# COORDINATION

- Coordination in distributed system **handles the communication and cooperation between processes**.
- It forms the glue that binds the activities performed by processes into a whole
- A system of parallel processes is said to be synchronous if all processes run using the same clock, and it is **asynchronous if each process has its own independent clock**.

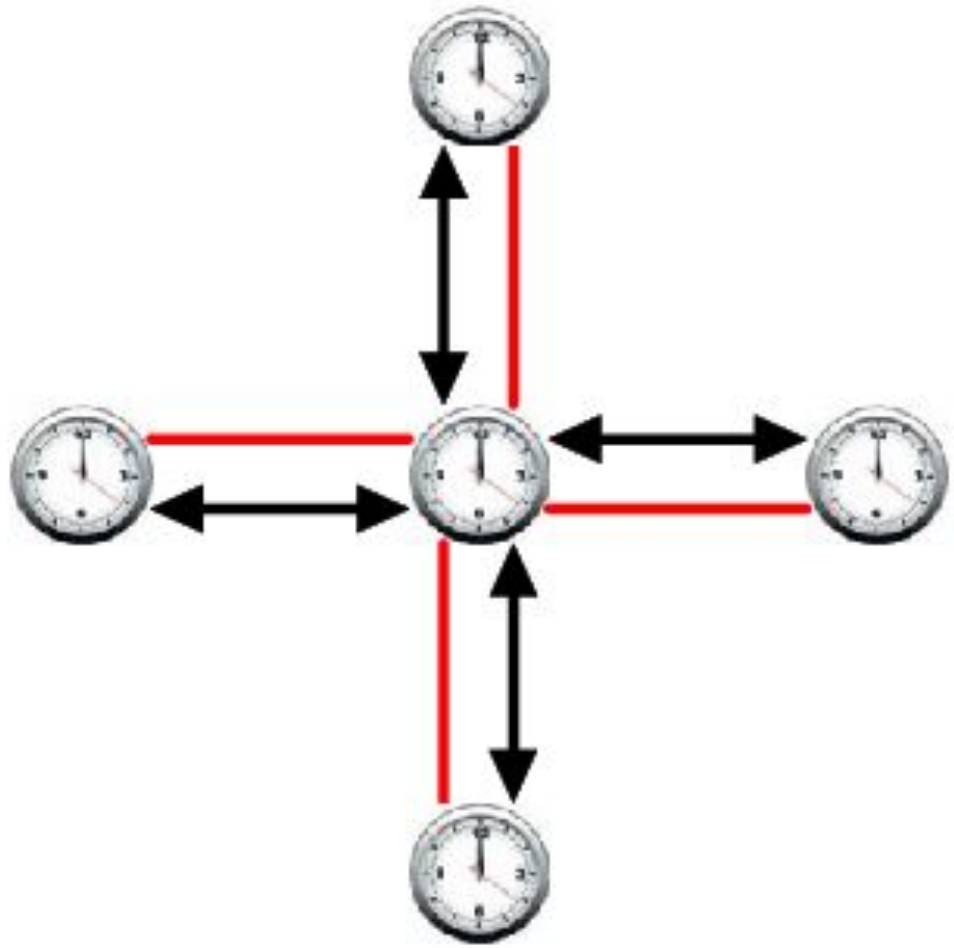
- For some distributed systems, it's necessary to get more coordination between components.
- The classic example is banking: we want to ensure that if I send you $100, both our banks agree that the transaction took place.
- Such coordination turns out to be surprisingly difficult if we are not careful.

# Clocks

- **In a distributed system** there are as many clocks as there are systems.
- The clocks are coordinated to keep them somewhat consistent but no one clock has the exact time.
- **Global Clock : A clock that feeds the entire device**.
- **Local Clock: Logical local time is used by the process to mark its own events,**

# Clock Synchronization

- Clock synchronization is **a method of synchronizing clock values of any two nodes in a distributed system with the use of external reference clock or internal clock value of the node**.



# Logical Clock

- **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.
- T(A)<T(B)
- Event A occurs before event B so timestamp of event B is always greater than A

