

Back to the protocol...

- To broadcast a message in round r , p sends $(init, p, m, r)$ to all
- A confirmation has the form $(echo, p, m, r)$
- A witness sends $(echo, p, m, r)$ if either:
 - it receives $(init, p, m, r)$ from p directly or
 - it receives confirmations for (p, m, r) from at least $f + 1$ processes (at least one correct witness)
- A process accepts (p, m, r) if it has received $n - f$ confirmations (as many as possible...)
- Protocol proceeds in **rounds**. Each round has 2 **phases**

Implementation of broadcast and accept

Phase $2r-1$

1: p sends $(init, p, m, r)$ to all

Phase $2r$

2: if q received $(init, p, m, r)$ in phase $2r-1$ then

3: q sends $(echo, p, m, r)$ to all /* q becomes a witness */

4: if q receives $(echo, p, m, r)$ from at least $n-f$ distinct processes in phase $2r$ then

5: q accepts (p, m, r)

Phase $j > 2r$

6: if q has received $(echo, p, m, r)$ from at least $f+1$ distinct processes in phases $(2r, 2r+1, \dots, j-1)$ then

7: q sends $(echo, p, m, r)$ to all processes /* q becomes a witness */

8: if q has received $(echo, p, m, r)$ from at least $n-f$ processes in phases $(2r, 2r+1, \dots, j)$ then

9: q accepts (p, m, r)

Is termination a problem?

The implementation is correct

Theorem

If $n > 3f$, the given implementation of $\text{broadcast}(p, m, r)$ and $\text{accept}(p, m, r)$ satisfies Unforgeability, Correctness, and Relay

Assumption

Channels are reliable (between correct processes) and authenticated

Correctness

If a correct process p
executes $\text{broadcast}(p, m, r)$
in round r , then all
correct processes will
execute $\text{accept}(p, m, r)$ in
round r

Correctness

If a correct process p executes $\text{broadcast}(p, m, r)$ in round r , then all correct processes will execute $\text{accept}(p, m, r)$ in round r

If p is correct then

- p sends (init, p, m, r) to all in round r (phase $2r-1$)
- by Validity of the underlying send and receive, every correct process receives (init, p, m, r) in phase $2r-1$
- every correct process becomes a witness
- every correct process sends (echo, p, m, r) in phase $2r$
- since there are at least $n-f$ correct processes, every correct process receives at least $n-f$ echoes in phase $2r$
- every correct process executes $\text{accept}(p, m, r)$ in phase $2r$ (in round r)

Unforgeability

If a correct process q executes $\text{accept}(p, m, r)$ in round $j \geq r$, and p is correct, then p did in fact execute $\text{broadcast}(p, m, r)$ in round r

- Suppose q executes $\text{accept}(p, m, r)$ in round j
- q received (echo, p, m, r) from at least $n - f$ distinct processes by phase k , where $k = 2j - 1$ or $k = 2j$
- Let k' be the earliest phase in which some correct process q' becomes a witness to (p, m, r)

Unforgeability

If a correct process q executes $\text{accept}(p, m, r)$ in round $j \geq r$, and p is correct, then p did in fact execute $\text{broadcast}(p, m, r)$ in round r

- Suppose q executes $\text{accept}(p, m, r)$ in round j
- q received (echo, p, m, r) from at least $n - f$ distinct processes by phase k , where $k = 2j - 1$ or $k = 2j$
- Let k' be the earliest phase in which some correct process q' becomes a witness to (p, m, r)

Case 1: $k' = 2r - 1$

- q' received (init, p, m, r) from p
- since p is correct, it follows that p did execute $\text{broadcast}(p, m, r)$ in round r

Unforgeability

If a correct process q executes $\text{accept}(p, m, r)$ in round $j \geq r$, and p is correct, then p did in fact execute $\text{broadcast}(p, m, r)$ in round r

- Suppose q executes $\text{accept}(p, m, r)$ in round j
- q received (echo, p, m, r) from at least $n - f$ distinct processes by phase k , where $k = 2j - 1$ or $k = 2j$
- Let k' be the earliest phase in which some correct process q' becomes a witness to (p, m, r)

Case 1: $k' = 2r - 1$

- q' received (init, p, m, r) from p
- since p is correct, it follows that p did execute $\text{broadcast}(p, m, r)$ in round r

Case 2: $k' > 2r - 1$

- q' has become a witness by receiving (echo, p, m, r) from $f + 1$ distinct processes
- at most f are faulty; one is correct
- this process was a witness to (p, m, r) before phase k'

CONTRADICTION

The first correct witness receives (init, p, m, r) from p !

Summing up...

- 👁 For q to accept, some correct process must become a witness.
- 👁 Earliest correct witness q' becomes so in phase $2r - 1$, and only if p did indeed executed broadcast (p, m, r)
- 👁 Any correct process that becomes a witness later can only do so if a correct process is already a witness.
- 👁 For any correct process to become a witness of a broadcast of some correct p , p must have executed broadcast (p, m, r)

Relay

If a correct process q executes $\text{accept}(p, m, r)$ in round $j \geq r$, then all correct processes will execute $\text{accept}(p, m, r)$ by round $j + 1$

Relay

If a correct process q executes $\text{accept}(p, m, r)$ in round $j \geq r$, then all correct processes will execute $\text{accept}(p, m, r)$ by round $j + 1$

- Suppose correct q executes $\text{accept}(p, m, r)$ in round j (phase $k = 2j - 1$ or $k = 2j$)
- q received at least $n - f$ (echo, p, m, r) from distinct processes by phase k
- At least $n - 2f$ of them are correct.
- All correct processes receive (echo, p, m, r) from at least $n - 2f$ correct processes by phase k
- From $n > 3f$, it follows that $n - 2f \geq f + 1$. Then, all correct processes become witnesses by phase k
- All correct processes send (echo, p, m, r) by phase $k + 1$
- Since there are at least $n - f$ correct processes, all correct processes will $\text{accept}(p, m, r)$ by phase $k + 1$ (round $2j$ or $2j + 1$)

What about asynchronous systems?

