Blockchains in the lens of BFT

Dahlia Malkhi
Diem Association and Novi

State-Machine-Replication (SMR)
with
Byzantine Fault Tolerance (BFT)

SIFT [1976]

A mission critical spacecraft control system is crafted with redundant sensors and compute units

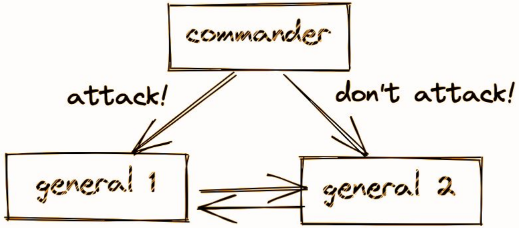
Sensors and compute units might fail arbitrarily

Control commands are exerted by consensus voting among units

State-Machine-Replication (SMR)
with
Byzantine Fault Tolerance

Byzantine Generals [LPS 1980]

Timeless foundations of concurrency and reliability
- consensus
- fault models
- solutions, impossibilities and lower bounds



State-Machine-Replication (SMR) with Byzantine Fault Tolerance

<u>SIFT [1976]</u>

A mission critical spacecraft control system is crafted with redundant sensors and compute units

Sensors and compute units might fail arbitrarily

Control commands are exerted by consensus voting among units

<u>Double Spend [2008]</u>

Money in its digital form requires keeping a ledger of transfers

This is easy to prevent if there is a trusted entity maintaining a centralized ledger

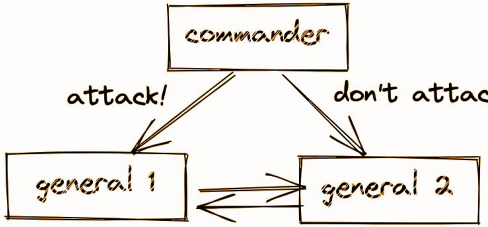
Users might try to duplicate coins or double-spend their balance

SMR forms agreement on a ledger among mistrusting parties

State-Machine-Replication (SMR)
with
Byzantine Fault Tolerance

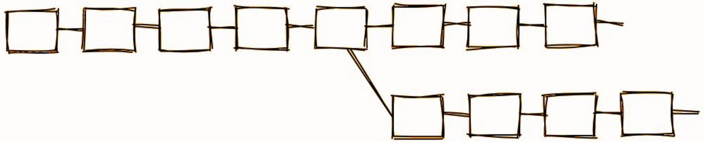
Byzantine Generals [LPS 1980]

Timeless foundations of concurrency and reliability
- consensus
- fault models
- solutions, impossibilities and lower bounds



Nakamoto Consensus [N2008]

New settings and use-cases:
- scale
- geo distributed interconnect
- incentives



Outline

What are we trying to solve

Classical SMR results

Enter Partial Synchrony

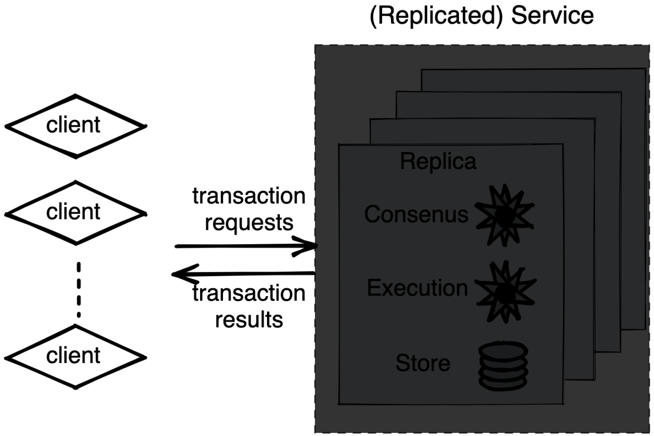
Bitcoin and Nakamoto Consensus

Scaling BFT

State-Machine-Replication (SMR) [L1978, S1990]

Untrusted/unreliable individual component, trusted/reliable service as whole

Core approach: A single server modeled as a **deterministic** state-machine, then replicated for fault tolerance



State-Machine-Replication (SMR) [L1978, S1990]

Untrusted/unreliable individual component, trusted/reliable service as whole

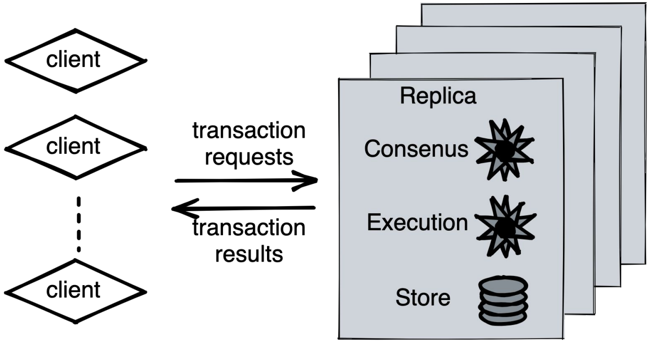
Core approach: A single server modeled as a **deterministic** state-machine, then replicated for fault tolerance

Linearizability [HW1990]: Correct execution modeled as a sequential state-machine, receive client requests, execute, store output, return response

Replicas have three key functions: Ordering, Execution, Store

Often, the same parties (**validators**, nodes, replicas, ..) provide all functions.

