Synchronization

Introduction

- Sharing system resources among multiple concurrent processes may be:
  - Cooperative or competitive in nature.
- Both require adherence to certain rules of behavior:
  - For enforcing correct interaction.

This requires synchronization mechanisms.

Synchronization-related issues

1. Clock synchronization
2. Event ordering
3. Mutual exclusion
4. Deadlock
5. Election algorithms

Clock synchronization

- Requirement of a timer mechanism for every computer:
  - To keep track of current time,
  - For various accounting purposes,
  - For correct results, and
  - helps in measuring the duration of distributed activities.

Applications

- For a distributed **on-line reservation system**, the only remaining seat booked almost simultaneously from two different nodes **should be offered to the client who booked first**, even if the time difference between two bookings is very small.
- may not be possible to guarantee this if the clocks of the nodes are not synchronized.
- to measure duration of distributed activities that start on one node and terminate on another node, E.g., calculating the time taken to transmit a message from one node to another.

Implementation of computer clocks

- Components of a computer clock:

1. **A quartz crystal**
   - Oscillates at a well-defined frequency.

2. **A counter register**
   - Keeps track of the oscillations of the quartz crystal.
   - If value is zero, an interrupt is generated and reinitialized to the value of constant register.

3. **A constant register**
   - Stores a constant value, based on the frequency of oscillation of the quartz crystal.

Cont…

- Each interrupt is known as a ***clock tick.***
- The value in the constant register is chosen so that 60 clock ticks occur in a second.
- The computer clock is synchronized with real time. For this:
- Two more values are stored: (i) **fixed starting date and time** and (ii) **number of ticks**
- E.g., in UNIX, time begins at 0000 on January 1, 1970.
- System converts entered **current date and time** to the number of ticks after the fixed starting date and time.

Drifting of clocks

- A clock always runs at a constant rate.
  - but may be difference in the crystals.
  - hence, two clocks may run at different rate
- For clocks based on a quartz crystals, the **drift rate** is approximately **10^{-6}**.
  - It must be periodically resynchronized with the real-time clock to keep it non-faulty.
- A clock is **non-faulty** if:
  - There is a bound on the amount of drift from real time for any given finite time interval.

Cont…

when

- real time = *t*, time value of a clock *p* is **$C_p(t)$.**

- If all clocks in the world are perfectly synchronized, we would have **$C_p(t) = t$** for all *p* and for all *t*.

- **Ideal case:**

  $dC/dt = 1$

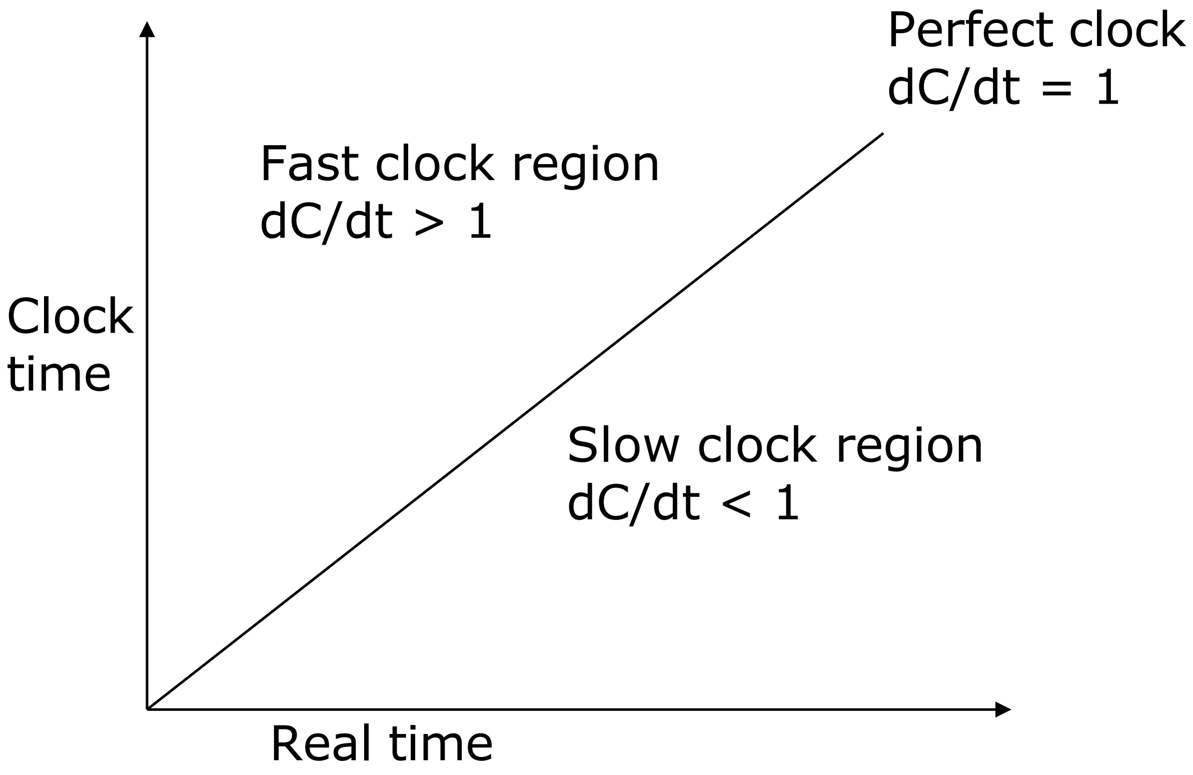
Cont…

- A clock is non-faulty if:

  $1 - \rho <= dC/dt <= 1 + \rho$

  where ρ is maximum allowable drift rate.

slow, perfect and fast clocks



Cont…

- $slow$ and fast clocks drift in opposite directions from the perfect clock.
- two clocks, if one is slow and one is fast, at a time Δt after they were synchronized, the maximum deviation between the time value of the two clocks will be $\boldsymbol{2\rho\Delta t}$.

- to guarantee that no two clocks in a set of clocks ever differ by more than δ, the clocks in the set must be resynchronized periodically, with the time interval between two synchronizations being $\leq \frac{\boldsymbol{\delta}}{\boldsymbol{2\rho}}$.

Cont…

- Types of clock synchronization in DS:
  1. Synchronization of the computer clocks with real-time (or external) clocks.
     - Required for real-time applications.
     - Coordinated Universal Time (UTC), an external time source is used for synchronization.
     - Also synchronized internally.

Cont…

2. Mutual (or internal) synchronization of the clocks of different nodes of the system.

- Used when a consistent view of time across all nodes of a DS is required &
- A measurement of the duration of distributed activities are required.

Clock Synchronization Issues

- The difference between the two clock values is called *clock skew*.
- Required to be synchronized if the clock skew is more than some specified constant δ.

Cont…

- Issues:
  1. Calculating the value of the unpredictable communication delays between two nodes to deliver a clock signal is practically impossible.
     - Depends on the amount of communication and computations,
  2. Time must never run backward.
     - It may cause problems like repetition of certain operations.

Clock synchronization algorithms

1. Centralized algorithms
2. Distributed algorithms

Centralized algorithms

- Use of time server node for referencing the correct time.
- These algorithms keep the clocks of all other nodes synchronized with the clock time of the time server node.

Cont…

- Algorithms depending on the role of time server node:

1. Passive time server Centralized Algorithm.
2. Active time server Centralized Algorithm.

Passive time server Centralized Algorithm

- Each node periodically sends a message for the current time to the time server at any time t0.
- Time server responds with the current time T at the time t1.
- The propagation time of the message :

  (t1 – t0) /2
- When the reply is received at the client’s node, its clock is readjusted to

  T + (t1 –t0) /2

Proposals to improve Centralized Algorithm

1. Assuming availability of some additional Info.

   - Assumes that the approximate time taken by the time server to handle the interrupt and process the request is known.

