Consensus in Partial Synchrony

PBFT, Tendermint, Hotstuff

Zeta Avarikioti
georgia.avarikioti@tuwien.ac.at

November 5th, 2025

TU WIEN

S&P Security & Privacy

Overview

- SMR in partial synchrony
- Practical Byzantine Fault Tolerance (PBFT)
- More partially-synchronous BFT protocols
  - Tendermint
  - Hotstuff
- Trade-offs

Consensus (B.B.) Universe

Byzantine

$3f \geq n$ impossible without PKI [FLM]

$2f < n$ possible with PKI
[e.g., Dolev-Strong,1982]

$3f < n$ is possible
common coin, O(1) rounds [Feld83]
shared coin, exp. rounds [Bracha87]
shared coin, poly(n) …35year later!

$2f \geq n$ impossible with PKI

$2f < n$ possible, randomized

Synchrony

Asynchrony

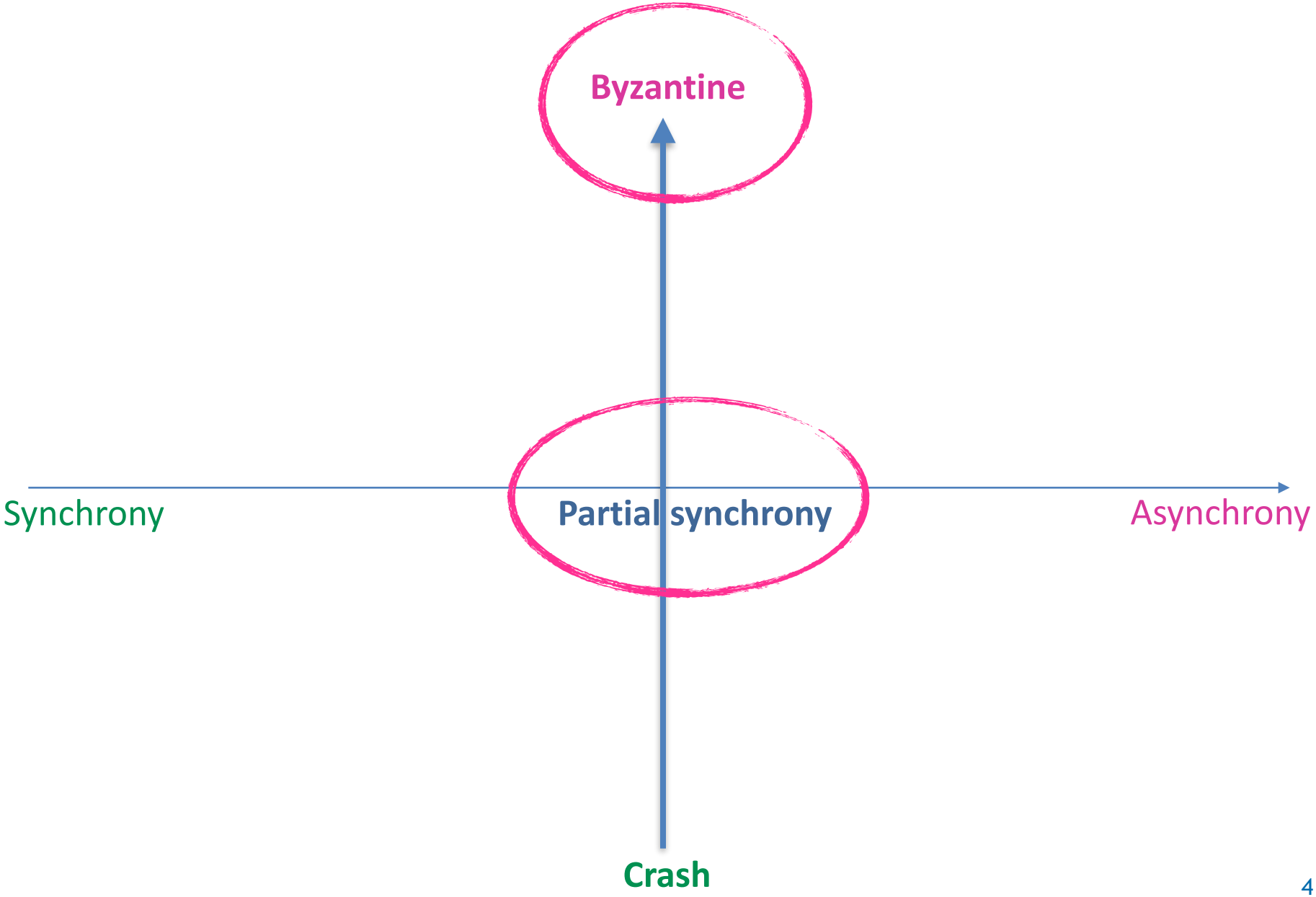
$2f \geq n$ impossible, randomized

Impossible in determinism
with a single crash

$f < n$ possible
(At least f+1 rounds)

Crash

Consensus (B.B.) Universe



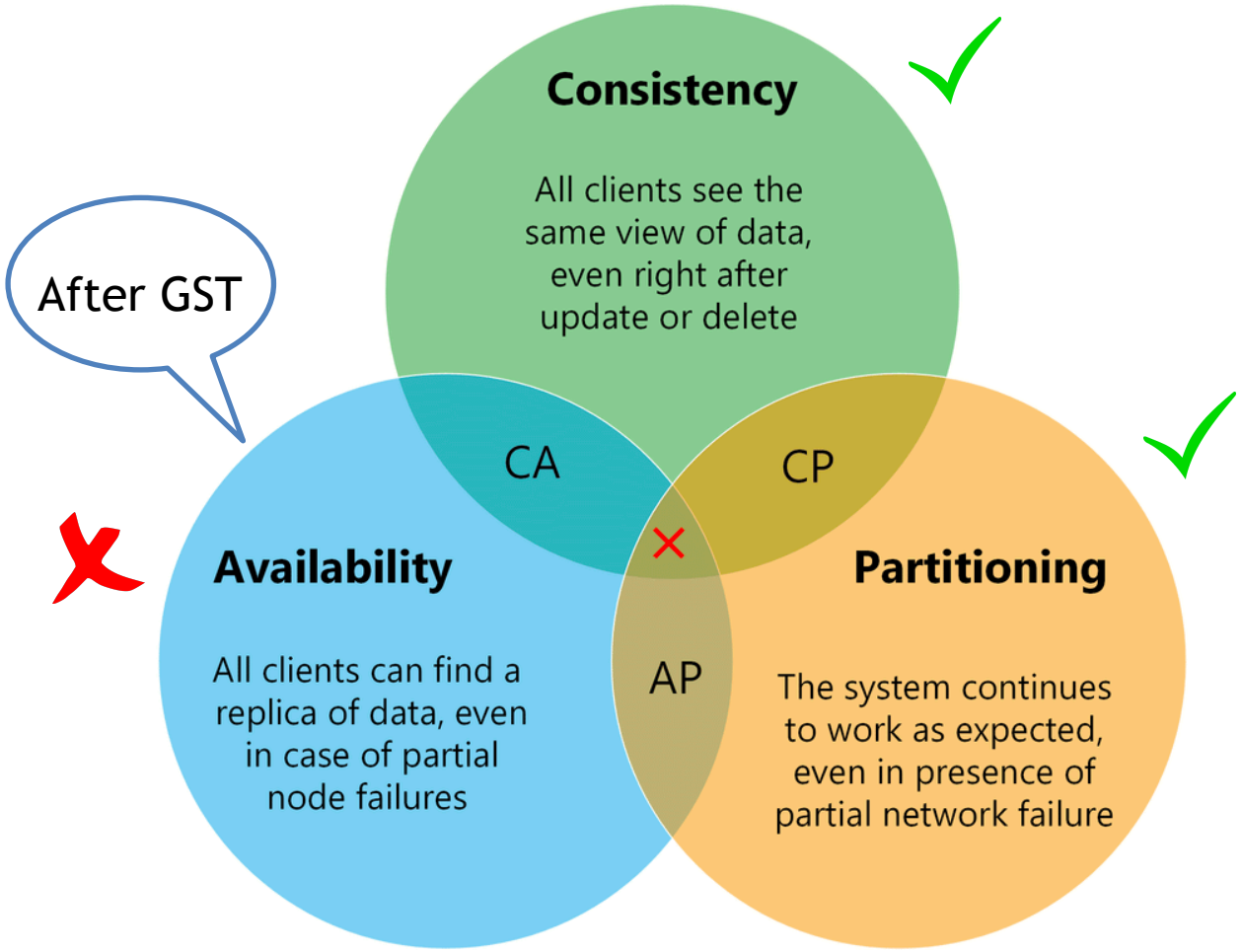
Consensus in Partial Synchrony

It is impossible to solve Consensus under partial synchrony against a Byzantine adversary if $f \geq n/3$ (DLS88 - Theorem 4.4).

