Finding Crash-Consistency Bugs with Bounded Black-Box Testing

Jayashree Mohan, Ashlie Martinez, Soujanya Ponnapalli, Pandian Raju, Vijay Chidambaram

vmware®

TEXAS

The University of Texas at Austin

Crashes

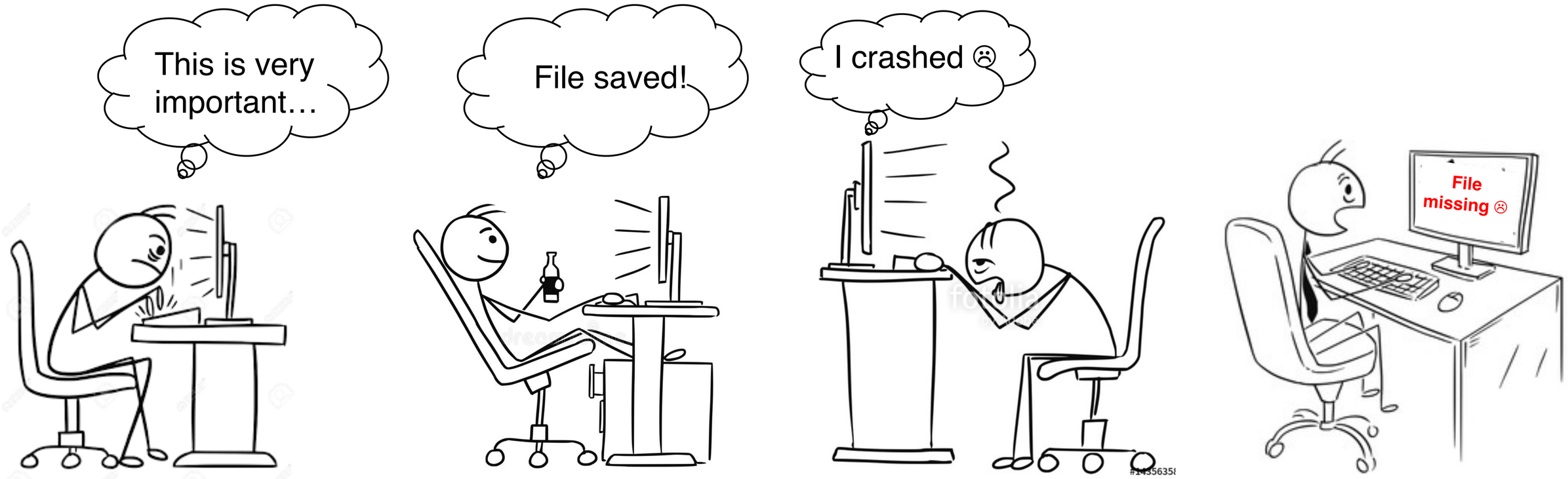
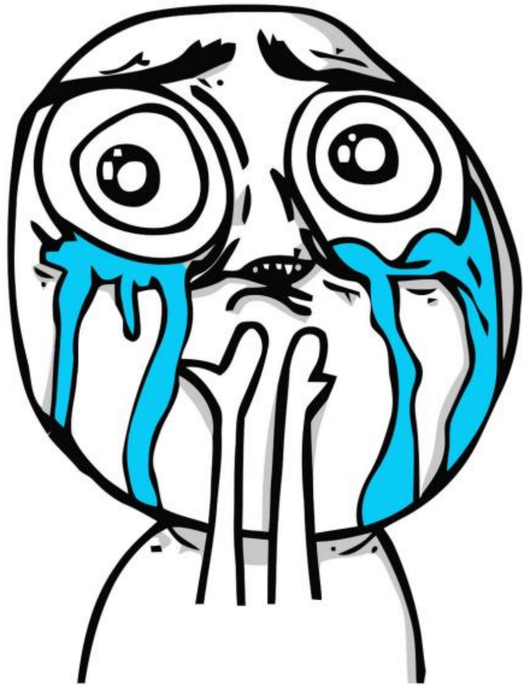


Image source : https://www.fotolia.com

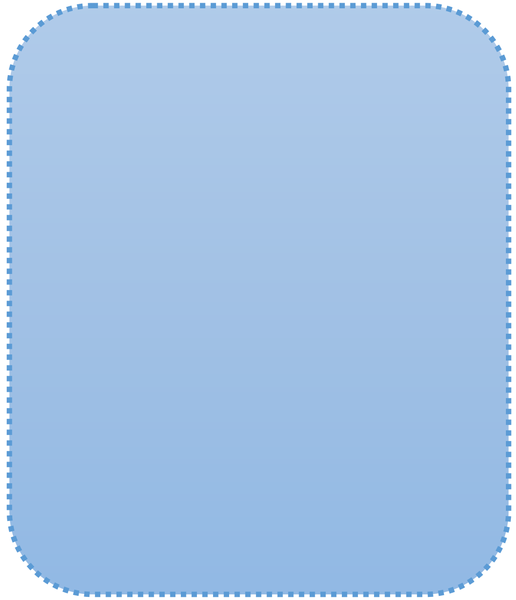
I wish filesystems were crash-consistent!



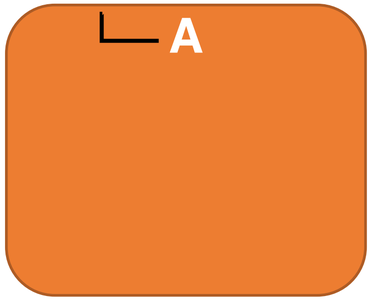
Rename atomicity bug in btrfs

Memory

Storage

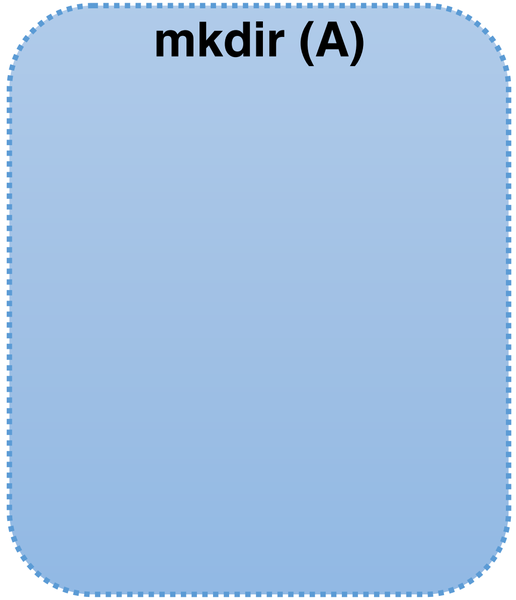


Rename atomicity bug in btrfs

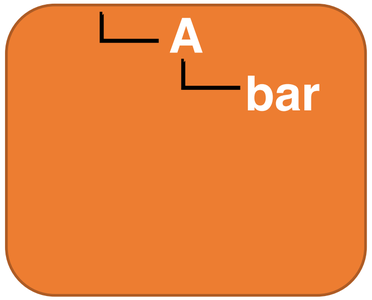


Memory

Storage



Rename atomicity bug in btrfs

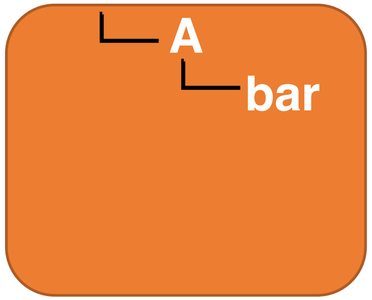


Memory

Storage

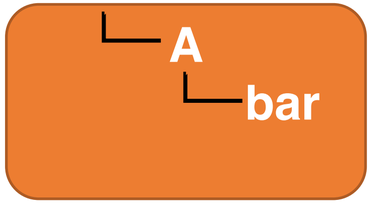
mkdir (A)
touch (A/bar)

Rename atomicity bug in btrfs



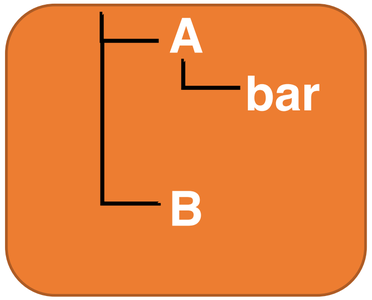
Memory

Storage



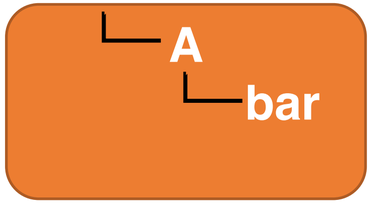
mkdir (A)
touch (A/bar)
fsync (A/bar)

Rename atomicity bug in btrfs



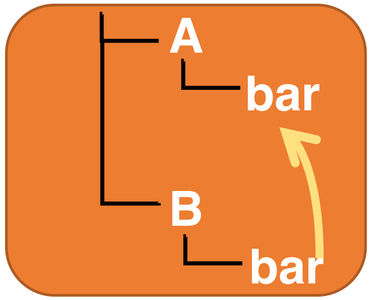
Memory

Storage



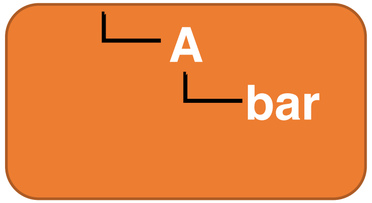
mkdir (A)
touch (A/bar)
fsync (A/bar)
mkdir (B)

Rename atomicity bug in btrfs



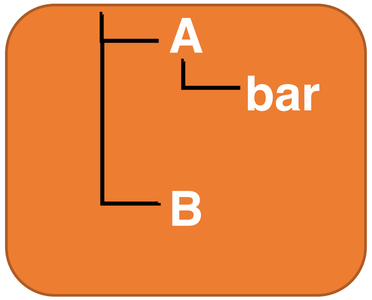
Memory

Storage



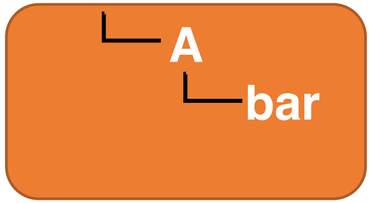
mkdir (A)
touch (A/bar)
fsync (A/bar)
mkdir (B)
touch (B/bar)

Rename atomicity bug in btrfs



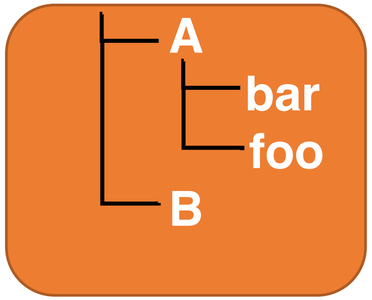
Memory

Storage



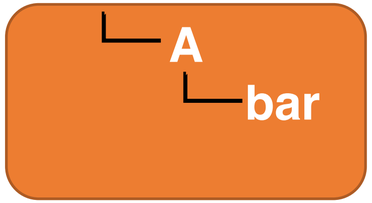
mkdir (A)
touch (A/bar)
fsync (A/bar)
mkdir (B)
touch (B/bar)
rename (B/bar, A/bar)

Rename atomicity bug in btrfs



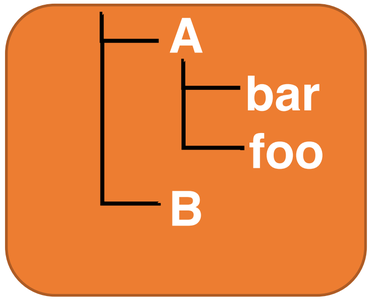
Memory

Storage



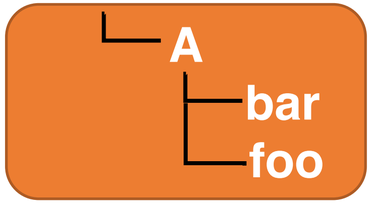
mkdir (A)
touch (A/bar)
fsync (A/bar)
mkdir (B)
touch (B/bar)
rename (B/bar, A/bar)
touch (A/foo)

Rename atomicity bug in btrfs



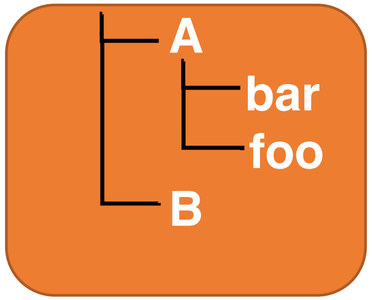
Memory

Storage



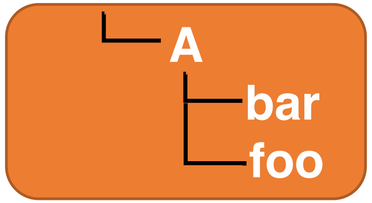
mkdir (A)
touch (A/bar)
fsync (A/bar)
mkdir (B)
touch (B/bar)
rename (B/bar, A/bar)
touch (A/foo)
fsync (A/foo)

Rename atomicity bug in btrfs



Memory

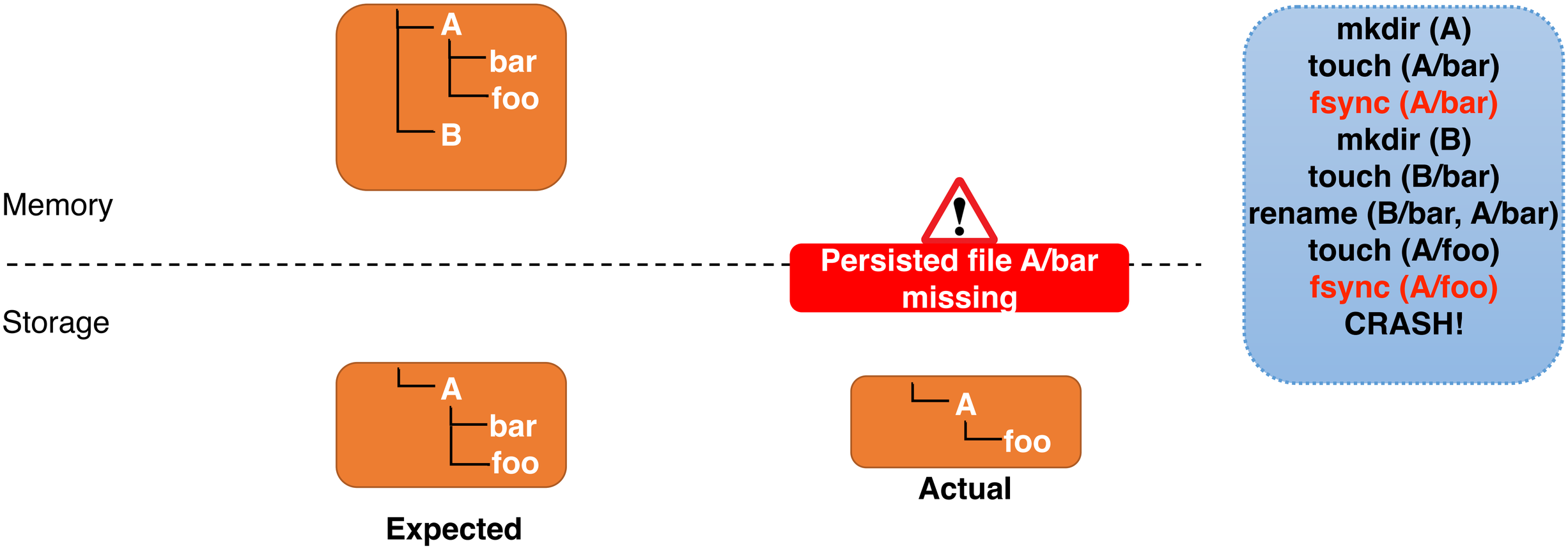
Storage



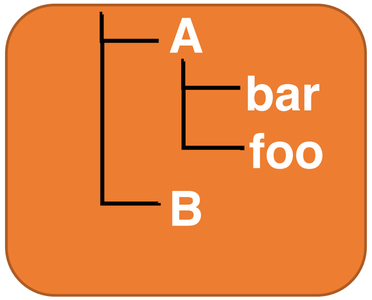
Expected

mkdir (A)
touch (A/bar)
fsync (A/bar)
mkdir (B)
touch (B/bar)
rename (B/bar, A/bar)
touch (A/foo)
fsync (A/foo)
CRASH!

Rename atomicity bug in btrfs



Rename atomicity bug in btrfs



Memory

Storage

Persisted file A/bar missing

mkdir (A)
touch (A/bar)
fsync (A/bar)
mkdir (B)
touch (B/bar)
rename (B/bar, A/bar)
touch (A/foo)
fsync (A/foo)
CRASH!

Exists in the kernel since 2014!
Found by ACE and CrashMonkey

Testing Crash Consistency Today

- Build FS from scratch

Verified Filesystems

Model Checking

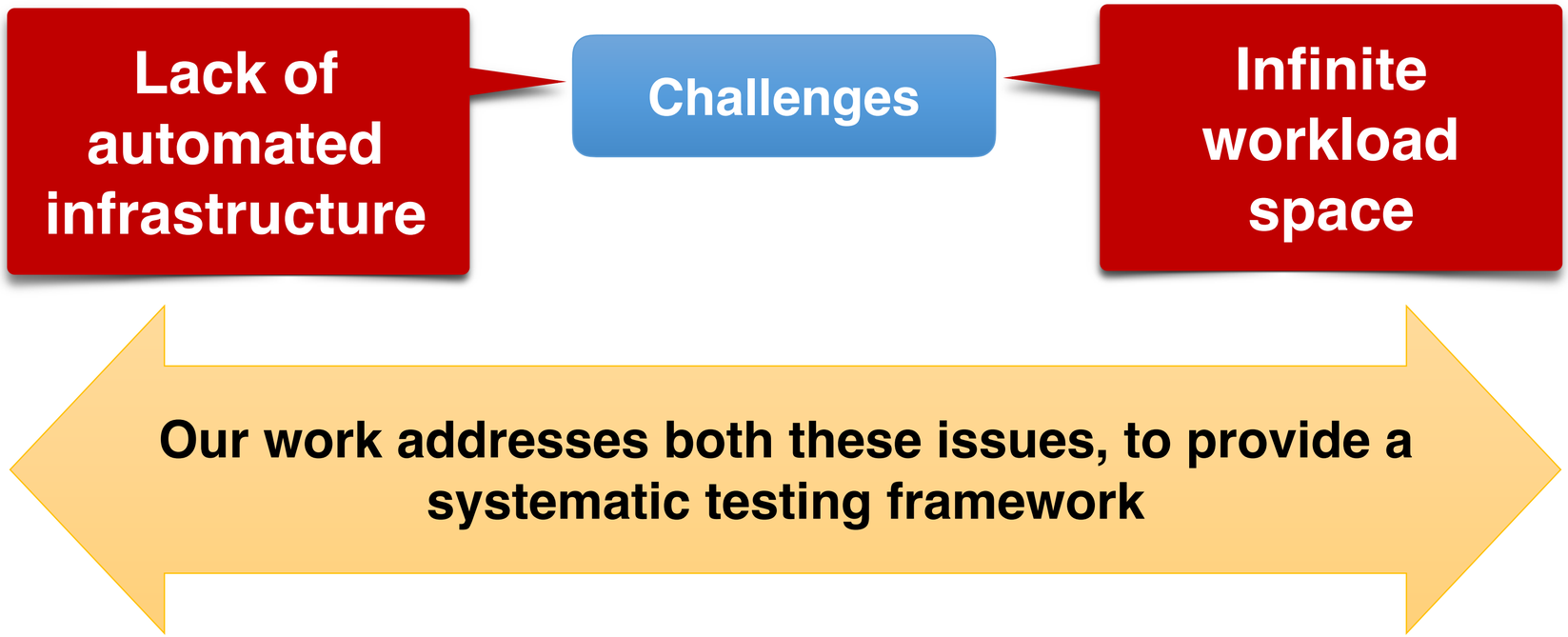
- Annotate filesystems
- Hard to do for existing FS