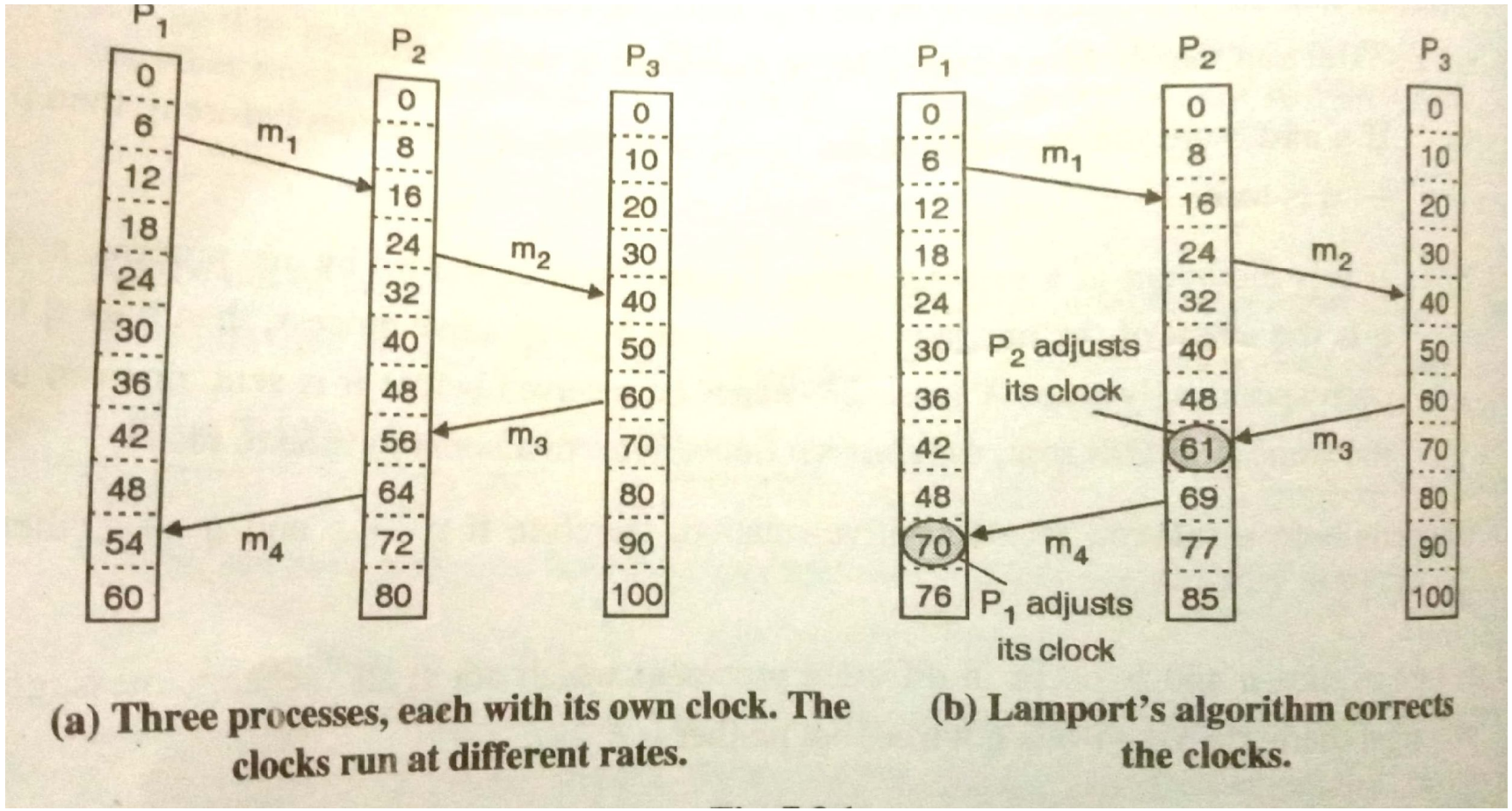
- Suppose, we have more than 10 PCs in a distributed system and every PC is doing it’s own work but then how we make them work together. There comes a solution to this i.e. LOGICAL CLOCK.

# Lamport's Logical Clock

- **Lamport’s Logical Clock** was created by Leslie Lamport.
- It is a procedure to determine the order of events occurring.
- It provides a basis for the more advanced Vector Clock Algorithm.
- Due to the absence of a Global Clock in a Distributed Operating System Lamport Logical Clock is needed.

- To synchronize logical clocks, Lamport defined a relation called happens-before.
- The expression a ->b is read as ‘a happens before b’ and means that all processes agree that event a occurs then event b occurs.
- From the definition of the happened-before relation, the clock condition mentioned above is satisfied if the following conditions hold:

- i. If a and b are two events within the same process and a occurs before b then a->b is true
- Ii. If a is message sent by process and b is the receipt of that message by process then a->b is also true.
- A message cannot be received before it is sent , or even at the same time it is sent, since it takes finite, nonzero amount of time to arrive.




(a) Three processes, each with its own clock. The clocks run at different rates.

(b) Lamport's algorithm corrects the clocks.

# Note :

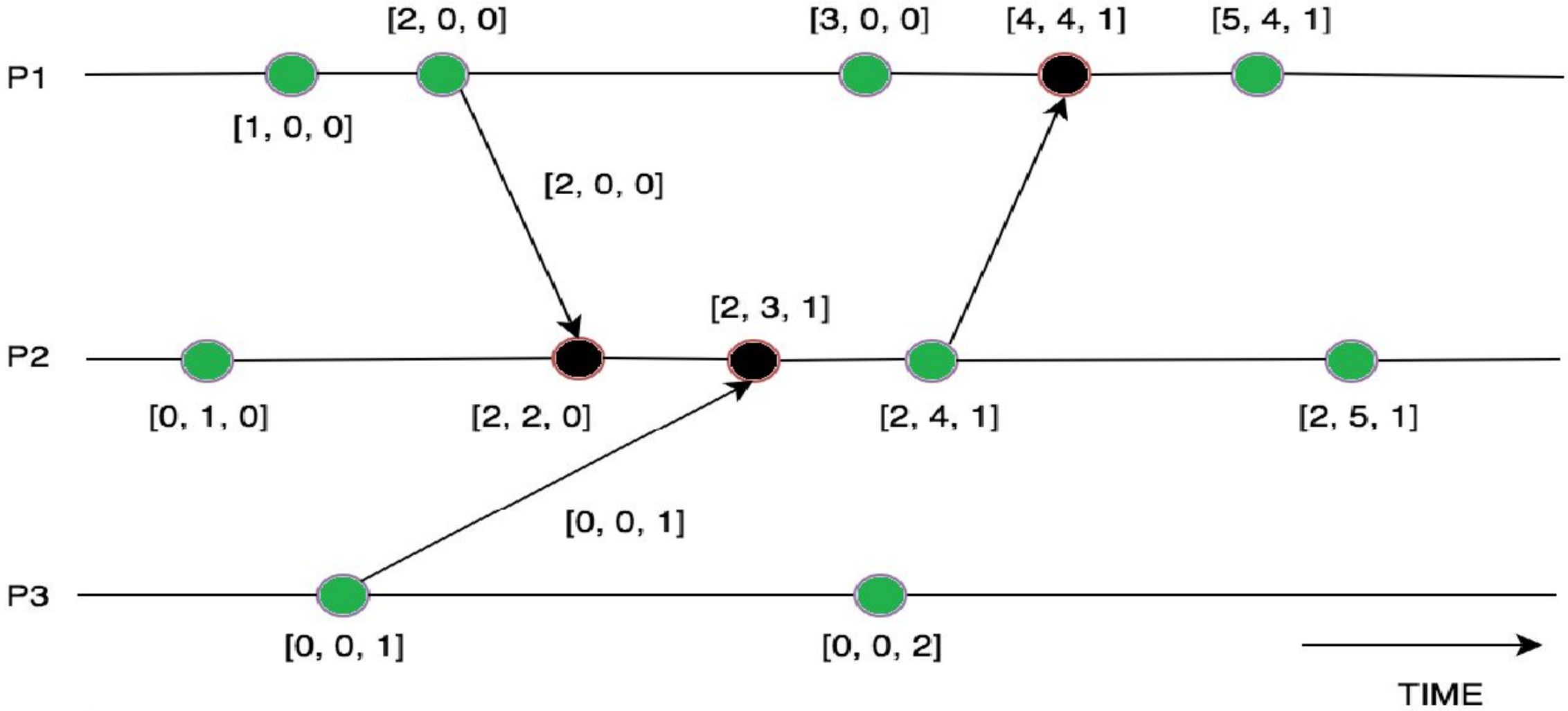
- A causal relationship exists **when one variable in a data set has a direct influence on another variable**.

