Practical Byzantine Fault Tolerance

Bibliography

M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery.
ACM Trans. Comput. Syst., 20:398–461, Nov. 2002.
http://www.disi.unitn.it/~montreso/ds/papers/PbftTocs.pdf

Assumptions

- System model
  - Asynchronous distributed system with N processes
  - Unreliable channels
- Unbreakable cryptography
  - Message m is signed by its sender i, and we write $\langle m \rangle_{\sigma(i)}$, through:
    - Public/private key pairs
    - Message authentication codes (MAC)
  - A digest $d(m)$ of message m is produced through collision-resistant hash functions

Specification

- State machine replication
  - Replicated service with a state and deterministic operations operating on it
  - Clients issue a request and block waiting for reply
- Safety
  - The system satisfies linearizability, provided that $N > 3f + 1$
  - Regardless of "faulty clients"...
    - all operations performed by faulty clients are observed in a consistent way by non-faulty clients
  - The algorithm does not rely on synchrony to provide safety...
- Liveness
  - It relies on synchrony to provide liveness
  - Assumes $delay(t)$ does not grow faster than t indefinitely
  - Weak assumption – if network faults are eventually repaired
  - Circumvent the impossibility results of FLP

Assumptions

- Failure model
    - Up to f Byzantine servers
    - $N > 3f$ total servers
    - (Potentially Byzantine clients)
- Independent failures
    - Different implementations of the service
    - Different operating systems
    - Different root passwords, different administrator

Optimality

Theorem

To tolerate up to f malicious nodes, N must be equal to $3f + 1$

Proof

- It must be possible to proceed after communicating with $N - f$ replicas, because the faulty replicas may not respond
- But the f replicas not responding may be just slow, so f of those that responded might be faulty
- The correct replicas who responded $(N - 2f)$ must outnumber the faulty replicas, so
$$N - 2f > f \Rightarrow N > 3f$$

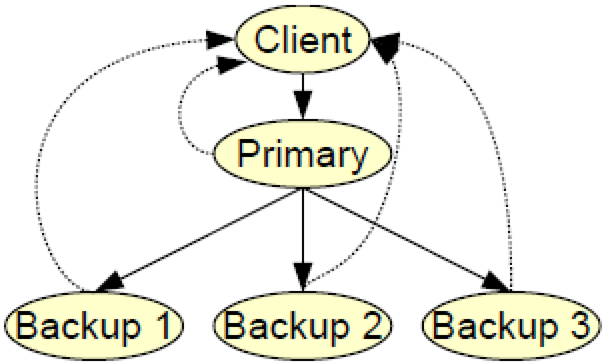
- So, $N > 3f$ to ensure that at least a correct replica is present in the reply set
- $N = 3f + 1$; more is useless
    - more and larger messages
    - without improving resiliency

Processes and views

- Replicas IDs: $0 \ldots N-1$
- Replicas move through a sequence of configurations called views
- During view v:
  - Primary replica is i: $i = v \bmod N$
  - The other are backups
- View changes are carried out when the primary appears to have failed

The algorithm

- To invoke an operation, the client sends a request to the primary
- The primary multicasts the request to the backups
- Quorums are employed to guarantee ordering on operations
- When an order has been agreed, replicas execute the request and send a reply to the client
- When the client receives at least $f + 1$ identical replies, it is satisfied

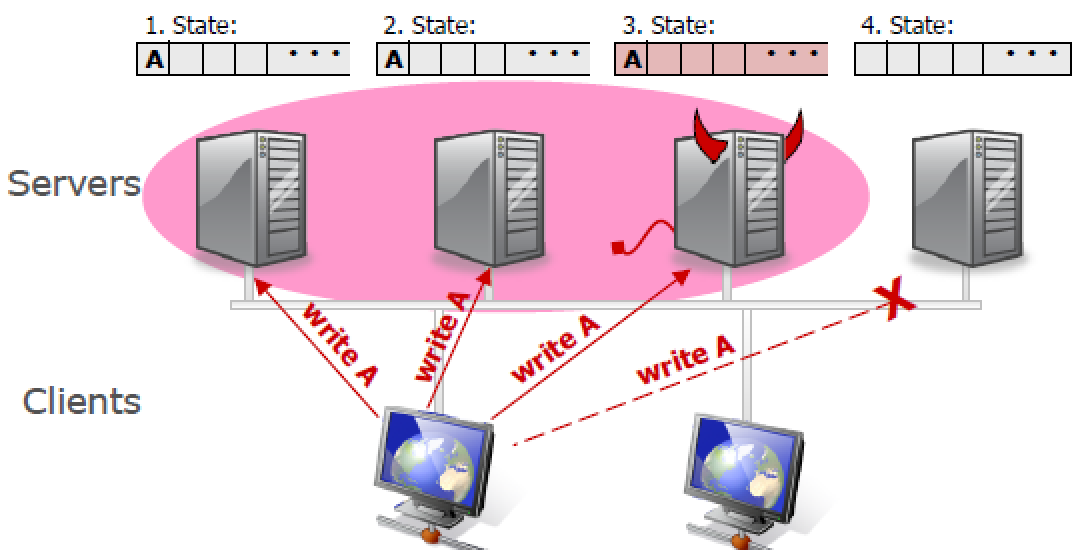


Problems

- 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
- Faulty replicas could incorrectly respond to the client!
  - Client waits for $f + 1$ matching replies before accepting response

