HotStuff: BFT Consensus in the Lens of Blockchain

Maofan Yin , Dahlia Malkhi , Michael K. Reiter , Guy Golan Gueta , and Ittai Abraham
123 CornellUniversity, VMwareResearch, UNC-ChapelHill

ECS265 Paper Review Presentation

Presenters: Tong Zhu, Hongxiang Zhang, Siyuan Liu, Yifeng Shi, Junchao Chen

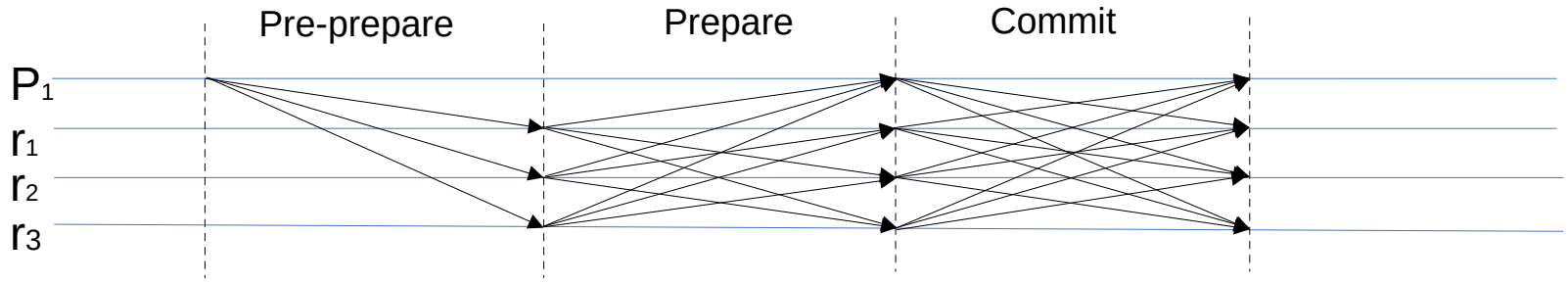
Table of Contents

PBFT (recap)

- Solve the Byzantine generals problem
- It is optimal, i.e., operates with 3f+1 nodes
- Deal with two things
  - Malicous primary/replicas
  - Consensus
- Limitation: Scaling issue (Communication Costs)

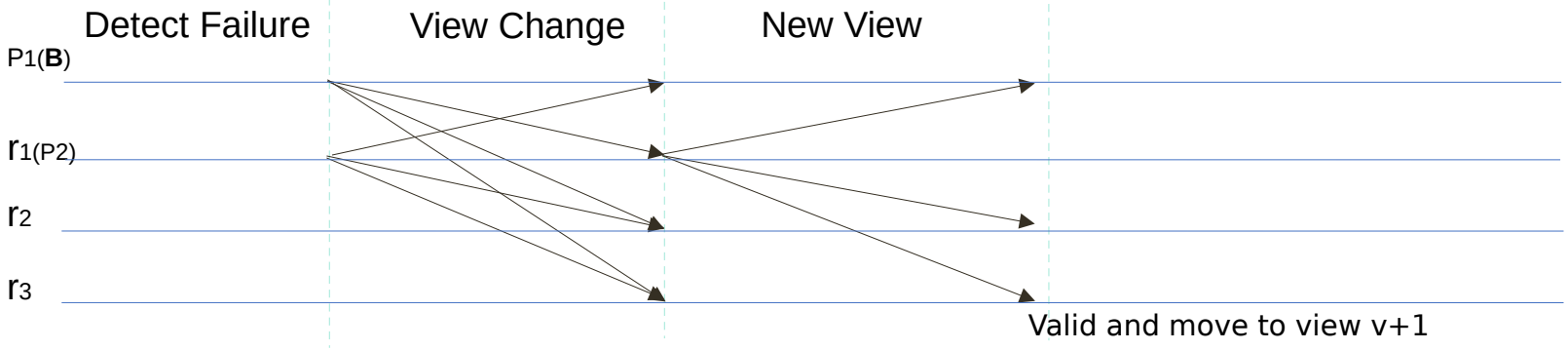
How to reduce communication cost ?

Normal case operation



Communication complexity: **O(n^2)**

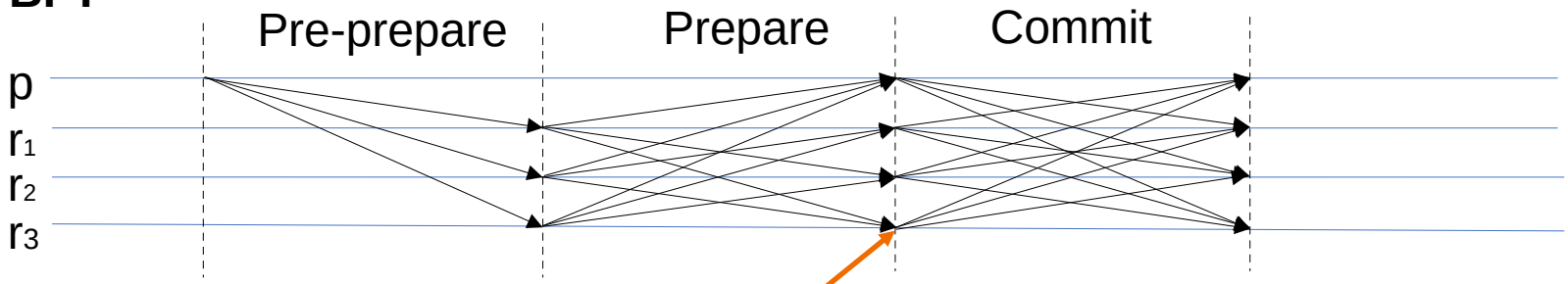
View change



Communication complexity: **O(n^3)**

How to reduce communication cost ?

PBFT

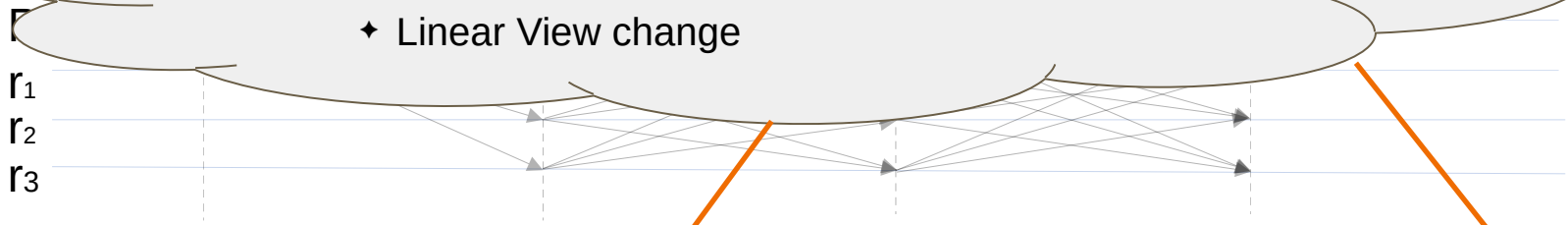


Comunication comlexity: **O(n^2)**

Ask a primary to generated the vote result ?
And how to reduced the message number (from primary replica to others)?

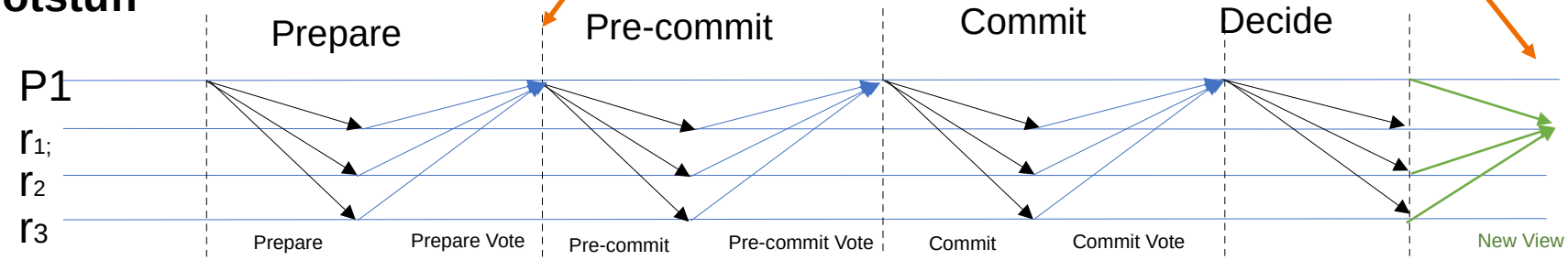
How to reduce communication cost ?

- Ask a primary to generate and broadcast vote results.
- Apply **Quorum Certificate (QC)** to reduce message number
- Linear View change



Communication complexity: **O(n^2)**

Hotstuff



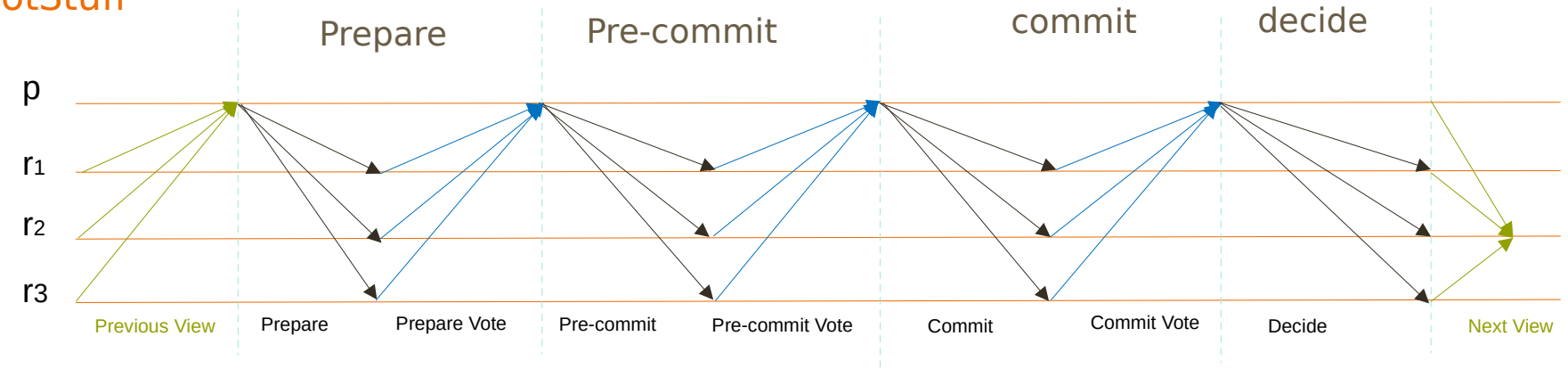
Communication complexity: **O(n)**

What is HotStuff ?

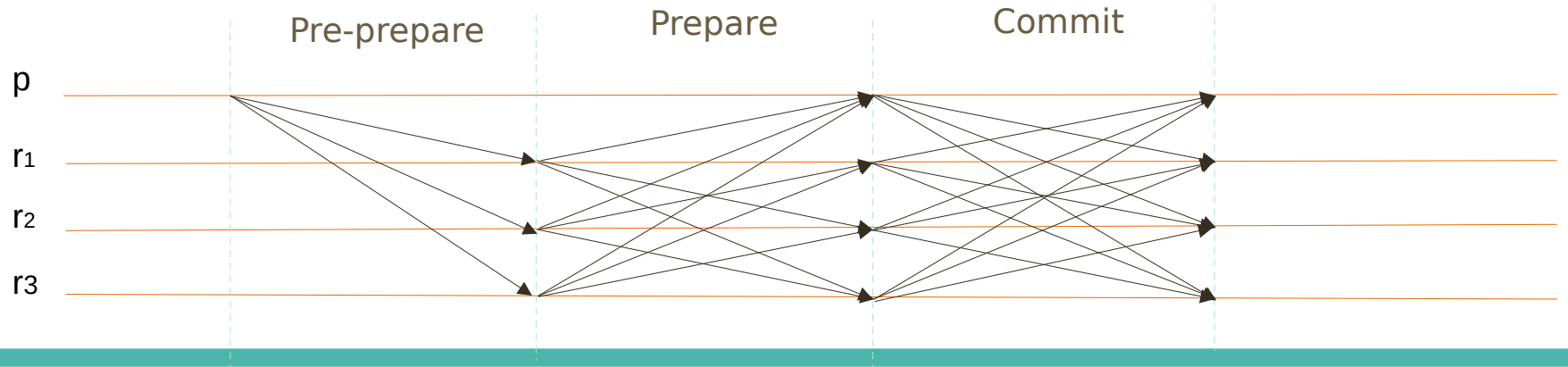
- A **primary-based** Byzantine fault-tolerant replication protocol for the **partial synchronous model**
  - A Paradigm distilled from quorum-based consensus – “**Quorum Certificate**”
  - No special treatment of **view change** (Linear view change)

Phases

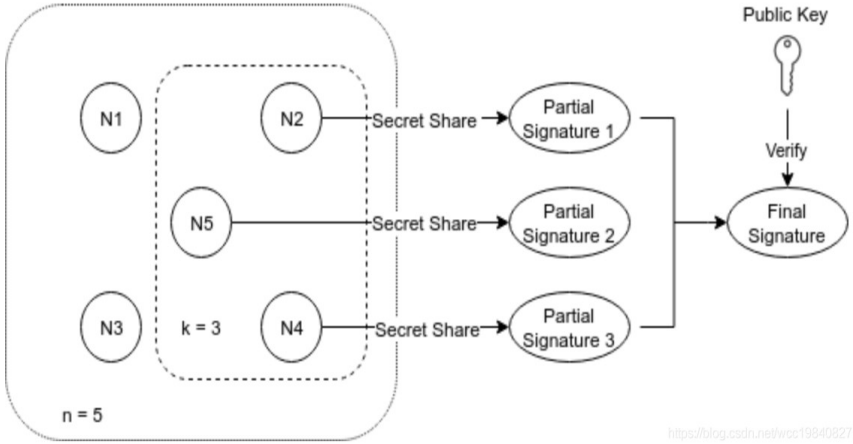
HotStuff



PBFT



Threshold Signature



In a (k,n) threshold signature scheme:

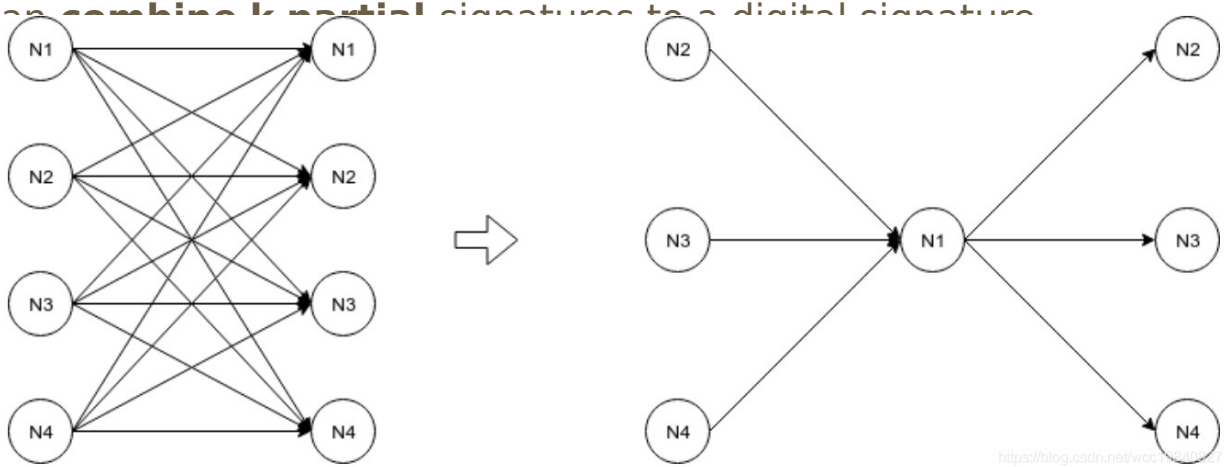
- A public key held by all replicas.

