CS 60002: Distributed Systems

T6:
Fault Tolerance

Department of Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Sandip Chakraborty
sandipc@cse.iitkgp.ac.in

Networks and Failures

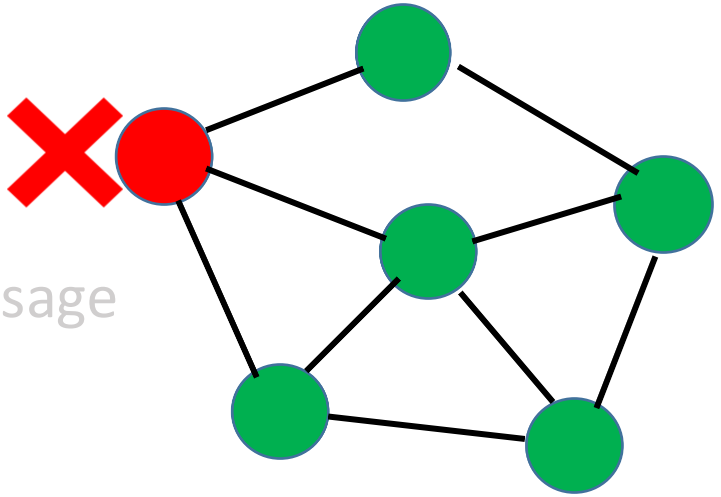
- Synchronous vs Asynchronous Networks
  - **Synchronous**: I am sure that I'll get the message within a predefined time threshold
  - **Asynchronous**: I am not sure whether and when the message will arrive

- Failures in a network --
  - **Crash Fault**: A node stops responding
  - **Link Fault** (or Network Fault): A link fails to deliver the message
  - **Byzantine Fault**: A node starts behaving maliciously

Failures in a Distributed System

- Synchronous vs Asynchronous Networks
  - **Synchronous**: I am sure that I'll get the message within a predefined time threshold
  - **Asynchronous**: I am not sure whether and when the message will arrive

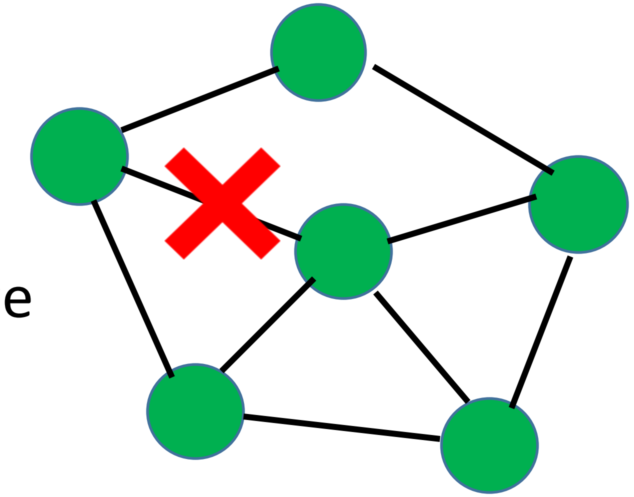
- Failures in a network --
  - **Crash Fault**: A node stops responding
  - **Link Fault** (or Network Fault): A link fails to deliver the message
  - **Byzantine Fault**: A node starts behaving maliciously



Failures in a Distributed System

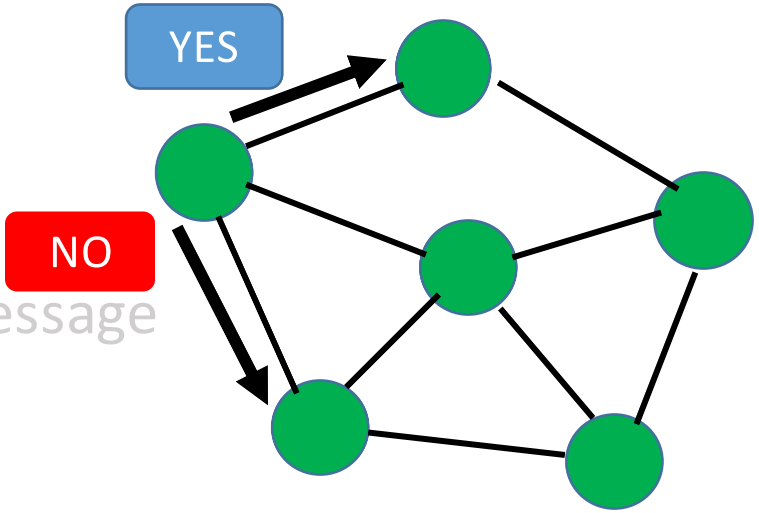
- Synchronous vs Asynchronous Networks
  - **Synchronous**: I am sure that I'll get the message within a predefined time threshold
  - **Asynchronous**: I am not sure whether and when the message will arrive

- Failures in a network --
  - **Crash Fault**: A node stops responding
  - **Link Fault** (or Network Fault): A link fails to deliver the message
  - **Byzantine Fault**: A node starts behaving maliciously



Failures in a Distributed System

- Synchronous vs Asynchronous Networks
  - **Synchronous**: I am sure that I'll get the message within a predefined time threshold
  - **Asynchronous**: I am not sure whether and when the message will arrive



- Failures in a network --
  - **Crash Fault**: A node stops responding
  - **Link Fault** (or Network Fault): A link fails to deliver the message
  - **Byzantine Fault**: A node starts behaving maliciously

Dependability in the Presence of Failures

- A measure of how dependable a system is in the face of failures

These slides are taken from the Distributed Systems course by Prof. Arobinda Gupta, IIT Kharagpur

Dependability in the Presence of Failures

- A measure of how dependable a system is in the face of failures
  - Reliability
  - Availability
  - Safety

These slides are taken from the Distributed Systems course by Prof. Arobinda Gupta, IIT Kharagpur

Dependability in the Presence of Failures

- A measure of how dependable a system is in the face of failures
  - Reliability
  - Availability
  - Safety

- **Reliability**
  - How often does the system fail?
  - What is the conditional probability that the system will work for the duration [0,t] given that it is working at time zero?
  - **Measured by**: **MTTF** (Mean Time To Failures), **MTTR** (Mean Time To Repair), **MTBF** (Mean Time Between Failures = MTTF + MTTR)

These slides are taken from the Distributed Systems course by Prof. Arobinda Gupta, IIT Kharagpur

Dependability in the Presence of Failures

- **Availability**
    - How available the system is
    - What is the probability that the system is up at time t?
    - Usually measured by uptime (ex. 99%, maximum downtime of 5 hours in 1 year, etc.)

These slides are taken from the Distributed Systems course by Prof. Arobinda Gupta, IIT Kharagpur

Dependability in the Presence of Failures

- **Availability**
  - How available the system is
  - What is the probability that the system is up at time t?
  - Usually measured by uptime (ex. 99%, maximum downtime of 5 hours in 1 year, etc.)

- **Safety**
  - How safe the system is, even if it fails
  - Does it always maintain some safety property?

These slides are taken from the Distributed Systems course by Prof. Arobinda Gupta, IIT Kharagpur

Reliability vs Availability

- A highly reliable system is also highly available
- A highly available system may or may not be highly reliable
  - Ex: If a system fails for 1 second every hour, it can still be considered highly available (99.97%) but not highly reliable
- The reliability of a system depends on the reliability of the components used to build the system
- Reliability/Availability can be of interest at different component levels
  - A memory chip
  - A disk controller with memory
  - A PC with disks
  - A cluster with a large number of PCs

Fault Tolerance

- The ability of a system to deliver desired services in spite of faults in its components

- Fault tolerance can be at the level of
  - A full service (specified behavior in fault-free state); ex. A primary-backup server system to tolerate one server failure
  - A degraded service (deviate from the specified behavior in fault-free state, but in a pre-defined manner); ex. A web service with multiple load balanced servers

Fault Tolerance

- The ability of a system to deliver desired services in spite of faults in its components

- Fault tolerance can be at the level of
  - A full service (specified behavior in fault-free state); ex. A primary-backup server system to tolerate one server failure
  - A degraded service (deviate from the specified behavior in fault-free state, but in a pre-defined manner); ex. A web service with multiple load balanced servers

- Many modern distributed system needs to be highly available
  - Gmail
  - Facebook
  - Airline reservation system

These slides are taken from the Distributed Systems course by Prof. Arobinda Gupta, IIT Kharagpur

Types of Fault Tolerance

- **Masking:** Always behave as per specification even in the presence of faults in the system

- **Non-masking**: System may violate specification in presence of faults, but behave in a well-defined manner

- A fault tolerant system should specify
  - Class of faults tolerated (**Fault Model)**
  - What tolerance is given for each class (**Fault Tolerance**)

Primitive Operations for Fault Tolerance

- Building reliable storage from unreliable disks
  - RAID
  - Centralized network storage
- Reliable communication over unreliable links
  - Unicast, multicast, broadcast
- Agreement/Consensus
- Enforce atomic actions
- Checkpoint and Recovery

Agreement Problem

- A set of n processes, m of them may be faulty
- Non-faulty processes need to agree on some value(s) even in the presence of faulty processes

- One of the most studied problems in Distributed System
  - Agreement (Typically used for handling Byzantine faults, so use the term **Byzantine agreement** or **Byzantine Generals Problem**)
  - Consensus
  - Interactive Consistency

- All three problems are equivalent; solution of any one of them can be used to solve the other two

Agreement Protocol

- One process x broadcast a value v
- All non-faulty processes must agree on a common value (**agreement condition**)
- The agreed upon value must be v is x is non-faulty (**validity condition**)

- This idea is used to solve the **Byzantine Generals Problem** → **Byzantine Agreement Protocols**

These slides are taken from the Distributed Systems course by Prof. Arobinda Gupta, IIT Kharagpur