- State of the Art : xfstest suite
  - Collection of 482 regression tests

Only 5% of tests in xfstest check for file system crash consistency

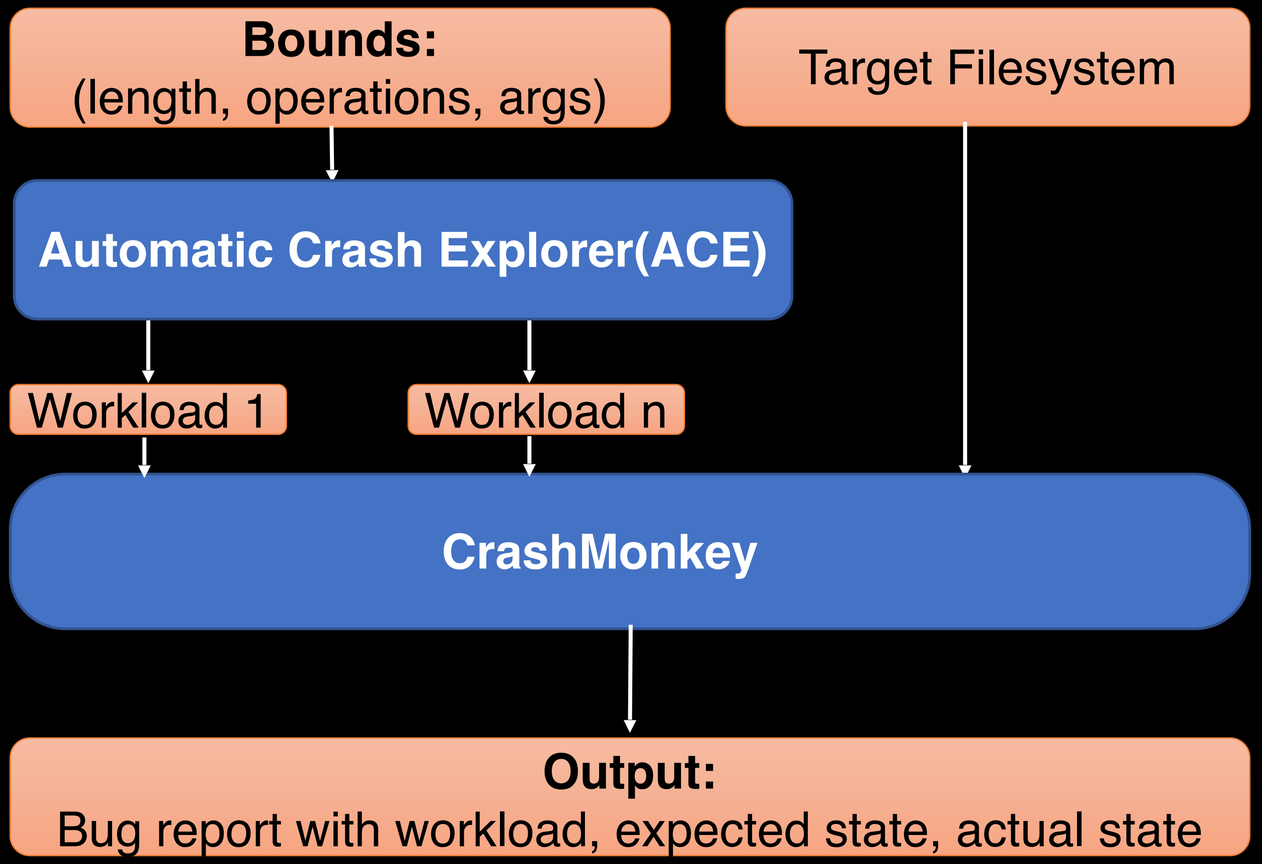
Challenges with systematic testing

Systematically generate workloads



Bounded Black-Box Crash Testing (B^3)

New approach to testing file-system crash consistency



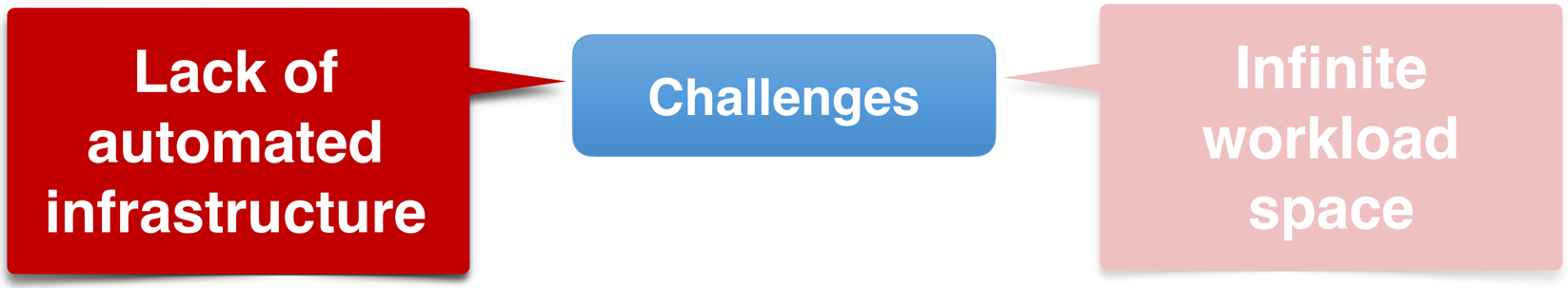
- ➡ Focus on reproducible bugs resulting in metadata corruption, data loss.
- ➡ Found 10 new bugs across btrfs and F2FS;
- ➡ Found 1 bug in FSCQ (verified file system)
- ➡ Filesystem agnostic – works with any POSIX file system

www.github.com/utsaslab/crashmonkey

Outline

- CrashMonkey
- Bounded Black Box Crash Testing
- Automatic Crash Explorer (ACE)
- Demo

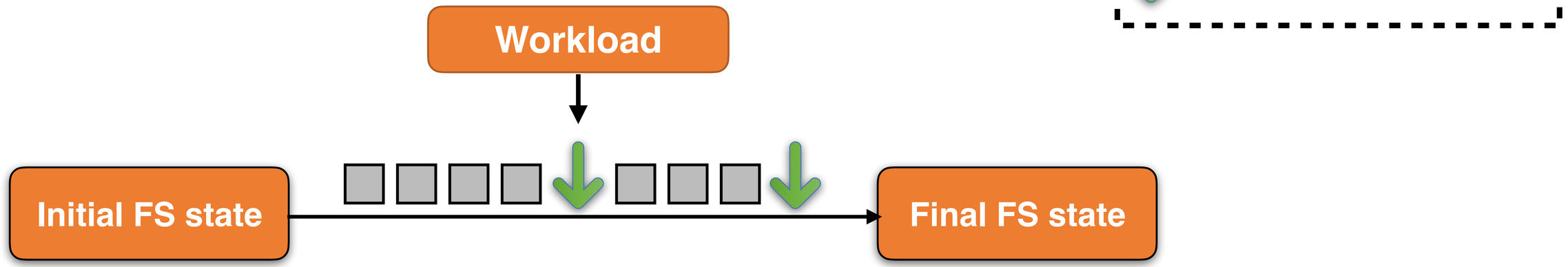
Challenges with systematic testing



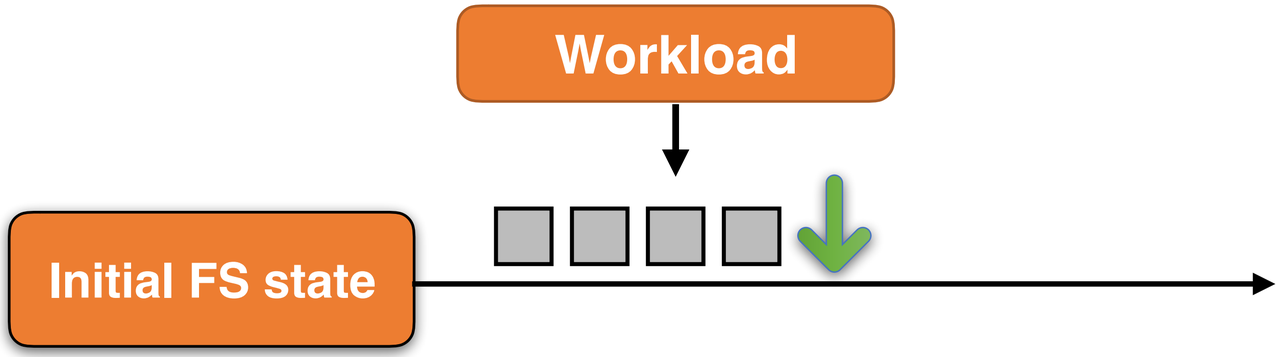
CrashMonkey

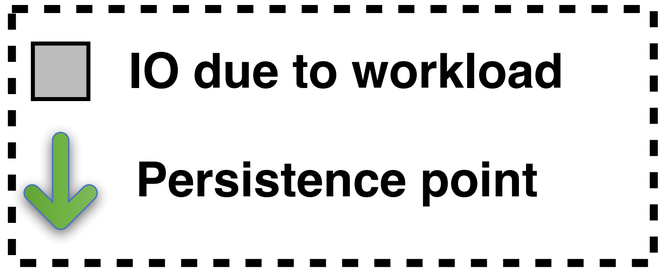
- Efficient infrastructure to record and replay block level IO requests
- Simulate crash at different points in the workload
- Automatically test for consistency after crash.
- Copy-on-write RAM block device

