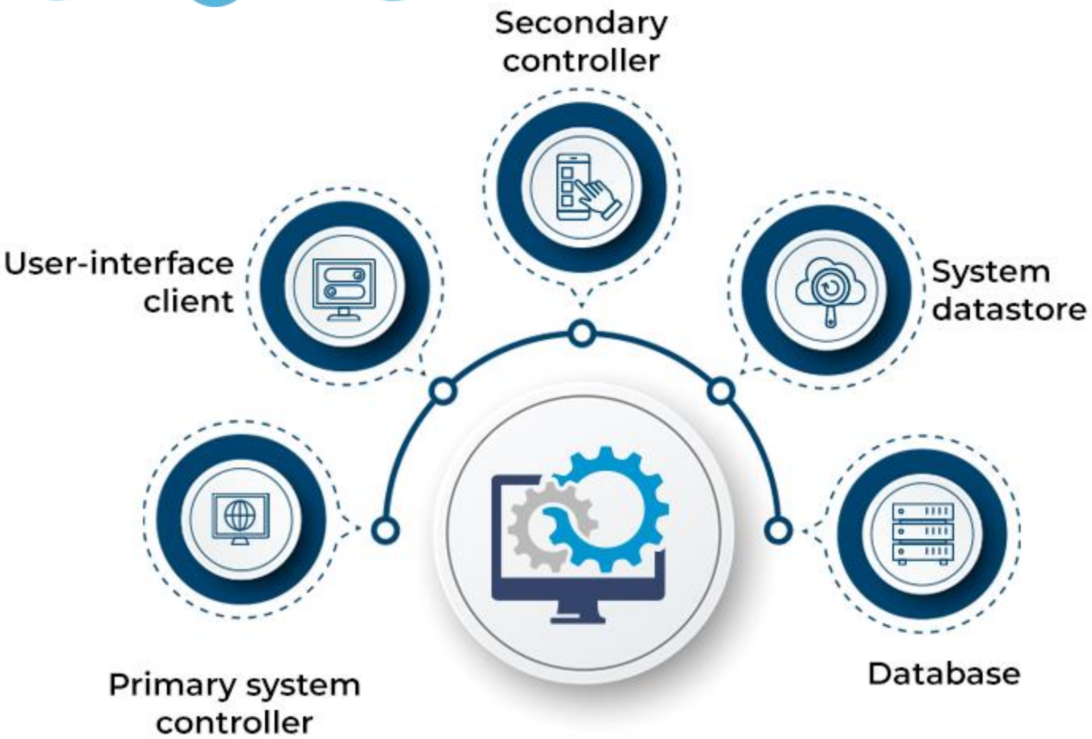
Sistem Terdistribusi

IF2222



05: Penamaan

2022-23 Ganjil

Teknik Informatika
Universitas Trunojoyo Madura

Sistem Terdistribusi 2022

1. Mengenal Sistem Terdistribusi
2. Review Jaringan Komputer (layer 2, 3, dan 4)
3. Arsitektur Sistem Terdistribusi
4. *Remote Procedure Calls* (RPC)
5. **Layanan Penamaan**
6. Sinkronisasi Data (2 pekan)
7. *Message Passing Interface* (MPI)
8. Contoh Arsitektur: Hadoop, Pregel, Blockchain
9. Teknik *Caching*
10. Teknik Replikasi Data (2 pekan)
11. Basis Data Terdistribusi
12. Toleransi Kegagalan

Capaian Pembelajaran

Kuliah ini bertujuan memberikan pemahaman mendalam dan pengalaman langsung tentang:

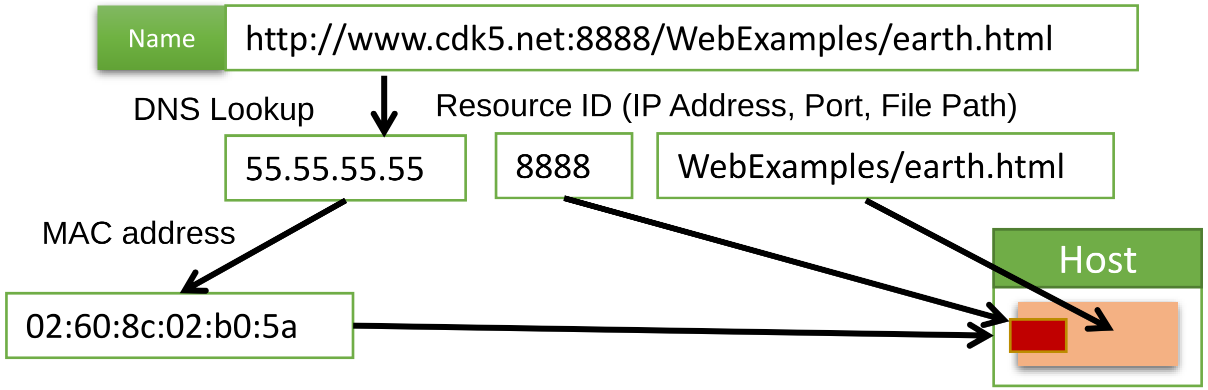
- Prinsip yang menjadi basis dari sistem terdistribusi
- Prinsip untuk optimalisasi sistem terdistribusi
- Model pemrograman sistem terdistribusi dan mesin analitika
- Bagaimana sistem terdistribusi modern memenuhi tuntutan aplikasi terdistribusi kontemporer

Today…

- Last Session:
  - *Remote Procedure Calls* (RPC)
- Today’s Session:
  - Layanan Penamaan (*naming*)
- Announcements:

Naming

- Names are used to uniquely identify entities in distributed systems
  - Entities may be processes, remote objects, newsgroups, etc.,
- Names are mapped to entities' locations using *name resolution*
- An example of name resolution:



Names, Addresses, and Identifiers

- An entity can be identified by three types of references
  a) Name
     - A name is a set of bits or characters that references an entity
     - Names can be human-friendly (or not)
  b) Address
     - Every entity resides on an access point, and access point has an address
     - Addresses may be location-dependent (or not)
     - E.g., IP Address + Port
  c) Identifier
     - Identifiers are names that *uniquely* identify entities
     - A *true identifier* is a name with the following properties:
       - An 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 (i.e. it is never reused)

Naming Systems

- A *naming system* is simply a middleware that assists in name resolution

- Naming systems can be classified into three classes, based on the type of names used:
  a. Flat naming
  b. Structured naming
  c. Attribute-based naming

Classes of Naming

- Flat naming
- Structured naming
- Attribute-based naming

Flat Naming

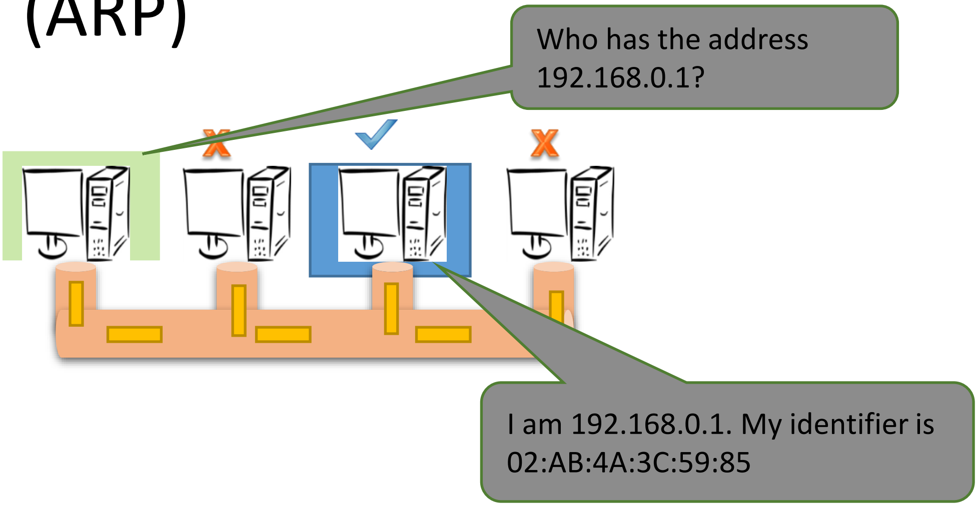
- In flat naming, identifiers are simply random bits of strings (known as *unstructured* or flat names)

