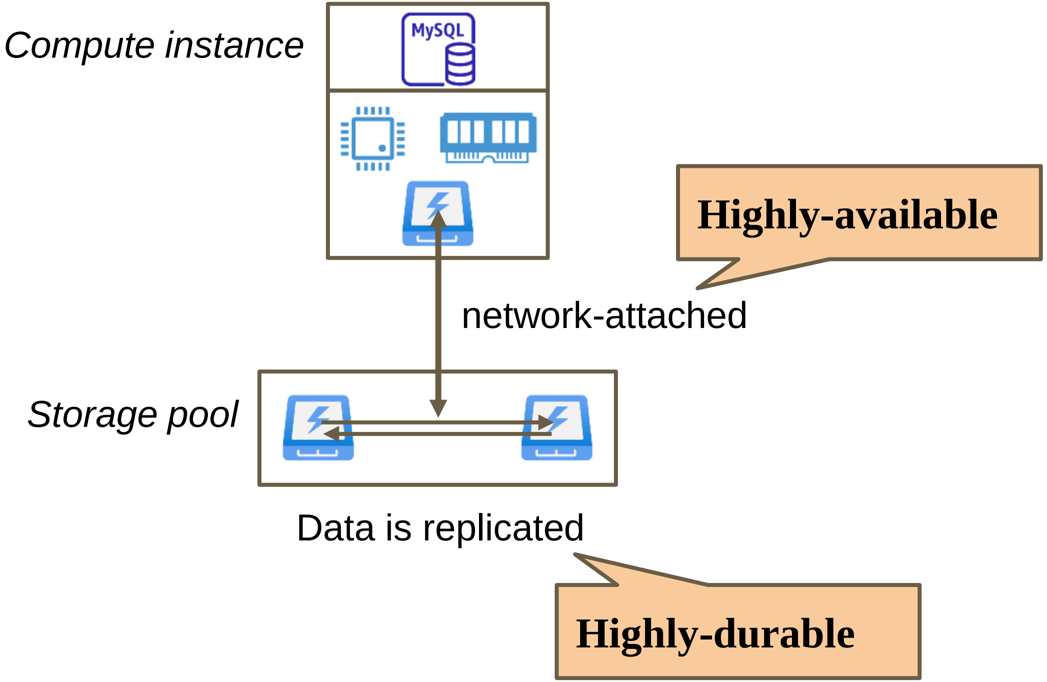
Speculative Recovery: Cheap, Highly Available Fault Tolerance with Disaggregated Storage

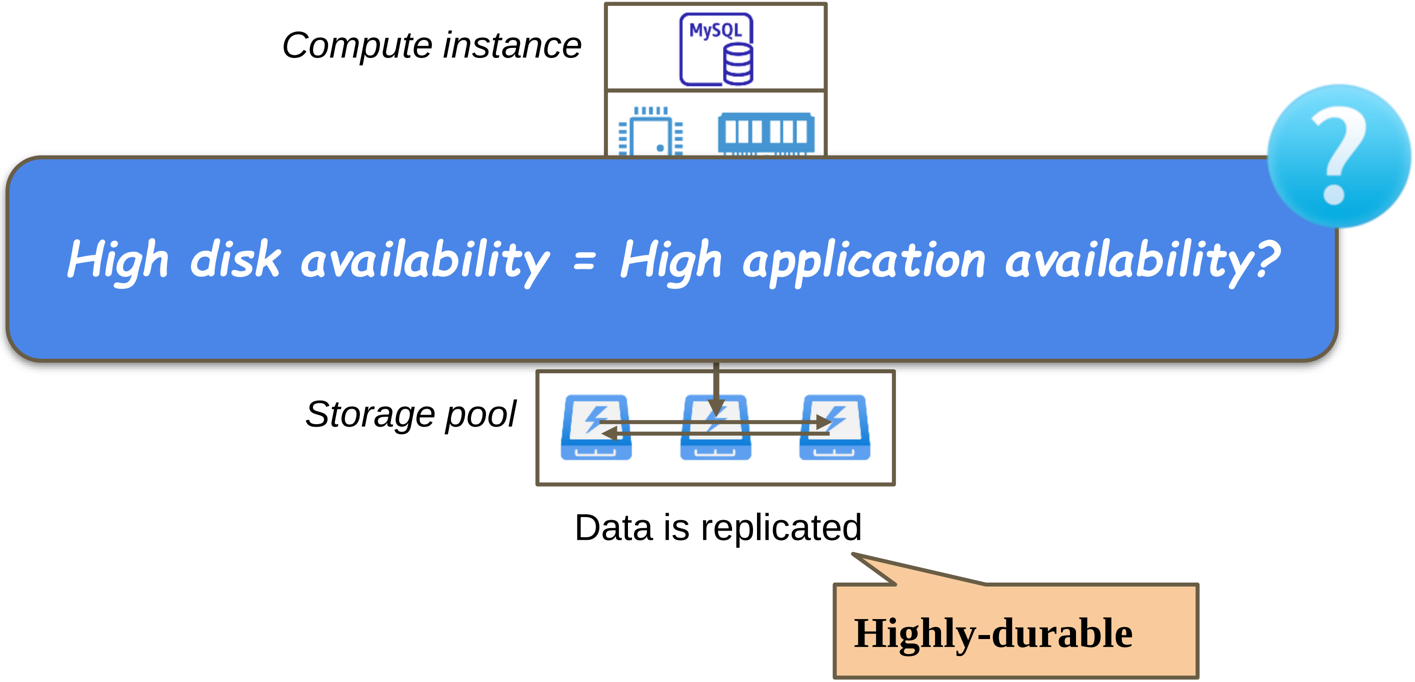
Nanqinqin Li, Anja Kalaba,
Michael J. Freedman, Wyatt Lloyd, and Amit Levy
Princeton University

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE
ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL
ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Disk "disaggregated:" highly durable and available



Disk “disaggregated:” highly durable and available

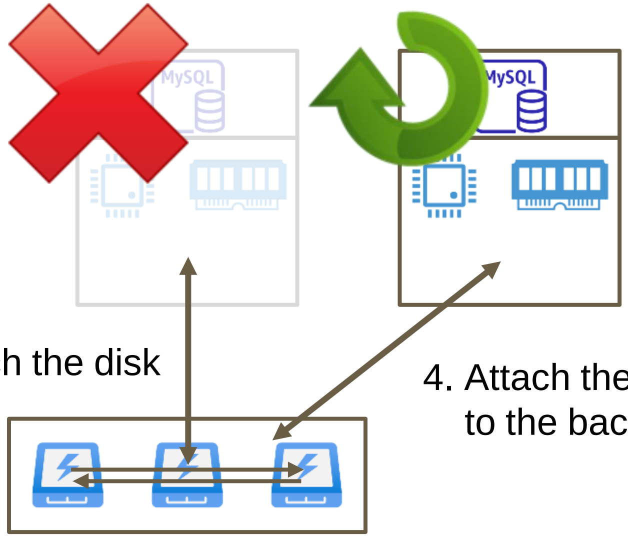


A nascent fault-tolerance technique

Fail, then recover

1. The *primary* failed

5. Restart the application



2. Spin up a new *backup*

3. Detach the disk

4. Attach the disk to the backup

<u>RE</u>covery from <u>D</u>isaggregated <u>S</u>torage (REDS)

REDS has low cost, and low availability as well

Low cost

Generally applicable to crash-consistent applications

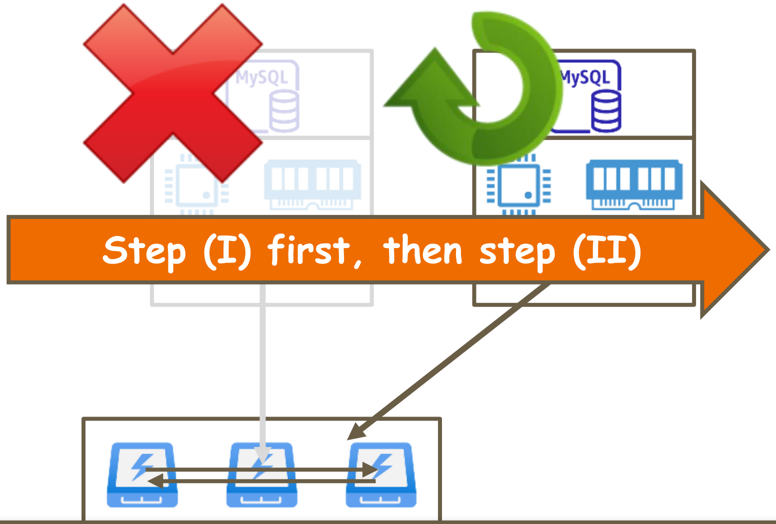
- Must persist writes before externalization
- Can recover from crashed state

