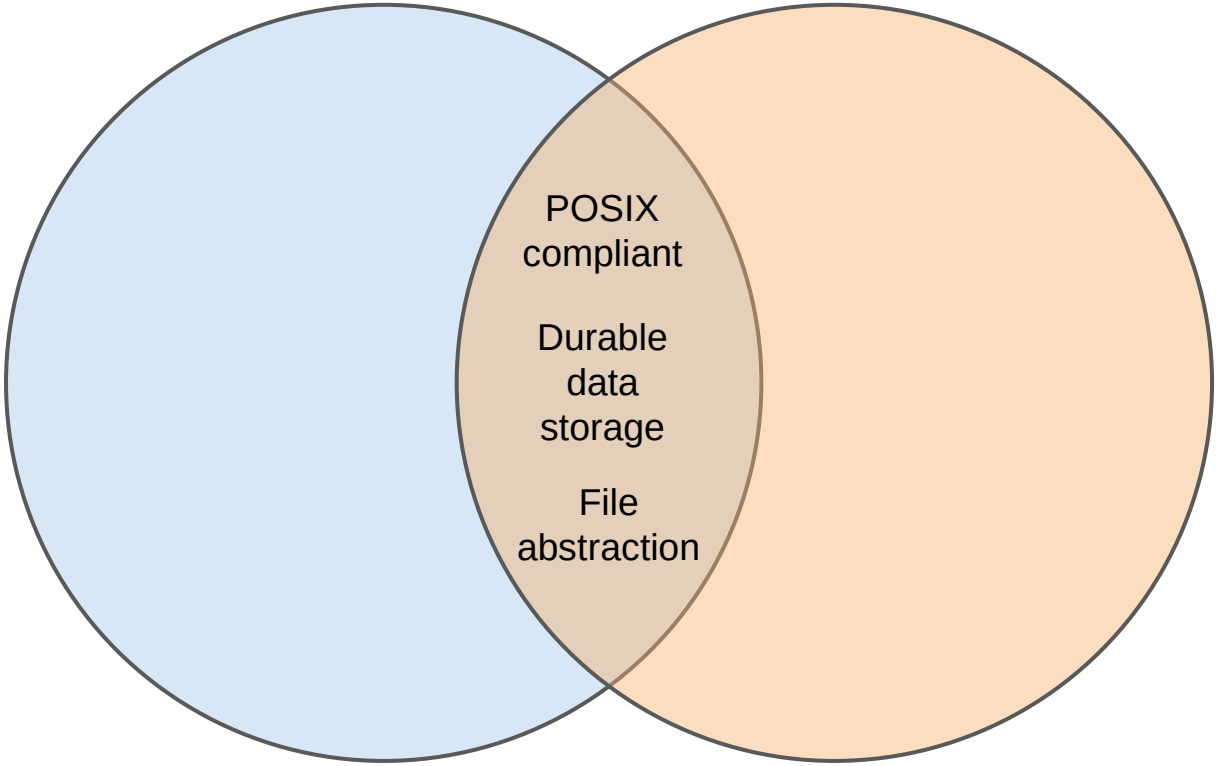
# Chipmunk: Investigating Crash-Consistency in Persistent-Memory File Systems

Hayley LeBlanc, Shankara Pailoor, Om Saran K R E, Isil Dillig, James Bornholt, Vijay Chidambaram

TEXAS
The University of Texas at Austin

vmware®

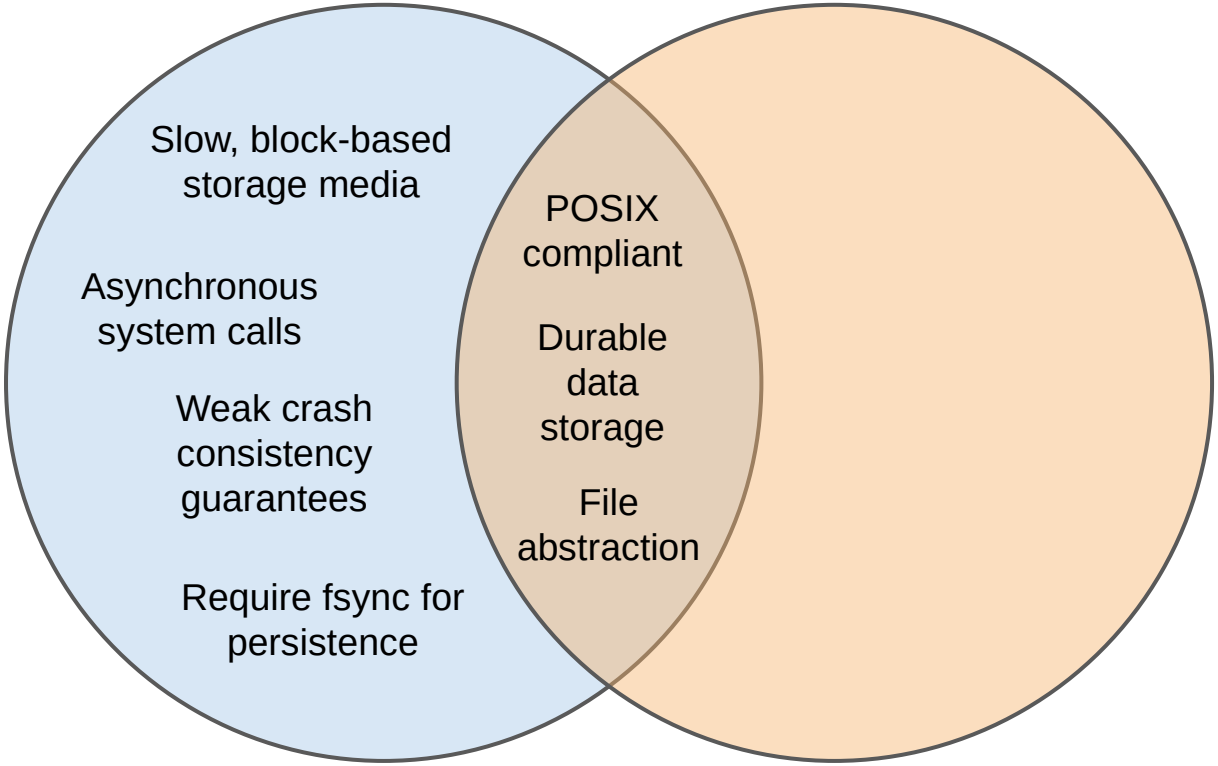
# File systems




Traditional FSes (HDD, SSD)

Persistent memory (PM) FSes

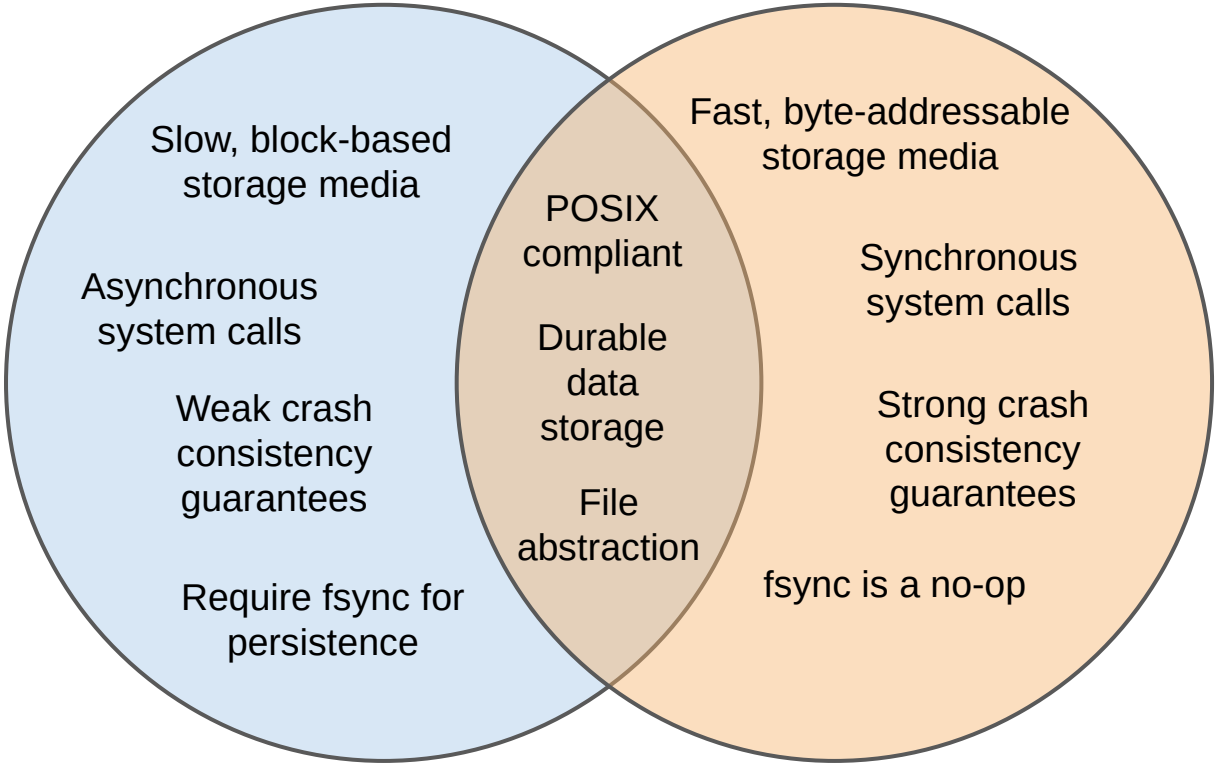
# File systems




Traditional FSes (HDD, SSD)

Persistent memory (PM) FSes

# File systems




Traditional FSes (HDD, SSD)

Persistent memory (PM) FSes

# PM file system design space

# PM file system design space

**New designs are necessary to maximize the performance of PM file systems**

# PM file system design space

**New designs are necessary to maximize the performance of PM file systems**

- New journaling and logging protocols

# PM file system design space

**New designs are necessary to maximize the performance of PM file systems**

- New journaling and logging protocols
- In-place updates

# PM file system design space

**New designs are necessary to maximize the performance of PM file systems**

- New journaling and logging protocols
- In-place updates
- File systems in user space

# PM file system design space

**New designs are necessary to maximize the performance of PM file systems**

- New journaling and logging protocols
- In-place updates
- File systems in user space
- Volatile indexes and allocators

# PM file system design space

**New designs are necessary to maximize the performance of PM file systems**

- New journaling and logging protocols
- In-place updates
- File systems in user space
- Volatile indexes and allocators

**How do we test these new designs for crash consistency?**

# Chipmunk

**https://github.com/utsaslab/chipmunk**

# Chipmunk

Record-and-replay crash-consistency testing framework for PM file systems

**https://github.com/utsaslab/chipmunk**

# Chipmunk

Record-and-replay crash-consistency testing framework for PM file systems

Found 23 new crash-consistency bugs across 5 file systems

**https://github.com/utsaslab/chipmunk**

# Chipmunk

Record-and-replay crash-consistency testing framework for PM file systems

Found 23 new crash-consistency bugs across 5 file systems

Compatible with POSIX-compliant file systems in user and kernel space

**https://github.com/utsaslab/chipmunk**

# Chipmunk

Record-and-replay crash-consistency testing framework for PM file systems

Found 23 new crash-consistency bugs across 5 file systems

Compatible with POSIX-compliant file systems in user and kernel space

Function-based instrumentation for recording updates to PM

**https://github.com/utsaslab/chipmunk**

# Chipmunk

Record-and-replay crash-consistency testing framework for PM file systems

Found 23 new crash-consistency bugs across 5 file systems

Compatible with POSIX-compliant file systems in user and kernel space

Function-based instrumentation for recording updates to PM

Supports fuzzing and bounded test generation

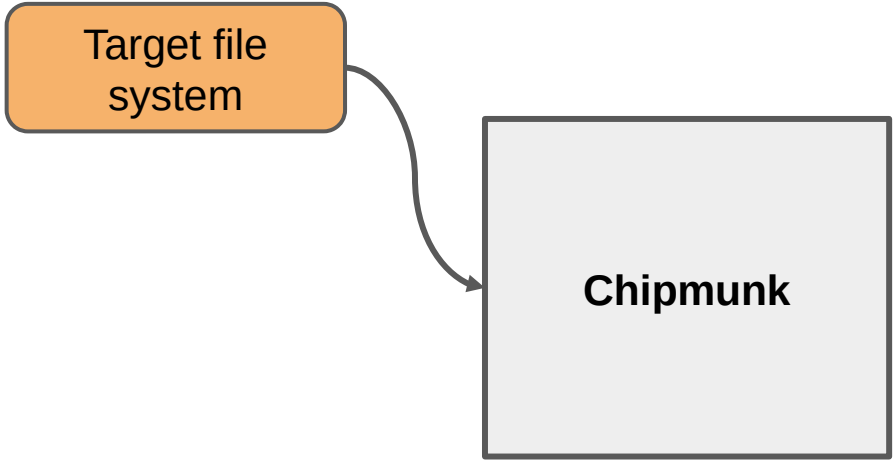
**https://github.com/utsaslab/chipmunk**

# Outline

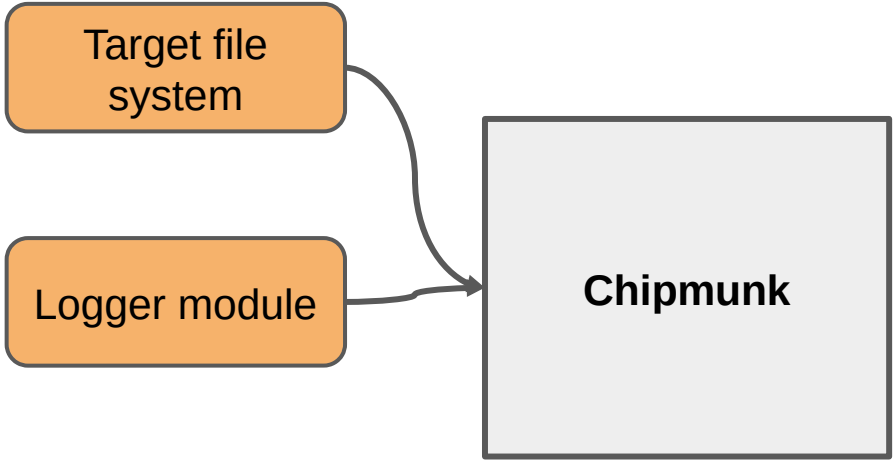
1. Introduction
2. **Chipmunk overview**
3. Experiments and bugs
4. Observations and insights

