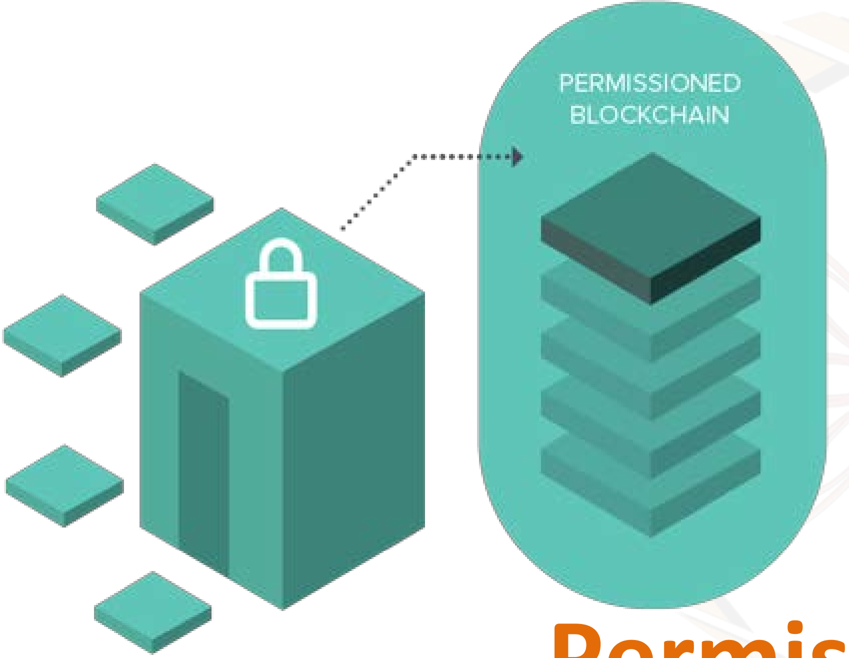
BLOCKCHAINS

ARCHITECTURE, DESIGN AND USE CASES

SANDIP CHAKRABORTY
COMPUTER SCIENCE AND ENGINEERING,
IIT KHARAGPUR

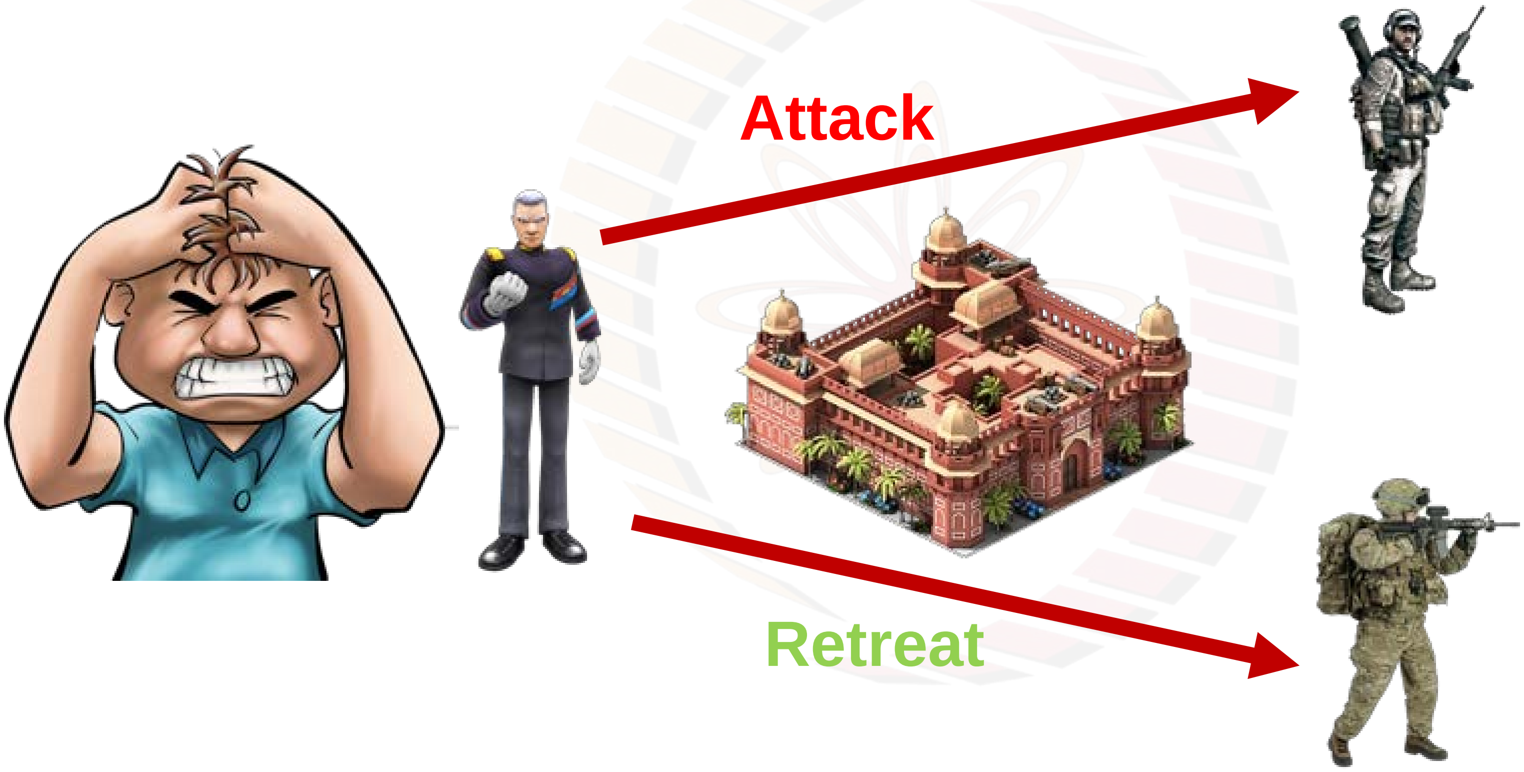
PRAVEEN JAYACHANDRAN
IBM RESEARCH,
INDIA