- A flat name does not contain any information on how to locate an entity

- We will study four types of name resolution mechanisms for flat names:
    1. Broadcasting
    2. Forwarding pointers
    3. Home-based approaches
    4. Distributed Hash Tables (DHTs)

1. Broadcasting

- Approach: Broadcast the name/address to the whole network; the entity associated with the name responds with its current identifier
- Example: Address Resolution Protocol (ARP)
  - Resolve an IP address to a MAC address
  - In this system,
    - IP address is the *address* of the entity
    - MAC address is the *identifier* of the access point



- Challenges:
  - Not scalable in large networks
    - This technique leads to flooding the network with broadcast messages
  - Requires all entities to *listen* (or *snoop*) to all requests

2. Forwarding Pointers

- Forwarding pointers enable locating *mobile* entities
  - Mobile entities move from one access point to another

- When an entity moves from location A to location B, it leaves behind (at A) a reference to its new location at B

- Name resolution mechanism:
  - Follow the *chain of pointers* to reach the entity
  - Update the entity's reference when the present location is found

- Challenges:
  - Long chains lead to longer resolution delays
  - Long chains are prone to failures due to broken links

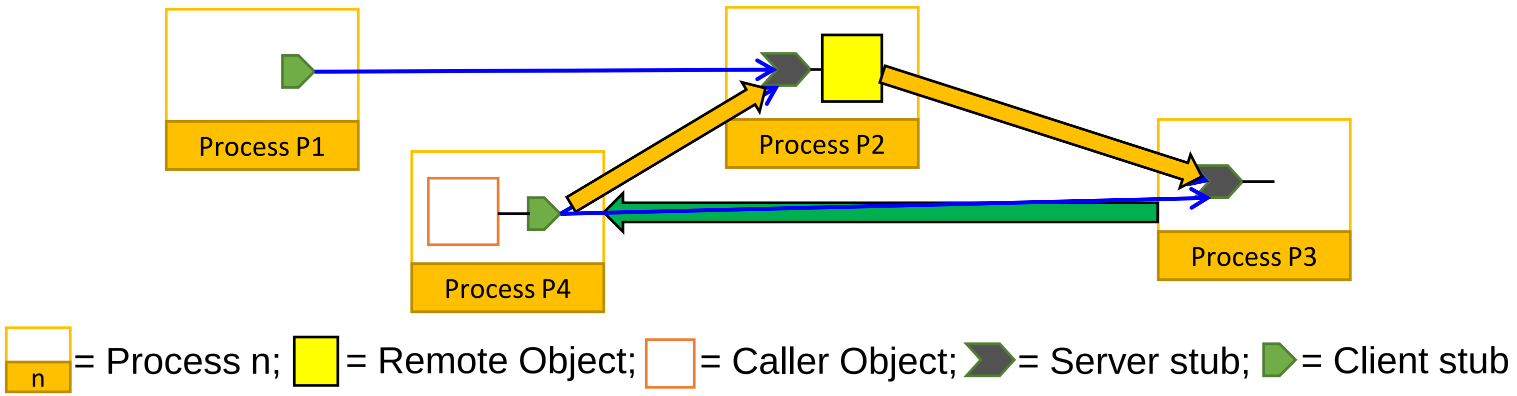
Forwarding Pointers – An Example

- Stub-Scion Pair (SSP) chains implement remote invocations for mobile entities using *forwarding pointers*
  - Server stub is referred to as *Scion* in the original paper
- Each forwarding pointer is implemented as a pair:

```
(client stub, server stub)
```

  - The server stub contains a local reference to the actual object or a local reference to another client stub

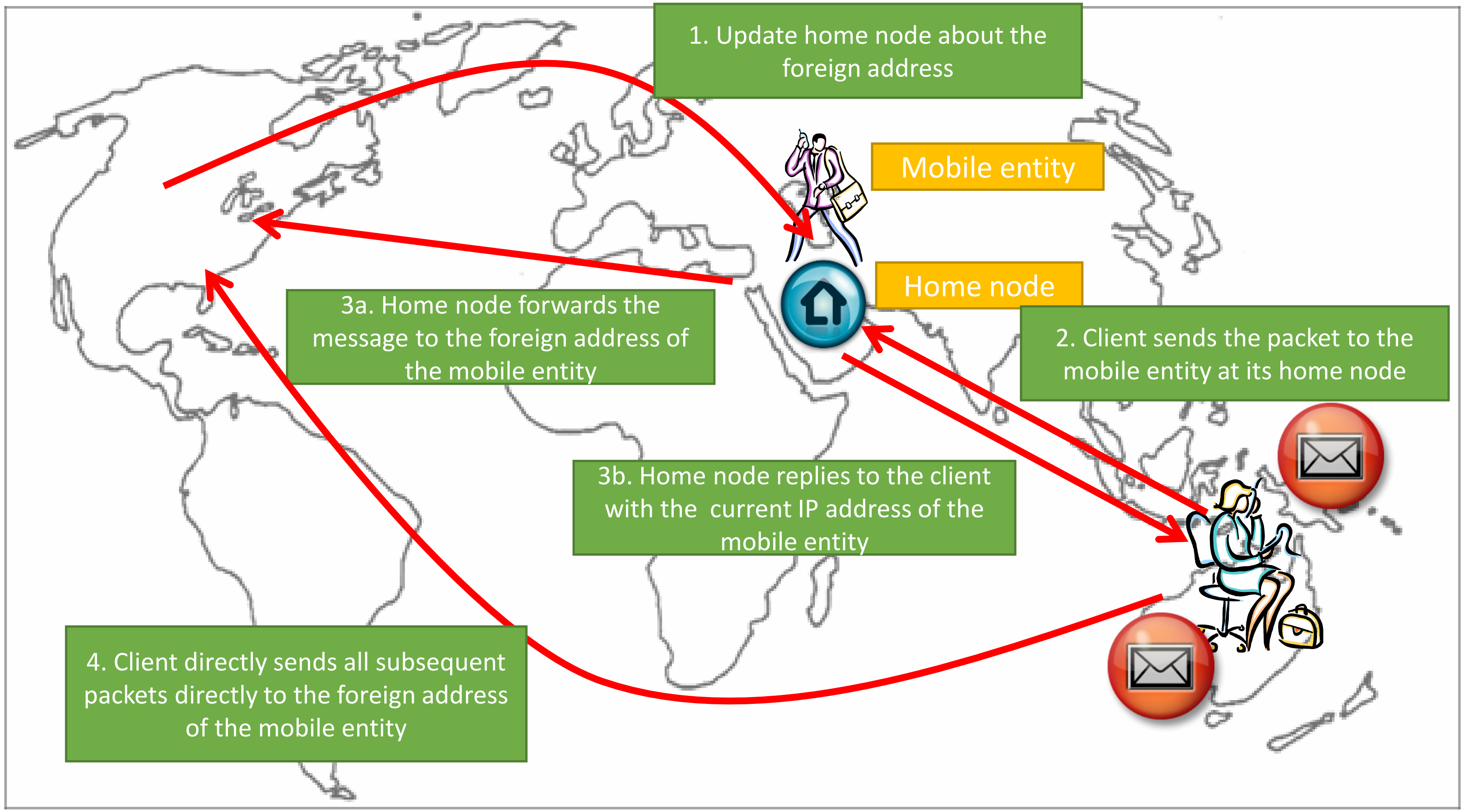
- When object moves from A (e.g., P2) to B (e.g., P3),
  - It leaves a client stub at A (i.e., P2)
  - It installs a server stub at B (i.e., P3)



