Fault-Tolerant Distributed Transactions on Blockchain

Practical Byzantine Fault-Tolerant Consensus

Suyash Gupta

Jelle Hellings

Mohammad Sadoghi

UCDAVIS
UNIVERSITY OF CALIFORNIA

ExpoLab
Creativity Unfolded

McMaster University

A Resilient Database Management System (RDBMS)

Client

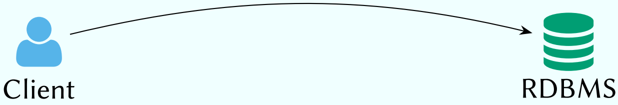
RDBMS

ParentOf	
Parent	Child
Alice	Carol
Bob	Carol
Carol	Dan
Carol	Eve
Dan	Faythe

The *RDBMS* should be resilient,
while serving as a *single coherent system* in a *transparent* way.

A Resilient Database Management System (RDBMS)

$\tau =$ "`SELECT Child`
`FROM ParentOf`
`WHERE parent = 'Carol';`"

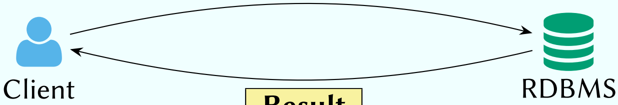


ParentOf	
Parent	Child
Alice	Carol
Bob	Carol
Carol	Dan
Carol	Eve
Dan	Faythe

The *RDBMS* should be resilient,
while serving as a *single coherent system* in a *transparent* way.

A Resilient Database Management System (RDBMS)

$\tau =$ "`SELECT Child`
`FROM ParentOf`
`WHERE parent = 'Carol';`"



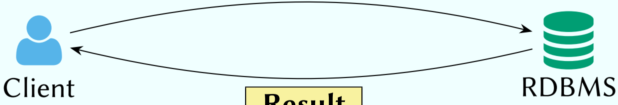
ParentOf	
Parent	**Child**
Alice	Carol
Bob	Carol
Carol	Dan
Carol	Eve
Dan	Faythe

Result
Child
Dan
Eve

The *RDBMS* should be resilient,
while serving as a *single coherent system* in a *transparent* way.

A Resilient Database Management System (RDBMS)

$\tau =$ “`SELECT Child`
`FROM ParentOf`
`WHERE parent = ‘Carol’;`”



ParentOf	
Parent	**Child**
Alice	Carol
Bob	Carol
Carol	Dan
Carol	Eve
Dan	Faythe

Result
Child
Dan
Eve

Reminder: Deterministic execution

All replicas in the RDBMS must perform the same execution of every transaction. E.g.,

$$\tau = \text{“Remove a child of Carol from the ParentOf table,”}$$

should result in all replicas removing the same child!

A Resilient RDBMS: What Can Go Wrong?

We assume *malicious* participation!

Malicious replicas can …

- try to insert *forged* transactions into the RDBMS;
- try to prevent *some* clients from using the RDBMS;
- try to send *invalid results* to clients using the RDBMS;
- try to *interfere* with the working of other replicas of the RDBMS;
- try to *disrupt* the consensus used by the RDBMS.

A *Practical* Definition of Consensus for Client-Server Services

Each replica $\textsc{q} \in \mathfrak{R}$ maintains an append-only *ledger* $\mathcal{L}_{\textsc{q}}$
(representing a sequence of *client transactions*).
A *consensus protocol* operates in rounds $\rho = 0, 1, 2, 3, \dots$ that each satisfy:

Termination Eventually, each good replica $\textsc{r} \in \mathcal{G}$ will append a single client transaction τ to their ledger such that: after round ρ, we have $\mathcal{L}_{\textsc{r}}[\rho] = \tau$.

Non-divergence If good replicas $\textsc{r}_1, \textsc{r}_2 \in \mathcal{G}$ appended τ_1 and τ_2 to their ledger in round ρ, then $\tau_1 = \tau_2$.

Validity If good replica $\textsc{r} \in \mathcal{G}$ appended τ to its ledger, then τ is requested by some client.

A *Practical* Definition of Consensus for Client-Server Services

Each replica $\textsc{q} \in \mathfrak{R}$ maintains an append-only *ledger* $\mathcal{L}_{\textsc{q}}$ (representing a sequence of *client transactions*).

A *consensus protocol* operates in rounds $\rho = 0, 1, 2, 3, \ldots$ that each satisfy:

Termination Eventually, each good replica $\textsc{r} \in \mathcal{G}$ will append a single client transaction τ to their ledger such that: after round ρ, we have $\mathcal{L}_{\textsc{r}}[\rho] = \tau$.

Non-divergence If good replicas $\textsc{r}_1, \textsc{r}_2 \in \mathcal{G}$ appended τ_1 and τ_2 to their ledger in round ρ, then $\tau_1 = \tau_2$.

Validity If good replica $\textsc{r} \in \mathcal{G}$ appended τ to its ledger, then τ is requested by some client.

Response If good replica $\textsc{r} \in \mathcal{G}$ appended τ to its ledger in round ρ, then the client that requested τ will receive the result of executing τ.

A *Practical* Definition of Consensus for Client-Server Services

Each replica $\textsc{q} \in \mathfrak{R}$ maintains an append-only *ledger* $\mathcal{L}_{\textsc{q}}$ (representing a sequence of *client transactions*).

A *consensus protocol* operates in rounds $\rho = 0, 1, 2, 3, \dots$ that each satisfy:

Termination Eventually, each good replica $\textsc{r} \in \mathcal{G}$ will append a single client transaction τ to their ledger such that: after round ρ, we have $\mathcal{L}_{\textsc{r}}[\rho] = \tau$.

Non-divergence If good replicas $\textsc{r}_1, \textsc{r}_2 \in \mathcal{G}$ appended τ_1 and τ_2 to their ledger in round ρ, then $\tau_1 = \tau_2$.

Validity If good replica $\textsc{r} \in \mathcal{G}$ appended τ to its ledger, then τ is requested by some client.

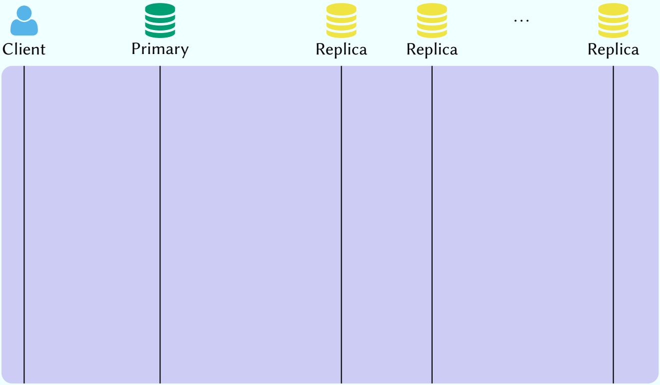
Response If good replica $\textsc{r} \in \mathcal{G}$ appended τ to its ledger in round ρ, then the client that requested τ will receive the result of executing τ.

Service If a good client requests τ, then eventually a good replica will append τ to its ledger.

Primary-Backup Replication

Primary Coordinates consensus: propose the order of transactions to replicate.

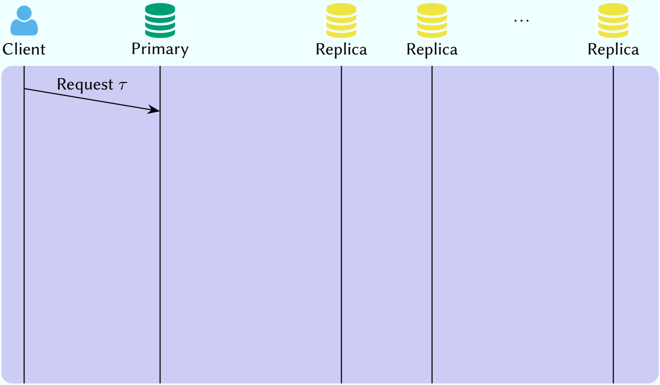
Backup Accept proposals and verifies behavior of the primary.



Primary-Backup Replication

Primary Coordinates consensus: propose the order of transactions to replicate.

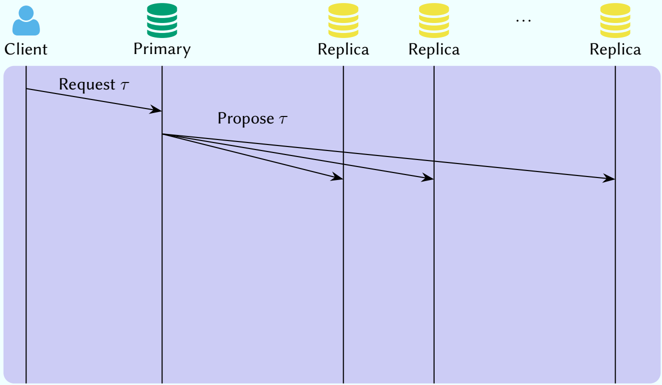
Backup Accept proposals and verifies behavior of the primary.



