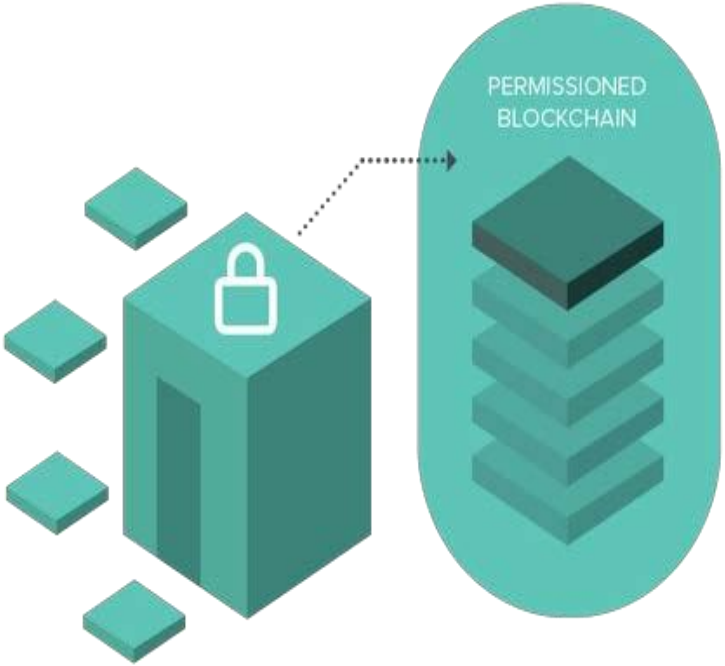


Image Source: https://nem.io/enterprise/

Permissioned Blockchain - III
Consensus Algorithms

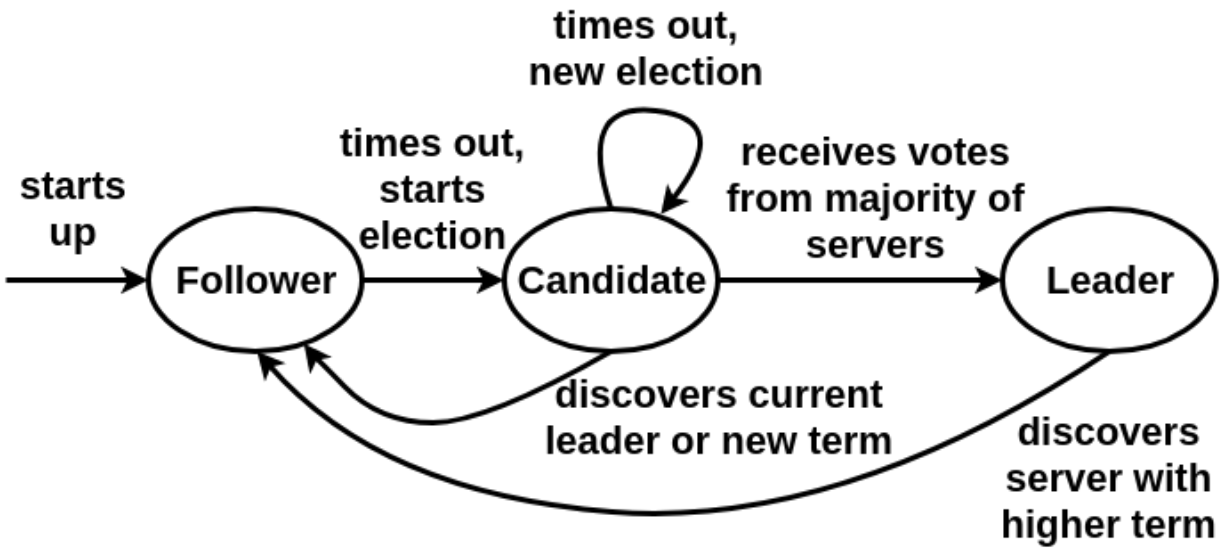
RAFT Consensus

- If you have leader in the system then achieving consensus becomes easier because in that case you can avoid the situation where multiple proposers proposing something all together then the protocol becomes complicated because acceptors have to accept one proposal with higher proposal number used as a **tie breaking mechanism**.
- In a distributed environment with synchronous assumptions it is possible to design consensus algorithms.
- In paxos, you did not have any kind of leader but in RAFT, one leader is there who will take decision based on certain mechanisms.

RAFT (Reliable, Replicated, Redundant, And Fault-Tolerant) Consensus

- Paxos is very complicated in theory but very simple in concept
- For its theoretical proof, you need to look for details at different levels of optimization. (Majority decision)
- Designed as **an alternative to Paxos** by Diego Ongaro and John in 2014
- Companies like **MongoDB, HashiCorp**, etc. are using RAFT.
- A generic way to distribute a state machine among a set of servers
  - Ensures that every server agrees upon same series of state transitions
- **Basic idea -**
  - The nodes collectively selects a *leader* ***(No leader in Paxos)***; others become *followers*
  - The leader is responsible for state transition log replication across the followers

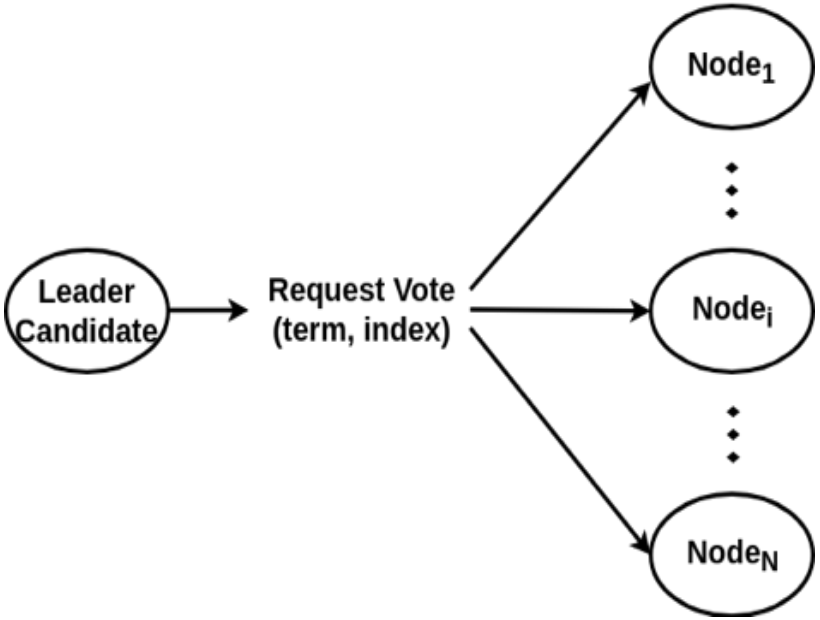
RAFT



- When system startup then it has followers, who looks for leader if no then starts election
- (re)electing a leader
- committing multiple values to the transaction log
- dealing with replicas failing

Source: Diego Ongaro and John Ousterhout, In Search of an Understandable Consensus Algorithm, Stanford University, February 22, 2014.

