Paxos week and the Temple of Doom

Doug Woos

Logistics notes

Next Monday: International Workers' Day

- No in-class lecture
- Will record a video lecture
- Please watch by next Wednesday!

Lab 2b due tonight at 9pm

Problem Set 2 due Friday

- *Typeset, short* answers, please!

Logistics notes

Friday: Paxos Made Moderately Complex

- *Not* an easy read

- Describes how to implement a practical version of Paxos, in some detail

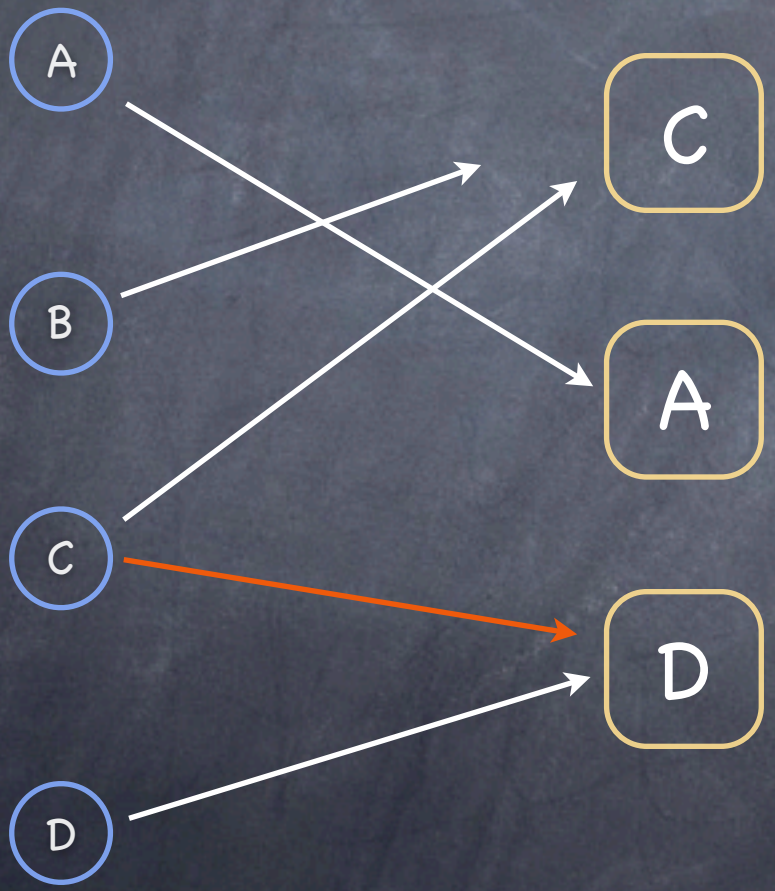
- Please spend an hour on it!

Review / Clarifications

State Machine Replication

- Applying a series of client commands to some replicated state s.t. all nodes agree on the ordering of commands
- Paxos: which command should we do next?

Proposers and acceptors



A = Put k1 v1
B = PutAppend k2 v2
C = Get k3
D = Delete k1

Proposal numbers

- Attach numbers to proposals
- Used to ensure progress
- Must be unique—only used for one value!
- Unique if:
  - Each node uses increasing proposal numbers
  - And proposal numbers unique across nodes

Proposal numbers

A. 0, 4, 8, 12, 16, ...

B. 1, 5, 9, 13, 17, ...

C. 2, 6, 10, 14, 18, ...

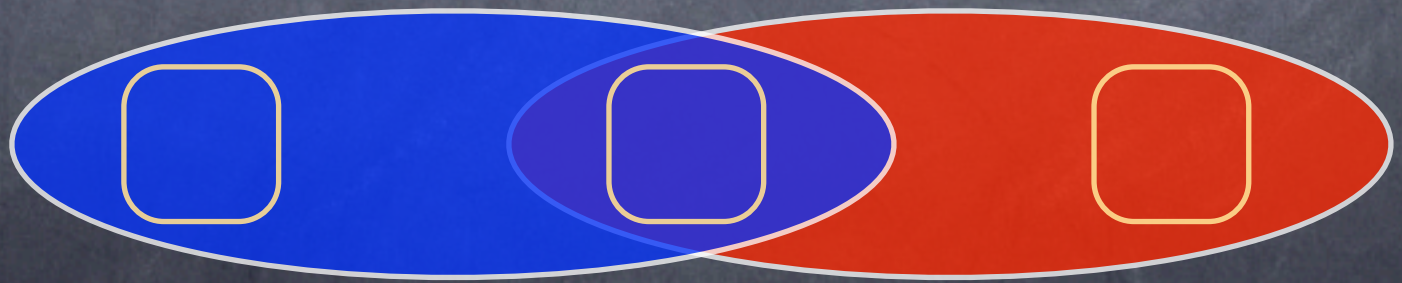
D. 3, 7, 11, 15, 19, ...

