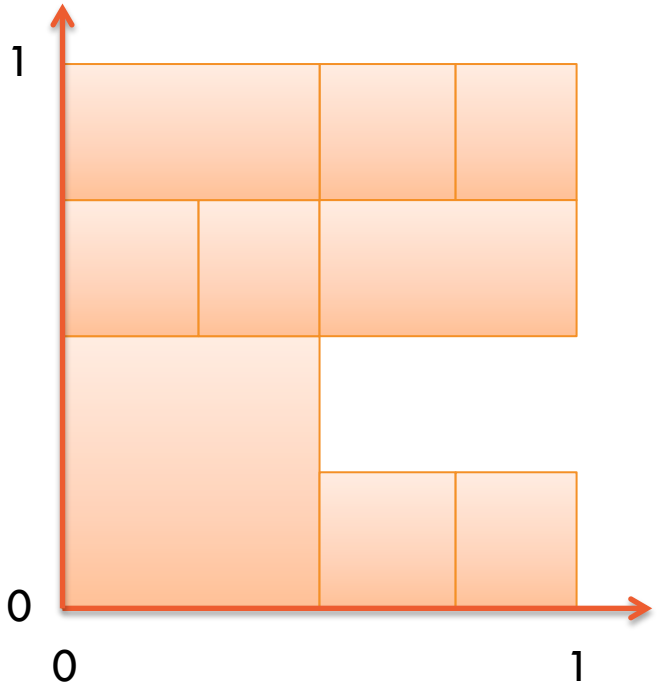


Distributed hash table

CAN (con't)

› d-dimensional structure:

- Each node has a zone in the (0,1)x(0,1) 2-dimensional space (d=2)
- Each node is responsible of the files whose keys fall in its region
- Each new node chooses a random coordinate, lookups the node responsible for this point, and becomes responsible of half of its zone
- A node takes care of being responsible of the zone of the leaving node

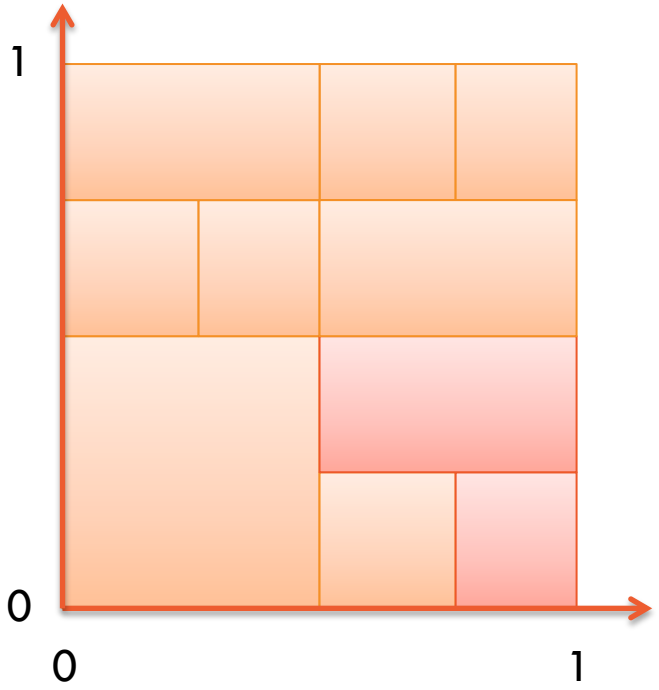


Distributed hash table

CAN (con't)

› d-dimensional structure:

- Each node has a zone in the (0,1)x(0,1) 2-dimensional space (d=2)
- Each node is responsible of the files whose keys fall in its region
- Each new node chooses a random coordinate, lookups the node responsible for this point, and becomes responsible of half of its zone
- A node takes care of being responsible of the zone of the leaving node

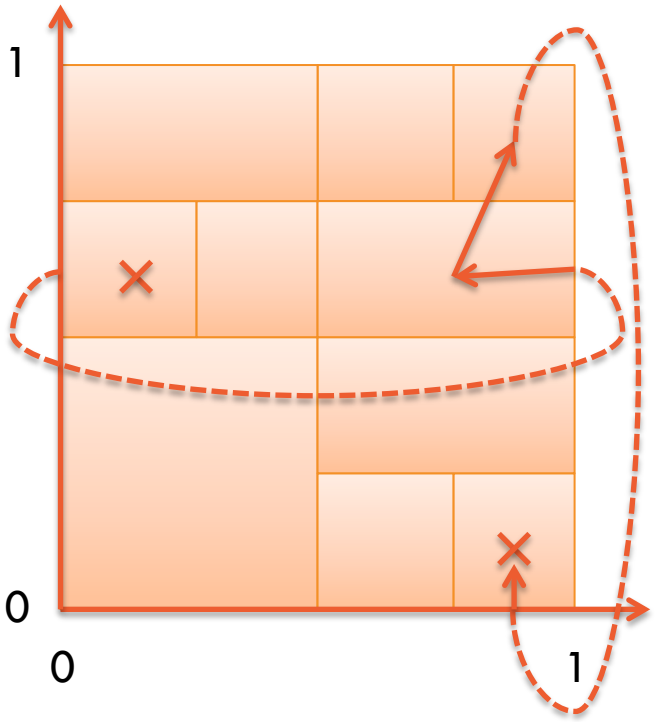


Distributed hash table

CAN (con't)