3. Home-Based Approaches

- Each entity is assigned a home node
  - The home node is typically *static* (has fixed access point and address)
  - It keeps track of the *current* address of the entity
- Entity-home interaction:
  - Entity’s home address is registered at a naming service
  - The entity updates the home about its current address (foreign address) whenever it moves
- Name resolution:
  - Client contacts the home to obtain the foreign address
  - Client then contacts the entity at the foreign location

3. Home-Based Approaches – An Example



3. Home-Based Approaches – Challenges

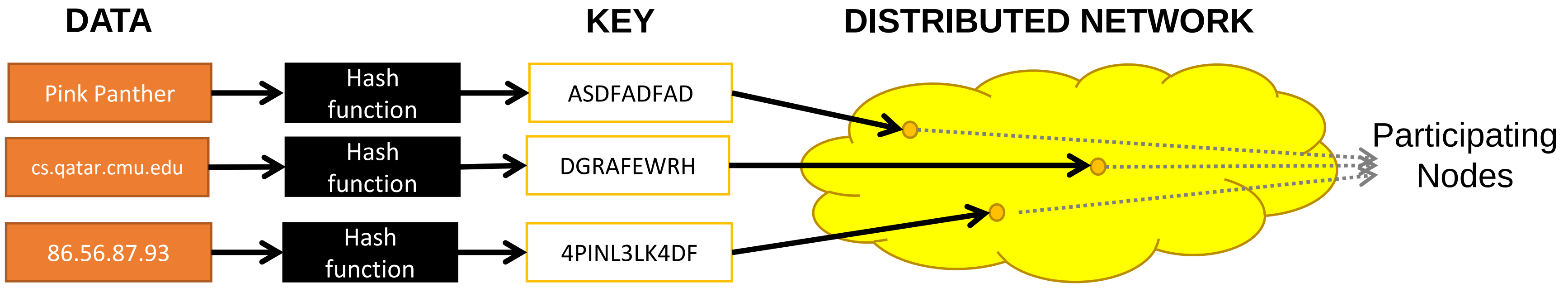
- The static home address is permanent for an entity's lifetime
  - If the entity permanently moves, then a *simple* home-based approach incurs higher communication overhead

- Connection set-up overheads due to communication between the client and the home can be excessive
  - Consider the scenario where the clients are nearer to the mobile entity than the home entity

4. Distributed Hash Table (DHT)

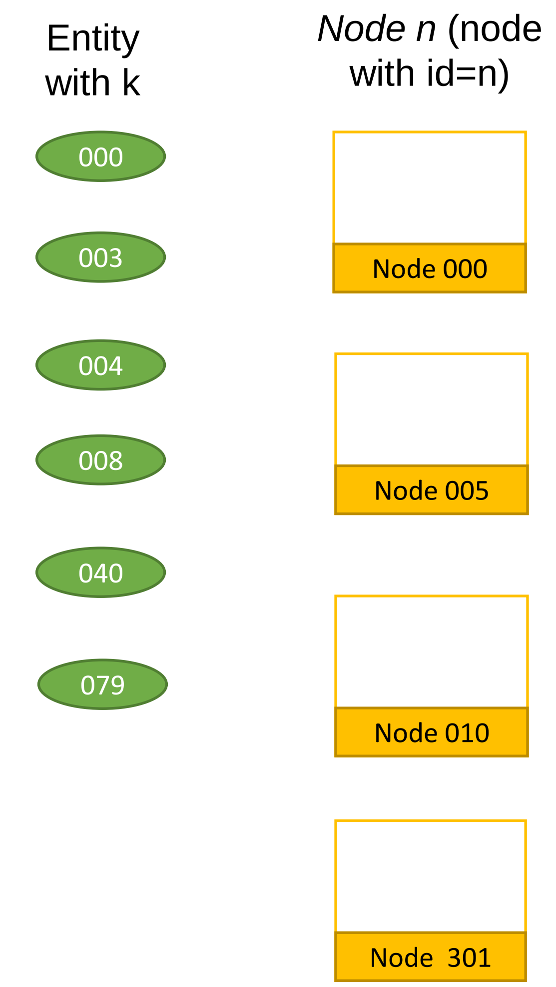
- DHT is a distributed system that provides a lookup service similar to a hash table
  - *(key, value)* pair is stored in the nodes participating in the DHT
  - The responsibility for maintaining the mapping from keys to values is distributed among the nodes
  - Any participating node can serve in retrieving the value for a given key

- We will study a representative DHT known as Chord



Chord

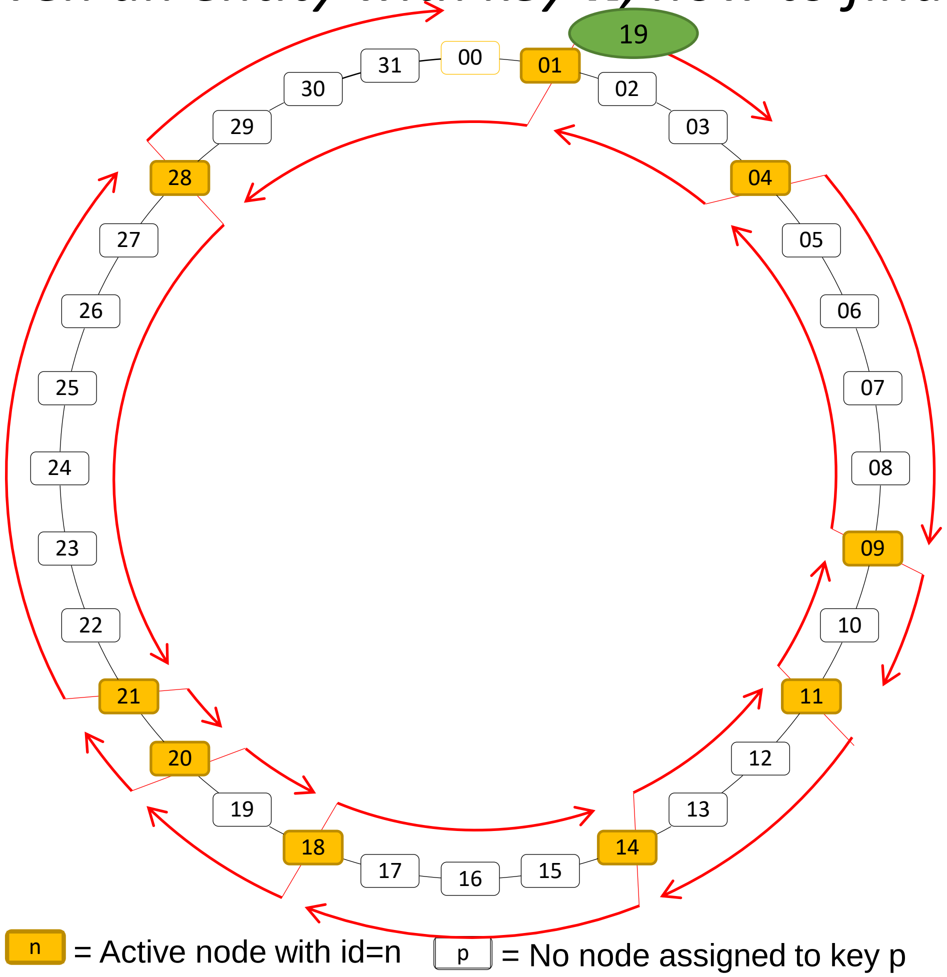
- Chord is a protocol and algorithm for a peer-to-peer distributed hash table
- Chord assigns an *m-bit identifier* (randomly chosen) to each node
  - A node can be contacted through its network address
- Alongside, it maps each entity to a node
  - Entities can be processes, files, etc.,
- Mapping of entities to nodes
  - Each node is responsible for a set of entities
  - An entity with key `k` falls under the jurisdiction of the node with the smallest identifier `id >= k`. This node is known as the *successor of* `k`, and is denoted by `succ(k)`



