Why Distributed Consensus is difficult?

- Arbitrary message delays (asynchronous network)
- Independent parties (nodes) can go offline (and also back online)
- Network partitions
- Message reorderings
- Malicious (Byzantine) parties
Why Distributed Consensus is difficult?

- Arbitrary message delays (asynchronous network)
- Independent parties (nodes) can go offline (and also back online)
- Network partitions
- Message reorderings
- Malicious (Byzantine) parties
Authors

- Leslie Lamport
  - you again!
  - we all know him
- Robert Shostak
  - PhD in Applied Math, Harvard
  - SRI International
  - Founder, Ansa Software
  - Founder, Mira Tech
  - Borland Software
  - Founder Portera System
  - Founder Vocera
- Marshall Pease
Another story from Lamport?

Time, Clocks, and the Ordering of Events in a Distributed System 1978

The part-time parliament 1990
Another story from Lamport?

*Time, Clocks, and the Ordering of Events in a Distributed System* 1978

*The Byzantine Generals Problem* 1982

*The part-time parliament* 1990
How this story came

“
I have long felt that, because it was posed as a cute problem about philosophers seated around a table, Dijkstra's dining philosopher's problem received much more attention than it deserves.

......

The popularity of the dining philosophers problem taught me that the best way to attract attention to a problem is to present it in terms of a story.

”

http://lamport.azurewebsites.net/pubs/pubs.html#byz
How this story came

“

There is a problem in distributed computing that is sometimes called the Chinese Generals Problem, in which two generals have to come to a common agreement on whether to attack or retreat, but can communicate only by sending messengers who might never arrive.

”

http://lamport.azurewebsites.net/pubs/pubs.html#byz
How this story came

“

I stole the idea of the generals and posed the problem in terms of a group of generals, some of whom may be traitors, who have to reach a common decision.

”

http://lamport.azurewebsites.net/pubs/pubs.html#byz
What is the Byzantine generals problem
Byzantine generals problem

“several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can communicate with one another only by messenger. After observing the enemy, they must decide upon a common plan of action.”

*castle: http://simpleicon.com/castle.html
*lieutenant: https://www.clipartmax.com/max/m282Z58b1H7N4h7/
Byzantine generals problem

- Generals should reach a consensus on the plan
- It could be ATTACK
Byzantine generals problem

- Generals should reach a consensus on the plan
- Or RETREAT

*castle: [http://simpleicon.com/castle.html](http://simpleicon.com/castle.html)
*lieutenant: [https://www.clipartmax.com/max/m28Z5i8b1H7N4h7/](https://www.clipartmax.com/max/m28Z5i8b1H7N4h7/)
Byzantine generals problem

- But there might be traitors
- All loyal generals should reach a consensus
Byzantine generals problem

- But traitors can act arbitrarily
- All loyal generals should reach a consensus

*castle: [http://simpleicon.com/castle.html](http://simpleicon.com/castle.html)
*lieutenant: [https://www.clipartmax.com/mag/mz28Z5i8b1H7N4h7/](https://www.clipartmax.com/mag/mz28Z5i8b1H7N4h7/)
*traitor: [https://thenounproject.com/term/traitor/](https://thenounproject.com/term/traitor/)
Byzantine generals problem

- But traitors can act arbitrarily
- All loyal generals should reach a consensus

*castle: [http://simpleicon.com/castle.html](http://simpleicon.com/castle.html)
*lieutenant: [https://www.clipartmax.com/mag/mz28Z5i8b1H7N4h7/](https://www.clipartmax.com/mag/mz28Z5i8b1H7N4h7/)
*traitor: [https://thenounproject.com/term/traitor/](https://thenounproject.com/term/traitor/)
Byzantine generals problem

- A simplified version

“A commanding general sends an order to his $n-1$ lieutenant generals such that

IC1. All **loyal** lieutenants obey the same order.

IC2. If the commanding general is **loyal**, then every loyal lieutenant obeys the order he sends.”
What is the byzantine generals problem

- IC1. All loyal lieutenants obey the same order
- IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.