Clock Synchronization Algorithms

- → Centralized →
  - → passive time-server centralized algo.
  - → Active time-server centralized algo.
- → Distributed

Centralized Algo.s

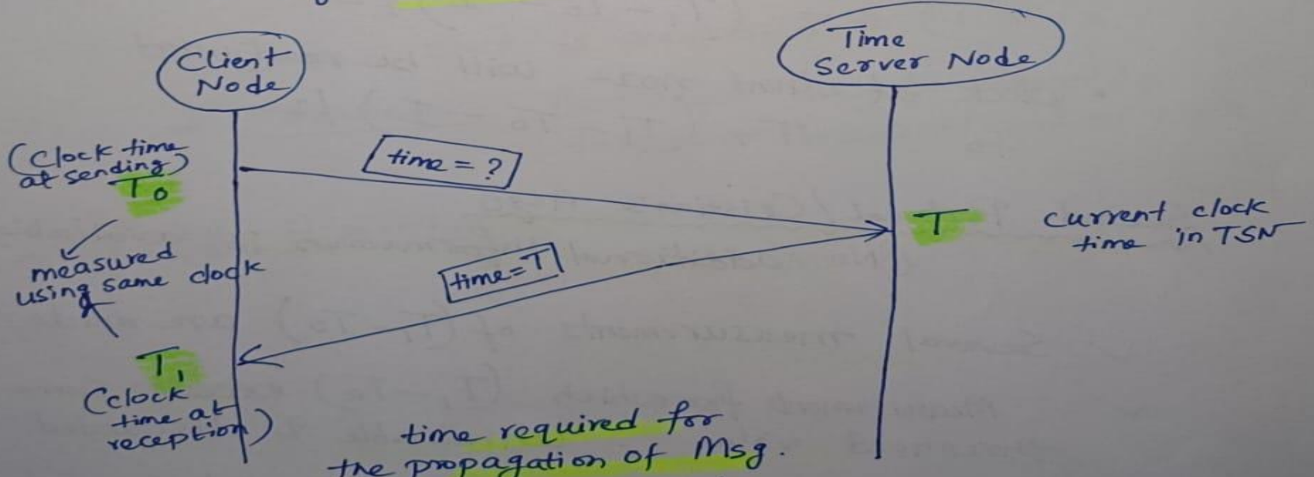
✓ one node has a real-time receiver (called time server node) (TSN)
↳ [clock time is regarded as correct and used as the reference time]

✓ Goal : Keep the clocks of all other nodes synchronized with the clock time of time server node.

Passive Time Server Centralized Algorithm

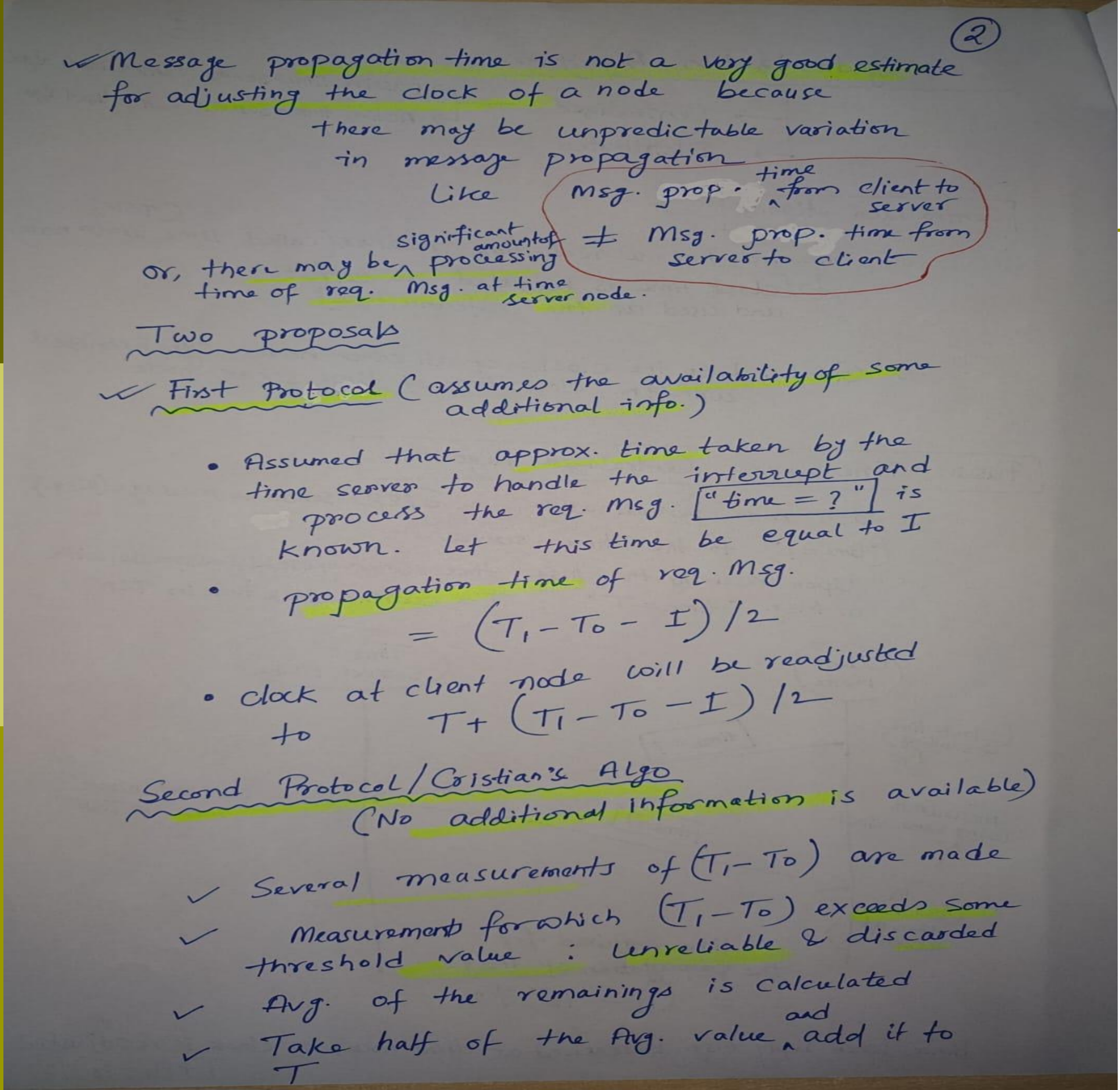
✓ each node periodically ($\leq \delta/2\rho$) sends a message (Msg.) ("time=?") to the time server.

✓ Upon receiving the Msg., time server quickly responds with a Msg. "time = T" ; T = current clock time in TSN


Client Node
Time Server Node
time = ?
time=T
(Clock time at sending) T0
measured using same clock
T1
(clock time at reception)
T
current clock time in TSN

time required for the propagation of Msg.
$= (T_1 - T_0)/2$

Hence, when reply Msg. is received at client node, its clock is readjusted to $T + (T_1 - T_0)/2$



✓ Message propagation time is not a very good estimate for adjusting the clock of a node because
there may be unpredictable variation in message propagation
Like Msg. prop. time from client to server $\neq$ Msg. prop. time from server to client

or, there may be significant amount of processing time of req. Msg. at time server node.

Two proposals

✓ First Protocol (assumes the availability of some additional info.)

