Distributed Computing

Synchronization

Synchronisation and Communication

- An important consideration in distributed systems is how processes cooperate and synchronize with each other.
- The correct behaviour of a concurrent program depends on synchronisation and communication between its processes.
- Synchronisation: the satisfaction of constraints on the interleaving of the actions of processes (e.g. an action by one process only occurring after an action by another).
- Communication: the passing of information from one process to another.
  - Concepts are linked since communication requires synchronisation, and synchronisation can be considered as content less communication.
  - Data communication is usually based upon either shared variables or message passing.
  - Rules for enforcing correct interaction=>synchronization mechanism

Issues of synchronization

- Clock synchronization
- Event ordering
- Mutual exclusion
- Deadlock handling
- Election algorithms

Issues of synchronization contd..

- Clock synchronization
  - Introduction
  - Clocks synchronization in distributed system
  - Drifting of clocks
  - Types of clock synchronization
  - Clock synchronization algorithms

Issues of synchronization contd..

Introduction

- In a non-distributed system, time is unambiguous.

  – When a process needs to know the time it makes a system call and the kernel returns the time maintained by the hardware.

- When multiple CPUS are present in a distributed system we can not count on all clocks running at exactly the same rate. (cpu utilization time, on _line reservation etc)
- Even if all clocks are initialized to exactly the same time at exactly the same instance, over time the clocks will differ – Due to clock skew because not all timers are exact.
- If it is not possible to synchronize all clocks in a distributed system to produce a single, unambiguous time standard, then can it be permissible to have distributed processes agree on a consistent view of logical time?

Implementation of computer clocks

❑Components of a computer clock

- Quartz crystal
- Counter register
- Constant register (based on the freq. of oscillation of the quartz crystal e.g; 60)

❑Drifting of clocks

the rate at which two clocks differ (10 raised to -6)

- Cp(t) = t for all p,t.

  where

  C - time value of clock **p**

  t – real time

- **a clock is non-faulty if** –

  (1-q <= dC/dt <= 1+ q)

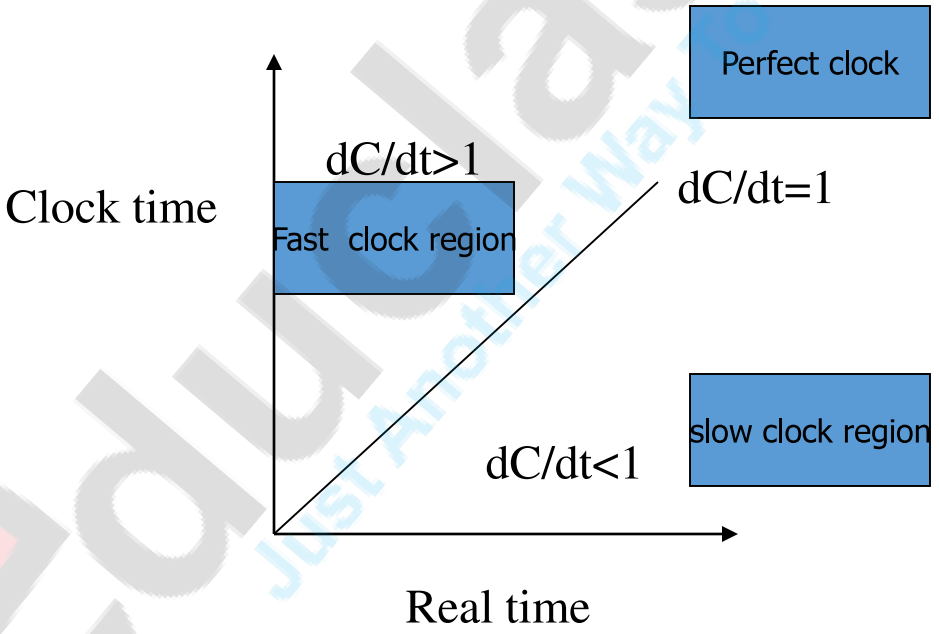
  where q is max drift rate allowable

Clocks should be resynchronized periodically by del/2q

del =specified constt

Clock skew =diff. in time values of the two clocks

Drifting of clocks



Types of clock synchronization in D.S

- Synchronization with real-time clocks (external) : are internally synchronized, **converse is not true**
  - UTC (coordinated universal time) is used as international std**.comparison** with radio, satellite etc.
  - used for real-time applications.
- Mutual Synchronization of clocks of different nodes of system (internal)
  - used across all nodes in distributed system for distributed activities.

Clock synchronization issues

- Max allowable time difference (del)
- Clock skew (difference in time between two clocks) < del
- Unpredictable communication delays
- Time must never run backwards (repetition of certain operations)
- Readjust the time gradually and not at once

➢Introduce the **Interrupt routine**

Clock synchronization algorithms
Centralized

- Time server node
- Passive time server centralized algo : server responds to requests from other nodes periodically (<del/2q)

  clock readjusted =T+(T1-T0-I)/2

- Active time server centralized algo : time server broadcasts time periodically , **clock readjusted =T+Ta;** drawback is that **it is not fault tolerant** (in case broadcast msg. is received late)

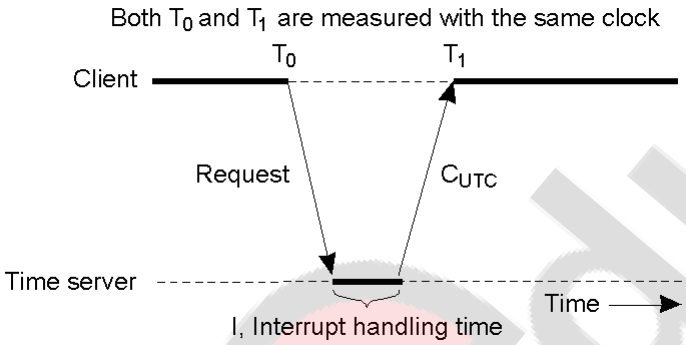
Drawbacks of centralized clock synchronization algorithms

- Single point of failure, unreliable
- Scalability

  Hence distributed algos

Clock synchronization algorithms Centralized contd..

Cristian's Algorithm



Getting the current time from a time server.

1. Client sends a request for a time to the server.
2. Server will send a message with time T.
3. Client will readjusted its clock with T+(T1-T0)/2.
4. With interrupt time as T+ (T1-T0-I)/2.

