Understanding and Dealing with Hard Faults in Persistent Memory Systems

Brian Choi, Randal Burns, Ryan Huang

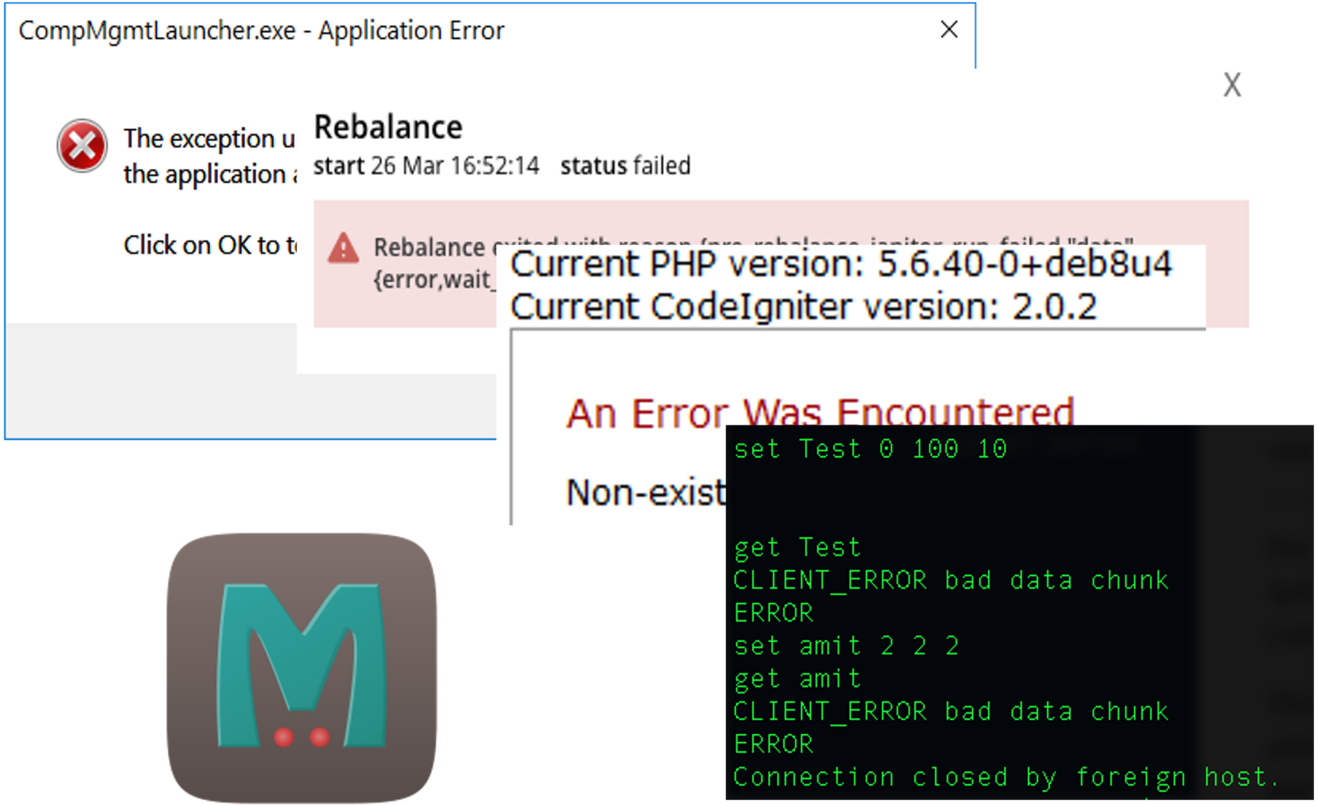
EuroSys '21

JOHNS HOPKINS UNIVERSITY

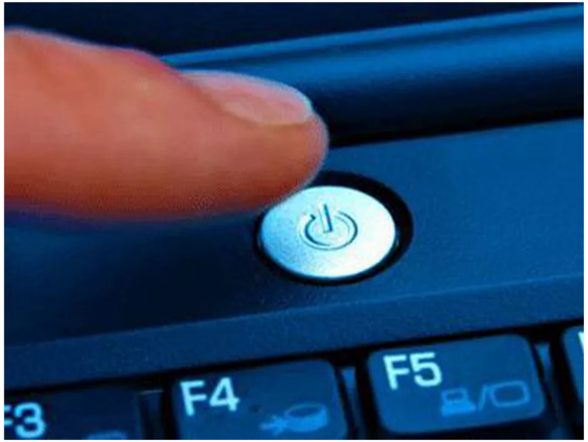
Summary

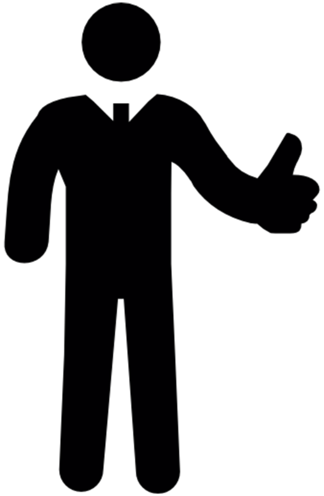
- We introduce a new class of faults in PM systems: **Soft-to-Hard Faults**
- In-depth study: 28 **real-world examples** in 7 PM systems
- Solution: Arthas, tool to **recover PM systems** that suffer from Soft-to-Hard Faults to a correct working state
  - minimal data loss

Errors Frequently Occur in Systems



First Thing to Try? Restart!





Soft and Hard Faults

- “Soft” **faults** -> Restart **fixes** the system
  - Bad volatile states: go away upon restart
  - Many production failures are soft/transient (Gray, 1985)
- “Hard” **faults** -> Faults are **recurring** even after restart
  - Bad states written to storage (ie. disk) permanently exist after restart

Soft and Hard Faults

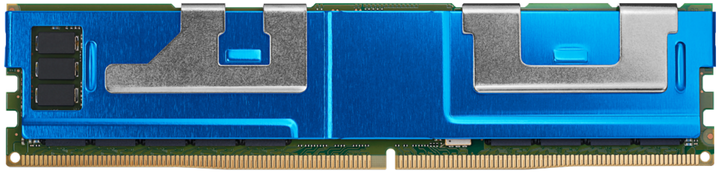
- “Soft” **faults** -> Restart **fixes** the system
  - Bad volatile states: go away upon restart
  - Many production failures are soft/transient (Gray, 1985)
- “Hard” **faults** -> Faults are **recurring** even after restart
  - Bad states written to storage (ie. disk) permanently exist after restart

- **Hard faults are particularly prominent in persistent memory**
  - Introduces new hard faults in your system
  - **Restart not effective** anymore!!

Persistent Memory = More Options

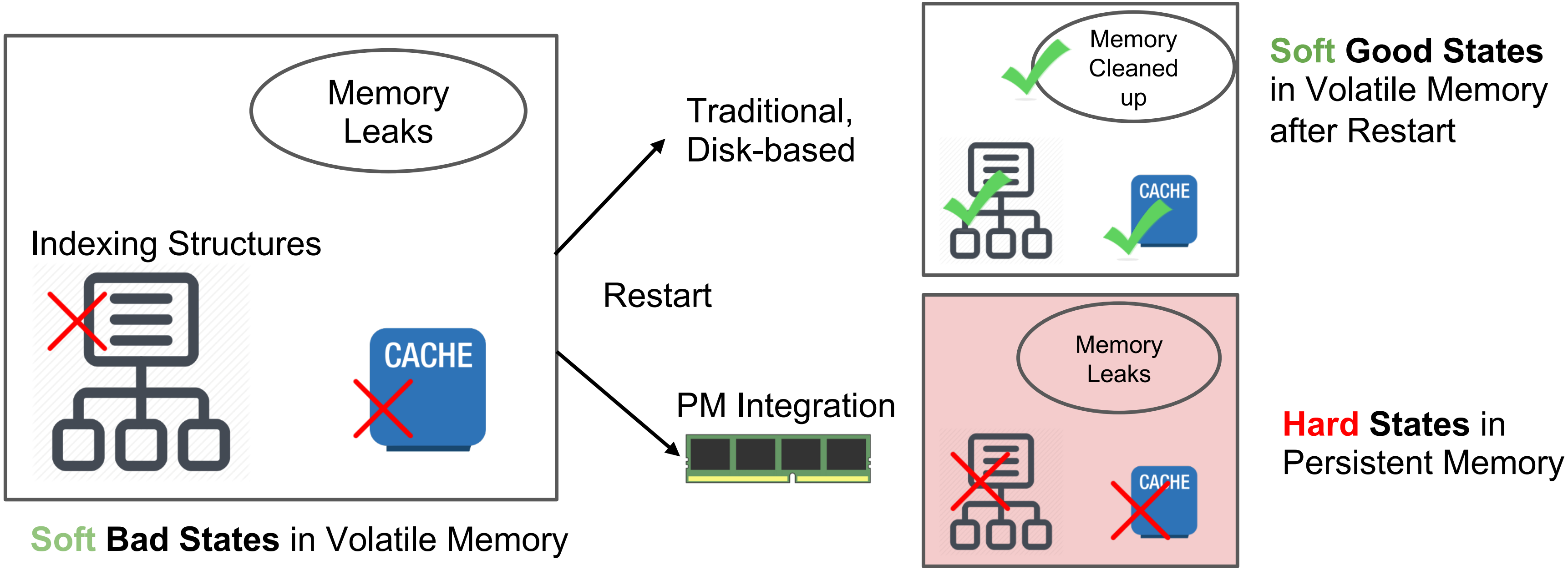
- Fast new storage technology called Persistent Memory (PM)
  - Latency comparable to DRAM
  - Persistently store data
- Developers now have more options to persist data
  - PM much faster than previous storage
  - More states become persistent (ie. cache)

