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
- We’ll take care of Berlusconi later

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

Supplies

- Each legislator receives
  - Ledger
  - Pen with indelible ink
  - Scratch paper
  - Hourglass
  - Lots of messengers
The Players

- Proposers
- Acceptors
- Learners

Choosing a value

Use a single acceptor

What if the acceptor fails?

6 is chosen!

Accepting a value

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

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 \((psn, \text{value})\)
- 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

“Any customer can have a car painted any color, as he wants so long as it is black”

Henry Ford
Choosing a unique value

“Any customer can have a car painted any color, as he wants so long as it is black”
Henry Ford

“Any acceptor can accept as many proposals as he wants, so long as they all propose the same value”
Leslie Lamport

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

It’s up to the Acceptors!

Recall P2a:

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

We strengthen it to:

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

What about P1?

How does $a_1$ 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!
It's up to the Proposers!

Recall P2a:

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 \( v \)

What to propose

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

Suppose \( p \) wants to issue a proposal numbered \( n \).

- If \( p \) can be certain that no proposal numbered \( n' < n \) has been chosen then \( p \) can propose any value!

- If a proposal numbered \( n' < n \) has been chosen, then it has been accepted by a majority set \( S \)
  - Any majority set \( S' \) must intersect \( S \)
  - If \( p \) can find one \( S' \) in which no acceptors has accepted a proposal numbered \( n' < n \), then no such proposal can have yet been chosen!

But how?

What to propose

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

Suppose \( p \) wants to issue a proposal numbered \( n \).

- If \( p \) can be certain that no proposal numbered \( n' < n \) has been chosen then \( p \) can propose any value!

- If a proposal numbered \( n' < n \) has been chosen, then it has been accepted by a majority set \( S \)
  - Any majority set \( S' \) must intersect \( S \)
  - If \( p \) can find \( S' \) s.t no acceptor has accepted (or will accept) a proposal numbered \( n' < n \), then no such proposal can have yet been chosen!
What to propose

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

Suppose \( p \) wants to issue a proposal numbered \( n \).

- If \( p \) can be certain that no proposal numbered \( n' < n \) has been chosen then \( p \) can propose any value!
- If not, how could we prove P2b?

It's up to an invariant!

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:

- \( \forall \) no acceptor in \( S \) has accepted any proposal numbered less than \( n \), or
- \( \forall \) \( v \) 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 $n$.

$S = (4,8)$
$S = (1,5)$
$S = (5,2)$

$P2c$ in action

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

$S = (4,8)$
$S = (18,2)$
$S = (3,2)$
$S = (5,2)$

Future telling?

$p$ 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.

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

But should be an invariant!

The invariant is violated.
Future telling?

- $p$ 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. 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

- 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 \(<\text{prepare},n>\) to a
    majority of acceptors
  
  - If an acceptor \( a \) receives \(<\text{prepare},n'>\), where
    \( n' > n \) of
    any \(<\text{prepare},n>\) to which it has responded, then it
    responds to \(<\text{prepare},n'>\) with
      - a promise not to accept any more proposals
      - numbered less than \( n' \)
      - the highest numbered proposal (if any) that it has
        accepted

- **Phase 2**

  - If the proposer receives a response to \(<\text{prepare},n>\)
    from a majority of acceptors, then it sends to each
    \(<\text{accept},n,v>\), 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 \(<\text{accept},n,v>\), it accepts the
    proposal unless it has in the meantime responded to
    \(<\text{prepare},n'>\), 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.

Was 6 chosen?

Propose something!

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.

Liveness

Progress is not guaranteed:

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

$P_1$

$<\text{propose},n_1>$

$<\text{accept}(n_1,v_1)>$

$<\text{propose},n_3>$

$P_2$

$<\text{propose},n_2>$

$<\text{accept}(n_2,v_2)>$

$<\text{propose},n_4>$

Time

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 $\lambda$ is elected...

Since $\lambda$ 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.
- $\lambda$ 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.
- $\lambda$ 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.
- $\lambda$ 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

$\lambda$ can efficiently execute phase 1 for infinitely many instances of consensus! (e.g. command 16 and higher)
- $\lambda$ 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
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?

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
Weakening termination

- Consensus requires correct processes to decide always
- Ben Or's algorithm only requires termination with probability 1
- They are not the same thing!

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

Validity

Ben Or's Algorithm

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 $(\exists v \in \{0, 1\} : \exists a \in A : a = v)$ then $b_p := v$
7: else $b_p := \bot$
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 $(\exists b \in B : b \neq \bot)$ then $a_p := b$
13: else $a_p := S$ \{ $S$ is chosen uniformly at random to be 0 or 1 \}
14: $r := r + 1$

Validity

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 $(\exists v \in \{0, 1\} : \exists a \in A : a = v)$ then $b_p := v$
7: else $b_p := \bot$
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 $(\exists a \in A : a \neq \bot)$ then $a_p := a$
13: else $a_p := S$ \{ $S$ is chosen uniformly at random to be 0 or 1 \}
14: $r := r + 1$
Validity

