

Choose the most accurate definition of Distributed Systems

Select one:

- a. A set of independent computers that are not connected to each other, provide services to users as a single computer.
- b. A set of homogeneous computers that connect with each other, provide services to users as a single computer.
- c. A set of heterogeneous computers that are interconnected, providing services to users as a local area network having multiple computers.
- d. A set of heterogeneous computers that are interconnected, providing services to users as a single coherent system. ✓

What is the role of Middleware layer in distributed systems?

Select one:

- a. It provides transparency for the upper-layer of distributed applications. ✓
- b. It is an archive of software.
- c. It provides shared resources for processes in the system.
- d. It contains the database of the entire system.

What is the difference between UDP socket and TCP socket?

Select one:

- a. These two socket types have no difference.
- b. UDP socket uses physical port, TCP socket uses logic port.
- c. The TCP socket uses a multiplexing mechanism, so the server does not need to create a new socket with each connection request. The UDP socket, on the other hand, will create a new socket for each connection request.
- d. The UDP socket uses a multiplexing mechanism, so the server does not need to create a new socket for each connection request. The TCP socket, on the other hand, will create a new socket for each connection request. ✓

What is the problem of RPC when using call-by-reference parameters?

Select one:

- a. Client and server machines have different data representation conventions. ✓
- b. Client and server machines have 2 different memory spaces.
- c. Client and server machines do not have shared cpu.
- d. Client and server machine have no shared clock.

The problem encountered in the RPC mechanism when passing reference parameters:

Select one:

- a. Client and server have two different memory spaces ✓
- b. Client and server have different type of data representation
- c. The client and the server do not have a common physical clock
- d. The client and the server do have a common physical clock

In multithreaded server architectures, what is the advantage of thread-per-request architecture?

Select one:

- a. No queues and low overhead of thread creation/destroy operations
- b. The server process never crashes and needs a queue
- c. No queuing and bandwidth can be maximized ✓
- d. Bandwidth can be maximized and server process never crashes

Which is not a disadvantage of the forwarding pointer mechanism?

Select one:

- a. The chain of references for a mobile entity can be very long making the location finding process expensive.
- b. Intermediate nodes must store forwarding pointers for as long as possible
- c. When a server stub is not pointed to by any client stub, it can be removed ✓
- d. When a reference pointer is broken, the entity cannot be found

Why can't the address of an Access Point be used to identify an entity?

Select one:

- a. Because the address of the AP and the name of the entity have different format
- b. Because entities can move from one AP to another, and APs can be attached to different entities. ✓
- c. Because AP is also an entity
- d. Because the address of the AP is of a structured type, and the name of the entity is of an unstructured type

What are the disadvantages of the Home-based solution for a naming system?

Select one:

- a. Whenever it wants to communicate with an entity, the client must first contact the home agent, even though it is far from the client. ✓
- b. Every time it wants to search for an entity, the Home Agent must broadcast the search message to all entities in the system.
- c. The name of the Home Agent must be hashed using a hash algorithm, generating a key and entering the system.
- d. When the entity moves, it must leave a reference pointer to the Home Agent. The reference string will grow longer over time.

What is the difference between logical clock synchronization and physical clock synchronization?

Select one:

- a. They are not different.
- b. Logical clock synchronization is only concerned with processes giving reasonable results.
- c. Logical clock synchronization is only concerned with the execution order of events. ✓
- d. Synchronized logical clocks arrange execution events in the correct order they are fired.

What is the well-known application of the logic clock synchronization of Lamport?

Select one:

- a. Prevent the attack "Man in the middle".
- b. Guaranty the Totally Ordered Multicasting ✓
- c. Guaranty the authentication the sending processes.
- d. Guaranty not to lose messages in the process of sending and receiving between processes.

What are the disadvantages of the centralized algorithm among the mutual exclusion algorithms?

Select one:

- a. The coordinator process has to use the queue for requests which consumes storage resources
- b. Processes that want to access the resource do not know which process to send the request to
- c. The coordinator process is at risk of being overloaded (bottleneck phenomenon) ✓
- d. Processes that want to use the resource must broadcast the request to all other processes

What is a consistency model?

Select one:

- a. A database
- b. A superserver
- c.
A contract between processes and the data store.
✓
- d. A serie of reading and writting operations

What is the deviation of ordering of update operations?

Select one:

- a. The number of the processes.
- b. The number of operations in the queue. ✓
- c. The number of submitted operations
- d. The number of the threads

Thế nào là *Tính sẵn sàng* của hệ thống?

Select one:

- a. Tính dễ dàng tự sửa chữa khi gặp lỗi.
- b. Hệ thống tạm thời gặp lỗi nhưng vẫn khắc phục được.
- c.
Là hệ thống sẵn sàng để được sử dụng ngay lập tức. Xác suất để hoạt động một cách chính xác tại bất kỳ thời điểm nào.
✓
- d. Là hệ thống chạy liên tục mà không gặp lỗi. Liên quan đến một khoảng thời gian.

Thế nào là Crash failure (lỗi sụp đổ)?

Select one:

- a. Một server không trả lời được các yêu cầu được gửi đến.
- b. Câu trả lời của server là không chính xác
- c. Câu trả lời của server kéo dài vượt quá khoảng thời gian cho phép (timeouts)
- d. Một máy chủ ngừng hoạt động, nhưng vẫn hoạt động chính xác đến khi nó ngừng hoạt động ✓

Lamport đã phát biểu điều kiện để có được sự đồng thuận của hệ thống là gì?

Select one:

- a. Sự đồng thuận của hệ thống có thể đạt được nếu có $2k+1$ tiến trình hoạt động đúng với tổng $(3k+1)$ tiến trình, trong đó có k tiến trình lỗi.
✓
- b. Sự đồng thuận của hệ thống có thể đạt được nếu có $2k+1$ tiến trình hoạt động đúng với tổng $5k$ tiến trình, trong đó có k tiến trình lỗi.
- c. Sự đồng thuận của hệ thống có thể đạt được nếu có k tiến trình hoạt động đúng với tổng $(3k+1)$ tiến trình, trong đó có k tiến trình lỗi.
- d. Sự đồng thuận của hệ thống có thể đạt được nếu có $3k+1$ tiến trình hoạt động đúng với tổng $(2k+1)$ tiến trình, trong đó có k tiến trình lỗi.



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

Chapter 1: Introduction and Architectures of Distributed Systems

Dr. Trần Hải Anh

Outline

1. Introduction
2. Characteristics of Distributes Systems
3. Software architecture styles and System architectures
4. Middleware in Distributed Systems

1. Introduction

1.1. Brief history

1.2. Definition

1.3. Examples

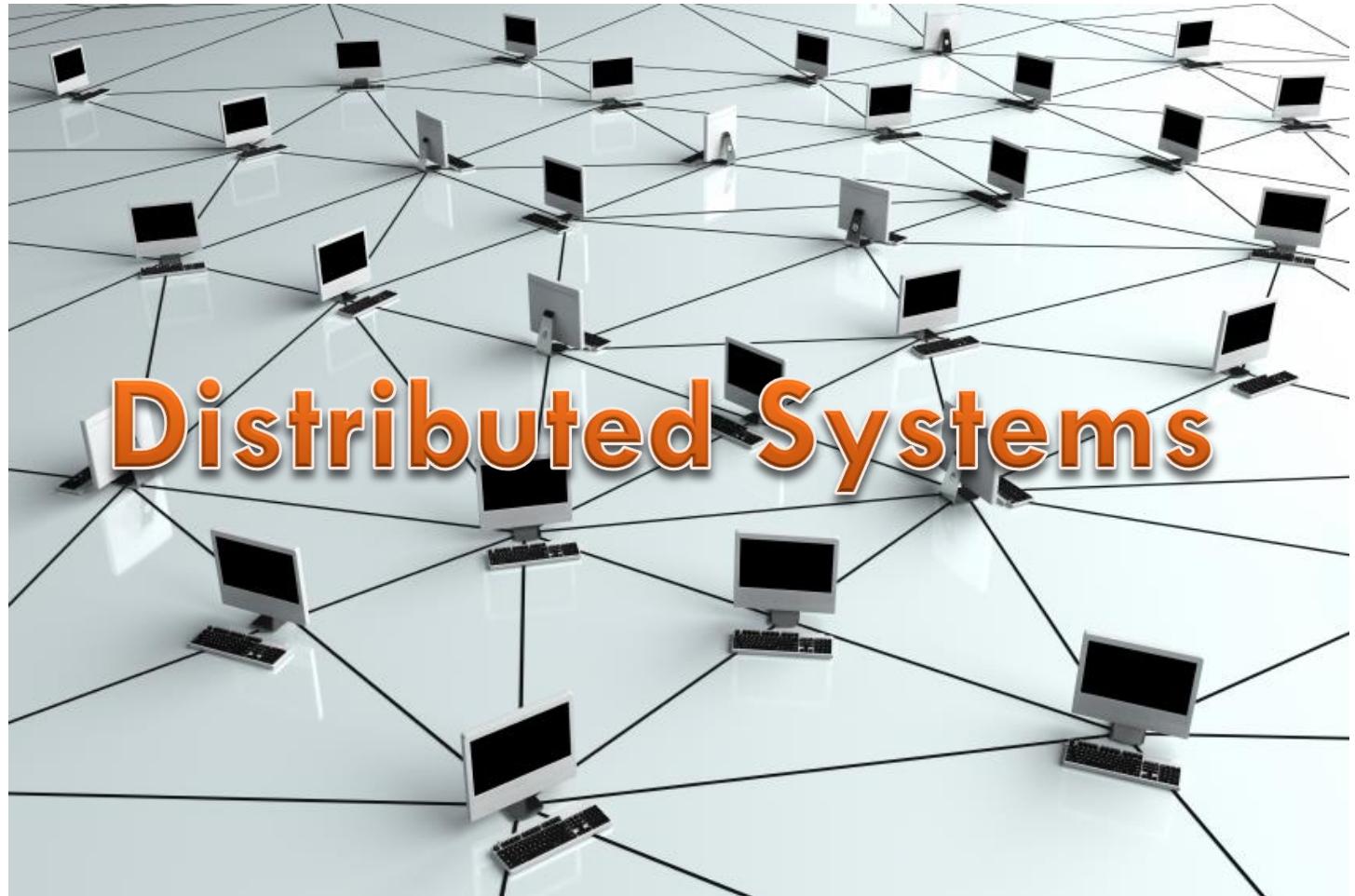
1.1. History

4

- History of computer
 - First generation (1937-1946)
 - vacuum tubes
 - ENIAC (Electronic Numerical Integrator And Computer)
 - Second generation: (1947 – 1962)
 - Transistor
 - Universal Automatic Computer (UNIVAC 1).
 - Third generation: 1963 - present
 - IC: Integrated Circuit
 - MS-Dos
 - IBM PC
- History of Computer Network
- Change the way of using PC
- Change the user requirements for quality of service

Distributed Systems

5



1.2. Definition

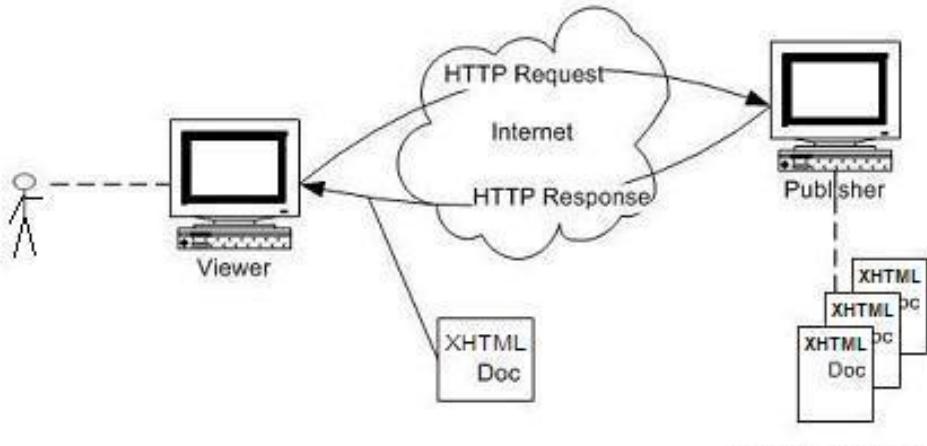
6

- Independent computers
 - ▣ They don't depend on each others. Different on hardware and software architecture.
- Connected
- Provide common service uniformly
- Users don't need to care about system's details
- *A collection of independent connected computers that provides services to its users as a single coherent system. [Tanenbaum 2006]*

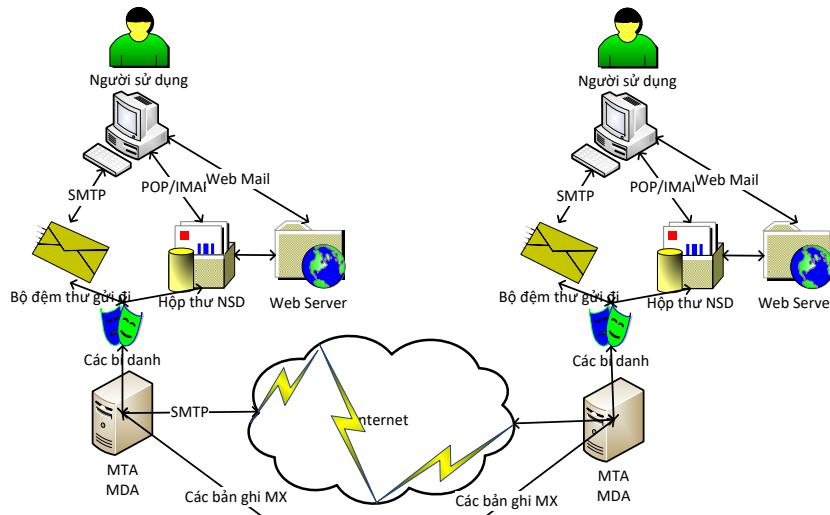
1.3. Examples

7

- WWW
- Email system
- Etc.



By FYIcenter.com



2. Characteristics

2.1. Making resources accessible

2.2. Distribution transparency

2.3. Openness

2.4. Scalability

2.1. Making resources accessible

9

- Easy to access remote resources
- Resources: anything (printers, computers, storage facilities, data, files, web pages, etc.)
- Example:
 - Sharing printer
 - Sharing supercomputer, high-performance storage system
 - Other expensive peripherals
- Working together: groupware
- Security problems: eavesdropping, intrusion on communication, etc.

2.2. Distribution Transparency

10

- Hide the fact that its processes and resources are physically distributed across multiple computers
- Appear as a single computer system → transparent

Types of transparency

11

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

Degree of transparency?

Attempting to completely hide all distribution aspects from users is not a good

2.3. Openness

12

- **Open distribution system** is a system that offers services according to standard rules that describe the syntax and semantics of those services.
- Protocols
- Services are specified through **interfaces**.
- Interface Definition Language (IDL).
- Interoperability
- Portability
- Extensible

2.4. Scalability

13

- Size
 - ▣ Add more users and resources
- Geographical scalability
 - ▣ Users and resources may lie far apart
- Administrative scalability
 - ▣ It spans many independent administrative organizations

Scalability problems

14

- Size:
 - Centralized system
 - Decentralized system
- Geographical scalability
 - LAN → wide area network
 - Broadcasting
 - Reliable communication
- Administrative scalability
 - Resource usage
 - Management
 - Security

Scaling techniques

15

- Asynchronous communication
- Distribution
- Replicate
- Caching

3. Software architecture styles and System architectures

3.1. Architecture

17

- Organization of a distributed system: → distinction between *the logical organization* and *the physical realization*
- The logical organization: the collection of software components that constitute the system → **software architecture**
- The physical realization: instantiate and place software components on real machines → **system architecture**

3.2. Software architecture styles

18

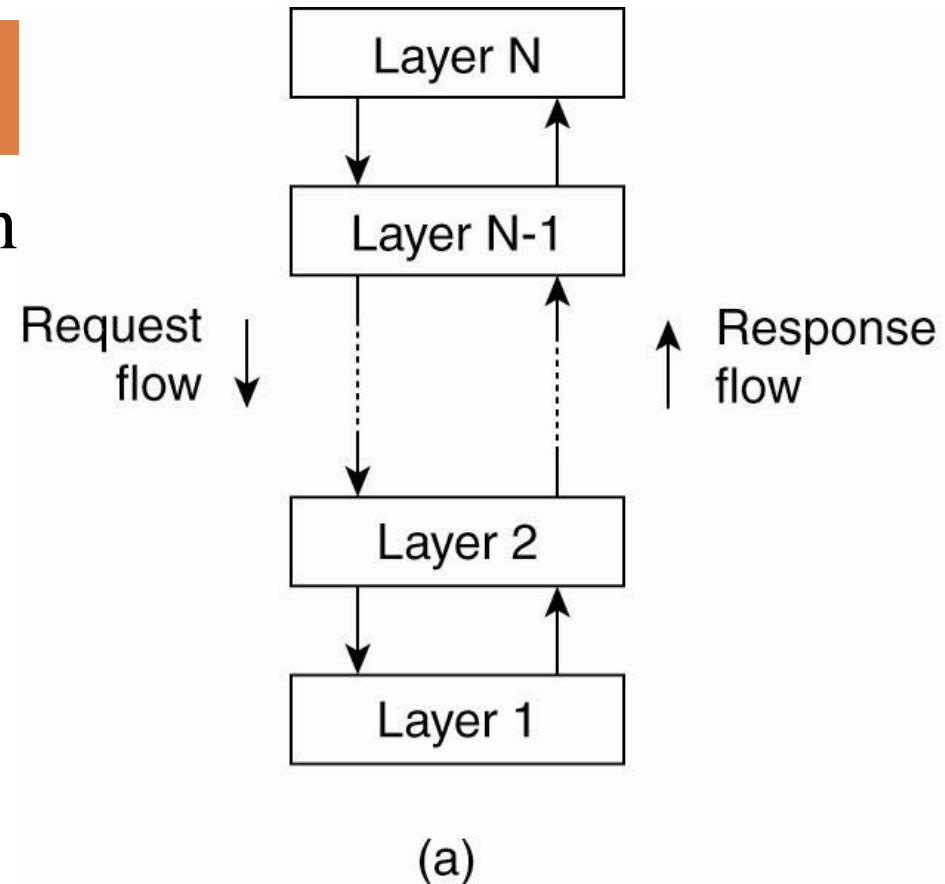
- Layered architectures
- Object-based architectures
- Data-centered architectures
- Event-based architectures
- Microservices

3.2.1. Layered architectures

19

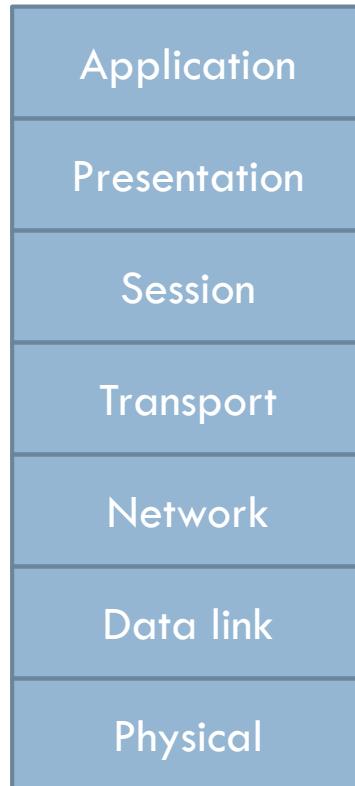
Layered fashion

- Each layer has its own task
- Transparency

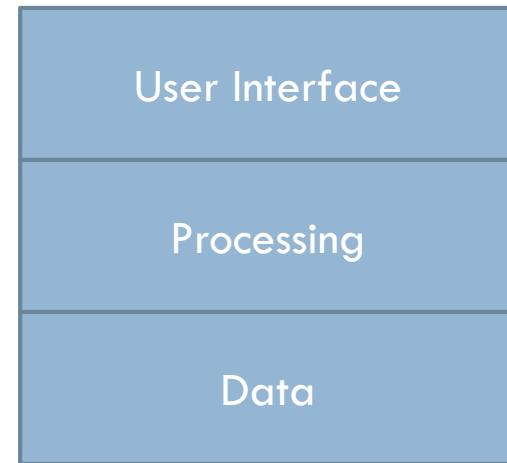


Layered architectures (con't)

20



OSI model



General layered architecture

Example 1: E-commerce system

21

User interface layer

Web
interface

Console
interface

Mobile
app
interface

Processing layer

Client
management

Product
management

Order
management

Transaction
Analyzing

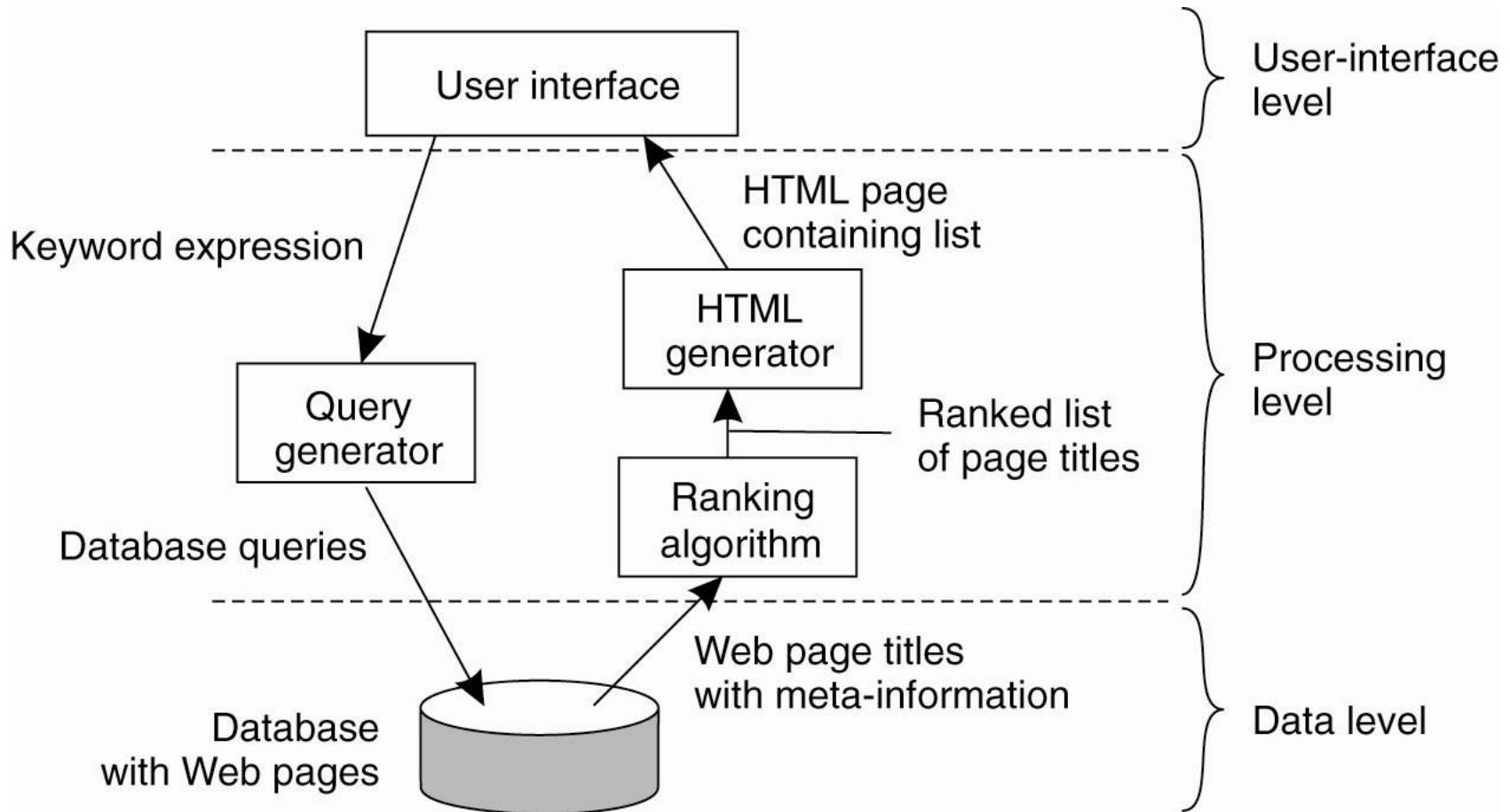
Product
suggestion

Data layer

database

Example 2: a search engine

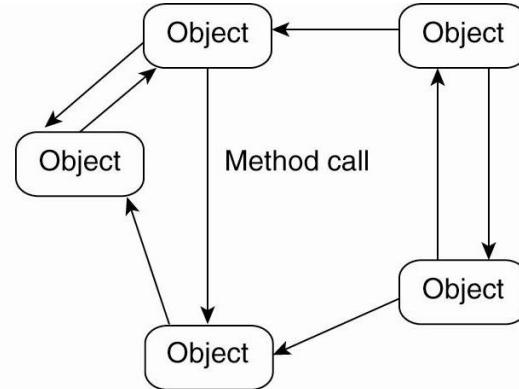
22



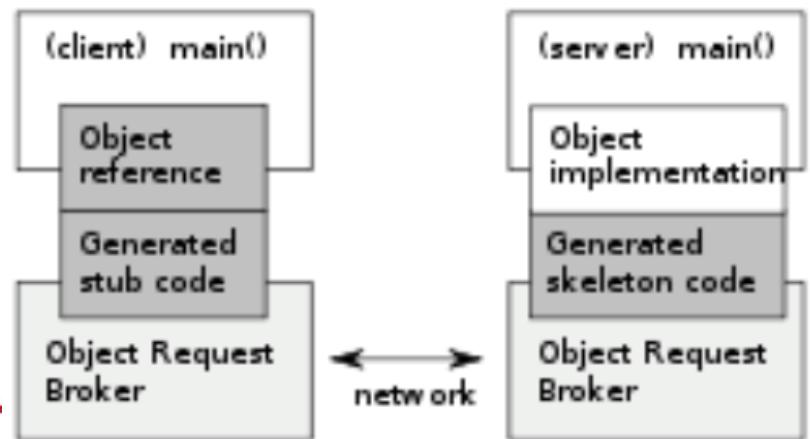
3.2.2. Object-based architectures

23

- Component: Object
- Connector: (Remote) Procedure call
- Object Client & Object server
- loosely-coupled systems
- E.g. CORBA



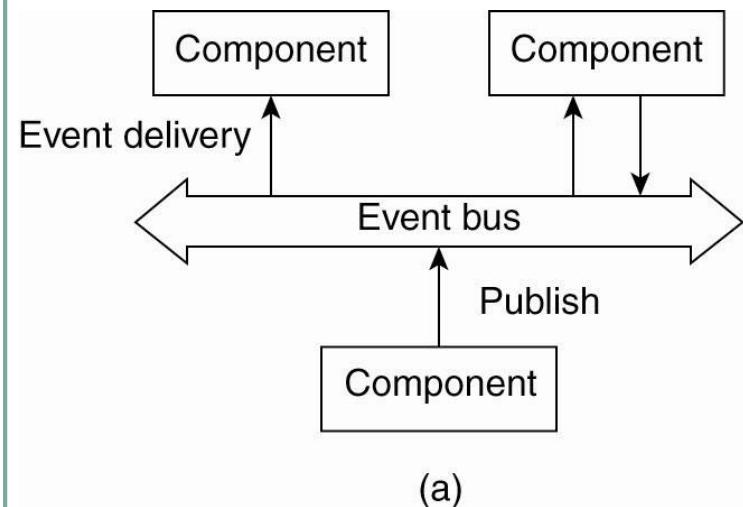
(b)



3.2.3. Event-based architectures

24

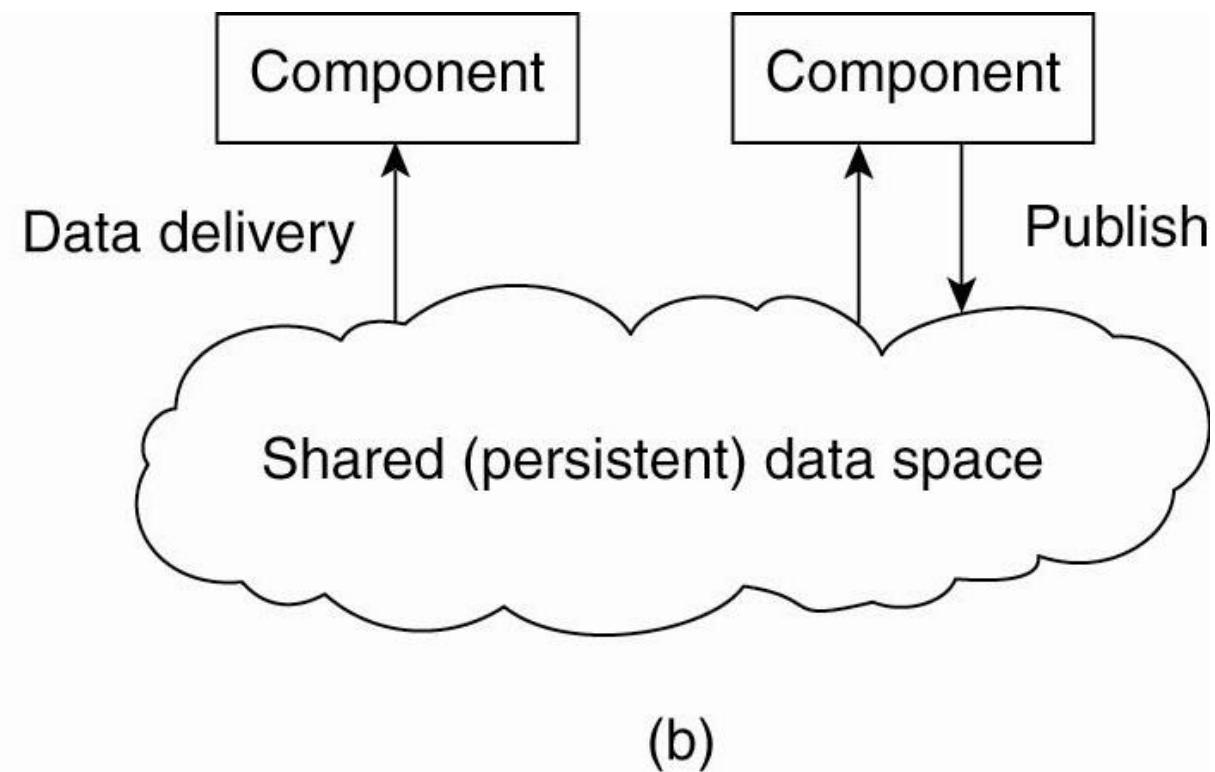
- Communicate through the propagation of events (optionally carry data)
- Publish/Subscribe systems
- Processes are loosely coupled
- loosely-coupled systems



3.2.4. Data-centered architecture

25

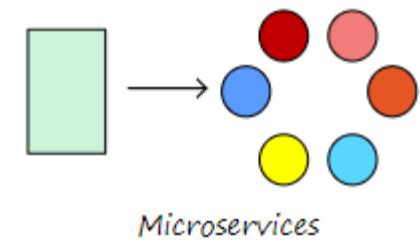
- Communicate through a common repository



3.2.5. Microservices

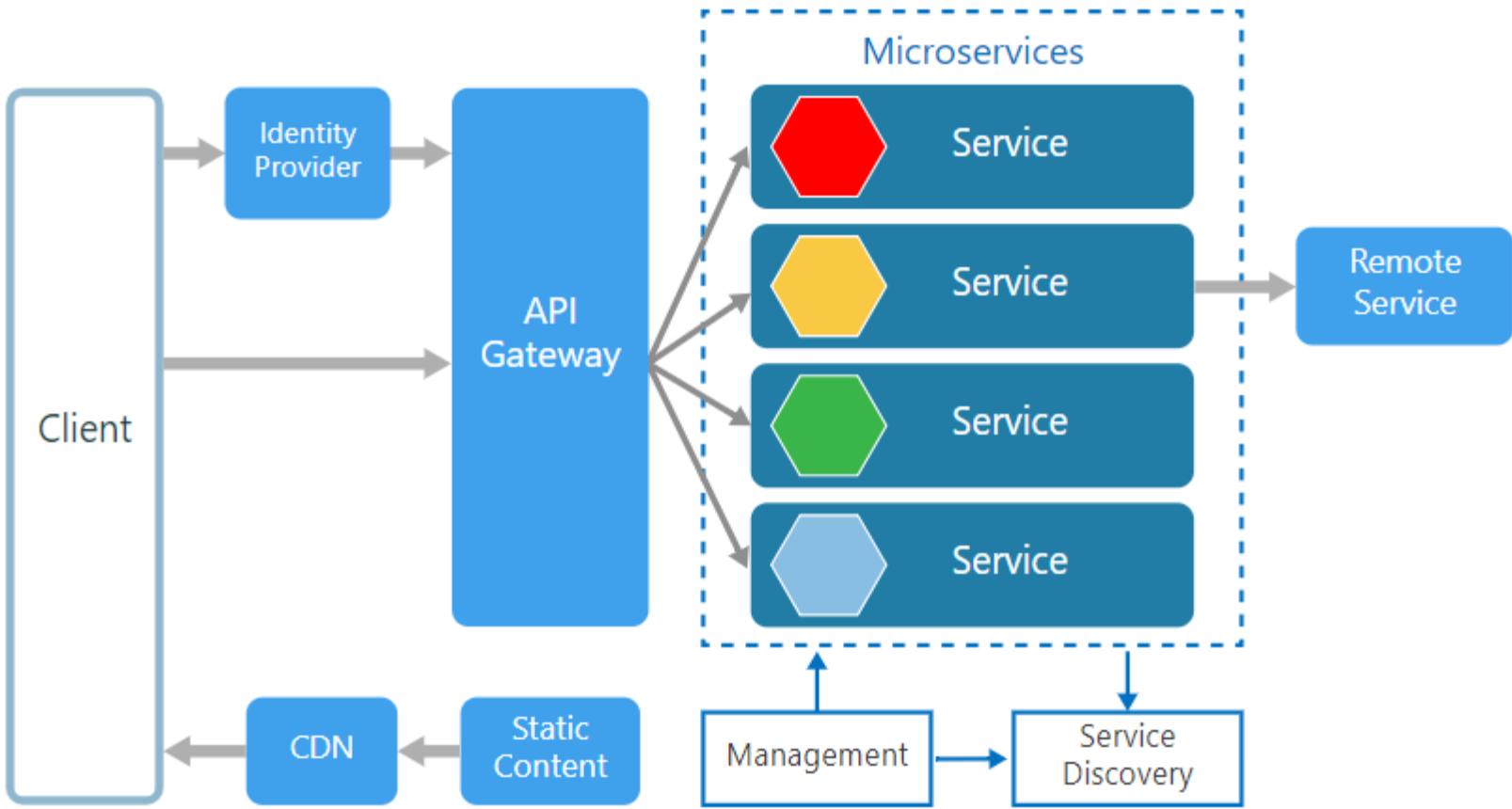
26

- Monolithic → microservices
- build an application as a suite of small services, each running in its own process and are independently deployable.
- Benefits:
 - Simpler To Deploy
 - Simpler To Understand
 - Reusability Across Business
 - Faster Defect Isolation



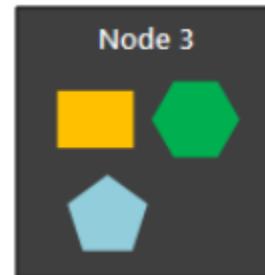
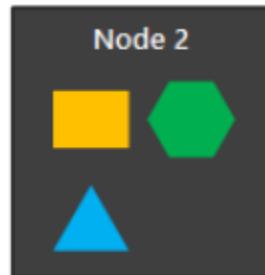
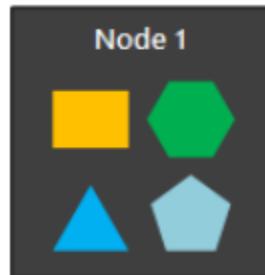
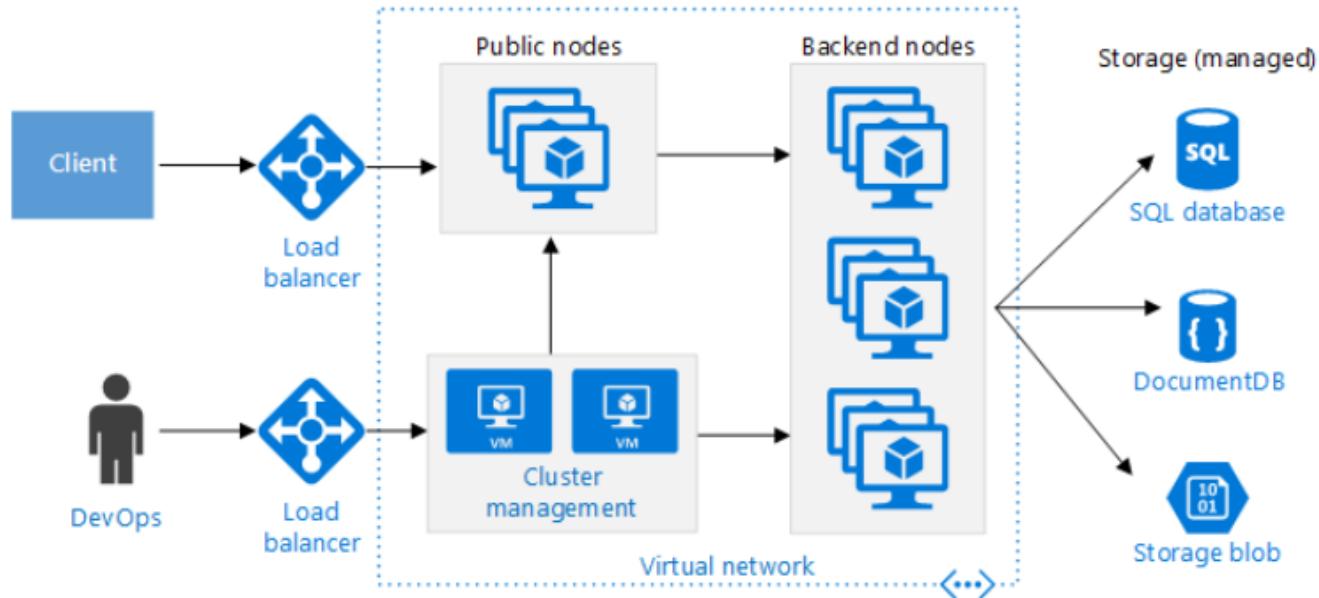
Microservices

27



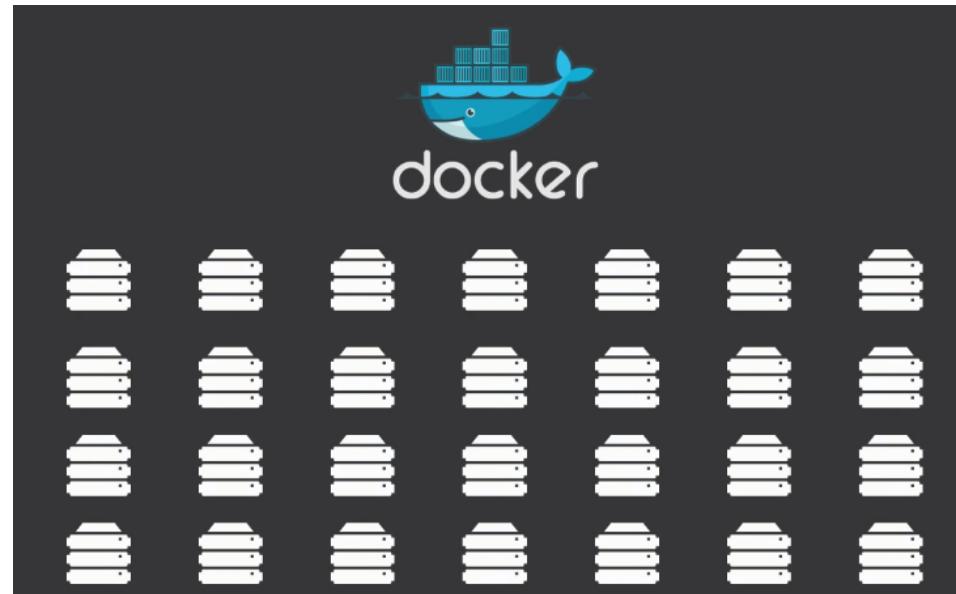
Microservices

28



Problem

29



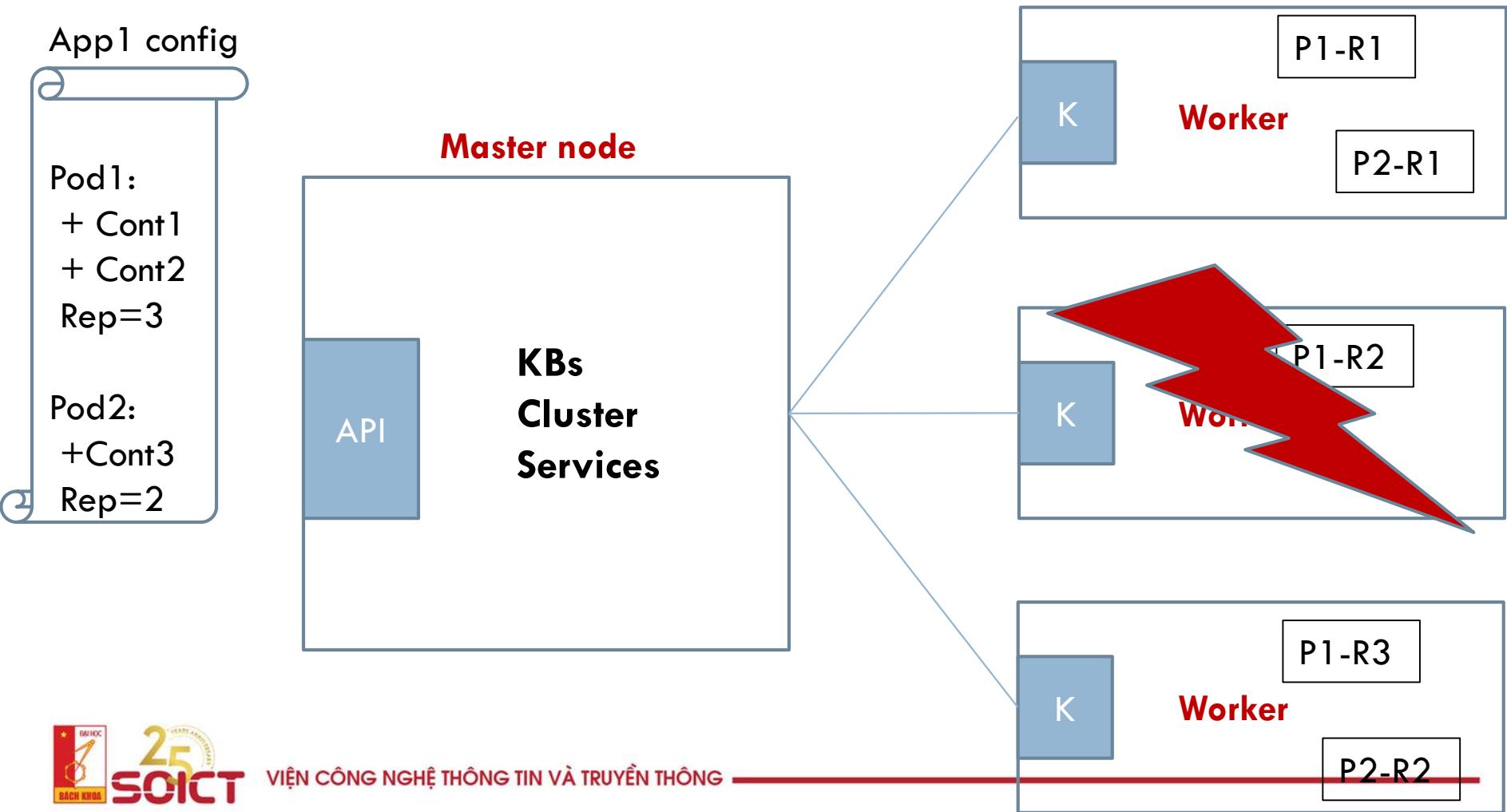
Container Orchestration tools

30

- Amazon ECS (EC2 Container Service)
- Azure Container Service (ACS)
- Cloud Foundry's Diego
- CoreOS Fleet
- Docker Swarm
- Kubernetes

Kubernetes

31



3.3. System architectures

32

Centralized architectures

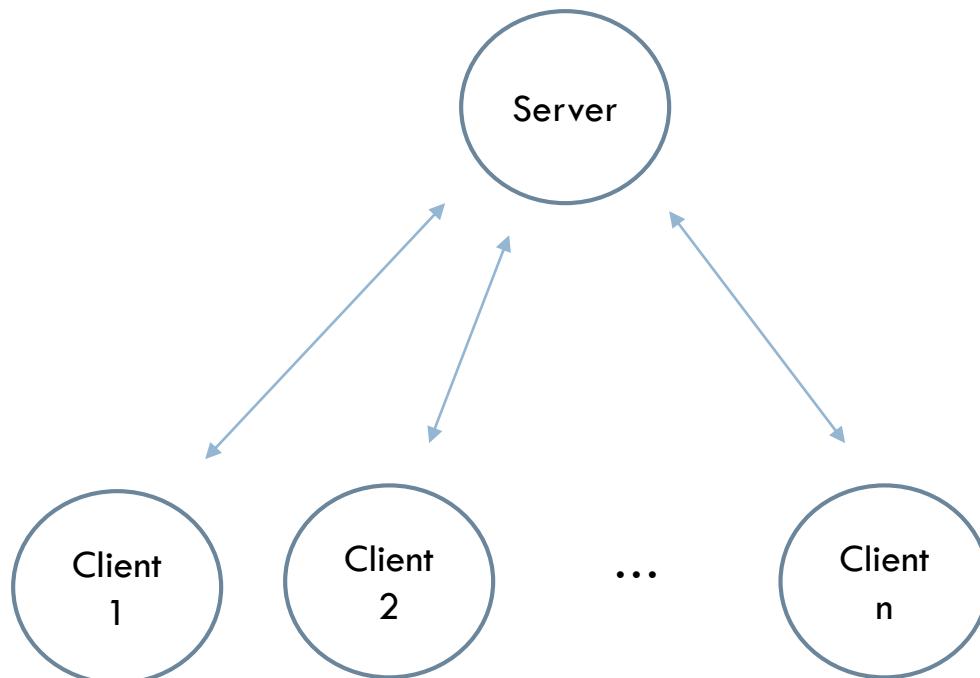
Decentralized architectures

Hybrid architectures

3.3.1. Centralized architectures

33

- Client-server architectures
- Multitiered architectures



3.3.1.1. Client-server architecture

34

Client:

- Send the requests, receive the results, show to the users

Server:

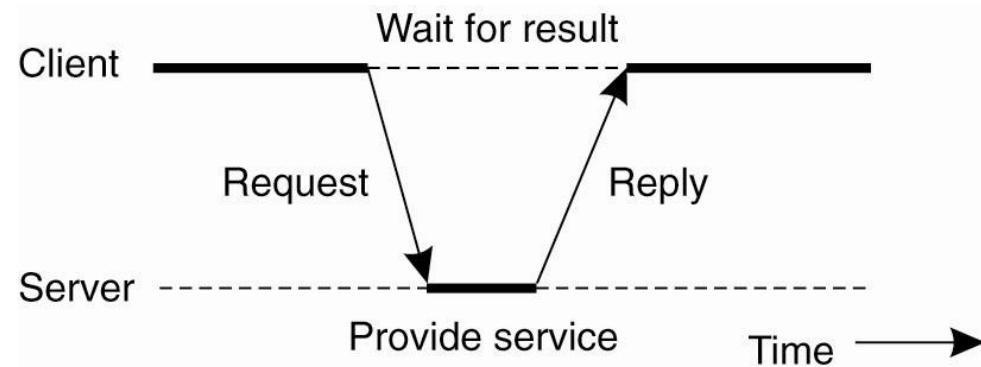
- Listen; receive the request, processing, reply

Connection-oriented protocol

Connectionless protocol

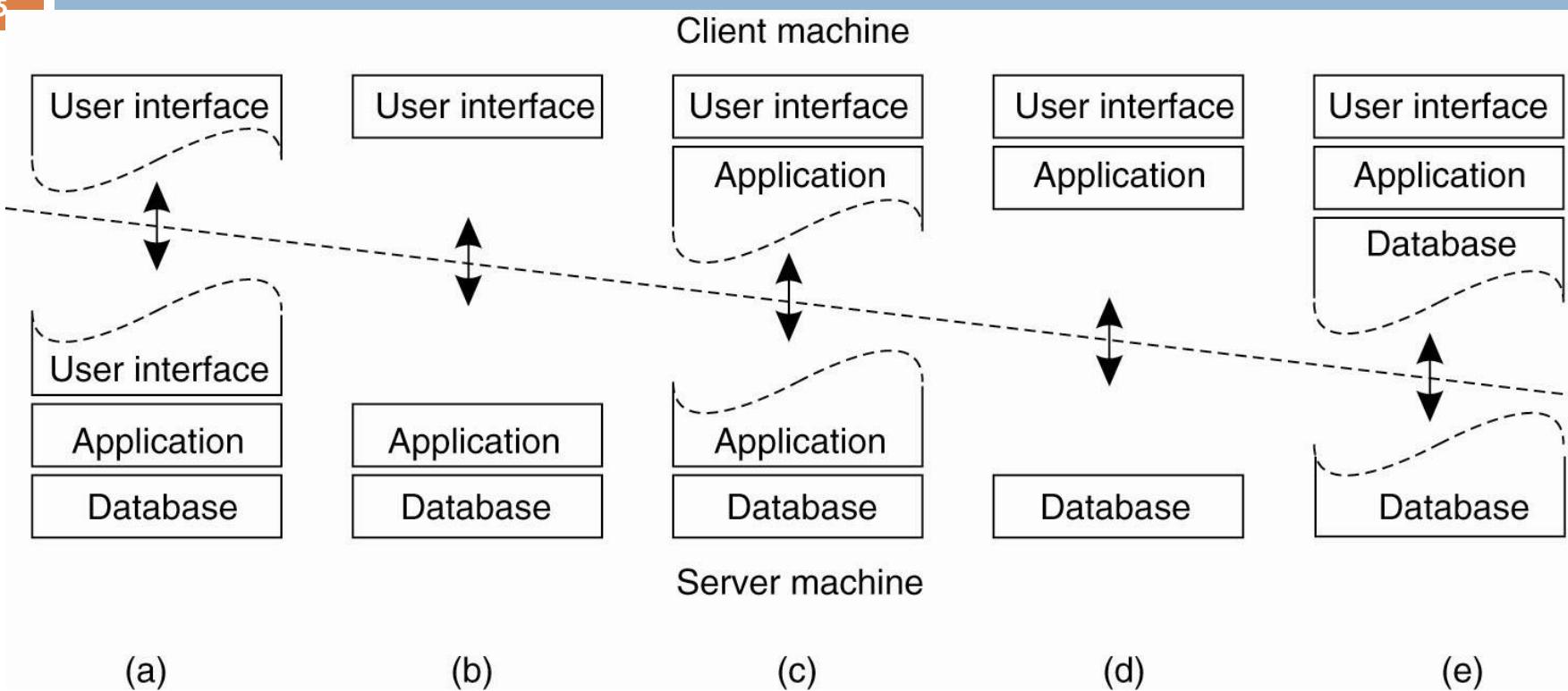
Issues:

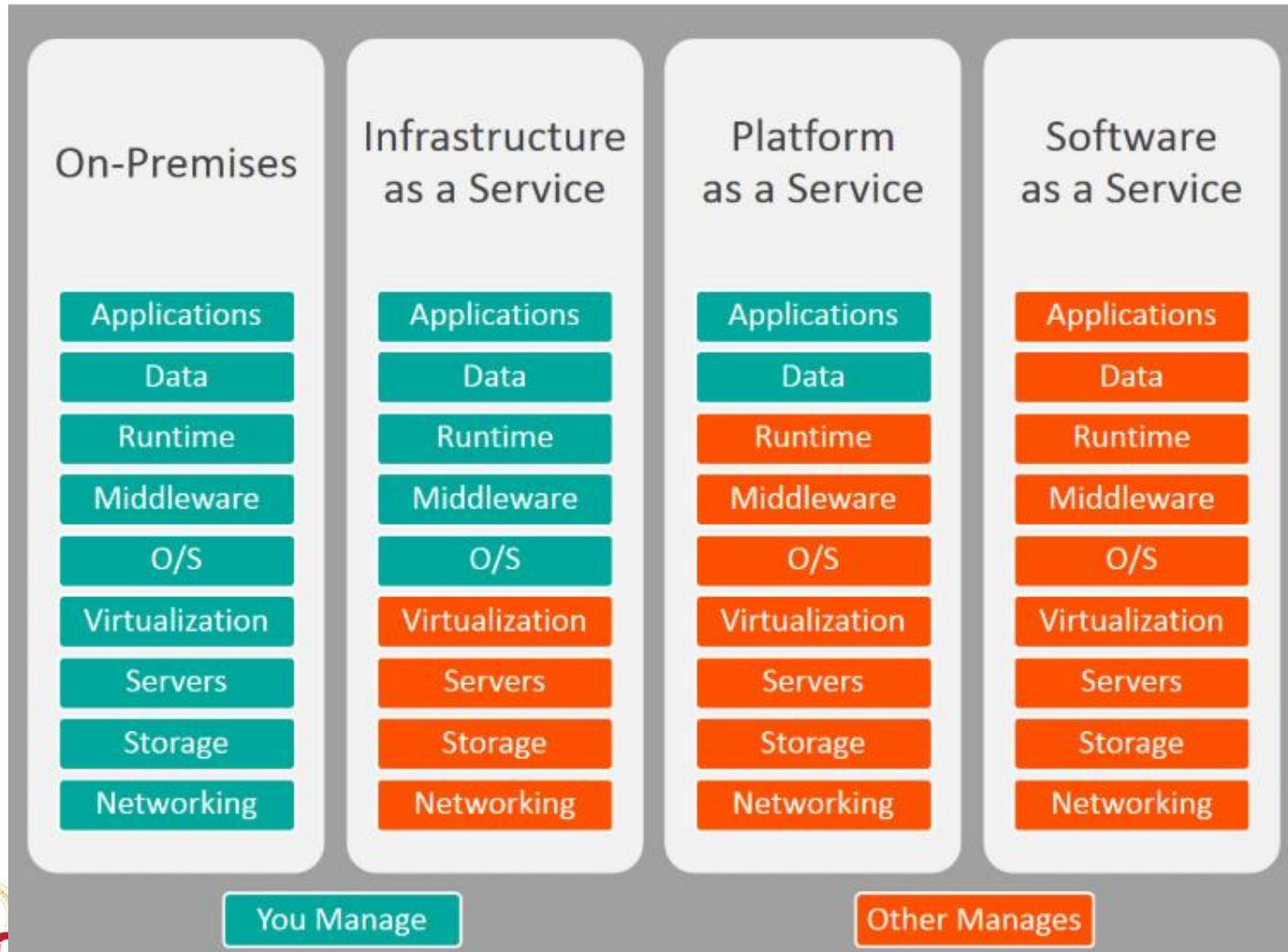
- Register the server
- Idempotent
- Stateful or Stateless server



3.3.1.2. Multitiered architectures

35





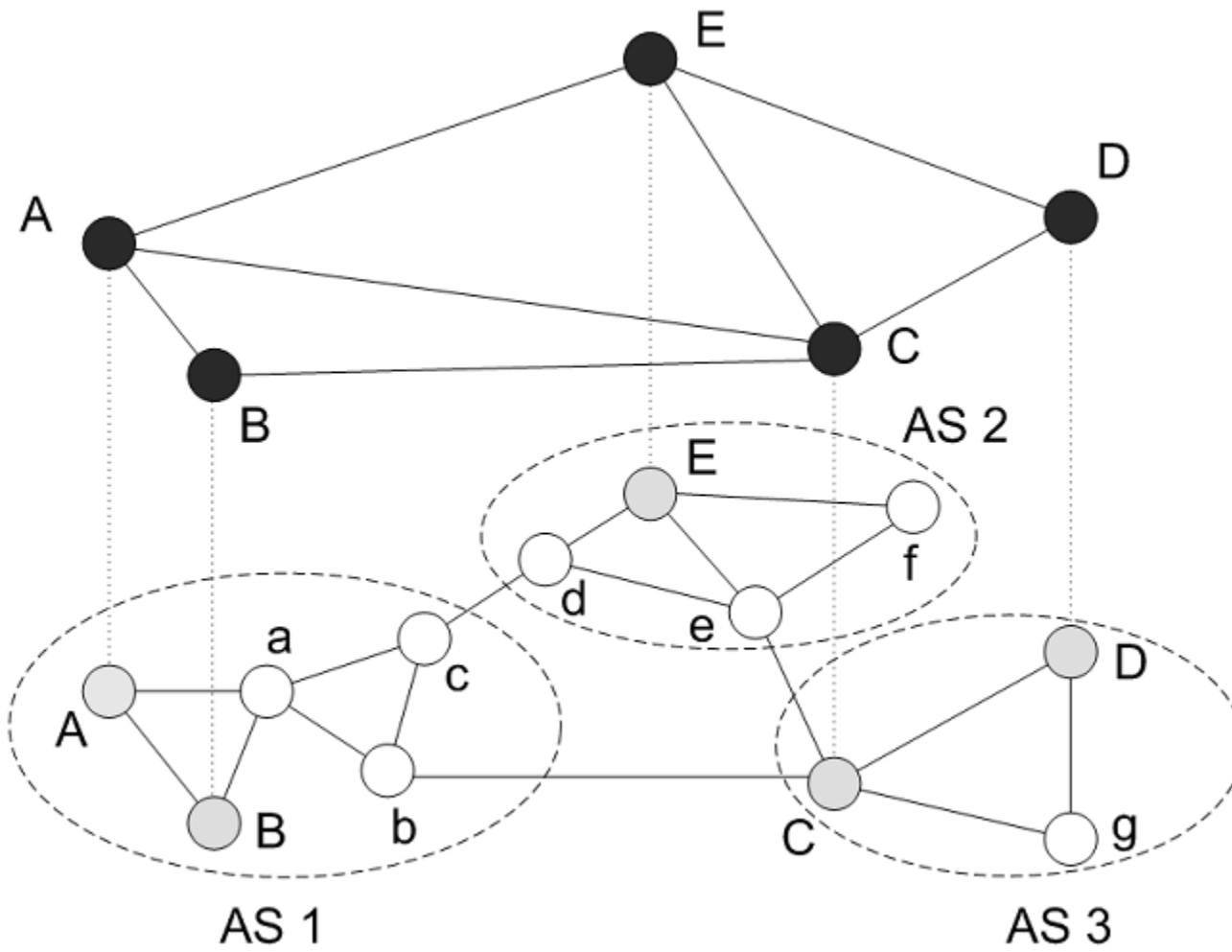
3.3.2. Decentralized Architectures

37

- No role of client and server
- Use Overlay network
- Structured/Unstructured P2P architectures

Overlay network

38



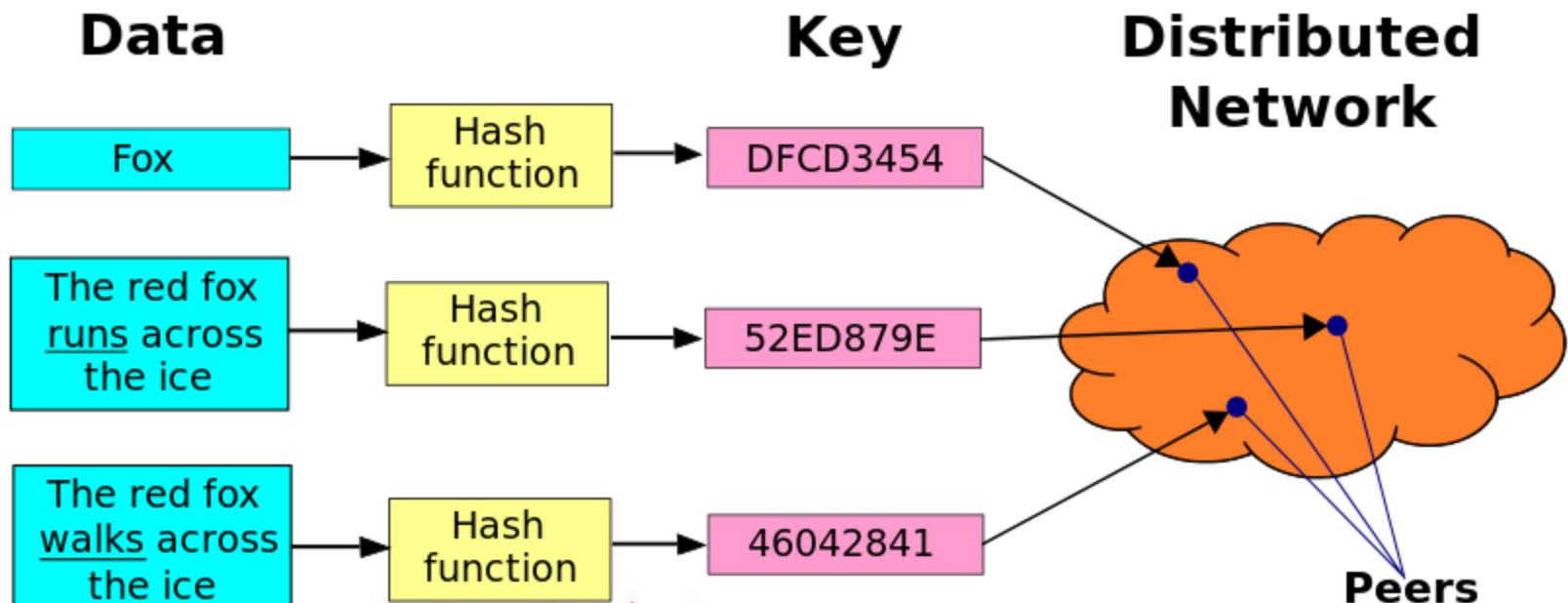
Overlay
Network

Physical
Network

2.2.1. Structured P2P

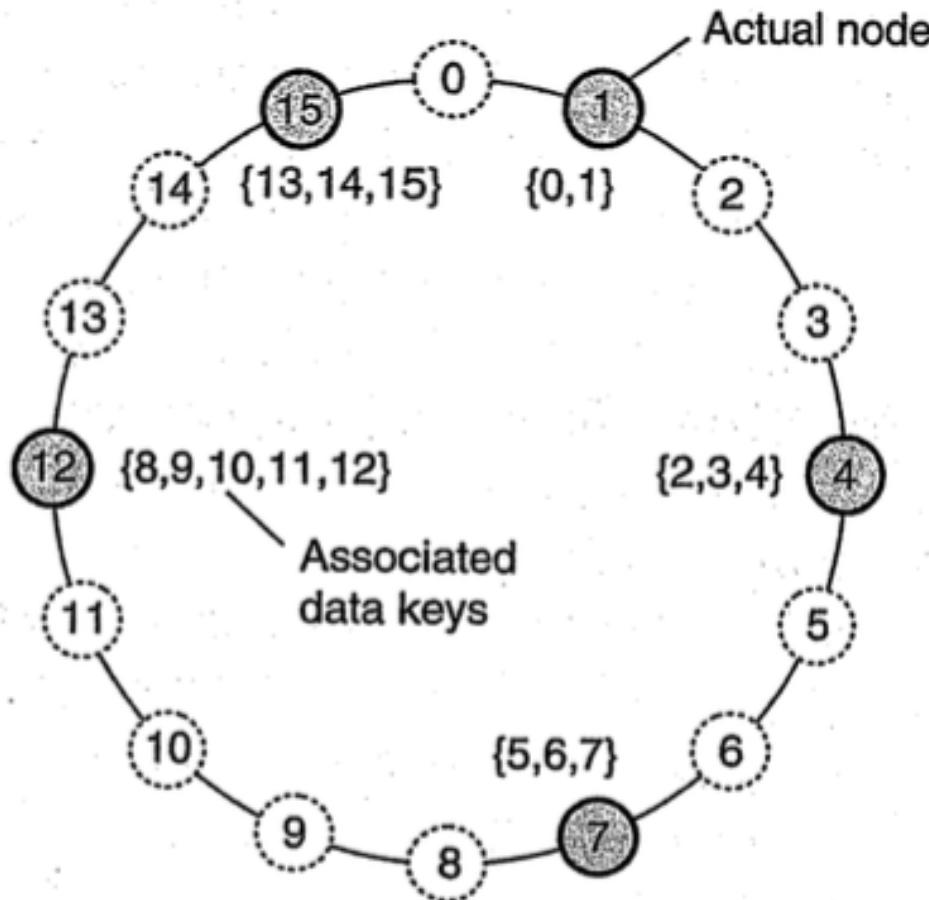
39

- Overlay network is constructed using a deterministic procedure.
- DHT (Distributed Hash Table)



Chord system

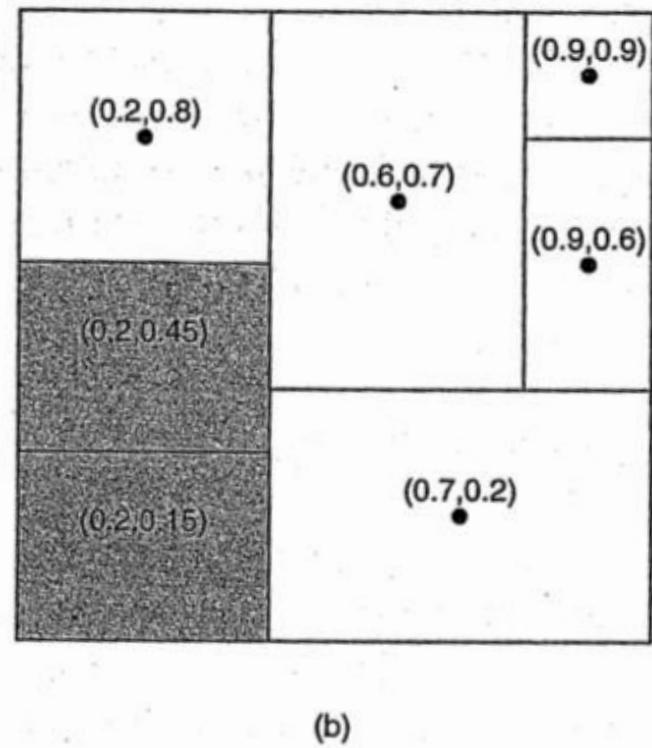
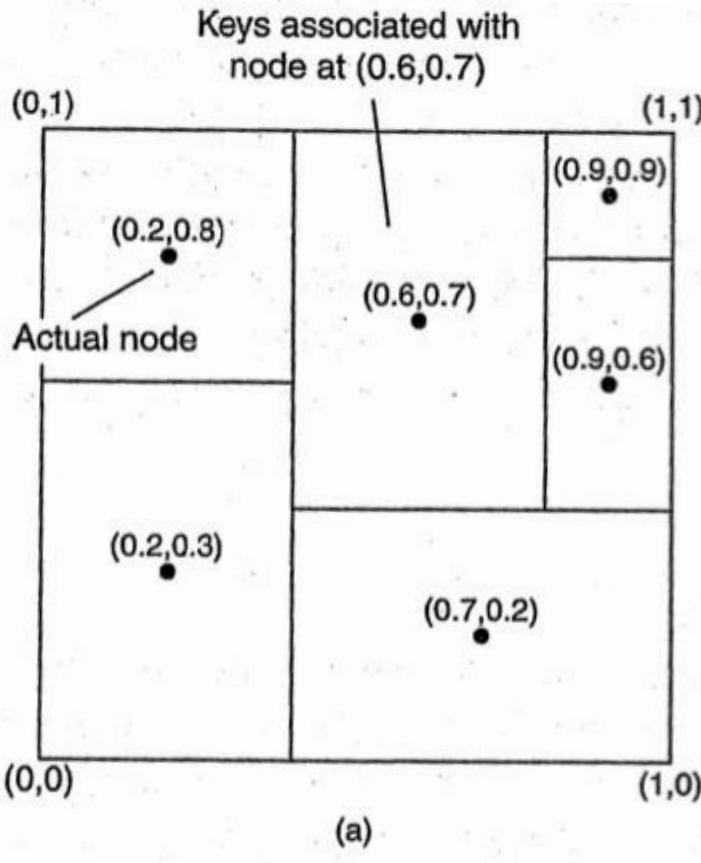
40



- Logically organized in a ring
- $\text{Succ}(k)$
- Function $\text{LOOKUP}(k)$
- When a node wants to join the system
- When a node wants to leave the system

CAN system (Content Addressable Network)

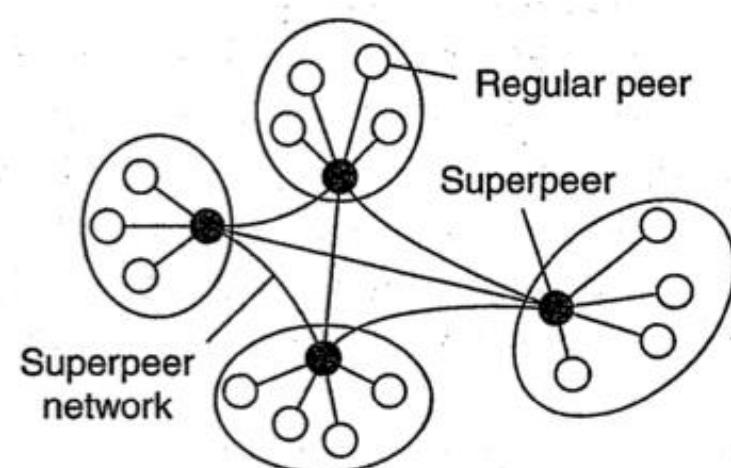
41



2.2.2. Unstructured P2P architecture

42

- Randomized algorithms for constructing an overlay network.
- Each node maintains a list of neighbors
- Data items are assumed to be randomly placed on nodes → locating a specific data item needs flooding the network
- =>superpeers



2.3. Hybrid architectures

43

- Edge-Server Systems
- Collaborative Distributed Systems

Edge-Server Systems

44

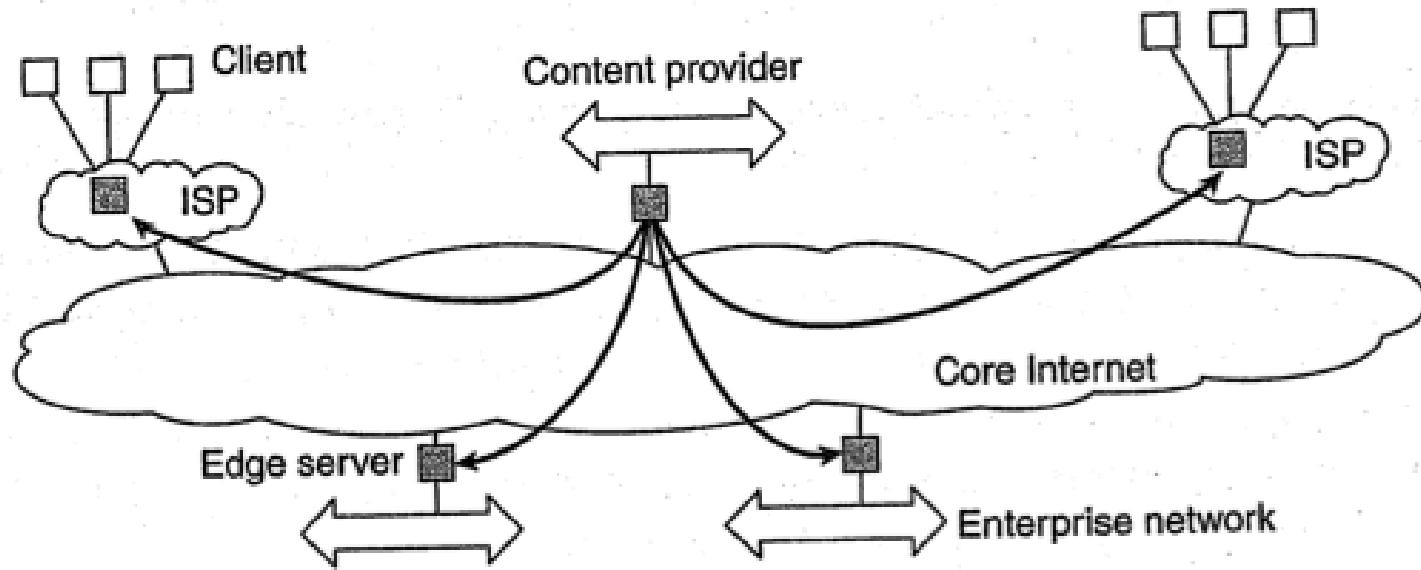
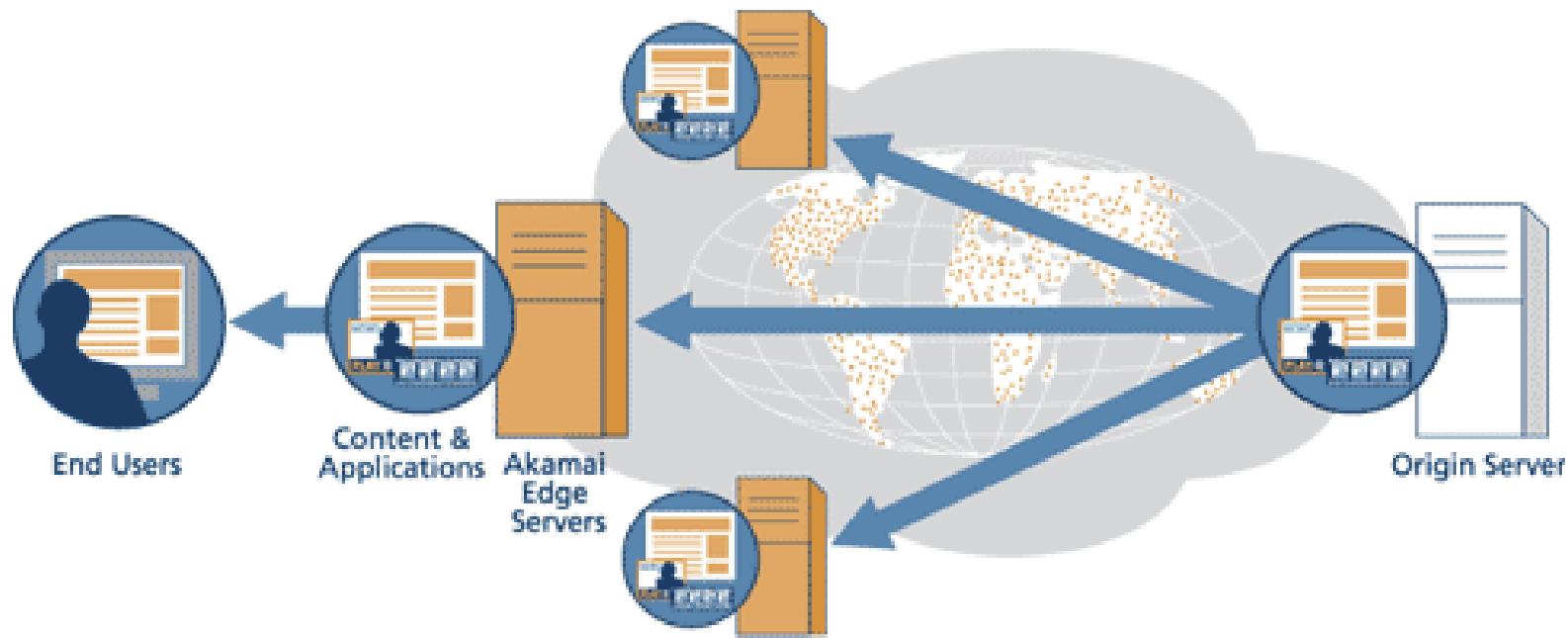


Figure 2-13. Viewing the Internet as consisting of a collection of edge servers.

