Paxos

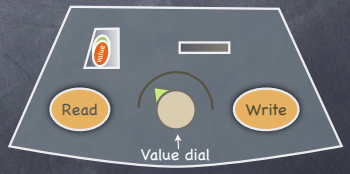
- Always safe
- Live during periods of synchrony
- Leader (primary) responsible for proposing the consensus value
- Features Dijkstra as a cheese inspector

Paxos

- Always safe
- Live during periods of synchrony
- Leader (primary) responsible for proposing the consensus value
- Features Dijkstra as a cheese inspector
- Somewhat popular…
  - Part-time Parliament [L98]
  - Frangipani [TML97]
  - Byzantine Paxos [CL99]
  - Disk Paxos [GL00]
  - Deconstructing Paxos [BDFG01]
  - Reconstructing Paxos [BDFG01]
  - Active Disk Paxos [CM02]
  - Separating Agreement & Execution [YMAD 03]
  - Byzantine Disk Paxos [ACKM04]
  - Fast Byzantine Paxos [MA05]
  - Fast Paxos [L05]
  - Hybrid Quorums [CMLRS06]
  - Chubby [B06]
  - Paxos Register [LCAA07]
  - Zyzzyva [KADCW07]
  - ….

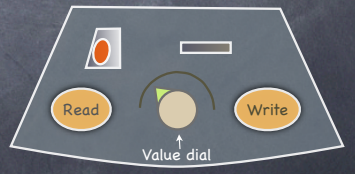
The Game of Paxos

- Processes are competing to write a value in a write-once register
- To learn the final value:
  1. Push the read button and examine the token that falls into the tray
  2. If the token is green, GAME OVER – the final value of the register is stamped on the token!
  3. If the token is red and stamped with a value, place the token in the slot, set the dial to the same value, and push the write button



The Game of Paxos

- Processes are competing to write a value in a write-once register
- To learn the final value:
  1. Push the read button and examine the token that falls into the tray
  2. If the token is green, GAME OVER – the final value of the register is stamped on the token!
  3. If the token is red and stamped with a value, place the token in the slot, set the dial to the same value, and push the write button
  4. If the token is red and not stamped, place the token in the slot, set the dial to any value, and push the write button



Quorum Systems

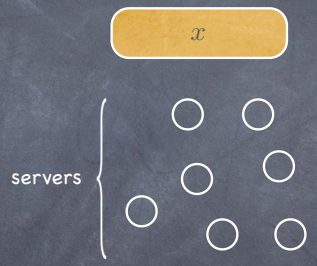
Given a set of servers $\mathcal{U}, |\mathcal{U}| = n$

a quorum system is a set $\mathcal{Q} \subseteq 2^{\mathcal{U}}$ such that

$$\forall Q_1, Q_2 \in \mathcal{Q} : Q_1 \cap Q_2 \neq \emptyset$$

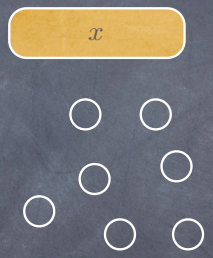
Each Q in $\mathcal{Q}$ is a quorum

A R/W Register



store at each server
a (v, ts) pair

A R/W Register

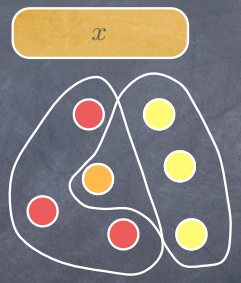


store at each server
a (v, ts) pair

Write(x, d)

- Ask servers in some Q for their ts
- Set $ts_c > \max(\{ts\} \cup$ any previous $ts_c)$
- Update some Q' with (d, ts_c)

A R/W Register



store at each server
a (v, ts) pair

Write(x, d)

- Ask servers in some Q for their ts
- Set $ts_c > \max(\{ts\} \cup$ any previous $ts_c)$
- Update some Q' with (d, ts_c)

Read(x)

- Ask servers in some Q for their (v, ts)
- Select most recent (v, ts)

System Model

- Universe U of servers, |U| = n
- Byzantine faulty servers
  - modeled as a non-empty fail-prone system $\mathcal{B} \subseteq 2^U$
  - no $B \in \mathcal{B}$ is contained in another
  - some $B \in \mathcal{B}$ contains all faulty servers
- Clients are correct (can be weakened)
- Point-to-point authenticated and reliable channels

A correct process q receives a message from another correct process p if and only if p sent it

Masking Quorum System

[Malkhi and Reiter, 1998]

A quorum system $\mathcal{Q}$ is a masking quorum system for a fail-prone system $\mathcal{B}$ if the following properties hold:

M-Consistency

$$\forall Q_1, Q_2 \in \mathcal{Q}\ \forall B_1, B_2 \in \mathcal{B} : (Q_1 \cap Q_2) \setminus B_1 \not\subseteq B_2$$

M-Availability

$$\forall B \in \mathcal{B}\ \exists Q \in \mathcal{Q} : B \cap Q = \emptyset$$

Dissemination Quorum System

A masking quorum system for self-verifying data

client can detect modification by faulty server

D-Consistency

$$\forall Q_1, Q_2 \in \mathcal{Q}\ \forall B \in \mathcal{B} : (Q_1 \cap Q_2) \not\subseteq B$$