- 👁 FLP says consensus cannot be solved...
- 👁 For benign failures, Paxos provides next best thing
 - ❑ always safe
 - ❑ live during periods of sufficient synchrony
- 👁 And for Byzantine failures?

PBFT:

A Byzantine Renaissance

- 👁 Practical Byzantine Fault-Tolerance (CL99, CL00)
 - ❑ first to be **safe** in asynchronous systems
 - ❑ live under weak synchrony assumptions—Byzantine Paxos!
 - ❑ **fast!** PBFT uses MACs instead of public key cryptography
 - ❑ uses **proactive recovery** to tolerate more failures over system lifetime: now need no more than f failures in a “window of time”
- 👁 BASE (RCL 01)
 - ❑ uses **abstraction** to reduce correlated faults

The Setup

System Model

- Asynchronous system
- Unreliable channels

Crypto

- Public/Private key pairs
- MACs
- Collision-resistant hashes
- Unbreakable

Service

- Byzantine clients
- Up to f Byzantine servers
- $N > 3f$ total servers

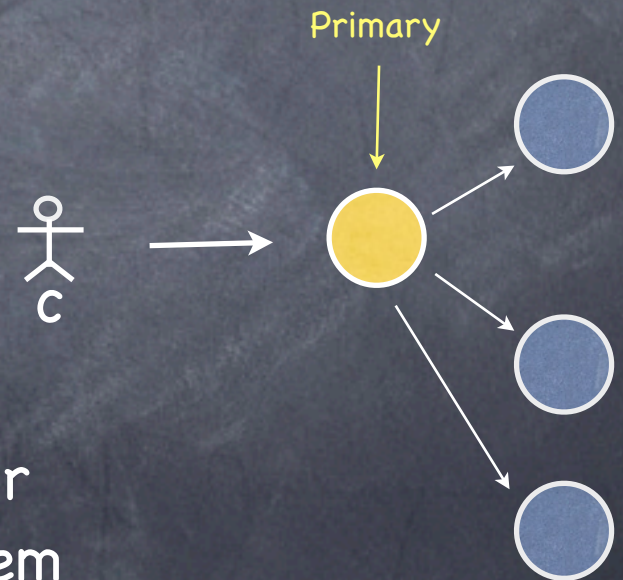
System Goals

- Always safe
- Live during periods of synchrony

The General Idea

👁️ Primary-backup + quorum system

- ❑ executions are sequences of **views**
- ❑ clients send signed commands to primary of current view
- ❑ primary assigns sequence number to client's command
- ❑ primary writes $\langle \text{sequence number, command} \rangle$ to the register implemented by the quorum system defined by all the servers (primary included)



What could possibly go wrong? 😊

👁️ The Primary could be faulty!

- ▶️ could ignore commands; assign same sequence number to different requests; skip sequence numbers; etc
- ❑ Backups monitor primary's behavior and trigger **view changes** to replace faulty primary

👁️ Backups could be faulty!

- ▶️ could incorrectly store commands forwarded by a correct primary
- ❑ use **dissemination Byzantine quorum systems** [MR98]
 - ▶️ can detect data tampering

👁️ Faulty replicas could incorrectly respond to the client!

What could possibly go wrong? 😊

👁️ The Primary could be faulty!

- ▶ could ignore commands; assign same sequence number to different requests; skip sequence numbers; etc
- ❑ Backups monitor primary's behavior and trigger **view changes** to replace faulty primary

👁️ Backups could be faulty!

- > could incorrectly store commands forwarded by a correct primary
- ❑ use **dissemination Byzantine quorum systems** [MR98]
 - ▶ can detect data tampering

👁️ Faulty replicas could incorrectly respond to the client!

- ❑ Client waits for $f+1$ matching replies before accepting response

Me, or your lying eyes?

- 👁️ Algorithm steps are justified by **certificates**
 - ❑ Sets (quorums) of signed messages from distinct replicas proving that a property of interest holds
- 👁️ With quorums of size at least $2f+1$
 - ❑ Any two quorums intersect in at least one correct replica
 - ❑ Always one quorum contains only correct replicas

PBFT: The site map

👁 Normal operation

- ❑ How the protocol works in the absence of failures – hopefully, the common case

👁 View changes

- ❑ How to depose a faulty primary and elect a new one

👁 Garbage collection

- ❑ How to reclaim the storage used to keep certificates

👁 Recovery

- ❑ How to make a faulty replica behave correctly again

Normal Operation

☉ Three phases:

- ☐ **Pre-prepare** assigns sequence number to request
- ☐ **Prepare** ensures fault-tolerant consistent ordering of requests within views
- ☐ **Commit** ensures fault-tolerant consistent ordering of requests across views

☉ Each replica i maintains the following state:

- ☐ Service state
- ☐ A message log with all messages sent or received
- ☐ An integer representing i 's current view

