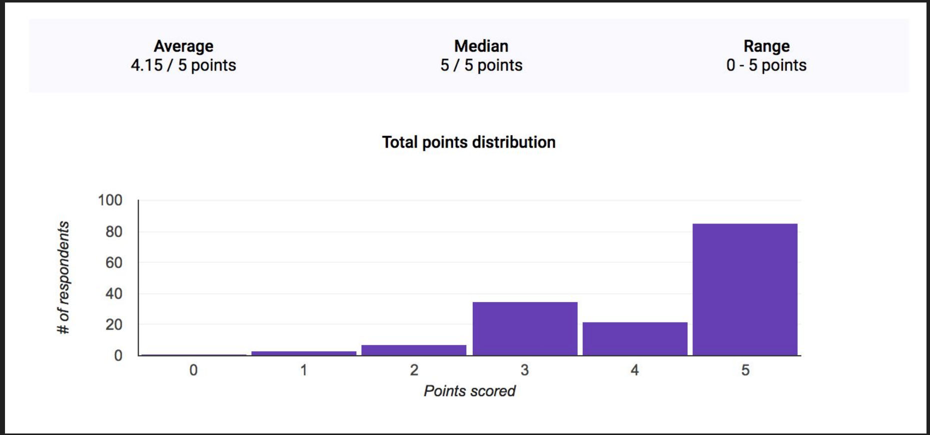
Consensus Algorithms

Administrivia

- Quiz 2 grades/solutions released



- HW2 released - due next Thursday!

Paxos

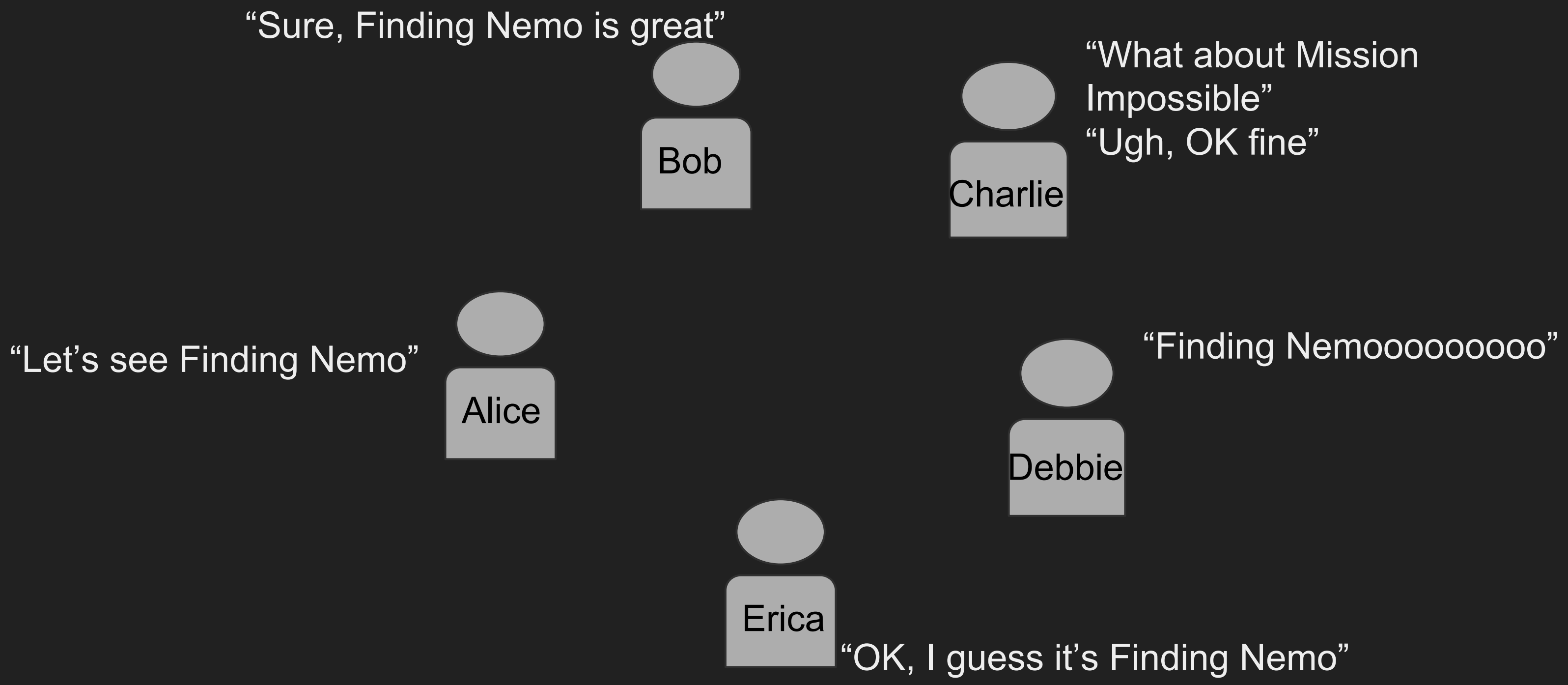
Citation: Google Tech Talks video on Paxos