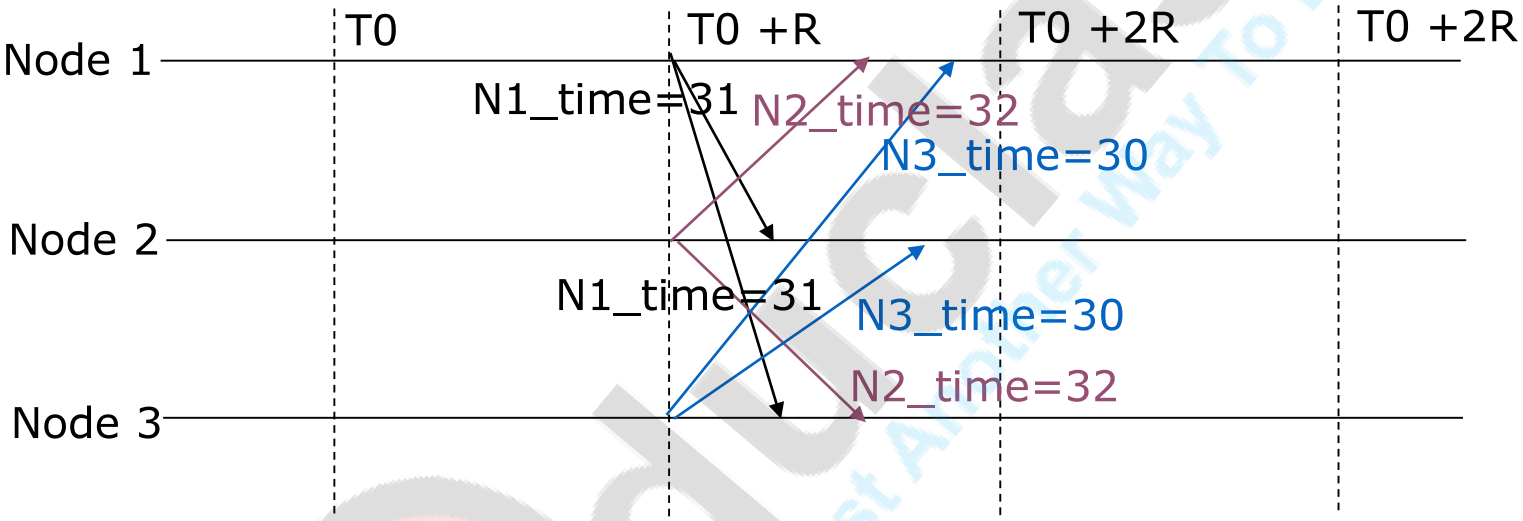
Clock synchronization algorithms Distributed

- Global averaging distributed algorithm
- Each node broadcast its local clock time = T0+iR where T0 is fixed time agreed upon by nodes is a system , R is a parameter such as no. of nodes, max drift rate, as **a "resync" message.**
- The clock process **collects all resync** messages.
- It **records the time**,according to its own clock..
- At the end of **waiting period**,the clock process **estimates the skew of its clock with respect to each of the nodes**.
- It then computes a **fault tolerant average of estimated skews and uses it to correct the local clock.**
- N/W traffic and broadcast facility are the issues

Clock synchronization algorithms Distributed contd..

- **Localized averaging distributed algorithm**
  - The nodes of a distributed system are logically arranged in some kind of **pattern, such as a ring or a grid.**
  - Each node exchanges its clock time with **its neighbors** in the ring.
  - Sets the local time as average of its own and neighbor clock times.

Clock synchronization algorithms Distributed Algorithm – Averaging Algorithm



- Assumption: R is large enough to wait for all broadcast messages
- All nodes broadcast their time periodically
- Each node computes average.
- Improvement:
  - Discard outlying time messages.
  - Exchange their time with their local neighbors.

Event ordering

- Lamport (1978) showed that clock synchronization need not be absolute.
- If two processes do **not interact**, it is **not necessary** that their clocks be synchronized because the **lack of synchronization would not be observable.**
- Lamport realized that it is not important that all processes agree on time, but rather, that they agree on the **order in which events occur**.

Event ordering contd..

❑Happened-Before Relation

- conditions are –

  1. If a and b are events in the same process and a occurs before b, then a → b.

  2. If a is the event of sending a message by one process and b is the event of the receipt of the same message by another process, then

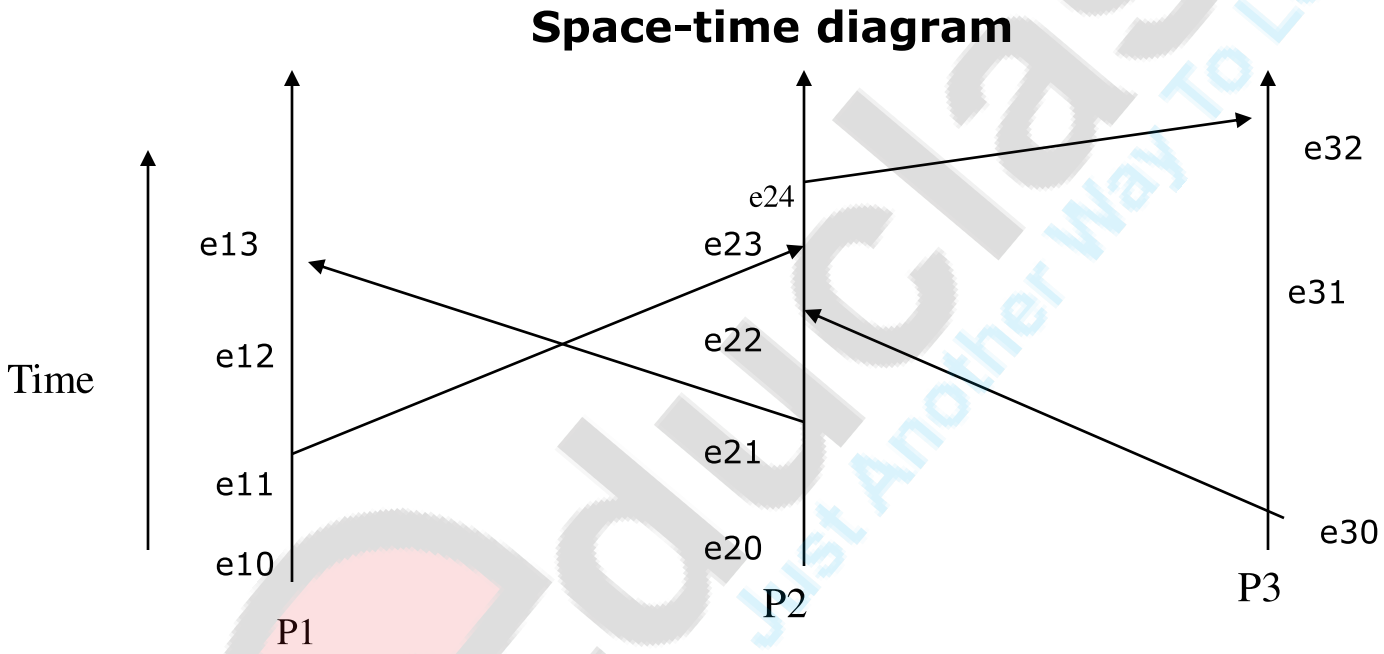
  a → b. (law of causality, sender to receiver is always positive)

  3. If a → b and b → c, then

  4 can a → a be true (transitive relation)

- **Concurrent events :** happened in two diff processes which are not related
- not related by the happened-before relation.