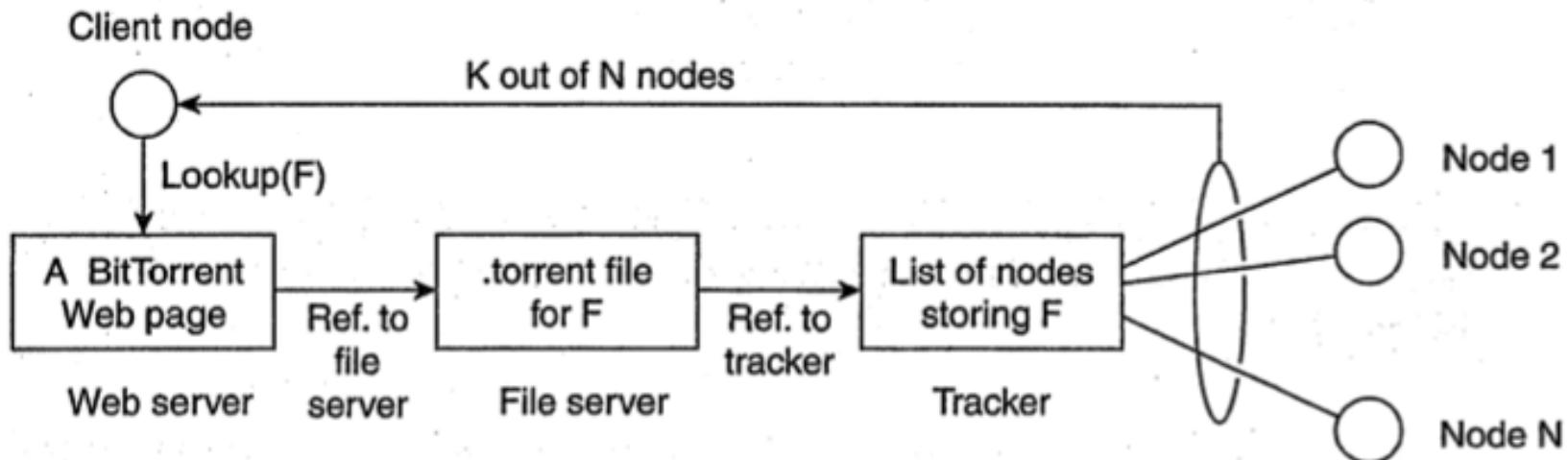
Content Delivery Network

45



Collaborative Distributed Systems

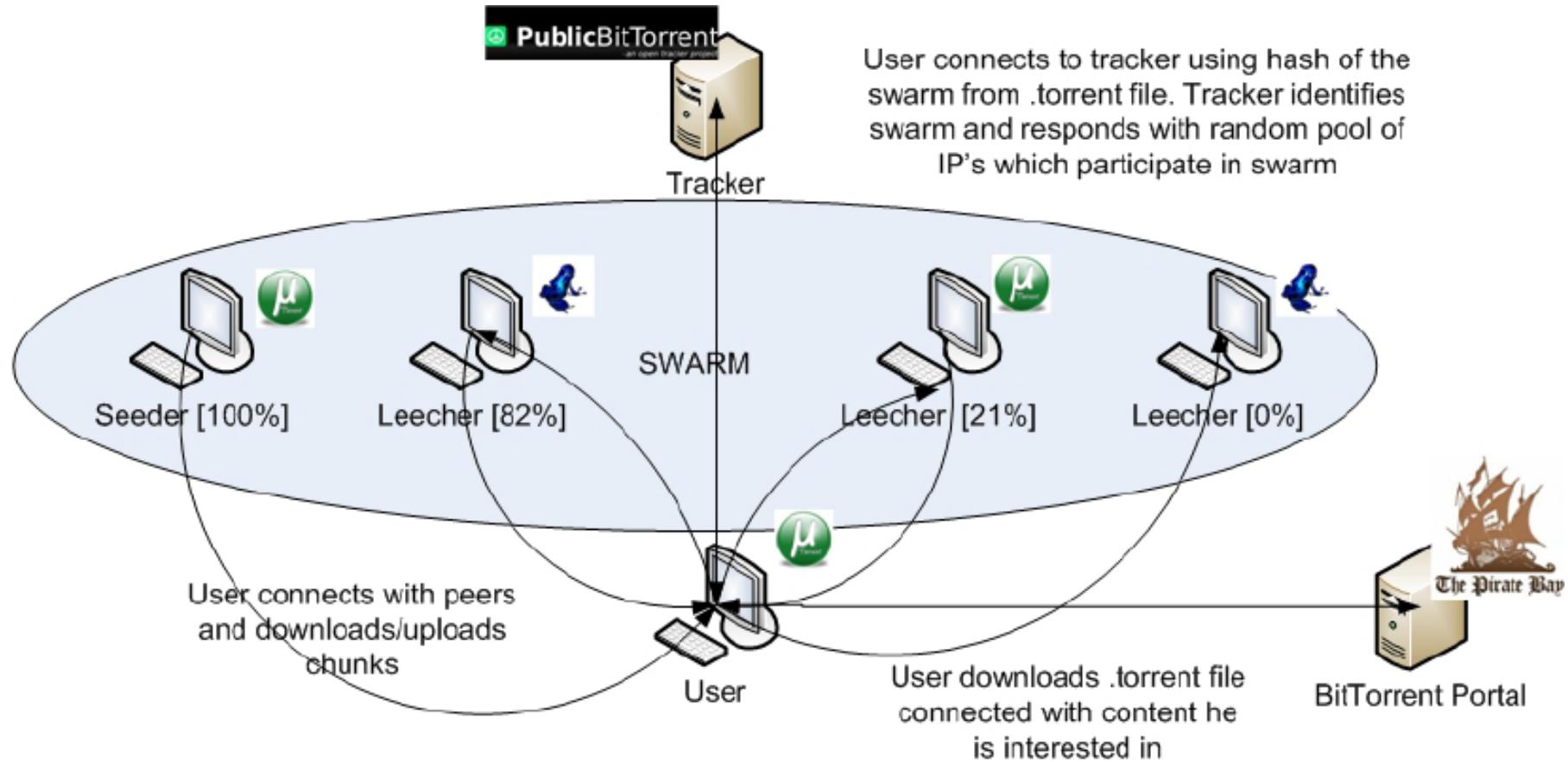
46



BitTorrent system

Example: BitTorrent

47



4. Middleware in Distributed Systems

4.1. DOS, NOS

4.2. Middleware

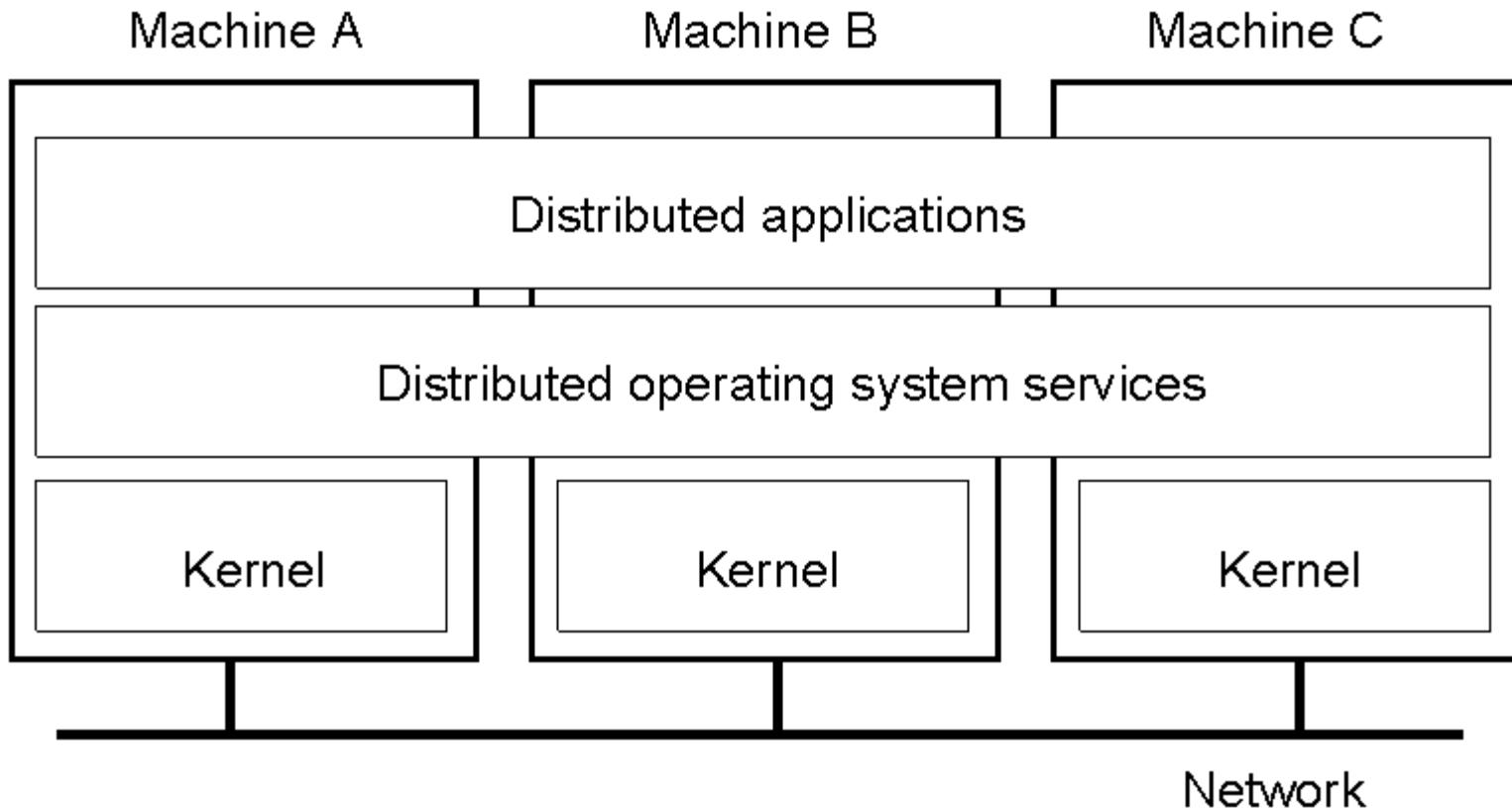
4.1. Software

System	Description	Main Goal
DOS	Multicomputer, multiprocessors	Transparency
NOS	NOS on local machines	Local services for other machines
Middleware	Provide basic services to develop apps	Distributed transparency

- DS is similar to OS
 - Handle the resources
 - Hide the complexity and heterogeneity
- 2 types:
 - tightly-coupled systems (DOS)
 - loosely coupled systems (NOS)

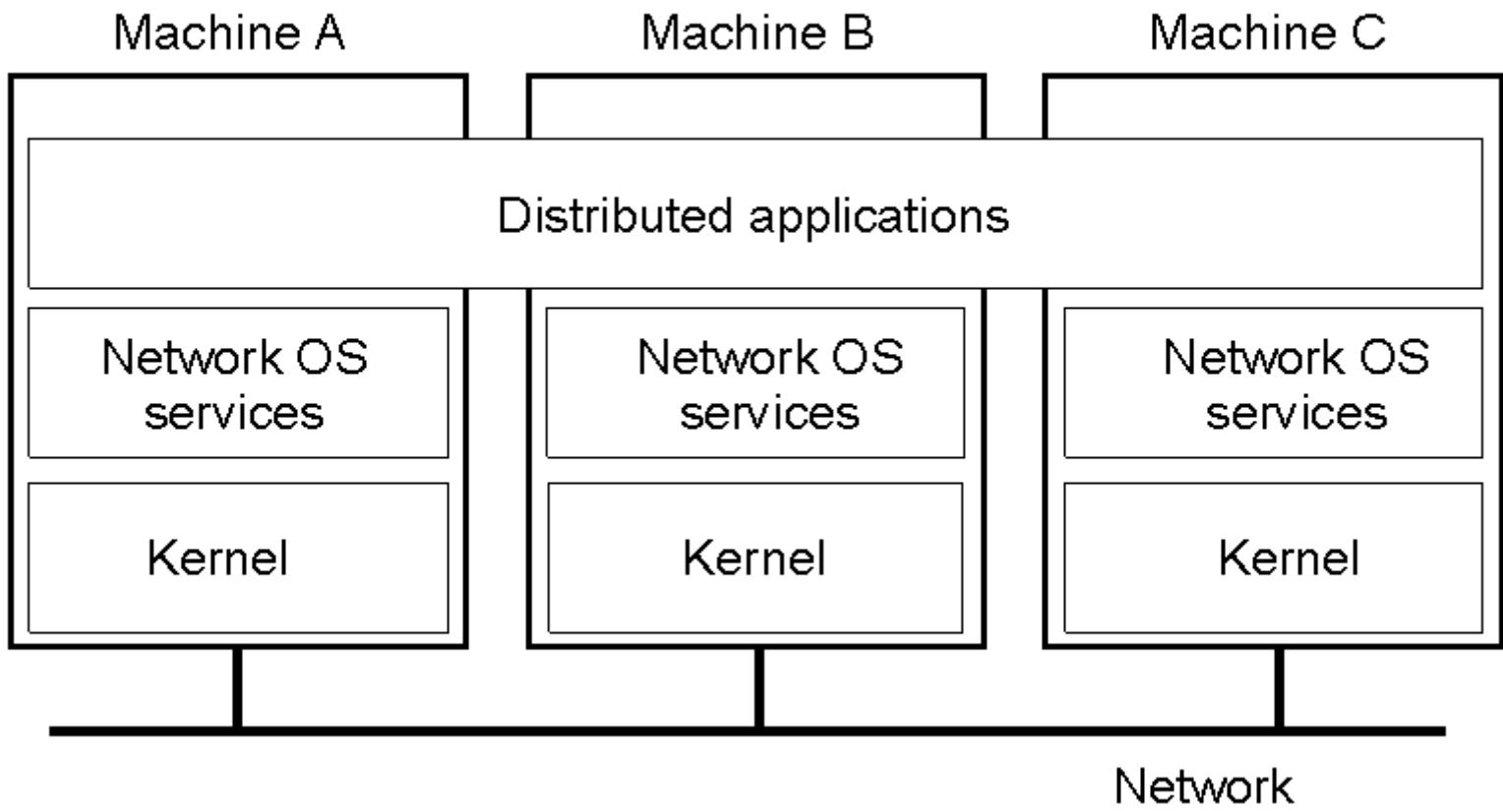
4.1.1. Distributed Operating Systems (DOS)

50



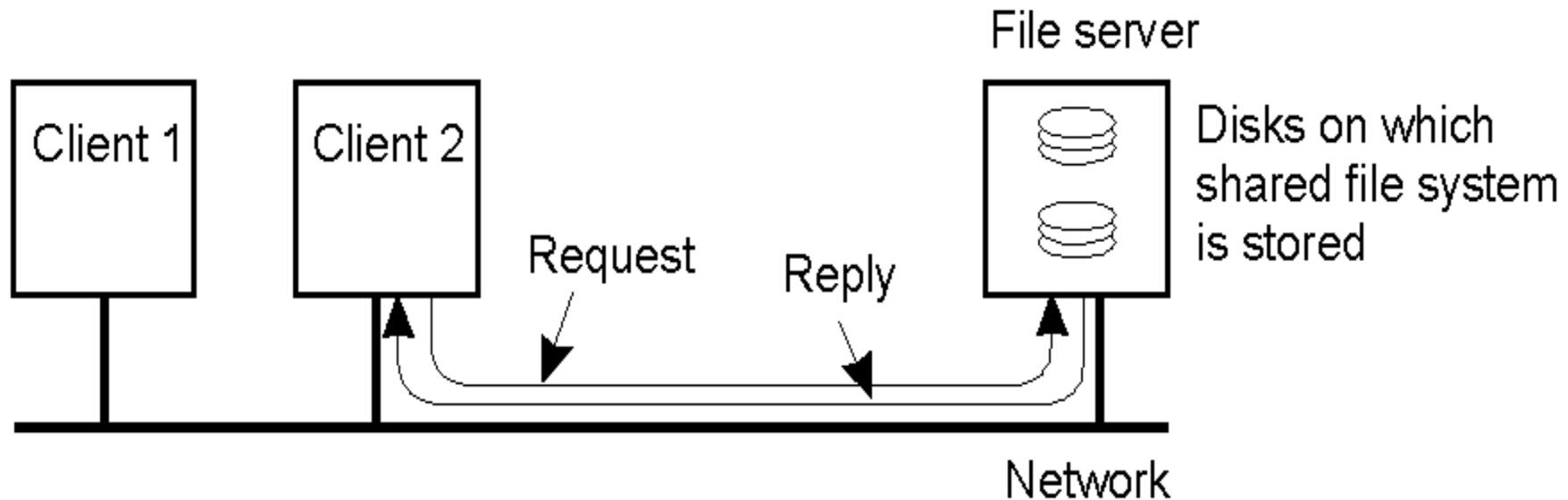
4.1.2. Network OS

51



Network OS

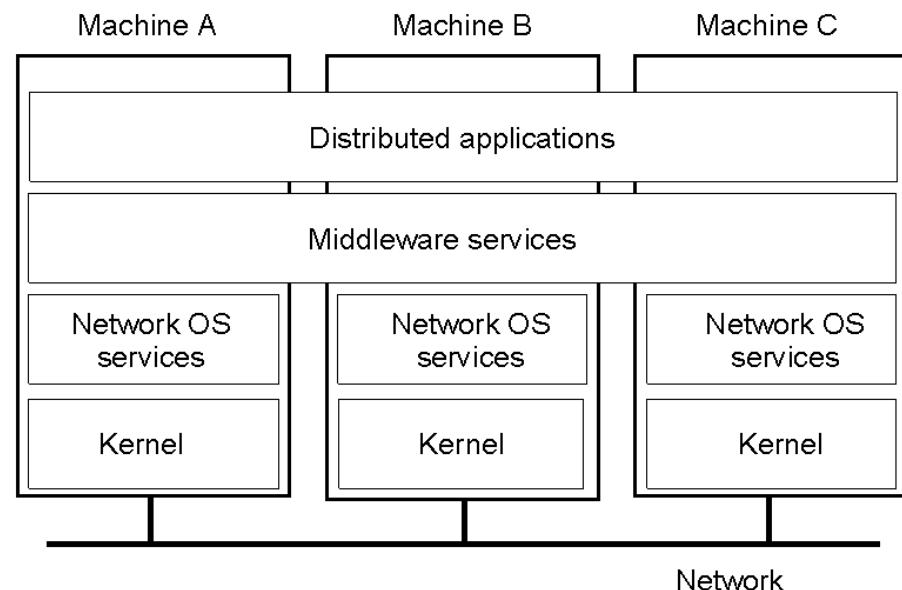
52



4.2. Middleware

53

- Combine advantages of DOS and NOS
- Middleware
- E.g:
 - File system in UNIX
 - RPC
- Middleware service:
 - Transparent access
 - High level communication facilities



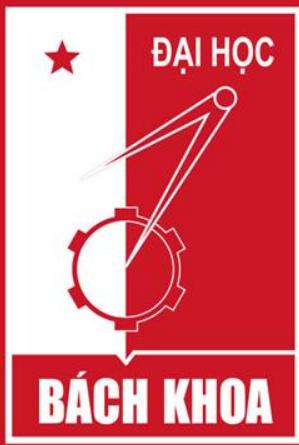


25
YEARS ANNIVERSARY
SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Questions?





25 YEARS ANNIVERSARY
SICT

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

DISTRIBUTED SYSTEMS



HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

CHAPTER 2: PROCESSES AND COMMUNICATION IN DISTRIBUTED SYSTEMS

Dr. Trần Hải Anh

Outline

3

1. Process and Thread
2. Overview of Communication in DS
3. RPC
4. Message-Oriented Communication
5. Stream-Oriented Communication

1. Process and Thread

1.1. Introduction

1.2. Threads in centralized systems

1.3. Threads in distributed systems

1.1. Introduction

5

❑ Process

- ❑ A program in execution
- ❑ Creating a process:
 - Create a complete independent address space
 - Allocation = initializing memory segments by zeroing a data segment, copying the associated program into a text segment, setup a stack for temporary data
- ❑ Resources
 - Execution environment, memory space, registers, CPU...
 - Virtual processors
 - Virtual memory
- ❑ Concurrency transparency
- ❑ Switching the CPU between processes: Saving the CPU context + modify registers of MMU, ...

Thread

6

- A thread executes its own piece of code, independently from other threads.
- Process has several threads → multithreaded process
- Threads of a process use the process' context together
- Thread context: CPU context with some other info for thread management.
- Exchanging info by using shared variable (mutex variable)
- Protecting data against inappropriate access by threads within a single process is left to application developers.

1.2. Multi-threading and multi-processing

7

- Parallel processing benefits with sequential processing
- Multithreaded program vs multi-processes program
 - Programming cost
 - Switching context
 - Blocking system calls

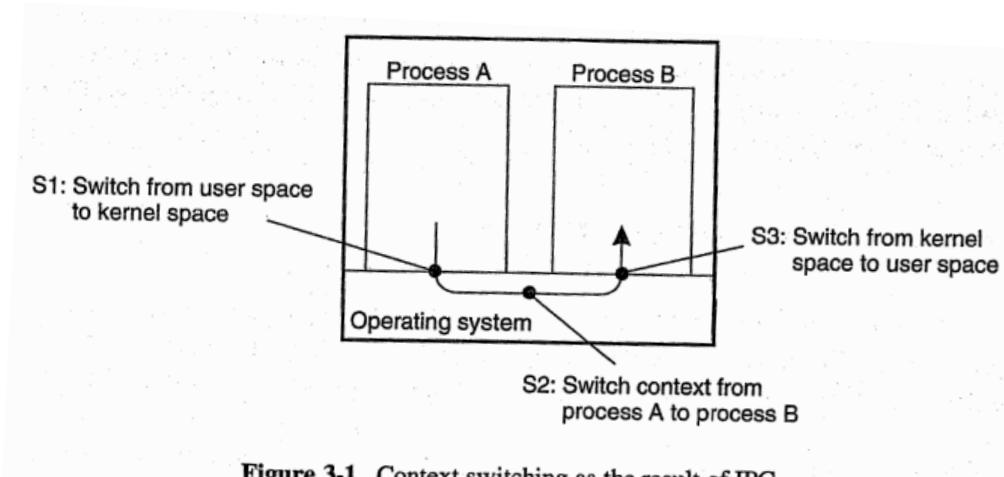


Figure 3-1. Context switching as the result of IPC.

Thread implementation

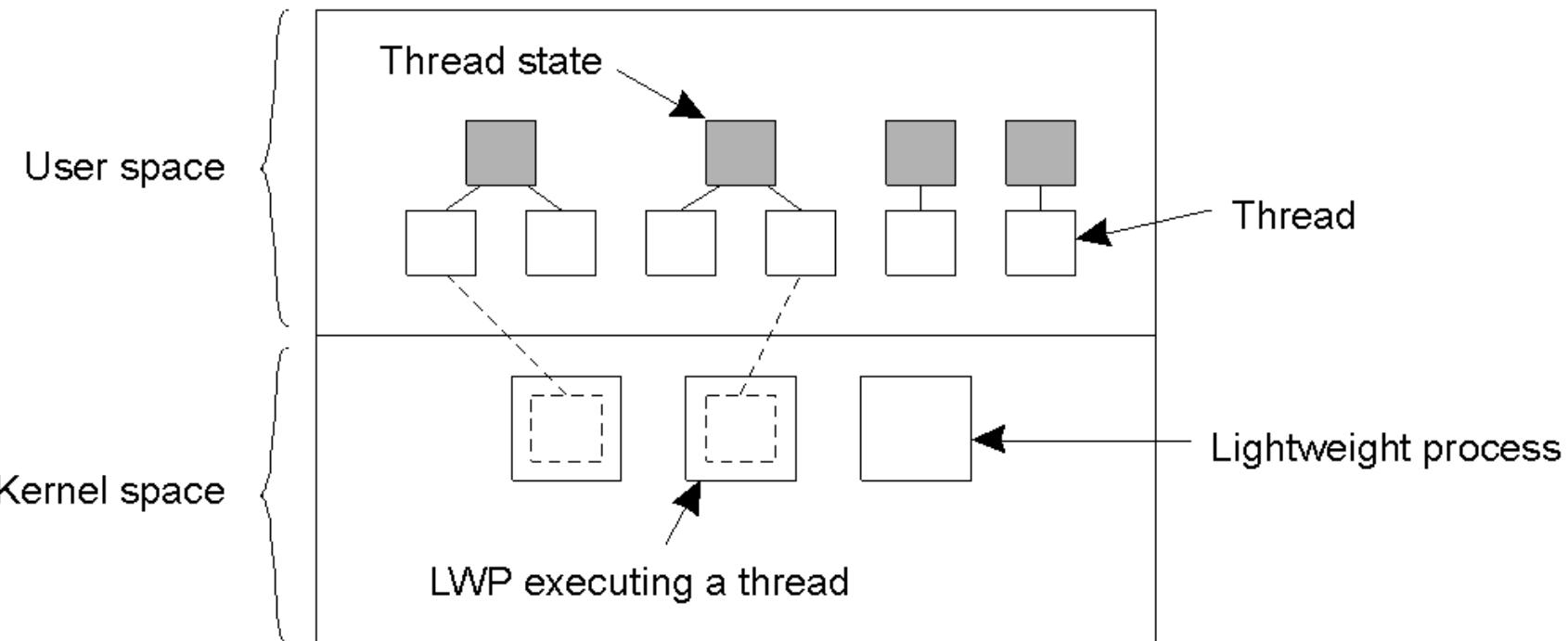
8

- Thread package:
 - Creating threads (1)
 - Destroying threads (2)
 - Synchronizing threads (3)
- (1), (2), (3) can be operated in user mode and kernel mode:
 - User mode:
 - Cheap to create and destroy threads
 - Easy to switch thread context
 - Invocation of a blocking system call will block the entire process
 - Kernel mode:

Lightweight processes (LWP) on Linux

9

- Combining kernel-level lightweight processes and user-level threads.



1.3. Threads in Distributed Systems

10

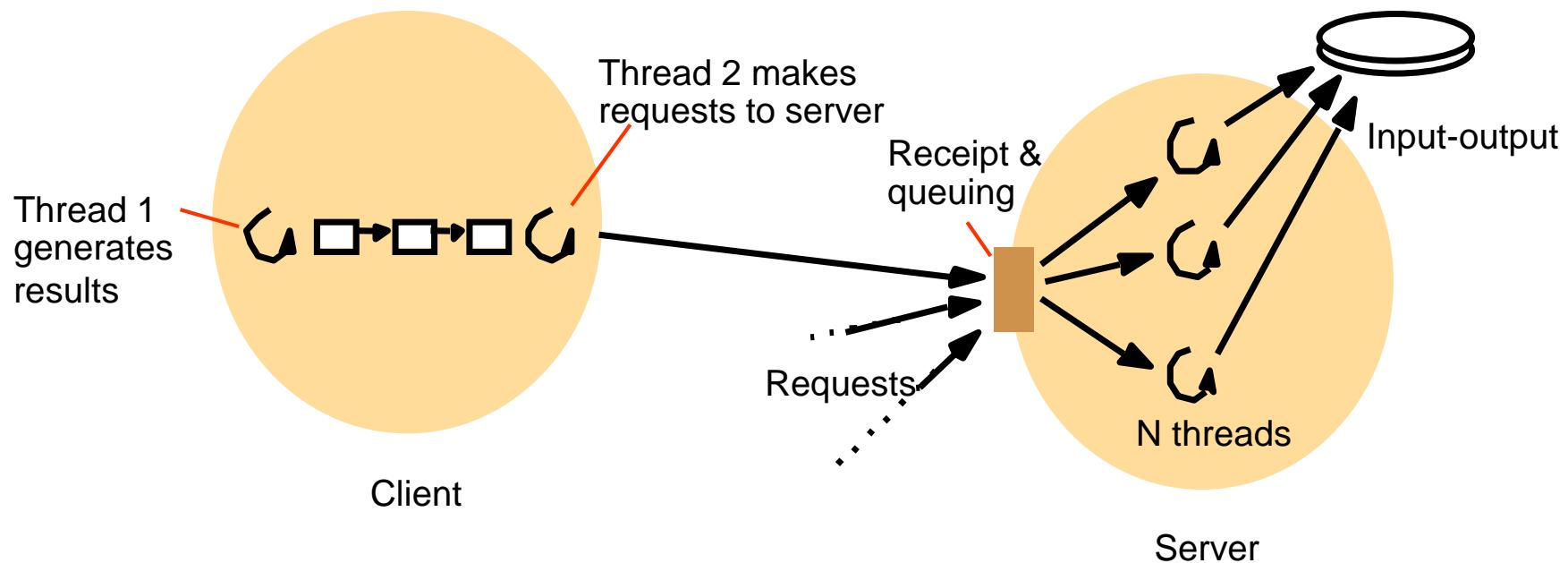
- Single-threaded server
 - One request at one moment
 - Sequentially
 - Do not guarantee the transparency



Multi-threaded server

Multithreaded Client and server

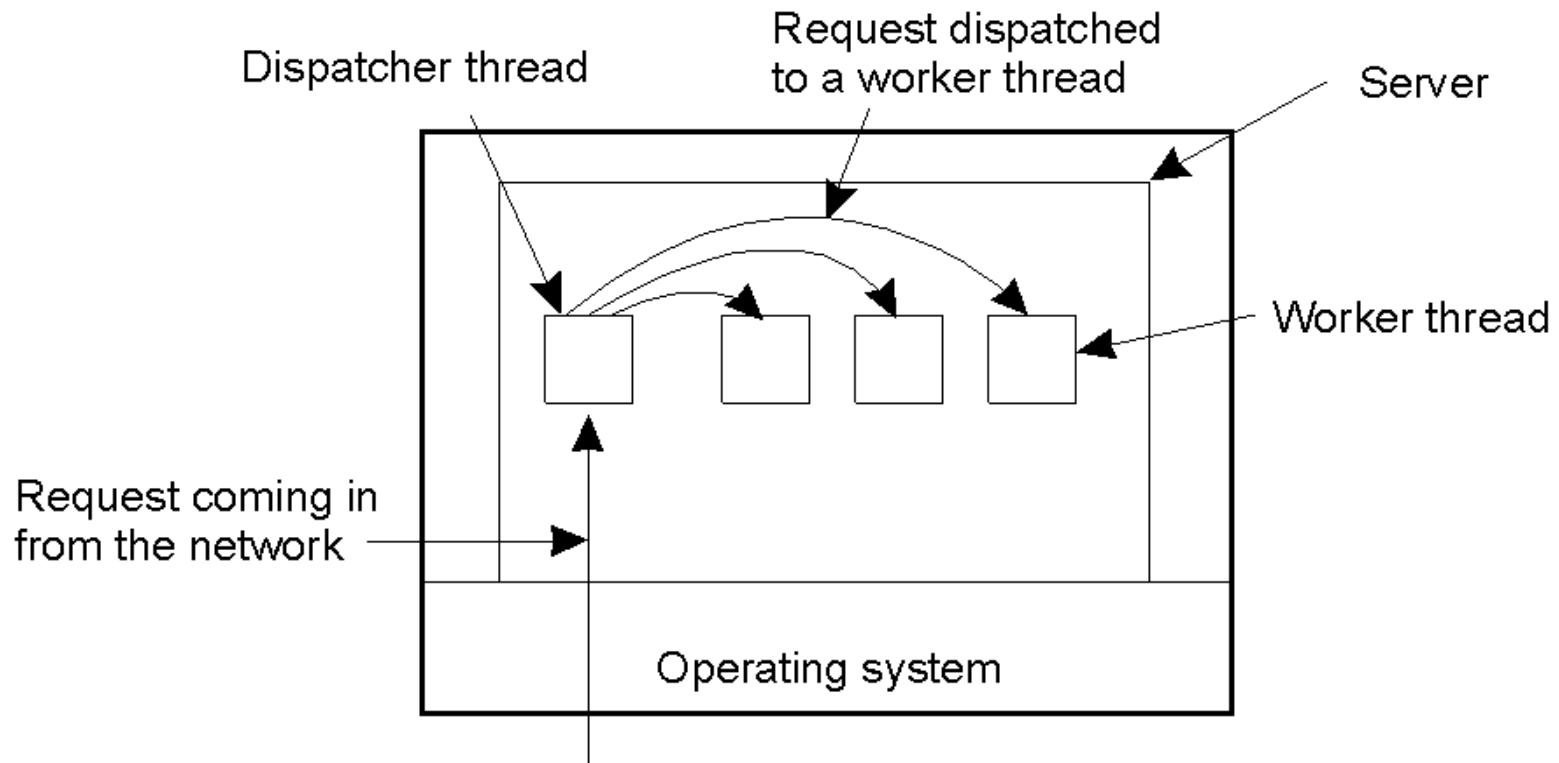
11



1.4. Server in Distributed Systems

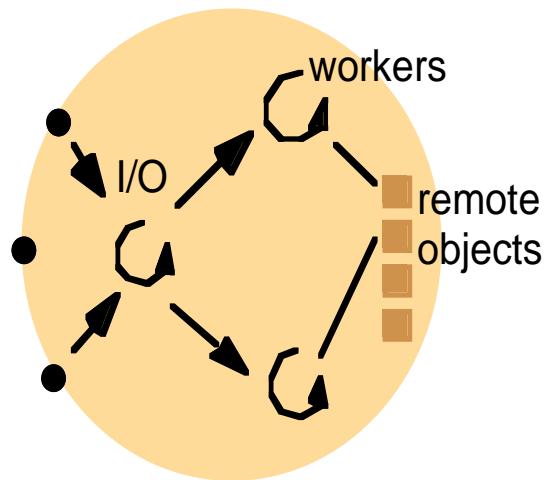
12

Server dispatcher model

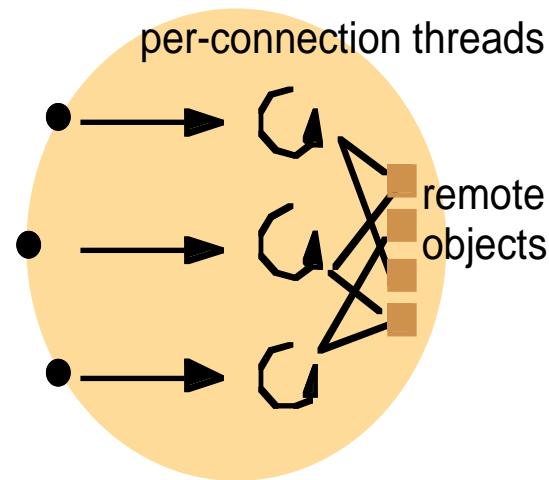


Multithreaded Server

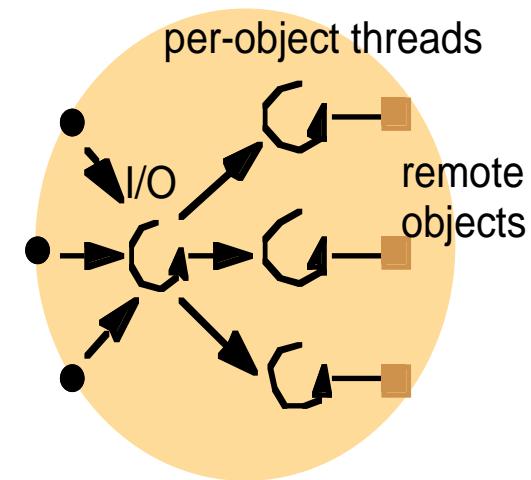
13



a. Thread-per-request



b. Thread-per-connection



c. Thread-per-object

Finite-state machine

14

- Only one thread
- Non-blocking (asynchronous)
- Record the state of the current request in a table
- Simulating threads and their stacks
- Example: Node.js
 - ▣ Asynchronous and Event-driven
 - ▣ Single threaded but highly scalable

Comparison

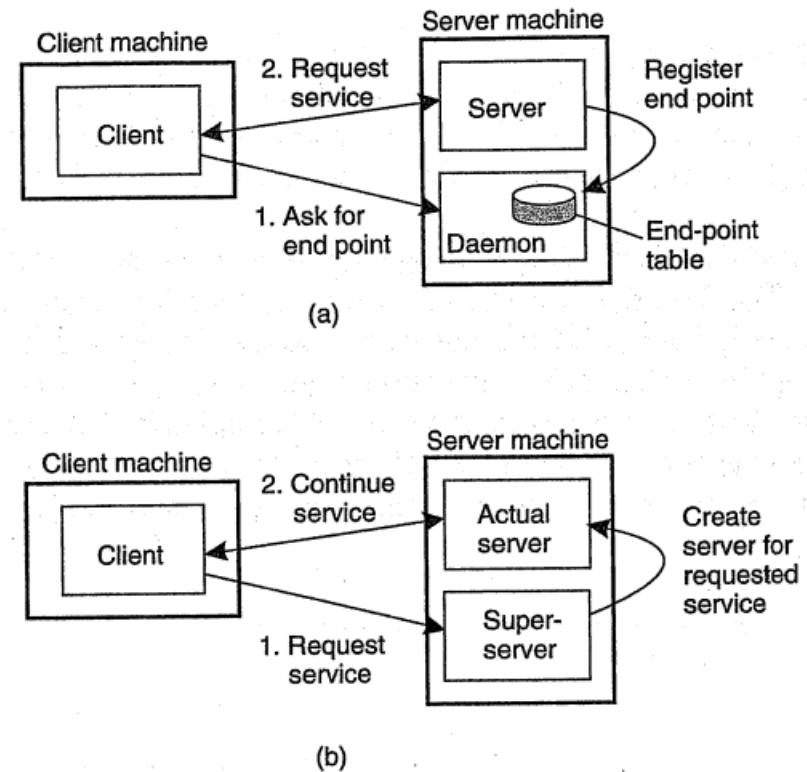
15

Model	Characteristics
Threads	Parallelism, Blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite-state machine	Parallelism, Non-blocking system calls

General design issues

16

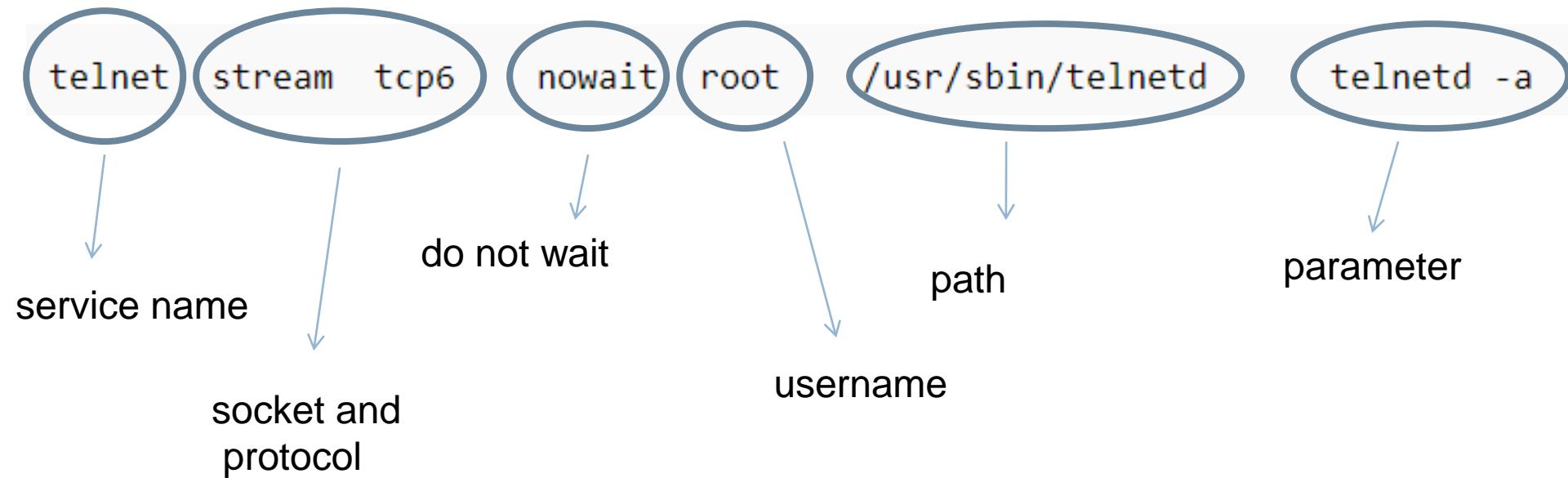
- Organize server
 - Iterative server
 - Concurrent server
- Find server:
 - End-point (port)
 - Daemon
 - Superserver
- Interrupt server
- Stateless & stateful server



Inetd

17

- Configuration info in the file */etc/inetd.conf*



Example:

□ A program *errorLogger.c*

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
    const char *fn = argv[1];
    FILE *fp = fopen(fn, "a+");

    if(fp == NULL)
        exit(EXIT_FAILURE);

    char str[4096];
    //inetd passes its information to us in stdin.
    while(fgets(str, sizeof(str), stdin)) {
        fputs(str, fp);
        fflush(fp);
    }
    fclose(fp);
    return 0;
}
```

Configure inetd

19

- Insert info into */etc/services*

