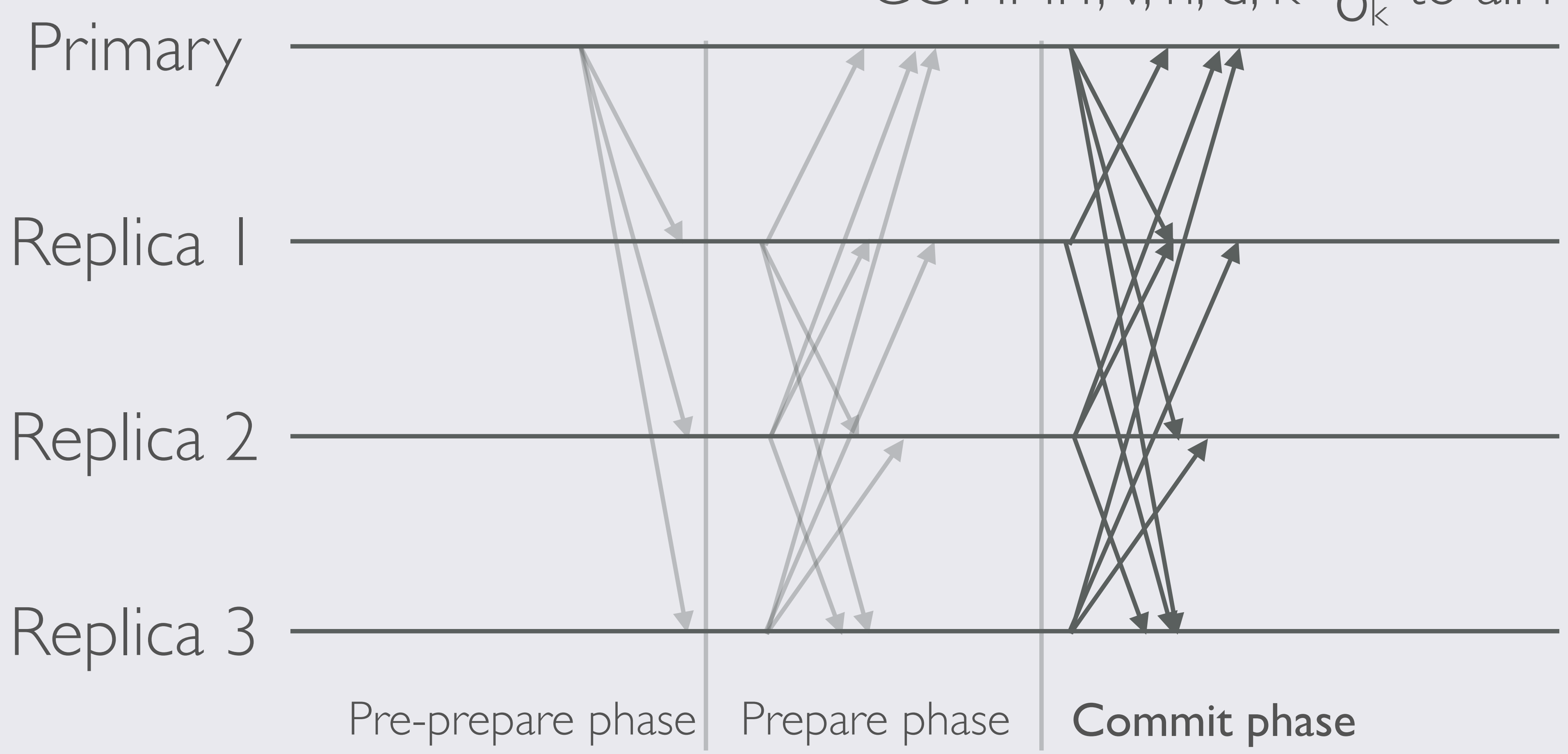
EECS 591
Distributed Systems

Manos Kapritsos
Fall 2021

Commit

After collecting a P-Certificate, replica k sends <COMMIT, v, n, d, k>$_{\sigma_k}$ to all replicas

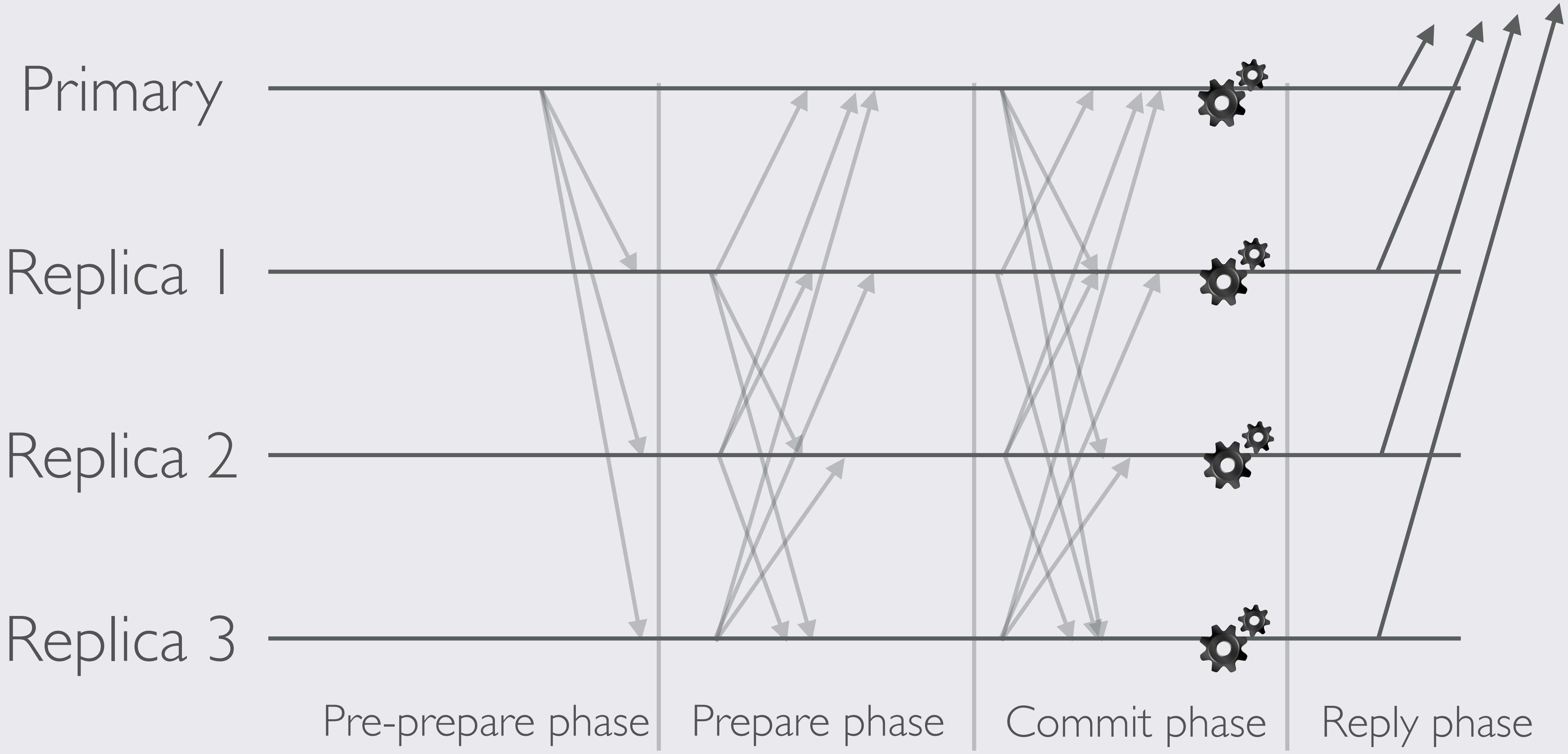


Commit Certificate

- C-Certificates ensure consistent order of requests **across** views
  - **Cannot miss** a P-Certificate during view change
- A replica has a C-Certificate(**m**,**v**,**n**) iff:
  - it had a P-Certificate(**m**,**v**,**n**)
  - its log contains $2f + 1$ matching COMMIT messages from distinct replicas (including itself)
- A replica executes a request when:
  - it gets a C-Certificate for it
  - it has executed all requests with smaller sequence numbers

