Hyperledger Fabric

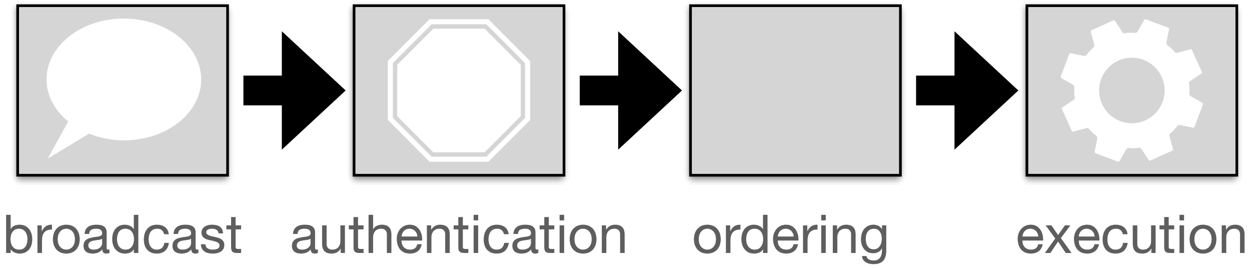
Execute-Order pipeline

Leander Jehl

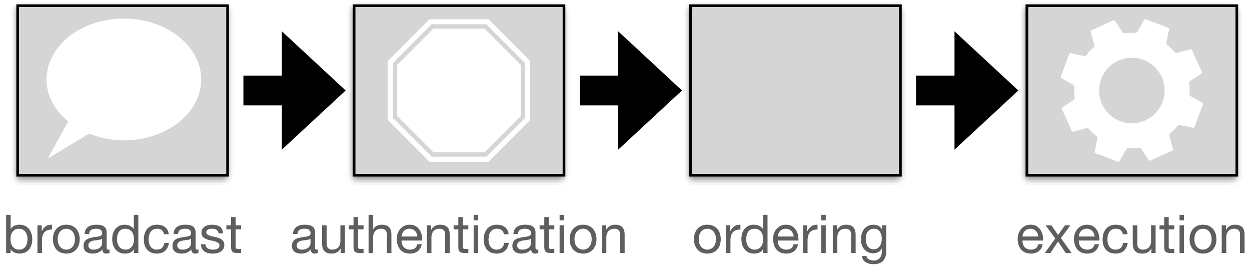
Hyperledger fabric

- a toolbox to run your own, permissioned blockchain
- Permissioned:
    - Several well known participants are responsible to run the blockchain
    - Distrust each other.

Transaction processing pipeline

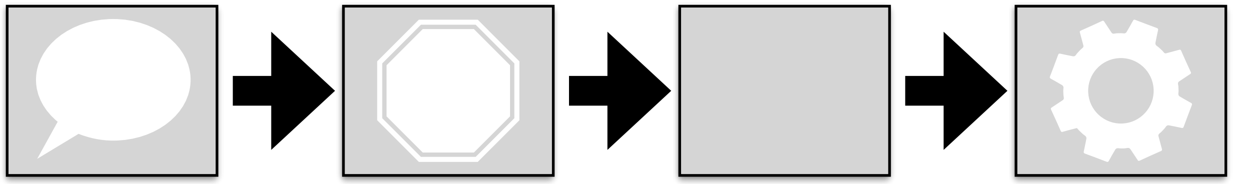


Transaction processing pipeline



- broadcast: send out transaction requires network resources
- validation: requires state
- ordering: requires coordination
- execution: requires state, must be deterministic.

Transaction processing pipeline

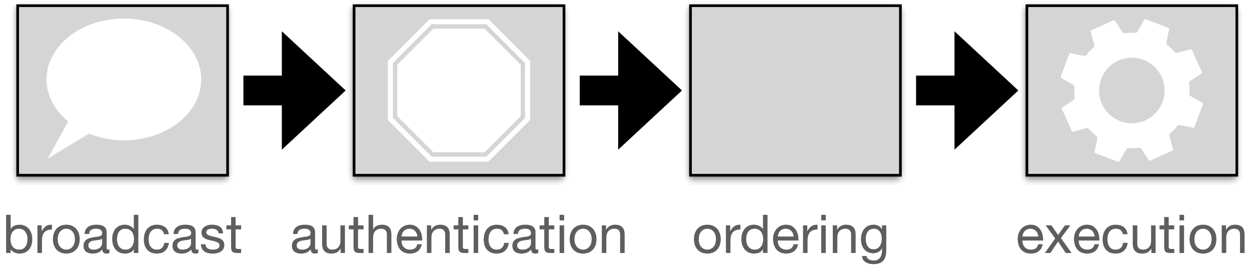


broadcast authentication ordering execution

- broadcast: send out transaction requires network resources
- validation: access rights
- ordering: requires coordination
- execution: requires erministic.

