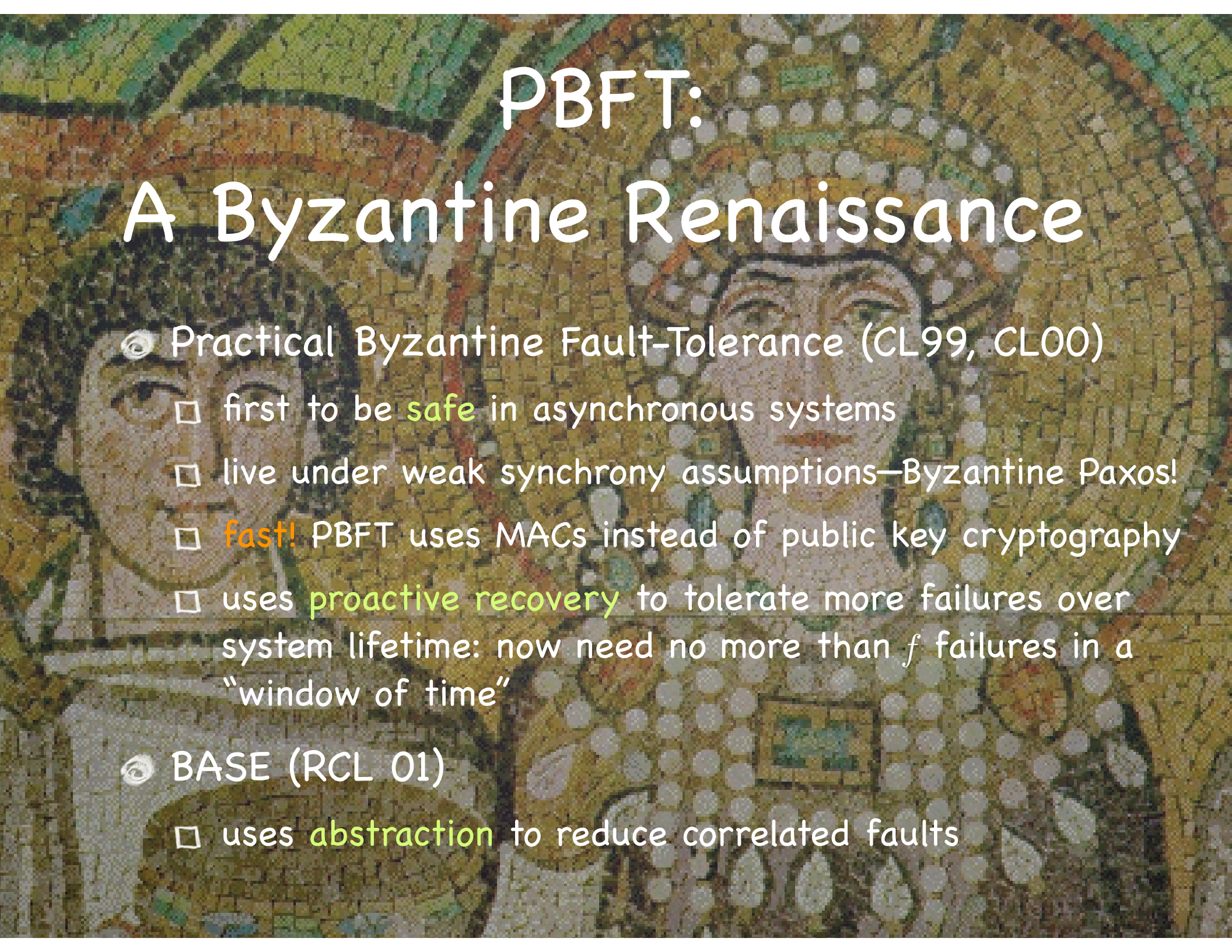


PBFT: A Byzantine Renaissance

- 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

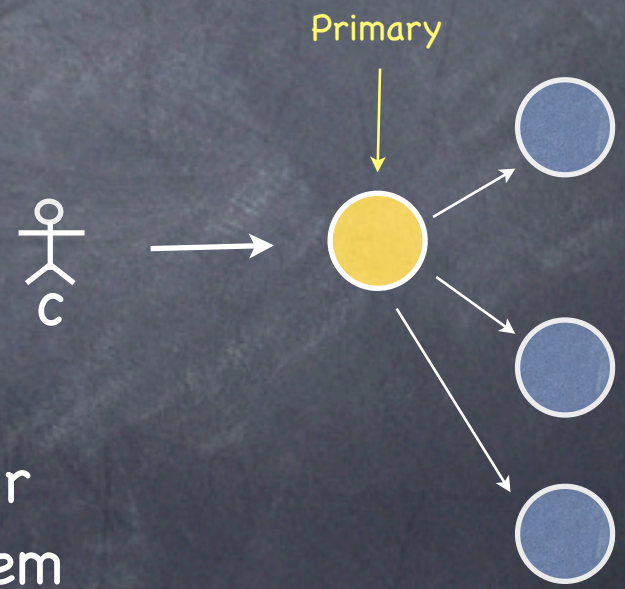
- Byzantine clients
- Up to f Byzantine servers
- $N > 3f$ total servers

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 <sequence number, command> 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: no equivocation!
  - 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