Byzantine Generals Problem

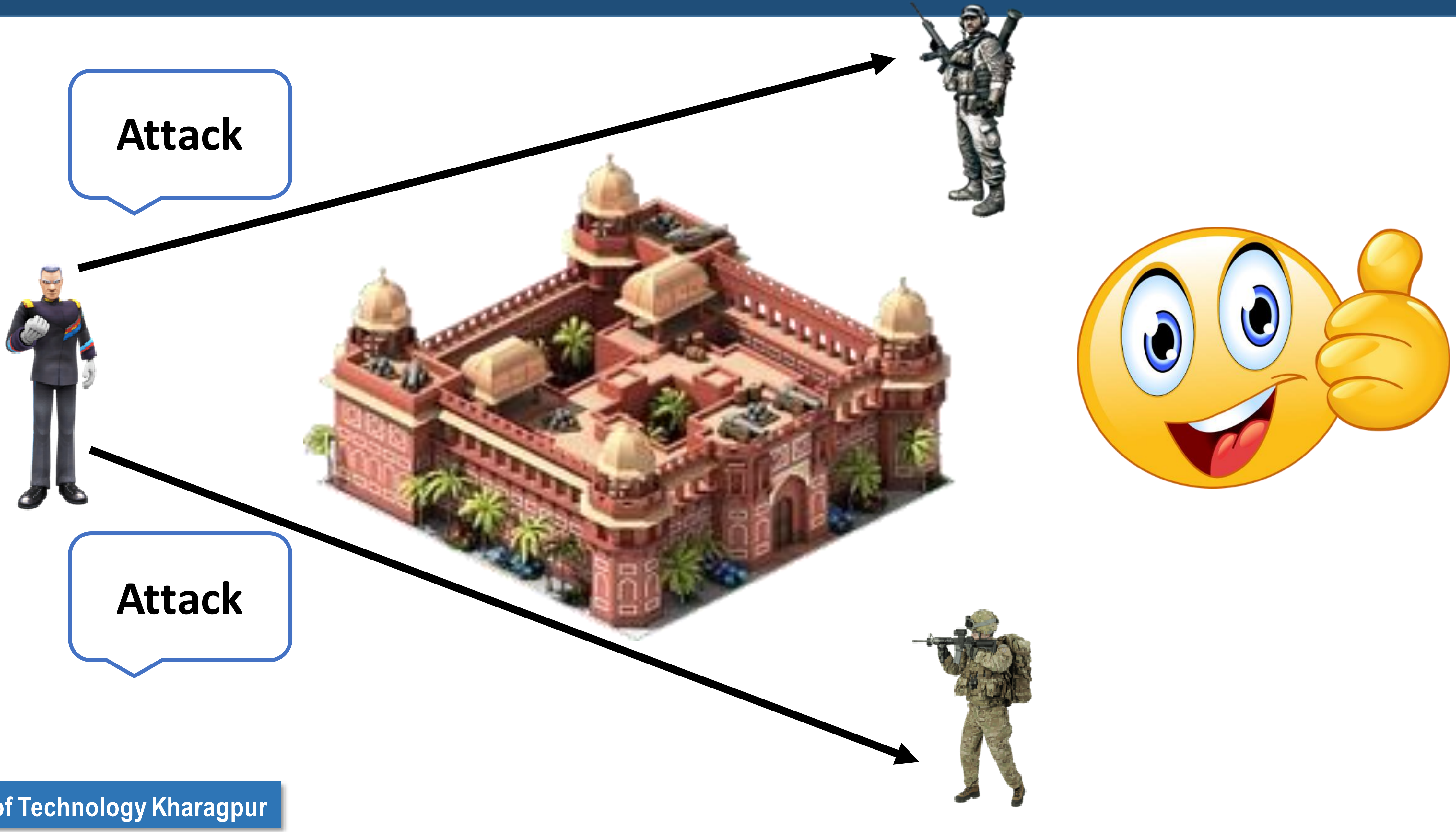
Commander

Click to add text

Lieutenant - 1

Lieutenant - 2

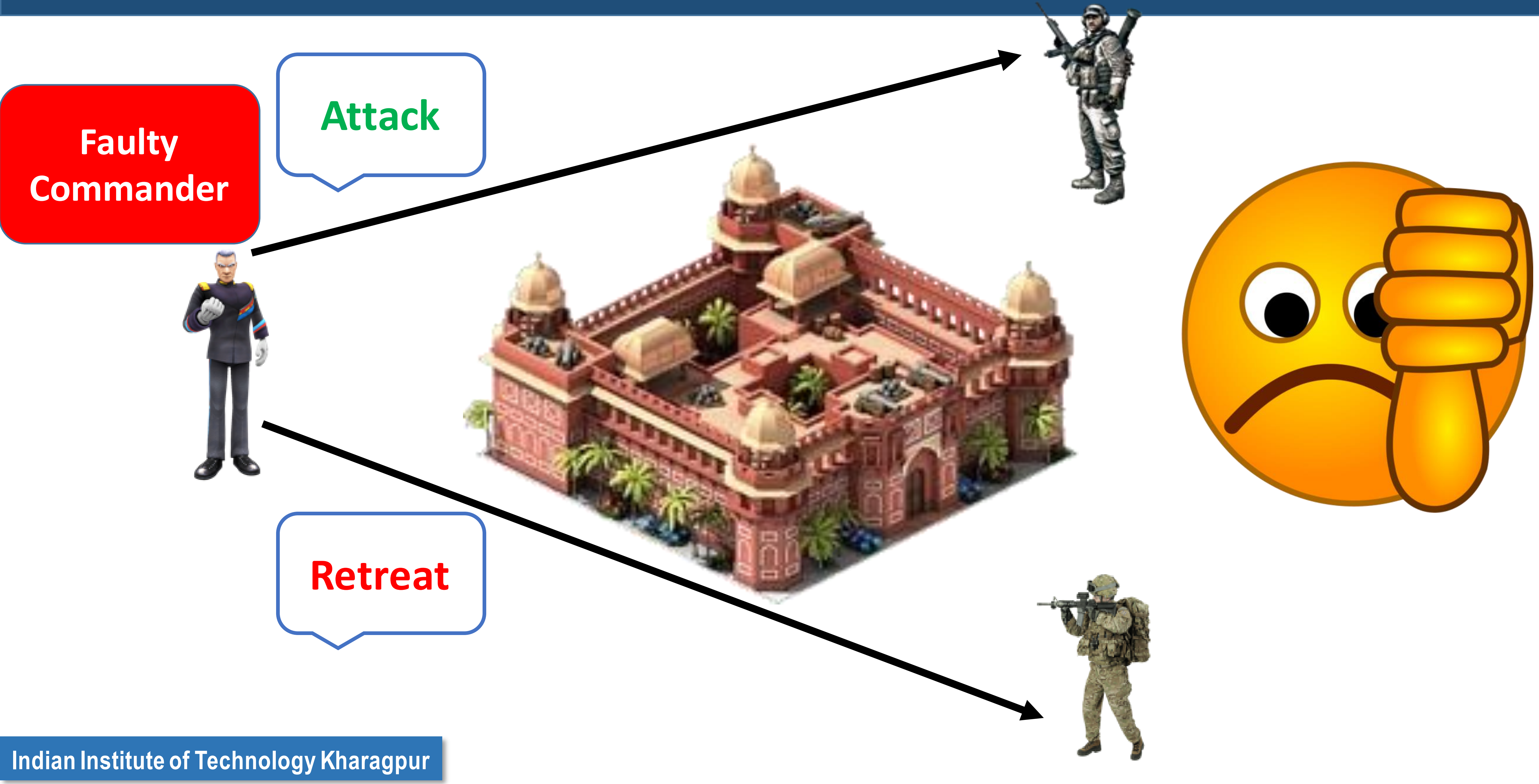
Byzantine Generals Problem



Attack

Attack

Byzantine Generals Problem

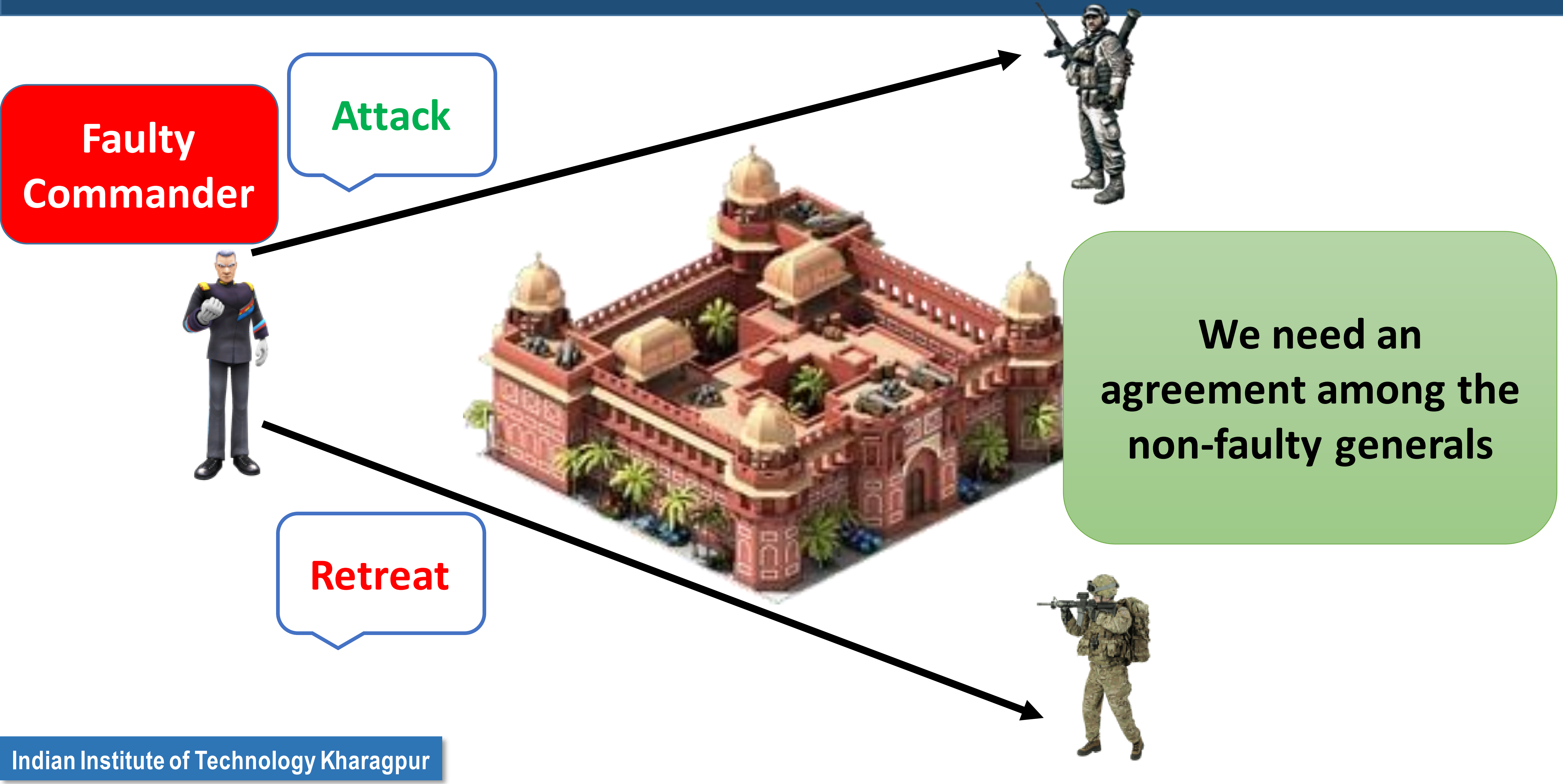


Faulty Commander

Attack

Retreat

Byzantine Generals Problem



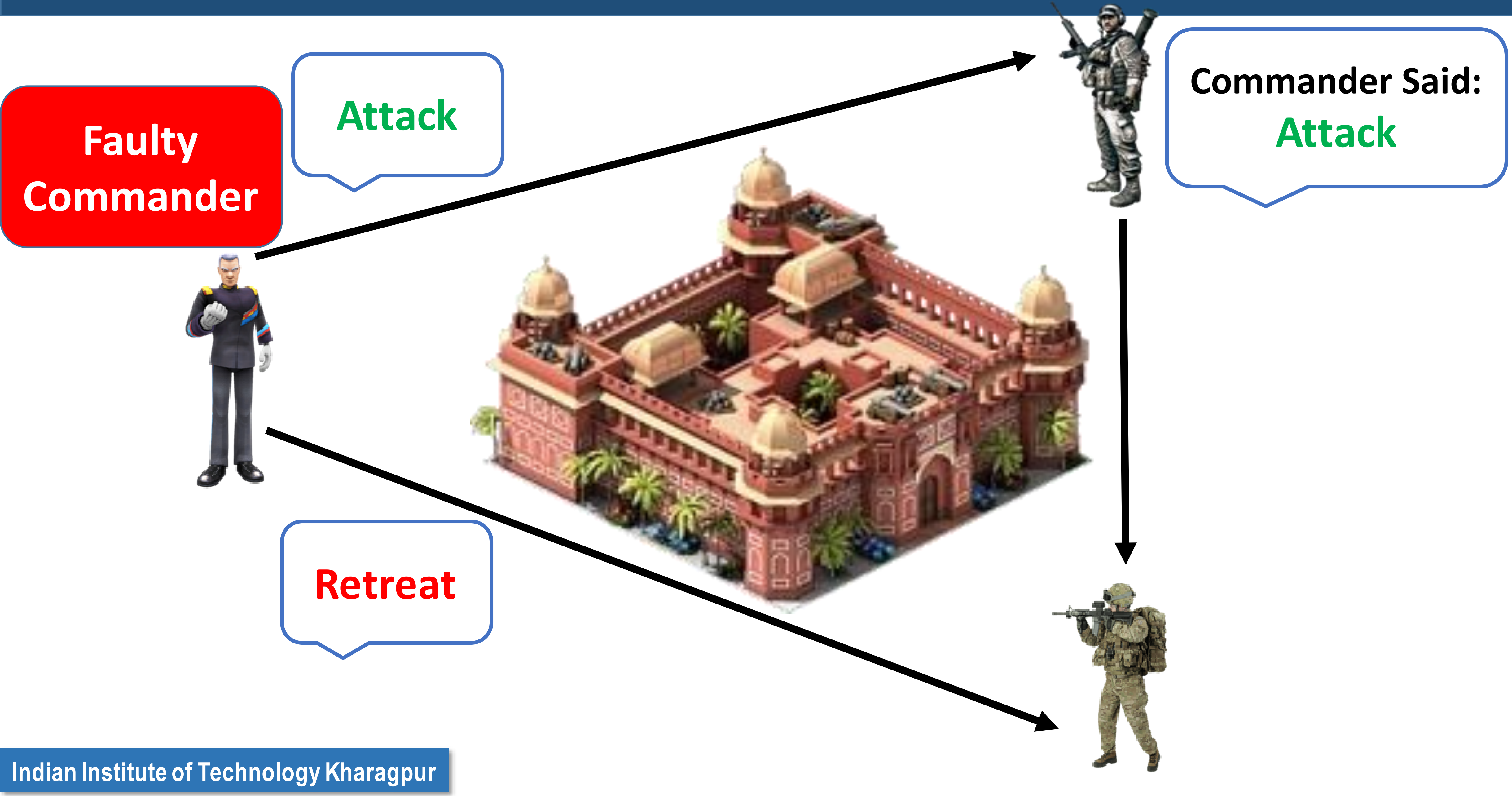
Faulty Commander

Attack

Retreat

We need an agreement among the non-faulty generals

Byzantine Generals Problem



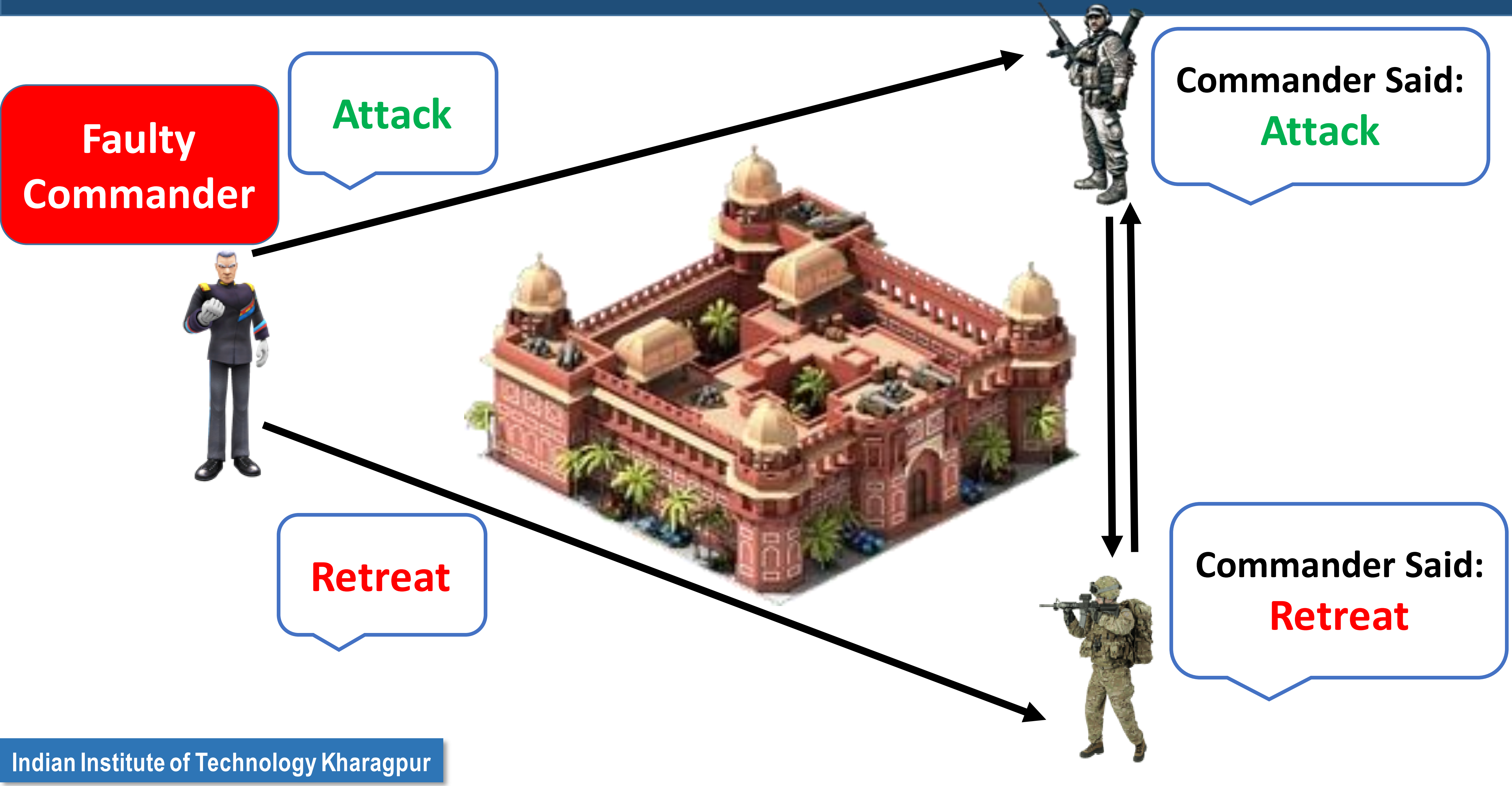
Faulty Commander

Attack

Retreat

Commander Said: Attack

Byzantine Generals Problem



Faulty Commander

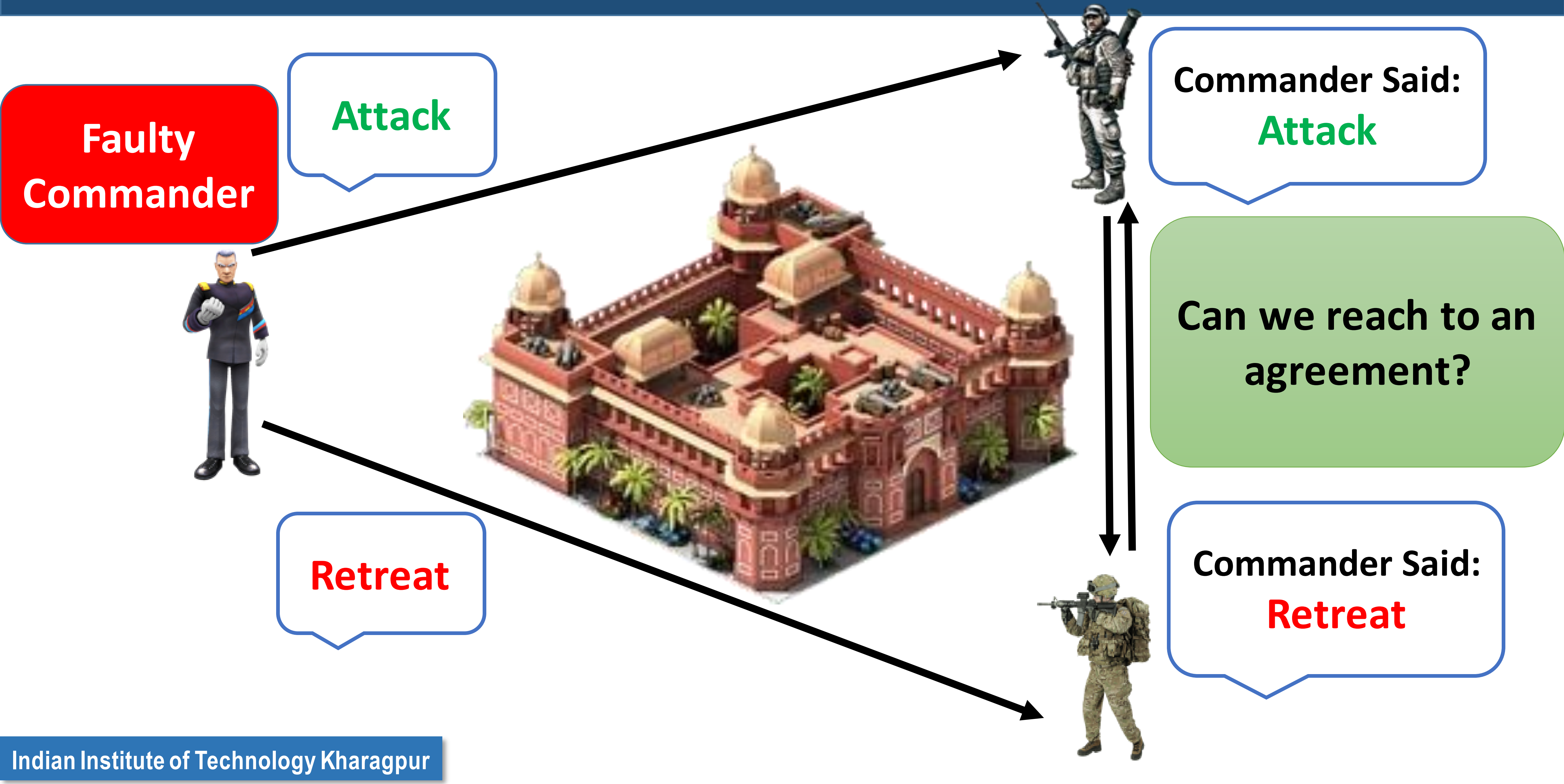
Attack

Retreat

Commander Said: Attack

Commander Said: Retreat

Byzantine Generals Problem



Faulty Commander

Attack

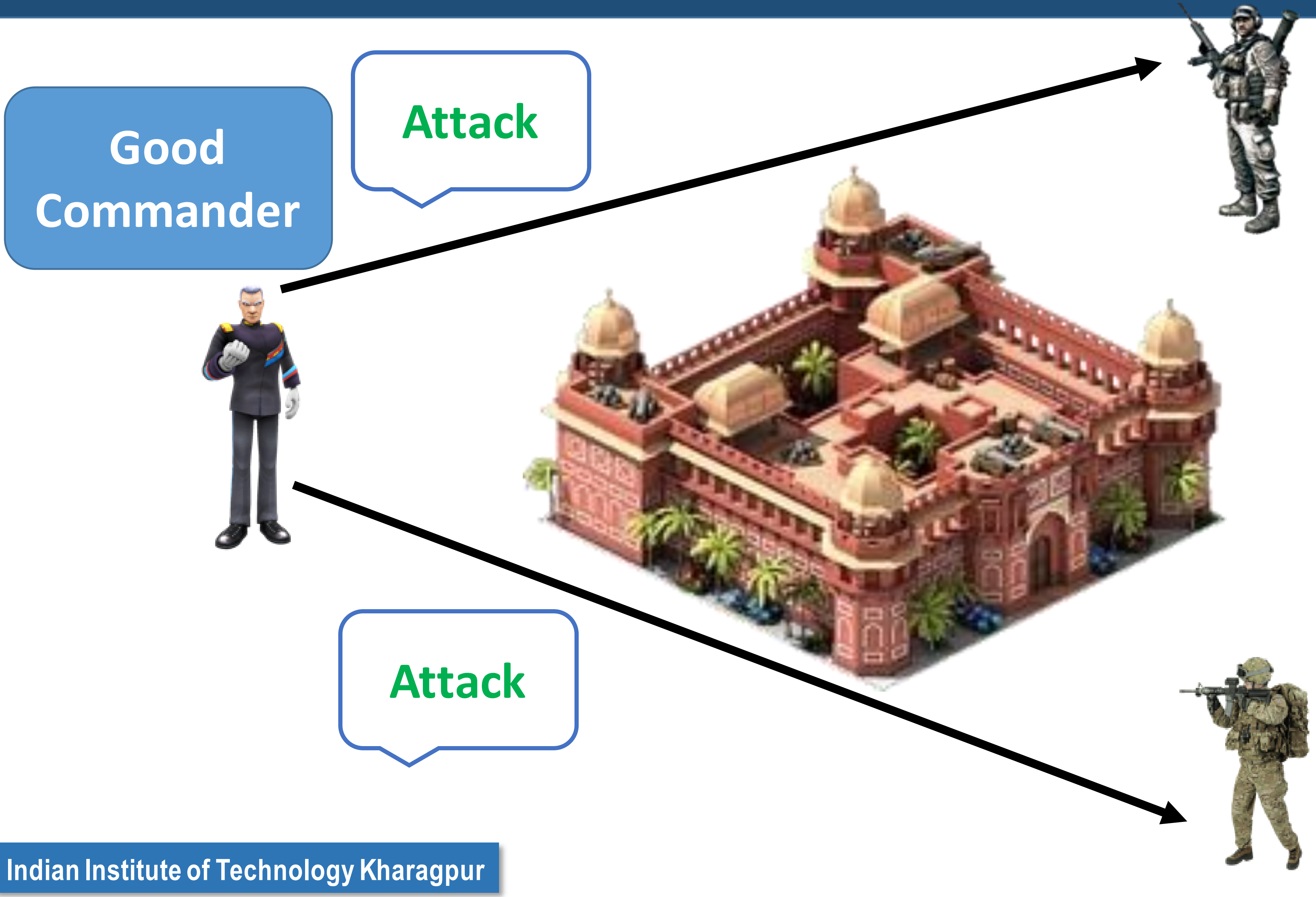
Retreat

Commander Said: Attack

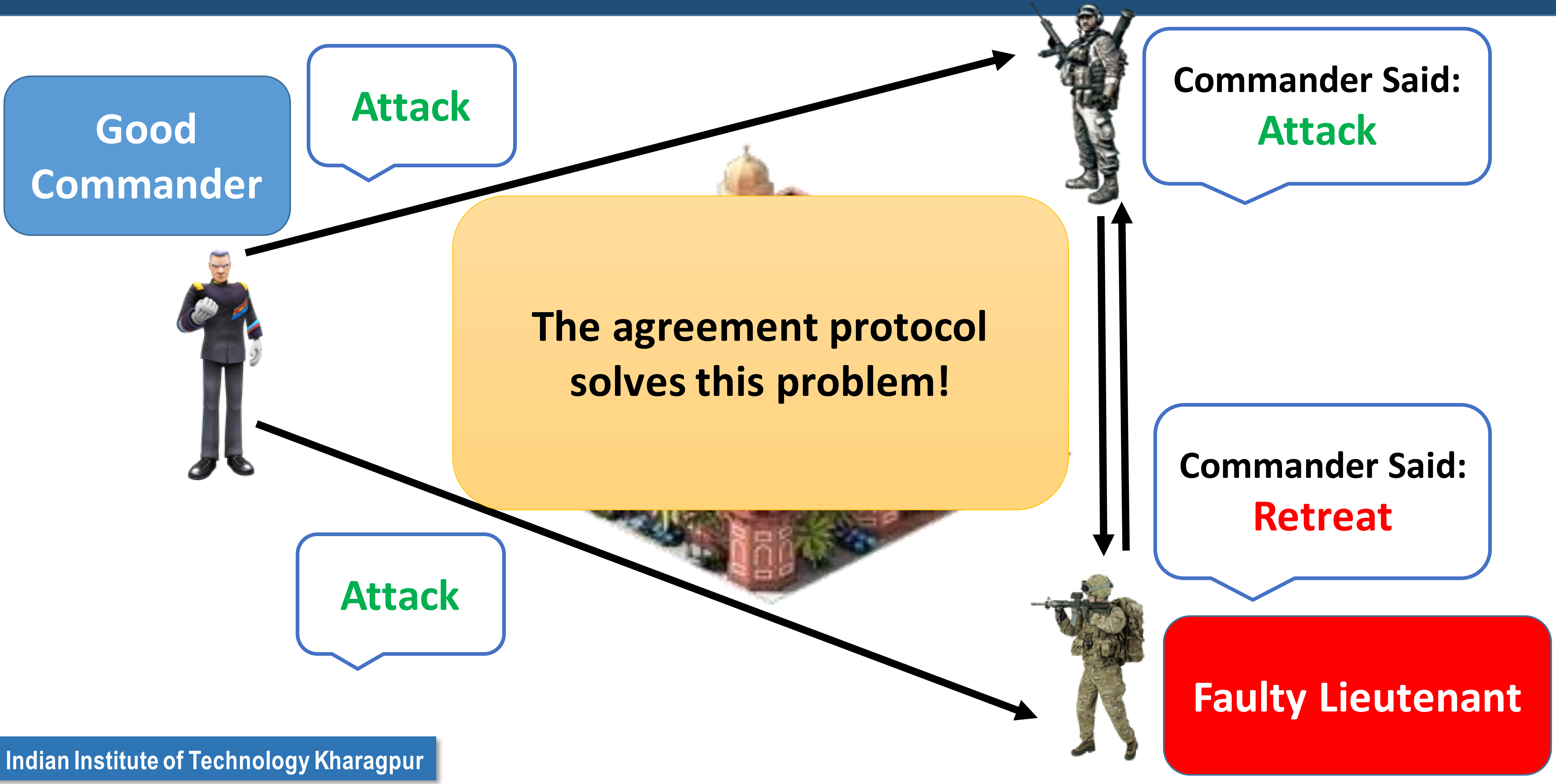
Can we reach to an agreement?

Commander Said: Retreat

Byzantine Generals Problem



Byzantine Generals Problem



Good Commander

Attack

Commander Said: Attack

The agreement protocol solves this problem!

Commander Said: Retreat

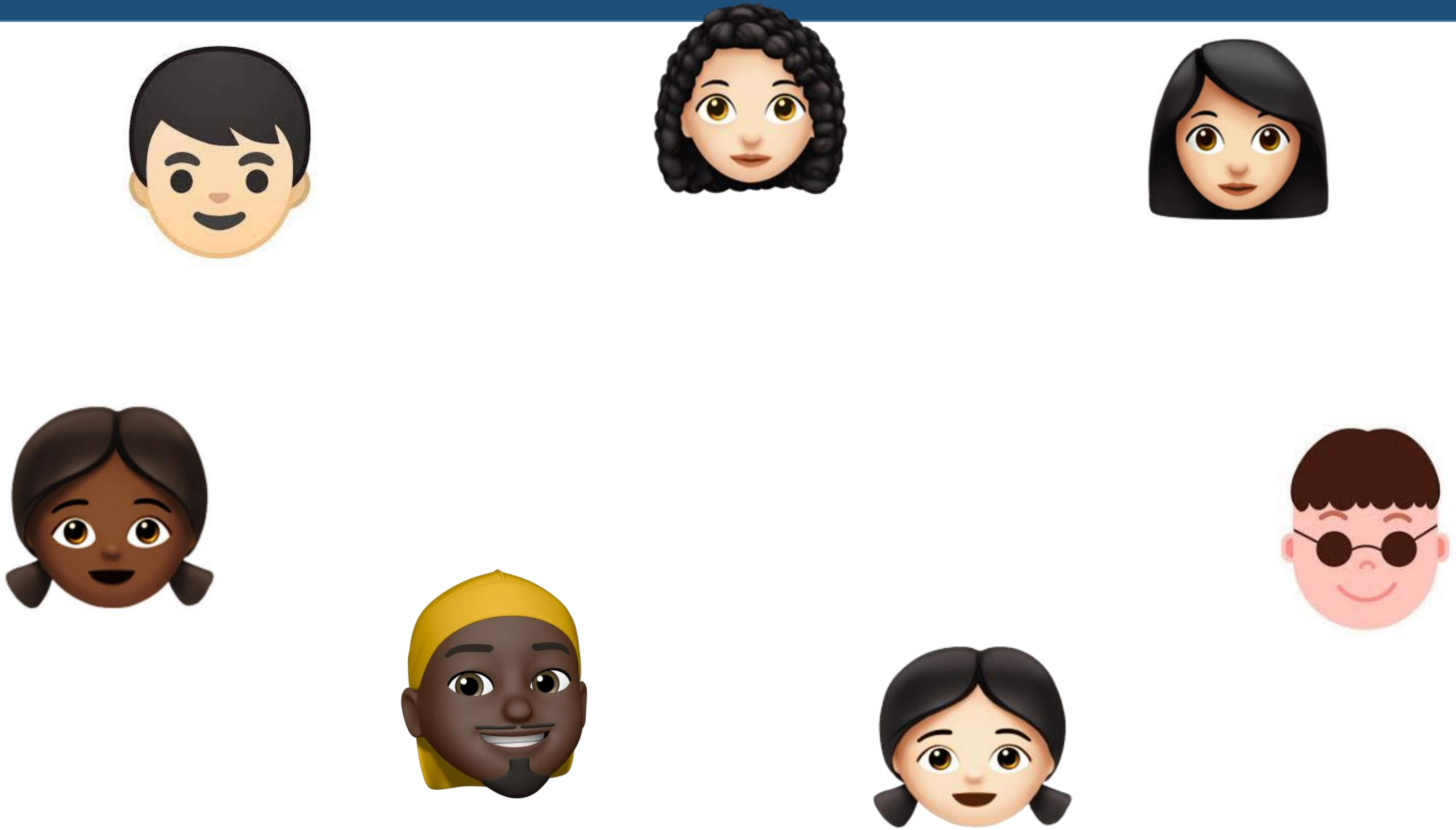
Attack

Faulty Lieutenant

Consensus

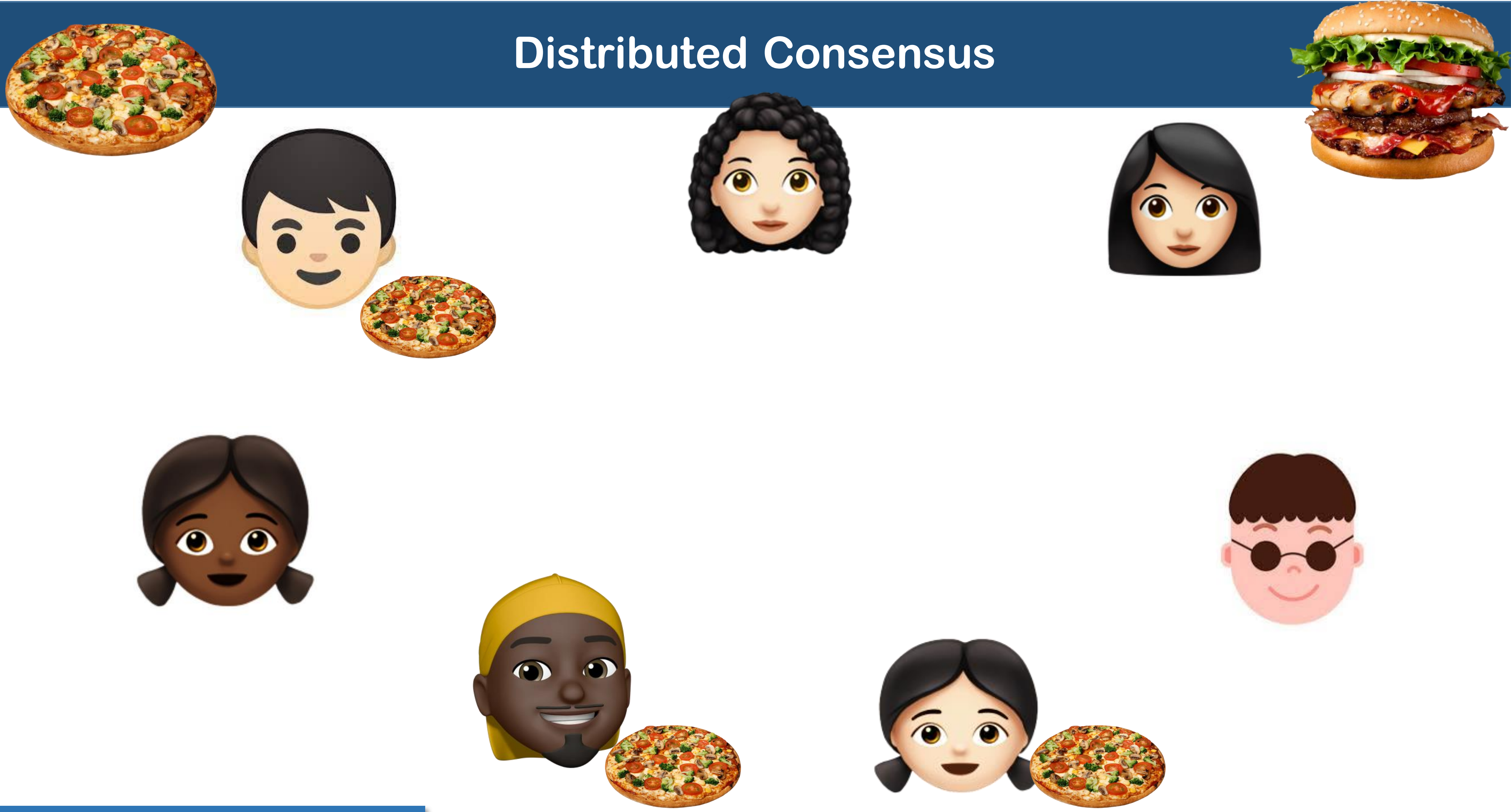
- Each process broadcast its initial value
  - Satisfy agreement condition
  - If initial value of all non-faulty processes is v, then the agreed upon value must be v

Distributed Consensus

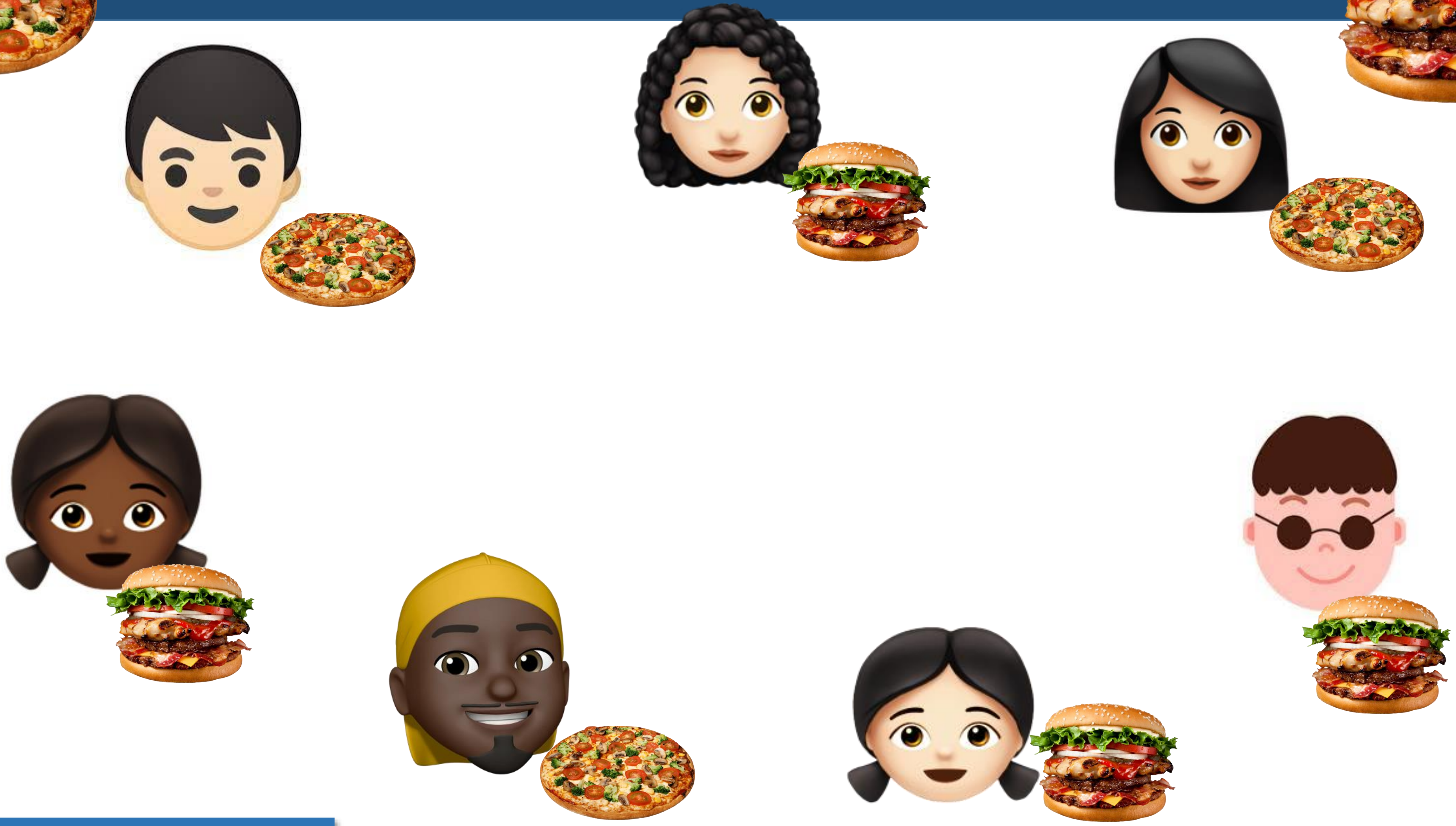


Distributed Consensus

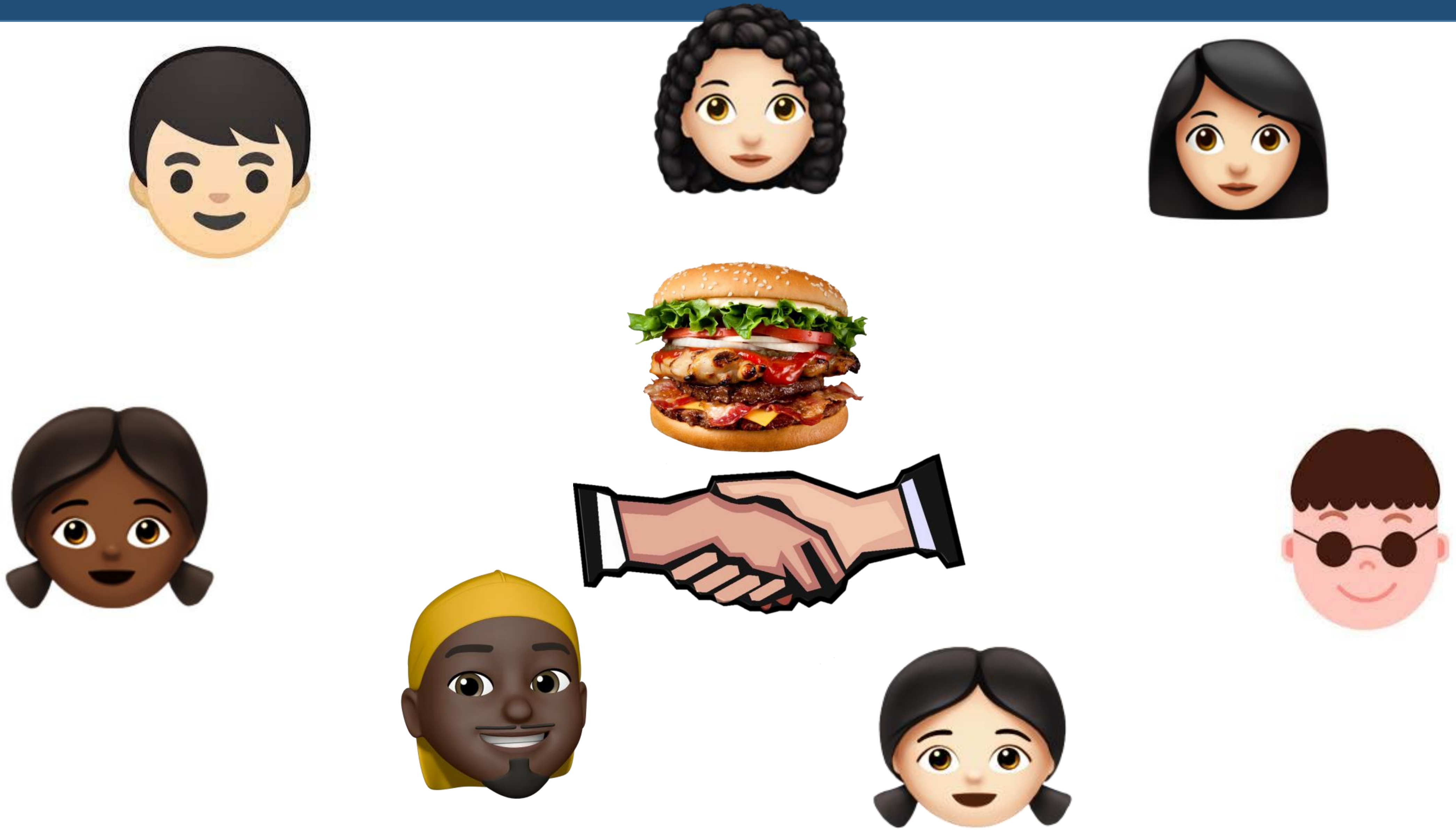
Distributed Consensus



Distributed Consensus



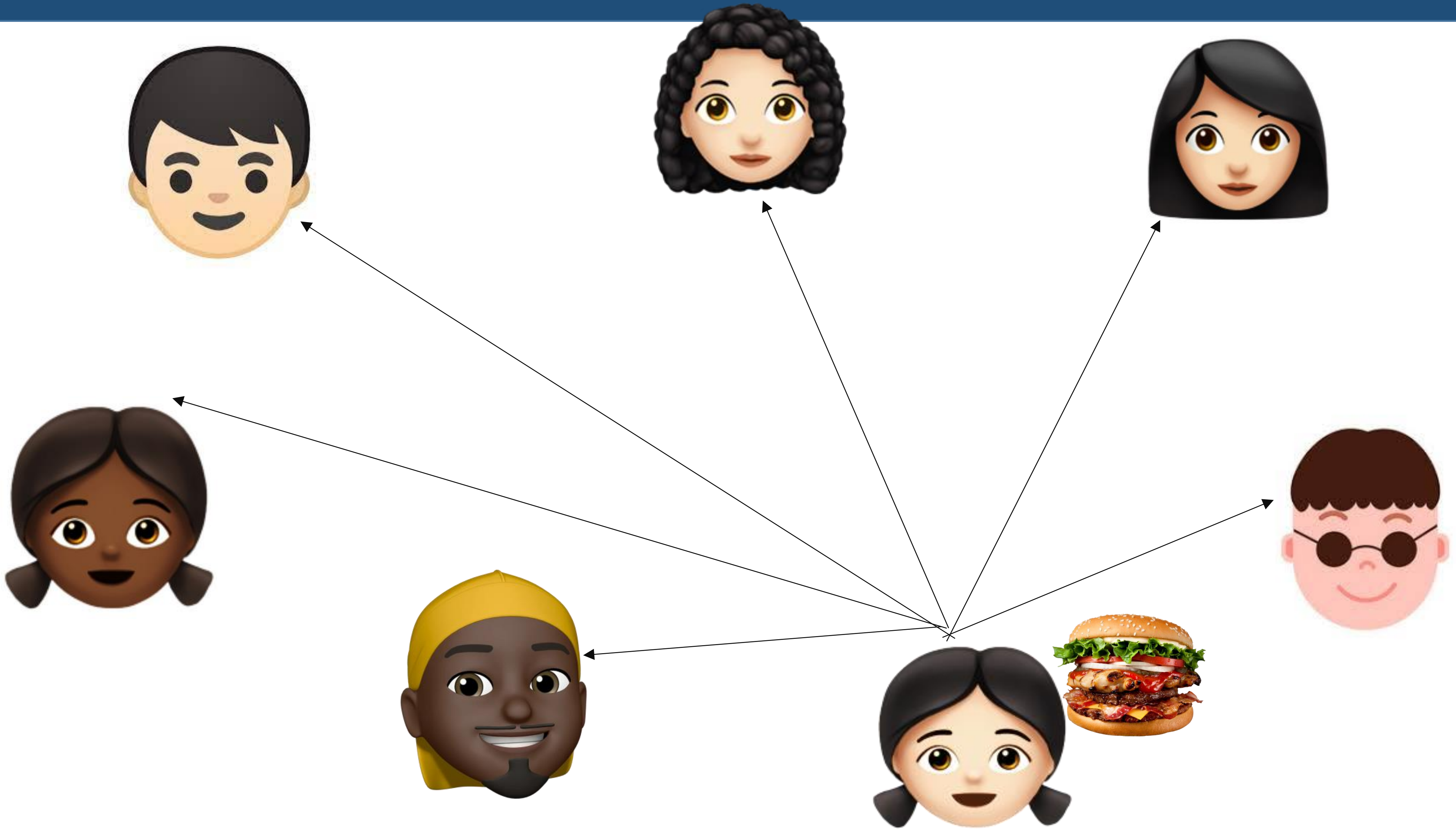
Distributed Consensus



Distributed Consensus

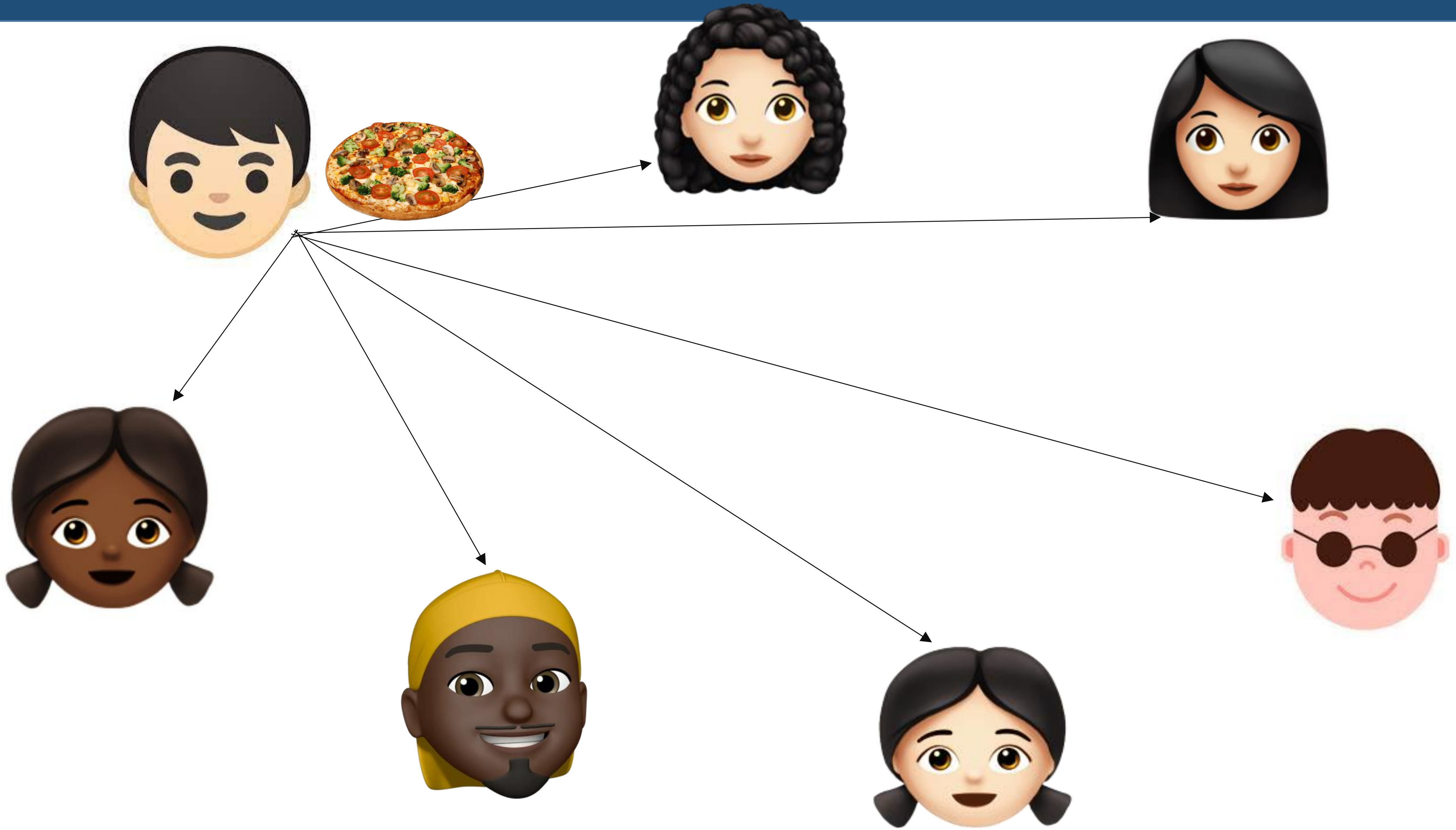
How can we make this decision in a distributed way?