(Lamport calls it *Interactive Consistency*)
What is the byzantine generals problem

- Consistency/Agreement
- IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.
What is the byzantine generals problem

- Consistency/Agreement
- Validity
What is the byzantine generals problem

- Consistency/Agreement
- IC2. If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.
What is the byzantine generals problem

- Consistency/Agreement
- Validity
- Liveness/Termination?
Impossibility Result
Impossibility result

“if the generals can send only oral messages, then no solution will work unless more than $\frac{2}{3}$ of the generals are loyal.”
Impossibility result

“if the generals can send only *oral messages*, then no solution will work unless more than \( \frac{2}{3} \) of the generals are loyal.”

what are oral messages?
Impossibility result

oral messages:

● every message that is sent is delivered correctly
● the receiver of a message knows who sent it
● the absence of a message can be detected
Impossibility result

oral messages:

- every message that is sent is delivered correctly
- the receiver of a message knows who sent it
- the absence of a message can be detected
Impossibility result

oral messages:

- every message that is sent is delivered correctly
- authenticated channel
- the absence of a message can be detected
Oral messages:

- every message that is sent is delivered correctly
- authenticated channel
- the absence of a message can be detected
Impossibility result

oral messages:

● every message that is sent is delivered correctly
● authenticated channel
● synchronous network
Impossibility result

“if the generals can send only **oral messages**, then no solution will work unless more than $\frac{2}{3}$ of the generals are loyal.”

**in a synchronous** network, with **authenticated channel**, when $m$ generals are traitors, no solution will work unless there are more than $3m$ generals.
impossibility result - proof

- case $m = 1$:
impossibility result - proof

- case $m = 1$:
  - scenario 1:
    - the commander is loyal
    - one lieutenant is a traitor
impossibility result - proof

- **case** $m = 1$:
  - scenario 1:
    - the commander is loyal
    - one lieutenant is a traitor
    - the left lieutenant should **ATTACK**

the commander said “**RETREAT!**”
impossibility result - proof

- case $m = 1$:
  - scenario 2:
    - the commander is a traitor

  - the commander said “RETREAT!”
  - the commander said “ATTACK!”
Three scenarios

I should ATTACK!

I should RETREAT!

the commander said “RETREAT!”

the commander said “ATTACK!”

the commander said “RETREAT!”

the commander said “ATTACK!”
Three scenarios

Consistency broken!

Consistency: All loyal lieutenants obey the same order
impossibility result

prove $m > 1$ by contradiction

- assume we have a solution protocol $f$ for $3m$ generals when $m > 1$
- we can solve $m = 1$ case by leveraging $f$
prove \( m > 1 \) by contradiction

- assume the three generals are \( x, y, z \), and \( x \) is the commander;
- according to protocol \( f \)
  - \( x \) simulates one commander and \( m-1 \) lieutenants
  - each of \( y \) and \( z \) simulates \( m \) lieutenants
impossibility result

prove $m > 1$ by contradiction

- assume the three generals are $x$, $y$, $z$, and $x$ is the commander;
- according to protocol $f$
  - $x$ simulates one commander and $m-1$ lieutenants
  - each of $y$ and $z$ simulates $m$ lieutenants
- at most one of $x$, $y$, $z$ is a traitor
  - at most $m$ simulated traitors
  - protocol $f$ can solve the case when there are at most $m$ traitors
impossibility result

prove \( m > 1 \) by contradiction

- if we can solve case \( m > 1 \) then we can solve \( m = 1 \)
- we proved case \( m = 1 \) cannot be solved
- contradiction!
Oral messages’ fault

oral messages:

- every message that is sent is delivered correctly
- the receiver of a message knows who sent it
- the absence of a message can be detected

- With only oral messages, traitors can lie by telling the wrong command they received
Three scenarios

The commander said “RETREAT!”

ATTACK!

I should ATTACK!

The commander said “ATTACK!”

ATTACK!

ATTACK!

I should ATTACK!

The commander said “RETREAT!”

ATTACK!

The commander said “ATTACK!”

