# An introduction to Libra and LibraBFT consensus algorithm

Maurizio Murgia[1]

[1]University of Trento

De Cifris Athesis Seminar — Trento, October 30

# Outline of the talk

- Brief overview of Libra blockchain.
- Background:
  - Byzantine consensus.
  - Distributed systems.
- Basic HotStuff.
- Chained HotStuff and LibraBFT.

# Motivation

Limits of traditional banking:

- Transferring money internationally is very expensive and slow.
- $\sim$ 1.8 billions of people do not have access to banks.

Limits of cryptocurrency:

- Scalability curse.
- Relatively fast for international transfers, but local payments...
- Solutions based on PoW waste energy.
- Expensive.
- Highly volatile.

# Goals

Infrastructure for moving money (and other assets). Should be:

- Global.
- Open, even to people without access to traditional banks.
- Instant.
- Low cost.
- Stable.

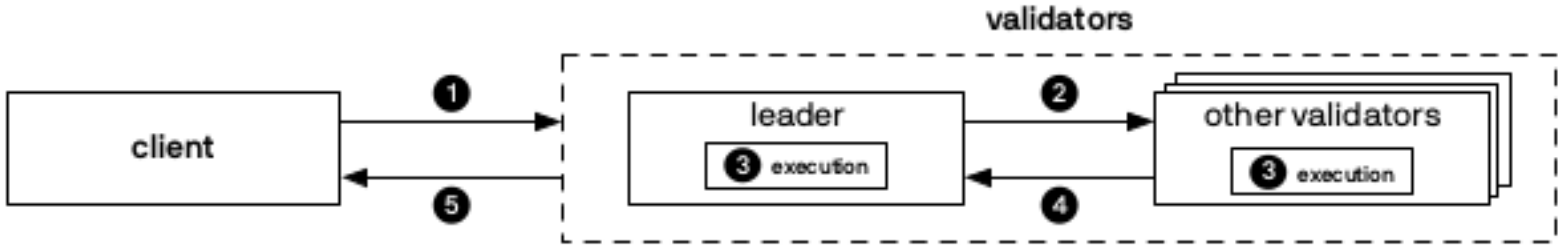
# Libra blockchain (1)

- 1k transactions per second, with latency of 10 seconds.
- Libra coin is backed by a reserve of assets designed to give it intrinsic value.
- User can deploy their own resources, accessible through smart contracts.

# Libra blockchain (2)

- Libra will be permissioned, at least for the first period. Transition to the permissioned setting has been planned to occur within five years.
- Libra uses a Byzantine Fault Tolerant consensus algorithm.
- Smart contracts: a new programming language, Move, with first class resources.

# On permissioned BFT-based blockchains




Pros:

- Fast.
- Scalable.
- Block finality.

Cons:

- Decentralised?
- Truly open?

# The Libra Association (1)

# The Libra Association (2)

From `https://libra.org`:

- The initial members of the council are the Founding Members and serve as the network's initial validator node.
- To be such a node, an entity needs to commit $10 million to the project.
- Each $10 million committed entitles one vote in the council, subject to a cap.
- The Libra Association Council will prevent related entities from presenting themselves as two distinct Founding Members in order to avoid the circumvention of the above measure.

## Calibra Wallet

Calibra will be the official wallet of Libra coins.

- Integrated with Facebook products:
  - Facebook.
  - Whatsapp.
  - Instagram.
- Sending money like sending a message or a photo.
- A bit like WeChat Pay in China.
- Potential for a great impact in people lives.

## Miscellaneus on Libra

- Work in progress.
- Release plan: half 2020.
- Open source.
- Testnet available.

# Blockchain? (1)

From the White Paper:

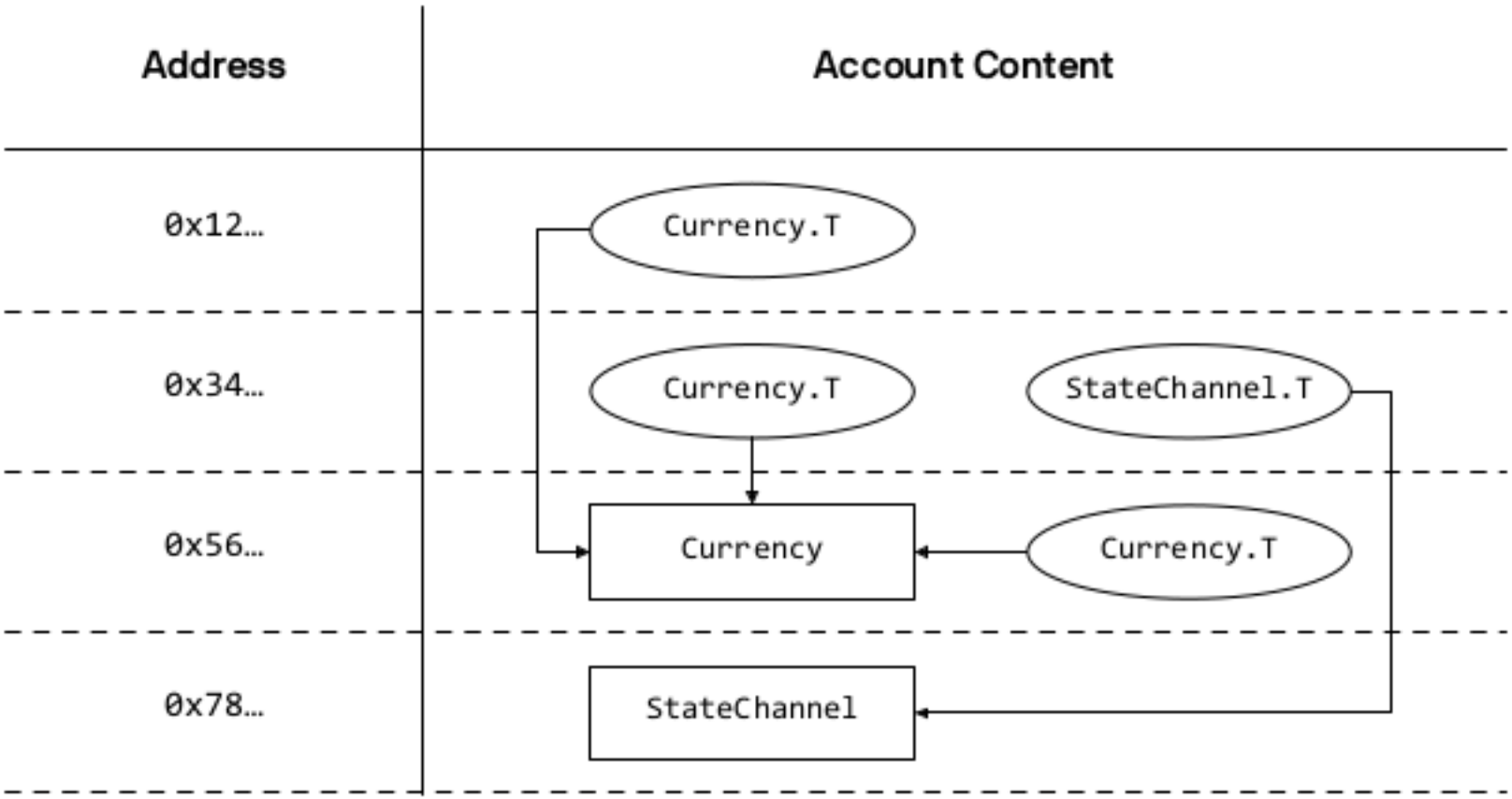
"Unlike previous blockchains, which view the blockchain as a collection of blocks of transactions, the Libra Blockchain is **a single data structure that records the history of transactions and states over time.** This implementation simplifies the work of applications accessing the blockchain, allowing them to read any data from any point in time and verify the integrity of that data using a unified framework."