# Vector Clock

- Vector Clocks are **used in a distributed systems to determine whether pairs of events are causally related**.
- Using Vector Clocks, timestamps are generated for each event in the system, and their causal relationship is determined by comparing those timestamps.
- This algorithm helps us label every process with vector (list ) with an integer for each local clock of every process within the system.
- So for N given processes, there will be vector/ array of size N.

- **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,




= EVENT EXECUTING RULE 1

= EVENT EXECUTING RULE 2

# Physical Clock

- 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.
- Coordinated Universal Time or UTC is the primary time standard by which the world regulates clocks and time.

- What is a time server used for?
- A time server is a server computer **that reads the actual time from a reference clock and distributes this information to its clients using a computer network**.

# i. Cristian’s Algorithm

- **Cristian’s Algorithm** is a clock synchronization algorithm is used to synchronize time with a time server by client processes.

- This algorithm works well with low-latency networks where Round Trip Time is short as compared to accuracy .

- *Round Trip Time refers to the time duration between the start of a Request and the end of the corresponding Response.*

# Algorithm

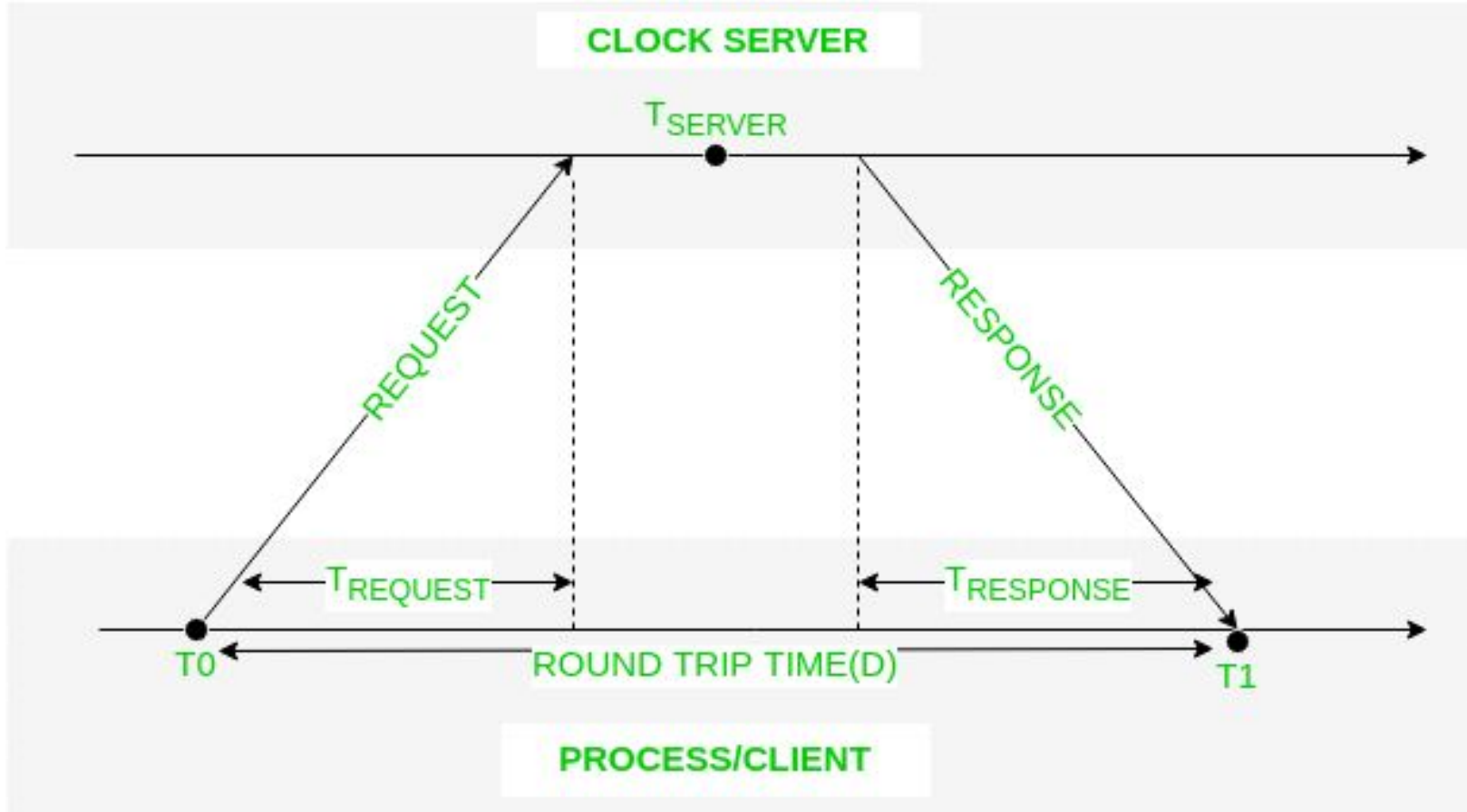
1) The process on the client machine sends the request for fetching clock time(time at the server) to the Clock Server at time .

