Paxos Made Simple

« The Paxos algorithm, when presented in plain English, is very simple… In fact, it is among the simplest and most obvious of distributed algorithms.»

-- Leslie Lamport 2001

But in reality it is complex!

« The dirty little secret of the community is that at most 5 people really, truly understand every part of Paxos.»

-- NSDI reviewer

« There are significant gaps between the description of the Paxos algorithm and the needs of a real-world system… The final system will be based on an unproven protocol.»

-- Chubby authors

What is to reach consensus with Paxos?

Consensus is agreeing on **one** result.
Once a **majority** agrees on a proposal, that is the consensus.
The reached consensus can be **eventually** known by everyone.
The involved parties want to agree on **any** result, not on their proposal.
Communication channels may be **faulty**, that is, messages can get lost.

Problem

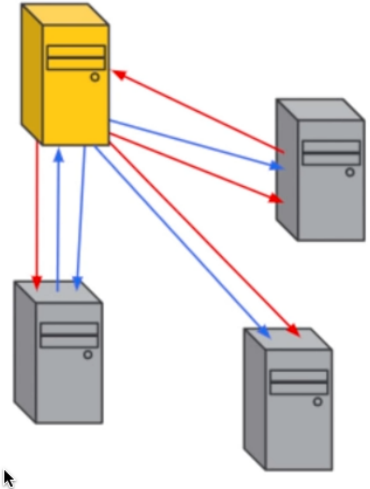
- How to reach consensus/data consistency in distributed system that can tolerate non-malicious failures?

Why do systems need to reach consensus?

Infinitely powerful and scalable single computer

So, either

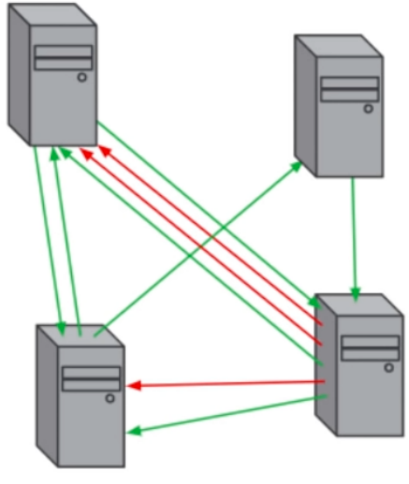
Leader-replicas schema



If the leader becomes unavailable, nodes must reach **consensus** to elect a new one

