DISTRIBUTED SYSTEMS
Principles and Paradigms

Second Edition

ANDREW S. TANENBAUM
MAARTEN VAN STEEN

Chapter 5
Naming

*** Modified by Prof. Rivalino Matias, Jr.**

Outline

- Names, addresses, identifiers
- Flat (simple) naming
- Structured naming
- Attribute-based naming

Naming (1)

- **Names** play an important role in all computer systems, they are used to:
  - Share resources.
  - Uniquely identify entities (objects, resources).
  - Refer to locations, more…
- A name can be resolved to the entity it refers to.
  - **Name resolution** allows a process to access the named entity.
  - To **resolve** names it is necessary to implement a **naming system**.
- In a DS, the naming system (NS) is itself often distributed across multiple machines.
  - How this distribution is done plays a key role in the efficiency and scalability of the naming system.

Names, Addresses, Identifiers

- What a name actually is?
  - In DS, a **name** is a string of characters that is used to refer to an **entity**.
  - An **entity** in a DS can be practically anything.
    - Users, hosts, printers, disks, files, processes, procedures, mailboxes, Web pages, graphical windows, network connections, and so on.
- To operate on an entity, it is necessary to access it:
  - This access is done through an **access point**.
  - The **name** of an **access point** of an **entity** is called an **address**.
- An **entity** can offer more than one **access point**.
  - A **phone** can be seen as an **access point** of a person (**entity**), whereas the phone number corresponds to an **address**.
  - In a DS, a server process (**entity**) is accessed through its address that is the combination of the host's IP and the service's port number.

Names, Addresses, Identifiers

- Because an access point is tightly associated with an entity, it would be seemed convenient to use its address as a regular name for the entity.
- But there are problems with this approach:
  - This is hardly done as such naming is inflexible and human unfriendly.
  - It is not uncommon that a server system may run in a different host than previously.
  - An entity may easily change the access point (AP), and an AP may be reassigned to a different entity.
- If an address (e.g., IP) is used as a name of an entity, we will have an invalid reference if the access point changes or is reassigned to another entity.
  - Therefore it is much better to let the entity be known by a separate name **independent** of the entity's address.

Names, Addresses, Identifiers

- A name that is independent from its address is called **location independent**.
- A name that uniquely identify an entity is a true identifier.
- Properties of a true identifier:
  - 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.
- By using true identifiers, it becomes much easier to unambiguously refer to an entity.
  - An **identifier** refers to at most one entity.
  - An address that is reused to another entity is not a true identifier.
- Examples of true identifiers:
  - CPF, ISBN, DOI, MAC address, Standard TCP/IP Port number, …

Naming (2)

- Identifiers usually are not human friendly
  - They are a random string of bits in a machine-readable form.
  - E.g., an Ethernet address is a random string of 48 bits.

- Humans prefer to use readable names.
  - Represented as a string of characters and they are often structured. E.g.,
    - www.ufu.br
    - /classes/gsi028/students.txt, C:\docs\recommendation-letter.txt

Naming (3)

- Having names, addresses, and identifiers brings us to a central theme: How to resolve names and identifiers to addresses?
- The answer is a **Naming systems**:
  - It maintains a **name-to-address** binding:
    - The simplest form is a table of (name, address) pairs.
    - In large distributed system, a centralized table will not work.
  - In SD, usually the name is decomposed into several parts, and the name resolution takes place via a recursive lookup of those parts.
  - For example:
    - A client needing to know the address of a ftp server (e.g., **ftp.facom.ufu.br**).
    - 1st (**br**), 2nd (**ufu**), 3rd (**facom**)

Naming (4)

- Three different Classes of Naming system
  - Flat naming
  - Structured naming.
  - Attributed based naming.

Flat Naming

- Applied to resolve identifiers to addresses.
- Identifier:
  - String of bits that are unstructured (flat) names.
  - It does not contain any information on how to locate the entity's address.
- Given an entity's identifier, how get its address?
  - **Solutions**:
    - Broadcasting & Multicasting.
    - Forwarding Pointers
    - Distributed Hash Tables (DHT)
    - Hierarchical approach

Flat Naming – Broad/Multicasting

- Simple solution.
- Applicable only to local-area networks (LANs).
  - Broadcasting or Multicasting is difficult to implement efficiently in large-scale networks.
- How it works?
  - A message containing the entity's **identifier** is broadcast to each host in the network, and each host is requested to check whether it has that entity.
  - Only the hosts that can offer an access point for the entity send a reply message containing the **address** of the access point.
  - An example of this approach is the ARP (address resolution protocol) to find a MAC address for a given IP address.

Flat Naming – Broad/Multicasting

- Drawbacks:
  - It becomes inefficient when the network grows.
  - Bandwidth is wasted by request messages.
  - Many hosts are interrupted by requests they cannot answer.

Flat Naming – Forwarding Pointers

- Used to locate mobile entities.
  - When an entity moves from **A** to **B**, it leaves behind in **A** a reference to its new location at **B**.
    - Once the entity has been located, thru a traditional naming service (e.g., DNS), a client can look up the current address by following the chain of forwarding pointers.
  - Drawbacks:
    - A chain can become so long that locating that entity is prohibitive.
    - All intermediate locations in a chain will have to maintain their part of the chain as long as needed.
    - As soon as any forward pointer is lost, the entity can no longer be reached.

Forwarding Pointers (1)

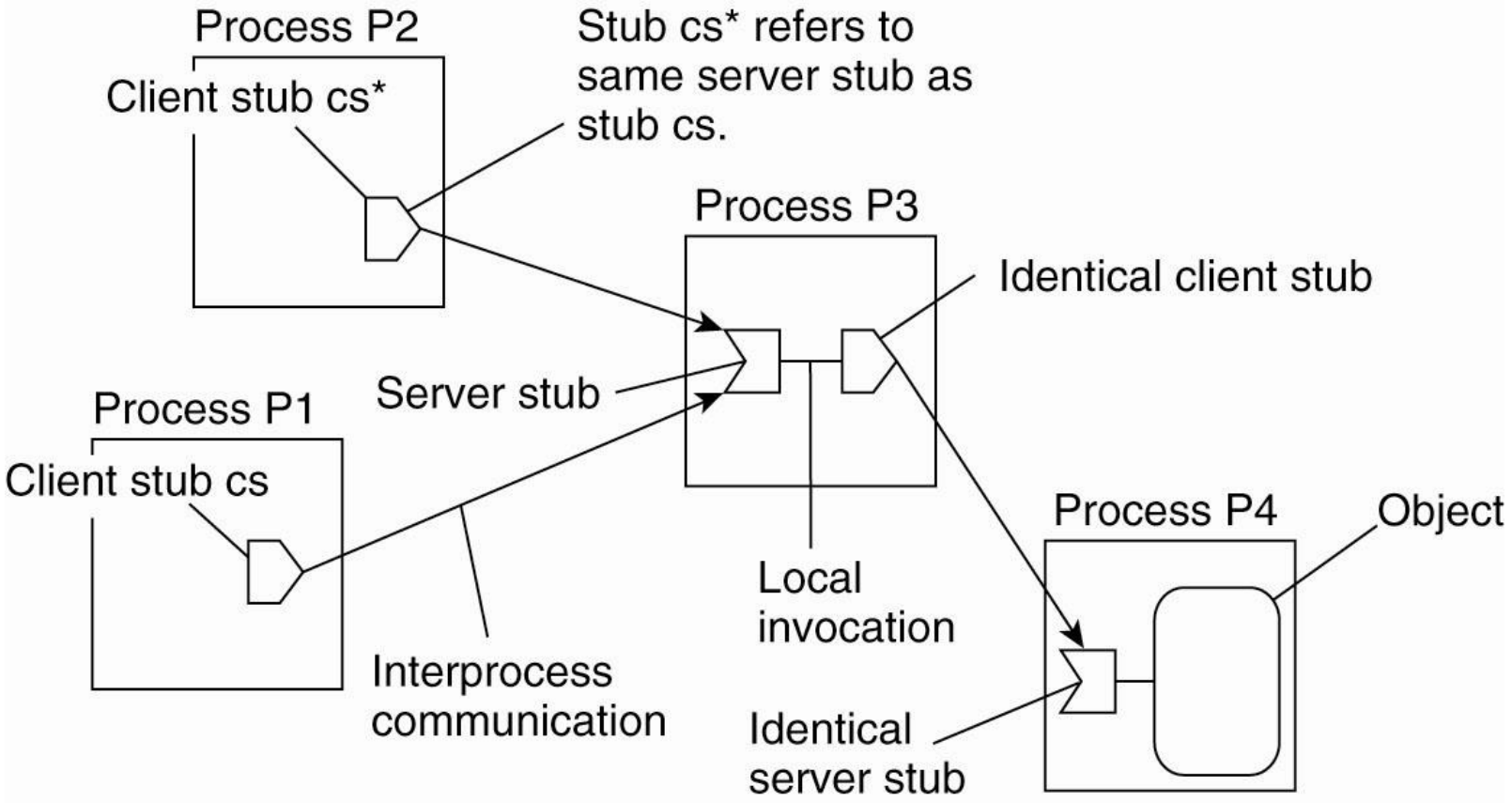

Figure 5-1. The principle of forwarding pointers using (client stub, server stub) pairs.

