# Lab2
# Design & Evalution

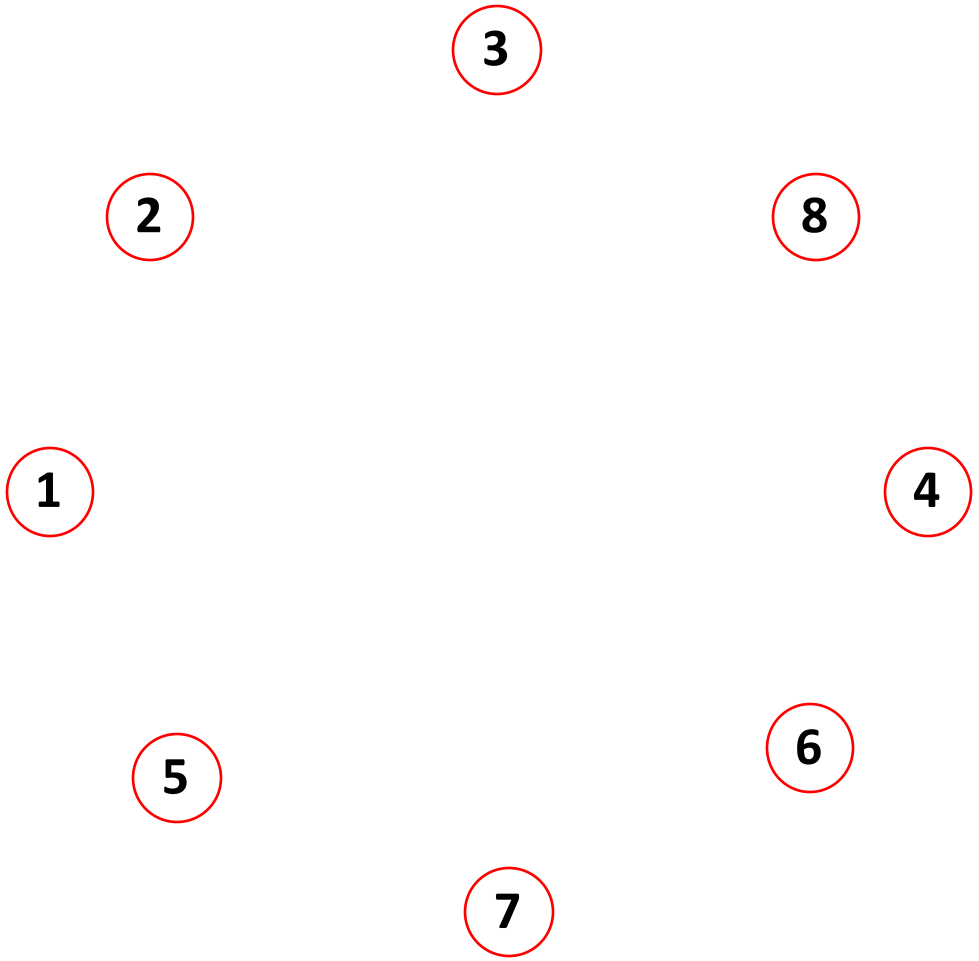
# Leader Election

# Motivation

- ‘Bully Algorithm’ with some modification
- Always targeted last server ( i.e., server 8) as coordinator.
- Sequentially, sent election request to the server with highest server ID, If it fails then sent to second highest server ID and so on.