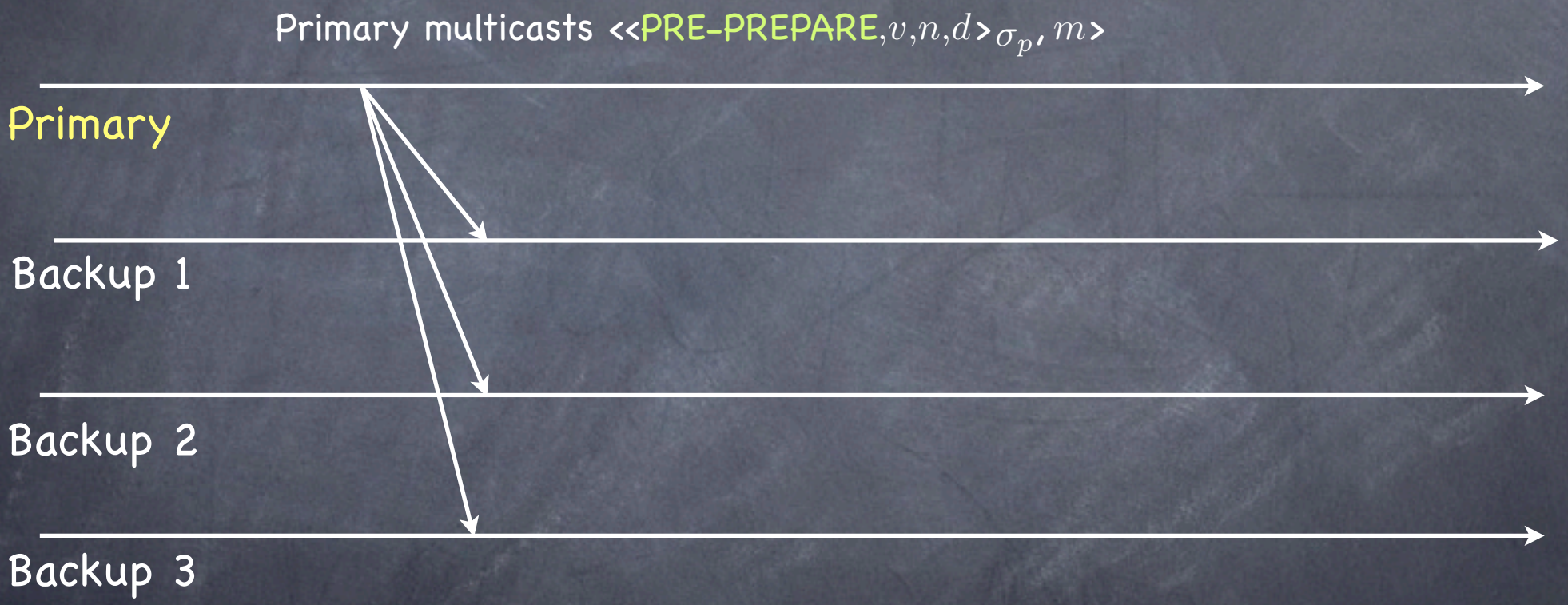
Client issues request

Client issues request

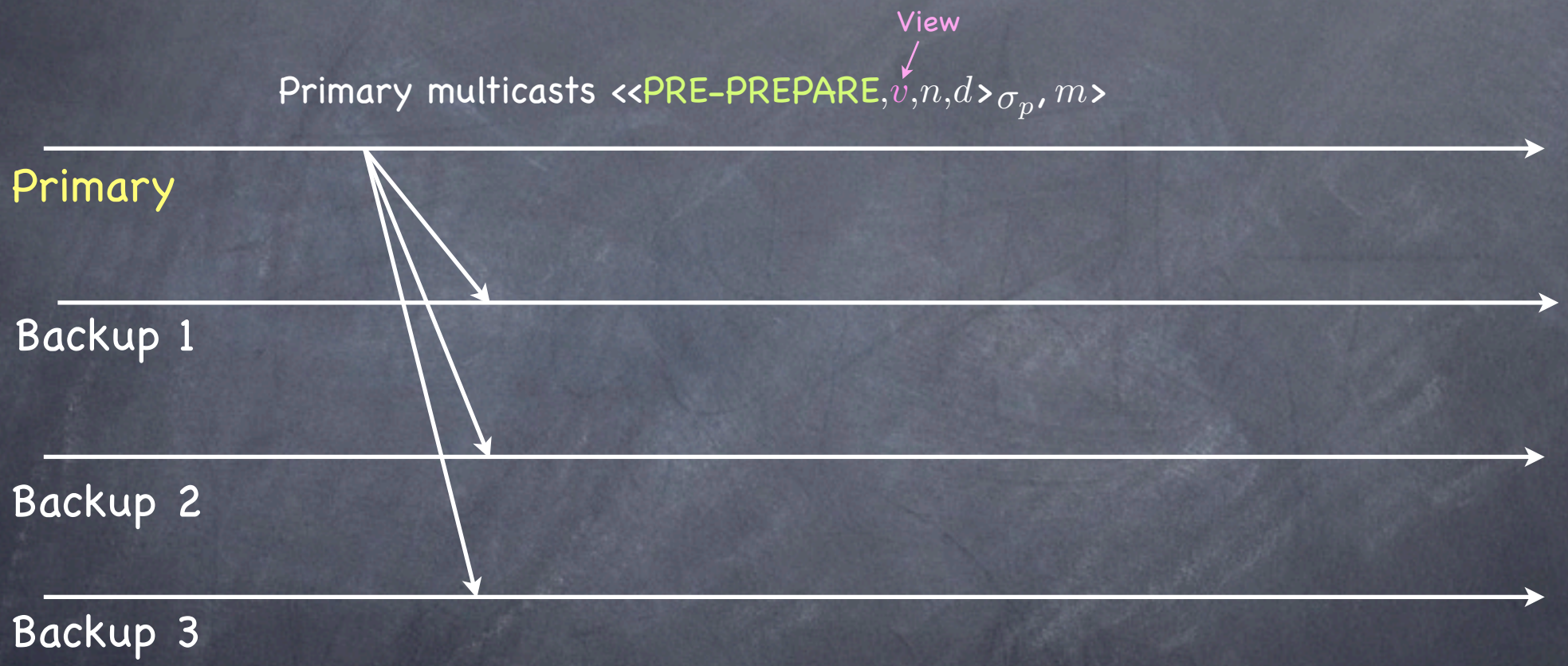
Client issues request

