Consensus Algorithms - II
(RAFT & Byzantine General Problem)

PAXOS (A review)

- Very complicated algorithm in theory, but very simple in concept
- Need leader election at the end of paxos
  - Then only problem solved
  - If a leader exist in the system → achieving consensus becomes much easier
  - Can then avoid the multiple proposer proposing something altogether

- Theoretical proof for PAXOS → Very difficult

PAXOS (A review)

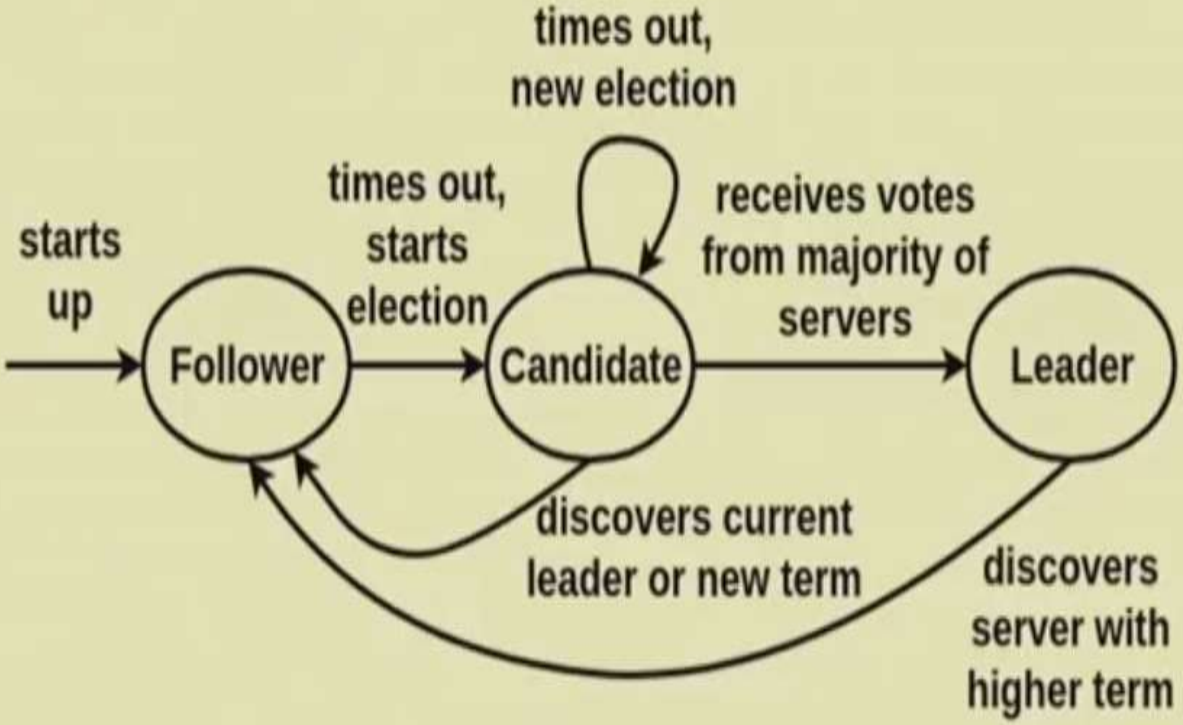
- An alternative to paxos
- Need leader election at the end of paxos
  - Then only problem solved
  - If a leader exist in the system → achieving consensus becomes much easier
  - Can then avoid the multiple proposer proposing something altogether

RAFT Consensus

- Designed as an alternative to Paxos
- 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*; others become *followers*
  - The leader is responsible for state transition log replication across the followers

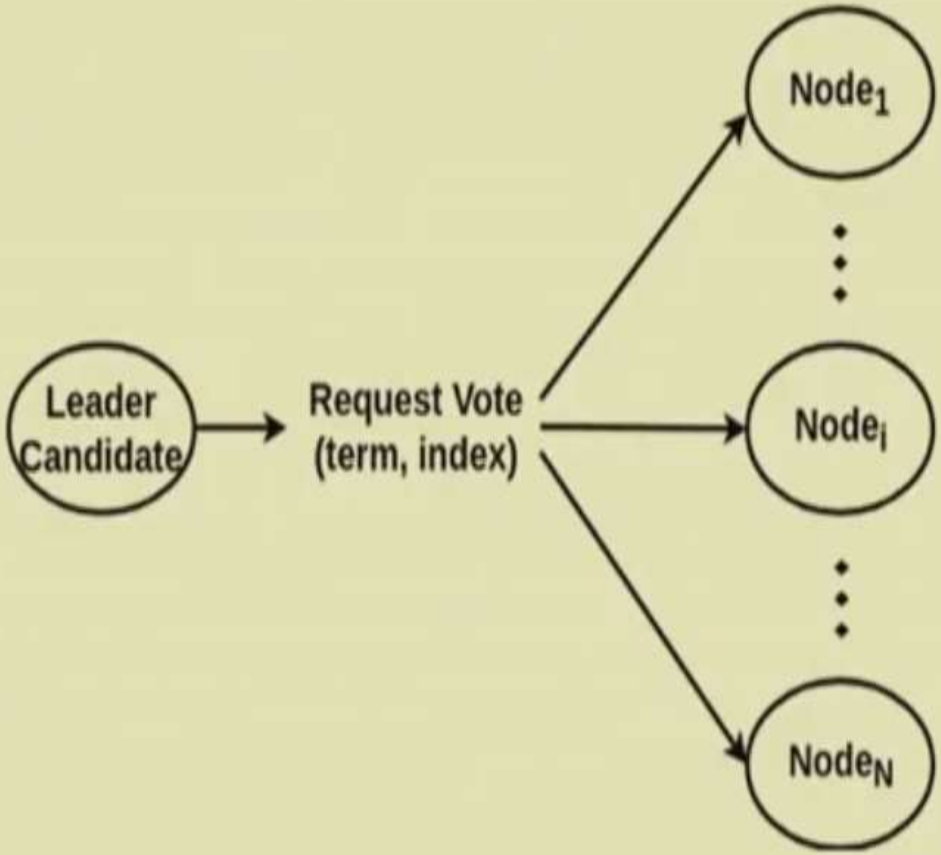
Basic idea of RAFT (**Reliable, Replicated, Redundant, And Fault-Tolerant**) is very simple

RAFT



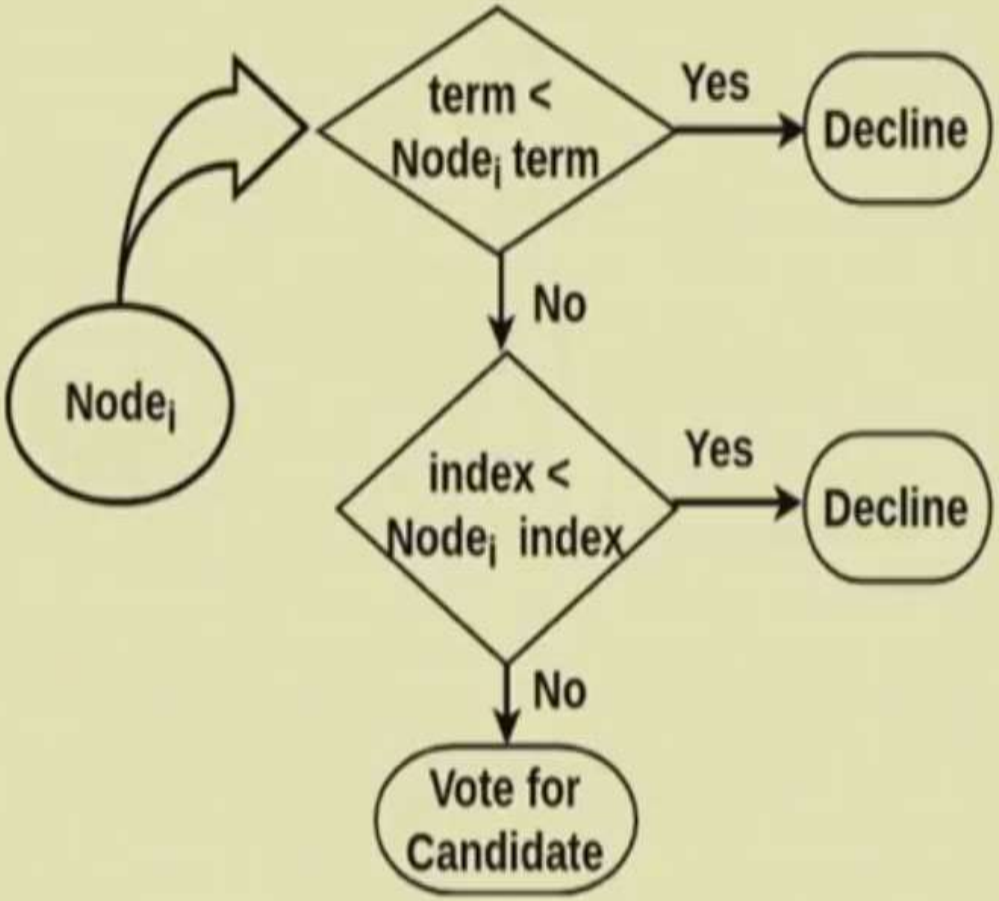
- (re)electing a leader
- committing multiple values to the transaction log
- dealing with replicas failing

Electing the Leader: Voting Request



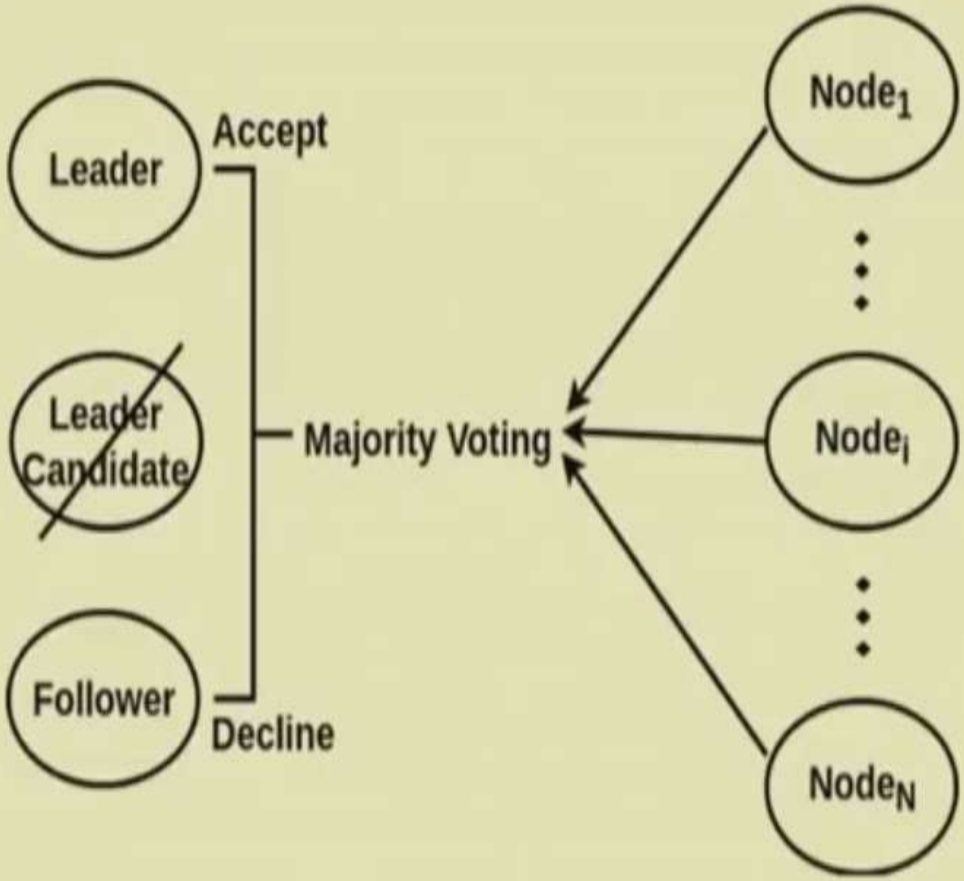
- **term:** last calculated # known to candidate + 1
- **index:** committed transaction available to the candidate