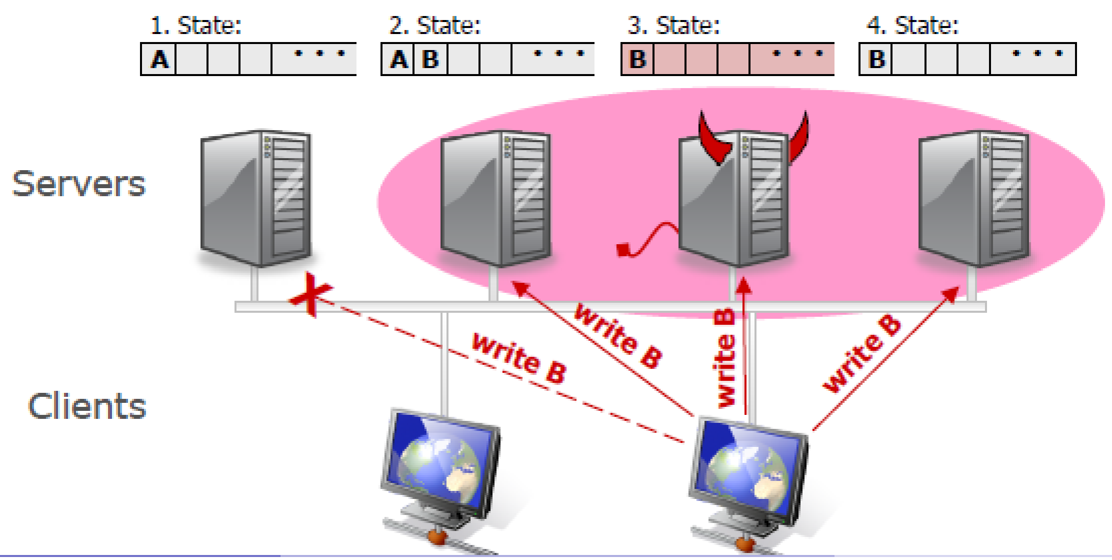
The general idea

- 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
  - There is always one quorum that contains only non-faulty replicas



The general idea

- 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
  - There is always one quorum that contains only non-faulty replicas



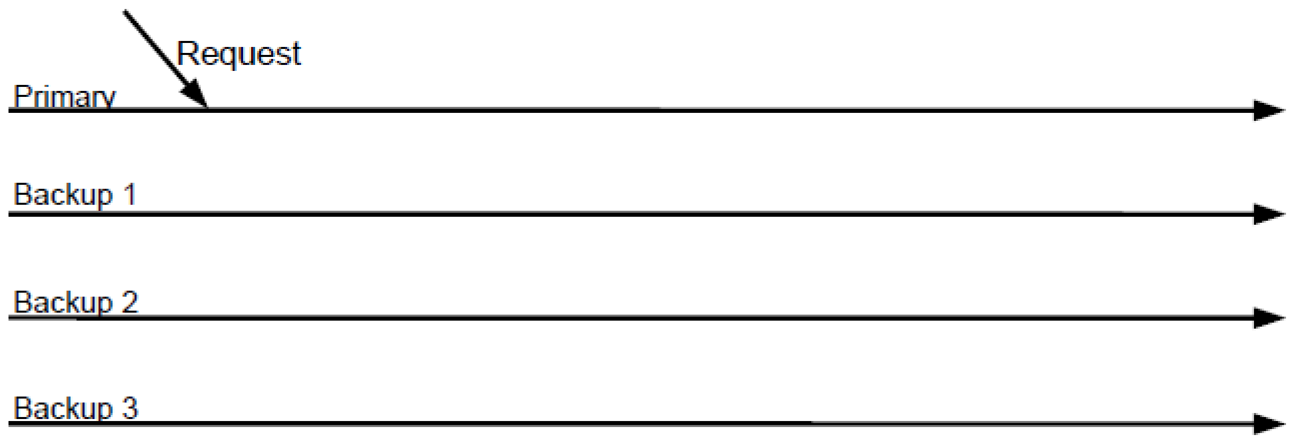
Protocol schema

- 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 (not here)

State

- The internal state of each of the replicas include:
  - the state of the actual service
  - a message log containing all the messages the replica has accepted
  - an integer denoting the replica current view

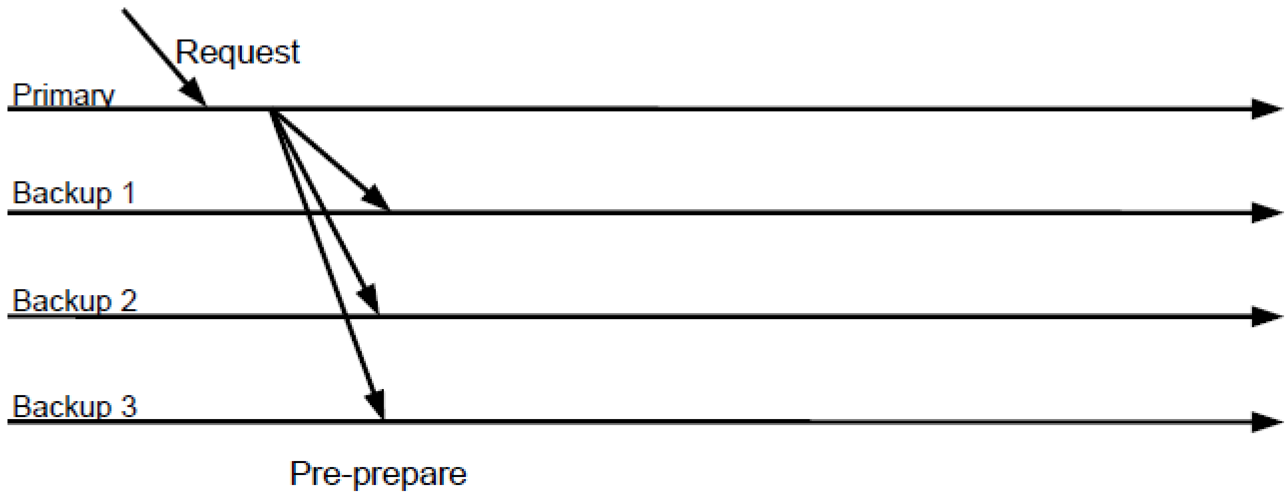
Client request



$\langle \text{REQUEST}, o, t, c \rangle_{\sigma(c)}$

- o: state machine operation
- t: timestamp (used to ensure exactly-once semantics)
- c: client id
- $\sigma(c)$: client signature

Pre-prepare phase



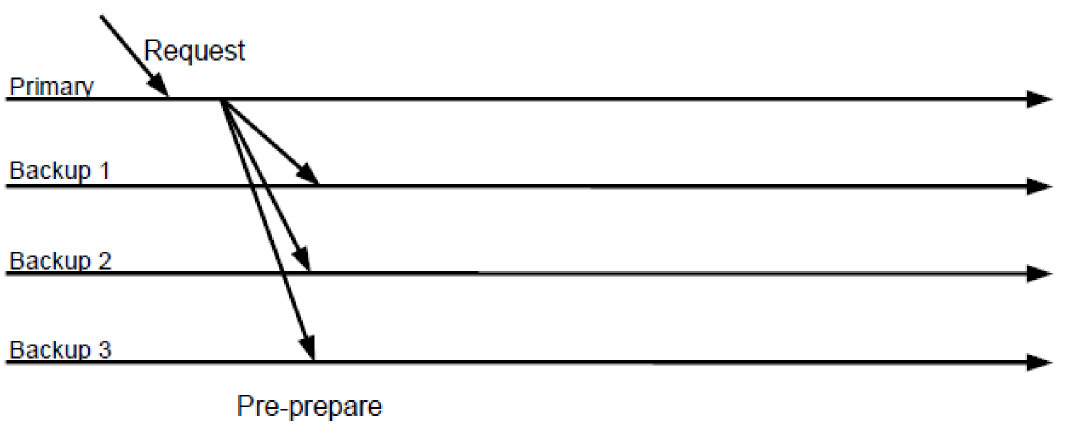
$\langle\langle \text{PRE-PREPARE}, v, n, d(m)\rangle_{\sigma(p)}, m\rangle$