```
errorLogger 9999/udp
```

- Insert info into */etc/inetd.conf*

```
errorLogger dgram udp wait root  
/usr/local/bin/errlogd errlogd  
/tmp/logfile.txt
```

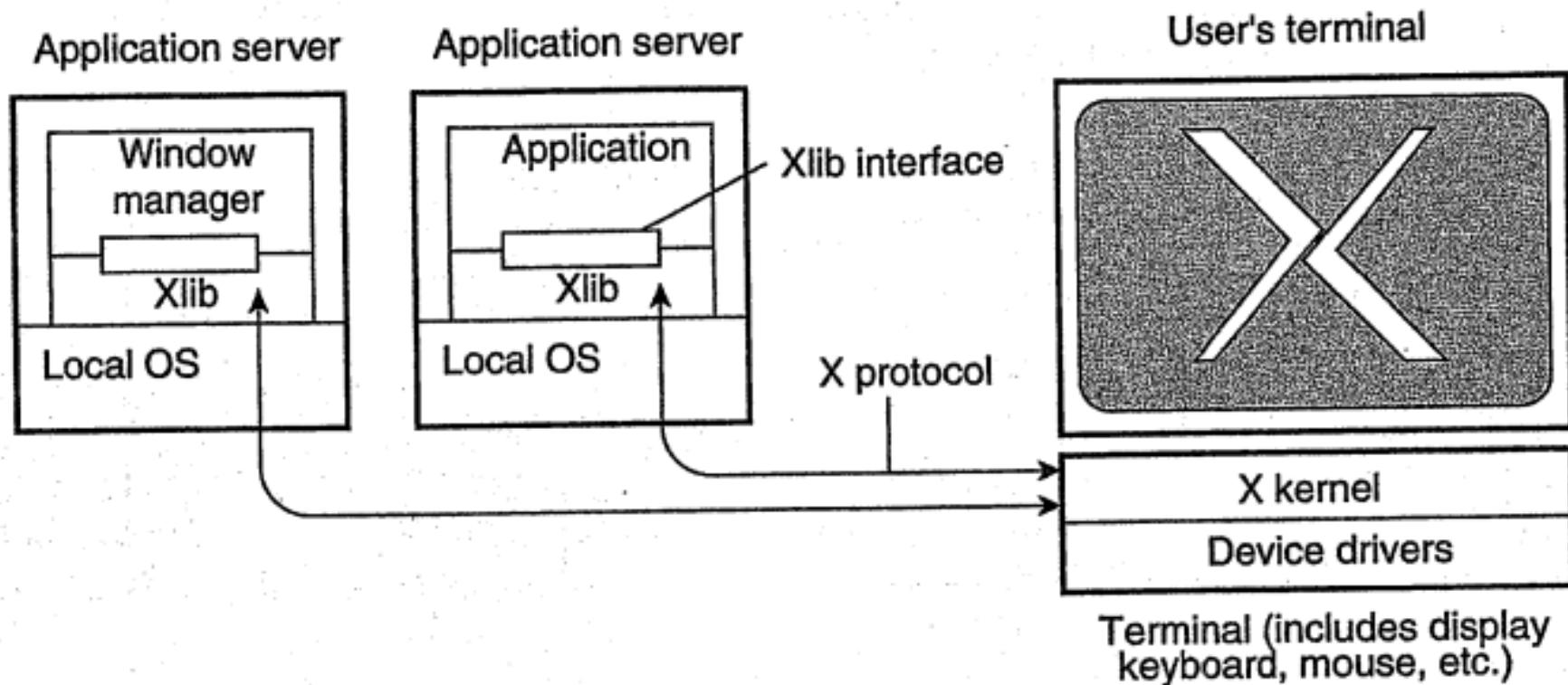
1.5. Client in Distributed Systems

20

- Multi-threaded client:
 - Separate user interface and handle
 - Solve the problem of mutual exclusion
 - Enhance performance when working with many different servers
 - Example: Website loading

Solution for thin-client model: X Window System

21

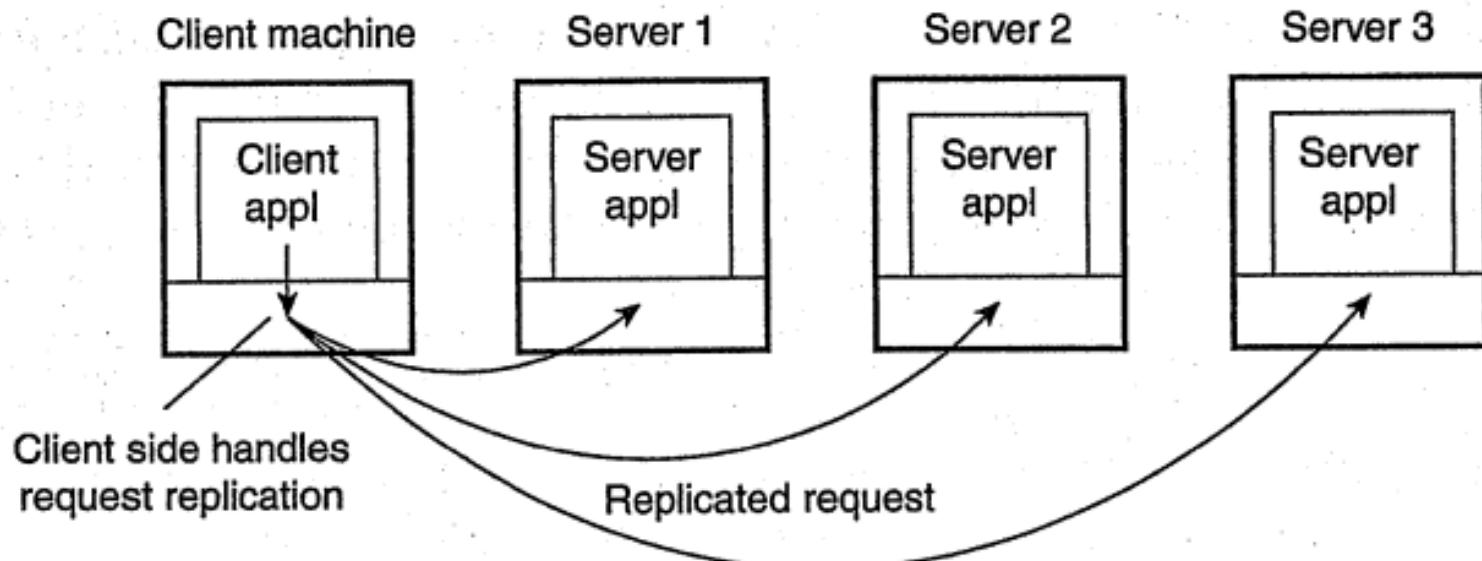


Client-side software for distribution transparency

22

- ❖ Transparent distribution:

- ❖ Transparent access
- ❖ Transparent migration
- ❖ Transparent replication
- ❖ Transparent faults



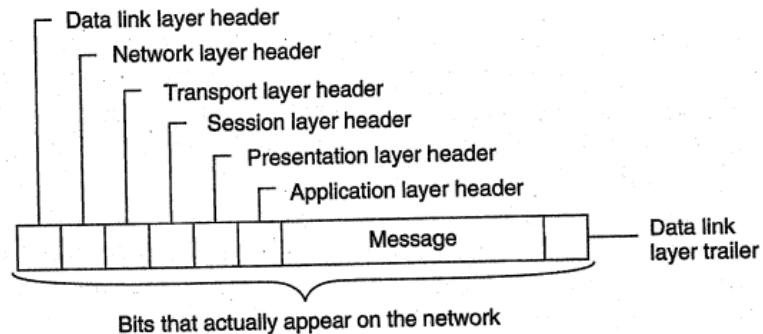
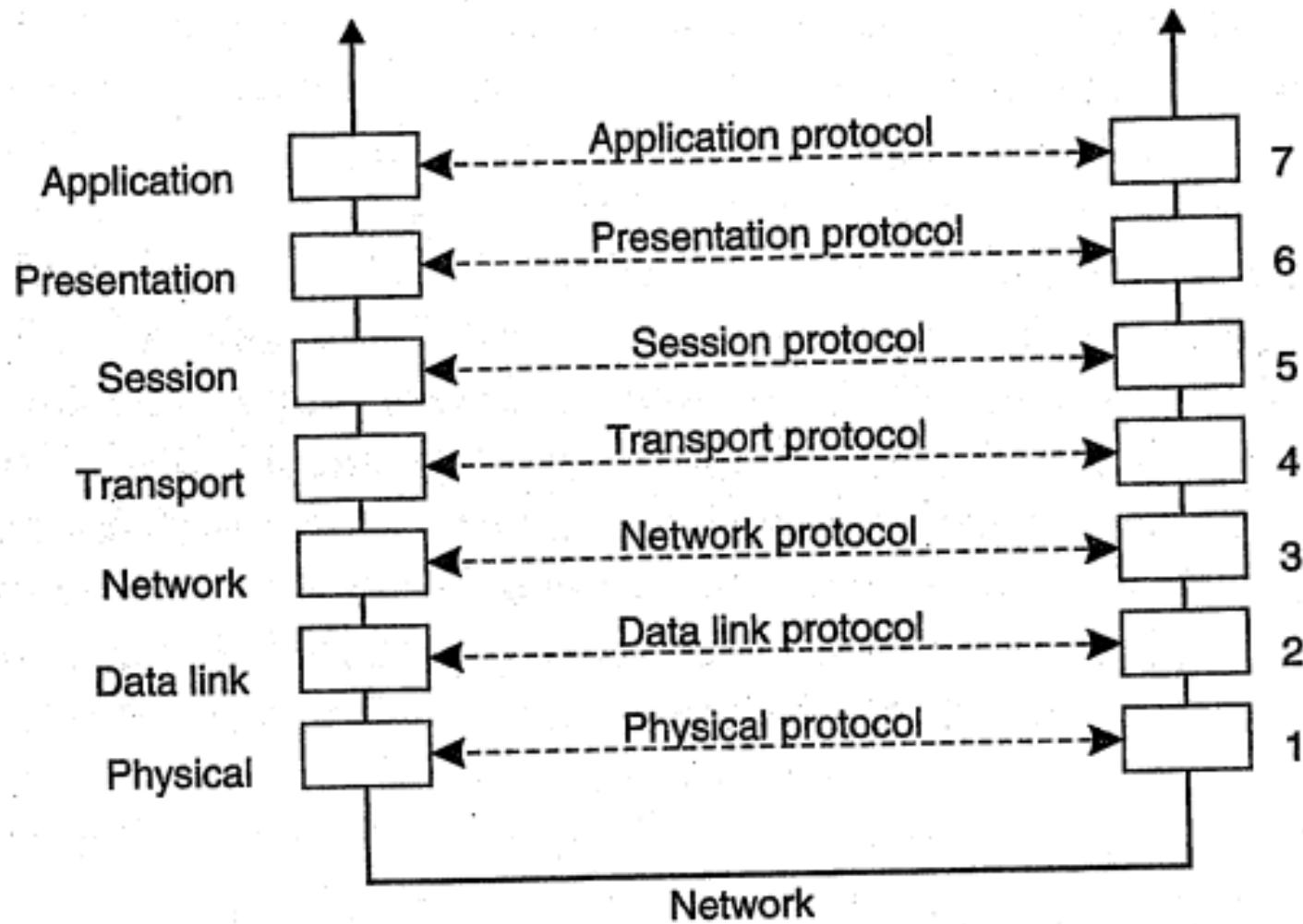
23

2. Communication

2.1. Definition

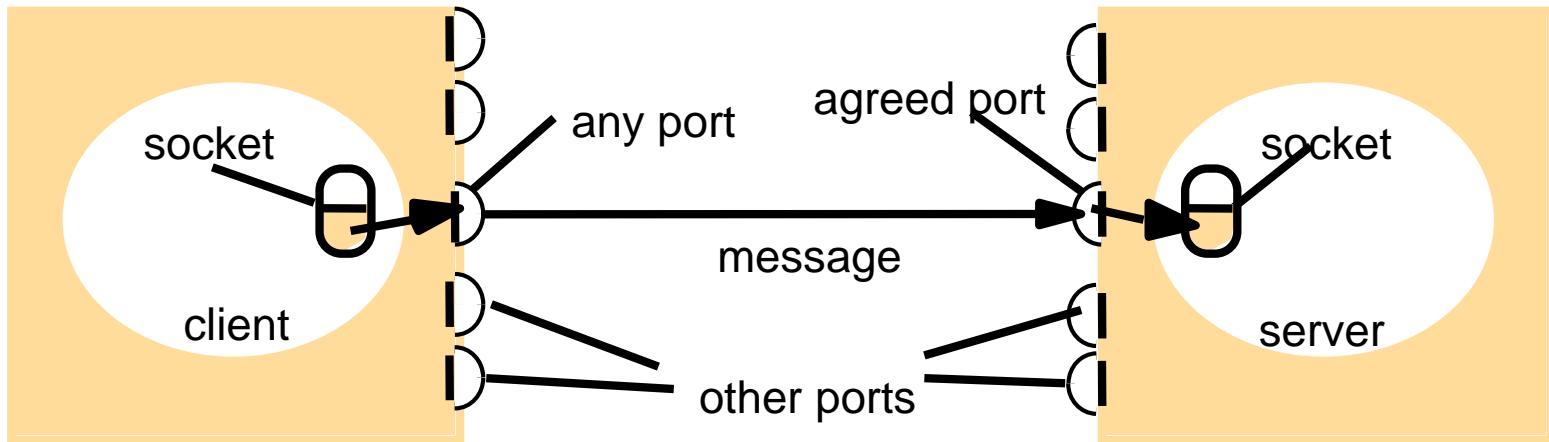
24

- Agreements are needed at a variety of levels, varying from the low-level details of bit transmission to the high-level details of how information is to be expressed.
- Protocol
 - Message format
 - Message size
 - Message order
 - Faults detection method
 - Etc.
- Layered
- Protocol types:
 - Connection oriented/connectionless protocols, Reliable/Unreliable protocols
- Protocol issues:
 - Send, receive primitives
 - Synchronous, Asynchronous, Blocking or non-blocking



Socket-port

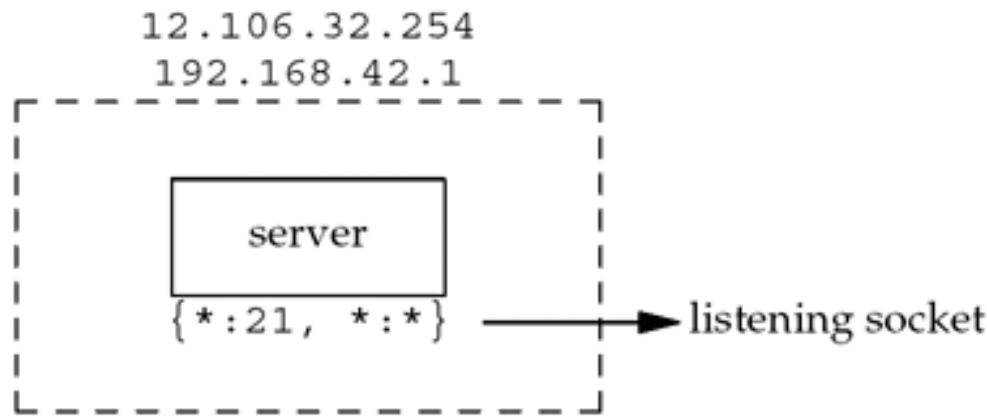
26



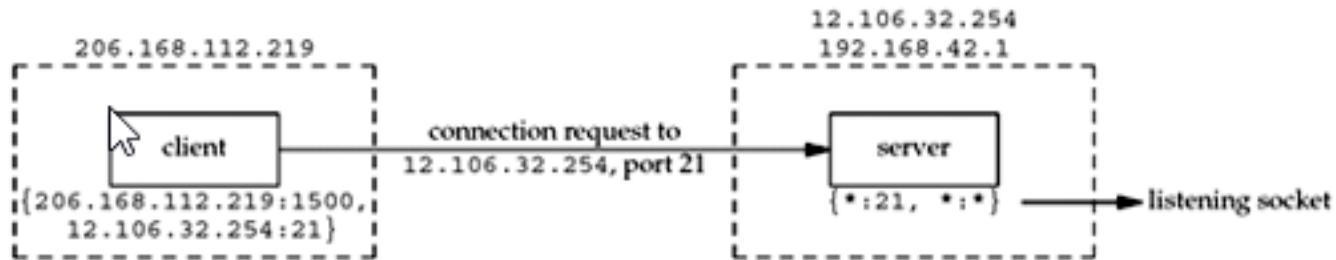
Internet address = 138.37.94.248

Internet address = 138.37.88.249

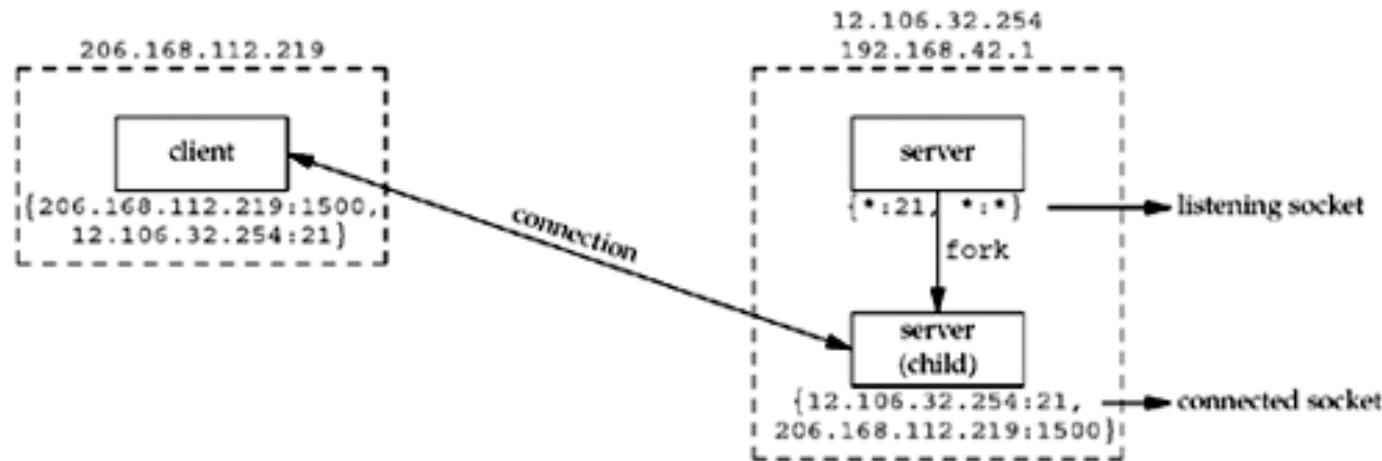
TCP Port Numbers and Concurrent Servers (1)



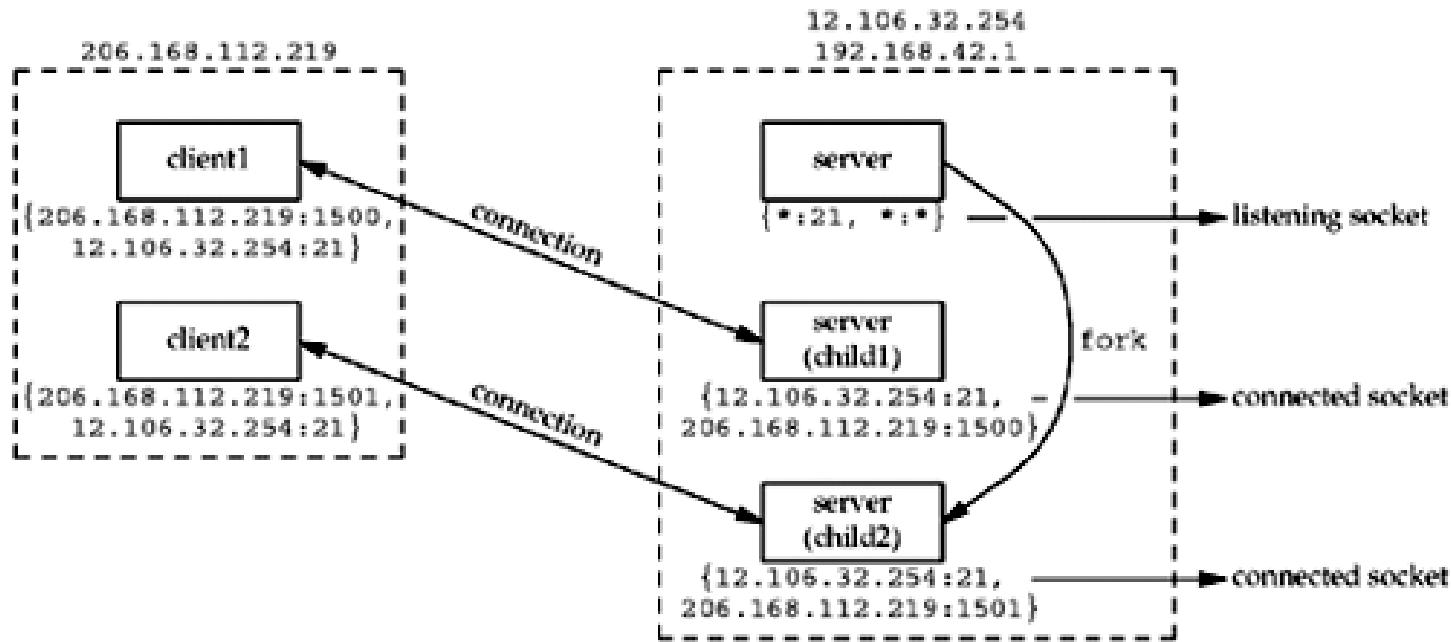
TCP Port Numbers and Concurrent Servers (2)



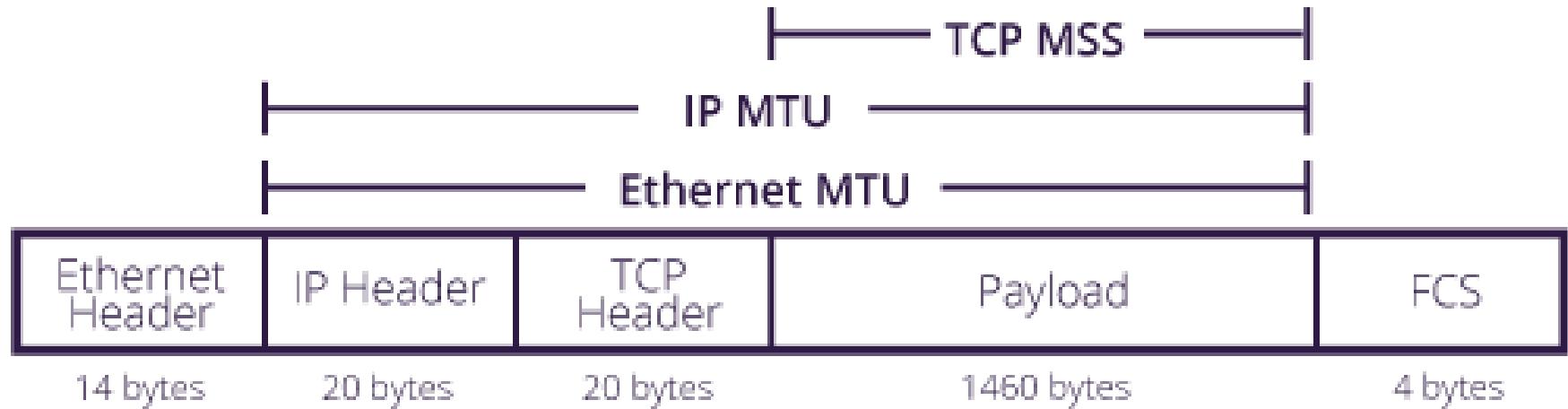
TCP Port Numbers and Concurrent Servers (3)



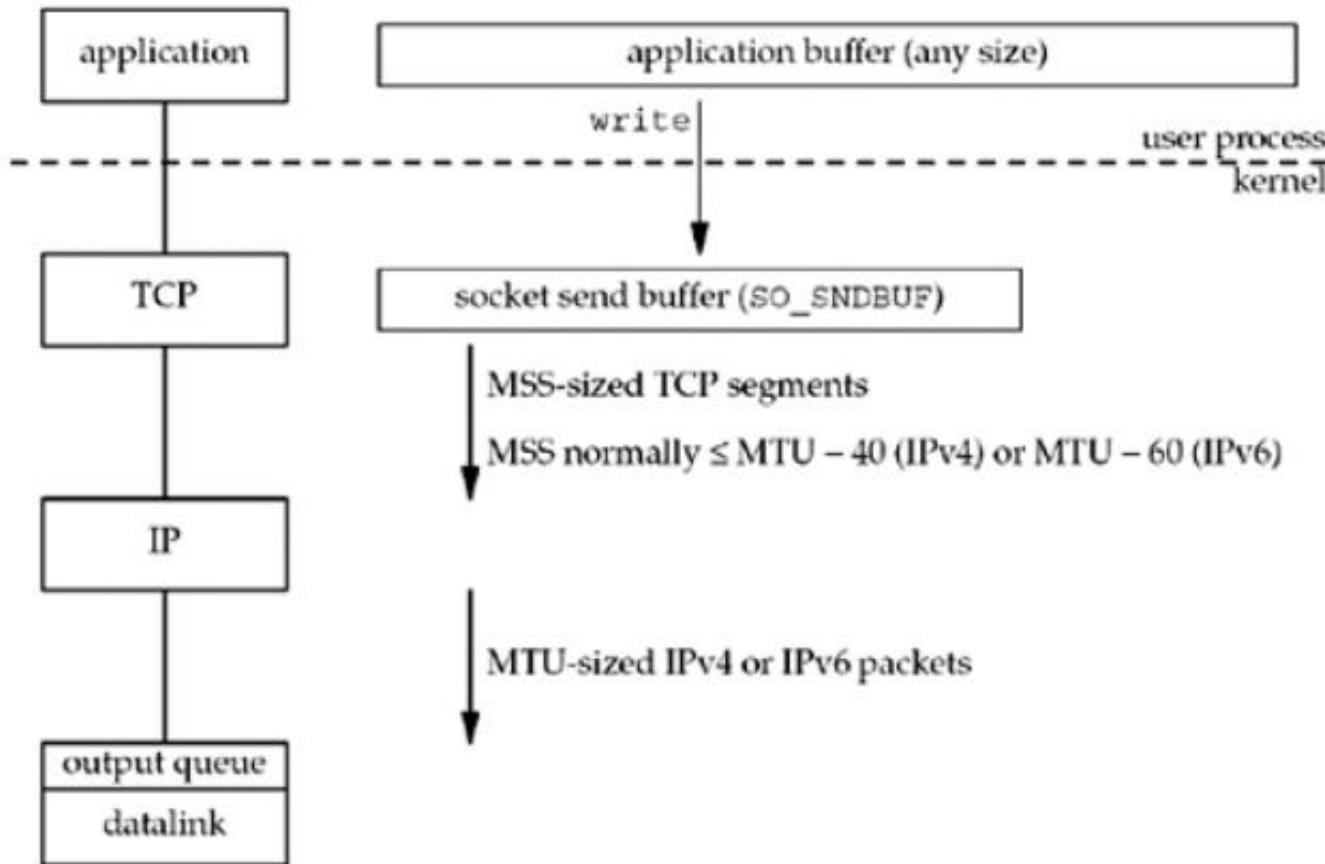
TCP Port Numbers and Concurrent Servers (4)



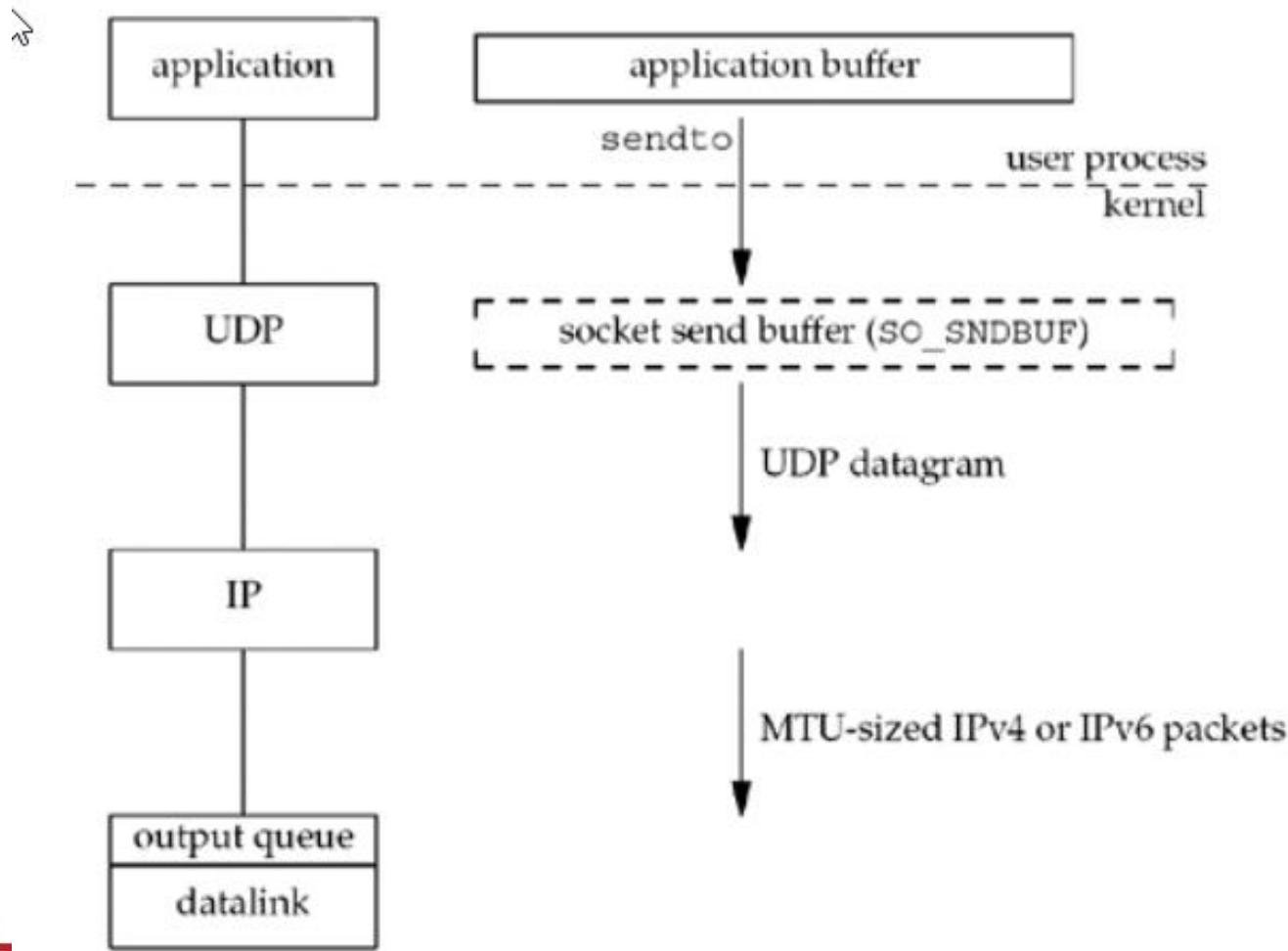
Buffer Sizes and Limitations



TCP output



UDP output



Communication with UDP and TCP

34

- UDP
 - connectionless protocol
 - No handshake protocol
 - Unreliable protocol
 - Asynchronous protocol
- TCP
 - connection-oriented protocol
 - With handshake protocol (SYN, SYN-ACK, ACK)
 - Reliable protocol
 - error recovery
 - acknowledgment segments
 - Synchronous protocol

3. Remote Procedure Call

3.1. Request-reply protocol

3.2. RPC

3.3. RMI

3.1. Request-reply protocol

36

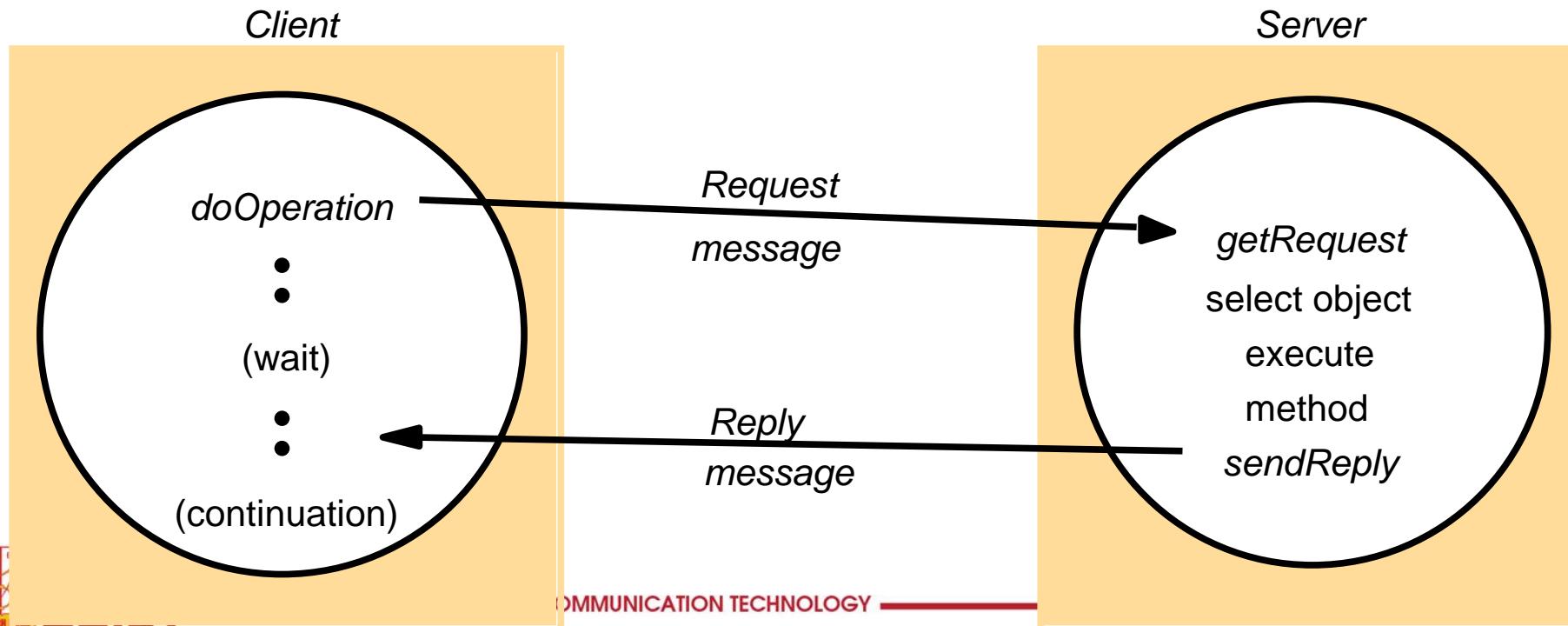
- a pattern on top of message passing
- support the two-way exchange of messages as encountered in client-server computing
- synchronous
- reliable

Request-reply protocol

37

□ Characteristics:

- No need of Acknowledgement
- No need of Flow control



Message structure

38

messageType

int (0=Request, 1= Reply)

requestId

int

remoteReference

RemoteRef

operationId

int or Operation

arguments

array of bytes

Example: HTTP

HTTP *request* message

<i>method</i>	<i>URL or pathname</i>	<i>HTTP version</i>	<i>headers</i>	<i>message body</i>
GET	//www.dcs.qmw.ac.uk/index.htm	HTTP/ 1.1		

HTTP *reply* message

<i>HTTP version</i>	<i>status code</i>	<i>reason</i>	<i>headers</i>	<i>message body</i>
HTTP/1.1	200	OK		resource data

3.2. RPC (Remote Procedure Call)

40

content
of this
section

Applications, services

Remote invocation, indirect communication

Underlying interprocess communication primitives:
Sockets, message passing, multicast support, overlay networks

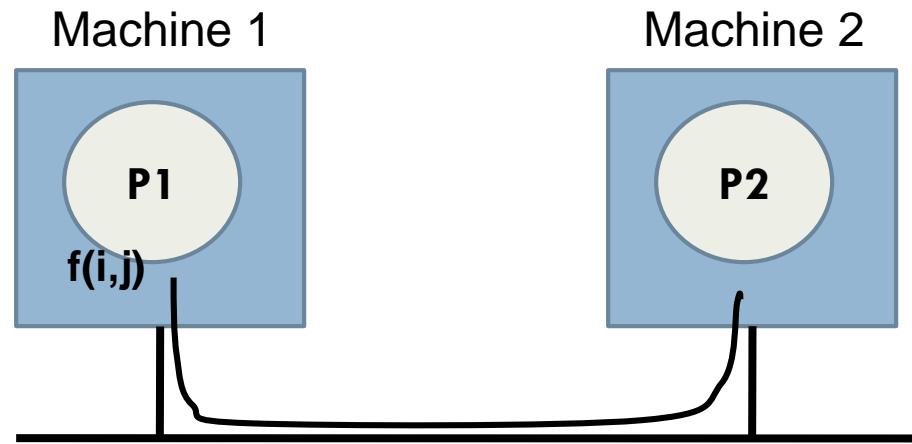
UDP and TCP

Middleware

Remote Procedure Call

41

- Access transparency
- Issues:
 - Heterogenous system
 - Different memory space
 - Different information representation
 - Faults appear



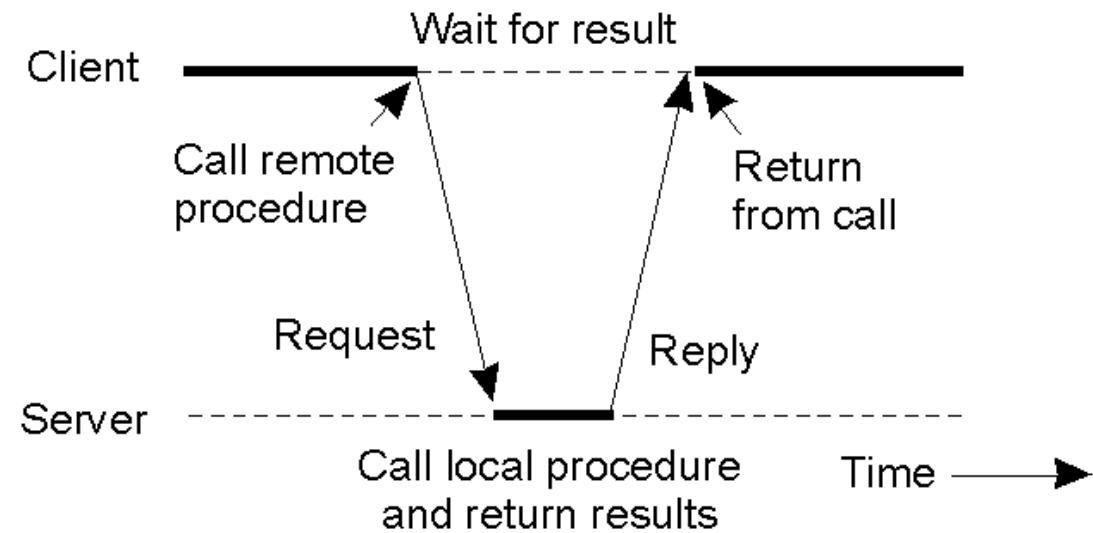
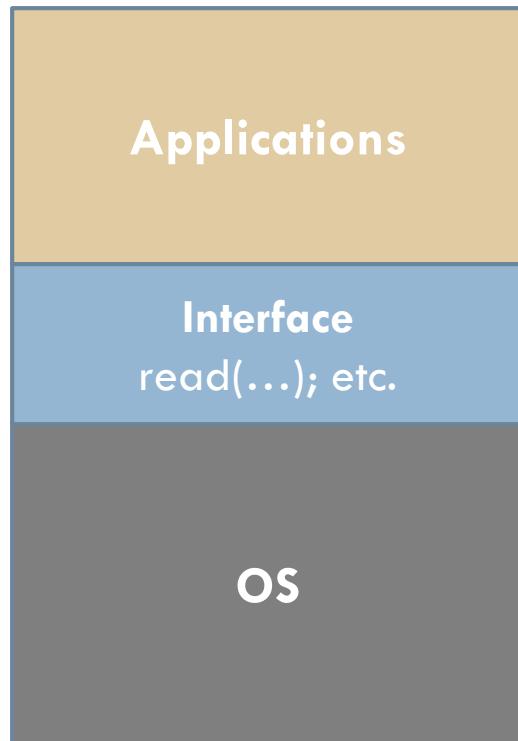
Parameters

42

- Call-by-value
- Call-by-reference
- Call-by-copy/restore
 - ▣ Copy the variables to the stack
 - ▣ Copy back after the call, overwrite caller's the original value

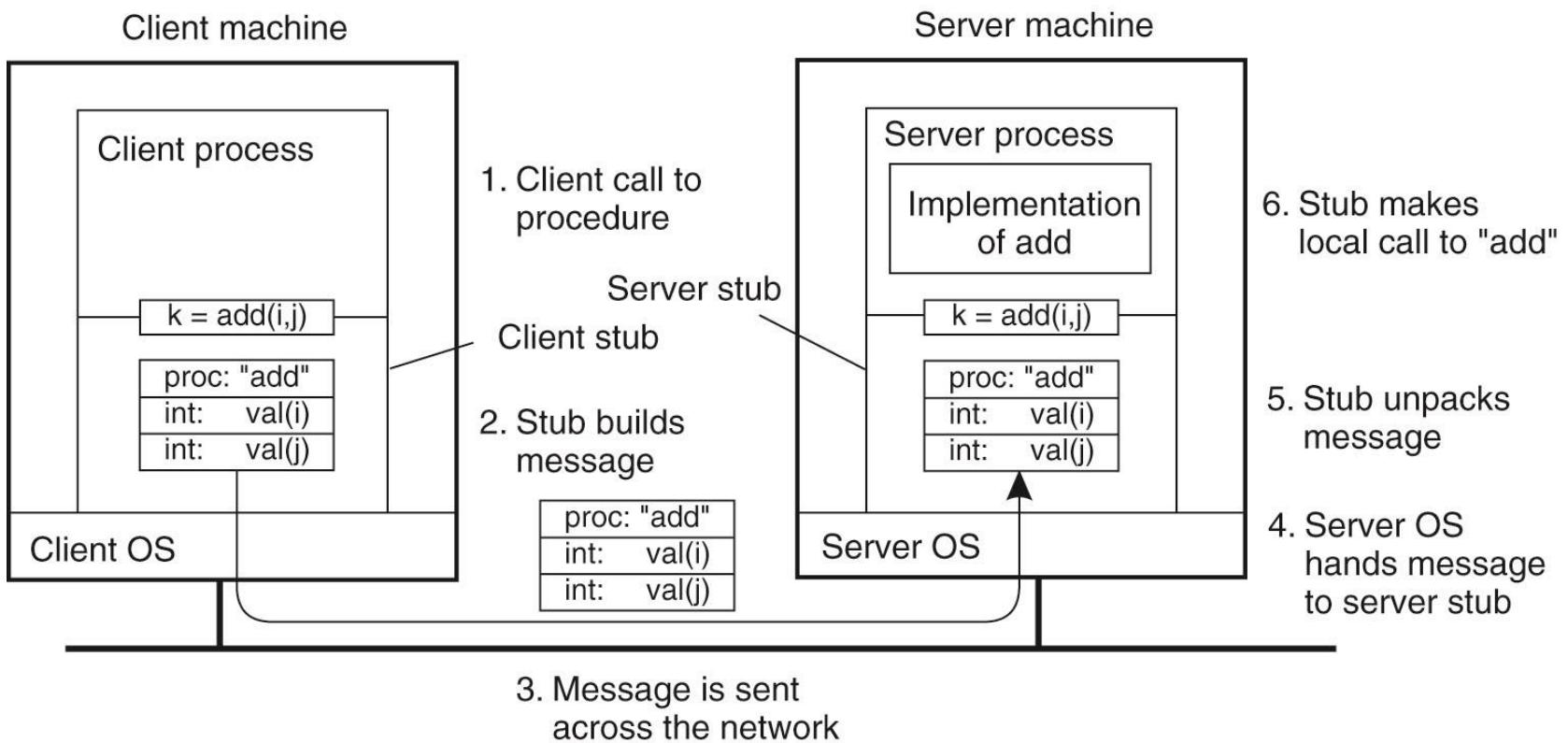
RPC mechanism

43



RPC mechanism

44



Passing Value Parameters

45

- Work well when the end-systems are uniform
- Problems:
 - Different representation for numbers, characters, and other data items

Issue: Different character format

46

Intel Pentium (little endian)

0	3	2	1	0
L	L	I	J	
0	5	6	5	4
L	L	I	J	

(a)

SPARC (big endian)

0	1	2	3
5	0	0	0
4	5	6	7
J	I	L	L

(b)

0	1	2	3
0	0	0	5
4	5	6	7
L	L	I	J

(c)

Passing Reference Parameters

47

- Issue: a pointer is meaningful only within the address space of the process in which it is being used.
- Solutions:
 - Forbid pointers and reference parameters → undesirable
 - Copy/Restore
 - Issue: costly (bandwidth, store copies)
- Unfeasible for structured data

Parameter specification

48

- The caller and the callee agree on the format of the messages they exchange.
- Agreements:
 - Message format
 - Representation of simple data structures (integers, characters, Booleans, etc.)
 - Method for exchanging messages.
 - Client-stub and server-stub need to be implemented.

```
foobar( char x; float y; int z[5] )  
{  
    ....  
}
```

(a)

foobar's local variables	
x	
y	
5	
z[0]	
z[1]	
z[2]	
z[3]	
z[4]	

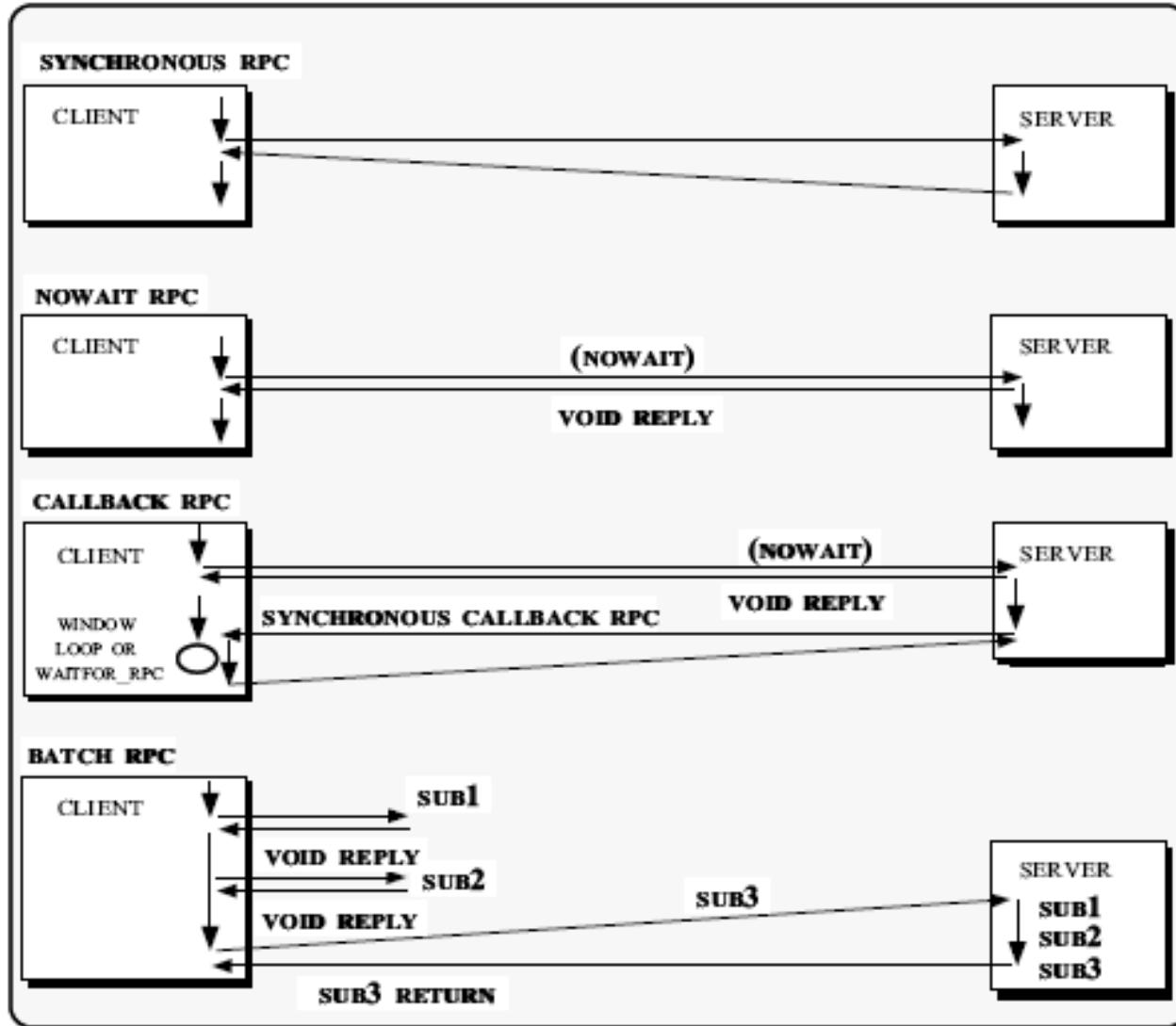
(b)

Openness of RPC

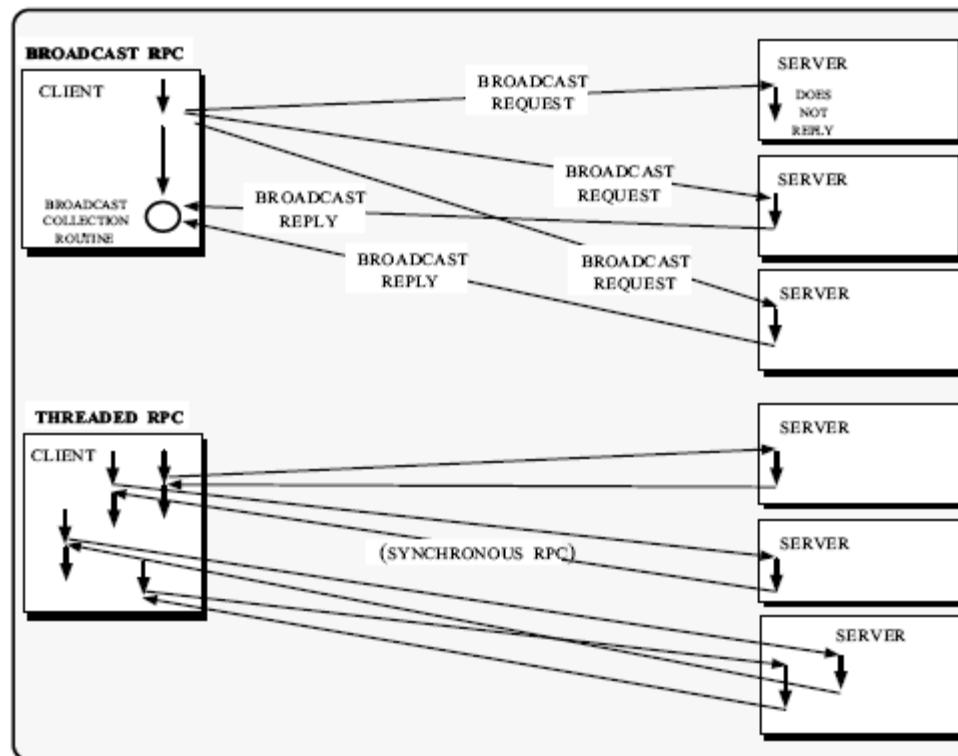
49

- Client and Server are installed by different providers.
- Common interface between client and server
 - Programming language independence
 - Full description and neutral
 - Using IDL

RPC models

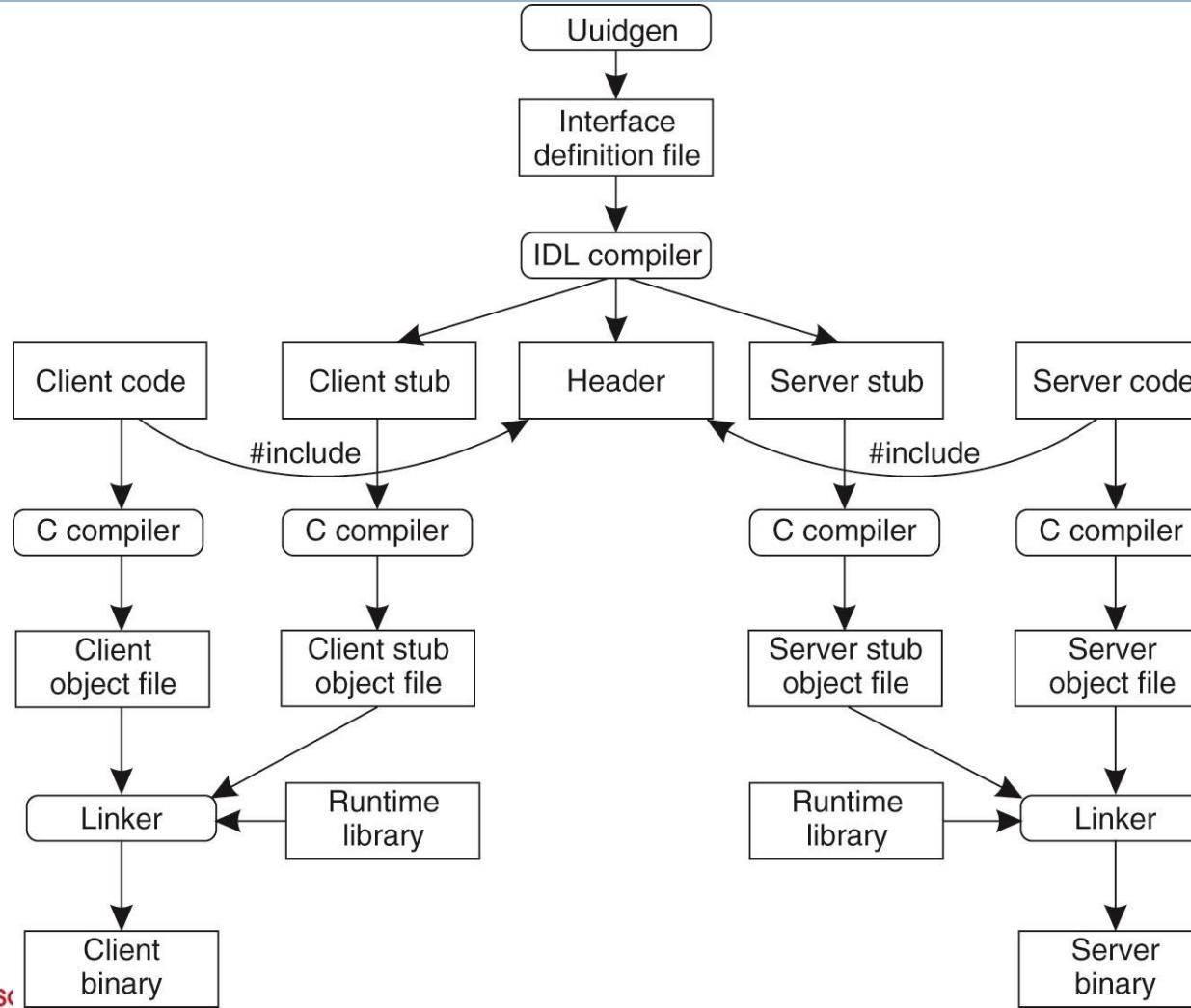


RPC models



Implementing RPC

52



Mini tutorial: RPC on Windows

53

- Link: <https://docs.microsoft.com/en-us/windows/win32/rpc/tutorial>
- Create interface definition and application configuration files.
- Use the MIDL compiler to generate C-language client and server stubs and headers from those files.
- Write a client application that manages its connection to the server.
- Write a server application that contains the actual remote procedures.
- Compile and link these files to the RPC run-time library to produce the distributed application.

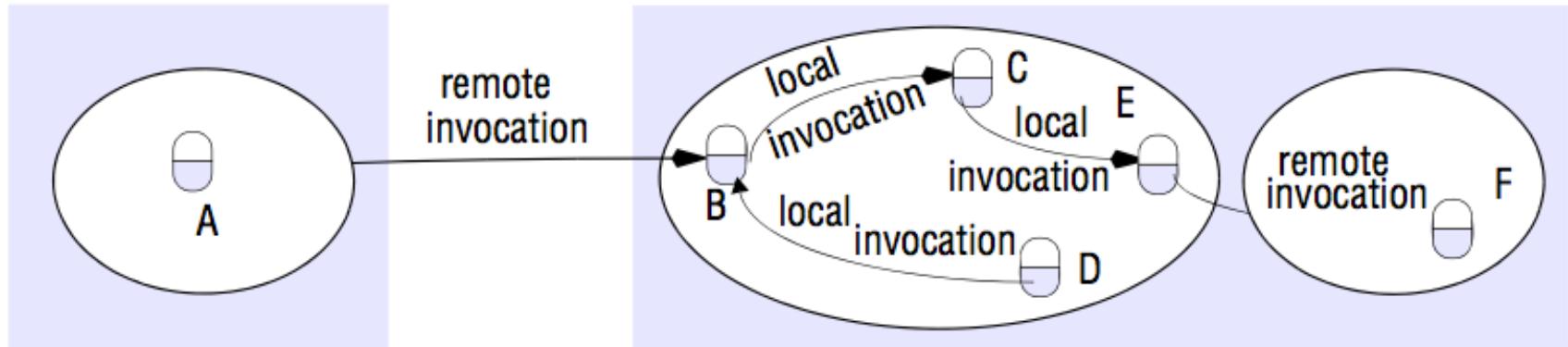
2.4. RMI (Remote Method Invocation)

54

- RMI vs. RPC
 - Common points:
 - Support programming with interface
 - Based on request-reply protocol
 - Transparency
 - Different point:
 - Benefits of OOP

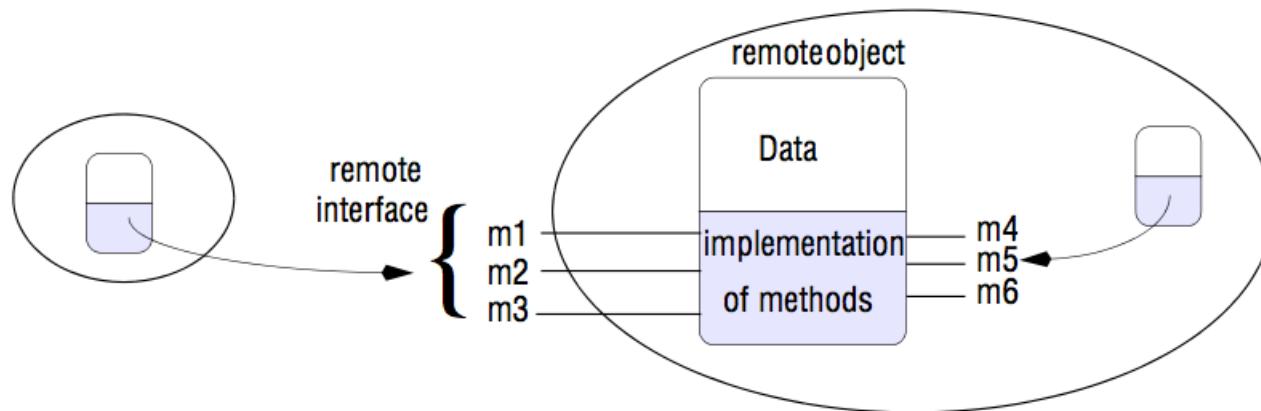
Distributed objects model

55



Remote object and Remote interface

56



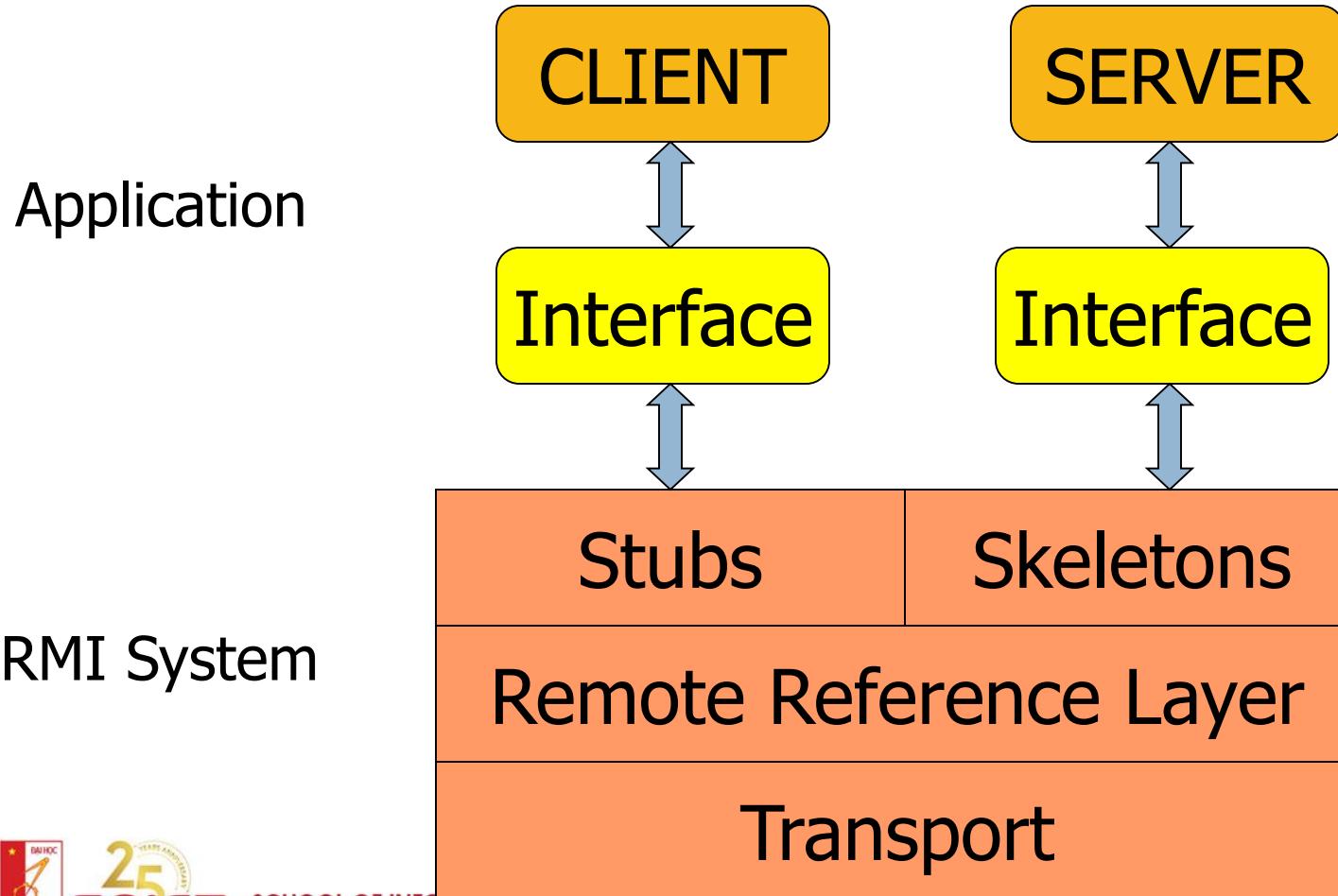
Characteristics

57

- Benefits
 - Simplicity
 - Transparency
 - Reliability
 - Security (supported by Java)
- Drawbacks:
 - Only support java

RMI Architecture

58



Mini tutorial:

59

- Link:
<https://docs.oracle.com/javase/tutorial/rmi/index.html>
- Overview of RMI
- Writing RMI server
- Creating RMI client
- Compiling & Running

4. Message-oriented communication

4.1. Message-oriented transient communication

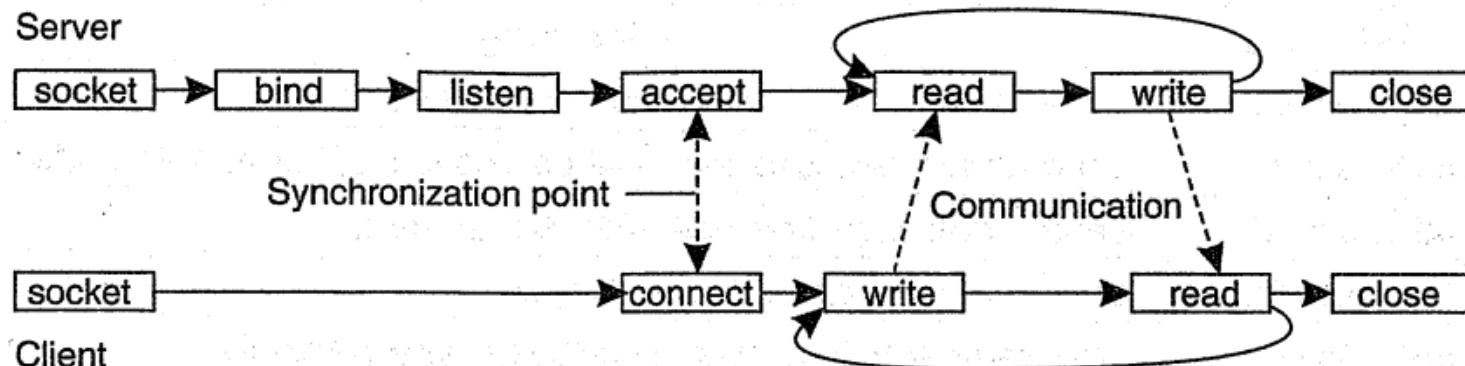
4.2. Message-oriented persistent communication

4.1. Message-oriented transient communication

61

□ Berkeley Sockets

Primitive	Meaning
Socket	Create a new communication end point
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection



socket function

- To perform network I/O, the first thing a process must do is call the *socket* function

```
#include <sys/socket.h>  
int socket (int family, int type, int protocol);
```

- Returns: non-negative descriptor if OK, -1 on error

family	Description
AF_INET	IPv4 protocols
AF_INET6	IPv6 protocols
AF_LOCAL	Unix domain protocols (Chapter 15)
AF_ROUTE	Routing sockets (Chapter 18)
AF_KEY	Key socket (Chapter 19)

family

type	Description
SOCK_STREAM	stream socket
SOCK_DGRAM	datagram socket
SOCK_SEQPACKET	sequenced packet socket
SOCK_RAW	raw socket

socket

Protocol	Description
IPPROTO_TCP	TCP transport protocol
IPPROTO_UDP	UDP transport protocol
IPPROTO_SCTP	SCTP transport protocol

protocol

connect Function

- The *connect* function is used by a TCP client to establish a connection with a TCP server.

```
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *servaddr,
            socklen_t addrlen);
```

- Returns: 0 if OK, -1 on error
- *sockfd* is a socket descriptor returned by the *socket* function
- The second and third arguments are a pointer to a socket address structure and its size.
- The client does not have to call *bind* before calling *connect*: the kernel will choose both an ephemeral port and the source IP address if necessary.

connect Function (2)

- Problems with *connect* function:
 1. If the client TCP receives no response to its SYN segment, ETIMEDOUT is returned. (If no response is received after a total of 75 seconds, the error is returned).
 2. If the server's response to the client's SYN is a reset (RST), this indicates that no process is waiting for connections on the server host at the port specified (i.e., the server process is probably not running). Error: ECONNREFUSED.
 3. If the client's SYN elicits an ICMP "destination unreachable" from some intermediate router, this is considered a soft error. If no response is received after some fixed amount of time (75 seconds for 4.4BSD), the saved ICMP error is returned to the process as either EHOSTUNREACH or ENETUNREACH.

bind Function

- The bind function assigns a local protocol address to a socket.

```
#include <sys/socket.h>
int bind (int sockfd, const struct sockaddr *myaddr,
          socklen_t addrlen);
```

- Returns: 0 if OK,-1 on error
- Example:

```
struct sockaddr_in address;
/* type of socket created in socket() */
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
/* 7000 is the port to use for connections */
address.sin_port = htons(7000);
/* bind the socket to the port specified above */
```

listen Function

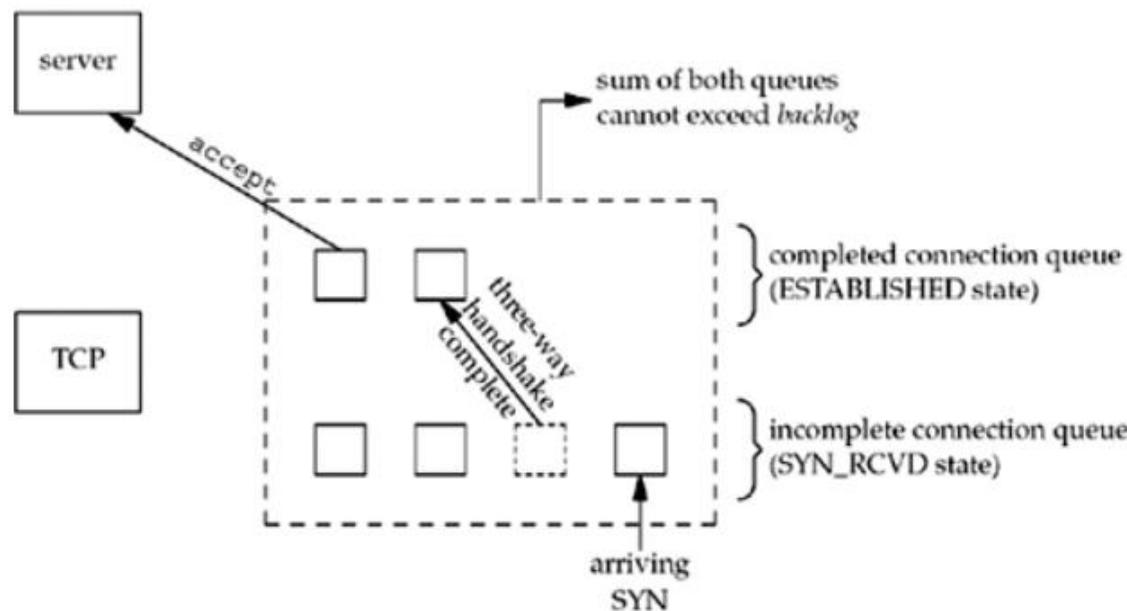
- The *listen* function is called only by a TCP server.
- When a socket is created by the *socket* function, it is assumed to be an active socket, that is, a client socket that will issue a *connect*.
- The *listen* function converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to this socket.
- Move the socket from the CLOSED state to the LISTEN state.

```
#include <sys/socket.h>
int listen (int sockfd, int backlog);
```

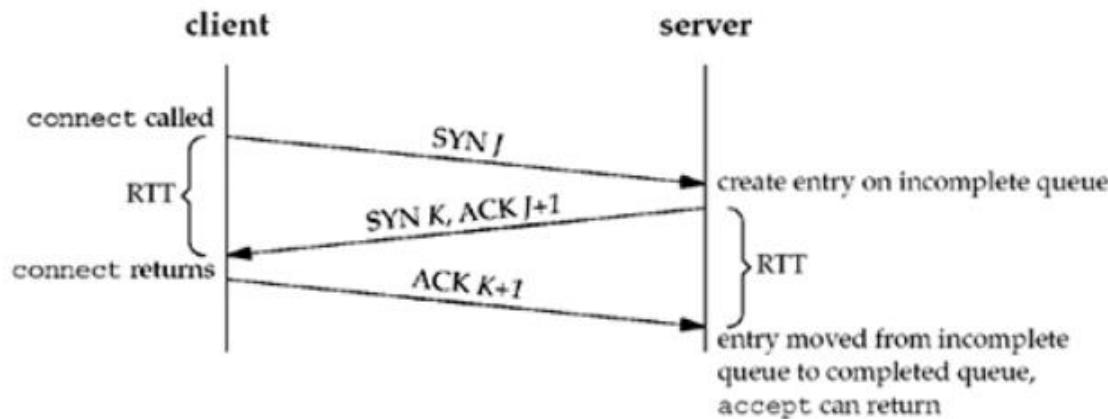
- Returns: 0 if OK, -1 on error

listen Function (2)

- The second argument (*backlog*) to this function specifies the maximum number of connections the kernel should queue for this socket.



listen Function (3)



TCP three-way handshake and the two queues for a listening socket.

accept Function

- *accept* is called by a TCP server to return the next completed connection from the front of the completed connection queue.
- If the completed connection queue is empty, the process is put to sleep.

```
#include <sys/socket.h>
int accept (int sockfd, struct sockaddr *cliaddr, socklen_t
            *addrlen);
```

- Returns: non-negative descriptor if OK, -1 on error
- The *cliaddr* and *addrlen* arguments are used to return the protocol address of the connected peer process (the client).
- *addrlen* is a value-result argument

accept Function

□ Example

```
int addrlen;
struct sockaddr_in address;

addrlen = sizeof(struct sockaddr_in);
new_socket = accept(socket_desc, (struct sockaddr *)&address, &addrlen);
if (new_socket<0)
    perror("Accept connection");
```

fork and *exec* Functions

```
#include <unistd.h>
pid_t fork(void);
```

- Returns: 0 in child, process ID of child in parent, -1 on error
- *fork* function (including the variants of it provided by some systems) is the only way in Unix to create a new process.
- It is called once but it returns twice.
- It returns once in the calling process (called the parent) with a return value that is the process ID of the newly created process (the child). It also returns once in the child, with a return value of 0.
- The reason *fork* returns 0 in the child, instead of the parent's process ID, is because a child has only one parent and it can always obtain the parent's process ID by calling *getppid*.

Example

```
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    printf("--beginning of program\n");

    int counter = 0;
    pid_t pid = fork();

    if (pid == 0)
    {
        // child process
        int i = 0;
        for (; i < 5; ++i)
        {
            printf("child process: counter=%d\n", ++counter);
        }
    }
    else if (pid > 0)
    {
        // parent process
        int j = 0;
        for (; j < 5; ++j)
        {
            printf("parent process: counter=%d\n", ++counter);
        }
    }
    else
    {
        // fork failed
        printf("fork() failed!\n");
        return 1;
    }

    printf("--end of program--\n");

    return 0;
}
```

Concurrent Servers

- fork a child process to handle each client

```
pid_t pid;
int listenfd, connfd;

listenfd = Socket( ... );

/* fill in sockaddr_in{} with server's well-known port */
Bind(listenfd, ... );
Listen(listenfd, LISTENQ);

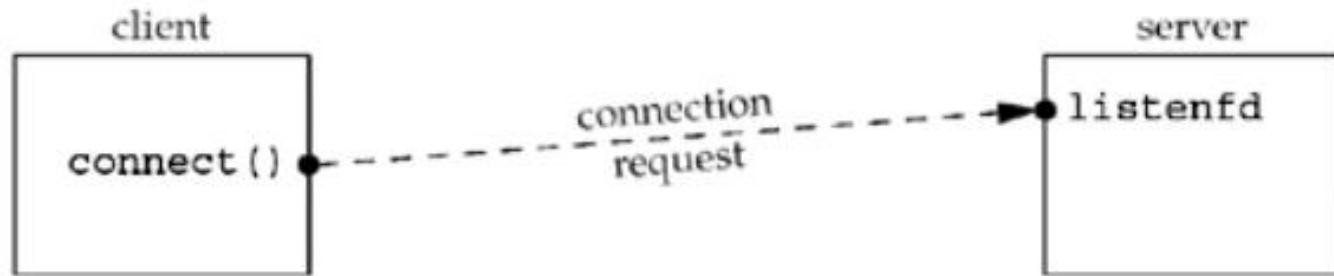
for ( ; ; ) {
    connfd = Accept (listenfd, ... );      /* probably blocks */

    if( (pid = Fork()) == 0) {
        Close(listenfd);      /* child closes listening socket */
        doit(connfd);        /* process the request */
        Close(connfd);       /* done with this client */
        exit(0);              /* child terminates */
    }

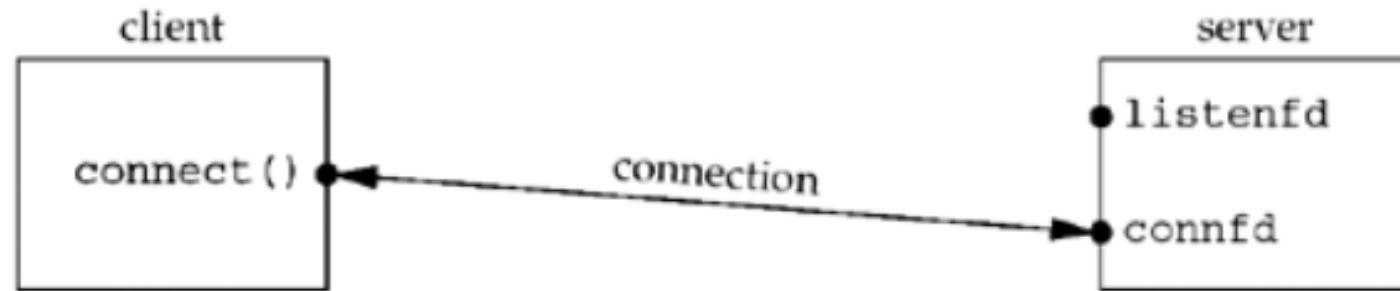
    Close(connfd);          /* parent closes connected socket */ -
```



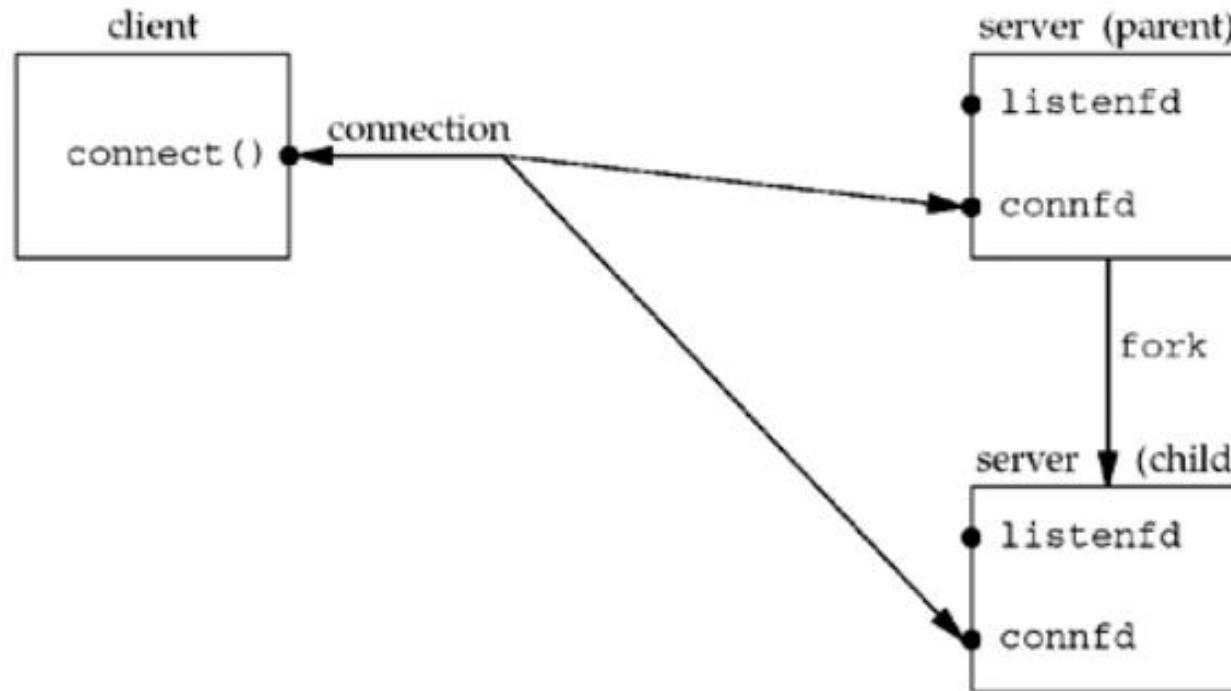
Status of client/server before call to *accept* returns.



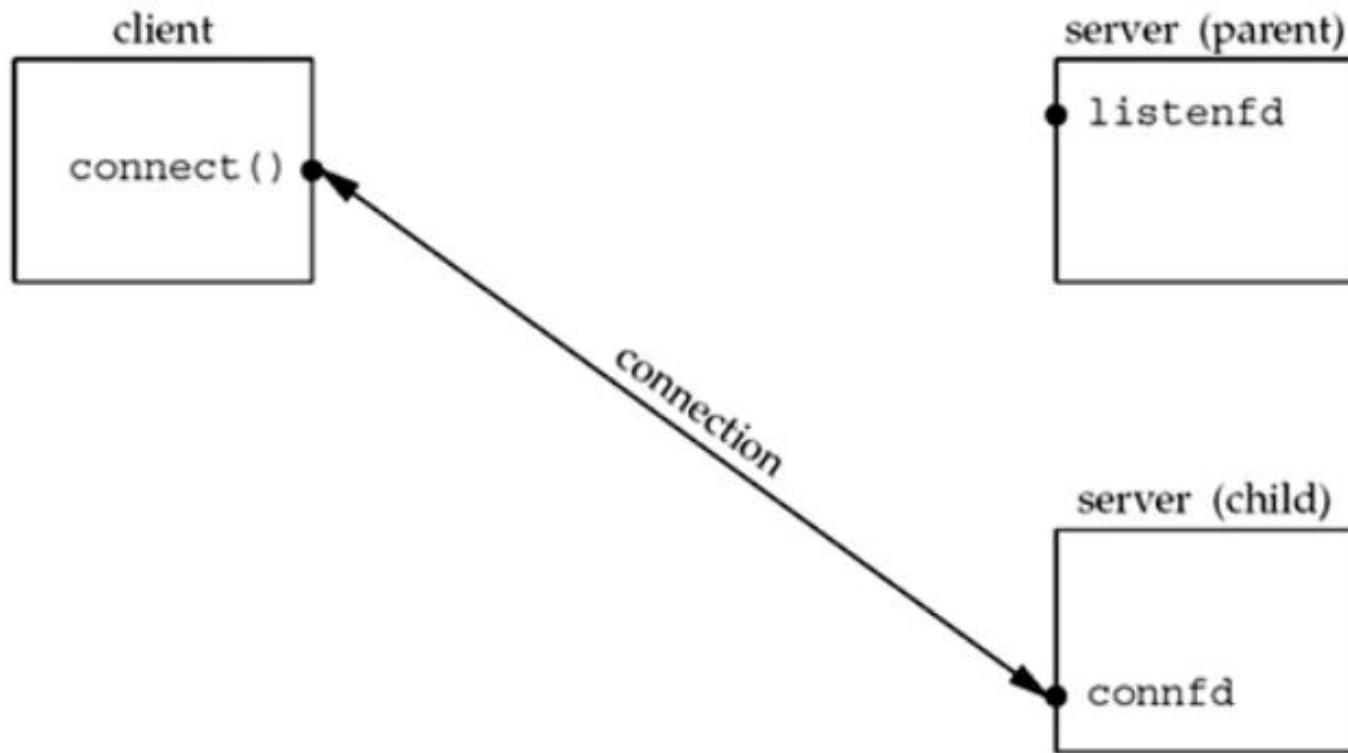
Status of client/server after return from *accept*.



Status of client/server after fork returns.



Status of client/server after parent and child close appropriate sockets.



close Function

- The normal Unix close function is also used to close a socket and terminate a TCP connection.