- Each replica holds a **distinct** private key.

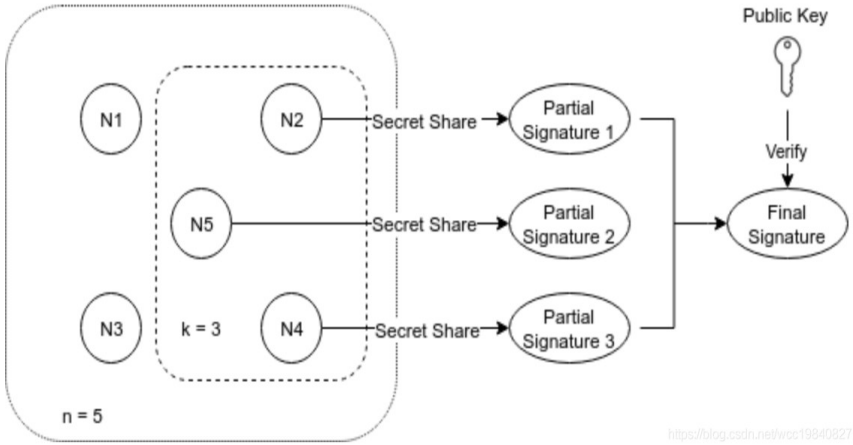
- Each Replica can sign the message using its private key.

- Each Replica c**ombine k partial** signatures to a digital signature

- Each Replica c



Threshold Signature



In a (k,n) threshold signature scheme:

- A public key held by all replicas.

- Each replica holds a **distinct** private key.

- Each Replica can sign the message using its private key.

- Each Replica can **combine k partial** signatures to a digital signature.

- Each Replica can **verify** the digital signature using the **public key**.

Advantages:

2f+1

- Reduce both the size of the message and the number of signatures to verify.
- Significantly reduce the scale of communication for each node.

Stable Primary vs Rotating Primary

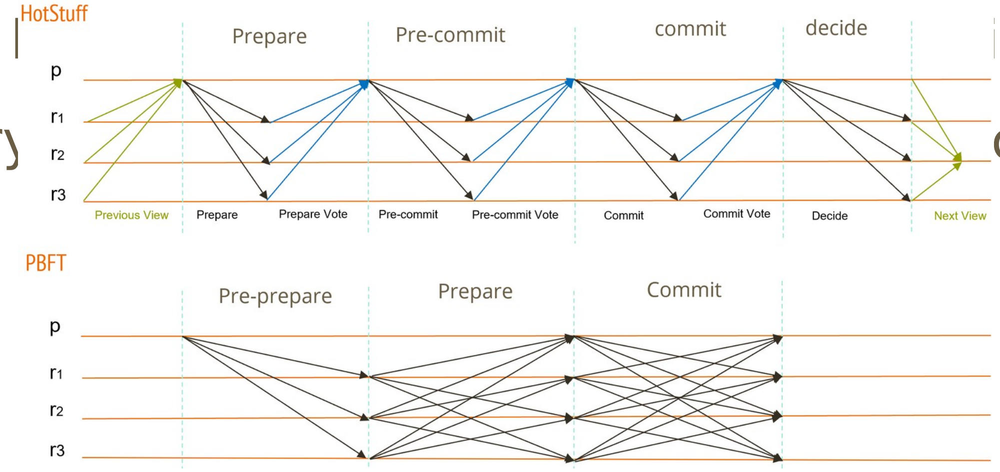
Most protocols have the ability to replace the primary via a view-change.

- PBFT is based on the stable primary paradigm:

A primary is changed only when a problem is detected.

- HotStuff is … gm:

A primar… commit a command.



Stable Primary vs Rotating Primary

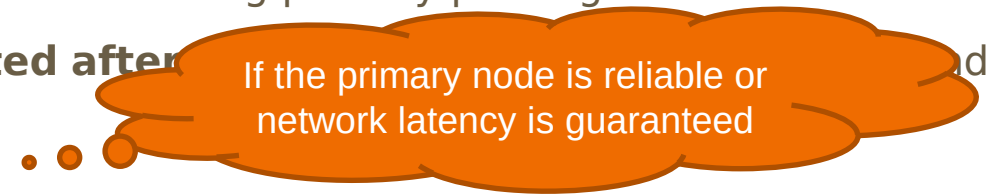
Most protocols have the ability to replace the primary via a view-change.

- PBFT is based on the stable primary paradigm:

A primary is changed **only when** a problem is detected.

- HotStuff is based on the rotating primary paradigm:

  A primary is **rotated after**

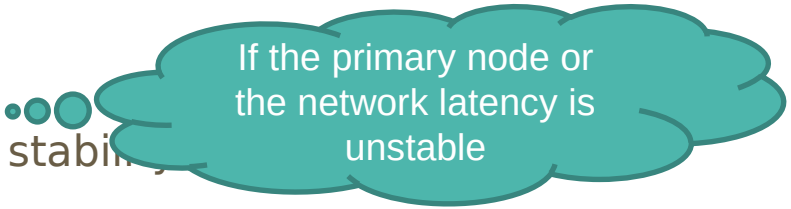


d.

It is a **trade-off**:

- Stable Primary:

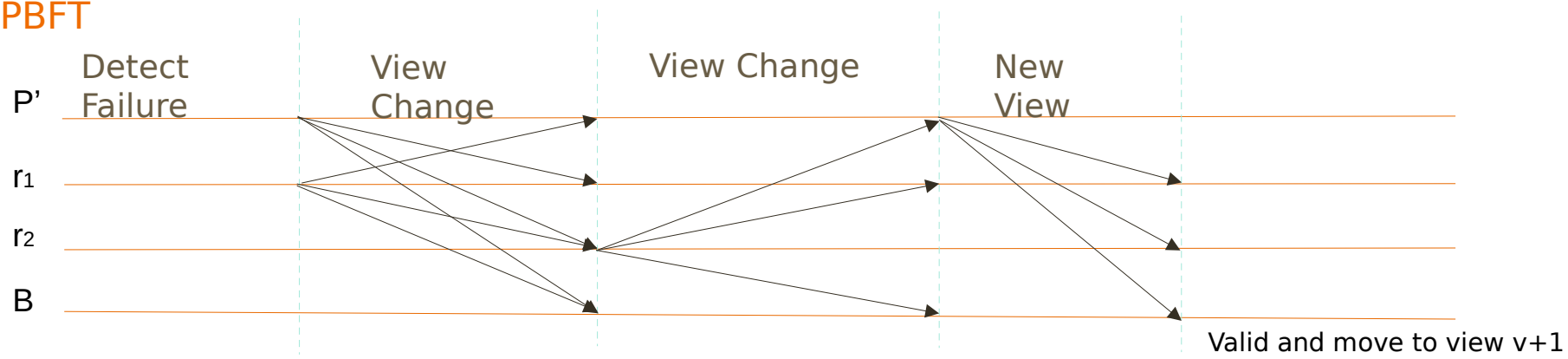
  - less overhead and better performance due to stabi



  - a stable malicious primary can cause undetectable malicious actions.

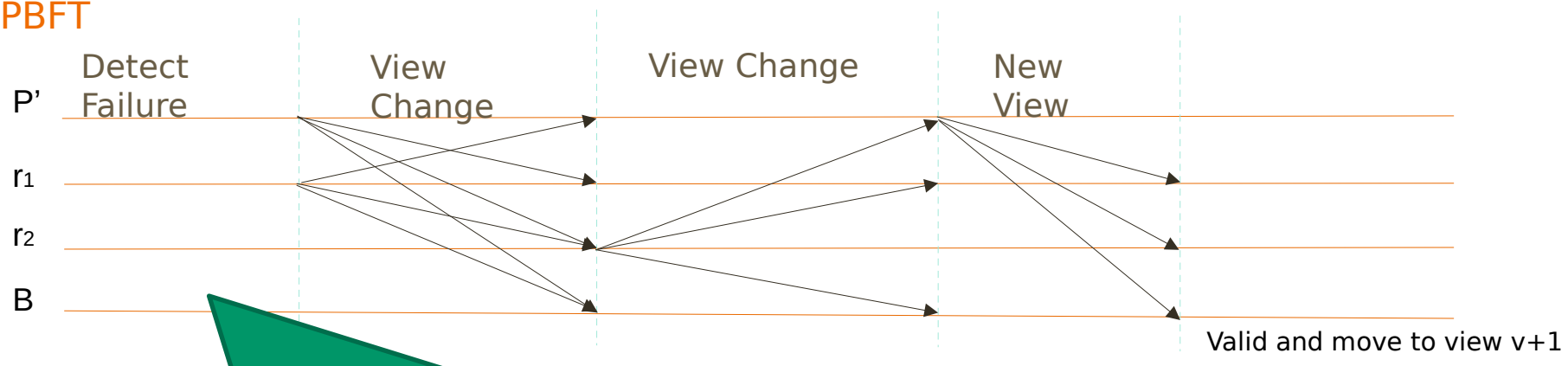
View Change

PBFT



View Change

PBFT



each replica set up a **timer**:

after receiving a Pre-prepare proposal from the primary

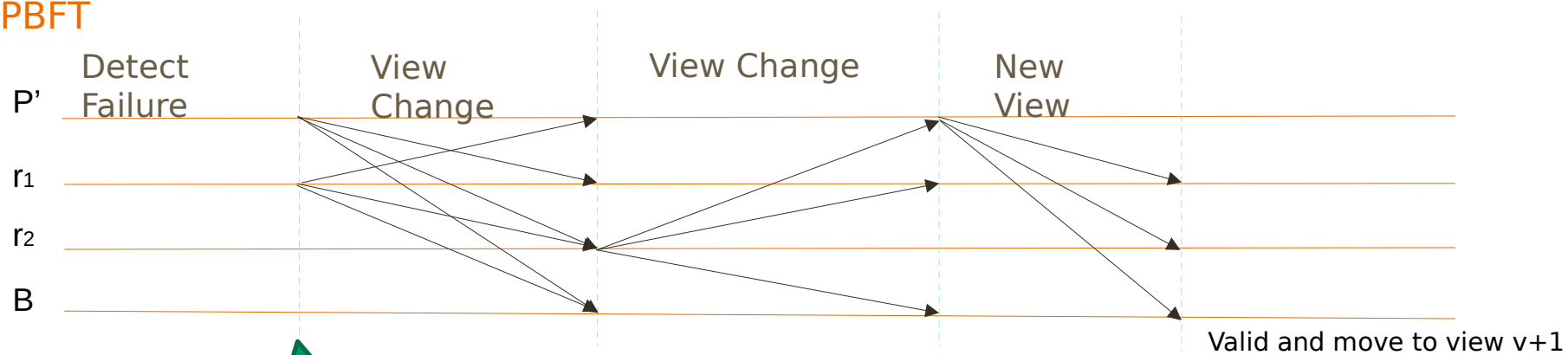
or

after forwarding a client request to the primary

if the timer **expires** before progress is made, a replica detects the failure of the current primary.

View Change

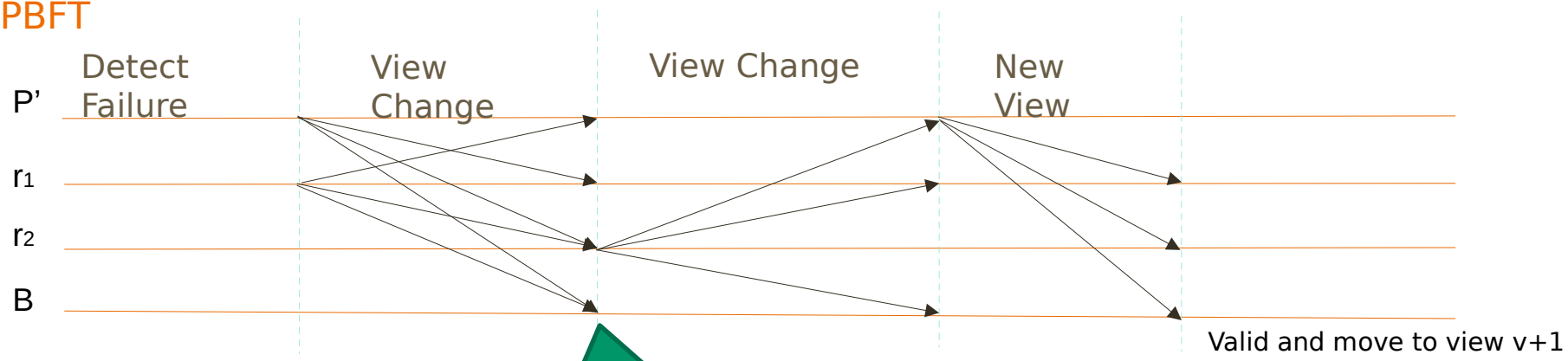
PBFT



halting current work, broadcasting a view change request.

View Change

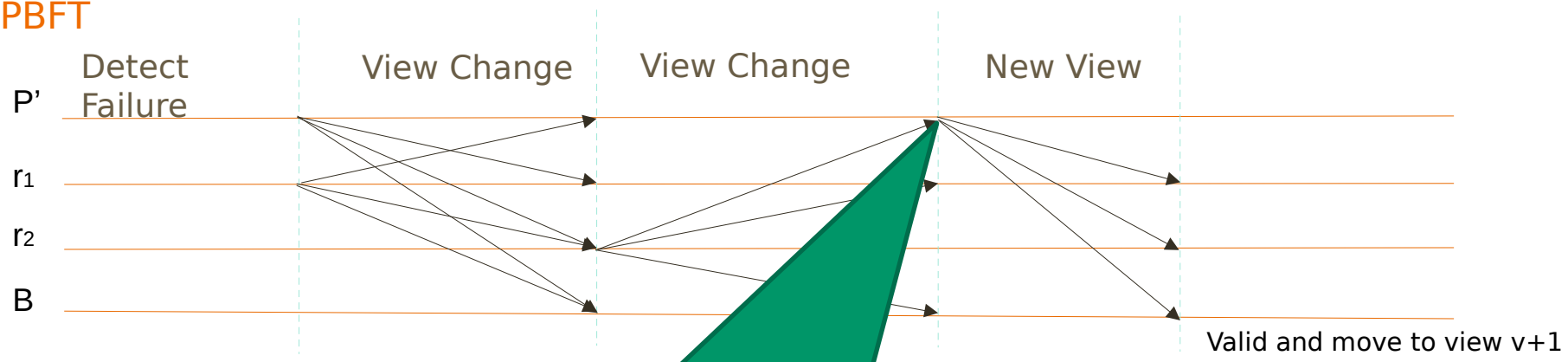
PBFT



R2 will eventually detect the failure owing to part of the replicas being halted and dealing with the view change process.