D-Availability

$$\forall B \in \mathcal{B}\ \exists Q \in \mathcal{Q} : B \cap Q = \emptyset$$

f-threshold Masking Quorum Systems

$$\mathcal{B} = \{B \subseteq U : |B| = f\}$$

M-Consistency

$$\forall Q_1, Q_2 \in \mathcal{Q} : |Q_1 \cap Q_2| \geq 2f + 1$$

M-Availability

$$|Q| \leq n - f$$

$$\mathcal{Q} = \left\{Q \subseteq U : |Q| = \left\lceil \frac{n + 2f + 1}{2} \right\rceil\right\}$$

$$n \geq 4f + 1$$

D-Consistency

$$\forall Q_1, Q_2 \in \mathcal{Q} : |Q_1 \cap Q_2| \geq f + 1$$

D-Availability

$$|Q| \leq n - f$$

$$\mathcal{Q} = \left\{Q \subseteq U : |Q| = \left\lceil \frac{n + f + 1}{2} \right\rceil\right\}$$

$$n \geq 3f + 1$$

A safe read/write protocol

Client c executes:

Write(d)
→ Ask all servers for their current timestamp t
← Wait for answer from |Q| different servers
Set ts_c > max({t} ∪ any previous ts_c)
→ Send (d,ts_c) to all servers
← Wait for |Q| acknowledgments

Read()
→ Ask all servers for latest value/timestamp pair
← Wait for answer from |Q| different servers
Select most recent (v,ts) for which at least f + 1 answers agree (if any)

verifiable

A simple observation

Client c (with current threshold f) executes:

Write(d)
→ Ask all servers for their current timestamp t
← Wait for answer from |Q| different servers
Set ts_c > max({t} ∪ any previous ts_c)
→ Send (d,ts_c) to all servers
← Wait for |Q| acknowledgements

Read()
→ Ask all servers for latest value/timestamp pair
← Wait for answer from |Q| different servers
Select most recent (v,ts) for which at least f + 1 answers agree (if any)

(Asynchronous) Authenticated Reliable channels

A correct process q receives a message from another correct process p if and only if p sent it

A-Masking Quorum Systems

A quorum system $\mathcal{Q}$ is an a-masking quorum system for a fail-prone system B if the following properties hold for Q_r and Q_w:

AM-Consistency

$$\forall Q_r \in \mathcal{Q}_r \; \forall Q_w \in \mathcal{Q}_w \; \forall B_1, B_2 \in \mathcal{B}$$
$$(Q_r \cap Q_w) \setminus B_1 \not\subseteq B_2 :$$

AM-Availability

$$\forall B \in \mathcal{B} \; \exists Q_r \in \mathcal{Q}_r : B \cap Q_r = \emptyset$$

Tradeoffs

best known n	confirmable	non-confirmable
self-verifying	$3f+1$	$2f+1$
generic	$4f+1$	$3f+1$

Tradeoffs

best known n	confirmable	non-confirmable
self-verifying and generic	$3f+1$	$2f+1$

Lower bound: never two rows again!

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"
- BASE (RCL 01)
  - uses abstraction to reduce correlated faults

The Setup

System Model
- Asynchronous system
- Unreliable channels

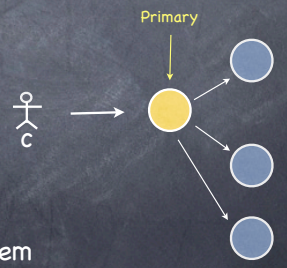
Crypto
- Public/Private key pairs
- MACs
- Collision-resistant hashes
- Unbreakable

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

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

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 non-faulty replicas

PBFT: The site map

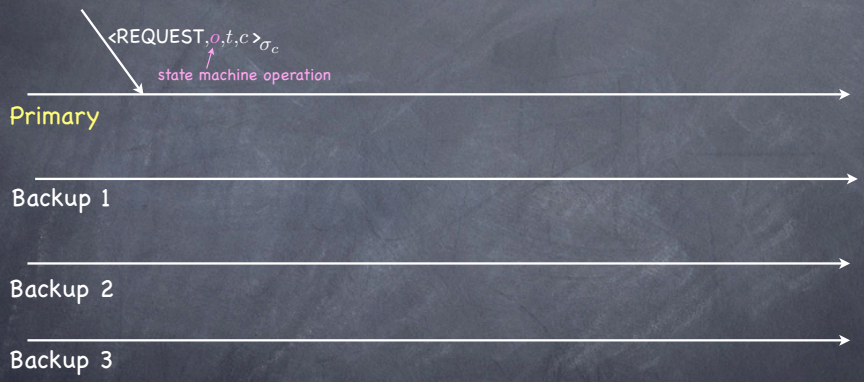
- Normal operation
  - How the protocol works in the absence of failures - hopefully, the common case
- View changes
  - How to depose a faulty primary and elect a new one
- Garbage collection
  - How to reclaim the storage used to keep certificates
- Recovery
  - How to make a faulty replica behave 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