CrashMonkey in Action

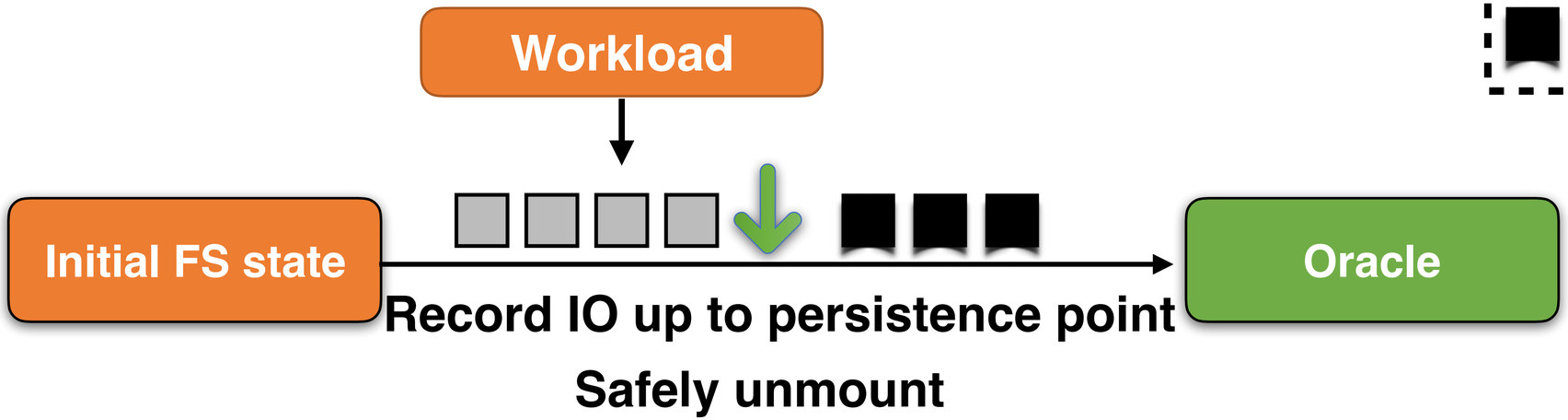


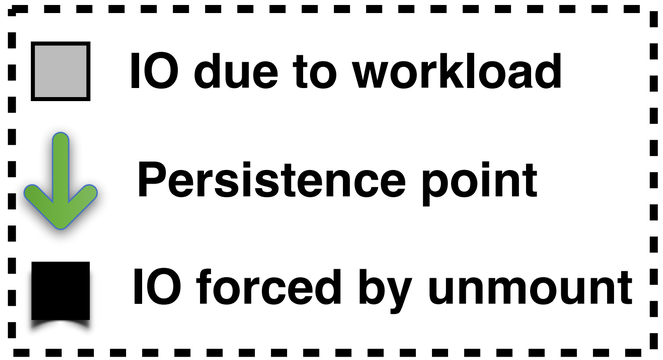
CrashMonkey in Action



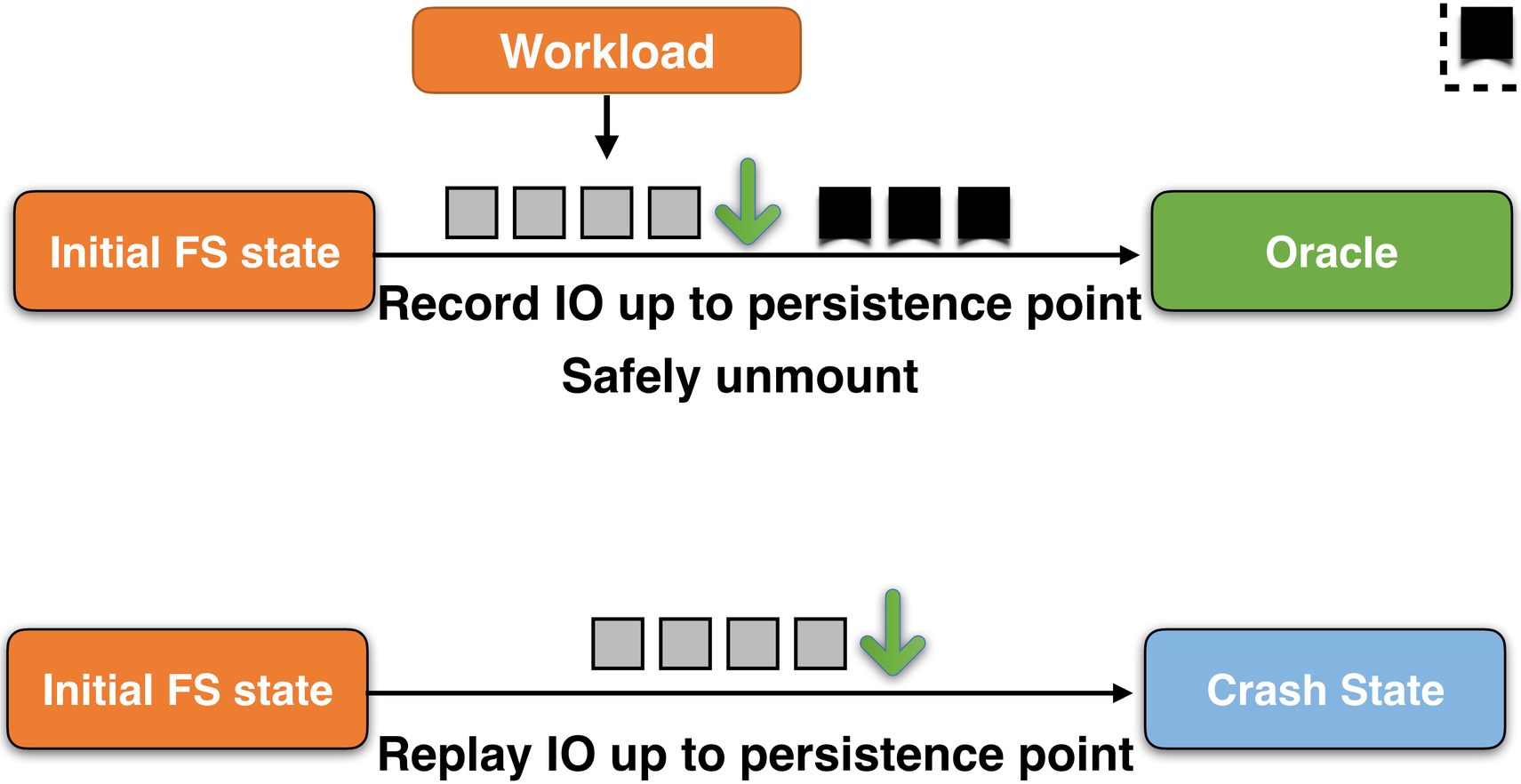


Phase 1 : Record IO

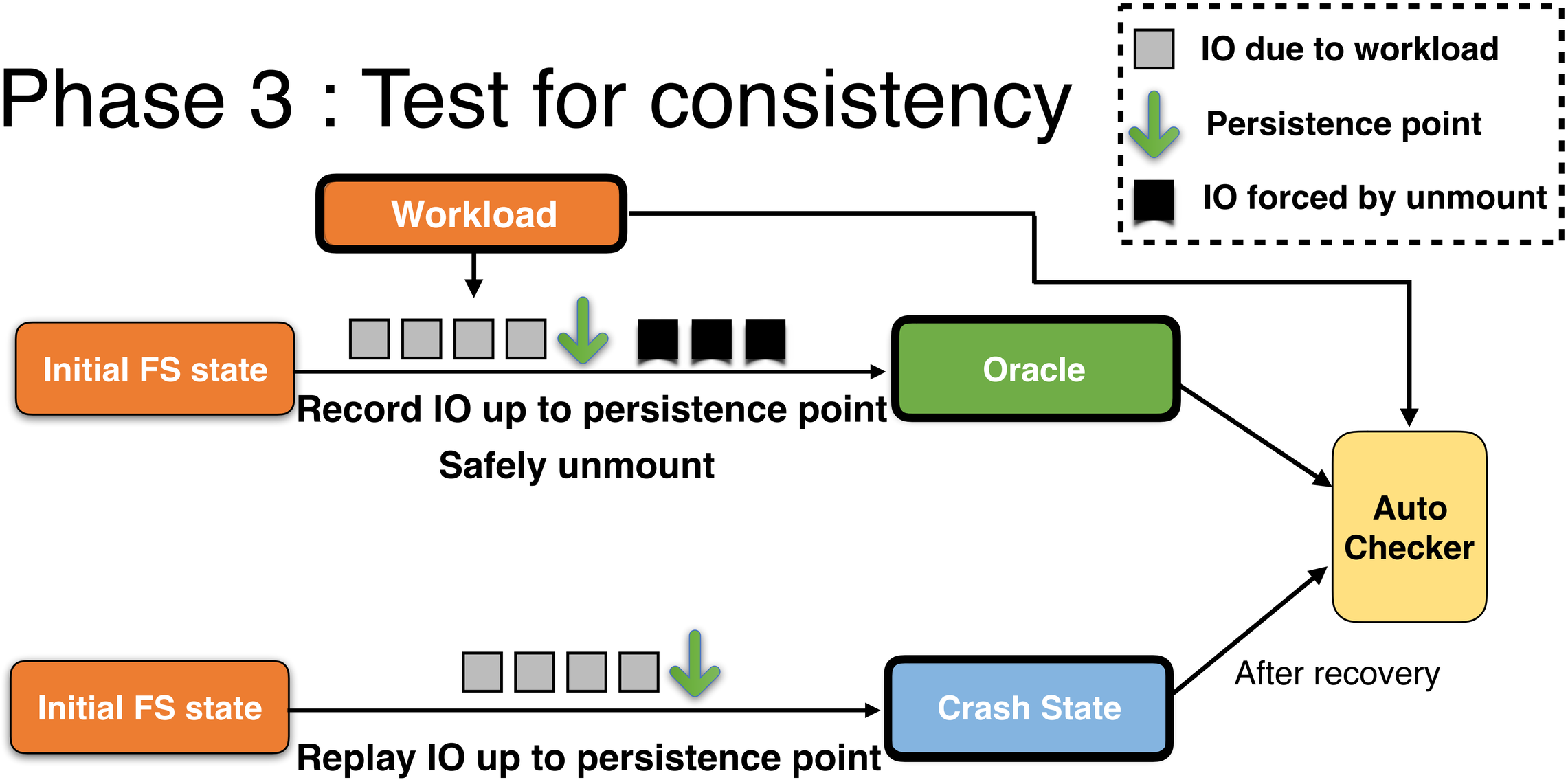




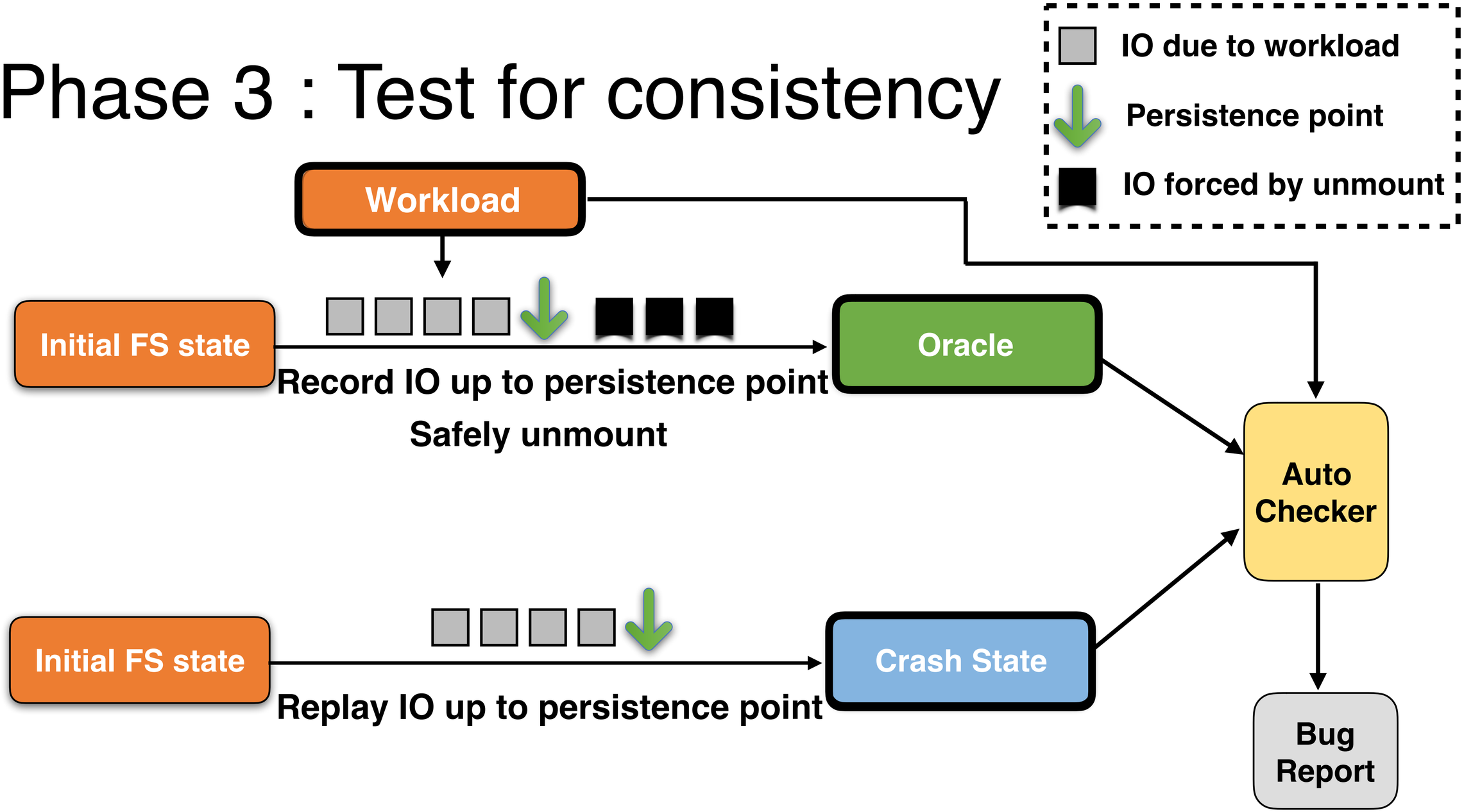
Phase 2 : Replay IO



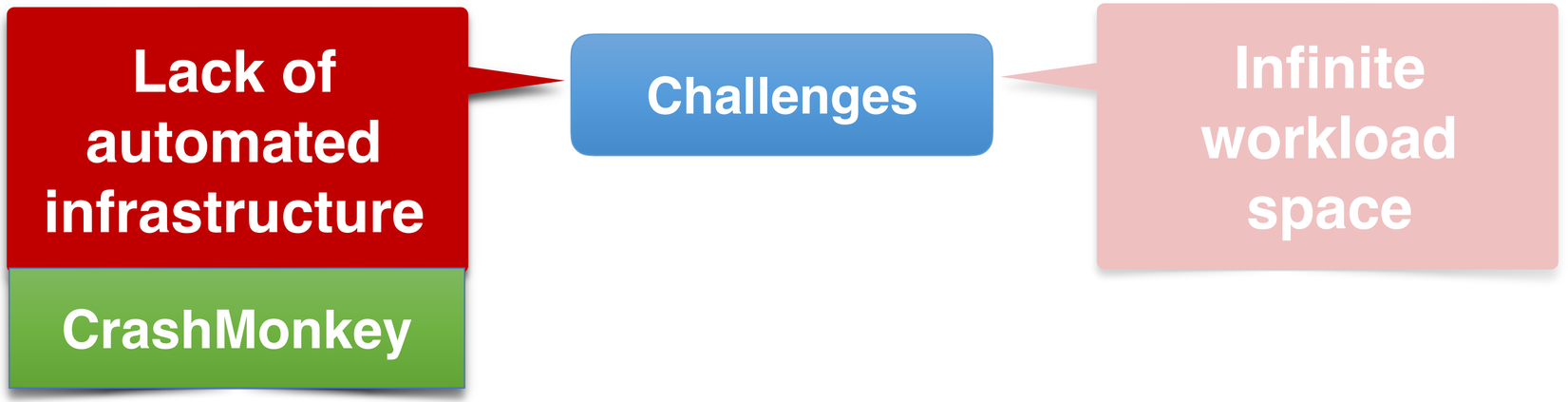
Phase 3 : Test for consistency



Phase 3 : Test for consistency



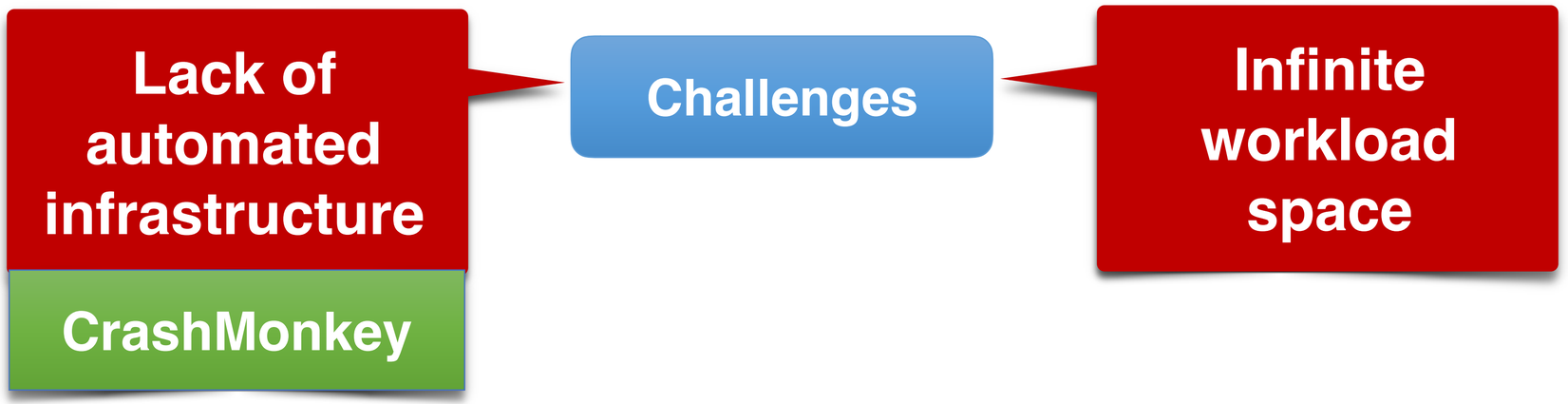
Challenges with Systematic Testing



So Far…

- Given a workload compliant to POSIX API, we saw how CrashMonkey generates crash states and automatically tests for consistency

Challenges with Systematic Testing



So Far…

- Given a workload compliant to POSIX API, we saw how CrashMonkey generates crash states and automatically tests for consistency

- Next question : How to automatically generate workloads in an the infinite workload space?

Exploring the infinite workload space

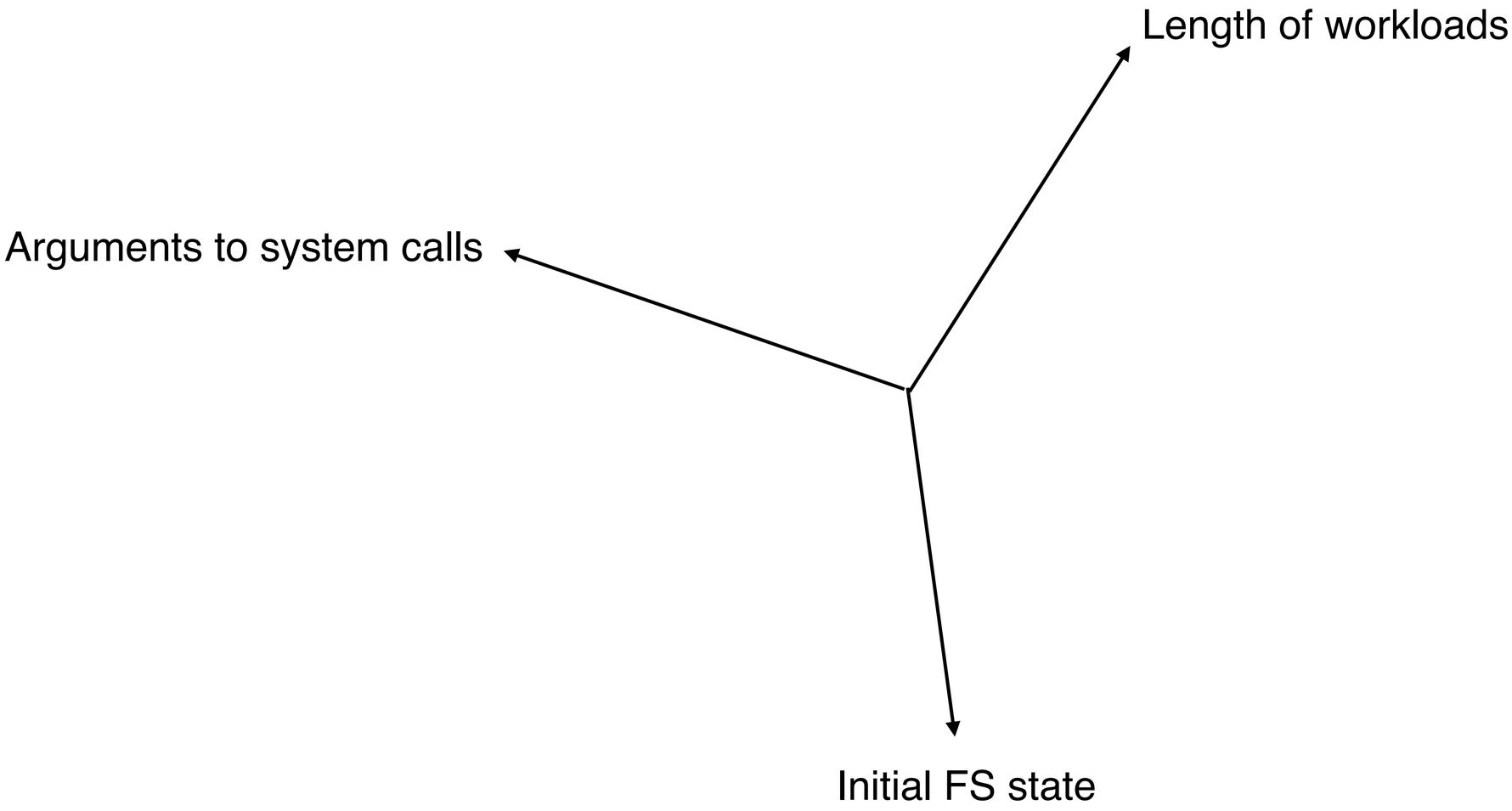
Challenges:

- Infinite length of workloads
- Large set of filesystem operations
- Infinite parameter options (file/directory names, depth)
- Infinite options for initial filesystem state
- When in the workload to simulate a crash?

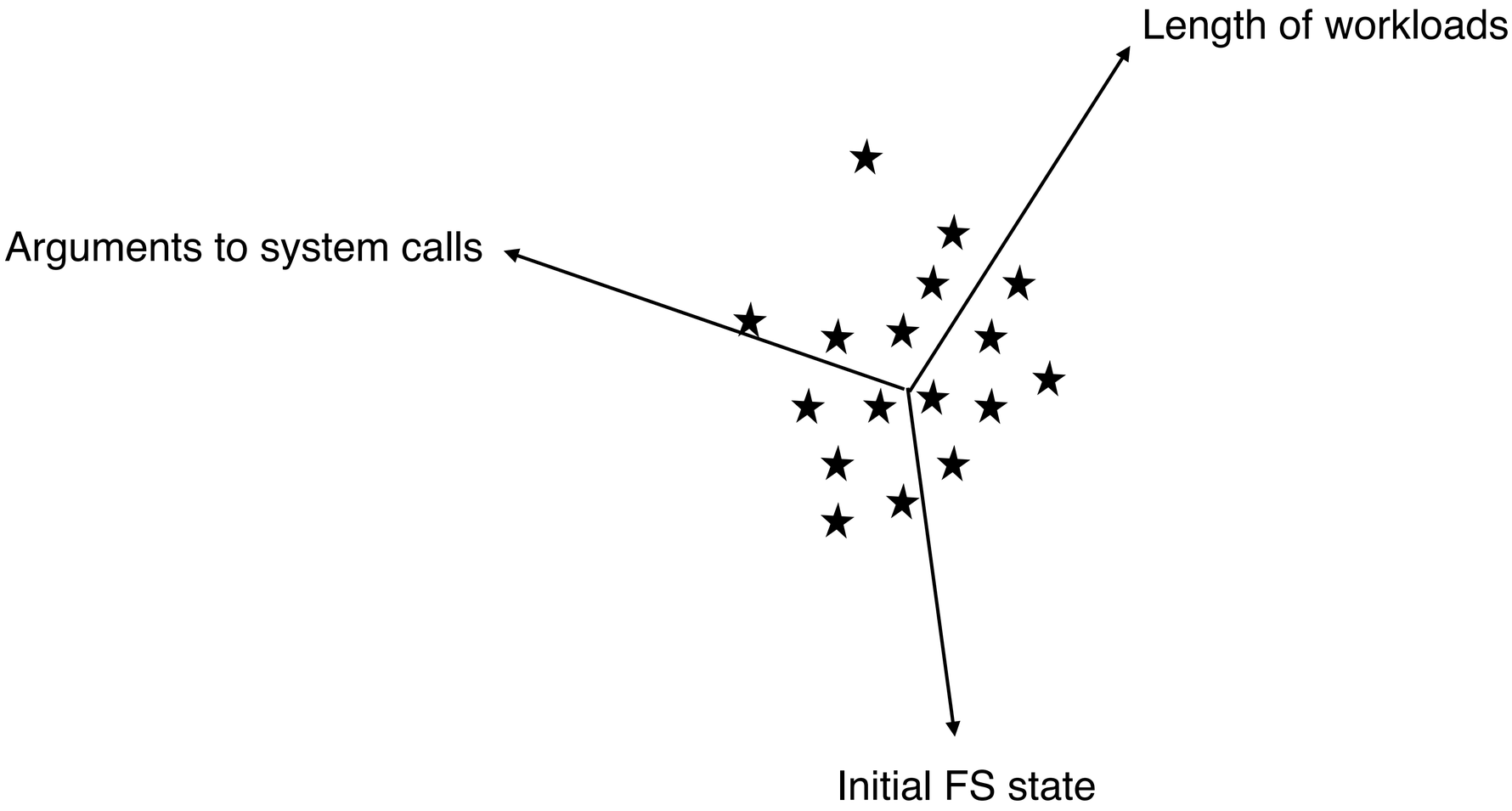
Outline

- CrashMonkey
- **Bounded Black Box Crash Testing**
- Automatic Crash Explorer (ACE)
- Demo