Majorities

- Why does Paxos use majorities?
- Majorities intersect: for any two majorities S and S', there is some node in both S and S'

Majorities

- Why does Paxos use majorities?
- Majorities intersect: for any two majorities S and S', there is some node in both S and S'



A few terms

- A proposal is proposed when it is sent from a proposer to an acceptor
- A proposal is accepted when an acceptor accepts it
- A proposal is chosen when it is accepted by a majority of acceptors

Our approach

- Started with a broad definition of consensus

  We should eventually choose a value

  We should only choose one value

Our approach

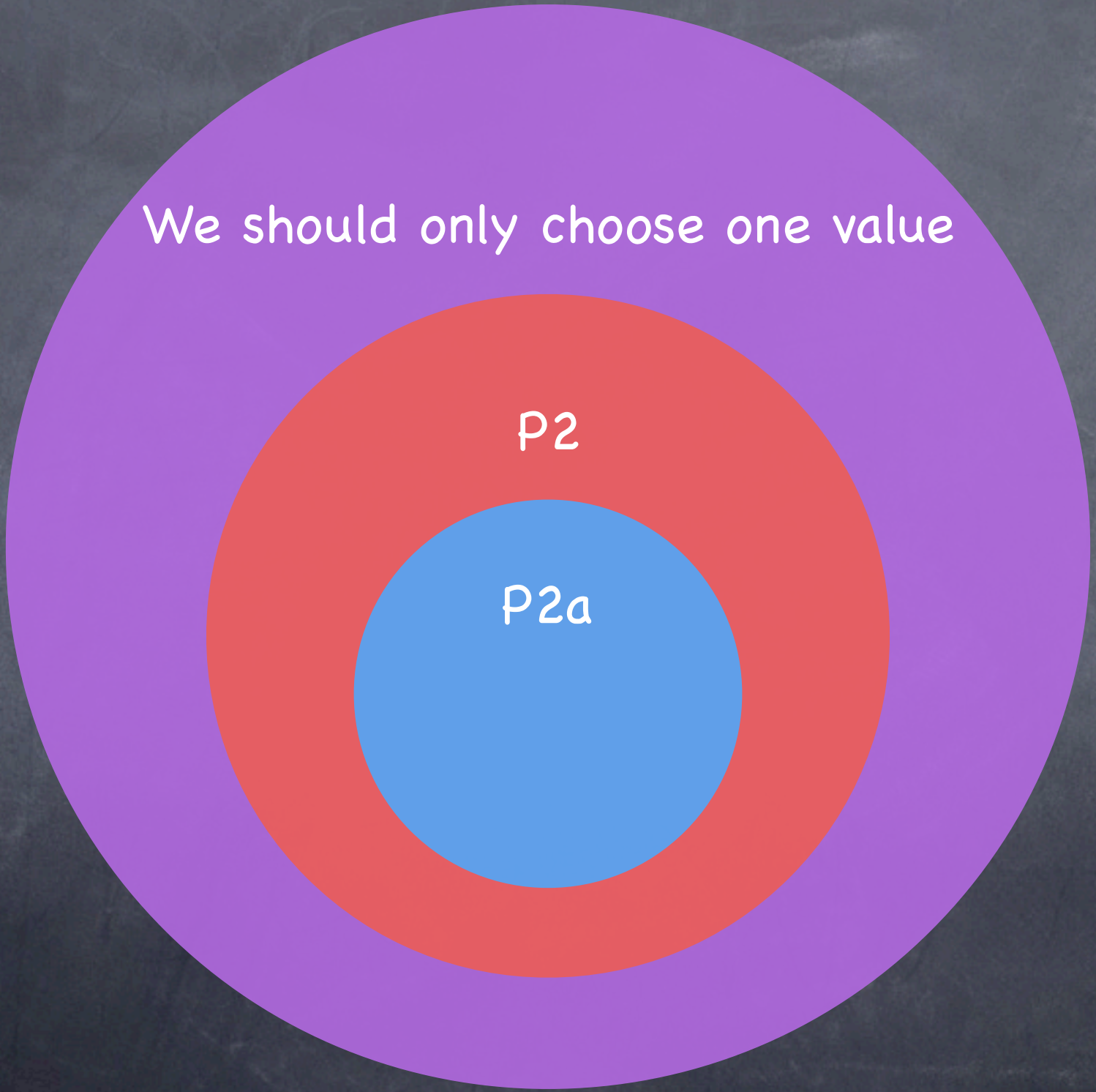
- Refined further

  P1: An acceptor must accept the first proposal that it receives

  P2. If a proposal with value v is chosen, then every higher-numbered proposal that is chosen has value v

Our approach

- We're refining our specification until we end up at an actual implementation
- At each step, we're arguing that the refinement is valid—e.g., that P2a implies P2
  - All systems that have property P2a have property P2


We should only choose one value
P2
P2a

The story so far

P1: An acceptor must accept the first proposal that it receives

P2. If a proposal with value v is chosen, then every higher-numbered proposal that is chosen has value v

P2a. If a proposal with value v is chosen, then every higher-numbered proposal accepted by any acceptor has value v

Another take on P2

- Recall P2a:

  If a proposal with value v is chosen, then every higher-numbered proposal accepted by any acceptor has value v

We strengthen it to:

P2b: If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v

Implementing P2 (I)

P2b: If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v

Suppose a proposer p wants to issue a proposal numbered n. What value should p propose?

- If (n',v) with $n' < n$ is chosen, then in every majority set S of acceptors at least one acceptor has accepted (n',v)...
- ...so, if there is a majority set S where no acceptor has accepted (or will accept) a proposal with number less than n, then p can propose any value

Implementing P2 (II)

P2b: If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v

What if for all S some acceptor ends up accepting a pair (n',v) with $n' < n$?

Claim: p should propose the value of the highest numbered proposal among all accepted proposals numbered less than n

Proof: By induction on the number of proposals issued after a proposal is chosen

Implementing P2 (II)

What do we know about the third acceptor?

Could it have accepted (1,A)?

Could it have accepted (2,B)?

Implementing P2 (II)

What do we know about the third acceptor?

Could it have accepted (1,A)? No.

Could it have accepted (2,B)? Yes.

Proposal with highest number is the only proposal that could have been chosen!

Implementing P2 (III)

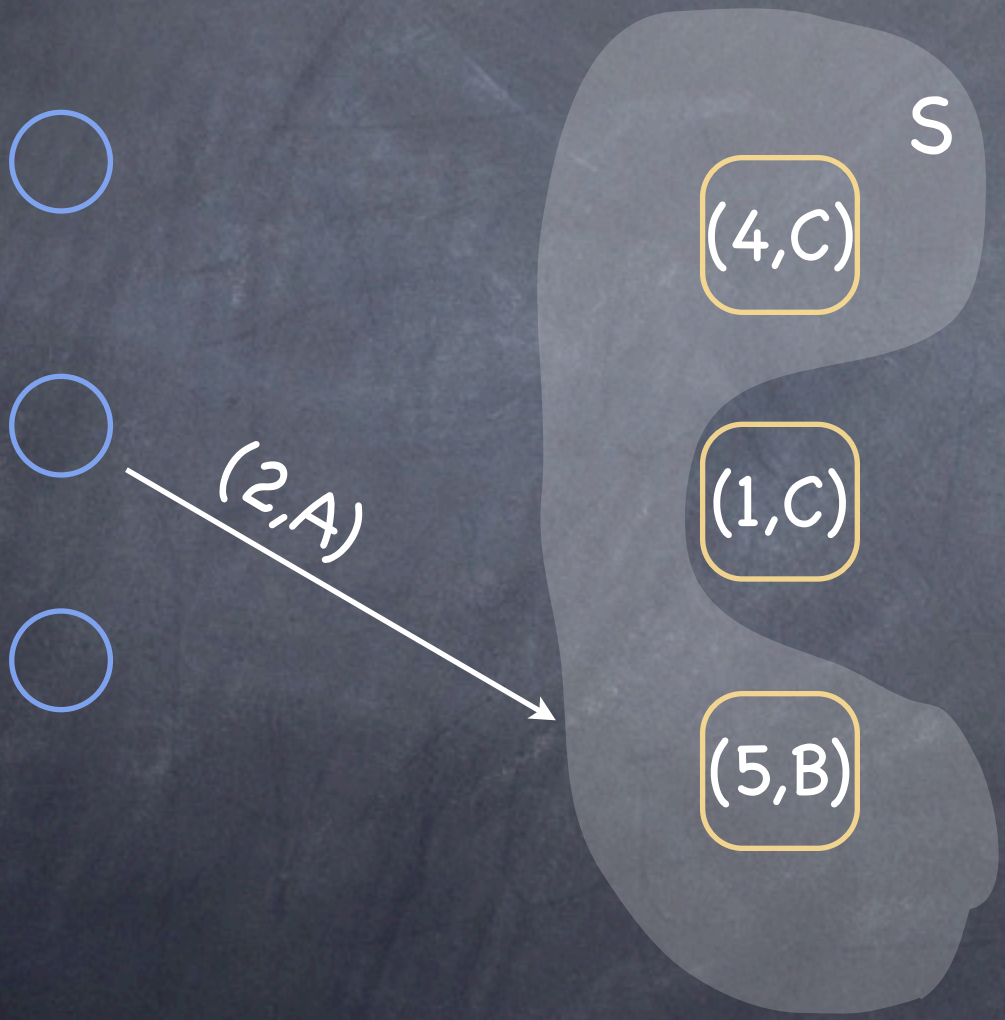
P2b: If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v