# Blockchain? (2)

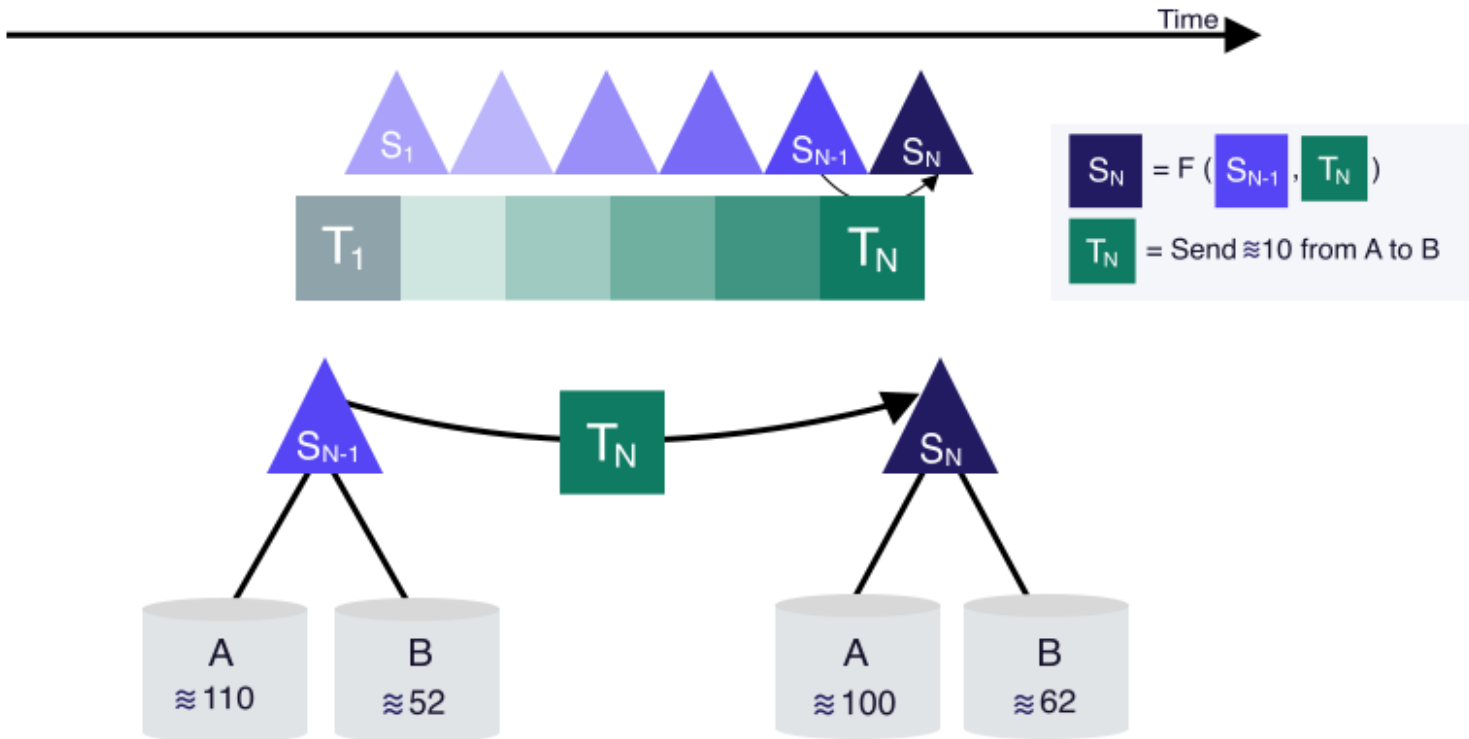
From the report of LibraBFT (State Machine Replication in the LibraBlockchain):

"The leader proposes a new **block consisting of transactions** and sends it to the rest of the validators, who approve the new block if it consists of valid transactions. Once the leader collects a majority of votes, she sends it to the rest of the validators. If a leader fails to propose a valid block or does not aggregate enough votes, a timeout mechanism will force a new round, and a new leader will be chosen from the validators. This way, **new blocks extend the blockchain**."