Client issues request

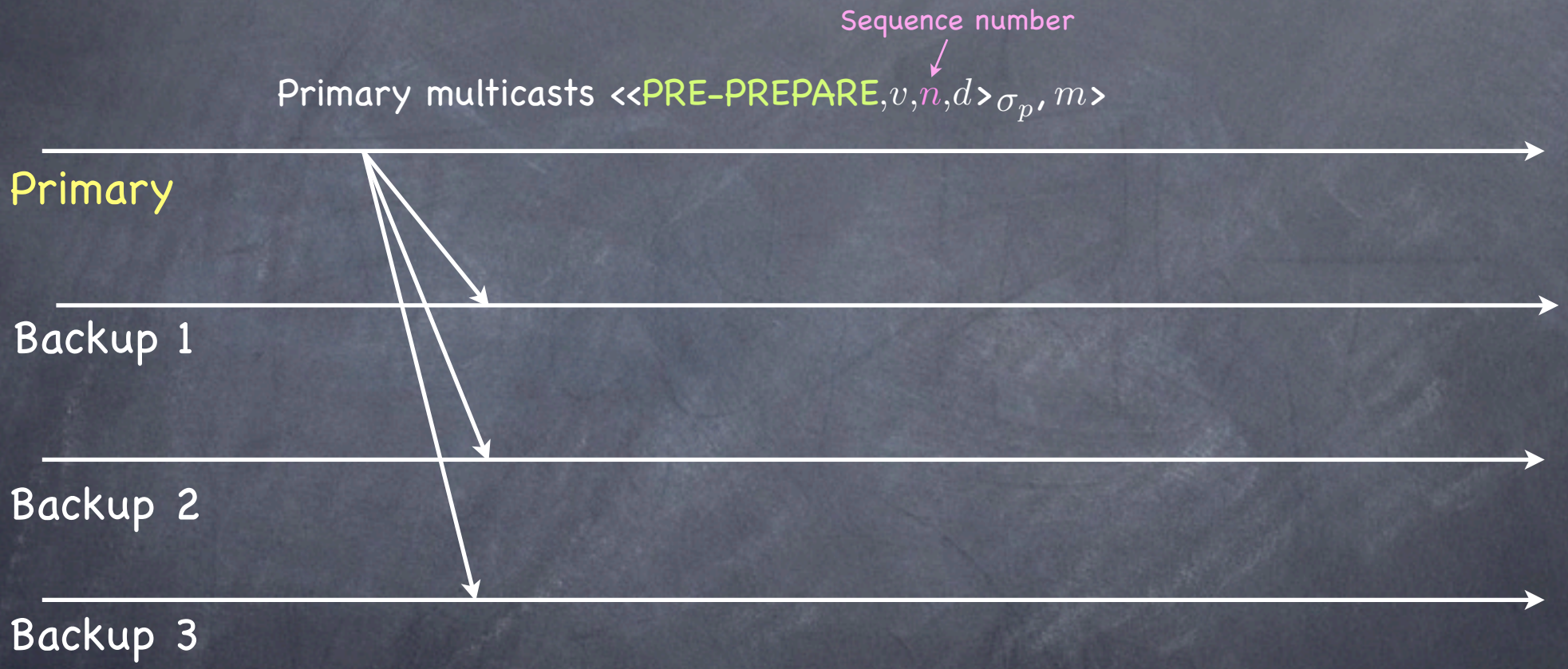
Pre-prepare


Primary multicasts <<PRE-PREPARE,v,n,d>$_{\sigma_p}$, m>
Primary
Backup 1
Backup 2
Backup 3