- Assumed that approx. time taken by the time server to handle the interrupt and process the req. msg. $\boxed{\text{"time = ?"}}$ is known. Let this time be equal to I
- propagation time of req. Msg.
$$= (T_1 - T_0 - I)/2$$
- clock at client node will be readjusted to $T + (T_1 - T_0 - I)/2$

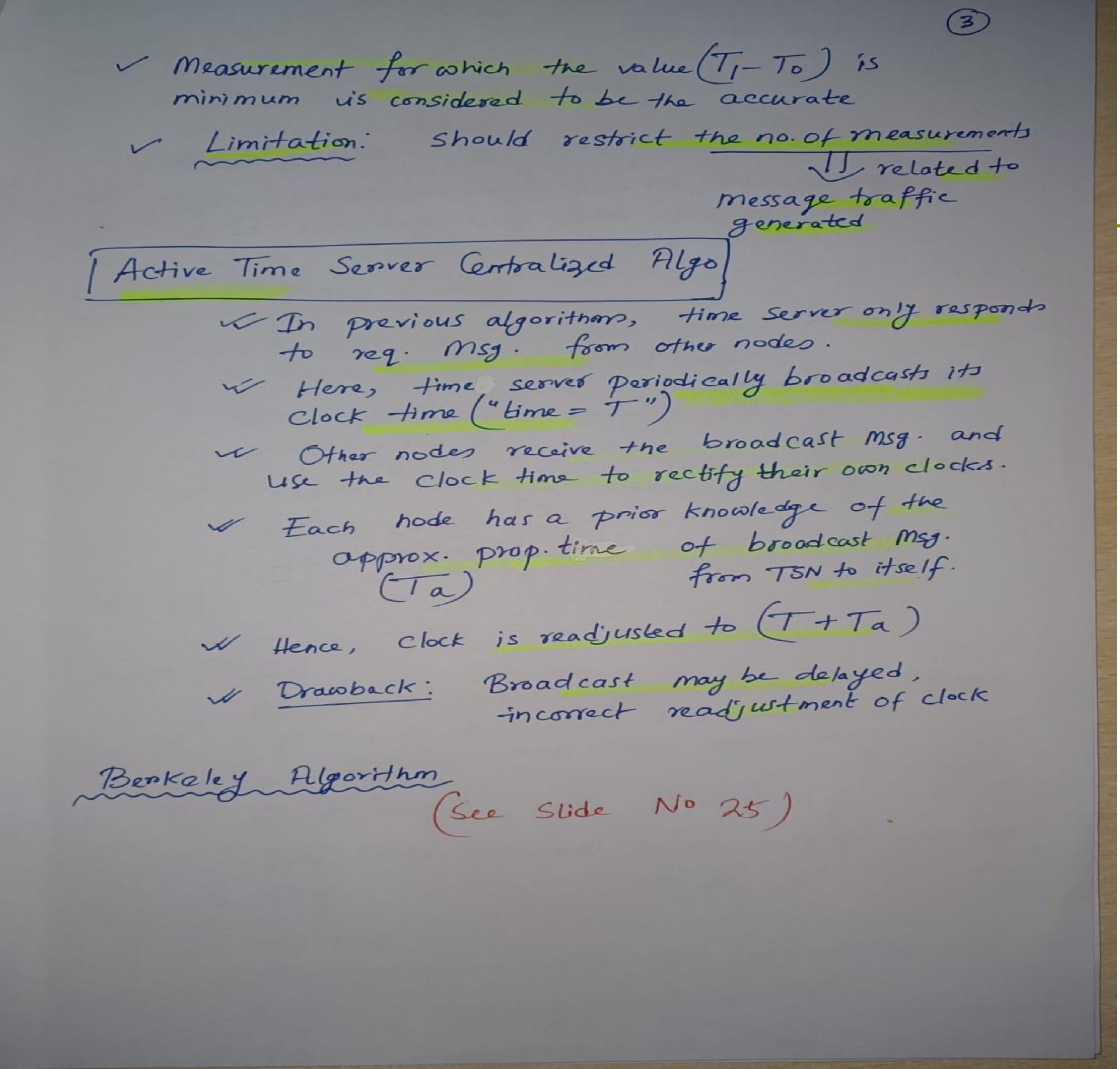
Second Protocol / Cristian's Algo
(No additional information is available)

✓ Several measurements of $(T_1 - T_0)$ are made

✓ Measurements for which $(T_1 - T_0)$ exceeds some threshold value : unreliable & discarded

✓ Avg. of the remainings is calculated

✓ Take half of the Avg. value and add it to T



- ✓ Measurement for which the value $(T_1 - T_0)$ is minimum is considered to be the accurate
- ✓ Limitation: should restrict the no. of measurements
  $\Downarrow$ related to
  message traffic generated

Active Time Server Centralized Algo

- ✓ In previous algorithms, time server only responds to req. msg. from other nodes.
- ✓ Here, time server periodically broadcasts its clock time ("time = T")
- ✓ Other nodes receive the broadcast msg. and use the clock time to rectify their own clocks.
- ✓ Each node has a prior knowledge of the approx. prop. time of broadcast msg. from TSN to itself. (T_a)
- ✓ Hence, clock is readjusted to $(T + T_a)$
- ✓ Drawback: Broadcast may be delayed, incorrect readjustment of clock

Berkeley Algorithm

(See Slide No 25)

Berkeley Algorithm

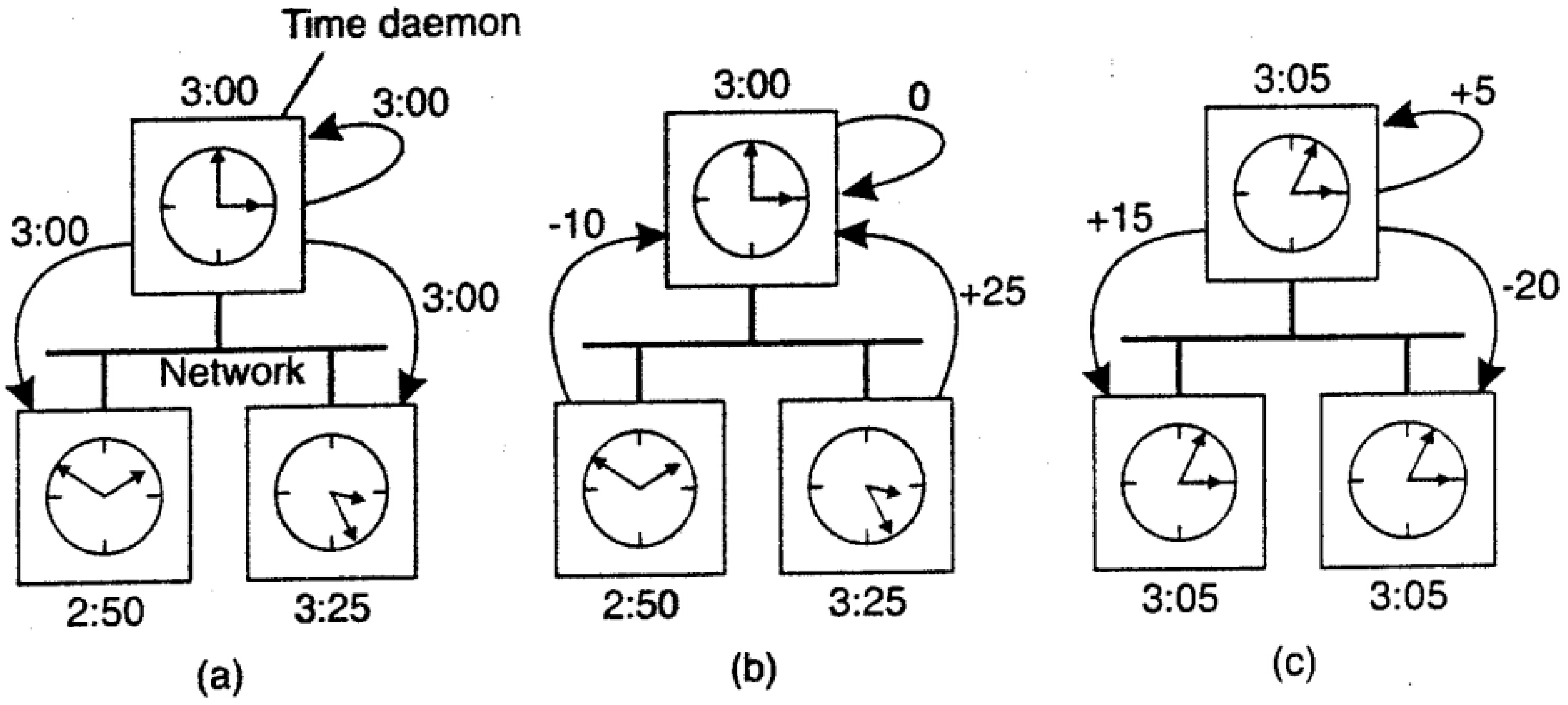


Figure 6-7. (a) The time daemon asks all the other machines for their clock values. (b) The machines answer. (c) The time daemon tells everyone how to adjust their clock.

