Optimistic Recovery for High-Availability Software via Partial Process State Preservation

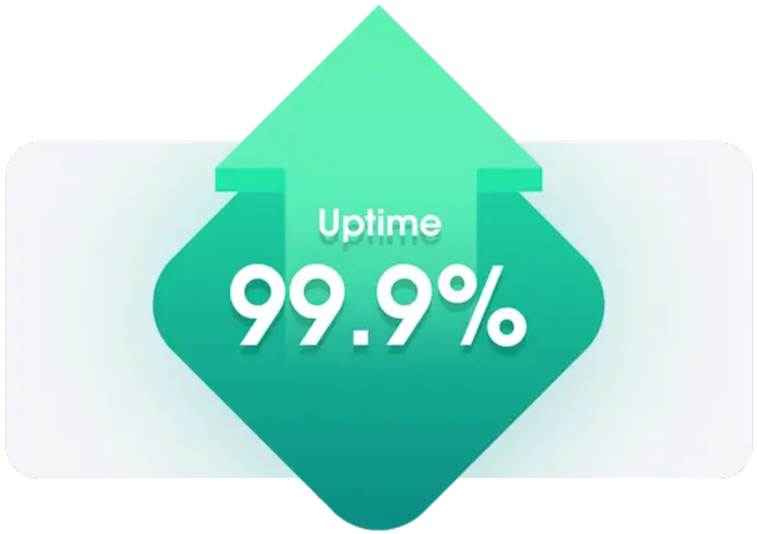
Yuzhuo Jing, Yuqi Mai, Angting Cai, Yi Chen, Wanning He,
Xiaoyang Qian, Peter M. Chen, Peng Huang

University of Michigan

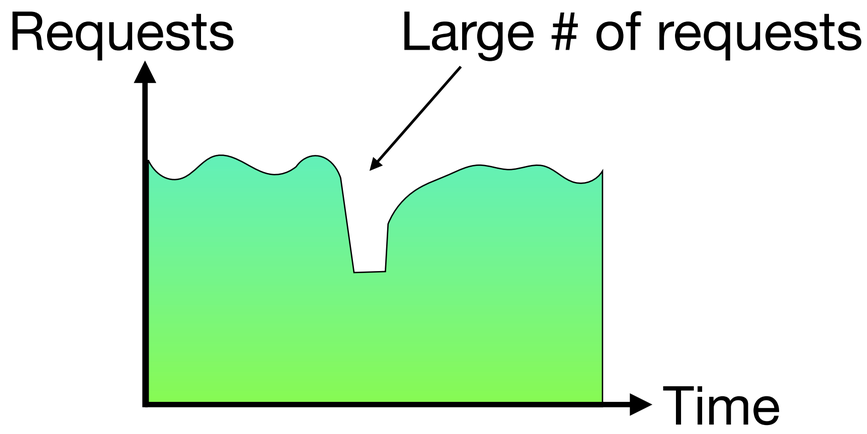
SOSP '25

MICHIGAN ENGINEERING
UNIVERSITY OF MICHIGAN

Availability Is Crucial

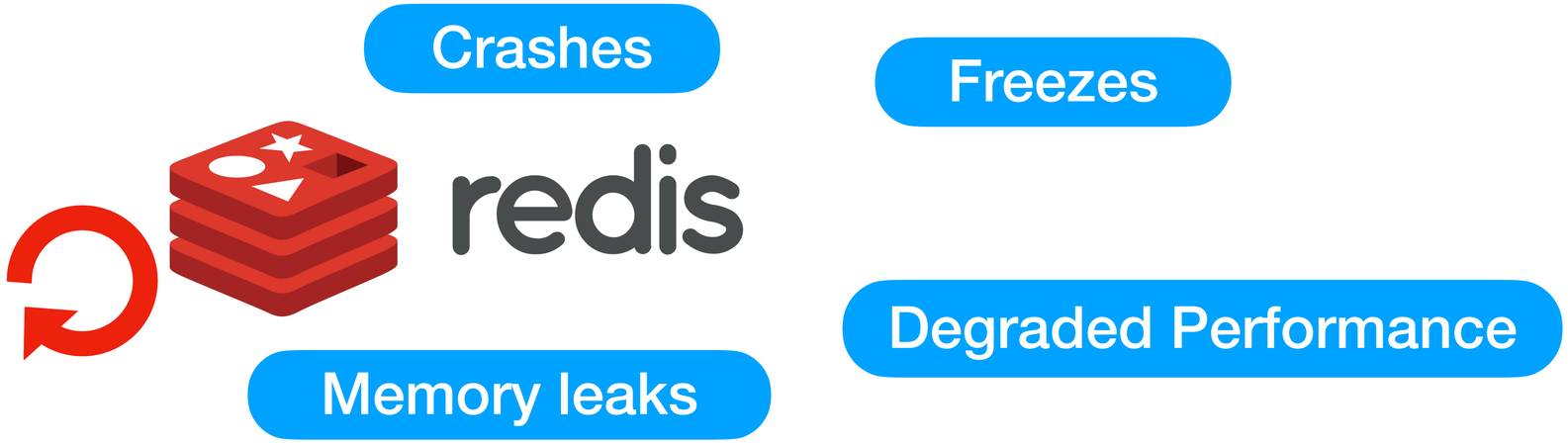


High SLO
Requirement



Brief downtime
but high impact

Software Faults Are Inevitable

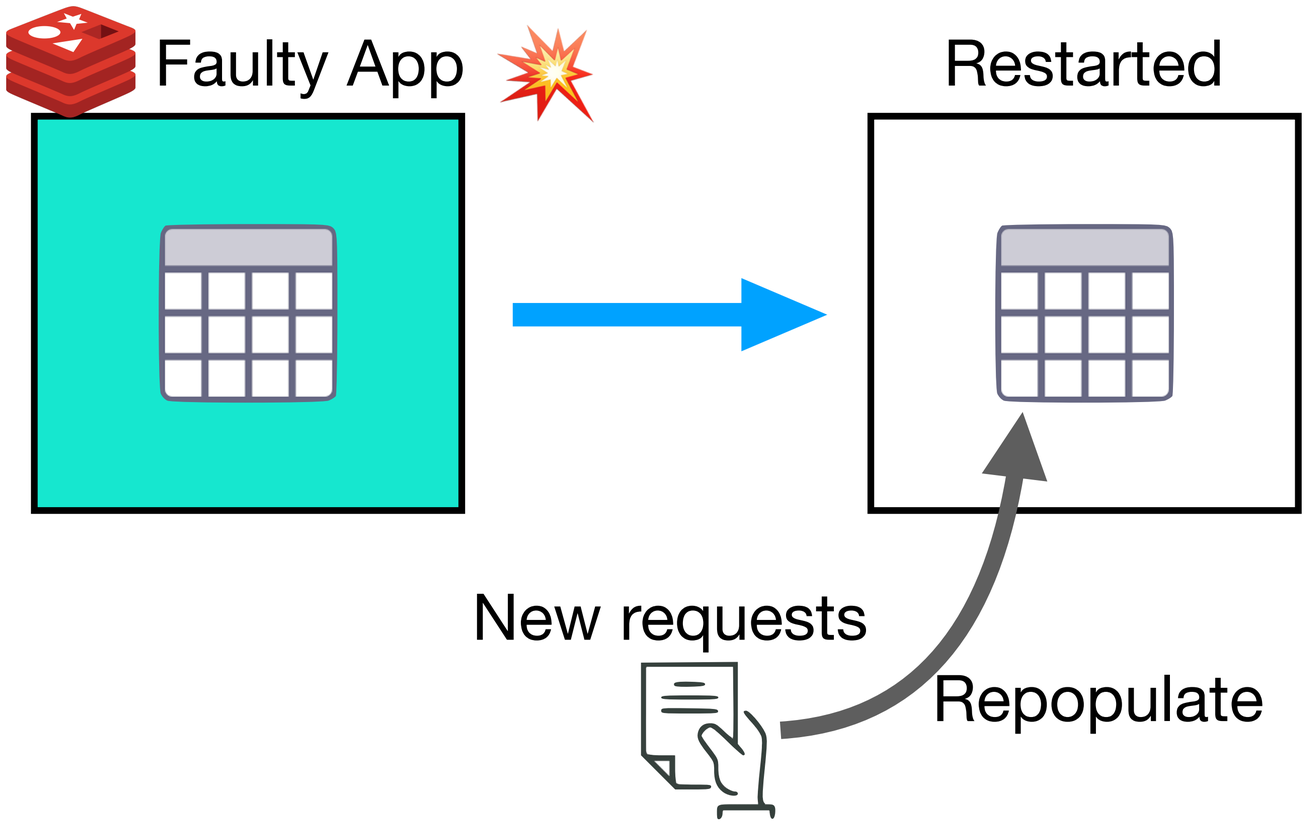


Recovery requirements:

1. **High availability**: must recover both *quickly* and *correctly*
2. **Meaningful availability**: service not only being *up* but also *high performance*

Recovery In Practice

Practice 1: Restart with Empty State



Pros:

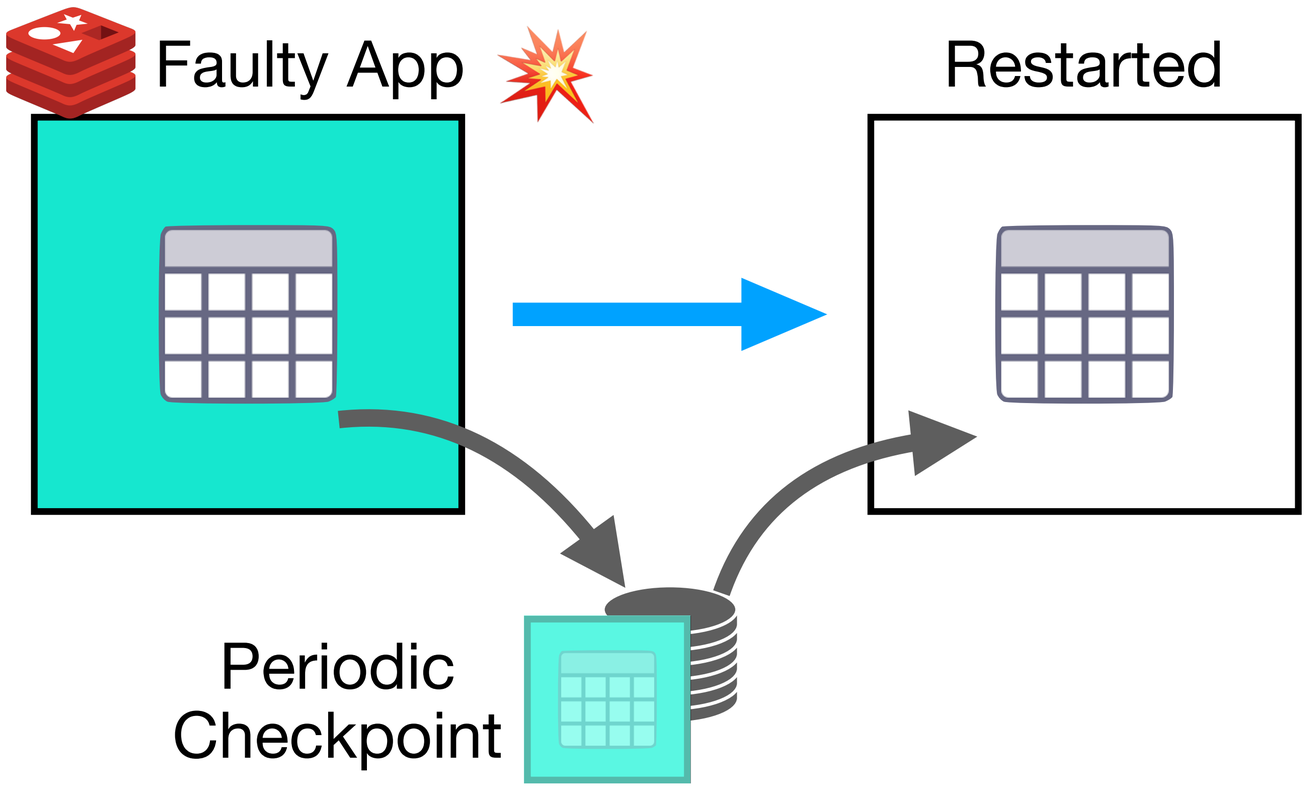
- Simplicity
- Eliminate bad state

Cons:

- Poor availability post-restart

Recovery In Practice

Practice 2: Process Checkpointing



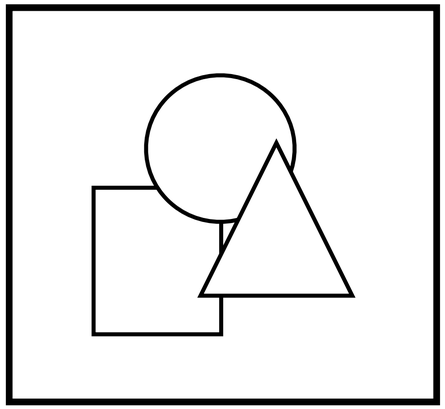
Pros:

- Better post-restart performance

Cons:

- Risks persisting faulty state

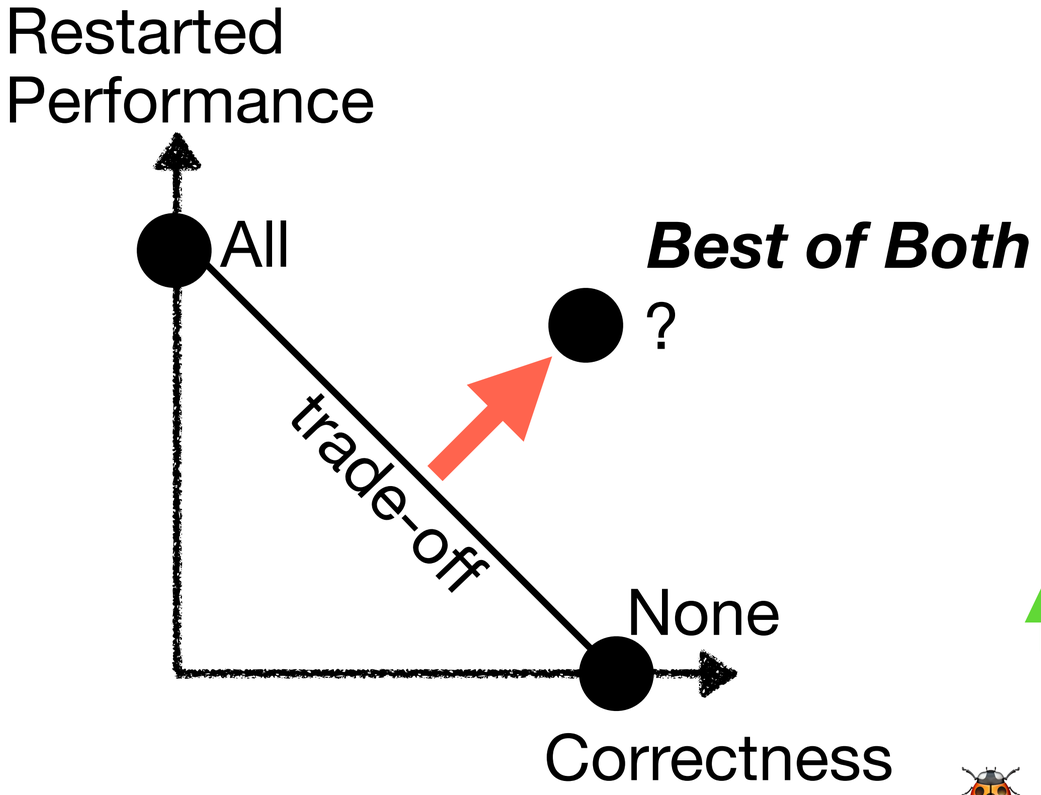
Dilemma

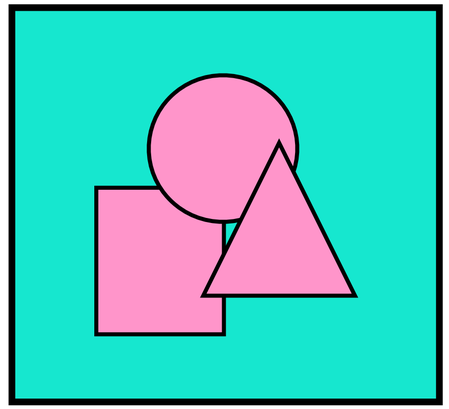


Preserve None

✅ Correct

🐌 **Poor availability during warmup**





Preserve All

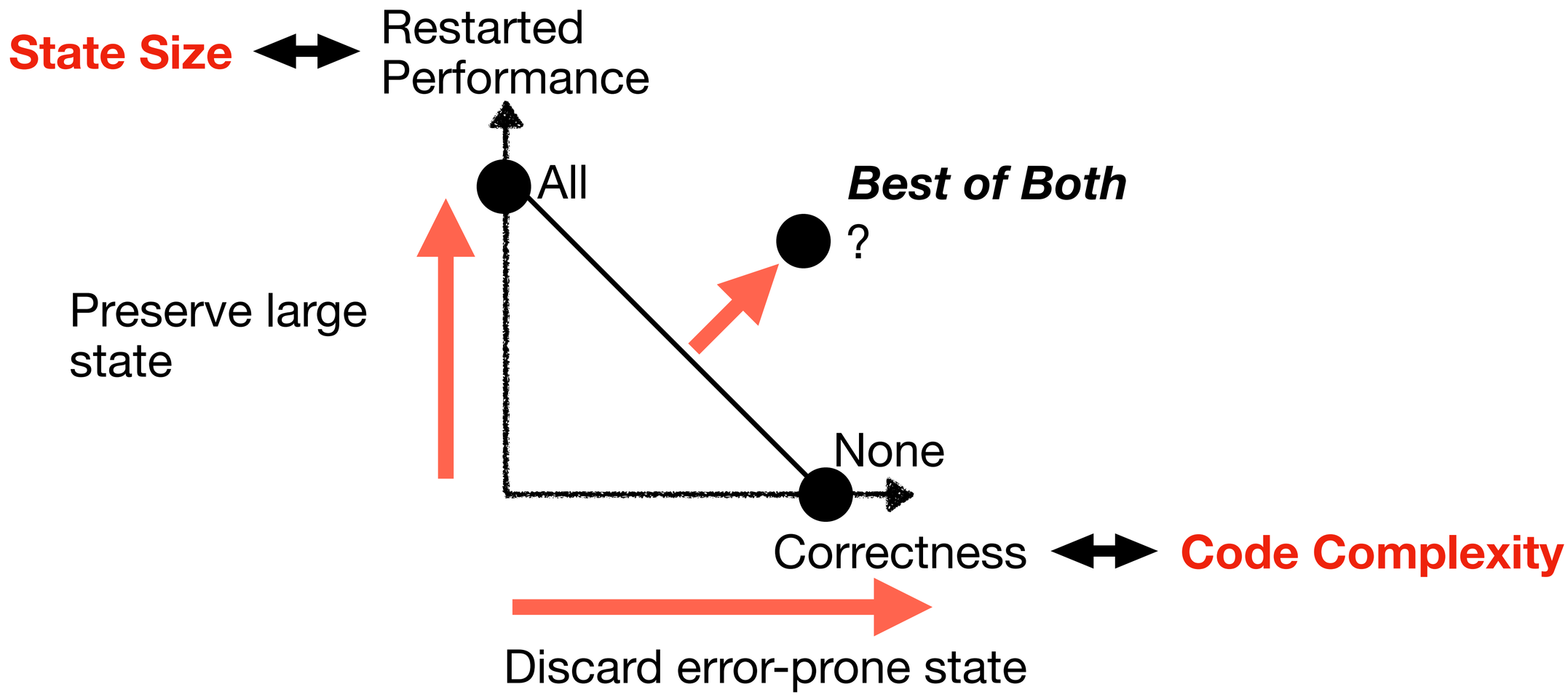
⬆️ Serves latest state

🐞 **Buggy state preserved**

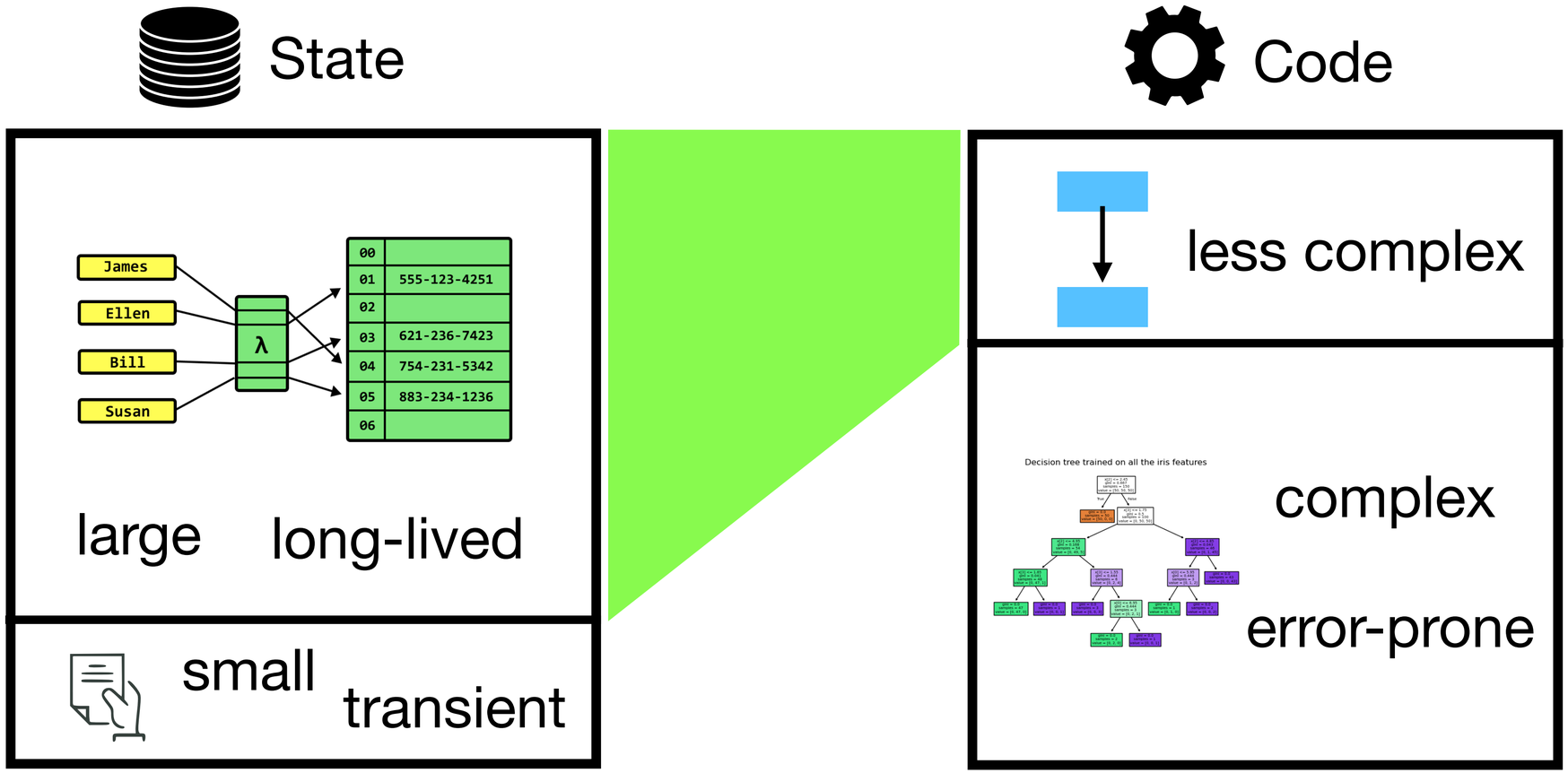
Challenges

- How to determine what state to preserve?
  - **Partial Process State Preservation**
- How to get high availability without compromising correctness?
  - **Optimistic Recovery**

What state to preserve?