Achieved by enforcing the following invariant

P2c: For any v and n, if a proposal with value v and number n is issued, then there is a set S consisting of a majority of acceptors such that either:

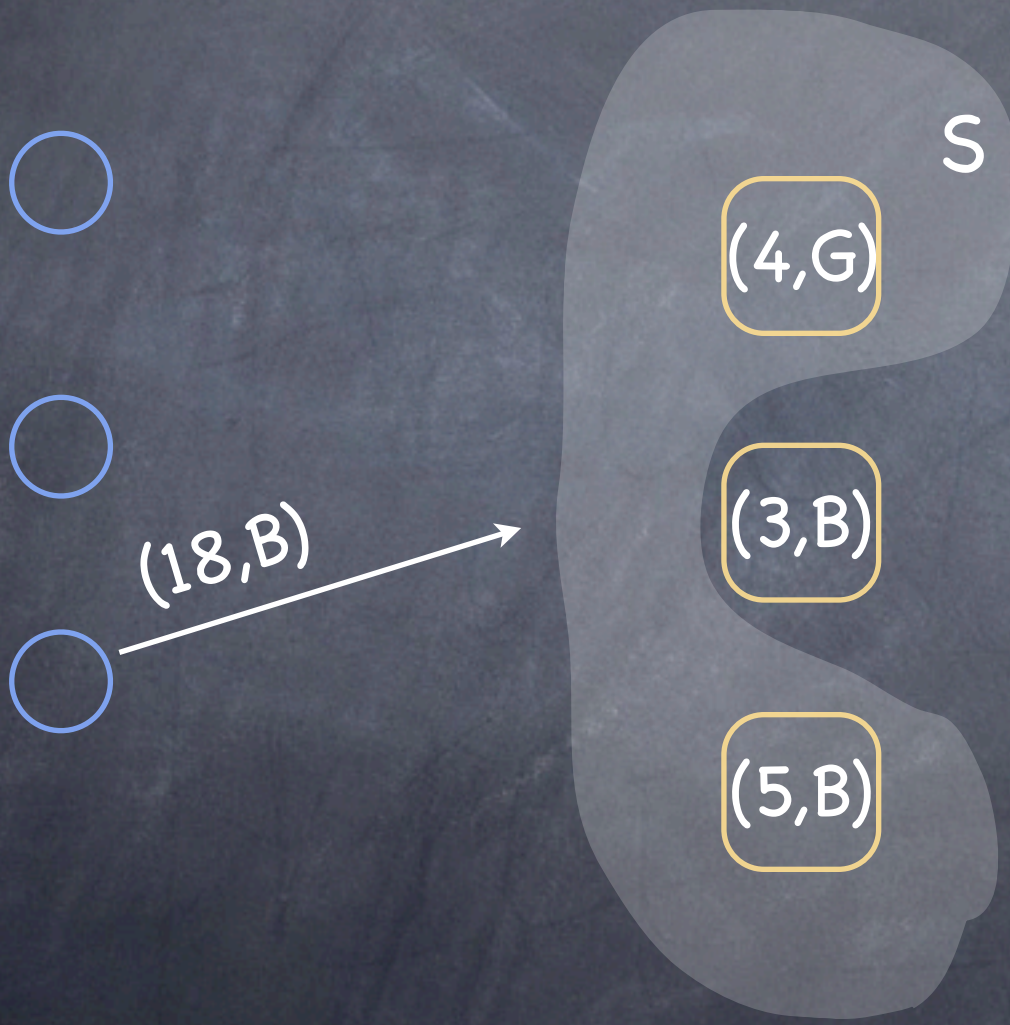
- no acceptor in S has accepted any proposal numbered less than n, or
- v is the value of the highest-numbered proposal among all proposals numbered less than n accepted by the acceptors in S