Client issues request



Client issues request



Client issues request



Client issues request

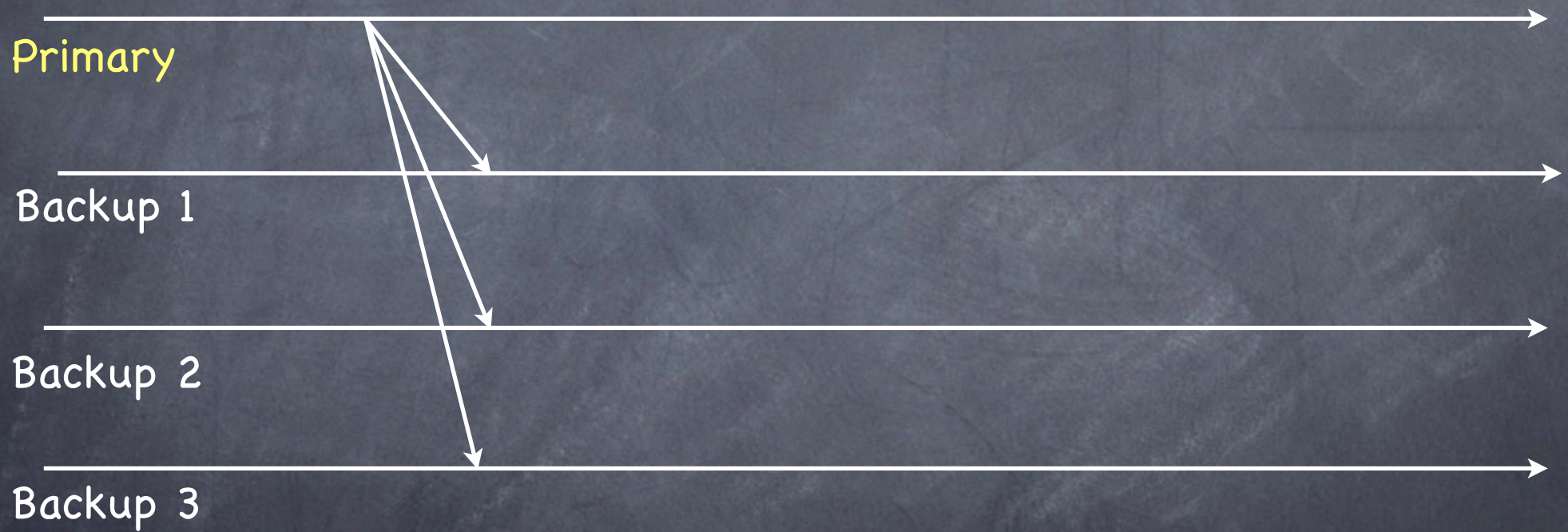


Client issues request

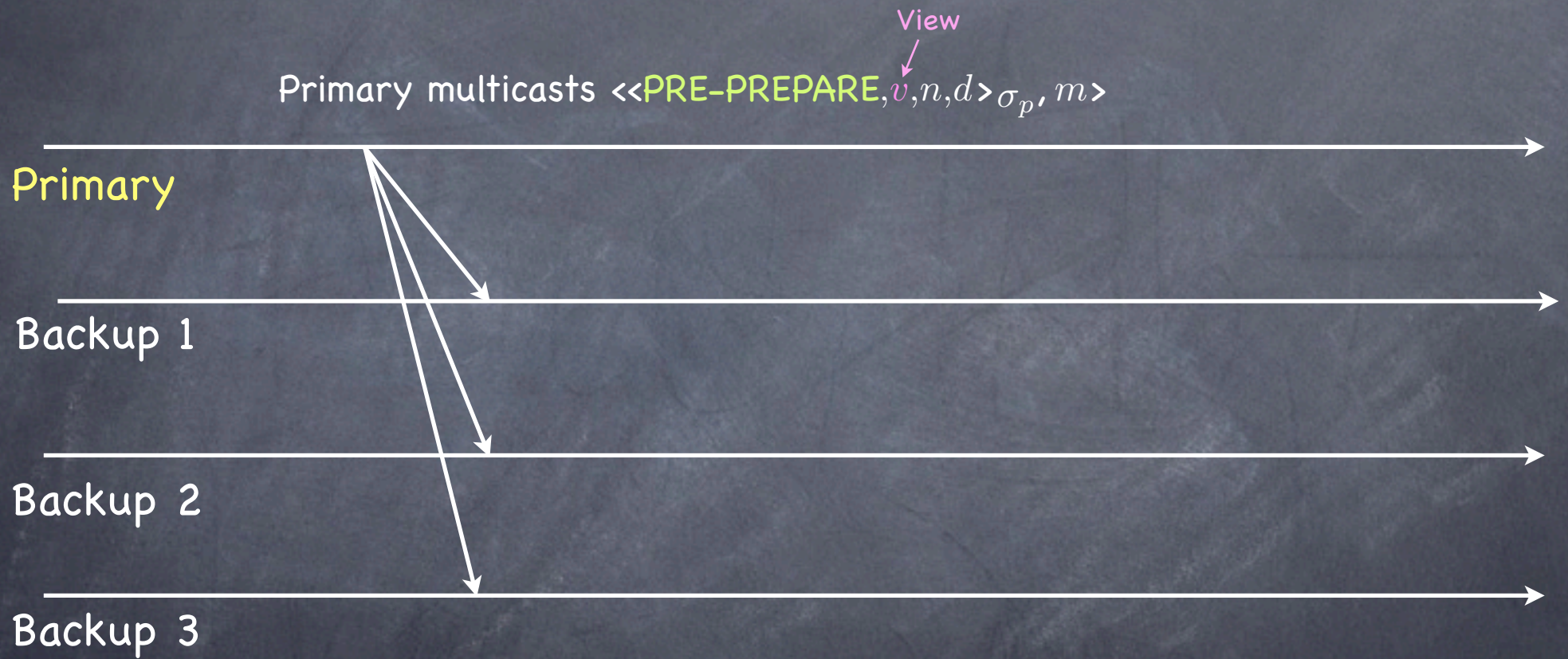


Pre-prepare

Primary multicasts $\langle\langle\text{PRE-PREPARE}, v, n, d\rangle_{\sigma_p}, m\rangle$