Cont…

- time server (a time daemon) is active, polling every machine from time to time to ask what time it is there.
- Based on the answers, it computes an average time and tells all the other machines to advance their clocks to the new time or slow their clocks down.
- The time server sends the amount of time by which each node should readjust its clock.

Cont…

Berkeley Algorithm (PK Sinha)

- The time server periodically sends a message to the group.
- Receivers send back their clock value to the time server.

Cont…

- Based on the priori knowledge of propagation time, time server readjusts the clock value of the reply messages.
- It uses a fault-tolerant average.
- The time server sends the amount of time by which each node should readjust its clock.

Cont…

- Drawbacks of centralized clock synchronized algorithms:

1. Subject to a single-point failure (if time server node fails).
2. Not acceptable from the scalability point of view (in a large system, it puts a heavy burden on time server process)

Distributed algorithms

- Externally synchronized clocks are internally synchronized for better accuracy.

- Algorithms for internal synchronization:
  - Global averaging distributed algorithms.
  - Localized averaging distributed algorithms.

Global averaging distributed algorithms

- Clock process of each node broadcast its local time at the beginning of every fixed-length resynchronization interval.
  - Can’t happen simultaneously from all nodes.
  - Hence each node waits for some time T.
- After this, the clock process collects resync messages broadcast by other nodes.
  - Records their time,
  - Estimates the skew of its clock, &
  - Computes the fault-tolerant average.

Cont…

- Algorithms for computing fault-tolerant average of the estimated skews:
  - Takes the average of the estimated skews and use it as the correction for the local clock.
    - Use threshold for comparison.
  - Each node limits the impact of faulty clocks by first discarding the m highest and m lowest estimated skews.
    - Calculate the average of the remaining skews for correcting the local clock.

Cont…

- Drawback
  - Not scalable: due to broadcast facility and a large amount of network traffic.

Localized averaging distributed algorithms

- The nodes are logically arranged in some kind of pattern (such as ring or a grid).
- Periodically, nodes exchange their clock time with their neighbours.
  - Sets their clock time to the average.

Event Ordering

- Lamport observed that for most applications, clock synchronization is not required.
- Total order of all events is sufficient that is consistent with observed behavior.
- For partial ordering of events, following can be used:
  - Relation of Happened-before, &
  - Logical clocks.

Happened-Before relation

- A relation (→) on a set of events satisfies the following conditions:
  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 same message by another process, then a→b.
  3. If a→b and b→c, then a→c [a transitive relation].

Cont…

- Happened-before is an irreflexible partial ordering on the set of all events in the system.

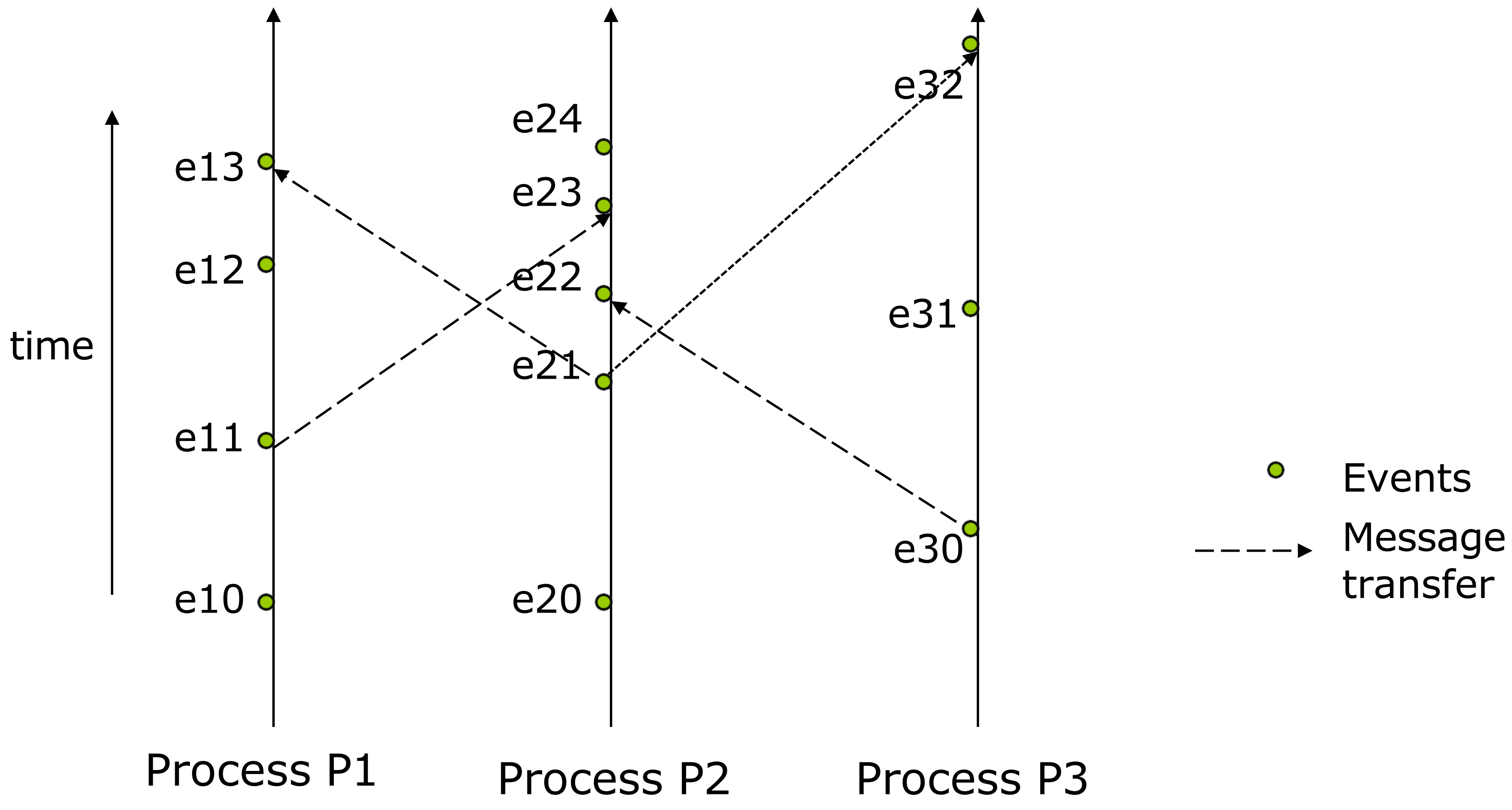
  a→a is not true since an event can’t happen before itself.

- Concurrent events are not related by this relation.

  a→b and b→a

  - That why, happened before is sometimes also known as the relation of causal ordering.

Space-time diagram for three processes



Cont…

- Some of the events that are related by the happened-before relation:

  e10→e11

  e20→e24

  e11→e23

  e21→e13

  e30→e24 (since e30→e22 and e22→e24)

  e11→e32 (since e11→e23, e23→e24 and)

Cont…

- Concurrent events:

e12 and e20,	e21 and e30,
e10 and e30,	e11 and e31,
e12 and e32,	e13 and e22

This is because no path exists between these.

Logical clocks concept

- It is a way to associate a timestamp with each system event for correct working of happened-before relation.

- Each process P_i has a clock C_i associated with it that assigns a number $C_i(a)$ to any event ‘a’ in that process.
  - The clock of each process is known as logical clock.

Cont…

- The timestamps assigned to the events by the system of logical clocks must satisfy the clock condition:

  For any two events a and b, if $a \rightarrow b$ then $C(a) < C(b)$.