	Sequential Read	Random Read	Write
DRAM	81.4 ns	83.2 ns	157.7 ns
PMEM	179.0 ns	317.6 ns	160.4 ns

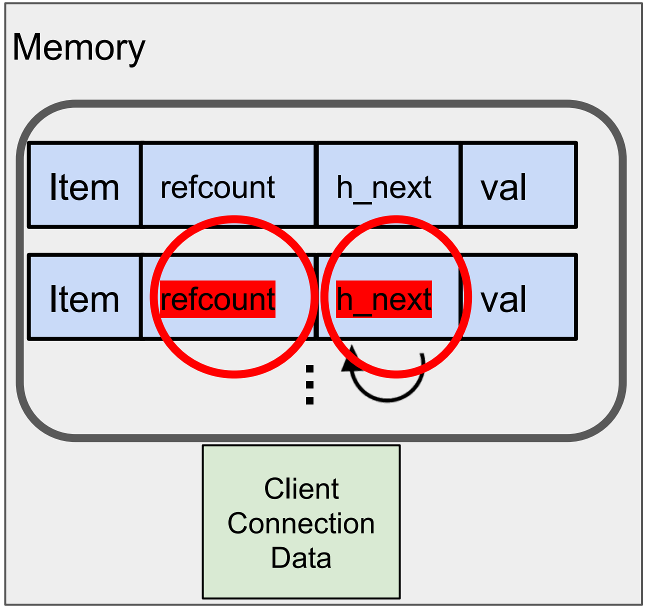


Soft-to-Hard Faults

- **PM persists many states**
  - "Soft" states move to PM more frequently, become "hard" states
  - Restart can't fix any issues with these "Soft-to-Hard" states



A Real Soft-to-Hard Fault

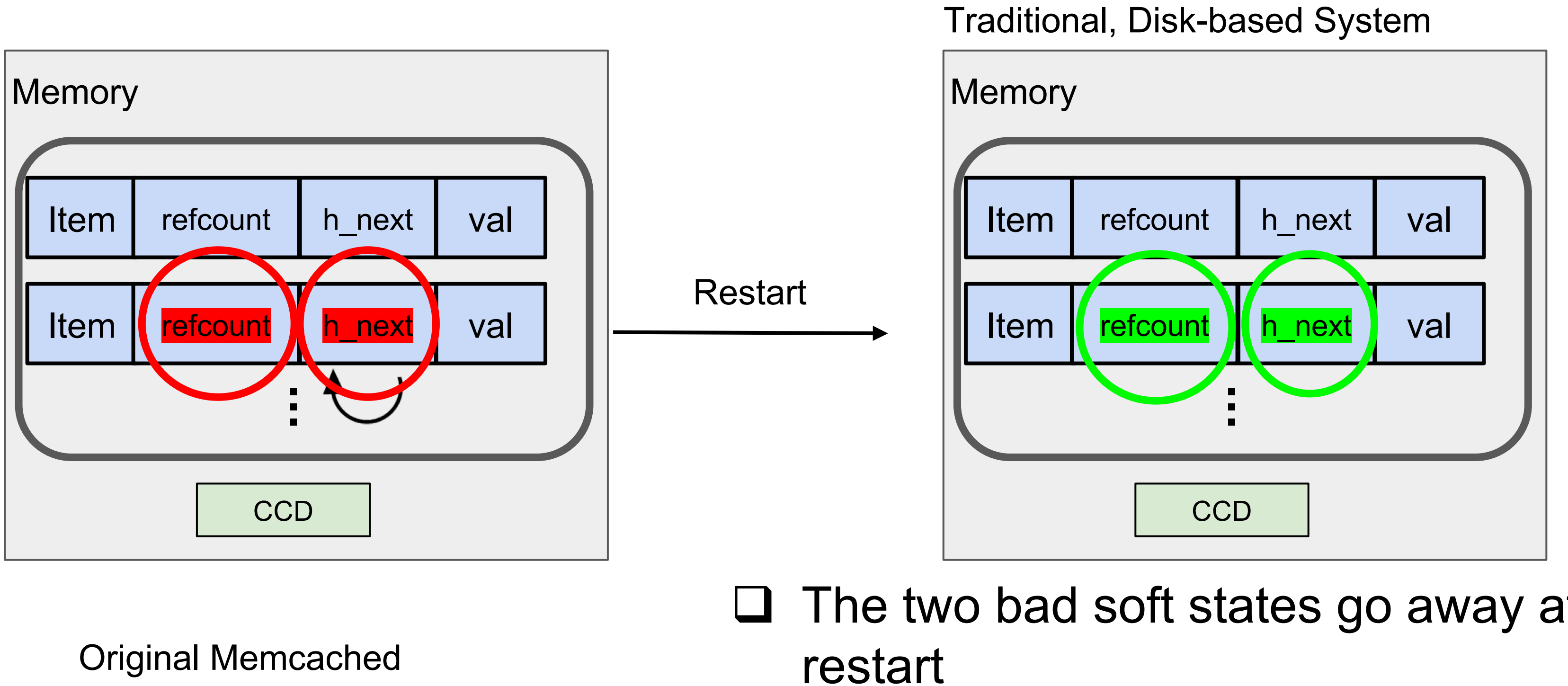


Original Memcached

- Soft Fault in Memcached
- Items and client connection data all in volatile memory
- Two bad states: refcount and h_next corrupted

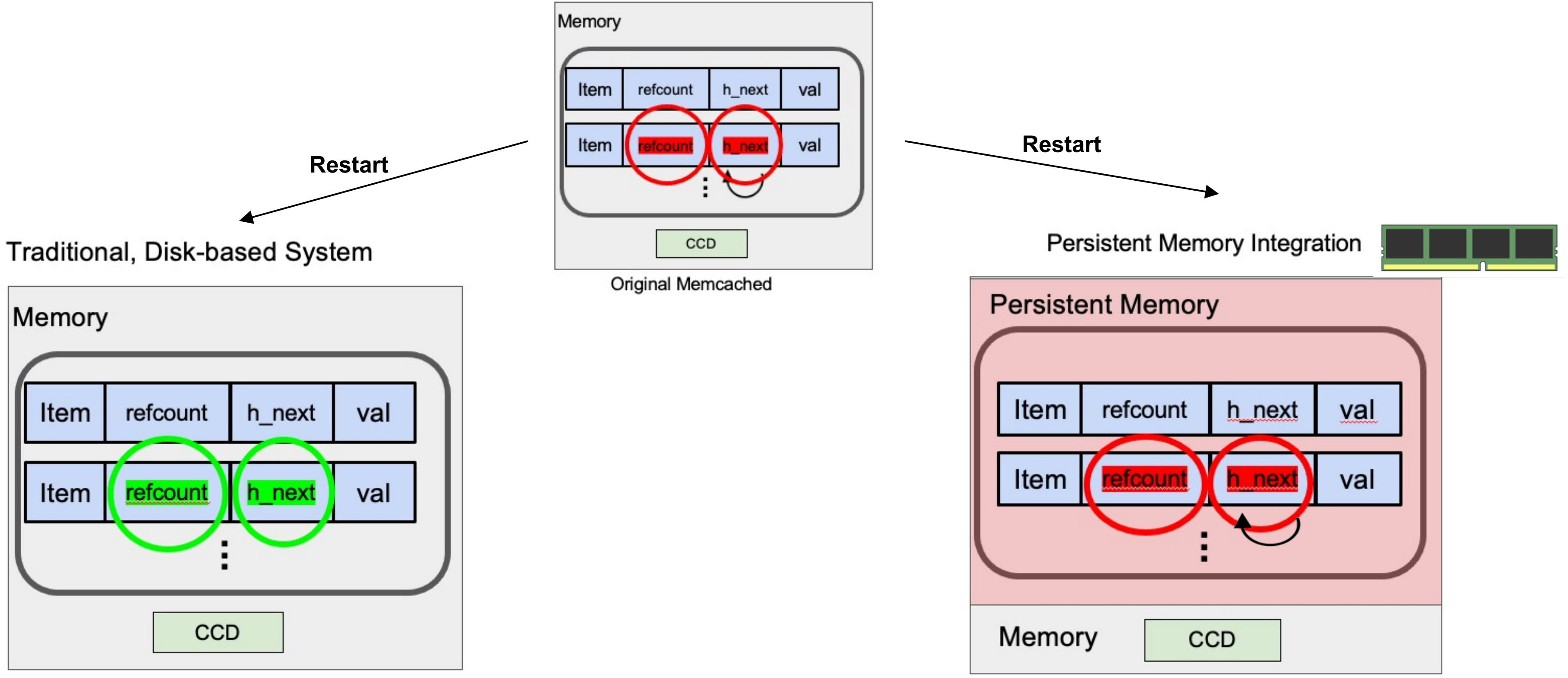
https://github.com/memcached/memcached/issues/271

