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
- pen with indelible ink
- scratch paper
- lots of messengers
- 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

Have a single acceptor
Choosing a value

Have a single acceptor of a majority

Using a majority set guarantees that at most one value is chosen
Accepting a value

- Suppose only one proposer proposes a single value
- Assume no failures
- That value should be accepted!
Accepting a value

- Suppose only one proposer proposes a single value
- Assume no failures
- That value should be accepted!

P1: Acceptors must accept first received proposal
Accepting a value

P1: Acceptors must accept first received proposal

- Choosing a value requires a majority of acceptors to accept that value
- What if we have multiple proposers, each proposing a different value?
- Acceptors must accept multiple proposals (each identified by pair \((n, \text{value})\))
Guaranteeing uniqueness

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

How do we implement P2?

What about: If a proposal with value $\nu$ is chosen, then every higher-numbered proposal accepted by any acceptor has value $\nu$.

	It satisfies P1 and P2, but it not implementable in an asynchronous system!
Another take on P2

If a proposal with value $\nu$ is chosen, then every higher-numbered proposal accepted by any acceptor has value $\nu$
Another take on P2

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

If a proposal with value $\nu$ is chosen, then every higher-numbered proposal issued by any proposer has value $\nu$
Implementing P2

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

How would we enforce this? Use as inspiration a possible proof!

- Assume some $(m, v)$ has been chosen by a set $C$ of acceptors
- Assume, by induction, that all proposal issued with numbers in the range $m..n-1$ proposed $v$
- Then, any acceptor that accepts a proposal with number $m..n-1$ has value $v$
- The proposal with number $n$ has value $v$ if the following invariant holds:
- Let $S$ be a majority set. of acceptors When a proposer issues a value $v$
Implementing P2

If a proposal with value $\nu$ is chosen, then every higher-numbered proposal issued by any proposer has value $\nu$

Achieved by enforcing the following invariant

For any $\nu$ and $n$, if a proposal with value $\nu$ and pid $n$ is issued, then there is a majority-set $S$ of acceptors such that one of the following holds:

- no acceptor in $S$ has accepted any proposal numbered less than $n$
- $\nu$ is the value of the highest-numbered proposal among all proposal numbered less than $n$ accepted by the acceptors in $S$
The proposer’s protocol

1. A proposer chooses a new \( n \) and sends \(<\text{prepare},n>\) 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.

2. If proposer receives a response from a majority of acceptors, then it can issue \(<\text{accept}(n,v)>\) where \( v \) is the value of the highest numbered proposal among the responses, or is any value selected by the proposer if responders returned no proposals.
The acceptor’s protocol

1. Can ignore any request without violating safety

2. Can always respond to *prepare* messages

3. Can respond to \(<\text{accept}(n,v)\)> iff it has not promised not to—i.e. it has not responded to \(<\text{prepare},n'\) with \(n'\) > \(n\)

**Acceptor must remember**
- highest numbered proposal ever accepted
- highest numbered prepare request to which it responded
Learning chosen values

Once a value is chosen, it is forwarded to the learners. Many strategies are possible:

i. Each acceptor informs each learner

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

iii. Something in between
Liveness

Progress is not guaranteed:

\[ n_1 < n_2 < n_3 < n_4 < ... \]

```plaintext
P_1

<propose, n_1>

<accept(n_1, v_1)>

<propose, n_3>

P_2

<propose, n_2>

<accept(n_2, v_2)>

<propose, n_4>
```

Time
All proposers are equal, but some more so than others

- Elect a **distinguished proposer**
- Can’t be done reliably in asynchronous systems, so...
  - real time
  - randomization
Arbitrary failures with message authentication

- Process can send conflicting messages to different receivers
- Messages are signed with unforgeable signatures

Arbitrary (Byzantine) failures
Valid messages

A **valid** message \( m \) has the following form:

- **in round 1:**
  
  \[ < m : s > \text{ (} m \text{ is signed by the sender) } \]

- **in round \( r > 1 \), if received by \( p \) from \( q \):**

  \[ < m : p_1 : p_2 : \ldots : p_r > \text{ where } \]
  
  - \( p_1 = \text{ sender; } p_r = q \)
  - \( p_1, \ldots, p_r \) are distinct from each other and from \( p \)
  - message has not been tampered with
AFMA: The Idea

