# EECS 591
# Distributed Systems

Manos Kapritsos
Fall 2021

## Abstract

The Paxos algorithm, when presented in plain English, is very simple.

# Three types of processes

**Proposers** A proposer is a process that has a value to propose

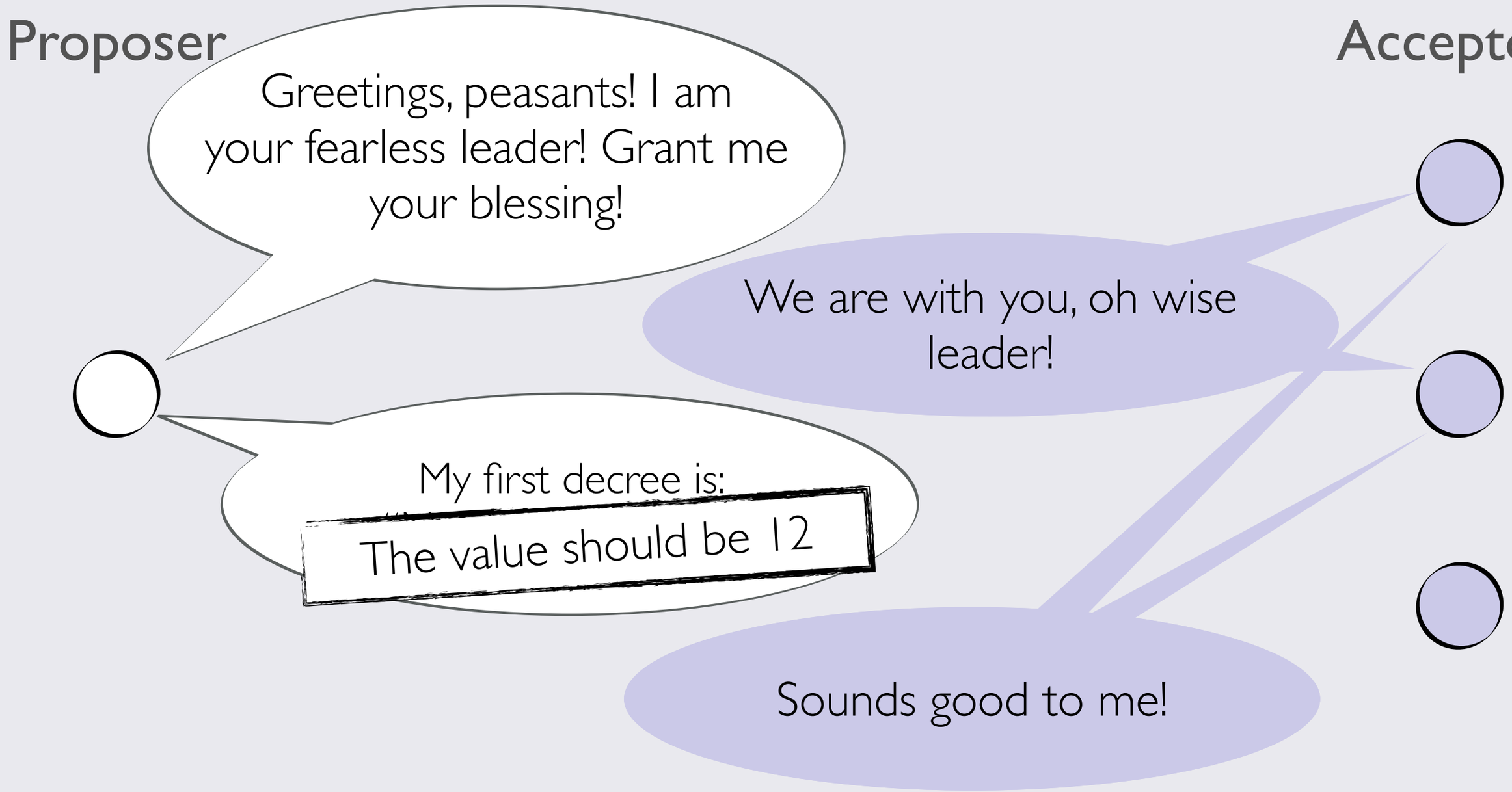
**Acceptors (2f+1)** Acceptors are the processes that ultimately choose which proposed value will be decided