  C is the clock function assigned to events.

Implementation of logical clocks

Clock conditions

- C1: If ‘a’ and ‘b’ are two events within the same process Pi and a→b, then Ci(a)<Ci(b)
- C2: if ‘a’ is the sending of a message ‘m’ by process P_i, and ‘b’ is the receipt of that message by process P_j, then $C_i(a)<C_j(b)$.
- C3: a clock C_i associated with a process P_i must always go forward, never backward.

Cont…

Implementation Rules:

- IR1: Each process P_i increments C_i between any two successive events.
    - Ensures condition C1.

Cont…

- IR2: If event ‘a’ is the sending of a message ‘m’ by process P_i , the message ‘m’ contains a timestamp $Tm=C_i(a)$, and upon receiving the message ‘m’, a process P_j sets C_j greater than or equal to its present value but greater than T_m.

  - Ensures the condition C2.
  - Both ensures the condition C3.

Logical Clock Implementation

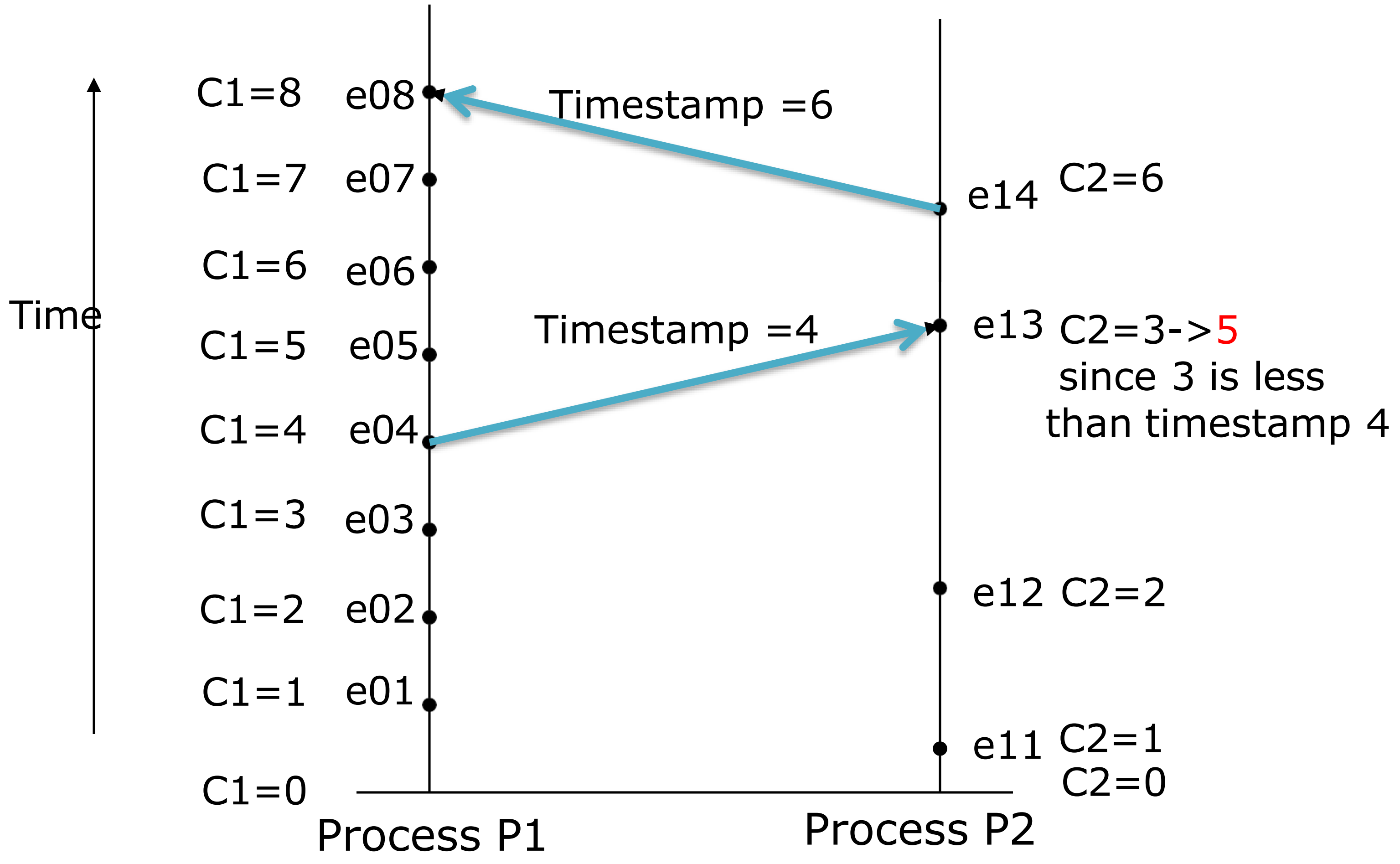
- Implementation of logical clocks by using
  1. Counters
  2. Physical clocks

Using counters

- Each process has a counter like C1 and C2 for process P1 and P2, respectively.

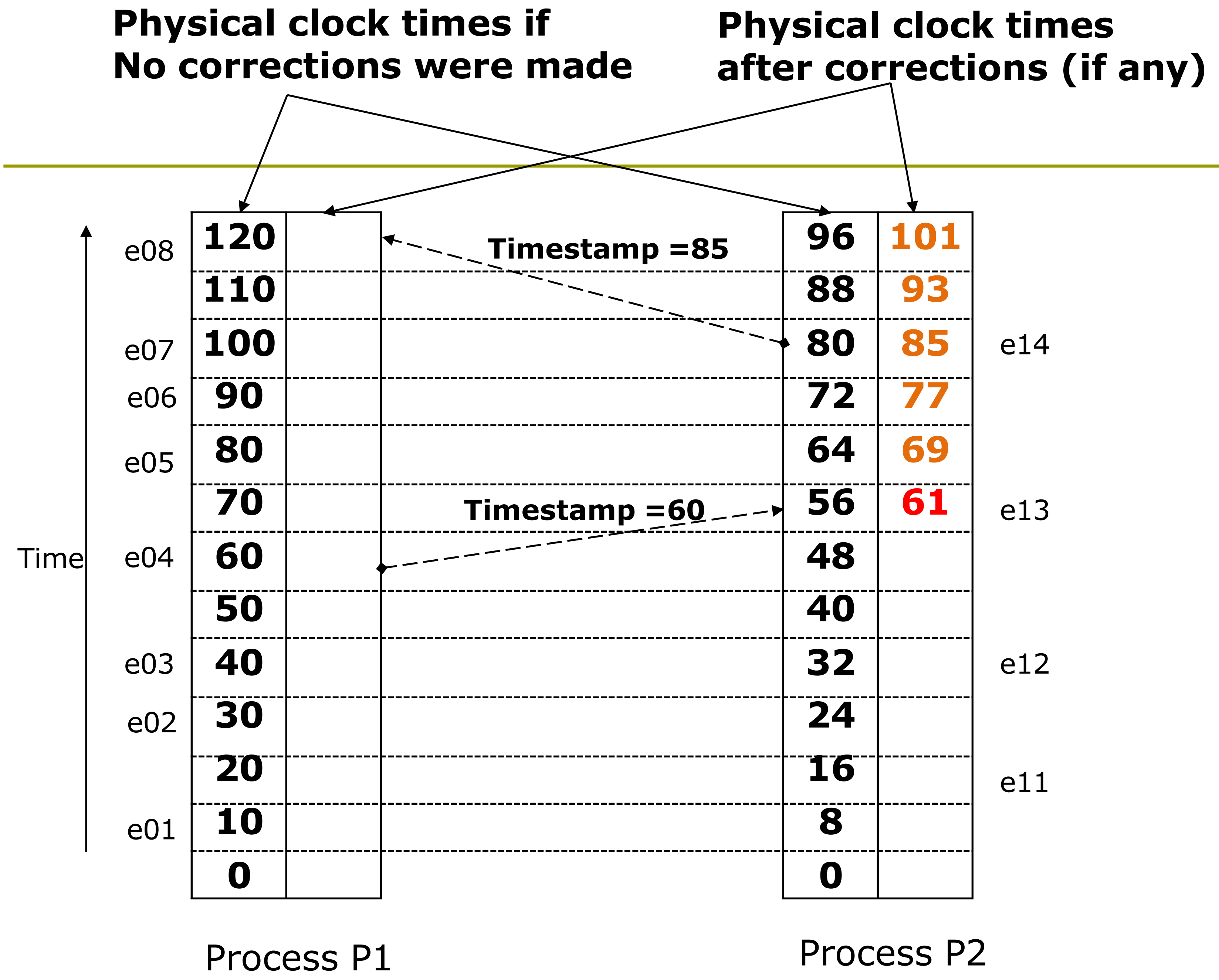
- Counters
  - Act as logical clocks.
  - Initialized to zero.
  - Increments by 1 on events of the process.