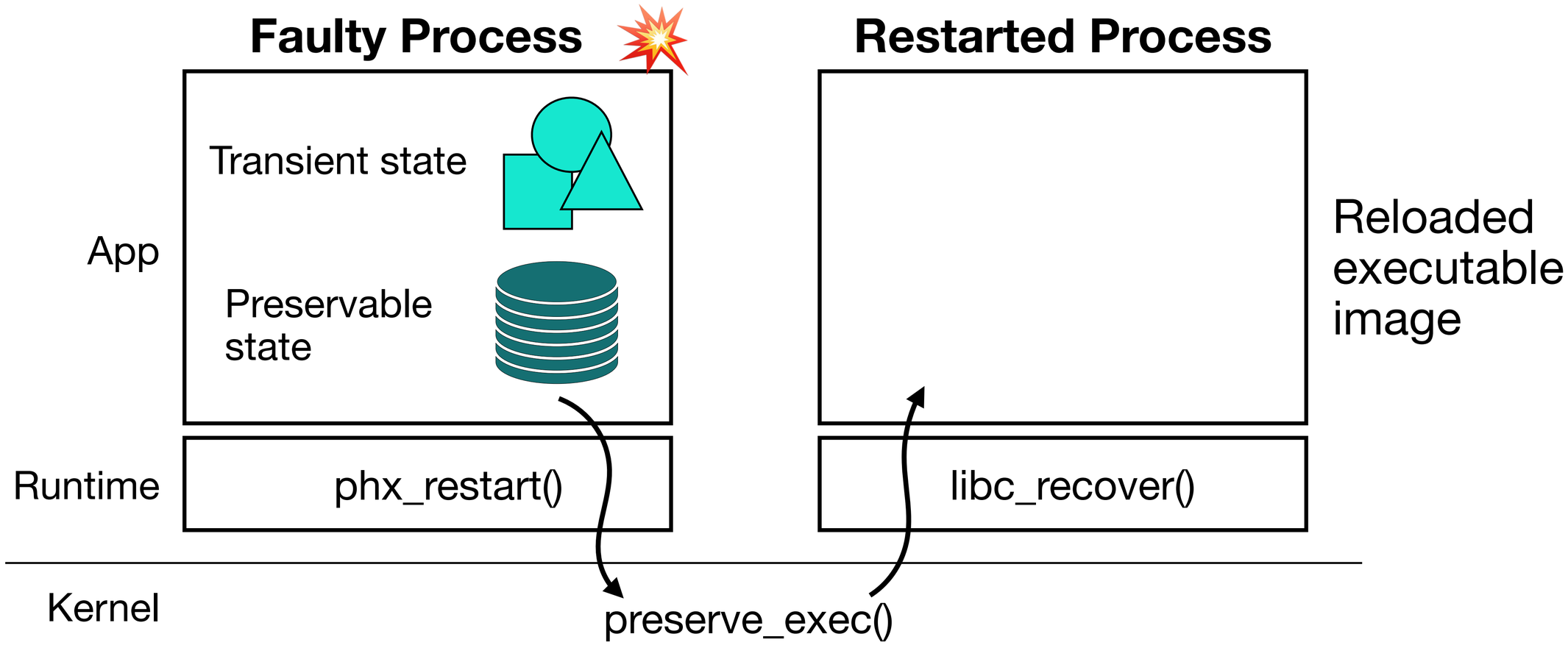


Insight: State Size & Complexity Imbalance

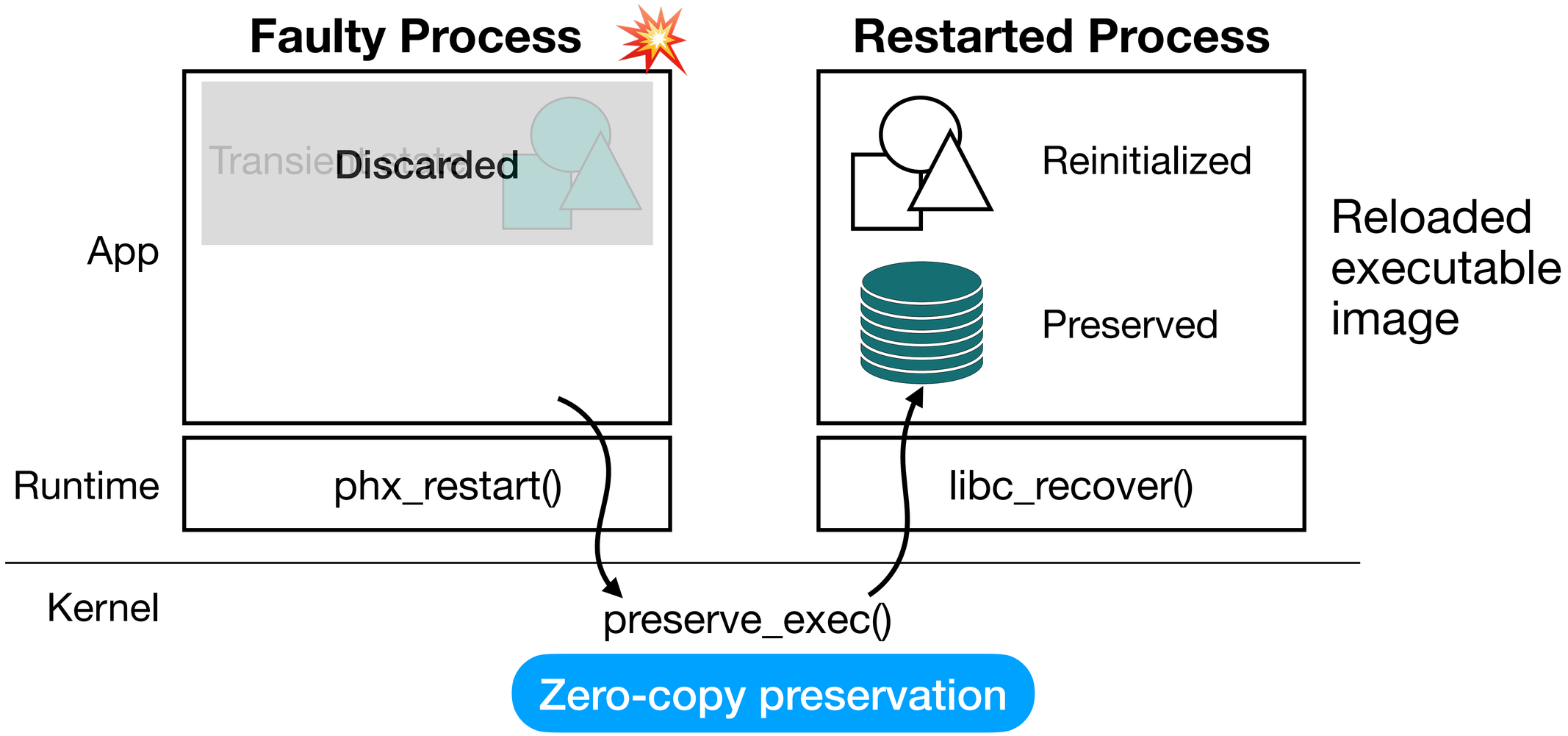


Opportunity: preserve large state with low chance of incorrectness

Phoenix: Partial State Preservation



Phoenix: Partial State Preservation



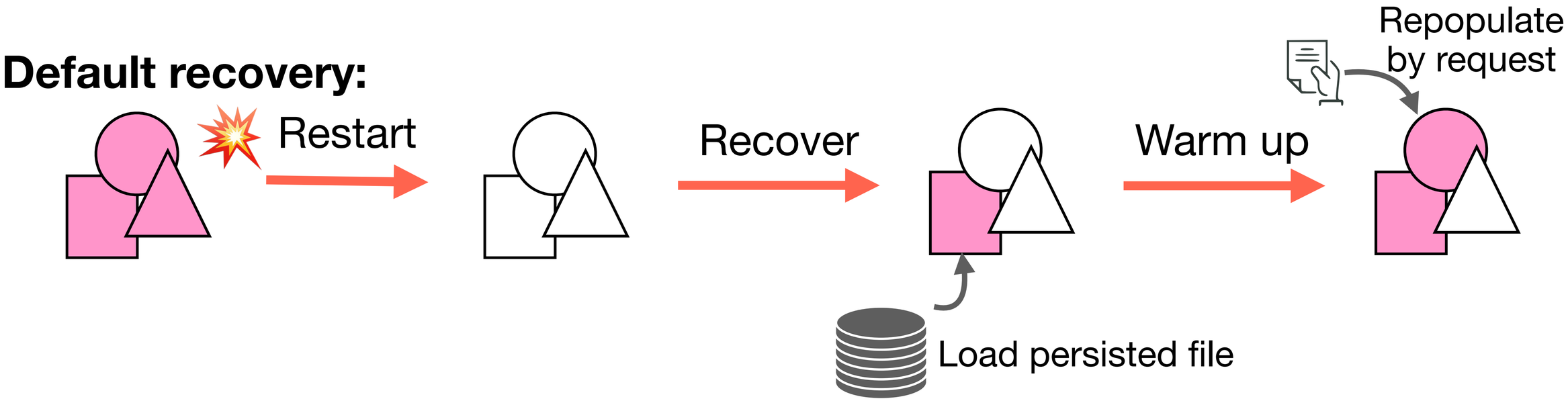
Phoenix

- Partial Process State Preservation

- **Optimistic Recovery**