P2c in action



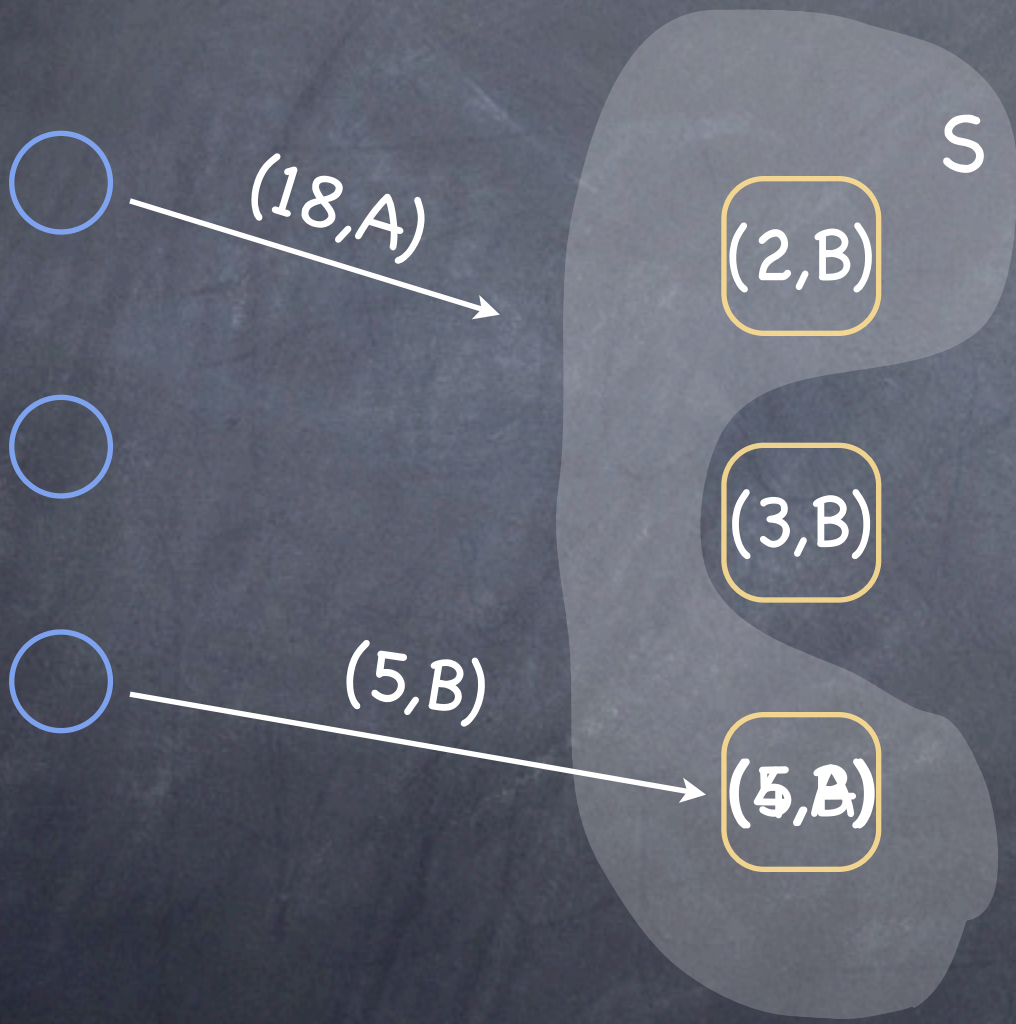
- No acceptor in S has accepted any proposal numbered less than n

P2c in action



- v is the value of the highest-numbered proposal among all proposals numbered less than n and accepted by the acceptors in S

P2c in action



- v is the value of the highest-numbered proposal among all proposals numbered less than n and accepted by the acceptors in S

The invariant is violated

Future telling?

