Blockchain and its applications

Prof. Sandip Chakraborty

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Lecture 26: Consensus for Permissioned Models

CONCEPTS COVERED

- Permissioned Model
- State Machine Replication

KEYWORDS

- **Consensus on State Machines**

Permissioned Model

- A blockchain architecture where users are authenticated a priori
  - A Membership Service Provider (MSP) helps to obtain the chain membership
- Users know each other
  - However, users may not trust each other
  - Security and consensus are still required.
- Run blockchain among known and identified participants

Permissioned Model

- Particularly interesting for business applications – execute contracts among a closed set of participants

- Example: Provenance tracking of assets in a supply chain

A Use Case

Smart Contracts over Closed Networks

- **Smart Contracts:** "A self-executing contract in which the terms of the agreement between the buyer and the seller is directly written into the lines of code" - http://www.scalablockchain.com/

Smart Contracts over Closed Networks

- **Agreement on a Smart Contract Execution:**
  - Store the contract on a blockchain
  - Once an event is triggered, execute the codes locally on each peer
  - Generate transactions as the output of the contract execution
  - The peers of the blockchain network validates the transaction, and the output is committed in the blockchain – may trigger the next event to execute the code further

Smart Contracts over Closed Networks

- **Agreement on a Smart Contract Execution:**
  - Store the contract on a blockchain
  - Once an event is triggered, execute the codes locally on each peer
  - Generate transactions as t[...]e output of the contract exe[...]
  - The[...] tra[...] blo[...]e coo[...]

Do we really need to execute the code on each peer?

When does each peer execute the code?

Smart Contracts over Closed Networks

- Execute contract at a subset of nodes, and ensure that the same state is propagated to all the nodes
  - Majority of the peers should agree on the state
  - **Validation:** Generate a "***proof***" that a peer has agreed on the "*state of execution*"

Smart Contracts over Closed Networks

- Execute contract at a subset of nodes, and ensure that the same state is propagated to all the nodes
  - Majority of the peers should agree on the state
  - **Validation:** Generate a "***proof***" that a peer has agreed on the "*state of execution*"

How will we generate the proof?

Smart Contracts over Closed Networks

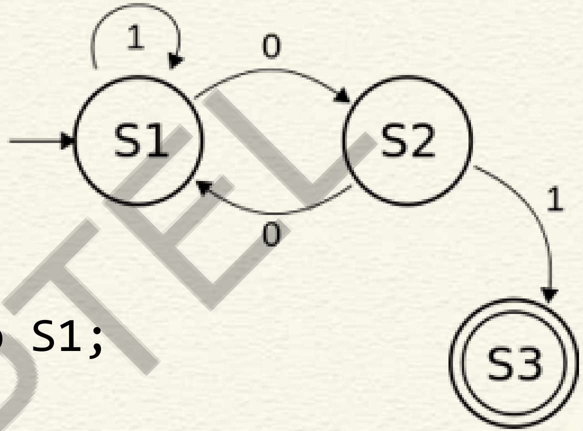
- Execute contract at a subset of nodes, and ensure that the same state is propagated to all the nodes
  - Majority of the peers should agree on the state
  - **Validation:** Generate a "***proof***" that a peer has agreed on the "*state of execution*"

Smart Contract as State Machines

- **State Machine Replication:**
  - Represent the smart contract as a state machine – Remember, any deterministically executable code can be represented as a state machine

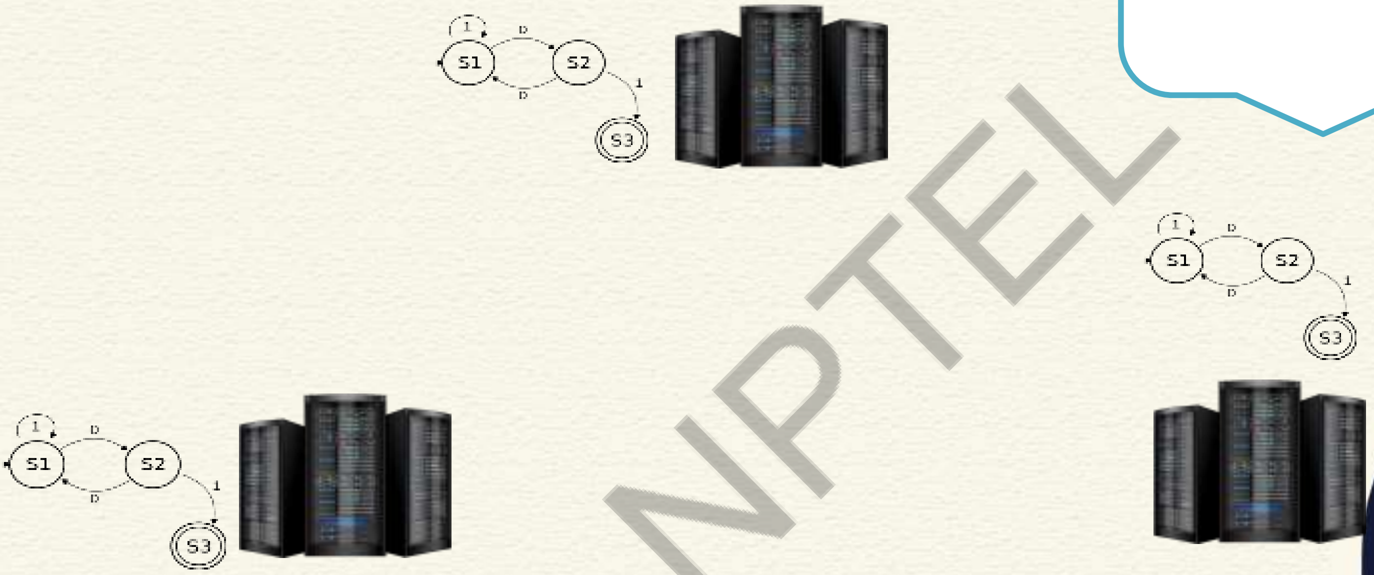
Smart Contract as State Machines

```
S1:
while (moreGoods == 1)
   DeliverGoods();
S2:
if (allOrderComplete == 0) goto S1;
else {
   S3:
   printf("Goods transfer complete");
}
```

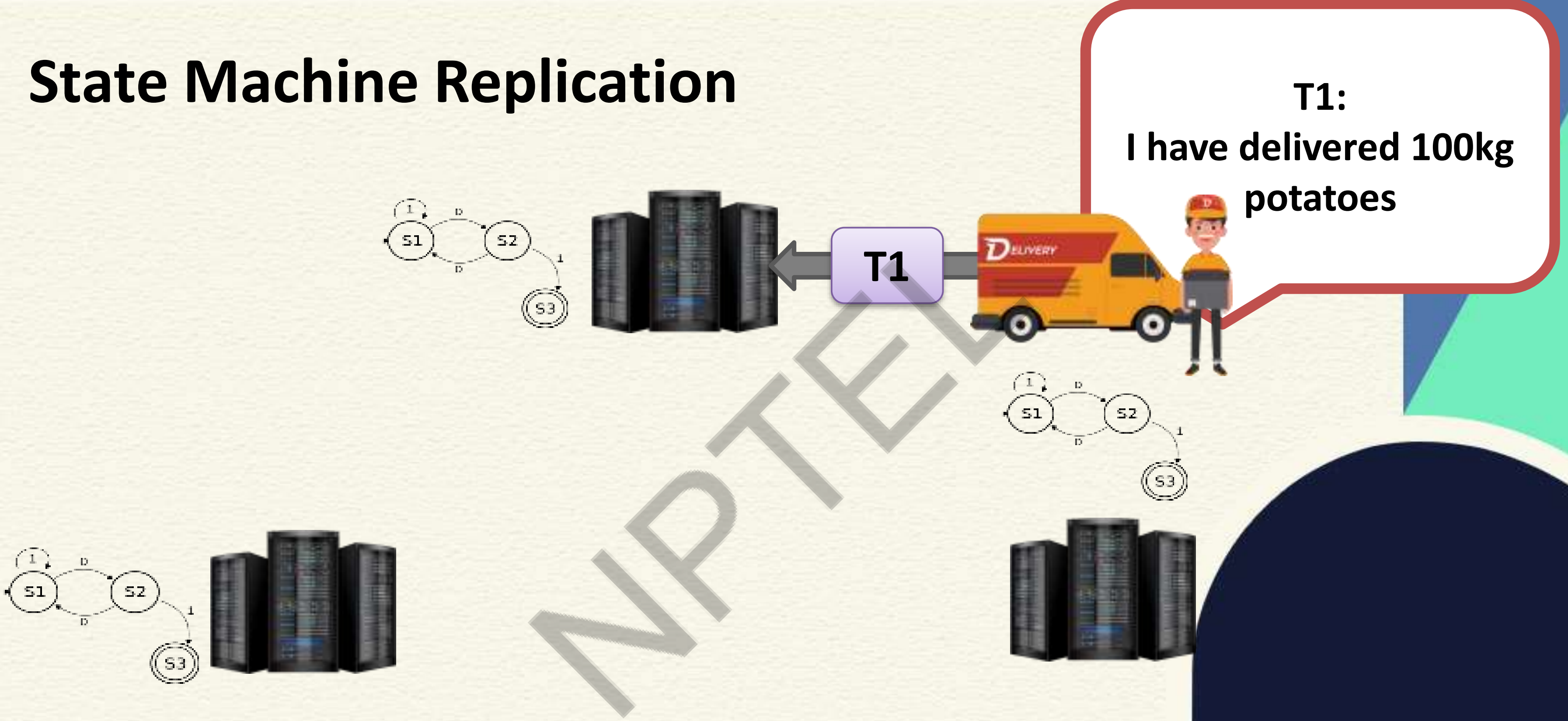


State Machine Replication

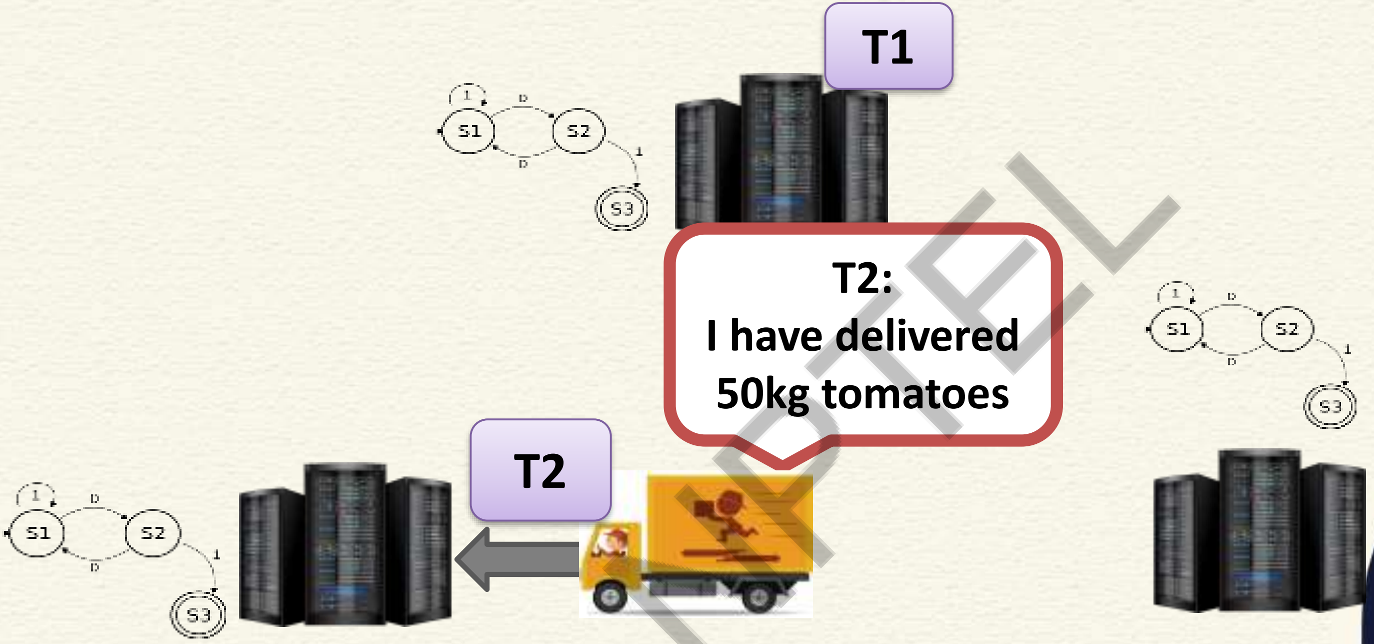
Replicate the state machine on multiple independent servers



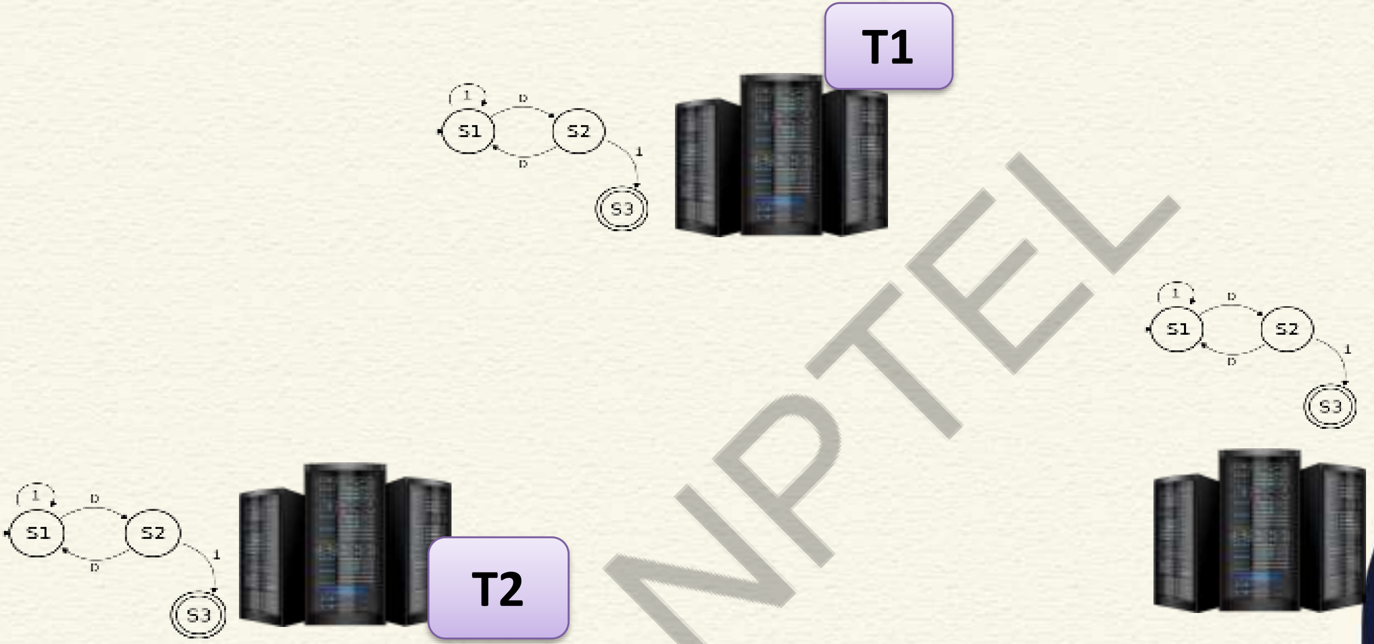
State Machine Replication



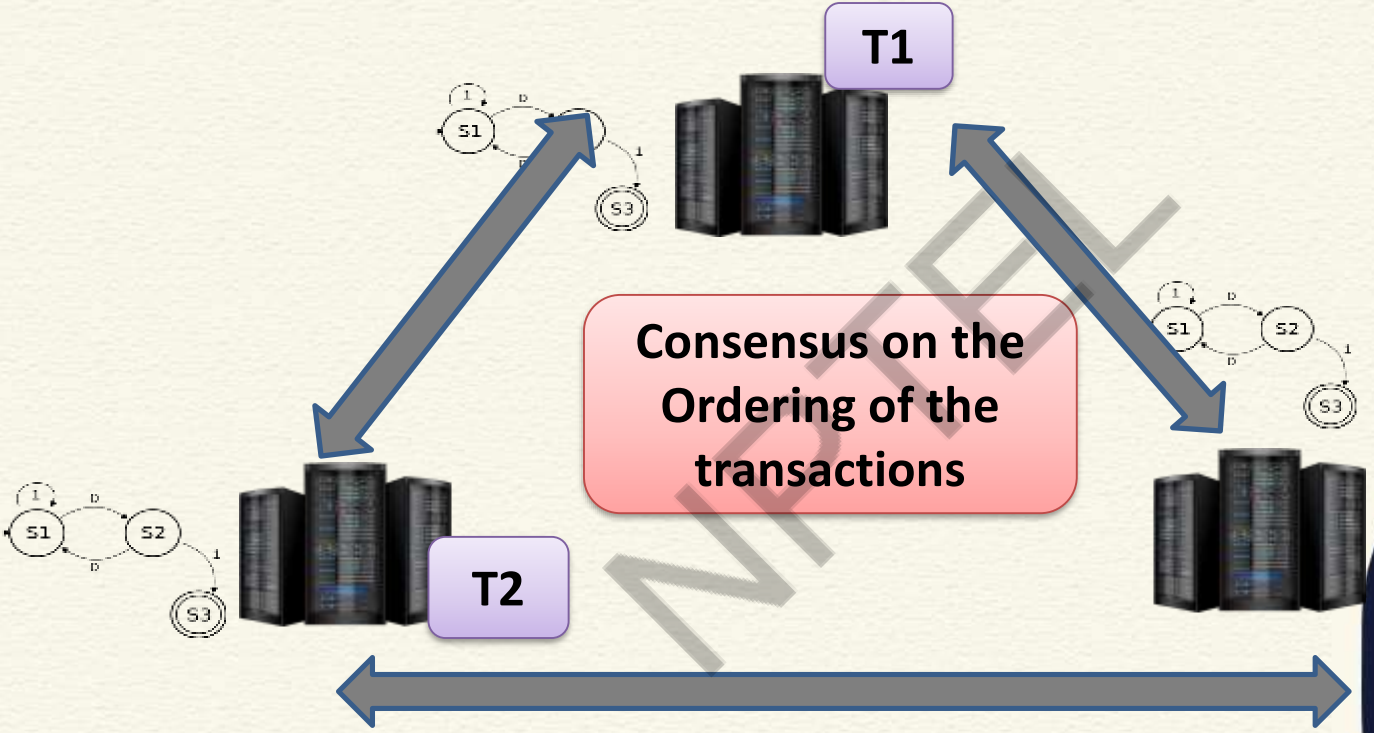
State Machine Replication



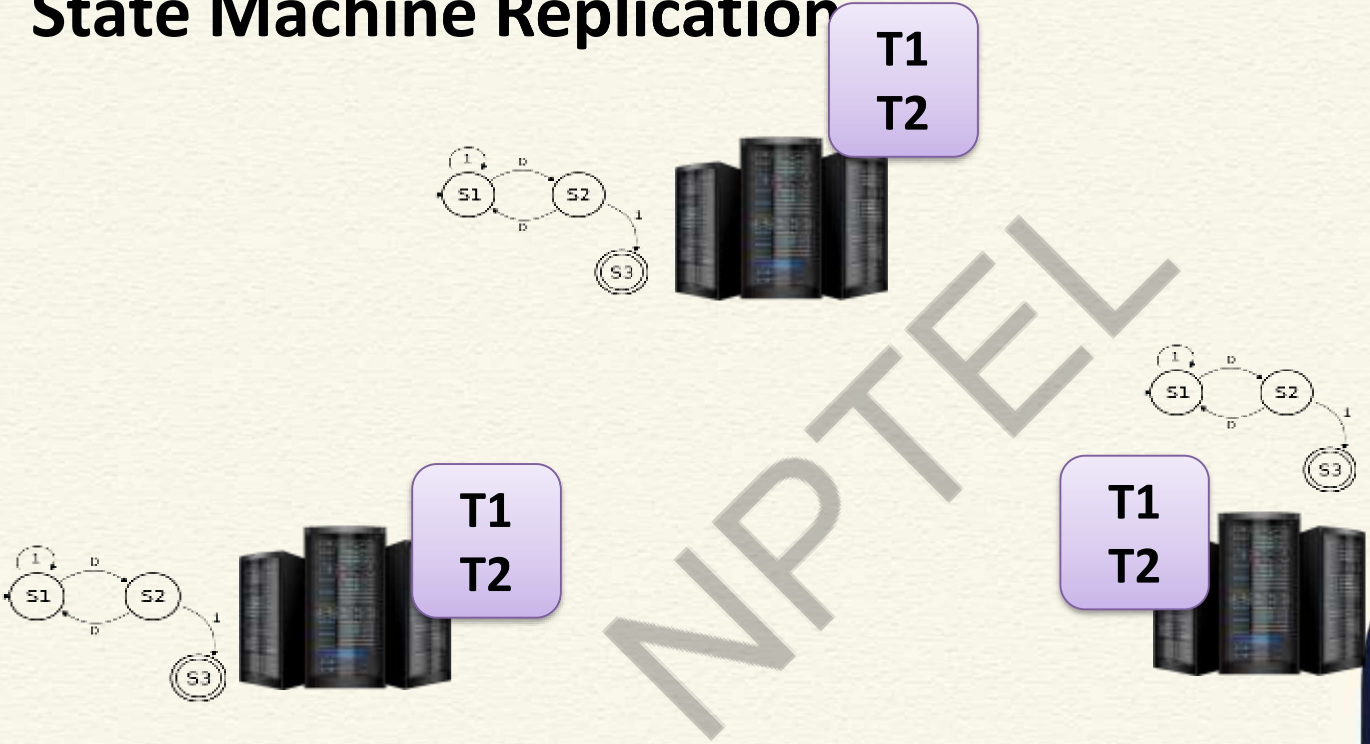
State Machine Replication



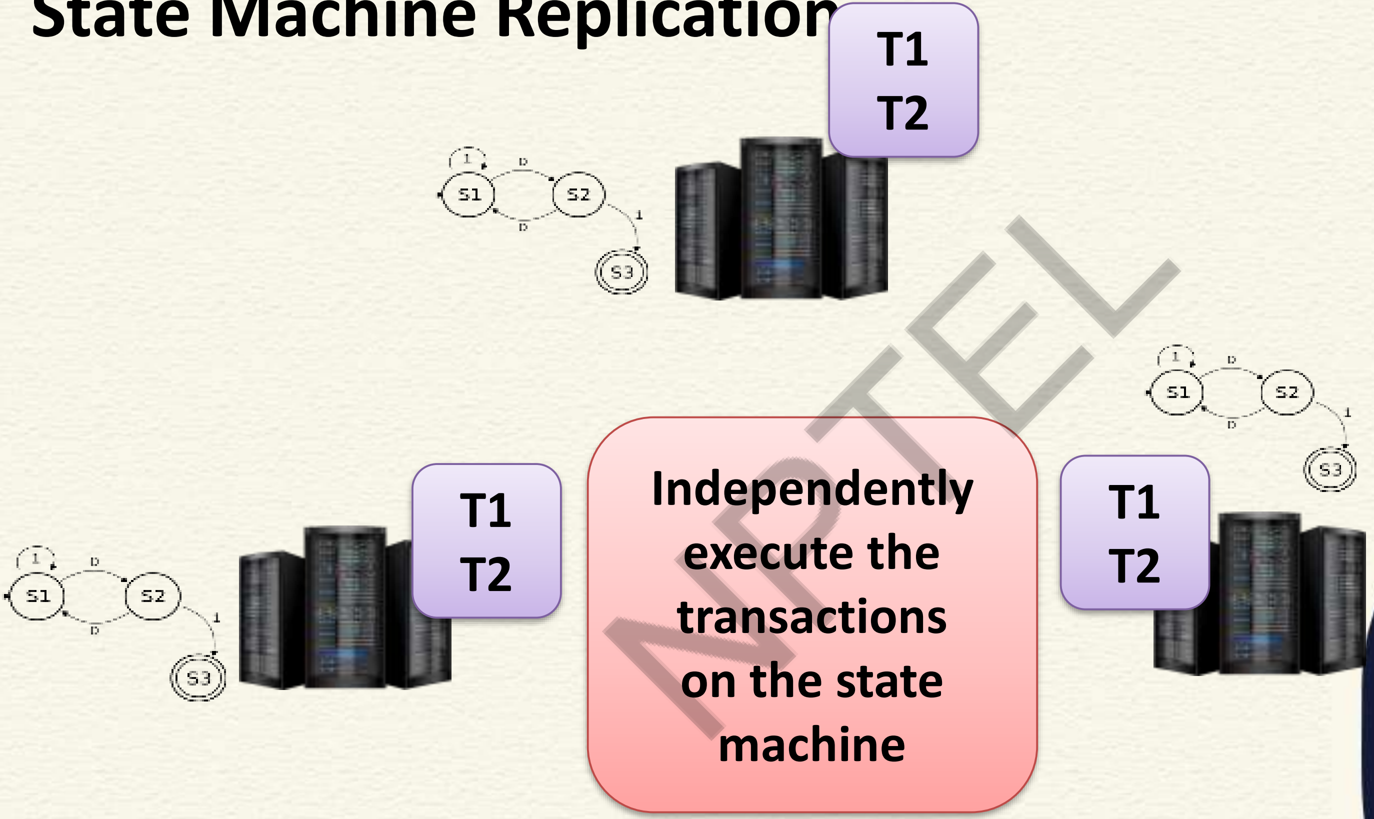
State Machine Replication



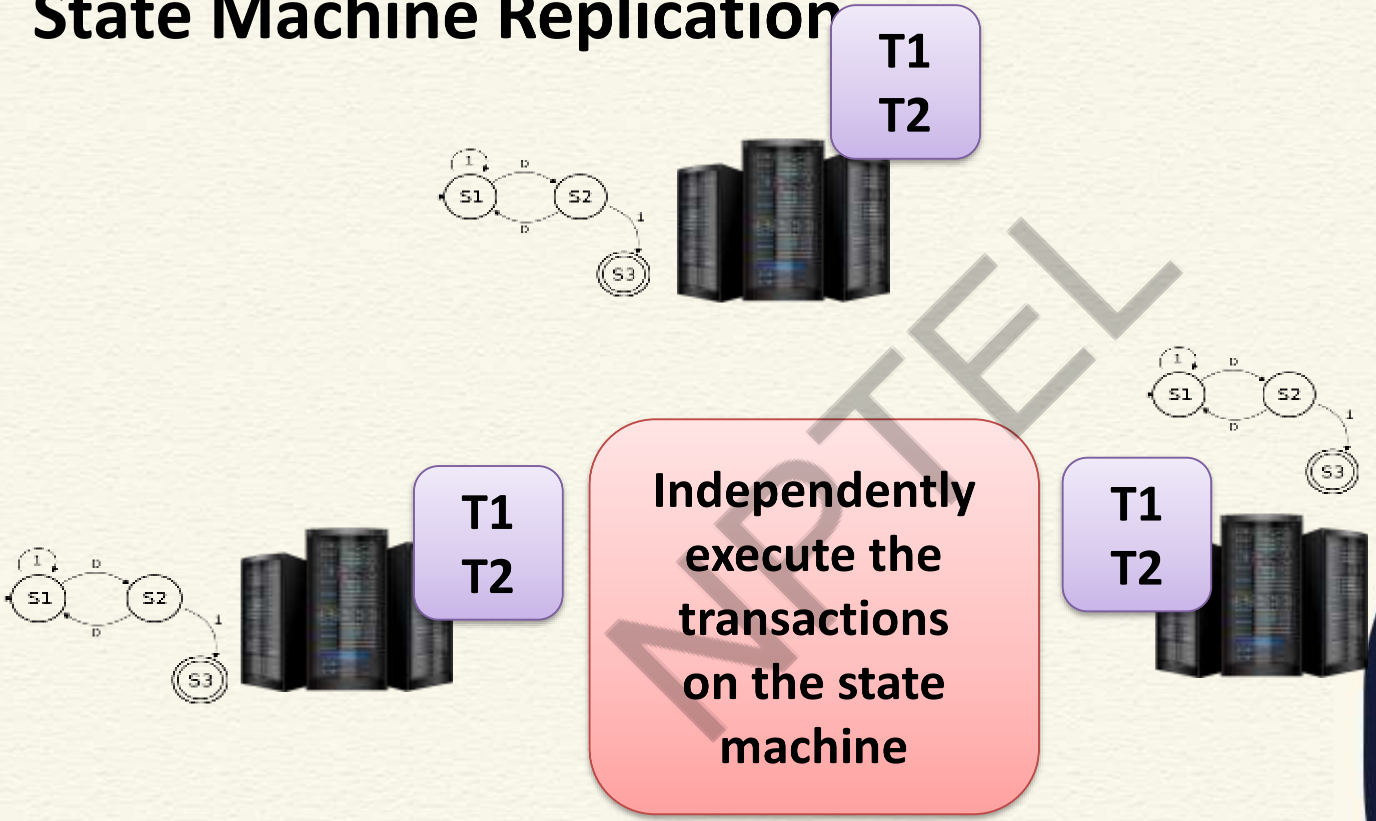
State Machine Replication



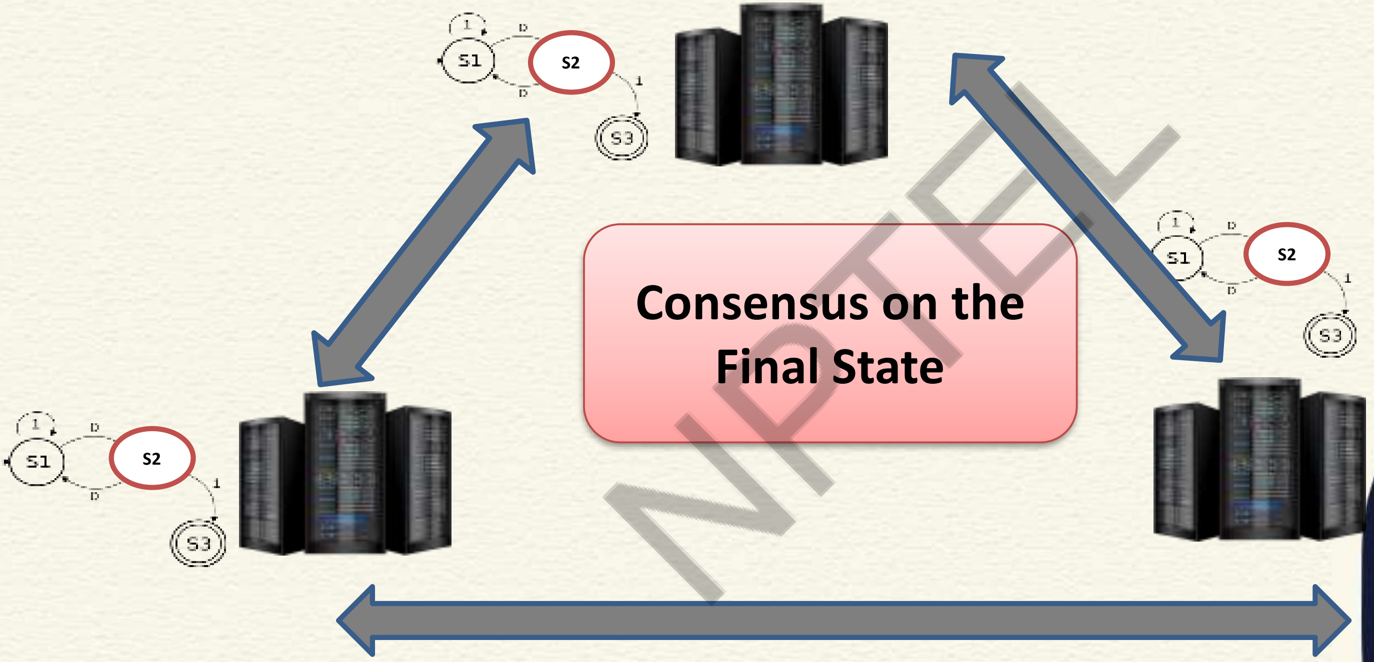
State Machine Replication



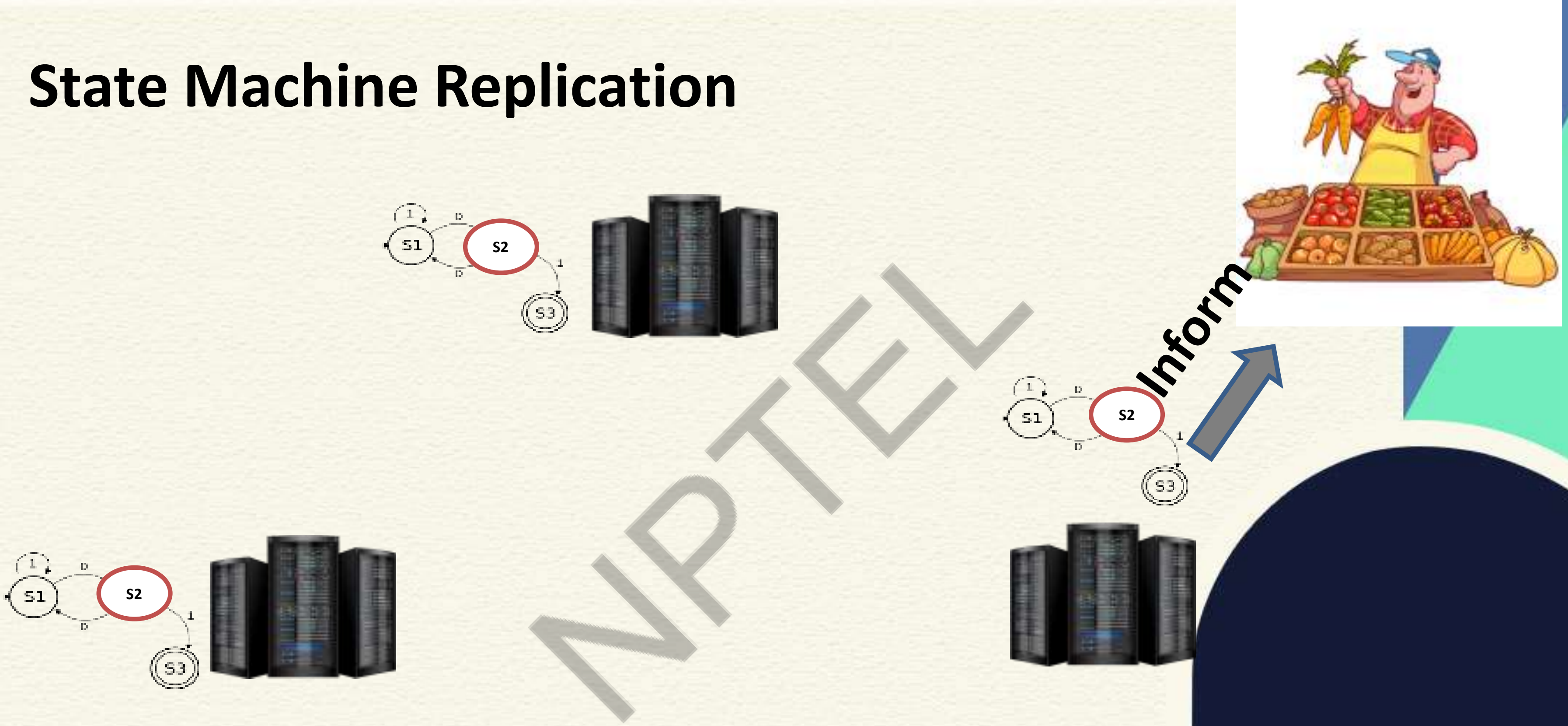
State Machine Replication



State Machine Replication

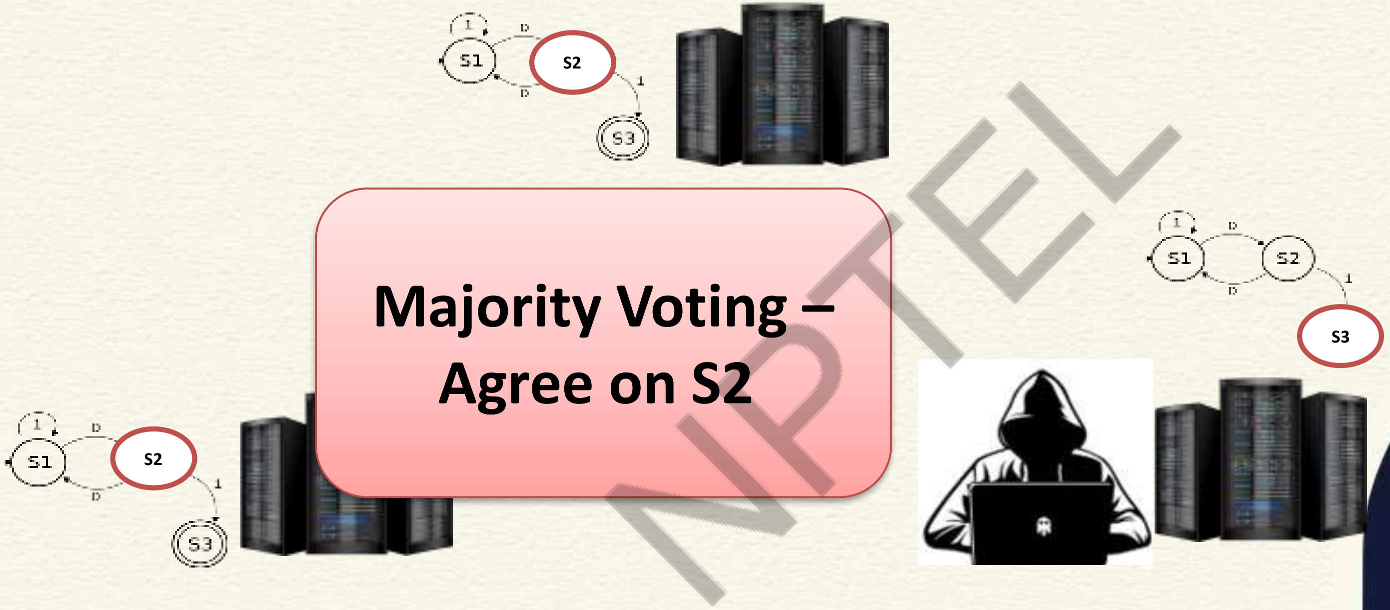


State Machine Replication



Importance of Consensus

Importance of Consensus



Majority Voting – Agree on S2

Conclusion

- Consensus is still required within a closed environment
- How can we extend state machine replication for permissioned models?


Thank you

Blockchain and its applications

Prof. Sandip Chakraborty

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Lecture 27: State Machine Replication as Distributed Consensus

CONCEPTS COVERED

- State Machine Replication as a Consensus
- Synchronous vs Asynchronous Consensus with Crash Faults

KEYWORDS

- **Crash Fault Tolerance**
- **Paxos**

State Machine Replication as Consensus

- There is a natural reason to use state machine replication-based consensus over permissioned blockchains
  - The network is closed, the nodes know each other, so state replication is possible among the known nodes
  - Avoid the overhead of mining - do not need to spend anything (like power, time, bitcoin) other than message passing
  - However, consensus is still required - machines can be faulty or behave maliciously

State Machine Replication as Consensus

- There is a natural reason to use state machine repli... ned bloc...
  - T... er, so s... odes
  - A... pend a... essage p...
  - H... can be faulty or be... aliciously

But, we need a bit redesign !

State Machine Replication as Consensus

- There ... chine rep ... oned blo
  - ... her, so ... nodes
  - ... spend ... message
  - ... es can be faulty or be ... e maliciously

But, we need a bit redesign !