Electing the leader: Follower Node's Decision Making



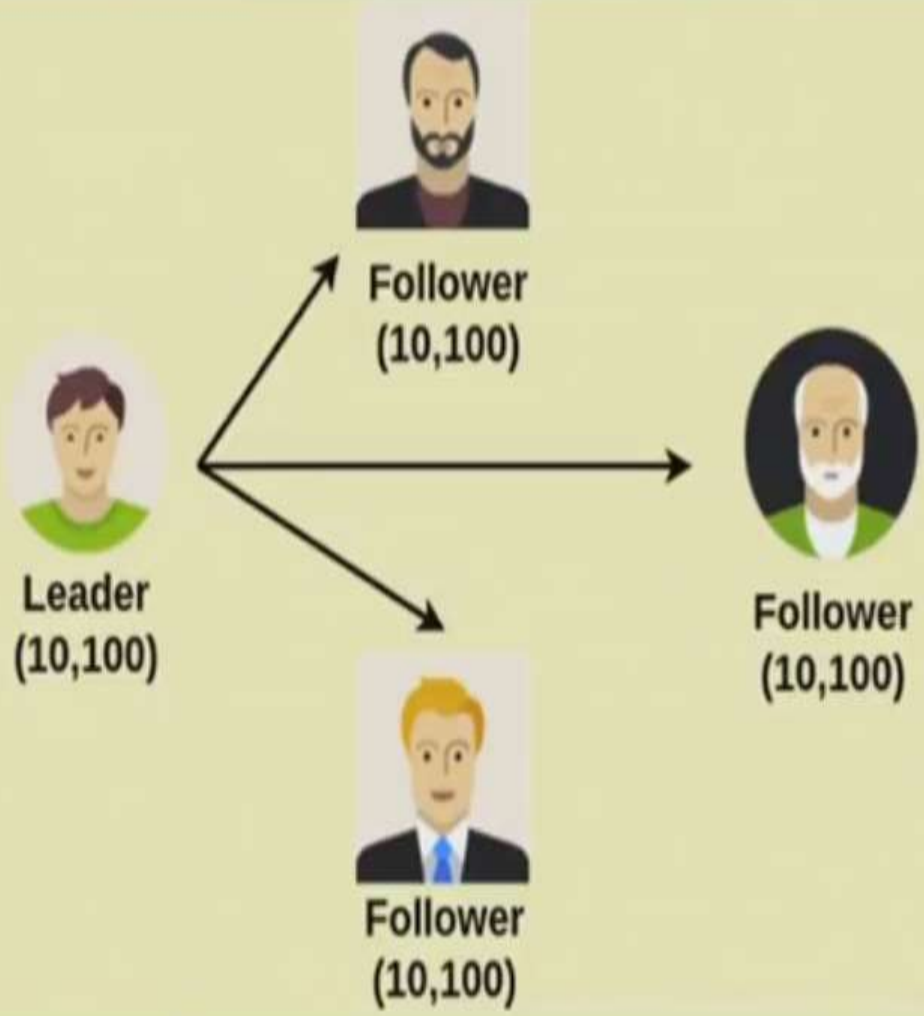
- Each node compares received term and index with corresponding current known values

Electing the leader: Majority Voting



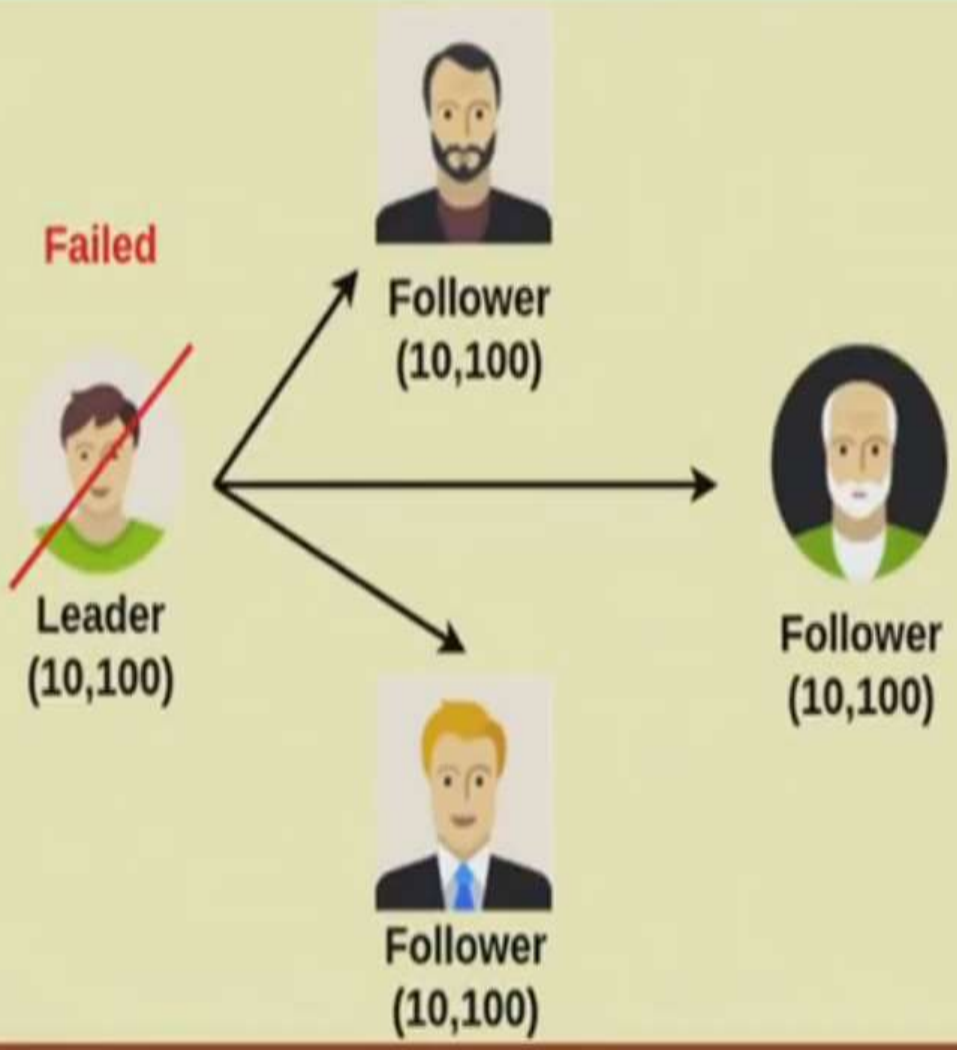
- Use of **Majority voting**
  - leader selection
  - commit the log entry

Multiple Leader Candidates: Current Leader Failure



- A leader with three followers
- **term:** 10
- **commit index:** 100

Multiple Leader Candidates: Current Leader Failure



- The leader node failed

Multiple Leader Candidates: Current Leader Failure

Follower
(11,100)

Recovered

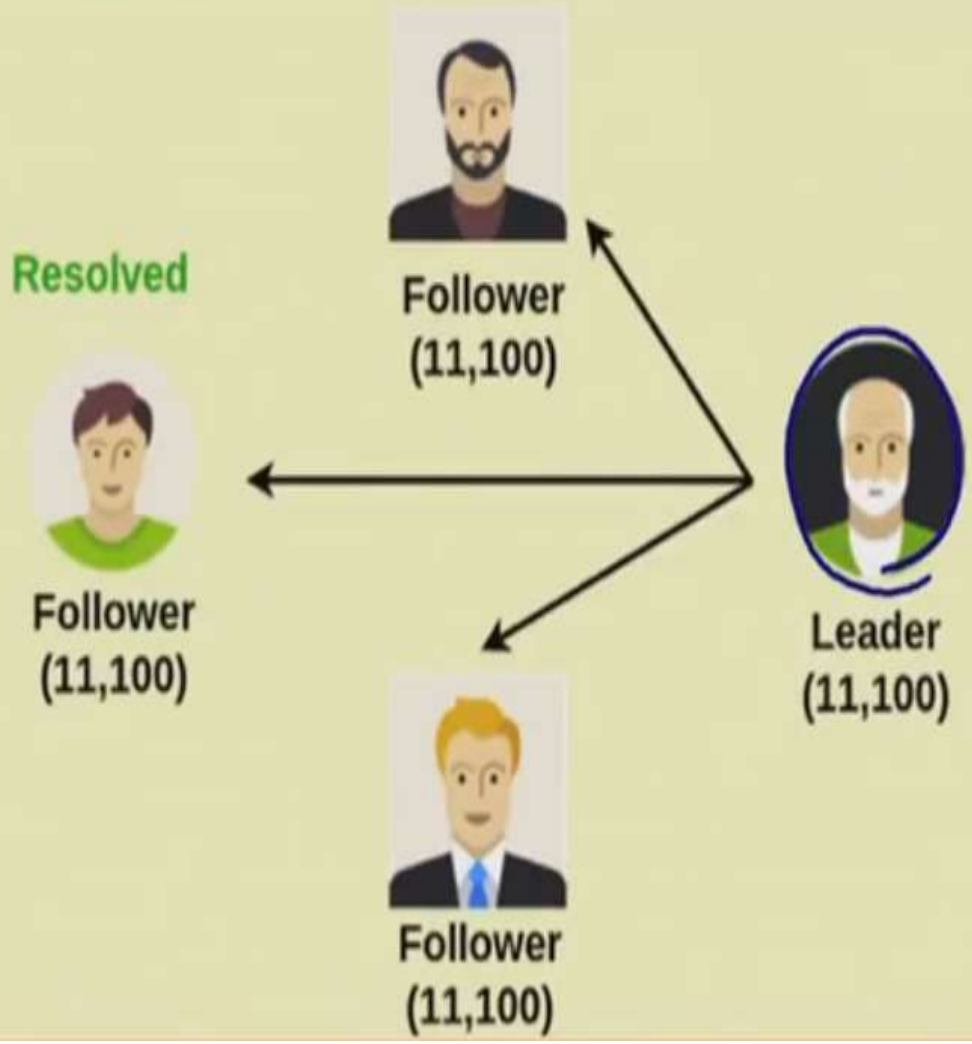
Leader
(10,100)

Leader
(11,100)

Follower
(11,100)

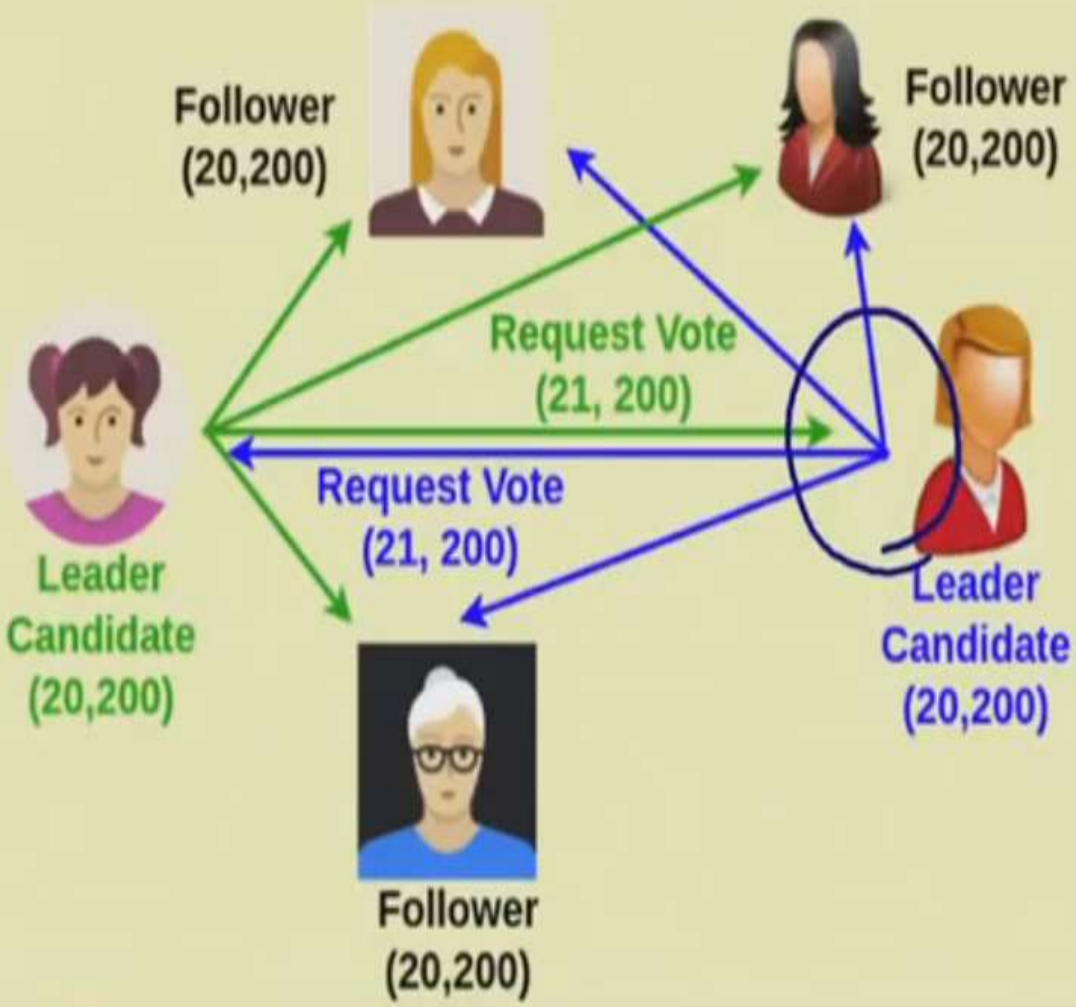
- New leader elected with term 11
- Old leader recovered