B³ : Bounded Black Box Crash Testing



B³ : Bounded Black Box Crash Testing



B³ : Bounded Black Box Crash Testing

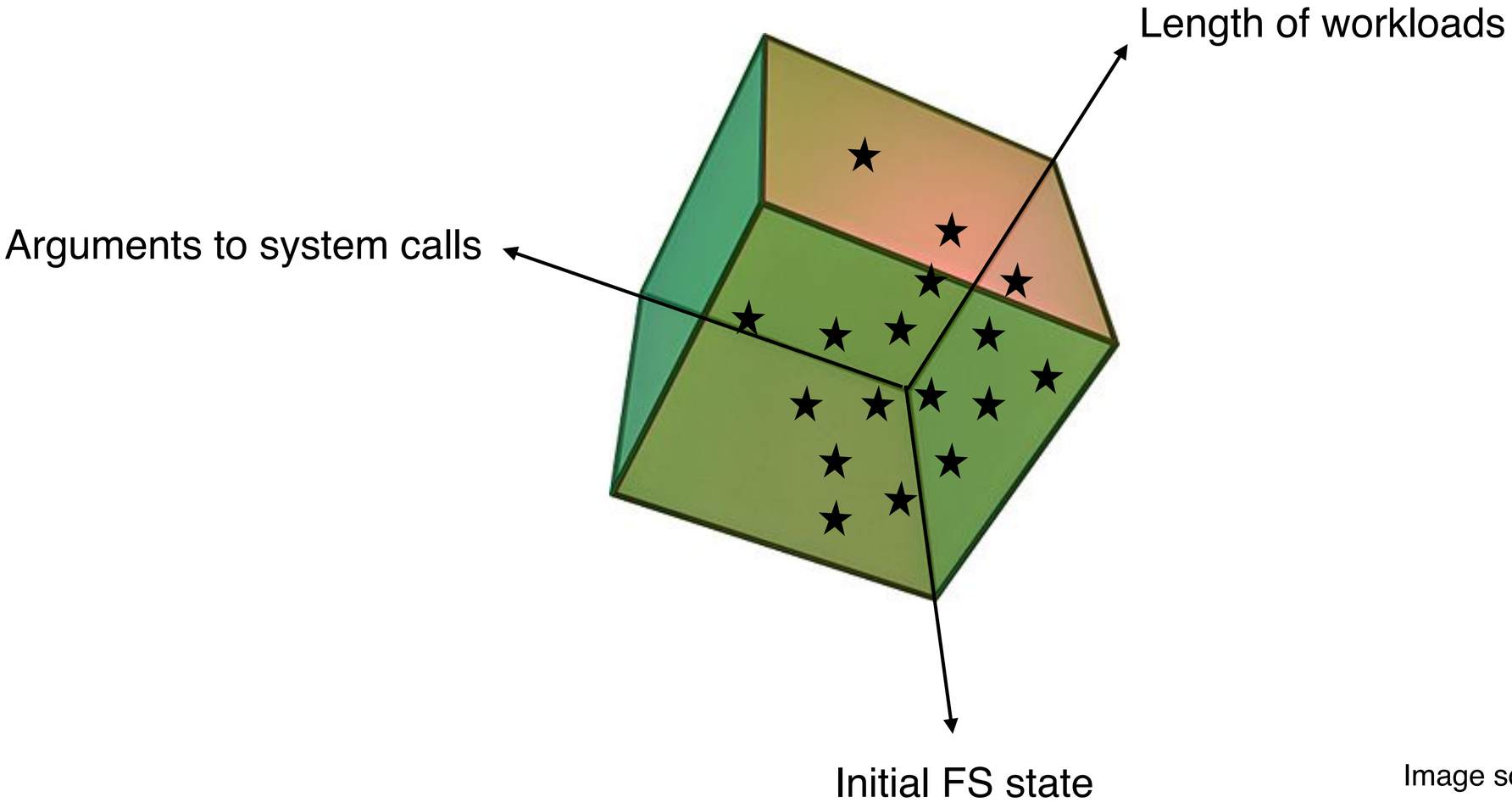
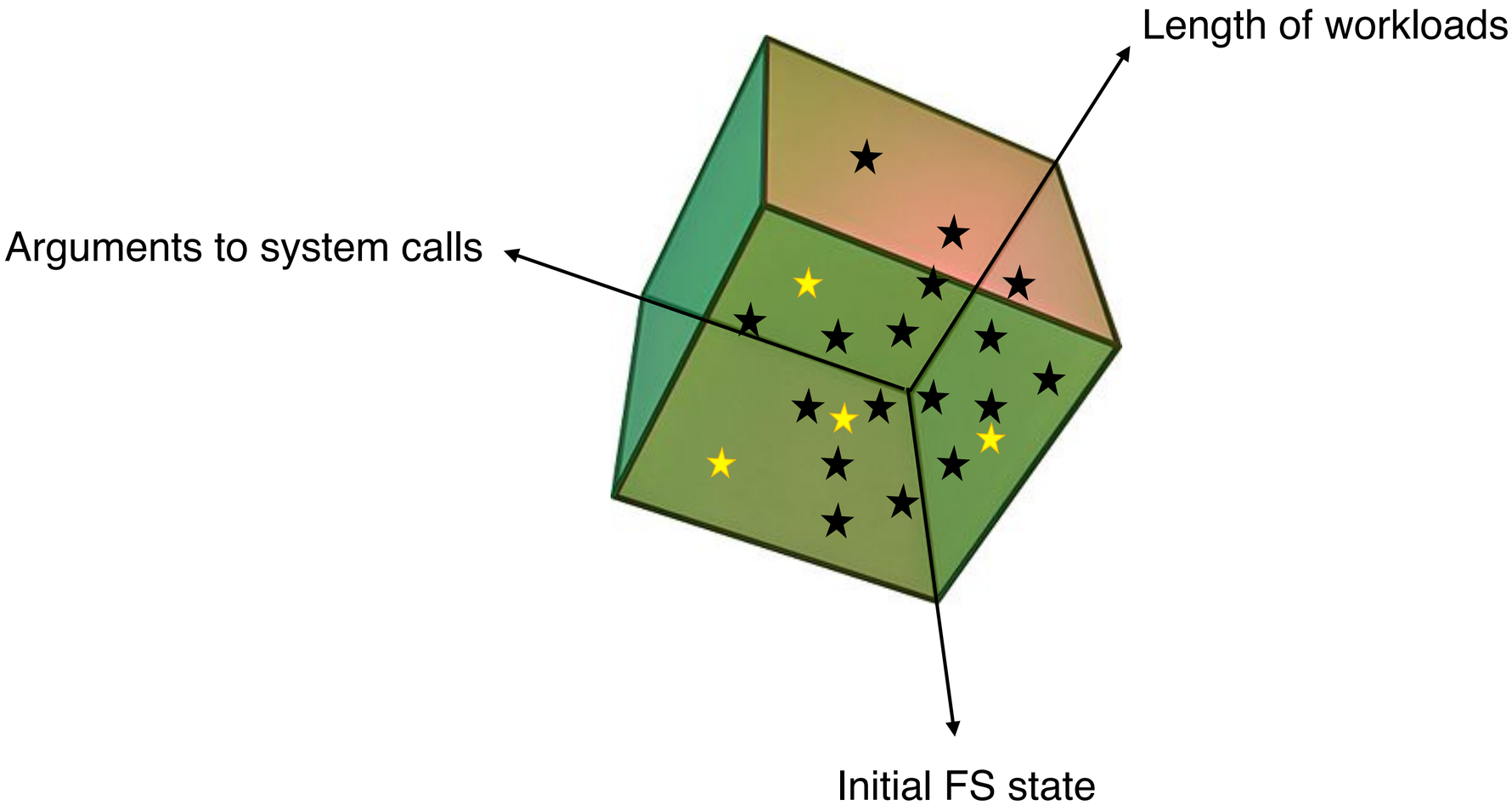
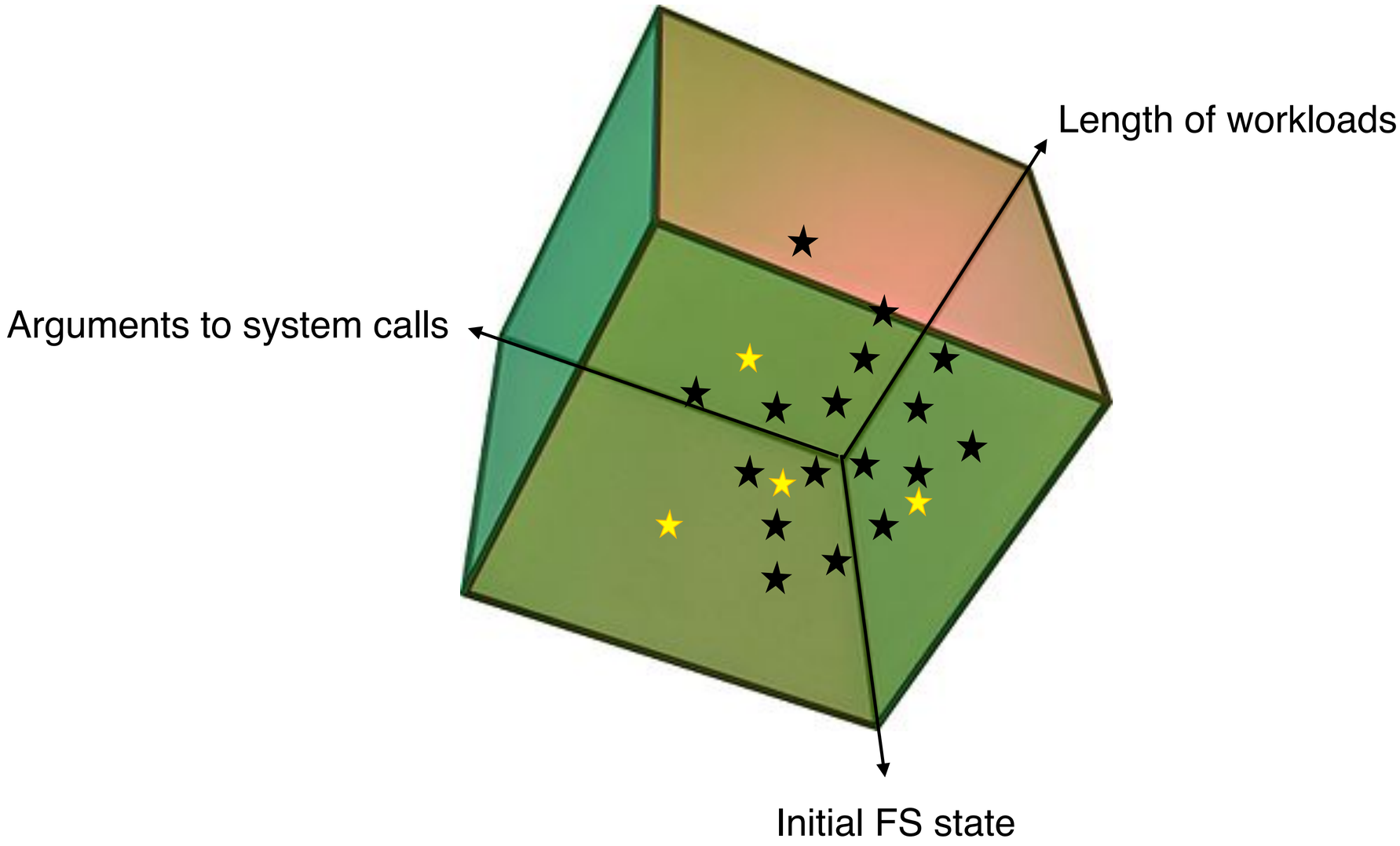


Image source: https://en.wikipedia.org/wiki/Cube

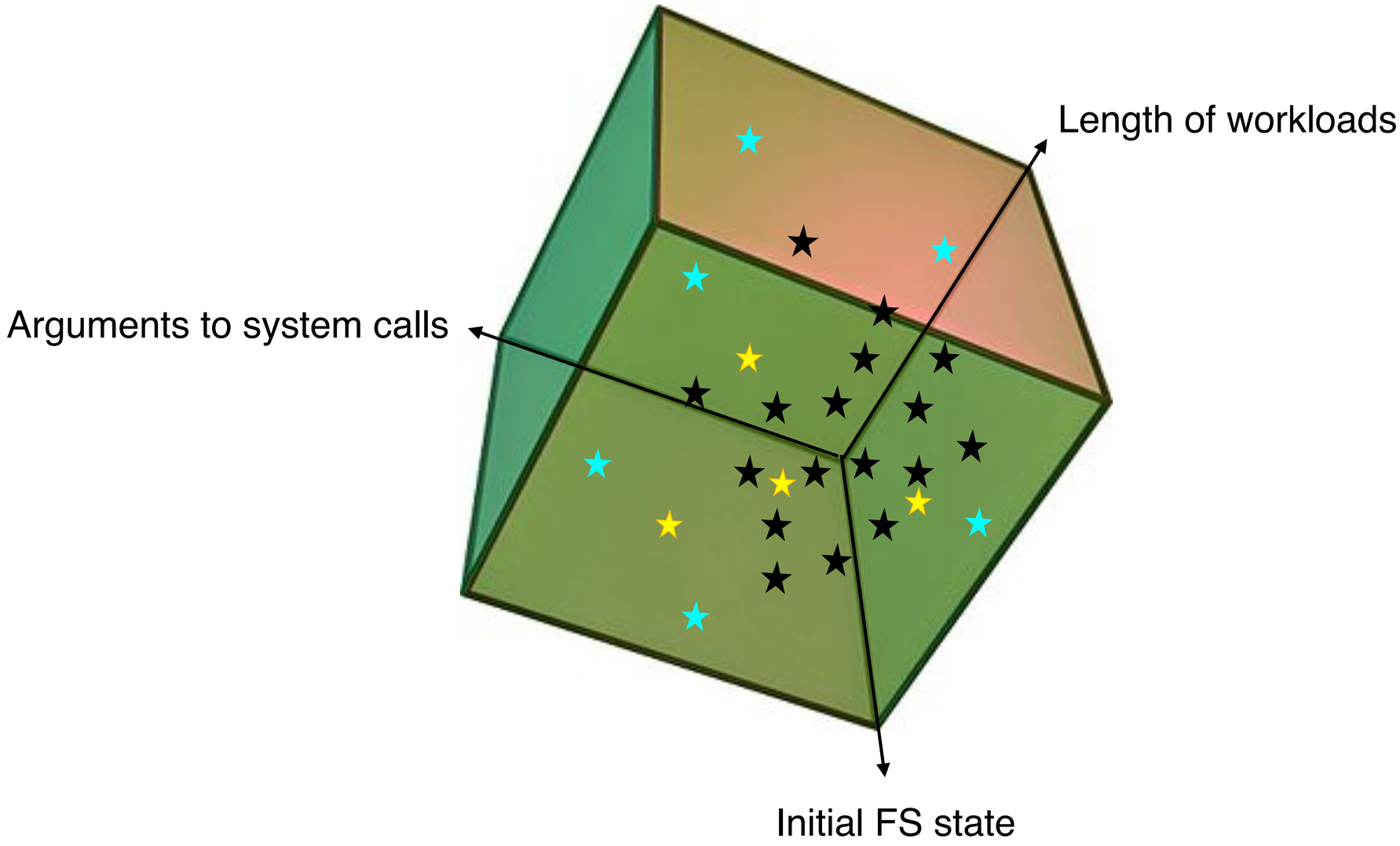
B³ : Bounded Black Box Crash Testing



B³ : Bounded Black Box Crash Testing



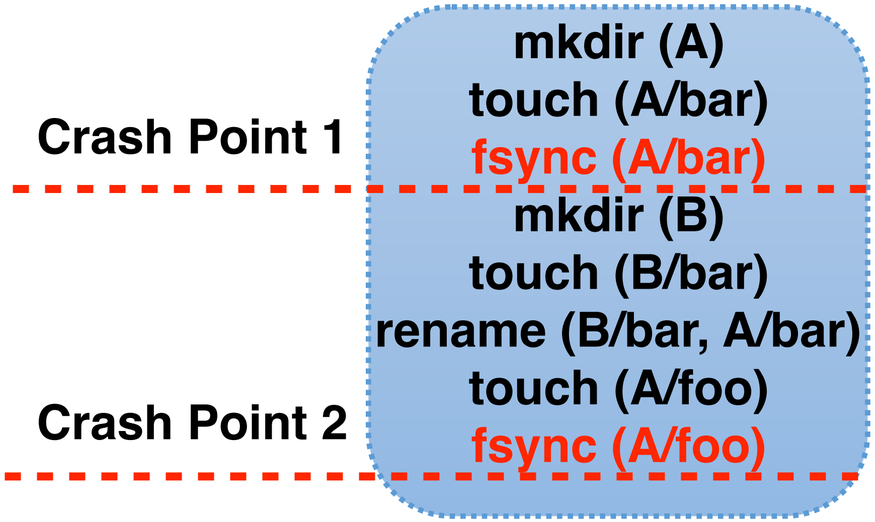
B³ : Bounded Black Box Crash Testing



B^3 : Bounded Black Box Crash Testing

Choice of crash point

- Only after fsync(), fdatasync() or sync()
- Not in the middle of system call



- Developers are motivated to patch bugs that break semantics of persistence operations
- Crashing in the middle of system calls leads to exponentially large crash-states.

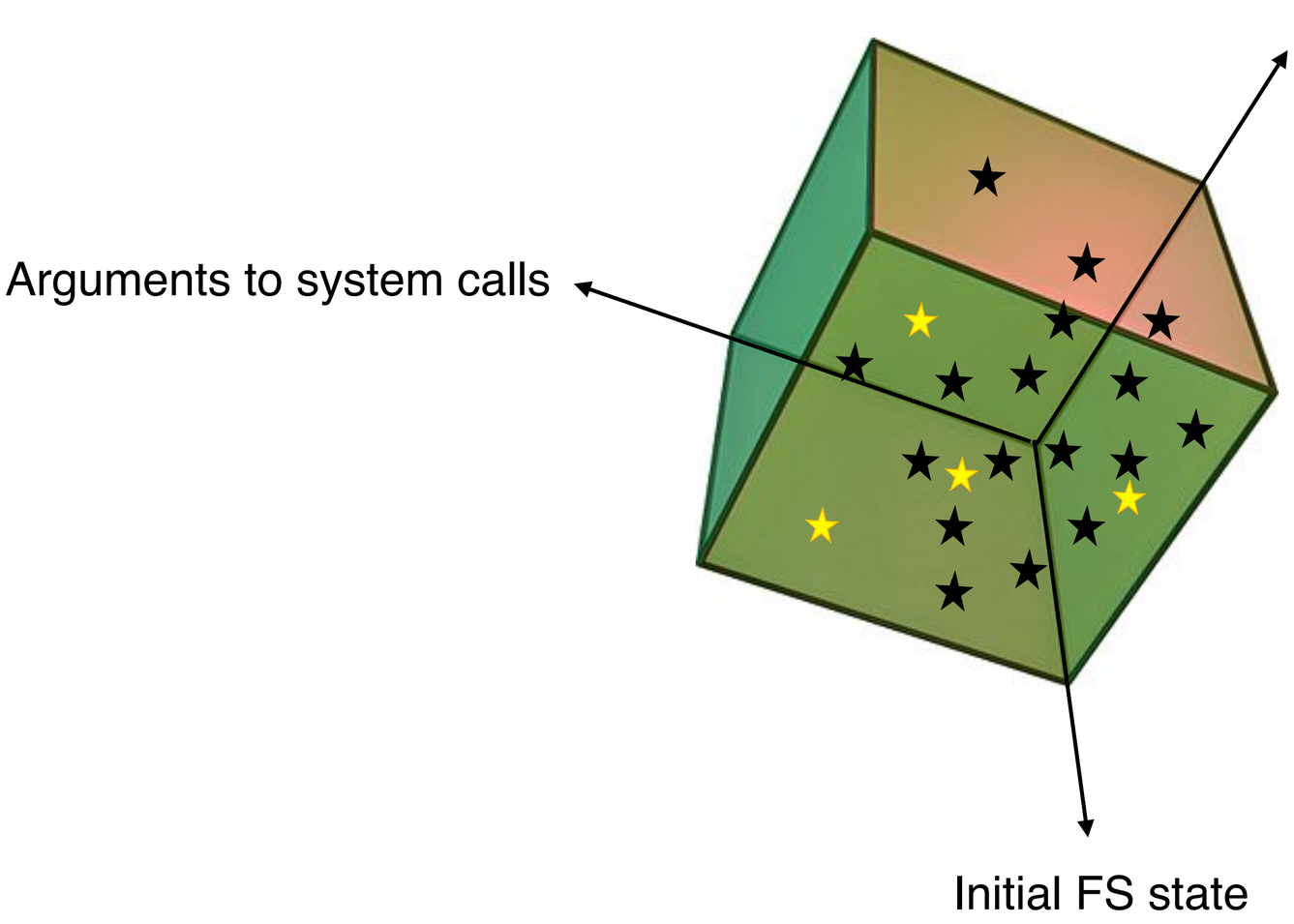
Limitations of B^3

- No guarantee of finding **all** crash-consistency bugs in a filesystem
- Assumes the correct working of crash-consistency mechanism like journaling or CoW
  - Does not crash in the middle of system calls
- Can only reveal if a bug has occurred, not the reason or origin of bug.
- Needs larger compute to test higher sequence lengths

Outline

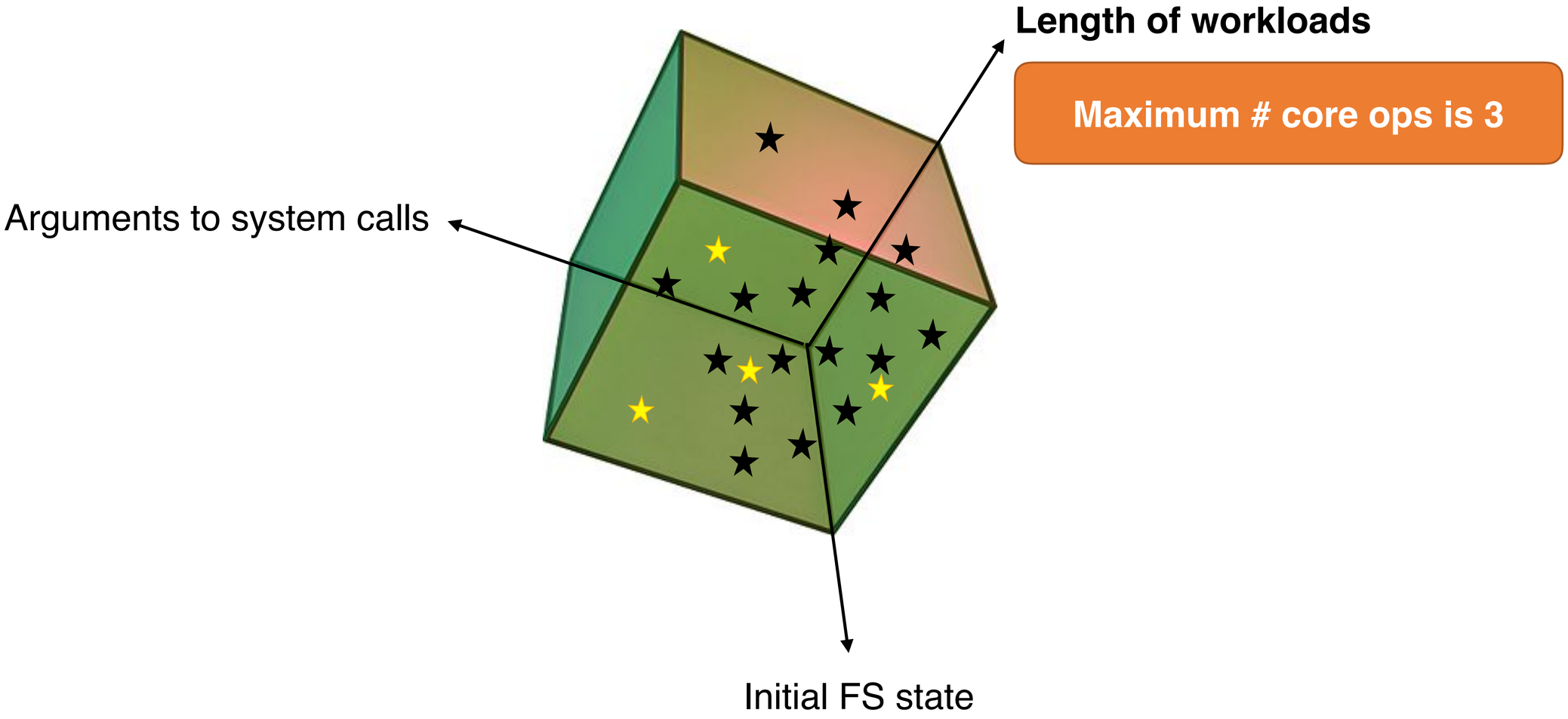
- CrashMonkey
- Bounded Black Box Crash Testing
- **Automatic Crash Explorer (ACE)**
- Demo

Bounds chosen by ACE

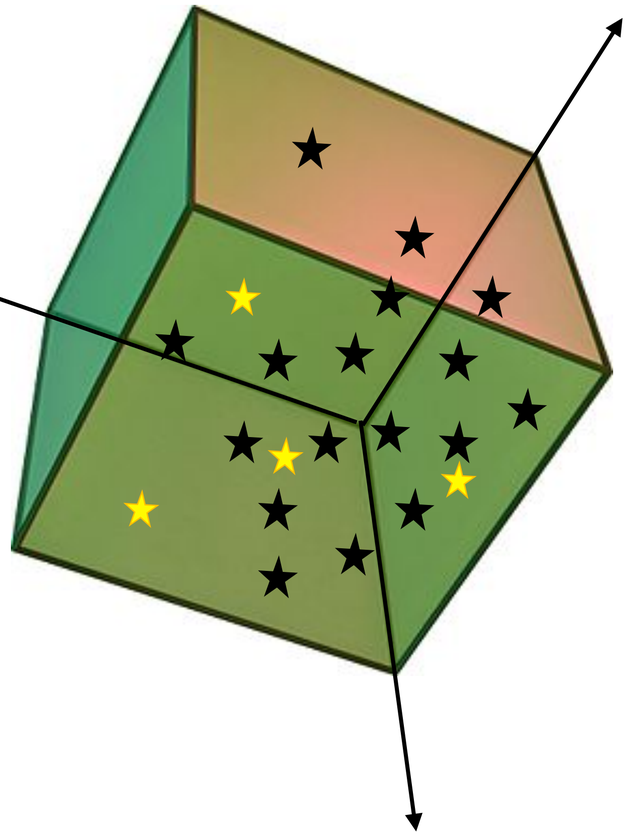


Bounds picked based on insights from the study of crash-consistency bugs reported on Linux file systems over the last 5 years