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

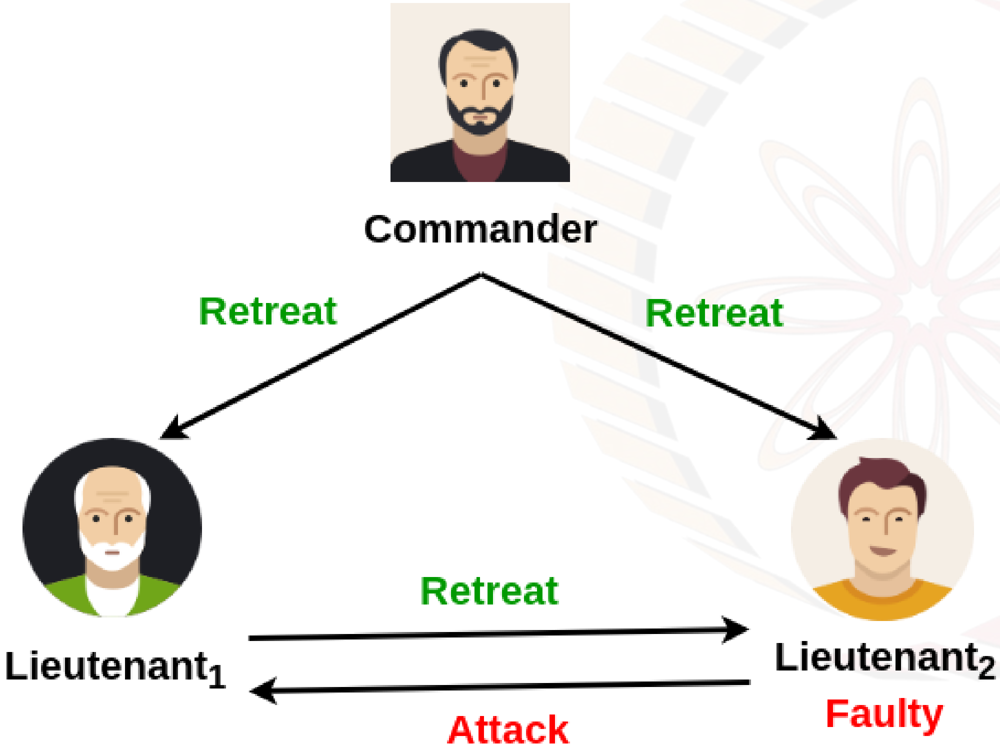


Permissioned Blockchain - IV
Consensus Algorithms

Byzantine Generals Problem

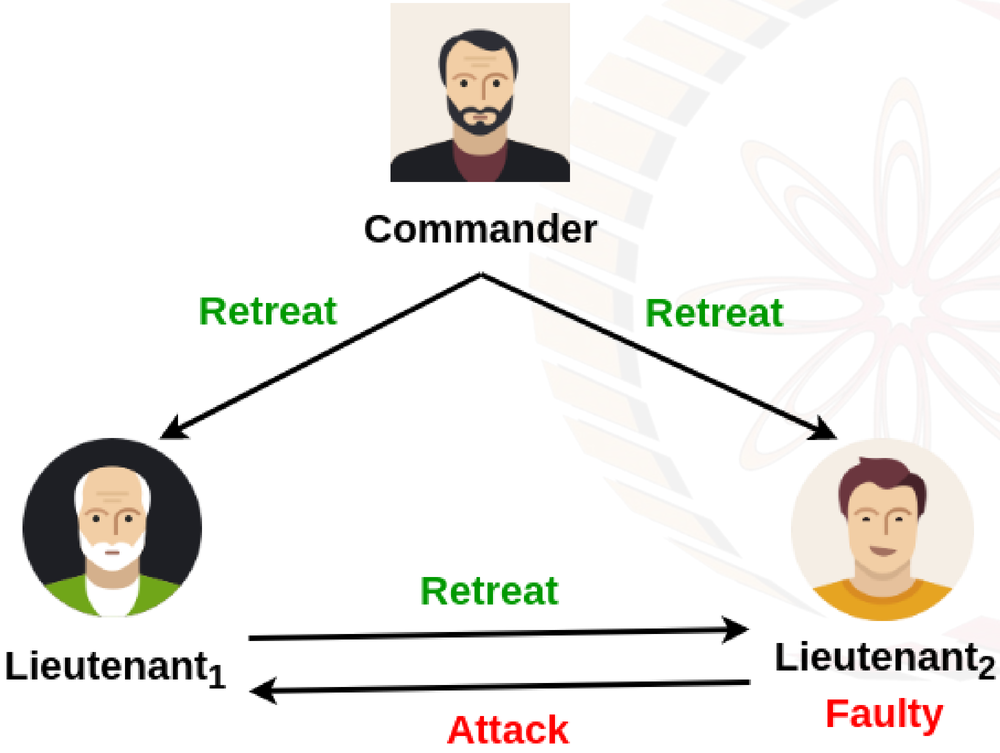


Three Byzantine Generals Problem: Lieutenant Faulty



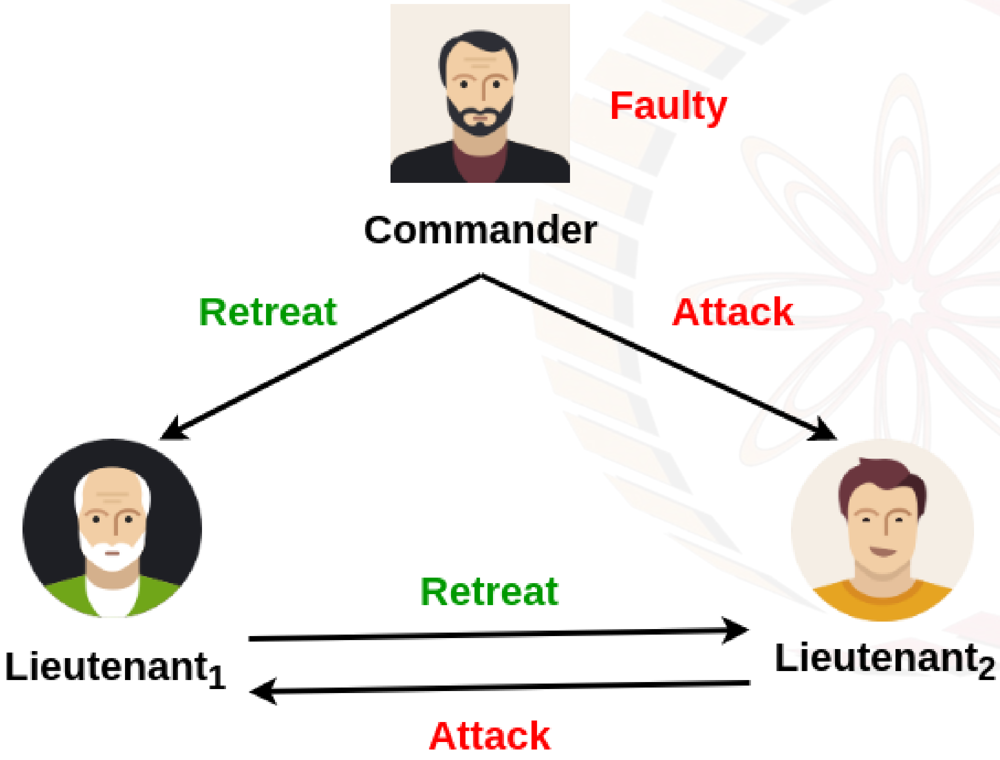
- Round1:
  - Commander correctly sends same message to Lieutenants
- Round 2:
  - Lieutenant$_1$ correctly echoes to Lieutenant$_2$
  - Lieutenant$_2$ **incorrectly** echoes to Lieutenant$_1$