Event ordering contd..



Vertical lines = process

dots =event

Msg transfer, concurrent process ?

Event ordering contd..

- **Logical clock**

- To determine which event happened first, we need either common clock or set of perfectly synchronous clocks.
- Lamport provide a solution with **timestamp** for each event.
- Each process Pi has a clock Ci associated with it that assigns a number Ci(a) to any event a in that process.
- C(b) = Cj(b) if b is an event in process Pj.
- The time stamps assigned to the events by the system of logical clocks must satisfy clock condition

for two events a & b if a → b , then C (a) < C (b)

Event ordering contd..

- **Implementation of logical clocks**

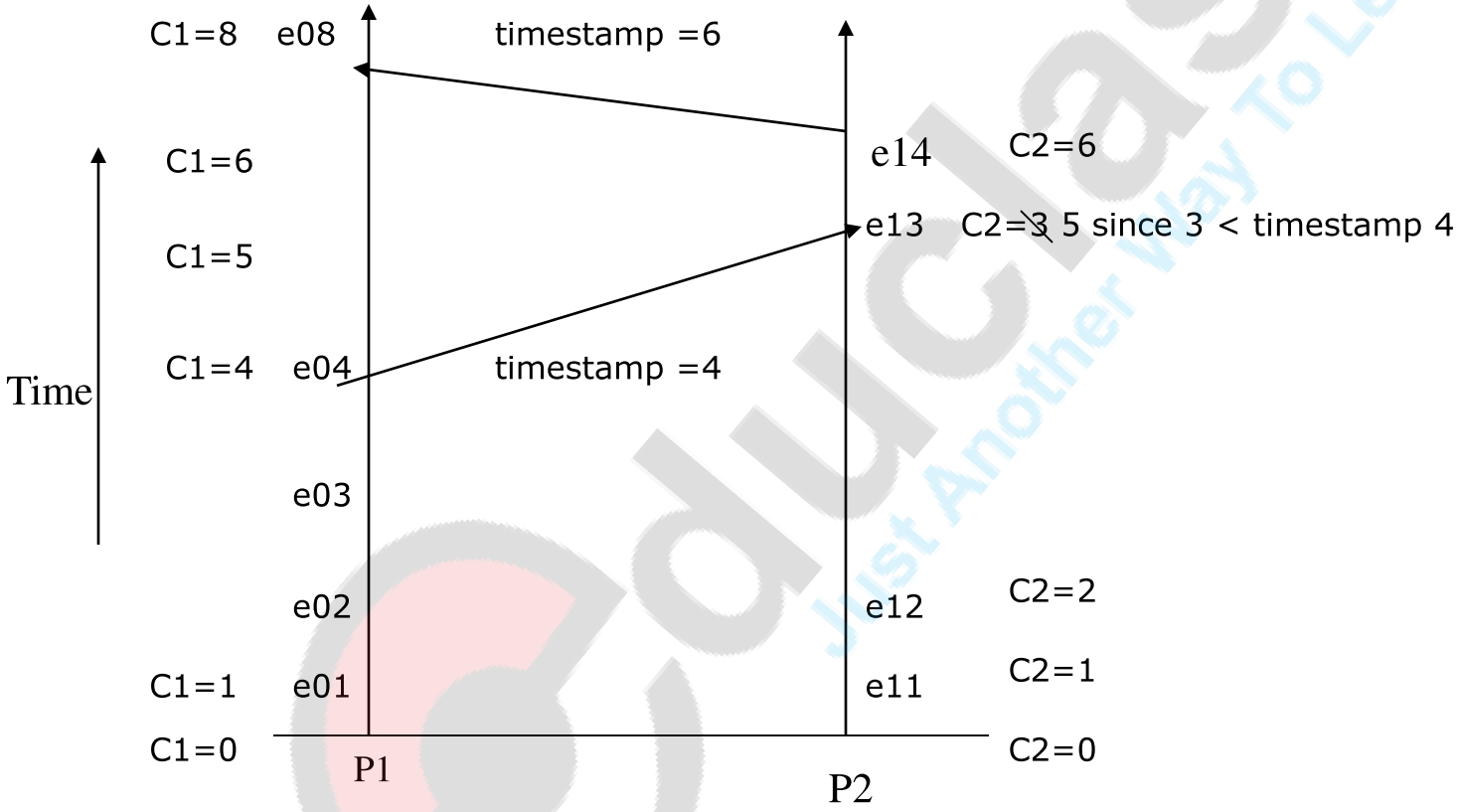
- C1: if a and b are two events within the same process Pi and a occurs before b,then Ci(a)<Ci(b).
- C2: if a is the sending of a message by process Pi and b is the receipt of that message by process Pj, then Ci(a)<Cj(b).
- C3: A clock Ci associated with a process Pi must always go forward, never backward, i.e. corrections to time of a logical clock must always be made by adding a positive value to the clock, never by subtracting value.
- IR1: each process Pi increments Ci between any two successive events**.=>c1**
- IR2: if event a is the sending of a message m by process Pi, the message m contains a timestamp Tm=Ci(a), and upon receiving the message m a process Pj sets Cj greater than or equal to its present value but greater than Tm**=>c2**

example given in the next slide

Event ordering contd..

- **Implementation of logical clocks.. by using counters**
- The counters are initialized to zero.
- Increments the counter when an **event occurs.**
- If the event is **sending of a message,the process includes the incremented value of the counter in the message**.
- If it is **receiving a message,a check to see if the incremented counter <= timestamp** in the received message. If so it is corrected to 1+ timestamp otherwise let it be as it is.

Event ordering contd..