Multiple Leader Candidates: Current Leader Failure



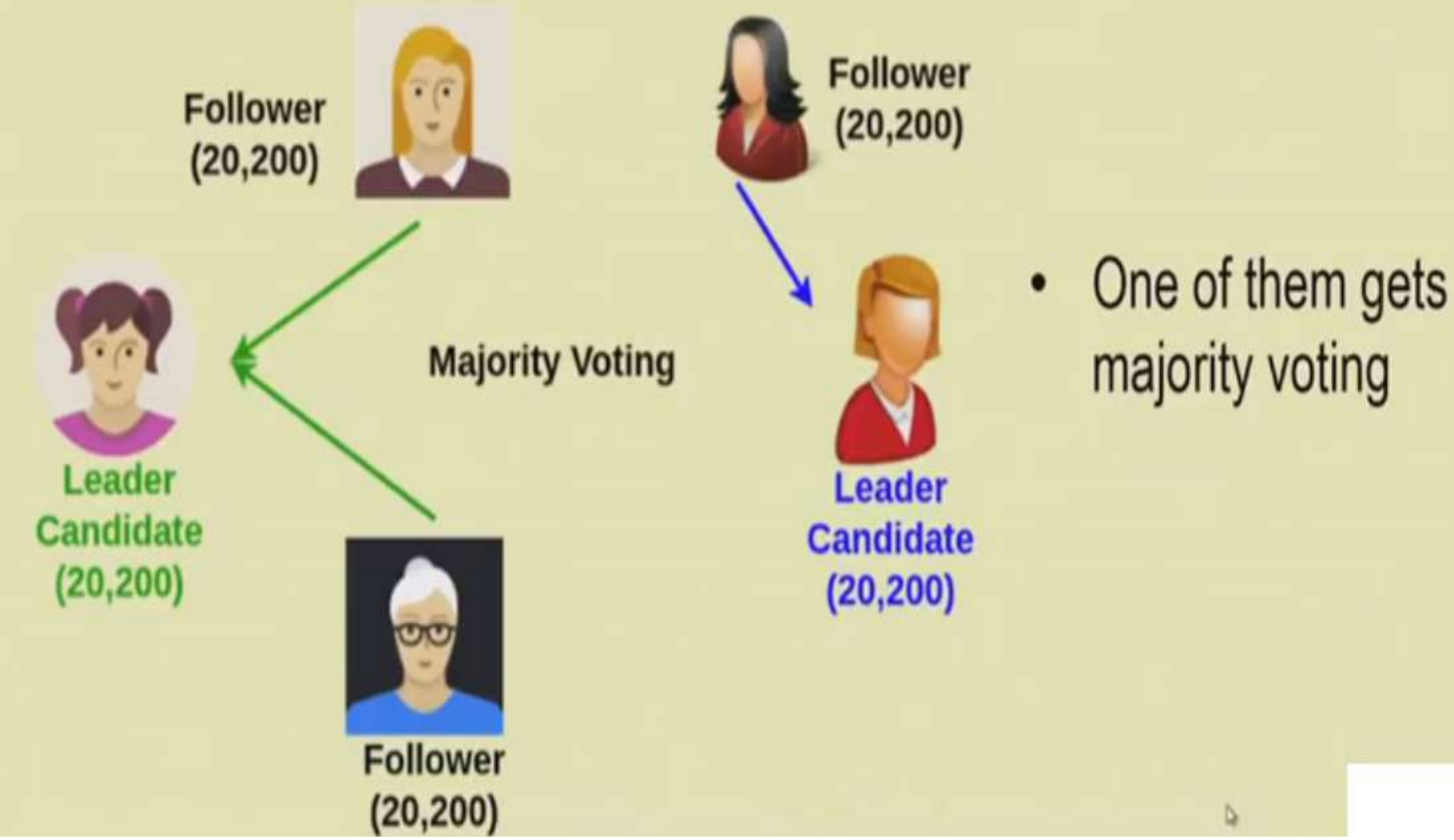
- Old leader receive heartbeat message from new leader with greater term
- Old leader drops to follower state

Multiple Leader Candidates: Simultaneous Request Vote



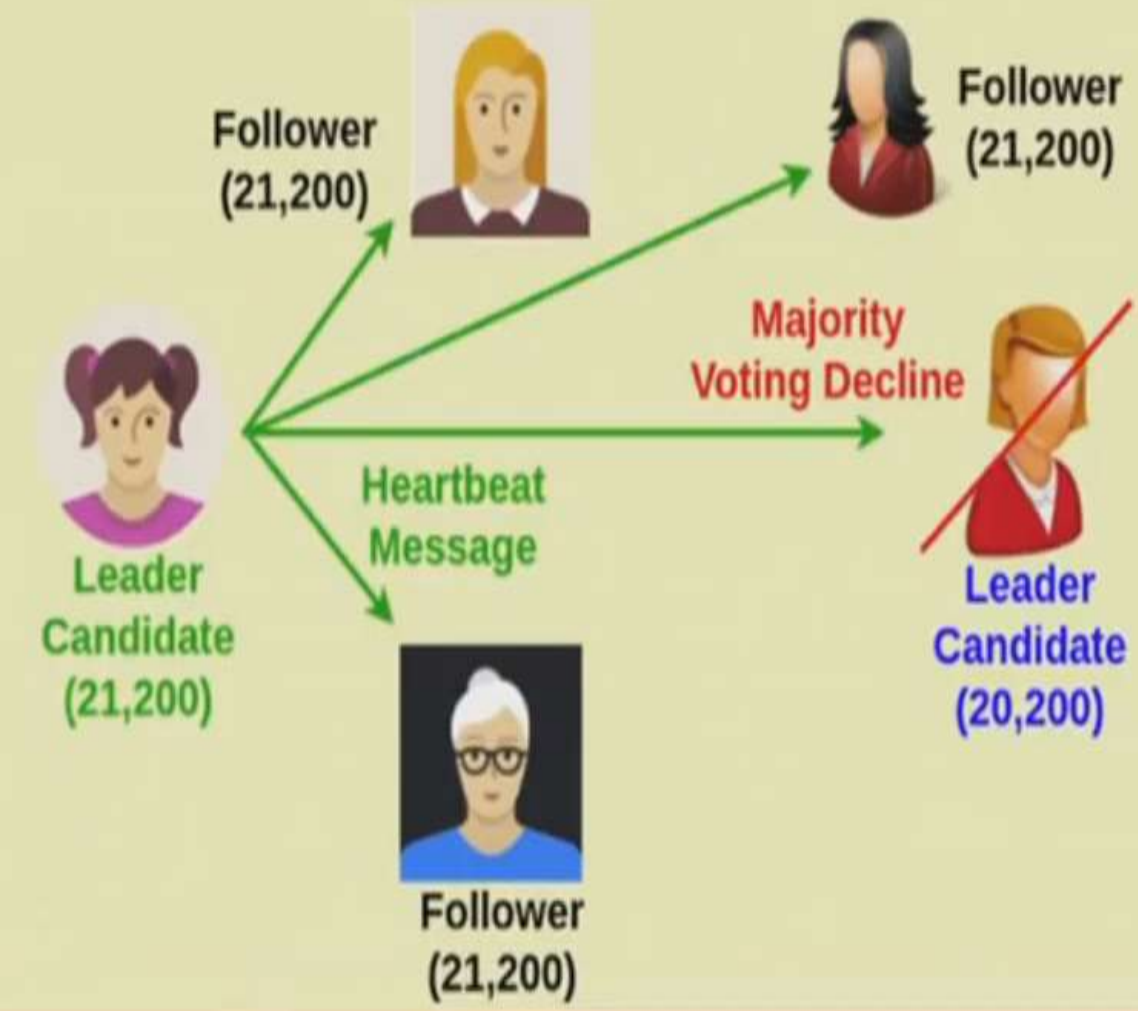
- Two nodes send Request vote message with term 21 at the same time

Multiple Leader Candidates: Simultaneous Request Vote



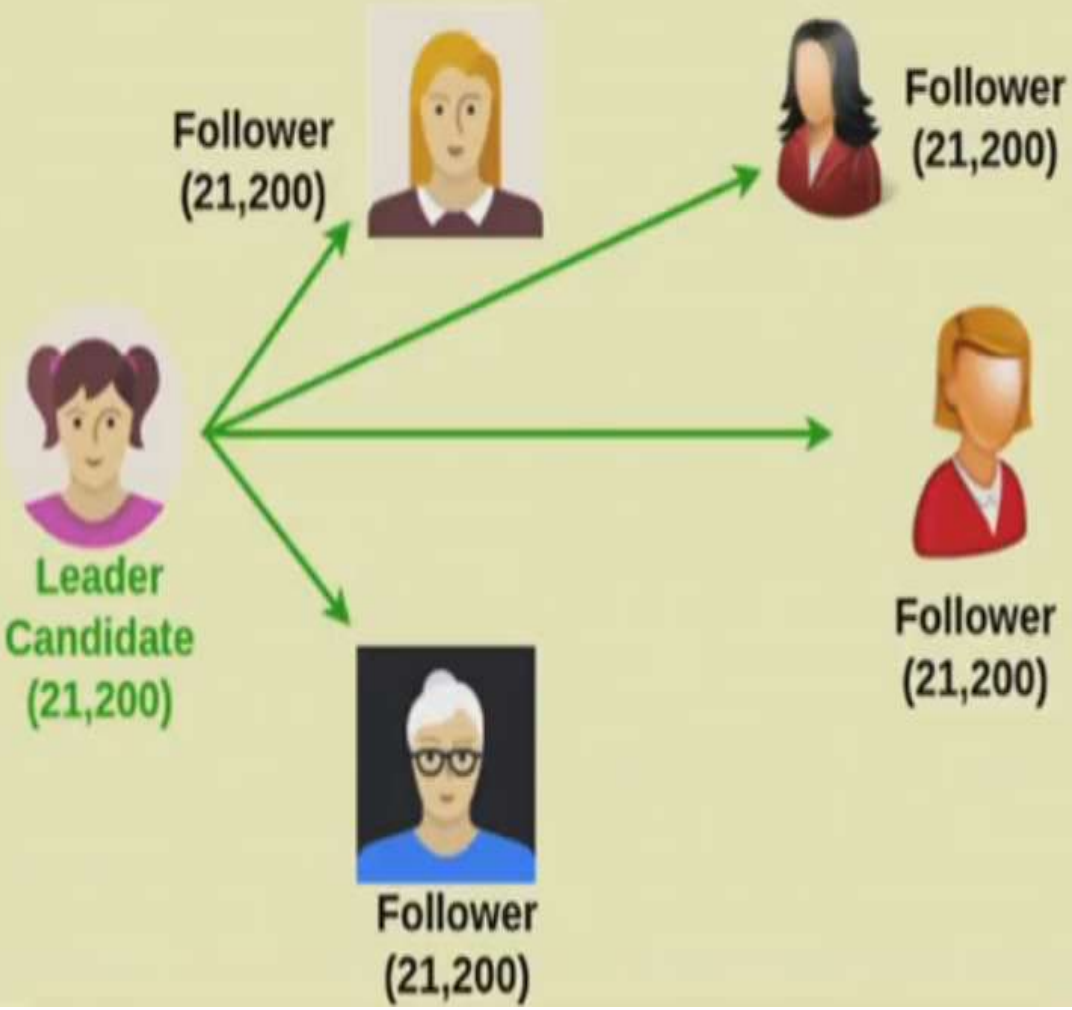
- One of them gets majority voting

Multiple Leader Candidates: Simultaneous Request Vote



- Winner sends heartbeat message

Multiple Leader Candidates: Simultaneous Request Vote



- Other leader candidate switches to follower state

Commiting Entry Log

Follower
(10,100)

Leader
(10,100)

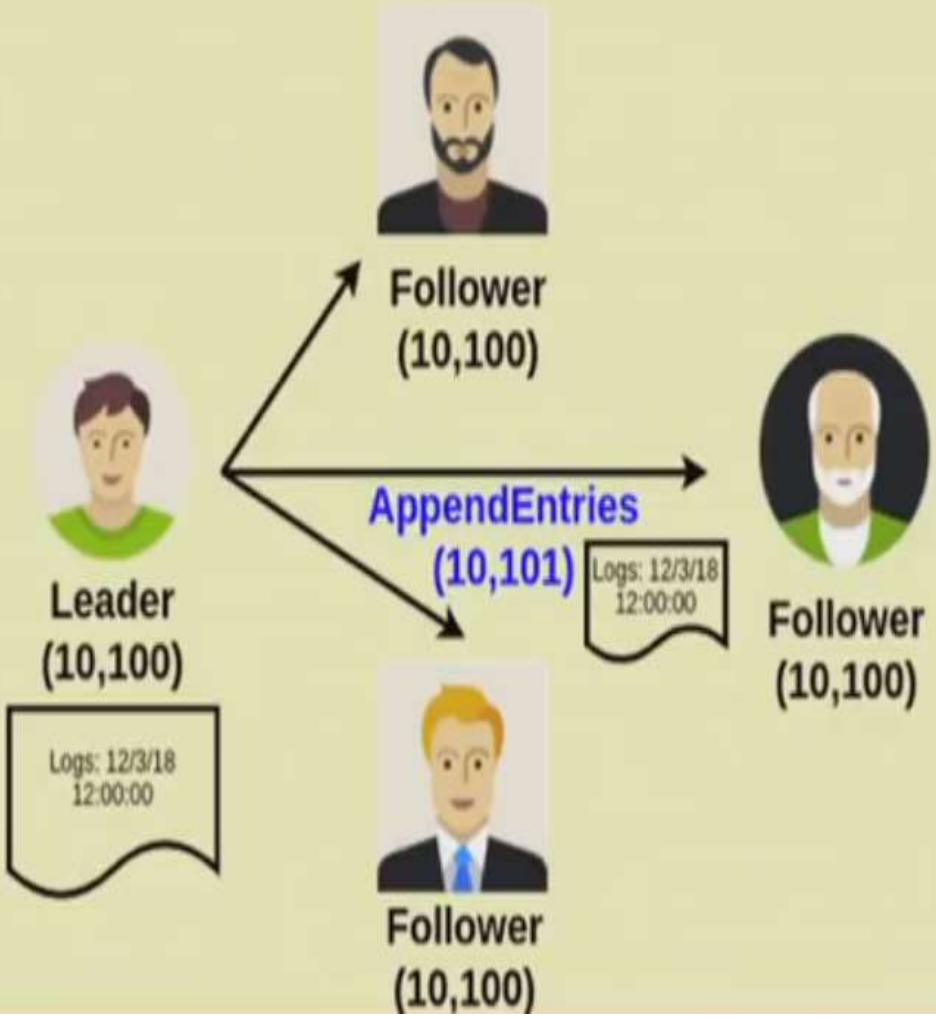
Logs: 12/3/18
12:00:00

Follower
(10,100)