What is the bottleneck?

Transaction processing pipeline



- For complex workloads, and small scale BFT systems, execution is the bottleneck.
  - Single threaded execution to be deterministic
  - Can be complex workloads
  - Execution has privacy concerns (need access to data)

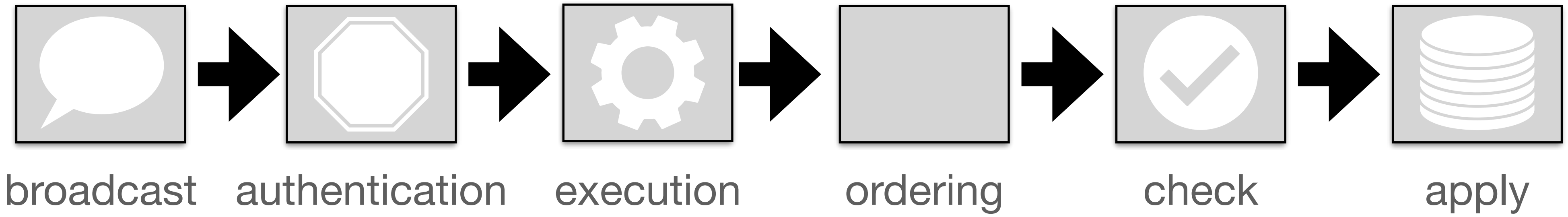
Transaction execution

execution

Two approaches exist for crash fault tolerant systems:

- **Deterministic processing**: Each replica can process transaction and arrive at the same result.
- **Applying state change**: One replica executes transaction. Records state change Δ. All replicas apply Δ.

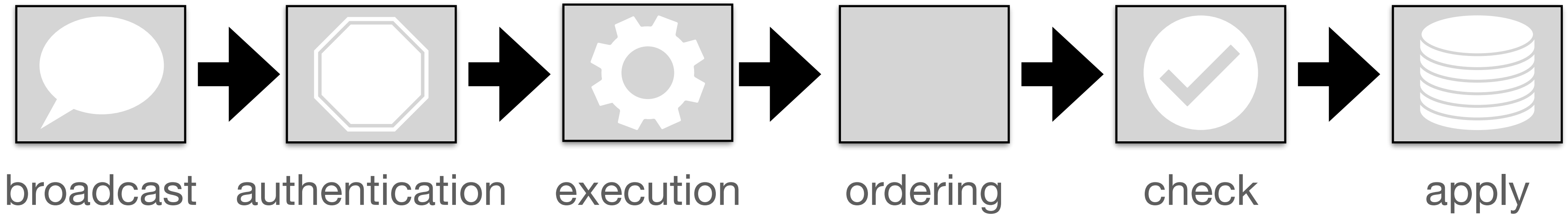
Transaction processing in Hyperledger fabric



- Execution happens before ordering.
- Execution policies, i.e. require *n* nodes to get the same result.
- Changes are submitted to ordering with signature from *n* nodes.
- During check, possibly inconsistent transactions are removed (aborted).

Transaction processing in Hyperledger fabric

State



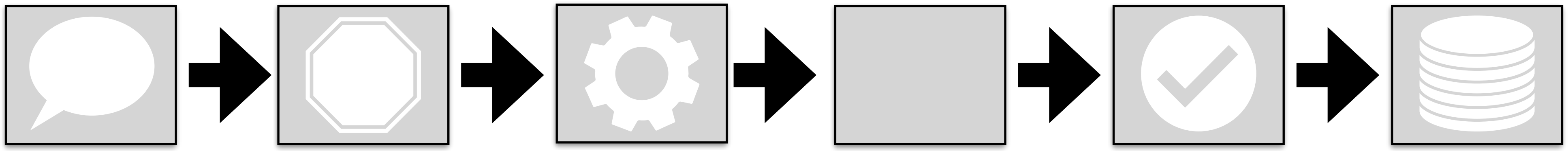
- State is organized as (key, value) pairs.