RETREAT!

I should RETREAT!
Signed message

- With only oral messages, traitors can lie by telling the wrong command they received

- Signed messages
  - cannot be forged
  - anyone can verify the authenticity
Solutions:
oral messages and signed messages
Solutions - with oral messages (k - number of traiters)

- OM(k)
  - k == 0
    - commander sends the value to every one
    - everyone return the value they received
Solutions - with oral messages

- **OM(k)**
  - **k == 0**
    - Commander sends the value to every one
    - Everyone return the value they received
  - **k > 0**
    - Commander sends the value to every one
    - Everyone start a smaller bgp **OM(k-1)** containing all ones but the current commander and become the new commander
    - Everyone participated \( n-1 \) **OM(k-1)** and get \( n-1 \) values, return the majority
OM(1)
$\text{OM}(1) - 3 \times \text{OM}(0)$
Solutions - with oral messages

- **OM(k)**
  - **k == 0**
    - commander sends the value to every one
    - everyone return the value they received
  - **k > 0**
    - commander sends the value to every one
    - everyone start a smaller bgp OM(k-1) containing all ones but the current commander and become the new commander
    - everyone participated n-1 OM(k-1) and get n-1 values, return the majority

- Intuition: for every message M received, solve a smaller bgp containing all but the current commander to tell others you received M
Solutions - with oral messages

- OM(k)
  - k == 0
    - commander sends the value to everyone
    - everyone return the value they received
  - k > 0
    - commander sends the value to everyone
    - everyone start a smaller bgp OM(k-1) containing all ones but the current commander and become the new commander
    - everyone participated n-1 OM(k-1) and get n-1 values, return the majority

- Intuition: for every message M received, solve a smaller bgp containing all but the current commander to tell others you received M

- OM(m) for m traitors when 3m < n
Solutions - with oral messages

● OM(k) - Message complexity: \( (n-1) \times MC(OM(k-1)) + n-1 = O(n^m) \)
  ○ \( k = 0 \)
    ■ commander sends the value to every one
    ■ everyone return the value they received
  ○ \( k > 0 \)
    ■ commander sends the value to every one
    ■ everyone start a smaller bgp OM(k-1) containing all ones but the current commander
    and become the new commander
    ■ everyone participated \( n-1 \) OM(k-1) and get \( n-1 \) values, return the majority

● Intuition: for every message \( M \) received, solve a smaller bgp containing all but
the current commander to tell others you received \( M \)

● OM(m) for m traitors when \( 3m < n \) (a Theorem, see Lamport’s paper)
Solutions - with signed messages

- **SM(m)**
  - every lieutenant maintains a value set $V(i)$
  - the commander (0) sends the value to every lieutenant with its signature
Solutions - with signed messages

- **SM(m)**
  - every lieutenant maintains a value set $V(i)$
  - the commander (0) sends the value to every lieutenant with its signature
  - for every lieutenant $i$
    - If $i$ receives a message $v:0$ from the commander
      - he lets $V(i)$ to be $\{v\}$
      - he sends the message $v:0:i$ to every other lieutenant
    - If $i$ receives a message $v:0:j_1:...:j_k$ and $v$ is not in $V(i)$, then
      - Add $v$ to $V(i)$
      - if $k < m$ then he sends the message $v:0:j_1:...:j_k:i$ to all lieutenants other than $j_1:...:j_k$
Solutions - with signed messages

- SM(m)
  - every lieutenant maintains a value set $V(i)$
  - the commander (0) sends the value to every lieutenant with its signature
  - for every lieutenant $i$
    - If $i$ receives a message $v:0$ from the commander
      - he lets $V(i)$ to be $\{v\}$
      - he sends the message $v:0:i$ to every other lieutenant
    - If $i$ receives a message $v:0:j_1:\ldots:j_k$ and $v$ is not in $V(i)$, then
      - Add $v$ to $V(i)$
      - if $k < m$ then he sends the message $v:0:j_1:\ldots:j_k:i$ to all lieutenants other than $j_1:\ldots:j_k$
  - when there will be no more messages, return $\text{choice}(V(i))$
  - choice($V$)
    - if $V = \{v\}$ return $v$
    - return RETREAT when $|V| = 0$
SM(1)