- v: current view
- n: sequence number
- $d(m)$: digest of client message
- $\sigma(p)$: primary signature
- m: client message

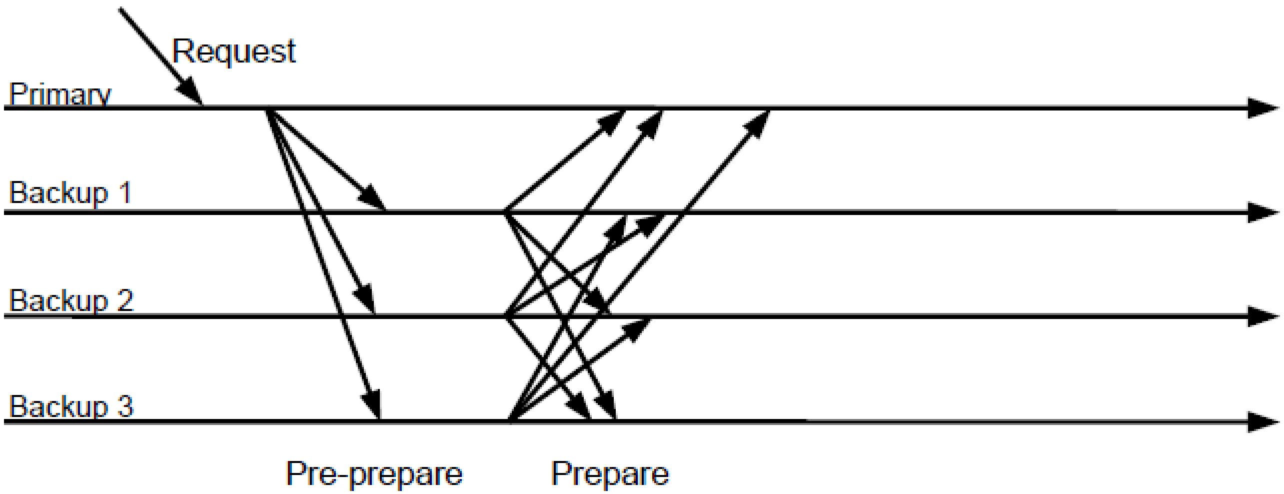
Pre-prepare phase

$\langle\langle \textsc{pre-prepare}, v, n, d(m)\rangle_{\sigma(p)}, m\rangle$

- Correct replica i accepts PRE-PREPARE if:
  - the PRE-PREPARE message is well-formed
  - the current view of i is v
  - i has not accepted another PRE-PREPARE for v, n with a different digest
  - n is between two water-marks L and H
    (to avoid sequence number exhaustion caused by faulty primaries)
- Each accepted PRE-PREPARE message is stored in the accepting replica's message log (including the primary's)
- Non-accepted PRE-PREPARE messages are just discarded



Prepare phase



$\langle \textsc{prepare}, v, n, d(m) \rangle_{\sigma(i)}$

- Accepted by correct replica j if:
  - the PREPARE message is well-formed
  - current view of j is v
  - n is between two water-marks L and H
- Replicas that send PREPARE accept the sequence number n for m in view v
- Each accepted PREPARE message is stored in the accepting replica's message log

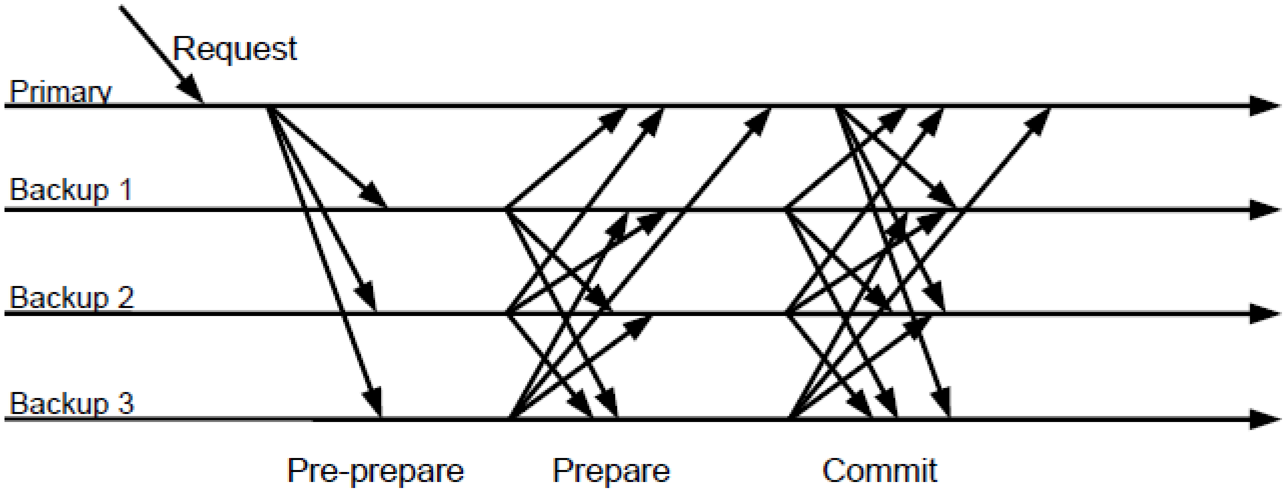
Prepare certificate (P-certificate)

- Replica i produces a prepare certificate **prepared**(m, v, n, i) iff its log holds:
  - The request m
  - A PRE-PREPARE for m in view v with sequence number n
  - Log contains $2f$ PREPARE messages from different backups that match the PRE-PREPARE
- **prepared**(m, v, n, i) means that a quorum of $(2f + 1)$ replicas agrees with assigning sequence number n to m in view v

Theorem

There are no two non-faulty replicas i, j such that **prepared**(m, v, n, i) and **prepared**(m', v, n, j), with $m \neq m'$