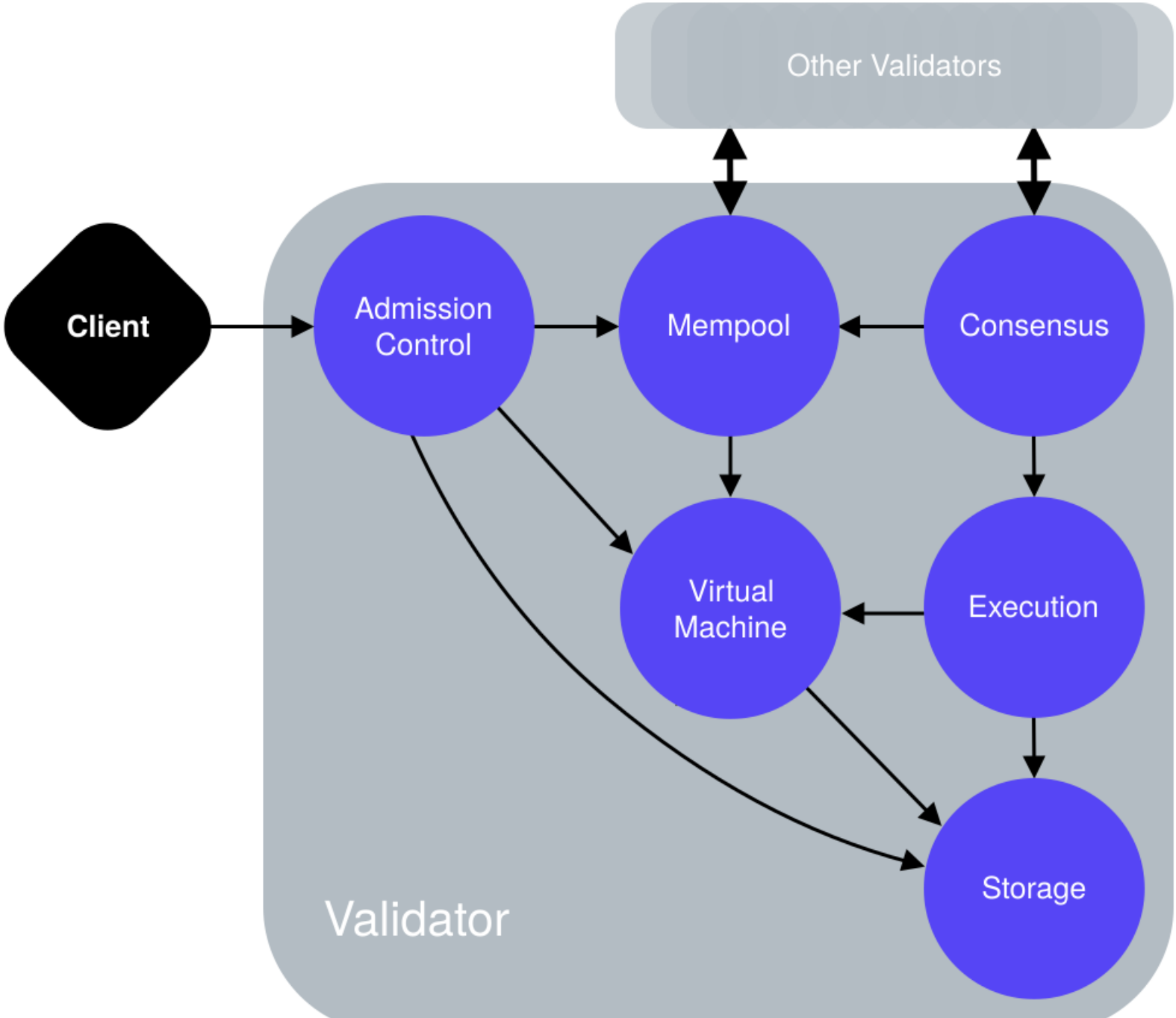
# Blockchain state



# Transactions



# Lifecycle of the Transaction

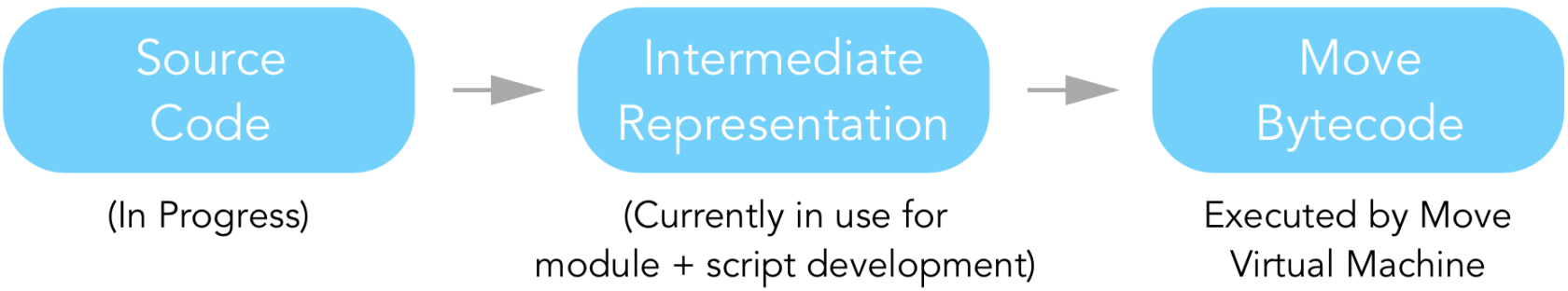


## Smart contracts

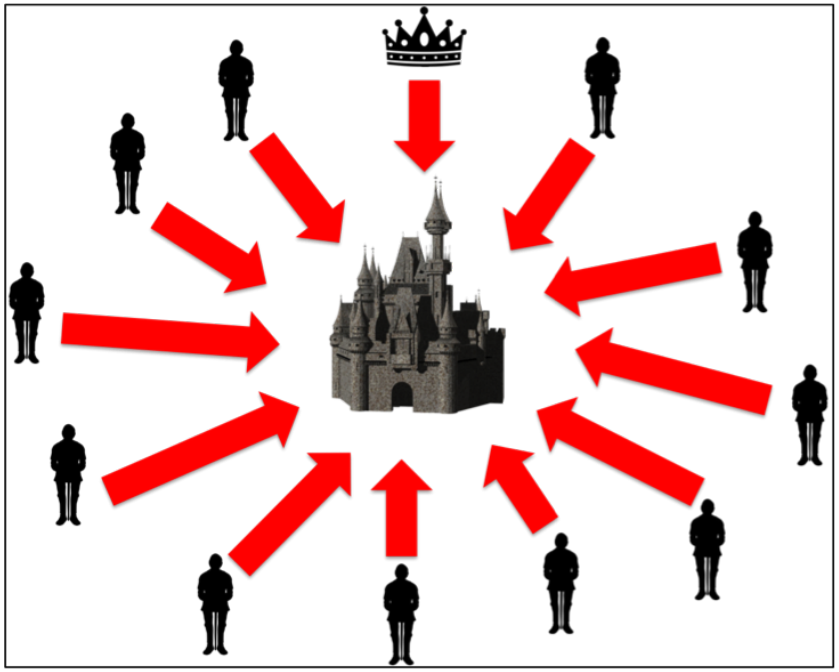
- Libra allows users to create custom resources.
- Resources might be exchanged, under the rules of the associated smart contract.
- Resources and smart contracts are implemented in the Move programming language
- Even Libra coin will be implemented as a smart contract.