ATTACK! : 0

RETREAT! : 0

1

2
SM(1)

ATTACK!:0

RETREAT!:0

ATTACK!:0:1

RETREAT!:0:2
SM(1)

V(1) = V(2)
SM(1)

\[
\text{Choice}(V(1)) = \text{Choice}(V(2))
\]
Solutions - with signed messages

- SM(m) - message complexity: $O(n^2)$
  - every lieutenant maintains a value set $V(i)$
  - the commander (0) sends the value to every lieutenant with its signature
  - for every lieutenant $i$
    - If $i$ receives a message $v:0$ from the commander
      - he lets $V(i)$ to be $\{v\}$
      - he sends the message $v:0:i$ to every other lieutenant
    - If $i$ receives a message $v:0:j_1:...:j_k$ and $v$ is not in $V(i)$, then
      - Add $v$ to $V(i)$
      - if $k < m$ then he sends the message $v:0:j_1:...:j_k:i$ to all lieutenants other than $j_1:...:j_k$
    - when there will be no more messages, return $\text{choice}(V(i))$
  - $\text{choice}(V)$
    - if $V = \{v\}$ return $v$
    - return RETREAT when $|V| = 0$

- Intuition: ensure every message received by a loyal lieutenant is sent to every loyal lieutenant
- The protocol is safe as it is now stuck
Solutions - with signed messages

● SM(m) - message complexity: O(n^2)
  ○ every lieutenant maintains a value set V(i)
  ○ the commander (0) sends the value to every lieutenant with its signature
  ○ for every lieutenant i
    ○ If i receives a message v:0 from the commander
      ○ he lets V(i) to be {v}
      ○ he sends the message v:0:i to every other lieutenant
    ○ If i receives a message v:0:j_1:...:j_k and v is not in V(i), then
      ○ Add v to V(i)
      ○ if k < m then he sends the message v:0:j_1:...:j_k:i to all lieutenants other than j_1:...:j_k
    ○ when there will be no more messages, return choice(V(i))
  ○ choice(V)
    ■ if V = {v} return v
    ■ return RETREAT when |V| = 0

● Intuition: ensure every message received by a loyal lieutenant is sent to every loyal lieutenant
● The protocol is safe as it is now stuck
Minimum number required for which an $f$-resilient consensus protocol exists

<table>
<thead>
<tr>
<th></th>
<th>synchrony</th>
<th>asynchrony</th>
<th>partial synchrony</th>
</tr>
</thead>
<tbody>
<tr>
<td>fail-stop</td>
<td>$f+1$</td>
<td>inf</td>
<td>$2f+1$</td>
</tr>
<tr>
<td>crash</td>
<td>$f+1$</td>
<td>inf</td>
<td>$2f+1$ (Paxos)</td>
</tr>
<tr>
<td>byzantine with digital signature</td>
<td>$f+1$ (SM($f+1$))</td>
<td>inf</td>
<td></td>
</tr>
<tr>
<td>byzantine with authenticated channel</td>
<td>$3f+1$ (OM($f$))</td>
<td>inf</td>
<td></td>
</tr>
</tbody>
</table>

**Partial synchrony:**
fixed bounds on processor speed and message delays exist but they aren’t known a priori.
Minimum number required for which an $f$-resilient consensus protocol exists

<table>
<thead>
<tr>
<th></th>
<th>synchrony</th>
<th>asynchrony</th>
<th>partial synchrony</th>
</tr>
</thead>
<tbody>
<tr>
<td>fail-stop</td>
<td>$f+1$</td>
<td>$\infty$</td>
<td>$2f+1$</td>
</tr>
<tr>
<td>crash</td>
<td>$f+1$</td>
<td>$\infty$</td>
<td>$2f+1$ (Paxos)</td>
</tr>
<tr>
<td>byzantine with digital</td>
<td>$f+1$ (SM($f+1$))</td>
<td>$\infty$</td>
<td>???</td>
</tr>
<tr>
<td>signature</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>byzantine with</td>
<td>$3f+1$ (OM($f$))</td>
<td>$\infty$</td>
<td></td>
</tr>
<tr>
<td>authenticated channel</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Partial synchrony:**
fixed bounds on processor speed and message delays exist but they aren't known a priori.
Byzantine with digital signature in partial synchrony