Forwarding Pointers (2)

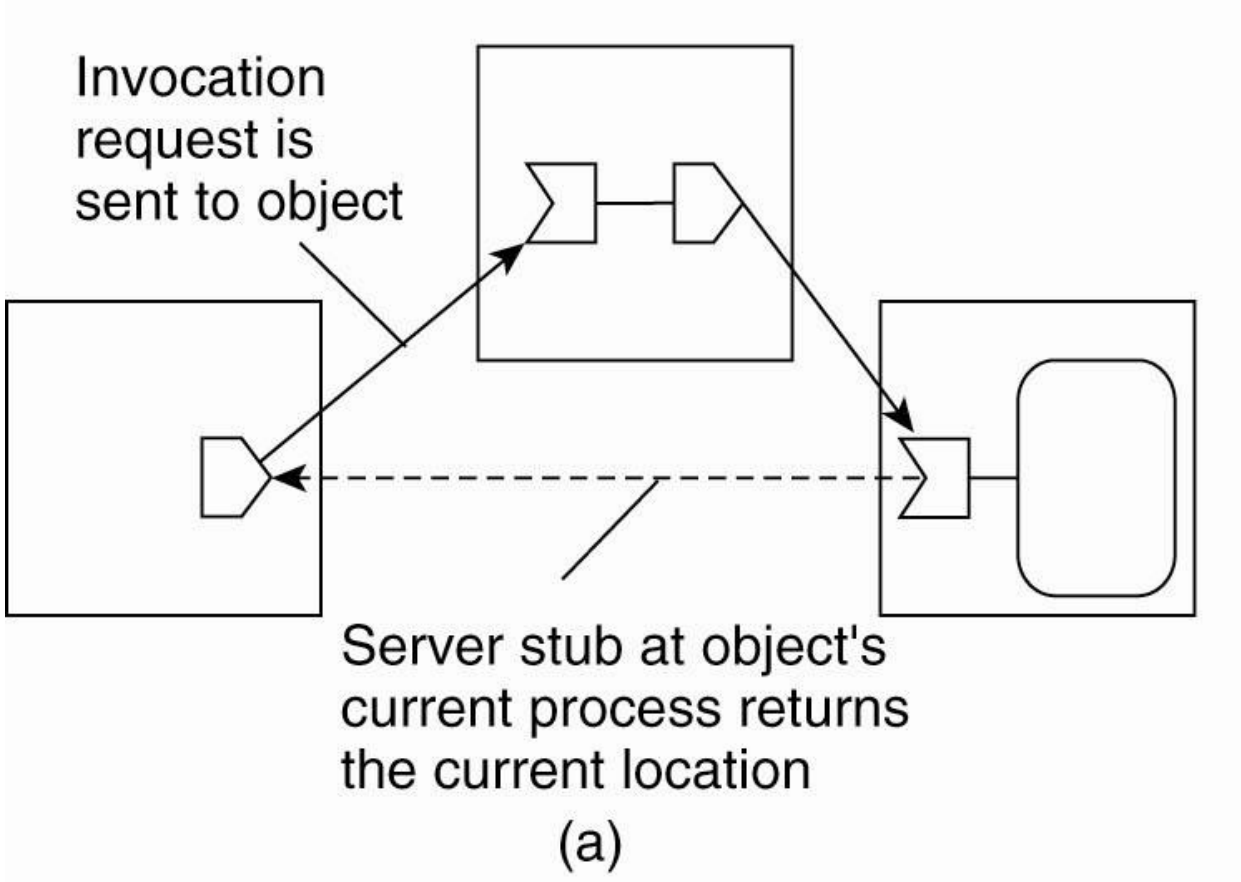

Figure 5-2. Redirecting a forwarding pointer by storing a shortcut in a client stub.

Forwarding Pointers (3)

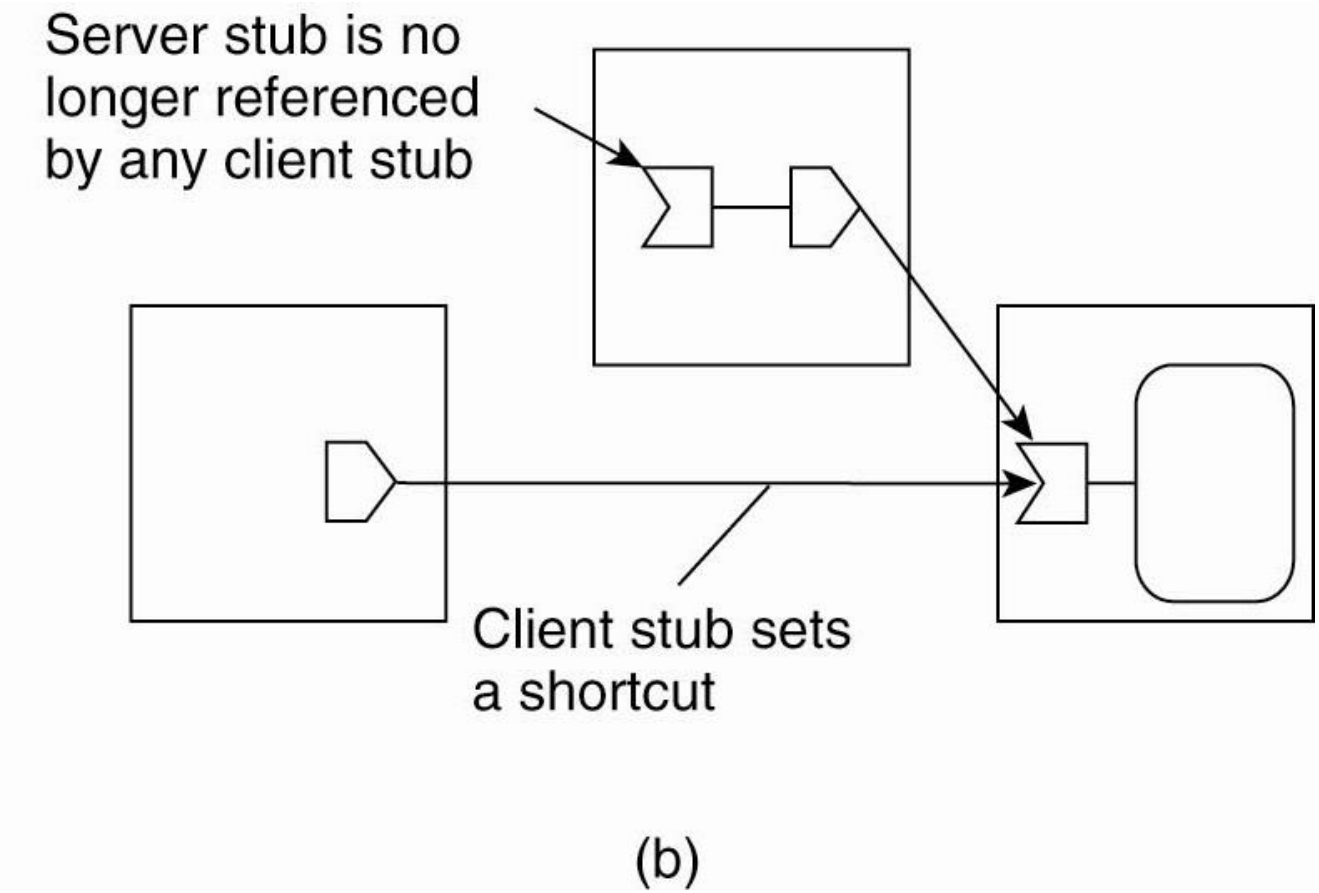

Figure 5-2. Redirecting a forwarding pointer by storing a shortcut in a client stub.

Flat Naming – Native location

- Popular approach to support mobile entities in large-scale network.
  - Each mobile host uses a fixed IP address.
  - All communication to that IP address is initially directed to the mobile host's **home agent** (**HA**).
  - HA is located at the LAN corresponding to the network address contained in the mobile host's IP address.
  - When the mobile host moves to another network it request a temporary address.
  - This **care-of address** is registered at the HA.
  - When the **HA** receives a packet for the mobile host, it looks up the host's current location. If it is on the LAN, simply forward the packed. Otherwise, it is tunneled (**IPIP**) to the host's current location. At the same time, the sender is informed of the host's current location.

Home-Based Approaches

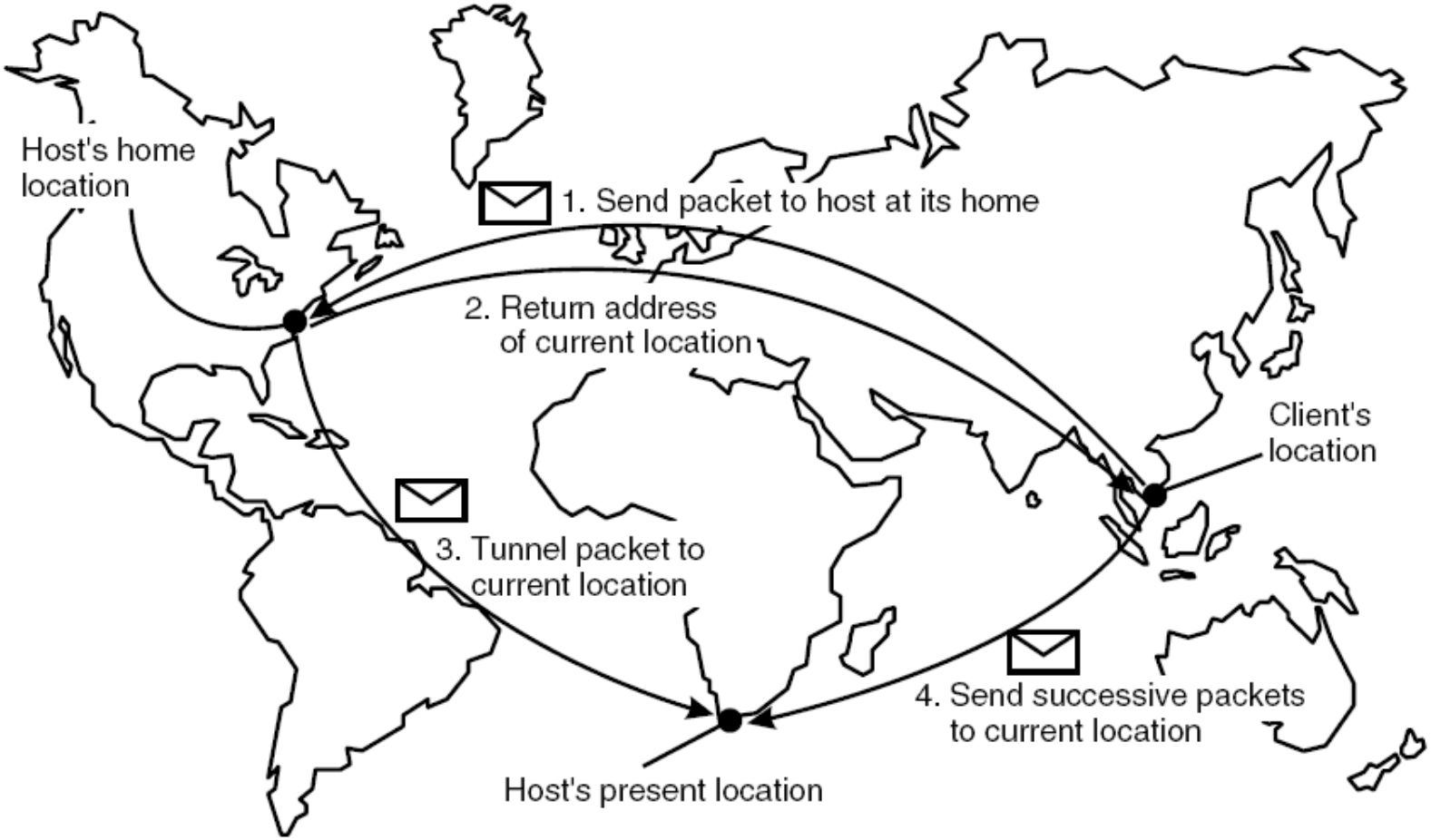

Figure 5-3. The principle of Mobile IP.