Bounds chosen by ACE



Bounds chosen by ACE



Length of workloads

Maximum # core ops is 3

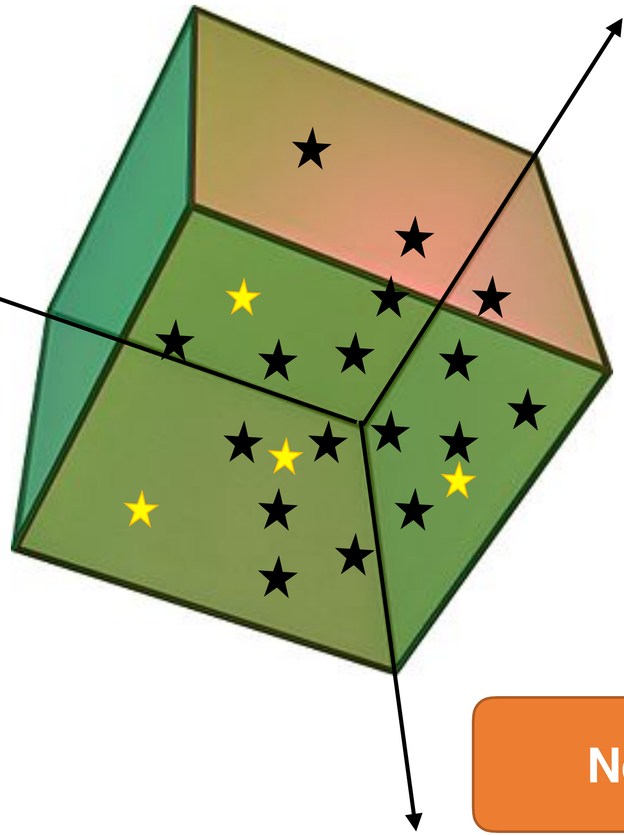
Arguments to system calls

Root
- **A → (foo, bar)**
- **B → (foo, bar)**

Overwrites to start, middle, end of a file and append

Initial FS state

Bounds chosen by ACE



Length of workloads

Maximum # core ops is 3

Arguments to system calls

Root
- **A → (foo, bar)**
- **B → (foo, bar)**

Overwrites to start, middle, end and append

New, 100MB FS

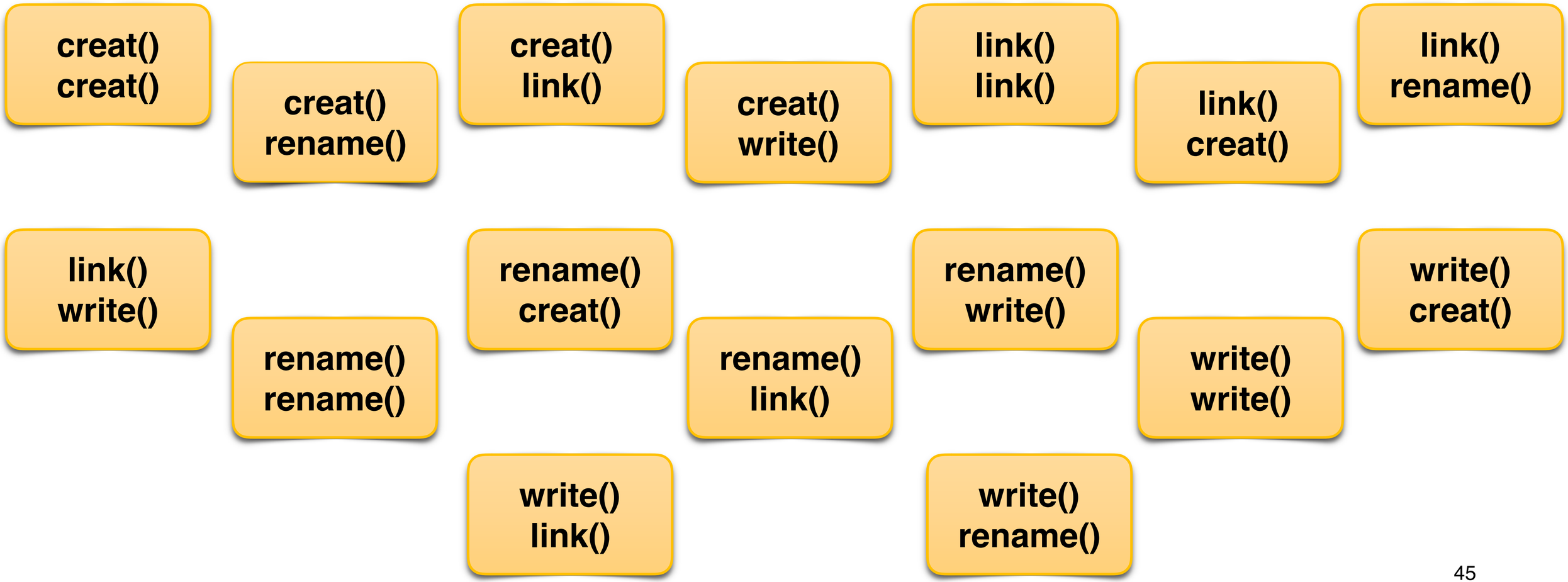
Initial FS state

Phases of ACE

Operation Set

creat()
link()
rename()
write()

Generating skeletons of sequence-2. : 4*4 = 16

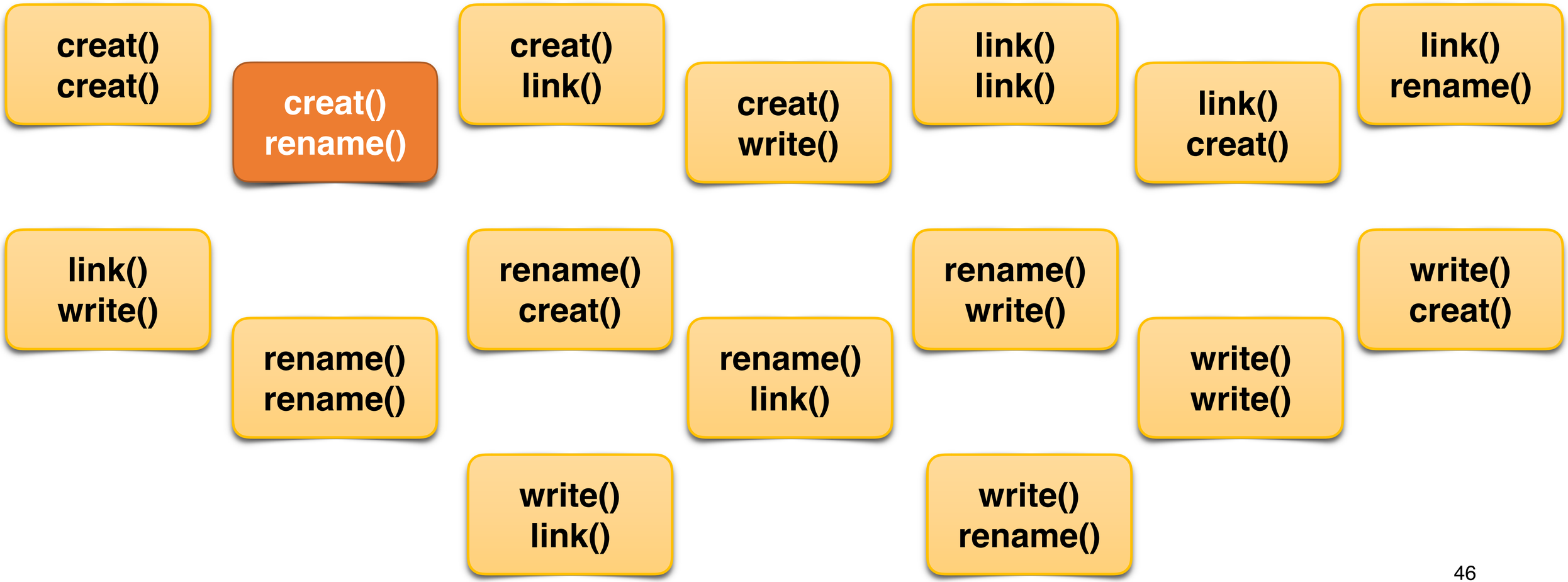


Phases of ACE

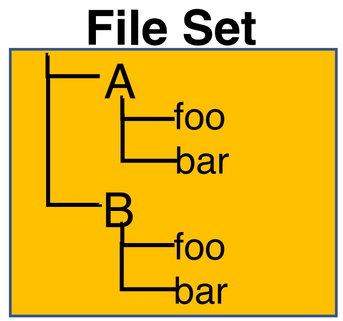
Operation Set

creat()
link()
rename()
write()

Generating skeletons of sequence-2. : 4*4 = 16



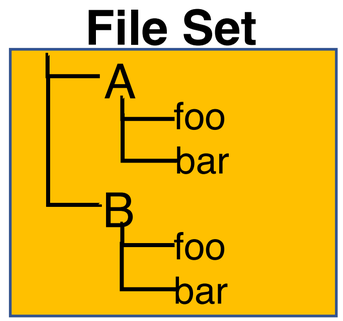
Phases of ACE



1. Select Operations

1. **creat()**
2. **rename()**

Phases of ACE



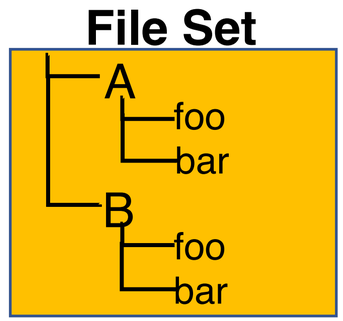
1. Select Operations

1. **creat()**
2. **rename()**

- If metadata operations, pick file or directory names
- If data operations, pick a range of offset and length

2. Select Parameters

Phases of ACE



1. Select Operations

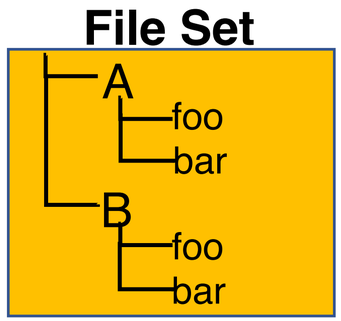
1. **creat()**
2. **rename()**

- If metadata operations, pick file or directory names
- If data operations, pick a range of offset and length

2. Select Parameters

1. **creat(A/bar)**
2. **rename(B/bar, A/bar)**

Phases of ACE



1. Select Operations

1. **creat()**
2. **rename()**

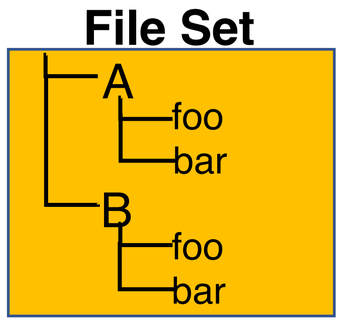
2. Select Parameters

1. **creat(A/bar)**
2. **rename(B/bar, A/bar)**

3. Add Persistence

- Between each core operation, add a persistence operation
- Consistency will be checked at these points
- Parameter to the persistence function is again chosen from the file/ directory pool

Phases of ACE



1. Select Operations

1. **creat()**
2. **rename()**

2. Select Parameters

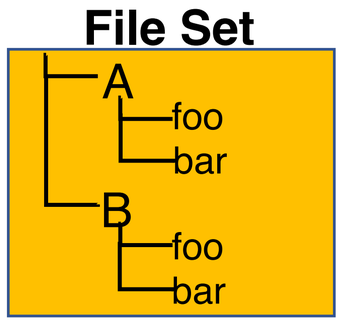
1. **creat(A/bar)**
2. **rename(B/bar, A/bar)**

3. Add Persistence

1. **creat(A/bar)**
   fsync(A/bar)
2. **rename(B/bar, A/bar)**
   fsync(A/foo)

- Between each core operation, add a persistence operation
- Consistency will be checked at these points
- Parameter to the persistence function is again chosen from the file/ directory pool

Phases of ACE



1. Select Operations

1. **creat()**
2. **rename()**

2. Select Parameters

1. **creat(A/bar)**
2. **rename(B/bar, A/bar)**

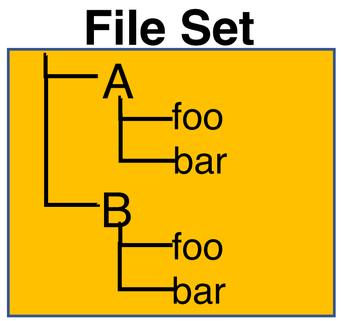
3. Add Persistence

1. **creat(A/bar)**
   fsync(A/bar)
2. **rename(B/bar, A/bar)**
   fsync(A/foo)

4. Add Dependencies

- Add file create/open/close to ensure the workload executes on any POSIX compliant filesystem.

Phases of ACE



1. Select Operations

1. creat()
2. rename()

2. Select Parameters

1. creat(A/bar)
2. rename(B/bar, A/bar)

3. Add Persistence

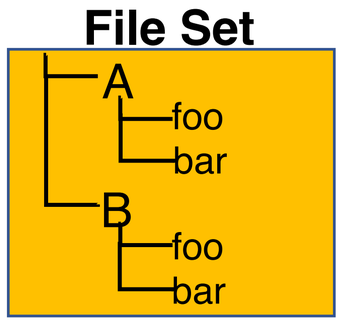
1. creat(A/bar)
   fsync(A/bar)
2. rename(B/bar, A/bar)
   fsync(A/foo)

- Add file create/open/close to ensure the workload executes on any POSIX compliant filesystem.

4. Add Dependencies

mkdir(A)
1. creat(A/bar)
   fsync(A/bar)
   mkdir(B)
   creat(B/bar)
2. rename(B/bar, A/bar)
   creat(A/foo)
   fsync(A/foo)
   close(A/foo)

Phases of ACE



1. Select Operations

1. creat()
2. rename()

2. Select Parameters

1. creat(A/bar)
2. rename(B/bar, A/bar)

This workload with 2 core operations is the same workload required to trigger rename atomicity bug!

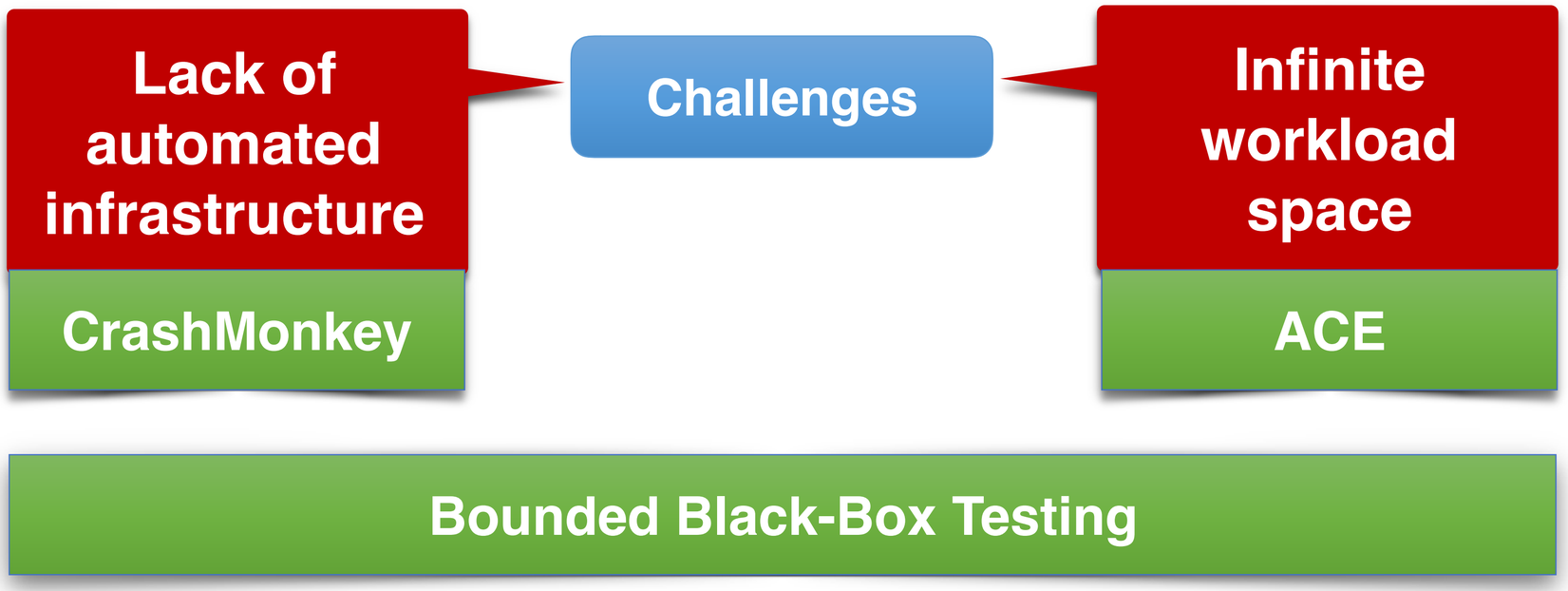
3. Add Persistence

1. creat(A/bar)
   fsync(A/bar)
2. rename(B/bar, A/bar)
   fsync(A/foo)

4. Add Dependencies

   mkdir(A)
1. creat(A/bar)
   fsync(A/bar)
   mkdir(B)
   creat(B/bar)
2. rename(B/bar, A/bar)
   creat(A/foo)
   fsync(A/foo)
   close(A/foo)

Challenges with Systematic Testing



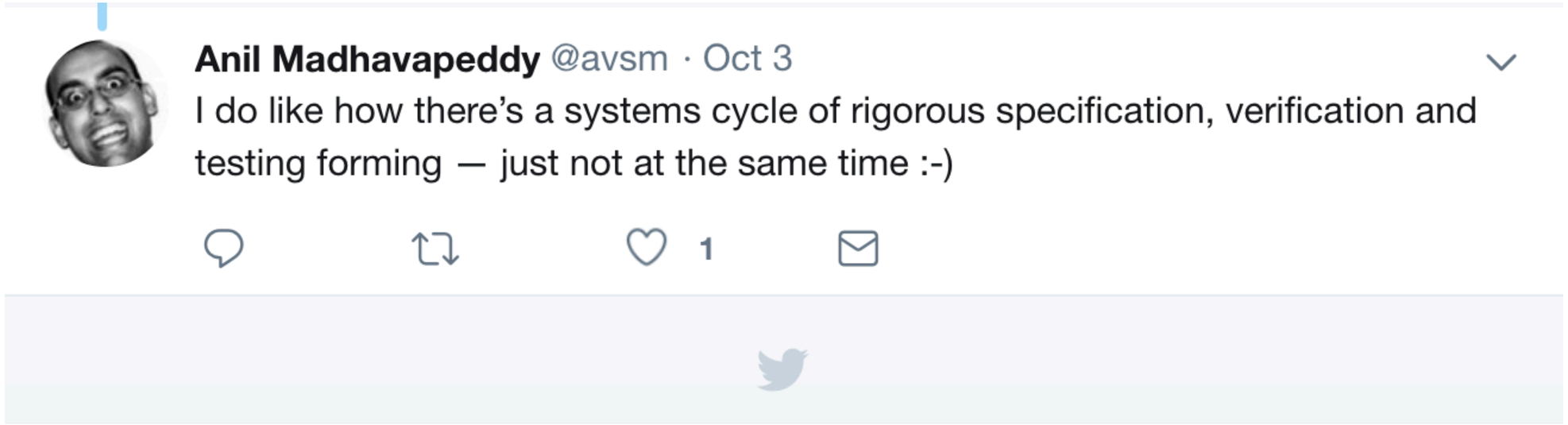
Results

- Reproduced 24/26 known bugs across ext4, btrfs and F2FS
- Found 10 new bugs across btrfs and F2FS
- Found 1 bug in a verified file system, FSCQ