Low availability

Why low availability #1: failover must be sequential

(I). Failure detection
on the *primary*

Timeout unresponsiveness

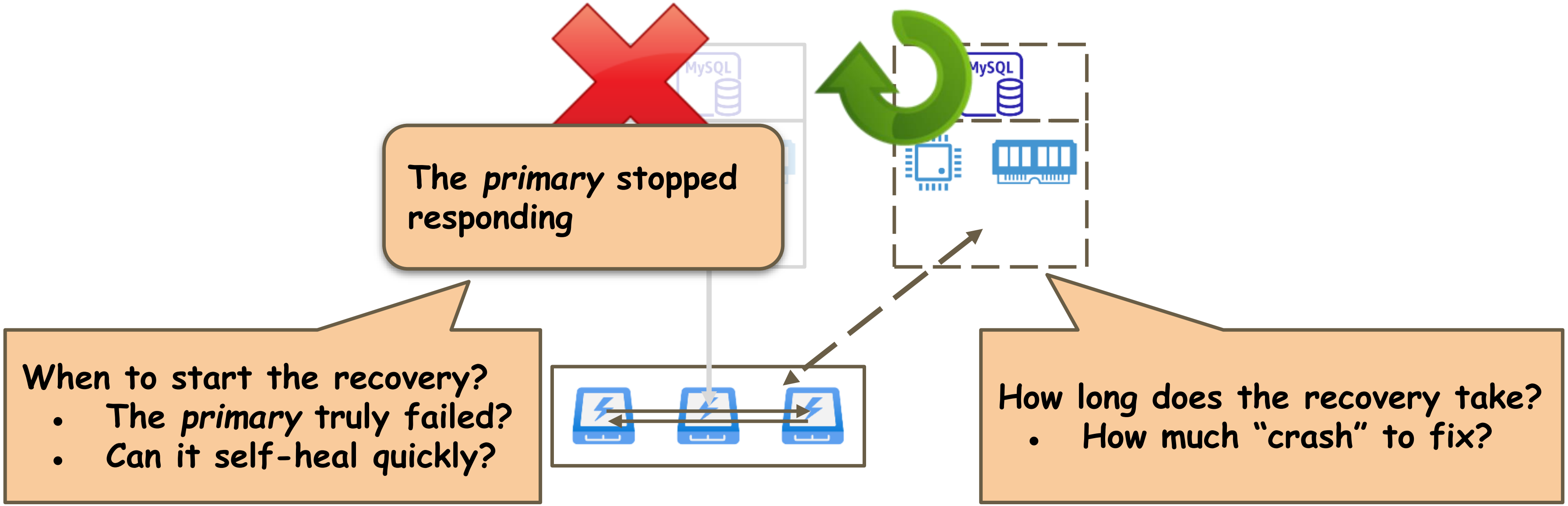


(II). Recovery
on the *backup*

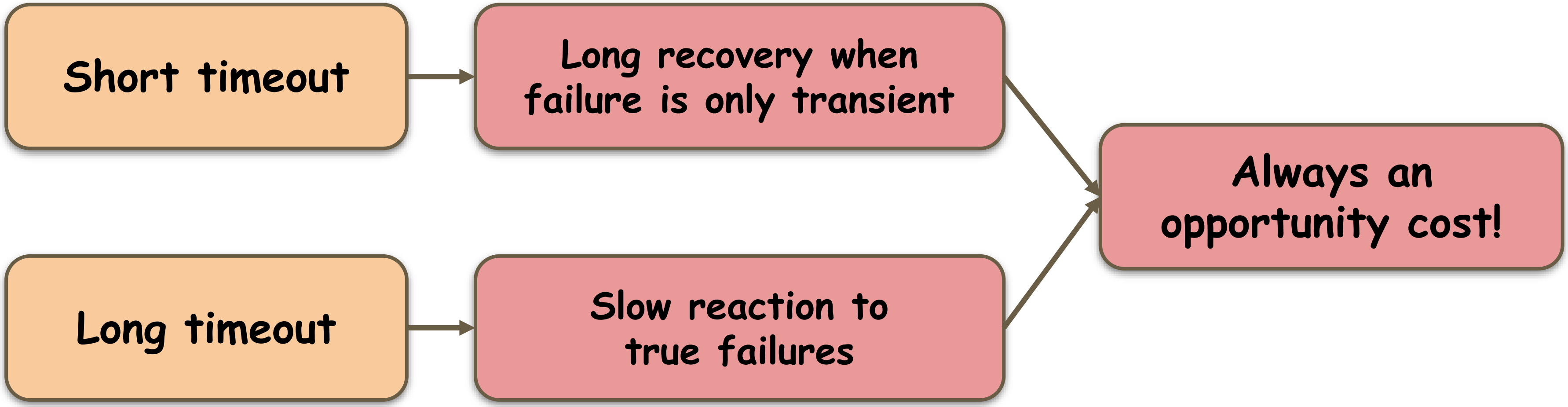
Move over the disk and restart the application

How long should step one (timeout) be?

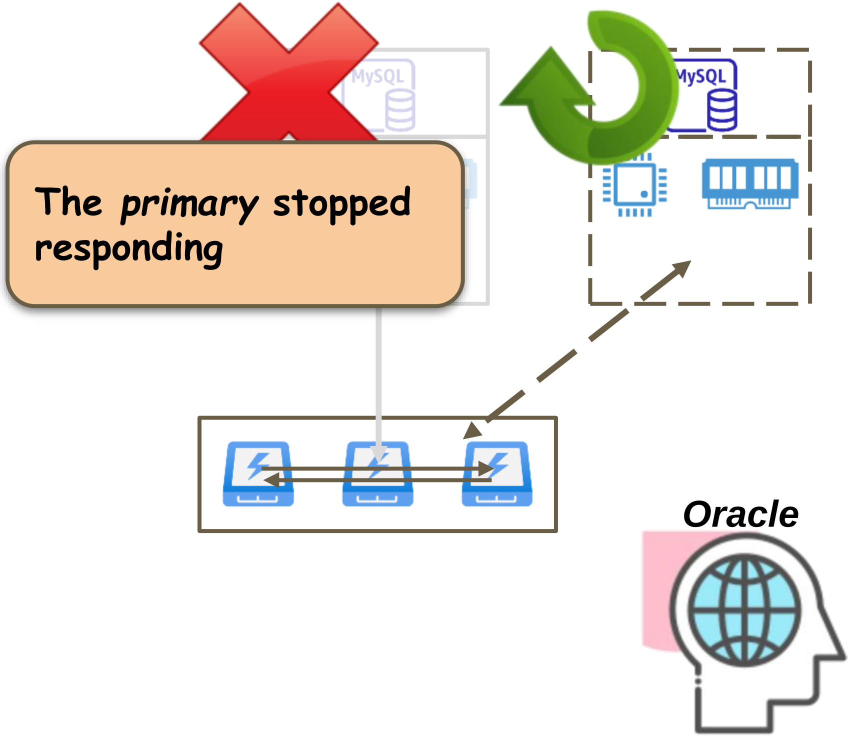
Why low availability #2: the future is unknown



