Understanding and Finding Crash-Consistency Bugs in Parallel File Systems

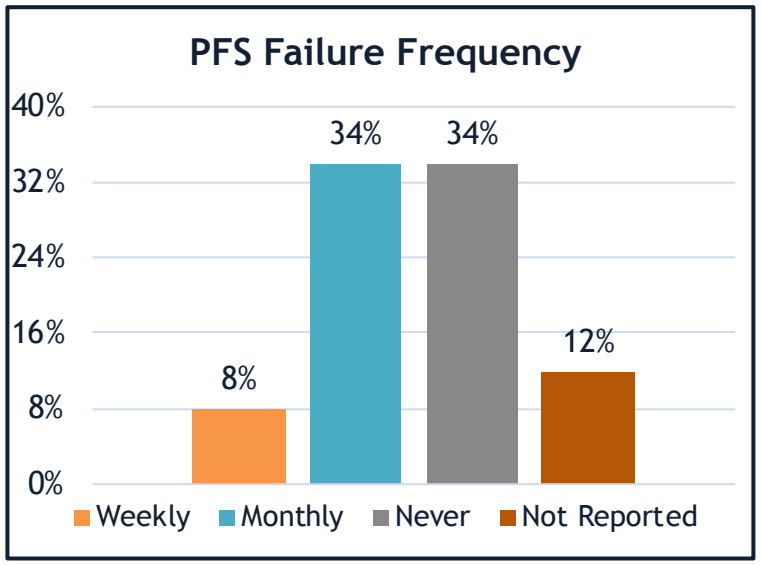
Jinghan Sun, Chen Wang, Jian Huang, and Marc Snir

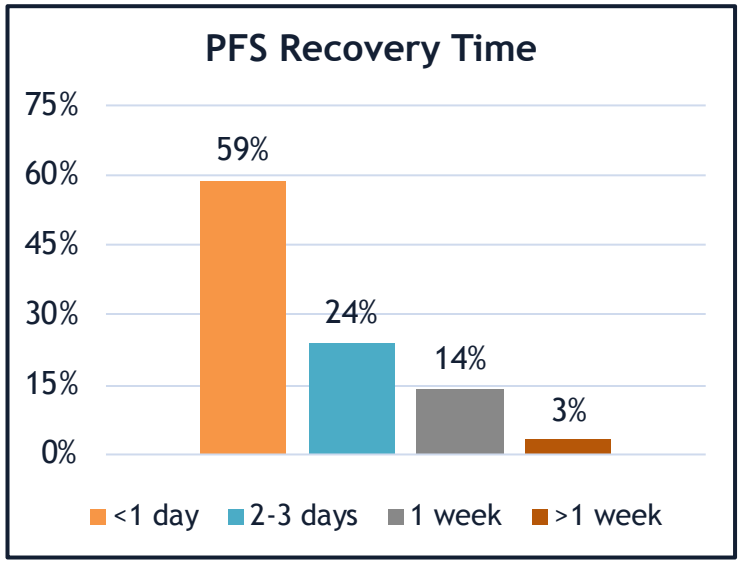
University of Illinois at Urbana-Champaign

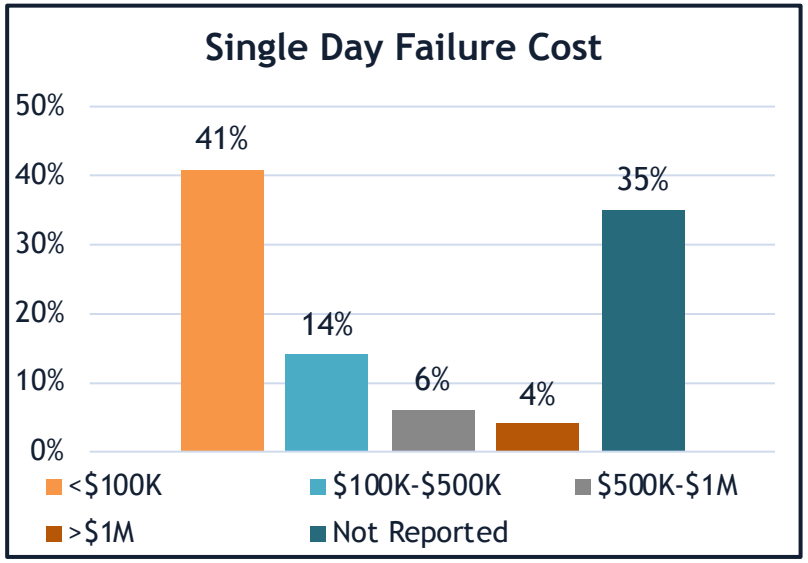
ILLINOIS

Contact: Jinghan Sun (js39@illinois.edu)