Flat Naming – DHT (1)

- Distributed Hash Table
  - A DHT is a distributed system that provides **lookup** service.
  - It is similar to a hash table, i.e., key-value pairs are stored in a DHT, and any participant node (host) can retrieve the value associated with a given key.
  - Many DHT-based systems have been developed in the past decade, with the Chord system being a typical representative.

Flat Naming – DHT (2)

- **Chord**
  - It uses a **m-bit** identifier space to assign randomly chosen **identifiers** to nodes as well as **keys** to specific **entities (**anything**)**.
  - The number **m** is usually 128 or 160, depending on which hash function is used
  - An entity with key *k* falls under the jurisdiction of the **node** with the smallest identifier $id \geq k$. This node is referred to as the successor of *k* and denoted as *succ*(*k*).

Flat Naming – DHT (3)

- **Chord**
  - **How to efficiently resolve a key *k* to the address of *succ*(*k*)?**
    - Linear approach
    - Finger table

Flat Naming – DHT (4)

- **Chord**
  - **Linear approach**
    - Each node $\boldsymbol{p}$ keeps track of the successor $succ(\boldsymbol{p}+1)$ as well as its predecessor, $pred(\boldsymbol{p})$.
    - Whenever a node $\boldsymbol{p}$ receives a request to resolve key k, it will forward the request to one of its two neighbors, unless $pred(\boldsymbol{p}) < k \leq \boldsymbol{p}$ in which case node $\boldsymbol{p}$ returns its own address.
    - This solution is not scalable, since this will lead to a $O(N)$ query time where N is the number of nodes in the ring.

Flat Naming – DHT (5)

- **Chord**
  - **Finger table**
    - Each node maintains a **table** containing $s \leq m$ entries.
    - If $\mathbf{FT}_p$ denotes the table of node $\boldsymbol{p}$, then $\mathbf{FT}_p[i] = \text{succ}(\boldsymbol{p}+2^{i-1})$

      The i-th entry points to the first node succeeding $\boldsymbol{p}$ by at least 2^{i-1}.

      Note that these entries are actually *shortcuts* to existing nodes in the identifier space, where the short-cutted distance from node p increases exponentially as the index i increases.
    - To look up a key k, the node $\boldsymbol{p}$ forwards the request to node $\boldsymbol{q}$ with index j in p's table, i.e.:

      $\boldsymbol{q} = \mathbf{FT}_p[j] \leq k < \mathbf{FT}_p[j+1]$, or

      $\boldsymbol{q} = \mathbf{FT}_p[1]$ when $\boldsymbol{p} < k < \mathbf{FT}_p[1]$.
    - It can be shown that a lookup will require $O(\log(N))$ steps, with N being the number of nodes in the system.

Distributed Hash Tables

General Mechanism

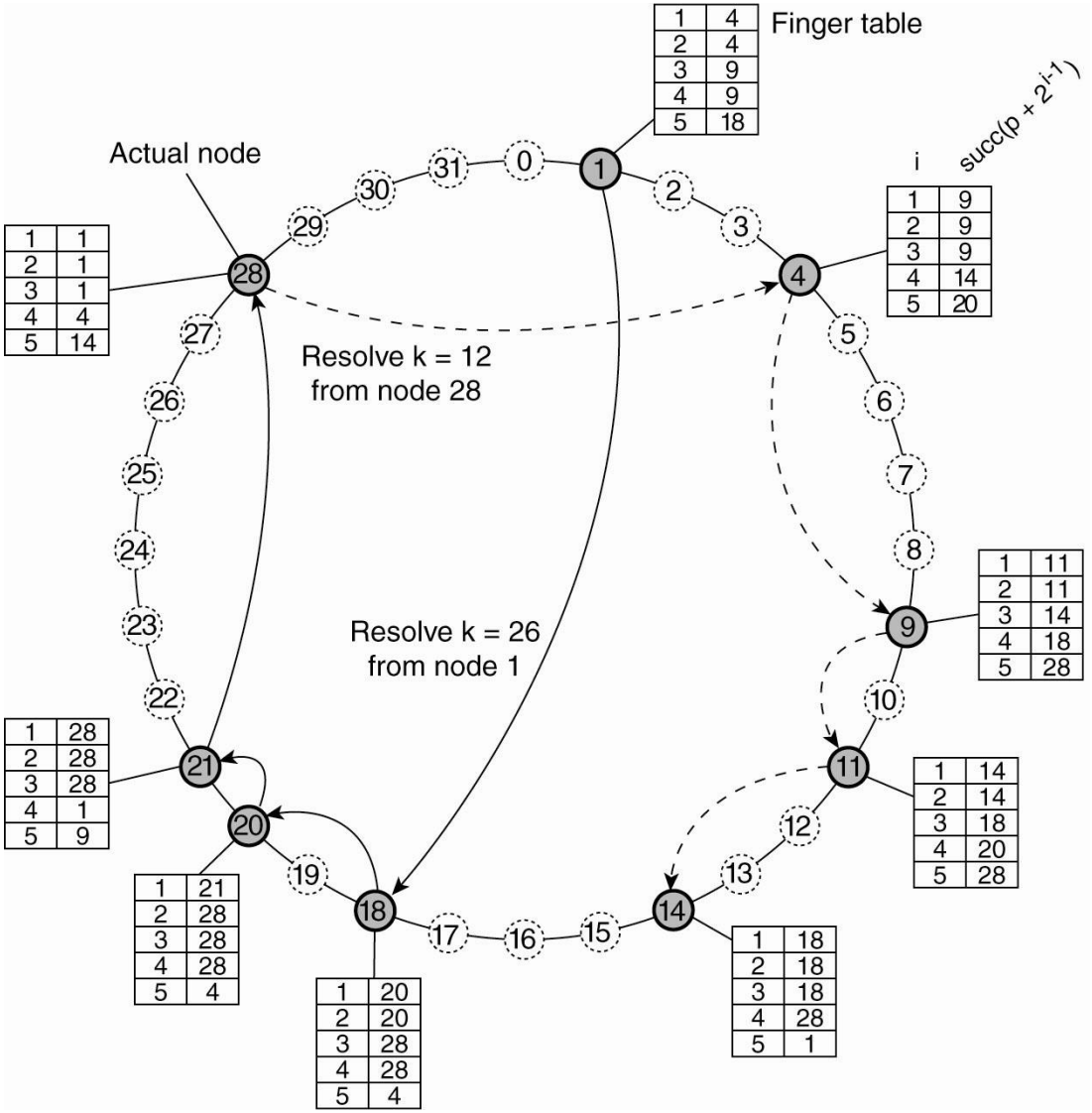

Figure 5-4. Resolving key 26 from node 1 and key 12 from node 28 in a Chord system.

Flat Naming – Hierarchical (1)

- In a Hierarchical scheme, a network is divided into a collection of **domains**.
  - There is a single top-level domain that spans the entire network.
  - Each domain can be subdivided into multiple, smaller subdomains.
    - A lowest-level domain, called a **leaf domain**, typically corresponds to a local-area network.
  - The general assumption is that within a smaller domain the average time it takes to transfer a message from one node to another is less than in a large domain.
    - Each domain **D** has an associated directory node dir(**D**) that keeps track of the **D**'s entities. This leads to a tree of directory nodes.
    - The directory node of the top-level domain, called the **root (directory) node**, knows about all entities.

Hierarchical Approaches (1)

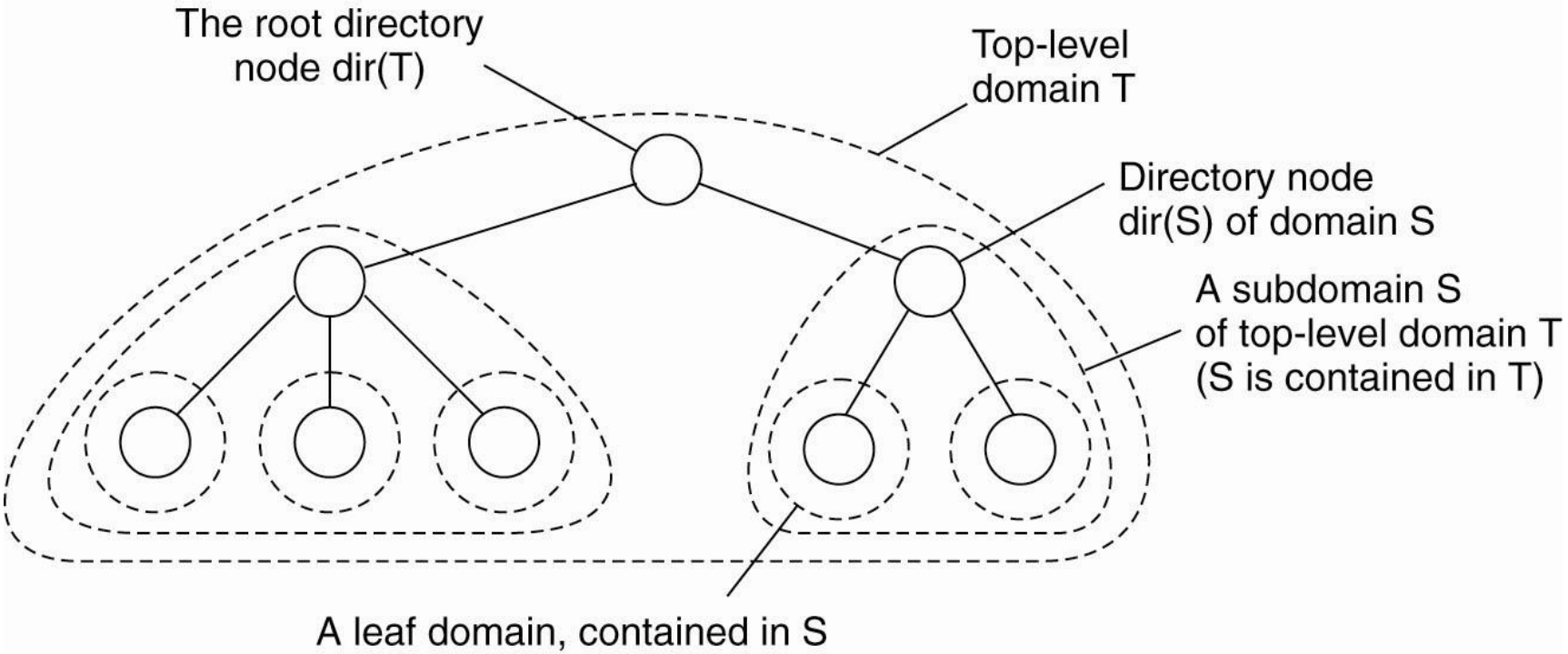

Figure 5-5. Hierarchical organization of a location service into domains, each having an associated directory node.

