DISTRIBUTED SYSTEMS
Principles and Paradigms
Second Edition

ANDREW S. TANENBAUM
MAARTEN VAN STEEN

Bearbeitung von Matthias Wallnöfer

Naming

Names, Identifiers, And Addresses

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

Forwarding Pointers (1)

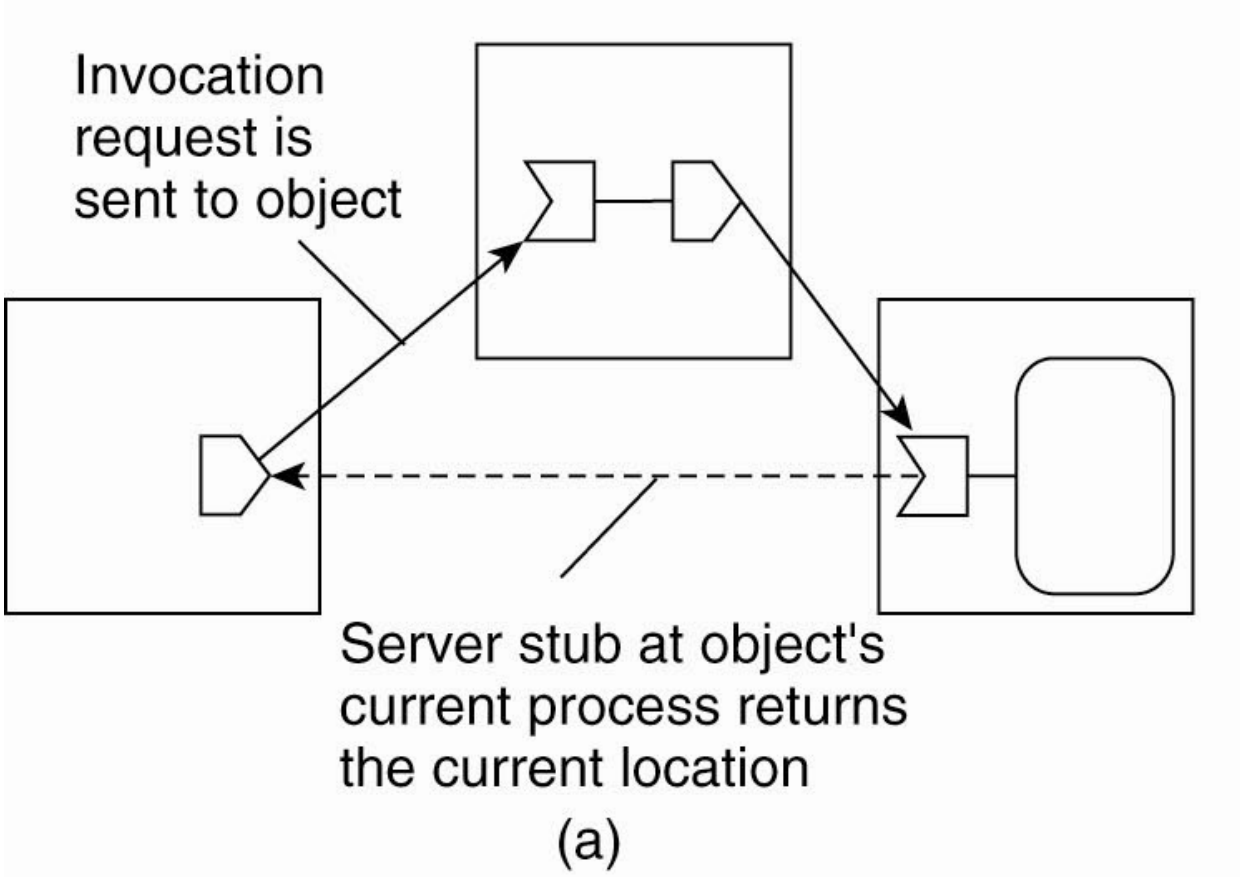


Figure 5-2. Redirecting a forwarding pointer by storing a shortcut in a client stub (RMI).

Forwarding Pointers (2)

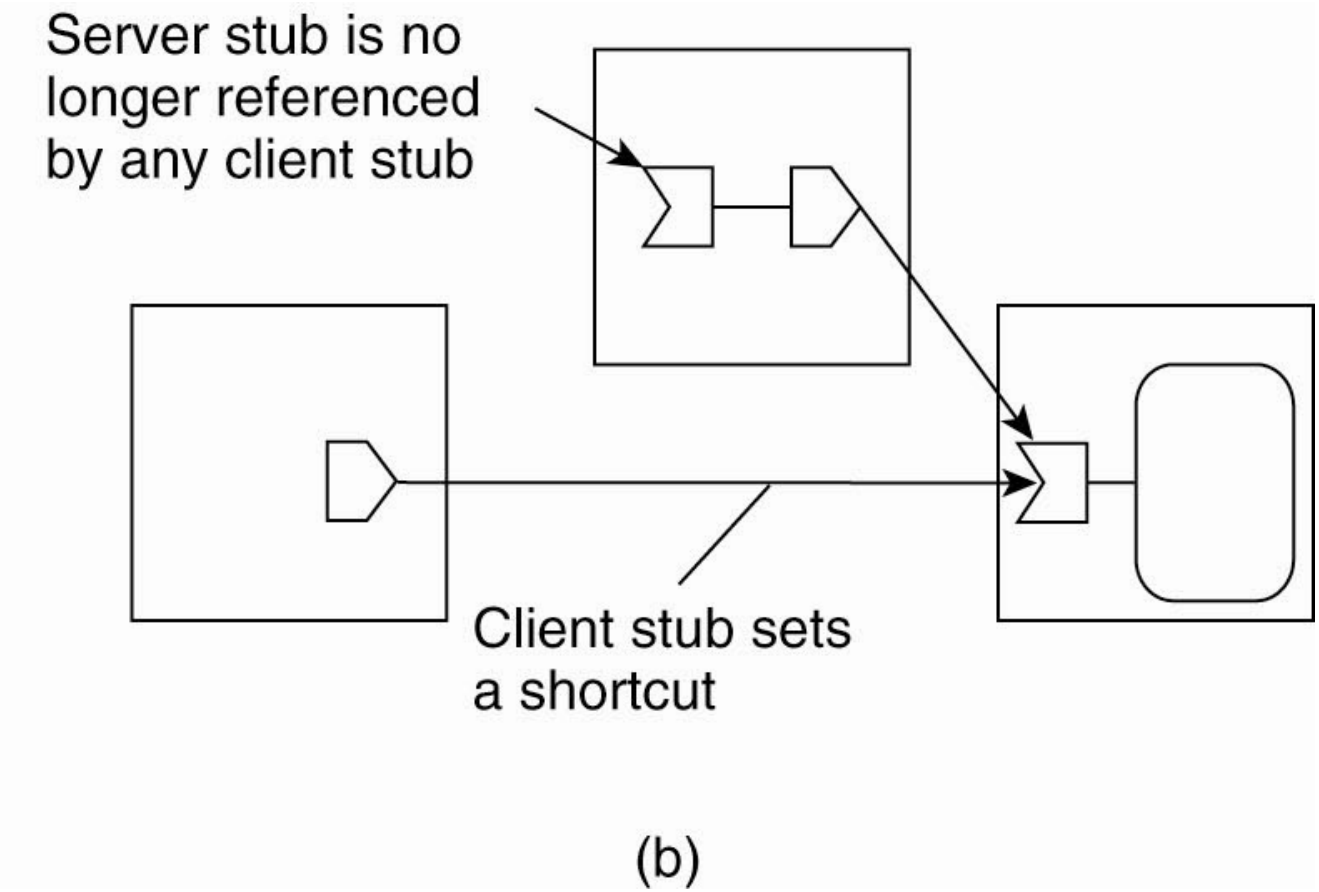


Figure 5-2. Redirecting a forwarding pointer by storing a shortcut in a client stub.

Home-Based Approaches

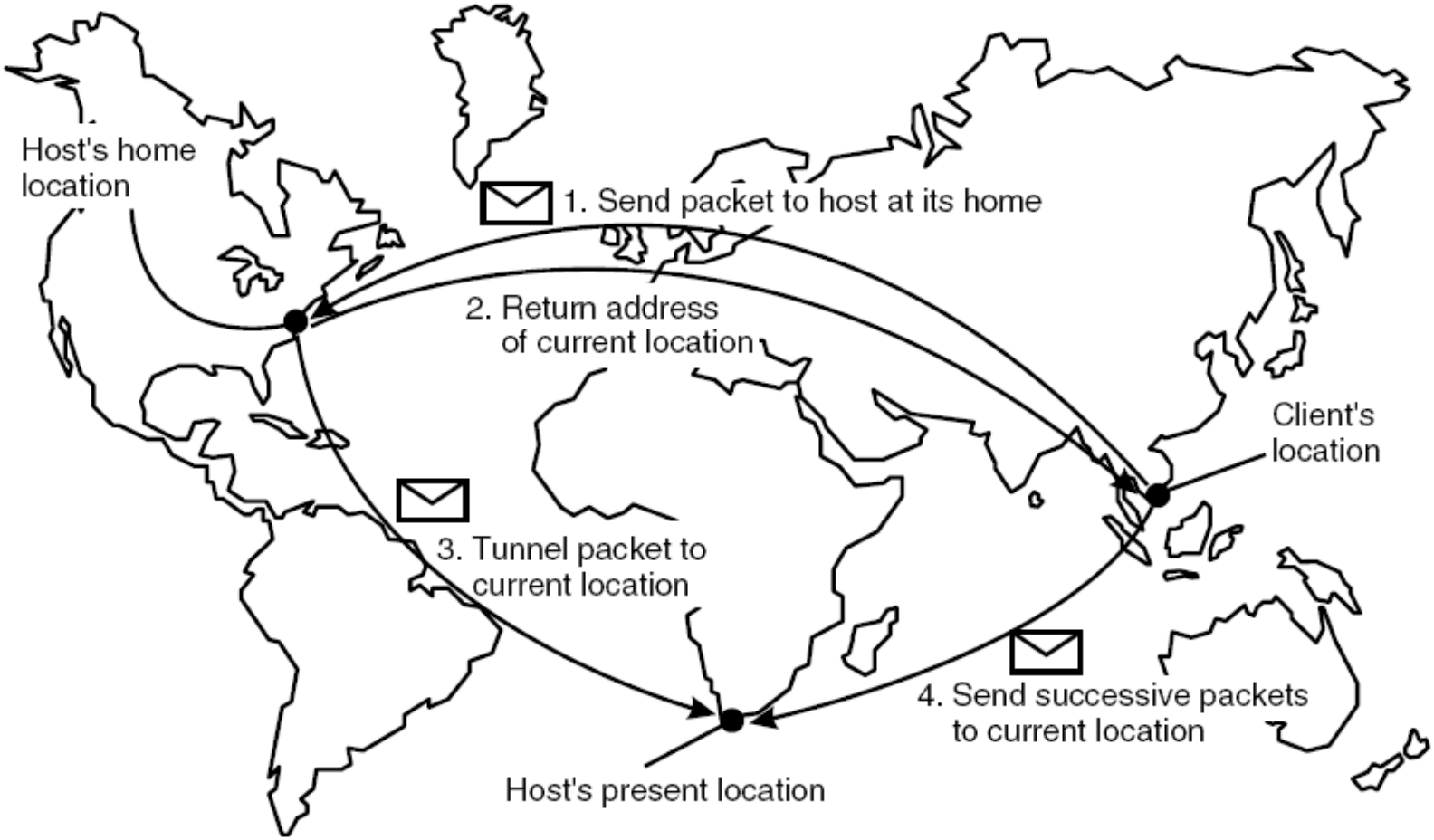


Figure 5-3. The principle of Mobile IP (IPv6 extension).

Distributed Hash Tables

General Mechanism

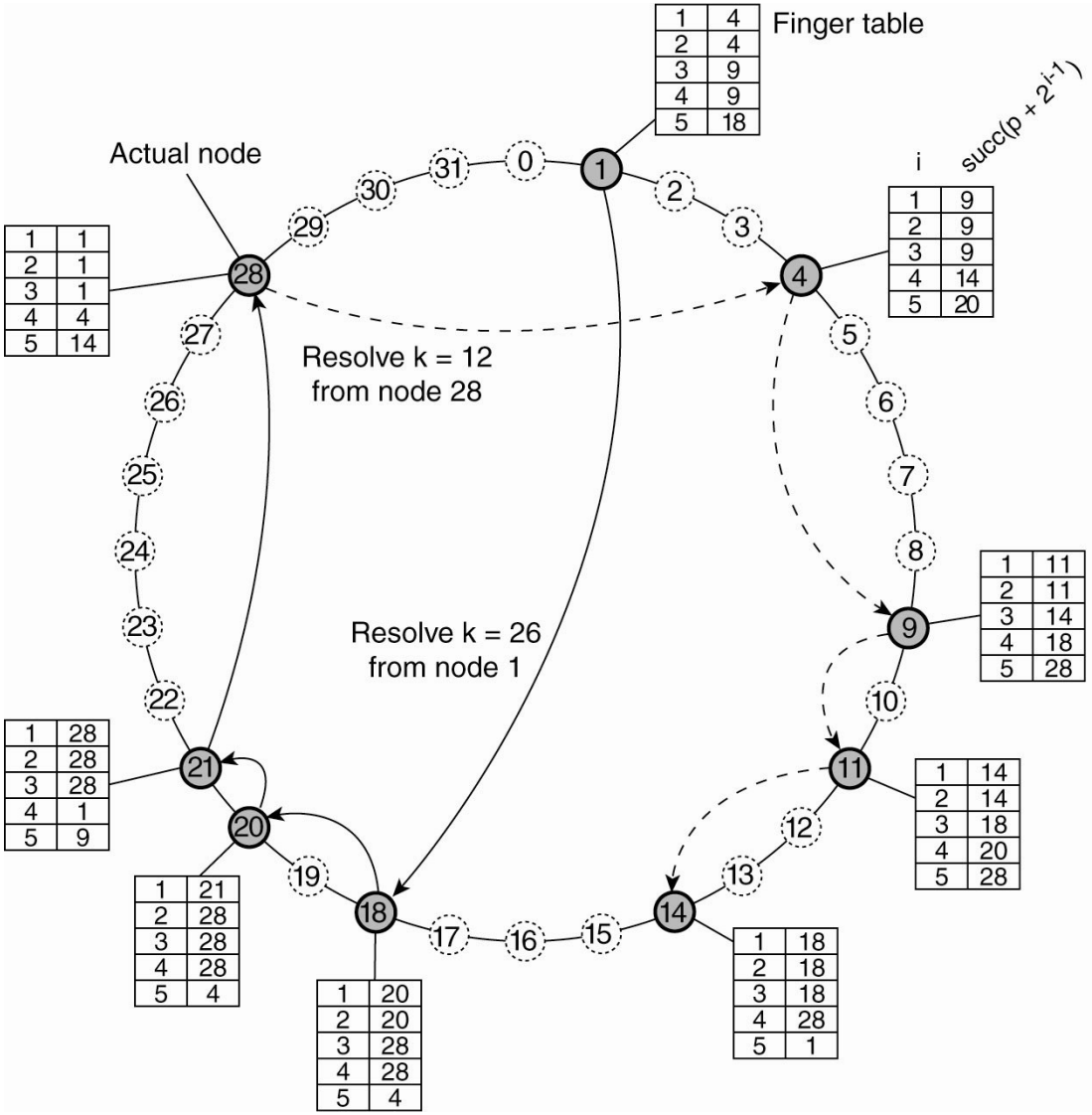


Figure 5-4. Resolving key 26 from node 1 and key 12 from node 28 in a Chord system.