Authenticated store: succinct proofs of membership



SMR and Consensus

The main building block for SMR is **log replication**

It is reducible to a sequence of single-shot Consensus decisions

Much of the academic literature focuses on the Consensus problem, including important **impossibilities and lower bounds**

There are differences: receiving request from clients and sending output to them changes what is considered **valid** as output, and when is it **solvable**

SMR practical solution optimize a sequence of single-shot decisions with a (cheaper) **steady-state** leader regime and a (more complex) **view-change**

SMR Problem Model

Known set of N validators

Safety - validators store and execute the same log of transactions

Liveness - every client request is eventually executed by validators

(External) Validity - transactions are (signed) requests by clients

Fault Model

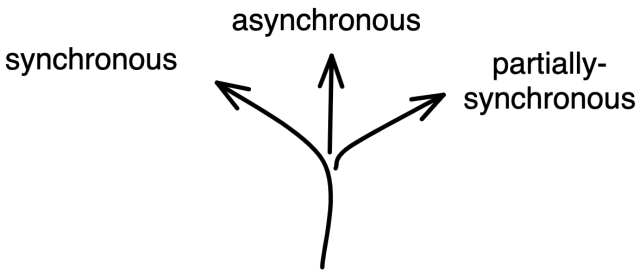
Communication model: modeled as an **adversary** that controls the network

Synchronous model – there is a known bound Δ on message transmission delays imposed by the adversary

Asynchronous model – the adversary can cause unbounded delays

Partial synchrony model – there is **Global Stabilization Time (GST)** after which there is a known bound Δ on message transmission delays imposed by the adversary

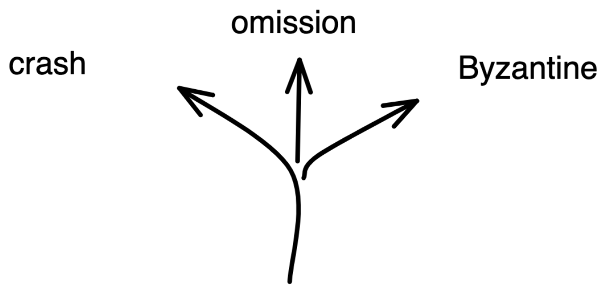
synchronous < partial synchrony < asynchronous



Fault Model

Decentralization, trust(less) systems and the Byzantine faults: modeled as an **adversary** that corrupts validators

- **Fraction of faults**: threshold, probabilistic, power, incentives
- **Failure modes**: crash, omission, Byzantine
- **Authentication**: confidential messages, signatures



crash < omission < Byzantine

Outline

What are we trying to solve

Classical SMR results

Enter Partial Synchrony

Bitcoin and Nakamoto Consensus

Scaling BFT

Which model should I use?

Under asynchrony:

- [FLP 1985] Liveness is not guaranteed against even a single failure, and a log replication algorithm must have (under network duress) non-terminating executions

Under partial synchrony:

- [Folklore] Under network transmission delays, Consensus requires $F < N/2$.
- [DLS 1988] Under network transmission delays, Byzantine Consensus requires $F < N/3$.

Under synchrony:

- [FLM 1985] If there is no public key setup, Byzantine Consensus requires $F < N/3$
- [Folklore] Under omissions faults, Consensus requires $F < N/2$
- [AT 1999] Consensus must have executions with $F+1$ rounds
- [DR 1982] Consensus must have executions with a quadratic number of messages.

	safety against asynch?	liveness against asynch?	progress during synch?
$f < n/3$	✓	✗	net speed
$n/3 \le f < n/2$	✗	✓	slow
$n/2 \le f$	✗	✗	✗

Outline

What are we trying to solve

Classical SMR results

Enter Partial Synchrony

Bitcoin and Nakamoto Consensus

Scaling BFT

Practical BFT Settings [LPS 1982, DLS 1988, L1989-1998, CL1999]

Partial synchrony model

$N = 3F+1$ permissioned/known validators

PKI enables validators to sign messages

Adversary controls up to F validators

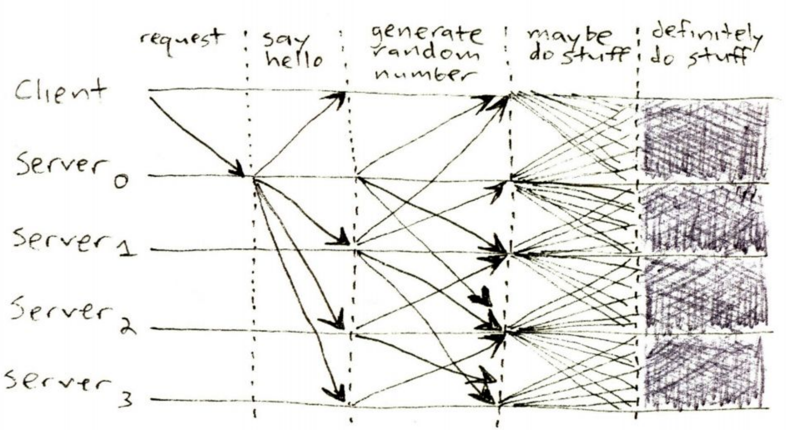
Focus on a single agreement decision

Classical BFT SMR

Super quadratic communication

Error-prone

Developer-unfriendly



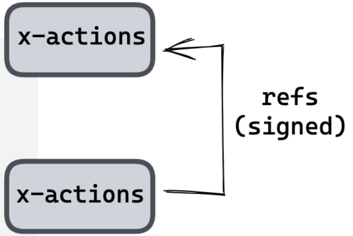
J. Mickens, 2013

BFT in the Lens of Blockchains

Simple and transparent

Blocks carry client-requests + signed-references (chaining)

Chain rules to participate and to commit finality



PBFT[CL1999] in the Lens of Blockchains

<u>Steady leader protocol:</u>

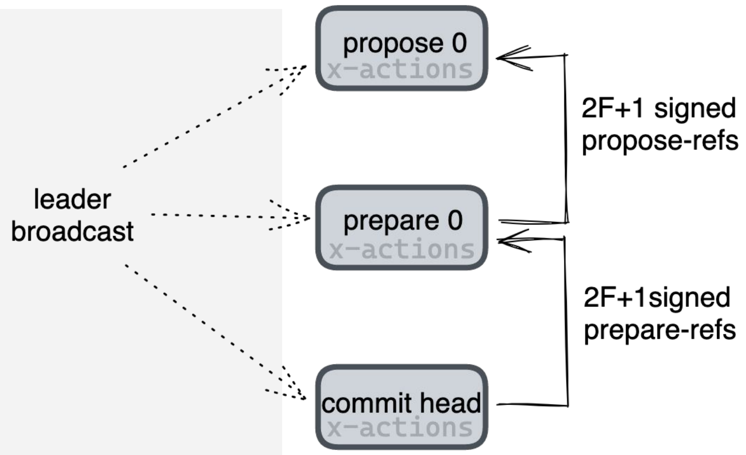
Broadcast blocks to validators
(e.g., via gossip)