Flat Naming – Hierarchical (2)

- Keeping track of entities:
  - Each entity in a domain **D** is represented by a **location record** in the directory node dir(**D**).
  - A **location record** for entity **E** in the **directory** node **N** for a leaf domain **D** contains the entity's current address in that domain.
    - In contrast, the directory node **N**' for the next higher-level domain **D'** that contains D has a location record for **E** containing a pointer to N.
    - Likewise, the parent node of **N**' stores a location record for **E** containing a pointer to N'.
    - So, the root node will have a location record for each entity, where each location record stores a pointer to the directory node of the next lower-level subdomain where that record's associated entity is currently located.

Flat Naming – Hierarchical (3)

- An entity may have multiple addresses, e.g., if it is replicated:
  - If an entity has an address in leaf domain **D1** and **D2** respectively, then the directory node of the smallest domain containing both **D1** and **D2**, will have two pointers, one for each.

Hierarchical Approaches (2)

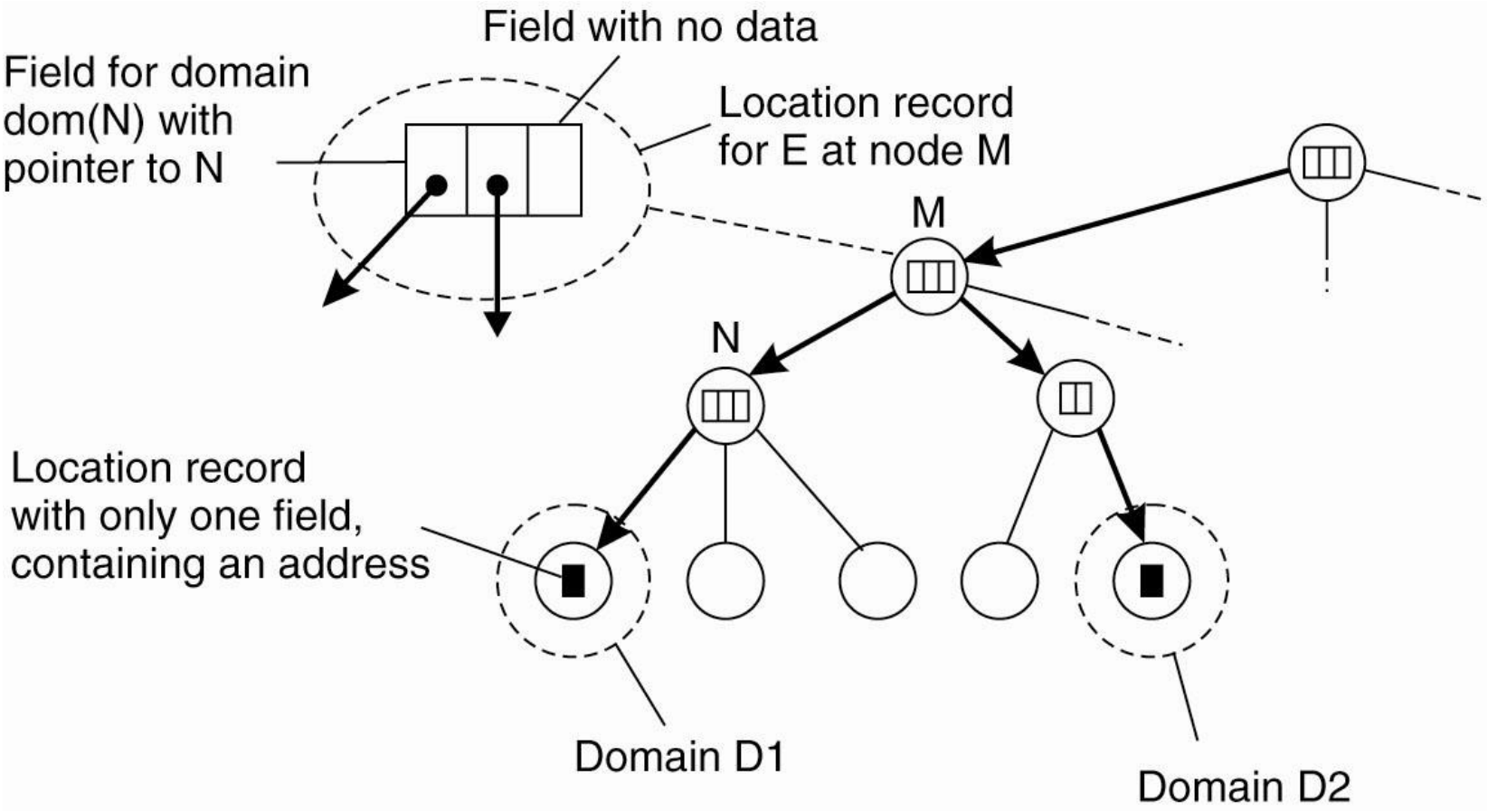

Figure 5-6. An example of storing information of an entity having two addresses in different leaf domains.

Hierarchical Approaches (3)

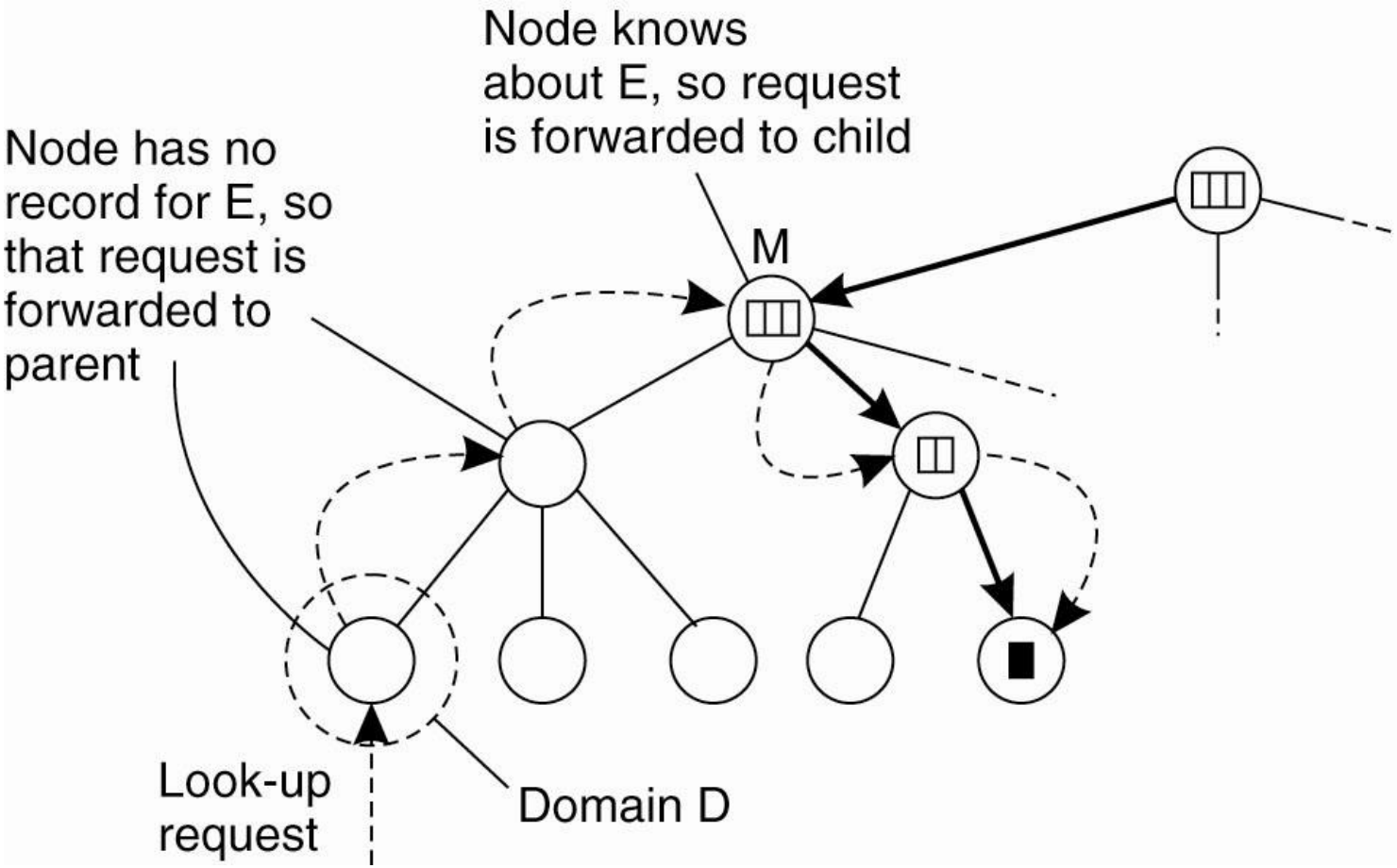

Figure 5-7. Looking up a location in a hierarchically organized location service.

Hierarchical Approaches (4)

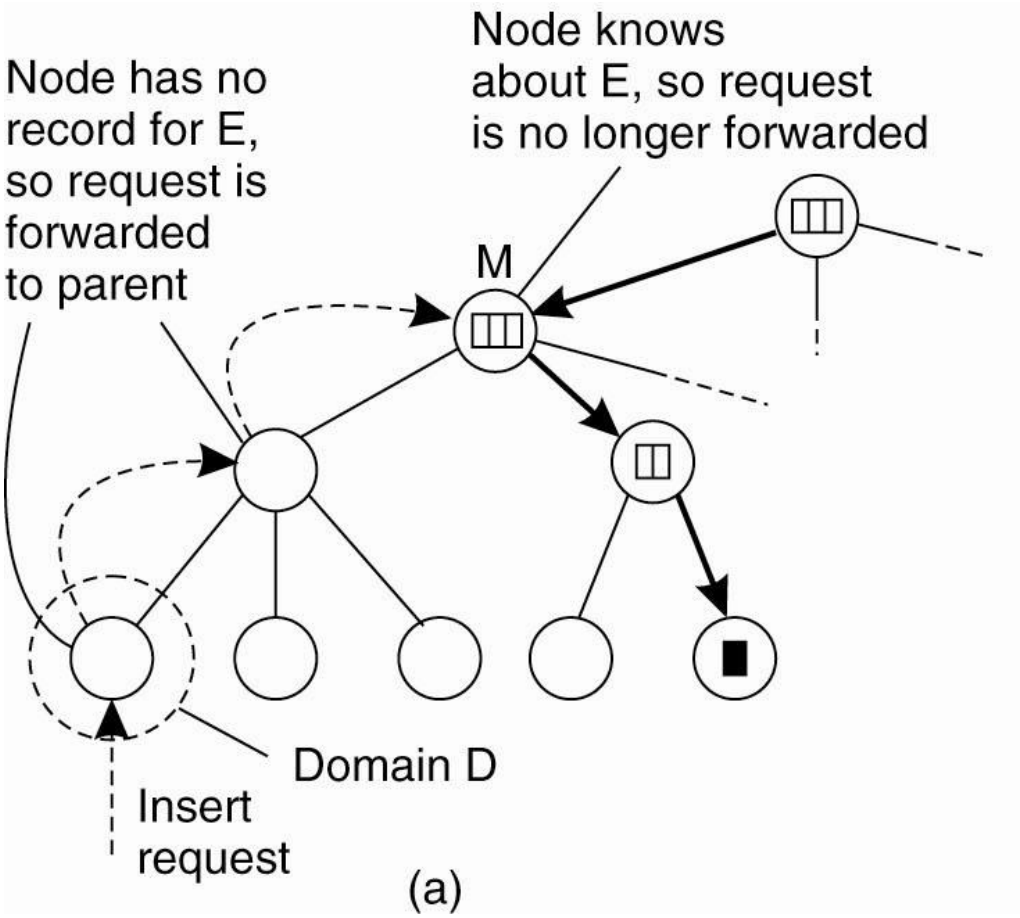

Figure 5-8. (a) An insert request is forwarded to the first node that knows about entity E.