Paxos

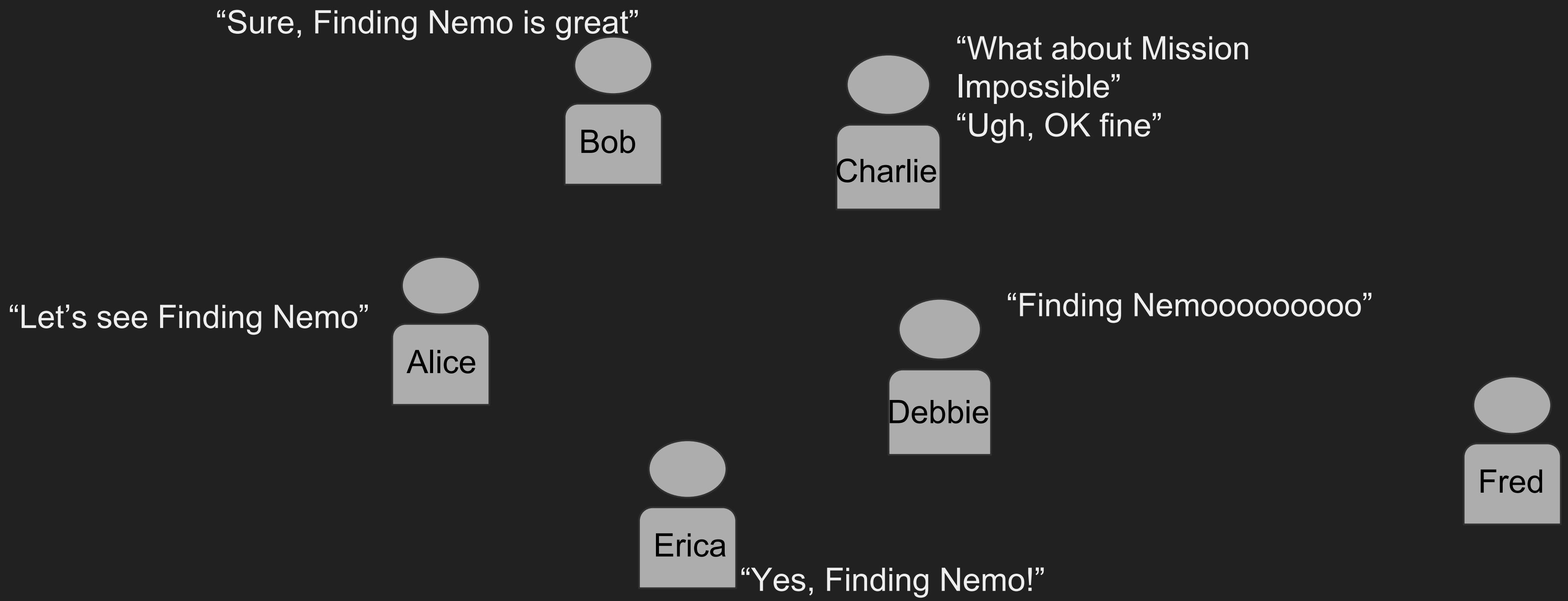
What do we mean by consensus?

- Consensus is on **one** value.
- Consensus is reached once a **majority** of participants agree.
- Once a consensus is reached, everyone can **eventually** know the result.
- Participants are happy to reach consensus on **any** result, not just the one they propose.
- Communication channels are **not** perfect (messages may be lost).

Paxos


"Sure, Finding Nemo is great"
Bob
"What about Mission Impossible"
"Ugh, OK fine"
Charlie
"Let's see Finding Nemo"
Alice
"Finding Nemooooooooo"
Debbie
Erica
"OK, I guess it's Finding Nemo"

Paxos



"Sure, Finding Nemo is great"
Bob
"What about Mission Impossible"
"Ugh, OK fine"
Charlie
"Let's see Finding Nemo"
Alice
"Finding Nemooooooooo"
Debbie
Fred
Erica
"Yes, Finding Nemo!"

Paxos

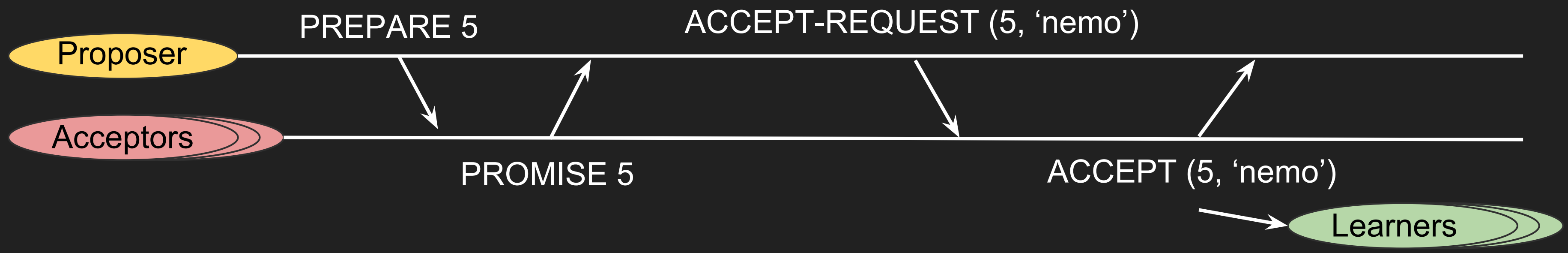
Paxos defines three different roles for the nodes in the system:

- Proposers
  - These propose values for consensus.
- Acceptors
  - These “vote” on proposals and form the majority.
- Learners
  - These record whatever the acceptors have accepted as the decision.

Decisions must be persistent. Nodes must know how many acceptors there are.

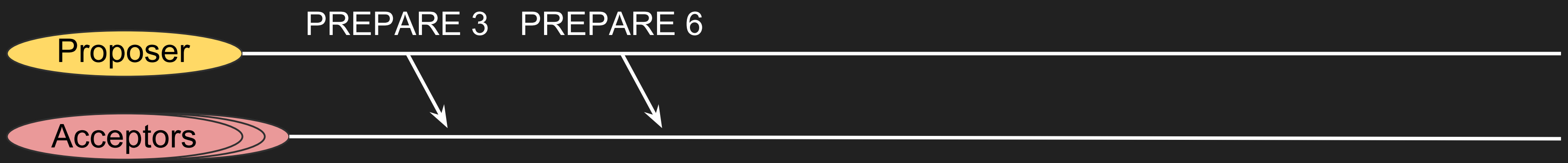
Note: we’re talking about them separately, but in practice any single machine can play any number of roles simultaneously