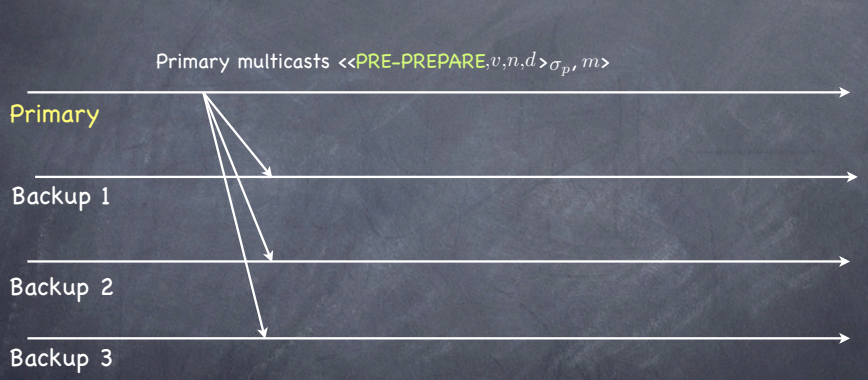


Client issues request

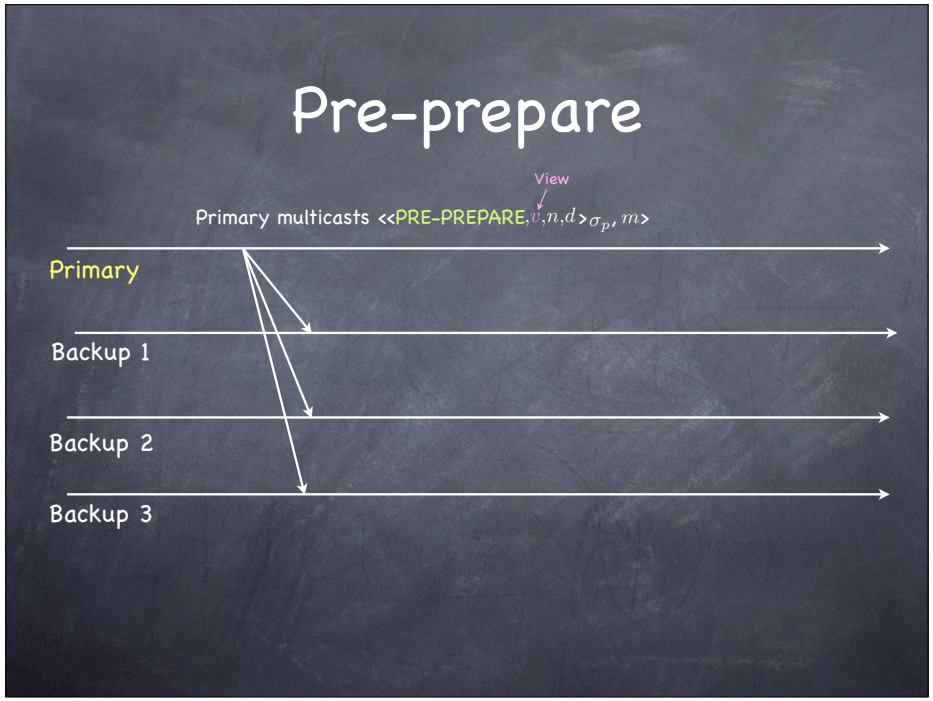
Client issues request

Client issues request

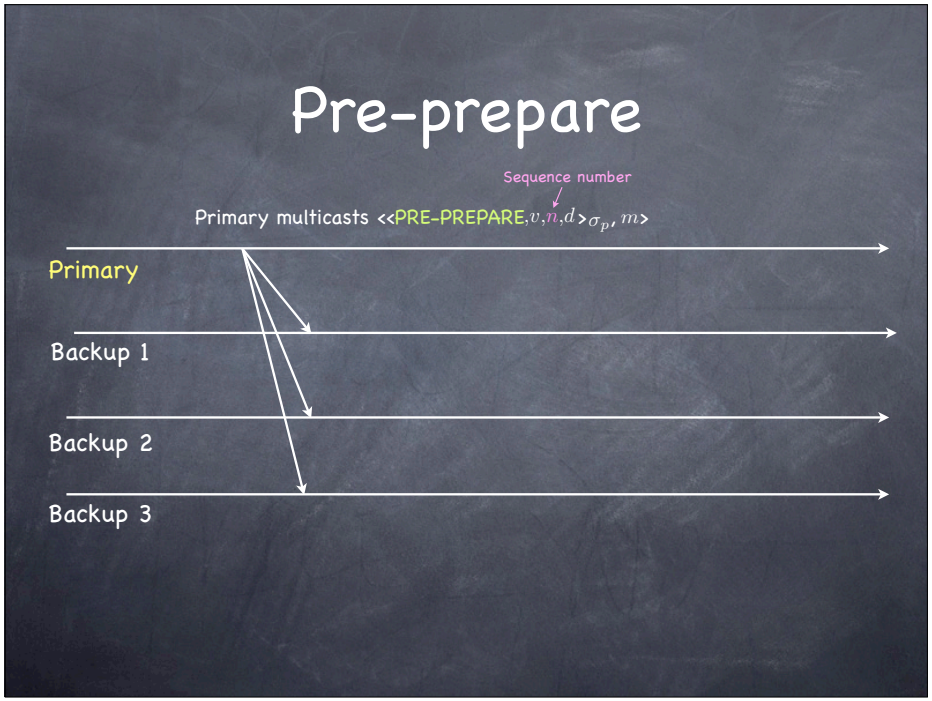
Pre-prepare



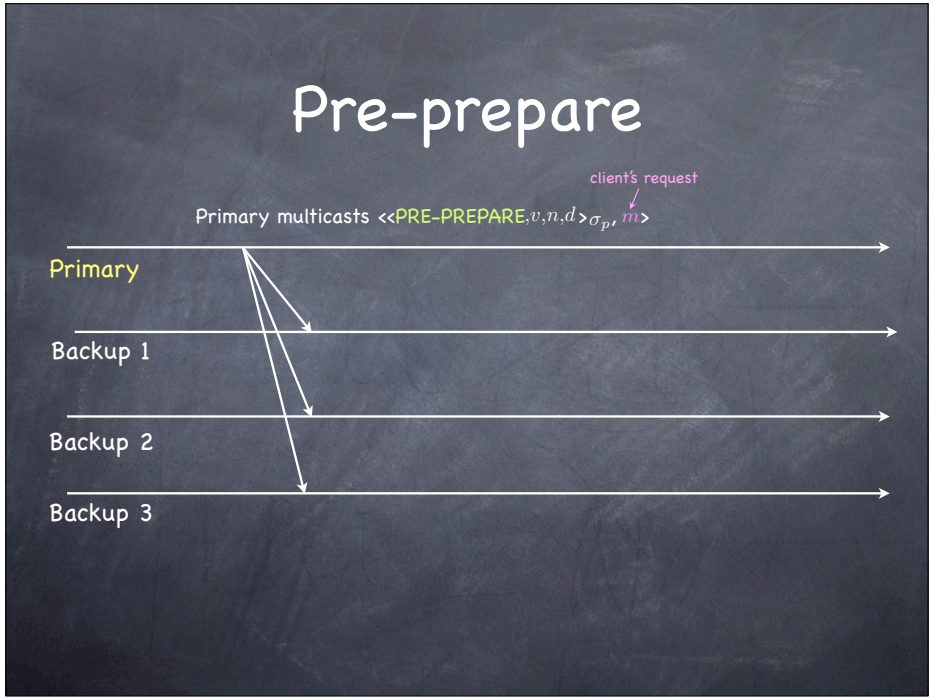