Distributed Consensus – Message Passing



Indian Institute of Technology Kharagpur

Distributed Consensus – Message Passing

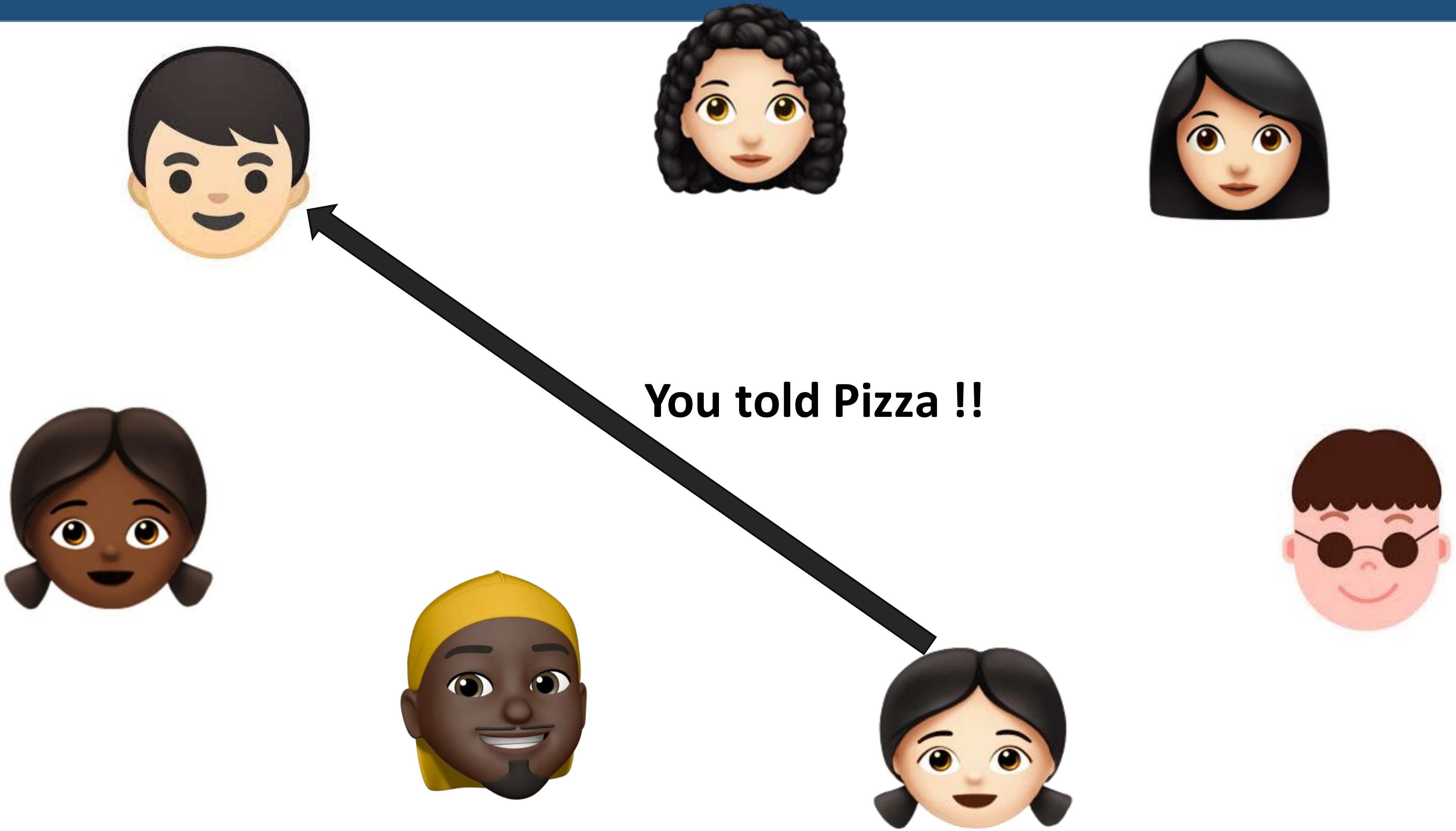


Distributed Consensus – Message Passing

3
4
3
4
3
4
3
4
3
4
3
4
3
4

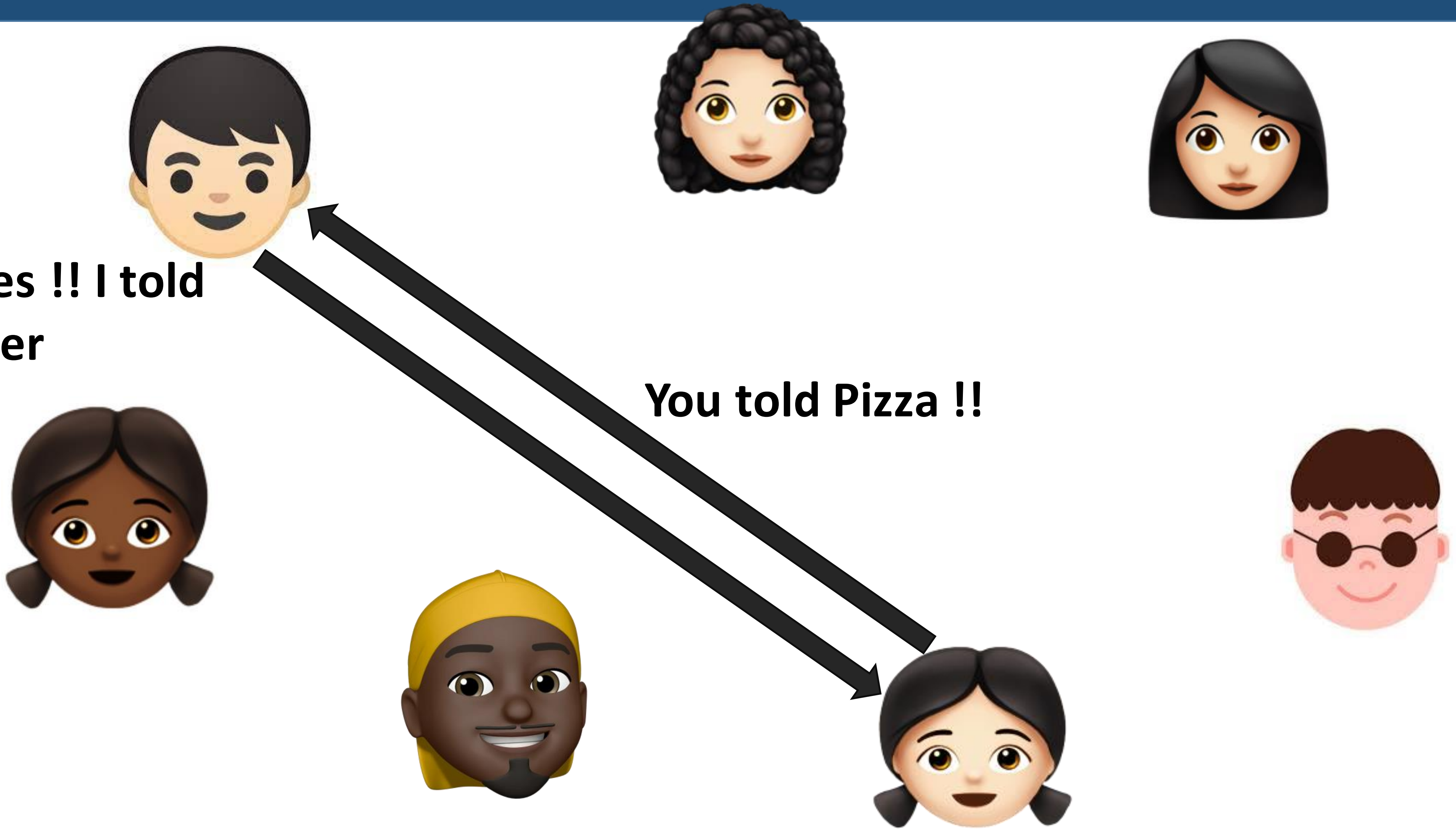
Count votes and decide!

Distributed Consensus – Message Passing



You told Pizza !!

Distributed Consensus – Message Passing



Nopes !! I told Burger

You told Pizza !!

Interactive Consistency

- Each process i broadcasts its own value v_i
  - All non-faulty processes agree on a common vector $\{v_1, v_2, \ldots, v_n\}$
  - If ith process is non-faulty, then the ith value in the vector agreed upon by non-faulty processes must be v_i

Distributed Consensus

- 1985: FLP Impossibility Theorem – Fischer, Lynch, Paterson
  - Consensus is impossible in a fully asynchronous system even with a single crash fault

Distributed Consensus

- 1985: FLP Impossibility Theorem – Fischer, Lynch, Paterson
  - Consensus is impossible in a fully asynchronous system even with a single crash fault
  - Cannot ensure "Safety" and "Liveness" together

Distributed Consensus

- 1985: FLP Impossibility Theorem – Fischer, Lynch, Paterson
  - Consensus is impossible in a fully asynchronous system even with a single crash fault
  - Cannot ensure "**Safety**" and "Liveness" together

Correct processes will yield the correct output

Distributed Consensus

- 1985: FLP Impossibility Theorem – Fischer, Lynch, Paterson
  - Consensus is impossible in a fully asynchronous system even with a single crash fault
  - Cannot ensure "**Safety**" and "**Liveness**" together

Correct processes will yield the correct output

The output will be produced within a finite amount of time (eventual termination)

Distributed Consensus

- 1985: FLP Impossibility Theorem – Fischer, Lynch, Paterson
  - Consensus is impossible in a fully asynchronous system even with a single crash fault
  - Cannot ensure "**Safety**" and "**Liveness**" together
- 1989: Lamport started talking about "Paxos"
  - Supports safety but not the liveness

Distributed Consensus

- 1985: FLP Impossibility Theorem – Fischer, Lynch, Paterson
  - Consensus is impossible in a fully asynchronous system even with a single crash fault
  - Cannot ensure "**Safety**" and "**Liveness**" together
- 1989: Lamport started talking about "Paxos"
  - Supports safety but not the liveness
- 1990's: Everyone were confused about the correctness of Paxos

Distributed Consensus

- 1985: FLP Impossibility Theorem – Fischer, Lynch, Paterson
  - Consensus is impossible in a fully asynchronous system even with a single crash fault
  - Cannot ensure "**Safety**" and "**Liveness**" together

- 1989: Lamport started talking about "Paxos"
  - Supports safety but not the liveness

- 1990's: Everyone were confused about the correctness of Paxos

- 1998: Paxos got published in ACM Transactions on Computer Systems

Distributed Consensus

- 2001: FLP Impossibility paper wins Dijkstra Prize
    - People starts talking about Distributed Systems

Distributed Consensus

- 2001: FLP Impossibility paper wins Dijkstra Prize
  - People starts talking about Distributed Systems
- 2009: Zookeeper released
  - Service for managing distributed applications
- 2010's onward: Different types of consensus algorithms released
  - Multi-Paxos
  - Raft
  - Byzantine Fault Tolerance
  - PBFT
  - ...

Another Interesting Impossibility Result

- [Santoro and Widmayer, 1989] **Even in a synchronous model, consensus is not possible even with a single link failure**.

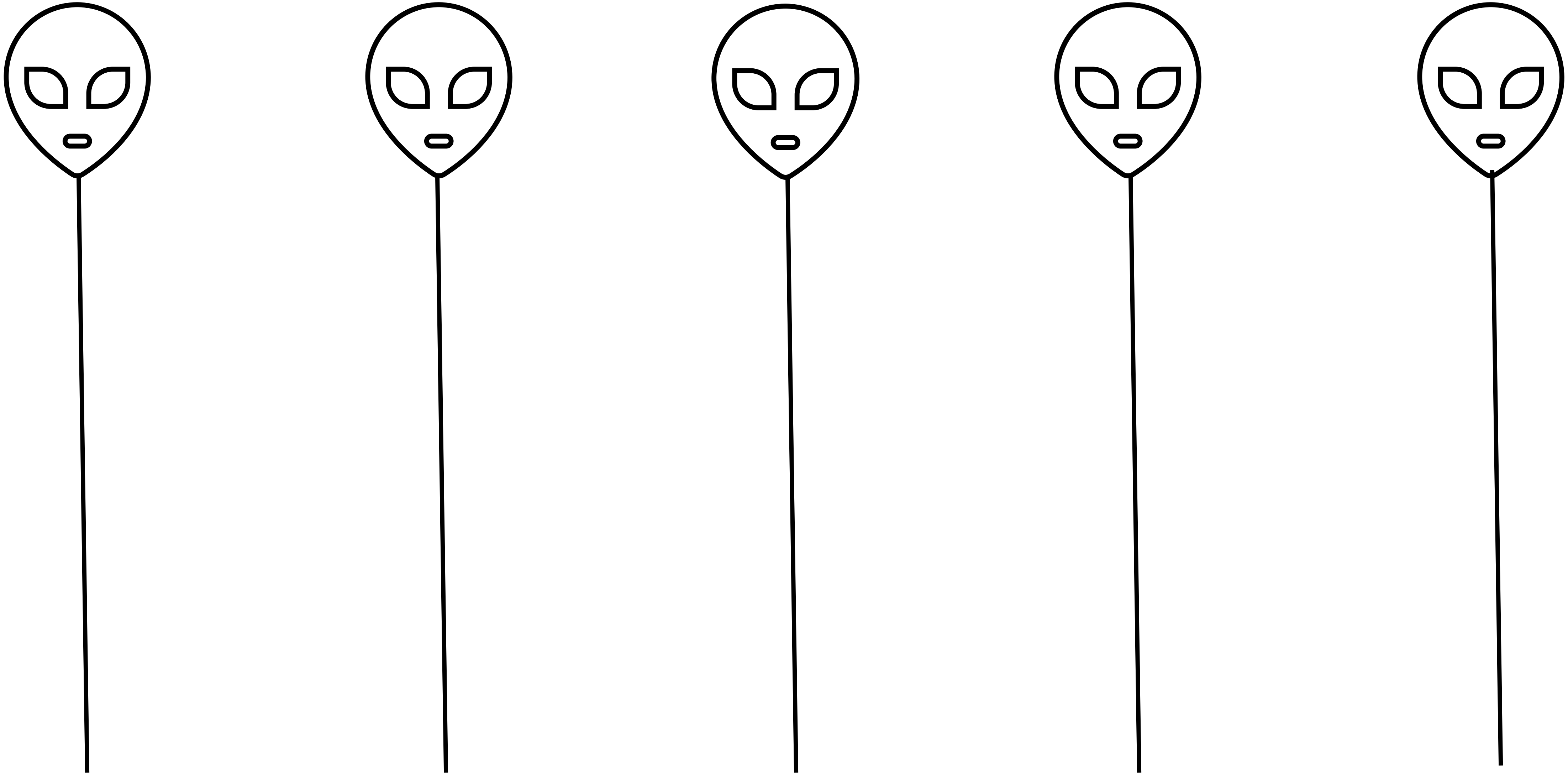
Santoro, Nicola, and Peter Widmayer. "Time is not a healer." *Annual Symposium on Theoretical Aspects of Computer Science*. Springer, Berlin, Heidelberg, 1989.

Asynchronous Consensus with Crash Faults

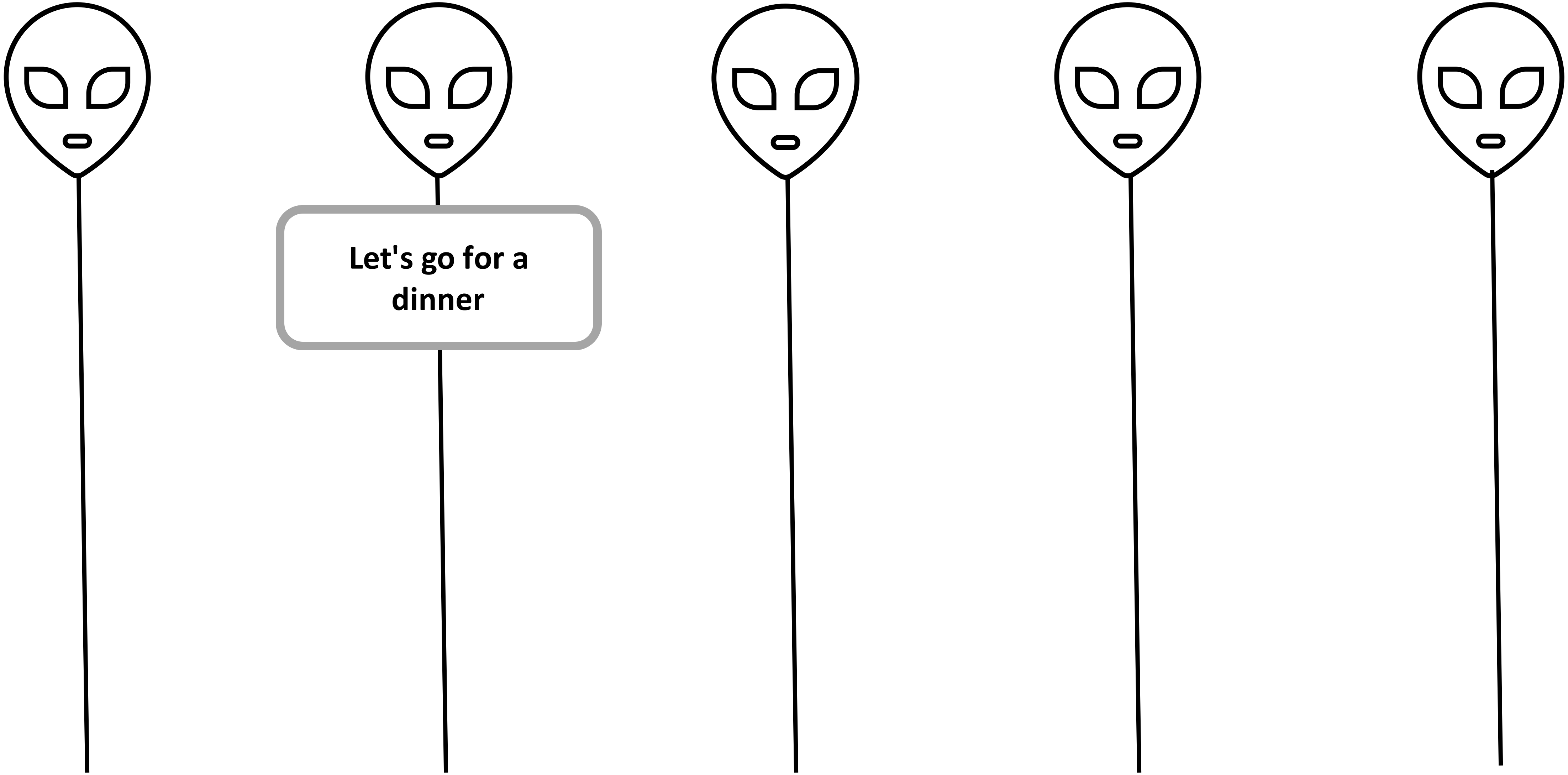
- Remember the **FLP Impossibility**
  - Give priority to safety over liveness