A Real Soft-to-Hard Fault



- The two bad soft states go away after restart

A Real Soft-to-Hard Fault



Our Contributions

- Hard Fault Study where we examine 28 **real-world bugs in PM systems** and analyze the bugs.

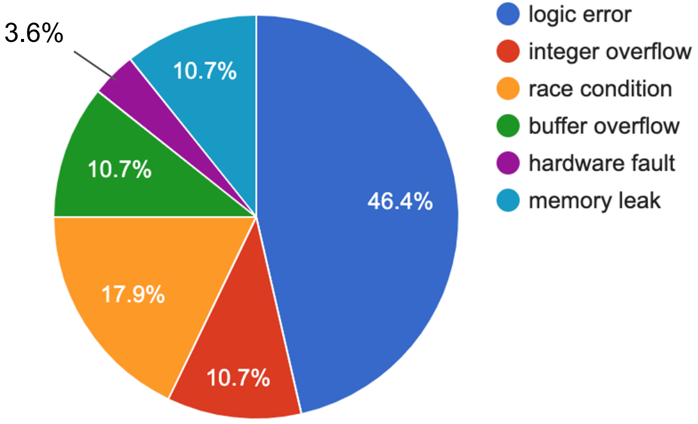
- **Arthas: tool to recover PM systems that suffer from Soft-to-Hard Faults**
  - Dependency-based rollback - minimize data loss
  - Static Analysis - to aid dependency formulation

Hard Fault Study

- We found 28 bugs from 7 PM systems that demonstrate the soft-to-hard fault problem.

System	Type	Cases	Description
Redis	Port	11	in-memory key-value store
Memcached	Port	9	in-memory key-value store
Recipe	Port	2	ported concurrent indexes to PM
LevelHash	New	2	PM hashing index scheme
PMEMKV	New	2	PM key-value store
Dash	New	1	scalable PM hashtable
CCEH	New	1	PM cacheline-conscious indexing structure

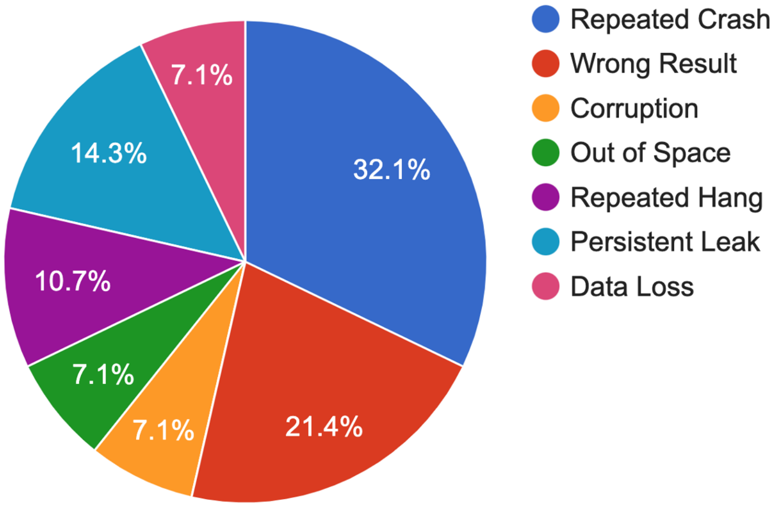
Finding 1: Root Causes Are Diverse



Root Cause Distribution

- ❑ PM Hard Faults can be caused by **many different types** of root causes
- ❑ Logic Errors are most prevalent (46%), but other types of root causes are relatively **evenly distributed**

Finding 2: Consequences Are Severe



Consequence Distribution

- ❑ PM Hard Faults cause **severe problems**, such as **repeated crashes** and **wrong results**, not just minor issues

Finding 3: Over Half Propagate Bad State

Type I: PM variable has a bad value that **directly causes** the problem
Type II: PM variable has a bad value that **propagates bad state** and **indirectly causes the problem**

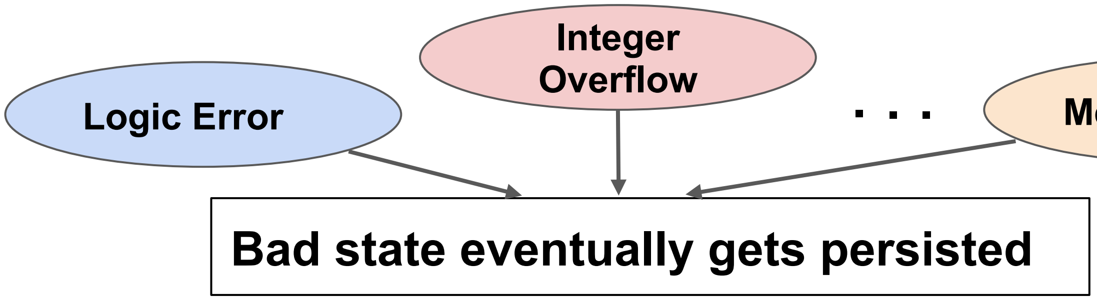
Type	Distribution
Type I (direct)	18%
Type II (indirect propagation)	68%
Type III (miscellaneous)	14%

Type Distribution of Cases

- ❑ **68% of the bugs propagate bad state** among volatile and persistent variables **throughout the system**

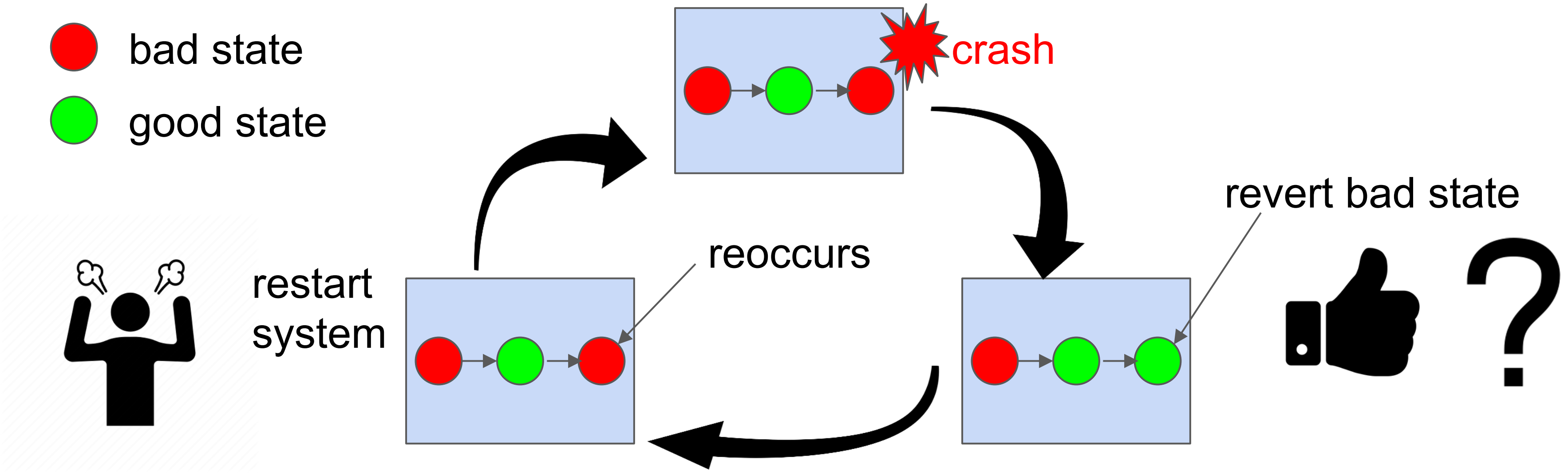
PM Hard Faults Lead to Bad State

- **PM hard faults are a diverse set of bugs**
  - Challenging to statically find all bugs with one solution
- **Insight: At runtime, PM hard faults eventually cause bad state to be persisted**
  - Revert bad states to good states



We Must Revert the Root Cause

- **68% of the bugs propagate bad state**
  - Design Principle: Even when **one bad PM state is rolled back**, the **PM system could still quickly hit the same failure** if the **root cause** of the bad state is not reverted.