  - **Unsafe Region**: efficient consistency check
  - **Cross-Check Validation**: long-term assurance

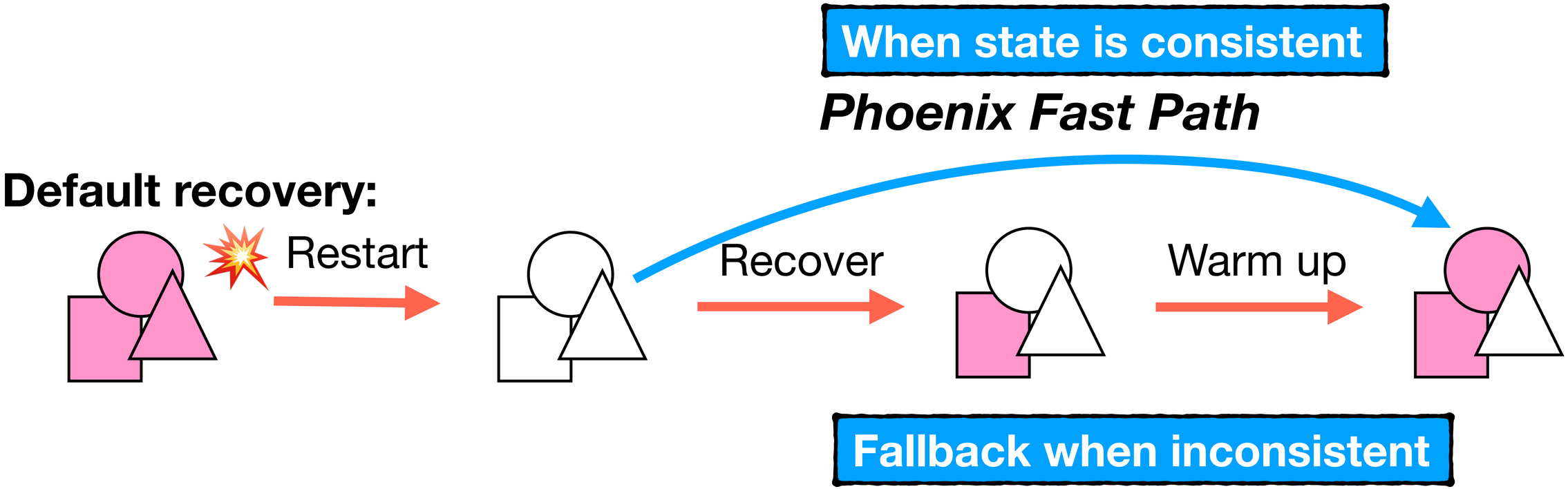
Optimistic Recovery

Designed to provide **high chance** of quick recovery



Optimistic Recovery

Designed to provide **high chance** of quick recovery



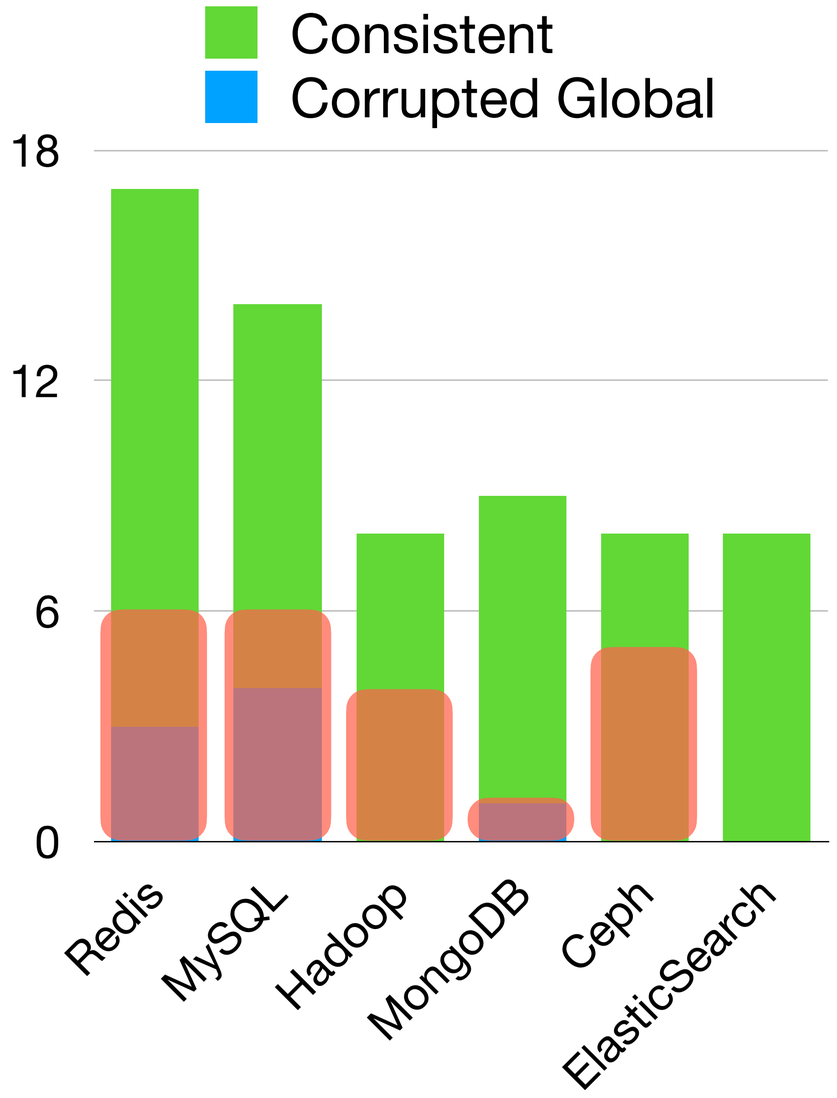
How to detect inconsistency?

