Paxos

The Part-Time Parliament

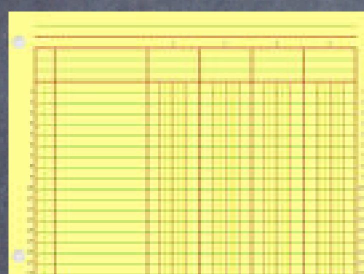
- Parliament determines laws by passing sequence of numbered decrees
- Legislators can leave and enter the chamber at arbitrary times
- No centralized record of approved decrees- instead, each legislator carries a ledger

Government 101

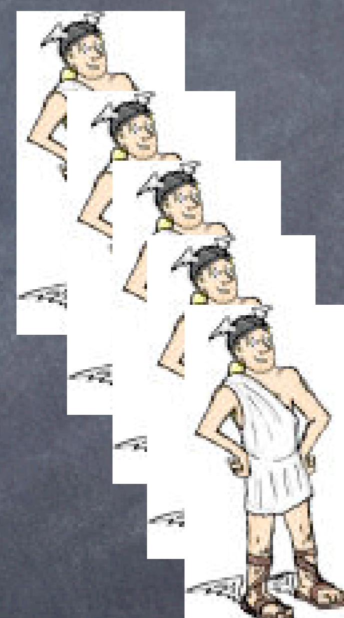
- No two ledgers contain contradictory information
- If a majority of legislators were in the Chamber and no one entered or left the Chamber for a sufficiently long time, then
  - any decree proposed by a legislator would eventually be passed
  - any passed decree would appear on the ledger of every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

Supplies

Each legislator receives



ledger



lots of messengers

scratch paper

pen with indelible ink

hourglass

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 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

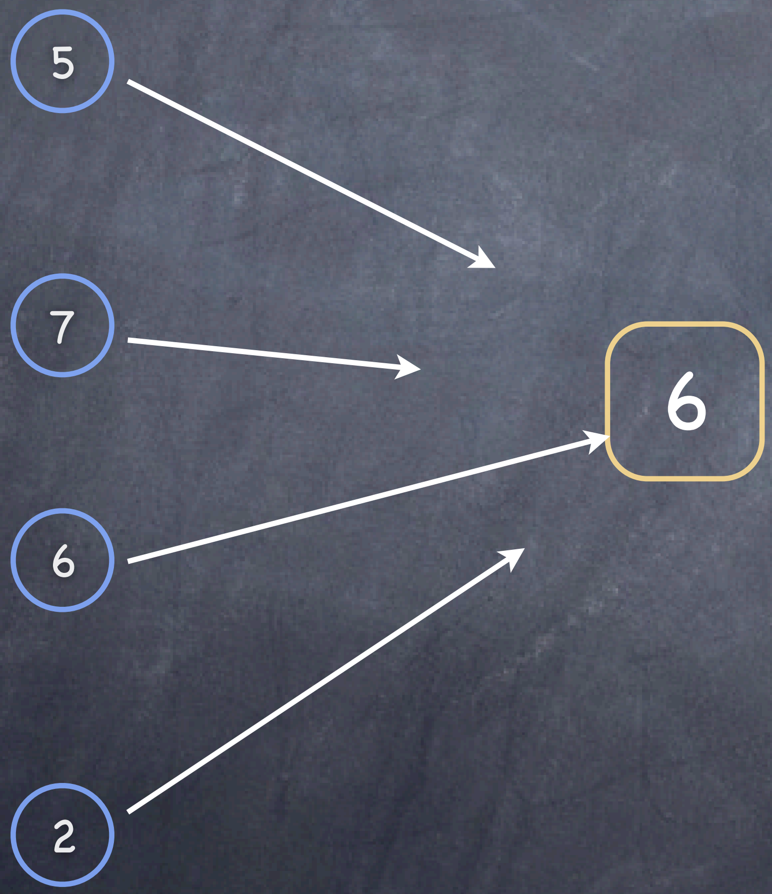
The Players

- Proposers
- Acceptors
- Learners

Choosing a value

Use a single acceptor

Choosing a value



Use a single acceptor

What if
the acceptor fails?

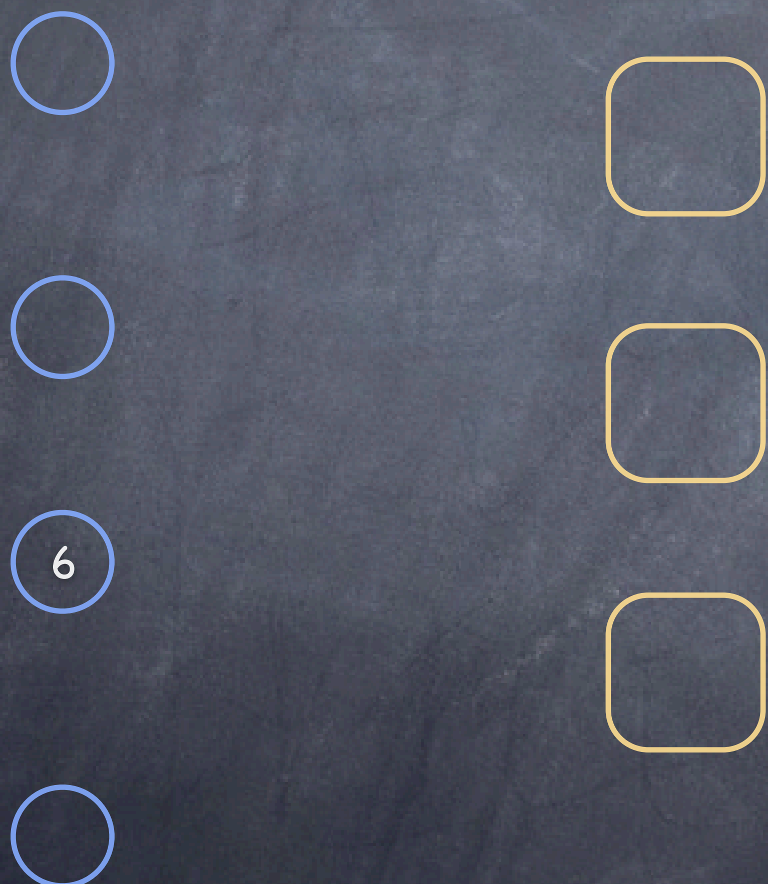
What if the acceptor fails?

- Choose only when a "large enough" set of acceptors <u>accepts</u>

What if
the acceptor fails?

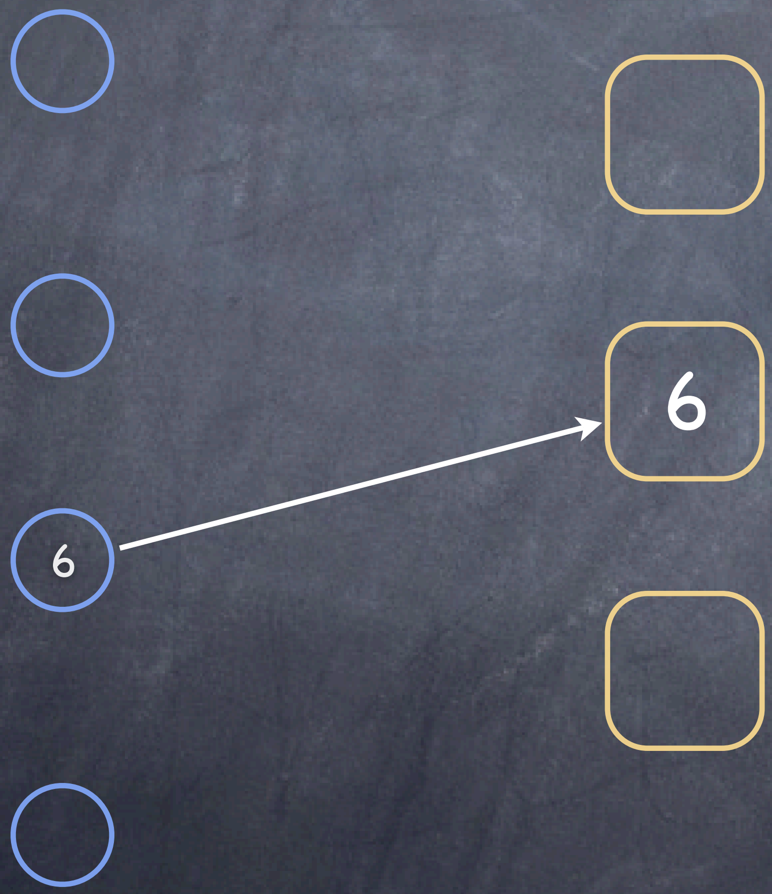
- Choose only when a “large enough” set of acceptors <u>accepts</u>
- Using a majority set guarantees that at most one value is chosen

What if the acceptor fails?