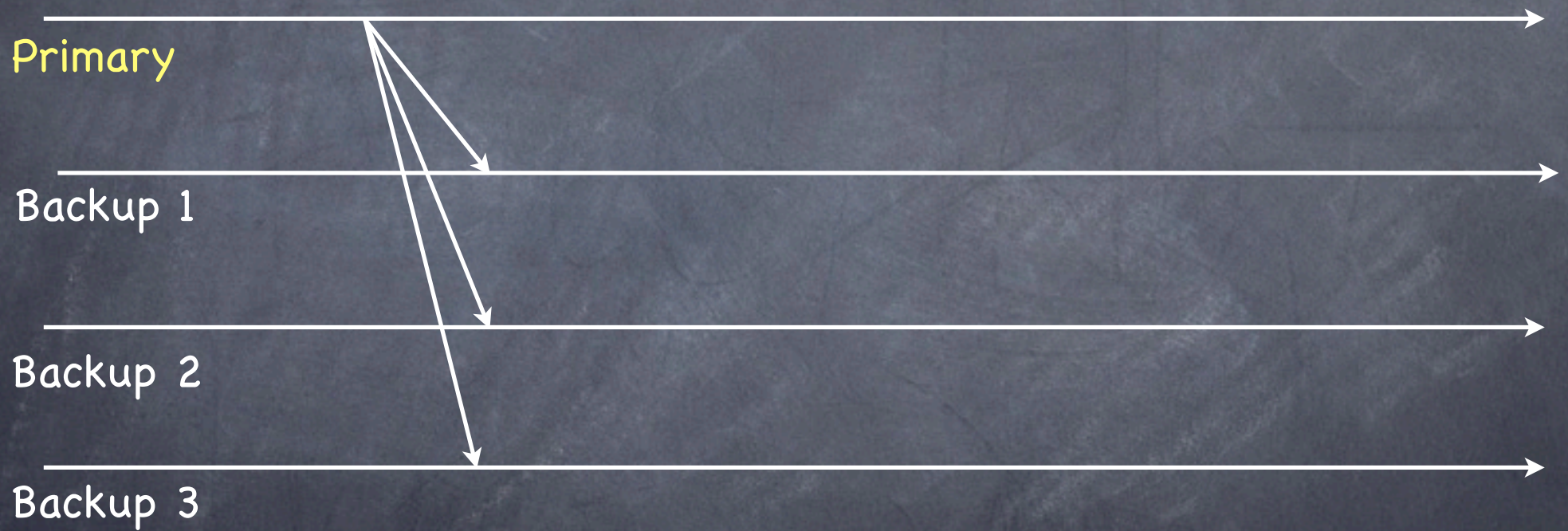
Pre-prepare



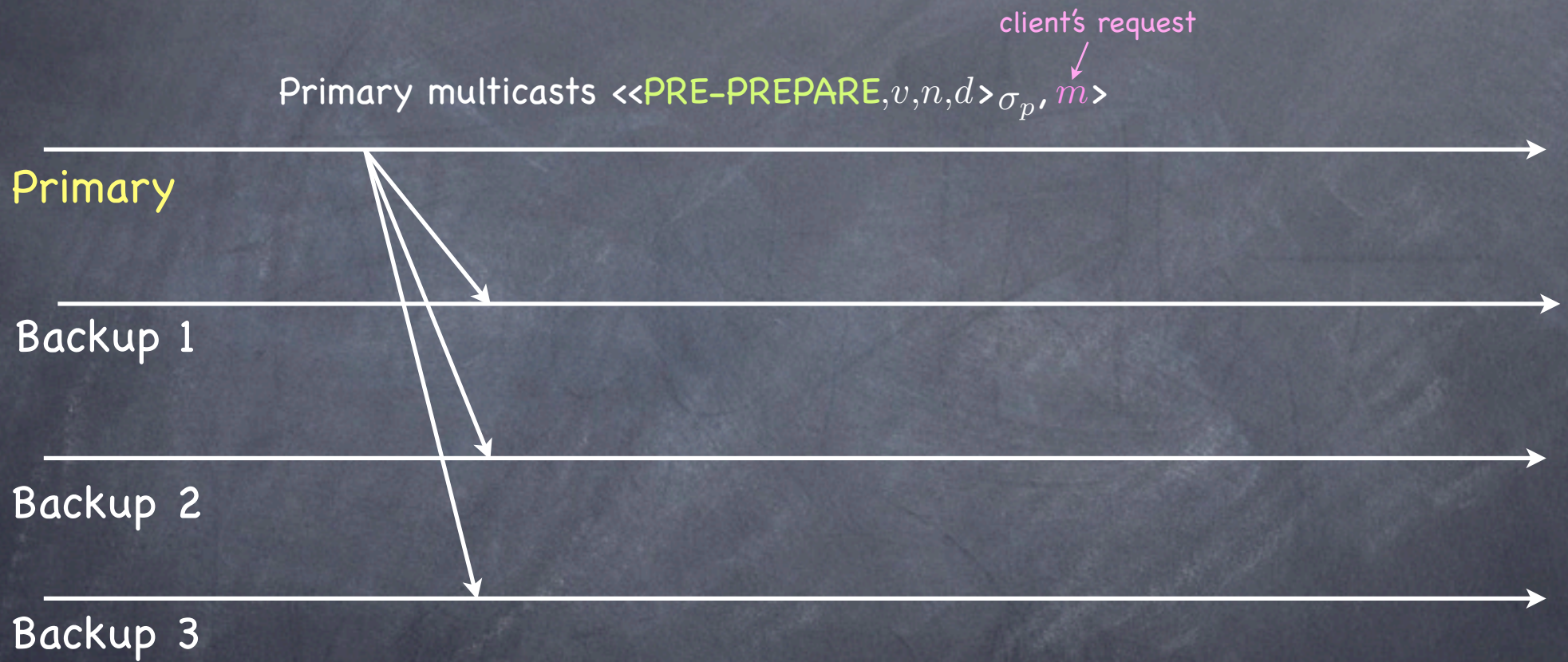
Pre-prepare

Sequence number

Primary multicasts $\langle\langle\text{PRE-PREPARE}, v, n, d\rangle_{\sigma_p}, m\rangle$