How to Revert Bad States?

Goal of effective mitigation: hard fault disappears and doesn't reappear again

- **Rollback bad states** to a **previous, older version** that is a **good state**
- Checkpoint data: we keep multiple versions of state
- Revert all propagated bad states including the root cause

Standard Checkpoint/Rollback

time:t

```
{
   {key1: value1},
   {key2: value2}
}
```

SNAPSHOT

Standard Checkpoint/Rollback

time:t

```
{
  {key1: value1},
  {key2: value2}
}
```

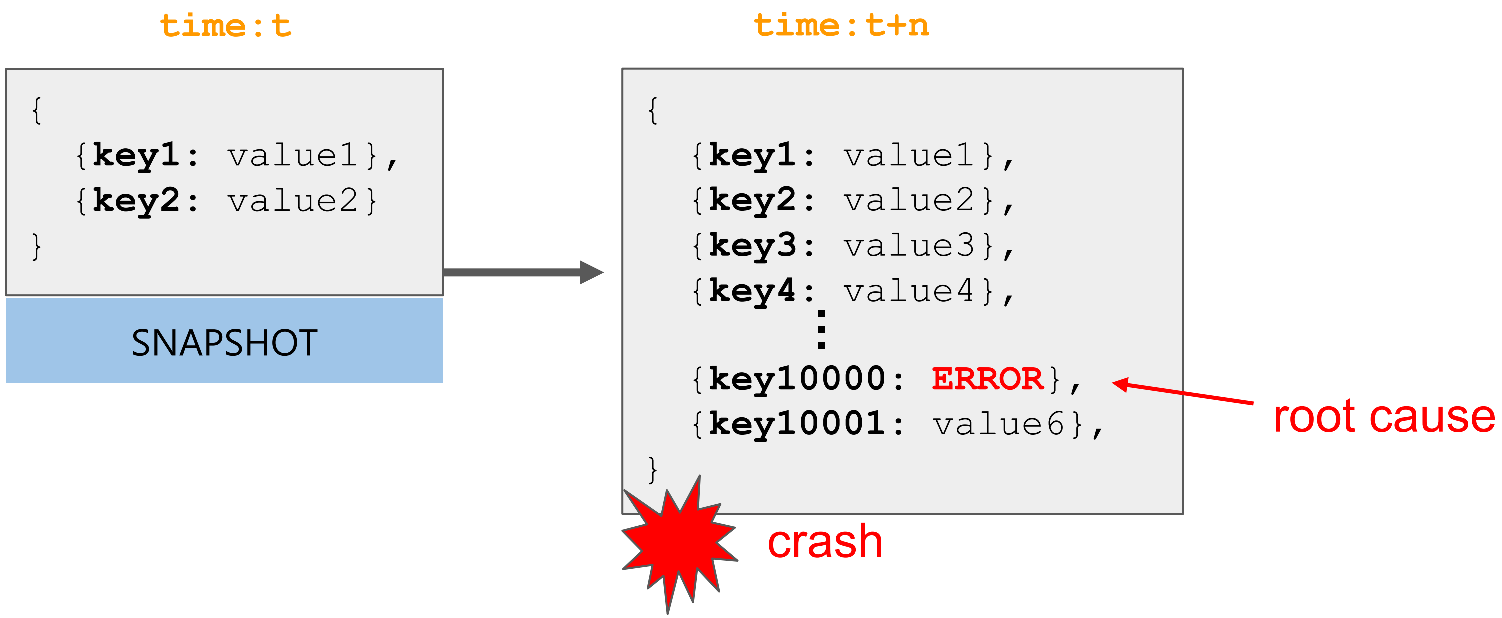
SNAPSHOT

time:t+n

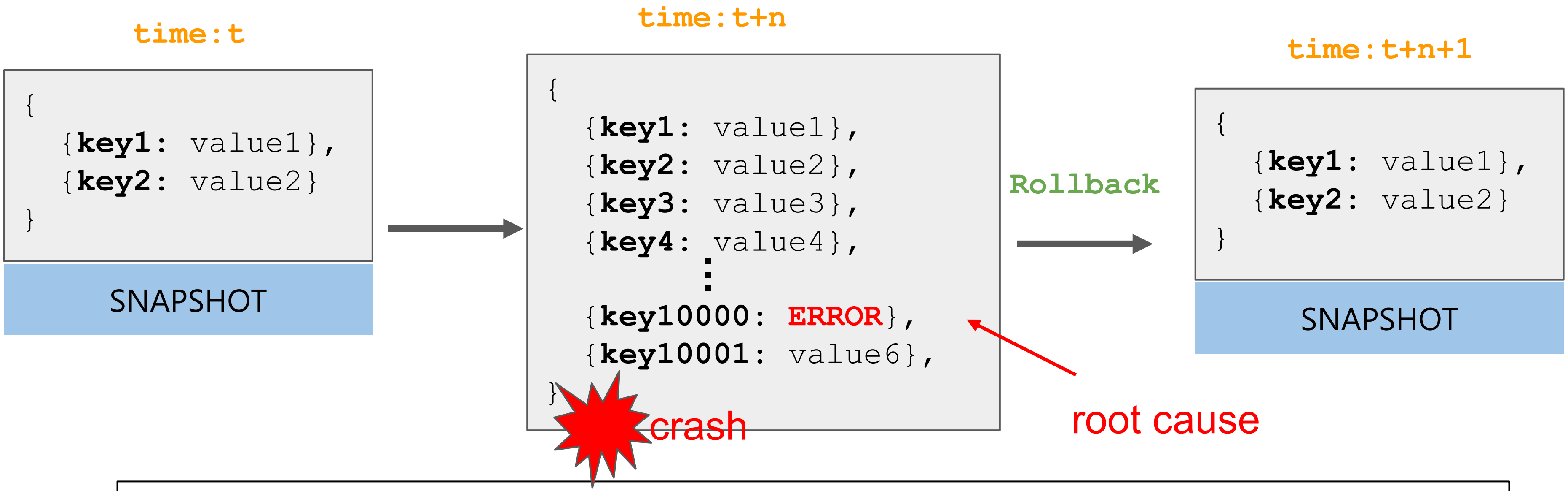
```
{
  {key1: value1},
  {key2: value2},
  {key3: value3},
  {key4: value4},
         ⋮
  {key10000: ERROR},
  {key10001: value6},
}
```