PFS failures are frequent and expensive



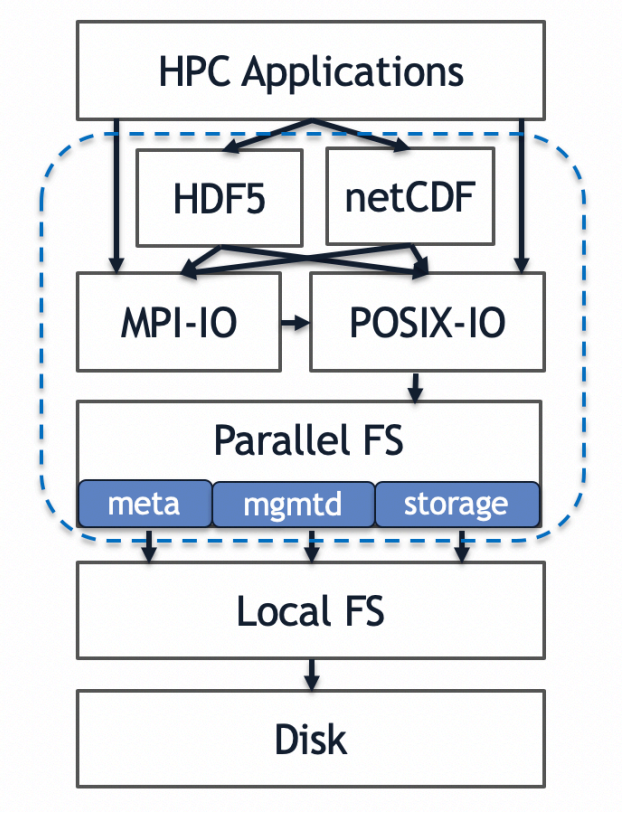




41% of PFSes suffer from monthly or weekly failures,
their recovery process is expensive & time consuming

Source: Hyperion Research 2019

Introduction to parallel file systems



Parallel I/O library

- Higher level abstractions: Datasets, groups, collective I/O APIs

Parallel file system

- Data striping
- Separate metadata management
- POSIX-compliant

HPC I/O stack is much more complex than the traditional I/O stack

A PFS failure example

To All HPCC Customers and Partners,

As we have informed you earlier, the Experimental Sciences Building experienced a major power outage Sunday, Jan. 3 and another set of outages Tuesday, Jan. 5 that occurred while file systems were being recovered from the first outage. As a result, there were major losses of important parts of the file systems for the work, scratch and certain experimental group special Lustre areas.

The HPCC staff have been working continuously since these events on recovery procedures to try to restore as much as possible of the affected file systems. These procedures are extremely time-consuming, taking days to complete in some cases. Although about a third of the affected file systems have been recovered, work continues on this effort and no time estimate is possible at present.

User home areas have been recovered successfully. At present, no user logins are being permitted while recovery efforts proceed on the remaining Lustre areas. Your understanding and patience are appreciated.

If you have questions, please contact us at hpccsupport@ttu.edu or 806-742-4350. Thanks.

