Unit-6

Content:

- Clock Synchronization
- Logical Clocks
- Mutual Exclusion
- Election Algorithm
- Location system
- Distributed Event Matching
- Gossip-based coordination

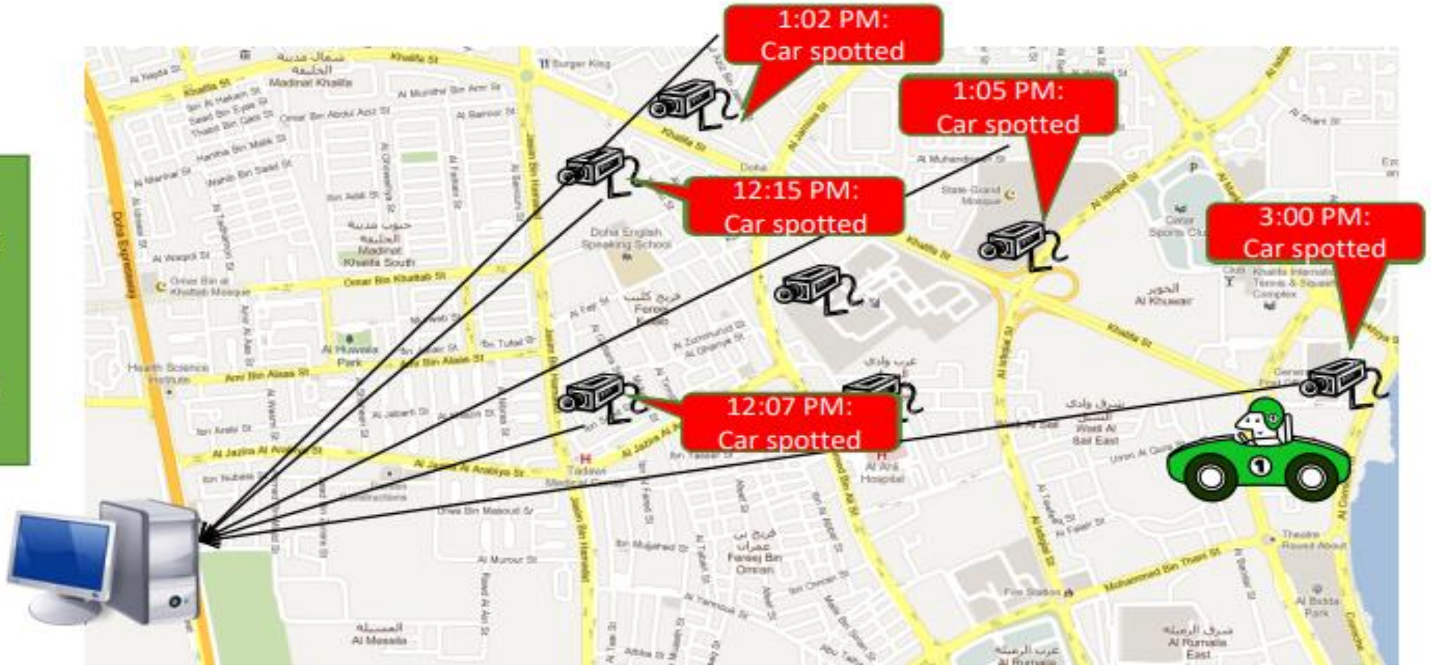
Coordination:

- In distributed coordination-based systems, the focus is on how coordination between the processes takes place.
- The coordination part of a distributed system handles the communication and cooperation between processes.

Synchronization

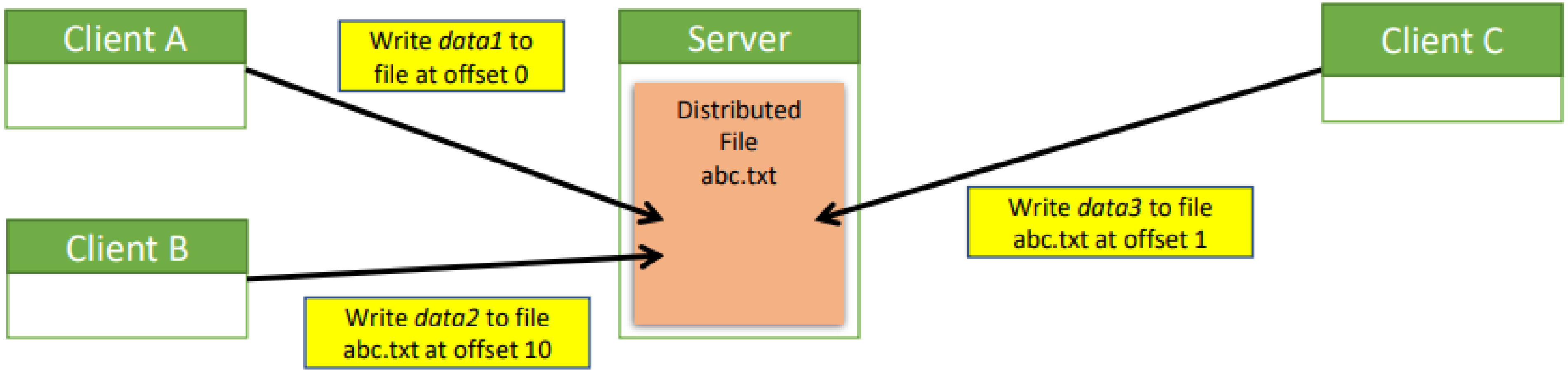
- Vehicle tracking in a City Surveillance System using a Distributed Sensor Network of Cameras
    - *Objective:* To keep track of suspicious vehicles
    - Camera Sensor Nodes are deployed over the city
    - Each Camera Sensor that detects the vehicle reports the time to a central server
    - Server tracks the movement of the suspicious vehicle

If the sensor nodes do not have a consistent version of the time, the vehicle cannot be reliably tracked



Need for Synchronization – Example 2

- Writing a file in a Distributed File System



Client A
Write *data1* to file at offset 0
Server
Client C
Distributed File abc.txt
Write *data3* to file abc.txt at offset 1
Client B
Write *data2* to file abc.txt at offset 10

If the distributed clients do not synchronize their write operations to the distributed file, then the data in the file can be corrupted

- **Synchronization:**
- Cordination of action between performance is synchronization.
- for mutual exclusion
- for event ordering
- forreducing time and loss of data.
- Distributed System is a collection of computers connected via a high-speed communication network.
- In the distributed system, the hardware and software components communicate and coordinate their actions by message passing. Each node in distributed systems can share its resources with other nodes.
- So, there is a need for proper allocation of resources to preserve the state of resources and help coordinate between the several processes.
- To resolve such conflicts, synchronization is used. Synchronization in distributed systems is achieved via clocks.
- The physical clocks are used to adjust the time of nodes. Each node in the system can share its local time with other nodes in the system. The time is set based on UTC (Universal Time Coordination). UTC is used as a reference time clock for the nodes in the system.
- Clock synchronization can be achieved by 2 ways: **External** and **Internal** Clock Synchronization.

- **External clock synchronization** is the one in which an external reference clock is present. It is used as a reference and the nodes in the system can set and adjust their time accordingly.
- **Internal clock synchronization** is the one in which each node shares its time with other nodes and all the nodes set and adjust their times accordingly.

There are 2 types of clock synchronization algorithms:

Centralized and Distributed.

Centralized event ordering is clear because all events are timed by the same clock**. Centralized** is the one in which a time server is used as a reference. The single time-server propagates it's time to the nodes, and all the nodes adjust the time accordingly. It is dependent on a single time-server, so if that node fails, the whole system will lose synchronization. Examples of centralized are-Berkeley the Algorithm, Passive Time Server, Active Time Server etc.

Distributed

- littlebit harder from centralize
- no shared memory
- no common clock
- is the one in which there is no centralized time-server present. Instead, the nodes adjust their time by using their local time and then, taking the average of the differences in time with other nodes. Distributed algorithms overcome the issue of centralized algorithms like scalability and single point failure. Examples of Distributed algorithms are – Global Averaging Algorithm, Localized Averaging Algorithm, NTP (Network time protocol), etc.

Centralized clock synchronization algorithms suffer from two major drawbacks:

They are subject to a single-point failure. If the time-server node fails, the clock synchronization operation cannot be performed. This makes the system unreliable. Ideally, a distributed system should be more reliable than its individual nodes. If one goes down, the rest should continue to function correctly.

From a scalability point of view, it is generally not acceptable to get all the time requests serviced by a single-time server. In a large system, such a solution puts a heavy burden on that one process.

Introduction to clock synchronization

- Clock synchronization is the mechanism to synchronize the time of all the computers in the distributed environments or system.
- Assume that there are three systems present in a distributed environment. To maintain the data i.e. to send, receive and manage the data between the systems with the same time in synchronized manner you need a clock that has to be synchronized. This process to synchronize data is known as Clock Synchronization.

Synchronization in distributed system is more complicated than in centralized system because of the use of distributed algorithms.

- **Properties of Distributed algorithms to maintain Clock synchronization:**

- Relevant and correct information will be scattered among multiple machines.
- The processes make the decision only on local information.
- Failure of the single point in the system must be avoided.
- No common clock or the other precise global time exists.

As the distributed systems has its own clocks. The time among the clocks may also vary. So, it is possible to synchronize all the clocks in distributed environment.

Types of Clock Synchronization

- Physical clock synchronization
- Logical clock synchronization
- Mutual exclusion synchronization
- **Physical clock synchronization:**
- All computer have their own clock
- The physical clocks are needed to adjust the time of nodes. All the nodes in the system can share their local time with all other nodes in the system.
- The time will be set based on UTC (Universal Coordinate Timer).
- The time difference between the two computers is known as “Time drift”. Clock drifts over the time is known as “Skew”. Synchronization is necessary here.