Cont…


Time
C1=8
e08
C1=7
e07
C1=6
e06
C1=5
e05
C1=4
e04
C1=3
e03
C1=2
e02
C1=1
e01
C1=0
Timestamp =6
Timestamp =4
e14
C2=6
e13
C2=3->5
since 3 is less
than timestamp 4
e12
C2=2
e11
C2=1
C2=0
Process P1
Process P2

Using physical clocks

- Each process has a physical clock associated with it.
- Each clock runs at a constant rate (rates at which different clocks run are different).
- Example:
  - When Process p1 has ticked 10 times, process p2 has ticked only 8 times.


Physical clock times if No corrections were made
Physical clock times after corrections (if any)
Time
e08
e07
e06
e05
e04
e03
e02
e01
120
110
100
90
80
70
60
50
40
30
20
10
0
Timestamp =85
Timestamp =60
96 101
88 93
80 85
72 77
64 69
56 61
48
40
32
24
16
8
0
e14
e13
e12
e11
Process P1
Process P2

Total ordering of events

- two events a and b that are not related by happened-before relation may have the same timestamps
- E.g., if a and b happen respectively in processes P1 and P2 when the clocks of both processes show same time (say 100), both events will have a timestamp of 100.
- nothing can be said about their order.
- **No events can occur at exactly the same time**.

Total ordering of events

- **Lamport** proposed the use of any arbitrary total ordering of the processes.
- E.g., **process id may be used to break ties and to create a total ordering** of events.
- For instance, the timestamps associated with events a and b will be 100.001 and 100.002, respectively, where the process identity numbers of processes PI and. P2 are 001 and 002, respectively.
- We now have a way to assign a unique timestamp to each event.

Concurrent Access of Resources

- Requirements of an algorithm for implementing mutual exclusion:

1. Mutual exclusion
2. No starvation

Cont…

- Mutual exclusion:
  - Given a shared resource accessed by multiple concurrent processes, at any time only one process should access the resource.
  - Can be implemented in single-processor systems, using semaphores, monitors and similar constructs.
- No starvation:
  - If every process that is granted the resource eventually releases it, every request must be eventually granted.

Cont…

- Approaches:

1. **Centralized approach**
2. **Distributed approach**
3. **Token passing approach**

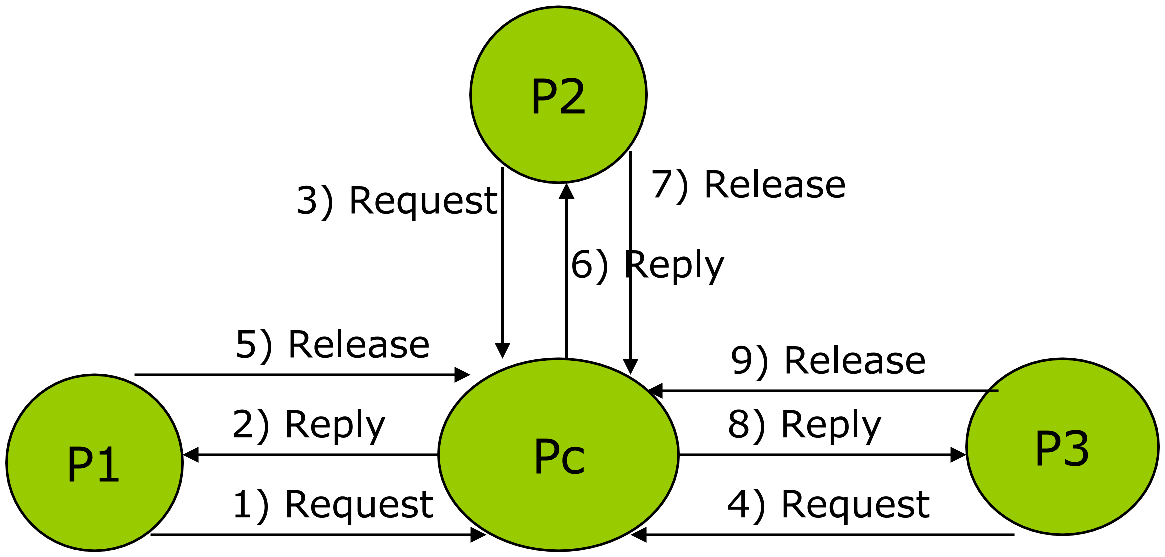
Centralized approach

- One of the processes in the system is elected as the coordinator.
- The coordinator coordinates the entry to the critical sections (CS).
- Every process takes permission from the coordinator before entering CS.
- If no process is currently in CS, coordinator can immediately grant permission to requesting process.

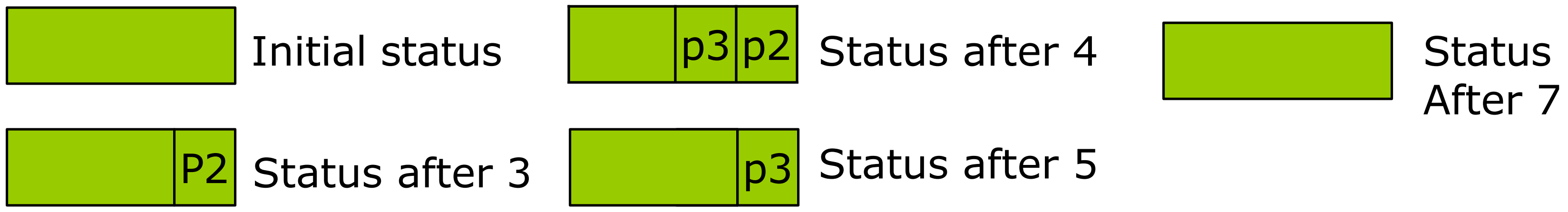
Centralized approach

- If two or more processes concurrently ask for permission to enter the CS, coordinator grants permission to only one process at a time w.r.t. some scheduling algorithm.
- When a process exits CS, it must notify the coordinator so that the coordinator can grant permission to another process (if any).
- Request queue is used to grant requests.
  - First-come, first-serve

Example illustrating the centralized approach for mutual exclusion



Status of request queue:



Cont…

- Mutual exclusion is ensured:
  - Only one process is allowed to enter a critical section.

- No starvation will occur:
  - Due to use of first-come, first-served scheduling policy.

Cont…

- **Advantages:**
  - Simple to implement
  - Requires only three messages per CS entry:a request msg., a reply msg. and a release msg.
- **Drawbacks:**
  - a single coordinator is subject to a single point of failure.
  - a performance bottleneck in a large system.

Distributed approach

- The decision making for mutual exclusion is distributed across the entire system.
  - All processes that want to enter the same CS cooperate with each other before reaching a decision on which process will enter the CS next.

Ricart and Agrawala Algorithm

- Lamport's event-ordering scheme is used to generate a unique timestamp for each event.
- When a process wants to enter a CS, it sends a request message to all other processes.

Cont…

- **Message contains:**
  - Process id
  - Name of the critical section
  - A unique timestamp generated by sender process for the request msg.