First round: 2F+1 signed **proposal**-refs to prepare

Second round: 2F+1 signed **prepare**-refs to commit

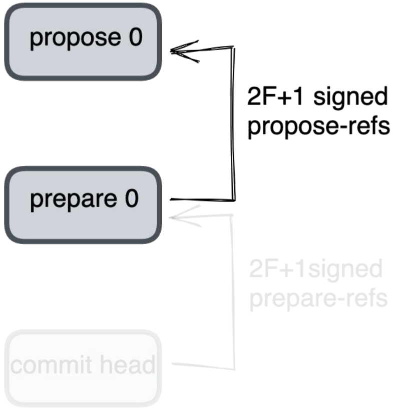
Only the **head** of chain committed

Omitted: chaining, pipelining



PBFT in the Lens of Blockchains

View-change protocol (by new leader):



PBFT in the Lens of Blockchains

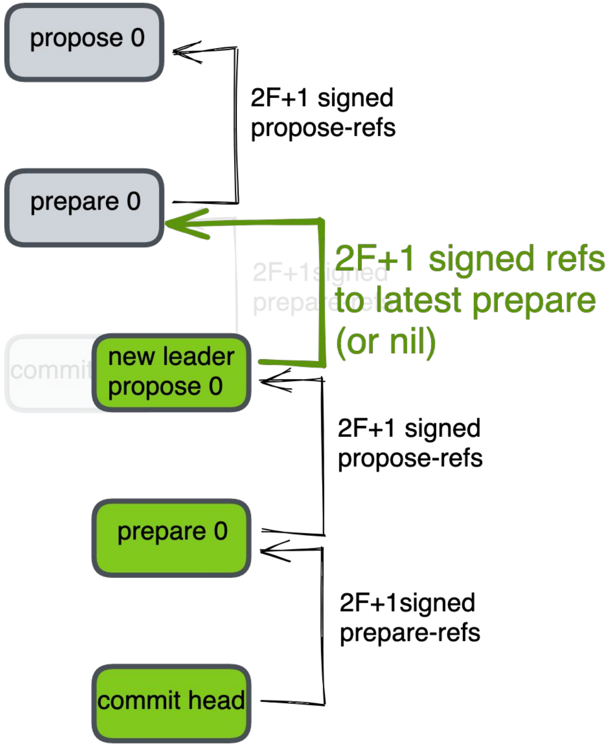
<u>View-change protocol (by new leader):</u>

Broadcast **justified** proposal carrying 2F+1 signed prepare-refs

Safety: a leader cannot hide a previous commit

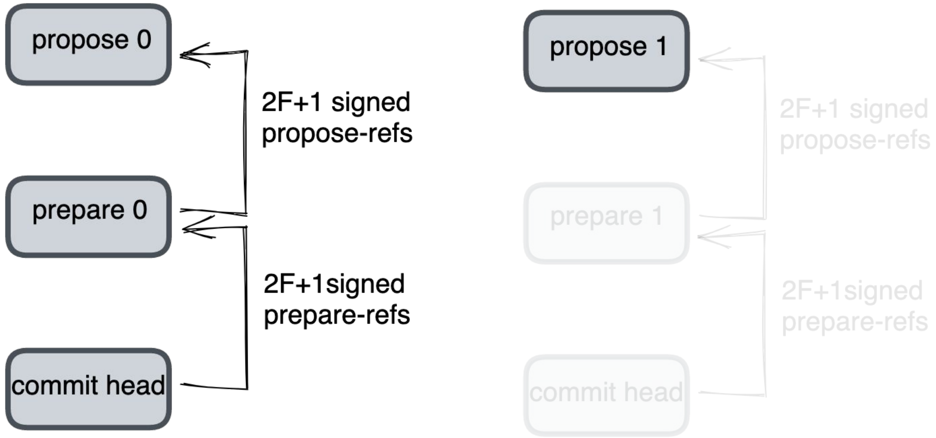
- F may be nil
- F may lie
- At least one must refer to prepare if it has been committed

Liveness: a leader can elicit 2F+1 latest-prepare refs



PBFT in the Lens of Blockchains

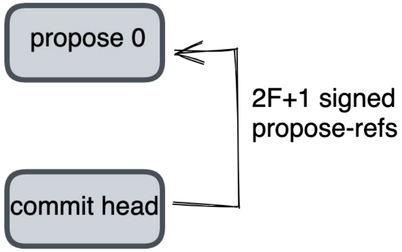
Why 2F+1?



PBFT in the Lens of Blockchains

Why two rounds?

Imagine a one-round protocol

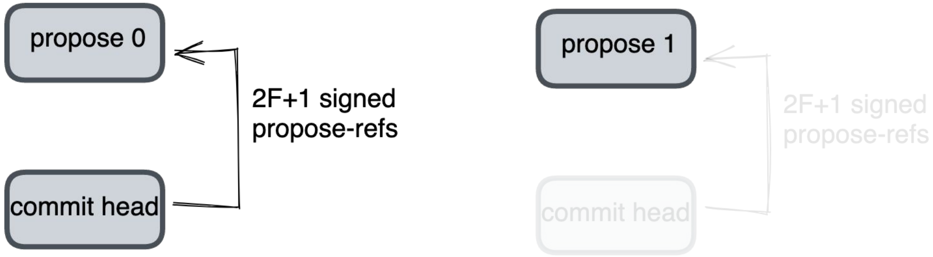


PBFT in the Lens of Blockchains

Why two rounds?

Imagine a one-round protocol

It can prevent equivocation by the first leader



BFT in the Lens of Blockchains

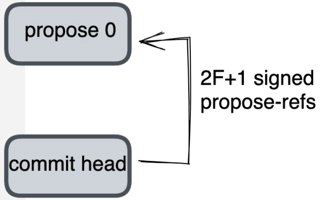
Why two rounds?