- 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

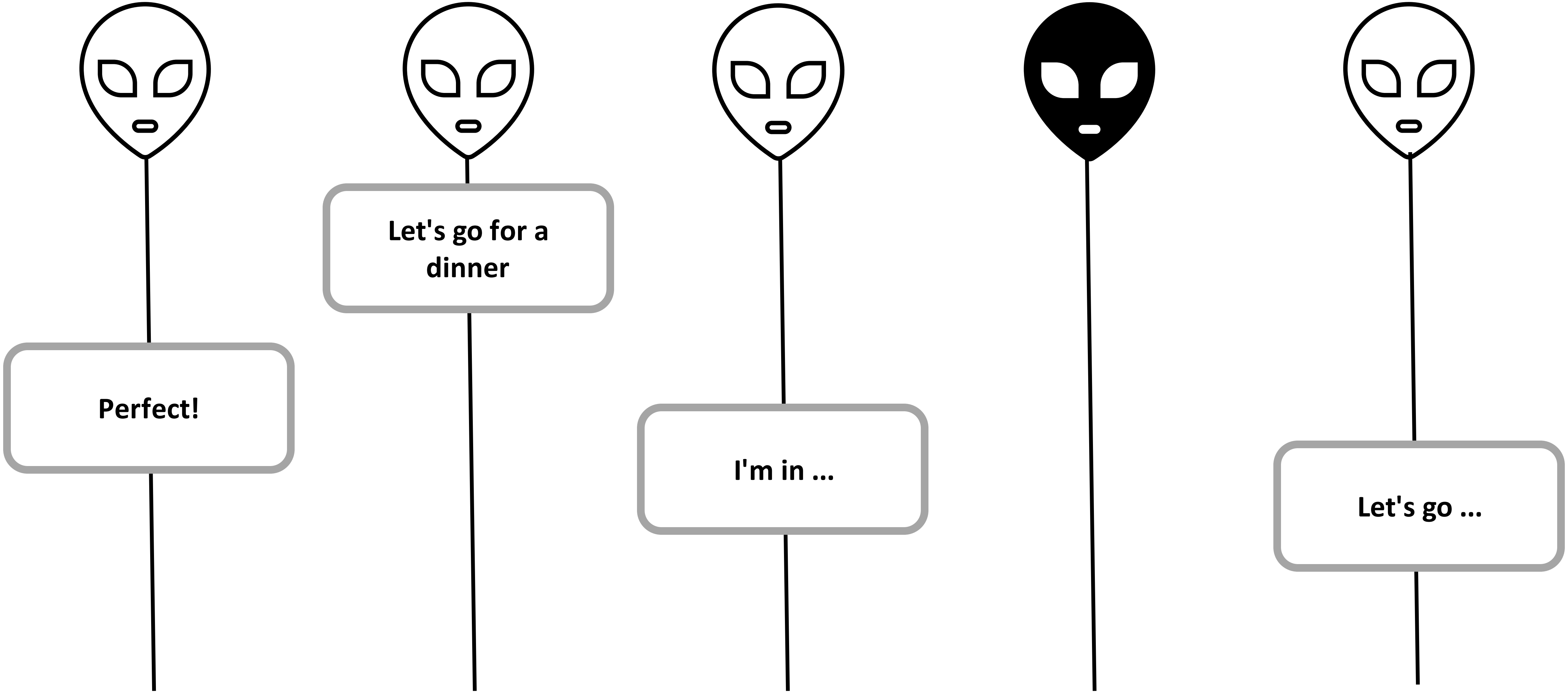
CFT Consensus in a Synchronous System



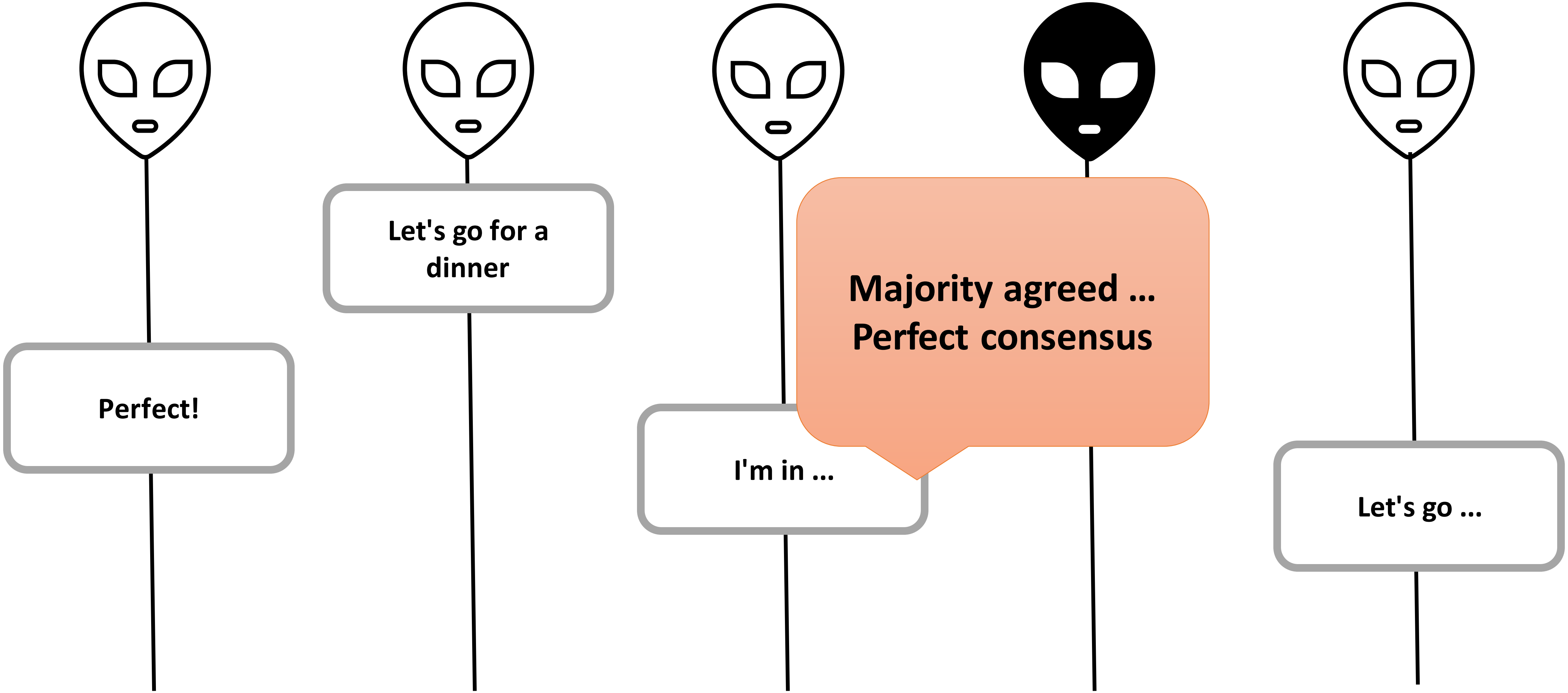
CFT Consensus in a Synchronous System



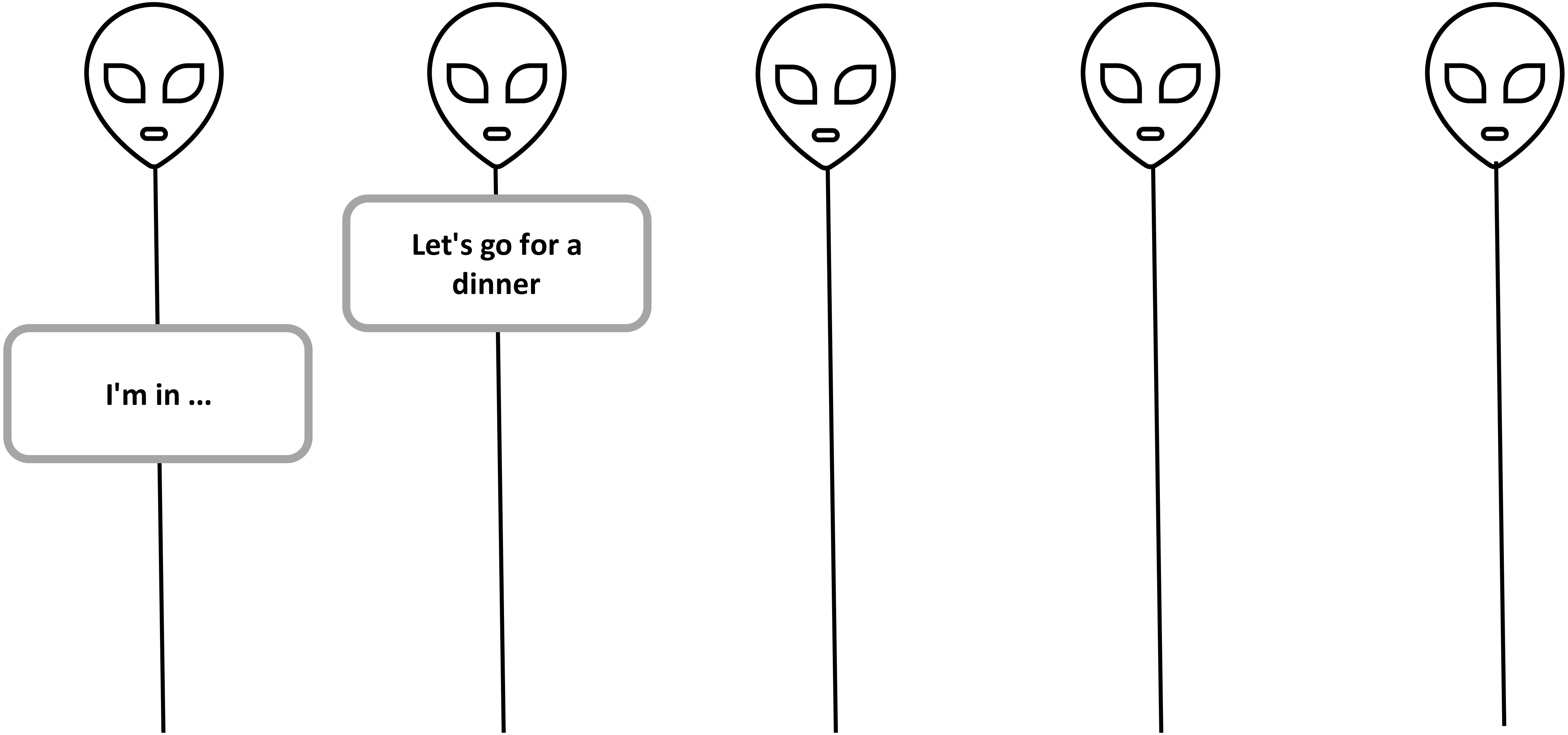
CFT Consensus in a Synchronous System



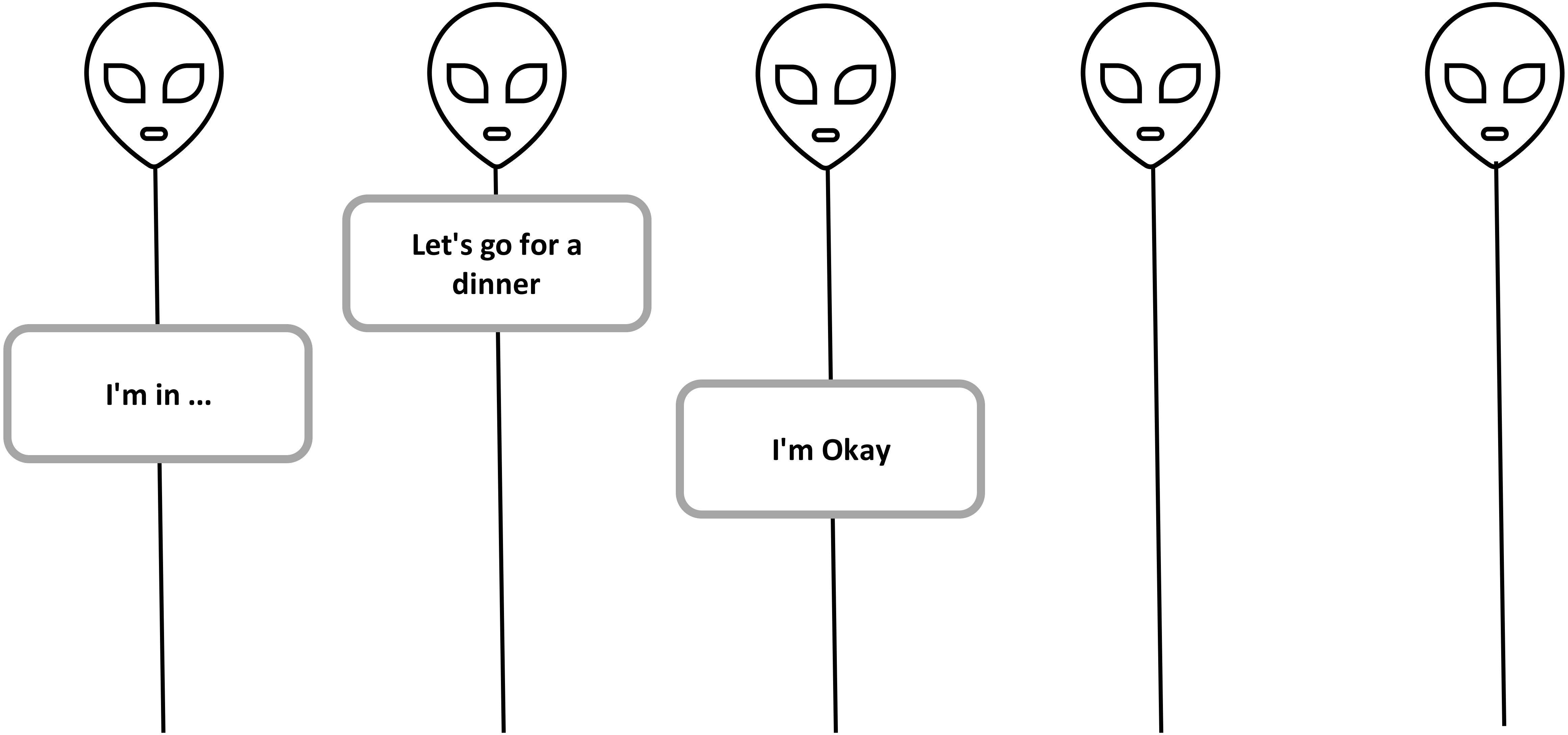
CFT Consensus in a Synchronous System



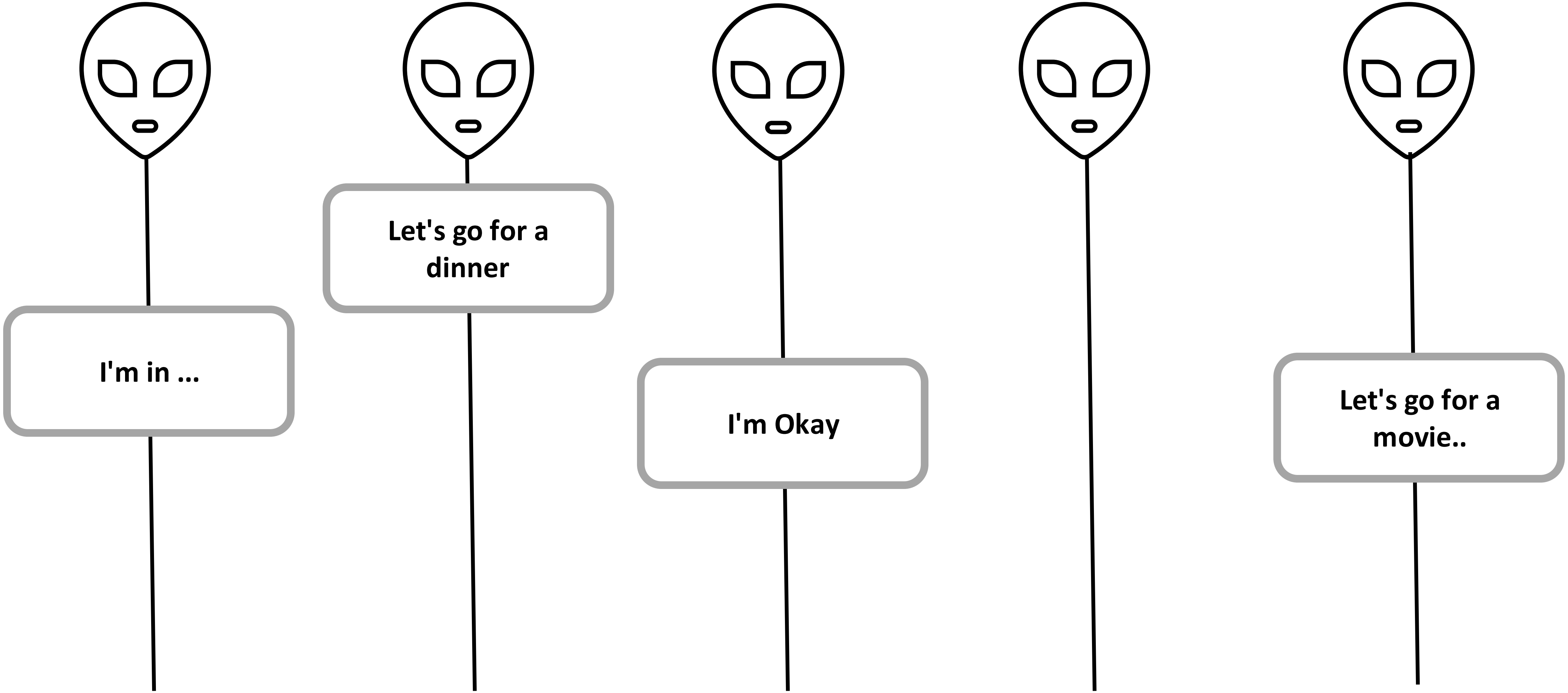
CFT Consensus in an Asynchronous System



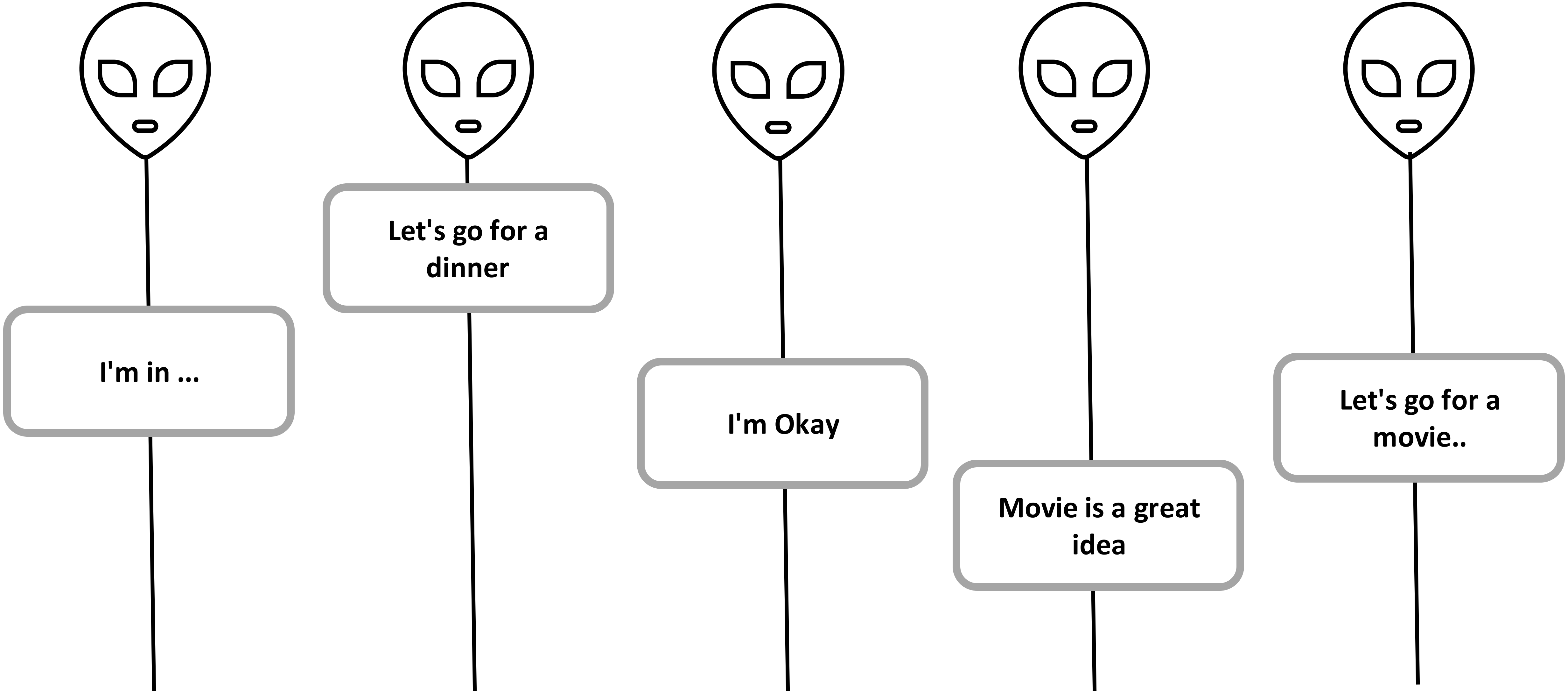
CFT Consensus in an Asynchronous System



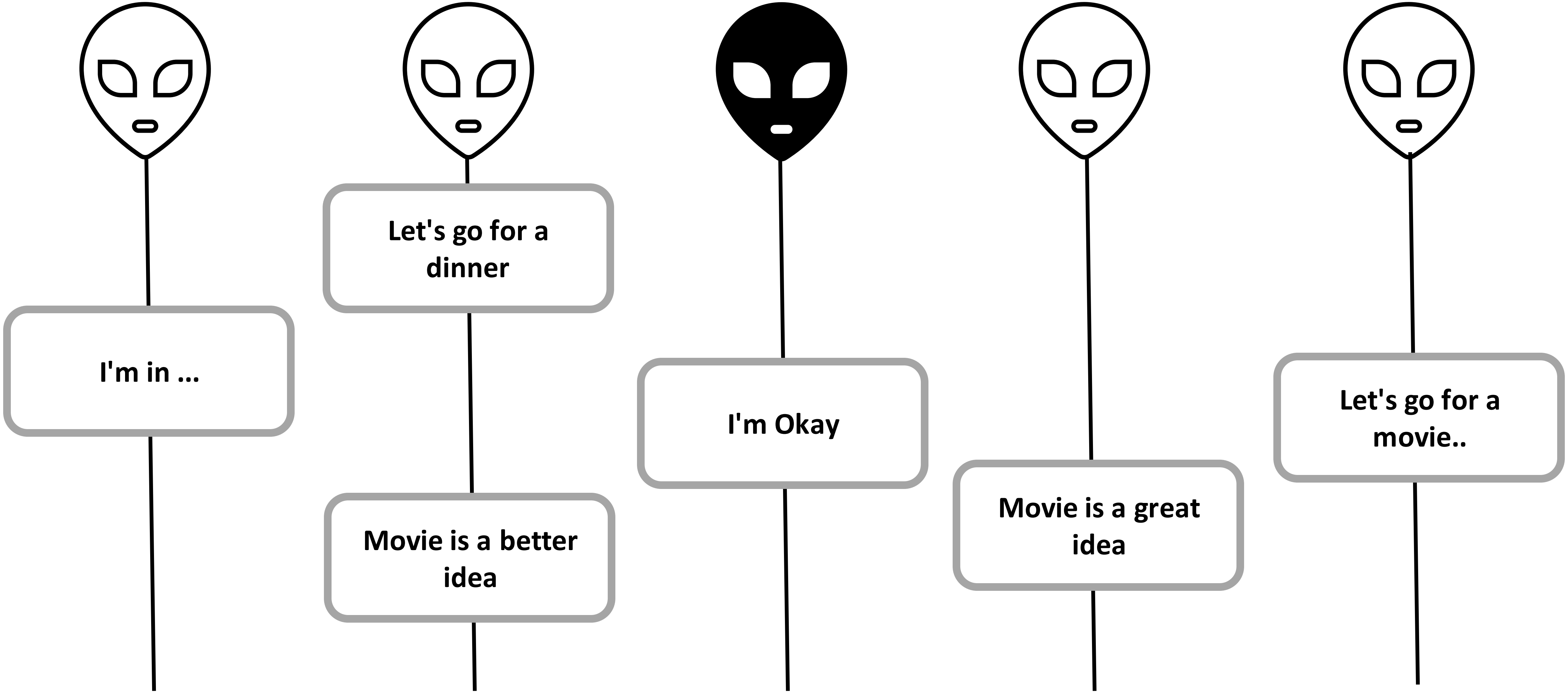
CFT Consensus in an Asynchronous System



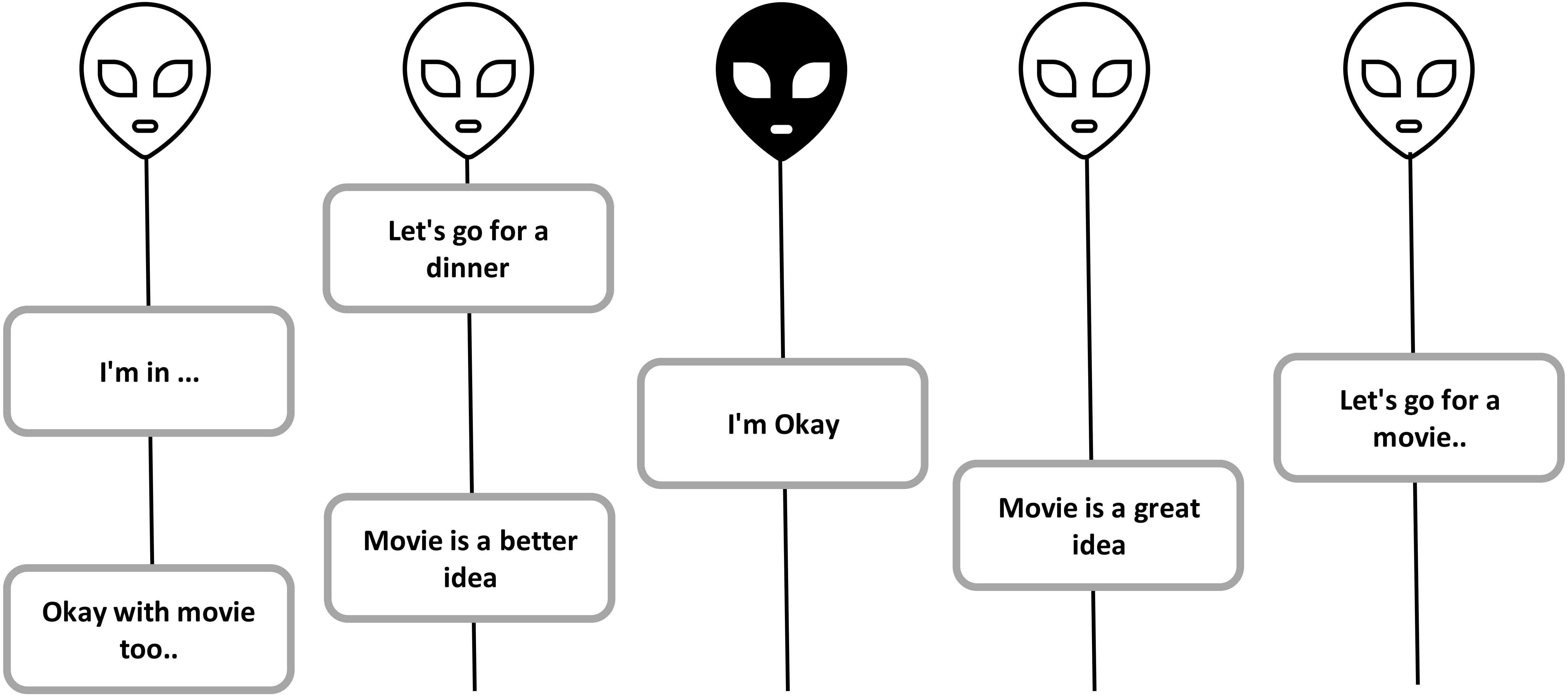
CFT Consensus in an Asynchronous System



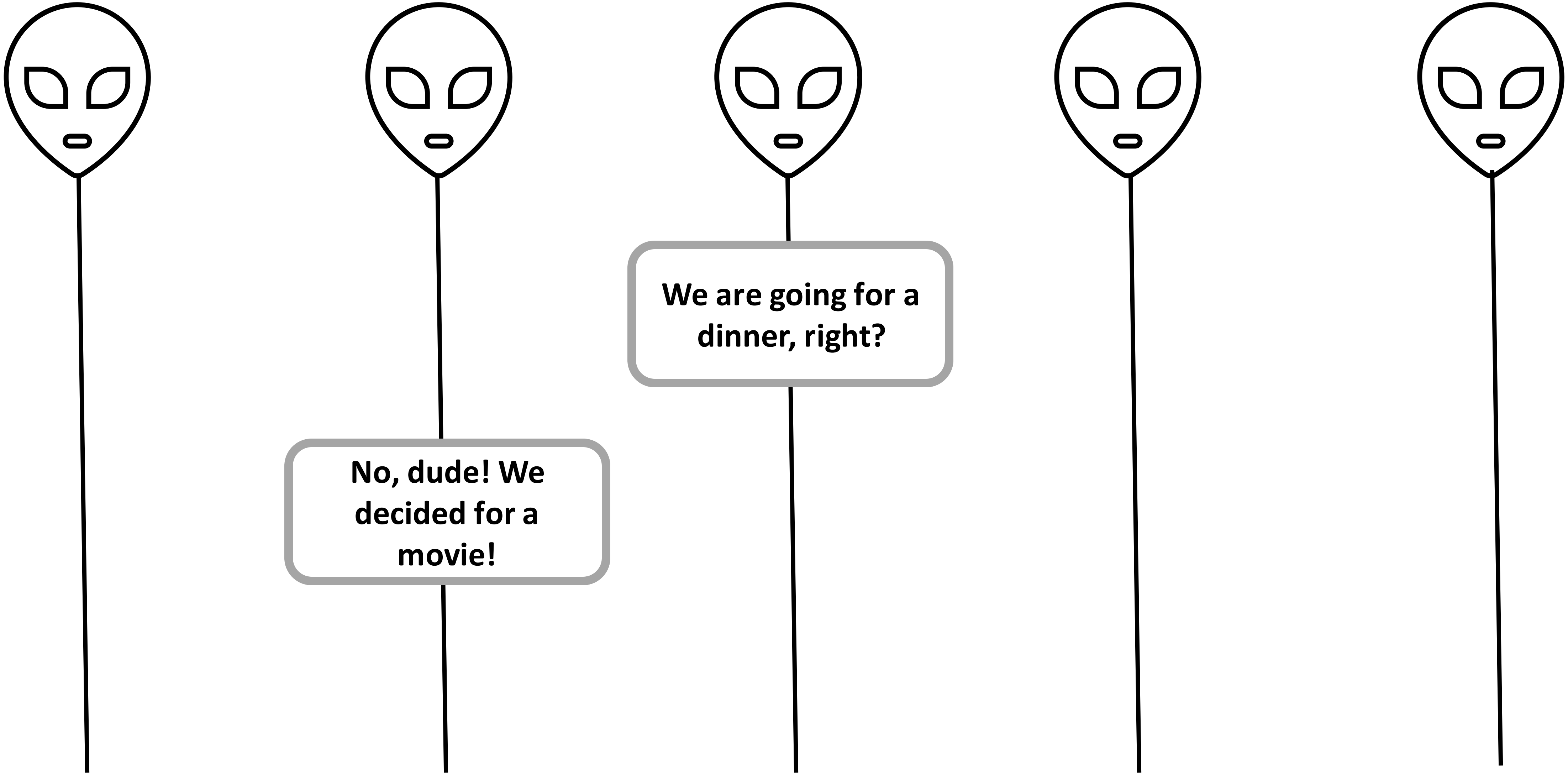
CFT Consensus in an Asynchronous System



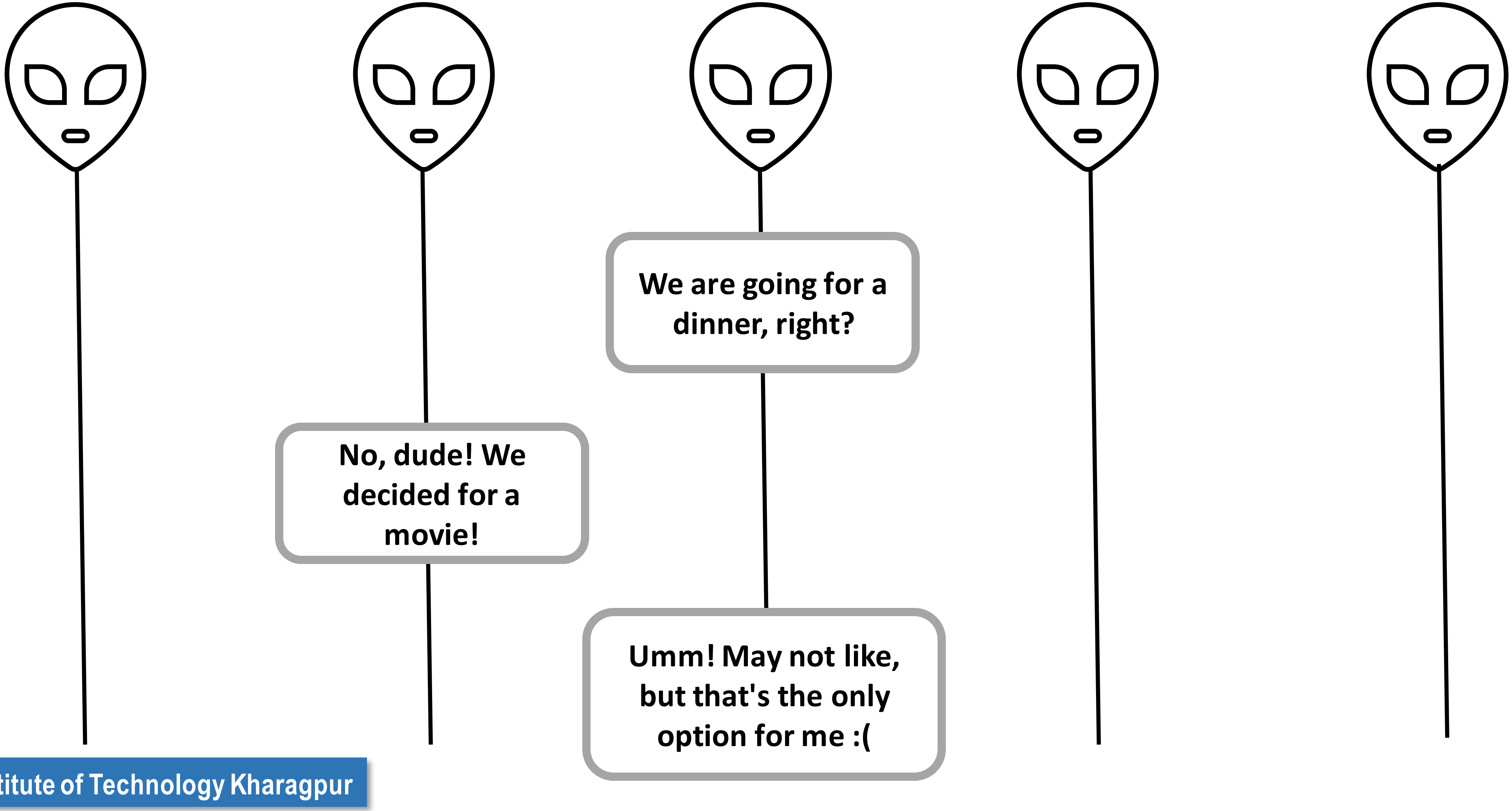
CFT Consensus in an Asynchronous System



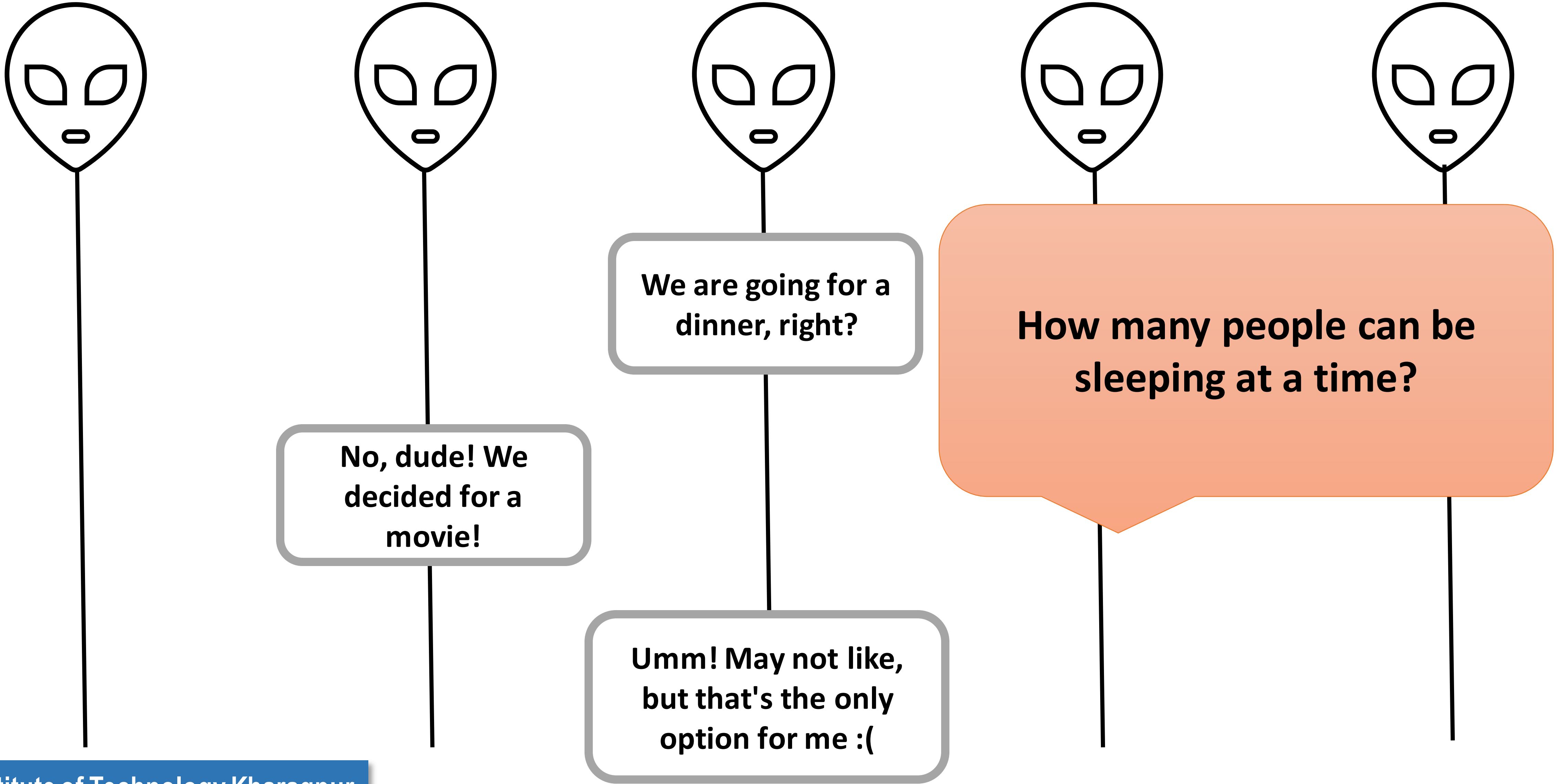
CFT Consensus in an Asynchronous System


We are going for a dinner, right?
No, dude! We decided for a movie!

CFT Consensus in an Asynchronous System


We are going for a dinner, right?
No, dude! We decided for a movie!
Umm! May not like, but that's the only option for me :(

CFT Consensus in an Asynchronous System



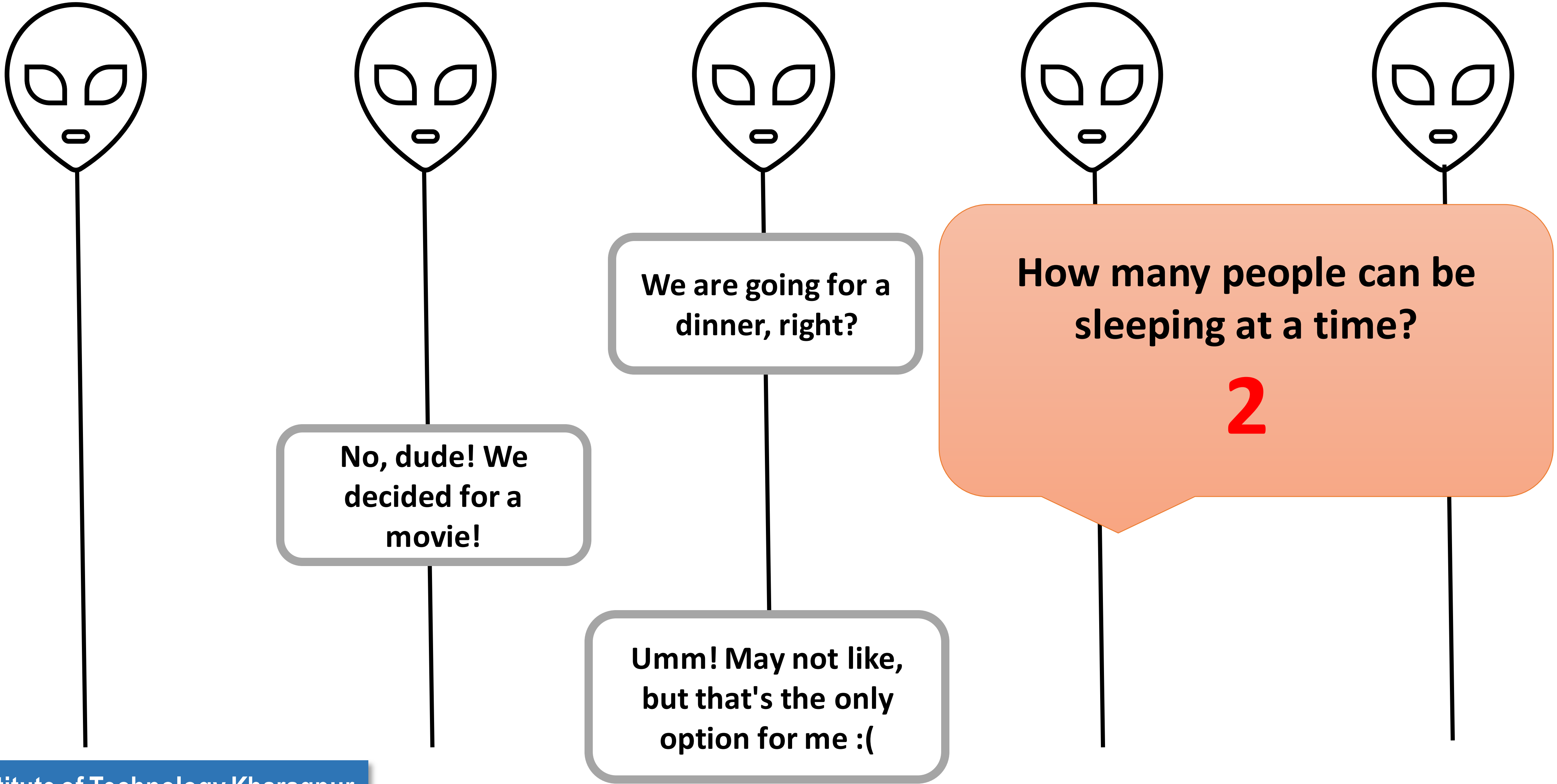
No, dude! We decided for a movie!

We are going for a dinner, right?

Umm! May not like, but that's the only option for me :(

How many people can be sleeping at a time?

CFT Consensus in an Asynchronous System



We are going for a dinner, right?

No, dude! We decided for a movie!

Umm! May not like, but that's the only option for me :(

How many people can be sleeping at a time?