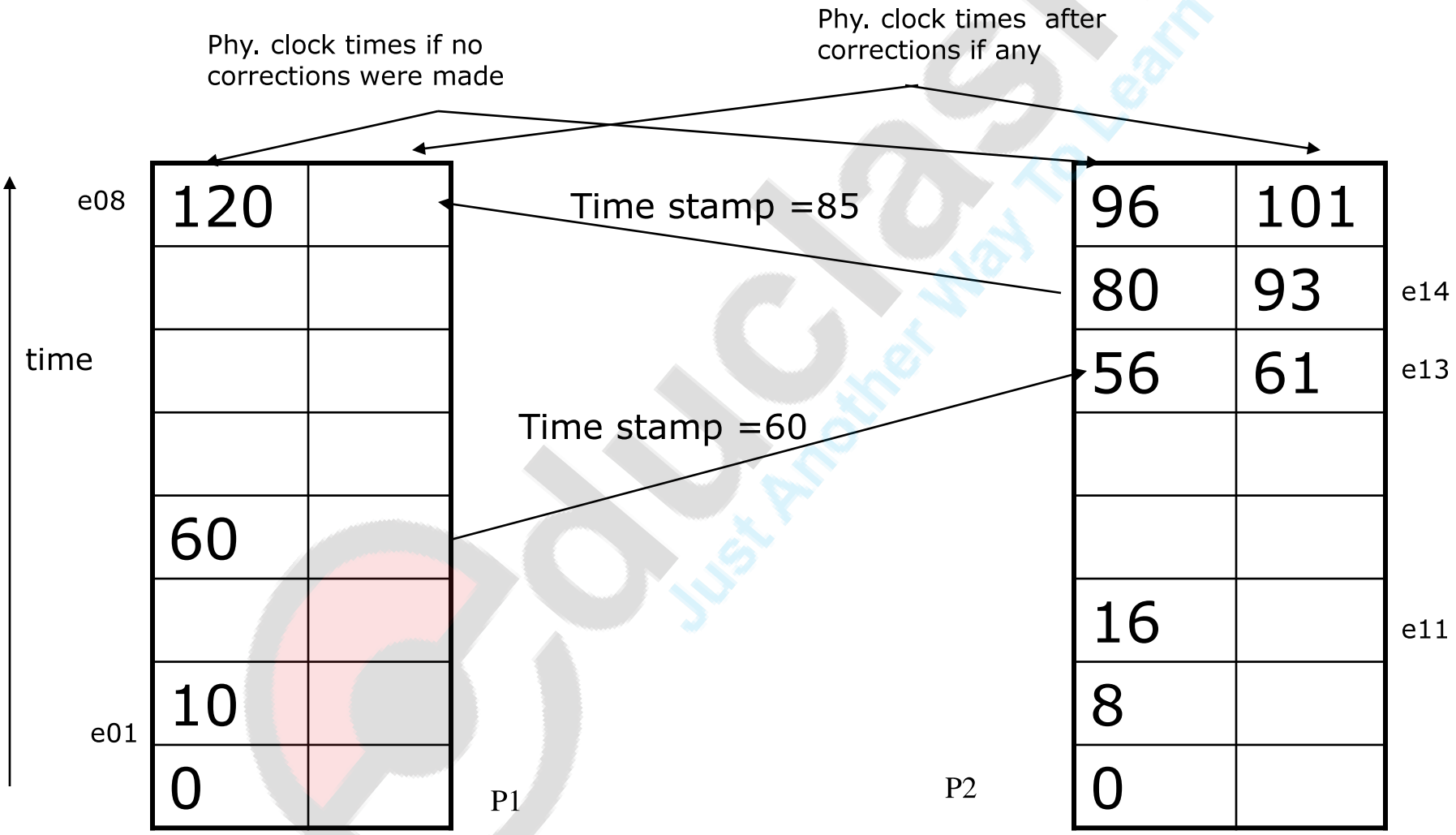


Implementation of logical clocks.. by using counters

Event ordering contd..

- **Implementation of logical clocks.. by using physical clocks**
- Each process has a physical clock.
- The ticks may be different for physical clocks.
- A message sending event send the message and its current physical time.
- A message receiving event check its current physical time.
- If it is less than time included in the message,it is corrected by receiving time +1.

Event ordering contd..



Implementation of logical clocks.. by using physical clocks

Total ordering of events

- If two events of two different processes, have the same timestamps, then Lamport proposed total ordering of processes.
- **Process identity numbers may be used to break ties.**

For ex. the time stamps of A & B are A=100.001, B=100.002 **if**

process id of A & B are 001 002 resp

Mutual exclusion

- Several resources in a system should not be used simultaneously by multiple processes.

  for ex. File, tape drive ,printer etc.
- Exclusive access to such a shared resource by a process must be ensured.
- This exclusiveness of access is called mutual exclusion between processes.
- Means to prevent processes from executing concurrently within their associated critical sections.
- **An algorithm must satisfy two requirements for M.E.-**

➢ **1. Mutual exclusion (acquire, use release)**

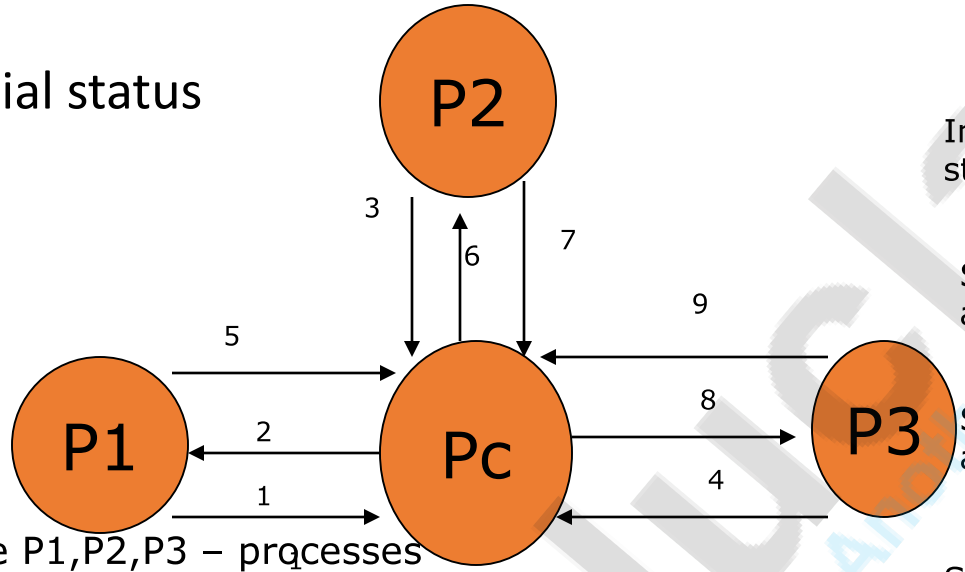
➢ **2. No starvation (req . eventually granted)**