Primary-Backup Replication

Primary Coordinates consensus: propose the order of transactions to replicate.

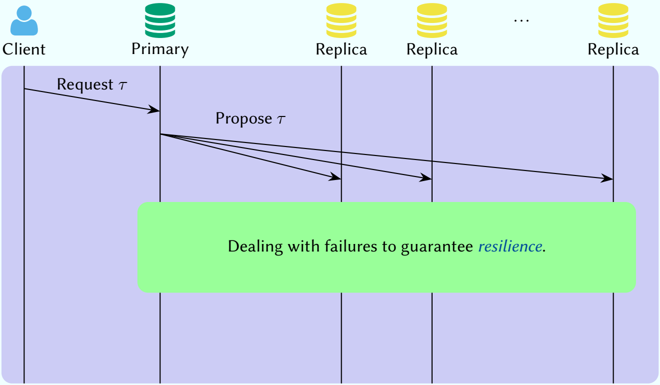
Backup Accept proposals and verifies behavior of the primary.



Primary-Backup Replication

Primary Coordinates consensus: propose the order of transactions to replicate.

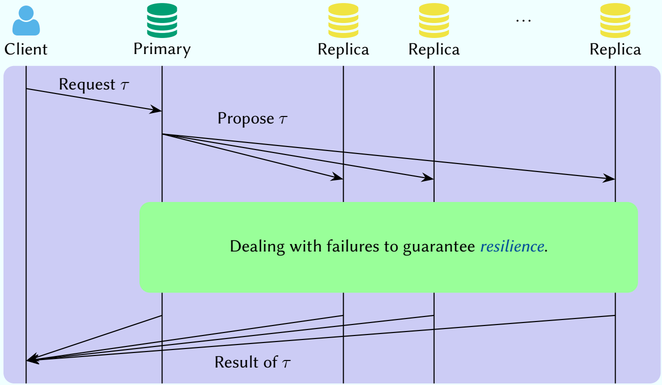
Backup Accept proposals and verifies behavior of the primary.



Primary-Backup Replication

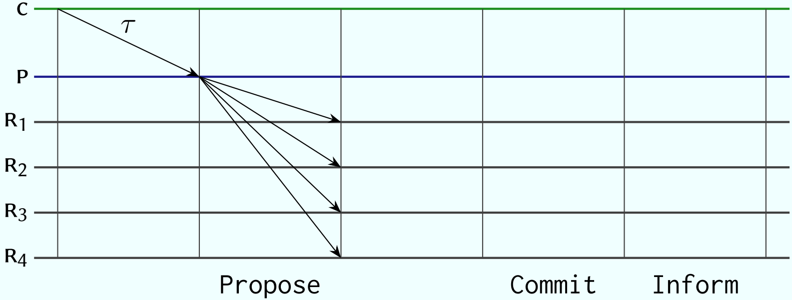
Primary Coordinates consensus: propose the order of transactions to replicate.

Backup Accept proposals and verifies behavior of the primary.



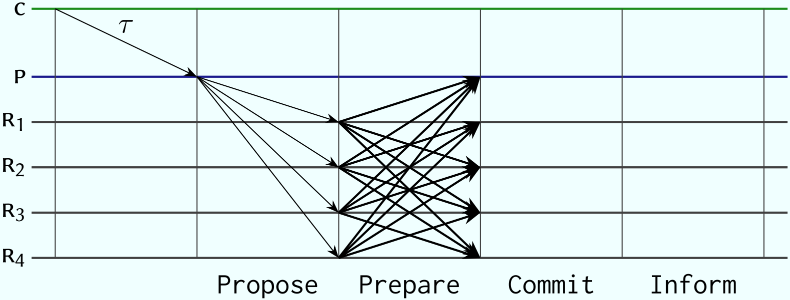
Primary-Backup Replication: Dealing with Byzantine Failures

Enforce "If good replicas *pledge*, then they all should do so for the same transaction."



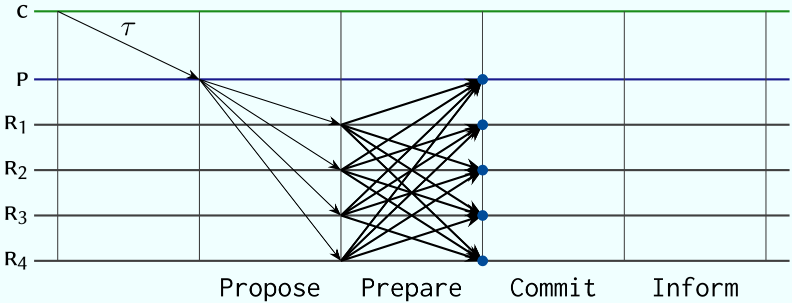
Primary-Backup Replication: Dealing with Byzantine Failures

Enforce "If good replicas *pledge*, then they all should do so for the same transaction."



Primary-Backup Replication: Dealing with Byzantine Failures

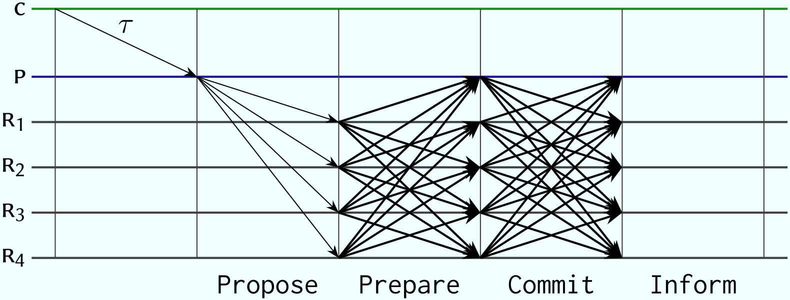
Enforce "If good replicas *pledge*, then they all should do so for the same transaction."



- Replicas *pledge* only if they receive *sufficient* **matching** Prepare-messages.

Primary-Backup Replication: Dealing with Byzantine Failures

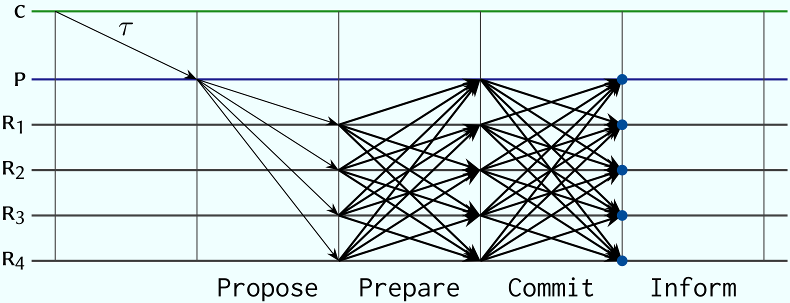
Enforce "If good replicas *pledge*, then they all should do so for the same transaction."



- Replicas *pledge* only if they receive *sufficient* **matching** Prepare-messages.

Primary-Backup Replication: Dealing with Byzantine Failures

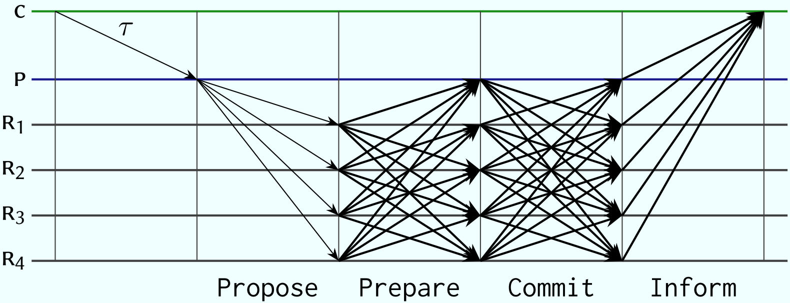
Enforce "If good replicas *pledge*, then they all should do so for the same transaction."



- Replicas *pledge* only if they receive *sufficient* **matching** `Prepare`-messages.
- Replicas *commit* only if they receive *sufficient* **matching** `Commit`-messages.

Primary-Backup Replication: Dealing with Byzantine Failures

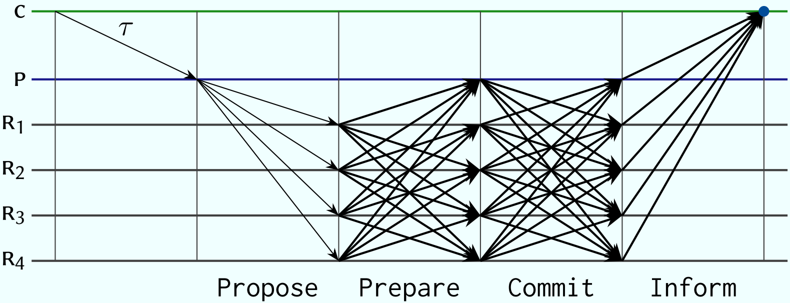
Enforce "If good replicas *pledge*, then they all should do so for the same transaction."