Three Byzantine Generals Problem: Lieutenant Faulty



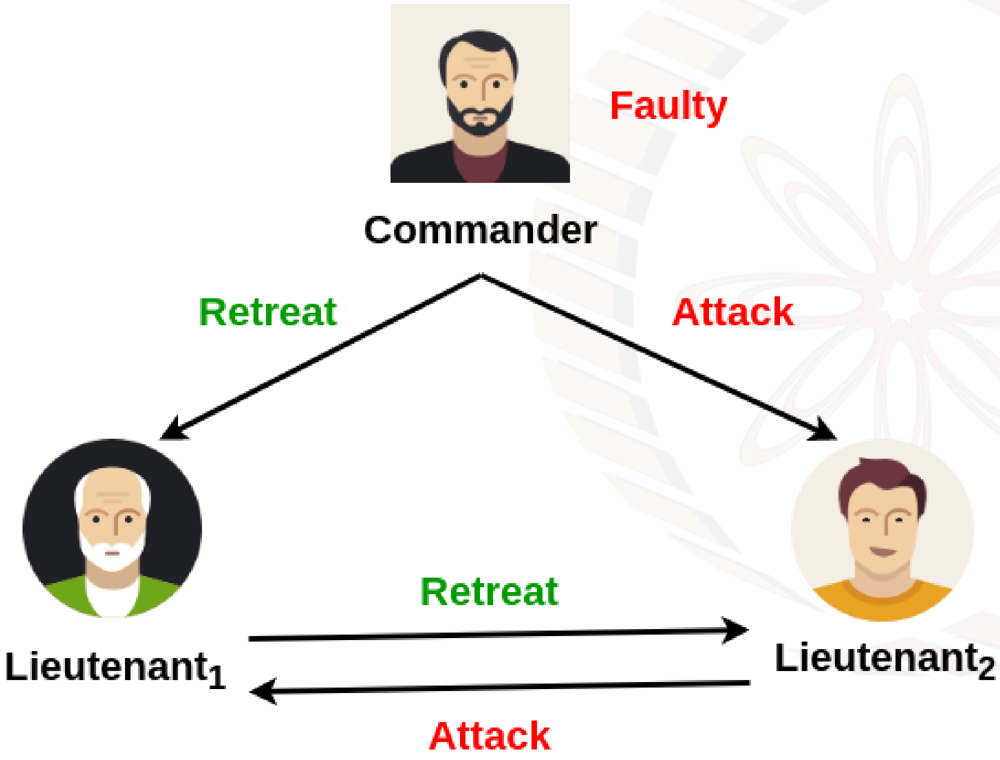
- Lieutenant$_1$ received differing message
- By integrity condition, Lieutenant$_1$ bound to decide on Commander message
- What if Commander is faulty??

Three Byzantine Generals Problem: Commander Faulty



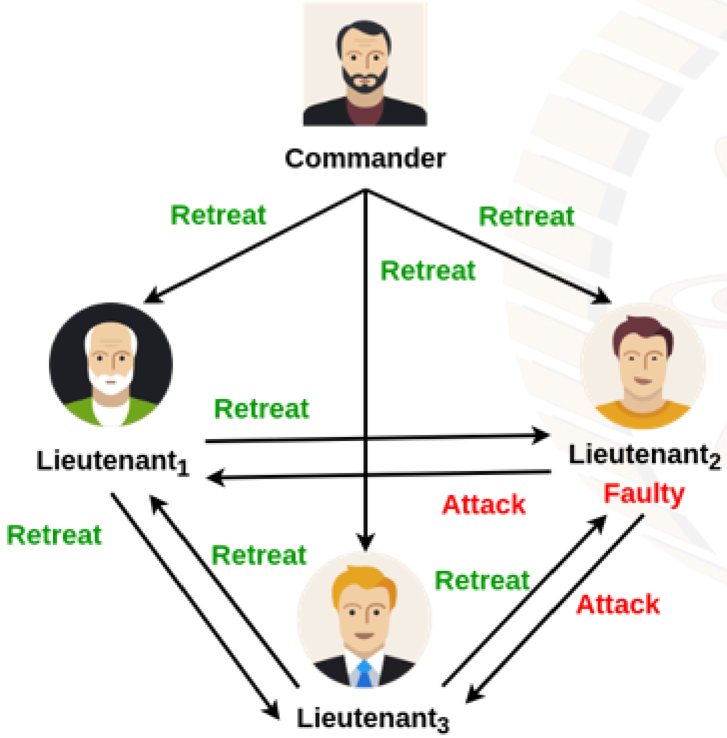
- Round 1:
  - Commander sends differing message to Lieutenants
- Round 2:
  - Lieutenant$_1$ correctly echoes to Lieutenant$_2$
  - Lieutenant$_2$ correctly echoes to Lieutenant$_1$

Three Byzantine Generals Problem: Commander Faulty



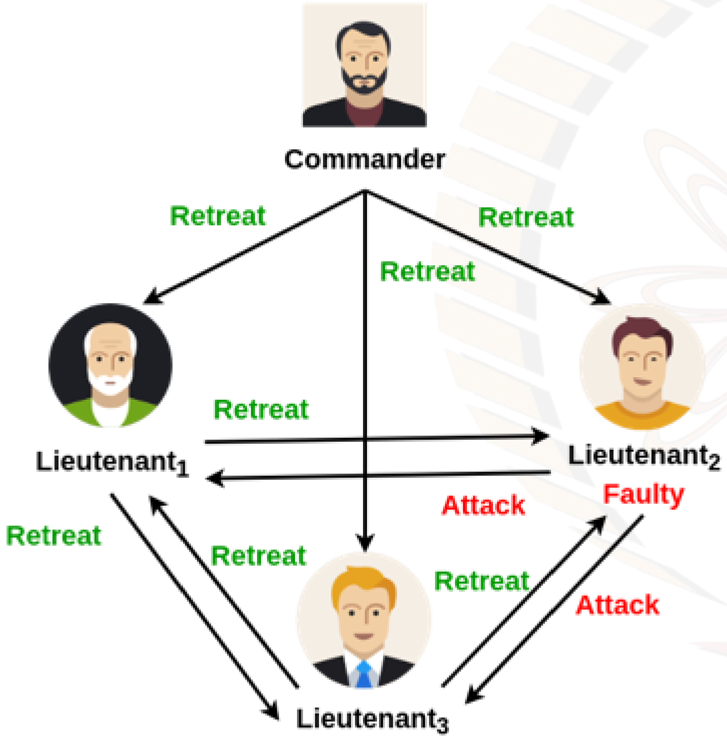
- Lieutenant$_1$ received **differing message**
- By integrity condition, both Lieutenants conclude with Commander's message
- This contradicts the agreement condition
- No solution possible for three generals including one faulty

Four Byzantine Generals Problem: Lieutenant Faulty



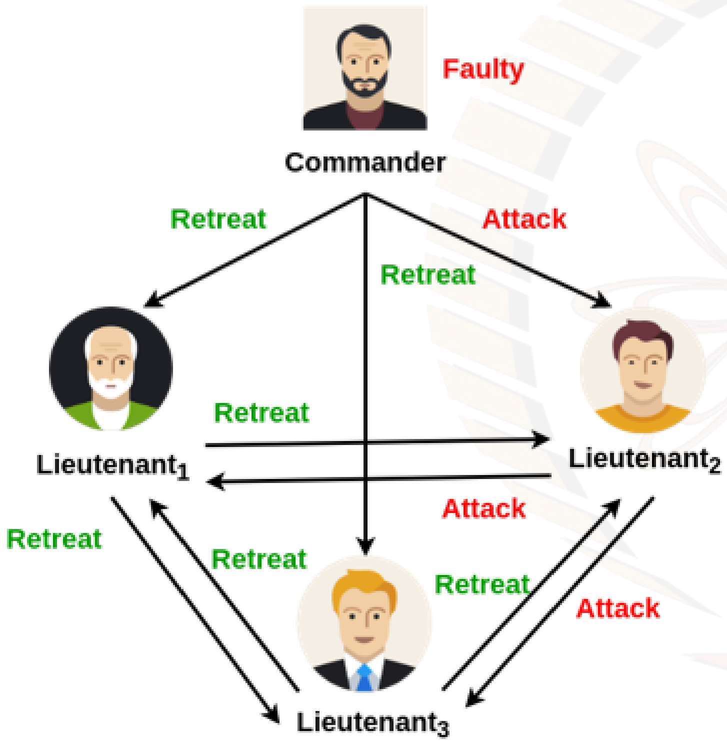
- Round 1:
  - Commander sends a message to each of the Lieutenants
- Round 2:
  - Lieutenant$_1$ and Lieutenant$_3$ correctly echo the message to others
  - Lieutenant$_2$ **incorrectly** echoes to others