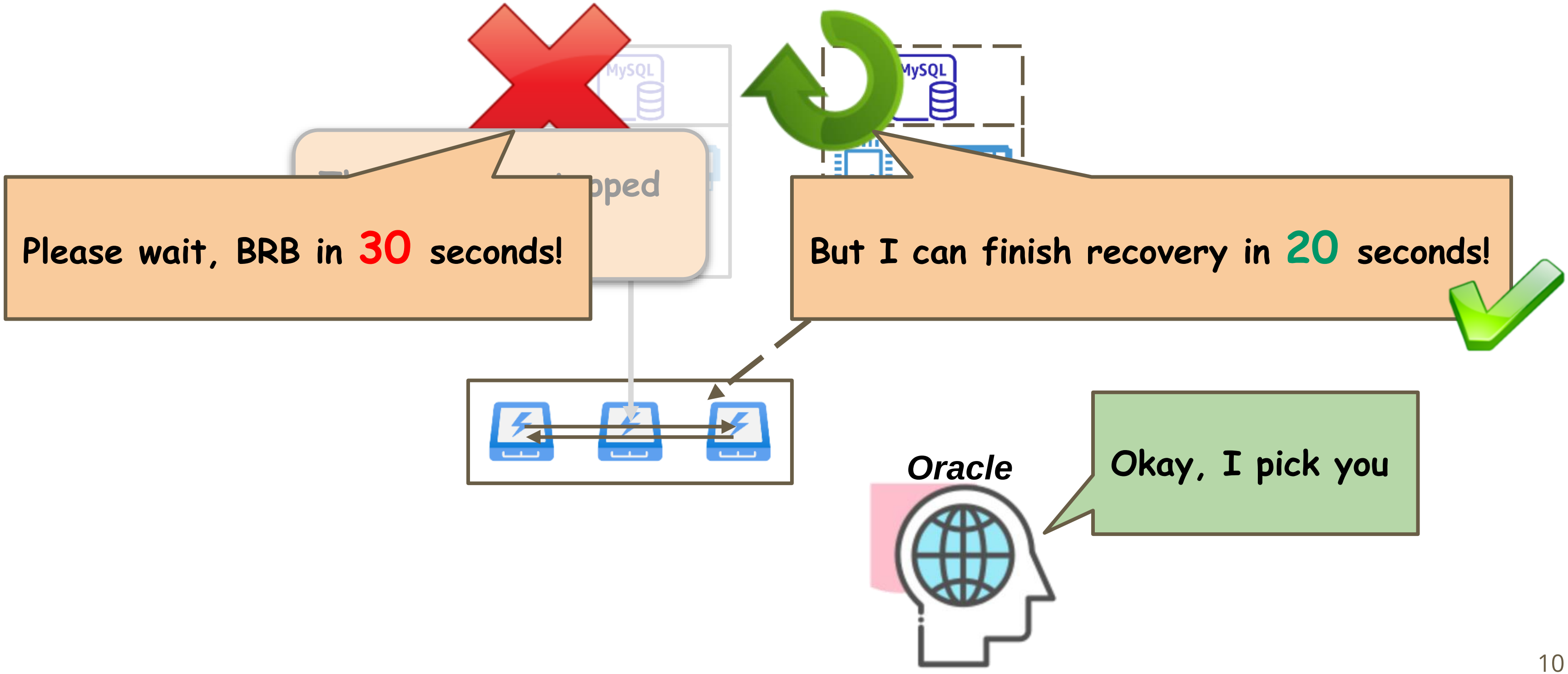
Why low availability #2: the future is unknown



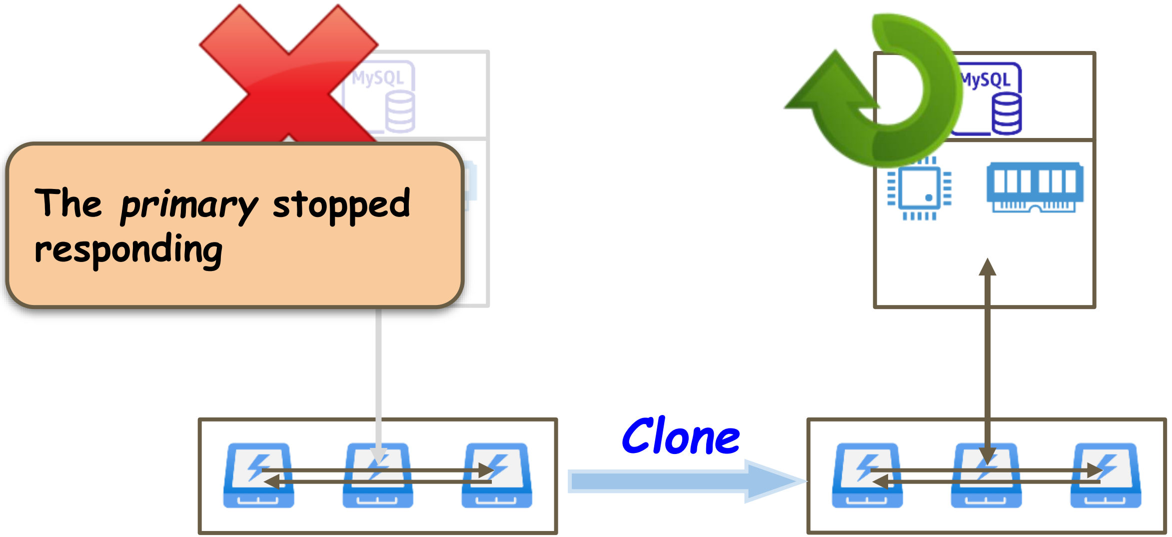
Suppose an Oracle knew the future



Suppose an Oracle knew the future



Speculative Recovery: similarly optimal decision!

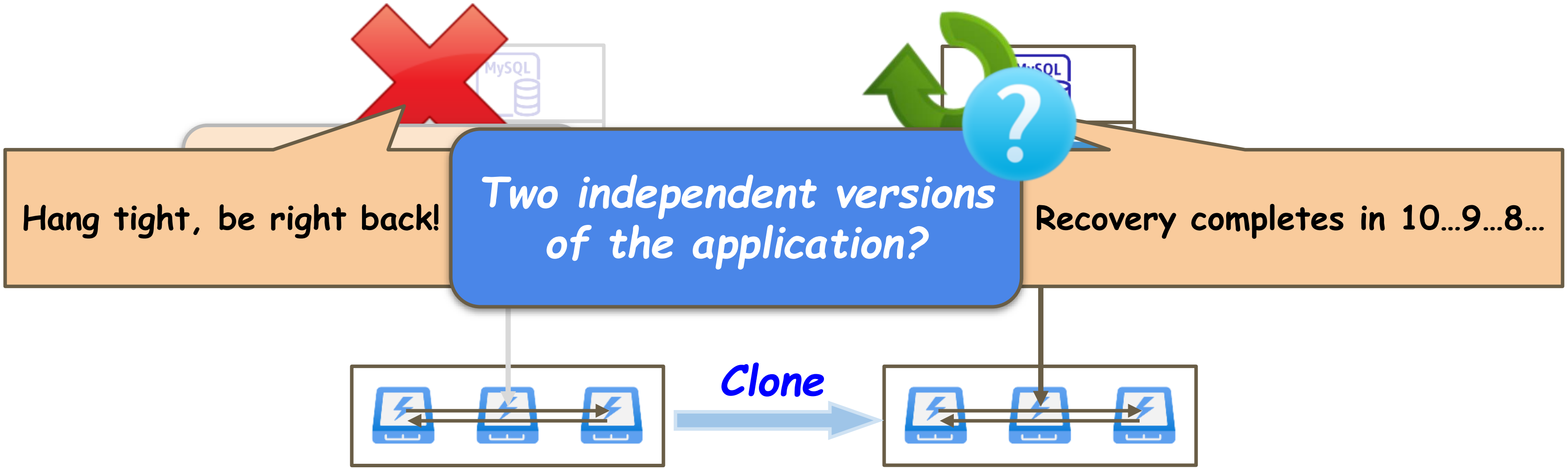


Speculative Recovery: similarly optimal decision!

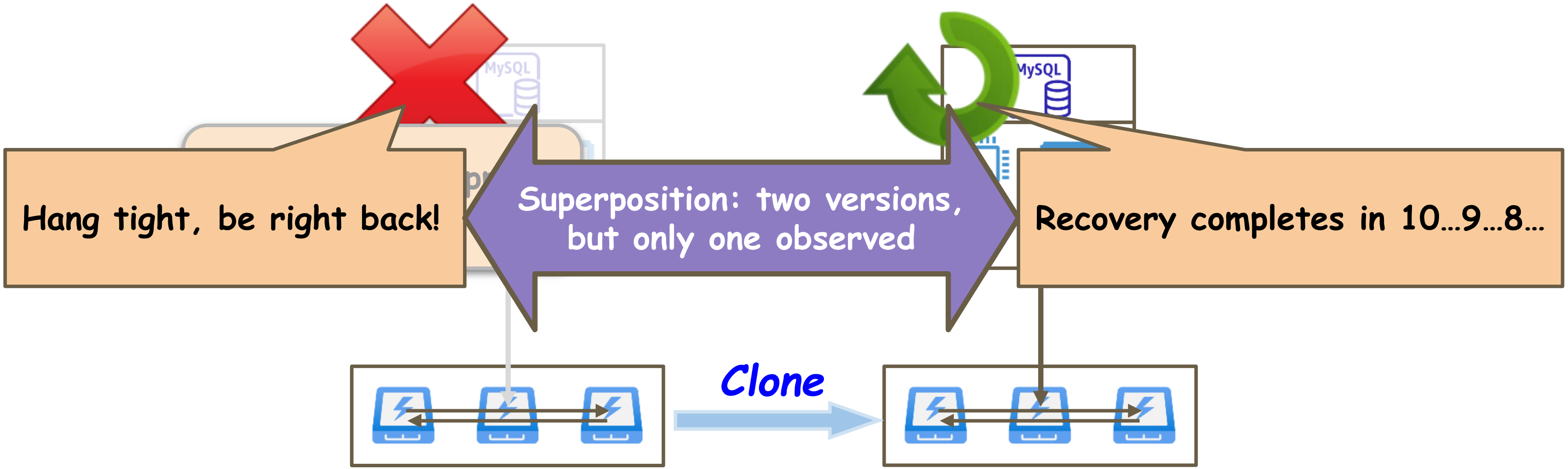
Outline

- Introduction
- **Speculative recovery and disk superposition**
- Two new primitives `super` and `collapse`
- Evaluation