View Change

PBFT

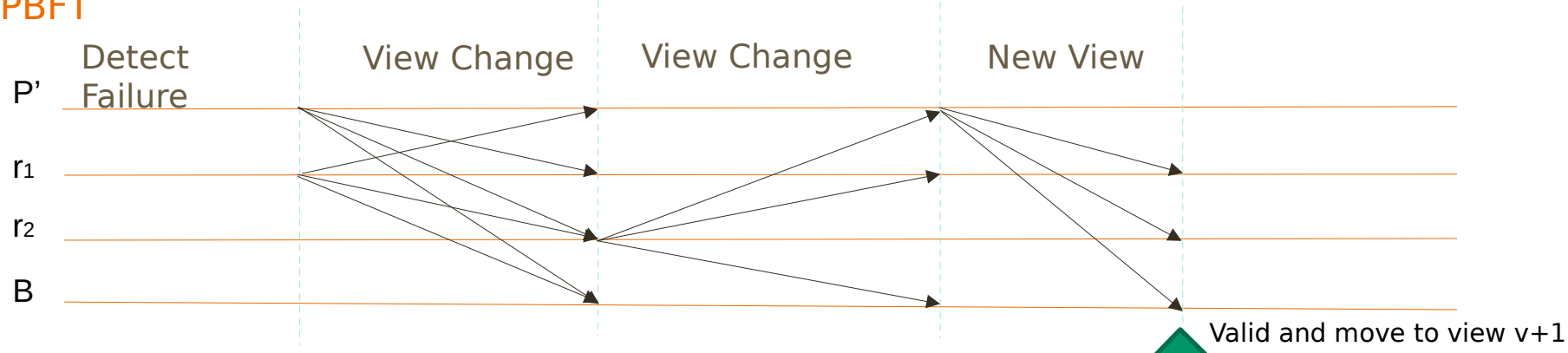


After receiving 2f+1 view change messages, the new primary node will broadcast a new view message with v=v+1.

New primary node p', which id(p') =(v+1) mod n.

View Change

PBFT

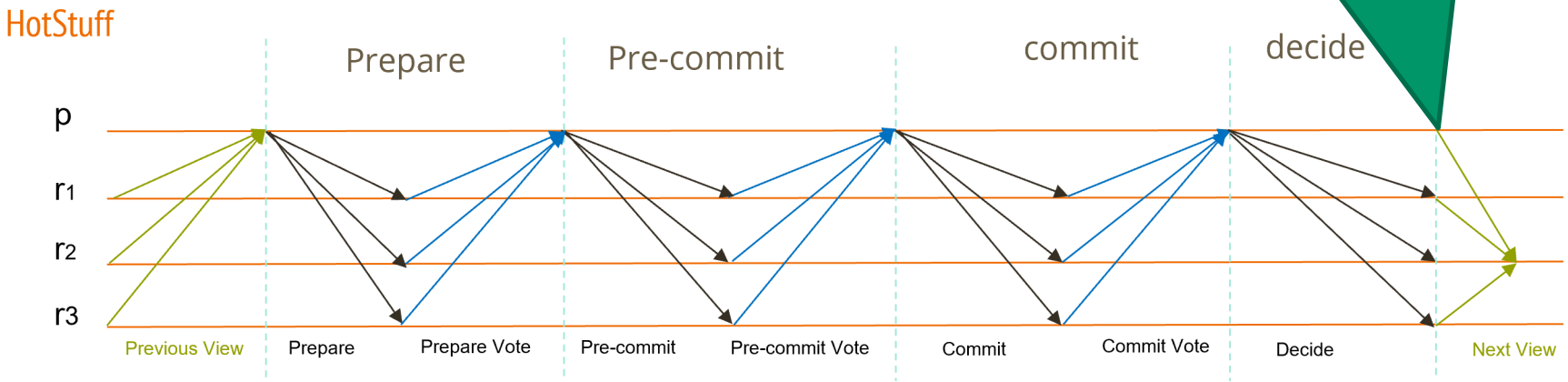


If the new view message is valid, all replicas will move into the next view.

View Change

Each Replica, including the current primary, will send a new-view message to the new primary:
1. When timeout.
2. After committing a proposal.

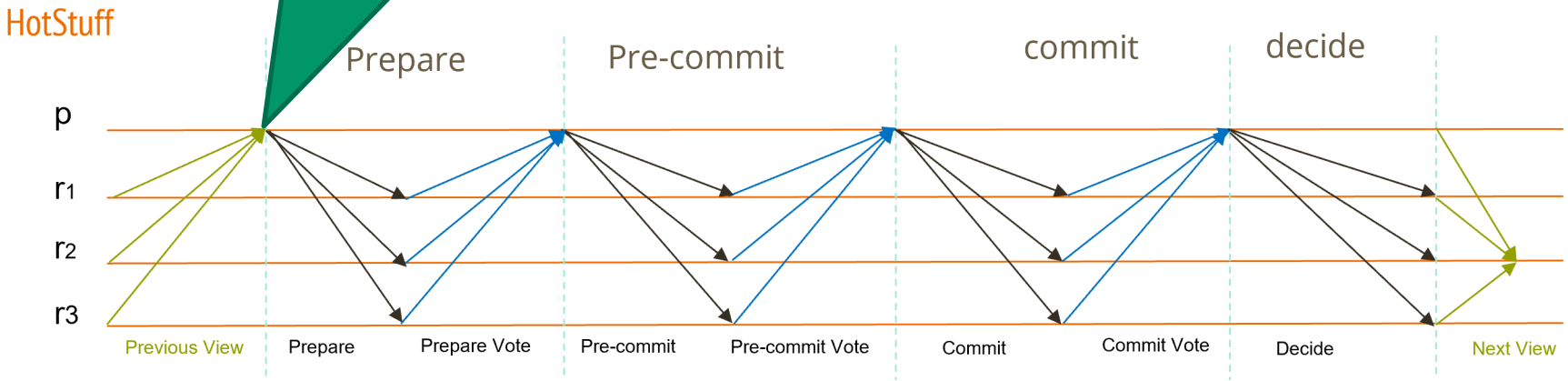
Including its prepare QC.



View Change

New primary will:

1. Calculate the highest QC among 2f+1 new-view messages received.
2. Attaching the highest QC in the Prepare proposal of processing a new client request.



Safety

What is **Safety** ?

Safety

What is **Safety** ?

The replicated service “behaves like a centralized implementation that executes operations atomically one at a time”

Safety

What is **Safety** ?

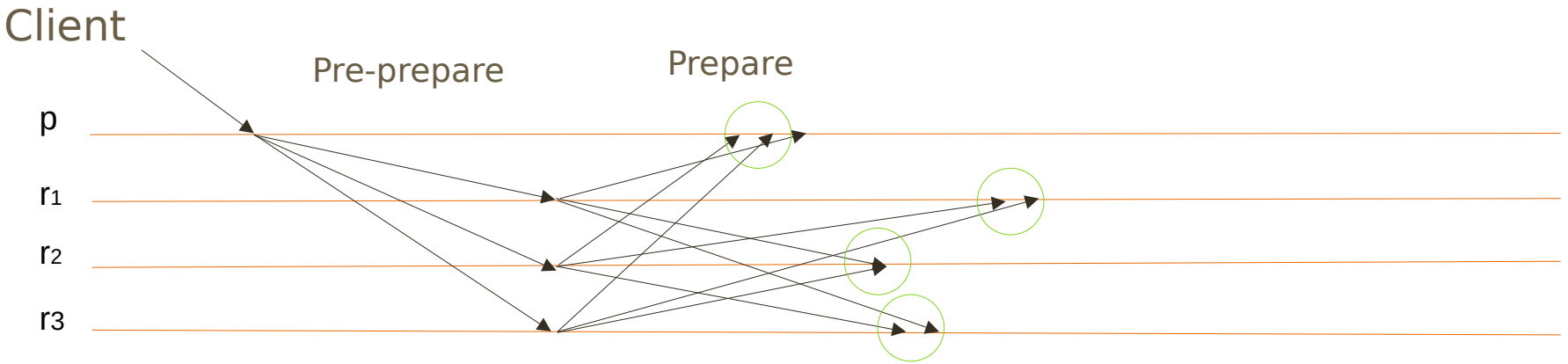
The replicated service “behaves like a centralized implementation that executes operations atomically one at a time”.

Each state machine replica should make the same decision and reach the same state for a client request.

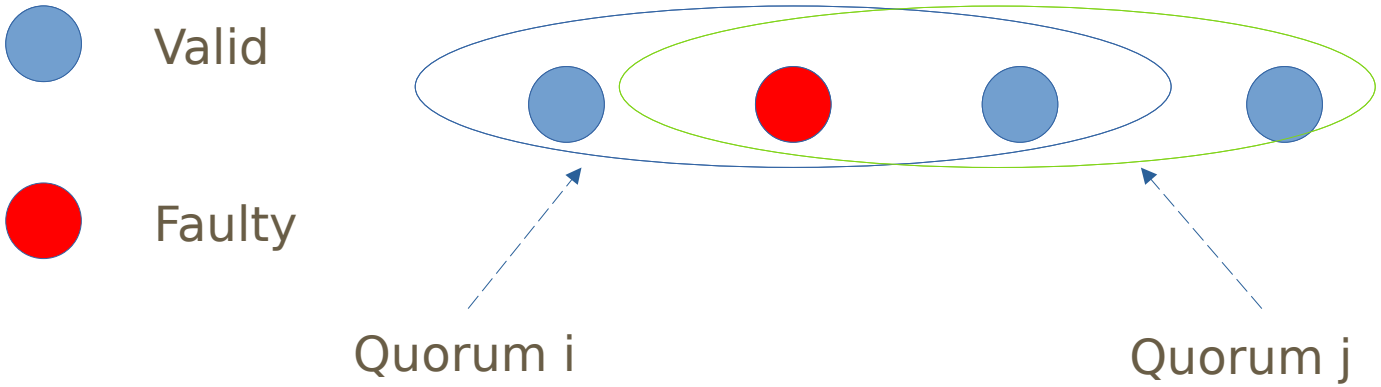
Safety

In PBFT, Pre-prepare and prepare ensure replicas agree to commit the same value for a client request in the same view

A replica receives 2F + 1 prepare messages that match the pre-prepare message it receives to become prepared



Safety

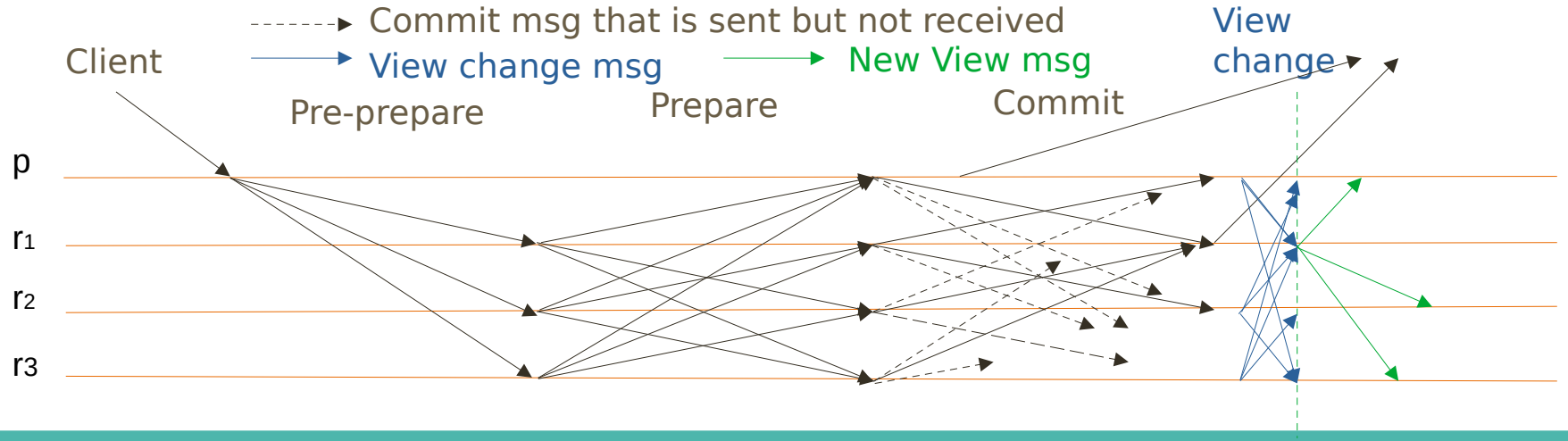


When F = 1, N = 4, the intersection of quorums for any two valid replica i, j after receiving 2F + 1 prepare messages in prepare stage contains at least one valid replica. Because valid replica does not lie, this ensures them to agree on the same value to commit.