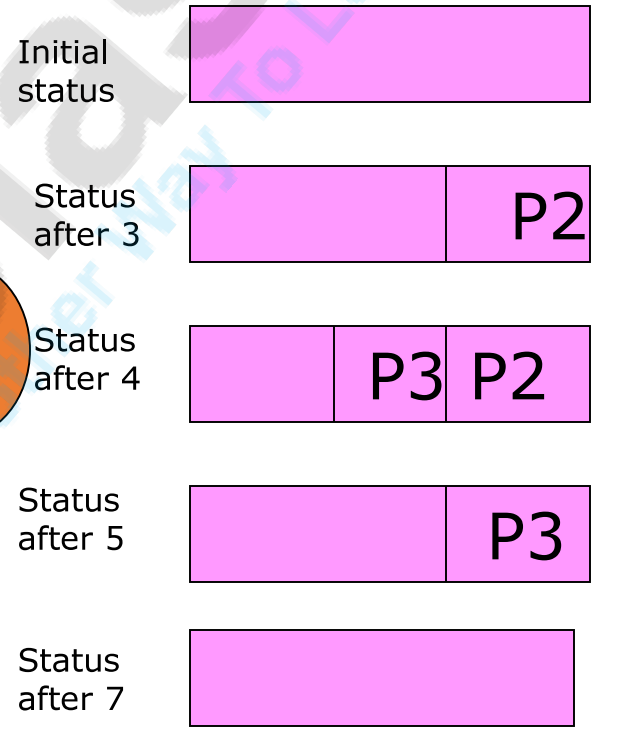
Mutual exclusion Contd…

- **Mutual Exclusion** : **exclusiveness of access e.g** updation of a file
  - Systems that involve multiple process often utilize critical regions (sections of a prog. that need exclusive access to shared recourses).
    - When a process has to read or update certain shared structures it
      - Enters a critical section
      - Performs its operation
      - Leaves the critical section
- – We use special constructs to serialize access to critical sections (semaphores, monitors, … **for single processor sys.)**
- Most of the techniques that we know do not support distributed systems because there is no single shared memory image.
- Need new techniques to achieve mutual exclusion
  - Centralized and distributed techniques.

Mutual exclusion Centralized approach (for entering the critical section)

Initial status





Status of request queue

Where P1,P2,P3 – processes

Pc- coordinator process

1-request,2-reply,3-request,4-request,5-release

6-reply,7-release,8-reply,9-release

- **Mutual exclusion**
- **No starvation**
- **But a single point of failure**

RRR

Mutual exclusion
Centralized approach contd..

Simulate how mutual exclusion is performed in a shared memory system.

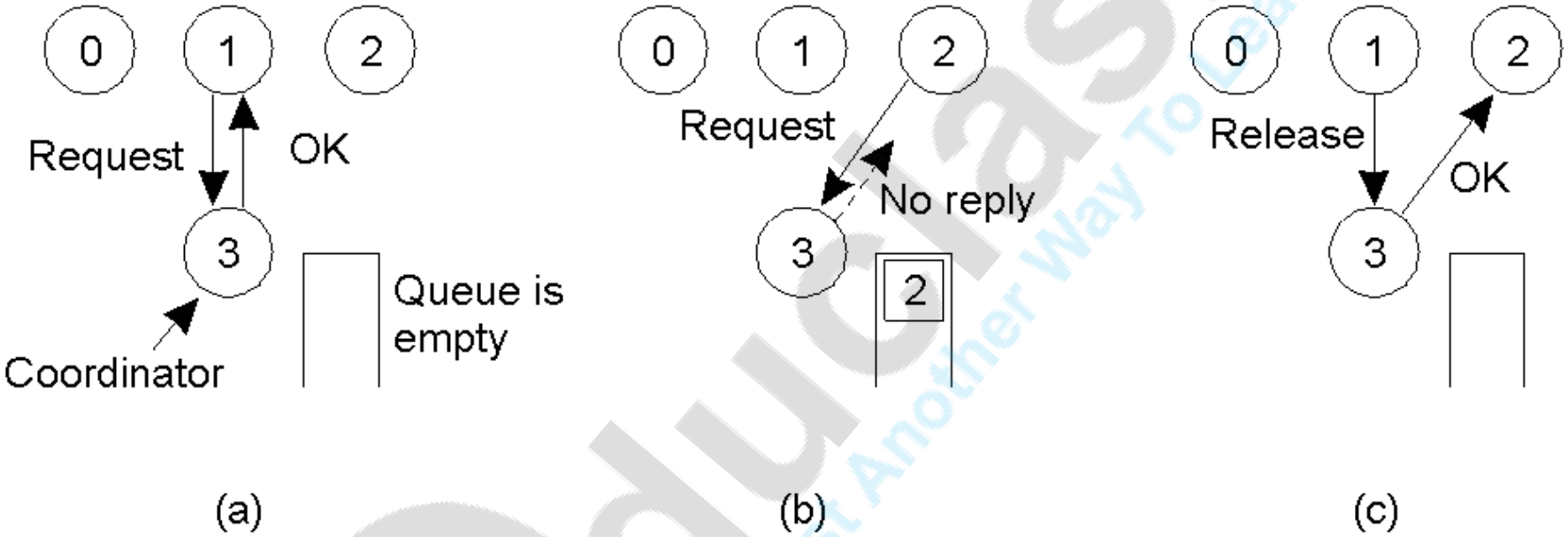
- One process acts as a coordinator.
- Coordinator maintains a queue of requests for access to a critical section
- All processes send a message to the coordinator specifying the critical region that it wants to enter.
- The coordinator either grants permission to the requesting process or **queues** the request if another process is using the critical section.
- When a process exits the critical section it sends a message to the coordinator.
- The coordinator now grants permission to the next process in the request queue, if any exists.

Mutual exclusion
Centralized approach contd..

Algorithm guarantees mutual exclusion.

- Algorithm is fair (FIFO ordering)
- Algorithm is easy to implement
  - **RPC's**
  - **Message Passing**
  - **Need 3 messages: Request, Grant, Release**
- The coordinator is a single point of failure.
- If the coordinator crashes the entire system will go down.
- **Process can not tell the difference between a request denied (blocking) and a dead coordinator. (elect a new coordinator and build up a new status req queue)**
- **A single coordinator may become a performance bottleneck** .