2) The Clock Server listens to the request made by the client process and returns the response in form of clock server time.

3) The client process fetches the response from the Clock Server at time and calculates the synchronized client clock time using the formula given below.

$$T_{CLIENT} = T_{SERVER} + (T_1 - T_0)/2$$

where,
Tclient,  refers to the synchronized clock time,
Tserver, refers to the clock time returned by the server,
To, refers to the time at which request was sent by the client process,
T1, refers to the time at which response was received by the client process


CLOCK SERVER
T SERVER
REQUEST
RESPONSE
T REQUEST
T RESPONSE
T0
ROUND TRIP TIME(D)
T1
PROCESS/CLIENT

# •ii.The Berkeley’s Algorithm

- Berkeley’s Algorithm is a clock synchronization technique used in distributed systems.
- The algorithm assumes that each machine node in the network either doesn’t have an accurate time source or doesn’t possess a UTC ( Coordinated Universal Time) server.

- It then takes a fault tolerant average of clock values of all the computers.
- The calculated average is the current time to which all clocks should be readjusted.
- The time server readjusts its own clock to this value and instead of sending the current time to other computers it sends the amount of time each computer needs for readjustment.
- This can be positive or negative value and is calculated based on the knowledge the time server has about the propagation of message.

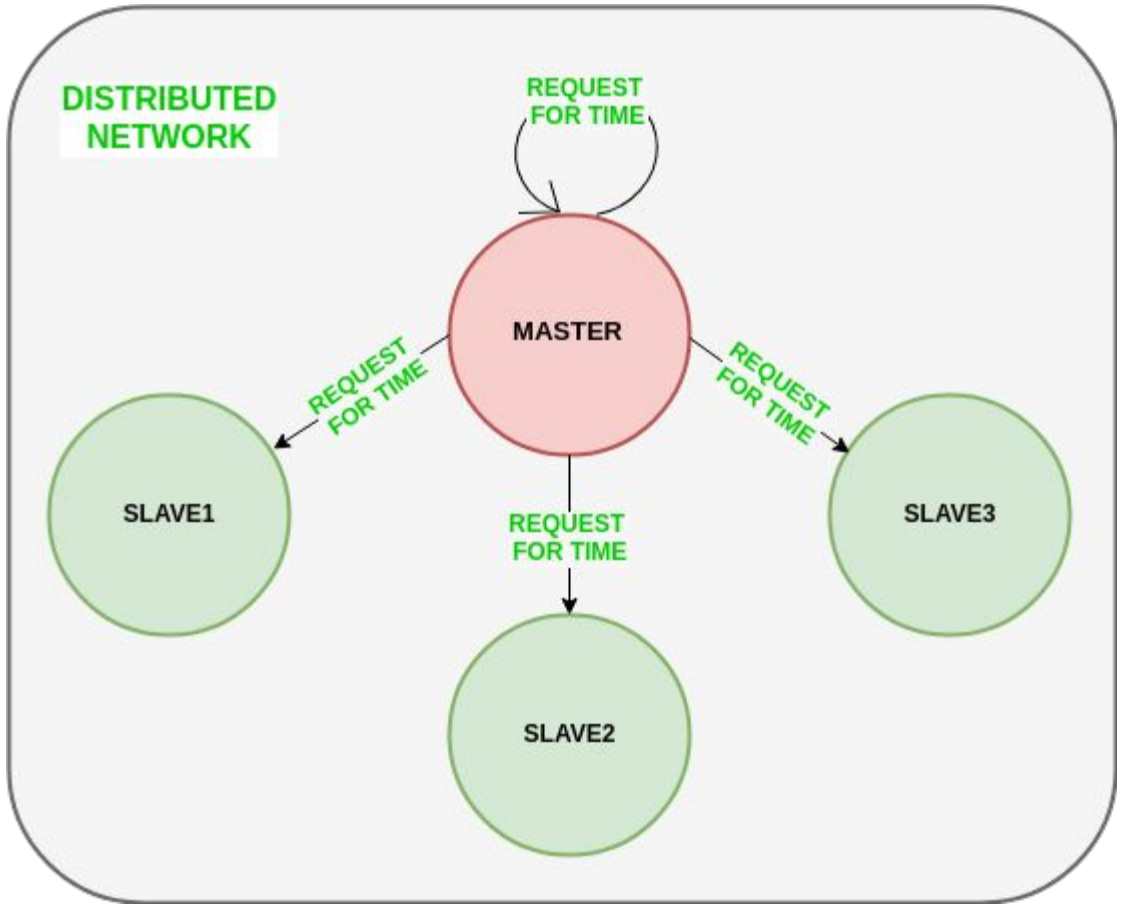
# Algorithm

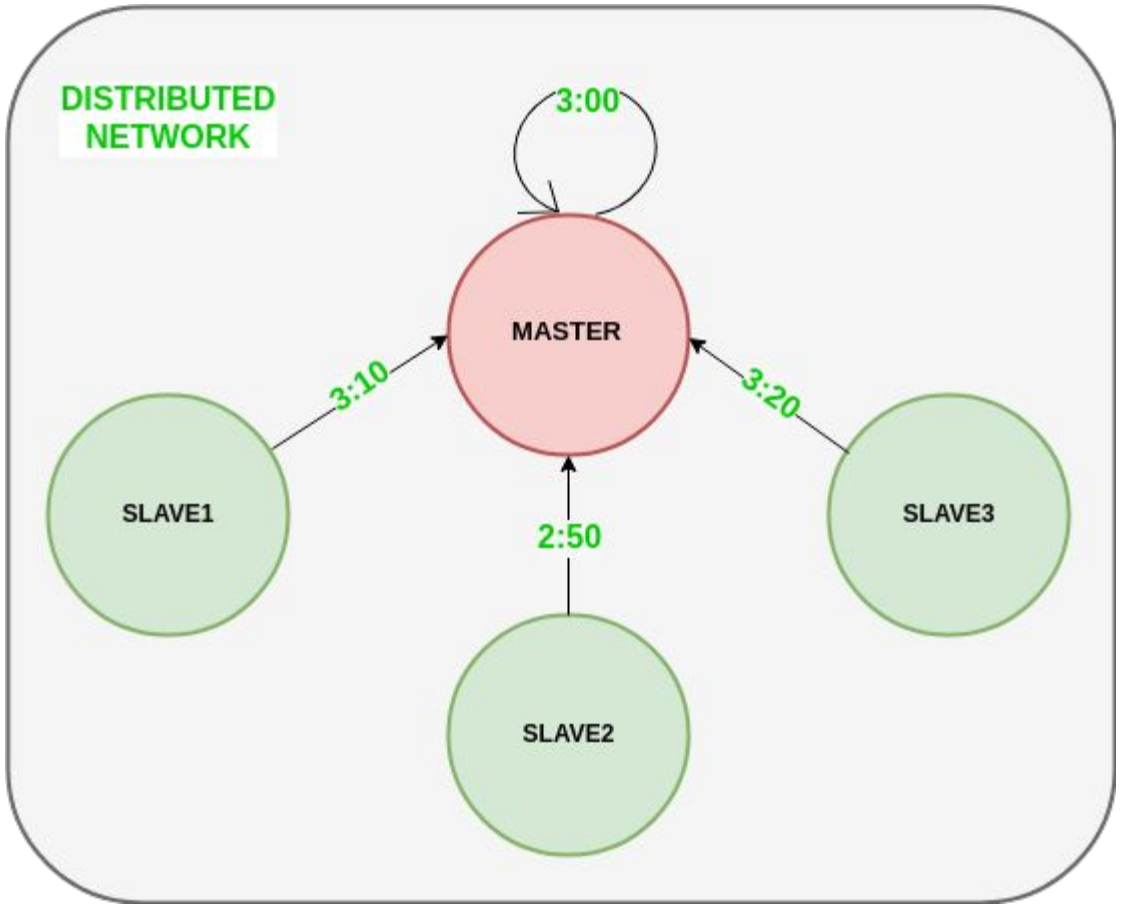
1) An individual node is chosen as the master node from a pool node in the network.

This node is the main node in the network which acts as a master and the rest of the nodes act as slaves.

The master node is chosen using an election process/leader election algorithm.

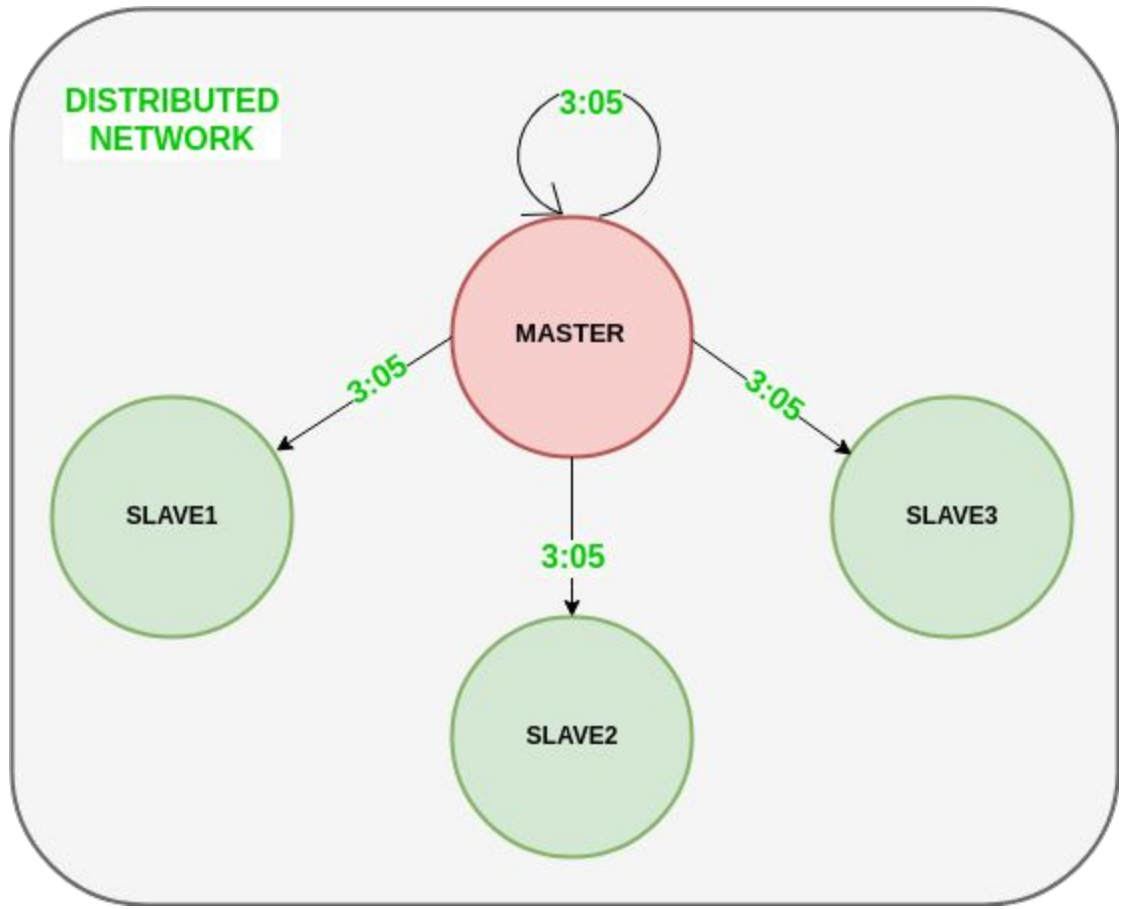
2) Master node periodically pings slaves nodes and fetches clock time at them.