Safety

In PBFT, Prepare and Commit phase ensure replicas agree to commit the same value for a clilent request across views

A replica receives 2F + 1 commit messages that match the pre-prepare message it receives to be commited

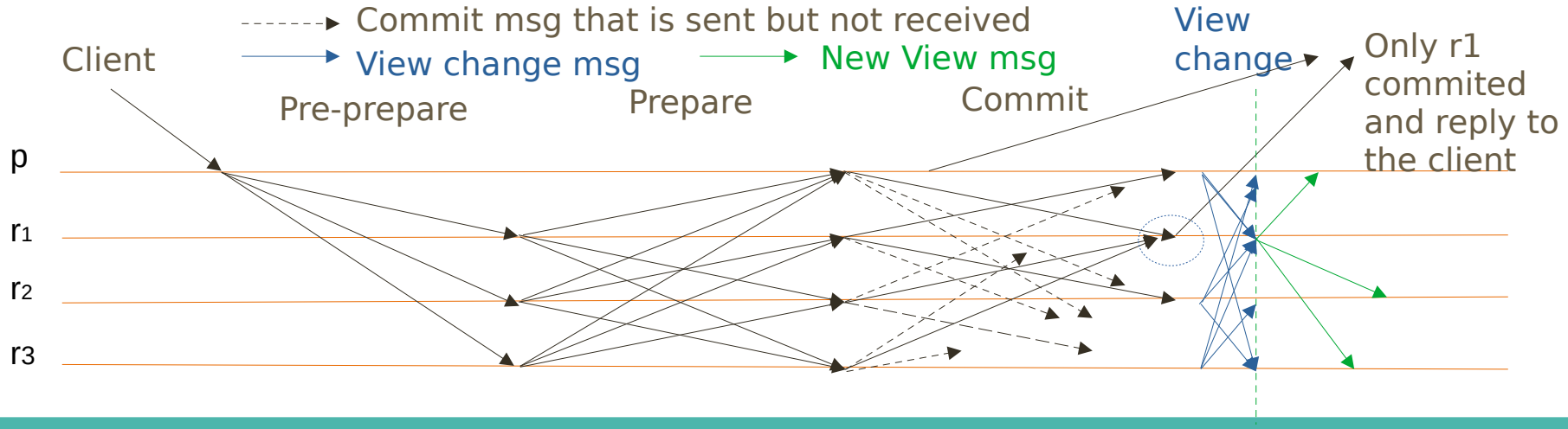


Safety

In PBFT, Prepare and Commit phase ensure replicas agree to commit the same value for a sequence across views

A replica receives 2F + 1 commit messages that match the pre-prepare message it receives to be commited

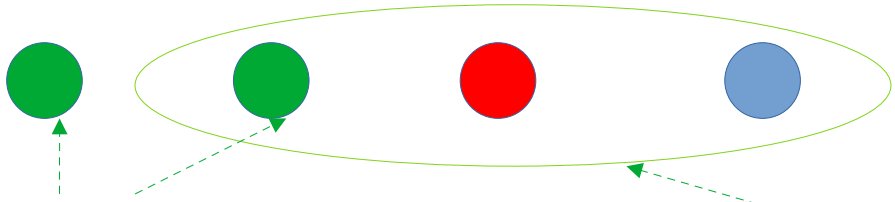
For example, suppose only f_1 is commited before view change, at least F + 1 valid node have prepared



Safety

Valid

Faulty



F + 1 valid node that is both prepared and requested the view change

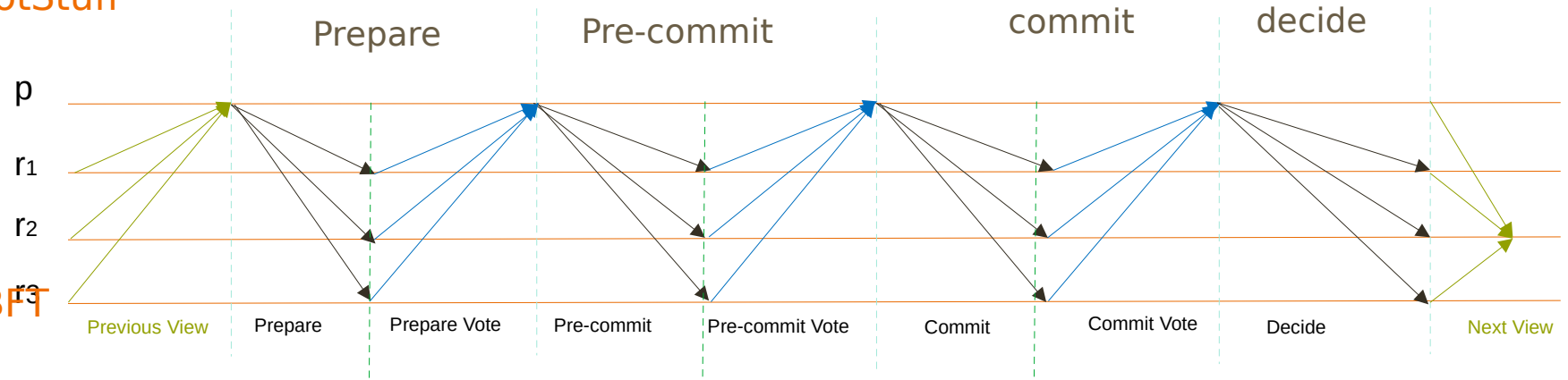
2F + 1 nodes requested view change

Valid and prepared for some message in previous view

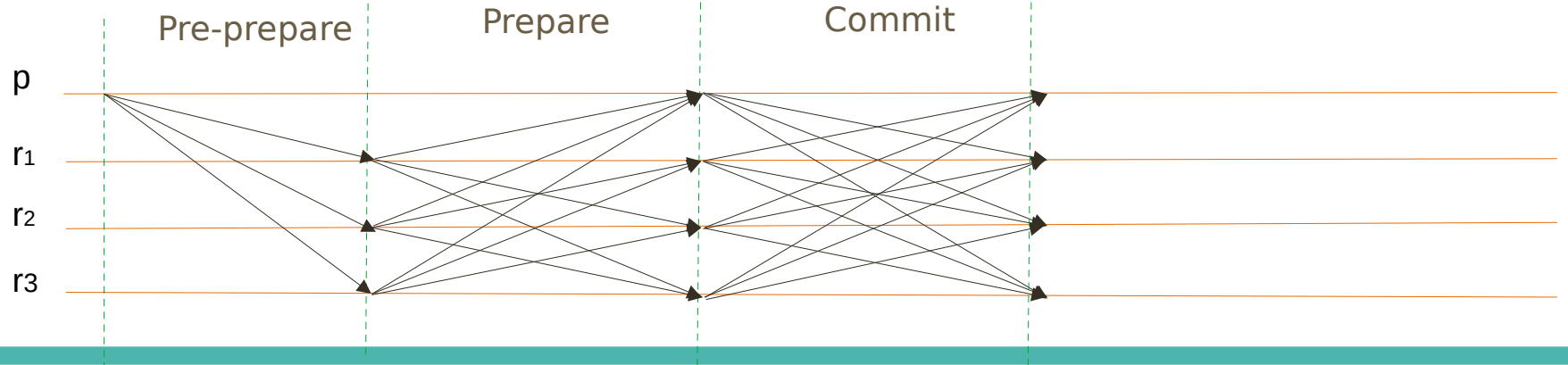
When F = 1, N = 4, there is at least one valid node that is both prepared and requested the view change. This ensures the request previously commited is propogated to the new view and we can redo this request with 3 phases as we have described.

Safety

HotStuff



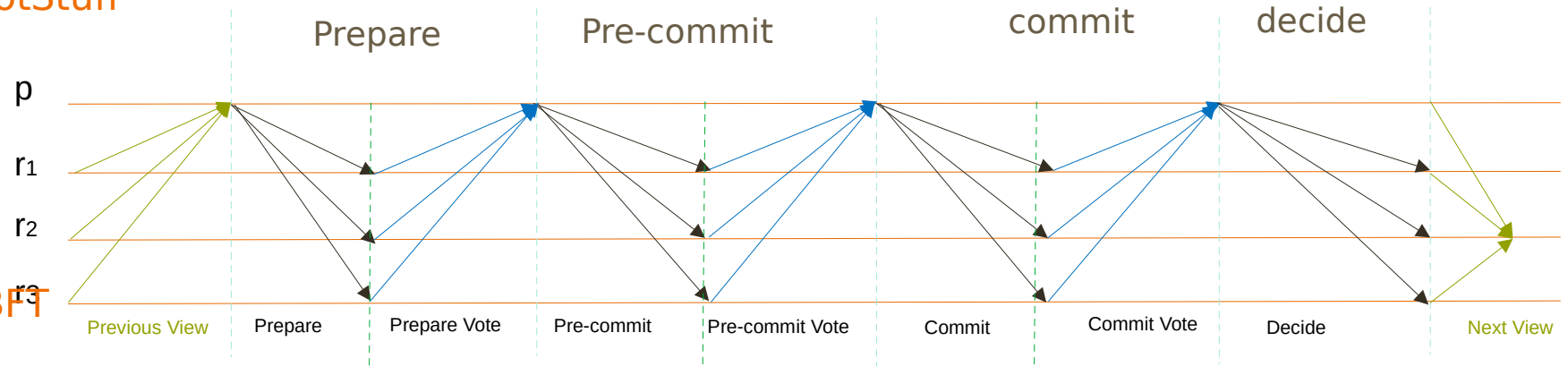
PBFT



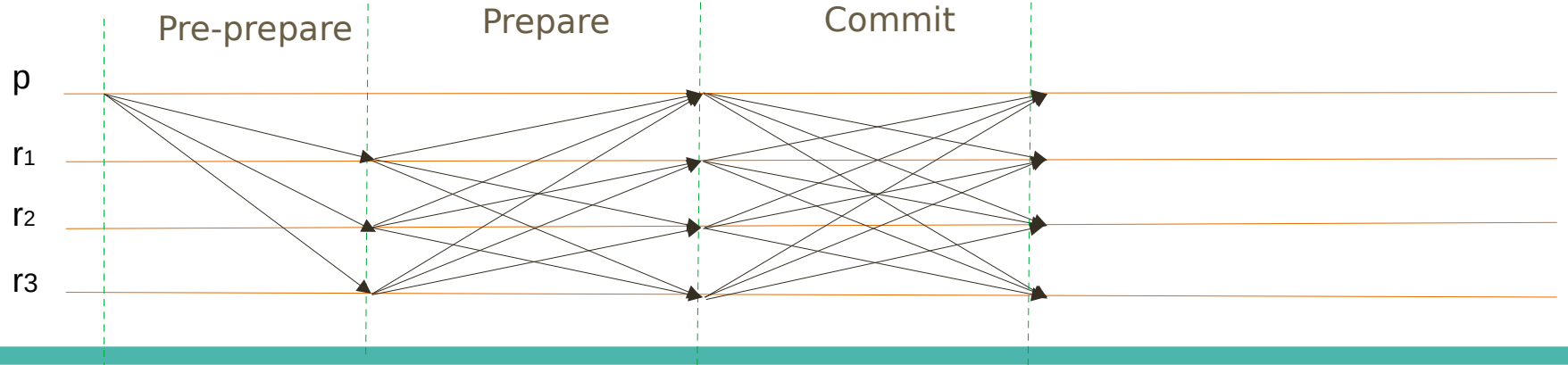
Safety

So why does HotStuff need Decide phase?

HotStuff

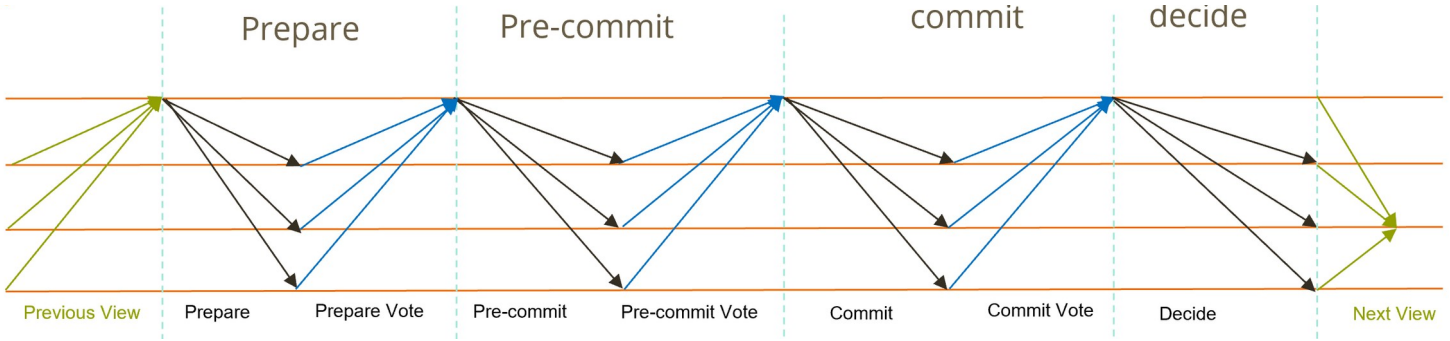


PBFT



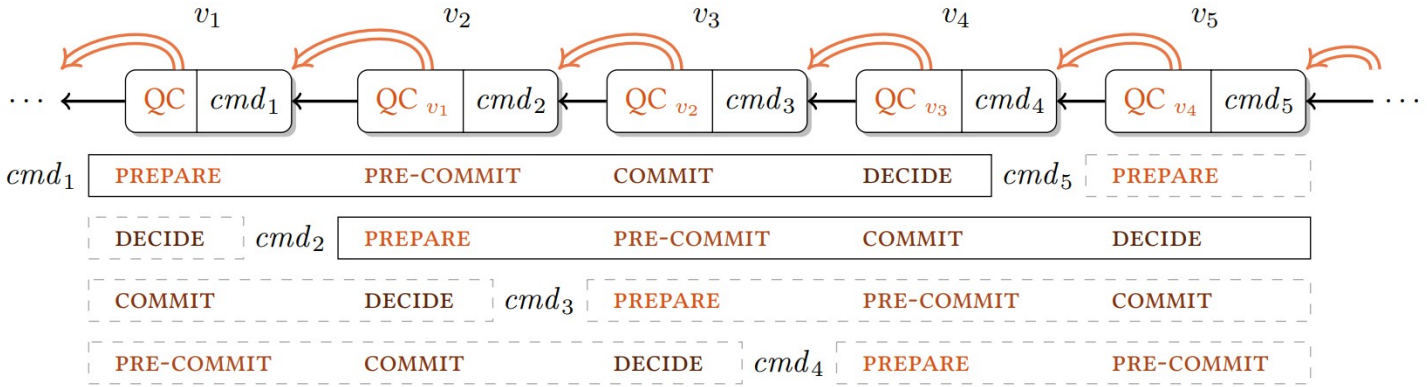
Chained HotStuff

Basic

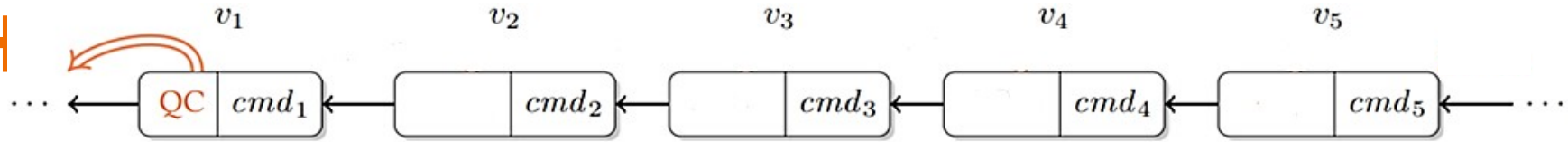


Change the view on every prepare phase
-> each proposal has its own view.

