Time and Coordination in Distributed Systems

Hong-Linh Truong
Distributed Systems Group,
Vienna University of Technology

truong@dsg.tuwien.ac.at
dsg.tuwien.ac.at/staff/truong

What is this lecture about?

- Understand the time synchronization problems
- Understand some basic algorithms for synchronizing times
- Understand why we do need logical clocks
- Understand basic methods for coordination

Learning Materials

- Main reading:
  - Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2e, (c) 2007 Prentice-Hall
    - Chapter 6
  - Roberto Baldoni, Michel Raynal: Fundamentals of Distributed Computing: A Practical Tour of Vector Clock Systems. IEEE Distributed Systems Online 3(2) (2002) http://www.dis.uniroma1.it/~baldoni/baldoni-112865.pdf
  - George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair„Distributed Systems – Concepts and Design“, 5nd Edition
    - Chapter 14 & 15
  - Sukumar Ghosh, Distributed Systems: An Algorithmic Approach,Chapman and Hall/CRC, 2007, Chapters 6, 7, 11

Outline

- Clock synchronization
  - Physical clock
  - Logical clock
  - Vector Clock
- Distributed coordination
  - Mutual exclusion
  - Leader election
- Summary

PHYSICAL CLOCK SYNCHRONIZATION

Fact

- Most clocks have limited precision
  - Clock drift requires clock adjustment/correction
- Atomic clocks
  - Very accurate, almost no drift
  - But very expensive
- In a large-scale distributed system we cannot have many atomic clocks

→ Clocks provide different times (clock skew problem)!

Why do we need clock/time synchronization?

Documentation\Installation\Regulatory Compliance\NYSE

The New York Stock Exchange

The New York Stock Exchange has various regulations regarding the synchronization of clocks used for timestamping, particularly in regards to use of the Front End Systemic Capture (FESC) system.

NYSE Rules 123 and, in particular, 132A detail these requirements. NYSE Information Memo 03-26, June 10, 2003 specifies:

"New Rule 132A requires members to synchronize the business clocks used to record the date and time of any event that the Exchange requires to be recorded. The Exchange will require that the date and time of orders in Exchange-listed securities to be recorded. The Rule also requires that members maintain the synchronization of this equipment in conformity with procedures prescribed by the Exchange."

Specific NYSE Time Synchronization Requirements
Rule 132A contains two specific requirements:

- **Clocks Synchronized to Commonly Used Time Standard**
  All computer clocks and mechanical timestamping devices must be synchronized to a commonly used time standard, either the National Institute of Standards and Technology (NIST) or United States Naval Observatory (USNO) atomic clocks.
- **Synchronization must be maintained**
  Rule 132A also indicates that the member must ensure that their systems remain synchronized.

How to Use Domain Time II to comply with the NYSE Rule 132A Requirements

https://www.greyware.com/software/domaintime/instructions/quickstart/regulatory-nyse.asp

http://online.wsj.com/articles/regulators-traders-are-out-of-sync-1405295799

During an incident investigation, network administrators should be able to identify which internal hosts have communicated with which IP addresses and what type of traffic was generated. DNS queries, proxy activity, and unusual network activity, such as port scanning, are also important data that may be required in an incident investigation. System auditing features, log retention durations, and time synchronization should be properly managed.

Source https://ics-cert.us-cert.gov/sites/default/files/documents/Incident_Handling_Brochure_Nov2010a.pdf

- Some reasons
  - Accountability of processes
  - Consistency in processing messages
  - Validity of important messages
  - Fairness in processing requests

Real clock synchronization

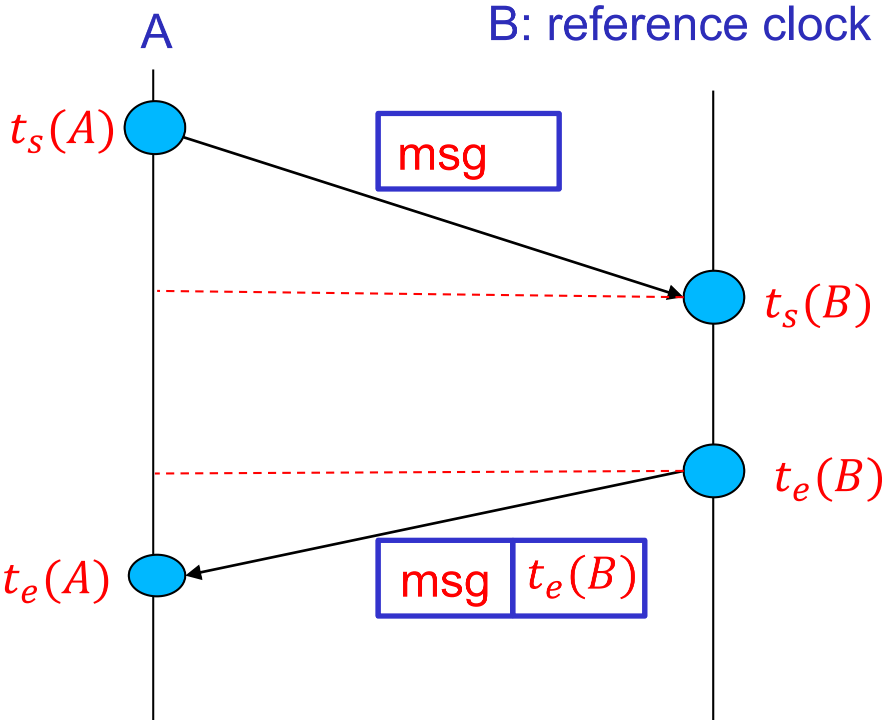
Challenging issue: it is impossible to guarantee timers/clocks in different computers due to the clock drift problem

- Establish/Decide reference physical clocks → to provide an accurate timing system
  - Coordinated Universal Time (UTC)
    - Based on atomic time produced by the most accurate physical clocks using atomic oscillators
- Operate/Utilize accuracy physical clocks providing UTC time
- Synchronize other physical clocks using time synchronization algorithms

Time provided by real physical clocks

- Computer clocks/timers
  - Every computer has a clock/timer
- Radio clocks receiving time codes via radio wave
  - Radio transmitter connects to an accuracy time source based on UTC time standard
- GPS (Global Positioning System) - a system of satellites, each broadcasts
  - its positions and the time stamps, based on its local time

Cristian's Algorithm



$$RTT = (t_e(A) - t_s(A)) - (t_e(B) - t_s(B))$$

The most simple case: Assume that times spent in sending messages are the same and that the processing time at B is 0 then

$$RTT = (t_e(A) - t_s(A))$$

Based on B's clock the message should arrive at A at

$$t'_e(A) = (t_e(B) + RTT/2)$$

A's clock:

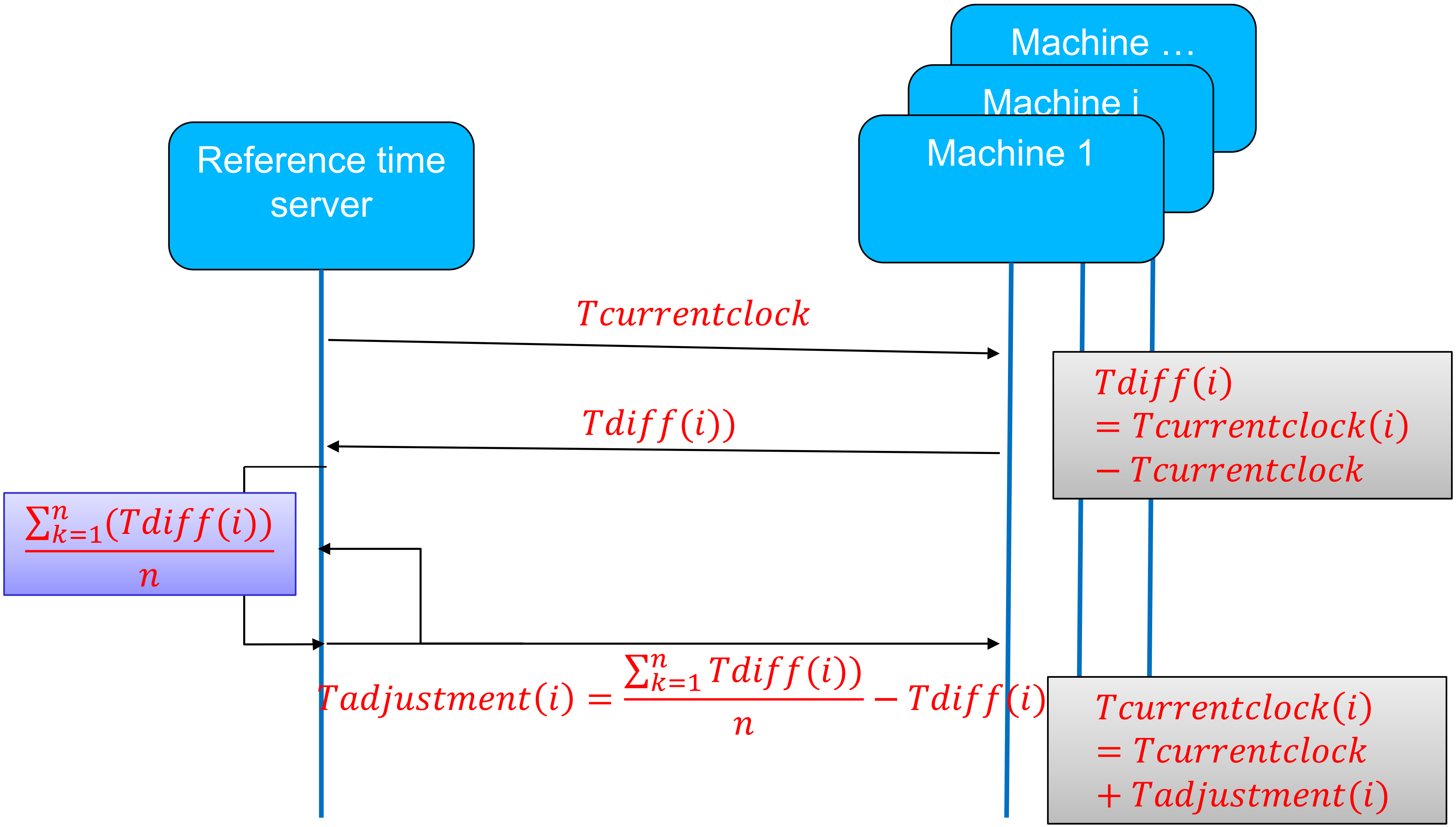
$$\max((t_e(A), (t_e(B) + \frac{RTT}{2}))$$

Q1: Drawbacks of this algorithm?