- A correct process $p$ discards all non-valid messages it receives.
- If a message is valid,
  - it “extracts” the value from the message
  - it relays the message, with its own signature appended
- At round $f + 1$:
  - if it extracted exactly one message, $p$ delivers it
  - otherwise, delivers SF
AFMA: The Protocol

sender $s$ in round 0:
1: $\text{extract } m$

sender in round 1:
2: $\text{send } < m:s > \text{ to all}$

Process $p$ in round $k$, $1 \leq k \leq f+1$
3: if $p$ extracted $m$ from a valid message $< m:p_1: \ldots :p_{k-1}>$ in round $k-1$ and $p \neq \text{sender}$ then
4: $\text{send } < m:p_1: \ldots :p_{k-1}:p > \text{ to all}$
5: receive round $k$ messages from all processes
6: for each valid round $k$ message $< m:p_1: \ldots :p_{k-1}:p_k >$ received by $p$
7: if $p$ has not previously extracted $m$ then
8: $\text{extract } m$
9: if $k = f+1$ then
10: if in the entire execution $p$ has extracted exactly one $m$ then
11: $\text{deliver}(m)$
12: else $\text{deliver(SF)}$
13: $\text{halt}$
Termination

sender s in round 0:
1: extract m

sender in round 1:
2: send < m:s > to all

Process p in round k, 1 ≤ k ≤ f+1
3: if p extracted m from a valid message <m:p₁: ... :pk-1>
in round k - 1 and p ≠ sender then
4: send <m:p₁: ... :pk-1:p> to all
5: receive round k messages from all processes
6: for each valid round k message < m:p₁: ... :Pk-1:Pk>
   received by p
7: if p has not previously extracted m then
8: extract m
9: if k = f+1 then
10: if in the entire execution p has extracted exactly
    one m then
11: deliver(m)
12: else deliver(SF)
13: halt

In round $f+1$, every correct process delivers either $m$ or SF and then halts.
sender s in round 0:
1: extract m
sender in round 1:
2: send <m:s> to all
Process p in round k, 1 ≤ k ≤ f+1
3: if p extracted m from a valid message <m:p1: ... :pk-1> in round k - 1 and p ≠ sender then
   4: send <m:p1: ... :pk-1:p> to all
5: receive round k messages from all processes
6: for each valid round k message <m:p1: ... :pk-1:pk> received by p
7: if p has not previously extracted m then
   8: extract m
9: if k = f+1 then
10: if in the entire execution p has extracted exactly one m then
11: deliver(m)
12: else deliver(SF)
13: halt

Lemma If a correct process extracts m, then every correct process eventually extracts m

Proof
Let r be the earliest round in which some correct process extracts m. Let that process be p.
• if p is the sender, then in round 1 p sends a valid message to all. All correct processes extract message in round 1
• otherwise, p has received in round r a message
   <m:p1:p2: ... :Pr>
• Claim: p1, p2, ..., pr are all faulty
  - true for p1 = s
  - Suppose p_j, 1 ≤ j ≤ r, were correct
    • p_j signed and relayed message in round j
    • p_j extracted message in round j - 1
      CONTRADICTION
• If r ≤ f, p will send a valid message
   <m:p1:p2: ... :Pr:p> in round r + 1 ≤ f + 1 and every correct process will extract it in round r + 1 ≤ f + 1
• If r = f + 1, by Claim above, p_1, p_2, ..., p_{f+1} faulty
  - At most f faulty processes
  - CONTRADICTION
Validity

sender s in round 0:
1: extract m
sender in round 1:
2: send < m:s > to all
Process p in round k, 1 ≤ k ≤ f+1
3: if p extracted m from a valid message <m:p_1: ... :p_k-1>
in round k - 1 and p ≠ sender then
4: send <m:p_1: ... :p_k-1:p> to all
5: receive round k messages from all processes
6: for each valid round k message <m:p_1: ... :p_k-1:p_k>
   received by p
7: if p has not previously extracted m then
8: extract m
9: if k = f+1 then
10: if in the entire execution p has extracted exactly
    one m then
11: deliver(m)
12: else deliver(SF)
13: halt

From Agreement and the observation that the sender, if correct, delivers its own message.
TRB for arbitrary failures

Fail-stop → Crash
Send Omission → Receive Omission
General Omission
Arbitrary failures with message authentication
Arbitrary (Byzantine) failures

Srikanth, T.K., Toueg S.
Simulating Authenticated Broadcasts to Derive Simple Fault-Tolerant Algorithms
Distributed Computing 2 (2), 80-94
AF: The Idea

- Identify the essential properties of message authentication that made AFMA work
- Implement these properties without using message authentication
AF: The Approach

- Introduce two primitives
  - \texttt{broadcast}(p,m,i) \ (executed by p in round i)
  - \texttt{accept}(p,m,i) \ (executed by q in round j \geq i)
- Give axiomatic definitions of broadcast and accept
- Derive an algorithm that solves TRB for AF using these primitives
- Show an implementation of these primitives that does not use message authentication
Properties of broadcast and accept

- **Correctness** If a correct process $p$ executes broadcast($p$, $m$, $i$) in round $i$, then all correct processes will execute accept($p$, $m$, $i$) in round $i$.

- **Unforgeability** If a correct process $q$ executes accept($p$, $m$, $i$) in round $j \geq i$, and $p$ is correct, then $p$ did in fact execute broadcast($p$, $m$, $i$) in round $i$.

- **Relay** If a correct process $q$ executes accept($p$, $m$, $i$) in round $j \geq i$, then all correct processes will execute accept($p$, $m$, $i$) by round $j + 1$. 

