Paxos

Seif Haridi

Single Value Uniform Consensus

- Validity
  - Only proposed values may be decided
- Uniform Agreement
  - No two processes decide different values
- Integrity
  - Each processes can decide a value at most once
- Termination
  - Every process eventually decides a value

Single Value Uniform Consensus

- (Uniform) Consensus is not solvable in the Fail-Silent model (asynchronous system model)
- Given a fixed set of deterministic processes there is no algorithm that solves consensus in the asynchronous model if one process may crash and stop
- There are some infinite executions that where processes are not able to decide on a single value
- Fischer, Lynch and Patterson FLP result

Assumptions

- Partially synchronous system
- Fail-noisy model
- Message duplication, loss, re-ordering

Importance

- Paxos is arguably the most important algorithm in distributed computing
- This presentation follows the paper "Paxos Made Simple" (Lamport, 2001)

High Level View of Paxos

- Elect a single proposer using Ω
  - Proposer imposes its proposal to everyone
  - Everyone decides

- Problem with Ω
  - Several processes might initially be proposers (contention)

High Level View of Paxos

- Elect a single proposer using Ω
  - Proposer imposes its proposal to everyone
  - Everyone decides
- Problem with Ω
  - Several processes might initially be proposers (contention)
- Solution is Abortable Consensus
  - Processes attempt to impose their proposals
  - Might abort if there is contention (safety) (multiple proposers)
  - Ω ensures eventually 1 proposer succeeds (liveness)

PAXOS ALGORITHM

Terminology

- Proposers
  - Will attempt imposing their proposal to set of acceptors
- Acceptors
  - May accept values issued by proposers
- Learners
  - Will decide depending on acceptors acceptances

- Each process plays all 3 roles in classic setting

Naïve Approach

- Centralized solution
  - Proposer sends value to a central acceptor
  - Acceptor decides first value it gets
- Problem
  - Acceptor is a single-point of failure

Abortable Consensus

- Decentralizes, i.e. proposers talks to set of acceptors

- Tolerate failures, i.e. acceptors might fail (needs only a majority of acceptors surviving)

- Proposers might fail to impose its proposal (aborts)

Decentralization & Fault-tolerance

- Quorum approach
  - Each proposer tries to impose its value v on the set of acceptors
  - If majority of acceptors accept v, then v is chosen
  - Learners try to decide the chosen value

Ballot (round) Array (table)

- Describes the state of the acceptors at various rounds
- Each raw describes one round
- Each acceptor's state of a_i initially $\perp$

Round	a_1	a_2	a_3
n = 5			
...			
n=2			
n=1			
n=0	$\perp$	$\perp$	$\perp$

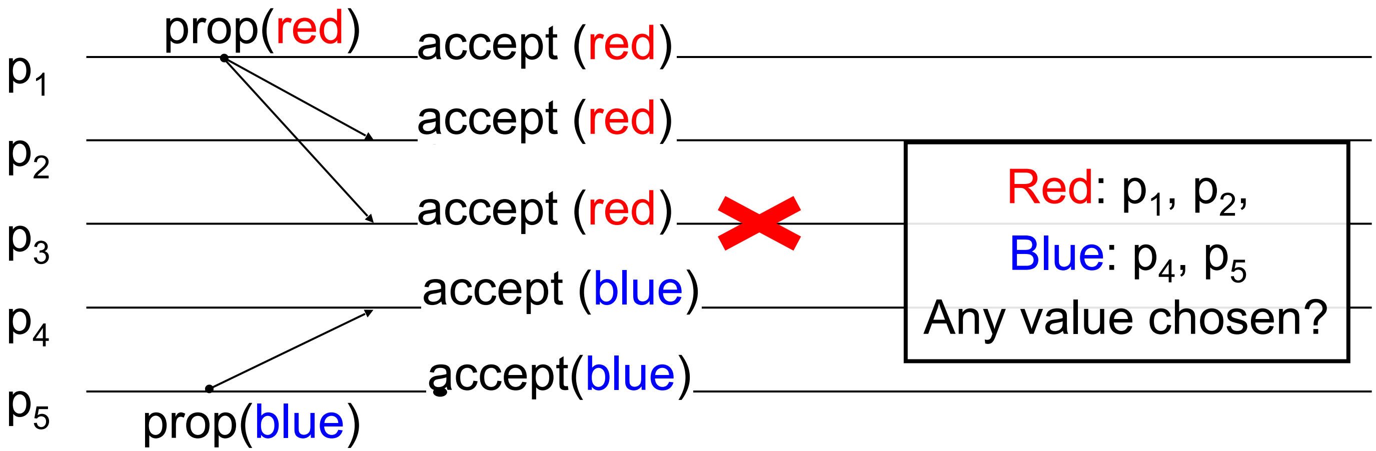
When to accept

- Ideally, there will be a single proposer
  - Should at least provide obstruction-free progress
    - Obstruction-free = if a single proposer executes without interference (contention) it makes progress

- Suggested invariant
  - P1. An acceptor accepts first proposal it receives

Attempt

- P1. An acceptor *accepts* first proposal it receives
- Problem
  - Impossible to later tell what was chosen
  - Forced to allow restarting! Let acceptors change their minds!



Ballot (round) Array (table)

- Two proposers p1 and p2 that propose red and blue
- But a_3 crashes

Round	a_1	a_2	a_3	a_4	a_4
n = 5					
...					
n=2					
n=1	red	red	red	blue	blue
n=0	$\perp$			$\perp$	$\perp$

Ballot (round) Array (table)