Speculative Recovery creates a “superposition”



Speculative Recovery creates a "superposition"

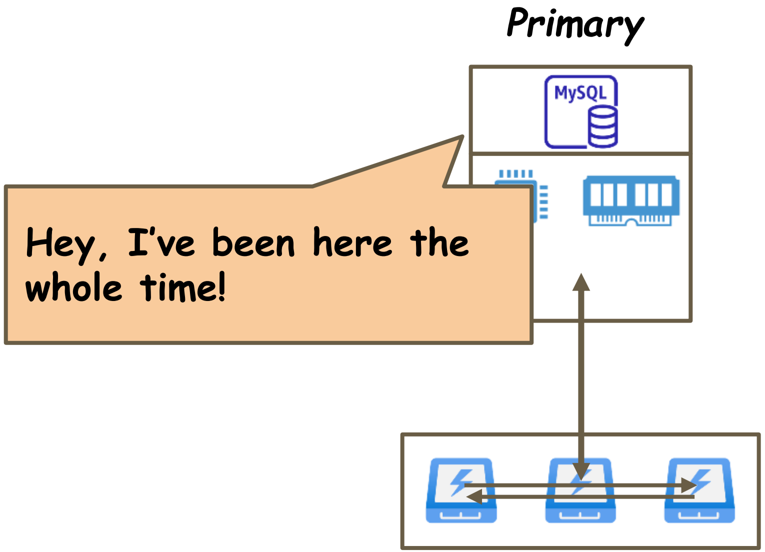


Speculative Recovery creates a “superposition”

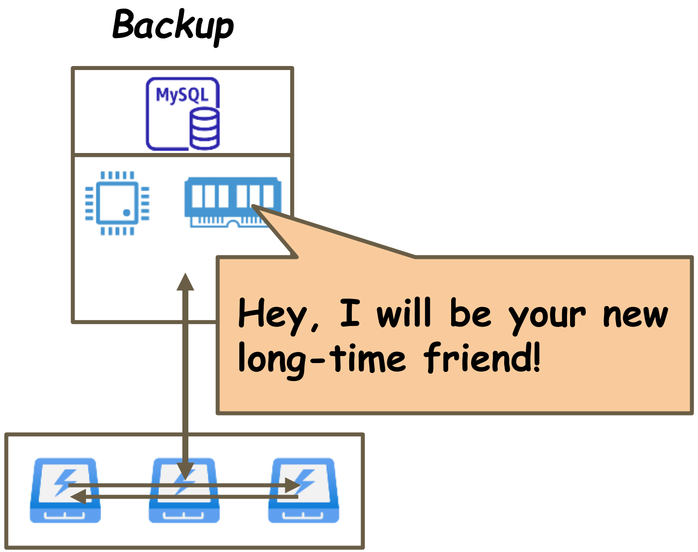
Schrödinger's Box

(Photo credit: https://www.sciencefocus.com/science/what-is-schrodingers-cat/)

Speculative Recovery creates a “superposition”



Speculative Recovery creates a "superposition"



Speculative Recovery creates a “superposition”

- **Creating** a superposition by creating a **disk clone**
  - The disk clone must be fast-to-create and performant
    - Copy-on-write has bad I/O performance after creation, especially under highly parallel write workload

- **Collapsing** a superposition when **one disk version is observed**
  - If the primary’s disk is **updated**, then the primary has been observed
    - Deallocating the backup
  - Otherwise if **no write** until the recovery completes
    - Promoting the backup and deallocating the primary

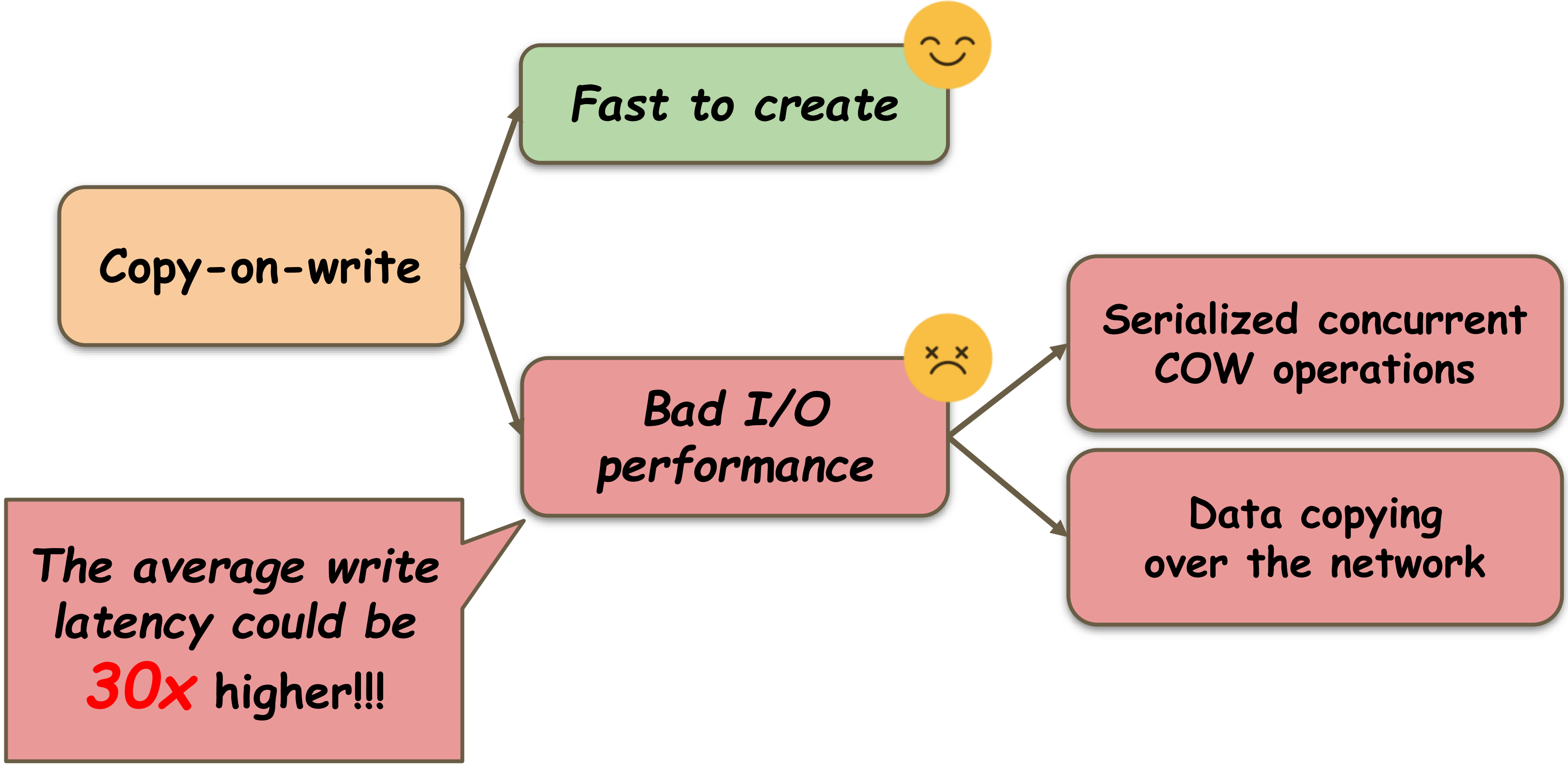
Outline

- ❑ Introduction
- ❑ Speculative recovery and disk superposition
- ❑ **Two new primitives `super` and `collapse`**
- ❑ Evaluation

A set of primitives: `super` and `collapse`

- **`super`**: creating a superposition by creating a disk clone
- **`collapse`**: collapsing the superposition by tracking writes to the primary's disk

`super` uses copy-on-write



super uses copy-on-write

More details in the paper!