Paxos



- Proposer picks a proposal ID (ID_p) and sends a PREPARE ID_p message to Acceptors
  - IDp must be unique (i.e. different proposers should pick different IDs)
  - Timeout? Pick higher ID_p
- Acceptor receives PREPARE request. Did it promise to ignore requests with ID_p?
  - If yes, then ignore request
  - If no, then promise to ignore ID < ID_p (and send PROMISE ID_p in reply)
- Proposer gets PROMISE response from majority of acceptors. It sends ACCEPT_REQUEST (ID_p, *value*) to Acceptors.
  - *value* can be anything
- Acceptor receives ACCEPT_REQUST. Did it promise to ignore IDp?
  - If yes, then ignore request
  - If no, reply ACCEPT (ID_p, *value*) and also send to Learners

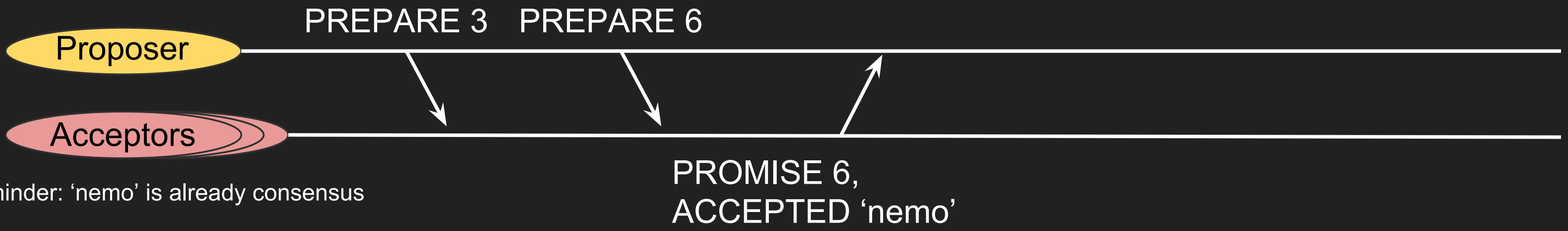
Paxos



Reminder: ‘nemo’ is already consensus

- Proposer picks a proposal ID (ID_p) and sends a PREPARE ID_p message to Acceptors
  - IDp must be unique (i.e. different proposers should pick different IDs)
  - Timeout? Pick higher ID_p
- Acceptor receives PREPARE request. Did it promise to ignore requests with ID_p?
  - If yes, then ignore request
  - If no, then promise to ignore ID < ID_p (and send PROMISE ID_p in reply)
- Proposer gets PROMISE response from majority of acceptors. It sends ACCEPT_REQUEST (ID_p, *value*) to Acceptors.
  - *value* can be anything
- Acceptor receives ACCEPT_REQUST. Did it promise to ignore IDp?
  - If yes, then ignore request
  - If no, reply ACCEPT (ID_p, *value*) and also send to Learners

Paxos

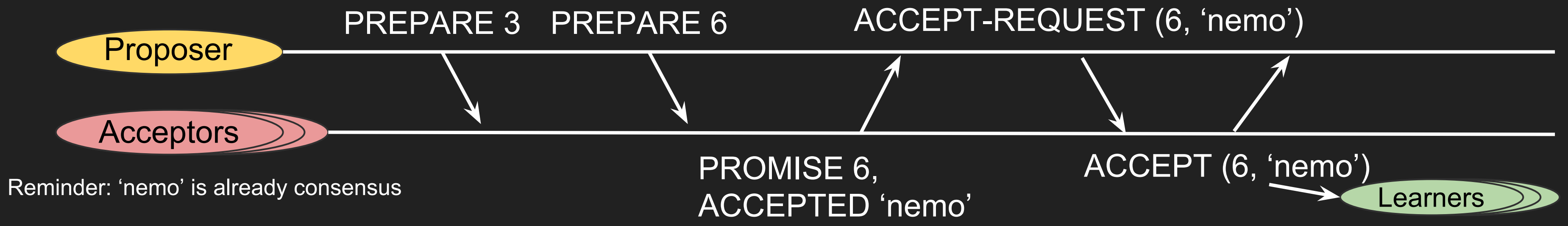


Reminder: ‘nemo’ is already consensus

- Proposer picks a proposal ID (ID_p) and sends a PREPARE ID_p message to Acceptors
  - IDp must be unique (i.e. different proposers should pick different IDs)
  - Timeout? Pick higher ID_p
- Acceptor receives PREPARE request. Did it promise to ignore requests with ID_p?
  - If yes, then ignore request
  - If no, then promise to ignore ID < ID_p. Did we already ACCEPT something?
    - If yes, send PROMISE ID_p, ACCEPTED *value*
    - If no, send PROMISE ID_p

- Proposer gets PROMISE response from majority of acceptors. It sends ACCEPT_REQUEST (ID_p, *value*) to Acceptors.
  - *value* can be anything
- Acceptor receives ACCEPT_REQUST. Did it promise to ignore IDp?
  - If yes, then ignore request
  - If no, reply ACCEPT (ID_p, *value*) and also send to Learners