# Chipmunk overview

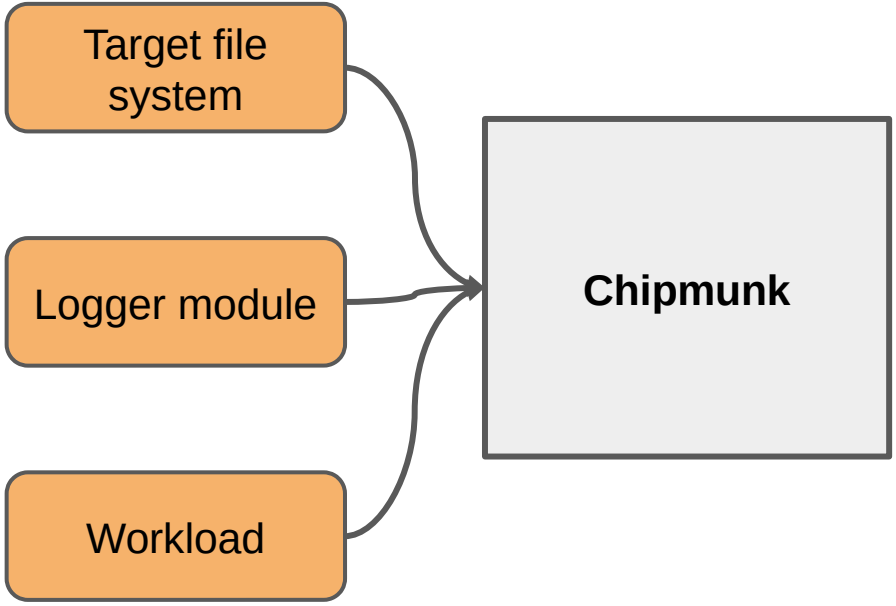
# Chipmunk overview



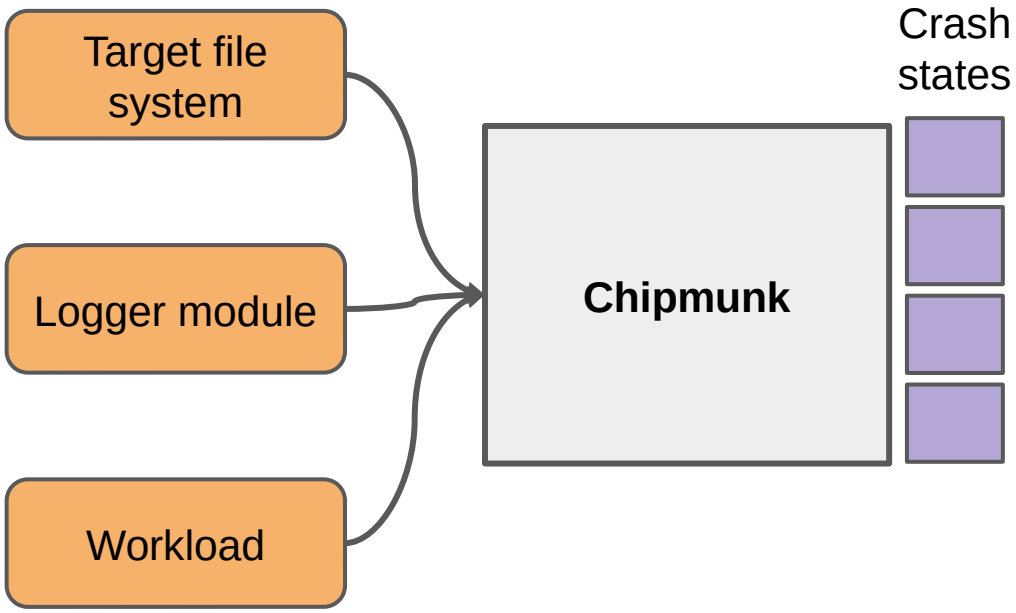
# Chipmunk overview



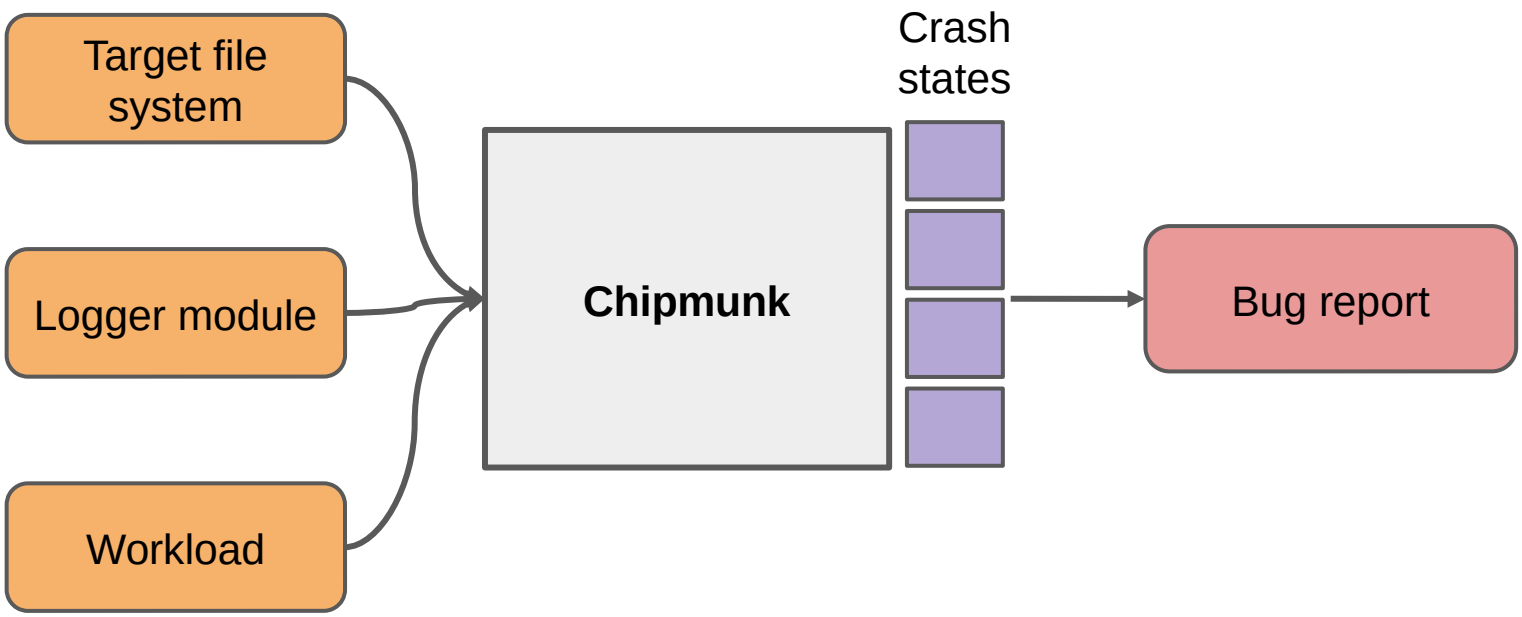
# Chipmunk overview



# Chipmunk overview

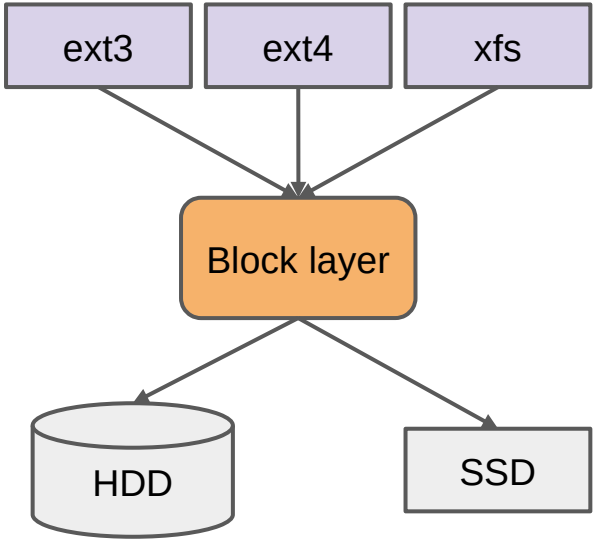


# Chipmunk overview



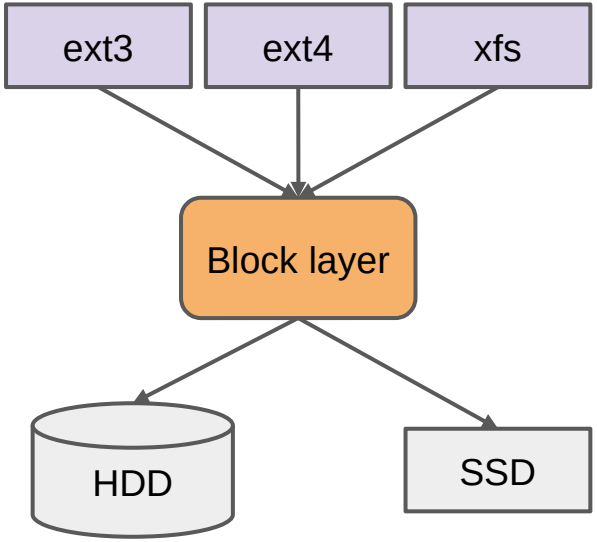
# Recording updates

Traditional file systems

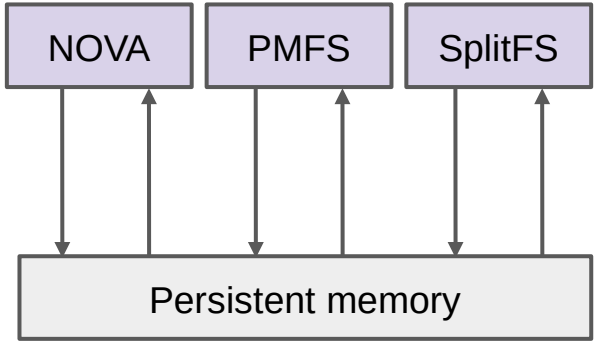


# Recording updates

Traditional file systems

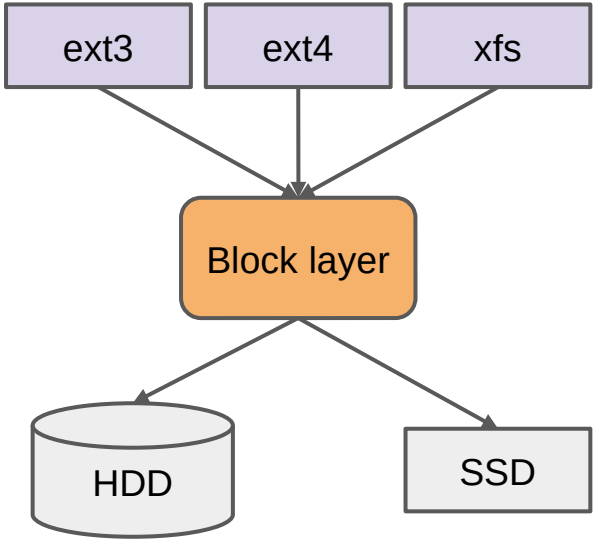



PM file systems

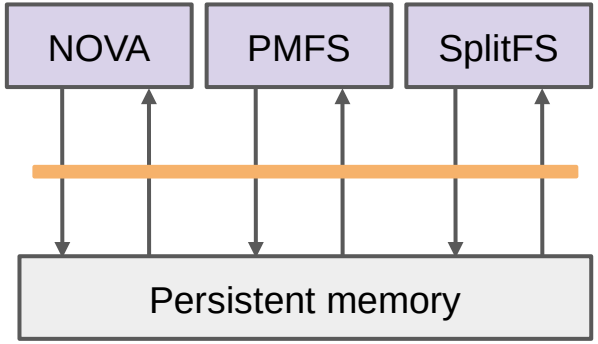


# Recording updates

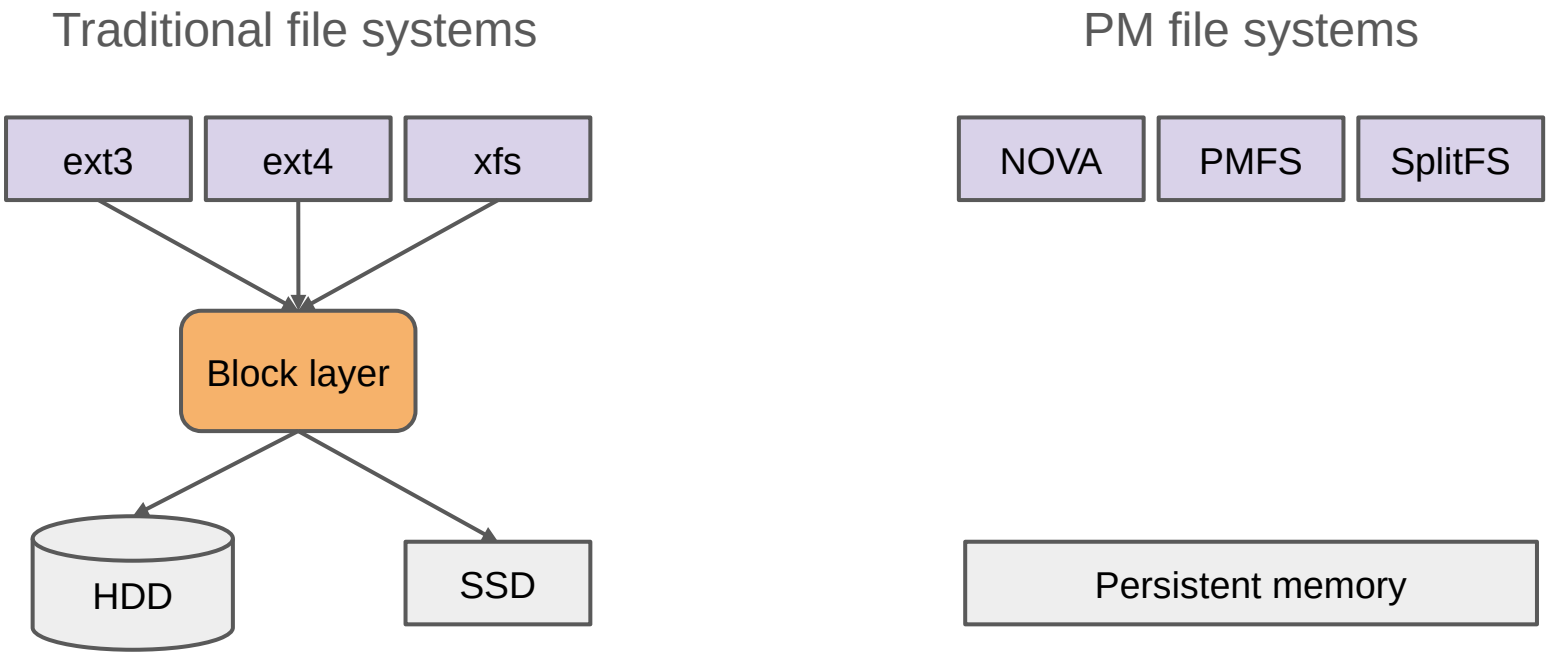
Traditional file systems




PM file systems

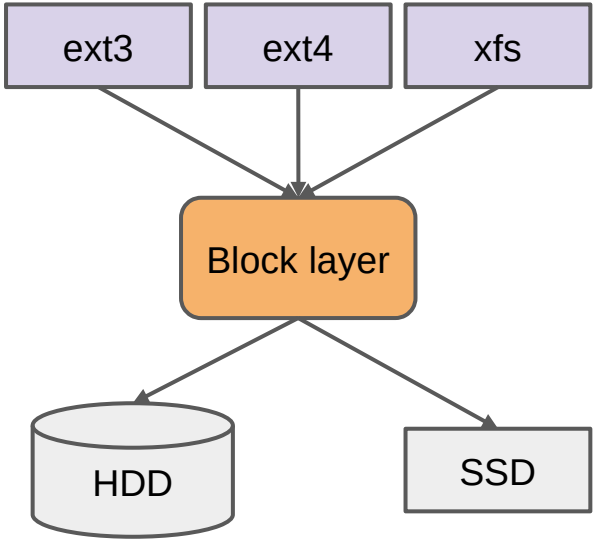


# Recording updates



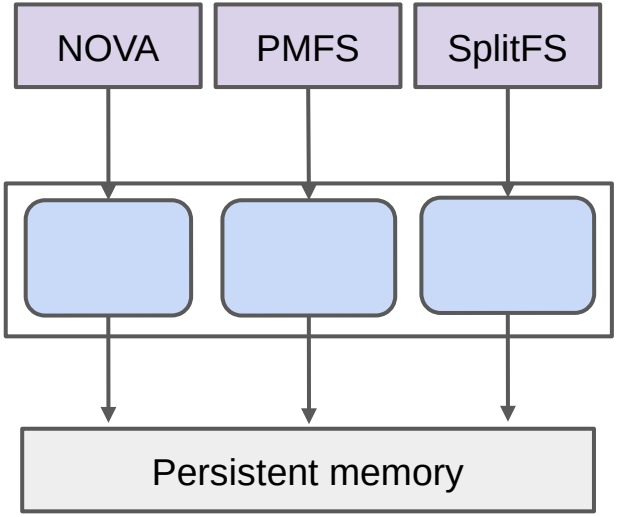
# Recording updates

Traditional file systems




PM file systems

Centralized persistence functions



# Outline

1. Introduction
2. Chipmunk overview
3. **Experiments and bugs**
4. Observations and insights

# Experiments

# Experiments

- Tested **NOVA** (FAST ‘16), **NOVA-Fortis** (SOSP ‘17), **PMFS** (EuroSys ‘14), **SplitFS** (SOSP ‘19), **WineFS** (SOSP ‘21), ext4-DAX, xfs-DAX

# Experiments

- Tested **NOVA** (FAST ‘16), **NOVA-Fortis** (SOSP ‘17), **PMFS** (EuroSys ‘14), **SplitFS** (SOSP ‘19), **WineFS** (SOSP ‘21), ext4-DAX, xfs-DAX

# Experiments

- Tested **NOVA** (FAST ‘16), **NOVA-Fortis** (SOSP ‘17), **PMFS** (EuroSys ‘14), **SplitFS** (SOSP ‘19), **WineFS** (SOSP ‘21), ext4-DAX, xfs-DAX

- Two workload generation techniques

# Experiments

- Tested **NOVA** (FAST ‘16), **NOVA-Fortis** (SOSP ‘17), **PMFS** (EuroSys ‘14), **SplitFS** (SOSP ‘19), **WineFS** (SOSP ‘21), ext4-DAX, xfs-DAX

- Two workload generation techniques

1. **ACE**: systematic bounded test generator for file systems

