All File Systems Are Not Created Equal

Thanumalayan Sankaranarayana Pillai, Vijay Chidambaram, Ramnatthan Alagappan,

Samer Al-Kiswany, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau

All File Systems Are Not Created Equal: On the Complexity of Crafting Crash-Consistent Applications

Thanumalayan Sankaranarayana Pillai, Vijay Chidambaram, Ramnatthan Alagappan,

Samer Al-Kiswany, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau

Crash Consistency

Maintaining data invariants across a system crash

- Example: Database transactions should be atomic

Crash Consistency

Maintaining data invariants across a system crash

- Example: Database transactions should be atomic

Important in systems

- File Systems
- Relational Databases
- Key-Value Stores

Crash Consistency

Maintaining data invariants across a system crash

- Example: Database transactions should be atomic

Important in systems

- File Systems
- Relational Databases
- Key-Value Stores

Hard to get right: ARIES invented only in 1992

- Proving ARIES took 5 more years (1997)

File-System Crash Consistency

Lots of work in *file system* crash consistency

- Journaling, copy-on-write, soft updates …

File-System Crash Consistency

Lots of work in *file system* crash consistency

- Journaling, copy-on-write, soft updates …

FS consistency focuses on internal metadata

- Do directories only contain valid directory entries?

File-System Crash Consistency

Lots of work in *file system* crash consistency

- Journaling, copy-on-write, soft updates …

FS consistency focuses on internal metadata

- Do directories only contain valid directory entries?

What about user-level data?

This work studies …

What guarantees do file systems give applications?

- That can be used for consistency of user-level data

This work studies …

What guarantees do file systems give applications?

- That can be used for consistency of user-level data

Do applications maintain consistency correctly?

- Important applications require user-level consistency
- Databases, key-value stores, distributed systems …

We find ...

File system guarantees vary widely

- Studied 16 configs of *ext2,ext3,ext4,btrfs,xfs,reiserfs*
- Guarantees vary among configs of same file system
- Guarantees often side-effects of FS implementation
- POSIX standards of guarantees, if any, are debated

We find ...

File system guarantees vary widely

- Studied 16 configs of *ext2,ext3,ext4,btrfs,xfs,reiserfs*
- Guarantees vary among configs of same file system
- Guarantees often side-effects of FS implementation
- POSIX standards of guarantees, if any, are debated

Applications depend on guarantees in subtle ways

- Studied 11 applications: Databases, Distributed systems, Virtualization software, Key-value stores, VCS
- 60 vulnerabilities under a weak file system model
- More than 30 vulnerabilities under *ext3, ext4, btrfs*

Outline

Introduction

An Example

BOB: Examining File System Behavior

ALICE: Examining Applications

Conclusion

Outline

Introduction

An Example

BOB: Examining File System Behavior

ALICE: Examining Applications

Conclusion

Toy Example: Overview

A file initially contains the string "a foo"

- Assume each character in "a foo" is a block of data

Task: Atomically change the contents to "a bar"

- On a power loss, we must retrieve either "a foo" or "a bar"

Toy Example: Simple Overwrite

Initial state

```
/x/f1    "a foo"
```

Modification

pwrite(/x/f1, 2, "bar")

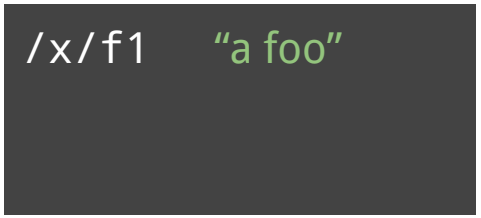
Final state

```
/x/f1    "a bar"
```

Toy Example: Simple Overwrite

Intermediate states possible on crash

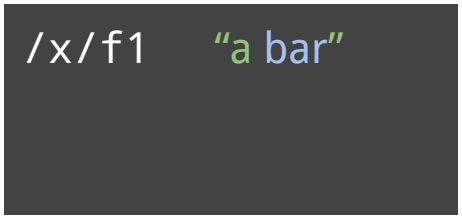
Initial state



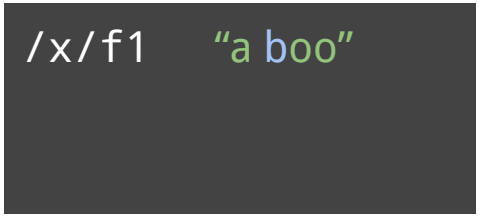
Modification

pwrite(/x/f1, 2, "bar")

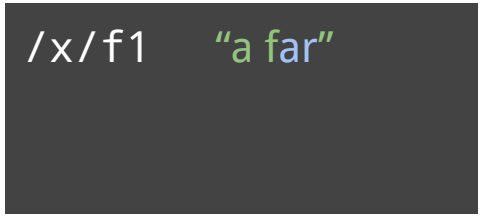
Final state



Intermediate state 1



Intermediate state 2



Intermediate states 3, 4, 5 ....

Toy Example: Maintaining Consistency

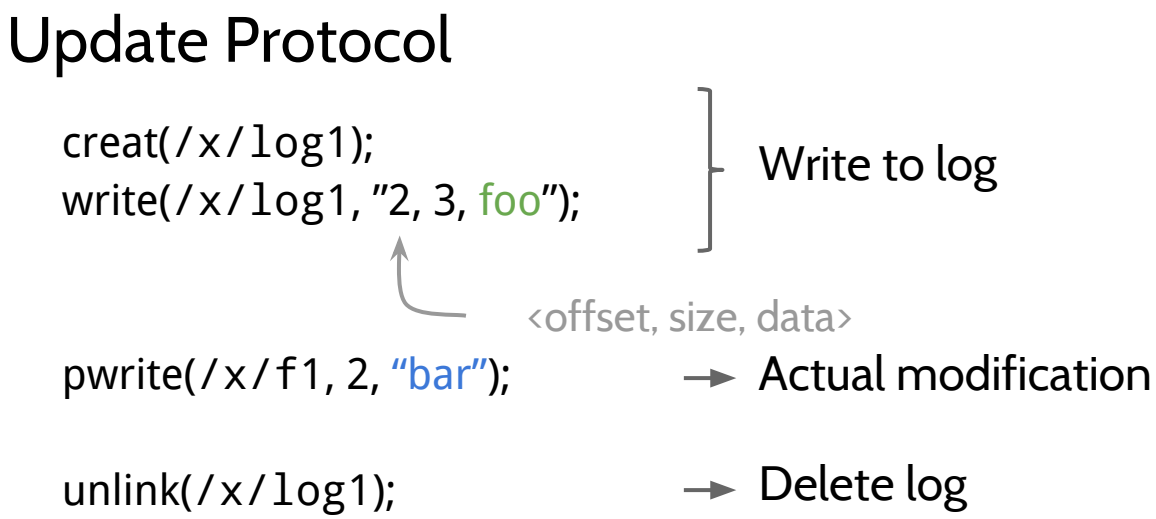
What if crash atomicity is needed?

Use application-level logging (a.k.a. *undo logging/rollback journaling*)

a. Make a copy of old data in "log" file
b. Modify actual file
c. Delete log file
d. On a crash, data can be recovered from the log

Toy Example: Protocol #1

What if crash atomicity is needed?



Toy Example: Protocol #1

Works in ext3(*data-journal*)!

Update Protocol

```
creat(/x/log1);
write(/x/log1, "2, 3, foo");

pwrite(/x/f1, 2, "bar");

unlink(/x/log1);
```

Some possible intermediate states

1.
```
/x/f1    "a foo"
/x/log1 ""
```

2.
```
/x/f1    "a foo"
/x/log1 "2, 3, f"
```

3.
```
/x/f1    "a boo"
/x/log1 "2, 3, foo"
```

Toy Example: Protocol #1

Works in ext3(*data-journal*)!

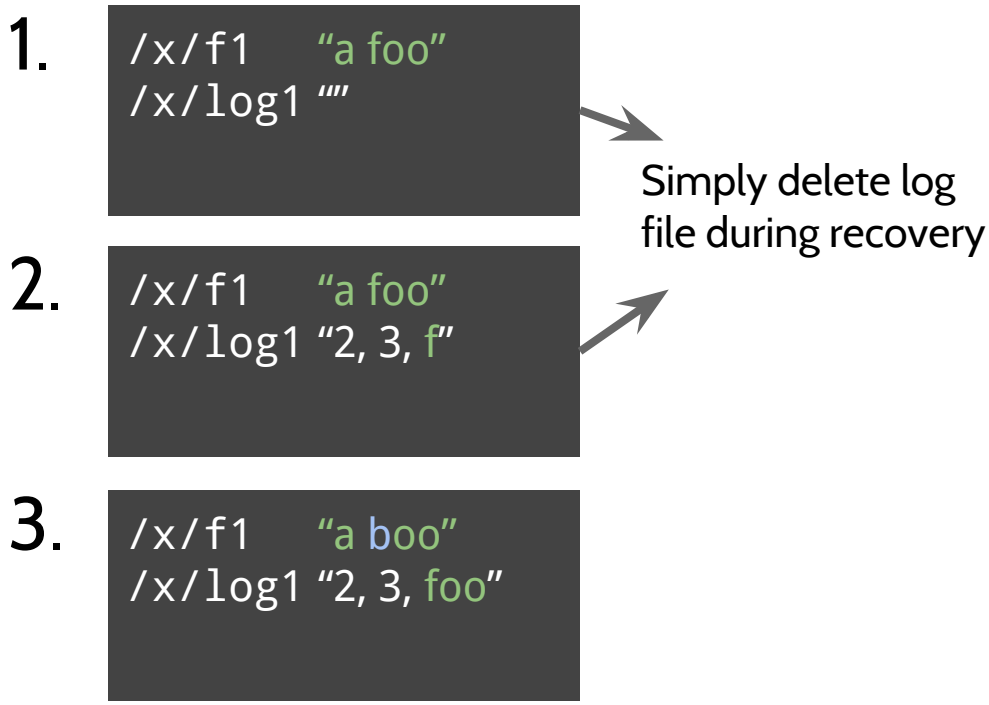
Update Protocol

```
creat(/x/log1);
write(/x/log1, "2, 3, foo");

pwrite(/x/f1, 2, "bar");

unlink(/x/log1);
```

Some possible intermediate states



Toy Example: Protocol #1

Works in ext3(*data-journal*)!