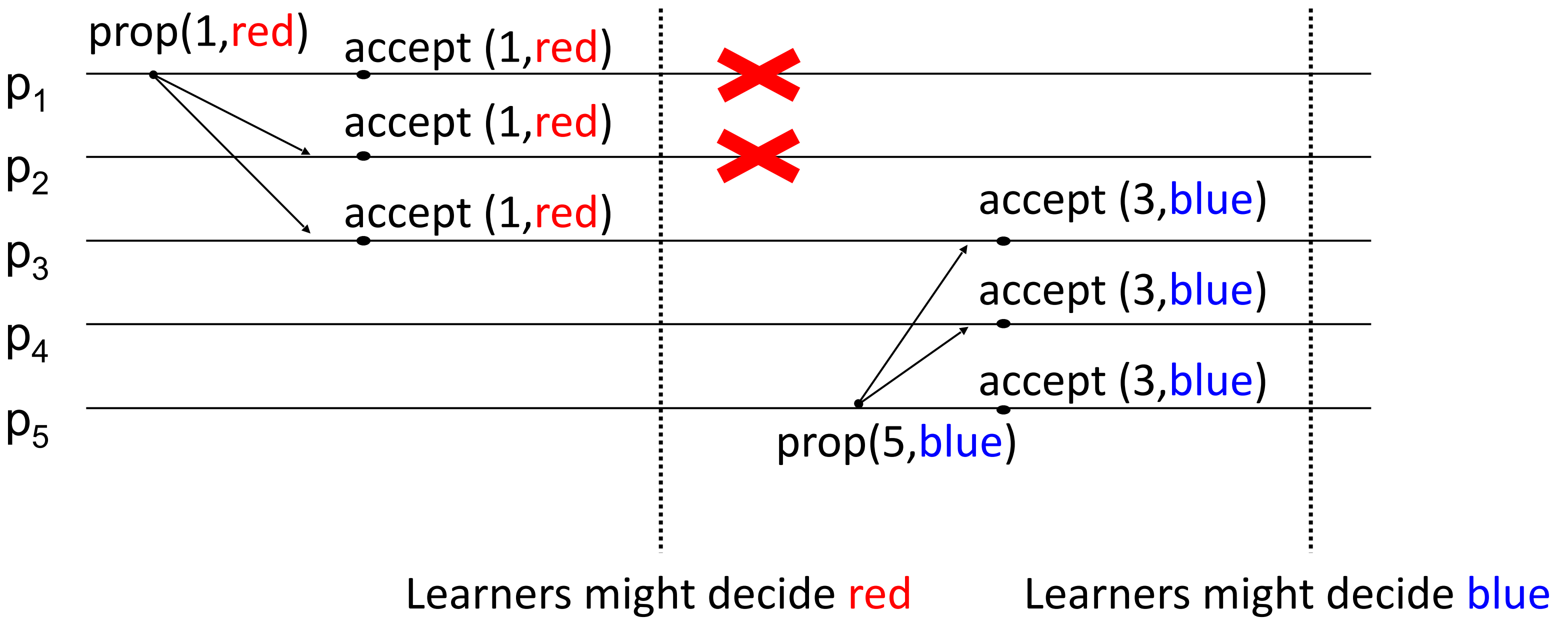
- Two proposers p1 and p2 that propose red and blue
- But a_3 crashes

Round	a_1	a_2	a_3	a_4	a_4
n = 5					
...					
n=2					
n=1	red	red		blue	blue
n=0	⊥	⊥	⊥	⊥	⊥

Enabling Restarting

- Proposer can try to propose again
  - Distinguish proposals with unique sequence number
  - Often called ballot number
  - Monotonically increasing
- Implementation with n nodes
  - process 1 uses seq: 1, n+1, 2n+1, 3n+1, …
  - process 2 uses seq: 2, n+2, 2n+2, 3n+2, …
  - process 3 uses seq: 3, n+3, 2n+3, 3n+3, …
- or…
  - Pair of values: (local clock or logical clock, local identifier)
  - Lexicographic order: if clock collides, choose highest pid

Problem with restart



Ballot (round) Array (table)

- p1 proposes (1,red) and p2 proposes (3, blue)
- But a_1 and a_1 crashed

Round	a_1	a_2	a_3	a_4	a_4
n = 5					
n = 4					
n = 3			blue	blue	blue
n=2	red	red	red	$\perp$	$\perp$
n=1	red	red	red	$\perp$	$\perp$
n=0	$\perp$			$\perp$	$\perp$

Ensuring Agreement

- Problem (previous slide):
  - If restarting allowed,
    - Majority may first accept red
    - Majority may later accept blue
- Solve it by enforcing:
  - P2. If proposal (n,v) is chosen, every higher numbered proposal chosen has value v

Birds-eye View

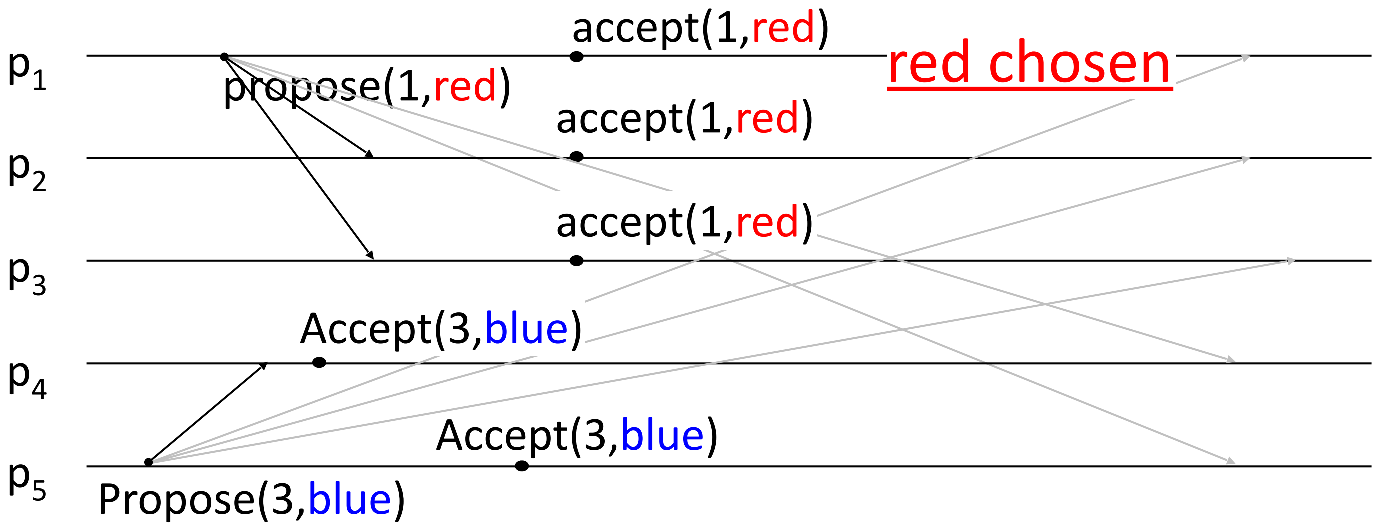
- Abortable Consensus in a nutshell
  - P1. An acceptor accepts first proposal it receives
  - P2. If v is chosen, every higher proposal chosen has value v
- Handwaving
  - P1 ensures obstruction-free progress and validity
  - P2 ensures agreement
  - Integrity trivial to implement
    - Remember if chosen before, at most choose once

