Paxos!

CSE 452

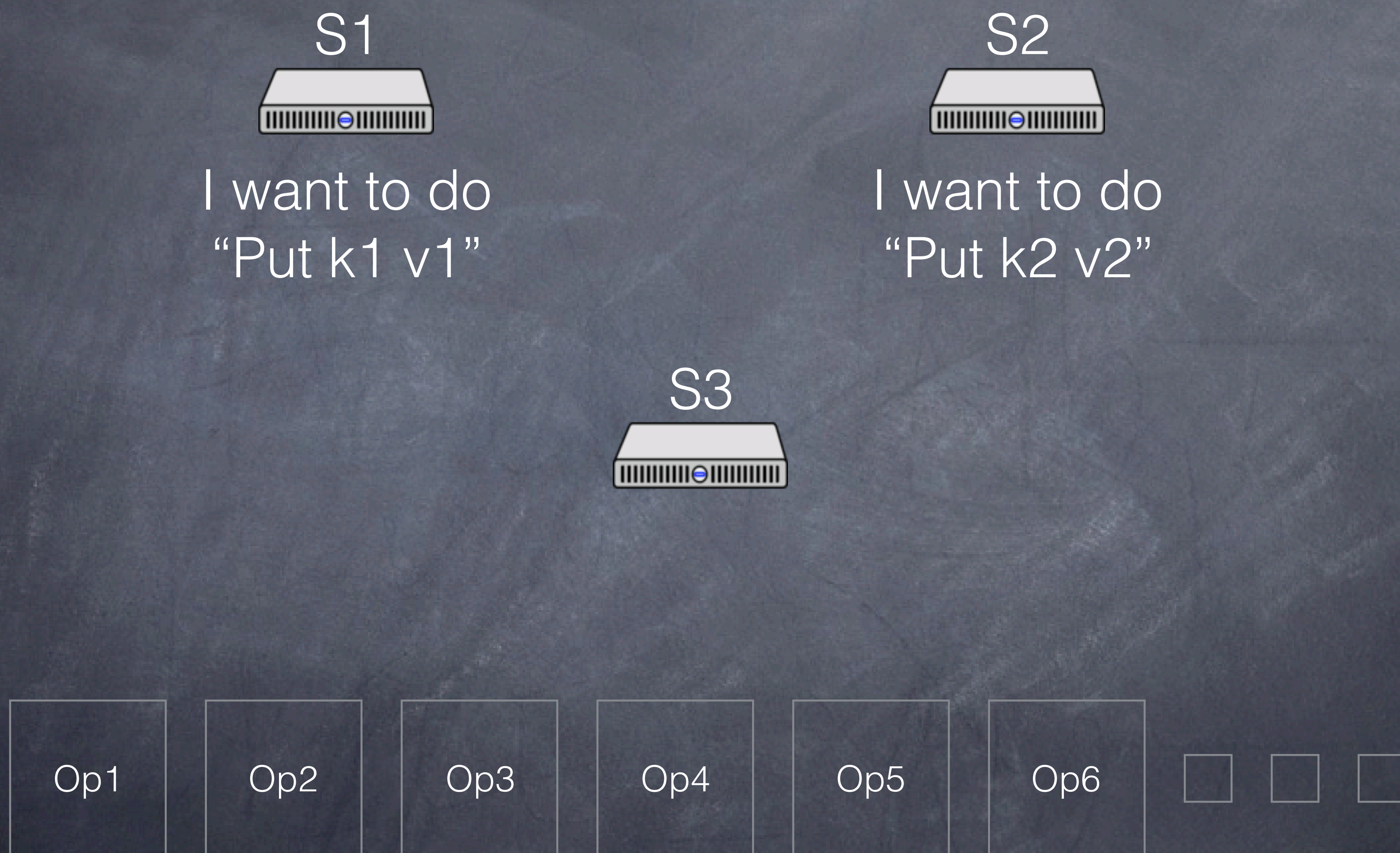
Slides from Lorenzo Alvisi, Doug Woos, Tom Anderson