Sincerely,
HPCC Staff

PFS may experience severe data loss after system-wide power outage

A study of crash vulnerabilities on PFSes

Atomic Replace via Rename

Write-ahead Logging

HDF5

- create
- delete
- rename
- resize
- update

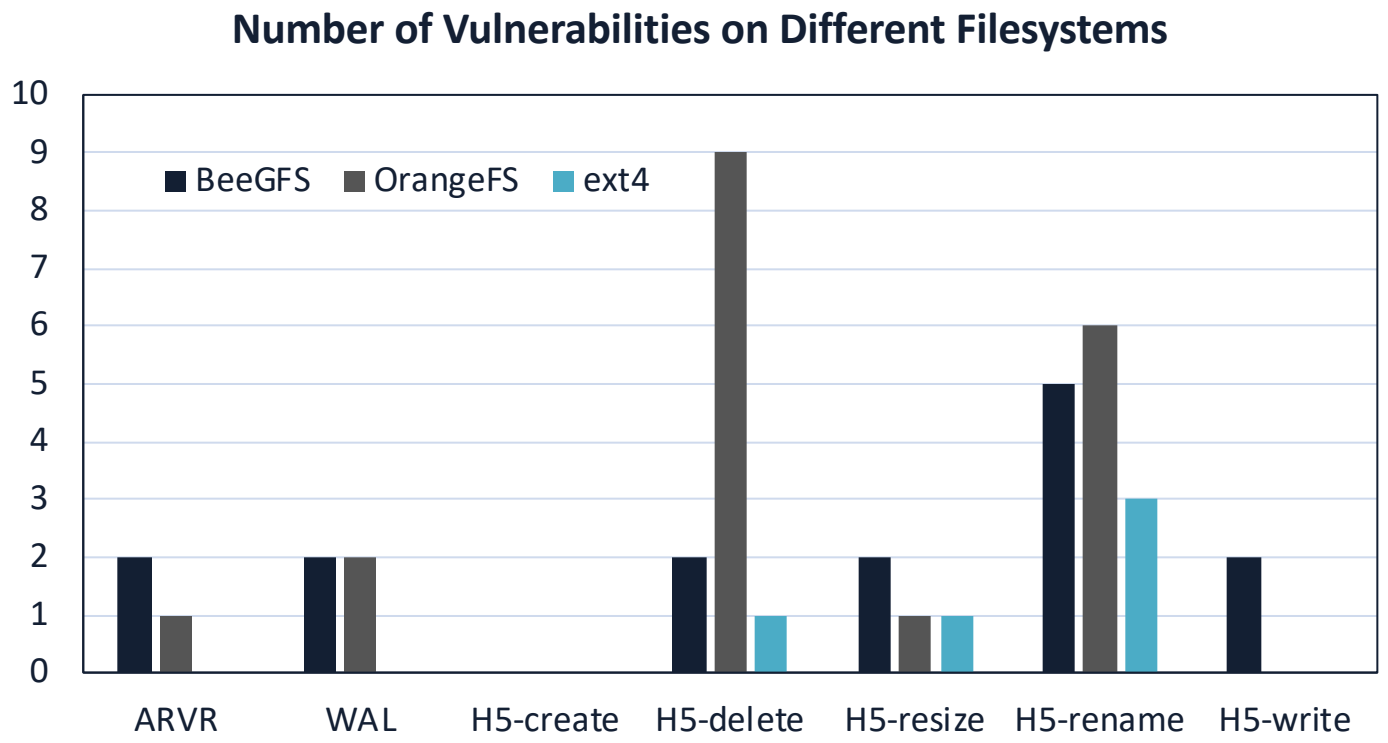
Data loss

DoS

Inaccessible dataset

Two Filesystems + Seven Workloads = 34 Vulnerabilities

PFS crash vulnerabilities



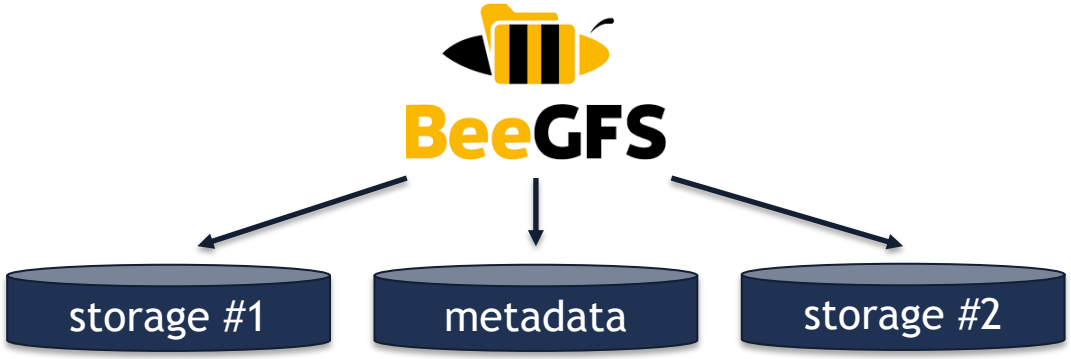
Vulnerability: Parallel I/O stack may corrupt user files if crash happens in the middle of the computation (depending on the precise timing of disk accesses)

The complexity of PFS stack makes it more vulnerable to system crashes

Crash vulnerability example

```
// atomic replace via rename (ARVR)
bool atomic_update(){
    int fd = create("file.tmp");
    write(fd, new, size);
    close(fd);
    rename("file.tmp","file.txt");
}
```

The function tries to update a file content atomically



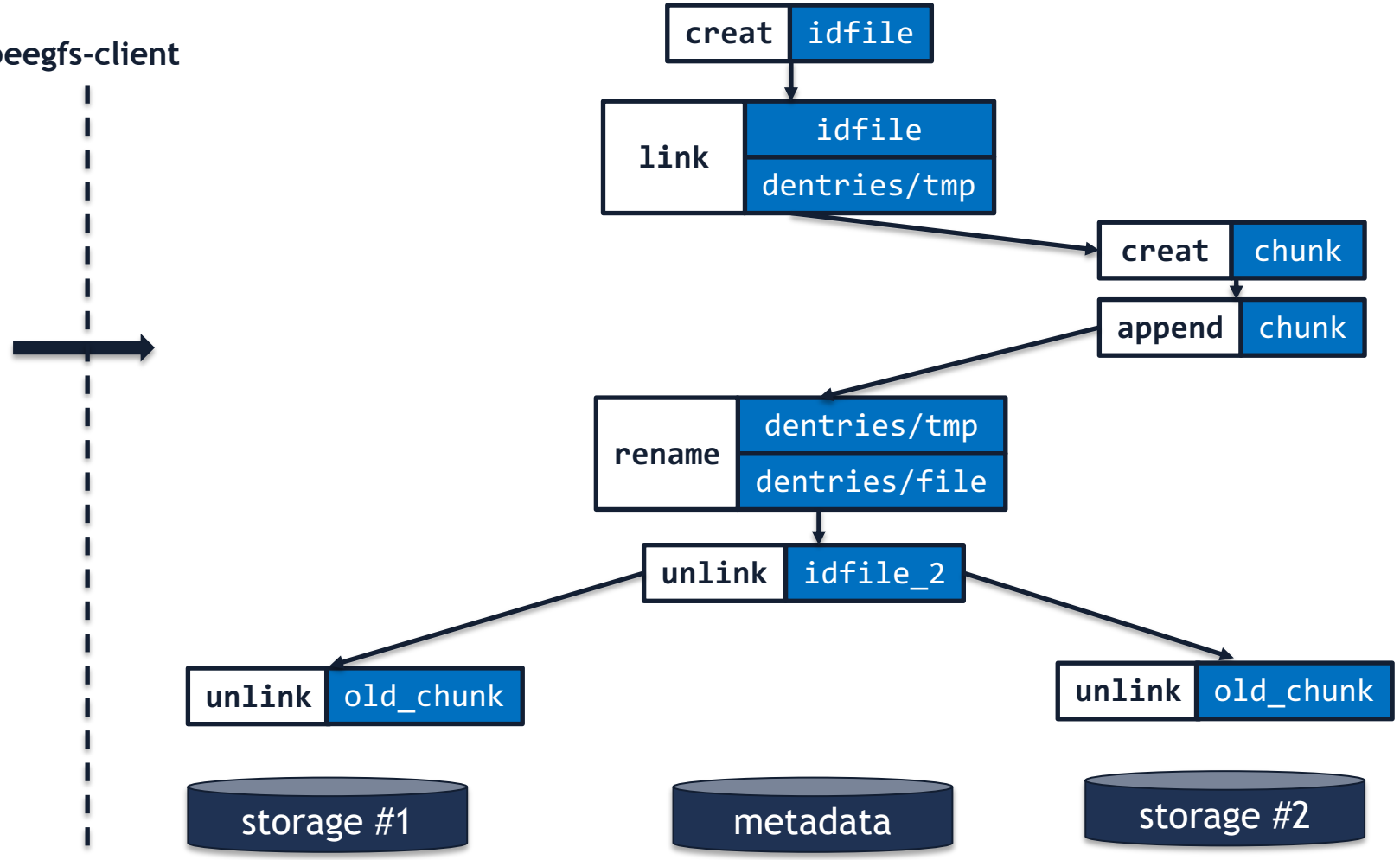
BeeGFS with 2 storage and 1 metadata server