- **Partial Synchrony**
  The network is always synchronous but the upper bound of the maximum delay is unknown.
  (Or) The network is asynchronous until an unknown Global Stabilization Time (GST) after which the networks is synchronous with known bounded delay.

- **Goals (for SMR)**
  Clients submit transactions. Every node maintains an append only data structure (i.e., an ordered sequence of transactions), called the ledger, that satisfies:
  - **Safety:** All nodes agree on the ledger (too strong!)
    (Some nodes may lack behind:) For any pair of nodes, either they share the same ledger or one is a prefix of the other.
  - **Liveness:** Honest nodes will eventually execute a transaction submitted by a client.
    Note the difference to blockchain liveness: Every transaction submitted will be eventually included in the ledger.

The CAP Theorem



PBFT Protocol

Assumptions

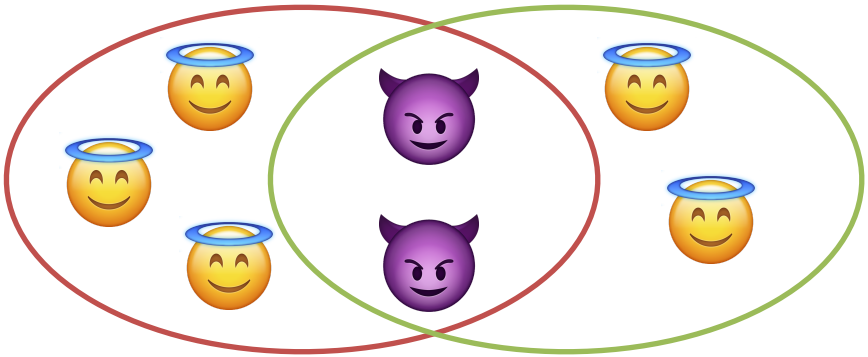
- PKI assumption aka Authenticated Agreement
- Pacemaker or (simplified Global Clock):
  All nodes have the same timeouts so they can agree on the current leader (after GST)
- The adversary may corrupt up to f<n/3 nodes

PBFT (in a nutshell)

- Leader-based protocol: the leader is elected round-robin and proposes a block (or value); it keeps going unless progress is not made
- Nodes (aka backups) vote on the proposed block
- Each node maintains its own ledger (in asynchrony, the “height” of the ledger may differ among nodes) & focuses on its current height
- For a fixed height, nodes keep proposing and voting until agreement

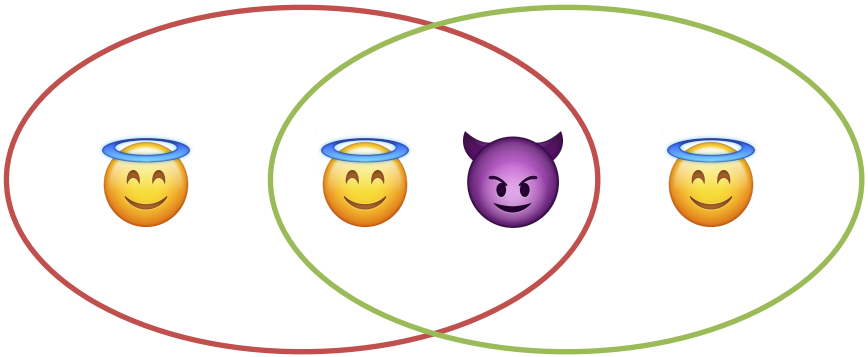
PBFT in a view

- A view corresponds to a leader
- A leader proposes a block, and gathers votes
- How many votes are enough for nodes to commit the block?
  - We know the adversary controls f out of n=3f+1
  - Is majority of votes enough? (e.g., f=2)



PBFT in a view

- **Quorum Certificates**
  The magic number of votes is 2f+1 (out of 3f+1=n)
  - If more, f (Byzantine) may crash - no liveness
  - If less, the adversary may partition the network - no safety



PBFT in a view

Leader

- Broadcast a new block B at latest (unconfirmed) height h

Nodes

If currently at height h,

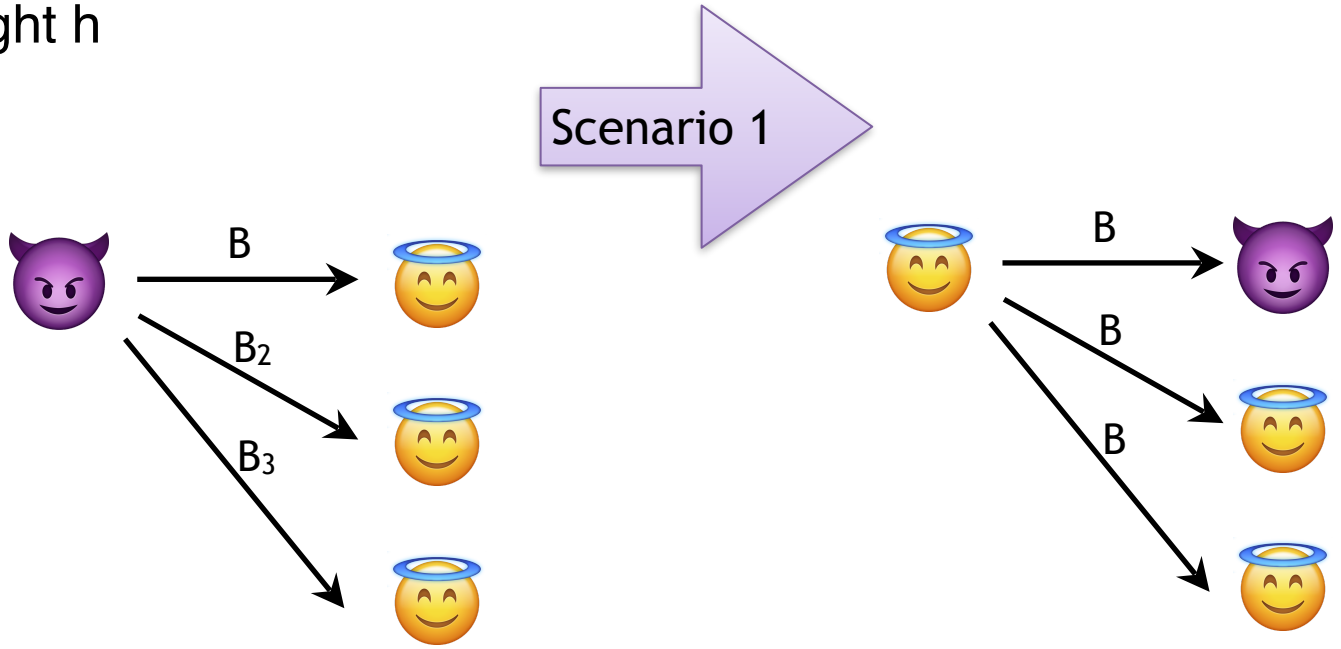
- Vote to find if there is a unique candidate block B and broadcast
- If B unique (QC), confirm block B

Else ignore block B

- What happens if there is no unique block? **View-change!**
  - Do nodes now vote only on the *same* block B, (or)
  - Are nodes allowed to *change* their vote to B'?

PBFT View Change

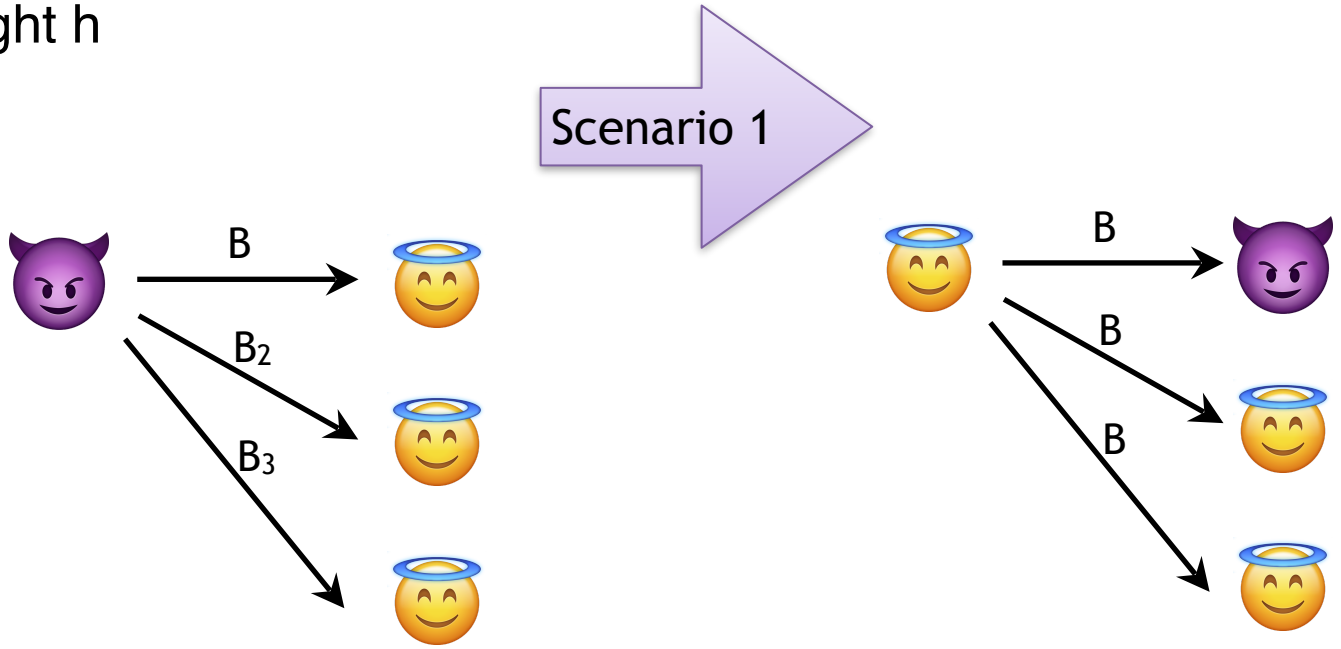
At height h



- All parties vote their block
- No QC, hence change of leader
- If parties cannot change their vote for height h even with the new leader…?