**Learners** A learner only cares about learning which value was decided

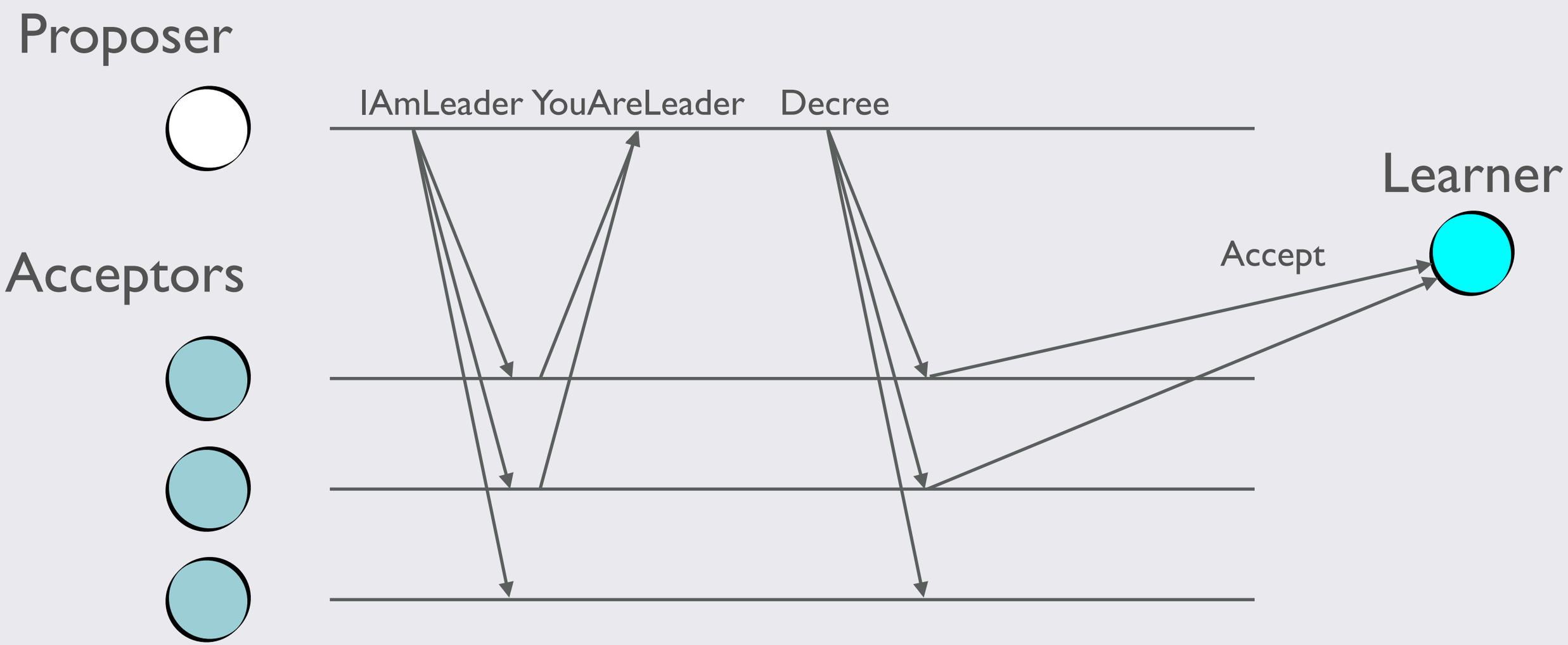
# How the game is played

- **Election:** Proposers first try to get a majority of acceptors to follow them.
- **Legislating:** After acquiring a majority, a proposer can ***try*** to enforce her value, by getting acceptors to accept it, **but…**
- **Playing nice:** If an elected proposer finds that some previous value has been proposed, she proposes that value instead.
- **Winning the game:** once a majority of acceptors have accepted a value, the value is **chosen/decided**