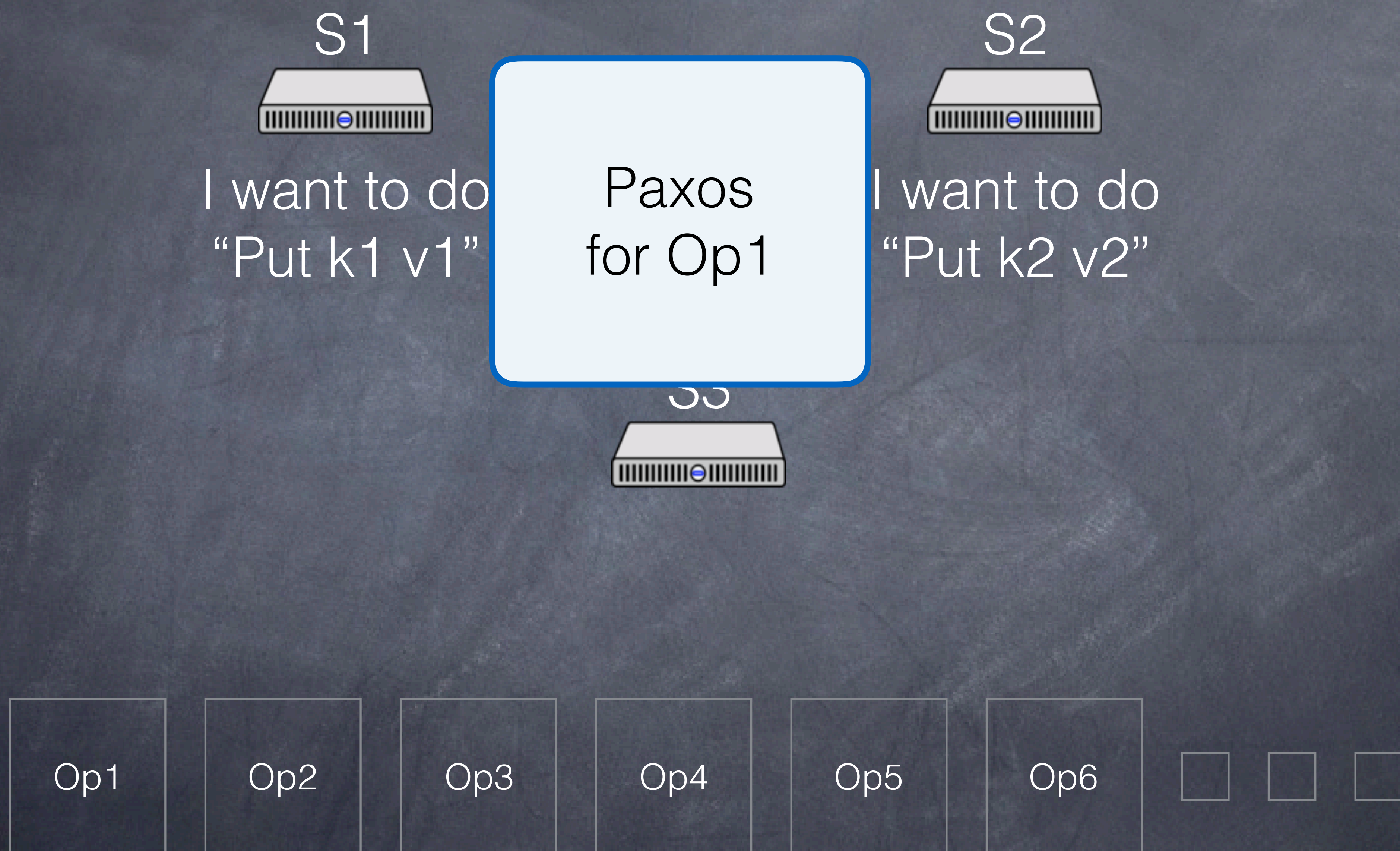
State machine replication

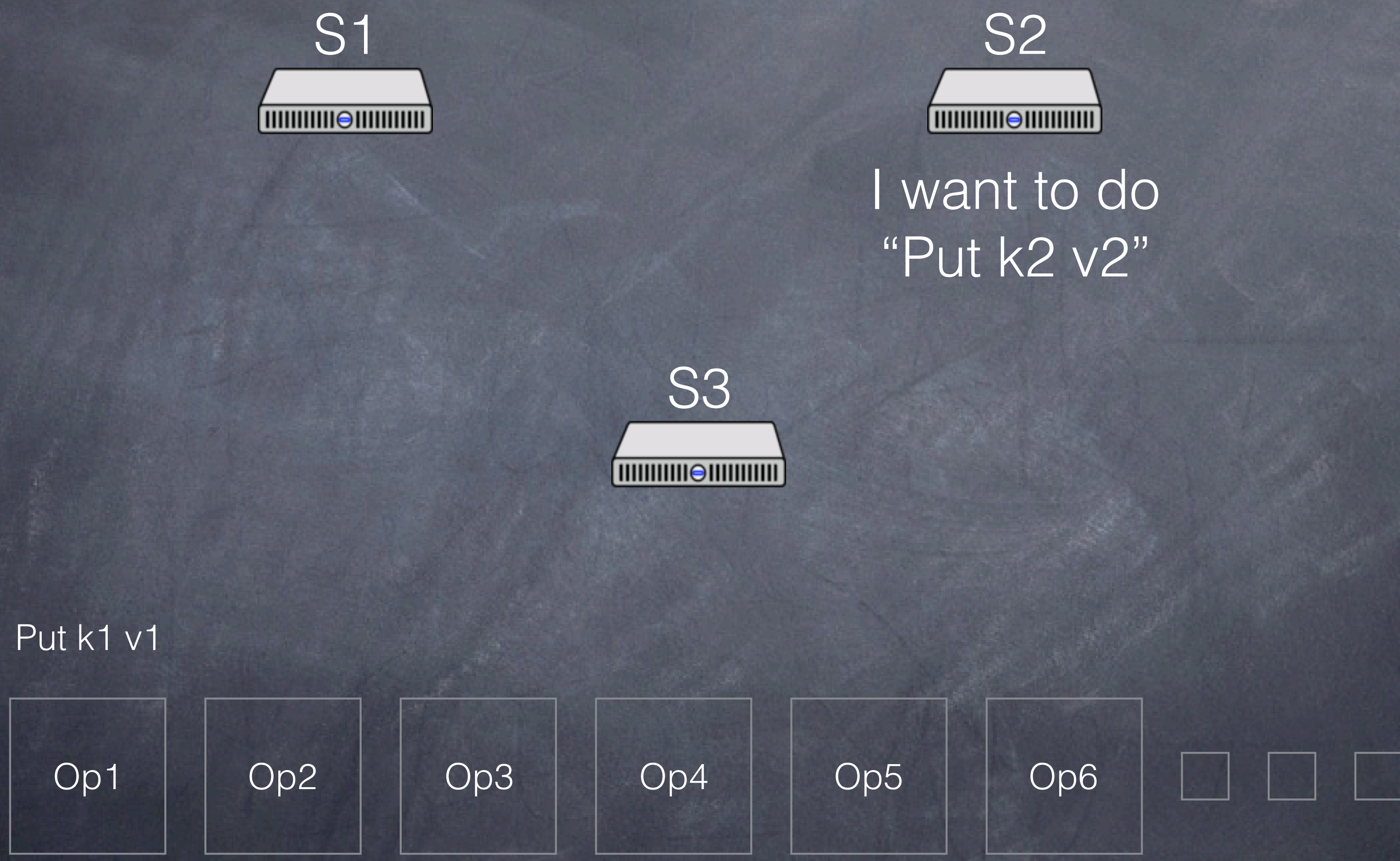
Want to agree on order of ops

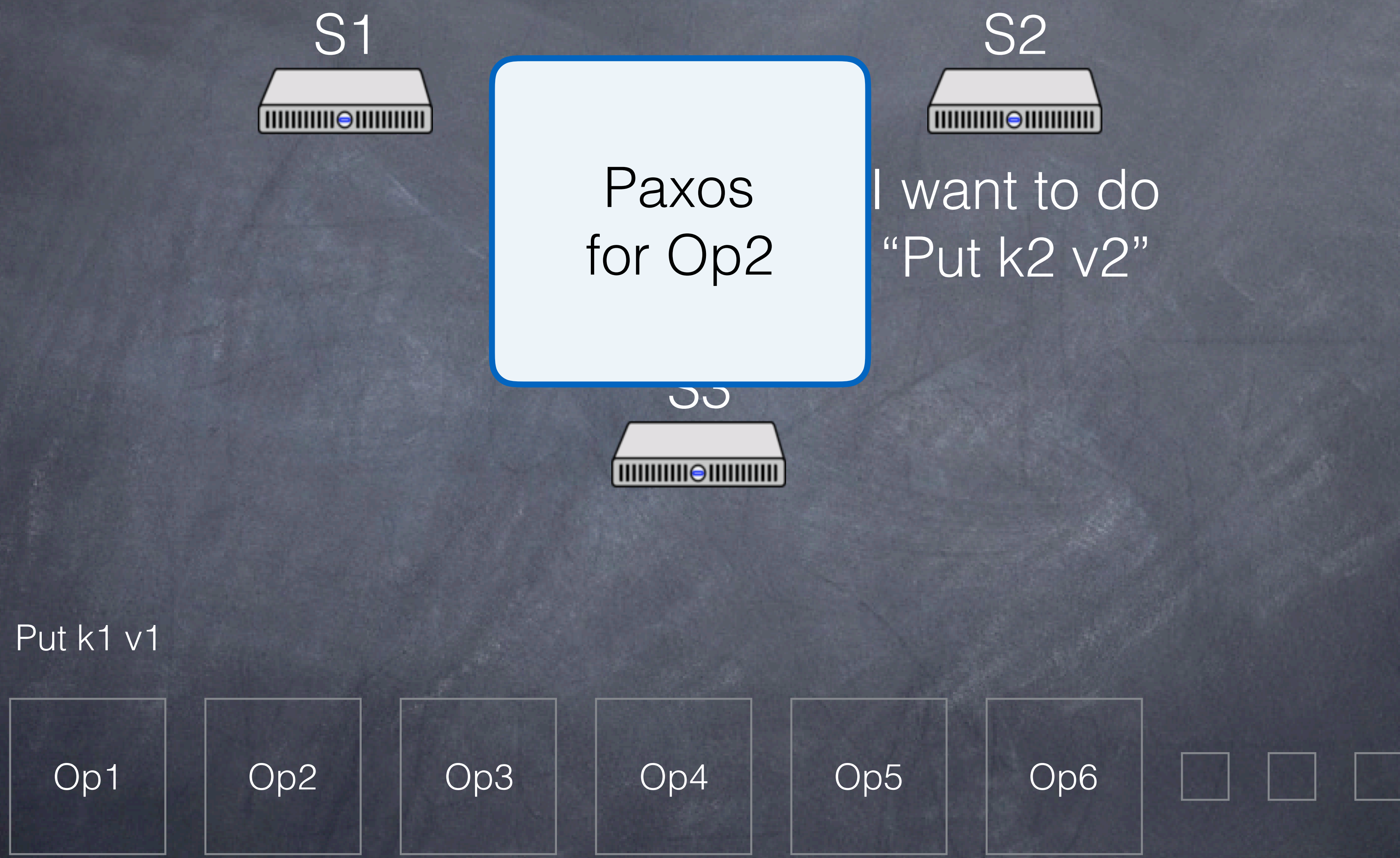
Can think of operations as a log


S1
S2
S3
Op1
Op2
Op3
Op4
Op5
Op6

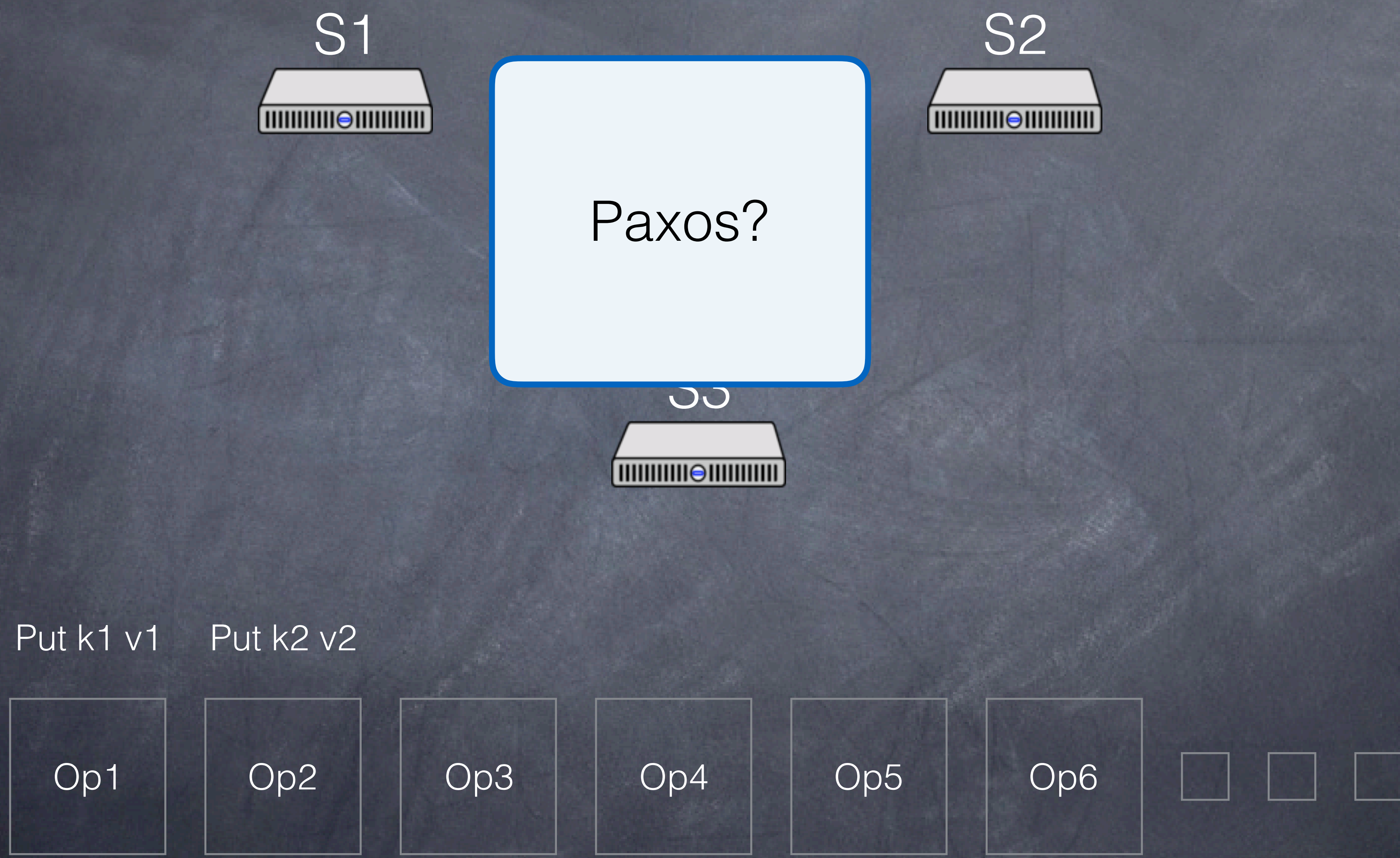

S1
I want to do
“Put k1 v1”
S2
I want to do
“Put k2 v2”
S3
Op1
Op2
Op3
Op4
Op5
Op6

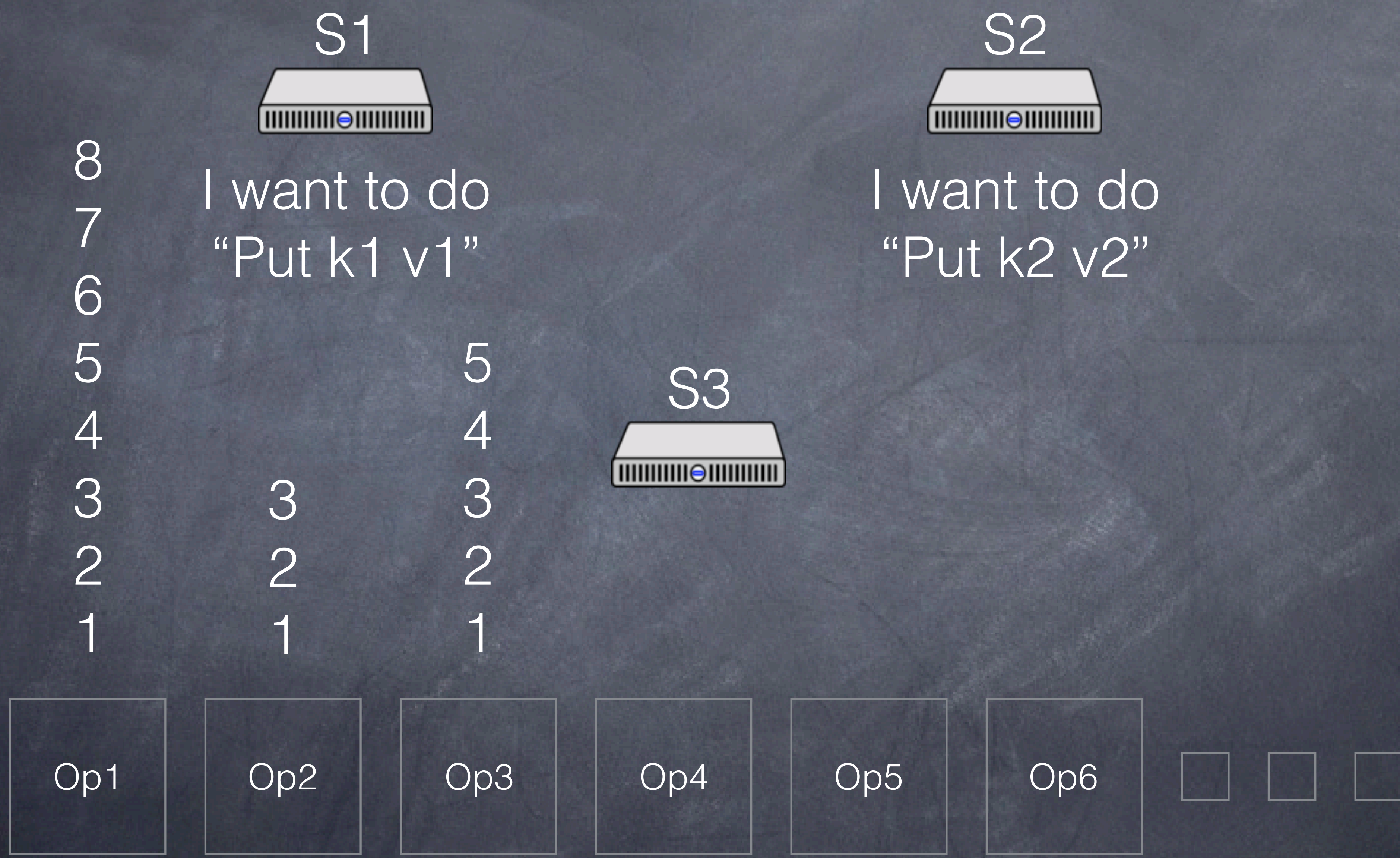

S1
S2
I want to do
“Put k1 v1”
Paxos
for Op1
I want to do
“Put k2 v2”
Op1
Op2
Op3
Op4
Op5
Op6


S1
S2
I want to do
“Put k2 v2”
S3
Put k1 v1
Op1
Op2
Op3
Op4
Op5
Op6


S1
S2
Paxos for Op2
I want to do "Put k2 v2"
Put k1 v1
Op1
Op2
Op3
Op4
Op5
Op6

S1
S2
S3
Put k1 v1
Put k2 v2
Op1
Op2
Op3
Op4
Op5
Op6


S1
S2
Paxos?
Put k1 v1
Put k2 v2
Op1
Op2
Op3
Op4
Op5
Op6


S1
I want to do
"Put k1 v1"
S2
I want to do
"Put k2 v2"
S3
8
7
6
5
4
3
2
1
3
2
1
5
4
3
2
1
Op1
Op2
Op3
Op4
Op5
Op6

Why Multiple Proposals?