Update Protocol

```
creat(/x/log1);
write(/x/log1, "2, 3, foo");

pwrite(/x/f1, 2, "bar");

unlink(/x/log1);
```

Some possible intermediate states

1.
```
/x/f1    "a foo"
/x/log1 ""
```

2.
```
/x/f1    "a foo"
/x/log1 "2, 3, f"
```

3.
```
/x/f1    "a boo"
/x/log1 "2, 3, foo"
```

→ Recover from log file during recovery

Toy Example: Protocol #1

Works in ext3(*data-journal*)!

Doesn't work in ext3(*data-ordered*)

Update Protocol

```
creat(/x/log1);
write(/x/log1, "2, 3, foo");


pwrite(/x/f1, 2, "bar");

unlink(/x/log1);
```

Toy Example: Protocol #1

Works in ext3(*data-journal*)!

Doesn't work in ext3(*data-ordered*)

Update Protocol

```
creat(/x/log1);
write(/x/log1, "2, 3, foo");

pwrite(/x/f1, 2, "bar");

unlink(/x/log1);
```

ext3(*ordered*) can re-order these two requests, sending pwrite(f1) to disk first, before write(log1)

Toy Example: Protocol #1

Works in ext3(*data-journal*)!

Doesn't work in ext3(*data-ordered*)

Update Protocol

```
creat(/x/log1);
write(/x/log1, "2, 3, foo");

pwrite(/x/f1, 2, "bar");

unlink(/x/log1);
```

A possible intermediate state

```
/x/f1    "a boo"
/x/log1 ""
```

Recovery not possible!

Toy Example: Protocol #2

Works in ext3(*data-journal*), (*data-ordered*)!

Update Protocol

```
creat(/x/log1);
write(/x/log1, ”2, 3, foo”);
fsync(/x/log1);

pwrite(/x/f1, 2, “bar”);
fsync(/x/f1);
unlink(/x/log1);
```

Toy Example: Protocol #2

Works in ext3(*data-journal*), (*data-ordered*)!

Doesn't work in ext3(*writeback*)

Update Protocol

Crash here →

```
creat(/x/log1);
write(/x/log1, "2, 3, foo");
fsync(/x/log1);

pwrite(/x/f1, 2, "bar");
fsync(/x/f1);
unlink(/x/log1);
```

A possible intermediate states

```
/x/f1     "a foo"
/x/log1 "2, 3, #!@"
```

File size alone increases for `log1`, and garbage occurs. Recovery cannot differentiate between garbage and data!

Toy Example: Protocol #3

Works in ext3(*data-journal*), (*data-ordered*), (*writeback*)

Update Protocol

```
creat(/x/log1);
write(/x/log1, ”2, 3, checksum, foo”);
fsync(/x/log1);

pwrite(/x/f1, 2, “bar”);
fsync(/x/f1);
unlink(/x/log1);
```

Toy Example: Protocol #3

Works in ext3(*data-journal*), (*data-ordered*), (*writeback*)

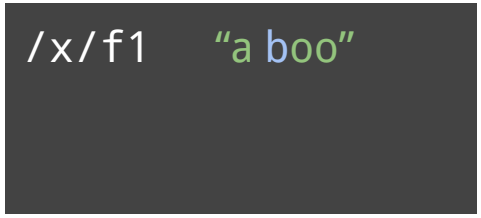
Not enough, according to Linux manpages

Update Protocol

```
creat(/x/log1);
write(/x/log1, "2, 3, checksum, foo");
fsync(/x/log1);

pwrite(/x/f1, 2, "bar");
fsync(/x/f1);
unlink(/x/log1);
```

A possible intermediate states



The log file's directory entry might never be created

Toy Example: Protocol #4

Works in all file systems

Update Protocol

```
creat(/x/log1);
write(/x/log1, ”2, 3, checksum, foo”);
fsync(/x/log1);
fsync(/x);
pwrite(/x/f1, 2, “bar”);
fsync(/x/f1);
unlink(/x/log1);
```

Toy Example: Protocol #5

Works in all file systems

(Additional fsync() required for durability in all FS)

Update Protocol

```
creat(/x/log1);
write(/x/log1, ”2, 3, checksum, foo”);
fsync(/x/log1);
fsync(/x);
pwrite(/x/f1, 2, “bar”);
fsync(/x/f1);
unlink(/x/log1);
fsync(/x);
```

Example: Summary

File systems vary in crash-related behavior

- ext3(*ordered*) re-orders, while ext3(*journaled*) does not

Applications usually *depend* on some behavior

- Depend on ordering: Some `fsync()` calls can be omitted

Outline

Introduction

An Example

BOB: Examining File System Behavior

ALICE: Examining Applications

Conclusion

FS Behavior: Persistence Properties

Two classes of properties: atomicity and ordering

- Atomicity example: Is a write() call atomic in the FS?
- Ordering example: Are write() calls sent to disk in-order?

Studied *ext2, ext3, ext4, btrfs, xfs, reiserfs*

- We studied 16 configurations of the six file systems

Methodology: The Block-Order Breaker (BOB)

1. Run user-level workloads stressing the property
   - Example: write(8KB) for testing atomicity of write() calls

Methodology: The Block-Order Breaker (BOB)

1. Run user-level workloads stressing the property
   - Example: write(8KB) for testing atomicity of write() calls
2. Record block-level trace of the workload

Methodology: The Block-Order Breaker (BOB)

1. Run user-level workloads stressing the property
   - Example: write(8KB) for testing atomicity of write() calls
2. Record block-level trace of the workload
3. Reconstruct disk-states possible on a power-loss
   - All states possible if disk-cache does not re-order
   - A few states where disk-cache re-orders

Methodology: The Block-Order Breaker (BOB)

1. Run user-level workloads stressing the property
   - Example: write(8KB) for testing atomicity of write() calls
2. Record block-level trace of the workload
3. Reconstruct disk-states possible on a power-loss
   - All states possible if disk-cache does not re-order
   - A few states where disk-cache re-orders
4. Run FS recovery, verify property on each disk-state
   - Example: Is all 8KB written atomically?

File System Study: Results

File system configuration		Atomicity				Ordering			
		One sector overwrite	*Append content*	*Many sector overwrite*	*Directory operation*	*Overwrite → Any op*	*Append → Any op*	*Dir-op → Any op*	*Append → Rename*
ext2	*async*		✘	✘	✘	✘	✘	✘	✘
	sync		✘	✘	✘				
ext3	*writeback*		✘	✘		✘	✘		✘
	ordered			✘		✘			
	data-journal			✘					
ext4	*writeback*		✘	✘		✘	✘		✘
	ordered			✘		✘	✘		
	no-delalloc			✘		✘			
	data-journal			✘					
btrfs				✘			✘	✘	
xfs	*default*			✘		✘	✘		
	wsync			✘		✘			

File System Study: Results

File System

Different Configurations of File System

File system configuration		Atomicity				Ordering			
		One sector overwrite	*Append content*	*Many sector overwrite*	*Directory operation*	*Overwrite → Any op*	*Append → Any op*	*Dir-op → Any op*	*Append → Rename*
ext2	*async*		✘	✘	✘	✘	✘	✘	✘
	sync		✘	✘	✘				
ext3	*writeback*		✘	✘		✘	✘		✘
	ordered			✘		✘			
	data-journal			✘					
ext4	*writeback*		✘	✘		✘	✘		✘
	ordered			✘		✘	✘		
	no-delalloc			✘		✘			
	data-journal			✘					
btrfs				✘			✘	✘	
xfs	*default*			✘		✘	✘		
	wsync			✘		✘			

File System Study: Results

Persistence Properties considered

File system configuration		Atomicity				Ordering			
		One sector overwrite	*Append content*	*Many sector overwrite*	*Directory operation*	*Overwrite → Any op*	*Append → Any op*	*Dir-op → Any op*	*Append → Rename*
ext2	*async*		✘	✘	✘	✘	✘	✘	✘
	sync		✘	✘	✘				
ext3	*writeback*		✘	✘		✘	✘		✘
	ordered			✘		✘			
	data-journal			✘					
ext4	*writeback*		✘	✘		✘	✘		✘
	ordered			✘		✘	✘		
	no-delalloc			✘		✘			
	data-journal			✘					
btrfs				✘			✘	✘	
xfs	*default*			✘		✘	✘		
	wsync			✘		✘			

File System Study: Results

Is a directory operation, like rename(), atomic on a system crash?