Pre-prepare



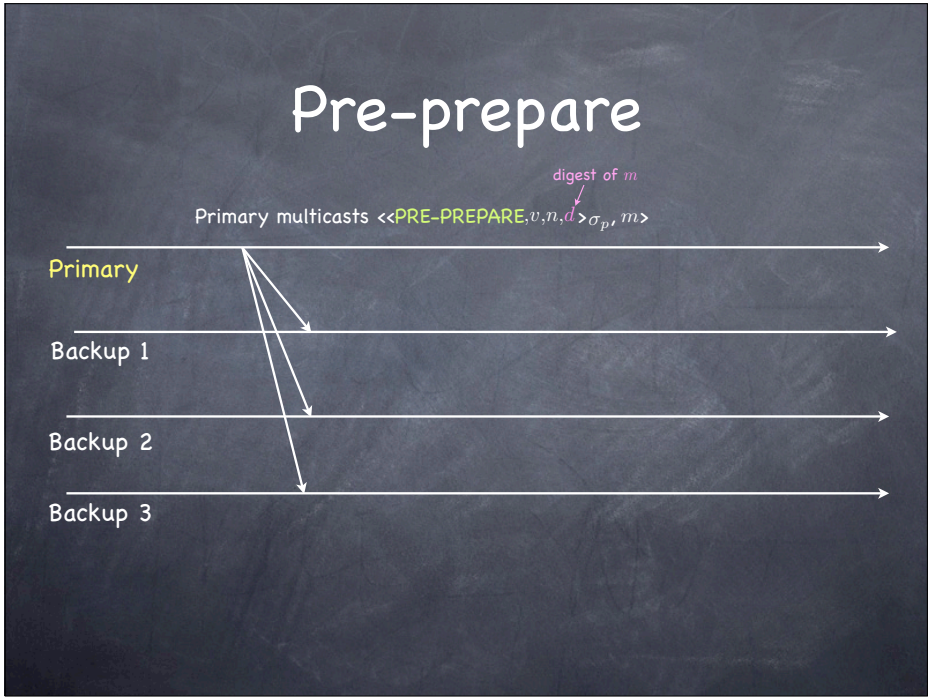
Pre-prepare



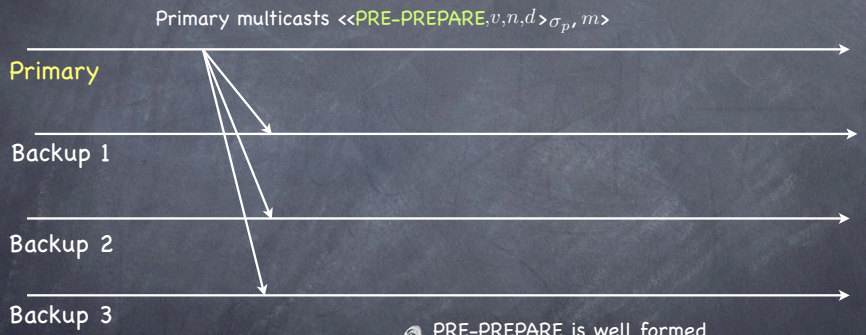
Pre-prepare



Pre-prepare



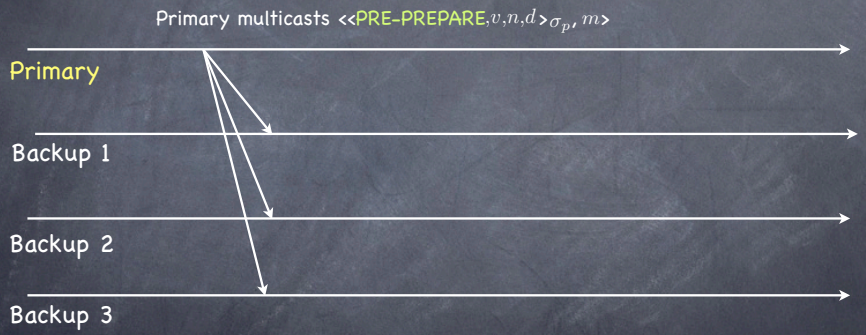
Pre-prepare



Correct backup i accepts PRE-PREPARE if:

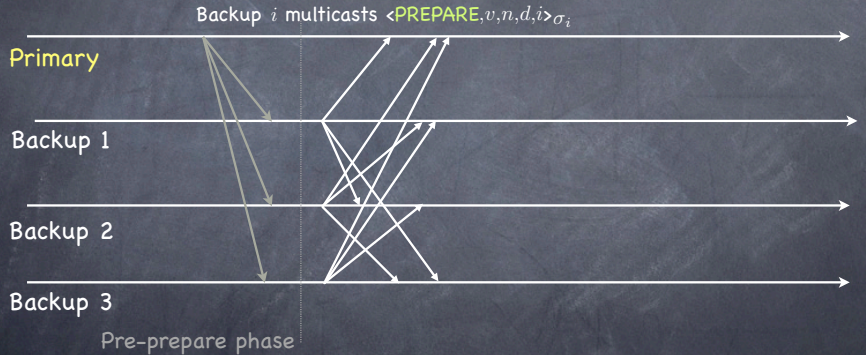
- PRE-PREPARE is well formed
- i is in view v
- i has not accepted another PRE-PREPARE for v, n with a different d
- n is between two water-marks L and H (to prevent sequence number exhaustion)