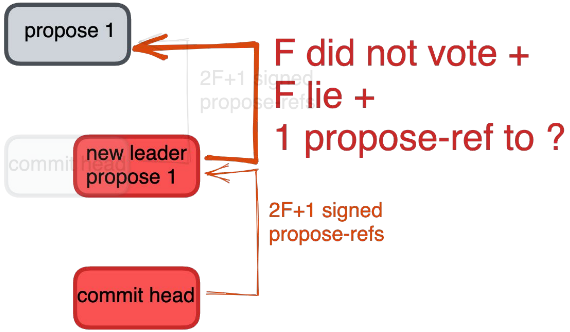
Imagine a one-round protocol

It can prevent equivocation by the first leader

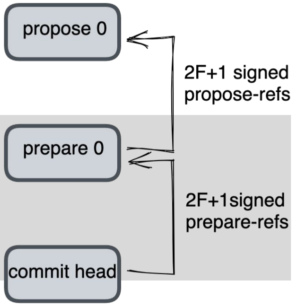
But it cannot convince a new leader of a commit outcome

Special case: Leader itself can commit after a single round [DLS1988]





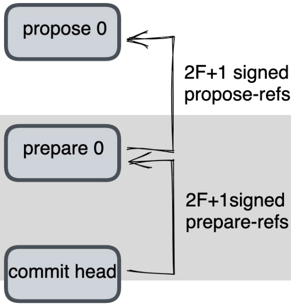
PBFT Complexity



How do we measure complexity?

Count cryptographic validations

PBFT Complexity



How do we measure complexity?

Count cryptographic validations

Steady leader protocol:

Broadcast proposal to participants (e.g., via gossip)

- O(N) to validate leader proposal

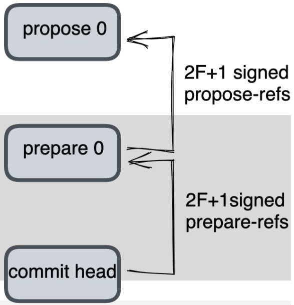
First round: 2F+1 signed proposal-refs to prepare

- O(N x N) to validate leader prepare carrying O(N) signatures on propose-refs

Second round: 2F+1 signed prepare-refs to commit

- same

PBFT Complexity



How do we measure complexity?

Count cryptographic validations

Steady leader protocol:

Broadcast proposal to participants (e.g., via gossip)

- O(N) to validate leader proposal

First round: 2F+1 signed proposal-refs to prepare

- O(N x N) to validate leader prepare carrying O(N) signatures on propose-refs

Second round: 2F+1 signed prepare-refs to commit

- same

View-change protocol (by new leader):

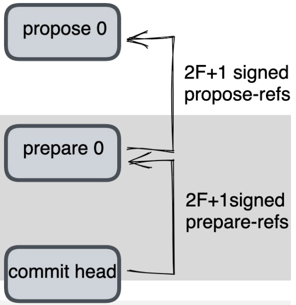
Broadcast **justified** proposal :
2F+1 signed prepare-refs (possibly different), each prepare contains 2F+1 signatures on propose-refs

- O(N x N^2) to validate leader proposal with:
  O(N) signatures on
  O(N) signed propose-refs

Cascading view-changes:

- O(N) x O(N^3)

Vote Aggregation [CKPS2001] and SBFT [GA+2019]



How do we measure complexity?

Count cryptographic validations

Steady leader protocol:

Broadcast proposal to participants (e.g., via gossip)

- O(N)/O(N) to validate leader proposal

First round: 2F+1 signed proposal-refs to prepare

- O(N x N)/O(N) to validate leader prepare carrying O(N)/O(1-aggregate) signatures on propose-refs

Second round: 2F+1 signed prepare-refs to commit

- same

View-change protocol (by new leader):

Broadcast **justified** proposal :
2F+1 signed prepare-refs (possibly different), each prepare contains 2F+1 signatures on propose-refs

- O(N x N^2)/O(NxN) to validate leader proposal with:
  O(N) (non-aggregate-able) signatures on
  O(N)/O(1-aggregate) signed propose-refs

Cascading view-changes:

- O(N) x O(N^3)/O(N) x O(N^2)

Practical BFT SMR for Partial Synchrony

	LPS 1982	DLS 1988	PBFT 1999
Safe against F < N/3 byz faults	👍	👍	👍
Safe against asynchrony	👎	👍	👍
Number of messages to consensus decision	poly	poly	quadratic*
Number of messages to rotate leader	poly	poly	quadratic
Network speed	N/A	👎	👍

*Can be linear with threshold-cryptography

Outline

What are we trying to solve

Classical SMR results

Enter Partial Synchrony

Bitcoin and Nakamoto Consensus

Scaling BFT

Bitcoin/Nakamoto Consensus [N2008]

NYTimes piece on Bitcoin [Andreesen, 2014]: "*Bitcoin is the first practical solution to a longstanding problem in computer science called the Byzantine Generals Problem.*"

Bitcoin/Nakamoto Consensus

Nakamoto Consensus (NC) is based on two mechanisms.

Proof-of-work
"*Pricing via Processing or Combatting Junk Mail*" [DN 1992]
This creates scarcity, a new coin can be minted every X time period

Hash chains
"*How to timestamp a Digital Document*" [HS92]
This creates an incentive for agreement, a coin has value only if it is part of the longest existing chain

Bitcoin/Nakamoto Consensus

Nakamoto Consensus (NC) is based on two mechanisms.

<u>Proof-of-work</u>
"*Pricing via Processing or Combatting Junk Mail*" [DN92]
This creates scarcity, a new coin can be minted every X time period

<u>Hash chains</u>
"*How to timestamp a Digital Document*" [HS 1992]
This creates an incentive for agreement, a coin has value only if it is part of the longest existing chain

Bitcoin/Nakamoto Consensus

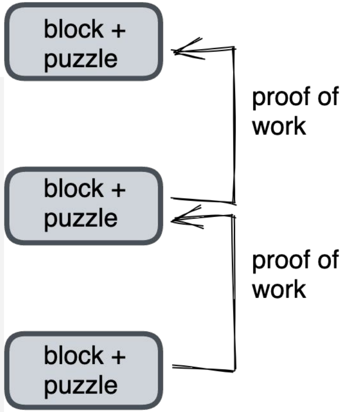
Nakamoto Consensus (NC) is based on two mechanisms.