- Replicas *pledge* only if they receive *sufficient* **matching** `Prepare`-messages.
- Replicas *commit* only if they receive *sufficient* **matching** `Commit`-messages.

Primary-Backup Replication: Dealing with Byzantine Failures

Enforce "If good replicas *pledge*, then they all should do so for the same transaction."



- Replicas *pledge* only if they receive *sufficient* **matching** `Prepare`-messages.
- Replicas *commit* only if they receive *sufficient* **matching** `Commit`-messages.
- Client *observes* outcome only if they receive *sufficient* **matching** `Inform` messages.

Primary-Backup Replication: Dealing with Byzantine Failures

Enforce "If good replicas *pledge*, then they all should do so for the same transaction."

What is a *sufficient* amount of **matching** Prepare-messages

Assume we received matching Prepare messages from $\mathbf{m} \leq \mathbf{nf} = \mathbf{n} - \mathbf{f}$ replicas.

Primary-Backup Replication: Dealing with Byzantine Failures

Enforce "If good replicas *pledge*, then they all should do so for the same transaction."

What is a *sufficient* amount of **matching** Prepare-messages

Assume we received matching Prepare messages from $\mathbf{m} \leq \mathbf{nf} = \mathbf{n} - \mathbf{f}$ replicas.

- $\mathbf{f}$-out-of-$\mathbf{m}$ replicas can be Byzantine and send out *several* Prepare messages.
- $\mathbf{n} - \mathbf{m}$ replicas could have received *different* proposals from the primary.

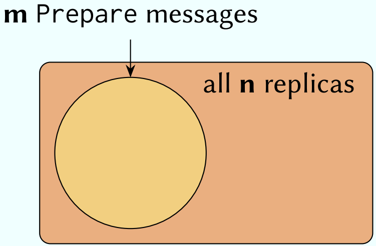
Primary-Backup Replication: Dealing with Byzantine Failures

Enforce "If good replicas *pledge*, then they all should do so for the same transaction."

What is a *sufficient* amount of **matching** Prepare-messages

Assume we received matching Prepare messages from $\mathbf{m} \leq \mathbf{nf} = \mathbf{n} - \mathbf{f}$ replicas.

- $\mathbf{f}$-out-of-$\mathbf{m}$ replicas can be Byzantine and send out *several* Prepare messages.
- $\mathbf{n} - \mathbf{m}$ replicas could have received *different* proposals from the primary.



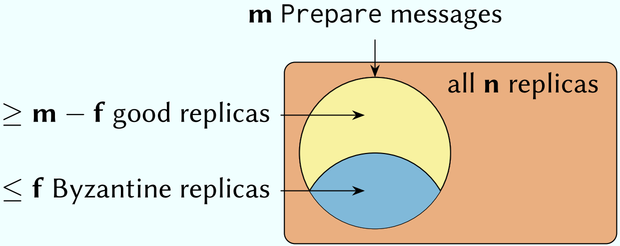
Primary-Backup Replication: Dealing with Byzantine Failures

Enforce "If good replicas *pledge*, then they all should do so for the same transaction."

What is a *sufficient* amount of **matching** Prepare-messages

Assume we received matching Prepare messages from $\mathbf{m} \leq \mathbf{nf} = \mathbf{n} - \mathbf{f}$ replicas.

- $\mathbf{f}$-out-of-$\mathbf{m}$ replicas can be Byzantine and send out *several* Prepare messages.
- $\mathbf{n} - \mathbf{m}$ replicas could have received *different* proposals from the primary.



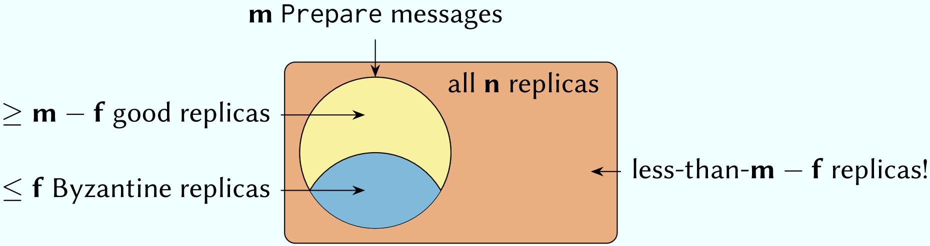
Primary-Backup Replication: Dealing with Byzantine Failures

Enforce "If good replicas *pledge*, then they all should do so for the same transaction."

What is a *sufficient* amount of **matching** Prepare-messages

Assume we received matching Prepare messages from $\mathbf{m} \leq \mathbf{nf} = \mathbf{n} - \mathbf{f}$ replicas.

- $\mathbf{f}$-out-of-$\mathbf{m}$ replicas can be Byzantine and send out *several* Prepare messages.
- $\mathbf{n} - \mathbf{m}$ replicas could have received *different* proposals from the primary.



Primary-Backup Replication: Dealing with Byzantine Failures

Enforce "If good replicas *pledge*, then they all should do so for the same transaction."

What is a *sufficient* amount of **matching** Prepare-messages

Assume we received matching Prepare messages from $\mathbf{m} \leq \mathbf{nf} = \mathbf{n} - \mathbf{f}$ replicas.

- $\mathbf{f}$-out-of-$\mathbf{m}$ replicas can be Byzantine and send out *several* Prepare messages.
- $\mathbf{n} - \mathbf{m}$ replicas could have received *different* proposals from the primary.

We must have: $\mathbf{n} - \mathbf{m} < \mathbf{m} - \mathbf{f}$

Take maximum value for $\mathbf{m}$: $\mathbf{m} = \mathbf{nf} = \mathbf{n} - \mathbf{f}$. We must have:

$$\mathbf{n} - (\mathbf{n} - \mathbf{f}) < (\mathbf{n} - \mathbf{f}) - \mathbf{f}$$

Primary-Backup Replication: Dealing with Byzantine Failures

Enforce "If good replicas *pledge*, then they all should do so for the same transaction."

What is a *sufficient* amount of **matching** Prepare-messages

Assume we received matching Prepare messages from $\mathbf{m} \leq \mathbf{nf} = \mathbf{n} - \mathbf{f}$ replicas.

- $\mathbf{f}$-out-of-$\mathbf{m}$ replicas can be Byzantine and send out *several* Prepare messages.
- $\mathbf{n} - \mathbf{m}$ replicas could have received *different* proposals from the primary.

We must have: $\mathbf{n} - \mathbf{m} < \mathbf{m} - \mathbf{f}$

Take maximum value for $\mathbf{m}$: $\mathbf{m} = \mathbf{nf} = \mathbf{n} - \mathbf{f}$. We must have:

$$\begin{aligned} \mathbf{n} - (\mathbf{n} - \mathbf{f}) &< (\mathbf{n} - \mathbf{f}) - \mathbf{f} && \text{(simplify terms)} \\ \mathbf{f} &< \mathbf{n} - 2\mathbf{f} \end{aligned}$$

Primary-Backup Replication: Dealing with Byzantine Failures

Enforce "If good replicas *pledge*, then they all should do so for the same transaction."

What is a *sufficient* amount of **matching** Prepare-messages

Assume we received matching Prepare messages from $\mathbf{m} \leq \mathbf{nf} = \mathbf{n} - \mathbf{f}$ replicas.

- $\mathbf{f}$-out-of-$\mathbf{m}$ replicas can be Byzantine and send out *several* Prepare messages.
- $\mathbf{n} - \mathbf{m}$ replicas could have received *different* proposals from the primary.

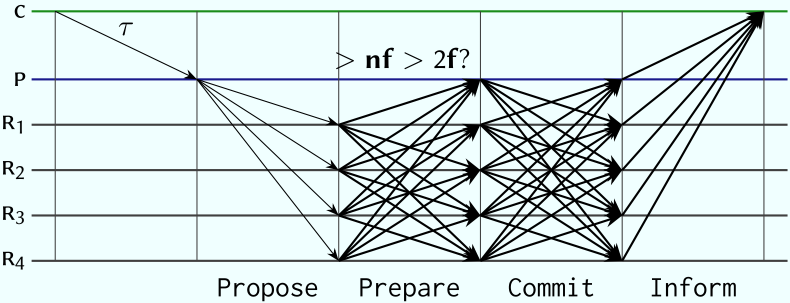
We must have: $\mathbf{n} - \mathbf{m} < \mathbf{m} - \mathbf{f}$

Take maximum value for $\mathbf{m}$: $\mathbf{m} = \mathbf{nf} = \mathbf{n} - \mathbf{f}$. We must have:

$$\begin{aligned} \mathbf{n} - (\mathbf{n} - \mathbf{f}) &< (\mathbf{n} - \mathbf{f}) - \mathbf{f} && \text{(simplify terms)} \\ \mathbf{f} &< \mathbf{n} - 2\mathbf{f} && \text{(rearrange terms)} \\ 3\mathbf{f} &< \mathbf{n} \end{aligned}$$

Primary-Backup Replication: Dealing with Byzantine Failures

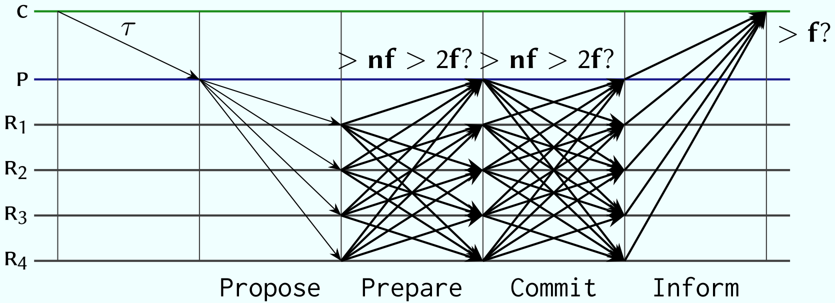
Enforce “If good replicas *pledge*, then they all should do so for the same transaction.”