```
#include <unistd.h>
int close (int sockfd);
```

- Returns: 0 if OK, -1 on error
- If the parent doesn't close the socket, when the child closes the connected socket, its reference count will go from 2 to 1 and it will remain at 1 since the parent never closes the connected socket. This will prevent TCP's connection termination sequence from occurring, and the connection will remain open.

Message-Passing Interface

79

Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and wait until receipt starts
MPI_recv	Receive a message; block if there is none
MPI_irecv	Check if there is an incoming message, but do not block

The socket primitives for TCP/IP



SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

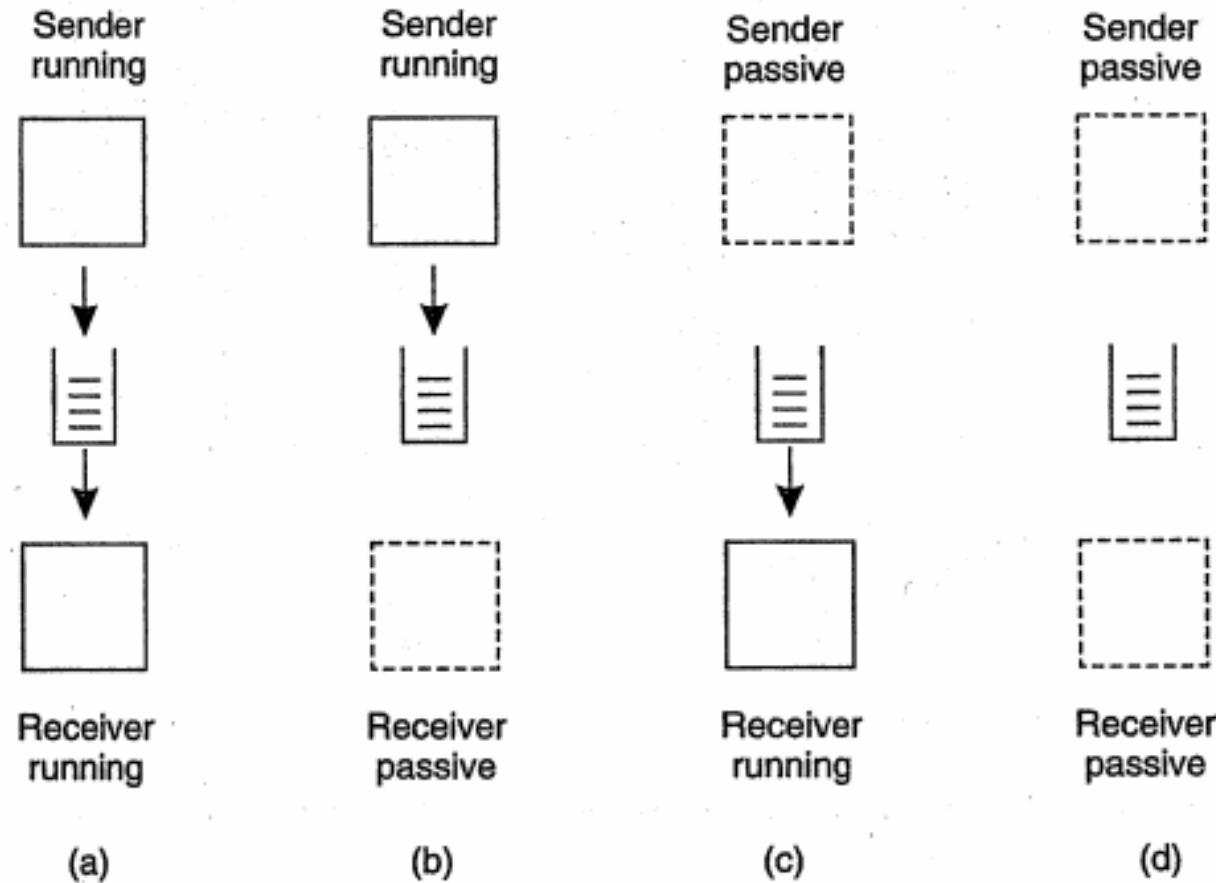
4.2. Message-Oriented Persistent Communication

80

- Very important class of message-oriented middleware services: Message-Queuing Systems, or MOM (Message-Oriented Middleware).
- Message-Queuing Systems provide extensive support for persistent asynchronous communication.
- Offer intermediate-term storage capacity for messages
- Latency tolerance
- Example: Email system

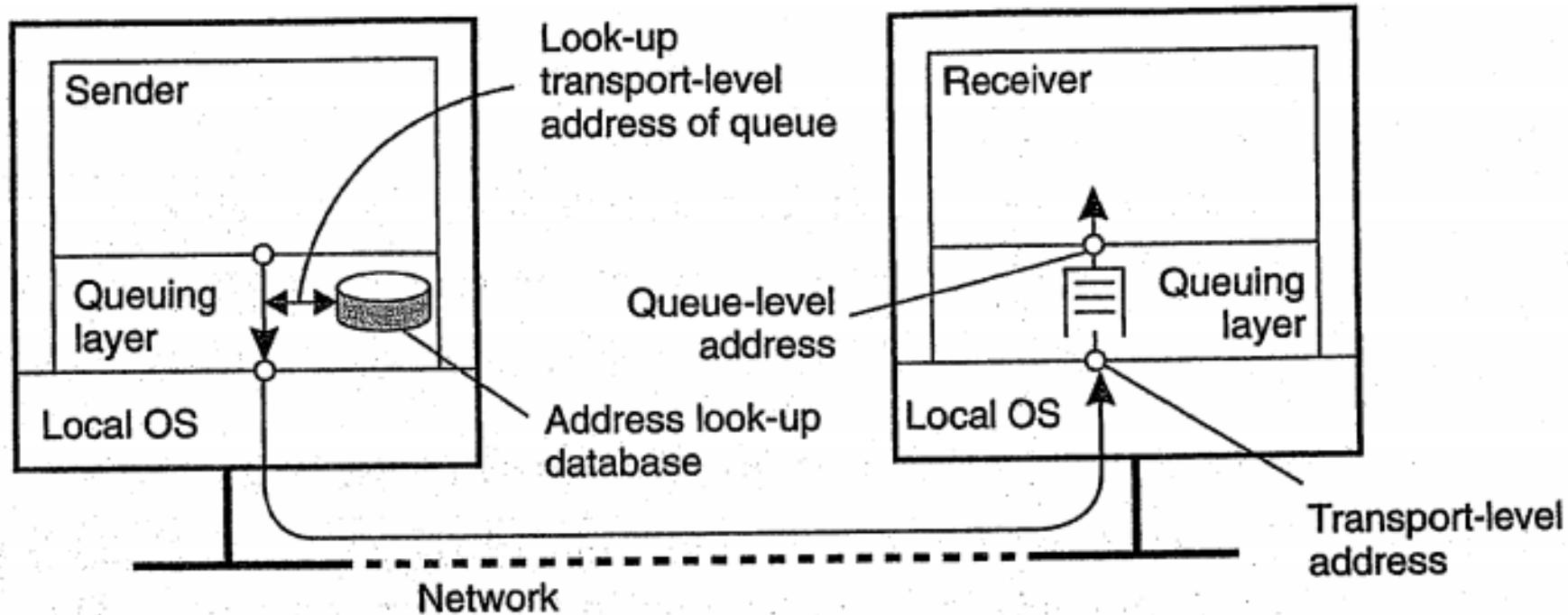
Message-Queuing System

81



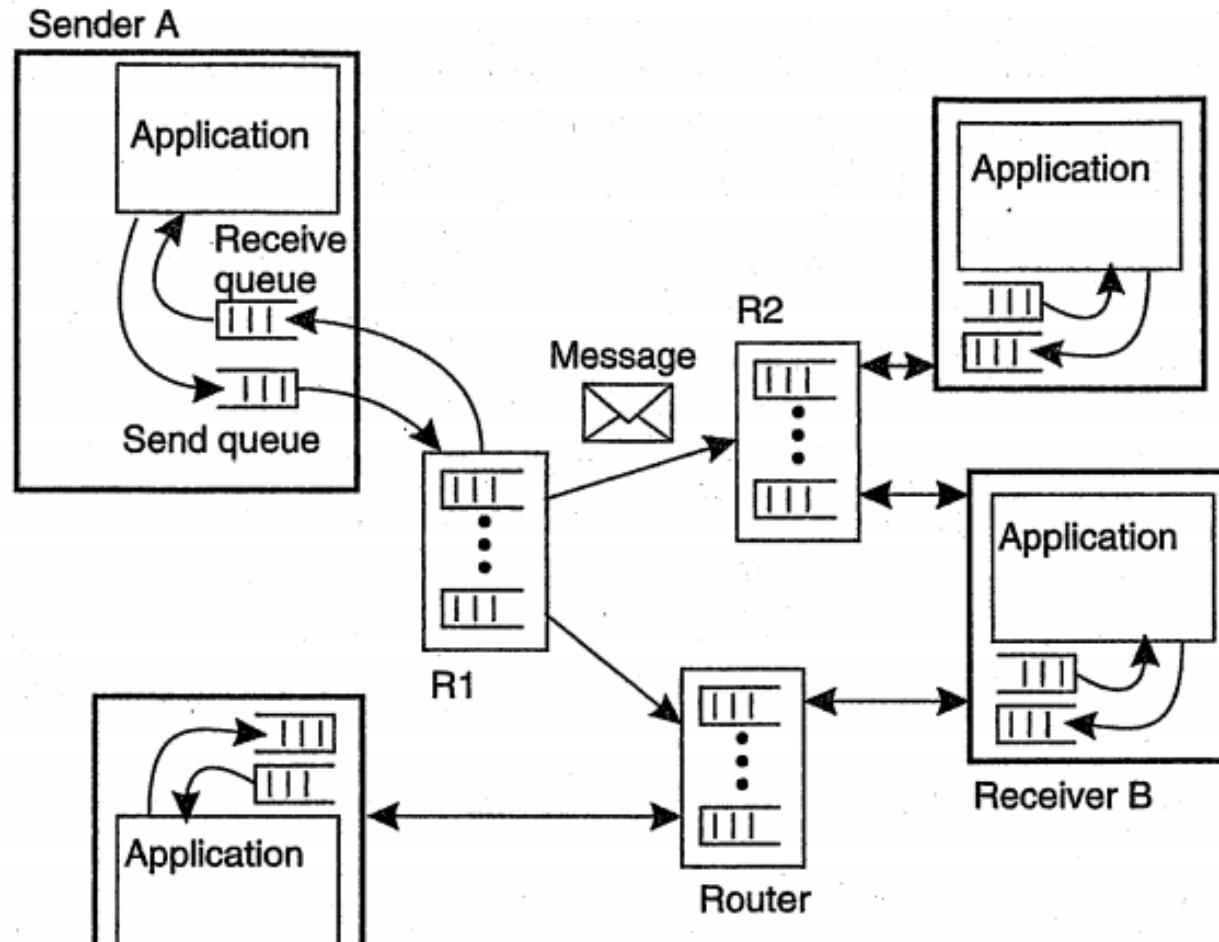
The relationship between queue-level addressing and network-level addressing

82



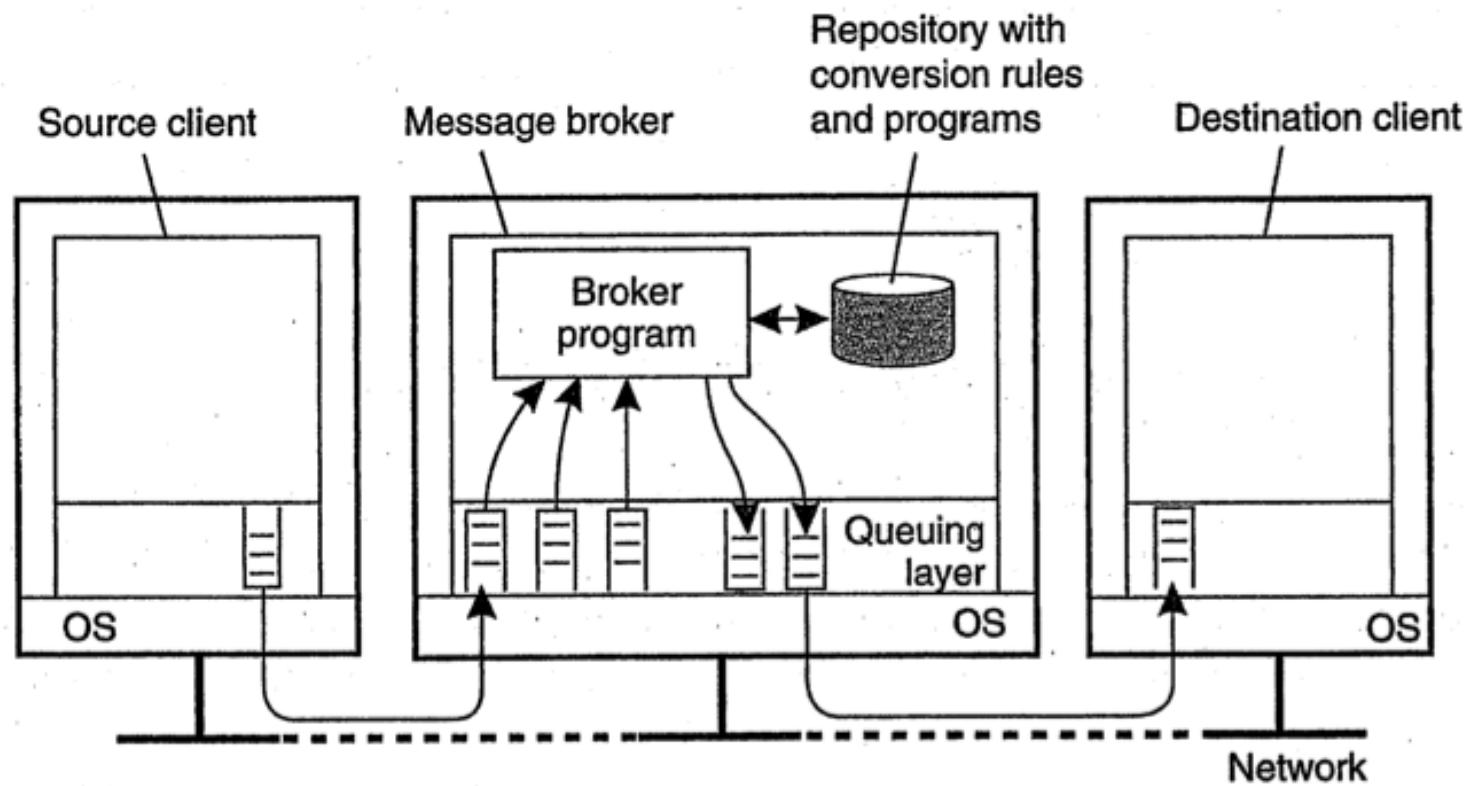
Routing with Queueing system

83



Message Broker

84



RabbitMQ

85

1 "Hello World!"

The simplest thing that does something



- > Python
- > Java
- > Ruby
- > PHP
- > C#
- > Javascript
- > Go
- > Elixir
- > Objective-C

2 Work queues

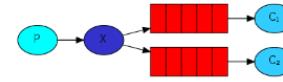
Distributing tasks among workers



- > Python
- > Java
- > Ruby
- > PHP
- > C#
- > Javascript
- > Go
- > Elixir
- > Objective-C

3 Publish/Subscribe

Sending messages to many consumers at once



- > Python
- > Java
- > Ruby
- > PHP
- > C#
- > Javascript
- > Go
- > Elixir
- > Objective-C

4 Routing

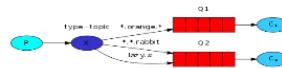
Receiving messages selectively



- > Python
- > Java
- > Ruby
- > PHP
- > C#
- > Javascript
- > Go
- > Elixir
- > Objective-C

5 Topics

Receiving messages based on a pattern



- > Python
- > Java
- > Ruby
- > PHP
- > C#
- > Javascript
- > Go
- > Elixir
- > Objective-C

6 RPC

Remote procedure call implementation



- > Python
- > Java
- > Ruby
- > PHP
- > C#
- > Javascript
- > Go
- > Elixir

5. Stream-oriented Communication

5.1. Support for Continuous Media

5.2. Streams and QoS

5.3. Stream synchronization

4.1. Support for Continuous Media

87

- The medium of communication
 - Storage
 - Transmission
 - Representation (screen, etc.)
- Continuous/discrete media

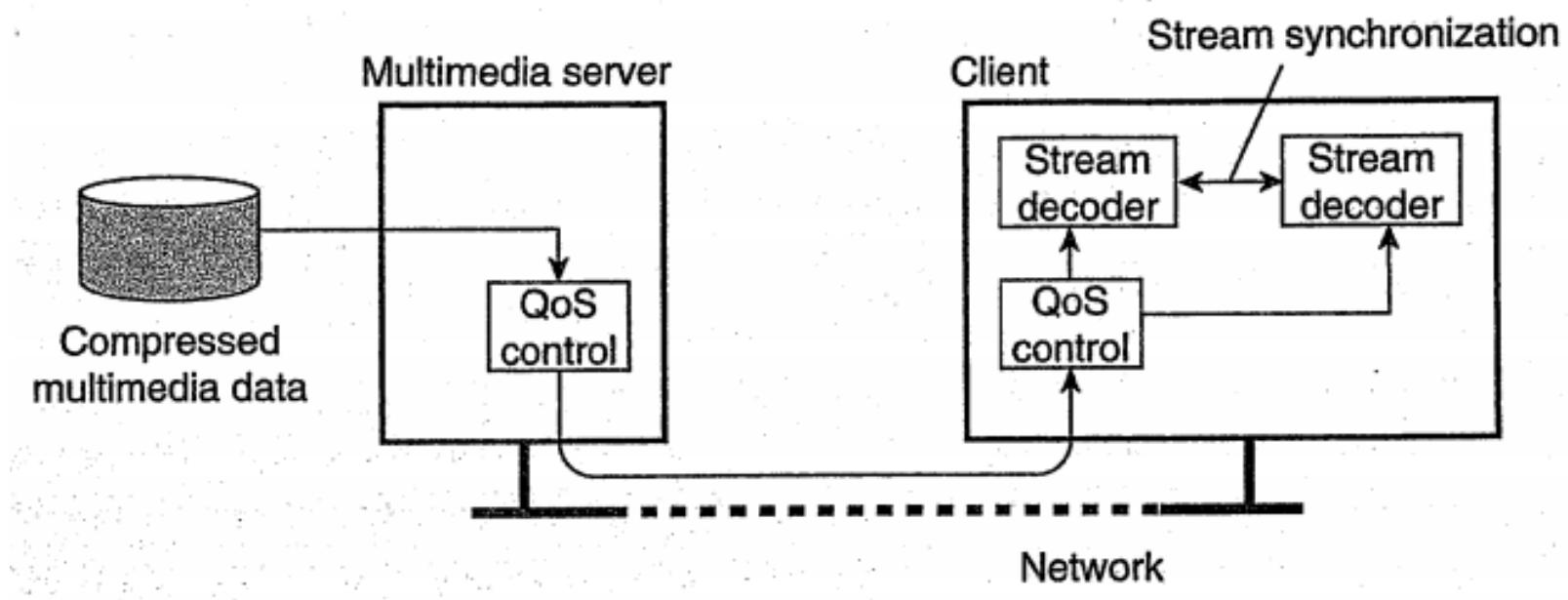
Data stream

88

- Sequence of data units
- Can be applied to discrete and continuous media
- Timing aspects
 - asynchronous
 - synchronous
 - isochronous
- A simple stream: only a single sequence of data
- A complex stream: several related simple streams
- Issues:
 - Data compression
 - QoS
 - Synchronization

Data stream (cont.)

89



A general architecture for streaming stored multimedia data over a network

4.2. Streams and QoS

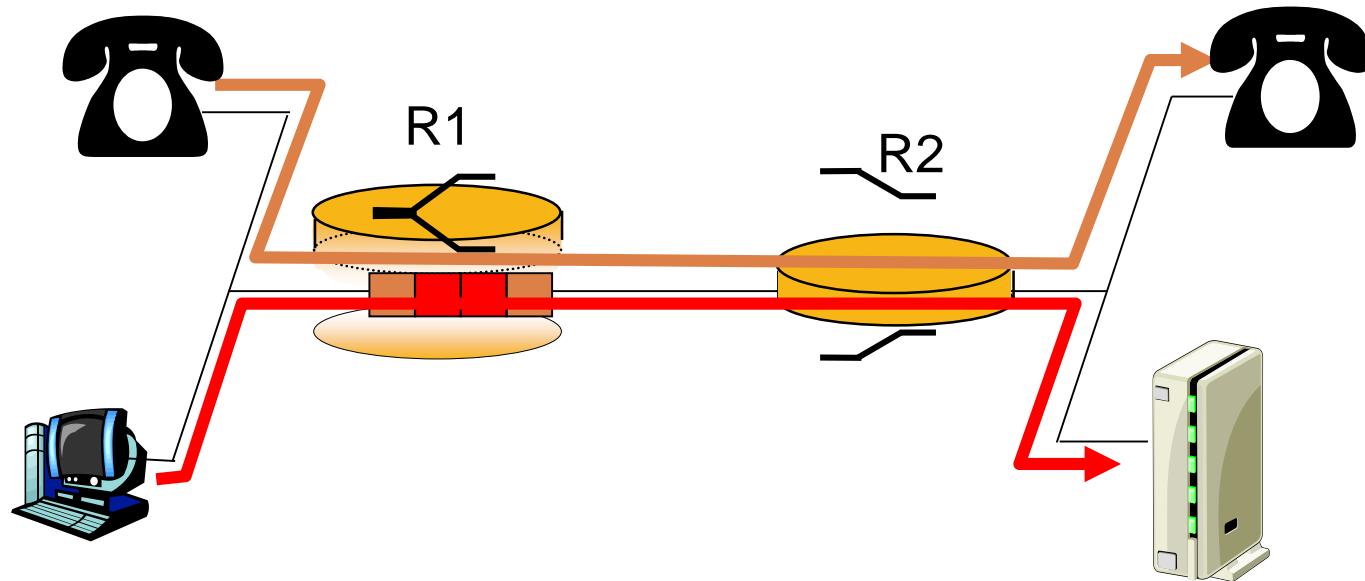
90

- Quality of Service (QoS):
 - bit-rate,
 - delay
 - e2e delay
 - jitter
 - round-trip delay
- Based on IP layer
 - Simple in using best-effort policy

Enforcing QoS

91

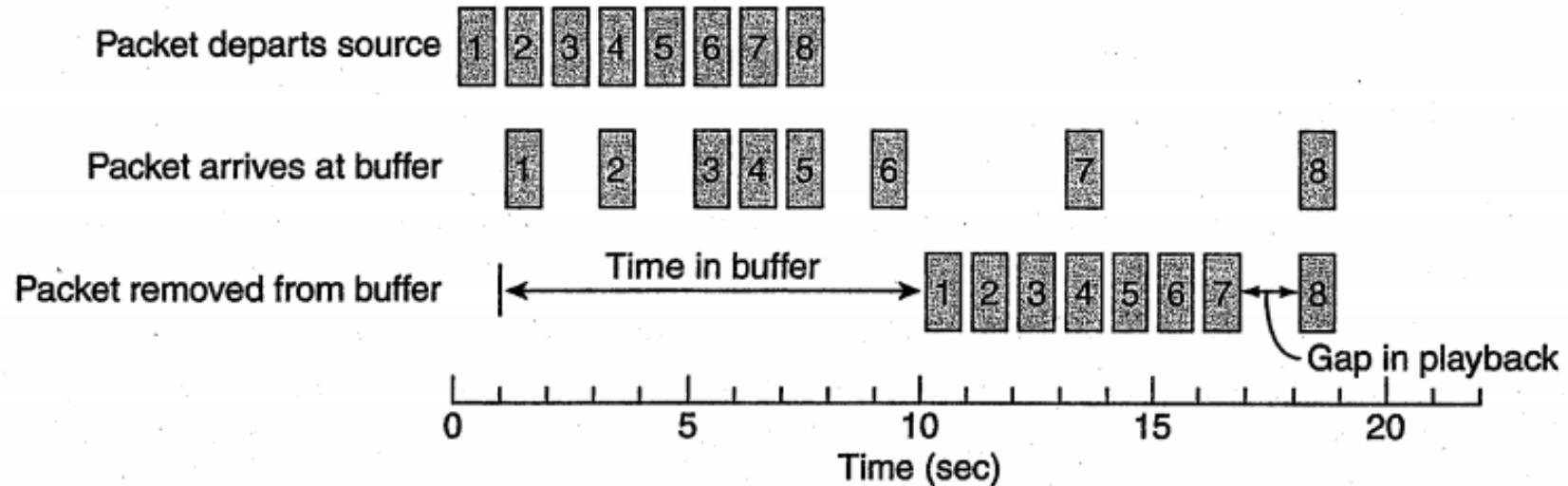
- Differentiated services



Enforcing QoS (cont.)

92

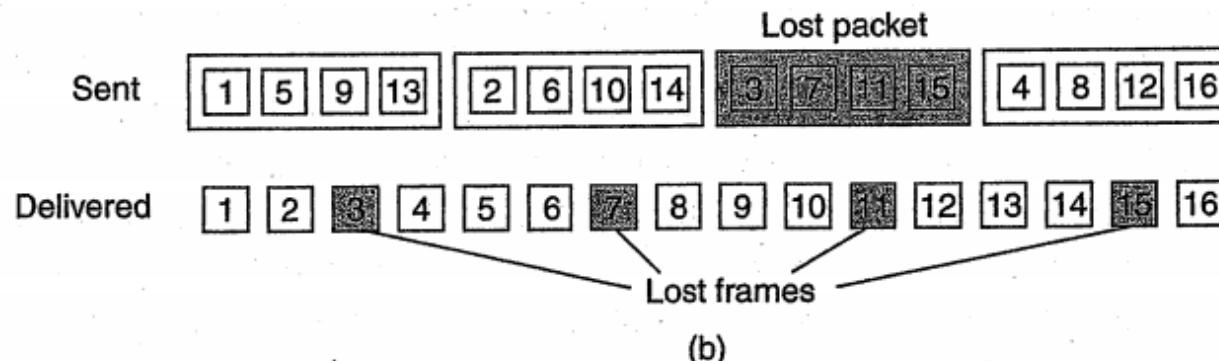
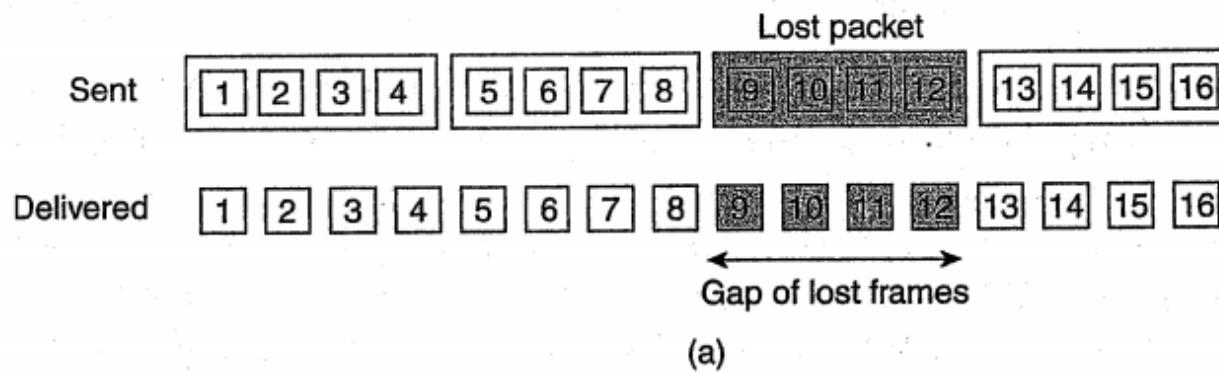
- Using a buffer to reduce jitter



Enforcing QoS (cont.)

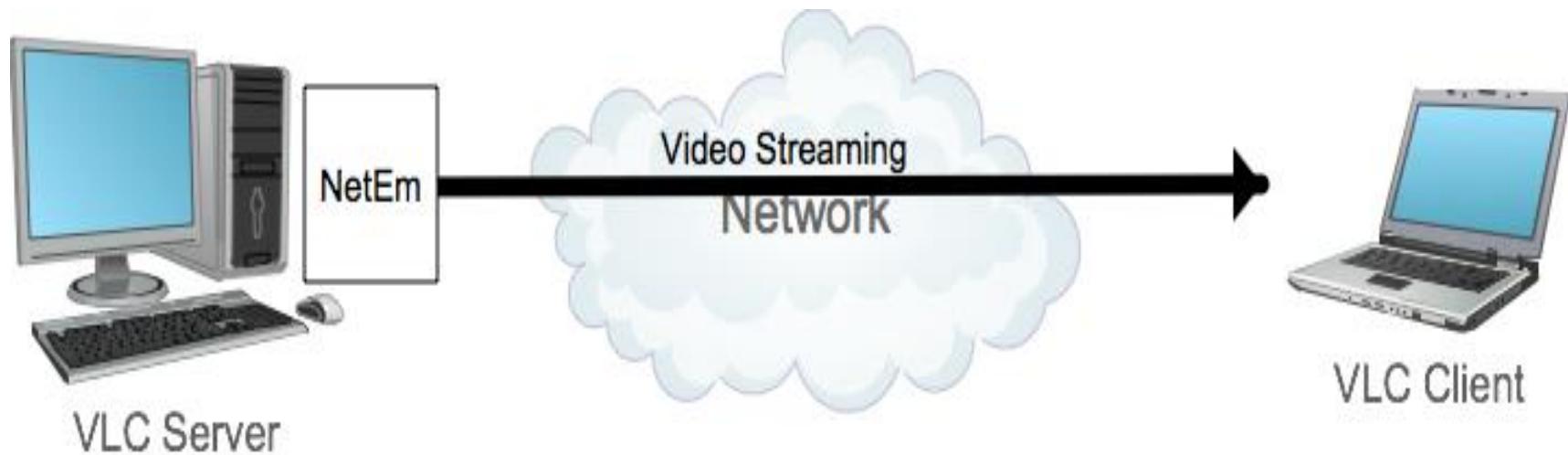
93

- Forward error correction (FEC)
 - Interleaved transmission



Labwork

94



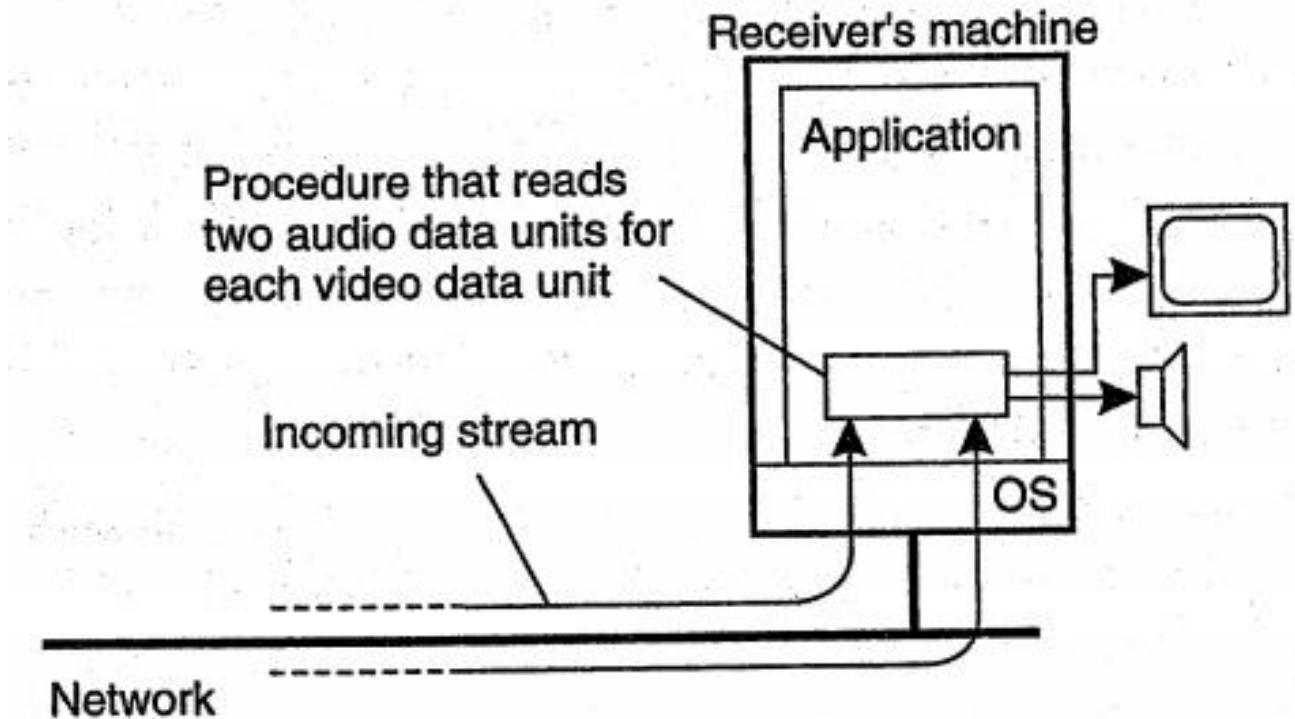
4.3. Stream Synchronization

95

- Needs of stream synchronization
- 2 types:
 - Synchronize *discrete data stream* and *continuous data stream*.
 - Synchronize 2 *continuous data streams*.

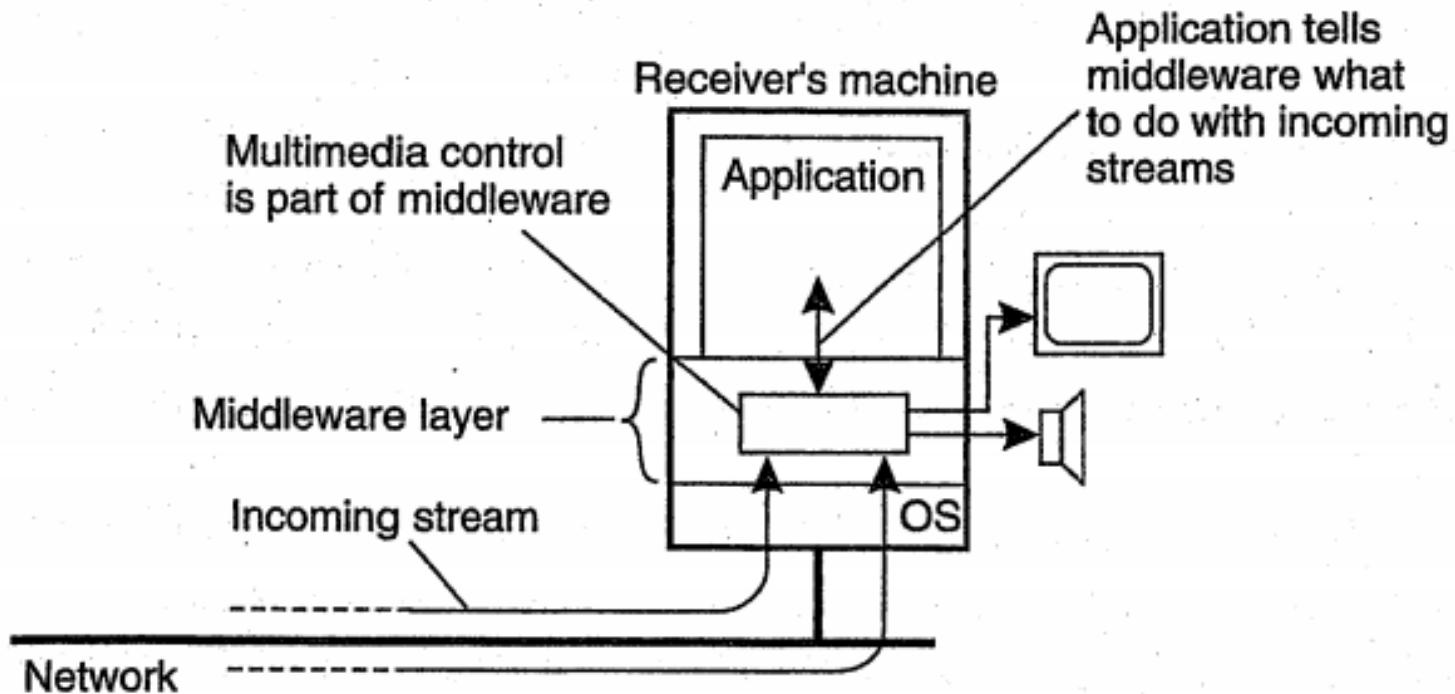
Explicit synchronization on the level data units

96



Synchronization as supported by high-level interfaces

97



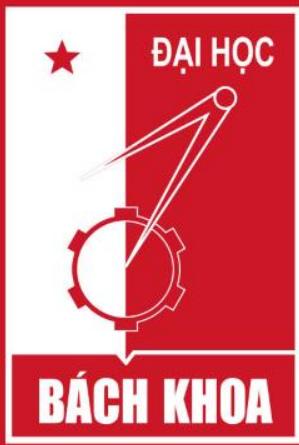


25
YEARS ANNIVERSARY
SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Questions?





25 YEARS ANNIVERSARY
SICT

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

DISTRIBUTED SYSTEMS



HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Chapter 3: Naming in Distributed Systems

Outline

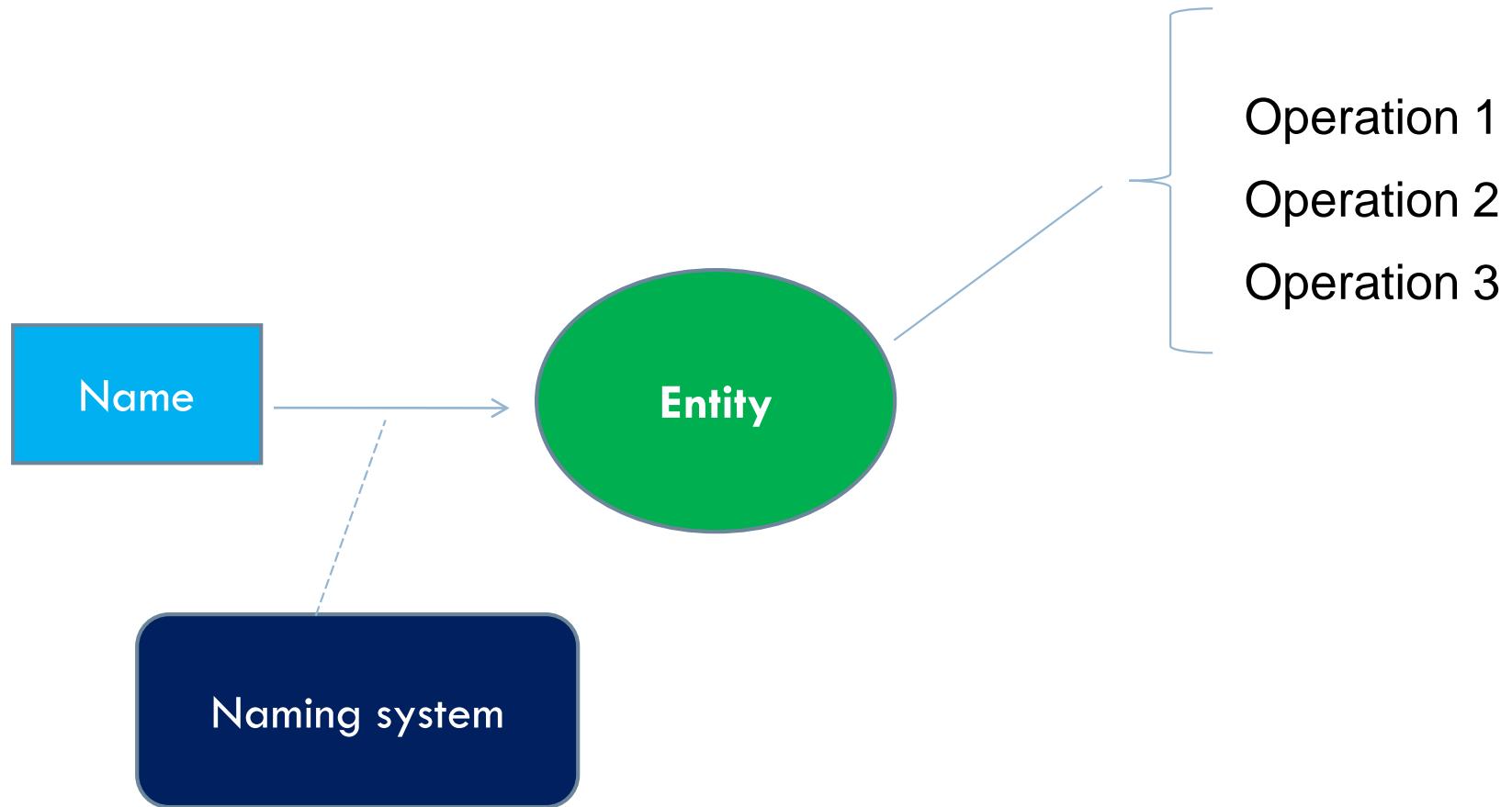
3

- 1. Names. Identifiers and Address**
- 2. Flat Naming**
- 3. Structured Naming**

1. Names. Identifiers and Address

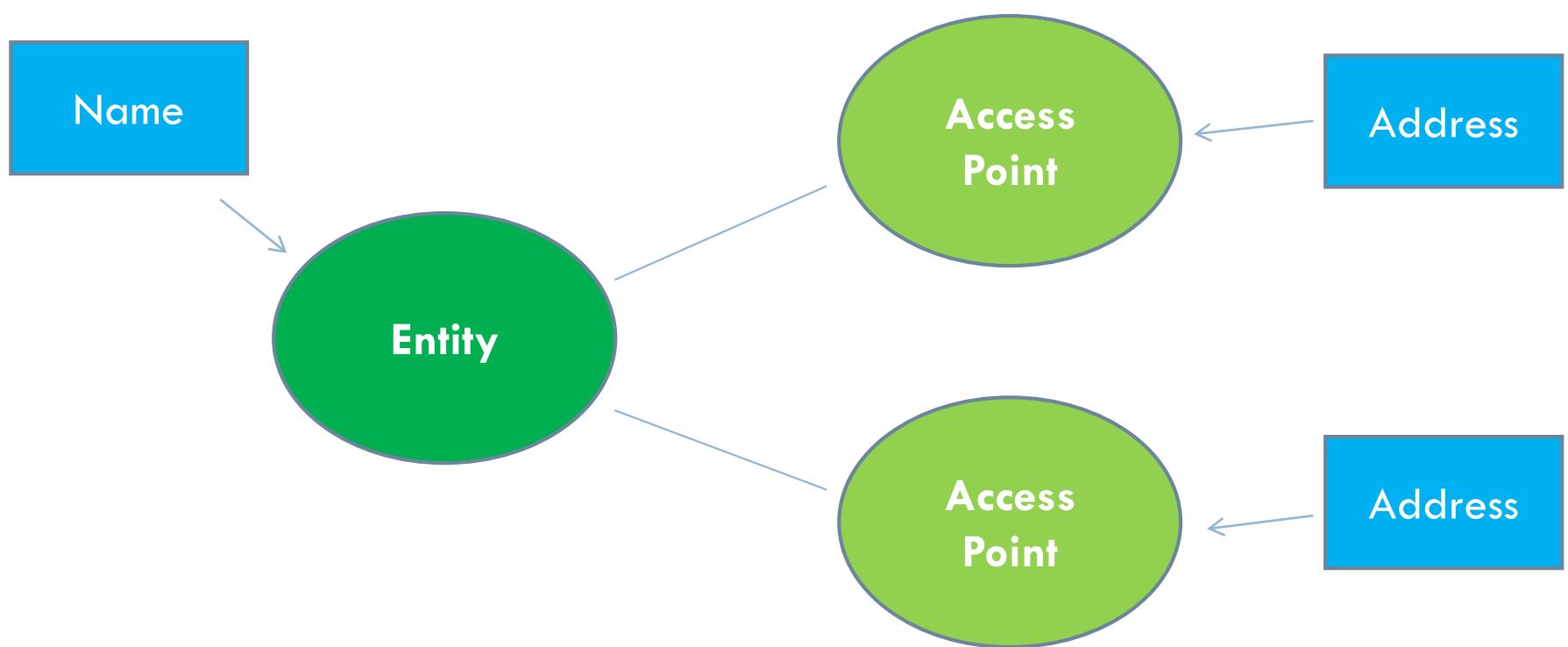
Entity & Name

5



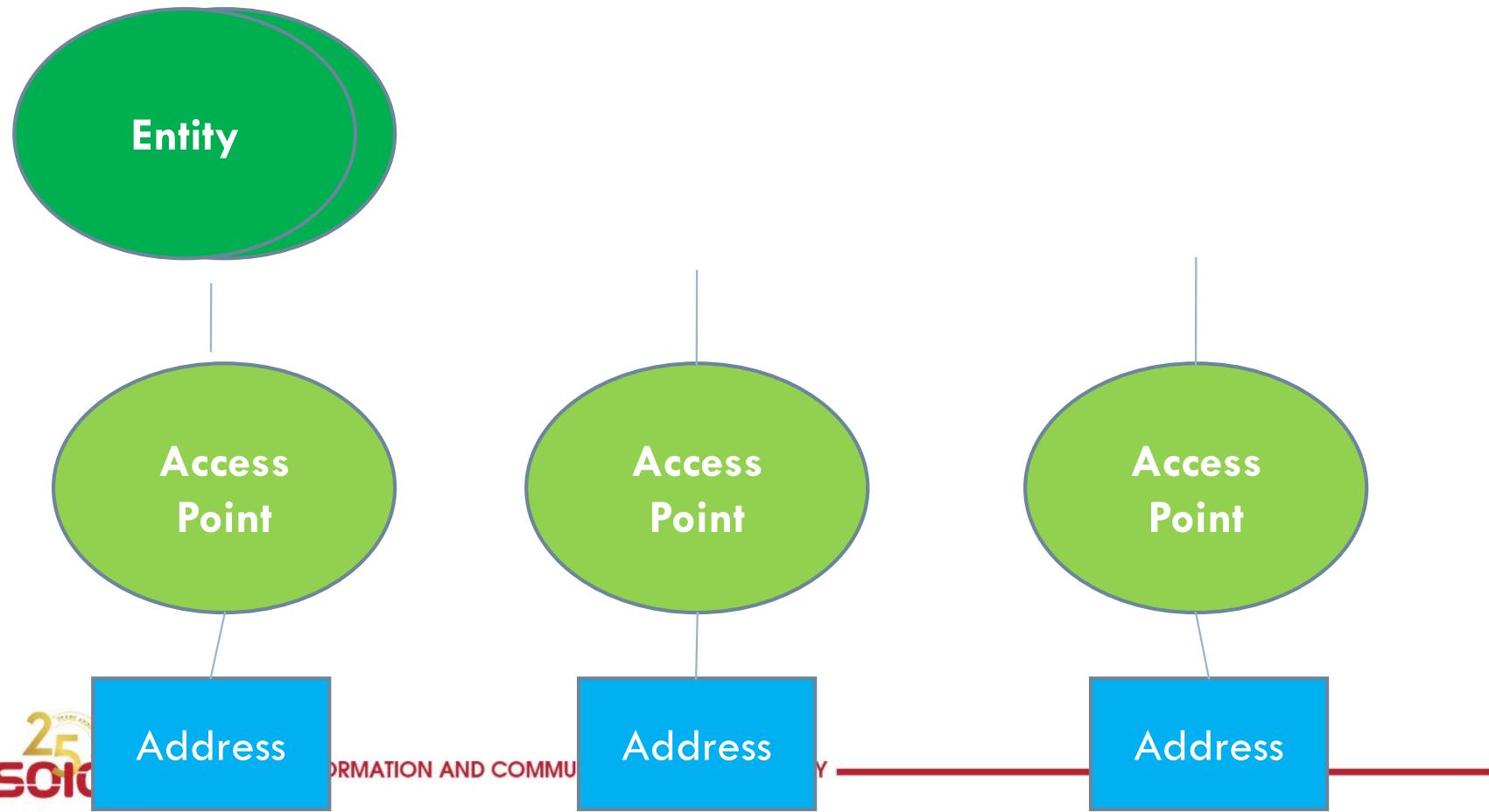
Entity, A.P

6



Location independent

7



Identifier

8

- 3 properties:
 1. An identifier refers to at most one entity.
 2. Each entity is referred to by at most one identifier.
 3. An identifier always refers to the same entity (it is never reused)
- Problems: The exhaustion of Identifier
- Solutions:
 - Extending the namespace
 - Re-assign identifier to new entities

Resolving names and identifiers to addresses

9

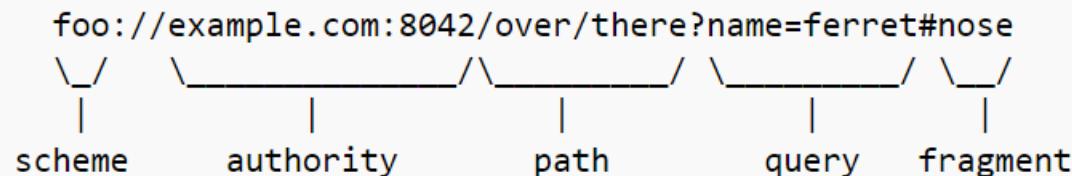
- Centralized approach
 - Name-to-address binding
 - Problem: not appropriate to large network
- □ Distributed naming systems

URI, URL và URN

10

□ URI:

- a string of characters used to identify a resource.
- interact with representations of the resource over a network
- URL and URN
- It comprises 5 parts: scheme, authority, path, query and fragment



□ URN:

- ISBN 0486275574 (run:isbn:0-486-27557-4)

□ URL:

- file:///home/username/RomeoAndJuliet.pdf

11

2. Flat naming

2.1. Definition

12

- Identifiers are simply random bit strings (unstructured)
- It does not contain any information of location
- Goal: how flat names can be resolved
 1. Simple solutions
 2. Home-based Approaches
 3. Distributed Hash Tables
 4. Hierarchical Approaches

2.2. Simple Solutions

13

- **2.2.1. Broadcasting and Multicasting**
- **2.2.2. Forwarding pointers**

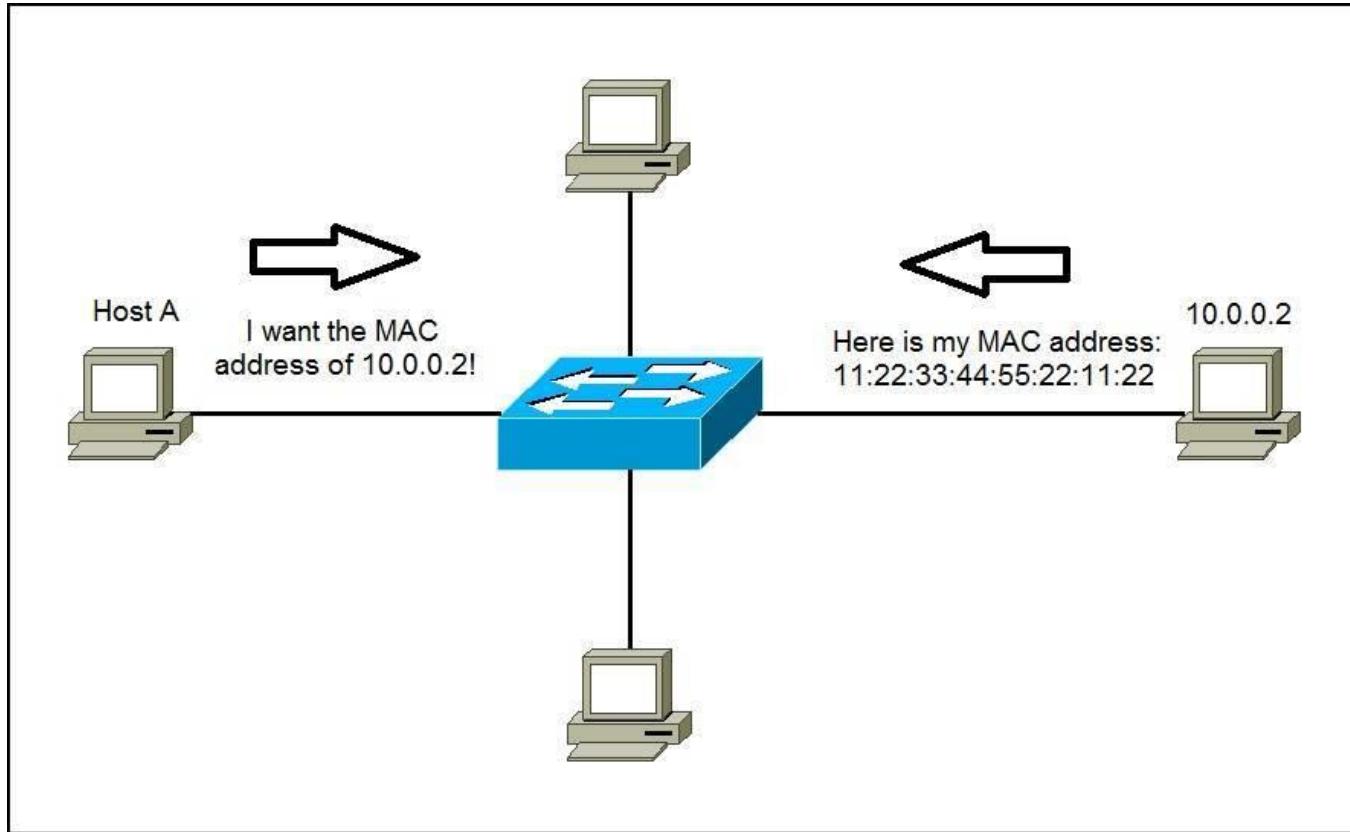
2.2.1. Broadcasting and Multicasting

14

- **Condition: System supports broadcasting facilities:**
 - A message containing the identifier of the entity is broadcasted to all other machines.
 - Each machine is requested to check whether it has that entity.
 - Only the machines that can offer an access point for the entity send a reply message containing the address of that access point.

Example: ARP

15



2.2.1. Broadcasting and Multicasting

16

□ Scalability problem:

- Wast network bandwidth by request messages
- Too many hosts may be interrupted by requests they cannot answer.

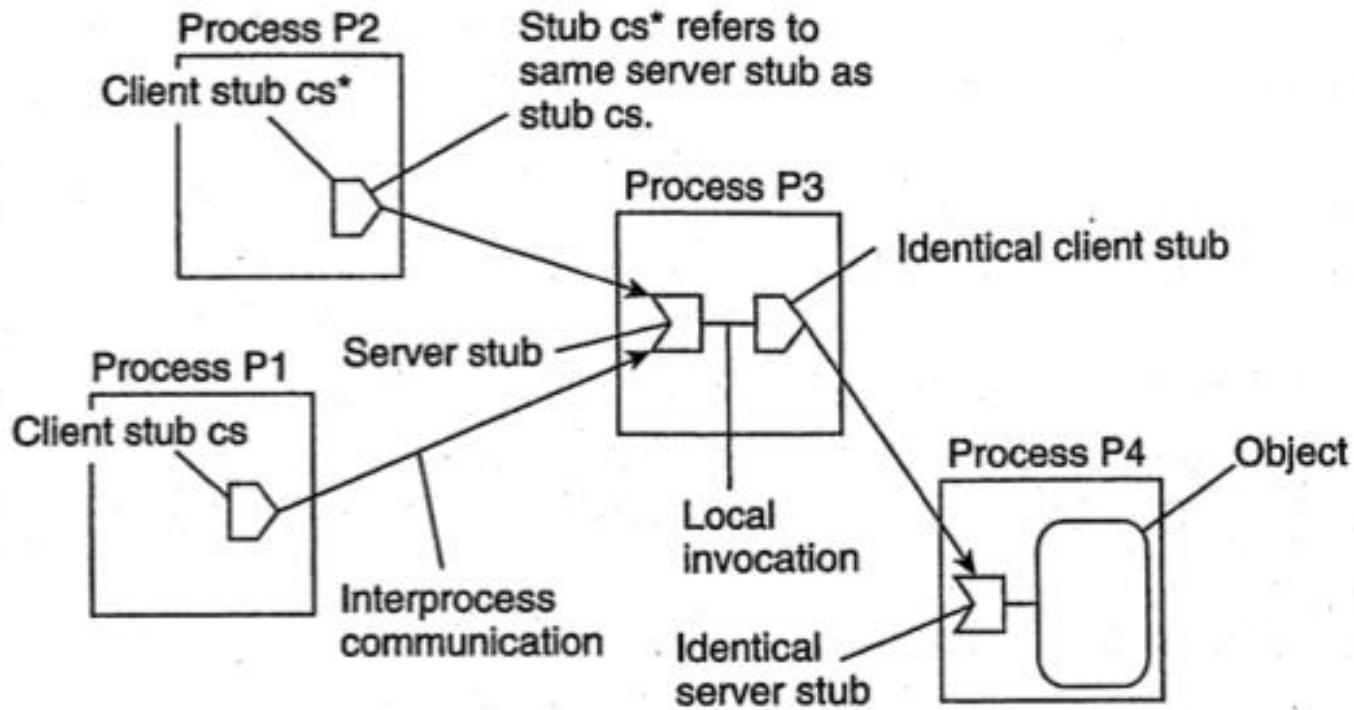
2.2.2. Forwarding Pointer

17

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

Forwarding Pointer mechanism

18



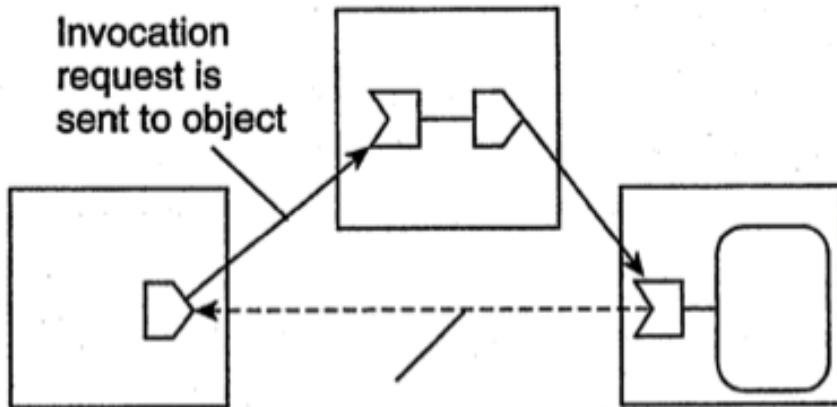
Forwarding Pointer mechanism

- **Advantage:**
 - Simplicity: By using a traditional naming service, a client can look up the current address by following the chain of forwarding pointers.
- **Drawbacks**
 - A chain of FP can become so long → locating that entity is expensive.
 - All intermediate nodes have to maintain their part of the chain.
 - Broken links → cannot reach the entity

Solution: Redirecting a FP

20

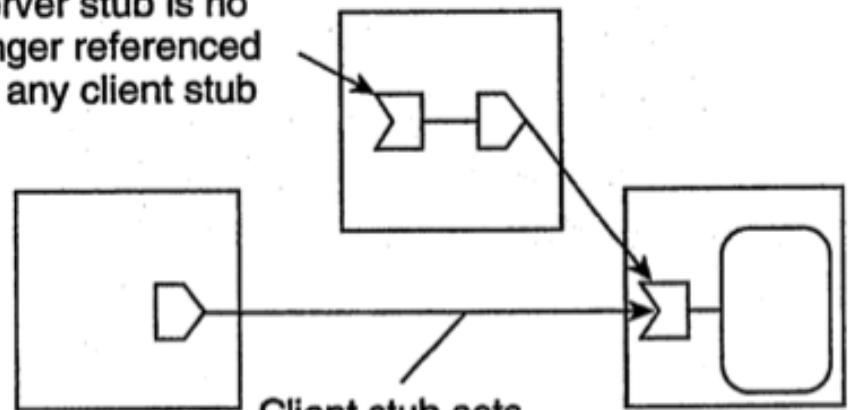
Invocation request is sent to object



Server stub at object's current process returns the current location

(a)

Server stub is no longer referenced by any client stub

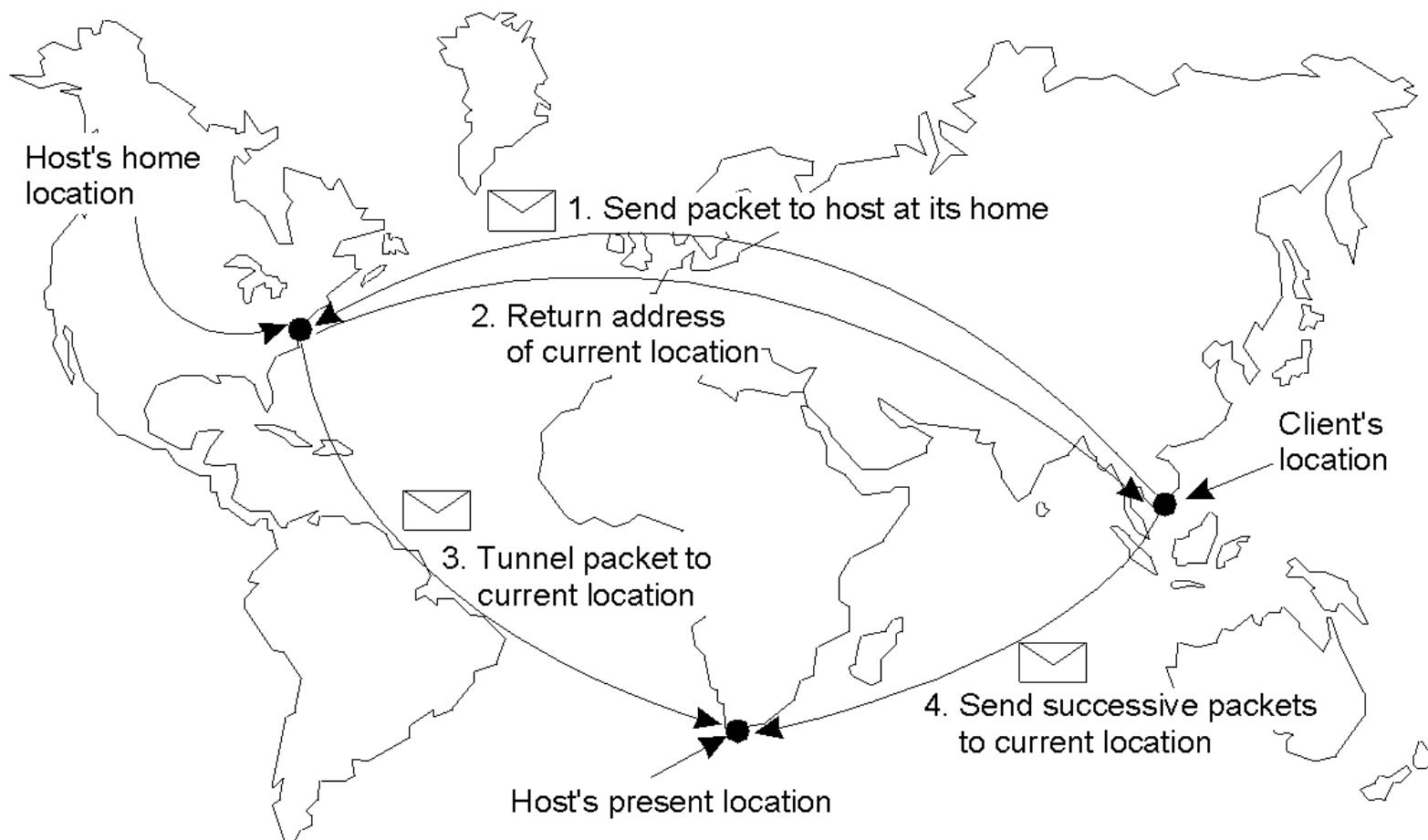


Client stub sets a shortcut

(b)

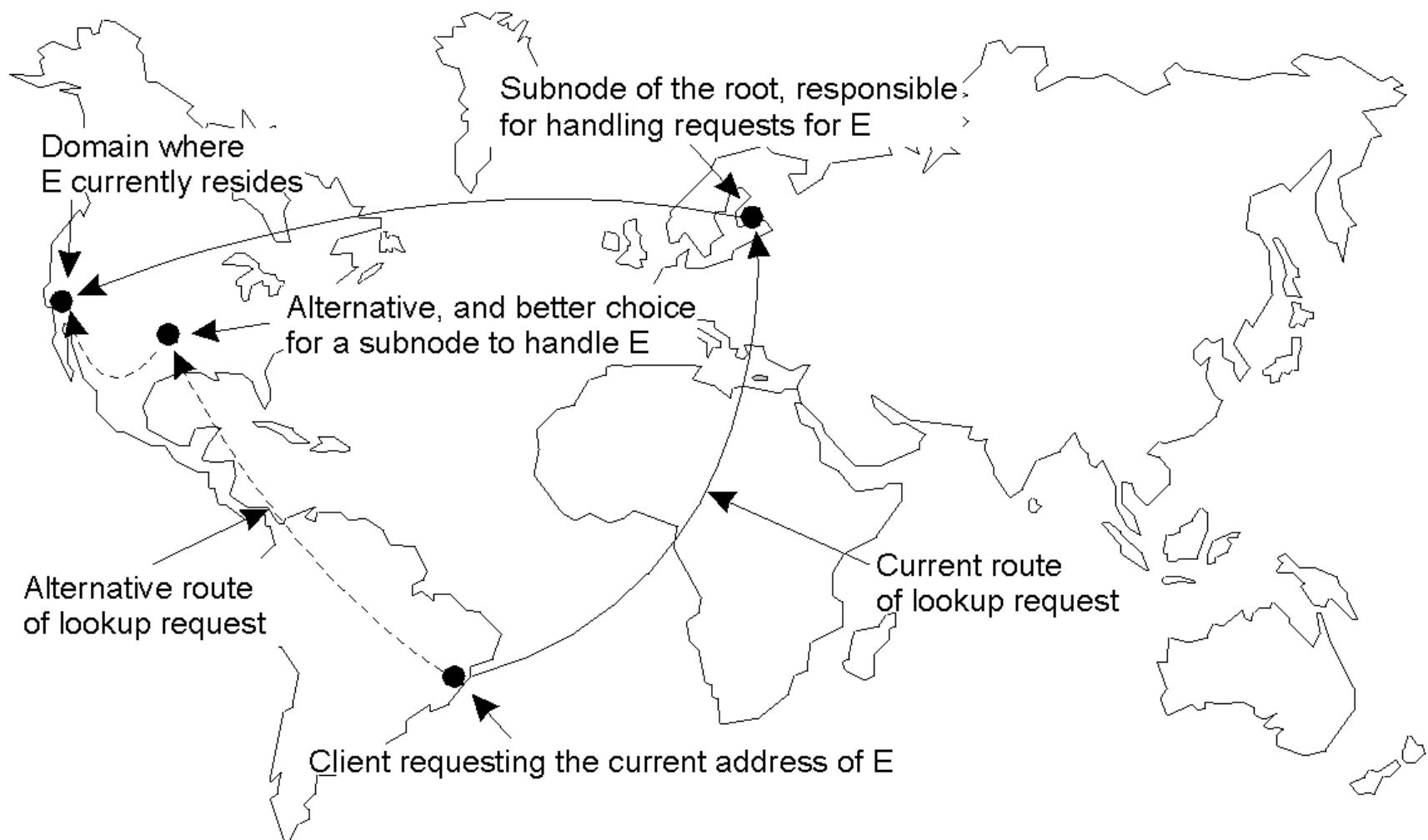
2.3. Home-based Approaches

21



Solution for stable home problem

22



2.4. Distributed Hash Tables

23

- Chord system
- Create the ring with prev(n) and succ(n)
- Use finger table to determine the succ(k) of key k
- FTp is the finger table of node p:

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

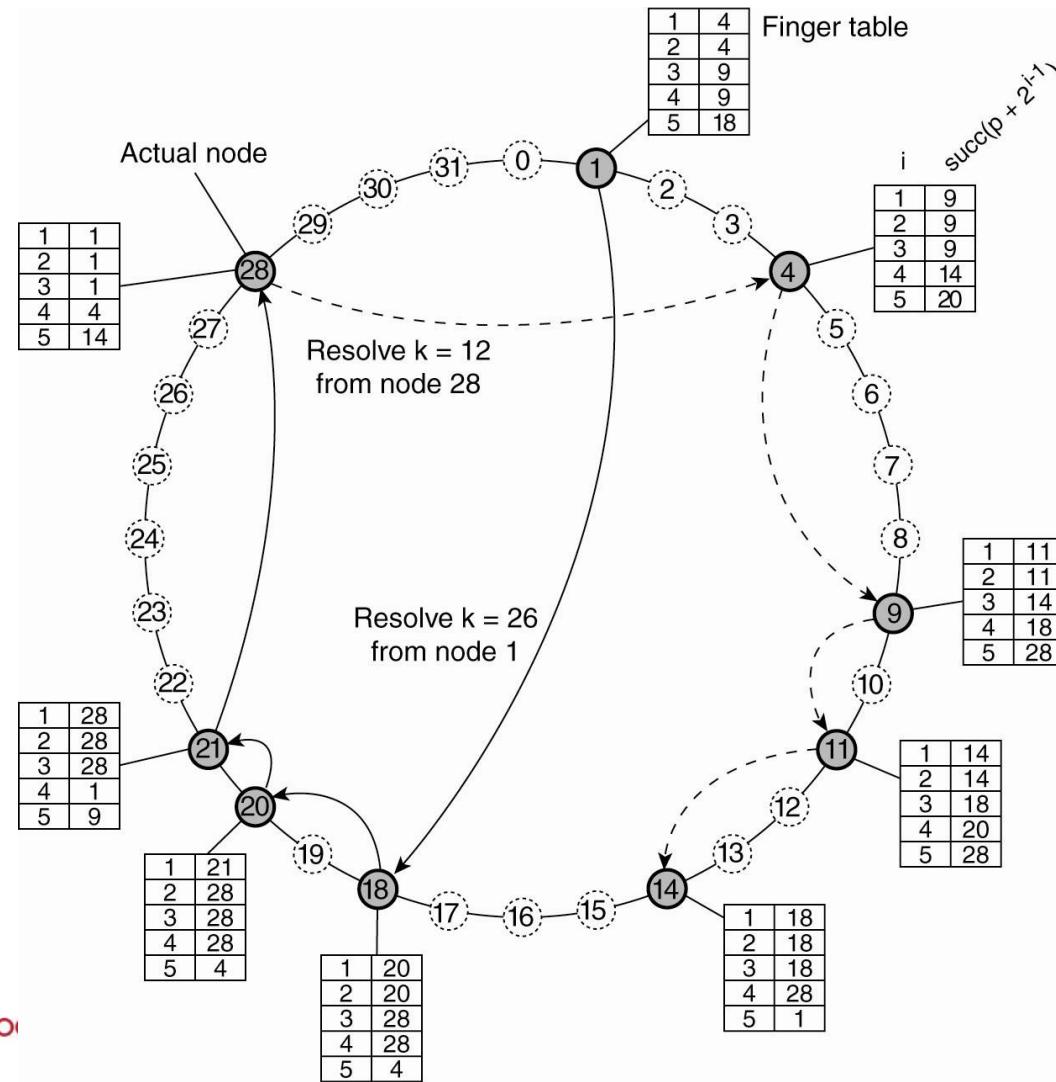
- To look up a key k , node p will then immediately forward the request to node q :

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

- Update the finger tables after inserting a new node

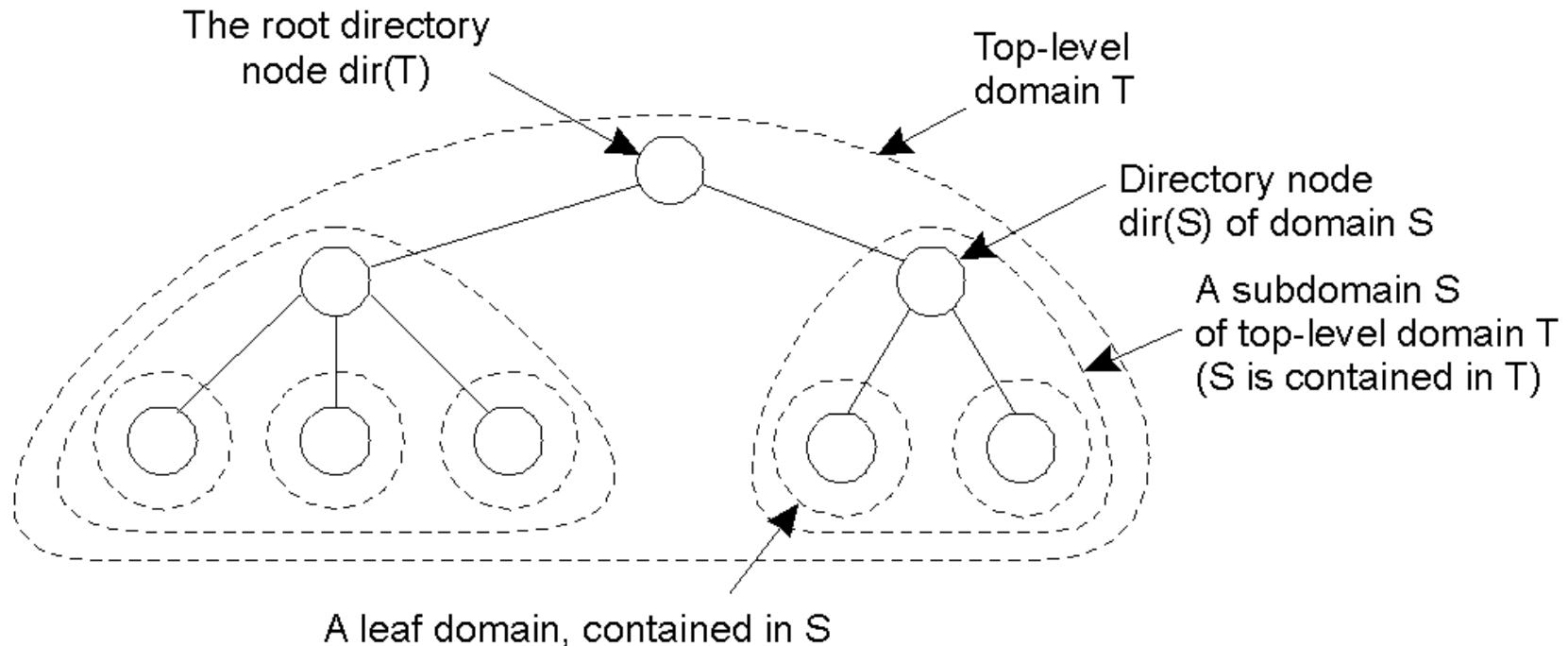
Chord system with finger tables

24



2.5. Hierarchical Approaches

25

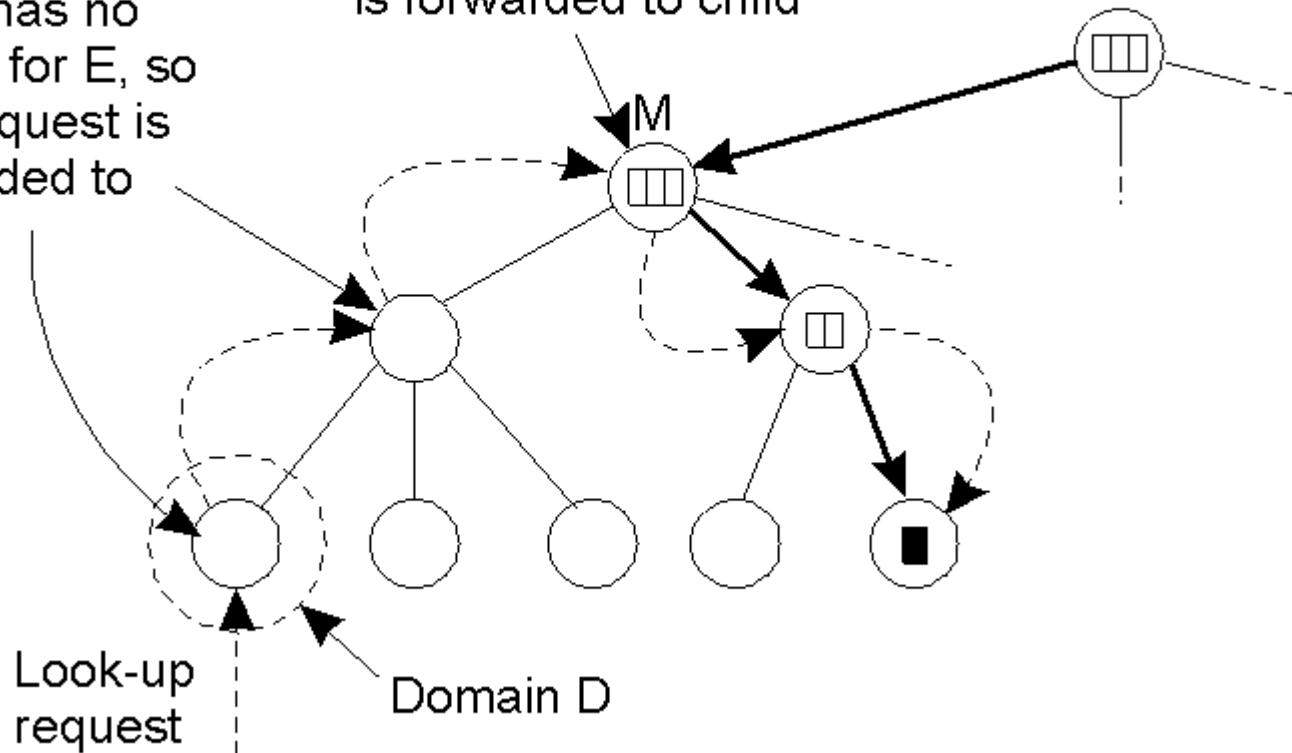


Looking-up

26

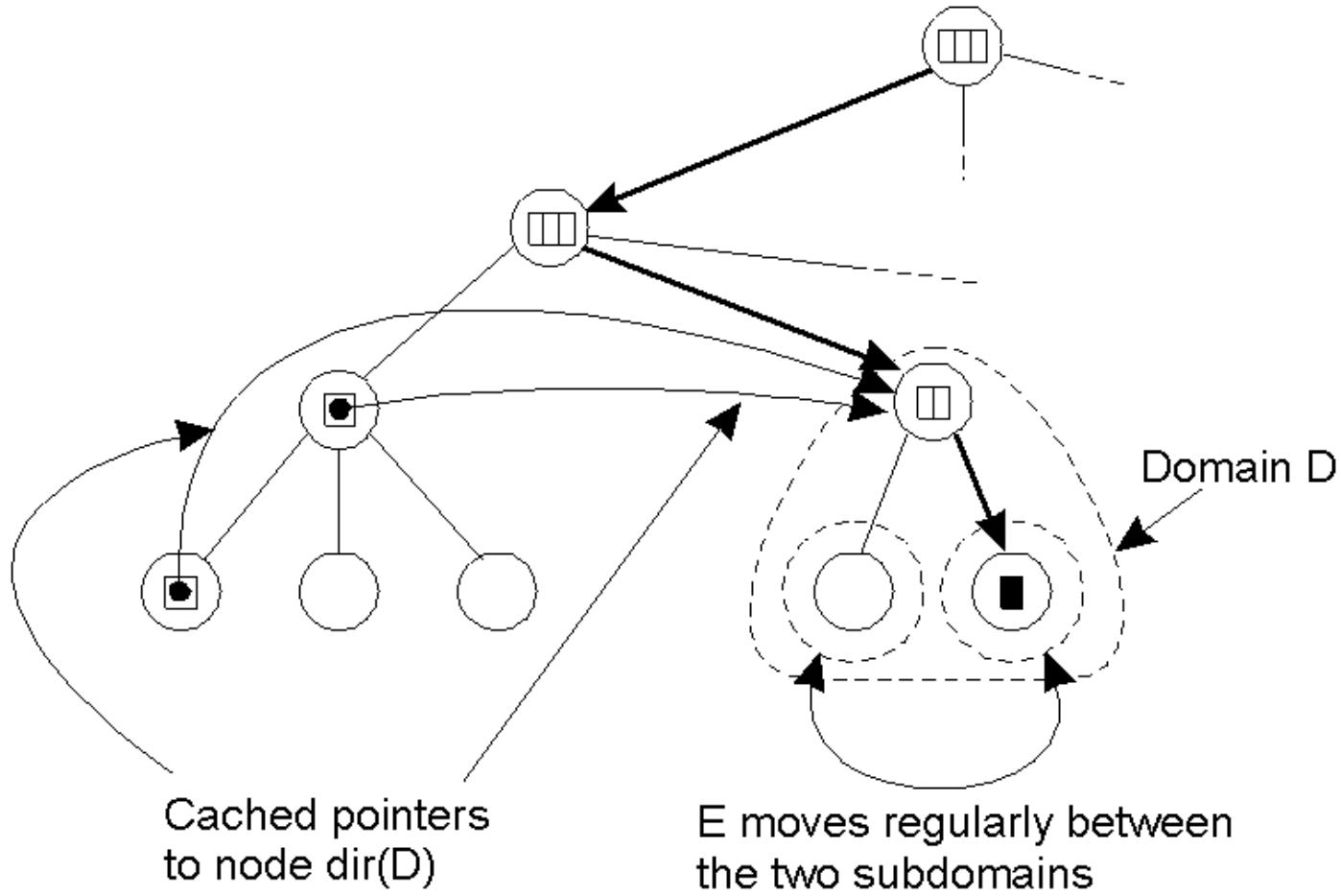
Node has no record for E, so that request is forwarded to parent

Node knows about E, so request is forwarded to child



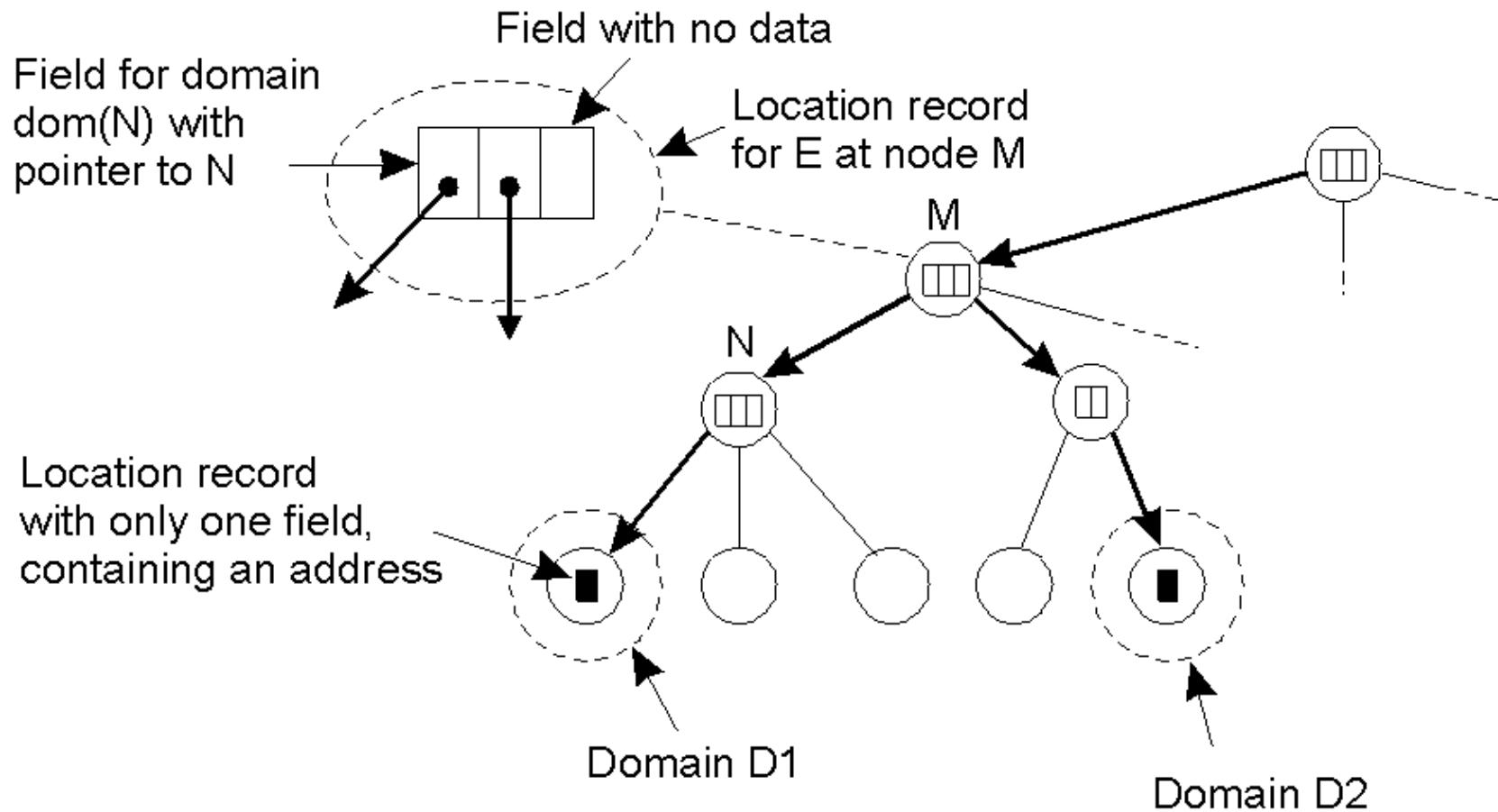
Caching

27



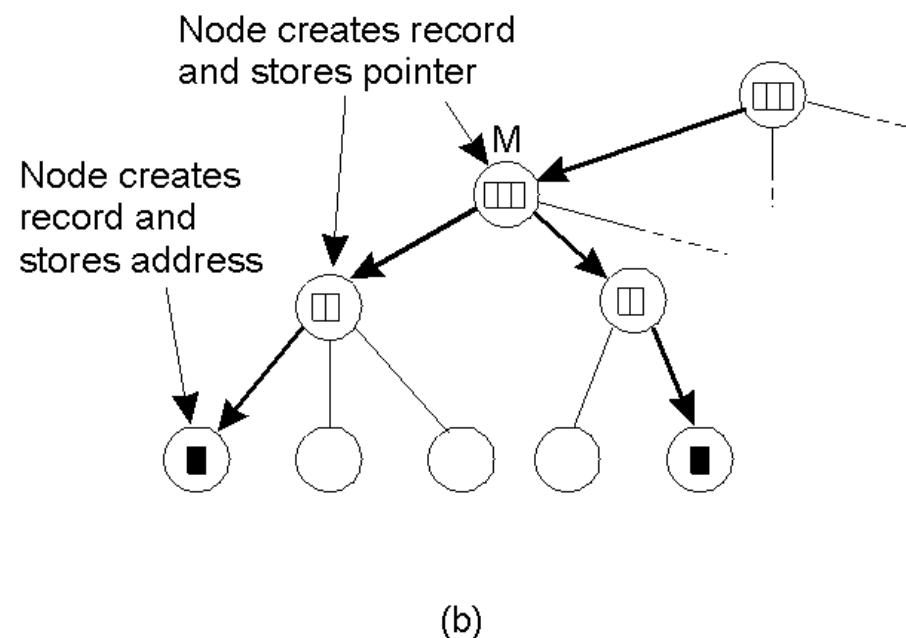
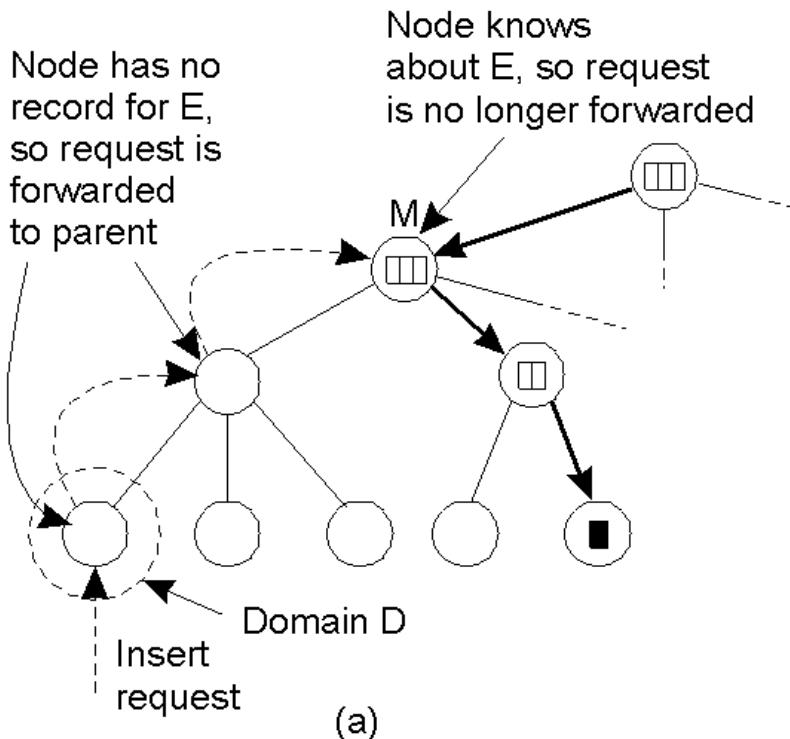
An entity having two addresses in different leaf domains

28



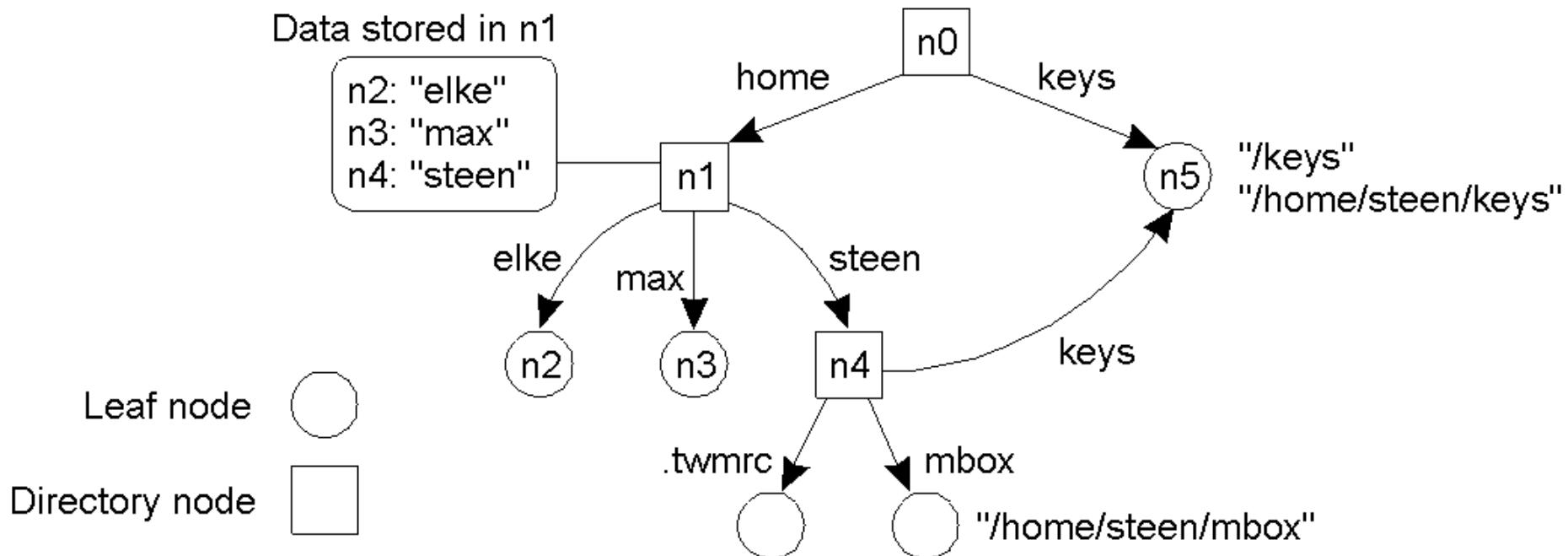
Updating

29



3. Structured Naming

Structured Name Space



A general naming graph

Name Spaces

32

- **Leaf node:**
 - No outgoing edge
 - Store information of its address
- **Directory node:**
 - Outgoing edge
 - Store a table with info (edge label, node identifier)
- **Path name: N: <label1, label2, label3, label4, ...>**
- **Absolute path name/Relative path name**

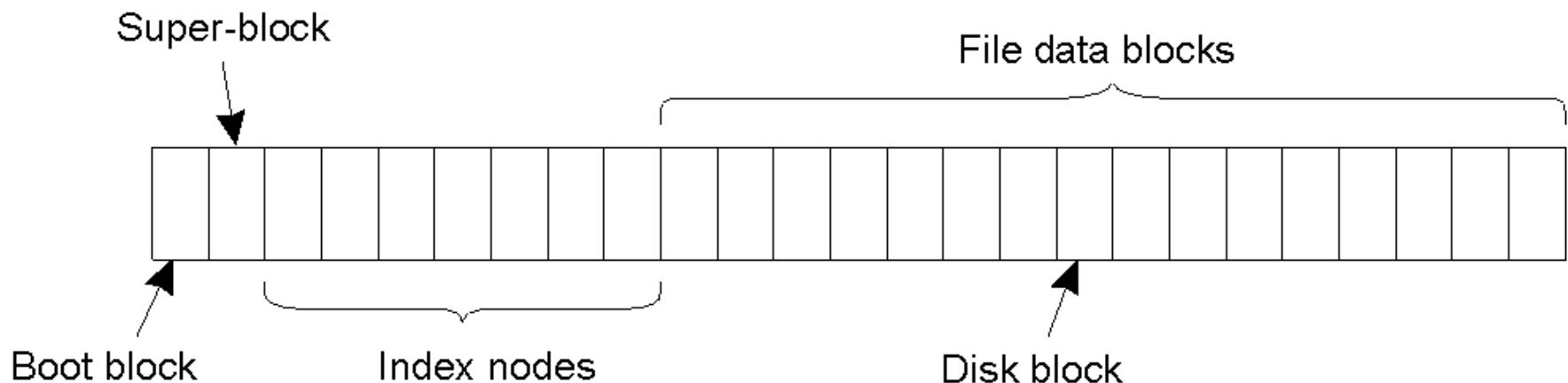
Name resolution

33

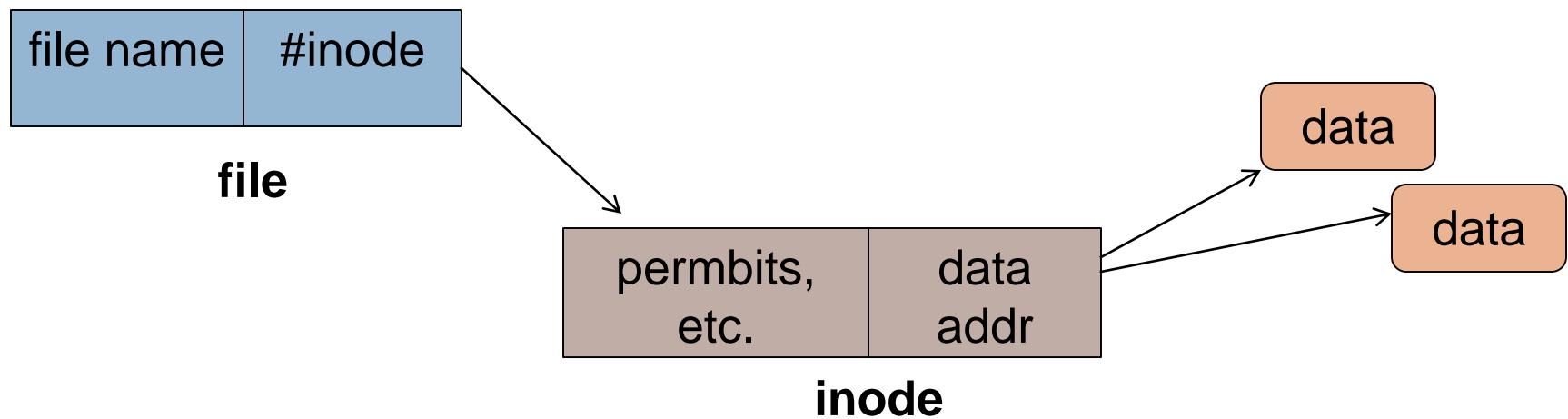
- Consider a path name: N:<label1, label2, ..., labeln>
- Start at node N of the naming graph, where the name label1 is looked up in the directory table, and which returns the identifier of the node to which label1 refers.
- Continue at the identified node by looking up the name label2
- So on ...
- Relatively with the UNIX file system

General organization of the UNIX file system

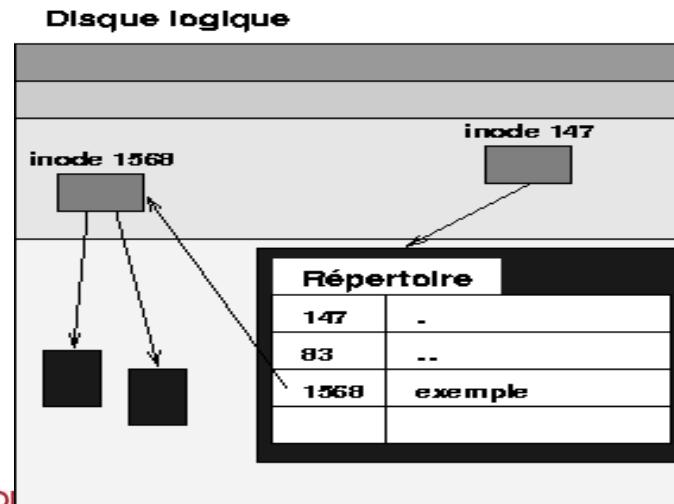
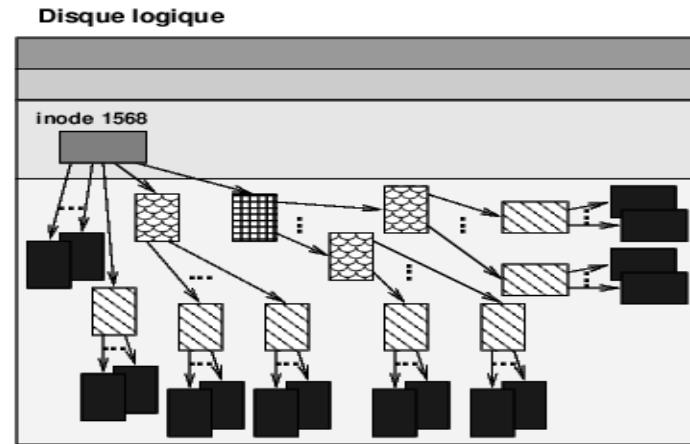
34



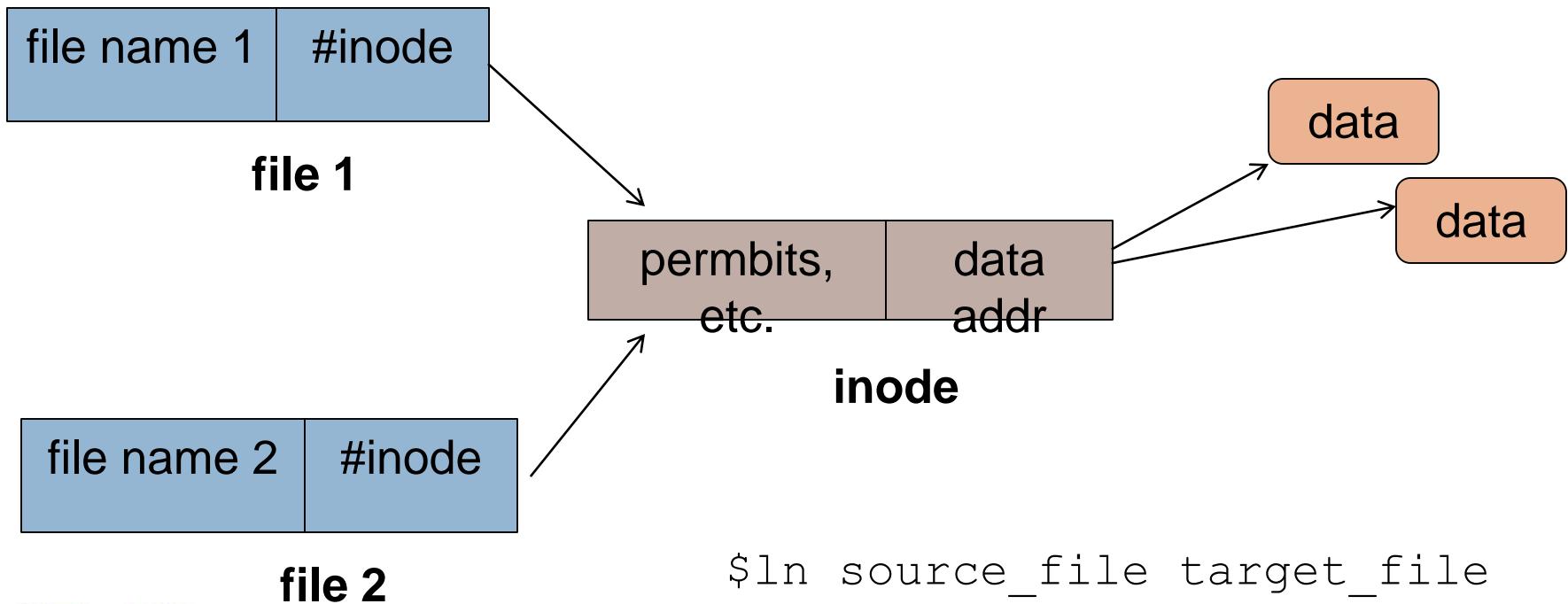
File system in UNIX



Directory node (folder)

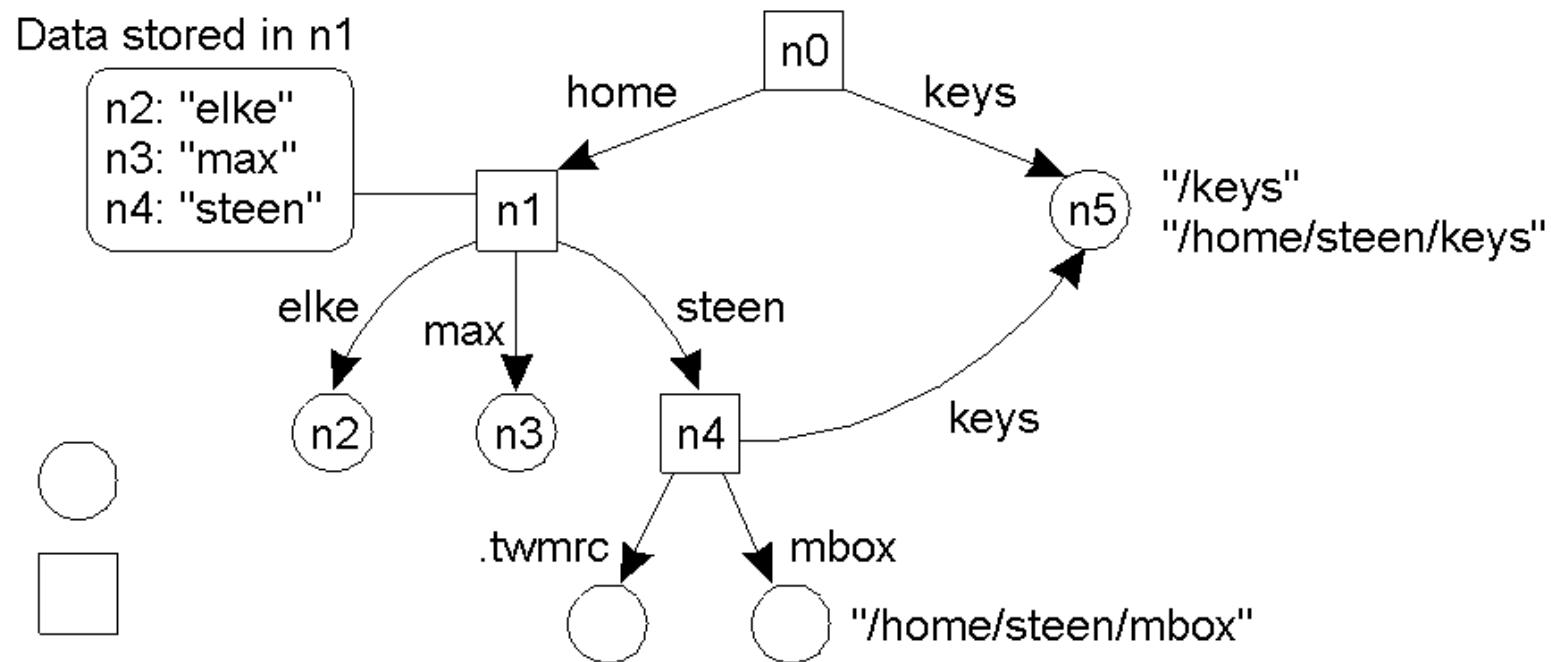


Hard link

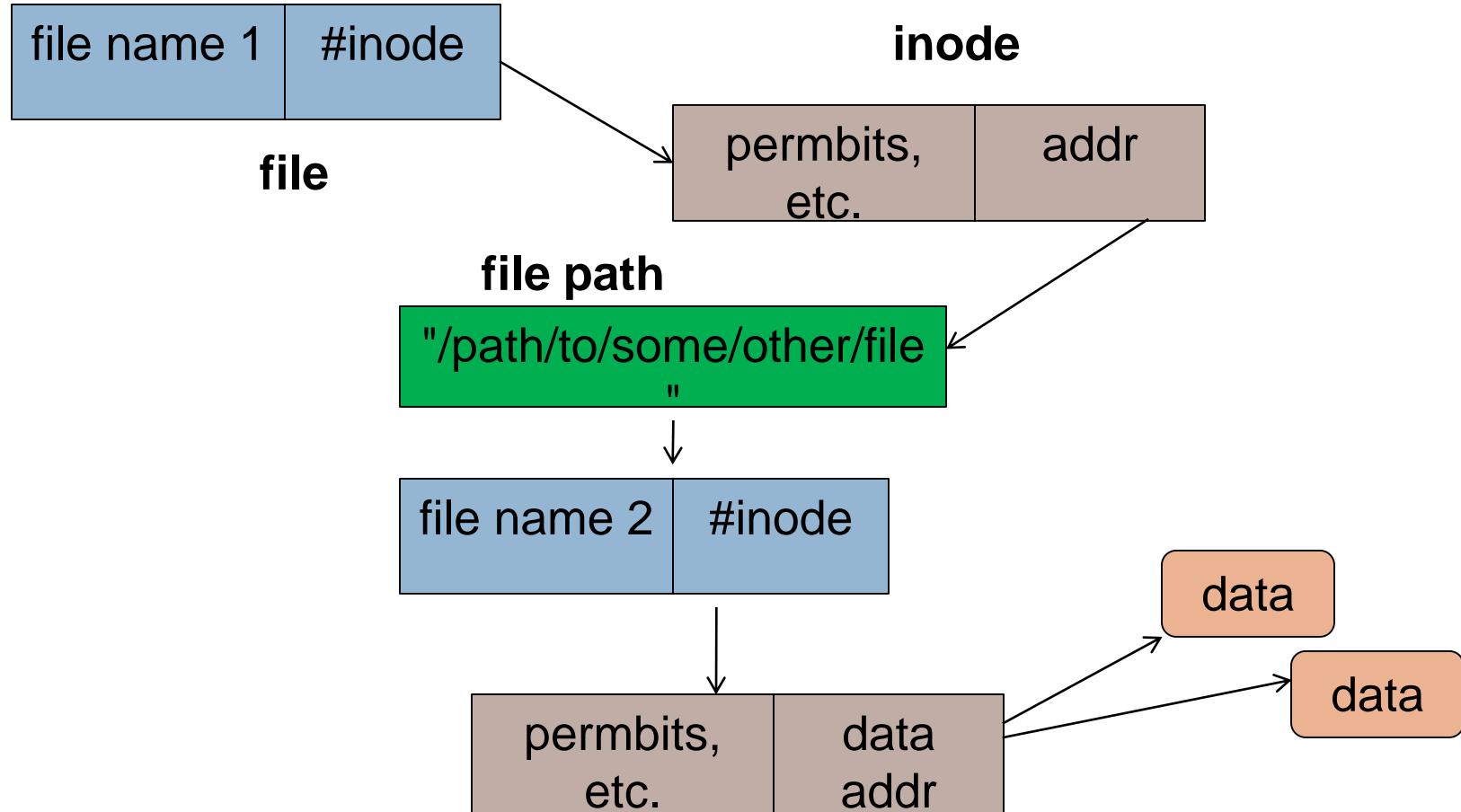


Hard link (cont.)

38

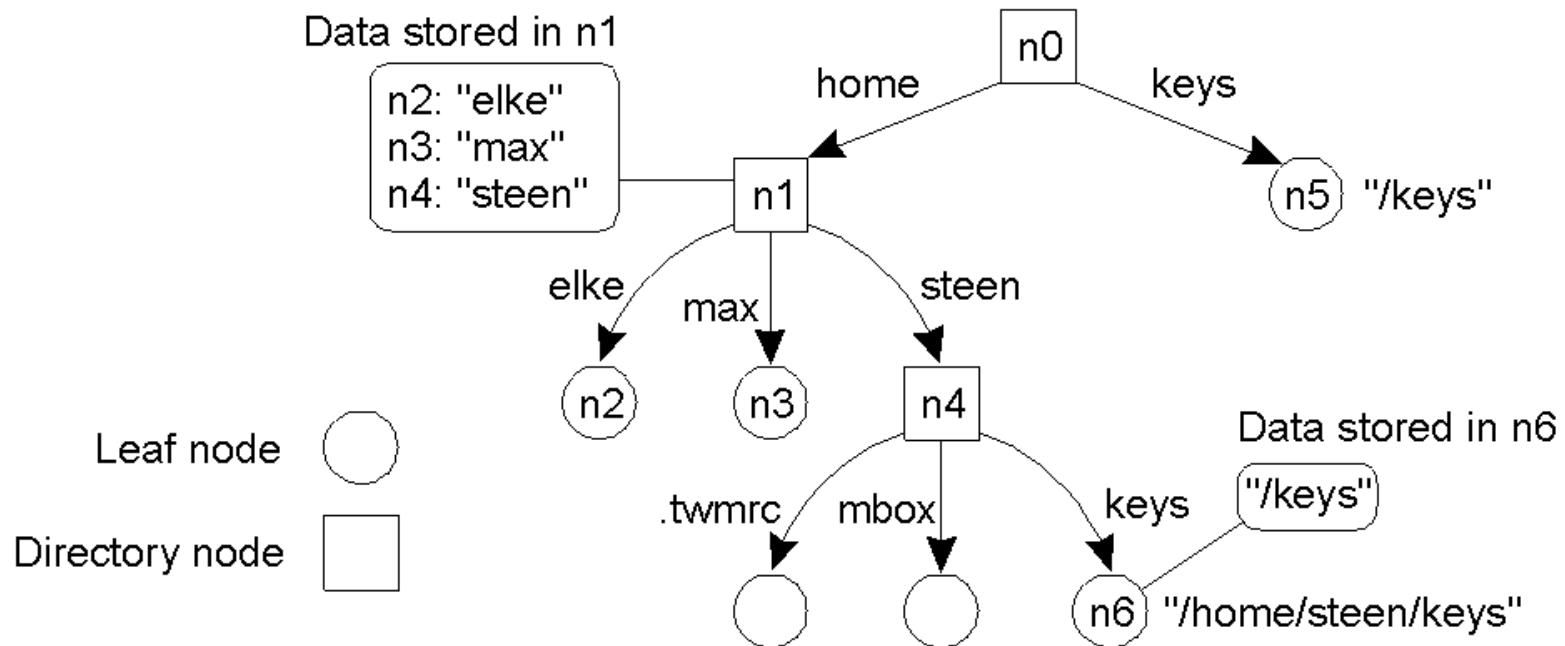


Soft link



Soft link

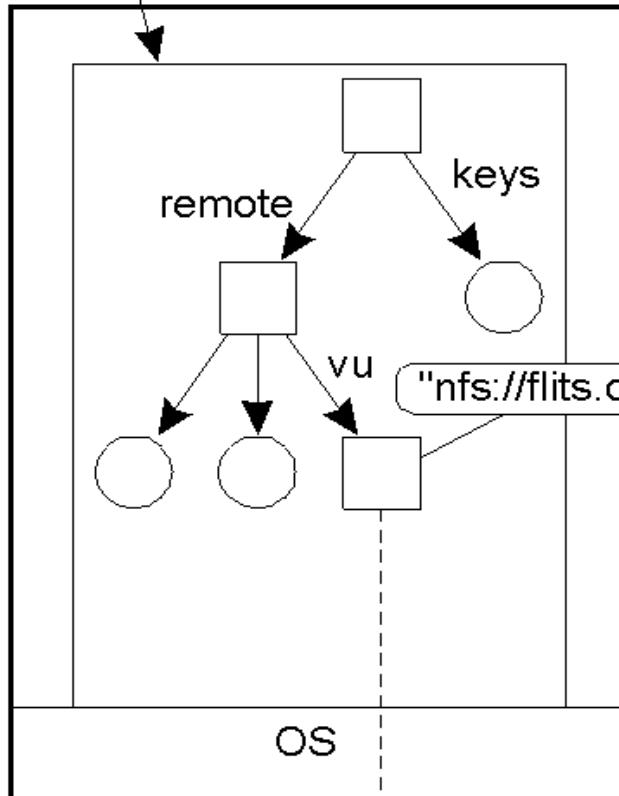
40



Mounting

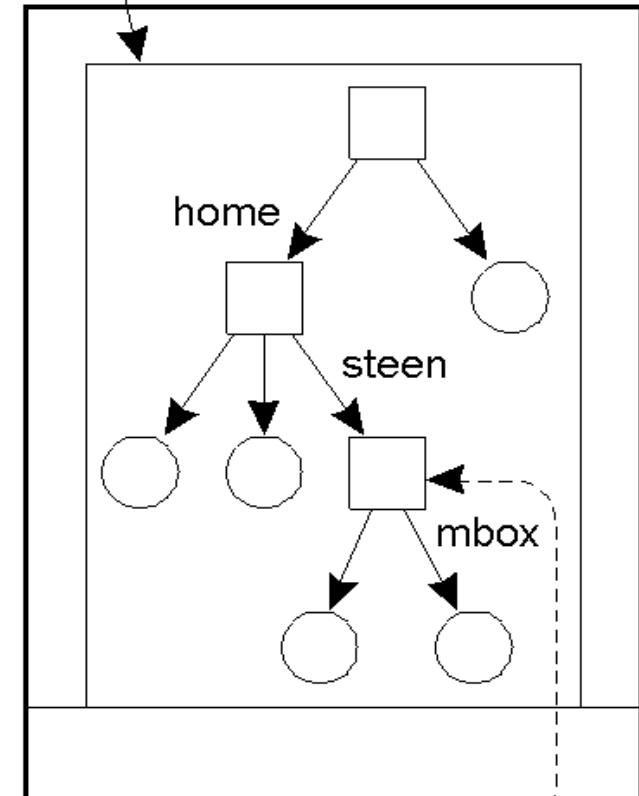
Name server

Machine A



Name server for foreign name space

Machine B

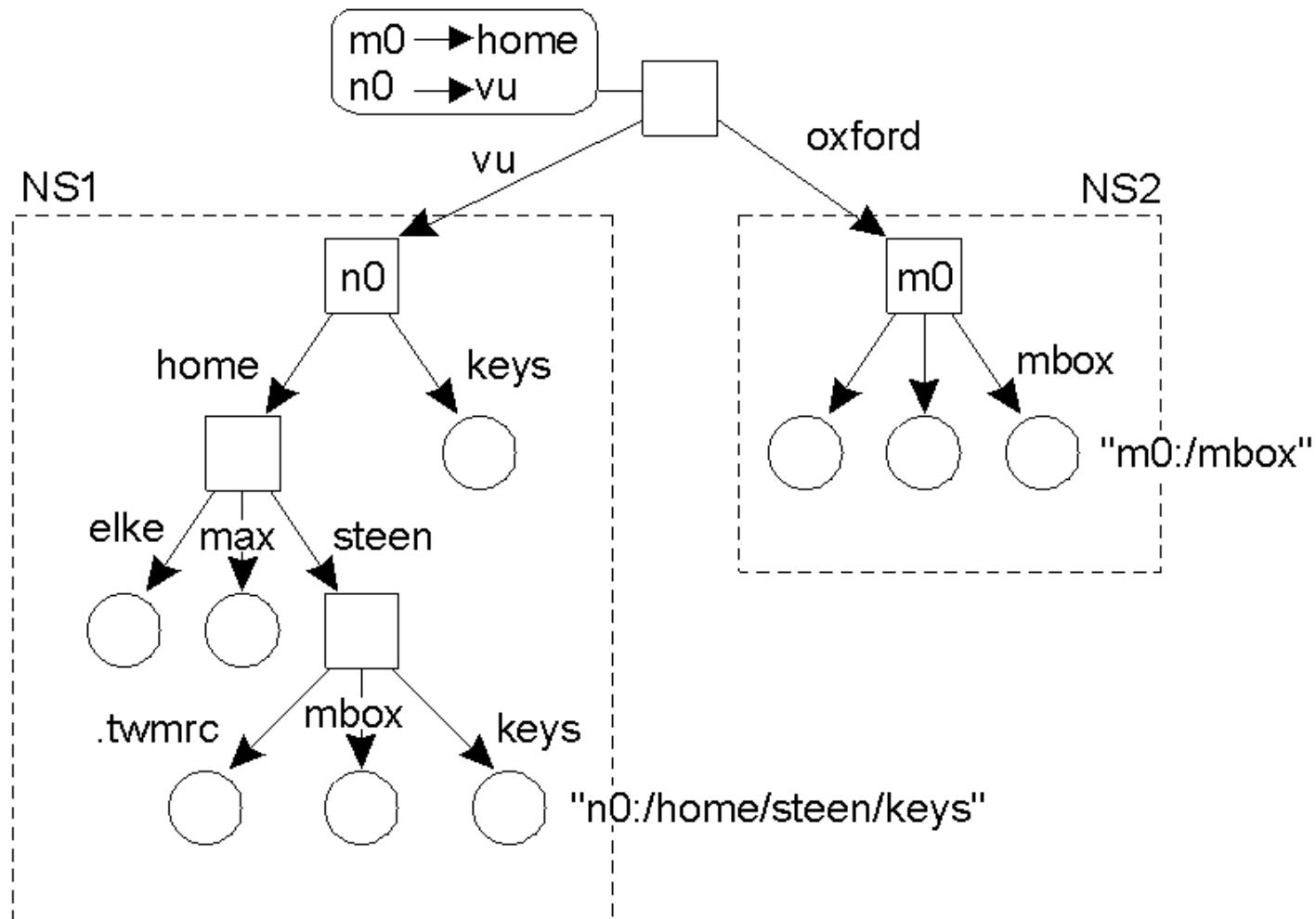


Reference to foreign name space

Network

Merging

42



Naming service

43

- Functions:
 - Adding names
 - Removing names
 - Looking up names
- Naming service is implemented by name servers
- In large-scale distributed systems (many entities, large geographical area) → distribute the implementation of a name space over multiple name servers

Hierarchical organization

44

□ **Global layer**

- root node + directory nodes logically close to the root (children)
- Stability (rarely changed)
- represent organization, or group of organization

□ **Administrational layer**

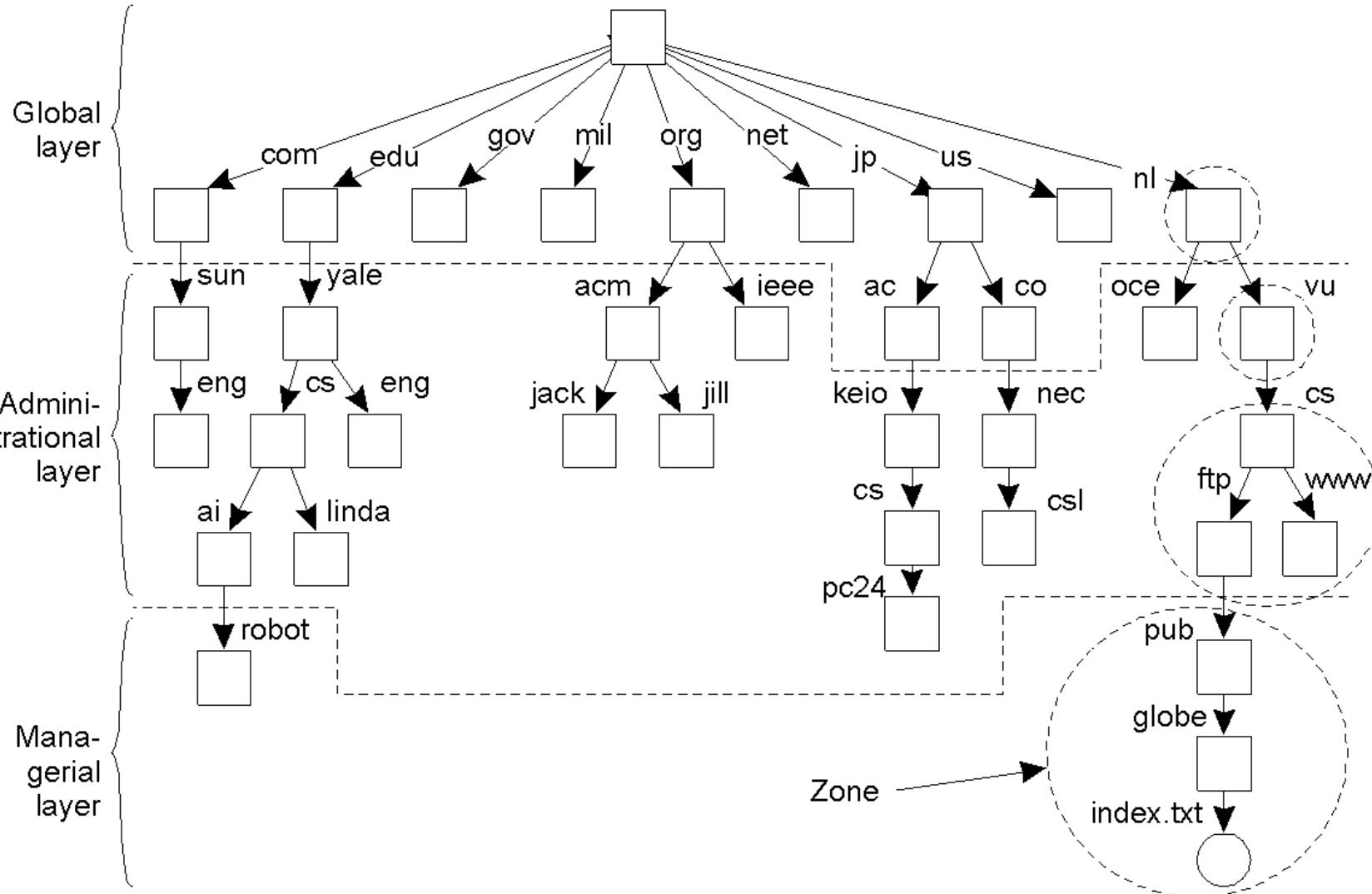
- represent groups of entities that belong to the same organization

□ **Managerial layer**

- consist of nodes that may change regularly

DNS name space

4.



Comparison of three layers

46

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

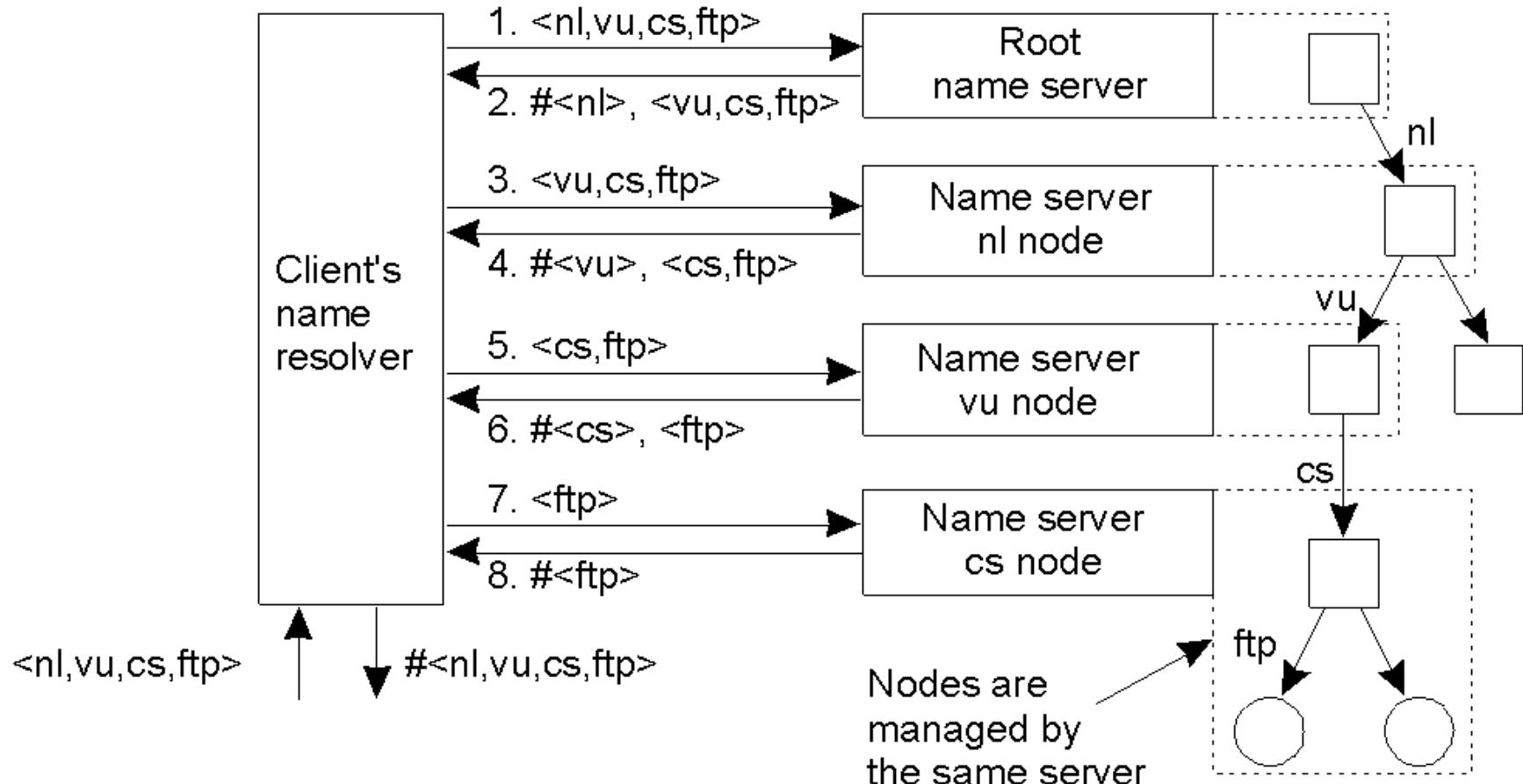
Implementation of Name Resolution

47

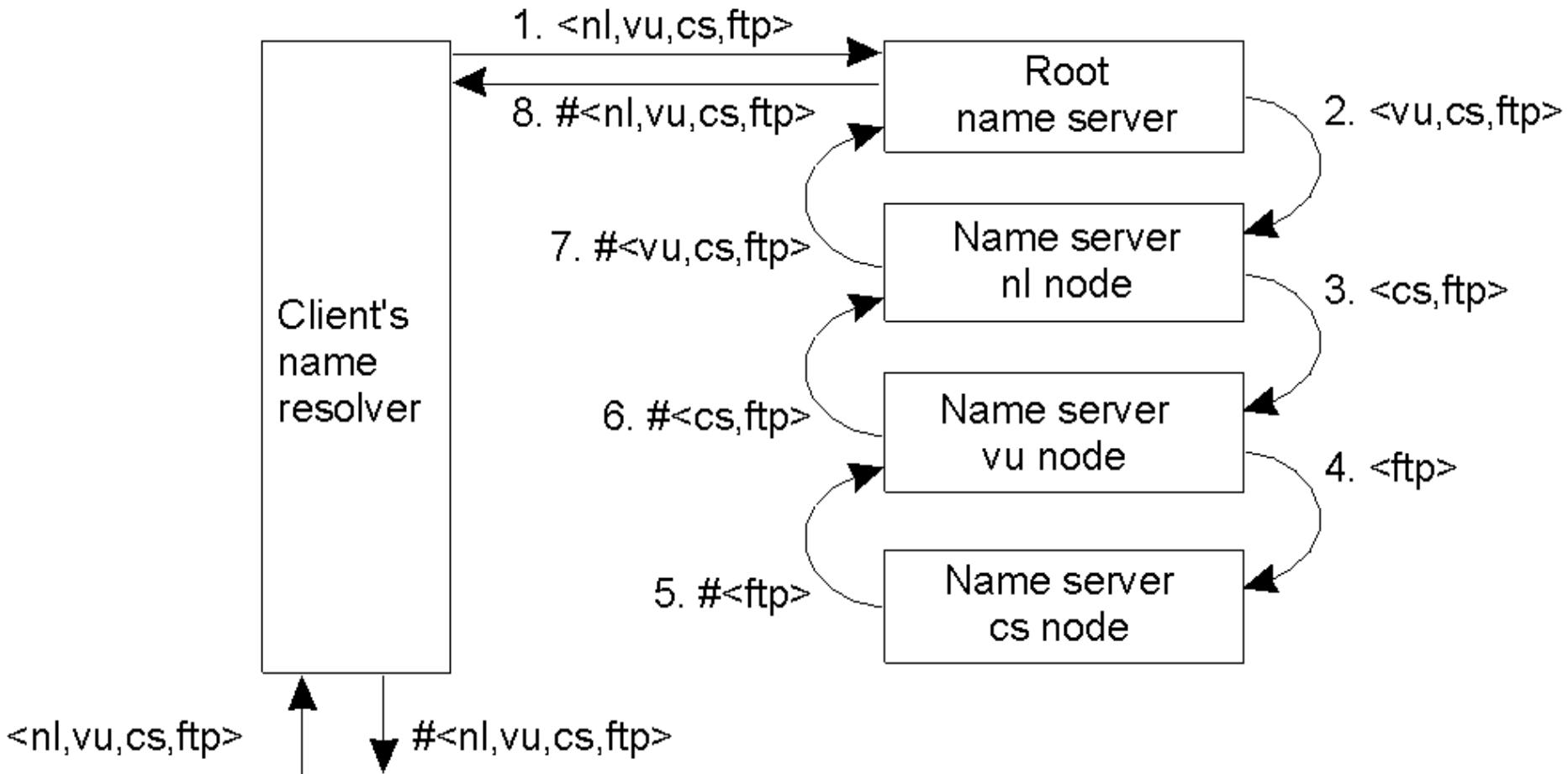
- Depend on the distribution of a name space across multiple name servers
- Each client has a name resolver
- 2 ways of implementation of name resolution:
 - ▣ Iterative name resolution
 - ▣ Recursive name resolution

Iterative name resolution

48

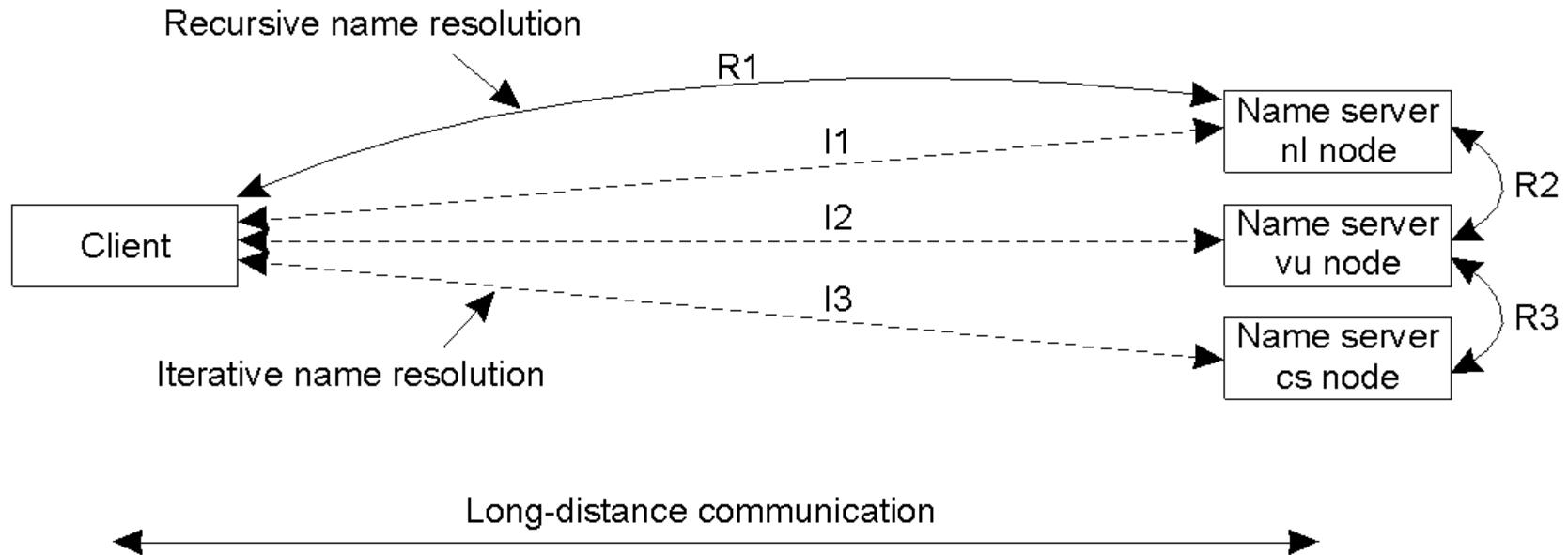


Recursive name resolution



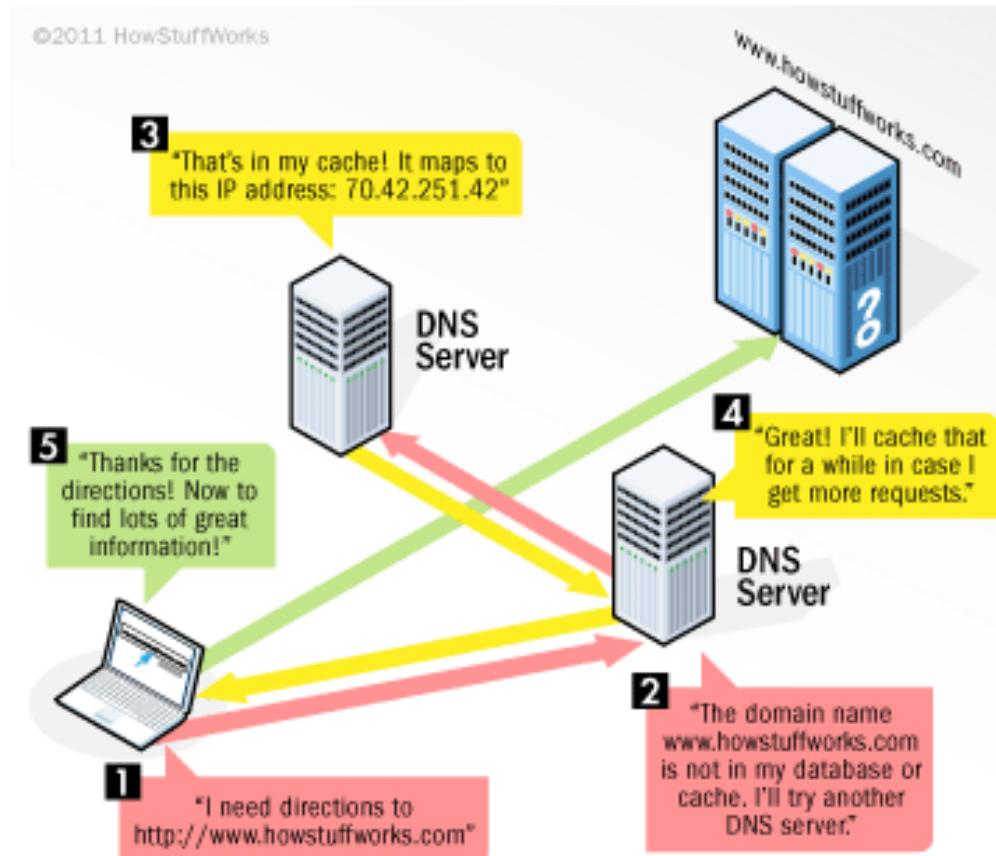
Recursive vs. iterative name resolution

50



Example: DNS

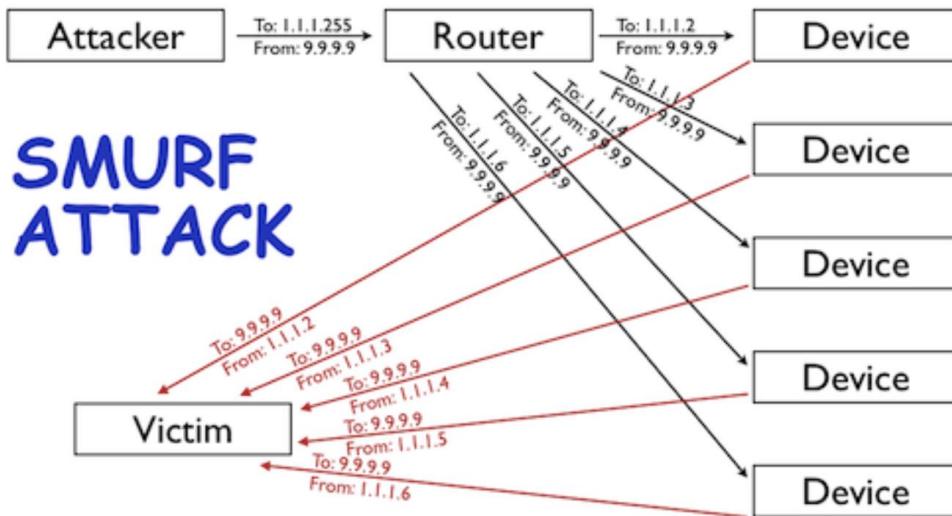
51



Security risk of recursive name resolution

52

Amplification attack



Problem: ICMP or UDP has no authentication mechanism

DNS Amplification attack

53