or

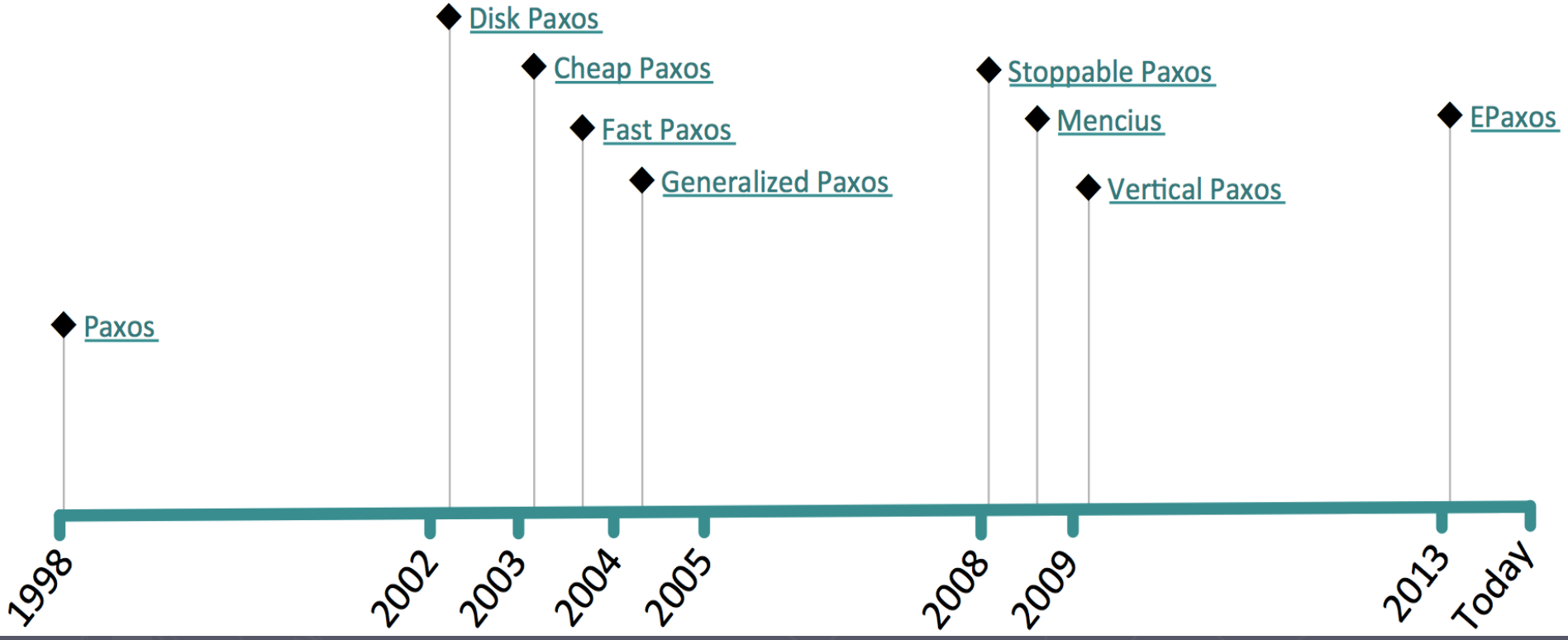
Peer-to-peer schema



The nodes need to reach **consensus** continuously so as to guarantee consistency

Applications

- Chubby: distributed lock service (Google)
- Spanner & Megastore: NewSQL DBs (Google)
- ZooKeeper: distributed lock service (Apache)
- Petal: distributed virtual disks
- Frangipani: scalable distributed file system
- SVC: storage virtualization (IBM)
- …


Disk Paxos
Cheap Paxos
Fast Paxos
Generalized Paxos
Stoppable Paxos
Mencius
Vertical Paxos
EPaxos
Paxos
1998
2002
2003
2004
2005
2008
2009
2013
Today

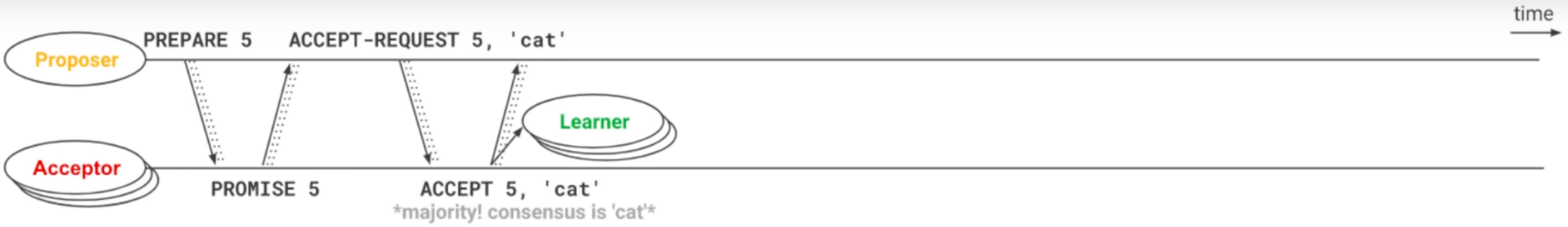
Paxos Basics

Paxos defines three roles: **proposers**, **acceptors**, and **learners**.

Paxos nodes can take multiple roles, even all of them.
Paxos nodes must know how many **acceptors** a majority is
(two majorities will always overlap in at least one node).
Paxos nodes must be persistent: they can't forget what they accepted.

A **Paxos run** aims at reaching a **single consensus**.
Once a consensus is reached, it **cannot progress** to another consensus.
In order to reach **another consensus**, a different **Paxos run** must happen.

The Paxos Algorithm



⇨ **Proposer** wants to propose a certain value:
It sends **PREPARE** **IDp** to a majority (or all) of **Acceptors**.
IDp must be unique, e.g. slotted timestamp in nanoseconds.
e.g. **Proposer** 1 chooses IDs 1, 3, 5...
Proposer 2 chooses IDs 2, 4, 6..., etc.
Timeout? retry with a new (higher) IDp.

⇨ **Acceptor** receives a **PREPARE** message for IDp:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Will promise to ignore any request lower than IDp.
(?) Reply with **PROMISE** **IDp**.

1 **If a majority of acceptors promise, no ID<IDp can make it through.**

⇨ **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.