DISTRIBUTED
NETWORK
REQUEST
FOR TIME
MASTER
REQUEST
FOR TIME
REQUEST
FOR TIME
REQUEST
FOR TIME
SLAVE1
SLAVE3
SLAVE2


DISTRIBUTED
NETWORK
3:00
MASTER
3:10
3:20
2:50
SLAVE1
SLAVE3
SLAVE2

3) Master node calculates the average time difference between all the clock times received and the clock time given by the master’s system clock itself.

This average time difference is added to the current time at the master’s system clock and broadcasted over the network.


DISTRIBUTED
NETWORK
3:05
MASTER
3:05
3:05
3:05
SLAVE1
SLAVE2
SLAVE3

# 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

- **Mutual exclusion** is a concurrency control property which is introduced to prevent race conditions.
- It is the requirement that a process can not enter its critical section while another concurrent process is currently present or executing in its critical section i.e only one process is allowed to execute the critical section at any given instance of time.

# Requirements of Mutual exclusion Algorithm:

- No Deadlock:
  Two or more process should not endlessly wait for any message that will never arrive.
- No Starvation:
  Every process who wants to execute critical section should get an opportunity to execute it in finite time.
- Fairness:
  Each process should get a fair chance to execute critical section.
- Fault Tolerance:
  In case of failure, it should be able to recognize it by itself in order to continue functioning without any disruption.

# Mutual Exclusion Algorithm

- Lamport's Algorithm
- Ricart and Agrawala's Algorithm
- Maekawas Algorithm
- Token Based Algorithm
- Vector Algorithm

# Lamport's Algorithm

Uses logical timestamps to ensure mutual exclusion.

- Each process in the system maintains a logical clock, and before entering the critical section.

# Steps

- When a process wants to enter the critical section, it sends a **request message** with a timestamp to all other processes.
- Upon receiving the request, each process replies when it is not in its critical section or when its timestamp is earlier than the one in the request.
- A process enters the critical section when it has received a reply from every other process.
- After exiting the critical section, the process sends a **release message** to inform other processes that it’s no longer using the resource.

- **Advantages**: Simple to implement and guarantees mutual exclusion.
- **Disadvantages**: Requires message passing and careful handling of timestamps.

**Ricart and Agrawala's Algorithm (1981)**:

This is a more optimized version of Lamport’s algorithm.

**Methods:**

- Similar to Lamport's, but instead of replying to a request when not in the critical section, a process sends a reply only when it has no intention of entering the critical section.

- If two processes request the critical section simultaneously, their timestamps are used to decide who gets to enter.

**Advantages**: Less message passing because the requesting process waits for replies instead of sending messages when it can’t access the critical section.

**Disadvantages**: More complex than Lamport's algorithm.

**Maekawa's Algorithm (1985)**:

Uses voting to determine access to the critical section.

**Method:**

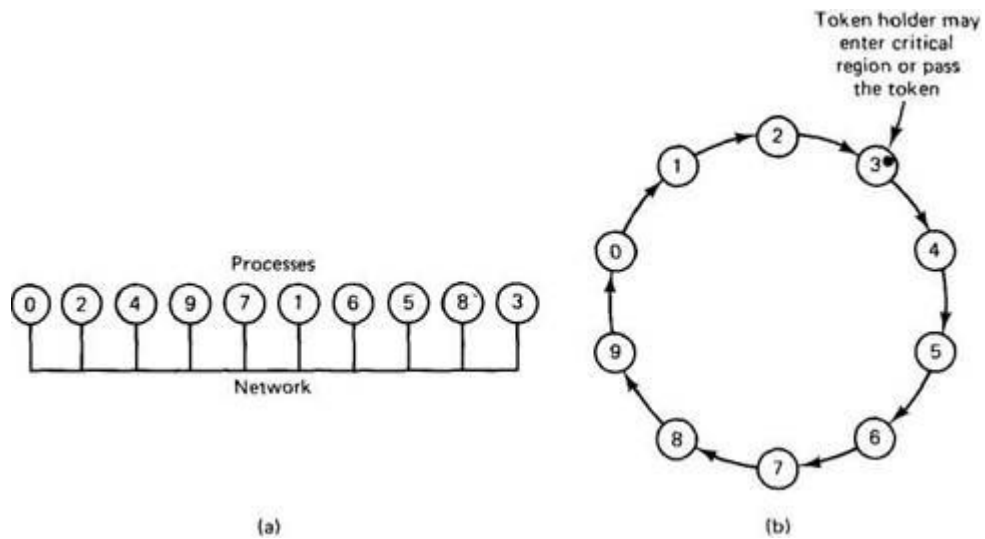
The basis of this algorithm is a quorum-like approach where any one process needs to seek permissions from number of other process.