Crypto is the saver

Crypto + Distributed Consensus = Consensus for Permissioned Blockchain

State Machine Replication as Consensus

- Classical Distributed Consensus Algorithms (**Paxos**, **RAFT**, **Byzantine Agreement**) are based on State Machine Replication
  - Let us (re)visit those algorithms

Faults in a Distributed Systems

- **<u>Crash Faults</u>**: The node stops operating – hardware or software faults
  - In an asynchronous system: You do not know whether messages have been delayed or the node is not responding
  - Rely on majority voting – progress as and when you have received the confirmation from the majority
  - Propagation of the consensus information – nodes on a slow network will receive it eventually

Faults in a Distributed Systems

- **<u>Byzantine Faults</u>:** Nodes misbehave – send different information to different peers (partition the network)
    - More difficult to handle
    - More suitable for blockchains

Asynchronous Consensus with Crash Faults

- Remember the **FLP Impossibility**
  - Give priority to safety over liveness

- Guarantees the followings --
  - **Validity**: If all correct process proposes the same value v, then any correct process decides v
  - **Agreement:** No two correct processes decide differently
  - **Termination**: Every correct process eventually decides

Asynchronous Consensus with Crash Faults

- Guarantees the followings --
  - **Validity**: If all correct process proposes the same value v, then any correct process decides v **(Unlikely to happen in PoW)**
  - **Agreement:** No two correct processes decide differently **(Safety – Not in PoW)**
  - **Termination**: Every correct process eventually decides **(Liveness – Priority in PoW)**

CFT Consensus

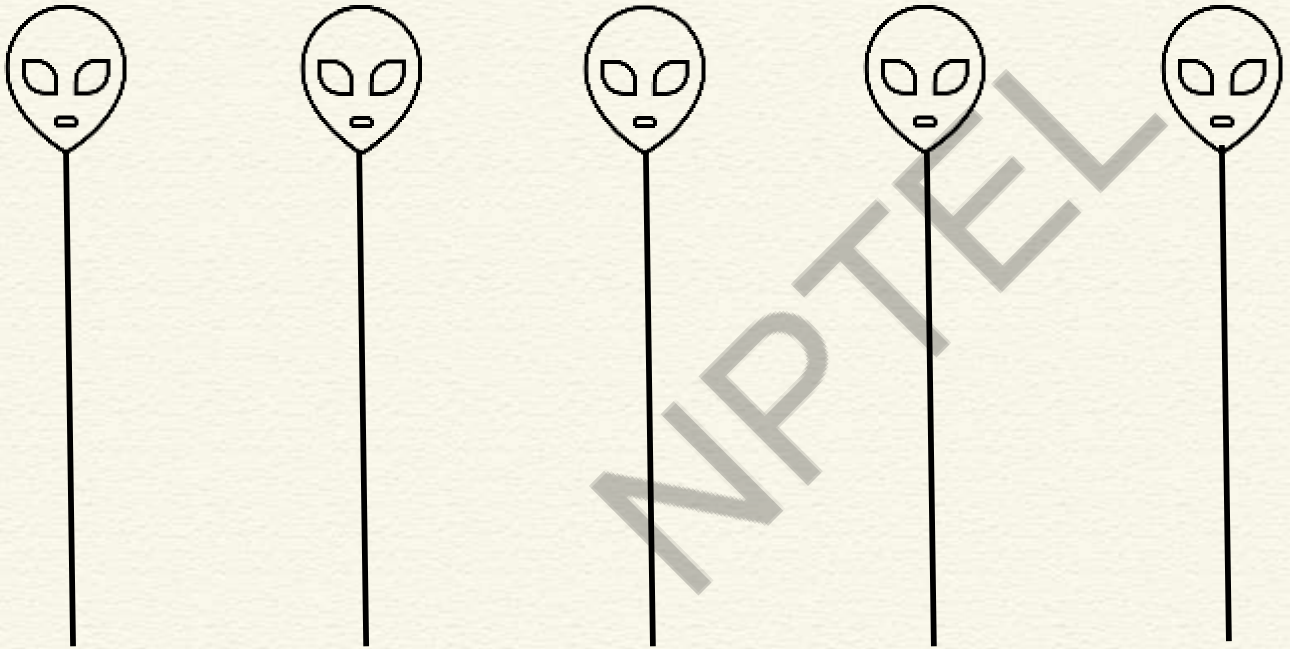
- CFT Consensus
  - **Paxos** (Proposed by Lamport, the most fundamental CFT) -- used in DynamoDB
  - **RAFT** (Much simpler than Paxos) -- Used in Fabric Transaction Ordering

CFT Consensus

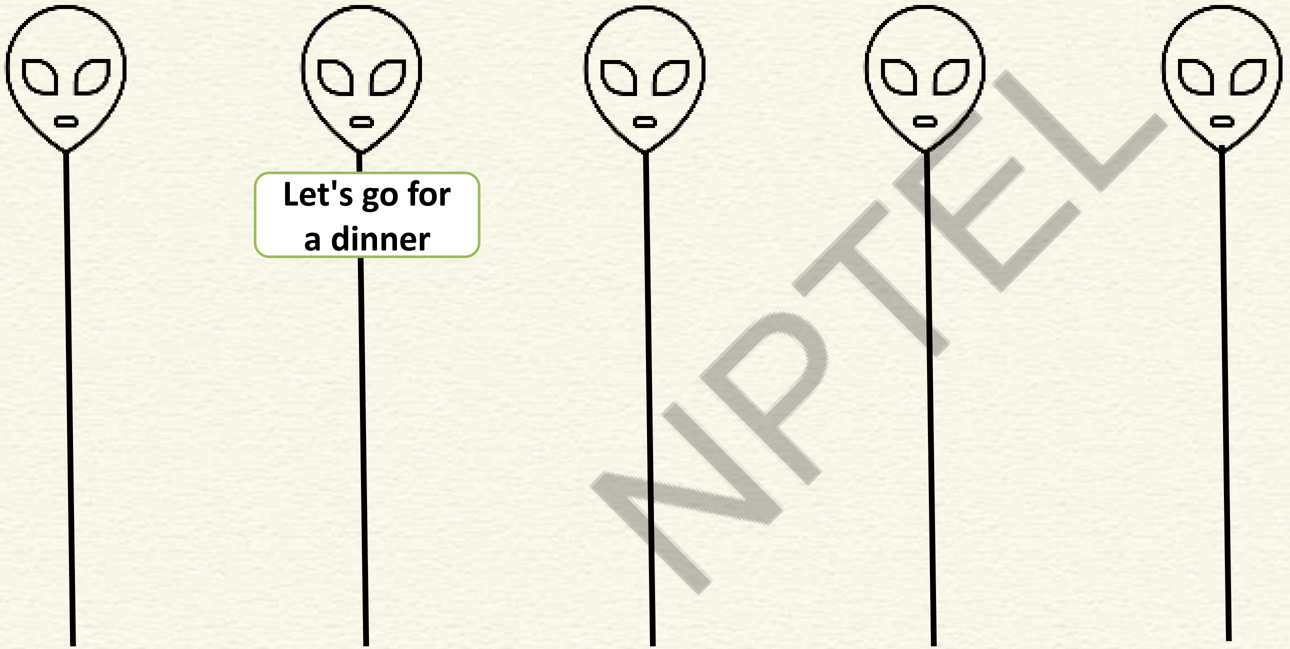
We'll see how Paxos works

- CFT Consensus
  - **Paxos** (Proposed by Lamport, the most fundamental CFT) -- used in DynamoDB
  - **RAFT** (Much simpler than Paxos) -- Used in Fabric Transaction Ordering

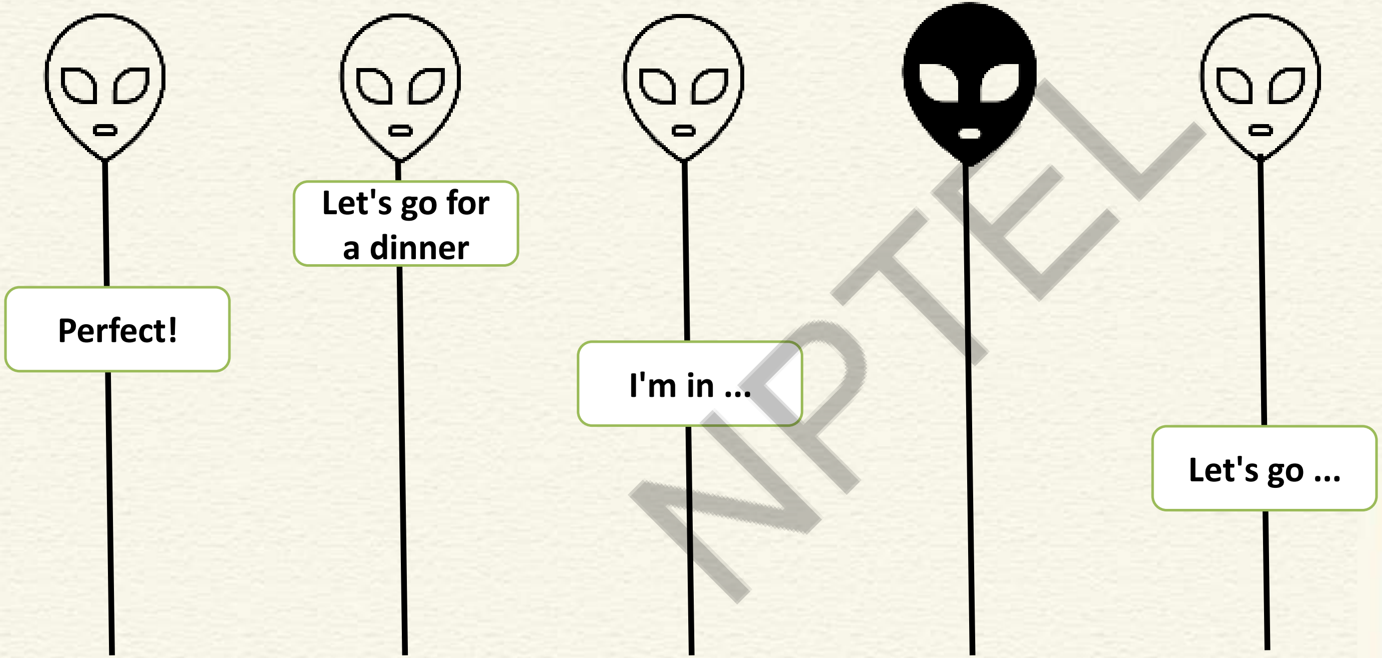
CFT in a Synchronous System



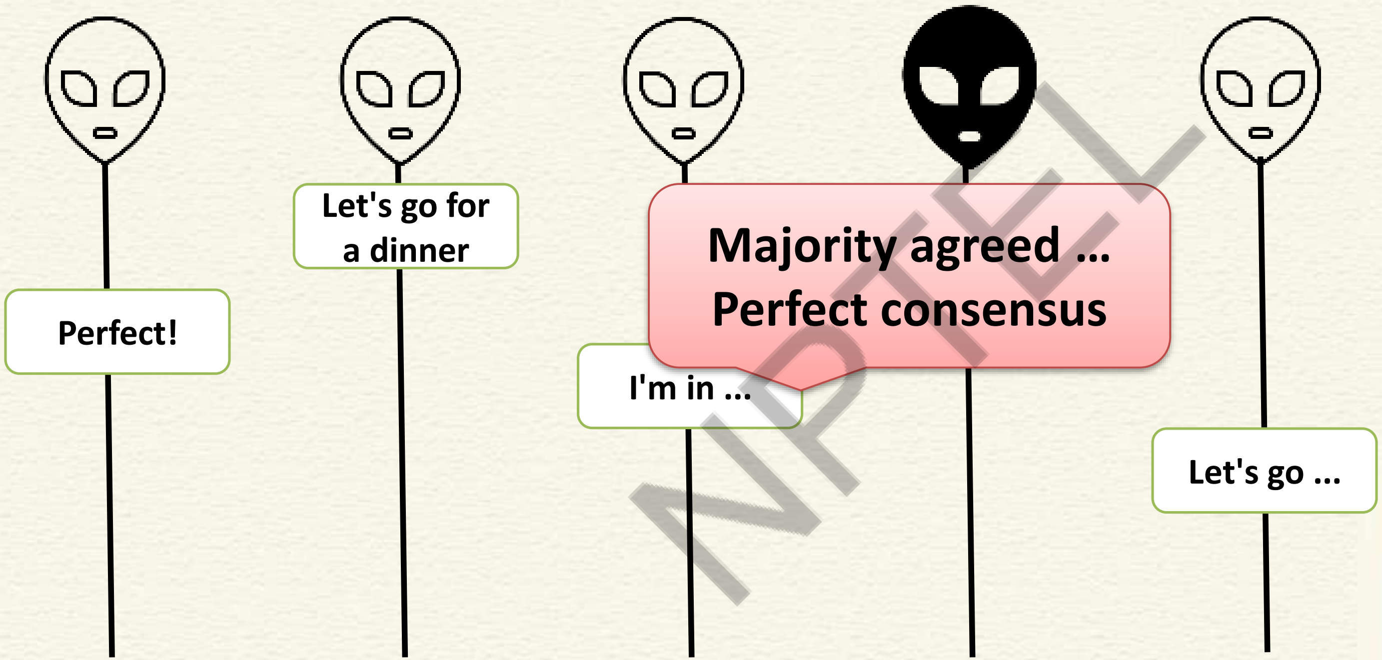
CFT in a Synchronous System



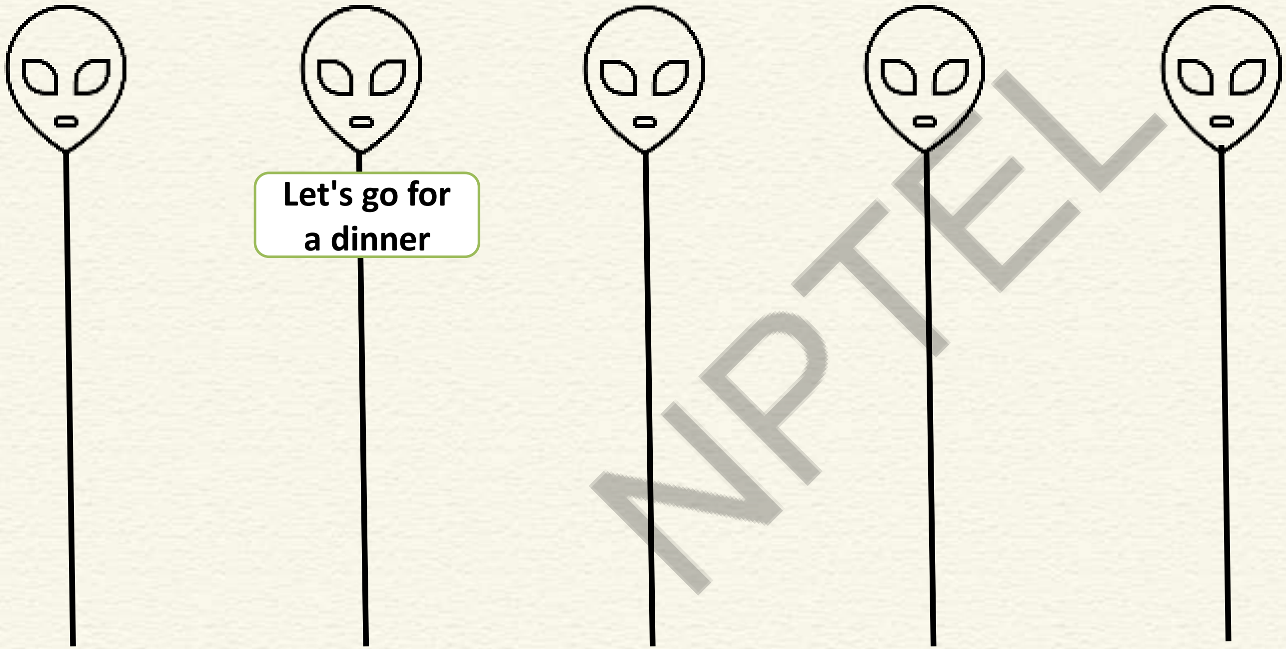
CFT in a Synchronous System



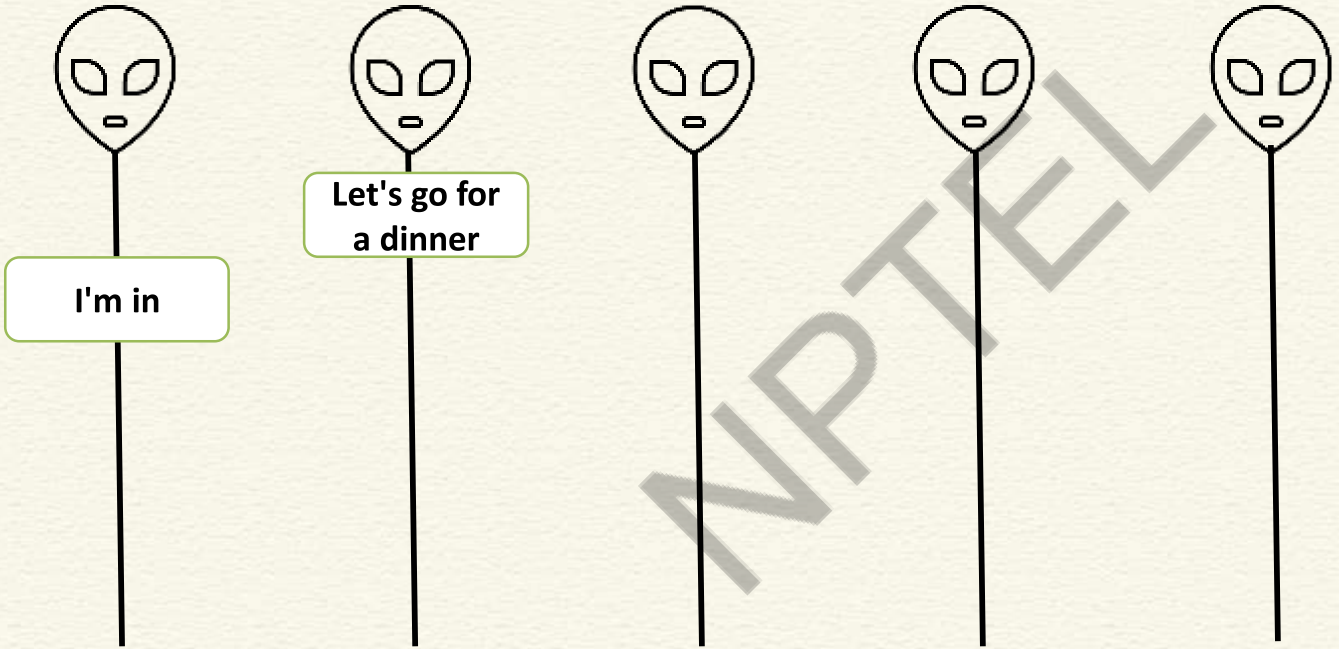
CFT in a Synchronous System



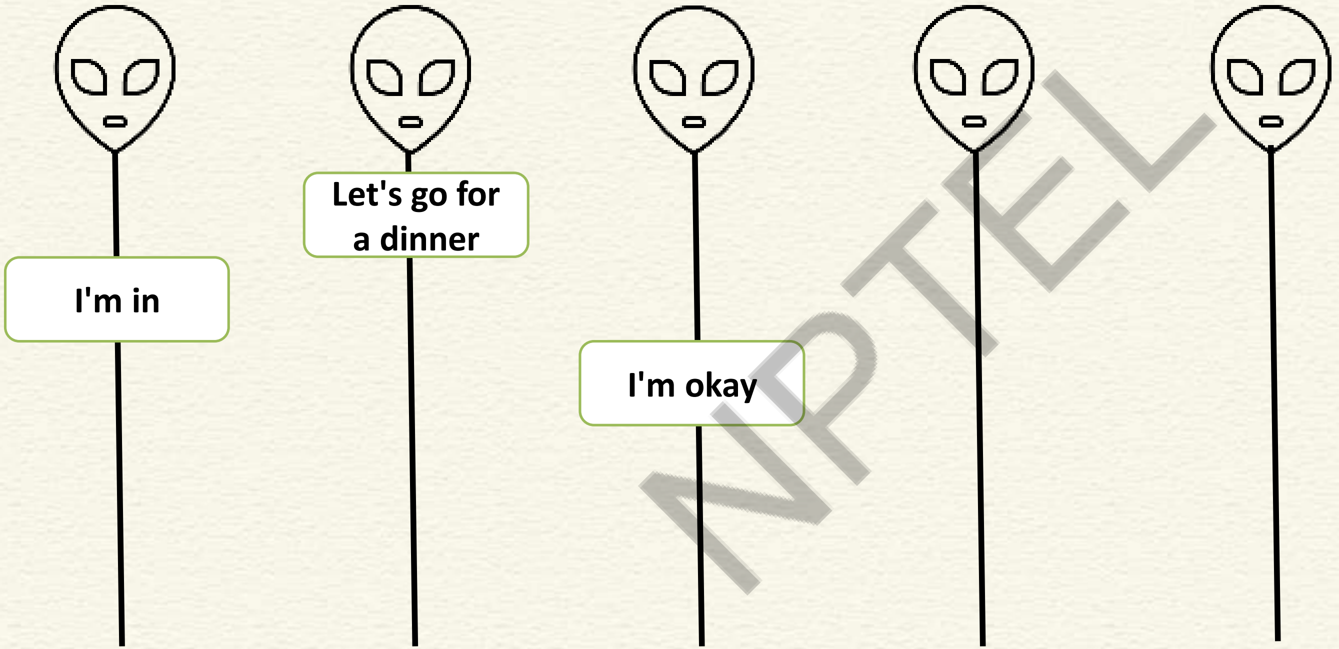
CFT in an Asynchronous System



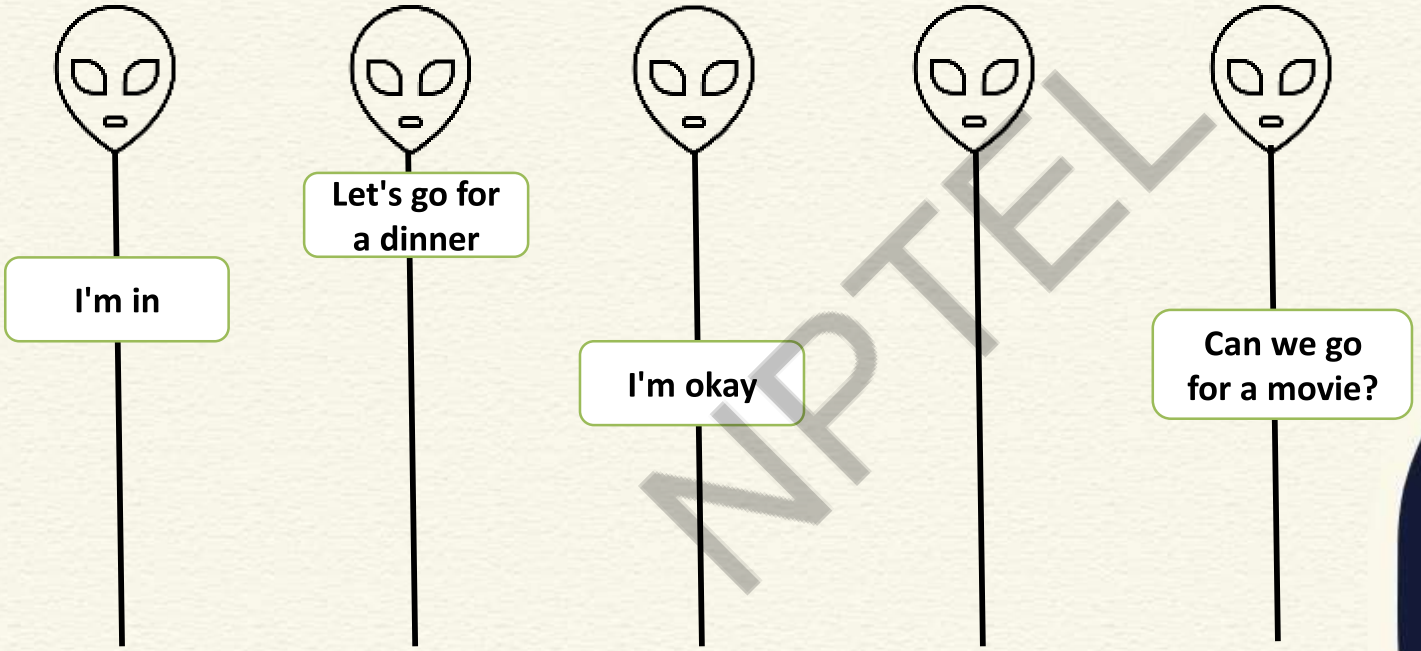
CFT in an Asynchronous System



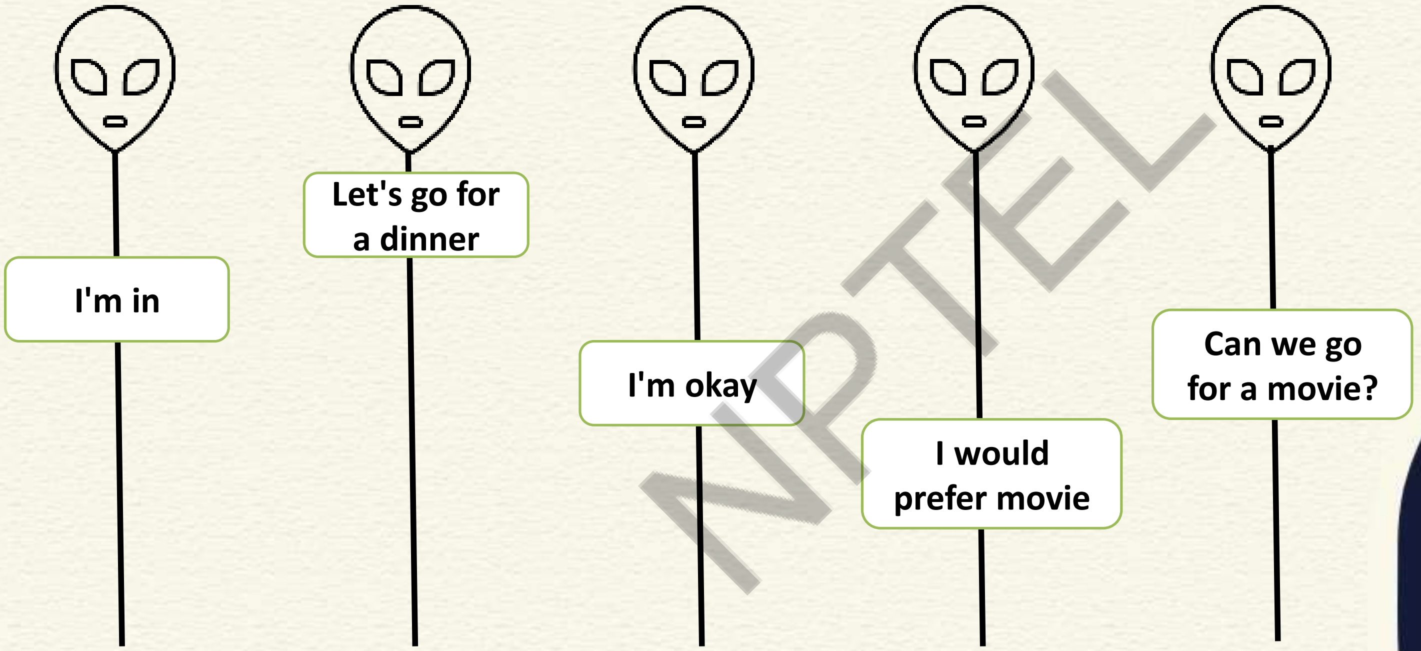
CFT in an Asynchronous System



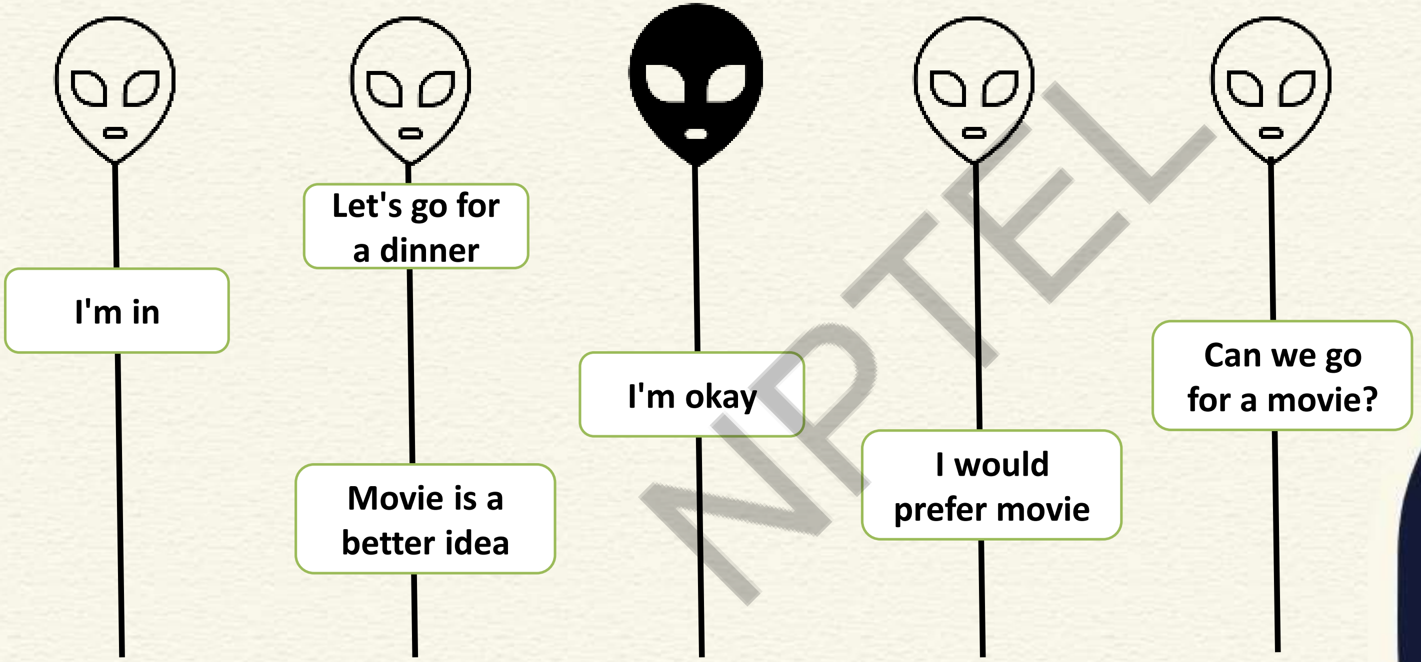
CFT in an Asynchronous System



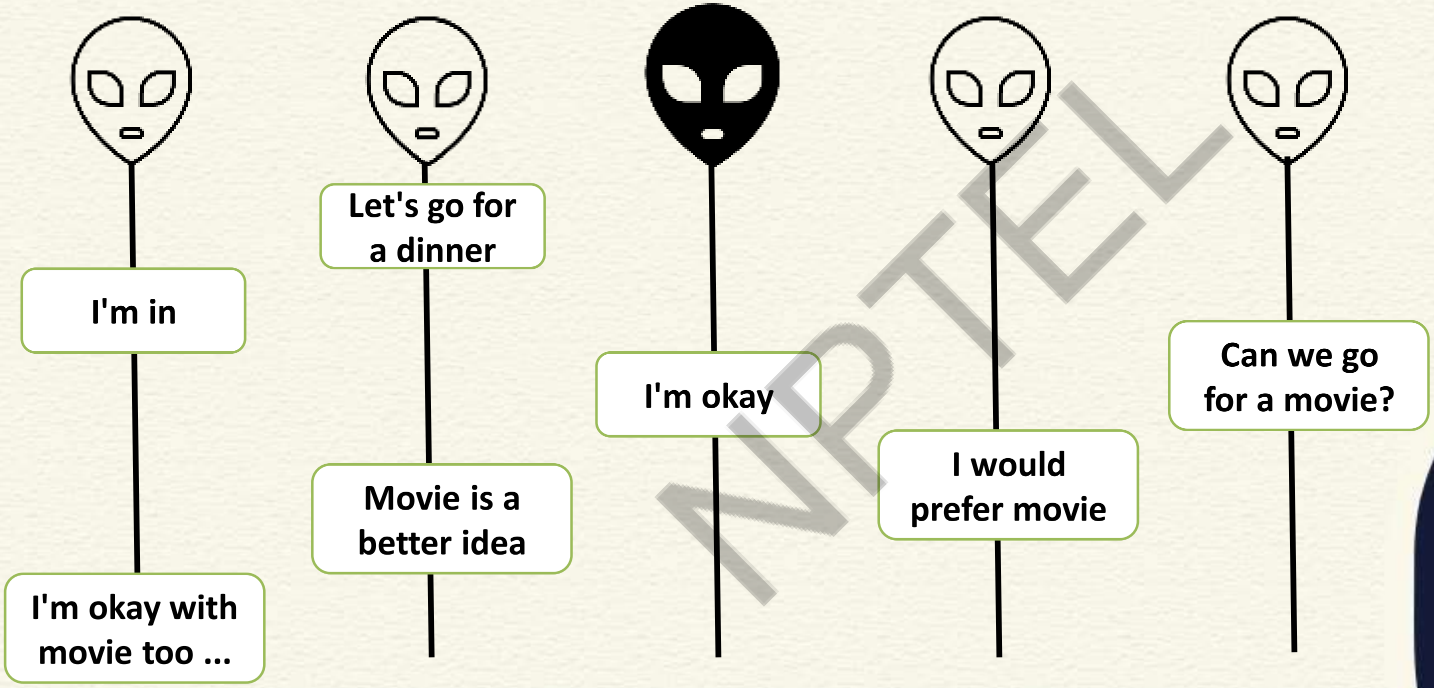
CFT in an Asynchronous System



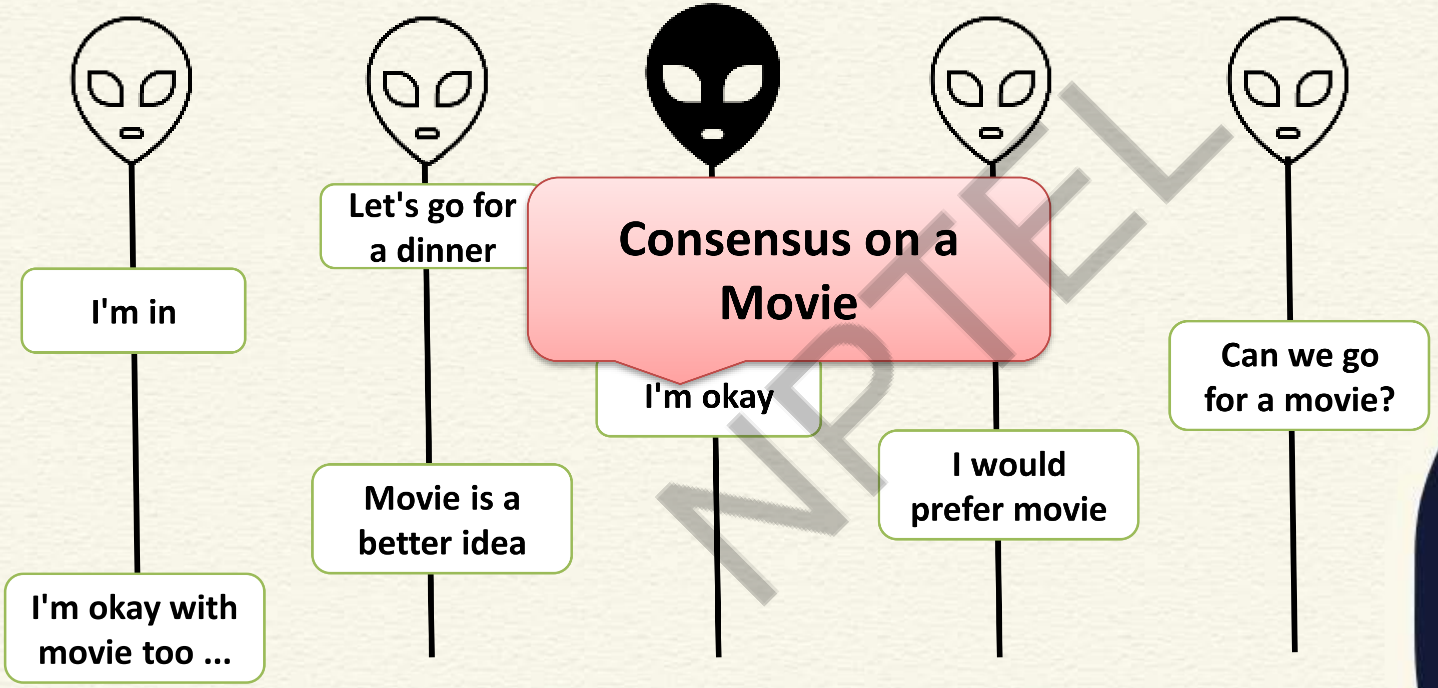
CFT in an Asynchronous System



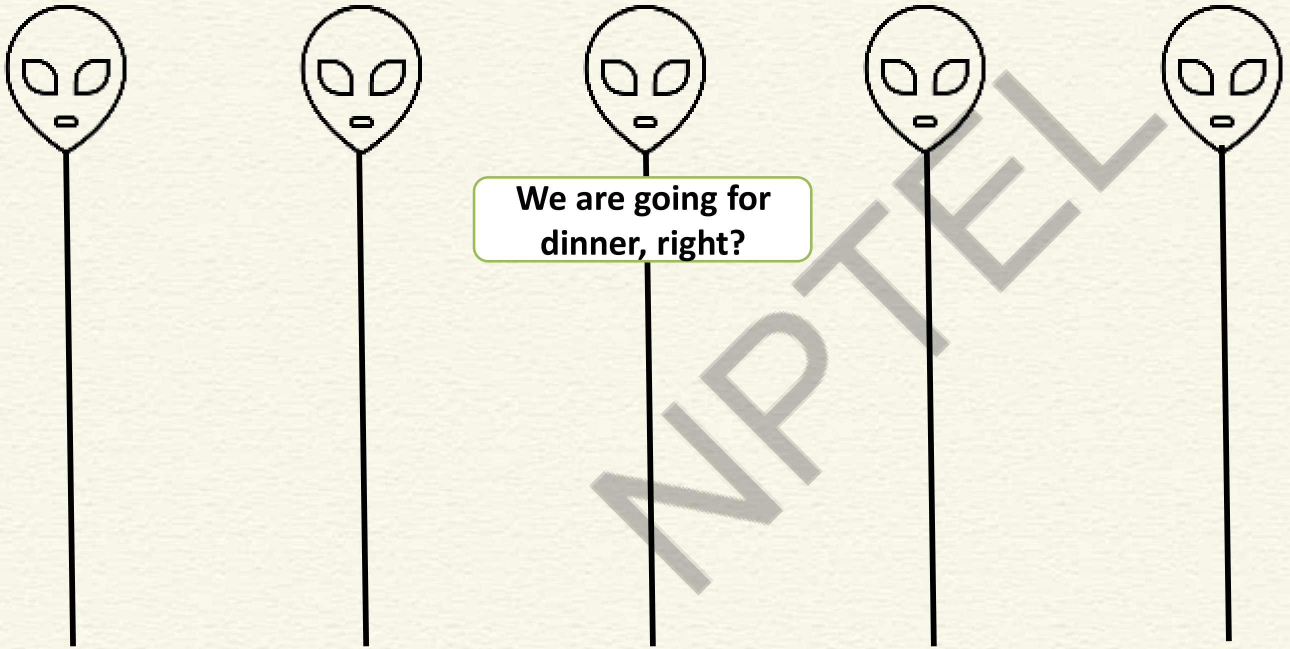
CFT in an Asynchronous System



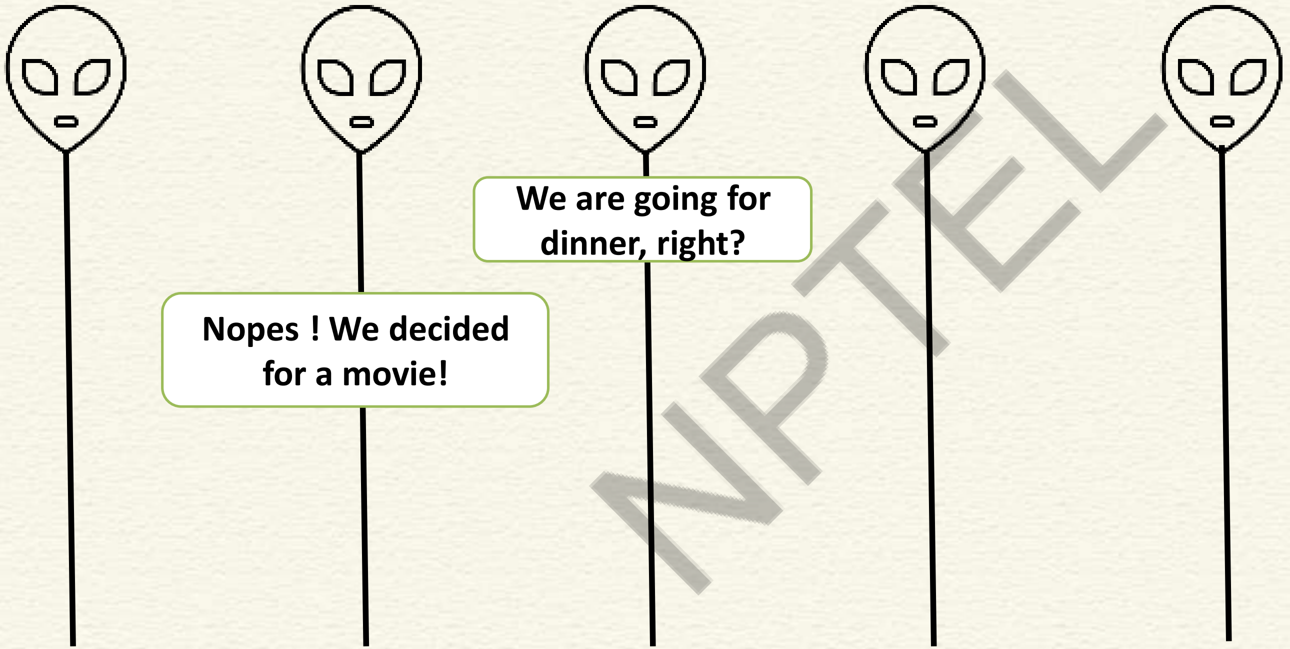
CFT in an Asynchronous System



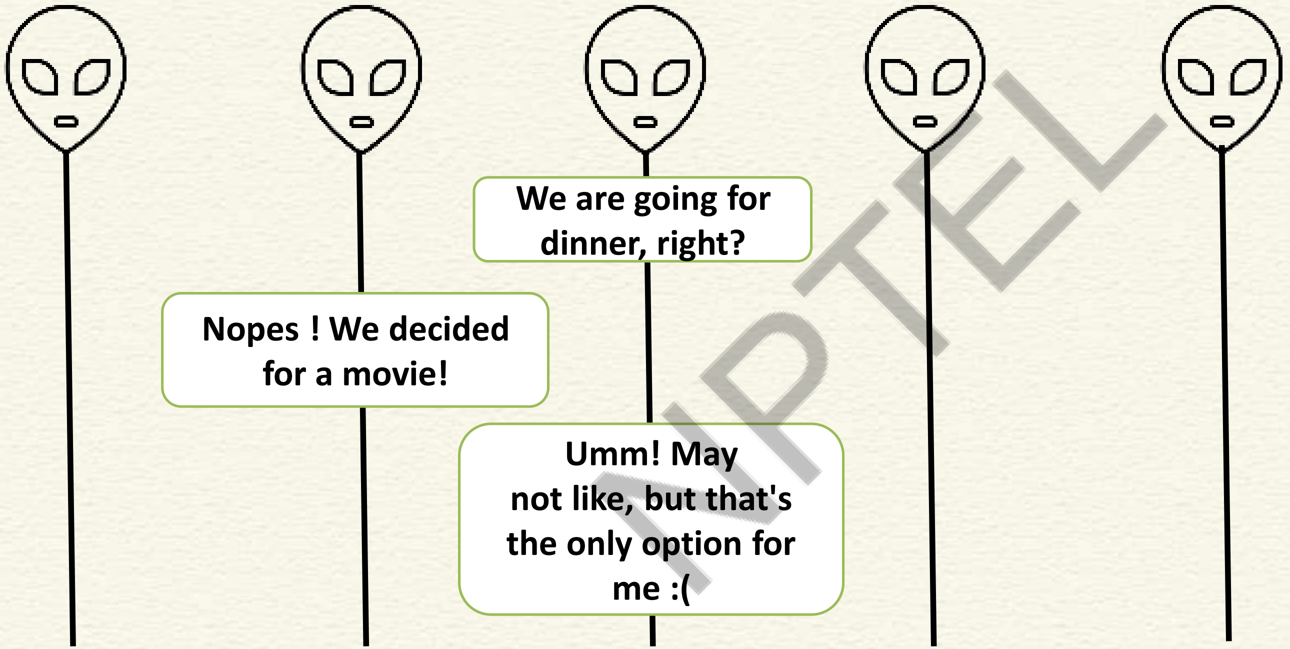
CFT in an Asynchronous System



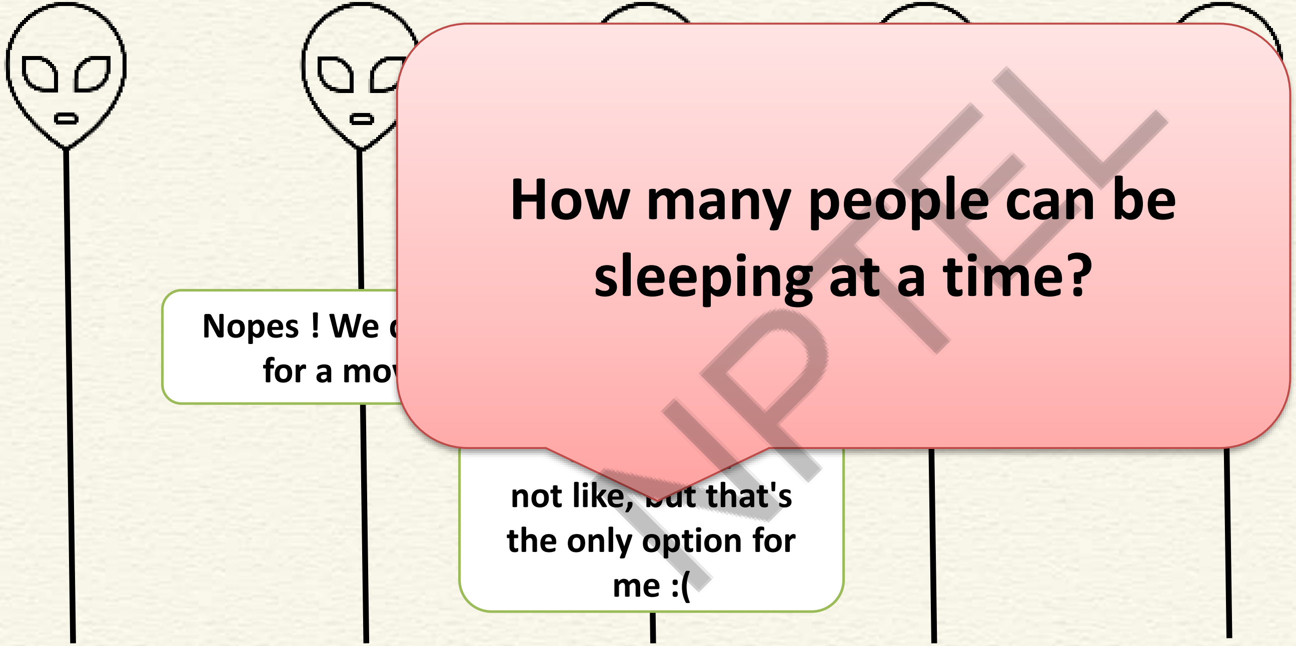
CFT in an Asynchronous System



CFT in an Asynchronous System

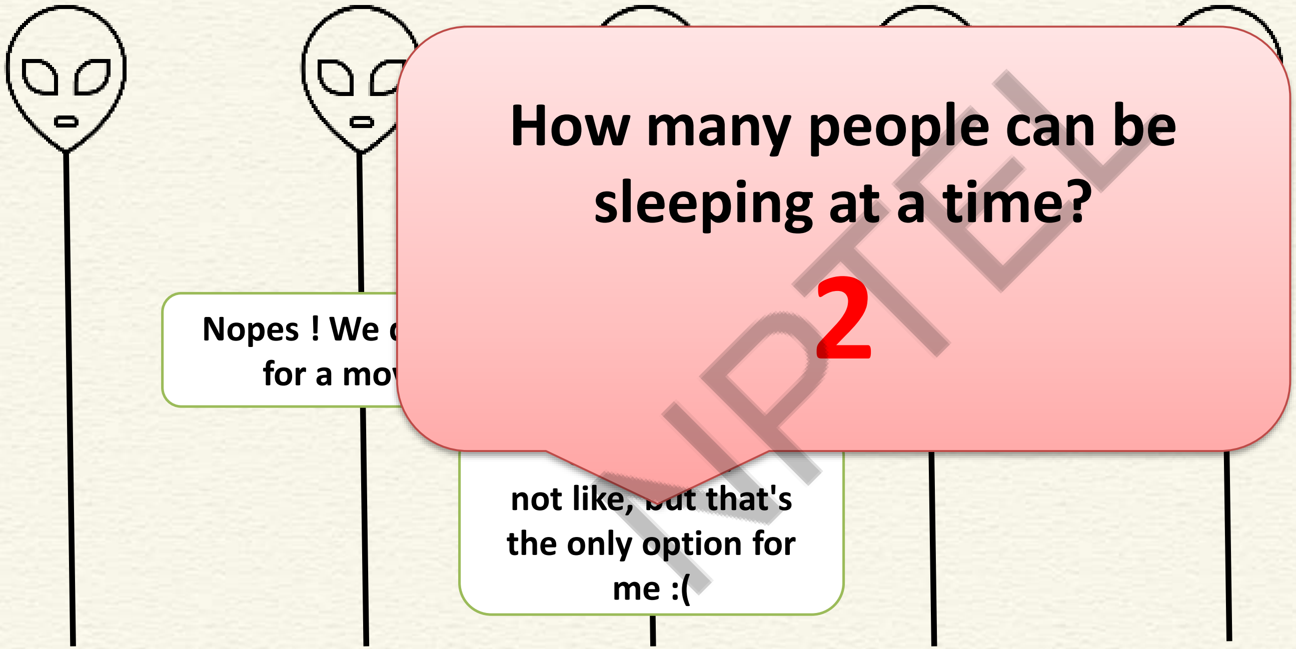


CFT in an Asynchronous System



How many people can be sleeping at a time?