Outline

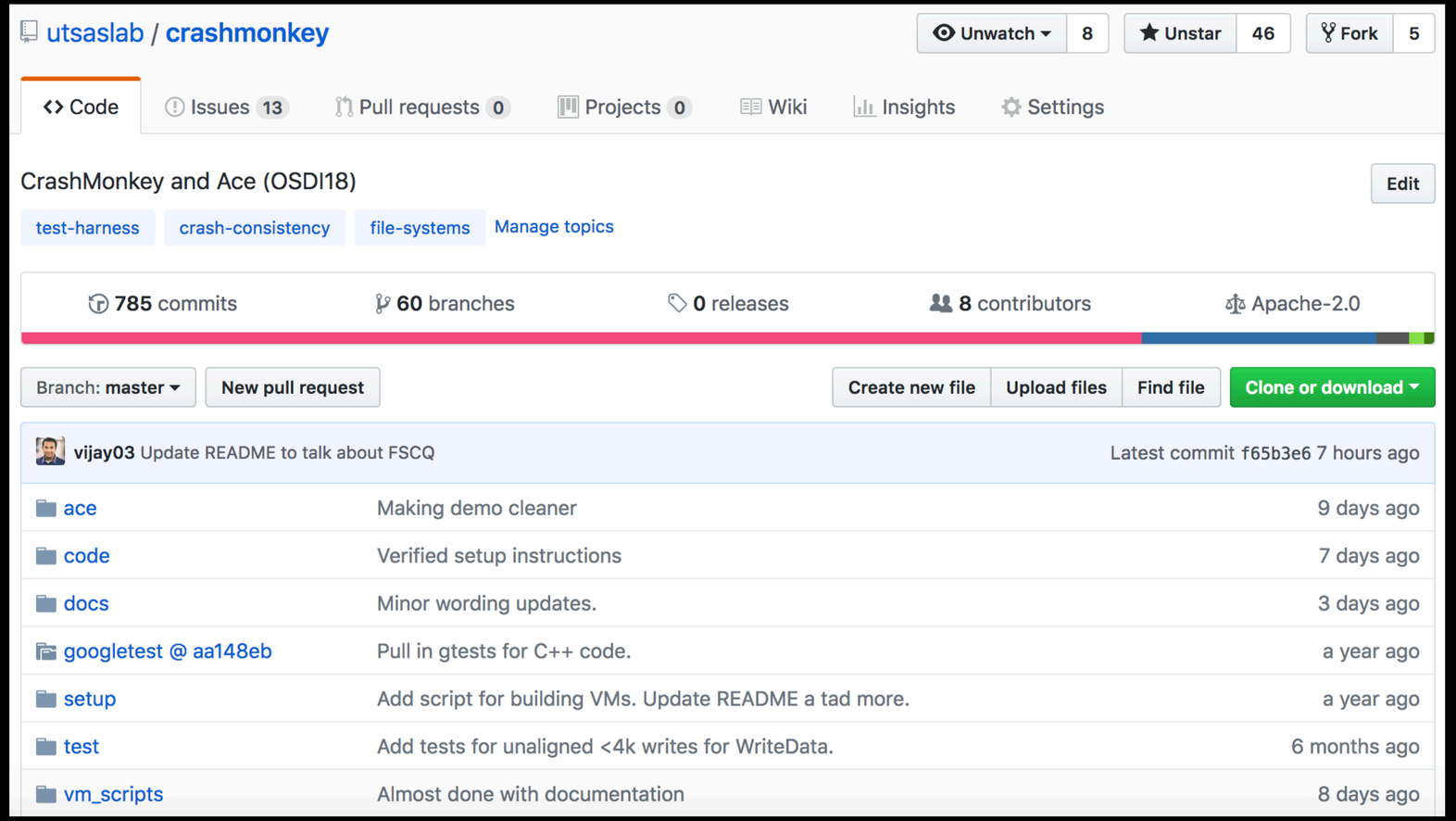
- CrashMonkey
- Bounded Black Box Crash Testing
- Automatic Crash Explorer (ACE)
- **Demo**

Testing, specification, and verification



Bounded Black-Box Crash Testing (Poster #4)

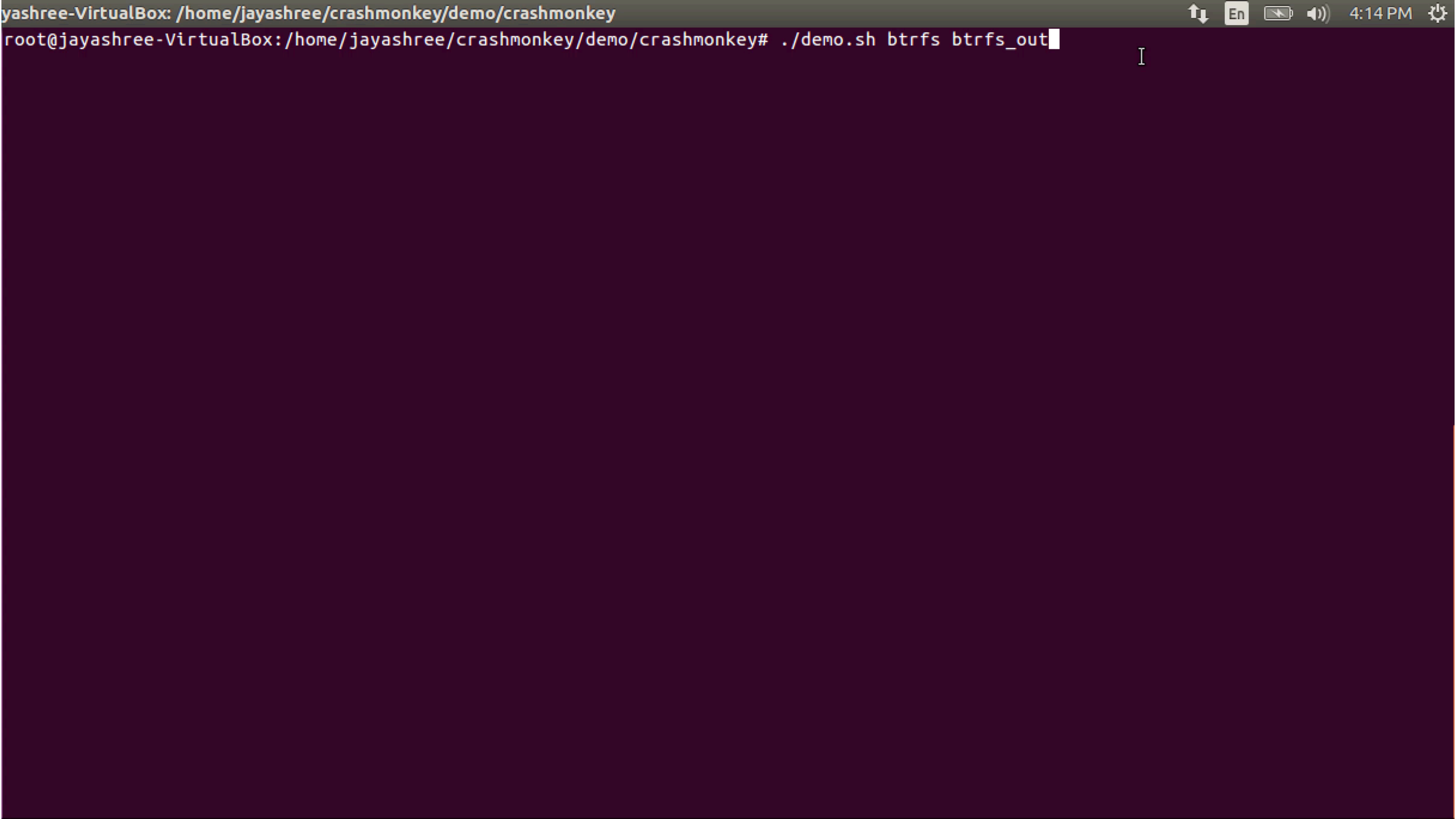
- B³ makes exhaustive testing feasible using informed bound selection
- Easily generalizable to test larger workloads if more compute is available
- Found 10 new bugs across btrfs and F2FS, most of which existed since 2014
- Found 1 bug in FSCQ



Thanks!
Questions?

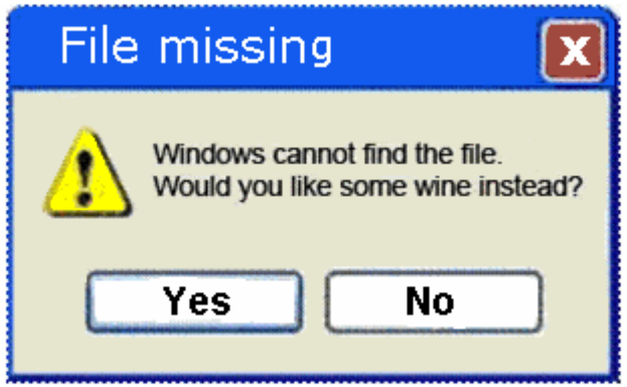
Try our tools : https://github.com/utsaslab/crashmonkey

Backup slides


yashree-VirtualBox: /home/jayashree/crashmonkey/demo/crashmonkey
En
4:14 PM
root@jayashree-VirtualBox:/home/jayashree/crashmonkey/demo/crashmonkey# ./demo.sh btrfs btrfs_out

Crash Consistency

- Filesystem operations change multiple blocks on storage that needs to be ordered
  - Inode, bitmaps, data blocks, superblock
  - Data and metadata must be consistent on a crash

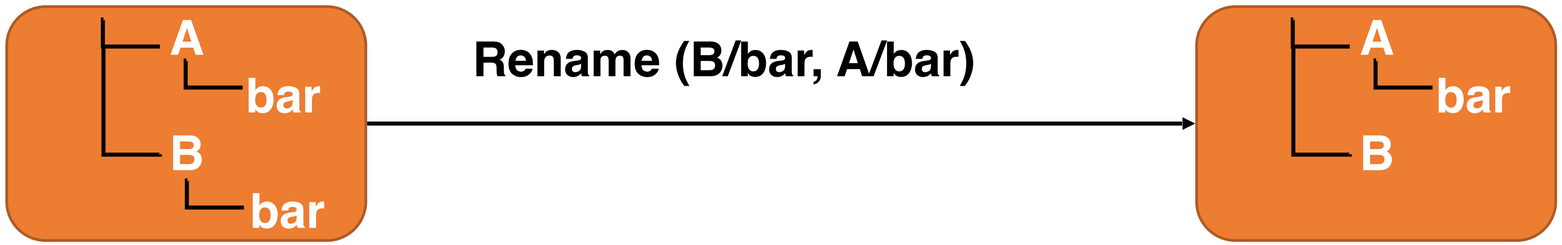


Metadata Corruption

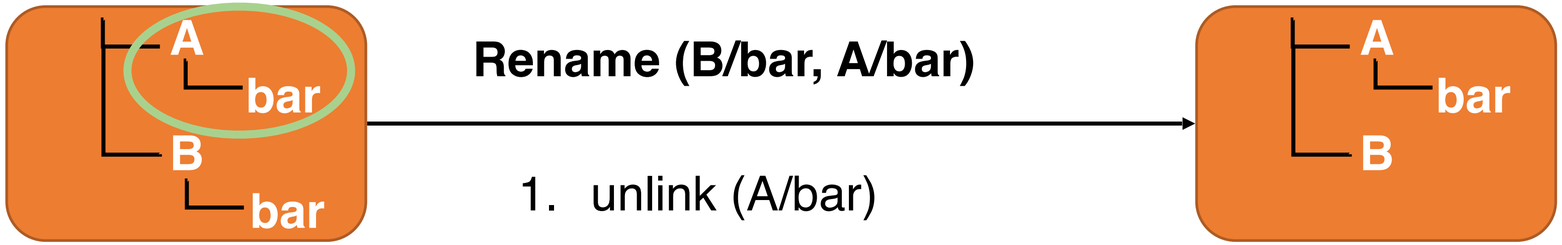
Data Corruption

Unmountable FS

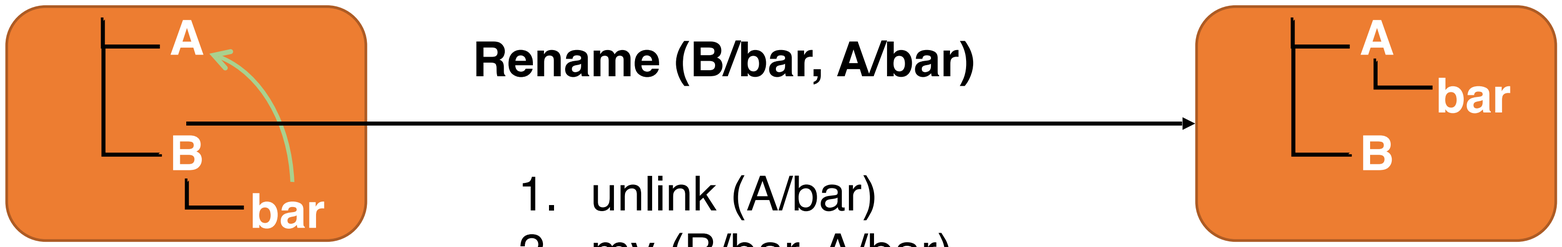
What just happened?



What just happened?

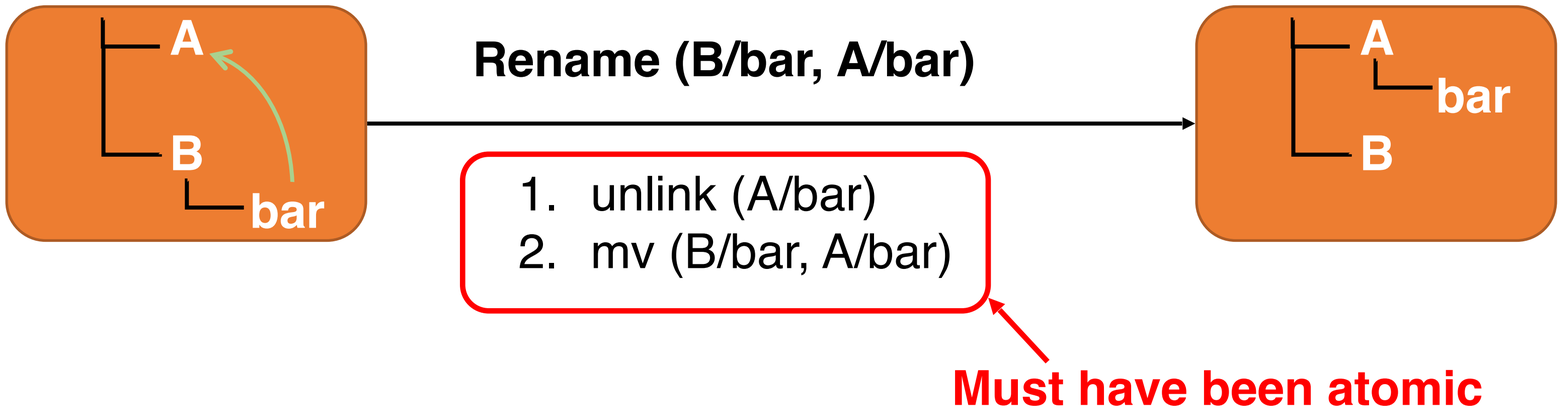


What just happened?



1. unlink (A/bar)
2. mv (B/bar, A/bar)

What just happened?



mkdir (A)
touch (A/bar)
fsync (A/bar)
mkdir (B)
touch (B/bar)
rename (B/bar, A/bar)
touch (A/foo)
fsync (A/foo)
CRASH!

- fsync(A/foo) commits tx that unlinks A/bar
- Which means step 1 above is persisted, but rename is not persisted
- **End up losing file A/bar**
- Exists in the kernel since 2014

Study of crash consistency bugs in the wild

- Study the workload pattern and impacts of crash consistency bugs reported in the **past 5 years**
  - Kernel mailing lists
  - Crash consistency tests submitted to xfstests

- **26 unique bugs** across ext4, F2FS, and btrfs

Study of crash consistency bugs in the wild

Consequence	# bugs
Corruption	17
Data inconsistency	6
Unmountable FS	3
Total	**26**

Filesystem	# bugs
Ext4	2
F2FS	2
btrfs	24
Total	**28**

# ops	# bugs
1	3
2	14
3	9
Total	**26**

Study of crash consistency bugs in the wild

Consequence	# bugs
Corruption	17
Data inconsistency	6
Unmountable FS	3
Total	**26**

Filesystem	# bugs
Ext4	2
F2FS	2
btrfs	24
Total	**28**

# ops	# bugs
1	3
2	14
3	9
Total	**26**

1. Crash consistency bugs are hard to find

- Bugs have been around in the kernel for up to 7 years before being identified and patched
- Usually involve reuse of files/ directories

Study of crash consistency bugs in the wild

Consequence	# bugs
Corruption	17
Data inconsistency	6
Unmountable FS	3
Total	**26**

Filesystem	# bugs
Ext4	2
F2FS	2
btrfs	24
Total	**28**

# ops	# bugs
1	3
2	14
3	9
Total	**26**

1. Crash consistency bugs are hard to find
2. **Small workloads are sufficient to reveal bugs**
   - 2-3 core operations on a new, empty file-system

Study of crash consistency bugs in the wild

Consequence	# bugs
Corruption	17
Data inconsistency	6
Unmountable FS	3
Total	**26**

Filesystem	# bugs
Ext4	2
F2FS	2
btrfs	24
Total	**28**

# ops	# bugs
1	3
2	14
3	9
Total	**26**

1. Crash consistency bugs are hard to find
2. Small workloads are sufficient to reveal bugs
3. **Crash after persistence points**
   - Sufficient to crash after a call to fsync(), fdatasync(), or sync()

Study of crash consistency bugs in the wild

Consequence	# bugs
Corruption	17
Data inconsistency	6
Unmountable FS	3
Total	**26**

Filesystem	# bugs
Ext4	2
F2FS	2
btrfs	24
Total	**28**

# ops	# bugs
1	3
2	14
3	9
Total	**26**

1. Crash consistency bugs are hard to find
2. Small workloads are sufficient to reveal bugs
3. Crash after persistence points
4. **Systematic testing is required**

Study of crash consistency bugs in the wild

Consequence	# bugs
Corruption	17
Data inconsistency	6
Unmountable FS	3
Total	**26**

Filesystem	# bugs
Ext4	2
F2FS	2
btrfs	24
Total	**28**

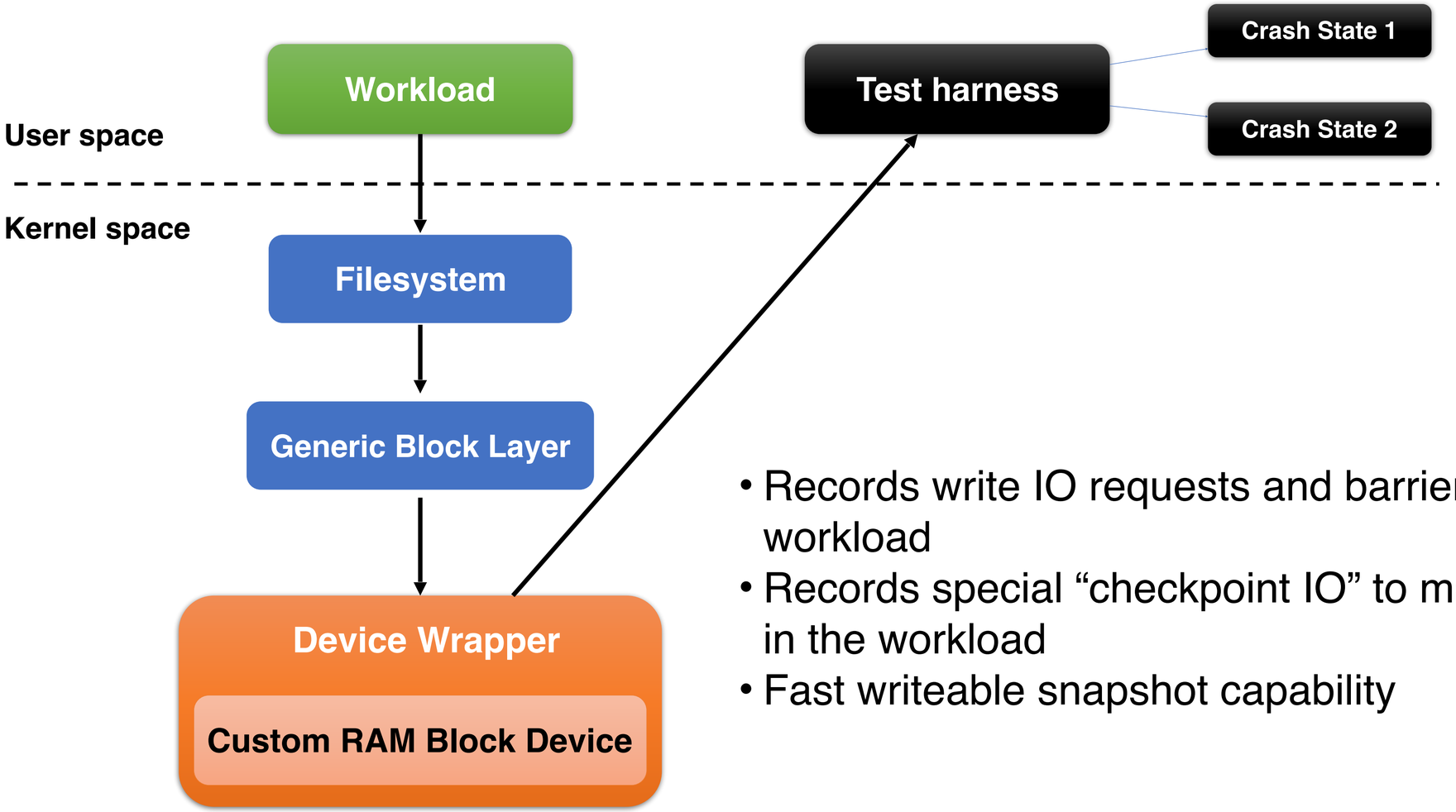
# ops	# bugs
1	3
2	14
3	9
Total	**26**

1. Crash consistency bugs are hard to find
2. Small workloads are sufficient to reveal bugs
3. Crash after persistence points
4. **Systematic testing is required**