File system configuration		Atomicity				Ordering			
		One sector overwrite	*Append content*	*Many sector overwrite*	*Directory operation*	*Overwrite → Any op*	*Append → Any op*	*Dir-op → Any op*	*Append → Rename*
ext2	*async*		✘	✘	✘	✘	✘	✘	✘
	sync		✘	✘	✘				
ext3	*writeback*		✘	✘		✘	✘		✘
	ordered			✘		✘			
	data-journal			✘					
ext4	*writeback*		✘	✘		✘	✘		✘
	ordered			✘		✘	✘		
	no-delalloc			✘		✘			
	data-journal			✘					
btrfs				✘			✘	✘	
xfs	*default*			✘		✘	✘		
	wsync			✘		✘			

File System Study: Results

Property certainly not obeyed

File system configuration		Atomicity				Ordering			
		One sector overwrite	*Append content*	*Many sector overwrite*	*Directory operation*	*Overwrite → Any op*	*Append → Any op*	*Dir-op → Any op*	*Append → Rename*
ext2	*async*		✘	✘	✘	✘	✘	✘	✘
	sync		✘	✘	✘				
ext3	*writeback*		✘	✘		✘	✘		✘
	ordered			✘		✘			
	data-journal			✘					
ext4	*writeback*		✘	✘		✘	✘		✘
	ordered			✘		✘	✘		
	no-delalloc			✘		✘			
	data-journal			✘					
btrfs				✘			✘	✘	
xfs	*default*			✘		✘	✘		
	wsync			✘		✘			

We did not see a violation

File System Study: Results

Main result: File systems vary in their persistence properties

File system configuration		Atomicity				Ordering			
		One sector overwrite	*Append content*	*Many sector overwrite*	*Directory operation*	*Overwrite → Any op*	*Append → Any op*	*Dir-op → Any op*	*Append → Rename*
ext2	*async*		✘	✘	✘	✘	✘	✘	✘
	sync		✘	✘	✘				
ext3	*writeback*		✘	✘		✘	✘		✘
	ordered			✘		✘			
	data-journal			✘					
ext4	*writeback*		✘	✘		✘	✘		✘
	ordered			✘		✘	✘		
	no-delalloc			✘		✘			
	data-journal			✘					
btrfs				✘			✘	✘	
xfs	*default*			✘		✘	✘		
	wsync			✘		✘			

File System Study: Conclusion

Applications should not rely on persistence properties

Testing applications on a specific FS is not enough

- ext3(*data-journal*): Re-ordering vulnerabilities are hidden

Outline

Introduction

An Example

BOB: Examining File System Behavior

ALICE: Examining Applications

Conclusion

ALICE: Goal

"Application-Level Intelligent Crash Explorer"

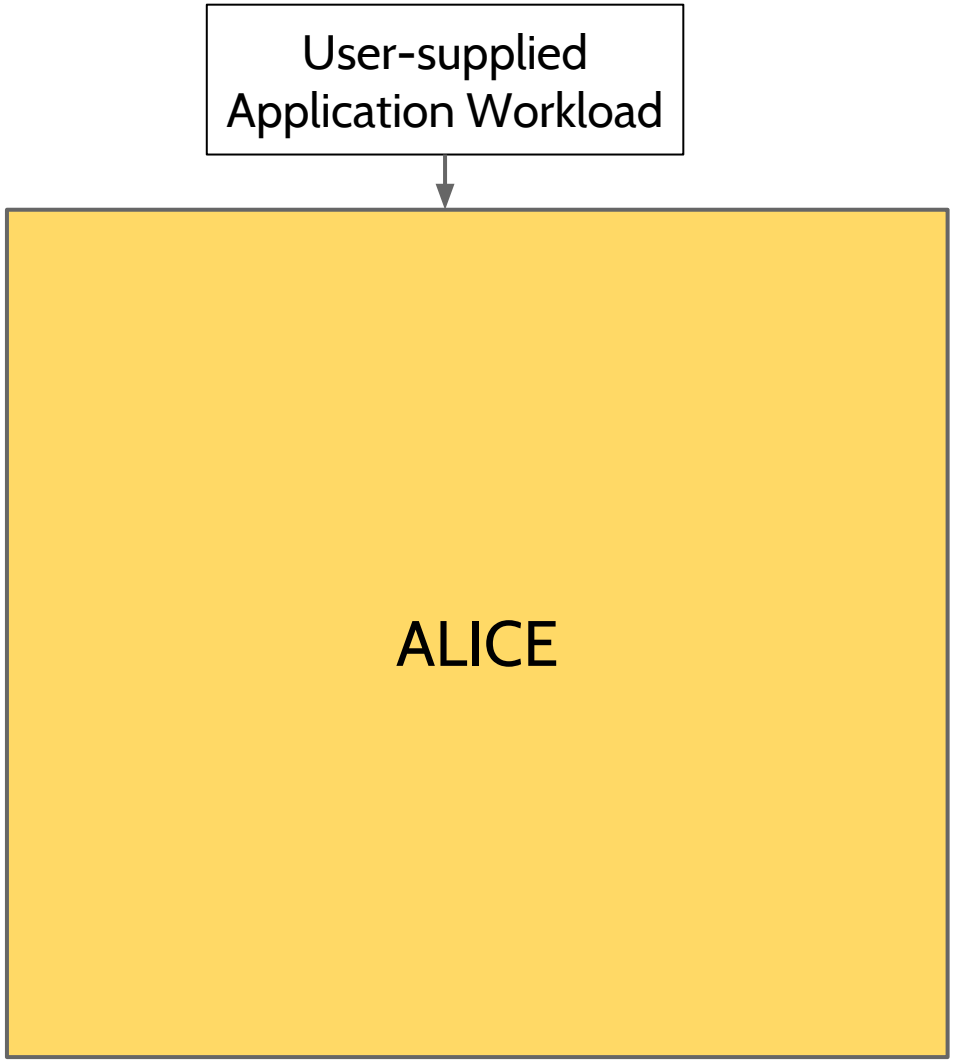
Goal: Tool to find crash vulnerabilities of an application

- Find vulnerabilities across all file systems
- Relate vulnerabilities to specific source lines
- Relate vulnerabilities to file system behavior

ALICE: Technique

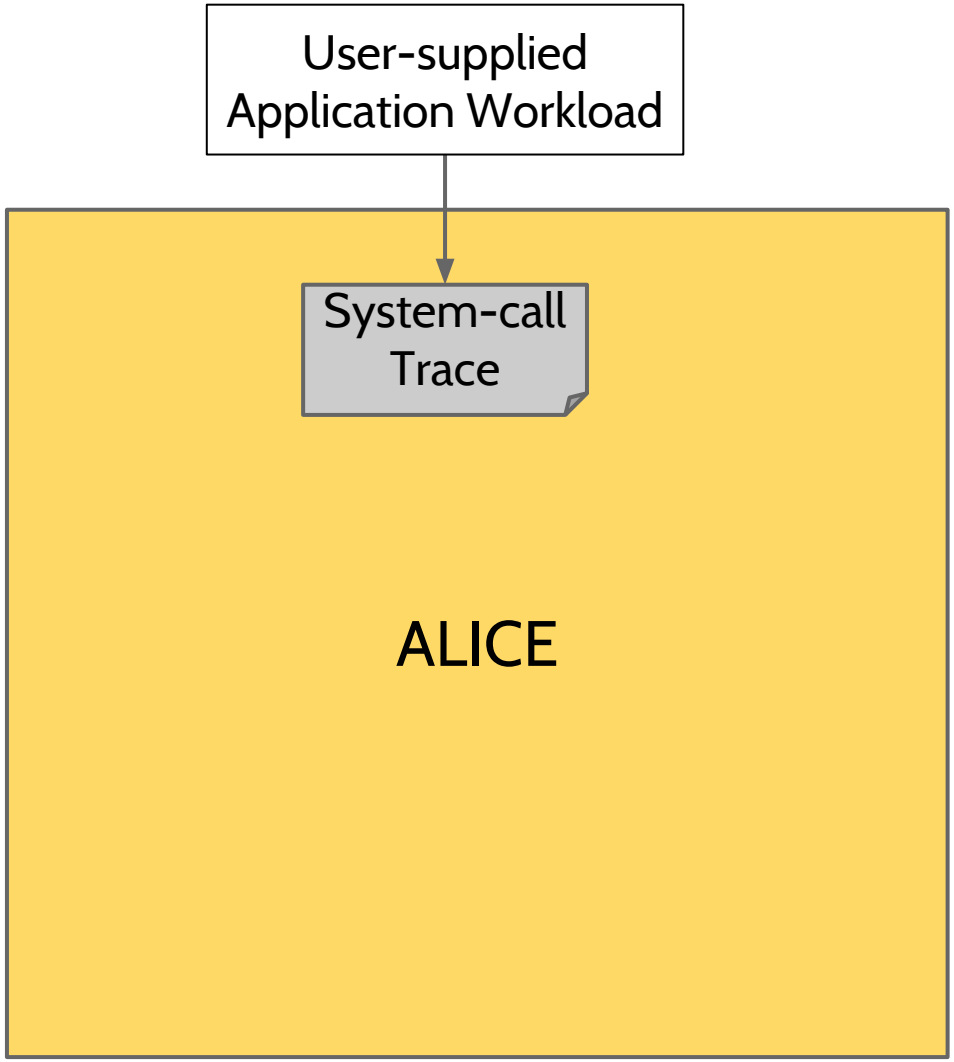
User supplies ALICE with an application workload