- On receiving a req. msg.

✓ A process either immediately sends back a reply msg. to sender or defers sending a reply based on following rules:

1. If receiver process is itself currently executing in CS, it simply queues the req. msg. and defers sending a reply.

Cont…

2. If receiver process is waiting for its turn to enter CS, it compares the timestamp in the received req. msg. with the timestamp in its own req. msg. that it has sent to other processes.

- ✓ If timestamp of the received req. msg. is lower, it means that the sender process made a request before the receiver process. Therefore, the receiver process immediately sends back a reply msg. to the sender.

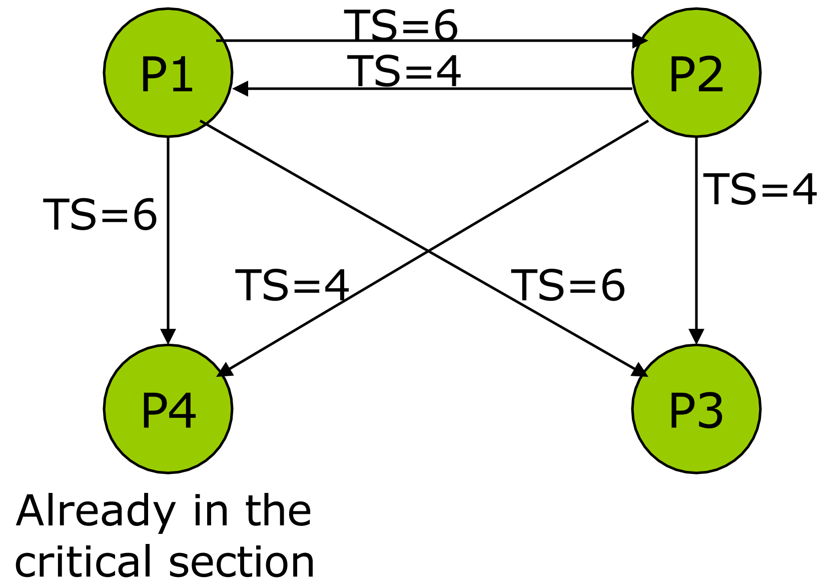
- ✓ If receiver process's own req. msg. has a lower timestamp, the receiver queues the received req. msg. and defers sending a reply message.

Cont…

3. If receiver process neither is in CS nor is waiting for its turn to enter the CS, it immediately sends back a reply message.

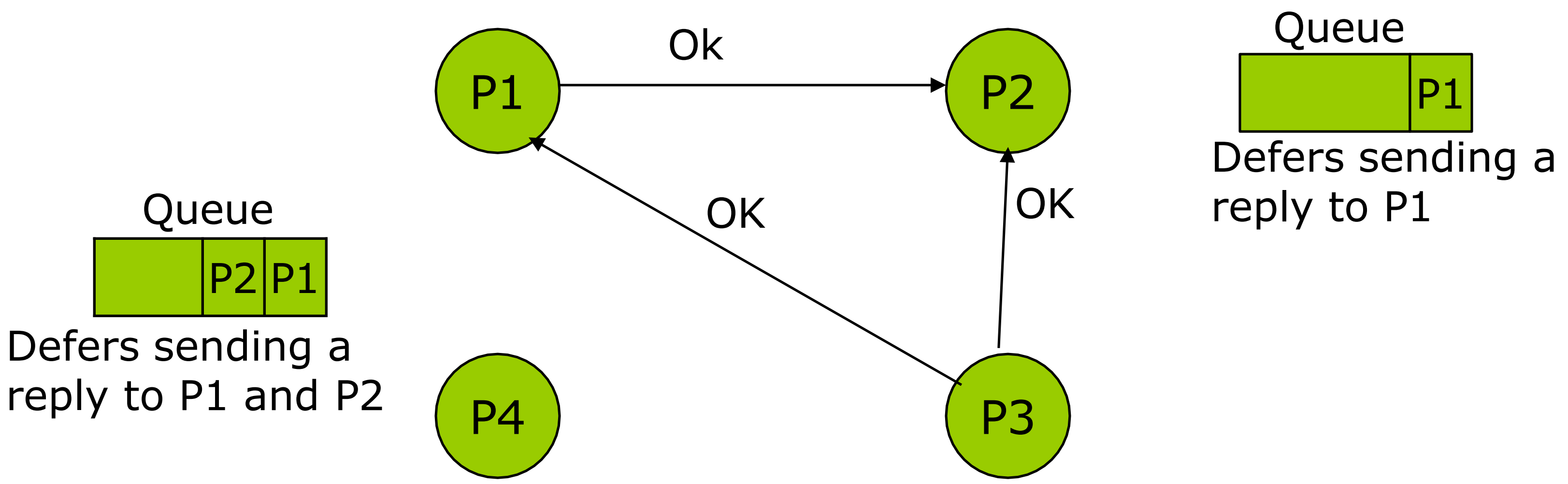
- A process **enters CS** as soon as it has received reply messages from all processes.
- After it exits CS, it sends reply message to all processes in its queue and deletes them from its queue.

Example



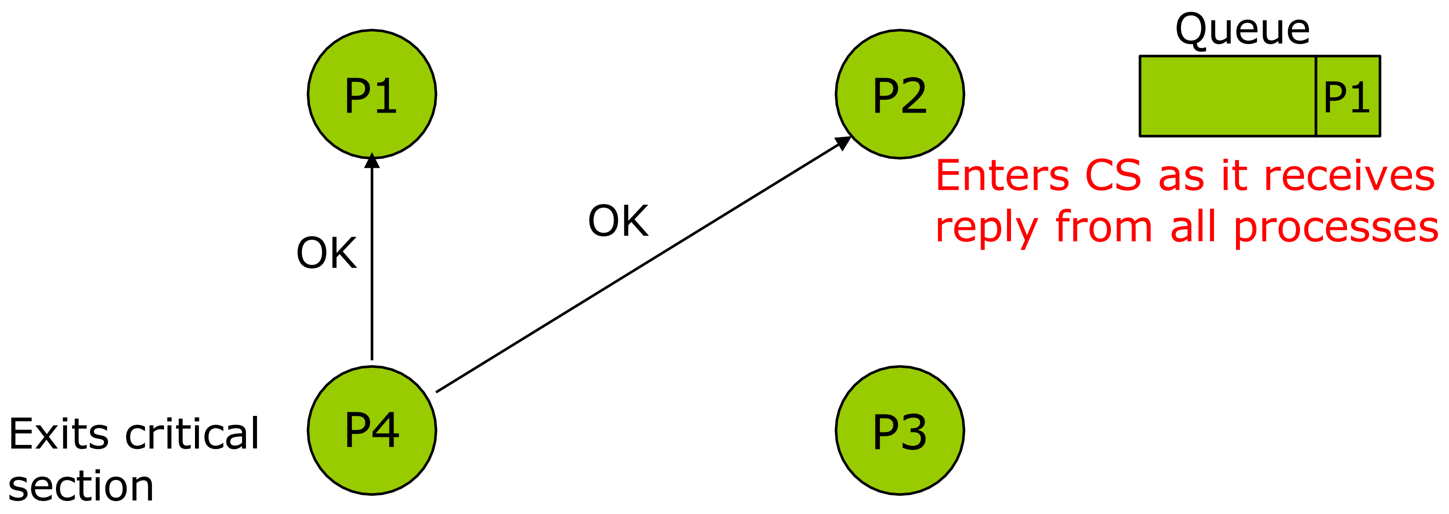
(a) **Status when processes P1 and P2 send request messages to other processes while process p4 is already in the CS.**

Cont…


Queue
P2
P1
Defers sending a reply to P1 and P2
P1
Ok
P2
OK
OK
Queue
P1
Defers sending a reply to P1
P4
P3