First part of RAFT-Electing the Leader: Voting Request

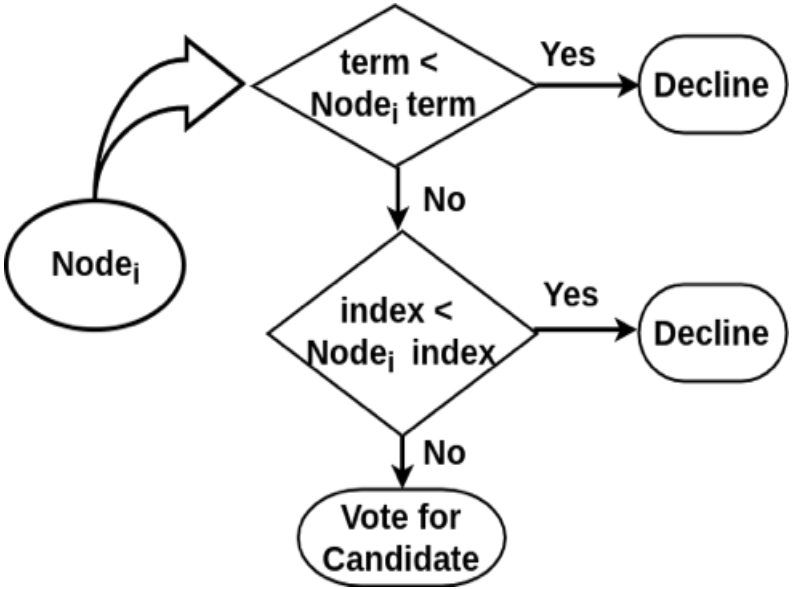


- **term:** This parameters decides in which round you are, last calculated # known to candidate + 1

  e.g) Earlier round was 20 when T=20 then you have to elect new leader then the new round is 21 then T=21

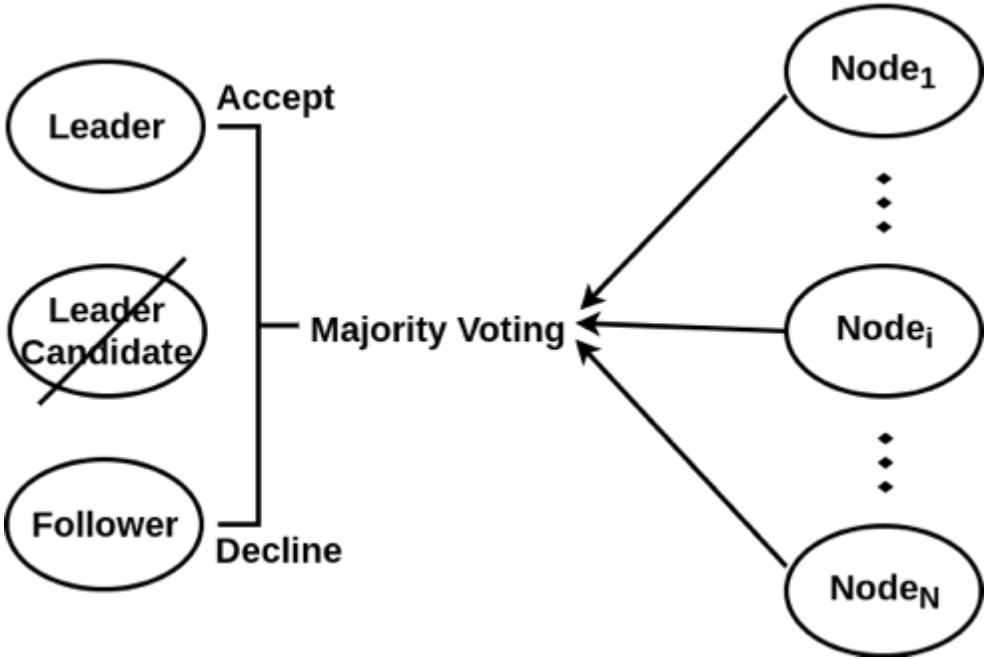
- **Index:** means committed transaction available to the candidate.
- Like Paxos, RAFT is also executed in rounds and each round you need to take some decision.
- Then, the request vote is sent to all nodes available in the n/w

Electing the leader: Follower Node's Decision Making



- Once the nodes received the *"request message"* then their task is to elect the leader.
- Each node compares received term and index with corresponding current known values
- E.g) if candidate (c) send the proposal message P(20, i) to node (n), now the n looks what term it has, say if n is at term 21(current term) that means c is requesting for some earlier term then **so decline it**
- Otherwise, look for index value (i), if P (20, 100) and the node has current index value as 150, which indicates that *n* already committed transaction up to 150 so we should not allow that candidate because that Tx already committed then **decline the vote in favor of the candidate.**
- **Otherwise vote for the favor of candidate**

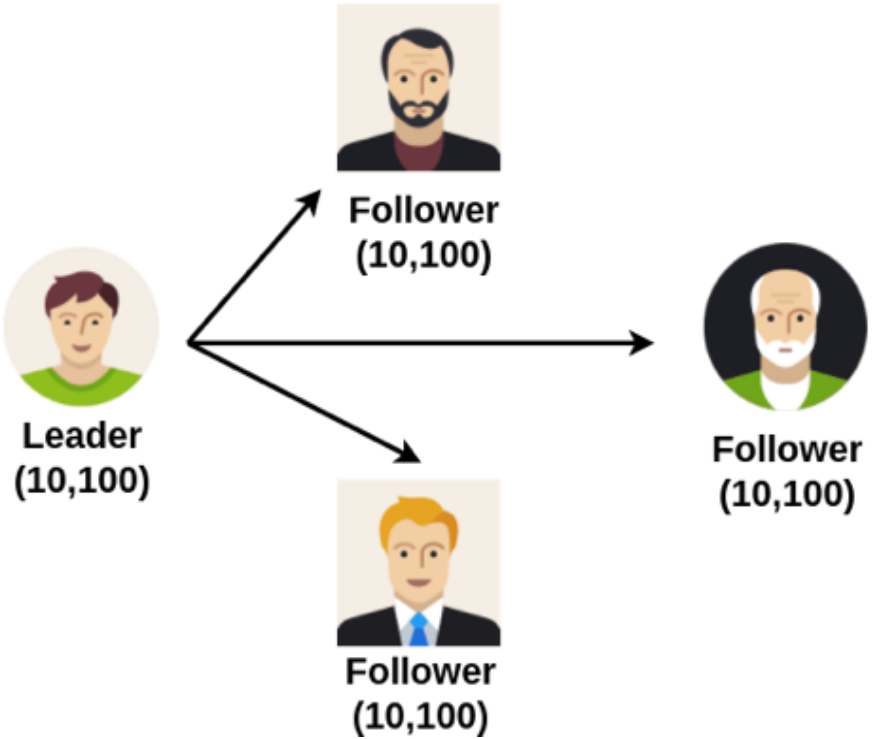
Electing the leader: Majority Voting



- Use of **Majority voting**
  - leader selection
  - Commit the corresponding log entry

  e.g) if any leader candidate received majority of the votes from the nodes then that particular candidate becomes the leader and other becomes the followers of that node