Proof-of-work
"*Pricing via Processing or Combatting Junk Mail*" [DN92]
This creates scarcity, a new coin can be minted every X time period

Hash chains
"*How to timestamp a Digital Document*" [HS 1992]
This creates an incentive for agreement, a coin has value only if it is part of the longest existing chain

Putting them together

Puzzle solution must becomes part of the chain for mining/transfers to have effect



Bitcoin/Nakamoto Consensus

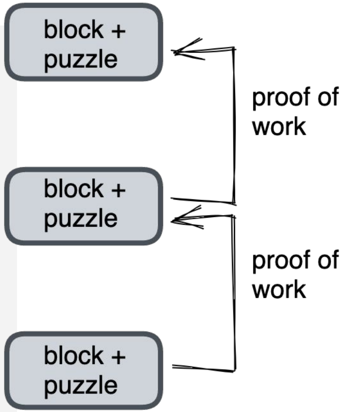
In order to participate in NC, a validator needs to mine blocks and append them to the chain.

NC is based on the following three rules:

1. **Longest fork wins**. A validator adopts the longest proof-of-work (PoW) chain to its knowledge (breaking ties arbitrarily) and attempts to mine a new block that extends this longest chain.

2. **Propagation**. Upon adopting a new longest chain, either through mining or by receiving from others, a validator broadcasts the newly acquired block(s);

3. **k-depth commit**. A validator commits a block if it is buried at least k blocks deep in the longest chain adopted by the validator. Here, k is a security parameter (6 is common in practice) that controls the probability of incorrect commit

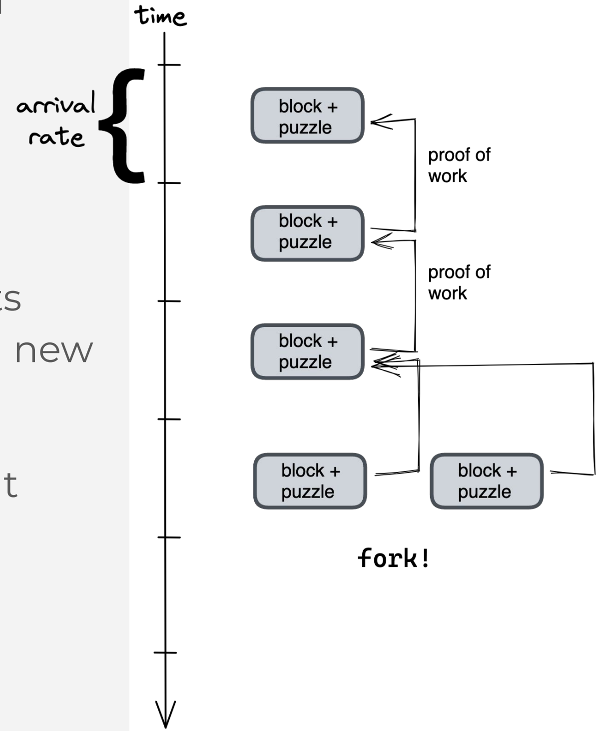


Bitcoin/Nakamoto Consensus

In order to participate in NC, a validator needs to mine blocks and append them to the chain.

NC is based on the following three rules:

1. **Propagation**. A validator broadcasts the each new block
2. **Longest fork wins**. A validator adopts the longest chain to its knowledge (breaking ties arbitrarily) and attempts to mine a new block that extends this longest chain.
3. **k-depth commit**. A validator commits a block if it is buried at least k blocks deep in the longest chain

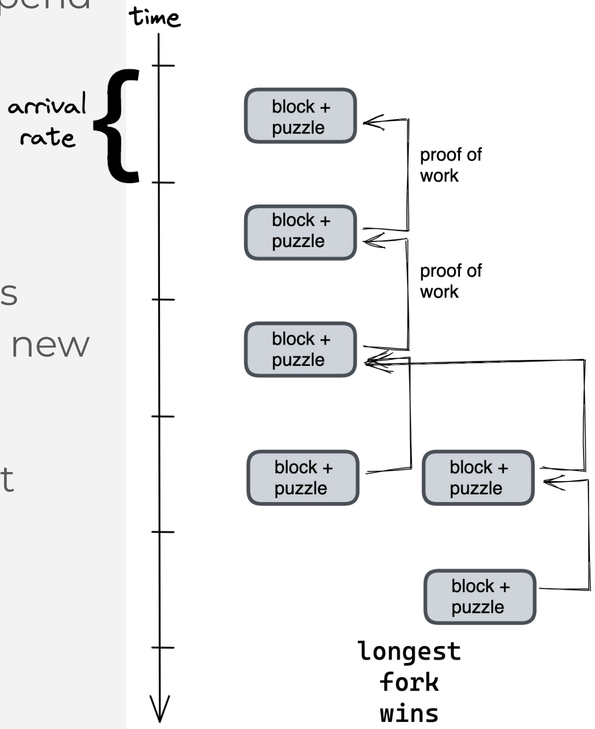


Bitcoin/Nakamoto Consensus

In order to participate in NC, a party needs to mine blocks and append them to the chain.

NC is based on the following three rules:

1. **Propagation**. A validator broadcasts the each new block
2. **Longest fork wins**. A validator adopts the longest chain to its knowledge (breaking ties arbitrarily) and attempts to mine a new block that extends this longest chain.
3. **k-depth commit**. A validator commits a block if it is buried at least k blocks deep in the longest chain



When is NC Safe?

[R2019, N2021]

NC In the Len of BFT

Imagine a round-based protocol among a set of N validators, with F Byzantine

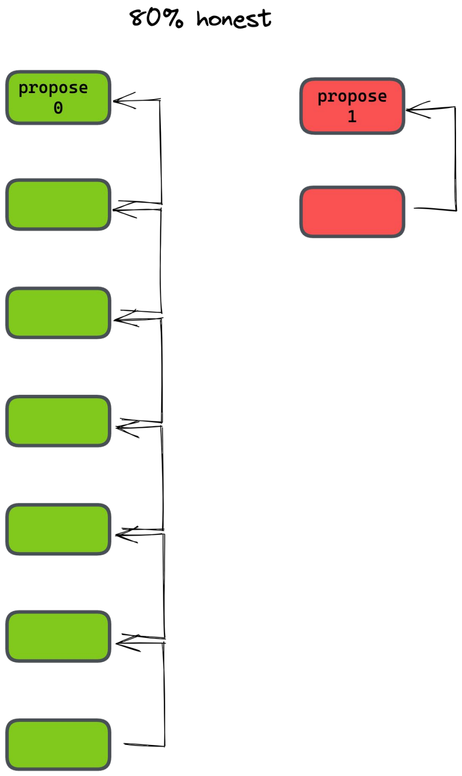
In each round, one validator selected uniformly at random to broadcast a proposal/vote