Map each entity with key `k` to node `succ(k)`

A Naïve Key Resolution Algorithm

- The main issue in DHT is to efficiently resolve a key *k* to the network location of `succ(k)`
  - *Given an entity with key `k`, how to find the node `succ(k)`?*

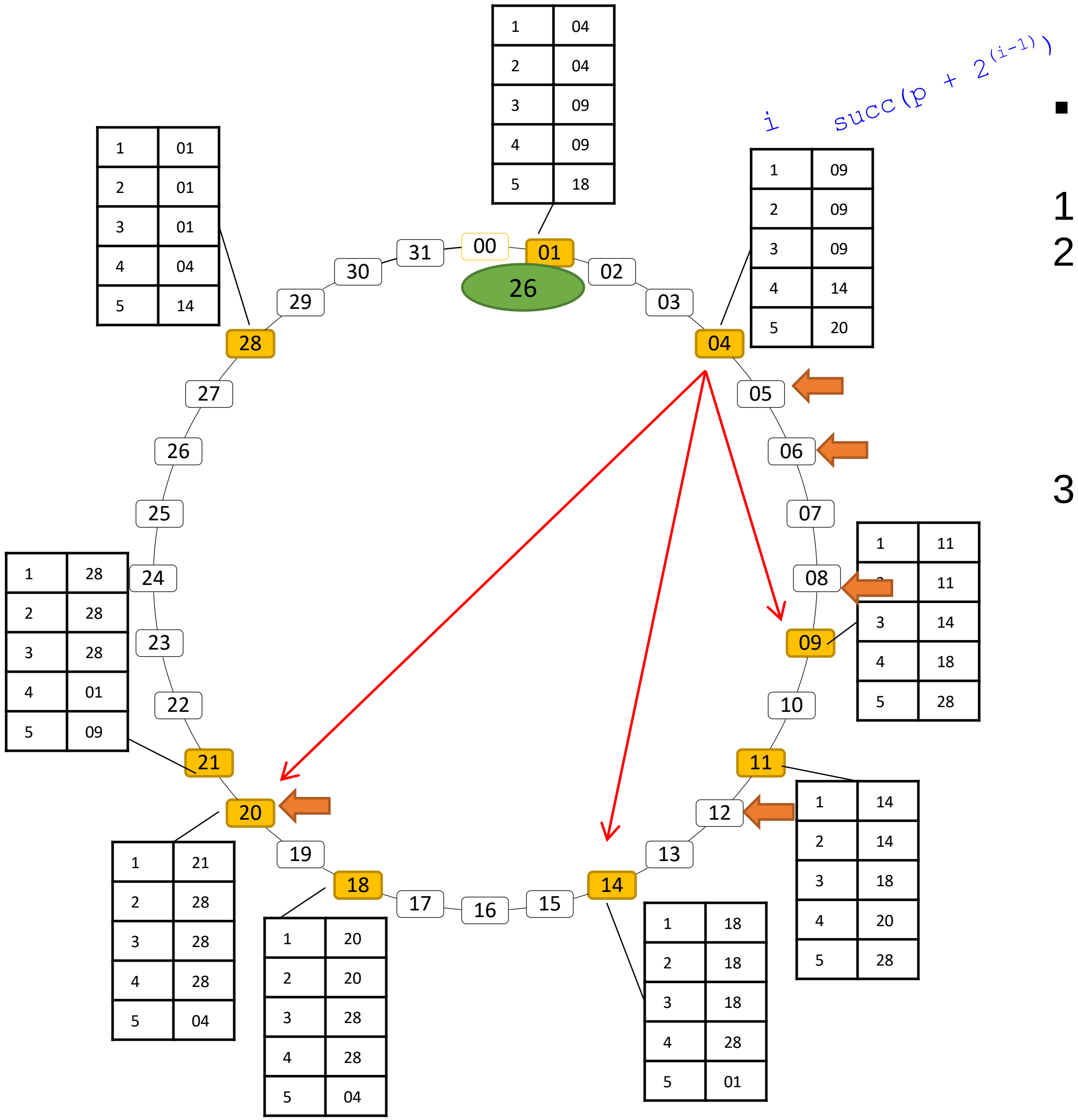


1. All nodes are arranged in a logical ring according to their IDs
2. Each node '`p`' keeps track of its immediate neighbors: `succ(p)` and `pred(p)`
3. If '`p`' receives a request to resolve key '`k`':
   - If `pred(p) < k <=p`, node `p` will handle it
   - Else it will forward it to `succ(n)` or `pred(n)`

Solution is not scalable:

- As the network grows, forwarding delays increase
  - Key resolution has a time complexity of `O(n)`

Key Resolution in Chord



- Chord improves key resolution by reducing the time complexity to `O(log n)`

1. All nodes are arranged in a logical ring according to their IDs
2. Each node 'p' keeps a table FT_p of at-most *m* entries. This table is called Finger Table

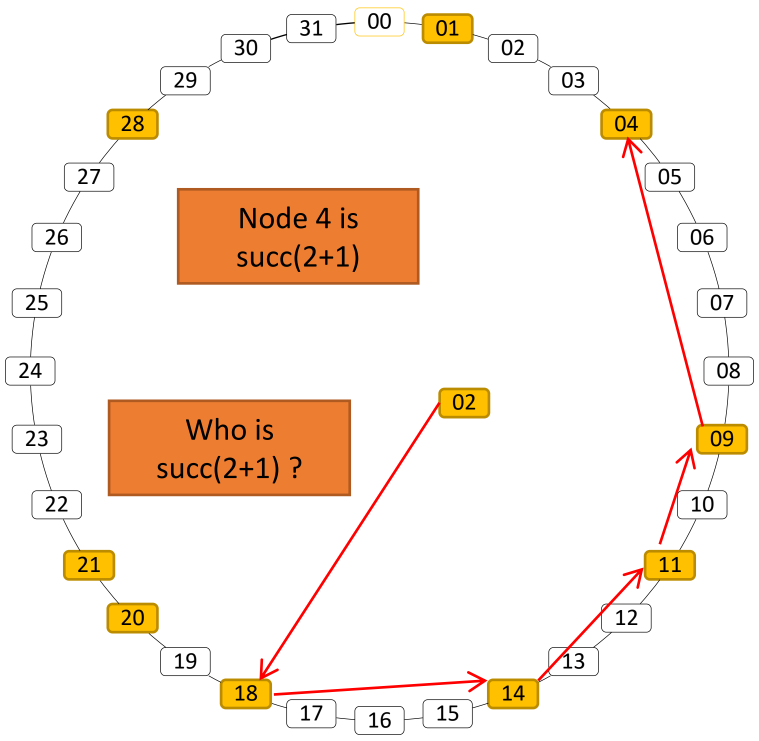
   $$FT_p[i] = succ(p + 2^{(i-1)})$$

   NOTE: $FT_p[i]$ increases exponentially

3. If node 'p' receives a request to resolve key 'k':
   - Node p will forward it to node q with index j in F_p where $q = FT_p[j] <= k < FT_p[j+1]$
     - If $k > FT_p[m]$, then node p will forward it to $FT_p[m]$
     - If $k < FT_p[1]$, then node p will forward it to $FT_p[1]$

Chord – Join and Leave Protocol

- In large-scale distributed Systems, nodes dynamically *join* and *leave* (voluntarily or due to failures)
- If a node p wants to join:
  - It contacts arbitrary node, looks up for `succ(p+1)`, and inserts itself into the ring
- If node p wants to leave:
  - It contacts `pred(p)` and `succ(p+1)` and updates them