Insight from Real-World Bug Study

Collect 64 case from 6 systems

- 56 transient-only
  - **Discarding transient** state already give correctness
- 8 corruption in long-lived state
  - High-correlation: all happen **during modification** operations

Solution: fallback on inconsistent write

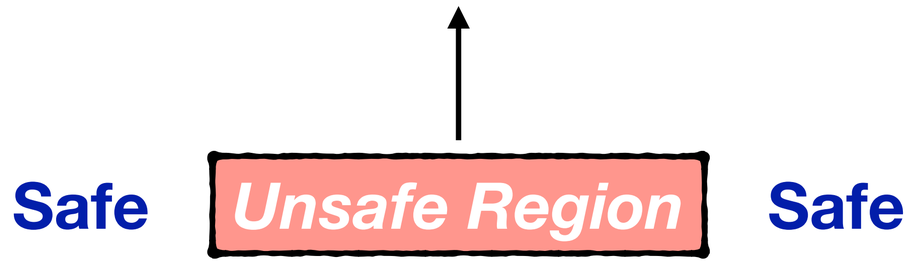


Unsafe Region

Write portion of preservable state
in one **transaction**

Example:

Network connect Request parsing	Data structure updates	Request reply Stats update



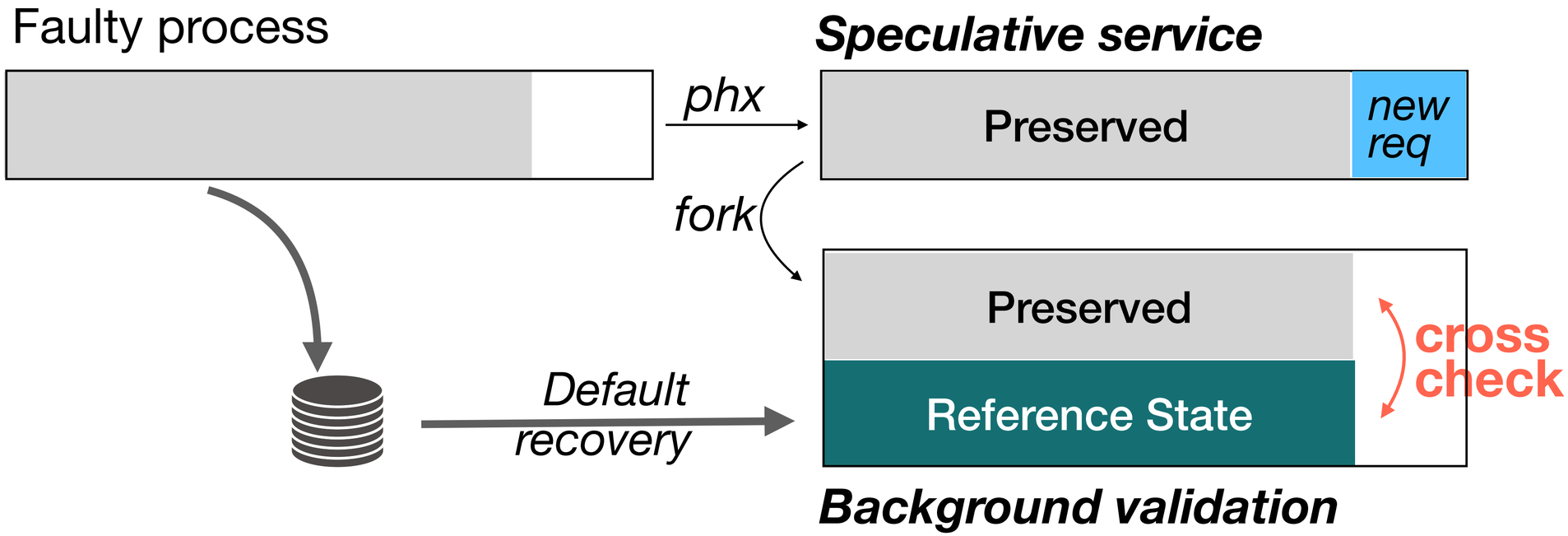
Either manual marking or using Phoenix
compiler auto-instrumentation

Unsafe Region

- Safe and effective indication of inconsistency in experiment

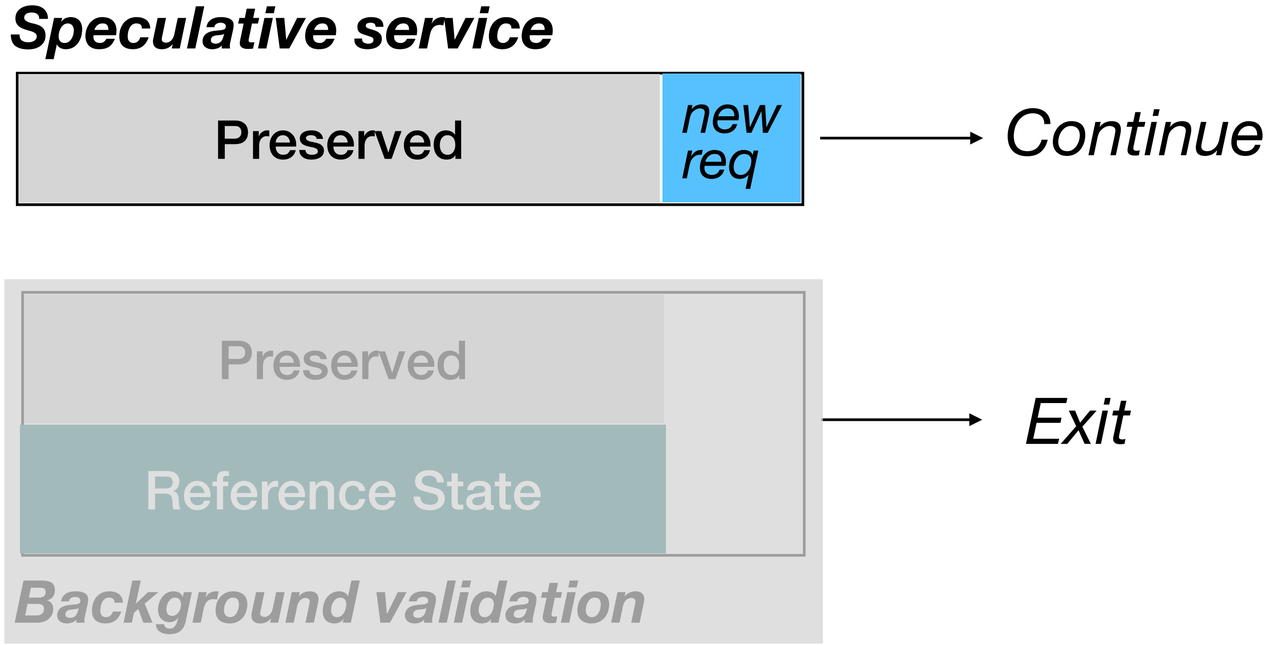
- However, indirect indication is not assurance
  - **Background Cross-Check Validation**