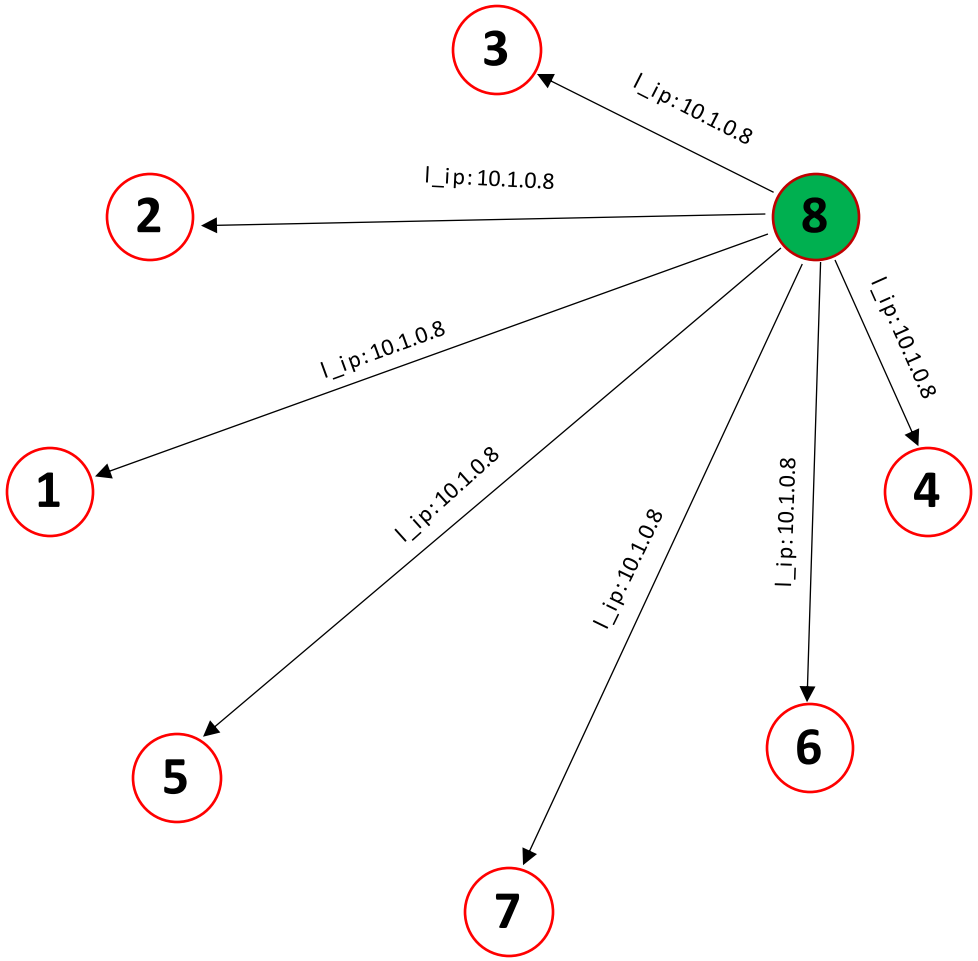
- Leader always handle "add on board" requests
  - Generate new ID with current time stamp
  - Propagate to other servers
- Other servers just forward "add on board" requests to Leader