CFT in an Asynchronous System



Asynchronous CFT

- If there are F faulty nodes (crash fault), we need atleast 2F+1 nodes to reach consensus

- **Paxos:** A family of distributed algorithms to reach consensus in an asynchronous CFT

What is Paxos?

- We'll discuss vanilla Paxos
- Proposed by Lamport in 1989
- Received a lot of criticism about its proof of correctness
- Accepted in ACM Transactions on Computer Systems in 1998, titled *"The Part-time Parliament"*
- Lamport received the Turing award in 2013

Conclusion

- Consensus is harder on asynchronous environment
- For asynchronous CFT, we need 2F+1 nodes with F crash faults only
- Let's explore Paxos in the next class

Thank you

Blockchain and its applications

Prof. Sandip Chakraborty

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Lecture 28: Paxos

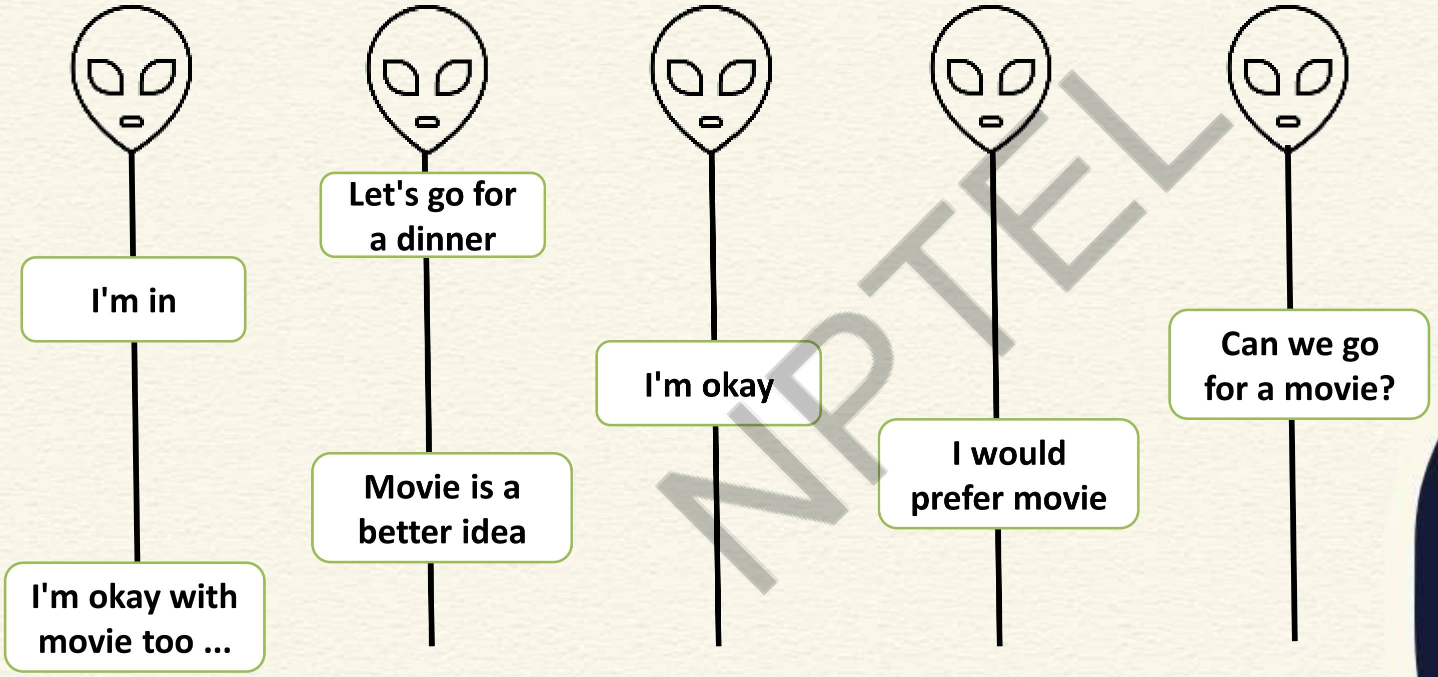
CONCEPTS COVERED

- **Paxos – CFT Consensus**

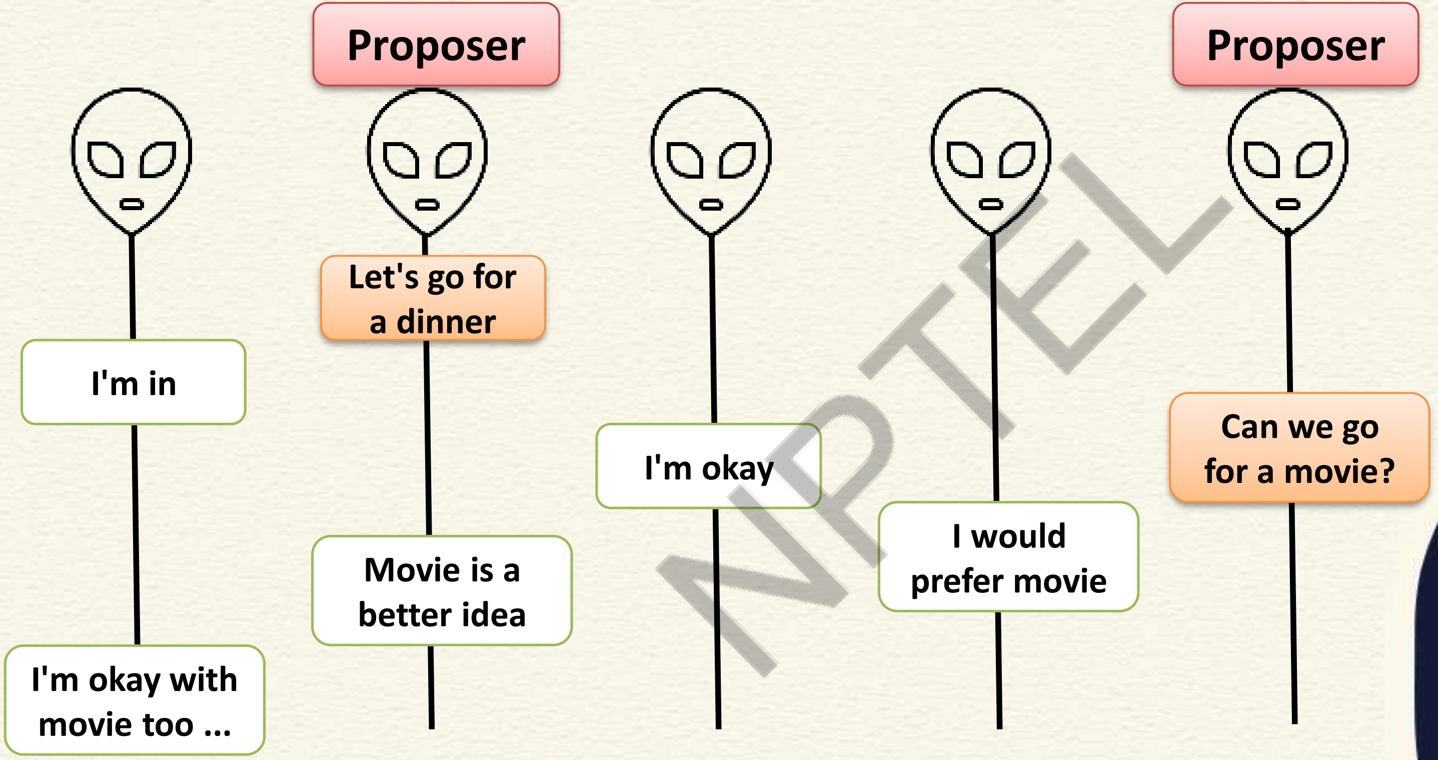
KEYWORDS

- **Paxos**
- **CFT**

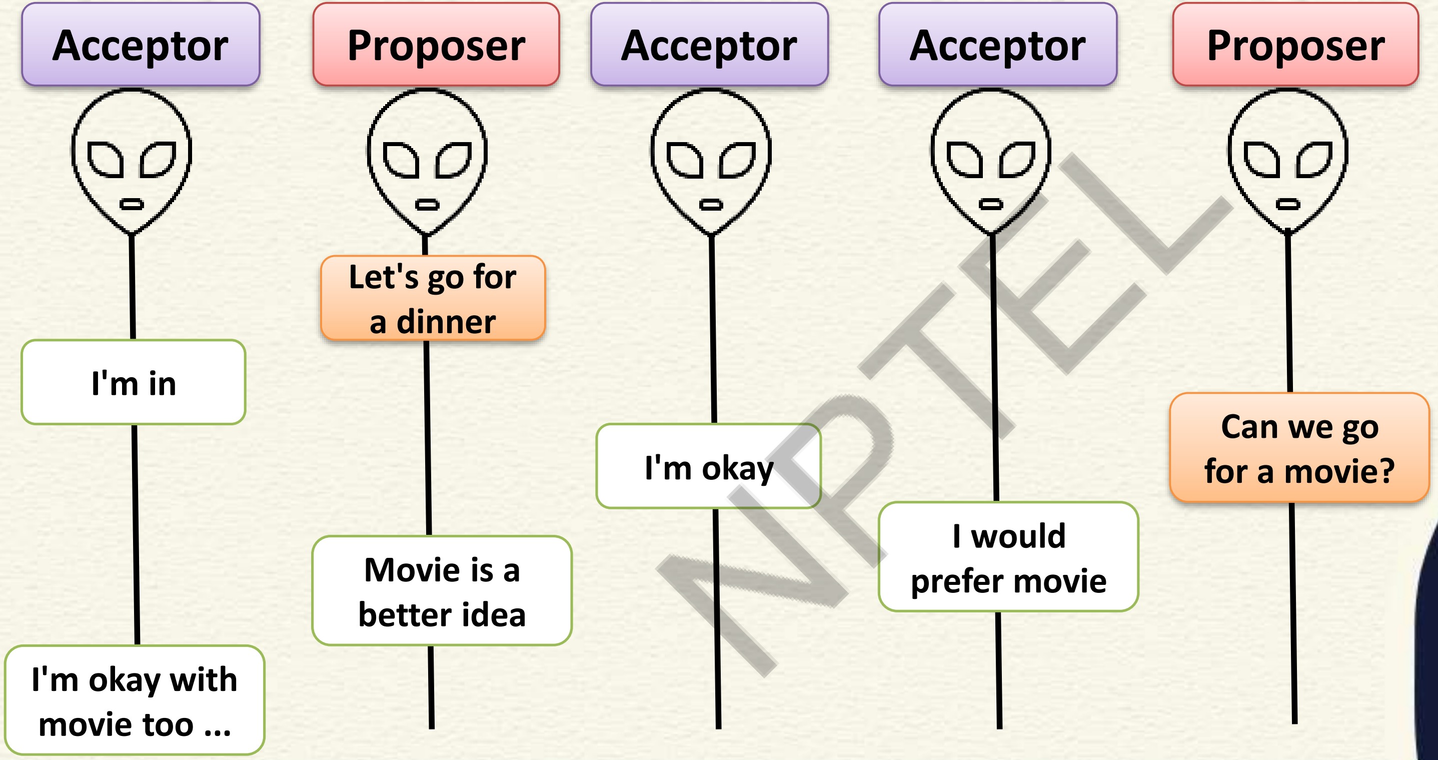
Paxos – The Roles of Individuals

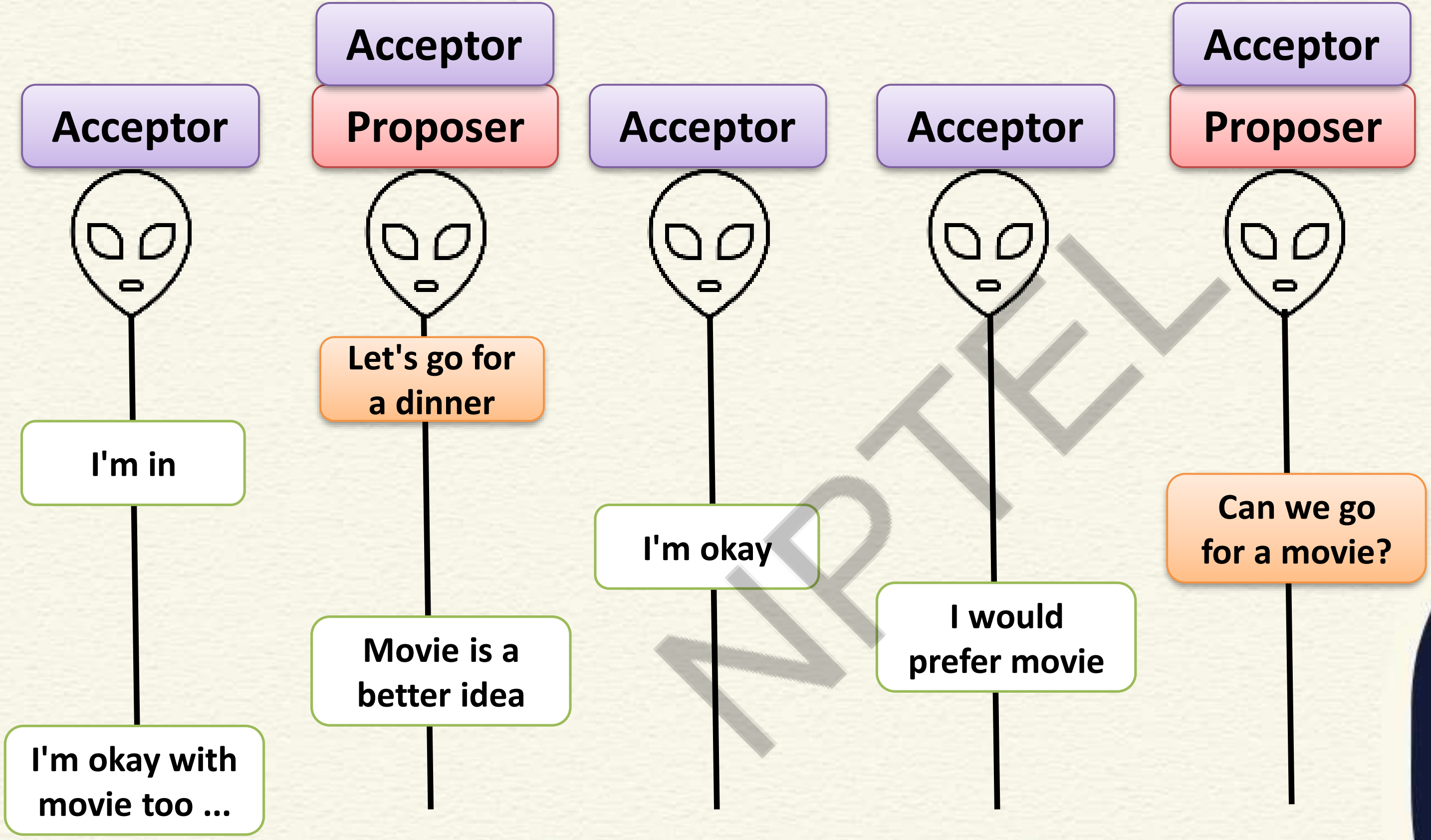


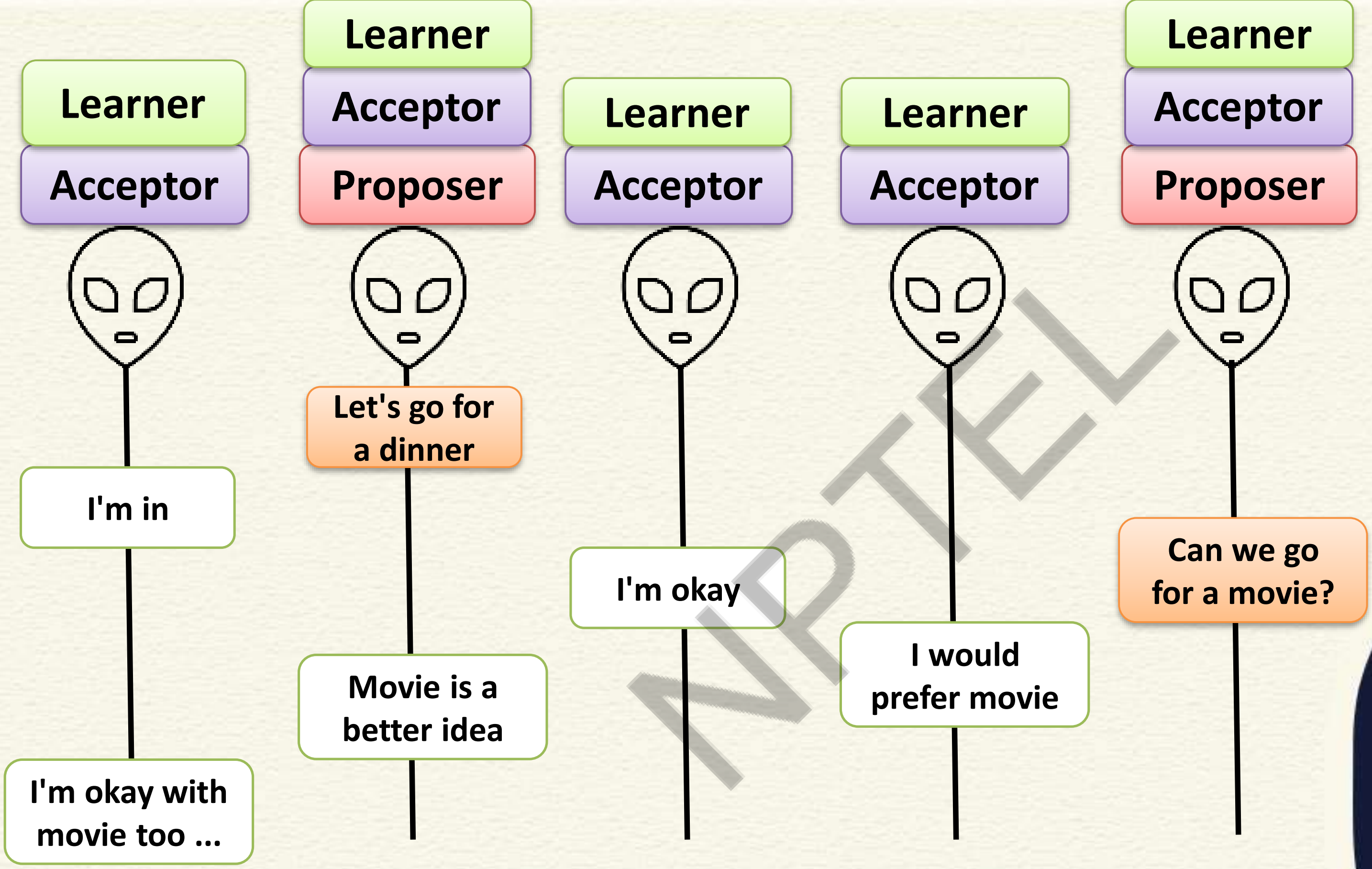
Paxos – The Roles of Individuals

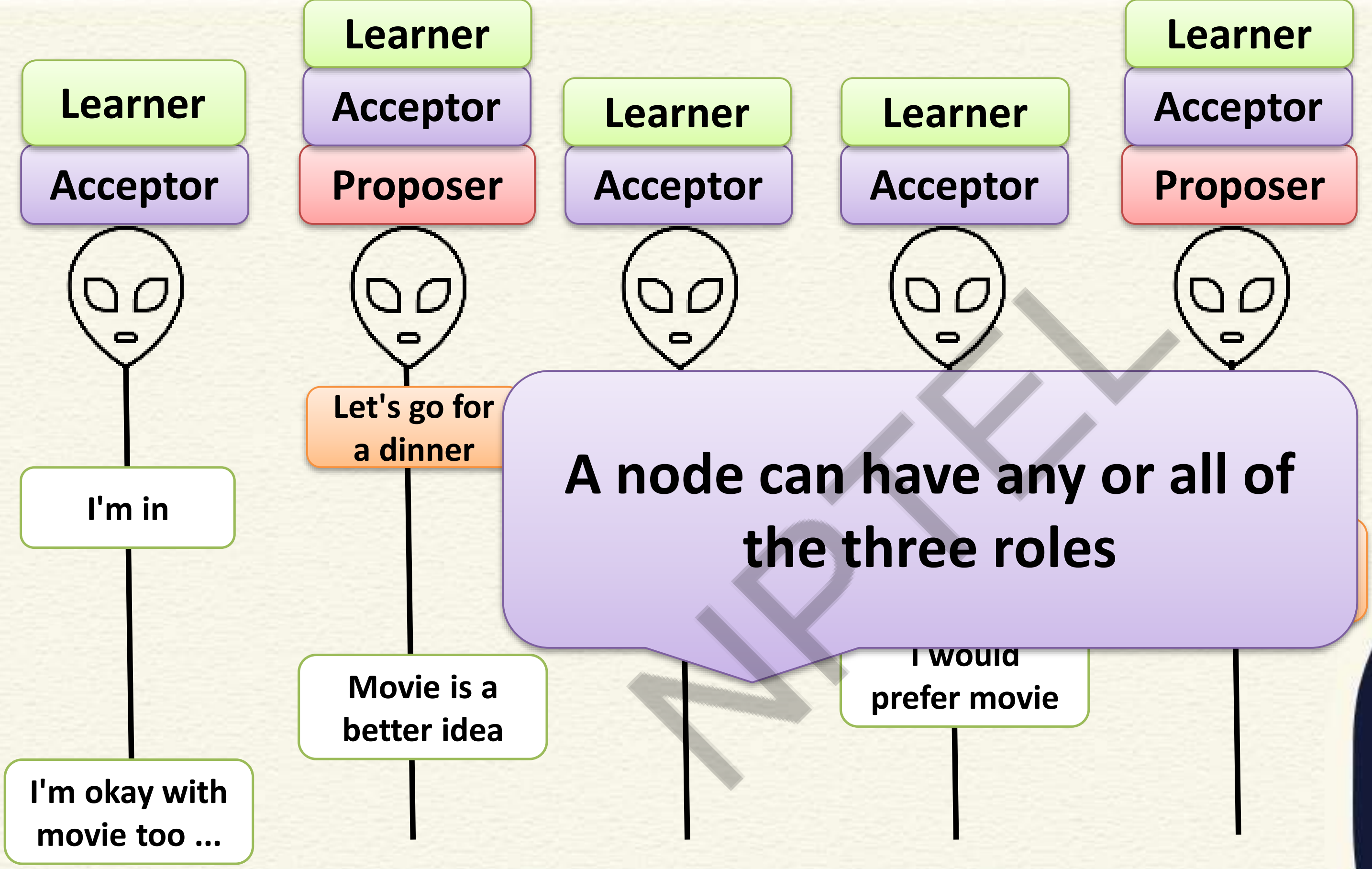


Paxos – The Roles of Individuals

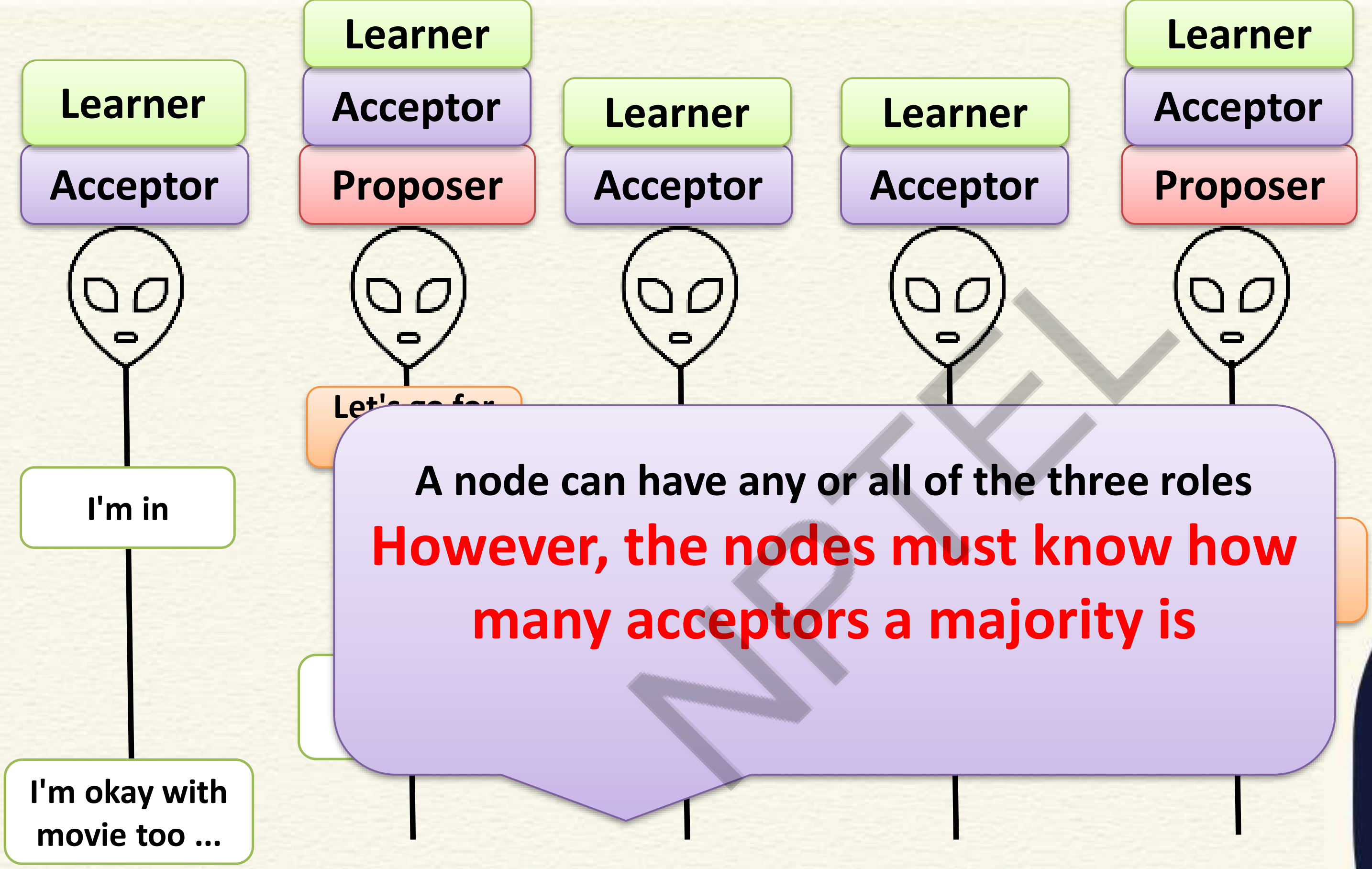



Acceptor
I'm in
I'm okay with movie too ...
Acceptor
Proposer
Let's go for a dinner
Movie is a better idea
Acceptor
I'm okay
Acceptor
I would prefer movie
Acceptor
Proposer
Can we go for a movie?


Learner
Acceptor
I'm in
I'm okay with movie too ...
Learner
Acceptor
Proposer
Let's go for a dinner
Movie is a better idea
Learner
Acceptor
I'm okay
Learner
Acceptor
I would prefer movie
Learner
Acceptor
Proposer
Can we go for a movie?

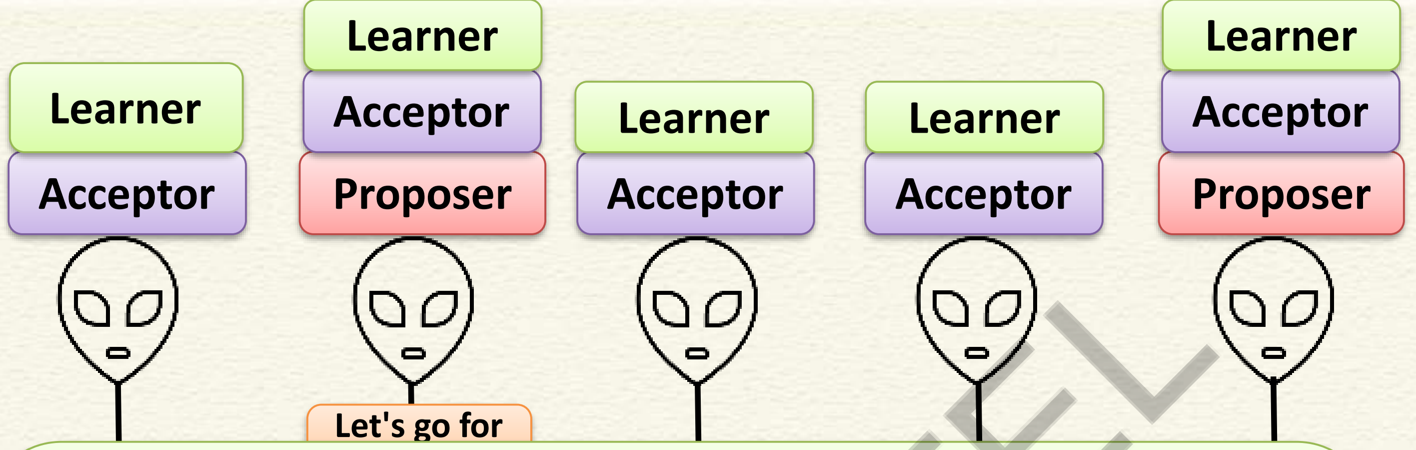


Learner
Acceptor
Learner
Acceptor
Proposer
Learner
Acceptor
Learner
Acceptor
Learner
Acceptor
Proposer
Let's go for a dinner
I'm in
Movie is a better idea
I'm okay with movie too ...
I would prefer movie
A node can have any or all of the three roles



A node can have any or all of the three roles

However, the nodes must know how many acceptors a majority is



Two majorities will always overlap in atleast one nodes

5 acceptors, majority = 3,

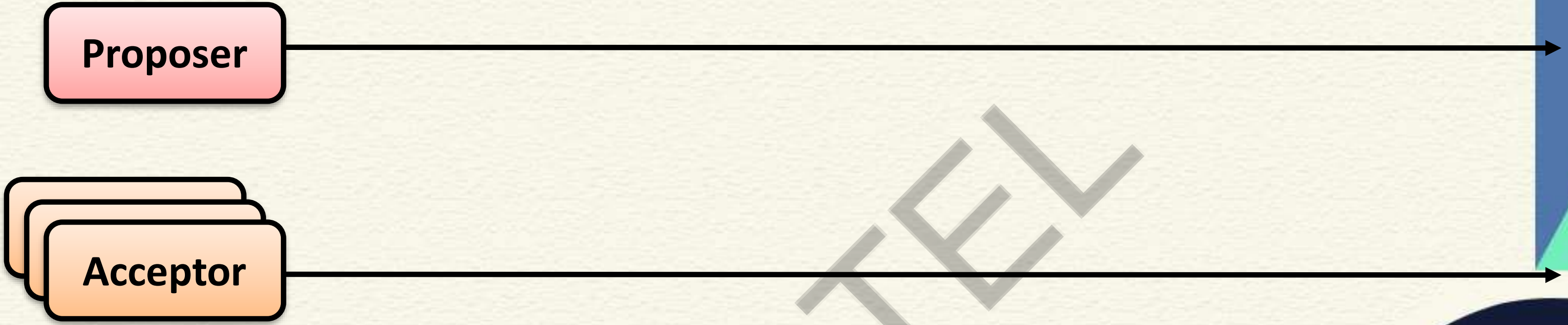
2 proposers:

To accept based on majority voting, at least one acceptor need to choose between one of the two proposals

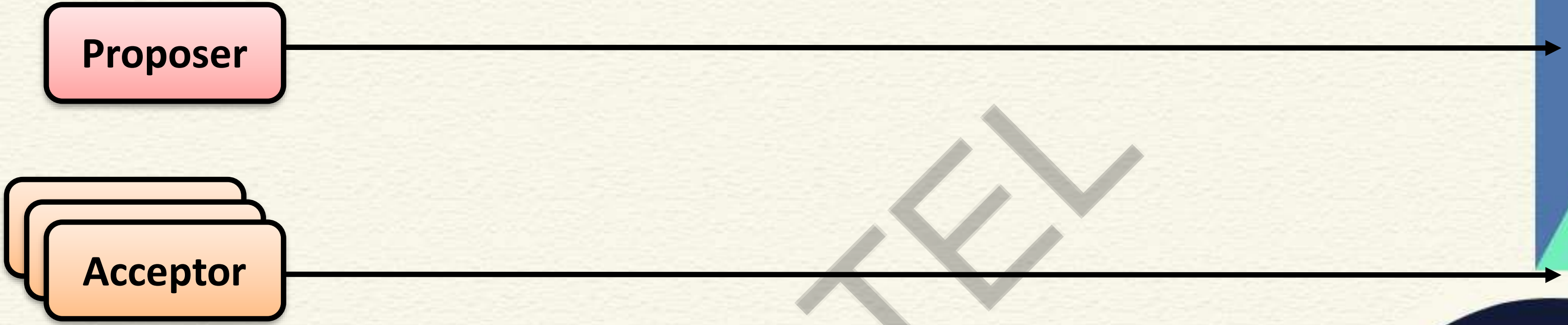
Paxos Basics

- Paxos is based on state-machine replication
  - Proposers and Acceptors maintain a state of the running epochs
  - Uses a variable IDp where p is an epoch number – maintains the state
- A Paxos run aims at reaching a **single consensus**
  - Once a consensus is reached, Paxos cannot progress to another consensus
  - To reach multiple consensus, you need to run Paxos in rounds (Multi-Paxos)

Paxos Algorithm

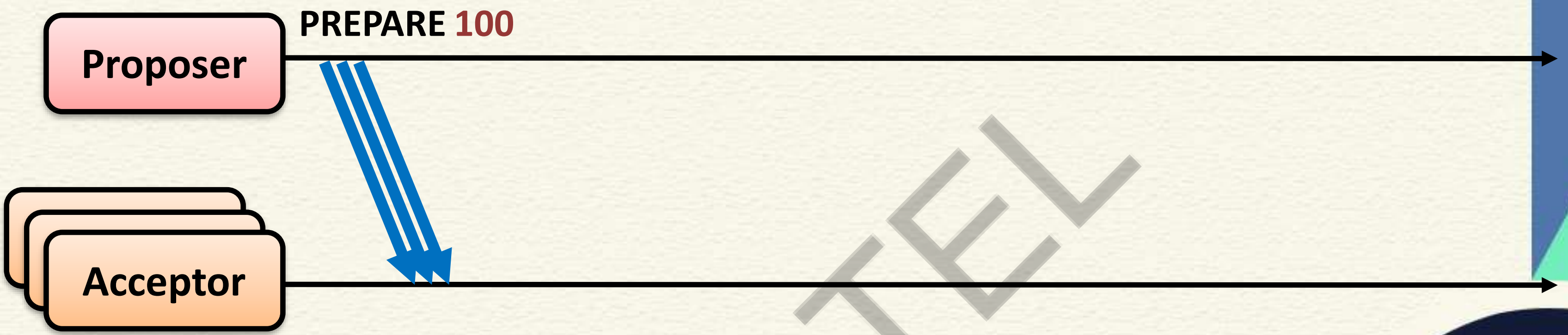


Paxos Algorithm



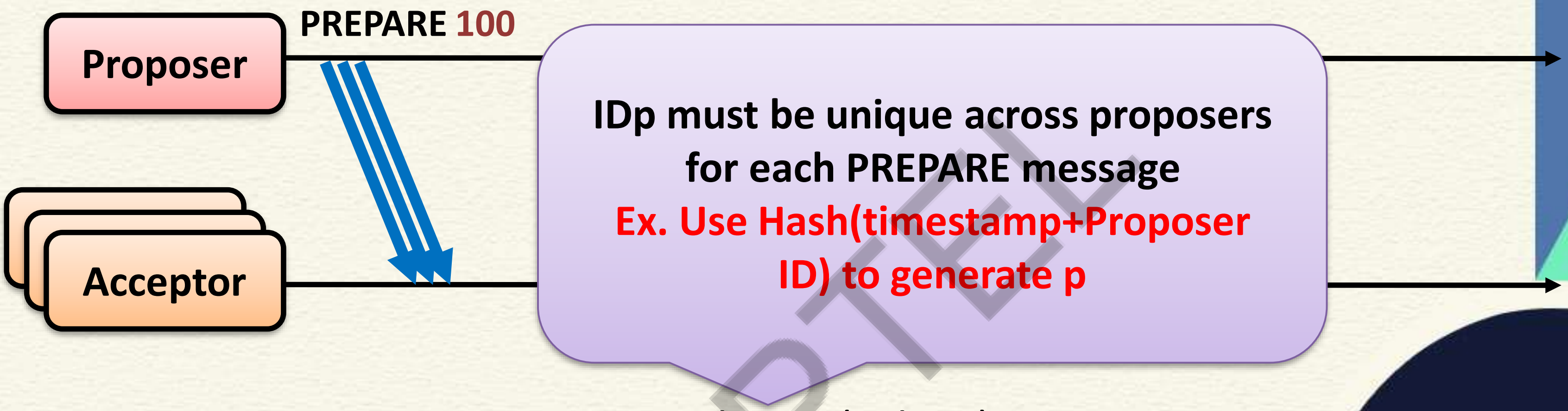
- **Proposer** wants to propose its choice (values):
  - Sends PREPARE IDp to a majority (or all) of the **acceptors**

Paxos Algorithm



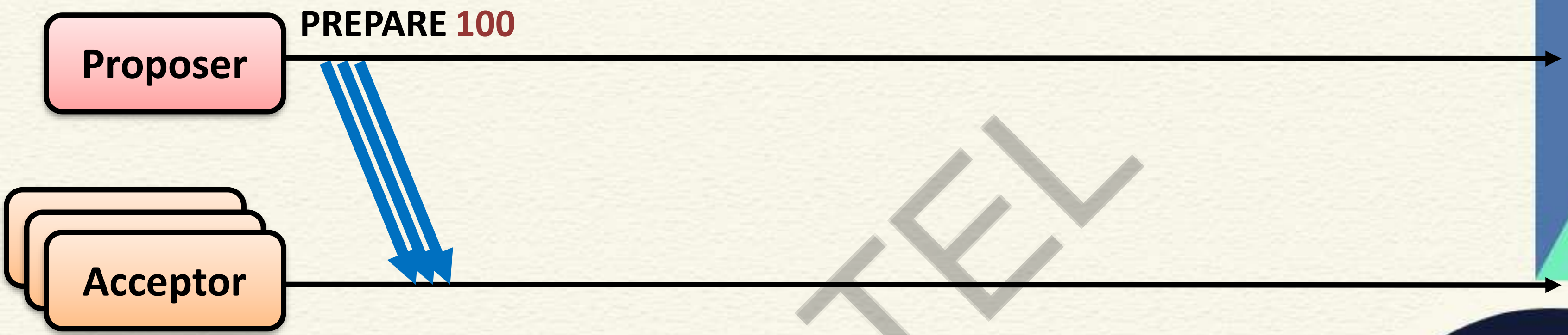
- **Proposer** wants to propose its choice (values):
  - Sends PREPARE IDp to a majority (or all) of the **acceptors**

Paxos Algorithm



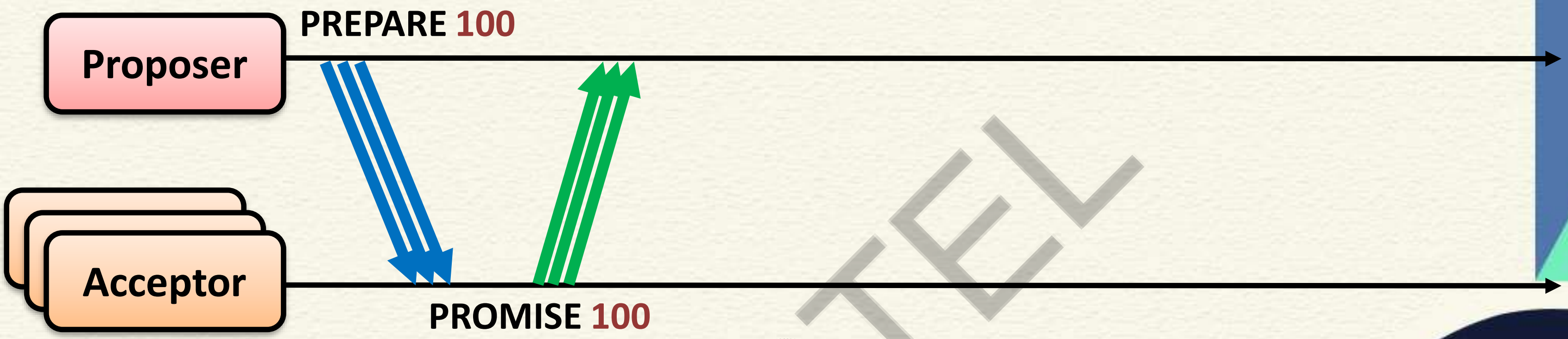
- **Proposer** wants to propose its choice (values):
  - Sends PREPARE IDp to a majority (or all) of the **acceptors**

Paxos Algorithm



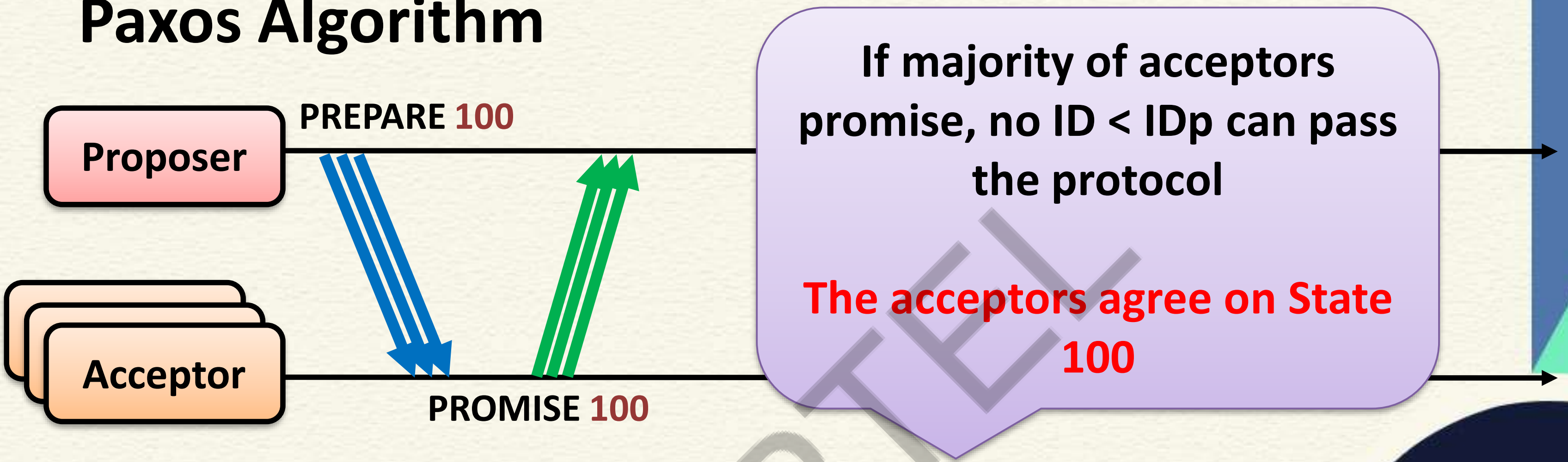
- **Acceptor** received a PREPARE message with IDp:
  - Did it promised to ignore requests with this IDp?
    - **YES:** Ignore
    - **NO:** Will promise to ignore any request lower than IDp
      - (?) Reply with PROMISE IDp