Hierarchical Approaches (1)

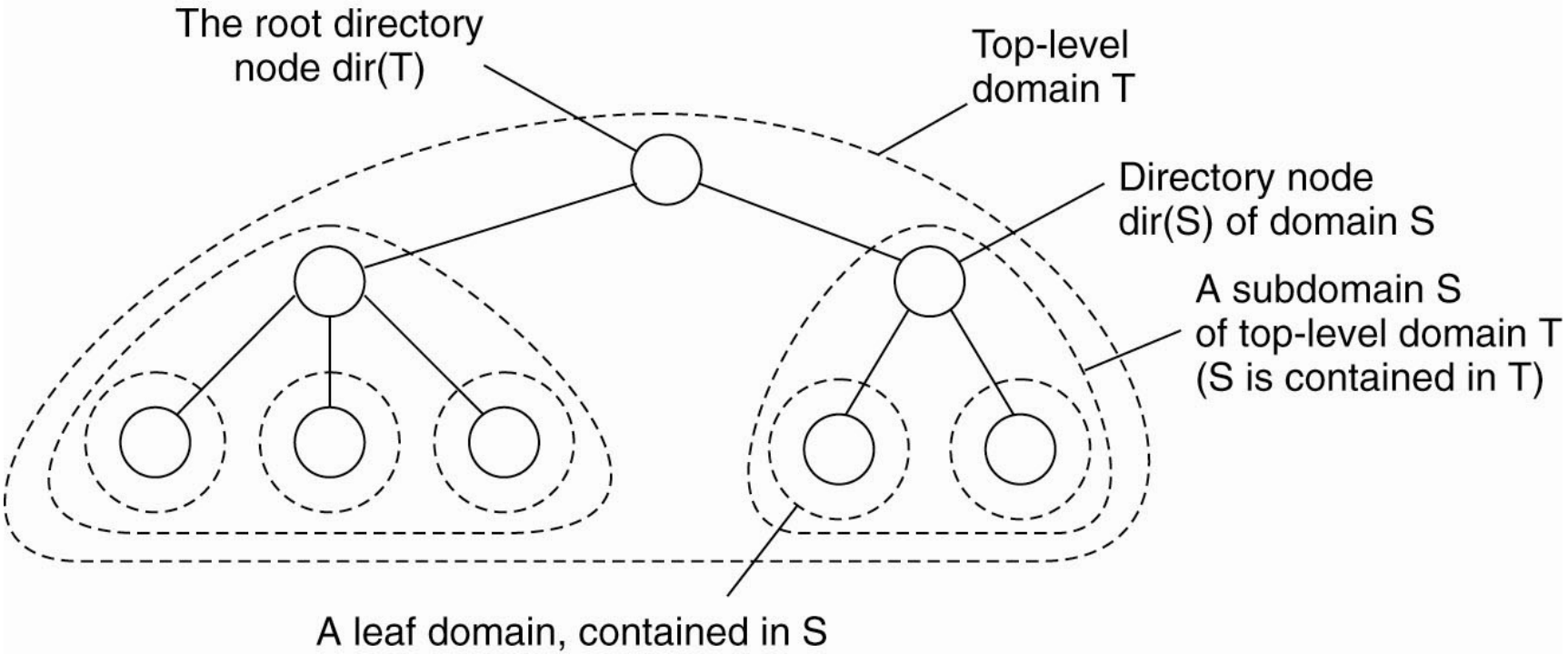


Figure 5-5. Hierarchical organization of a location service into domains, each having an associated directory node.

Hierarchical Approaches (2)

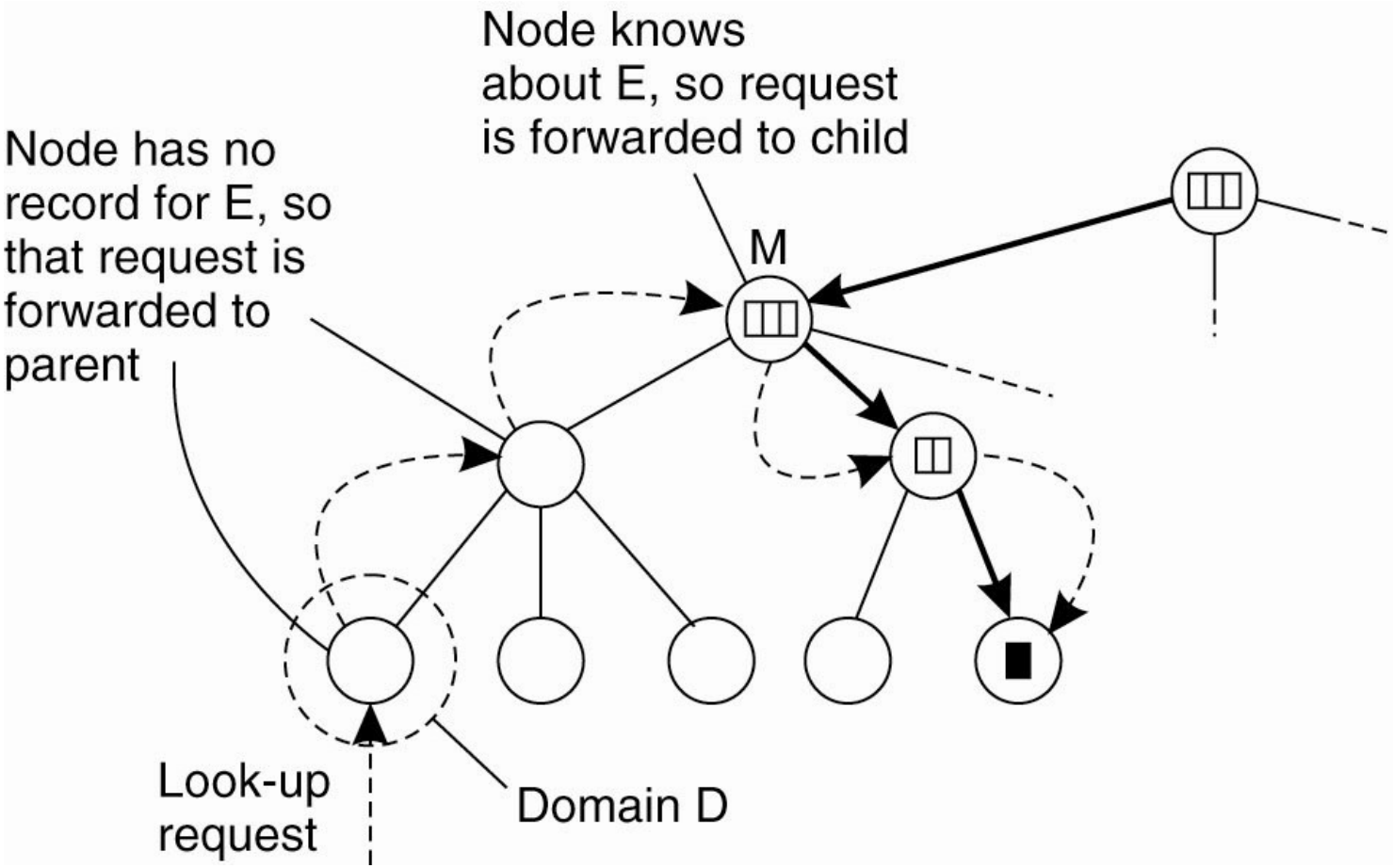


Figure 5-7. Looking up a location in a hierarchically organized location service.

Name Spaces

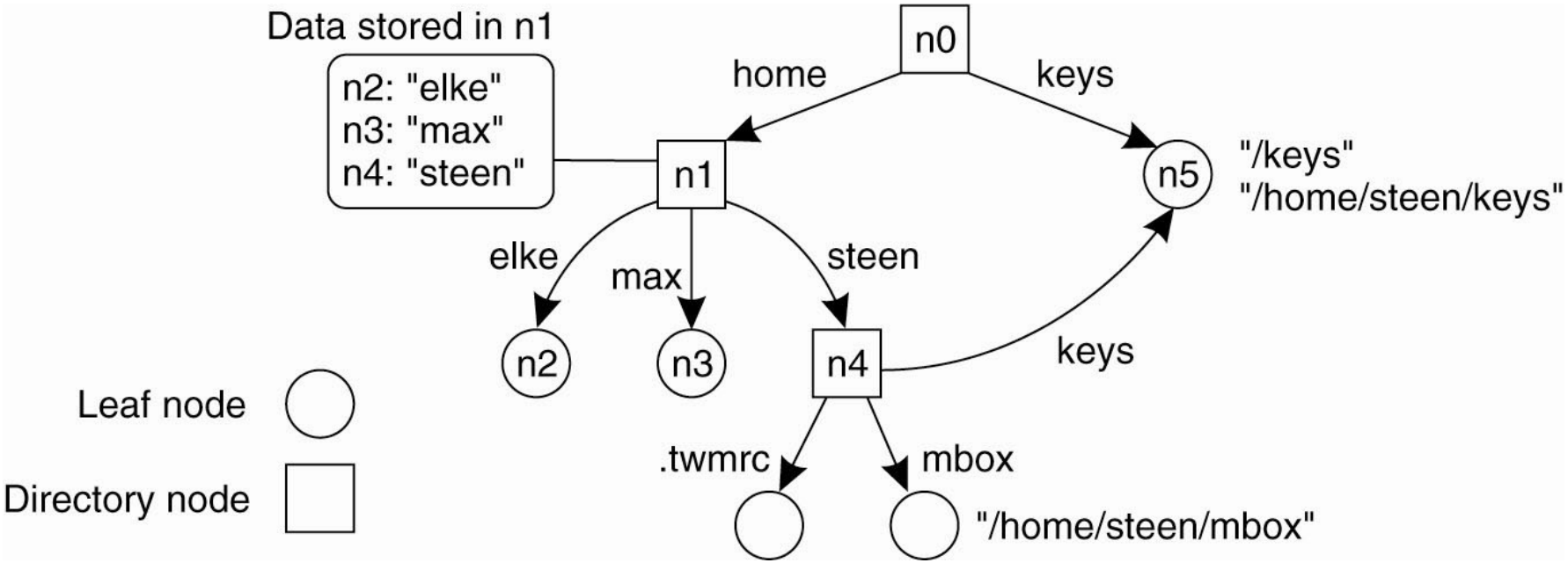


Figure 5-9. A general naming graph with a single root node. *n5* is an inode which is referenced two times → hard link

Linking and Mounting (1)

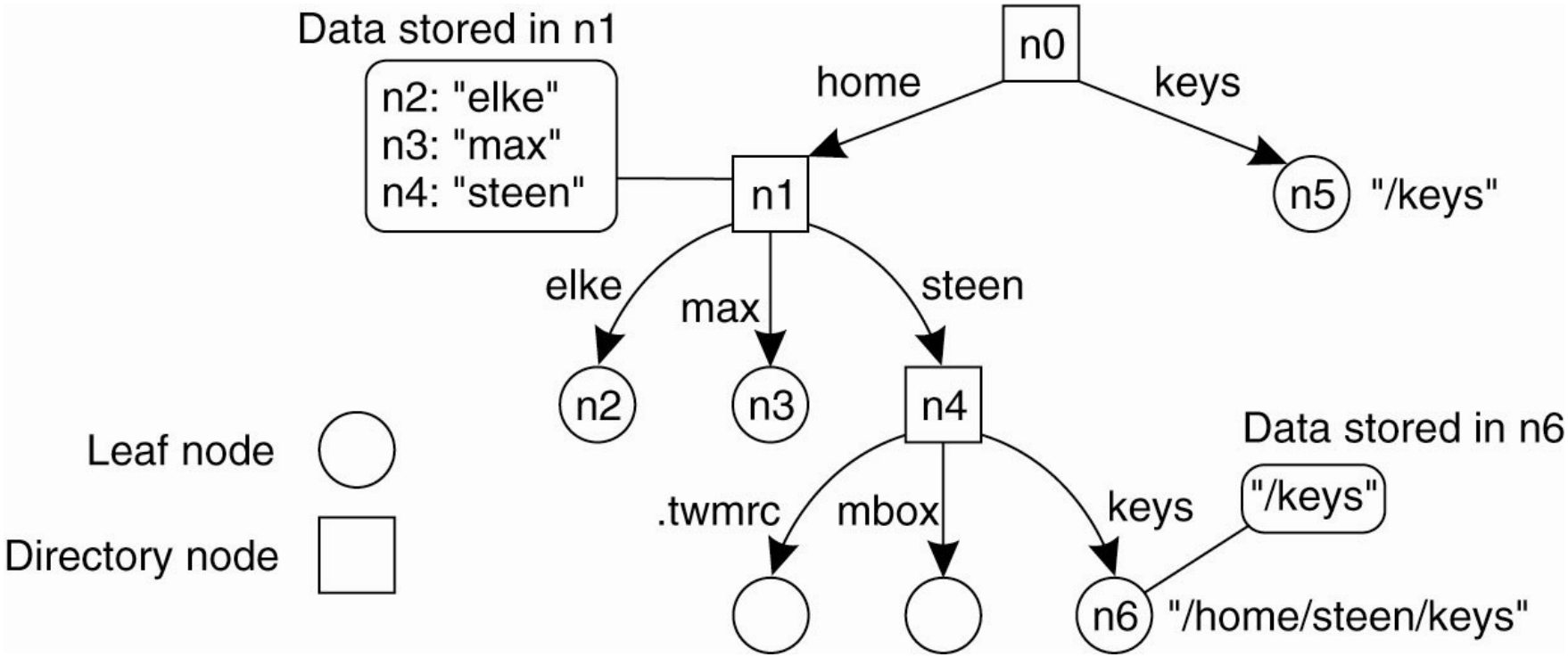


Figure 5-11. The concept of a symbolic link explained in a naming graph.