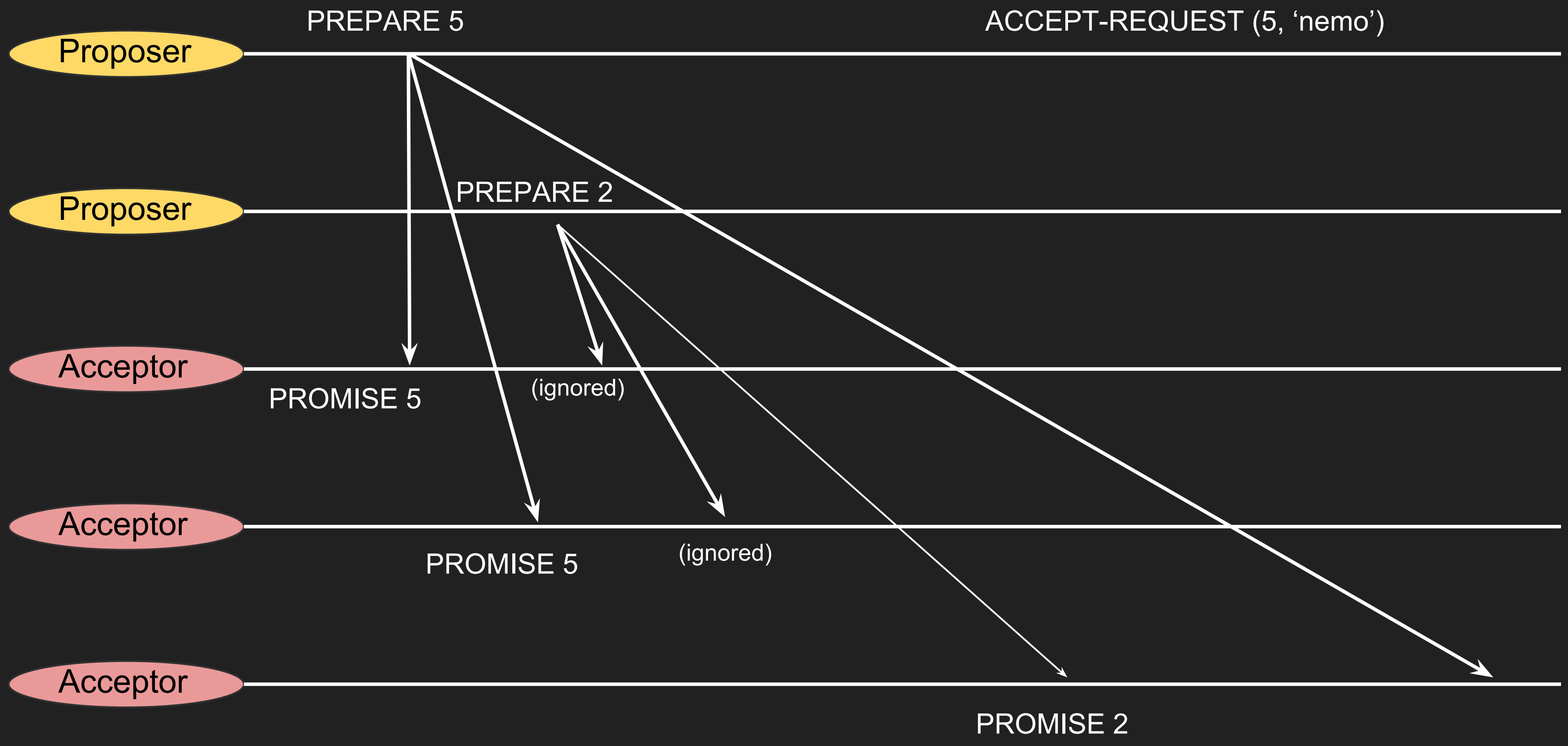
Paxos



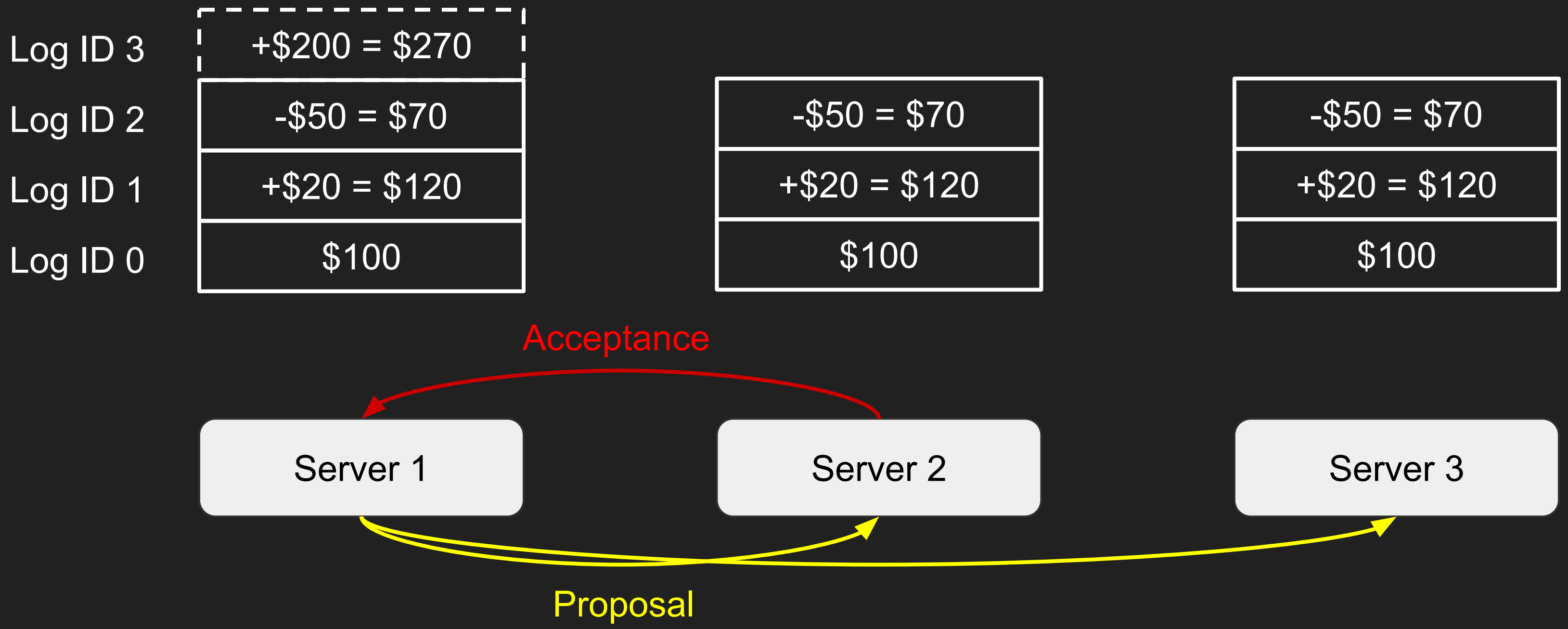
- Proposer picks a proposal ID (ID_p) and sends a PREPARE ID_p message to Acceptors
  - IDp must be unique (i.e. different proposers should pick different IDs)
  - Timeout? Pick higher ID_p
- Acceptor receives PREPARE request. Did it promise to ignore requests with ID_p?
  - If yes, then ignore request
  - If no, then promise to ignore ID < ID_p. Did we already ACCEPT something?
    - If yes, send PROMISE ID_p, ACCEPTED *value*
    - If no, send PROMISE ID_p