Cross-Check Validation



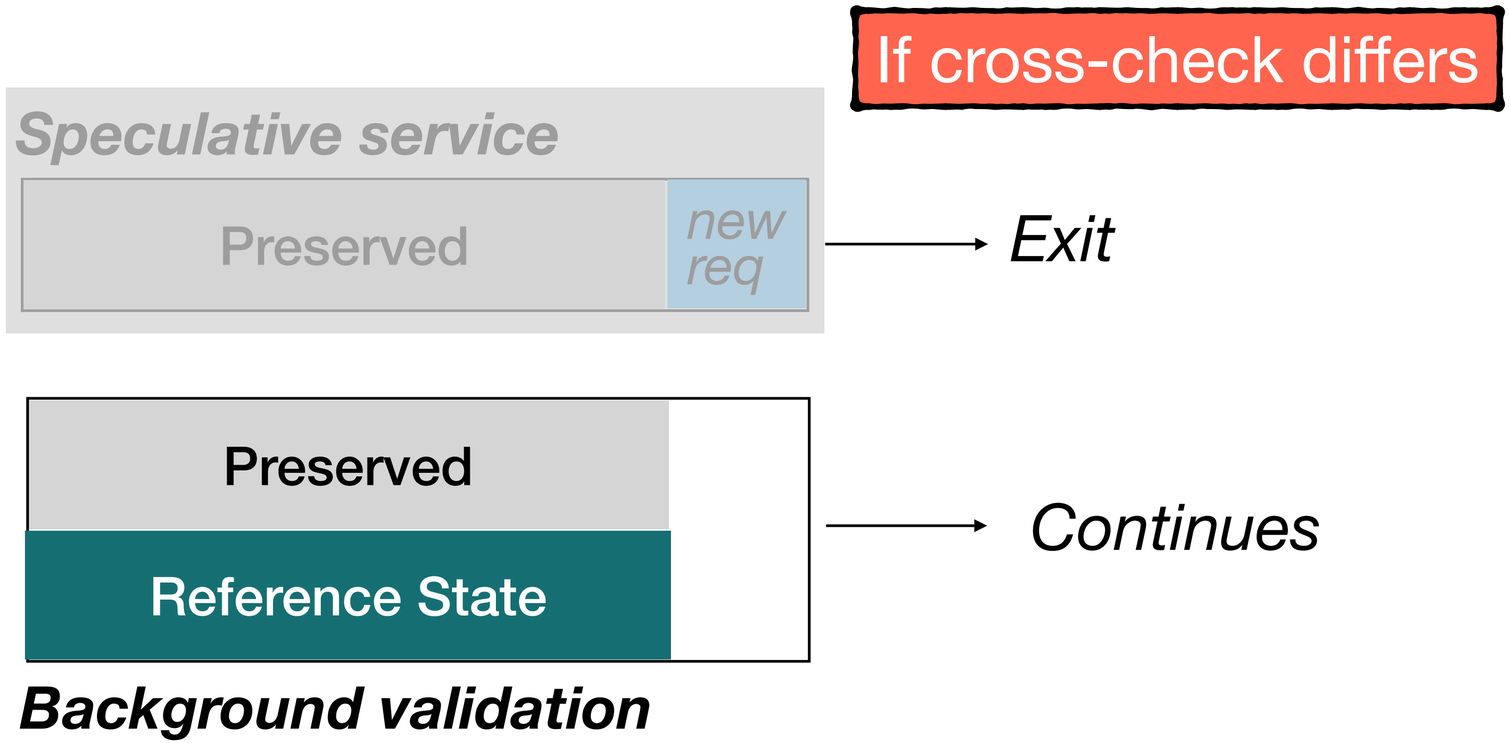
Cross-Check Validation

If cross-check matches

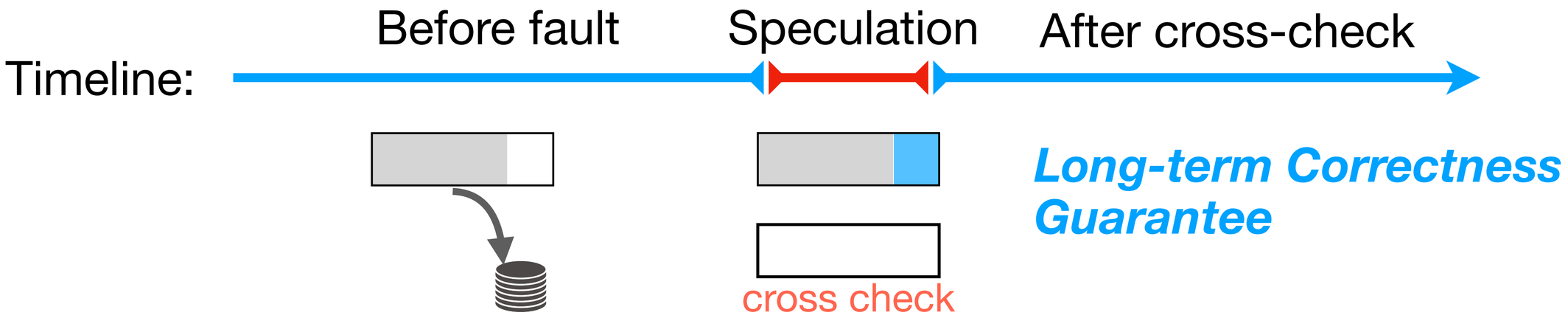


Availability gain from Phoenix restart!

Cross-Check Validation



Cross-Check Validation



Potential inconsistency bounded to cross-check window

Evaluation

- Implemented Phoenix in Linux kernel 5.15
- Ported 6 applications *without knowing the bugs*
- We try to answer questions:
  - **Effectiveness**: How much availability improvement does Phoenix achieve?
  - **Correctness**: Does Phoenix recover correctly?
  - **Effort**: What effort and overhead are required to employ Phoenix?

Ported Applications

App	Preserved States	Codebase LoC	Changes
Redis	In-mem KV hash table	140,996	348 (0.25%)
LevelDB	Skiplist memory tables	21,192	312 (1.50%)
Varnish	Web page cache objects	109,564	281 (0.26%)
Squid	Web page cache objects	186,219	190 (0.10%)
XGBoost	Gradients and model	38,906	158 (0.41%)
VPIC	Particles and physical fields	44,773	272 (0.61%)

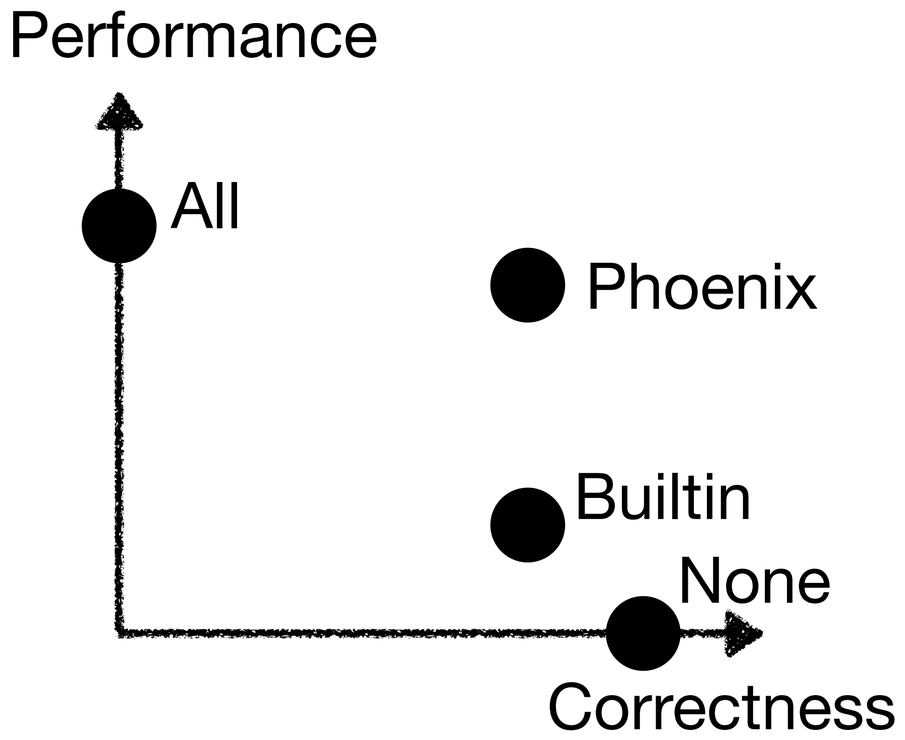
Only moderate usage efforts

Availability

Reproduce 17 real-world bugs from various categories.

Compare between 4 setups

- **None**: Original application without persistence
- **All**: Whole process checkpointing (CRIU)
- **Builtin**: Application **default recovery** method
- **Phoenix:** Phoenix with Builtin as fallback