Linking and Mounting (2)

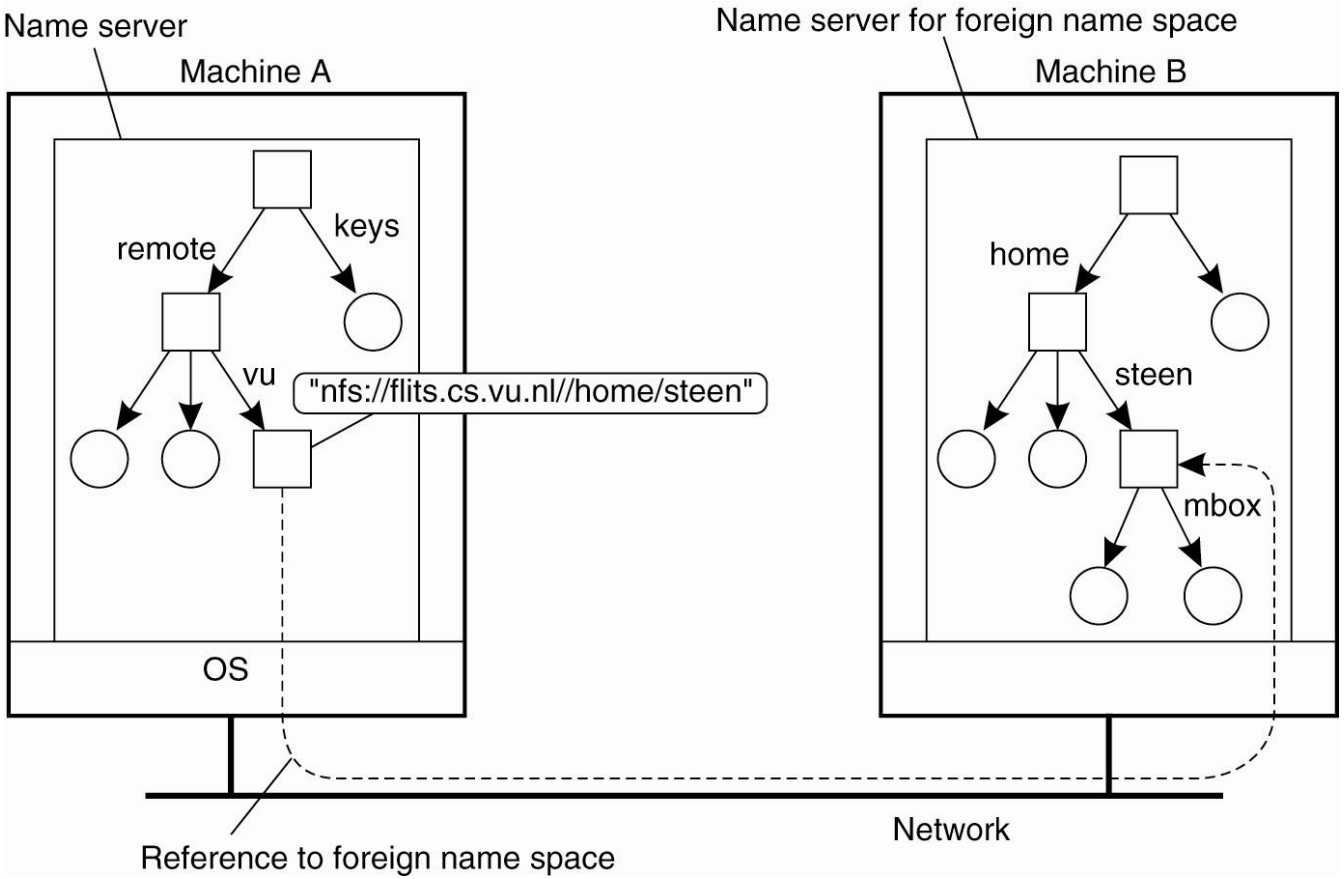


Figure 5-12. Mounting remote name spaces through a specific access protocol (NFS).

Linking and Mounting (3)

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.

Name Space Distribution

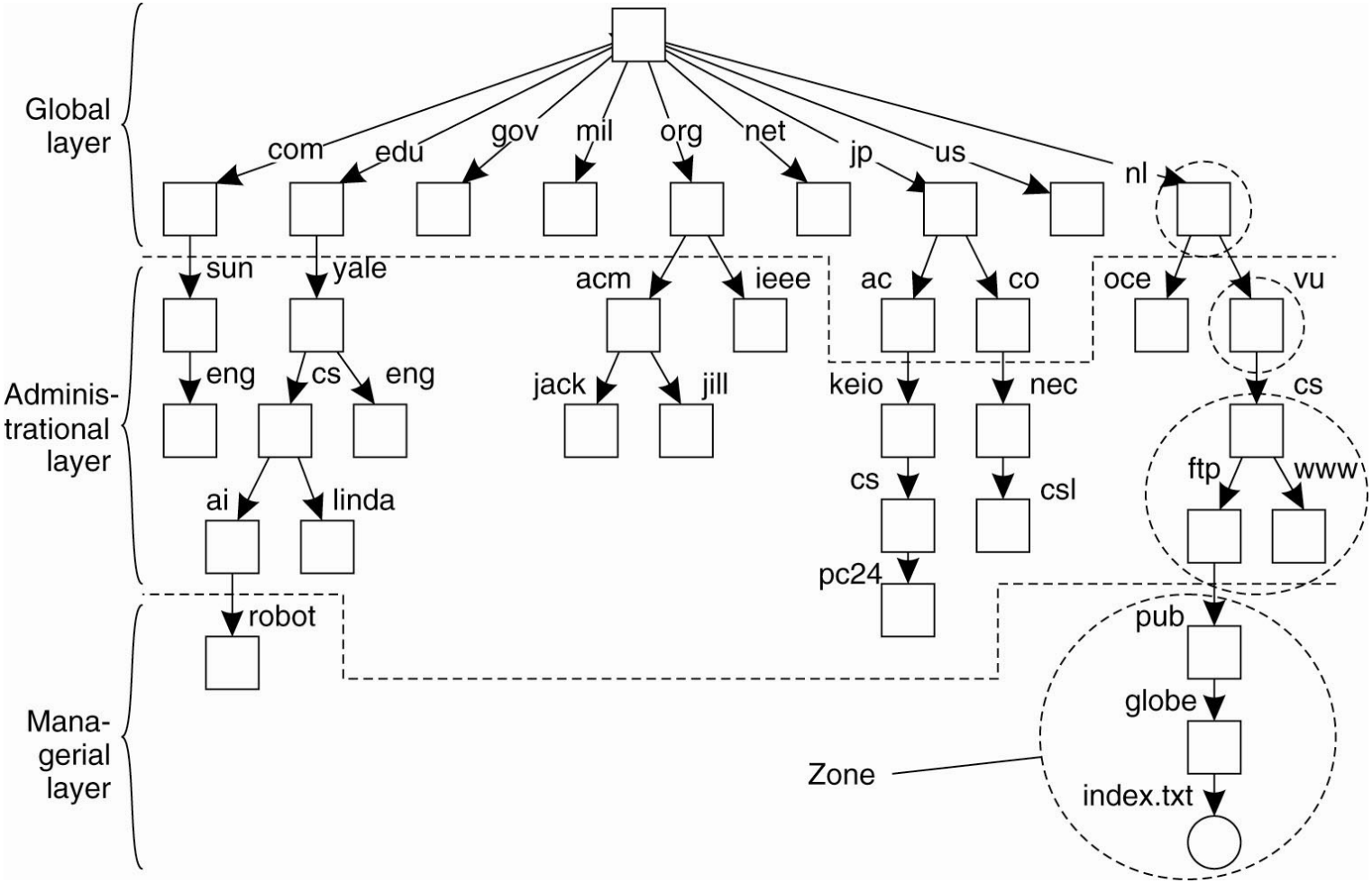


Figure 5-13. An example partitioning of the DNS name space, including Internet-accessible files, into three layers.

Implementation of Name Resolution (1)

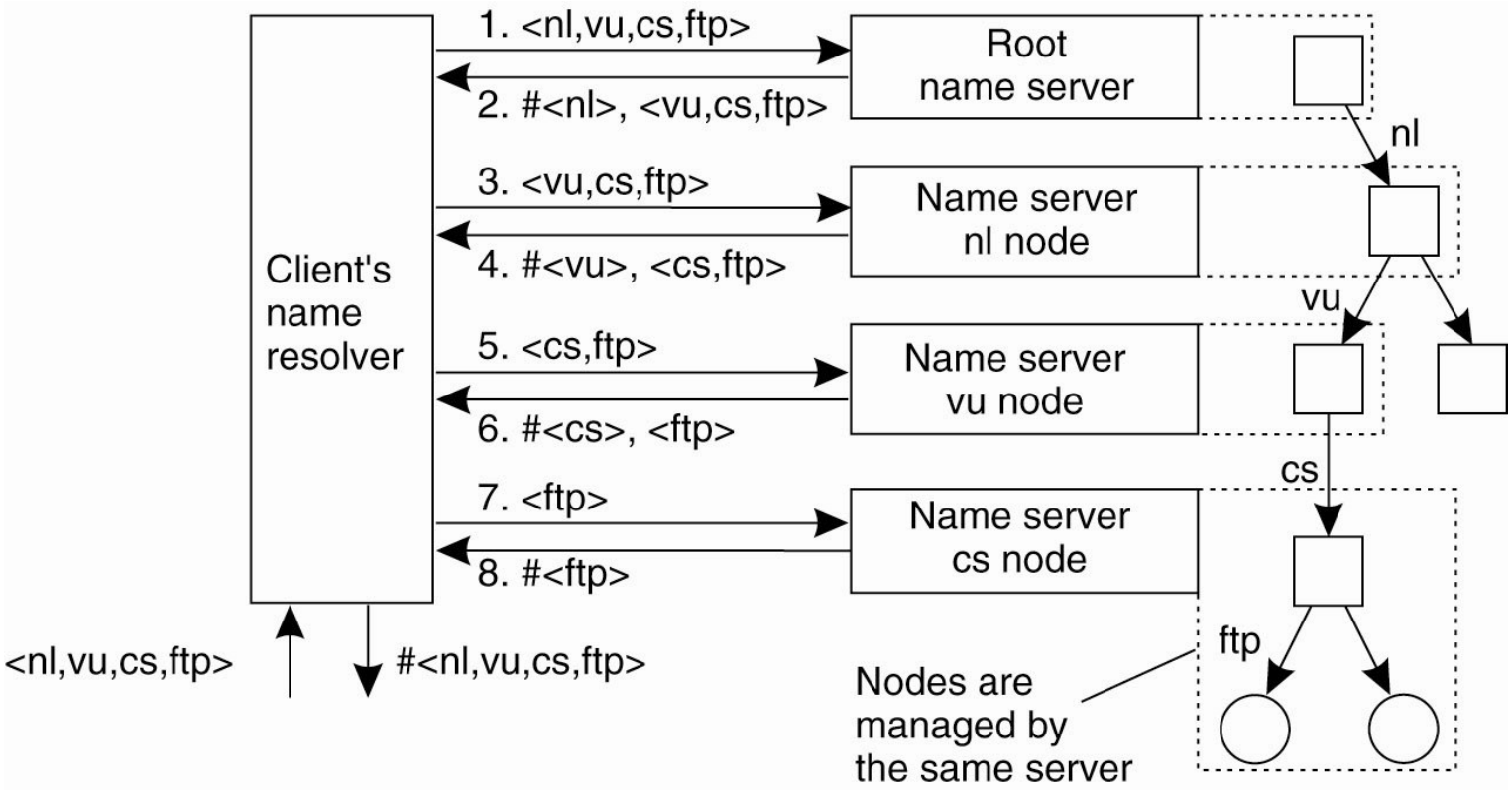


Figure 5-15. The principle of **iterative** name resolution.