Chained



Chained HotStuff- Dive into chained H



Build a QC

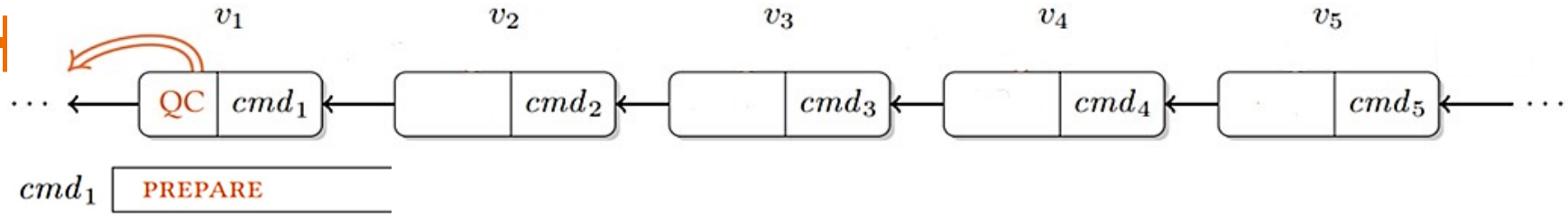
P1 P2 P3 P4 P5

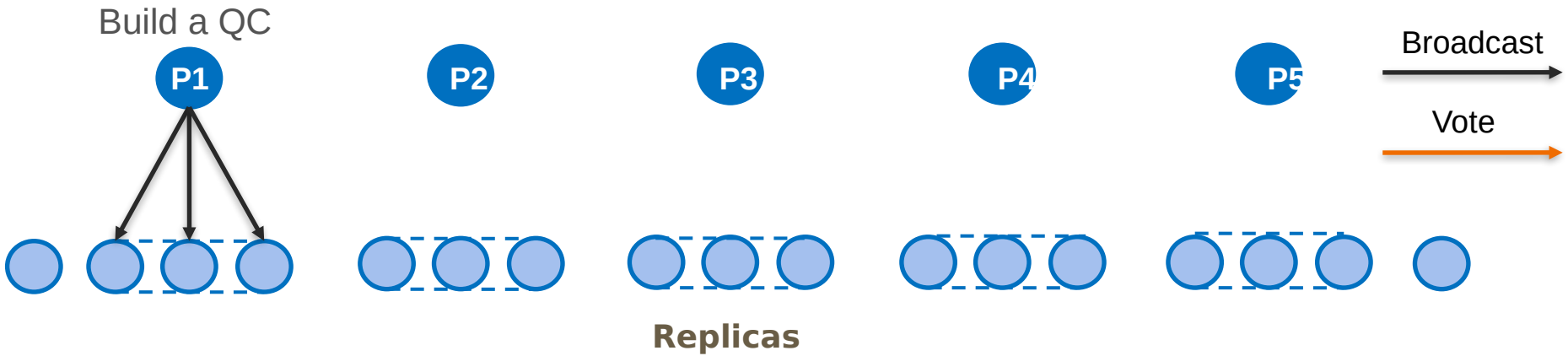
Broadcast

Vote

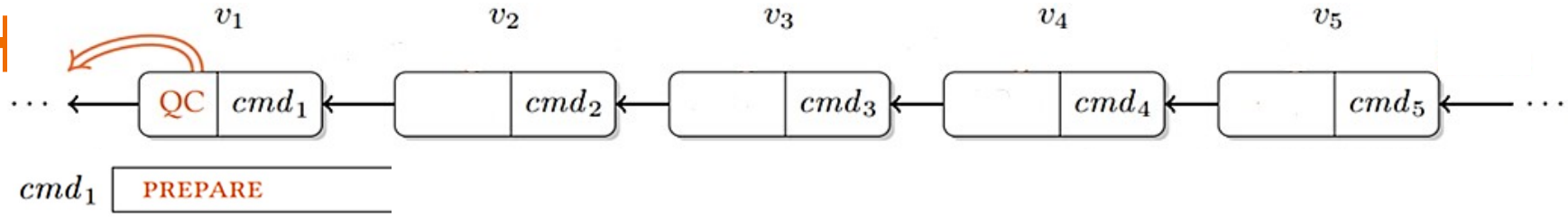
Replicas

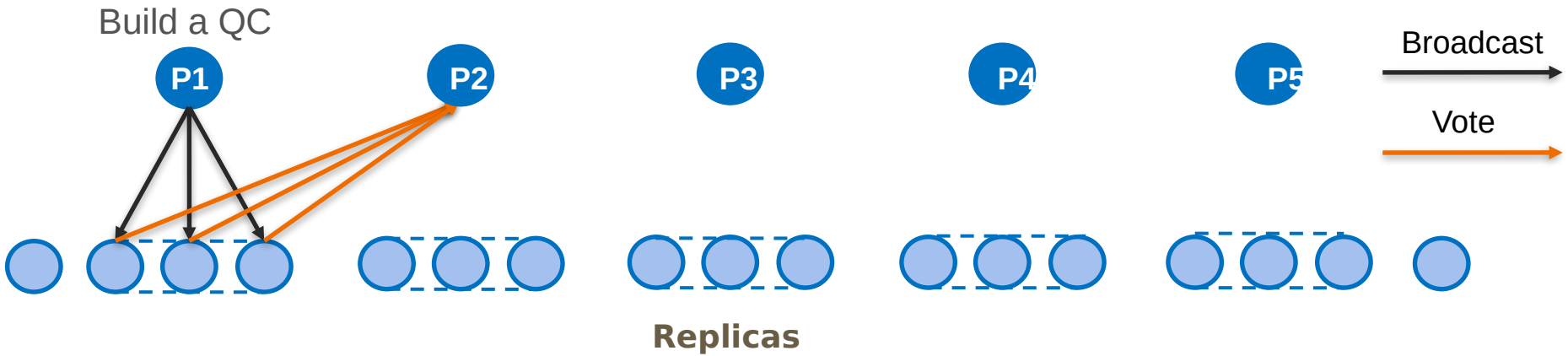
Chained HotStuff- Dive into chained H



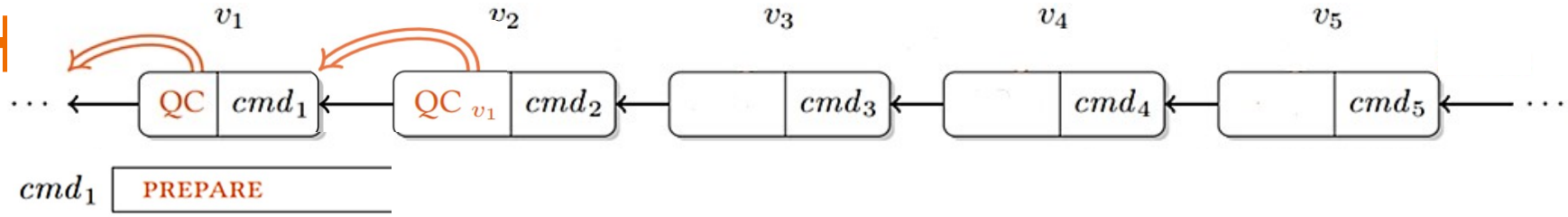


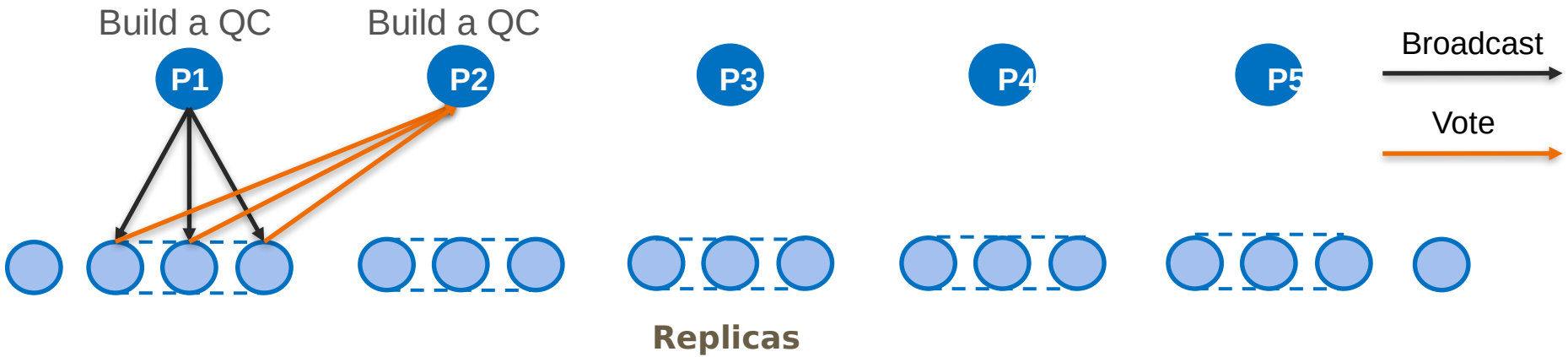
Chained HotStuff- Dive into chained H



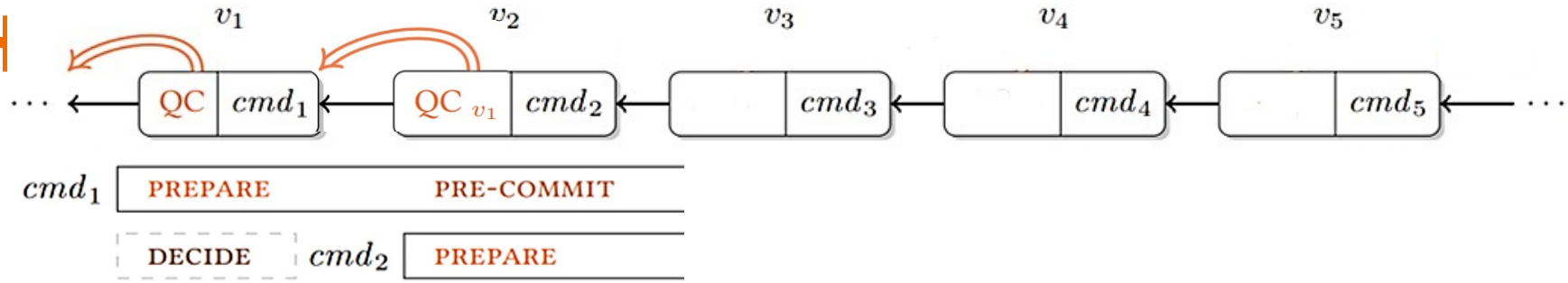


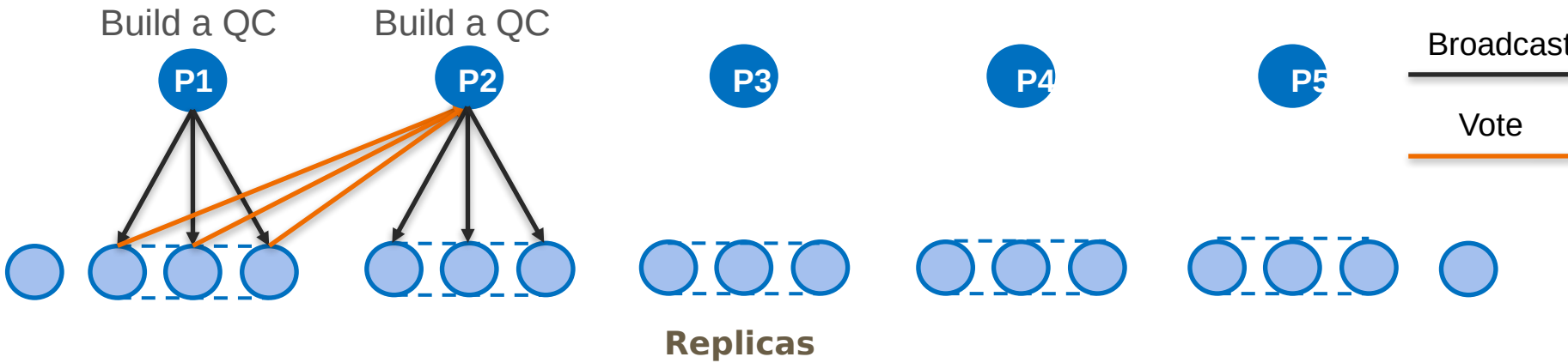
Chained HotStuff- Dive into chained H



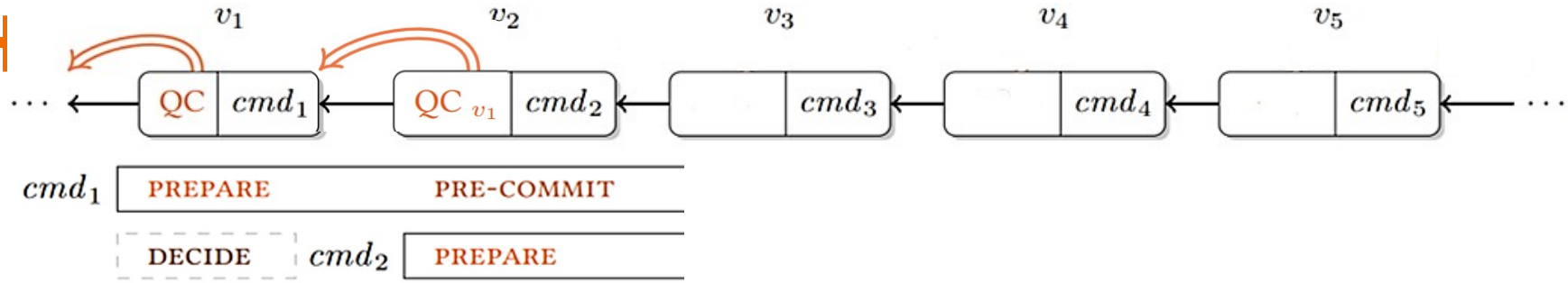