Four Byzantine Generals Problem: Lieutenant Faulty



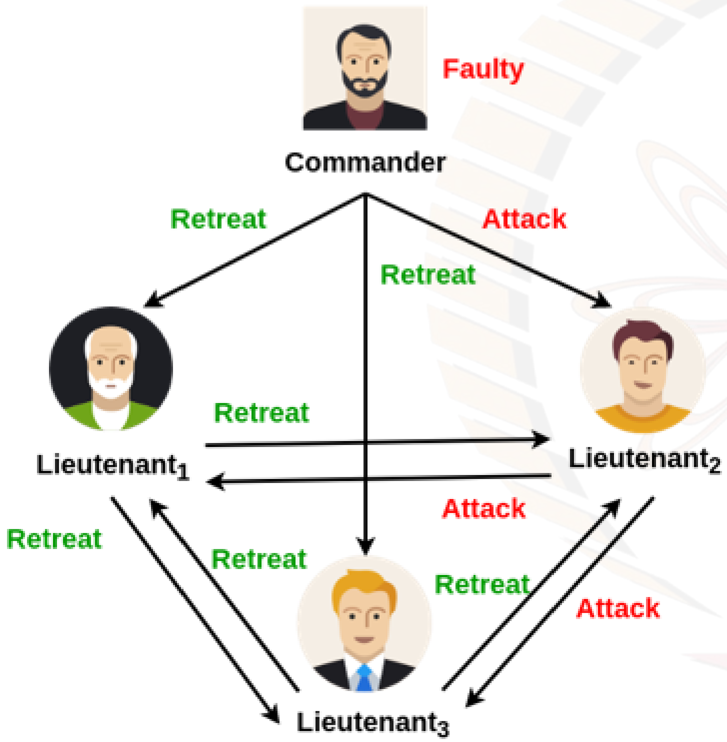
- Lieutenant$_1$ decides on *majority*(Retreat,Attack,Retreat)= Retreat
- Lieutenant$_3$ decides on *majority*(Retreat,Retreat,Attack)= Retreat

Four Byzantine Generals Problem: Commander Faulty



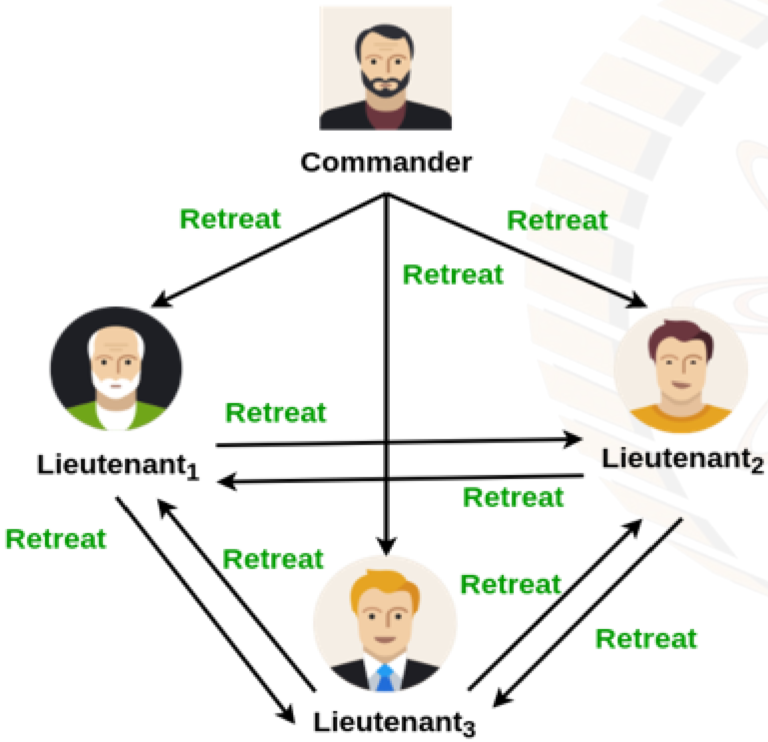
- Round 1:
  - Commander sends differing message to Lieutenants
- Round 2:
  - Lieutenant$_1$, Lieutenant$_2$ and Lieutenant$_3$ correctly echo the message to others

Four Byzantine Generals Problem: Commander Faulty



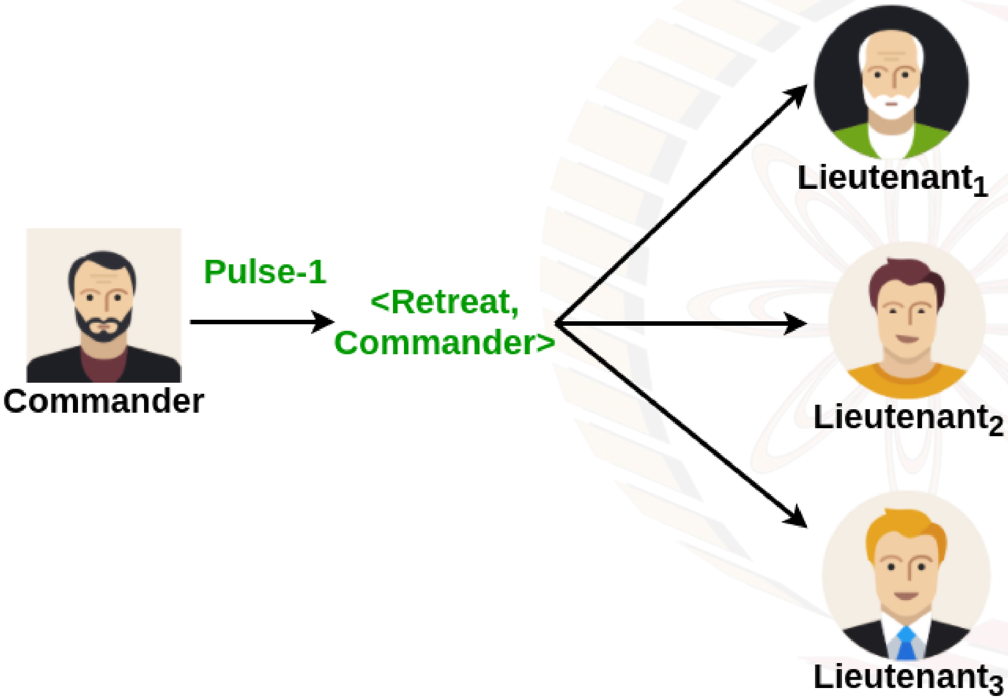
- Lieutenant$_1$ decides on *majority*(Retreat,Attack,Retreat)= Retreat
- Lieutenant$_2$ decides on *majority*(Attack,Retreat,Retreat)= Retreat
- Lieutenant$_3$ decides on *majority*(Retreat,Retreat,Attack)= Retreat