- Execution result records, new values for certain keys and which keys have been read.

- Based on read and write keys, *check* can remove inconsistent transactions

Transaction processing in Hyperledger fabric

State



- State is organized as (key, value) pairs.
- Execution result records, new values for certain keys and which keys have been read.
- Based on read and write keys, *check* can remove inconsistent transa

For all stages but execution. values can be

A BFT protocol

Simplified HotStuff

Leander Jehl

PoW or Certificates

Idea

PoW:

- In PoW the **difficulty decides the rate** at which the blocks are created, (throughput)
- High difficulty -> few forks
- Need to give time for a block to be propagated!
- But time to next block varies a lot!

PoW or Certificates

Idea

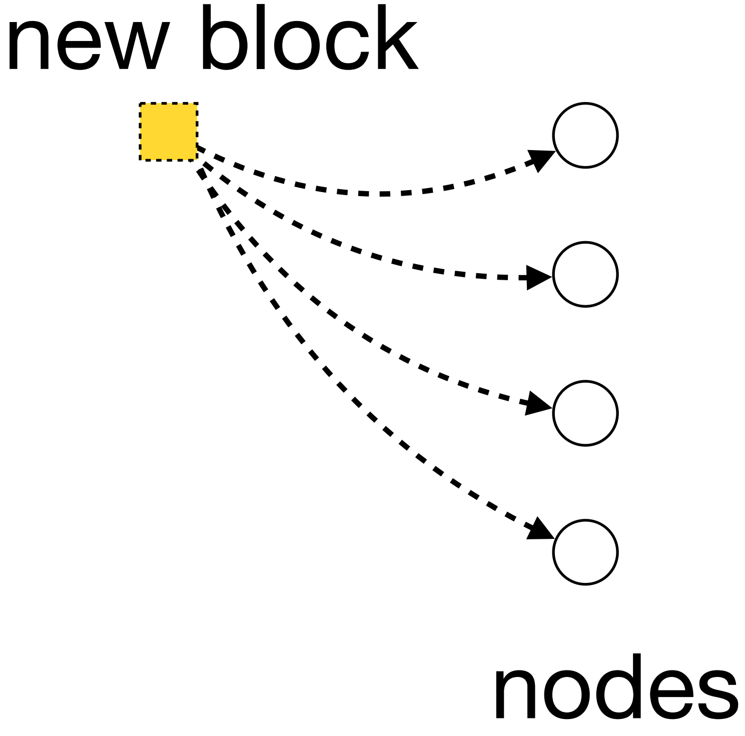
Goal:

- Throughput and confirmation only limited by time to propagate block,
  no extra waiting.

BFT protocol

Certificate vs. PoW

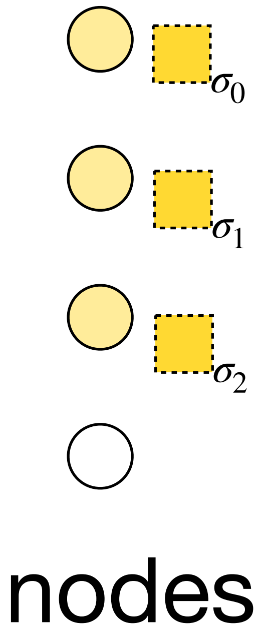
Idea: Send new block to nodes for validation and signature. Then collect certificate.



BFT protocol

Certificate vs. PoW

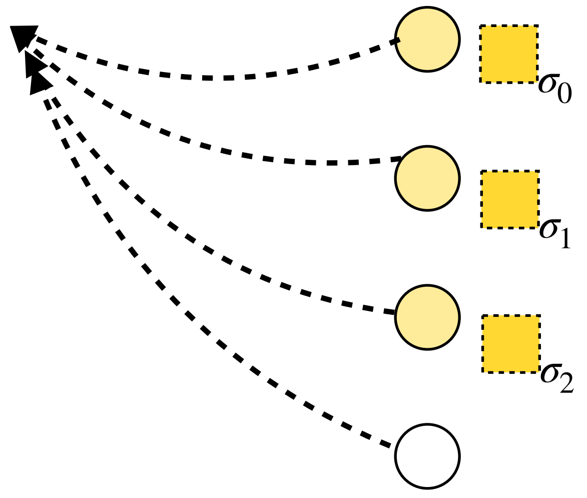
Idea: Send new block to nodes for validation and signature. Then collect certificate.