Reply

After executing a request, replica k replies to the client with $\langle \text{REPLY}, v, t, c, k, r \rangle_{\sigma_k}$



How many matching requests must the client wait for?

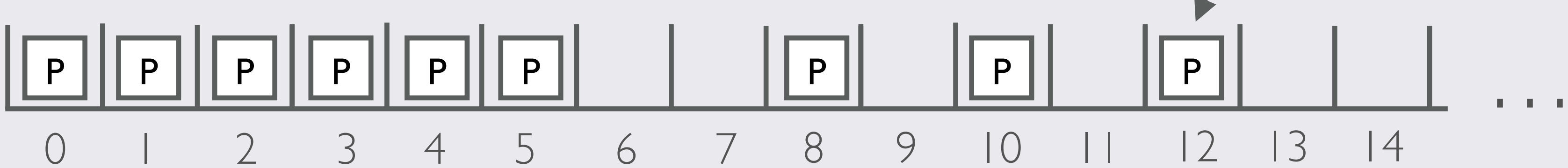
TO ARMS, REPLICAS!!

- A disgruntled replica mutinies:
  - Stops accepting messages (except for VIEW-CHANGE and NEW-VIEW messages)
  - sends <VIEW-CHANGE, **v**+1, $\mathcal{P}$>$_{\sigma_k}$
  - $\mathcal{P}$ contains all P-Certificates known to replica **k**
- A replica joins mutiny after seeing $f + 1$ distinct VIEW-CHANGE messages
- Mutiny succeeds if the 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" **p'** (replica **v**+1 mod **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 P

h=12



  - two sets $\mathcal{O}$ and $\mathcal{N}$:
    - if there is a P-certificate for **n, m** in $\mathcal{V}$, where **n** ≤ **h** add <PRE-PREPARE, **v**+1, **n**, **m**>$_{\sigma_{p'}}$ to $\mathcal{O}$
    - otherwise, if **n** ≤ **h** but there is no P-Certificate add <PRE-PREPARE, **v**+1, **n**, ***null***>$_{\sigma_{p'}}$ to $\mathcal{N}$

- **p'** sends <NEW-VIEW, **v**+1, $\mathcal{V}$, $\mathcal{O}$, $\mathcal{N}$>$_{\sigma_{p'}}$ to all replicas

On to view v+1: the replica

- A replica accepts a NEW-VIEW message for **v+1** if
  - it is signed properly
  - it contains in $\mathcal{V}$ 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 (as did **p'**)
- Sends a PREPARE to all replicas for each message in $\mathcal{O}$
- Adds all PREPARE messages to its log and enters new view

BFT: A PERSPECTIVE

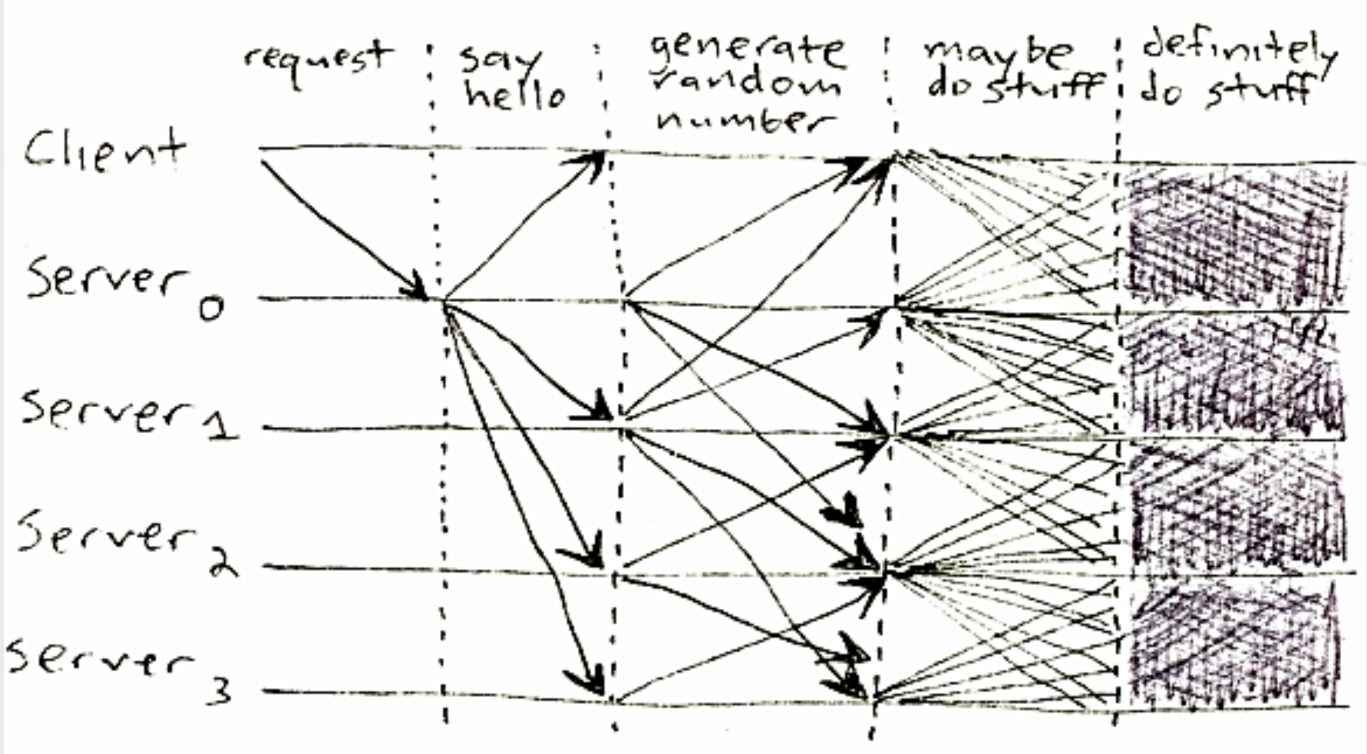


Figure 1: Typical Figure 2 from Byzantine fault paper: Our network protocol

[Mickens 2013]

On the other hand:
Google has used BFT in its datacenters and so do many blockchain approaches