# Design – Leader Election



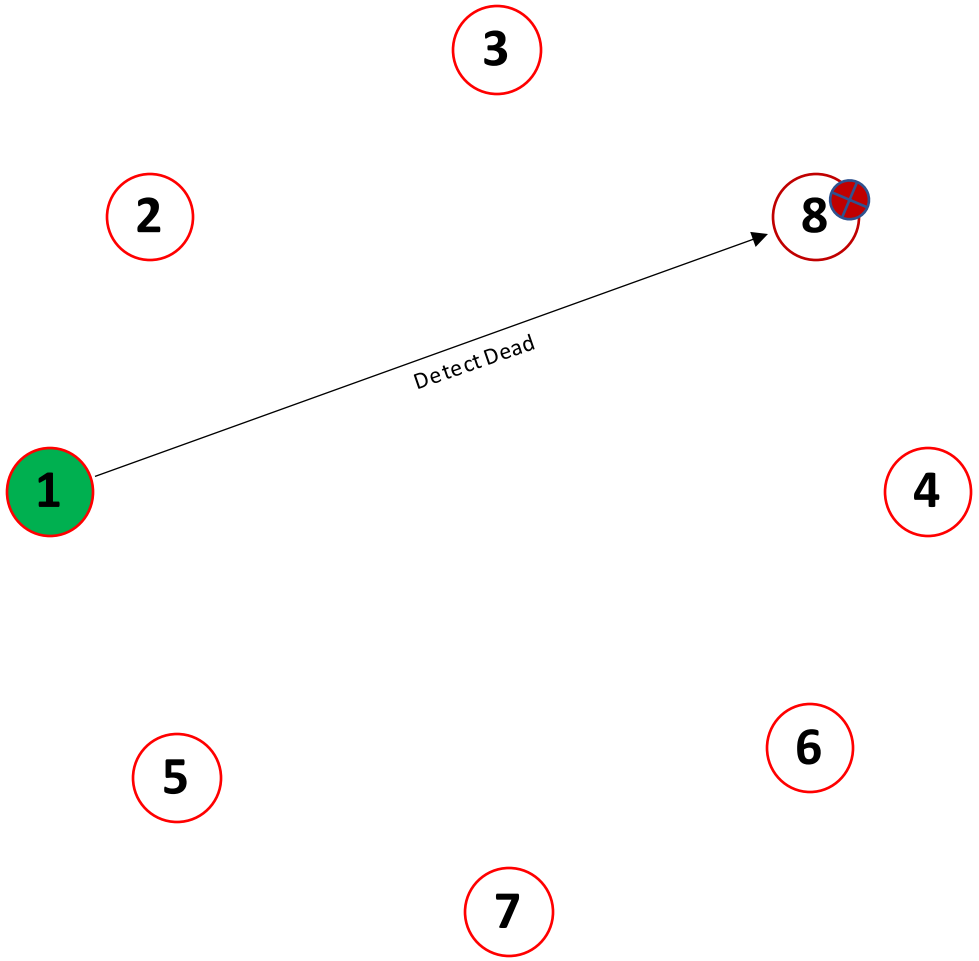

- No Leader

# Design - LE (Servers Started)



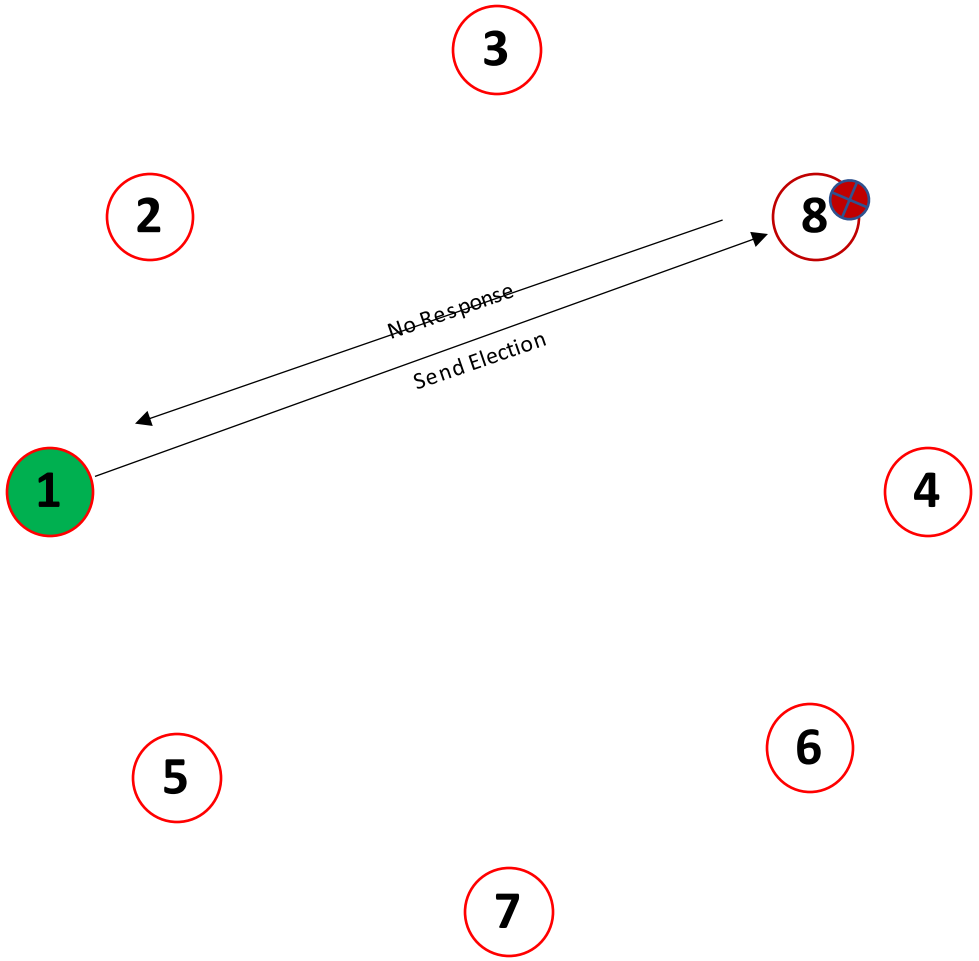

- Servers Started
- Last host (I.e. 8) starts the election
- Check in the host list for highest server_id