Consensus is easy if only one client request at a time.

So, select a leader:

- clients send requests to leader
- leader picks what goes first, tells everyone else

What about split brain? (leader failed, or slow)

- if old leader is slow, might have two leaders!
- if old and new leader are slow, might have three!

Each makes a proposal for what to go next

Primary-Backup Replication?

- Suppose using primary/hot standby replication
- How can we tell if primary has failed versus is slow? (if slow, might end up with two primaries!)
- Rely on view server to decide?
- What if view server goes down? Replicate?
- How can we tell if view server replica has failed or is slow?
- ...

The Part-Time Parliament

- Parliament determines laws by passing sequence of numbered decrees
- Legislators can leave and enter the chamber at arbitrary times
- No centralized record of approved decrees- instead, each legislator carries a ledger

Government 101

- No two ledgers contain contradictory information
- If a majority of legislators were in the Chamber and no one entered or left the Chamber for a sufficiently long time, then
  - any decree proposed by a legislator would eventually be passed
  - any passed decree would appear on the ledger of every legislator

Government 102

- Paxos legislature is non-partisan, progressive, and well-intentioned
- Legislators only care that something is agreed to, not what is agreed to
- To deal with Byzantine legislators, see Castro and Liskov, SOSP 99

Back to the future

- A set of processes that can propose values
- Processes can crash and recover
- Processes have access to stable storage
- Asynchronous communication via messages
- Messages can be lost and duplicated, but not corrupted

The Players

- Proposers
- Acceptors
- Learners

Terminology

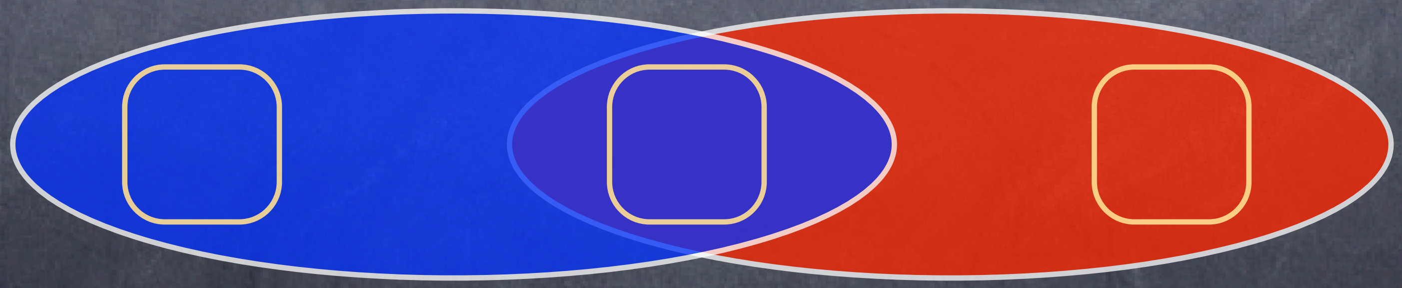
- Value: a possible operation to put in the next slot in the operation log (letter values)
- Proposal: to select a value; proposals are uniquely numbered
- Accept: of a specific proposal, value
- Chosen: Proposal/value accepted by a majority
- Learned: Fact that proposal is chosen is known

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′



The Game: Consensus

SAFETY

- Only a value that has been proposed can be chosen
- Only a single value is chosen
- A process never learns that a value has been chosen unless it has been

LIVENESS

- Some proposed value is eventually chosen
- If a value is chosen, a process eventually learns it

Our approach

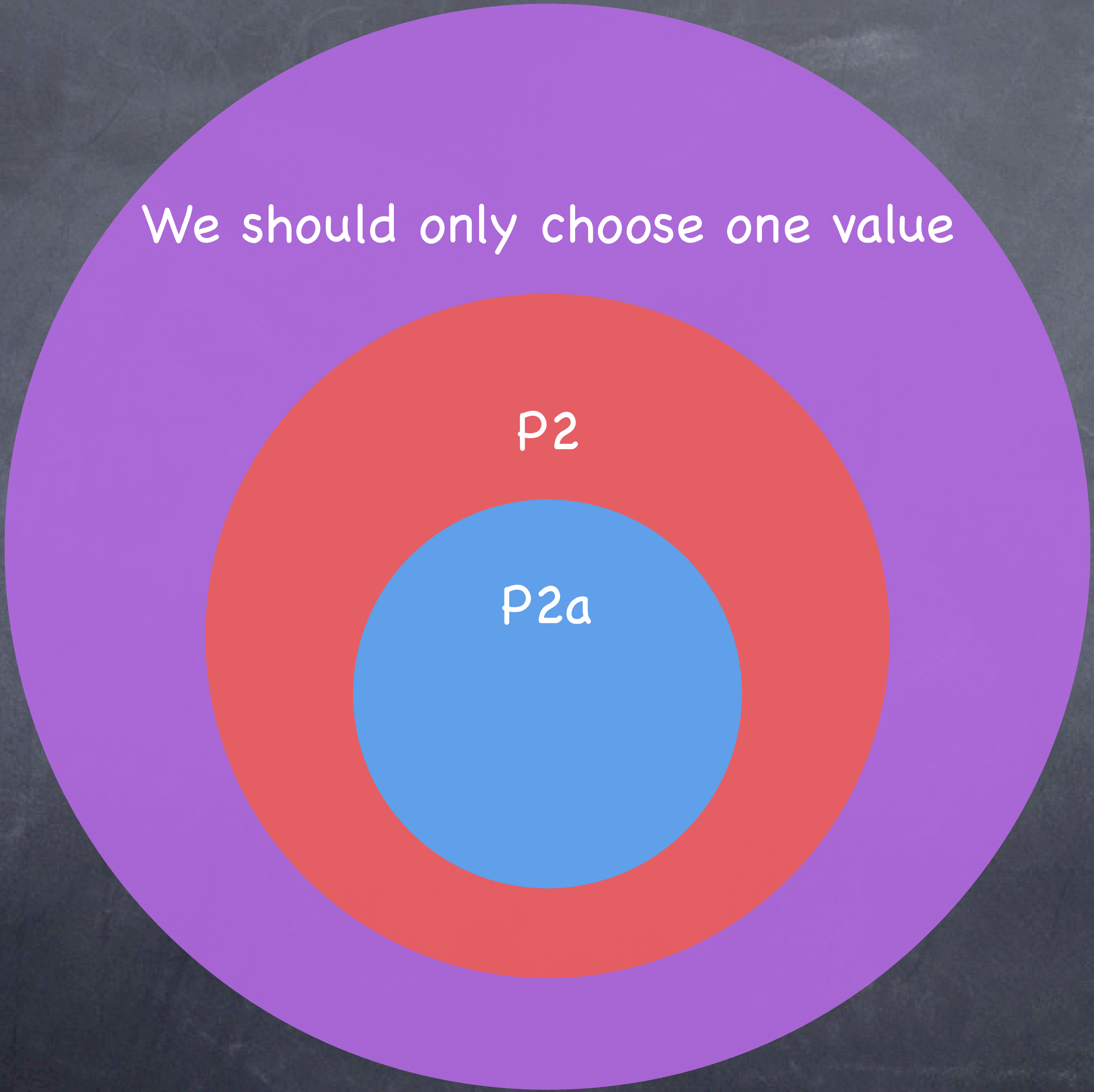
- Start with a broad definition of consensus