Chord – Finger Table Update Protocol

- For any node q, $FT_q[1]$ should be up-to-date
  - It refers to the next node in the ring
  - Protocol:
    - Periodically, request `succ(q+1)` to return `pred(succ(q+1))`
    - If $q = pred(succ(q+1))$, then information is up-to-date
    - Otherwise, a new node p has been added to the ring such that $q < p < succ(q+1)$
      - $FT_q[1] = p$
      - Request `p` to update $pred(p) = q$
    - Similarly, node `p` updates each entry `i` by finding $succ(p + 2^{(i-1)})$

Exploiting Network Proximity in Chord

- The logical organization of nodes in the overlay network may lead to inefficient message transfers
  - Node `k` and node `succ(k +1)` may be far apart

- Chord can be optimized by considering the network location of nodes

  1. Topology-Aware Node Assignment
     - Two nearby nodes get identifiers that are close to each other

  2. Proximity Routing
     - Each node `q` maintains '`r`' successors for `i`th entry in the finger table
     - $FT_q[i]$ now refers to r successor nodes in the range

       $$[p + 2^{(i-1)}, p + 2^{i} -1]$$
     - To forward the lookup request, pick one of the r successors closest to the node `q`

Today…

- Last Lecture:
  - Naming- Part I
- Today's Session:
  - Naming- Part II
- Announcements:
  - PS2 is due tomorrow by midnight
  - Quiz I will take place on Feb 14 during the class time
  - P1 is due on Monday, Feb 21 by midnight

Classes of Naming

- Flat naming
- Structured naming
- Attribute-based naming

Classes of Naming

- Flat naming
- Structured naming
- Attribute-based naming

Structured Naming

- Structured names are composed of simple human-readable names
  - Names are arranged in a specific structure

- Examples:
  - File-systems utilize structured names to identify files
    - /home/userid/work/dist-systems/naming.txt

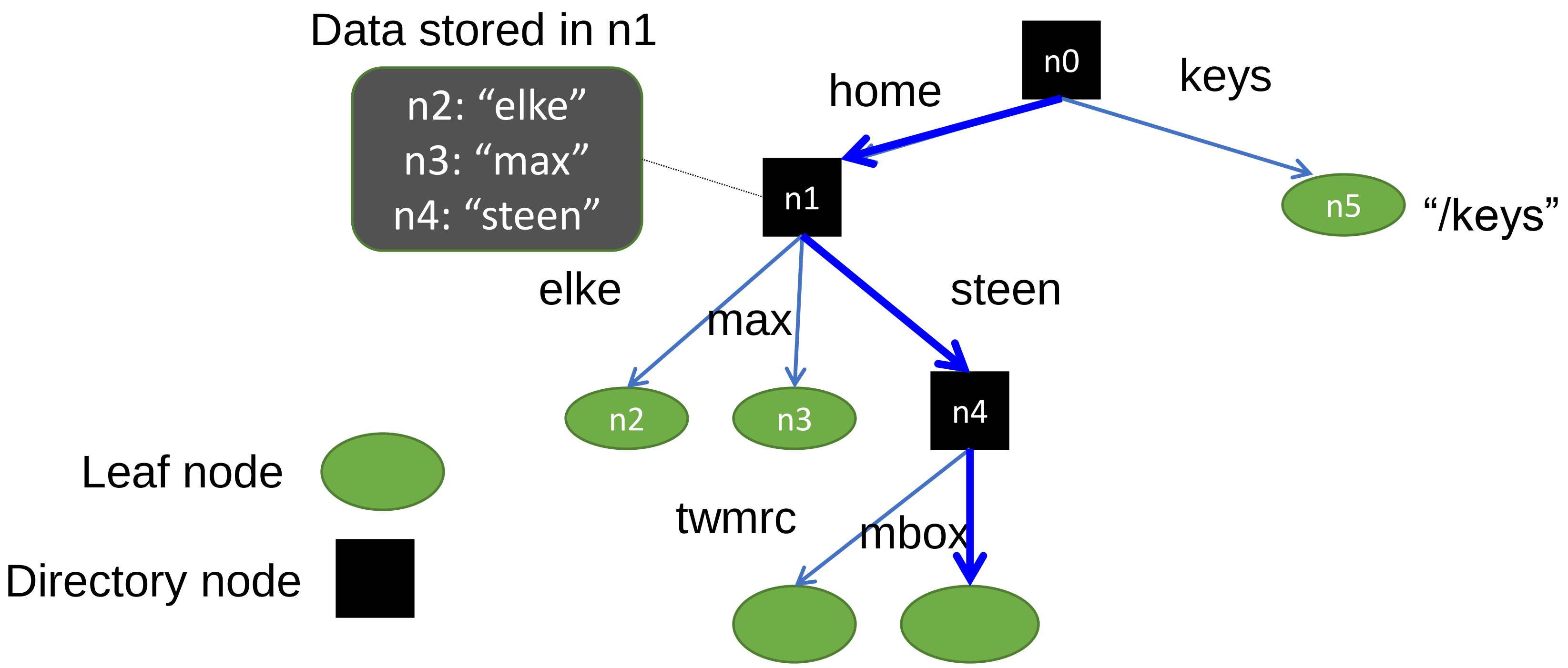
  - Websites can be accessed through structured names
    - www.cs.qatar.cmu.edu

Name Spaces

- Structured names are organized into *name spaces*
- A name space is a *directed graph* consisting of:
  - Leaf nodes
    - Each leaf node represents an entity
    - A leaf node generally stores the *address* of an entity (e.g., in DNS), or the *state* of (or the *path* to) an entity (e.g., in file systems)
  - Directory nodes
    - Directory node refers to other leaf or directory nodes
    - Each outgoing edge is represented by (*edge label, node identifier*)
- Each node can store any type of data
  - I.e., State and/or address (*e.g., to a different machine*) and/or path

Name Spaces: An Example

Looking up for the entity with name “/home/steen/mbox”



Name Resolution

- The process of looking up a name is called *name resolution*
- Closure mechanism:
  - Name resolution cannot be accomplished without an *initial directory node*
  - The *closure mechanism* selects the implicit context from which to start name resolution
  - Examples:
    - www.qatar.cmu.edu: start at the DNS Server
    - /home/steen/mbox: start at the root of the file-system

Name Linking

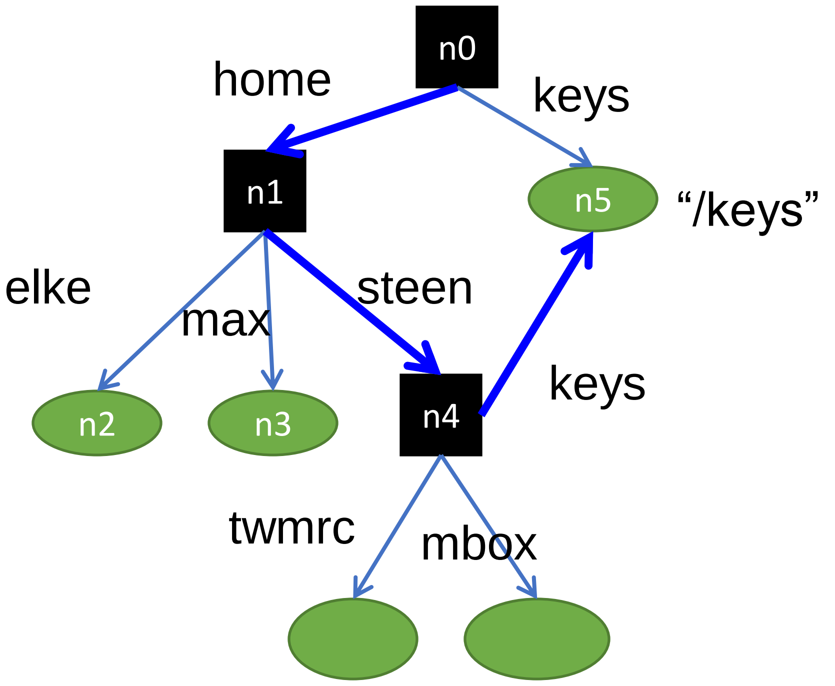
- The name space can be effectively used to link two different entities