- So server 8, Declare itself as "Coordinator"
- And notify other hosts with Leader IP

# Design - LE (Server 8 Dead)



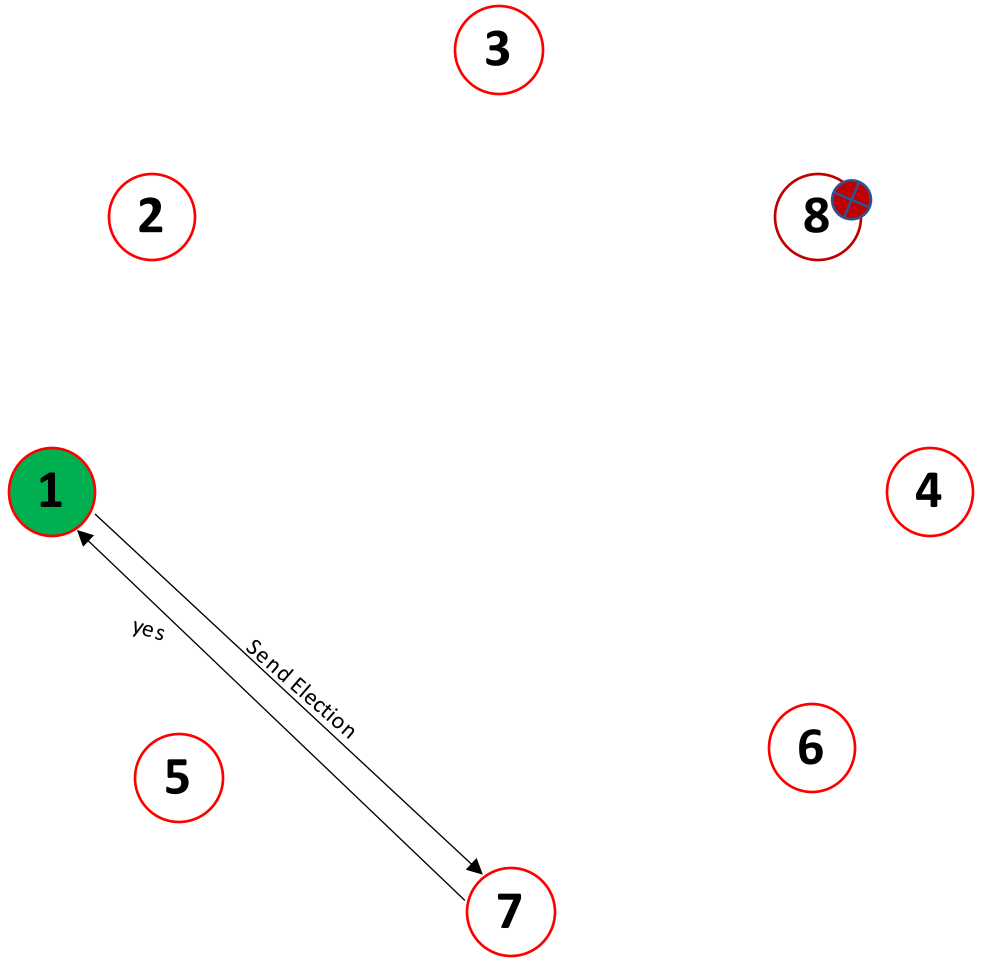

- Server 1 , Detect 8 is dead, While adding a post on board.
- Server 1 , Start Election