Attempt

- P2. If v is chosen, every higher proposal chosen has value v
  - **How to implement it?**
- P2a. If v is chosen, every higher proposal accepted has value v
- Lemma
  - P2a => P2

Problem

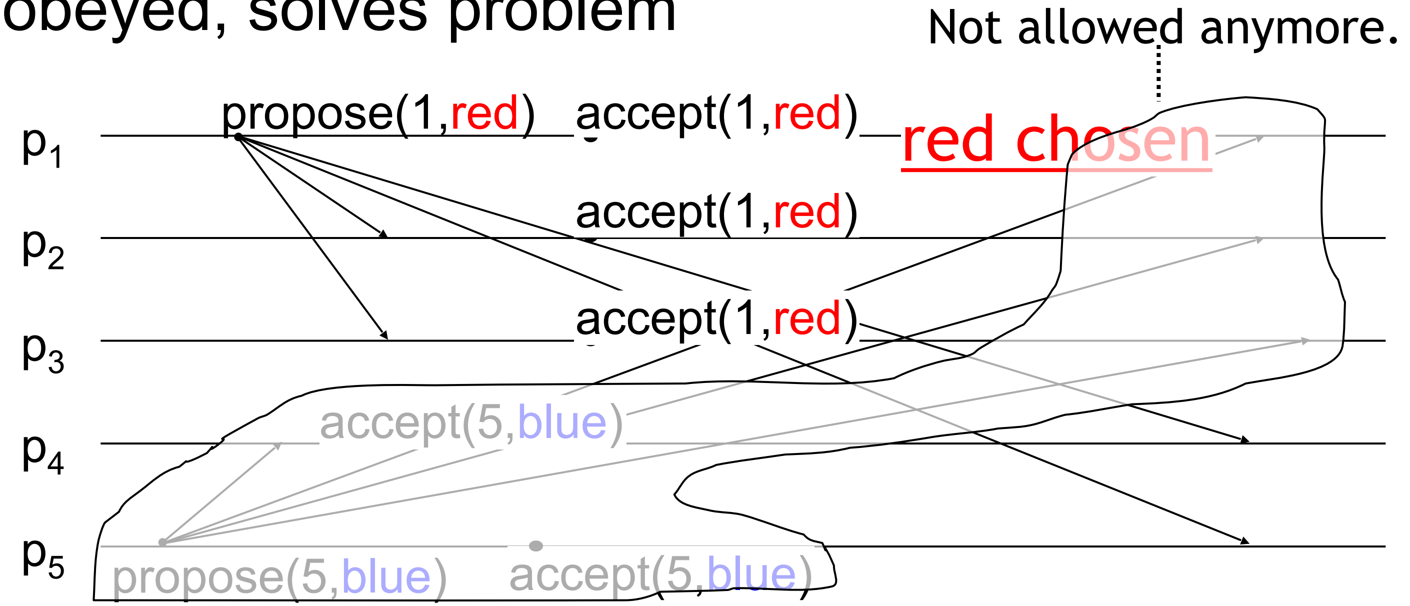
- Recall
  - P1. An acceptor accepts first proposal it receives
  - P2a. If v is chosen, every higher proposal accepted has value v
- Problem: we cannot prevent an acceptor from accepting higher value proposal



Solution

- Strengthen P2a
  - P2b. If v is chosen, every higher proposal issued has value v

- If obeyed, solves problem



Ballot (round) Array (table)

- p1 proposes (1,red) and p2 proposes (3, blue)
- But a_2 and a_3 crashed before p2 proposes (3, blue)

Round	a_1	a_2	a_3	a_4	a_4
n = 5					
n = 4					
n = 3			red	⊥	⊥
n=2	red	red	red	⊥	⊥
n=1	red	red	red	⊥	⊥
n=0	⊥			⊥	⊥

Ballot (round) Array (table)

- p1 proposes (1,red) and p2 proposes (3, blue)
- At round 3 p2 **has to issue** (1,red)

Round	a_1	a_2	a_3	a_4	a_4
n = 5					
n = 4					
n = 3			red	red	red
n=2	red	red	red	$\perp$	$\perp$
n=1	red	red	red	$\perp$	$\perp$
n=0	$\perp$			$\perp$	$\perp$

P2 Preserved

- P2. If v is chosen, every higher proposal chosen has value v
- P2a. If v is chosen, every higher proposal accepted has value v
- P2b. If v is chosen, every higher proposal issued has value v

- Lemma
  - P2b => P2a
- Recall P2a => P2.
  - Thus P2b => P2

Main Lemma

- P2c. If any proposal (n,v) is issued, there is a majority set S of acceptors such that either
  - (a) no one in S has accepted any proposal numbered less than n
  - (b) v is the value of the highest proposal among all proposals less than n accepted by acceptors in S
- Lemma: P2c => P2b

Main lemma

- (a) no one in S has accepted any proposal number > 3
- p2 issues (3, blue) at round 3

Round	a_1	a_2	a_3	a_4	a_4
n = 5					
n = 4					
n = 3	red	red	blue	blue	blue
n=2	red	red	$\perp$	$\perp$	$\perp$
n=1	red	red	$\perp$	$\perp$	$\perp$
n=0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

Main lemma

- (b) v is the value of the highest proposal among all proposals less than n accepted by acceptors in S
- red is chosen at round 3, no proposer at round 4
- Proposer at round 5 will always get red querying any majority

Round	a_1	a_2	a_3	a_4	a_4
n = 5					
n = 4					
n = 3	red	red	red	?	?
n=2	red	red	?	?	?
n=1	red	red	⊥	⊥	⊥
n=0	⊥	⊥	⊥	⊥	⊥

Main lemma

- (b) v is the value of the highest proposal among all proposals less than n accepted by acceptors in S
- red is chosen at round 3, no proposer at round 4
- Proposer at round 5 will always get red querying any majority