2

Asynchronous CFT

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

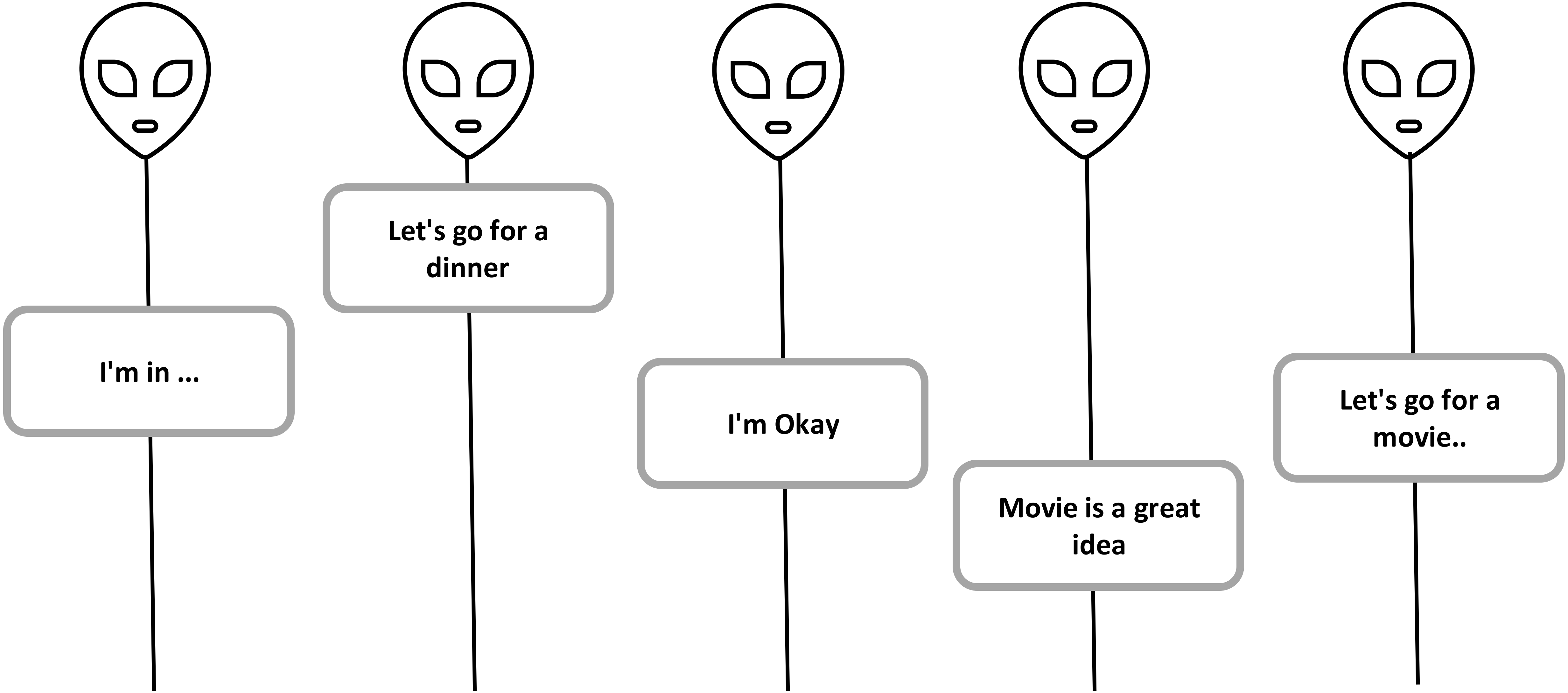
- **Paxos:** A family of distributed algorithms to reach consensus in an asynchronous CFT

Asynchronous CFT

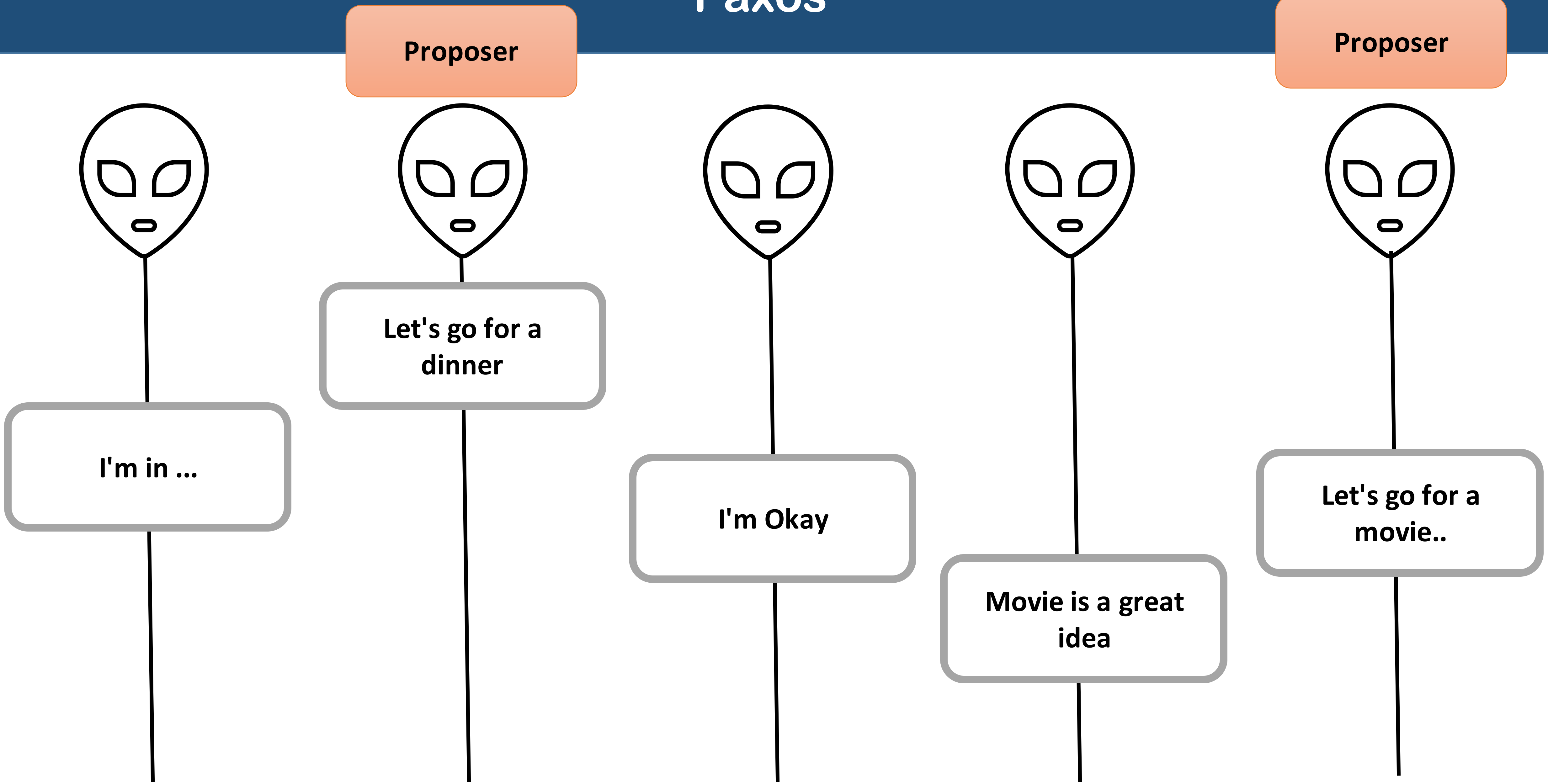
- If there are F faulty nodes (crash fault), we need at least 2F+1 nodes to reach consensus

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

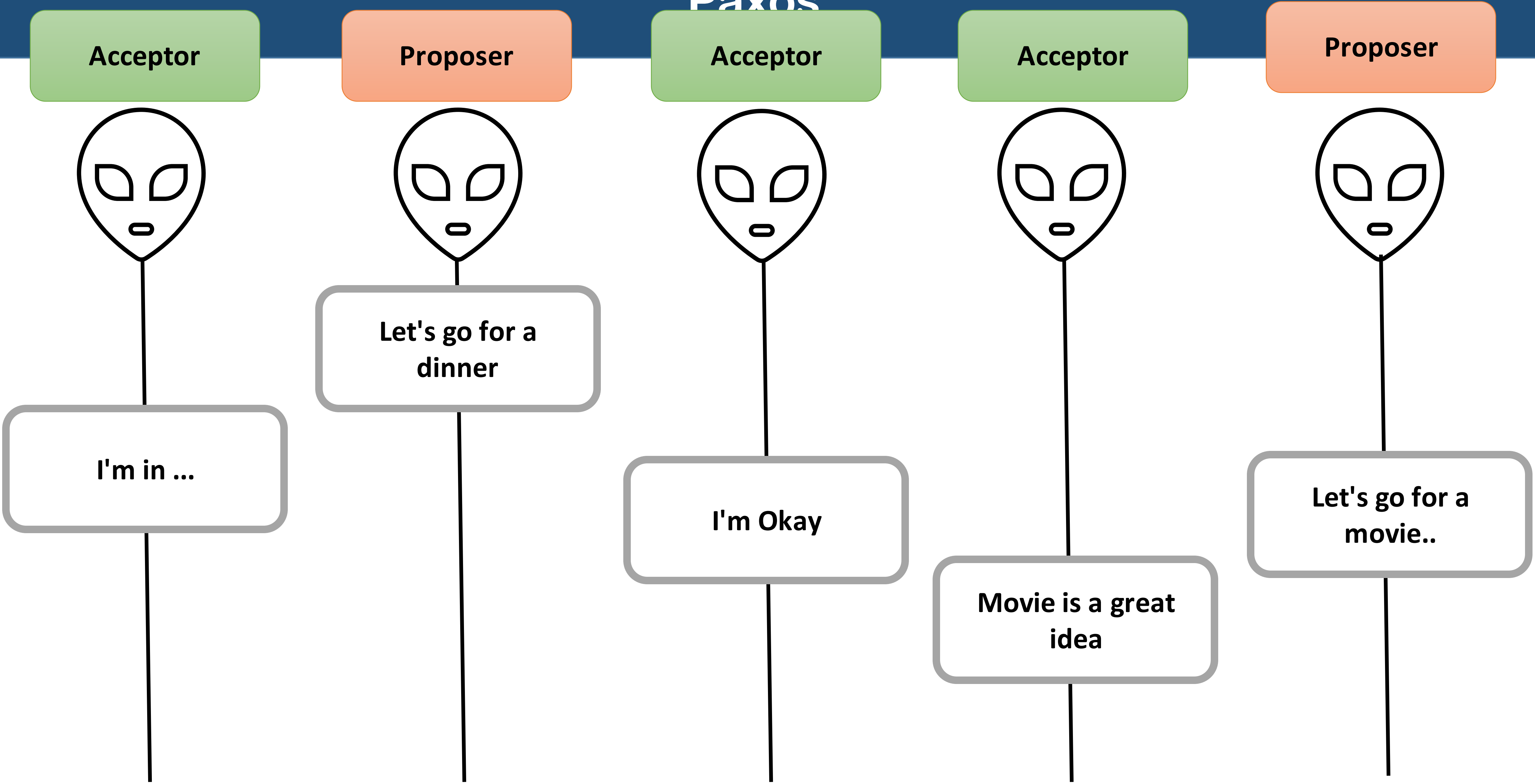
Paxos


Let's go for a dinner
I'm in ...
I'm Okay
Movie is a great idea
Let's go for a movie..

Paxos

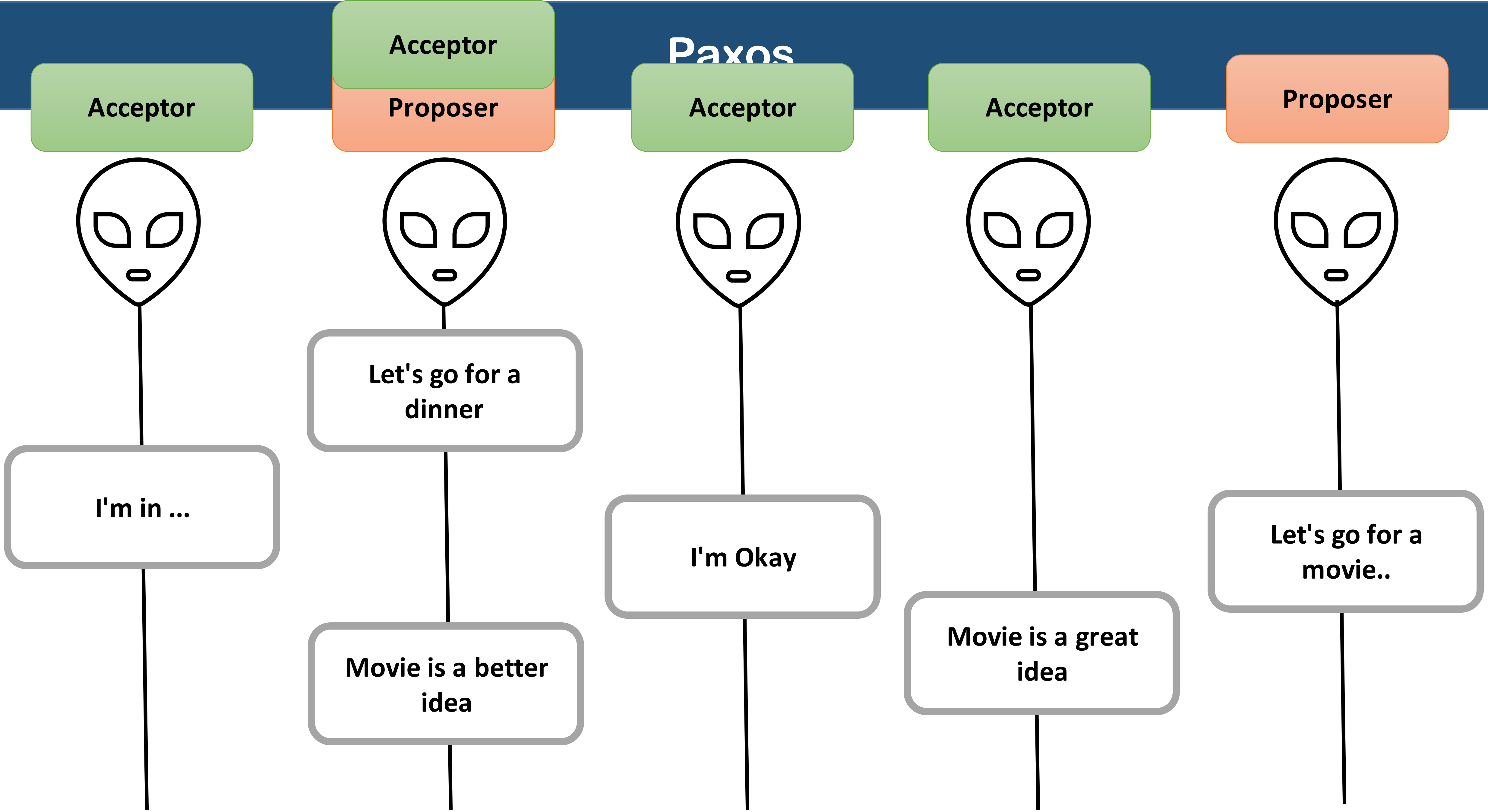

Proposer
Proposer
I'm in …
Let's go for a dinner
I'm Okay
Movie is a great idea
Let's go for a movie..

Paxos



Acceptor
Proposer
Acceptor
Acceptor
Proposer
Let's go for a dinner
I'm in ...
I'm Okay
Movie is a great idea
Let's go for a movie..

Paxos



Acceptor

I'm in ...

Acceptor
Proposer

Let's go for a dinner

Movie is a better idea

Acceptor

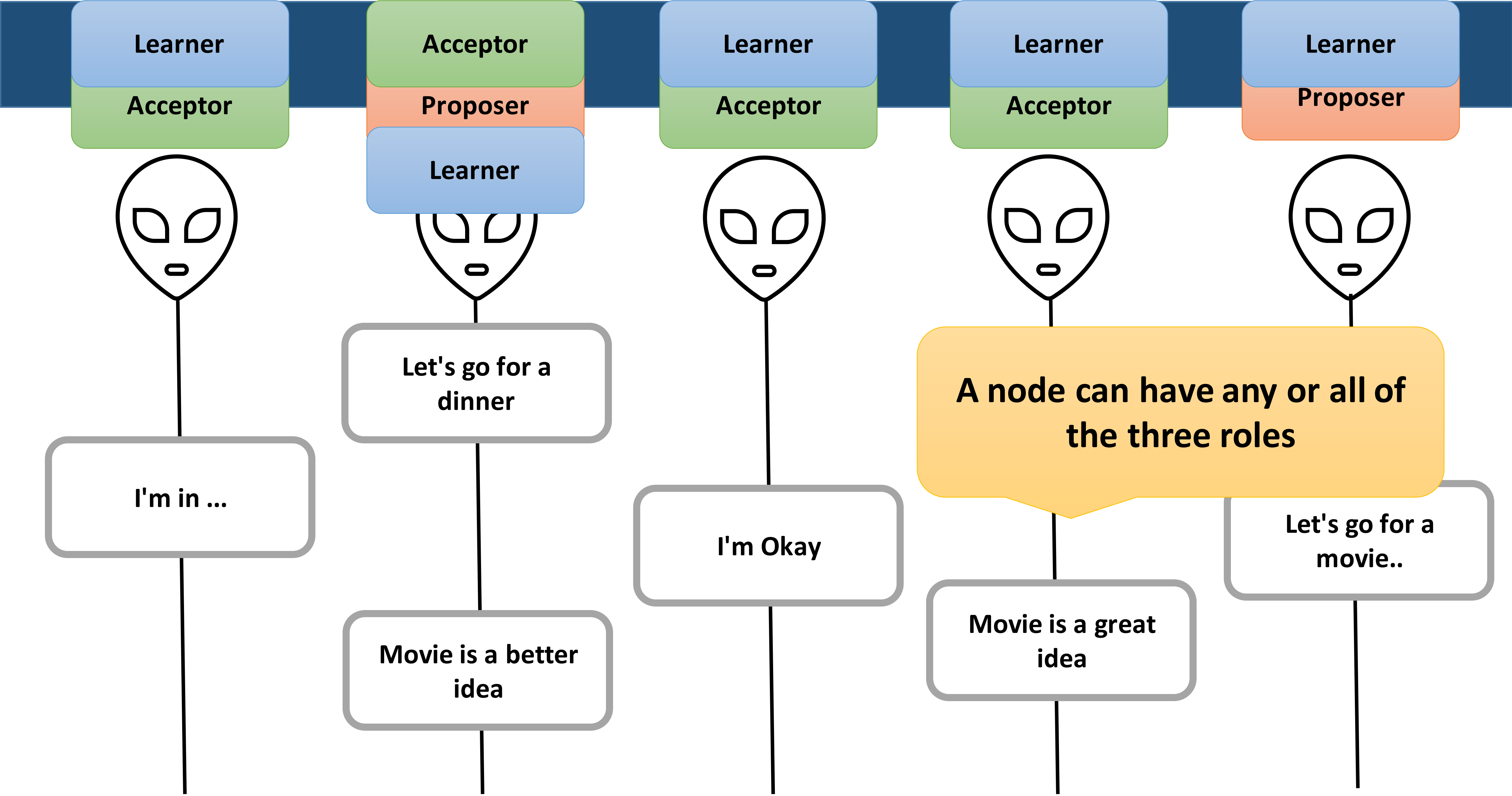
I'm Okay

Acceptor

Movie is a great idea

Proposer

Let's go for a movie..



Learner
Acceptor

I'm in …

Acceptor
Proposer
Learner

Let's go for a dinner

Movie is a better idea

Learner
Acceptor

I'm Okay

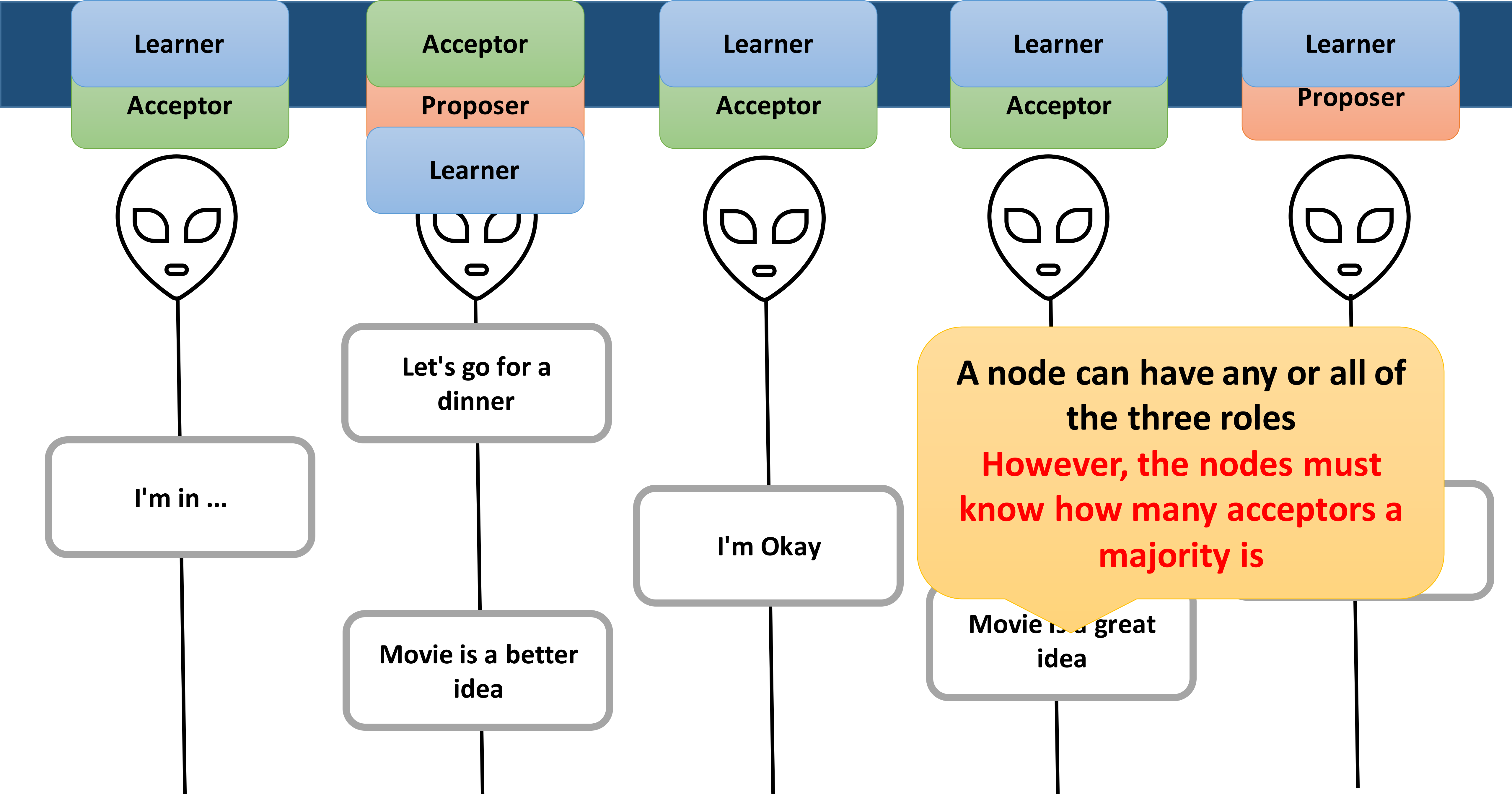
Learner
Acceptor

Movie is a great idea

Learner
Proposer

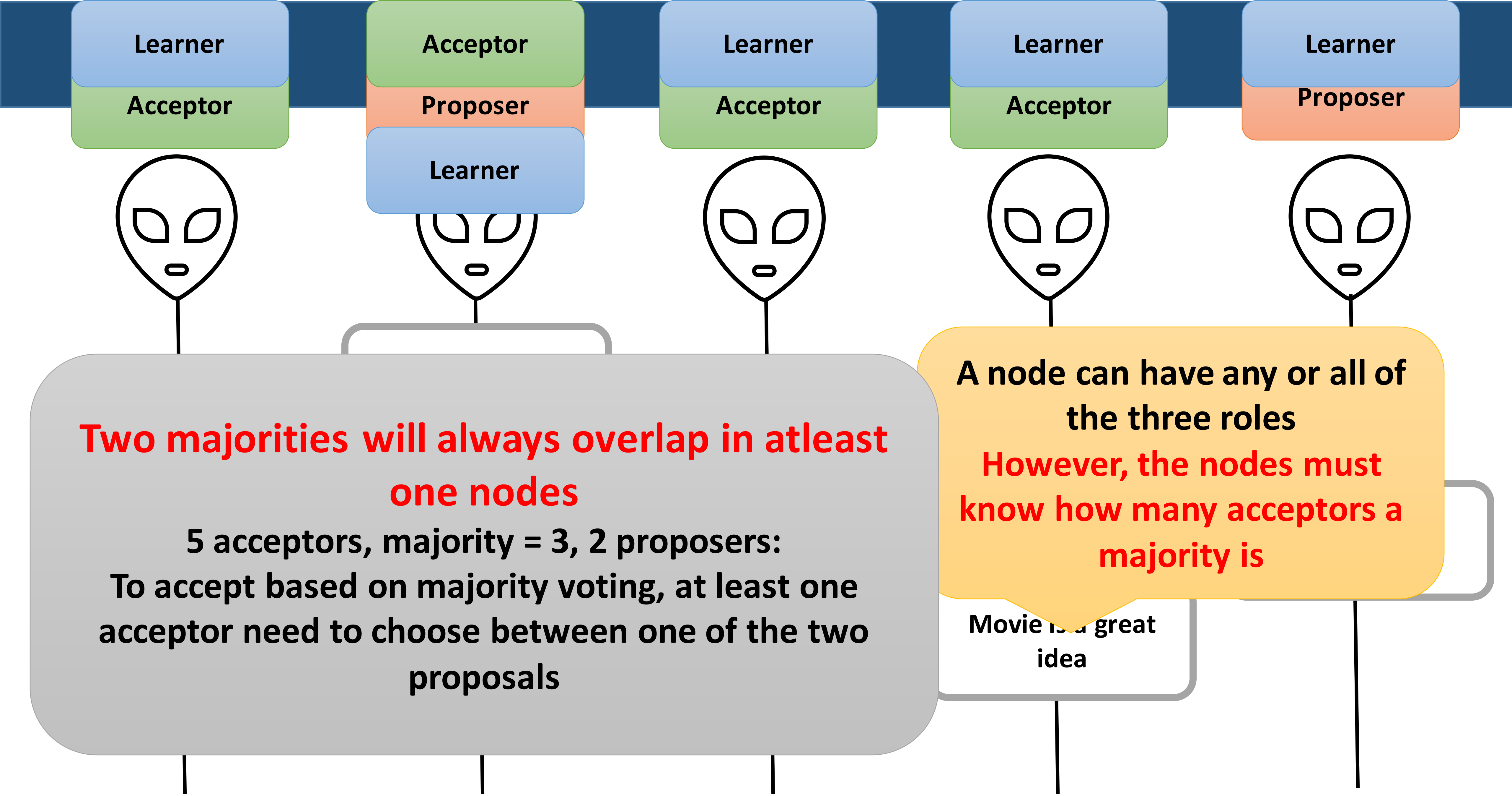
Let's go for a movie..

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

A node can have any or all of the three roles

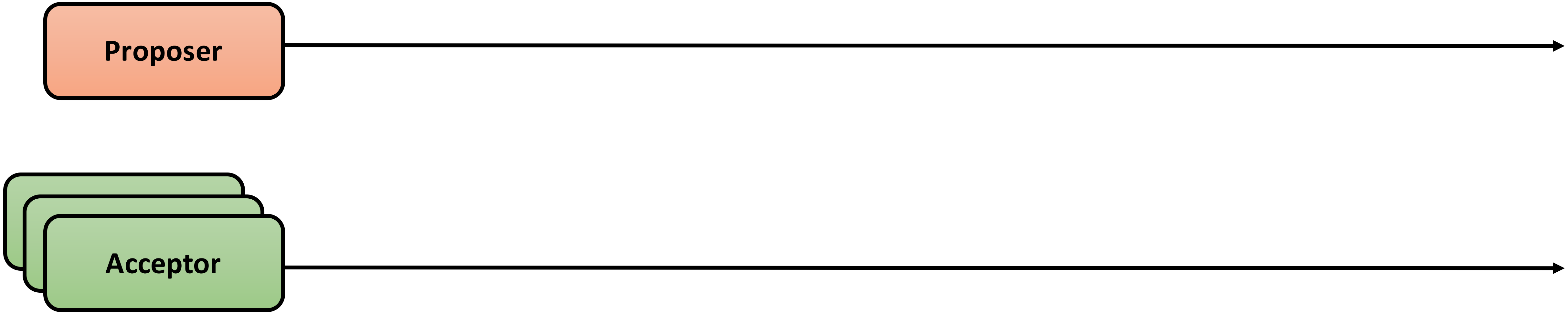
However, the nodes must know how many acceptors a majority is

Paxos Basics

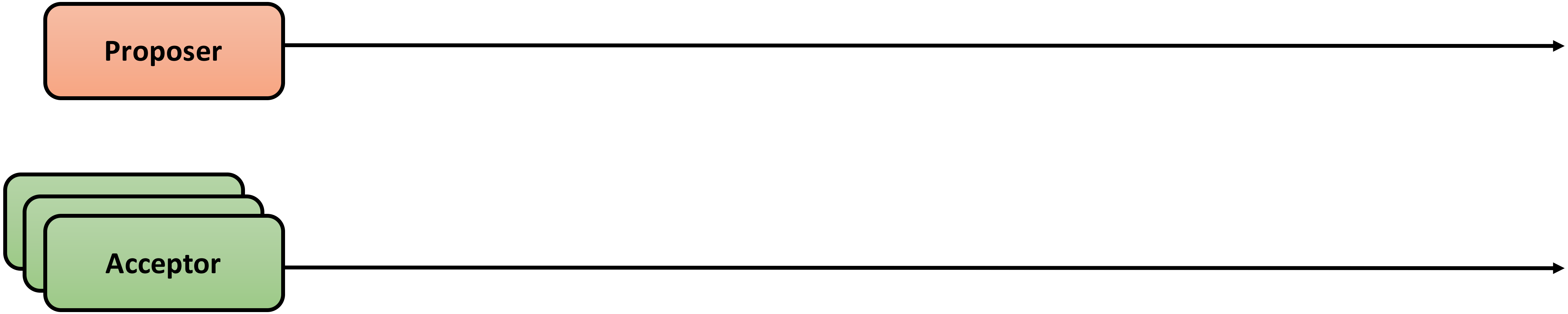
- Paxos is based on state-machine replication
  - Proposers and Acceptors maintain a state of the running epochs
  - Uses a variable IDp where p is an epoch number – maintains the state
  - **We'll see the concept of state-machine replication later in details**

- 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



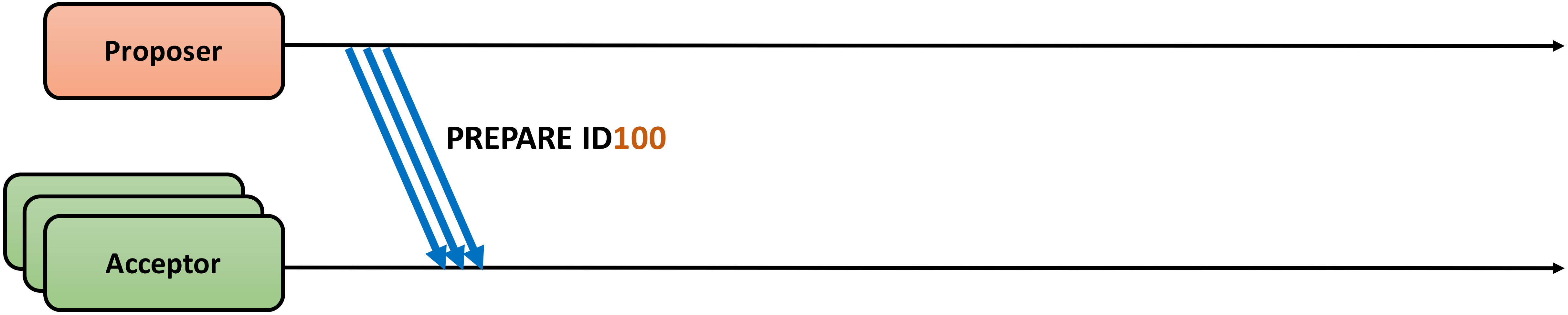
Paxos Algorithm



Proposer wants to propose its choice (values):

- Sends PREPARE IDp to a majority (or all) of the **acceptors**

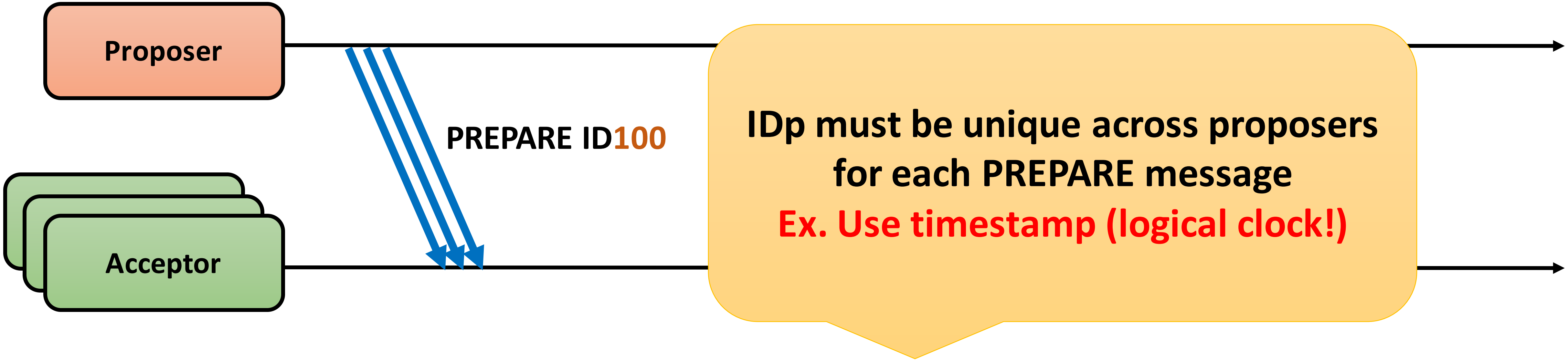
Paxos Algorithm



Proposer wants to propose its choice (values):

- Sends PREPARE IDp to a majority (or all) of the **acceptors**

Paxos Algorithm



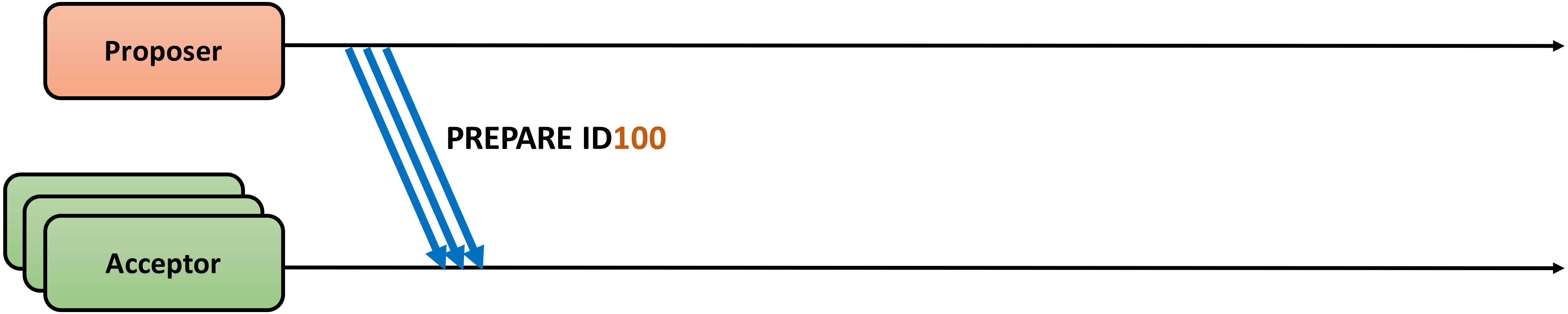
IDp must be unique across proposers for each PREPARE message

Ex. Use timestamp (logical clock!)