# Experiments

- Tested **NOVA** (FAST ‘16), **NOVA-Fortis** (SOSP ‘17), **PMFS** (EuroSys ‘14), **SplitFS** (SOSP ‘19), **WineFS** (SOSP ‘21), ext4-DAX, xfs-DAX

- Two workload generation techniques

1. **ACE**: systematic bounded test generator for file systems
2. **Syzkaller**: coverage-guided kernel fuzzer

# Bugs found by Chipmunk

**23 unique bugs found across 5 file systems**

- Make file system unmountable (3 bugs)
- Violate atomicity guarantees (5 bugs)
- Lose file data (6 bugs)
- Violate synchrony guarantees (3 bugs)
- …

# Bugs found by Chipmunk

**23 unique bugs found across 5 file systems**

- Make file system unmountable (3 bugs)
- Violate atomicity guarantees (5 bugs)
- Lose file data (6 bugs)
- Violate synchrony guarantees (3 bugs)
- …

**What can we learn from these bugs?**

# Outline

1. Introduction
2. Chipmunk overview
3. Experiments and bugs
4. **Observations and insights**

# Most bugs are logic bugs

# Most bugs are logic bugs

- Prior work on PM testing focuses on low-level cache management errors

# Most bugs are logic bugs

- Prior work on PM testing focuses on low-level cache management errors
- **19/23** bugs found by Chipmunk are caused by higher-level logic bugs

# Most bugs are logic bugs

- Prior work on PM testing focuses on low-level cache management errors
- **19/23** bugs found by Chipmunk are caused by higher-level logic bugs
- 15/23 are related to in-place updates or rebuilding lost volatile state

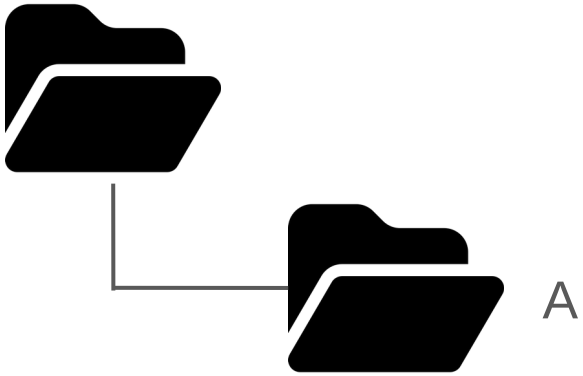
# Most bugs are logic bugs

- Prior work on PM testing focuses on low-level cache management errors
- **19/23** bugs found by Chipmunk are caused by higher-level logic bugs
- 15/23 are related to in-place updates or rebuilding lost volatile state

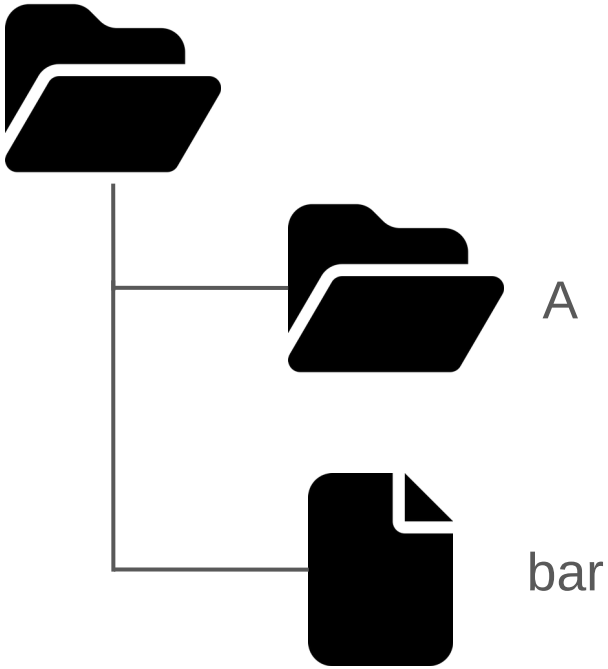
**Implication**: PM file system crash-consistency testing tools must check high-level consistency properties that cannot be validated at the level of individual writes