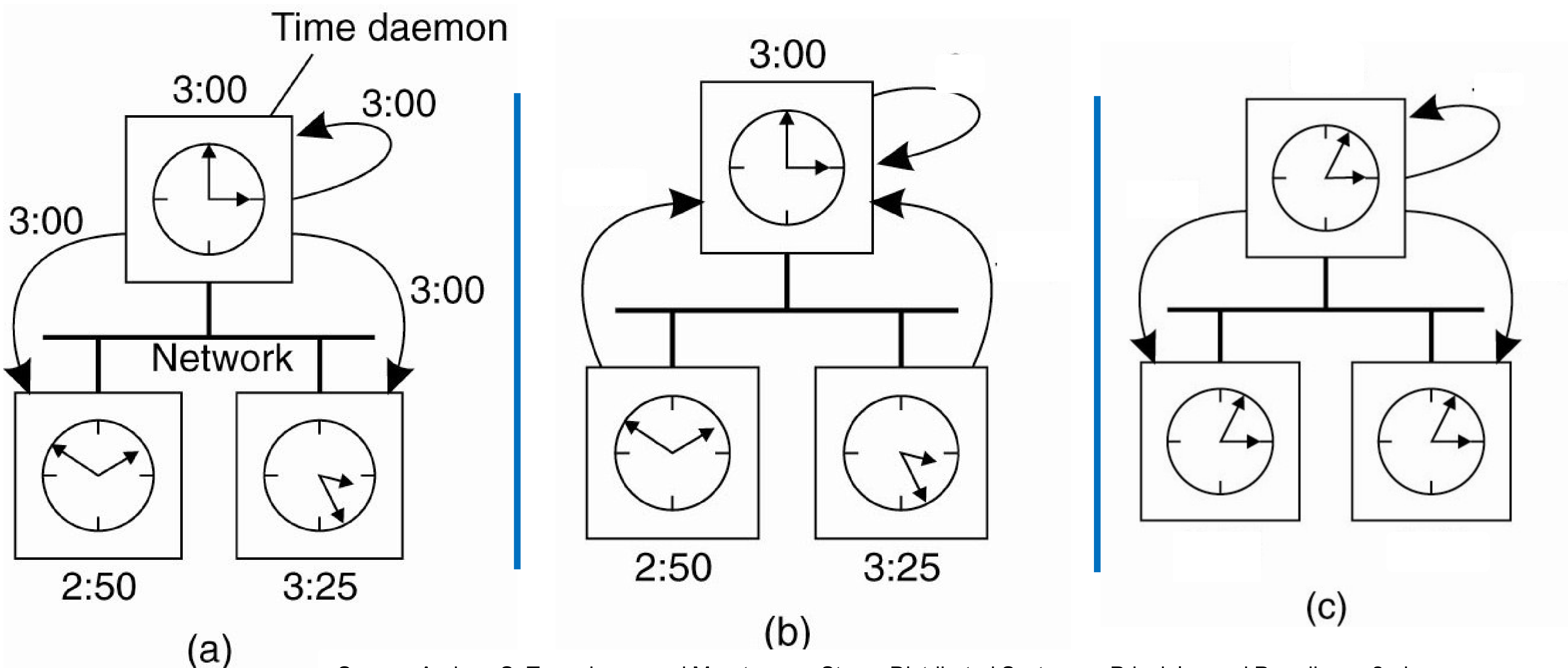
Q2: RTT is varying, how can the accuracy of this method be improved?

Homework: Assume we know the minimum time required for sending a message, Can you estimate the accuracy?

Berkeley Algorithm



Berkeley Algorithm

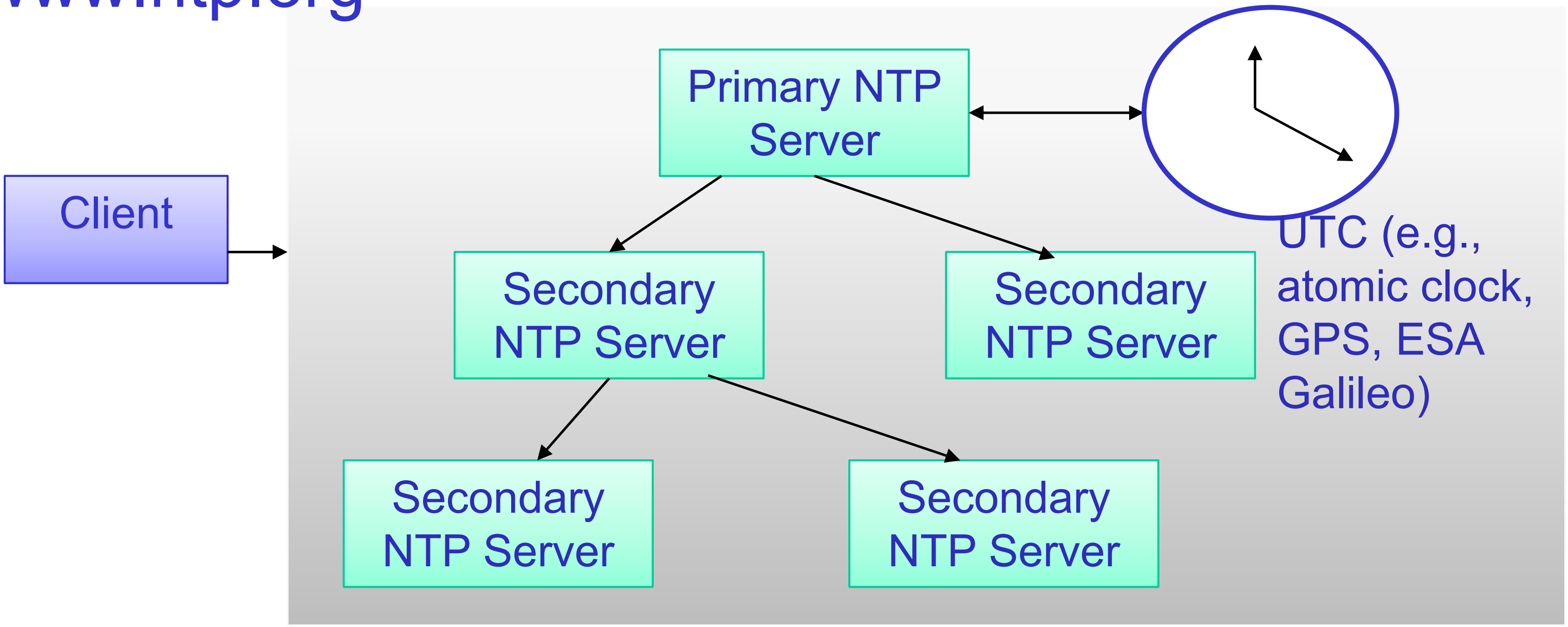


Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Q: Why is it not good to use this algorithm outside a LAN?

Example: Network Time Protocol (NTP)

www.ntp.org



Protocol variants using unreliable communication (UDP):

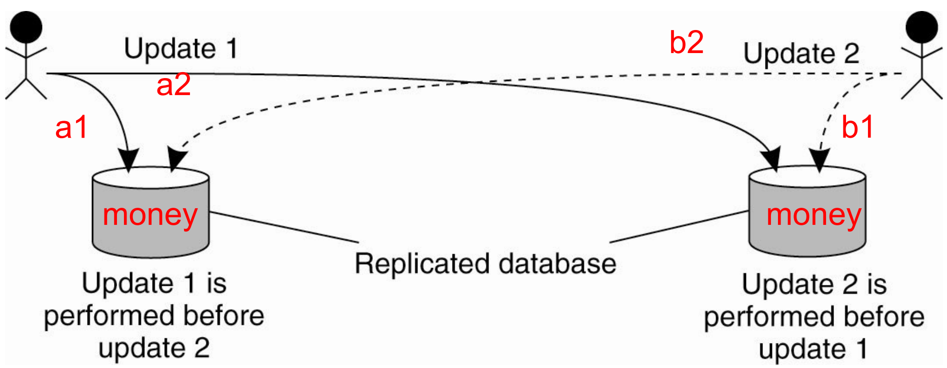
- Multicast (servers send the time), client/server (similar to Cristina's algorithm), symmetric (between high and lower level server)

LOGICAL CLOCKS

Logical clocks

- In many cases: we do not need an exact physical timing, as long as we able to maintain the physical causality