- Proposer gets PROMISE response from majority of acceptors. It sends ACCEPT_REQUEST (ID_p, *value*) to Acceptors. Did PROMISES come with *values*?
  - If yes, Proposer must use *value* with highest ID_p.
  - If no, *value* can be anything
- Acceptor receives ACCEPT_REQUST. Did it promise to ignore IDp?
  - If yes, then ignore request
  - If no, reply ACCEPT (ID_p, *value*) and also send to Learners

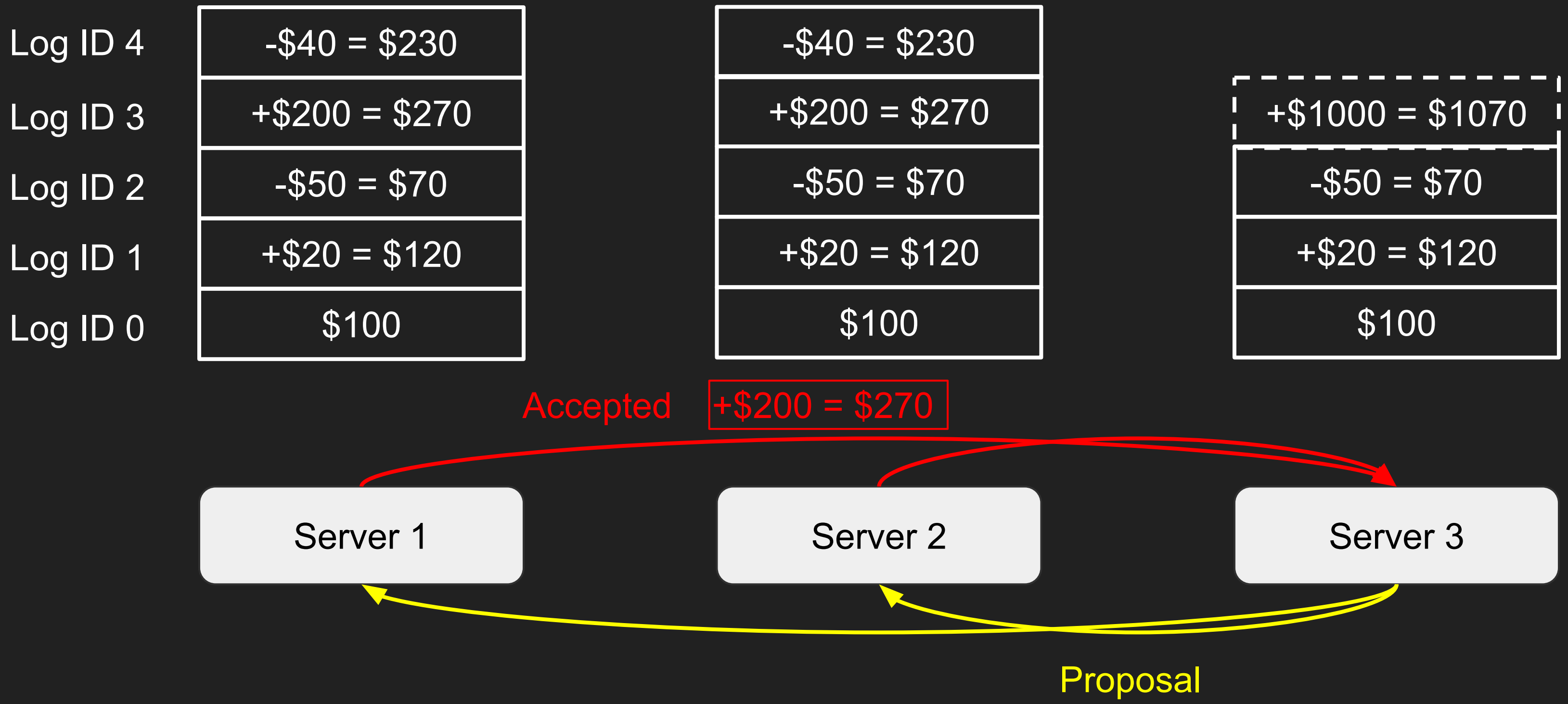
Paxos


PREPARE 5
ACCEPT-REQUEST (5, 'nemo')
Proposer
PREPARE 2
Proposer
Acceptor
PROMISE 5
(ignored)
Acceptor
PROMISE 5
(ignored)
Acceptor
PROMISE 2