BFT protocol

Certificate vs. PoW

Idea: Send new block to nodes for validation and signature. Then collect certificate.

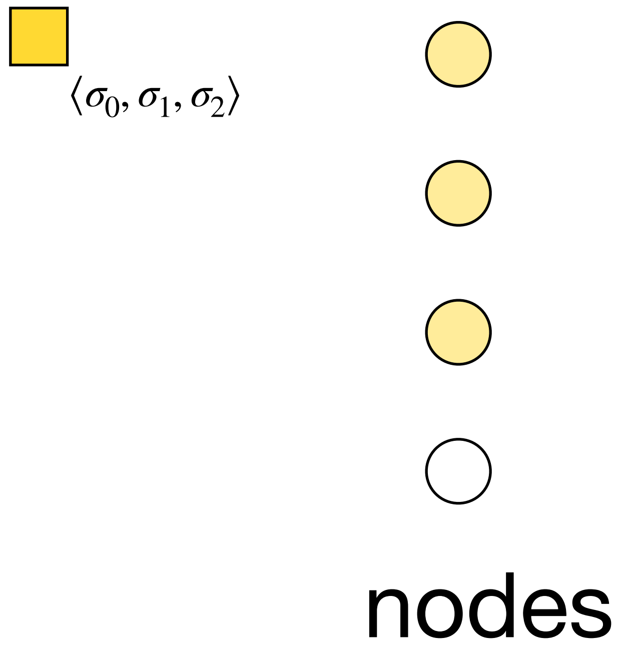


nodes

BFT protocol

Certificate vs. PoW

Idea: Send new block to nodes for validation and signature. Then collect certificate.



BFT protocol
Model

Model:

- We assume a permissioned system with $N = 3f + 1$ nodes.
- Nodes have unique ids and unique, known cryptographic keys.
- At most f of the nodes are byzantine faulty.

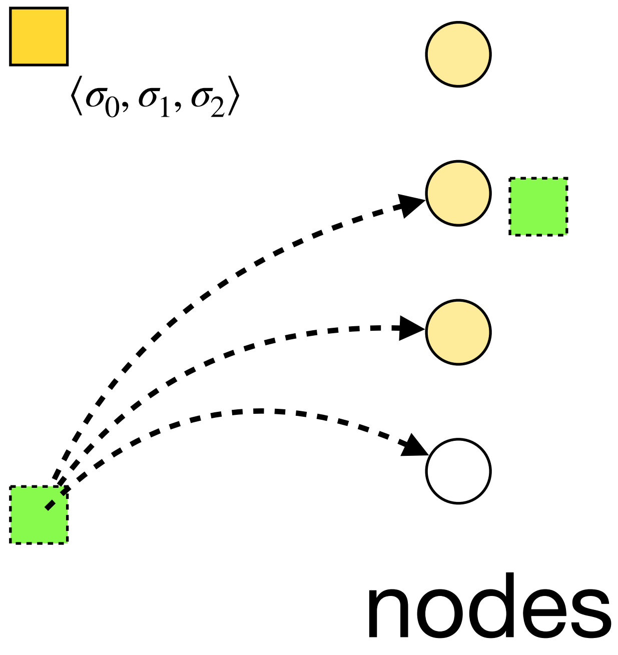
Certificate:

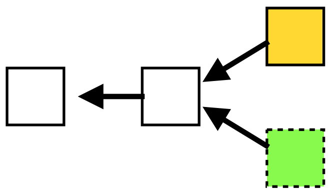
- A block has a certificate, if it contains signatures of $2f + 1$ nodes.

BFT protocol

Certificate vs. PoW

Idea: Send new block to nodes for validation and signature. Then collect certificate.