# Design - LE (Server 8 Dead)



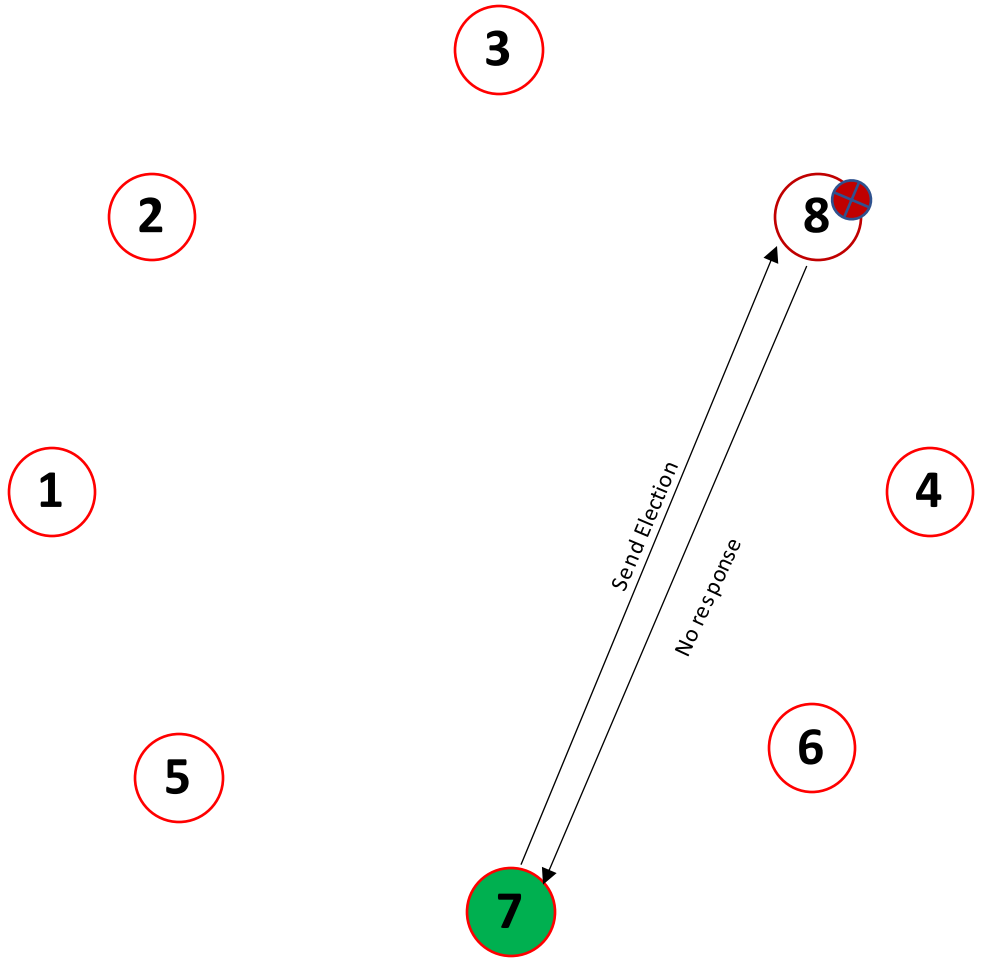

- Server 1 , Send election to 8
- Server 8, no response