Commit phase



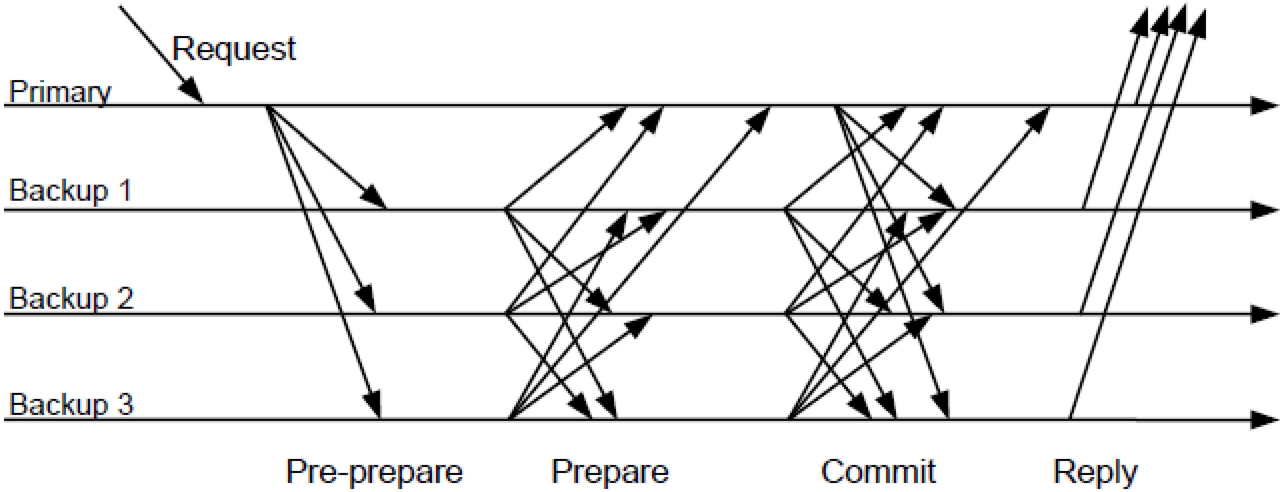
$\langle \textsc{commit}, v, n, d(m), i \rangle_{\sigma(i)}$

- After having collected a P-certificate **prepared**(m, v, n, i), replica i sends a COMMIT message
- Accepted if:
  - The COMMIT message is well-formed
  - Current view of i is v
  - n is between two water-marks L and H

Commit certificate (C-Certificate)

- Commit certificates ensure total order across views
  - we guarantee that we can't miss prepare certificates during a view change
- A replica has a certificate **committed**(m, v, n, i) if:
  - it had a P-certificate **prepared**(m, v, n, i)
  - log contains $2f + 1$ matching COMMIT from different replicas (possibly including its own)
- Replica executes a request after it gets commit certificate for it, and has cleared all requests with smaller sequence numbers

Reply phase



$\langle \text{REPLY}, v, t, c, i, r \rangle_{\sigma(i)}$

- r is the reply
- Client waits for $f + 1$ replies with the same t, r
- If the client does not receive replies soon enough, it broadcast the request to all replicas

View change

- A un-satisfied replica backup i mutinies:
  - stops accepting messages (except VIEW-CHANGE and NEW-VIEW)
  - multicasts $\langle \text{VIEW-CHANGE}, v+1, P, i \rangle_{\sigma(i)}$
  - P contains a P-certificate P_m for each request m (up to a given number, see garbage collection)
- Mutiny succeeds if the new primary collects a new-view certificate V:
  - a set containing $2f+1$ VIEW-CHANGE messages
  - indicating that $2f+1$ distinct replicas (including itself) support the change of leadership

View change

The "primary elect" p' (replica $v+1 \bmod N$):

- extracts from the new-view certificate V the highest sequence number h of any message for which V contains a P-certificate
- creates a new PRE-PREPARE message for any client message m with sequence number $n \leq h$ and add it to the set O
  - if there is a P-certificate for n, m in V

$$O \leftarrow O \cup \langle \text{PRE-PREPARE}, v+1, n, d_m \rangle_{\sigma(p')}$$

  - Otherwise

$$O \leftarrow O \cup \langle \text{PRE-PREPARE}, v+1, n, d_{null} \rangle_{\sigma(p')}$$

- p' multicasts $\langle \text{NEW-VIEW}, v+1, V, O \rangle_{\sigma(p')}$

View change

- Backup accepts a $\langle \textsc{new-view}, v+1, V, O\rangle_{\sigma(p')}$ message for $v+1$ if
  - it is signed properly by p'
  - V contains valid VIEW-CHANGE messages for $v+1$
  - the correctness of O can be locally verified (repeating the primary's computation)
- Actions:
  - Adds all entries in O to its log (so did p'!)
  - Multicasts a PREPARE for each message in O
  - Adds all PREPAREs to the log and enters new view

Garbage collection

- 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 proven during a view change
- Truncate log with stable checkpoints
  - Each replica i periodically (after processing k requests) checkpoints state and multicasts $\langle \text{CHECKPOINT}, n, d, i \rangle$
    - n: last executed request
    - d: state digest
- A set S containing $2f + 1$ equivalent CHECKPOINT messages from distinct processes are a proof of the checkpoint's correctness (stable checkpoint certificate)

View Change, revisited

- Message $\langle \text{VIEW-CHANGE}, v+1, n, S, C, P, i\rangle_{\sigma(i)}$
  - n: the sequence number of the last stable checkpoint
  - S: the last stable checkpoint
  - C: the checkpoint certificate ($2f+1$ checkpoint messages)
- Message $\langle \text{NEW-VIEW}, v+1, n, V, O\rangle_{\sigma(p')}$
  - n: the sequence number of the last stable checkpoint
  - V, O: contains only requests with sequence number larger than n

Optimizations

- Reducing replies
  - One replica designated to send reply to client
  - Other replicas send digest of the reply
- Lower latency for writes (4 messages)
  - Replicas respond at Prepare phase (tentative execution)
  - Client waits for $2f + 1$ matching responses
- Fast reads (one round trip)
  - Client sends to all; they respond immediately
  - Client waits for $2f + 1$ matching responses