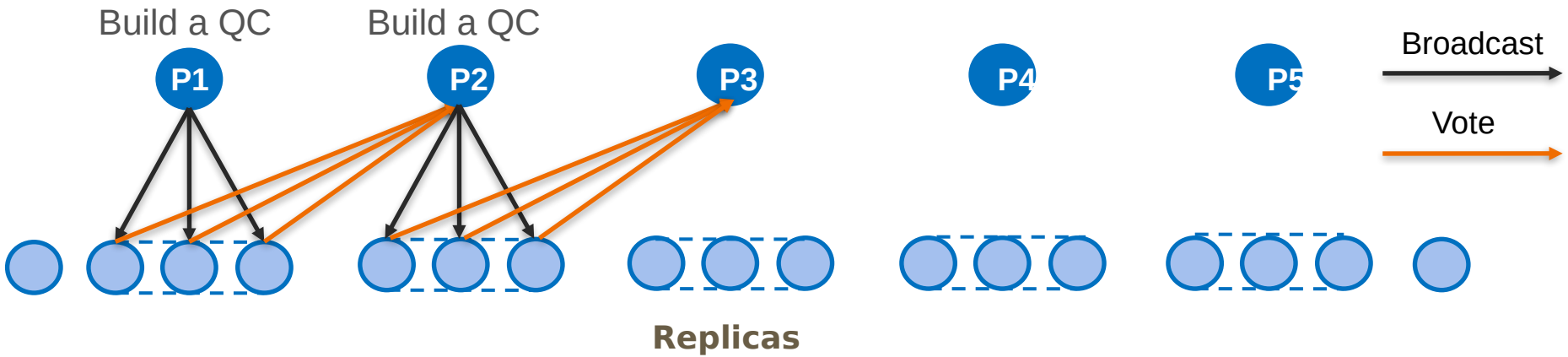
Chained HotStuff- Dive into chained H



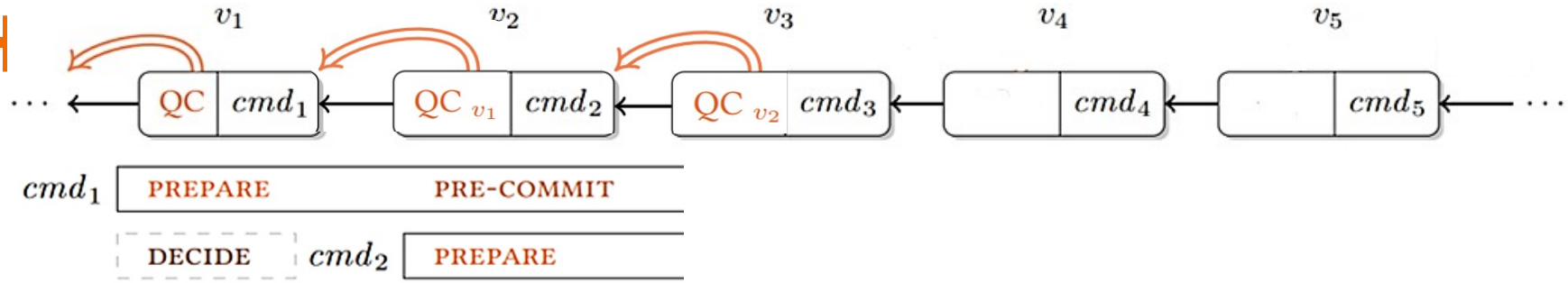


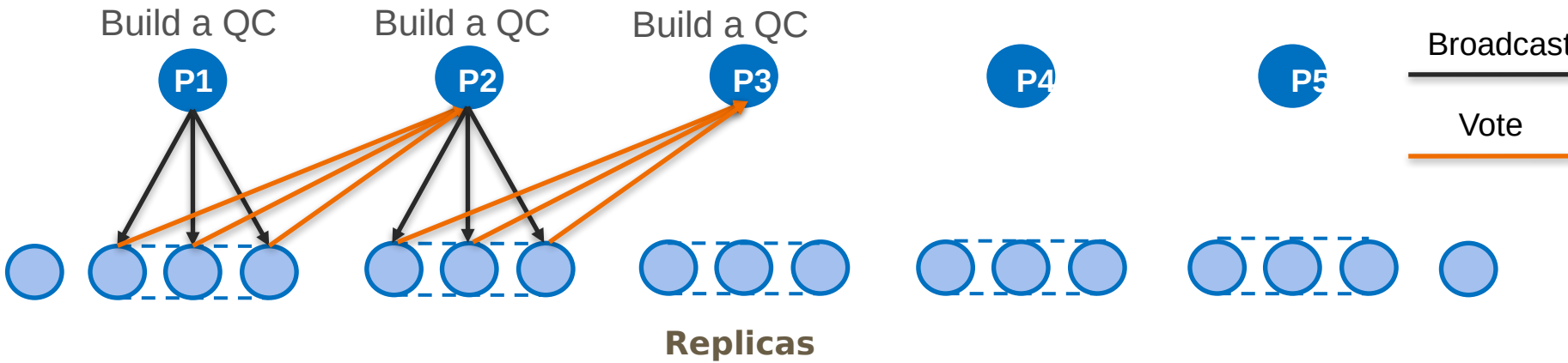
Chained HotStuff- Dive into chained H



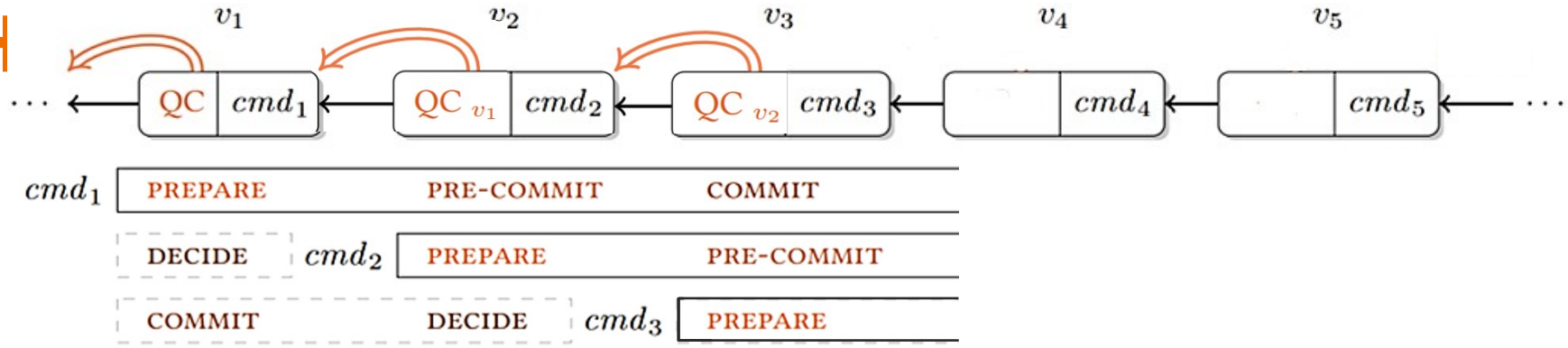


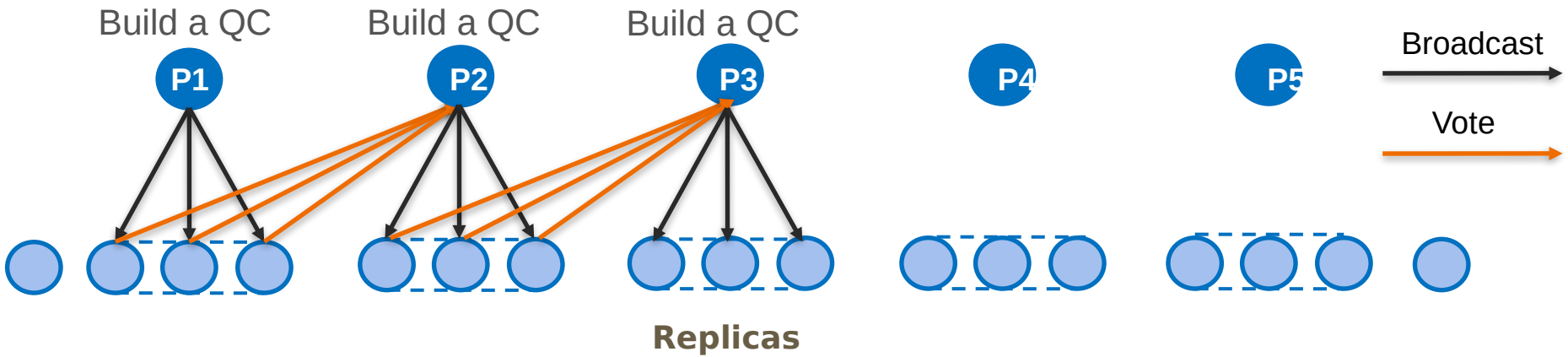
Chained HotStuff- Dive into chained H



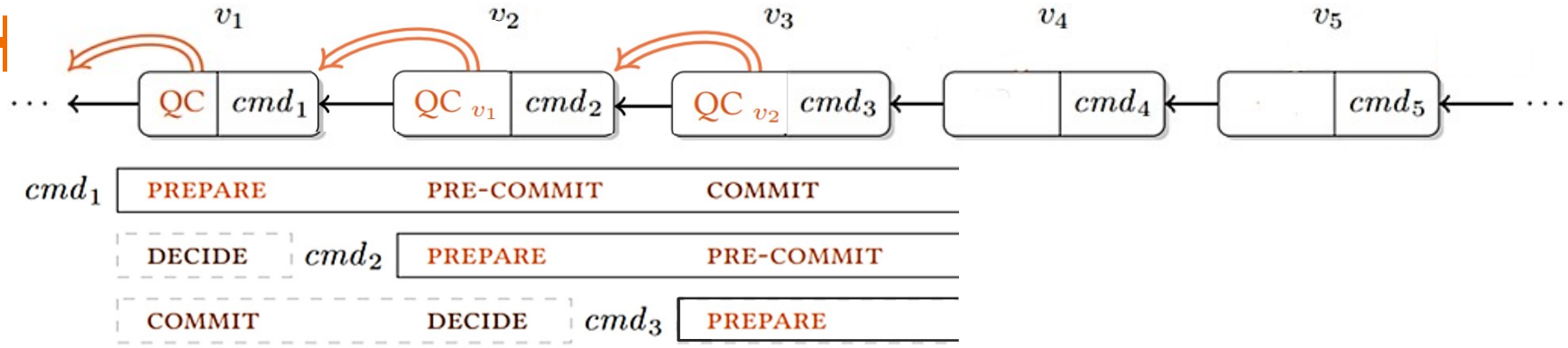


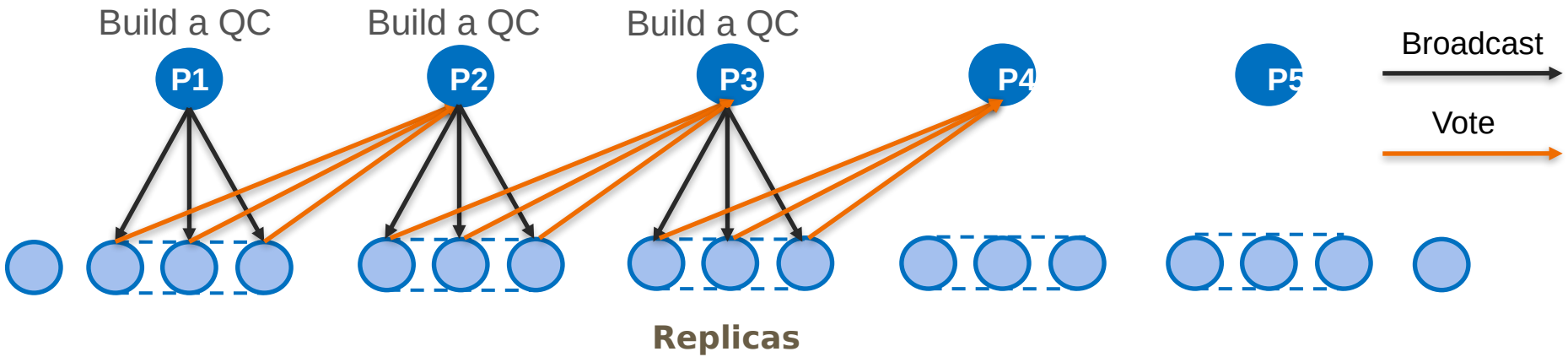
Chained HotStuff- Dive into chained H



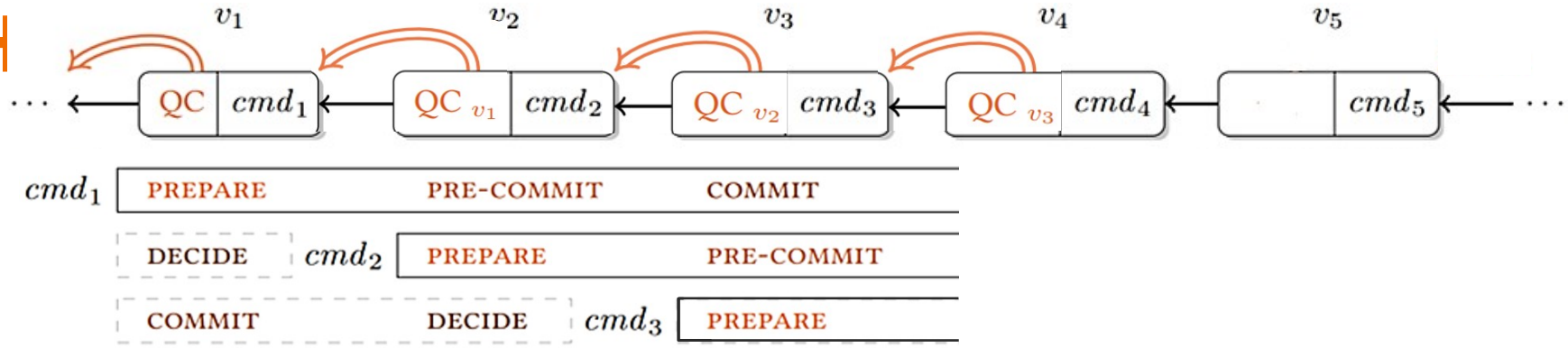


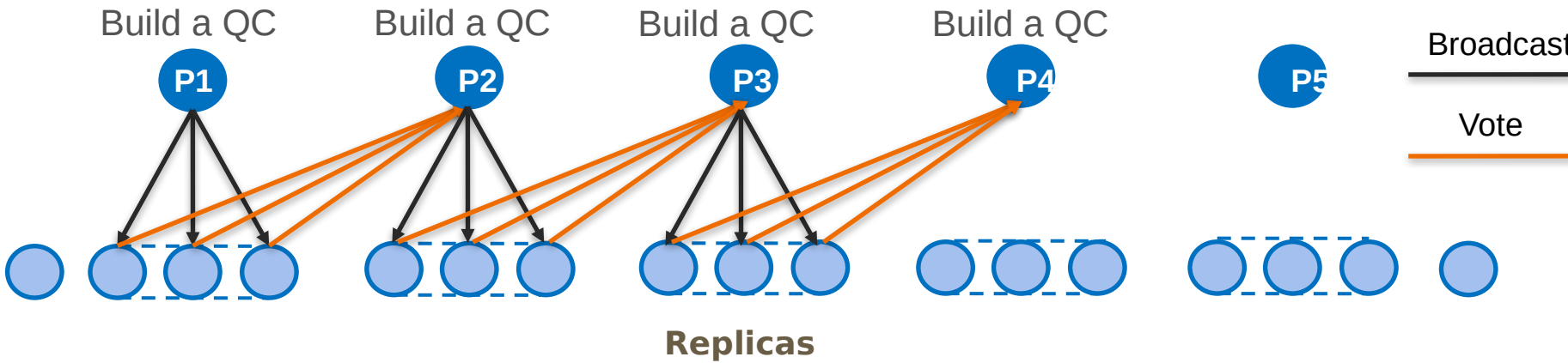
Chained HotStuff- Dive into chained H