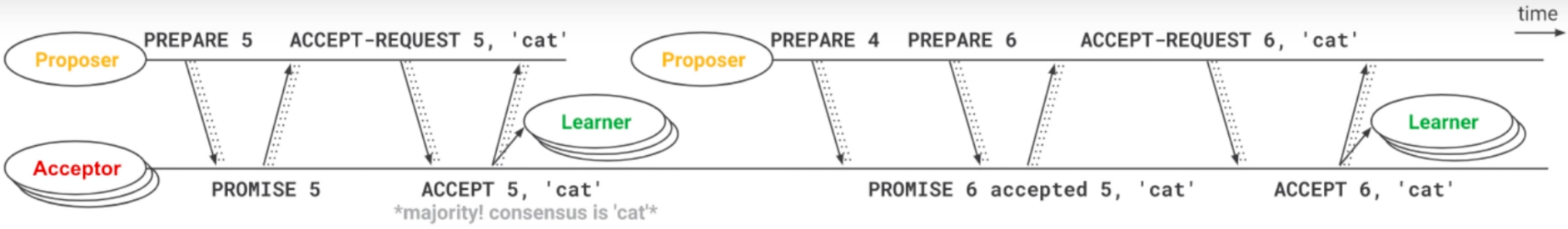
⇨ **Acceptor** receives an **ACCEPT-REQUEST** message for IDp, value:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Reply with **ACCEPT** **IDp**, **value**. Also send it to all **Learners**.

2 **If a majority of acceptors accept IDp, value, consensus is reached.**
Consensus is and will always be on value (not necessarily IDp).

⇨ **Proposer** or **Learner** get **ACCEPT** messages for IDp, value:

3 **If a proposer/learner gets majority of accept for a specific IDp,**
they know that consensus has been reached on value (not IDp).

The Paxos Algorithm



⇨ **Proposer** wants to propose a certain value:
It sends **PREPARE IDp** to a majority (or all) of **Acceptors**.
IDp must be unique, e.g. slotted timestamp in nanoseconds.
e.g. **Proposer** 1 chooses IDs 1, 3, 5...
Proposer 2 chooses IDs 2, 4, 6..., etc.
Timeout? retry with a new (higher) IDp.

⇨ **Acceptor** receives a **PREPARE** message for IDp:
Did it promise to ignore requests with this IDp?
Yes -> then 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**.

1 **If a majority of acceptors promise, no ID<IDp can make it through.**

⇨ **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 - > It picks the value with the highest **IDa** that it got.
No -> It picks any value it wants.

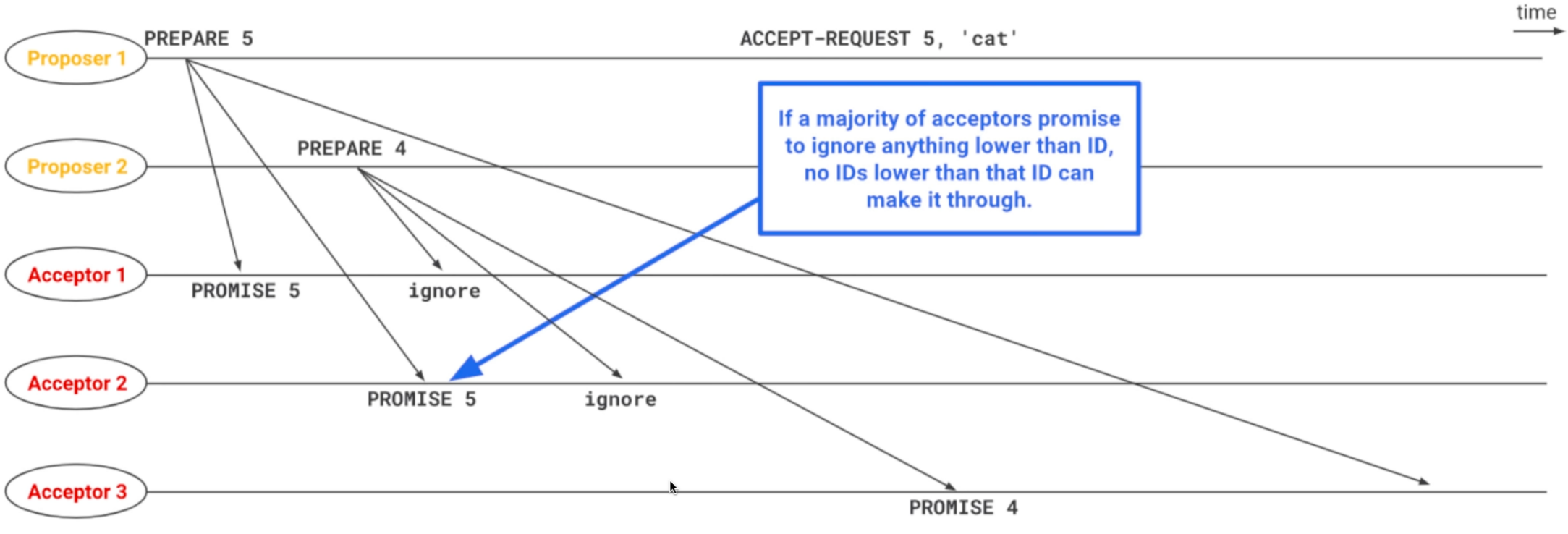
⇨ **Acceptor** receives an **ACCEPT-REQUEST** message for IDp, value:
Did it promise to ignore requests with this IDp?
Yes -> then ignore
No -> Reply with **ACCEPT IDp, value**. Also send it to all **Learners**.