# Design - LE (Server 8 Dead)



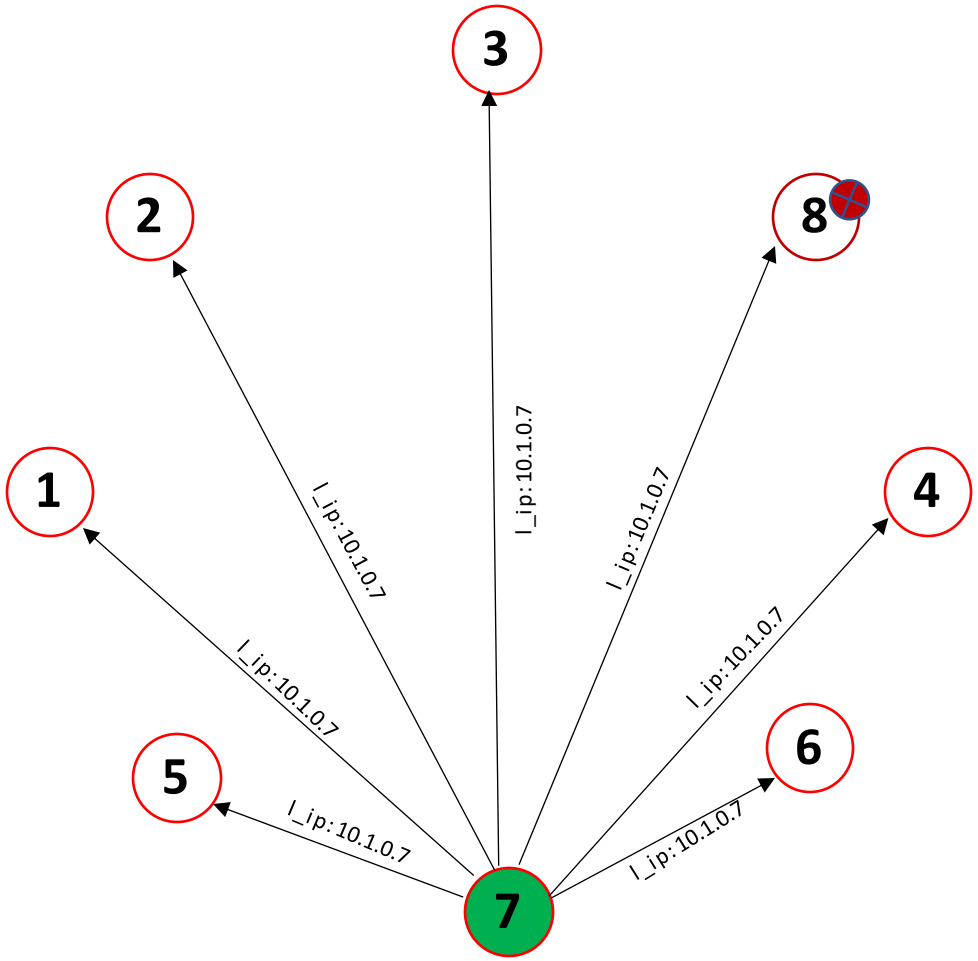

- Server 1 , Send Select to 7
- Server 7 response with "yes"