- Replicas *pledge* only if they receive *sufficient* **matching** `Prepare`-messages.
- Replicas *commit* only if they receive *sufficient* **matching** `Commit`-messages.
- Client *observes* outcome only if they receive *sufficient* **matching** `Inform` messages.

Primary-Backup Replication: Dealing with Byzantine Failures

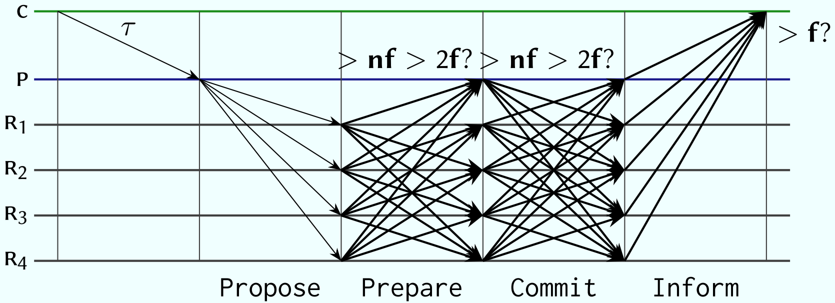
Enforce "If good replicas *pledge*, then they all should do so for the same transaction."



- Replicas *pledge* only if they receive *sufficient* **matching** `Prepare`-messages.
- Replicas *commit* only if they receive *sufficient* **matching** `Commit`-messages.
- Client *observes* outcome only if they receive *sufficient* **matching** `Inform` messages.

Primary-Backup Replication: Dealing with Byzantine Failures

Enforce "If good replicas *pledge*, then they all should do so for the same transaction."

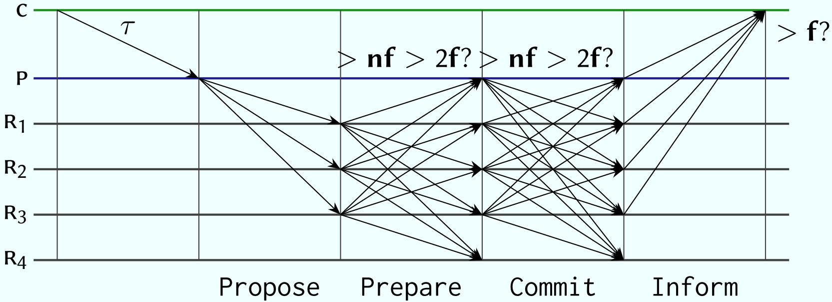


Theorem

If the primary is good and the network is reliable,
then all good replicas will commit and the client will observe outcome.

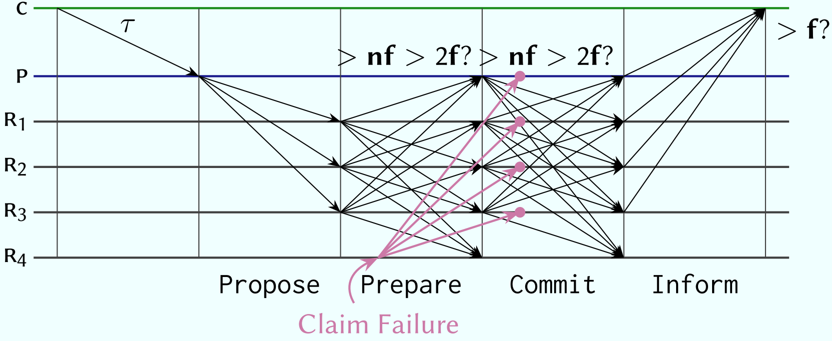
Recovering from Failure: Detecting Failures

Case 1: Primary failure, ignores replica R_4



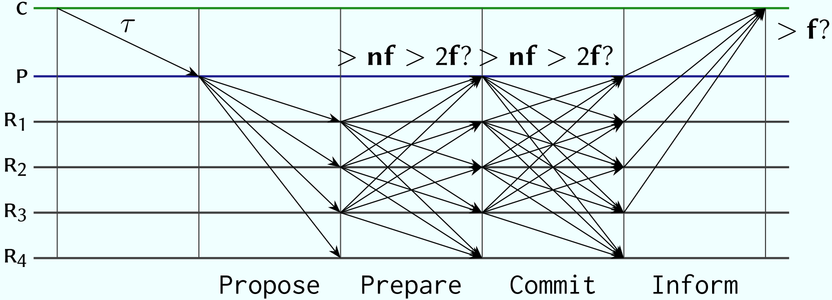
Recovering from Failure: Detecting Failures

Case 1: Primary failure, ignores replica R_4



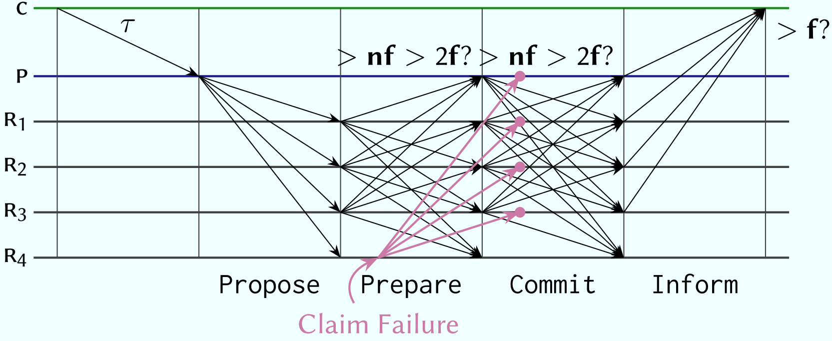
Recovering from Failure: Detecting Failures

Case 2: Replica failure at R_4, pretends primary failed



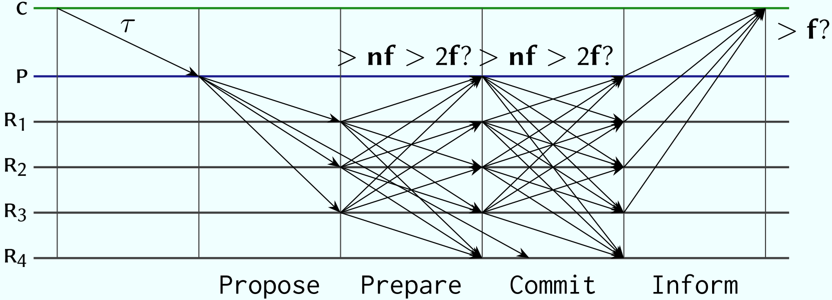
Recovering from Failure: Detecting Failures

Case 2: Replica failure at R_4, pretends primary failed



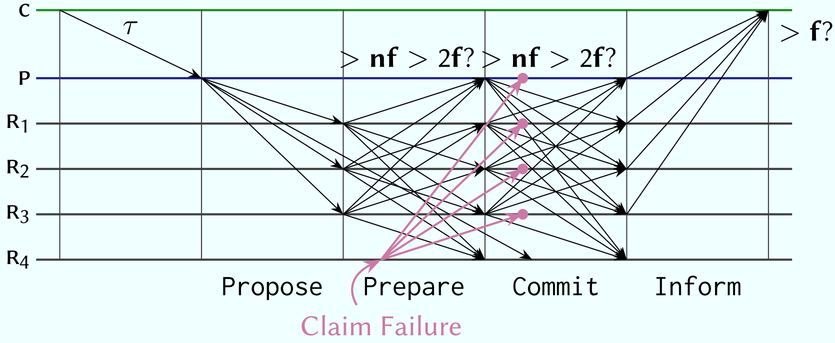
Recovering from Failure: Detecting Failures

Case 3: Message delays



Recovering from Failure: Detecting Failures

Case 3: Message delays



Recovering from Failure: Detecting Failures

What do replicas R_1, R_2, and R_3 see?