Copy-on-write

Fast to create

Bad I/O performance

The average write latency could be 30x higher!!!

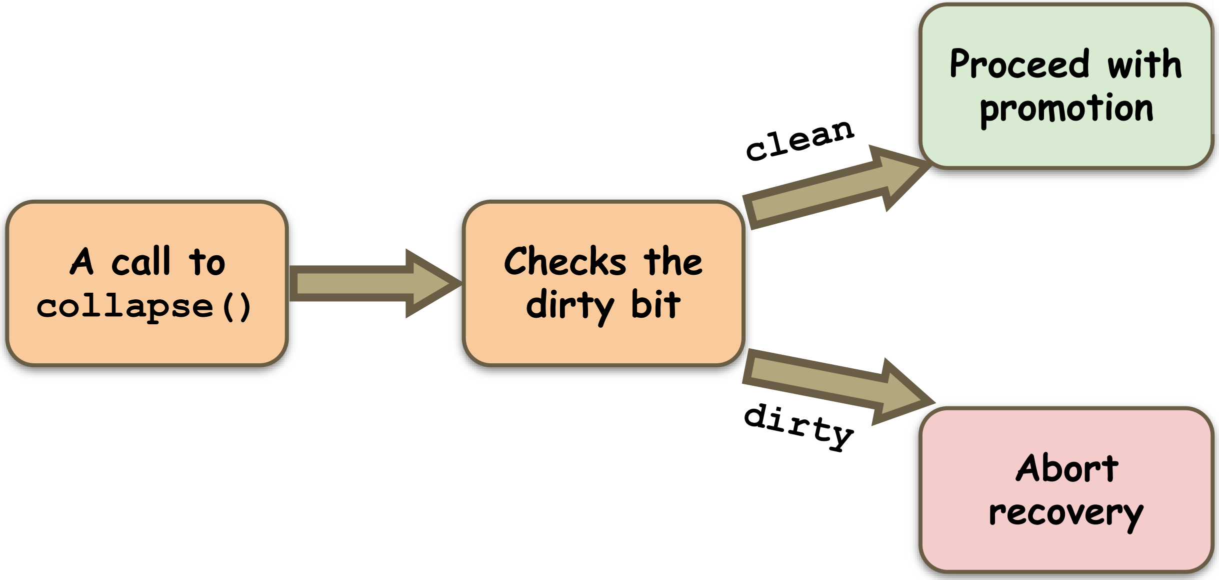
Collocated-clone

I/O performance close to a regular disk!

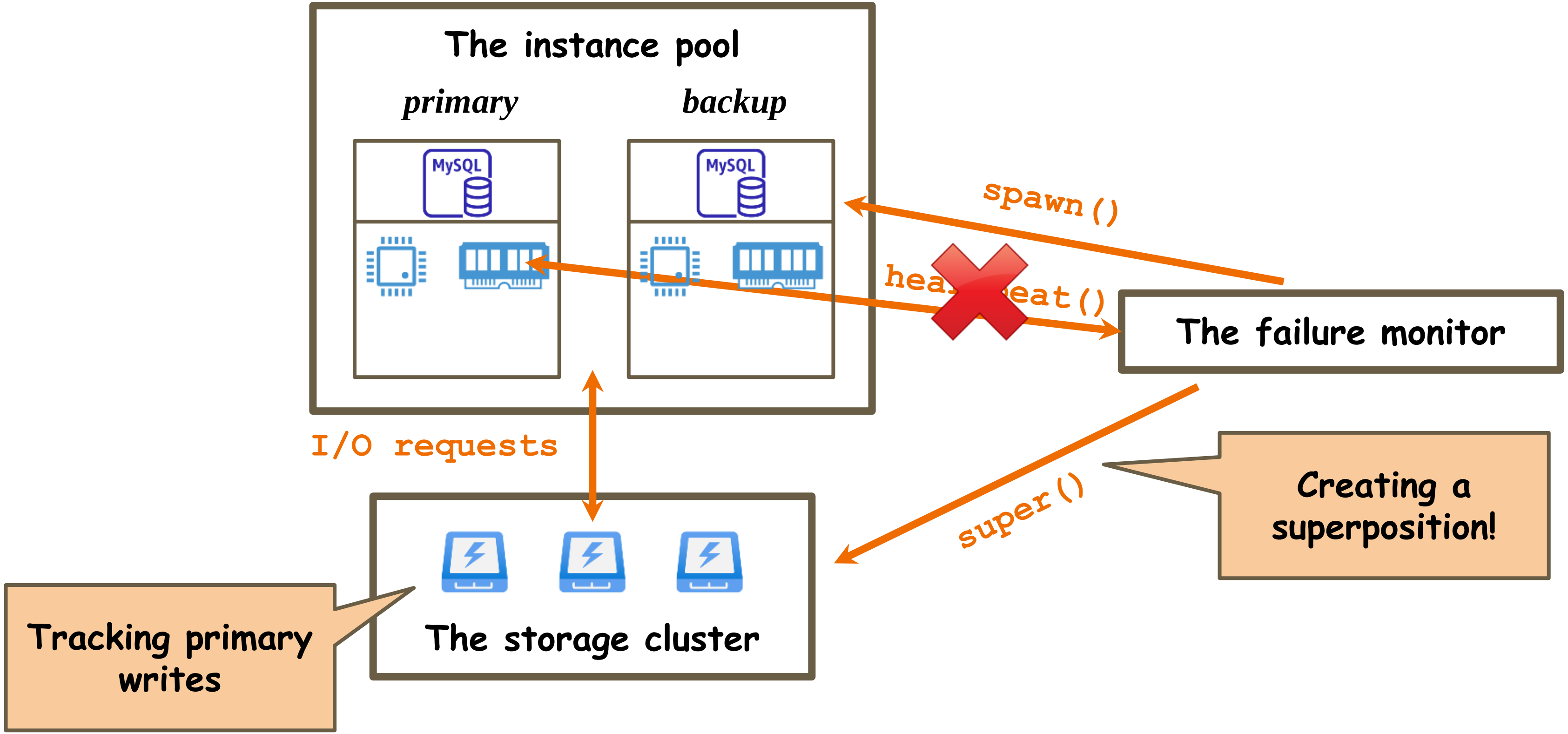
`collapse` uses a dirty bit to monitor writes

More details in the paper!

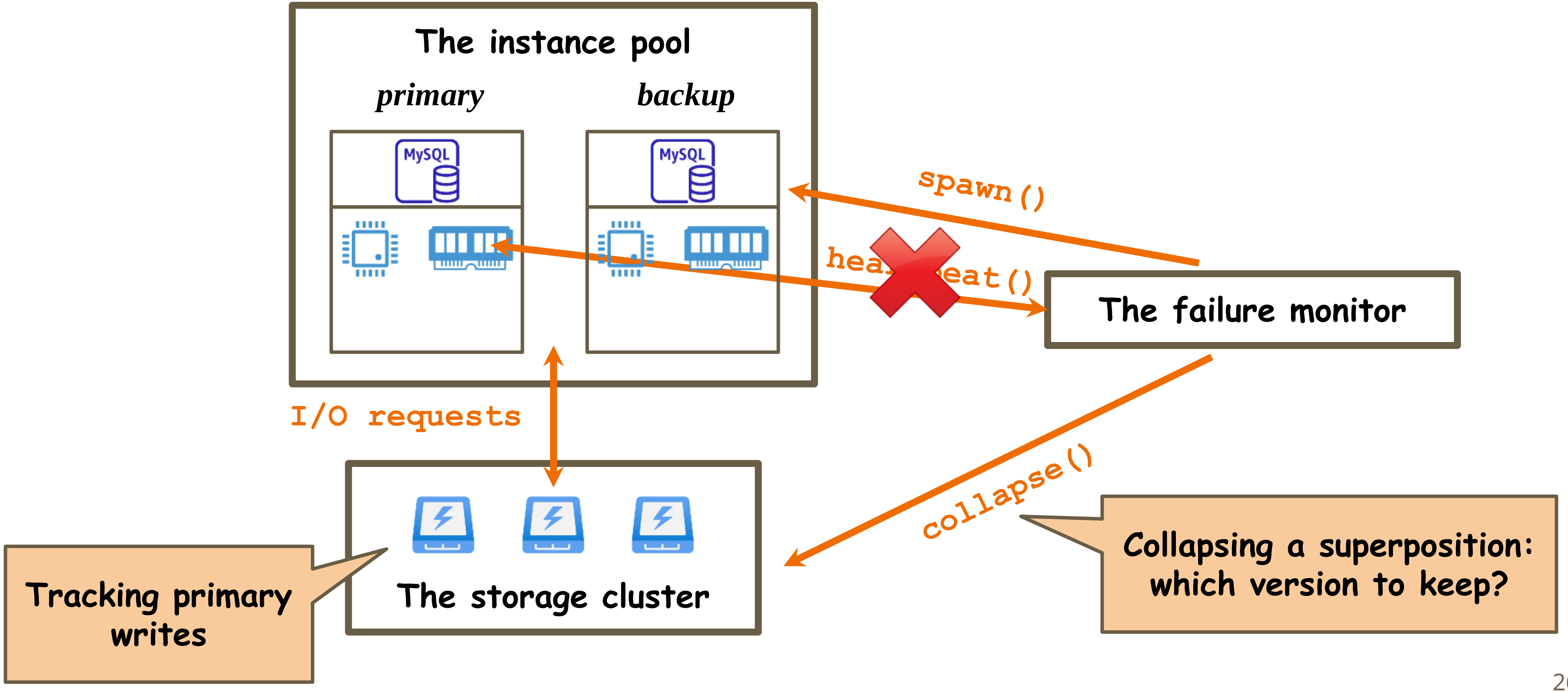
- A dirty bit for the primary's disk: **any write sets it dirty**