ADMINISTRIVIA

Midterm

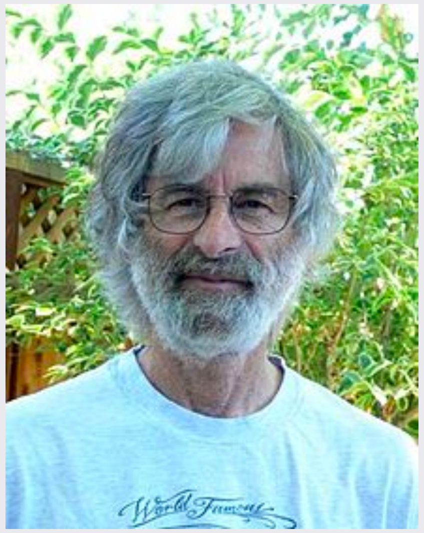
- Next Wednesday 10/27, 12-1:20pm, during class
  - You can use any material listed on the course website

No class on Monday 10/25

- Conflict with SOSP

Research part

- Starts on Monday 11/1
  - You should read both papers and you can review either one



Leslie
Lamport

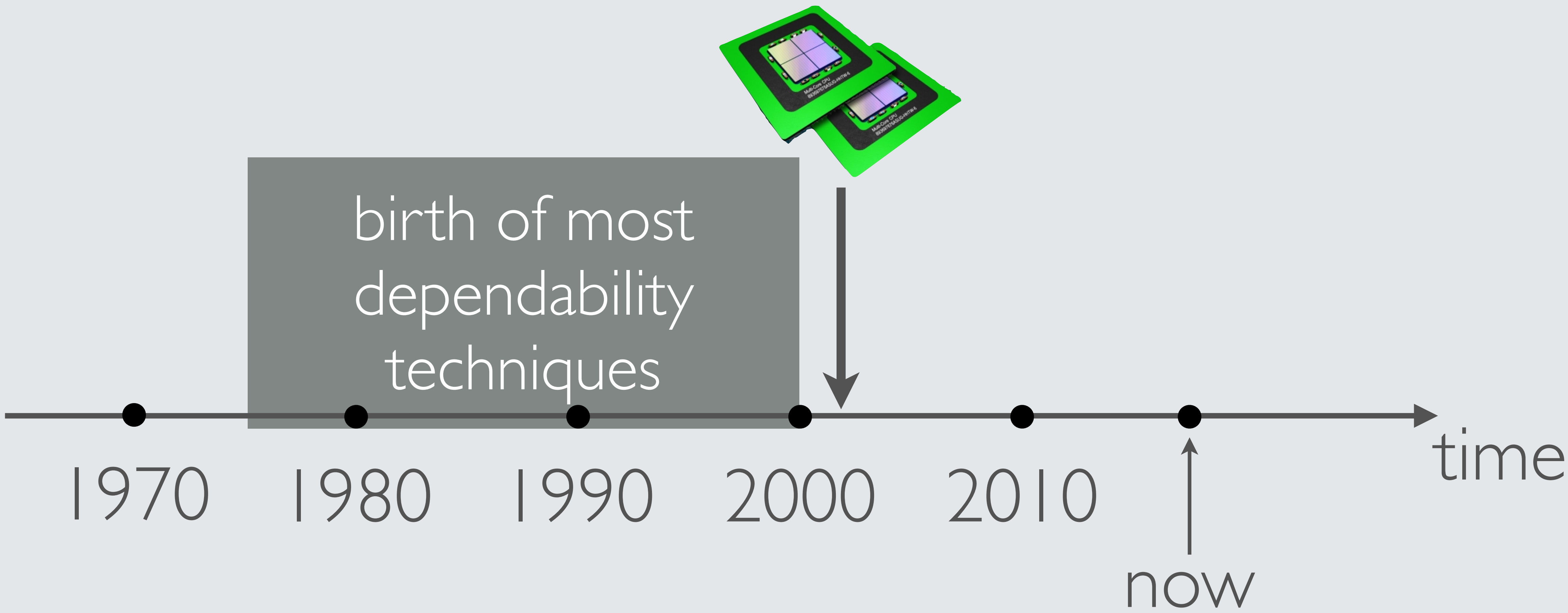
Barbara
Liskov

Lorenzo
Alvisi

EVE: REPLICATING MULTITHREADED SERVERS

Kapritsos, Wang, Quema, Clement, Alvisi, Dahlin

THE ACHILLES' HEEL OF REPLICATION



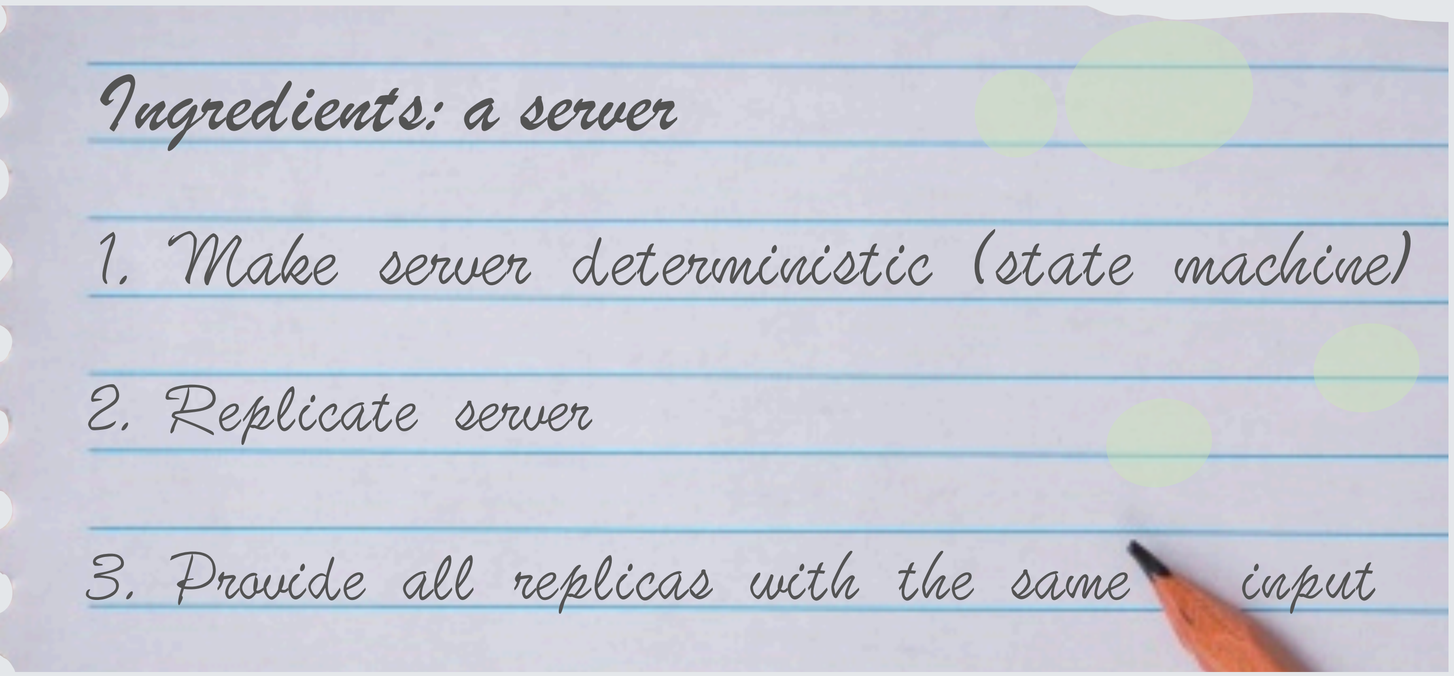
Challenge: scale to **multithreaded execution**

How do we build dependable ~~multithreaded~~ services?