Pre-prepare



Pre-prepare

Primary multicasts $\langle\langle \text{PRE-PREPARE}, v, n, d \rangle_{\sigma_p}, m \rangle$

digest of m

Primary

Backup 1

Backup 2

Backup 3



Pre-prepare

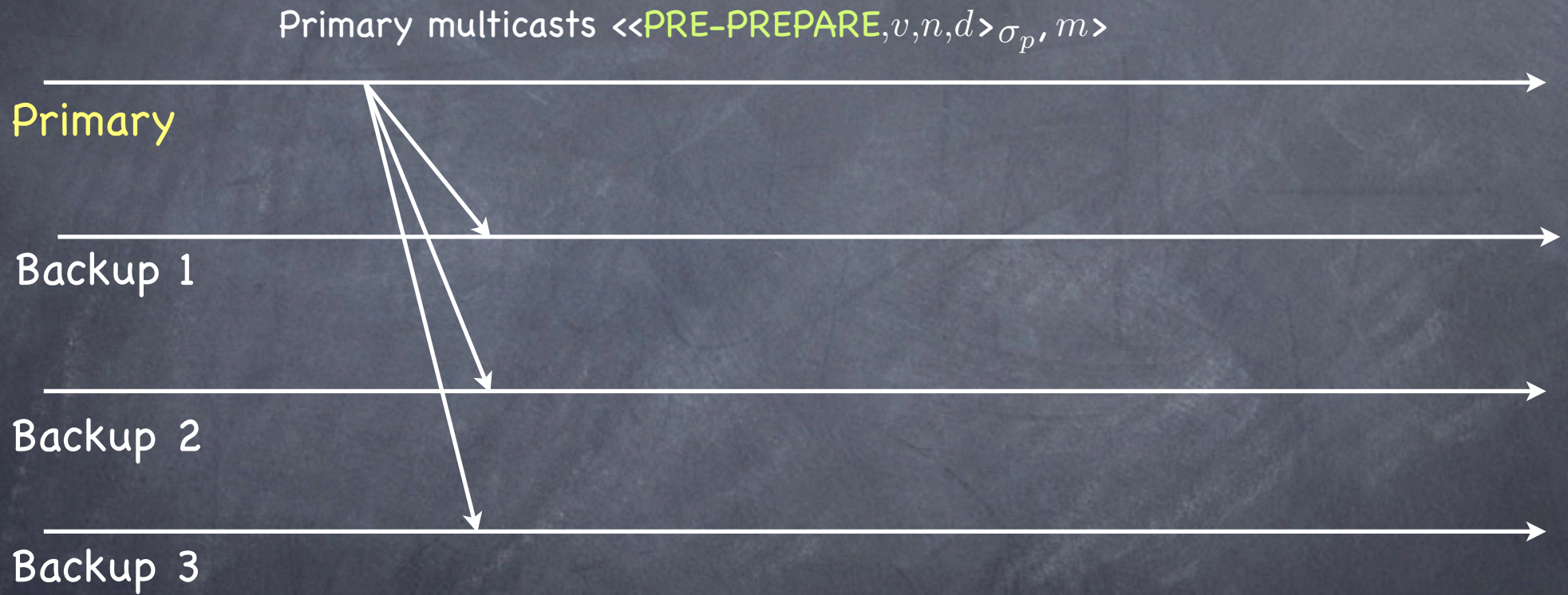
Primary multicasts $\langle\langle \text{PRE-PREPARE}, v, n, d \rangle_{\sigma_p}, m \rangle$



Correct backup i
accepts
PRE-PREPARE if:

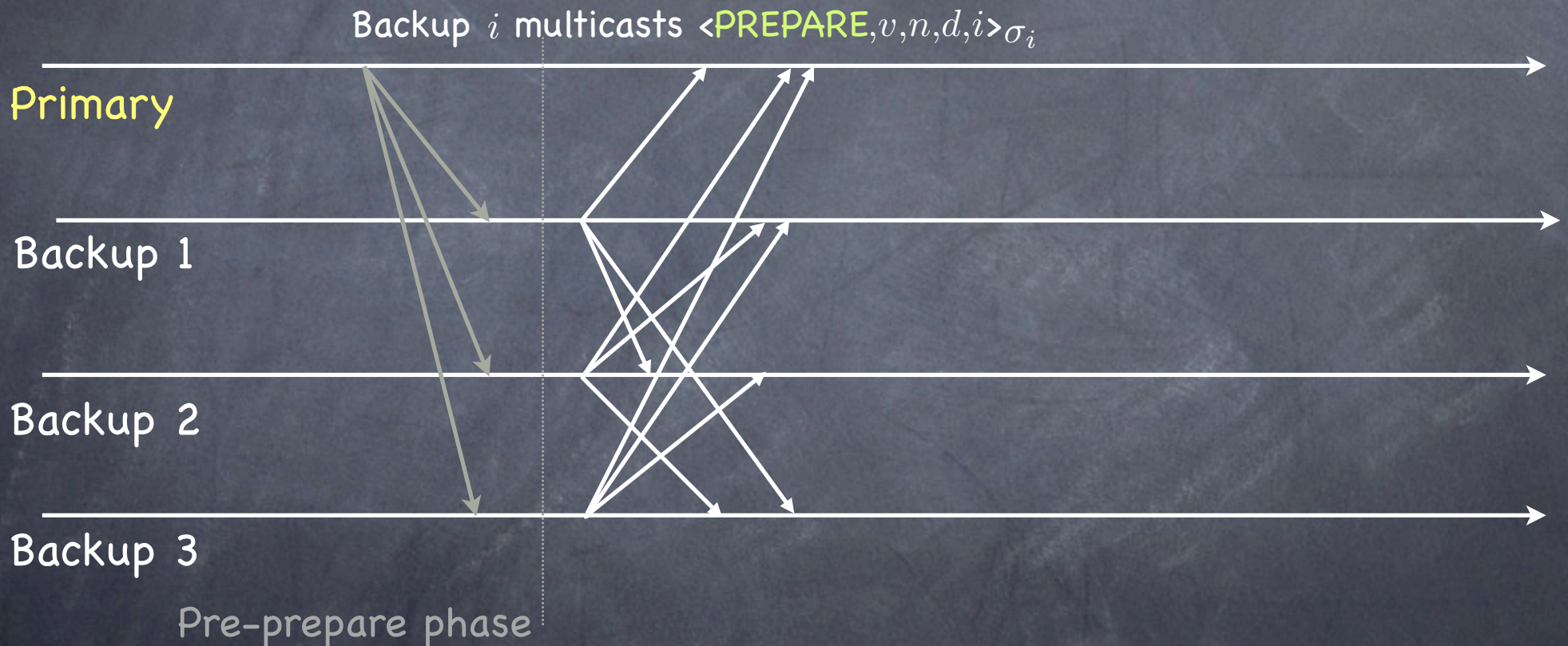
- PRE-PREPARE is well formed
- i is in view v
- i has not accepted another PRE-PREPARE for v, n with a different d
- n is between two water-marks L and H (to prevent sequence number exhaustion)