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

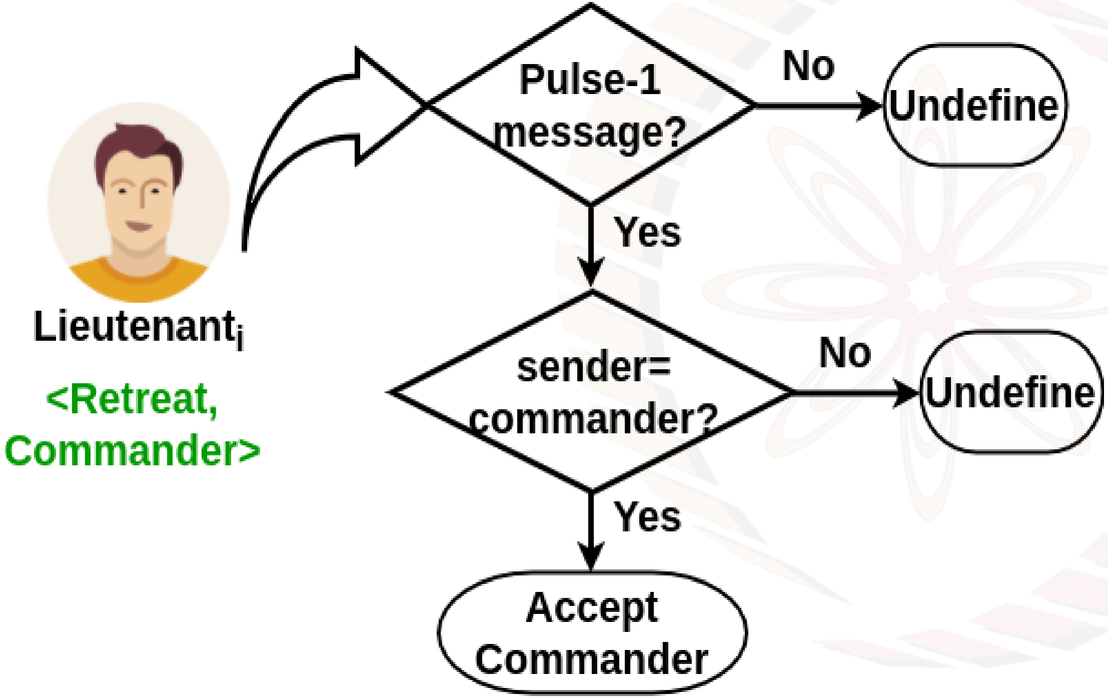
Lamport-Shostak-Pease Algorithm



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

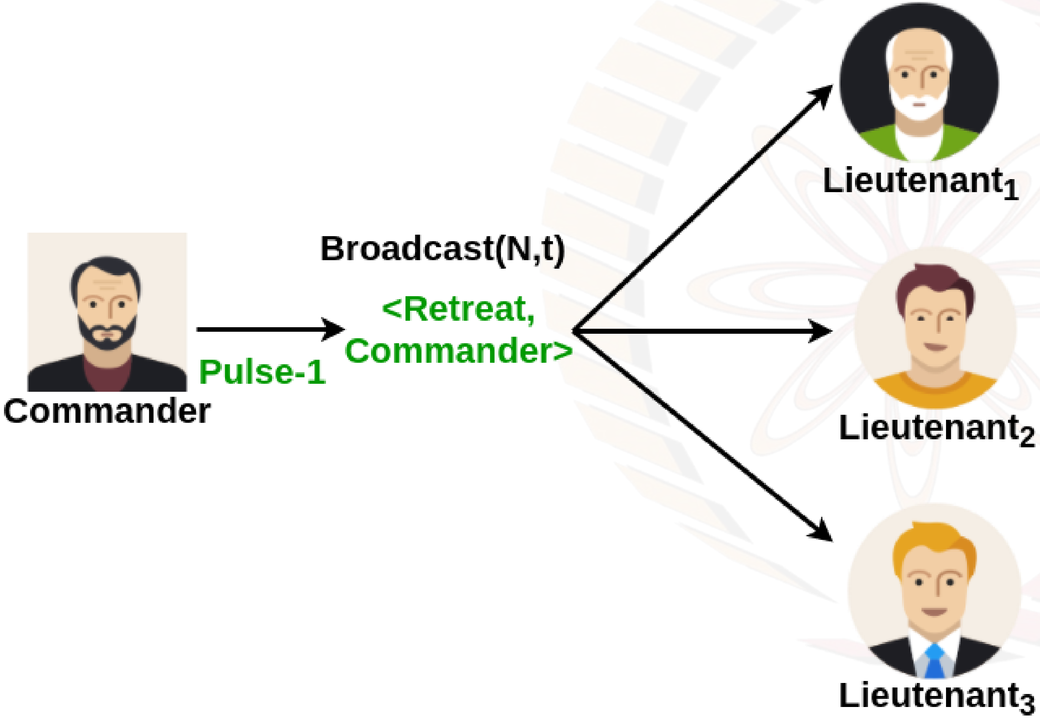
Source: Lamport, Leslie, Robert Shostak, and Marshall Pease. "The Byzantine generals problem." ACM Transactions on Programming Languages and Systems (TOPLAS) 4.3 (1982): 382-401.

Lamport-Shostak-Pease Algorithm



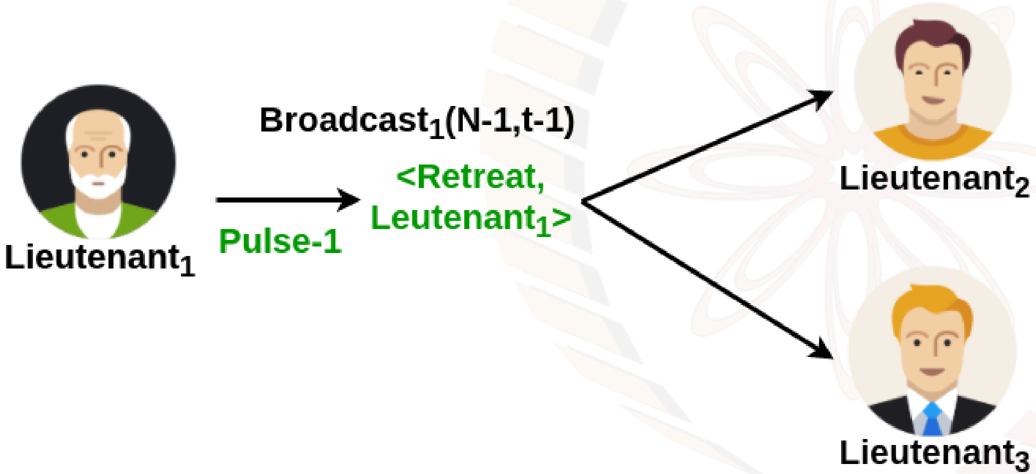
- Base Condition: Broadcast(N,t=0)
  - N: number of processes
  - t: algorithm parameter
- Lieutenants decision by sender matching