Paxos Algorithm



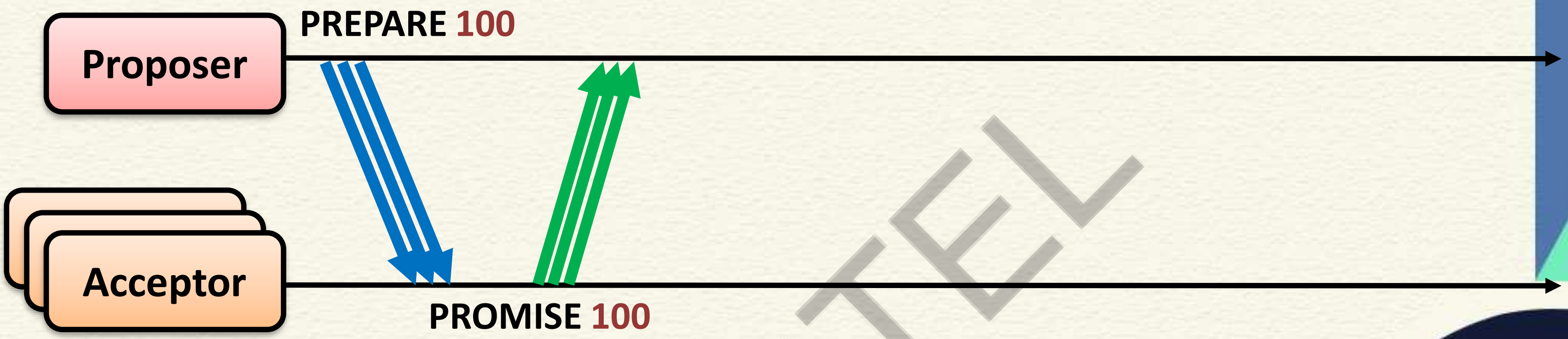
- **Acceptor** received a PREPARE message with IDp:
  - Did it promised to ignore requests with this IDp?
    - **YES:** Ignore
    - **NO:** Will promise to ignore any request lower than IDp
      - (?) Reply with PROMISE IDp

Paxos Algorithm



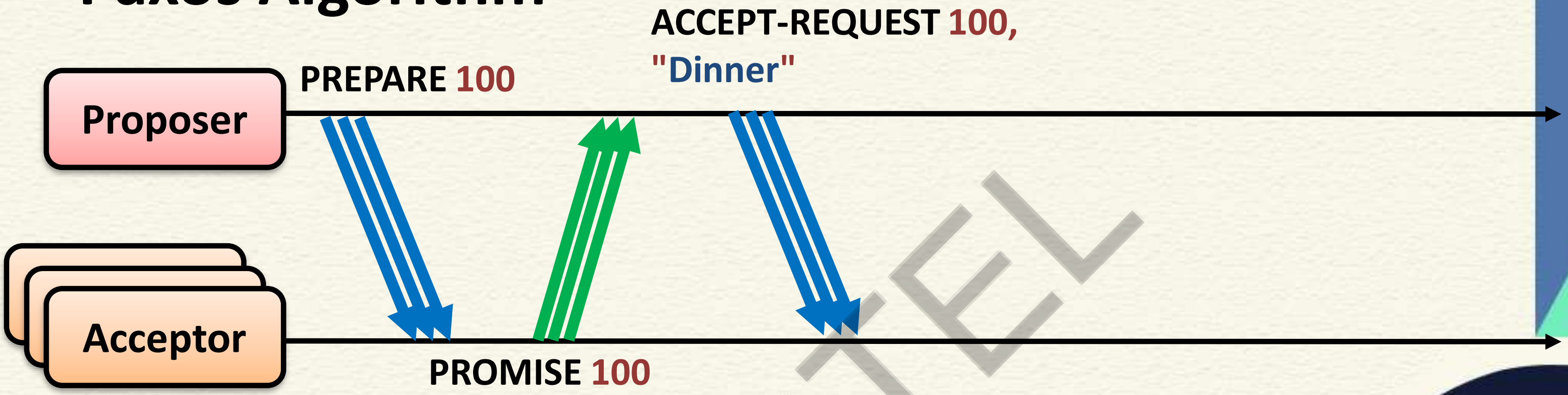
- **Acceptor** received a PREPARE message with IDp:
  - Did it promised to ignore requests with this IDp?
    - **YES:** Ignore
    - **NO:** Will promise to ignore any request lower than IDp
      - (?) Reply with PROMISE IDp

Paxos Algorithm



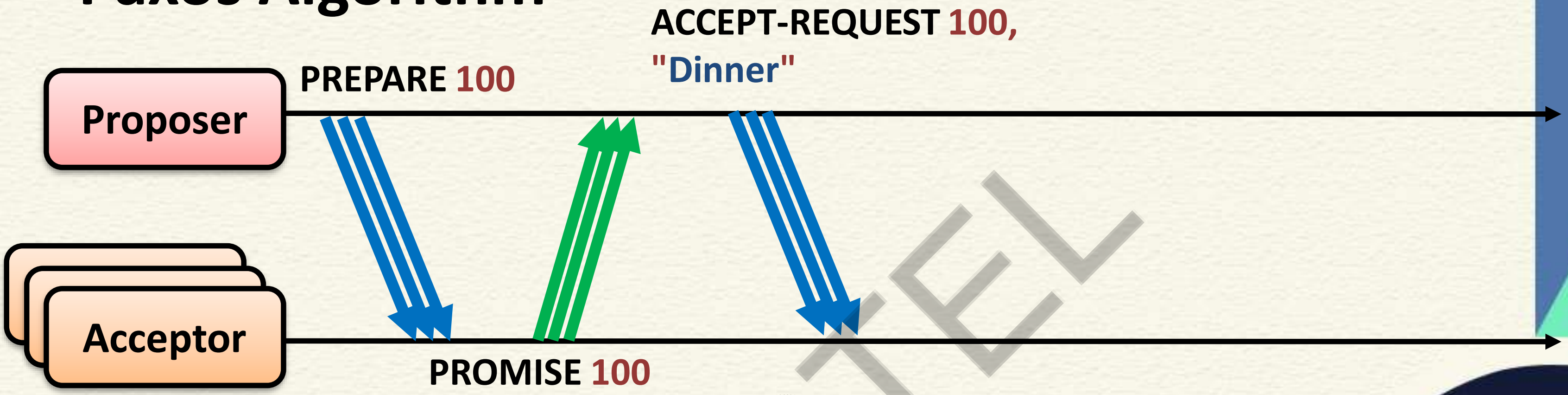
- **Proposer** gets majority of PROMISE messages for a specific IDp:
  - Sends **ACCEPT-REQUEST IDp, <u>VALUE</u>** to a majority (or all) of **Acceptors**
    - (?) It picks any value of its choice

Paxos Algorithm



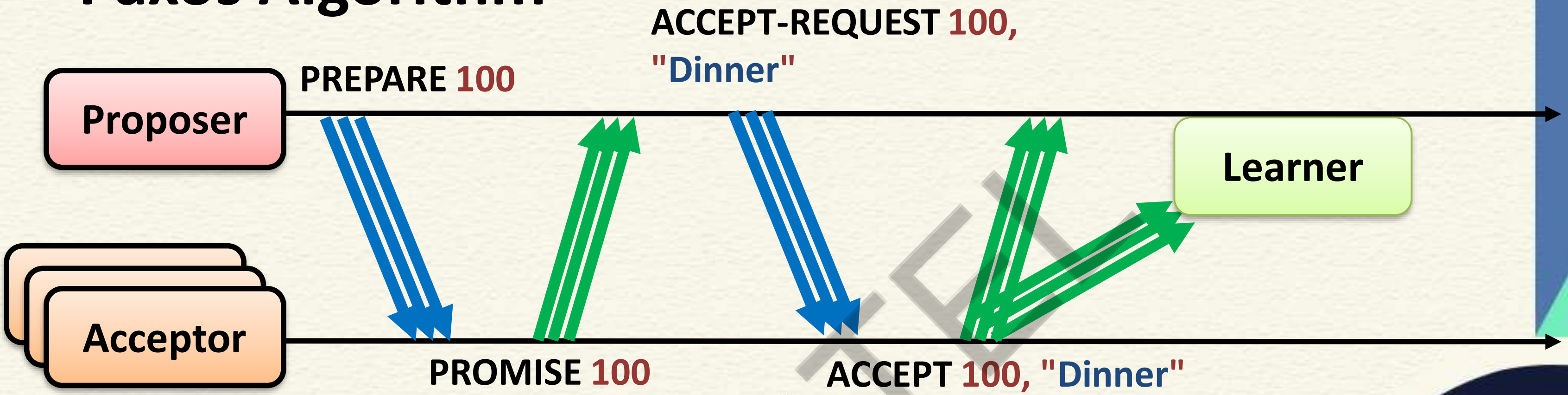
- **Proposer** gets majority of PROMISE messages for a specific IDp:
  - Sends **ACCEPT-REQUEST IDp**, **VALUE** to a majority (or all) of **Acceptors**
    - (?) It picks any value of its choice

Paxos Algorithm



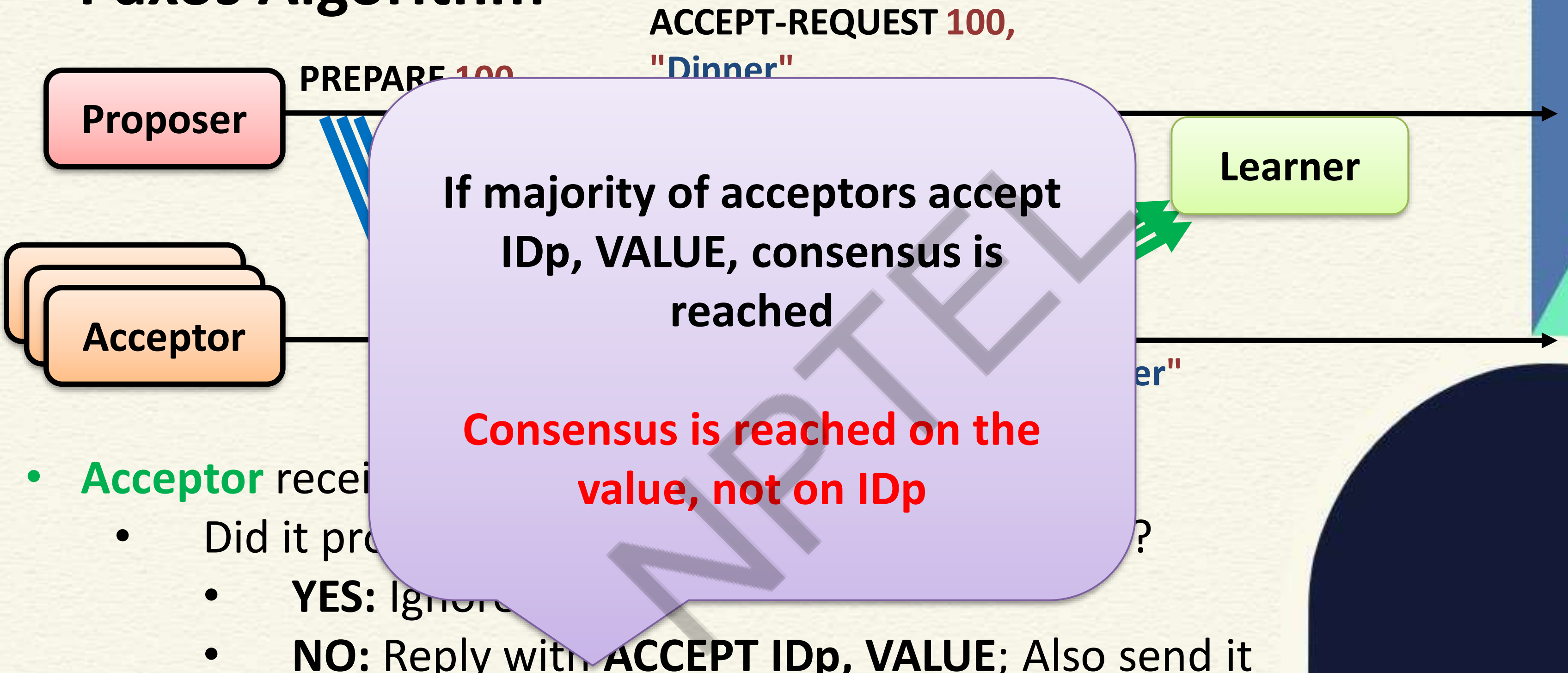
- **Acceptor** receives an ACCEPT-REQUEST IDp, VALUE :
  - Did it promised to ignore request with this IDp?
    - **YES:** Ignore
    - **NO:** Reply with **ACCEPT IDp, VALUE**; Also send it to all learners

Paxos Algorithm



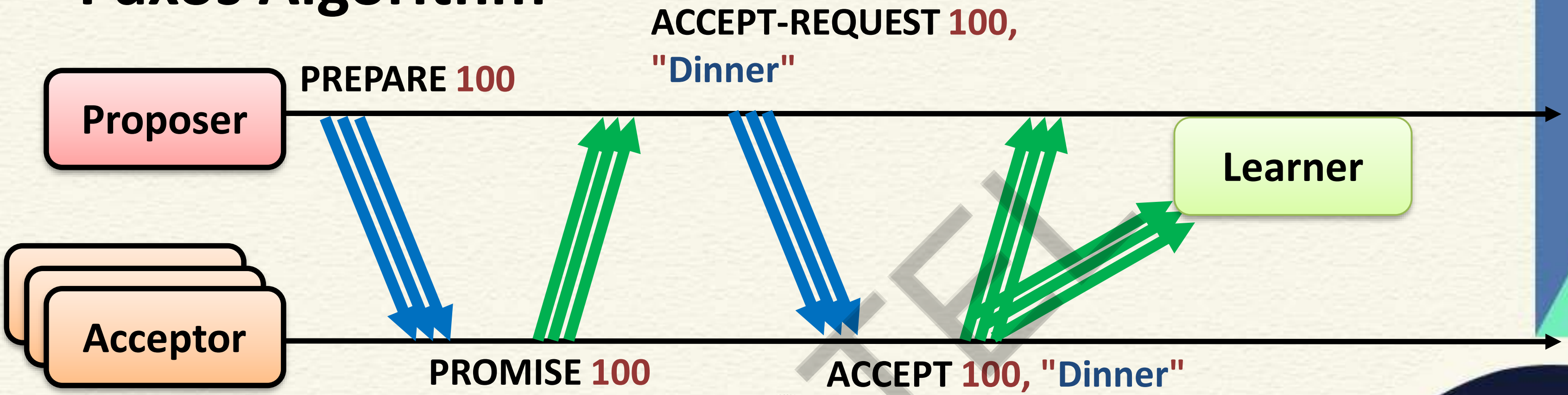
- **Acceptor** receives an ACCEPT-REQUEST IDp, VALUE :
  - Did it promised to ignore request with this IDp?
    - **YES:** Ignore
    - **NO:** Reply with **ACCEPT IDp, <u>VALUE</u>**; Also send it to all learners

Paxos Algorithm



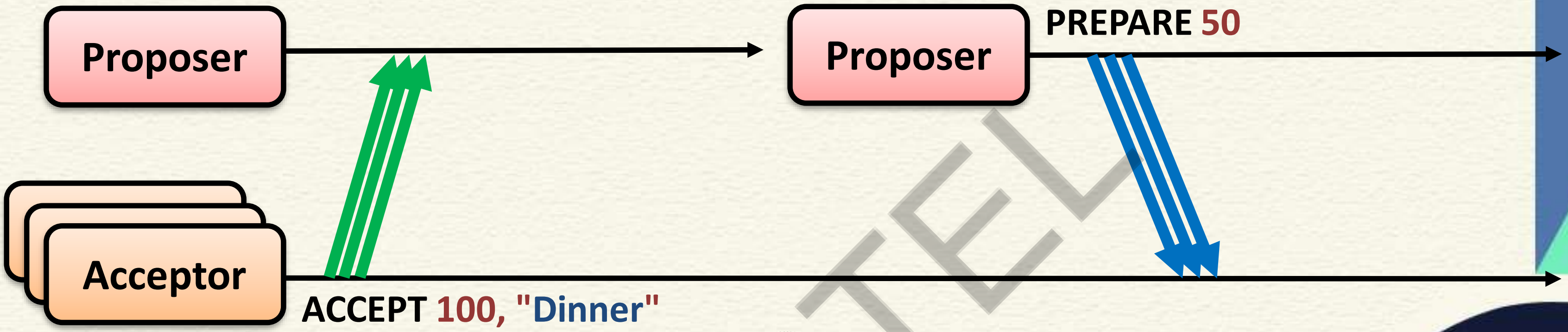
- **Acceptor** recei
  - Did it pr ?
    - **YES:** Ignore
    - **NO:** Reply with **ACCEPT IDp, VALUE**; Also send it to all learners

Paxos Algorithm

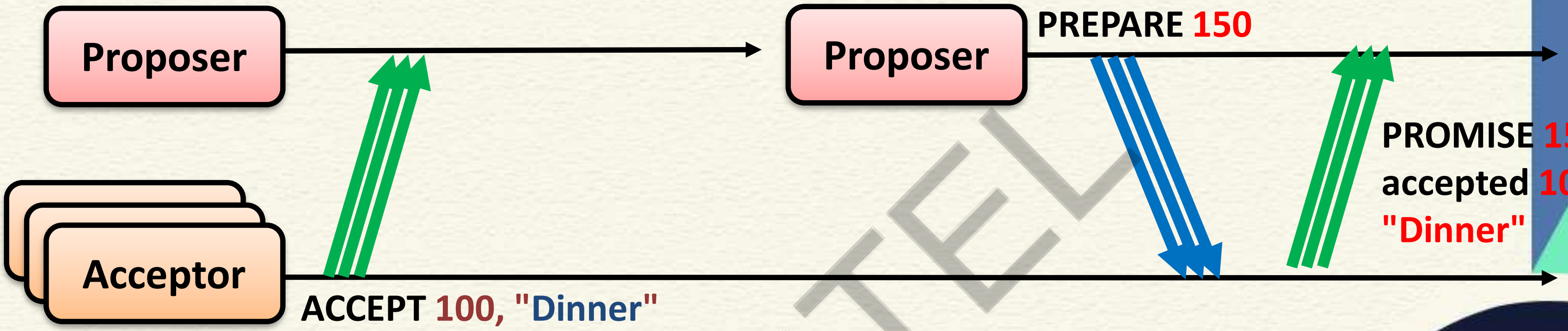


- **Proposer** or **Learner** gets ACCEPT message with IDp, VALUE:
  - If a proposer/learner gets majority of accept for a specific IDp, they know that consensus is reached for the value (not IDp).

Paxos – Multiple Proposers

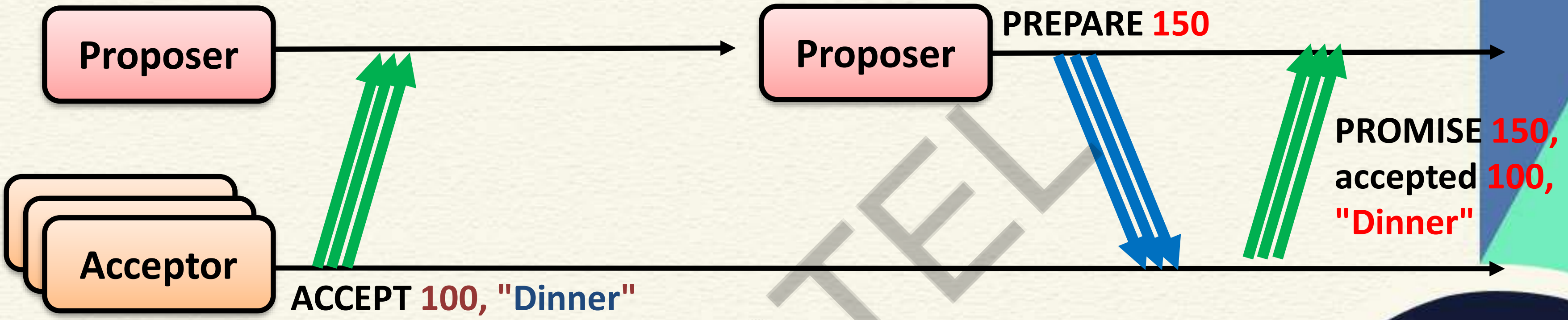


Paxos – Multiple Proposers



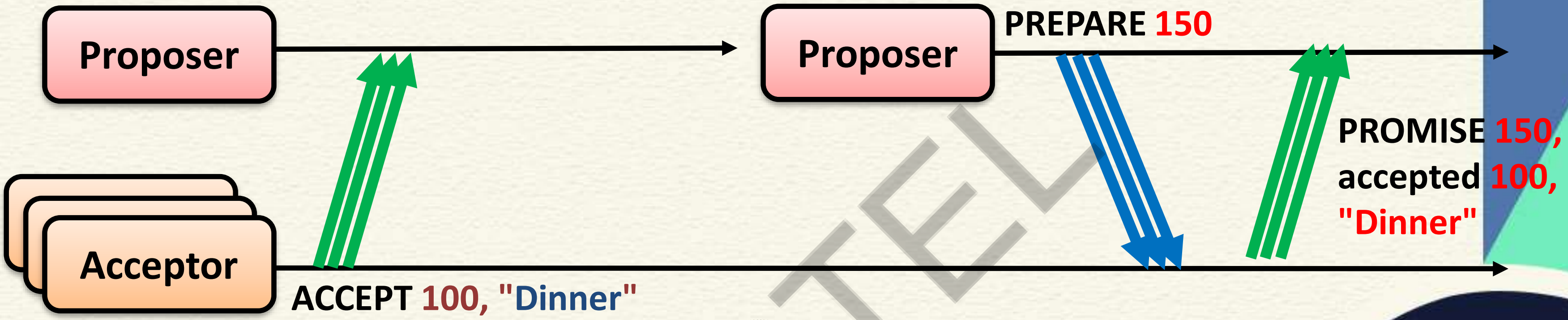
- **Acceptor** received a PREPARE message with IDp:
  - Did it promised to ignore requests with this IDp?
    - **YES:** Ignore
    - **NO:** Will promise to ignore any request lower than IDp
      - Has it ever accepted anything? (Assume accepted ID = IDa)
        - **YES:** Reply with **PROMISE IDp accepted IDa, <u>VALUE</u>**
        - **NO:** Reply with **PROMISE IDp**

Paxos – Multiple Proposers



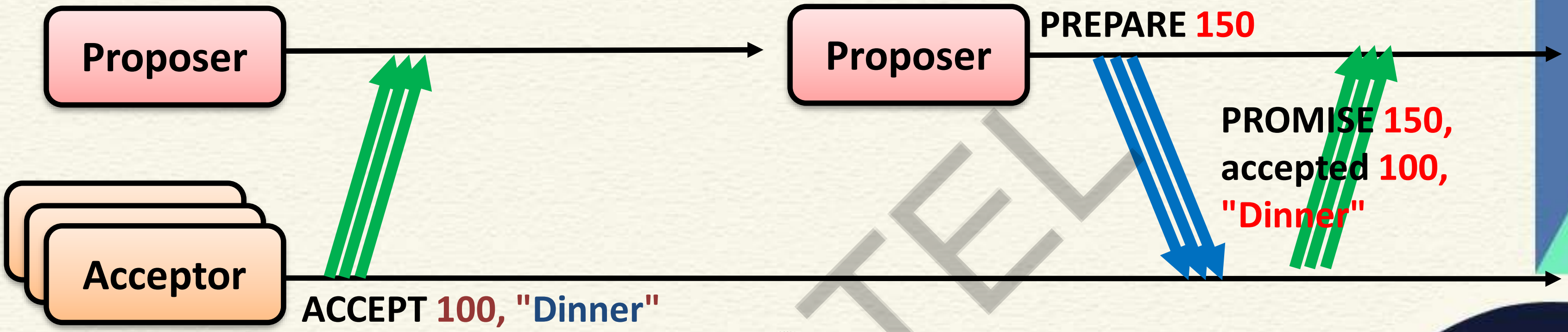
What the proposer will do?

Paxos – Multiple Proposers



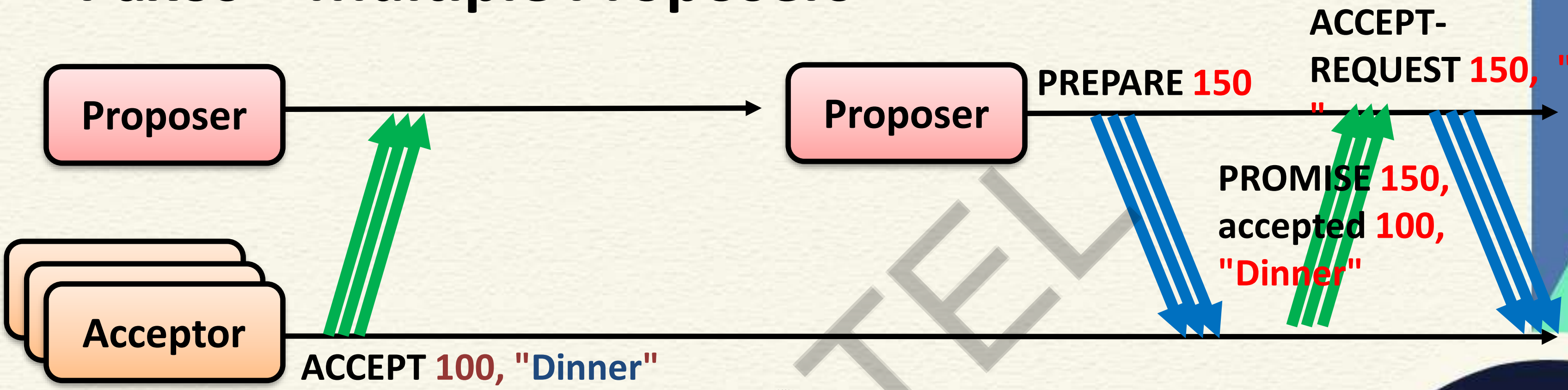
- **Proposer** gets majority of PROMISE messages for a specific IDp:
  - It sends **ACCEPT-REQUEST IDp**, **VALUE** to a majority (or all) of **Acceptors**
    - (?) It picks any value it wants

Paxos – Multiple Proposers



- **Proposer** gets majority of PROMISE messages for a specific IDp:
  - It sends **ACCEPT-REQUEST IDp**, **VALUE** to a majority (or all) of **Acceptors**
    - Has it got any already accepted value from promises?
      - **YES:** Picks the value with the highest IDa
      - **NO:** Picks the value of its choice

Paxos – Multiple Proposers



- **Proposer** gets majority of PROMISE messages for a specific IDp:
  - It sends **ACCEPT-REQUEST IDp**, **VALUE** to a majority (or all) of **Acceptors**
    - Has it got any already accepted value from promises?
      - **YES:** Picks the value with the highest IDa
      - **NO:** Picks the value of its choice

Conclusion

- Paxos works in two rounds
  - Agreement on the state (ID)
  - Agreement on the value

- **Safety and liveness of Paxos?**

Thank you

NPTEL ONLINE CERTIFICATION COURSES

Blockchain and its applications

Prof. Sandip Chakraborty

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Lecture 29: Paxos – Safety and Liveness

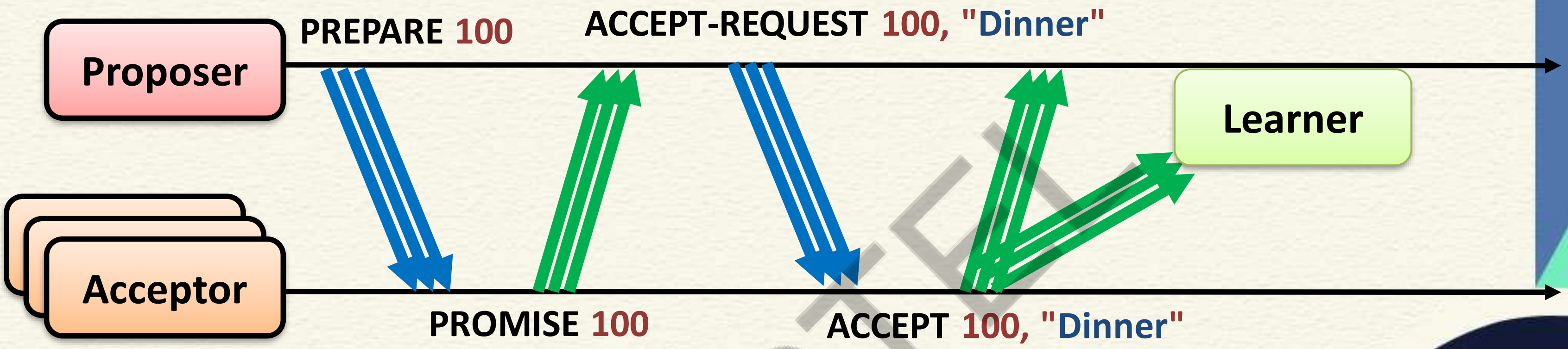
CONCEPTS COVERED

- **Safety and Liveness of Paxos**

KEYWORDS

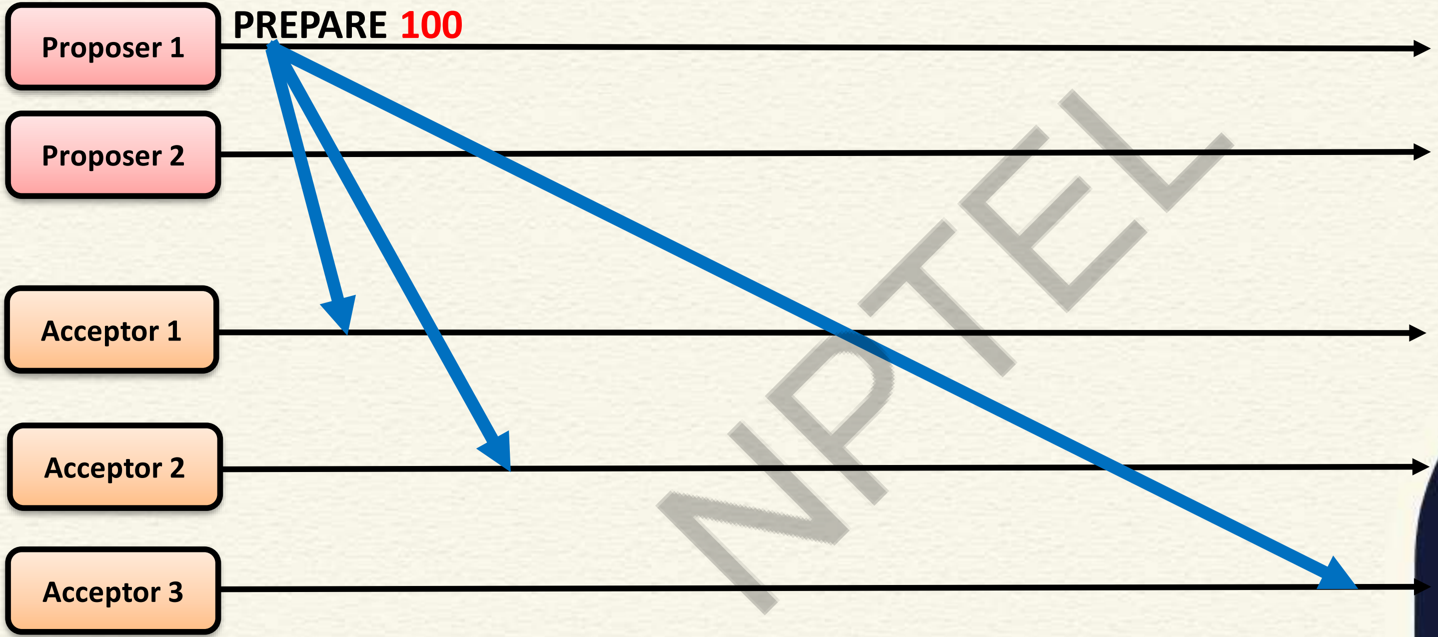
- **Paxos: Correctness**
- **Leader Election**
- **Multi-Paxos**

Paxos – Message Exchanges

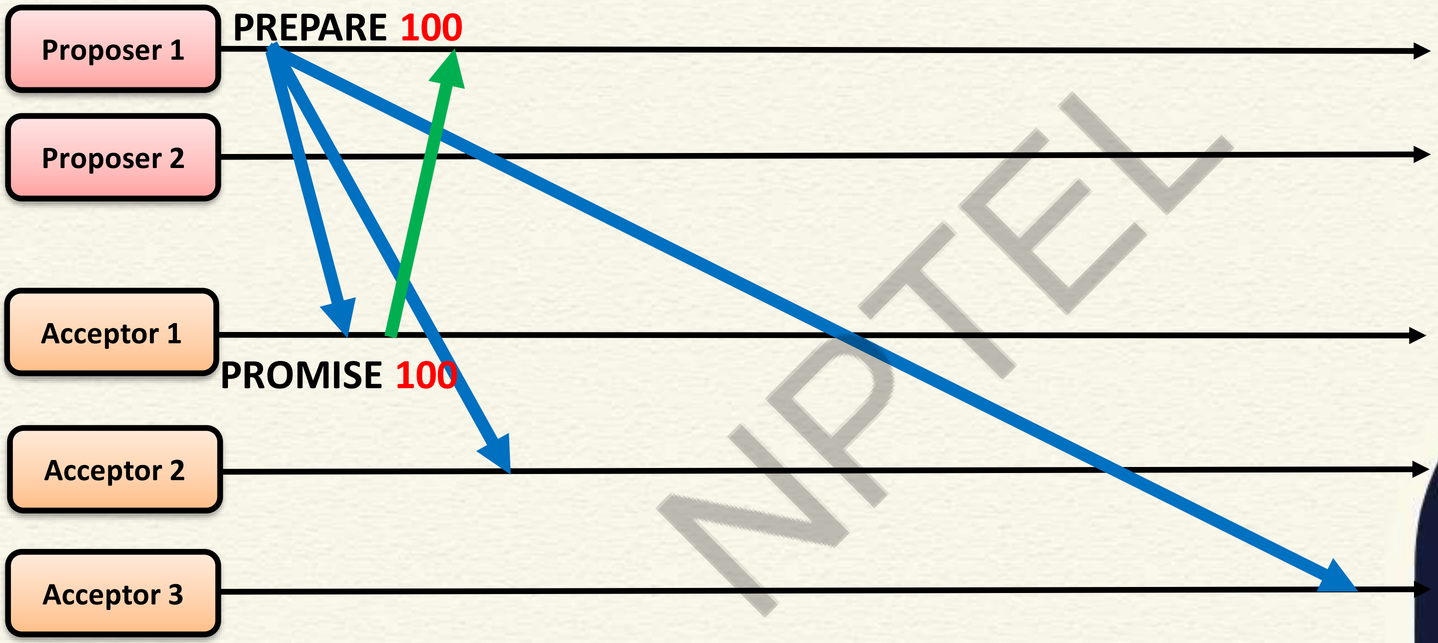


- Two rounds of message exchanges
  - PREPARE – PROMISE: Agree on a state (ID)
  - ACCEPT-REQUEST – ACCEPT: Agree on a value
- The consensus is on the "***value***"

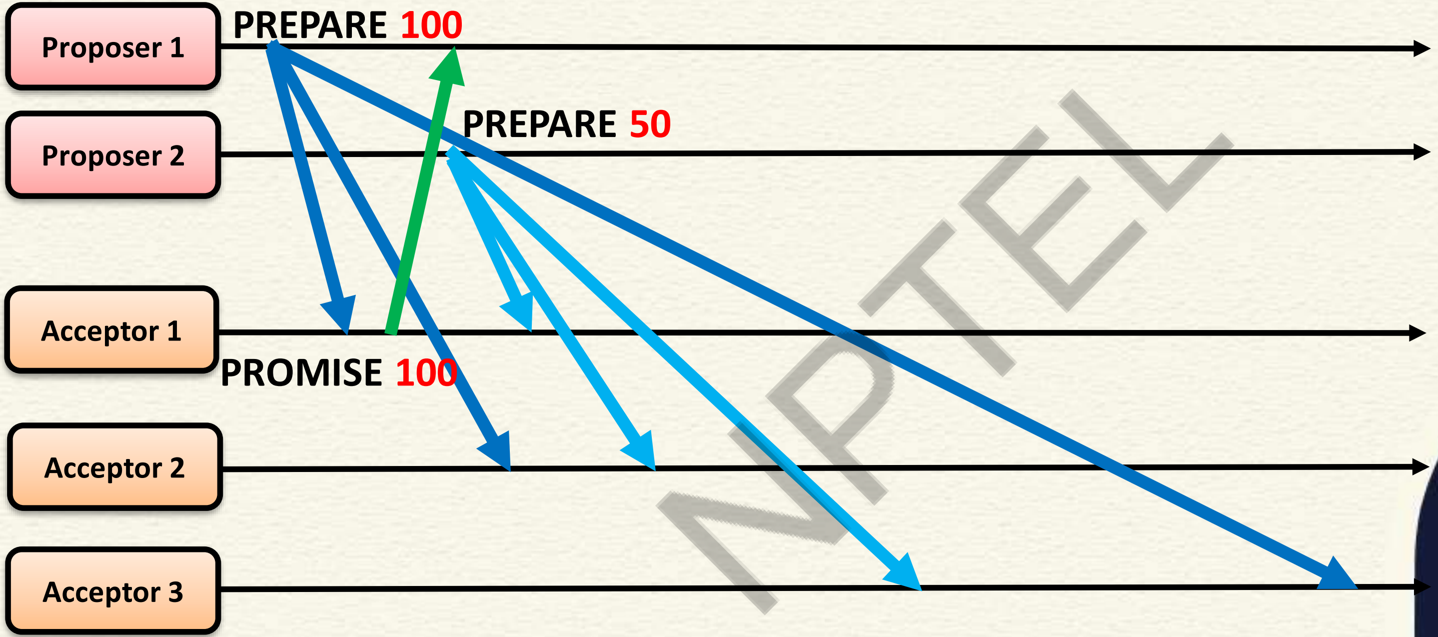
Majority Voting



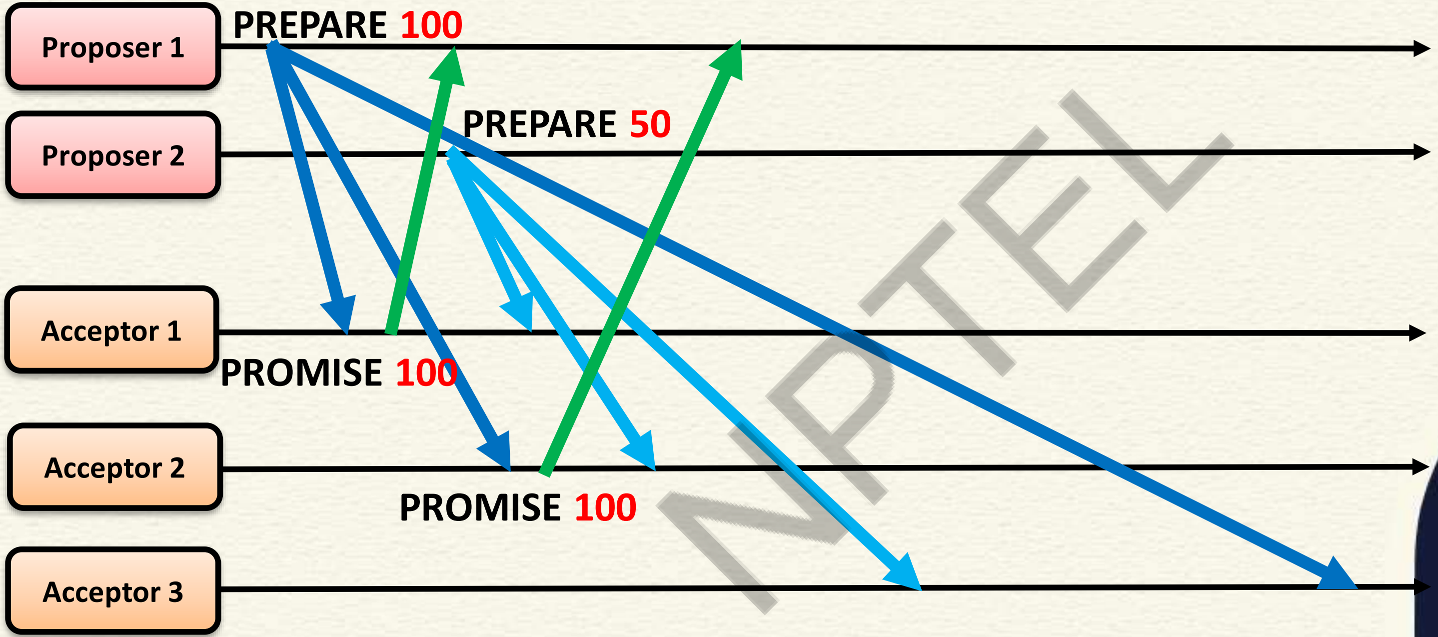
Majority Voting



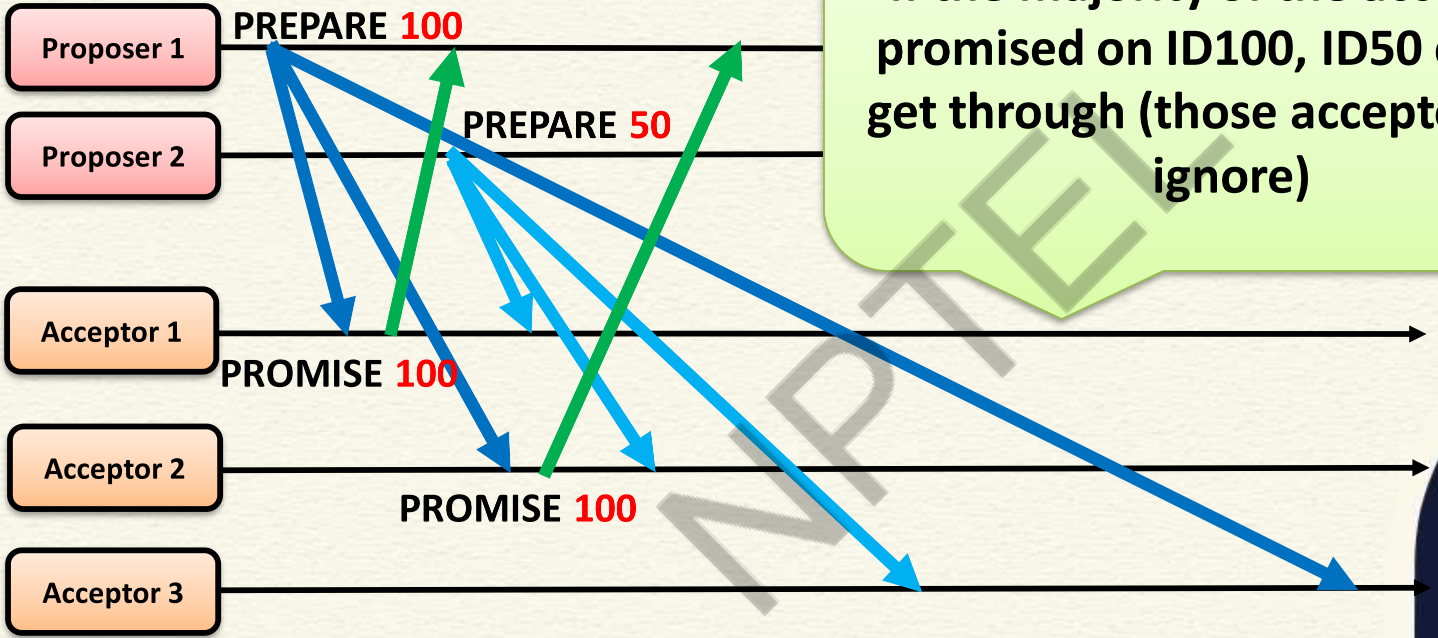
Majority Voting



Majority Voting

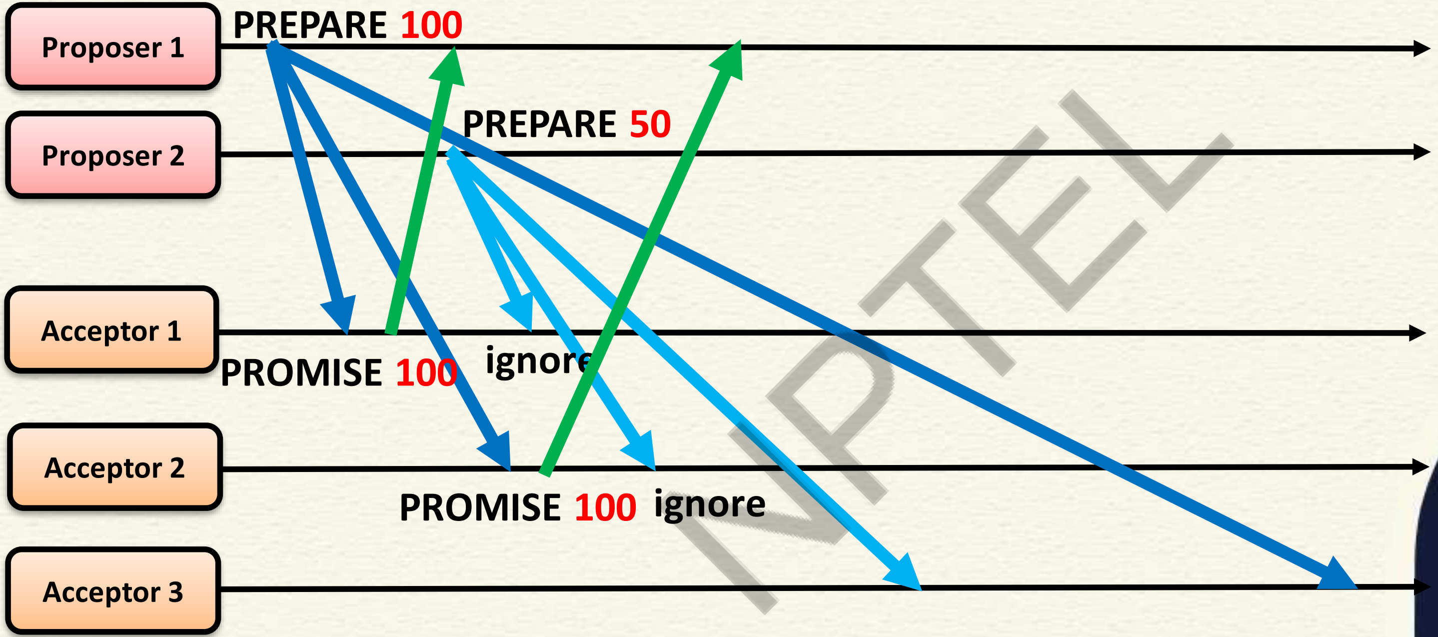


Majority Voting

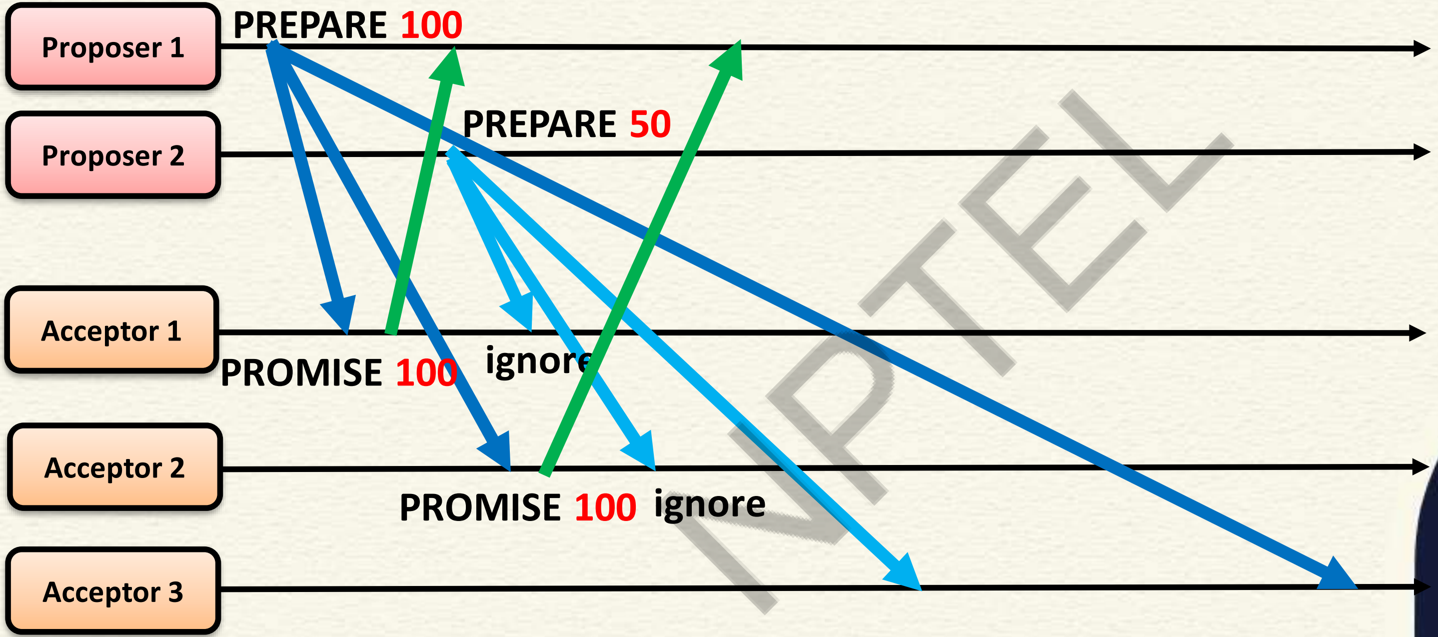


If the majority of the acceptors promised on ID100, ID50 cannot get through (those acceptors will ignore)