PBFT View Change

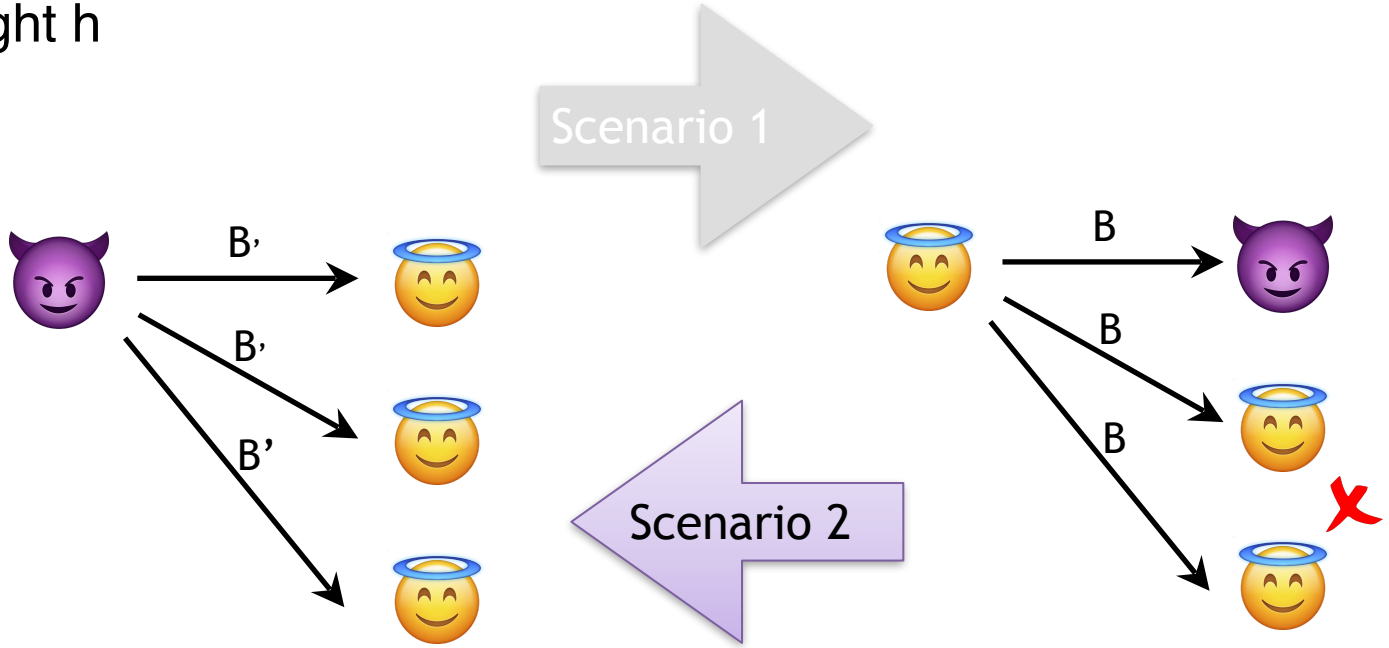
At height h



- All parties vote their block
- No QC, hence change of leader
- If parties cannot change their vote for height h even with the new leader
- Liveness violation!

PBFT View Change

At height h

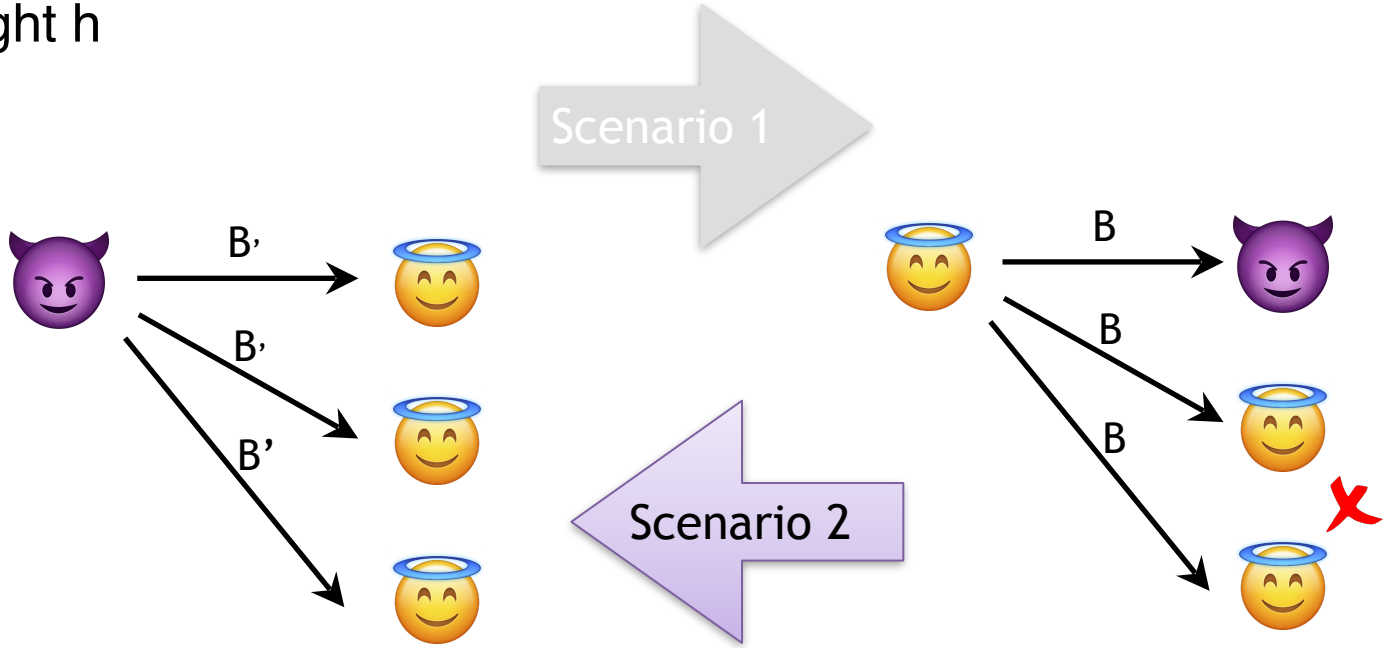


- All parties vote their block
- No QC, hence change of leader
- If parties cannot change their vote for height h even with the new leader
- Liveness violation!

- Leader receives 2 votes on B and commits
- Adversary delays network: signed votes to other honest parties do not arrive in time, and they change to next leader
- Next leader is Byzantine and proposes B'