**AF: The Protocol – 1**

sender s in round 0:
0: extract m

sender s in round 1:
1: broadcast (s,m,1)

Process p in round k, 1 ≤ k ≤ f + 1
2: if p extracted m in round k - 1 and p ≠ sender then
4: broadcast (p,m,k)
5: if p has executed at least k accept(qi,m,ji) 1 ≤ i ≤ k in rounds 1 through k
   (where (i) qi distinct from each other and from p, (ii) one qi is s, and
    (iii) 1 ≤ ji ≤ k ) and p has not previously extracted m then
6: extract m
7: if k = f+1 then
8: if in the entire execution p has extracted exactly one m then
9: deliver(m)
10: else deliver(SF)
11: halt
Termination

sender s in round 0:
0: extract m
sender s in round 1:
1: broadcast (s,m,1)

Process p in round k, 1 ≤ k ≤ f+1
2: if p extracted m in round k - 1 and p ≠ sender then
4: broadcast (p,m,k)
5: if p has executed at least k accept(qi,m,jj) 1 ≤ i ≤ k in rounds 1 through k
   (where (i) qi distinct from each other and from p, (ii) one qi is s, and (iii) 1 ≤ jj ≤ k)
   and p has not previously extracted m then
6: extract m
7: if k = f+1 then
8: if in the entire execution p has extracted exactly one m then
9: deliver(m)
10: else deliver(SF)
11: halt

In round f+1, every correct process delivers either m or SF and then halts
Agreement - 1

sender s in round 0:
0: extract m

sender s in round 1:
1: broadcast (s,m,1)

Process p in round k, 1 ≤ k ≤ f+1
2: if p extracted m in round k - 1 and p ≠ sender then
4: broadcast (p,m,k)
5: if p has executed at least k accept(qi,m,ji) 1 ≤ i ≤ k in rounds 1 through k
   (where (i) qi distinct from each other and from p, (ii) one qi is s, and (iii) 1 ≤ j ≤ k )
   and p has not previously extracted m then
6: extract m
7: if k = f+1 then
8: if in the entire execution p has extracted exactly one m then
9: deliver(m)
10: else deliver(SF)
11: halt

Lemma
If a correct process extracts m, then every correct process eventually extracts m

Proof
Let r be the earliest round in which some correct process extracts m. Let that process be p.

if r = 0, then p = s and p will execute broadcast(s,m,1) in round 1. By CORRECTNESS, all correct processes will execute accept (s,m,1) in round 1 and extract m

if r > 0, the sender is faulty. Since p has extracted m in round r, p has accepted at least r triples with properties (i), (ii), and (iii) by round r

r ≤ f By RELAY, all correct processes will have accepted those r triples by round r + 1
p will execute broadcast(p,m,r + 1) in round r + 1
By CORRECTNESS, any correct process other than p, q1, q2,...,qr will have accepted r + 1 triples (qk,m,jk), 1 ≤ jk ≤ r + 1, by round r + 1
q1, q2,...,qr,p are all distinct
every correct process other than q1, q2,...,qr,p will extract m
p has already extracted m; what about q1, q2,...,qr?
Agreement - 2

sender s in round 0:
0: extract m
sender s in round 1:
1: broadcast (s,m,1)

Process p in round k, 1 ≤ k ≤ f+1
2: if p extracted m in round k - 1 and p ≠ sender then
4: broadcast (p,m,k)
5: if p has executed at least k accept(qi,m,ji) 1 ≤ i ≤ k
in rounds 1 through k
   (where (i) qi distinct from each other and from
   p, (ii) one qi is s, and (iii) 1 ≤ ji ≤ k)
   and p has not previously extracted m then
6: extract m
7: if k = f+1 then
8: if in the entire execution p has extracted exactly
   one m then
9: deliver(m)
10: else deliver(SF)
11: halt

Claim: q₁, q₂, ..., qᵣ are all faulty
  > Suppose qₖ were correct
  > p has accepted (qₖ, m, jₖ) in round jₖ ≤ r
  > By UNFORCEABILITY, qₖ executed
    broadcast (qₖ, m, jₖ) in round jₖ
  > qₖ extracted m in round jₖ-1 < r
    CONTRADICTION

□ Case 2: r = f + 1
  □ Since there are at most f faulty processes, some process qᵢ
    in q₁, q₂, ..., qᵢ+1 is correct
  □ By UNFORCEABILITY, qᵢ executed
    broadcast (qᵢ, m, jᵢ) in round jᵢ ≤ r
  □ qᵢ has extracted m in round jᵢ-1 < f + 1
    CONTRADICTION
Validity

A correct sender executes broadcast\((s, m, 1)\) in round 1

By CORRECTNESS, all correct processes execute accept\((s, m, 1)\) in round 1 and extract \(m\)

In order to extract a different message \(m'\), a process must execute accept\((s, m', 1)\) in some round \(i \leq f + 1\)

By UNFORGEABILITY, and because \(s\) is correct, no correct process can extract \(m' \neq m\)

All correct processes will deliver \(m\)