# Design - LE (Server 8 Dead)



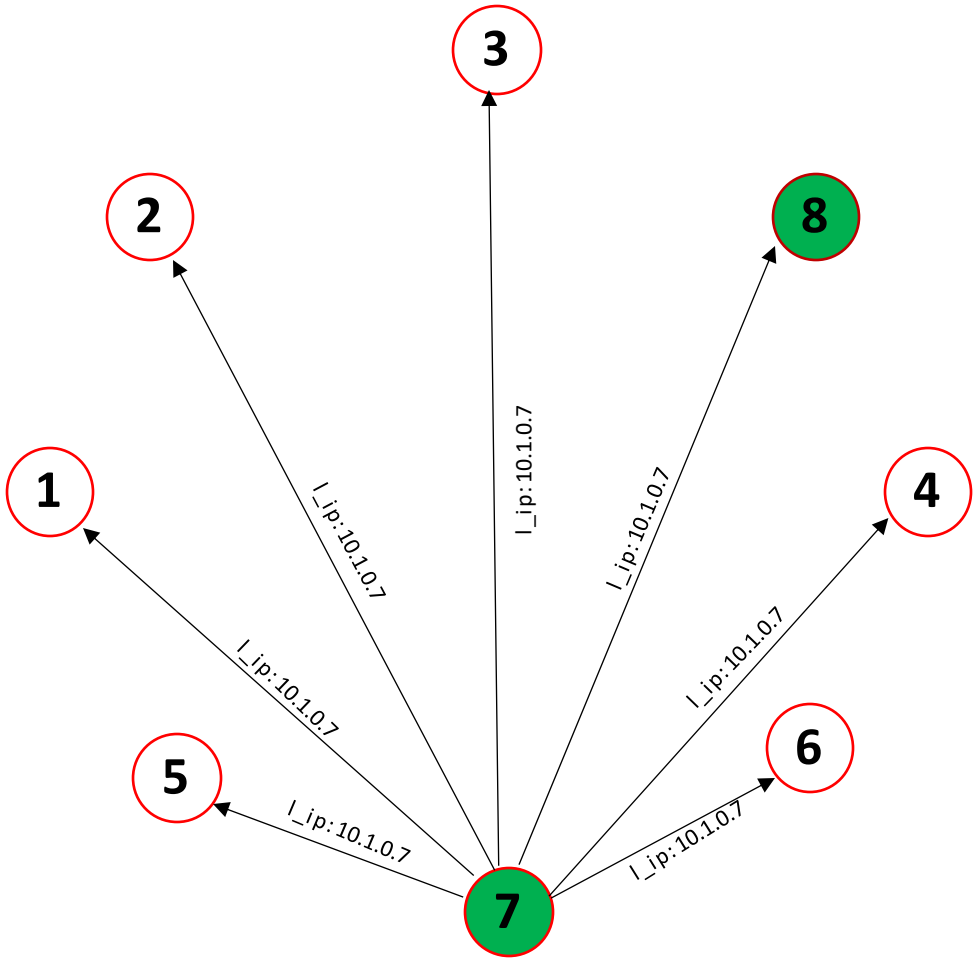

- Server 7 , Start Election
- Send Election to 8
- Server 8 does not respond

# Design - LE (Server 8 Dead)



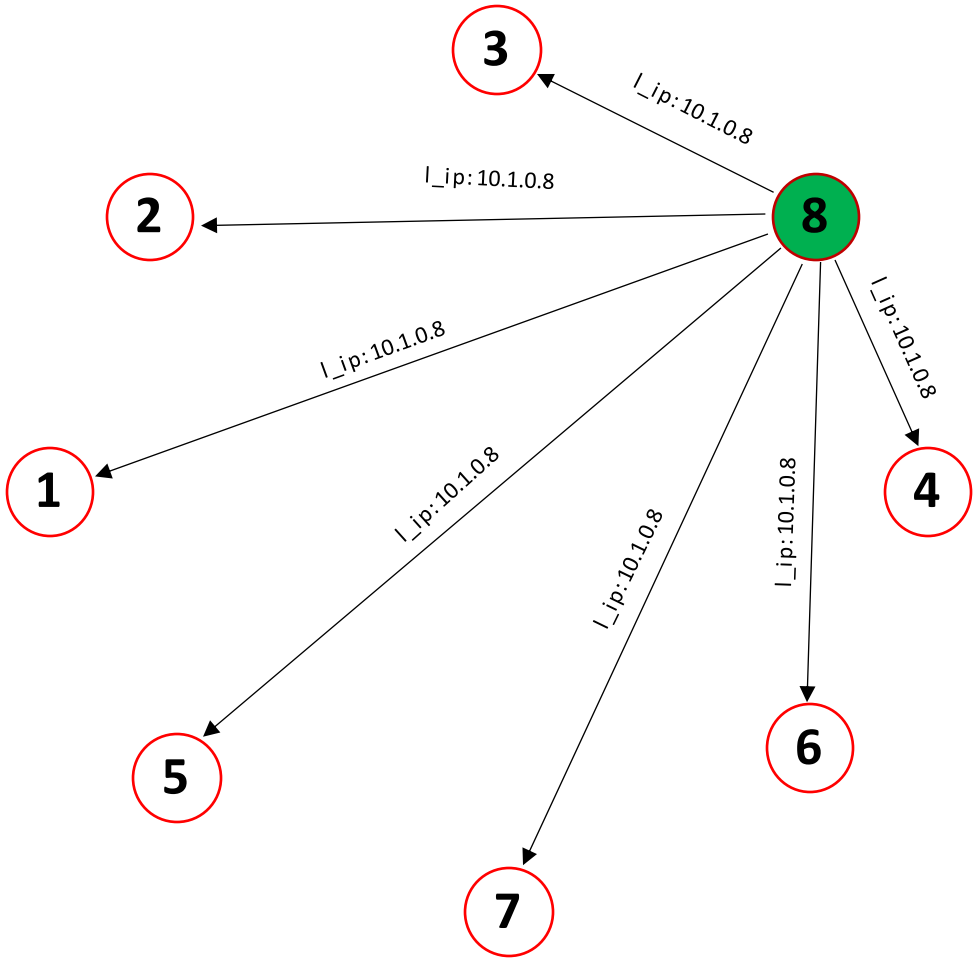

- Server 7 , declare itself as "Coordinator"
- Notify other hosts with its IP that leader changed