PBFT View Change

At height h

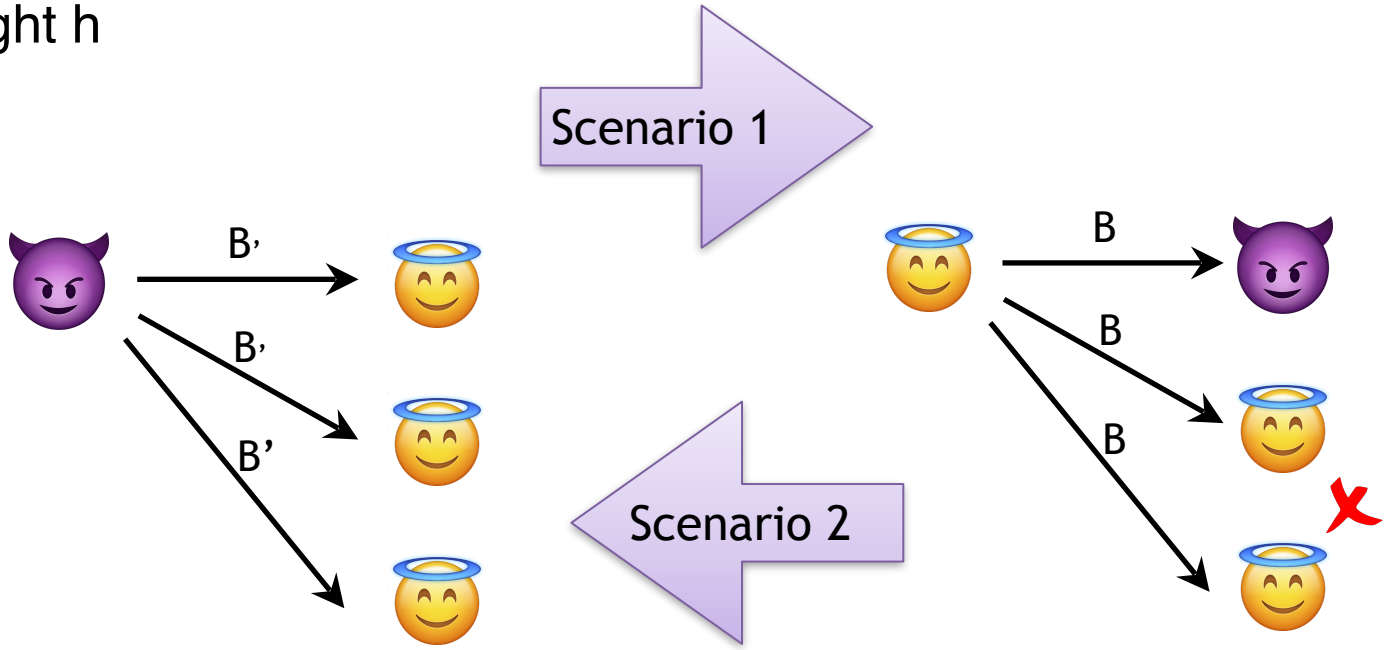


- All parties vote their block
- No QC, hence change of leader
- If parties cannot change their vote for height h even with the new leader
- Liveness violation!

- Leader receives 2 votes on B and commits
- Adversary delays network: signed votes to other honest parties do not arrive in time, and they change to next leader
- Next leader is Byzantine and proposes B'
- Two honest parties vote and commit B'
- Safety violation!

PBFT View Change

At height h



We must either forgo safety or liveness, so we cannot execute upon one QC vote!

2 Phases of Voting!

- First time voting (lock) ensures that no two nodes will execute different blocks at this height under the same leader

- Second time voting (commit) ensures that no two nodes will execute different blocks at this height even later with different leaders.

PBFT in a view

Leader

- Broadcast a new block B at latest (unconfirmed) height h

Nodes

If currently at height h,

- Vote once to find if there is a unique candidate block B and broadcast
- If B unique (QC), vote a second time to confirm block B and broadcast
- If B confirmed (QC), execute block B and reply to the client

Else ignore block B

PBFT View Change

a) Who is the next leader?
b) When do we change leader?
c) What does the leader learn?
d) How does the leader convince others to vote?

Nodes

- (1) If timeout expires, proceed to next leader and broadcast all blocks that are *unique*
- (4) Upon receiving L and V from the next leader, verify L's correctness; Return to normal operation (voting)

Leader

- (2) When activated as leader, accumulate all unique blocks to V and create proposed ledger L
- (3) Broadcast L and V; Return to normal operation

Do they know that we know that they know?

PBFT Normal Operation

Leader

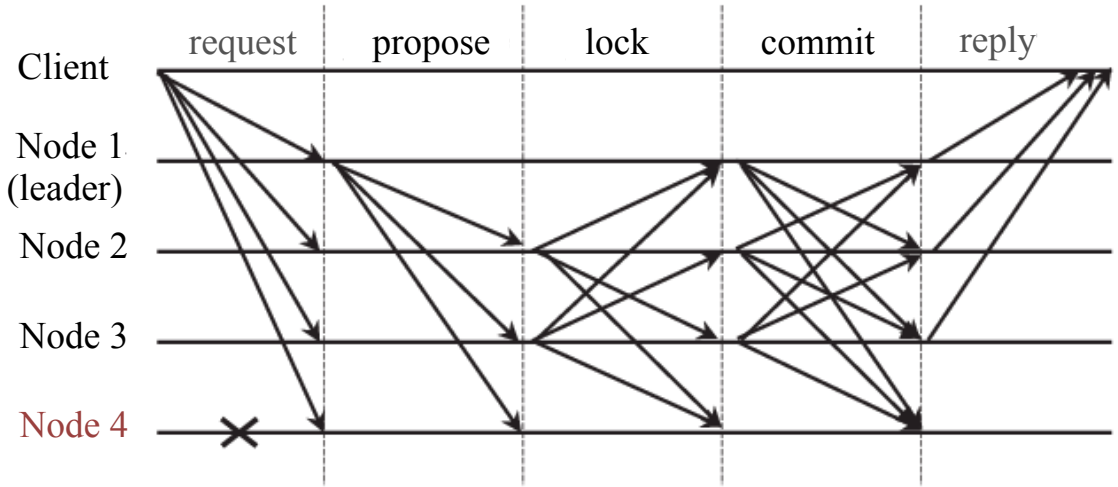
- The leader of view v broadcasts a proposal for block B at height h: <propose,B,h,v>

Nodes

If currently at height h,

- Vote on block B, unless it voted on another block B' at the same view, and broadcasts *<lock,B,h,v>*
- Upon receiving 2f+1 (lock) votes (QC_{lock}) on B at height h at view v, lock B and broadcast *<commit,B,h,v>*
- Upon receiving 2f+1 commits (QC_{commit}) on B at height h at view v, commit B and reply to the client

Else ignore block B



Quorum Certificates

If $QC_s(B,h,v) = QC_s(B',h,v)$, then $B=B'$
(s=lock or commit)

- Recall a QC consists of 2f+1 votes on the same block B, for the same height h, during the same view v (leader).

  Proof
  - At most f didn't vote for B
  - At most f didn't vote for B'
  - At least 3f+1 - (f+f) = f+1 voted for both B and B'
  - At most f are Byzantine
  - At least one honest voted for both B and B', hence B=B'

- When a node gets a QC_{commit} then it commits a block. In this case, we know that no other block could have been committed within that view as it is unique from QC_{lock} (safety within view).

- The QC_{commit} will also ensure that no future leader can cause a safety violation (across-view safety).

PBFT View Change

Nodes

- (1) If timeout expires:
  - Stop voting for view v and proceed to the next (leader) view v+1
  - Broadcast <*view-change,L_i,v+1*>, L_i being all your blocks that have at least QC_{lock}
- (4) Upon receiving L and V from the next leader, verify L is created correctly from V; Return to normal operation (voting)

Leader (of view v+1)

- (2) Upon receiving 2f+1 distinct <*view-change,L_i,v+1*> (=V), construct ledger L composed of one *propose* message per block height h:
  - if V contains at least one block with h, find the block B that corresponds to the latest leader (highest view) and add <*propose,B,h,v+1*>
  - Else, <*propose,null,h,v+1*>
- (3) Broadcast L and V; Return to normal operation

Checkpoints: Another round of voting on committed blocks (occasionally) to prune the ledger

Safety in View Change

- If there exists a node that committed to *B* at height *h* at view *v*, $QC_{commit}(B,h,v)$, then this node received *2f+1* $QC_{lock}(B,h,v)$
- At most f (slow) honest nodes have not signed $QC_{lock}(B,h,v)$
- A new leader in view *v'>v*, receives the unique blocks of *2f+1* nodes
  - At most *f* are the slow nodes that have not seen $QC_{lock}(B,h,v)$
  - There are *f* malicious nodes that will not send $QC_{lock}(B,h,v)$
  - There is at least *(2f+1)-f-f=1* honest node that has seen $QC_{lock}(B,h,v)$, and will send it to the new leader at view v'.
- An honest leader will see the $QC_{lock}(B,h,v)$, and propose block *B* for height *h*
- A malicious leader cannot make a correct proposal without proposing *B* for height *h*

Liveness (key ideas)

- Double the timeout every consecutive view-change
- Nodes can catch-up by changing to a higher view if f+1 other nodes are already ahead
- Byzantine nodes cannot cause a view change as backups (only f)
- Byzantine leaders can cause a view change but only up to f consecutive views