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

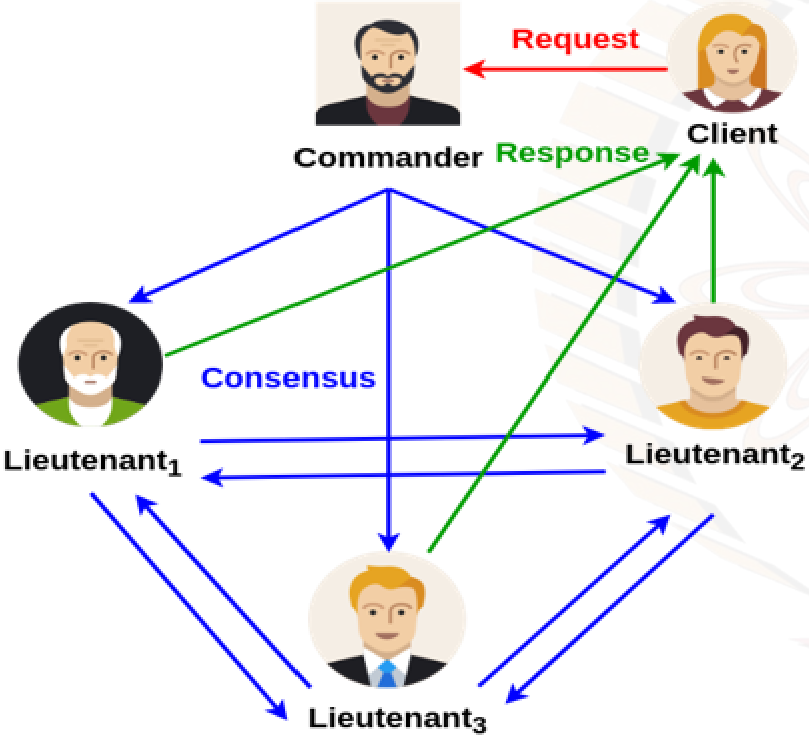


- General Condition: Broadcast(N,t)
  - N: number of processes
  - t: algorithm parameter
- All lieutenants broadcast their values to the other lieutenants except the senders

Practical Byzantine Fault Tolerant

- Why Practical?
  - Ensures safety over an asynchronous network (not liveness!)
  - Byzantine Failure
  - Low overhead
- Real Applications
  - Tendermint
  - IBM's Openchain
  - ErisDB
  - Hyperledger

Practical Byzantine Fault Tolerant Model



- Asynchronous distributed system
  - delay, out of order message
- Byzantine failure handling
  - arbitrary node behavior
- Privacy
  - tamper-proof message, authentication

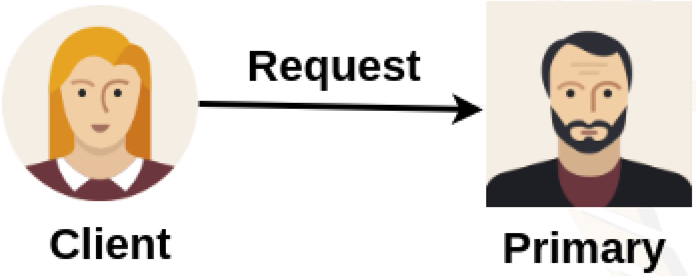
Practical Byzantine Fault Tolerant Model

- A state machine is replicated across different nodes
- $3f + 1$ replicas are there where f is the number of faulty replicas
- The replicas move through a successions of configurations, known as *views*
- One replica in a *view* is *primary* and others are *backups*

Practical Byzantine Fault Tolerant Model

- Views are changed when a *primary* is detected as faulty
- Every view is identified by a unique integer number v
- Only the messages from the current views are accepted

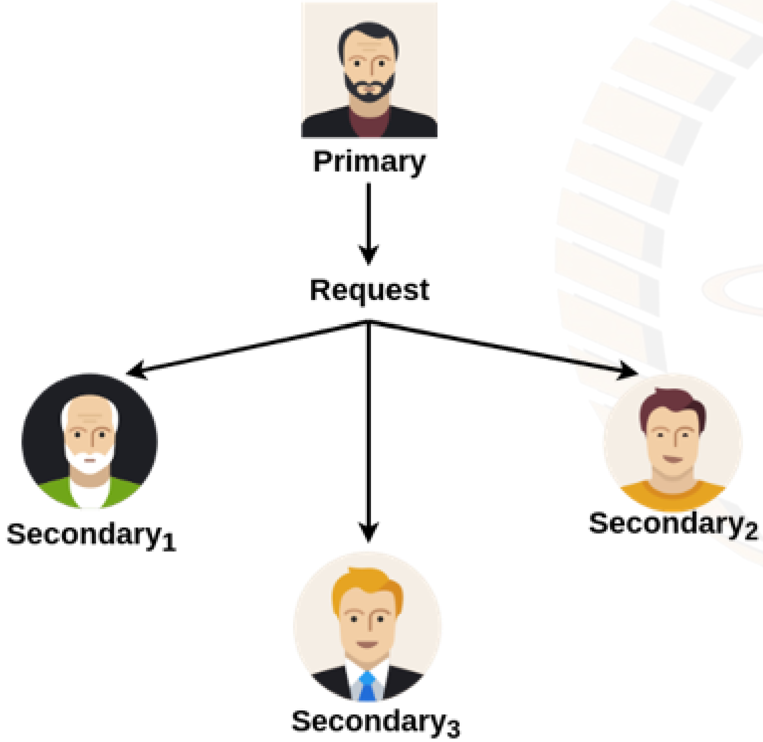
Practical Byzantine Fault Tolerant Algorithm



- A client sends a request to invoke a service operation to the primary

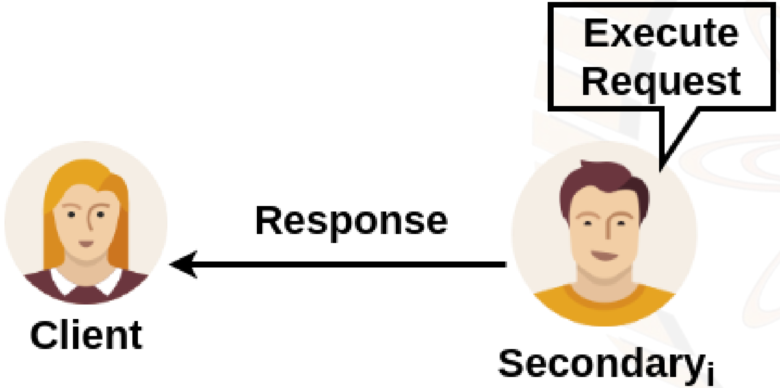
Castro, Miguel, and Barbara Liskov. "Practical Byzantine fault tolerance." OSDI. Vol. 99. 1999.