Follower
(10,100)

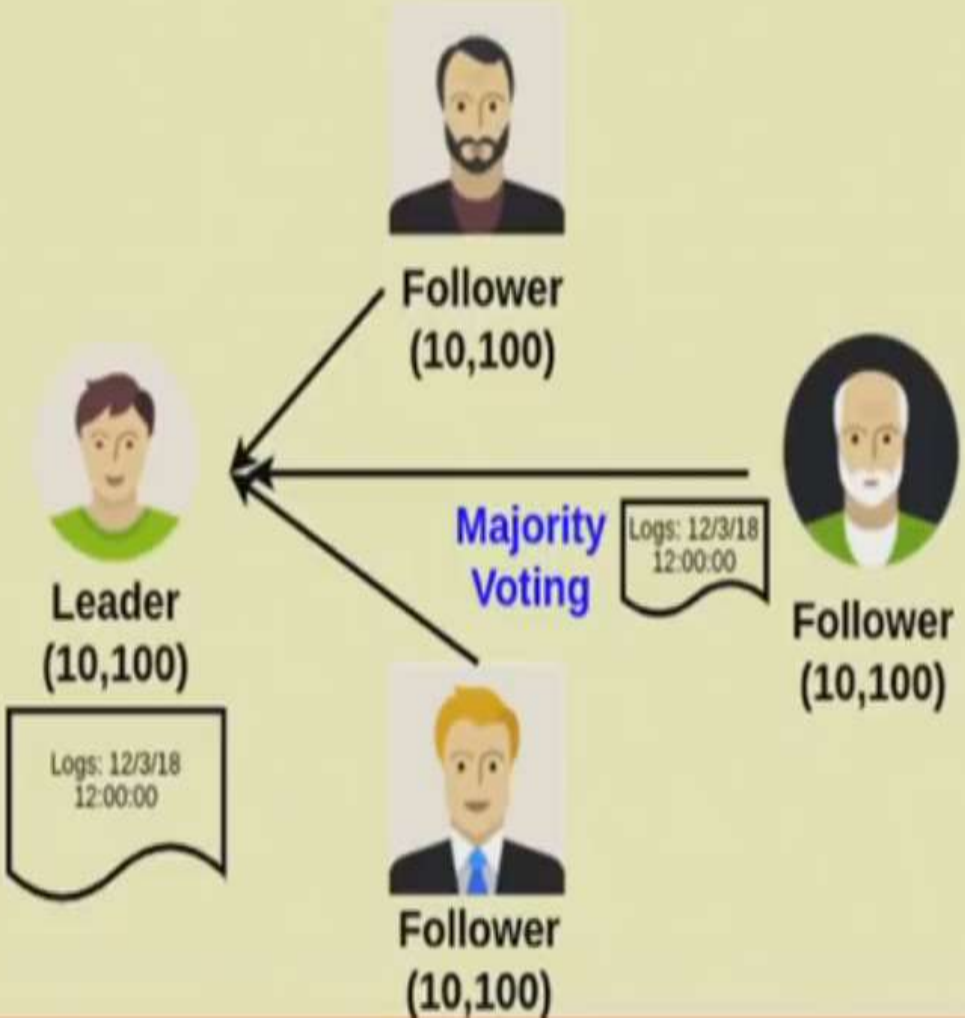
- Leader adds entry to log with term 10 and index 101

Commiting Entry Log



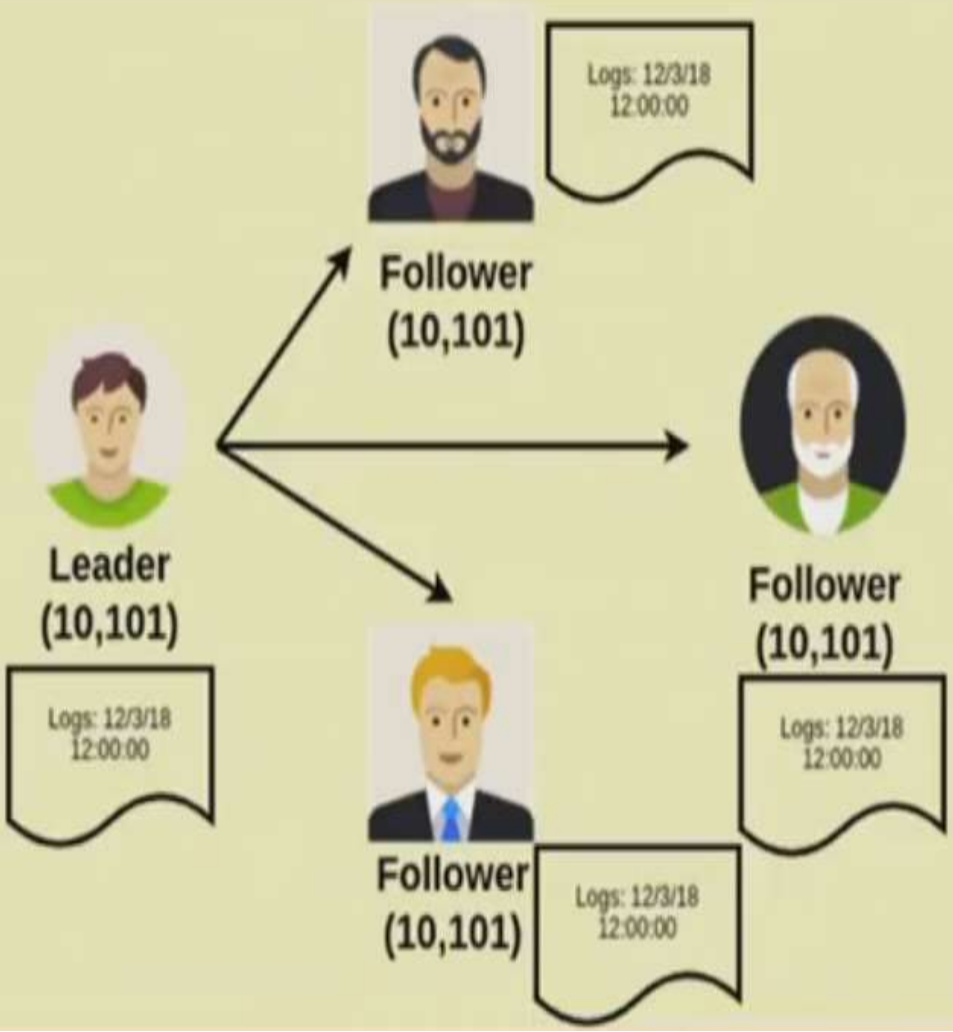
- Leader sends *AppendEntries* message to followers with index 101

Commiting Entry Log



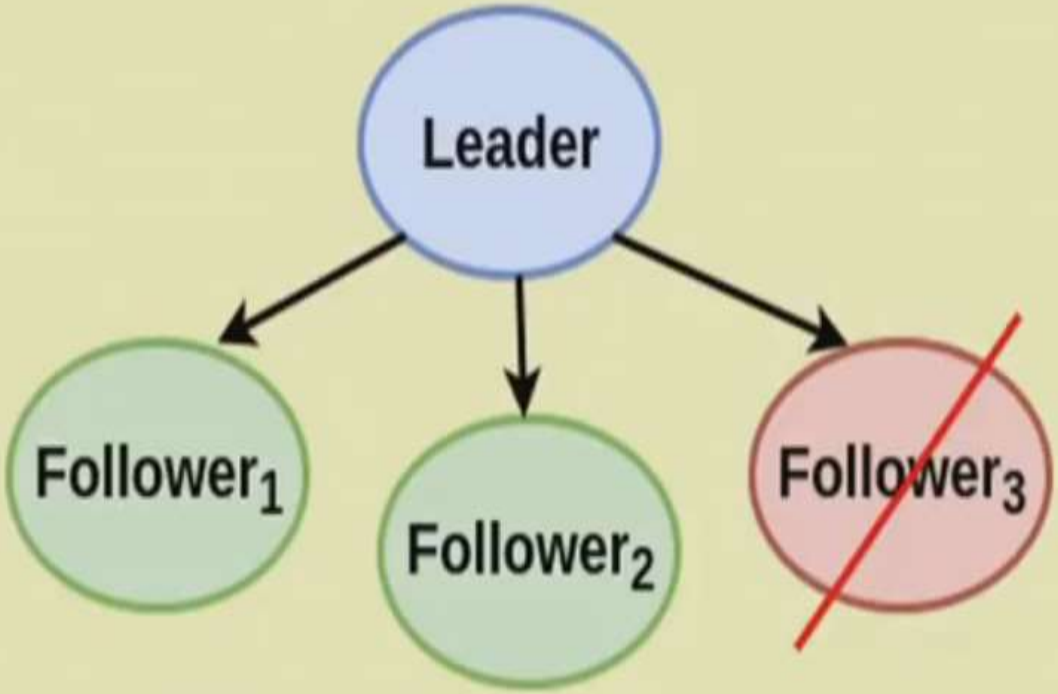
- Majority voting decides to accept or reject the entry log