The number of processes in a quorum is typically at least the ceiling of `N/2`, where `N` is the total number of processes in the system.

- **Advantages**: Reduces the total number of messages that need to be passed.
- **Disadvantages**: More complex to implement and analyze.

**Token-Based Algorithms (e.g., Suzuki-Kasami's Token Algorithm)**:

- A token is passed between processes in a logical ring or tree structure. A process can only enter the critical section if it holds the token.
- **Method**: One process in the system holds a unique token. The process with the token can access the critical section. When it finishes, it passes the token to the next process.
- **Advantages**: Simple and efficient in terms of message passing, as only the token needs to be passed around.
- **Disadvantages**: If the token is lost or delayed, the system could be stuck until recovery mechanisms are applied.


Processes
0
2
4
9
7
1
6
5
8
3
Network
(a)
Token holder may enter critical region or pass the token
0
1
2
3
4
5
6
7
8
9
(b)

# Vector Clocks

- Vector clocks are used for ordering events in a distributed system and can be used to determine the ordering of requests for mutual exclusion.
- **Method:** Each process maintains a vector of logical clocks. The vector is updated when a process sends or receives messages. The ordering of vector clocks helps determine the precedence of requests and can prevent conflicts when accessing the critical section.
- **Advantages**: Provides more detailed information about event ordering and causality in distributed systems.
- **Disadvantages**: Requires more memory and computation to manage the vector clocks.

# Election Algorithms:

- **Election algorithms choose a process from group of processors to act as a coordinator.**
- **If the coordinator process crashes due to some reasons, then a new coordinator is elected on other processor.**
- **Election algorithm basically determines where a new copy of coordinator should be restarted.**
- Election algorithm assumes that every active process in the system has a unique priority number.

- The process with highest priority will be chosen as a new coordinator.
- Hence, when a coordinator fails, this algorithm elects that active process which has highest priority number.
- Then this number is send to every active process in the distributed system.
- We have two election algorithms for two different configurations of distributed system.

- Bully Algorithm
- Ring Algorithm

# BULLY ALGORITHM

This algorithm applies to system where every process can send a message to every other process in the system.

**Algorithm** – Suppose process P sends a message to the coordinator.

1. If the coordinator does not respond to it within a time interval T, then it is assumed that coordinator has failed.
2. Now process P sends an election messages to every process with high priority number.

1. It waits for responses, if no one responds for time interval T then process P elects itself as a coordinator.
2. Then it sends a message to all lower priority number processes that it is elected as their new coordinator.
3. However, if an answer is received within time T from any other process Q,
    - (I) Process P again waits for time interval T’ to receive another message from Q that it has been elected as coordinator.
    - (II) If Q doesn’t responds within time interval T’ then it is assumed to have failed and algorithm is restarted.

# RING ALGORITHM

This algorithm applies to systems organized as a ring.

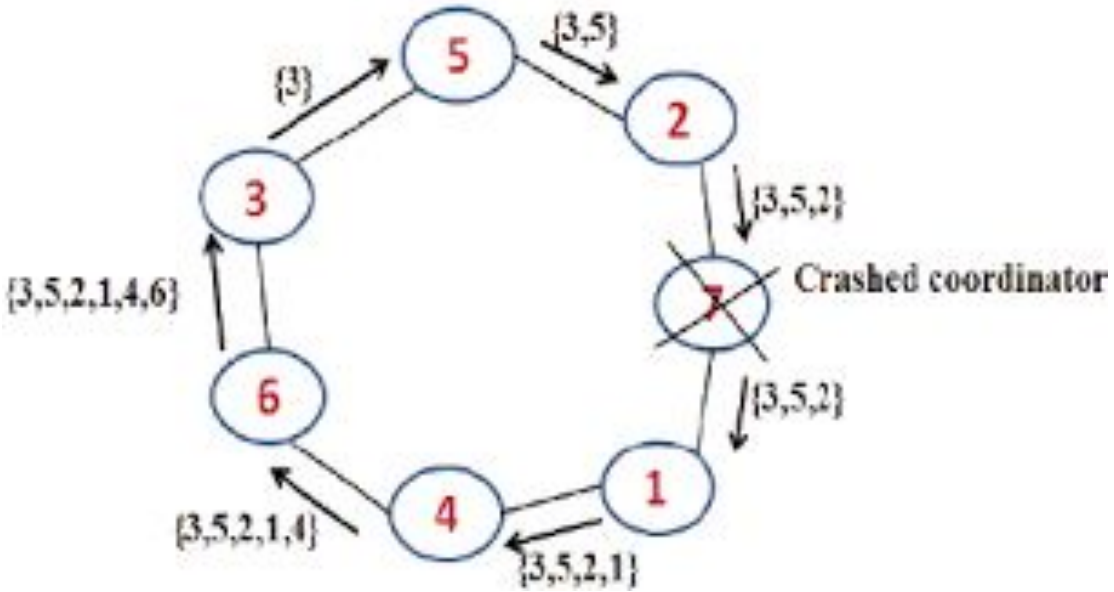
In this algorithm we assume that the link between the process are unidirectional and every process can message to the process on its right only.

An active list is maintained to store the nodes id.