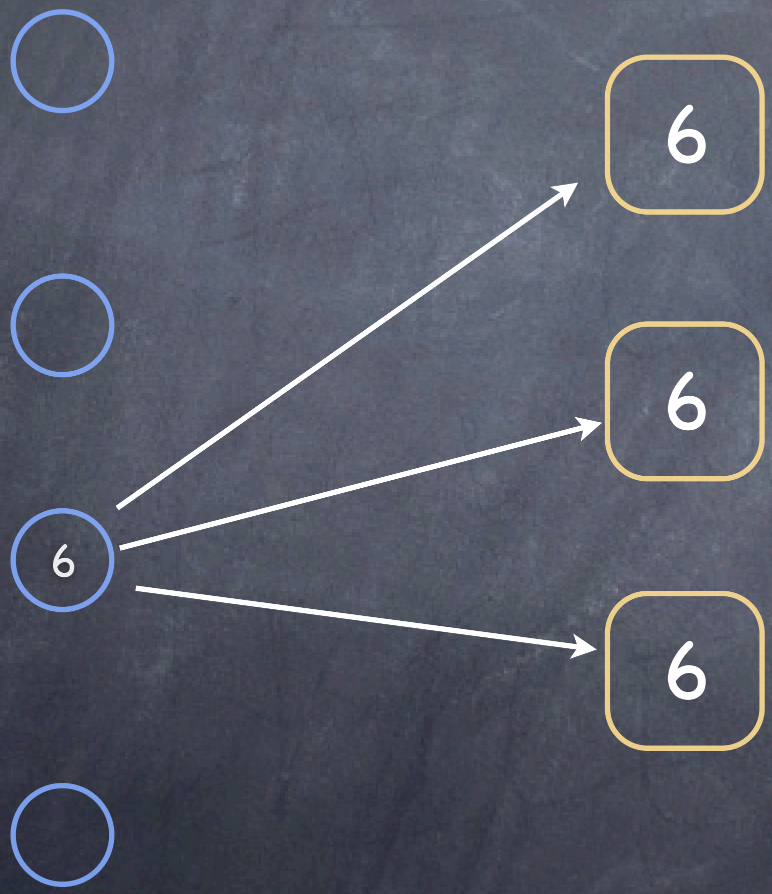
- Choose only when a "large enough" set of acceptors <u>accepts</u>
- Using a majority set guarantees that at most one value is chosen

What if the acceptor fails?



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

What if the acceptor fails?



6 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

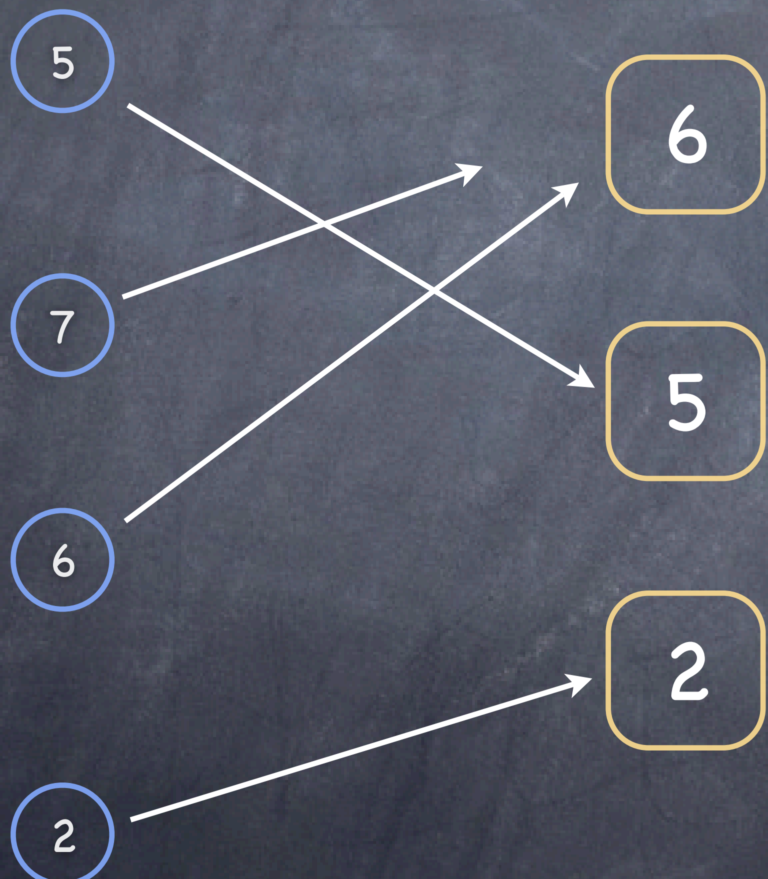
- Suppose only one value is proposed by a single proposer.
- That value should be chosen!
- First requirement:

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

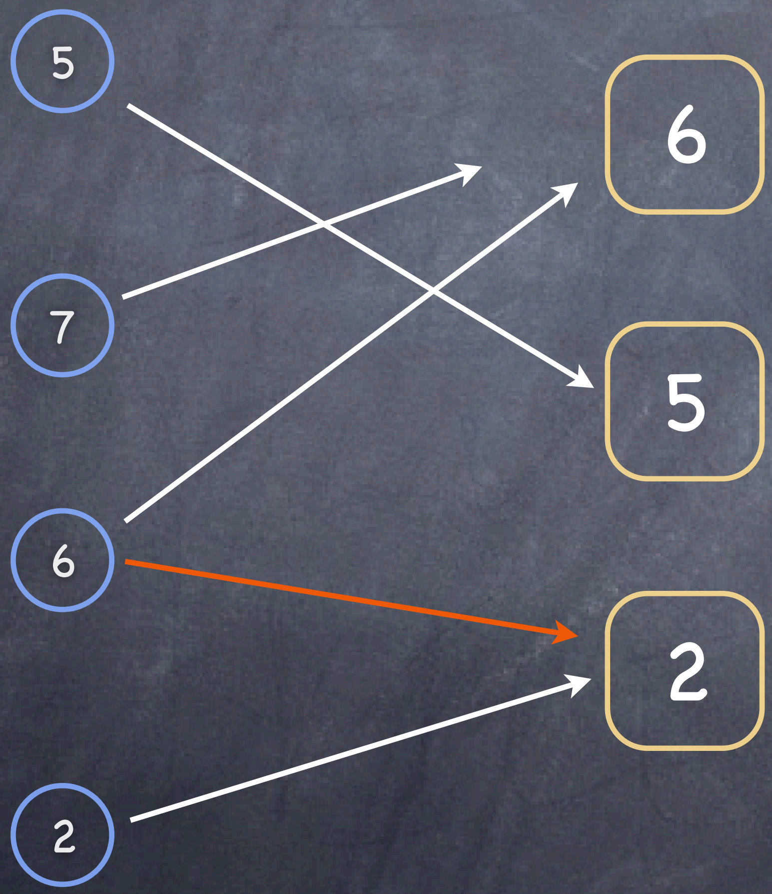
- ...but what if we have multiple proposers, each proposing a different value?

P1 + multiple proposers

P1 + multiple proposers



P1 + multiple proposers



No value is chosen!

Handling multiple proposals

- Acceptors must 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

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?

What about P1?

- Do we still need P1?

What about P1?

- Do we still need P1?

  YES, to ensure that *some* proposal is accepted

What about P1?

- Do we still need P1?

  YES, to ensure that *some* proposal is accepted

- How well do P1 and P2a play together?

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...

What about P1?

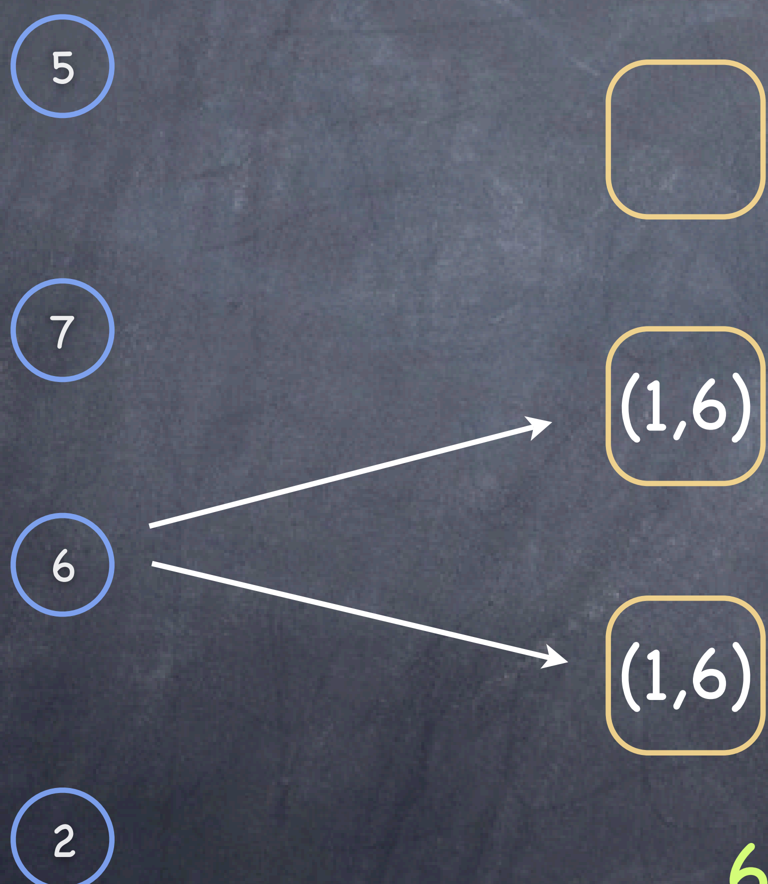
- Do we still need P1?