Several methods are used to attempt the synchronization of the physical clocks in Distributed synchronization:

- UTC (Universal coordinate timer)
- Christian’s algorithm
- Berkely’s algorithm

Universal Coordinate Time (UTC)

- All the computers are generally synchronized to a standard time called Coordinated Universal Time (UTC)
- UTC is the primary time standard by which the world regulates clocks and time
- UTC is broadcasted via the satellites
- UTC broadcasting service provides an accuracy of 0.5 msec
- Computer servers and online services with UTC receivers can be synchronized by satellite broadcasts
- Many popular synchronization protocols in distributed systems use UTC as a reference time to synchronize clocks of computers

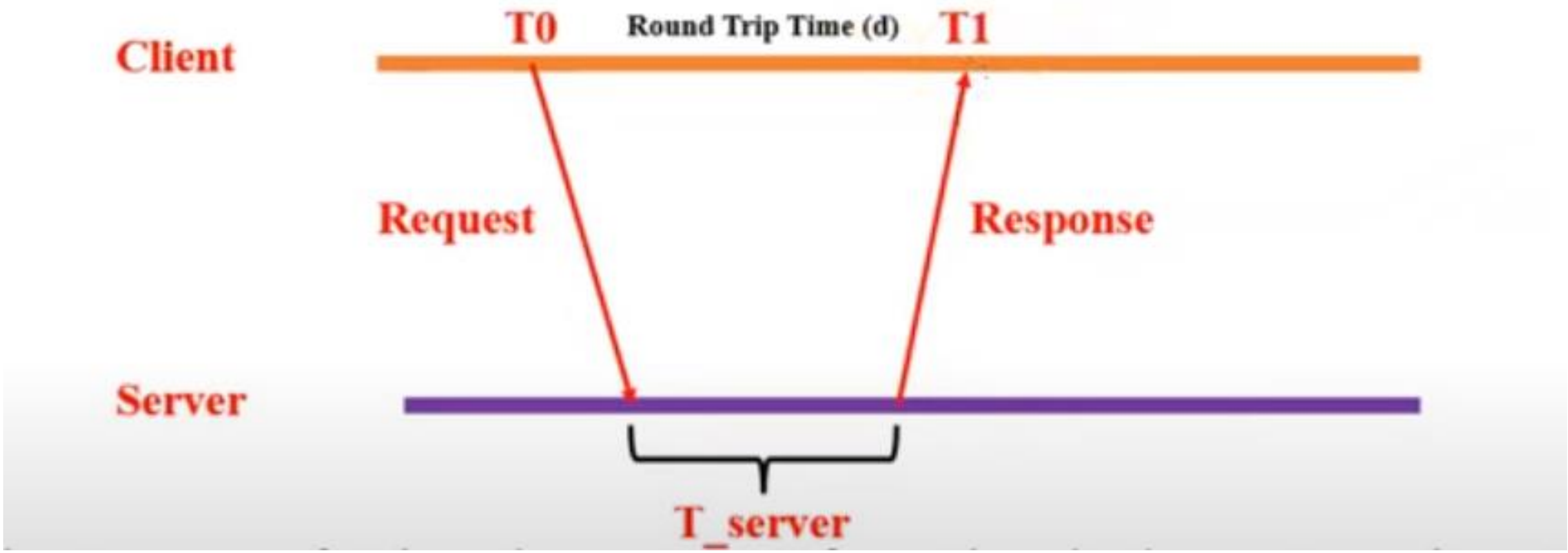
Christian's algorithm

- Flaviu Cristian (in 1989) provided an algorithm to synchronize networked computers with a time server
- **The basic idea:**
- Identify a network time server that has an accurate source for time (e.g., the time server has a UTC receiver)
- All the clients contact the network time server for synchronization
- However, the network delays incurred while the client contacts the time server will have outdated the reported time
- The algorithm estimates the network delays and compensates for it

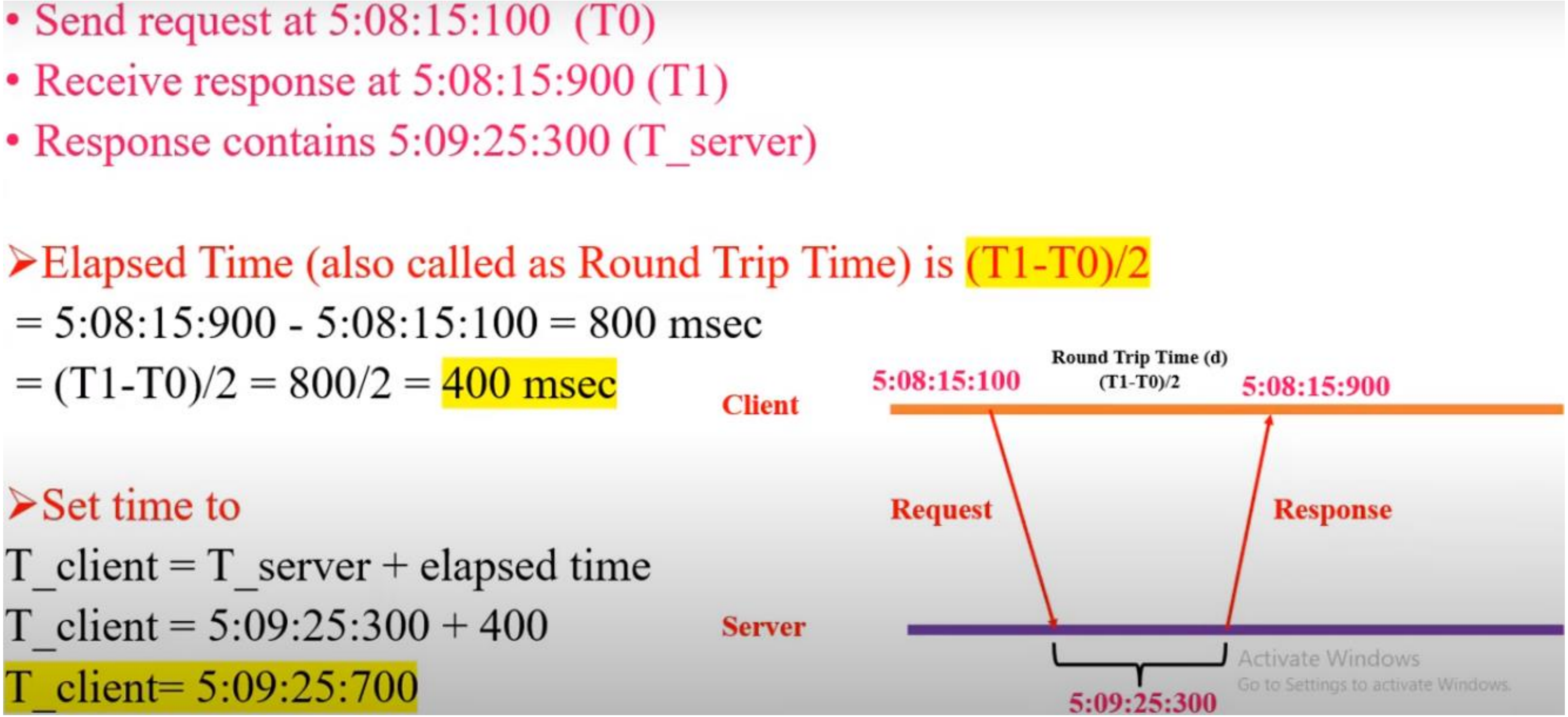
Algorithm:

- The process on the client issues RPC to the Time Server at time T0 to obtain the time.
- In response to the client process's request, the clock server listens and responds with clock server time.
- The client process retrieves the response from the clock server at time T_1 and uses the formula below to determine the synchronized client clock time.
- **$T_{CLIENT} = T_{SERVER}$ plus $(T_1 - T_0)/2$.**

- Client makes a request T0 in d/2 seconds
- d=max difference between the clock and UTC.



Example



- Send request at 5:08:15:100 (T0)
- Receive response at 5:08:15:900 (T1)
- Response contains 5:09:25:300 (T_server)

➢Elapsed Time (also called as Round Trip Time) is (T1-T0)/2

= 5:08:15:900 - 5:08:15:100 = 800 msec

= (T1-T0)/2 = 800/2 = 400 msec

➢Set time to

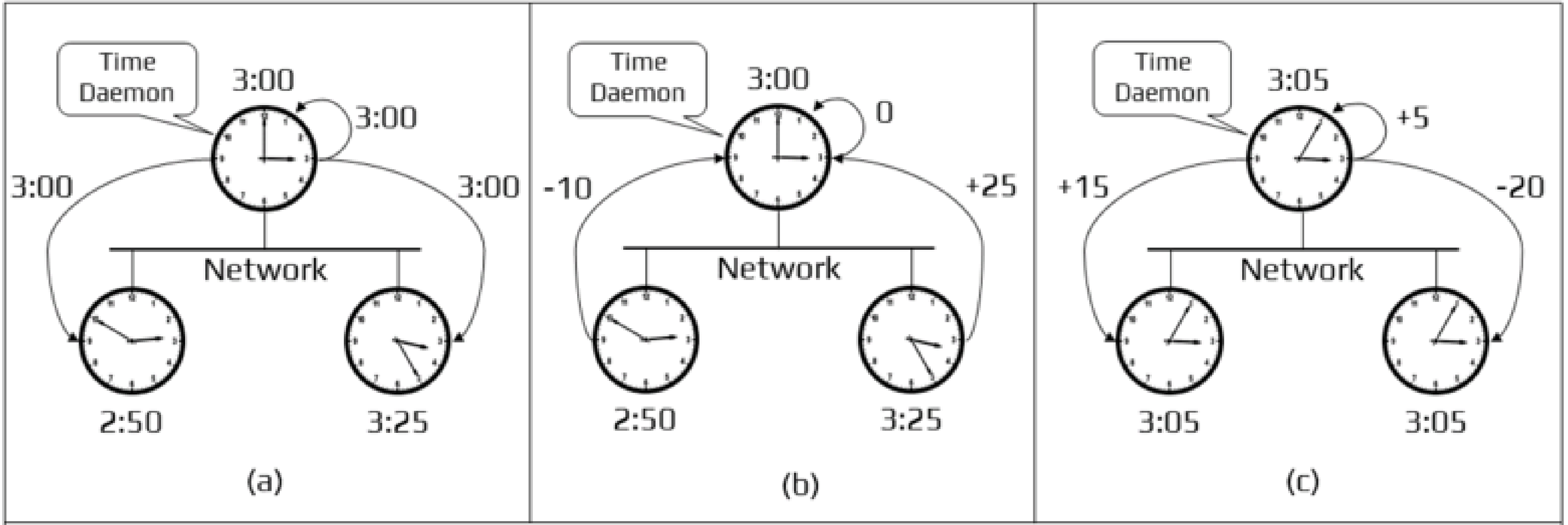
T_client = T_server + elapsed time

T_client = 5:09:25:300 + 400

T_client= 5:09:25:700

Berkeley Algorithm basics:

- Berkeley's Algorithm is aa physical clock synchronization technique used in distributed systems.
- **Assumptions:** Each machine node in the network doesn't have accurate time source.