- Example: A database transaction



ALICE: Technique

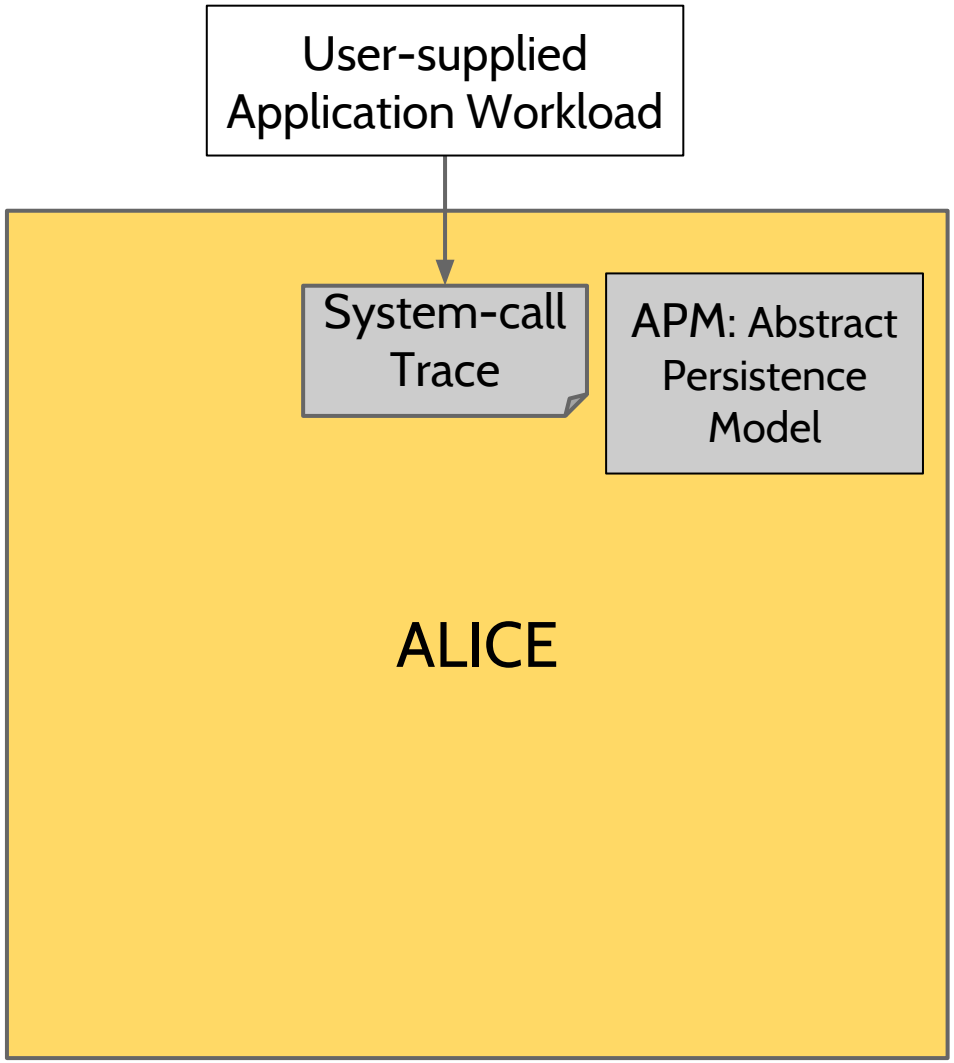
ALICE runs workload and records system-call trace



ALICE: Technique

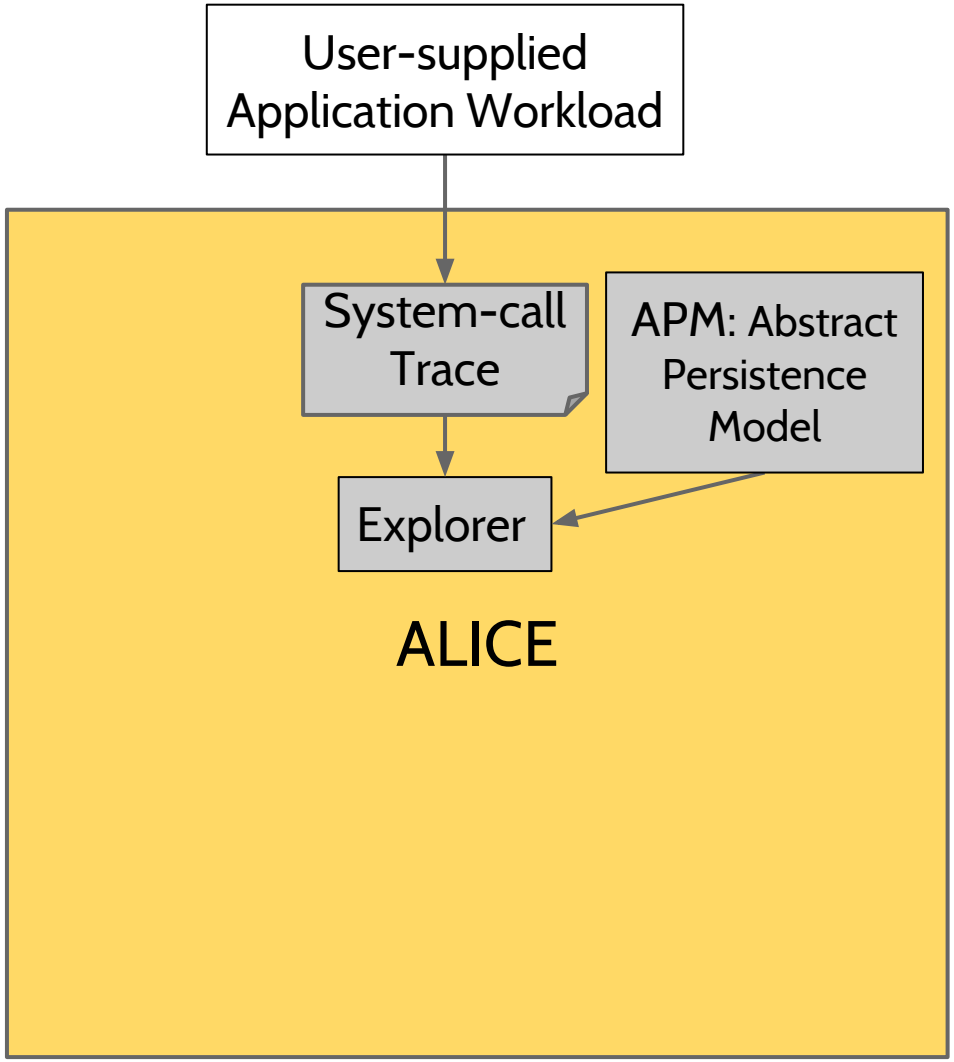
APM models all crash states that can occur on an FS

- Default, weak APM allows many possible states
- Custom APMs can be configured by user for a specific file system
- Eg: ext3(*ordered*) APM allows states with overwrites re-ordered; ext3(*data-journal*) APM does not