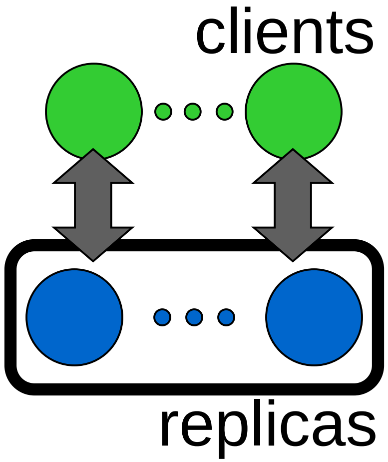
Optimizations: cryptography

- Reducing overhead
  - Public-key cryptography only for view changes
  - MACs (message authentication codes) for all other messages
- To give an idea (Pentium 200Mhz)
  - Generating 1024-bit RSA signature of a MD5 digest: 43ms
  - Generating a MAC of the same message: 10μs

Talk Overview

- Problem
- Assumptions
- Algorithm
- Implementation
- Performance
- Conclusions

Algorithm Properties



- Arbitrary replicated service
  - complex operations
  - mutable shared state
- Properties (safety and liveness):
  - system behaves as correct centralized service
  - clients eventually receive replies to requests
- Assumptions:
  - *3f+1* replicas to tolerate *f* Byzantine faults (optimal)
  - strong cryptography
  - **only for liveness:** eventual time bounds

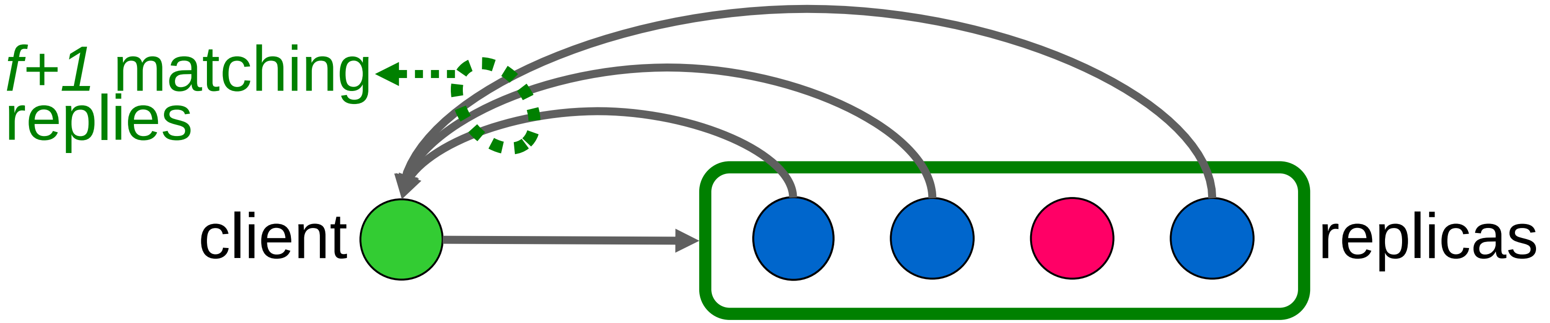
Algorithm Overview

State machine replication:

– deterministic replicas start in same state

– replicas execute same requests in same order

– correct replicas produce identical replies

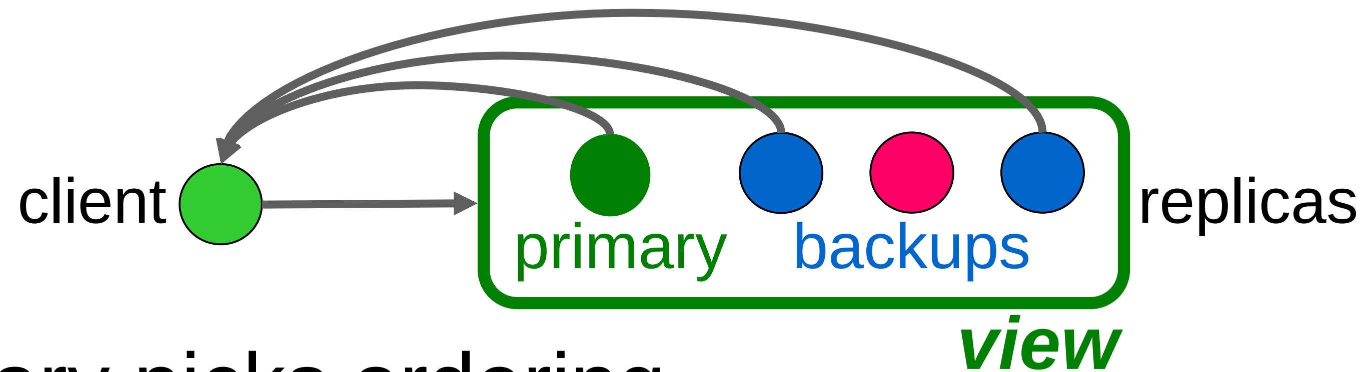


Hard: ensure requests execute in same order

Ordering Requests

Primary-Backup:

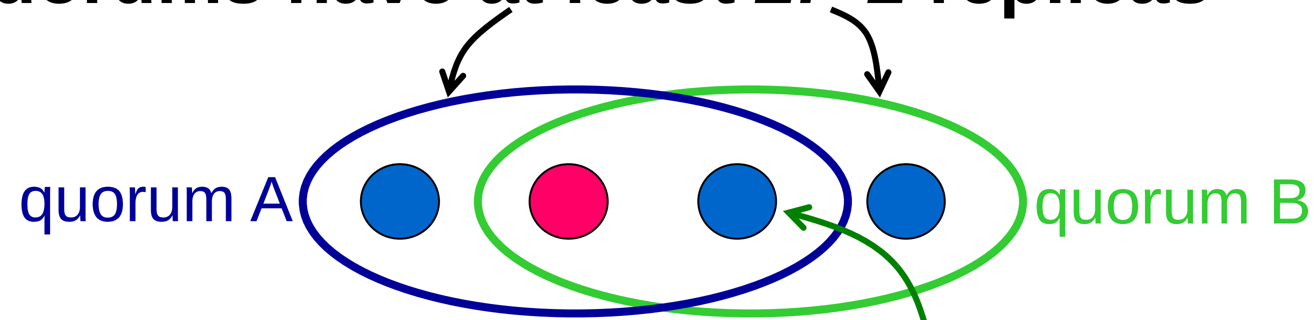
- View designates the primary replica



- Primary picks ordering
- Backups ensure primary behaves correctly
  - certify correct ordering
  - trigger view changes to replace faulty primary

Quorums and Certificates

quorums have at least *2f+1* replicas



3f+1 replicas

quorums intersect in at least one correct replica

- **Certificate ≡** set with messages from a quorum
- Algorithm steps are justified by certificates