Implementation of Name Resolution (2)

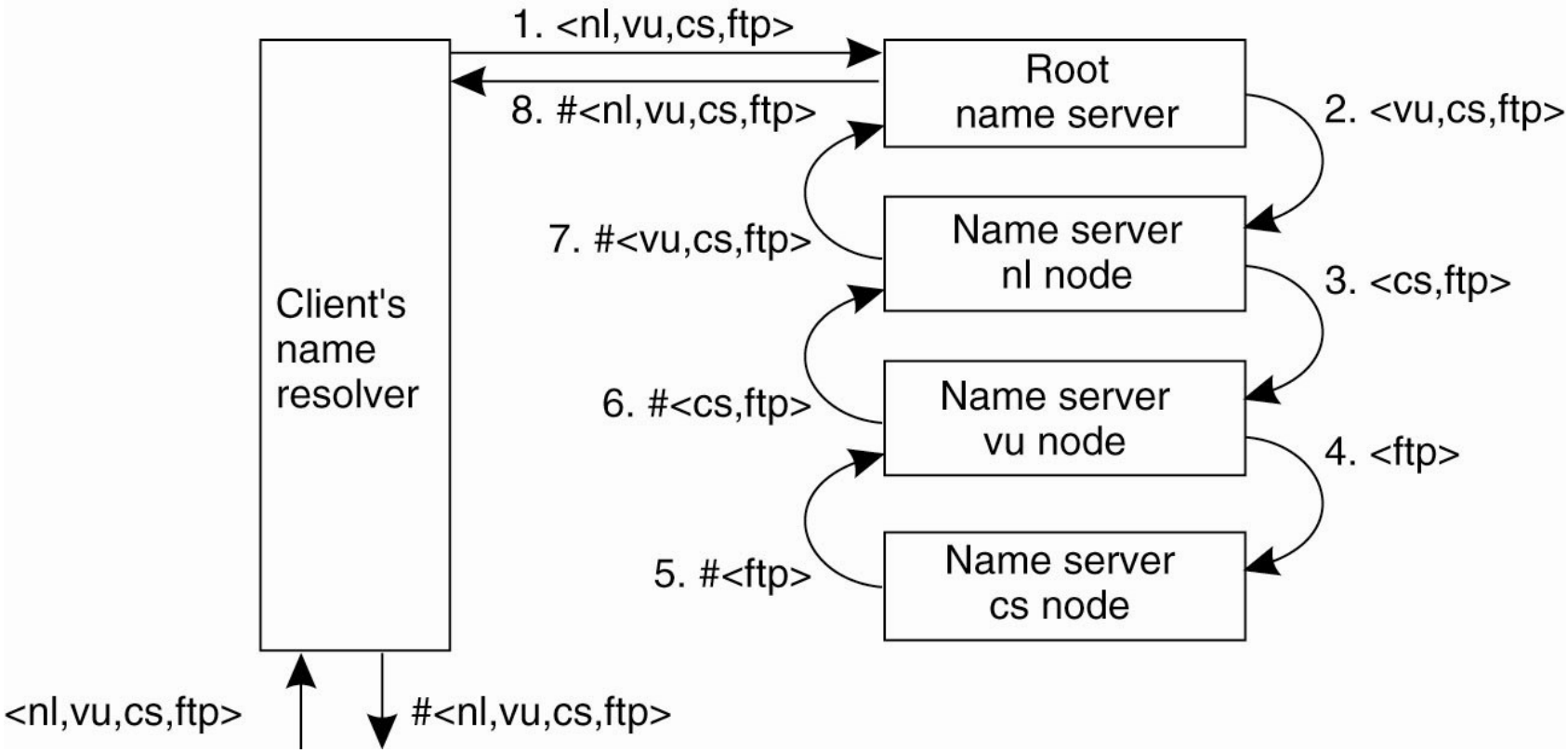


Figure 5-16. The principle of **recursive** name resolution.

Example: The Domain Name System

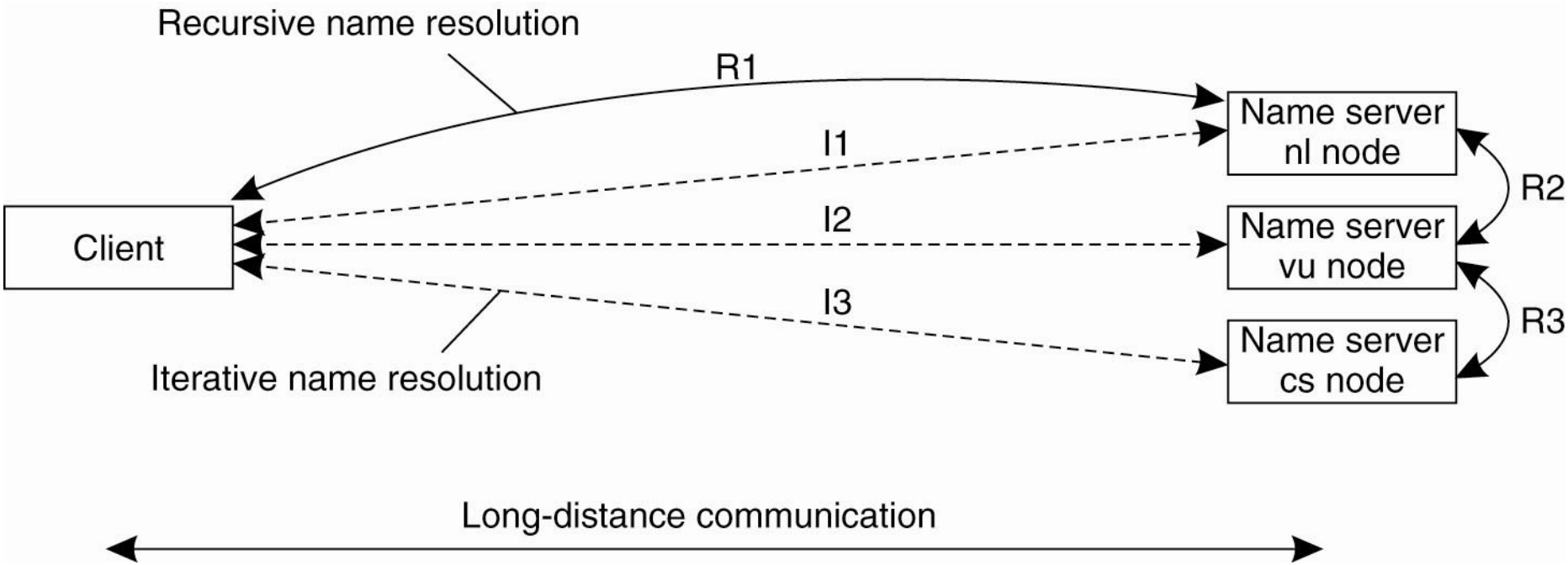


Figure 5-18. The comparison between recursive and iterative name resolution with respect to communication costs. Usually the **iterative** variant preferred over the recursive one, to keep server load low.

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

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*.

Hierarchical Implementations: LDAP (1)

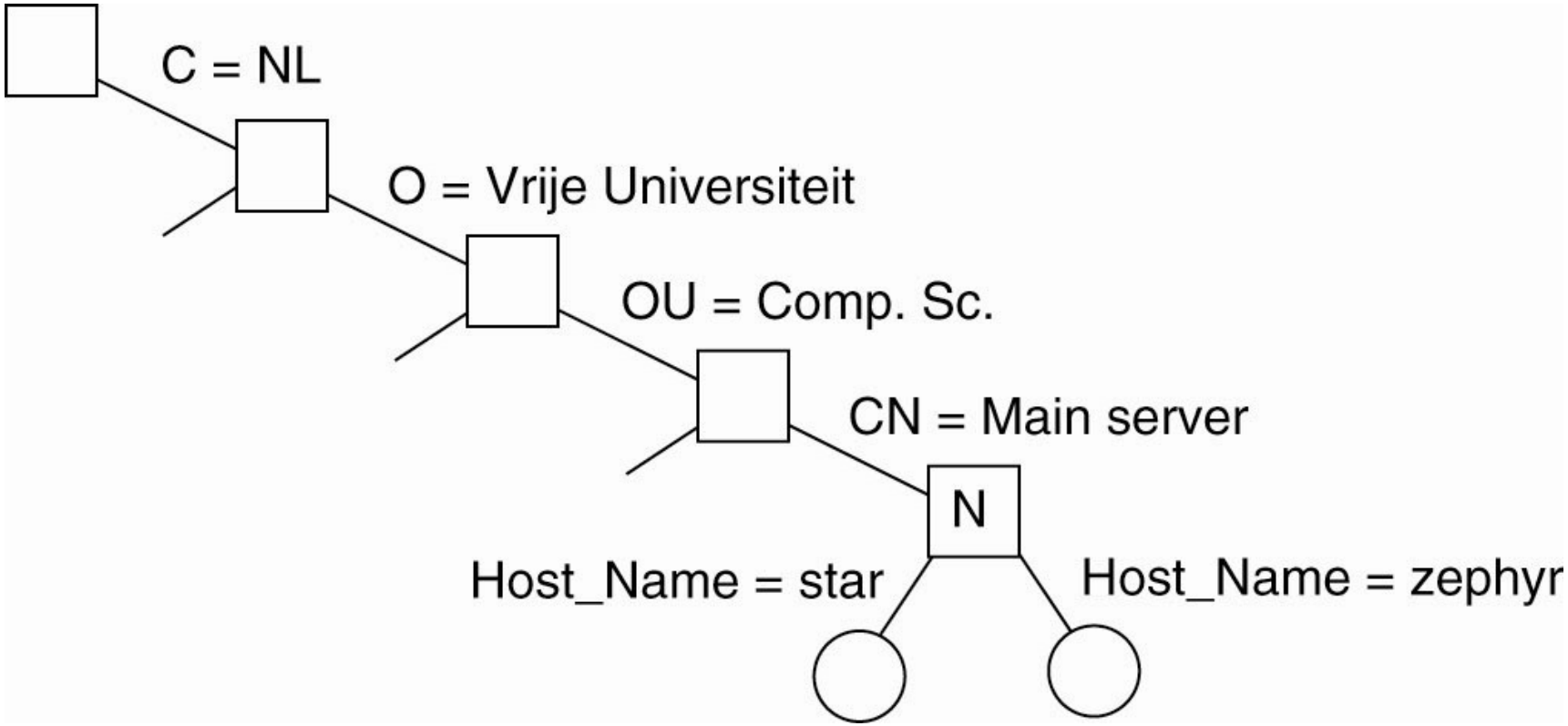


Figure 5-23. (a) Part of a directory information tree.

Hierarchical Implementations: LDAP (2)

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 (3)

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) The two Host (*Host_Name*) directory entries

DISTRIBUTED SYSTEMS
Principles and Paradigms
Second Edition
ANDREW S. TANENBAUM
MAARTEN VAN STEEN

Bearbeitung von Matthias Wallnöfer

Synchronization

Clock Synchronization

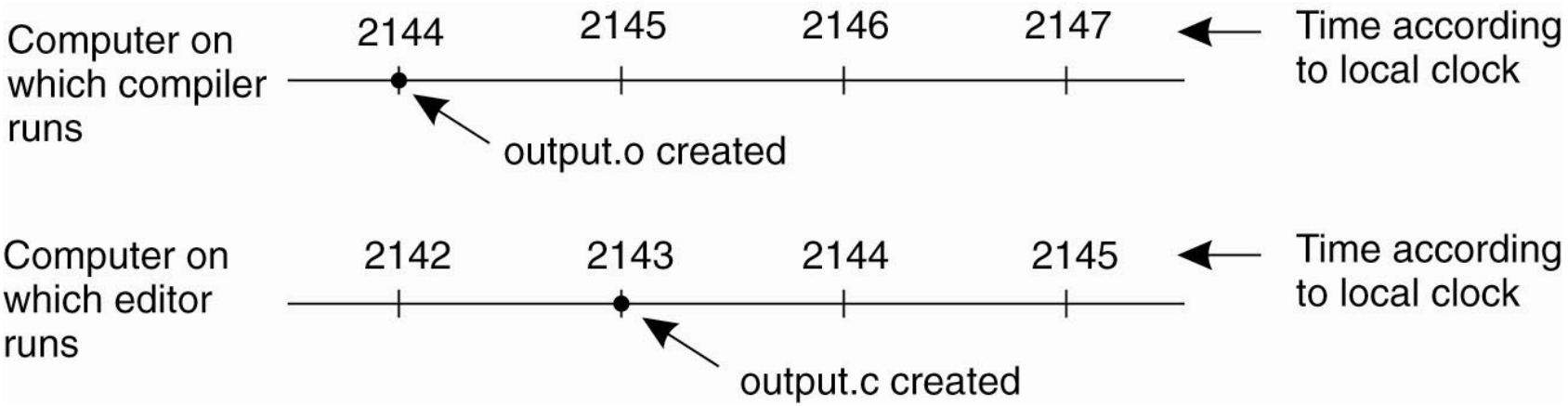


Figure 6-1. When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

Physical Clocks (1)