Pre-prepare



Each accepted PRE-PREPARE message is stored in the accepting replica's message log (including the Primary's)

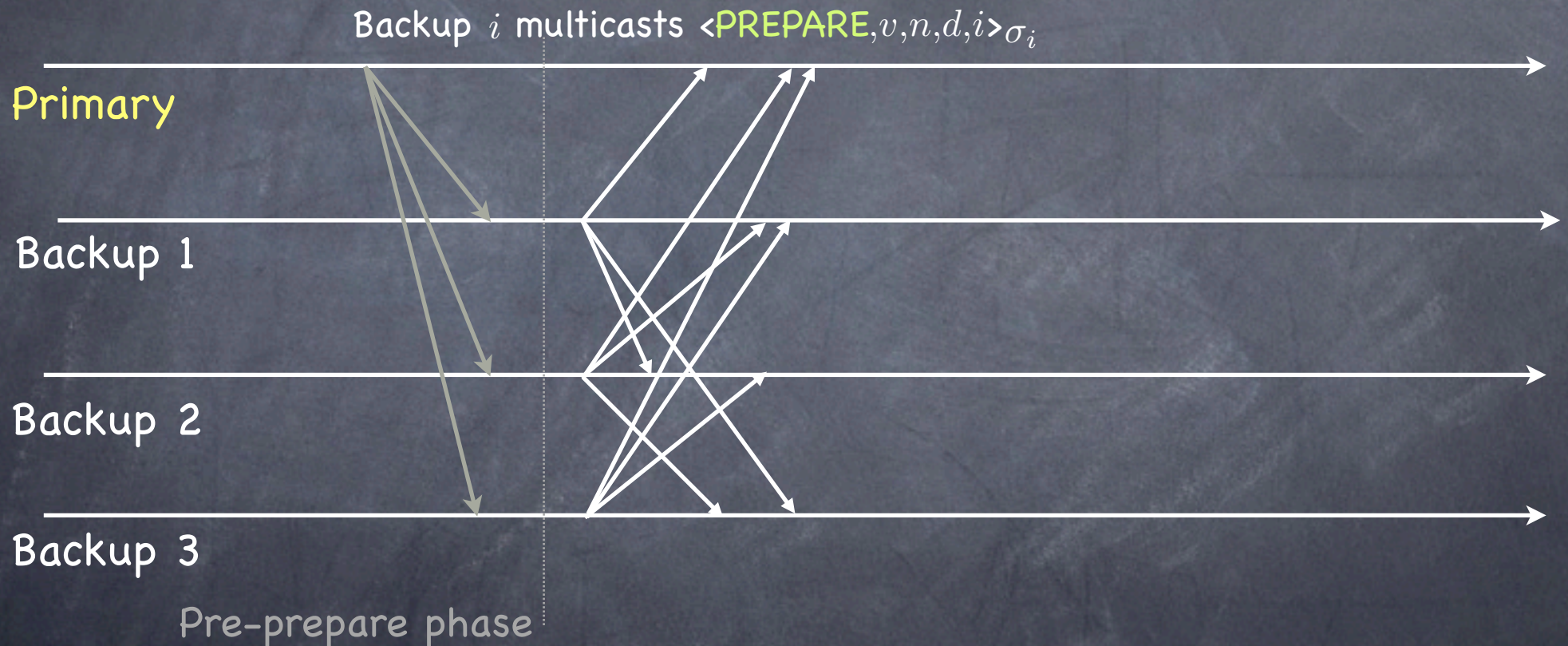
Prepare



Correct replica i
accepts **PREPARE** if:

- PREPARE is well formed
- i is in view v
- n is between two water-marks L and H

Prepare



- Replicas that send **PREPARE** accept seq.# n for m in view v
- Each accepted **PREPARE** message is stored in the accepting replica's message log

Prepare Certificate

- P-certificates ensure total order within views

Prepare Certificate

- 👁 **P-certificates** ensure total order within views
- 👁 Replica produces $P\text{-certificate}(m, v, n)$ iff its log holds:
 - ❑ The request m
 - ❑ A **PRE-PREPARE** for m in view v with sequence number n
 - ❑ $2f$ **PREPARE** from different backups that match the pre-prepare