# The move language

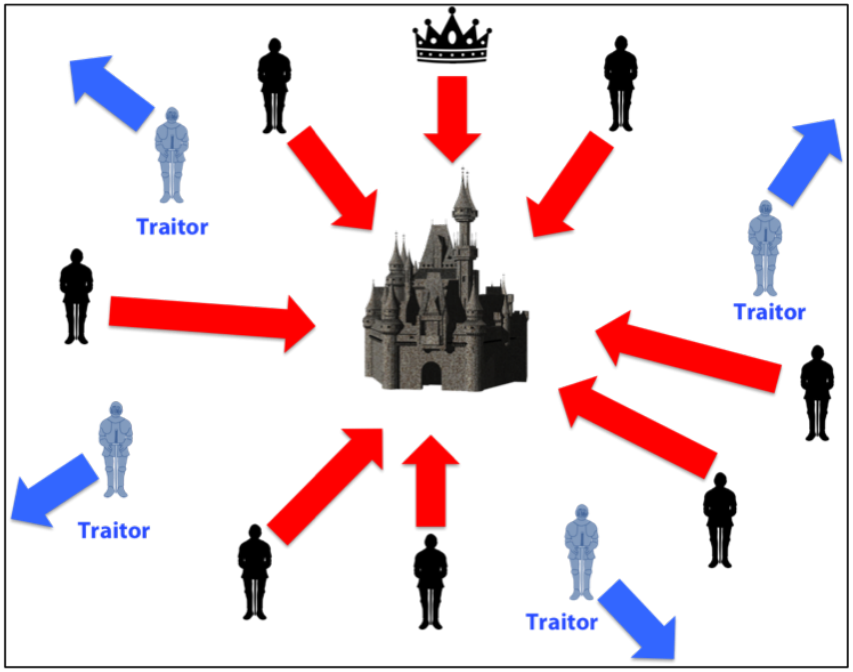
- Allows declaration of resources.
- Ensures that resources are not copied, but just moved.
- Linear type system.
- Termination is enforced with a gas mechanism.



# Byzantine Generals Problem (1)



Coordinated Attack Leading to Victory




Uncoordinated Attack Leading to Defeat

# The Byzantine Generals Problem (2)

- Participants can communicate only through message passing.
- Byzantine fault: faulty participants (traitors) can behave arbitrarily, e.g. sending misleading messages.
- Correctness:
  - IC1: All loyal lieutenants obey the same order.
  - IC2: If the commanding general is loyal, then every loyal lieutenant obeys the order he sends.
- No solution is possible if at least one third of participants are traitors (in partially synchronous networks).

# State Machine Replication (SMR)

According to Wikipedia: State machine replication [...] is a general method for implementing a fault-tolerant service by replicating servers and coordinating client interactions with server replicas.

- All replicas start with the same initial values.
- All replicas receiving the same request shall output the same execution result and end up in the same state.
- Byzantine version: replicas are subject to byzantine faults.
- The main challenge is guaranteeing that replicas execute the same requests in the same order.

## Network synchrony

Synchronous networks:

- Operations are coordinated in rounds.
- Message delivery time is bounded by a fixed delay $\Delta$.
- Not realistic!

Asynchronous networks:

- Processes can coordinate only with messages.
- Message delivery is guaranteed, but in unbounded time.
- Most general setting, consensus problem proved impossible for deterministic algorithms.

Partially synchronous networks:

- Network delay is unbounded.
- Long enough runs approximate synchronous runs.
- Basically, assumes that network faults are eventually fixed and DoS attacks eventually terminates: reasonable in practice!

# The GST model

- A simple partially synchronous model.
- Network is asynchronous, until Global Stabilisation Time (GST).
- After GST, network behaves synchronously.

## Correctness of distributed systems

Properties of distributed systems can be categorised as follows:

- Safety properties: something (bad) never happens.
- Liveness properties: something (good) eventually happens.

Safety and liveness in blockchain BFT consensus:

- Safety: two correct replicas never commit conflicting blocks.
- Liveness: a new block is eventually produced (after GST).