Commiting Entry Log



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

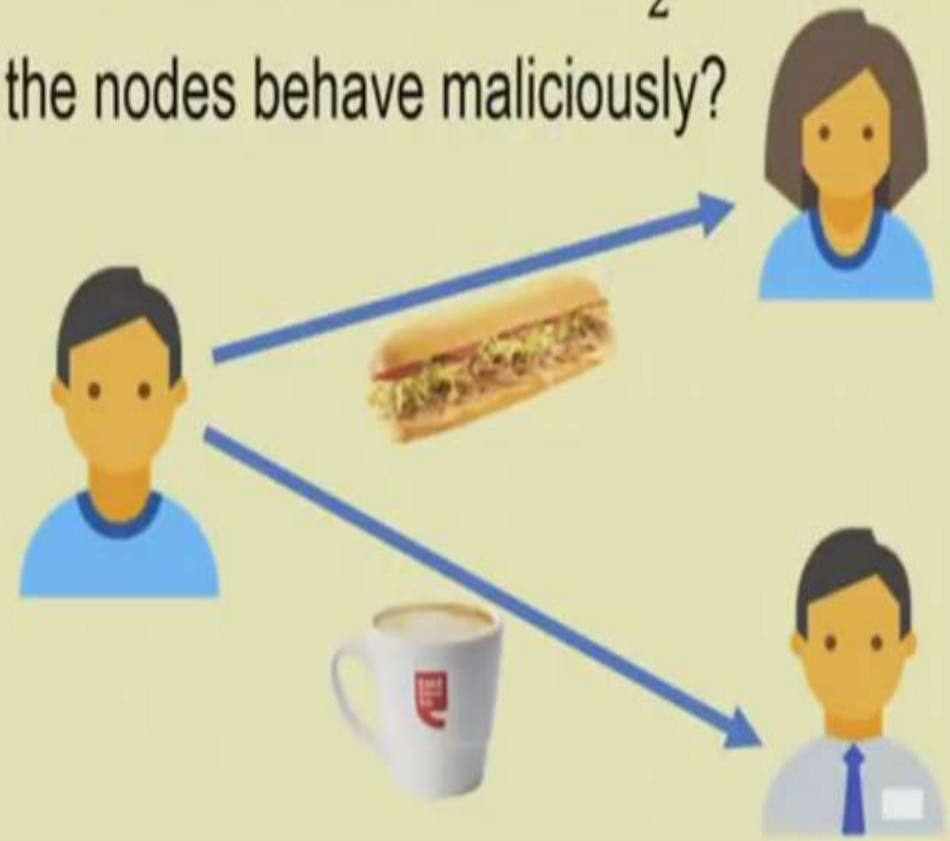
Handling Failure



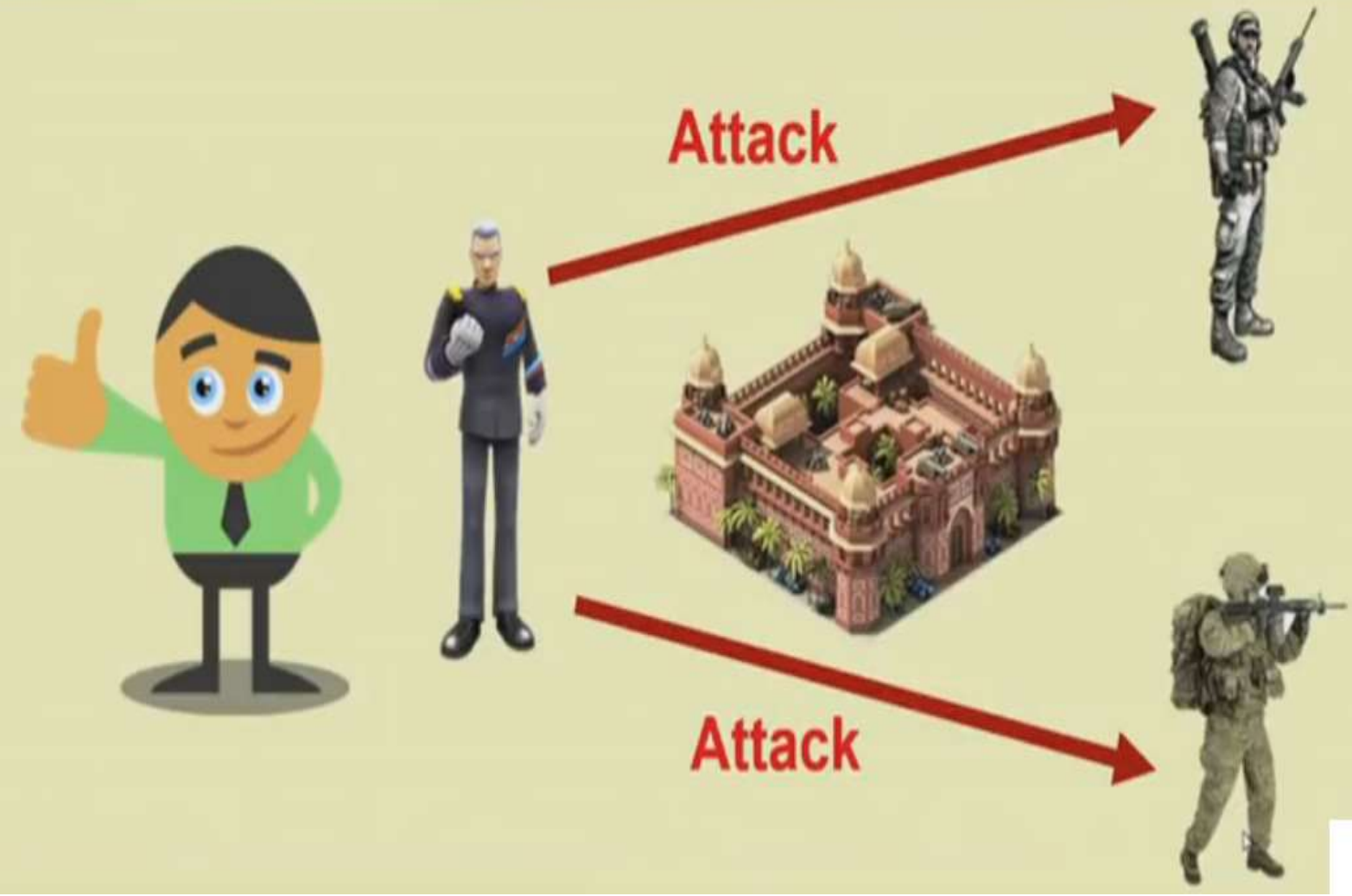
- Failure of up to N/2 - 1 nodes does not affect the system due to majority voting

Byzantine Generals Problem

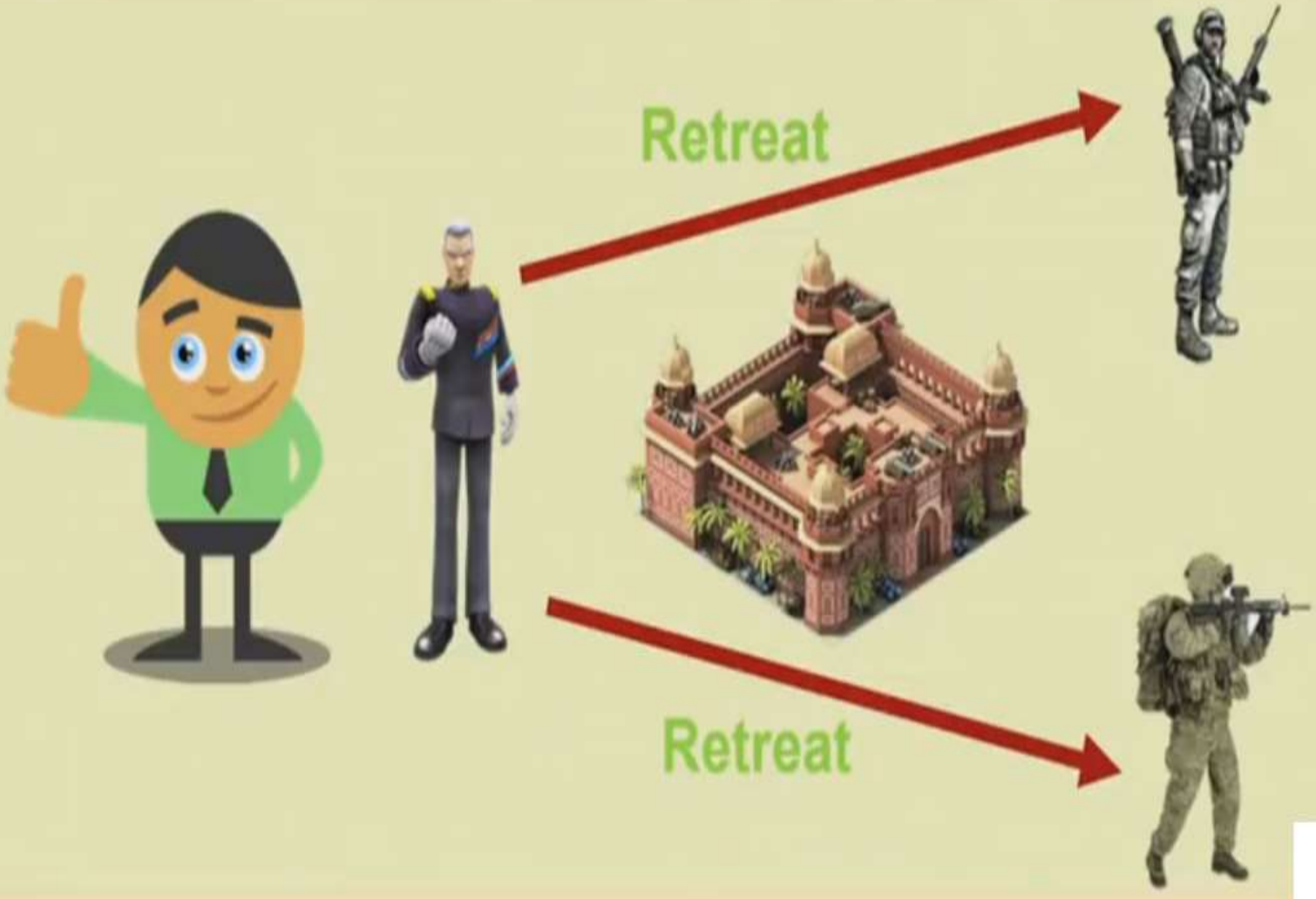
- Paxos and Raft can tolerate up to $\frac{N}{2} - 1$ number of crash faults
- What if the nodes behave maliciously?