  We should eventually choose a value

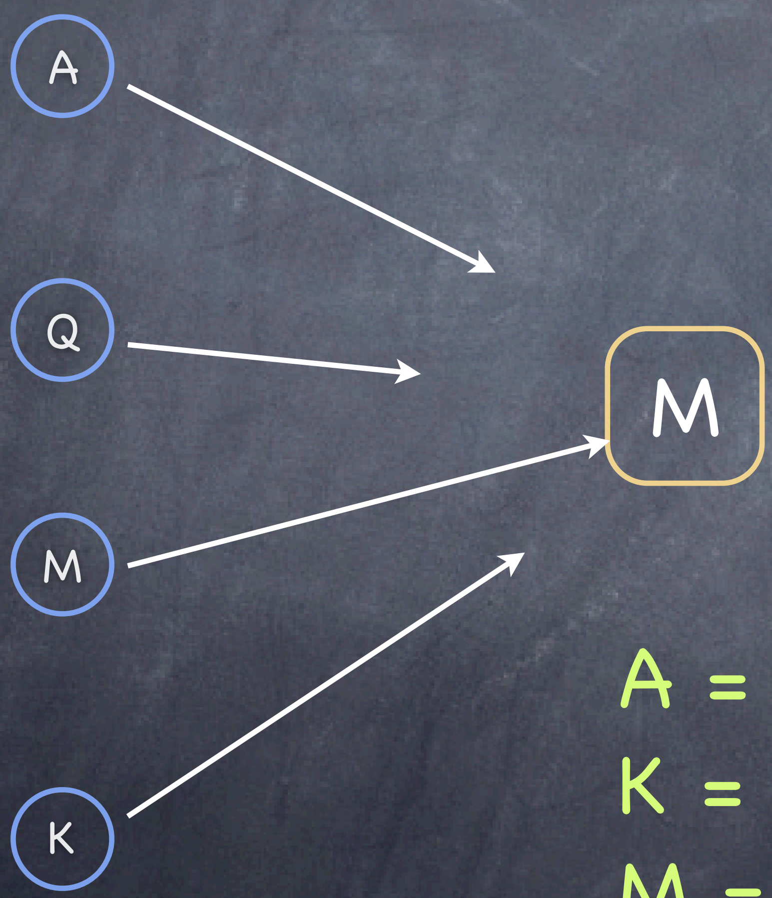
  We should only choose one value

- Refine/narrow definition to something we can implement

- At each step, Lamport must argue the refinement is valid, e.g., P2a => P2


We should only choose one value
P2
P2a

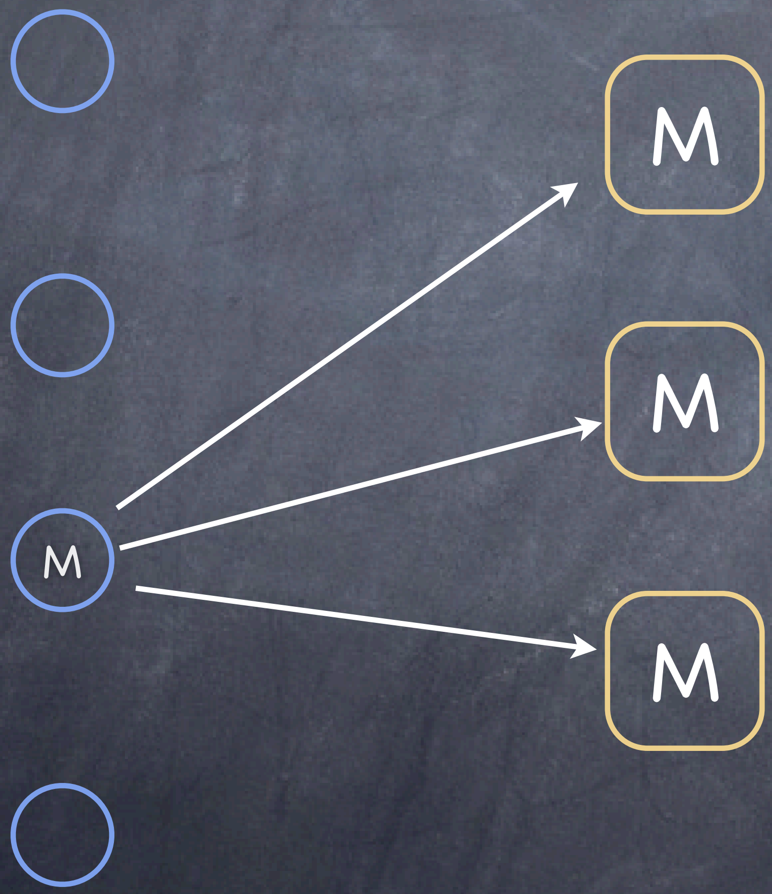
Choosing a value



Use a single acceptor

A = Put k1 v1
K = PutAppend k2 v2
M = Get k3
Q = Delete k1

What if the acceptor fails?



M is chosen!

- Choose only when a "large enough" set of acceptors accepts
- Using a majority set guarantees that at most one value is chosen

Accepting a value

- Suppose only one value is proposed by a single proposer.
- That value should be chosen!
- First requirement:

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

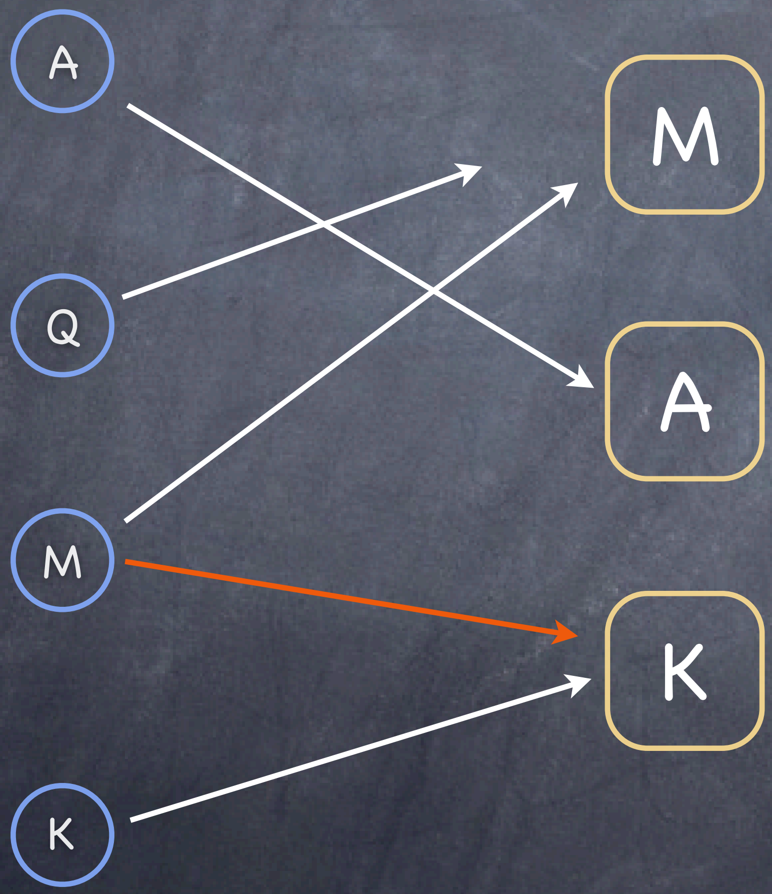
Accepting a value

- Suppose only one value is proposed by a single proposer.
- That value should be chosen!
- First requirement:

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

- ...but what if we have multiple proposers, each proposing a different value?