Mutual Exclusion: A Centralized Algorithm



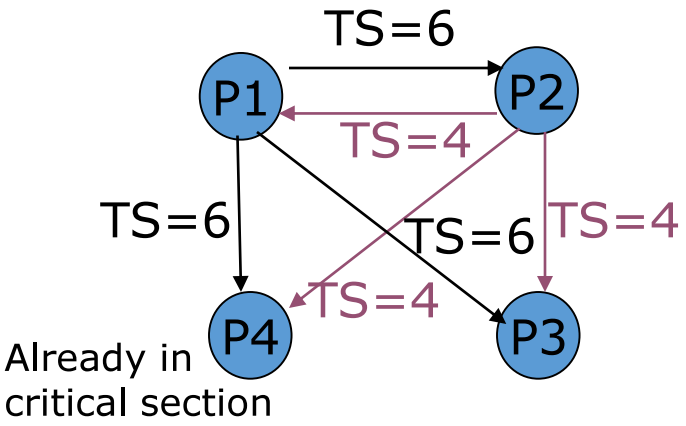
a) Process 1 asks the coordinator for permission to enter a critical region. Permission is granted

b) Process 2 then asks permission to enter the same critical region. The coordinator does not reply.

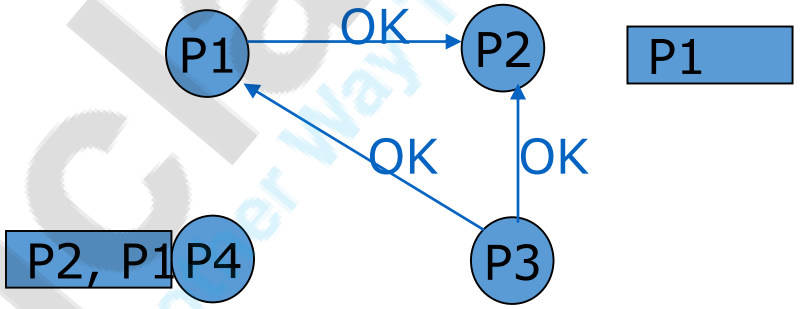
c) When process 1 exits the critical region, it tells the coordinator, which then replies to 2

Mutual Exclusion
Distributed Approach

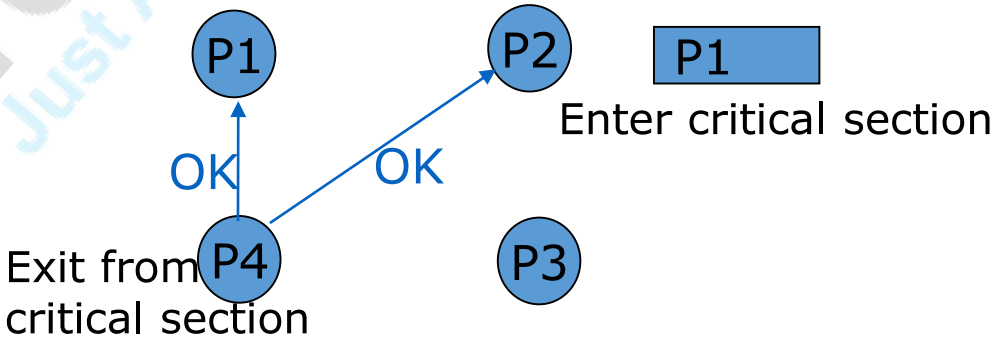
(A) P1 and P2 send request messages



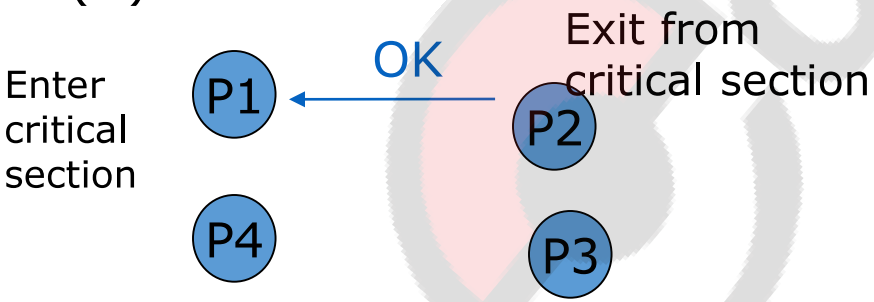
(B) P4 is in critical section.



(C) P4 exits from critical section.



(D) P2 exits from critical section.



Mutual Exclusion
Distributed Approach contd..

pid cs name time stamp

Info contained in a msg

- Mutual exclusion
  - No starvation
  - Deadlock free

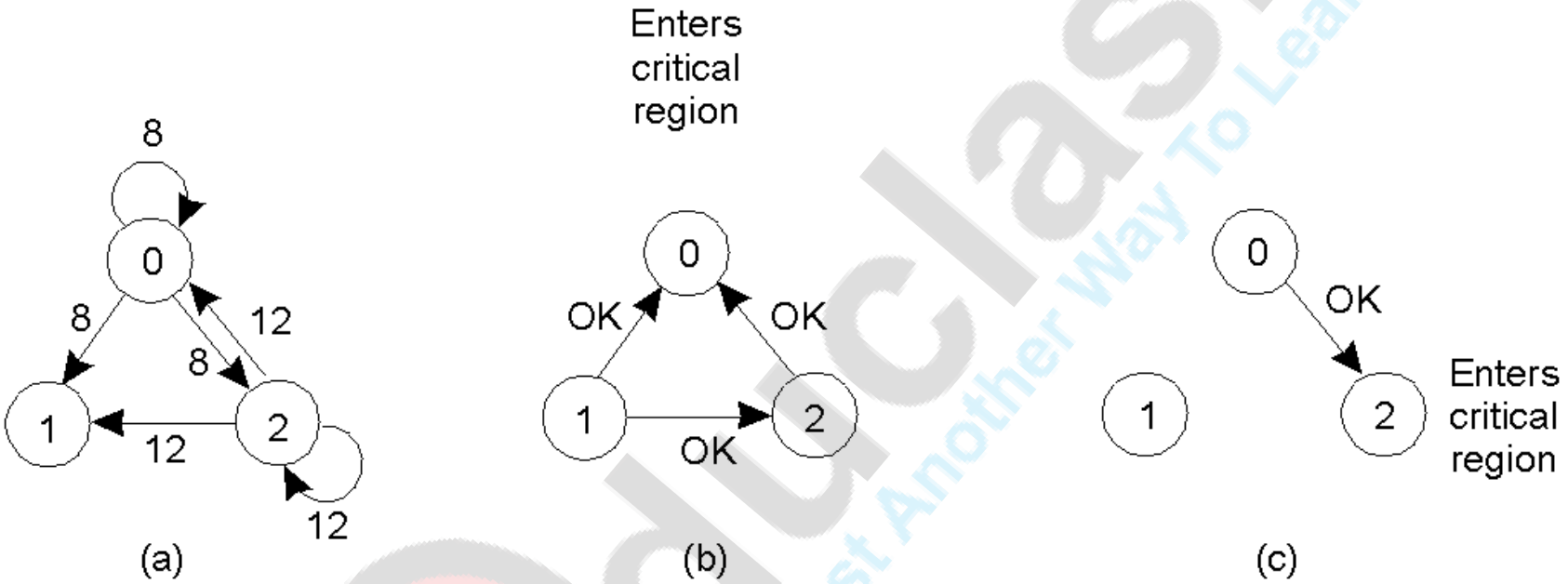
- Drawbacks
  - Fault intolerant messages (n pts of failure)