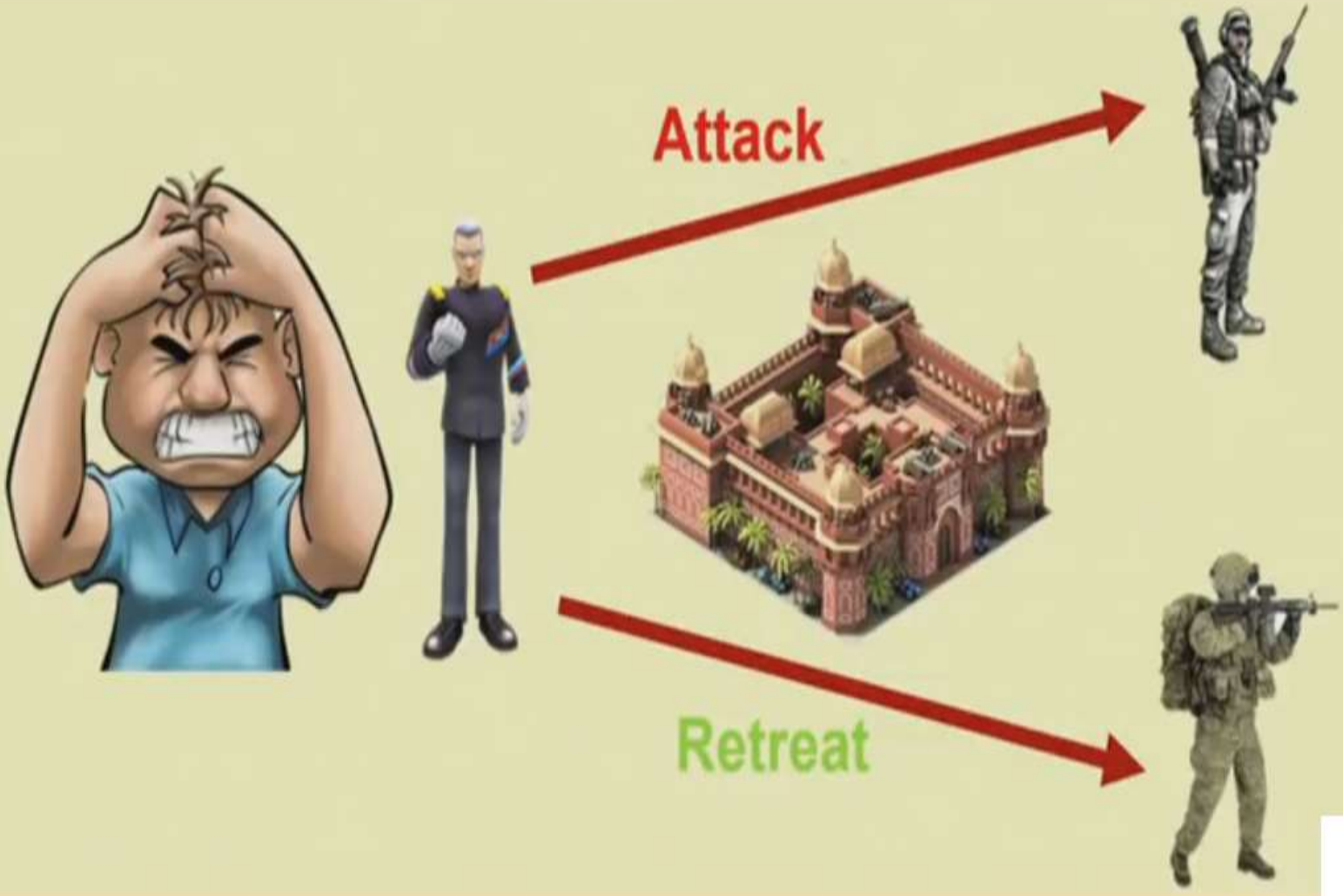
Byzantine Generals Problem



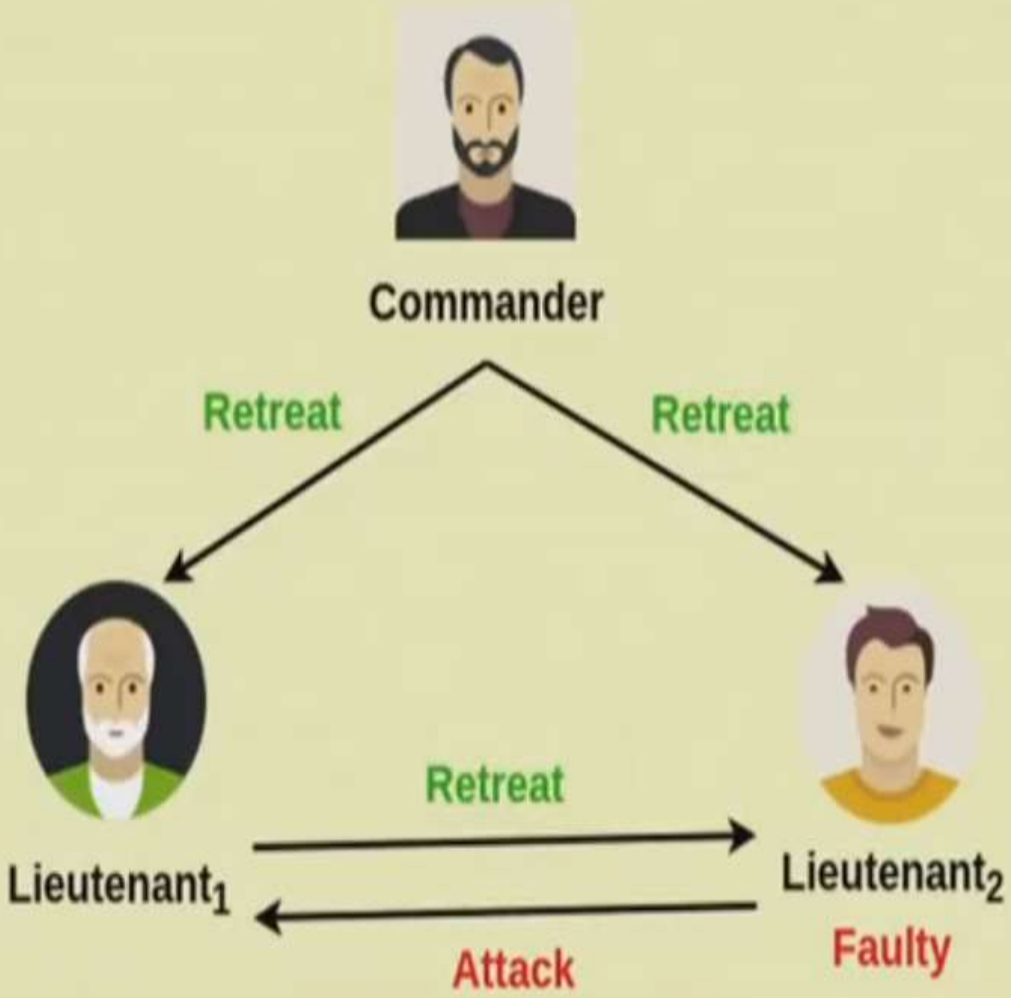
Byzantine Generals Problem



Byzantine Generals Problem

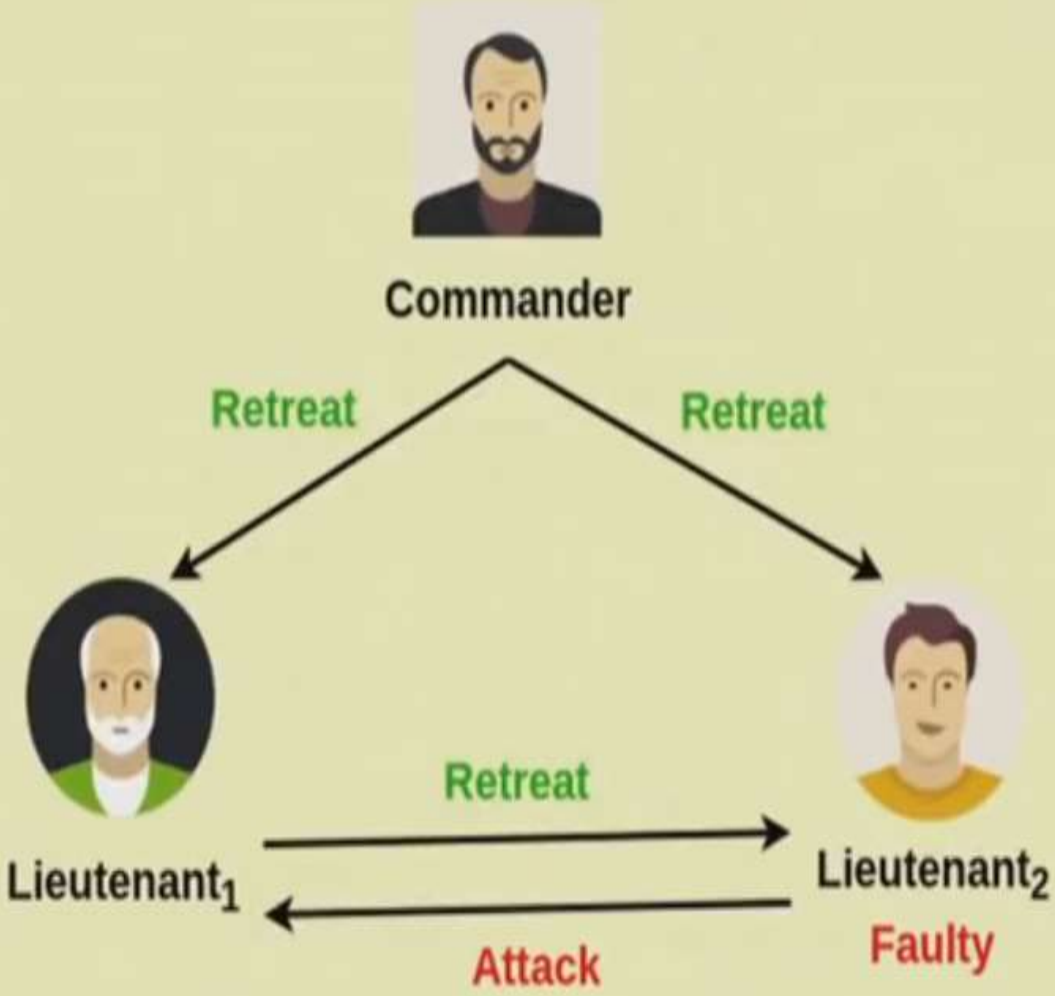


Three Byzantine Generals Problem: Lieutenant Faulty



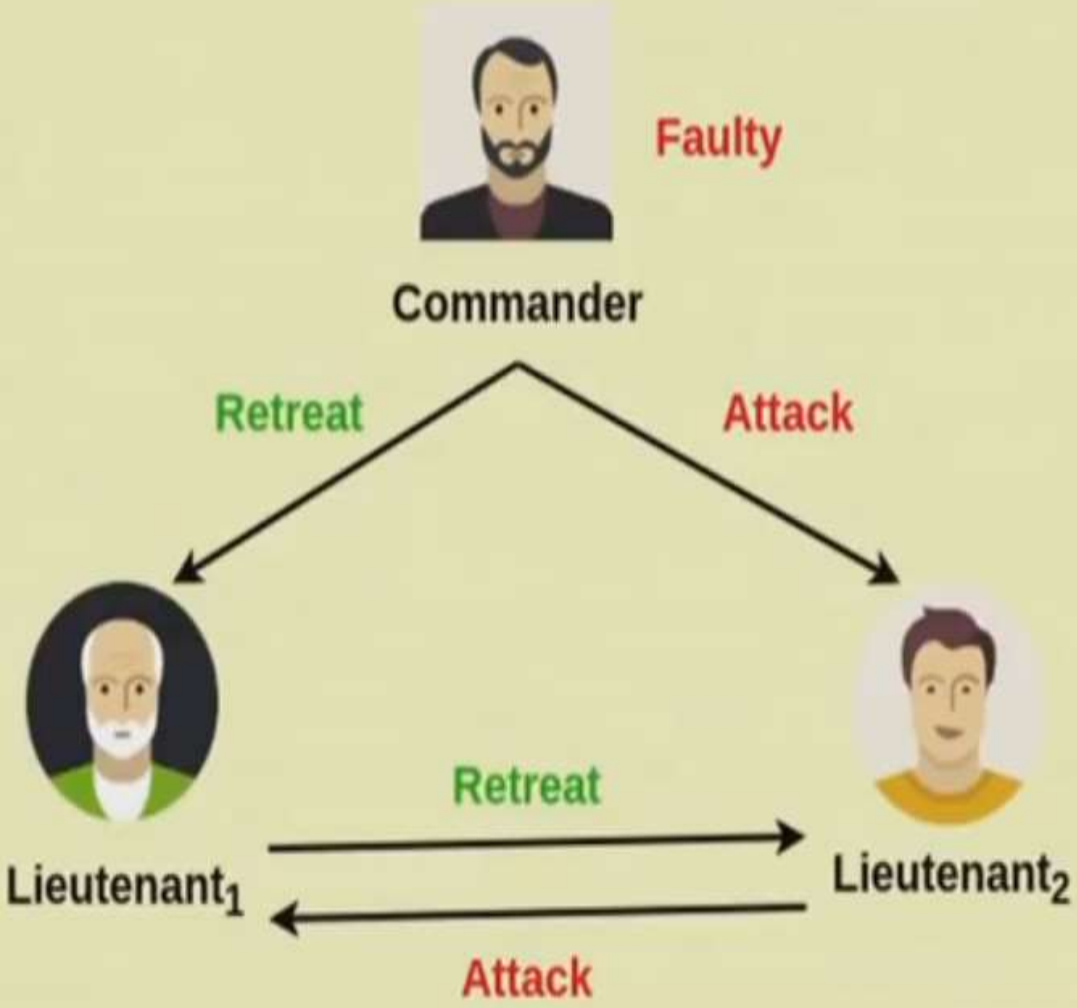
- Round1:
  - Commander correctly sends same message to Lieutenants
- Round 2:
  - Lieutenant$_1$ correctly echoes to Lieutenant$_2$
  - Lieutenant$_2$ **incorrectly** echoes to Lieutenant$_1$

Three Byzantine Generals Problem: Lieutenant Faulty



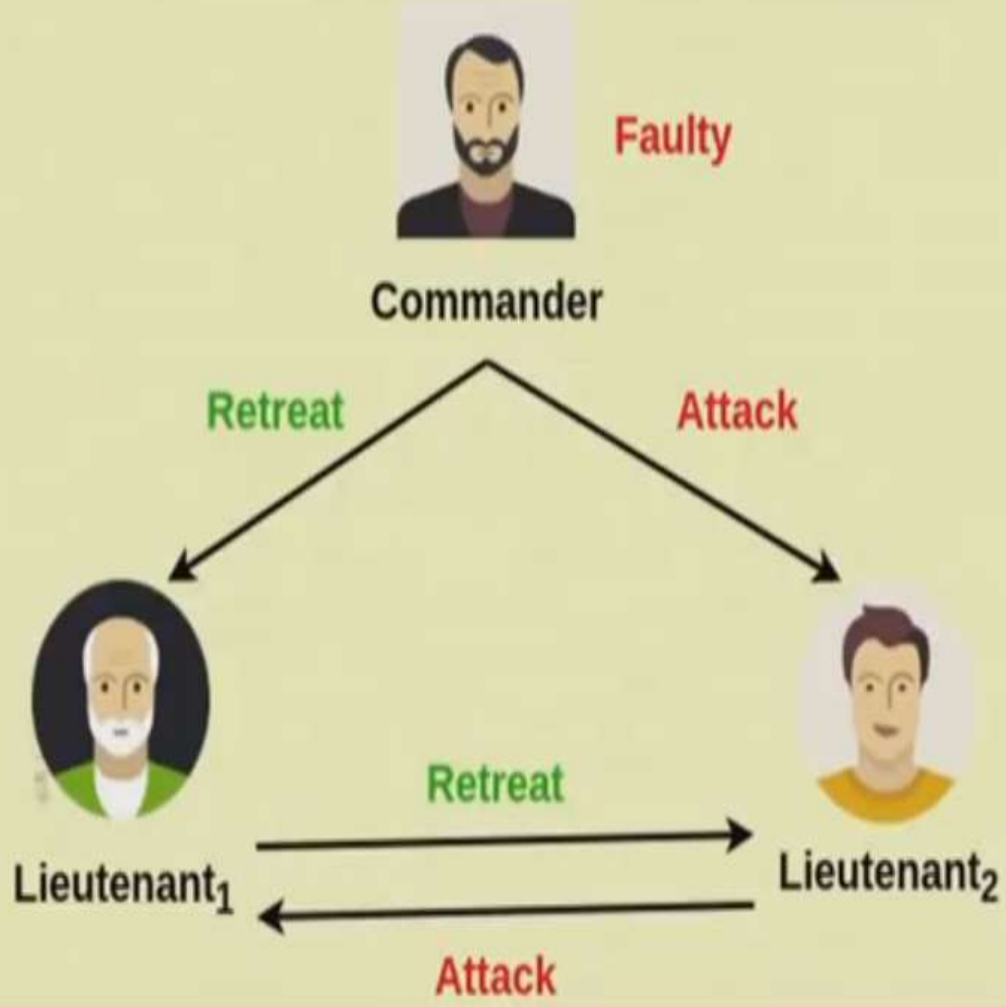
- $Lieutenant_1$ received **differing message**
- By integrity condition, $Lieutenant_1$ bound to decide on Commander message
- **What if Commander is faulty??**

Three Byzantine Generals Problem: Commander Faulty



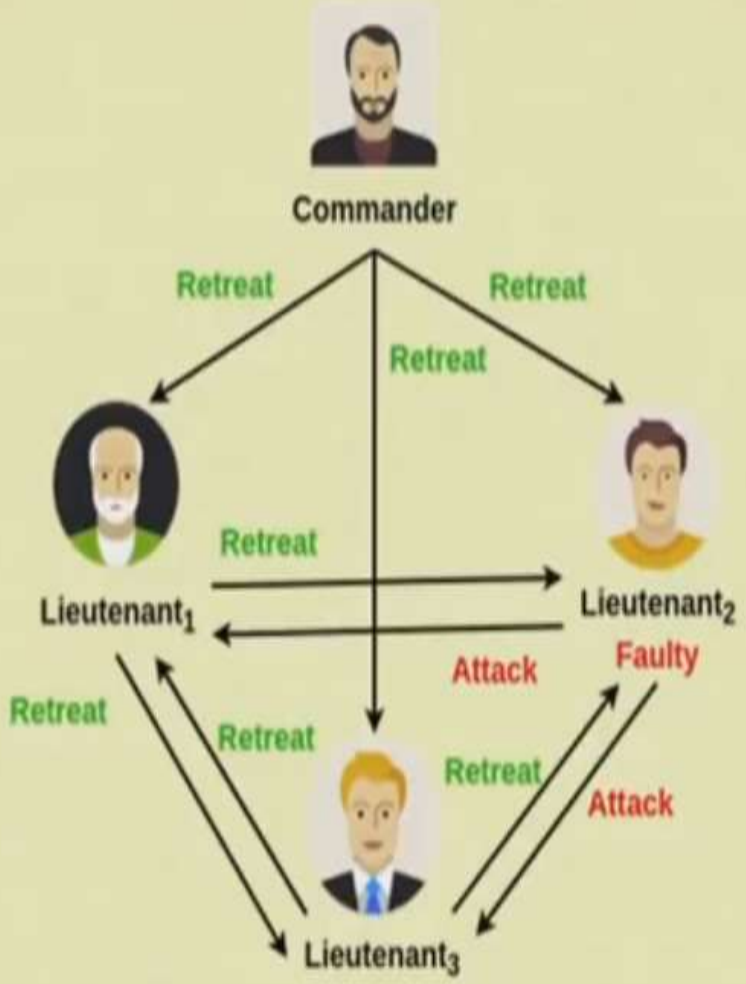
- Round 1:
  - Commander sends differing message to Lieutenants
- Round 2:
  - Lieutenant$_1$ correctly echoes to Lieutenant$_2$
  - Lieutenant$_2$ correctly echoes to Lieutenant$_1$