Crash vulnerability example

```
// atomic replace via rename (ARVR)
bool atomic_update(){
    int fd = creat("file.tmp");
    write(fd, new, size);
    close(fd);
    rename("file.tmp","file.txt");
}
```

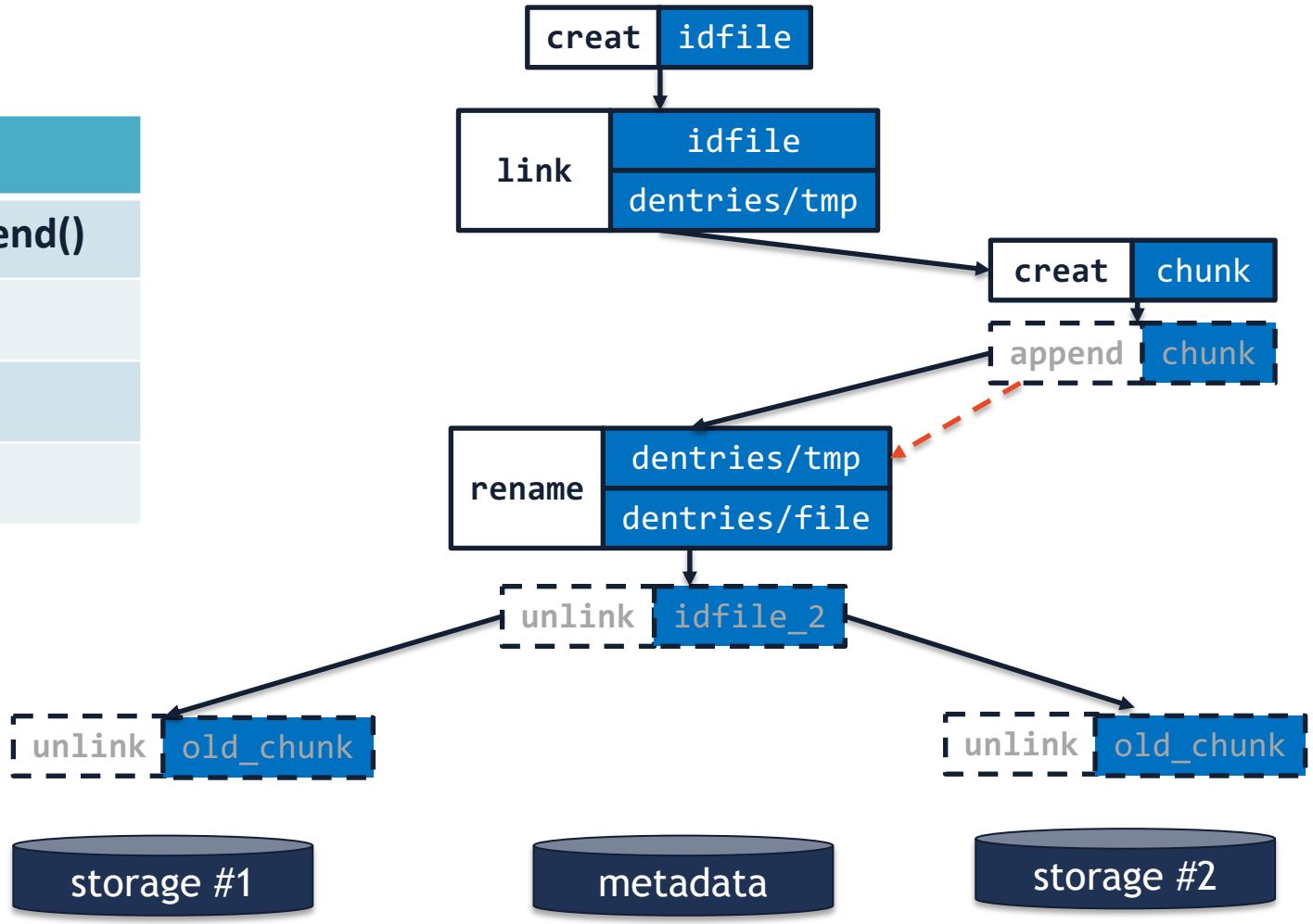
Persistence order ≠ Program order!

Two vulnerabilities discovered at system crash!



Crash vulnerability example

Inconsistency No.1	
Cause	**rename()** persisted before **append()**
Ordering	Cross-node dependency
Consequence	Data loss
Fixed by fsck?	No