Fallocate : punch_hole : 2015

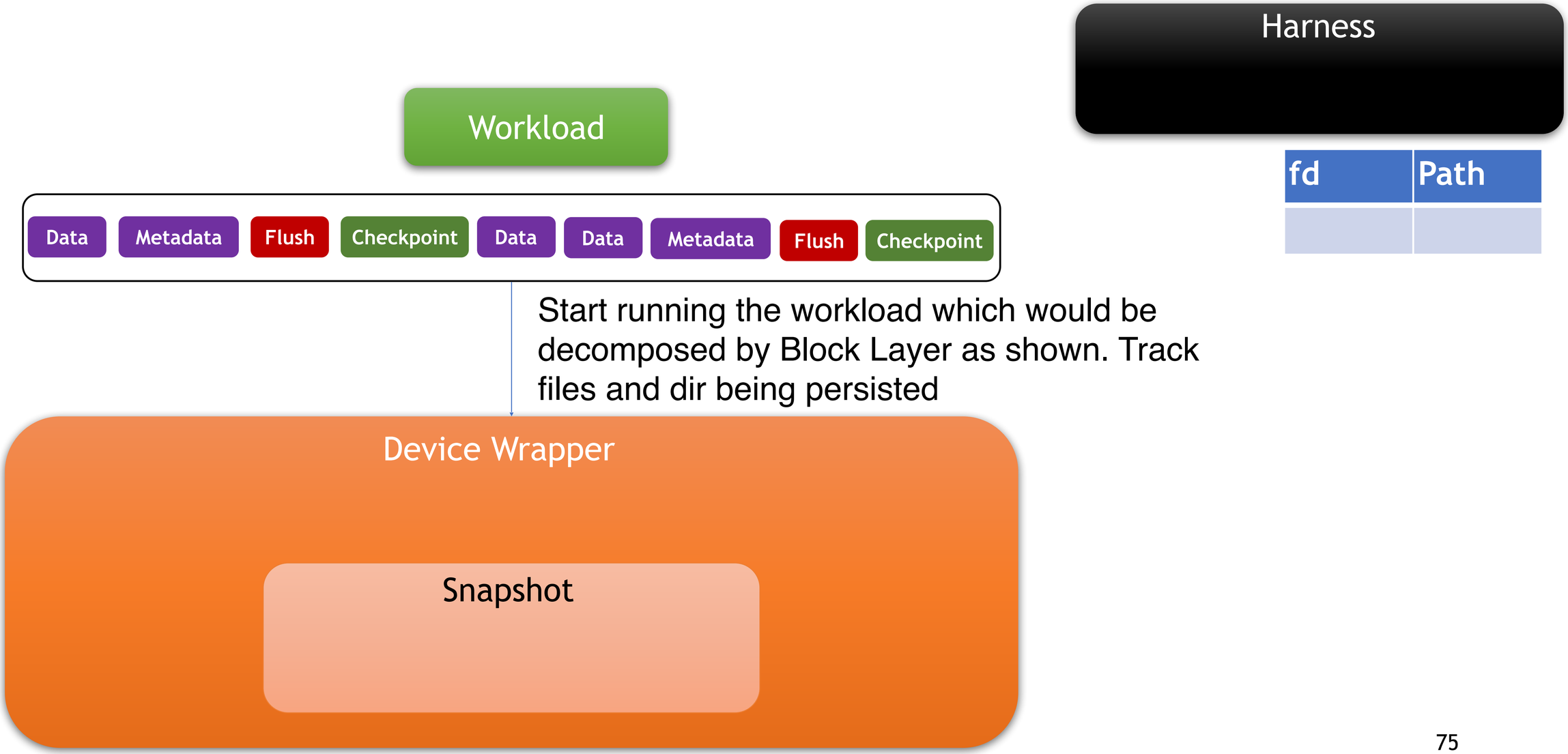
Fallocate : zero_range : 2018

CrashMonkey Internals

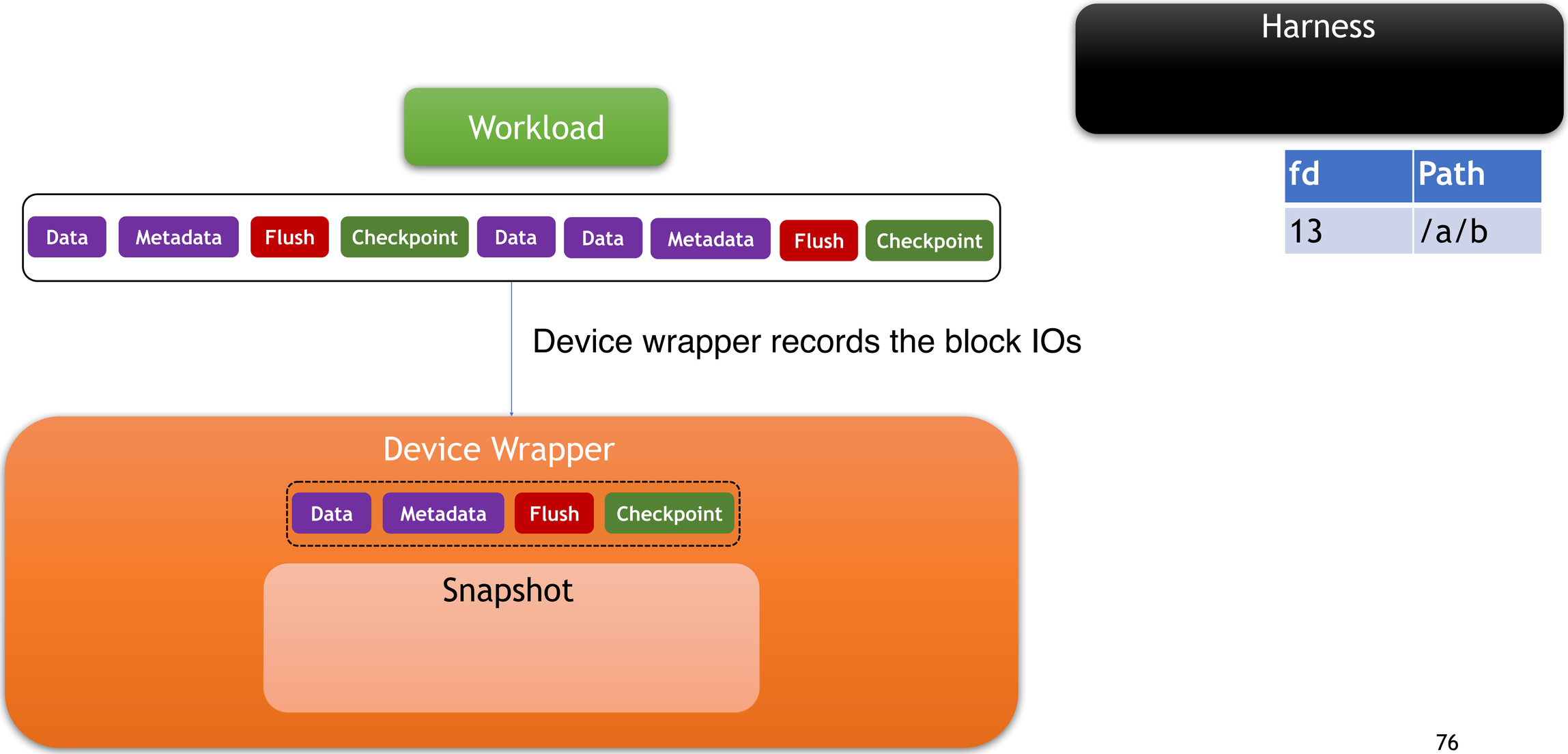


- Records write IO requests and barriers (flush/FUA) in the workload
- Records special “checkpoint IO” to mark persistence points in the workload
- Fast writeable snapshot capability