1. $a_i$ := input bit; $i = 1$;
2. repeat forever
3. (phase 1)
4. send ($a_r$) to all
5. Let A be the multiset of the first $n$ a-values with timestamp r received
6. if ($a_i$ ∈ {0, 1}) then $a_i$ := 1
7. else $a_i$ := 0
8. (phase 2)
9. send ($a_r$) to all
10. Let B be the multiset of the first $n$ b-values with timestamp r received
11. if ($b_i$ ∈ {0, 1}) then $b_i$ := 1
12. else if ($b_i$ ∈ {0, 1}) then $b_i$ := 0
13. else $b_i$ := 1
14. $r := r + 1$

A useful observation

1. $a_i$ := input bit; $i = 1$;
2. repeat forever
3. (phase 1)
4. send ($a_r$) to all
5. Let A be the multiset of the first $n$ a-values with timestamp r received
6. if ($a_i$ ∈ {0, 1}) then $a_i$ := 1
7. else $a_i$ := 0
8. (phase 2)
9. send ($a_r$) to all
10. Let B be the multiset of the first $n$ b-values with timestamp r received
11. if ($b_i$ ∈ {0, 1}) then $b_i$ := 1
12. else if ($b_i$ ∈ {0, 1}) then $b_i$ := 0
13. else $b_i$ := 0
14. $r := r + 1$

A useful observation

1. $a_i$ := input bit; $i = 1$;
2. repeat forever
3. (phase 1)
4. send ($a_r$) to all
5. Let A be the multiset of the first $n$ a-values with timestamp r received
6. if ($a_i$ ∈ {0, 1}) then $a_i$ := 1
7. else $a_i$ := 0
8. (phase 2)
9. send ($a_r$) to all
10. Let B be the multiset of the first $n$ b-values with timestamp r received
11. if ($b_i$ ∈ {0, 1}) then $b_i$ := 1
12. else if ($b_i$ ∈ {0, 1}) then $b_i$ := 0
13. else $b_i$ := 0
14. $r := r + 1$

Lemma For all $r$, either $b_{p,r} \in \{0, 1\}$ for all $p$ or $b_{p,r} \in \{0, 1\}$ 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 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 $r$

A useful observation

1. $a_i$ := input bit; $i = 1$;
2. repeat forever
3. (phase 1)
4. send ($a_r$) to all
5. Let A be the multiset of the first $n$ a-values with timestamp r received
6. if ($a_i$ ∈ {0, 1}) then $a_i$ := 1
7. else $a_i$ := 0
8. (phase 2)
9. send ($a_r$) to all
10. Let A be the multiset of the first $n$ a-values with timestamp r received
11. if ($a_i$ ∈ {0, 1}) then $a_i$ := 1
12. else if ($a_i$ ∈ {0, 1}) then $a_i$ := 0
13. else $a_i$ := 0
14. $r := r + 1$

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

1. $a_i$: input bit; $i = 1$
2. repeat forever
3. {phase 1}
4. send ($a_i$, $p_i$) to all
5. Let $A$ be the multiset of the first $n - f$-values with timestamp $r$ received
6. if $(b_i \in (0, 1) \land e \in A \land v = e)$ then $b_i = v$
7. else $b_i = 1$
8. {phase 2}
9. send ($b_i$, $r$) to all
10. Let $B$ be the multiset of the first $n - f$-values with timestamp $r$ received
11. if $(b_i \in (0, 1) \land e \in B \land v = e)$ then decide($b_i$) $a_i = v$
12. else if $(b_i \in (0, 1) \land v $ then $a_i = b$
13. else $a_i = 8$ (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 $n - f$ "1" from distinct processes
every other correct process has received "1" 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 decide, $p$ must have received "1" in round $r$, will do so by the end of round $r + 1$

**Termination I**

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

Remember that by Validity, if all (correct) processes propose the same value "1" in phase 1 of round $r$, then every correct process decides "1" in round $r$.
The probability of all processes proposing the same input value (a landslide) in round 1 is $Pr[\text{landslide in round 1}] = 1/2^n$.

What can we say about the following rounds?

**Termination II**

1. $a_i$: input bit; $i = 1$
2. repeat forever
3. {phase 1}
4. send ($a_i$, $p_i$) to all
5. Let $A$ be the multiset of the first $n - f$-values with timestamp $r$ received
6. if $(b_i \in (0, 1) \land e \in A \land v = e)$ then $b_i = v$
7. else $b_i = 1$
8. {phase 2}
9. send ($b_i$, $r$) to all
10. Let $B$ be the multiset of the first $n - f$-values with timestamp $r$ received
11. if $(b_i \in (0, 1) \land e \in B \land v = e)$ then decide($b_i$) $a_i = v$
12. else if $(b_i \in (0, 1) \land v $ then $a_i = b$
13. else $a_i = 8$ (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 round $r$ there is a positive probability (at least $1/2^2$) for a landslide.
Hence, for any round $r$

$Pr[\text{no landslide at round } r] \leq 1 - 1/2^n$

Since coin flips are independent:
$Pr[\text{no landslide for first } k \text{ rounds}] \leq (1 - 1/2^n)^k$

When $k = 2^n$, this value is about $1/e$; then, if $k = \sqrt{n}$

$Pr[\text{landslide within } k \text{ rounds}] \geq 1 - (1 - 1/2^n)^k \approx 1 - \frac{1}{\sqrt{n}}$

which converges quickly to 1 as $c$ grows.