Answer:

State Machine Replication

State Machine Replication

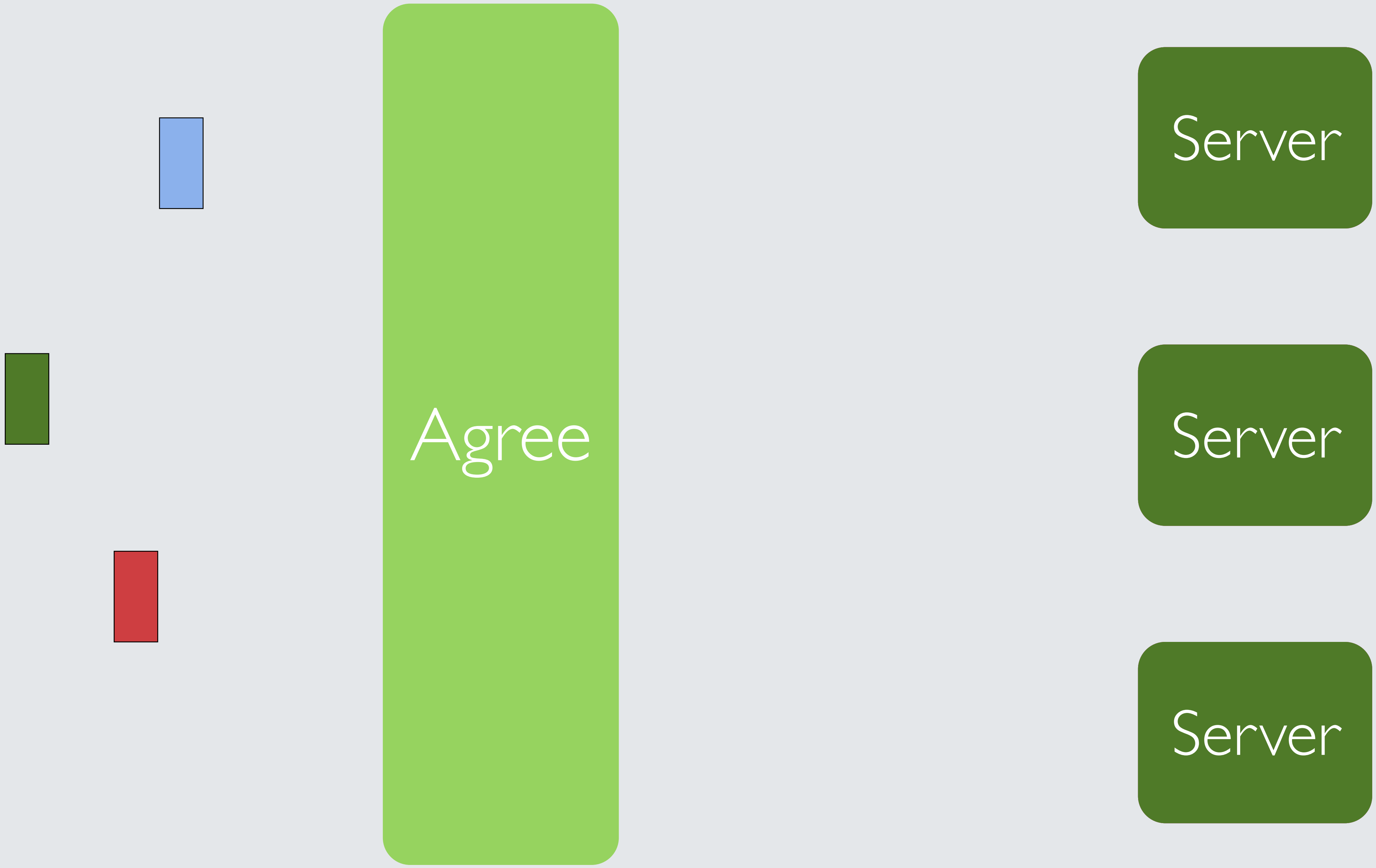


input

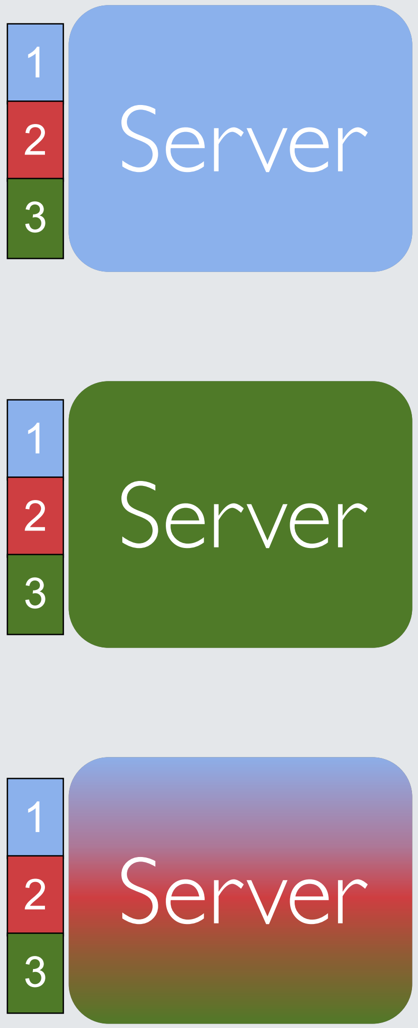
Server

Guarantee: correct replicas will produce the same output

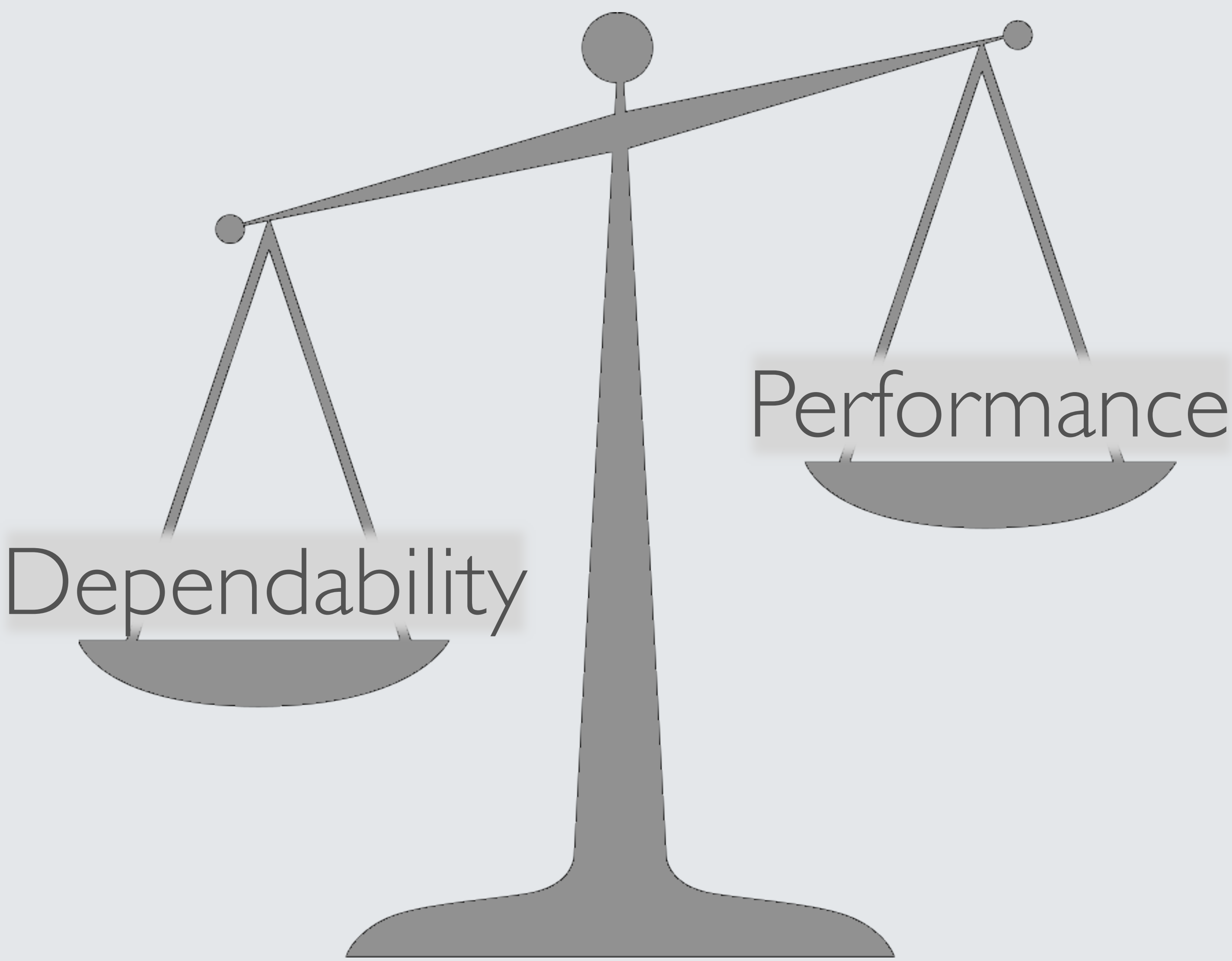
SMR IMPLEMENTATION



How do we build dependable ~~multithreaded~~ services?