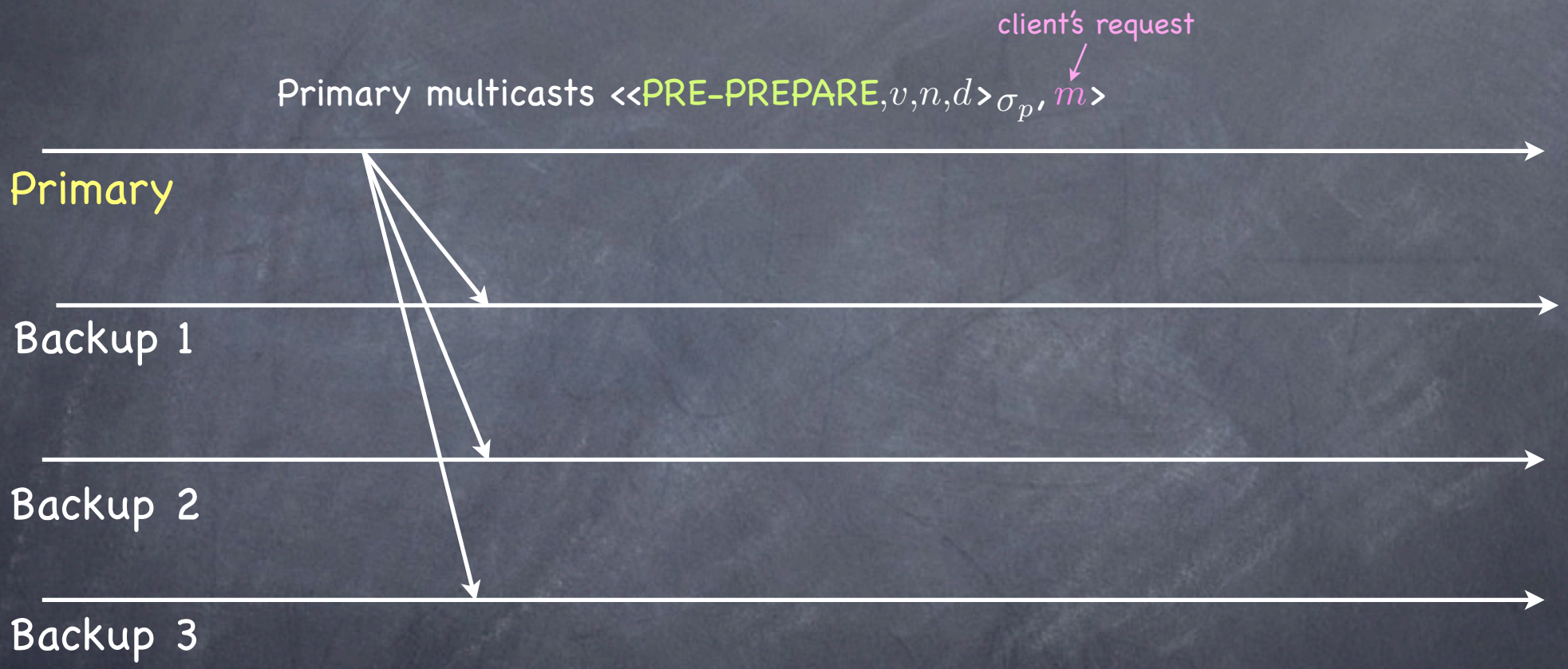
Pre-prepare


View
Primary multicasts <<PRE-PREPARE,v,n,d>σ_p, m>
Primary
Backup 1
Backup 2
Backup 3

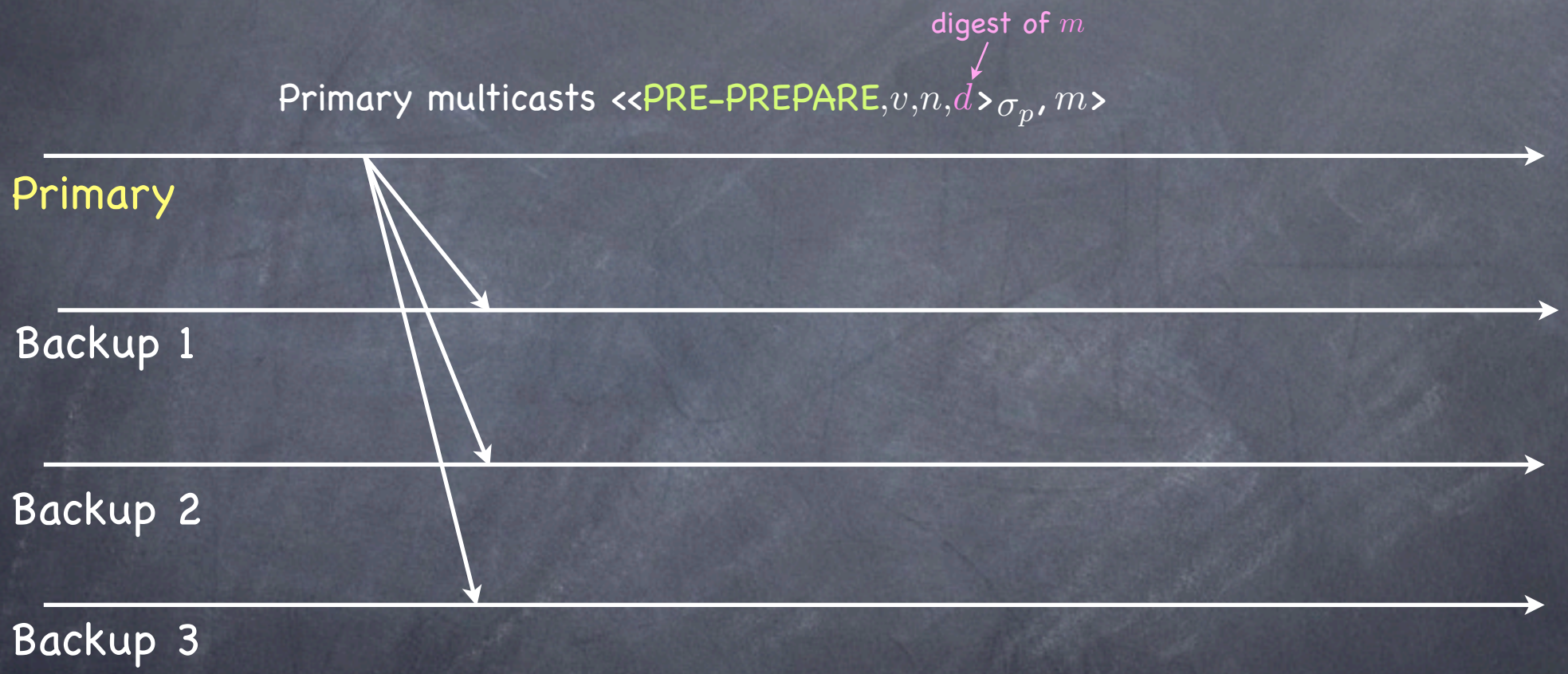
Pre-prepare


Sequence number
Primary multicasts <<PRE-PREPARE,v,n,d>σ_p, m>
Primary
Backup 1
Backup 2
Backup 3