Algorithm Components

- Normal case operation
- View changes
- Garbage collection
- Recovery

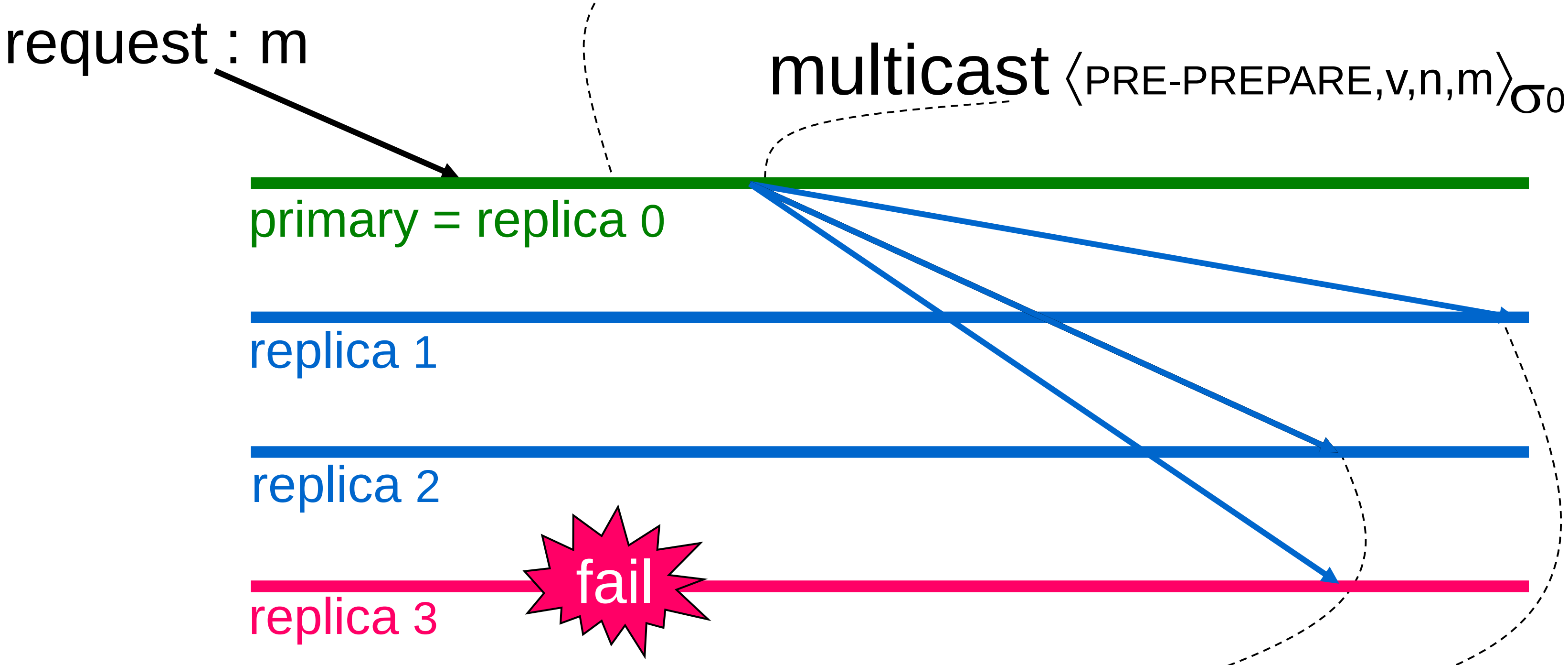
All have to be designed to work together

Normal Case Operation

- Three phase algorithm:
  - *pre-prepare* picks order of requests
  - *prepare* ensures order within views
  - *commit* ensures order across views
- Replicas remember messages in log
- Messages are authenticated
  - $\langle \bullet \rangle_{\sigma k}$ denotes a message sent by k

Pre-prepare Phase

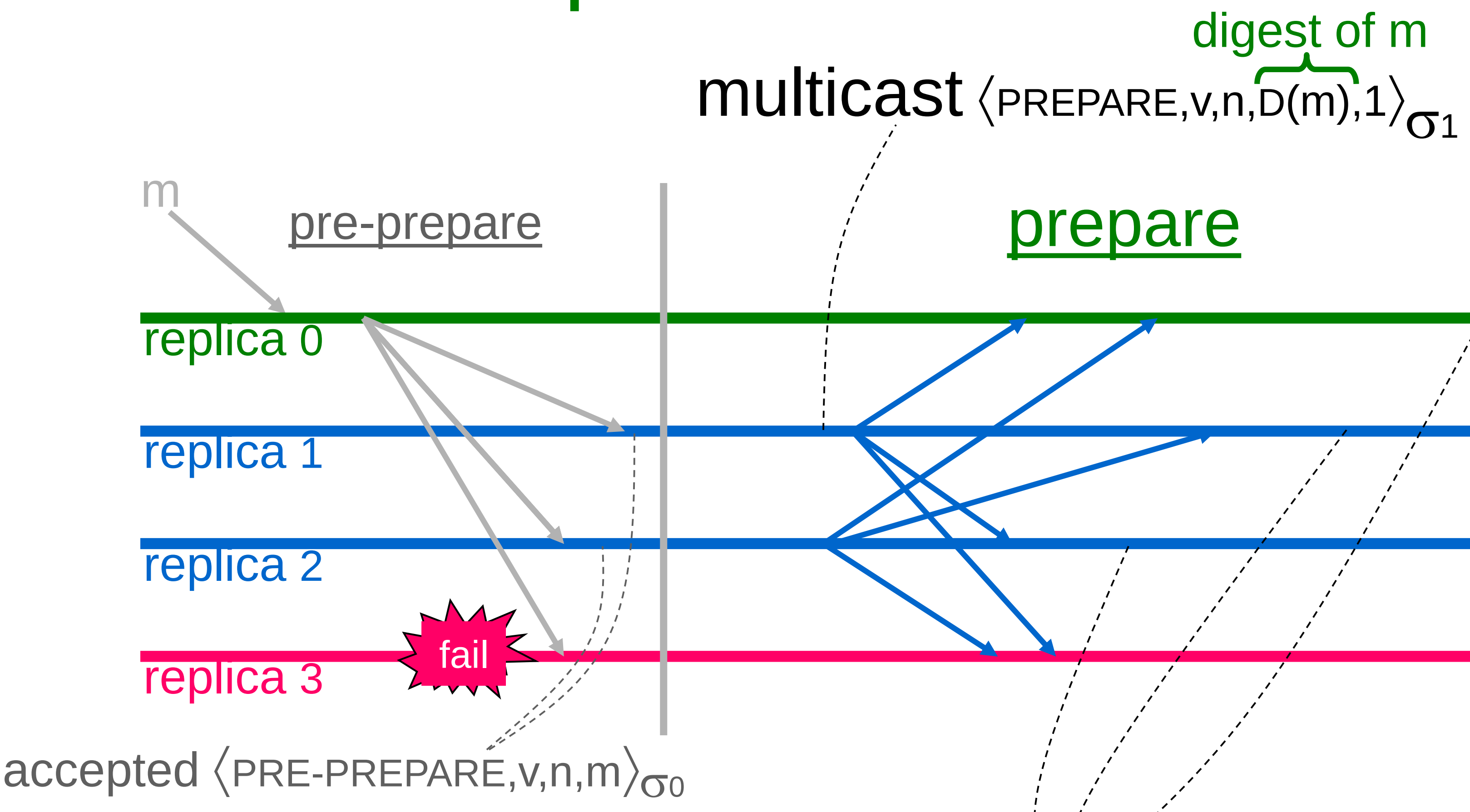
assign sequence number n to request m in view v