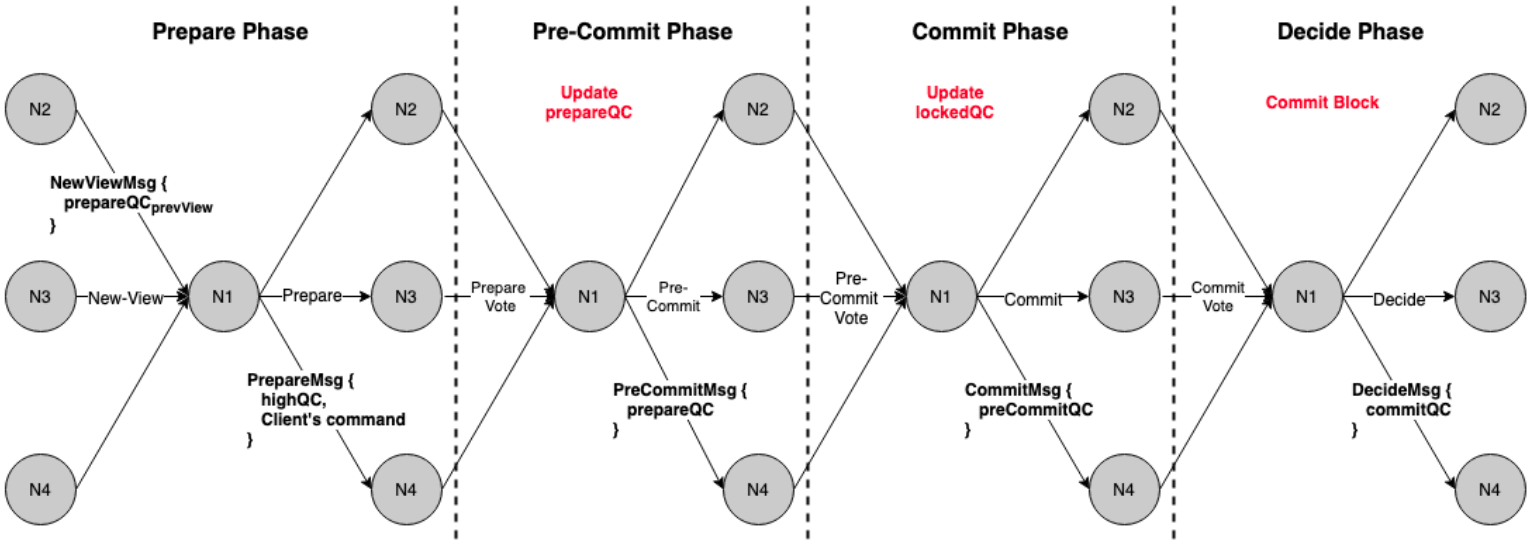
## HotStuff

- Recent BFT consensus algorithm.
- Fast leader election.
- Linear communication cost.
- Optimistic responsiveness: after GST, if the current leader is correct, consensus is reached at the pace of actual (vs. maximum) network delay.

## Preliminaries

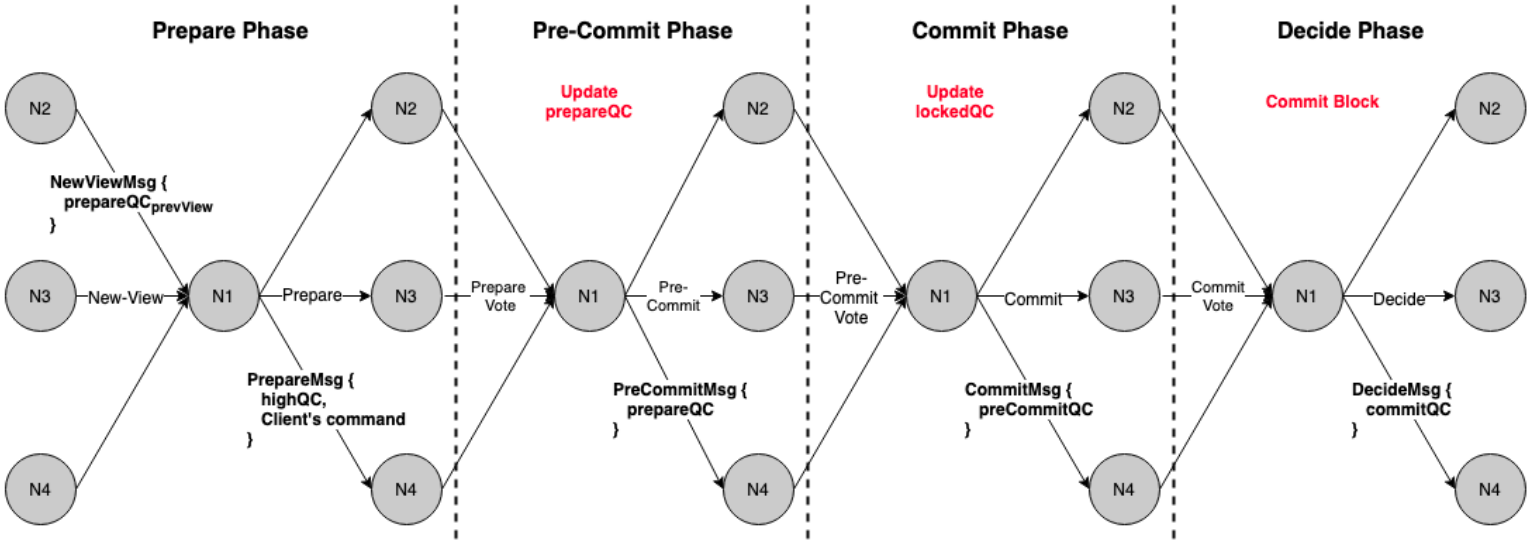
- $f$ number of byzantine replicas.
- $n$ total number of replicas.
- **BFT assumption:** $n \geq 3f + 1$.
- Fact: let $A$ and $B$ be two sets of replicas, with at least $2f + 1$ elements each. Then $A \cap B$ contains at least one correct replica.
- Quorum Certificate (QC): a set of $2f + 1$ (signed) votes of distinct replicas.
- View: a progressive number; each view has an associated leader.