Round	a_1	a_2	a_3	a_4	a_4
n = 5		red	red	red	
n = 4					
n = 3	red	red	red	?	?
n=2	red	red	?	?	?
n=1	red	red	$\perp$	$\perp$	$\perp$
n=0	$\perp$	$\perp$	$\perp$	$\perp$	$\perp$

How to implement P2c

- A proposer at round n needs a query phase to get the value of highest round number + a promise that the state of S does not change until round n

Round	a_1	a_2	a_3	a_4	a_4
n = 5					
n = 4					
n = 3	red	red	red	?	?
n=2	red	red	?	?	?
n=1	red	red	⊥	⊥	⊥
n=0	⊥	⊥	⊥	⊥	⊥

How to implement P2c

- A proposer issues prop(n, v)
- Guarantee?
  - v is the value of the highest proposal among all proposals less than n accepted by acceptors in S
- Need a prepare(n) phase Before issuing prop(n, v)
  - Extract a promise from a majority of acceptors not to accept a proposal less than n
  - Acceptor sends back its highest numbered accepted value

Abortable Consensus

Proposer

- Pick unique sequence n, send prepare(n) to all acceptors

3) Proposer upon majority S of promises:
   - Pick value v of highest proposal number in S, or if none available pick v freely
   - Issue accept(n,v) to all acceptors

5) Proposer upon majority S of responses:
   - If got majority of acks
   decide(v) and broadcast decide(v);
   - Otherwise abort

Acceptors

2) Upon prepare(n):
   - Promise not accepting proposals numbered less than n
   - Send highest numbered proposal accepted with number less than n (promise)

5) Upon accept(n,v):
   - If not responded to prepare m>n, accept proposal (ack); otherwise reject (nack)

abortable consensus satisfies:

P2c. If (n,v) is issued, there is a majority of acceptors S such that:

a) no one in S has accepted any proposal numbered “<“ n, OR
b) v is value of highest proposal among all proposals “<“ n accepted by acceptors in S

Paxos Correctness

- P2b. If v is chosen, every higher proposal issued has value v
- P2c. If any prop (n,v) is issued, there is a set S of a majority of acceptors s.t. either
  - (a) no one in S has accepted any proposal numbered less than n
  - (b) v is the value of the highest proposal among all proposals less than n accepted by acceptors in S

- Lemma: P2c => P2b
  - Proof map:
    - Prove lemma by assuming P2c, prove P2b follows
      - Prove P2b follows by assuming v is chosen, prove every higher proposal issued has value v
- Thus: if P2c is true, and prop (n,v) chosen
  - Show by induction every higher proposal issued has value v

- P2b. If v is chosen, every higher proposal issued has value v

- P2c. If any prop (n,v) is issued, there is a set S of a majority of acceptors s.t. either
  - (a) no one in S has accepted any proposal numbered less than n
  - (b) v is the value of the highest proposal among all proposals less than n accepted by acceptors in S

- Thus: P2c is true, and prop (n,v) chosen
  - Show by induction on (on prop number) every higher proposal issued has value v
    - Need to show by induction that

      all proposals (m,u), where m≥n, have value u=v

Round	a_1	a_2	a_3
5			
4			
3			
2	v	v	
1	w	⊥	⊥
0	⊥	⊥	⊥

- P2b. If v is chosen, every higher proposal issued has value v
- P2c. If any prop (n,v) is issued, there is a set S of a majority of acceptors s.t. either
  - (a) no one in S has accepted any proposal numbered less than n
  - (b) v is the value of the highest proposal among all proposals less than n accepted by acceptors in S

- Thus: P2c is true, and prop (n,v) chosen
  - Show by induction that all proposals (m,u), where m≥n, have value u=v
  - Induction base
    - Inspect proposal (n,u).
    - Since (n,v) chosen & proposals are unique, u=v

Round	a_1	a_2	a_3
5			
4			
3			
2	v	v	
1	w	⊥	⊥
0	⊥	⊥	⊥

- Induction step
  - Assume proposals n, n+1, n+2,…, m have value v (ind.hypothesis)
    - Show proposal (m+1,u) has u=v
  - P2c implies proposal (m+1,u) has a majority S that either
    - a) no one in S has accepted any proposal numbered less than m+1
    - b) u is the value of the highest proposal among all proposals less than m+1 accepted by acceptors in S
      - a) cannot be, as (n,v) accepted by a majority overlapping with S
      - b) must be true
  - Hence, u is the value of the highest proposal among all proposals less than m+1 accepted by acceptors in S
  - By the induction hypothesis, all proposals n,…,m have value v. Majority of prop m+1 intersects with majority of prop n, thus u=v

- Induction step
  - Assume proposals n, n+1, n+2,…, m have value v (ind.hypothesis)
    - Show proposal (m+1,u) has u=v
  - u is the value of the highest proposal among all proposals less than m+1 accepted by acceptors in S
  - By the induction hypothesis, all proposals n,…,m have value v. Majority of prop m+1 intersects with majority of prop n, thus u=v

Round	a_1	a_2	a_3
5			
4			v
3		v	
2	v	v	
1	w	$\perp$	$\perp$
0	$\perp$	$\perp$	$\perp$

Agreement Satisfied

- This algorithm satisfies P2c
  - accept(n,v) only issued if a majority S responded to prepare(n), s.t. for each p_i in S:

    a) either: p_i hadn't accepted any prop less than n, or

    b) v is value of highest proposal less than n accepted by p_i

  - By their promise, a) and b) will not change

- prepare(n) often called read(n)
- accept(n,v) often called write(n,v)

Agreement

- P2c. If (n,v) is issued, there is a majority of acceptors S s.t.
  - a) no one in S has accepted any proposal numbered less than n, or
  - b) v is the value of the highest proposal among all proposals less than n accepted by acceptors in S
- P2. If (n,v) is chosen, every higher proposal chosen has value v
- We proved that if P2c is satisfied, then P2 is satisfied
  - P2c => P2
- Thus the algorithm satisfies agreement (safety)

Obstruction Freedom and Validity

- P1. An acceptor accepts first "proposal" it receives
- P1 is satisfied because we accept
  - if prepare(n) & accept(n,v) received first
- Thus the algorithm satisfies obstruction-free progress (liveness)

Getting Familiar with Paxos

Abortable Consensus

Proposer

1) Pick unique sequence n, send prepare(n) to all acceptors

3) Proposer upon majority S of promises:
- Pick value v of highest proposal number in S, or if none available pick v freely
- Issue accept(n,v) to all acceptors

5) Proposer upon majority S of responses:
- If got majority of acks

decide(v) and broadcast decide(v);
- Otherwise abort

Acceptors

2) Upon prepare(n):
- Promise not accepting proposals numbered less than n
- Send highest numbered proposal accepted with number less than n (promise)

4) Upon accept(n,v):
- If not responded to prepare m>n, accept proposal (ack); otherwise reject (nack)

Message loss and failures

- Many sources of abort
  - Contention (multiple proposals competing)
  - Message loss (e.g. not getting an ack)
  - Process failure (e.g. proposer dies)

- So Proposers try Abortable Consensus again…
  - Prepare(5), Accept(5,v), prepare(15), …
  - Eventually the Paxos should terminate (FLP85?)