Pre-prepare



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

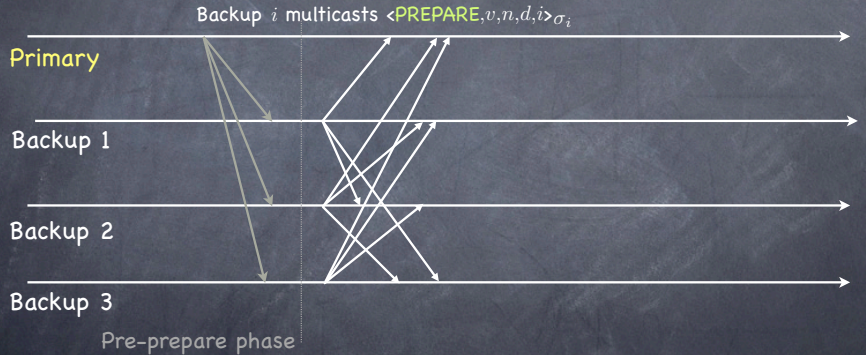
Prepare



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

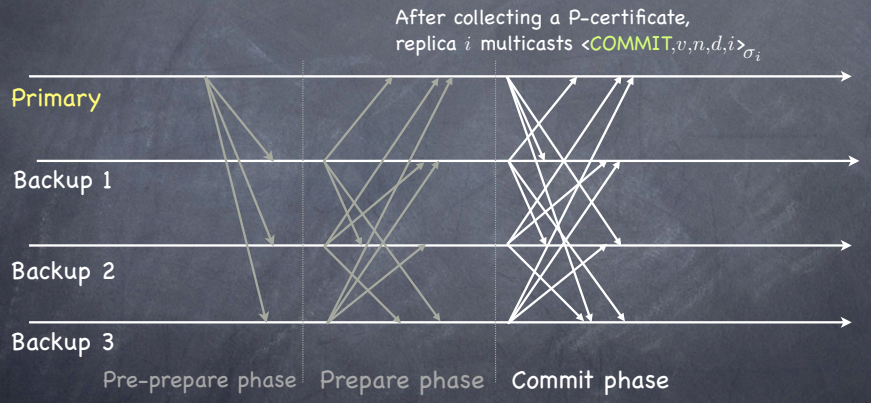
Prepare Certificate

- 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

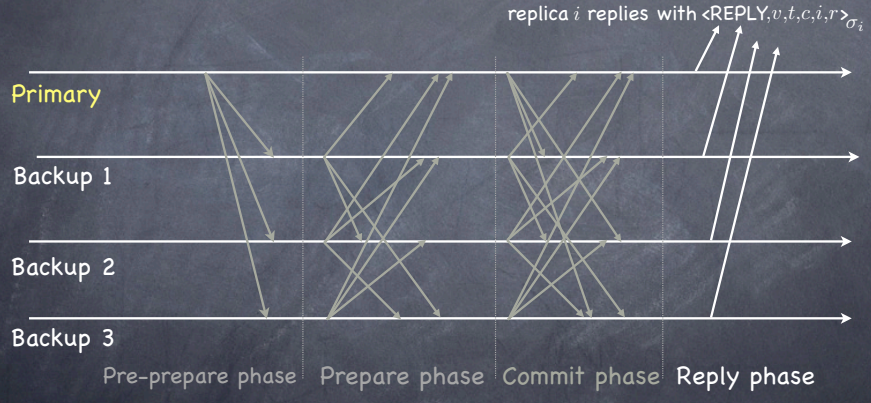
Commit



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



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 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 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 in $\mathcal{V}$ a valid VIEW-CHANGE messages for $v+1$
  - it can verify locally that $\mathcal{O}$ is correct (repeating the primary's computation)
- Adds all entries in $\mathcal{O}$ to its log (so did $\hat{p}$!)
- Multicasts a PREPARE for each message in $\mathcal{O}$
- Adds all PREPARE to log and enters new view

Garbage Collection

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

View change, revisited

- A disgruntled backup multicasts

<VIEW-CHANGE,$v+1$,n,s,$\mathcal{C}$,$\mathcal{P}$,i>$_{\sigma_i}$

View change, revisited

- A disgruntled backup multicasts

<VIEW-CHANGE,$v+1$,n,s,$\mathcal{C}$,$\mathcal{P}$,i>$_{\sigma_i}$

sequence number of last stable checkpoint

View change, revisited

- A disgruntled backup multicasts

<VIEW-CHANGE,$v+1$,n,s,$\mathcal{C}$,$\mathcal{P}$,i>$_{\sigma_i}$

last stable checkpoint

View change, revisited

- A disgruntled backup multicasts

$$\langle \text{VIEW-CHANGE}, v+1, n, s, \mathcal{C}, \mathcal{P}, i \rangle_{\sigma_i}$$

$\mathcal{C}$: stable certificate for s

View change, revisited

- A disgruntled backup multicasts

$$\langle \text{VIEW-CHANGE}, v+1, n, s, \mathcal{C}, \mathcal{P}, i \rangle_{\sigma_i}$$

$\mathcal{P}$: P certificates for requests with sequence number $> n$

View change, revisited

- A disgruntled backup multicasts

$$\langle \text{VIEW-CHANGE}, v+1, n, s, \mathcal{C}, \mathcal{P}, i \rangle_{\sigma_i}$$

- $\hat{p}$ multicasts

$$\langle \text{NEW-VIEW}, v+1, n, \mathcal{V}, \mathcal{O}, \mathcal{N} \rangle_{\sigma_{\hat{p}}}$$

n: sequence number of last stable checkpoint

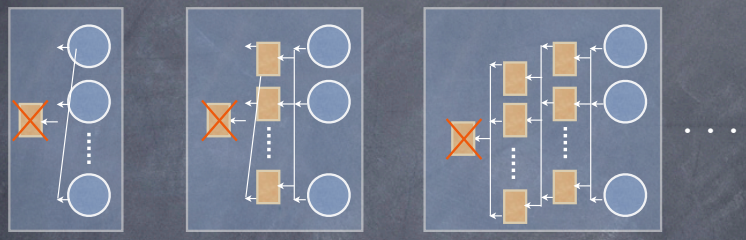
Citius, Altius, Fortius: Towards deployable BFT

- Reducing the costs of BFT replication
- Addressing confidentiality
- Reducing complexity

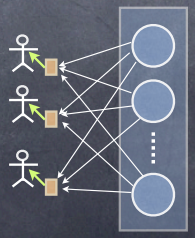
Reducing the costs of BFT replication

- Who cares? Machines are cheap...
  - Replicas should fail independently in software, not just hardware
  - How many independently failing implementations of non-trivial services do actually exist?