# Basic HotStuff



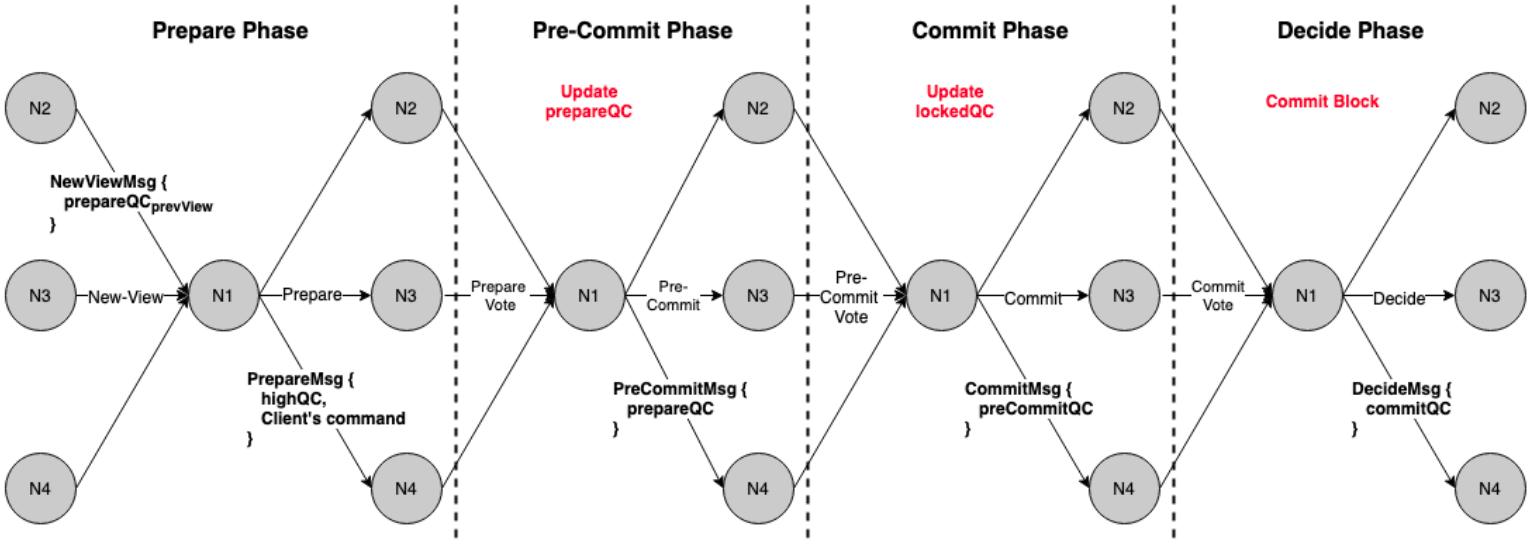

- Basic version of HotStuff, composed of three rounds.
- Note that replicas only speaks with the current leader.

# Prepare Phase



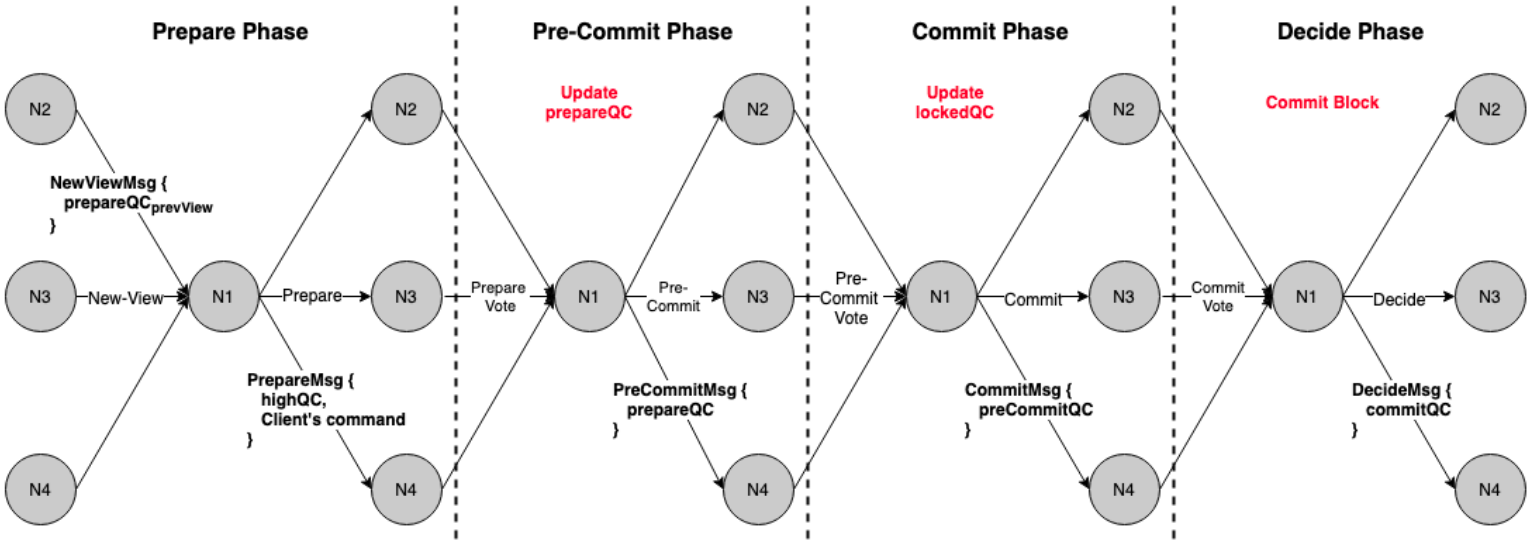

- Leader:
  - waits for $2f + 1$ NEW-VIEW messages (sent at the end of previous round), and selects the QC with maximal view number.
  - creates a new block, with parent the selected block, and broadcasts it in a PREPARE message.
- Replica:
  - waits for a PREPARE message, and accepts block if:
    - block extends from lockedQC (safety rule), or
    - $m.QC.view > lockedQC.view$ (liveness rule).
  - if message was accepted, sends vote.