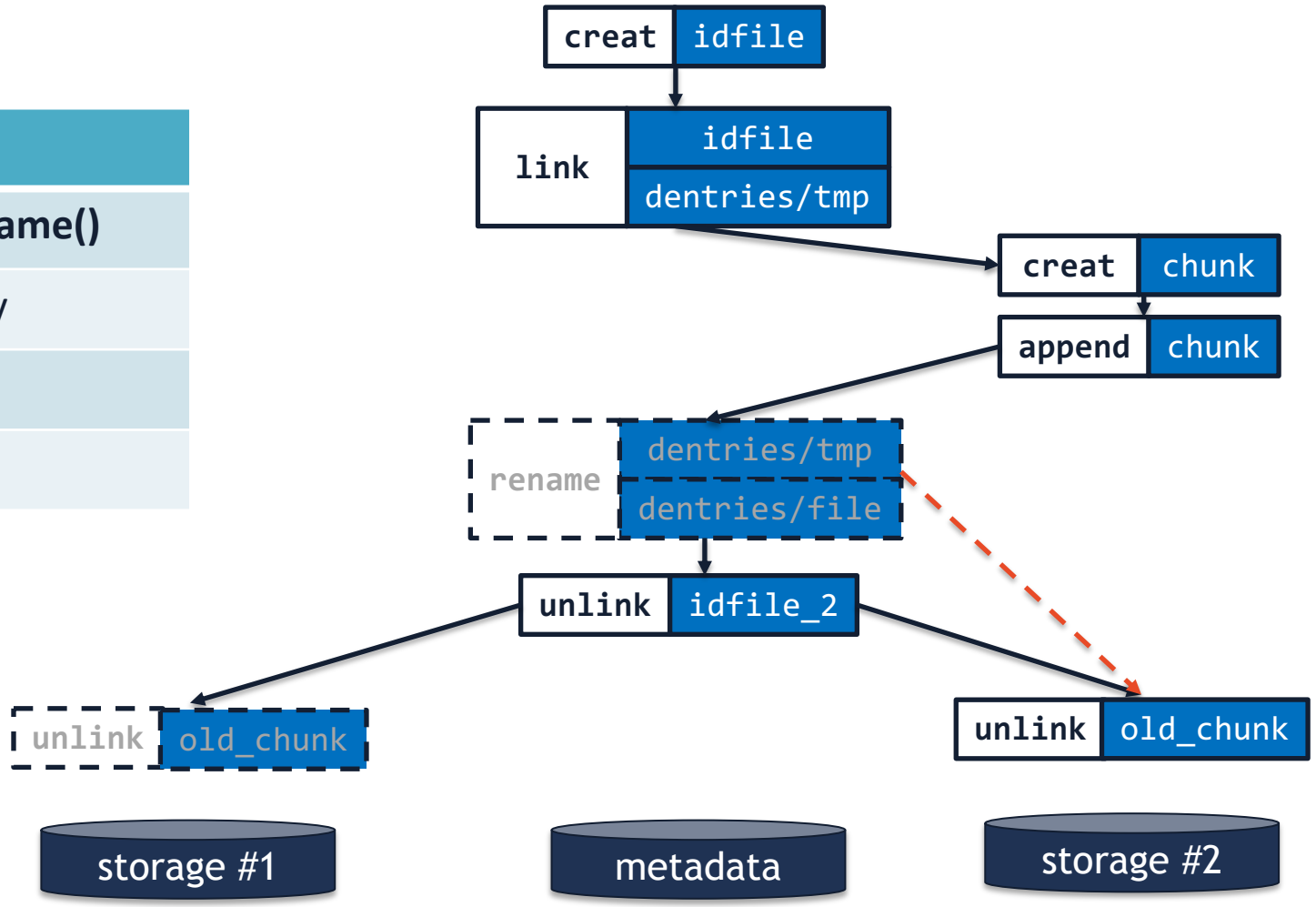


op param **Persisted operations**

op param **Non-persisted operations**

Crash vulnerability example

Inconsistency No.2	
Cause	**unlink()** persisted before **rename()**
Ordering	Cross-node dependency
Consequence	Data loss
Fixed by fsck?	No