- Two types of links can exist between the nodes:
  1. Hard Links
  2. Symbolic Links

1. Hard Links

- There is a directed link from the hard link to the actual node
- Name resolution:
  - Similar to the general name resolution
- Constraint:
  - There should be no cycles in the graph

“/home/steen/keys” is a hard link to “/keys”

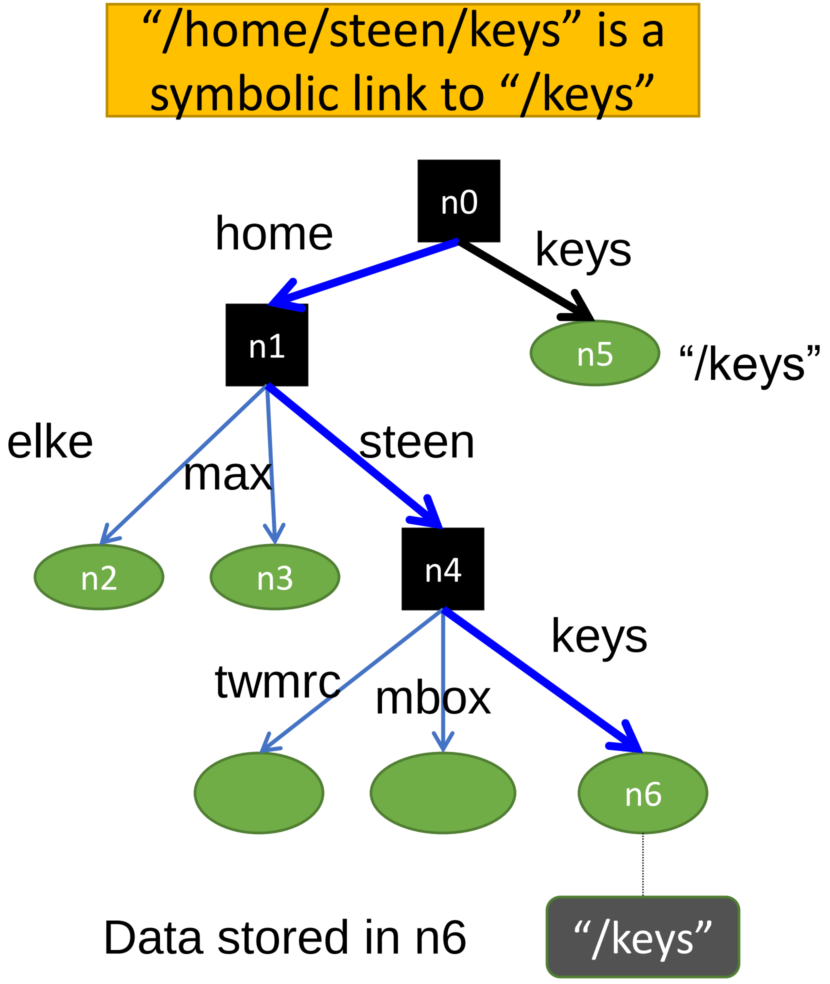


2. Symbolic Links

- Symbolic link stores the name of the original node as *data*

- Name resolution for a symbolic link SL
  - First resolve SL's name
  - Read the content of SL
  - Name resolution continues with content of SL

- Constraint:
  - No cyclic references should be present



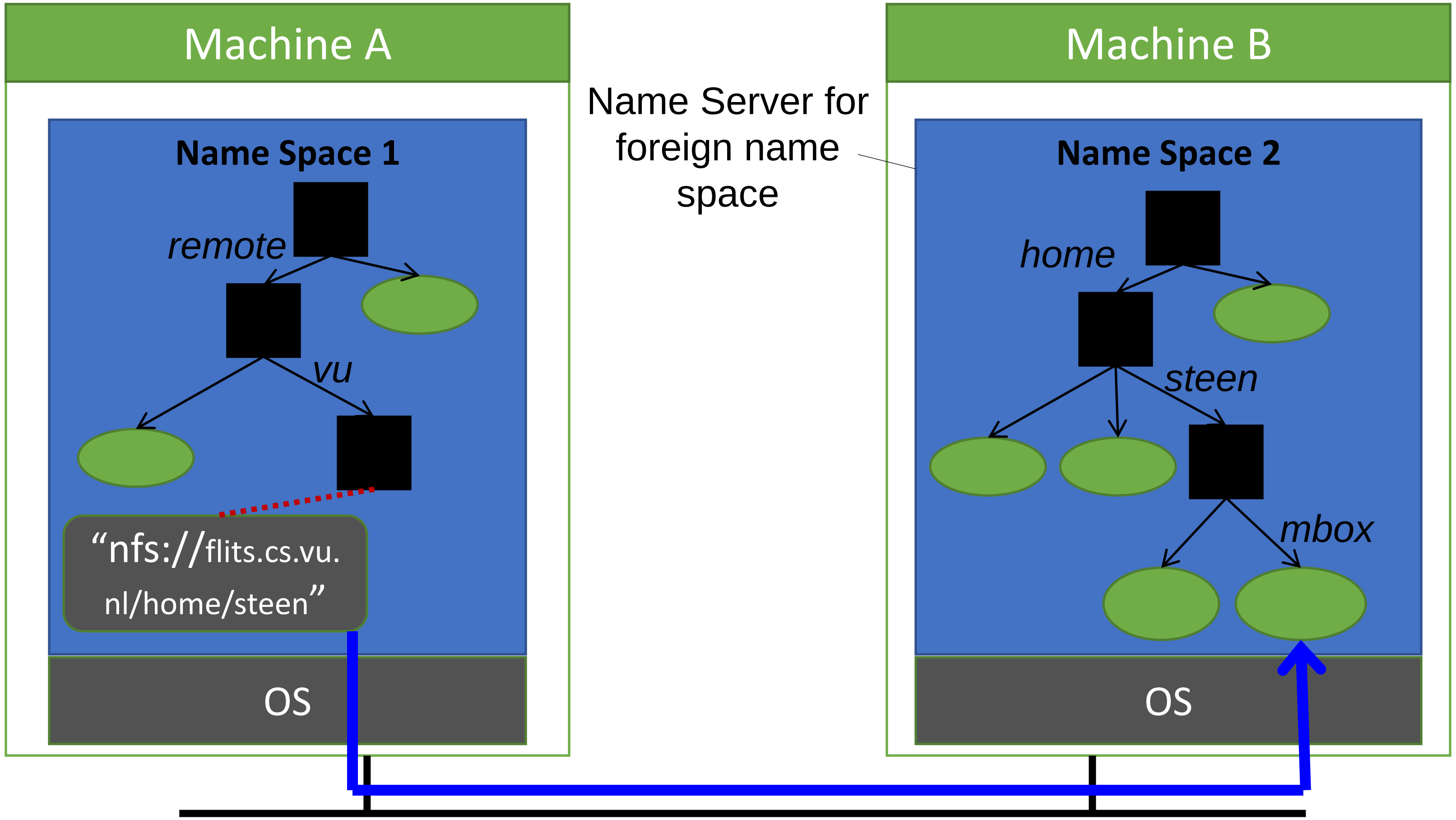
Mounting of Name Spaces

- Two or more name spaces can be merged transparently by a technique known as *mounting*

- With mounting, a directory node in one name space will store the identifier of the directory node of another name space

- Network File System (NFS) is an example where different name spaces are mounted
  - NFS enables *transparent* access to remote files

