Need to try to get a majority to accept
Informally

- Allow multiple proposals to be chosen, but guarantee that all chosen proposals have the same value.

- If proposal $N$ with value $v$ is chosen, every higher numbered proposal issued by any proposer should have value $v$.

- A proposer wanting to issue a proposal numbered $N$ must learn the highest-numbered proposal $< N$ (if any) that has been or will be accepted by a majority.
Informally

• A proposer wanting to issue a proposal numbered $N$ must learn the highest-numbered proposal $< N$ (if any) that has been or will be accepted by a majority
  – Easy to learn about values already accepted
  – Hard to predict the future

• Control the future by extracting a promise that there will not be any acceptances of proposals $< N$
Paxos – phase 1

Prepare N

Client

Proposer

Accept

Learner

Highest-numbered proposal accepted

Highest-numbered prepare request acknowledged

Written to stable store

Client

Value v'

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner

Proposer

Accept

Learner
Paxos – phase 2

propose \( N, v \)

propose \( N, v \)

value \( v' \)
Paxos – communicate agreement

- client
  - propose
    - proposer
      - acceptor
        - learner
      - learner
    - acceptor
      - learner
    - proposer
      - acceptor
        - learner
    - proposer
      - acceptor
        - learner
      - proposer
        - acceptor
          - learner
          - value $v'$
Paxos – majority learns outcome
Paxos – learning chosen value

\[
\text{prepare } N', \quad \text{value } v', \quad \text{prepare } N' \\
\text{prepare } N', \quad \text{prepare } N' \\
\text{prepare } N', \quad \text{prepare } N'
\]
Paxos – propagate chosen value
Paxos – everyone learns outcome
Example (contd.)

proposers

v

v'

v''

pro (2,v')

pro (2,v')

acceptors

(0,v_0)

(2,v'_1)

(2, _)

proposers

v

v'

v''

ACK-pre (3,v_0)

ACK-pre (3, v'_2)

acceptors

(3,v_0)

(3,v'_2)

proposers

v

v'

v''

ACK-pro (2, v'_2)

ACK-pro (2, v'_2)

acceptors

(0,v_0)

(2,v'_2)

(2,v'_2)

proposers

v

v'

v''

pro (3,v')

pro (3,v')

acceptors

(3,v_0)

(3,v'_2)

proposers

v

v'

pre (3)

pre (3)

acceptors

(0,v_0)

(2,v'_2)

proposers

v

v'

ACK-pro (3, v'_3)

ACK-pro (3, v'_3)

acceptors

(3,v'_3)

(3,v'_3)
Lamport: implementing a state machine

• How to run multiple instances of Paxos
  – Assume the existence of a distinguished proposer (leader)
  – A leader will run Paxos for a number of instances
  – The leader may crash, at which point there may be gaps in the chosen instances (1-134, 138, ..)
  – A new leader will try to fill in those slots or propose *no-op*
  – As soon as gap fills, commands can be executed

• Multi-Paxos
  – New leader: execute phase 1 for infinitely many instances
  – Acceptors can respond with reasonably short messages
  – Cost of Paxos effectively the cost of executing phase 2
Multi-Paxos

- New leader @N
- Learn accepted values for past instances

If a majority has not accepted anything for instances > I

Skip prepare phase until a propose is rejected!
Multi-Paxos

Servers play all roles

Replicas write to disk prior to sending ACK