Three components of a speculative recovery system



Three components of a speculative recovery system



Outline

- Introduction
- Speculative recovery and disk superposition
- Two new primitives `super` and `collapse`
- **Evaluation**

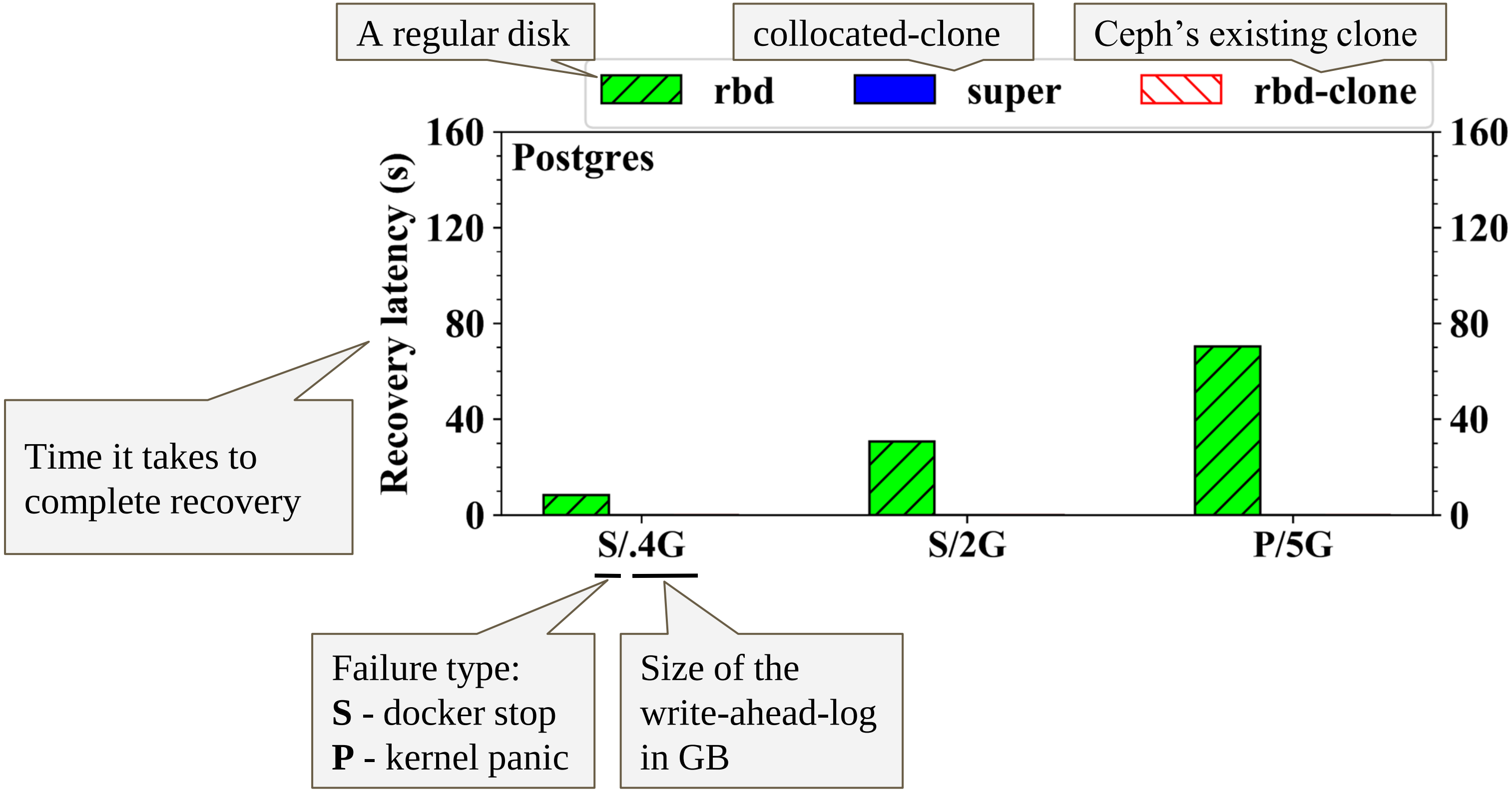
The experiment setup

- We implemented a prototype speculative recovery system: ***SpecREDS***
  - Based on Ceph's block device interface ***rbd***.
- The instance pool: ***docker containers***
- The storage cluster: high-end ***NVMe SSD*** drives

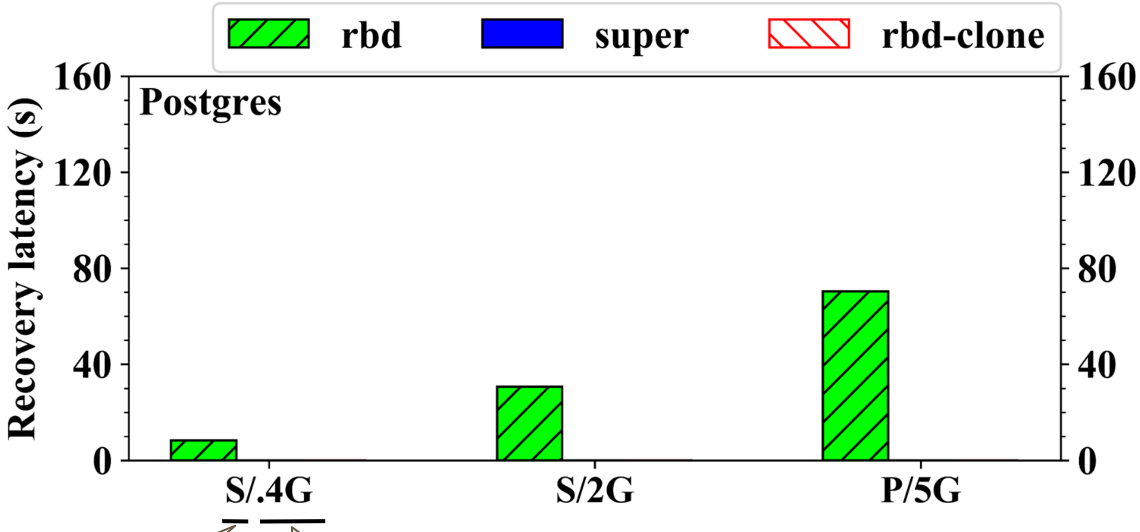
The experiment setup

- We compare three disk types
  - ***rbd*** (a regular disk)
  - ***rbd-clone*** (with Ceph's existing clone implementation)
  - ***super*** (with collocated-clone)
- We compare three systems
  - ***REDS*** (using rbd)
  - ***SpecREDS*** (using super)
  - ***Oracle*** (using rbd)
- Three database applications running ***TPC-C*** workload with ***oltpbench***:
  - ***MySQL*** with InnoDB
  - ***PostgreSQL***
  - ***MariaDB*** with RocksDB