Example of Mounting Name Spaces in NFS



Name resolution for "/remote/vu/home/steen/mbox" in a distributed file system

Distributed Name Spaces

- In large-scale distributed systems, it is essential to distribute name spaces over multiple name servers
  - Distribute the nodes of the naming graph
  - Distribute the name space management
  - Distribute the name resolution mechanisms

Layers in Distributed Name Spaces

- Distributed name spaces can be divided into three *layers*

Global Layer

- Consists of high-level directory nodes
- Directory nodes are jointly managed by different administrations

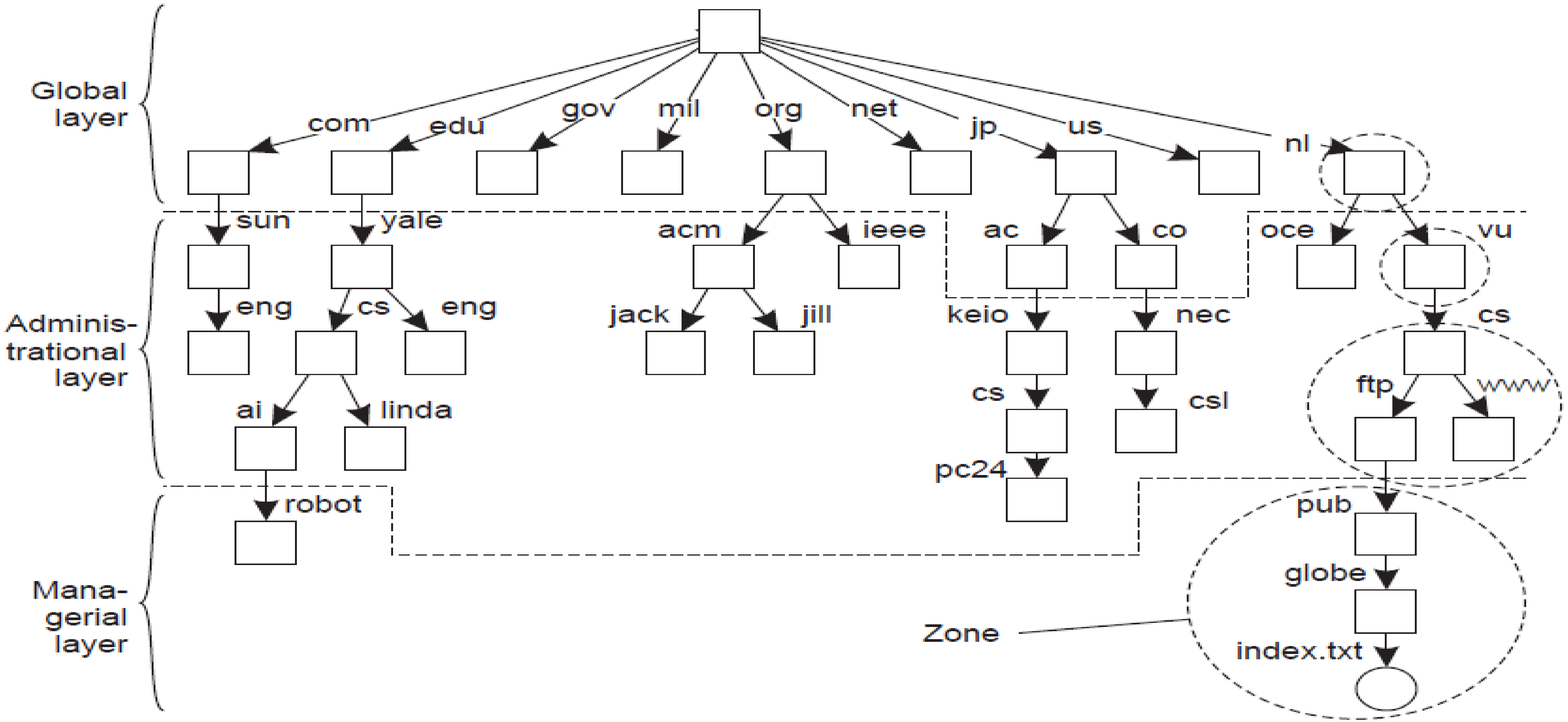
Administrat-ional Layer

- Contains mid-level directory nodes
- Directory nodes grouped together in such a way that each group is managed by an administration

Managerial Layer

- Contains low-level directory nodes within a single administration
- The main issue is to efficiently map directory nodes to local name servers

Distributed Name Spaces – An Example



Comparison of Name Servers at Different Layers

	Global	Administrational	Managerial
Geographical scale of the network	Worldwide	Organization	Department
Total number of nodes	Few	Many	Vast numbers
Number of replicas	Many	None or few	None
Update propagation	Lazy	Immediate	Immediate
Is client side caching applied?	Yes	Yes	Sometimes
Responsiveness to lookups	Seconds	Milliseconds	Immediate

Distributed Name Resolution

- Distributed name resolution is responsible for mapping names to addresses in a system where:
  - Name servers are distributed among participating nodes
  - Each name server has a local *name resolver*

- We will study two distributed name resolution algorithms:
  1. Iterative Name Resolution
  2. Recursive Name Resolution

1. Iterative Name Resolution

1. Client hands over the complete name to *root name server*

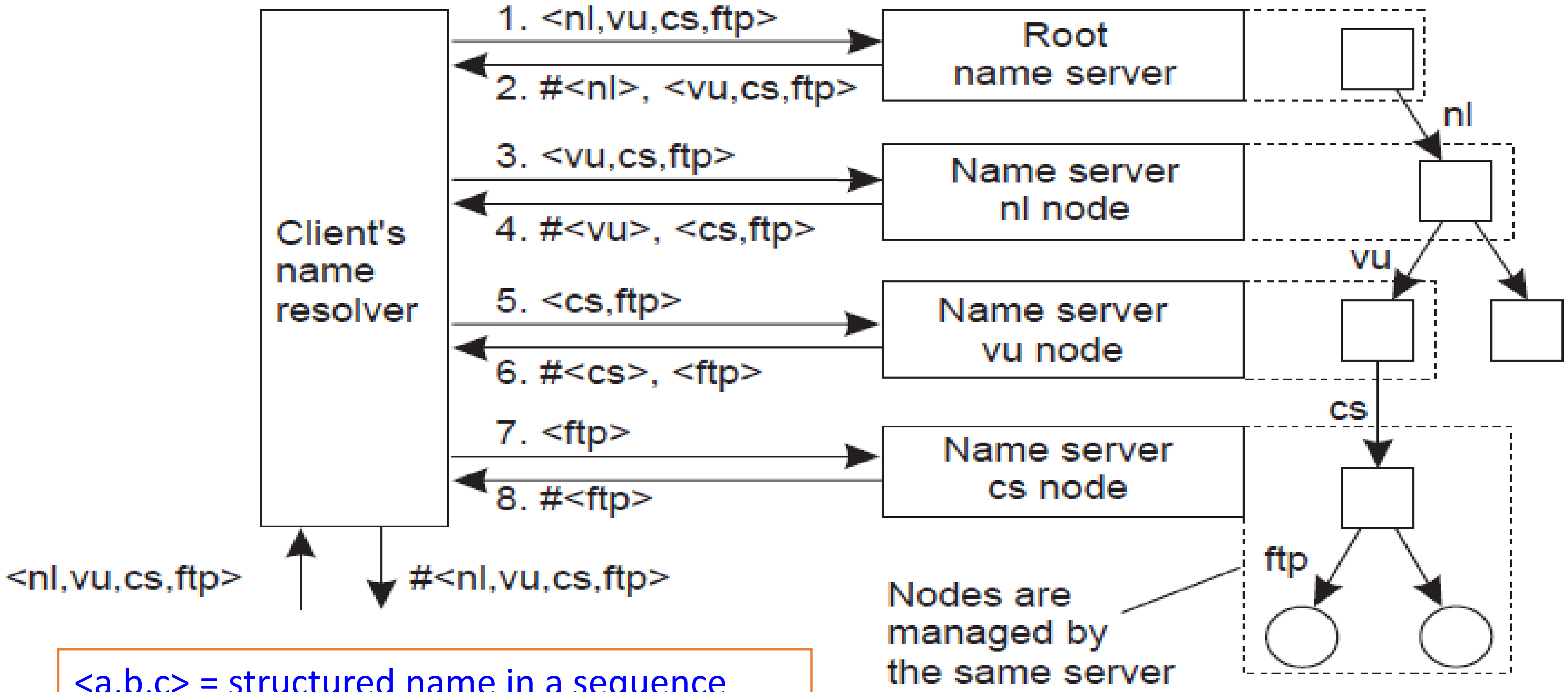
2. Root name server resolves the name as far as it can, and returns the result to the client
   - The root name server returns the address of the next-level name server (say, NLNS) if address is not completely resolved