- No partial synchronous protocols can tolerate \( \frac{1}{3} \) faults.
- Sound familiar?
- But there is a protocol that achieves safety for \((3f + 1)\)
Practical Byzantine Fault Tolerance (PBFT)

• Introduced by Miguel Castro & Barbara Liskov in 1999
  • almost 10 years after Paxos

• Addresses real-life constraints on Byzantine systems:
  • Partially-synchronous network
  • Byzantine failure
  • Message senders cannot be forged (via public-key crypto)
PBFT Terminology and Layout

- **Replicas** — nodes participating in a consensus (no more acceptor/proposer dichotomy)

- A *dedicated replica* *(primary)* acts as a commander
  - A primary can be re-elected if suspected to be compromised
  - **Backups** — other, non-primary replicas (lieutenants)

- **Clients** — communicate directly with primary/replicas

- The protocol uses *time-outs* (partial synchrony) to detect faults
  - *E.g.*, a primary not responding for too long is considered compromised
Practical Byzantine Fault Tolerance

- Commander sends the value to every lieutenant
- Every lieutenant
  - if it receives a new value $v$, broadcast (prepare, $v$)
  - if it receives $2f+1$ (prepare, $v$), broadcast (commit, $v$)
  - if it receives $2f+1$ (commit, $v$), broadcast (committed, $v$)
  - if it receives $f+1$ (committed, $v$), broadcast (committed, $v$)
Practical Byzantine Fault Tolerance

- Commander sends the value to every lieutenant
- Every lieutenant
  - if it receives a new value v, broadcast (pre-prepare, v)
  - if it receives 2f+1 (prepare, v), broadcast (commit, v)
  - if it receives 2f+1 (commit, v), broadcast (committed, v)
  - if it receives f+1 (committed, v), broadcast (committed, v)
- Ensure agreement
- Ensure liveness under an loyal commander
Practical Byzantine Fault Tolerance

- Commander sends the value to every lieutenant
- Every lieutenant
  - if it receives a new value $v$, broadcast (pre-prepare, $v$)
  - if it receives $2f+1$ (prepare, $v$), broadcast (commit, $v$)
  - if it receives $2f+1$ (commit, $v$), broadcast (committed, $v$)
  - if it receivers $f+1$ (committed, $v$), broadcast (committed, $v$)
- Ensure agreement
- Ensure liveness under an loyal commander
- What if the commander is faulty?
  - we need view change
Overview of the Core PBFT Algorithm

Request → Pre-Prepare → Prepare → Commit → Reply

Executed by Client

Executed by Replicas
Request

Client C sends a message to all replicas

```
m(v)  | [pre-prepare, 0, m, D(m)]  | [prepare, i, 0, D(m)]  | [commit, i, 0, D(m)]  | [reply, i, ...]
------|-----------------------------|-------------------------|------------------------|-----------------------
client C |                             |                         |                        |                       
replica 0 |                             |                         |                        |                       
replica 1 |                             |                         |                        |                       
replica 2 |                             |                         |                        |                       
replica 3 |                             |                         |                        |                       
```
Pre-prepare

- Primary (0) sends a signed pre-prepare message with the to all backups
  - It also includes the digest (hash) $D(m)$ of the original message

<table>
<thead>
<tr>
<th>m(v)</th>
<th>[pre-prepare, 0, m, D(m)]</th>
<th>[prepare, i, 0, D(m)]</th>
<th>[commit, i, 0, D(m)]</th>
<th>[reply, i, …]</th>
</tr>
</thead>
<tbody>
<tr>
<td>client C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Prepare