```
dig ANY isc.org @x.x.x.x
```

64 bytes query

3,223 byte response



DNS Terminology, Components, and Concepts

Top-Level Domain

Hosts

SubDomain

Fully Qualified Domain Name (FQDN)

Name Server

Zone File

Records

Record types

Start of Authority (SOA)

```
domain.com. IN SOA ns1.domain.com. admin.domain.com. (
                    12083      ; serial number
                    3h         ; refresh interval
                    30m       ; retry interval
                    3w         ; expiry period
                    1h         ; negative TTL
)
```

A and AAAA Records

```
host      IN      A          IPv4_address
host      IN      AAAA      IPv6_address
```

CNAME records

```
server1      IN  A        111.111.111.111
www          IN  CNAME    server1
```

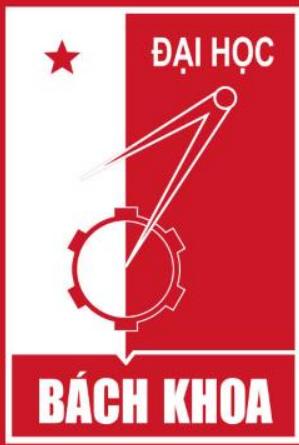
Record types

MX records

```
IN  MX  10  mail1.domain.com.  
IN  MX  50  mail2.domain.com.  
mail1  IN  A       111.111.111.111  
mail2  IN  A       222.222.222.222
```

NS records