# Link atomicity bug in NOVA

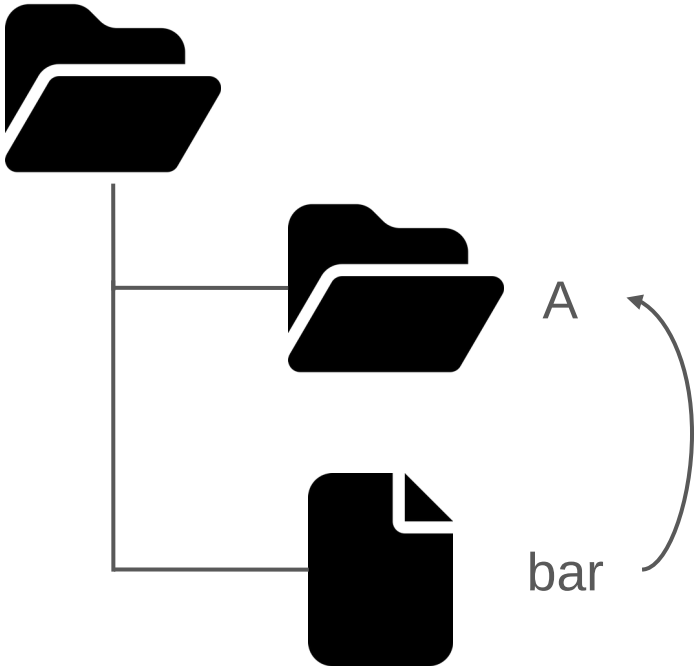
# Link atomicity bug in NOVA



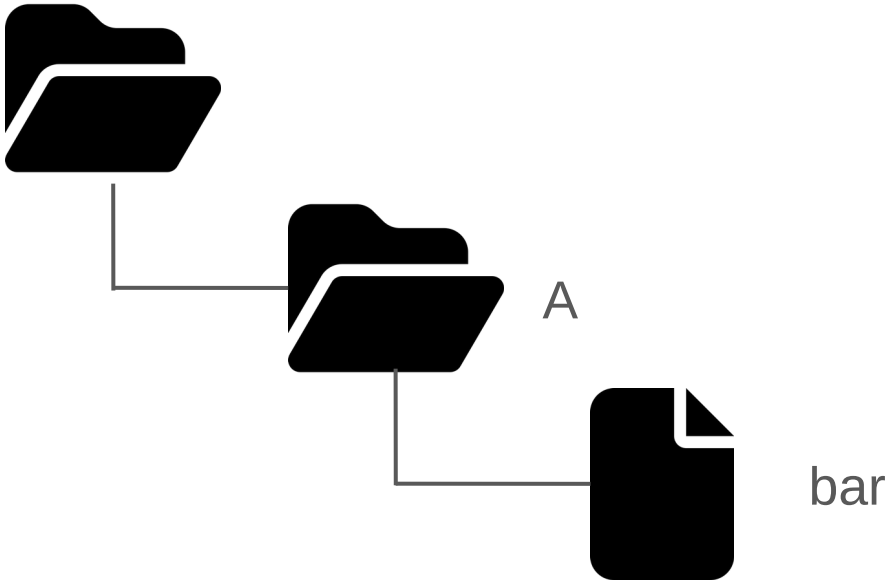
# Link atomicity bug in NOVA



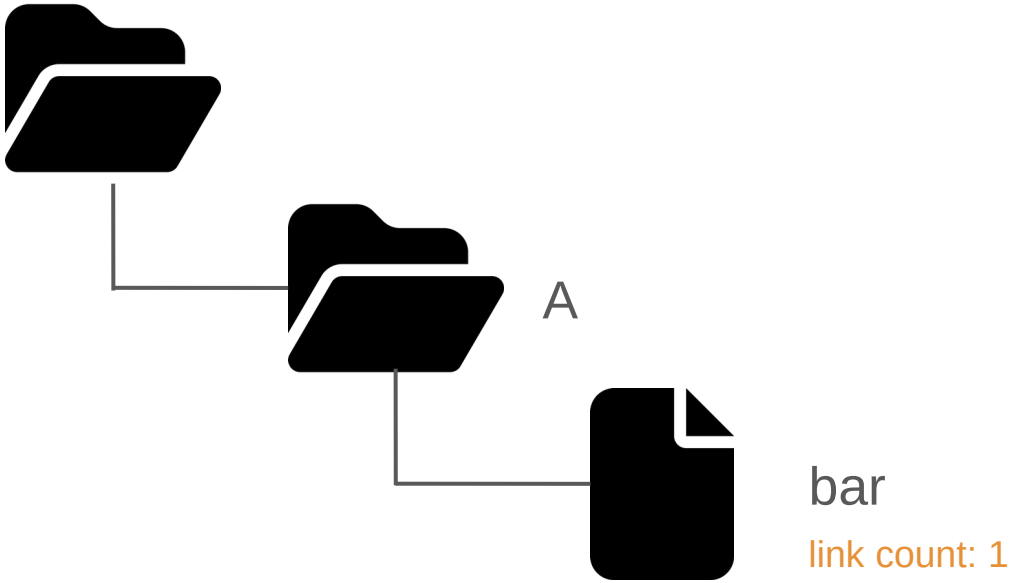
# Link atomicity bug in NOVA



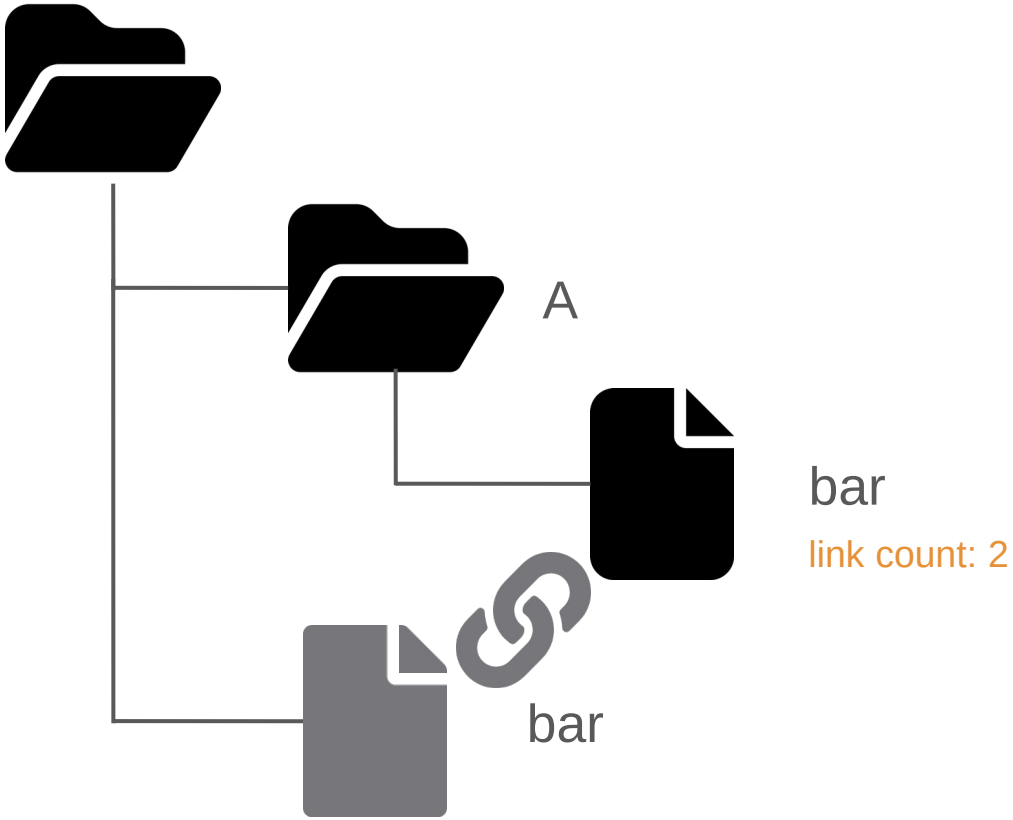
# Link atomicity bug in NOVA



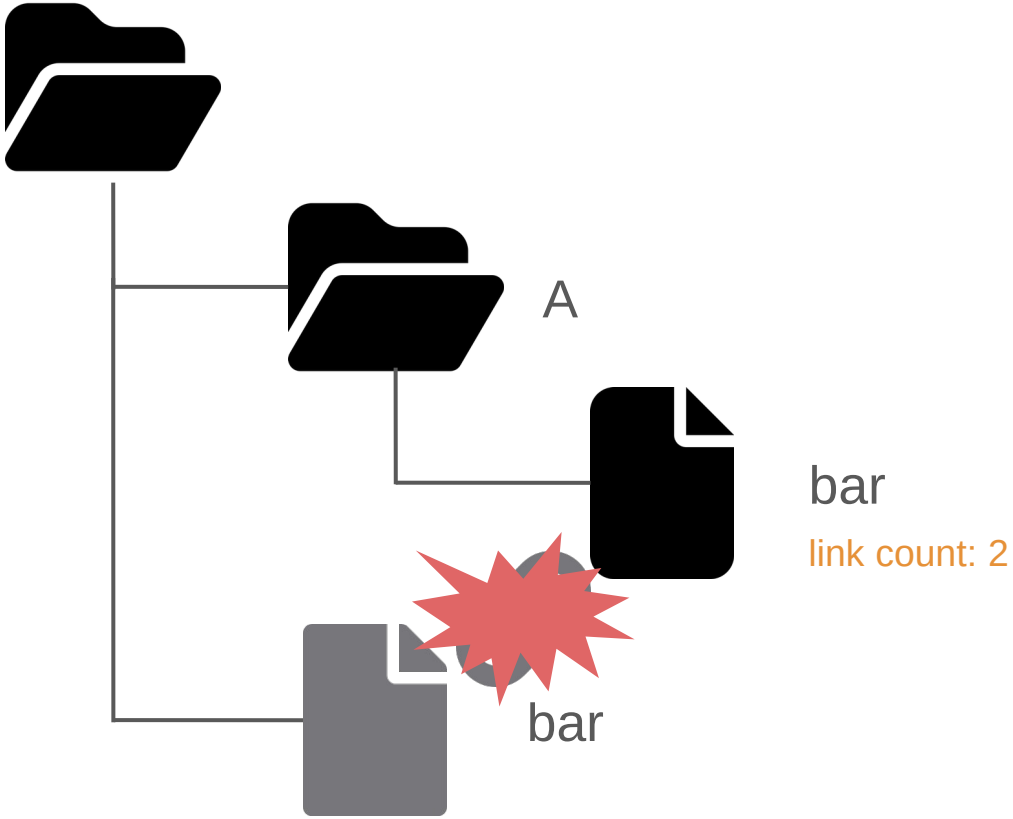
# Link atomicity bug in NOVA



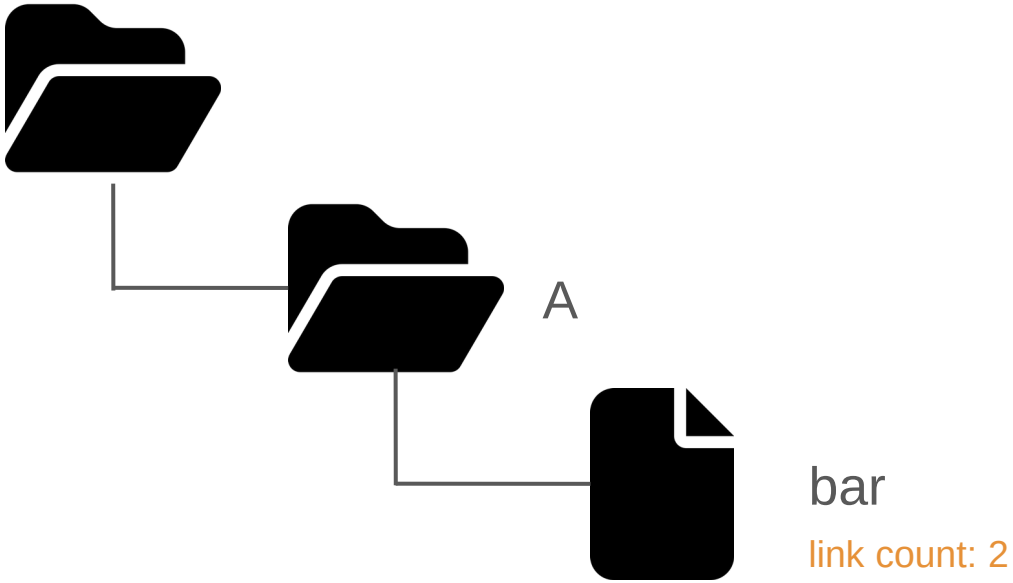
# Link atomicity bug in NOVA



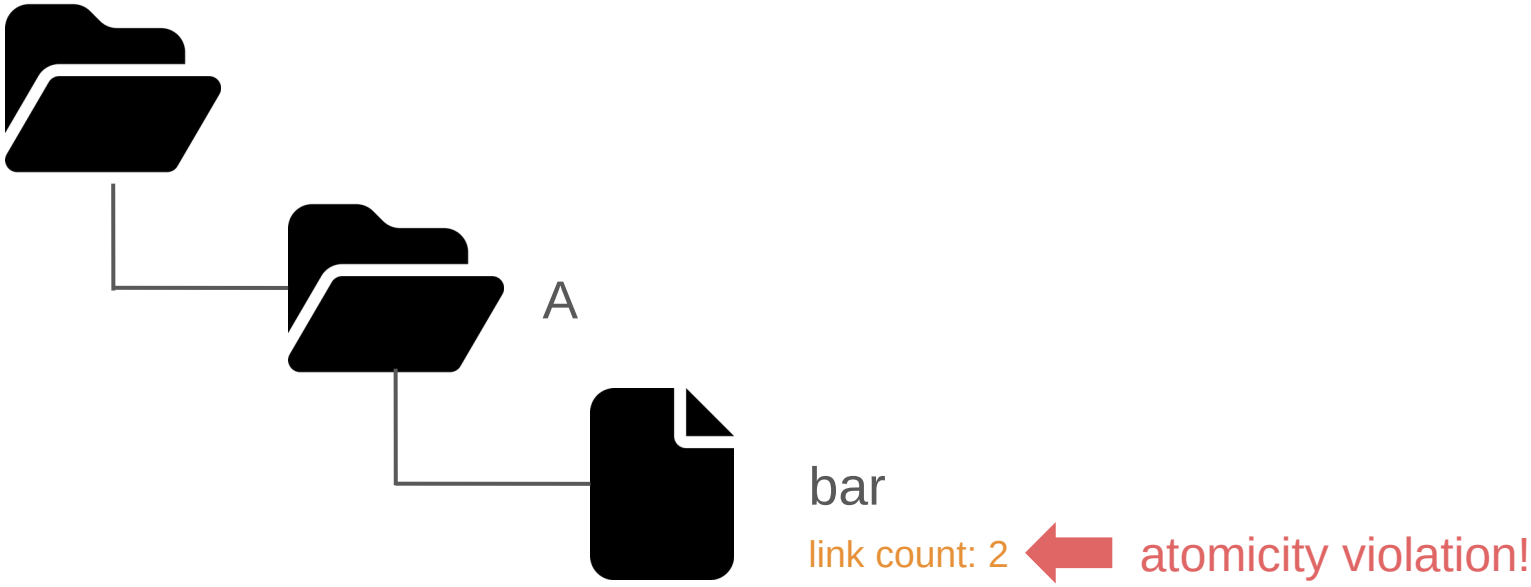
# Link atomicity bug in NOVA



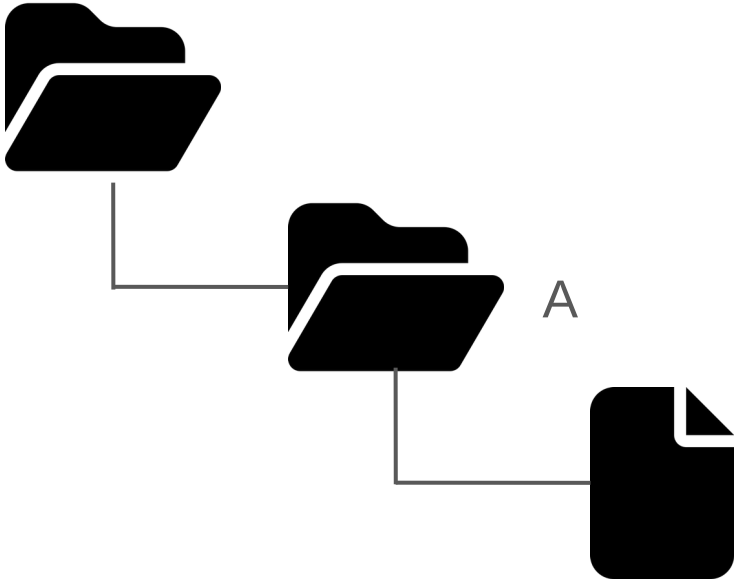
# Link atomicity bug in NOVA



# Link atomicity bug in NOVA



# Link atomicity bug in NOVA




atomicity violation!

**Root cause: in-place update optimization to avoid journaling**

# Exposing bugs requires checking more states

# Exposing bugs requires checking more states

- **11/23** bugs **require** a crash while a system call is executing

# Exposing bugs requires checking more states

- **11/23** bugs **require** a crash while a system call is executing
- Prior work only checks crash states after fsync…

# Exposing bugs requires checking more states

- **11/23** bugs **require** a crash while a system call is executing
- Prior work only checks crash states after fsync…
- But fsync is a no-op in most PM file systems

# Exposing bugs requires checking more states

- **11/23** bugs **require** a crash while a system call is executing
- Prior work only checks crash states after fsync…
- But fsync is a no-op in most PM file systems

# Exposing bugs requires checking more states

- **11/23** bugs **require** a crash while a system call is executing
- Prior work only checks crash states after fsync…
- But fsync is a no-op in most PM file systems

**Implication**: In order to find all bugs, testing must cover a large number of intermediate crash states that are not checked by existing file system testing tools.