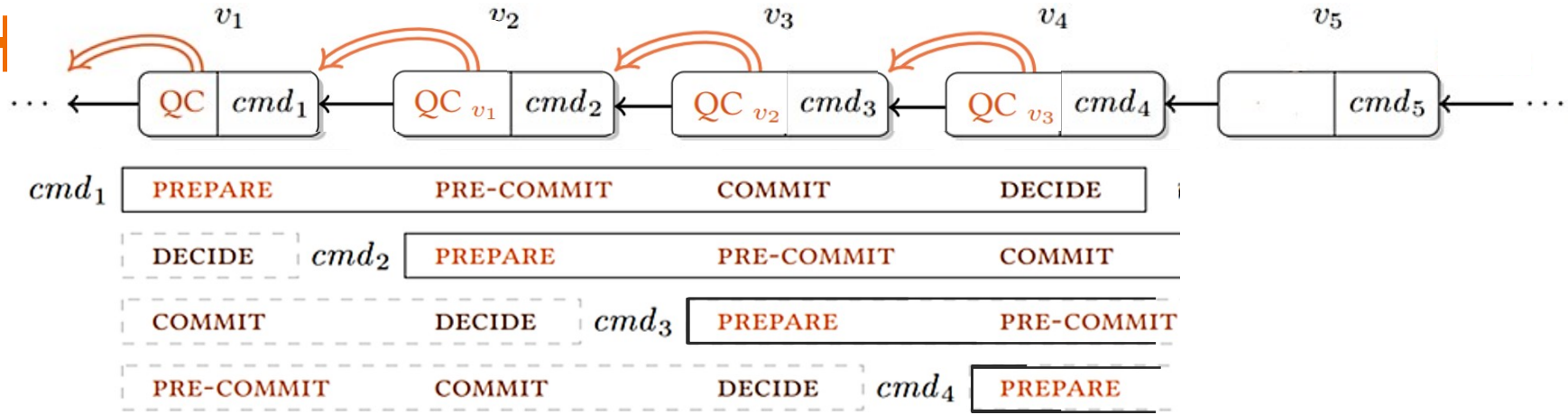


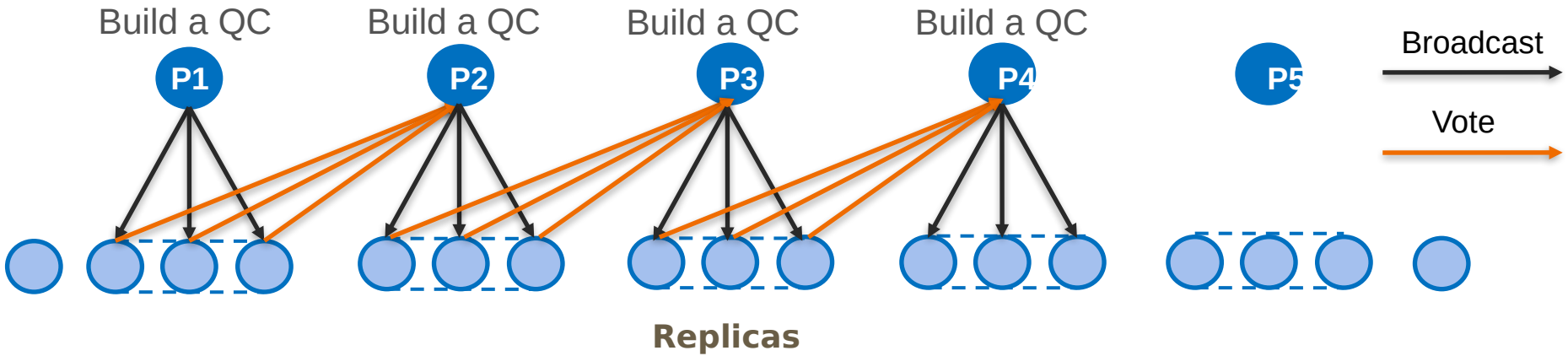
Chained HotStuff- Dive into chained H



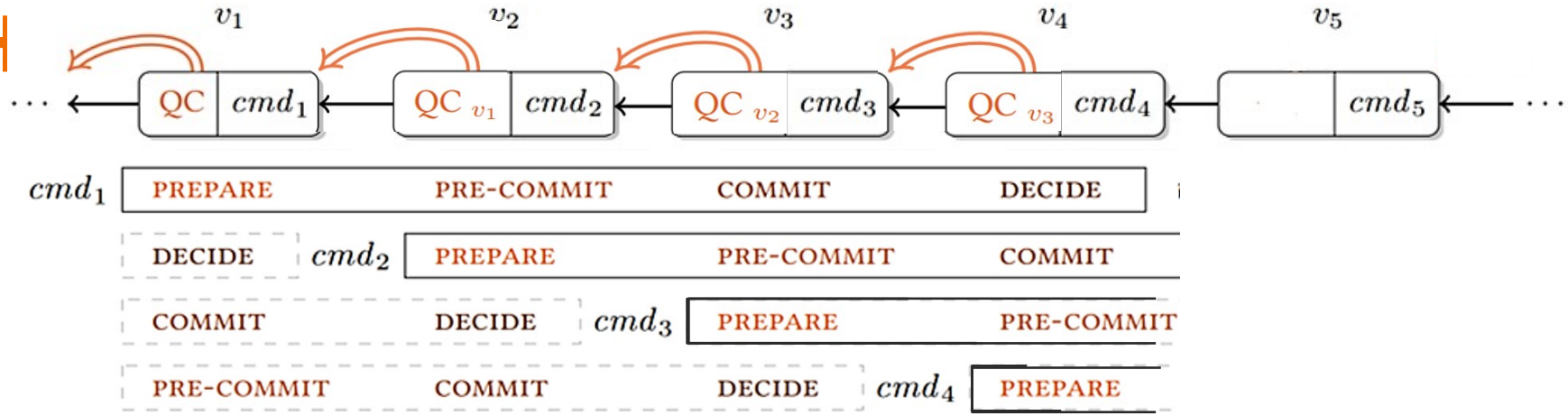


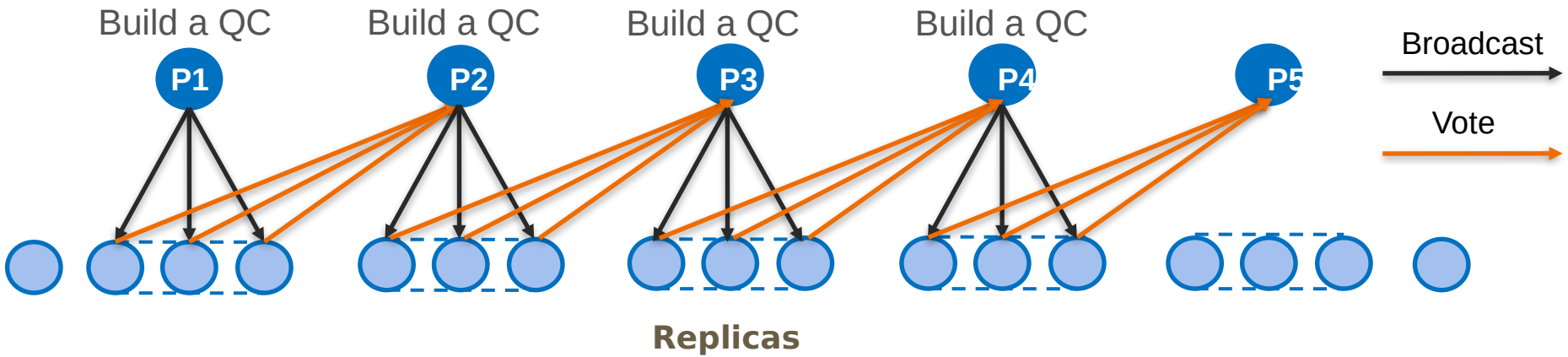
Chained HotStuff- Dive into chained H



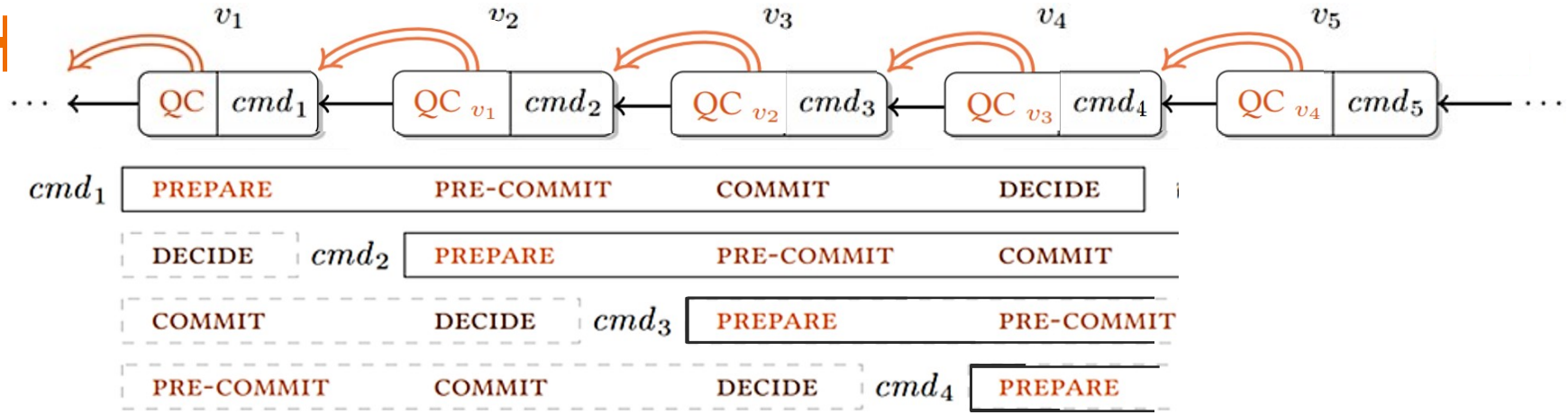


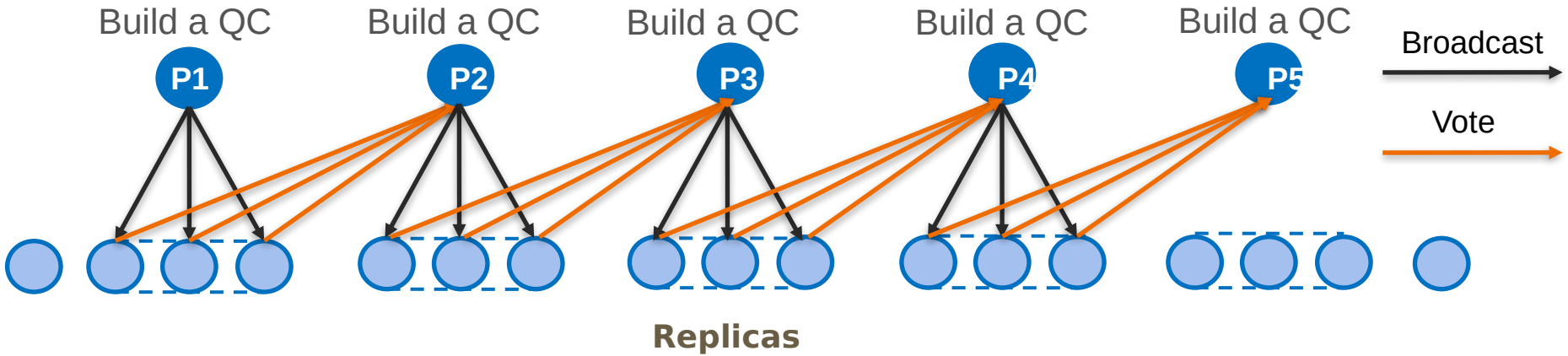
Chained HotStuff- Dive into chained H



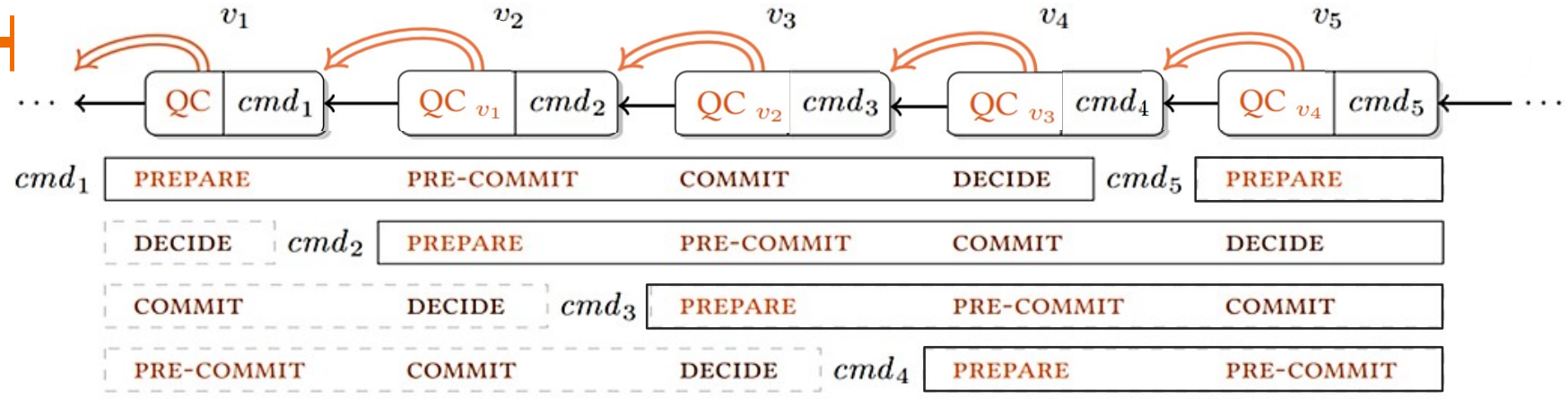


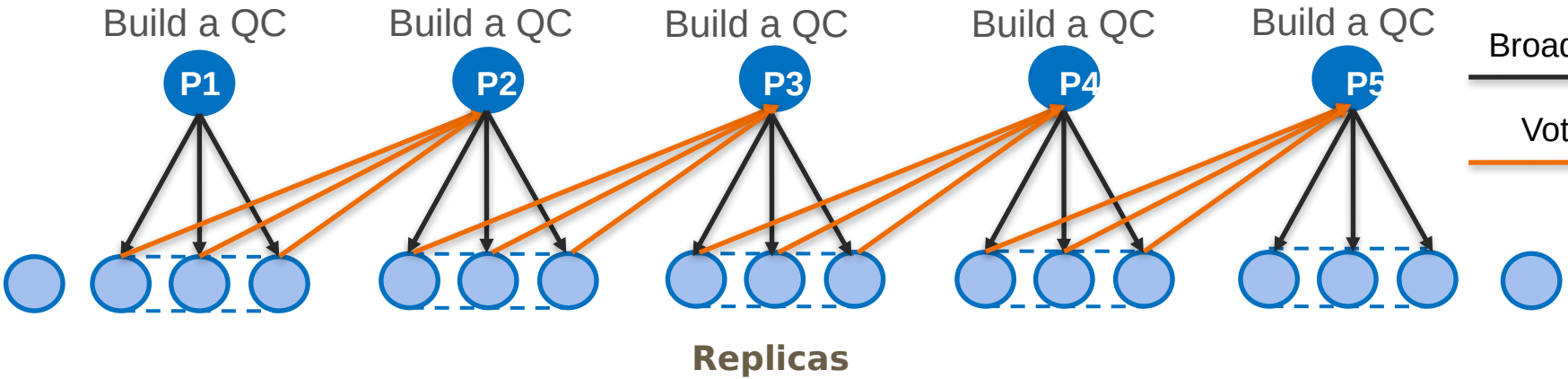
Chained HotStuff- Dive into chained H





Chained HotStuff- Dive into chained H





Chained HotStuff – Pseudo code

Algorithm 3 Chained HotStuff protocol.