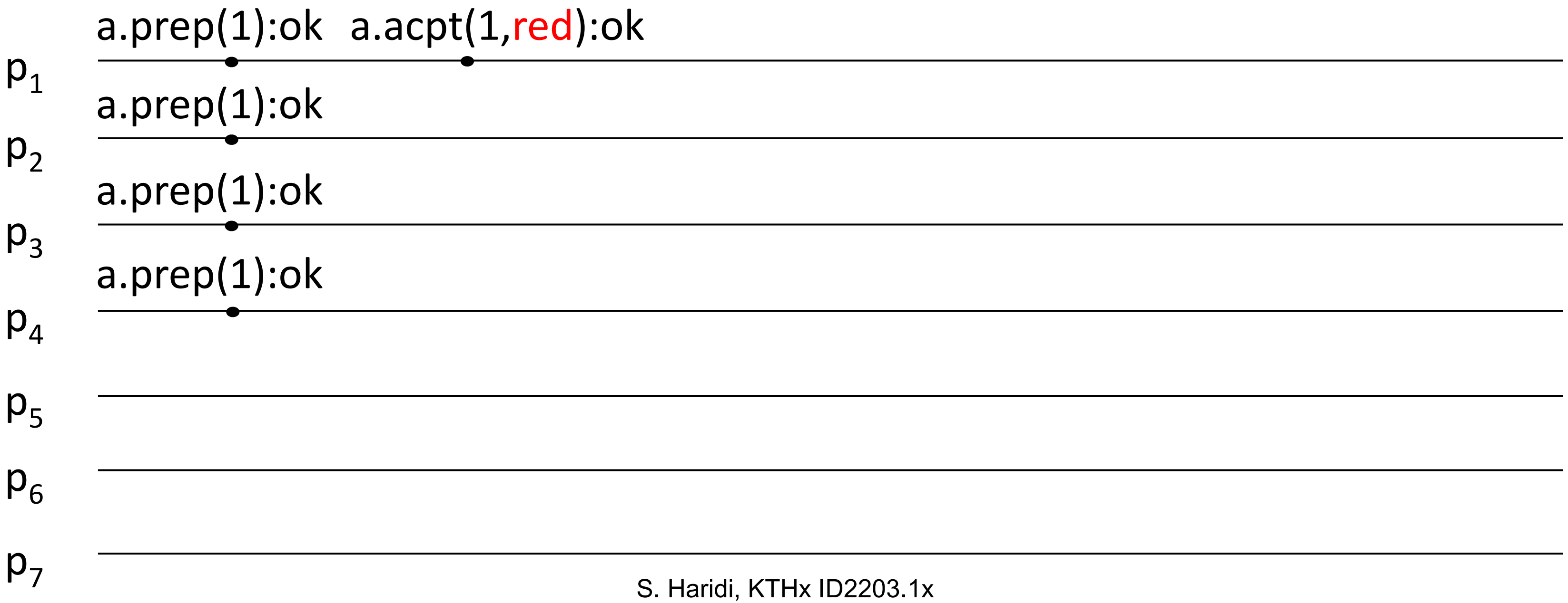
FLP ghost

p_1	a.prep(1):ok	b.prep(3):ok	a.acpt(1,v):fail	a.prep(4):ok	b.acpt(3,v):fail
p_2	a.prep(1):ok	b.prep(3):ok	a.acpt(1,v):fail	a.prep(4):ok	b.acpt(3,v):fail
p_3	a.prep(1):ok	b.prep(3):ok	a.acpt(1,v):fail	a.prep(4):ok	b.acpt(3,v):fail

- proposers a and b forever racing…
  - Eventual leader election (Ω) ensures liveness
  - Eventually only one proposer => termination

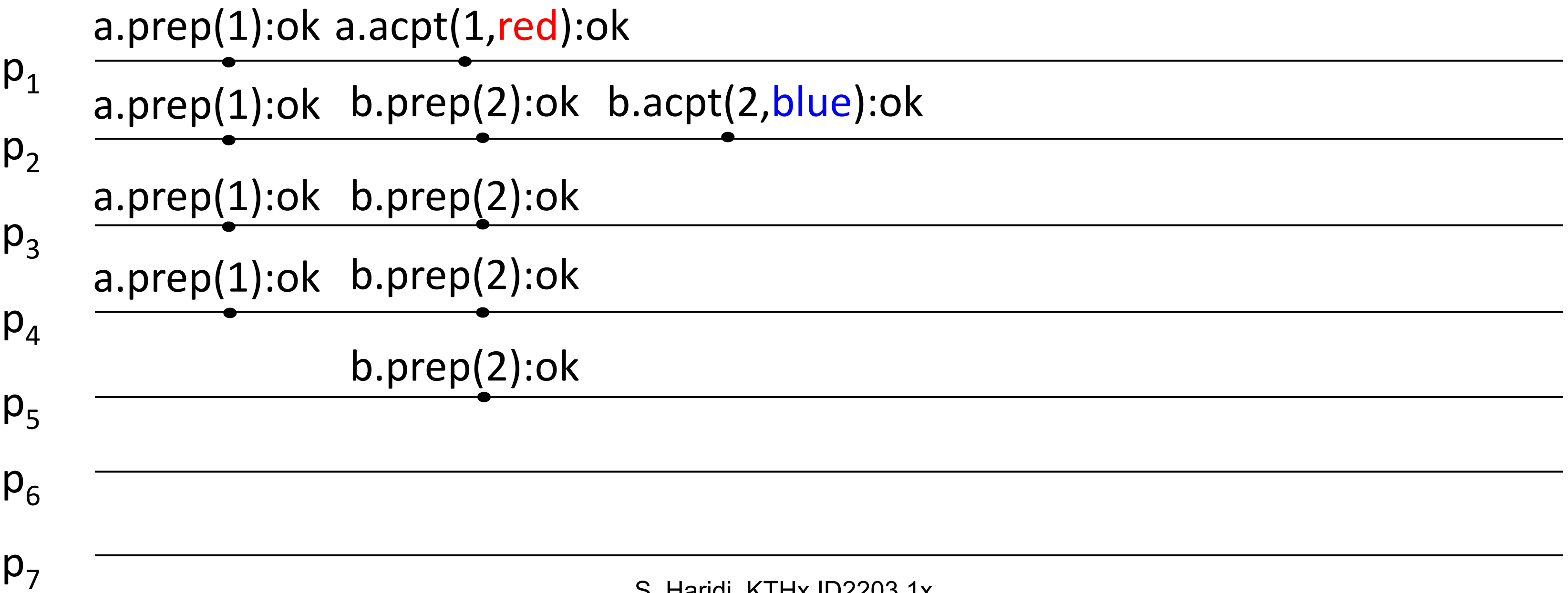
Familiarizing with Paxos (1/4)