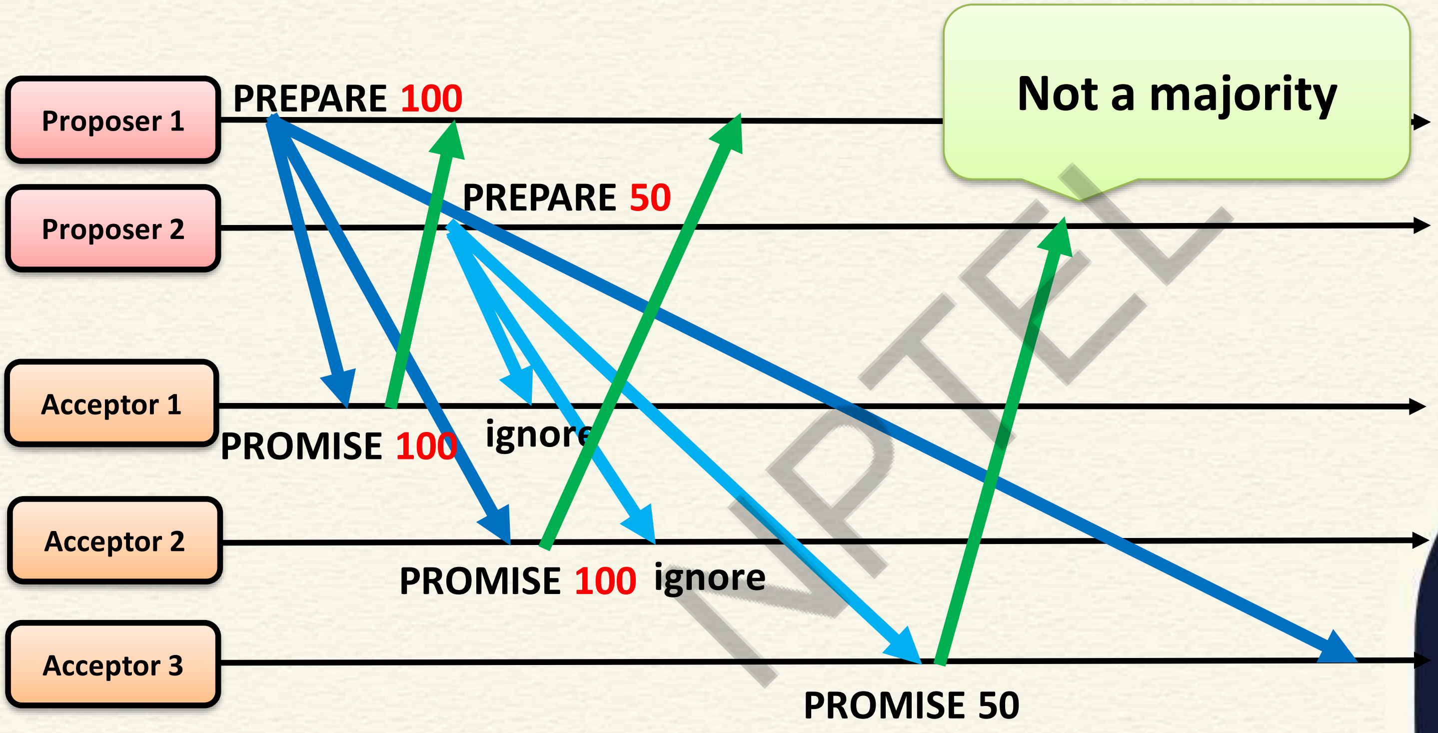
Majority Voting



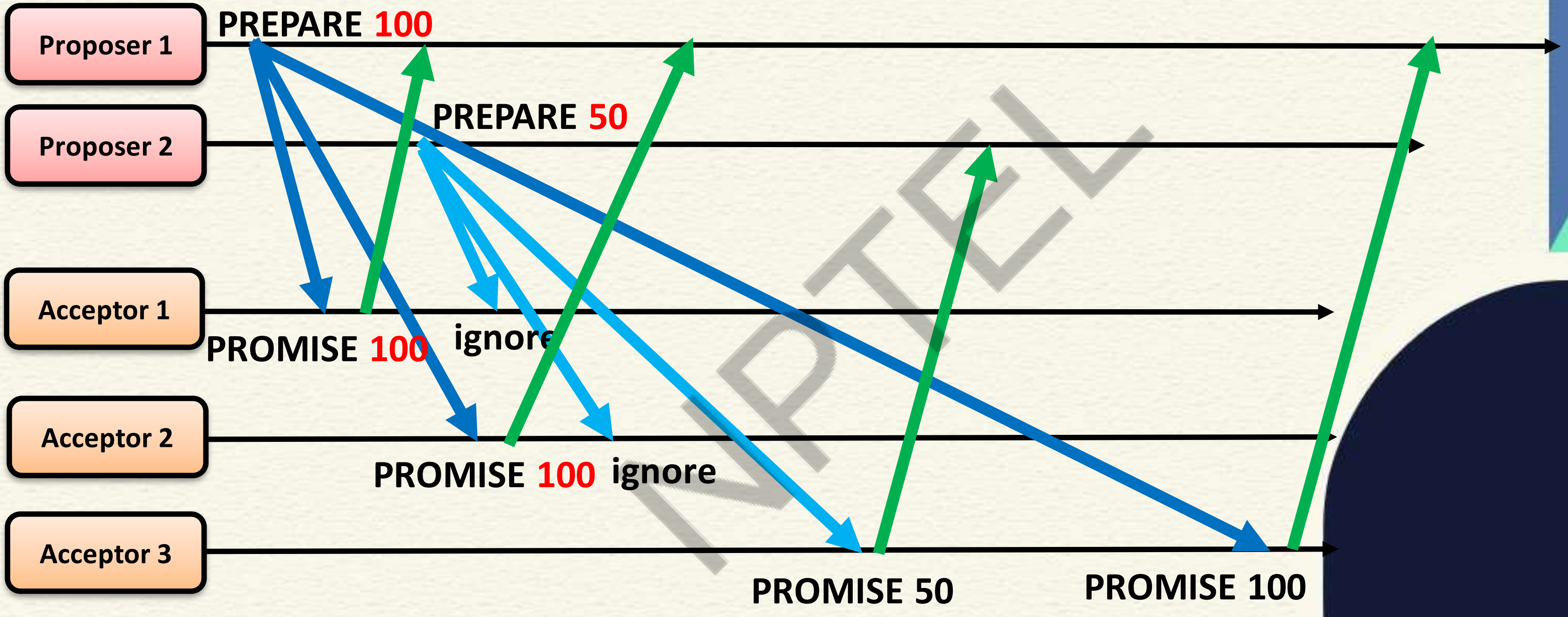
Majority Voting



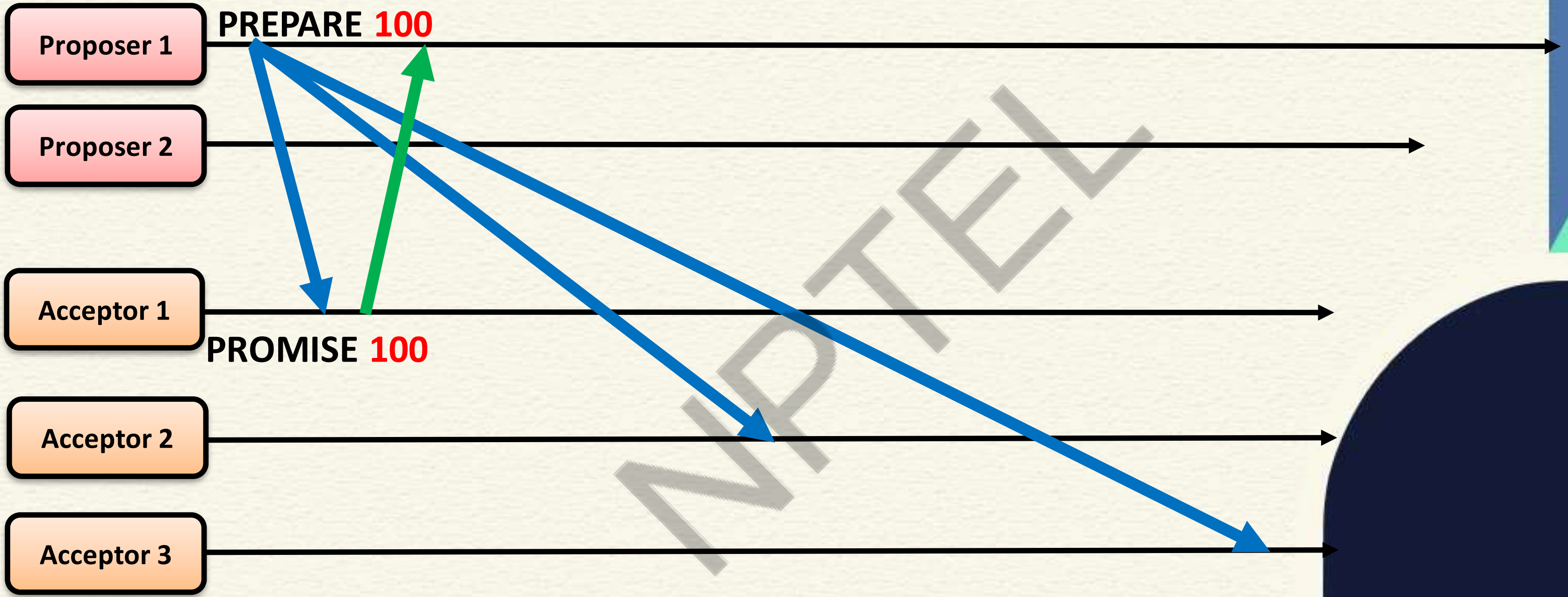
Majority Voting



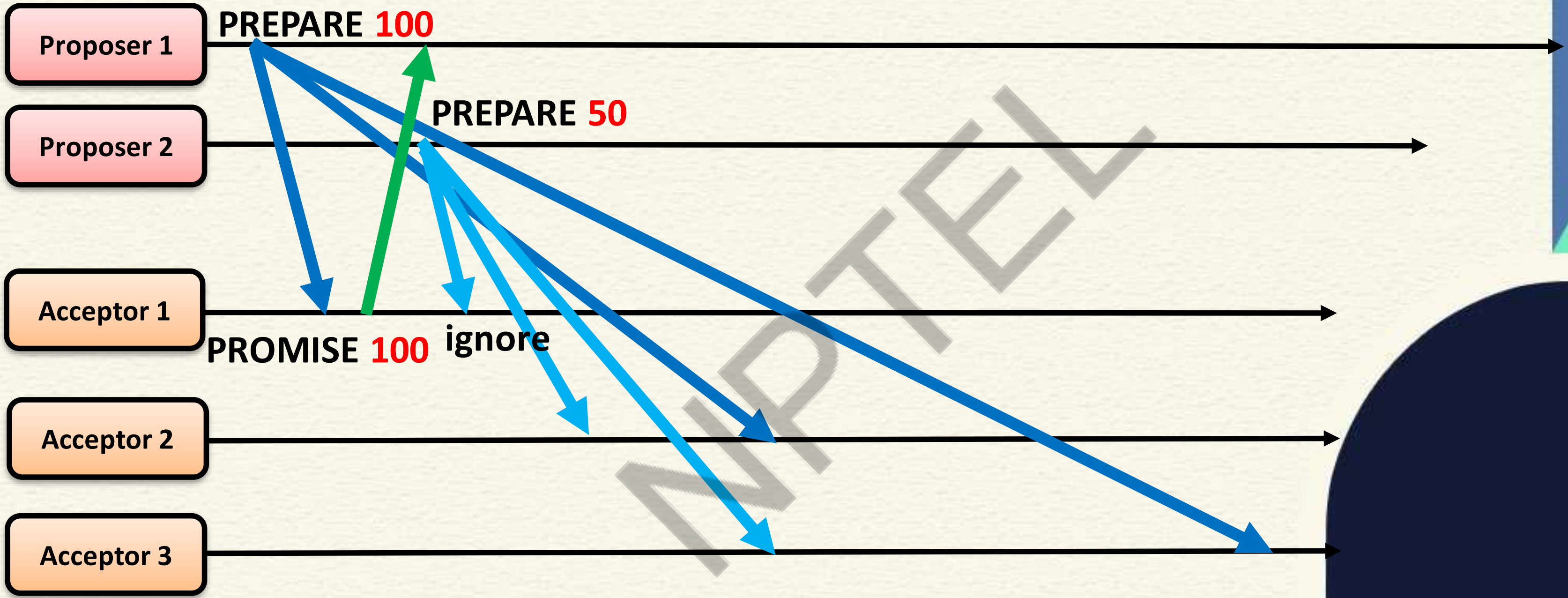
Majority Voting



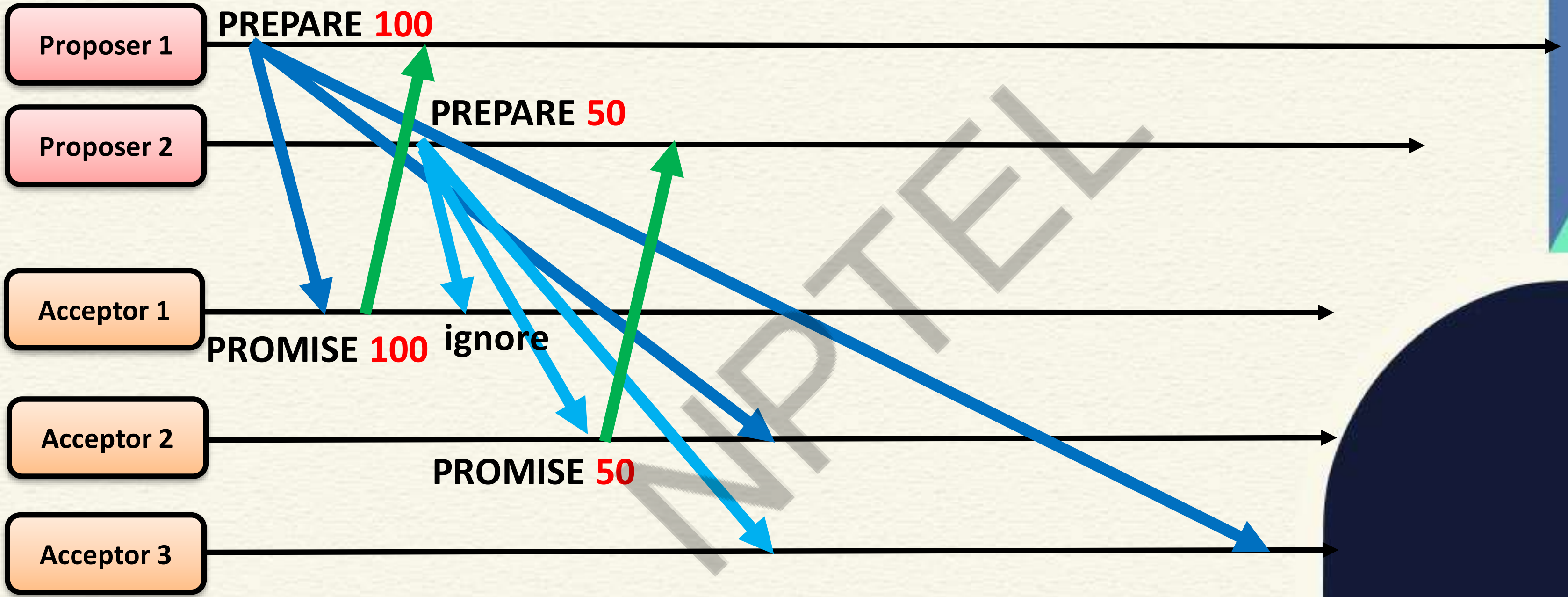
Majority Voting – Case 2



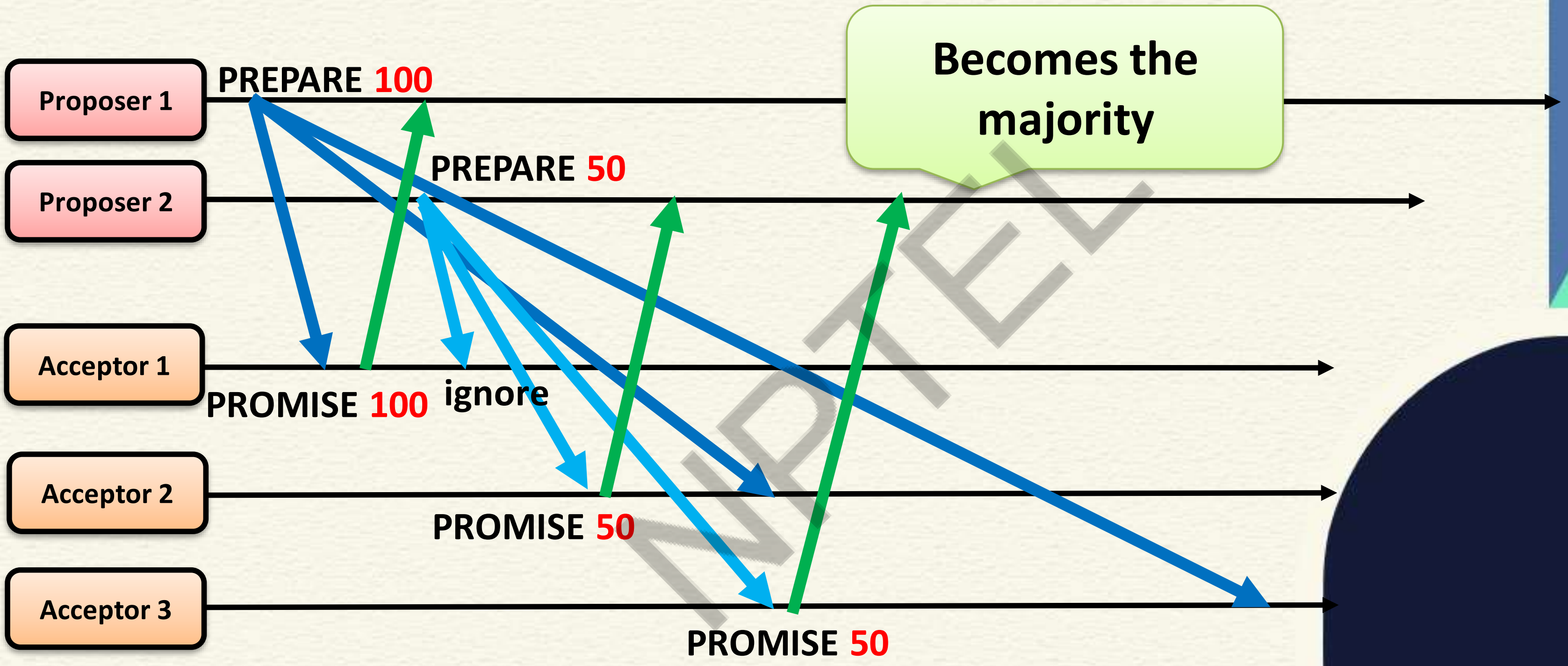
Majority Voting – Case 2



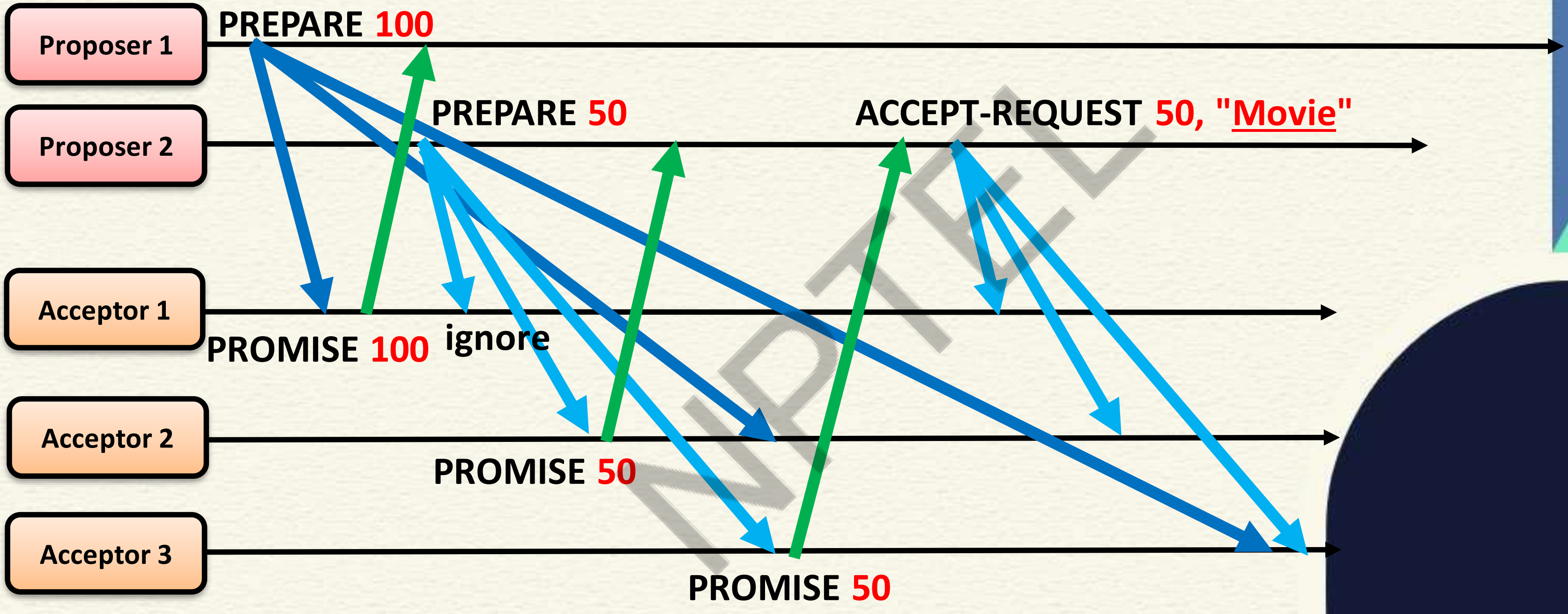
Majority Voting – Case 2



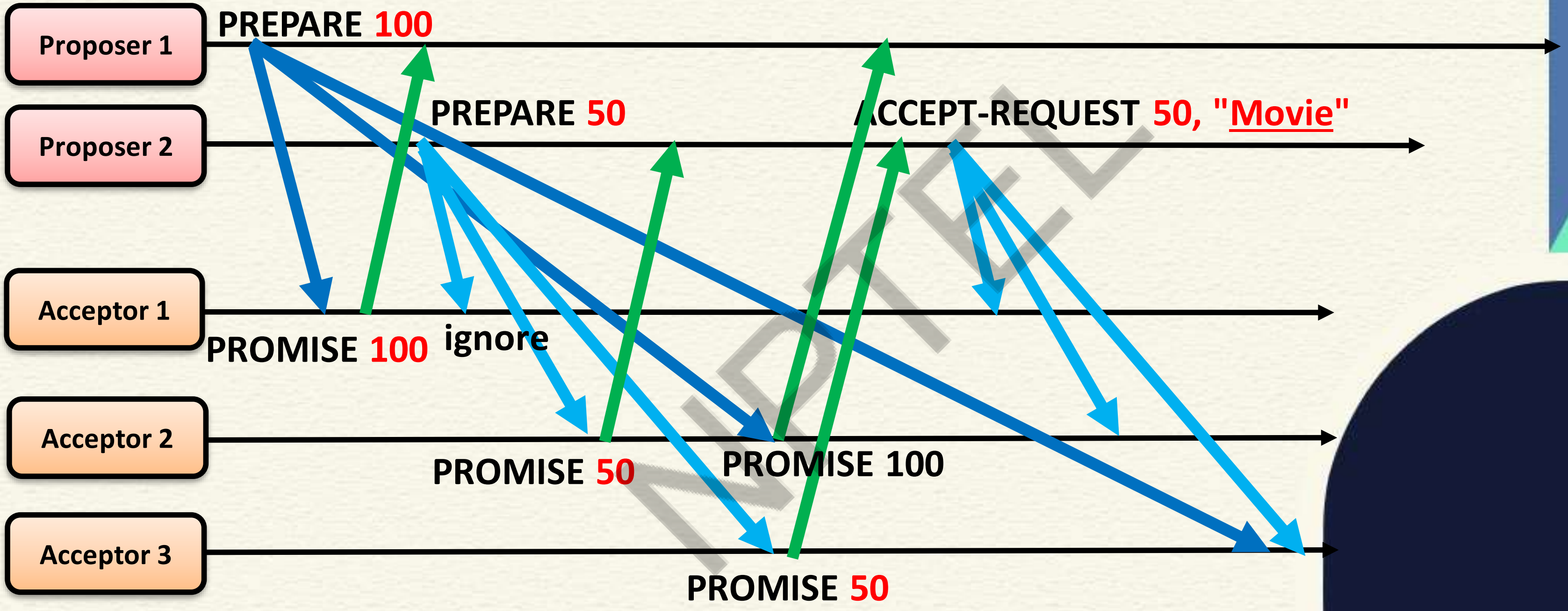
Majority Voting – Case 2



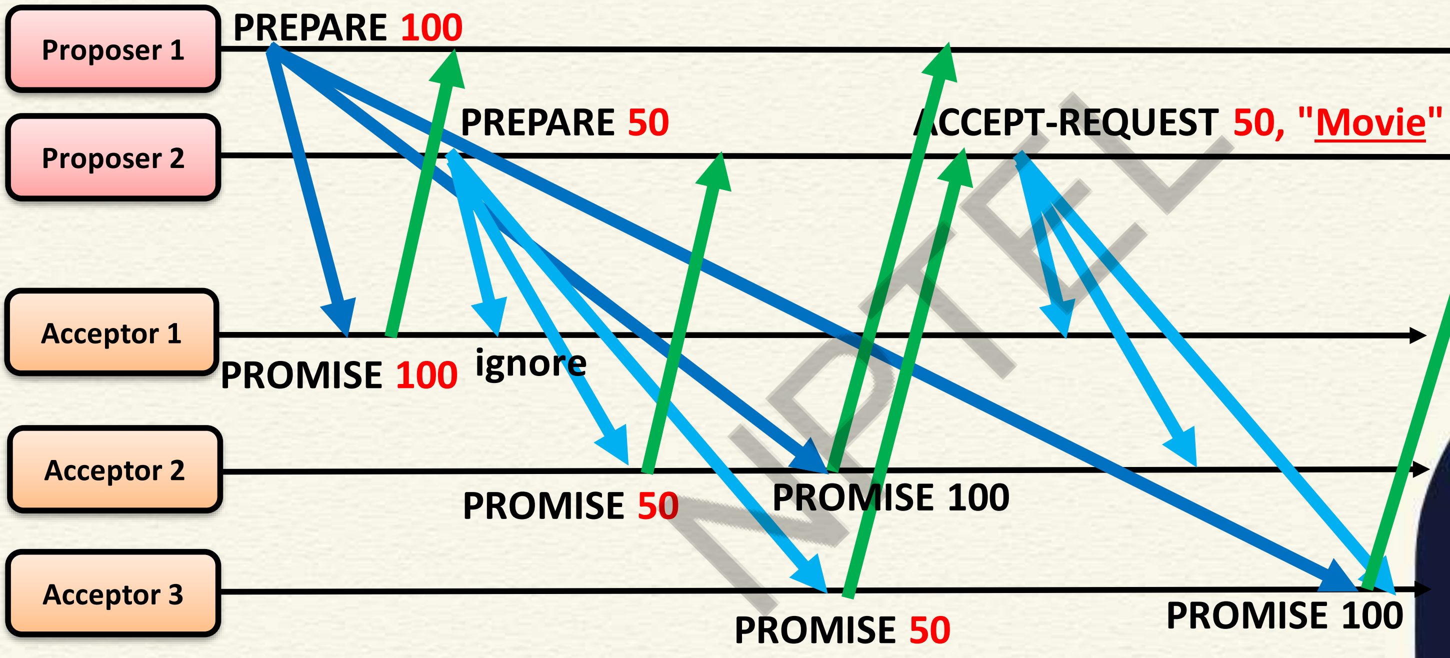
Majority Voting – Case 2



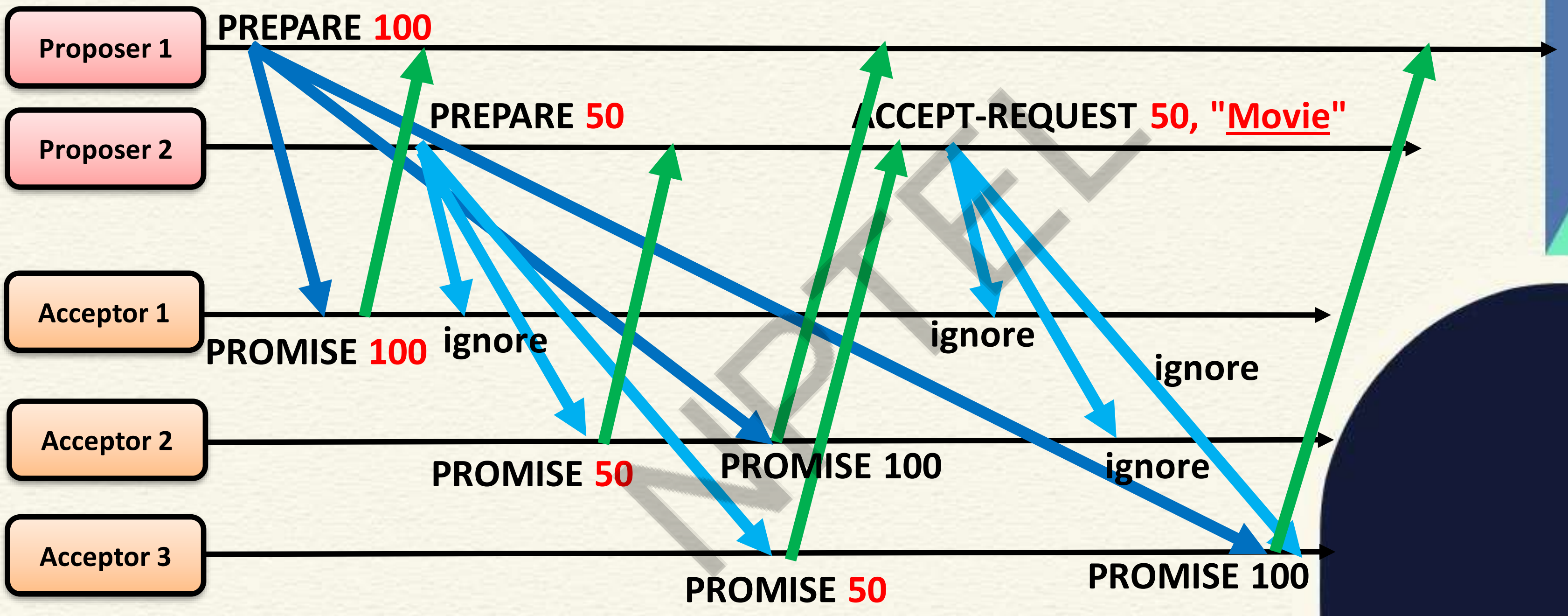
Majority Voting – Case 2



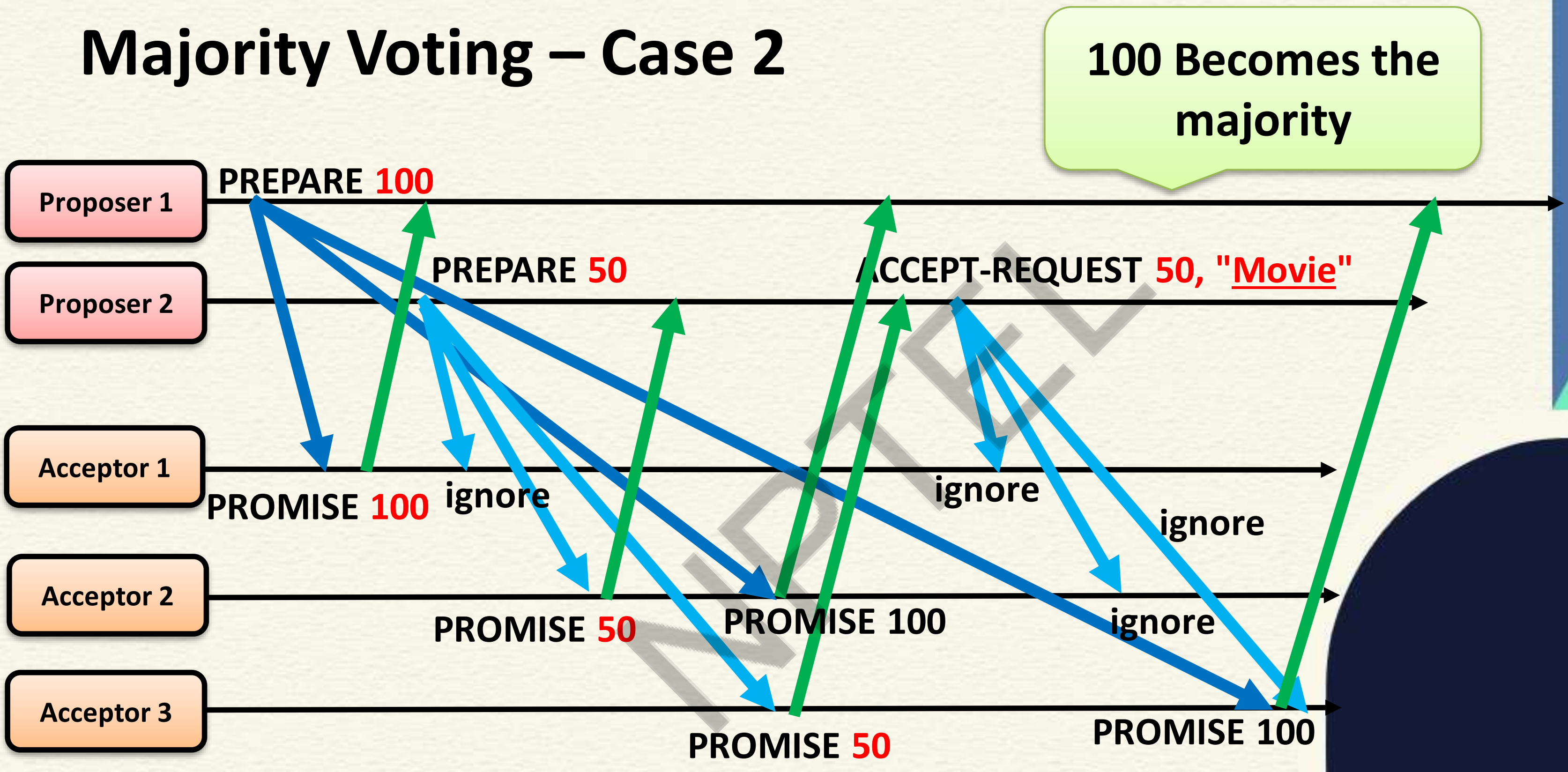
Majority Voting – Case 2



Majority Voting – Case 2



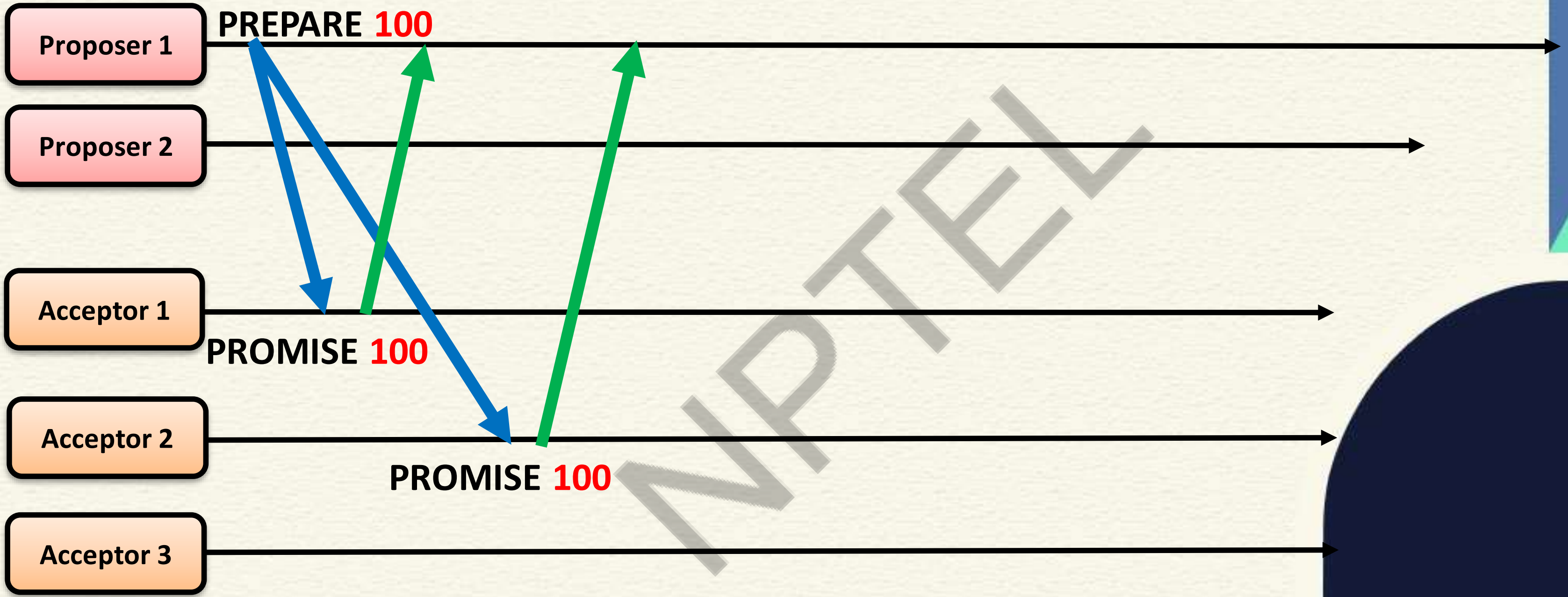
Majority Voting – Case 2



Liveness

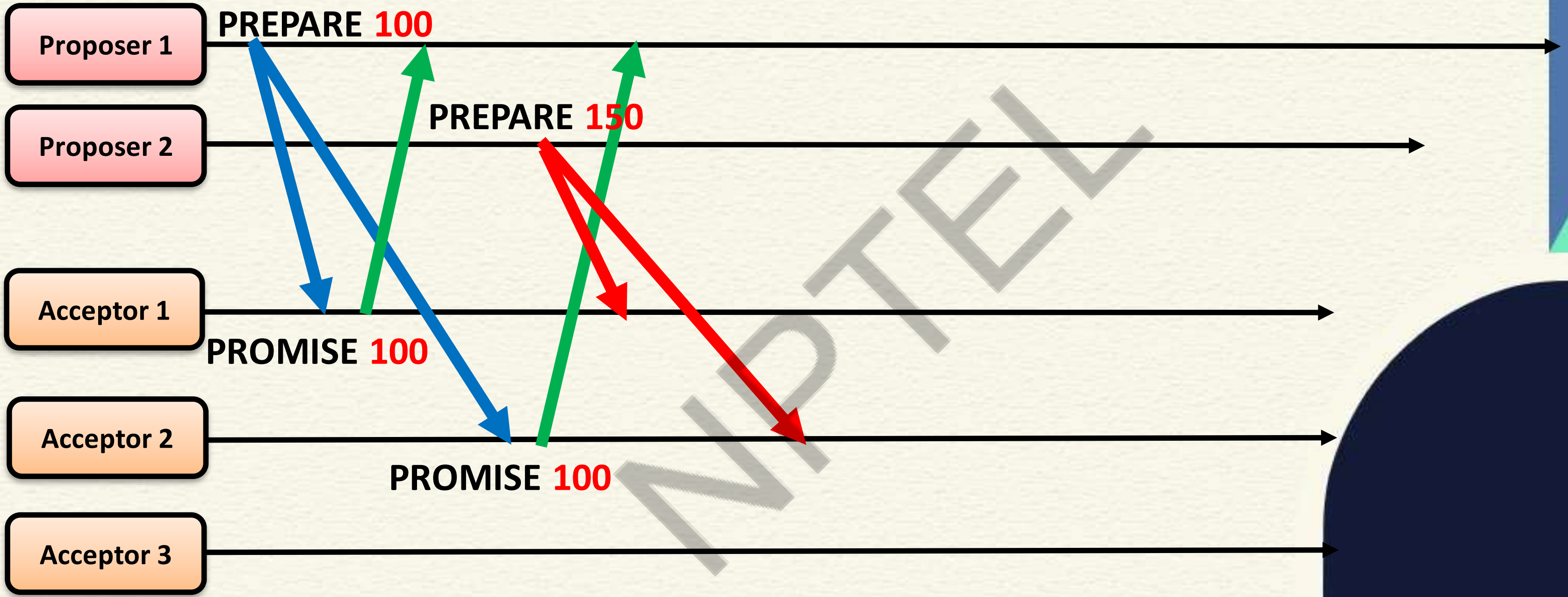
PREPARE 100
Proposer 1
Proposer 2
Acceptor 1
Acceptor 2
Acceptor 3