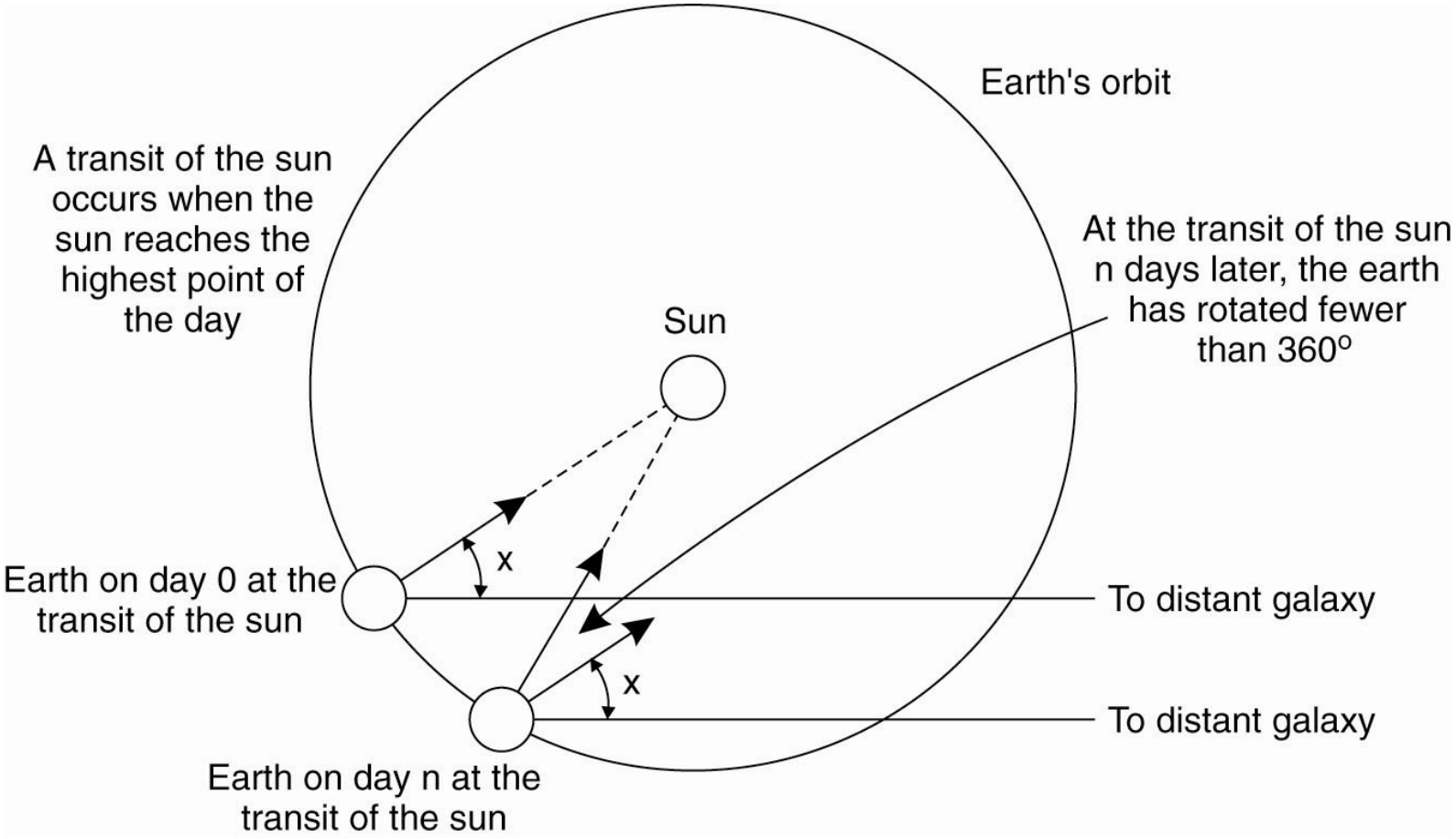


Figure 6-2. Computation of the mean solar day.

Physical Clocks (2)

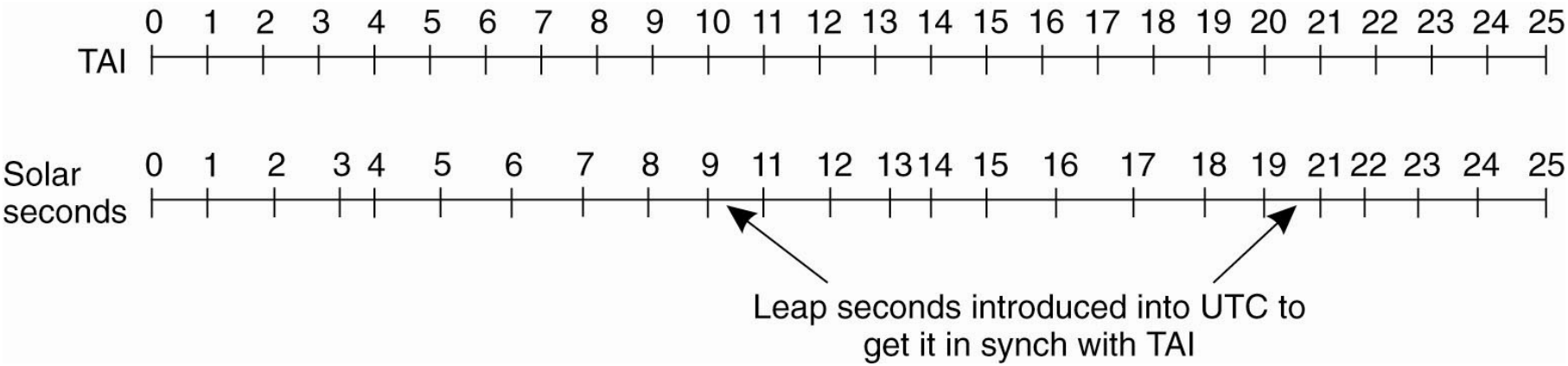


Figure 6-3. TAI seconds are of constant length (precision 10^{-16}), unlike solar seconds. Leap seconds are introduced when necessary to keep in phase with the sun.

Clock Synchronization Algorithms

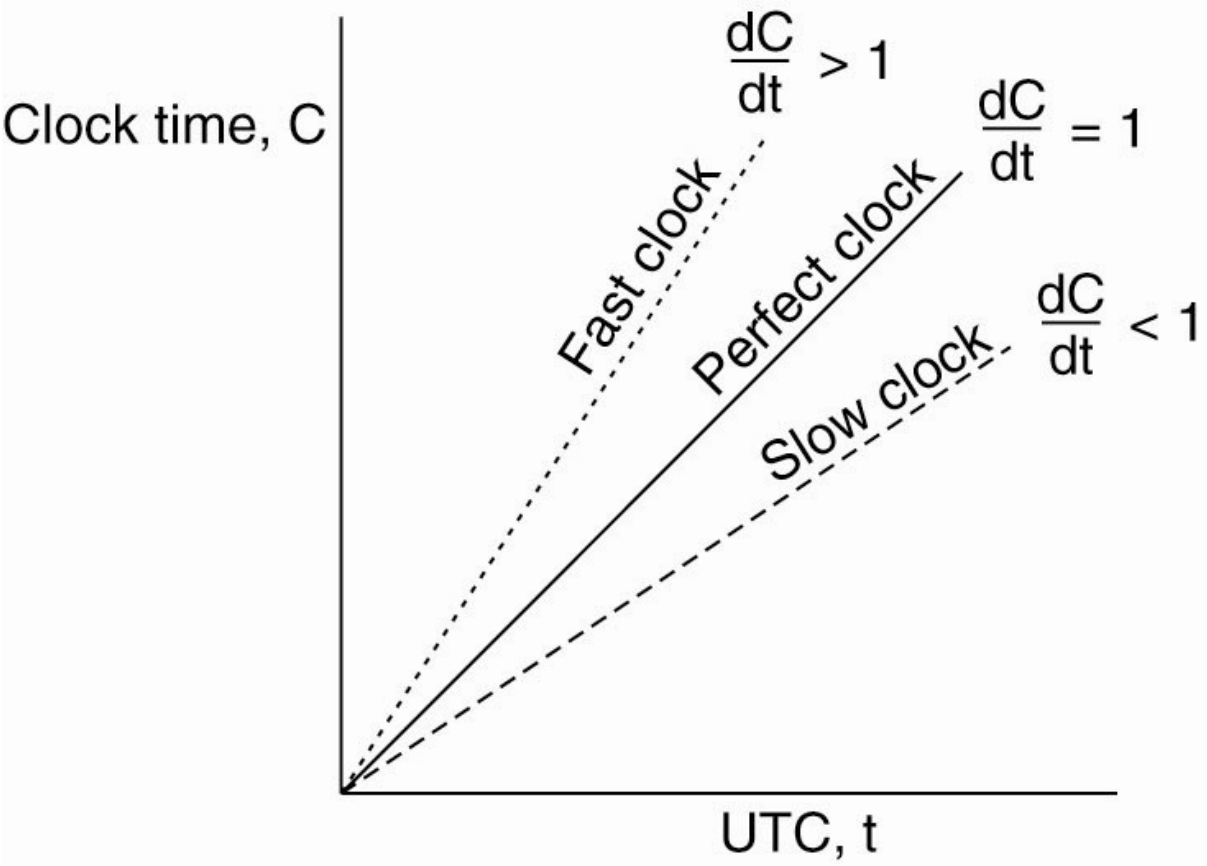


Figure 6-5. The relation between clock time and UTC when clocks tick at different rates.

Network Time Protocol
Cristian‘s Algorithm

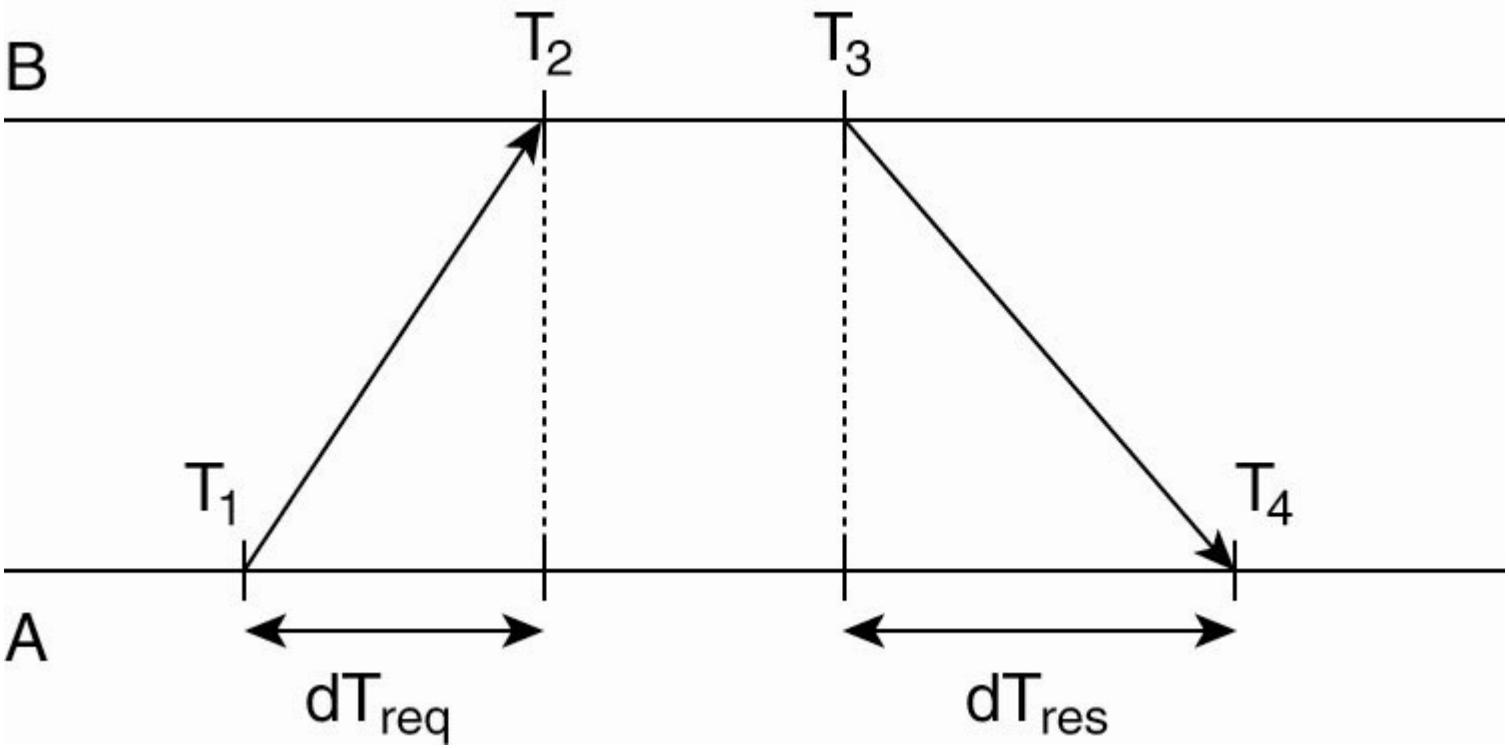


Figure 6-6. Getting the current time from a time server.
Accuracy: 10 ms, in LAN also < 1ms

T_4-T_1 = round-trip time (RTT)

The Berkeley Algorithm (1)

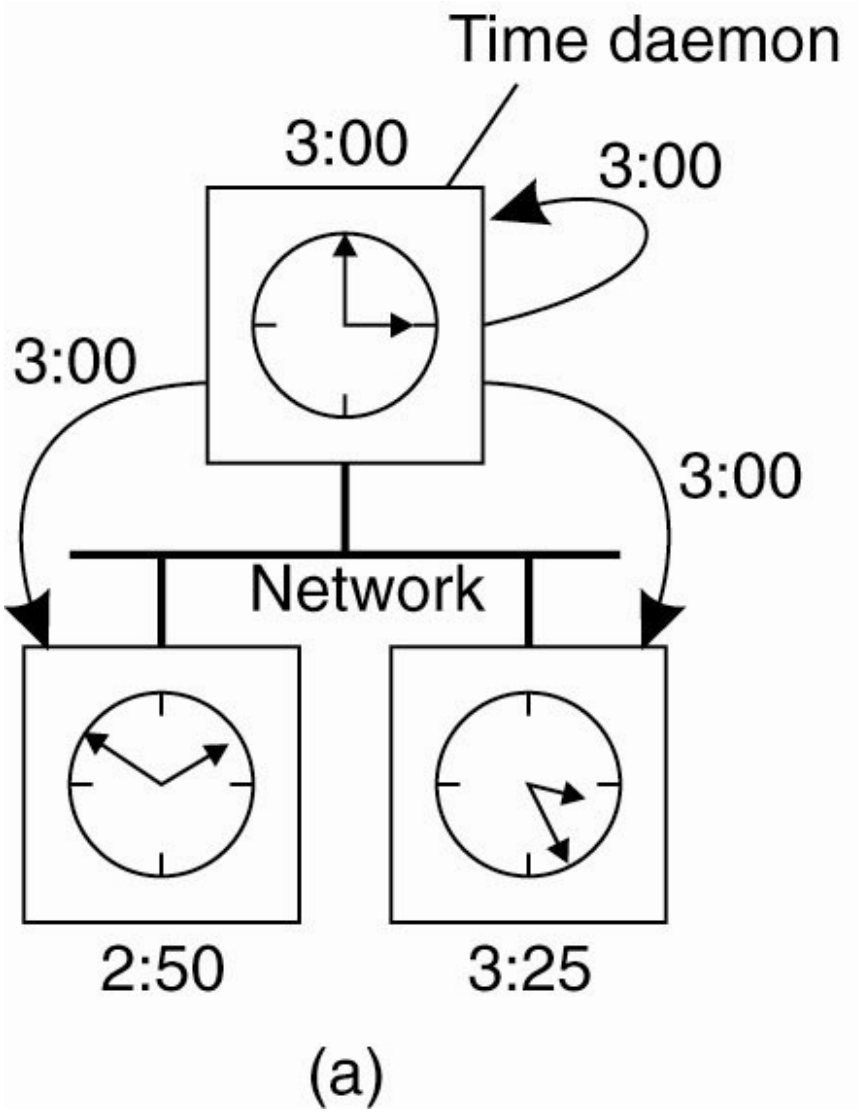


Figure 6-7. (a) The time daemon asks all the other machines for their clock values.