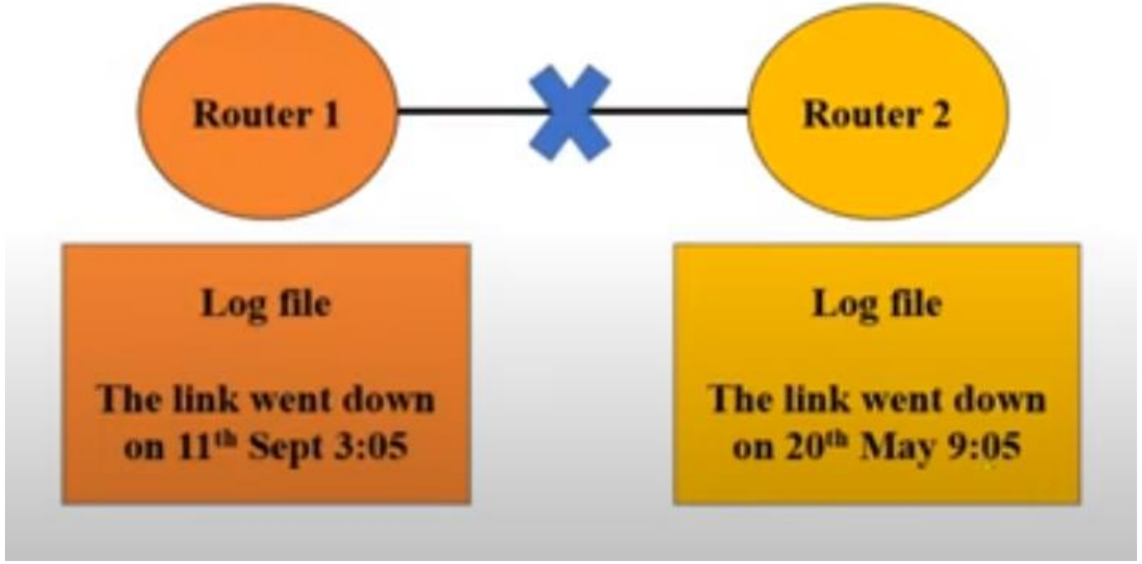


(a) The time daemon asks all the other machines for their clock values.
(b) The machines answer.
(c) The time daemon tells everyone how to adjust their clock.

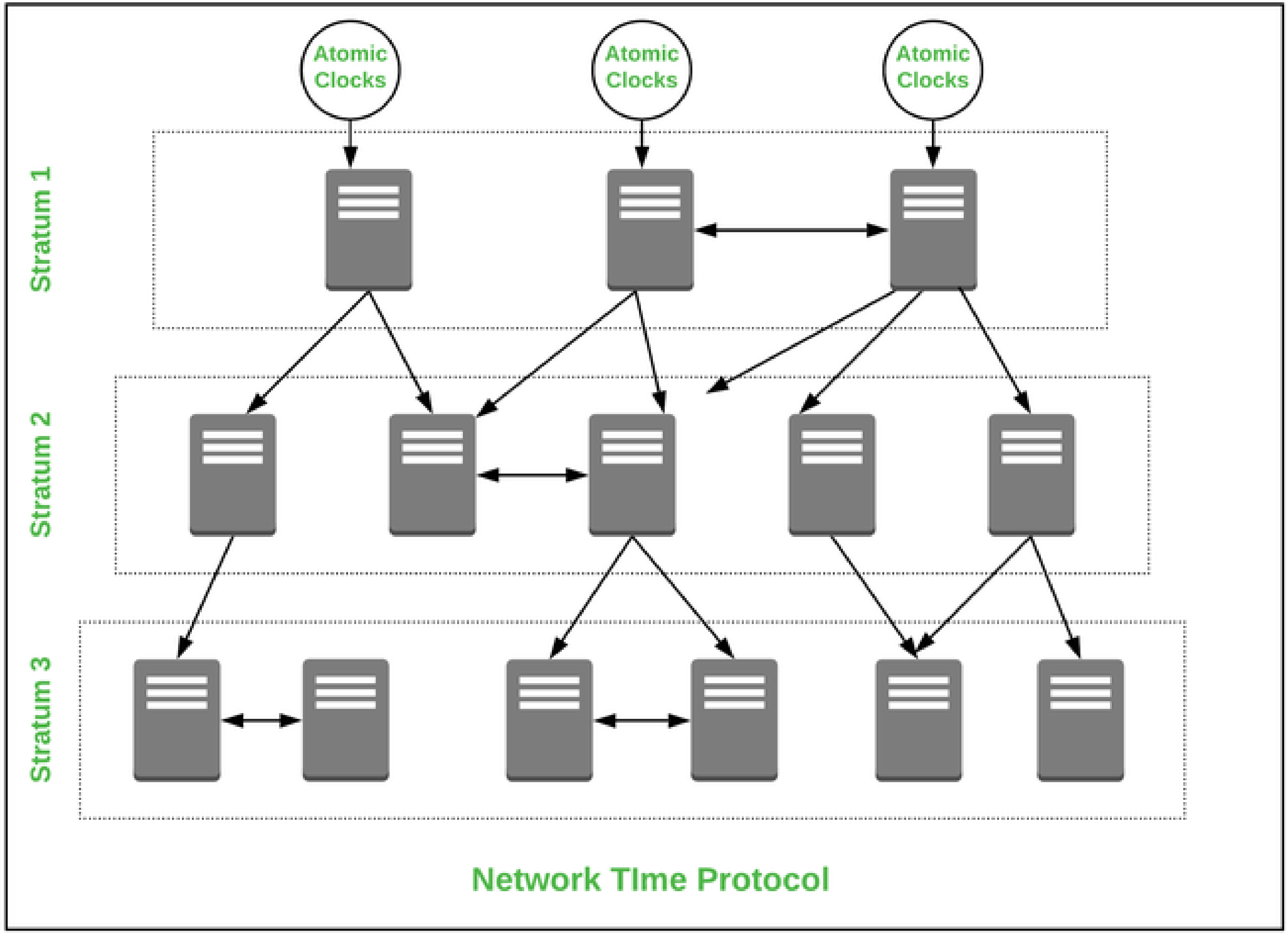
Berkeley's Algorithm:

1. An individual node is chosen as the master node form a pool nodes in the network. This node is the main node in the network which acts as a master and rest of the nodes act as slaves.
2. Master node periodically pings the slave node and asks for the time in their clock.
3. When the slave nodes send their responses, Master node calculates average time difference between all the clock times received and the clock time given by master's system clock itself.
4. This average time difference is added to the current time at master's system clock and broadcasted over the network.
5. Algorithm has provisions for ignoring readings from the clocks whose skew is too large.

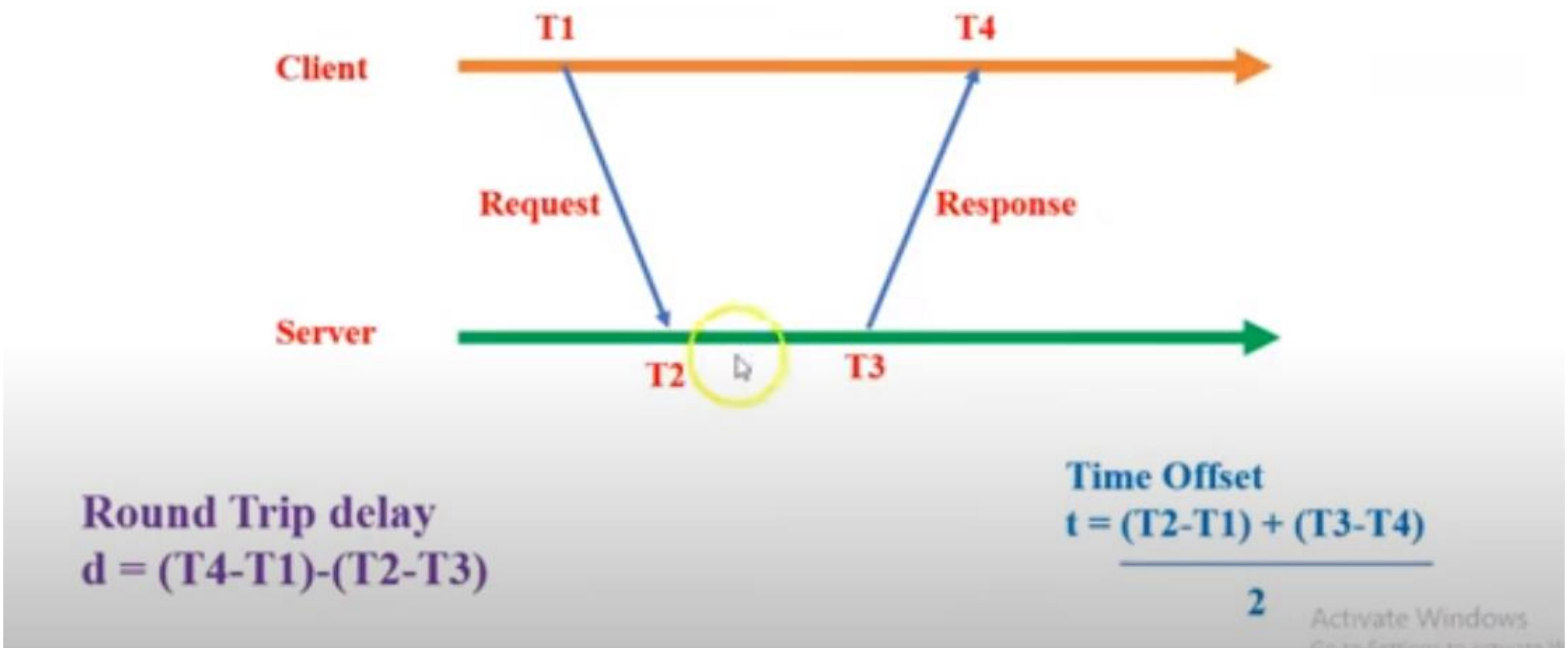
- **Network Time Protocol Algorithm:**
- **Network Time Protocol** (NTP) is a protocol that helps the computers clock times to be synchronized in a network.
- **NTP** algorithm is a physical clock synchronization technique used in distributed system.
- It is decentralized algorithm.
- Every device in the network will have their own internal clock.
- Consider example:



Router 1
Router 2
Log file
The link went down on 11th Sept 3:05
Log file
The link went down on 20th May 9:05



At the top – reference cock
Stratum value – 0 to 15
Stratum specifies the accuracy of clock

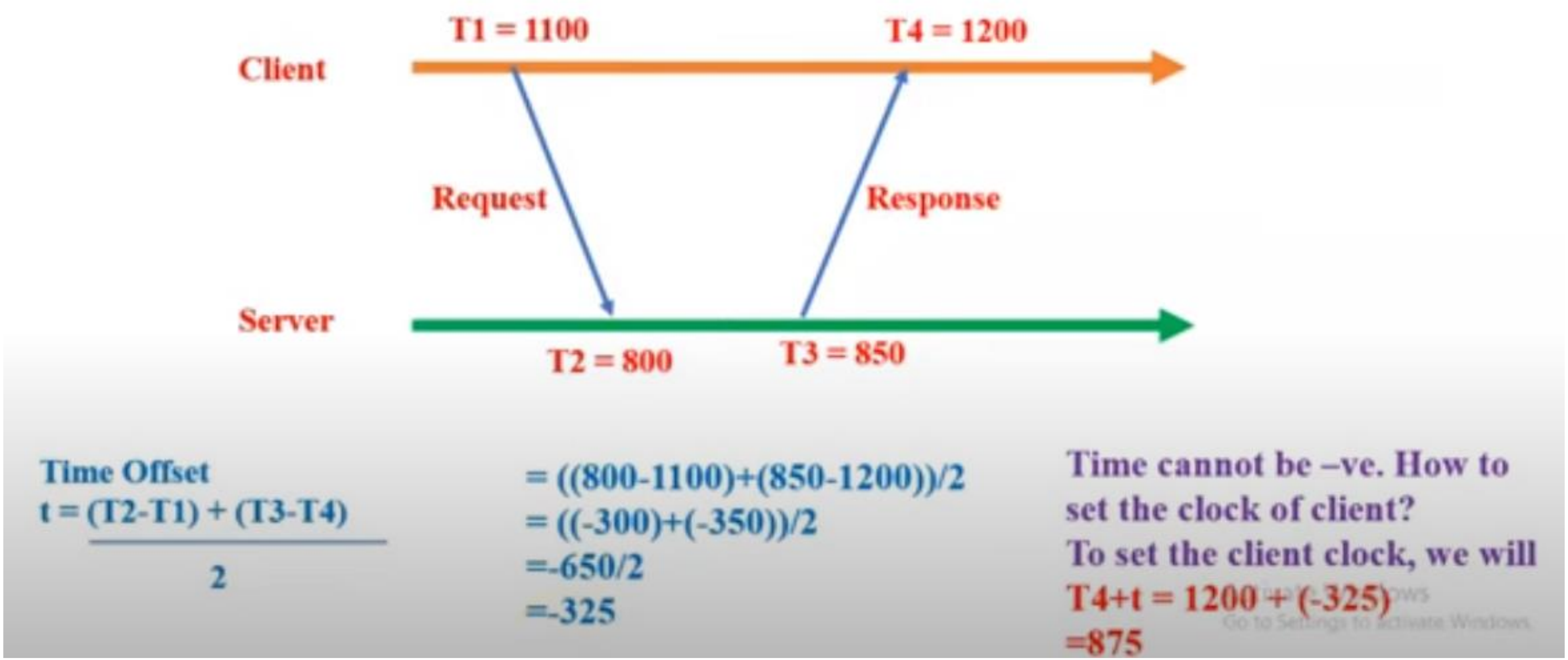


Round Trip delay

$d = (T4-T1)-(T2-T3)$

Time Offset

$$t = \frac{(T2-T1) + (T3-T4)}{2}$$



Time Offset

$$t = \frac{(T2-T1) + (T3-T4)}{2}$$

$= ((800-1100)+(850-1200))/2$

$= ((-300)+(-350))/2$

$=-650/2$

$=-325$

Time cannot be –ve. How to set the clock of client?

To set the client clock, we will

$T4+t = 1200 + (-325)$

$=875$

Why Logical Clocks?

- Lamport (in 1978) showed that:
- Clock synchronization is not necessary in all scenarios
- If two processes do not interact, it is not necessary that their clocks are synchronized
- Many times, it is sufficient if processes agree on the order in which the events has occurred in a DS
- For example, for a distributed make utility, it is sufficient to know if an input file was modified before or after its object file

Logical Clocks

- Logical clocks are used to define an order of events without measuring the physical time at which the events occurred .

- **Logical Clocks** refer to implementing a protocol on all machines within your distributed system, so that the machines are able to maintain consistent ordering of events within some virtual timespan.
- Distributed systems may have no physically synchronous global clock, so a logical clock allows global ordering on events from different processes in such systems.
- **Example :**
  If we go outside then we have made a full plan that at which place we have to go first, second and so on. We don’t go to second place at first and then the first place. We always maintain the procedure or an organization that is planned before. In a similar way, we should do the operations on our PCs one by one in an organized way.
- Suppose, we have more than 10 PCs in a distributed system and every PC is doing it’s own work but then how we make them work together. There comes a solution to this i.e. LOGICAL CLOCK.

- **Method-1:**
  To order events across process, try to sync clocks in one approach.
- This means that if one PC has a time 2:00 pm then every PC should have the same time which is quite not possible. Not every clock can sync at one time. Then we can't follow this method.
- **Method-2:**
  Another approach is to assign Timestamps to events.
- Taking the example into consideration, this means if we assign the first place as 1, second place as 2, third place as 3 and so on. Then we always know that the first place will always come first and then so on. Similarly, If we give each PC their individual number than it will be organized in a way that 1st PC will complete its process first and then second and so on.
- BUT, Timestamps will only work as long as they obey causality.

- **What is causality ?**
  Causality is fully based on HAPPEN BEFORE RELATIONSHIP.
- Taking single PC only if 2 events A and B are occurring one by one then TS(A) < TS(B).
- If A has timestamp of 1, then B should have timestamp more than 1, then only happen before relationship occurs.
- Taking 2 PCs and event A in P1 (PC.1) and event B in P2 (PC.2) then also the condition will be TS(A) < TS(B). Taking example- suppose you are sending message to someone at 2:00:00 pm, and the other person is receiving it at 2:00:02 pm.Then it’s obvious that TS(sender) < TS(receiver).

Properties Derived from Happen Before Relationship –

- **Transitive Relation –**
  If, TS(A) <TS(B) and TS(B) <TS(C), then TS(A) < TS(C)
- **Causally Ordered Relation –**
  a->b, this means that a is occurring before b and if there is any changes in a it will surely reflect on b.
- **Concurrent Event –**
  This means that not every process occurs one by one, some processes are made to happen simultaneously i.e., A || B.

- **two types of logical clocks**

1. Lamport’s Clock
2. Vector Clock

1. Lamport’s Clock

- Lamport advocated maintaining logical clocks at the processes to keep track of the order of events
- To synchronize logical clocks, Lamport defined a relation called “happened-before”
- The expression a→b (read as “a happened before b”) means that all entities in a DS agree that event a occurred before event b

- **Algorithm:**
- **Happened before relation(->):** a -> b, means ‘a’ happened before ‘b’.
- **Logical Clock:** The criteria for the logical clocks are:
  - [C1]: C_i (a) < C_i(b), [C_i -> Logical Clock, If ‘a’ happened before ‘b’, then time of ‘a’ will be less than ‘b’ in a particular process.]
  - [C2]: C_i(a) < C_j(b), [Clock value of C_i(a) is less than C_j(b)]

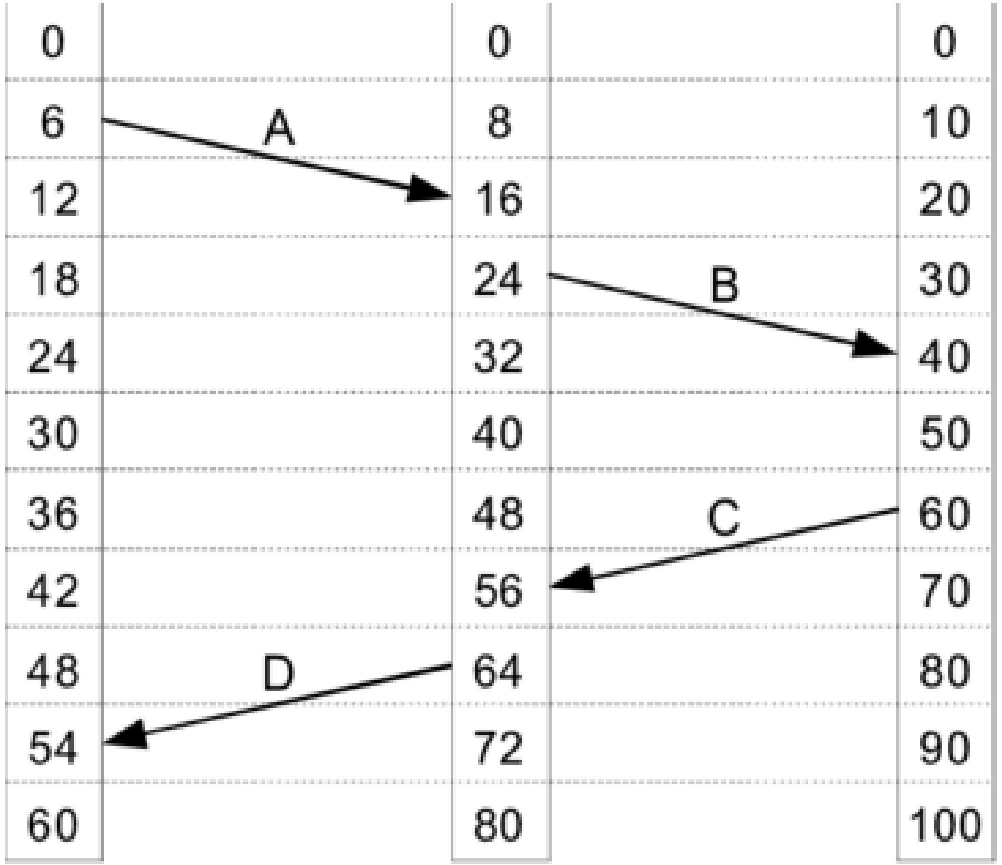
- The **happened-before** relation can be observed directly in two situations:
  1. If a and b are events in the same process, and a occurs before b, then a→b is true
  2. If a is an event of message m being sent by a process, and b is the event of the message m being received by another process, the a→b is true.
- The happened-before relation is transitive • If a→b and b→c, then a→c