# Pre-Commit Phase



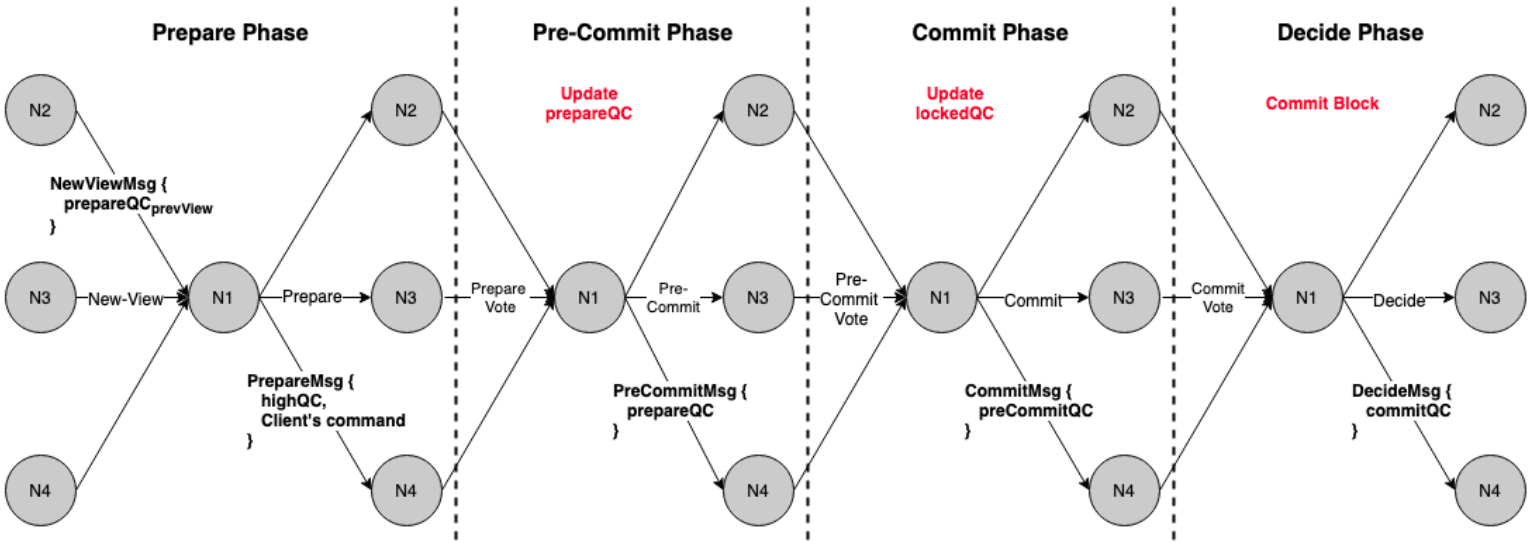

- Leader:
  - waits for $2f + 1$ PREPARE votes.
  - broadcasts a PRE-COMMIT message, with QC of received votes.
- Replica:
  - waits for a PRE-COMMIT message.
  - updates prepareQC.
  - sends vote.

# Commit Phase



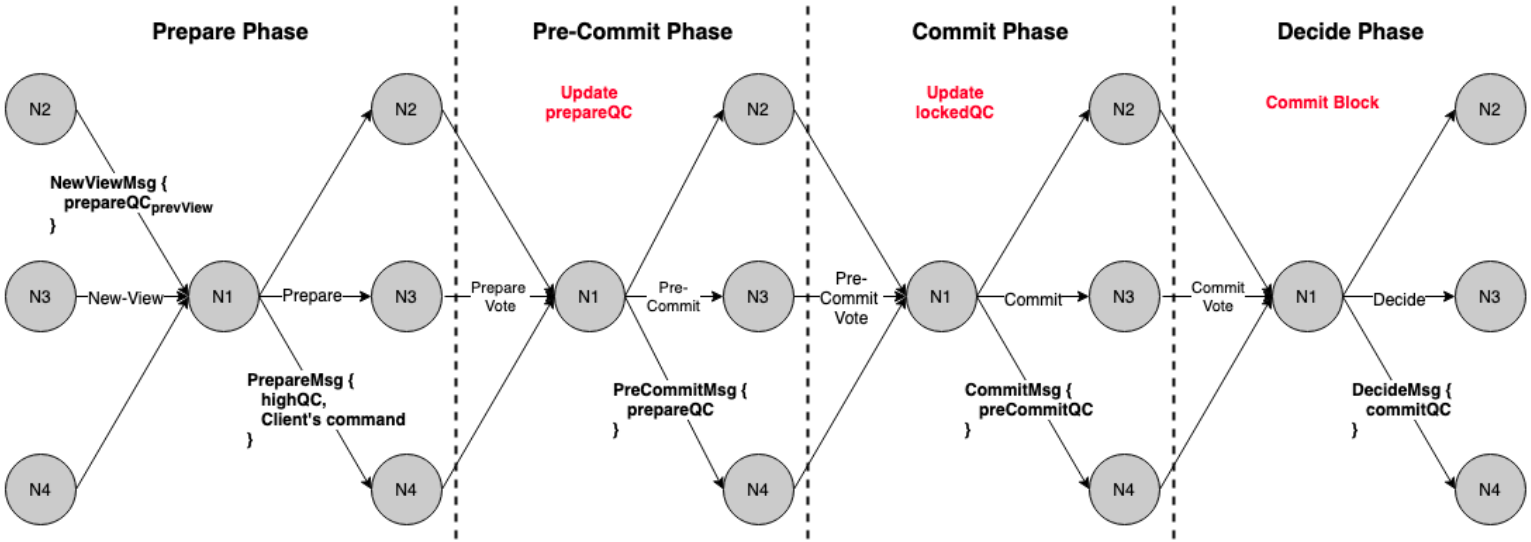

- Leader:
  - waits for $2f + 1$ PRE-COMMIT votes.
  - broadcasts a COMMIT message, with QC of received votes.
- Replica:
  - waits for a COMMIT message.
  - updates lockedQC.
  - sends vote.

# Decide Phase




- Leader:
  - waits for $2f + 1$ COMMIT votes.
  - broadcasts a DECIDE message, with QC of received votes.
- Replica:
  - waits for a DECIDE message.
  - increments current view, and sends a NEW-VIEW message.

# Timeouts




- If an expected message does not arrive, a timeout is triggered.
- A timedout replica increments current view, and sends a NEW-VIEW message.

## Safety of Basic HotStuff

- Fact 1: two conflicting QC of the same type must refer to different views.
- Otherwise, a correct replica voted twice in the same phase: contradiction!

**Theorem (Safety)**

*If two blocks are conflicting, they cannot be both commited, each by a correct replica.*

## Safety "proof"

- By contradiction: suppose $b1$ and $b2$ are conflicting and committed. They must have two QC $qcCommit1$ and $qcCommit2$ of type COMMIT.
- By Fact 1 they have different view numbers, say $v1$ and $v2$.
- Suppose w.l.o.g. $v1 < v2$.
- Let $qcPrepare3$ be the minimal QC of type PREPARE with view number $v1 < v3 \leq v2$ and conflicting with $b1$.
- Let $r$ be a correct replica that voted both in $qcCommit1$ and $qcPrepare3$.
- At view $v1$, $r$ updated her lockedQC to a precommitQC on $b1$.
- When $r$ reached the PREPARE message that she voted in $qcPrepare3$, her lockedQC was still a precommitQC on $b1$ (due to minimality).
- Then both safety and liveness rules does not hold, and $r$ could not vote for $qcPrepare3$: contradiction!

## On liveness

- Fact 2: If a correct replica il locked such that lockedQC = precommitQC, then at least $f + 1$ correct replicas voted for some prepareQC matching lockedQC.
- PrepareQC was voted by at least $2f + 1$ replicas. The thesis follows as there are at most $f$ byzantine replicas.