The Berkeley Algorithm (2)

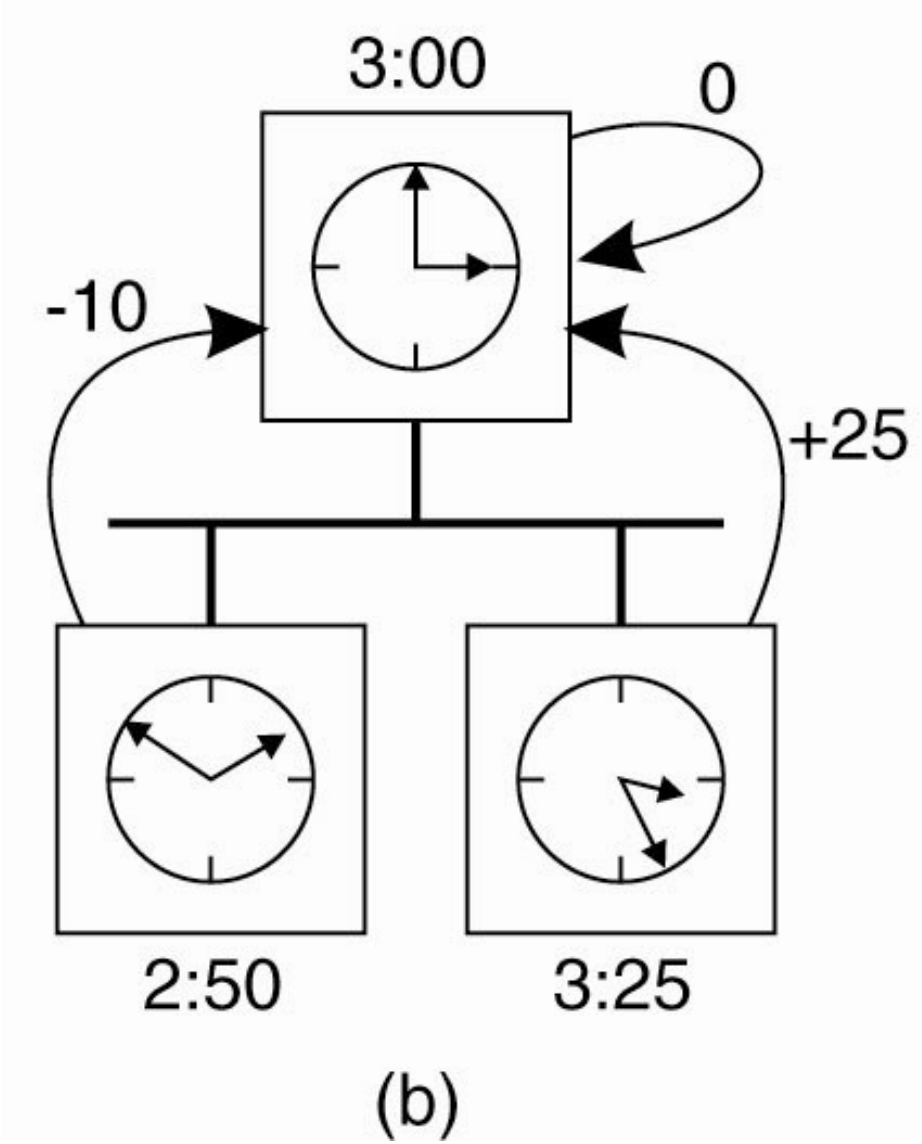


Figure 6-7.

(b) The machines answer. The time server also observes the various round-trip times (RTT).

The Berkeley Algorithm (3)

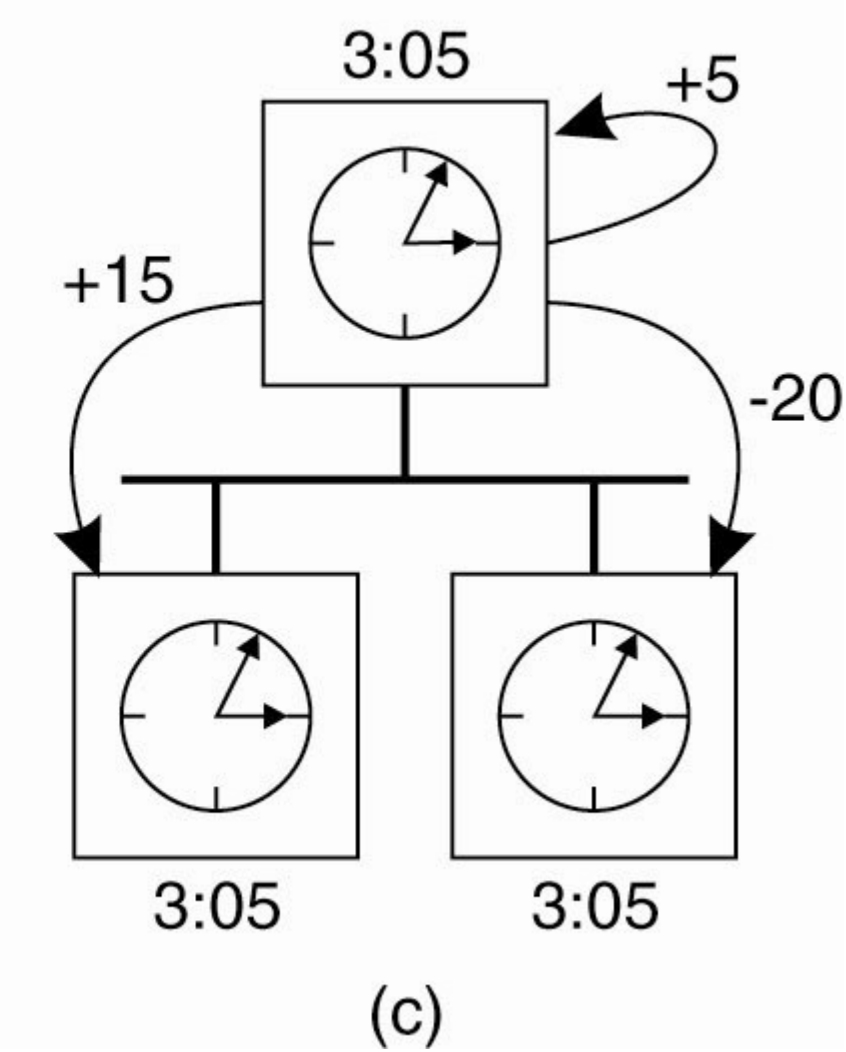


Figure 6-7. (c) The time daemon tells everyone how to adjust their clock.

The value is determined by the average time skew.

Lamport's Logical Clocks (1)

The "happens-before" relation $\rightarrow$ can be observed directly in two situations:

- If ***a*** and ***b*** are events in the **same process**, and ***a* occurs before *b***, then $a \rightarrow b$ is true.
- If ***a*** is the event of a message being **sent** by **one process**, and ***b*** is the event of the message being **received** by **another process**, then $a \rightarrow b$

Lamport's Logical Clocks (2)

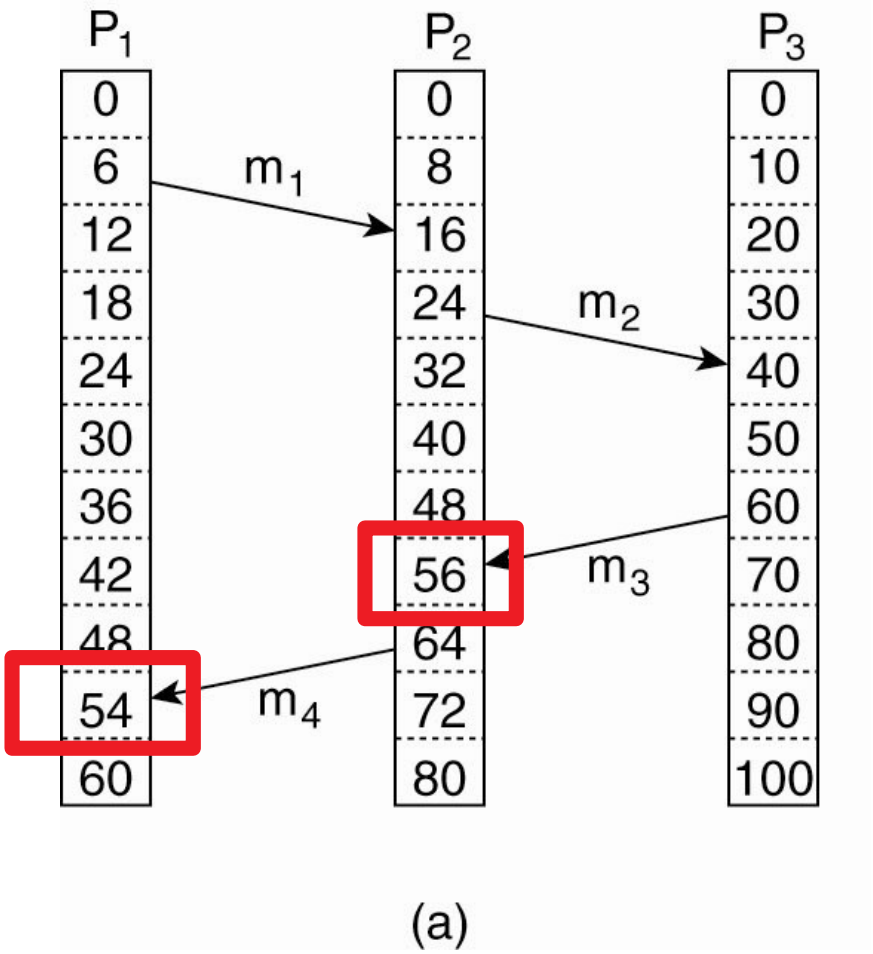


Figure 6-9. (a) Three processes, each with its own clock. The clocks run at different rates.

Lamport's Logical Clocks (3)

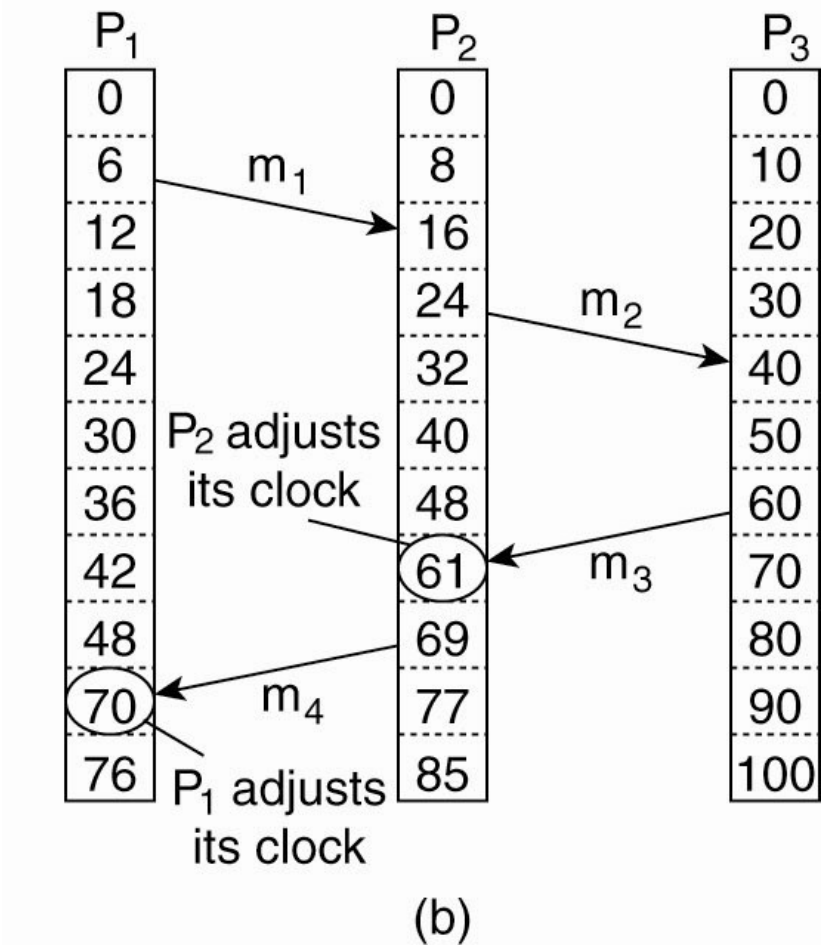


Figure 6-9. (b) Lamport's algorithm corrects the clocks.