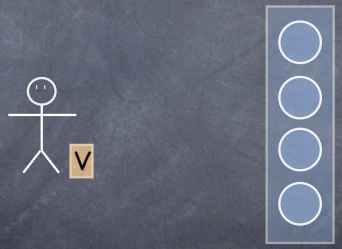
Back the old conundrum



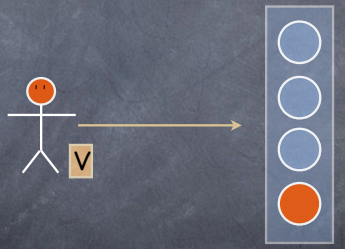
A: voter and client share fate!



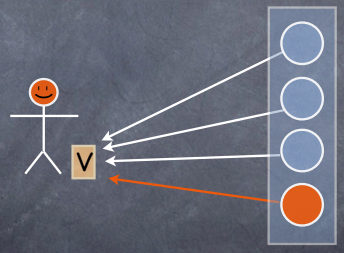
Not so fast...



Not so fast...



Not so fast...



No confidentiality!

Rethinking State Machine Replication

Not Agreement + Order

but rather Agreement on Order + Execution

Rethinking State Machine Replication

Not Agreement + Order

but rather Agreement on Order + Execution

Benefits:

- 3f+1 state machine replicas

Rethinking State Machine Replication

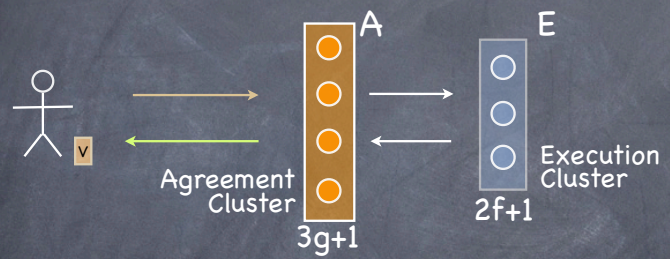
Not Agreement + Order

but rather Agreement on Order + Execution

Benefits:

- 3f+1 state machine replicas
- Replication helps confidentiality

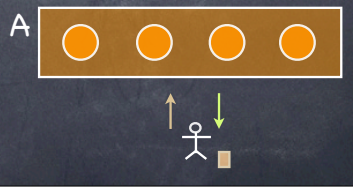
Separation reduces replication costs



- Not all nodes are created equal!
  - Nodes in E: expensive
    - (different across applications and within same application)
  - Nodes in A: cheap
    - (simple and reusable across applications)

Separation enables confidentiality

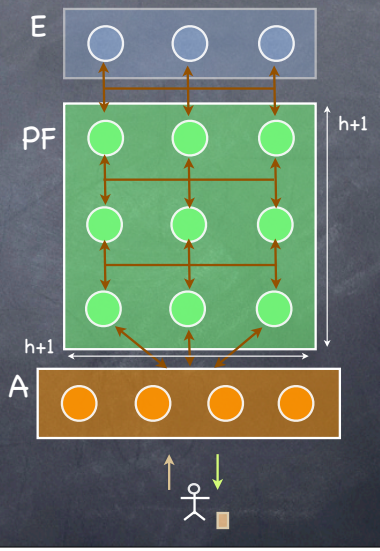
Three design principles:



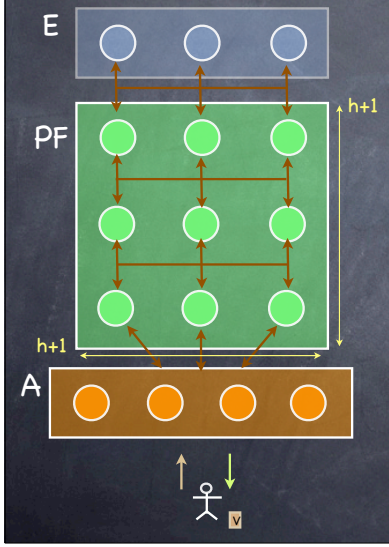
Separation enables confidentiality

Three design principles:

1. Use redundant filters for fault tolerance
2. Restrict communication
3. Eliminate nondeterminism

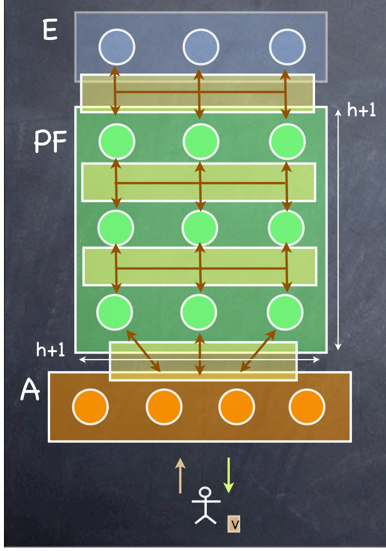


The Privacy Firewall



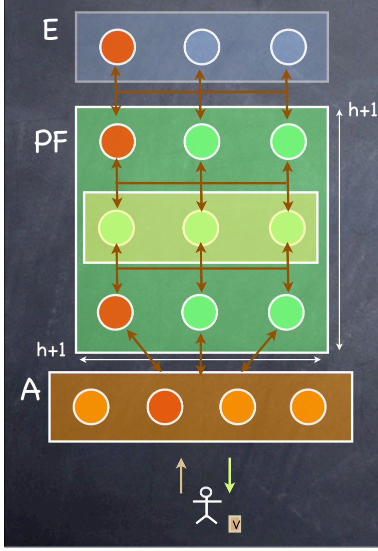
- $(h+1)^2$-filter grid tolerates h Byzantine failures
- A filter only communicates with filters immediately above or below
- Each filter checks both reply and request certificates
- Safe
  - h+1 rows → one is correct
- Live
  - h+1 columns → one is correct
- Restricts nondeterminism
  - threshold cryptography for replies
  - cluster A locks rsn
  - controlled message retransmission