- Different processes accept different values , same process accepts different values
- Assume 4 proposers {a,b,c,d}, 7 acceptors $\{p_1,\ldots,p_7\}$



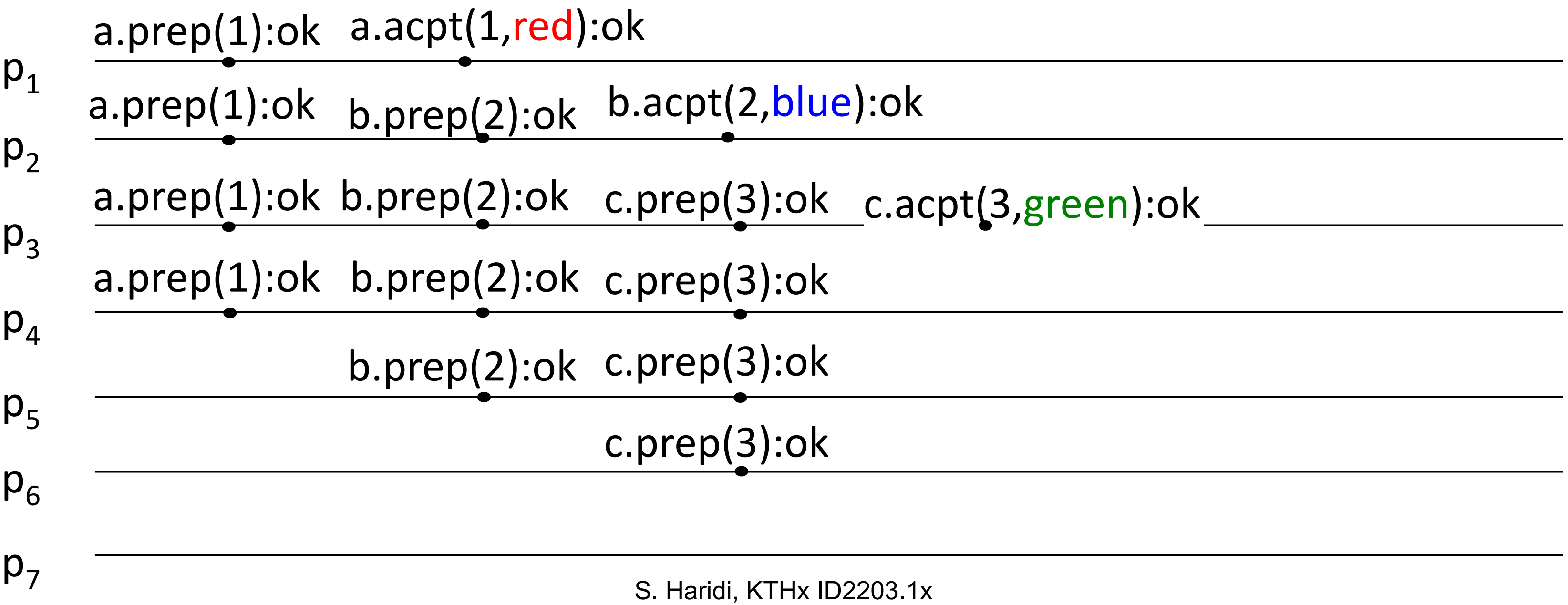
Familiarizing with Paxos (2/4)

- Different nodes accept different values , same node accepts different values
- Assume 4 proposers {a,b,c,d}, 7 acceptors {p_1,...,p_7}



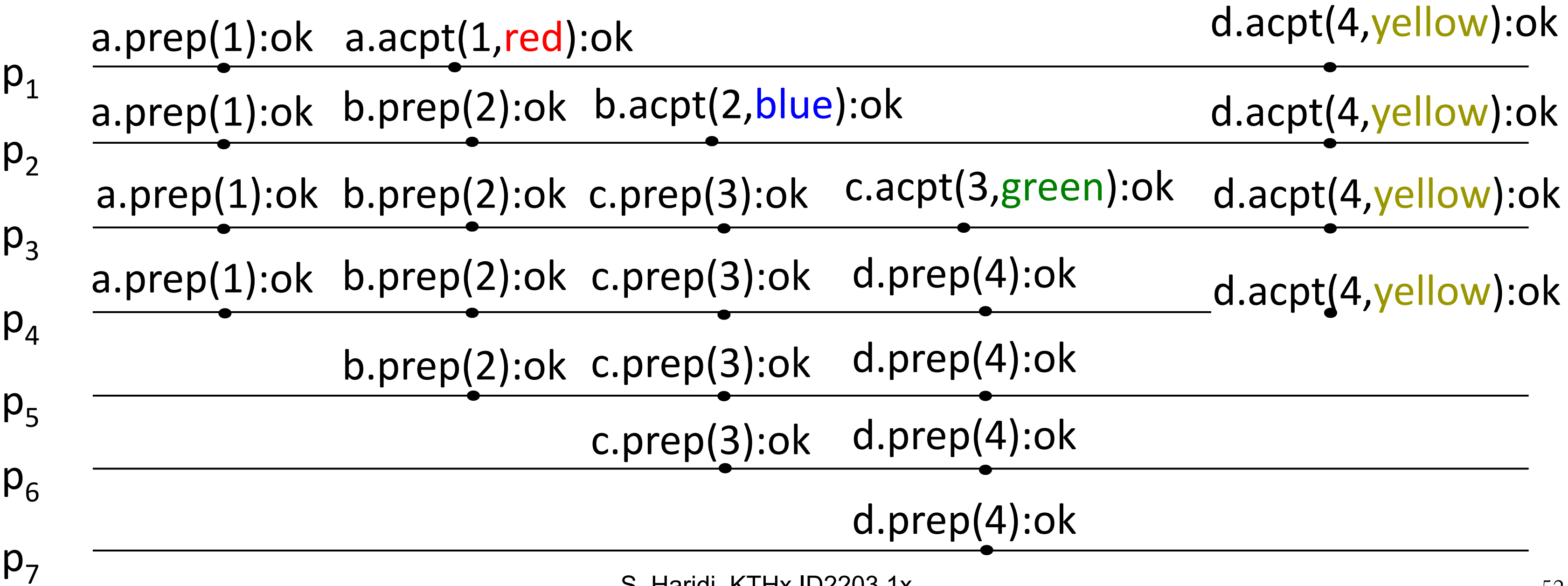
Familiarizing with Paxos (3/4)