Lamport's Logical Clocks (4)

Updating counter C_i for process P_i

1. **Before** executing an event P_i executes $C_i \leftarrow C_i + 1$.
2. When process P_i sends a message m to P_j, it sets *m*'s timestamp *ts (m)* equal to C_i after having executed the previous step.
3. **Upon the receipt** of a message *m*, process P_j adjusts its own local counter as $C_j \leftarrow \max\{C_j , ts\ (m)\}$, after which it then executes the first step and delivers the message to the application.

Problem: Totally Ordered Multicasting

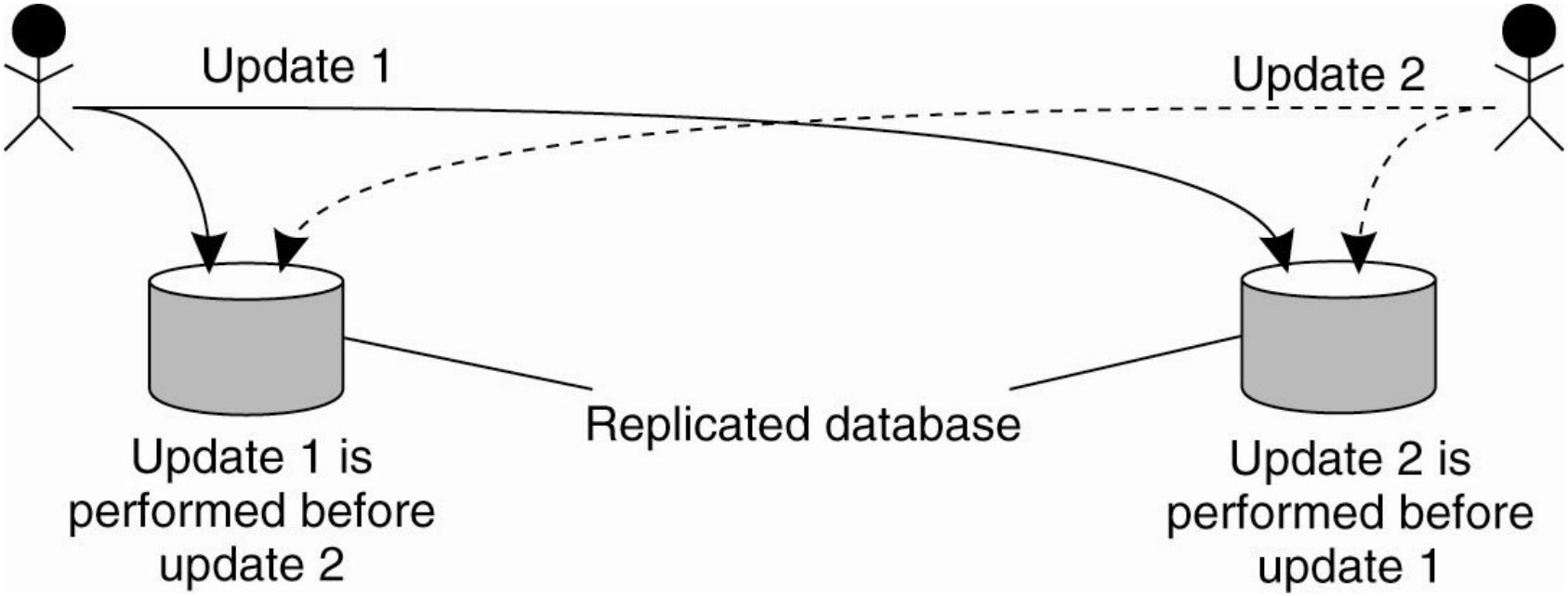


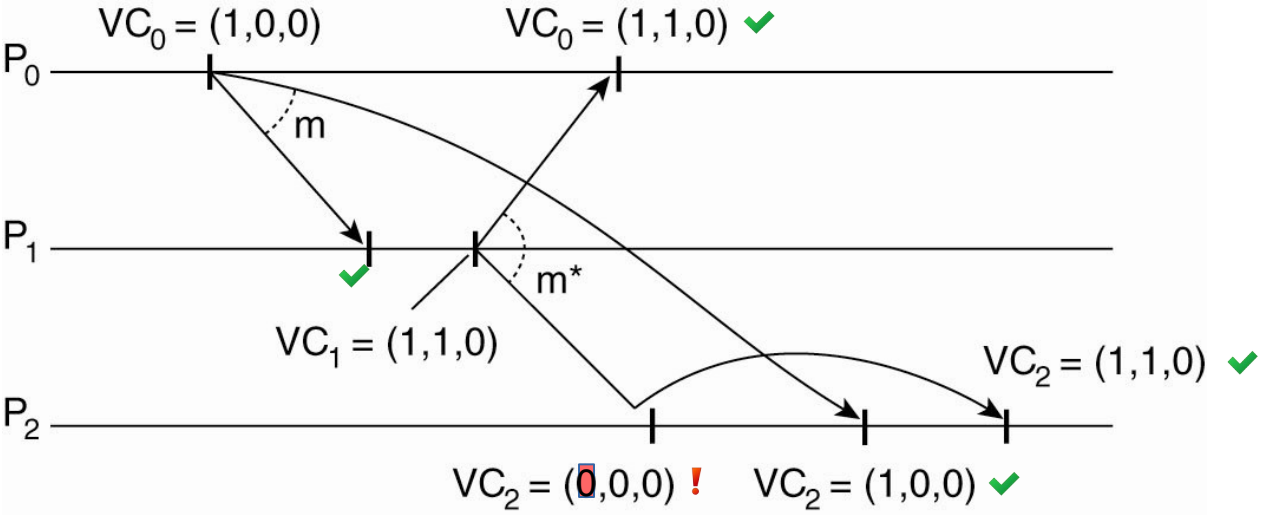
Figure 6-11. Updating a replicated database **at the same time** and leaving it in an inconsistent state.

Vector Clocks (1)

Vector clocks are constructed by letting each process P_i maintain a vector VC_i with the following two properties:

1. $VC_i[i]$ is the **number of (sending) events** that have **occurred so far at P_i**. In other words, $VC_i[i]$ is the local logical clock at process P_i .
2. If $VC_i[j] = k$ then **P_i knows that k events have occurred at P_j**. It is thus P_i's knowledge of the local time at P_j .

Vector Clocks (2)



Observation: We can now ensure that a message is delivered only if all causally preceding messages have already been delivered.

Adjustment: P_i increments $VC_i[i]$ only when sending a message, and P_j only adjusts VC_j when receiving a message (i.e., does not increment $VC_j[j]$.

P_j postpones delivery of m until:

- $ts(m)[i] = VC_j[i] + 1$ **and**
- $ts(m)[k] \leq VC_j[k]$ for $k \neq i$.

P_i (source) $\rightarrow$ P_j (destination)

Mutual Exclusion
A Centralized Algorithm (1)

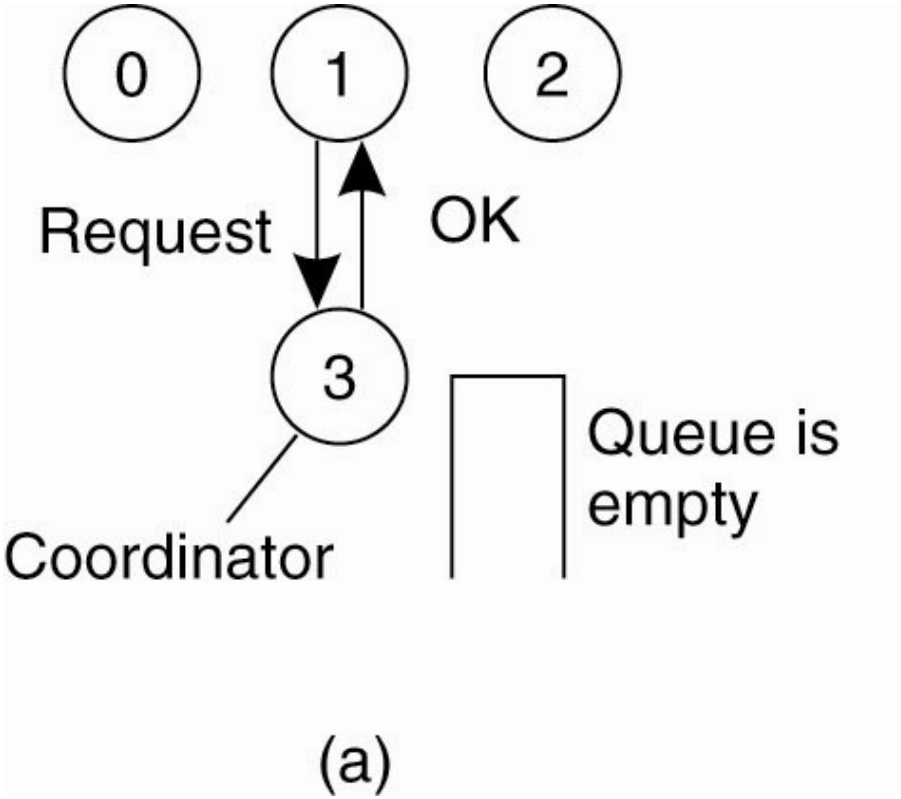


Figure 6-14. (a) Process 1 asks the coordinator for permission to access a shared resource. Permission is granted.

Mutual Exclusion
A Centralized Algorithm (2)

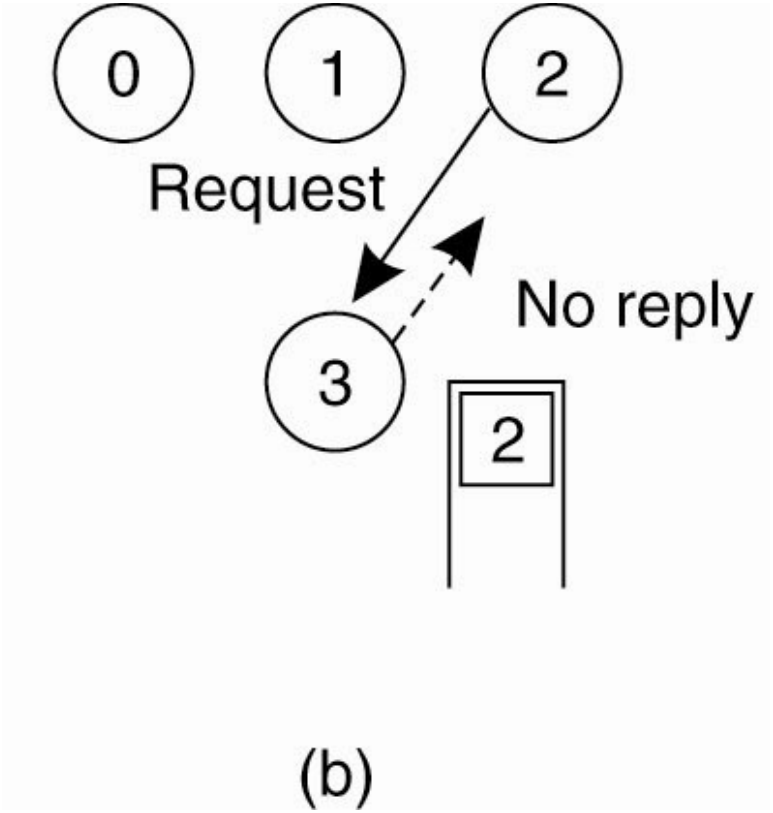


Figure 6-14. (b) Process 2 then asks permission to access the same resource. The coordinator does not reply.

Mutual Exclusion
A Centralized Algorithm (3)

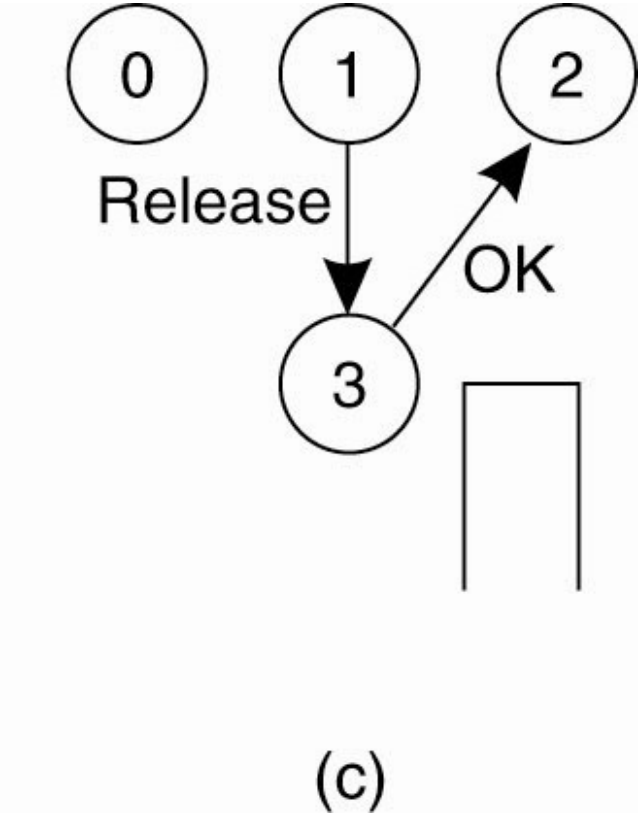


Figure 6-14. (c) When process 1 releases the resource, it tells the coordinator, which then replies to 2.

A Distributed Algorithm (1)

Three different cases:

1. If the receiver **is not accessing** the resource and does not want to access it, **it sends back an OK** message to the sender.
2. If the receiver **already has access** to the resource, it simply **does not reply**. Instead, it queues the request.
3. If the receiver wants to access the resource as well but has not yet done so, **it compares the timestamp** of the incoming message with the one contained in the message that it has sent everyone. **The lowest one wins**.