Three Byzantine Generals Problem: Commander Faulty



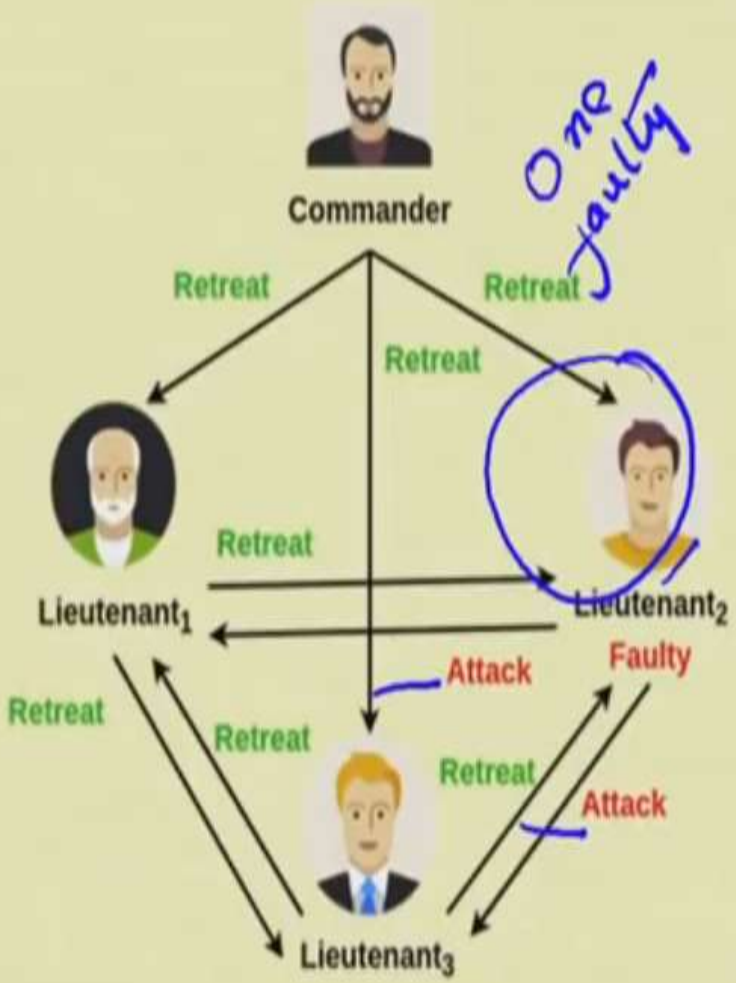
- $Lieutenant_1$ received **differing message**
- By integrity condition, both Lieutenants conclude with Commander's message
- This contradicts the agreement condition
- No solution possible for three generals including one faulty

Four Byzantine Generals Problem: Lieutenant Faulty



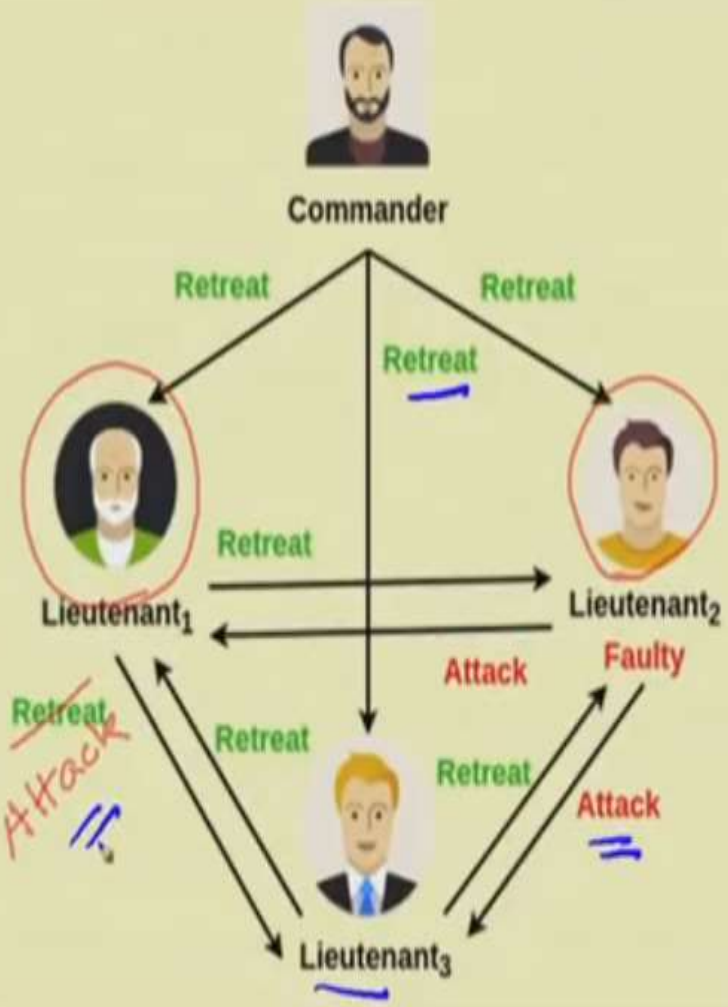
- Round 1:
  - Commander sends a message to each of the Lieutenants
- Round 2:
  - $Lieutenant_1$ and $Lieutenant_3$ correctly echo the message to others
  - $Lieutenant_2$ **incorrectly** echoes to others

Four Byzantine Generals Problem: Lieutenant Faulty



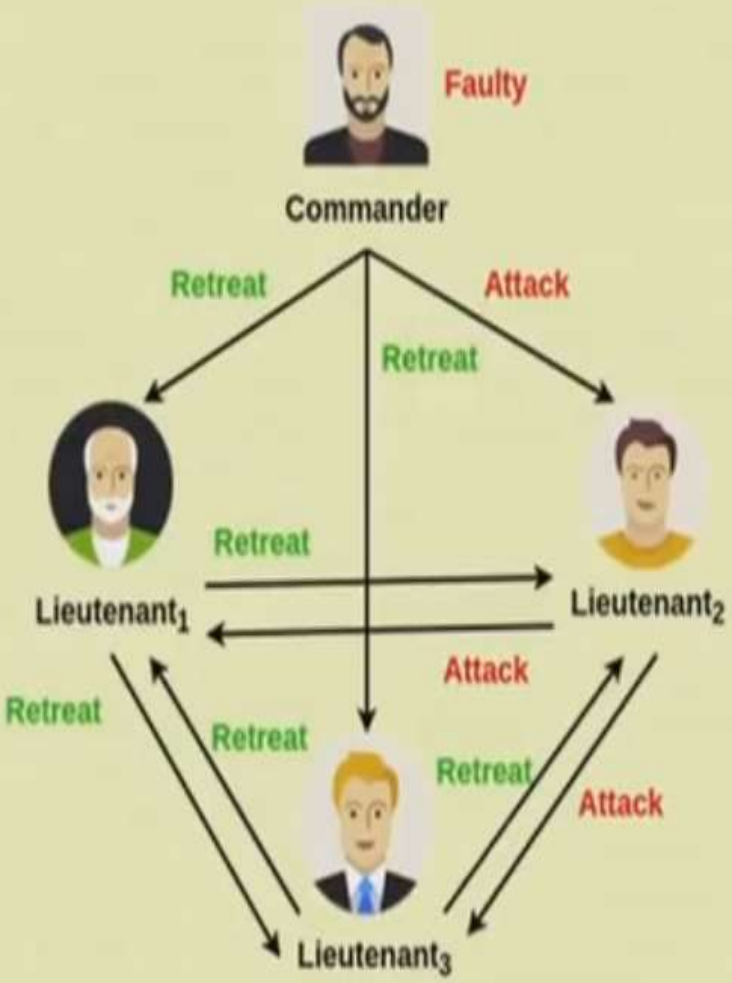
- Lieutenant$_1$ decides on *majority*(Retreat,Attack,Retreat)= Retreat
- Lieutenant$_3$ decides on *majority*(Retreat,Retreat,Attack)= Retreat

Four Byzantine Generals Problem: Lieutenant Faulty



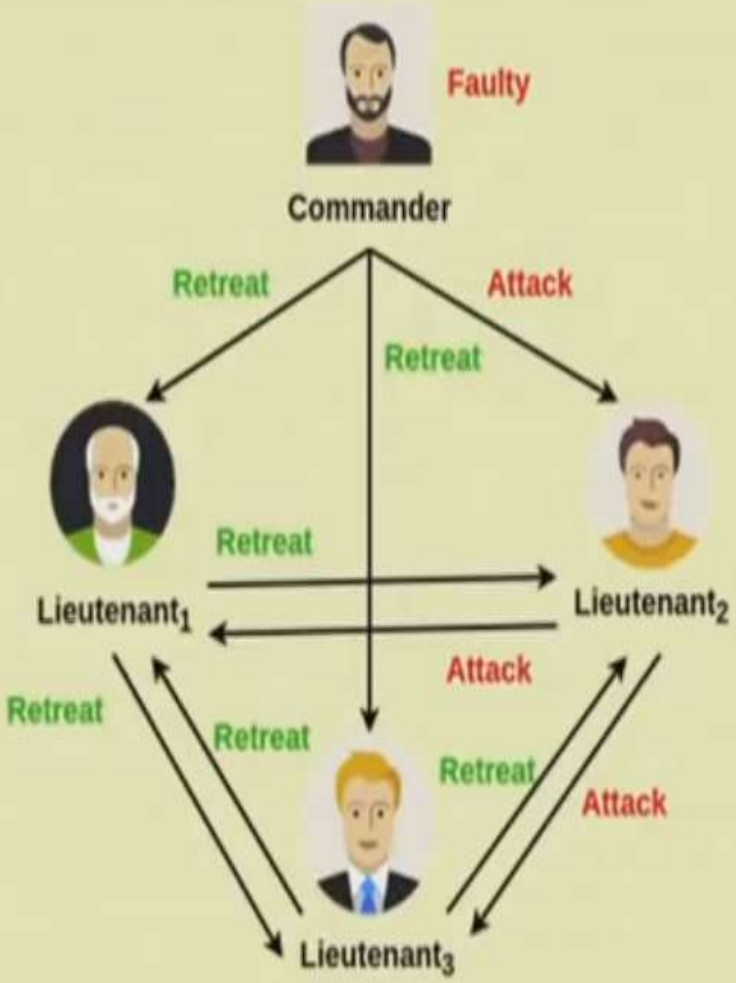
- Lieutenant$_1$ decides on *majority*(Retreat,Attack,Retreat)= Retreat
- Lieutenant$_3$ decides on *majority*(Retreat,Retreat,Attack)= Retreat

Four Byzantine Generals Problem: Commander Faulty



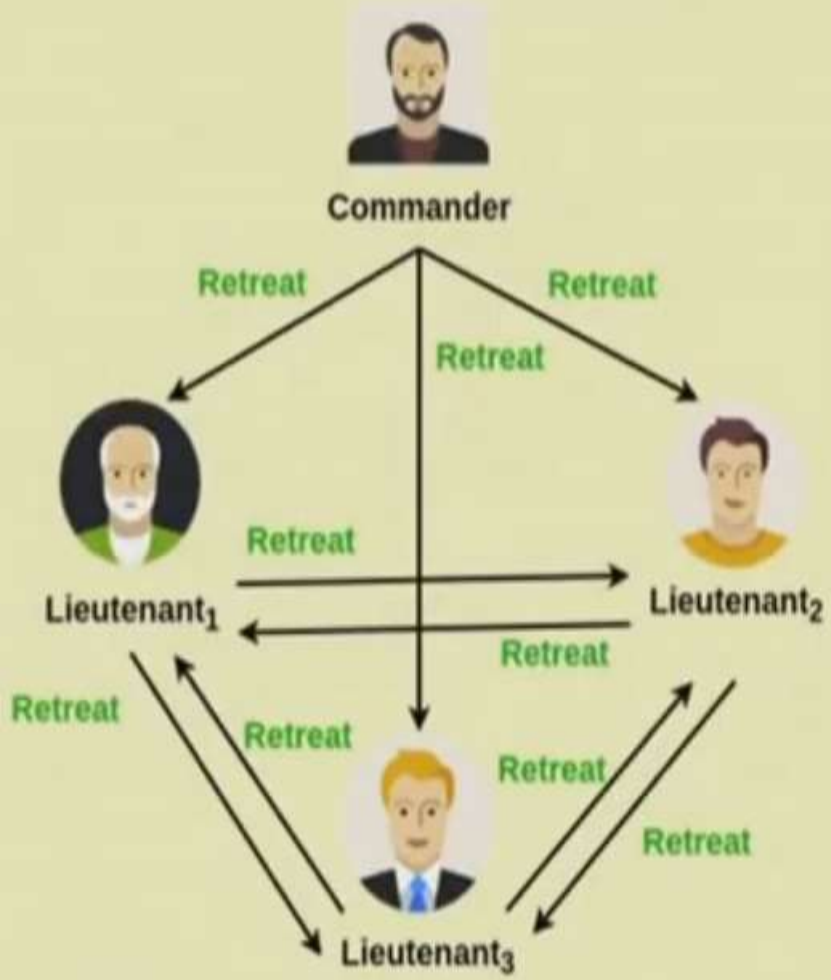
- Round 1:
  - Commander sends differing message to Lieutenants
- Round 2:
  - $Lieutenant_1$, $Lieutenant_2$ and $Lieutenant_3$ correctly echo the message to others