ALICE: Technique

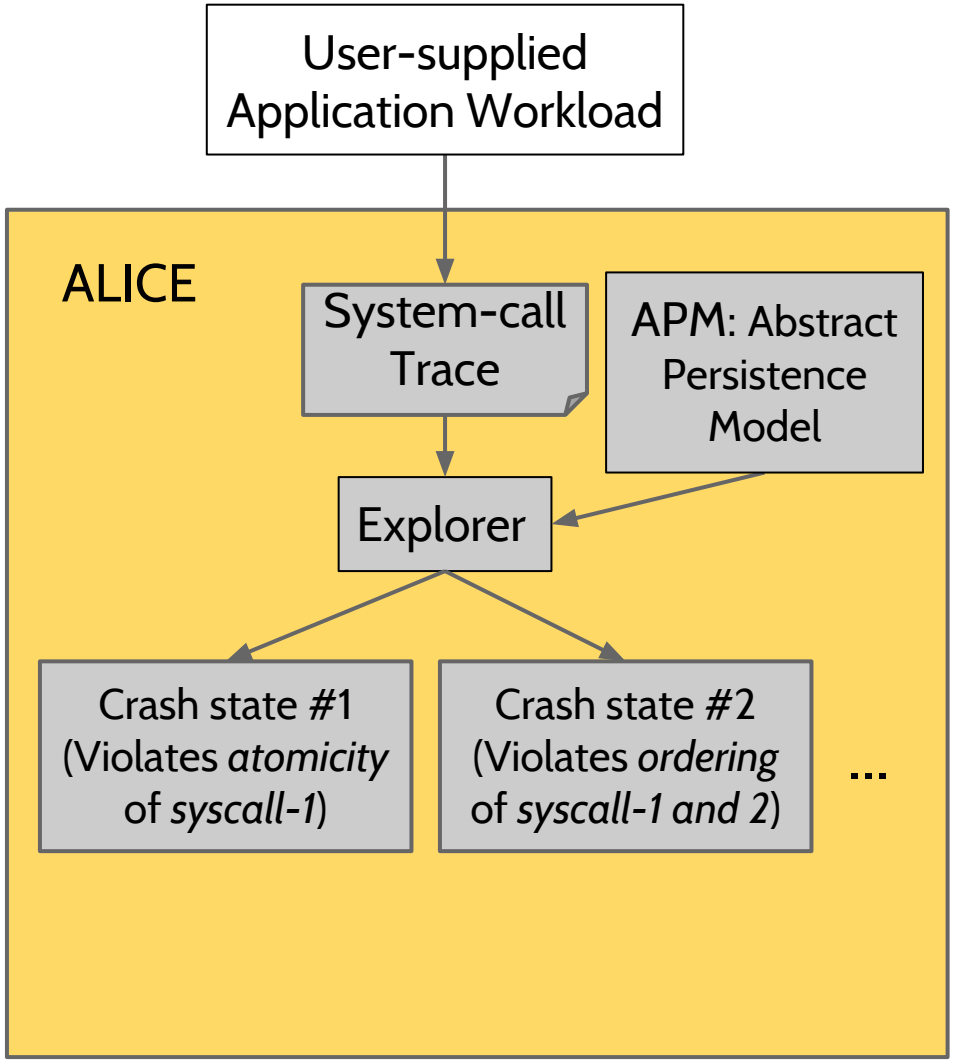
Explorer reconstructs some states using the APM



ALICE: Technique

Explorer targets specific states

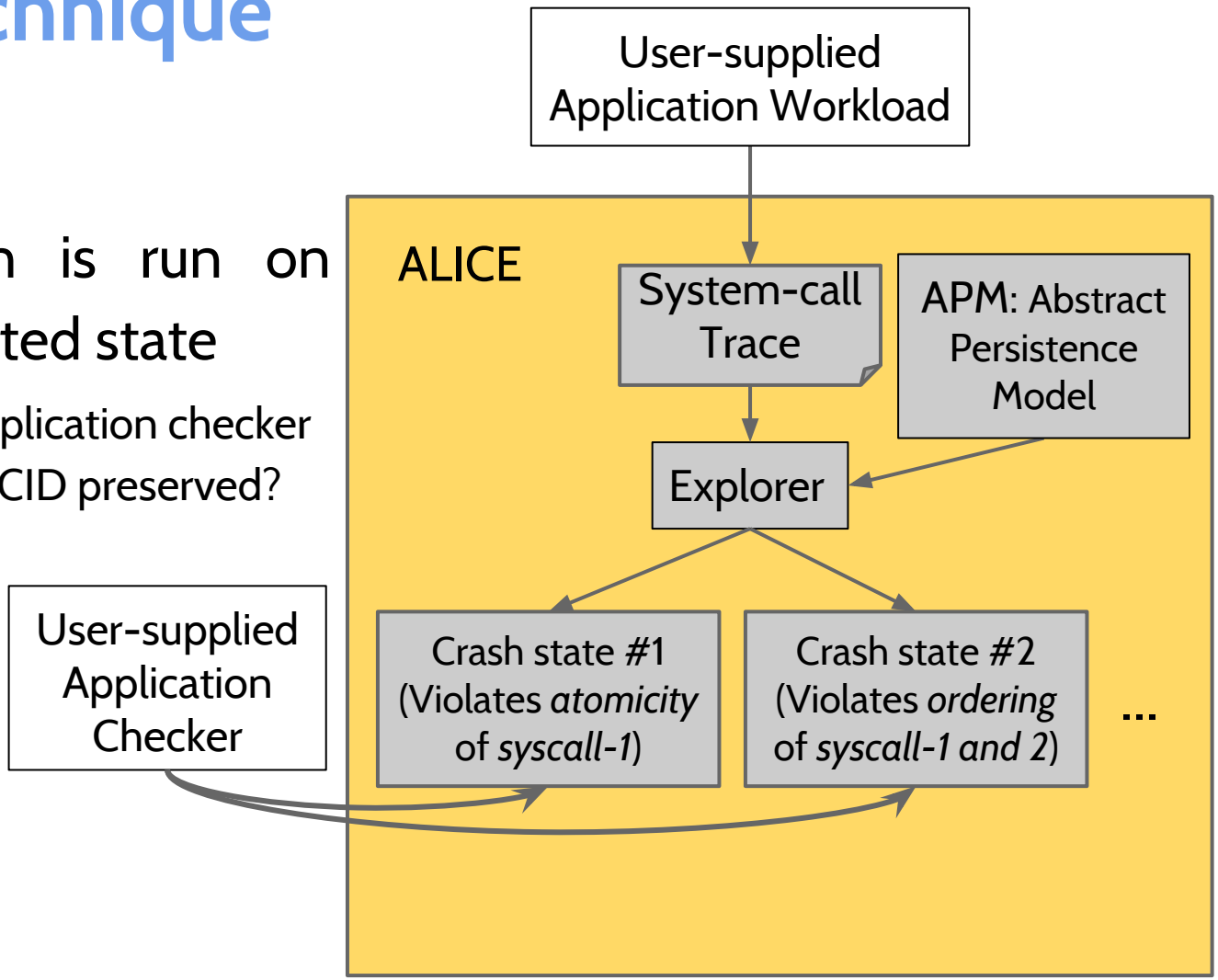
- Relating to atomicity and re-ordering of each syscall



ALICE: Technique

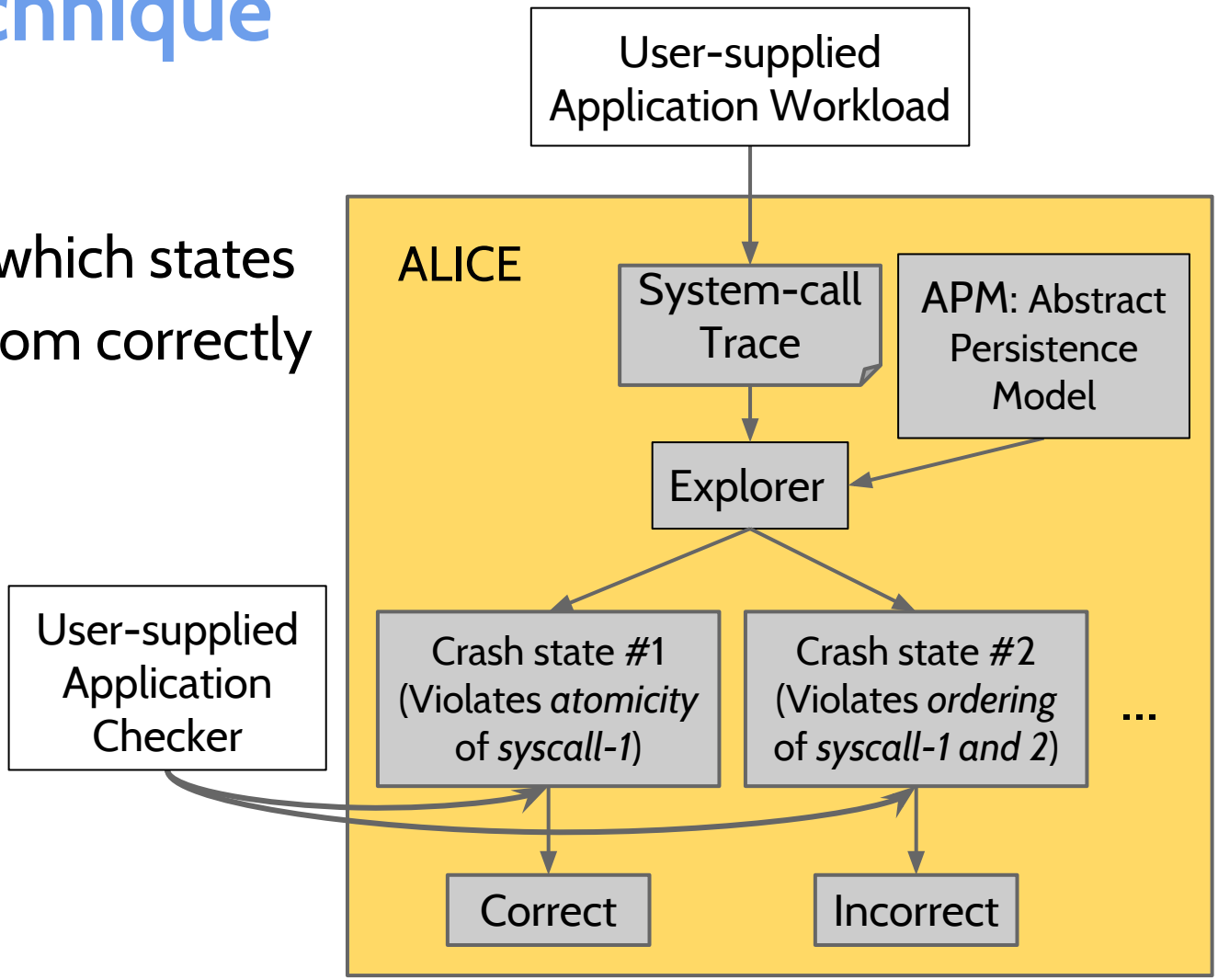
The application is run on each reconstructed state

- User supplies application checker
- Example: Was ACID preserved?