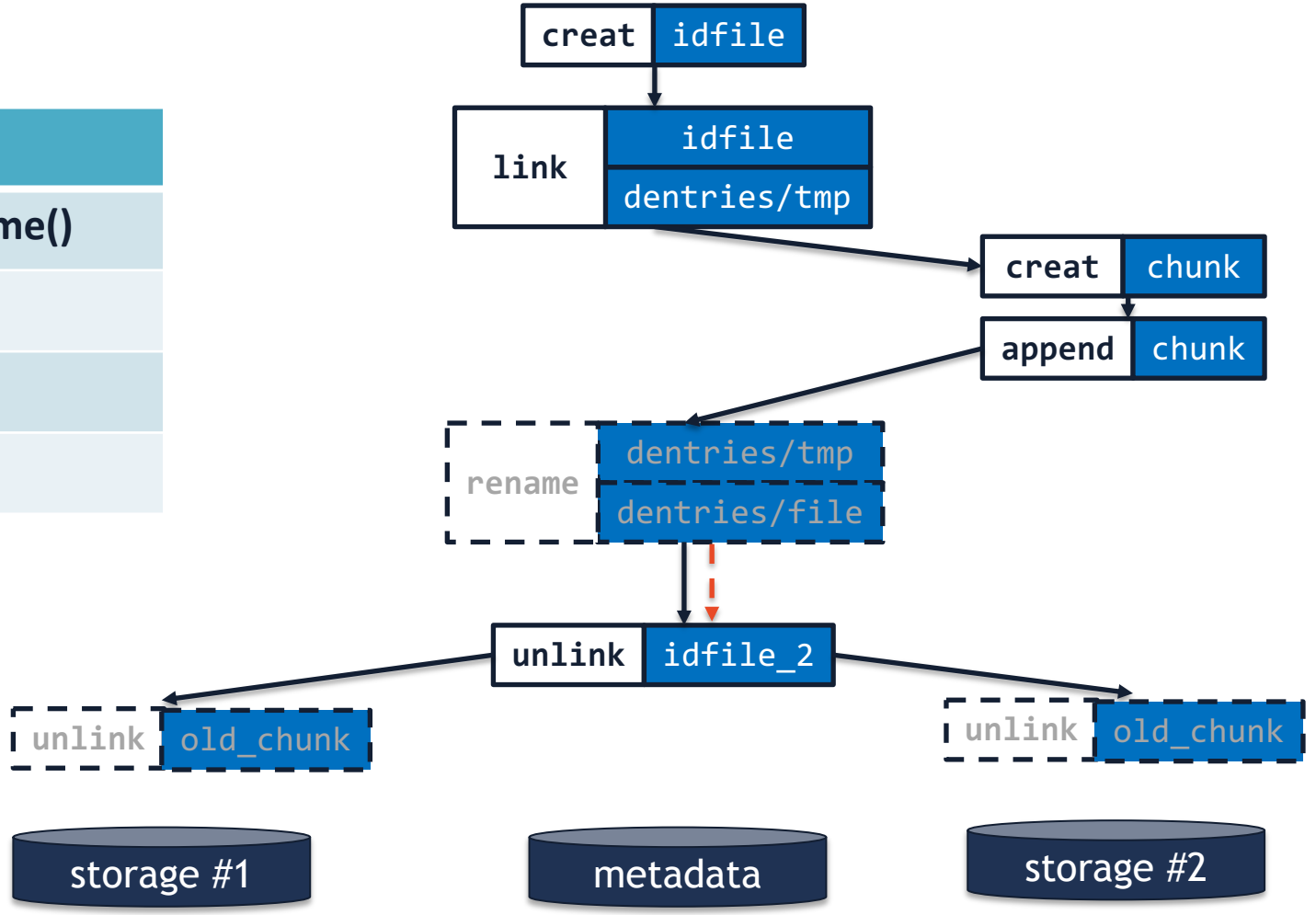


op param **Persisted operations**

op param **Non-persisted operations**

Crash vulnerability example

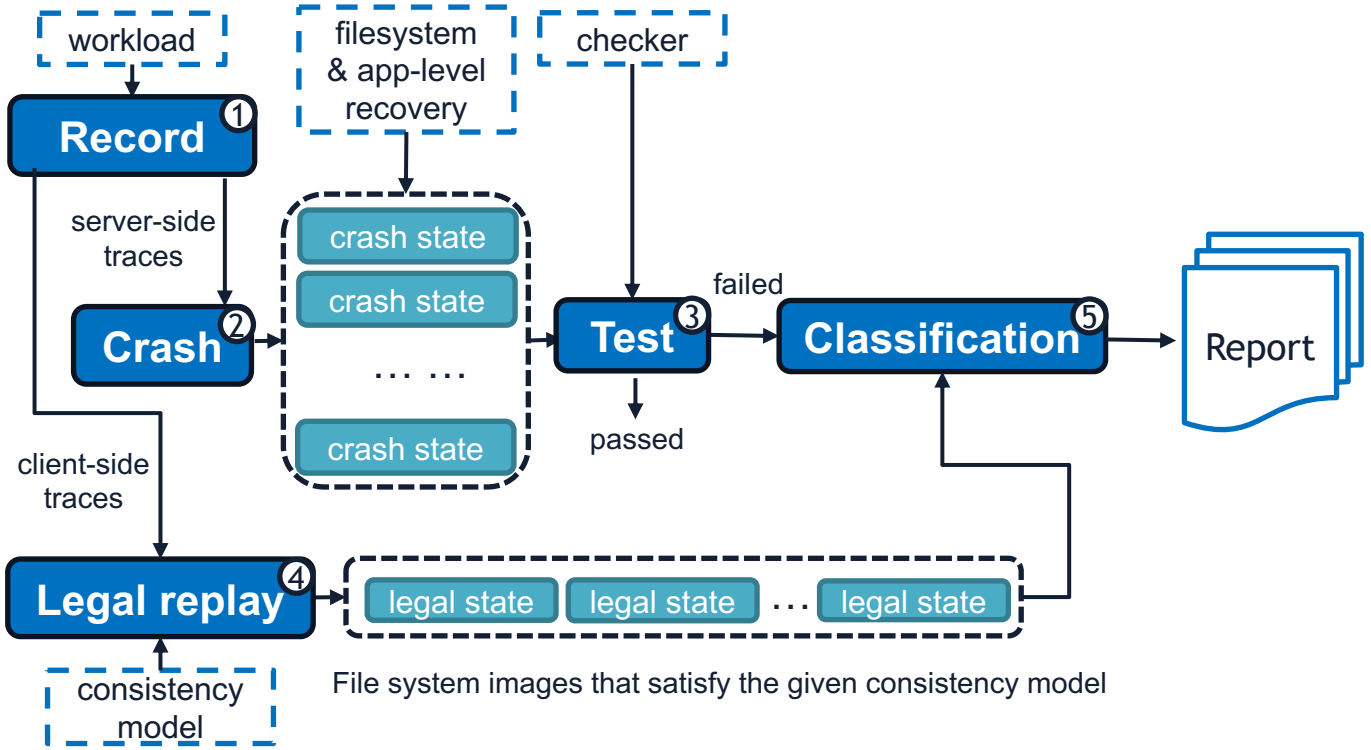
Inconsistency No.3	
Cause	**unlink()** persisted before **rename()**
Ordering	Intra-node dependency
Consequence	Data loss
Fixed by fsck?	Yes