You withdraw your money



Your bank increases the interest for your money

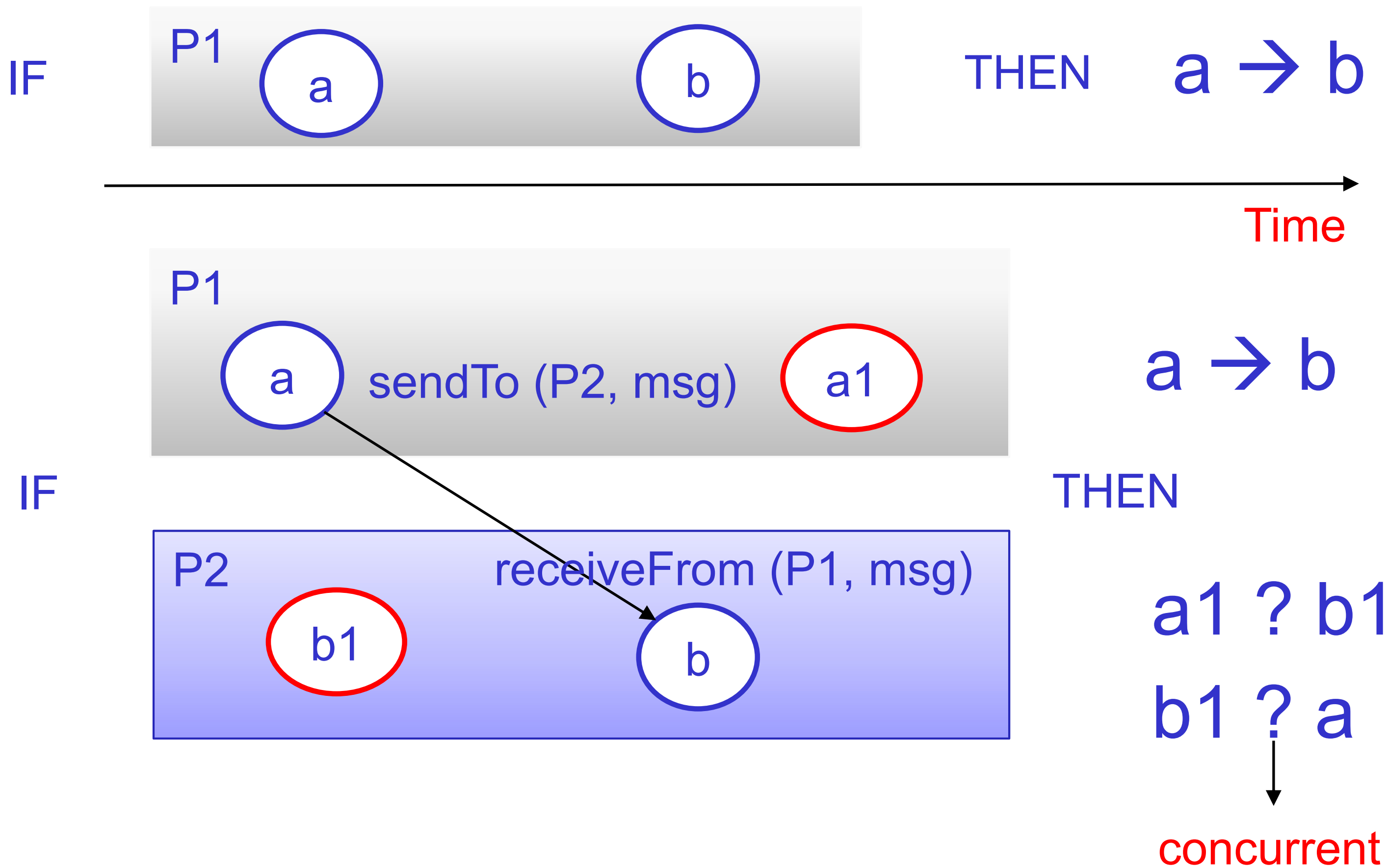
Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Intention:
We just need (a1,a2) being executed before (b1,b2) or another way around

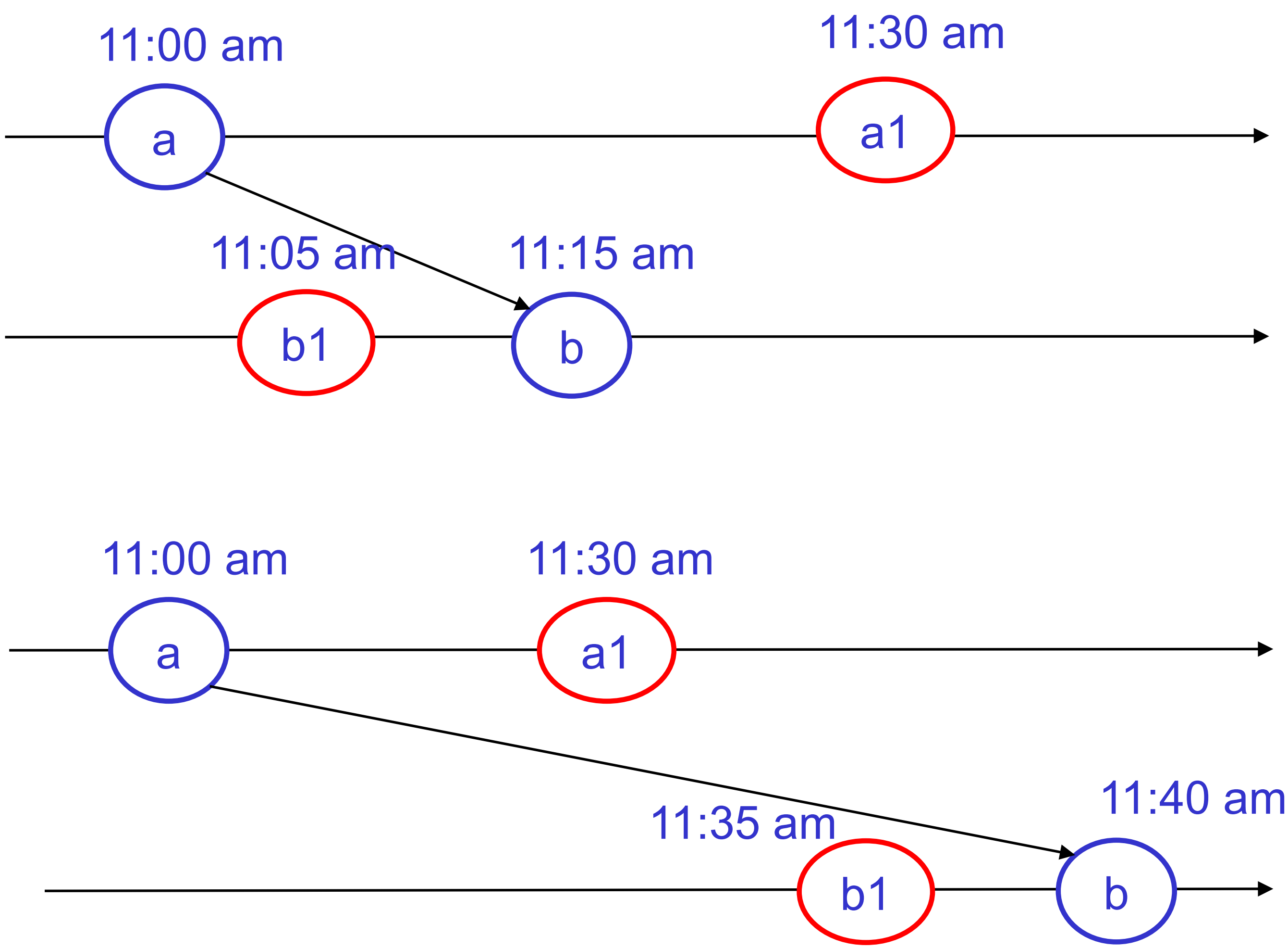
Logical clock: using physical causality model for ordering events among distributed processes

Happen-before relation

Happen-before (→) relation between a and b indicates that event a occurs before b logically. It is possible that a affects b