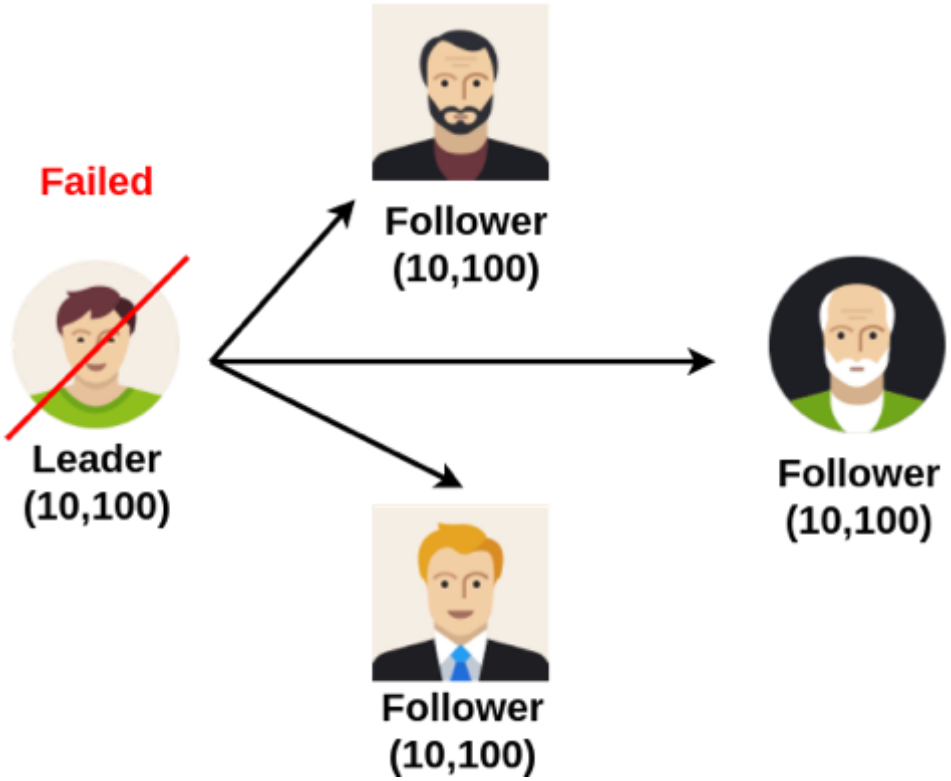
Example of Multiple Leader Candidates at the Current Leader Failure



Consider and example where a leader has three followers

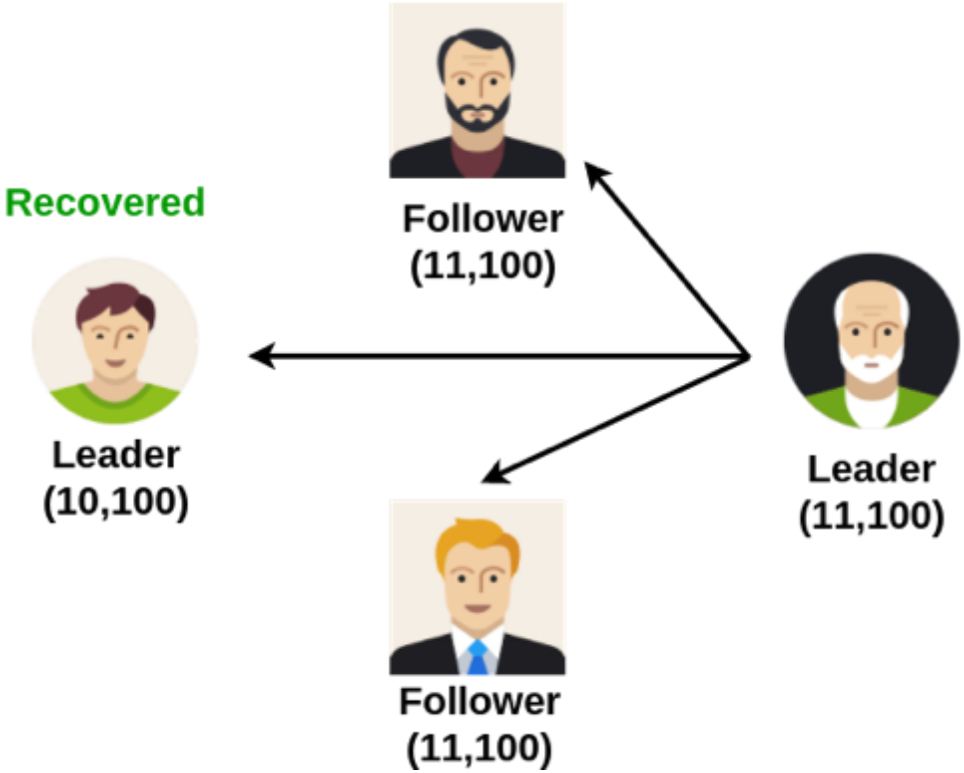
- **Current term:** 10
- **Commit index:** 100

Multiple Leader Candidates: Current Leader Failure



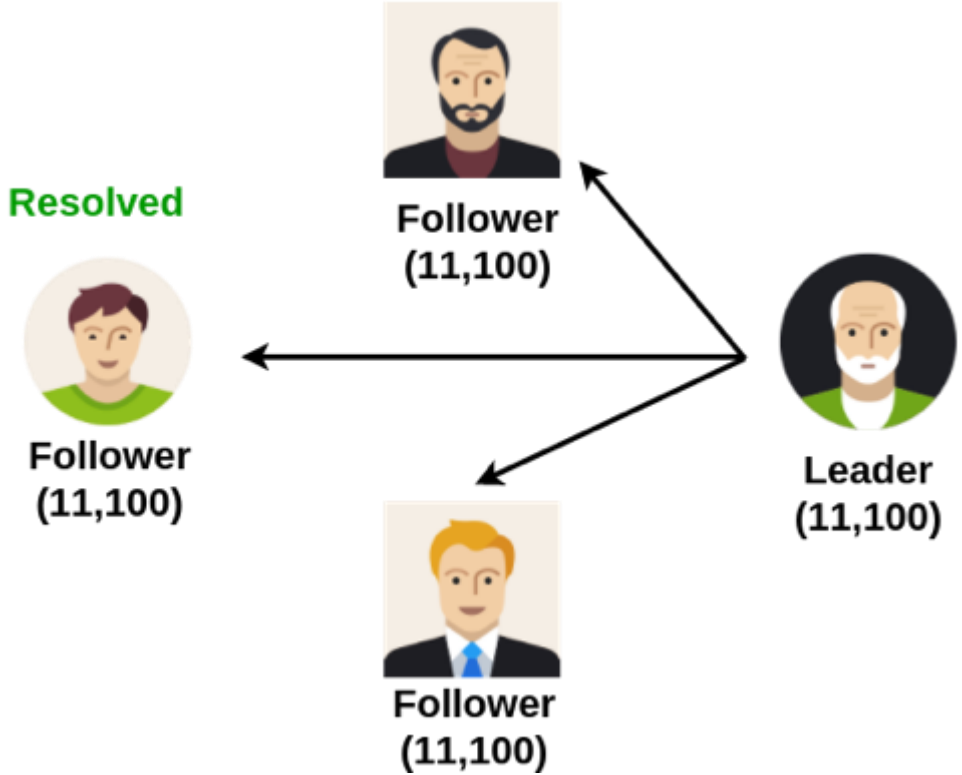
- Say the leader node has failed
- Once it happened then you need to elect new leader ???