Hierarchical Approaches (5)

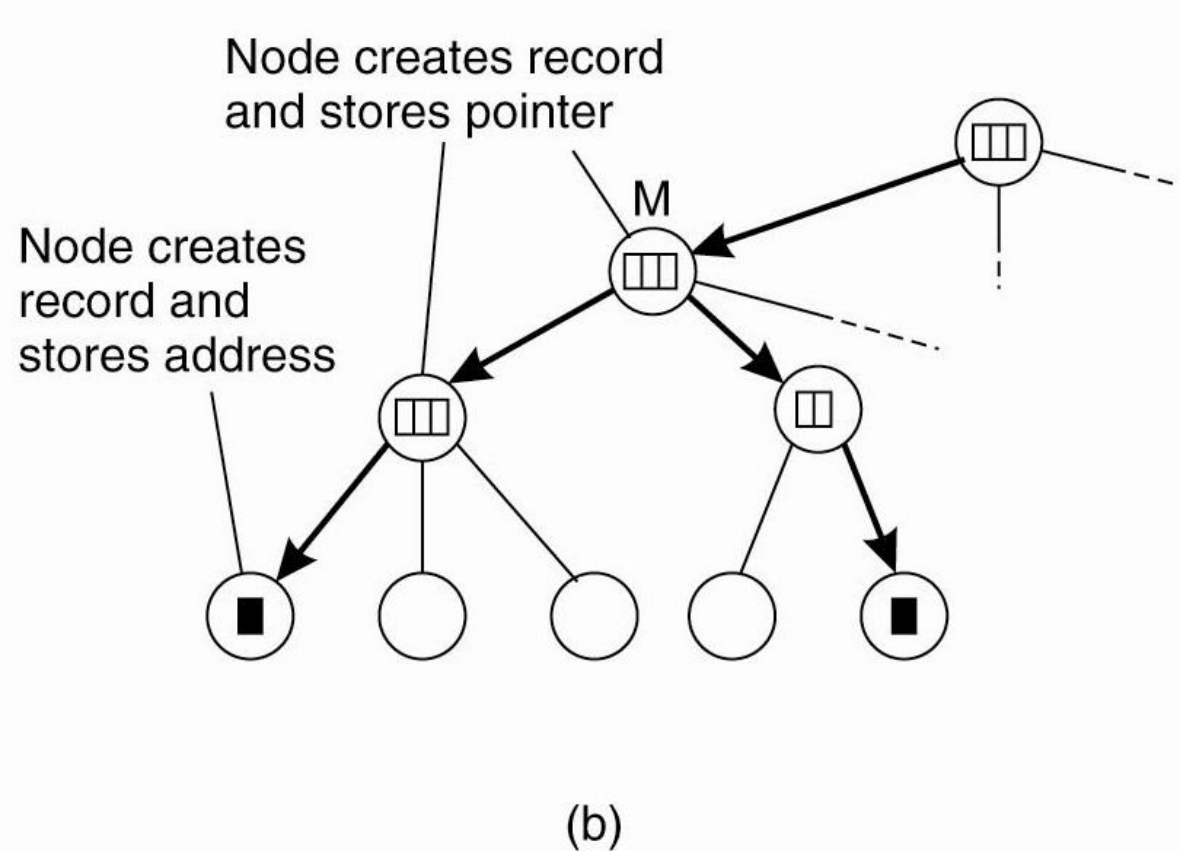

Figure 5-8. (b) A chain of forwarding pointers to the leaf node is created.

Structured Naming

- Flat names are good for machines but are not very convenient for humans to use.
- As an alternative, naming systems support **structured names** (human-readable names).
  - E.g., file names, host names, etc.
- This approach organizes names in **name spaces**.
  - Name spaces can be represented as a labeled, directed graph with two types of nodes:
    - **leaf node**: represents a named entity and stores information on the entity it is representing (e.g., IP address).
    - **directory node**: has a number of outgoing edges referring to other nodes.
  - The directory node has a table <node identifier, edge label>

Name Spaces (1)

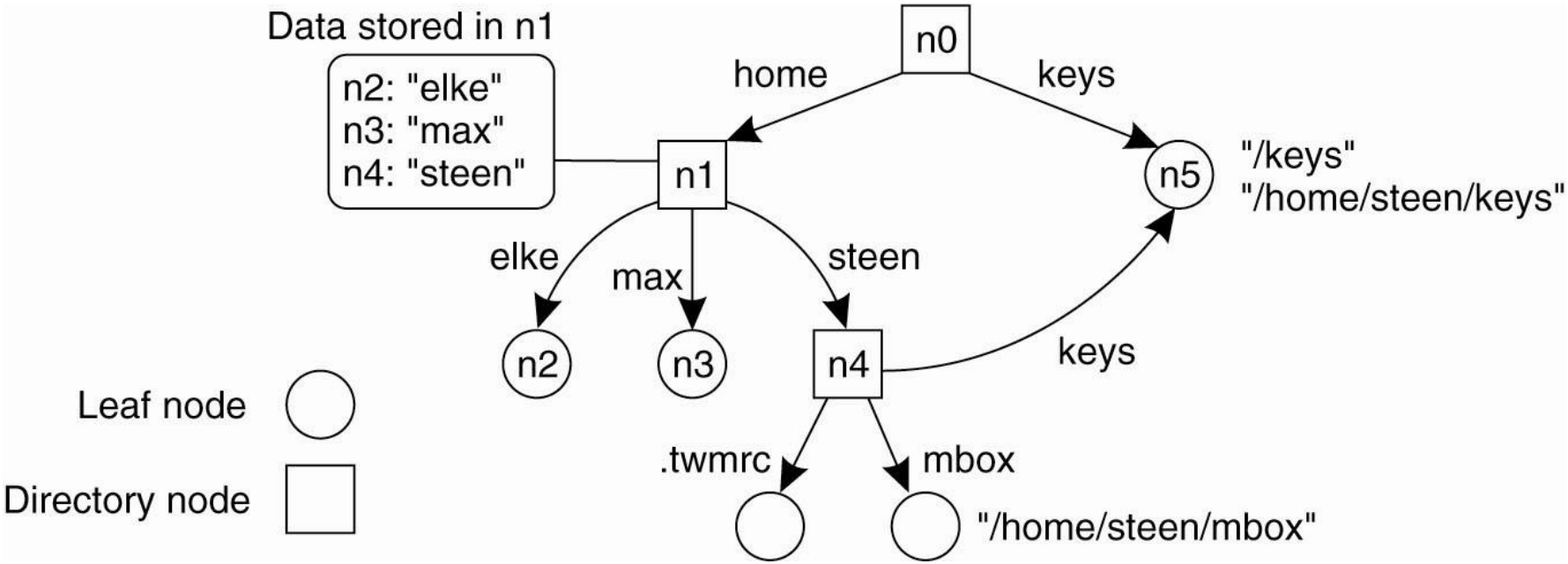

Figure 5-9. A general naming graph with a single root node.

Name Spaces (2)

- The naming graph has usually one **root node**.
- Each path in a naming graph can be referred to by the sequence of edges' labels in the path:
  - **N**:[$label_1$, $label_2$, …, $label_n$]; **N** refers to the 1st node in the path name.
  - If the first node in a pathname is the root, it is called **absolute path name**. Otherwise, it is called **relative pathname**.

Name Resolution

- Name spaces offer a mechanism for storing and retrieving information about entities by means of names.
  - Given a path name, it should be possible to look up any information stored in the node referred to by that name.
  - The process of look up a name is called **name resolution**.
- Name resolution can take place only if we know how and where to start.
  - Knowing how and where to start name resolution is referred to **closure mechanism**.
  - The **closure mechanism** requires to know the context of the name resolution.
    - www.ufu.br, resolution starts at the DNS root server (.br).
    - /home/johndoe/mbox, resolution start at the root node of the file system, which is known.

Linking and Mounting (1)

- Linking
  - An **alias** is another name for the same entity.
  - It can be implemented by **hard links** or **symbolic links**.
    - **Hard links**: multiple absolute paths names to refer to the same node in a naming graph.
    - **Symbolic links**: represent an entity by a leaf node, but instead of storing the address or state of that entity, the node stores an absolute path name to another node.