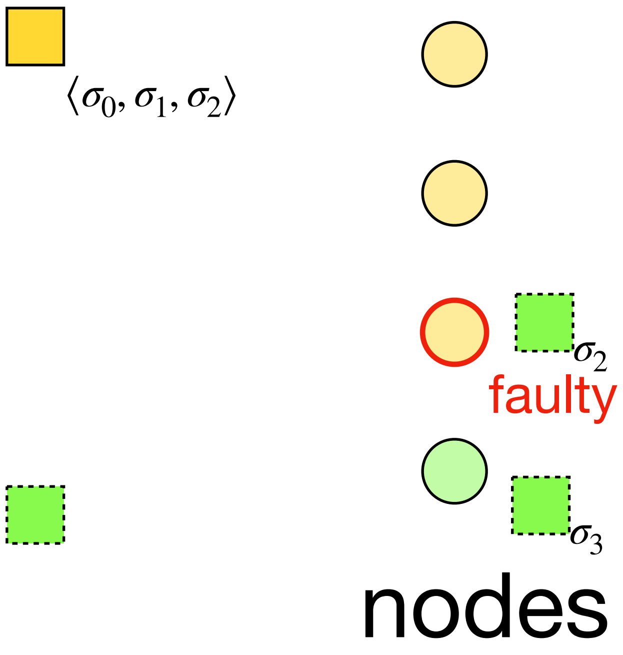


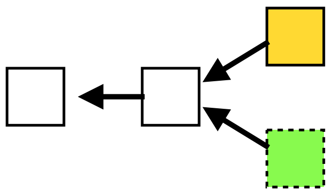


BFT protocol

Certificate vs. PoW

Idea: Send new block to nodes for validation and signature. Then collect certificate.

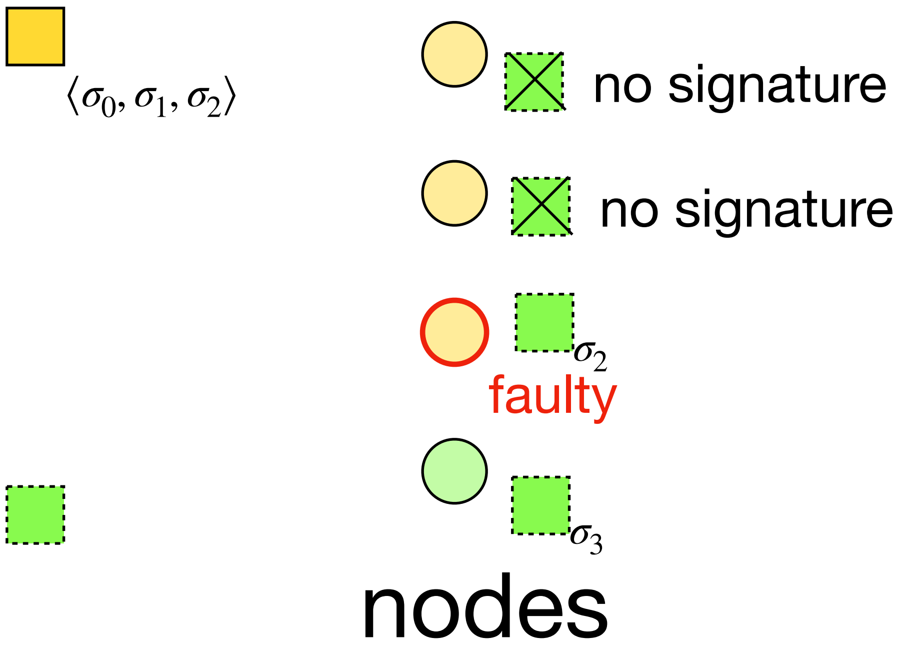


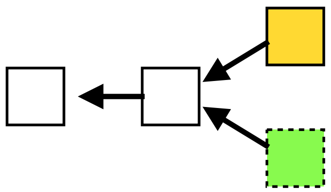


BFT protocol

Certificate vs. PoW

Idea: Send new block to nodes for validation and signature.
Then collect certificate.