- They got `Proposal` and `Commit` messages from the primary.
- They got `Prepare` and `Commit` messages from each other.
- They got a *failure claim* from R_4.

Recovering from Failure: Detecting Failures

What do replicas R_1, R_2, and R_3 see?

- They got `Proposal` and `Commit` messages from the primary.
- They got `Prepare` and `Commit` messages from each other.
- They got a *failure claim* from R_4.

For replicas R_1, R_2, and R_3: the three case are *indistinguishable*.

Recovering from Failure: Detecting Failures

What do replicas R_1, R_2, and R_3 see?

- They got `Proposal` and `Commit` messages from the primary.
- They got `Prepare` and `Commit` messages from each other.
- They got a *failure claim* from R_4.

For replicas R_1, R_2, and R_3: the three case are *indistinguishable*.

Implications

- We cannot detect all failures.
- Byzantine replicas can lie about primary failure.
- Network failure can look like primary failure.

Recovery from Failure: Two Cases

We cannot detect all failures.

Recovery from Failure: Two Cases

We cannot detect all failures.

Assume (for now): No network failures

Upon a failure claim, we can distinguish two cases:

We cannot pinpoint a failure

"A few failure claims (at-most-**f**)"

We can pinpoint a failure

"A lot of failure claims (at-least-**f**)"

Recovery from Failure: Two Cases

We cannot detect all failures.

Assume (for now): No network failures

Upon a failure claim, we can distinguish two cases:

We cannot pinpoint a failure

"A few failure claims (at-most-**f**)"

- *Sufficient* replicas can commit.

We can pinpoint a failure

"A lot of failure claims (at-least-**f**)"

- *Sufficient* replicas fail to commit.

Recovery from Failure: Two Cases

We cannot detect all failures.

Assume (for now): No network failures

Upon a failure claim, we can distinguish two cases:

We cannot pinpoint a failure

"A few failure claims (at-most-**f**)"

- *Sufficient* replicas can commit.
- Primary or backup failure.

We can pinpoint a failure

"A lot of failure claims (at-least-**f**)"

- *Sufficient* replicas fail to commit.
- The primary failed.

Recovery from Failure: Two Cases

We cannot detect all failures.

Assume (for now): No network failures

Upon a failure claim, we can distinguish two cases:

We cannot pinpoint a failure

"A few failure claims (at-most-**f**)"

- *Sufficient* replicas can commit.
- Primary or backup failure.
- Keep the primary in charge.

We can pinpoint a failure

"A lot of failure claims (at-least-**f**)"

- *Sufficient* replicas fail to commit.
- The primary failed.
- Elect a new primary.

Recovery from Failure: Two Cases

We cannot detect all failures.

Assume (for now): No network failures

Upon a failure claim, we can distinguish two cases:

We cannot pinpoint a failure

"A few failure claims (at-most-**f**)"

- *Sufficient* replicas can commit.
- Primary or backup failure.
- Keep the primary in charge.
- Use *checkpoints* to recover any backups.

We can pinpoint a failure

"A lot of failure claims (at-least-**f**)"

- *Sufficient* replicas fail to commit.
- The primary failed.
- Elect a new primary.
- Use *view-change* to recover failed state.

PBFT Operates in Views

In view v, the replica $\textsc{p}$ with $\mathrm{id}(\textsc{p}) = v \bmod \mathbf{n}$ is the primary.

- View v will perform consensus rounds until failure.
- If view v fails to perform rounds: we assume failure of $\textsc{p}$.
- Upon failure of $\textsc{p}$, we move to view $v + 1$.
- View $v + 1$ must recover *all* requests with possibly-observed outcomes.

PBFT Operates in Views

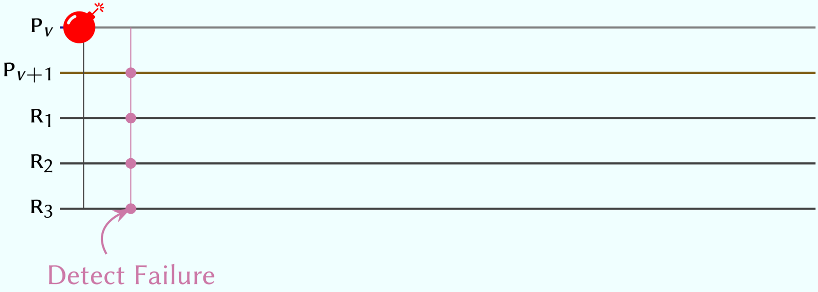
In view v, the replica $\textsc{p}$ with $\text{id}(\textsc{p}) = v \bmod \mathbf{n}$ is the primary.

- View v will perform consensus rounds until failure.
- If view v fails to perform rounds: we assume failure of $\textsc{p}$.
- Upon failure of $\textsc{p}$, we move to view $v + 1$.
- View $v + 1$ must recover *all* requests with possibly-observed outcomes.

The two phases of a view-change

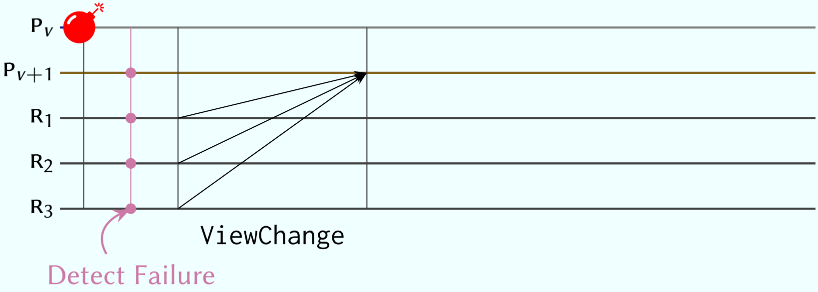
- Phase 1: *Synchronize* failure detection.
- Phase 2: *New-View* proposal.

Recovery from Failure: *New-View* Proposal



New primary P_{v+1} needs to recover requests

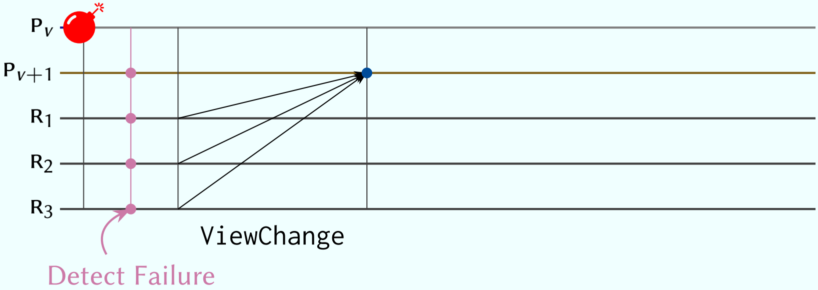
Recovery from Failure: *New-View* Proposal



New primary P_{v+1} needs to recover requests

- Each replica R sends to P_{v+1} a **signed** `ViewChange` message m_R.
  (m_R summarizes *all proposals*, *all pledges*, and *all commits* by R.)

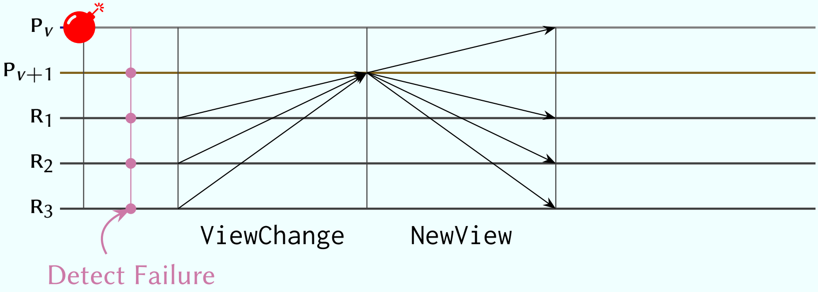
Recovery from Failure: *New-View* Proposal



New primary P_{v+1} needs to recover requests

- Each replica R sends to P_{v+1} a **signed** `ViewChange` message m_R.
  (m_R summarizes *all proposals*, *all pledges*, and *all commits* by R.)
- P_{v+1} selects a set N of $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ *well-formed* `ViewChange` messages.

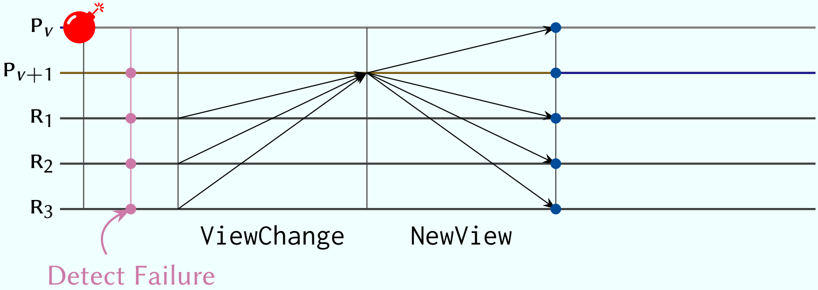
Recovery from Failure: *New-View* Proposal



New primary P_{v+1} needs to recover requests

- Each replica R sends to P_{v+1} a **signed** `ViewChange` message m_R.
  (m_R summarizes *all proposals*, *all pledges*, and *all commits* by R.)
- P_{v+1} selects a set N of $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ *well-formed* `ViewChange` messages.
- P_{v+1} proposes a `NewView` message with content N as the basis for recovery.