Linking and Mounting (2)

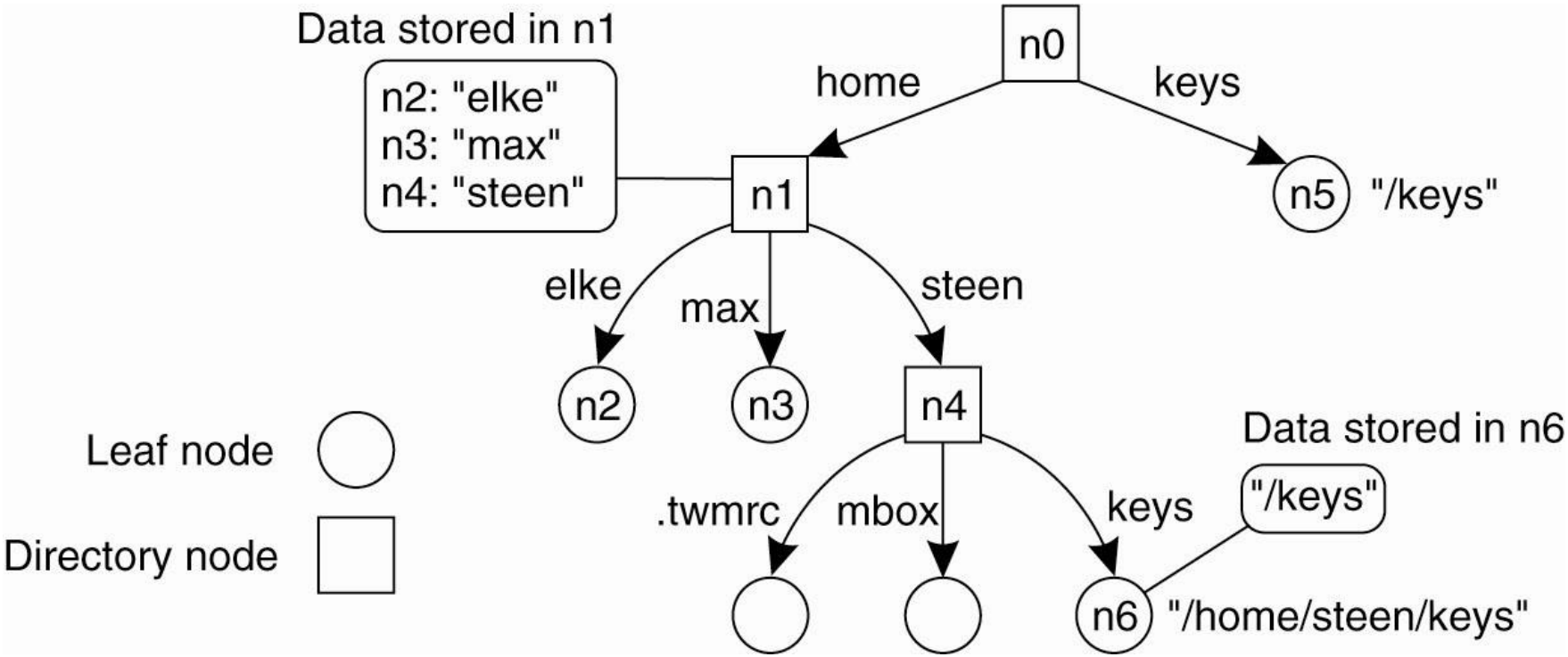

Figure 5-11. The concept of a symbolic link explained in a naming graph.

Linking and Mounting (3)

- So far, **name resolution** takes place within a single name space.
  - It can also be used to merge different name spaces, transparently.
  - This process is called **mounting**.
- Information required to mount a **foreign name space** in a distributed system
  - The name of an access protocol.
  - The name of the server.
  - The name of the mounting point in the foreign name space.
- The mounting point (foreign) is mounted on the mount point (local).

Linking and Mounting (4)

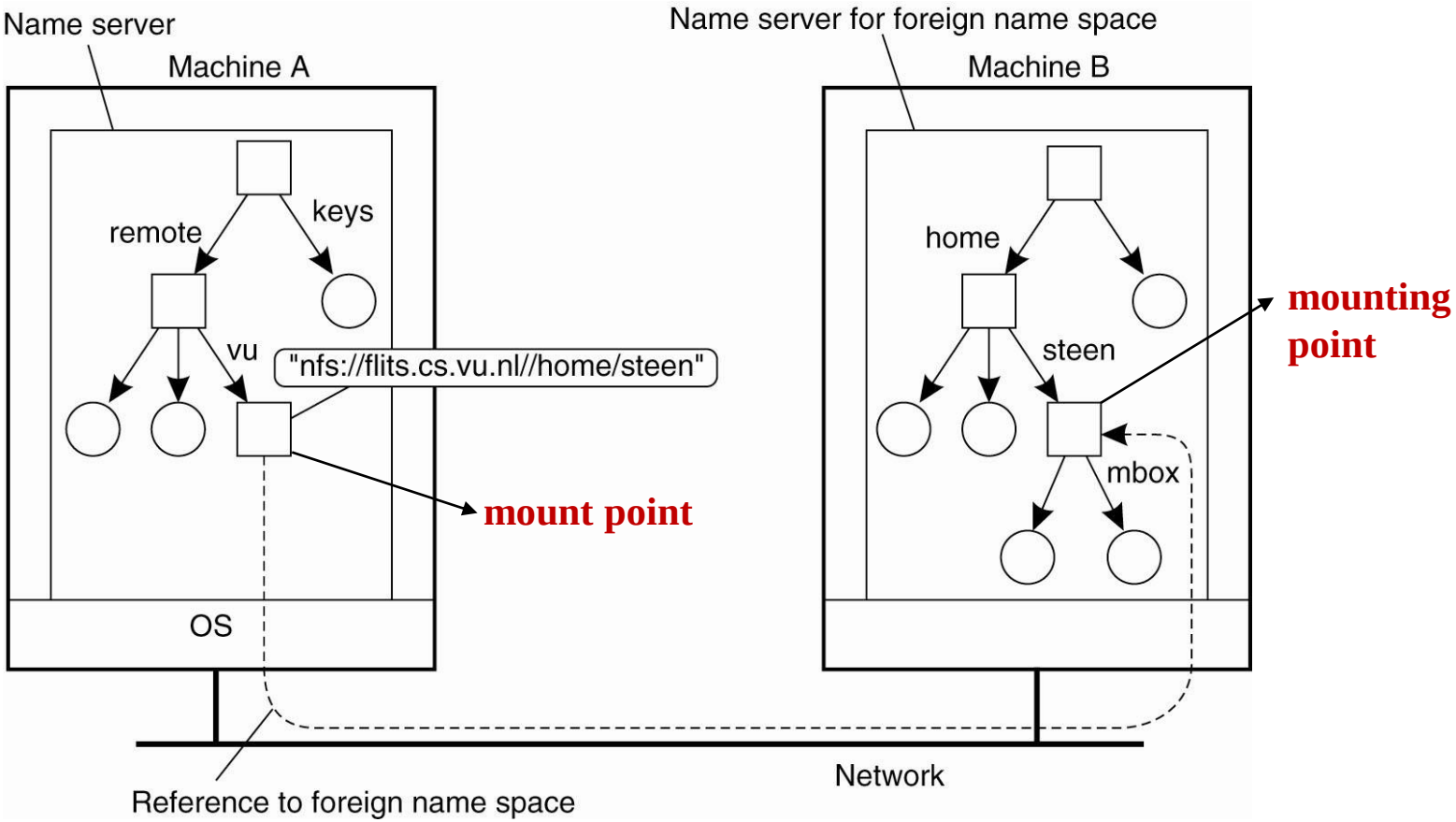

Figure 5-12. Mounting remote name spaces through a specific access protocol.

Name Spaces: Implementation

- A name space forms the core of a naming service.
  - This service allows users and processes to **add**, **remove**, and **look up** names.
  - A naming service is implemented by **name servers**.
- A DS limited to a LAN can implement a naming service as a single server.
  - However, in large-scale DS, the name space should be distributed over multiple name servers.
  - In these cases, the name space is usually organized hierarchically.

Name Spaces: Implementation

- In large name spaces, it is convenient to partition it into logical layers:
  - **Global layer**
    - Root node and its children nodes.
    - Characterized by nodes' stability (directory's tables are rarely changed).
    - May represent organizations.
  - **Administrational layer**
    - Is formed of directory nodes.
    - These nodes represent groups of entities within an organization.
    - Moderate stability.
  - **Managerial layer**
    - Consists of nodes that may typically change regularly.
    - Nodes representing local hosts, users, etc.

Name Space Distribution (1)

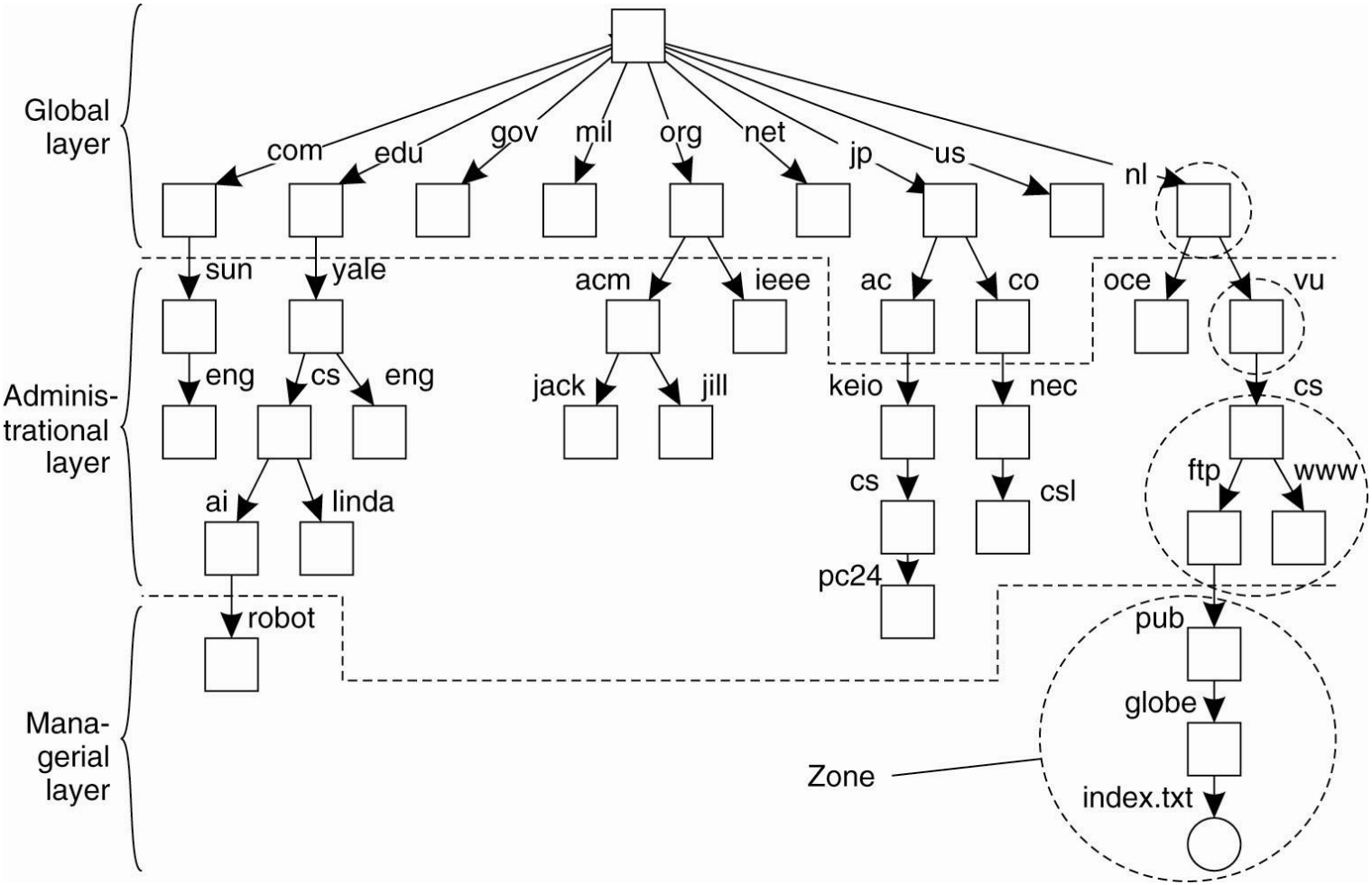

Figure 5-13. An example partitioning of the DNS name space, including Internet-accessible files, into three layers.