  YES, to ensure that *some* proposal is accepted

- How well do P1 and P2a play together?

  Asynchrony is a problem...

What about P1?



- Do we still need P1?

  YES, to ensure that *some* proposal is accepted

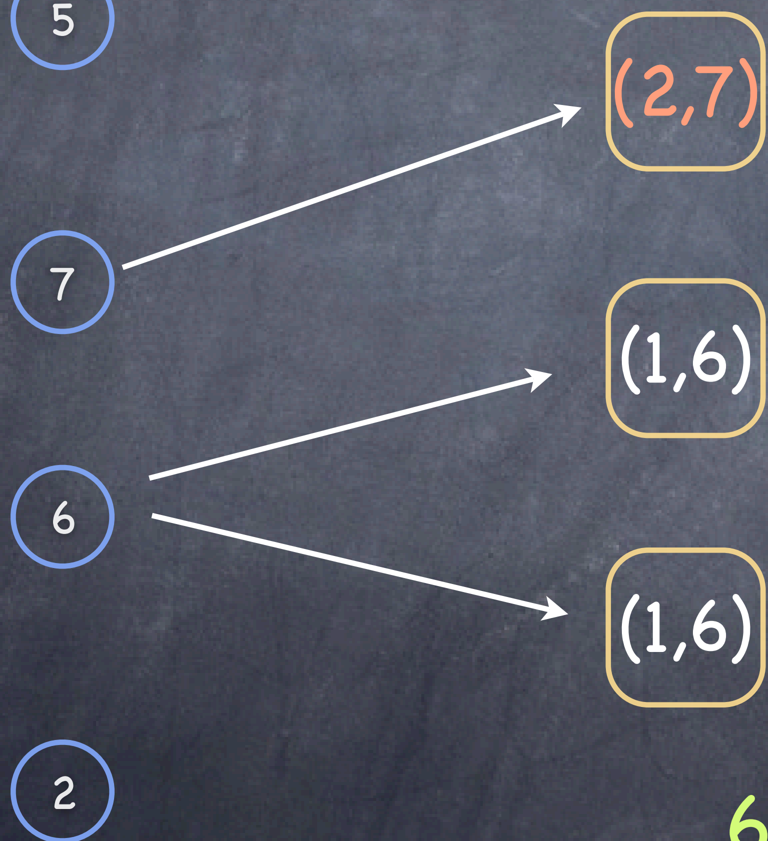
- How well do P1 and P2a play together?

  Asynchrony is a problem...

6 is chosen!

What about P1?

How does it know
it should not accept?



- 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...

6 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 always exists a majority set S where no acceptor has accepted 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)

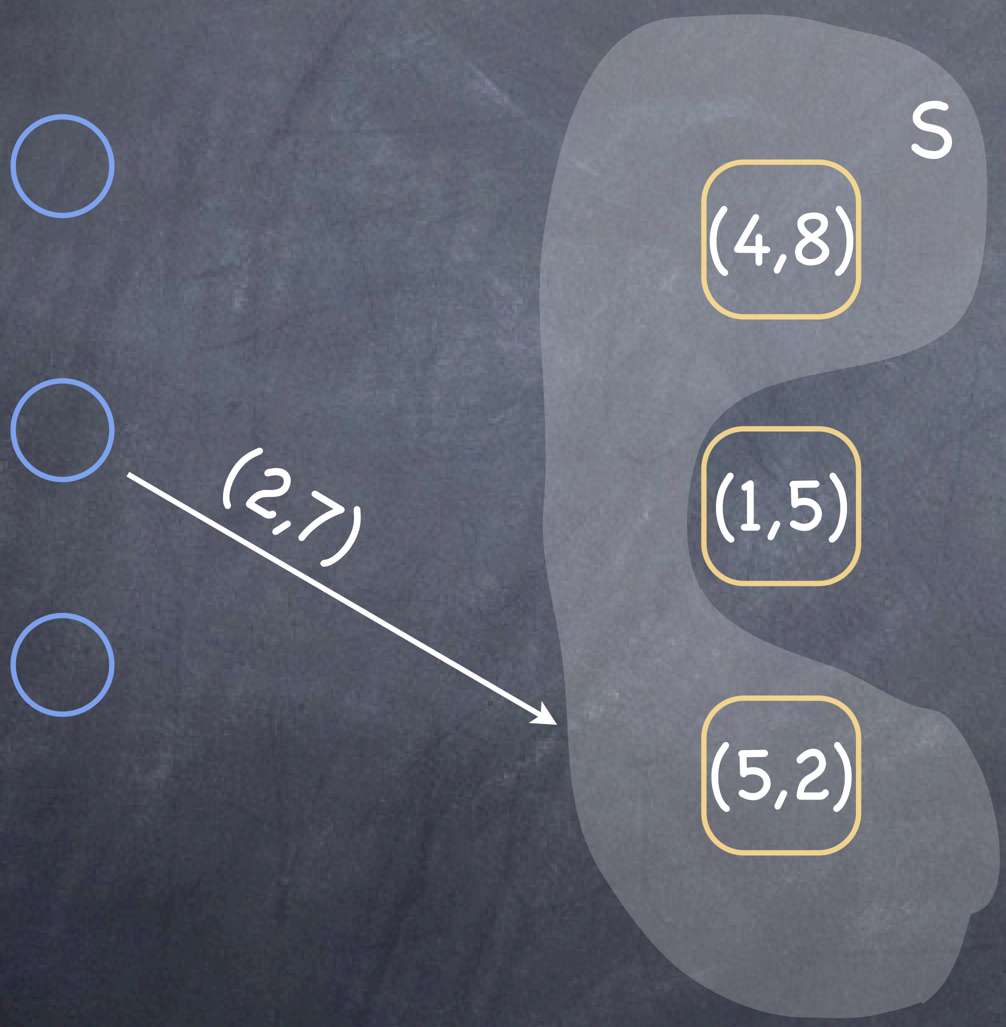
P2b: If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v

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:

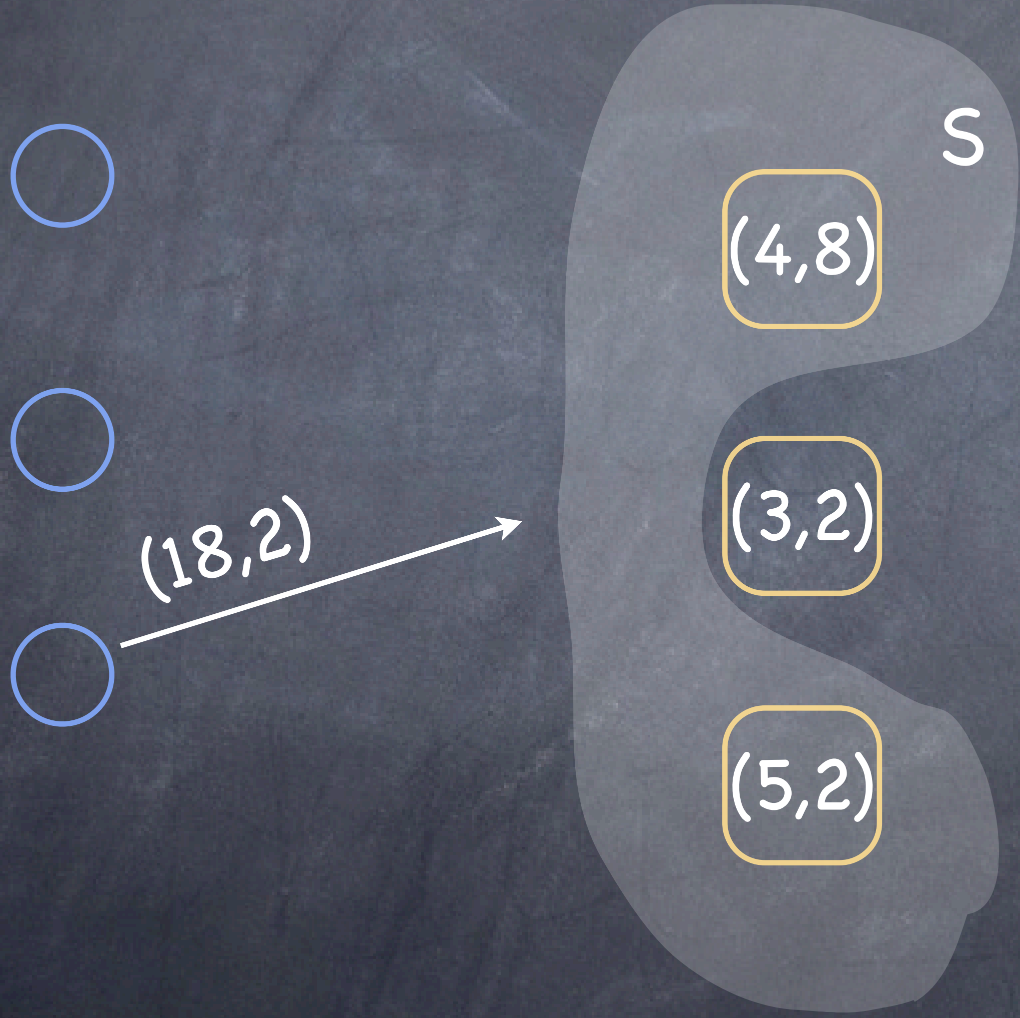
- no acceptor in S has accepted any proposal numbered less than n, or
- v is the value of the highest-numbered proposal among all proposals numbered less than n accepted by the acceptors in S