Application recovery latency from various disk states



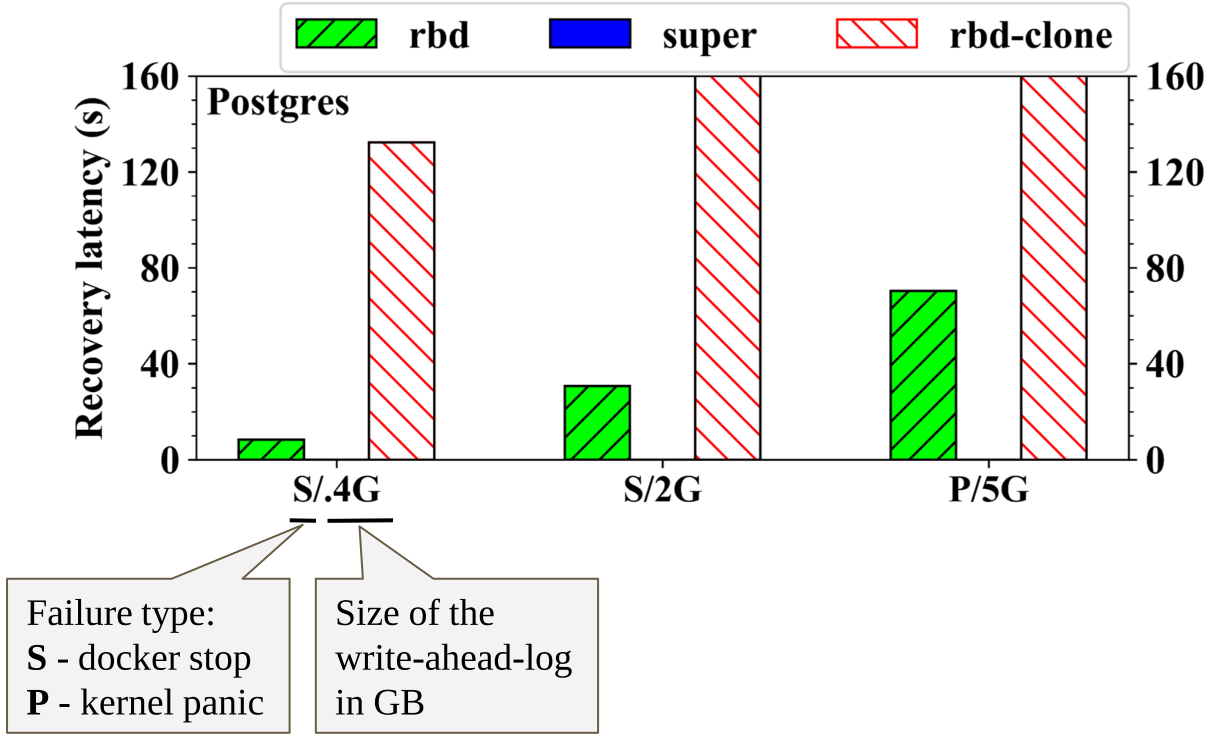
Application recovery latency from various disk states



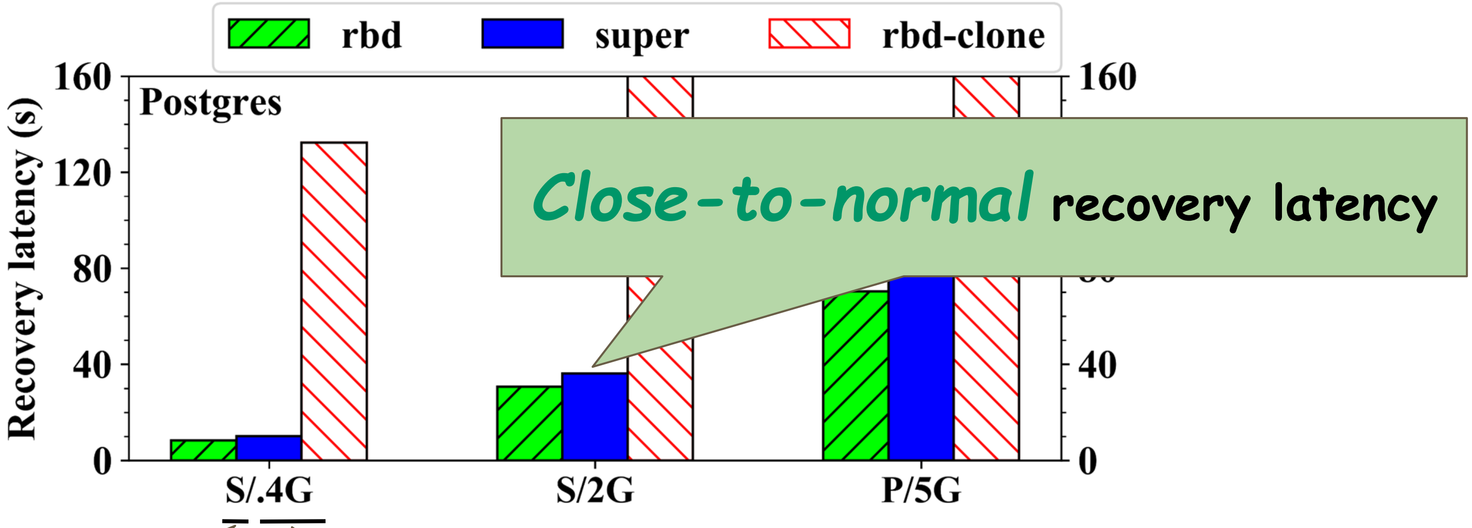
Failure type:
S - docker stop
P - kernel panic