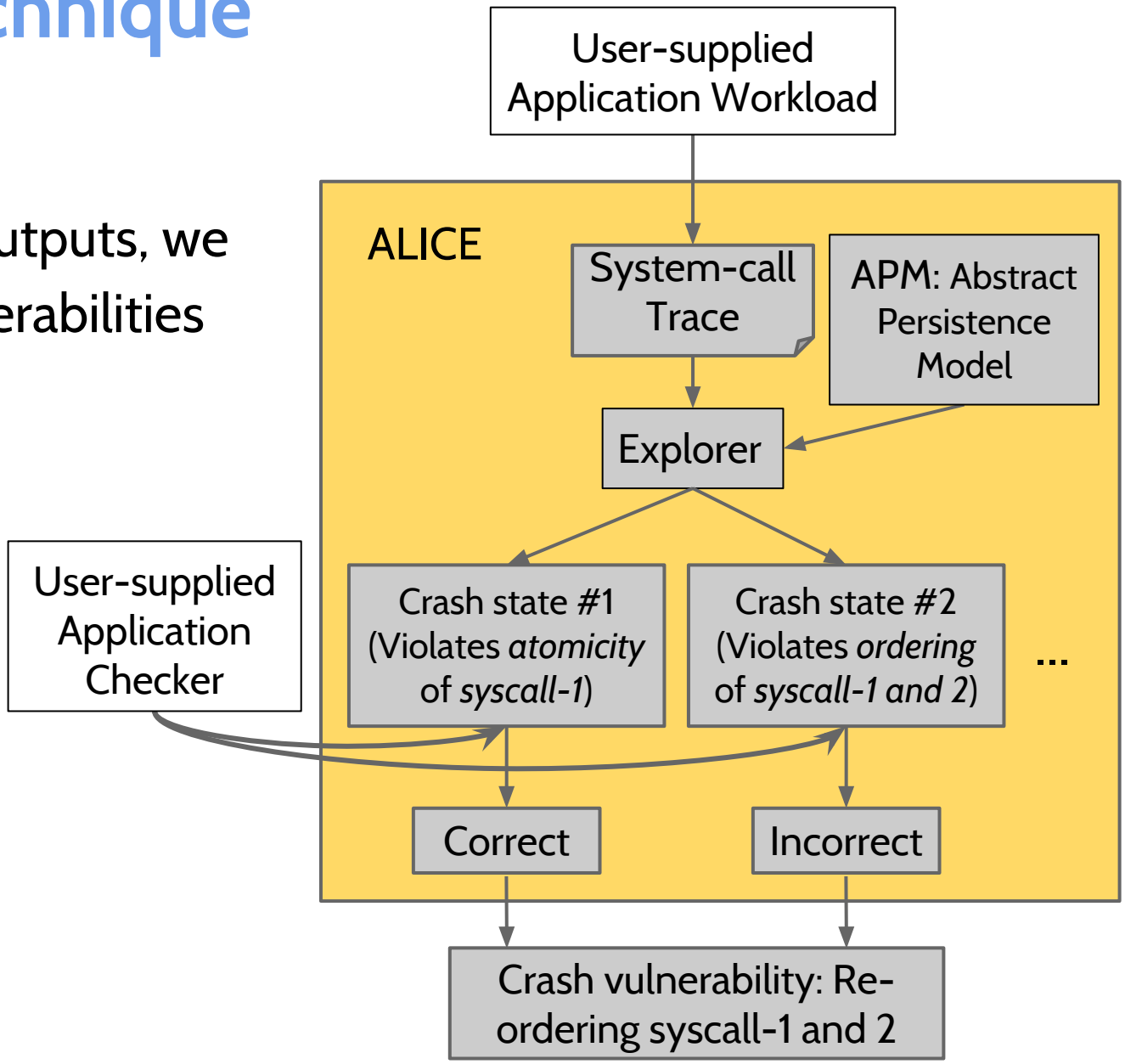
ALICE: Technique

Checker shows which states are recovered from correctly

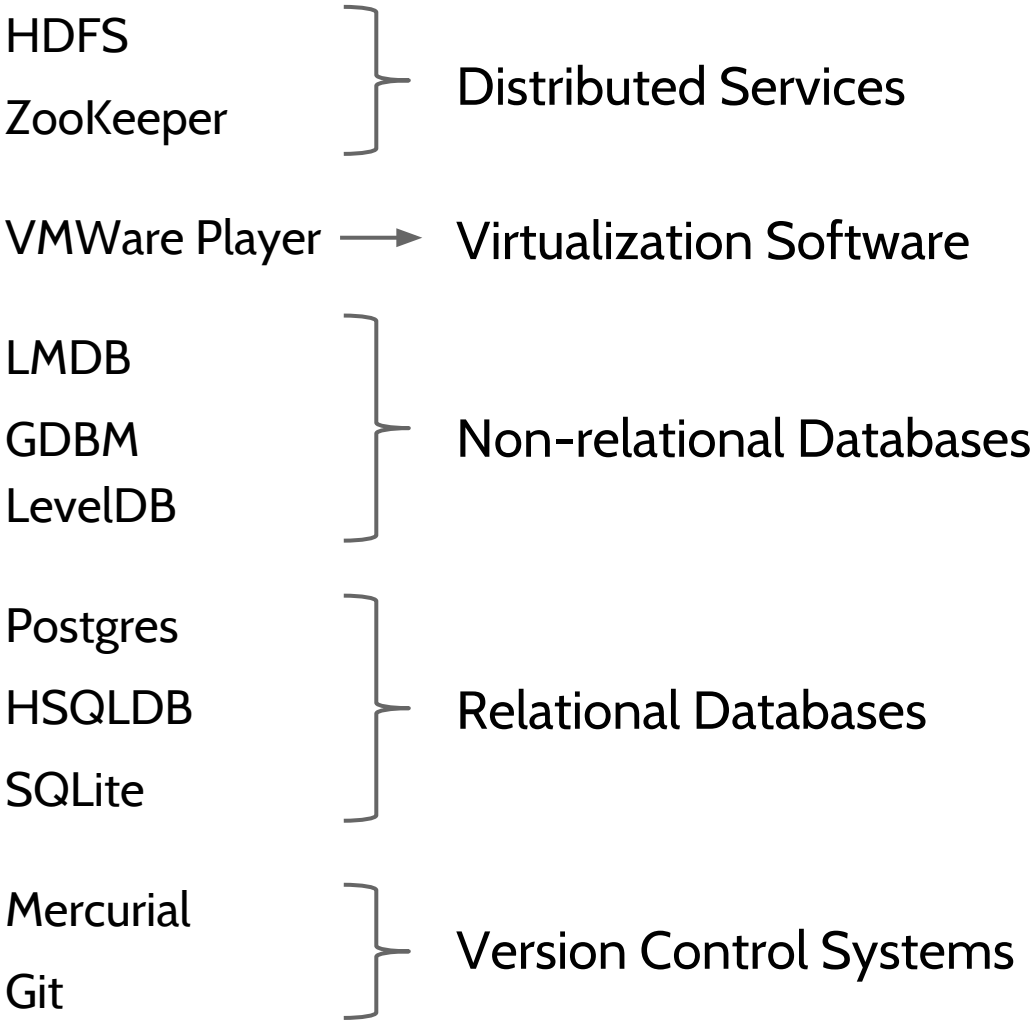


ALICE: Technique

From checker outputs, we determine vulnerabilities



Vulnerability Study: Applications



Example: Git

creat(index.lock)
mkdir(o/x)
creat(o/x/tmp_y)
append(o/x/tmp_y)
fsync(o/x/tmp_y)
link(o/x/tmp_y, o/x/y)
unlink(o/x/tmp_y)
append(index.lock)
rename(index.lock,index)
stdout(finished add)

Read the full paper to correctly interpret results!