backups accept pre-prepare if:

- in view v
- never accepted pre-prepare for v,n with different request

Prepare Phase



all collect pre-prepare and *2f* matching prepares

P-certificate(m,v,n)

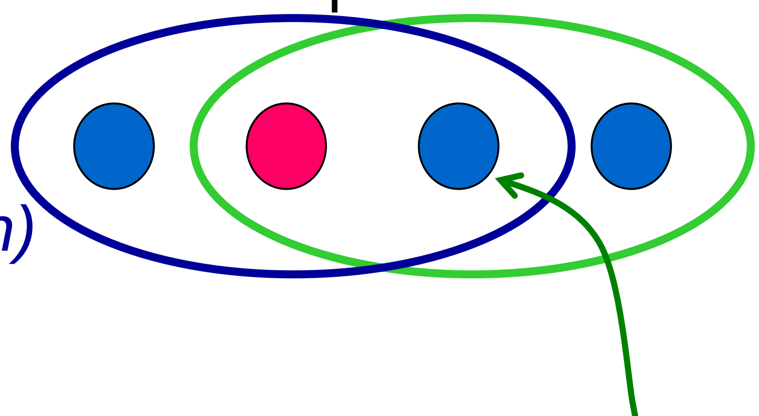
Order Within View

No *P-certificates* with the same view and sequence number and different requests

If it were false:

replicas

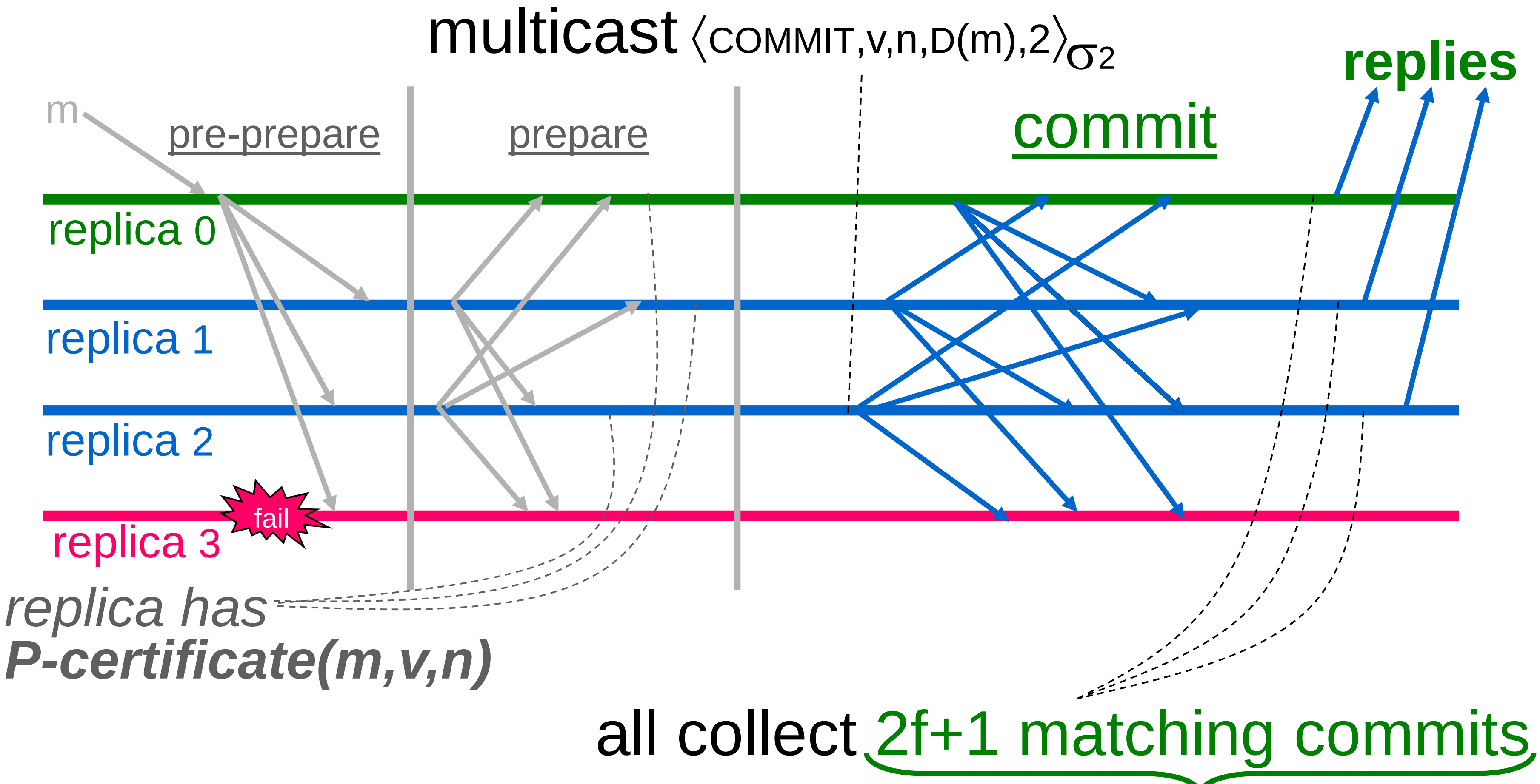
quorum for *P-certificate(m,v,n)*



quorum for *P-certificate(m’,v,n)*

one correct replica in common ⇒ m = m’