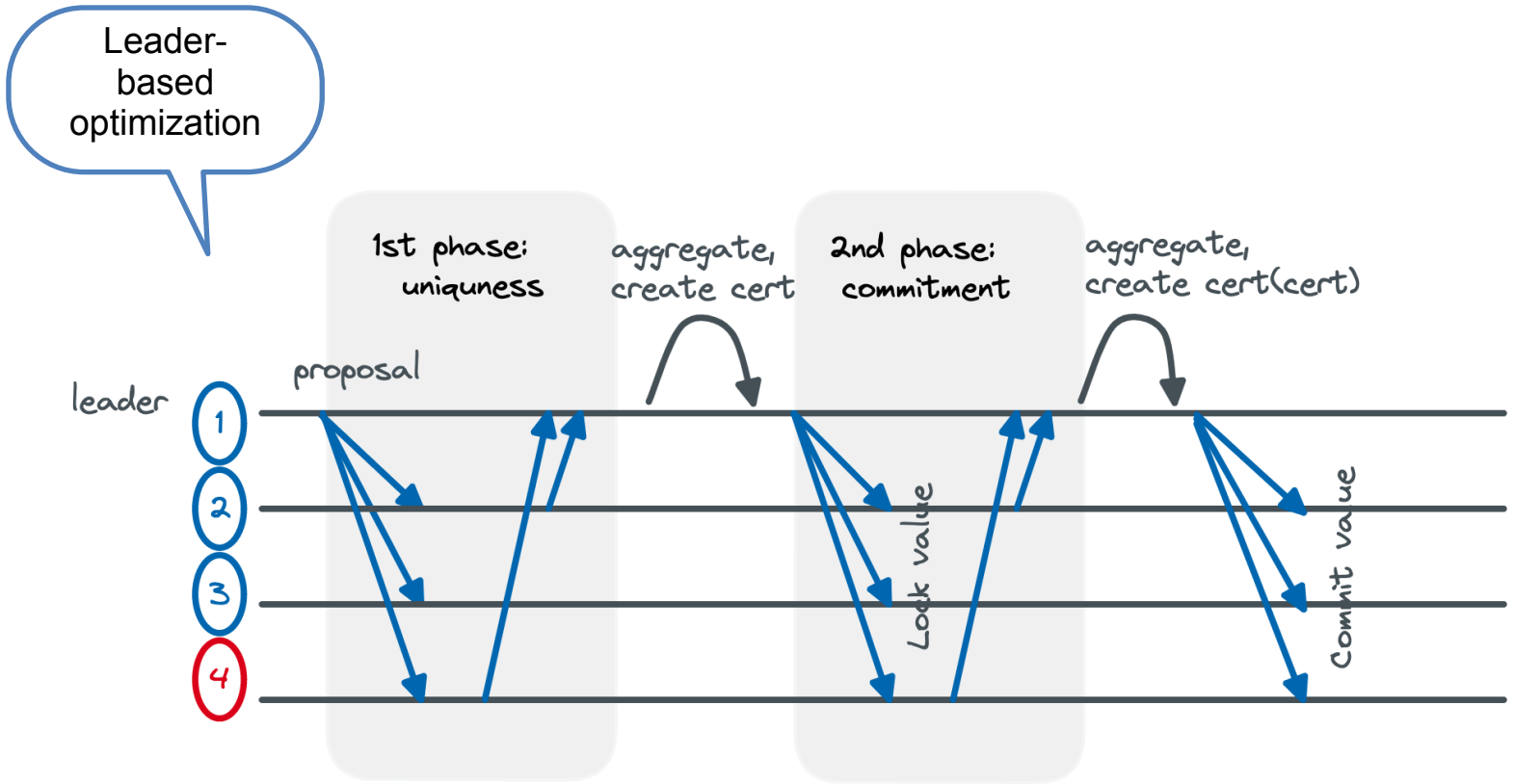
Other SMR protocols

in partial synchrony

Properties

- Safety
- Liveness
- Responsiveness: progress with the *actual* network delay
- Communication Complexity (aka bit complexity)
  - View-change
  - Worst-case
- Latency (phases of voting)

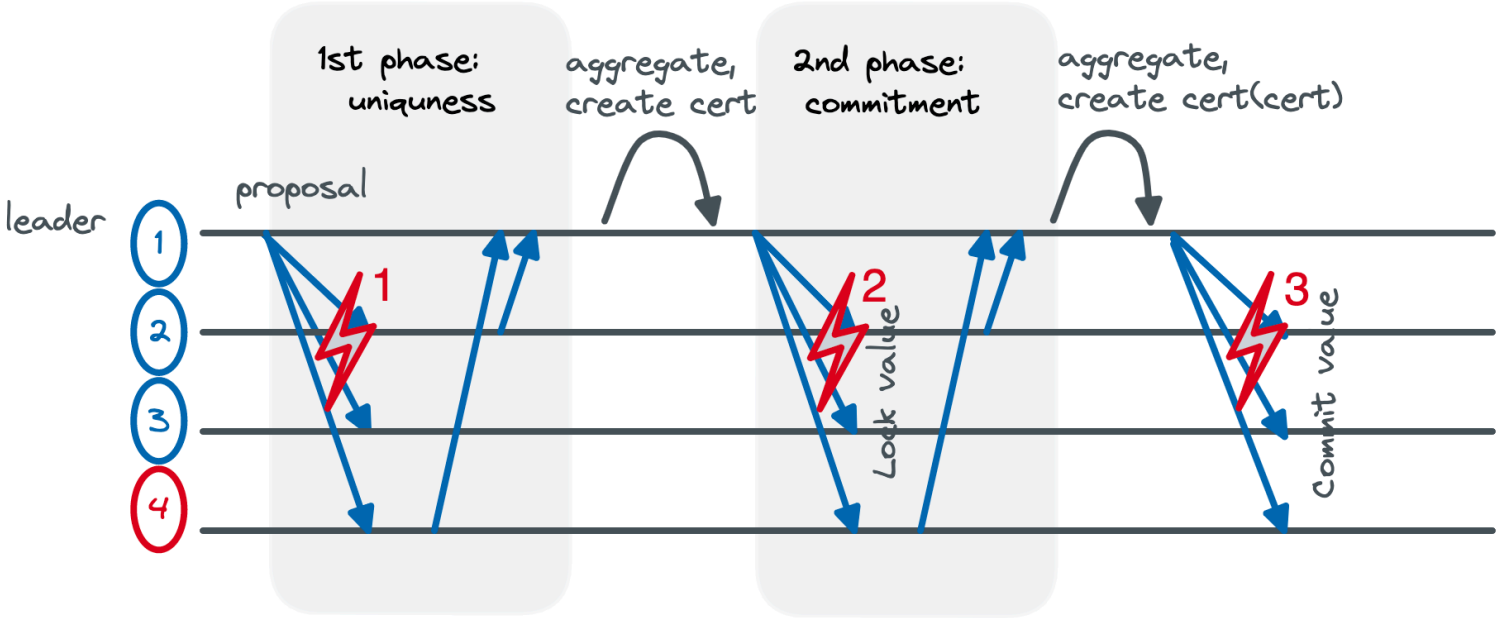
Lock-Commit Paradigm



Safety first!

- When a node commits B, at least 2f+1 nodes have locked B
- At least f+1 will safeguard B, i.e., they do not vote on a conflicting values unless they are shown a proof that it is safe to do so

Lock-Commit Paradigm



Three possible points of failure (either due to Byzantine leader or network failure):

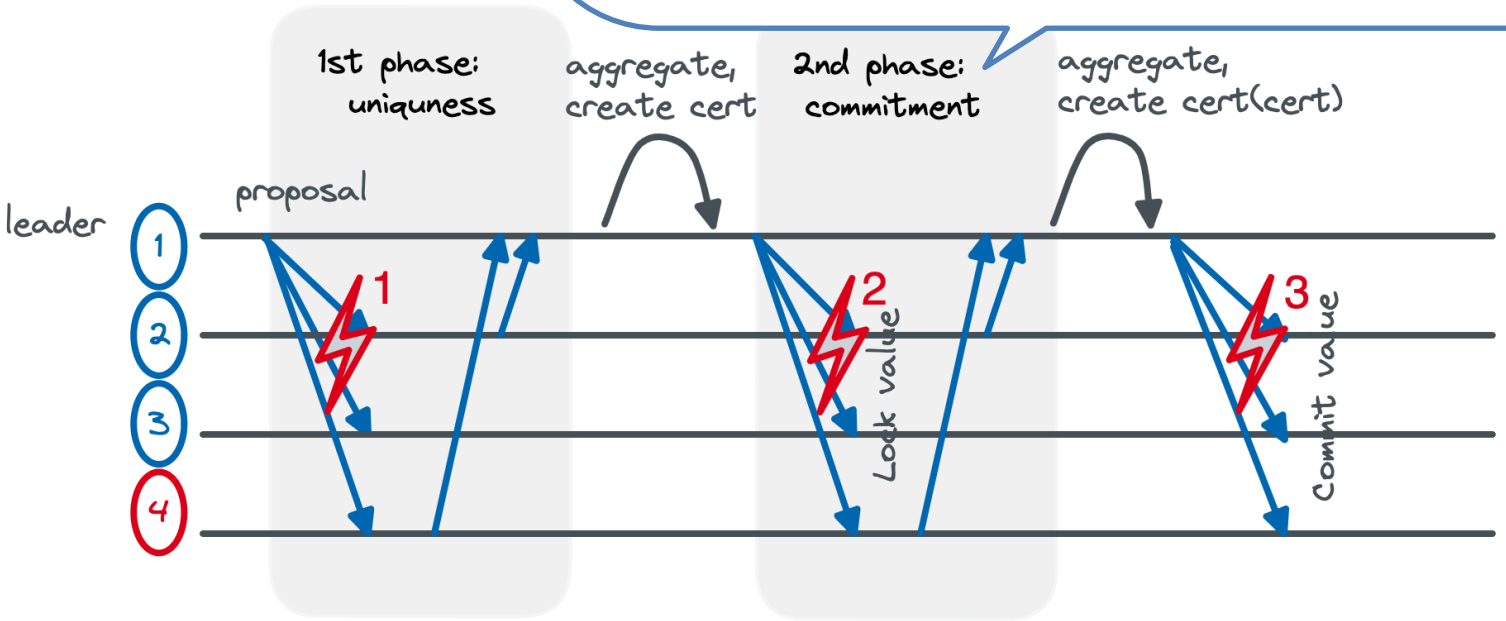
(1) No node has locked or committed (free to change votes)

(2) Some nodes locked, none committed

(3) At least 2f+1 locked, some but not all committed

Lock-Commit Paradigm

a) What does the leader learn about the status of the system?
b) How does the leader convince other nodes about the status of the system (and thus vote for its proposal)?