# Synchronous behavior is a double-edged sword

# Synchronous behavior is a double-edged sword

Simple semantics for users and application developers…

# Synchronous behavior is a double-edged sword

Simple semantics for users and application developers…

But difficult to build and test!

# Synchronous behavior is a double-edged sword

Simple semantics for users and application developers…

But difficult to build and test!

Maximizing performance requires exploring the complicated new design space

# Synchronous behavior is a double-edged sword

Simple semantics for users and application developers…

But difficult to build and test!

Maximizing performance requires exploring the complicated new design space

Testing must check more states and stronger properties

# Synchronous behavior is a double-edged sword

Simple semantics for users and application developers…

But difficult to build and test!

Maximizing performance requires exploring the complicated new design space

Testing must check more states and stronger properties

**PM presents a deceptively simple interface for file system development and requires new tools to help build correct and performant systems**

# Conclusion

Chipmunk found **23 new bugs** across 5 PM file systems, many of which have severe consequences

Compatible with POSIX-compliant PM file systems in both user- and kernel-space

New insights into how to test PM file systems

Try Chipmunk: https://github.com/utsaslab/chipmunk

# Extra slides

# Related work: traditional file system testing

CrashMonkey (OSDI ‘18): black-box record-and-replay crash consistency testing framework

Hydra (SOSP ‘19): file system fuzzer focusing on crash consistency and POSIX violations

eXplode (OSDI ‘06): file system model checker focusing on crash consistency

# Related work: PM file system testing

Yat (ATC ‘14): hypervisor-based tester designed for PMFS

PMTest (ASPLOS ‘19): checks durability and ordering using developer-provided annotations

Vinter (ATC ‘22): hypervisor-based tester used to test NOVA and NOVA-Fortis

# Related work: PM application testing

Pmemcheck: Valgrind-based tool developed by Intel for use with their PMDK library

XFDetector (ASPLOS ‘20): focuses on finding “cross-failure bugs” involving regular and recovery code

Agamotto (OSDI ‘20): symbolic execution tool focusing on crash consistency

PMFuzz (ASPLOS ‘21): PM program fuzzer used with Pmemcheck and XFDetector

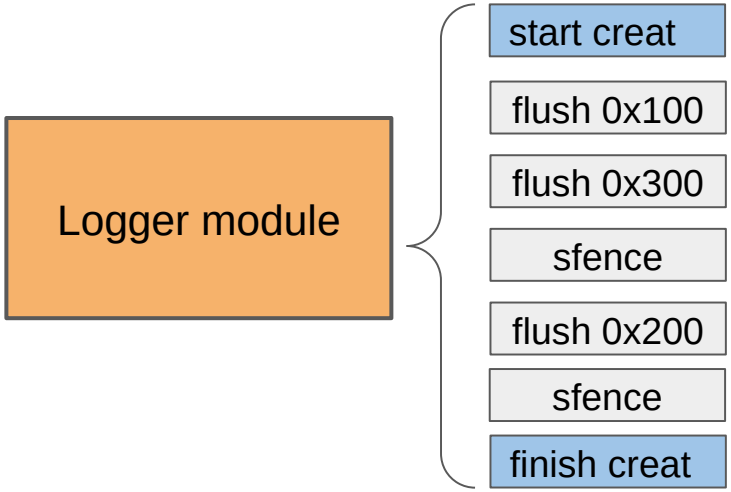
# Related work: PM application testing

Witcher (SOSP ‘21): PM key-value store tester for PM programming errors and “persistence atomicity violations”