Liveness



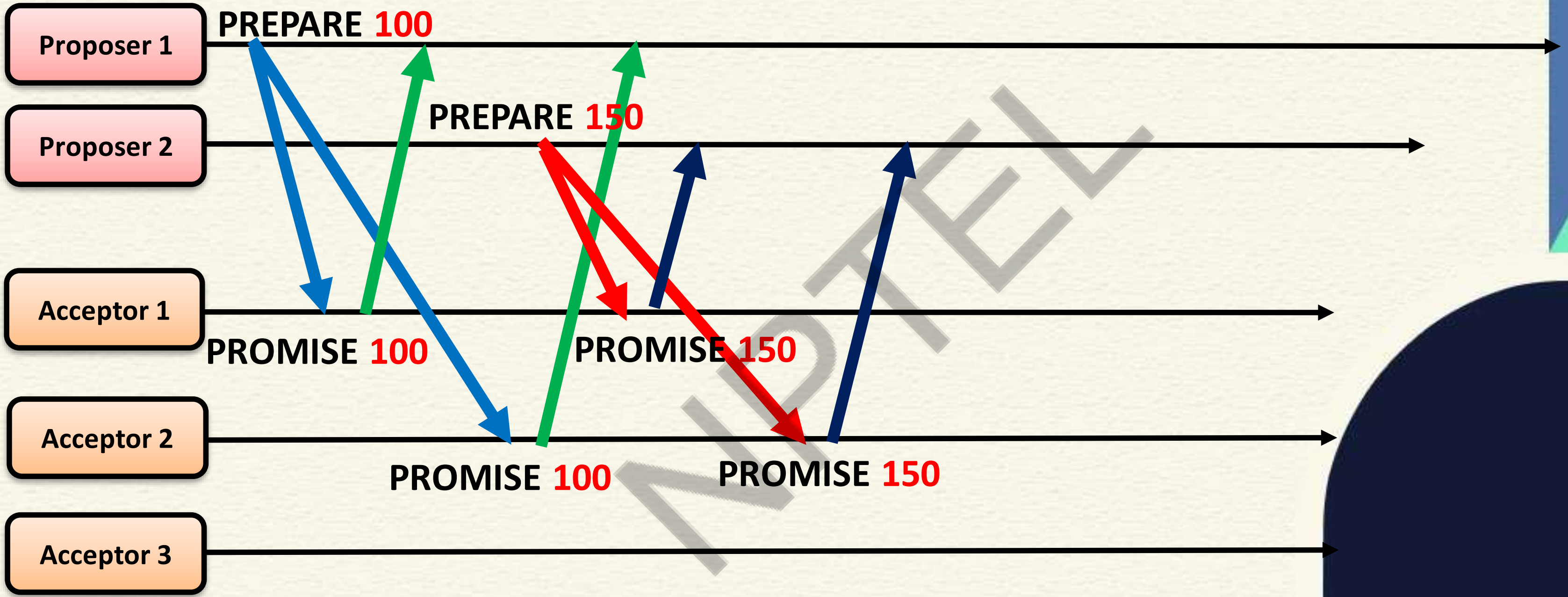
Proposer 1
Proposer 2
Acceptor 1
Acceptor 2
Acceptor 3
PREPARE 100
PROMISE 100
PROMISE 100

Liveness

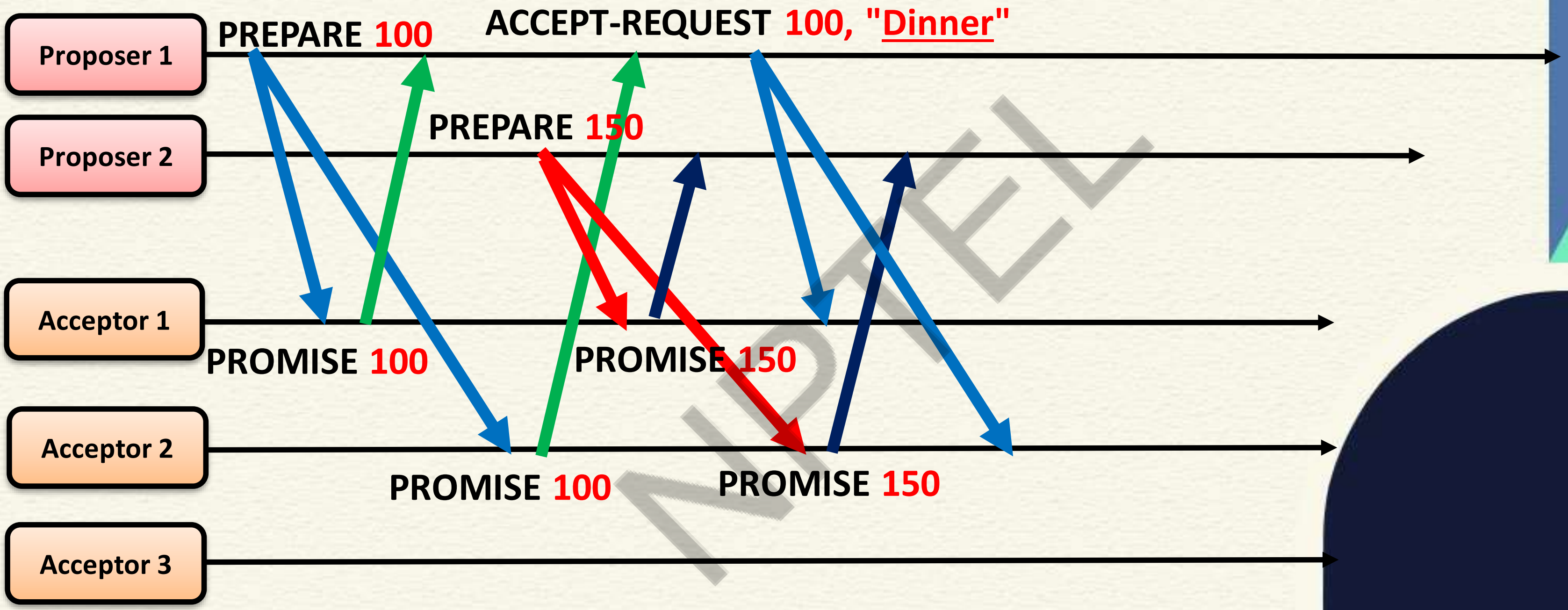


Proposer 1
Proposer 2
Acceptor 1
Acceptor 2
Acceptor 3
PREPARE 100
PREPARE 150
PROMISE 100
PROMISE 100

Liveness

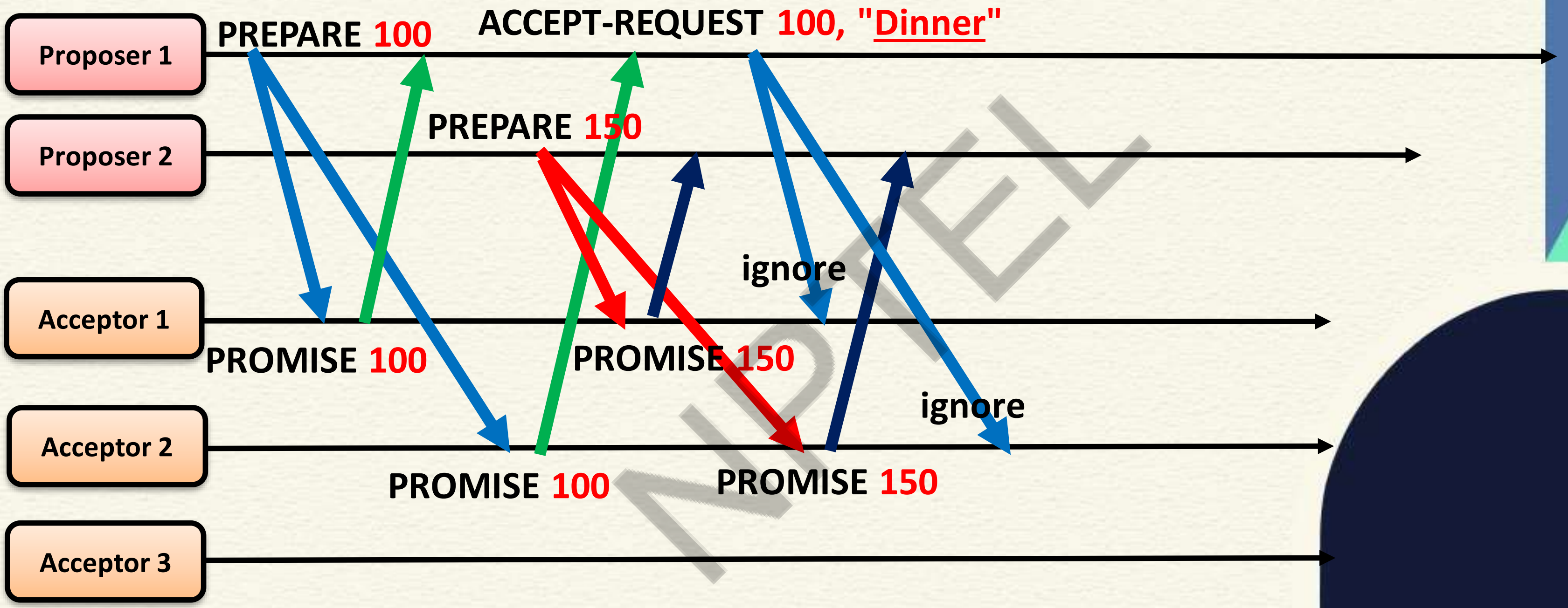

Proposer 1
Proposer 2
Acceptor 1
Acceptor 2
Acceptor 3
PREPARE 100
PREPARE 150
PROMISE 100
PROMISE 150
PROMISE 100
PROMISE 150

Liveness



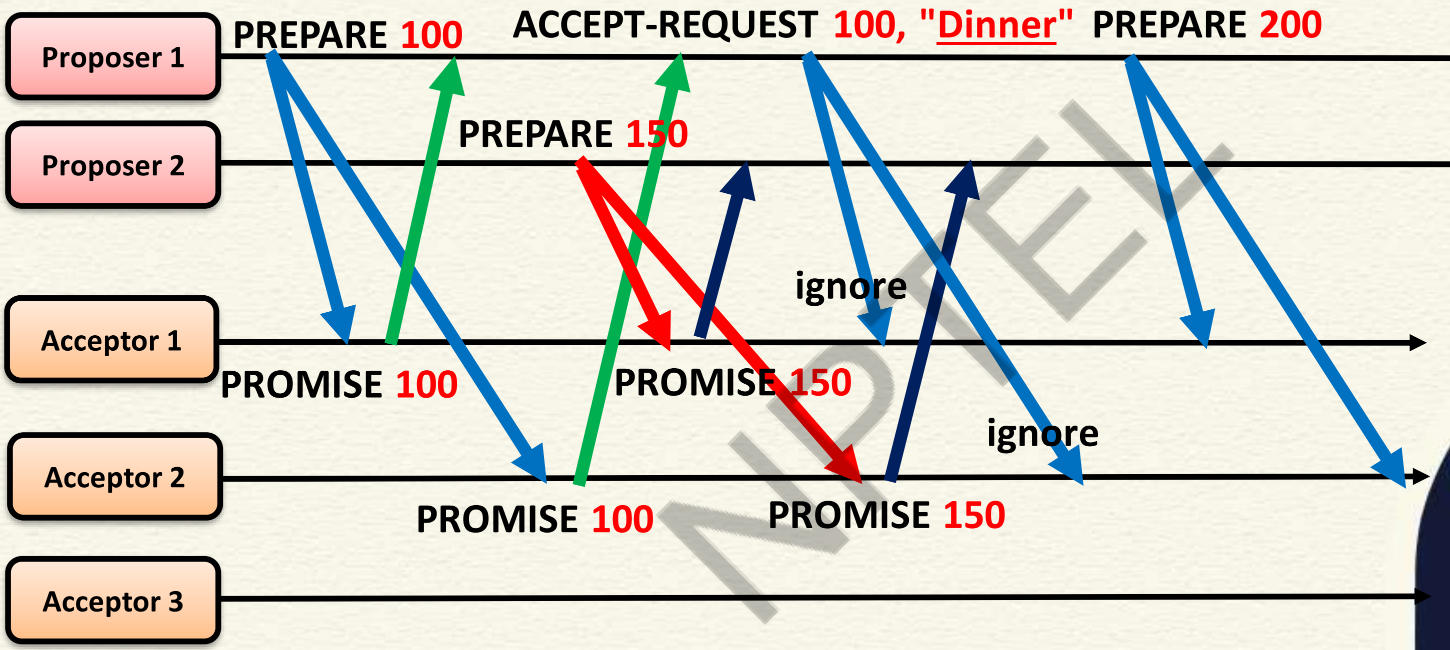
Proposer 1
Proposer 2
Acceptor 1
Acceptor 2
Acceptor 3
PREPARE 100
ACCEPT-REQUEST 100, "Dinner"
PREPARE 150
PROMISE 100
PROMISE 150
PROMISE 100
PROMISE 150

Liveness



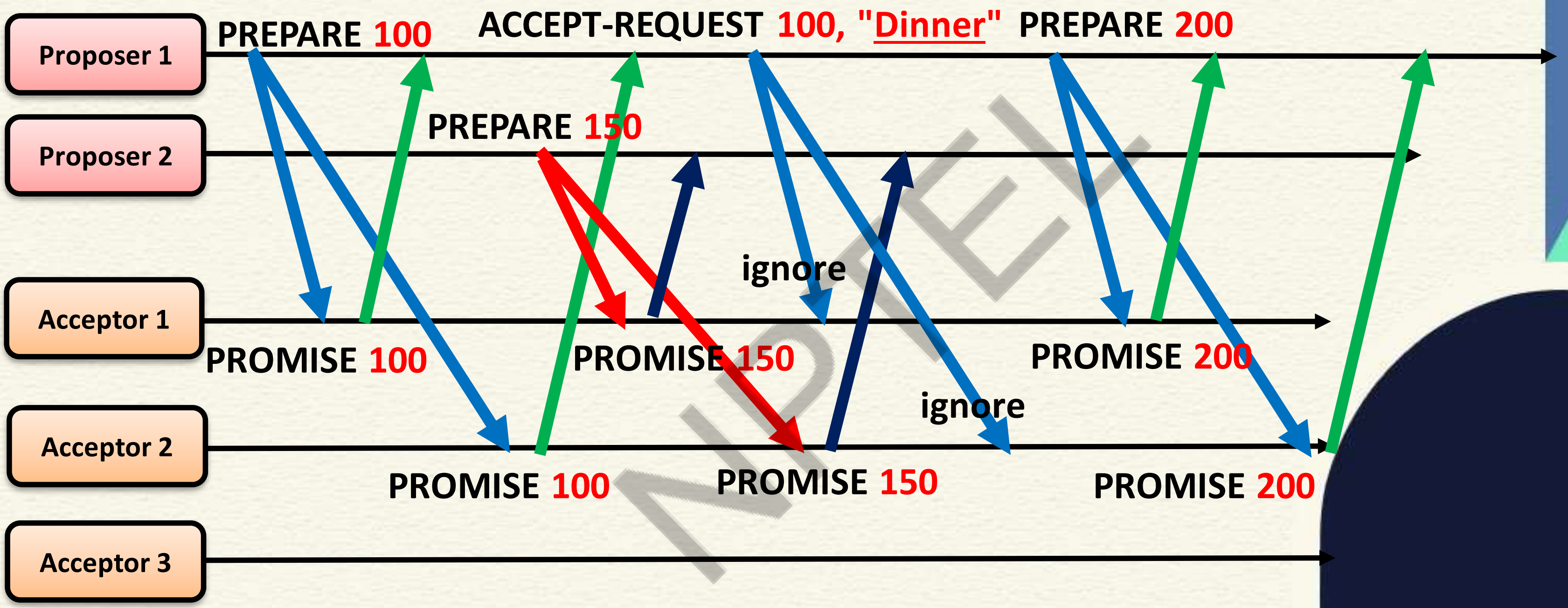
Proposer 1
Proposer 2
Acceptor 1
Acceptor 2
Acceptor 3
PREPARE 100
ACCEPT-REQUEST 100, "Dinner"
PREPARE 150
ignore
PROMISE 100
PROMISE 150
ignore
PROMISE 100
PROMISE 150

Liveness



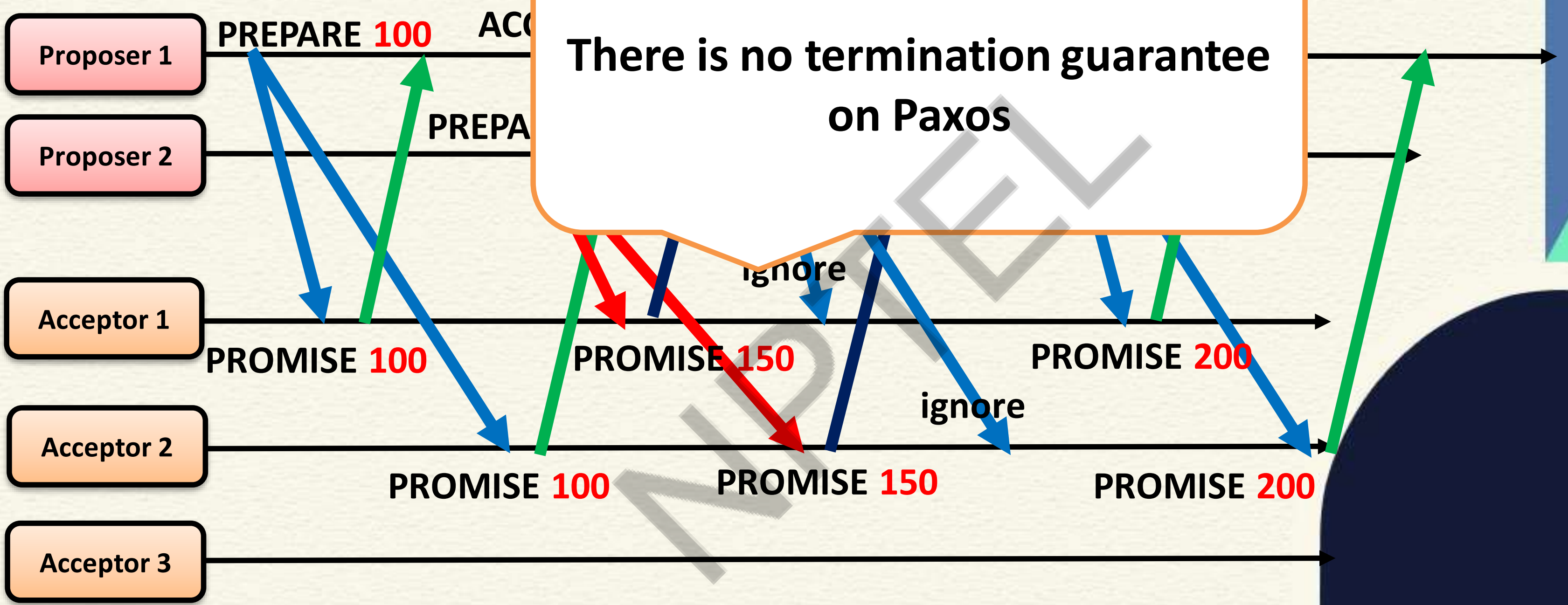
Proposer 1
Proposer 2
Acceptor 1
Acceptor 2
Acceptor 3
PREPARE 100
ACCEPT-REQUEST 100, "Dinner"
PREPARE 200
PREPARE 150
ignore
PROMISE 100
PROMISE 150
ignore
PROMISE 100
PROMISE 150

Liveness



Proposer 1
Proposer 2
Acceptor 1
Acceptor 2
Acceptor 3
PREPARE 100
ACCEPT-REQUEST 100, "Dinner"
PREPARE 200
PREPARE 150
ignore
PROMISE 100
PROMISE 150
PROMISE 200
ignore
PROMISE 100
PROMISE 150
PROMISE 200

Liveness



There is no termination guarantee on Paxos

Majority of Accepts

- Majority of acceptors accept a request with an ID and a value
  - Consensus has been reached
  - The consensus is on the **value**

- Accept request with a lower ID
  - Will not be accepted by the majority (Would require majority of promises with the lower ID, but we got for a higher one, hence the accept request)

Majority of Accepts

- Majority of acceptors accept a request with an ID and a value
  - Consensus has been reached
  - The consensus is on the **value**

- Accept request with a lower ID
  - Will not be accepted by the majority (Would require majority of promises with the lower ID, but we got for a higher one, hence the accept request)

Majority of Accepts

- Accept request with a higher ID but a **<u>different value</u>**
  - Will not be accepted by the majority
  - At least one acceptor will piggyback the previously accepted value (Remember, two majority implies that there is a common node)

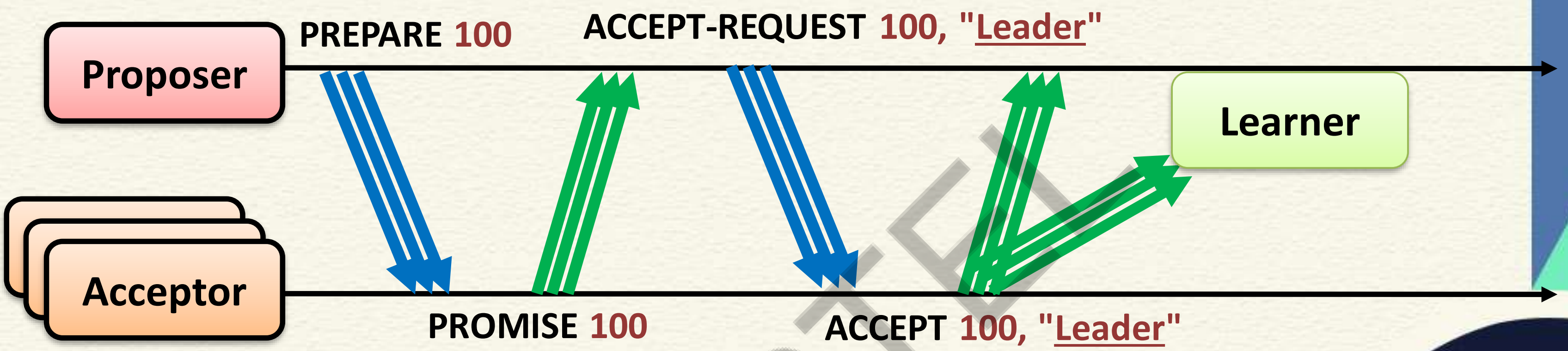
Majority of Accepts

- Accept request with a higher ID but a **<u>different value</u>**
  - Will not be accepted by the majority

So, the consensus is on the value

We need the ID to maintain the <u>current state of promise and accept</u>, so that multiple values does not propagate

Paxos for Leader Election



Multi-Paxos

- Applications often needs a continuous stream of agreed values
  - Commit the transactions in a replicated database – each transaction needs a consensus to be agreed upon by the replicas

- Run multiple instances of Paxos with different round numbers
  - Each value is associated with a round number

Multi-Paxos

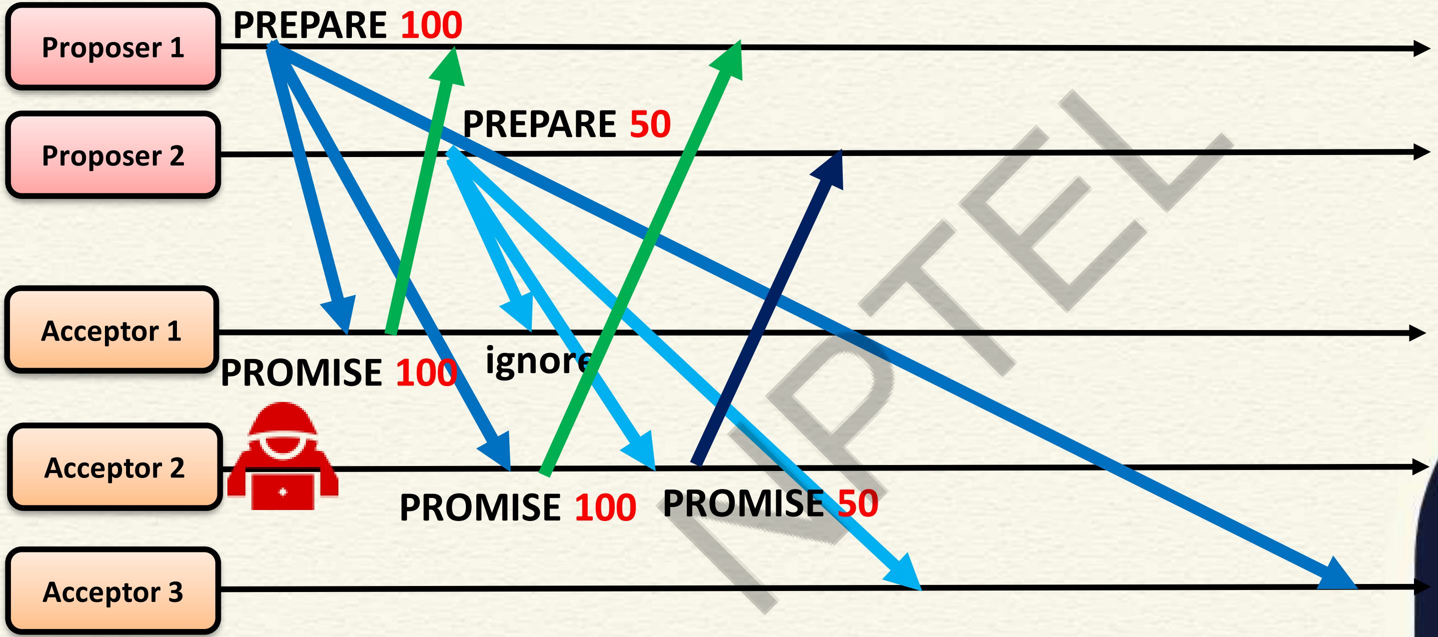
- If a value is already accepted for Round *n*, ignore the accept requests for a different value under Round *n*
  - Forward an ACCEPT IDp, <u>(ROUNDn, VALUE)</u> only when no value has been agreed upon for the Round *n*

Conclusion

- CFT consensus in asynchronous system – Paxos
    - Safety is ensured, but liveness is compromised

- Does Paxos work when a node sends a wrong message?

Conclusion – Attack on Paxos




Thank
you

NPTEL ONLINE CERTIFICATION COURSES

Blockchain and its applications

Prof. Sandip Chakraborty

Department of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Lecture 30: Byzantine Faults