- Different nodes accept different values , same node accepts different values
- Assume 4 proposers {a,b,c,d}, 7 acceptors {p_1,...,p_7}



Familiarizing with Paxos (4/4)

- Different nodes accept different values , same node accepts different values
- Assume 4 proposers {a,b,c,d}, 7 acceptors {p_1,...,p_7}



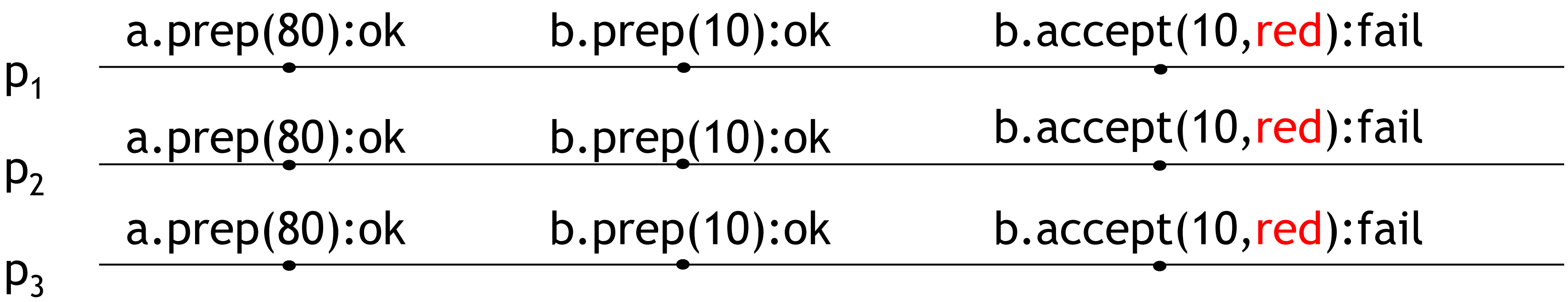
Optimizations

Paxos (AC) in a nutshell

- Necessary
  - Reject accept(n,v) if answered prepare(m) : m>n
    - i.e. prepare extracts promise to reject lower accept

Possible scenario #1

- Caveat
  - Proposers {a,b,c}, acceptors $\{p_1,p_2,p_3\}$



- accept(10) will be rejected, why answer prepare(10)?
- No point answering prepare(n) if accept(n,v) will be rejected

Summary of Optimizations

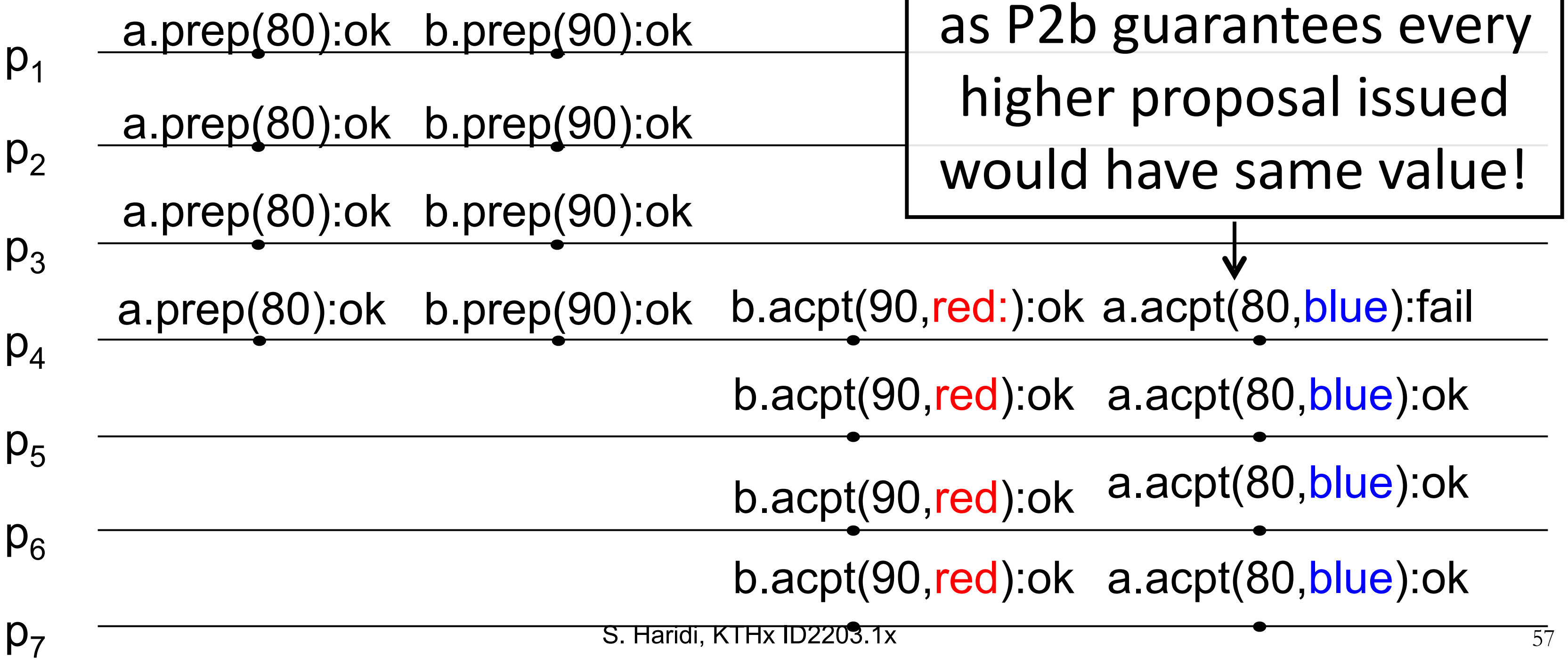
- Necessary
  - Reject accept(n,v) if answered prepare(m) : m>n
    - i.e. prepare extracts promise to reject lower accept