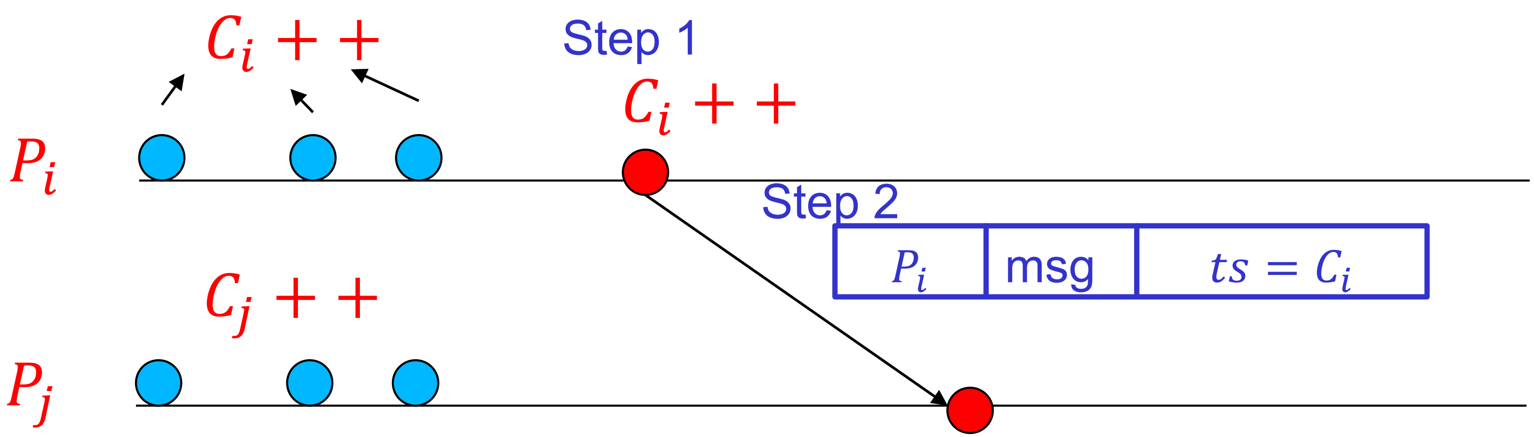
Example of concurrent events



Lamport‘s logical clock

- Used to synchronize a logical clock C_i of process P_i

Increase the clock before executing an event



Step 1 $C_j = \max(C_j, ts)$

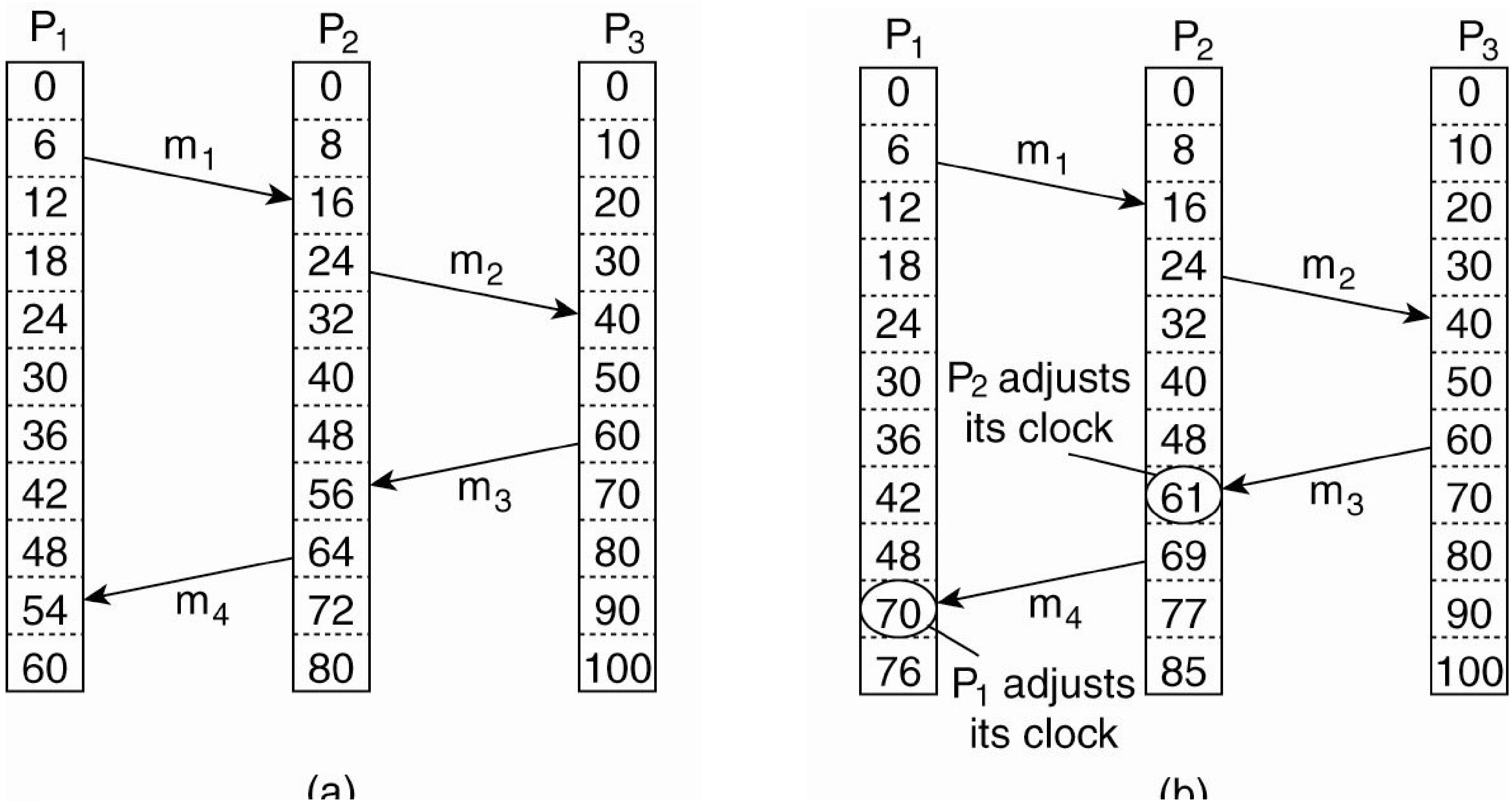
Step 2 C_j++

Step 3: process the message

Example of Lamport‘s logical clock

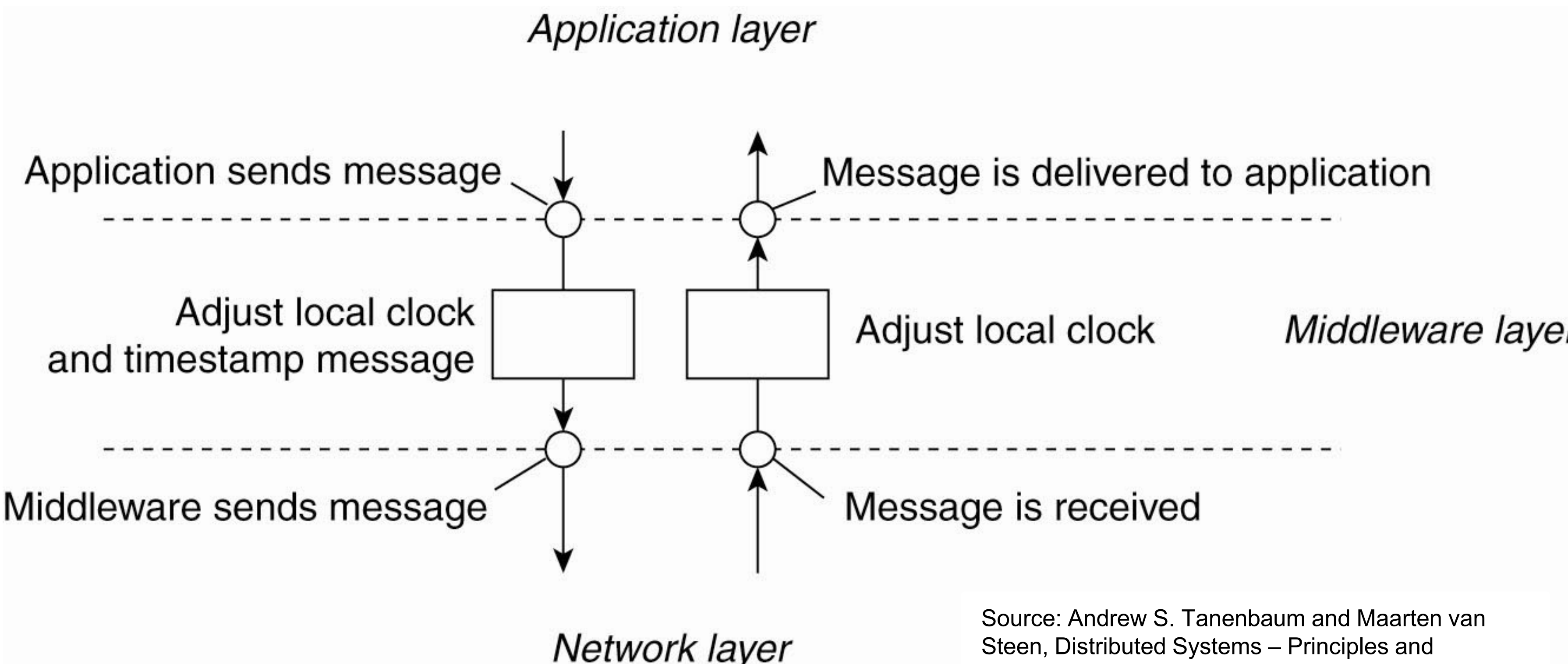
Without Lamport‘s logical clock

With Lamport‘s logical clock



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

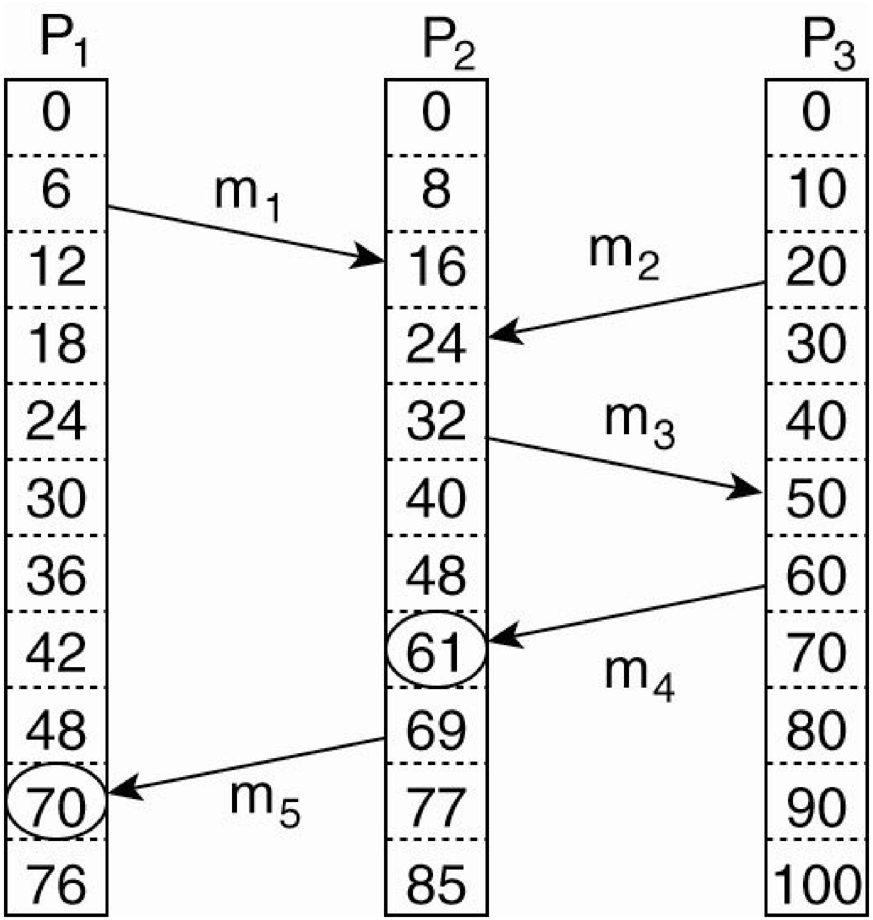
Message interception and logical clock adjustment implementation



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Home work: work out on in detail how Lamport‘s logical clock could be used for the update problem with replicated database

Limitation of Lamport's logical clock



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

recv(m4) < send(m5):

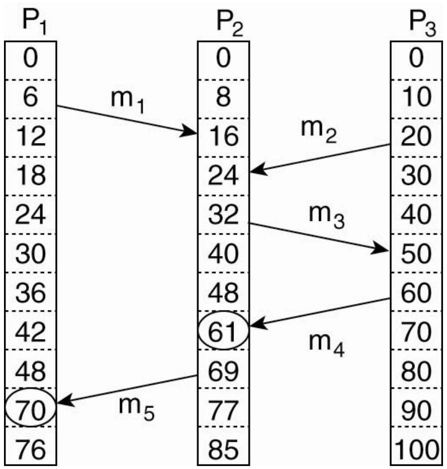
- Maybe m5 is dependent on m4 (causality)

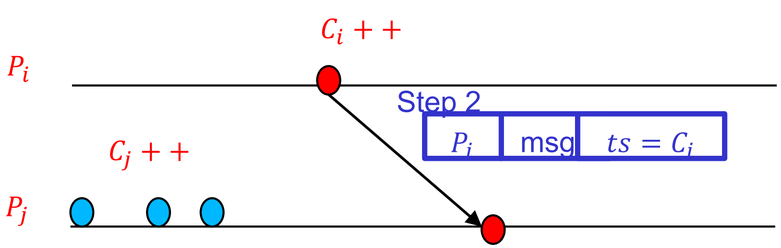
recv (m1) < send (m2):