Recovery from Failure: *New-View* Proposal



New primary P_{v+1} needs to recover requests

- Each replica R sends to P_{v+1} a **signed** ViewChange message m_R.
  (m_R summarizes *all proposals*, *all pledges*, and *all commits* by R.)
- P_{v+1} selects a set N of $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ *well-formed* ViewChange messages.
- P_{v+1} proposes a NewView message with content N as the basis for recovery.
- Each replica updates their internal state in accordance with N.

Interpretation of a `NewView` Message

Consider any set N of $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ *well-formed* `ViewChange` messages for view $v + 1$.

Informal goal: "View $v + 1$ must recover *all* requests with possibly-observed outcomes".

Interpretation of a `NewView` Message

Consider any set N of $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ *well-formed* `ViewChange` messages for view $v + 1$.

Informal goal: "View $v + 1$ must recover *all* requests with possibly-observed outcomes".

- Possibly-observed outcome for τ: only if *one* good replica *committed* τ.

Interpretation of a NewView Message

Consider any set N of $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ *well-formed* `ViewChange` messages for view $v + 1$.

Informal goal: "View $v + 1$ must recover *all* requests with possibly-observed outcomes".

- Possibly-observed outcome for τ: only if *one* good replica *committed* τ.
- Possibly-committed τ: only if $\mathbf{nf} - \mathbf{f}$ good replicas *pledged* τ.

Interpretation of a `NewView` Message

Consider any set N of $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ *well-formed* `ViewChange` messages for view $v + 1$.

Informal goal: "View $v + 1$ must recover *all* requests with possibly-observed outcomes".

- Possibly-observed outcome for τ: only if *one* good replica *committed* τ.
- Possibly-committed τ: only if $\mathbf{nf} - \mathbf{f}$ good replicas *pledged* τ.

A *minimal* view-change guarantee

A view-change to view $v + 1$ can only succeed if the change recover *all* requests to which at-least $\mathbf{nf} - \mathbf{f}$ good replicas *pledged* in a round ρ of a preceding view $w \leq v$.

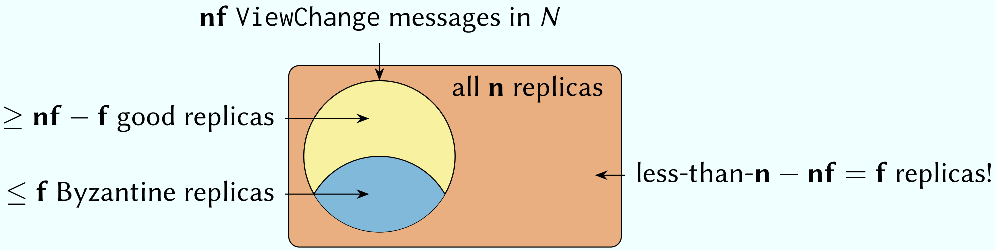
Interpretation of a NewView Message

Consider any set N of $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ *well-formed* `ViewChange` messages for view $v + 1$.

A *minimal* view-change guarantee

A view-change to view $v + 1$ can only succeed if the change recover *all* requests to which at-least $\mathbf{nf} - \mathbf{f}$ good replicas *pledged* in a round ρ of a preceding view $w \leq v$.

Base case: $w = v$, $\mathbf{n} > 3\mathbf{f}$



Interpretation of a NewView Message

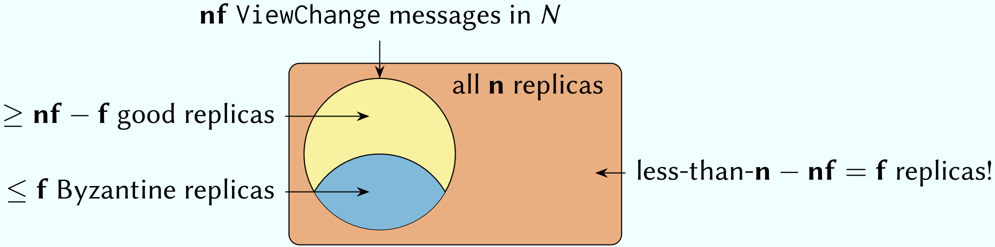
Consider any set N of $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ *well-formed* `ViewChange` messages for view $v + 1$.

A *minimal* view-change guarantee

A view-change to view $v + 1$ can only succeed if the change recover *all* requests to which at-least $\mathbf{nf} - \mathbf{f}$ good replicas *pledged* in a round ρ of a preceding view $w \leq v$.

Base case: $w = v$, $\mathbf{n} > 3\mathbf{f}$

Consider any set M of $\mathbf{nf} - \mathbf{f}$ good replicas that pledged τ in round ρ of view v.



Interpretation of a NewView Message

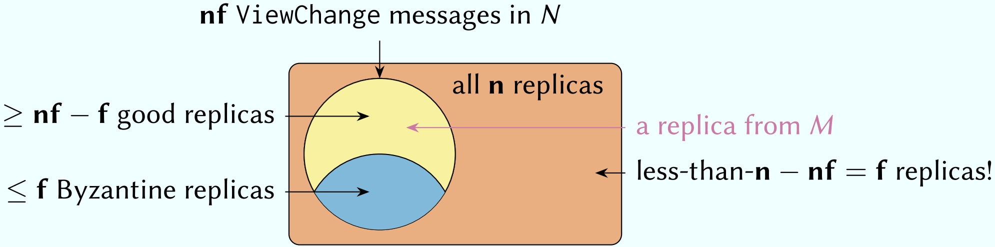
Consider any set N of $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ *well-formed* `ViewChange` messages for view $v + 1$.

A *minimal* view-change guarantee

A view-change to view $v + 1$ can only succeed if the change recover *all* requests to which at-least $\mathbf{nf} - \mathbf{f}$ good replicas *pledged* in a round ρ of a preceding view $w \leq v$.

Base case: $w = v$, $\mathbf{n} > 3\mathbf{f}$

Consider any set M of $\mathbf{nf} - \mathbf{f}$ good replicas that pledged τ in round ρ of view v.



Interpretation of a NewView Message

Consider any set N of $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ *well-formed* `ViewChange` messages for view $v + 1$.

A *minimal* view-change guarantee

A view-change to view $v + 1$ can only succeed if the change recover *all* requests to which at-least $\mathbf{nf} - \mathbf{f}$ good replicas *pledged* in a round ρ of a preceding view $w \leq v$.

Base case: $w = v$, $\mathbf{n} > 3\mathbf{f}$

Let $\textsc{q}$ be a replica in M whose `ViewChange` message is in N.

Interpretation of a `NewView` Message

Consider any set N of $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ *well-formed* `ViewChange` messages for view $v + 1$.

A *minimal* view-change guarantee

A view-change to view $v + 1$ can only succeed if the change recover *all* requests to which at-least $\mathbf{nf} - \mathbf{f}$ good replicas *pledged* in a round ρ of a preceding view $w \leq v$.

Base case: $w = v$, $\mathbf{n} > 3\mathbf{f}$

Let Q be a replica in M whose `ViewChange` message is in N.

- Q pledged τ in round ρ of view v.
- Q did not pledge in round ρ of views $u > v$ (v is latest view).
- Q has $\mathbf{nf}$ `Prepare` messages to prove validity of the *pledge*: a *pledge proof*.

Interpretation of a NewView Message

Consider any set N of $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ *well-formed* `ViewChange` messages for view $v + 1$.

A *minimal* view-change guarantee

A view-change to view $v + 1$ can only succeed if the change recover *all* requests to which at-least $\mathbf{nf} - \mathbf{f}$ good replicas *pledged* in a round ρ of a preceding view $w \leq v$.

Base case: $w = v$, $\mathbf{n} > 3\mathbf{f}$

Let Q be a replica in M whose `ViewChange` message is in N.

- Q pledged τ in round ρ of view v.
- Q did not pledge in round ρ of views $u > v$ (v is latest view).
- Q has $\mathbf{nf}$ `Prepare` messages to prove validity of the *pledge*: a *pledge proof*.

For simplicity: We include *prepare certificates* (pledge proofs) in `ViewChange` messages.

Interpretation of a `NewView` Message

Consider any set N of $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ *well-formed* `ViewChange` messages for view $v + 1$.