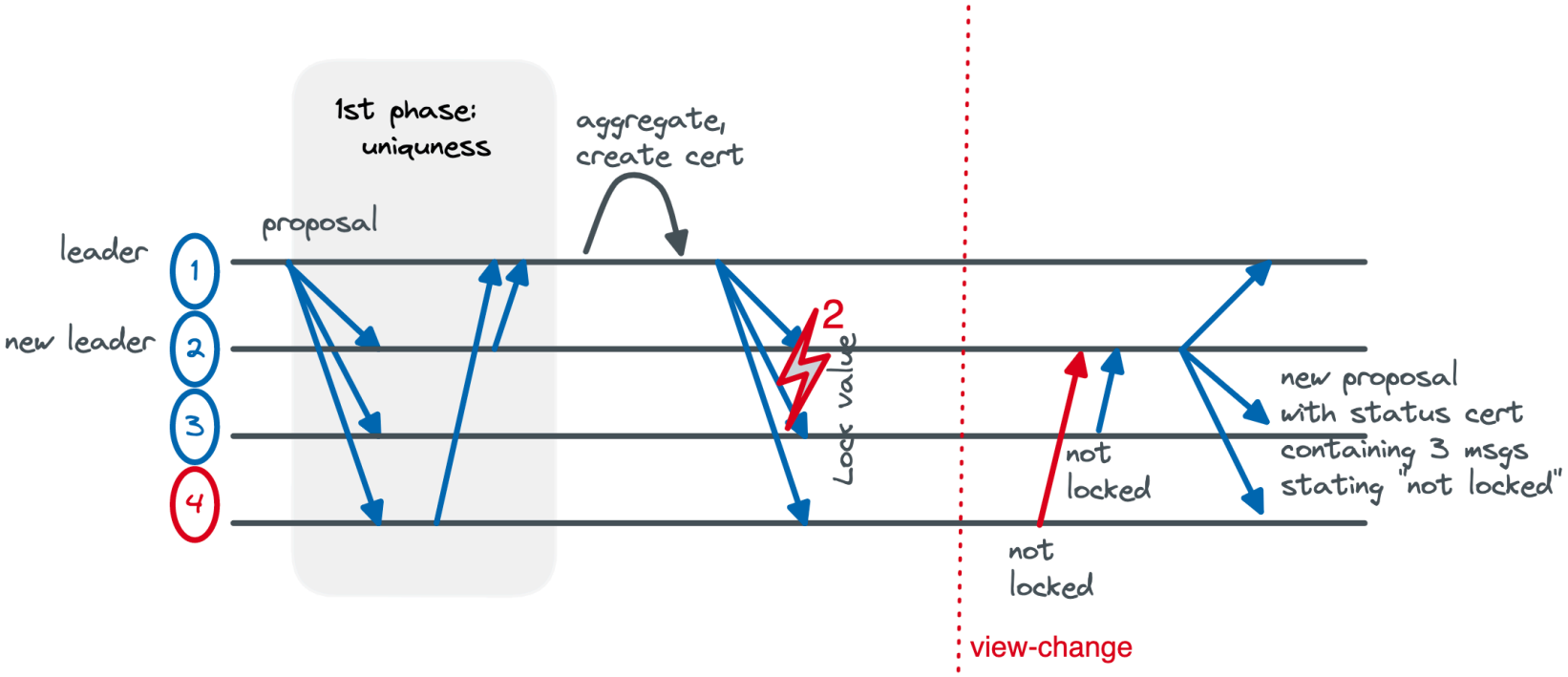
Three possible points of failure (either due to Byzantine leader or network failure):

(1) No node has locked or committed (free to change votes)
(2) Some nodes locked, none committed
(3) At least 2f+1 locked, some but not all committed

PBFT

Case 3: The leader will get at least one view-change message with the committed value; the nodes will receive that information in the new-view message (2f+1 locks)

Case 2: The leader knows no honest node committed a value, thus can safely propose new value; nodes can verify this in the new-view message



PBFT

✓ Safety

✓ Liveness

✓ Responsiveness

Communication Complexity

- View-change: $O(n^2)$
- Worst-case: $O(n^3)$

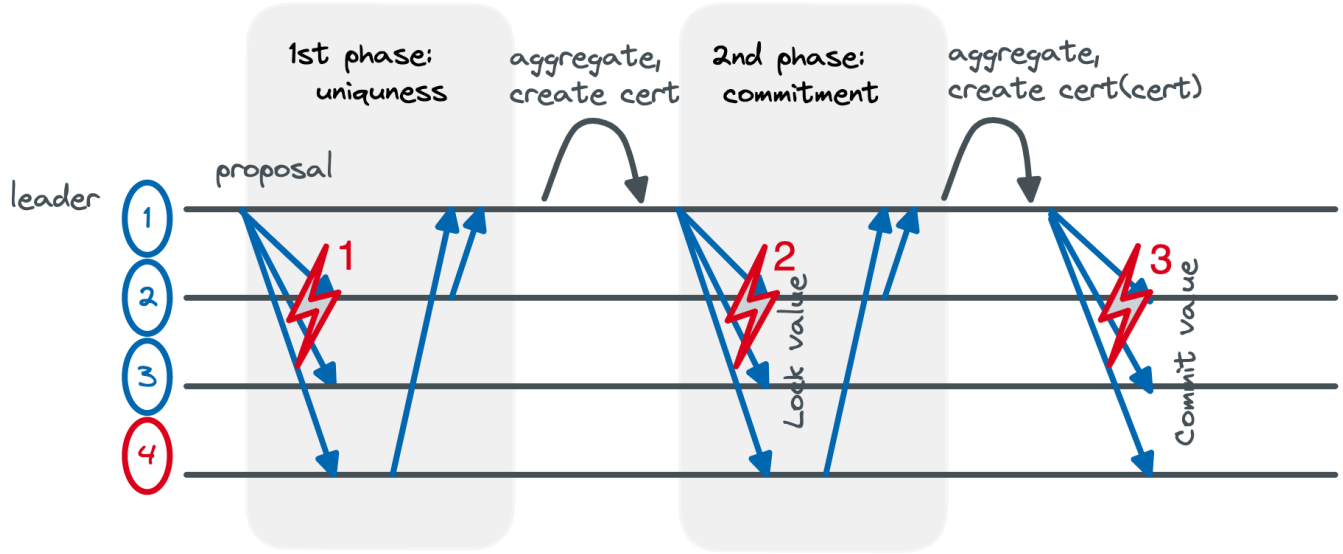
Latency (phases of voting): **2**

Tendermint (Casper)

Goal = linear complexity view change

- What is the problem with PBFT, if the leader does not send all the received view-change messages?

Nodes cannot distinguish between case 2 and case 3!



Tendermint (Casper)

How does the leader convince honest nodes to vote on its proposal?

- For height h, the leader proposes the block B, corresponding to the **lock with the highest view** v', and broadcasts <propose, B,h,v> along with the $QC_{lock}(B,h,v')$
- A node votes on <propose, B,h,v>, only if: (a) either it never locked for height h or (b) it locked a value for height h at view $v'' \leq v'$

Is that enough?