Multiple Leader Candidates: Current Leader Failure



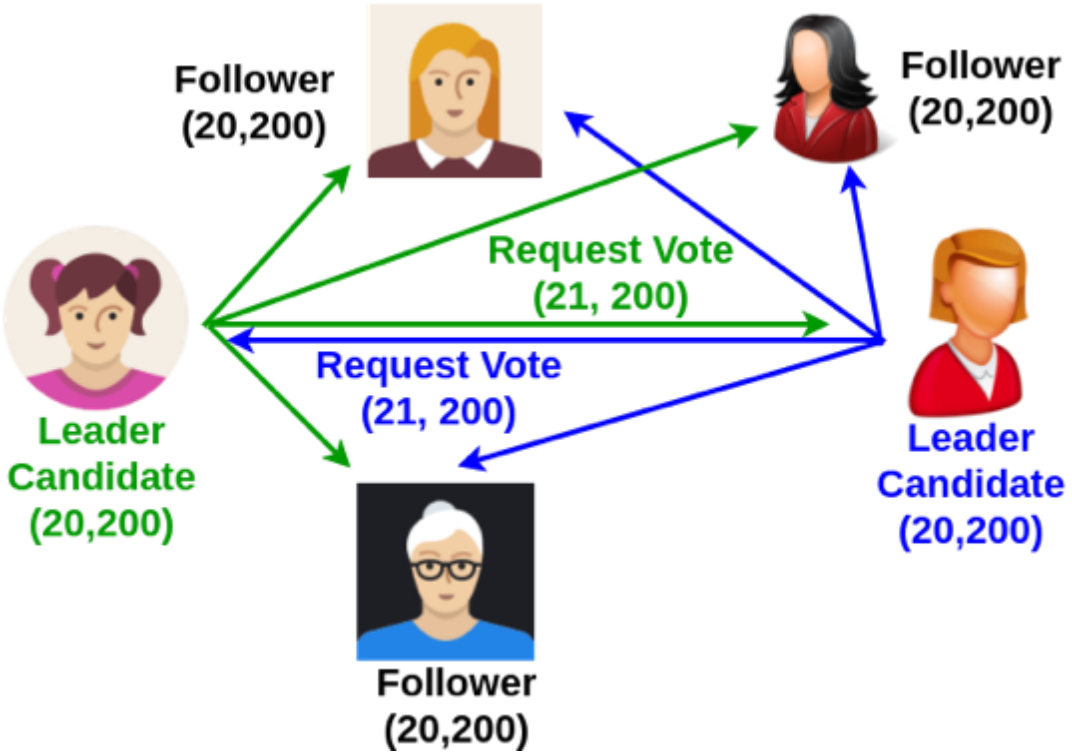
- New leader needs to be elected with term 11, means you need to move from round 10 to round 11
- But there may be chances that the **old leader gets recovered**
- **And once the old leader received the message from the new leader then ???**

Multiple Leader Candidates: Current Leader Failure



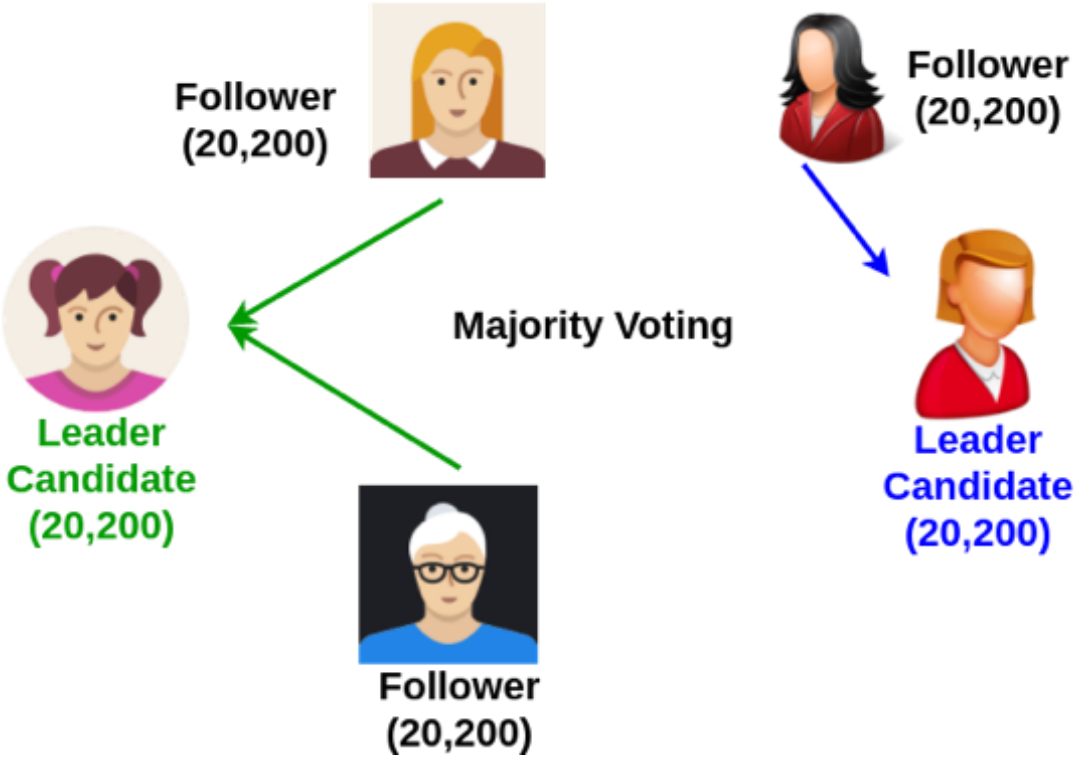
- Old leader receive heartbeat message from new leader with greater term
- Then, the old leader drops to follower state in round 11