› Routing: through abutting zones

- Each node maintains the addresses of its neighbours, that are nodes responsible of zones abutting his
- The structure is actually a torus, so that topmost zones are considered abutting bottommost zones
- To route towards the destination, each node chooses its closest neighbour to the destination in terms of euclidean distances



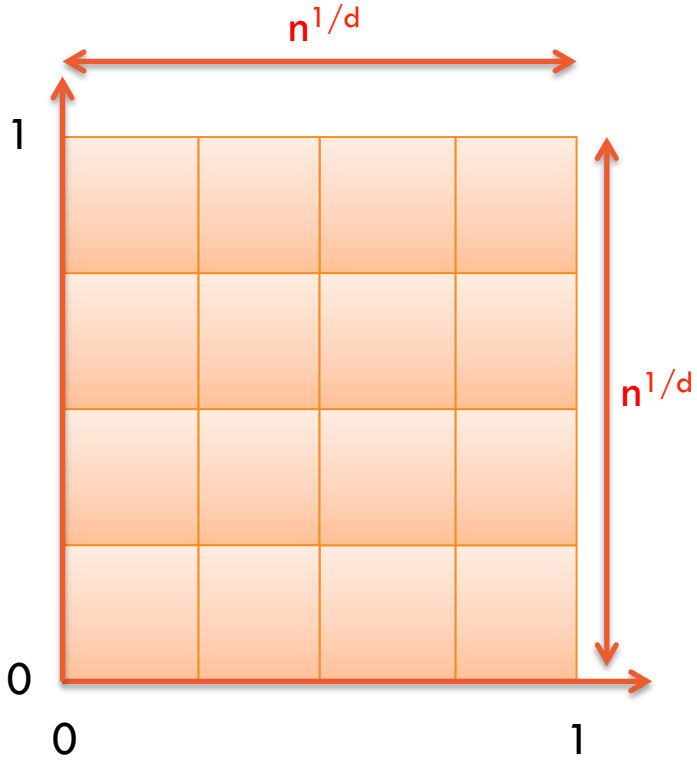
Distributed hash table

CAN (con't)

› Complexity

- Route of lookup (for d=2)
  - Assume id uniformly distributed in the range
  - Worst-case: has to route through half of the height of the zone +
  - half of the width of the zone

  ⇒ $O(n^{1/2})$
- Generally speaking ⇒ $O(1/2 * dn^{1/d}) = O(dn^{1/d})$



Distributed hash table

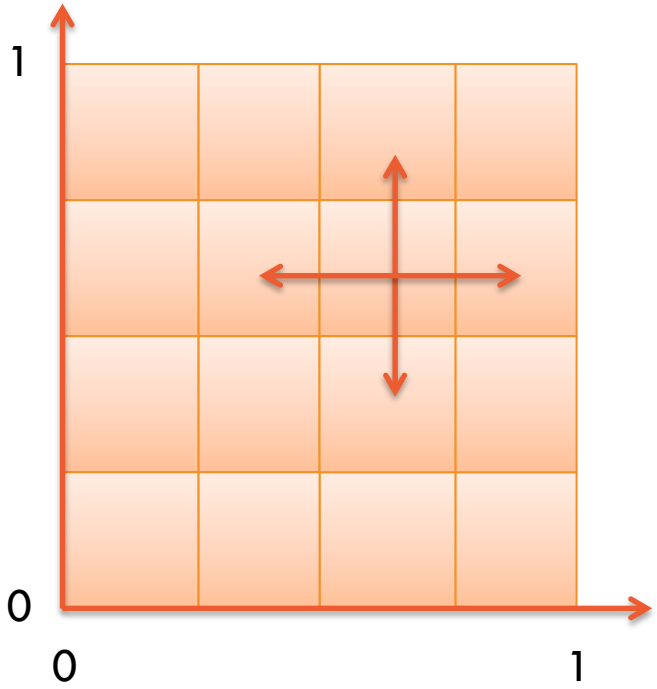
CAN (con't)

› Complexity

- Route of lookup (for d=2)
  - Assume id uniformly distributed in the range
  - Worst-case: has to route through half of the height of the zone +
  - half of the width of the zone

  ⇒ $O(n^{1/2})$
- Generally speaking ⇒ $O(1/2 * dn^{1/d}) = O(dn^{1/d})$
- Information maintained per node:
  - One in each direction (top, bottom, left, right)

  ⇒ O(d)



Distributed hash table

General characteristics

- Fault tolerance:
  - Fully distributed, no central point of control
  - Each node knows only a small amount of neighbours:
    - In CAN: O(d) constant independent from n
    - In Chord: O(m) = O(log n) logarithmic in n

⇒ Small number of changes needed to adapt to churn (i.e., node leave/join)

- Scalability to a large number n of participants:
  - Each node knows the right nodes to route to:
    - In CAN: it takes $O(dn^{1/d})$ to lookup a key
    - In Chord: it takes O(log n) to lookup a key

⇒ The routes are short compared to n (if $n=2^m$, then routes length can be O(m))

Conclusion

- Naming hierarchy helps distributing name resolution
  - This allows DNS to scale to a very large number of clients
- Naming helps masking lookups
  - This allows NFS to represent local and remote directories identically
- Caching helps lowering communication
  - Static information are cached (stored locally) for efficient retrieval
- Fully distributed naming solution tolerate many failures
  - By maintaining a small amount of information per node
  - By leveraging this information to route rapidly