Honest validators extend the longest chain they know

Byzantine validators may extend any chain they choose

> The best attack strategy is to maintain their own chain k levels deeps and then expose it

If 49% are Byzantine then the bad chain will be longer than the good chain with probability exponentially small in k



NC In the Len of BFT

Imagine a round-based protocol among a set of N validators, with F Byzantine

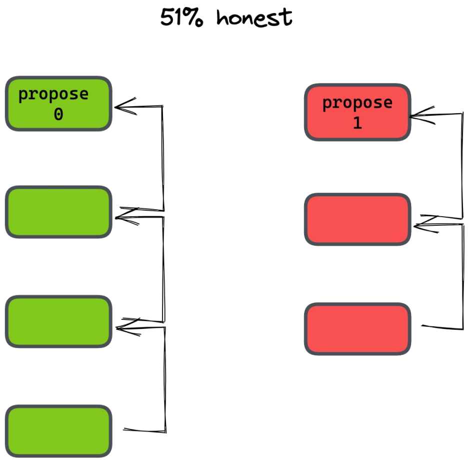
In each round, one validator selected uniformly at random to broadcast a proposal

Honest validators extend the longest chain they know

Byzantine validators may extend any chain they choose

> The best attack strategy is to maintain their own chain k levels deeps and then expose it

If 49% are Byzantine then the bad chain will be longer than the good chain with probability exponentially small in k



NC In the Len of BFT

In NC, there are no rounds and no known set of participants

Mining is modeled as a Possion process with expected interval *g*

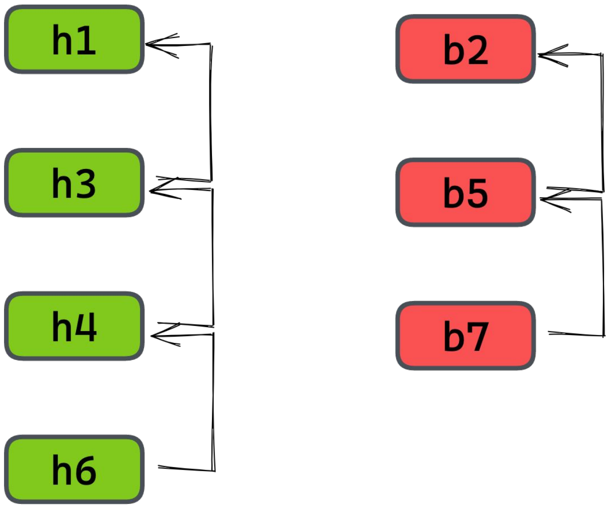
The adversary has power P < 50% to mine

We mark the arrivals of the process h1, b2, h3, h4, b5, h6, b7, ..

A proposal by a good validator takes less than Δ to propagate

since $g \gg \Delta$ we ignore the possibility of two honest arrivals within < Δ

A fork succeeds if there is a segment with more than k Byzantine arrivals and less than k honest arrivals



Outline

What are we trying to solve

Classical SMR results

Enter Partial Synchrony

Bitcoin and Nakamoto Consensus

Scaling BFT

Practical BFT SMR for Partial Synchrony

	LPS 1982	DLS 1988	PBFT 1999
Safe against F < N/3 byz faults	👍	👍	👍
Safe against asynchrony	👎	👍	👍
Number of messages to consensus decision	poly	poly	quadratic*
Number of messages to rotate leader	poly	poly	quadratic
Network speed	N/A	👎	👍

*Can be linear with threshold-cryptography

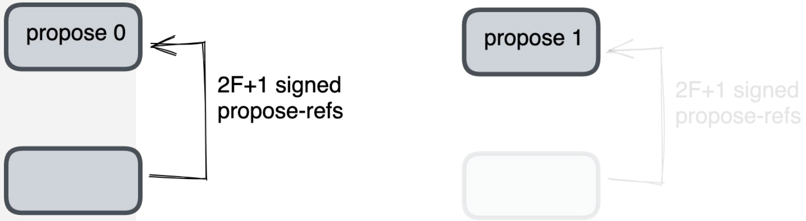
Practical BFT SMR for Partial Synchrony

	LPS 1982	DLS 1988	PBFT 1999	Casper 2017	HotStuff 2019
Safe against $f < n/3$ byz faults	👍	👍	👍	👍	👍
Safe against asynchrony	👎	👍	👍	?	👍
Number of messages to consensus decision	poly	poly	quadratic*	quadratic*	linear
Number of messages to rotate leader	poly	poly	quadratic	quadratic*	linear
Network speed	N/A	👎	👍	👎	👍

*Can be linear with threshold-cryptography

HotStuff [YM+ 2019]

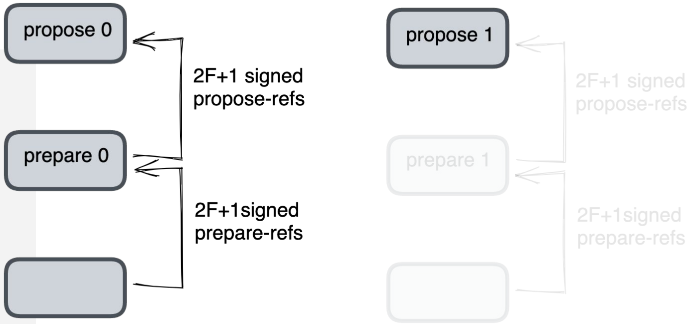
Recall, one round prevents equivocation



HotStuff [YM+ 2019]