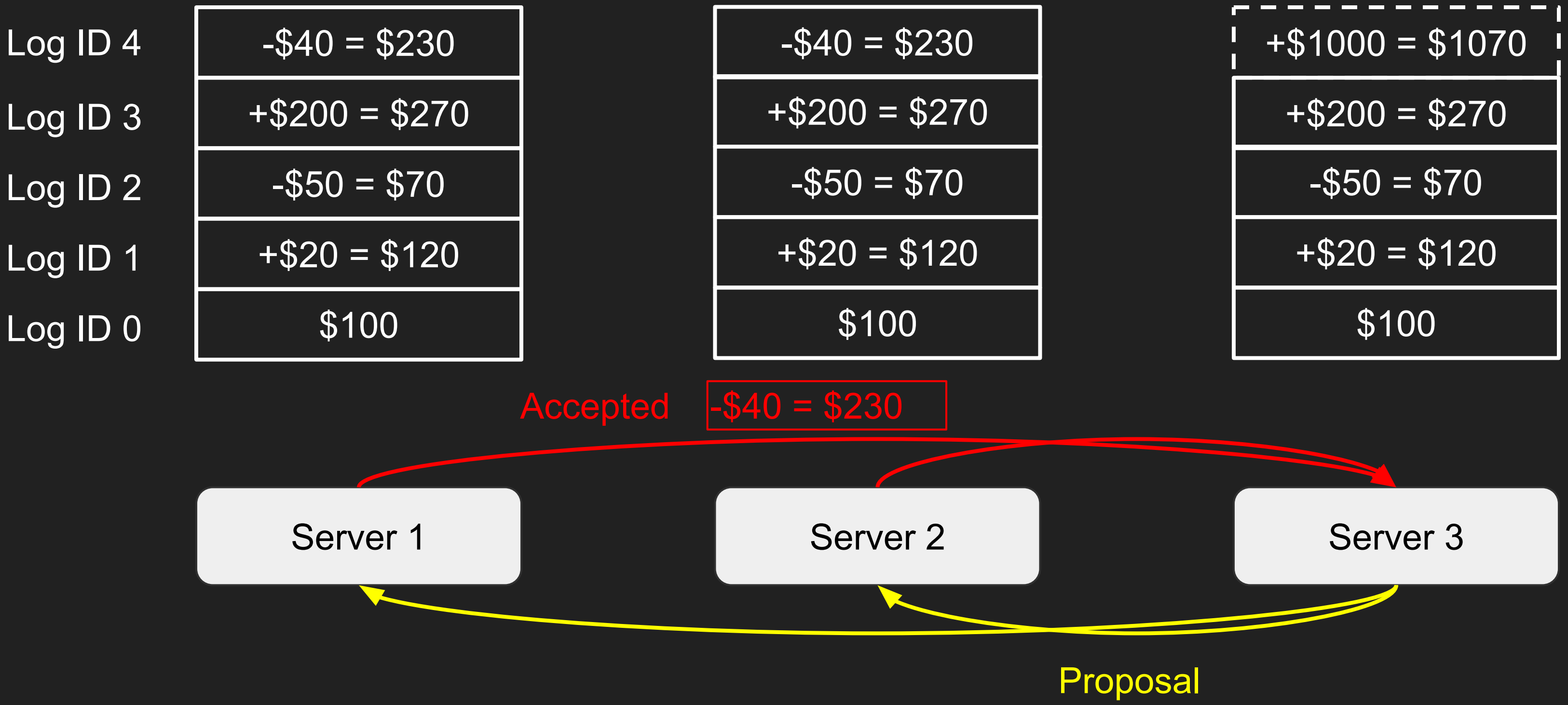
Paxos: Distributed Storage



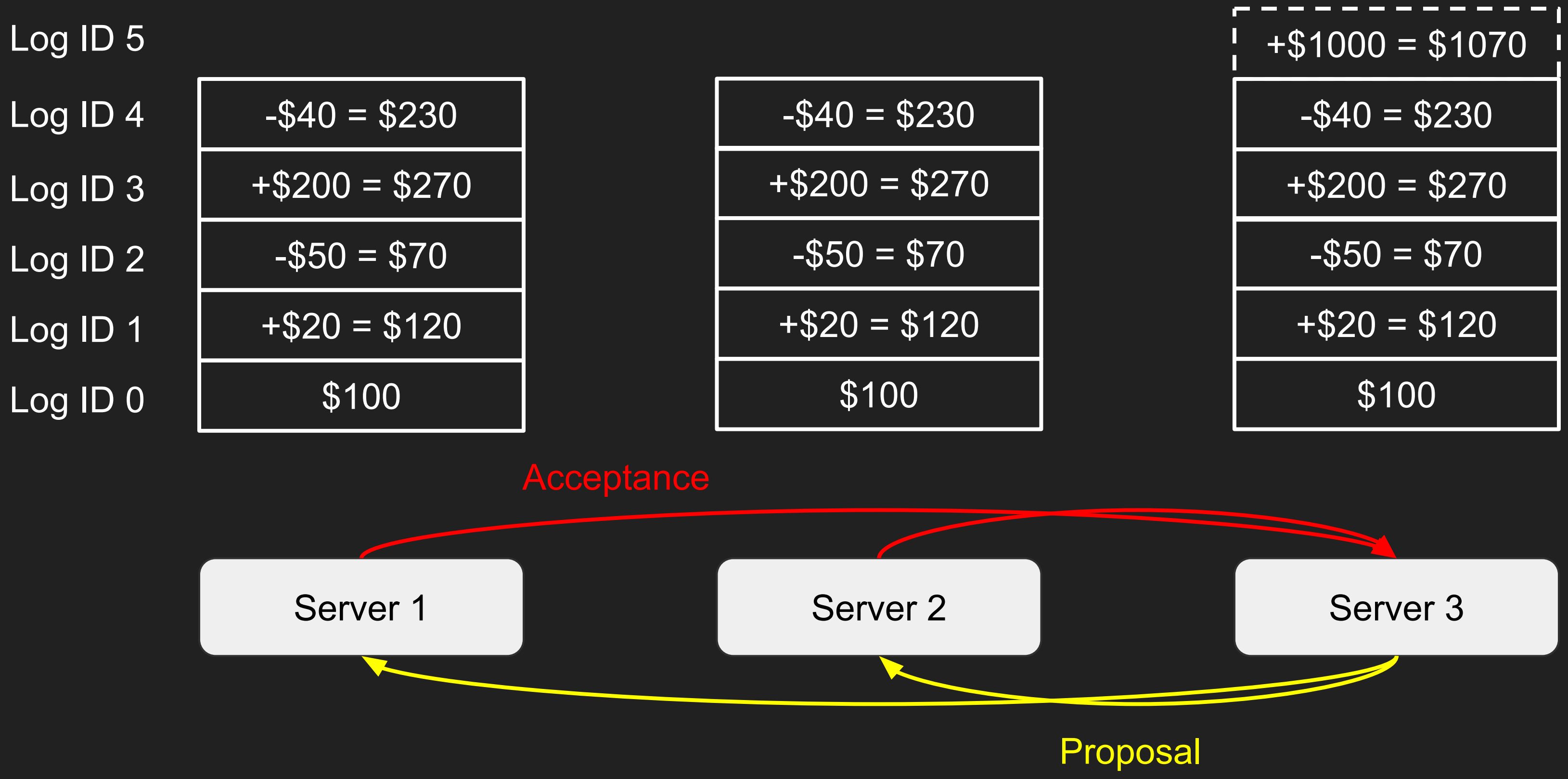
Paxos: Distributed Storage



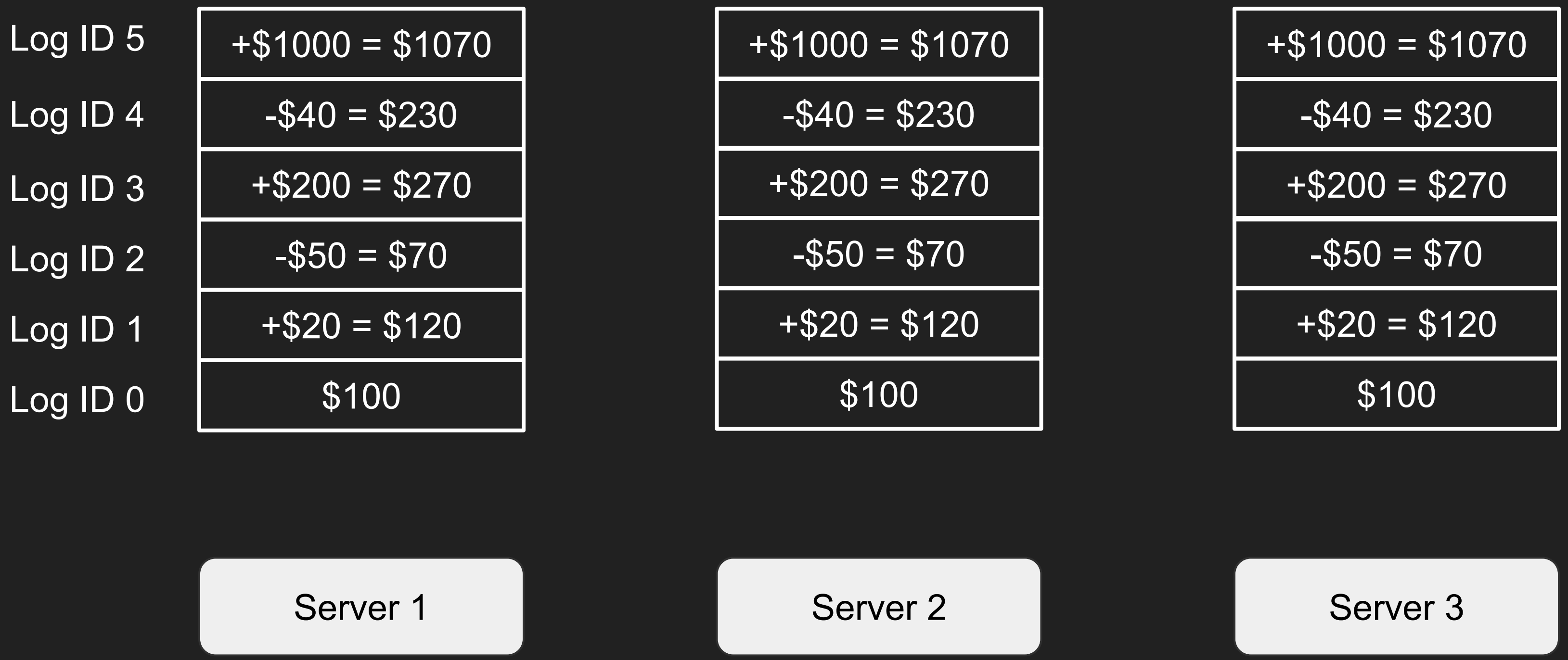
Paxos: Distributed Storage



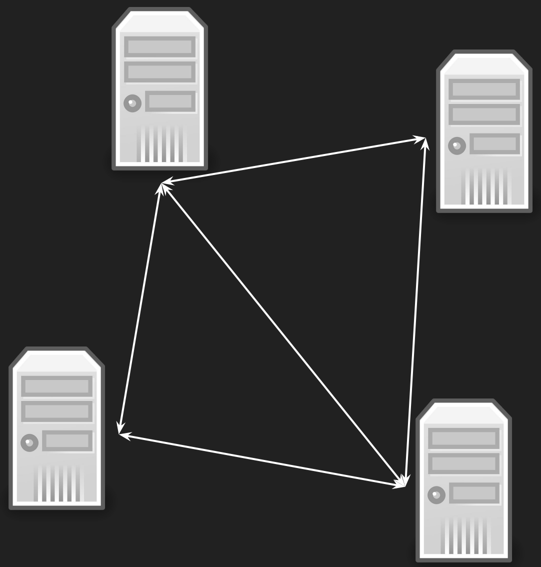
Paxos: Distributed Storage



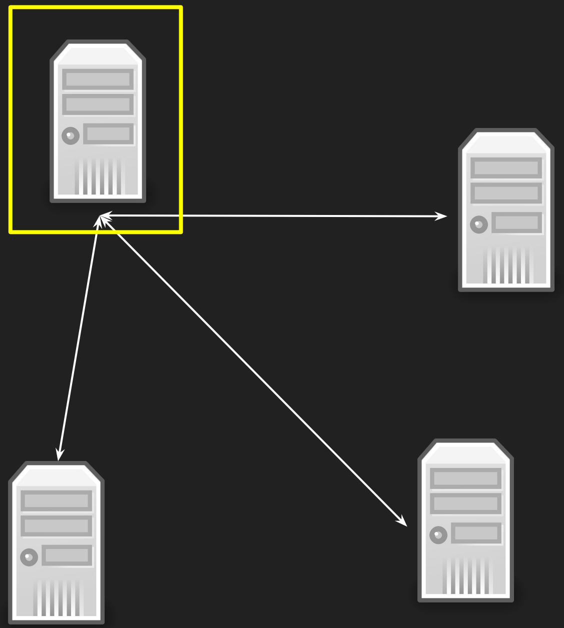
Paxos: Distributed Storage



Paxos: Master Election

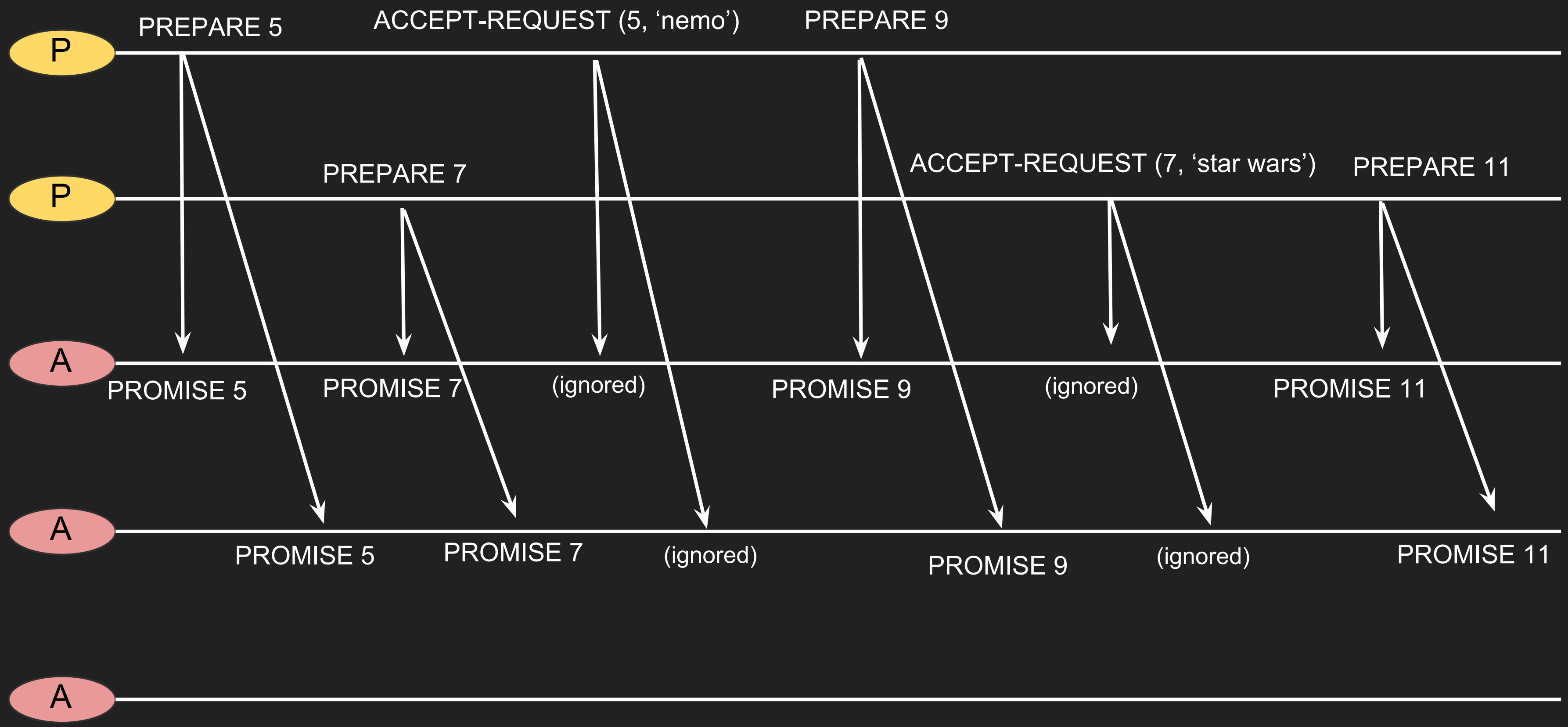


Peer-to-peer



Primary/Secondary

Paxos



P
P
A
A
A
PREPARE 5
ACCEPT-REQUEST (5, ‘nemo’)
PREPARE 9
PREPARE 7
ACCEPT-REQUEST (7, ‘star wars’)
PREPARE 11
PROMISE 5
PROMISE 7
(ignored)
PROMISE 9
(ignored)
PROMISE 11
PROMISE 5
PROMISE 7
(ignored)
PROMISE 9
(ignored)
PROMISE 11

Digression: Contention