# Design - LE (Server 8 Rebooted)




- Server 8 Rebooted ,
- Start the election after 3 secs
- Check it's ID and declare itself as "Coordinator"
- Notify other servers with its IP

# Design - LE (Servers 8 Rebooted)




- Server 7 got new Leader IP
- Reset its leader_ip and stop data processing thread

# Evalution

- Increased consitency
- But coordinator need handle all the reqeusts, single point of failer can make inconsistant data
- Other servers don't know anything about the status (busy or dead) of Leader
- Rebooted leader always need to fetch old data from other servers to maintain consistancy
- If volumn of data in board is high syncrhonization will take huge amount of time

# Data Propagation & Synchronization

- Leader server has its own data processor thread
- If a message found by "add_on_board" API
- Message is stored inside a temporary "queue" of "DataProcessor" processor thread
- "DataProcessor" thread always check the queue and if any new data found in queue it generate a new key and propagate to all servers.
- Since the key is getting generated from a single server and propagate by the same server data is synchronized

# Global Data variables

- leader_ip
- board = dict() # global board

These two global variable we used to store the Leader IP and board content, because we need to access these data from every class of our project.

# API : add_on_board

- Receive a new entry
- If entry already have "id", it just store into board.
- If no "id", forward to "coordinator"

# API: election

- Start a election

# API: leader

- Get modified leader IP (i_ip) forwarded by elected leader
- If host is already a "Coordinator" it stop its "DataProcessor" thread and change its global leader_ip variable

# API: modify_delete

- Delete or modify the existing content

# API: sync

- Newly elected leader call this API to fetch all stored board data

# Changed Files

- Server.py : All API
- data_processor.py : (Infinity loop Thread) Only works on Leader Server. Process the add on board data and propagate to all other server.
- leader_election.py : (Thread) Implemented leader election algorithm
- server_data.py : Stored the leader_ip and board informaitons
- Test.py : Implemented test cases
- Blackboard.tpl : added server title

# Reboot server command

- python3 server/server.py --id 8 --servers 10.1.0.1,10.1.0.2,10.1.0.3,10.1.0.4,10.1.0.5,10.1.0.6,10.1.0.7,10.1.0.8

# Task 4: Cost LE ( #number of server n = 8)

Cost will include :

1. Number of election request
2. Number of notify request
3. Syncronization cost = 1 (becase we, fetch all the data in single request)

- Best case (all servers running):
  - Number of election request = 1
  - Number of notify request ( for notifying leader IP) = n – 1
  - So, Cost = n + 1 (Syncronization cost )

- Worst Case (Just server 1 running):
  - Number of election request = n - 1
  - Number of notify request ( for notifying leader IP) = n – 1
  - So, Cost = 2n –2 + 1 (Syncronization cost )

# Task 4: Our LE properties

- Intiation
  - Any process who unable to communicate with current Leader will start the election
  - If multiple server cannot communicate with current Leader then, multiple simultaneous election will start
- Termination
  - All election process will elect the last server_id at the end of election
- Naming
  - All servers know about server_ip and server_id of every other process. But initially any server have no idea about server crashing

**So, Our LE algorithm fullfill all properties**

# Task 4: Cost of Adding message to the board

- Cost for non-leader server to leader = 1
- Cost to propagate = n-1
- Final Cost = n

# Task 4 : Pros and Cons

- Pros:
  1. Easy to Implement
  2. Easy to detect dead server
- Cons
  1. All requests need to handle by one server, so time consuming
  2. If leader dead during execution, Data may loss
  3. Olnly good for less amount of request
  4. Bottleneck possible
  5. No multiple coordinator possible
  6. Other servers cannot detect that leader too much busy now