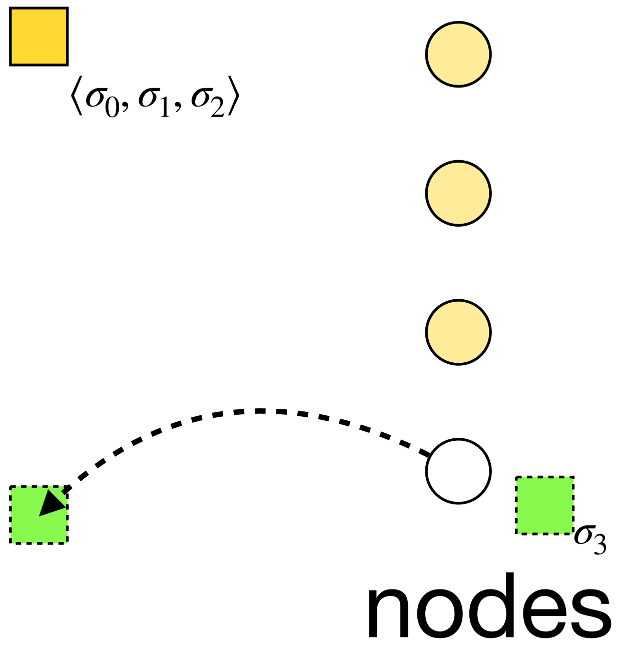


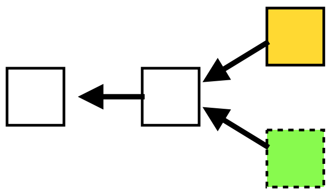


BFT protocol

Certificate vs. PoW

Idea: Send new block to nodes for validation and signature. Then collect certificate.





BFT protocol

Certificate vs. PoW

Idea: Send new block to nodes for validation and signature. Then collect certificate.

Correct nodes sign only one block at given depth.
Obs: Faulty nodes may sign multiple blocks!

BFT protocol

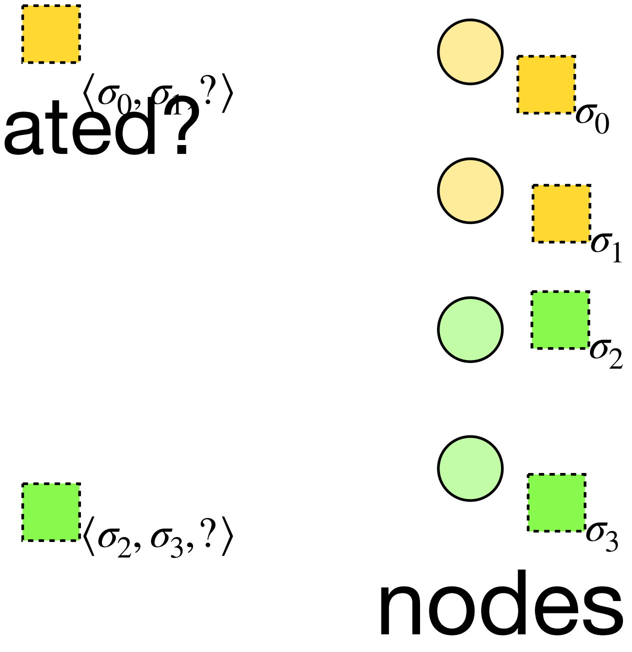
Certificate vs. PoW problem

Idea: Send new block to nodes for validation and signature.
Correct nodes sign only one block at given depth.
Then collect certificate.

Problem: How to ensure that a certificate is created?

- Nodes may sign different blocks
- No block gets a certificate

- **Solution:**



$\langle \sigma_0, \sigma_1, ? \rangle$
σ_0
σ_1
σ_2
σ_3
$\langle \sigma_2, \sigma_3, ? \rangle$
nodes

BFT protocol

Certificate vs. PoW problem

Idea: Send new block to nodes for validation and signature.
Correct nodes sign only one block at given depth.
Then collect certificate.

Problem: How to ensure that a certificate is created?

  - Nodes may sign different blocks
  - No block gets a certificate

- **Solution:** Leader

BFT protocol

Certificate vs. PoW problem

Idea: Send new block to nodes for validation and signature.
Correct nodes sign only one block at given depth.
Then collect certificate.

Problem: How to know that a certificate was created?

- A certificate may be collected by a single node
- The node with the certificate may fail and come back later

- **Solution:**

BFT protocol

Certificate vs. PoW problem

Idea: Send new block to nodes for validation and signature.
Correct nodes sign only one block at given depth.
Then collect certificate.

Problem: How to know that a certificate was created?

- A certificate may be collected by a single node
- The node with the certificate may fail and come back later

- **Solution:** Require multiple certificates

BFT protocol

Simple HotStuff (2 chain)

Preliminary:

- Every block includes a parent link (*previous block).*
  => Blocks form a **tree.**
  Every block includes a **round number/view number**

- Every block must include a **certificate** for its parent

- A blocks' **round** must be larger than that of its parent

BFT protocol

Simple HotStuff (2 chain)

Rules

- *Every block must contain certificate for parent*
- *Every block must have round > round of parent*

- **Rule 1:** After signing a block at round $max = r$, a node may only sign at round $r' > max$.
- **Rule 2 preliminary:** After signing a block with parent p and only sign blocks in subtree starting at p

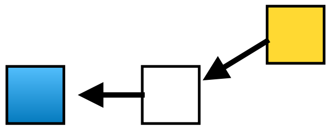
Keep maximum value for max and $lock$ in local variables.