Name Space Distribution (2)

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

Figure 5-14. 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.

Name resolution: Implementation

- The distribution of a name space across multiple name servers affects the name resolution implementation.
- Each client has access to a local **name resolver**.
  - It is responsible for ensuring that the name resolution process is carried out.
- Two approaches:
  - **Iterative name resolution**
  - **Recursive name resolution**

Implementation of Name Resolution (1)

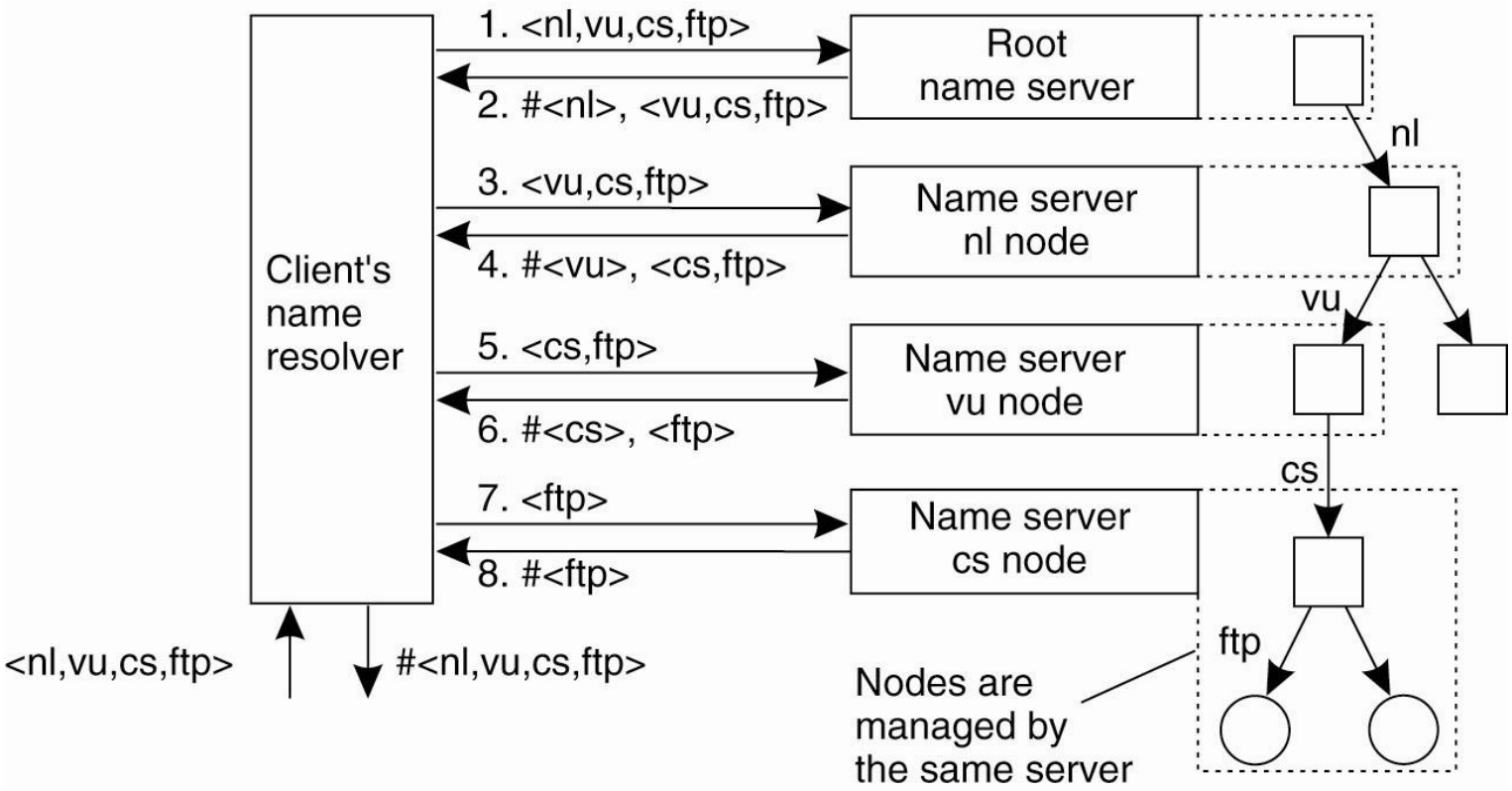

Figure 5-15. The principle of iterative name resolution.

Implementation of Name Resolution (2)

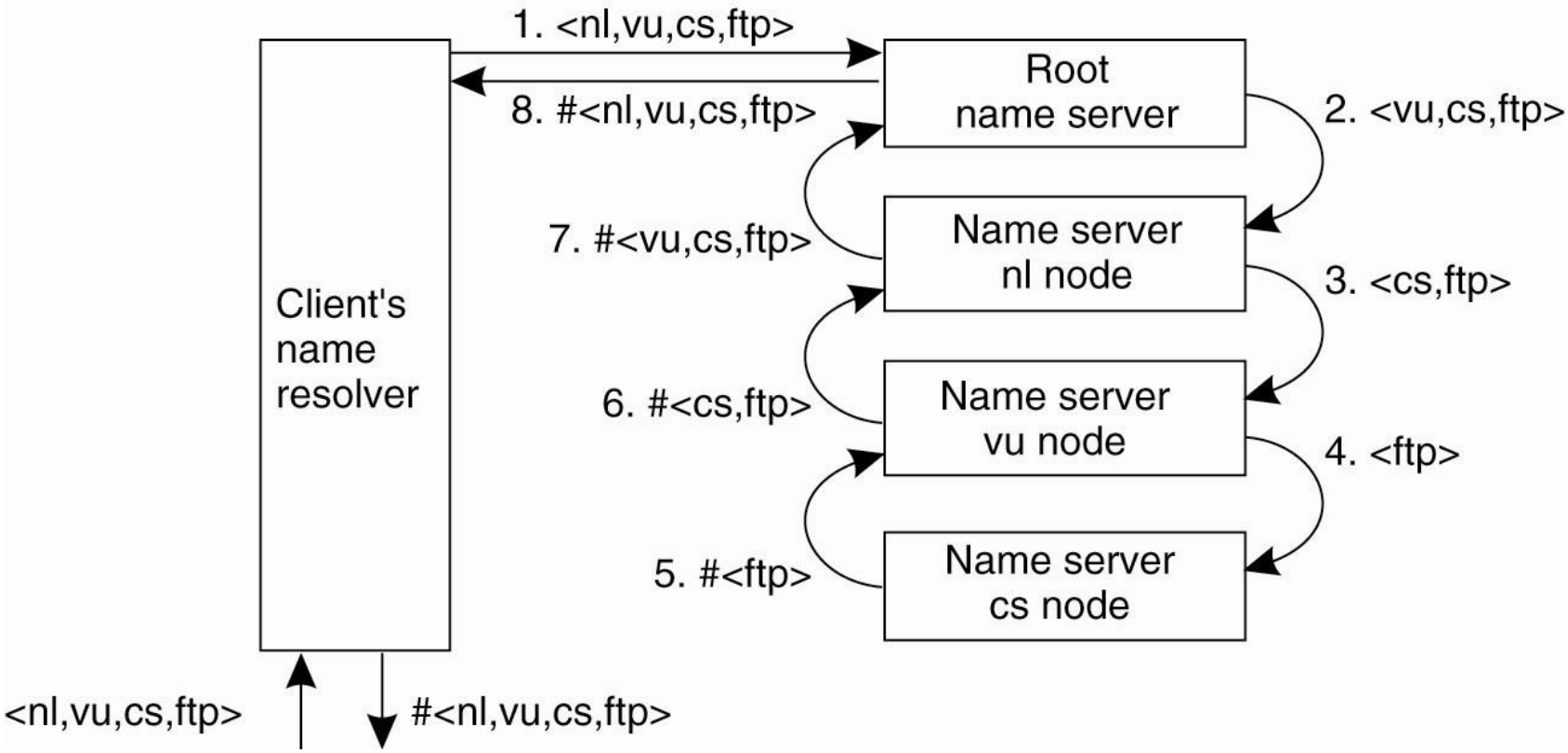

Figure 5-16. The principle of recursive name resolution.

Implementation of Name Resolution (3)

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>

Figure 5-17. Recursive name resolution of <*nl, vu, cs, ftp*>. Name servers cache intermediate results for subsequent lookups.

Example: The Domain Name System

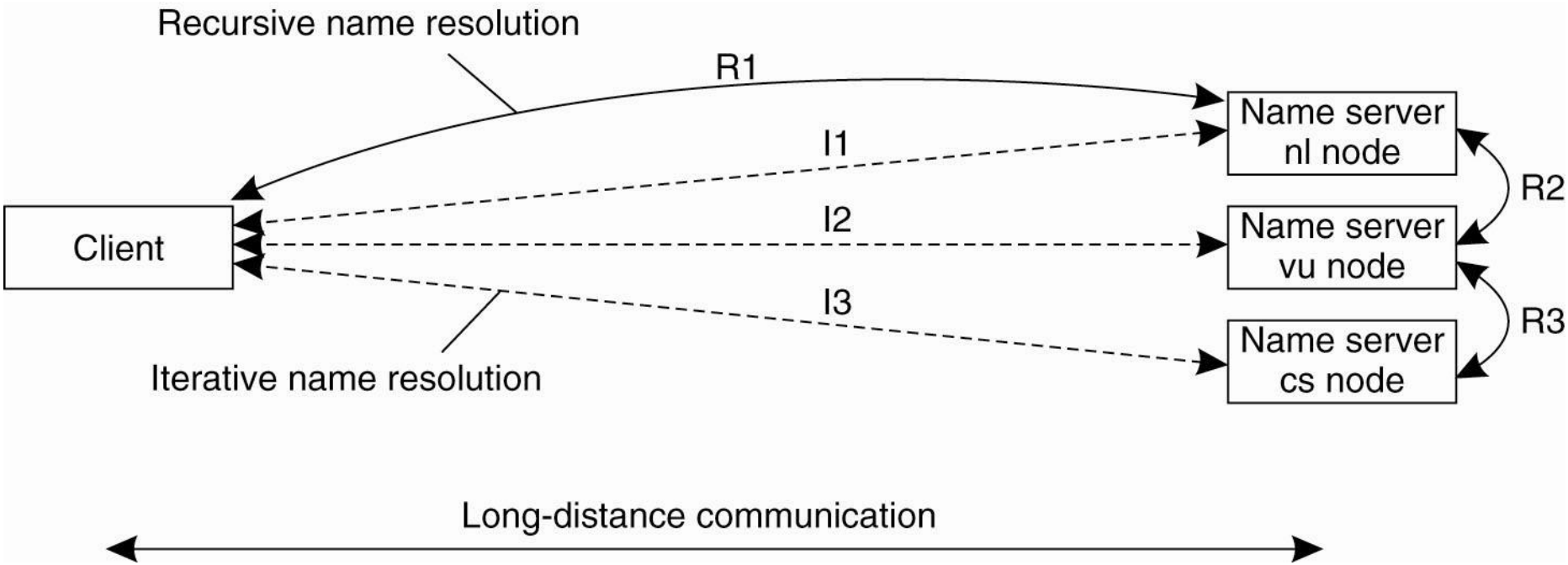

Figure 5-18. The comparison between recursive and iterative name resolution with respect to communication costs.