Multiple Leader Candidates: Simultaneous Request Vote



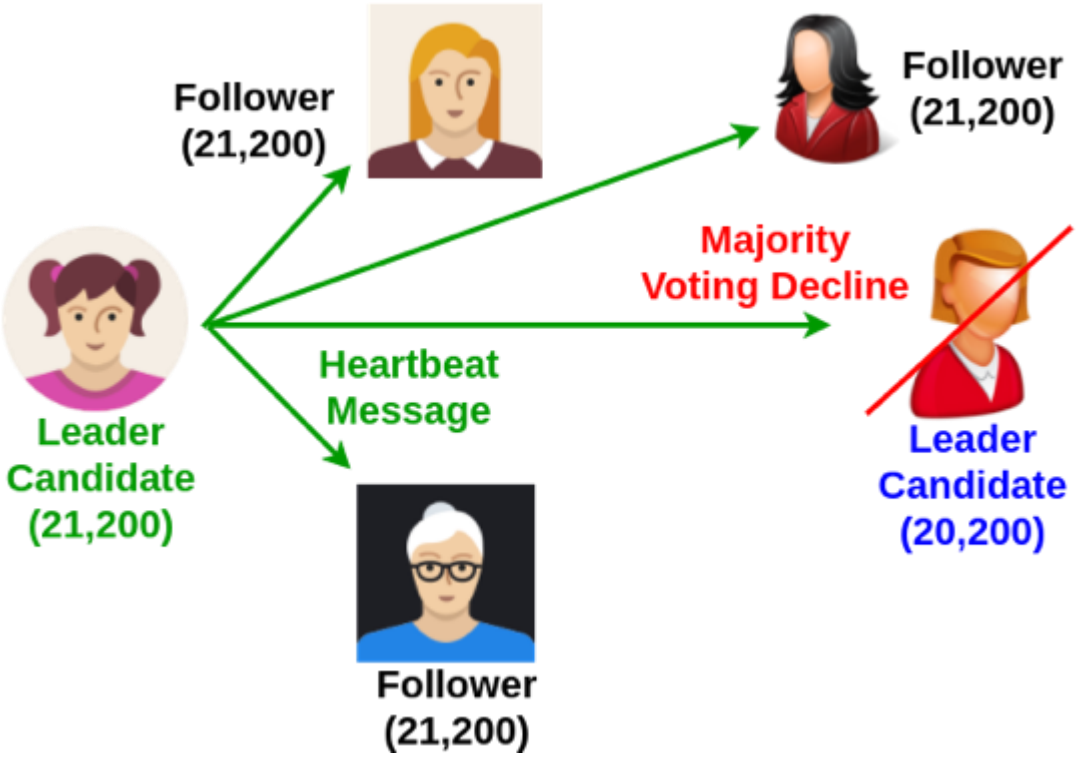
- Another case, when **two nodes send request vote message with term 21 at the same time**
- For example, if there are two leader candidates and the system is currently at (term) round 20
- Then ??? How to tackle it

Multiple Leader Candidates: Simultaneous Request Vote



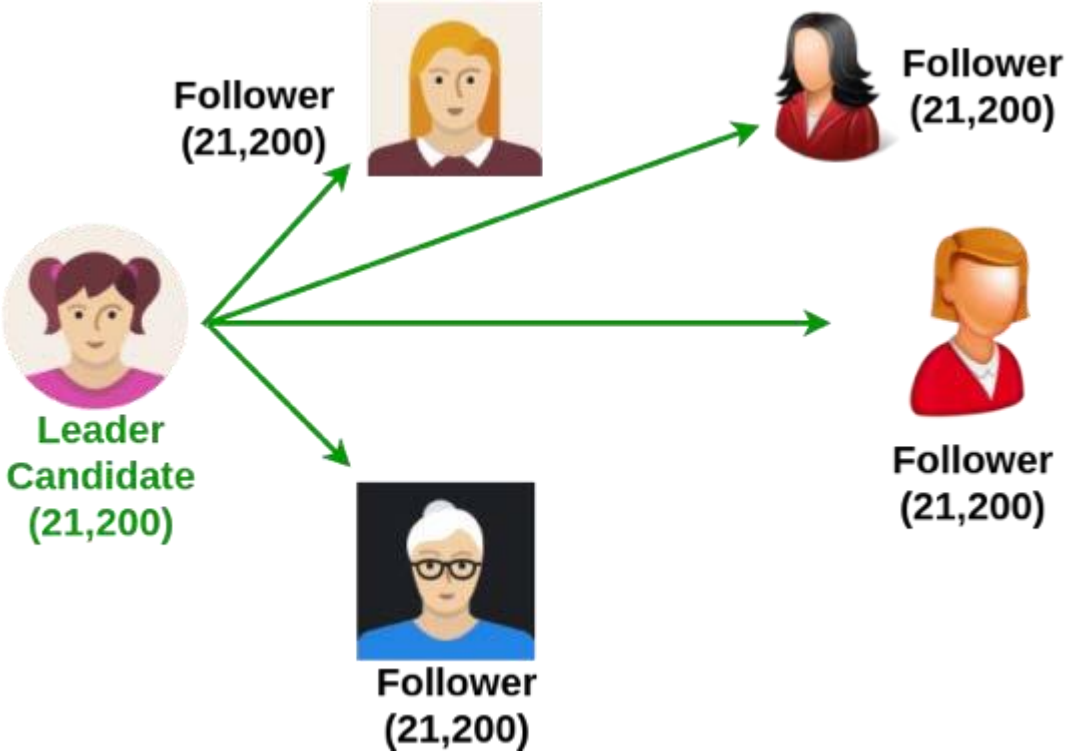
In this case, one of them gets majority voting

Multiple Leader Candidates: Simultaneous Request Vote



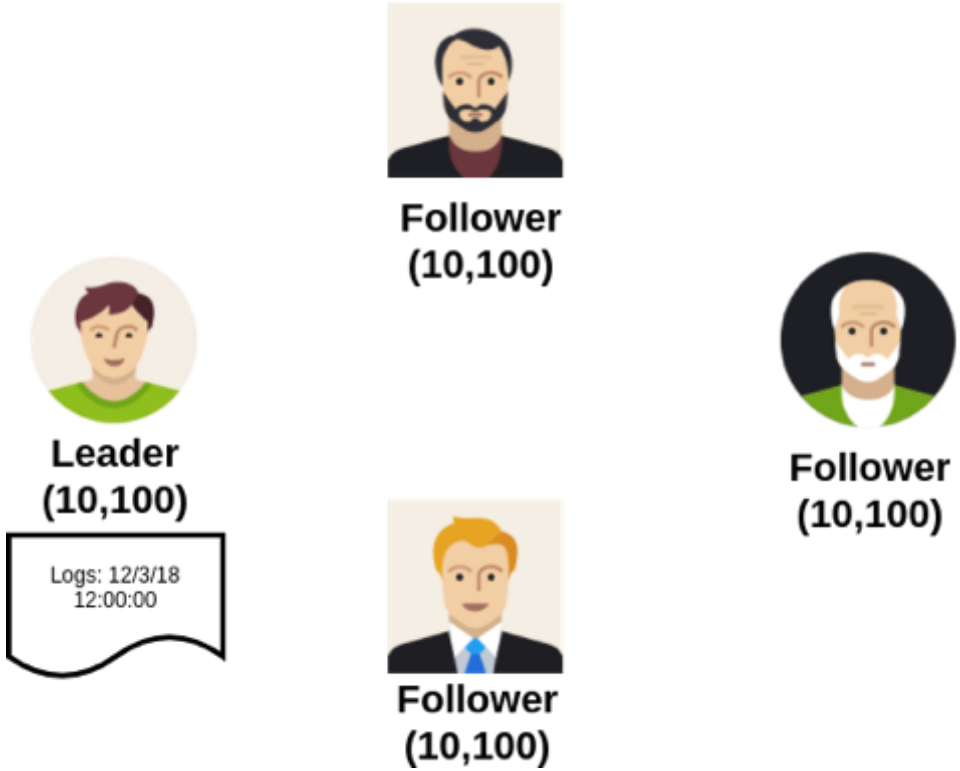
- Winner sends heartbeat message
- This message ensures that we have received the majority of the voting and took the final decision