- To maintain P2c, a proposer that wishes to propose a proposal numbered n must learn the highest-numbered proposal with number less than n, if any, that has been or will be accepted by each acceptor in some majority of acceptors

Future telling?

- To maintain P2c, a proposer that wishes to propose a proposal numbered n must learn the highest-numbered proposal with number less than n, if any, that has been or will be accepted by each acceptor in some majority of acceptors
- Avoid predicting the future by extracting a promise from a majority of acceptors not to subsequently accept any proposals numbered less than n

The proposer's protocol (I)

- A proposer chooses a new proposal number n and sends a request to each member of some (majority) set of acceptors, asking it to respond with:
  a. A promise never again to accept a proposal numbered less than n, and
  b. The accepted proposal with highest number less than n if any.

...call this a prepare request with number n

The proposer's protocol (II)

- If the proposer receives a response from a majority of acceptors, then it can issue a proposal with number n and value v, where v is
  a. the value of the highest-numbered proposal among the responses, or
  b. is any value selected by the proposer if responders returned no proposals

A proposer issues a proposal by sending, to some set of acceptors, a request that the proposal be accepted.

...call this an accept request.

The acceptor's protocol

- An acceptor receives prepare and accept requests from proposers. It can ignore these without affecting safety.
  - It can always respond to a prepare request
  - It can respond to an accept request, accepting the proposal, iff it has not promised not to, e.g.

P1a: An acceptor can accept a proposal numbered n iff it has not responded to a prepare request having number greater than n

...which subsumes P1.

Small optimizations

- If an acceptor receives a prepare request r numbered n when it has already responded to a prepare request for $n' > n$, then the acceptor can simply ignore r.
- An acceptor can also ignore prepare requests for proposals it has already accepted

...so an acceptor needs only remember the highest numbered proposal it has accepted and the number of the highest-numbered prepare request to which it has responded.

This information needs to be stored on stable storage to allow restarts.

Choosing a value: Phase 1

- A proposer chooses a new n and sends <*prepare*,n> to a majority of acceptors
- If an acceptor a receives <*prepare*,n'>, where $n' > n$ of any <*prepare*,n> to which it has responded, then it responds to <*prepare*, n'> with
  - a promise not to accept any more proposals numbered less than n'
  - the highest numbered proposal (if any) that it has accepted

Choosing a value: Phase 2

- If the proposer receives a response to <*prepare*,*n*> from a majority of acceptors, then it sends to each <*accept*,*n*,*v*>, where v is either
  - the value of the highest numbered proposal among the responses
  - any value if the responses reported no proposals
- If an acceptor receives <*accept*,*n*,*v*>, it accepts the proposal unless it has in the meantime responded to <*prepare*,*n'*> , where $n' > n$

Learning chosen values (I)

Once a value is chosen, learners should find out about it. Many strategies are possible:

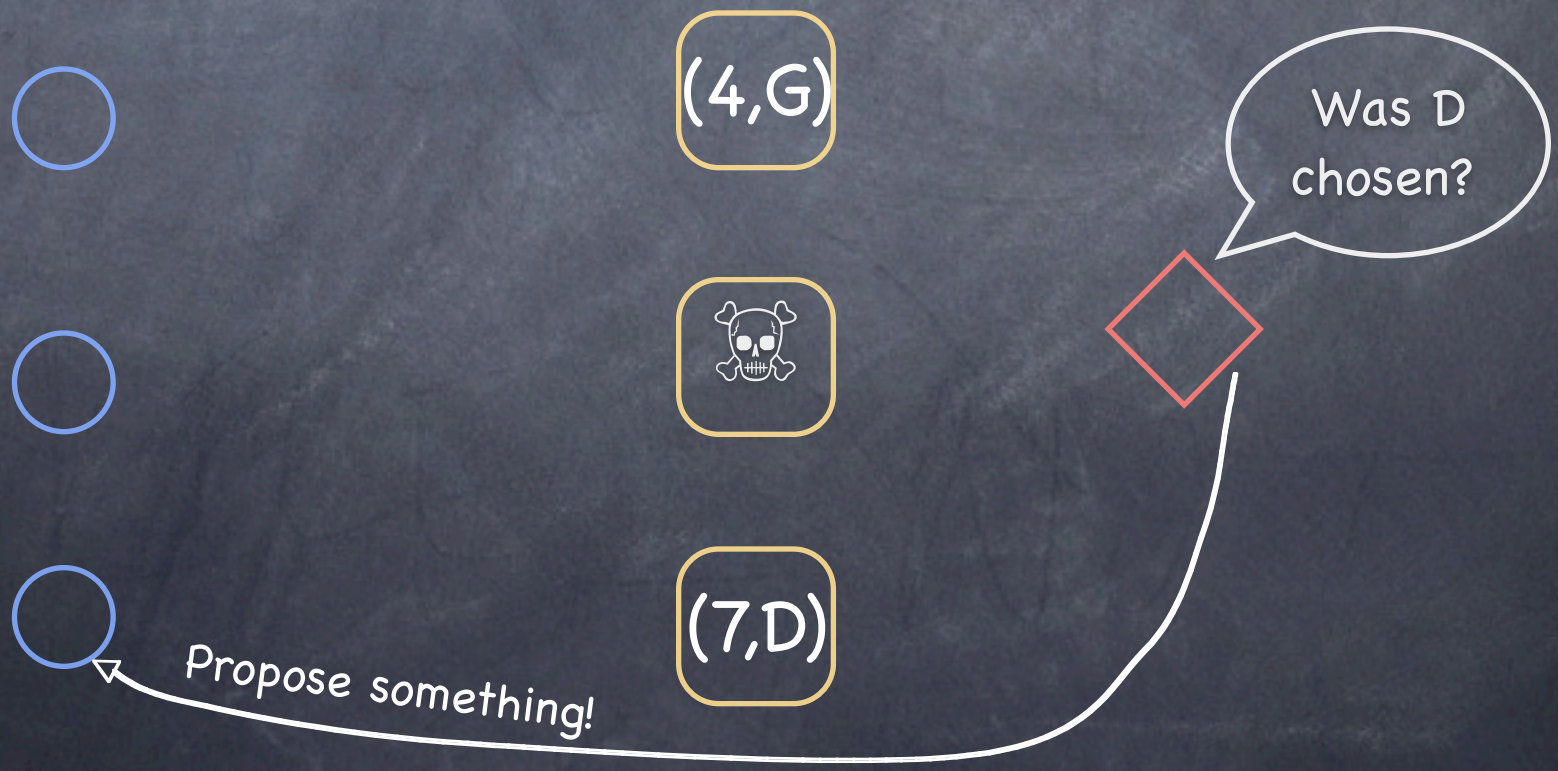
i. Each acceptor informs each learner whenever it accepts a proposal.

ii. Acceptors inform a distinguished learner, who informs the other learners

iii. Something in between (a set of not-quite-as-distinguished learners)

Learning chosen values (II)