op param **Persisted operations**

op param **Non-persisted operations**

PFSCheck design



Discovering PFS crash vulnerabilities systematically & efficiently

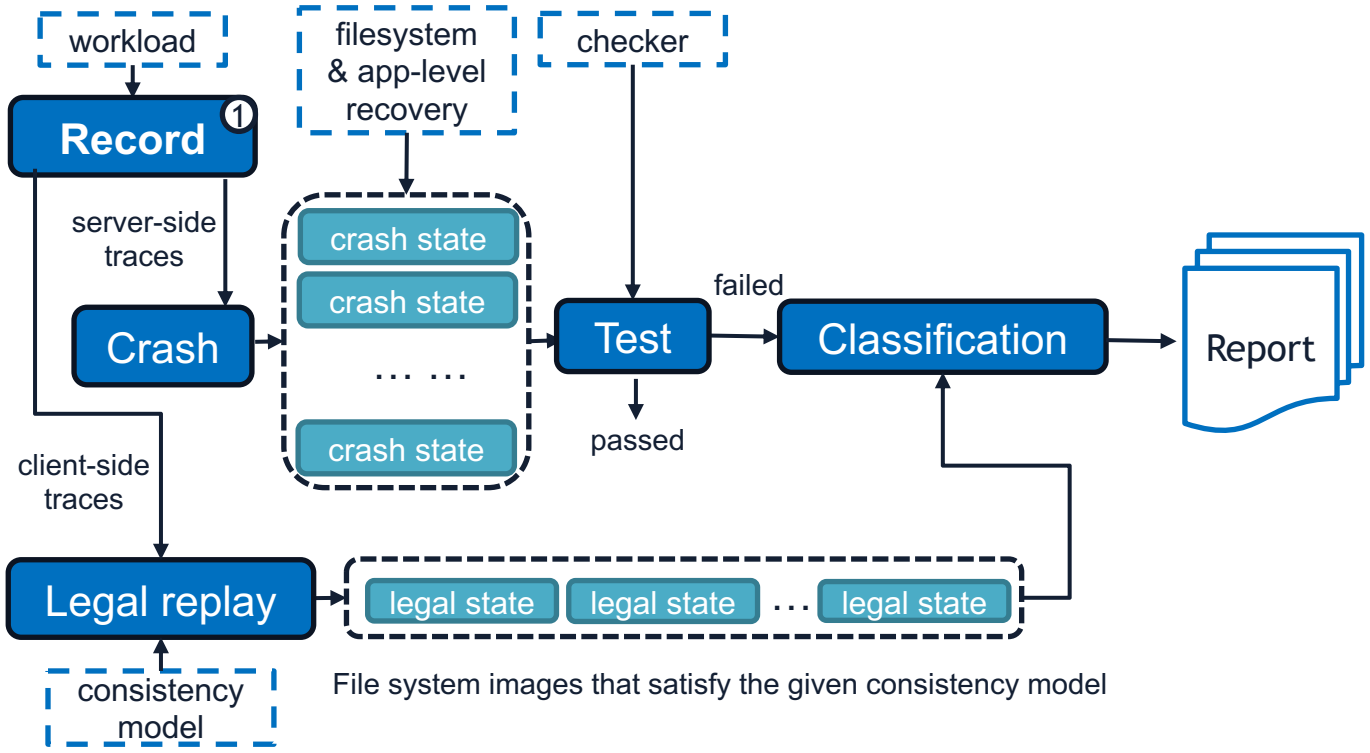
The PFSCheck design

Automated workload generation

- Unified API for I/O libraries

1. Multi-level tracing

- Joint server-side & client-side I/O calls tracing
- Network packet tracing
- Correlation between server & client operations