**Theorem**

*After GST, there exists a bounded time period $T_f$ such that if all correct replicas remain in view $v$ during $T_f$ and the leader for view $v$ is correct, then a decision is reached.*

## "Proof"

- At the beginning of the round, the leader receives $2f + 1$ NEW-VIEW, calculates its highQC and sends a PREPARE message.
- Let precommitQC* be the highest lockedQC among correct replicas.
- By Fact 2 there are at least $f + 1$ correct replicas that voted for a prepareQC* matching precommitQC*, and have sent it to the leader in the NEW-VIEW message.
- The leader must have received (at least) one of such messages, and hence highQC matches precommitQC*.
- By assumption, all replicas are in view $v$ and the leader is correct.
- Then, all correct replicas vote in the PREPARE phase, because liveness rule holds.
- As we are after GST, all the phases terminate in bounded time, which can be set to be $T_f$.

## The pacemaker

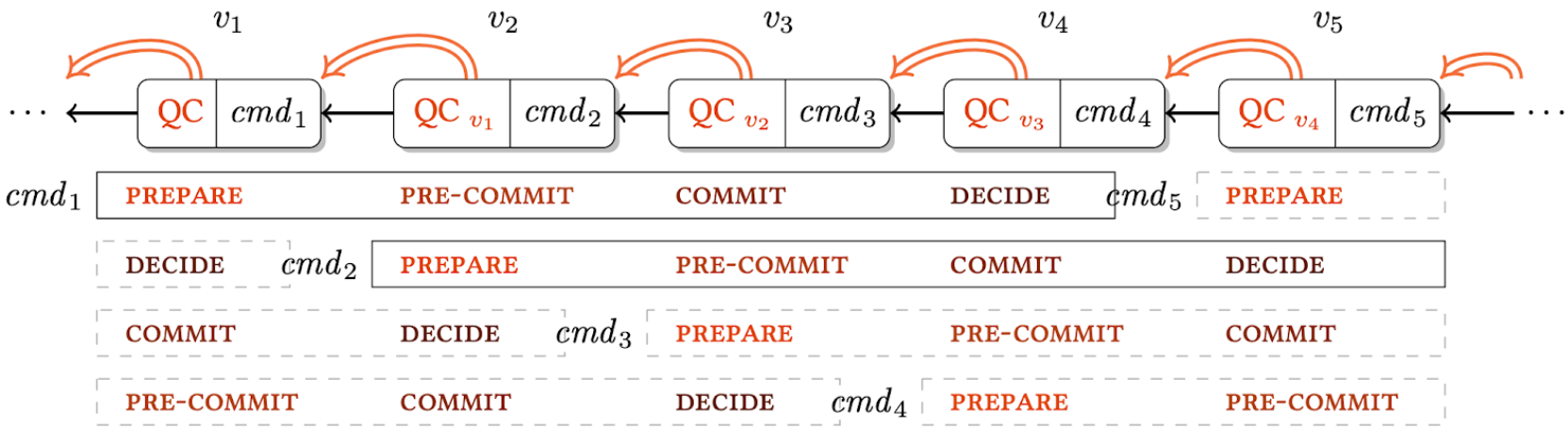
The pacemaker of HotStuff is a module which handles leader selection and timeout duration. By the above theorem, in order to obtain liveness, we need that:

- A correct leader is eventually elected.
- All correct replicas eventually stay in the same view for $T_f$.

They can be achieved as follows:

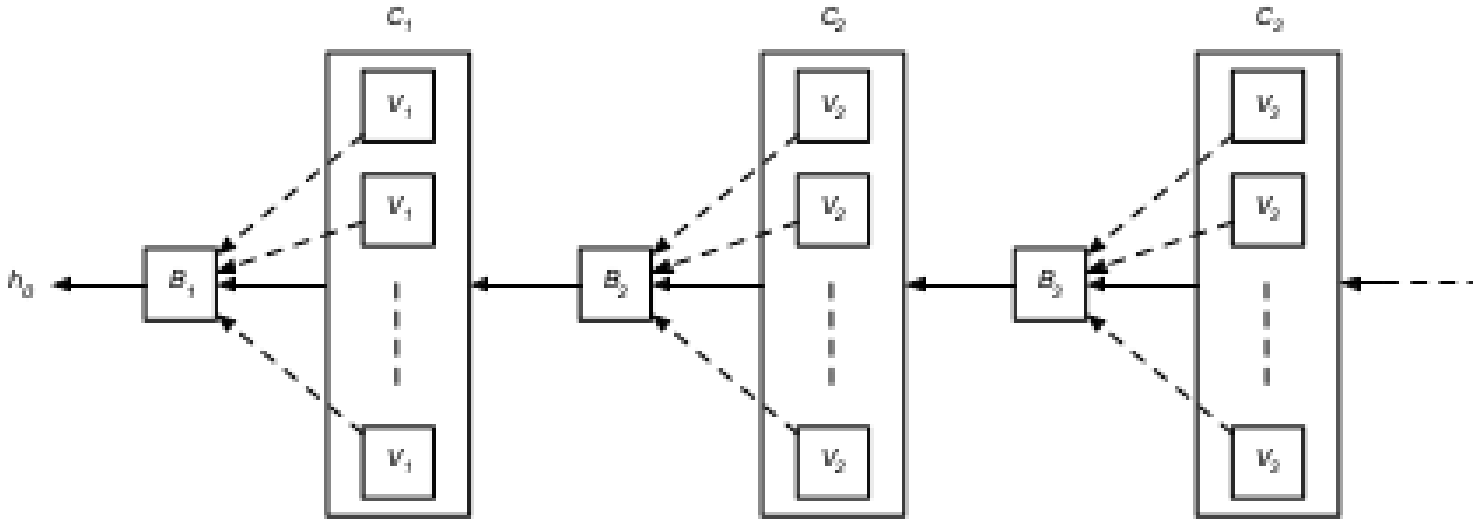
- Leader selection must guarantee that every replica eventually becomes a leader: any deterministic mapping that rotates all leaders is fine.
- At each round, double timeout duration (and start from a non zero value!).

# Chained HotStuff




- A vote for a block is an implicit vote for parent block.
- Three chain rule: a block (and its ancestors) is final if it is at the head of three contiguos blocks.

# LibraBFT




- Based on Chained HotStuff.
- Several low level details.
- Some changes in the chain structure.
- Supports updates of validators.
- Customised Pacemaker.
- Leader selection: work in progress, probably based on Verifiable Random Functions.

# Thanks!