A *minimal* view-change guarantee

A view-change to view $v + 1$ can only succeed if the change recover *all* requests
to which at-least $\mathbf{nf} - \mathbf{f}$ good replicas *pledged* in a round ρ of a preceding view $w \leq v$.

Base case: $w = v$, $\mathbf{n} > 3\mathbf{f}$

For simplicity: We include *prepare certificates* (pledge proofs) in `ViewChange` messages.

N holds a prepare certificate for τ if $\mathbf{nf} - \mathbf{f}$ good replicas pledged τ in round ρ of view v.

Interpretation of a `NewView` Message

Consider any set N of $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ *well-formed* `ViewChange` messages for view $v + 1$.

A *minimal* view-change guarantee

A view-change to view $v + 1$ can only succeed if the change recover *all* requests
to which at-least $\mathbf{nf} - \mathbf{f}$ good replicas *pledged* in a round ρ of a preceding view $w \leq v$.

Base case: $w = v$, $\mathbf{n} > 3\mathbf{f}$

For simplicity: We include *prepare certificates* (pledge proofs) in `ViewChange` messages.

N holds a prepare certificate for τ if $\mathbf{nf} - \mathbf{f}$ good replicas pledged τ in round ρ of view v.
Likewise: N holds a *commit certificate* for τ if $\mathbf{nf} - \mathbf{f}$ good replicas committed τ.

Interpretation of a NewView Message

Consider any set N of $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ *well-formed* `ViewChange` messages for view $v + 1$.

A *minimal* view-change guarantee

A view-change to view $v + 1$ can only succeed if the change recover *all* requests to which at-least $\mathbf{nf} - \mathbf{f}$ good replicas *pledged* in a round ρ of a preceding view $w \leq v$.

Base case: $w = v$, $\mathbf{n} > 3\mathbf{f}$

For simplicity: We include *prepare certificates* (pledge proofs) in `ViewChange` messages.

N holds a prepare certificate for τ if $\mathbf{nf} - \mathbf{f}$ good replicas pledged τ in round ρ of view v.
Likewise: N holds a *commit certificate* for τ if $\mathbf{nf} - \mathbf{f}$ good replicas committed τ.

Recovery Rule

Recover transactions τ for round ρ for which a prepare certificates was included in N for a view $w \leq v$ such that no *more recent* certificates for round ρ exists.

Interpretation of a NewView Message

Consider any set N of $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ *well-formed* `ViewChange` messages for view $v + 1$.

A *minimal* view-change guarantee

A view-change to view $v + 1$ can only succeed if the change recover *all* requests to which at-least $\mathbf{nf} - \mathbf{f}$ good replicas *pledged* in a round ρ of a preceding view $w \leq v$.

Recovery Rule

Recover transactions τ for round ρ for which a prepare certificates was included in N for a view $w \leq v$ such that no *more recent* certificates for round ρ exists.

Inductive case: $w < v$, $\mathbf{n} > 3\mathbf{f}$

Consider a view-change to view w', $w < w' < v$:

- View-change *fails*—View w' will not make new prepare certificates for any rounds.
- View-change *succeeds*—View w' can make new prepare certificates for any round ρ', but *only* if no transactions where recovered for round ρ'.

Interpretation of a NewView Message

Consider any set N of $\mathbf{nf} = \mathbf{n} - \mathbf{f}$ *well-formed* `ViewChange` messages for view $v + 1$.

A *minimal* view-change guarantee

A view-change to view $v + 1$ can only succeed if the change recover *all* requests to which at-least $\mathbf{nf} - \mathbf{f}$ good replicas *pledged* in a round ρ of a preceding view $w \leq v$.

Recovery Rule

Recover transactions τ for round ρ for which a prepare certificates was included in N for a view $w \leq v$ such that no *more recent* certificates for round ρ exists.

Start of a new view

Consider a round ρ. If N contains

- no prepare certificates for ρ, then consider nothing proposed yet;
- a commit certificate for ρ, then consider round ρ committed;
- a prepare certificate for ρ, then repropose the certified transaction.

View-Changes and Authenticated Communication

We described a view-change protocol with *message forwarding*: digital signatures.

View-changes with authenticated communication only is possible, but more complex.

Recovery from Failure: Starting a View-Change

Consider a replica R

- When does R start participating in a view-change?

Recovery from Failure: Starting a View-Change

Consider a replica R

- When does R start participating in a view-change?
  After it detects a failure.

Recovery from Failure: Starting a View-Change

Consider a replica R

- When does R start participating in a view-change?
  After it detects a failure.

- How does R detect failure?

Recovery from Failure: Starting a View-Change

Consider a replica R

- When does R start participating in a view-change?
  After it detects a failure.

- How does R detect failure?
  - When it expects to commit a proposal, but fails to do so *on time*?

Recovery from Failure: Starting a View-Change

Consider a replica R

- When does R start participating in a view-change?
  After it detects a failure.

- How does R detect failure?
  - When it expects to commit a proposal, but fails to do so *on time*?
  - If a good replica *claims failure*: At-least $\mathbf{f} + 1$ failure claims.

Recovery from Failure: Starting a View-Change

Consider a replica R

- When does R start participating in a view-change?
  After it detects a failure.
- How does R detect failure?
  - When it expects to commit a proposal, but fails to do so *on time*?
  - If a good replica *claims failure*: At-least $\mathbf{f} + 1$ failure claims.
- When is a commit not *on time* for R?

Recovery from Failure: Starting a View-Change

Consider a replica R

- When does R start participating in a view-change?
  After it detects a failure.

- How does R detect failure?
  - When it expects to commit a proposal, but fails to do so *on time*?
  - If a good replica *claims failure*: At-least $\mathbf{f} + 1$ failure claims.

- When is a commit not *on time* for R?
  R uses an internal *network delay estimate* (remember: asynchronous communication).

Recovery from Failure: Starting a View-Change

Consider a replica R

- When does R start participating in a view-change?
  After it detects a failure.
- How does R detect failure?
  - When it expects to commit a proposal, but fails to do so *on time*?
  - If a good replica *claims failure*: At-least $\mathbf{f}+1$ failure claims.
- When is a commit not *on time* for R?
  R uses an internal *network delay estimate* (remember: asynchronous communication).
- When can R expect any commit?

Recovery from Failure: Starting a View-Change

Consider a replica R

- When does R start participating in a view-change?
  After it detects a failure.

- How does R detect failure?
  - When it expects to commit a proposal, but fails to do so *on time*?
  - If a good replica *claims failure*: At-least $\mathbf{f} + 1$ failure claims.

- When is a commit not *on time* for R?
  R uses an internal *network delay estimate* (remember: asynchronous communication).

- When can R expect any commit?
  - If R forwarded a client request to the current primary.

Recovery from Failure: Starting a View-Change

Consider a replica R

- When does R start participating in a view-change?
  After it detects a failure.

- How does R detect failure?
  - When it expects to commit a proposal, but fails to do so *on time*?
  - If a good replica *claims failure*: At-least $\mathbf{f} + 1$ failure claims.

- When is a commit not *on time* for R?
  R uses an internal *network delay estimate* (remember: asynchronous communication).

- When can R expect any commit?
  - If R forwarded a client request to the current primary.
  - If R received a `Proposal` message or received $\mathbf{f} + 1$ `Prepare` or `Commit` messages.

Recovery from Failure: Out-of-Sync

What if …

- R_1 starts the view-change at $t_1 = 15$, with an expected duration of 4.
- R_2 starts the view-change at $t_2 = 20$, with an expected duration of 2.
- R_3 starts the view-change at $t_3 = 12$, with an expected duration of 1.
- R_4 does not start the view-change (the current and Byzantine primary).

Recovery from Failure: Out-of-Sync

What if …

- R_1 starts the view-change at $t_1 = 15$, with an expected duration of 4.
- R_2 starts the view-change at $t_2 = 20$, with an expected duration of 2.
- R_3 starts the view-change at $t_3 = 12$, with an expected duration of 1.
- R_4 does not start the view-change (the current and Byzantine primary).

View-change itself will fail!

Recovery from Failure: Out-of-Sync

What if …

- R_1 starts the view-change at $t_1 = 15$, with an expected duration of 4.
- R_2 starts the view-change at $t_2 = 20$, with an expected duration of 2.
- R_3 starts the view-change at $t_3 = 12$, with an expected duration of 1.
- R_4 does not start the view-change (the current and Byzantine primary).

View-change itself will fail!

Replicas need to start view-change roughly at the same time.
Replicas must wait long enough for the new primary to be able to finish.

Recovery from Failure: *Synchronize* Failure Detection

Assume: Replica $\textsc{r}$ uses network delay $\delta(\textsc{r}, v)$ in view v

Recovery from Failure: *Synchronize* Failure Detection

Assume: Replica $\textsc{r}$ uses network delay $\delta(\textsc{r}, v)$ in view v

- If $\textsc{r}$ detects failure, it starts broadcasting `Failure` messages.