- Each replica sends a prepare-message to all other replicas
- It proceeds if it receives $2/3*N + 1$ prepare-messages consistent with its own

<table>
<thead>
<tr>
<th></th>
<th>m(v)</th>
<th>[pre-prepare, 0, m, D(m)]</th>
<th>[prepare, i, 0, D(m)]</th>
<th>[commit, i, 0, D(m)]</th>
<th>[reply, i, …]</th>
</tr>
</thead>
<tbody>
<tr>
<td>client C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>replica 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Commit

- Each replica sends a signed commit-message to all other replicas.
- It commits if it receives \( \frac{2}{3}N+1 \) commit-messages consistent with its own.

\[
\begin{array}{c|c|c|c|c}
 m(v) & [\text{pre-prepare, } 0, m, D(m)] & [\text{prepare, } i, 0, D(m)] & [\text{commit, } i, 0, D(m)] & [\text{reply, } i, \ldots] \\
\hline
\text{client C} & & & & \\
\hline
\text{replica 0} & & & & \\
\hline
\text{replica 1} & & & & \\
\hline
\text{replica 2} & & & & \\
\hline
\text{replica 3} & & & & \\
\end{array}
\]
• Each replica sends a signed response to the initial client
• The client trusts the response once she receives $N/3 + 1$ matching ones
What if Primary is compromised?

• Thanks to large quorums, it won’t break integrity of the good replicas

• Eventually, replicas and the clients will detect it via time-outs
  
  • Primary sending inconsistent messages would cause the system to “get stuck” between the phases, without reaching the end of commit

• Once a faulty primary is detected, backups-will launch a view-change, re-electing a new primary

• View-change is similar to reaching a consensus but gets tricky in the presence of partially committed values
  
  • See the Castro & Liskov ’99 PBFT paper for the details…
PBFT in Industry

- Widely adopted in practical developments:
  - Tendermint
  - IBM’s Openchain
  - Elastico/Zilliqa
  - Chainspace
- Used for implementing to speed-up blockchain-based consensus
- Many blockchain solutions build on similar ideas
  - Stellar Consensus Protocol, HotStuff
Minimum number required for which an $f$-resilient consensus protocol exists

<table>
<thead>
<tr>
<th></th>
<th>synchrony</th>
<th>asynchrony</th>
<th>partial synchrony</th>
</tr>
</thead>
<tbody>
<tr>
<td>fail-stop</td>
<td>$f+1$</td>
<td>$\infty$</td>
<td>$2f+1$</td>
</tr>
<tr>
<td>crash</td>
<td>$f+1$</td>
<td>$\infty$</td>
<td>$2f+1$ (Paxos)</td>
</tr>
<tr>
<td>byzantine with digital signature</td>
<td>$f+1$ (SM($f+1$))</td>
<td>$\infty$</td>
<td>$3f+1$ (PBFT)</td>
</tr>
<tr>
<td>byzantine with authenticated channel</td>
<td>$3f+1$ (OM($f$))</td>
<td>$\infty$</td>
<td></td>
</tr>
</tbody>
</table>
Conclusions

- Defined Byzantine generals problem
- Proved lower bound in synchronous environment with authenticated channel
- Introduced solutions in synchronous environment with authenticated channel and with digital signature
- PBFT Can be used only for a fixed set of replicas
  - Agreement is based on fixed-size quorums
  - Open systems (used in Blockchain Protocols) rely on alternative mechanisms of Proof-of-X (e.g., Proof-of-Work, Proof-of-Stake)
  - Also see Algorand
Timeline

1982
- The Byzantine Generals Problem
  - OM() sync/authenticated channel
  - SM() sync/digital signature

1990
- The part-time parliament
  - Paxos: async/non-byzantine(crash-failure)

1998
- Practical Byzantine Fault Tolerance
  - PBFT: partial sync/digital signature/state machine replication

2008
- Bitcoin: A peer-to-peer electronic cash system
  - Blockchain: partial sync/proof of work/state machine replication

2019
- Lots of improvements on PBFT
  - HotStuff
  - Stellar
  - Algorand