Prepare Certificate

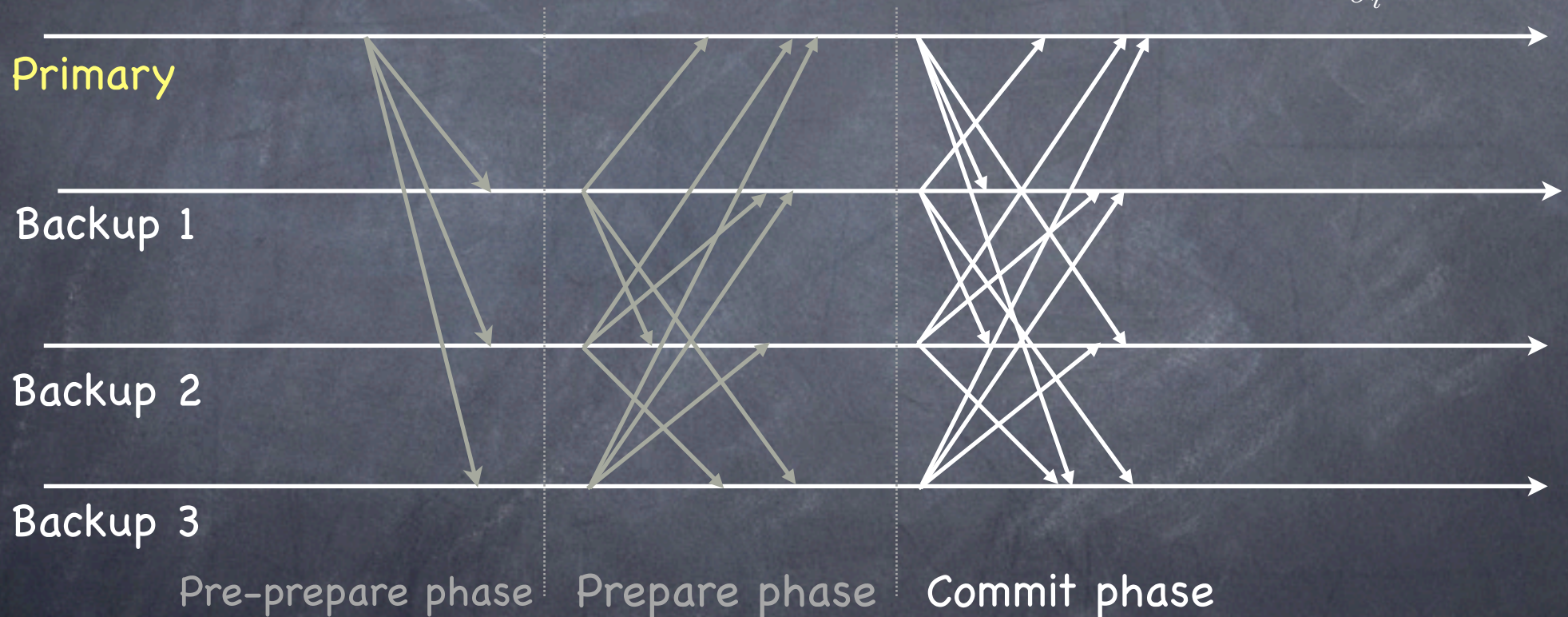
- **P-certificates** ensure total order within views
- Replica produces $P\text{-certificate}(m, v, n)$ iff its log holds:
 - The request m
 - A **PRE-PREPARE** for m in view v with sequence number n
 - $2f$ **PREPARE** from different backups that match the pre-prepare
- A $P\text{-certificate}(m, v, n)$ means that a quorum agrees with assigning sequence number n to m in view v
 - NO two non-faulty replicas with $P\text{-certificate}(m_1, v, n)$ and $P\text{-certificate}(m_2, v, n)$

P-certificates are not enough

- ⑥ A P-certificate proves that a majority of correct replicas has agreed on a sequence number for a client's request
- ⑥ Yet that order could be modified by a new leader elected in a **view change**

Commit

After collecting a P-certificate,
replica i multicasts $\langle \text{COMMIT}, v, n, d, i \rangle_{\sigma_i}$