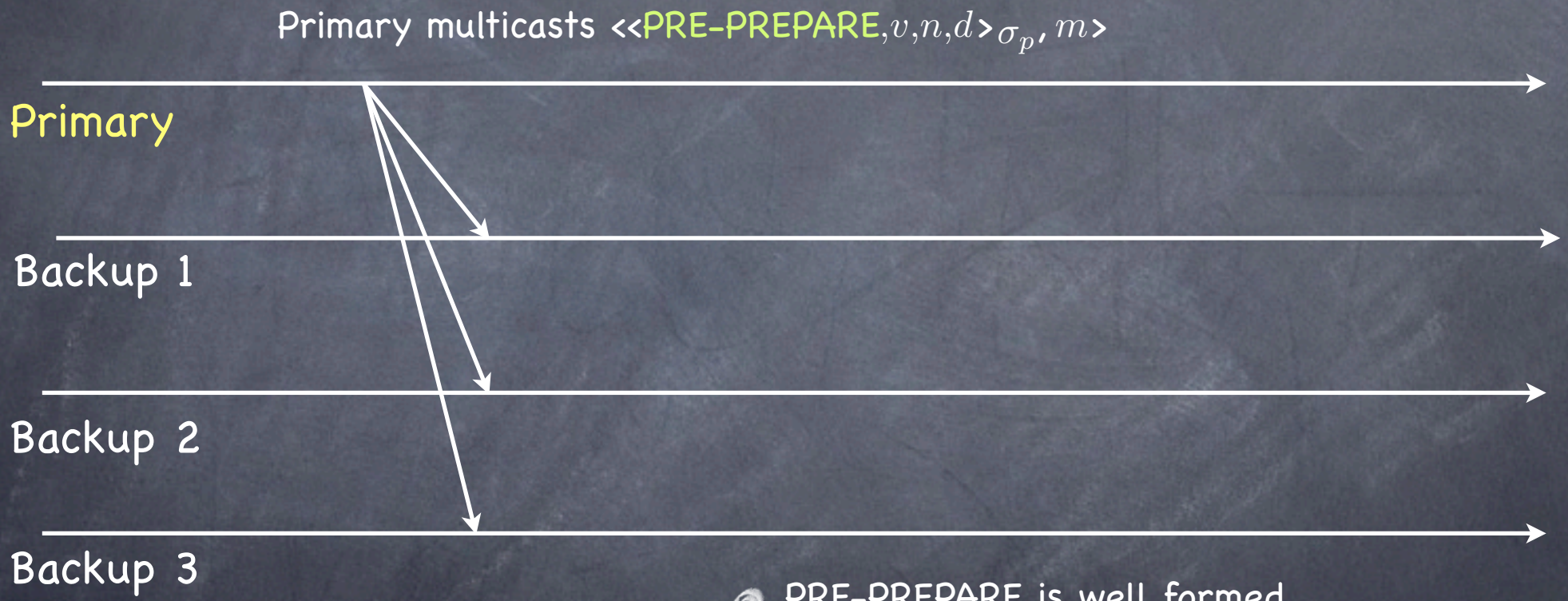
Pre-prepare


client's request
Primary multicasts <<PRE-PREPARE,v,n,d>σ_p, m>
Primary
Backup 1
Backup 2
Backup 3

Pre-prepare


digest of m
Primary multicasts <<PRE-PREPARE,v,n,d>σ_p, m>
Primary
Backup 1
Backup 2
Backup 3

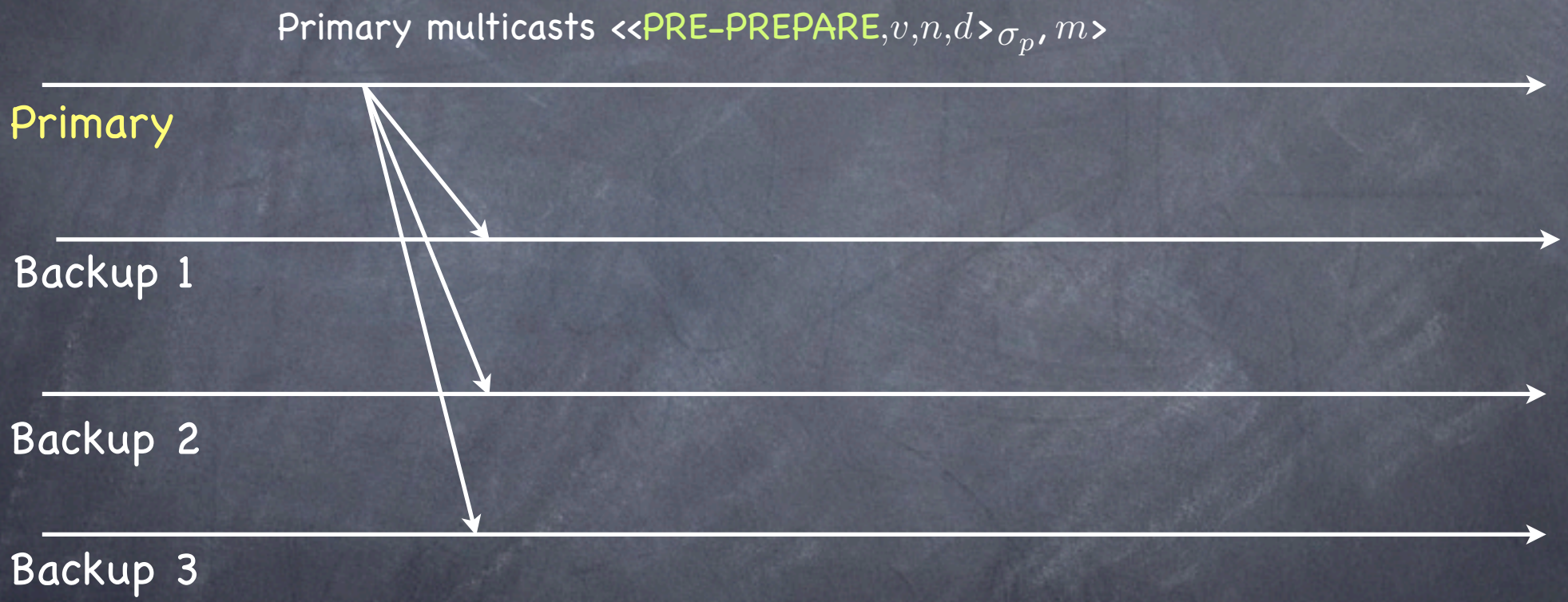
Pre-prepare



Correct backup i accepts PRE-PREPARE if:

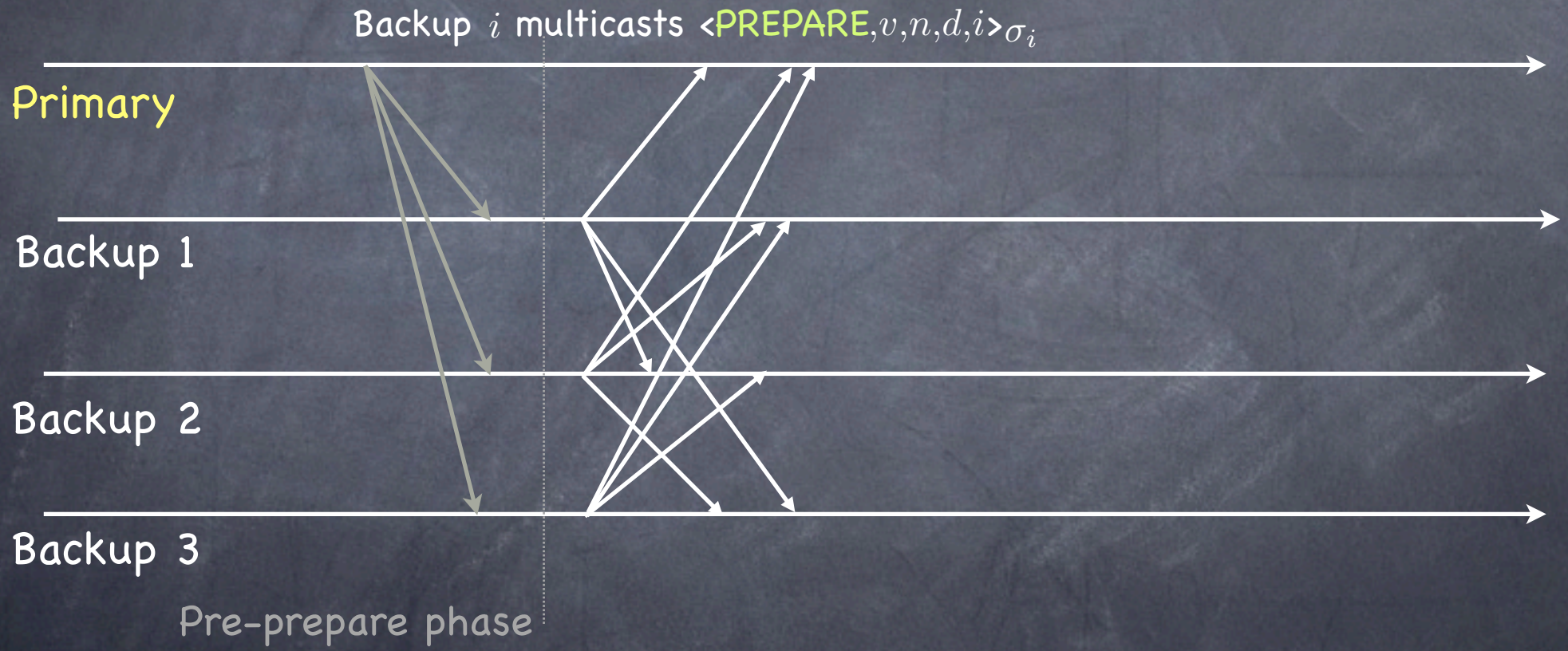
- PRE-PREPARE is well formed
- i is in view v
- i has not accepted another PRE-PREPARE for v, n with a different d
- n is between two water-marks L and H (to prevent sequence number exhaustion)