PmDebugger (ASPLOS ‘21): tool for collecting and analyzing PM access traces

Durinn (OSDI ‘22): durable linearizability checker

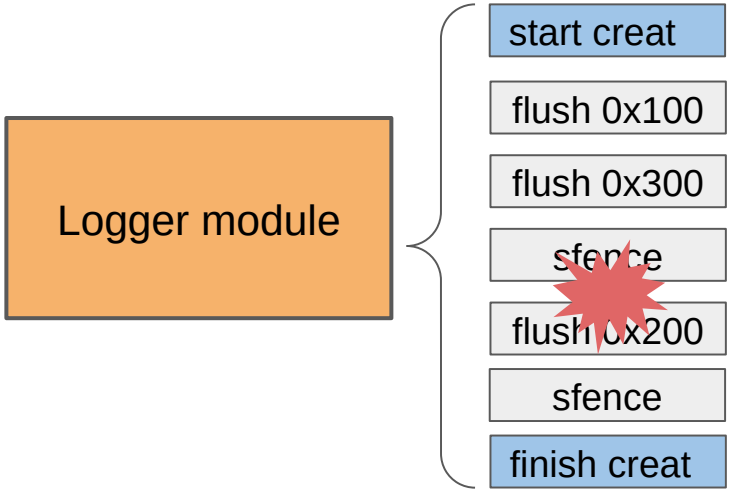
# Record and replay




recorded via IOCTL issued by Chipmunk

recorded via Kprobes/Uprobes trigger on FS function

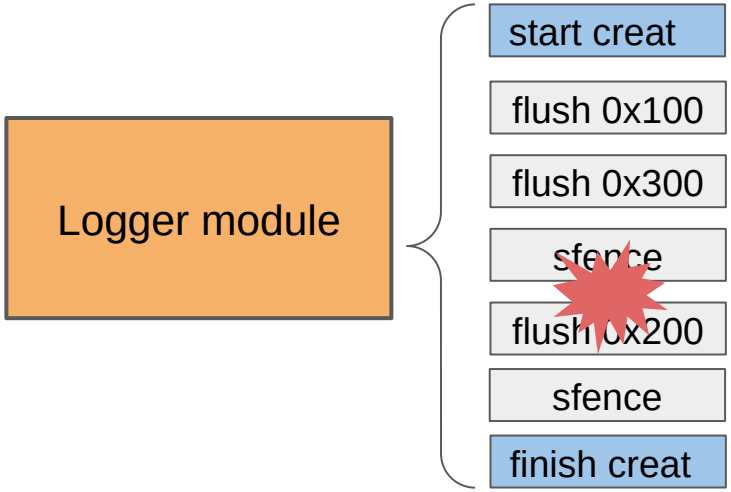
# Record and replay




recorded via IOCTL issued by Chipmunk

recorded via Kprobes/Uprobes trigger on FS function

# Record and replay

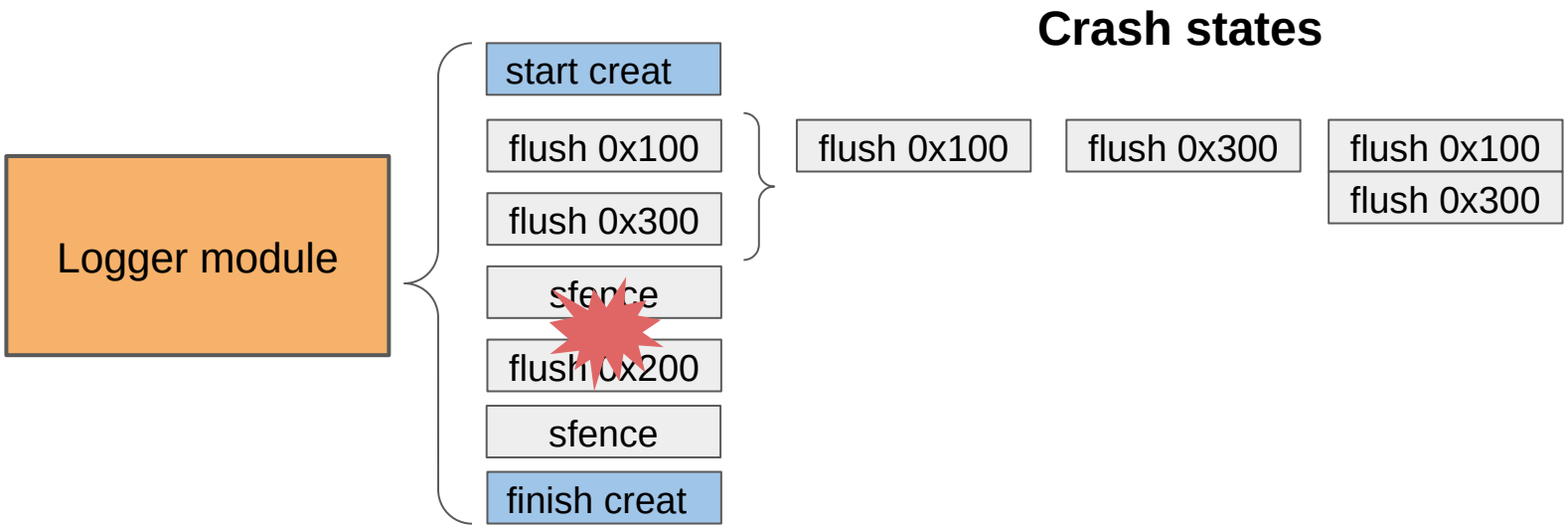



**Crash states**

recorded via IOCTL issued by Chipmunk

recorded via Kprobes/Uprobes trigger on FS function

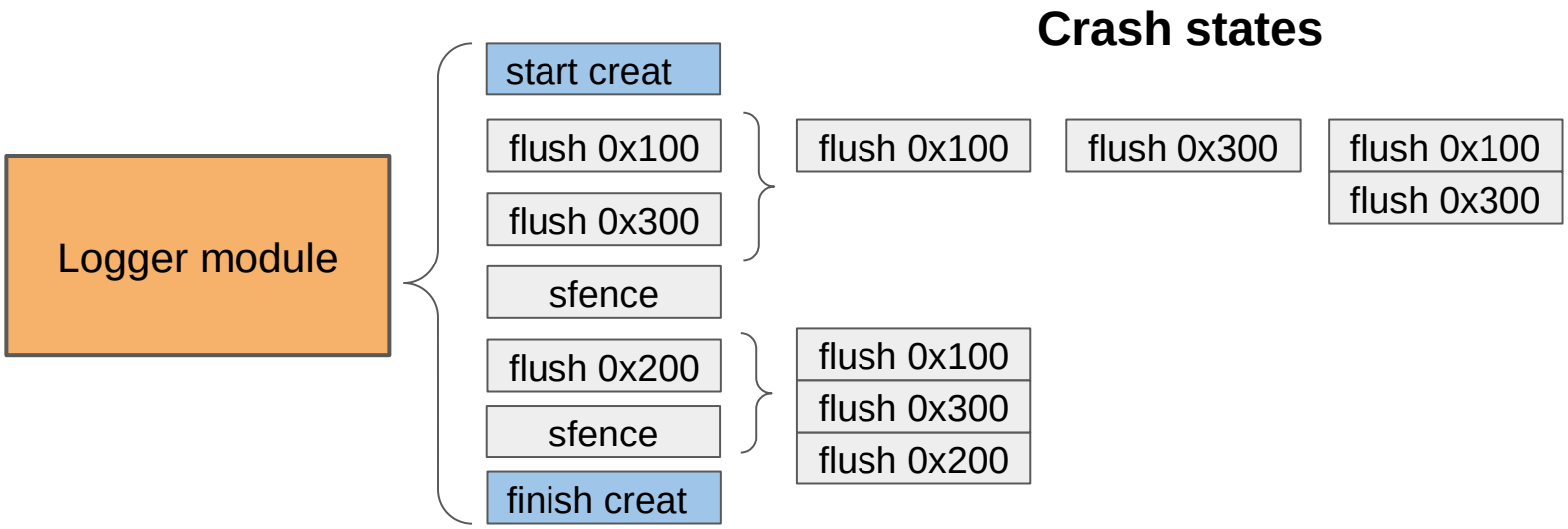
# Record and replay




recorded via IOCTL issued by Chipmunk

recorded via Kprobes/Uprobes trigger on FS function

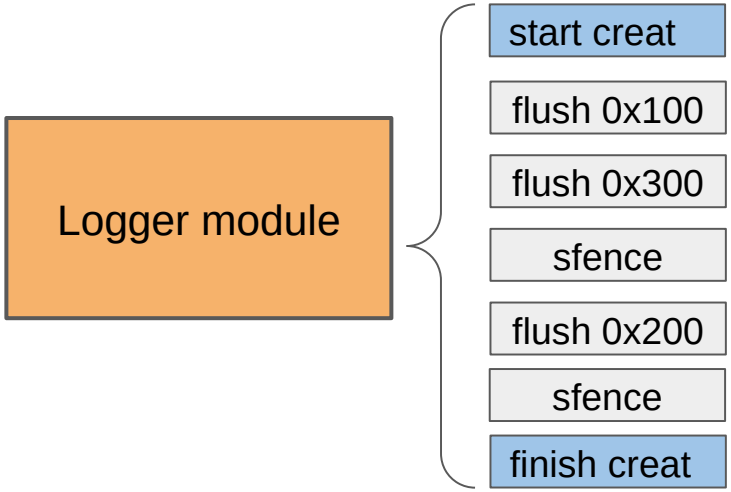
# Record and replay



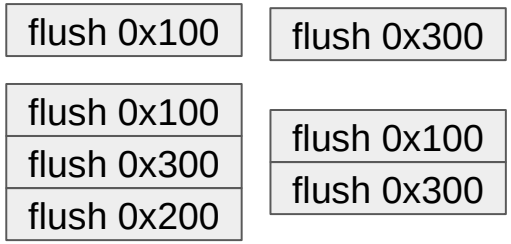

recorded via IOCTL issued by Chipmunk

recorded via Kprobes/Uprobes trigger on FS function

# Record and replay



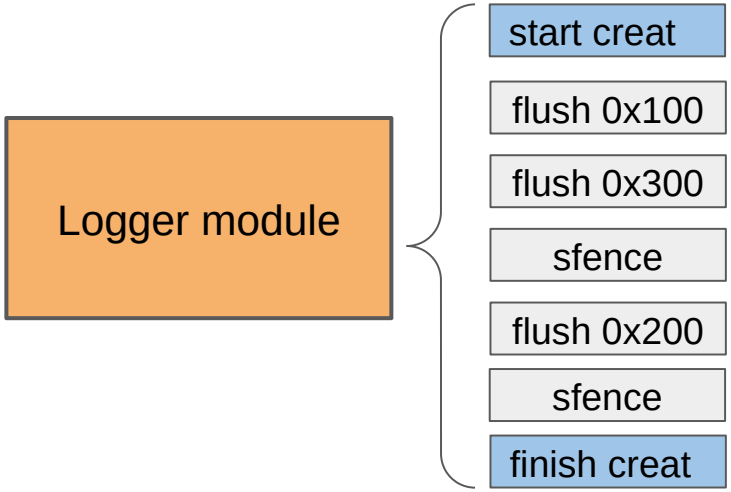

## Crash states



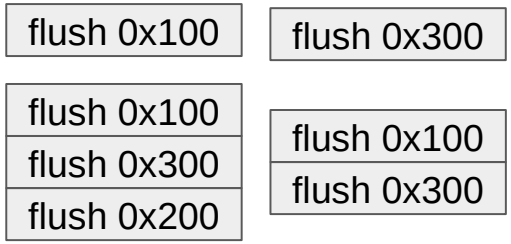

recorded via IOCTL issued by Chipmunk

recorded via Kprobes/Uprobes trigger on FS function

# Record and replay




## Crash states

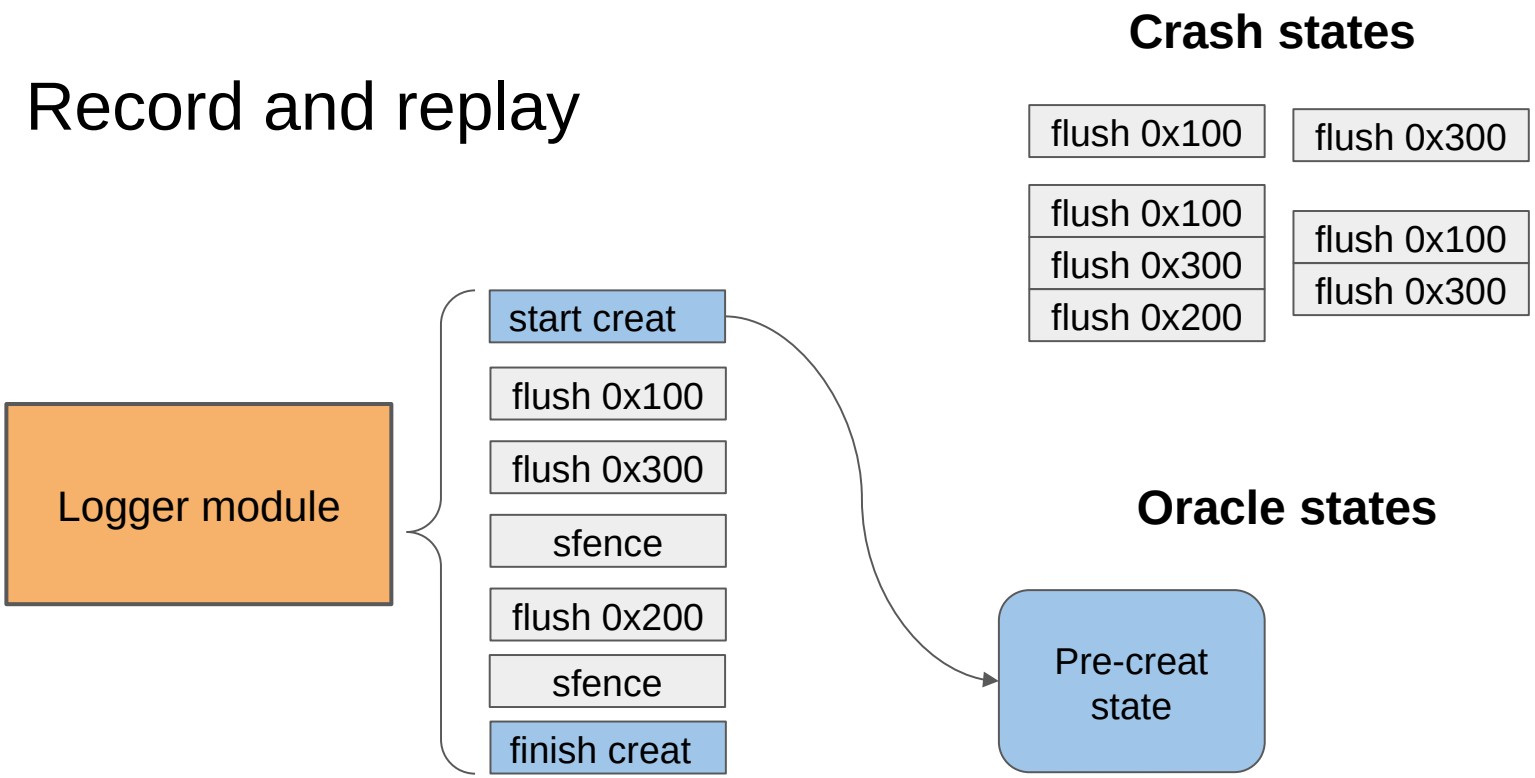



## Oracle states

recorded via IOCTL issued by Chipmunk

recorded via Kprobes/Uprobes trigger on FS function

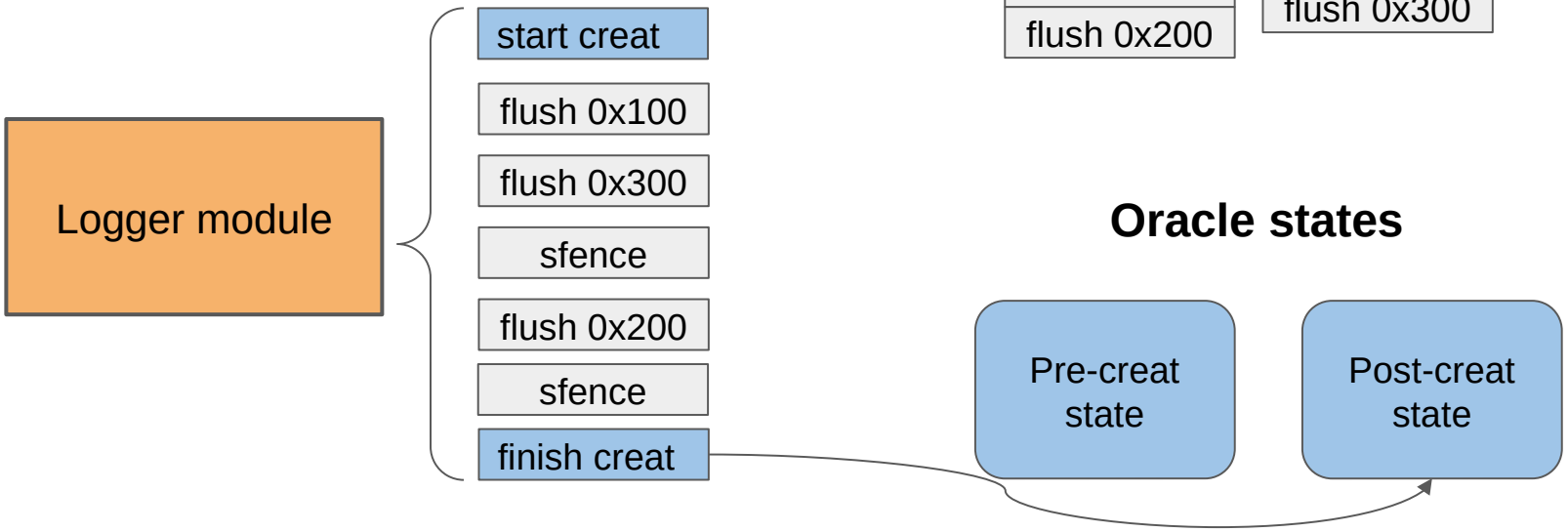
# Record and replay




recorded via IOCTL issued by Chipmunk

recorded via Kprobes/Uprobes trigger on FS function

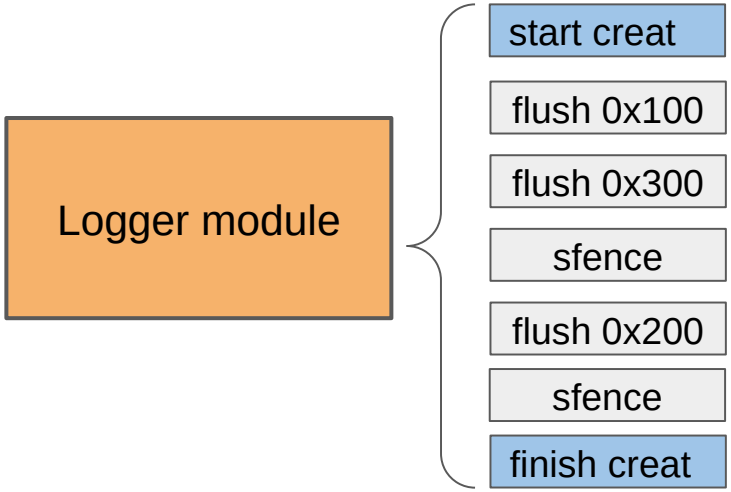
# Record and replay



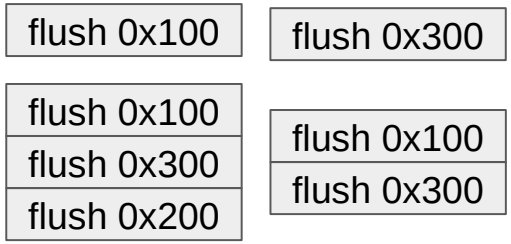