Practical Byzantine Fault Tolerant Algorithm



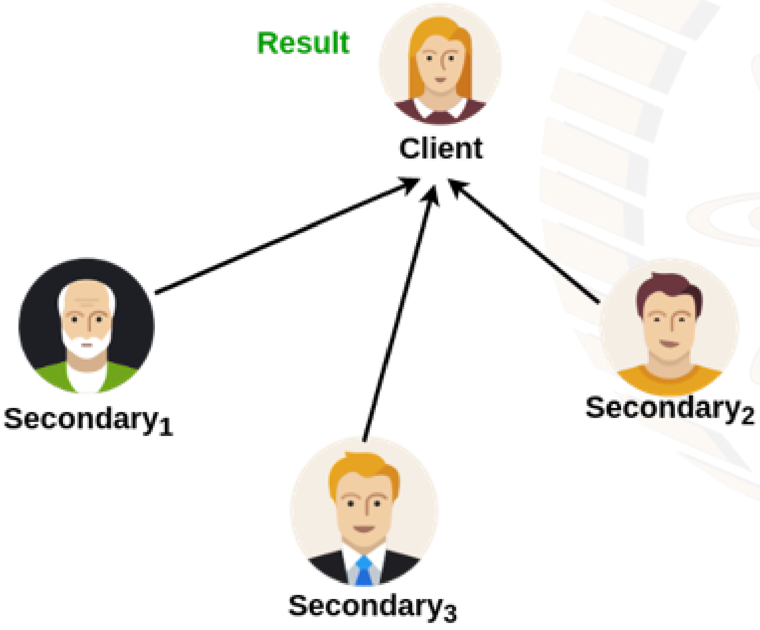
- The primary multicasts the request to the backups

Practical Byzantine Fault Tolerant Algorithm



- Backups execute the request and send a reply to the client

Practical Byzantine Fault Tolerant Algorithm

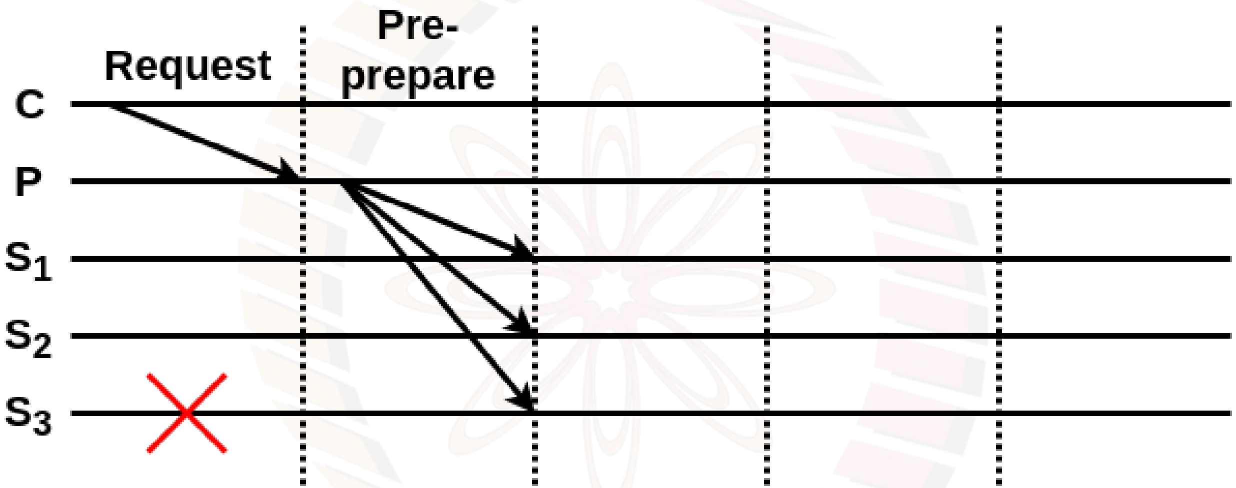


- The client waits for $f + 1$ replies from different backups with the same result
  - f is the maximum number of faulty replicas that can be tolerated

Three Phase Commit Protocol - Pre-Prepare

- **Pre-prepare:** Primary assigns a sequence number n to the request and multicast a message $<< PRE-PREPARE, v, n, d >_{\sigma\_p}, m >$ to all the backups
  - v is the current view number
  - n is the message sequence number
  - d is the message digest
  - σ_p is the private key of primary - works as a digital signature
  - m is the message to transmit

Three Phase Protocol

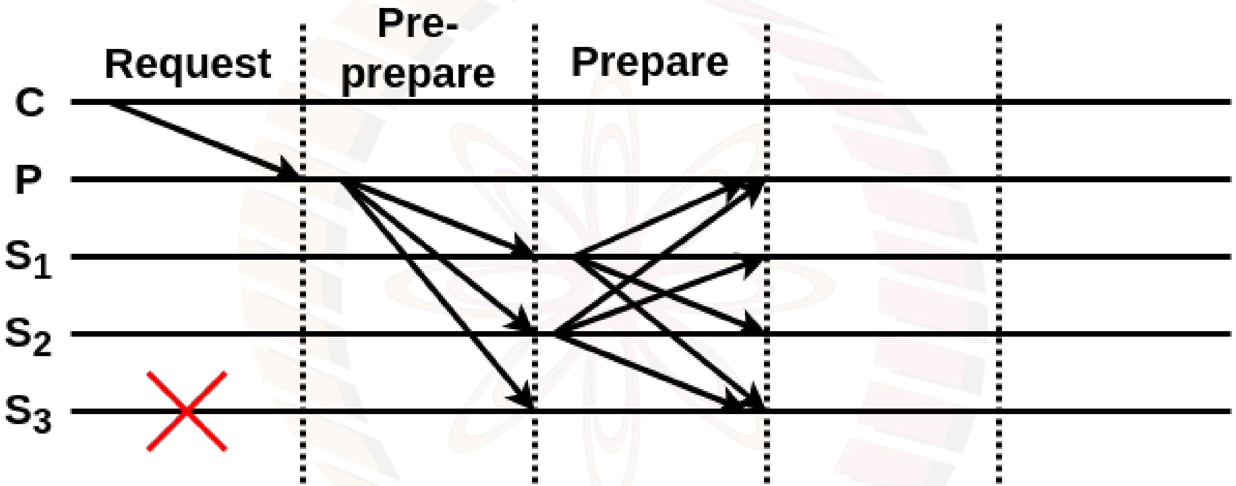


- Pre-prepare:
  - Acknowledge the request by a unique sequence number

Three Phase Commit Protocol - Pre-Prepare

- Pre-prepare messages are used as a proof that request was assigned sequence number n is the view v
- A backup accepts a pre-prepare message if
  - The signature is correct and d is the digest for m
  - The backup is in view v
  - It has not received a different PRE-PREPARE message with sequence n and view v with a different digest
  - The sequence number is within a threshold

Three Phase Protocol



- Prepare:
  – Replicas agree on the assigned sequence number

Three Phase Commit Protocol - Prepare

- If the backup accepts the PRE-PREPARE message, it enters prepare phase by multicasting a message $< PREPARE, v, n, d, i >_{\sigma\_i}$ to all other replicas

- A replica (both primary and backups) accepts prepare messages if
  - Signatures are correct
  - View number equals to the current view
  - Sequence number is within a threshold

Three Phase Commit Protocol

- Pre-prepare and prepare ensure that non-faulty replicas guarantee on a total order for the requests within a view

- Commit a message if
  - $2f$ prepares from different backups matches with the corresponding pre-prepare
  - You have total $2f + 1$ votes (one from primary that you already have!) from the non-faulty replicas

Three Phase Commit Protocol

- Why do you require $\boldsymbol{3f+1}$ replicas to ensure safety in an asynchronous system when there are $\boldsymbol{f}$ faulty nodes?
  - If you have $2f+1$ replicas, you need all the votes to decide the majority - boils down to a synchronous system
  - You may not receive votes from certain replicas due to delay, in case of an asynchronous system
  - $f+1$ votes do not ensure majority, may be you have received f votes from Byzantine nodes, and just one vote from a non-faulty node (note Byzantine nodes can vote for or against - You do not know that a priori!)