- **Time values in Logical Clocks**
- For every event a, assign a logical time value C(a) on which all processes agree
- Time value for events have the property that

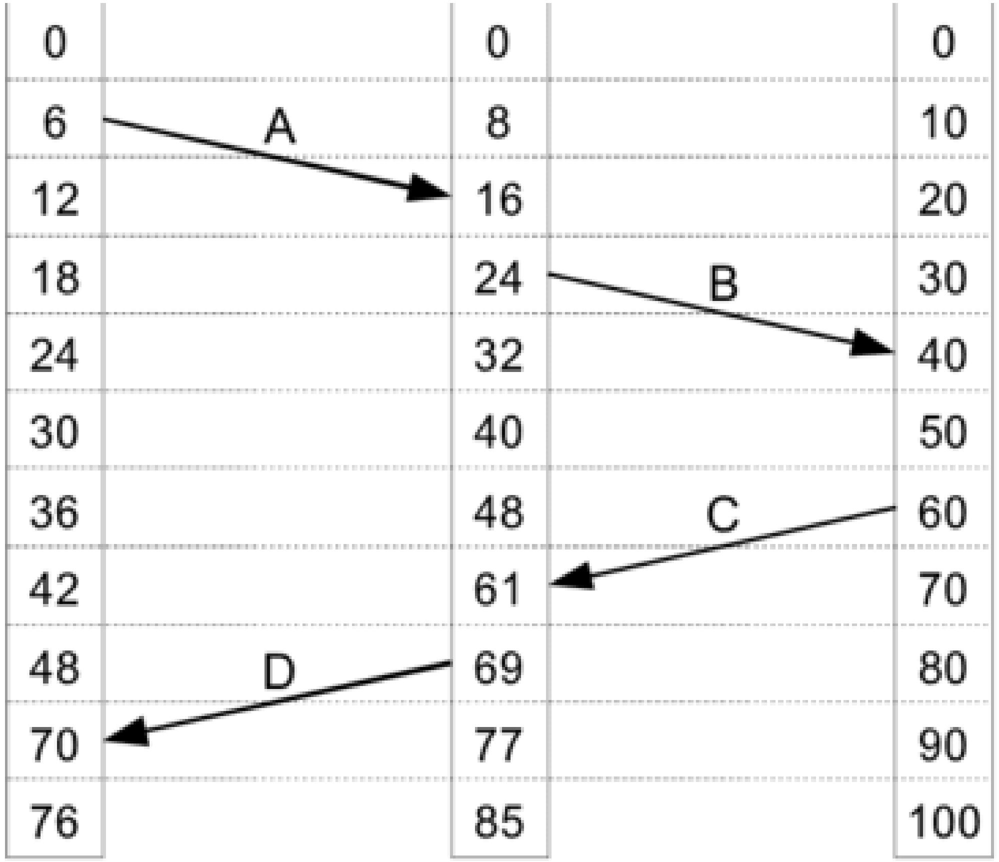
  If a→b, then C(a)< C(b)

Properties of Logical Clock

- From happened-before relation, we can infer that:
- If two events a and b occur within the same process and a→b, then assign C(a) and C(b) such that C(a) < C(b)
- If a is the event of sending the message m from one process, and b is the event of receiving the message m, then
- the time values C(a) and C(b) are assigned such that all processes agree that C(a) < C(b)
- The clock time C must always go forward (increasing)



0	0	0
6	8	10
12	16	20
18	24	30
24	32	40
30	40	50
36	48	60
42	56	70
48	64	80
54	72	90
60	80	100



0	0	0
6	8	10
12	16	20
18	24	30
24	32	40
30	40	50
36	48	60
42	61	70
48	69	80
70	77	90
76	85	100

Vector Clocks

- Vector Clocks was proposed to overcome the limitation of Lamport’s clock:

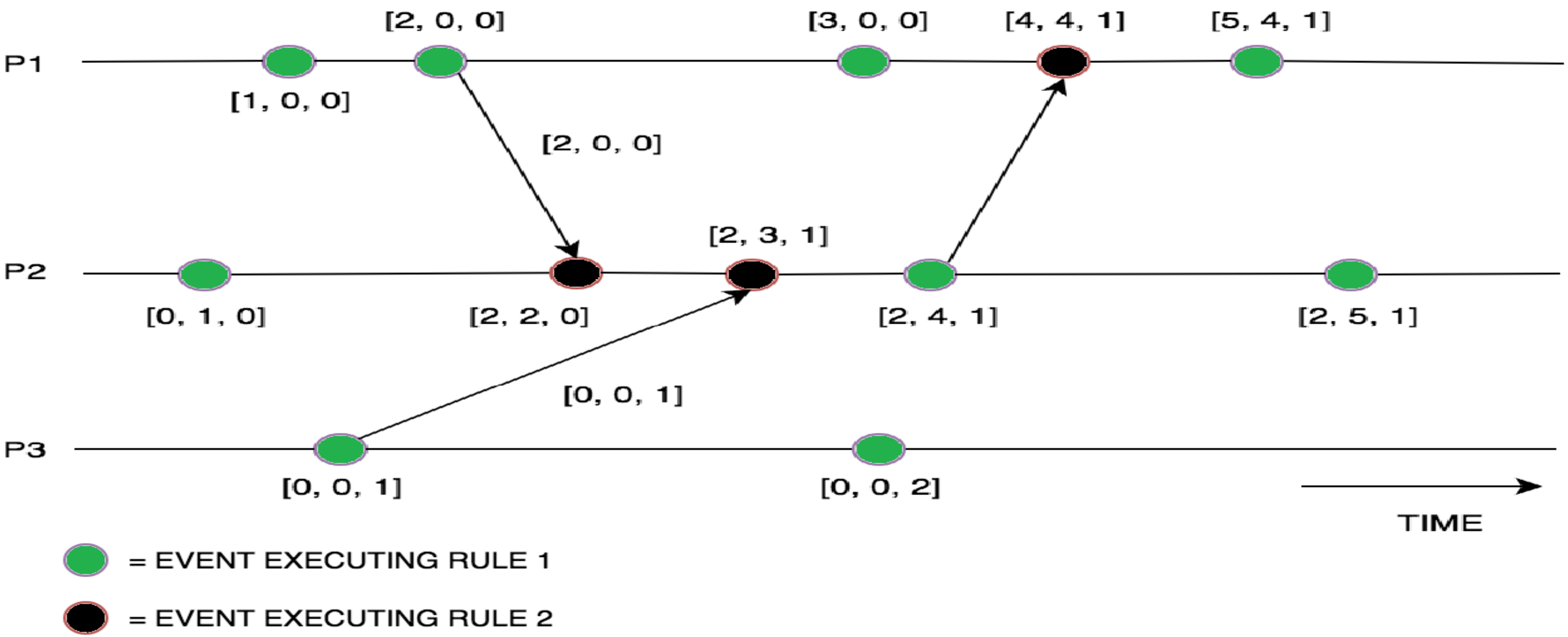
  the fact that C(a)<C(b) does not mean that a→b
- The property of inferring that a occurred before b is called as causality property.
- These clocks expand on Scalar time to facilitate a causally consistent view of the distributed system, they detect whether a contributed event has caused another event in the distributed system
- A Vector clock for a system of N processes is an array of N integers .
- Every process Pi stores its own vector clock VCi

How does the vector clock algorithm work :

- Initially, all the clocks are set to zero.
- Every time, an Internal event occurs in a process, the value of the processes's logical clock in the vector is incremented by 1
- Also, every time a process sends a message, the value of the processes's logical clock in the vector is incremented by 1.

Every time, a process receives a message, the value of the processes's logical clock in the vector is incremented by 1, and moreover, each element is updated by taking the maximum of the value in its own vector clock and the value in the vector in the received message (for every element).

- **Example :**
  Consider a process (P) with a vector size N for each process: the above set of rules mentioned are to be executed by the vector clock:



- To sum up, Vector clocks algorithms are used in distributed systems to provide a **causally consistent** ordering of events but the entire Vector is sent to each process for every message sent, in order to keep the vector clocks in sync.

Mutual Exclusion:

Mutual exclusion is a property of process synchronization which states that “no two processes can exist in the critical section at any given point of time”.

Mutual exclusion methods are used in concurrent programming to avoid the simultaneous use of a common resource, such as a global variable, by pieces of computer code called critical sections

- Requirement of mutual exclusion is that, when process P1 is accessing a shared resource R1 then on other process should be able to access resource R1 until process P1 has finished its operation with resource R1.

Requirements of Mutual exclusion Algorithm:

- **No Deadlock:** Two or more site should not endlessly wait for any message that will never arrive.
- **No Starvation:** Every site who wants to execute critical section should get an opportunity to execute it in finite time. Any site should not wait indefinitely to execute critical section while other site are repeatedly executing critical section
- **Fairness:** Each site should get a fair chance to execute critical section. Any request to execute critical section must be executed in the order they are made i.e Critical section execution requests should be executed in the order of their arrival in the system.
- **Fault Tolerance:** In case of failure, it should be able to recognize it by itself in order to continue functioning without any disruption.

Types of Distributed Mutual Exclusion

- Mutual exclusion algorithms are classified into two categories.