Recovery from Failure: *Synchronize* Failure Detection

Assume: Replica R uses network delay $\delta(\text{R}, v)$ in view v

- If R detects failure, it starts broadcasting `Failure` messages.
- If R *knows* that all good replicas will detect failure: start timer.

Recovery from Failure: *Synchronize* Failure Detection

Assume: Replica R uses network delay $\delta(\text{R}, v)$ in view v

- If R detects failure, it starts broadcasting `Failure` messages.
- If R *knows* that all good replicas will detect failure: start timer.
- When R starts the timer, it sends `ViewChange` to the new primary.

Recovery from Failure: *Synchronize* Failure Detection

Assume: Replica R uses network delay $\delta(\text{R}, v)$ in view v

- If R detects failure, it starts broadcasting `Failure` messages.
- If R *knows* that all good replicas will detect failure: start timer.
- When R starts the timer, it sends `ViewChange` to the new primary.
- If a valid `NewView` message arrives on time: accept it.

Recovery from Failure: *Synchronize* Failure Detection

Assume: Replica R uses network delay $\delta(\text{R}, v)$ in view v

- If R detects failure, it starts broadcasting `Failure` messages.
- If R *knows* that all good replicas will detect failure: start timer.
- When R starts the timer, it sends `ViewChange` to the new primary.
- If a valid `NewView` message arrives on time: accept it.
- If no valid `NewView` message arrives: detect failure of view $v + 1$.
  (Use exponential backoff on the network delay: $\delta(\text{R}, v + 1) = 2\delta(\text{R}, v)$.)

Recovery from Failure: *Synchronize* Failure Detection

Assume: Replica R uses network delay $\delta(\text{R}, v)$ in view v

- If R detects failure, it starts broadcasting `Failure` messages.
- If R *knows* that all good replicas will detect failure: start timer.
- When R starts the timer, it sends `ViewChange` to the new primary.
- If a valid `NewView` message arrives on time: accept it.
- If no valid `NewView` message arrives: detect failure of view $v + 1$.
  (Use exponential backoff on the network delay: $\delta(\text{R}, v + 1) = 2\delta(\text{R}, v)$.)

When does R *know* that all good replicas will detect failure?

- If $\mathbf{f} + 1$ good replicas detect failure, then everyone will receive $\mathbf{f} + 1$ `Failure` messages.
- If a replica receives $\mathbf{f} + 1$ `Failure` messages, it will also claim failure.
- Receiving $2\mathbf{f} + 1$ `Failure` messages implies $\mathbf{f} + 1$ came from good replicas.

Recovery from Failure: *Synchronize* Failure Detection

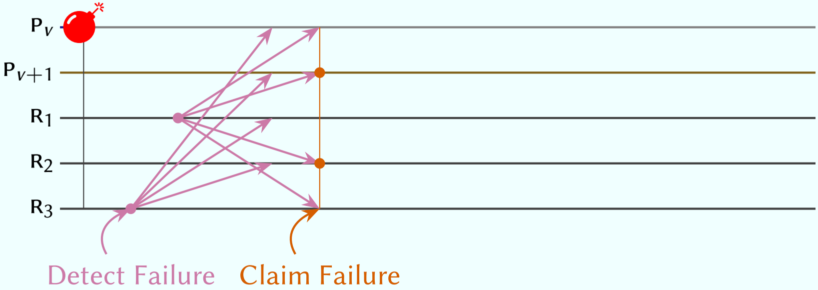
When does R *know* that all good replicas will detect failure?

- If $\mathbf{f}+1$ good replicas detect failure, then everyone will receive $\mathbf{f}+1$ `Failure` messages.
- If a replica receives $\mathbf{f}+1$ `Failure` messages, it will also claim failure.
- Receiving $2\mathbf{f}+1$ `Failure` messages implies $\mathbf{f}+1$ came from good replicas.

Recovery from Failure: *Synchronize* Failure Detection

When does R *know* that all good replicas will detect failure?

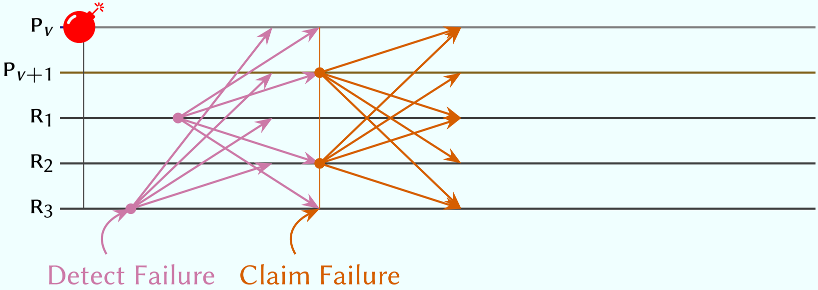
- If $\mathbf{f}+1$ good replicas detect failure, then everyone will receive $\mathbf{f}+1$ `Failure` messages.
- If a replica receives $\mathbf{f}+1$ `Failure` messages, it will also claim failure.
- Receiving $2\mathbf{f}+1$ `Failure` messages implies $\mathbf{f}+1$ came from good replicas.



Recovery from Failure: *Synchronize* Failure Detection

When does R *know* that all good replicas will detect failure?

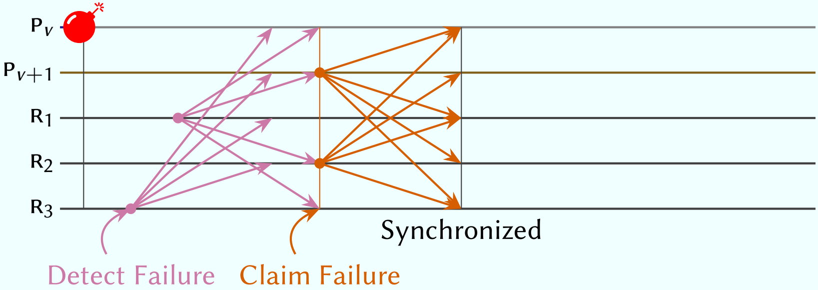
- If $\mathbf{f} + 1$ good replicas detect failure, then everyone will receive $\mathbf{f} + 1$ `Failure` messages.
- If a replica receives $\mathbf{f} + 1$ `Failure` messages, it will also claim failure.
- Receiving $2\mathbf{f} + 1$ `Failure` messages implies $\mathbf{f} + 1$ came from good replicas.



Recovery from Failure: *Synchronize* Failure Detection

When does R *know* that all good replicas will detect failure?

- If $\mathbf{f}+1$ good replicas detect failure, then everyone will receive $\mathbf{f}+1$ `Failure` messages.
- If a replica receives $\mathbf{f}+1$ `Failure` messages, it will also claim failure.
- Receiving $2\mathbf{f}+1$ `Failure` messages implies $\mathbf{f}+1$ came from good replicas.



Recovery from Failure: Remaining Issues

- Dealing with failures when we cannot pinpoint a failure.
  (“A few failure claims (at-most-$\mathbf{f}$)”).
- The unbounded number of rounds considered during view-changes:
  We do not want to have to reconsider the entire ledger during recovery.

Recovery from Failure: Remaining Issues

- Dealing with failures when we cannot pinpoint a failure.
  ("A few failure claims (at-most-$\mathbf{f}$)").
- The unbounded number of rounds considered during view-changes:
  We do not want to have to reconsider the entire ledger during recovery.

Solution: the *checkpoint* protocol

- After committing for all rounds up-to-ρ,
  replicas can broadcast a `Checkpoint` for round ρ.
- After receiving $\mathbf{f} + 1$ **matching** `Checkpoint` messages for round ρ:
  At-least one good replica committed in round $\rho \rightarrow$ Save to copy that commit decision!
- After receiving $\mathbf{n} - \mathbf{f}$ **matching** `Checkpoint` messages for round ρ:
  One can create a *checkpoint certificate*.

Recovery from Failure: Remaining Issues

- Dealing with failures when we cannot pinpoint a failure.
  ("A few failure claims (at-most-$\mathbf{f}$)").
- The unbounded number of rounds considered during view-changes:
  We do not want to have to reconsider the entire ledger during recovery.

Solution: the *checkpoint* protocol

- After committing for all rounds up-to-ρ,
  replicas can broadcast a `Checkpoint` for round ρ.
- After receiving $\mathbf{f} + 1$ **matching** `Checkpoint` messages for round ρ:
  At-least one good replica committed in round $\rho \rightarrow$ Save to copy that commit decision!
- After receiving $\mathbf{n} - \mathbf{f}$ **matching** `Checkpoint` messages for round ρ:
  One can create a *checkpoint certificate*.

Use checkpoint certificates to reduce the size of `ViewChange` messages:
Only include the last checkpoint certificate and details on rounds *after* that checkpoint