Recall, one round prevents equivocation

Two rounds guarantee there is at most one prepare per leader view

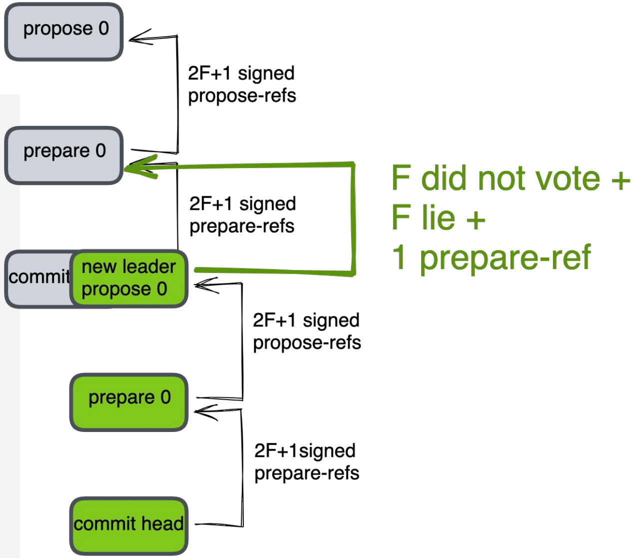


HotStuff [YM+ 2019]

One round prevents equivocation

Two rounds guarantee there is at most one prepare per leader view

> If there was a commit, even a single validator can tell a new leader a prepare which might have committed and is safe to propose

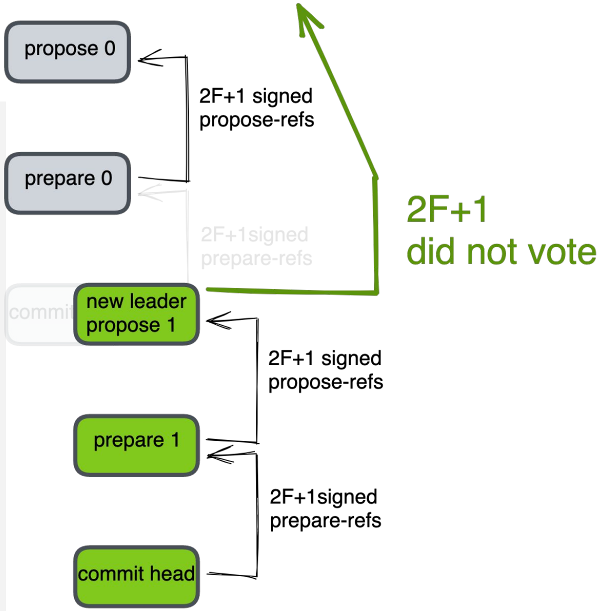


HotStuff [YM+ 2019]

One round prevents equivocation

Two rounds guarantee there is at most one prepare per leader view

> If there was a commit, even a single validator can tell a new leader a prepare which might have committed and is safe to propose
>
> If there was **no** commit, a leader can **prove** by including 2F+1 attestations they did not vote prepare
>
> For liveness, even if a validator ref'ed a higher prepare, it must accept the leader's proposal because it carries a **proof**



HotStuff [YM+ 2019]

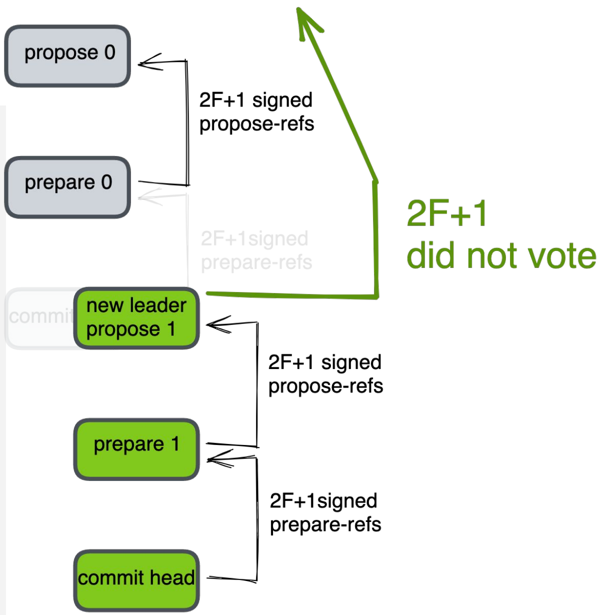
One round prevents equivocation

Two rounds guarantee there is at most one prepare per leader view

> If there was a commit, even a single validator can tell a new leader what might have committed and is safe to propose

What if the leader sends only one prepare? (subtle)

> If a validator has a **higher** prepare, it cannot trust the leader and abandon it
>
> Option-1 [Casper, VG 2016]: leader must wait Δ (maximal network delay)
>
> Option-2: Add a round



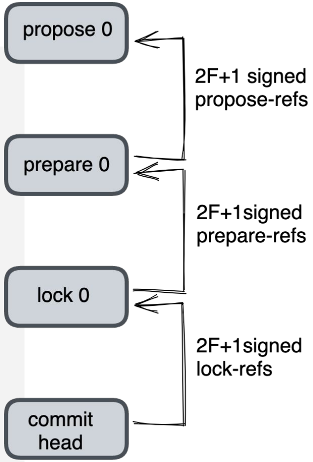
HotStuff [YM+ 2019]

One round prevents equivocation

Two rounds guarantee there is at most one prepare per leader view

> If there was a ~~commit~~ lock, even a single validator can tell a new leader a prepare which might have ~~committed~~ locked and is safe to propose

Three rounds



HotStuff [YM+ 2019]

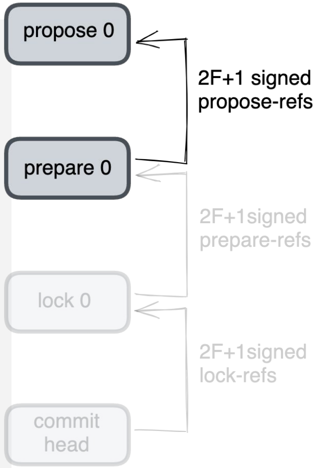
One round prevents equivocation

Two rounds guarantee there is at most one prepare per leader view

> If there was a commit, even a single validator can tell a new leader what might have committed and is safe to propose

Three rounds

> If a validator has a **higher** prepare, it **can** trust the leader and abandon it



HotStuff [YM+ 2019]