P2c in action



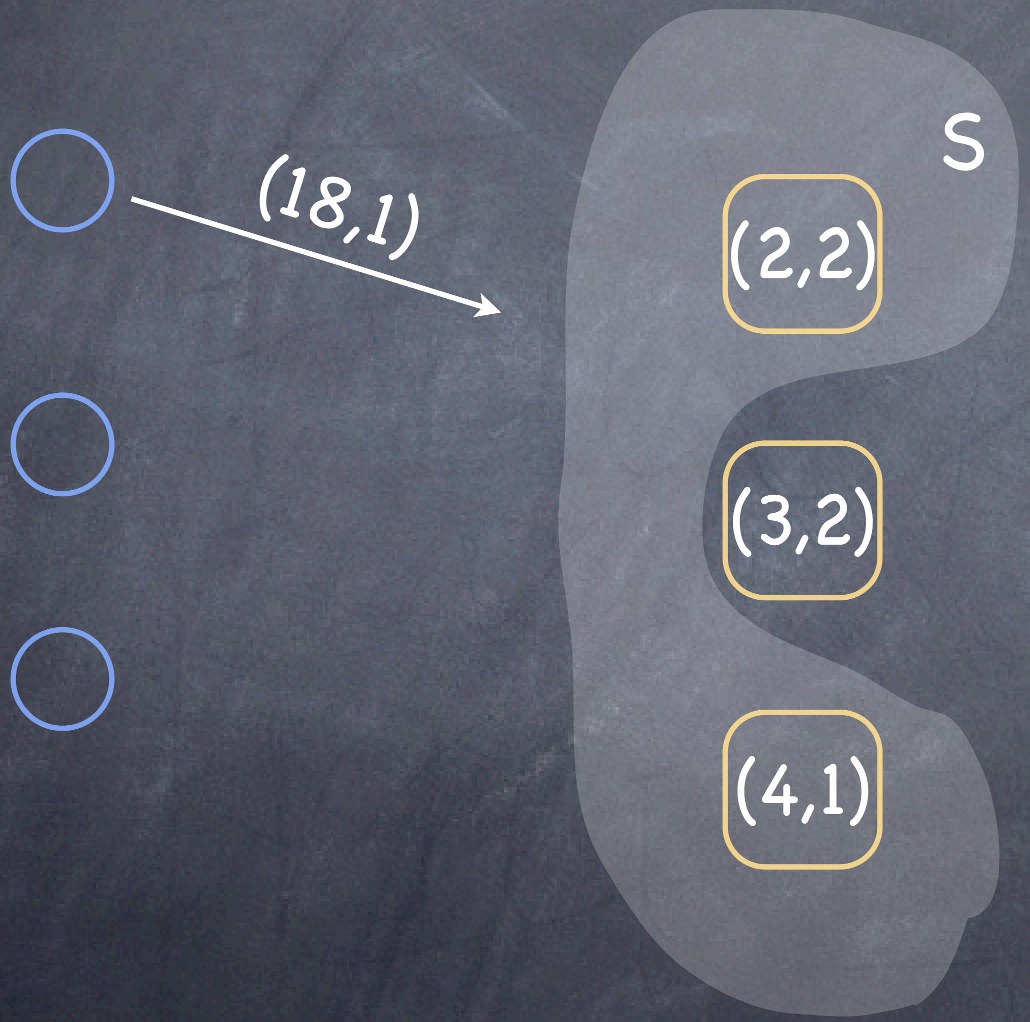
- No acceptor in S has accepted any proposal numbered less than n

P2c in action



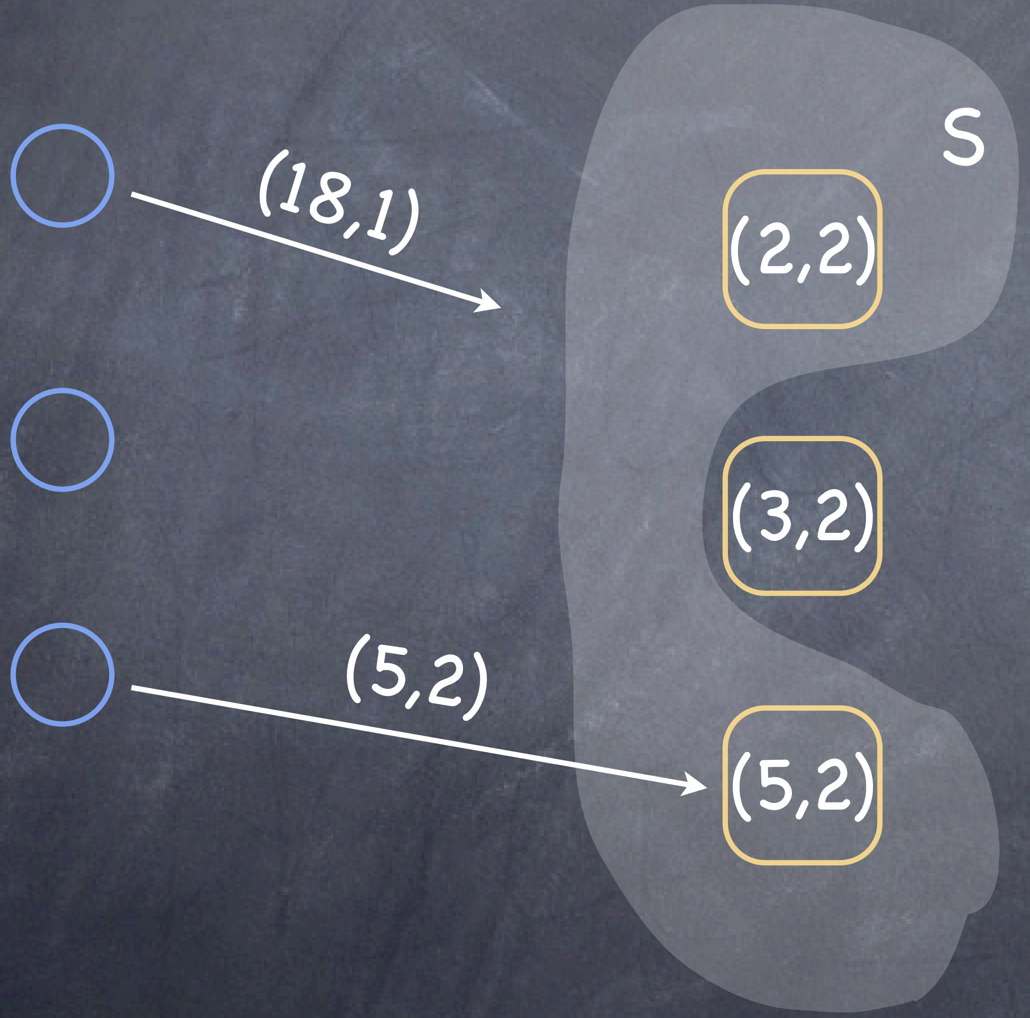
- v is the value of the highest-numbered proposal among all proposals numbered less than n and accepted by the acceptors in S

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

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 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 proposes 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

The acceptor's protocol

- An acceptor receives prepare and accept requests from proposers. It can ignore these without affecting safety.

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

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.

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

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

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

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.

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

Around FLP in 80 Slides

Condition-based Consensus

- Is it possible to identify the set of conditions on the input values under which consensus is solvable?