crash

Standard Checkpoint/Rollback

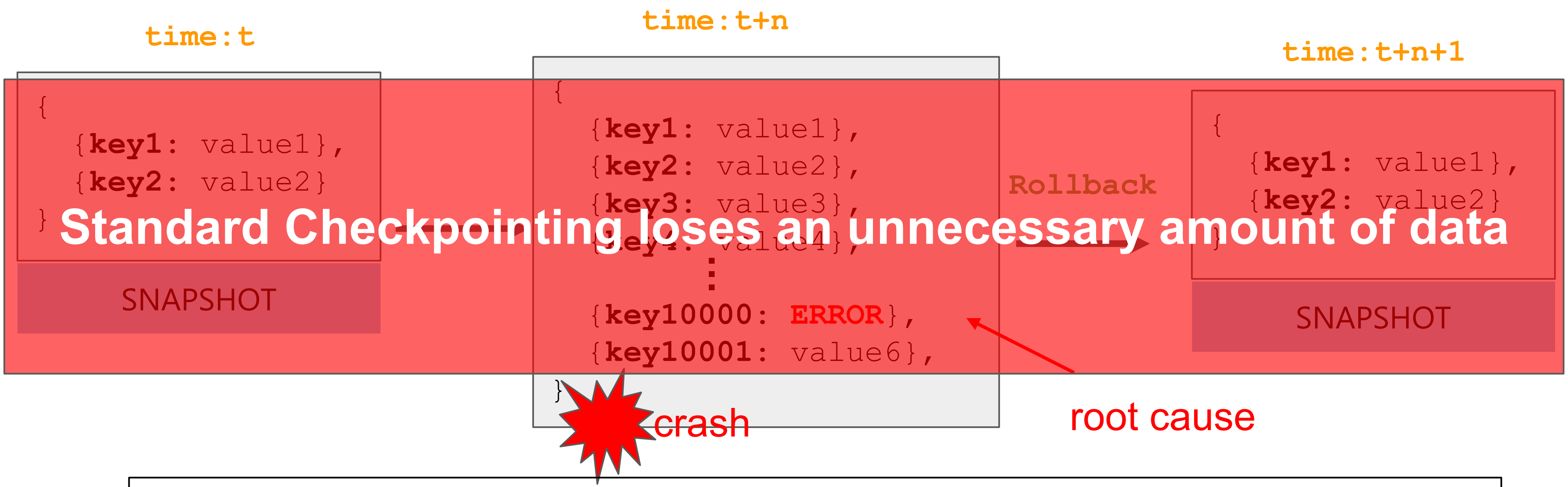


Standard Checkpoint/Rollback



Loses keys 3 to 10,001 when only key 10,000 was a bad state

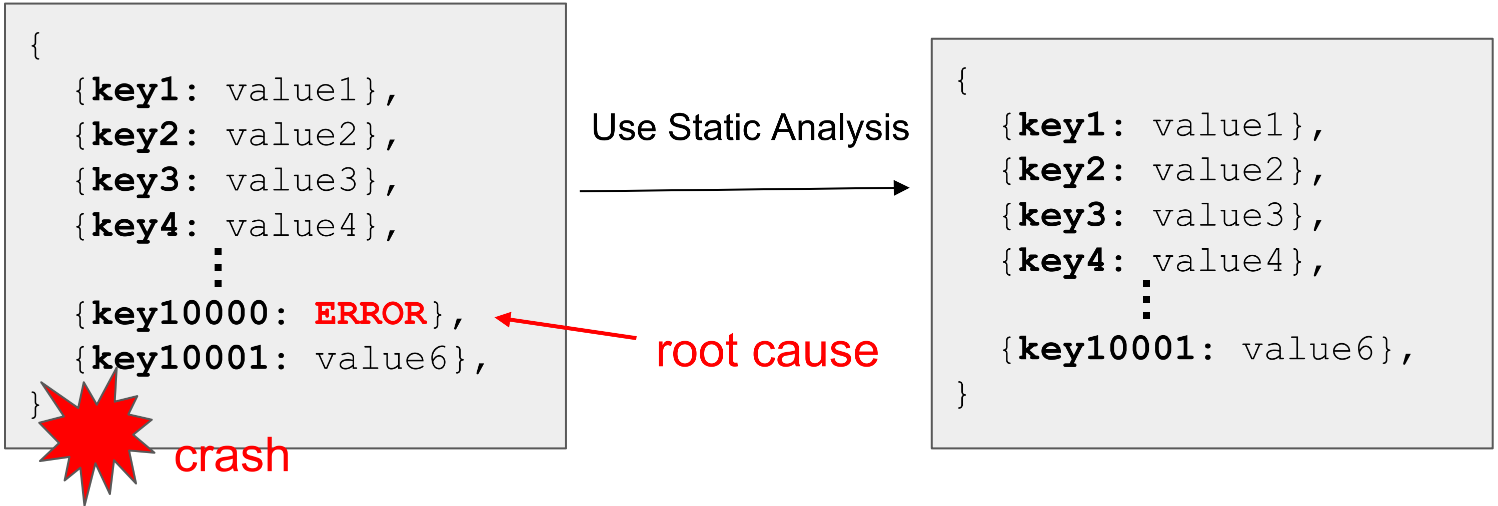
Standard Checkpoint/Rollback



Loses keys 3 to 10,001 when only key 10,000 was a bad state

Design Goal - Minimal Data Loss

- **Standard Checkpointing** approaches **lose too much data**
- **Design Principle:** Use **static analysis** on PM system to find dependencies of the **bad PM variables** and **revert only the bad PM states** using these dependencies



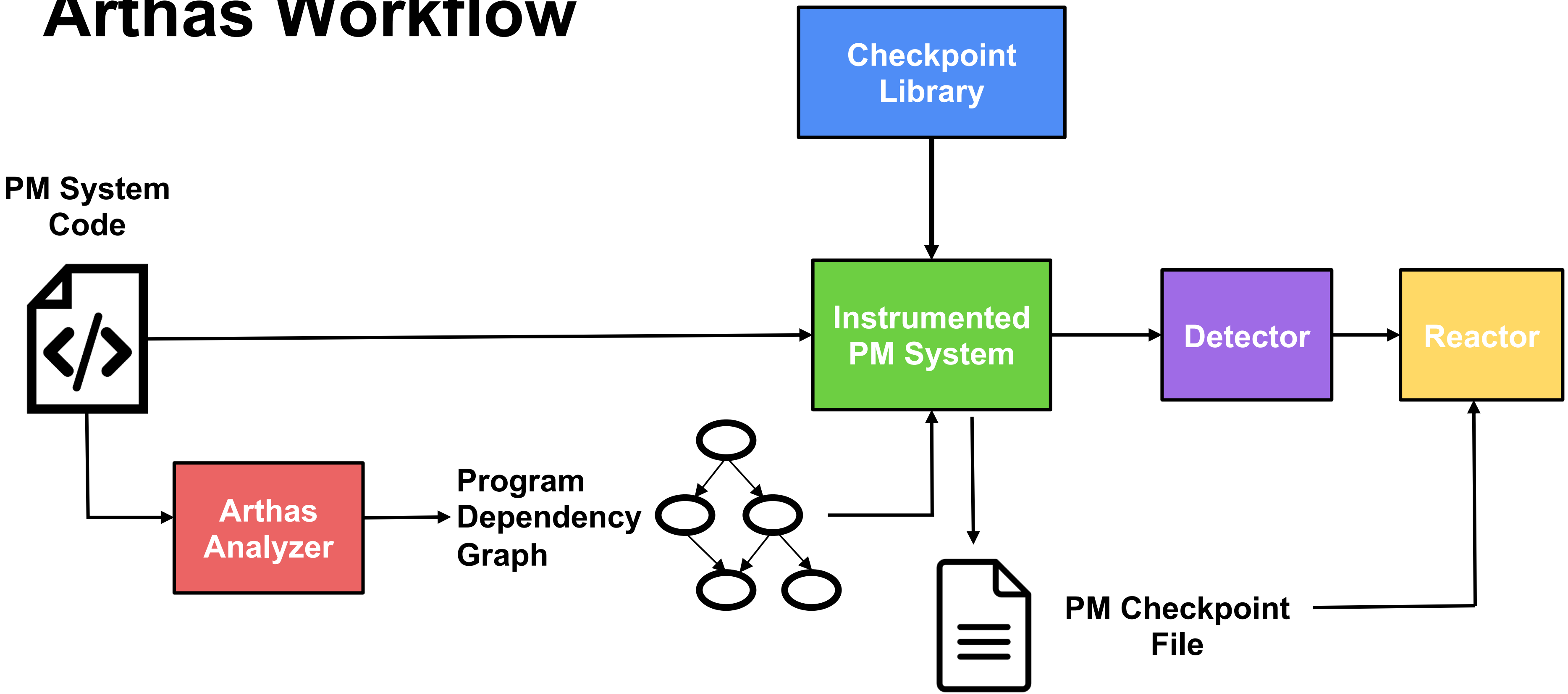
Arthas: Overview

- Arthas: tool that recovers PM systems from PM hard faults

- **Techniques**
  - Checkpoint old versions
  - Static analysis and dynamic tracing
  - Dependency-based rollback

- **Goals**
  - Recover PM system quickly
  - Minimize data loss
  - Small runtime overhead

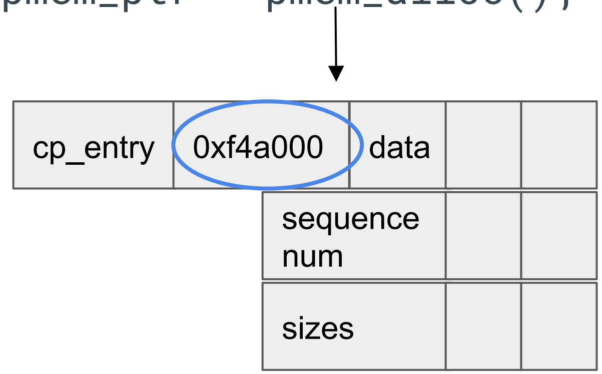
Arthas Workflow



Checkpoint Library

- ❑ **Checkpoint multiple versions of PM state to later revert**
- ❑ Implementation: Intercepts PM framework API calls
- ❑ Global sequence numbers assigned to each PM update
  - ❑ ensure order when reverting

```
pmem_ptr = pmem_alloc();
```