2 **If a majority of acceptors accept IDp, value, consensus is reached.**
Consensus is and will always be on value (not necessarily IDp).

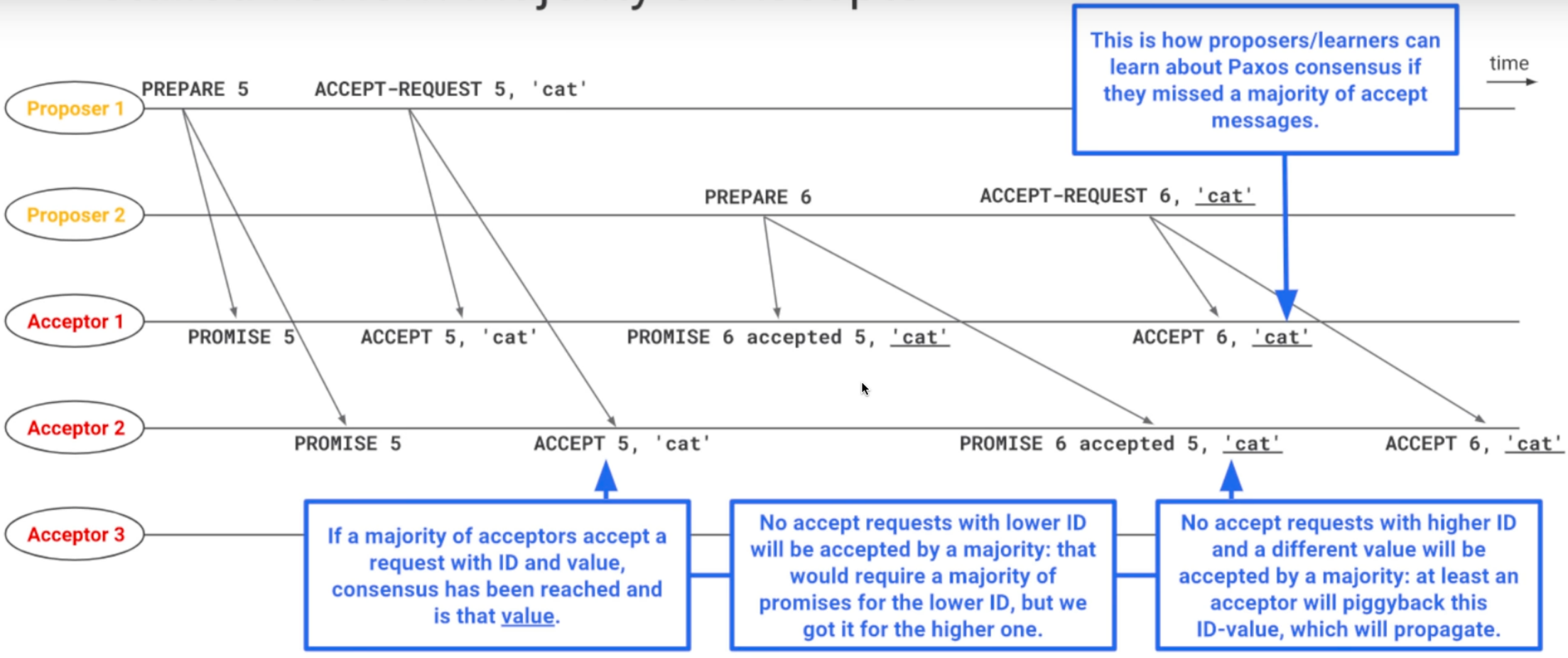
⇨ **Proposer** or **Learner** get **ACCEPT** messages for IDp, value:

3 **If a proposer/learner gets majority of accept for a specific IDp,**
they know that consensus has been reached on value (not IDp).