**Algorithm –**

When a node( which is the initiator)  notices that coordinator is dead :

1. Initiator builds and sends election message to nodes
2. At every steps nodes keep on adding its own id at the end of list.
3. The process stops when the initiator receives the message it initiated .
4. After this the node with highest ID is declared as the coordinator.
5. Initiator announces the coordinator by sending message to nodes .


5
{3,5}
{3}
2
3
{3,5,2}
{3,5,2,1,4,6}
7
Crashed coordinator
{3,5,2}
6
1
4
{3,5,2,1,4}
{3,5,2,1}

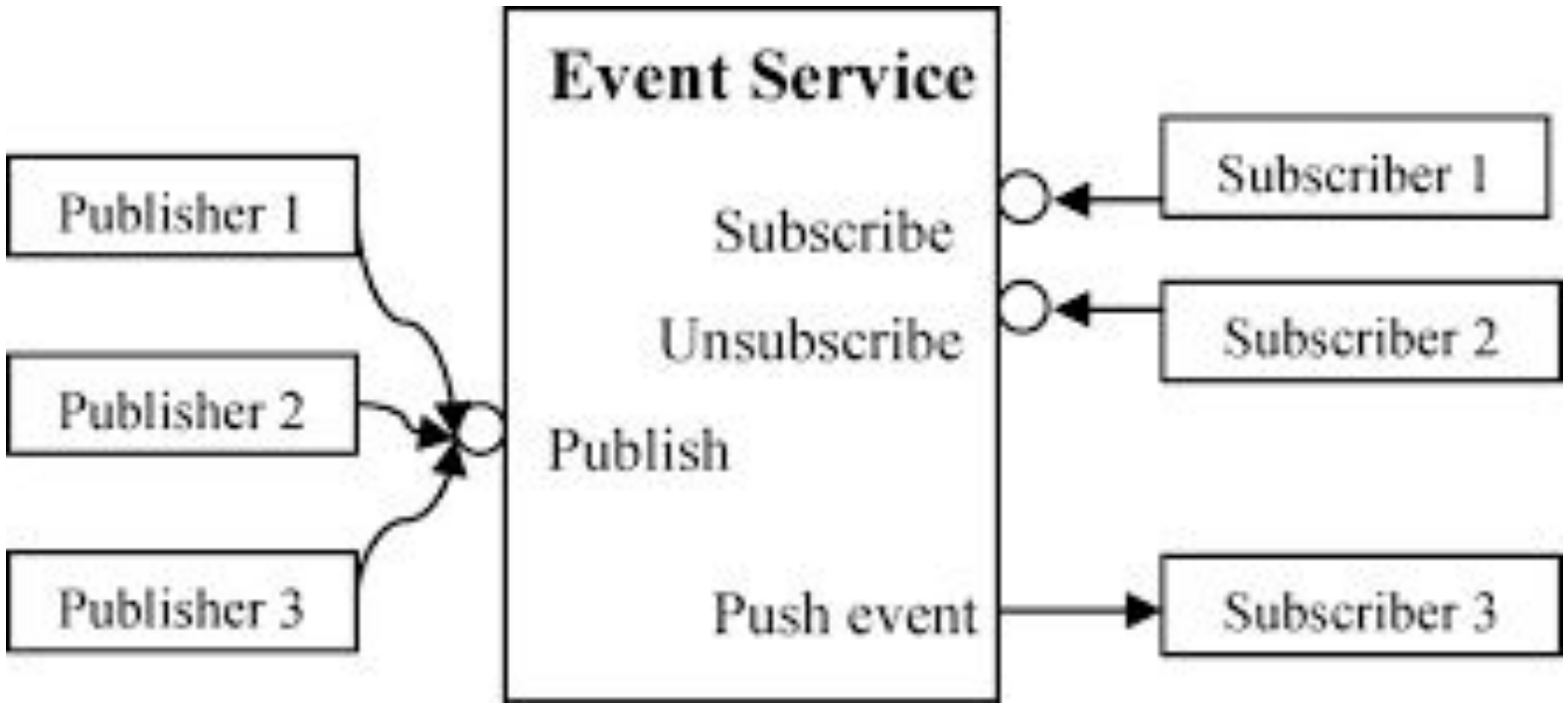
# Location System

- GPS
- The Global Positioning System, originally Navstar GPS, is a satellite-based radionavigation system owned by the United States government and operated by the United States Space Force.
- The **Global Positioning System** (**GPS**) tells you where you are on Earth.

# Distributed Event Matching

Event Matching , or more precisely , notification filtering is at the heart of **publish-subscribe systems**.

- A process specifies through a subscription S in which events it is interested .
- When a process publishes a notification N on the occurrence of an event , the system needs to see if S matches N.
- In the case of a match the system should send the notification N, possibly including the data associated with the event that took place , to the subscribers.
- Publisher push the message and subscriber uses it.


Event Service
Subscribe
Unsubscribe
Publish
Push event
Publisher 1
Publisher 2
Publisher 3
Subscriber 1
Subscriber 2
Subscriber 3

# Gossip based Coordination

- Gossip is a peer-to-peer communication protocol in which nodes periodically exchange state information about themselves and about other nodes they know about.

- Gossip protocol based algorithms are one of the most robust and scalable algorithms o and can piggy back any information on top of the gossip messages.

- Gossip is about letting each node send the **latest information** it happens to have to some set of other nodes eventually spreading that information throughout the network.
- It’s a way for nodes to build a global map from limited local interactions.

- And if there's a partition, nodes in sub-partitions will still happily gossip with each other.

- So a request that hits one side of a partition can get totally different answers than if it hits another side.