How do we build dependable multithreaded services?



Eve (OSDI '12)

Scaling replication to multithreaded execution

SMR requires replica convergence

Agree

Execute

Agree-Execute enforces sequential execution

EXECUTE-VERIFY

Execute

Verify

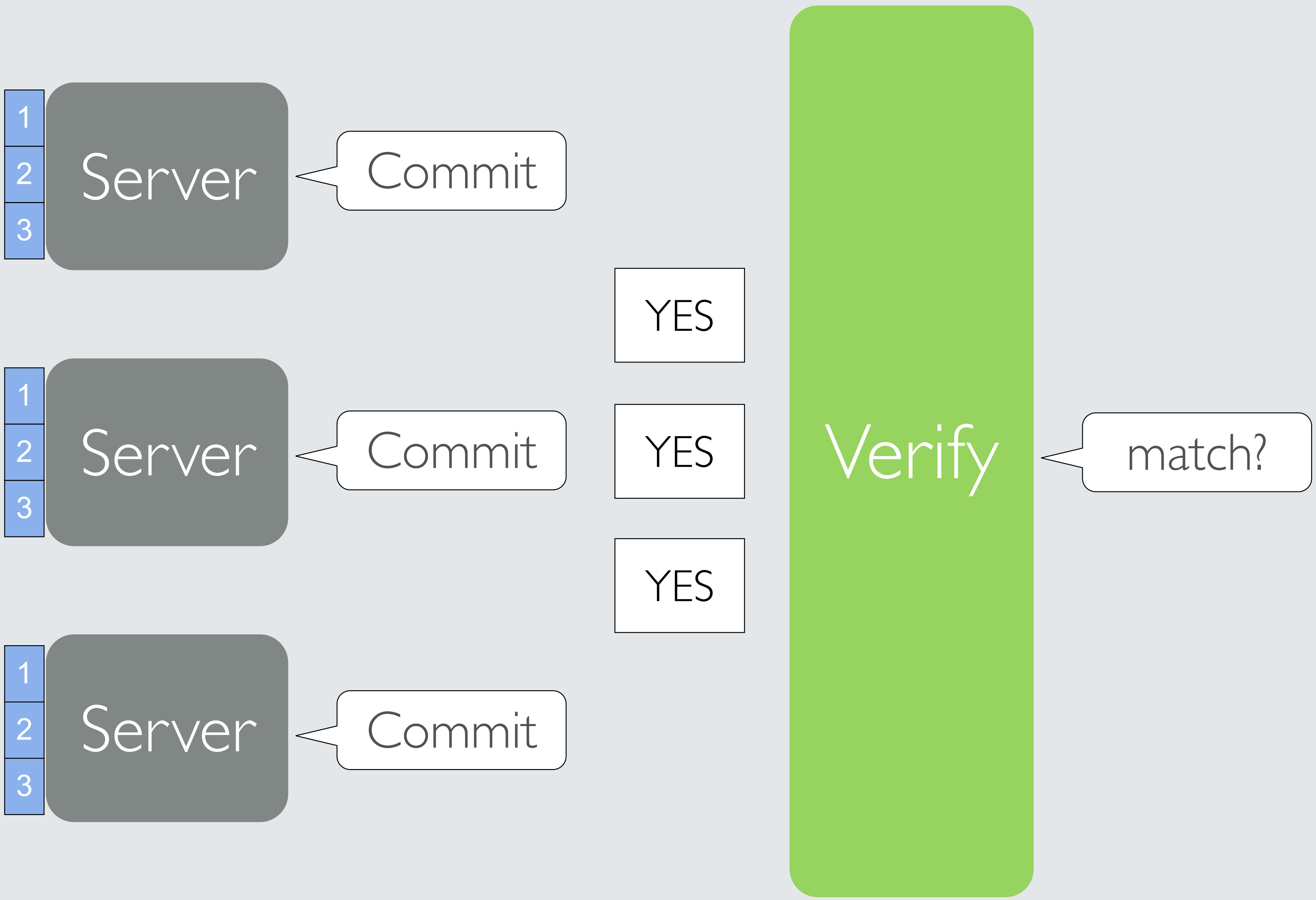
First execute…

(multithreaded and without agreeing on the order)