- Contention is a general issue in concurrent algorithms.
  - Race conditions
  - Deadlock
  - Livelock
- Concurrency is HARD!

Digression: Contention (Race Condition)

- What happens if two machines run this code at once?

```
def incrementSharedValue(value_server):
  x = value_server.get_value()
  y = x + 1
  value_server.set_value(y)
```

Digression: Contention (Race Condition)

- What happens if two machines run this code at once?

```
def incrementSharedValue(value_server):
  value_server.lock()
  x = value_server.get_value()
  y = x + 1
  value_server.set_value(y)
  value_server.unlock()
```

This will block if some other machine has the lock, until they release it

Digression: Contention (Deadlock)

- What happens if two machines are calling these functions?

```
def incrementSharedValue(s1, s2):
  s1.lock()
  s2.lock()
  x = s1.get_value() + 1
  y = s2.get_value() + 1
  s1.set_value(x)
  s2.set_value(y)
  s2.unlock()
  s1.unlock()
```

```
def incrementSharedValue(s1, s2):
  s2.lock()
  s1.lock()
  x = s1.get_value() + 1
  y = s2.get_value() + 1
  s1.set_value(x)
  s2.set_value(y)
  s1.unlock()
  s2.unlock()
```

Digression: Contention (Livelock)

- Sort of like deadlock, but state is changing
  - Paxos example from earlier
  - Two people walking toward each other in a hallway
- Possible solutions
  - Exponential backoff
  - Backoff fuzzing
- Both of those solutions are generally good practice for request retries