The DNS Name Space

Type of record	Associated entity	Description
SOA	Zone	Holds information on the represented zone
A	Host	Contains an IP address of the host this node represents
MX	Domain	Refers to a mail server to handle mail addressed to this node
SRV	Domain	Refers to a server handling a specific service
NS	Zone	Refers to a name server that implements the represented zone
CNAME	Node	Symbolic link with the primary name of the represented node
PTR	Host	Contains the canonical name of a host
HINFO	Host	Holds information on the host this node represents
TXT	Any kind	Contains any entity-specific information considered useful

Figure 5-19. The most important types of resource records forming the contents of nodes in the DNS name space.

DNS Implementation (1)

Name	Record type	Record value
cs.vu.nl.	SOA	star.cs.vu.nl. hostmaster.cs.vu.nl. 2005092900 7200 3600 2419200 3600
cs.vu.nl.	TXT	"Vrije Universiteit - Math. & Comp. Sc."
cs.vu.nl.	MX	1 mail.few.vu.nl.
cs.vu.nl.	NS	ns.vu.nl.
cs.vu.nl.	NS	top.cs.vu.nl.
cs.vu.nl.	NS	solo.cs.vu.nl.
cs.vu.nl.	NS	star.cs.vu.nl.
star.cs.vu.nl.	A	130.37.24.6
star.cs.vu.nl.	A	192.31.231.42
star.cs.vu.nl.	MX	1 star.cs.vu.nl.
star.cs.vu.nl.	MX	666 zephyr.cs.vu.nl.
star.cs.vu.nl.	HINFO	"Sun" "Unix"
zephyr.cs.vu.nl.	A	130.37.20.10
zephyr.cs.vu.nl.	MX	1 zephyr.cs.vu.nl.
zephyr.cs.vu.nl.	MX	2 tornado.cs.vu.nl.
zephyr.cs.vu.nl.	HINFO	"Sun" "Unix"

Figure 5-20. An excerpt from the DNS database for the zone *cs.vu.nl*.

DNS Implementation (2)

ftp.cs.vu.nl.	CNAME	soling.cs.vu.nl.
www.cs.vu.nl.	CNAME	soling.cs.vu.nl.
soling.cs.vu.nl.	A	130.37.20.20
soling.cs.vu.nl.	MX	1 soling.cs.vu.nl.
soling.cs.vu.nl.	MX	666 zephyr.cs.vu.nl.
soling.cs.vu.nl.	HINFO	"Sun" "Unix"
vucs-das1.cs.vu.nl.	PTR	0.198.37.130.in-addr.arpa.
vucs-das1.cs.vu.nl.	A	130.37.198.0
inkt.cs.vu.nl.	HINFO	"OCE" "Proprietary"
inkt.cs.vu.nl.	A	192.168.4.3
pen.cs.vu.nl.	HINFO	"OCE" "Proprietary"
pen.cs.vu.nl.	A	192.168.4.2
localhost.cs.vu.nl.	A	127.0.0.1

Figure 5-20. An excerpt from the DNS database for the zone *cs.vu.nl*.

Attribute-based Naming (1)

- Provides a richer way to describe entities in terms of (*attribute*, *value*) pairs.
- In this approach, an **entity** is assumed to have an associated collection of attributes.
  - Each attribute says something about that entity.
  - By specifying which values a specific attribute should have, a user essentially constrains the set of entities that he is interested in.
  - It is up to the naming system to return one or more entities that meet the user's description.

Attribute-based Naming (2)

- Attribute-based naming systems are also known as **directory services**
  - Systems that support flat and structured naming are generally called **naming systems**.
  - With **directory services**, entities have a set of associated attributes that can be used for searching.
  - For example, a **host** can have attributes like Locality (Country), Organization, Organizational unit, Common name, Host name, Host address, etc.
- Directory services return one or more entities that match with the input values provided by users.
  - E.g., search("(C=BR)(O=UFU)(OU=FACOM)(CN=*")

Attribute-based Naming (3)

- An important aspect of DS is designing an appropriate set of attributes:
  - This is not a trivial task! Mostly it is done manually.
- Research has been conducted on unifying the way resources (entities) can be described.
  - In this context, an important development is the resource description framework (RDF).
  - In RDF, resources are described as triplets consisting of a **subject**, **predicated** and the **object**.
  - E.g.: (Person, name, Alice) describes a resource named Person whose name is Alice.
  - References in RDF are essentially URLs.

Attribute-based Naming (4)

- Resource descriptions must be stored.
  - They are queried by applications.
  - Hence, they must be stored, centralized or distributed.
- Distributed storage of resource descriptions:
  - Unlike flat/structured naming, looking up values in an attribute-based naming system requires an exhaustive search through all descriptors.
    - Various techniques can be applied to avoid such exhaustive searches, one obvious being indexing.
  - When considering performance, an exhaustive search may be less of problem within a single, nondistributed data store
    - but sending a search query to hundreds of servers that jointly implement a distributed data store is generally not such a good idea.

Attribute-based Naming (5)

- A common approach to tackling distributed directory services is to combine structured naming with attribute-based naming.
  - This approach has been widely adopted, for example, in Microsoft's Active Directory service and other systems.
  - Many of these systems use or rely on the **LDAP** (Lightweight Directory Access Protocol)
- Conceptually, an LDAP directory service consists:
  - Of a number of records (directory entries, entities descriptions).
  - Each record is made up of a collection of (attribute, value) pairs.
  - Each attributed has an associated type.

LDAP Directory Entry

Attribute	Abbr.	Value
Country	C	NL
Locality	L	Amsterdam
Organization	O	Vrije Universiteit
OrganizationalUnit	OU	Comp. Sc.
CommonName	CN	Main server
Mail_Servers	—	137.37.20.3, 130.37.24.6, 137.37.20.10
FTP_Server	—	130.37.20.20
WWW_Server	—	130.37.20.20

Figure 5-22. A simple example of an LDAP directory entry using LDAP naming conventions.

Hierarchical Implementations: LDAP (1)

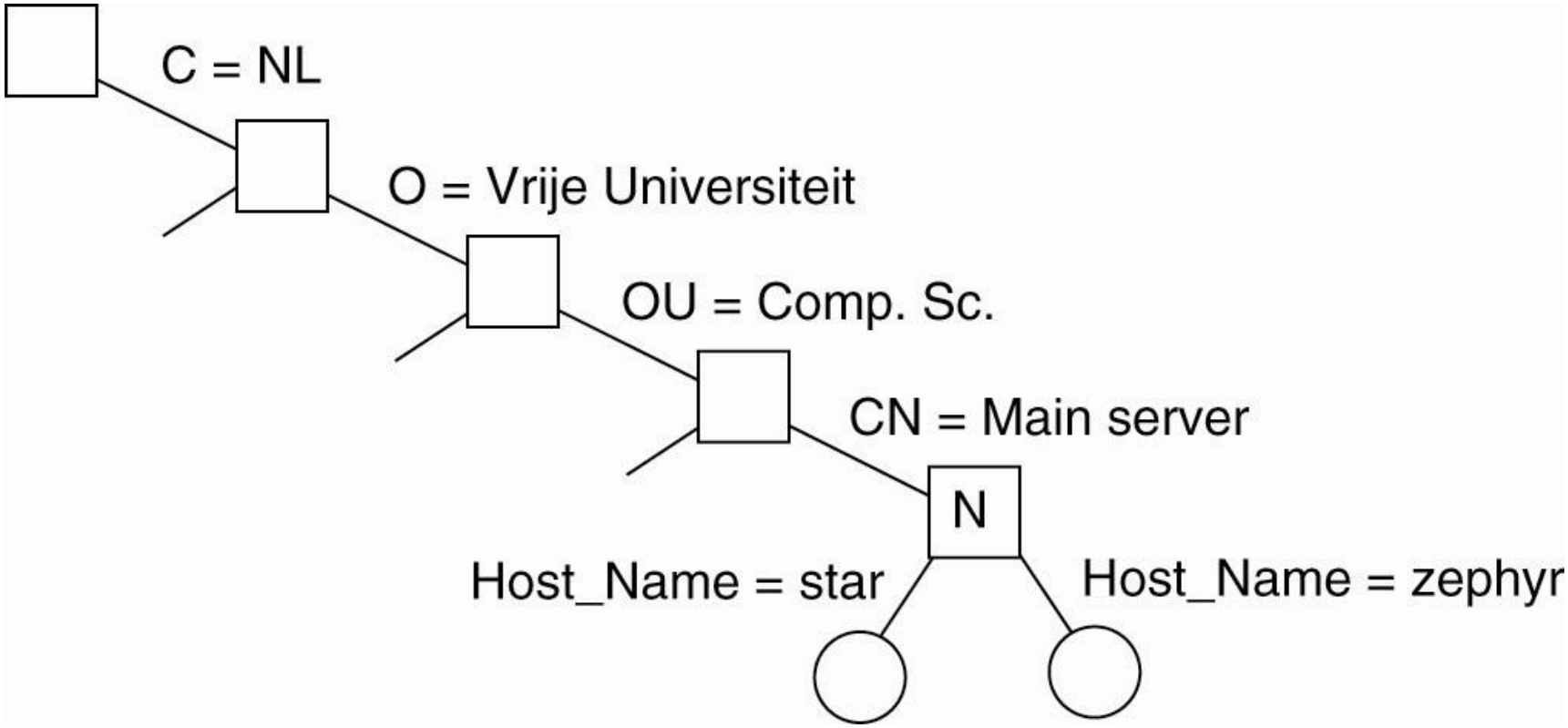

Figure 5-23. (a) Part of a directory information tree.

Hierarchical Implementations: LDAP (2)

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

(b)

Figure 5-23. (b) Two directory entries having *Host_Name* as RDN.