Simple HotStuff

Example

Example

- Nodes n_0, n_1, and n_2 sign block 🟨



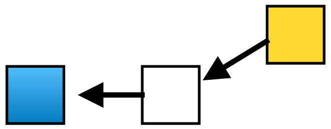
Node	*lock*
n_0	🟦
n_1	faulty
n_2	🟦
n_3	🟦

Simple HotStuff

Example

Example

- Nodes n_0, n_1, and n_2 sign block 🟨
- They set $lock$ to □



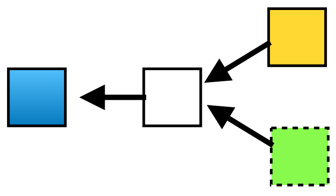
Node	$lock$
n_0	□
n_1	faulty
n_2	□
n_3	🟦

Simple HotStuff

Example

Example

- Nodes n_0, n_1, and n_2 sign block 🟨
- They set $lock$ to □
- n_3 signs 🟩

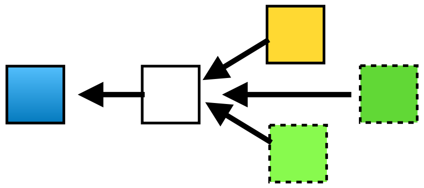


Node	$lock$
n_0	□
n_1	faulty
n_2	□
n_3	□

Simple HotStuff
Example

Example

- Nodes n_0, n_1, and n_2 sign block 🟨
- They set $lock$ to □
- n_3 signs 🟩
- n_3 creates 🟩



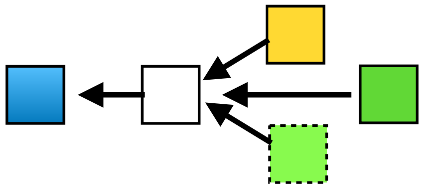
Node	$lock$
n_0	□
n_1	faulty
n_2	□
n_3	□

Simple HotStuff

Example

Example

- Nodes n_0, n_1, and n_2 sign block 🟨
- They set $lock$ to □
- n_3 signs 🟩
- n_3 creates 🟩
- n_3, n_1, and n_2 sign block 🟩



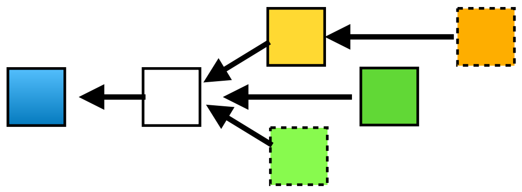
Node	$lock$
n_0	□
n_1	faulty
n_2	□
n_3	□

Simple HotStuff

Example

Example

- n_0 sign block 🟧



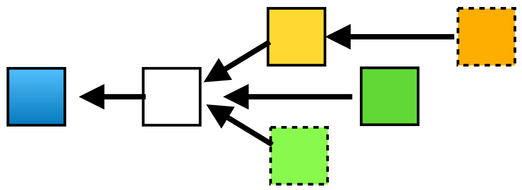
Node	$lock$
n_0	□
n_1	faulty
n_2	□
n_3	□

Simple HotStuff

Example

Example

- n_0 sign block 🟧



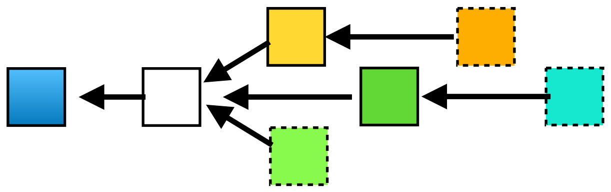
Node	$lock$
n_0	🟨
n_1	faulty
n_2	□
n_3	□

Simple HotStuff

Example

Example

- n_0 sign block 🟧
- n_3 signs 🟦



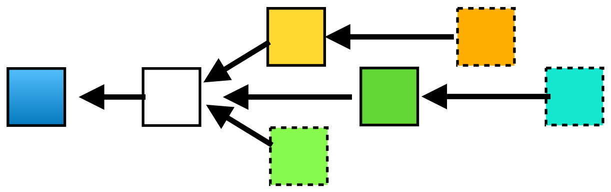
Node	$lock$
n_0	🟨
n_1	faulty
n_2	□
n_3	□