Multiple Leader Candidates: Simultaneous Request Vote



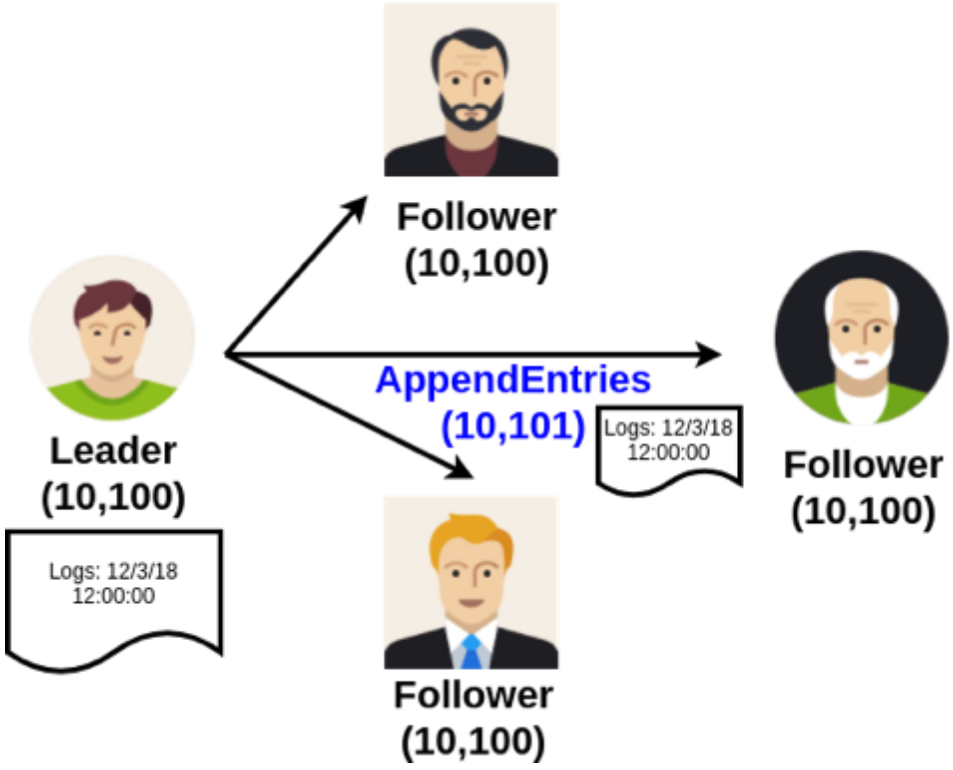
- Then, other leader candidate switches to follower state
- And term or round 21 gets started

Next part is how will you Commiting Entry Log



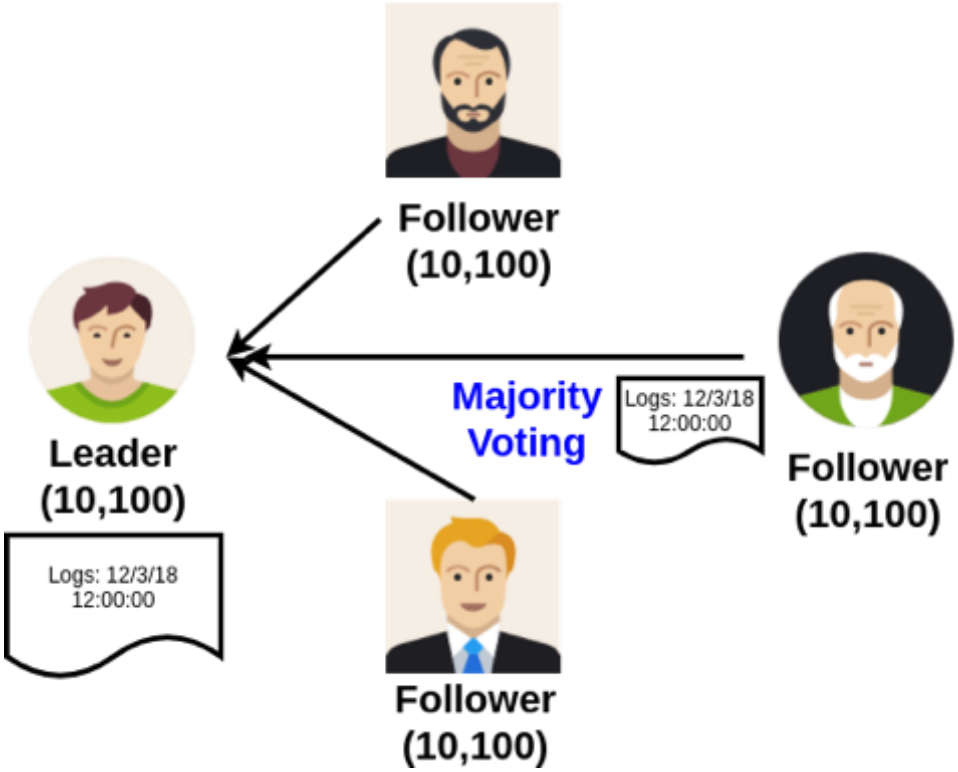
- Second part after the leader election is the index term or how will you commit the index log or to ensure the appropriate TX has been completed.
- Now the leader has the task to propose new TX
- Leader adds entry to log with term 10 and new index 101
- This particular entry made in the index log of the leader

Commiting Entry Log



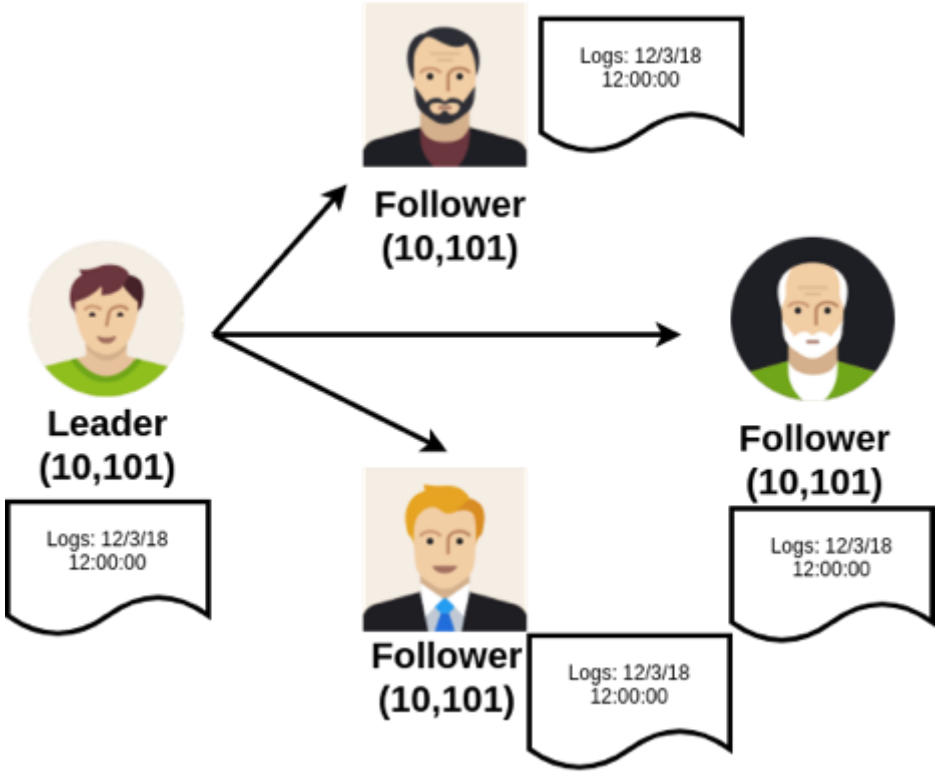
- Then, the leader sends *AppendEntries* message (it contains the actual proposal coming from the leader) to followers with index 101
- In that proposal, the leader says that now we are at index/round 10 and I am proposing Tx 101 irrespective whether you wants to commit it or not