- We do not know their relationship

C(a) < C(b) =! a → b, We miss causality information

Limitation of Lamport's logical clock





Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

The main problem

- A process does not keep track of (possibly causal) events happening in other processes

The solution

- A process should also maintain information about causal events occurring in other processes

Vector Clocks

Goal: a vector clock (VC) allows us to interpret if VC(a) < VC(b) then a causally precedes b

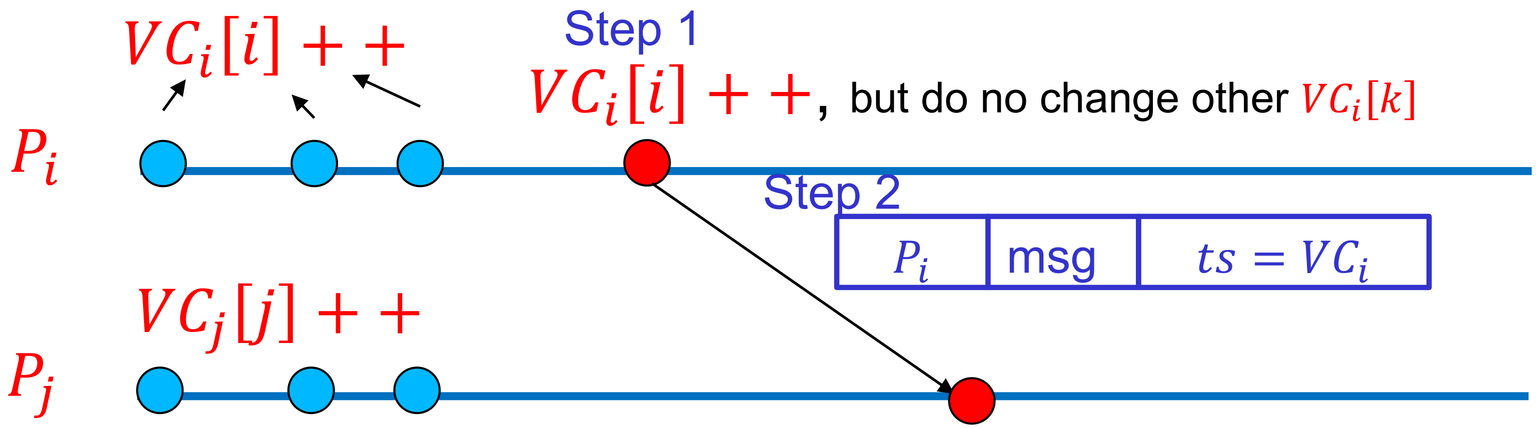
A process P_i maintains a vector clock VC_i where

- $VC_i[i]$ is the number of events happened in P_i
- $VC_i[j] = k$ means that P_i knows there were k events occurred in P_j that have causal relation with P_i

Implementation

- Each message is associated with a VC
- For event a and b, it is possible that a affects b, then a.VC <b.VC

Vector Clocks

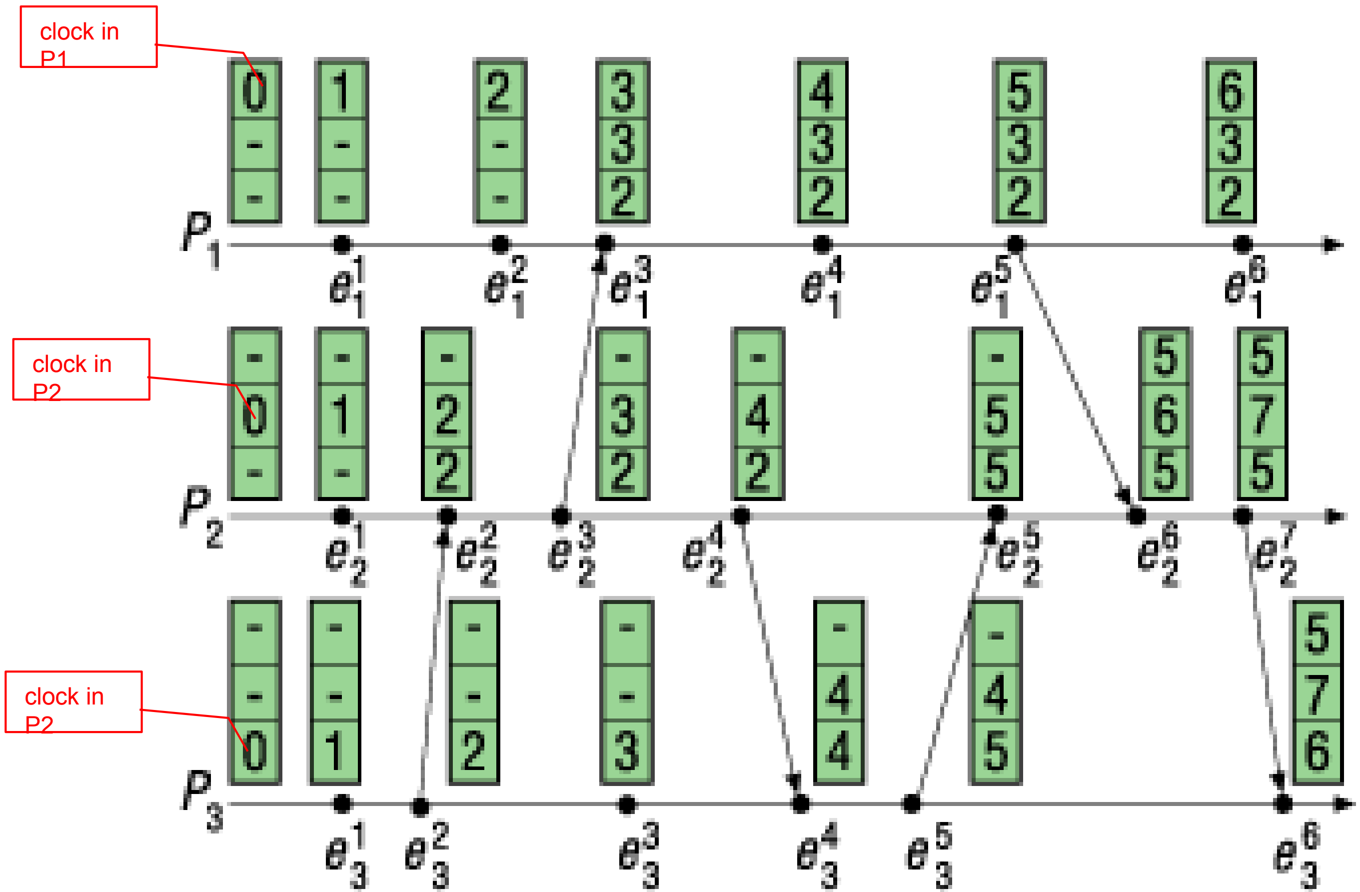


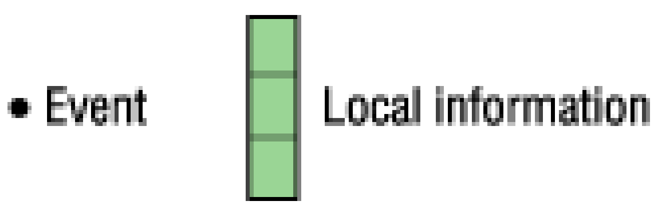
Step 1 $VC_j[k] = \max(VC_j[k], ts[k])$

Step 2 $VC_j[j]++$

Step 3: process the message

Example of vector clocks





Source: Roberto Baldoni, Michel Raynal: Fundamentals of Distributed Computing: A Practical Tour of Vector Clock Systems. IEEE Distributed Systems Online 3(2) (2002)

Applications of logical/vector clocks

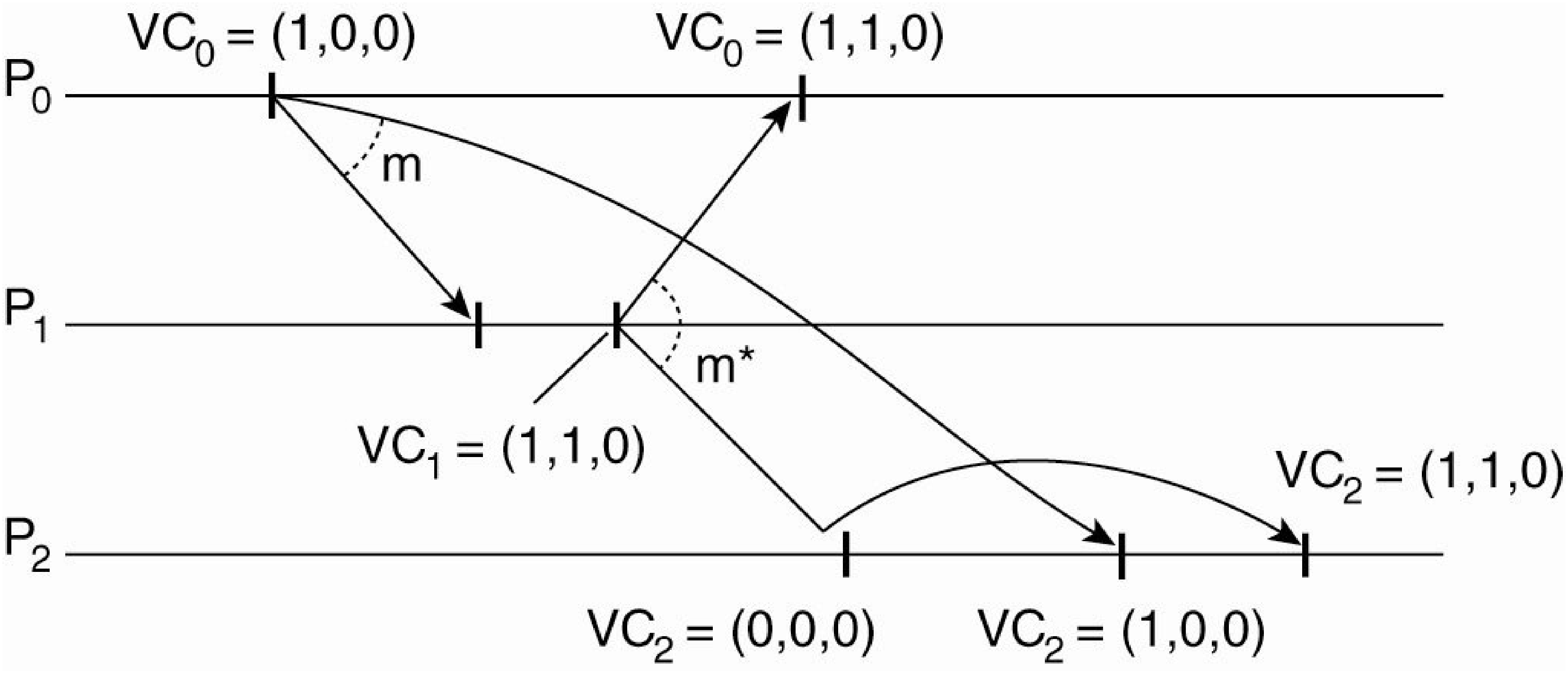
- Replication by using totally order multicast
  - atomic multicast in which all members accept messages in the same order
- Multimedia real-time applications, teleconferencing using causal multicast
  - If multicast(m1) → multicast(m2), then (m1) must be delivered before m2 for all processes