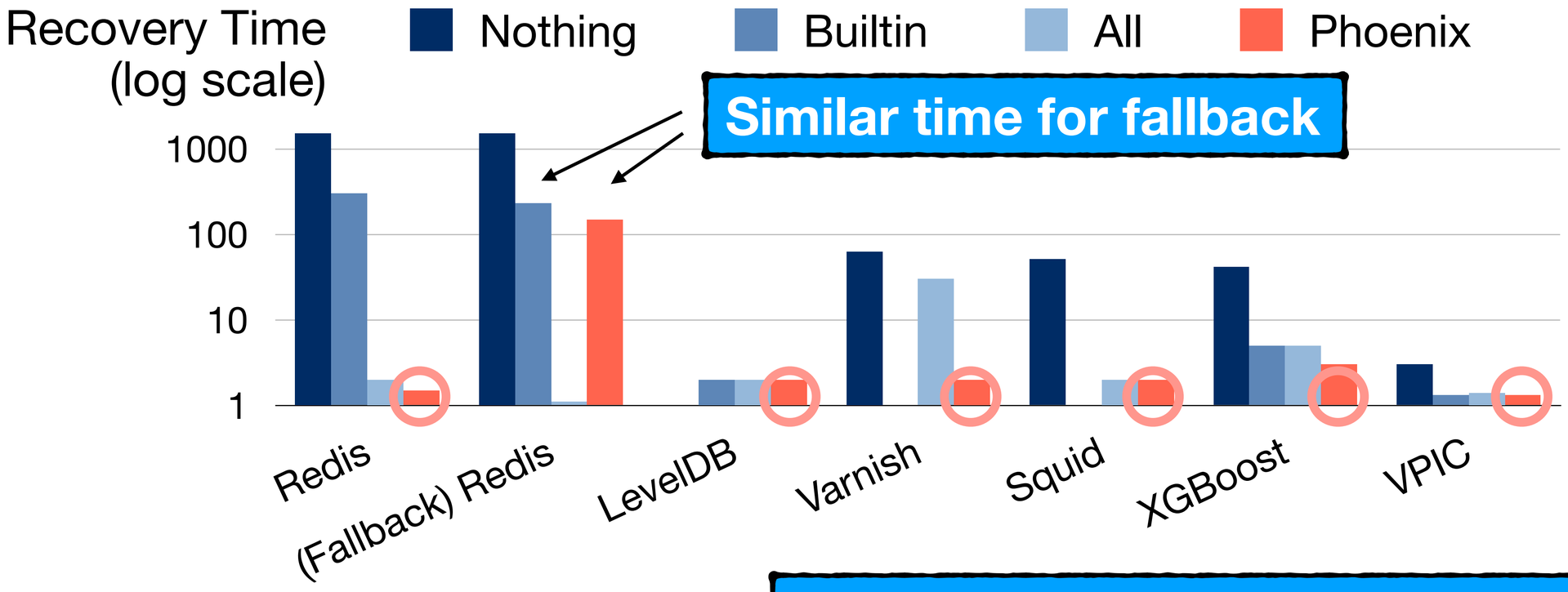
16 recovered by Phoenix

1 fallback to default recovery

Availability

Time to Recover to 90% of Effective Availability

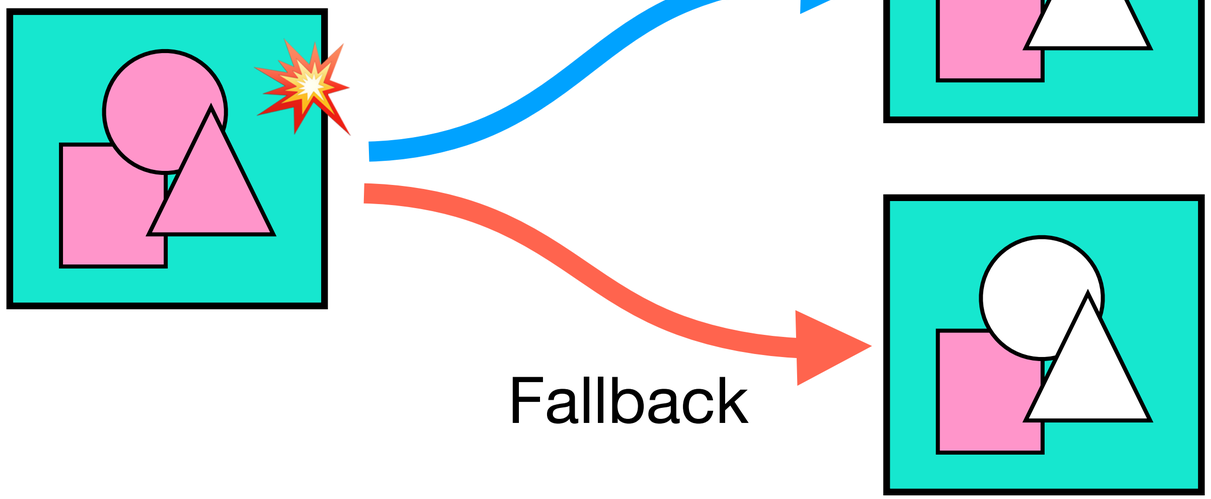
Hit rate for cache, throughput for others



Quickly restore performance to pre-restart

Correctness: Large Scale Injection

Only Unsafe Region



Total: 4,000 runs

3,287 **successful Phoenix** recovery

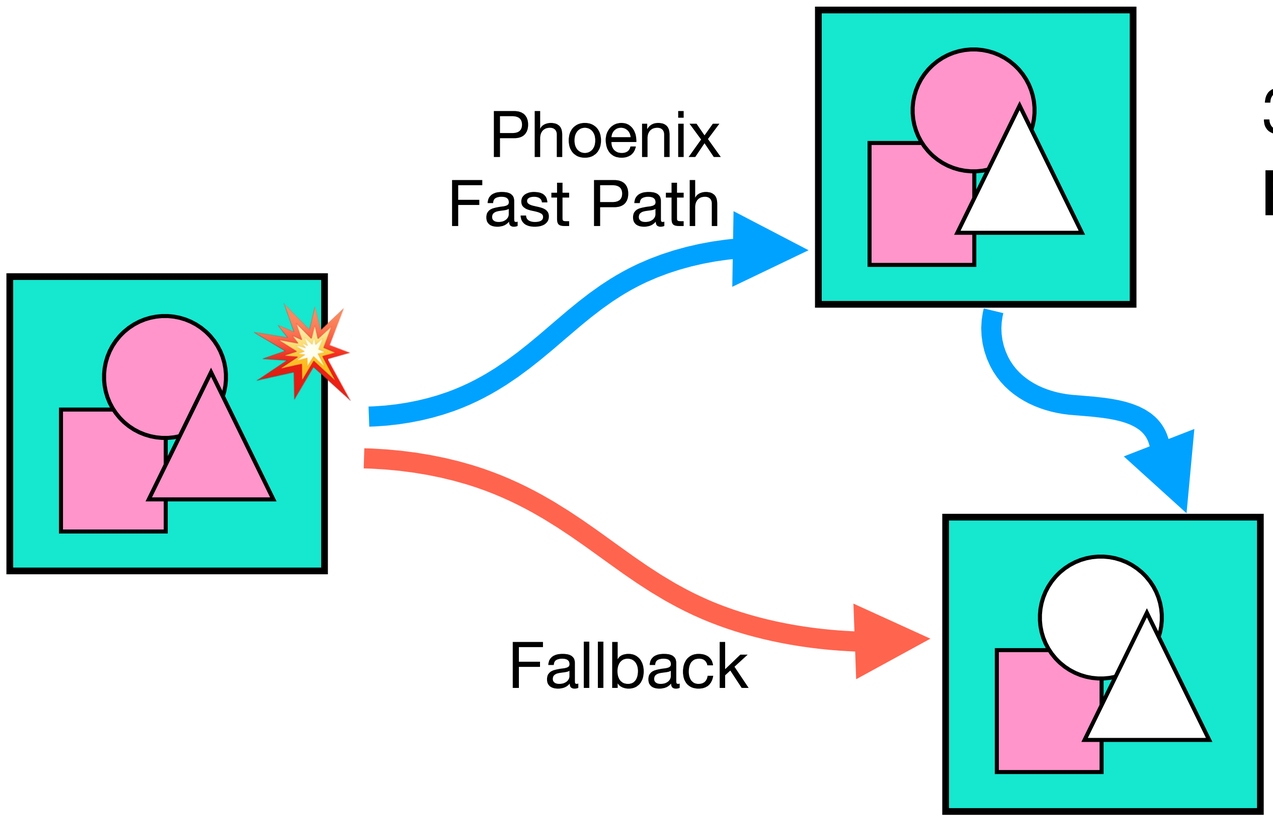
82% fast path taken

662 **unsafe region caught**

51 **faulty restart caught**

Correctness: Large Scale Injection

Cross-Check Mode



Total: 400 runs

325 successful Phoenix recovery
passing cross-check

5 **cross-check early fallbacks**

65 unsafe region caught

5 faulty restart caught

Conclusion

- **Partial state preservation**
  - Effectively preserves most state while discarding most faults
- **Optimistic recovery**
  - High availability with long-term correctness guarantee
- New OS-level mechanism ***Phoenix***
  - Significant availability and performance improvements

https://github.com/OrderLab/phoenix