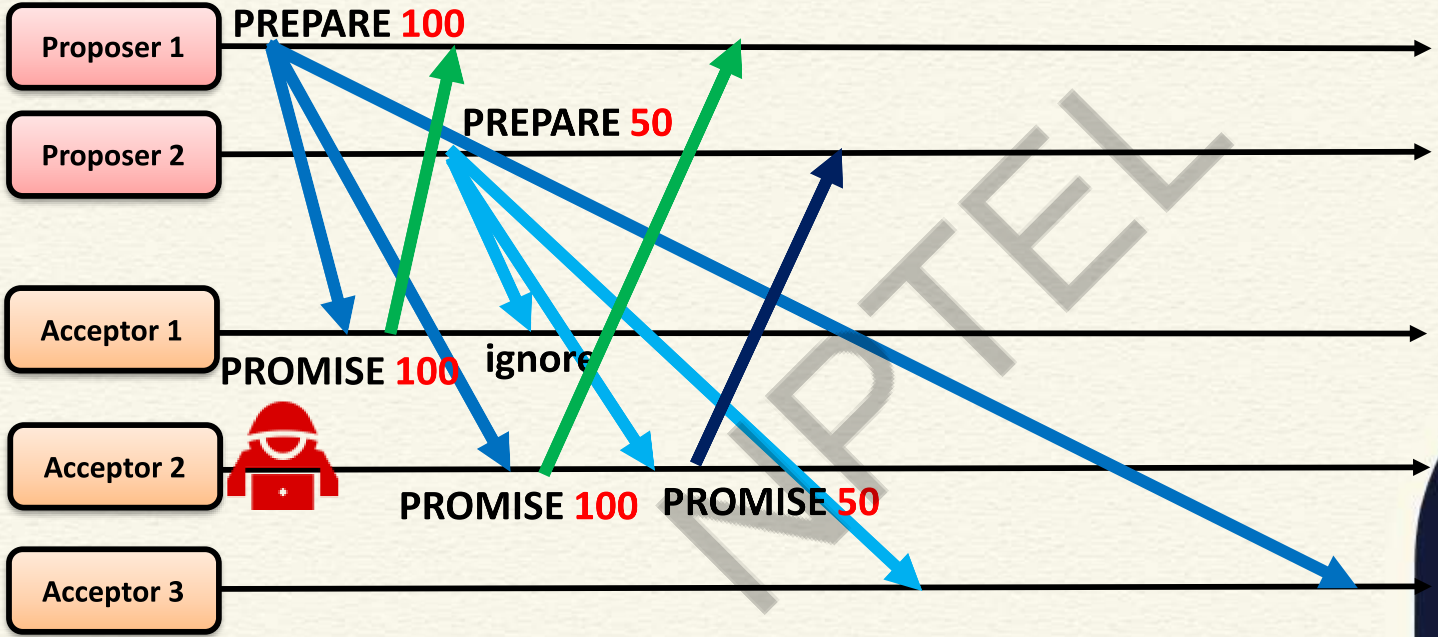
CONCEPTS COVERED

- Byzantine Faults
- Byzantine Agreement Protocols

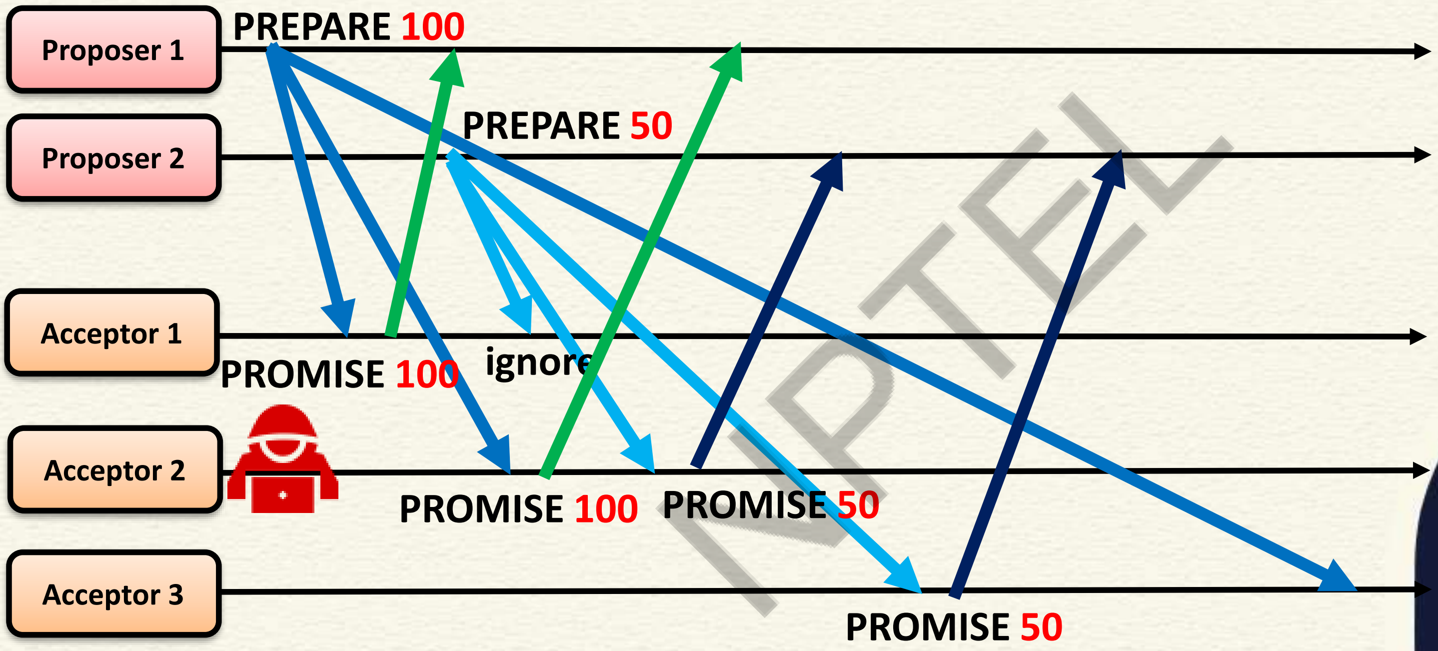
KEYWORDS

- BFT Consensus
- BFT Agreement

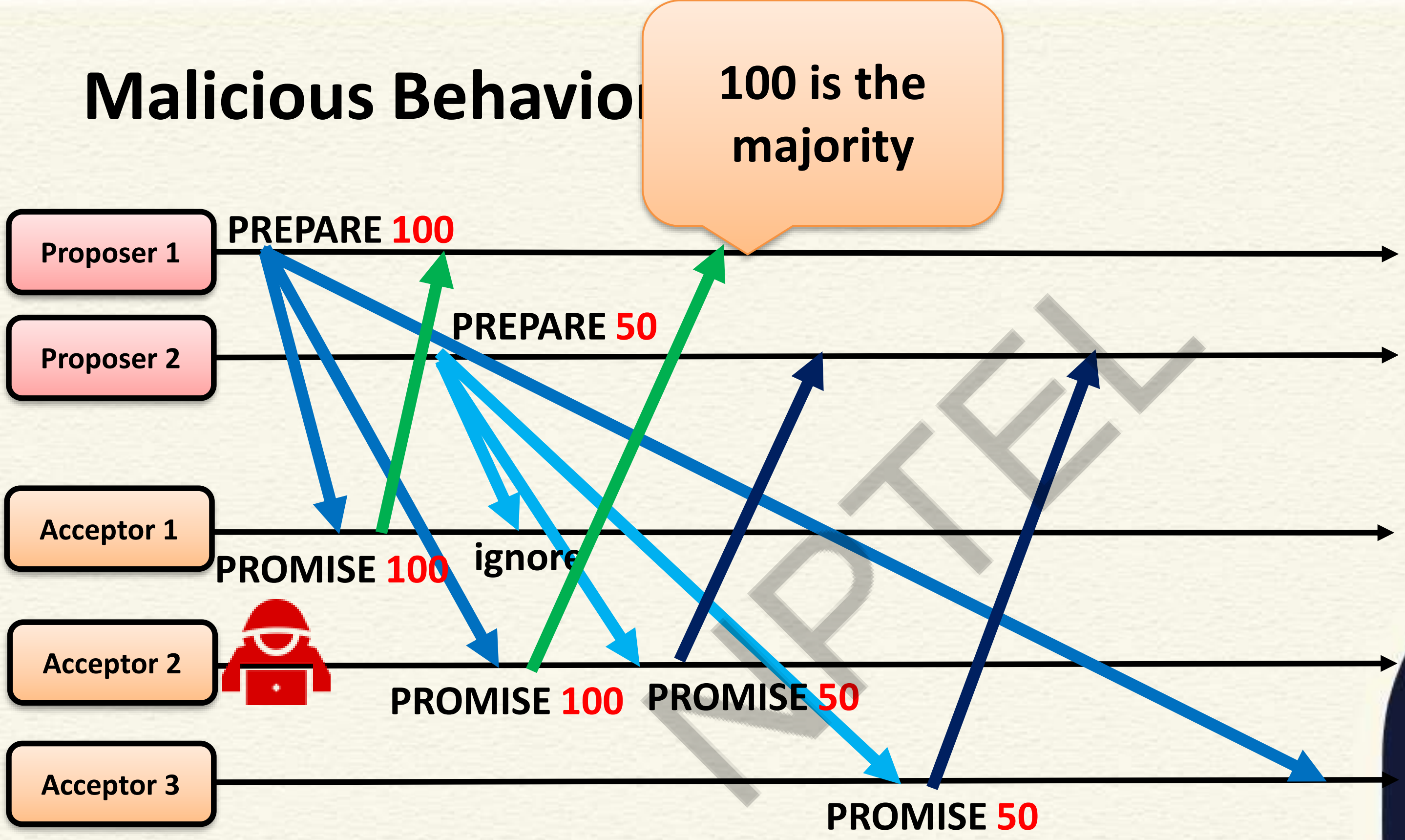
Malicious Behavior on Paxos



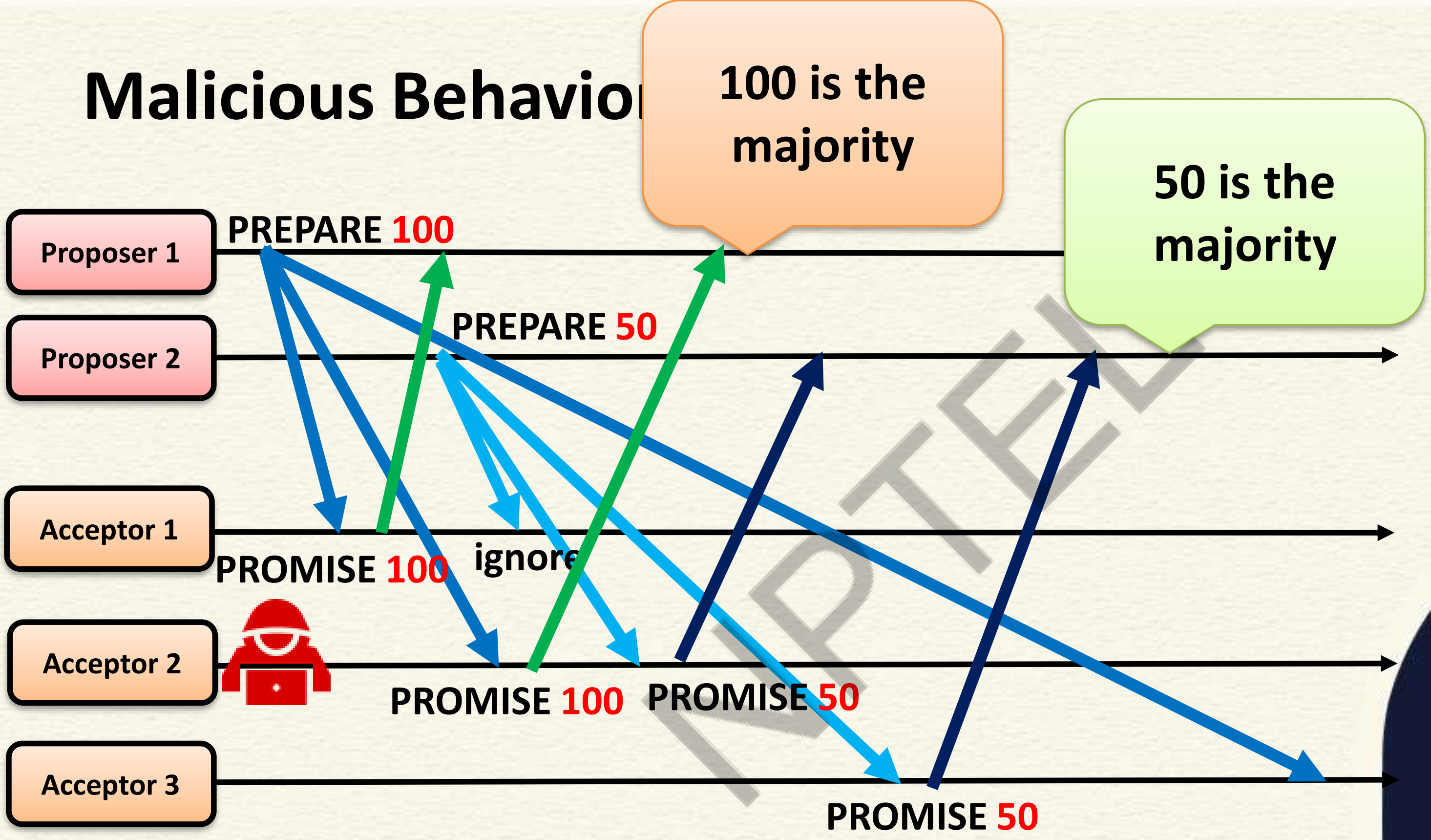
Malicious Behavior on Paxos



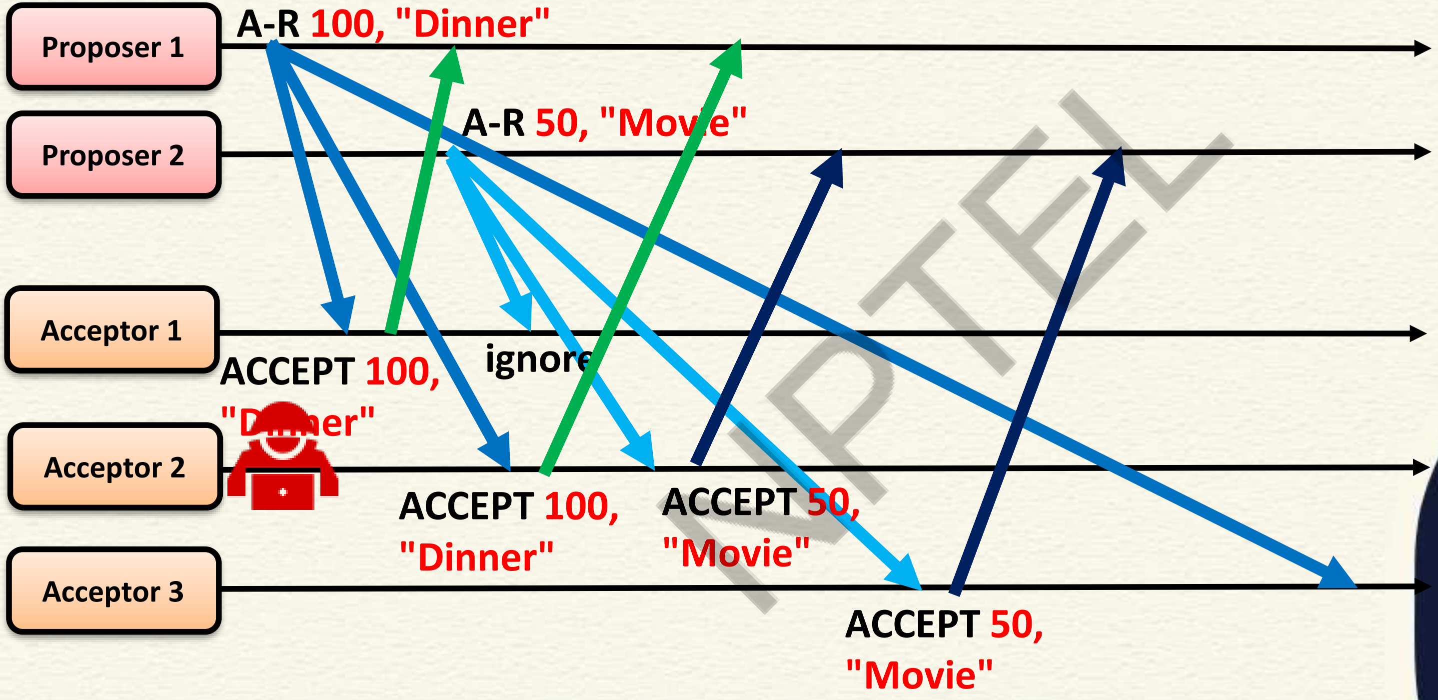
Malicious Behavior



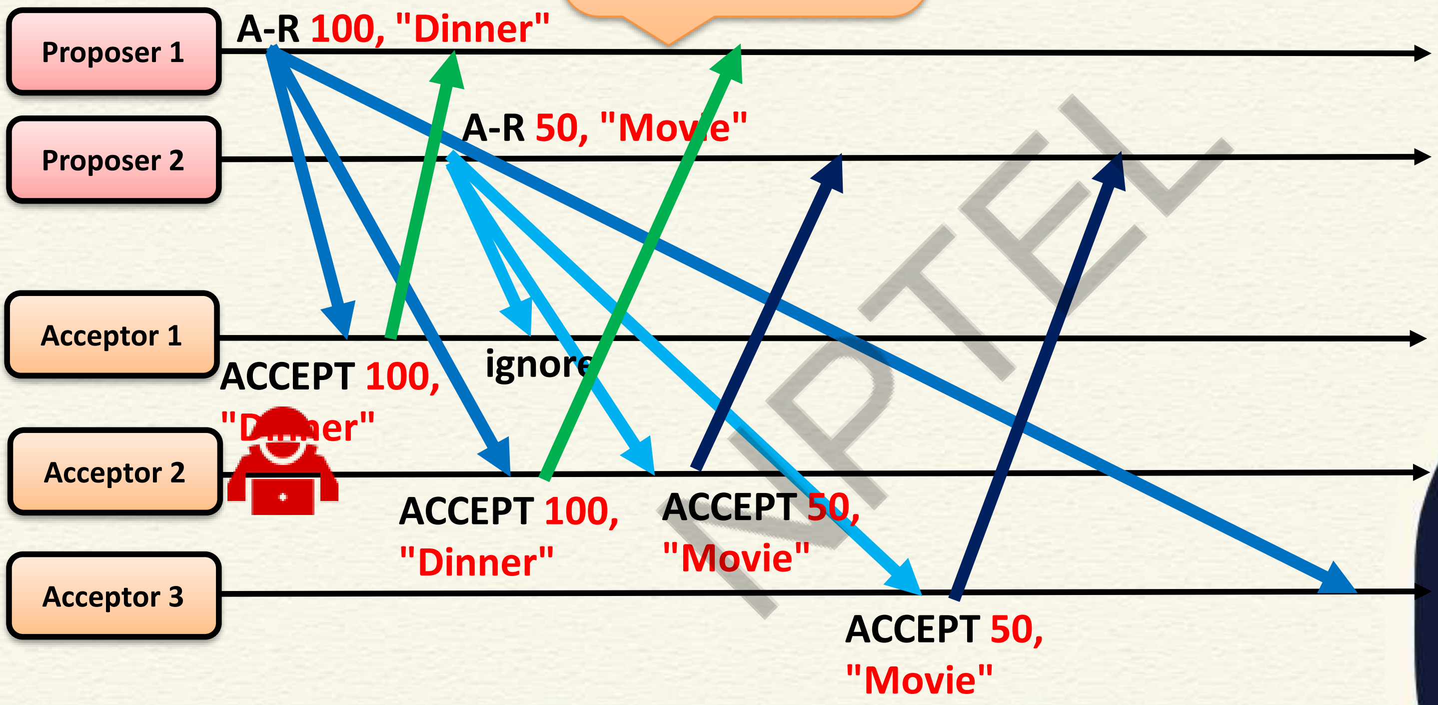
Malicious Behavior



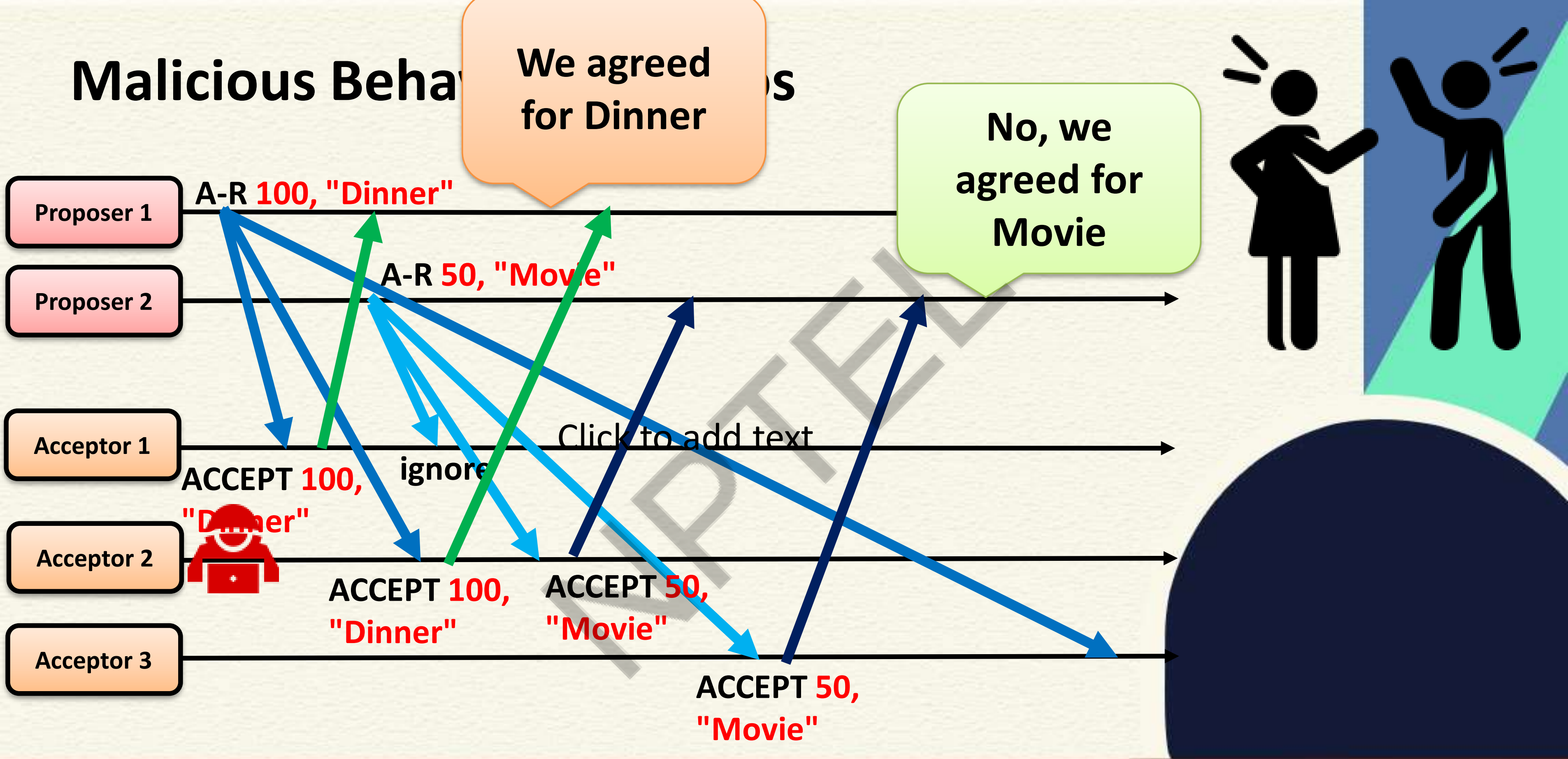
Malicious Behavior on Paxos



Malicious Behav s

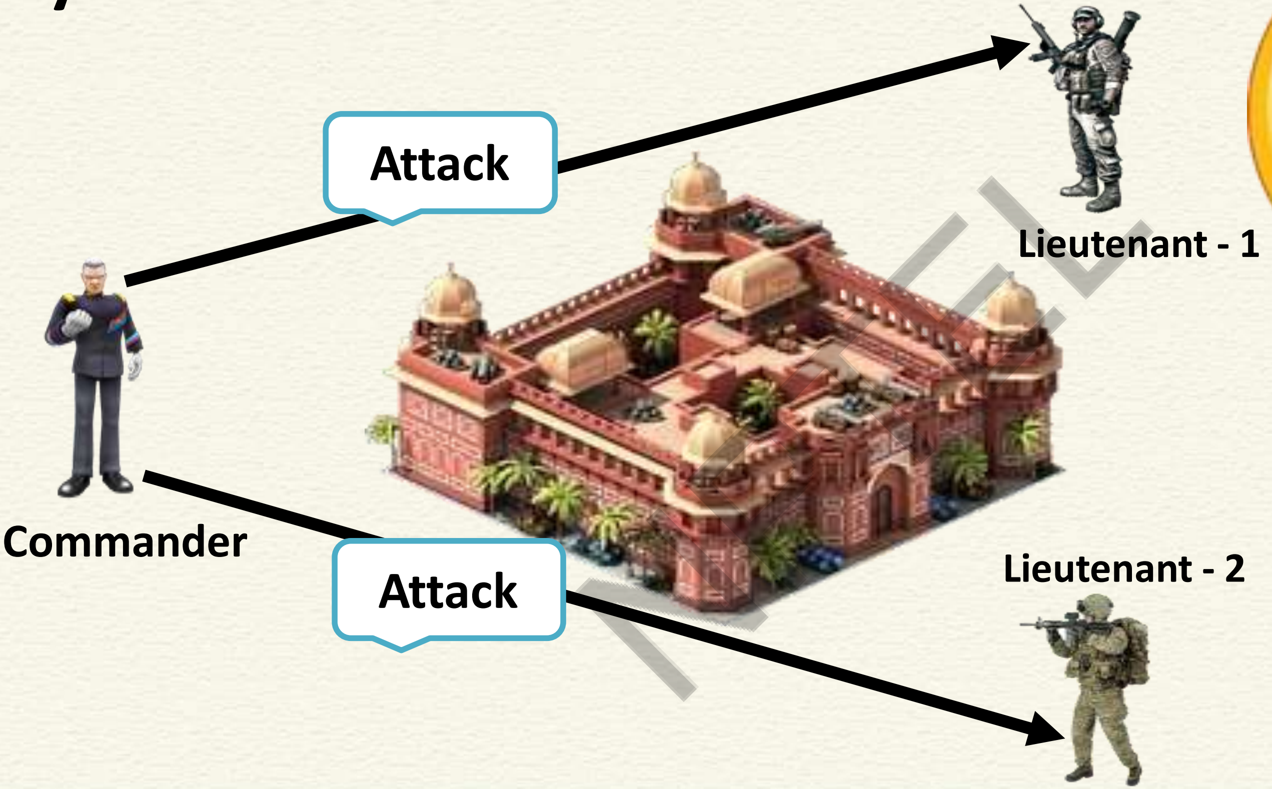


Malicious Behav s

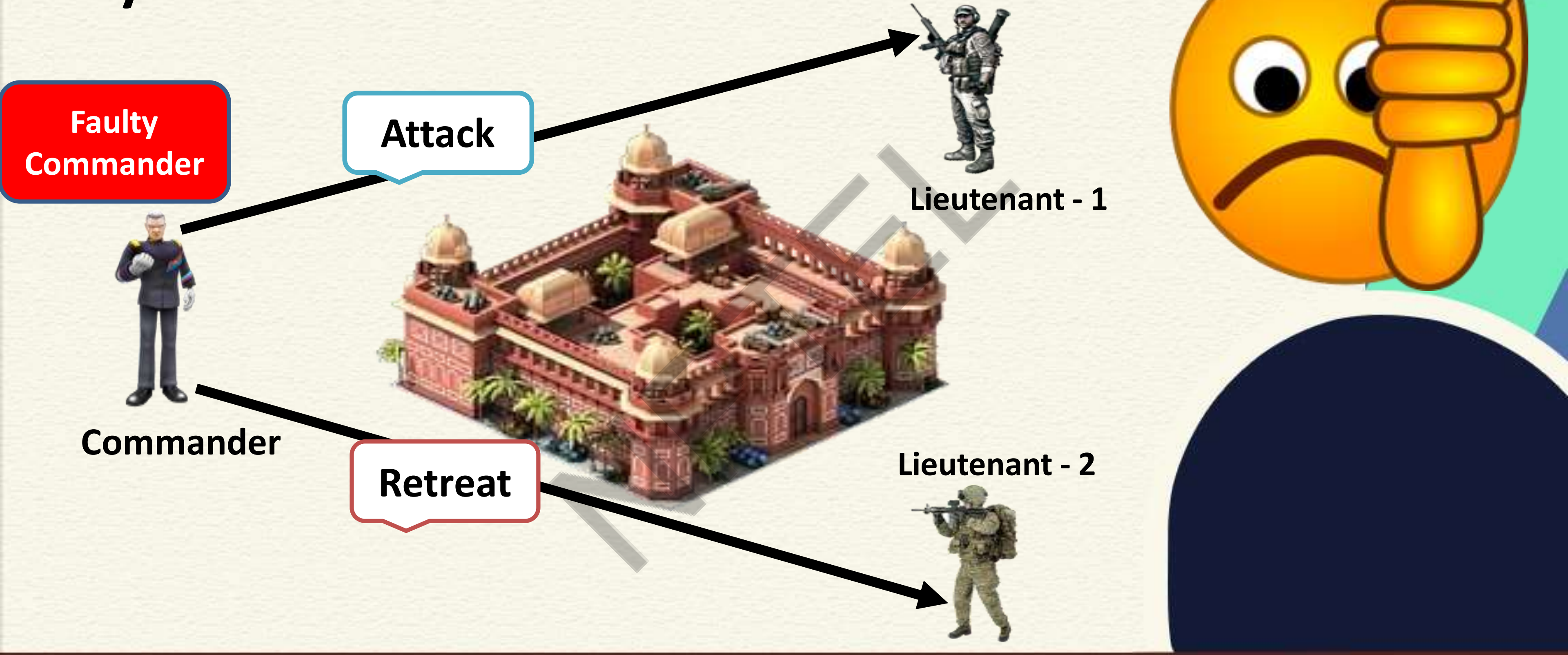


Byzantine Generals Problem

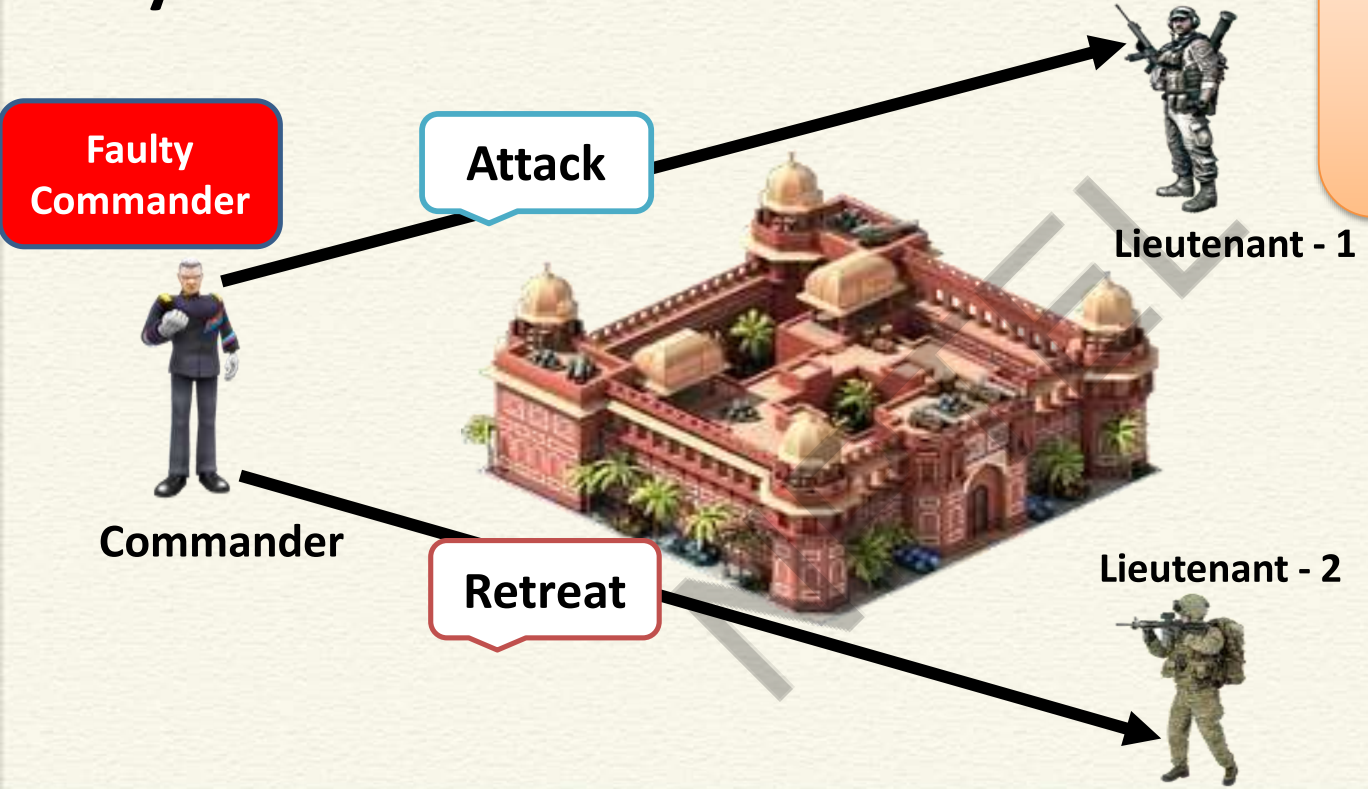
Byzantine Generals Problem



Byzantine Generals Problem

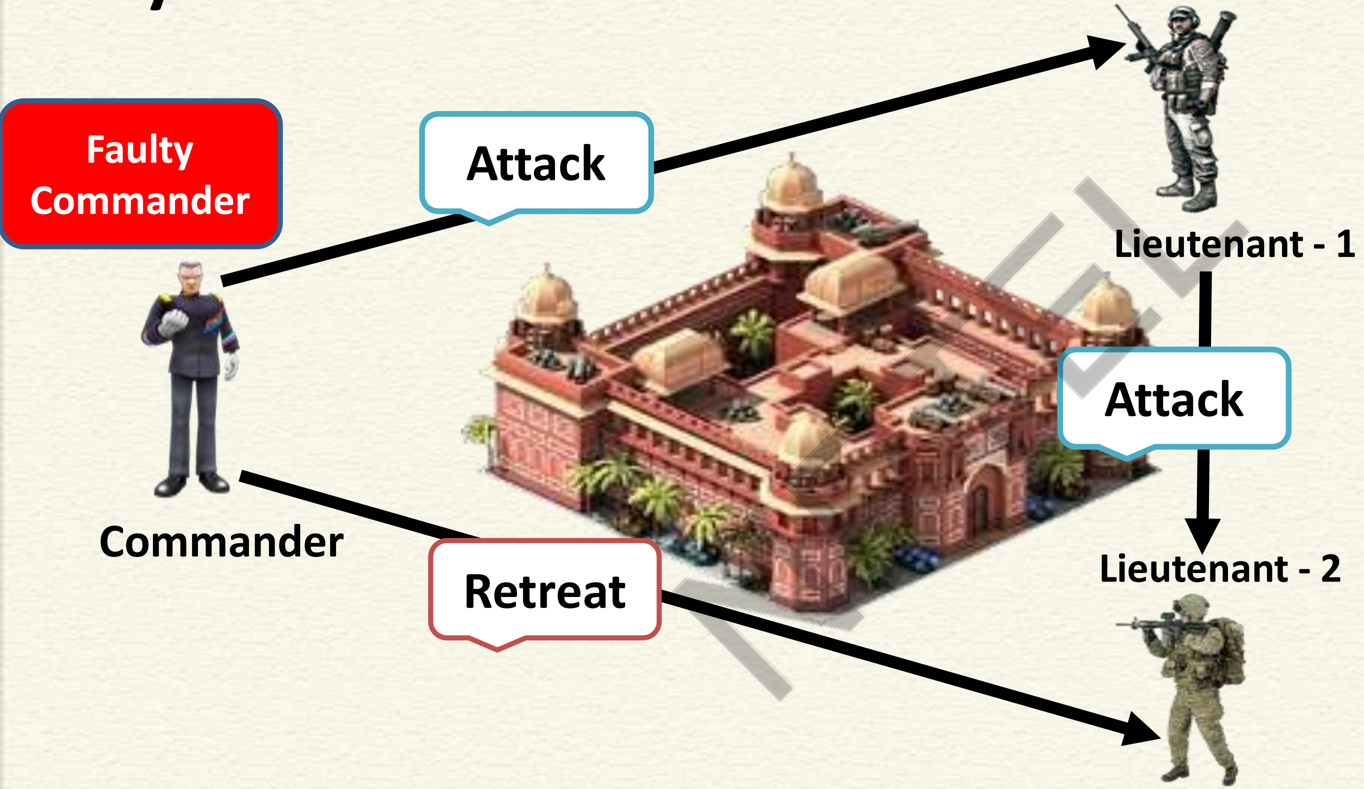


Byzantine Generals Problem

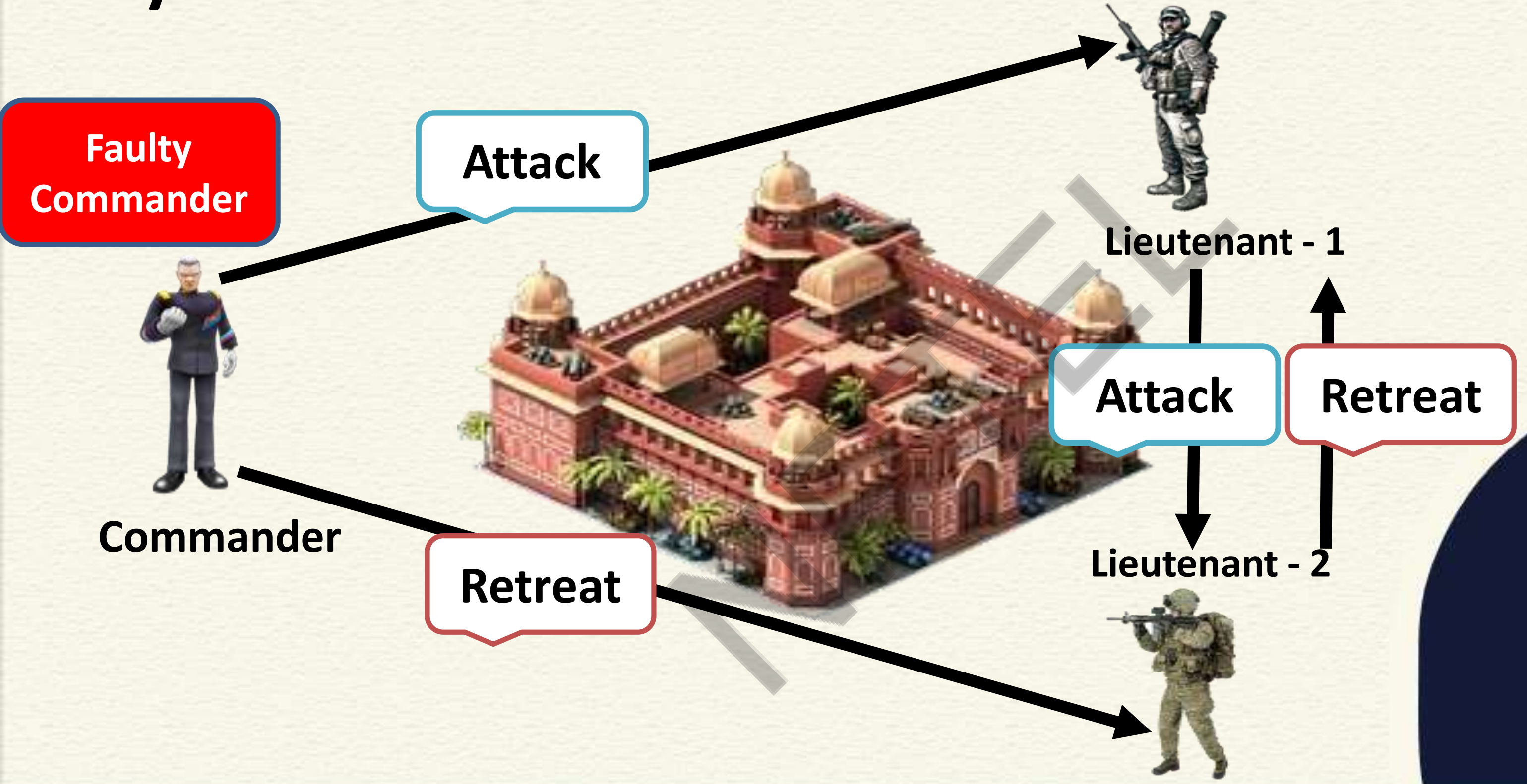


We need a consensus mechanism to decide who is faulty

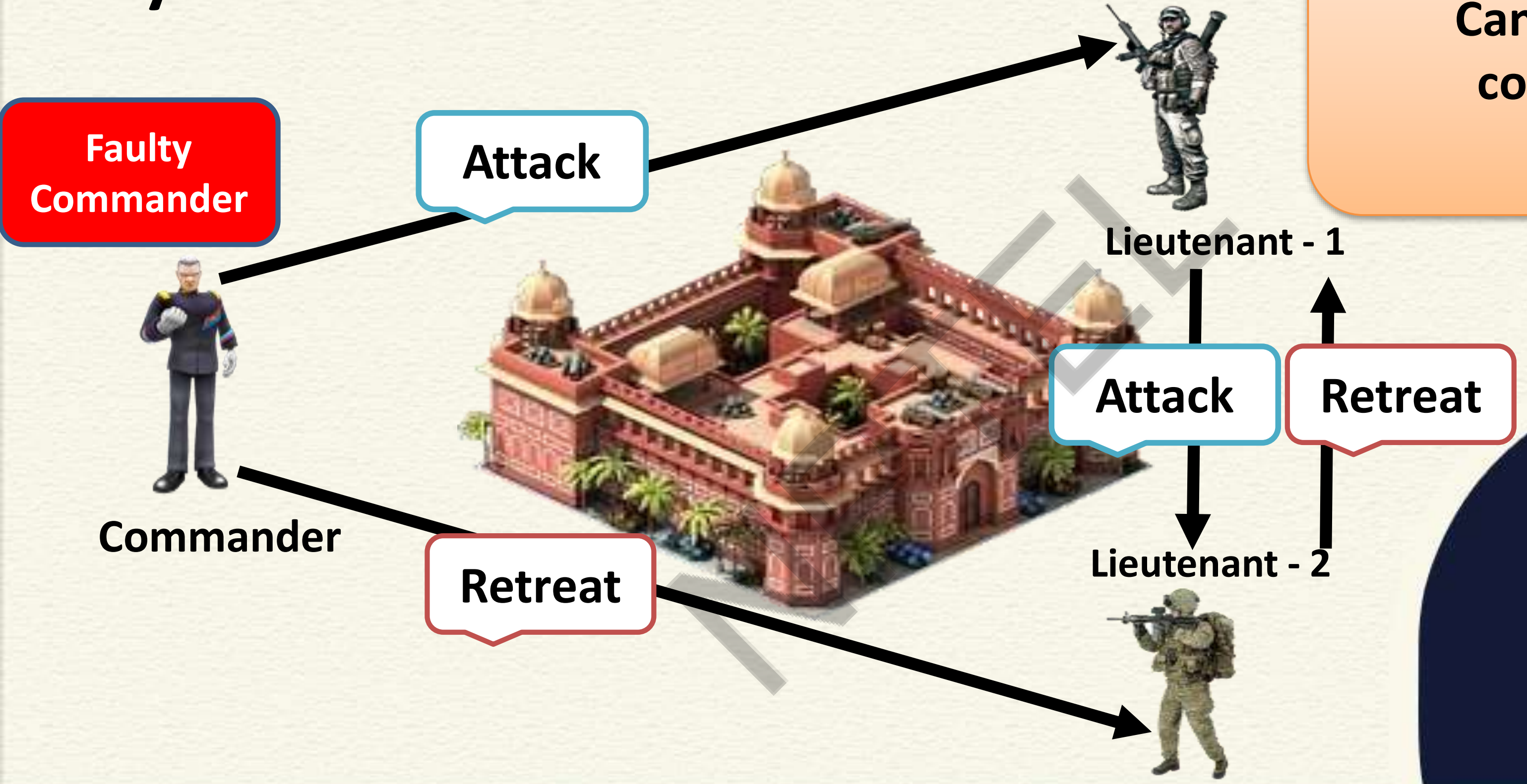
Byzantine Generals Problem



Byzantine Generals Problem

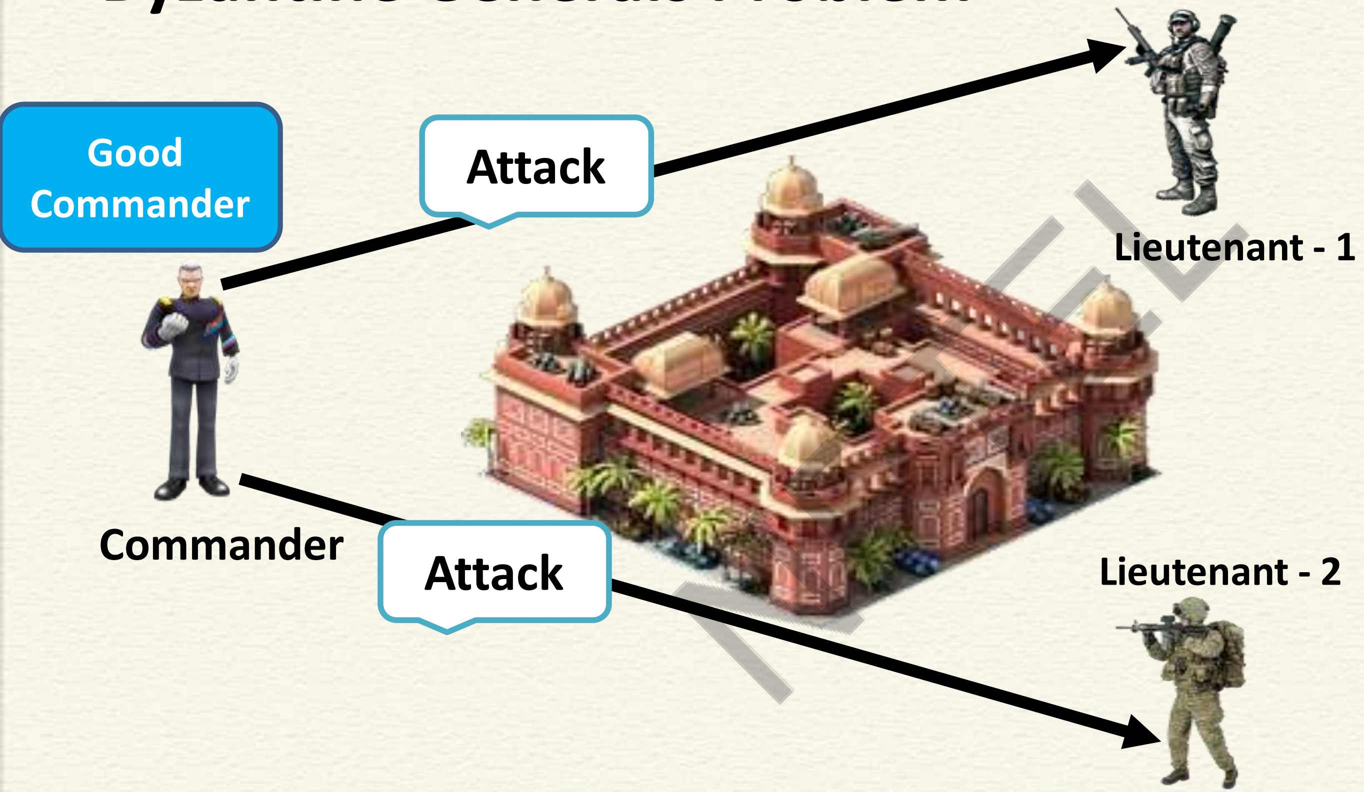


Byzantine Generals Problem

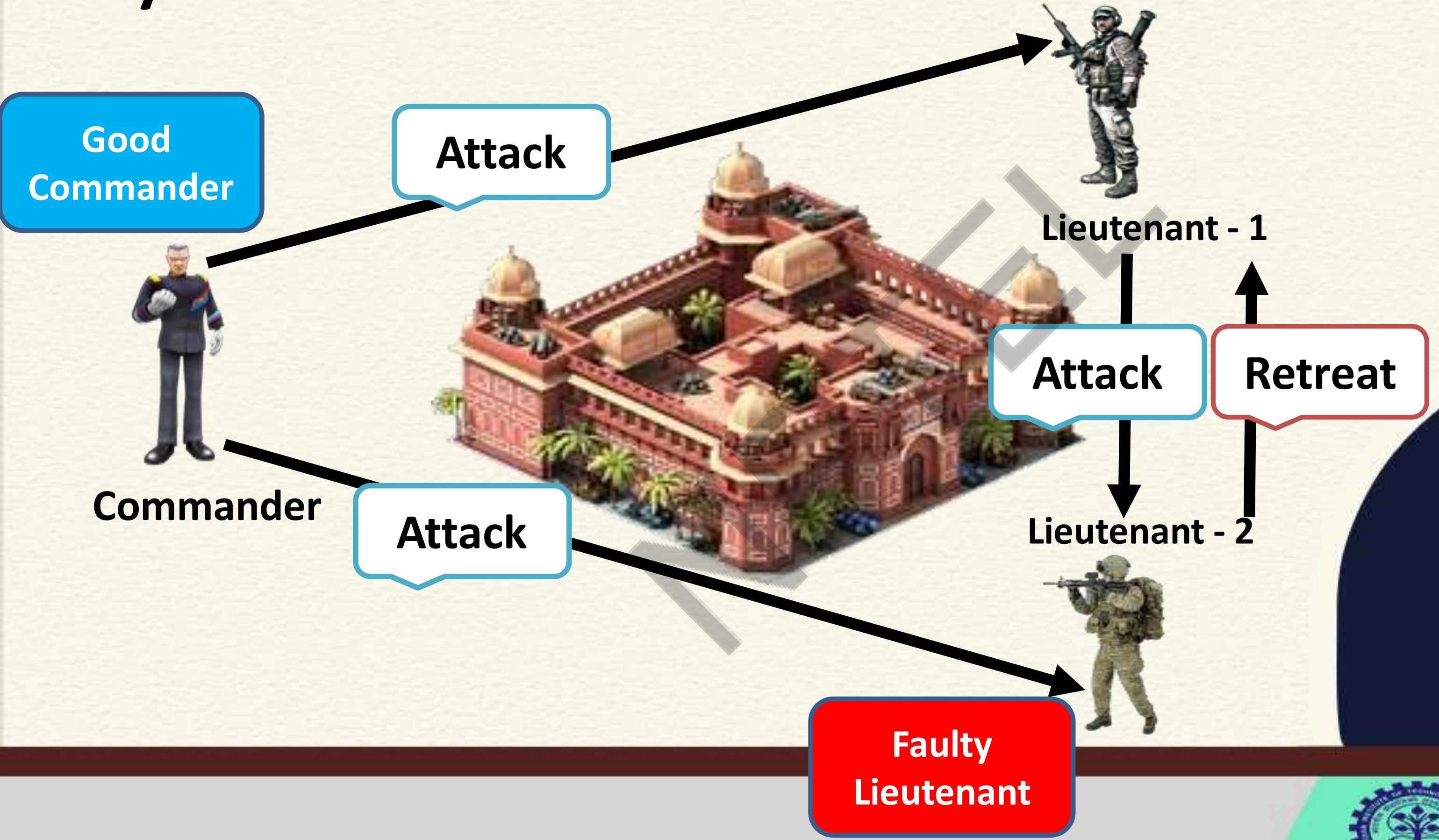


Can we reach consensus?

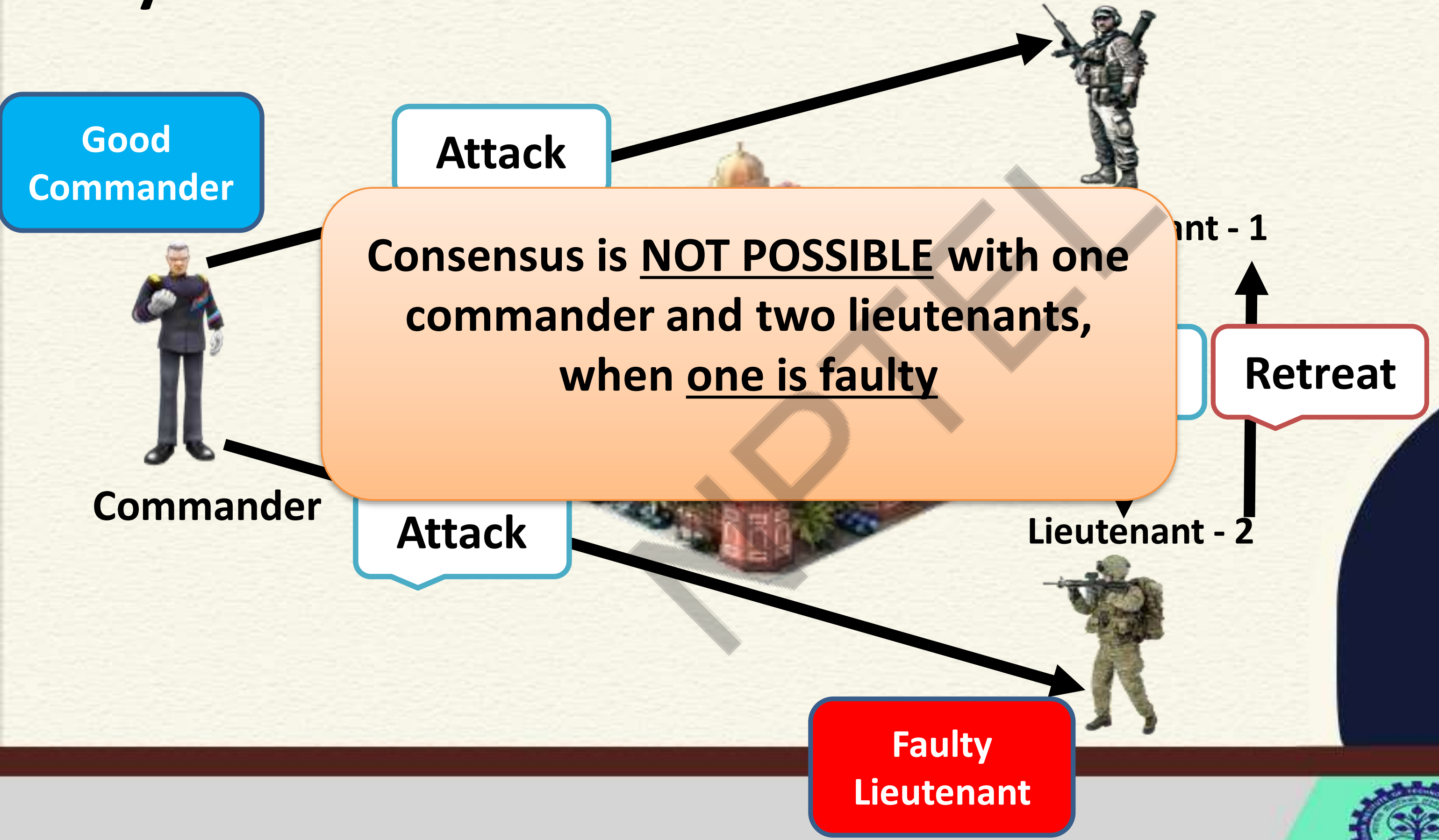
Byzantine Generals Problem



Byzantine Generals Problem

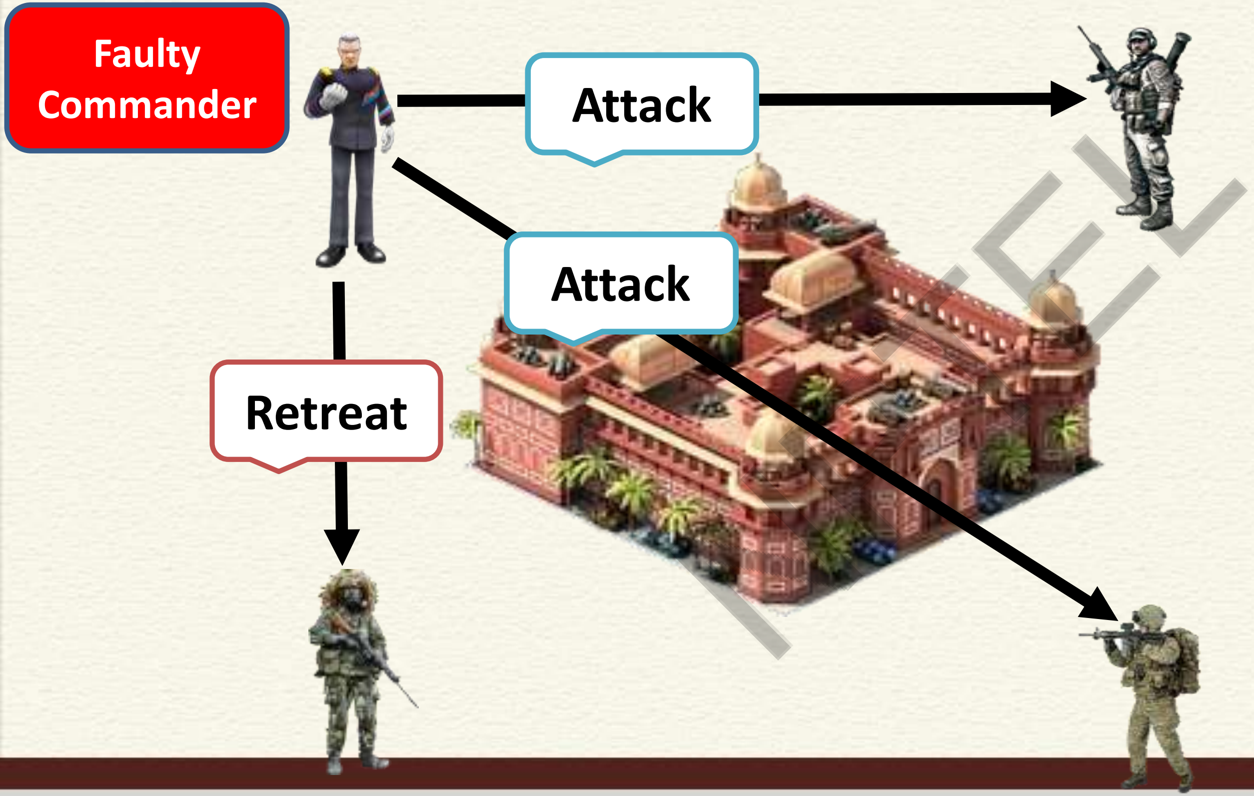


Byzantine Generals Problem

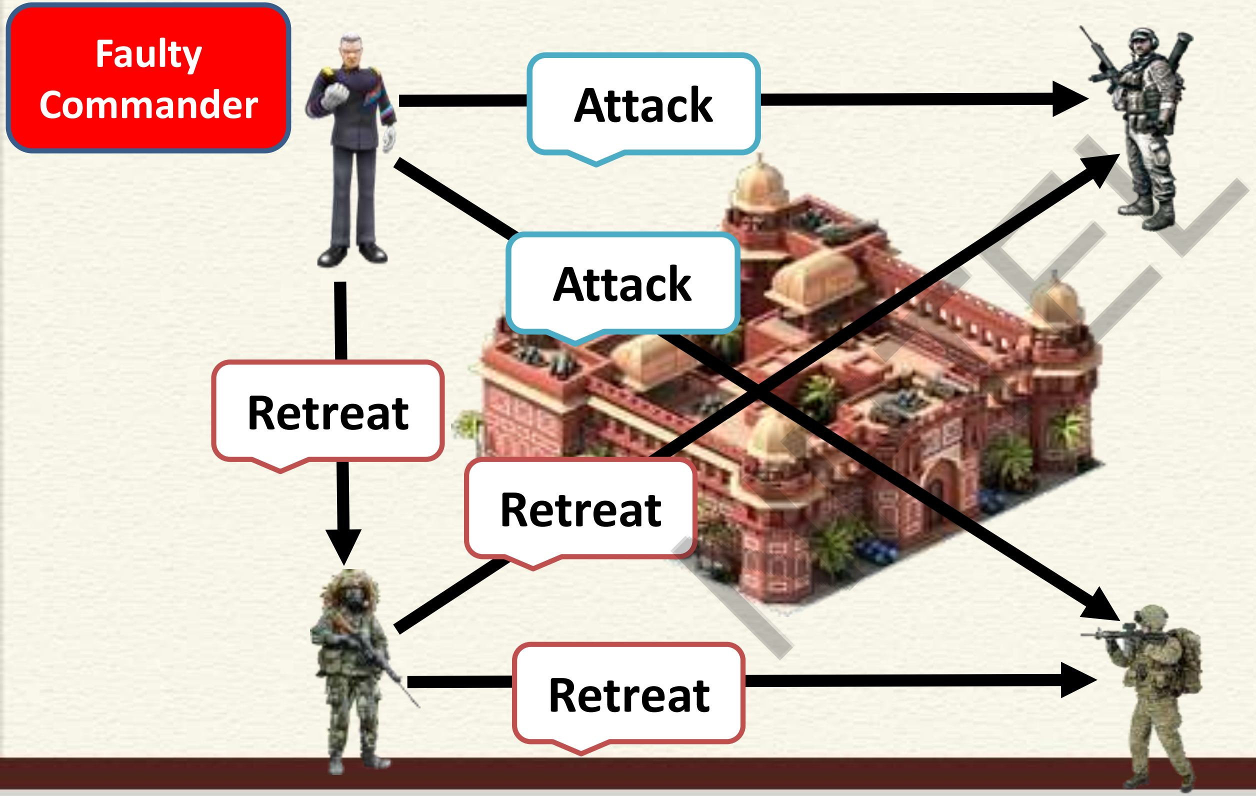


Consensus is NOT POSSIBLE with one commander and two lieutenants, when one is faulty