Causal broacast example

Assumption:

Upon sending a message P_i *only increases* $VC_i[i]$ *by* 1

When receiving a message only adjust $VC_j[k]$ *to* $\max(VC_j[k], ts[k])$

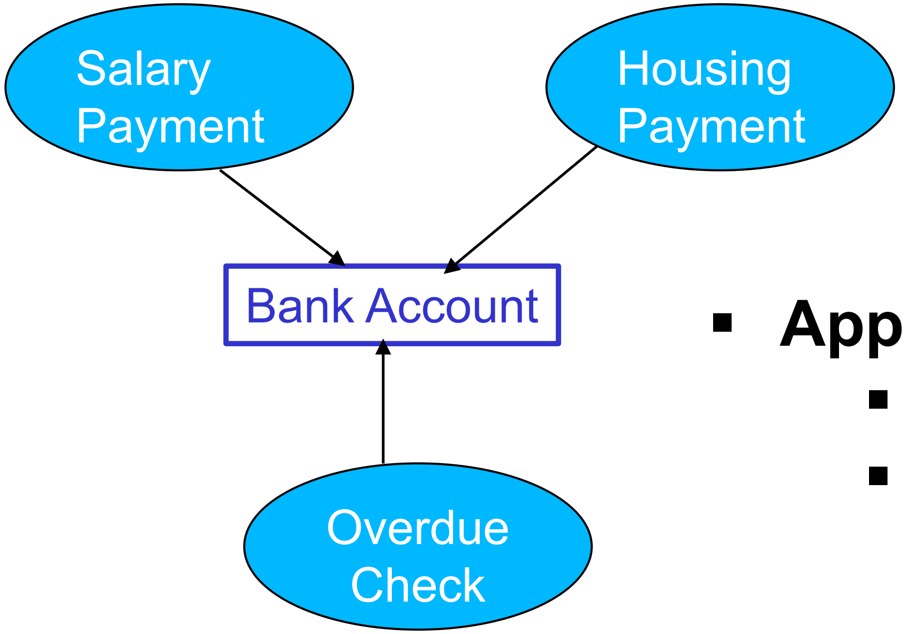


Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

MUTUAL EXCLUSION

Mutual exclusion in distributed systems

- Multiple processes might access the same resource
- Mutual exclusion: prevent them to use the resource at the same time to avoid making resource inconsistent/corrupted



- **Approaches:**
    - Token-based
    - Permission-based

Centralized Model

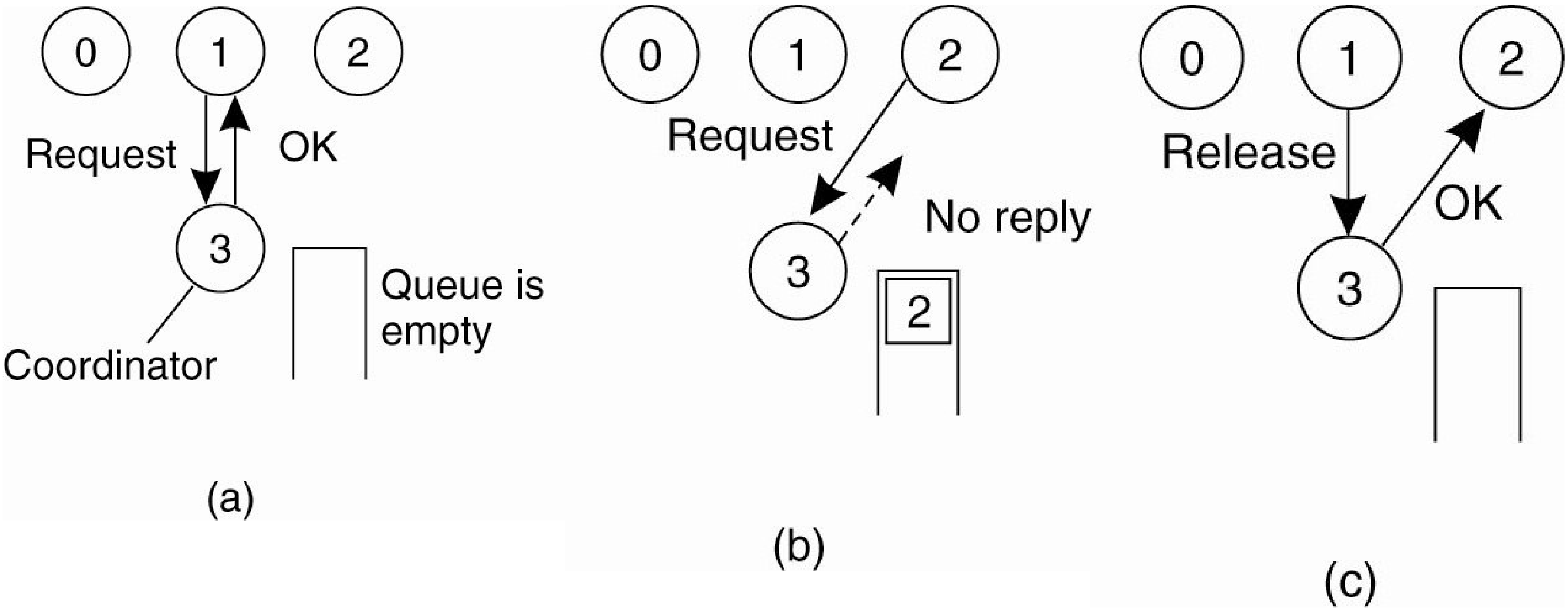
Permission-based approach: a dedicated server gives permission, emulating the execution of critical section

```
private static Lock lock =new ReentrantLock();
   public void criticalSection(){
       System.out.println("This is a critical section: access only with permission");
       System.out.println("======== I am "+id+" Waiting for the lock==========");
       lock.lock();
       System.out.println("I am "+id+" I got the lock now");
       System.out.println(id + " doing some work  ");
       try {
           Thread.sleep(5000);
       } catch (InterruptedException e) {
       }
       System.out.println("======== I am "+id+" releasing the lock============");
       lock.unlock();

   }
```

https://github.com/tuwiendsg/distributedsystemsexamples/tree/master/simplecentralizedmultualexclusion

Centralized Model



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Q1: What are the main problems with this centralized model?

Q2: How do you support access priority?

Homework: sketch a solution for time-based access priority

Example

- A very simple code
  - for a single resource using TCP communication
  - https://github.com/tuwiendsg/distributedsystemsexamples/tree/master/simplecentralizedmultualexclusion

java at.ac.tuwien.dsg.dsexamples.CentralizedMutualExclusion localhost 4001 no tuwien

→

java at.ac.tuwien.dsg.dsexamples.CentralizedMutualExclusion localhost 4001 yes null

Distributed algorithm (Ricart, Agrawala, Lamport)

Given a set of processes $\{P_1, P_2, \ldots, P_n\}$

If P_i wants to access a resource R, P_i broadcast a message msg(R, P_i, ts)

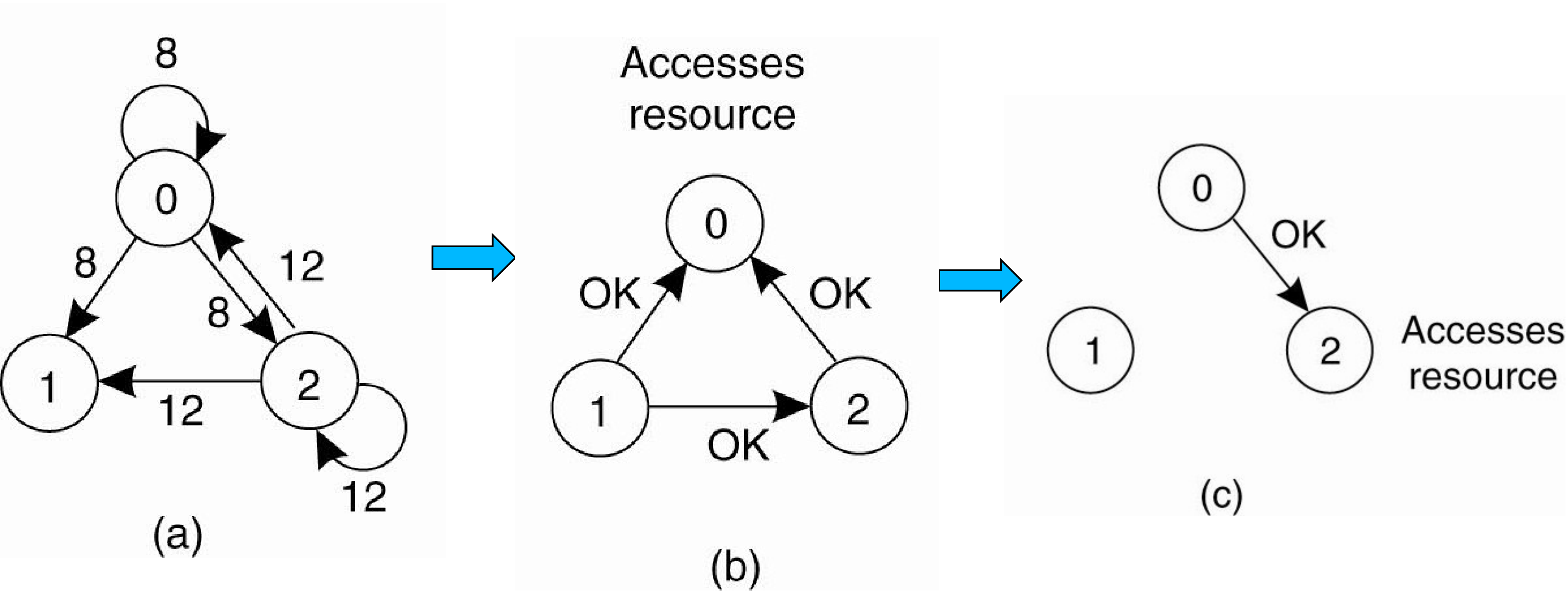
If P_j receives msg(R, P_i, ts) then

- No interest, no access → return „OK“
- Already accessing R then does not reply by putting the msg into the queue
- If already sent msg(R, P_j, tsj) but has not accessed R:
  - If ts < tsj then returns „OK“, otherwise put it in queue