▷ GENERIC phase

as a leader // $r = \text{LEADER}(curView)$

// M is the set of messages collected at the end of previous view by the leader of this view

$highQC \leftarrow \left(\arg\max_{m \in M} \{m.justify.viewNumber\} \right).justify$

if $highQC.viewNumber > genericQC.viewNumber$ **then** $genericQC \leftarrow highQC$

$curProposal \leftarrow \text{CREATELEAF}(genericQC.node, \text{client's command}, genericQC)$

// PREPARE phase

broadcast $\text{MSG}(\text{GENERIC}, curProposal, \perp)$

as a replica

wait for message $m : \text{MATCHINGMSG}(m, \text{GENERIC}, curView)$ from $\text{LEADER}(curView)$

$b^* \leftarrow m.node; b'' \leftarrow b^*.justify.node; b' \leftarrow b''.justify.node; b \leftarrow b'.justify.node$

if $\text{SAFENODE}(b^*, b^*.justify)$ **then**

send $\text{VOTEMSG}(\text{GENERIC}, b^*, \perp)$ to $\text{LEADER}(curView + 1)$

// start PRE-COMMIT phase on b^*'s parent

if $b^*.parent = b''$ **then**

$genericQC \leftarrow b^*.justify$

// start COMMIT phase on b^*'s grandparent

if $(b^*.parent = b'') \wedge (b''.parent = b')$ **then**

$lockedQC \leftarrow b''.justify$

// start DECIDE phase on b^*'s great-grandparent

if $(b^*.parent = b'') \wedge (b''.parent = b') \wedge (b'.parent = b)$ **then**

execute new commands through b, respond to clients

as the next leader

wait for all messages: $M \leftarrow \{m \mid \text{MATCHINGMSG}(m, \text{GENERIC}, curView)\}$

until there are $(n - f)$ votes: $V \leftarrow \{v \mid v.partialSig \neq \perp \wedge v \in M\}$

$genericQC \leftarrow \text{QC}(V)$

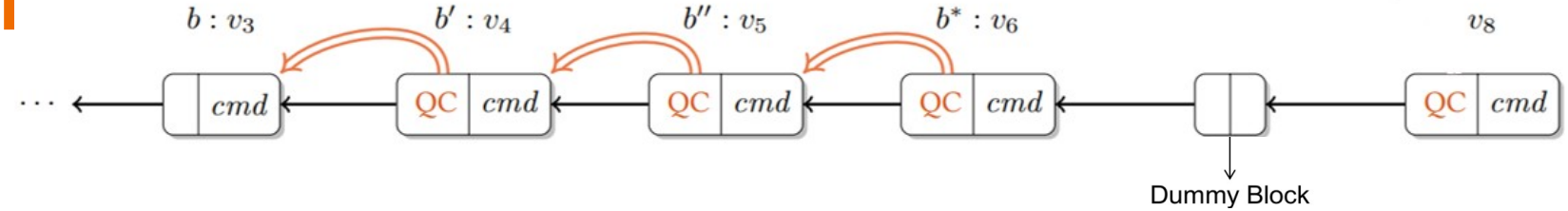
Leader/Primary:

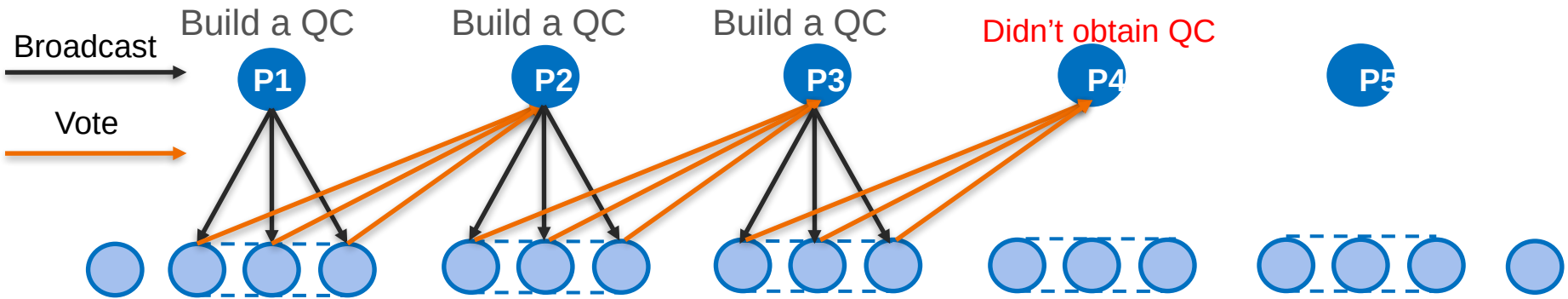
- Waiting for new view messages, sending his own proposal
- Waiting for other nodes' voting
- Sending new view message to next primary node

Replica

- Waiting for proposal
- Sending vote to next primary node
- Checking QC, updating highQC and lockedQC

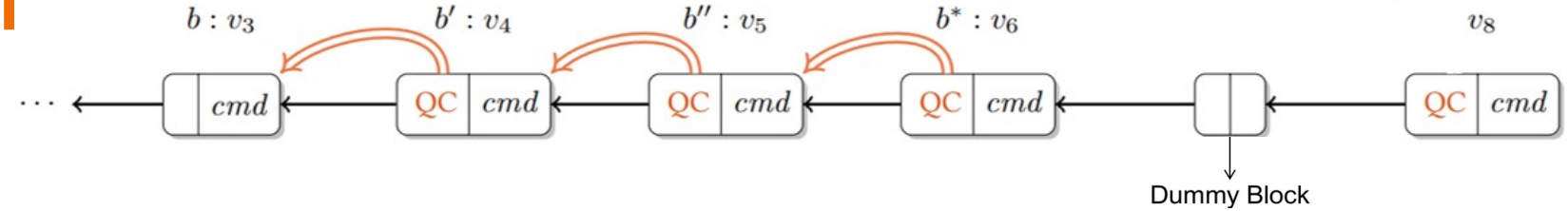
Chained HotStuff -What if one view di

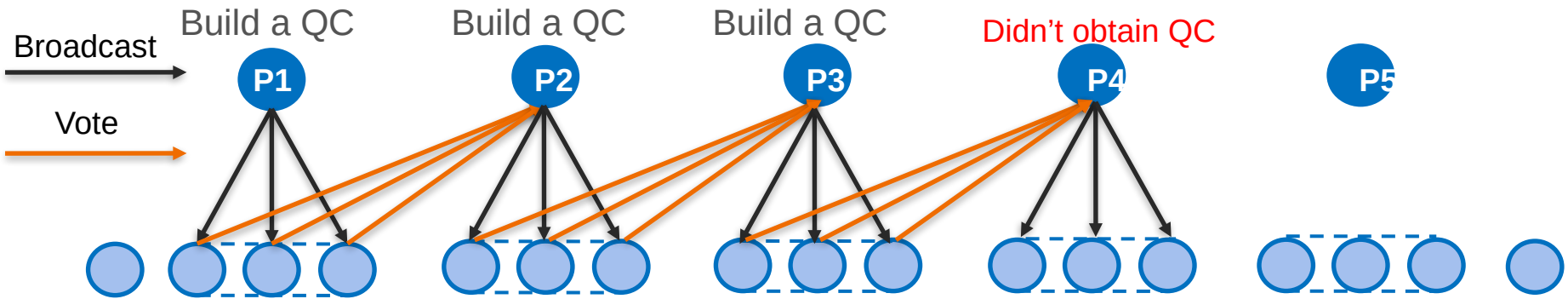




The genericQC used by a primary in some view viewNumber may not directly reference the proposal of the preceding view (viewNumber −1). The reason is that the primary of a preceding view fails to obtain a QC, either because there are conicting proposals, or due to a benign crash

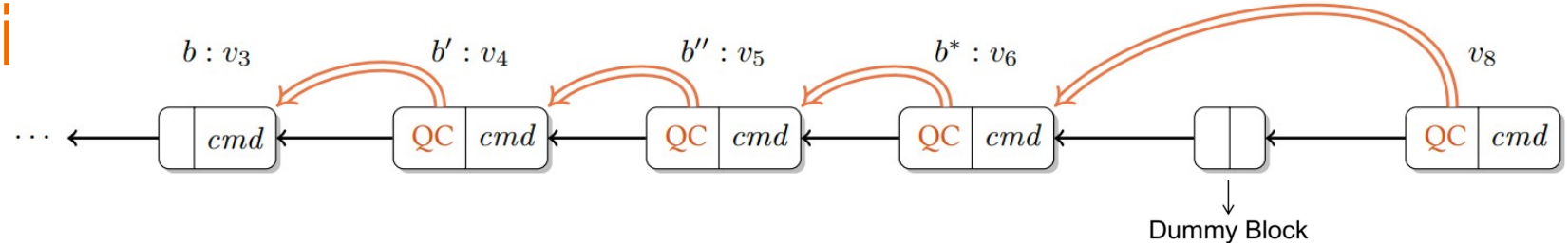
Chained HotStuff -What if one view di

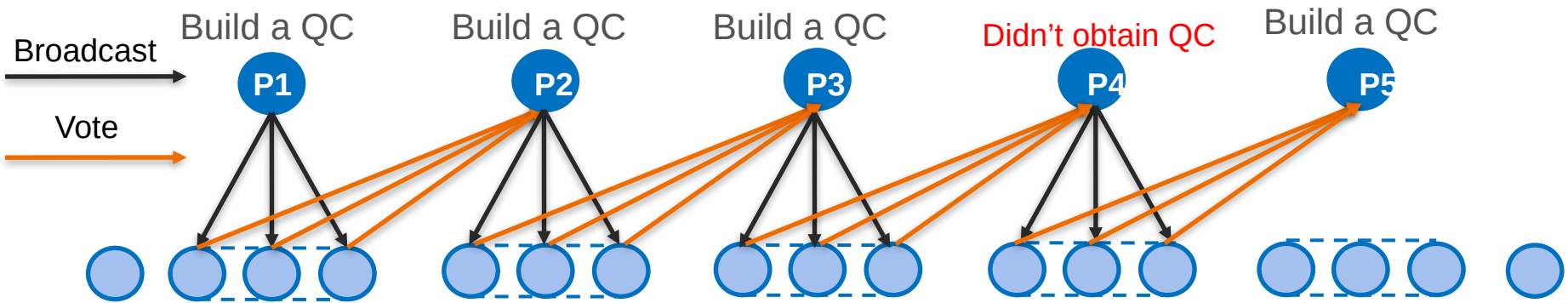




The genericQC used by a primary in some view viewNumber may not directly reference the proposal of the preceding view (viewNumber −1). The reason is that the primary of a preceding view fails to obtain a QC, either because there are conicting proposals, or due to a benign crash

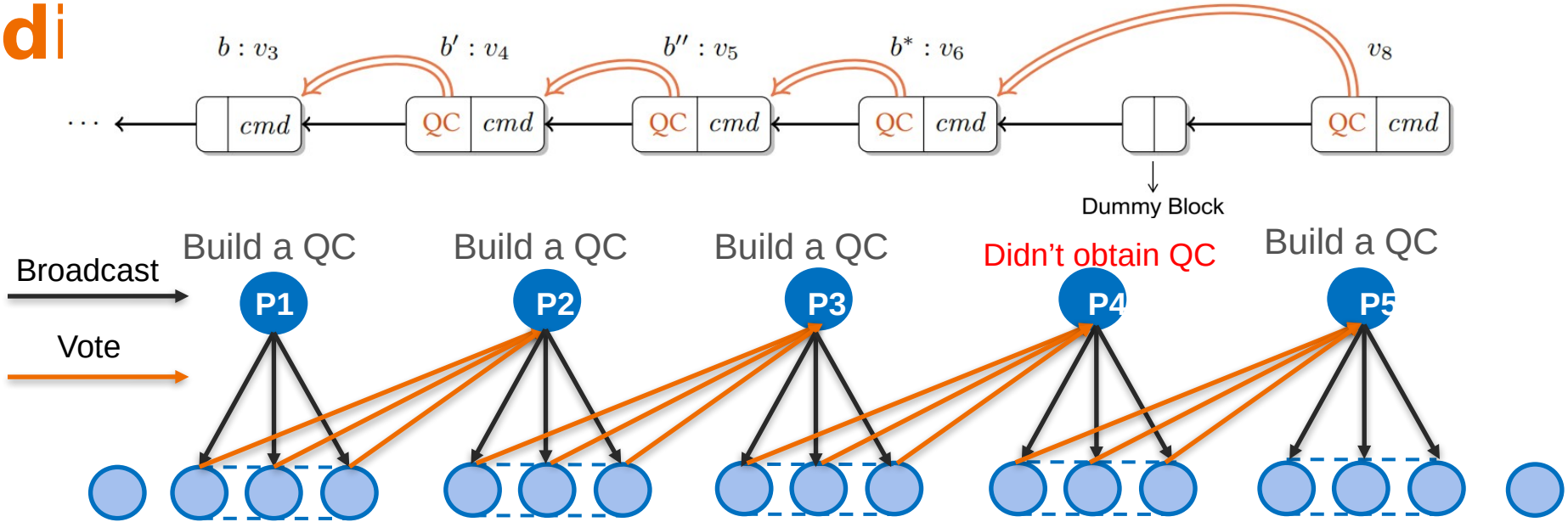
Chained HotStuff -What if one view di





The genericQC used by a primary in some view viewNumber may not directly reference the proposal of the preceding view (viewNumber −1). The reason is that the primary of a preceding view fails to obtain a QC, either because there are conicting proposals, or due to a benign crash

Chained HotStuff -What if one view di



The genericQC used by a primary in some view viewNumber may not directly reference the proposal of the preceding view (viewNumber −1). The reason is that the primary of a preceding view fails to obtain a QC, either because there are conicting proposals, or due to a benign crash

Reference

- Yin M, Malkhi D, Reiter M K, et al. HotStuff: BFT consensus in the lens of blockchain[J]. arXiv preprint arXiv:1803.05069, 2018.

- https://seafooler.com/2022/01/24/understanding-safety-hotstuff/

- https://www.youtube.com/watch?v=AVGD_AWf47g&t=332s

- https://www.youtube.com/watch?v=mEWrrdP4kGE

Thanks for Your Listening

Q&A