Condition-based Consensus

- Is it possible to identify the set of conditions on the input values under which consensus is solvable?
  - "all processes propose the same value"
  - .... ?

The Model

- n processes, $p_1, \ldots, p_n$
- At most f can crash, where $0 \leq f < n$
- Shared-memory system
- Memory is organized in arrays (e.g. $X[1, \ldots, n]$)
- $X[j]$ can be read by any p_i thorough $read(X[j])$
- $X[i]$ can only be written by p_i through $write(v, X[i])$
- p_i can atomically read X thorough $snapshot(X)$

The Problem

- Given $n, f,$ and a set of input values $\mathcal{V}$, a condition $\mathcal{C}$ defines the set of all vectors over $\mathcal{V}$ that can be proposed

- An f-fault tolerant protocol solves consensus for a condition $\mathcal{C}$ if in every execution whose input vector J belongs to $\mathcal{V}_f^n$, the protocol satisfies the following properties:

  - Validity: A decided value is a proposed value
  - Agreement: No two processes decide differently
  - BestEffort_Termination: every correct process decides if
    (i) J in $\mathcal{C}_f$ and no more than f failures <u>or</u>
    (ii) all processes are correct <u>or</u>
    (iii) a process decides

The Problem

- Given $n, f,$ and a set of input values $\mathcal{V}$, a condition $\mathcal{C}$ defines the set of all vectors over $\mathcal{V}$ that can be proposed

- An f-fault tolerant protocol solves consensus for a condition $\mathcal{C}$ if in every execution whose input vector J belongs to $\mathcal{V}^n_f$, the protocol satisfies the following properties:

  at most f $\perp$ entries

  - Validity: A decided value is a proposed value
  - Agreement: No two processes decide differently
  - BestEffort_Termination: every correct process decides if
    (i) J in $\mathcal{C}_f$ and no more than f failures <u>or</u>
    (ii) all processes are correct <u>or</u>
    (iii) a process decides

Conditions and Consensus

Theorem 1 If $\mathcal{C}$ is f-acceptable, then there exists an f-fault tolerant protocol solving consensus for $\mathcal{C}$

Acceptable Conditions

Given f and $\mathcal{V}$, let P be a predicate on $\mathcal{V}_f^n$, and S a function defined on (not necessarily all) $\mathcal{V}_f^n$

A condition $\mathcal{C}$ is acceptable if there exists P and S s.t. :

i) $T_{\mathcal{C}\to P} : I \in \mathcal{C} \Rightarrow \forall J \in \mathcal{I}_f : P(J)$

ii) $A_{P\to S} : \forall J1, J2 \in \mathcal{V}_f^n :$

$$(J1 \le J2) \wedge P(J1) \wedge P(J2) \Rightarrow S(J1) = S(J2)$$

iii) $V_{P\to S} : \forall J \in \mathcal{V}_f^n : P(J) \Rightarrow S(J) =$ a non-$\perp$ value of J

Given two vectors A and B, we write A ≤ B if

$$\forall k : A[k] \neq \perp \Rightarrow A[k] = B[k]$$

The Protocol

(1) $write(v_i, V[i])$

(2) repeat $V_i \leftarrow snapshot(V)$ until $|V_i| \geq n - f$

(3) if $P(V_i)$ then $w_i \leftarrow S(V_i)$ else $w_i \leftarrow \top$

(4) $write(w_i, W[i])$

(5) repeat $\forall j \in [1, \ldots, n]$ do $W_i[j] \leftarrow read(W[j])$

(6) if $\exists j : W_i[j] \neq \bot, \top$ then return$(W_i[j])$

(7) until $(\bot \notin W_i)$

(8) $\forall j \in [1, \ldots, n]$ do $Y_i[j] \leftarrow read(V[j])$

(9) return$(F(Y_i))$

Two arrays of atomic registers

$$V[1, \ldots, n] := [\bot, \ldots, \bot]$$

$$W[1, \ldots, n] := [\bot, \ldots, \bot]$$

The Protocol

$\mathit{write}(v_i, V[i])$

repeat $V_i \leftarrow \mathit{snapshot}(V)$ until $|V_i| \geq n-f$

if $P(V_i)$ then $w_i \leftarrow S(V_i)$ else $w_i \leftarrow \top$

$\mathit{write}(w_i, W[i])$

repeat $\forall j \in [1, \ldots, n]$ do $W_i[j] \leftarrow \mathit{read}(W[j])$

    if $\exists j : W_i[j] \neq \perp, \top$ then return$(W_i[j])$

until $(\perp \notin W_i)$

$\forall j \in [1, \ldots, n]$ do $Y_i[j] \leftarrow \mathit{read}(V[j])$

return$(F(Y_i))$

Two arrays of atomic registers

$$V[1, \ldots, n] := [\perp, \ldots, \perp]$$

$$W[1, \ldots, n] := [\perp, \ldots, \perp]$$

- p_i writes its input in V_i
- p_i repeatedly snapshots V until $n-f$ processes have written their input values in V

The Protocol

(1) $write(v_i, V[i])$

(2) repeat $V_i \leftarrow snapshot(V)$ until $|V_i| \geq n-f$

(3) if $P(V_i)$ then $w_i \leftarrow S(V_i)$ else $w_i \leftarrow \top$

(4) $write(w_i, W[i])$

(5) repeat $\forall j \in [1, \ldots, n]$ do $W_i[j] \leftarrow read(W[j])$

(6) if $\exists j : W_i[j] \neq \perp, \top$ then return$(W_i[j])$

(7) until $(\perp \notin W_i)$

(8) $\forall j \in [1, \ldots, n]$ do $Y_i[j] \leftarrow read(V[j])$

(9) return$(F(Y_i))$

Two arrays of atomic registers

$$V[1, \ldots, n] := [\perp, \ldots, \perp]$$

$$W[1, \ldots, n] := [\perp, \ldots, \perp]$$

- p_i tries to decide, evaluating P
- If P holds, then p_i can decide $w_i = S(V_i)$, otherwise it decides $\top$
- In either case, p_i writes its decision value to W_i to help other processes decide

The Protocol

$write(v_i, V[i])$
repeat $V_i \leftarrow snapshot(V)$ until $|V_i| \geq n - f$
if $P(V_i)$ then $w_i \leftarrow S(V_i)$ else $w_i \leftarrow \top$
$write(w_i, W[i])$
repeat $\forall j \in [1, \ldots, n]$ do $W_i[j] \leftarrow read(W[j])$
    if $\exists j : W_i[j] \neq \perp, \top$ then return$(W_i[j])$
until $(\perp \notin W_i)$
$\forall j \in [1, \ldots, n]$ do $Y_i[j] \leftarrow read(V[j])$
return$(F(Y_i))$

Two arrays of atomic registers

$$V[1, \ldots, n] := [\perp, \ldots, \perp]$$

$$W[1, \ldots, n] := [\perp, \ldots, \perp]$$

- p_i enters a loop, looking for a decision value other than $\perp, \top$
- It may never find it: but if p_i detects all $\top$, it can still decide!

The Protocol

$write(v_i, V[i])$

repeat $V_i \leftarrow snapshot(V)$ until $|V_i| \geq n-f$

if $P(V_i)$ then $w_i \leftarrow S(V_i)$ else $w_i \leftarrow \top$

$write(w_i, W[i])$

repeat $\forall j \in [1, \ldots, n]$ do $W_i[j] \leftarrow read(W[j])$

  if $\exists j : W_i[j] \neq \perp, \top$ then return$(W_i[j])$

until $(\perp \notin W_i)$

$\forall j \in [1, \ldots, n]$ do $Y_i[j] \leftarrow read(V[j])$

return$(F(Y_i))$

Two arrays of atomic registers

$$V[1, \ldots, n] := [\perp, \ldots, \perp]$$

$$W[1, \ldots, n] := [\perp, \ldots, \perp]$$

- p_i enters a loop, looking for a decision value other than $\perp, \top$
- It may never find it: but if p_i detects all $\top$, it can still decide!
- all p_j must have written their input v_j to V
- p_i decides by applying a deterministic F to V
- Note: termination is not guaranteed!