Three Phase Commit Protocol

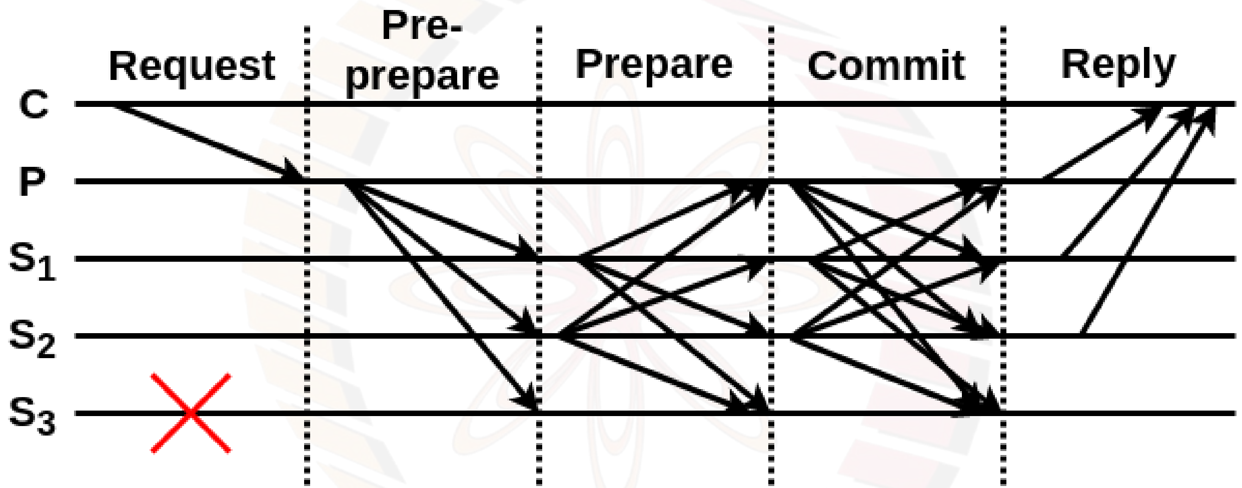
- Why do you require $\boldsymbol{3f+1}$ replicas to ensure safety in an asynchronous system when there are $\boldsymbol{f}$ faulty nodes?
  - If you do not receive a vote
    - The node is faulty and not forwarded a vote at all
    - The node is non-faulty, forwarded a vote, but the vote got delayed
  - Majority can be decided once $2f+1$ votes have arrived - even if f are faulty, you know $f+1$ are from correct nodes, do not care about the remaining f votes

Three Phase Commit Protocol - Commit

- Multicast $< COMMIT, v, n, d, i >_{\sigma\_i}$ message to all the replicas including primary

- Commit a message when a replica
  - Has sent a commit message itself
  - Has received $2f + 1$ commits (including its own)

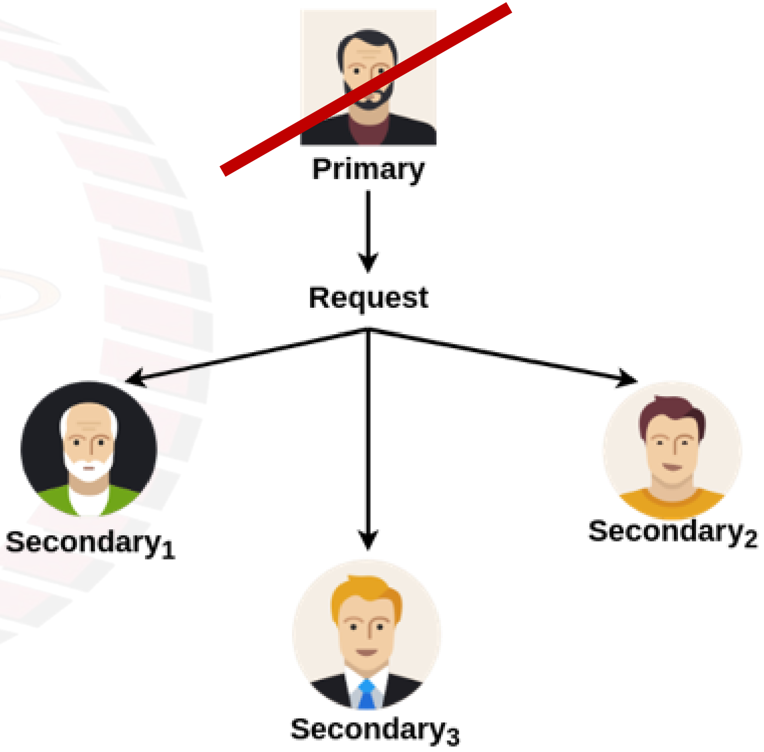
Three Phase Protocol



- Commit:
  - Establish consensus throughout the views

View Change

- What if the **primary** is **faulty**??
  - non-faulty replicas detect the fault
  - replicas together start view change operation



View Changes

- View-change protocol provides **liveness**
  - Allow the system to make progress when primary fails

- If the primary fails, backups will not receive any message (such as PRE_PREPARE or COMMIT) from the primary

- View changes are triggered by timeouts
  - Prevent backups from waiting indefinitely for requests to execute

View Change

- Backup starts a timer when it receives a request, and the timer is not already running
  - The timer is stopped when the request is executed
  - Restarts when some new request comes

- If the timer expires at view v
  - Backup starts a view change to move the system to view $v + 1$

View Change

- On timer expiry, a backup stops accepting messages except
  - Checkpoint
  - View-change
  - New-View

View Change

- Multicasts a $< VIEW\_CHANGE, v+1, n, \mathcal{C}, \mathcal{P}, i >_{,\sigma\_i}$ message to all replicas
  - n is the sequence number of the last stable checkpoint s known to i
  - $\mathcal{C}$ is a set of $2f+1$ valid checkpoint messages proving the correctness of s
  - $\mathcal{P}$ is a set containing a set $\mathcal{P}_m$ for each request m that prepared at i with a sequence number higher than n
    - Each set $\mathcal{P}_m$ contains a valid pre-prepare message and $2f$ matching

View Change

- The new view is initiated after receiving $2f$ view change messages

- The view change operation takes care of
  - Synchronization of checkpoints across the replicas
  - All the replicas are ready to start at the new view $v + 1$

Correctness

- **Safety:** The algorithm provides safety if all non-faulty replicas agree on the sequence numbers of requests that commit locally

Image Source: http://www.differencebetween.com/

Correctness

Liveness: To provide liveness, replicas must move to a new view if they are unable to execute a request

- A replica waits for $2f + 1$ view change messages and then starts a timer to initiate a new view (*avoid starting a view change too soon*)
- If a replica receives a set of $f + 1$ valid view change messages for views greater than its current view, it sends view change message (*prevents starting the next view change too late*)
- Faulty replicas are unable to impede progress by forcing frequent view change

For further details, look into

Castro, Miguel, and Barbara Liskov. "Practical Byzantine fault tolerance." *OSDI*. Vol. 99. 1999.

Consensus in Permissioned Model

- PBFT has well adopted in consensus for permissioned blockchain environments
  - Hyperledger
  - Tendermint Core

- Several scalability issues are still there, we'll discuss those in details in the later part of the course!

thank you!