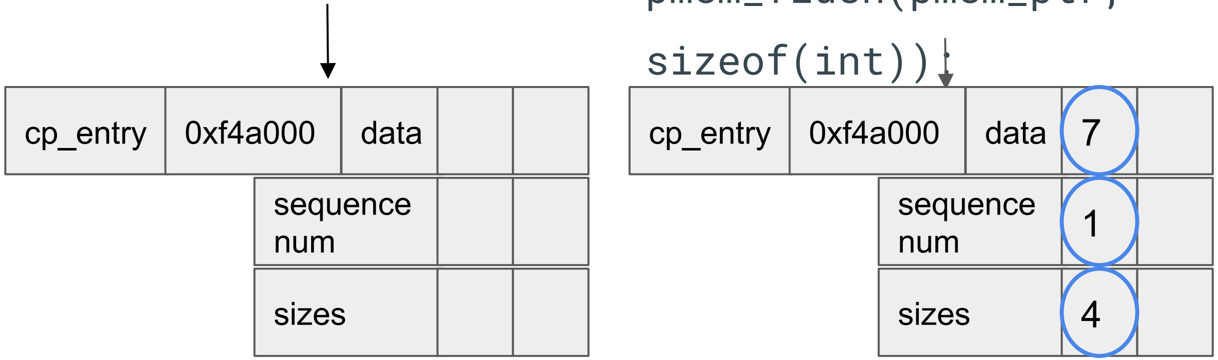


Checkpoint Library

- **Checkpoint multiple versions of PM state to later revert**
- Implementation: Intercepts PM framework API calls
- Global sequence numbers assigned to each PM update
  - ensure order when reverting

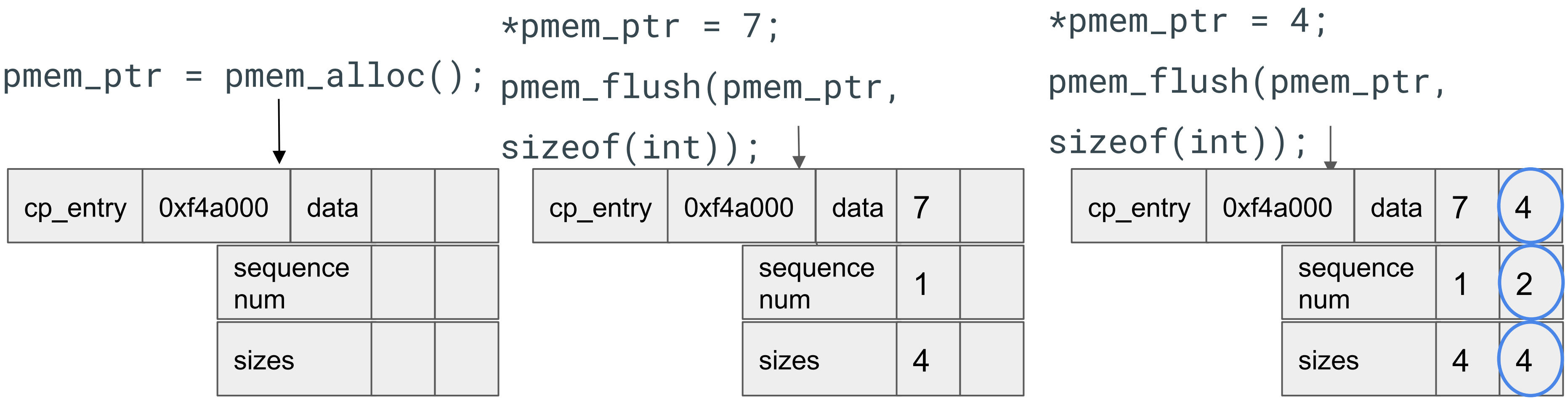
```
pmem_ptr = pmem_alloc();
```

```
*pmem_ptr = 7;
pmem_flush(pmem_ptr,
sizeof(int));
```



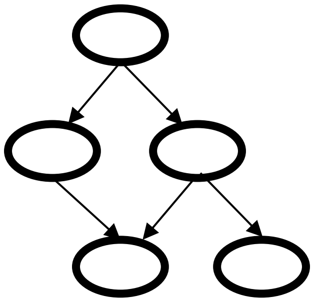
Checkpoint Library

- **Checkpoint multiple versions of PM state to later revert**
- Implementation: Intercepts PM framework API calls
- Global sequence numbers assigned to each PM update
    - ensure order when reverting



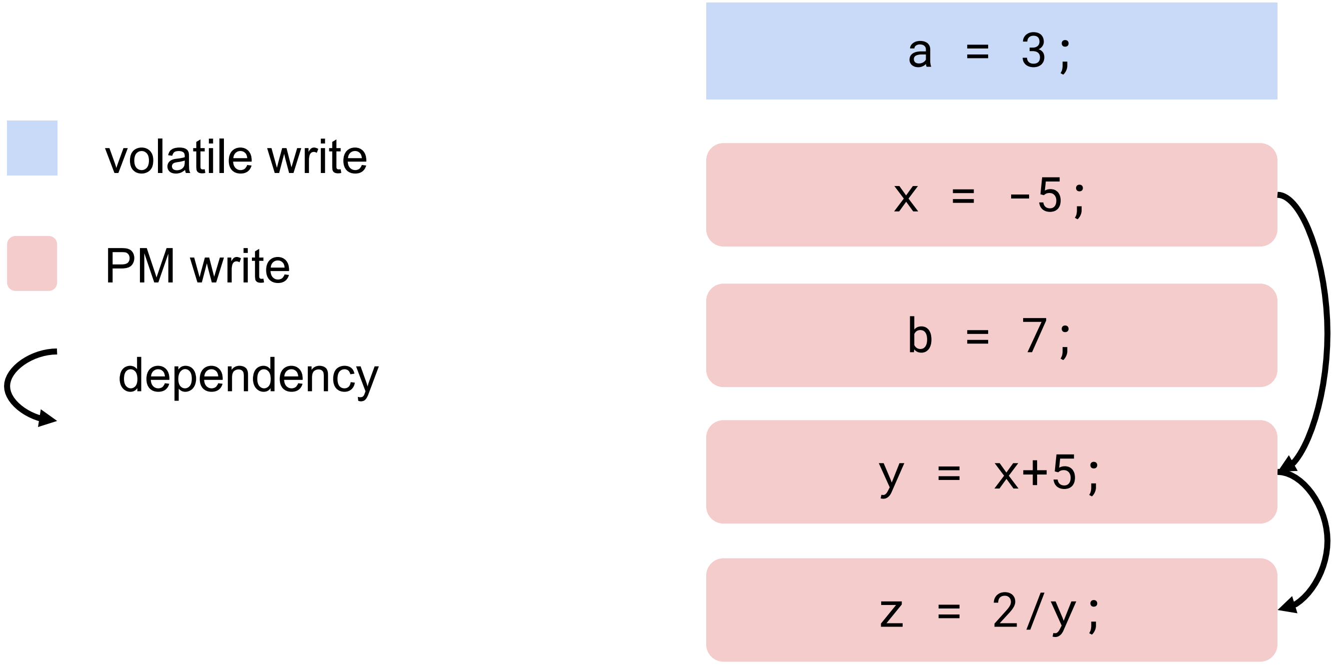
Analyzer: Dependency-based Rollback

- Dependency-based Rollback: More **targeted reversion**
- Use dependencies to **only revert necessary bad states.**
- **Program Dependency Graph** of PM System
  - Fault instruction -> starting point of dependency analysis
  - Slice: see what instructions influence this fault