Termination

```
write(v_i, V[i])
repeat V_i ← snapshot(V) until |V_i| ≥ n−f
if P(V_i) then w_i ← S(V_i) else w_i ← ⊤
write(w_i, W[i])
repeat ∀j ∈ [1,...,n] do W_i[j] ← read(W[j])
        if ∃j : W_i[j] ≠ ⊥,⊤ then return(W_i[j])
until (⊥ ∉ W_i)
∀j ∈ [1,...,n] do Y_i[j] ← read(V[j])
return(F(Y_i))
```

$i)\ \ T_{\mathcal{C}\to P} : I \in \mathcal{C} \Rightarrow \forall J \in \mathcal{I}_f : P(J)$

$ii)\ \ A_{P\to S} : \forall J1, J2 \in \mathcal{V}_f^n :$
$(J1 \leq J2) \wedge P(J1) \wedge P(J2) \Rightarrow S(J1) = S(J2)$

$iii)\ \ V_{P\to S} : \forall J \in \mathcal{V}_f^n : P(J) \Rightarrow S(J) = \text{a non-}\bot\text{ value of } J$

BestEffort_Termination: every correct process decides if

(i) J in $\mathcal{C}_f$ and no more than f failures <u>or</u>

<u>(ii)</u> all processes are correct <u>or</u>

(iii) a process decides

Lemma 1 The protocol satisfies (i)

Proof. Let p_i be a correct process

Termination

(1) $write(v_i, V[i])$

(2) repeat $V_i \leftarrow snapshot(V)$ until $|V_i| \geq n - f$

(3) if $P(V_i)$ then $w_i \leftarrow S(V_i)$ else $w_i \leftarrow \top$

(4) $write(w_i, W[i])$

(5) repeat $\forall j \in [1, \ldots, n]$ do $W_i[j] \leftarrow read(W[j])$

(6) if $\exists j : W_i[j] \neq \perp, \top$ then return$(W_i[j])$

(7) until $(\perp \notin W_i)$

(8) $\forall j \in [1, \ldots, n]$ do $Y_i[j] \leftarrow read(V[j])$

(9) return$(F(Y_i))$

$i)\ T_{\mathcal{C} \to P} : I \in \mathcal{C} \Rightarrow \forall J \in \mathcal{I}_f : P(J)$

$ii)\ A_{P \to S} : \forall J1, J2 \in \mathcal{V}_f^n :$
$(J1 \leq J2) \wedge P(J1) \wedge P(J2) \Rightarrow S(J1) = S(J2)$

$iii)\ V_{P \to S} : \forall J \in \mathcal{V}_f^n : P(J) \Rightarrow S(J) =$ a non-$\perp$ value of J

BestEffort_Termination: every correct process decides if

(i) J in $\mathcal{C}_f$ and no more than f failures <u>or</u>

<u>(ii)</u> all processes are correct <u>or</u>

(iii) a process decides

Lemma 1 The protocol satisfies (i)

Proof. Let p_i be a correct process

- p_i does not block at (2) and therefore gets $V_i \leq J$
- Since $J \in \mathcal{C}_f$, then $V_i \in \mathcal{C}_f$: from $T_{\mathcal{C} \to P}$, $P(V_i)$ is true
- At (3), $w_i \neq \perp, \top$ and at (6), at least $W_i[i] \neq \perp, \top$

Termination

(1) $write(v_i, V[i])$

(2) repeat $V_i \leftarrow snapshot(V)$ until $|V_i| \geq n - f$

(3) if $P(V_i)$ then $w_i \leftarrow S(V_i)$ else $w_i \leftarrow \top$

(4) $write(w_i, W[i])$

(5) repeat $\forall j \in [1, \ldots, n]$ do $W_i[j] \leftarrow read(W[j])$

(6) if $\exists j : W_i[j] \neq \perp, \top$ then return$(W_i[j])$

(7) until $(\perp \notin W_i)$

(8) $\forall j \in [1, \ldots, n]$ do $Y_i[j] \leftarrow read(V[j])$

(9) return$(F(Y_i))$

$i)\ \ T_{\mathcal{C} \to P} : I \in \mathcal{C} \Rightarrow \forall J \in \mathcal{I}_f : P(J)$

$ii)\ \ A_{P \to S} : \forall J1, J2 \in \mathcal{V}_f^n :$
$(J1 \leq J2) \wedge P(J1) \wedge P(J2) \Rightarrow S(J1) = S(J2)$

$iii)\ \ V_{P \to S} : \forall J \in \mathcal{V}_f^n : P(J) \Rightarrow S(J) = \text{a non-}\perp \text{ value of } J$

BestEffort_Termination: every correct process decides if

(i) J in $\mathcal{C}_f$ and no more than f failures or

(ii) all processes are correct or

(iii) a process decides

Lemma 2 The protocol satisfies (ii)

Proof. Assume all processes are correct

- They all exit the loop at (2)
- If they all find $\neg P(V_i)$, they all read $\top$ at (5) and decide at (9)

Termination

(1) $write(v_i, V[i])$

(2) repeat $V_i \leftarrow snapshot(V)$ until $|V_i| \geq n - f$

(3) if $P(V_i)$ then $w_i \leftarrow S(V_i)$ else $w_i \leftarrow \top$

(4) $write(w_i, W[i])$

(5) repeat $\forall j \in [1, \ldots, n]$ do $W_i[j] \leftarrow read(W[j])$

(6) if $\exists j : W_i[j] \neq \perp, \top$ then return$(W_i[j])$

(7) until $(\perp \notin W_i)$

(8) $\forall j \in [1, \ldots, n]$ do $Y_i[j] \leftarrow read(V[j])$

(9) return$(F(Y_i))$

$i)\ \ T_{\mathcal{C} \rightarrow P} : I \in \mathcal{C} \Rightarrow \forall J \in \mathcal{I}_f : P(J)$

$ii)\ \ A_{P \rightarrow S} : \forall J1, J2 \in \mathcal{V}_f^n :$
$(J1 \leq J2) \wedge P(J1) \wedge P(J2) \Rightarrow S(J1) = S(J2)$

$iii)\ \ V_{P \rightarrow S} : \forall J \in \mathcal{V}_f^n : P(J) \Rightarrow S(J) = \text{a non-}\perp \text{ value of } J$

BestEffort_Termination: every correct process decides if

.....

(iii) a process decides

Lemma 3 The protocol satisfies (iii)

Proof. Assume p_i decides

- p_i (and all correct processes) exit the loop at (2)
- If p_i decides at (6) on $W_i[j] \neq \top, \perp$, then all correct processes will find the same value and decide (6)
- If p_i decides at (9), every process wrote $\top$ at (4) and every correct process terminates at (9)

Agreement

(1) $write(v_i, V[i])$

(2) repeat $V_i \leftarrow snapshot(V)$ until $|V_i| \geq n - f$

(3) if $P(V_i)$ then $w_i \leftarrow S(V_i)$ else $w_i \leftarrow \top$

(4) $write(w_i, W[i])$

(5) repeat $\forall j \in [1, \ldots, n]$ do $W_i[j] \leftarrow read(W[j])$

(6) if $\exists j : W_i[j] \neq \bot, \top$ then return$(W_i[j])$

(7) until $(\bot \notin W_i)$

(8) $\forall j \in [1, \ldots, n]$ do $Y_i[j] \leftarrow read(V[j])$

(9) return$(F(Y_i))$

$i)\ T_{\mathcal{C} \rightarrow P} : I \in \mathcal{C} \Rightarrow \forall J \in \mathcal{I}_f : P(J)$

$ii)\ A_{P \rightarrow S} : \forall J1, J2 \in \mathcal{V}_f^n :$
$(J1 \leq J2) \wedge P(J1) \wedge P(J2) \Rightarrow S(J1) = S(J2)$

$iii)\ V_{P \rightarrow S} : \forall J \in \mathcal{V}_f^n : P(J) \Rightarrow S(J) = \text{a non-}\bot \text{ value of } J$

Lemma 4 Either all processes that decide do so at (6) or at (9)

Proof. Suppose p_i decides at (6)

- For some j, $W[j] \neq \bot, \top$
- No process can exit at (7) because its W contained only $\top$
- If a process decides, it does so at (6)

Agreement

(1) $write(v_i, V[i])$

(2) repeat $V_i \leftarrow snapshot(V)$ until $|V_i| \geq n - f$

(3) if $P(V_i)$ then $w_i \leftarrow S(V_i)$ else $w_i \leftarrow \top$

(4) $write(w_i, W[i])$

(5) repeat $\forall j \in [1, \ldots, n]$ do $W_i[j] \leftarrow read(W[j])$

(6) if $\exists j : W_i[j] \neq \bot, \top$ then return$(W_i[j])$

(7) until $(\bot \notin W_i)$

(8) $\forall j \in [1, \ldots, n]$ do $Y_i[j] \leftarrow read(V[j])$

(9) return$(F(Y_i))$

$i)$ $T_{\mathcal{C} \to P} : I \in \mathcal{C} \Rightarrow \forall J \in \mathcal{I}_f : P(J)$

$ii)$ $A_{P \to S} : \forall J1, J2 \in \mathcal{V}_f^n :$
$(J1 \leq J2) \wedge P(J1) \wedge P(J2) \Rightarrow S(J1) = S(J2)$

$iii)$ $V_{P \to S} : \forall J \in \mathcal{V}_f^n : P(J) \Rightarrow S(J) =$ a non-$\bot$ value of J

Lemma 4 Either all processes that decide do so at (6) or at (9)

Proof. Suppose p_i decides at (9)

- p_i did exit the loop at (7)
- Every process evaluated P to false and wrote $\top$ to W in (4)
- No process can decide at (6)