1. Permission-based Approaches

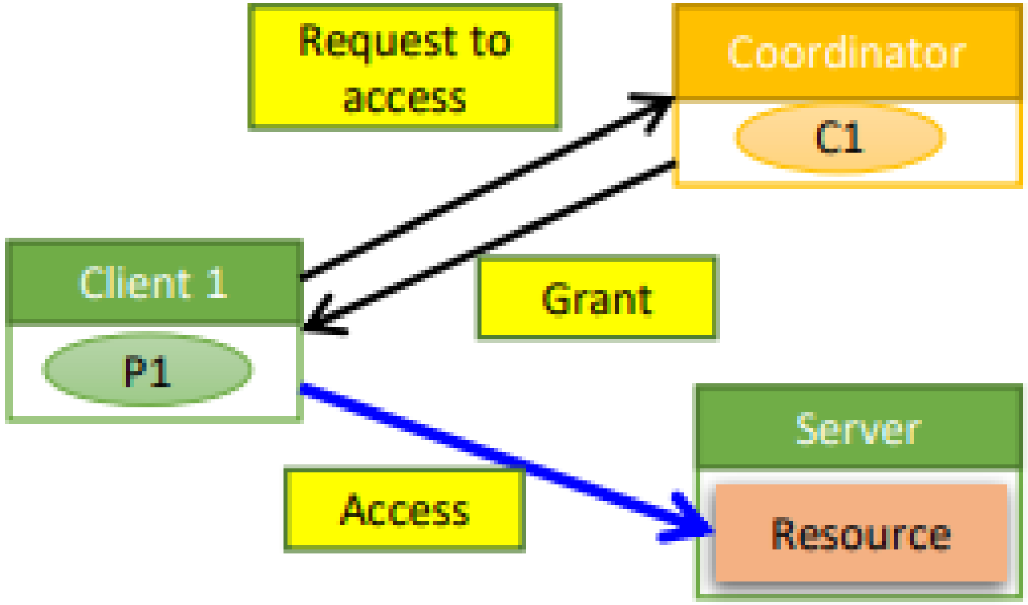
A process, which wants to access a shared resource, requests the permission from one or more coordinators

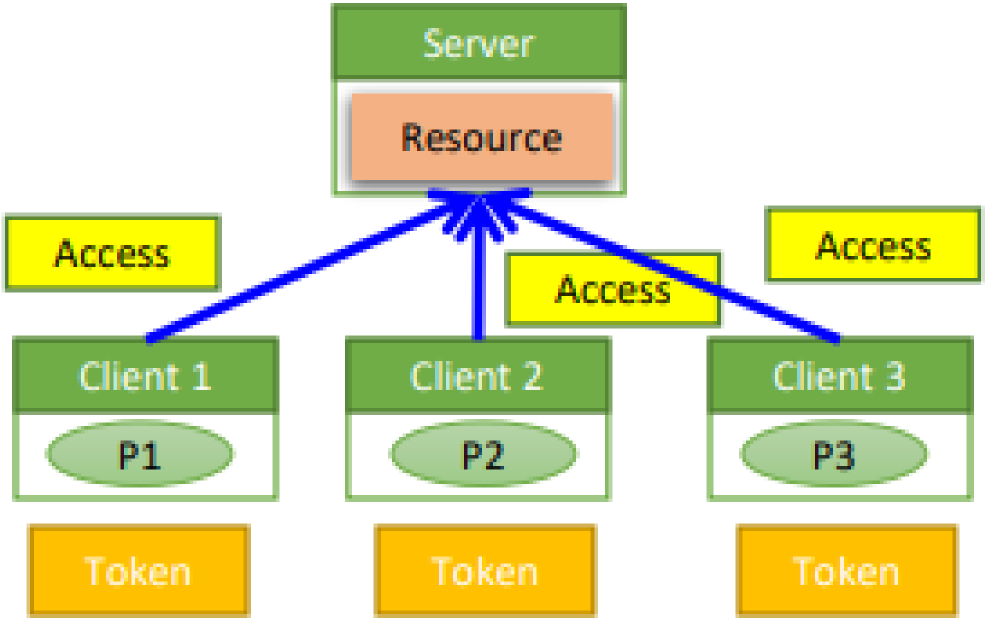
2. **Token-based Approaches**

Each shared resource has a token

Token is circulated among all the processes

A process can access the resource if it has the token


Request to access
Coordinator
C1
Client 1
P1
Grant
Server
Access
Resource


Server
Resource
Access
Access
Access
Client 1
P1
Client 2
P2
Client 3
P3
Token
Token
Token

- **Permission-based Approaches**

There are two types of permission-based mutual exclusion algorithms

a. Centralized Algorithms

b. Decentralized Algorithms

<u>Centralized Algorithms</u>

- One process is elected as a coordinator (C) for a shared resource
- Coordinator maintains a Queue of access requests
- Whenever a process wants to access the resource, it sends a request message to the coordinator to access the resource

- When the coordinator receives the request:

✓ If no other process is currently accessing the resource, it grants the permission to the process by sending a “grant” message

✓If another process is accessing the resource, the coordinator queues the request, and does not reply to the request

- The process releases the exclusive access after accessing the resource
- The coordinator will then send the “grant” message to the next process in the queue

Discussion about Centralized Algorithm

- **Blocking vs. non-blocking requests**

✓The coordinator can block the requesting process until the resource is free

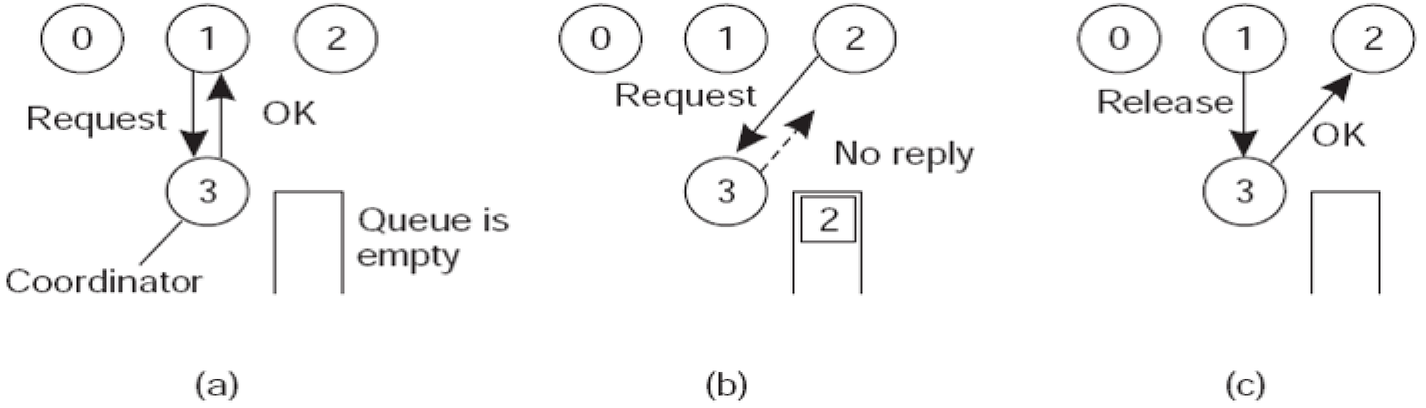
✓Otherwise, the coordinator can send a “permission-denied” message back to the process .

➢The process can poll the coordinator at a later time, or

➢The coordinator queues the request. Once the resource is released, the coordinator will send an explicit “grant” message to the process

- **The algorithm guarantees mutual exclusion, and is simple to implement**

- If process wants to enter the critical, it has to take the permission from the coordinator process. This permission is taking by sending a REQUEST message.



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

b) Process 2 then asks permission to access the same resource. The coordinate does not replay.

c) When process 1 release the resource, it tells the coordinator, which then replies to 2.

- **Distributed algorithm:**
- A distributed algorithm for mutual exclusion is presented.
- No particular assumption on the network topology are required, except connectivity.
- The process communicate by message only and there is no global controller.
- No process need to know or learn the global network topology.

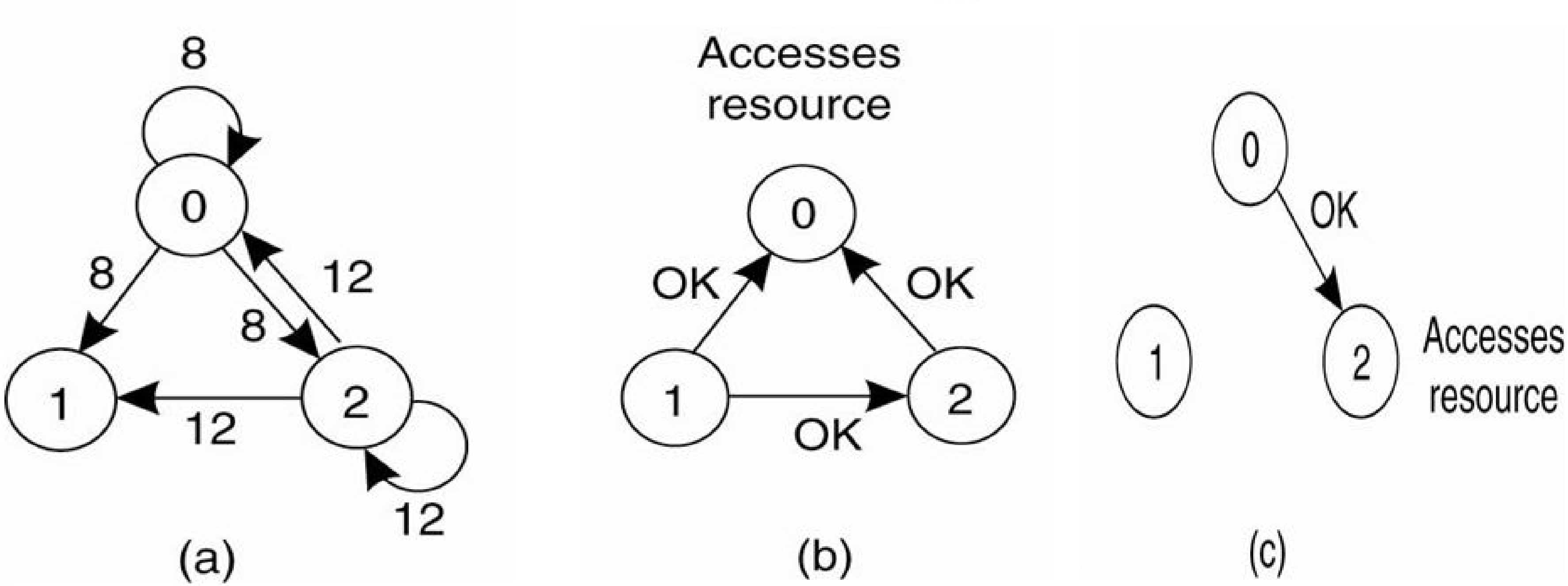
When a process wants to enter a critical region , it builds a message containing.

1. Name of the critical region
2. It’s process number
3. It’s current time

- **Three cases are possible here:**

1. If the message receiving process is not in the critical region and does not wish to enter it, it send it back.
2. Receiver is already in the critical region and does not replay.
3. Receiver wants to enter the same critical region and has not done so, it compares the “time stamp ” of the incoming message with the one it has sent to others for permission. The lowest one wins and can enter the critical region

Mutual Exclusion
A Distributed Algorithm



(a) Two processes want to access a shared resource 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 go ahead.