3. Client passes the unresolved part of the name to the NLNS

4. NLNS resolves the name as far as it can, and returns the result to the client (and probably its next-level name server)

5. The process continues untill the full name is resolved

1. Iterative Name Resolution – An Example



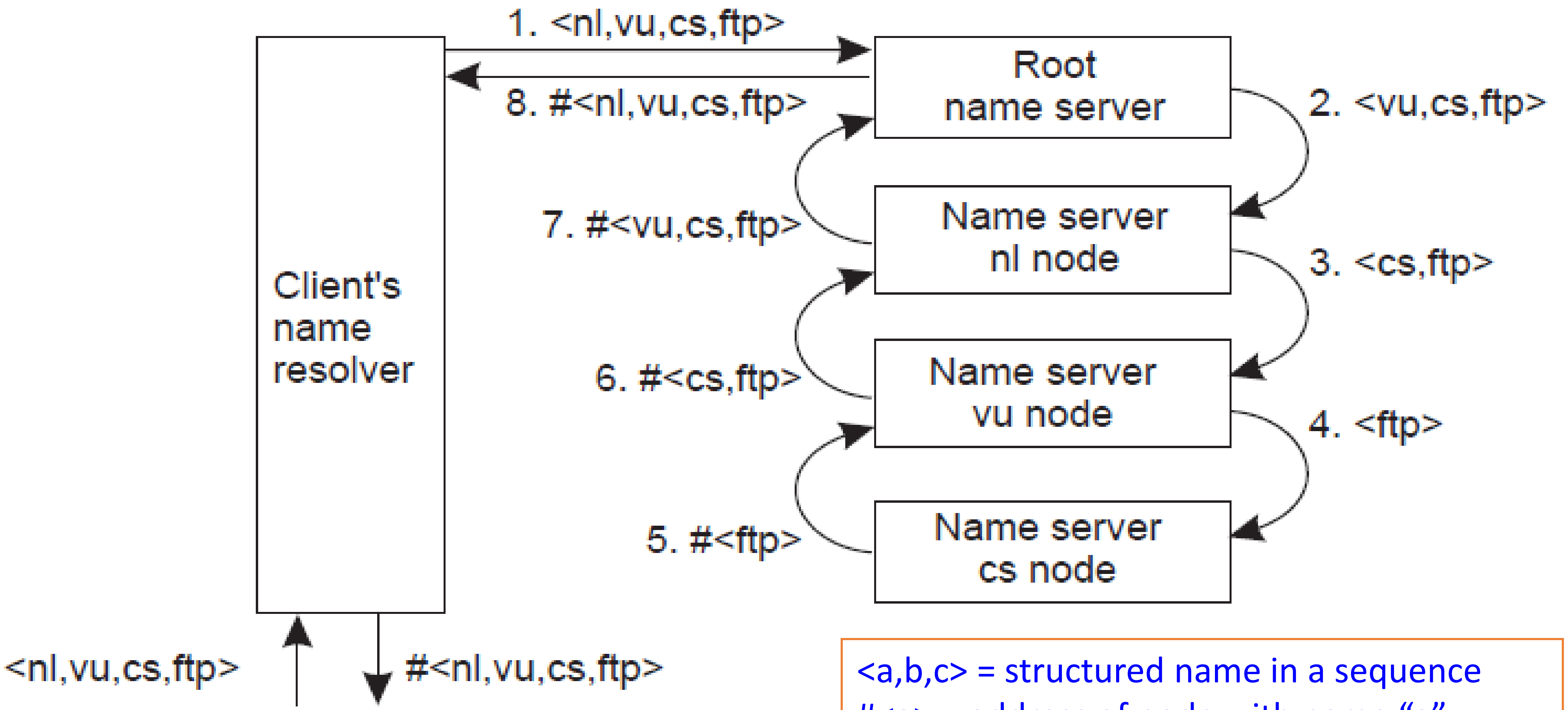
<a,b,c> = structured name in a sequence
#<a> = address of node with name “a”

Resolving the name “*ftp.cs.vu.nl*”

2. Recursive Name Resolution

- Approach:
  - Client provides the name to the root name server
  - The root name server passes the result to the next name server it finds
  - The process continues till the name is fully resolved
- Drawback:
  - Large overhead at name servers (especially, at the high-level name servers)

2. Recursive Name Resolution – An Example



<a,b,c> = structured name in a sequence
#<a> = address of node with name “a”

Resolving the name “*ftp.cs.vu.nl*”

Classes of Naming

- Flat naming
- Structured naming
- 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
  - Similar to traditional directory services (e.g., yellow pages)

- However, the lookup operations can be extremely expensive
  - They require to match requested attribute values, against actual attribute values, which might require inspecting all entities

- Solution: Implement basic directory service as a database, and combine it with traditional structured naming system

- We will study Light-weight Directory Access Protocol (LDAP); an example system that uses attribute-based naming

Light-weight Directory Access Protocol (LDAP)

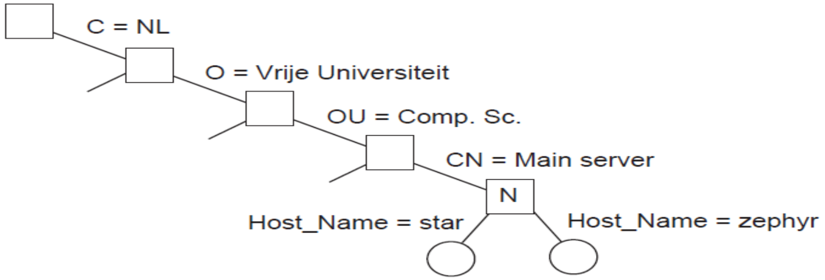
- LDAP directory service consists of a number of records called "directory entries"
  - Each record is made of (attribute, value) pairs
  - LDAP standard specifies five attributes for each record
- Directory Information Base (DIB) is a collection of all directory entries
  - Each record in a DIB is unique
  - Each record is represented by a distinguished name

```
E.g., /C=NL/O=Vrije Universiteit/OU=Comp. Sc.
```

Attribute	Value
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	star
Host_Address	192.31.231.42

Directory Information Tree in LDAP

- All the records in the DIB can be organized into a hierarchical tree called *Directory Information Tree (DIT)*



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

- LDAP provides advanced search mechanisms based on attributes by traversing the DIT
- Example syntax for searching all Main_Servers in Vrije Universiteit:

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

Summary

- Naming and name resolutions enable accessing entities in a distributed system
- Three types of naming:
  - Flat Naming
    - Broadcasting, forward pointers, home-based approaches, Distributed Hash Tables (DHTs)
  - Structured Naming
    - Organizes names into Name Spaces
    - Distributed Name Spaces
  - Attribute-based Naming
    - Entities are looked up using their attributes

Kuliah Berikutnya

- Teknik Sinkronisasi

Pertanyaan?