A Distributed Algorithm (2)

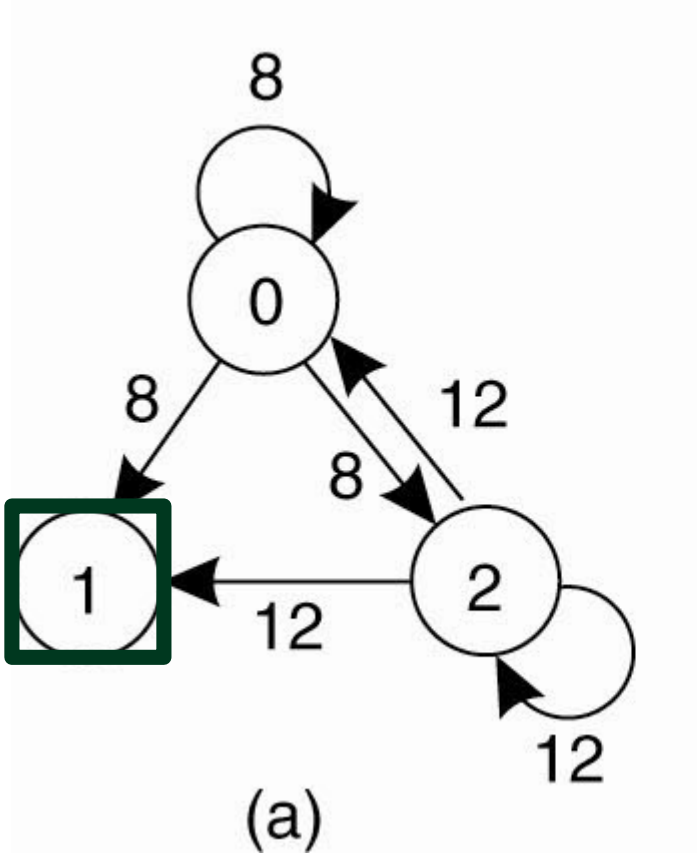


Figure 6-15. (a) Two processes want to access a shared resource (1) at the same moment.

A Distributed Algorithm (3)

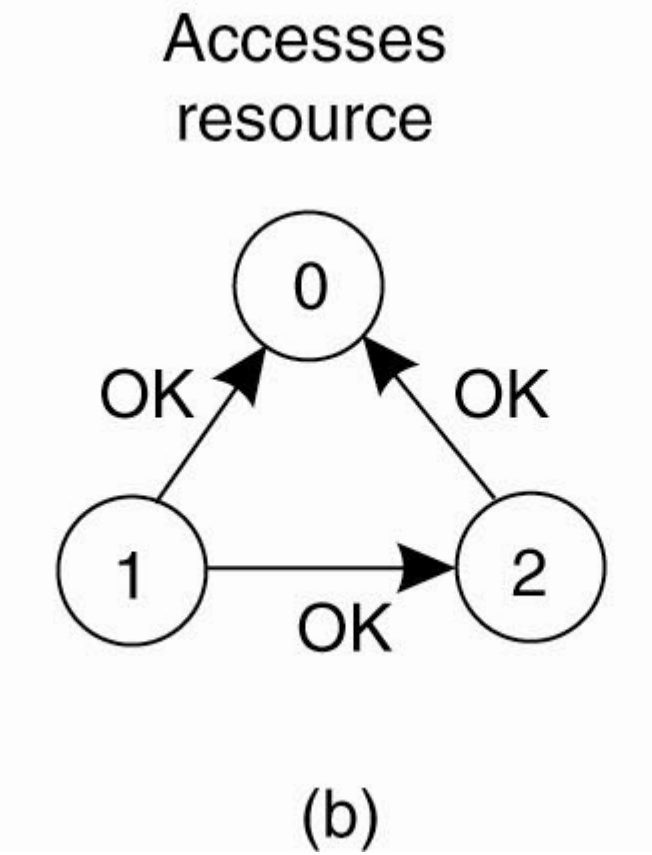


Figure 6-15. (b) Process 0 has the lowest timestamp, so it wins.

A Distributed Algorithm (4)

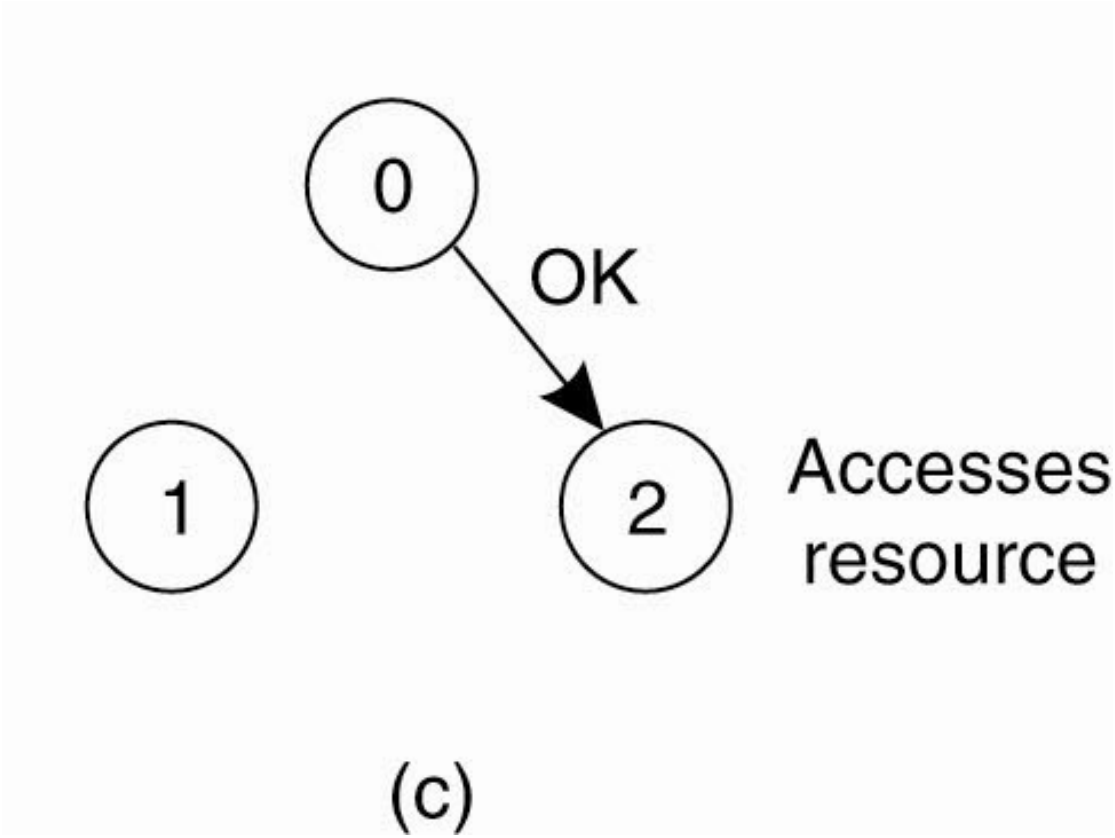


Figure 6-15. (c) When process 0 is done, it sends an OK also, so 2 can now go ahead.

Election Algorithms

The Bully Algorithm

1. *P* sends an *ELECTION* message to all processes with **higher numbers**.
2. If no one responds, ***P* wins the election** and becomes coordinator.
3. If one of the higher-ups answers, **it takes over**. *P*'s job is done.

The Bully Algorithm (1)

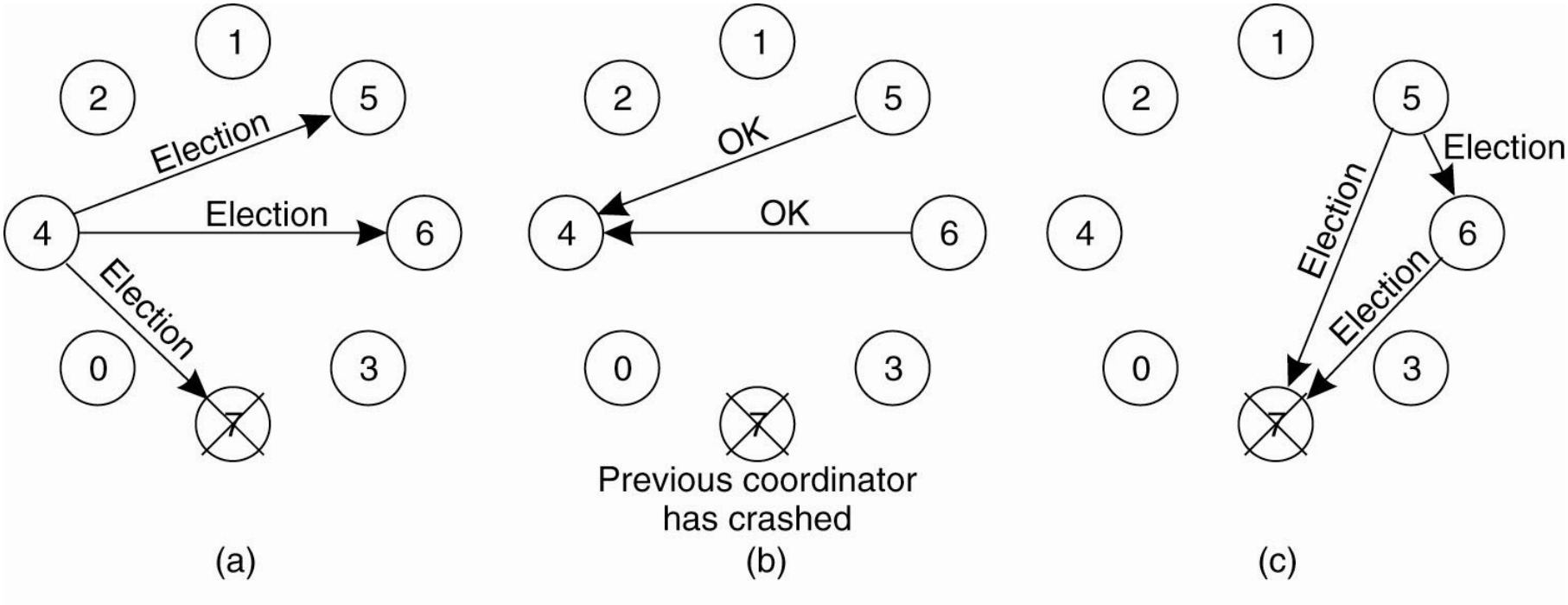


Figure 6-20. The bully election algorithm. (a) Process 4 holds an election. (b) Processes 5 and 6 respond, telling 4 to stop. (c) Now 5 and 6 each hold an election.

The Bully Algorithm (2)

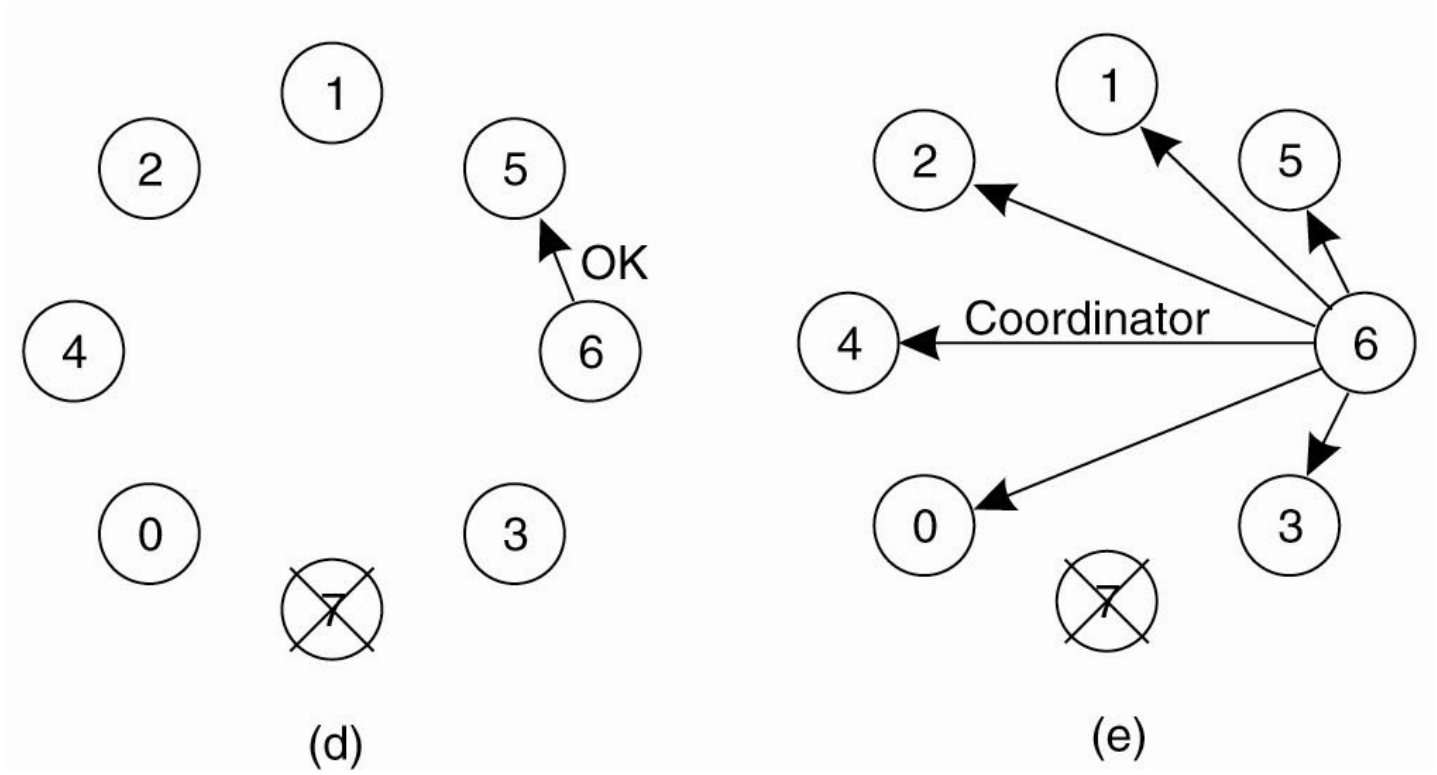


Figure 6-20. The bully election algorithm. (d) Process 6 tells 5 to stop. (e) Process 6 wins and tells everyone.