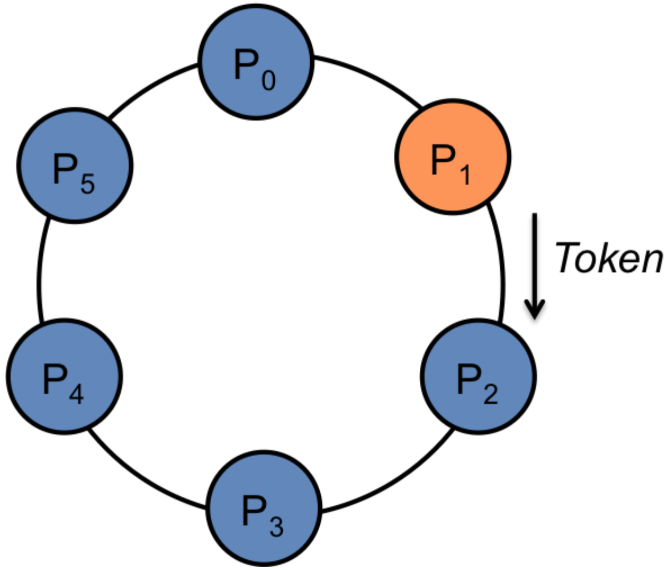
- If one process crashes, it will fail to respond. Thus other process will assume that the process is still working in the critical region which will make other process go through a starvation.
- Here each process must maintain the group membership list that includes process entering or leaving the group.
- Here all the process are involved in all decisions; this lead to bottle neck when the number of processes in the group are more.
- This algorithm is comparatively, expensive, slower and complex.

- **Token ring algorithm:**

Token Ring algorithm achieves mutual exclusion in a distributed system by creating a bus network of processes. A logical ring is constructed with these processes and each process is assigned a position in the ring. Each process knows who is next in line after itself.

Algorithm:

1. Token is initially given to one process.
2. When a process requires to enter the section , it waits until it gets token from its left neighbor and retains it . After it got the token ,it enters the critical section. After it left critical section, it passes token to its neighbor in clockwise direction.
3. If a process gets token but does not require to enter a critical section, it immediately passes token along a ring.



- As usual this algorithm has problems too and some of them are:

1. If adds loads to the network as token should be passed even the process does not need it.
2. If one process fail, no progress is possible until the faulty process is extracted from the ring.
3. Election process should be done if the processed holding the token fails.

- **Election Algorithm:**

It is algorithm which are designed to choose a coordinator.

Election algorithm is an algorithm choosing a unique process to paly a particular role. Any process can be elected but only one process must be elected and all other process must agree on the decision.

Election is done after the failure of current coordination of failure of process currently holding the token. An election process is typically in two phase.

1. Select the leader with highest priority
2. Inform all processed about the winner.

The distributed election algorithm are:

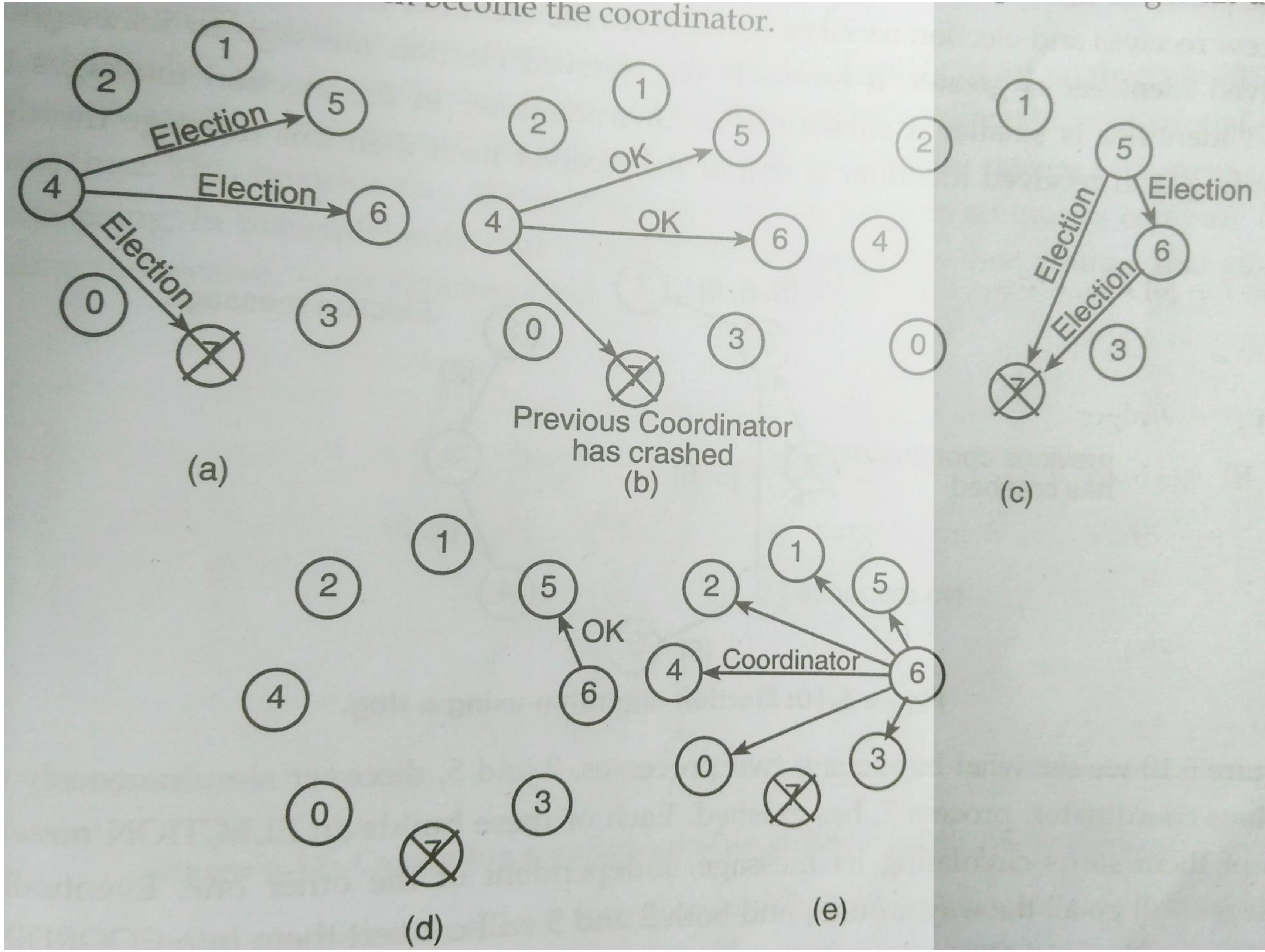
1. Bully algorithm
2. Ring algorithm

- **Bully Algorithm:**

Any process could fail during the election procedure. When any process notices that the coordinator is no longer responding to requests, it initiates an election. A process, P1,holds an election as follows:

1. P1 sends an ELECTION message to all processes with higher numbers.
2. If no node responds, P1 wins the election and becomes coordinator.
3. If one of the higher –ups answer , it takes over and p1’s job is done.

become the coordinator.



- The bully election algorithm:

a) Process 4 holds and election.

b) Process 5 AND 6 respond, telling 4 to stop.

c) Now 5 and 6 each hold the election

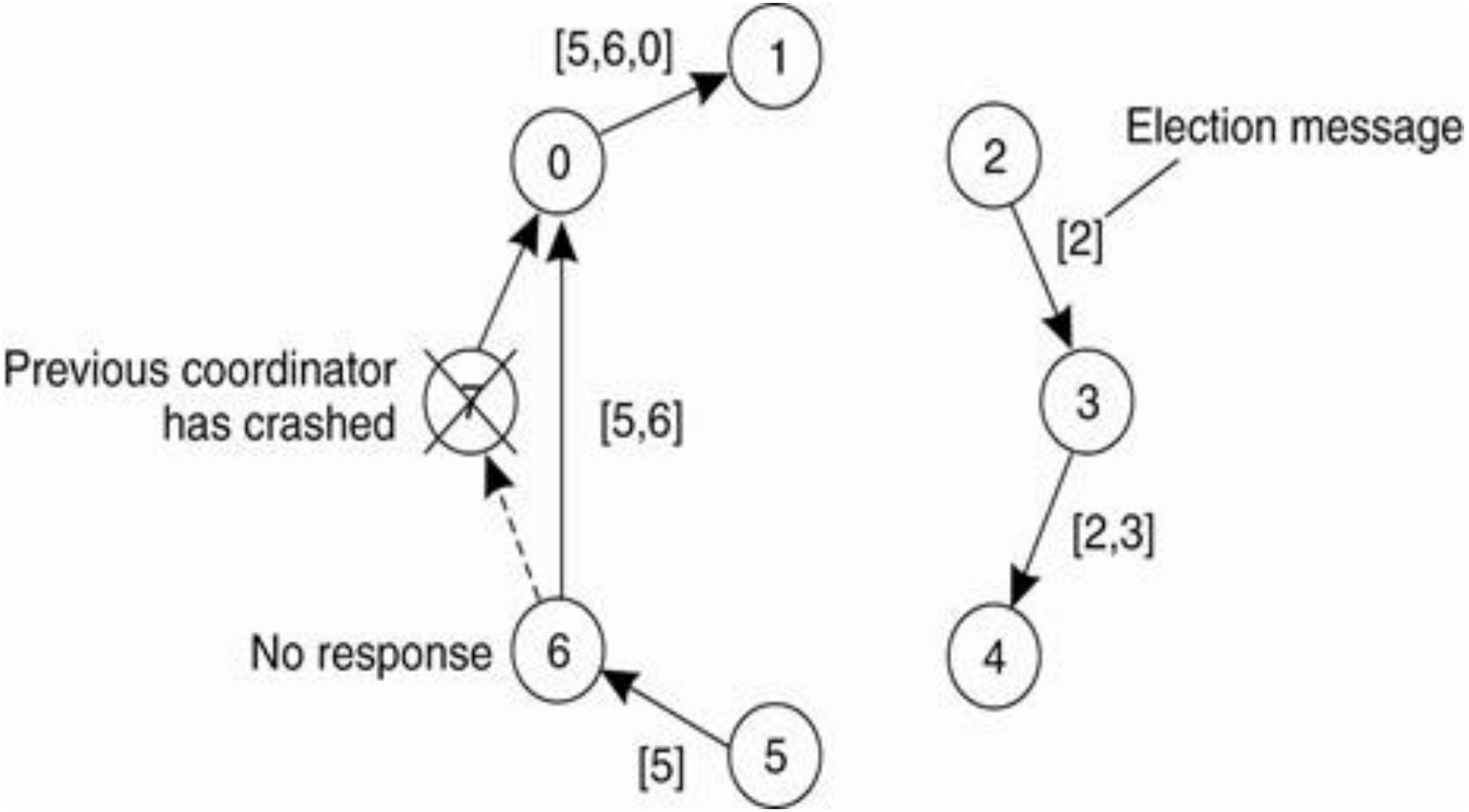
d) Process 6 tells 5 to stop

e) Process 6 wins and tells everyone.