Sol:permission denied message

- Expensive cost: 2(n-1)
- Suitable for a small group of cooperating processes

Fixed member processes

A Distributed Algorithm

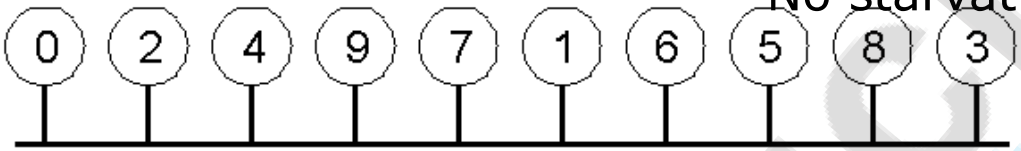


a) Two processes want to enter the same critical region at the same moment.
b) Process 0 has the lowest timestamp, so it wins.
c) When process 0 is done, it sends an OK also, so 2 can now enter the critical region.

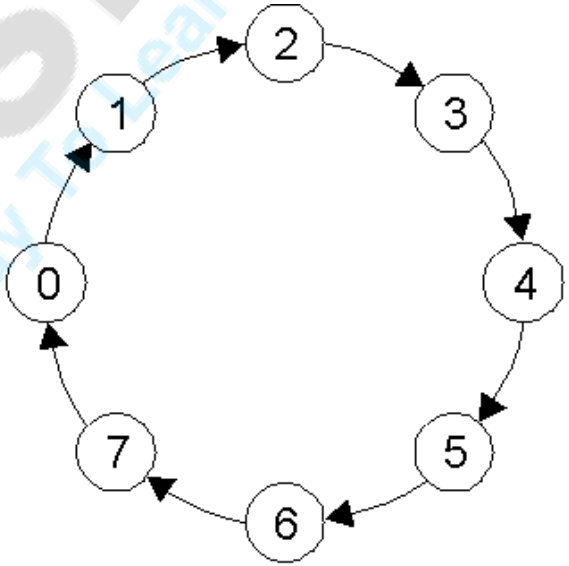
Mutual Exclusion

Token Ring Algorithm

- Mutual exclusion
- No starvation



(a)



(b)

- Process failure : ack.to the neighbor
- Lost token: monitor

(c)

a) An unordered group of processes on a network.
b) A logical ring constructed in software.(clockwise or anticlockwise)
c) Monitor is elected for "who has he token"

Comparison

Algorithm	Messages per entry/exit	Delay before entry (in message times)	Problems
Centralized	3	2	Coordinator crash
Distributed	2 (n – 1)	2 (n – 1)	Crash of any process
Token ring	1 to ∞	0 to n – 1	Lost token, process crash

A comparison of three mutual exclusion algorithms.

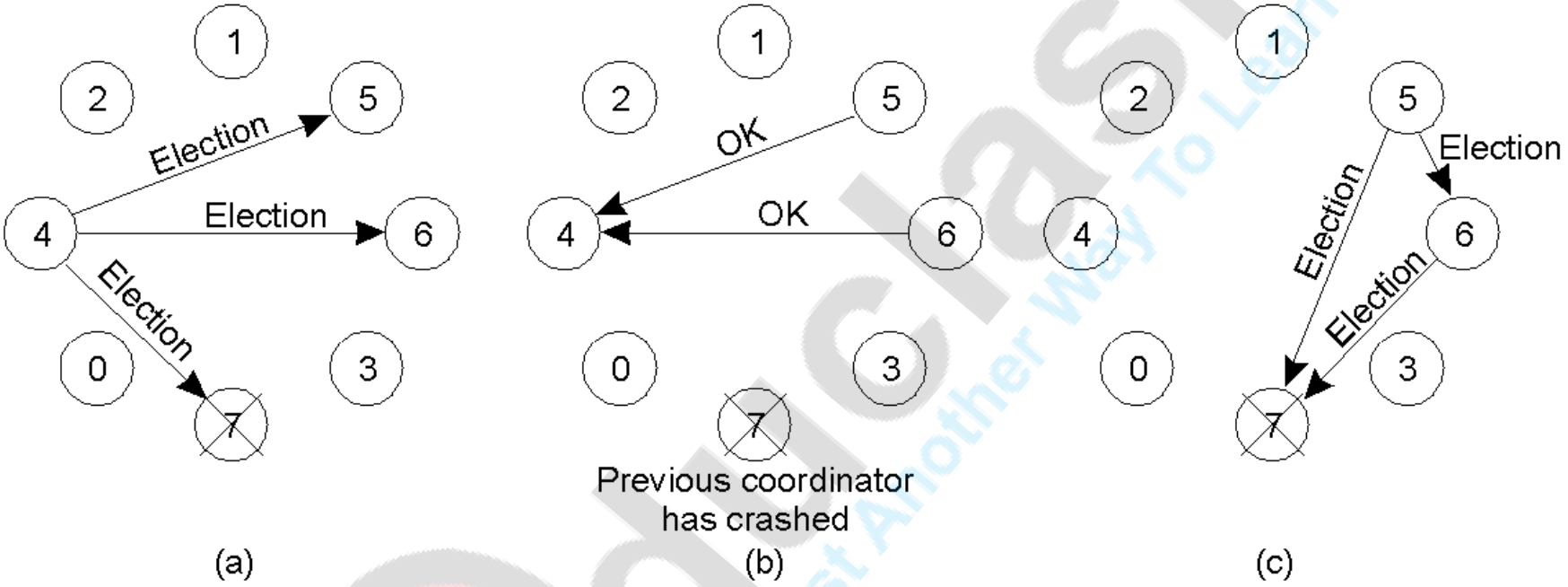
Election Algorithms

- Many distributed algorithms require one process to act as a coordinator.
- It generally does not matter which process takes on the special coordinator responsibility.
- The problem is how do we select a coordinator & hold an election.
- If a coordinator crashes how do we elect a new coordinator.
  – Process normally ends.
  – Process crashes and abnormally ends.
- Requirements (for an election)
  – Every process must have a unique number, for example its network address.
  – Every process knows the process number of every other process.
  – All process must agree on the new coordinator (i.e; with the highest priority) when the election ends.
  -On recovery the failed process can join the active processes