P1 + multiple proposers



No value is chosen!

Handling multiple proposals

- Acceptors must (be able to) accept more than one proposal
- To keep track of different proposals, assign a natural number to each proposal
  - A proposal is then a pair (psn, value)
  - Different proposals have different psn
  - A proposal is chosen when it has been accepted by a majority of acceptors
  - A value is chosen when a single proposal with that value has been chosen

Assigning Proposal Numbers

- Proposal numbers must be unique and infinite
- A proposal number server won't work...
- Instead, assign each proposer an infinite slice
- Proposer i of N gets: i, i+N, i+2N, i+3N, ...

Proposal numbers

- (A) 0, 4, 8, 12, 16, ...
- (Q) 1, 5, 9, 13, 17, ...
- (M) 2, 6, 10, 14, 18, ...
- (K) 3, 7, 11, 15, 19, ...

Choosing a unique value

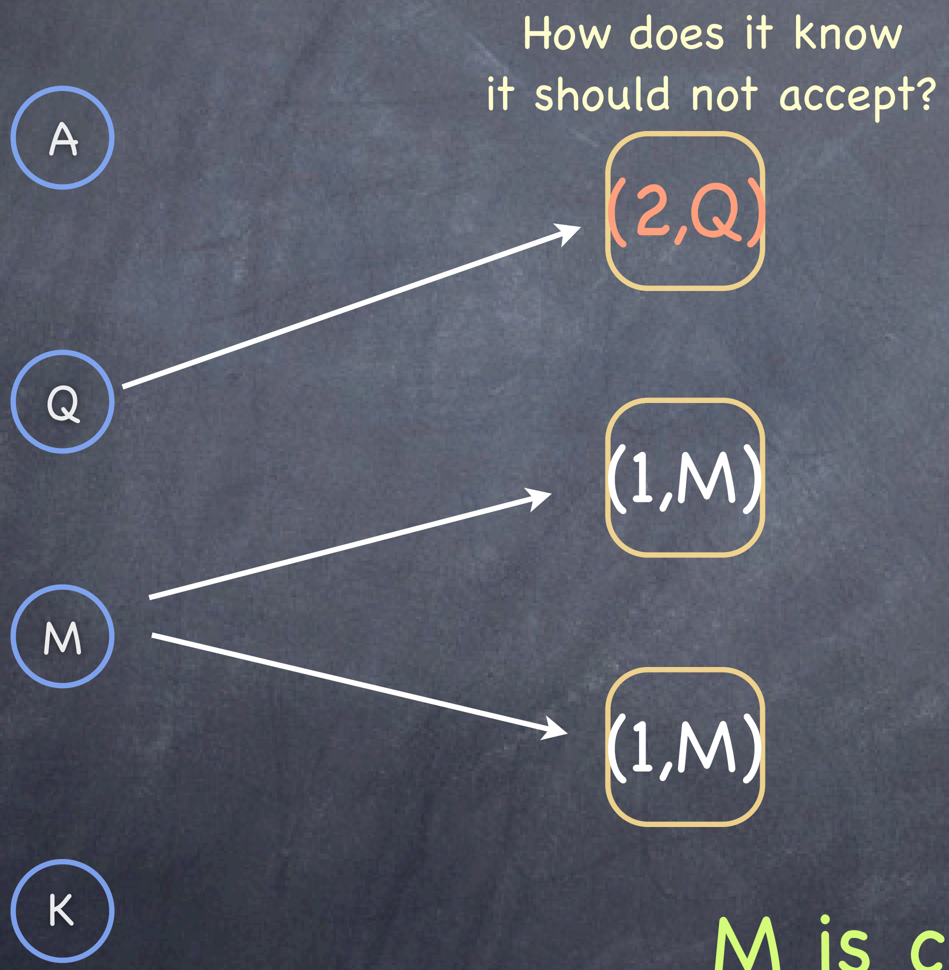
- We need to guarantee that all chosen proposals result in choosing the same value
- We introduce a second requirement (by induction on the proposal number):

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

  which can be satisfied by:

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

What about P1?



- Do we still need P1?

  YES, to ensure that *some* proposal is accepted

- How well do P1 and P2a play together?

  Asynchrony is a problem...

M is chosen!

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 (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

Implementing P2c

What do we know about the third acceptor?

Could it have accepted (1,A)?

Could it have accepted (2,K)?

Implementing P2c

(1,A) (2,K) ?

What do we know about the third acceptor?

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

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

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

Implementing P2c

(1,A) (2,K) nil

How many nodes do we need to consult?

Consult all 3?

Implementing P2c

(1,A) (2,K) nil

How many nodes do we need to consult?

Consult all 3? We know nothing was chosen!

Want to be non-blocking if a majority are up

Implementing P2c

(1,A) (2,K) nil

How many nodes do we need to consult?

Consult all 3? We know nothing was chosen!

Want to be non-blocking if a majority are up

Consult 1 and 2?

Consult 1 and 3?

Consult 2 and 3?

Implementing P2c

(1,A) (2,K) nil

How many nodes do we need to consult?

Consult all 3? We know nothing was chosen!

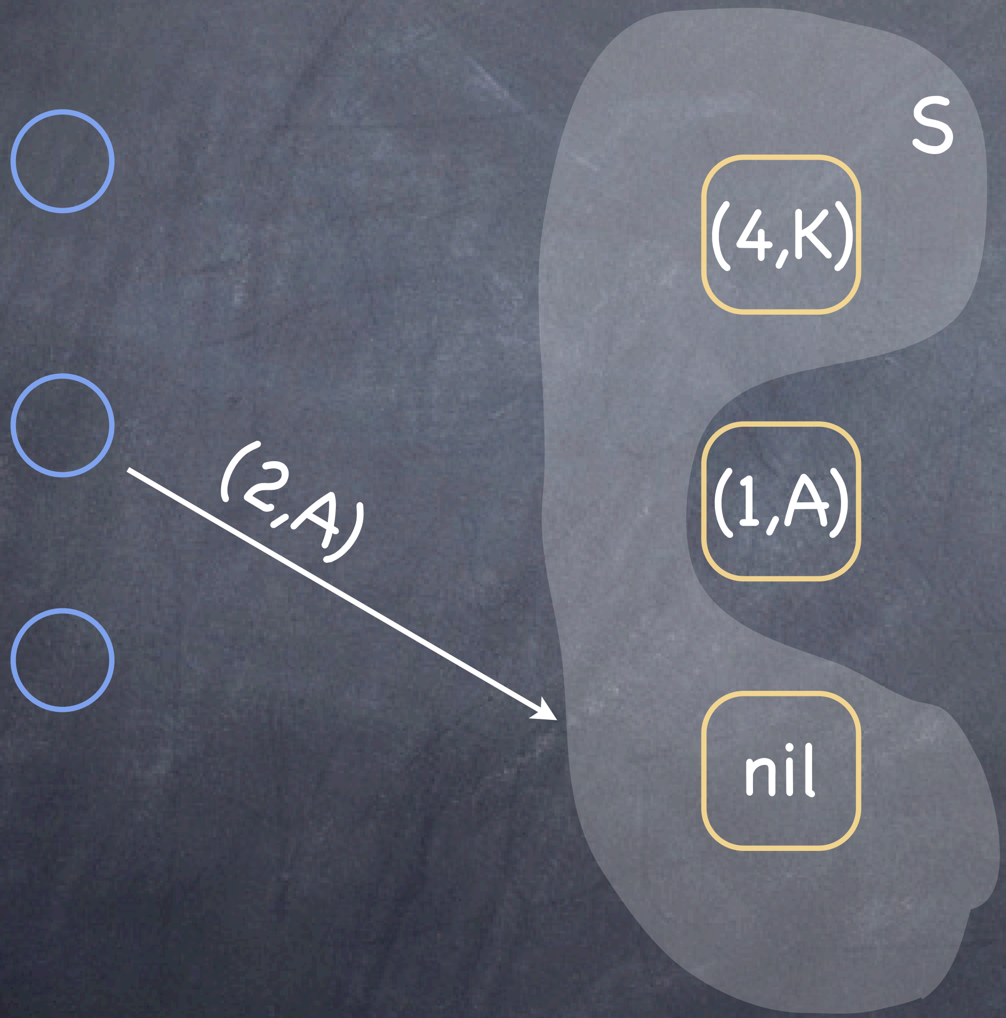
Want to be non-blocking if a majority are up