# How it is supposed to work

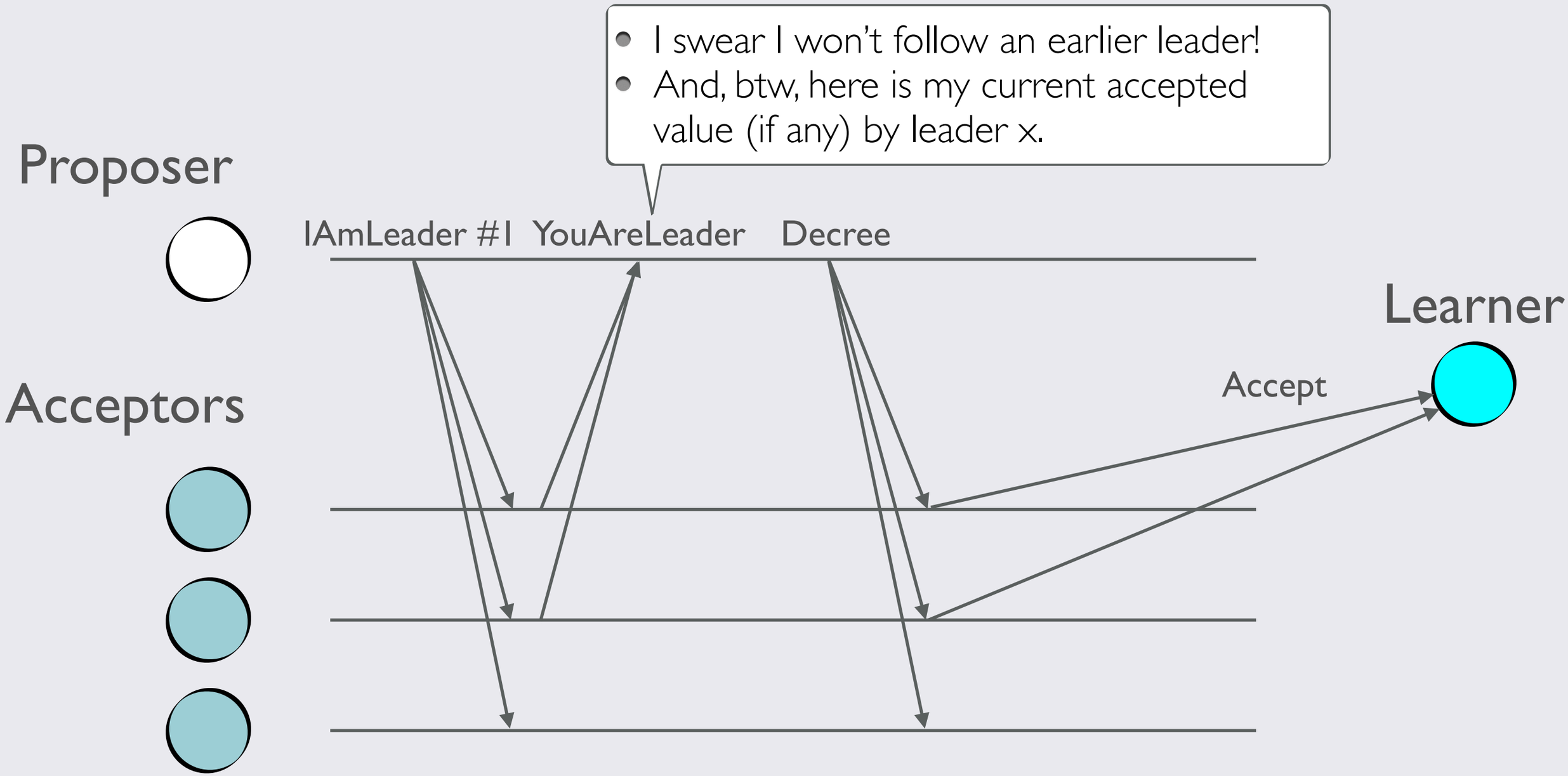


# How it is supposed to work

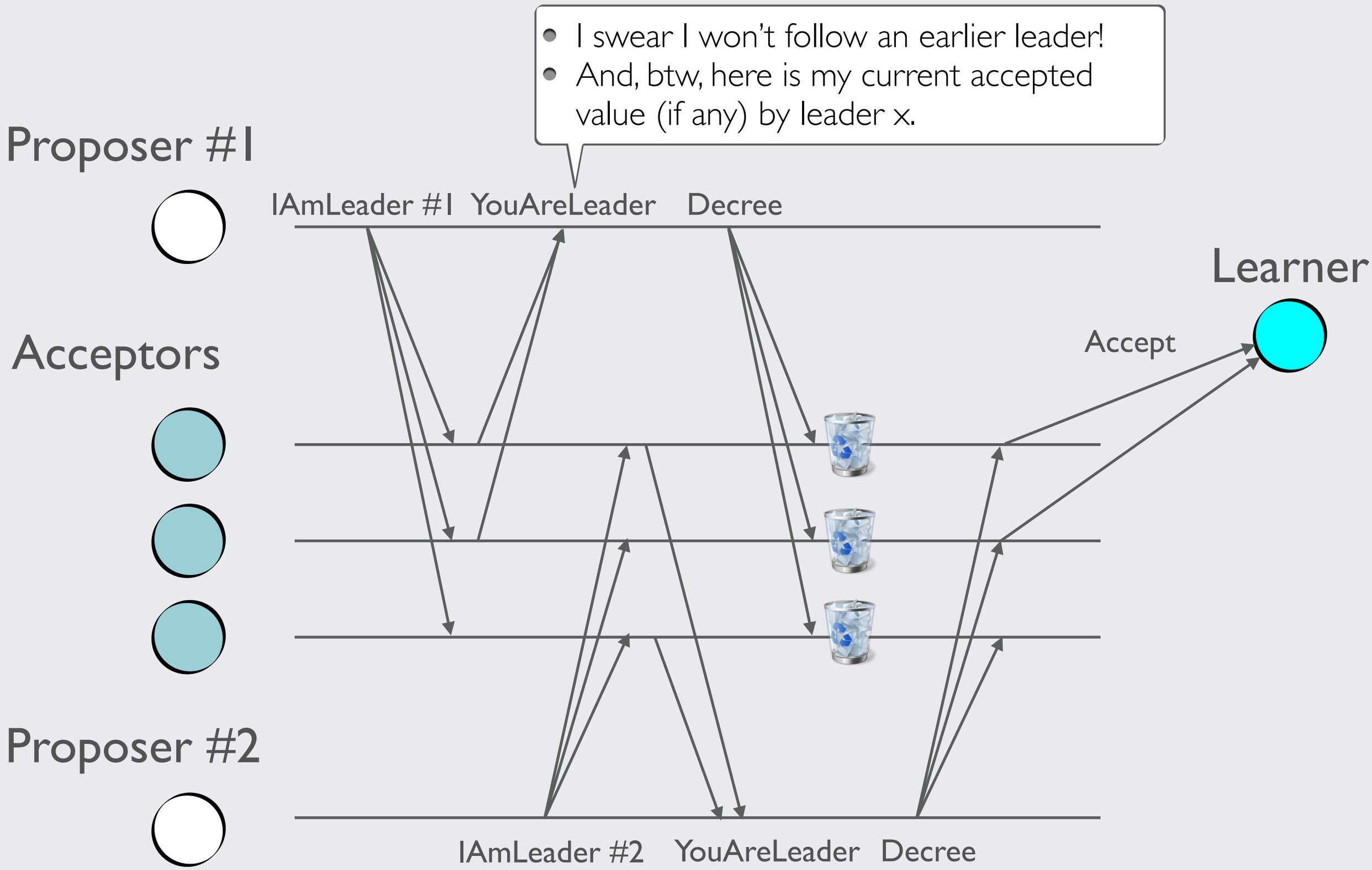


# DEALING WITH MULTIPLE PROPOSERS

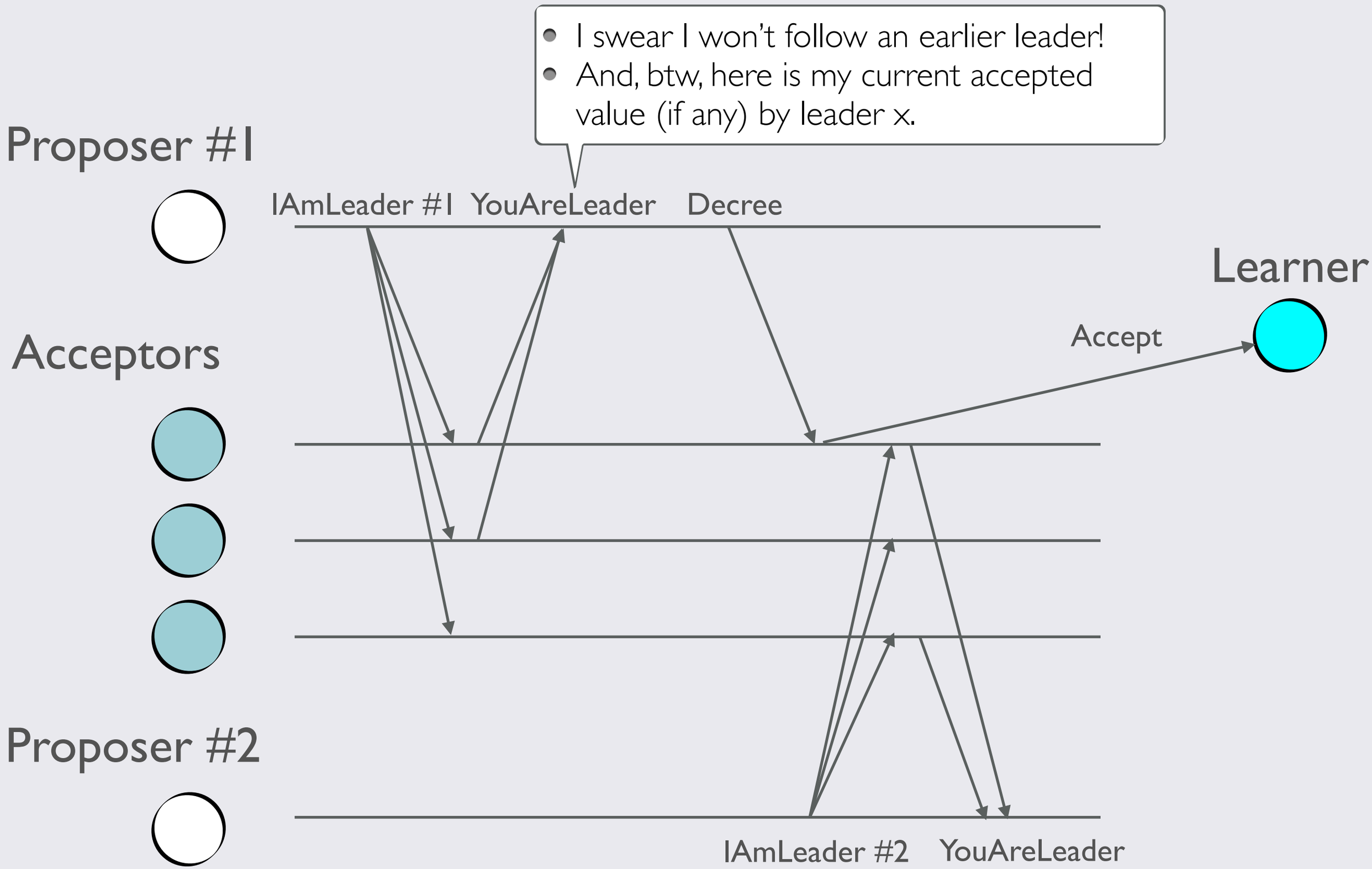