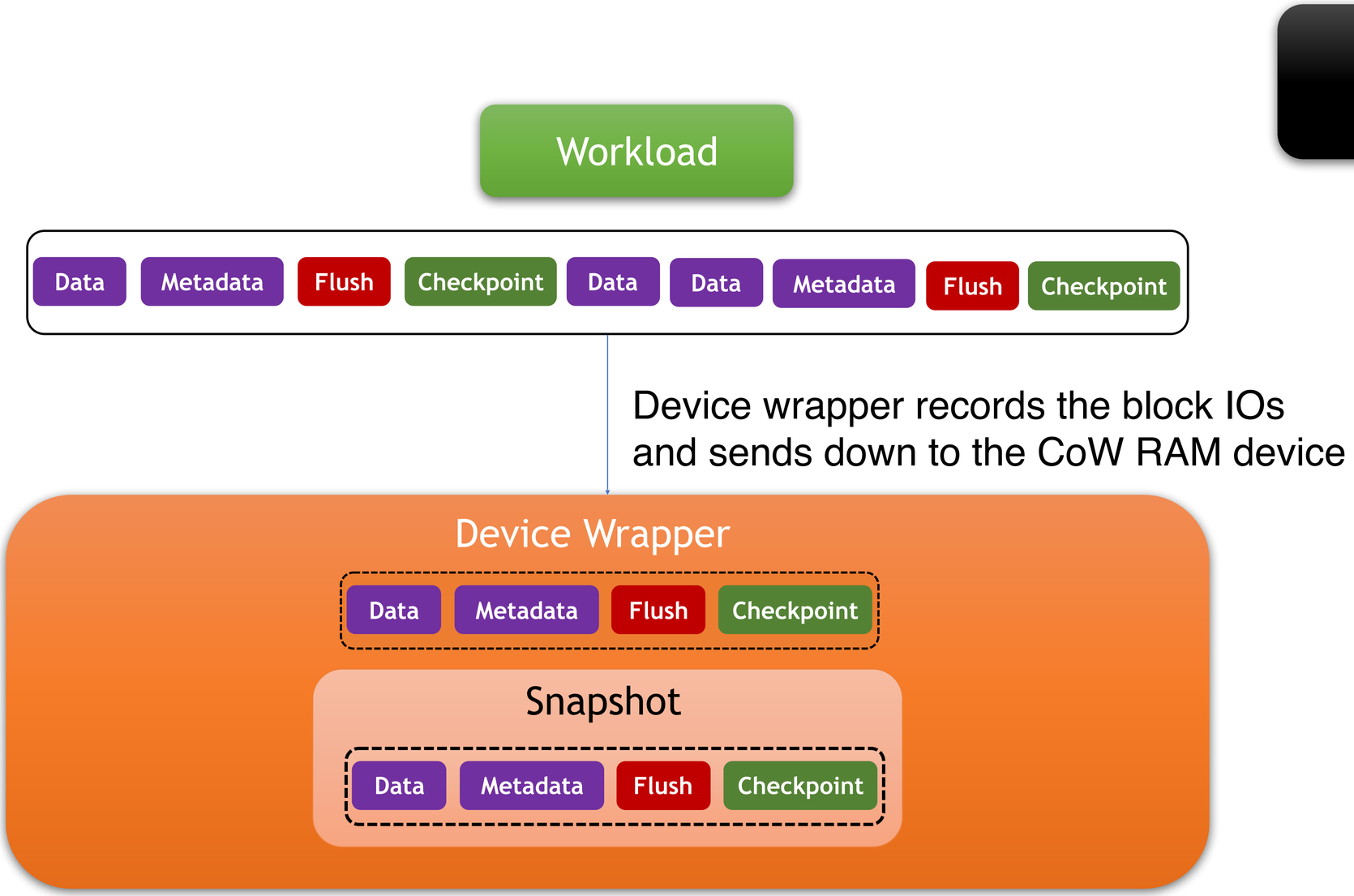
CrashMonkey in Action



CrashMonkey in Action : Profiling

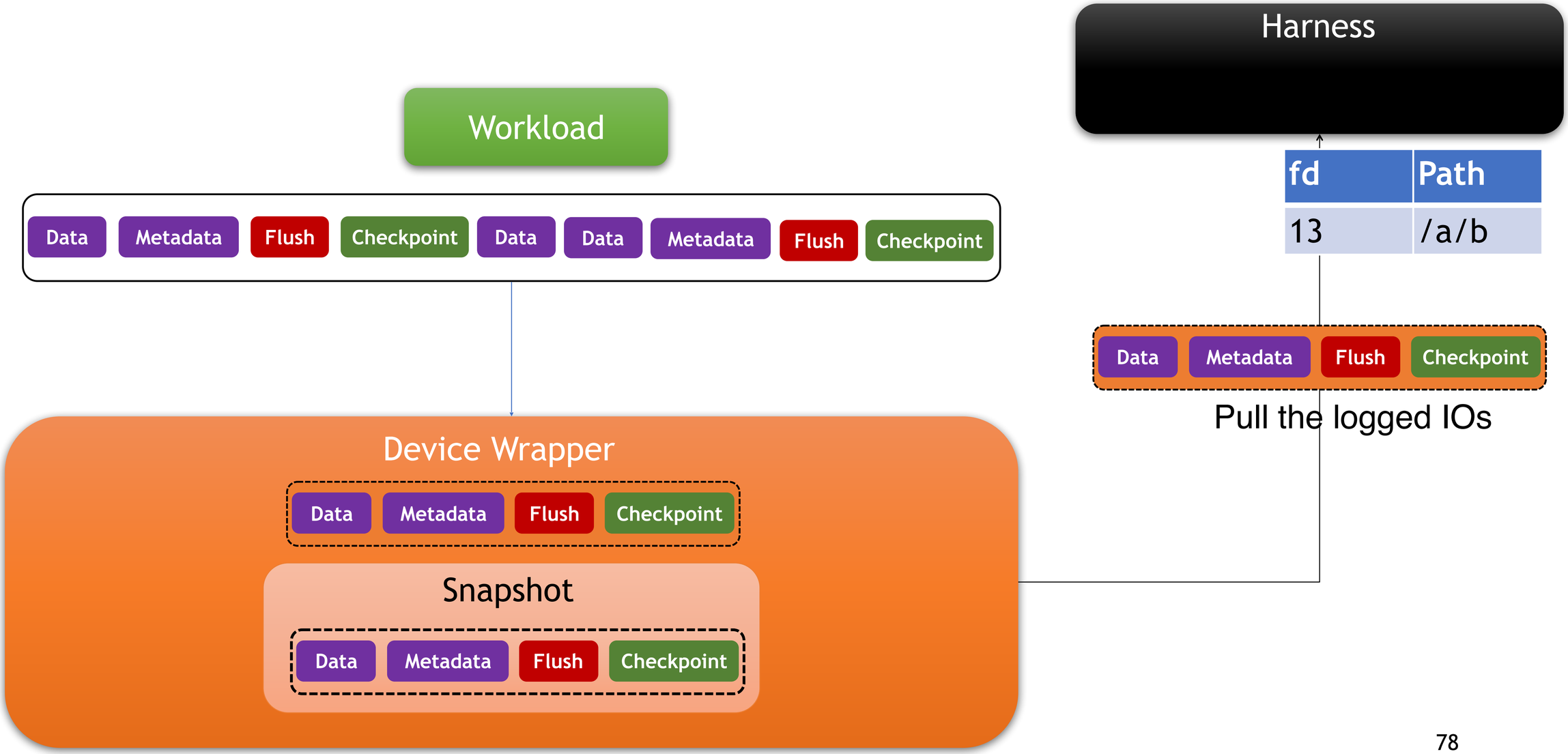


CrashMonkey in Action : Profiling

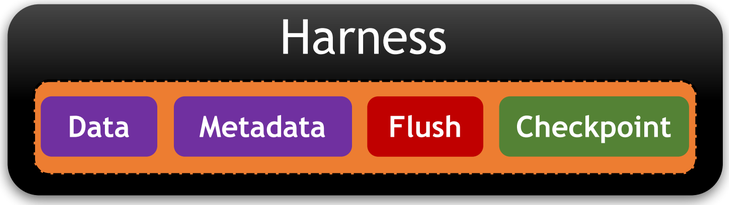


fd	Path
13	/a/b

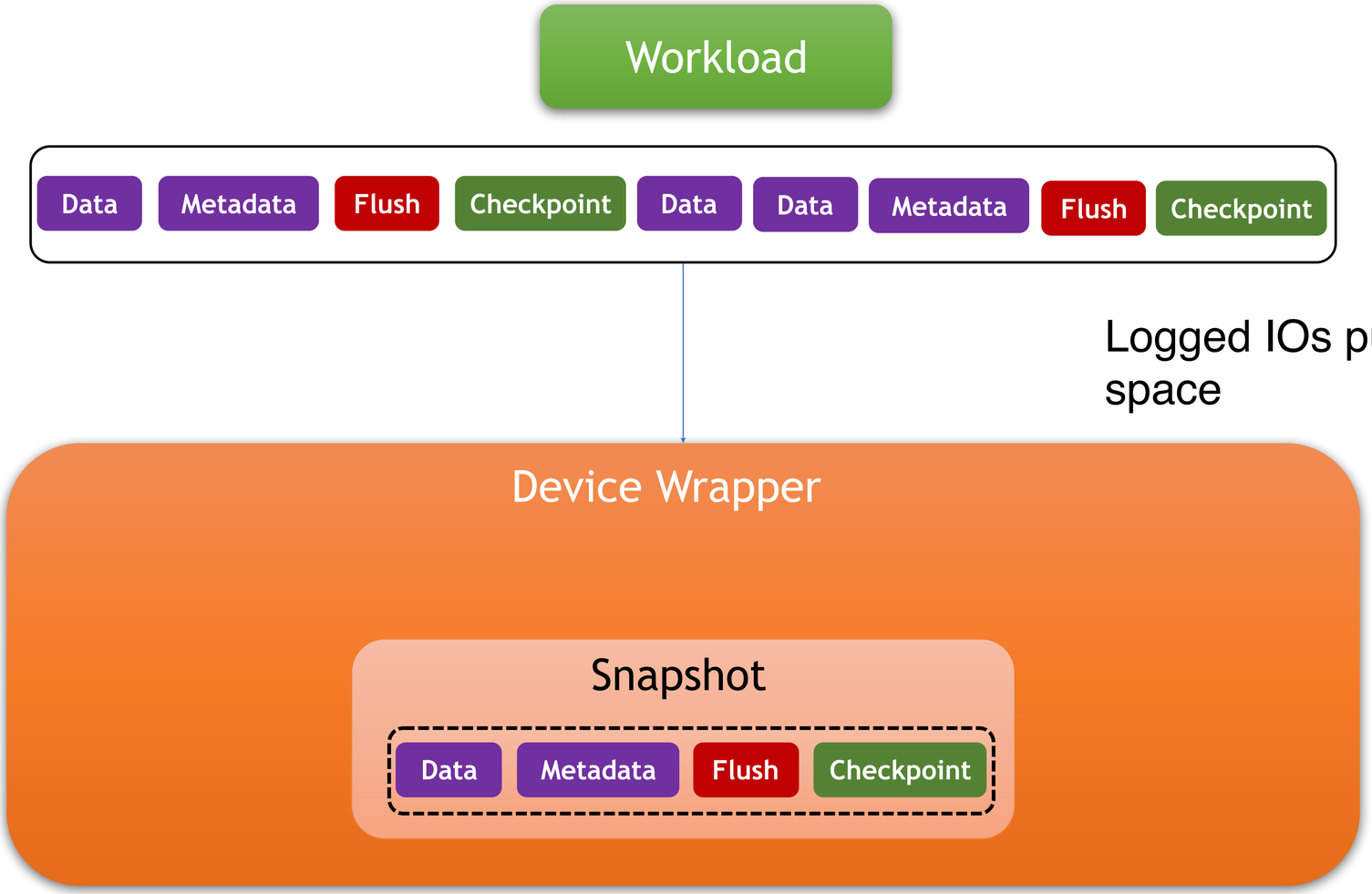
CrashMonkey in Action : Profiling



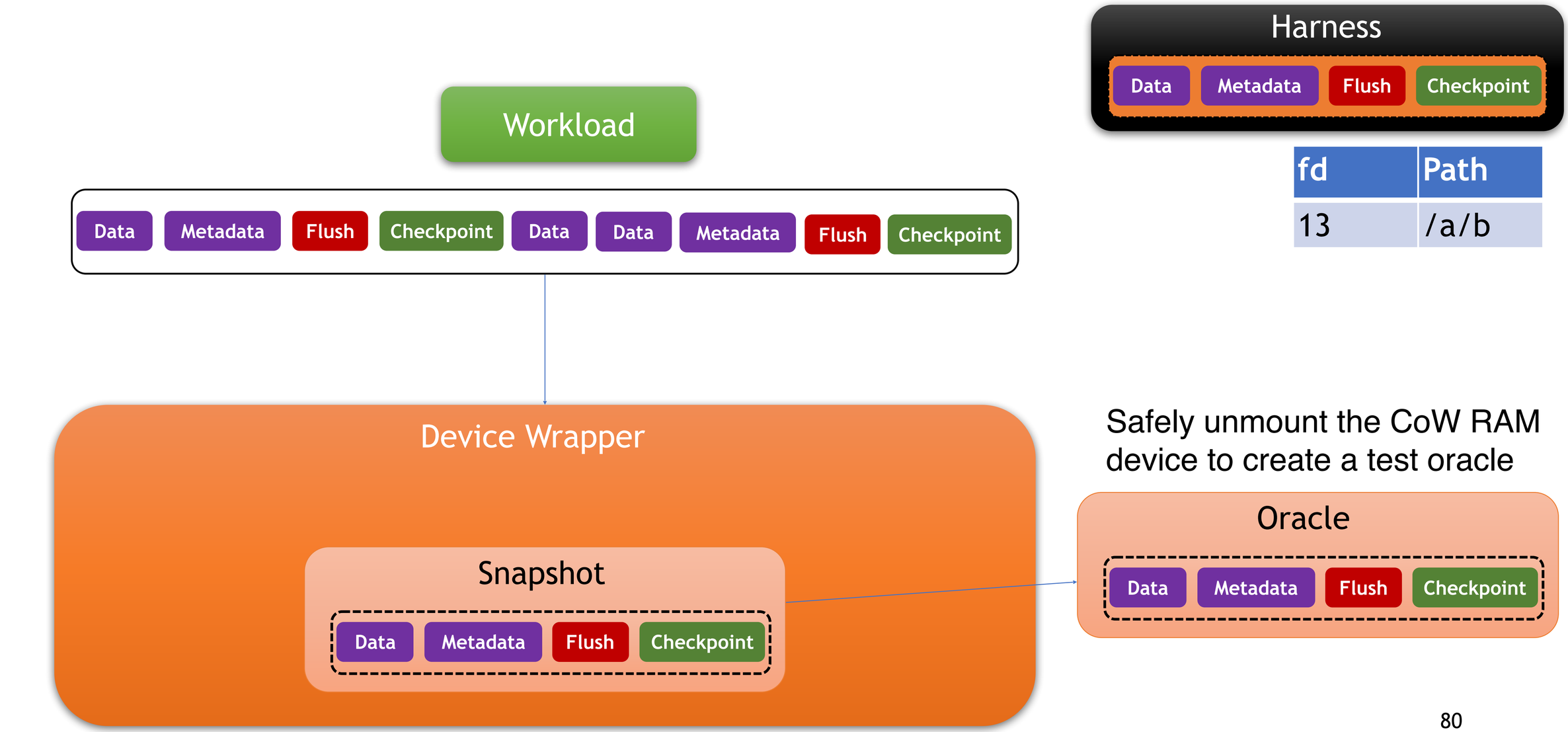
CrashMonkey in Action : Profiling



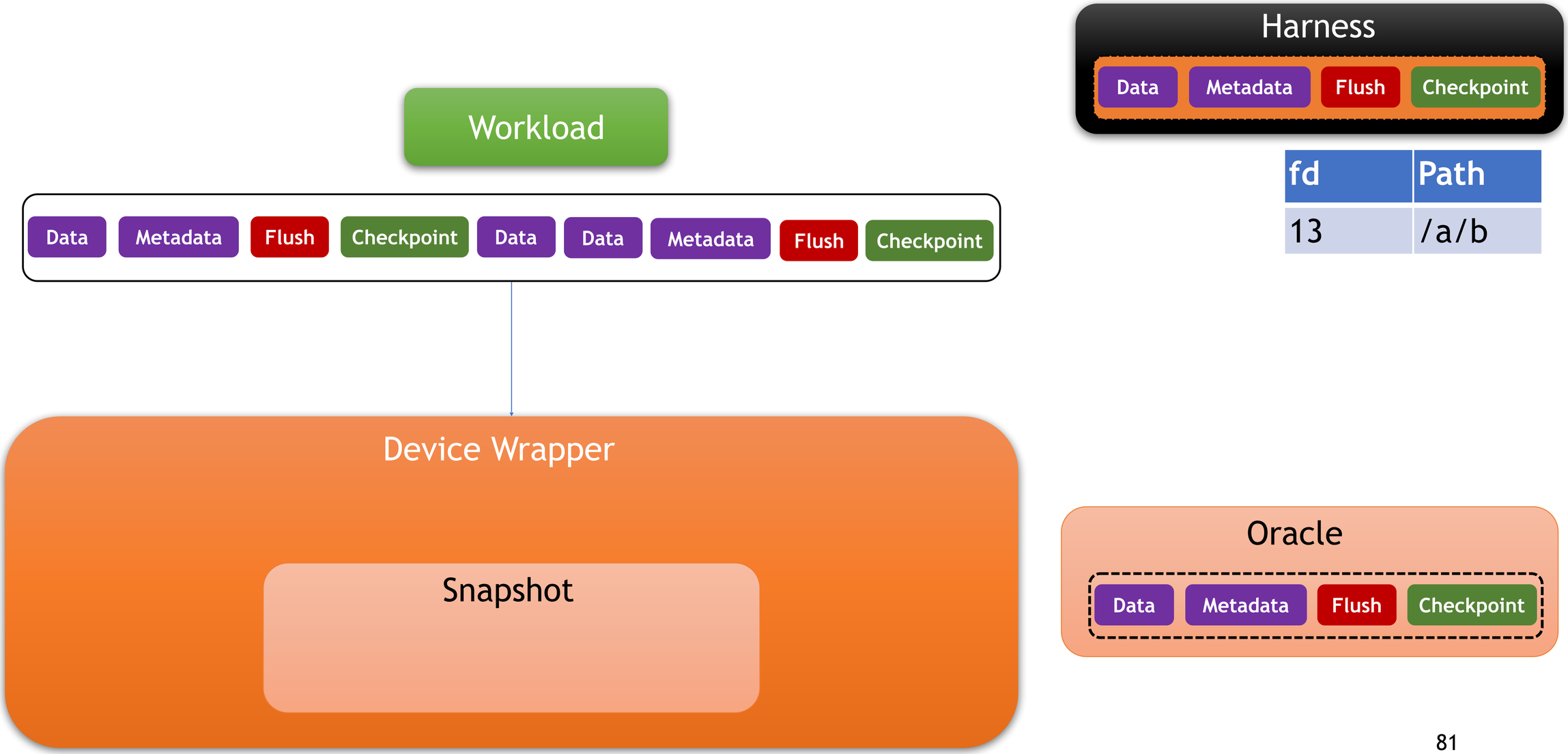
fd	Path
13	/a/b



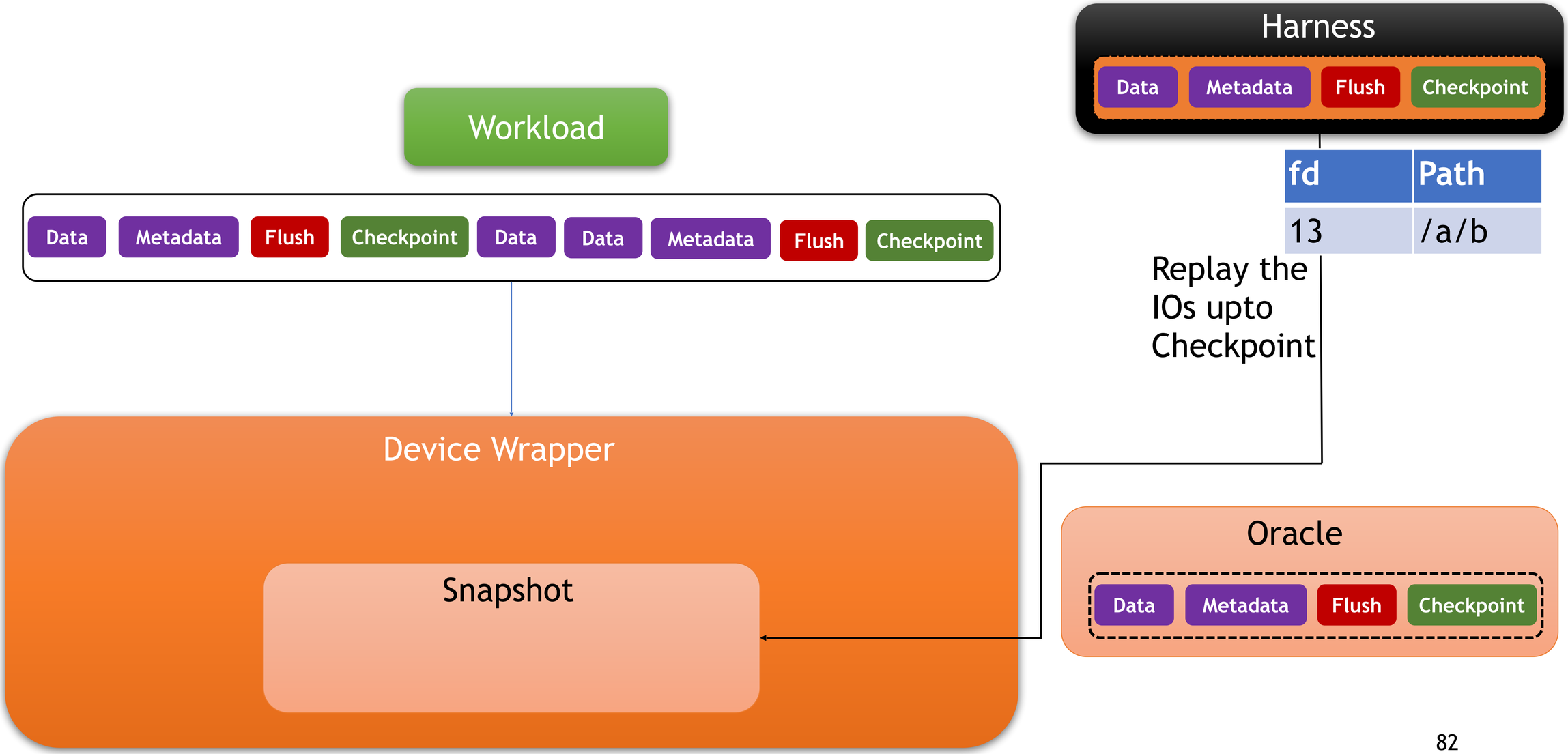
CrashMonkey in Action : Profiling



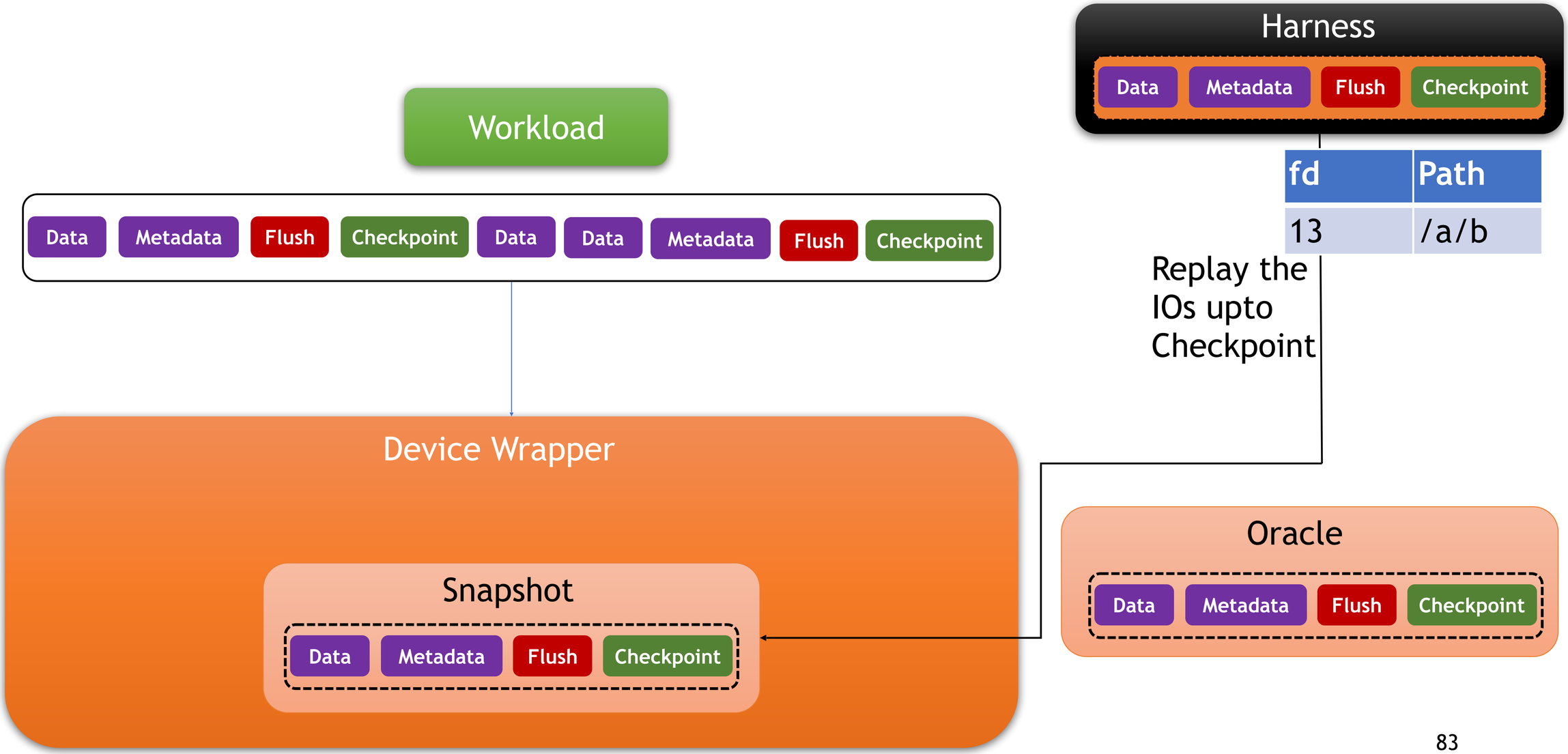
CrashMonkey in Action : Profiling



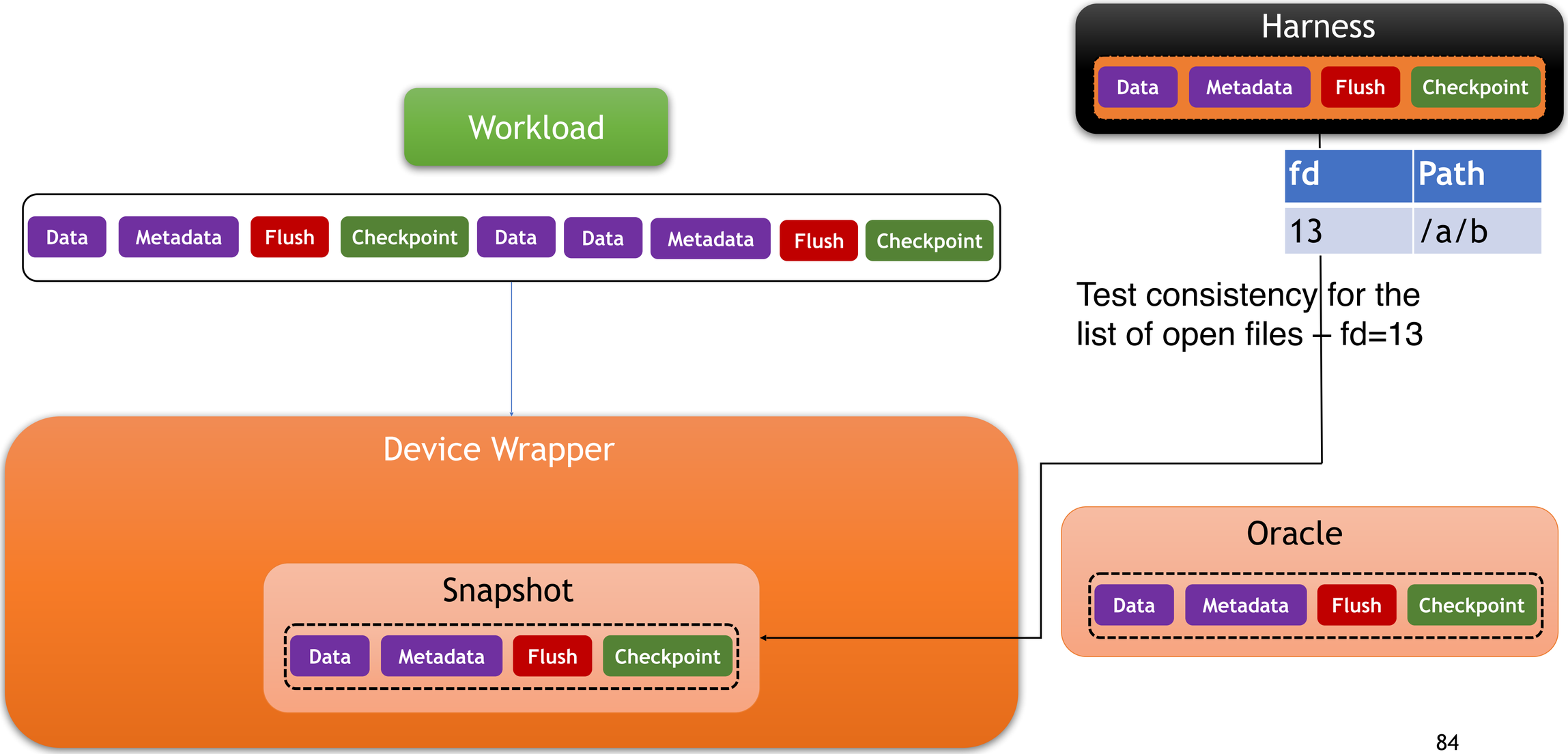
CrashMonkey in Action : Replay



CrashMonkey in Action : Replay



CrashMonkey in Action : Testing



Testing Strategy to find new bugs

- We test seq-1, seq-2 workloads on all filesystems : ext4, xfs, f2fs, btrfs
- We run all other workloads on btrfs and F2FS first.
  - For every workload that generated a bug, we run it on all other FS

- To run all workloads upto seq-3, you need to dedicate 2 days of compute per filesystem with (testing in parallel on 780 VM)

Results at a glance

Sequence Length	# workloads	# Bugs Reproduced	# Bugs found
Seq-1			
Seq-2			
Seq-3 metadata			
Seq-3 data			
Seq-3 nested			
Total			

- 25 million workloads
- Needs 15 days of testing on 780 VMs in parallel!

Results at a glance

Sequence Length	# workloads	# Bugs Reproduced	# Bugs found
Seq-1	300	3	3
Seq-2	254K	14	3
Seq-3 metadata	120K	5	2
Seq-3 data	1.5M	2	0
Seq-3 nested	1.5M	2	2
Total	3.37M	26	10