- **Ring Algorithm**

We assume that the process are arranged in a local ring each process knows the address of one process, which is its neighbor in the clockwise direction. The algorithm elects a single coordinator , which is the process with the highest identifier.

- **Algorithm –**
- If process P1 detects a coordinator failure, it creates new active list which is empty initially. It sends election message to its neighbour on right and adds number 1 to its active list.
- If process P2 receives message elect from processes on left, it responds in 3 ways:
    - (I) If message received does not contain 1 in active list then P1 adds 2 to its active list and forwards the message.
    - (II) If this is the first election message it has received or sent, P1 creates new active list with numbers 1 and 2. It then sends election message 1 followed by 2.
    - (III) If Process P1 receives its own election message 1 then active list for P1 now contains numbers of all the active processes in the system. Now Process P1 detects highest priority number from list and elects it as the new coordinator.

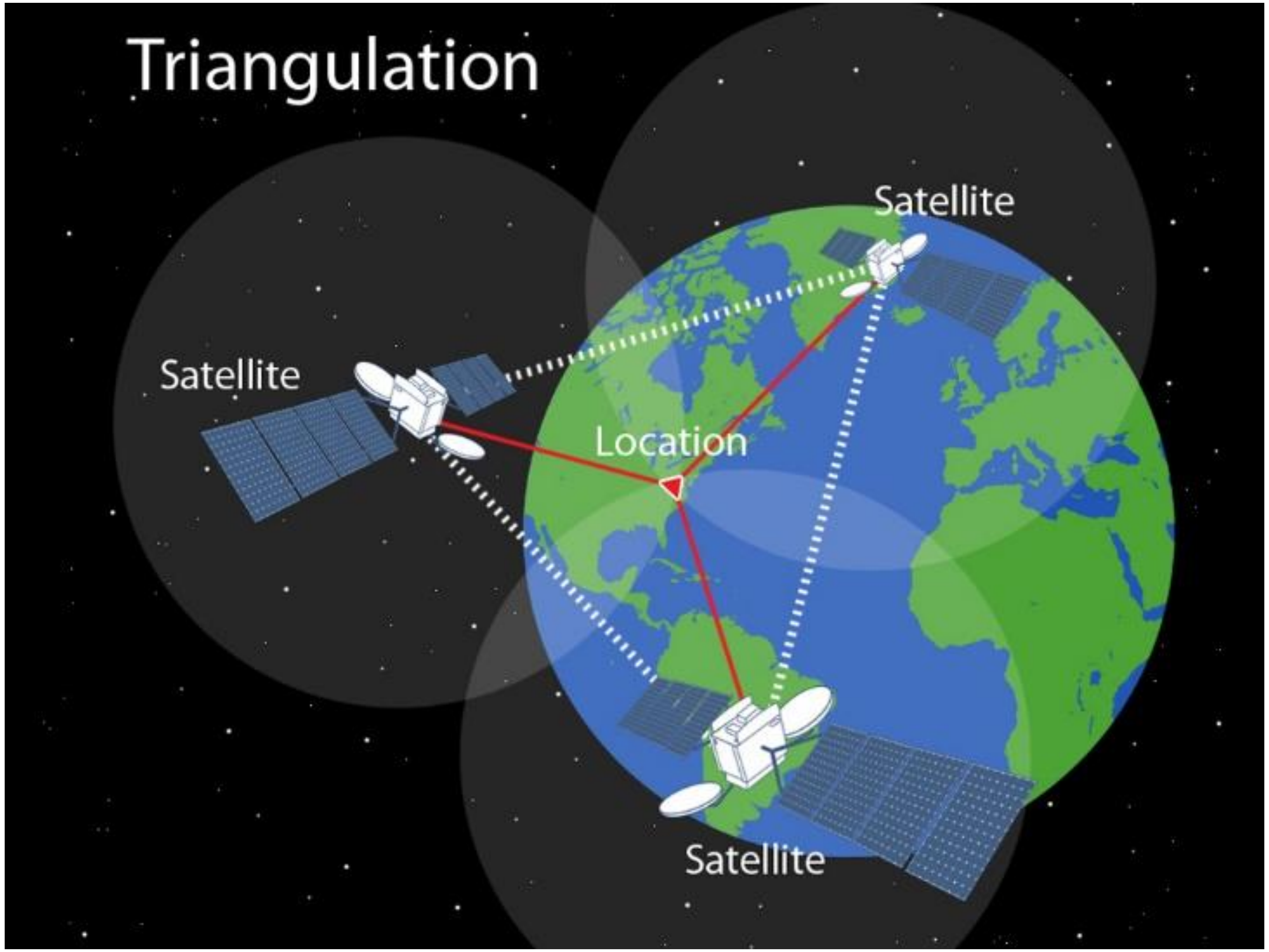

[5,6,0]
1
0
2
Election message
[2]
Previous coordinator
has crashed
7
[5,6]
3
[2,3]
No response
6
4
[5]
5

- **Location System:**

GPS(Global Positioning System):

- It was launched in Under the US air force project in 1979.
- Official name NAVSTAR GPS (Navigation Satellite Timing And Ranging).
- First satellites was launched in 1978 and last satellites was launched in 1995. after that it works properly.
- It was first released in 1980 for civilian used but there were some limitations after 2000 it was fully released with no any limitation.
- GPS provides the information regarding any places by getting the signal from three satellites.
- America launched total 72 satellites for the GPS purpose but only 33 Satellites are working currently.
- 2 revolution per day.

How does GPS work.



- **Gossip Based Coordination:**

Gossip based Coordination is a communication system, it is process where distributed computer are communicate to works.

Aggregation

Large-scale peer Sampling

Overlay Construction

Aggregation:

Gossiping can be used to discover nodes that have a few outgoing wide-area links, to subsequently apply directional gossiping. Another application area is simply collecting or actually aggregating information.

A peer sampling service:

An important aspect in epidemic protocol is the ability of a node P to choose another node Q at random from all available nodes in the network.

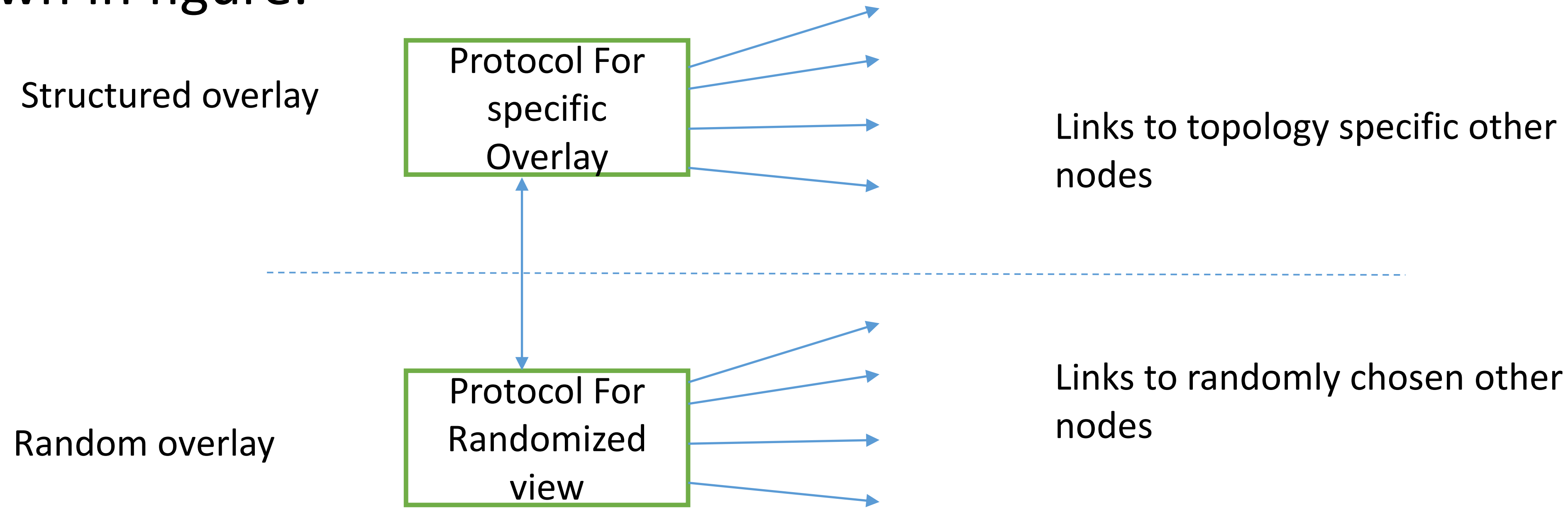
But there might be a problem: if the network consists of thousand of nodes how can p ever pick one of these nodes at random without having a complete over view of network. For similar network one could often resort to a central service that had registered every participating node. This approach can create problem.

- Solution is to create fully decentralized peer-sampling service or PSS for short.
- Each nodes maintain a list of c neighbors, where ideally each of these neighbors represent a randomly chosen live node from the current set of node. This list of neighbor also referred to as partial view.
- In this solution it is assumed that nodes regularly exchange entries from their partial view.

- **Gossip-based Overlay Construction:**

One key Observation is that by carefully exchanging and selecting entries from partial view. It is possible to construct and maintain specific topologies of overlay network.

These topology management is achieved by adopting a two-layered approach, as shown in figure.


Structured overlay
Protocol For specific Overlay
Links to topology specific other nodes
Random overlay
Protocol For Randomized view
Links to randomly chosen other nodes

END OF UNIT 6