```
IN  NS      ns1.domain.com.  
IN  NS      ns2.domain.com.  
ns1   IN  A       111.222.111.111  
ns2   IN  A       123.211.111.233
```



25 YEARS ANNIVERSARY
SIGHT

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

**DISTRIBUTES SYSTEMS AND
APPLICATIONS**



HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

CHAPTER 4: SYNCHRONIZATION

Outline

- Physical clock synchronization
- Logical clock synchronization
- Mutual exclusion algorithms
- Election algorithms

Introduction

- How process synchronize
 - Multiple process to not simultaneously access to the same resources: printers, files
 - Multiple process are agreed on the ordering of event.
 - Ex: message m1 of P is sent after m2 of Q
- Synchronization based on actual time
- Synchronization by relative ordering



HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

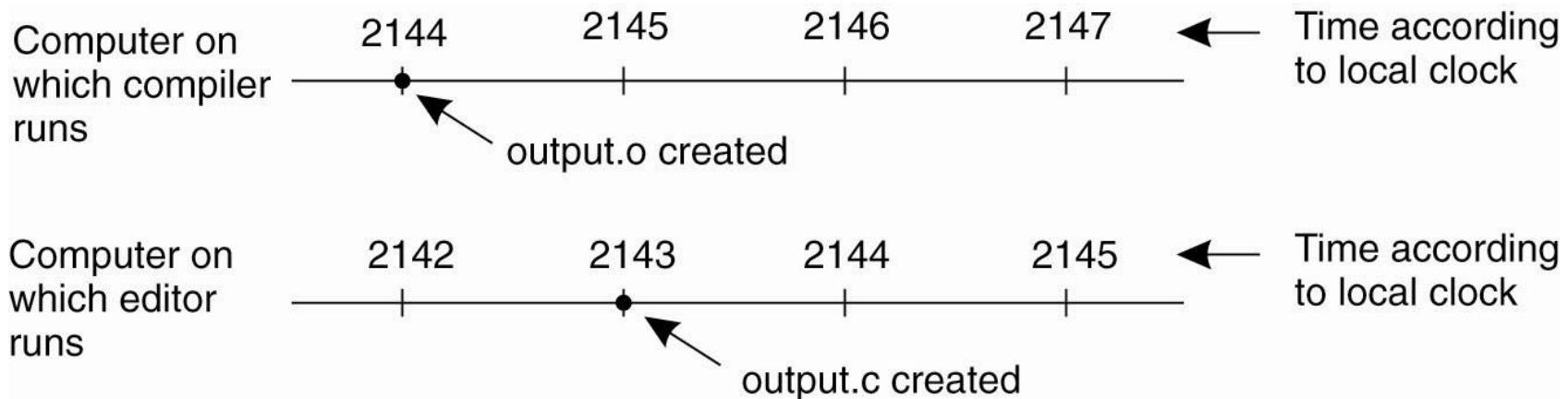
1. Physical clock Synchronization

1. Physical clock Synchronization

- Notion of synchronization
- Physical Clocks
- Global Positioning System
- Clock Synchronization Algorithms
- Use of Synchronized Clocks

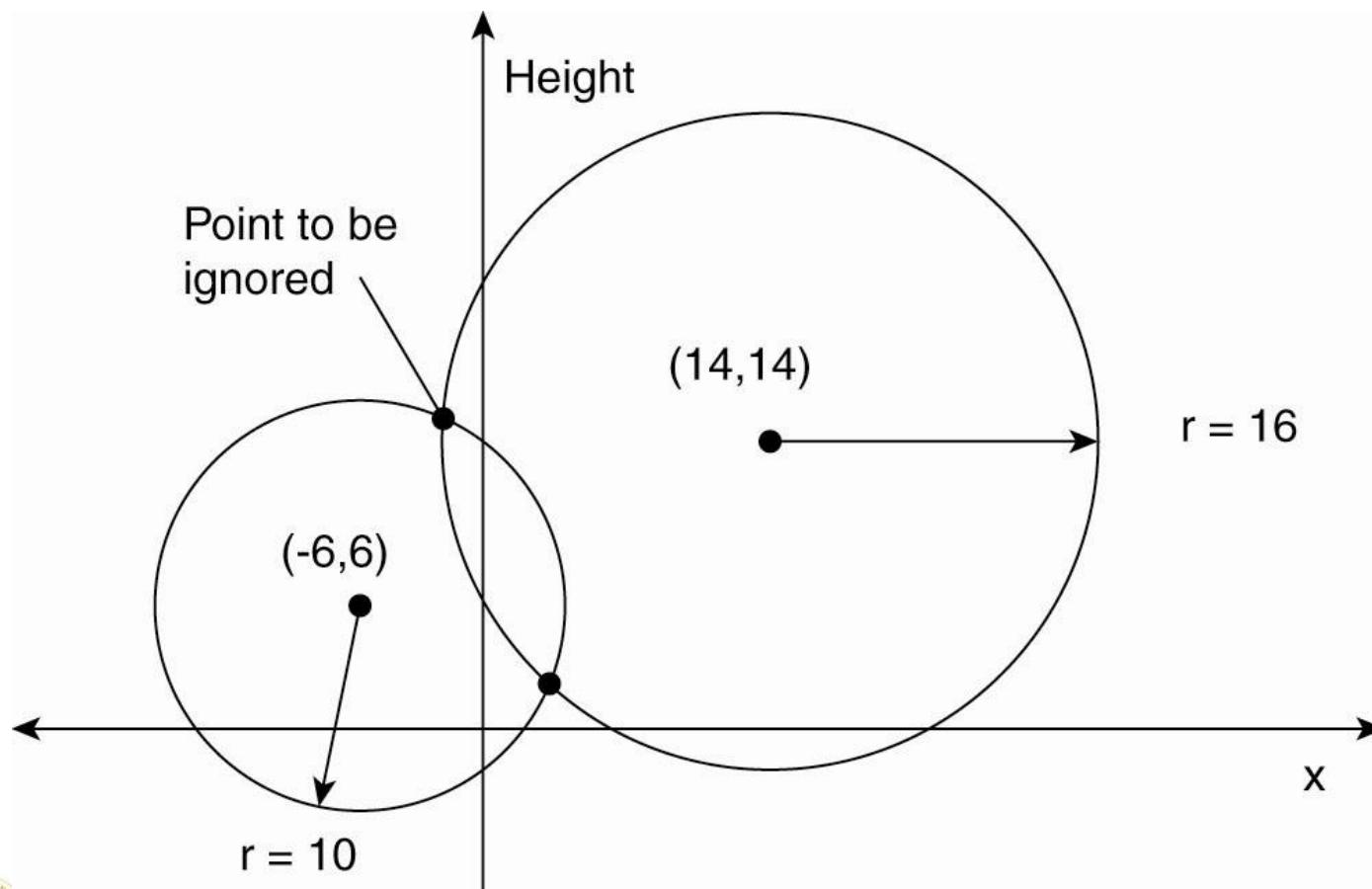
Why do we need it?

Example 1: Programming in DS



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

Example 2: Global Positioning System (1)



Global Positioning System (2)

- Real world facts that complicate GPS
 - 1. It takes a while before data on a satellite's position reaches the receiver.
 - 2. The receiver's clock is generally not in sync with that of a satellite.

Physical Clocks

- Timer
- Counter & Holding register
- Clock tick
- Problem in distributed systems:
 - How do we synchronize them with real-world?
 - How do we synchronize the clocks with each other?

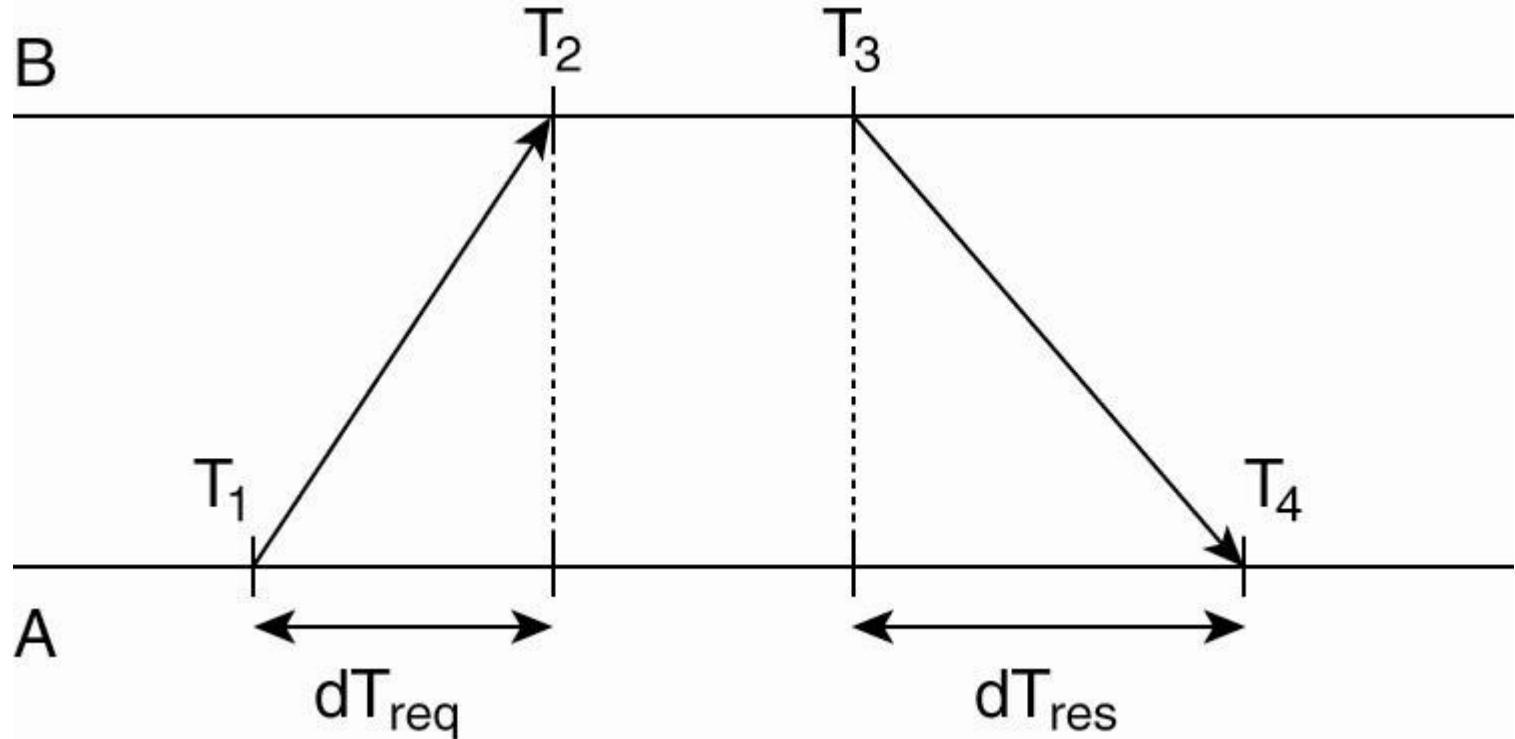


**RTC IC
(Real Time Clock)**

Physical Clock Synchronization Algorithms

- Network Time Protocol
- Berkeley Algorithm
- Clock Synchronization in Wireless Networks

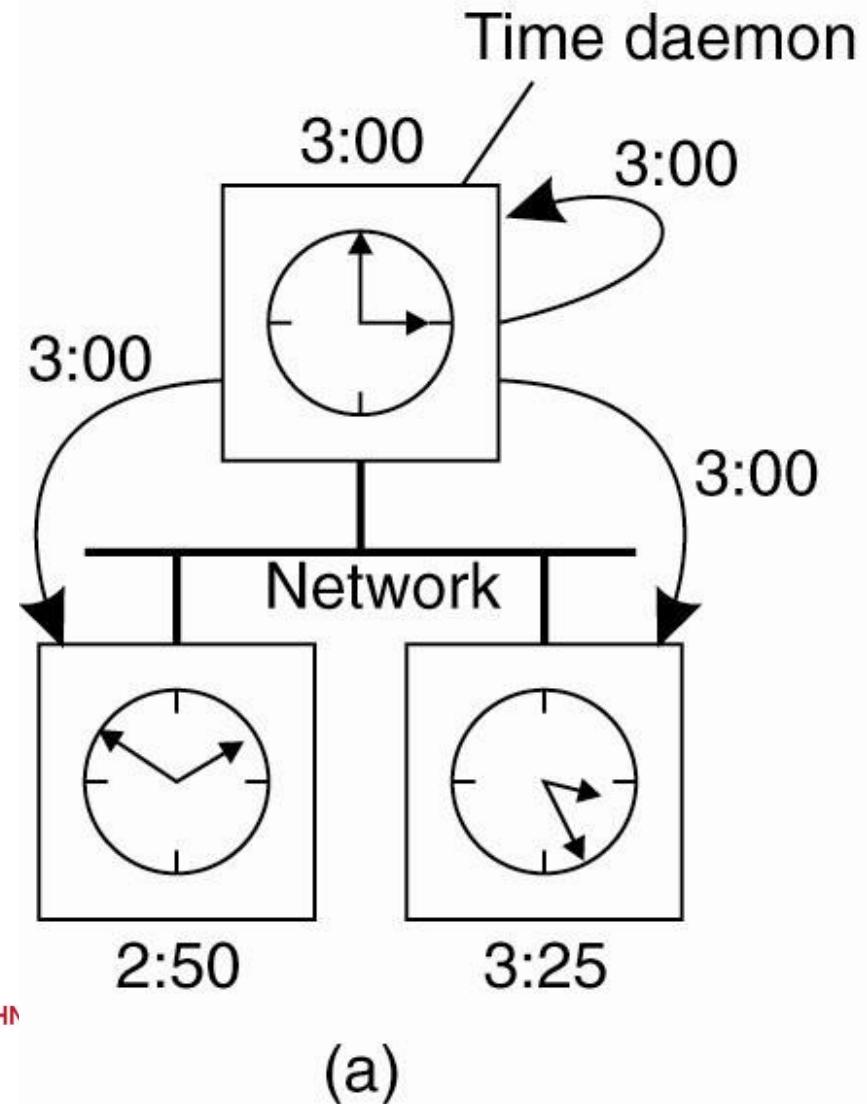
Network Time Protocol



$$\theta = T_3 + \frac{(T_2 - T_1) + (T_4 - T_3)}{2} - T_4 = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$$

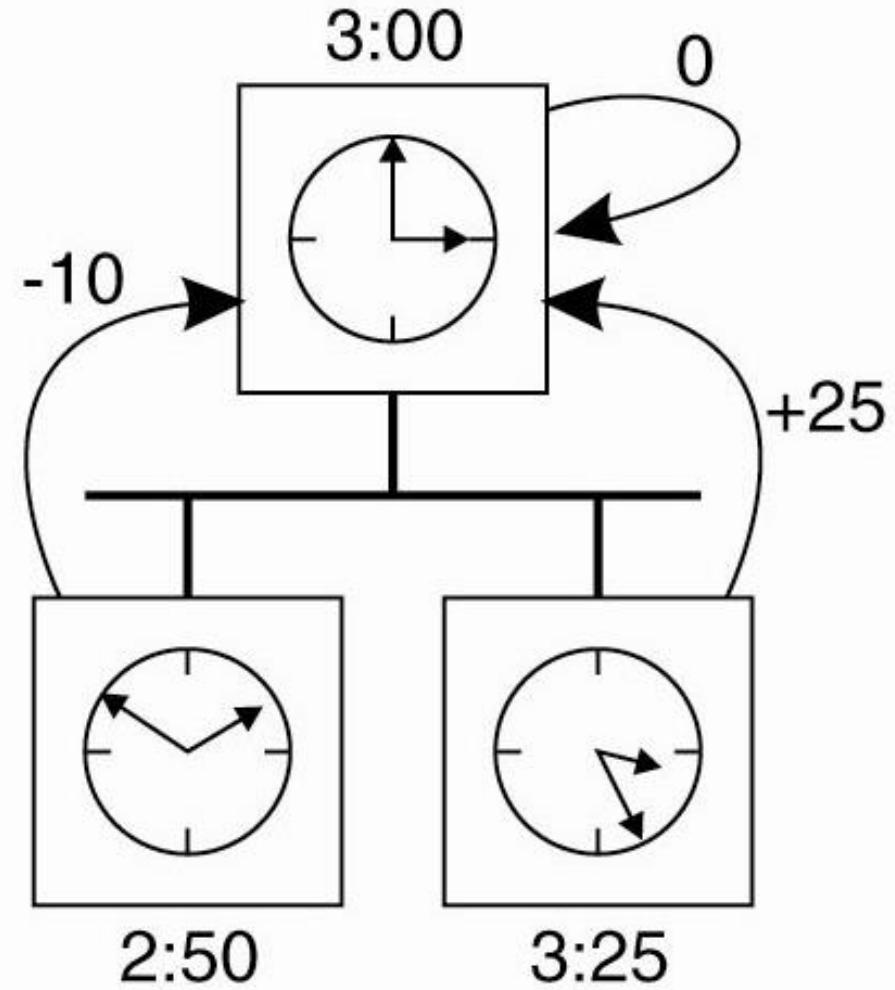
The Berkeley Algorithm (1)

- The time daemon asks all the other machines for their clock values.



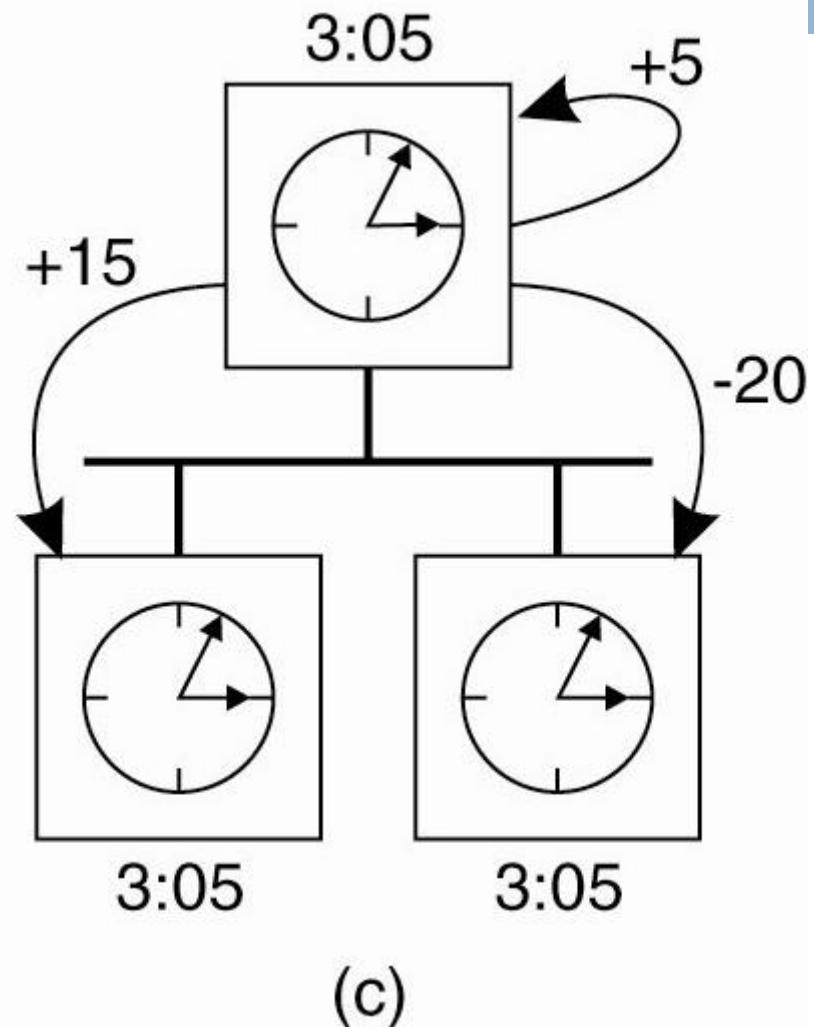
The Berkeley Algorithm (2)

- The machines answer.



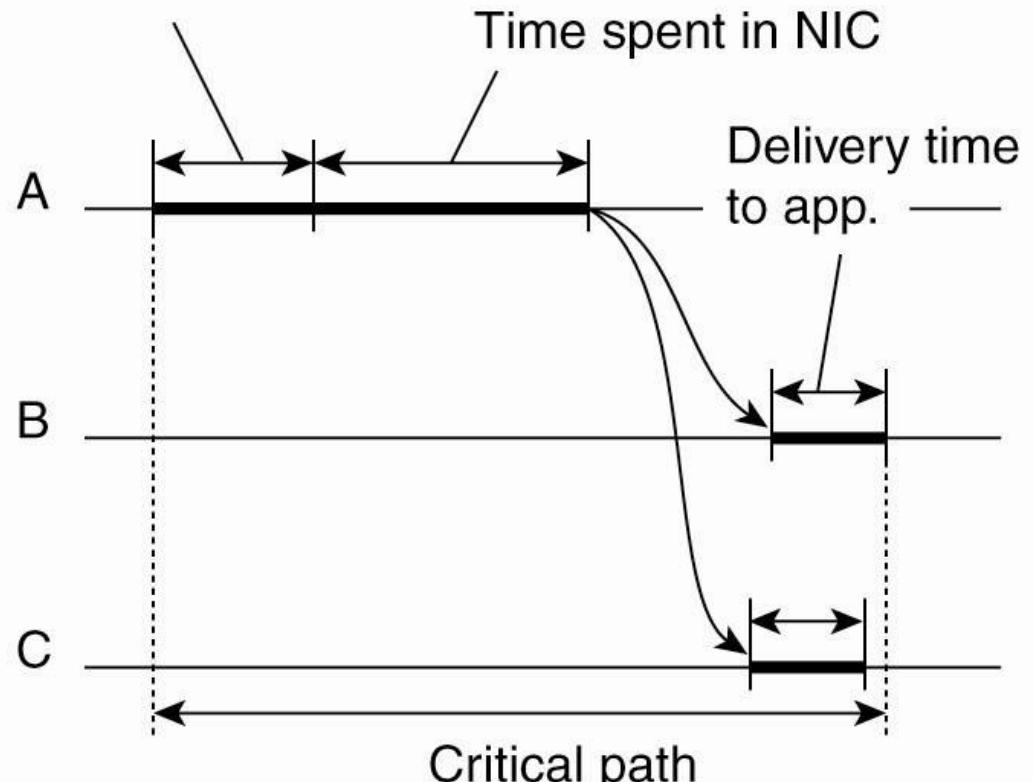
The Berkeley Algorithm (3)

- The time daemon tells everyone how to adjust their clock.



Clock Synchronization in Wireless Networks (1)

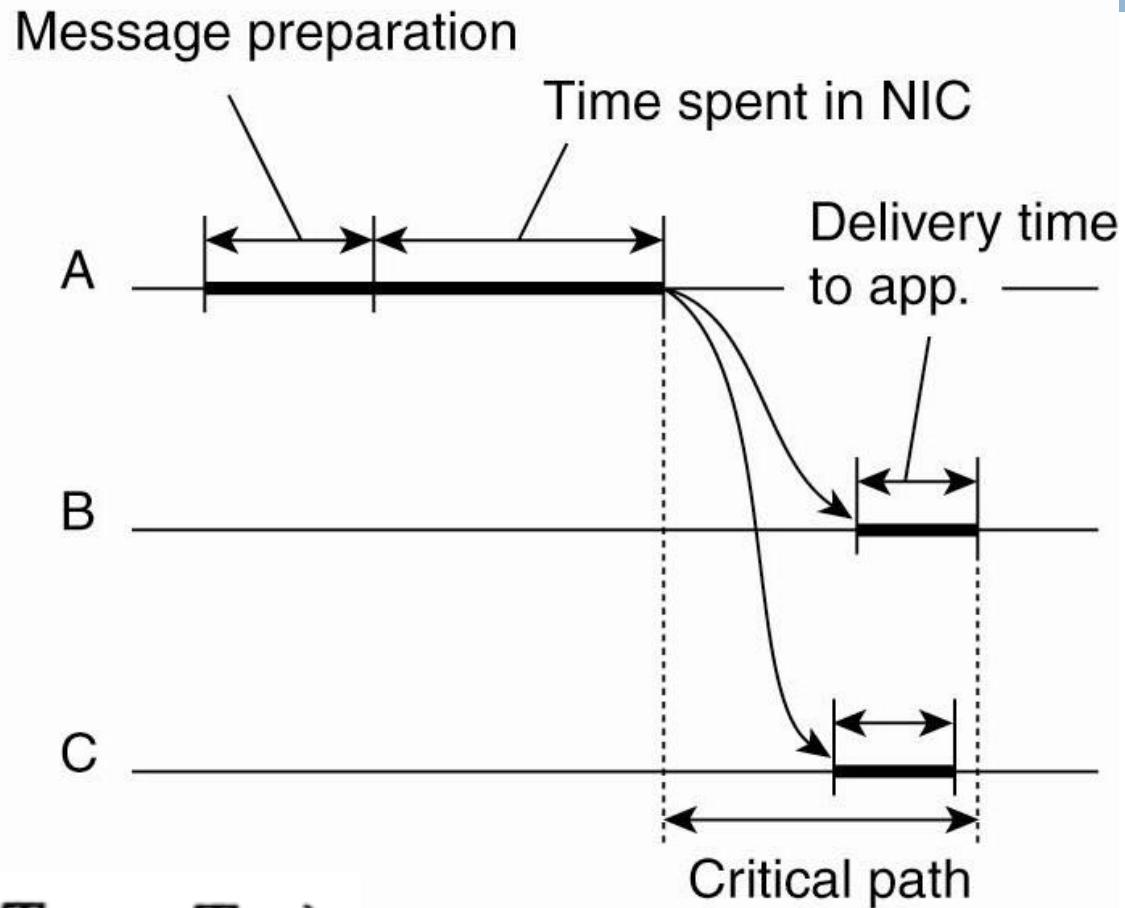
Message preparation



(a)

Clock Synchronization in Wireless Networks (2)

- The critical path in the case of RBS.



$$Offset[p, q] = \frac{\sum_{k=1}^M (T_{p,k} - T_{q,k})}{M}$$

(b)



HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

2. Logical clock synchronization

2. Logical clock synchronization

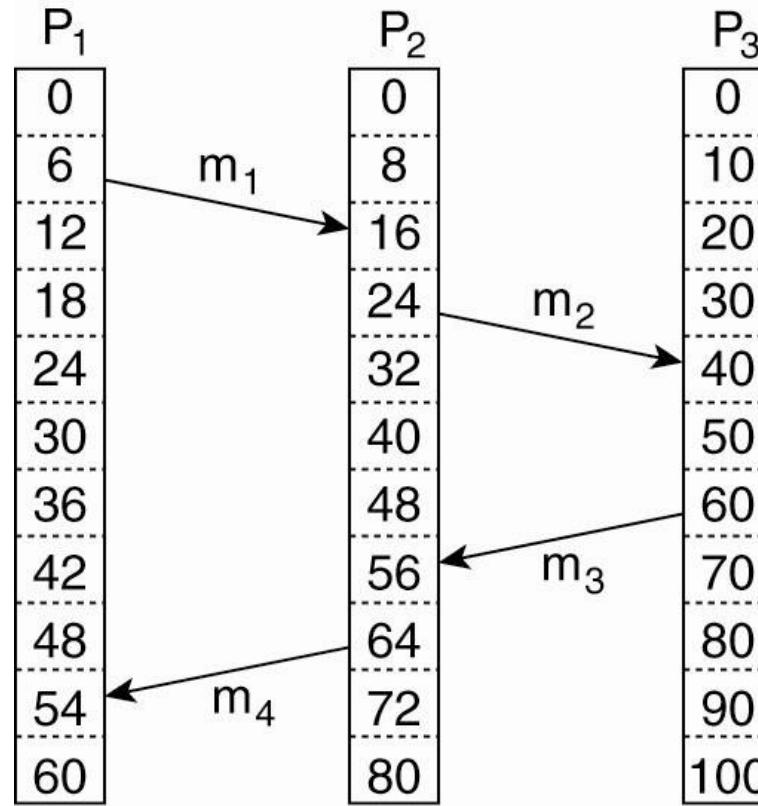
- Lamport logical clocks
- Vector clocks

2.1. Lamport's Logical Clocks (1)

- The "happens-before" relation \rightarrow can be observed directly in two situations:
 1. If a and b are events in the same process, and a occurs before b , then $a \rightarrow b$ is true.
 2. 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 $a \rightarrow b$
- Transitive relation: $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$
- Concurrent

Lamport's Logical Clocks (2)

- Three processes, each with its own clock.
The clocks run at different rates.

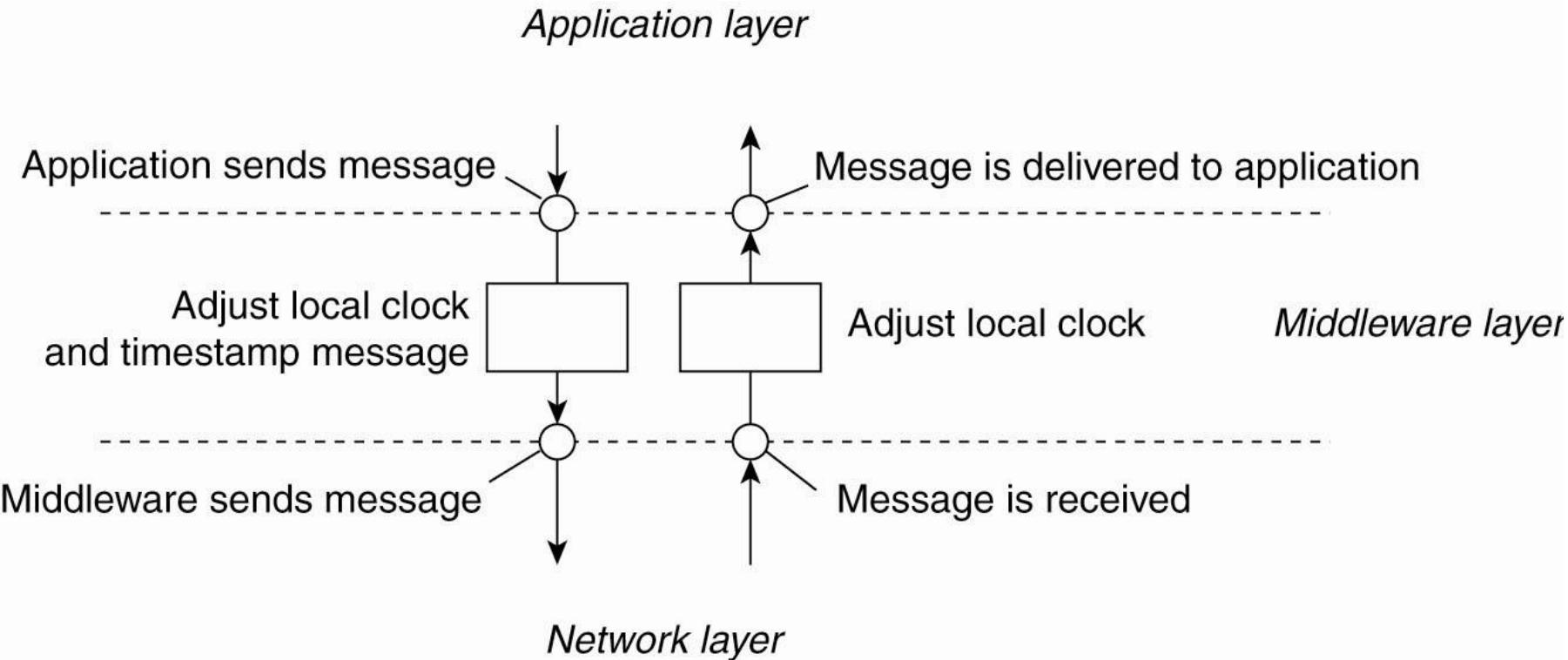


(a)

Lamport's Logical Clocks (3)

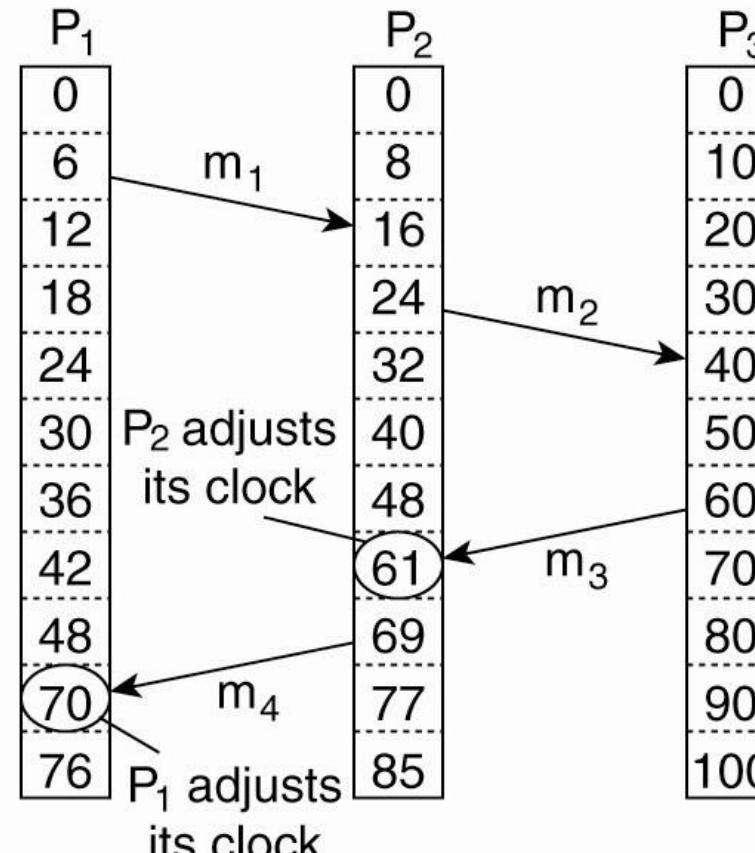
- Updating counter C_i for process P_i
 1. Before executing an event P_i executes
$$C_i \leftarrow C_i + 1.$$
 2. When process P_i sends a message m to P_j , it sets m' 's timestamp $ts(m)$ equal to C_i after having executed the previous step.
 3. Upon the receipt of a message m , process P_j adjusts 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.

Lamport's Logical Clocks (4)



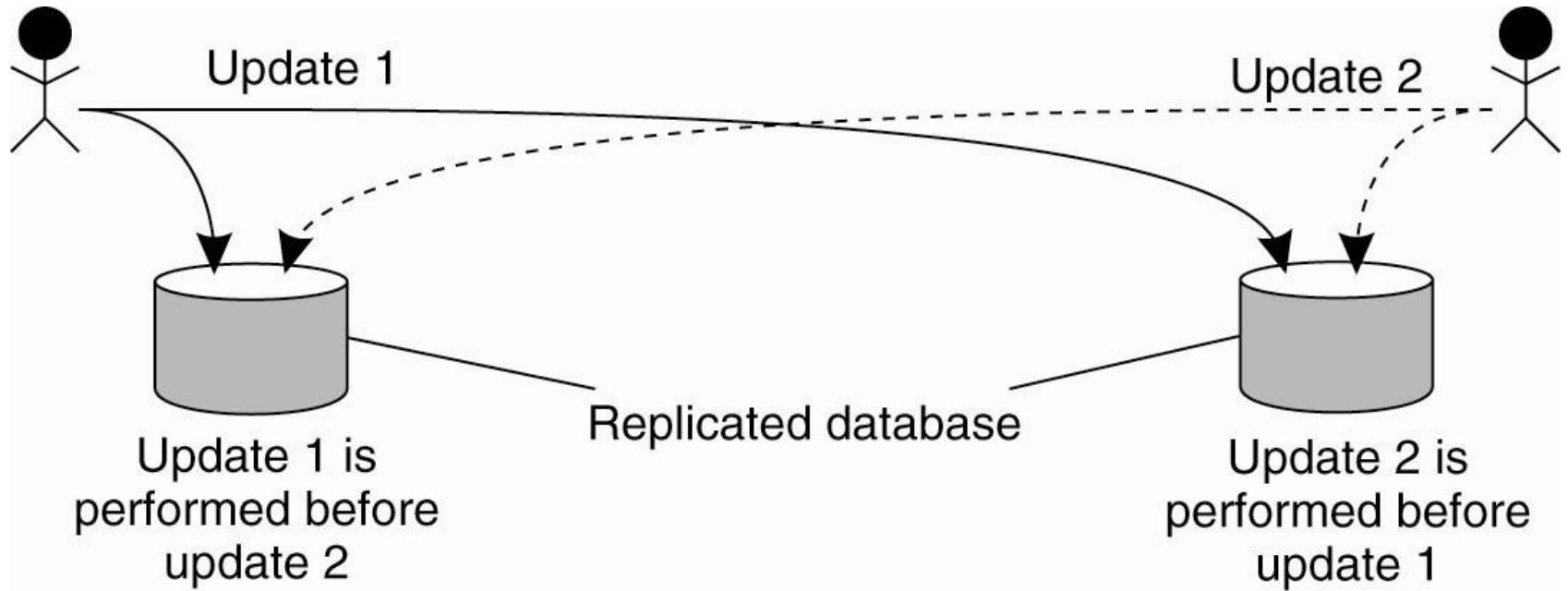
Lamport's Logical Clocks (5)

(b) Lamport's algorithm corrects the clocks.



(b)

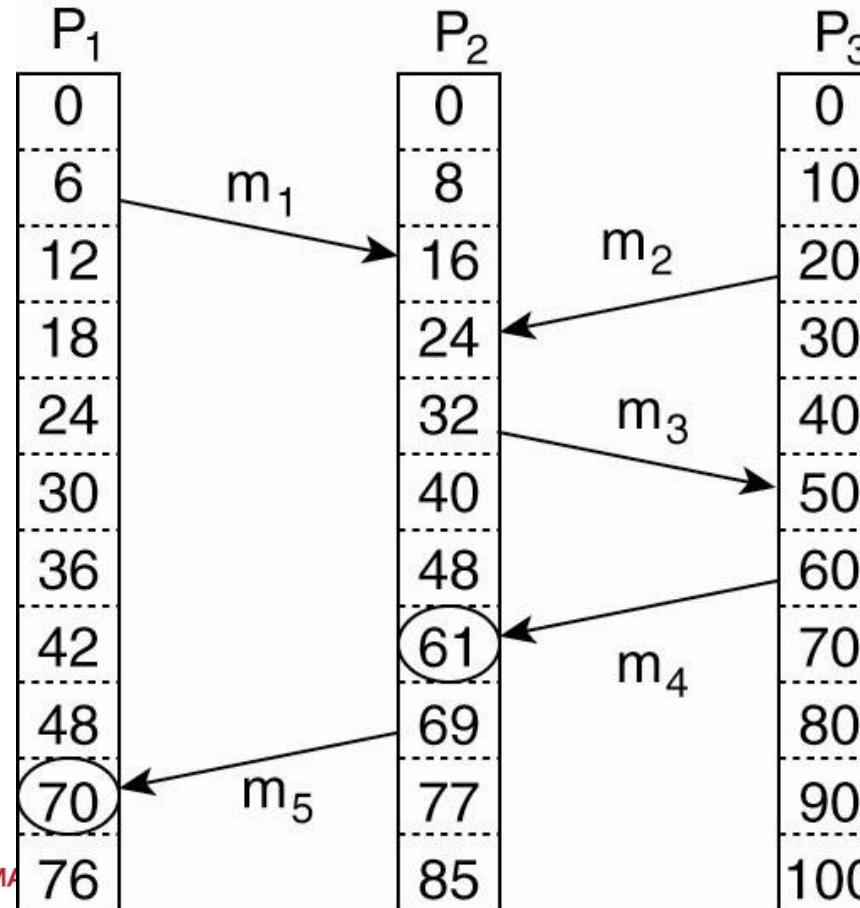
Example: Totally Ordered Multicasting



Updating a replicated database and leaving it in an inconsistent state.

2.2. Vector Clocks (1)

- Concurrent message transmission using logical clocks.



Vector Clocks (2)

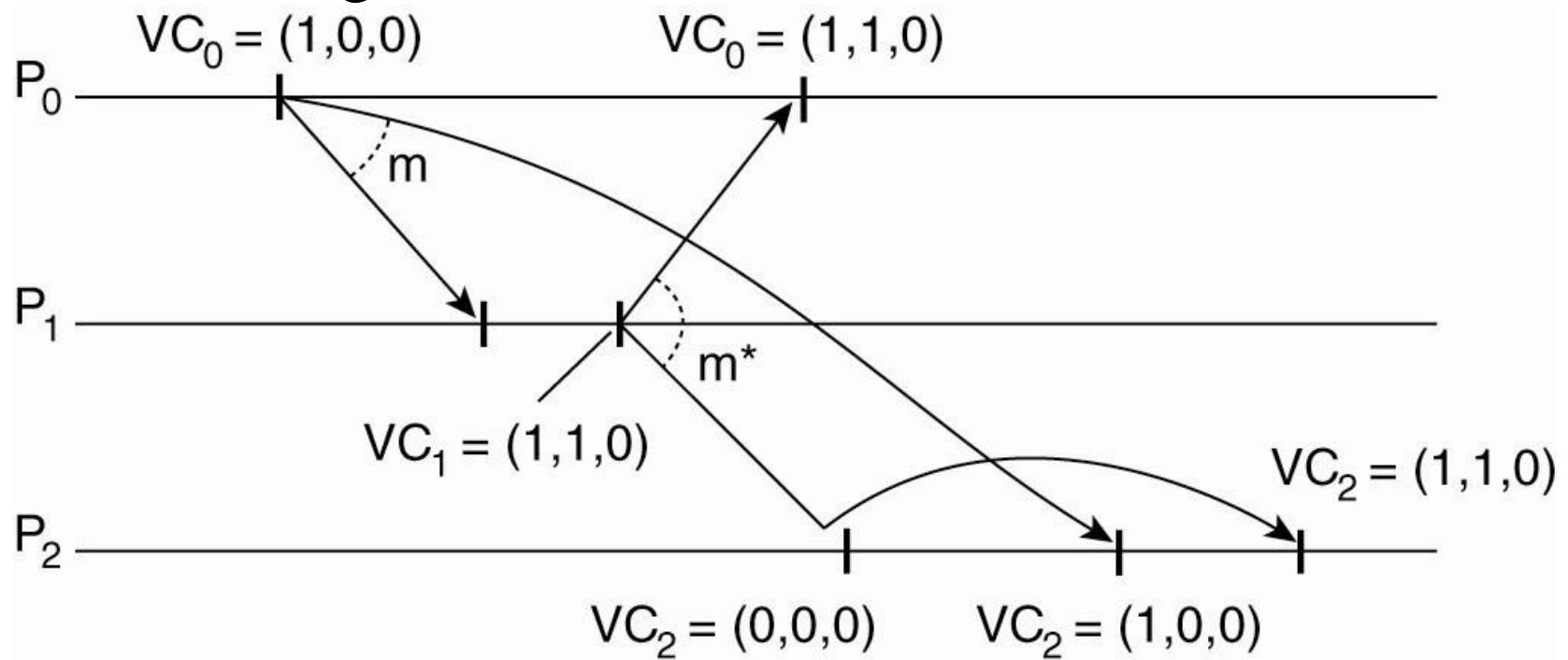
- Vector clocks are constructed by letting each process P_i maintain a vector VC_i with the following two properties:
 1. $VC_i [i]$ is the number of events that have occurred so far at P_i . In other words, $VC_i [i]$ is the local logical clock at process P_i .
 2. If $VC_i [j] = k$ then P_i knows that k events have occurred at P_j . It is thus P_i 's knowledge of the local time at P_j .

Vector Clocks (3)

- Steps carried out to accomplish property 2 of previous slide:
 1. Before executing an event P_i executes
$$VC_i[i] \leftarrow VC_i[i] + 1.$$
 2. When process P_i sends a message m to P_j , it sets m 's (vector) timestamp $ts(m)$ equal to VC_i after having executed the previous step.
 3. Upon the receipt of a message m , process P_j adjusts its own vector by setting
$$VC_j[k] \leftarrow \max\{VC_j[k], ts(m)[k]\}$$
 for each k , after which it executes the first step and delivers the message to the application.

Enforcing Causal Communication

- Enforcing causal communication.





HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

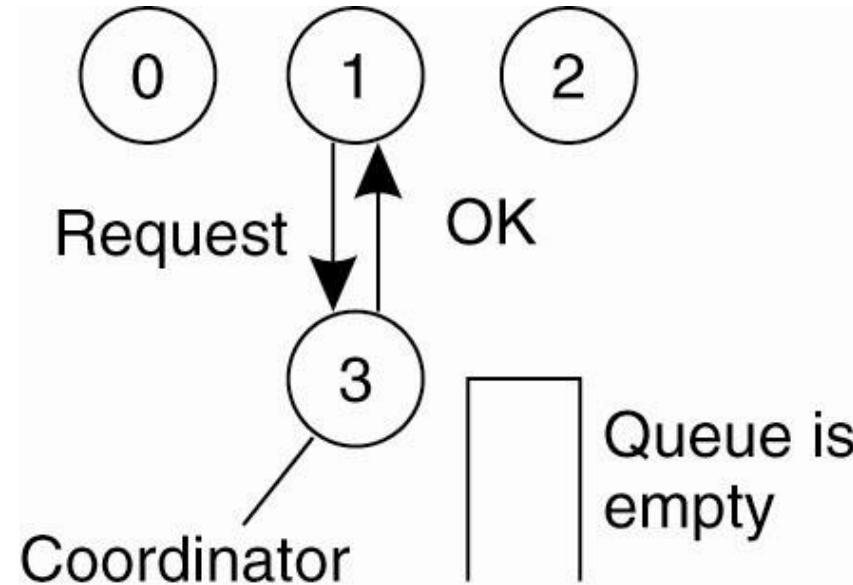
3. Mutual exclusion

3. Mutual exclusion algorithms

- Classification of Mutual exclusion algorithms:
 - Permission-based approach
 - A Centralized Algorithm
 - A Distributed Algorithm
 - Token-based solutions
 - Token Ring Algorithm
- Problems:
 - Starvation
 - Deadlocks
 - Token loss (for token-based approach)

Mutual Exclusion

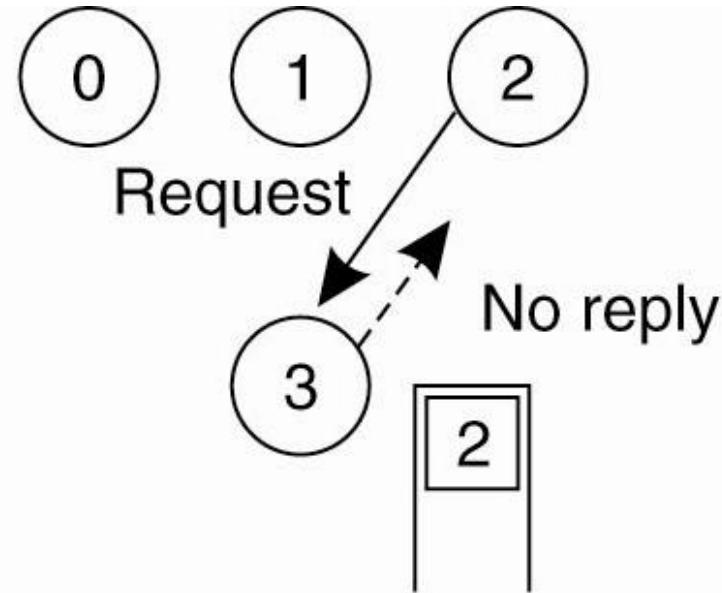
3.1. Centralized Algorithm (1)



(a)

- Process 1 asks the coordinator for permission to access a shared resource. Permission is granted.

Mutual Exclusion Centralized Algorithm (2)

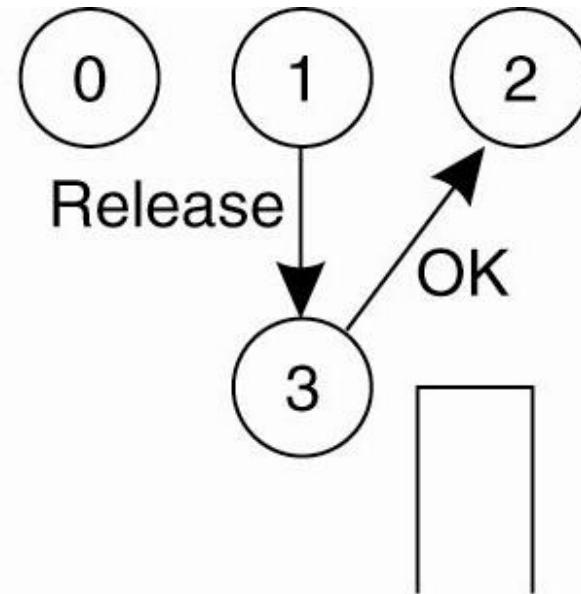


(b)

- Process 2 then asks permission to access the same resource. The coordinator does not reply.

Mutual Exclusion Centralized Algorithm (3)

- When process 1 releases the resource, it tells the coordinator, which then replies to 2.



(c)

Algorithm analysis

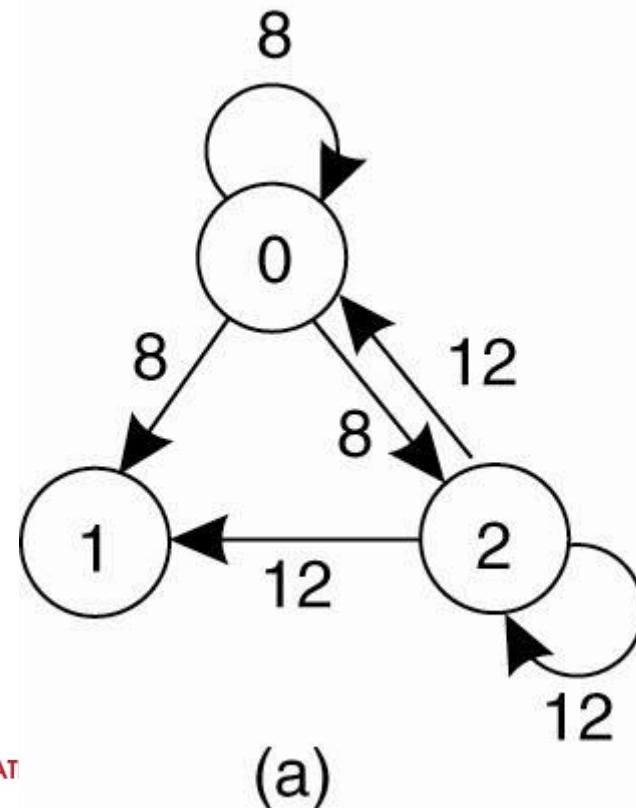
- Advantages
 - ▣ Simplicity
 - ▣ Resolve both *starvation* and *deadlocks* problems
- Disadvantages
 - ▣ Single point of failure
 - ▣ Performance bottleneck

3.2. 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.

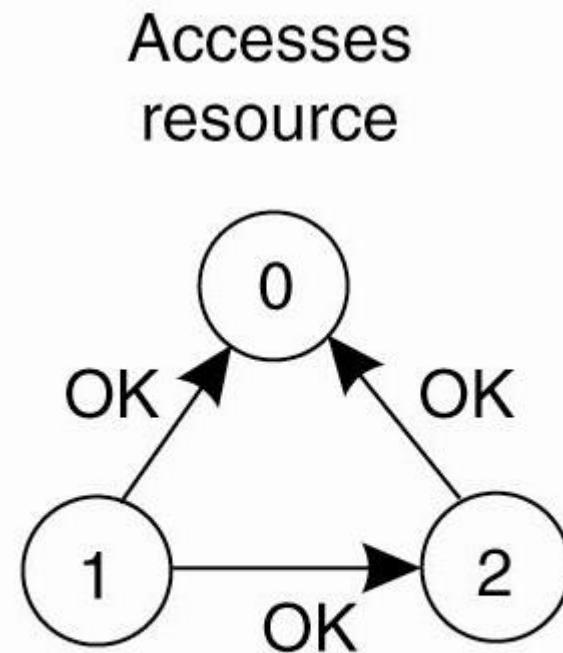
A Distributed Algorithm (2)

- Two processes want to access a shared resource at the same moment.



A Distributed Algorithm (3)

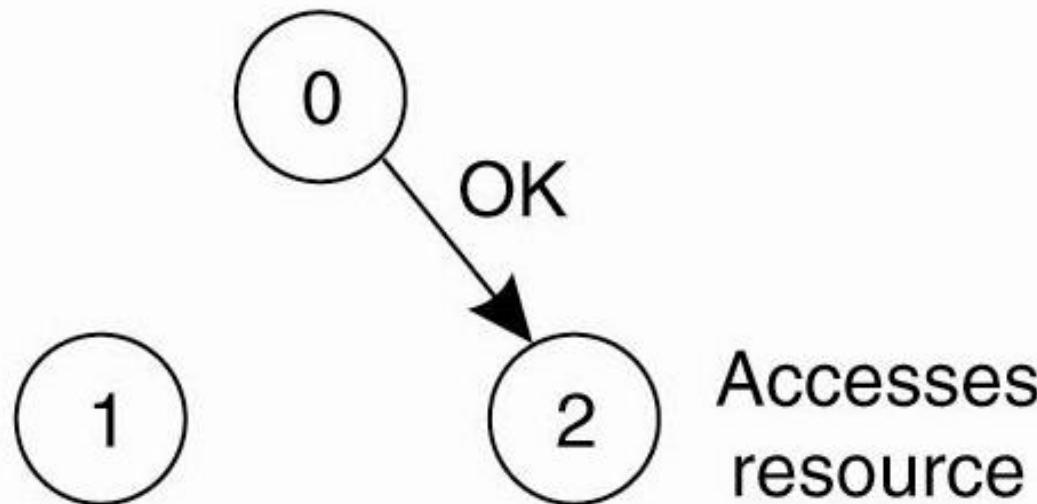
- Process 0 has the lowest timestamp, so it wins.



(b)

A Distributed Algorithm (4)

- When process 0 is done, it sends an OK also, so 2 can now go ahead.

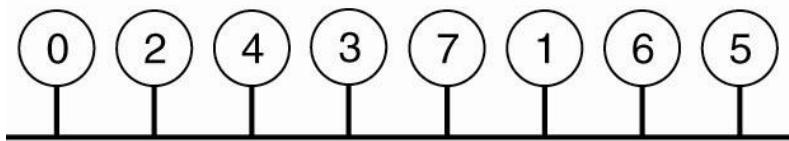


(c)

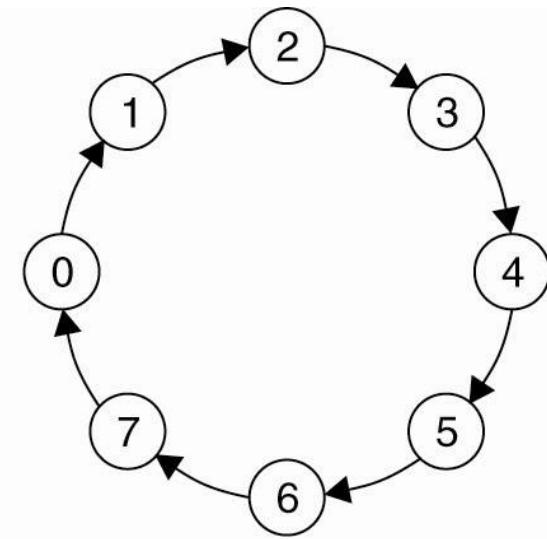
Algorithm analysis

- Advantages
 - ▣ Avoidance of deadlock and starvation problems
- Disadvantages:
 - ▣ Complicated
 - ▣ n points of failure
 - ▣ Broadcasting mechanism → scalability
 - ▣ More expensive, but less robust than the centralized algo

3.3. A Token Ring Algorithm



(a)

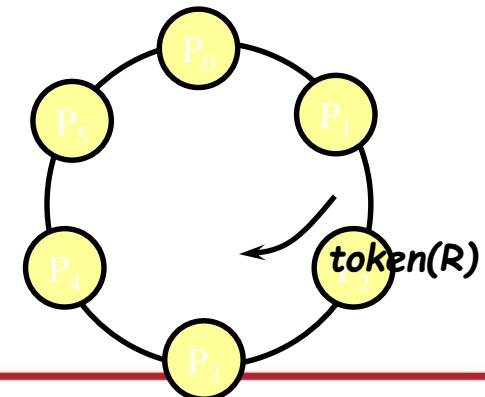


(b)

- (a) An unordered group of processes on a network.
(b) A logical ring constructed in software.

Token Ring algorithm

- Initialization
 - Process 0 gets token for resource R
- Token circulates around ring
 - From P_i to $P_{(i+1) \bmod N}$
- When process acquires token
 - Checks to see if it needs to enter critical section
 - If no, send token to neighbor
 - If yes, access resource
 - Hold token until done



Algorithm analysis

- Advantages
 - Avoidance of deadlock and starvation problems
- Disadvantages:
 - Token loss



HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

4. Election algorithms

4. Election Algorithms

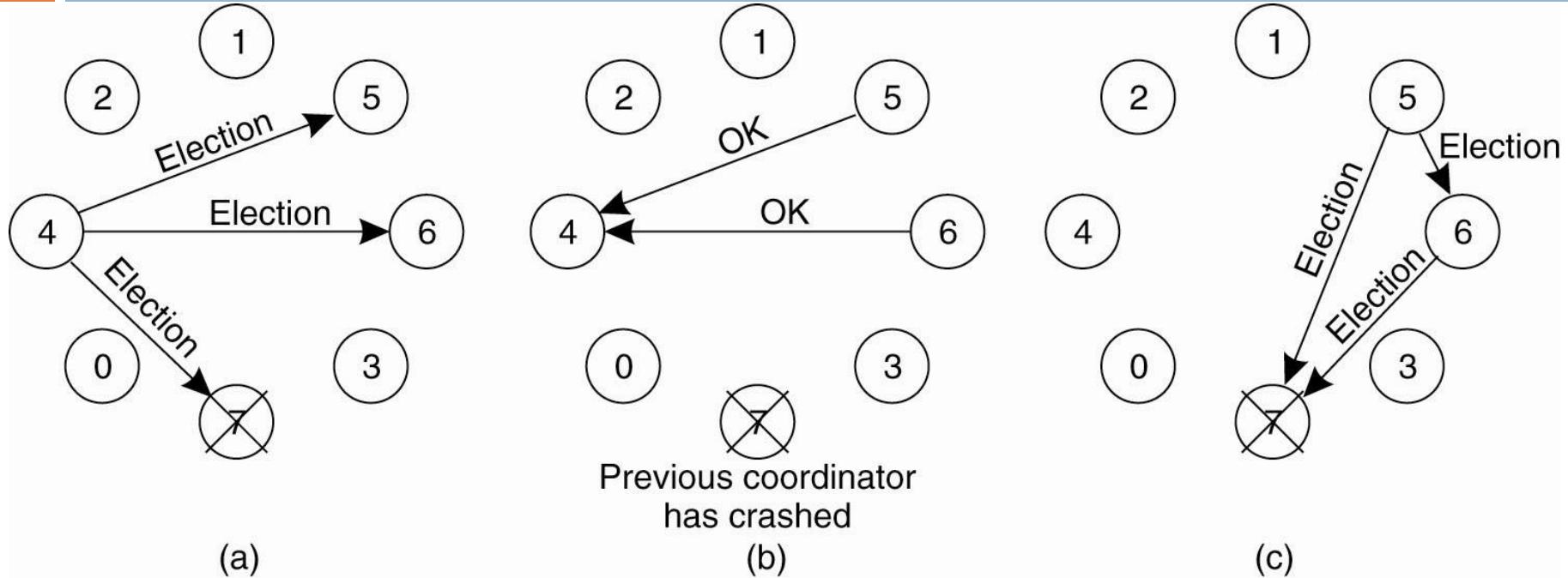
- Traditional Election algorithms
 - The Bully Algorithm
 - A Ring Algorithm
- Election in Wireless Environments
- Election in Large-Scale Systems

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.

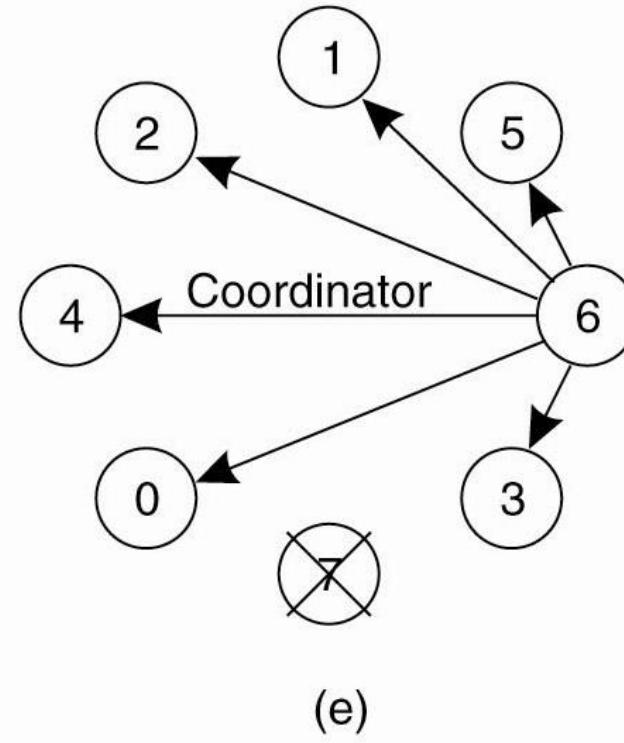
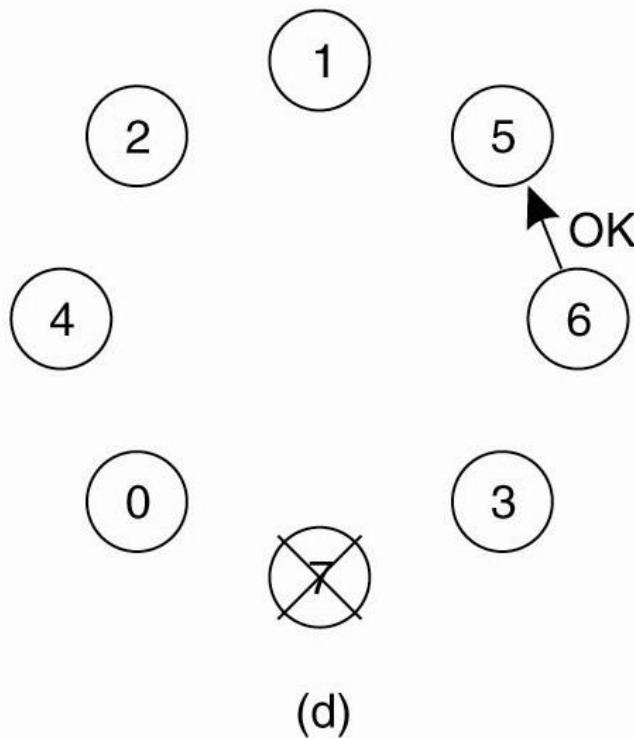
The Bully Algorithm (1)



- 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.

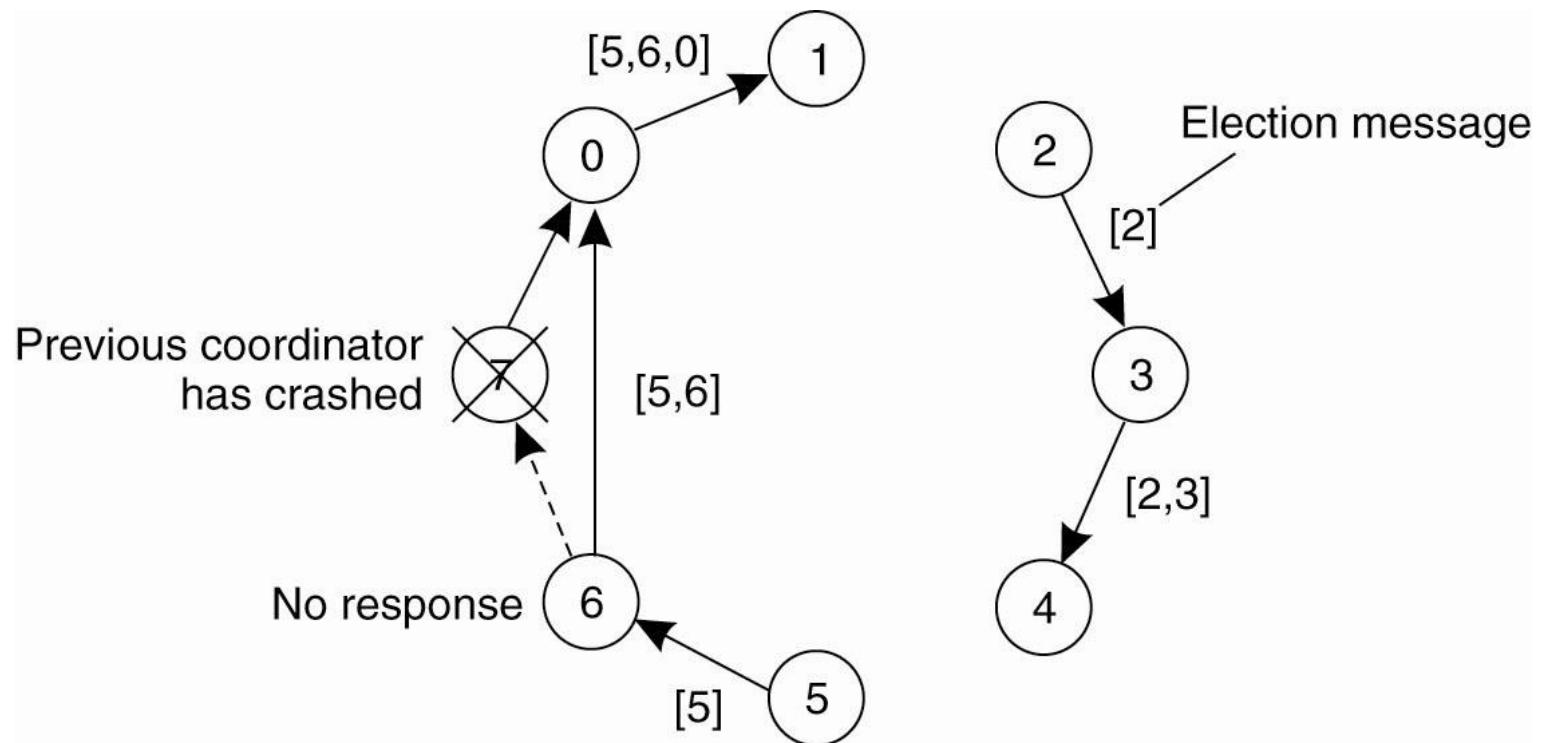
The Bully Algorithm (2)

- The bully election algorithm. (d) Process 6 tells 5

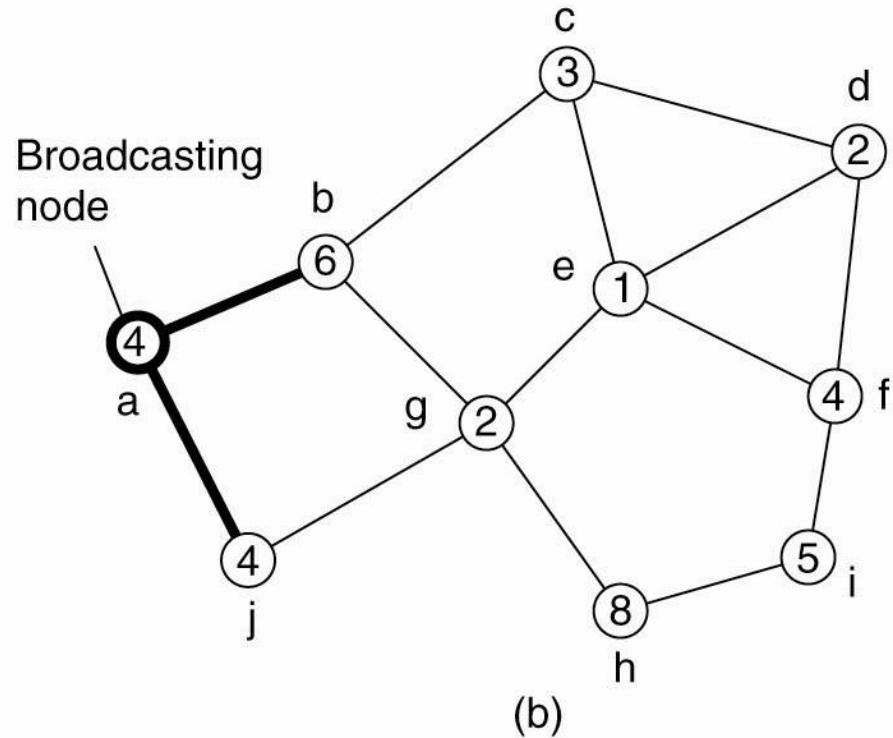
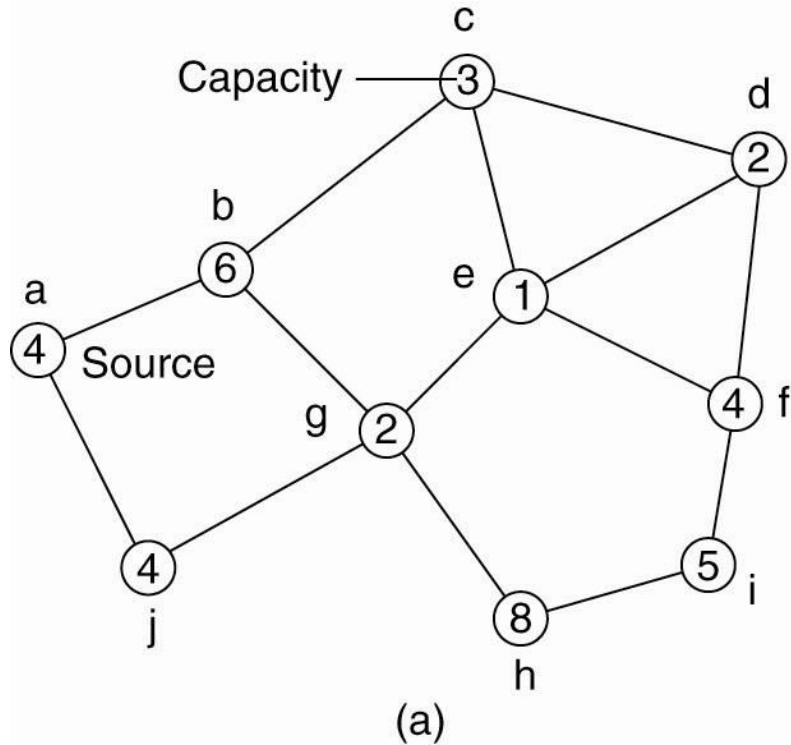


A Ring Algorithm

- Election algorithm using a ring.

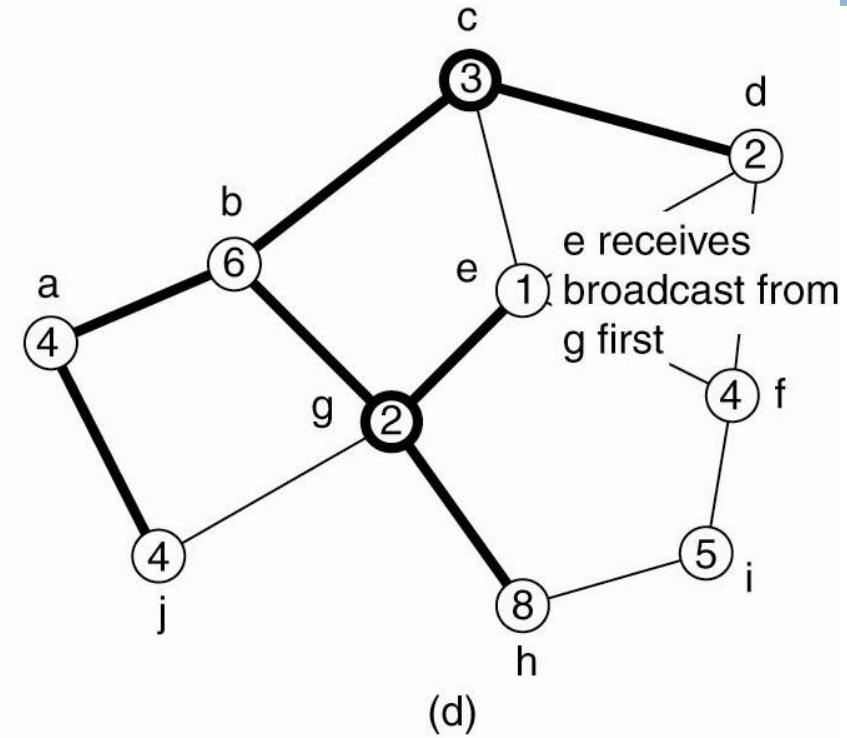
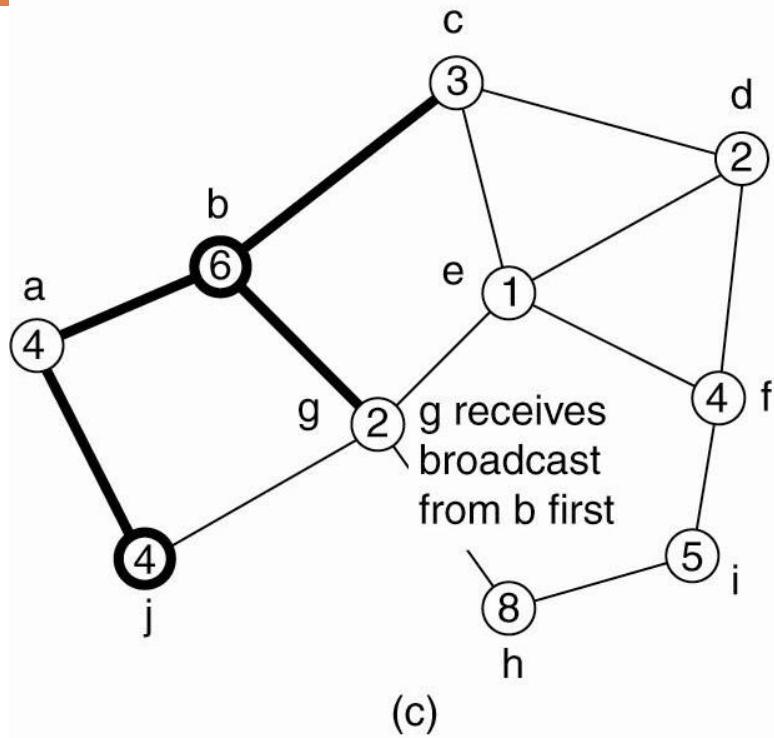


Elections in Wireless Environments (1)



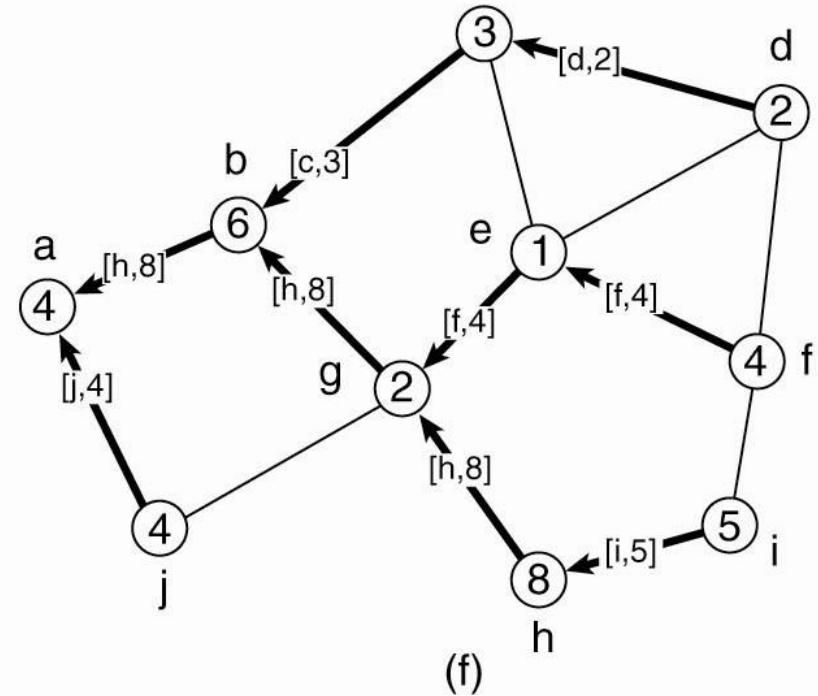
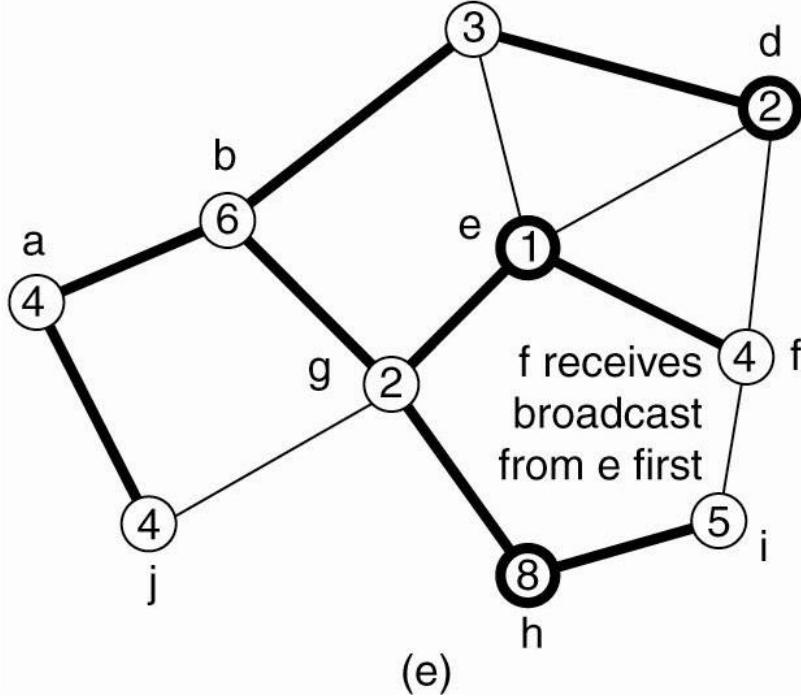
- Election algorithm in a wireless network, with node a as the source. (a) Initial network. (b)–(e) The build-tree phase

Elections in Wireless Environments (2)



Elections in Wireless Environments (3)

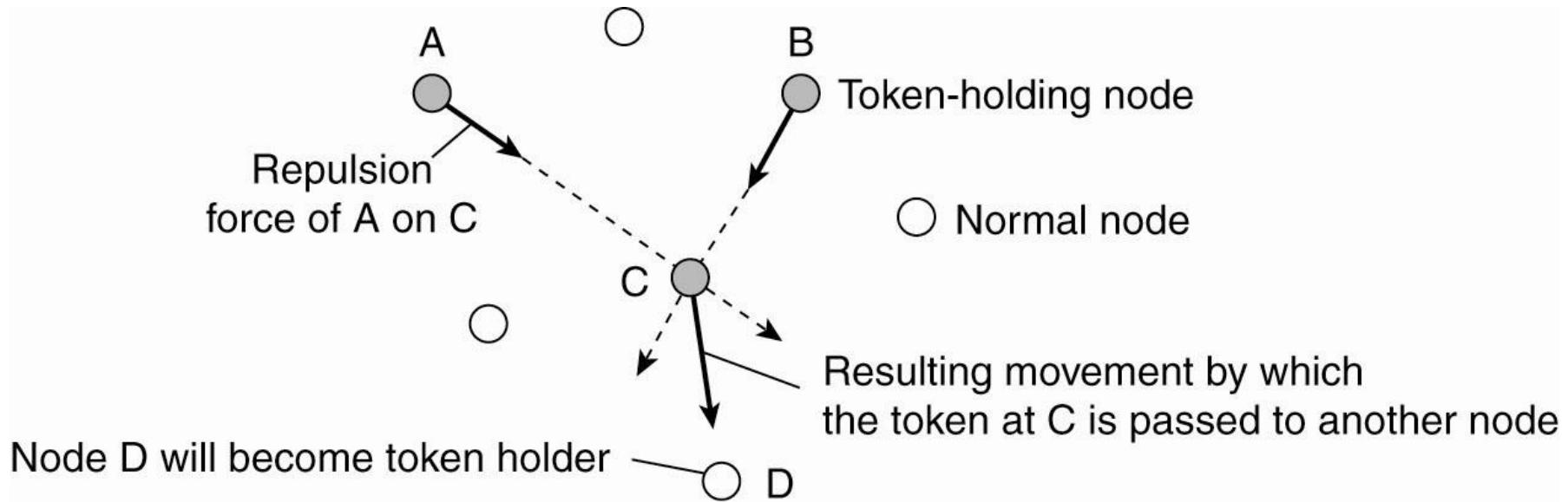
- (e) The build-tree phase.
- (f) Reporting of best node to source.



Elections in Large-Scale Systems (1)

- Requirements for superpeer selection:
 1. Normal nodes should have low-latency access to superpeers.
 2. Superpeers should be evenly distributed across the overlay network.
 3. There should be a predefined portion of superpeers relative to the total number of nodes in the overlay network.
 4. Each superpeer should not need to serve more than a fixed number of normal nodes.

Elections in Large-Scale Systems (2)



- Moving tokens in a two-dimensional space using repulsion forces.

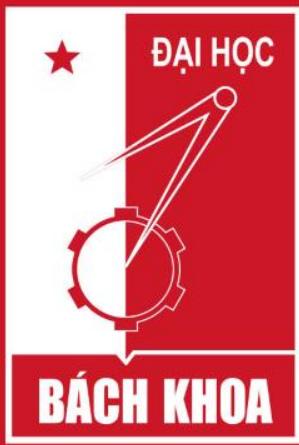


25
YEARS ANNIVERSARY
SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Questions ?





25 YEARS ANNIVERSARY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

CÁC HỆ THỐNG PHÂN TÁN VÀ ỨNG DỤNG

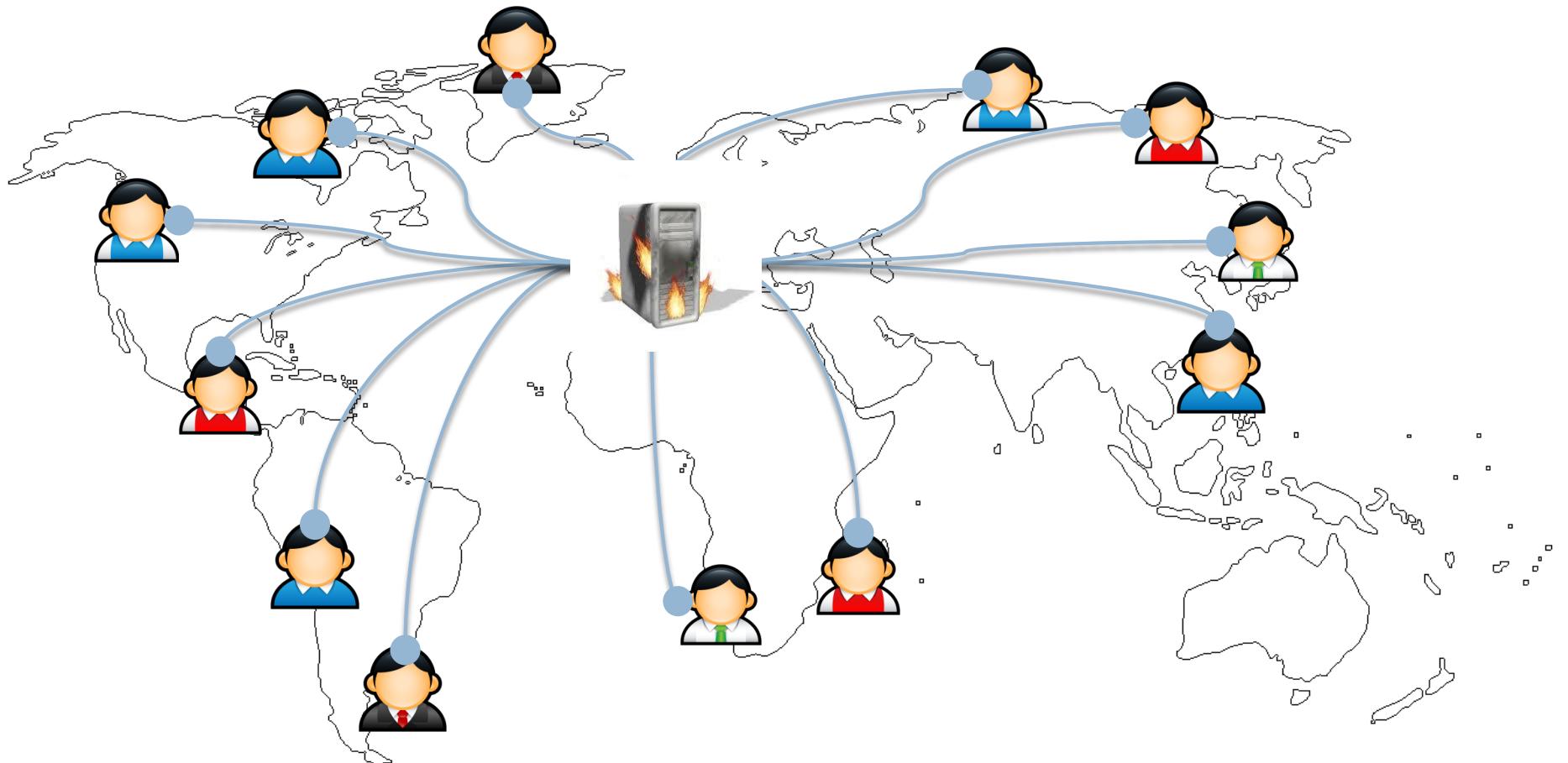


HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Chapter 5: Consistency and Replication

Problems

3

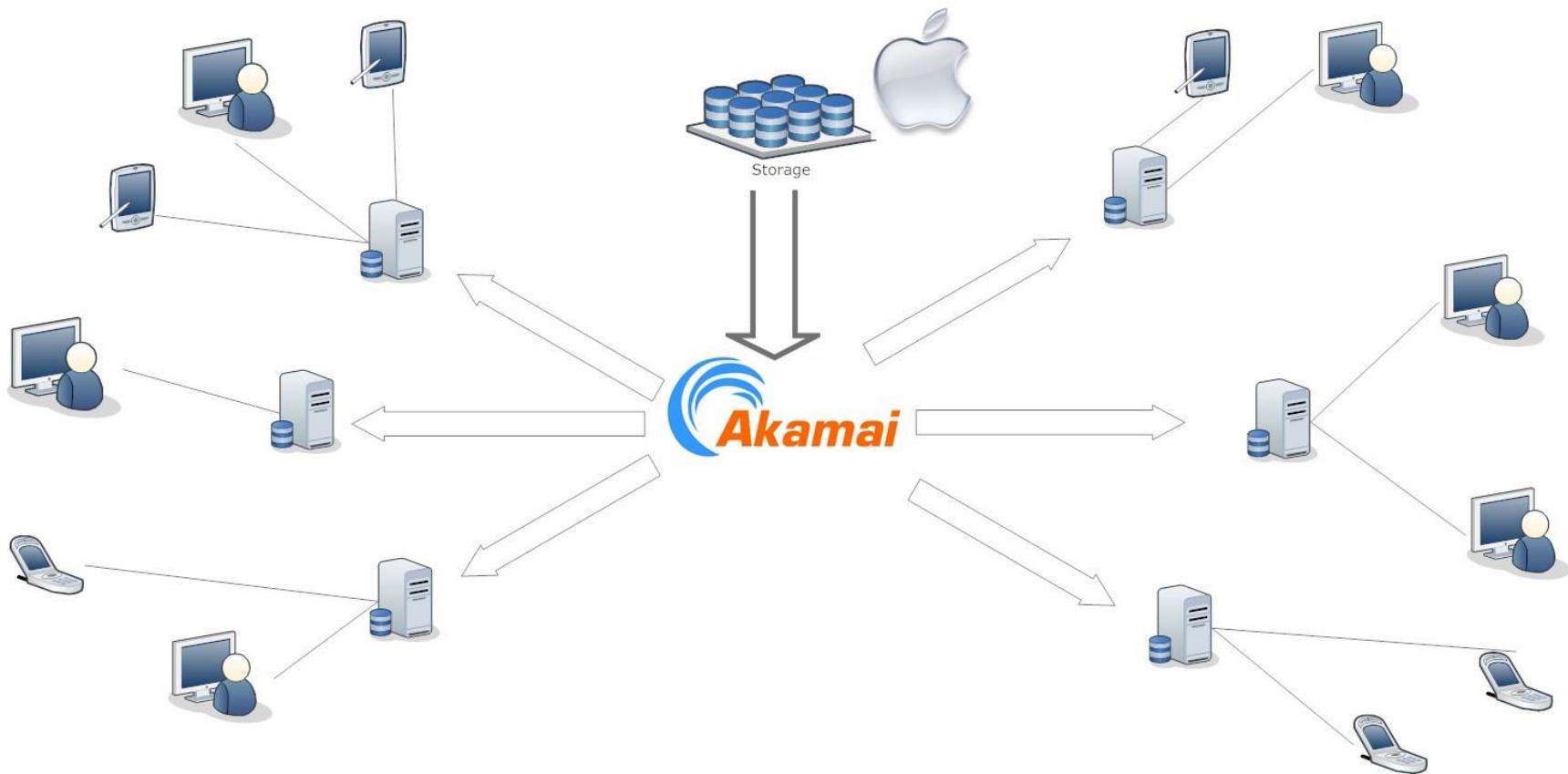


Content Delivery Network

4



AKAMAI



Outline

6

1. Introduction
2. Data-centric consistency models
3. Client-centric consistency models
4. Replica management
5. Consistency protocols



HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

1. Introduction

1.1. Why do we need replication

8

- Reliability
- Performance
- Scalability (?)

→ Consistency

1.2. Consistency

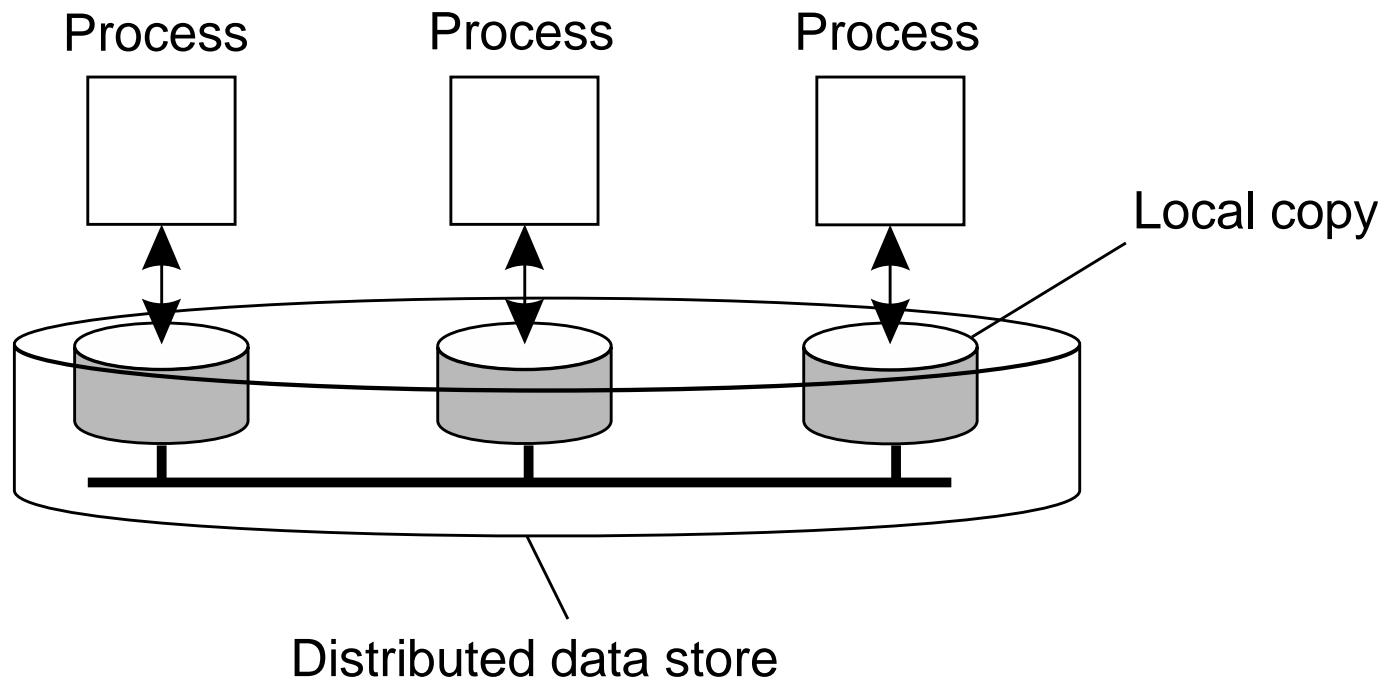
9

- Consistency of replicated data
 - Impossible to propagate the updates immediately
 - When? How?
- Strong consistency and Weak consistency
- Trade-off between consistency and performance

2. Data-centric consistency models

2.1. Distributed data store

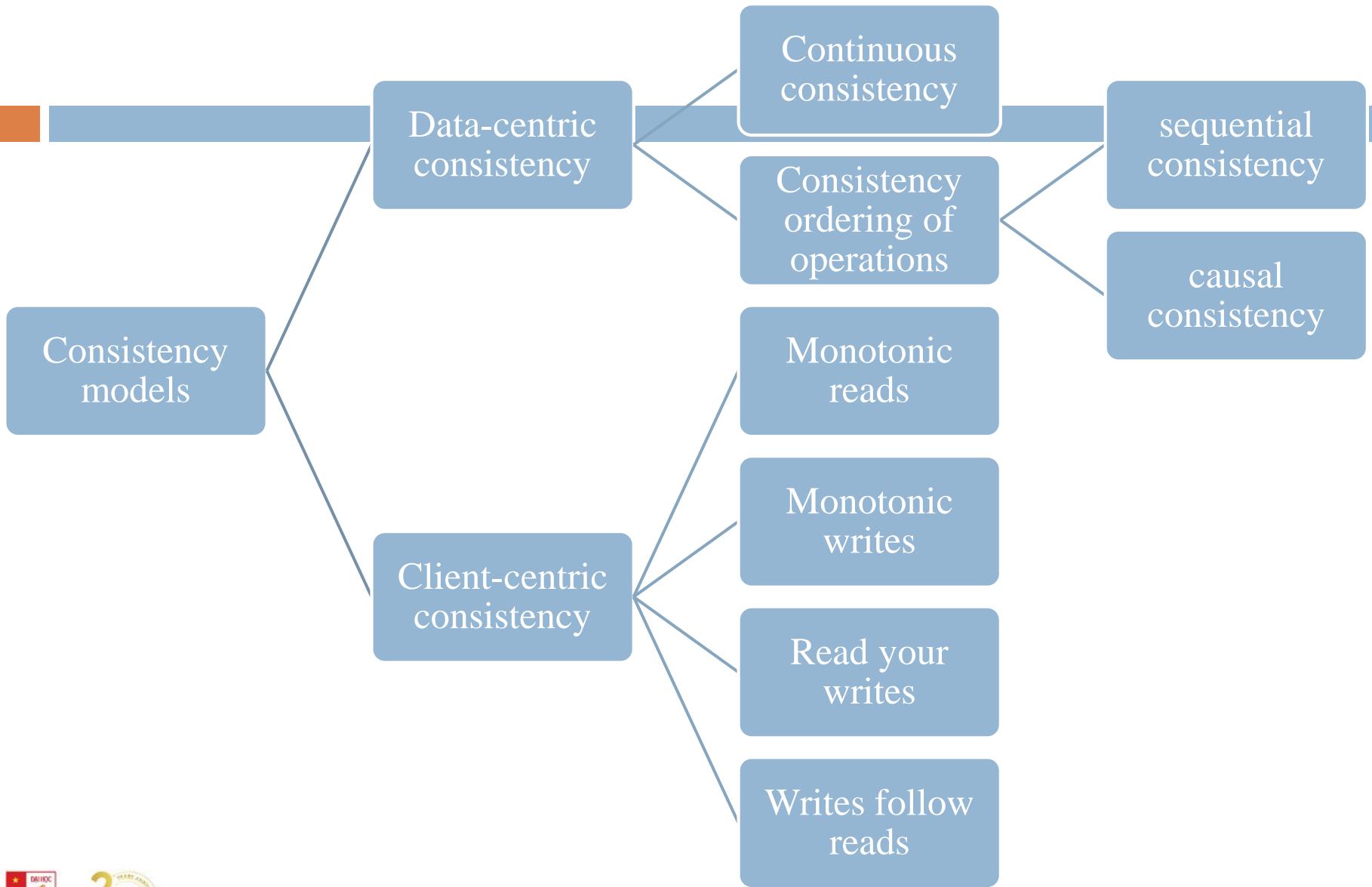
11



Consistency model

12

- A contract between processes and the data store.
- If processes agree to obey certain rules, the store promises to work correctly.
- Range of consistency models
- Major restrictions → easy
- Minor restrictions → difficult



2.2. Continuous consistency

14

- Factors for defining inconsistencies:
 - Deviation in numerical values
 - Deviation in staleness (the last time a replica was updated)
 - Deviation of ordering of update operations
- When the deviation exceeds a given value, Middleware will perform replication operations to bring the deviation back to the limit.

2.3. Conit (consistency unit)

15

Replica A

Conit		x = 6; y = 3
Operation	Result	
< 5, B>	x := x + 2	[x = 2]
< 8, A>	y := y + 2	[y = 2]
<12, A>	y := y + 1	[y = 3]
<14, A>	x := y * 2	[x = 6]

Replica B

Conit		x = 2; y = 5
Operation	Result	
< 5, B>	x := x + 2	[x = 2]
<10, B>	y := y + 5	[y = 5]

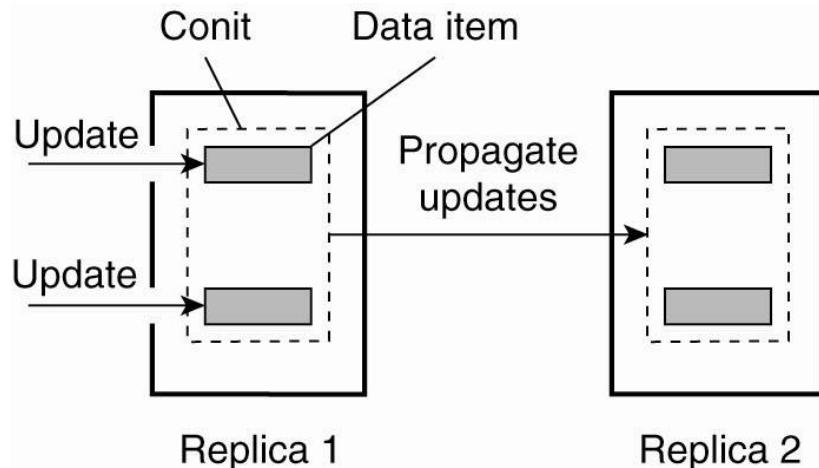
Time:?

Oder:?

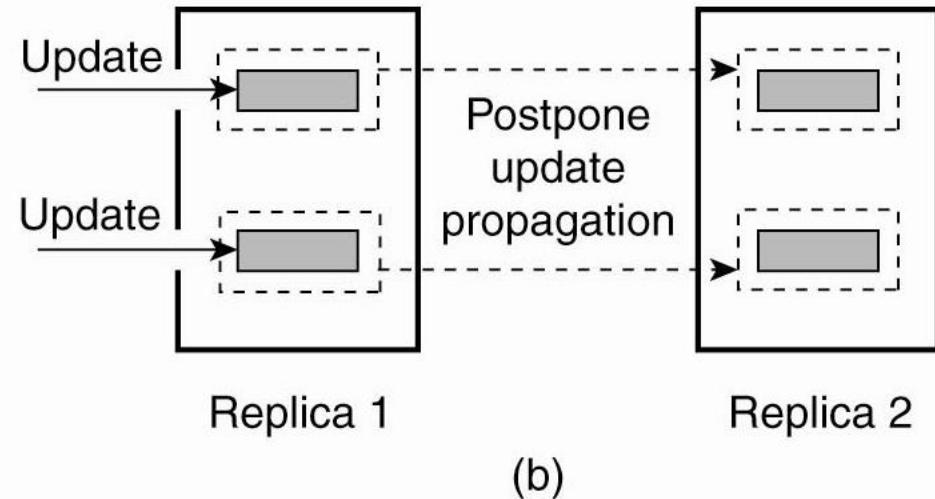
Value:?

Size of conit

16

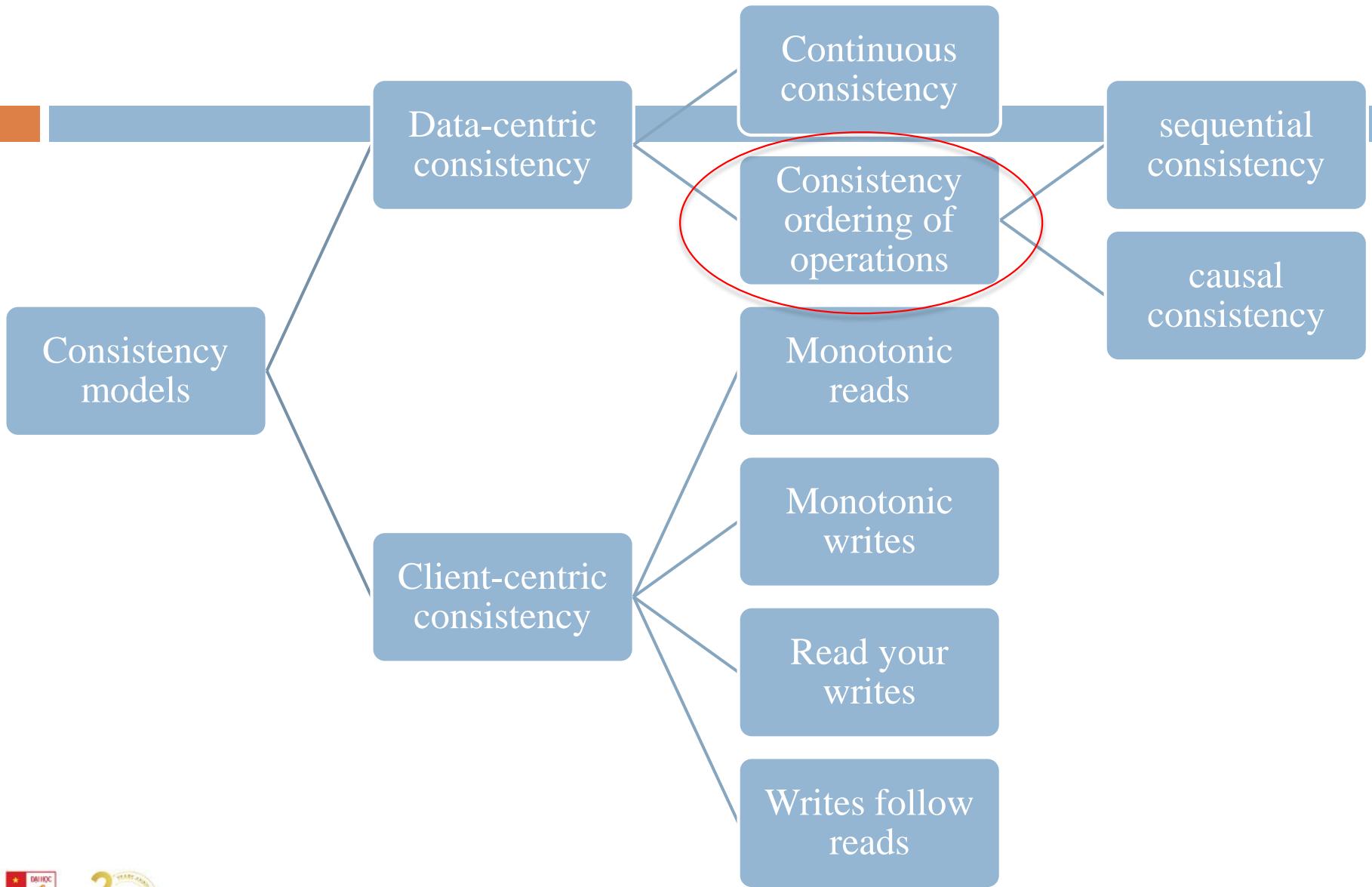


(a)



(b)

- A conit represents a lot of data → bring replica sooner in an inconsistent state
- Conit is very small → overhead related to managing the conit



2.4. Consistent Ordering of Operations

18

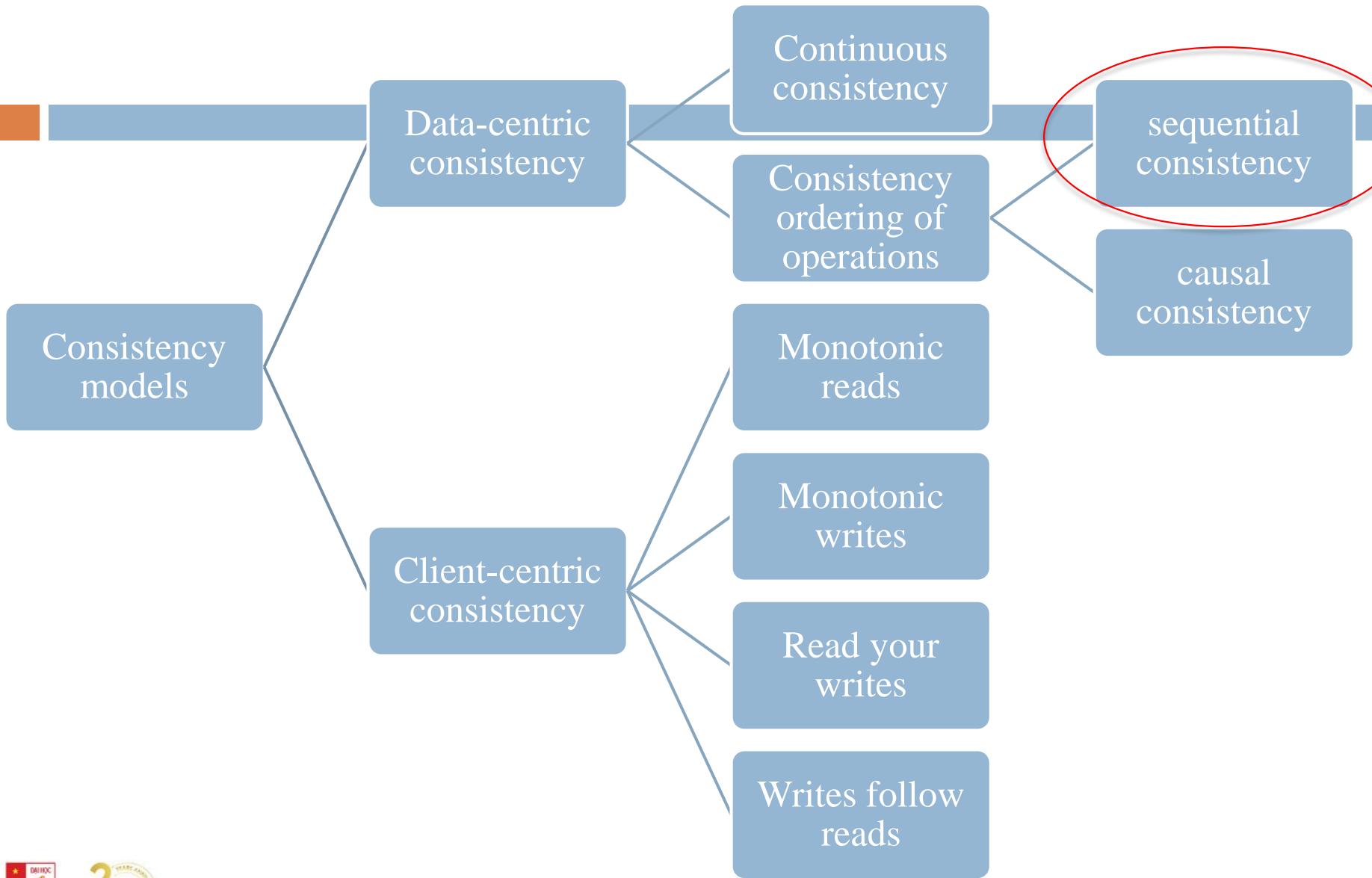
- Concurrent programming
- Parallel and distributed computing
- Express the semantics of concurrent accesses when shared resources are replicated
- Deal with consistently ordering operations on shared, replicated data.

Special notation

19

- Operations on data item x
 - Reading: $(R_i(x)b)$
 - Writing: $(W_i(x)a)$
 - Initial value of data item is NIL

$$\begin{array}{c} P1: \quad W(x)a \\ \hline P2: \quad \quad \quad R(x)NIL \quad R(x)a \end{array}$$



Sequential consistency

21

- Local sequence of operations
- Global sequence of operations
- A model is satisfied the sequential consistency if a global sequence of operations exists so that all local sequence of operations belong to that global sequence (in terms of operation order).
- Hint: All processes see the same order of *writes* operations.

Example

P1: $W(x)a$

P2: $W(x)b$

P3: $R(x)b$ $R(x)a$

P4: $R(x)b$ $R(x)a$

(a)

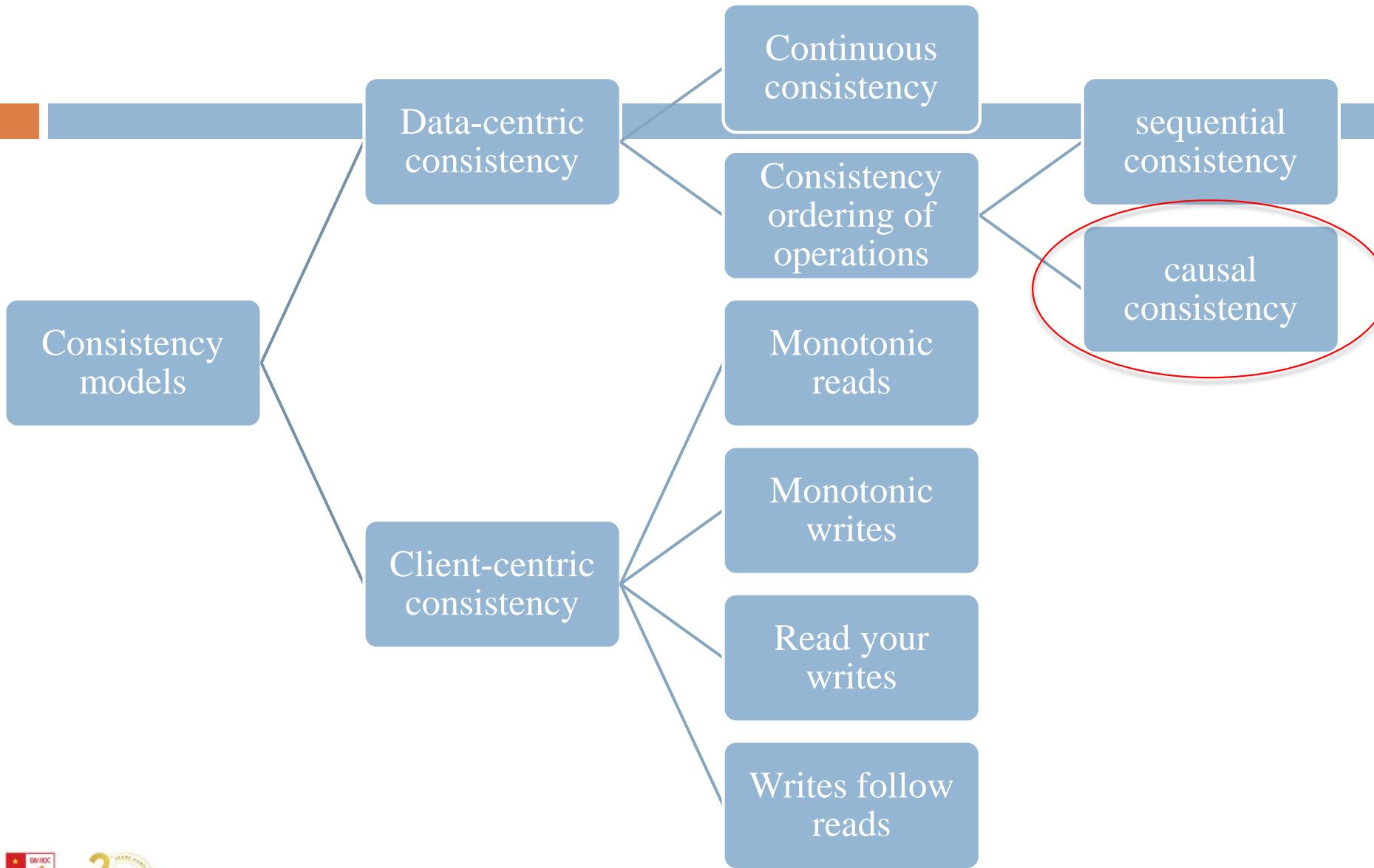
P1: $W(x)a$

P2: $W(x)b$

P3: $R(x)b$ $R(x)a$

P4: $R(x)a$ $R(x)b$

(b)



Causal consistency

24

- A distinction between events that are potentially causally related and those are not.
- *Writes that are potentially causally related must be seen by all processes in the same order. Concurrent writes may be seen in a different order on different machines.*

Causal consistency (cont.)

25

P1: $W(x)a$

P2: $R(x)a$ $W(x)b$

P3: $R(x)b$ $R(x)a$

P4: $R(x)a$ $R(x)b$

(a)

P1: $W(x)a$

P2: $W(x)b$

P3: $R(x)b$ $R(x)a$

P4: $R(x)a$ $R(x)b$

(b)

Grouping operations

26

- Sequential and causal consistency are defined at the level of read and write operations → appropriate for the hardware level (shared memory multiprocessor systems) → did not match the granularity as provided by applications.
- At application level: *read* and *write* operations are bracketed by the pair: ENTER_CS and LEAVE_CS

3 conditions

27

- A process does an acquire only after all the guarded shared data have been brought up to date.
- Before updating a shared data item, a process must enter a critical section.
- If a process wants to enter a critical region, it must check with the owner of the synchronization variable guarding to fetch the most recent copies

Example

28

P1: Acq(Lx) W(x)a Acq(Ly) W(y)b Rel(Lx) Rel(Ly)

P2: Acq(Lx) R(x) [] R(y) []

P3: Acq(Ly) R(y) []

3. Client-centric consistency

- 3.1. Eventual consistency
- 3.2. Monotonic reads
- 3.3. Monotonic writes
- 3.4. Read your writes
- 3.5. Writes follow reads

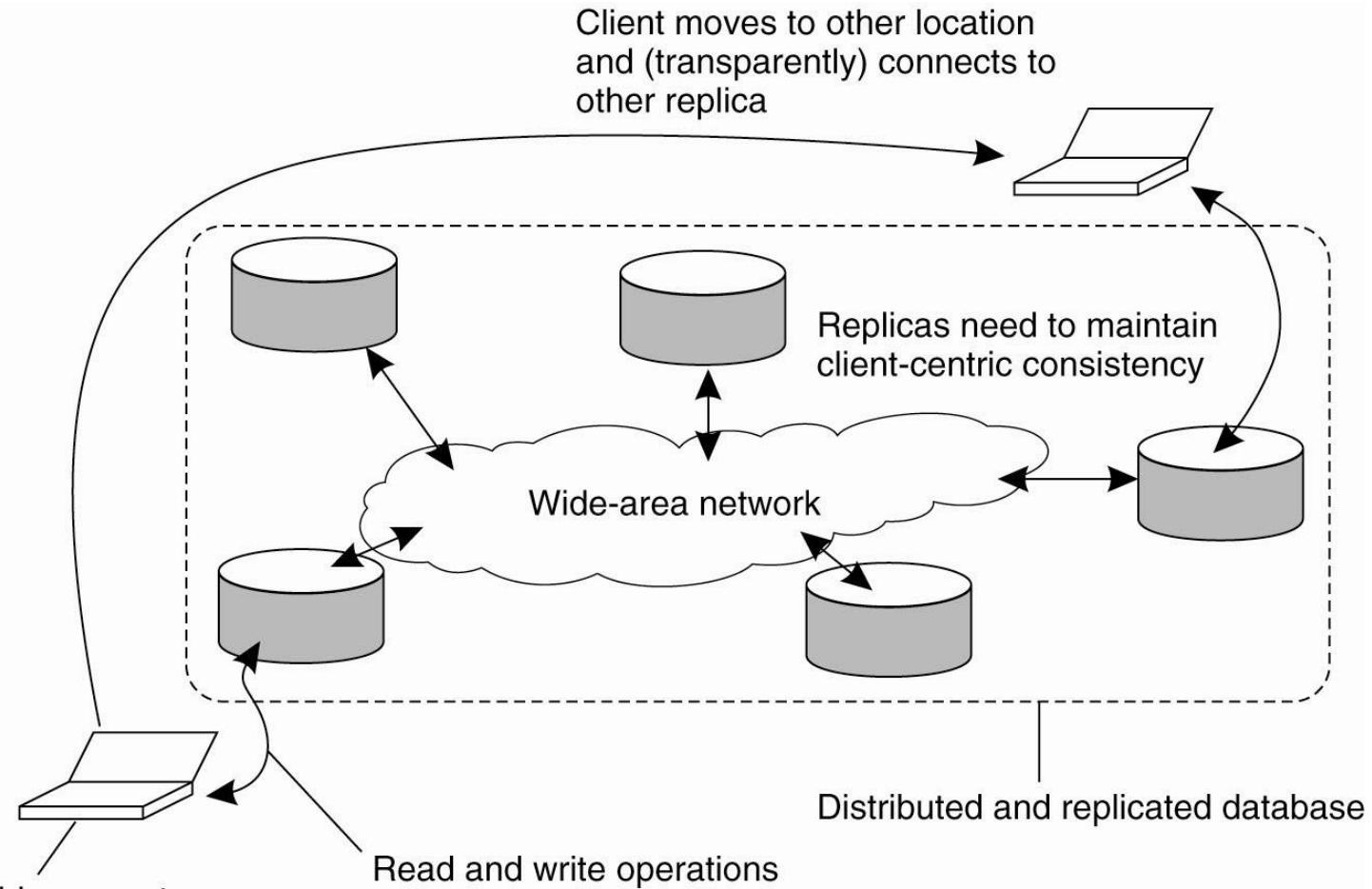
3.1. Eventual Consistency

30

- Consider two services: DNS, WWW
- Very little number of writes (updates), huge number of reads
- No write-write conflict, only the read-write conflicts.
- These systems tolerate a relatively high degree of inconsistency
- If no updates take place for a long time, all replicas will gradually become consistent.

Problem of Eventual Consistency

31



Client-centric consistency

32

- Provide guarantees for a single client concerning the consistency of accesses to a data store by that client.
- No guarantees for concurrent accesses by different clients.
- 4 types:
 - Monotonic reads
 - Monotonic writes
 - Read your writes
 - Writes follow reads

Notations

33

- L_i : ith local copy
- $x_i[t]$: data item x at L_i , time t
- $WS(x_i[t])$: writes operation at L_i that took place since initialization
- $WS(x_i[t_1]; x_j[t_2])$: All operations $WS(x_i[t_1])$ have been delivered to L_j , before t_2

3.2. Monotonic reads

34

$$\begin{array}{c} \text{L1: } \text{WS}(x_1) \qquad \qquad \qquad \mathbf{R}(x_1) - \\ \hline \text{L2: } \qquad \qquad \text{WS}(x_1; x_2) \qquad \qquad \qquad \mathbf{R}(x_2) \end{array}$$

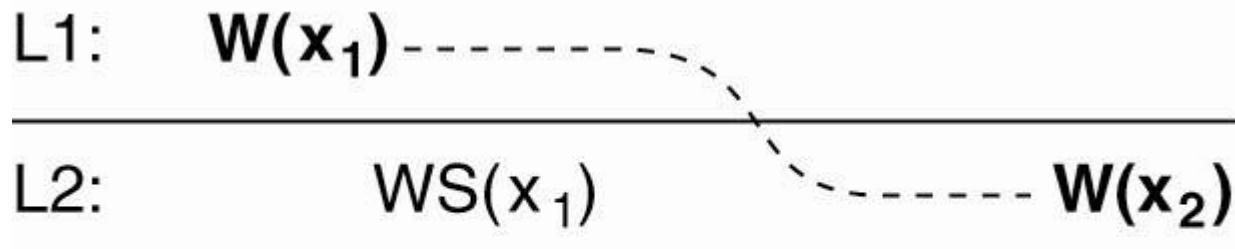
(a)

$$\begin{array}{c} \text{L1: } \text{WS}(x_1) \qquad \qquad \qquad \mathbf{R}(x_1) - \\ \hline \text{L2: } \qquad \qquad \text{WS}(x_2) \qquad \qquad \qquad \mathbf{R}(x_2) \end{array}$$

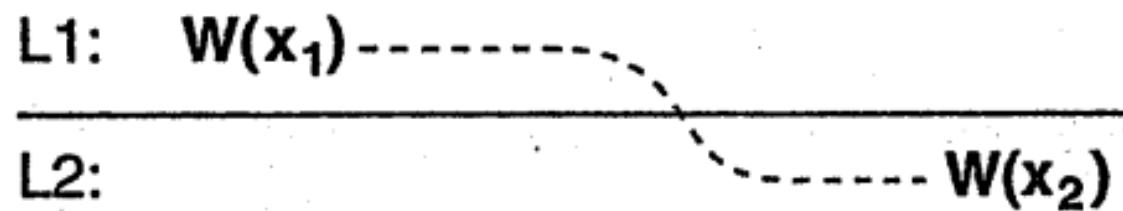
(b)

3.3. Monotonic writes

35



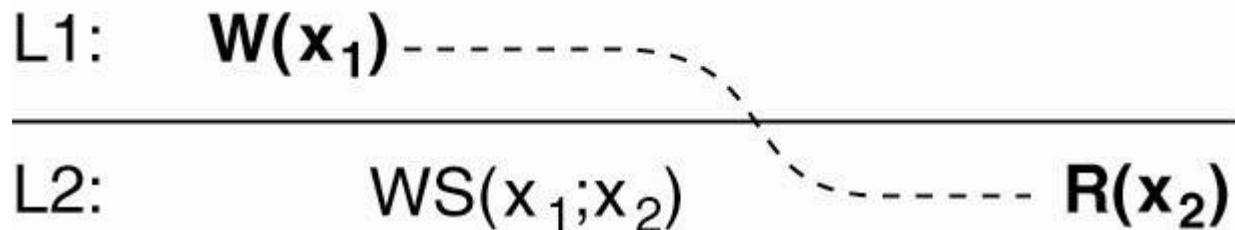
(a)



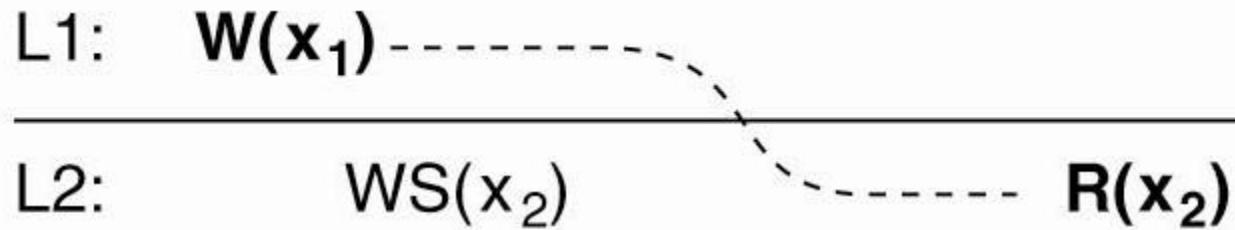
(b)

3.4. Read your writes

36



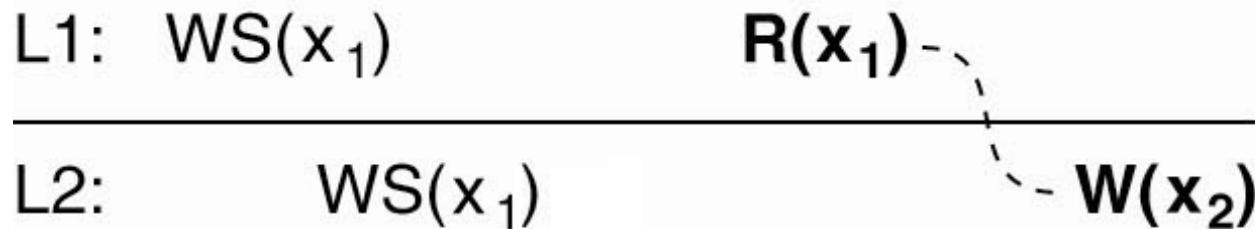
(a)



(b)

3.5. Writes follow reads

37



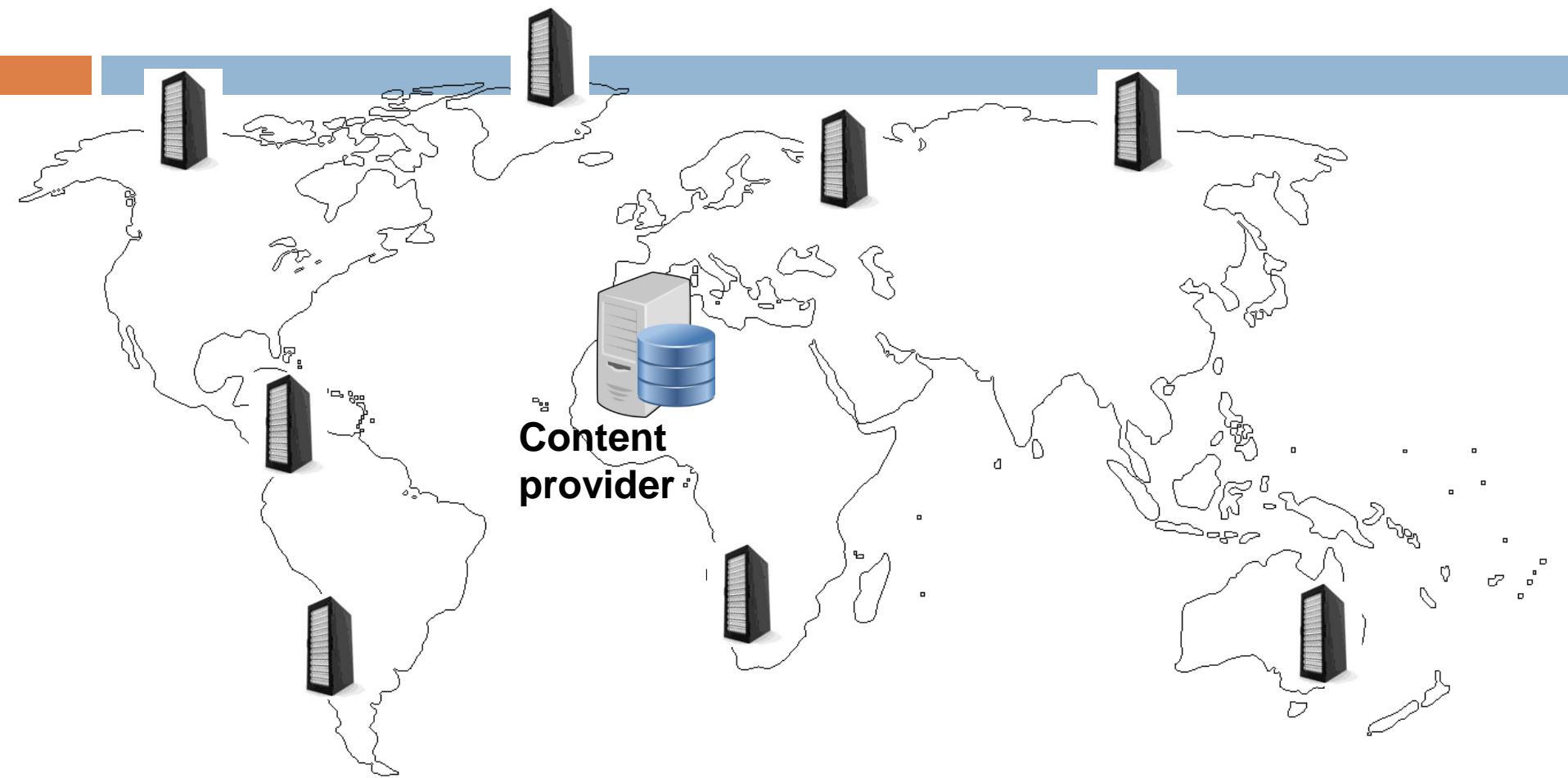
(a)



(b)

4. Replica management

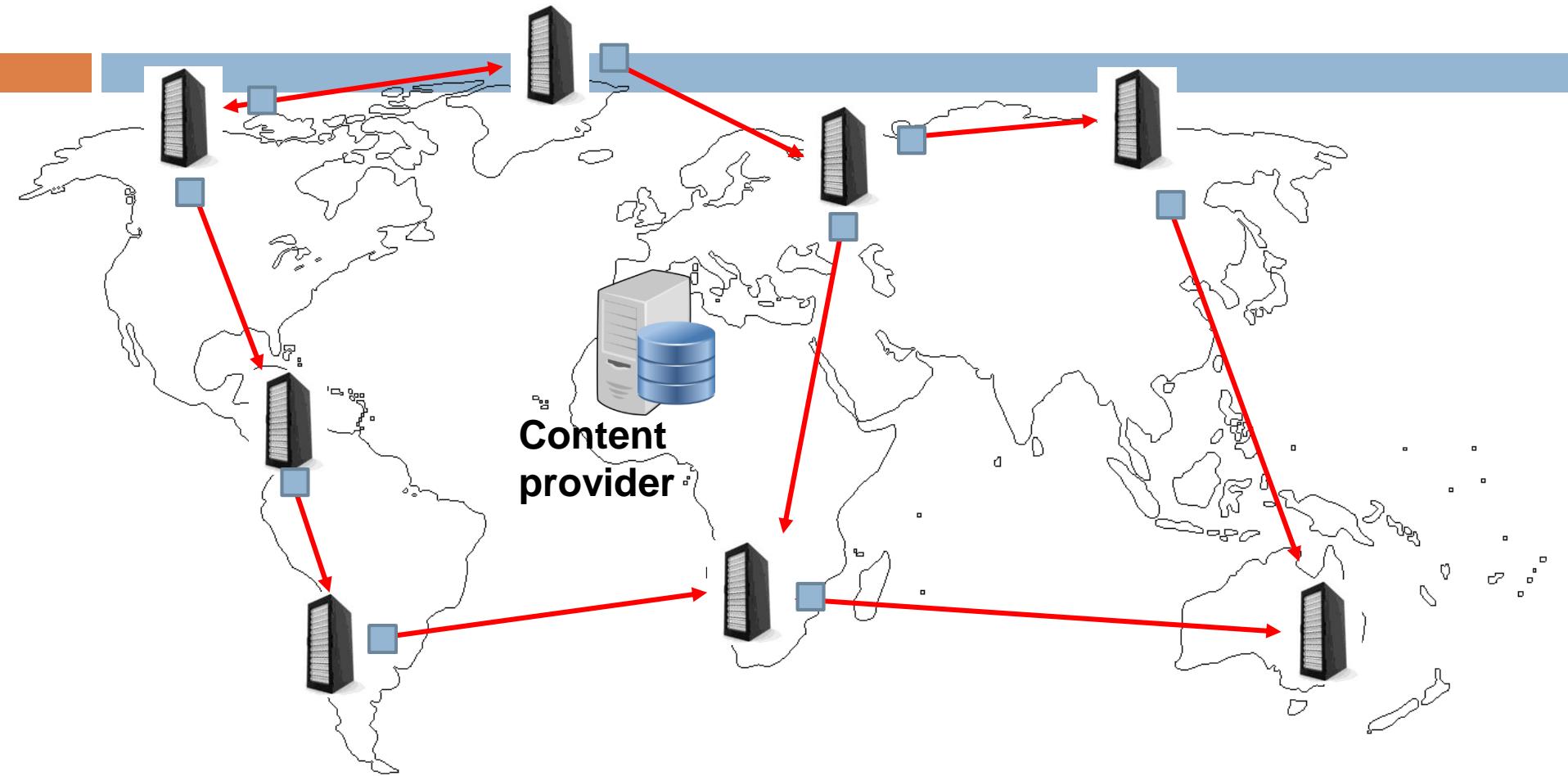
- 4.1. Replica server placement
- 4.2. Content replication and placement
- 4.3. Content distribution



4.1. Replica server placement

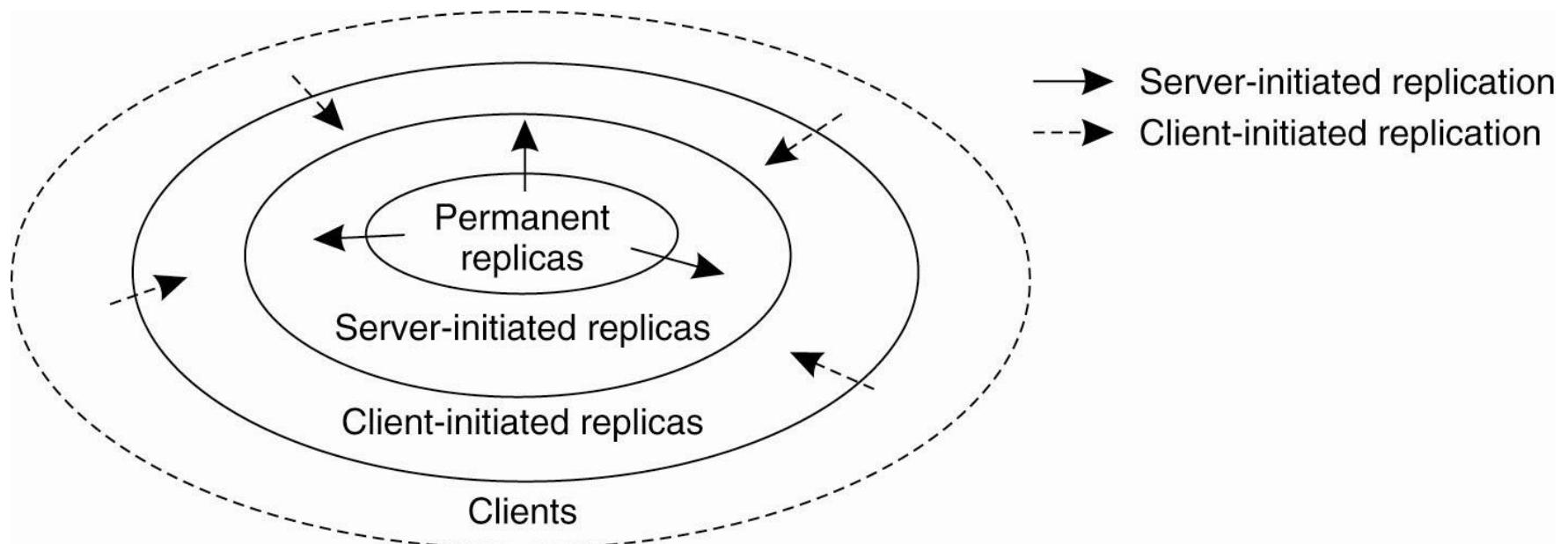
40

- Problem
 - N locations for replica placement
 - Determine K out of N locations
- Solution 1
 - Distance between clients and locations
 - Select one server at a time
- Solution 2: Ignoring the position of clients
 - Take the topology of the Internet
 - Sort the ASes
 - Place the server on the router with the largest number of Network interfaces
 - Continue with the sorted list



4.2. Content replication and placement

42



Permanent replicas

43

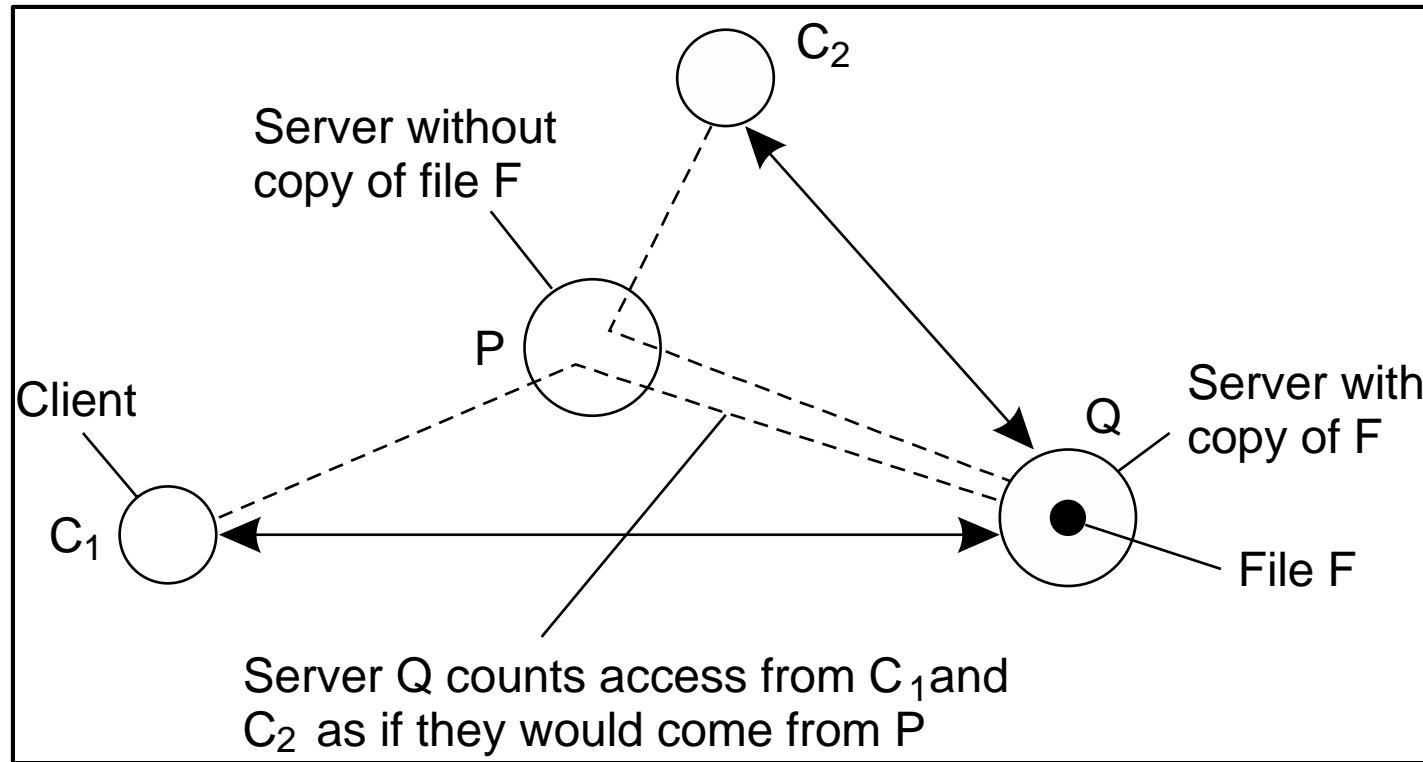
- ❑ The initial set of replicas
- ❑ The number of replica is small
- ❑ First kind of distribution
 - Data is replicated across a limited number of servers
 - For each request, it is forwarded to one of the servers (eg. using Round-robin strategy).
- ❑ 2nd kind of distribution: mirroring
 - Client simply chooses one of the various mirror sites.
- ❑ Shared-nothing architecture

Server-initiated Replicas

44

- Server is active
 - The number of requests increased suddenly
 - Activate other replicas
- Reduce load for replicas
- Update the data to a new replica closer to the client

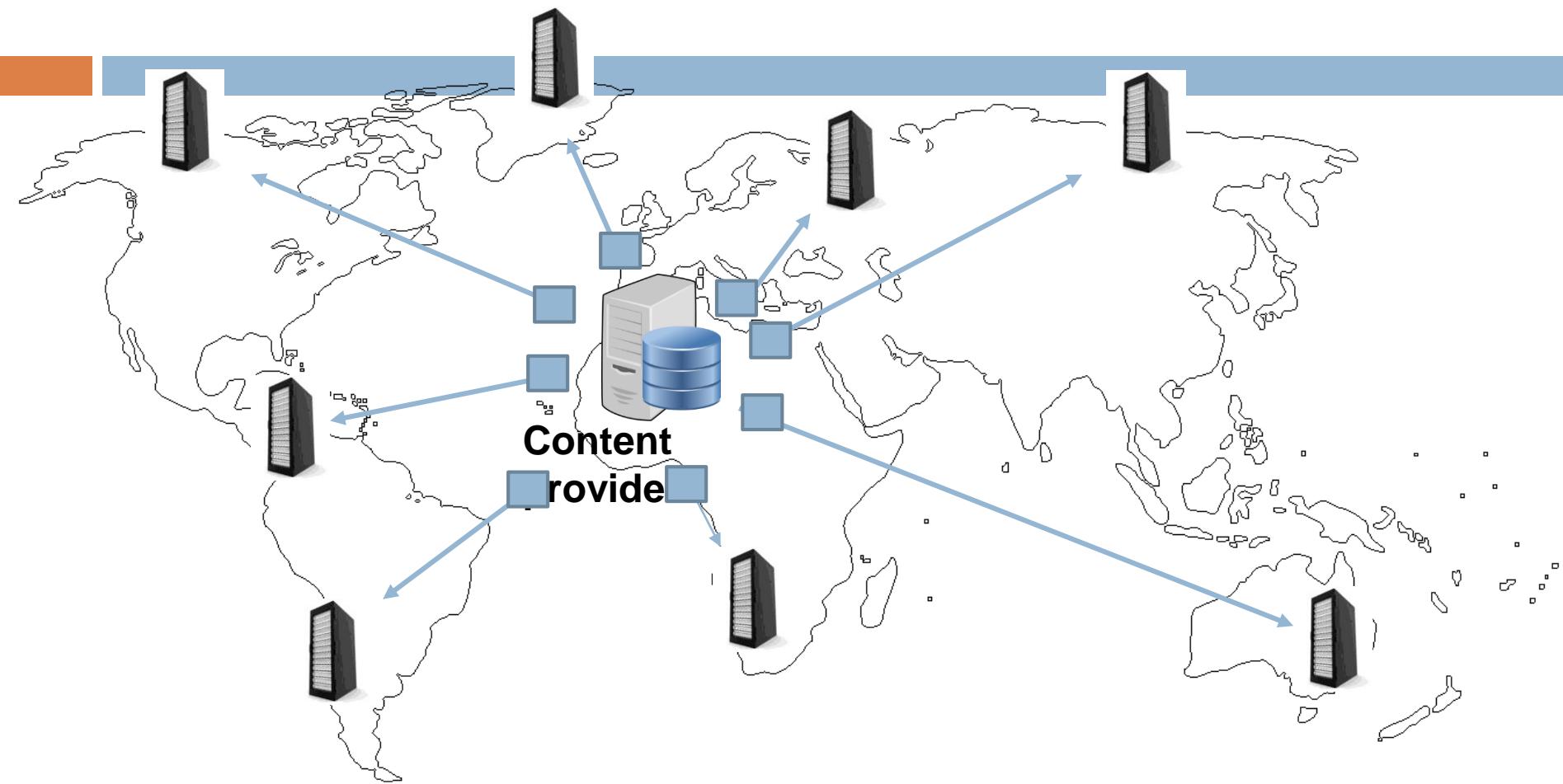
Server-initiated Replicas



Client-initiated Replicas

46

- Caching
 - Client manages the cache management, decides to update the cache
 - Erase
 - Write
 - Policy caching
- Can share caches between clients



4.3. Content Distribution

48

- State vs. Operations
- Pull vs. Push
- Unicast vs. Multicast

State vs. Operations

49

- Solutions for updating data:
 - ▣ Propagate only a notification of an update
 - Use little network bandwidth.
 - Read-to-write ratio is small
 - ▣ Transferring the modified data
 - Read-to-write ratio is high
 - ▣ Send update operation (active replication)

Pull/Push

50

- Push: server after updating notification data for all clients
 - Replica activated by server
 - Ensure high consistency
 - Weak interaction (eg when client or replica needs to update data)
 - The server should have a list of all connected clients
- Pull: client when need data will ask server
 - Usually used for client caches
 - Suitable for high writes-reads ratio
 - Increased access time (with cache miss)
- Mixed

Uni vs. multicast

51

- Multicasting:
 - Appropriate in case 1 replica wants to promote updates to $(N-1)$ other copies in a data store
 - More efficient and economical than sending $(N-1)$ times
 - Appropriate for the push-based approach
 - Not suitable if destination nodes belong to a LAN
- Unicasting:
 - Appropriate for pull-based

5. Consistency protocols

- 5.1. Continuous consistency
- 5.2. Primary-based protocols
- 5.3. Replicated write
- 5.4. Cache coherence