Proposer wants to propose its choice (values):

- Sends PREPARE IDp to a majority (or all) of the **acceptors**

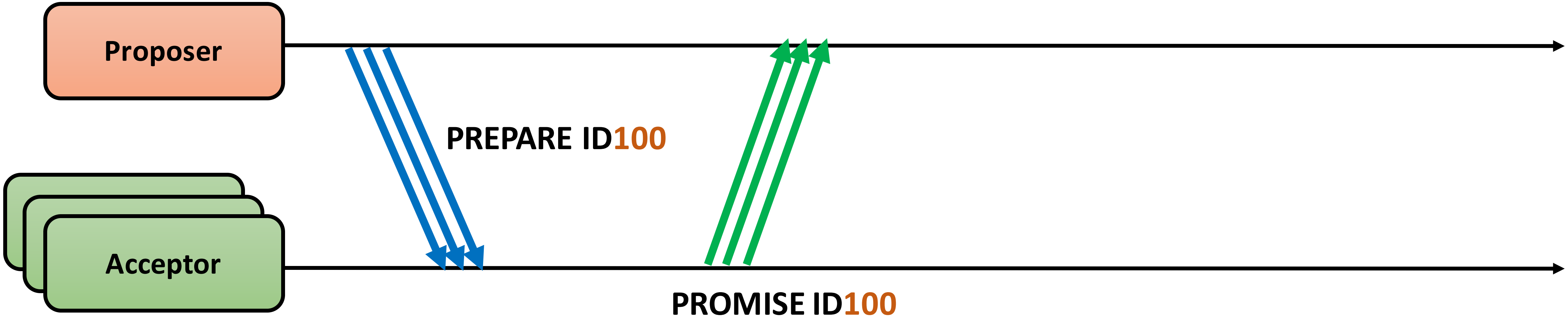
Paxos Algorithm



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

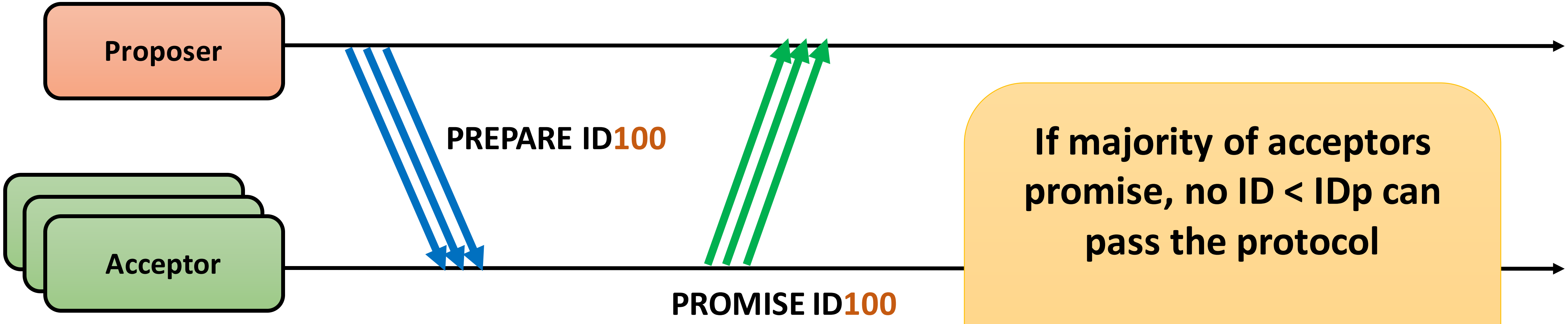
Paxos Algorithm



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



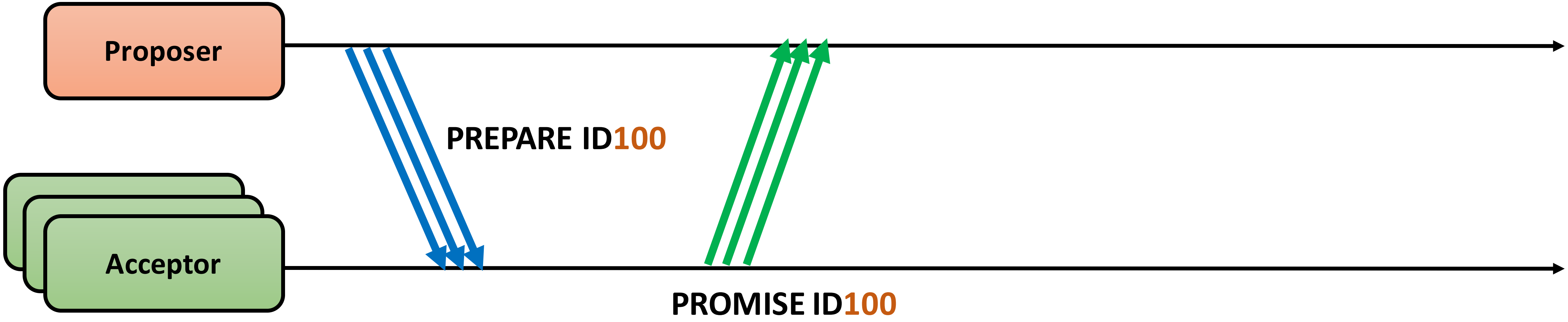
If majority of acceptors promise, no ID < IDp can pass the protocol

The acceptors agree on State 100

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

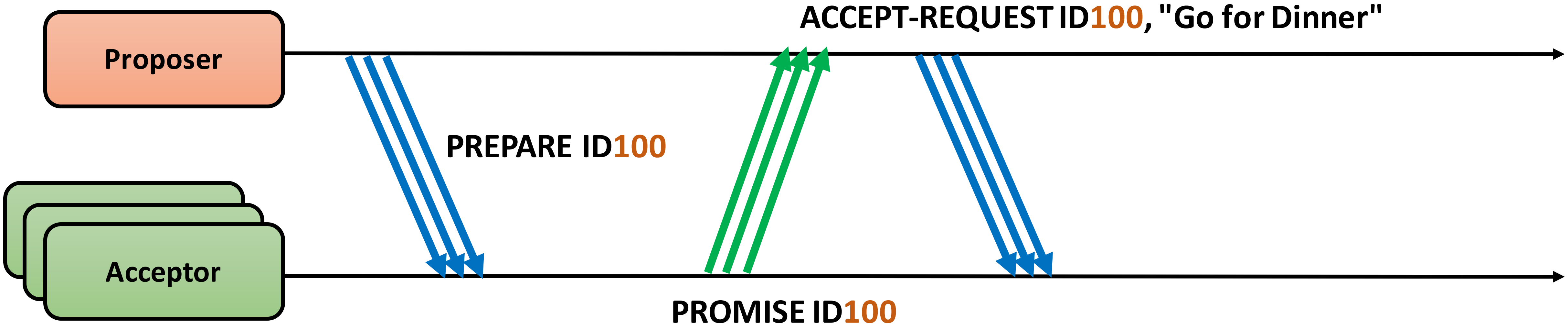
Paxos Algorithm



Proposer gets majority of PROMISE messages for a specific IDp:

- Sends **ACCEPT-REQUEST IDp**, **VALUE** to a majority (or all) of **Acceptors**
  - (?) It picks any value of its choice

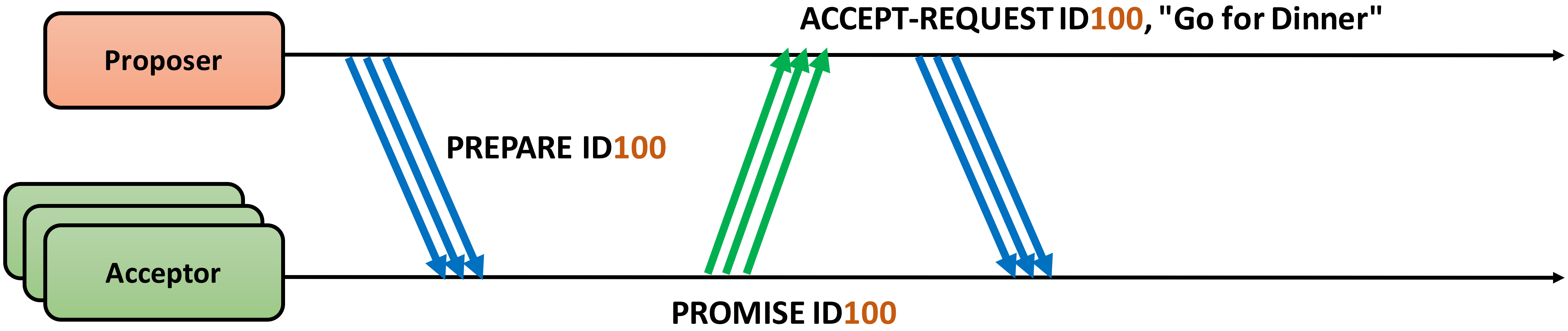
Paxos Algorithm



Proposer gets majority of PROMISE messages for a specific IDp:

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

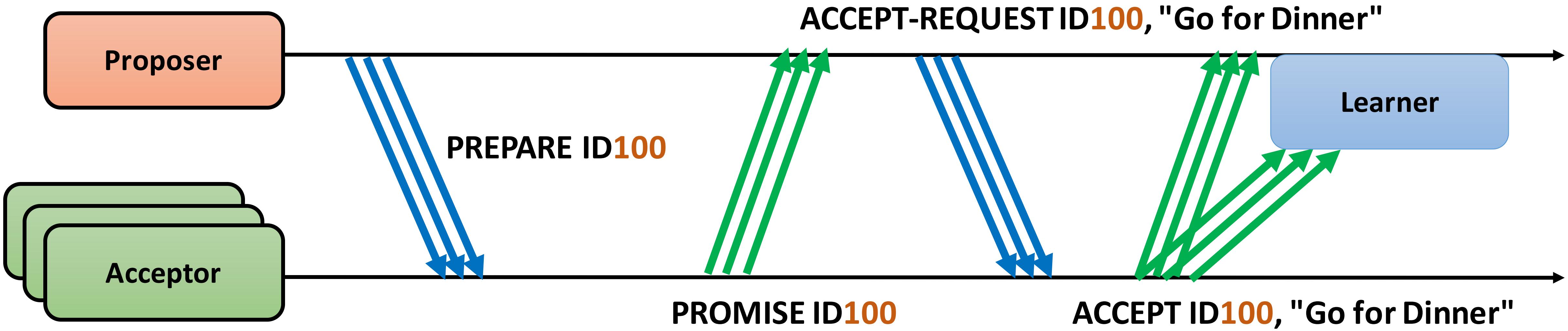
Paxos Algorithm



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

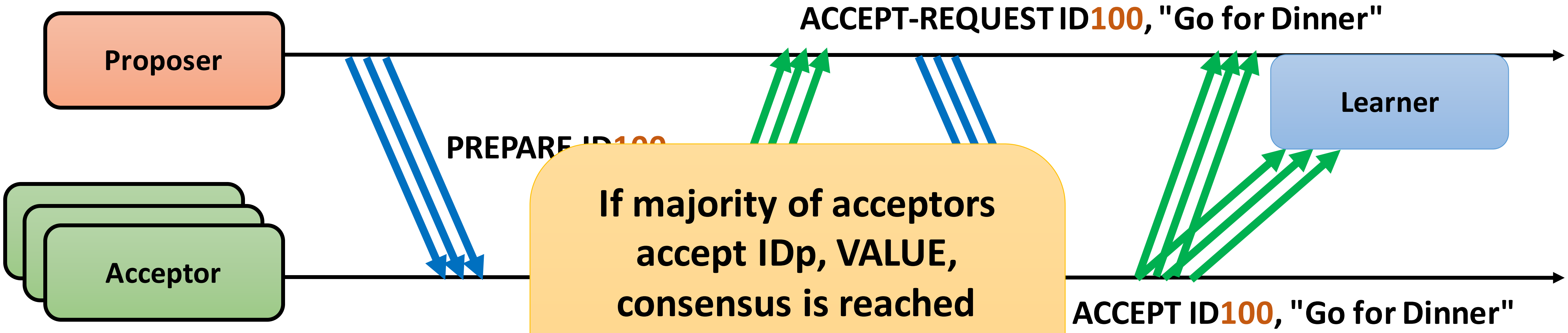
Paxos Algorithm


Proposer
Acceptor
Learner
ACCEPT-REQUEST ID100, "Go for Dinner"
PREPARE ID100
PROMISE ID100
ACCEPT ID100, "Go for Dinner"

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


ACCEPT-REQUEST ID100, "Go for Dinner"
Proposer
Learner
PREPARE ID100
Acceptor
ACCEPT ID100, "Go for Dinner"

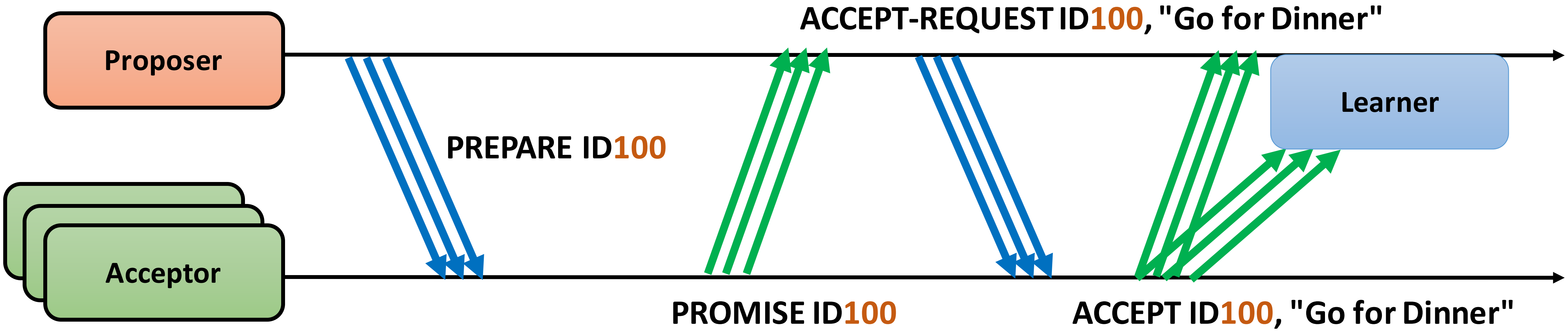
If majority of acceptors accept IDp, VALUE, consensus is reached

Consensus is reached on the value, not on IDp

Acceptor receives an ACCE

- Did it promised to ig
  - **YES:** Ignore
  - **NO:** Reply with **ACCEPT IDp, VALUE**; Also send it to all learners

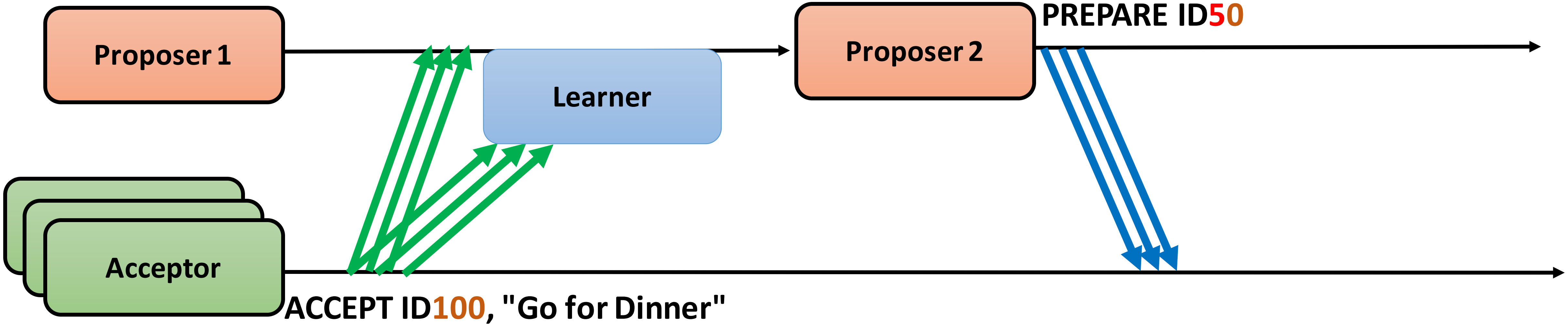
Paxos Algorithm


Proposer
Acceptor
Learner
PREPARE ID100
PROMISE ID100
ACCEPT-REQUEST ID100, "Go for Dinner"
ACCEPT ID100, "Go for Dinner"

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 Algorithm – Multiple Proposers



Proposer 1

Learner

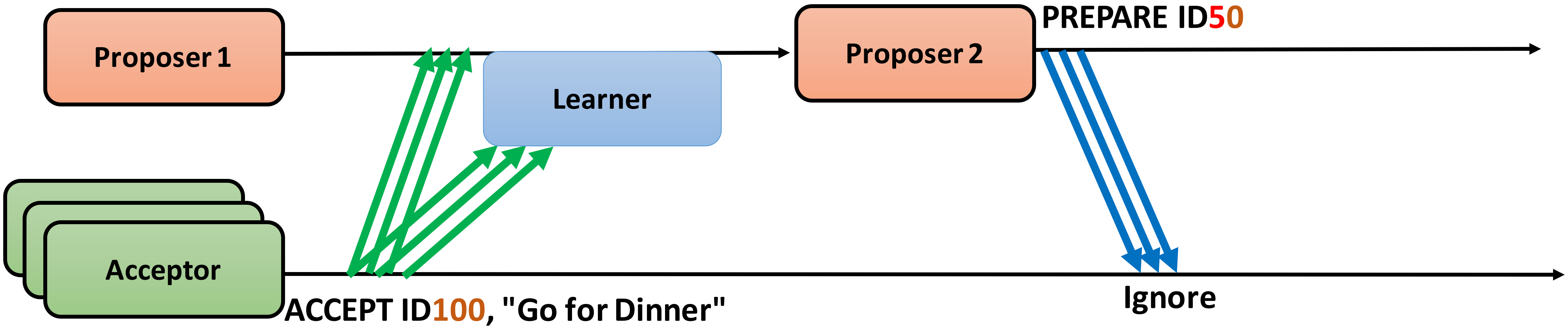
Proposer 2

PREPARE ID50

Acceptor

ACCEPT ID100, "Go for Dinner"

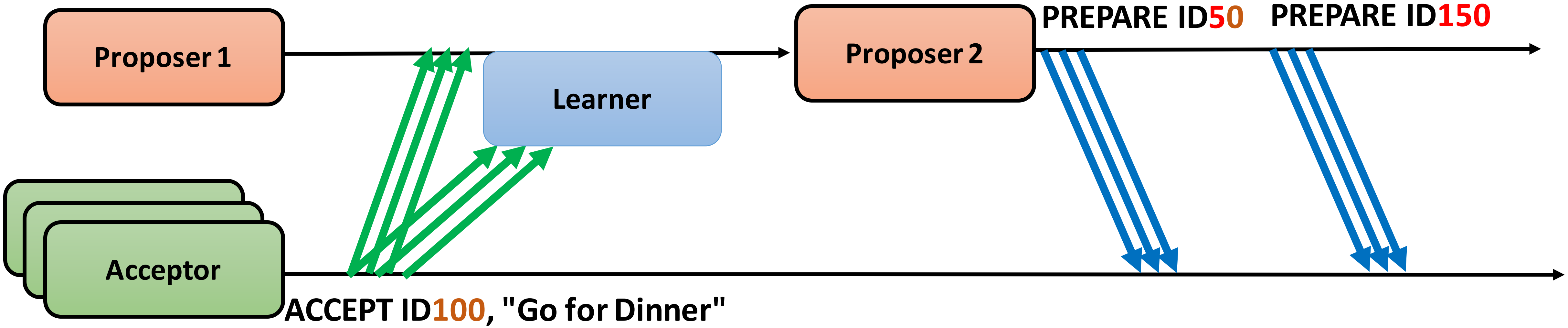
Paxos Algorithm – Multiple Proposers



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
    - (?) Reply with PROMISE IDp

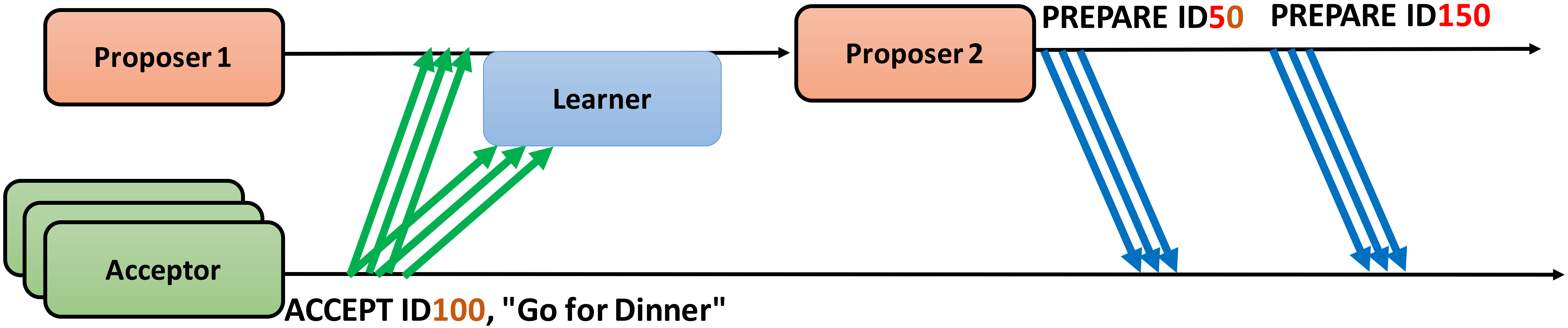
Paxos Algorithm – Multiple Proposers



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
    - (?) Reply with PROMISE IDp

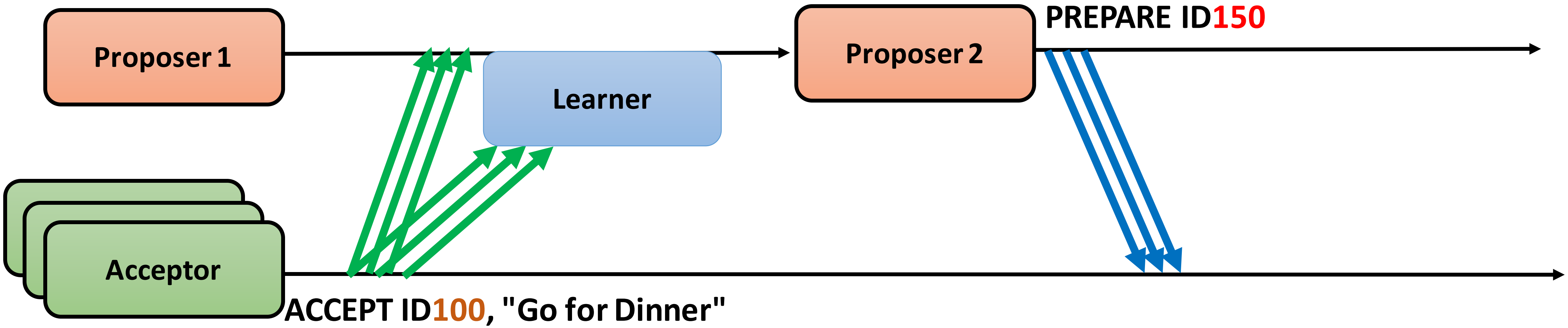
Paxos Algorithm – Multiple Proposers



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**

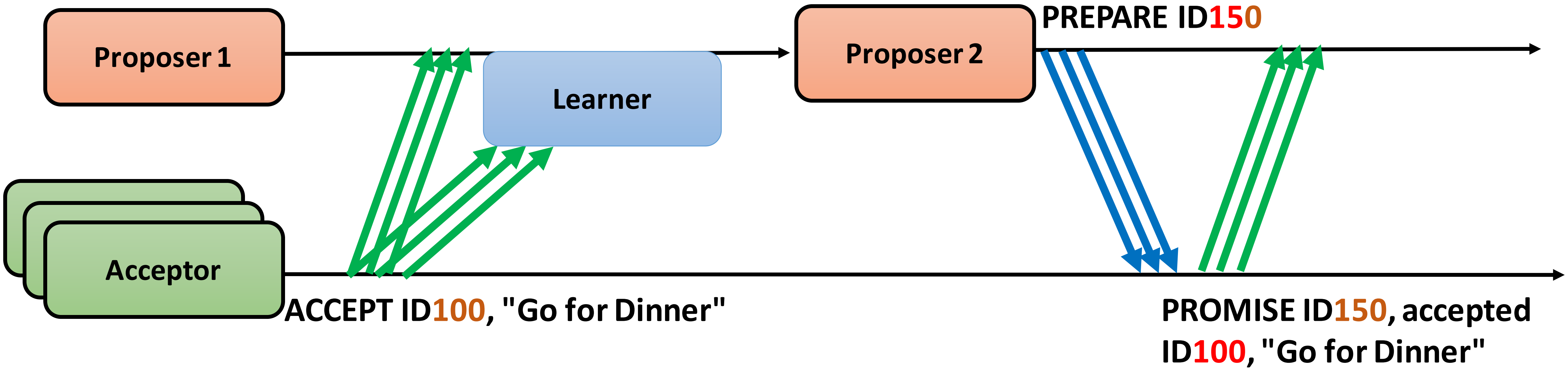
Paxos Algorithm – Multiple Proposers



Acceptor received a PREPARE message with IDp:

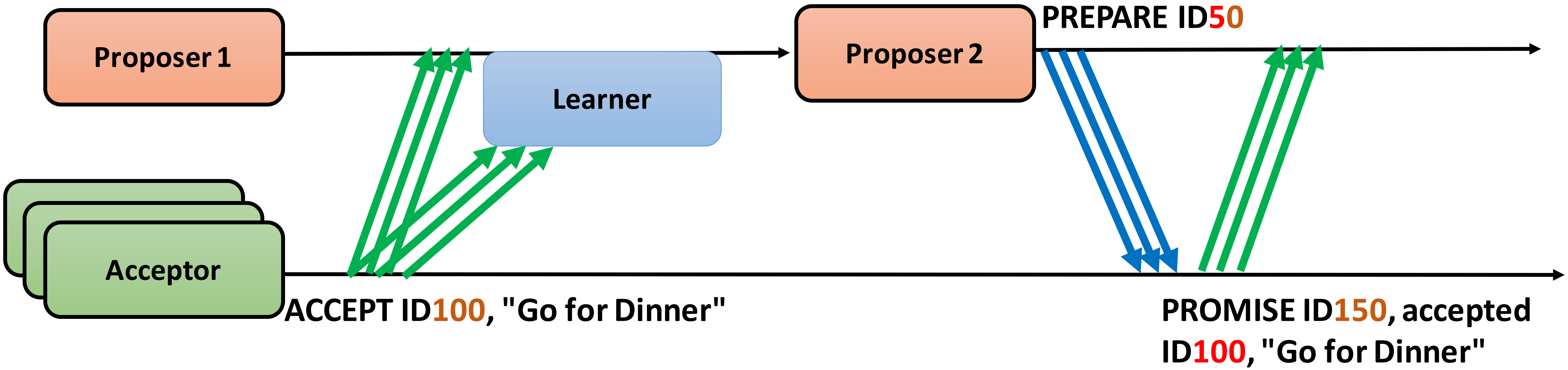
- Did it promised to ignore requests with this IDp?
  - **YES:** Ignore
  - **NO:** Will promise to ignore any request lower than IDp
    - Has it ever accepted anything? (Assume accepted ID = IDa)
      - **YES:** Reply with **PROMISE IDp accepted IDa, VALUE**
      - **NO:** Reply with **PROMISE IDp**

Paxos Algorithm – Multiple Proposers


Proposer 1
Learner
Proposer 2
PREPARE ID150
Acceptor
ACCEPT ID100, "Go for Dinner"
PROMISE ID150, accepted ID100, "Go for Dinner"

What the proposer will do?