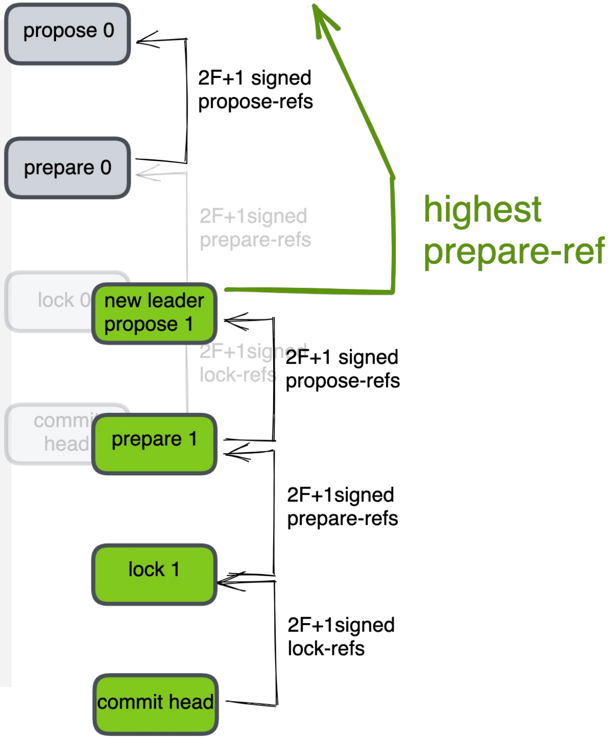
One round prevents equivocation

Two rounds guarantee there is at most one prepare per leader view

> If there was a commit, even a single validator can tell a new leader what might have committed and is safe to propose

Three rounds

> If a validator has a **higher** prepare, it **can** trust the leader and abandon it



HotStuff [YM+ 2019]

One round prevents equivocation

Two rounds guarantee there is at most one prepare per leader view

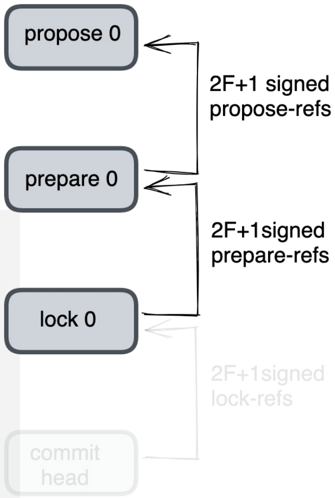
If there was a commit, even a single validator can tell a new leader what might have committed and is safe to propose

Three rounds

It works!

If a validator has a **higher** prepare, it **can** trust the leader and abandon it

If a validator has a higher **lock**, an honest leader cannot hide the highest prepare



HotStuff [YM+ 2019]

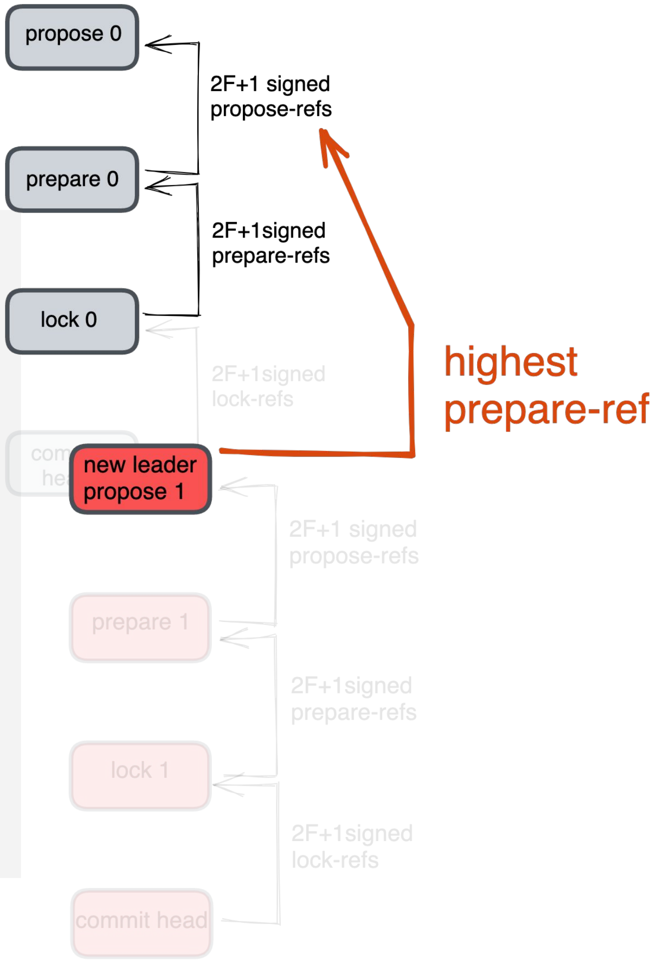
One round prevents equivocation

Two rounds guarantee there is at most one prepare per leader view

> If there was a commit, even a single validator can tell a new leader what might have committed and is safe to propose

Three rounds

> It works!
>
> If a validator has a **higher** prepare, it **can** trust the leader and abandon it
>
> If a validator has a higher **lock**, an honest leader cannot hide the highest prepare



An evolution of BFT consensus protocols

	PBFT 1999	Casper 2017	HotStuff 2019
Safe against *f* < *n/3* byz faults	👍	👍	👍
Safe against asynchrony	👍	?	👍
Number of messages to consensus decision	quadratic*	quadratic*	linear
Number of messages to rotate leader	quadratic	quadratic*	linear
Network speed	👍	👎	👍

*Can be linear with threshold-cryptography

An evolution of BFT consensus protocols

	PBFT 1999	Casper 2017	HotStuff 2019
Safe against $f < n/3$ byz faults	👍	👍	👍
Safe against asynchrony	👍	?	👍
Number of messages to consensus decision	quadratic*	quadratic*	linear
Number of messages to rotate leader	quadratic	quadratic*	linear
Network speed	👍	👎	👍
Rounds to commit	2	2	3

*Can be linear with threshold-cryptography

An evolution of BFT consensus protocols

	PBFT 1999	Casper 2017	HotStuff 2019	DiemBFT 2021
Safe against *f* < *n/3* byz faults	👍	👍	👍	👍
Safe against asynchrony	👍	?	👍	👍
Number of messages to consensus decision	quadratic*	quadratic*	linear	linear
Number of messages to rotate leader	quadratic	quadratic*	linear	linear + ε
Network speed	👍	👎	👍	👍
Rounds to commit	2	2	3	2

*Can be linear with threshold-cryptography