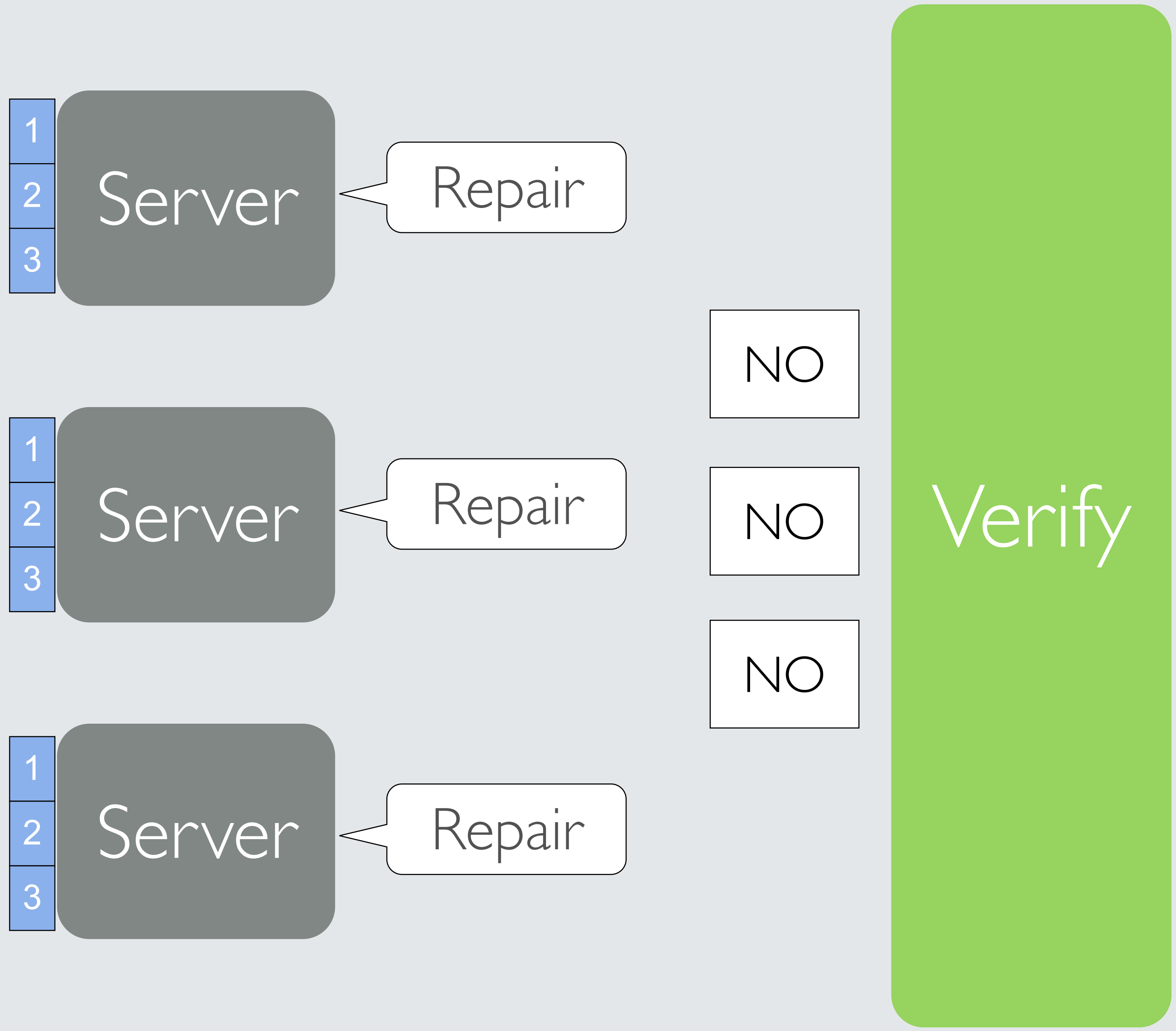
…then verify

(that replicas agree on the outcome)

On convergence



ON DIVERGENCE



Repair: rollback and re-execute sequentially

Eve's logic at a glance

Frequent

```
if (converged)
    commit
else
    repair divergence
```

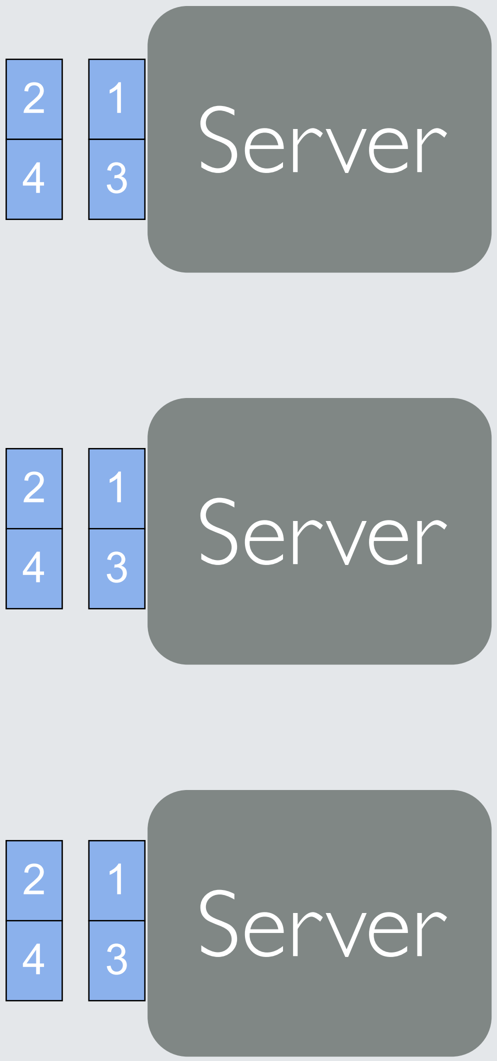
Uncommon

1. Make divergence uncommon

2. Detect divergence efficiently

3. Repair divergence efficiently

Making divergence uncommon



if (converged)
    commit
else
    repair divergence

Idea: identify commutative requests

Mixer: group together commutative requests

- Execute requests within a group in parallel

Mixer is a hint, not an oracle

EXAMPLE: TPC-W MIXER

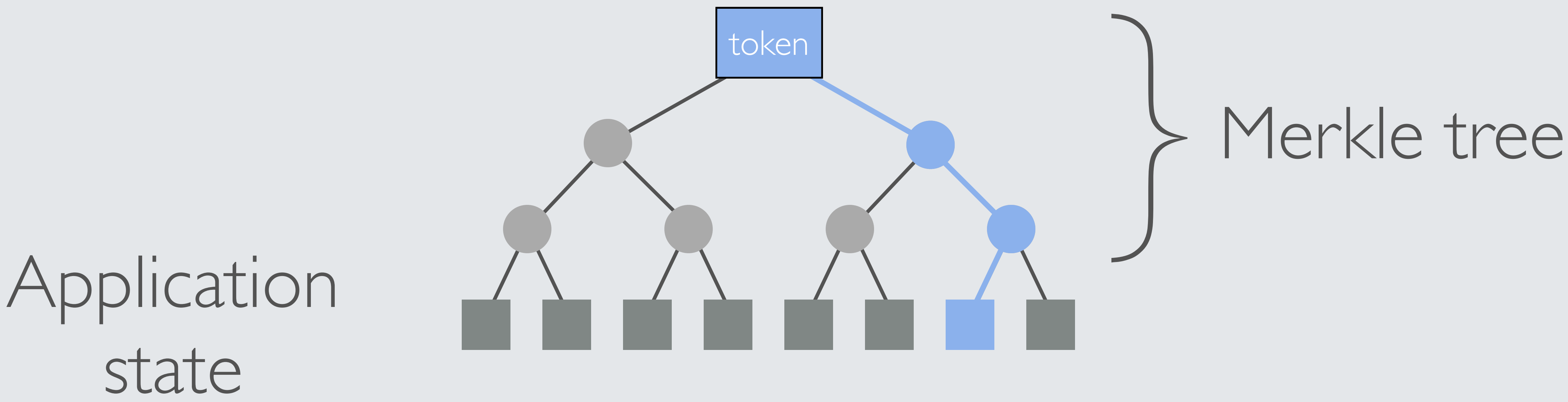
Transaction	Read tables	Write tables
getBestSellers	item, author, order_line	
doCart	item	shopping_cart_line, shopping_cart
doBuyConfirm	customer, address	order_line, item, cc_xacts, shopping_cart_line