# DEALING WITH MULTIPLE PROPOSERS



# DEALING WITH MULTIPLE PROPOSERS



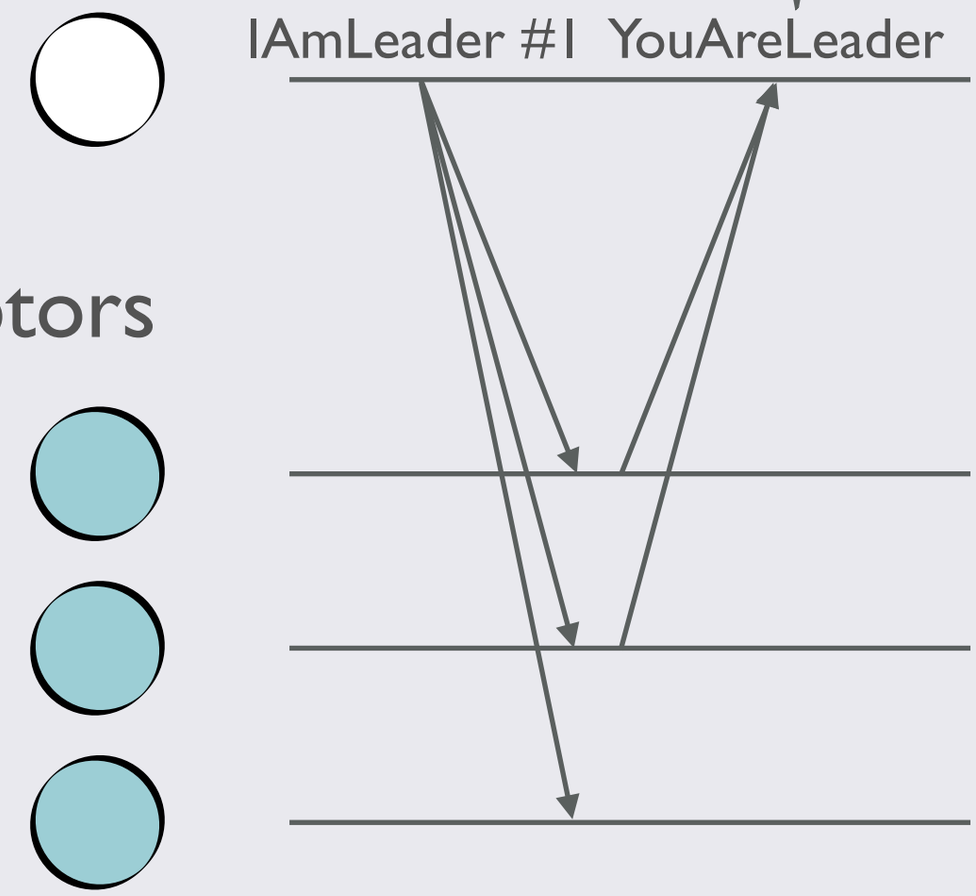
# DEALING WITH MULTIPLE PROPOSERS



# THE CRUCIAL YouAreLeader MESSAGE

- I swear I won't follow an earlier leader!
- And, btw, here is my current accepted value (if any) by leader x.

Proposer #1




Acceptors

1. Wait for a majority of ***YouAreLeader*** messages before proceeding.

2. If none of them contain a previously accepted value, propose your own
   Otherwise, propose the value of the **most recent** leader.

# THE CRUCIAL YouAreLeader MESSAGE

1. Wait for a majority of ***YouAreLeader*** messages before proceeding.
2. If none of them contain a previously accepted value, propose your own
   Otherwise, propose the value of the **most recent** leader.