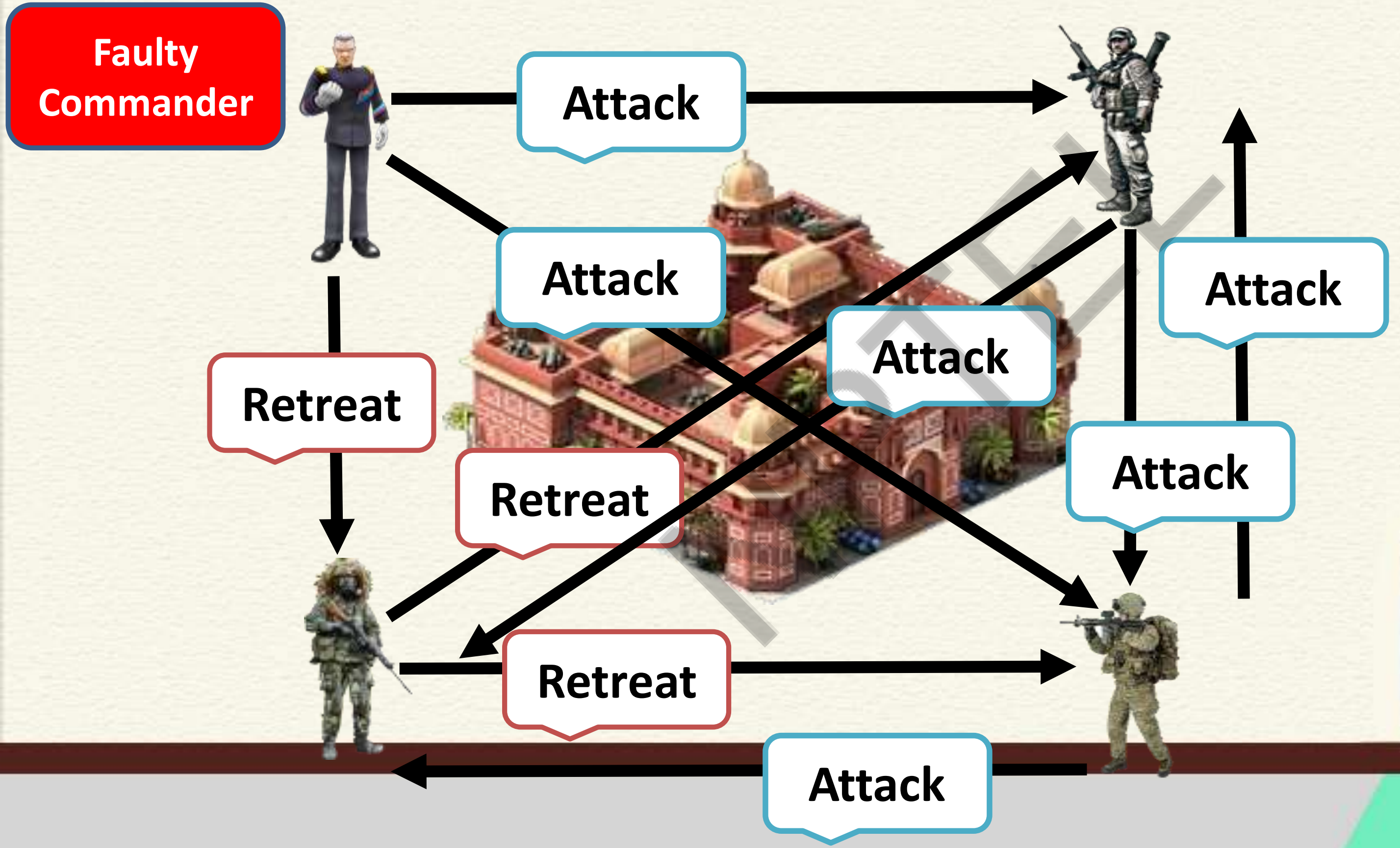
Byzantine Generals Problem – Case 2



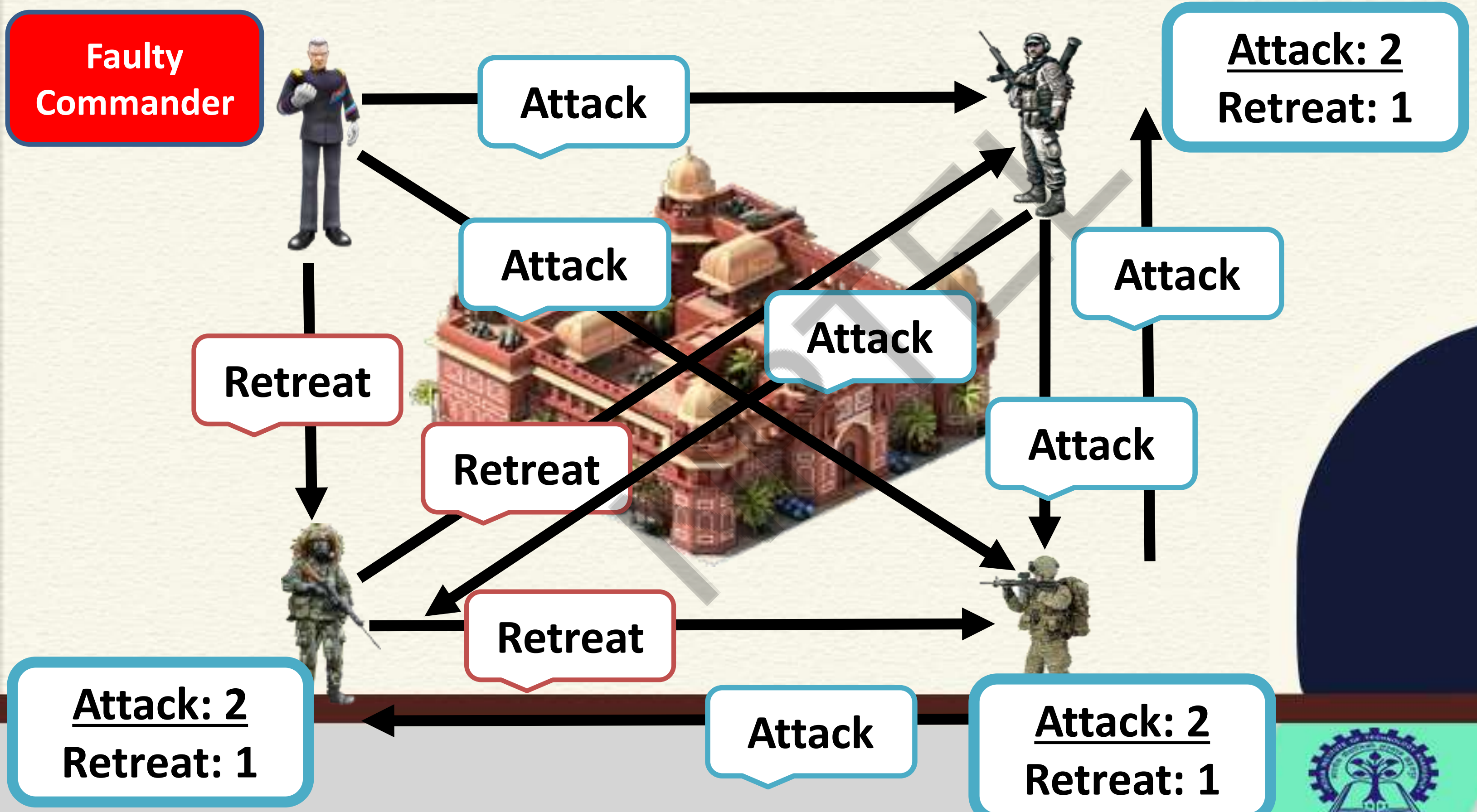
Byzantine Generals Problem – Case 2



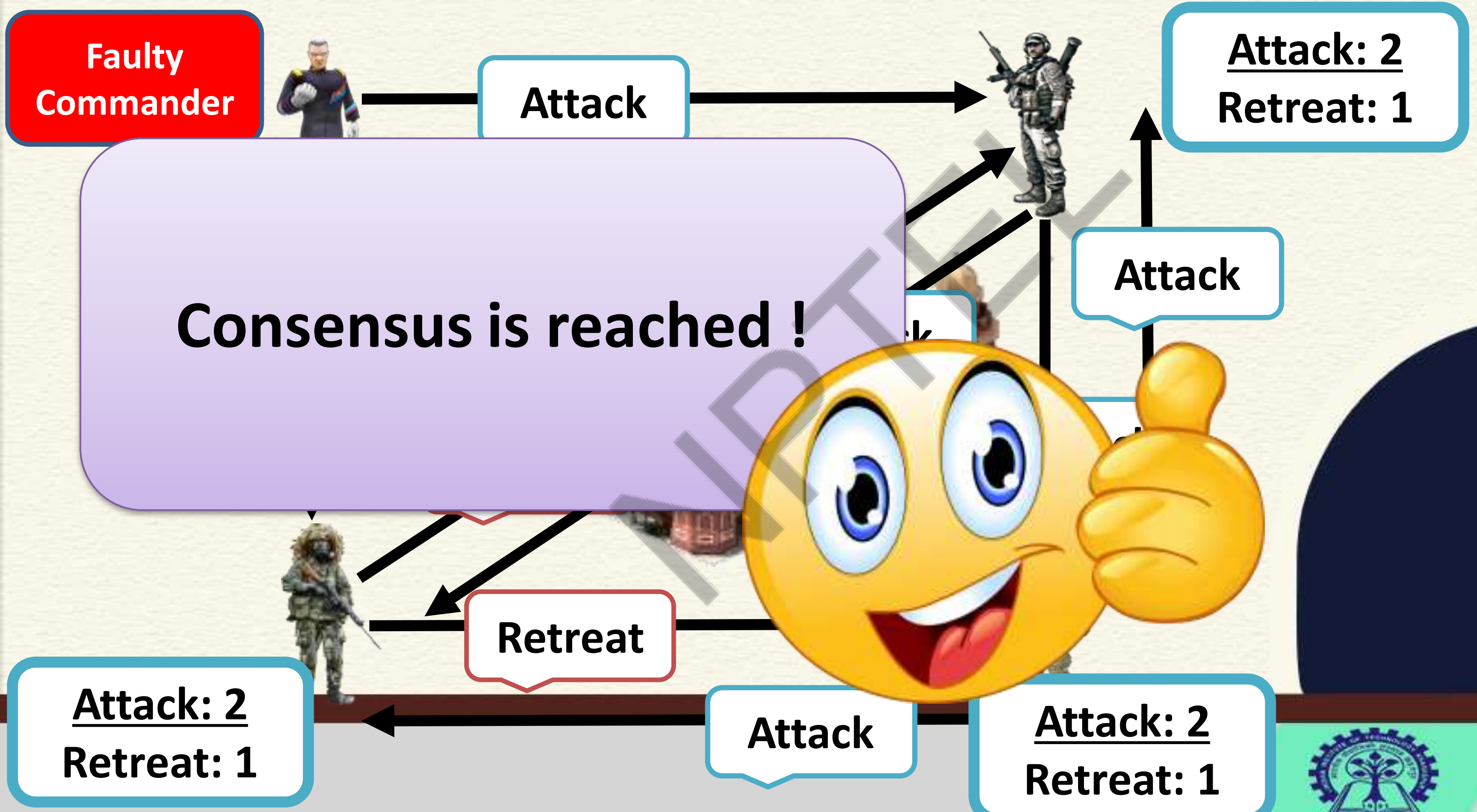
Byzantine Generals Problem – Case 2



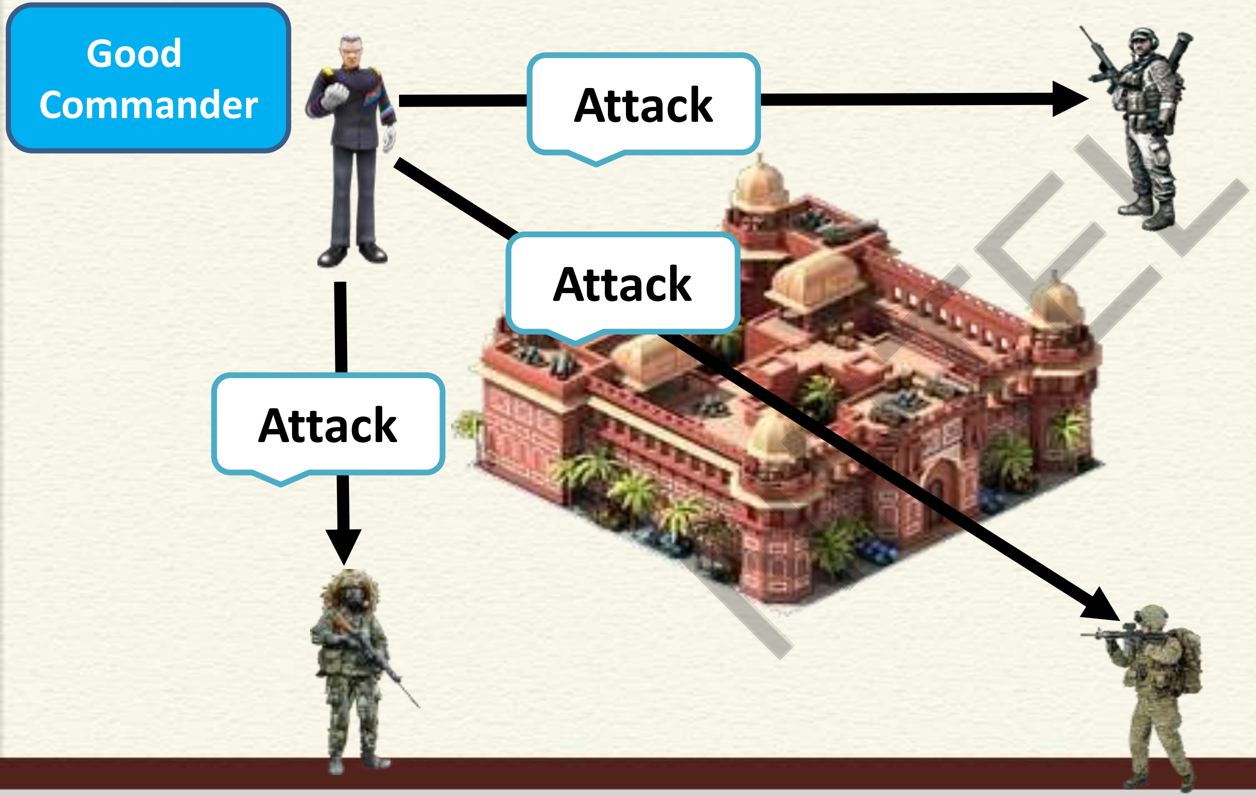
Byzantine Generals Problem – Case 2



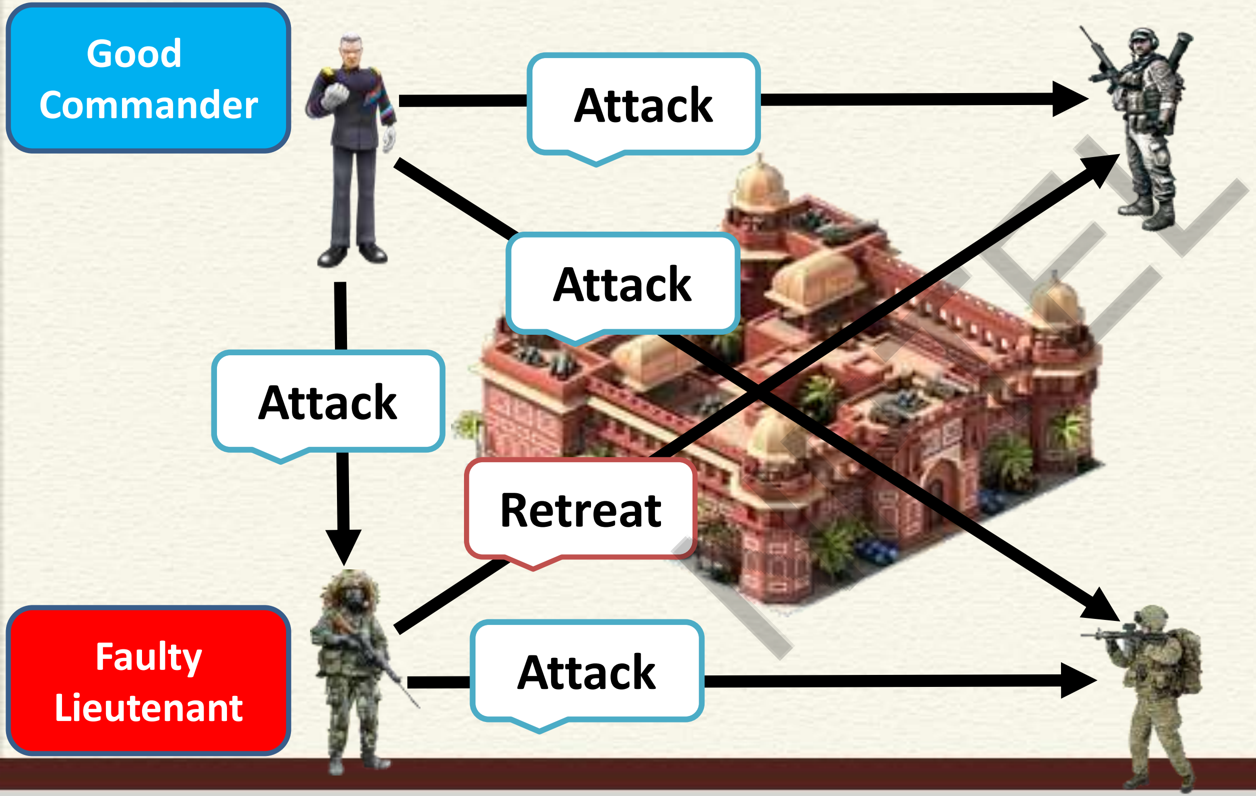
Byzantine Generals Problem – Case 2



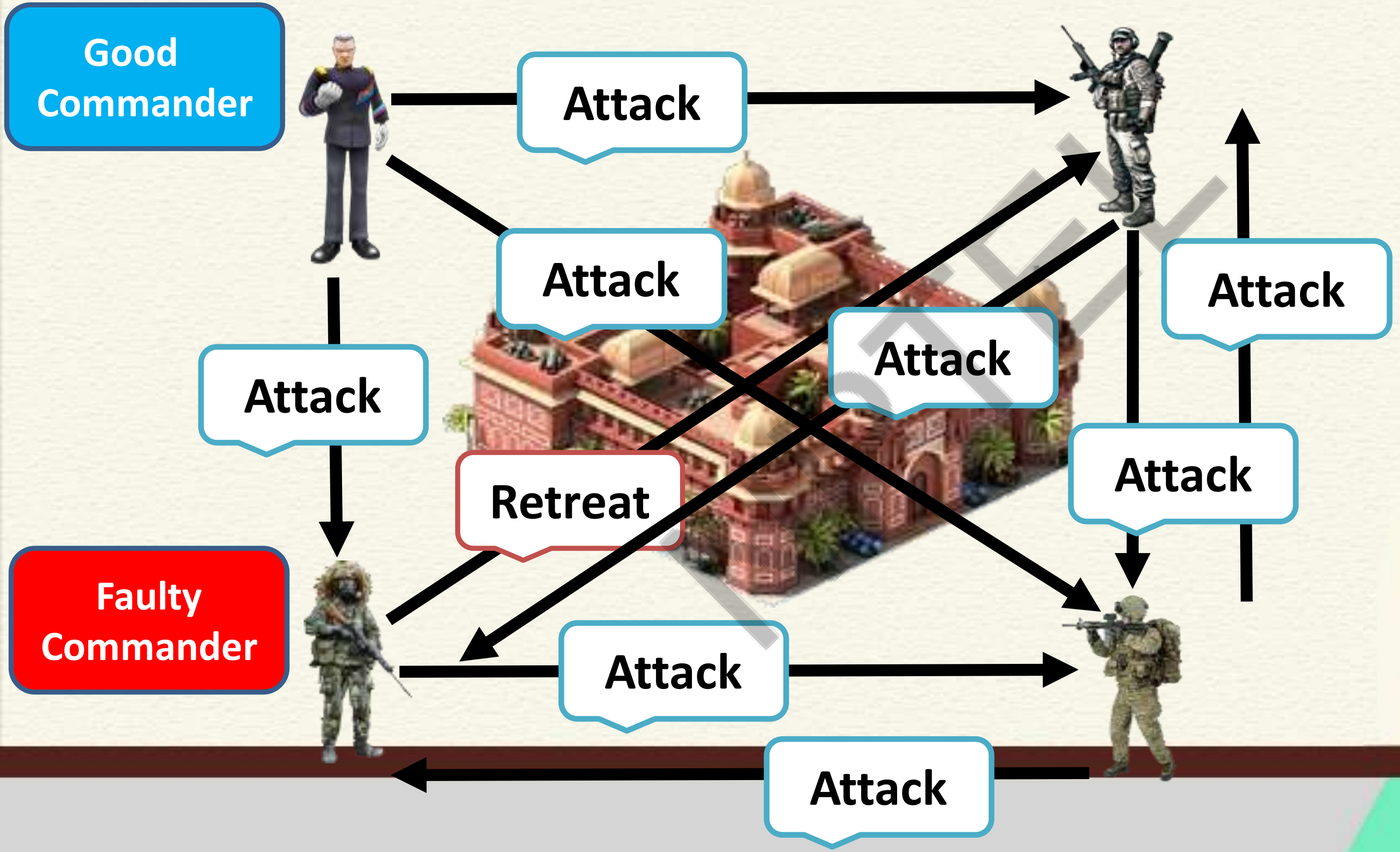
Byzantine Generals Problem – Case 2



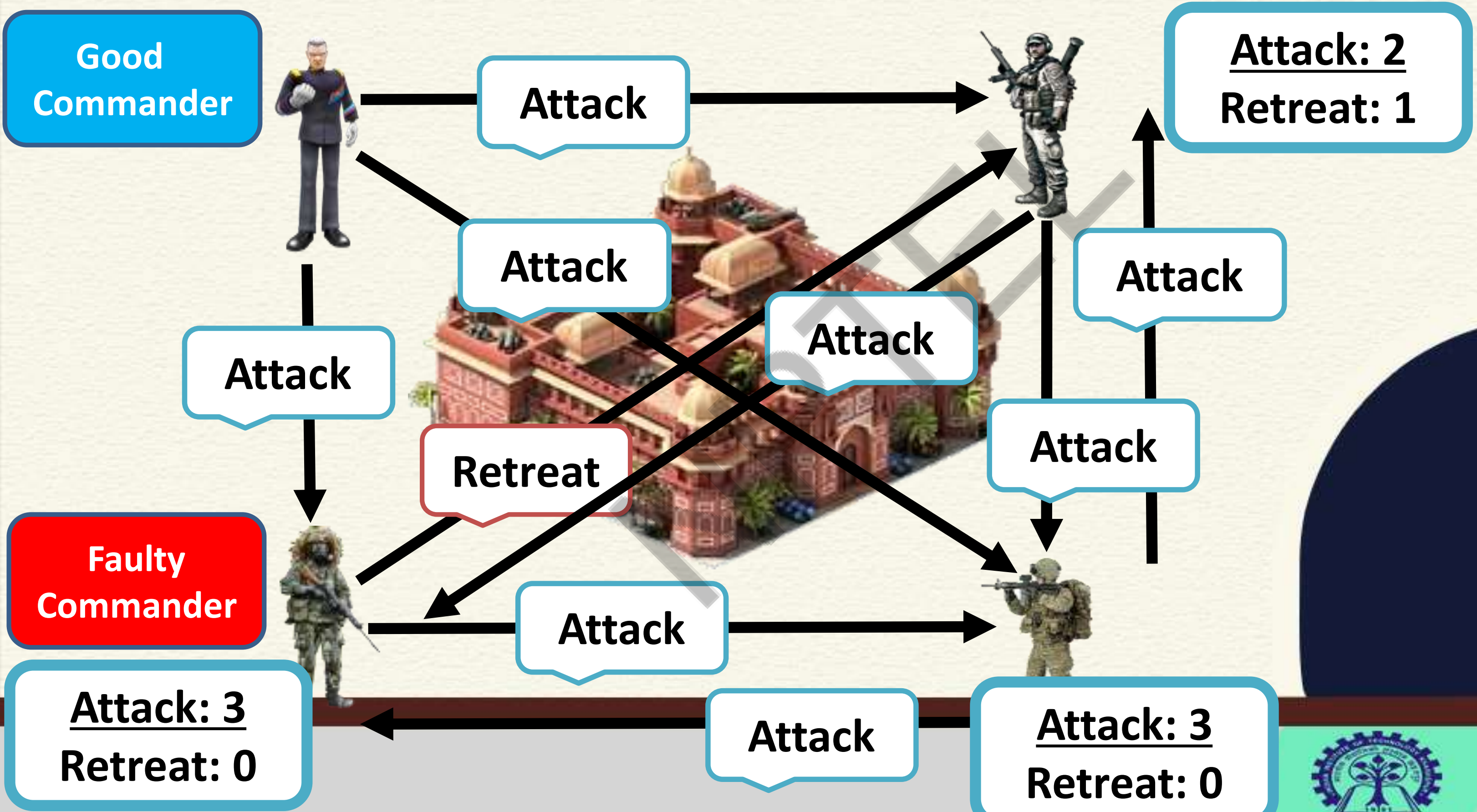
Byzantine Generals Problem – Case 2



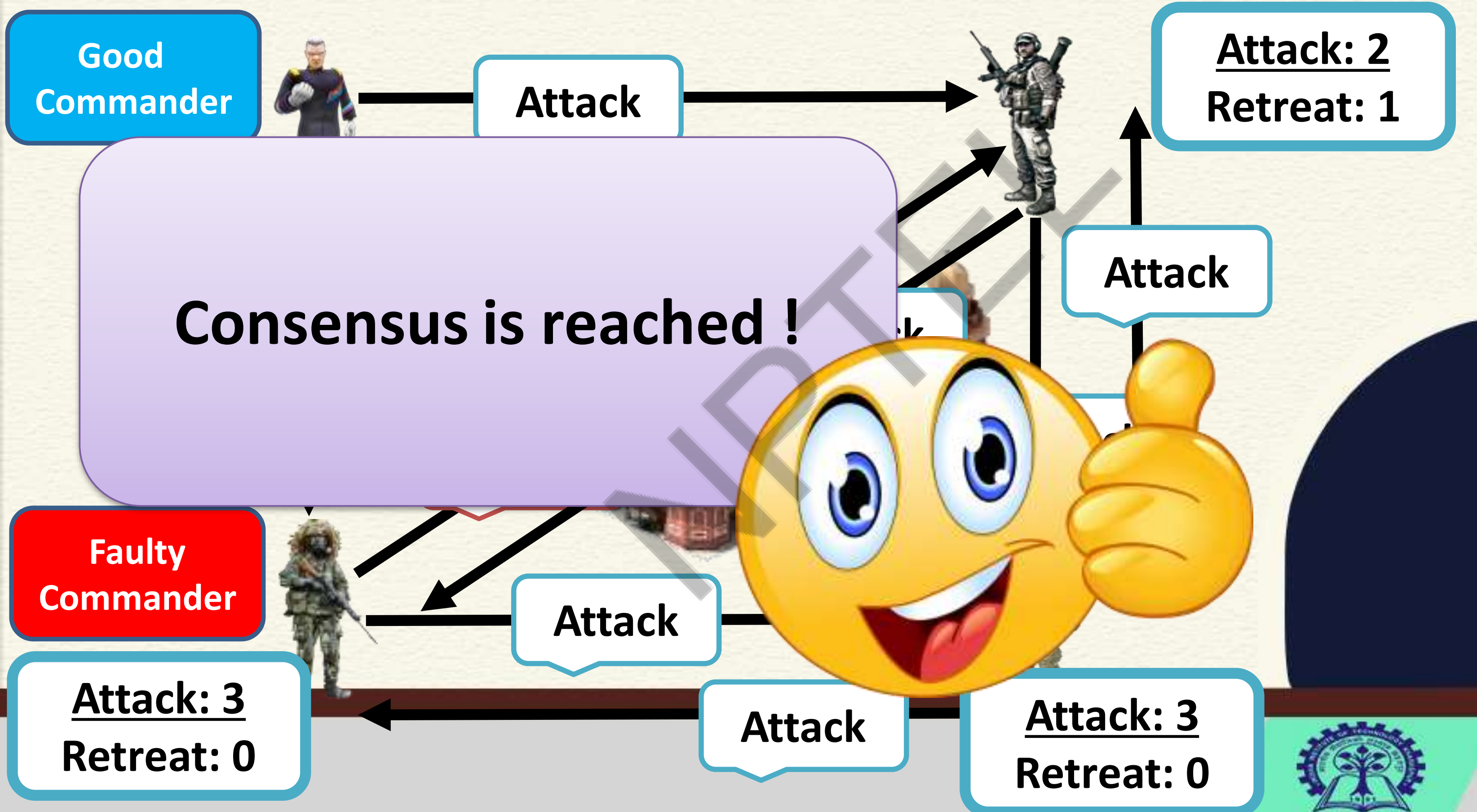
Byzantine Generals Problem – Case 2



Byzantine Generals Problem – Case 2



Byzantine Generals Problem – Case 2



Asynchronous Byzantine Agreement

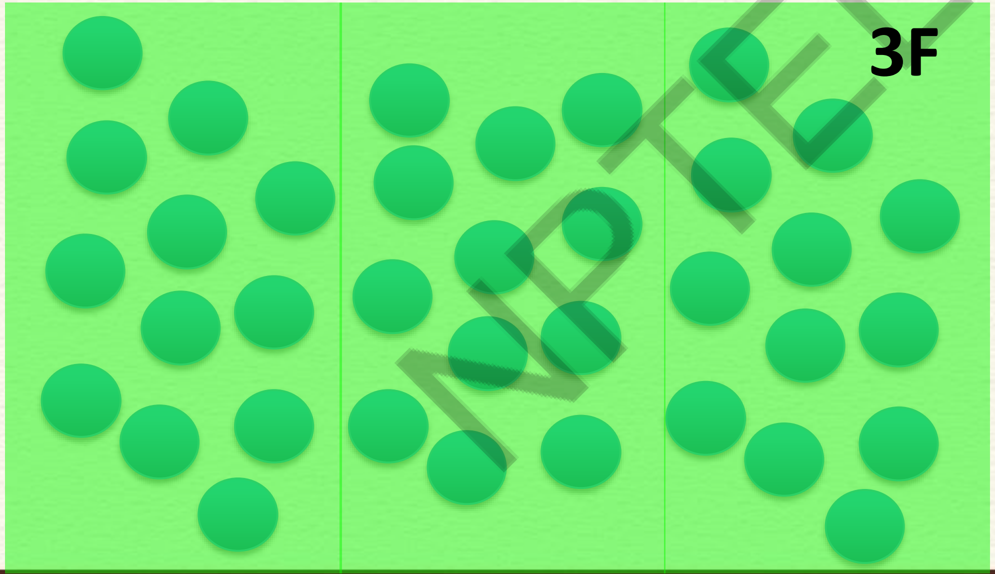
- F faulty nodes – need 3F + 1 nodes to reach consensus

Asynchronous Byzantine Agreement

- F faulty nodes – need 3F + 1 nodes to reach consensus
  - Faulty nodes create partition in the network

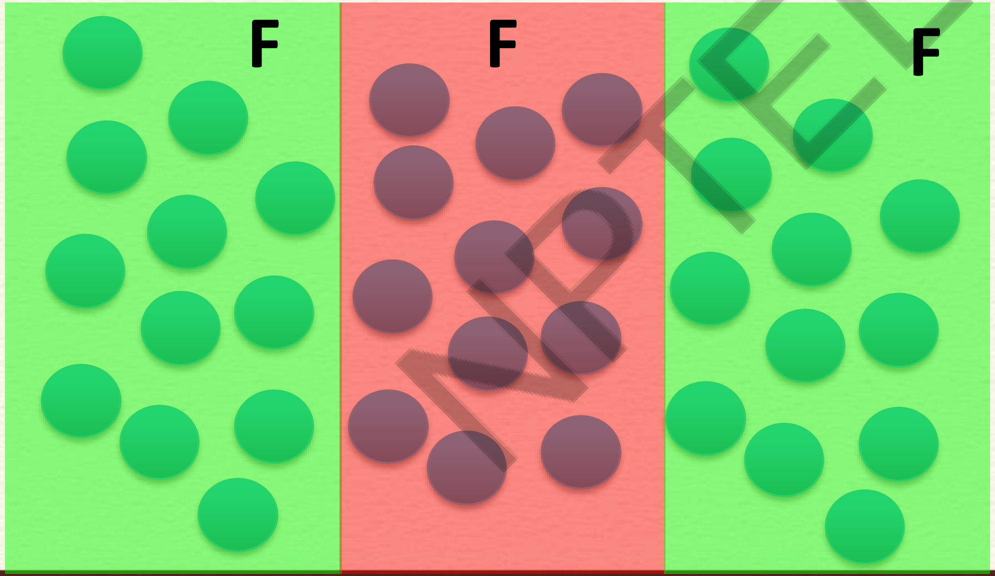
Asynchronous Byzantine Agreement

- F faulty nodes – need 3F + 1 nodes to reach consensus
  - Faulty nodes create partition in the network



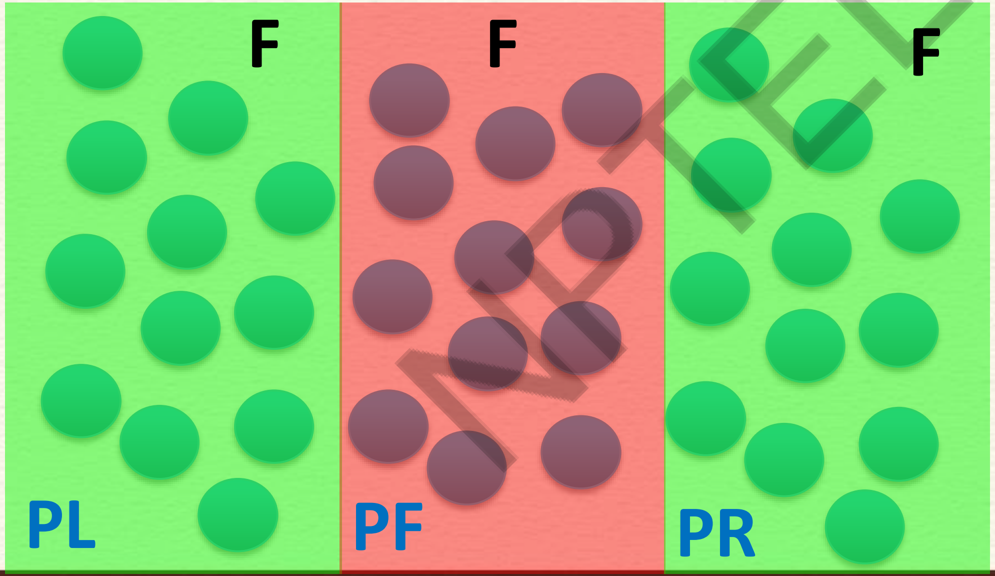
Asynchronous Byzantine Agreement

- F faulty nodes – need 3F + 1 nodes to reach consensus
  - Faulty nodes create partition in the network



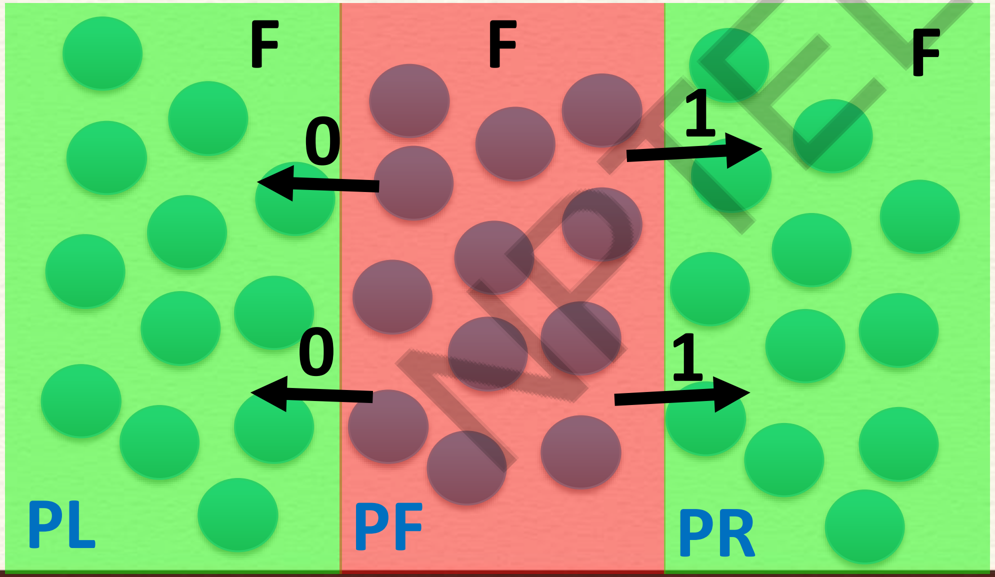
Asynchronous Byzantine Agreement

- F faulty nodes – need 3F + 1 nodes to reach consensus
  - Faulty nodes create partition in the network



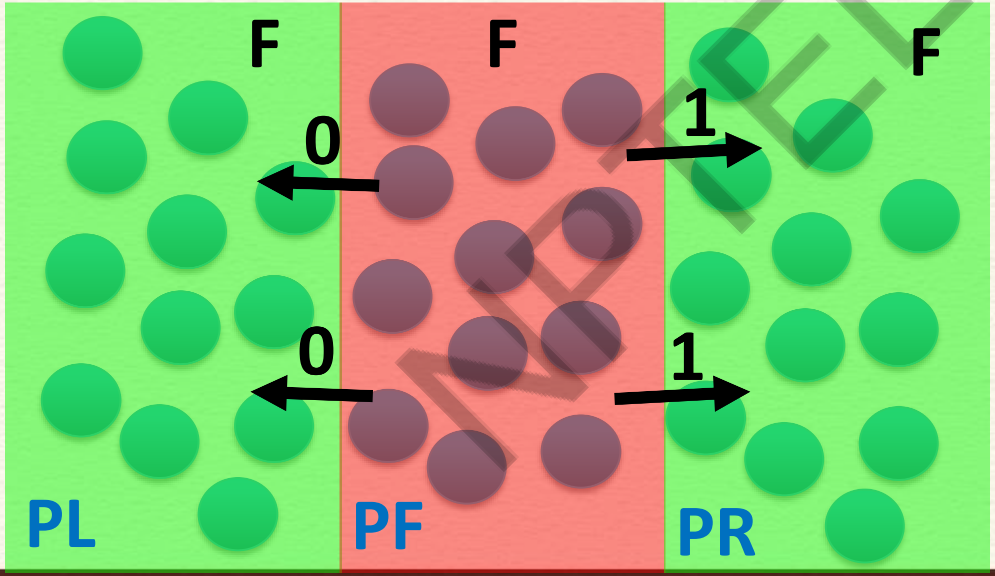
Asynchronous Byzantine Agreement

- F faulty nodes – need 3F + 1 nodes to reach consensus
  - Faulty nodes create partition in the network



Asynchronous Byzantine Agreement

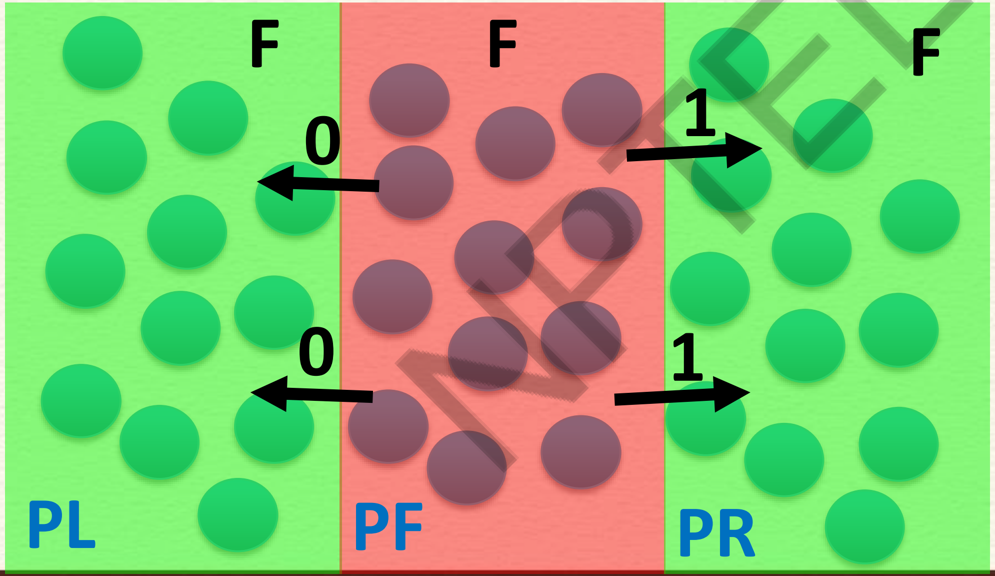
- F faulty nodes – need 3F + 1 nodes to reach consensus
  - Faulty nodes create partition in the network



Either PL or PR must break the tie

Asynchronous Byzantine Agreement

- F faulty nodes – need 3F + 1 nodes to reach consensus
  - Faulty nodes create partition in the network



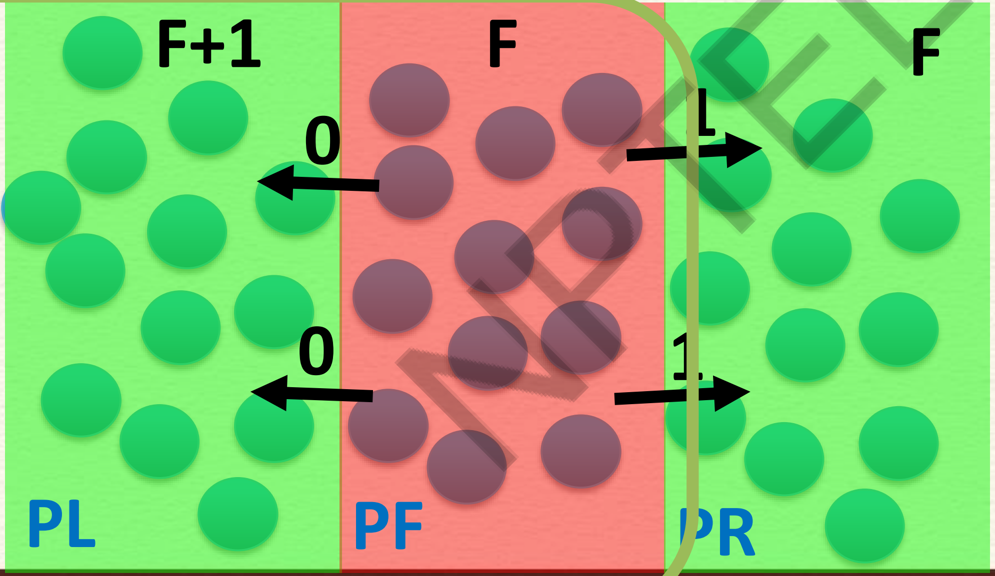
Put one additional node to PL / PR

Asynchronous Byzantine Agreement

- F faulty nodes – need 3F + 1 nodes to reach consensus
  - Faulty nodes create partition in the network

Put one additional node to PL / PR

Breaks the tie to reach consensus



Conclusion

- Paxos does not work with Byzantine faults
  - Much harder to solve than crash faults
- With F faulty nodes, we need 3F + 1 nodes to reach consensus in a BFT system

Thank you