Consult 1 and 2? Safe to propose (4,K)

Consult 1 and 3? Safe to propose (4,A)

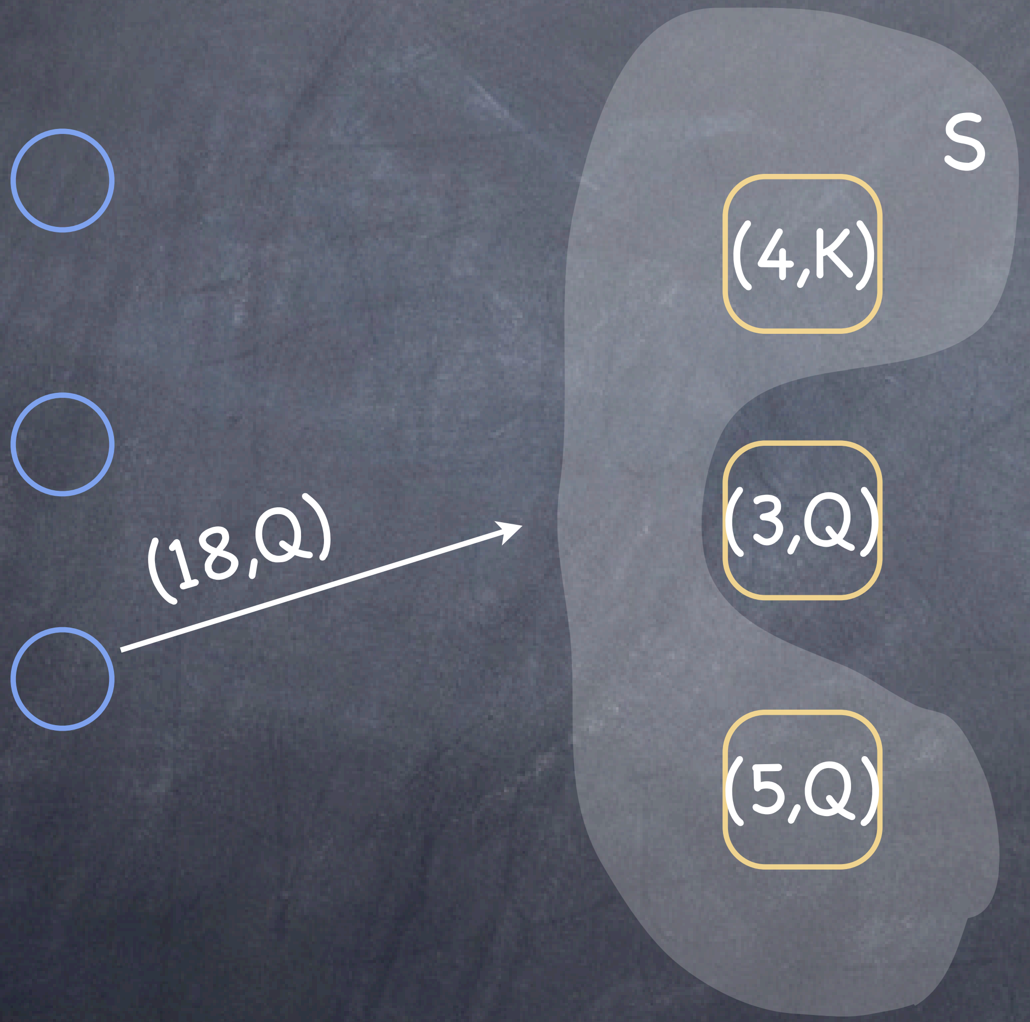
Consult 2 and 3? Safe to propose (4,K)

P2c in action



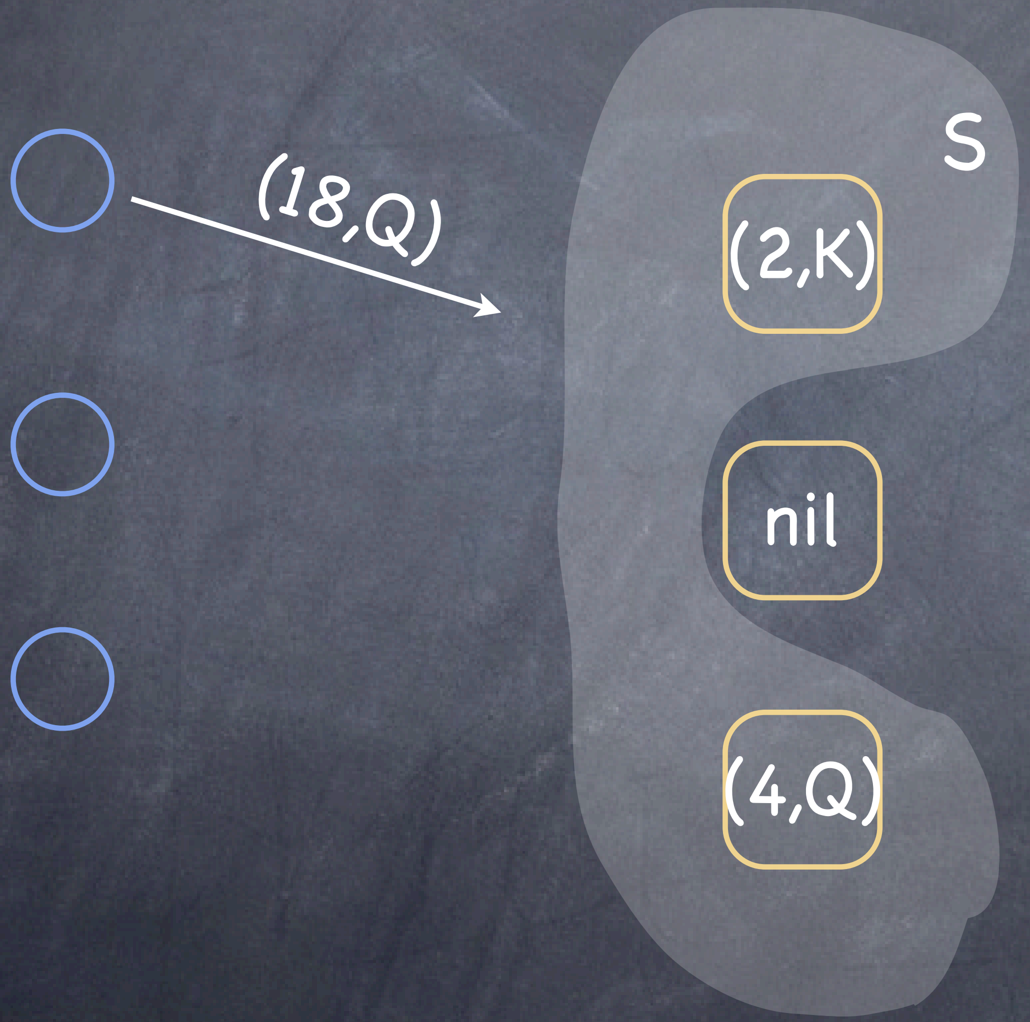
- No acceptor in S has accepted any proposal numbered less than n