3 frequent transactions of the TPC-W browsing workload

EFFICIENT DIVERGENCE DETECTION

Need to compare application states & responses frequently

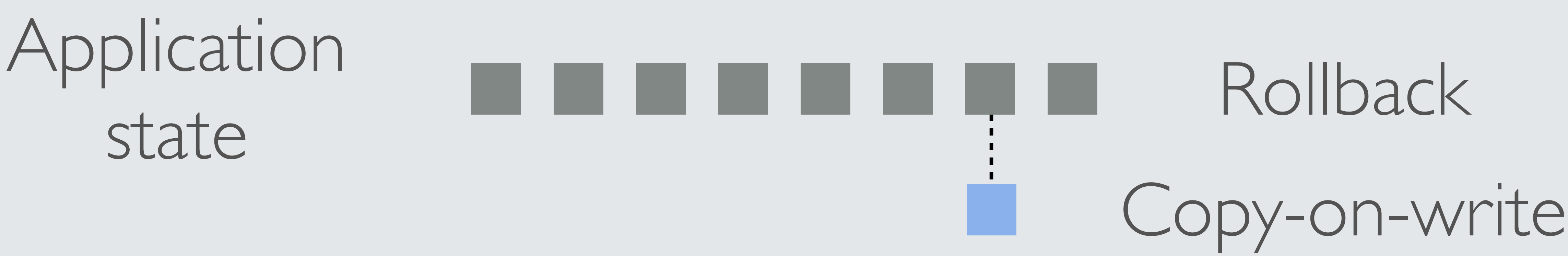
if (converged)
    commit
else
    repair divergence



EFFICIENT DIVERGENCE REPAIR

if (converged)
    commit
else
    repair divergence

Need to rollback application states after every divergence



```
if (converged)
    commit
else
    repair divergence
```