Commit Phase



C-certificate(m,v,n)

Request m executed after:

- having *C-certificate(m,v,n)*
- executing requests with sequence number less than n

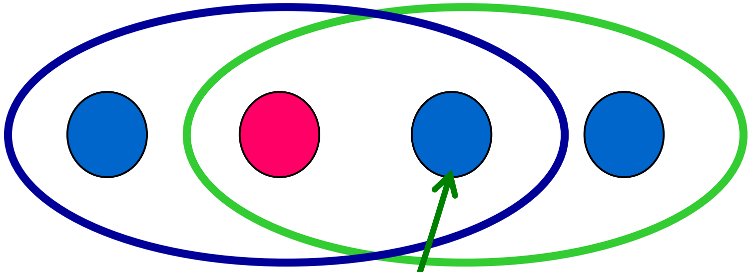
View Changes

- Provide liveness when primary fails:
  - timeouts trigger view changes
  - select new primary (≡ view number mod *3f+1*)
- But also need to:
  - preserve safety
  - ensure replicas are in the same view long enough
  - prevent denial-of-service attacks

View Change Safety

Goal: No *C-certificates* with the same sequence number and different requests

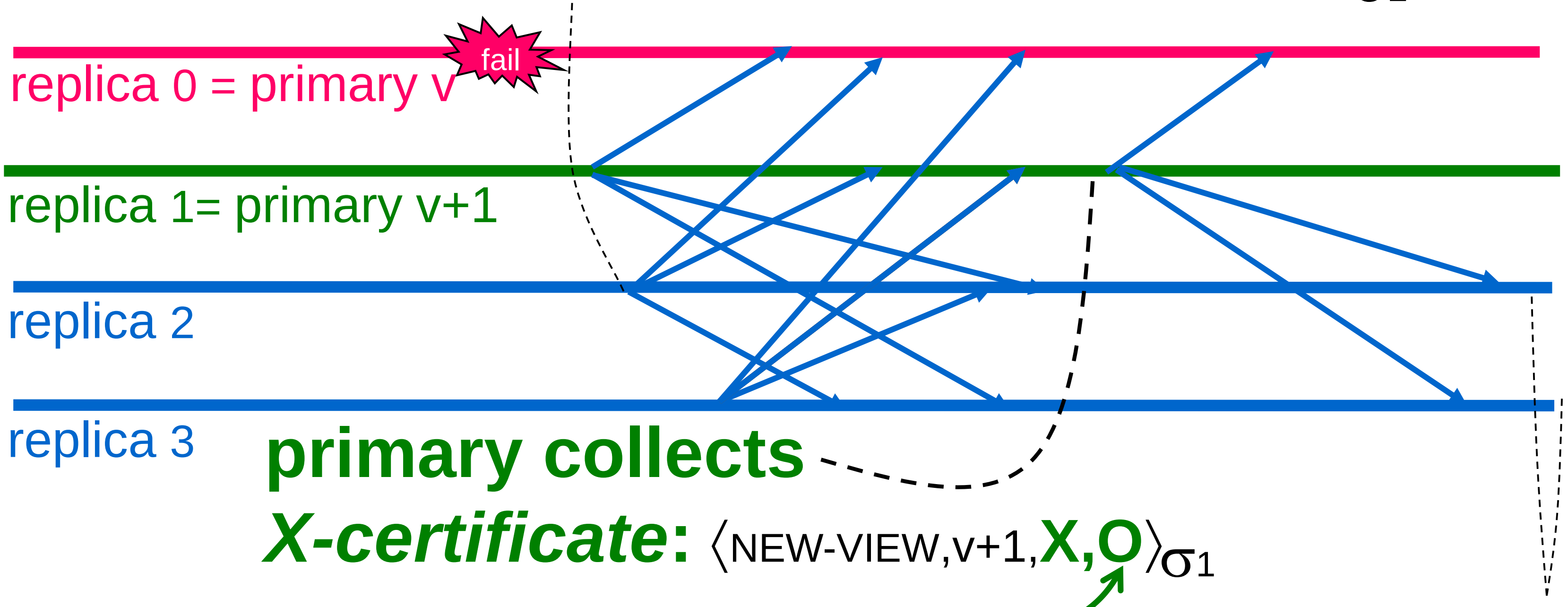
- Intuition: if replica has ***C-certificate(m,v,n)*** then



correct replica in Q has *P-certificate(m,v,n)*

View Change Protocol

send *P-certificates*: $\langle$VIEW-CHANGE,v+1,**P**,2$\rangle_{\sigma_2}$



primary collects
***X-certificate*:** $\langle$NEW-VIEW,v+1,**X,O**$\rangle_{\sigma_1}$

pre-prepares matching
***P-certificates* with highest views in X**

•pre-prepare for m,v+1,n in new-view

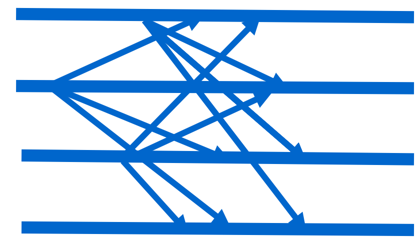
• Backups multicast prepare messages for m,v+1,n

backups multicast prepare messages for pre-prepares in O

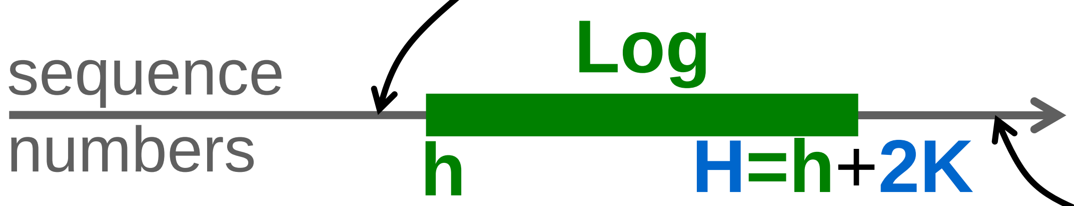
Garbage Collection

Truncate log with **certificate**:

- periodically checkpoint state (**K**)
- multicast $\langle \text{CHECKPOINT}, h, D(\text{checkpoint}), i \rangle_{\sigma_i}$
- all collect $\underbrace{\textit{2f+1}\text{ checkpoint messages}}_{\textbf{\textit{S-certificate(h,checkpoint)}}}$



discard messages and checkpoints



reject messages

send *S-certificate* and checkpoint in view-changes