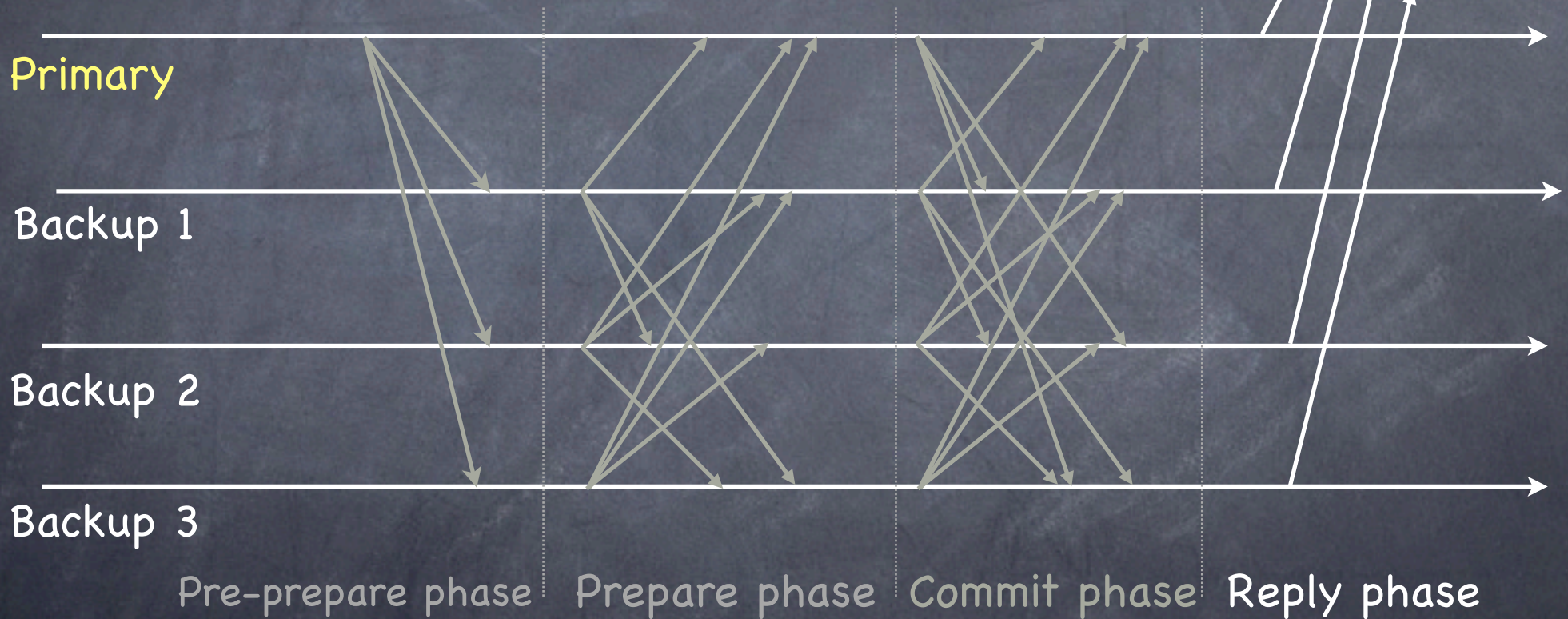


Commit Certificate

- 👁️ **C-certificates** ensure total order across views
 - ❑ can't miss P-certificate during a view change
- 👁️ A replica has a C-certificate(m, v, n) if:
 - ❑ it had a P-certificate (m, v, n)
 - ❑ log contains $2f + 1$ matching **COMMIT** from different replicas (including itself)
- 👁️ Replica executes a request after it gets C-certificate for it, and has cleared all requests with smaller sequence numbers

Reply

After executing request,
replica i replies with $\langle \text{REPLY}, v, t, c, i, r \rangle_{\sigma_i}$



Aux armes les backups!

- 👁️ A disgruntled backup mutinies:
 - ❑ stops accepting messages (but for VIEW-CHANGE & NEW-VIEW)
 - ❑ multicasts $\langle \text{VIEW-CHANGE}_{v+1}, \mathcal{P} \rangle_{\sigma_i}$
 - ❑ \mathcal{P} contains all P-Certificates known to replica i
- 👁️ A backup joins mutiny after seeing $f+1$ distinct VIEW-CHANGE messages
- 👁️ Mutiny succeeds if new primary collects a **new-view certificate** \mathcal{V} , indicating support from $2f+1$ distinct replicas (including itself)

On to view $v+1$: the new primary

- ⑥ The “primary elect” \hat{p} (replica $v+1 \bmod N$) extracts from the new-view certificate \mathcal{V} :
 - the highest sequence number h of any message for which \mathcal{V} contains a \mathcal{P} -certificate

On to view $v+1$: the new primary

- The “primary elect” \hat{p} (replica $v+1 \bmod N$) extracts from the new-view certificate \mathcal{V} :
 - the highest sequence number h of any message for which \mathcal{V} contains a P-certificate



On to view $v+1$: the new primary

- ⑥ The “primary elect” \hat{p} (replica $v+1 \bmod N$) extracts from the new-view certificate \mathcal{V} :
 - the highest sequence number h of any message for which \mathcal{V} contains a P-certificate
 - two sets \mathcal{O} and \mathcal{N} :
 - ▶ If there is a P-certificate for n, m in \mathcal{V} , $n \leq h$
$$\mathcal{O} = \mathcal{O} \cup \langle \text{PRE-PREPARE}, v+1, n, m \rangle_{\sigma_{\hat{p}}}$$
 - ▶ Otherwise, if $n \leq h$ but no P-certificate:
$$\mathcal{N} = \mathcal{N} \cup \langle \text{PRE-PREPARE}, v+1, n, \text{null} \rangle_{\sigma_{\hat{p}}}$$

On to view $v+1$: the new primary

- ⑥ The “primary elect” \hat{p} (replica $v+1 \bmod N$) extracts from the new-view certificate \mathcal{V} :
 - the highest sequence number h of any message for which \mathcal{V} contains a P-certificate
 - two sets \mathcal{O} and \mathcal{N} :
 - ▶ If there is a P-certificate for n, m in \mathcal{V} , $n \leq h$
$$\mathcal{O} = \mathcal{O} \cup \langle \text{PRE-PREPARE}, v+1, n, m \rangle_{\sigma_{\hat{p}}}$$
 - ▶ Otherwise, if $n \leq h$ but no P-certificate:
$$\mathcal{N} = \mathcal{N} \cup \langle \text{PRE-PREPARE}, v+1, n, \text{null} \rangle_{\sigma_{\hat{p}}}$$
- ⑥ \hat{p} multicasts $\langle \text{NEW-VIEW}, v+1, \mathcal{V}, \mathcal{O}, \mathcal{N} \rangle_{\sigma_{\hat{p}}}$

On to view $v+1$: the backup

- Backup accepts **NEW-VIEW** message for $v+1$ if
 - it is signed properly
 - it contains $2f+1$ valid **VIEW-CHANGE** messages for $v+1$
 - it can verify locally that \mathcal{O} and \mathcal{N} are correct (repeating the primary's computation)
- Adds all entries in \mathcal{O} and \mathcal{N} to its log (so did \hat{p} !)
- Multicasts a **PREPARE** for each message in \mathcal{O} and \mathcal{N}
- Adds all **PREPARE** to log and enters new view