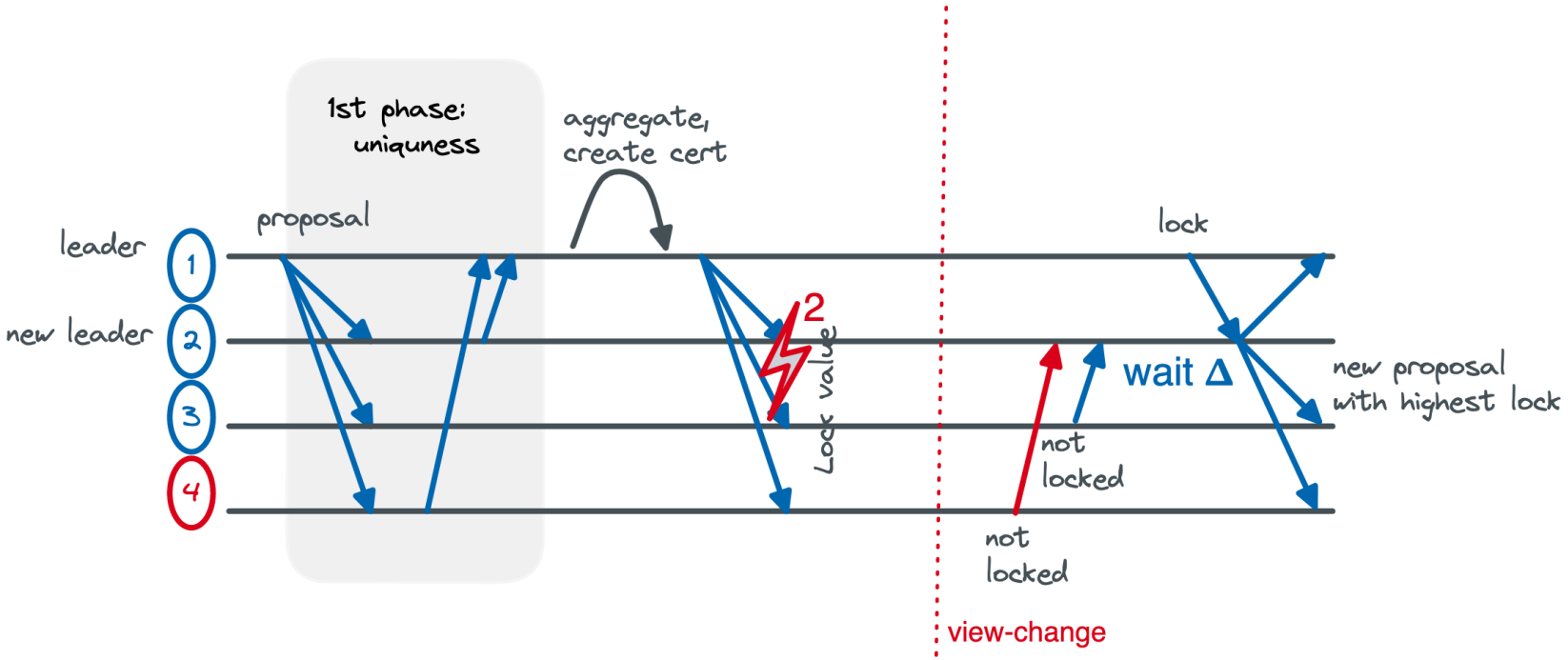
No! 1 honest node locking can hinder Liveness :(

- Suppose a node P locked in view v=100 block B but she is a very slow node so she never manages to send to the leader her lock within the first 2f+1 messages.
- The leader proposes block B' with the highest view v'=90 received from the 2f+1.
- The honest 2f nodes vote for B' while the f Byzantine do not respond. P does not vote because she safeguards her higher view lock (as v'=90<v=100).
- Note that P cannot tell if someone had committed later on her locked block, she only knows she should not change if the view of the proposed block B' is smaller!
- The protocol never makes progress: an honest leader after GST cannot get a QC.

Tendermint (Casper)

What does the leader learn about the system?

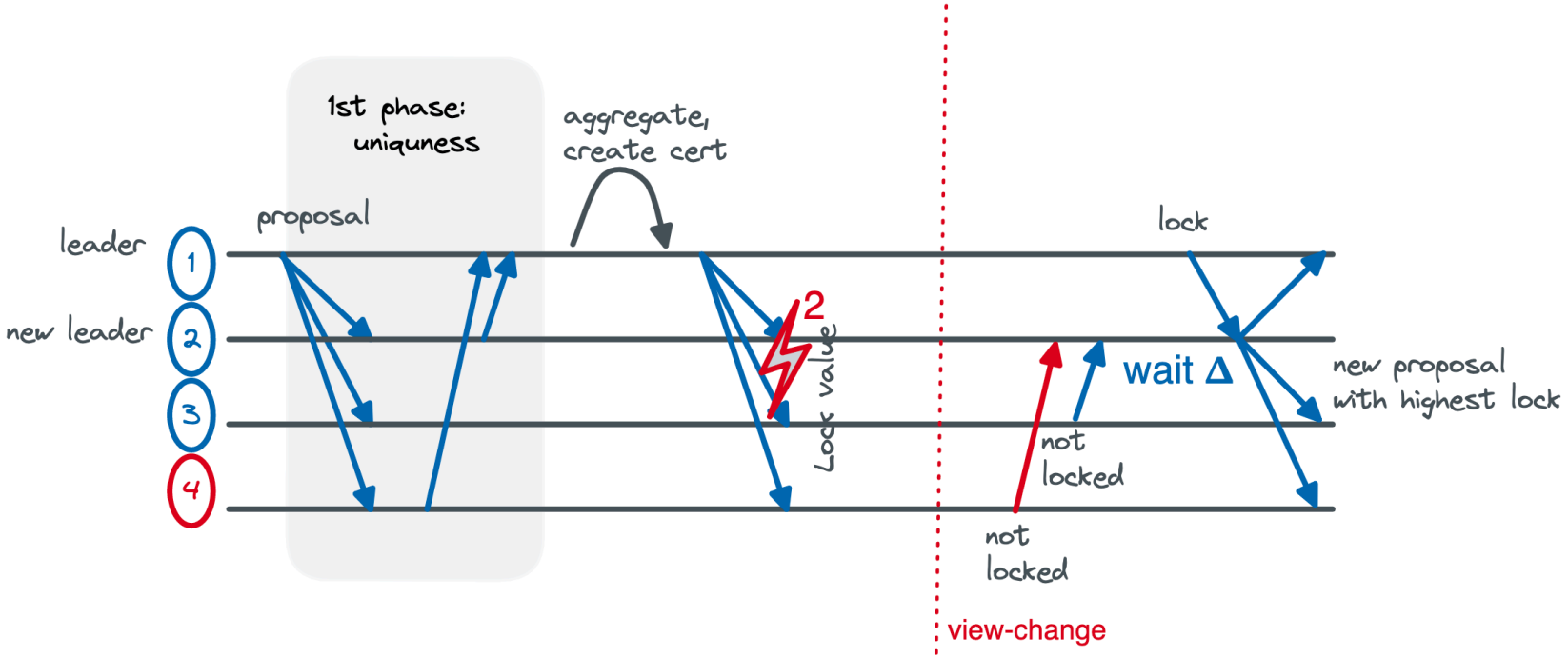
- The leader **waits for Δ time** and receives *all* honest nodes values



Tendermint (Casper)

While the leader learns the globally highest-ranked lock, the parties do not. However, the amount of information is sufficient to ascertain that the proposal is safe to vote on.

In other words, we don't distinguish between case 2 and Case 3! We just make sure if there is an honest node that has seen a QC_{lock} the leader will see it as well.



Tendermint (Casper)

Safety

Liveness

Responsiveness

Communication Complexity

- View-change
- Worst-case

Latency (phases of voting)

Tendermint (Casper)

✓ Safety

✓ Liveness

✗ Responsiveness - no, leader always waits for Δ

Communication Complexity

- View-change: O(n)
- Worst-case: $O(n^2)$

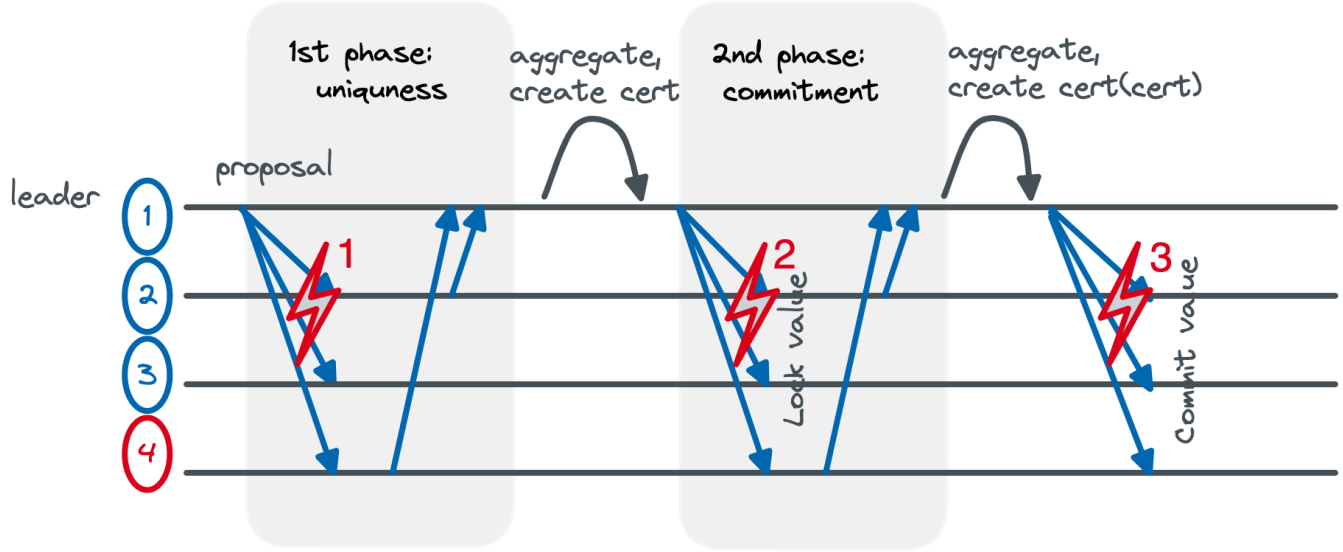
Latency (phases of voting): **2**

HotStuff

Goal = linear complexity view change + responsiveness

- What is the problem with Tendermint, if the leader does not wait for Δ time?