recorded via IOCTL issued by Chipmunk

recorded via Kprobes/Uprobes trigger on FS function

# Record and replay

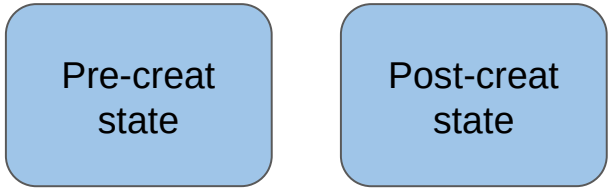



## Crash states




**vs**

## Oracle states



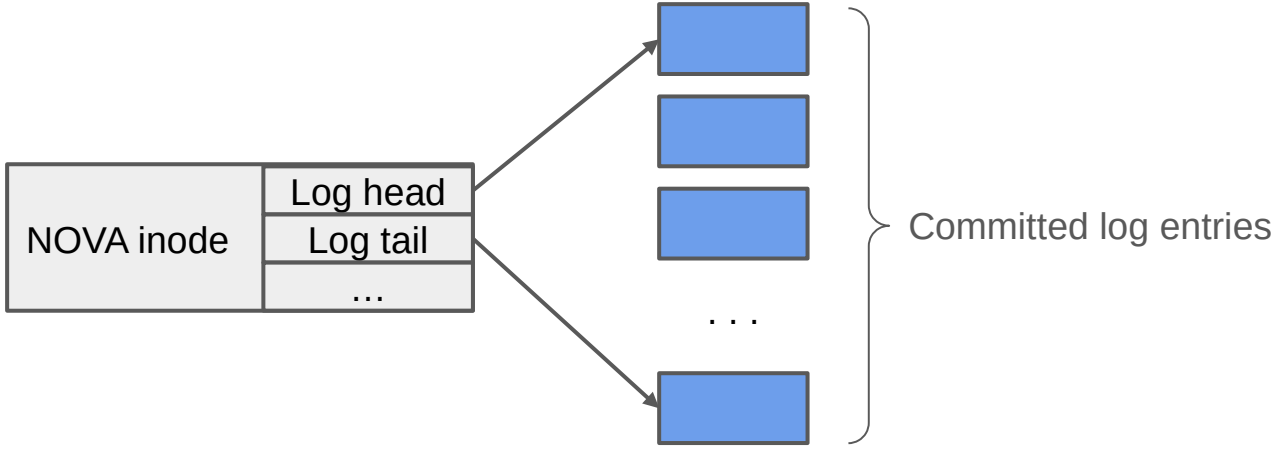

recorded via IOCTL issued by Chipmunk

recorded via Kprobes/Uprobes trigger on FS function

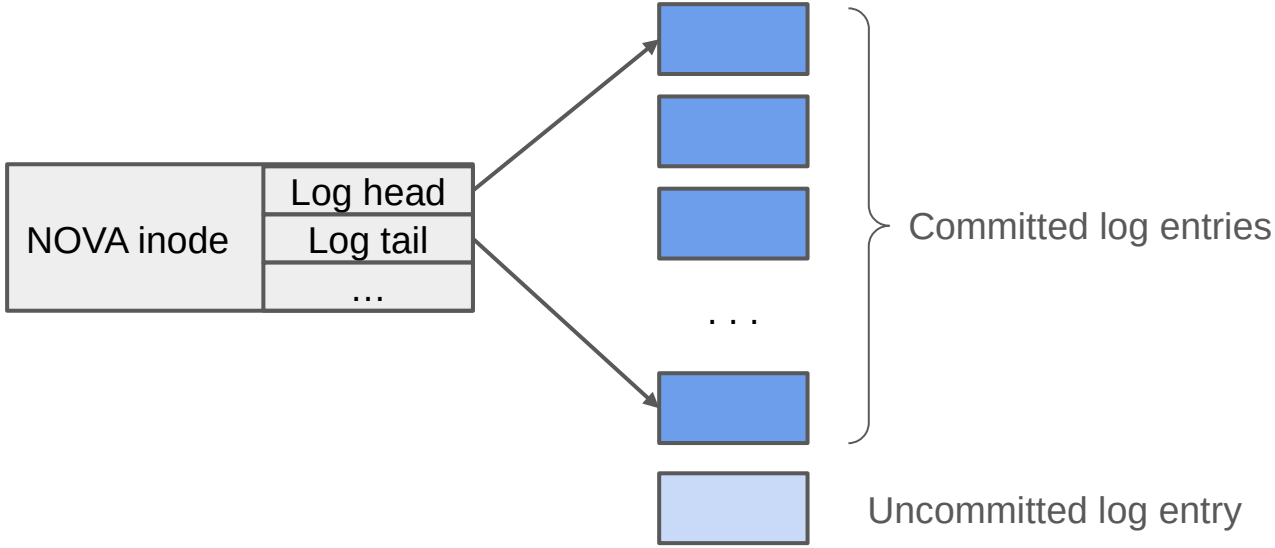
# Link atomicity bug in NOVA

| | |
|---|---|
| NOVA inode | Log head |
| | Log tail |
| | … |

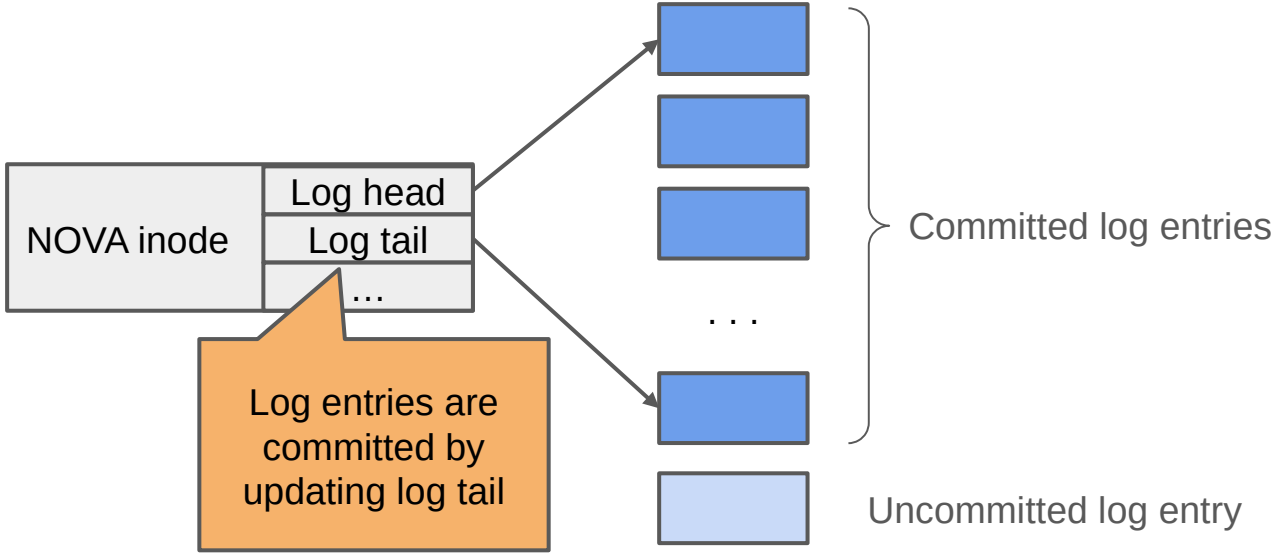
# Link atomicity bug in NOVA



# Link atomicity bug in NOVA



# Link atomicity bug in NOVA



# Link atomicity bug in NOVA

```
mkdir(“A”);
creat(“bar”);
rename(“bar”, “A/bar”);
link(“A/bar”, “bar”);
```

# Link atomicity bug in NOVA

```
mkdir("A");
creat("bar");
rename("bar", "A/bar");
link("A/bar", "bar");
```

**CRASH!!**

# Link atomicity bug in NOVA

```
mkdir(“A”);
creat(“bar”);
rename(“bar”, “A/bar”);
link(“A/bar”, “bar”);
```

**CRASH!!**

Result:

```
$ ls -l . A | awk ‘{print $2,$9}’;
2    A/bar
```

# Link atomicity bug in NOVA

```
mkdir("A");
creat("bar");
rename("bar", "A/bar");
link("A/bar", "bar");
```

**CRASH!!**

Result:

```
$ ls -l . A | awk '{print $2,$9}';
2    A/bar
```

Link count is 2, but link path bar does not exist!

# Link atomicity bug in NOVA

```
mkdir("A");
creat("bar");
rename("bar", "A/bar");
link("A/bar", "bar");
```

**CRASH!!**

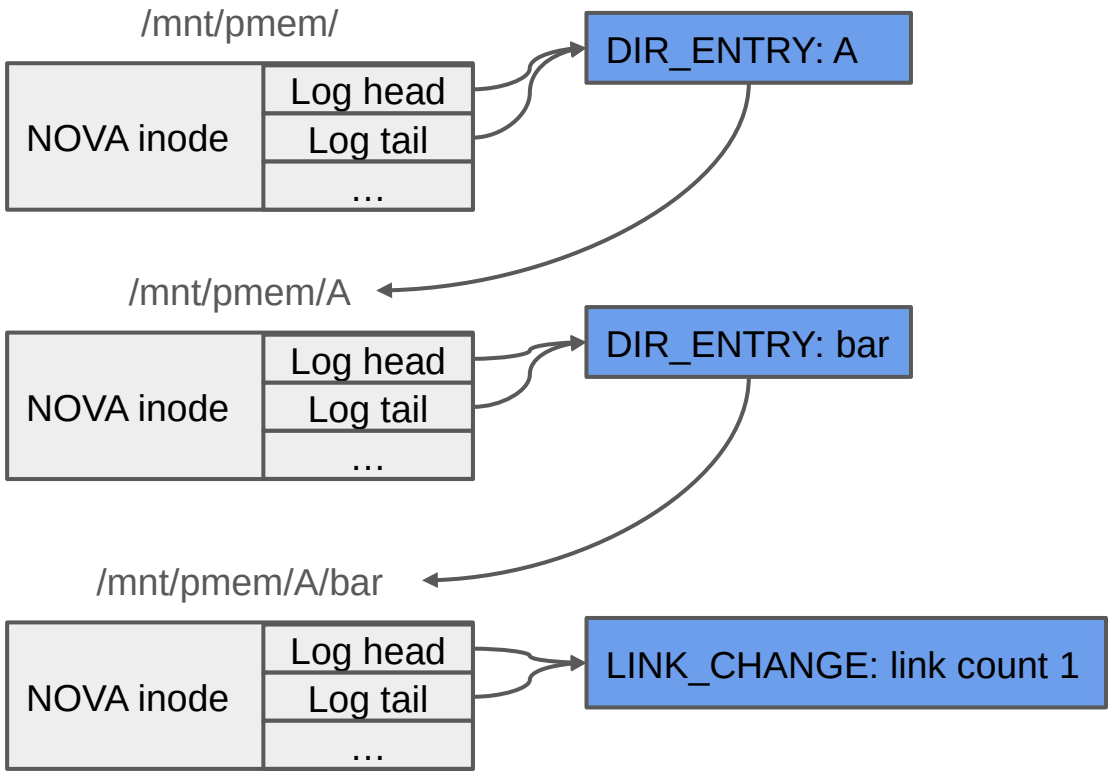
**Root cause: in-place update optimization to avoid journaling**

Result:

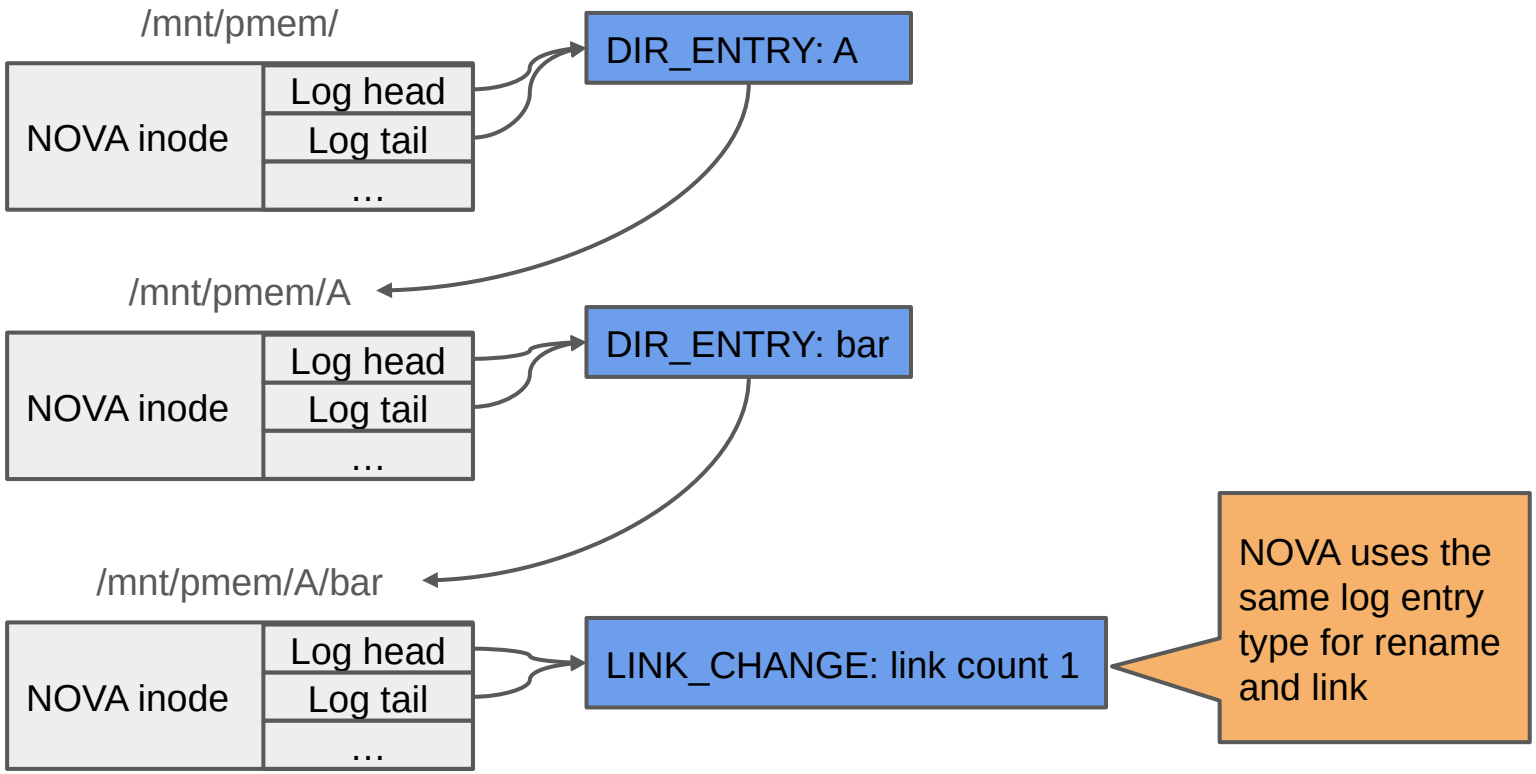
```
$ ls -l . A | awk '{print $2,$9}';
2    A/bar
```

Link count is 2, but link path bar does not exist!