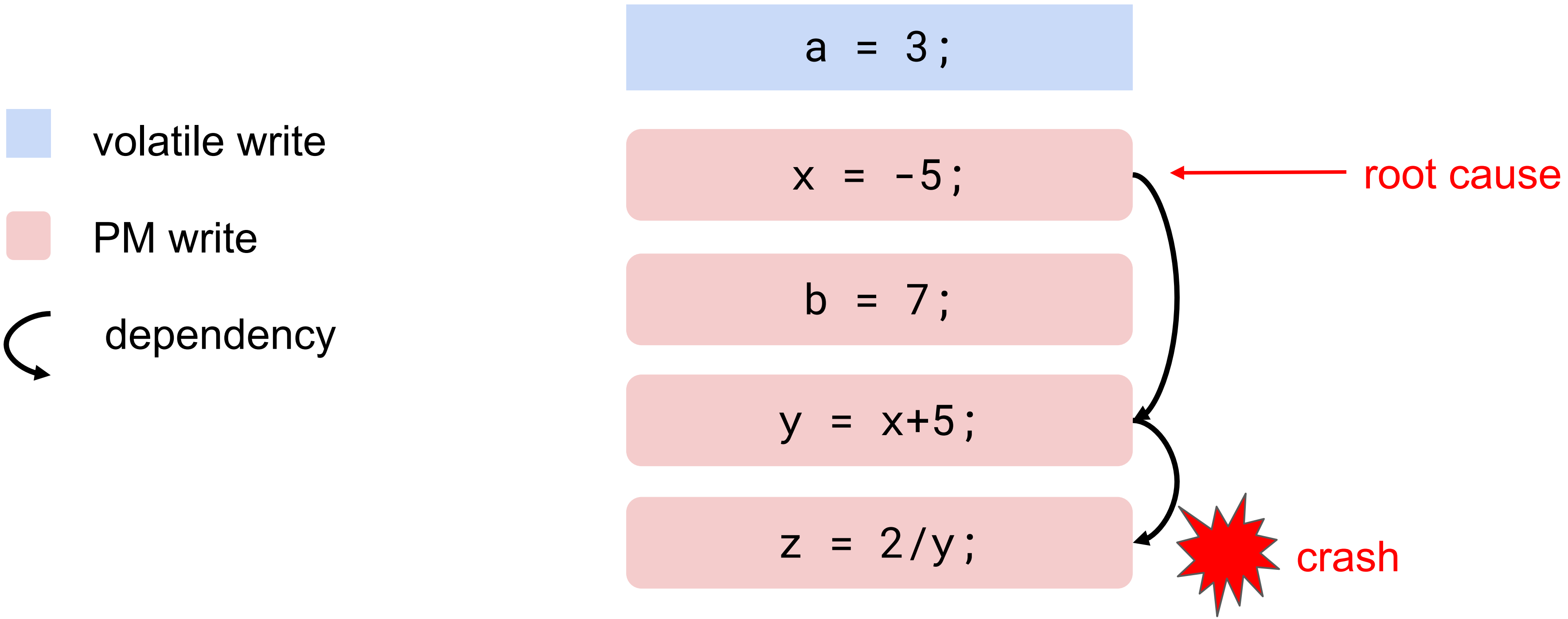
Analyzer: Dependency-based Rollback

- This is the Persistent Memory Write Timeline of a PM system



Analyzer: Dependency-based Rollback

- ❑ Crash at z, root cause due to x = -5;



Arthas Analyzer: Slicing

- Slice: to preserve data dependencies during reversion

```
a = 3;
x =-5;
b = 7;
y = x+5;
z = 2/y;
```

root cause

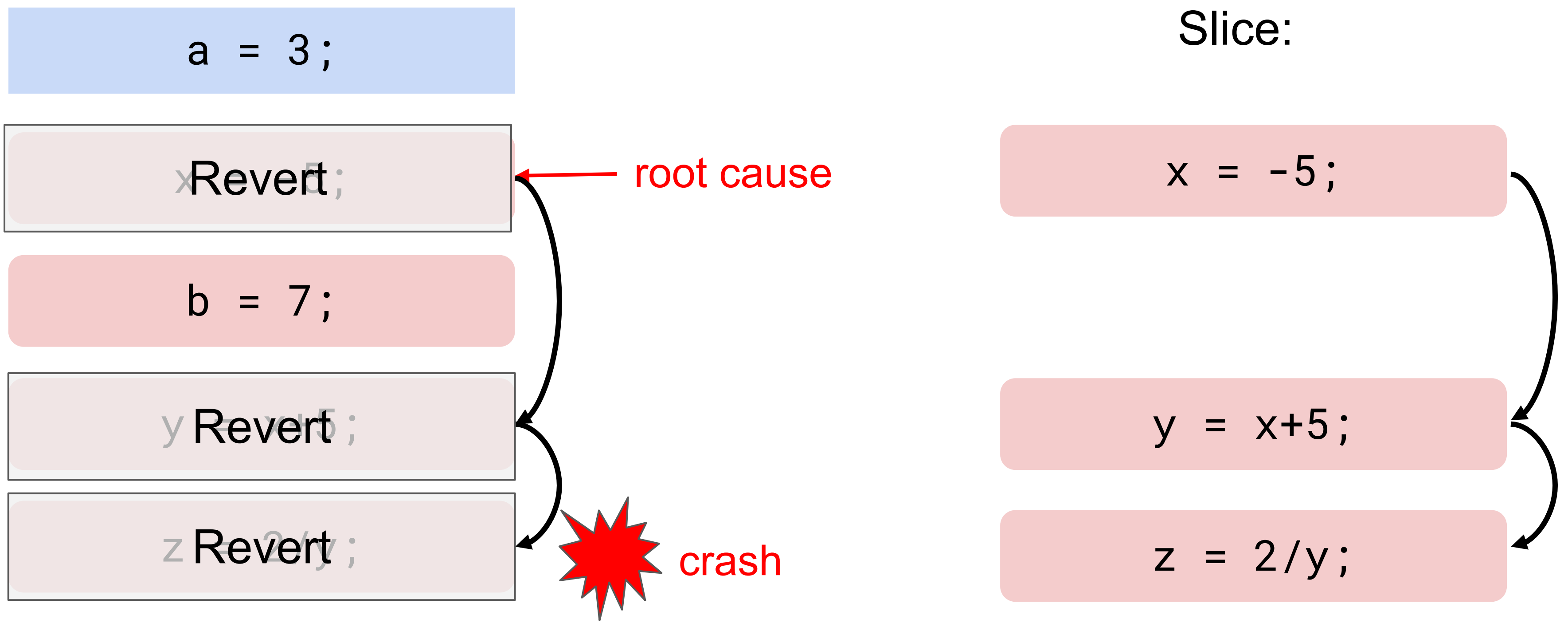
crash

Slice:

```
x = -5;
y = x+5;
z = 2/y;
```

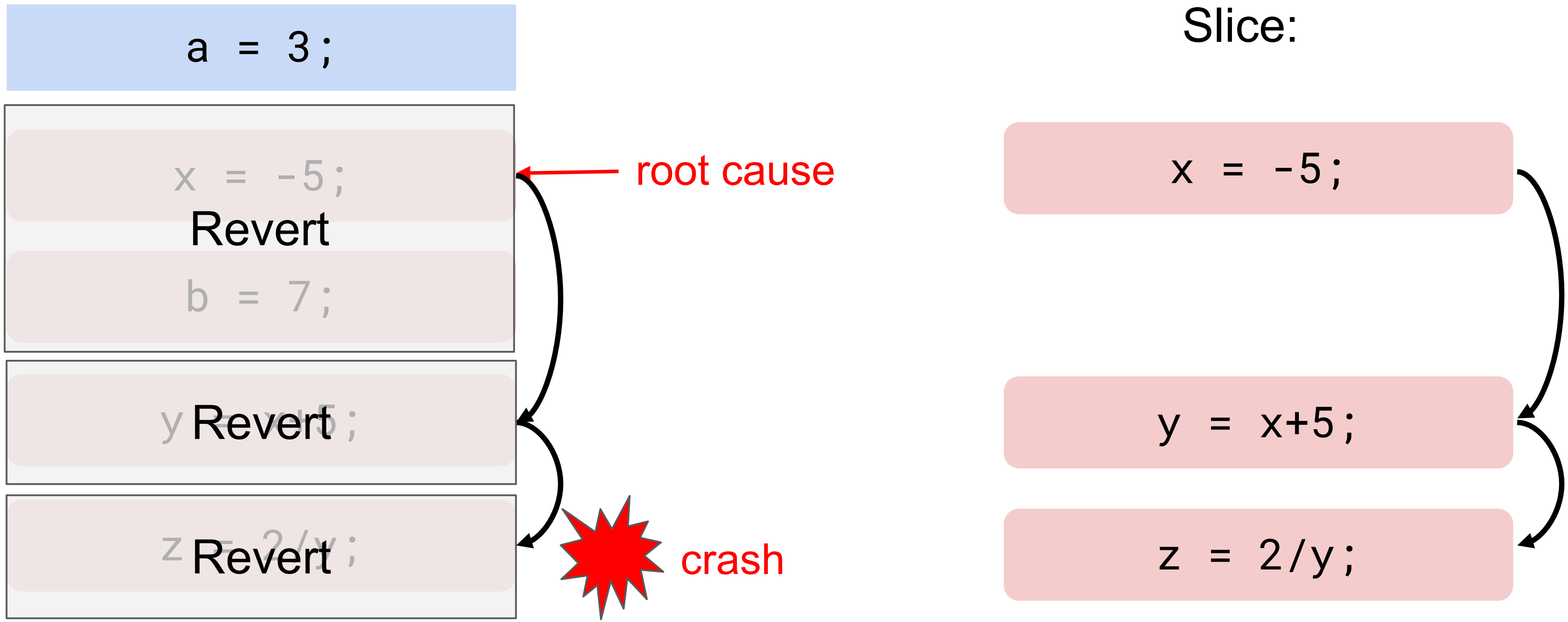
Arthas Analyzer: Purge Mode

- Minimizes data loss, but may not lead to a perfectly consistent system



Arthas Analyzer: Rollback Mode

- ❑ More conservative approach: reverts between dependent updates
- ❑ Captures dependencies of the system