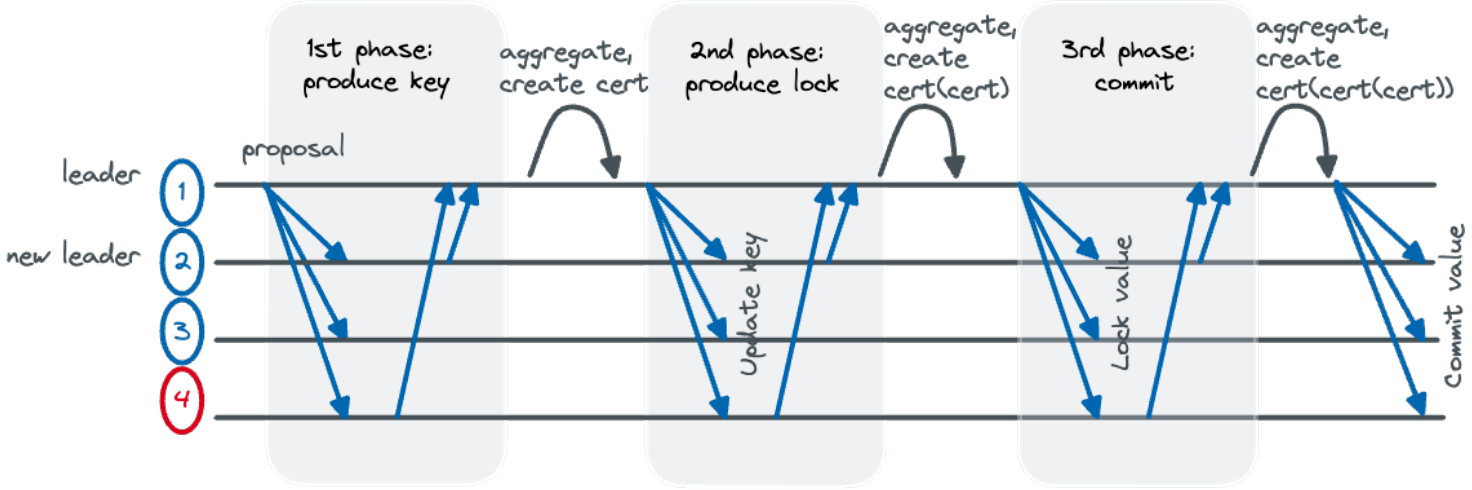
The leader may not receive the highest lock, hence the protocol will not make any progress as nodes will not vote the honest leader's proposal.



HotStuff

What does the leader learn about the system?

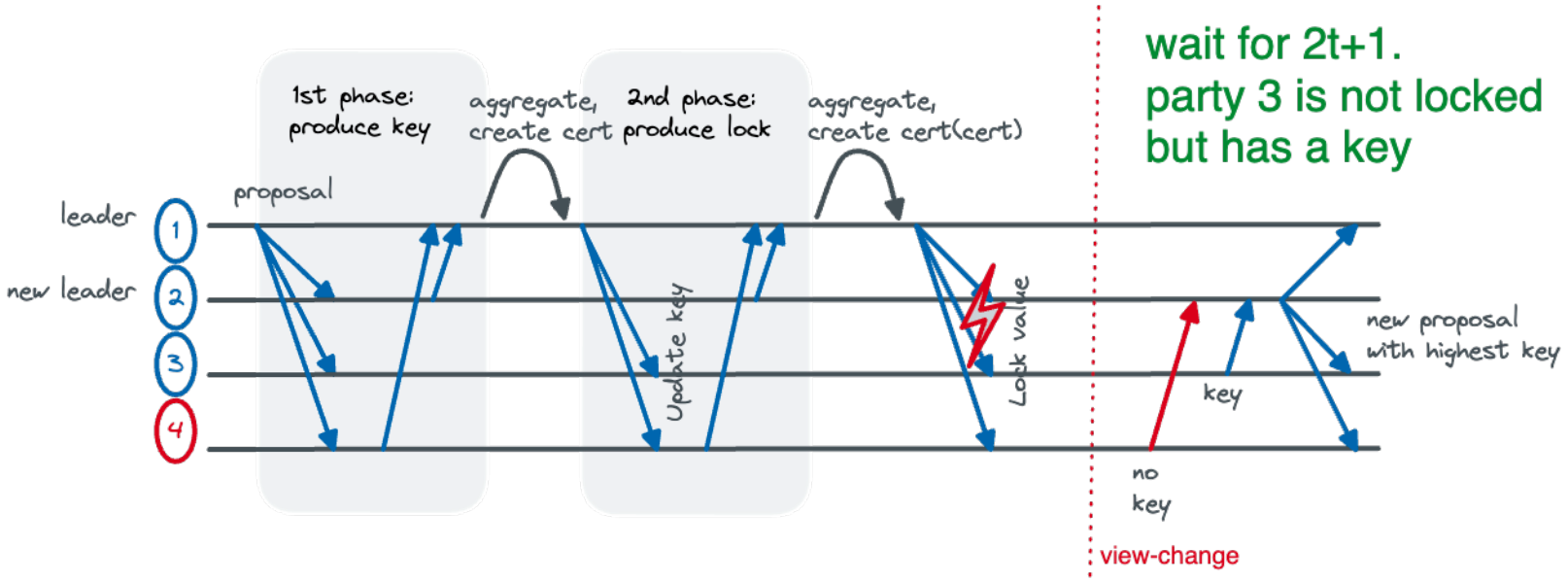
- The leader learns about any committed value through the view-change QC_{lock} (as in PBFT), and moreover learns the highest QC_{lock} (as in Tendermint). How?
- **Three-phase voting!**
  - The first vote guarantees 2f+1 votes on a unique value
  - The second vote guarantees 2f+1 nodes have a QC on a unique value
  - The third vote guarantees 2f+1 nodes have the highest-view QC on any value with votes at height h



HotStuff

How does the leader convince honest nodes to vote on its proposal?

- For height h, the leader proposes the block B, corresponding to the *key with the highest view* v', and broadcasts <propose, B,h,v> along with the $QC_{key}(B,h,v')$
- A node votes on <propose, B,h,v> only if: either (a) it never locked for height h (keys are not affecting safety) or (b) it locked a value for height h at view $v'' \leq v'$



HotStuff

Safety

Liveness

Responsiveness

Communication Complexity

- View-change
- Worst-case

Latency (phases of voting)

HotStuff

✓ Safety

✓ Liveness

✓ Responsiveness

Communication Complexity

- View-change: O(n)
- Worst-case: $O(n^2)$

Latency (phases of voting): **3**

Trade-offs

Metric/ Protocol	Phases	View-change communication complexity	Worst-case communication complexity	Responsive
PBFT	2	$O(n^2)$	$O(n^3)$	yes
Tendermint/ Casper	2	O(n)	$O(n^2)$	no
HotStuff	3	O(n)	$O(n^2)$	yes

Observations

- No parallel execution of different heights in Tendermint and HotStuff - high latency impact
- HotStuff is introduced in a blockchain fashion, each block pointing to the previous one and being finalised with the three-chain rule (three phases of voting)
- All protocols can be pipelined - phases can collapse
- There are other trade-offs beyond what we presented here: recent trent is to achieve optimistic finality in 1 round trip time if we relax the fault tolerance to f<n/5!

Consensus in Partial Synchrony

Questions?

Practical Byzantine Fault Tolerance. Miguel Castro and Barbara Liskov. OSDI, 1999.

What is the difference between PBFT, Tendermint, HotStuff, and HotStuff-2?
Kartik Nayak, Dahlia Malkhi. Decentralized Thoughts, 2023.

November 5th, 2025

TU WIEN

S&P Security & Privacy