Example: Git

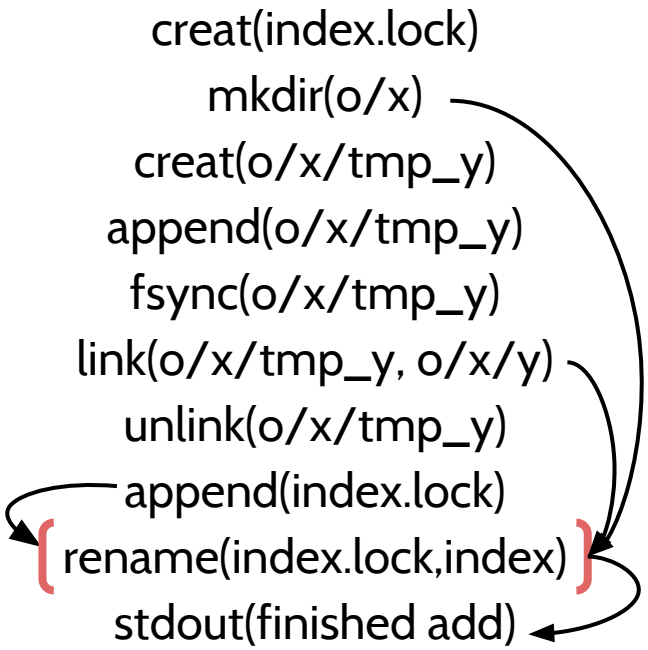
creat(index.lock)
mkdir(o/x)
creat(o/x/tmp_y)
append(o/x/tmp_y)
fsync(o/x/tmp_y)
link(o/x/tmp_y, o/x/y)
unlink(o/x/tmp_y)
append(index.lock)
[rename(index.lock,index)]
stdout(finished add)

Read the full paper to correctly interpret results!

[ ] Atomicity vulnerability

Which system calls need to be atomic?

Example: Git



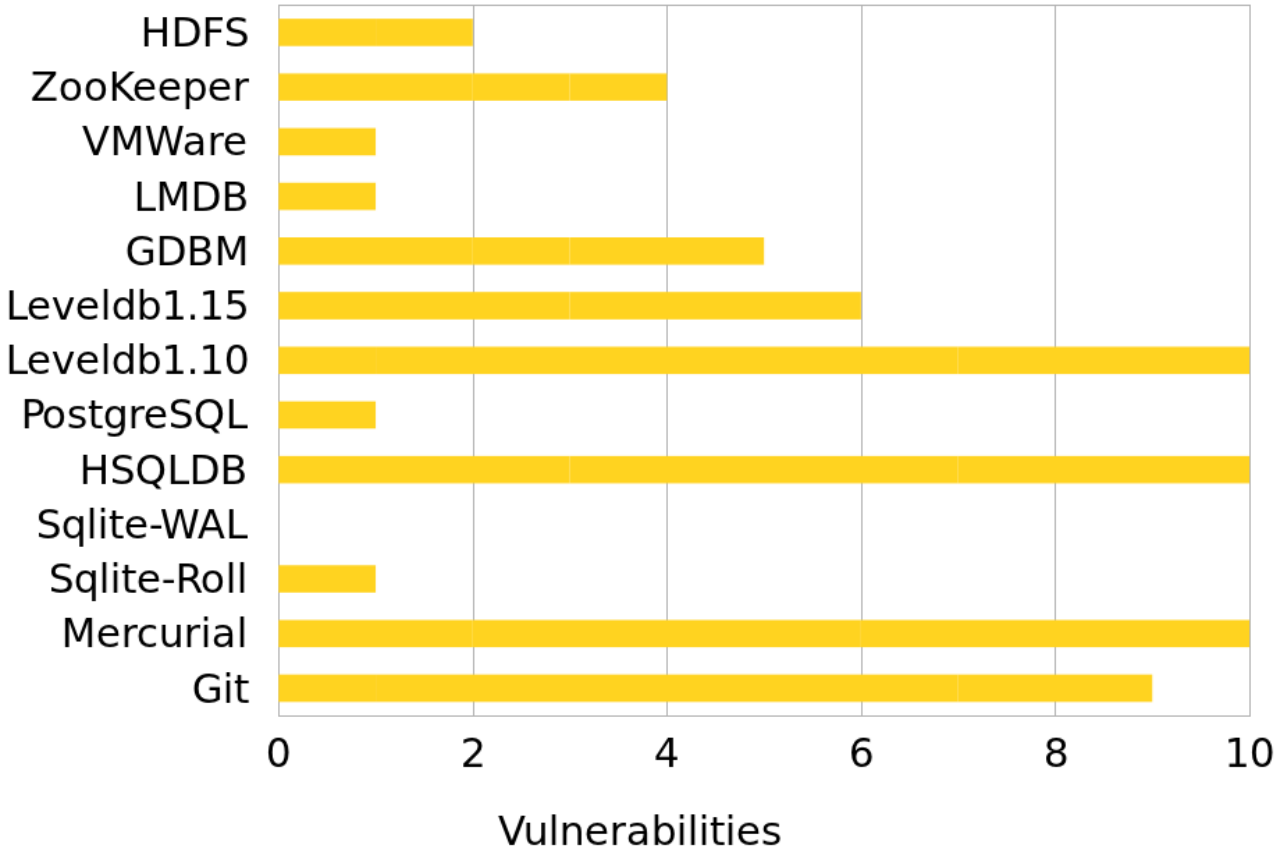
Read the full paper to correctly interpret results!

[ ] Atomicity vulnerability

→ Ordering vulnerability

Which system-call re-orderings cause incorrectness?

Vulnerability Study: Default (Weak) APM

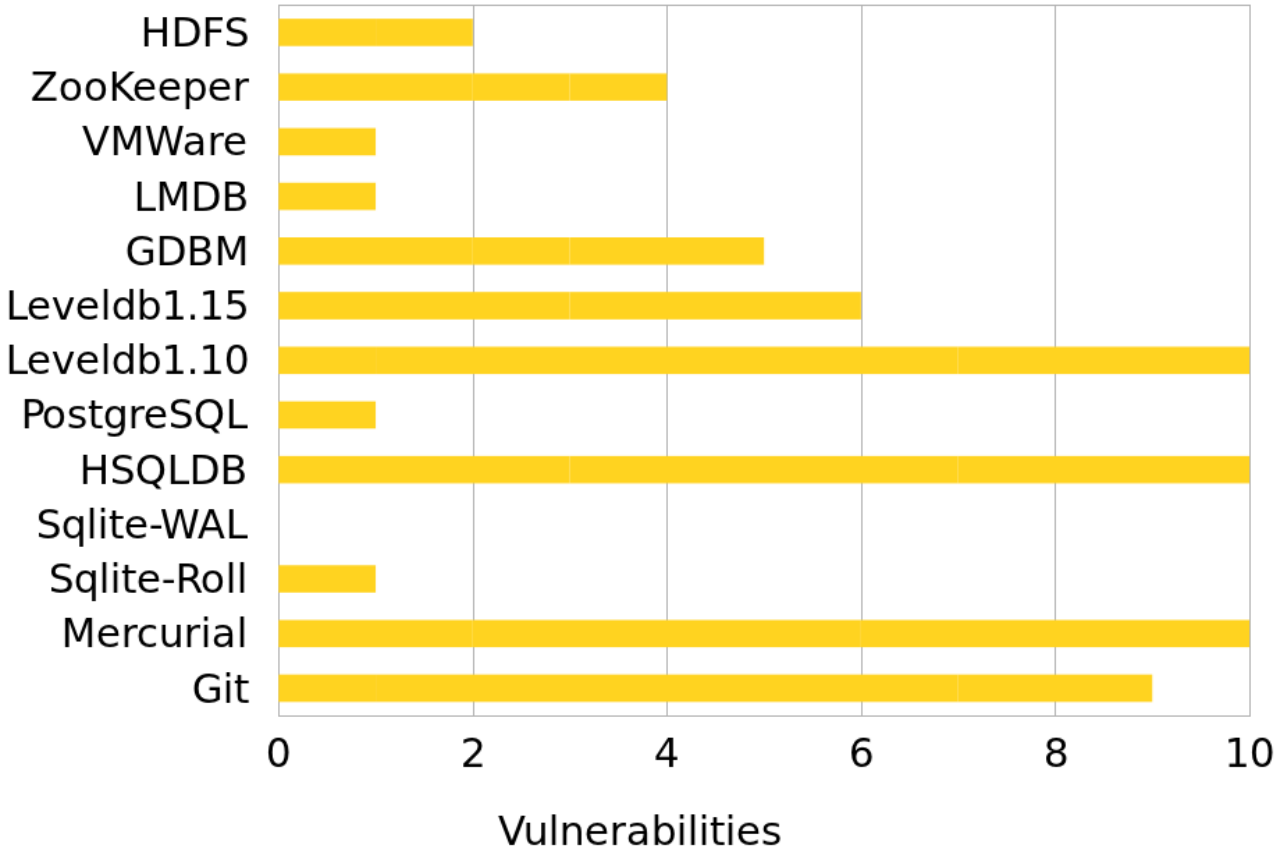


Read the full paper to correctly interpret results!