Commiting Entry Log



- Once the Log is broadcasted to all followers then they can either vote in favor or against the Tx.
- Means, **majority voting** decides to accept or reject the entry log

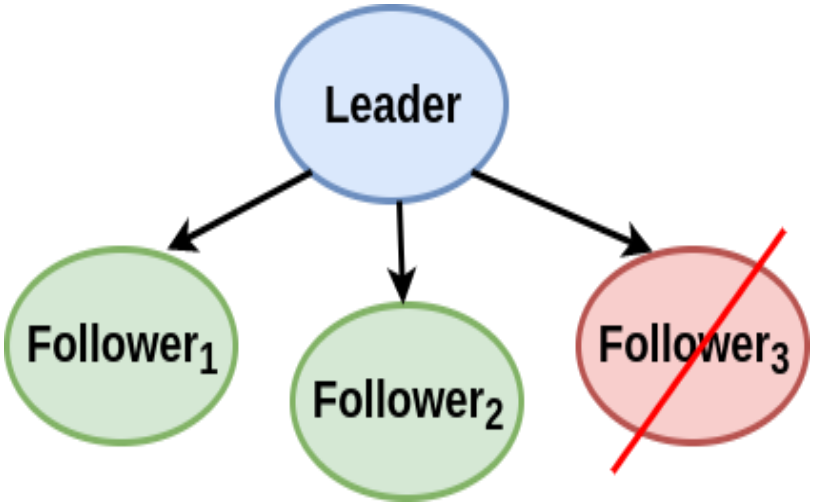
Commiting Entry Log



- Successfully accept entry log
  - All leader and followers update committed index to 101

  Here, Leader send accepted message based on majority to all followers so they update the committed index as 101

How to handle Failure in RAFT



- Paxos and RAFT are good for handling crash or network fault
- It may happen that follower may crash
- Then, Failure of up to N/2 - 1 nodes does not affect the system due to majority voting
- Because majority of the followers are non faulty and they can take part in voting
- And leader can take the decision whether to accept or reject a particular Tx

How to handle Failure in RAFT

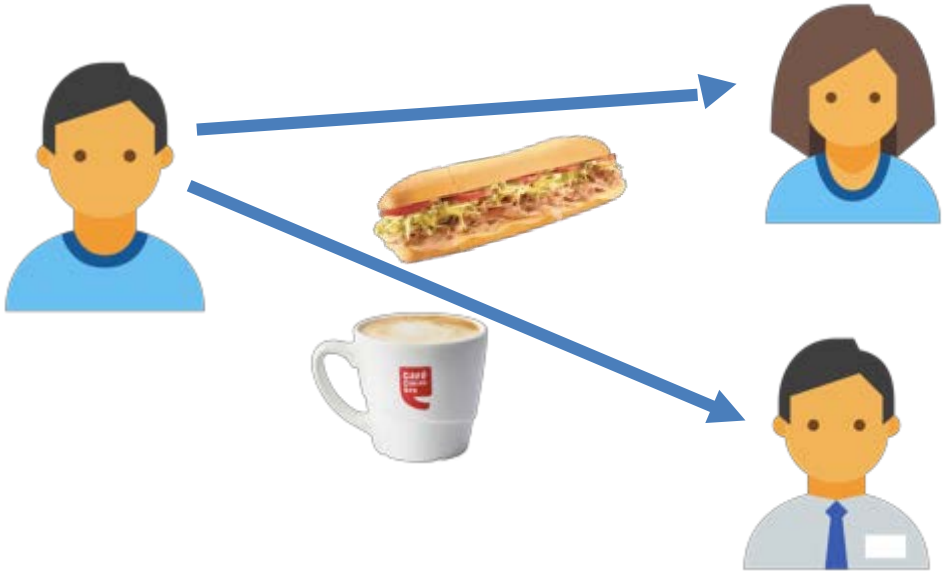
- Failure of up to N/2 - 1 nodes does not affect the system due to majority voting
- Because majority of the followers are non faulty and they can take part in voting
- Idea of Paxos was difficult to prove (because no leader in the system) and you wait for certain amount of time to see some one is proposing a value and if none of them proposing a value then you propose a value.
- If your proposal gets accepted based on majority then you send your accepted message to all nodes.
- Because there can be multiple proposer from the system so it becomes theoretically difficult to prove that how repeated execution of paxos (Multi Paxos) ???