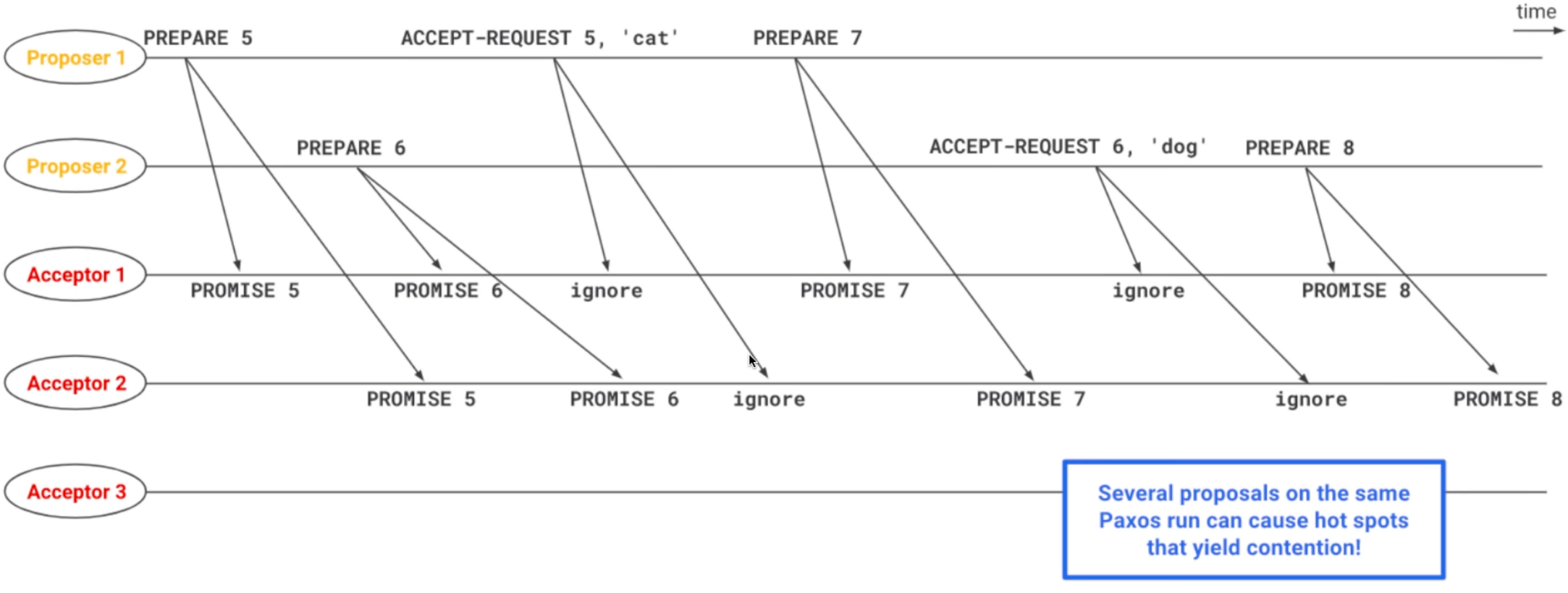
Detailed Review: Majority of Promises



Detailed Review: Majority of Accepts



Detailed Review: Contention

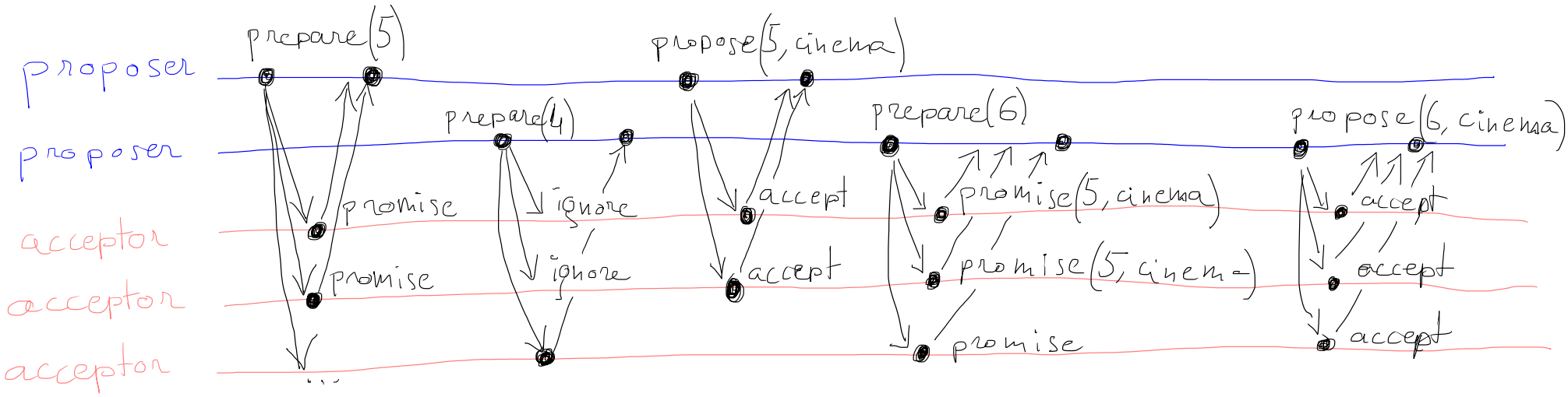


Paxos's properties

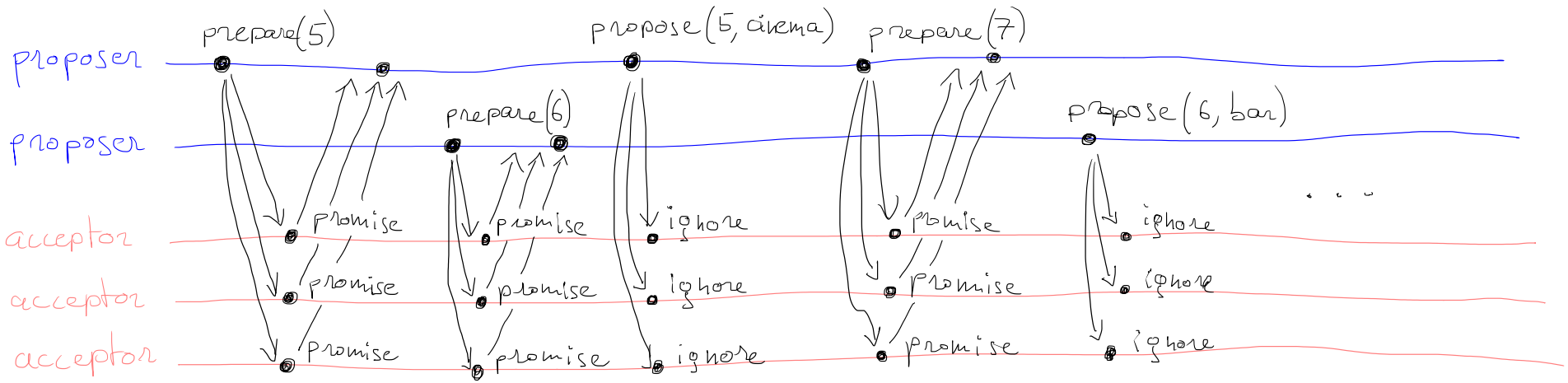
- P1: Any proposal number is unique.
- P2: Any two majority sets have at least one acceptor in common.
- P3: the value sent out in phase 2 (accept) is the value of the highest-numbered proposal of all the responses in phase 1 (prepare).

Proof of safety

- Claim: if a value v is chosen at proposal number n, any value that is sent out in phase 2 of any later proposal numbers must be also v.


proposer
proposer
acceptor
acceptor
acceptor
prepare(5)
promise
promise
prepare(4)
ignore
ignore
propose(5, cinema)
accept
accept
prepare(6)
promise(5, cinema)
promise(5, cinema)
promise
propose(6, cinema)
accept
accept
accept

INFINITE CONTENTION



Learning a chosen value

- There are some options:
  - Each acceptor, whenever it accepts a proposal, informs all the learners.
  - Acceptors informs a distinguished learner (usually the proposer) and let the distinguished learner broadcast the result.

Tunable knobs

- Acceptors have many options to response:
  - Prepare request: No/Yes
  - Accept request: No/Yes if it didn't promise not to do so
- Back off time after abandon a proposal: exponential back-off/pre-assigned values
- Should we wait for nodes to become online in each phase?