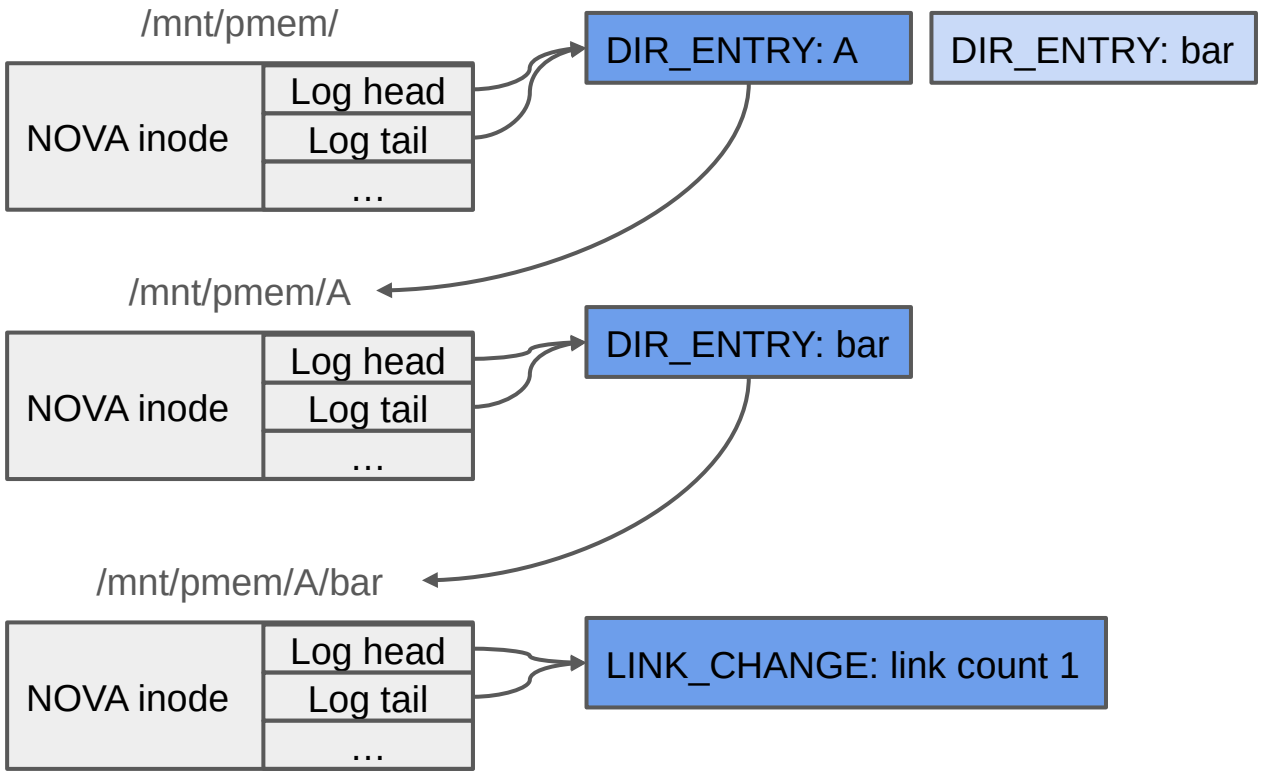
# Link atomicity bug in NOVA



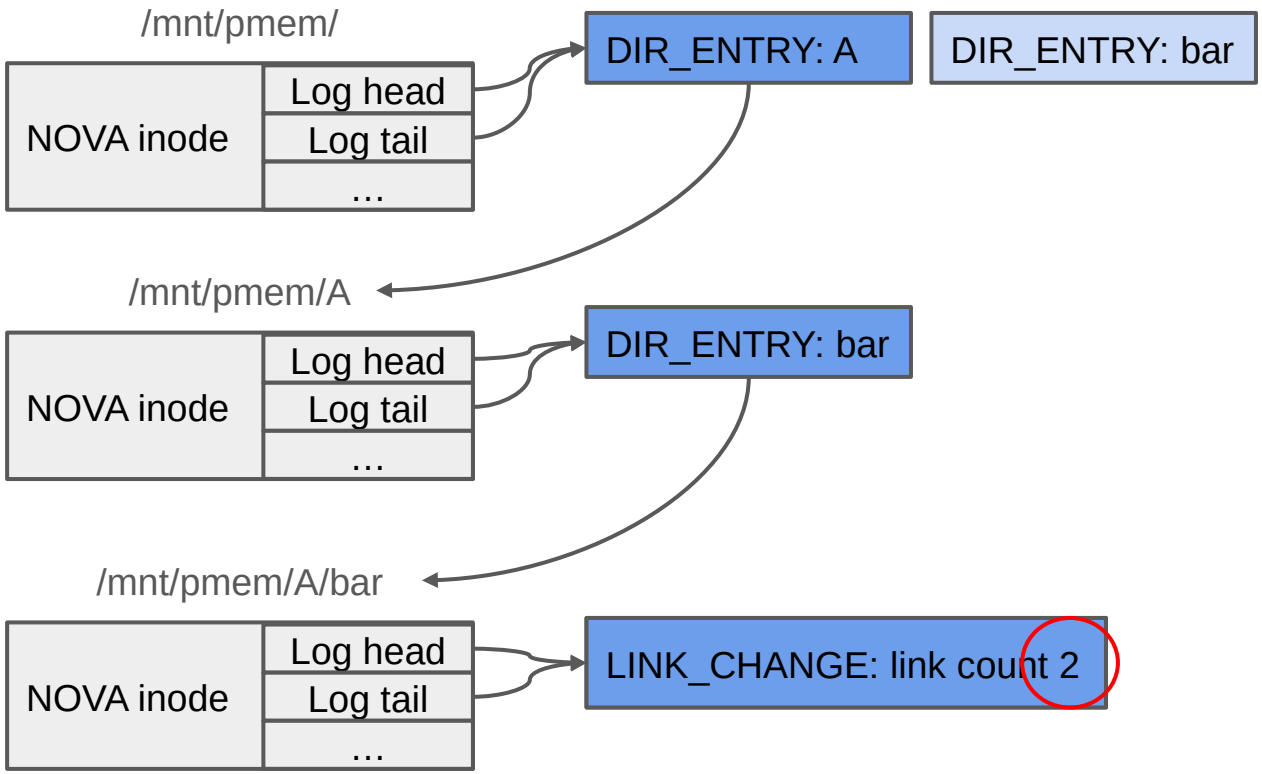
# Link atomicity bug in NOVA



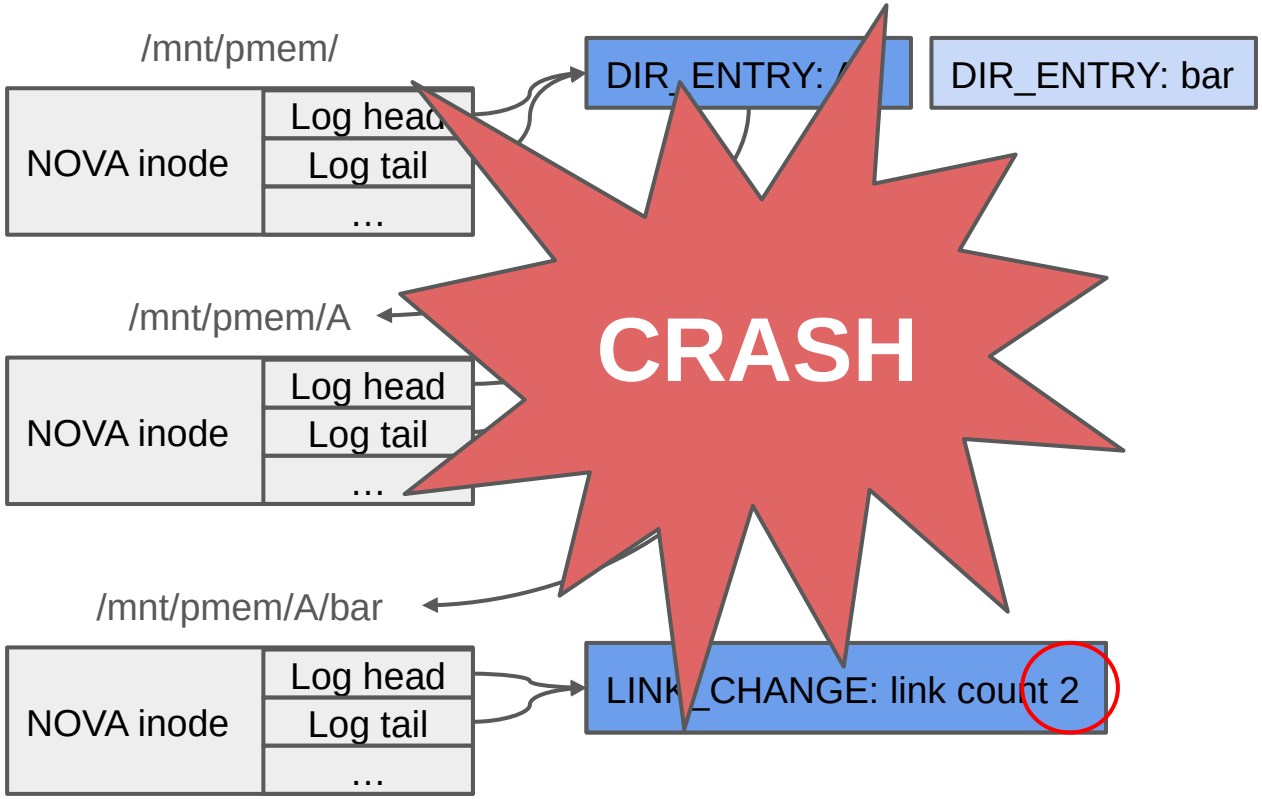
# Link atomicity bug in NOVA



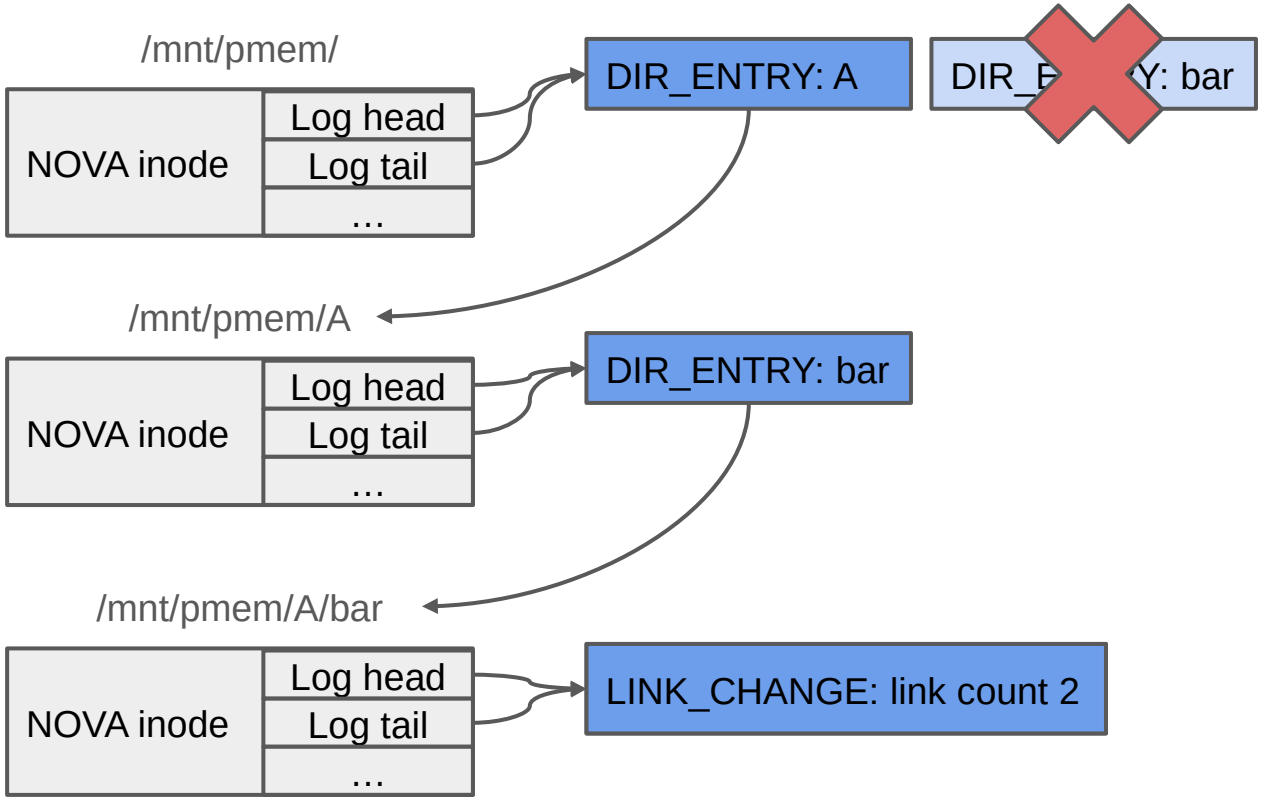
# Link atomicity bug in NOVA



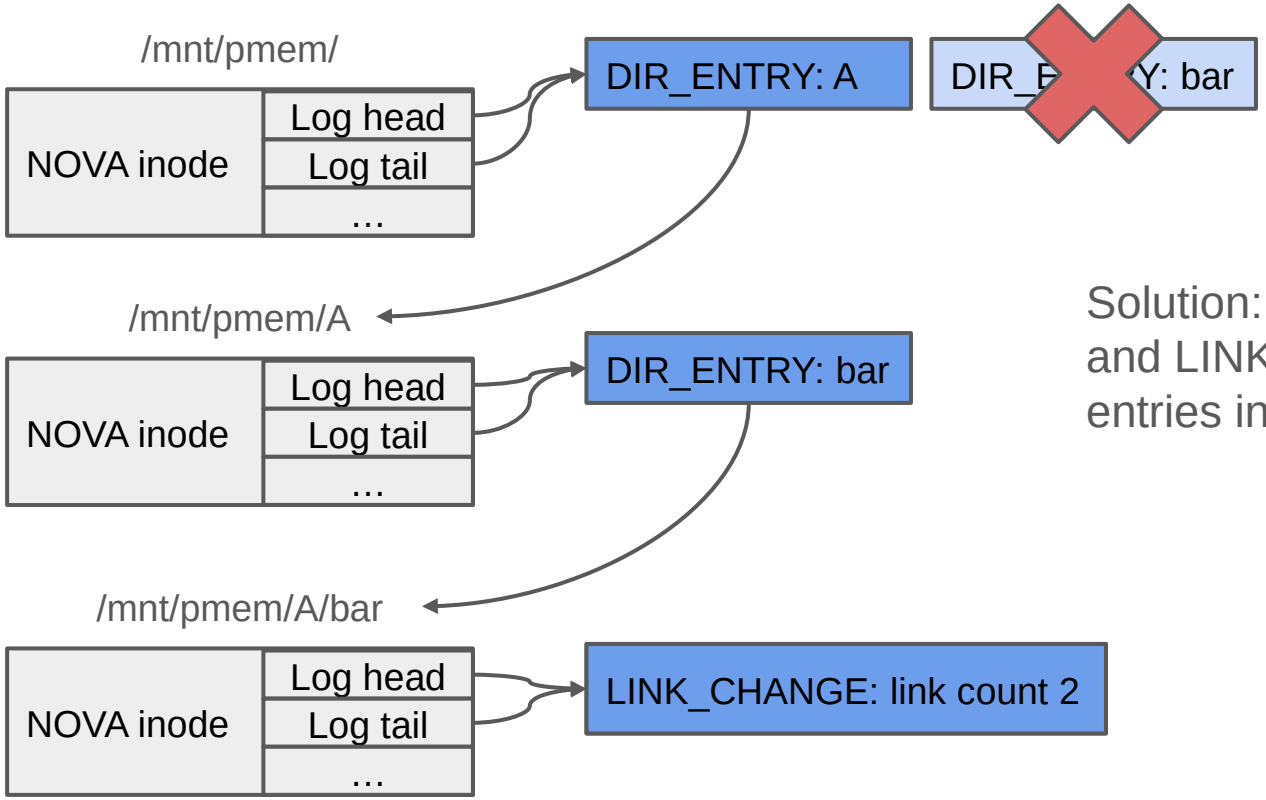
# Link atomicity bug in NOVA



# Link atomicity bug in NOVA



# Link atomicity bug in NOVA




Solution: add DIR_ENTRY and LINK_CHANGE log entries in a transaction