Pre-prepare



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

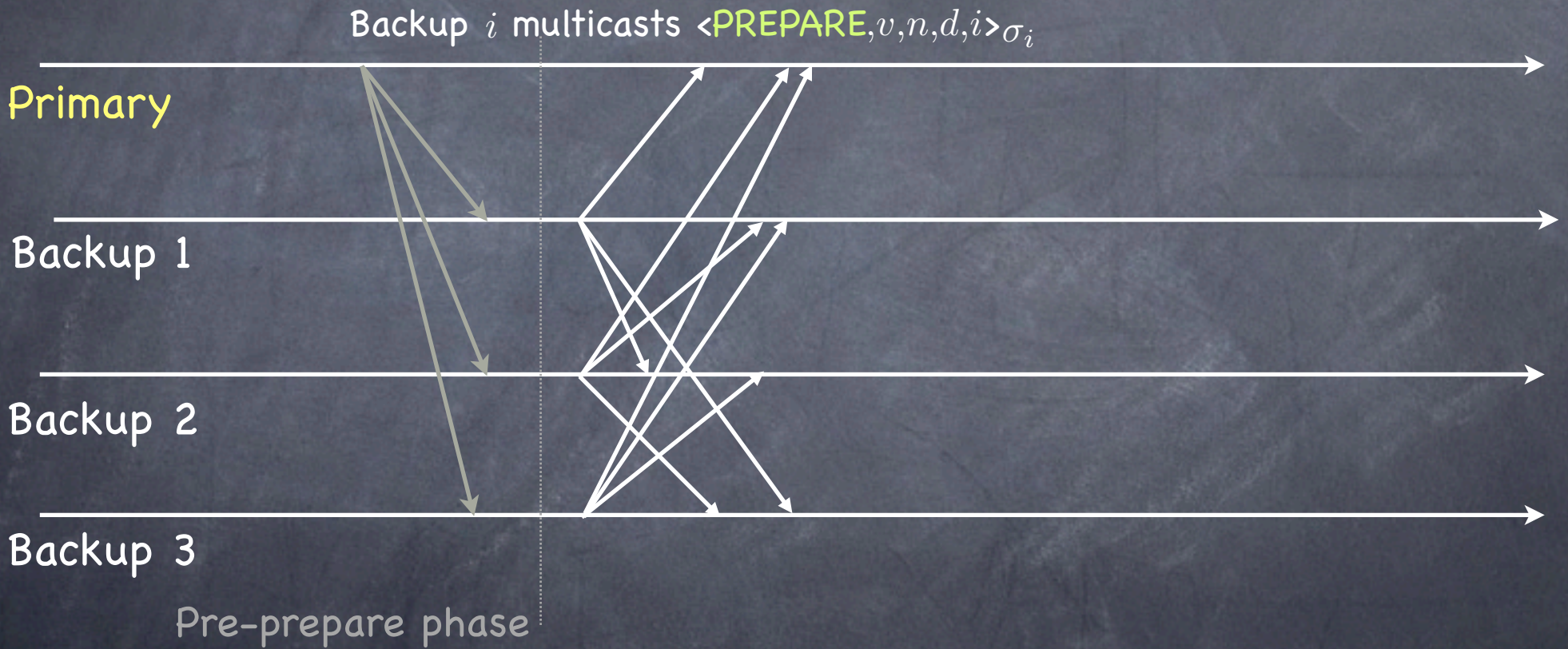
Prepare



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-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

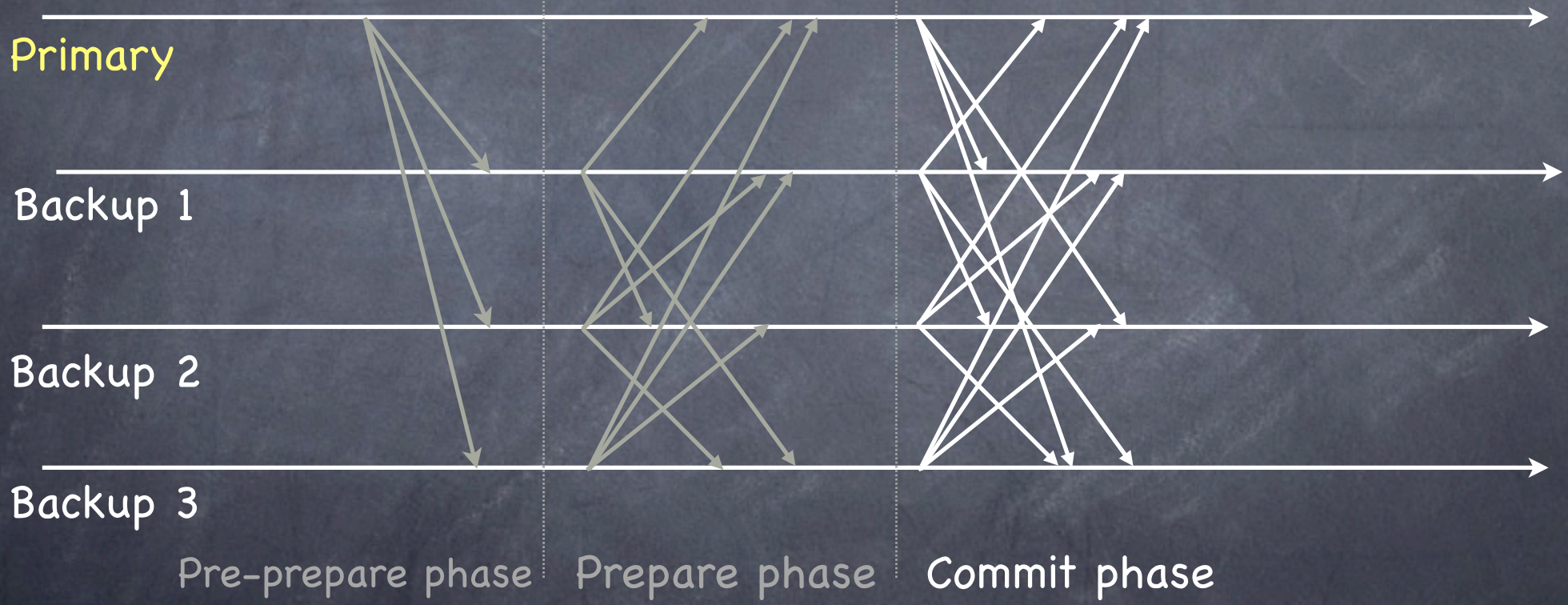
- P-certificates ensure total order within views
- Replica produces P-certificate(m,v,n) iff its log holds:
  - The request m
  - A PRE-PREPARE for m in view v with sequence number n
  - $2f$ PREPARE from different backups that match the pre-prepare
- A P-certificate(m,v,n) means that a quorum agrees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