Four Byzantine Generals Problem: Commander Faulty



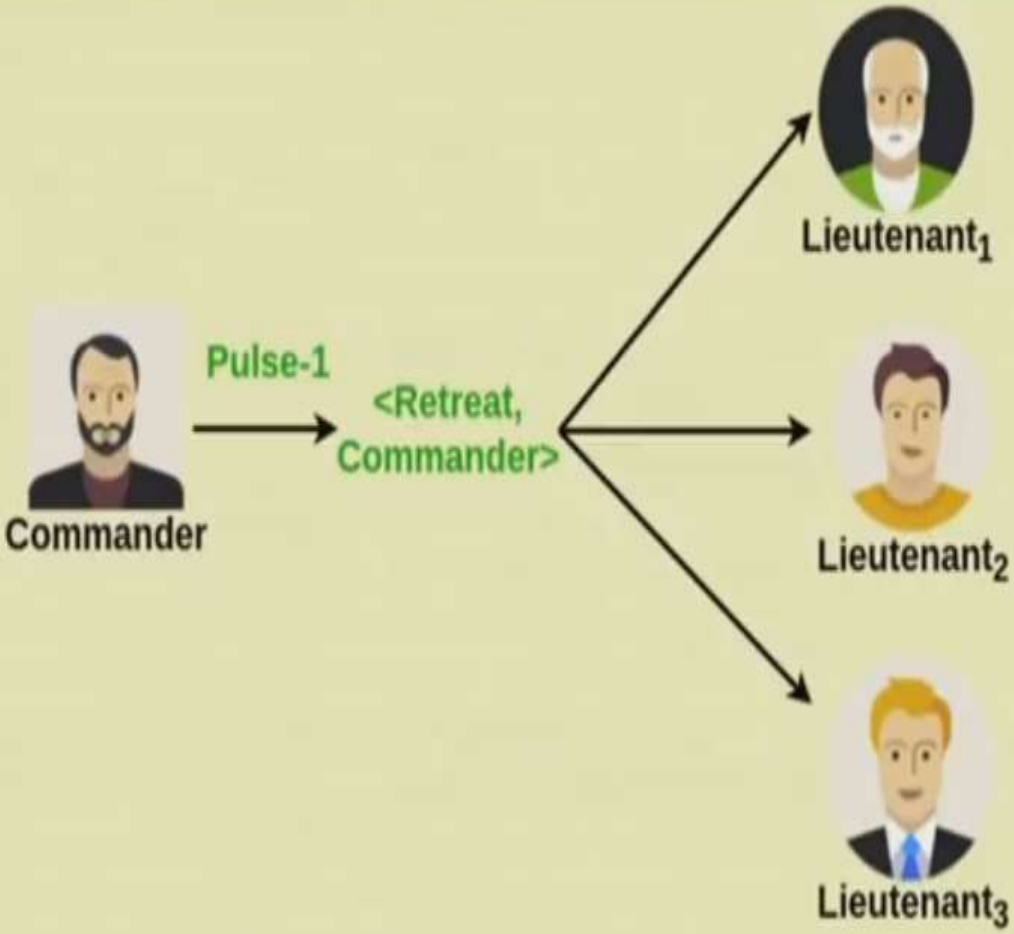
- Lieutenant$_1$ decides on *majority*(Retreat,Attack,Retreat)= Retreat
- Lieutenant$_2$ decides on *majority*(Attack,Retreat,Retreat)= Retreat
- Lieutenant$_3$ decides on *majority*(Retreat,Retreat,Attack)= Retreat

Byzantine Generals Model



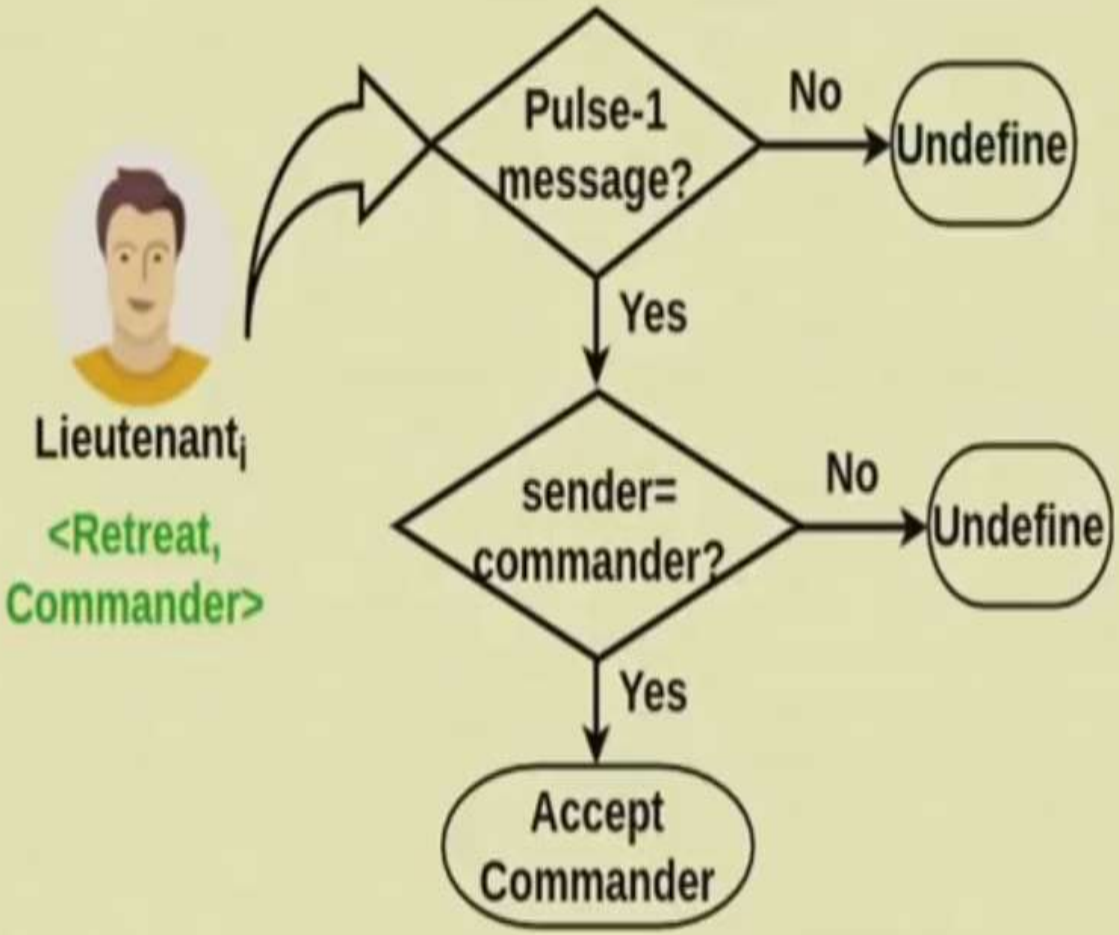
- *N* number of process with at most *f* faulty
- Receiver always knows the identity of the sender
- Fully connected
- Reliable communication medium
- Synchronous system

Lamport-Shostak-Pease Algorithm



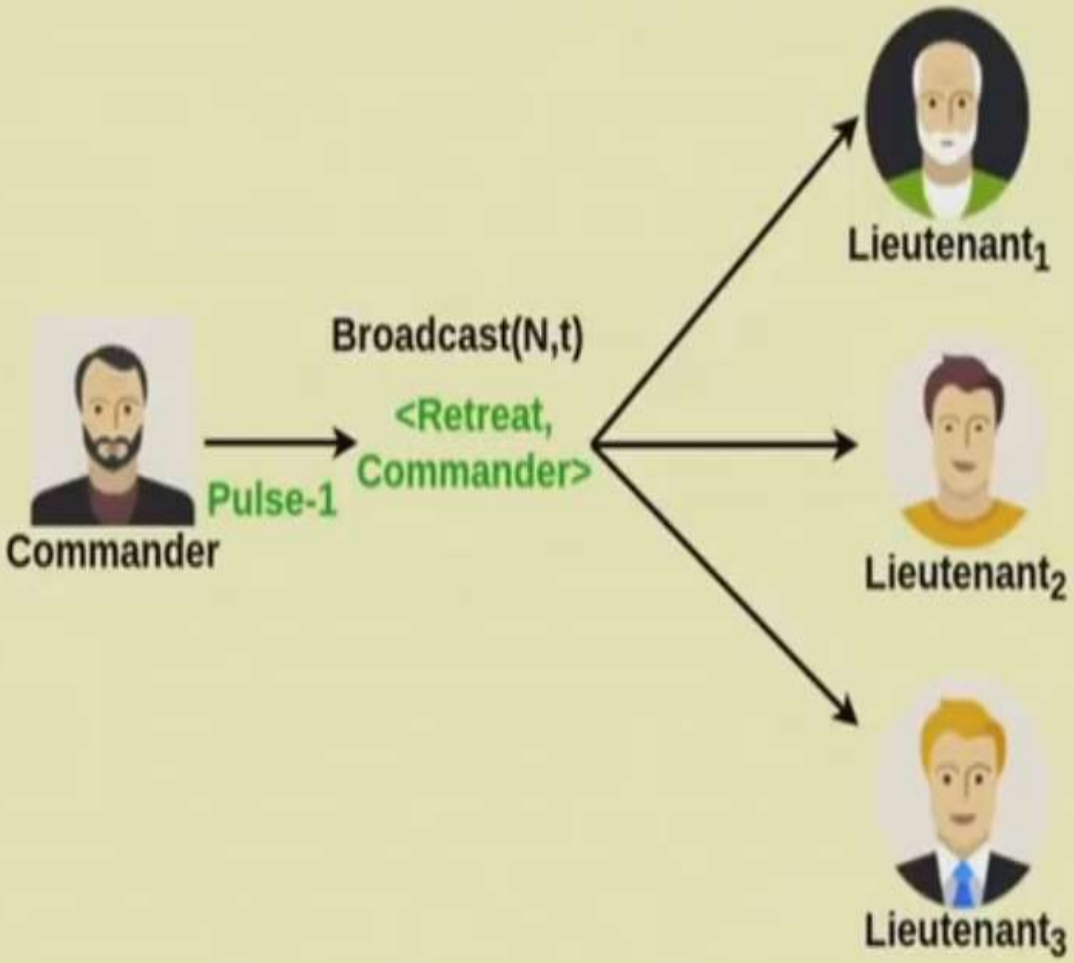
- **Base Condition:**
  Broadcast(N,t=0)
  - N: number of processes
  - t: algorithm parameter
- Commander decides on its own value

Lamport-Shostak-Pease Algorithm



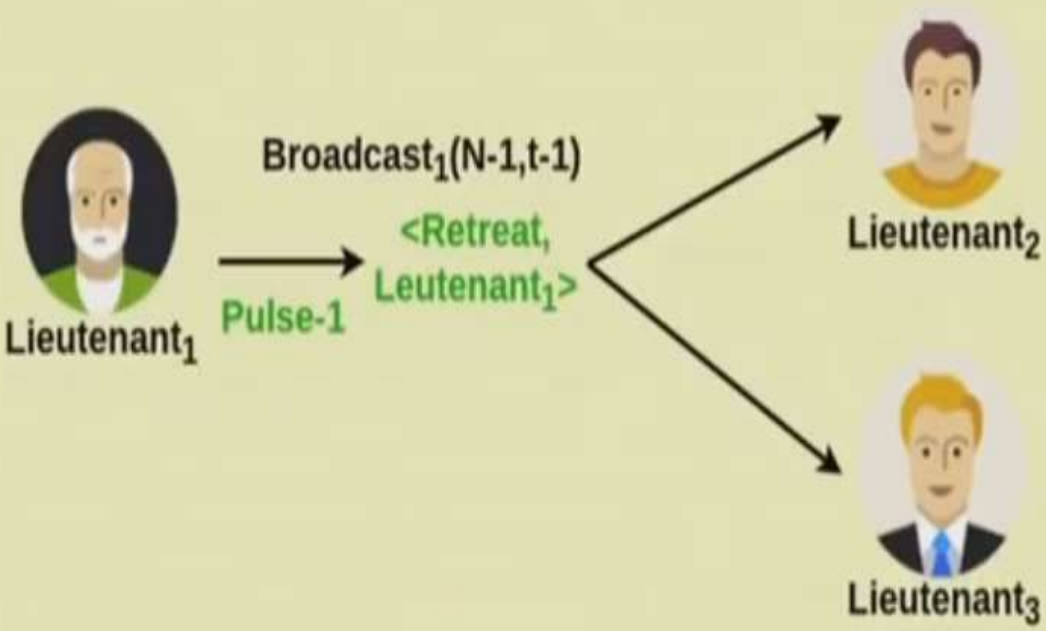
- **Base Condition:**
  Broadcast(N,t=0)
  - N: number of processes
  - t: algorithm parameter
- Lieutenants decision by sender matching

Lamport-Shostak-Pease Algorithm



- **General Condition:**
  Broadcast(N,t)
  - N: number of processes
  - t: algorithm parameter
- Only commander sends to all lieutenants

Lamport-Shostak-Pease Algorithm



- **General Condition:**
  Broadcast(N,t)
  - N: number of processes
  - t: algorithm parameter
- All lieutenants broadcast their values to the other lieutenants except the senders

THANK YOU