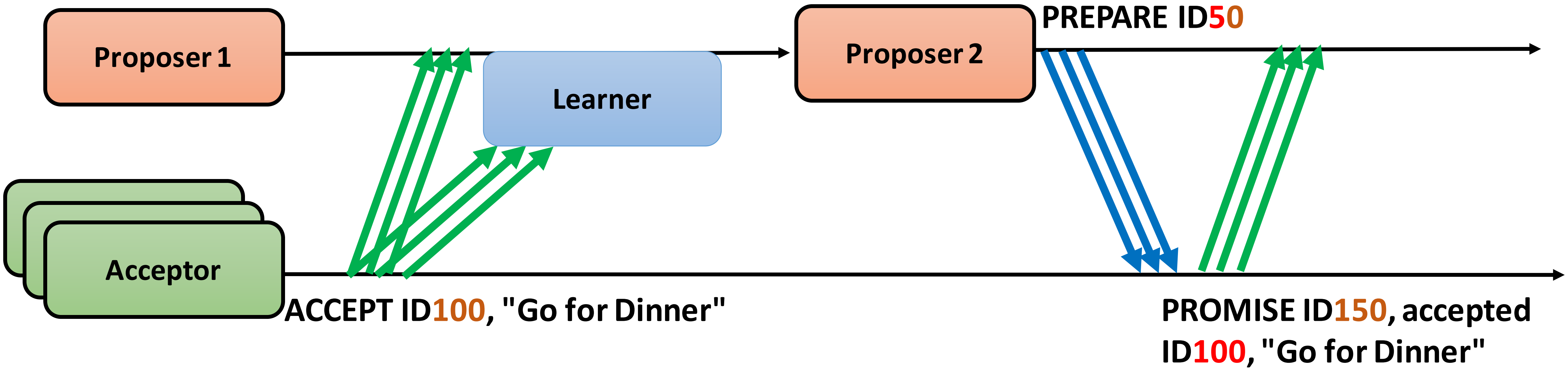
Paxos Algorithm – Multiple Proposers


Proposer 1
Learner
Proposer 2
PREPARE ID50
Acceptor
ACCEPT ID100, "Go for Dinner"
PROMISE ID150, accepted ID100, "Go for Dinner"

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 Algorithm – Multiple Proposers



Proposer 1

Learner

Proposer 2

PREPARE ID50

Acceptor

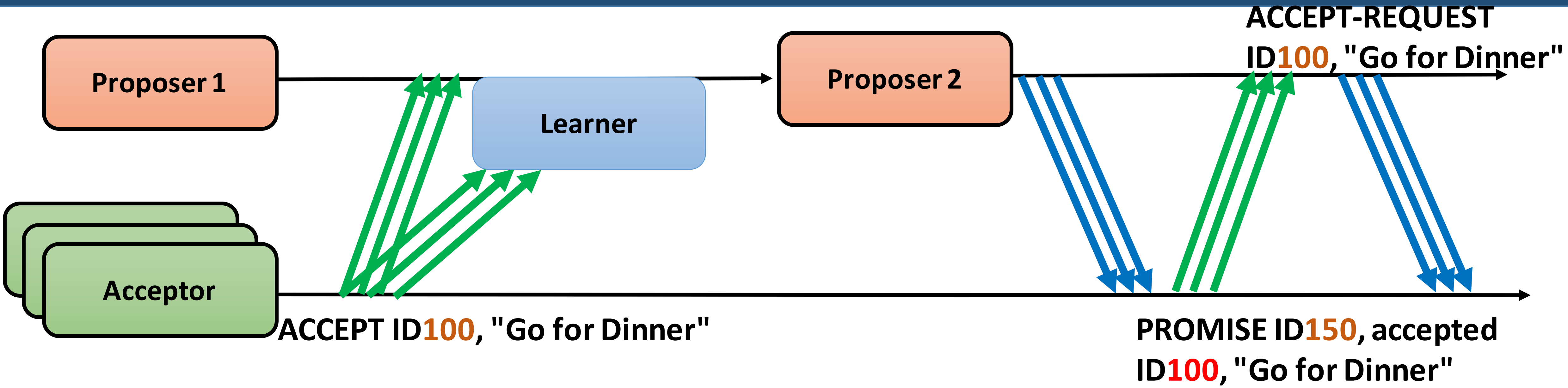
ACCEPT ID100, "Go for Dinner"

PROMISE ID150, accepted ID100, "Go for Dinner"

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

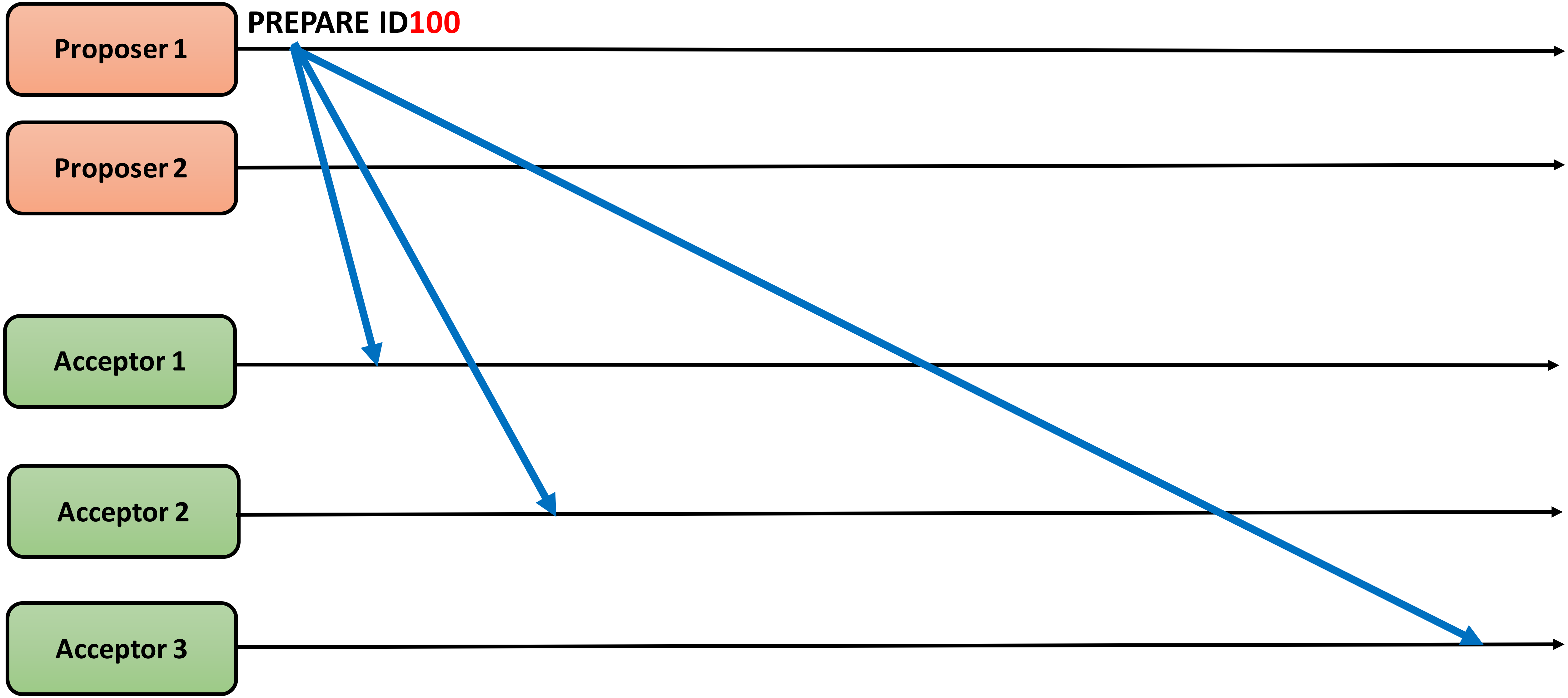
Paxos Algorithm – Multiple Proposers



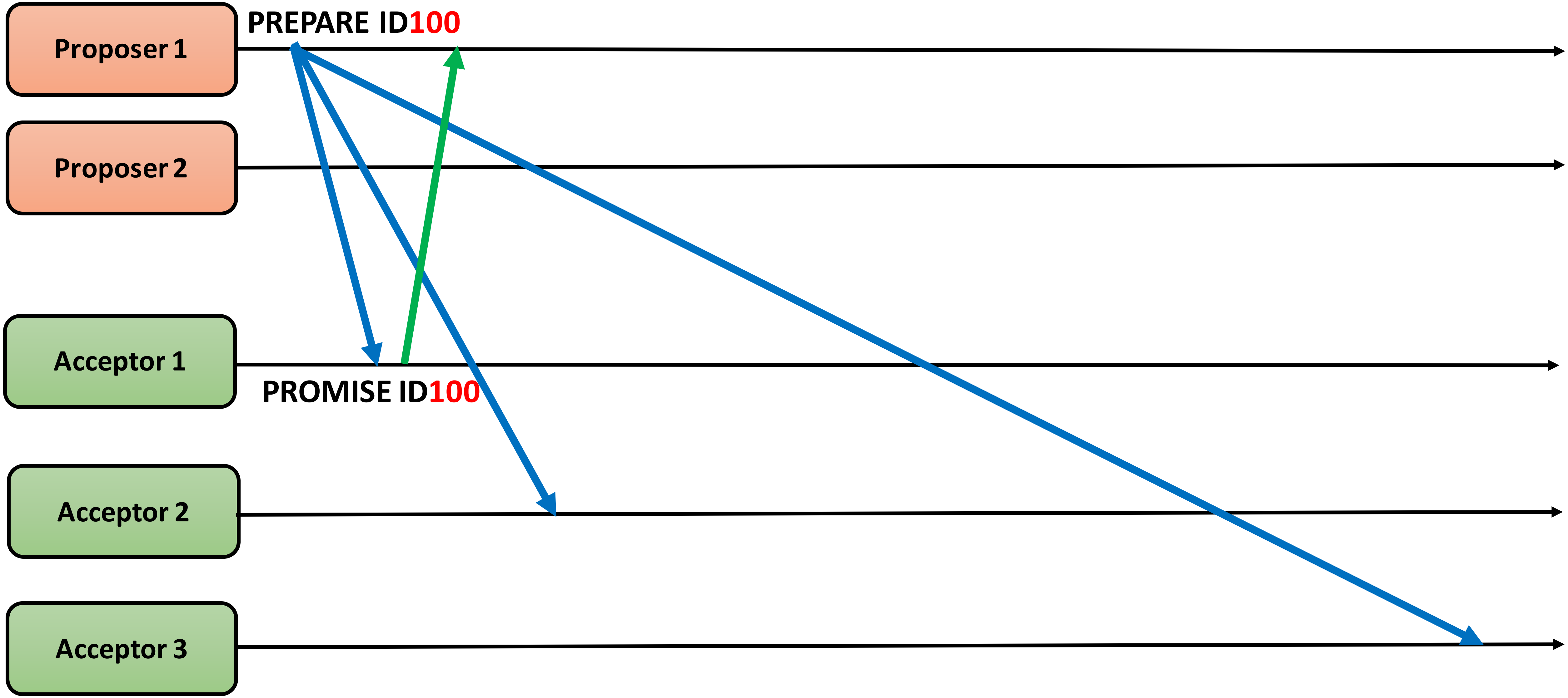
Proposer gets majority of PROMISE messages for a specific IDp:

- 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

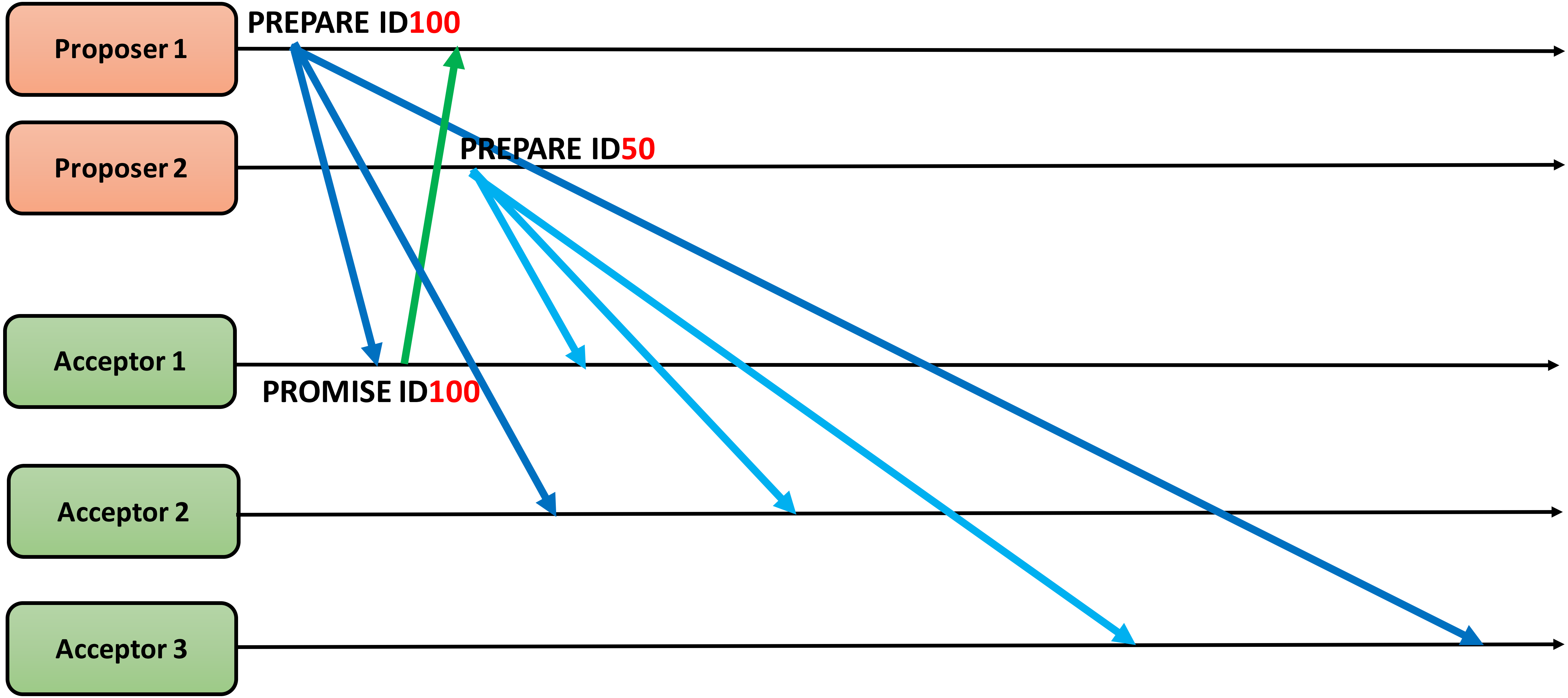
Paxos – How Majority Works



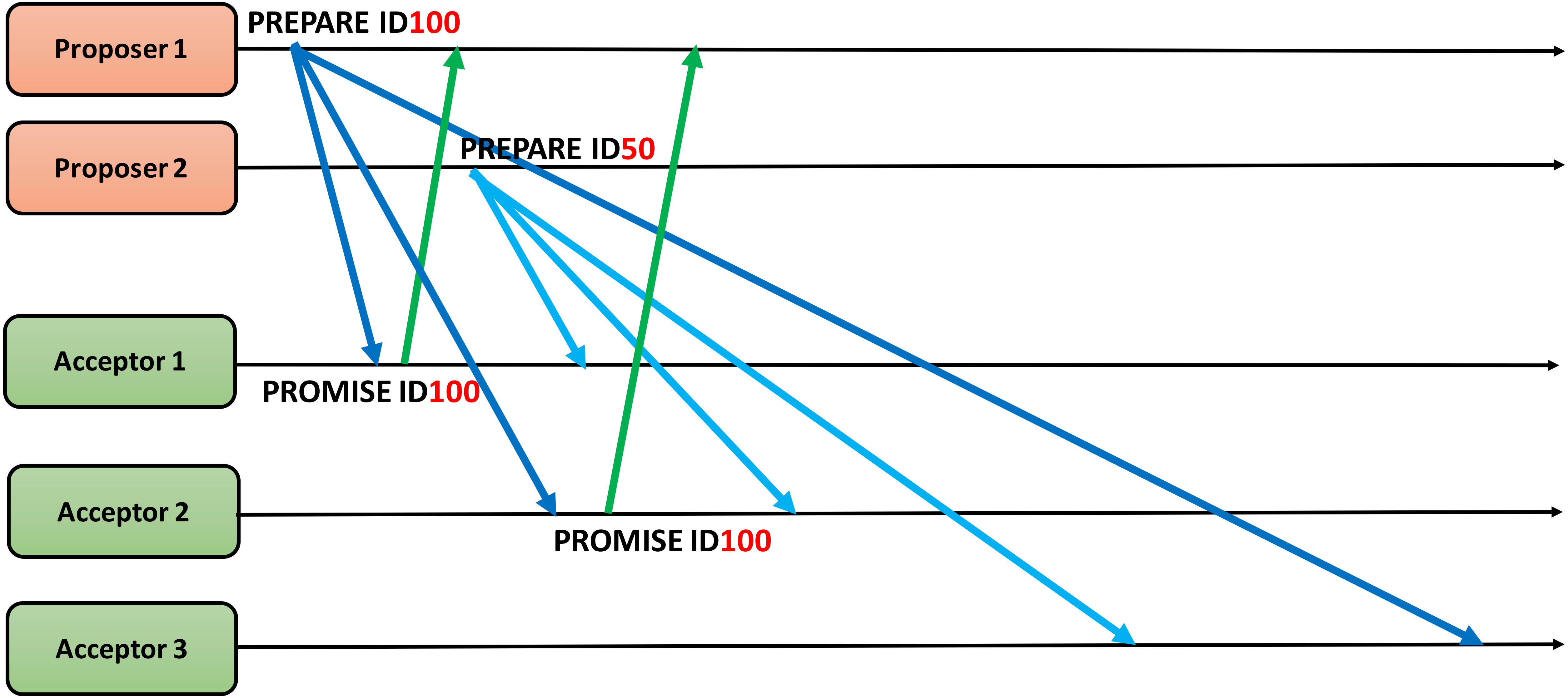
Paxos – How Majority Works



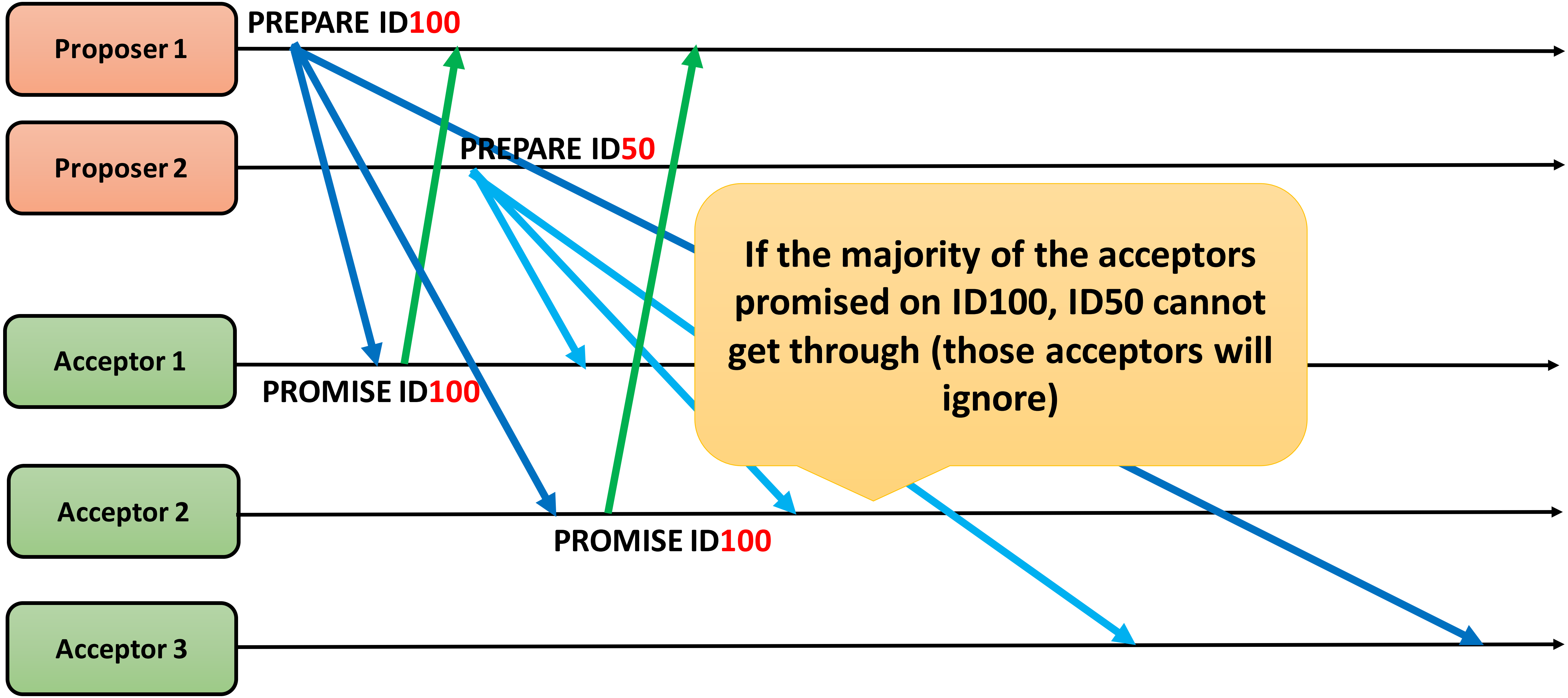
Paxos – How Majority Works



Paxos – How Majority Works

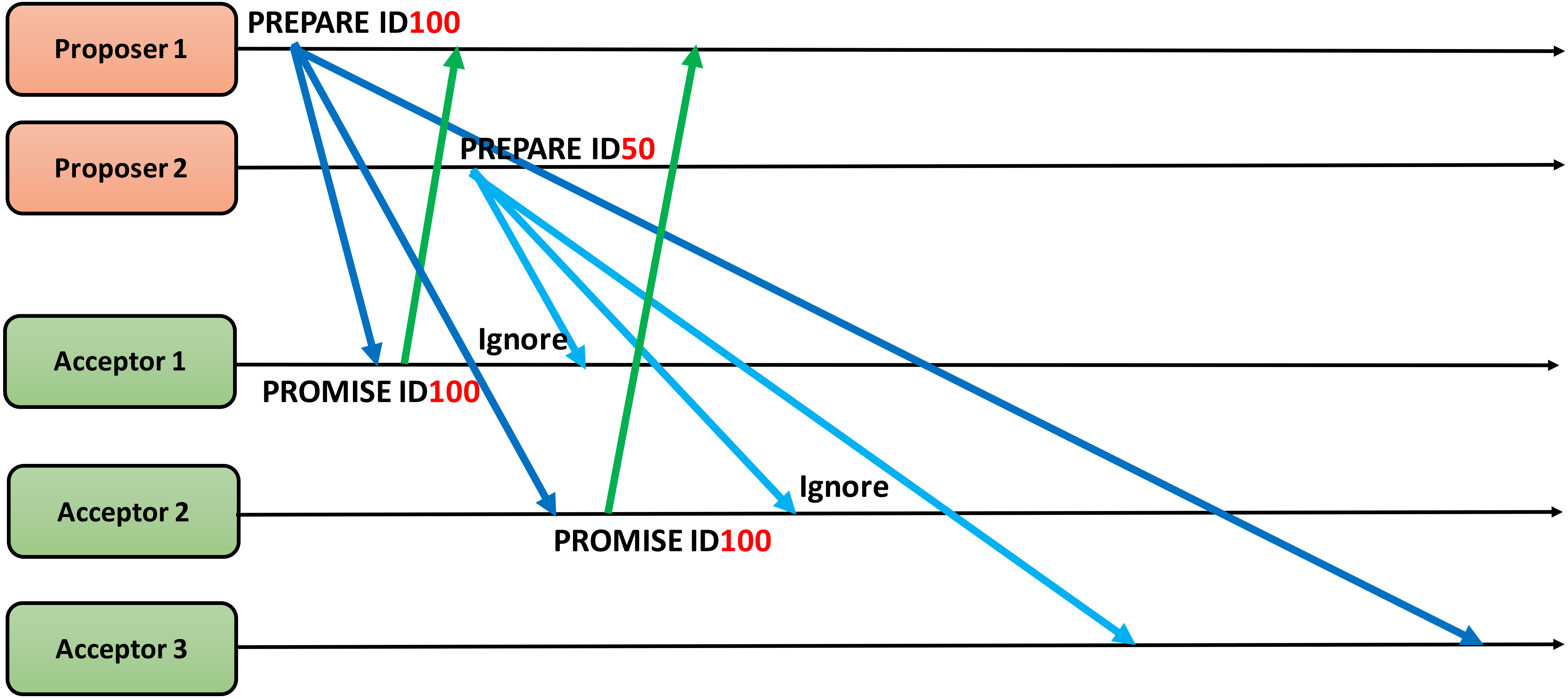


Paxos – How Majority Works

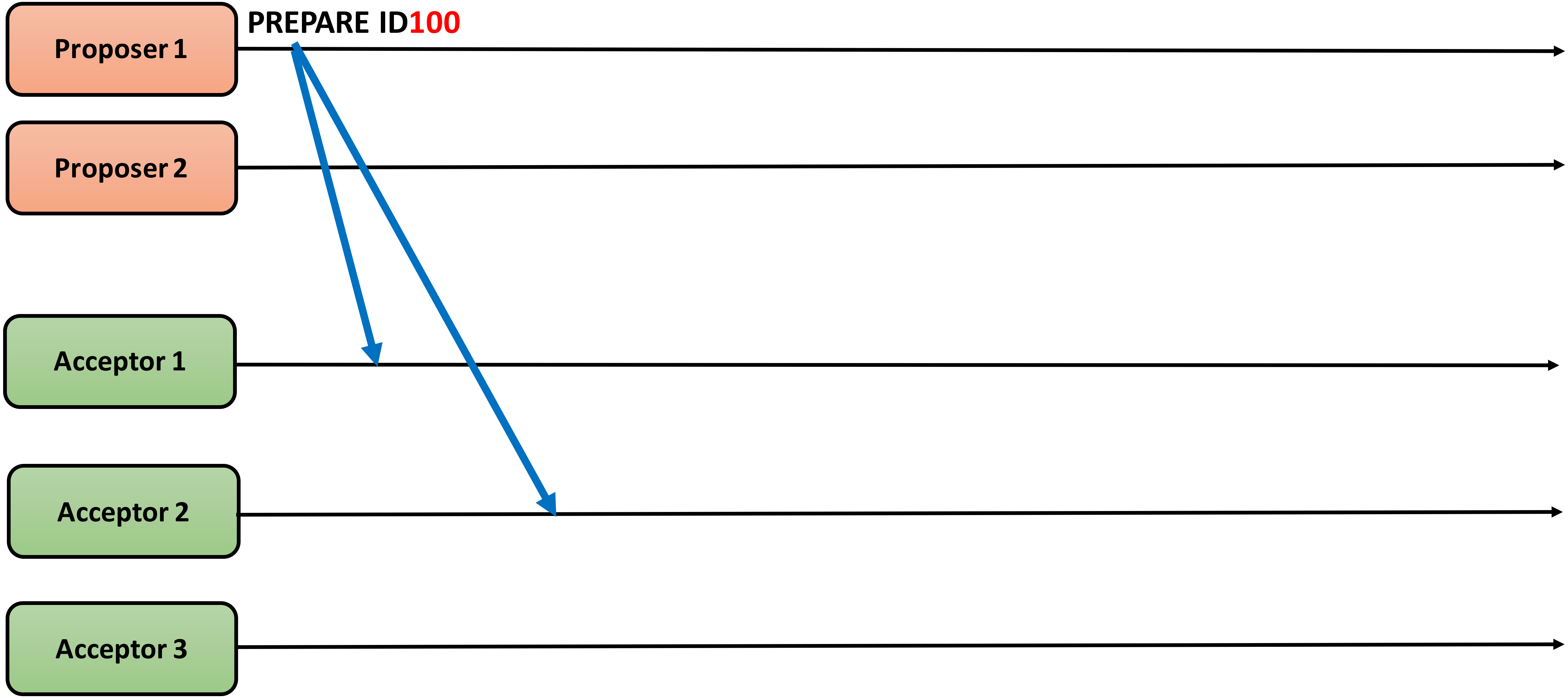


If the majority of the acceptors promised on ID100, ID50 cannot get through (those acceptors will ignore)