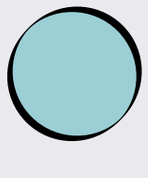
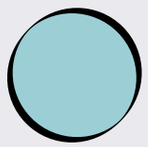
**Important**

By consulting a majority, the new leader makes sure she cannot have missed a chosen value

**(a value must be accepted by a majority to be chosen, and any two majorities overlap!)**

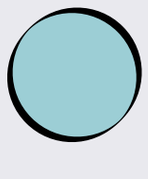
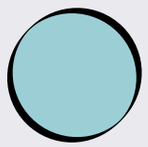
# EXAMPLES OF ACCEPTOR STATES

(as leader #50 comes to power)

| Acceptors | Value | By leader |
|---|---|---|
|  | x | 37 |
|  | - | - |
|  | - | - |
|  | - | - |
|  | - | - |

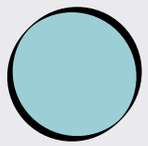
# EXAMPLES OF ACCEPTOR STATES

(as leader #50 comes to power)

| Acceptors | Value | By leader |
|---|---|---|
|  | x | 37 |
|  | y | 42 |
|  | - | - |
|  | x | 37 |
|  | y | 41 |

# EXAMPLES OF ACCEPTOR STATES

(as leader #50 comes to power)

| Acceptors | Value | By leader |
|---|---|---|
|  | x | 37 |
|  | y | 42 |
|  | - | - |
|  | x | 37 |
|  | x | 41 |

# Overview of Paxos

**Proposer** | **Acceptor**

Send *IAmLeader(n)* to all

Wait for a majority of responses

If ***n*** is the highest leader # I have seen:
respond with
*YouAreLeader(Value, LeaderWhoProposedValue)*

Once majority is received, send
***Propose(n,V)*** where V is the highest-leader proposal among the responses (or my own value, if none of the responses had a value)

If ***n*** is the highest leader # I have seen, send
***Accept(n,V)*** to the learner