How to handle Failure in RAFT

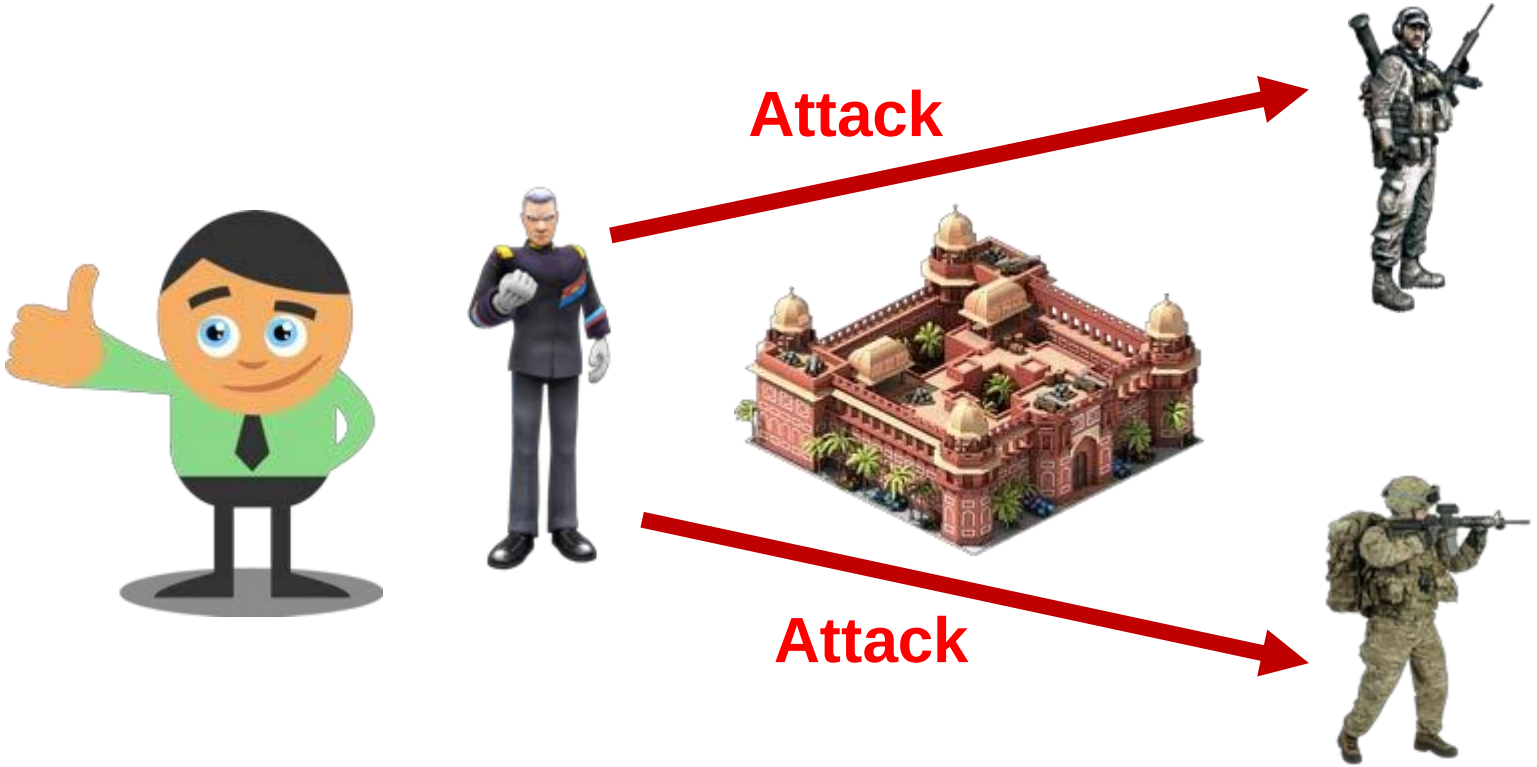
- RAFT has improved the concept of Paxos, rather than going for repeated paxos or multi-paxos, we will go for leader election mechanism.
- So in this leader election mechanism, we will apply certain consensus algorithms like majority voting
- Here in RAFT you will also have multiple leader candidates
- In RAFT, you first elect the leader based on majority then you can always **ensure that next series of TXs can be committed just by the elected leader**
- In case of multi-paxos, for every individual TXs you need to run the paxos algorithm repeatedly to come to the consensus.
- But in RAFT, you just need to execute this complicated paxos algorithm once to elect the leader so once you have elected the leader then entire things becomes streamed line because then you don’t have multiple proposer

Byzantine Generals Problem

- Paxos and Raft can tolerate up to $N/2 - 1$ number of crash faults. Means both handles only crash fault
- **What if the nodes behave maliciously**? Mean for some of the friend you propose go to NIM canteen and to some say go to K canteen. This is called as **Byzatine fault**



Byzantine fault comes from Generals Problem



Byzantine Generals Problem

Retreat

Retreat

Byzantine Generals Problem

Attack

Retreat

Limitations of Raft ?

- Raft is strictly single Leader protocol. Too much traffic can choke the system. Some variants of Paxos algorithm exist that address this bottleneck.
- There are a lot of assumptions considered to be acting, like non-occurrence of Byzantine failures, which sort of reduces the **real life applicability.**
- Raft is a more specialized approach towards a **subset of problems** which arise in achieving consensus.
- Cheap-paxos(a variant of Paxos), can work even when there is only one node functioning in the server cluster. To generalise, K+1 replicated servers can tolerate shutting down of/ fault in K servers.

Alternatives of Raft ?

Paxos – Variants :- multi-paxos, cheap paxos, fast paxos, generalised paxos
Practical Byzantine Fault Tolerance algorithm (PBFT)
Proof-of-Stake algorithm (PoS)
Delegated Proof-of-Stake algorithm (DPoS).

What are the Safety rules in Raft ?

Raft guarantees each of these safety properties :

Election safety: at most one leader can be elected in a given term.

Leader Append-Only: a leader can only append new entries to its logs (it can neither overwrite nor delete entries).

Log Matching: if two logs contain an entry with the same index and term, then the logs are identical in all entries up through the given index.

Leader Completeness: if a log entry is committed in a given term then it will be present in the logs of the leaders since this term

State Machine Safety: if a server has applied a particular log entry to its state machine, then no other server may apply a different command for the same log.

The first four rules are guaranteed by the details of the algorithm described in the previous slides. The State Machine Safety is guaranteed by a restriction on the election process.

What is the correlation of Timing and availability in Raft ?

Timing is critical in Raft to elect and maintain a steady leader over time, in order to have a perfect availability of your cluster. Stability is ensured by respecting the timing requirement of the algorithm :

broadcastTime << electionTimeout << MTBF

broadcastTime is the average time it takes a server to send request to every server in the cluster and receive responses. It is relative to the infrastructure you are using.

MTBF (Mean Time Between Failures) is the average time between failures for a server. It is also relative to your infrastructure.

electionTimeout is the same as described in the Leader Election section. It is something you must choose.

Typical number for these values can be 0.5ms to 20ms for **broadcastTime**, which implies that you set the **electionTimeout** somewhere between 10ms and 500ms. It can take several weeks or months between single server failures, which means the values are all right for a stable cluster to work.

Why it is called Raft ?

There's a few reasons that authors came up with the name Raft:
- It's not quite an acronym, but authors were thinking about the words **'reliable', 'replicated', 'redundant', and 'fault-tolerant'.**
- Authors were thinking about logs and what can be built using them.
- Authors were thinking about the island of Paxos and how to escape it.

As a plus, authors were using the randomly generated name Cheesomi in the paper before they came up with the name Raft in September 2012. The name appeared just over 100 times in their paper submission back then, so switching to the shorter name actually helped shrink the paper down quite a bit.

If you want even more detail, authors had trouble coming up with a good name, so authors made it an explicit agenda item during a **RAMCloud** meeting.

Findings

- It may happen that General has taken some bribe from enemies and started behaving maliciously.
- This problem we can call it as Byzantine General Problem
- RAFT and Paxos relies on the fact that faulty nodes will not send their votes.
- But, in reality it may happen that faulty or byzantine faulty nodes sends their votes to selective nodes.
- **For example**, if the leader behaves in a byzantine way then the leader can propose transaction to a selective number of nodes/followers.
- Then, the RAFT mechanism will not handle this situation.
- So in that case, we require another class of consensus algorithm, which handles the Byzantine fault and we call them as **Byzantine Fault Tolerance algorithms.**

Take home-Lab exercise

Implement Raft using any one-Go, C++, Java

Important References

1. The Raft Consensus Algorithm https://raft.github.io/
2. https://raft.github.io/raft.pdf
3. https://groups.google.com/forum/#!topic/raft-dev/95rZqptGpmU
4. https://www.geeksforgeeks.org/raft-consensus-algorithm/ (Good for Implementation and basic understanding)