Agreement

(1) $write(v_i, V[i])$

(2) repeat $V_i \leftarrow snapshot(V)$ until $|V_i| \geq n-f$

(3) if $P(V_i)$ then $w_i \leftarrow S(V_i)$ else $w_i \leftarrow \top$

(4) $write(w_i, W[i])$

(5) repeat $\forall j \in [1, \ldots, n]$ do $W_i[j] \leftarrow read(W[j])$

(6) if $\exists j : W_i[j] \neq \perp, \top$ then return$(W_i[j])$

(7) until $(\perp \notin W_i)$

(8) $\forall j \in [1, \ldots, n]$ do $Y_i[j] \leftarrow read(V[j])$

(9) return$(F(Y_i))$

$i)\ \ T_{\mathcal{C} \to P} : I \in \mathcal{C} \Rightarrow \forall J \in \mathcal{I}_f : P(J)$

$ii)\ \ A_{P \to S} : \forall J1, J2 \in \mathcal{V}_f^n :$
$(J1 \leq J2) \wedge P(J1) \wedge P(J2) \Rightarrow S(J1) = S(J2)$

$iii)\ \ V_{P \to S} : \forall J \in \mathcal{V}_f^n : P(J) \Rightarrow S(J) = \text{a non-}\perp \text{ value of } J$

Lemma 5 No two processes decide differently (Agreement)

Proof. Consider p_i, p_j that decide

- By Lemma 4, they decide on the same line–let it be (6)
- $\exists V_\ell, V_k : S(V_\ell) = w_\ell \neq \perp, \top$ and $S(V_k) = w_k \neq \perp, \top$
- Both $P(V_\ell)$ and $P(V_k)$ hold (1)
- V_ℓ and V_k come from snapshots. Hence $V_\ell \leq V_k \vee V_k \leq V_\ell$ (2)
- From (1), (2), and $A_{P \to S}$: $S(V_\ell) = S(V_k)$ and $w_\ell = w_k$

Agreement

(1) $write(v_i, V[i])$

(2) repeat $V_i \leftarrow snapshot(V)$ until $|V_i| \geq n - f$

(3) if $P(V_i)$ then $w_i \leftarrow S(V_i)$ else $w_i \leftarrow \top$

(4) $write(w_i, W[i])$

(5) repeat $\forall j \in [1, \ldots, n]$ do $W_i[j] \leftarrow read(W[j])$

(6) if $\exists j : W_i[j] \neq \bot, \top$ then return$(W_i[j])$

(7) until $(\bot \notin W_i)$

(8) $\forall j \in [1, \ldots, n]$ do $Y_i[j] \leftarrow read(V[j])$

(9) return$(F(Y_i))$

$i)$ $T_{\mathcal{C} \rightarrow P} : I \in \mathcal{C} \Rightarrow \forall J \in \mathcal{I}_f : P(J)$

$ii)$ $A_{P \rightarrow S} : \forall J1, J2 \in \mathcal{V}_f^n :$
$(J1 \leq J2) \wedge P(J1) \wedge P(J2) \Rightarrow S(J1) = S(J2)$

$iii)$ $V_{P \rightarrow S} : \forall J \in \mathcal{V}_f^n : P(J) \Rightarrow S(J) = \text{a non-}\bot \text{ value of } J$

Lemma 5 No two processes decide differently (Agreement)

Proof. Consider p_i, p_j that decide

- By Lemma 4, they decide on the same line–let it be (9)
- Each p_ℓ has executed (4): $W[\ell] \neq \bot$
- Each p_ℓ has executed (1): $V[\ell] = v_\ell$
- Hence $Y_i = Y_j = (v_1, \ldots, v_n)$
- Since both processors apply the same deterministic F, agreement follows

Validity

(1) $write(v_i, V[i])$

(2) repeat $V_i \leftarrow snapshot(V)$ until $|V_i| \geq n-f$

(3) if $P(V_i)$ then $w_i \leftarrow S(V_i)$ else $w_i \leftarrow \top$

(4) $write(w_i, W[i])$

(5) repeat $\forall j \in [1, \ldots, n]$ do $W_i[j] \leftarrow read(W[j])$

(6) if $\exists j : W_i[j] \neq \bot, \top$ then return$(W_i[j])$

(7) until $(\bot \notin W_i)$

(8) $\forall j \in [1, \ldots, n]$ do $Y_i[j] \leftarrow read(V[j])$

(9) return$(F(Y_i))$

$i)\ \ T_{\mathcal{C} \to P} : I \in \mathcal{C} \Rightarrow \forall J \in \mathcal{I}_f : P(J)$

$ii)\ \ A_{P \to S} : \forall J1, J2 \in \mathcal{V}_f^n :$
$(J1 \leq J2) \wedge P(J1) \wedge P(J2) \Rightarrow S(J1) = S(J2)$

$iii)\ \ V_{P \to S} : \forall J \in \mathcal{V}_f^n : P(J) \Rightarrow S(J) = \text{a non-}\bot \text{ value of } J$

Lemma 6 A decided value is a proposed value (Validity)

Proof. Suppose p_i at (6) decides
$W_i[j] = w_j \neq \bot, \top$

- Then, by (3), $P(V_j)$ holds and, from $V_{P \to S}$, $w_j = S(V_j) =$ a non-$\bot$ value of J

Validity

(1) $write(v_i, V[i])$

(2) repeat $V_i \leftarrow snapshot(V)$ until $|V_i| \geq n-f$

(3) if $P(V_i)$ then $w_i \leftarrow S(V_i)$ else $w_i \leftarrow \top$

(4) $write(w_i, W[i])$

(5) repeat $\forall j \in [1, \ldots, n]$ do $W_i[j] \leftarrow read(W[j])$

(6) if $\exists j : W_i[j] \neq \perp, \top$ then return$(W_i[j])$

(7) until $(\perp \notin W_i)$

(8) $\forall j \in [1, \ldots, n]$ do $Y_i[j] \leftarrow read(V[j])$

(9) return$(F(Y_i))$

$i)\ T_{\mathcal{C}\rightarrow P} : I \in \mathcal{C} \Rightarrow \forall J \in \mathcal{I}_f : P(J)$

$ii)\ A_{P\rightarrow S} : \forall J1, J2 \in \mathcal{V}_f^n :$
$(J1 \leq J2) \wedge P(J1) \wedge P(J2) \Rightarrow S(J1) = S(J2)$

$iii)\ V_{P\rightarrow S} : \forall J \in \mathcal{V}_f^n : P(J) \Rightarrow S(J) = \text{a non-}\perp \text{ value of } J$

Lemma 6 A decided value is a proposed value (Validity)

Proof. Suppose p_i decides at (9)

- Then, by (7), $\forall j : W_i[j] \neq \perp$
- All p_j have written v_j into $V[j]$
- Hence, $Y_i = [v_1, \ldots, v_n]$
- Since F outputs a value of Y_i, Validity follows

It gets really cool...

- Theorem 1 If $\mathcal{C}$ is f-acceptable, then there exists an f-fault tolerant protocol solving consensus for $\mathcal{C}$

It gets really cool...

- Theorem 1 If $\mathcal{C}$ is f-acceptable, then there exists an f-fault tolerant protocol solving consensus for $\mathcal{C}$
- Theorem 2 If there exists an f-fault tolerant protocol solving consensus for $\mathcal{C}$, then $\mathcal{C}$ is f-acceptable

So, how do these conditions look like?

$$C_1 : (I \in C_1) \textit{ iff } \#_{1st}(I) - \#_{2nd}(I) > f$$

$$P_1(J) \equiv \#_{1st}(J) - \#_{2nd}(J)) \; > f - \#_{\perp}(J)$$

$$S_1(J) = a : \#_a(J) = \#_{1st}(J)$$

$$C_2 : (I \in C_2) \textit{ iff } \#_{\max(I)}(I) > f$$

$$P_2(J) \equiv \#_{\max(J)}(J) > f - \#_{\perp}(J)$$

$$S_2(J) = \max(J)$$

The Triumph of Randomization

The Big Picture

- Does randomization make for more powerful algorithms?
  - Does randomization expand the class of problems solvable in polynomial time?
  - Does randomization help compute problems fast in parallel in the PRAM model?

The Big Picture

- Does randomization make for more powerful algorithms?
  - Does randomization expand the class of problems solvable in polynomial time?
  - Does randomization help compute problems fast in parallel in the PRAM model?

You tell me!

The Triumph of Randomization?

Well, at least for distributed computations!

- no deterministic 1-crash-resilient solution to Consensus
- f-resilient randomized solution to consensus ($f < n/2$) for crash failures
- randomized solution for Consensus exists even for Byzantine failures!