Election Algorithms The Bully Algorithm

- When a process notices that the coordinator is no longer responding to requests, it initiates an election
- Process ***P*** holds an election as follows:
- *P* sends and *ELECTION* message to all processes with higher numbers
- If nobody responds, *P* wins the election and becomes the new coordinator
- If one of the higher numbered processes answers, it takes over and *P*'s job is done
- Each higher numbered process now holds its own election based on the above process
- Eventually a process will hold an election and no other process will respond
- This process becomes the new coordinator and announces this fact to all of the other processes in the system (REC message)

The Bully Algorithm



The bully election algorithm

- Process 4 holds an election
- Process 5 and 6 respond, telling 4 to stop
- Now 5 and 6 each hold an election
- On recovery 7 either initiates an election or bullies other processes

Election Algorithms Ring Algorithm

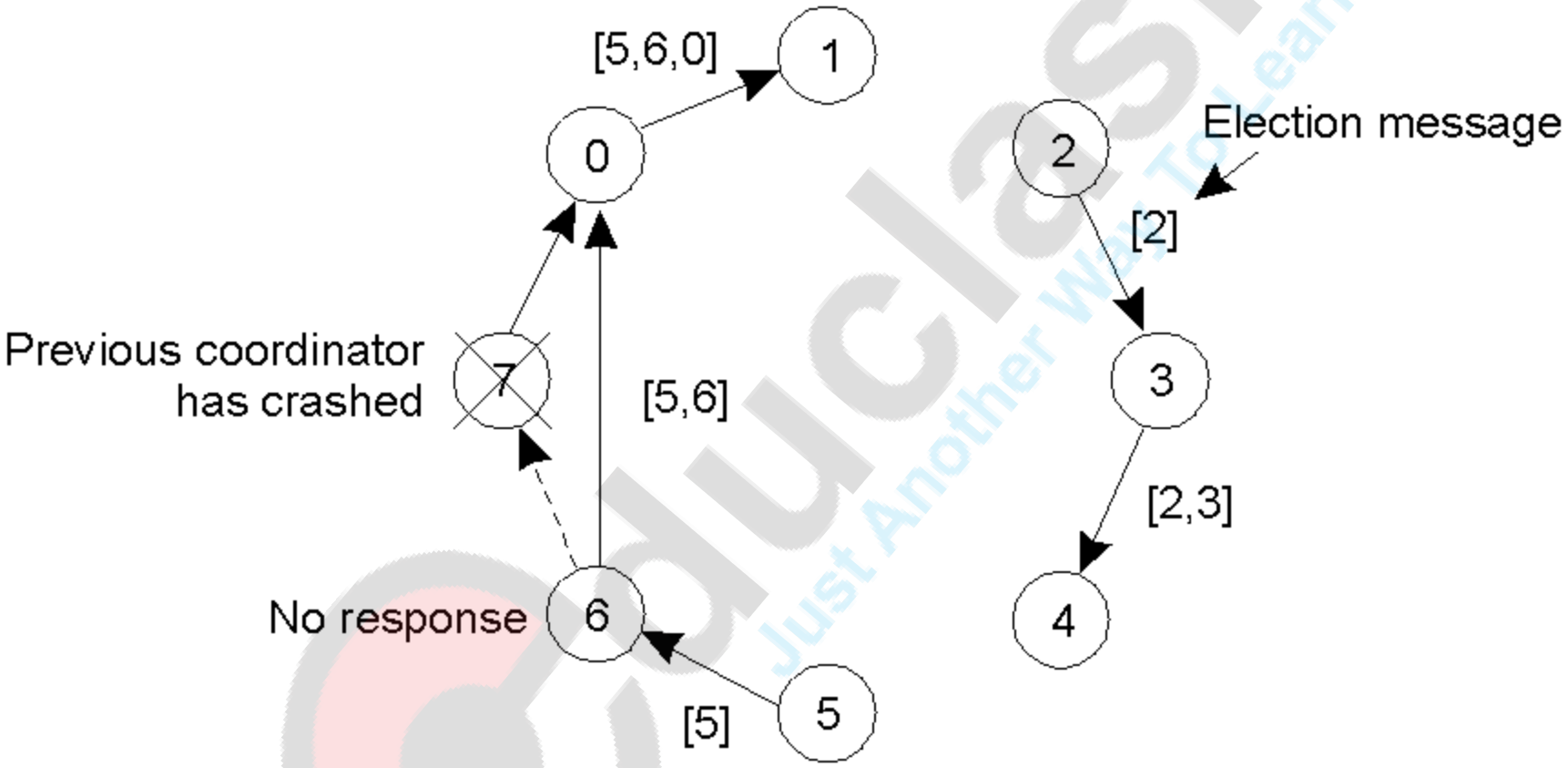
- The ring algorithm is based on arranging the distributed processes in a ring
- – No token is used in this algorithm
- Each process knows who its successor is
- When a process notices that the coordinator is not responding it builds an *ELECTION* message
- – This message includes the process number of the process that initiated the *ELECTION*
- The *ELECTION* message is then sent to the processes successor
- Once received, the process attaches its process number to the *ELECTION* message

Election Algorithms Ring Algorithm contd..

Eventually the message will go around the ring.

- When the original initiator of the *ELECTION* message receives the message again (after traversing the ring) the message will contain the process ID's of all of the operating processes in the distributed environment.
- The original initiator of the *ELECTION* message then extracts the ID of the largest process and generates a *COORDINATOR* message containing the largest processes ID.
- This message is then sent around the ring .
- Each process in the ring then records who is the new coordinator.

A Ring Algorithm



- Election algorithm using a ring.

Election Algorithms

Compare the two election algorithms

Applications of Distributed Synchronization

- Distributed synchronization techniques can be scaled up to enable distributed atomic transactions.
- Distributed Deadlock Management