Vulnerability Study: Default (Weak) APM

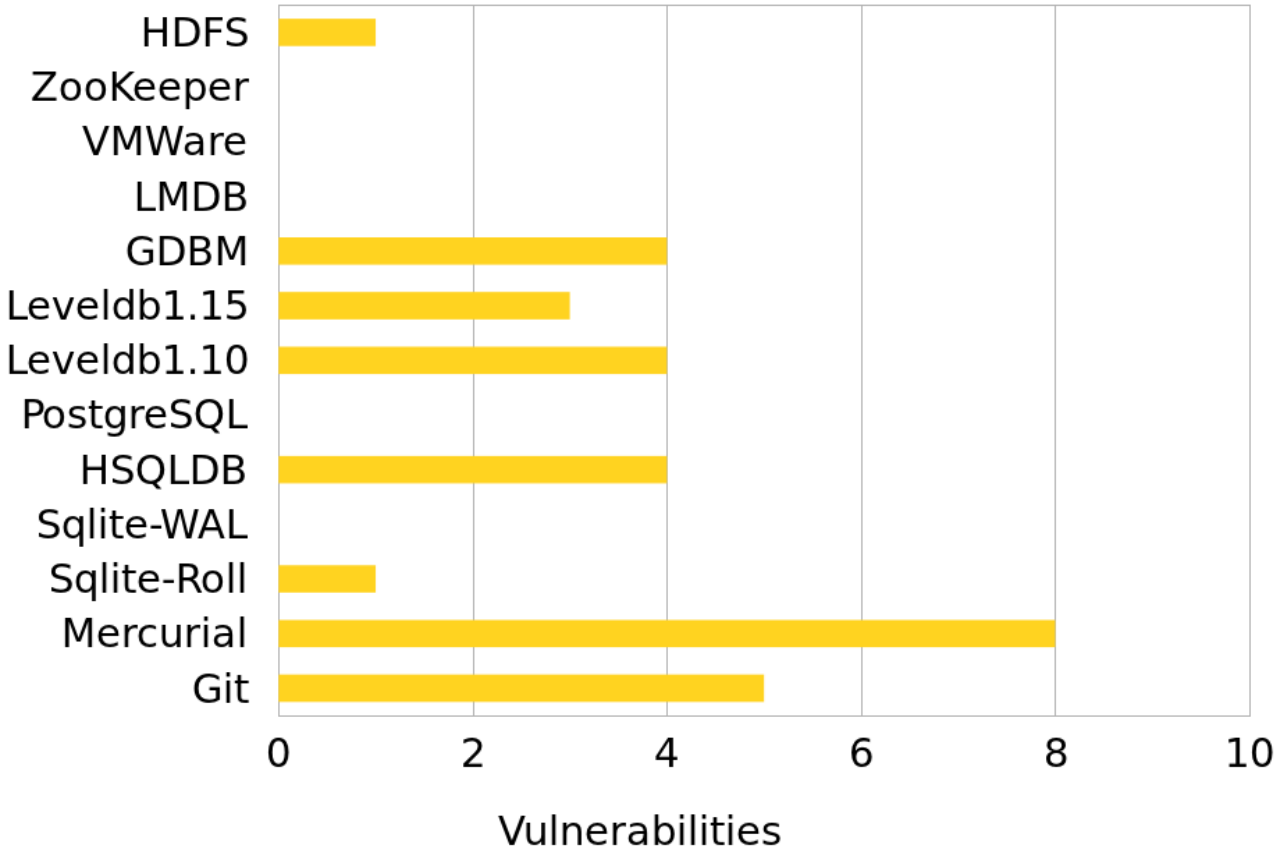
60 Vulnerabilities

Many result in silent data loss, inaccessible applications



Read the full paper to correctly interpret results!

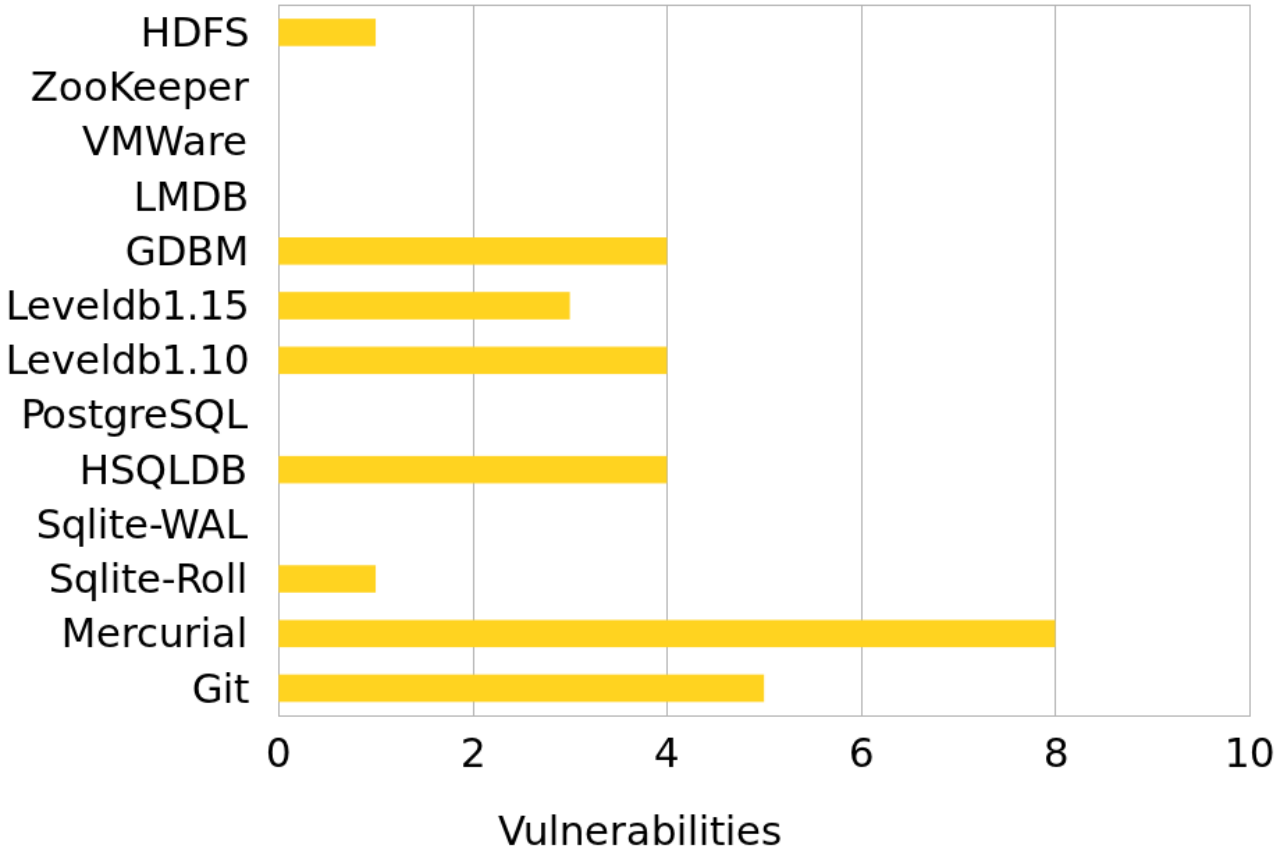
Vulnerability Study: Btrfs APM



Read the full paper to correctly interpret results!

Vulnerability Study: Btrfs APM

31 Vulnerabilities



Read the full paper to correctly interpret results!

What FS behavior affects applications?

Garbage during file-appends

- Affects 3 applications
- But, partial appends with actual data: 0 vulnerabilities

FS safety heuristics seemingly don't help much

- Only 2 found vulnerabilities by "Flush data before rename"
- Heuristics might help other types of applications

Non-synchronous directory operations

- Affects durability of 6 applications

In the paper …

In-depth: What FS behaviors affect applications

Vulnerabilities under other APMs

Interactions with application developers

How *not* to interpret our results

An efficient FS design with safety validated by ALICE

Summary

FS vary in behavior affecting application consistency

- *ext2, ext3, ext4, btrfs, xfs, reiserfs* vary even among their different configurations
- Subtle implementation details affect behavior

Application protocols are complex, vulnerable

- 60 vulnerabilities under weak APM
- More than half exposed under *ext3, ext4, btrfs*
- Depend (by design or unwittingly) on FS implementation

A parting note ....

Experienced App-Developer:
POSIX doesn't let FSes do that

Can you point us to the exact POSIX documentation?

Developer: I can't find it, but I remember someone saying so

A parting note ....

Experienced App-Developer:
POSIX doesn't let FSes do that

Can you point us to the exact POSIX documentation?

Developer: I can't find it, but I remember someone saying so

Experienced Academic:
Real file systems don't do that

But <...> does just that

Academic: My students would flunk class if they built a file system that way ...

A parting note ....

Experienced App-Developer:
POSIX doesn't let FSes do that

Can you point us to the exact POSIX documentation?

Developer: I can't find it, but I remember someone saying so

Experienced Academic:
Real file systems don't do that

But <...> does just that

Academic: My students would flunk class if they built a file system that way ...

Thank you!

Download tools: http://research.cs.wisc.edu/adsl/Software/alice