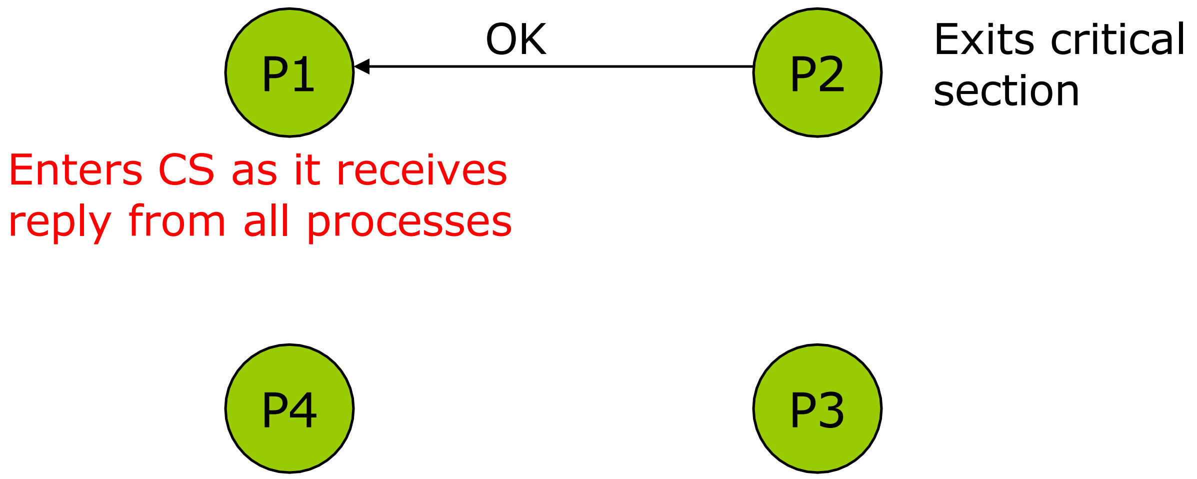
(b) **Status when process P4 is still in the CS.**

Cont…



(c) Status after process **P4 exits CS**

Cont…



(d) Status after **process P2 exits CS**

Cont…

- Ensures **mutual exclusion**.
- Ensures freedom from starvation.
- free from **deadlock**.

- For n processes, the algorithm requires
  - (n-1) request messages
  - (n-1) reply messages,
  - Thus, 2(n-1) messages per CS entry.

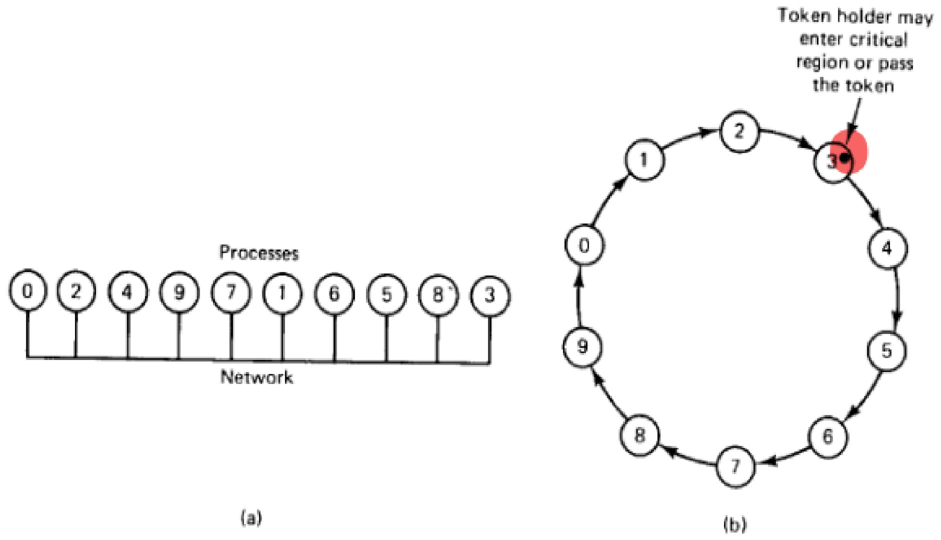
Cont…

- Drawbacks:
  1. In a system has n processes, the algorithm is liable to n points of failure (indefinite wait as failed process will not reply).
  2. Requirement that each process knows the identity of all the processes participating in the mutual-exclusion algorithm. (makes implementation complex) It is suitable only for groups whose member processes are fixed.
  3. waiting time from the moment the process makes a request to enter a CS until it actually enters CS is the time for exchanging 2(n-1) messages in a system having n processes.

Token Ring Algorithm

Assume:

- bus network, with no inherent ordering of the processes.
- a logical ring is constructed in which each process is assigned a position in the ring
- The ring positions may be allocated in numerical order of network addresses or some other means.
- It does not matter what the ordering is.
- All that matters is that each process knows who is next in line after itself.



Token Ring Algorithm

- When the ring is initialized, process 0 is given a token.
- The token circulates around the ring.
- It is passed from process k to process k+1 (modulo the ring size) in point-to-point messages.
- When a process acquires the token from its neighbor,
  - It checks to see if it is attempting to enter a critical region.
  - If so,
    1. the process enters the region,
    2. does all the work it needs to,
    3. leaves the region.
    4. After it has exited, it passes the token along the ring.
  - If not,
    1. it just passes it along [=> when no processes want to enter any critical regions, the token just circulates at high speed around the ring]
- It is not permitted to enter a second critical region using the same token.

Token passing approach

- Use of a single token to achieve mutual exclusion.
- Token is circulated among the processes in the system in a ring structure.
- Token: a special type of message that entitles its holder to enter a CS.
- **Mutual exclusion** is guaranteed.

Cont…

- If token holder wants to enter a critical section, it keeps token & enters CS.
- Otherwise, it just passes it along the ring to its neighbor process.

Cont…

- Types of failures:
  1. Process failure
  2. Lost token

Process failure

- Process receiving token from neighbour always sends an ACK
- Each node maintains current ring configuration
- If process fails, dynamic reconfiguration of ring is carried out.
- When process turns alive, it simply informs to others.

Lost Token

- To regenerate lost token – one process in the ring acts as a Monitor process.
- Monitor periodically circulates “who has the token message”
- Process containing token inserts its id in the special field
- If no id found implies token lost.
- Problems:
  - Monitor process may itself fail
  - "Who has the token" message may be lost

A Comparison of the Three Algorithms

Algorithm	*Messages*	*Delay before entry*	*Problems*
Centralized	3	2	Coordinator crash
Distributed	2(*n*-1)	2(*n*-1)	Crash of any process
Token ring	1 to infinity	0 to *n* -1	Lost token, process crash

Messages:

- The centralized algorithm is simplest and also most efficient:
  - requires only three messages to enter and leave a critical region: a request and a grant to enter, and a release to exit.
- The distributed algorithm
  - requires n -1 request messages, one to each of the other processes,
  - and an additional *n* -1 grant messages
- With the token ring algorithm, the number is variable.
  - If every process constantly wants to enter a critical region, then each token pass will result in one entry and exit, for an average of one message per critical region entered.
  - At the other extreme, the token may sometimes circulate for hours without anyone being interested in it => no. messages per entry into a critical region is unbounded.

A Comparison of the Three Algorithms

- Delay from the moment a process needs to enter a critical region until its actual entry also varies for the 3 algorithms:
  - When critical regions are short and rarely used, the dominant factor in the delay is the actual mechanism for entering a critical region.
  - When they are long and frequently used, the dominant factor is waiting for everyone else to take their turn.
  - It takes only 2 message times to enter a critical region in the centralized case, and 2(*n* -1) message times in the distrib. case,
    - assuming that the network can handle only one message at a time.
  - For the token ring, the time varies from 0 (token just arrived) to *n*-1 (token just departed).
- Event of crashes

whereas n-1 messages are needed when the process wants to enter CS just after it has passed the token to its neighbor process

  - Distributed alg
    centralized one.
  - In a fault-tolerant system, none of these would be suitable, but if crashes are very infrequent, they are all acceptable.