P2c in action



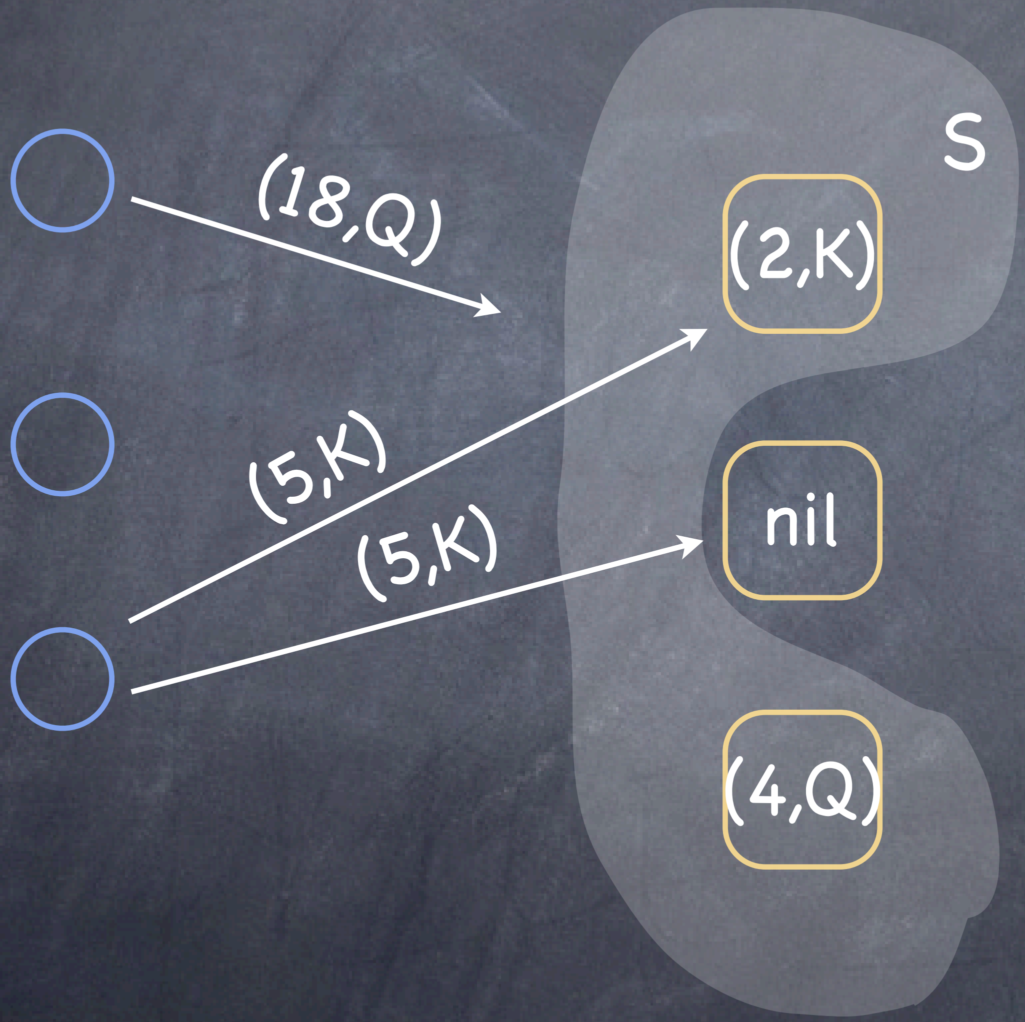
- v is the value of the highest-numbered proposal among all proposals numbered less than n and accepted by the acceptors in S

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

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

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:

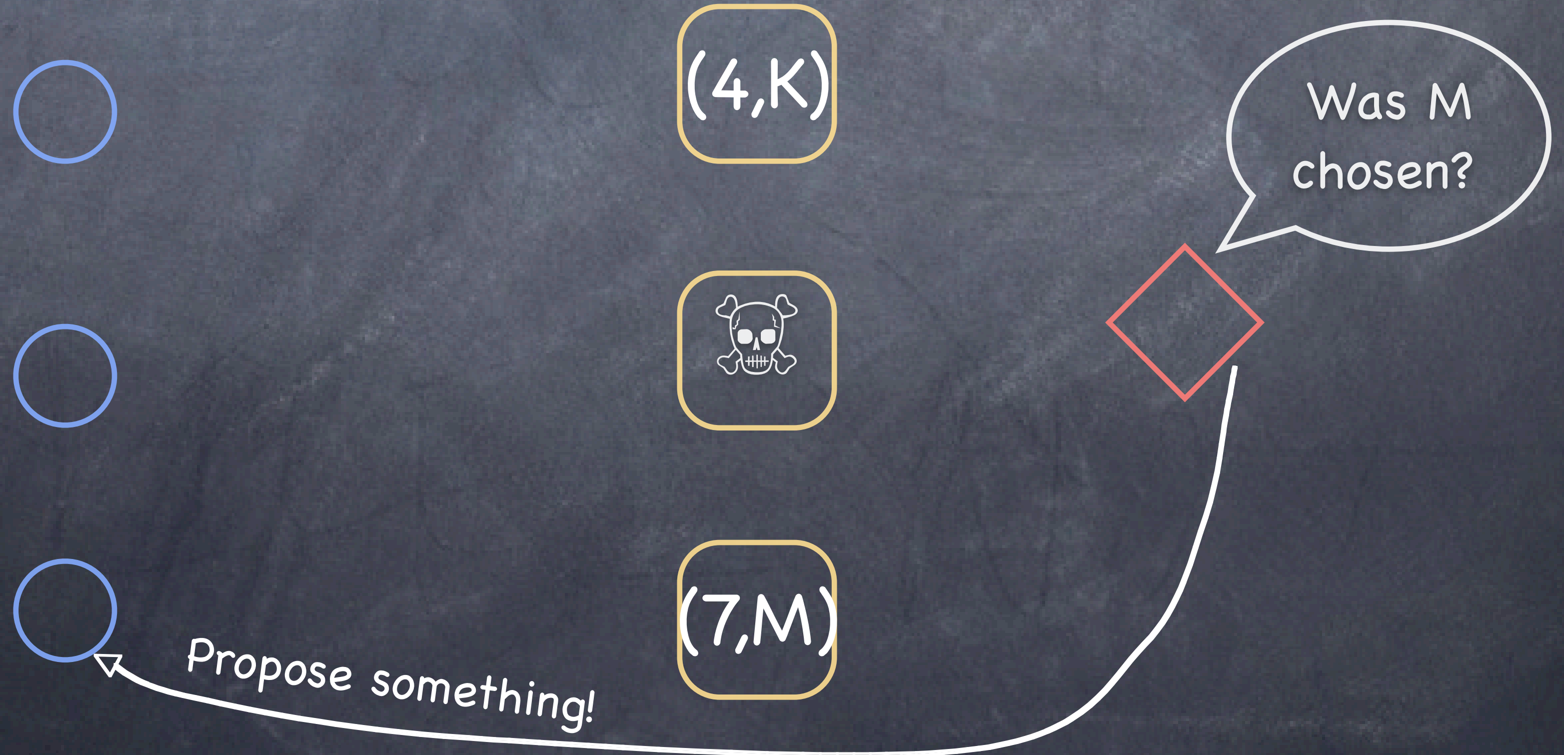
i. Each acceptor informs each learner whenever it accepts a proposal.

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:

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

p_1		p_2
$\langle propose, n_1 \rangle$	Time ↓	
		$\langle propose, n_2 \rangle$
$\langle accept(n_1, v_1) \rangle$		
$\langle propose, n_3 \rangle$		
		$\langle accept(n_2, v_2) \rangle$
		$\langle propose, n_4 \rangle$

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.

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

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