- Optimizations
  - a) Reject prepare(n) if answered prepare(m) : m>n
    - i.e. prepare extracts promise to reject lower prepare

Possible scenario #2

- Caveat

accept(80,blue) can anyway not get majority, as P2b guarantees every higher proposal issued would have same value!

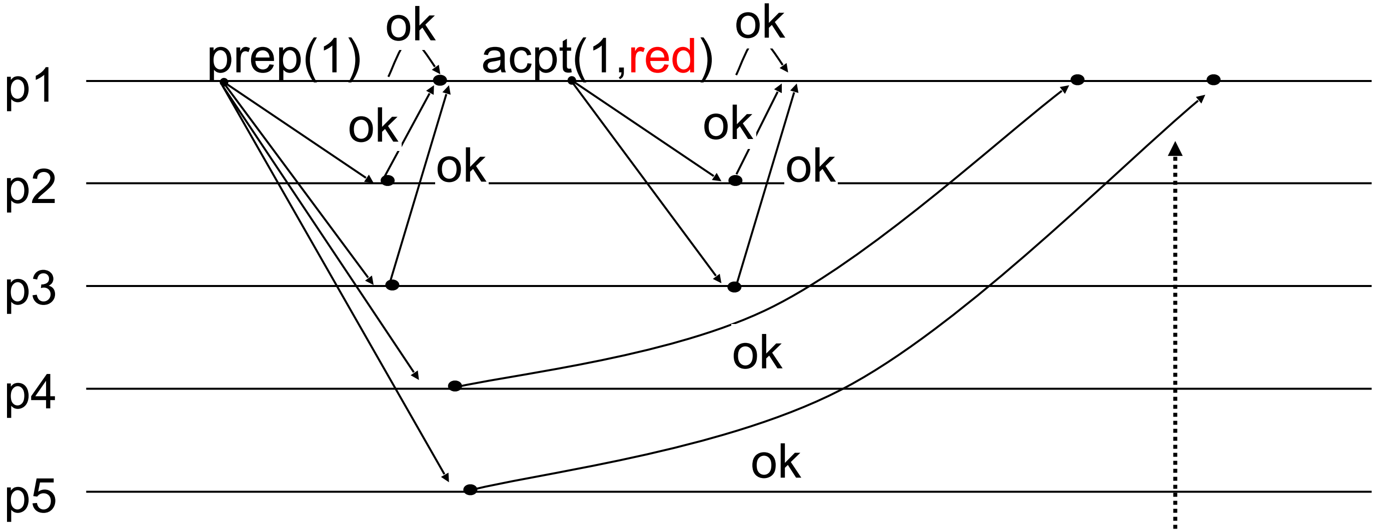


Summary of Optimizations (2)

- Necessary
  - Reject accept(n,v) if answered prepare(m) : m>n
    - i.e. prepare extracts promise to reject lower accept
- Optimizations
  - a) Reject prepare(n) if answered prepare(m) : m>n
    - i.e. prepare extracts promise to reject lower prepare
  - b) Reject accept(n,v) if answered accept(m,u) : m>n
    - i.e. accept extracts promise to reject lower accept
  - c) Reject prepare(n) if answered accept(m,u) : m>n
    - i.e. accept extracts promise to reject lower prepare

Possible scenario #3

- Caveat



Opt: ignore old responses

Summary of Optimizations (3)

- Necessary
  - Reject accept(n,v) if answered prepare(m) : m>n
    - i.e. prepare extracts promise to reject lower accept
- Optimizations
  - a) Reject prepare(n) if answered prepare(m) : m>n
    - i.e. prepare extracts promise to reject lower prepare
  - b) Reject accept(n,v) if answered accept(m,u) : m>n
    - i.e. accept extracts promise to reject lower accept
  - c) Reject prepare(n) if answered accept(m,u) : m>n
    - i.e. accept extracts promise to reject lower prepare
  - d) Ignore old messages to proposals that got majority

State to Remember

- Each acceptor remembers
  - Highest proposal (n,v) accepted
    - Needed when proposers ask prepare(m)
    - Lower prepares anyway ignored (optimization a & c)
  - Highest prepare it has promised
    - It has promised to ignore accept(m) with lower number
- Can be saved to stable storage (recovery)

One more optimizations - 1

- Paxos requires 2 round-trips (with no contention)
  - Prepare(n) : prepare phase (read phase)
  - Accept(n, v): accept phase (write phase)

- P2. If v is chosen, every higher proposal chosen has value v
- Optimization 1
- Proposer skips the accept phase if a majority of acceptors return the same value v

Performance

- Paxos requires 4 messages delays (2 round-trips)
  - Prepare(n) needs 2 delays (Broadcast & Get Majority)
  - Accept(n,v) needs 2 delays (Broadcast & Get Majority)

- In many cases only accept phase is run
  - Paxos only needs 2 delays to terminate
    - (Believed to be) optimal

Two more optimizations - 2

- Paxos requires 2 round-trips (with no contention)
  - Prepare(n) : prepare phase (read phase)
  - Accept(n, v): accept phase (write phase)
- We often need to run many consensus instances
  - Note that proposer needs not know value in prepare(n)
  - Initialize acceptors as if they accepted a prepare(1) of an initial leader l_1 among possible proposers
  - Initially l_1 runs only accept phase until suspected
  - Subsequent leaders can run prepare for many instances in advance (with highe ballot number)