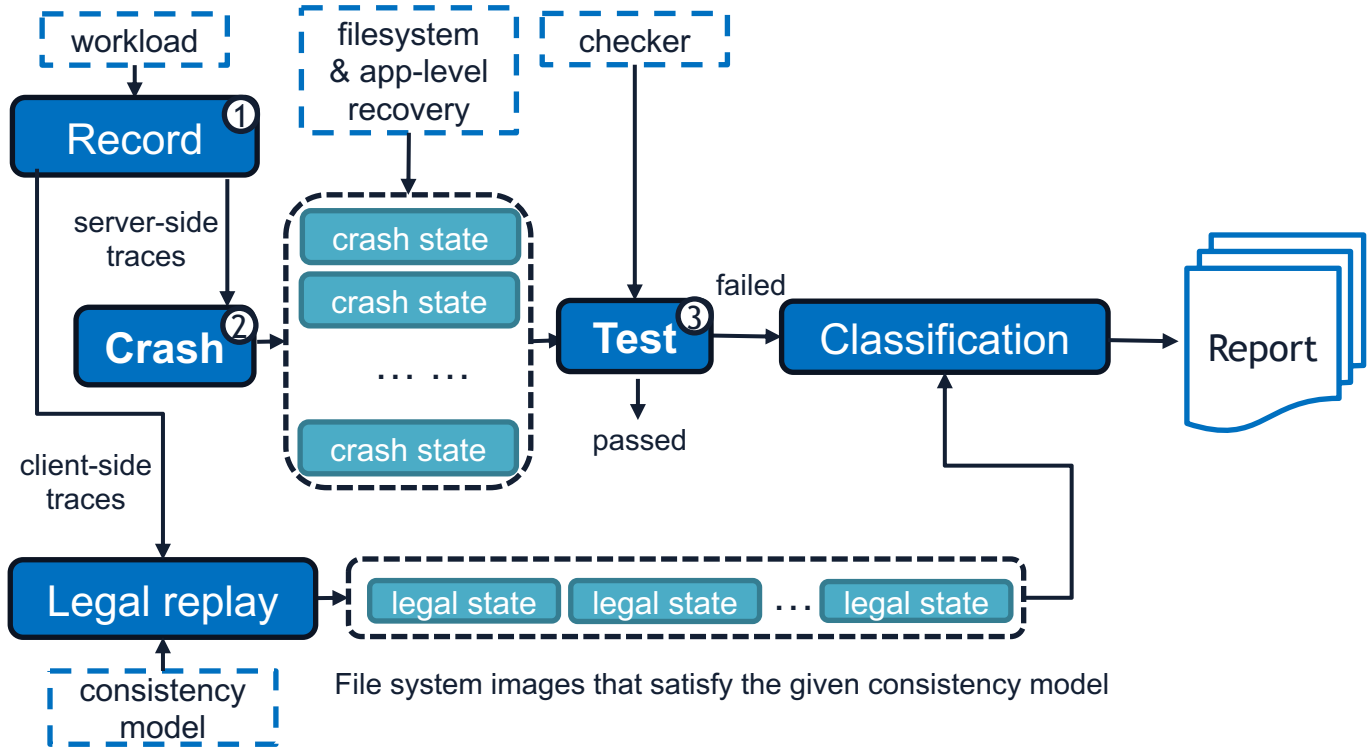
The PFSCheck design

2. Efficient crash state emulation

- Automated crash state generation via trace permutation
- Perform necessary post-crash recovery

3. Consistency testing

- Workload-specific consistency checker



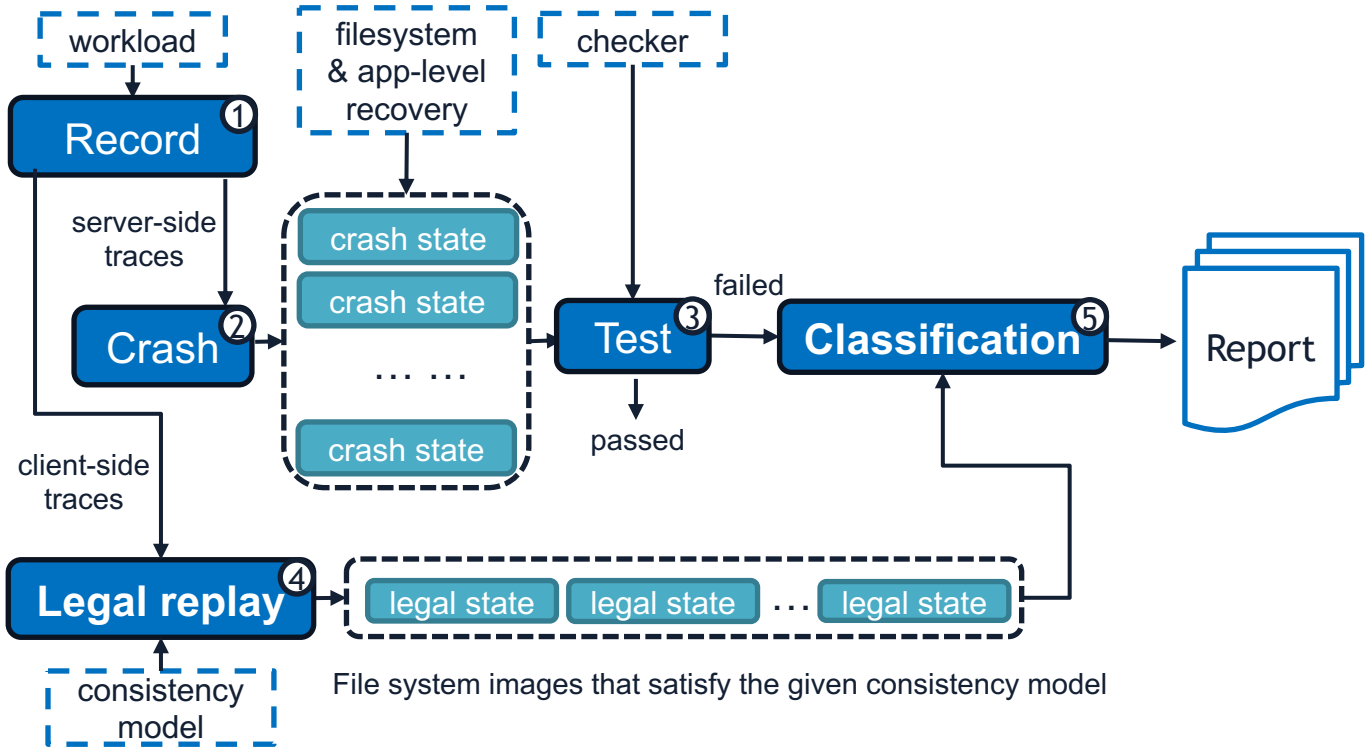
The PFSCheck design

4. Legal replay based on given consistency model

- Crash consistency model specifies the legitimate crash states of the parallel file system

5. Crash vulnerability classification

- If a vulnerable crash state is not a legal state, we attribute it to PFS
- Otherwise, I/O libraries are blamed



Conclusion

- **Motivation:** crash vulnerabilities could be exacerbated on PFSes, due to the complexity of the parallel I/O stack

- **Study:**
  - the number of crash consistency bugs on BeeGFS and OrangeFS is higher than local filesystem
  - the workload can fail in more ways on PFSes
  - the consistency relies on persistency reordering across nodes

- **Proposed framework:** PFS-specific crash consistency checker with a focus on automation and efficiency

Thank you!

Contact: Jinghan Sun (js39@illinois.edu)