Inside the PF



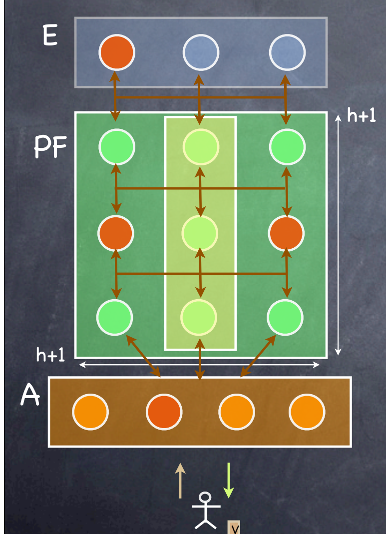
- $(h+1)^2$-filter grid tolerates h Byzantine failures
- A filter only communicates with filters immediately above or below
- Each filter checks both reply and request certificates
- Safe
  - h+1 rows → one is correct
- Live
  - h+1 columns → one is correct
- Restricts nondeterminism
  - threshold cryptography for replies
  - cluster A locks rsn
  - controlled message retransmission

Inside the PF



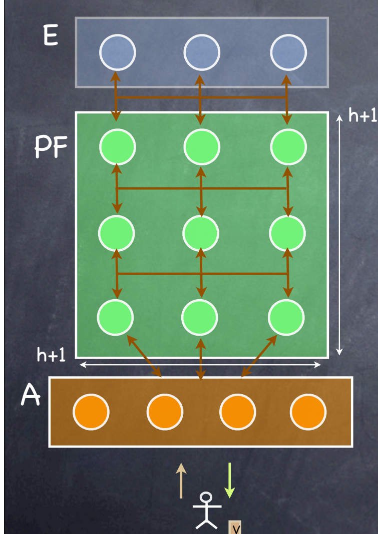
- $(h+1)^2$-filter grid tolerates h Byzantine failures
- A filter only communicates with filters immediately above or below
- Each filter checks both reply and request certificates
- Safe
  - h+1 rows → one is correct
- Live
  - h+1 columns → one is correct
- Restricts nondeterminism
  - threshold cryptography for replies
  - cluster A locks rsn
  - controlled message retransmission

Inside the PF



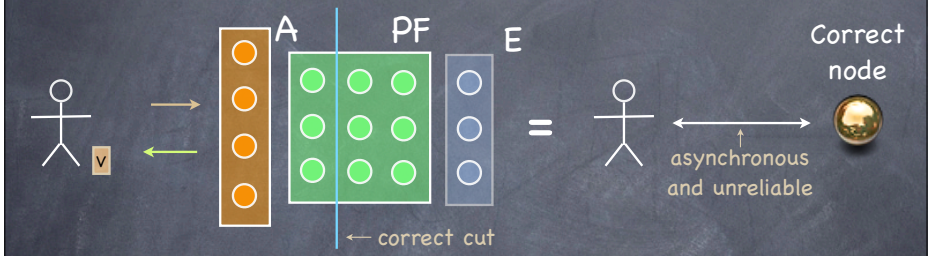
- $(h+1)^2$-filter grid tolerates h Byzantine failures
- A filter only communicates with filters immediately above or below
- Each filter checks both reply and request certificates
- Safe
  - h+1 rows → one is correct
- Live
  - h+1 columns → one is correct
- Restricts nondeterminism
  - threshold cryptography for replies
  - cluster A locks rsn
  - controlled message retransmission

Inside the PF



- $(h+1)^2$-filter grid tolerates h Byzantine failures
- A filter only communicates with filters immediately above or below
- Each filter checks both reply and request certificates
- Safe
  - h+1 rows → one is correct
- Live
  - h+1 columns → one is correct
- Restricts nondeterminism
  - threshold cryptography for replies
  - cluster A locks rsn
  - controlled message retransmission

Privacy Firewall guarantees



Output-set confidentiality

Output sequence through correct cut is a legal sequence of outputs produced by a correct node accessed trough an asynchronous, unreliable link

An exciting decade

- State machine replication
  - Practical Byzantine Fault Tolerance
  - Reuse of existing (non-deterministic) implementations [SOSP 01]
  - Reduced replication cost [SOSP 03]
  - Low-overhead confidentiality [SOSP 03]
  - High throughput [DSN 04]
  - Applications: Farsite[OSDI 02], Oceanstore [FAST 03]
- Quorums
  - Fault Scalability (Q/U) [SOSP 05]
  - Improved performance under contention (HQ) [OSDI 06]