1. Make divergence uncommon

Mixer

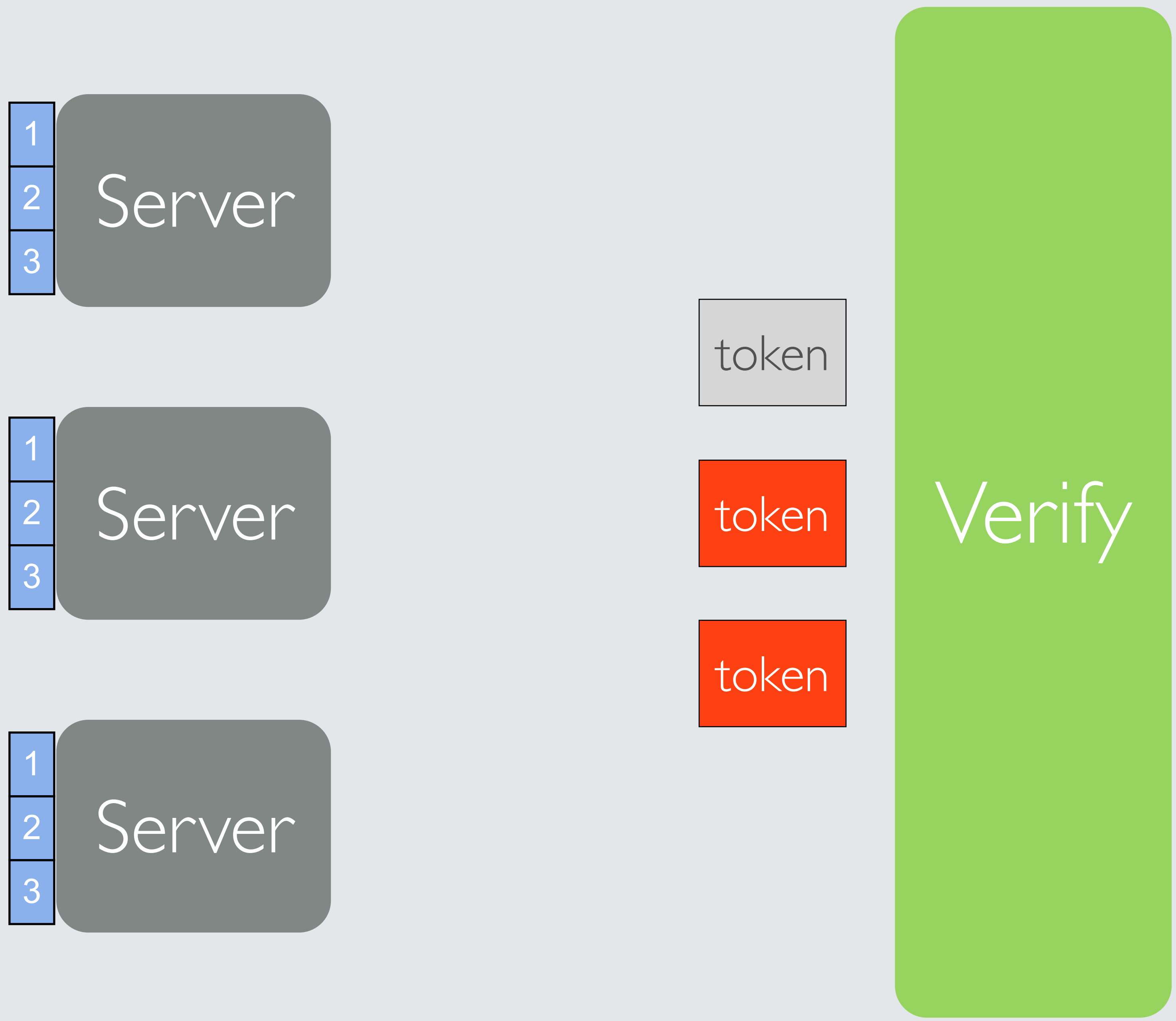
2. Detect divergence efficiently

Merkle tree

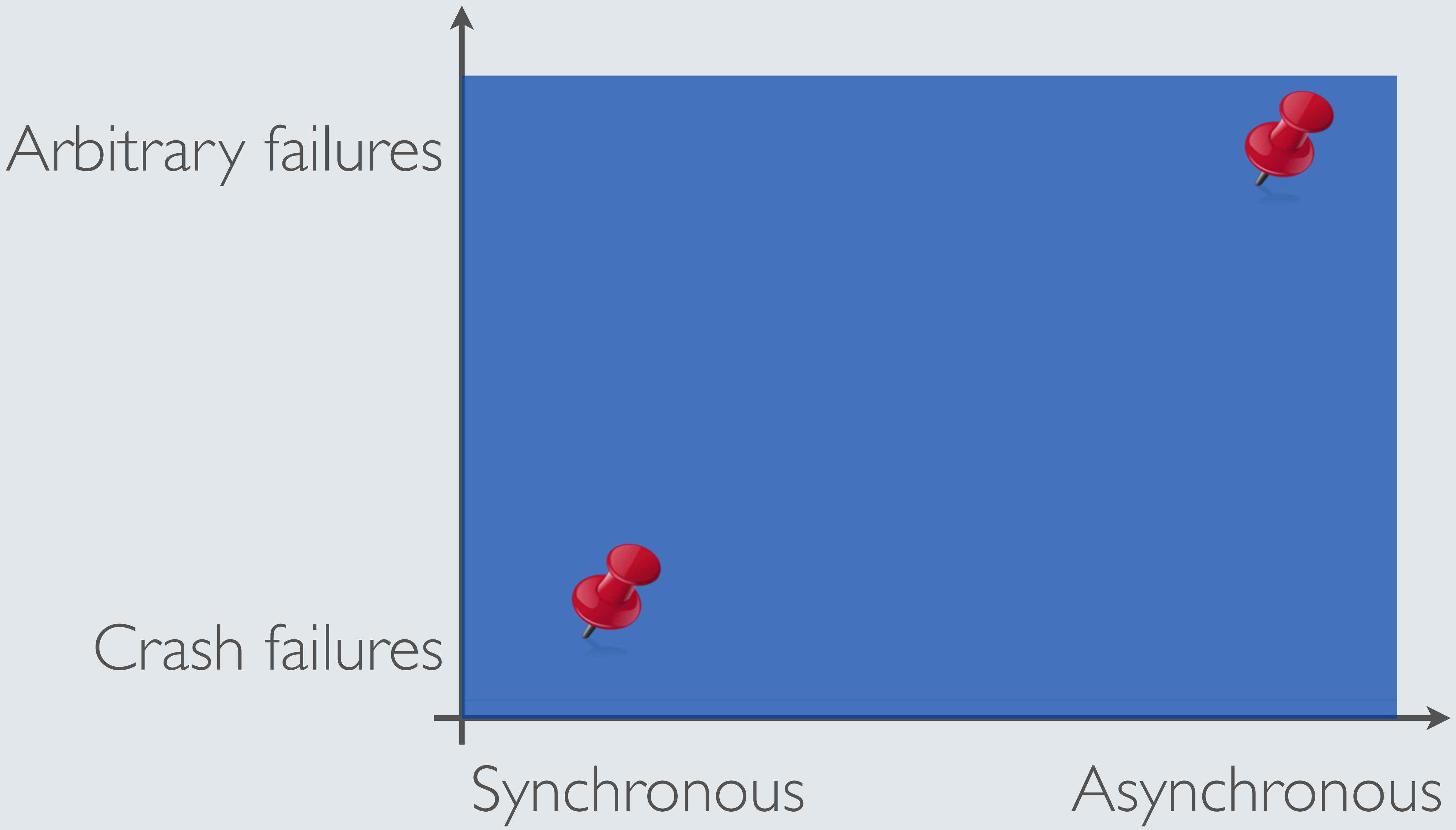
3. Repair divergence efficiently

Copy-on-Write

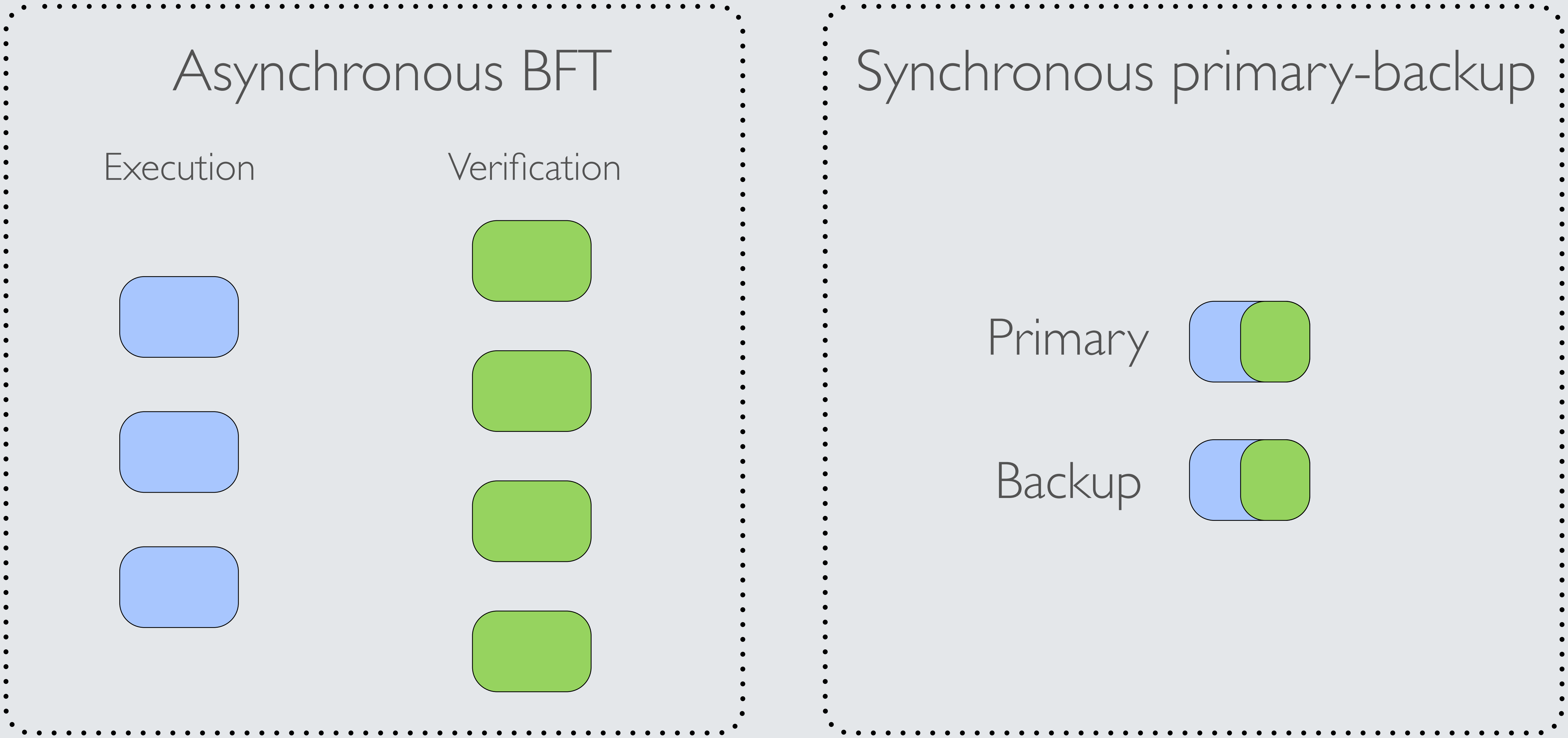
MASKING CONCURRENCY BUGS



Execute-Verify: An architectural change



Configurations



Tolerates 1 arbitrary fault

Tolerates 1 omission fault

EVALUATION

What is the performance benefit of Eve compared to traditional SMR systems?

Application: H2 Database Engine
Workload: TPC-W (browsing)