Size of the
write-ahead-log
in GB

Application recovery latency from various disk states



Application recovery latency from various disk states

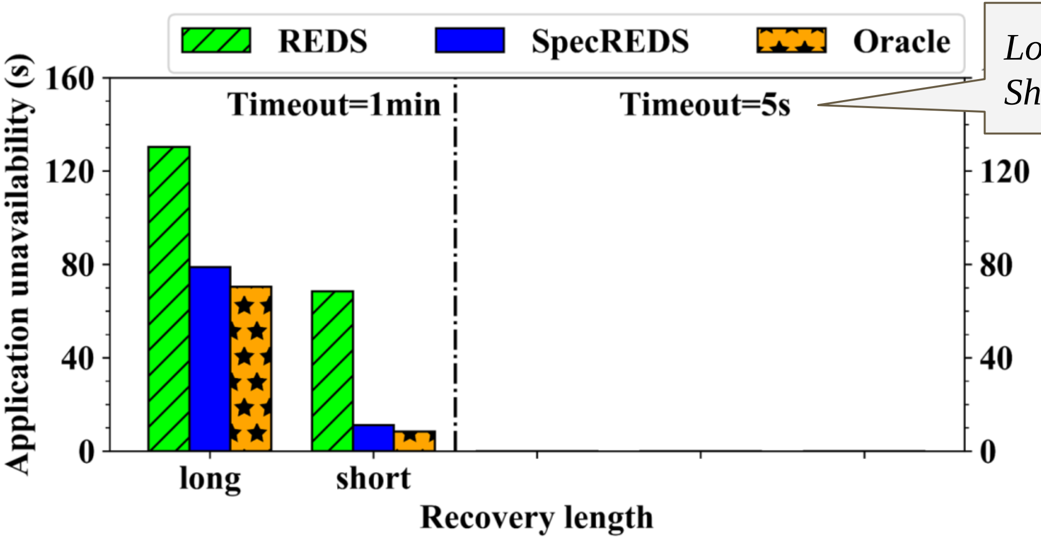


Failure type:
S - docker stop
P - kernel panic

Size of the write-ahead-log in GB

End-to-end failover latency with varying timeout/recovery

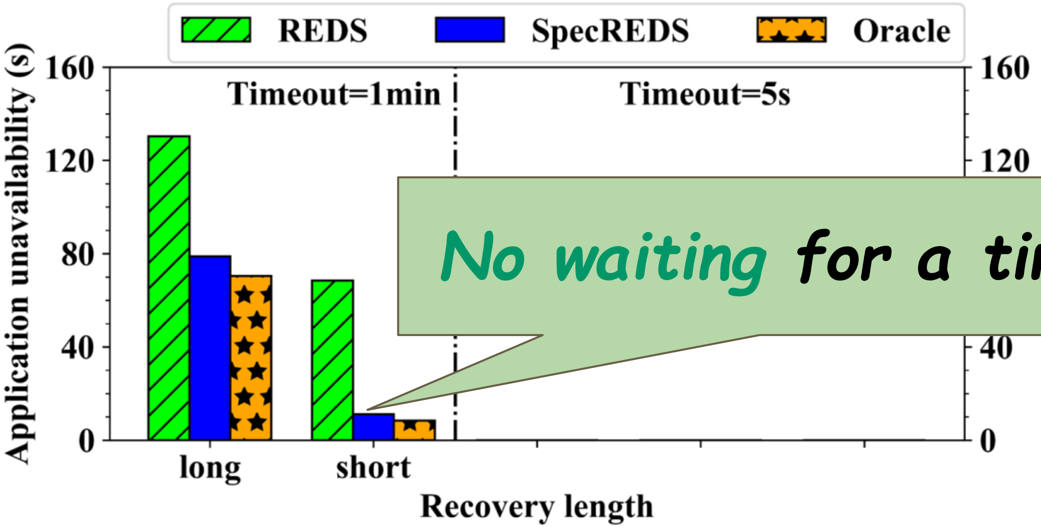
Simulation: adds latencies together



Long timeout – **1 minute**
Short timeout- **5 seconds**

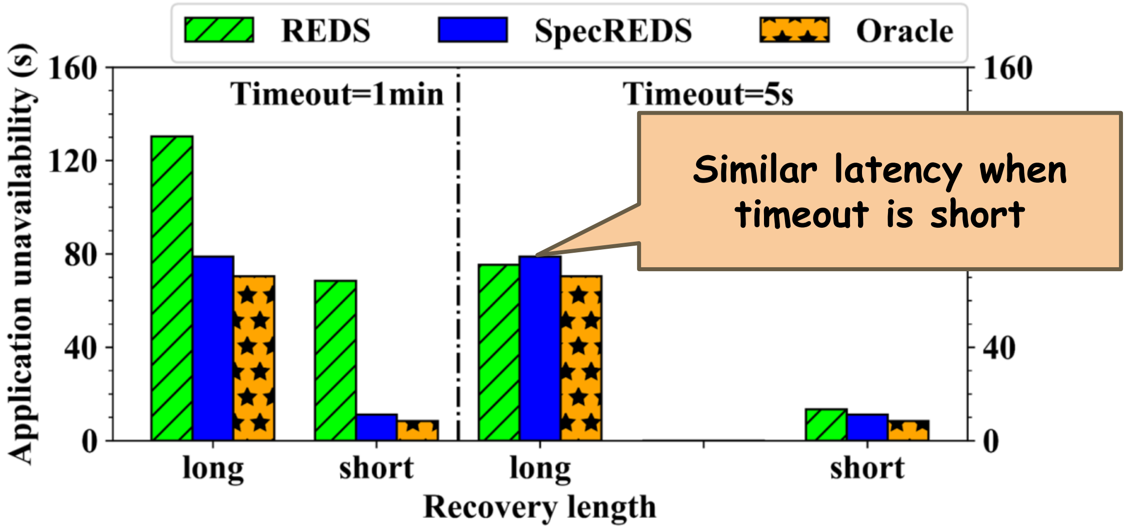
Long recovery - around **1 minute**
Short recovery - around **5 seconds**

End-to-end failover latency with varying timeout/recovery

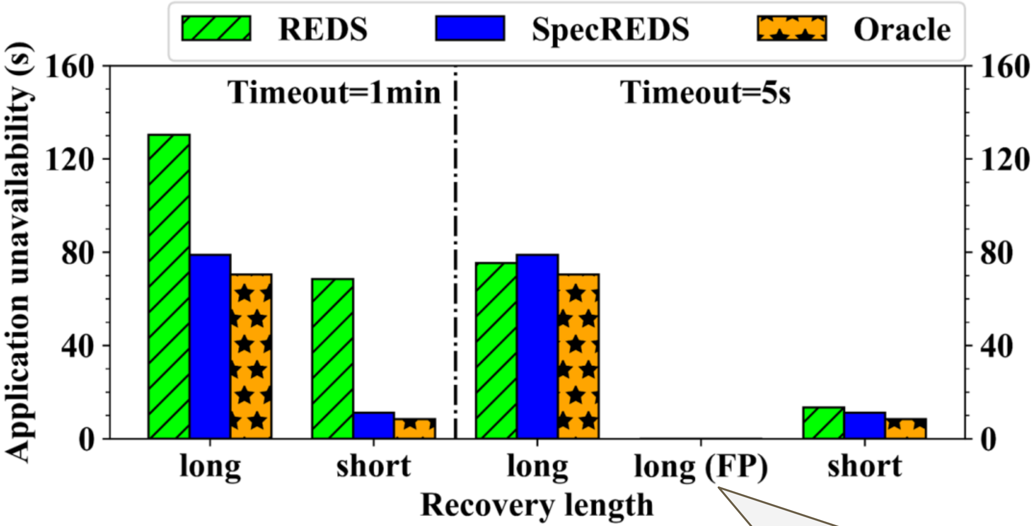


No waiting for a timeout!

End-to-end failover latency with varying timeout/recovery

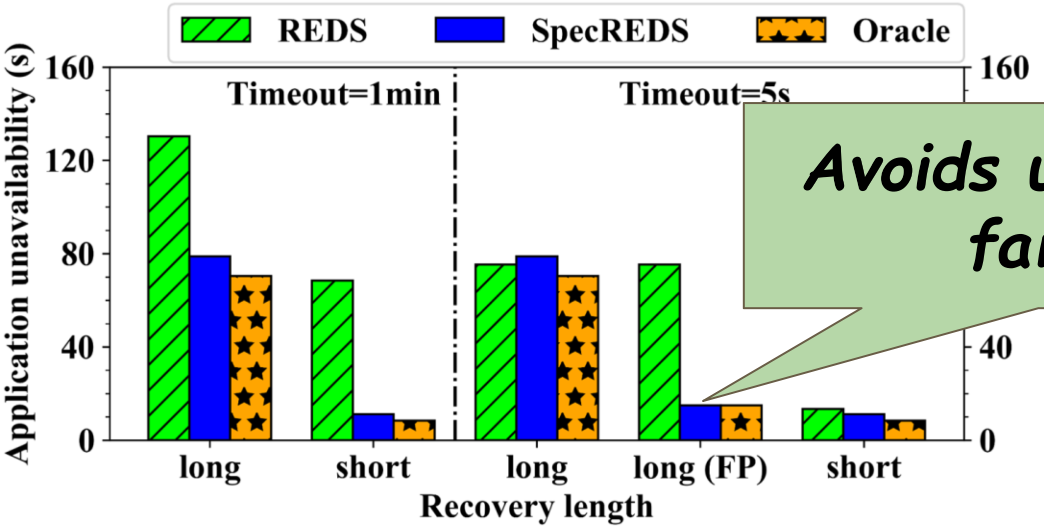


End-to-end failover latency with varying timeout/recovery



Simulated false positive:
The primary self-heals after **10** seconds

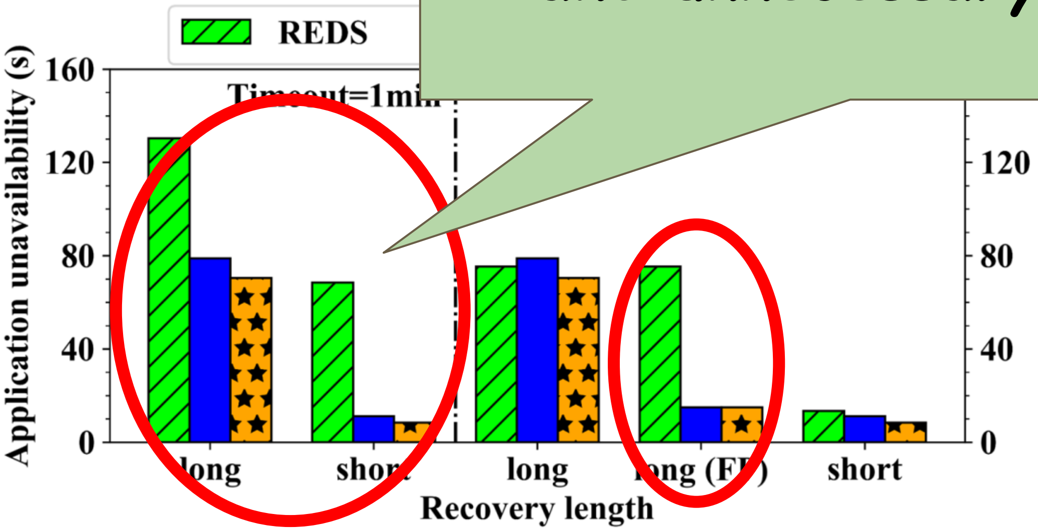
End-to-end failover latency with varying timeout/recovery



Avoids unnecessary failovers

End-to-end failover latenc

SpecREDS avoids long timeout and unnecessary failovers



Speculation once again improves performance!

Speculative recovery *safely* and *efficiently* parallelizes self-heal on the primary and recovery on the backup to push failover latency to the lower bound of REDS

Thank you!