If P_i gets all OK then it can access R, after that it sends an OK to all

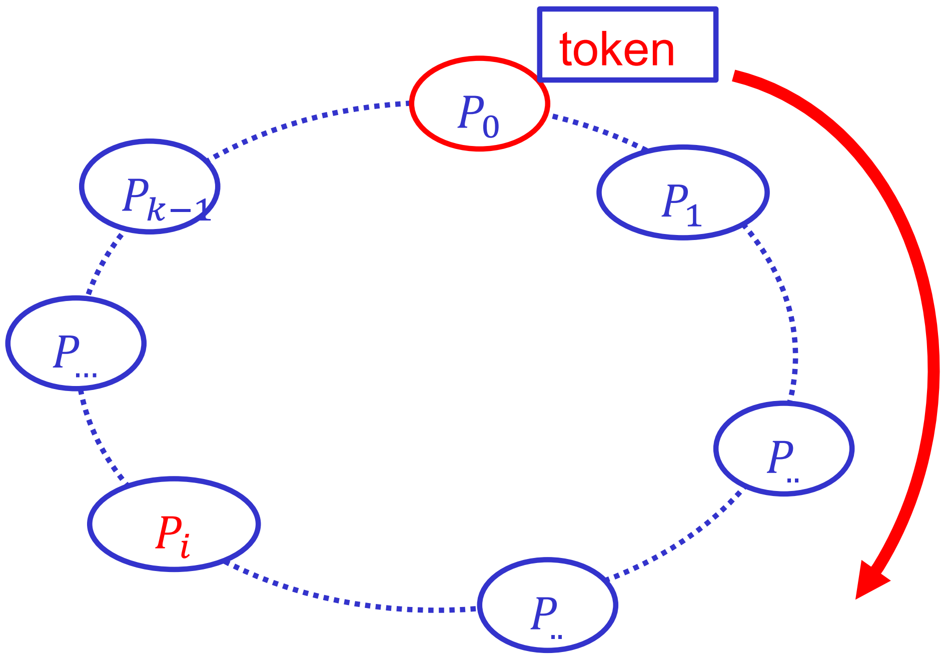
Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Example



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Ring algorithm



When P_i receives the token:

1. It accesses the resource and releases resource and passes the token
2. Otherwise it just passes the token

ELECTION ALGORITHMS

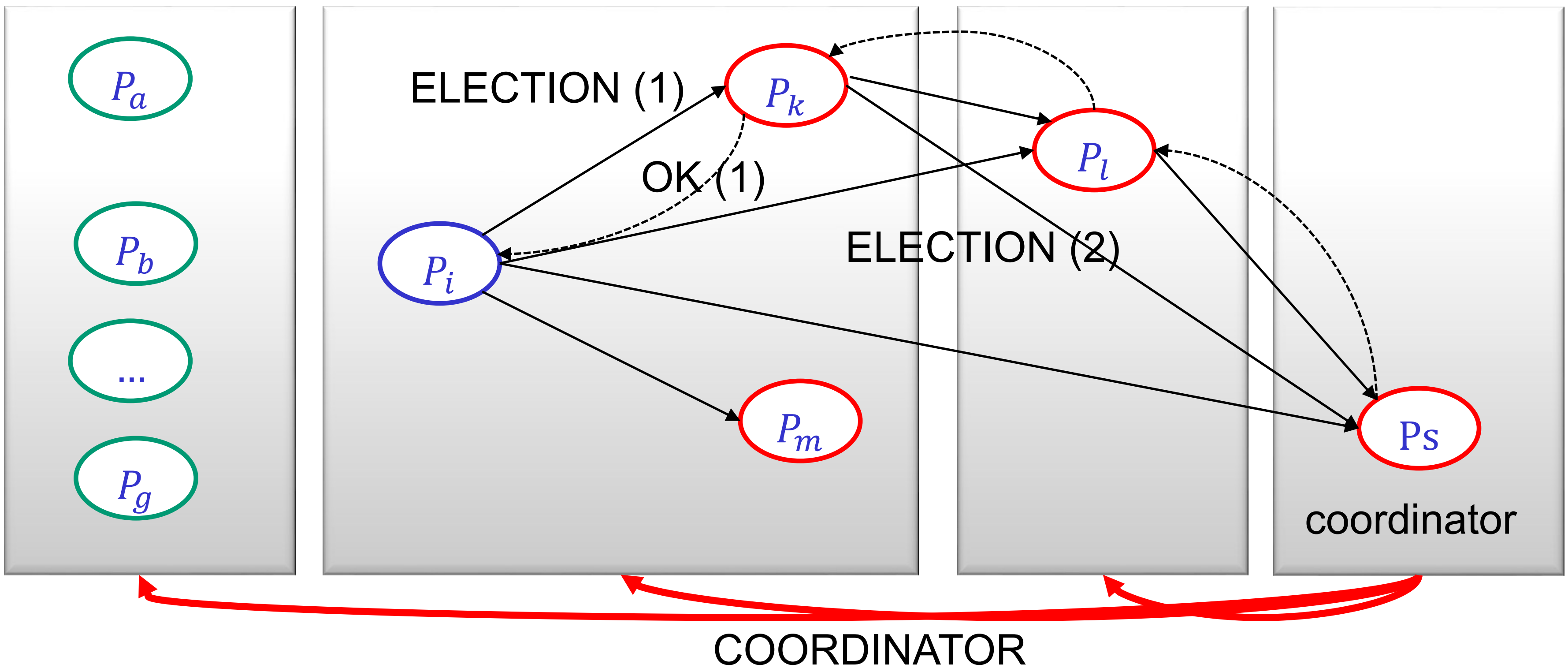
Leader election

- In many situations we need a coordinator
  - The coordinator is selected from a set of processes
- Why is it challenging to elect a coordinator?
  - Distributed, multiple processes involvement
- Election algorithms
  - Designed for electing leaders
  - Processes are uniquely identified, e.g., using process id
  - Process election occurs when
    - Initiating the systems, existing coordinator failed, etc.

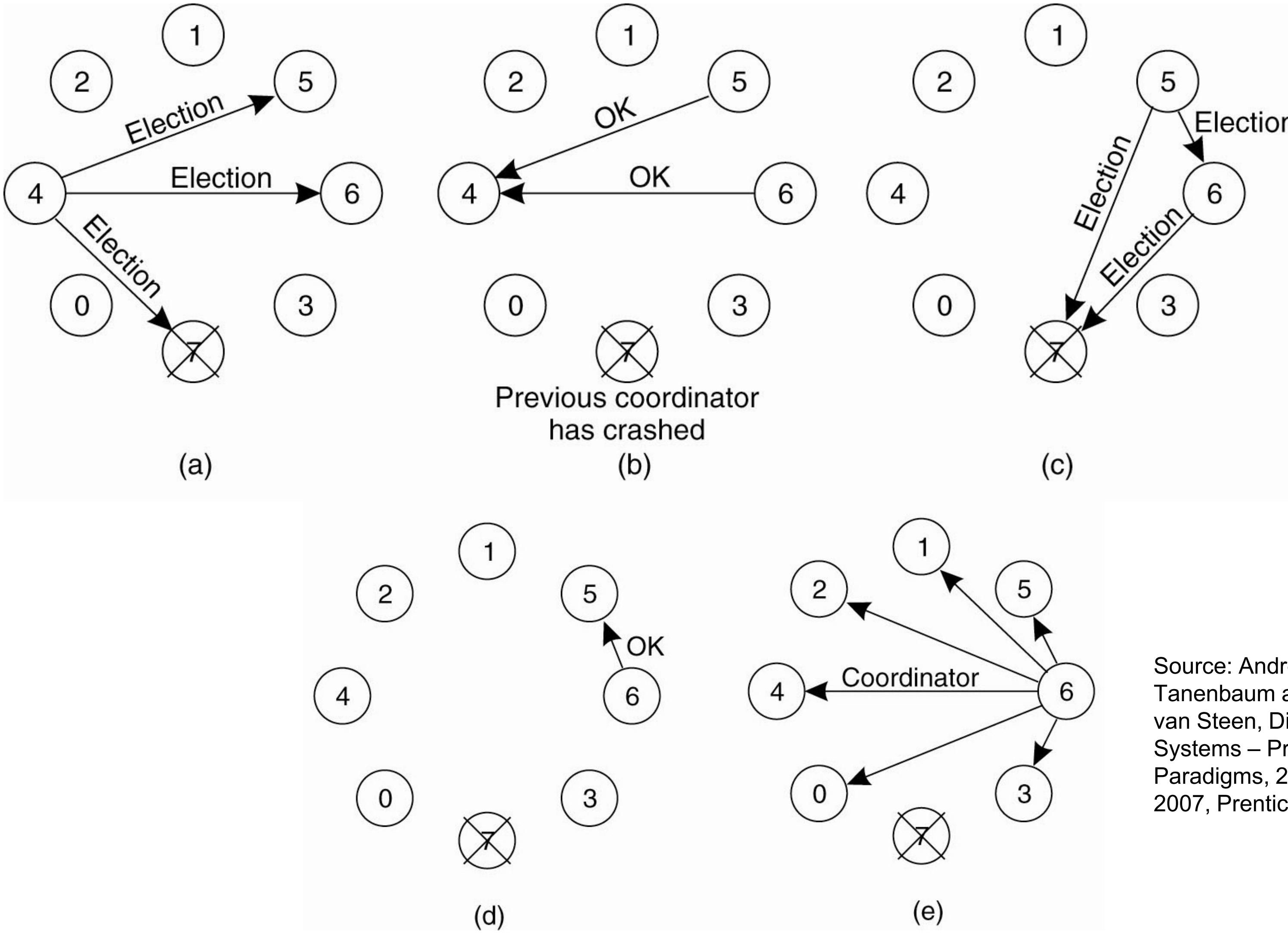
Bully algorithm

Lower rank processes

Higher rank processes



Example



Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

Ring algorithm

- From Le Lann, Chang and Roberts
- Processes are organized into a ring, initially „non-participant“ in the election
- Election message (ELECTION) and elected message (COORDINATION)
- Messages are forwarded or created and sent clockwise