5.1. Continuous consistency

53

- Bounding numerical deviation
- Bounding staleness deviation
- Bounding ordering deviation

Bounding numerical deviation

54

- Single data item x .
- Each write $W(x)$ has an associated weight that represents the numerical value by which x is updated
- The write's origin: $\text{origin}(W(x))$
- each server S_i keeps log L_i of writes that are performed on its own local copy of x .
- $TW[i,j]$ is the writes executed by S_i that originated from S_j

$$TW[i,j] = \sum \{ \text{weight}(W) \mid \text{origin}(W) = S_j \text{ & } W \in L_i \}$$

- $TW[k,k]$: aggregated writes submitted to S_k

Bounding numerical deviation

55

- Actual value of x

$$v(t) = v(0) + \sum_{k=1}^N TW[k,k]$$

$$v_i = v(0) + \sum_{k=1}^N TW[i,k]$$

$$v_i \leq v(t)$$

- The threshold:

$$v(t) - v_i \leq \delta_i$$

Bounding Staleness Deviation

56

- Can use local time of processes to evaluate
 - Server S_k has vector clock RVC_k
 - if $RVC_k[i] = T(i)$ $\Rightarrow S_k$ has seen all operations on S_i at $T(i)$
 - $T(i)$: local time of server I
 - When $T(k) - RVC_k[i] > \delta$ \Rightarrow eliminate operations having $T > RVC_k[i]$

Bounding ordering deviation

57

- Each replica has a write queue
- Global order should be considered
- The largest number of write operations are in the queue
- When this number exceeds, the server will stop the execution and will negotiate with other servers in order

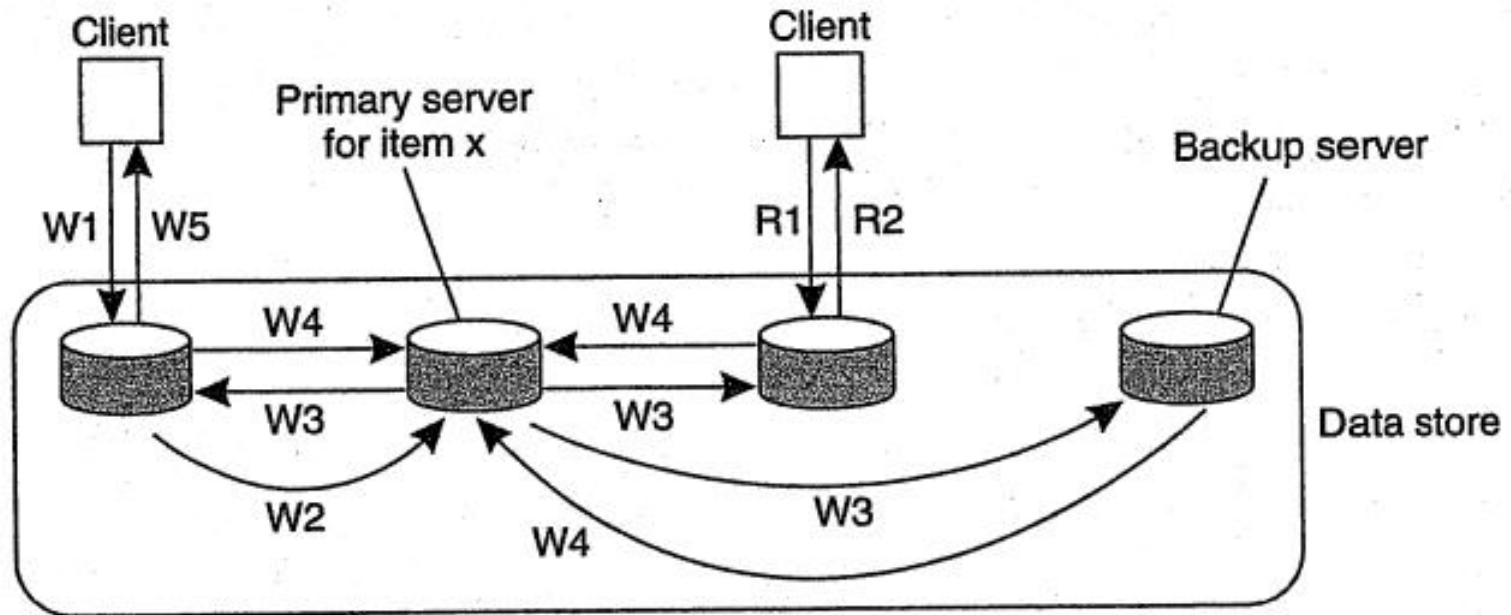
5.2. Primary-based protocols

58

- Consistency model => complex
- Developers need simpler models
- Each data item has a primary that is responsible for manipulating operations on that data items
- Fixed-primary (remote-write protocol)
- Local-primary (local write protocol)

Remote-write protocol

59

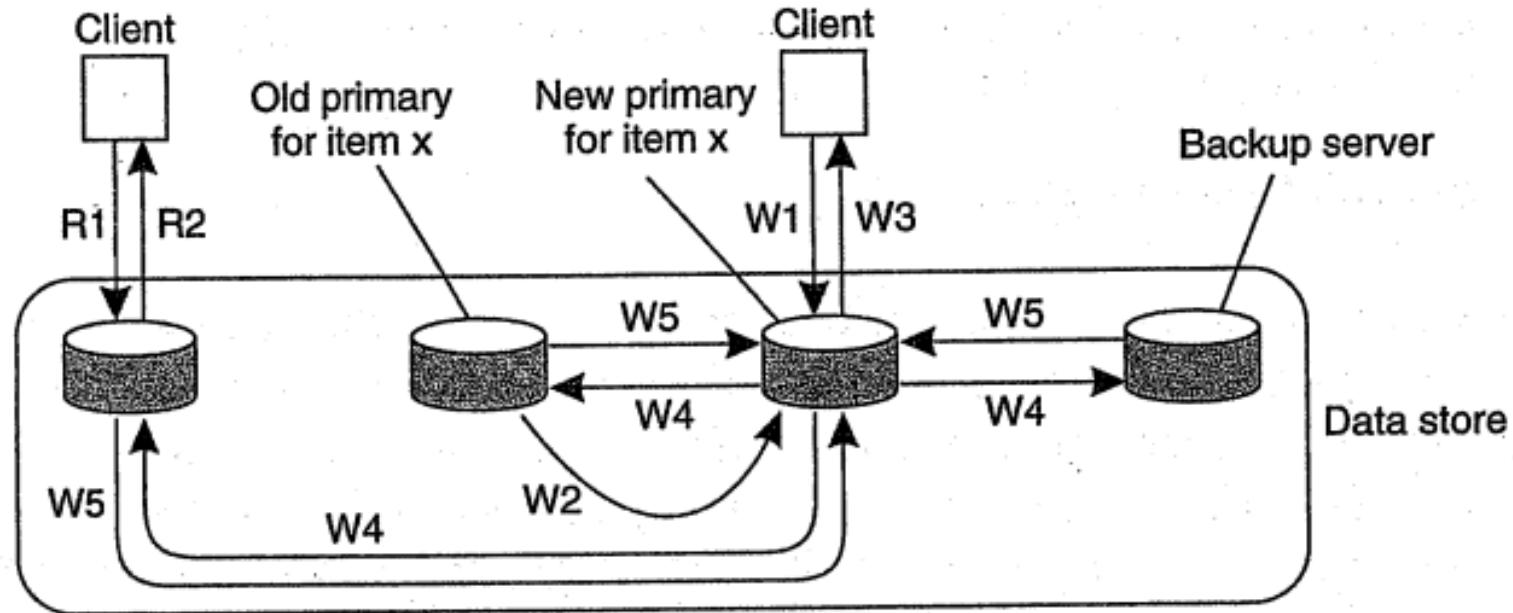


- W1. Write request
- W2. Forward request to primary
- W3. Tell backups to update
- W4. Acknowledge update
- W5. Acknowledge write completed

- R1. Read request
- R2. Response to read

Local-write protocol

60



W1. Write request

W2. Move item x to new primary

W3. Acknowledge write completed

W4. Tell backups to update

W5. Acknowledge update

R1. Read request

R2. Response to read



5.3. Replicated-write protocols

61

1. Active replication
2. Quorum-based protocol

5.3.1. Active replication

62

- A process is responsible for propagating the update operation to all replicas
- Need a total ordered mechanism
 - ▣ logical synchronization of Lamport
 - ▣ Sequencer

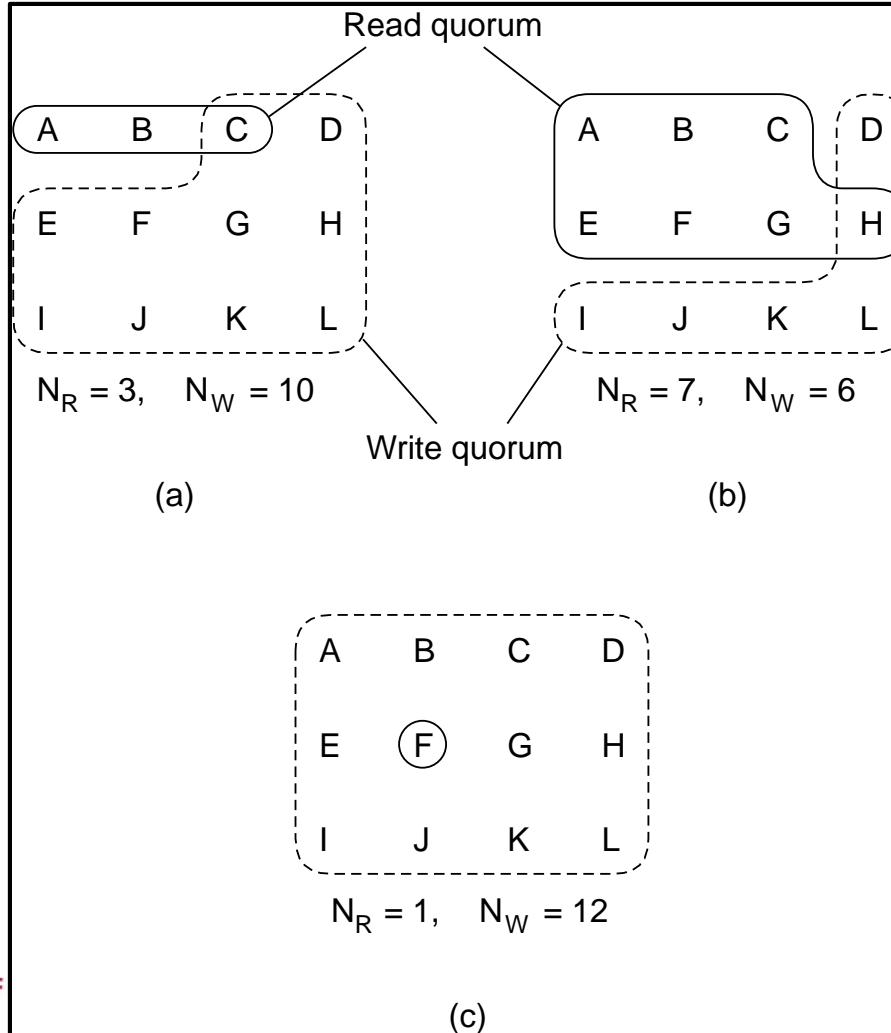
5.3.2. Quorum-based protocol

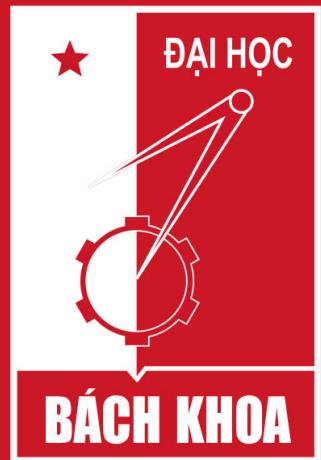
63

- For strong consistency => need to update all replicas
- After updating at a costly cost => not all replicas are read => wasted
- Is there a reduction in the number of replicas that need updating?
- When reading the data
 - Risk of reading the old data
 - Read more data in some other replicas => Select the copy with the latest data
- Write Quorum & Read Quorum
 - $N_R + N_w > N$
 - $N_w > N/2$

Example of quorum

64





25 YEARS ANNIVERSARY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

CÁC HỆ THỐNG PHÂN TÁN VÀ ỨNG DỤNG



HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Chapter 6: Fault Tolerance

Outline

3

1. Introduction to fault tolerance
2. Process resilience
3. Reliable client-Server Communication
4. Reliable Group Communication
5. Distributed Commit
6. Recovery

1. Introduction to fault tolerance

- 1.1. Basic concept
- 1.2. Failure models
- 1.3. Failure masking by redundancy

1.1. Basic concept

5

- Being *fault tolerant* related to ***Dependable systems*** which cover:
 - Availability
 - Reliability
 - Safety
 - Maintainability
- ***Fail/Fault***
- ***Fault Tolerance***
- ***Transient Faults***
- ***Intermittent Faults***
- ***Permanent Faults***

1.2. Failure models

6

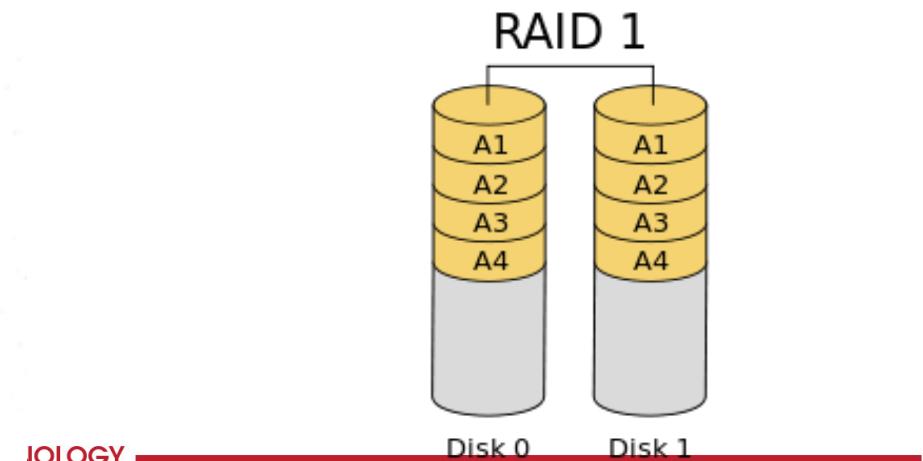
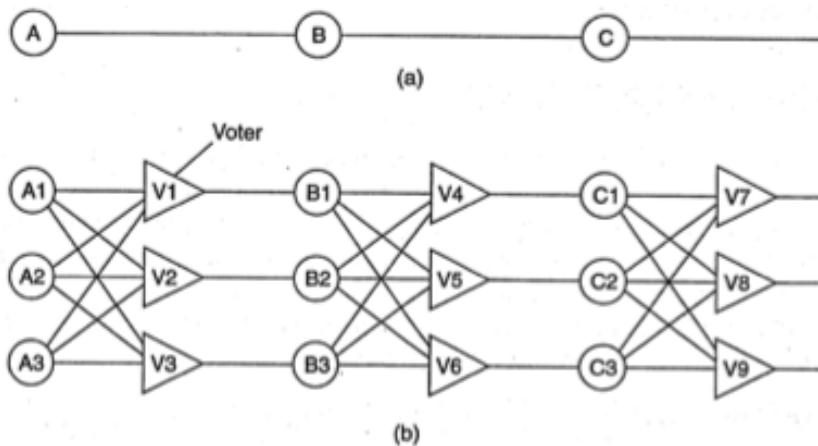
□ Different types of failures

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure	A server fails to respond to incoming requests
Receive omission	A server fails to receive incoming messages
Send omission	A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure	A server's response is incorrect
Value failure	The value of the response is wrong
State transition failure	The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times
Fail-stop failure	A server stops producing output and its halting can be detected by other systems
Fail-silent failure	Another process may incorrectly conclude that a server has halted
Fail-safe	A server produces random output which is recognized by other processes as plain junk

1.3. Failure masking by redundancy

7

- Three possible kinds for masking failure
 - *Information redundancy*
 - *Time redundancy*
 - *Physical redundancy*
- Triple Modular Redundancy (*TMR*)
- RAID 1



2. Process resilience

- 2.1. Design issues
- 2.2. Failure masking and replication
- 2.3. Agreement in faulty system
- 2.4. Failure detection

2.1. Design issues (1/3)

9

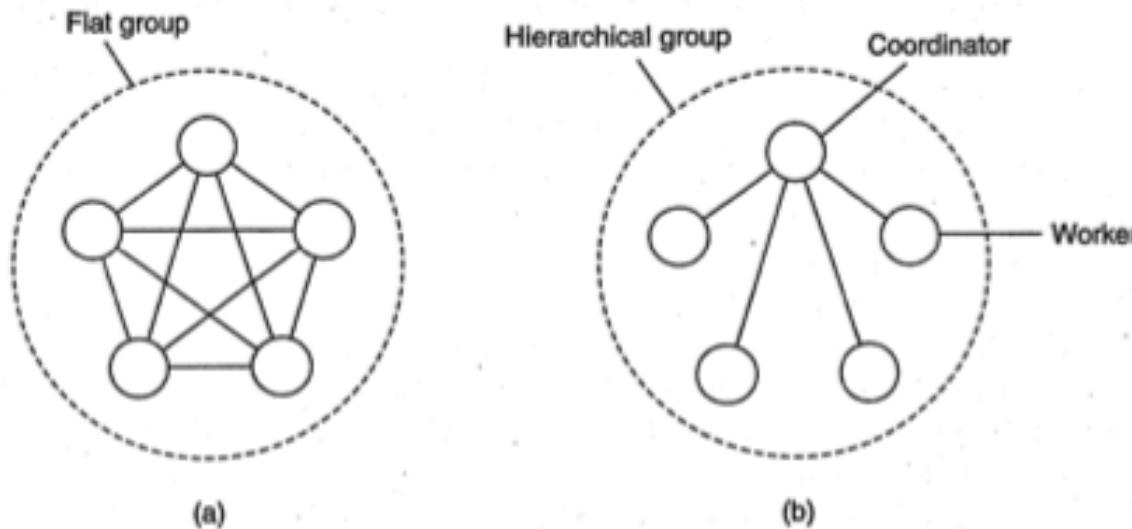
□ *Process group*

- Key approach: organize several identical processes into a group
- Key property: message is sent to the group itself and all members receive it
- Dynamic: create, destroy, join or leave

2.1. Design issues (2/3)

10

- *Flat Groups* versus *Hierarchical Groups*



□ Comparison

	Advantages	Disadvantages
Flat Groups  	Symmetrical No single point of failure Group still continues while one of the processes crashes	Complicated decision making
Hierarchical Groups	Easy decision making	Loss of coordinator brings the group to halt

2.1. Group membership(3/3)

11

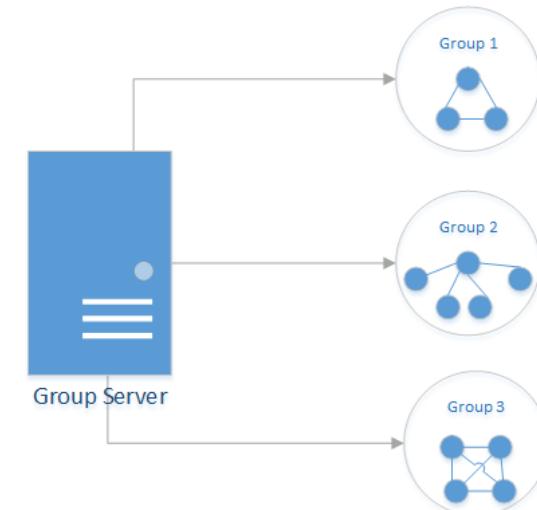
- ***Group Server***

Approach

- Send request
- Maintain databases of all groups
- Maintain their memberships

Disadvantages

- A single point of failure



- ***Distributed way***

Approach - each member communicates directly to all others

Disadvantages

- Fail-stop semantics are not appropriate
- Leaving and joining must be synchronous with data messages being sent

- ***Membership issues***

What happens when multiple machines crash at the same time?

2.2. Failure masking and Replication

12

- ***Primary-based protocols***
 - Used in form of primary-backup protocol
 - Organize group of processes in hierarchy
 - Backups execute election algorithm to choose a new primary
- ***Replicated-write protocols***
 - Used in form of *active replication* or *quorum-based protocols*
 - Organize a collection of identical processes into a flat group
 - Called ' k fault tolerant' if system can survive faults in k components.

2.3. Agreement in Faulty systems (1/3)

13

- *Different cases*
 1. Synchronous versus asynchronous system
 2. Communication delay is bounded or not
 3. Message delivery is ordered or not
 4. Message transmission is done through unicasting or multicasting
- *Circumstances under which distributed agreement can be reached*

		Message ordering				Communication delay
		Unordered		Ordered		
Process behavior	Synchronous	X	X	X	X	Bounded
	Asynchronous			X	X	Unbounded
		Unicast	Multicast	Unicast	Multicast	Communication delay
				X		

2.3. Agreement in Faulty systems (2/3)

14

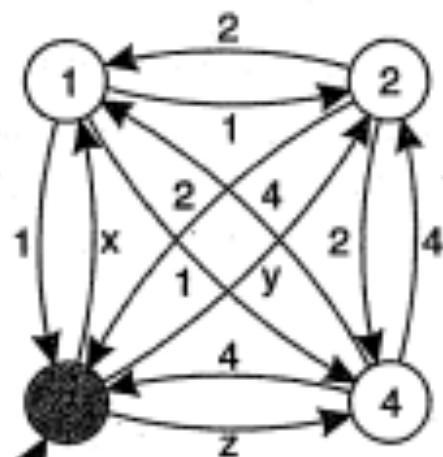
- *Byzantine agreement*

Assuming N processes, each process i provides a value v_i

Goal: construct a vector V of length N

If i is nonfaulty then $V[i] = v_i$

- *Example:* $N = 4$ and $k = 1$



(a)

1 Got(1, 2, x, 4)
2 Got(1, 2, y, 4)
3 Got(1, 2, z, 4)
4 Got(1, 2, 3, 4)

(b)

1 Got (1, 2, y, 4)	2 Got (1, 2, x, 4)	4 Got (1, 2, y, 4)
(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

(c)

2.3. Agreement in Faulty systems (3/3)

15

- *Lamport et al. (1982)* proved that agreement can be achieved if $2k+1$ correctly process for total of $3k + 1$, with k faulty processes (or more than $2/3$ correctly process with $2k+1$ nonfaulty processes)
- *Fisher et al. (1985)* proved that where messages is not delivered within a known and finite time -> No possible agreement if even only one process is faulty because arbitrarily slow processes are indistinguishable from crashed ones

2.4. Failure Detection

16

- Two mechanisms - *Active process and Passive Process*
 - *Timeout mechanism* is used to check whether a process has failed. Main disadvantages:
 - Possible wrong detection when simply stating failure due to unreliable networks. Thus, generate false positives and a perfectly healthy process could be removed from the membership list
 - Failure detection is plain crude, based only on the lack of a reply to a single message
 - How to *design* a failure *detection subsystem*?
 - Through gossiping
 - Through probe
 - Regular information exchange with neighbors -> a member for which the availability information is old, will presumably have failed
 - Failure detection *subsystem ability*?
 - Distinguish network failures from node failures by letting nodes decide whether one of its neighbors has crashed
- Inform nonfaulty processes about the failure detection using **FUSE** approach



SCHOOL OF INFORMATION TECHNOLOGY AND COMPUTER SCIENCE
BACH KHOA HANOI UNIVERSITY



HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

3. Reliable Client-Server Communication

3.1. Point-to-Point Communication

3.2. RPC Semantics in the Presence of Failures

3.1. Point-to-Point Communication

18

- Point-to-point communication is established by using **reliable transport protocols**
 - TCP masks omission failures by using acknowledgments and retransmissions -> failure is hidden from TCP client
 - Crash failures cannot be masked because TCP connection is broken
 - > client is informed through exception raised
 - > Let the distributed system automatically set up a new connection

3.2. RPC Semantics in the Presence of Failures (1/5)

19

- **RPC (Remote Procedure Calls)** hides communication by remote procedure calls
- **Failures occur** when:
 - Client is unable to locate the server
 - Request message from the client to the server is lost
 - Server crashes after receiving a request
 - Reply message from the server to the client is lost
 - Client crashes after sending a request

3.2. RPC Semantics in the Presence of Failures (2/5)

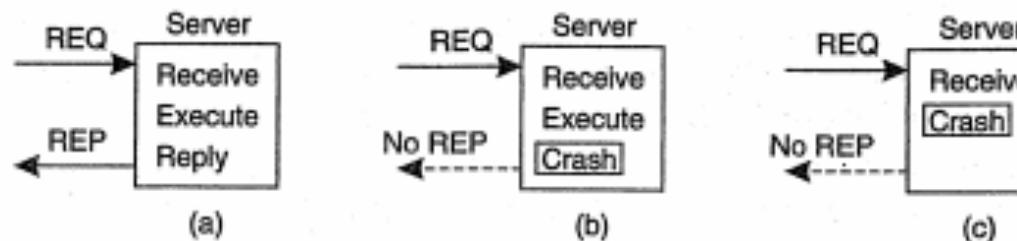
20

- Client is unable to locate the server, e.g. the client cannot locate a suitable server, or all servers are down...
-> Solution: raise **Exception**
Drawbacks:
 - not every language has exceptions or signals.
 - Exception destroys the transparency
- Lost request Messages, detected by setting a timer
 - Timer expires before a reply or ack -> resend message
 - True loss -> no difference between retransmission and original
 - So many messages lost -> client gives up and concludes that the server is down, which is back to “Cannot locate server”
 - No message lost: let the server to detect and deal with retransmission

3.2. RPC Semantics in the Presence of Failures (3/5)

21

- Server Crashes



(a) Normal Case (b) Crash after execution (c) Crash before execution

Difficult to distinguish between (b) and (c)

- (b) the system has to report failure back to the client
- (c) need to retransmit the request

3 philosophies for servers:

- At least once semantics
- At most once semantics
- Exactly once semantics

4 strategies for the client

- Client decide to never reissue a request
- Client decide to always reissue a request
- Client decide to reissue a request only when no acknowledgment received
- Client decide to reissue a request only when receiving acknowledgment

3.2. RPC Semantics in the Presence of Failures (4/5)

22

- Server Crashes (next)

8 considerable combinations but none is satisfactory

- 3 events: M (send message), P (print text), C (crash)
- **6 orderings
combinations**

**All possible
combinations**

1. $M \rightarrow P \rightarrow C$
2. $M \rightarrow C (-> P)$
3. $P \rightarrow M \rightarrow C$
4. $P \rightarrow C -(> M)$
5. $C (-> P \rightarrow M)$
6. $C (-> M \rightarrow P)$

Client	Strategy $M \rightarrow P$			Strategy $P \rightarrow M$		
	MPC	MC(P)	C(MP)	PMC	PC(M)	C(PM)
Always	DUP	OK	OK	DUP	DUP	OK
Never	OK	ZERO	ZERO	OK	OK	ZERO
Only when ACKed	DUP	OK	ZERO	DUP	OK	ZERO
Only when not ACKed	OK	ZERO	OK	OK	DUP	OK

OK = Text is printed once
DUP = Text is printed twice
ZERO = Text is not printed at all

Conclusion

The possibility of server crashes changes the nature of RPC and distinguishes single-processor systems from distributed systems

3.2. RPC Semantics in the Presence of Failures (5/5)

23

- Lost Reply Messages

- **Solution:** rely on a timer set by client's operating system

Difficulty -> The client is not really sure why there was no answer: lost or slow?

- **Idempotent request:** asking for the first 1024 bytes of a file has no side effects and executing as often as necessary without any harm
 - **Assign sequence number:** server keeps track of the most recently received sequence number from each client and refuse to carry out any request a second time

- Client crashes

- **Solution:** activate computation called “orphan”

Difficulty:

- Waste CPU cycles
 - Lock files or tie up valuable resources
 - Confusion if the client reboots and does RPC again

- **Alternative solutions:**

- Orphan extermination
 - Reincarnation
 - Gentle Reincarnation

Expiration

4. Reliable Group Communication

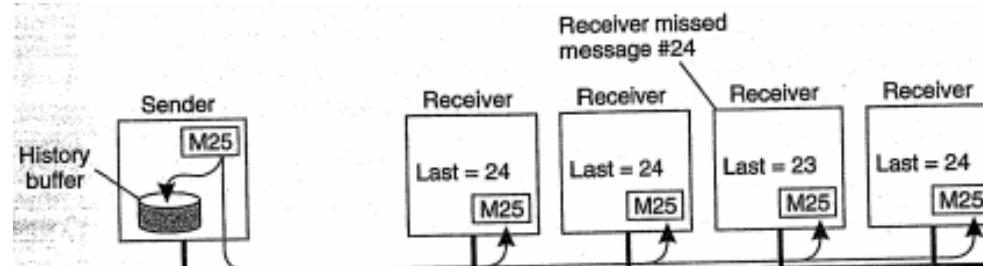
- 4.1. Basic Reliable – Multicasting Schemes
- 4.2. Scalability in Reliable Multicasting
- 4.3. Atomic Multicast

4.1. Basic Reliable – Multicasting Schemes

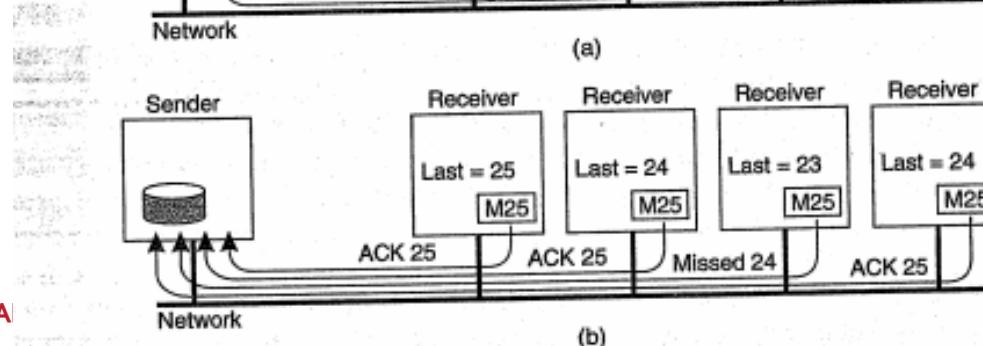
25

- **Multicasting** means that a message sent to a process group, should be delivered to each member of that group
- **In presence of faulty process:** multicasting is reliable when all nonfaulty group members receive the message
- Solution to reliable multicasting when all receivers are known and assumed not to fail

(a) Message Transmission



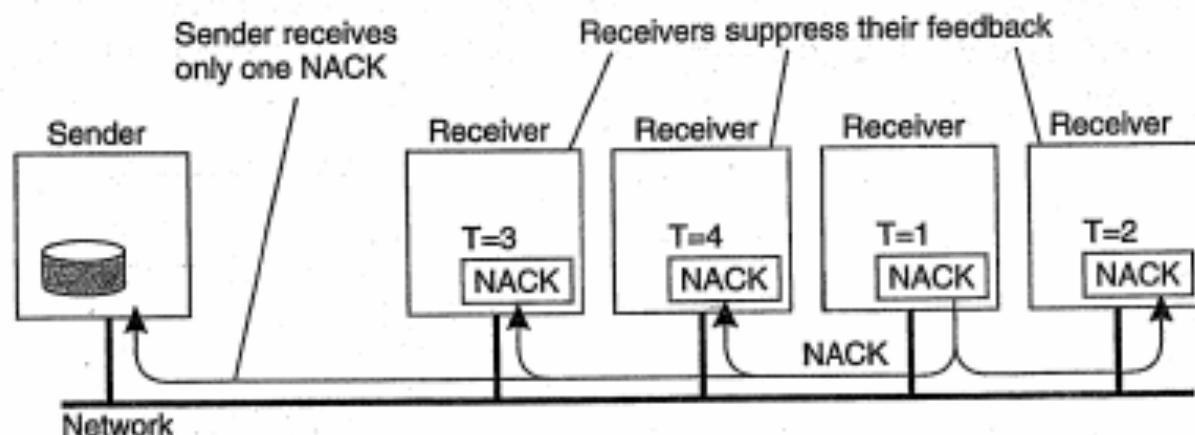
(b) Reporting feedback



4.2. Scalability in Reliable Multicasting (1/2)

26

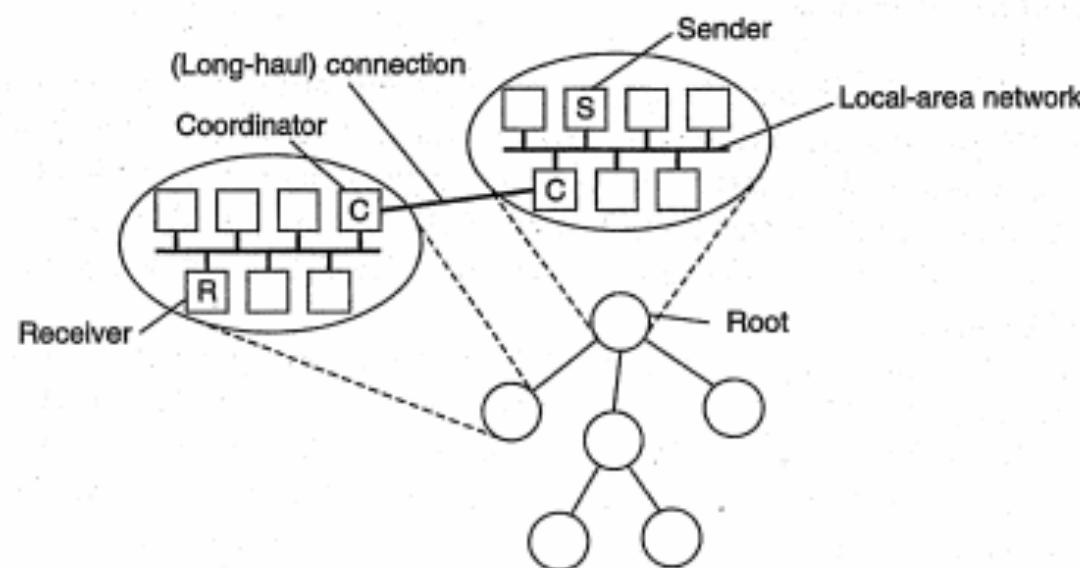
- **Problem of reliable multicast scheme** it that cannot support large numbers of receivers
- **Nonhierarchical feedback control**
 - Key: reduce the number of feedback messages returned
 - Model: feedback suppression which underlies the scalable reliable multicasting (SRM)
 - In SRM, receiver reports when missing message and multicasts its feedback to the rest of the group. Other group members will suppress its own feedback.



4.2. Scalability in Reliable Multicasting (2/2)

27

- **Hierarchical feedback control**
 - Achieving scalability for very large groups of receivers requires adopting hierarchical approaches
 - Each local coordinator forwards the message to its children and later handles retransmission requests



4.3. Atomic Multicast (1/6)

28

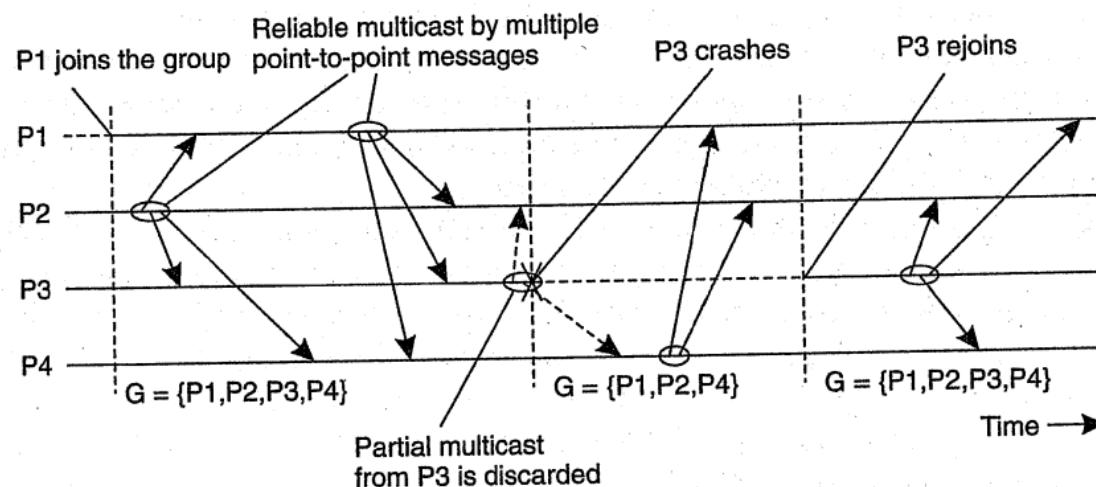
- **Atomic multicast:**
 - Guarantee that a message is delivered to either all processes or to none at all.
 - All messages are delivered in the same order to all processes
 - In non-atomic multicast, when there are multiple updates and a replica crashes, it is difficult to locate operations missing and the order these operations are to be performed
 - In atomic multicast, when replica crashes, it ensures that nonfaulty processes maintain a consistent view of the database and force reconciliation when a replica recovers and rejoins the group

4.3. Atomic Multicast (2/6)

29

- **Virtual Synchrony**

To distinguish between receiving and delivering message, adopt **distributed system model which consists of communication layer**



- Multicast message **m** is associated with a list of processes to which it should be delivered, named **group view**
- Each process on that list has the same view.
- Message m, group view G. While the multicast is taking place, another process joins or leaves the group -> **View change** – multicast a message vc announcing the joining or leaving of a process -> two multicast messages in transit: m and vc

4.3. Atomic Multicast (4/6)

30

- **Message Ordering**
 - Unordered multicasts

Process P1	Process P2	Process P3
sends m1	receives m1	receives m2
sends m2	receives m2	receives m1

Sample of three communicating processes in the same group -> the ordering of events per process is shown along the vertical axis

- FIFO-ordered multicasts

Process P1	Process P2	Process P3	Process P3
sends m1	receives m1	receives m3	receives m3
sends m2	receives m3	receives m1	receives m4
	receives m2	receives m2	
	receives m4	receives m4	

Sample of four processes in the same group with two different senders, and a possible delivery order of messages under FIFO-ordered multicasting

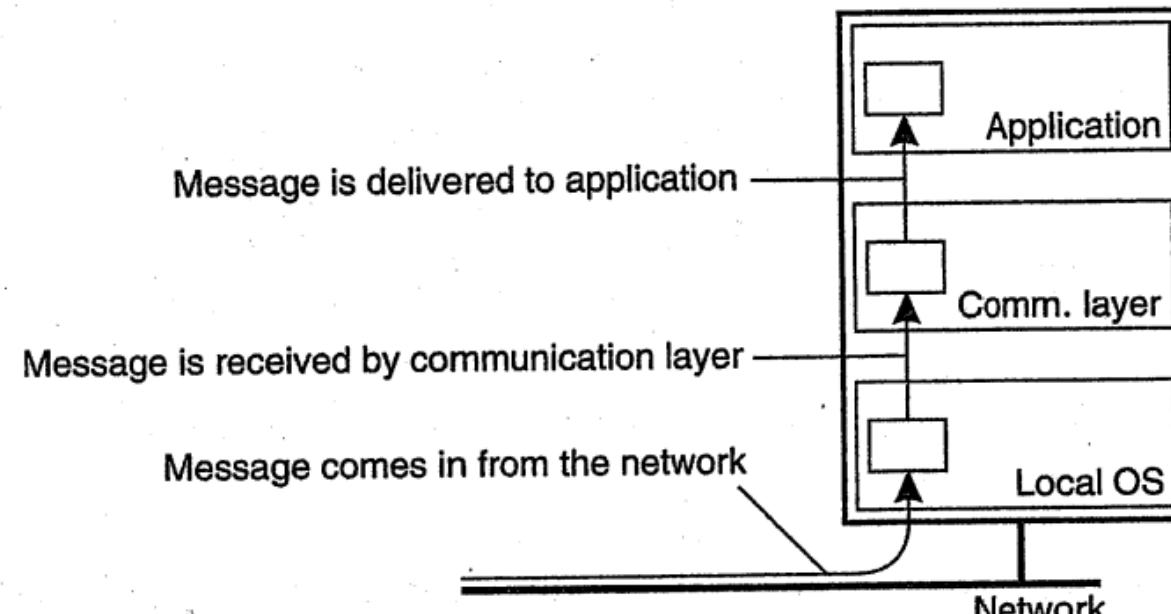
- Causally-ordered multicasts

- Totally-ordered multicasts

4.3. Atomic Multicast (5/6)

31

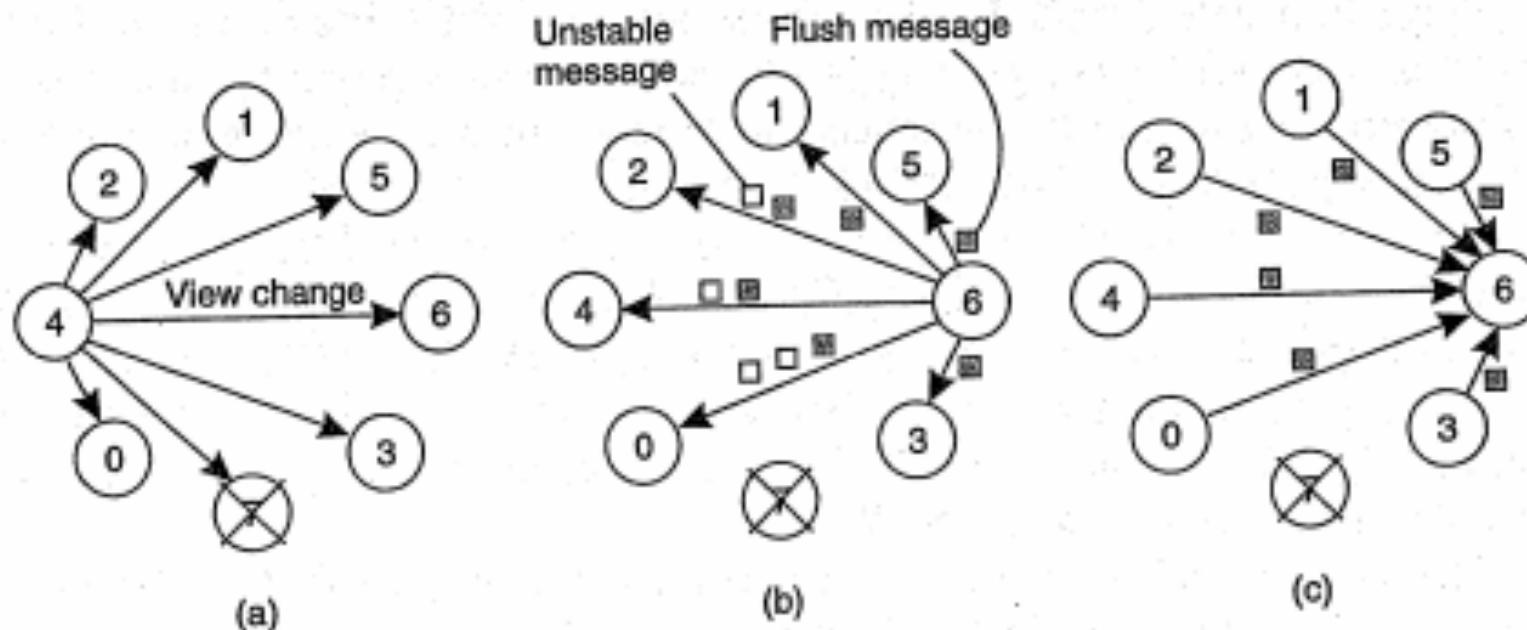
- **Implementing Virtual Synchrony**
 - Goal: Guarantee that all messages sent to view G are delivered to all nonfaulty processes in G before the view change.
 - Solution: Let every process in G keep m until it knows for sure that all members in G have received it.
 - Stable message



4.3. Atomic Multicast (6/6)

32

- **Implementing Virtual Synchrony**
- Illustration of selecting stable message
 - a) Process 4 notices that process 7 has crashed and sends a view change
 - b) Process 6 sends out all its unstable messages and subsequently marks it as being stable, followed by a flush message
 - c) Process 6 installs the new view when it has received a flush message from everyone else





HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

5. Distributed Commit

5.1. Two-Phase Commit

5.2. Three-Phase Commit

About Distributed Commit

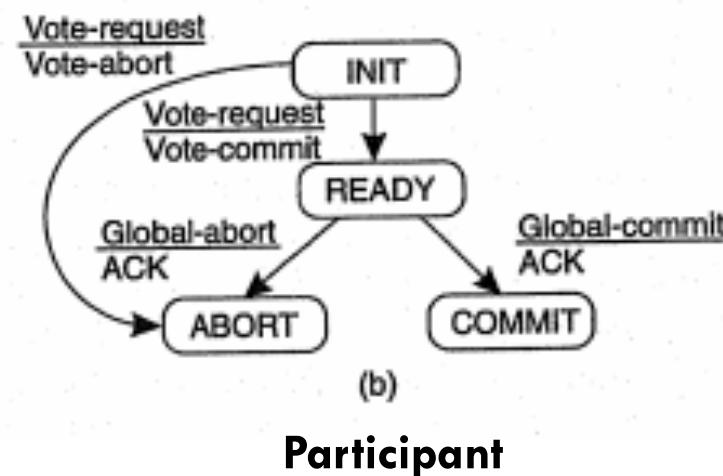
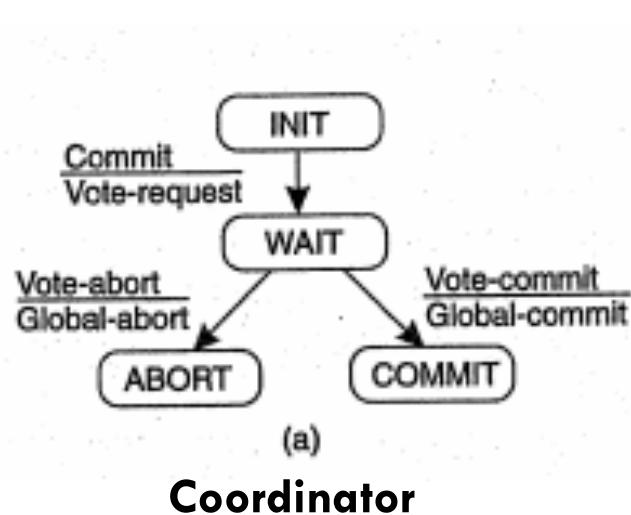
34

- Distributed commit involves **having an operation being performed by each member of a process group, or not at all**
 - Reliable multicasting: Operation = message delivery
 - Distributed transactions: Operation = transaction commit at the single site that takes part in the transaction
- Distributed commit is established by means of **coordinator**
- **One-phase commit protocol:** a simple scheme where a coordinator tells all other processes (called participants) whether or not to perform the operation in question.
- **Sophisticated schemes:** Two-phase commit or Three-phase commit

5.1. Two-Phase Commit - 2PC (1/5)

35

- Protocol consists **two phase**:
 - Phase 1
 - ✓ Coordinator sends a VOTE_REQUEST message to all participants
 - ✓ After receiving, participant returns VOTE_COMMIT or VOTE_ABORT message to the coordinator
 - Phase 2
 - ✓ Coordinator collects all votes and send GLOBAL_COMMIT message or GLOBAL_ABORT message to participants
 - ✓ Each participant that voted for a commit waits for the final reaction to commit or not the transaction



5.1. Two-Phase Commit - 2PC (2/5)

36

- **Participant Solution:**
 - use timeout mechanism or let a participant P contact
 - Let a participant P contact another participant Q and decide what it should do. If P is in READY status, here are various options

State of Q	Action by P
COMMIT	Make transition to COMMIT
ABORT	Make transition to ABORT
INIT	Make transition to ABORT
READY	Contact another participant

5.1. Two-Phase Commit - 2PC (3/5)

37

- Sample of actions taken in place by the participant:

```
write INIT to local log;  
wait for VOTE_REQUEST from coordinator;  
if timeout {  
    write VOTE_ABORT to local log;  
    exit;  
}  
if participant votes COMMIT {  
    write VOTE_COMMIT to local log;  
    send VOTE_COMMIT to coordinator;  
    wait for DECISION from coordinator;  
    if timeout {  
        multicast DECISION_REQUEST to other participants;  
        wait until DECISION is received; /* remain blocked */  
        write DECISION to local log;  
    }  
    if DECISION == GLOBAL_COMMIT  
        write GLOBAL_COMMIT to local log;  
    else if DECISION == GLOBAL_ABORT  
        write GLOBAL_ABORT to local log;  
    } else {  
        write VOTE_ABORT to local log;  
        send VOTE_ABORT to coordinator;  
    }  
}
```

5.1. Two-Phase Commit - 2PC (4/5)

38

- Each participant should be prepared to accept requests for a global decision from other participants

Actions for handling decision requests: /* executed by separate thread */

```
while true {
    wait until any incoming DECISION_REQUEST is received; /* remain blocked */
    read most recently recorded STATE from the local log;
    if STATE == GLOBAL_COMMIT
        send GLOBAL_COMMIT to requesting participant;
    else if STATE == INIT or STATE == GLOBAL_ABORT
        send GLOBAL_ABORT to requesting participant;
    else
        skip; /* participant remains blocked */
}
```

5.1. Two-Phase Commit - 2PC (5/5)

39

- **Coordinator solution**

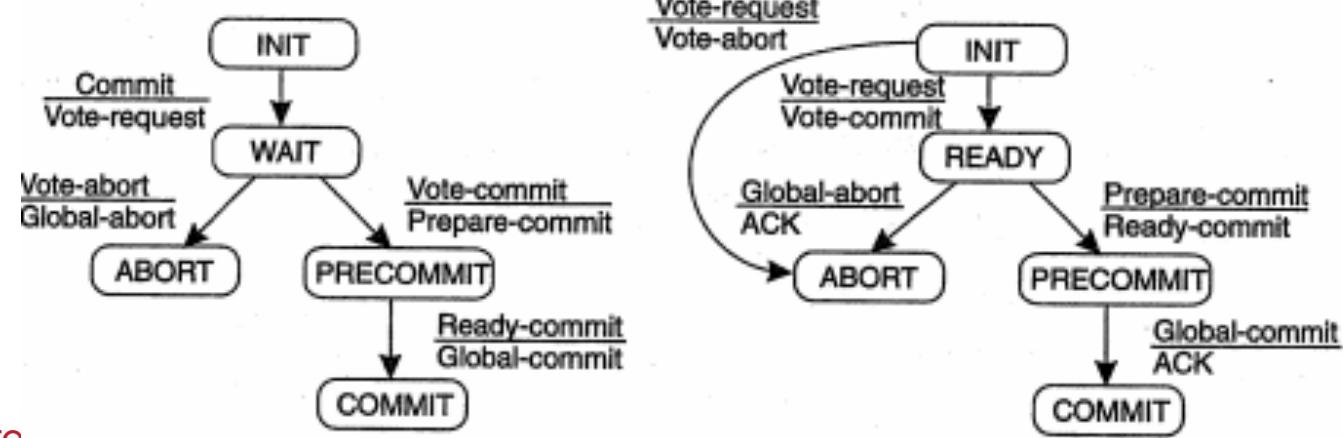
- Keep track of current state
- Sample of actions taken in place by the coordinator:

```
write START_2PC to local log;  
multicast VOTE_REQUEST to all participants;  
while not all votes have been collected {  
    wait for any incoming vote;  
    if timeout {  
        write GLOBAL_ABORT to local log;  
        multicast GLOBAL_ABORT to all participants;  
        exit;  
    }  
    record vote;  
}  
if all participants sent VOTE_COMMIT and coordinator votes COMMIT {  
    write GLOBAL_COMMIT to local log;  
    multicast GLOBAL_COMMIT to all participants;  
} else {  
    write GLOBAL_ABORT to local log;  
    multicast GLOBAL_ABORT to all participants;  
}
```

5.2. Three-Phase Commit (1/2)

40

- Two-phase problem: when the coordinator has crashed, participants may not be able make final decision
- Three-phase commit protocol (3PC) avoids blocking processes in when fail-stop crashes.
- Principle:
 - There is no single state from which it is possible to make a transition directly to either a COMMIT or an ABORT state
 - There is no state in which it is not possible to make a final decision, and from which a transition to a COMMIT state can be made
- Illustration



5.2. Three-Phase Commit (2/2)

41

- Actions taken by Participant in different cases

State of Participant P	State of Participant Q	State of all other participants	Action
INT			VOTE_ABORT
READY	INT		VOTE_ABORT
READY	READY	READY	VOTE_ABORT
READY	PRECOMMIT	PRECOMMIT	VOTE_COMMIT
PRECOMMIT	READY	READY	VOTE_ABORT
PRECOMMIT	PRECOMMIT	PRECOMMIT	VOTE_COMMIT
PRECOMMIT	COMMIT	COMMIT	VOTE_COMMIT

- Actions taken by Coordinator in different cases

State of Coordinator	Action
WAIT	GLOBAL_ABORT
PRECOMMIT	GLOBAL_COMMIT



Main difference with 2PC: if any participant is in READY state, no crashed process will recover to a state other than INT, ABORT or PRECOMMIT



HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

6. Recovery

6.1. Introduction

6.2. Checkpointing

6.1. Introduction (1/2)

43

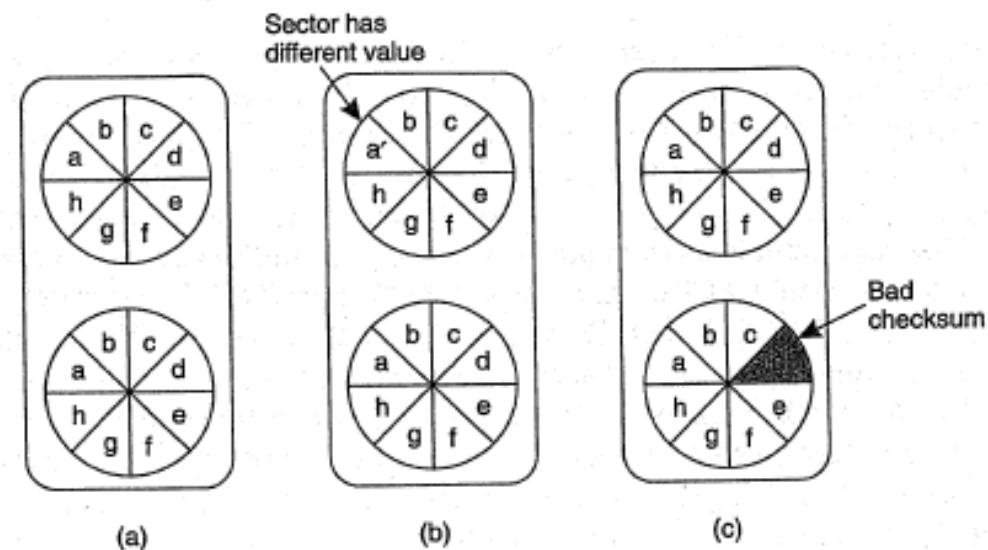
- **Backward recovery:** bring the system into a previously correct state.
 - Necessary to record the system's state, called checkpoint
 - Generally applied for recovering from failures in distributed systems
 - E.g. Reliable communication through packet retransmission
 - Drawback:
 - reduce performance
 - no guarantees that recovery has taken place
 - some states can never be rolled back to.
 - checkpoint could penalize performance and is costly
 - Solution for checkpoint: combine with message logging or use receiver-based logging
- **Forward recovery:** bring the system in a correct new state from which it can continue to execute
 - E.g. Erasure correction- a missing packet is constructed from other; successfully delivered packets

6.1. Introduction (2/2)

44

- **Stable Storage**
 - Information needed to enable recovery is safely stored in case of process crashes, site failures or various storage media failures
 - Three categories of storage: RAM memory, disk storage and stable storage
 - Sample of stable storage implementing with a pair of ordinary disk

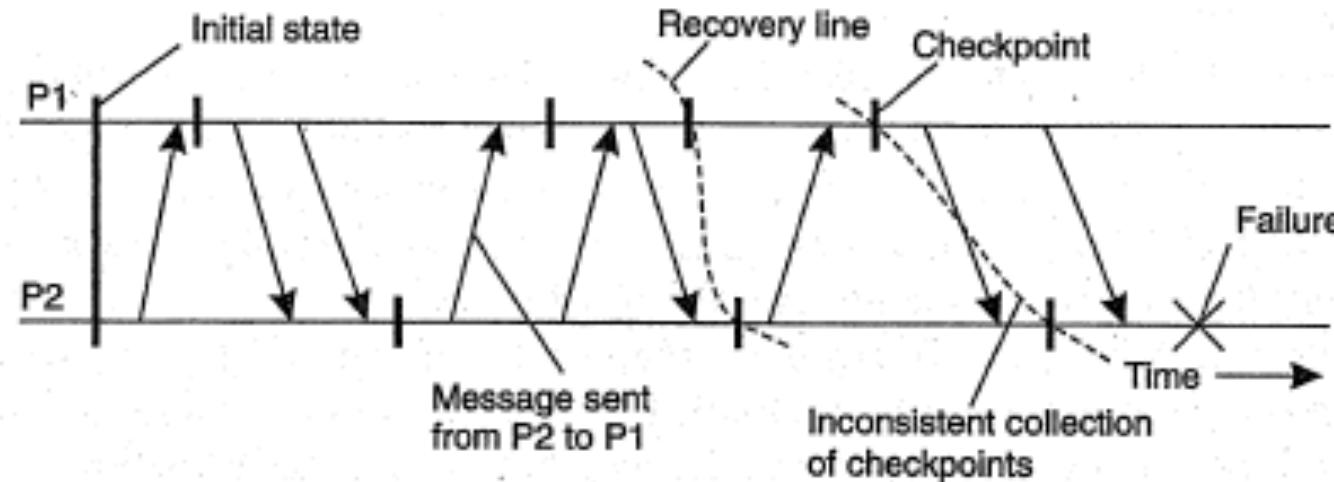
- (a) Stable storage
- (b) Crash after drive
1 is updated
- (c) Bad spot



6.2. Checkpointing (1/3)

45

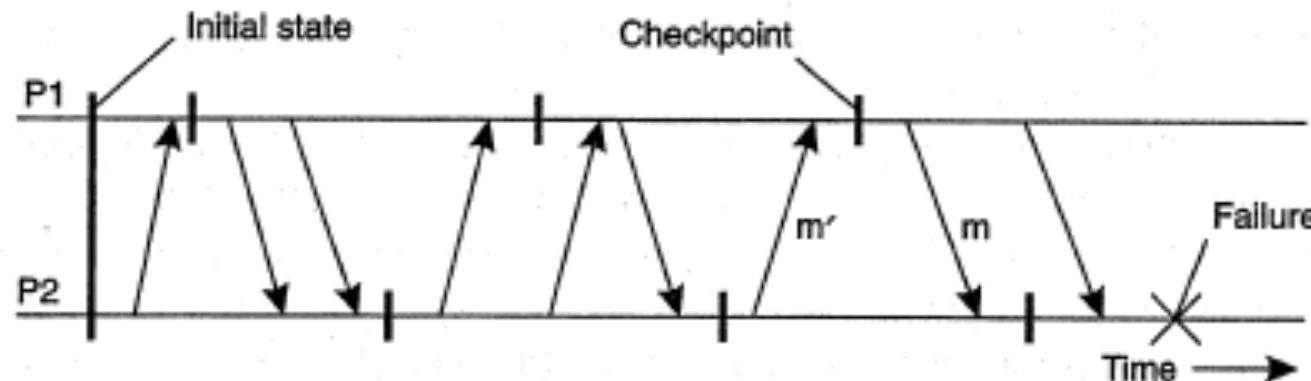
- **Distributed snapshot:** record a consistent global state.
 - If a process P records the receipt of a message, then there should also be a process Q that has recorded the sending of that message.
 - **Recovery line:** recover to the most recent distributed snapshot



6.2. Checkpointing (2/3)

46

- **Independent Checkpointing**
 - Domino effect: process to find a recovery line via cascaded rollback



- **Independent checkpointing:** processes take local checkpoints independent of each other.
- **Disadvantages:** Introduction of performance problem, need of periodical cleaning for local storage, difficult problem in computing the recovery line



25 YEARS ANNIVERSARY
SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Questions?