A simple randomized algorithm

M. Ben Or. "Another advantage of free choice: completely asynchronous agreement protocols" (PODC 1983, pp. 27-30)

- exponential number of operations per process
- BUT more practical protocols exist
  - down to $O(n\, log^2 n)$ expected operations/process
  - $n-1$ resilient

The protocol's structure

An infinite repetition of asynchronous rounds

- in round r, p only handles messages with timestamp r
- each round has two phases
- in the first, each p broadcasts an a-value which is a function of the b-values collected in the previous round (the first a-value is the input bit)
- in the second, each p broadcasts a b-value which is a function of the collected a-values
- decide stutters

Ben Or's Algorithm

a_p := input bit; r := 1;
repeat forever
{phase 1}
send (a_p, r) to all
Let A be the multiset of the first $n-f$ a-values with timestamp r received
if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
else b_p := $\perp$
{phase 2}
send (b_p, r) to all
Let B be the multiset of the first $n-f$ b-values with timestamp r received
if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p:= v
else if $(\exists b \in B : b \neq \perp)$ then a_p:= b
else a_p := $\$$ { $\$$ is chosen uniformly at random to be 0 or 1}
r := $r+1$

Validity

a_p := input bit; r:= 1;
repeat forever
{phase 1}
send (a_p, r) to all
Let A be the multiset of the first $n-f$ a-values with timestamp r received
if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
else b_p := $\perp$
{phase 2}
send (b_p, r) to all
Let B be the multiset of the first $n-f$ b-values with timestamp r received
if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p:= v
else if $(\exists b \in B : b \neq \perp)$ then a_p:= b
else a_p := $\$$ {$\$$ is chosen uniformly at random to be 0 or 1}
r := $r+1$

Validity

```
a_p := input bit;  r := 1;
repeat forever
  {phase 1}
  send (a_p, r) to all
  Let A be the multiset of the first n−f a-values with
      timestamp r received
  if (∃v ∈ {0,1} : ∀a ∈ A : a = v) then b_p := v
  else b_p := ⊥
  {phase 2}
  send (b_p, r) to all
  Let B be the multiset of the first n−f b-values with
      timestamp r received
  if (∃v ∈ {0,1} : ∀b ∈ B : b = v) then decide(v);  a_p := v
  else if (∃b ∈ B : b ≠ ⊥) then a_p := b
  else a_p := $   {$ is chosen uniformly at random
                   to be 0 or 1}
  r := r+1
```

- All identical inputs (i)
- Each process set a-value := i and broadcasts it to all
- Since at most f faulty, every correct process receives at least $n-f$ identical a-values in round 1
- Every correct process sets b-value := i and broadcasts it to all
- Again, every correct process receives at least $n-f$ identical i b-values in round 1 and decides

A useful observation

```
a_p := input bit; r := 1;
repeat forever
  {phase 1}
  send (a_p, r) to all
  Let A be the multiset of the first n−f a-values with
      timestamp r received
  if (∃v ∈ {0,1} : ∀a ∈ A : a = v) then b_p := v
  else b_p := ⊥
  {phase 2}
  send (b_p, r) to all
  Let B be the multiset of the first n−f b-values with
      timestamp r received
  if (∃v ∈ {0,1} : ∀b ∈ B : b = v) then decide(v);  a_p := v
  else if (∃b ∈ B : b ≠ ⊥) then a_p := b
  else a_p := $   {$ is chosen uniformly at random
                    to be 0 or 1}
  r := r+1
```

Lemma For all r, either

$b_{p,r} \in \{1, \perp\}$ for all p or

$b_{p,r} \in \{0, \perp\}$ for all p

A useful observation

```
1:  a_p := input bit;  r := 1;
2:  repeat forever
3:  {phase 1}
4:  send (a_p, r) to all
5   Let A be the multiset of the first n−f a-values with
        timestamp r received
6:  if (∃v ∈ {0,1} : ∀a ∈ A : a = v) then b_p := v
7:  else b_p := ⊥
8:  {phase 2}
9:  send (b_p, r) to all
10: Let B be the multiset of the first n−f b-values with
        timestamp r received
11: if (∃v ∈ {0,1} : ∀b ∈ B : b = v) then decide(v);  a_p := v
12: else if (∃b ∈ B : b ≠ ⊥) then  a_p := b
13: else  a_p := $   {$ is chosen uniformly at random
                        to be 0 or 1}
14: r := r+1
```

Lemma For all r, either

$b_{p,r} \in \{1, \perp\}$ for all p or

$b_{p,r} \in \{0, \perp\}$ for all p

Proof By contradiction.

Suppose p and q at round r such that

$b_{p,r} = 0$ and $b_{q,r} = 1$

From lines 6,7 p received $n-f$ distinct 0s, q received $n-f$ distinct 1s.

Then, $2(n-f) \leq n$, implying $n \leq 2f$

Contradiction

Corollary It is impossible that two processes p and q decide on different values at round r

Agreement

```
a_p := input bit; r := 1;
repeat forever
{phase 1}
send (a_p, r) to all
Let A be the multiset of the first n−f a-values with
     timestamp r received
if (∃v ∈ {0,1} : ∀a ∈ A : a = v) then b_p := v
else b_p := ⊥
{phase 2}
send (b_p, r) to all
Let B be the multiset of the first n−f b-values with
     timestamp r received
if (∃v ∈ {0,1} : ∀b ∈ B : b = v) then decide(v);  a_p := v
else if (∃b ∈ B : b ≠ ⊥) then a_p := b
else a_p := $   {$ is chosen uniformly at random
                 to be 0 or 1}
r := r+1
```

- Let r be the first round in which a decision is made
- Let p be a process that decides in r

Agreement

1: a_p := input bit; r:= 1;
2: repeat forever
3: {phase 1}
4: send (a_p, r) to all
5 Let A be the multiset of the first $n-f$ a-values with timestamp r received
6: if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
7: else b_p := $\perp$
8: {phase 2}
9: send (b_p, r) to all
10: Let B be the multiset of the first $n-f$ b-values with timestamp r received
11: if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p:= v
12: else if $(\exists b \in B : b \neq \perp)$ then a_p:= b
13: else a_p := $\$$ {$\$$ is chosen uniformly at random to be 0 or 1}
14: r := $r+1$

- Let r be the first round in which a decision is made
- Let p be a process that decides in r
- By the Corollary, no other process can decide on a different value in r
- To decide, p must have received $n-f$ "i" from distinct processes
- every other correct process has received "i" from at least $n-2f \geq 1$
- By lines 11 and 12, every correct process sets its new a-value to for round $r+1$ to "i"
- By the same argument used to prove Validity, every correct process that has not decided "i" in round r will do so by the end of round $r+1$

Termination I

a_p := input bit; r:= 1;
repeat forever
{phase 1}
send (a_p, r) to all
Let A be the multiset of the first $n-f$ a-values with timestamp r received
if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
else b_p := $\perp$
{phase 2}
send (b_p, r) to all
Let B be the multiset of the first $n-f$ b-values with timestamp r received
if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p:= v
else if $(\exists b \in B : b \neq \perp)$ then a_p:= b
else a_p := $\$$ {$\$$ is chosen uniformly at random to be 0 or 1}
r := $r+1$

- Remember that by Validity, if all (correct) processes propose the same value "i" in phase 1 of round r, then every correct process decides "i" in round r.
- The probability of all processes proposing the same input value (a landslide) in round 1 is

  Pr[landslide in round 1] = $1/2^n$
- What can we say about the following rounds?

Termination II

1: a_p := input bit; r:= 1;
2: repeat forever
3: {phase 1}
4: send (a_p, r) to all
5 Let A be the multiset of the first $n-f$ a-values with timestamp r received
6: if $(\exists v \in \{0,1\} : \forall a \in A : a = v)$ then b_p := v
7: else b_p := $\perp$
8: {phase 2}
9: send (b_p, r) to all
10: Let B be the multiset of the first $n-f$ b-values with timestamp r received
11: if $(\exists v \in \{0,1\} : \forall b \in B : b = v)$ then decide(v); a_p := v
12: else if $(\exists b \in B : b \neq \perp)$ then a_p:= b
13: else a_p := $\$$ {$\$$ is chosen uniformly at random to be 0 or 1}
14: r := $r+1$

- In round r > 1, the a-values are not necessarily chosen at random!
- By line 12, some process may set its a-value to a non-random value v
- By the Lemma, however, all non-random values are identical!
- Therefore, in every r there is a positive probability (at least $1/2^n$) for a landslide
- Hence, for any round r

  Pr[no lanslide at round r] $\leq 1 - 1/2^n$
- Since coin flips are independent:

  Pr[no lanslide for first k rounds] $\leq (1 - 1/2^n)^k$
- When $k = 2^n$, this value is about 1/e; then, if $k = c2^n$

  Pr[landslide within k rounds] ≥

  $$1 - (1 - 1/2^n)^k \approx 1 - 1/e^c$$

  which converges quickly to 1 as c grows