# TOLERATING $f$ FAILURES

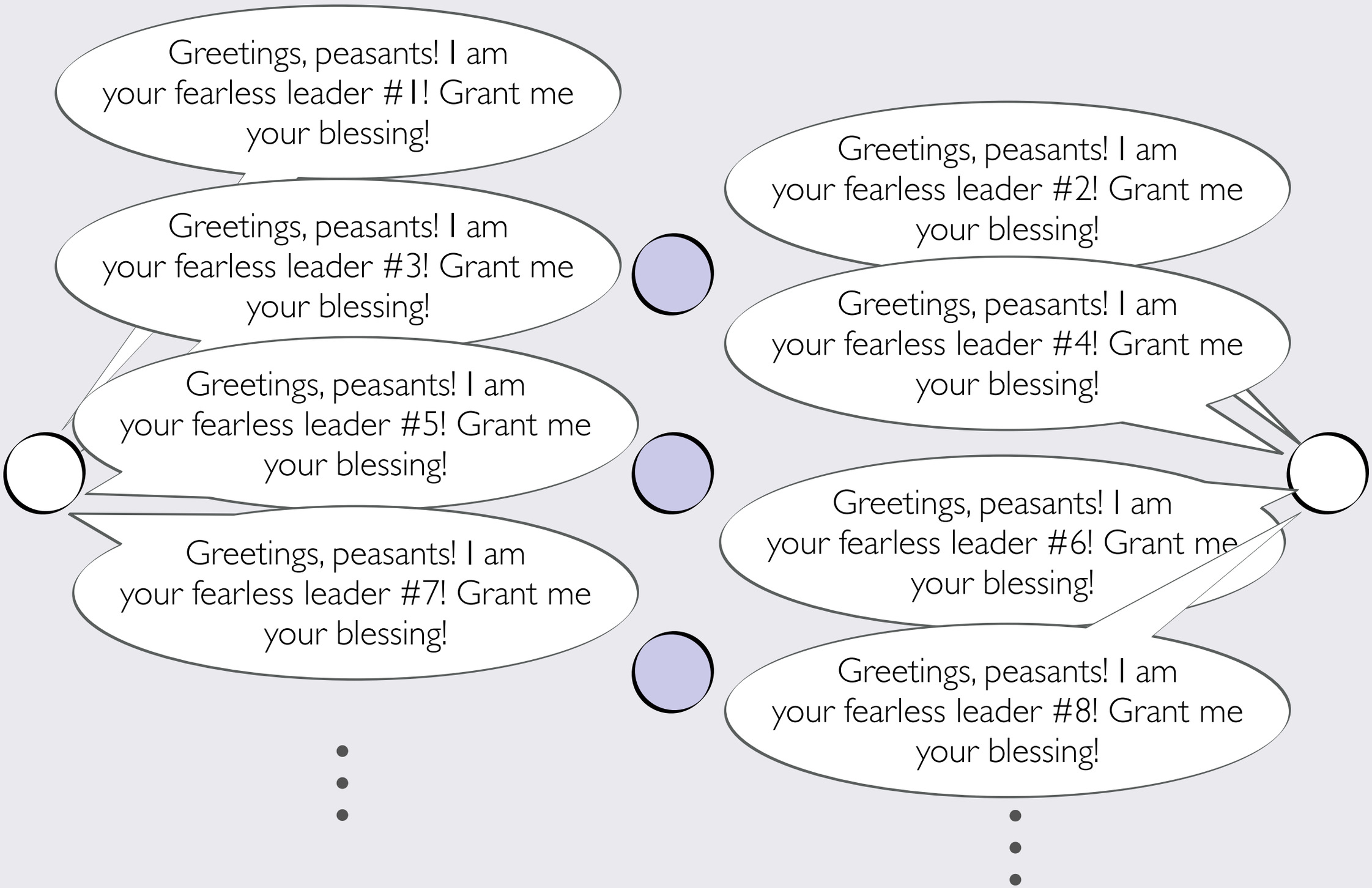
**Safety**

- There are $2f + 1$ acceptors
- A value is only chosen if accepted by a majority ($f + 1$)
- So, even if $f$ of those acceptors fail, one will remain and **will be part** of any future majority

**Liveness**

- The leader always waits for $f + 1$ responses. So, even if $f$ replicas fail, it will not block

# THE THREAT TO LIVENESS: DUELING PROPOSERS



# The threat to liveness: dueling proposers

This problem can be avoided during synchrony
(proposer faults can be detected accurately using timeouts)

It's **impossible** to avoid during asynchrony!

Well, we kind of knew that already…

# The beauty of Paxos

Paxos **cannot** be both safe and live during asynchrony!

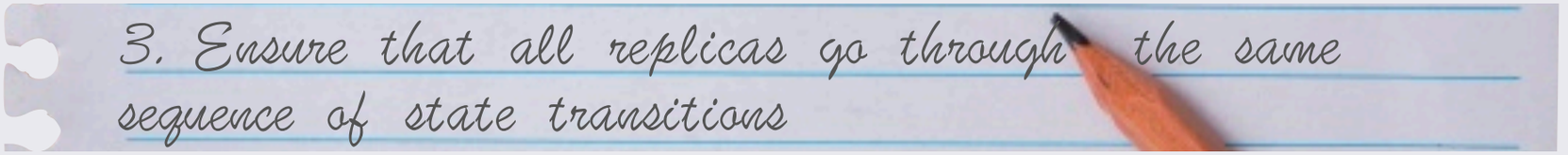
(that would violate FLP)

So it's doing the next best thing:
staying **safe all the time** and achieving liveness
when the system starts behaving synchronously

# Using (multi)Paxos to implement State Machine Replication

The original Paxos algorithm achieves agreement on **one** value

SMR required replicas to agree on the **sequence** of commands that will be executed





MultiPaxos: Run an instance of Paxos for each slot in the sequence

**Important**: we don't need to run phase 1 (election) every time!