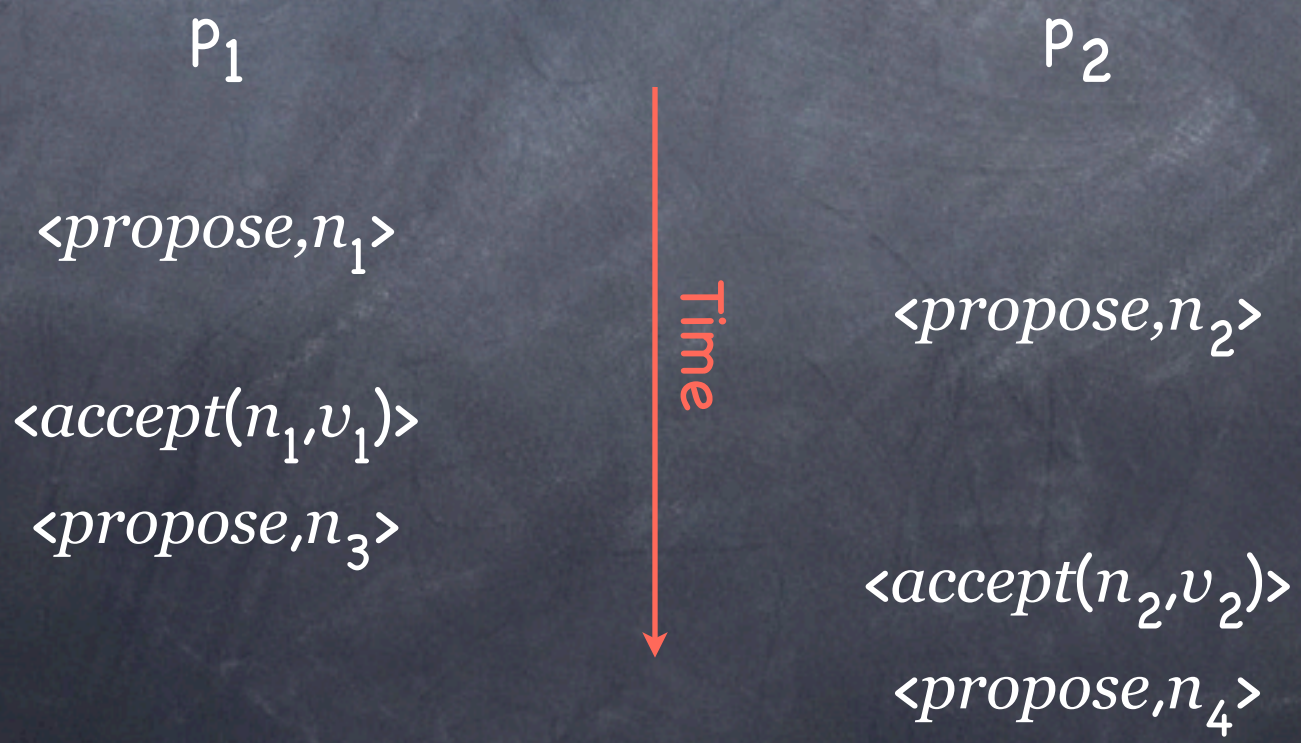
Because of failures (message loss and acceptor crashes) a learner may not learn that a value has been chosen



Liveness

Progress is not guaranteed:

$n_1 < n_2 < n_3 < n_4 < \ldots$



Implementing State Machine Replication

- Implement a sequence of separate instances of consensus, where the value chosen by the i^{th} instance is the i^{th} message in the sequence.
- Each server assumes all three roles in each instance of the algorithm.
- Assume that the set of servers is fixed

The role of the leader

- In normal operation, elect a single server to be a leader. The leader acts as the distinguished proposer in all instances of the consensus algorithm.
  - Clients send commands to the leader, which decides where in the sequence each command should appear.
  - If the leader, for example, decides that a client command is the k^{th} command, it tries to have the command chosen as the value in the k^{th} instance of consensus.

A new leader λ is elected...

- Since λ is a learner in all instances of consensus, it should know most of the commands that have already been chosen. For example, it might know commands 1-10, 13, and 15.
  - It executes phase 1 of instances 11, 12, and 14 and of all instances 16 and larger.
  - This might leave, say, 14 and 16 constrained and 11, 12 and all commands after 16 unconstrained.
  - λ then executes phase 2 of 14 and 16, thereby choosing the commands numbered 14 and 16

Stop-gap measures

- All replicas can execute commands 1-10, but not 13-16 because 11 and 12 haven't yet been chosen.
- λ can either take the next two commands requested by clients to be commands 11 and 12, or can propose immediately that 11 and 12 be no-op commands.
- λ runs phase 2 of consensus for instance numbers 11 and 12.
- Once consensus is achieved, all replicas can execute all commands through 16.

To infinity, and beyond

- λ can efficiently execute phase 1 for infinitely many instances of consensus! (e.g. command 16 and higher)
  - λ just sends a message with a sufficiently high proposal number for all instances
  - An acceptor replies non trivially only for instances for which it has already accepted a value

Paxos and FLP

- Paxos is always safe–despite asynchrony
- Once a leader is elected, Paxos is live.
- "Ciao ciao" FLP?
  - To be live, Paxos requires a single leader
  - "Leader election" is impossible in an asynchronous system (gotcha!)
- Given FLP, Paxos is the next best thing: always safe, and live during periods of synchrony

Delegation

- Paxos is expensive compared to primary/backup; can we get the best of both worlds?
- Paxos group leases responsibility for order of operations to a primary, for a limited period
- If primary fails, wait for lease to expire, then can resume operation (after checking backups)
- If no failures, can refresh lease as needed

Byzantine Paxos

- What if a Paxos node goes rogue? (or two?)
- Solution sketch: instead of just one node in the overlap between majority sets, need more: 2f + 1, to handle f byzantine nodes