Simple HotStuff

Example

Example

- n_0 sign block 🟧
- n_3 signs 🟦



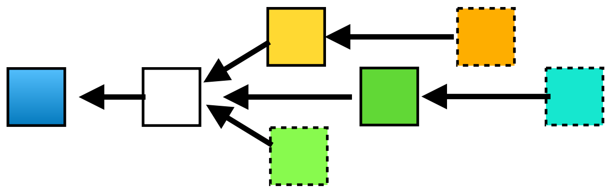
Node	$lock$
n_0	🟨
n_1	faulty
n_2	⬜
n_3	🟩

Simple HotStuff

Example

Example

- assume node n_0 is new leader.
- to get a certificate, n_0 must rely on faulty n_1



Node	$lock$
n_0	🟨
n_1	faulty
n_2	□
n_3	🟩

BFT protocol

Simple HotStuff (2 chain)

Rules

- *Every block must contain certificate for parent*
- *Every block must have round > round of parent*

- **Rule 1:** After signing a block at round $max = r$,
  a node may only sign at round $r' > max$.

- **Rule 2:** After signing a block with parent p and $p\,.\,round = lock$,
  only sign blocks with parents in round $pr \geq lock$

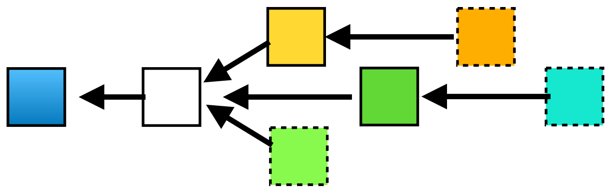
Keep maximum value for max and $lock$ in local variables.

Simple HotStuff

Example

Example

- assume node n_0 is new leader.
- to get a certificate, n_0 must
    - rely on faulty n_1
    - or extend 🟩



Node	$lock$
n_0	🟨
n_1	faulty
n_2	⬜
n_3	🟩

Solution:

n_0 waits for Δ time to get newest block

BFT protocol

Simple HotStuff

Def.:

a) A block with $round = r$ is **confirmed** if it has a child in $round = r + 1$, which has a certificate.

b) A block with $round = r$ is **confirmed** if it has a grandchild in $round = r + 2$

Theorem: *If a block is confirmed, only descendants of that block, can get a certificate.*

Proof: A majority of correct nodes have set their $lock$ to the confirmed node.

BFT protocol

Simple HotStuff - Leader

Idea 1: Every round has a designated leader.

Idea 2: Nodes wait for Δ time for a proposal in current round, before accepting at next round.

BFT protocol

Simple HotStuff - Leader

Idea 1: Every round has designated leader.

Idea 2: Nodes wait for Δ time for a proposal in current round, before accepting at next round.

How can a leader avoid the situation from the example?

Ask all nodes for most recent certificate.

Wait for Δ time to receive proposal from all correct nodes.

BFT protocol

Simple HotStuff (3 chain)

Rules

- *Every block must contain certificate for parent*
- *Every block must have round > round of parent*

- **Rule 1:** After signing a block at round $max = r$, a node may only sign at round $r' > max$.
- **Rule 2:** After signing a block with **grandparent** p and $p\,.\,round = lock$, only sign blocks with parents in round $pr \geq lock$

Keep maximum value for max and $lock$ in local variables.

BFT protocol

Simple HotStuff (3chain)

In this variant we need to wait longer for confirmation!

Def.:

a) A block with $round = r$ is **confirmed** if it has a **grandchild** in $round = r + 2$, which has a certificate.

b) A block with $round = r$ is **confirmed** if it has a **grand-grandchild** in $round = r + 3$

Theorem: *If a block is confirmed, only descendants of that block, can get a certificate.*

Proof: A majority of correct nodes have set their $lock$ to the confirmed node.

BFT protocol

Simple HotStuff - Leader

Idea 1: Every round has designated leader.

Idea 2: Nodes wait for Δ time for a proposal in current round, before accepting at next round.

How can a leader avoid the situation from the example?

Ask all nodes for most recent certificate.

Wait for $2f + 1$ replies

No leader needs not wait for Δ time!