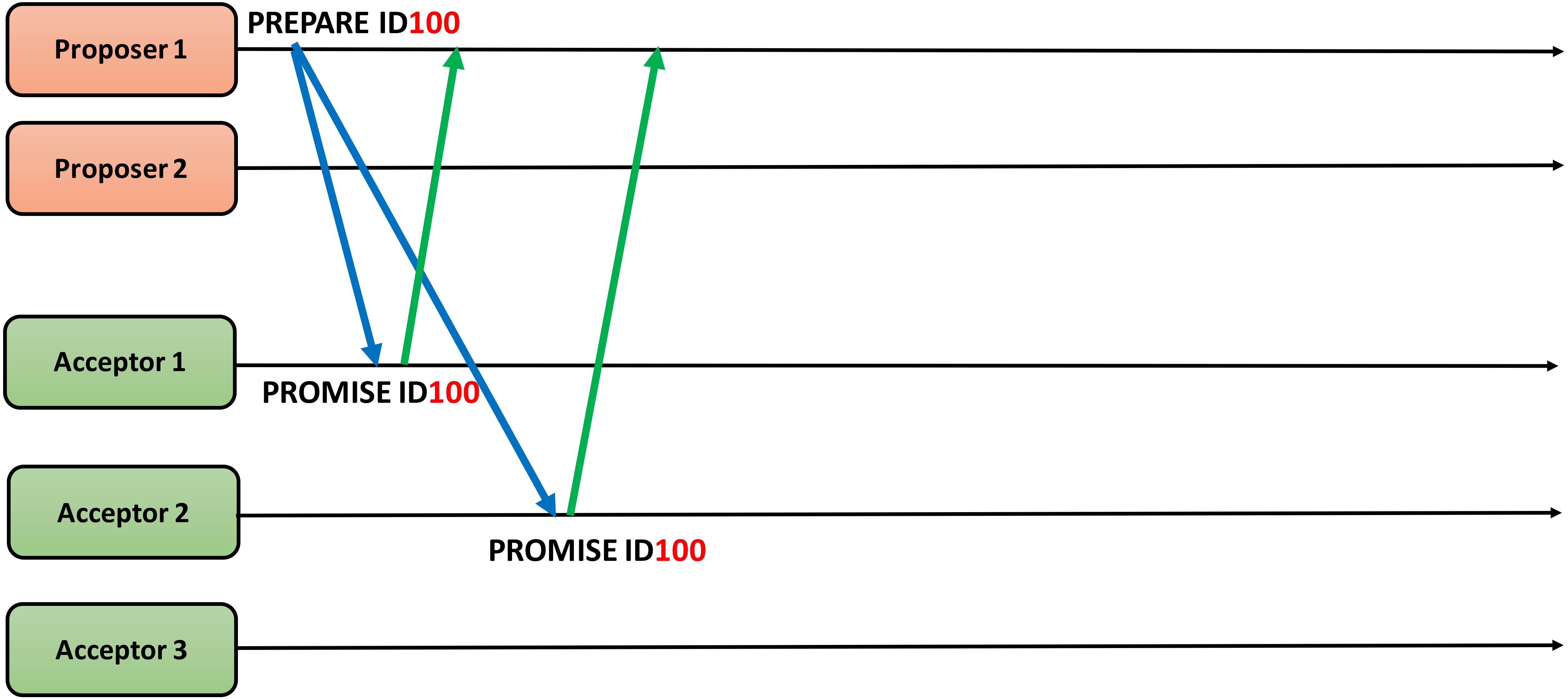
Paxos – How Majority Works



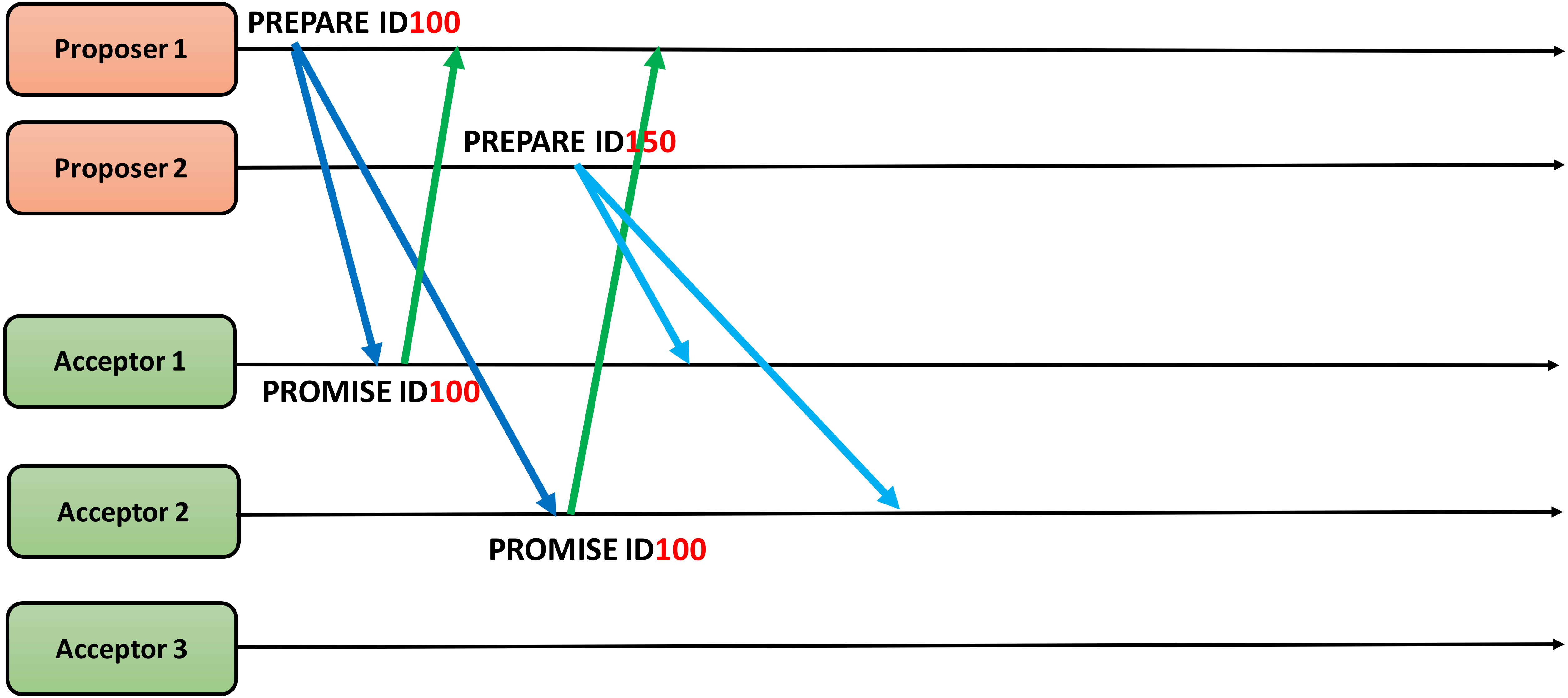
Paxos – Impact on Liveness



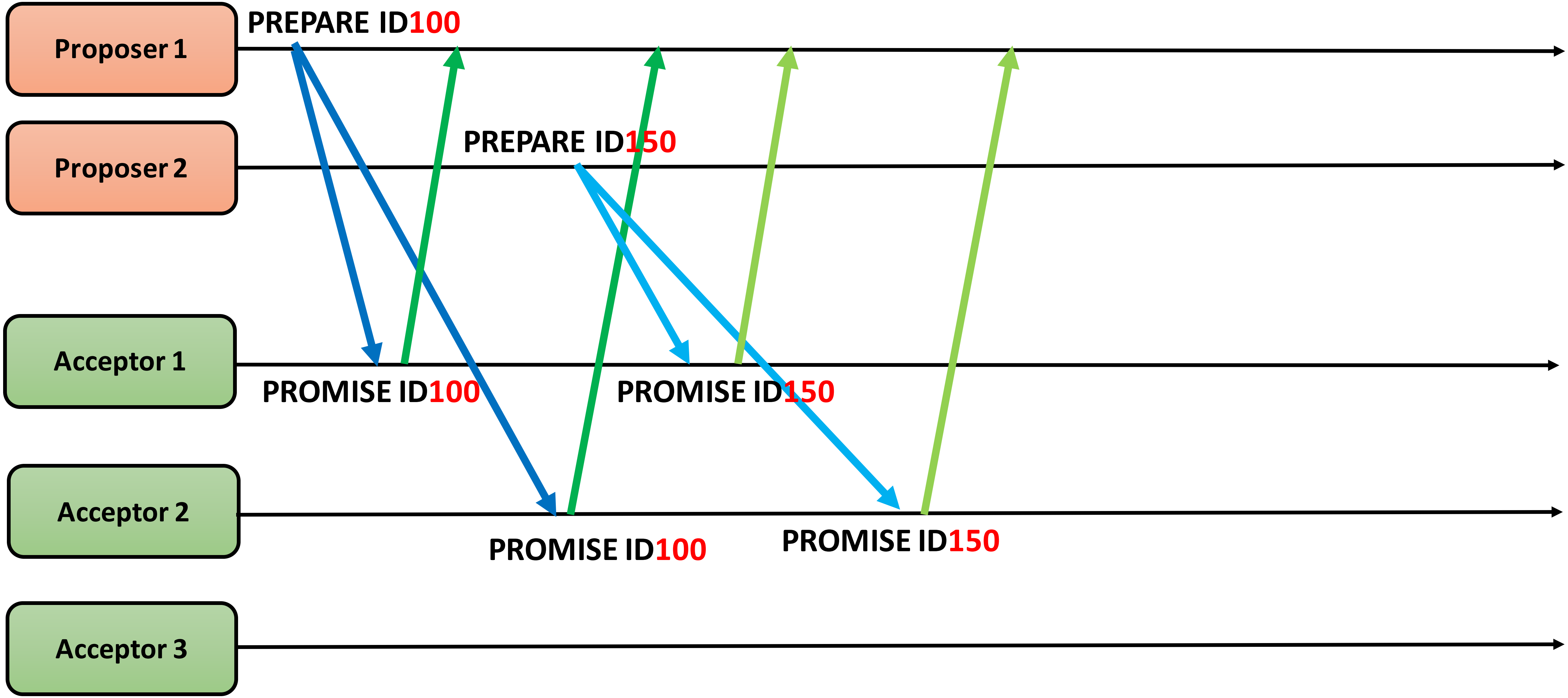
Paxos – Impact on Liveness



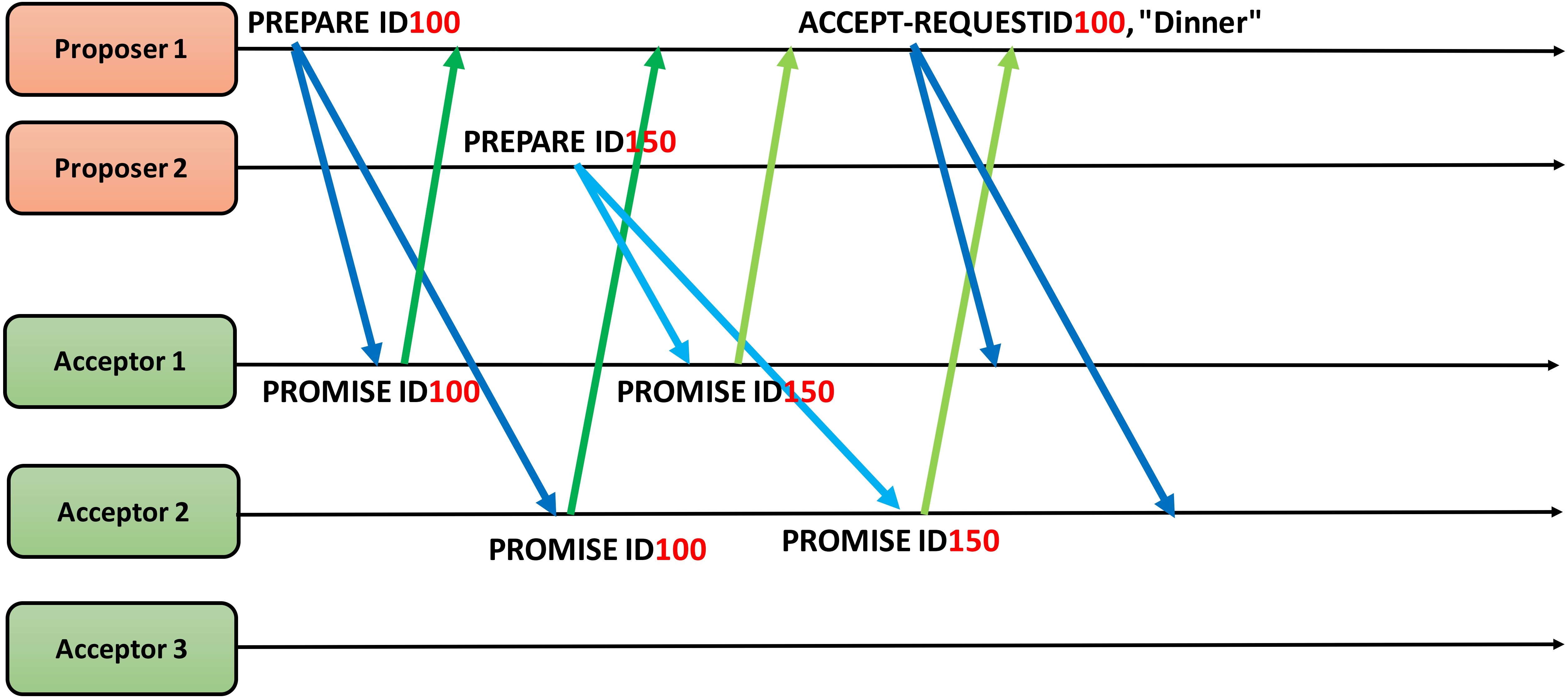
Paxos – Impact on Liveness



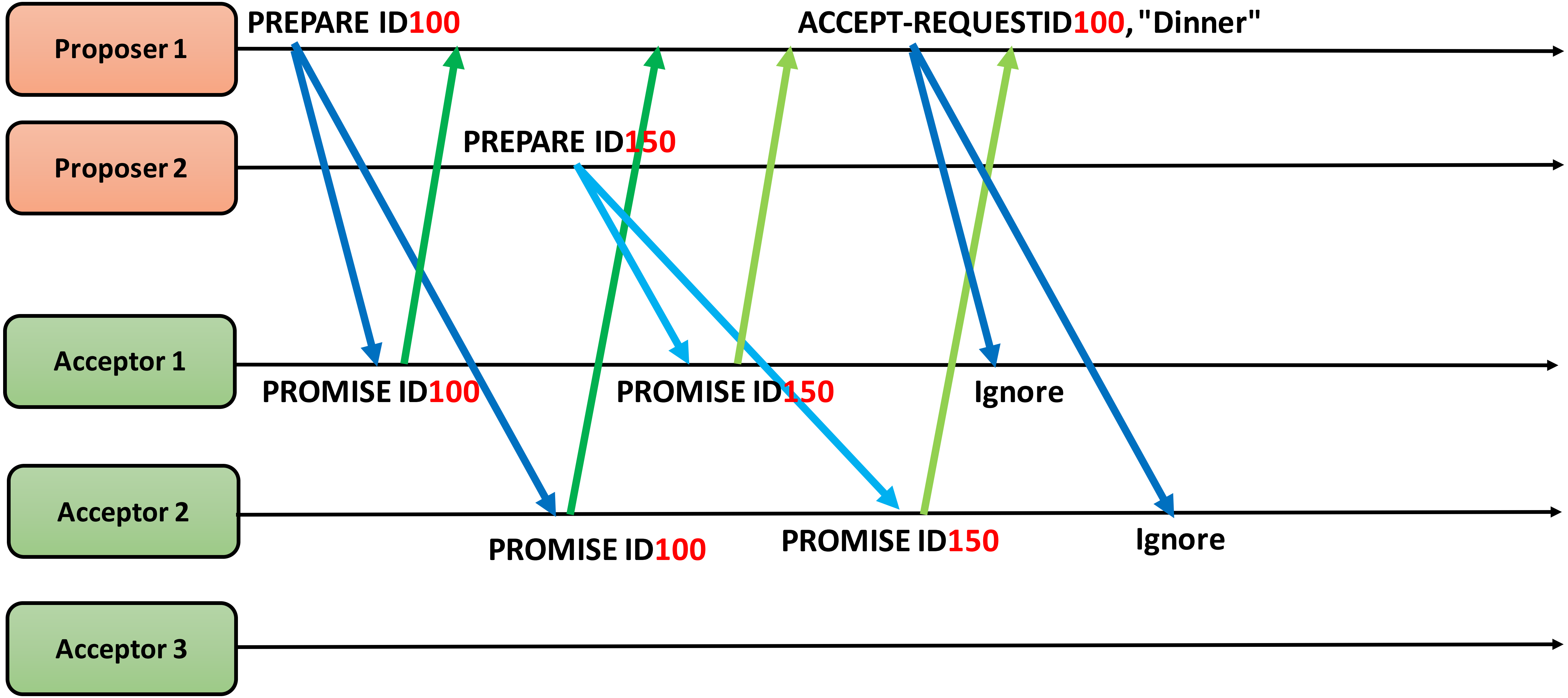
Paxos – Impact on Liveness



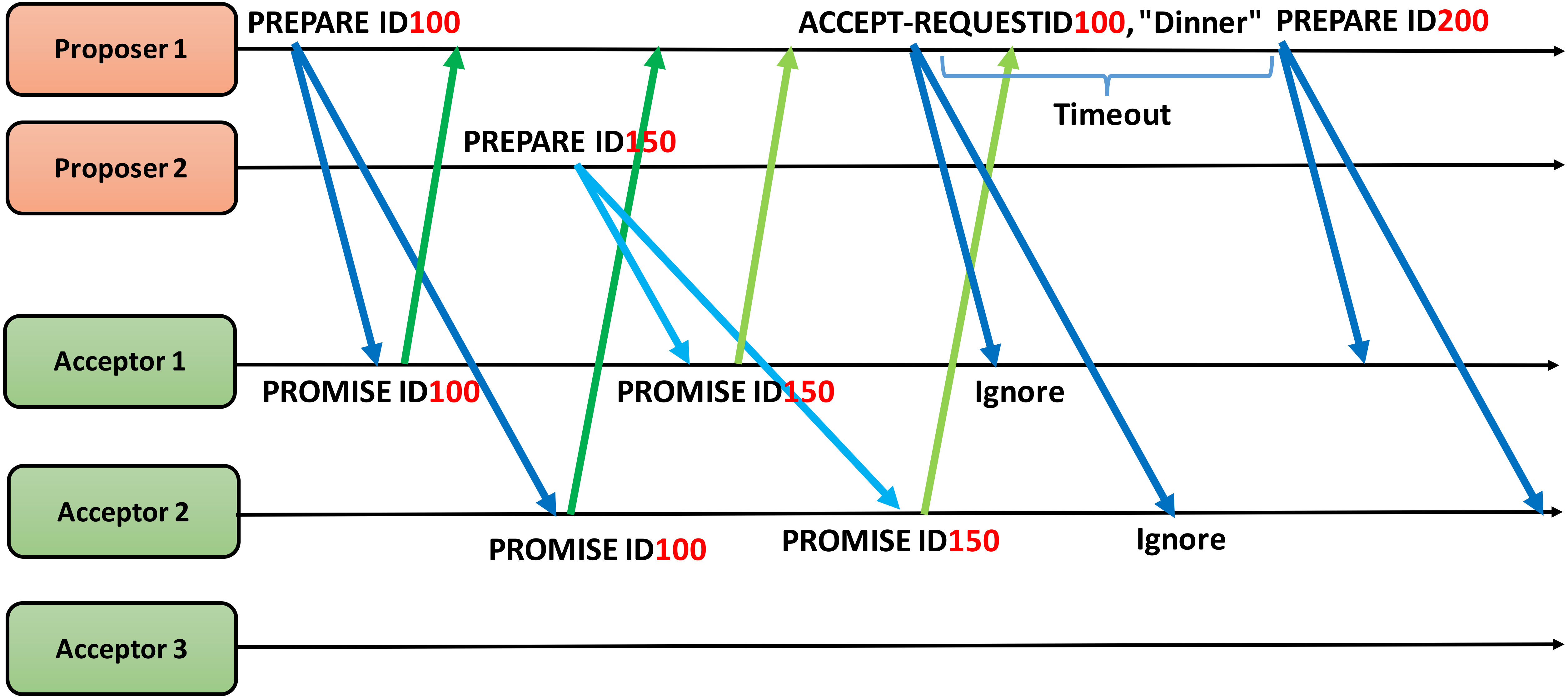
Paxos – Impact on Liveness



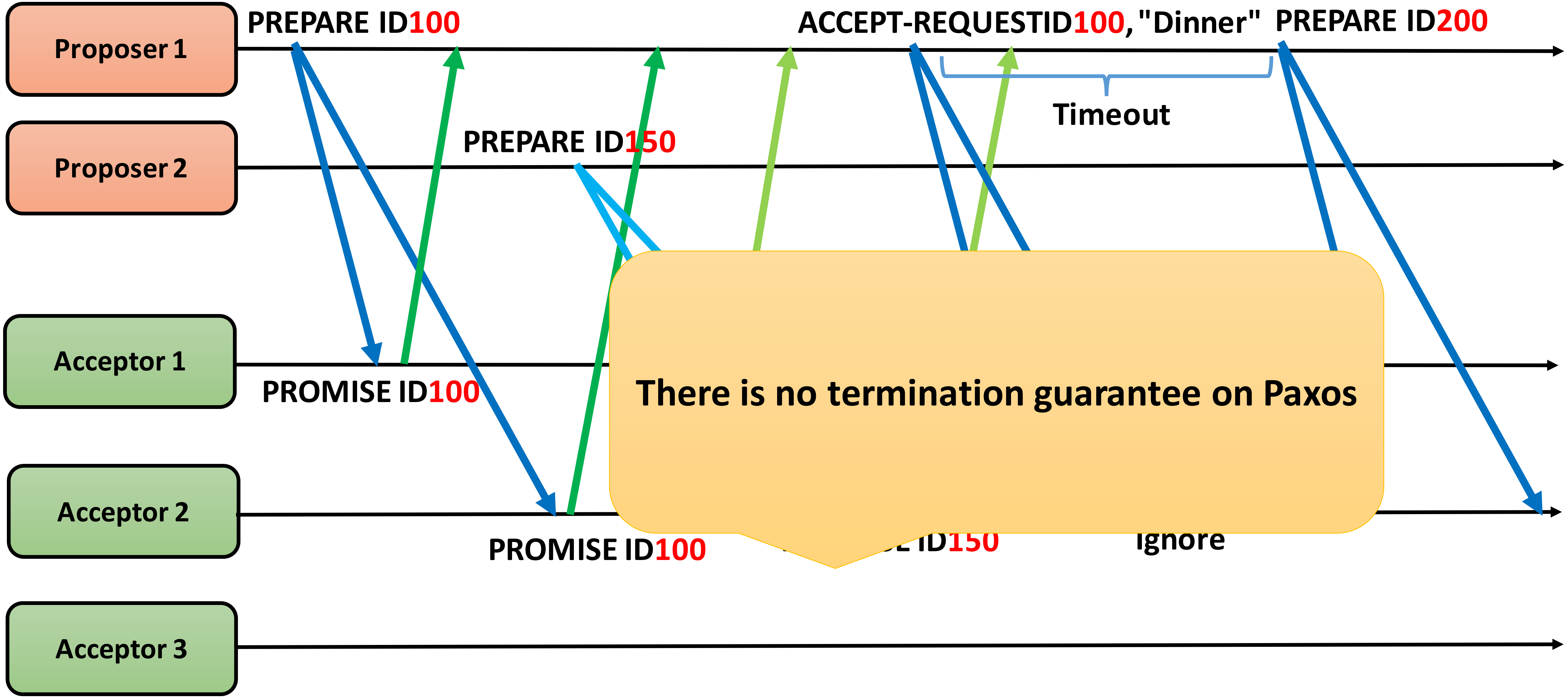
Paxos – Impact on Liveness



Paxos – Impact on Liveness



Paxos – Impact on Liveness



There is no termination guarantee on Paxos

Majority of Accepts

- Majority of accepts accepts 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 accepts accepts 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

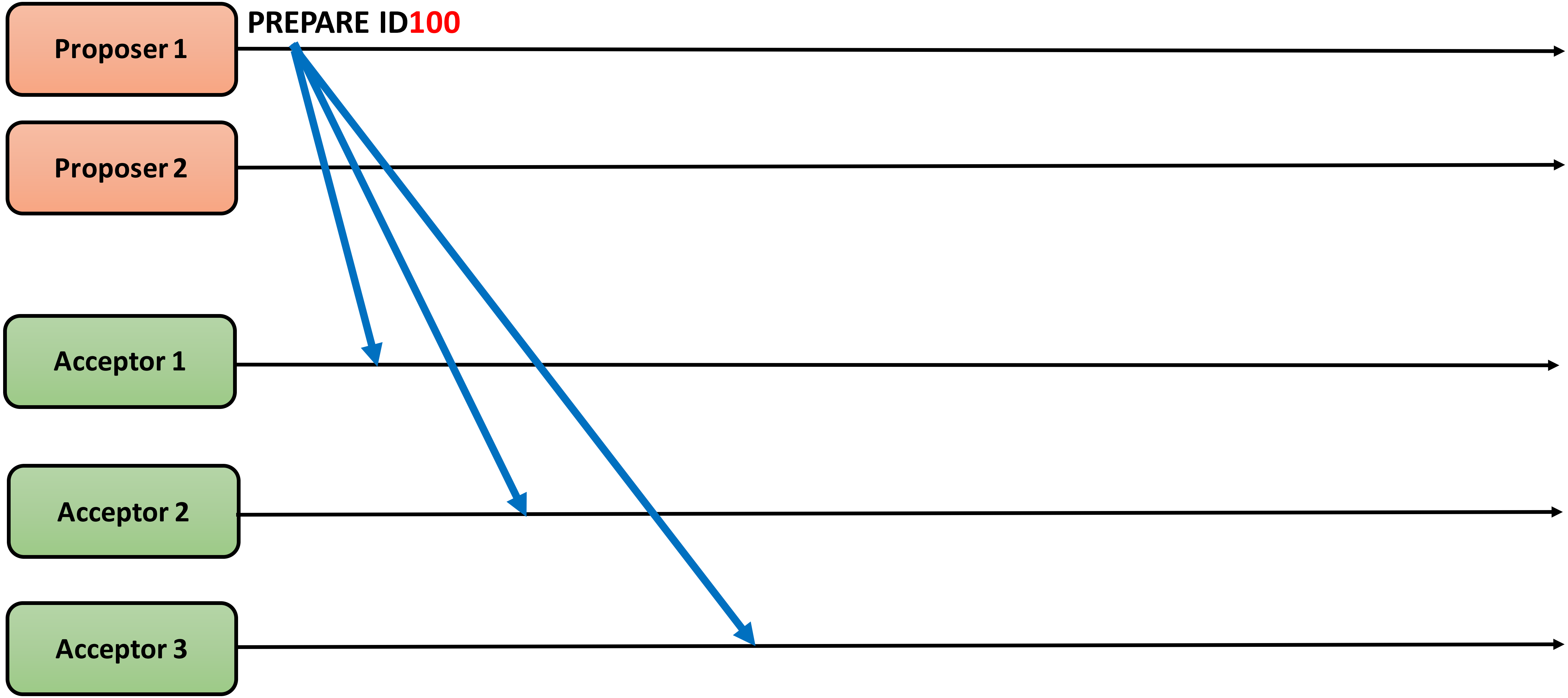
- Majority of accepts accepts a request with an ID and a value
  - Consensus has been reached
  - The consensus is on the **value**
- Accept request with a lower ID
  - Will not be accepted by the major e lower ID, but we got for a higher c

So, the consensus is on the value

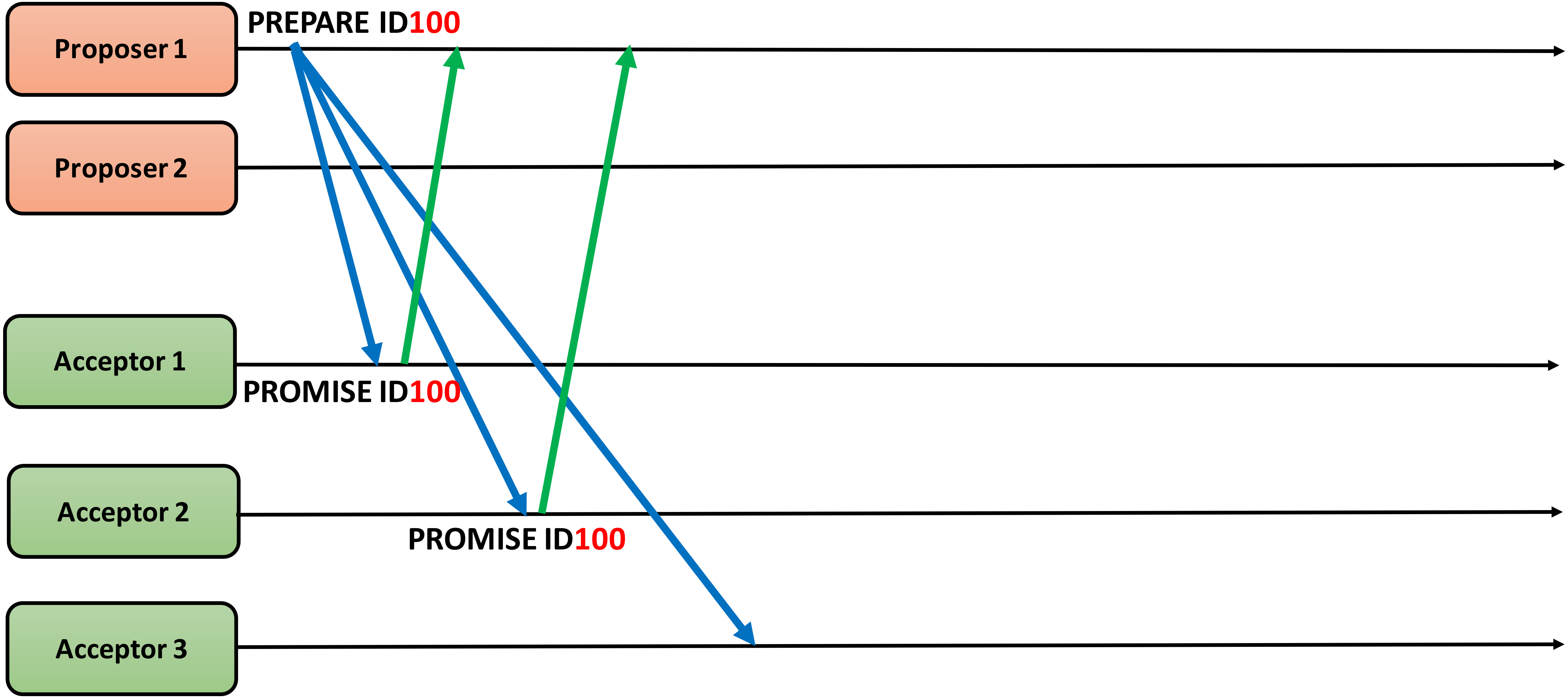
We need the ID to maintain the current state of promise and accept, so that multiple values does not propagate

- Accept request with a higher ID but a different value
  - 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)

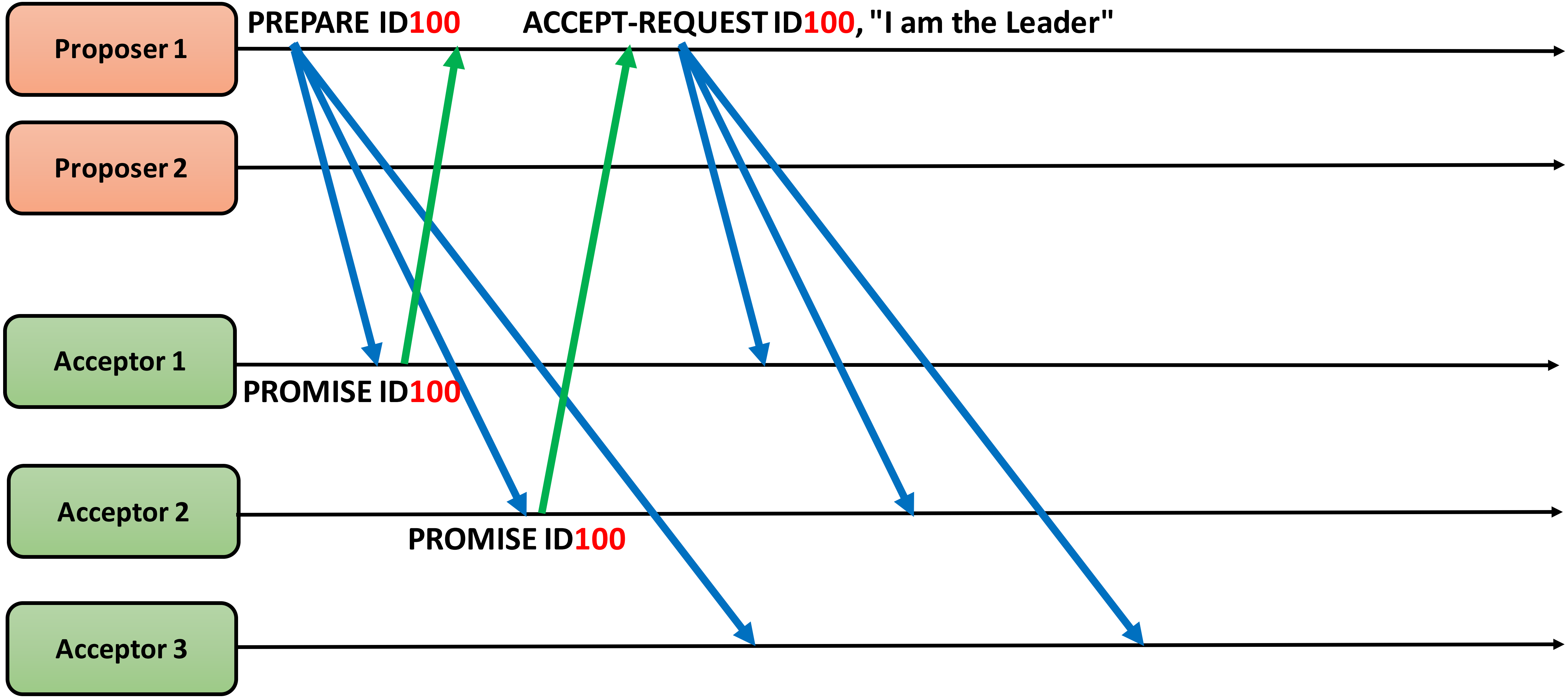
Paxos for Leader Election



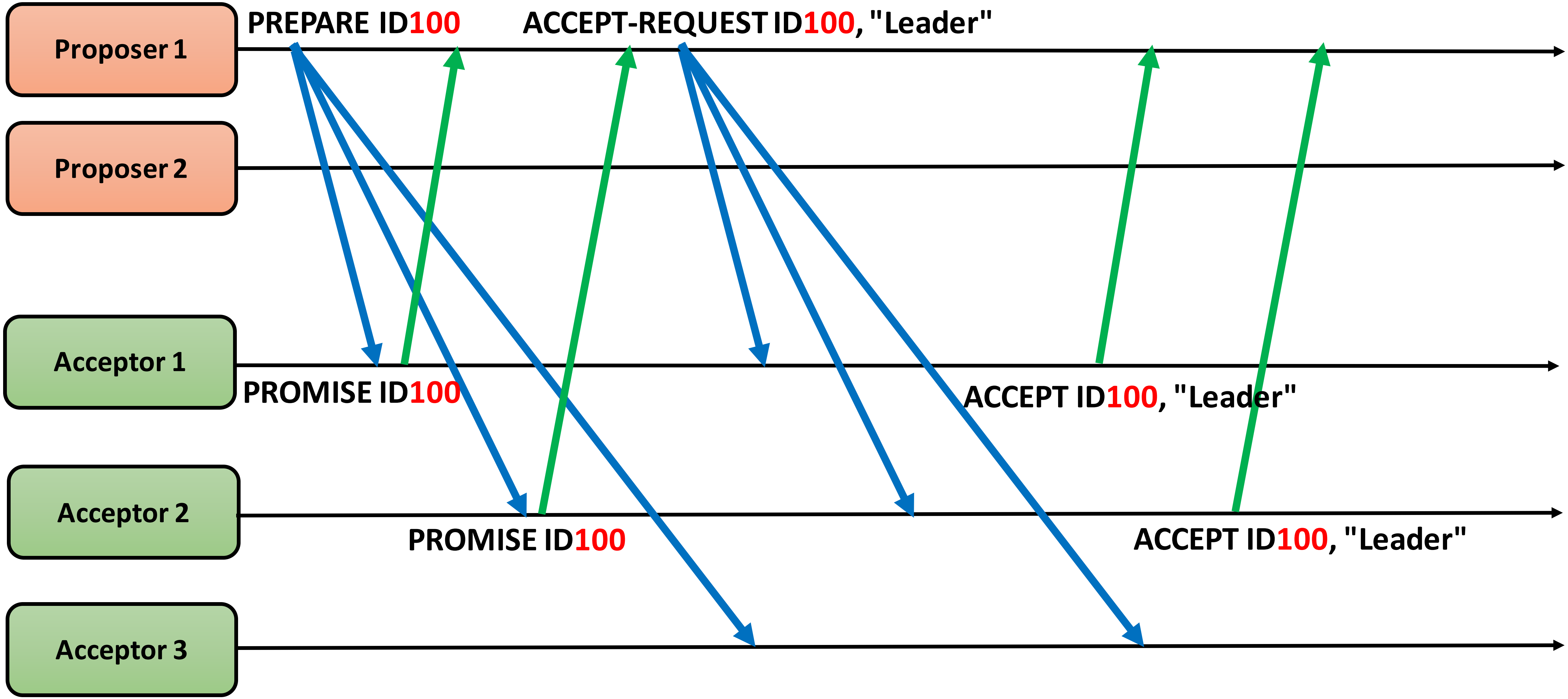
Paxos for Leader Election



Paxos for Leader Election



Paxos for Leader Election



Proposer 1

Proposer 2

Acceptor 1

Acceptor 2

Acceptor 3

PREPARE ID100

ACCEPT-REQUEST ID100, "Leader"

PROMISE ID100

PROMISE ID100

ACCEPT ID100, "Leader"

ACCEPT ID100, "Leader"

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

- 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*

Happy

Learning