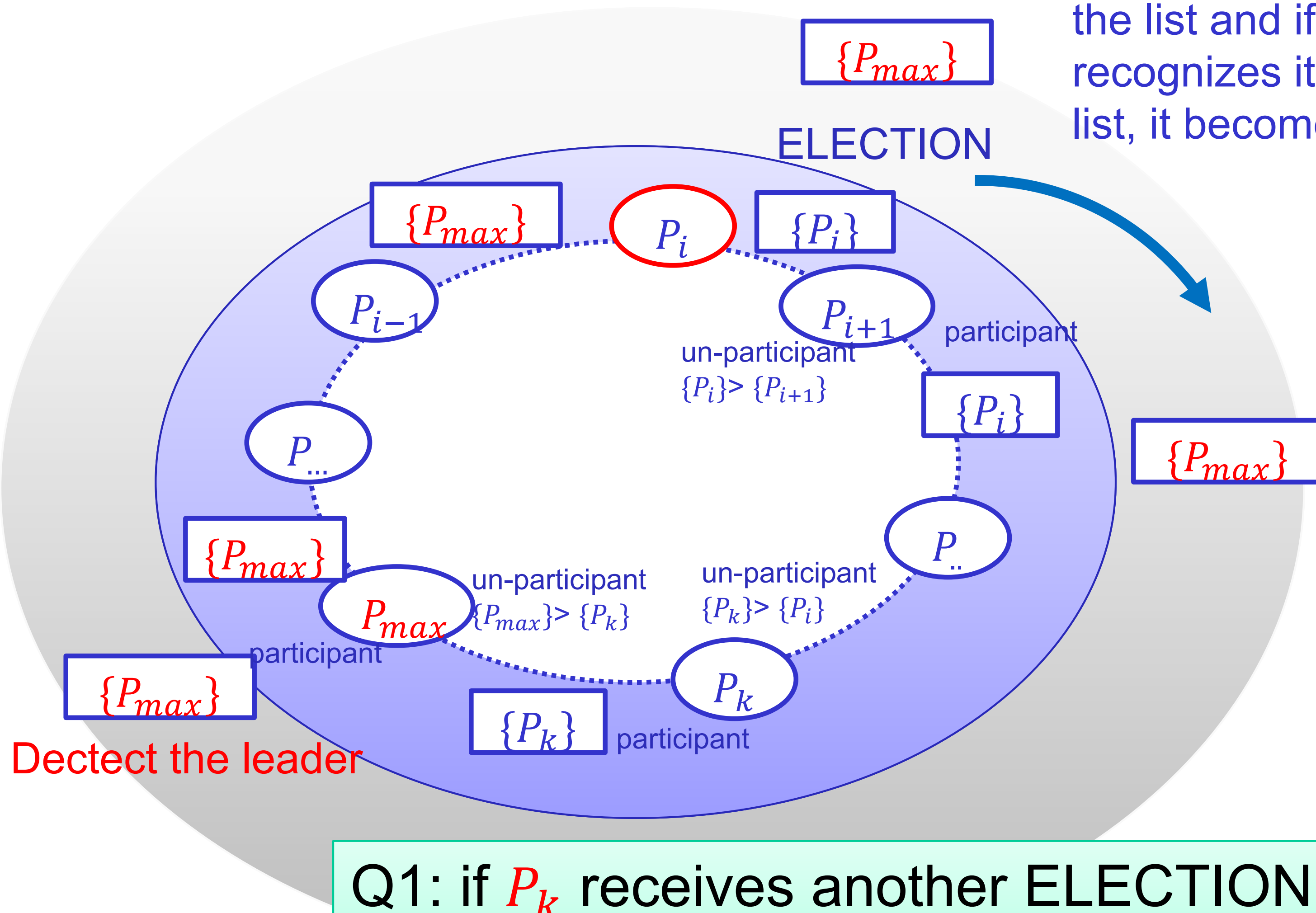
Source: Andrew S. Tanenbaum and Maarten van Steen, Distributed Systems – Principles and Paradigms, 2nd Edition, 2007, Prentice-Hall

George Coulouris, Jean Dollimore, Tim Kindberg, „Distributed Systems – Concepts and Design“, 2nd Edition, Chapter 10

Nancy A Lynch, Distributed Algorithms, 1996, Chapter 3.

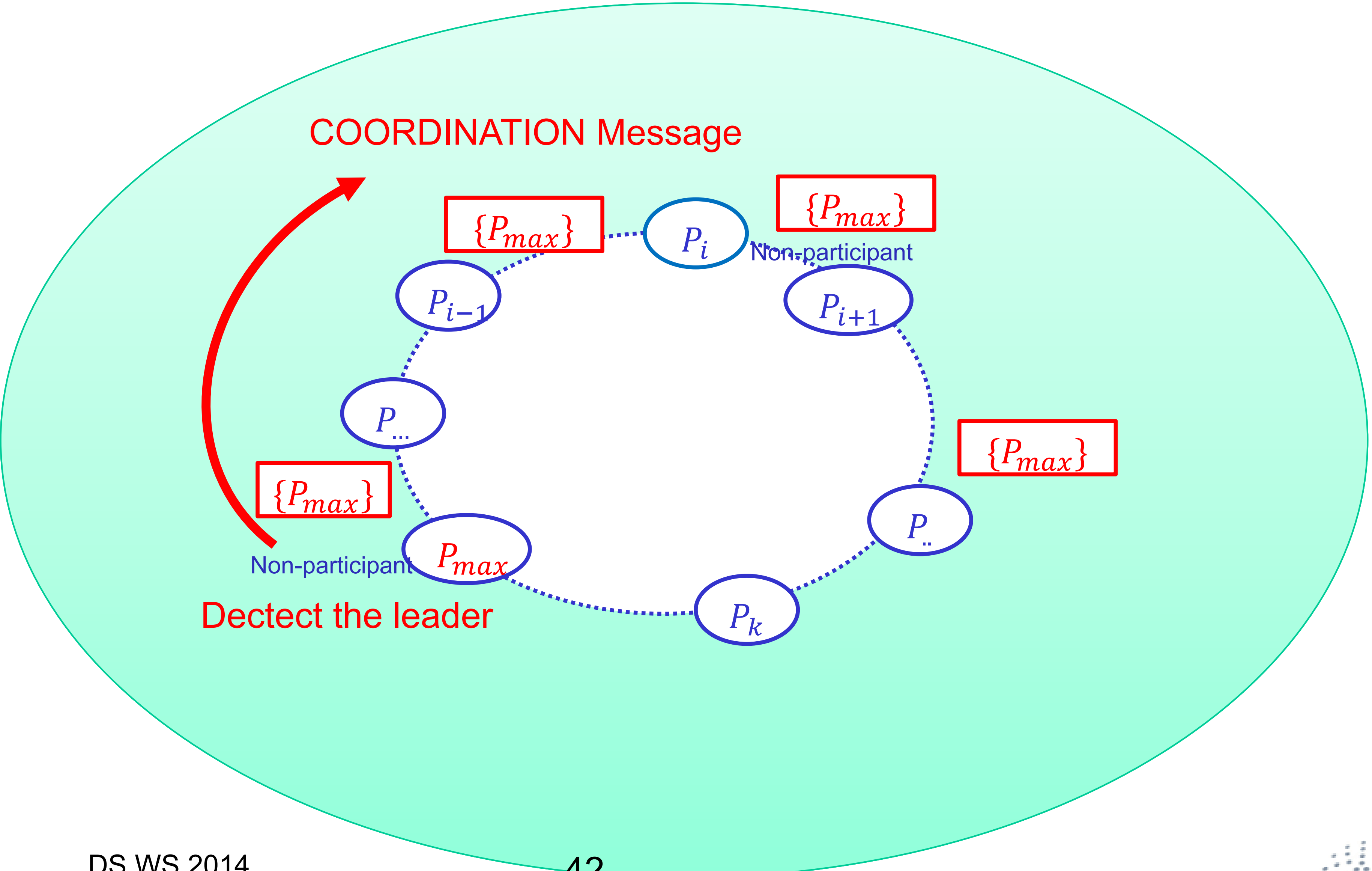
Ring algorithm

Another variant:
We do not compare and replace P_i but add P_i into the list and if a process recognizes its ID in the list, it becomes a leader



Q1: if P_k receives another ELECTION message with a smaller identifier after becoming participant, what should it do?

Ring algorithm



Simple Flooding Algorithm

Assumption: processes are structured into a directed graph

Steps

- P maintains the maximum unique process identifier (UID) it knows
- At a round, each P sends this UID to all nodes in its outgoing edges
- After n rounds, if a process P sees its ID equal to the maximum UID, then the process becomes the leader

Source: Nancy A Lynch, Distributed Algorithms, 1996, Chapter 4.

Summary

- Time synchronization is important in real-world
  - But a complex problem in distributed systems
  - Different algorithms with different pros and cons
- Logical clocks are useful in many situations
  - Happen-before or physical causality is the main principle
- Distributed coordination needs both mutual exclusion and election mechanism
- Dont forget to analyze algorithms to understand their pros and cons

Thanks for your attention

Hong-Linh Truong
Distributed Systems Group
Vienna University of Technology
truong@dsg.tuwien.ac.at
http://dsg.tuwien.ac.at/staff/truong