Evaluation

- One 8-core CPU (2.50GHz) and two 128 GB Intel Optane DC Persistent Memory DIMMs.
- We test on 12 bugs both from our study and other existing bugs
- We run Arthas against two baselines
  - **pmCRIU:** A state-of-the-art checkpoint and rollback system
  - **ArCkpt:** Alternate version of Arthas that reverts without the analyzer component, time-based

	Memcached	Redis	Pelikan	PMEMKV	CCEH
Type	Port	Port	Port	New	New
Bugs	5	3	2	1	1

12 Real-World Bugs

No.	System	Fault	Consequence
f1	Memcached	Refcount Overflow	Data loss
f2	Memcached	flush_all logic bug	Data loss
f3	Memcached	Hashtable lock data race	Data loss
f4	Memcached	Integer overflow in append	Segfault
f5	Memcached	Rehashing flag bit flip	Data loss
f6	Redis	Listpack buffer overflow	Segfault
f7	Redis	Logic bug in refcount	Server panic
f8	Redis	slowlogEntry leak	Persistent leak
f9	CCEH	Directory doubling bug	Infinite loop
f10	Pelikan	Value Length overflow	Segfault
f11	Pelikan	Null stats response	Segfault
f12	PMEMKV	Asynchronous lazy free	Persistent leak

We evaluate on a diverse set of bugs with varying consequences

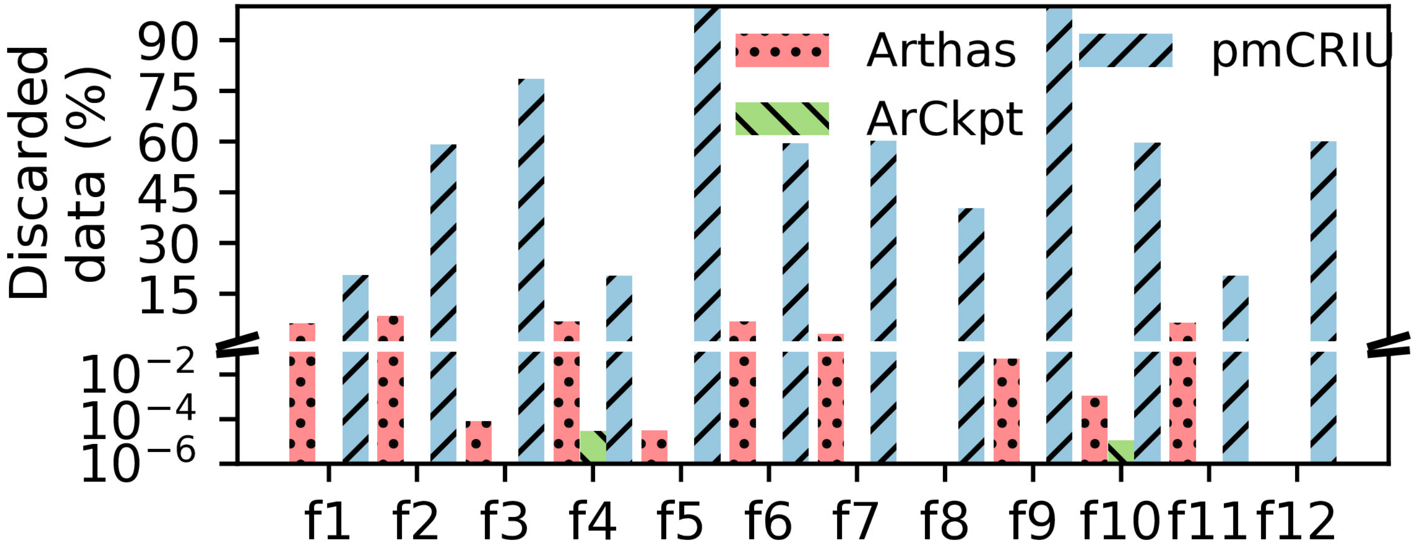
Effectiveness

- Arthas is able to resolve 12 out of the 12 bugs
- ArCkpt suffers from timeouts: 2 out of the 12 bugs
- pmCRIU is only able to reliably mitigate 9 out of the 12 bugs.

Solution	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12
pmCRIU	✓	✓	✗	✓	1/10	✓	✓	4/10	✓	✓	✓	✓
ArCkpt	✗	✗	✗	✓	✗	✗	✗	✗	✗	✓	✗	✗
Arthas	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

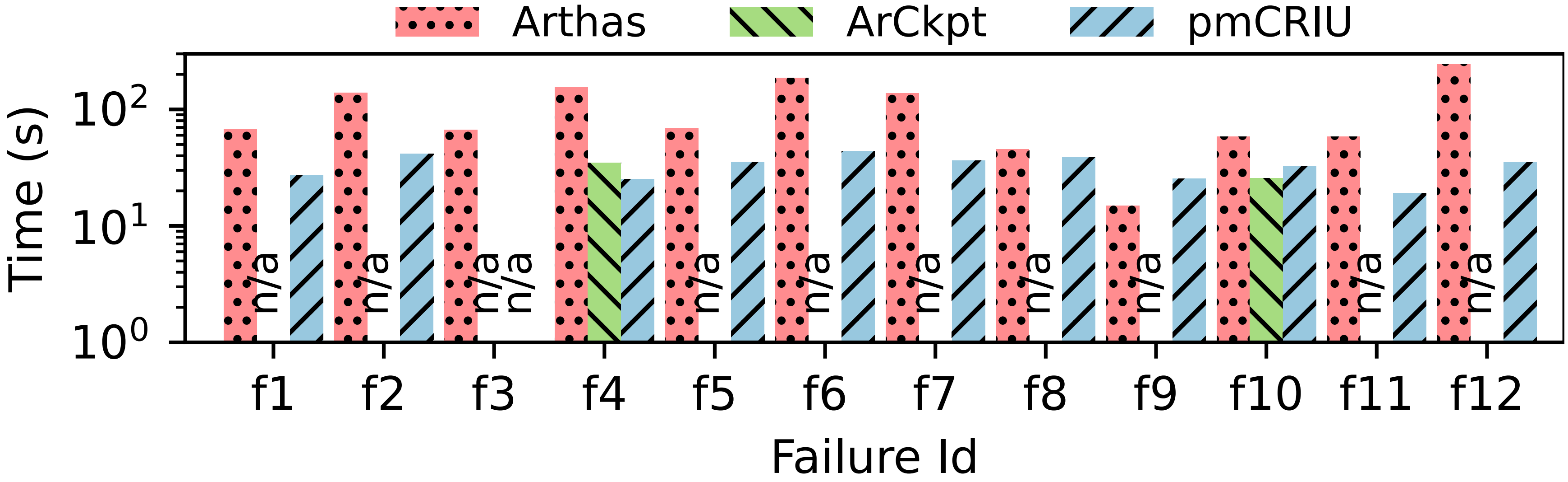
Data Discarded by Solutions

Arthas discards 10x less data than pmCRIU



Recovery Time

Arthas is slightly slower, but is still in an acceptable range of approximately **one minute more** than pmCRIU while also minimizing data loss.



Conclusion

- Soft-to-Hard Faults are an underexplored, yet significant problem for new PM systems.
- Arthas reliably detects and mitigates Hard Faults in PM systems
  - Dependency-based rollback - minimize data loss
- Mitigate 12 Hard Faults from 5 PM systems with 10x less data loss than pmCRIU with a reasonable performance overhead.
- Our tool is publicly available at https://github.com/OrderLab/Arthas