P-certificates are not enough

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

Commit

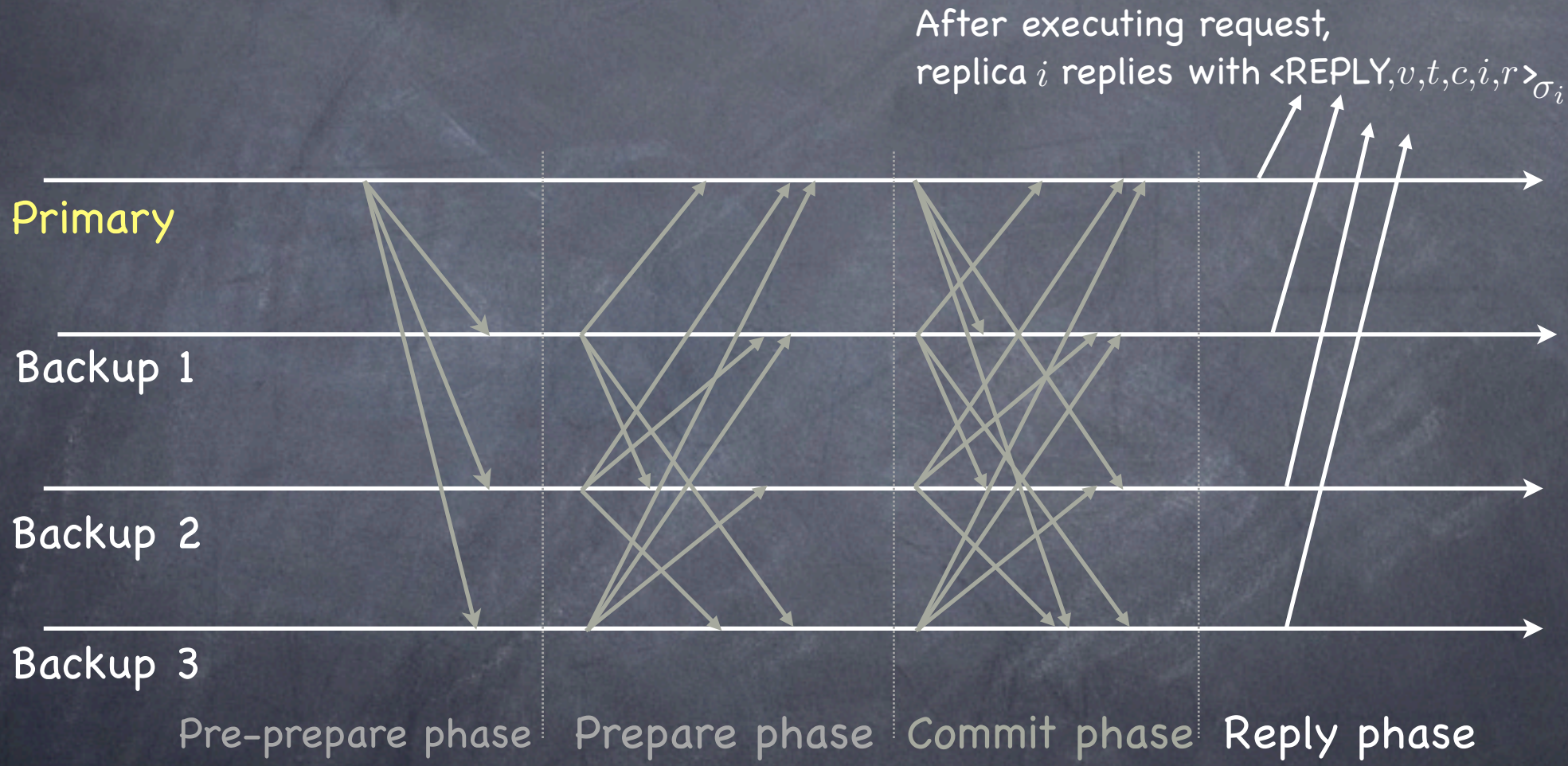
After collecting a P-certificate,
replica i multicasts <COMMIT,v,n,d,i>$_{\sigma_i}$



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 <REPLY,v,t,c,i,r>$_{\sigma_i}$
Primary
Backup 1
Backup 2
Backup 3
Pre-prepare phase
Prepare phase
Commit phase
Reply phase

Aux armes les backups!

- A disgruntled backup mutinies:
  - stops accepting messages (but for VIEW-CHANGE & NEW-VIEW)
  - multicasts <VIEW-CHANGE, $v+1, \mathcal{P}$ >$_{\sigma_i}$
  - $\mathcal{P}$ 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

- 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

- 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

- 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 initially empty sets $\mathcal{O}$ and $\mathcal{N}$:
    - If there is a P-certificate for n,m in $\mathcal{V}$, $n \leq h$
      $\mathcal{O} = \mathcal{O} \cup$ <PRE-PREPARE,$v+1$,n,m>$_{\sigma_{\hat{p}}}$
    - Otherwise, if $n \leq h$ but no P-certificate:
      $\mathcal{N} = \mathcal{N} \cup$ <PRE-PREPARE,$v+1$,n,$null$>$_{\sigma_{\hat{p}}}$

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
  - two initially empty sets $\mathcal{O}$ and $\mathcal{N}$:
    - If there is a P-certificate for n,m in $\mathcal{V}$, $n \leq h$
      $\mathcal{O} = \mathcal{O} \cup$ <PRE-PREPARE,$v+1$,n,m>$_{\sigma_{\hat{p}}}$
    - Otherwise, if $n \leq h$ but no P-certificate:
      $\mathcal{N} = \mathcal{N} \cup$ <PRE-PREPARE,$v+1$,n,$null$>$_{\sigma_{\hat{p}}}$
- $\hat{p}$ multicasts <NEW-VIEW,$v+1$,$\mathcal{V}$,$\mathcal{O}$,$\mathcal{N}$>$_{\sigma_{\hat{p}}}$

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

Garbage Collection

- For safety, a correct replica keeps in log messages about request o until
  - o has been executed by a majority of correct replicas, and
  - this fact can be proven during a view change
- Truncate log with a Stable Certificate
  - Each replica i periodically (after processing k requests) checkpoints state and multicasts <CHECKPOINT,n,d,i>

Garbage Collection

- For safety, a correct replica keeps in log messages about request o until
  - o has been executed by a majority of correct replicas, and
  - this fact can be proven during a view change
- Truncate log with a Stable Certificate
  - Each replica i periodically (after processing k requests) checkpoints state and multicasts <CHECKPOINT,n,d,i>

last executed request reflected in state

Garbage Collection

- For safety, a correct replica keeps in log messages about request o until
  - o has been executed by a majority of correct replicas, and
  - this fact can be proven during a view change
- Truncate log with a Stable Certificate
  - Each replica i periodically (after processing k requests) checkpoints state and multicasts <CHECKPOINT,n,d,i>

state's digest

Garbage Collection

- For safety, a correct replica keeps in log messages about request o until
  - o has been executed by a majority of correct replicas, and
  - this fact can be proven during a view change
- Truncate log with a Stable Certificate
  - Each replica i periodically (after processing k requests) checkpoints state and multicasts <CHECKPOINT,n,d,i>
  - $2f+1$ CHECKPOINT messages prove the checkpoint is correct and that it can be retrieved on view change