Monarch: A Fuzzing Framework for Distributed File Systems

Tao Lyu Liyi Zhang Zhiyao Feng Yueyang Pan Yujie Ren

Meng Xu Mathias Payer Sanidhya Kashyap

EPFL

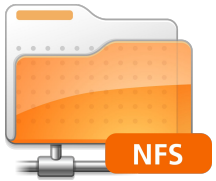
UNIVERSITY OF WATERLOO

DFSes are critical infrastructure

High Performance Computing

Serverless[1]

Machine Learning Model Training[2]



...

1. How Amazon EFS works 2. Building Meta's GenAI Infrastructure

But DFSes are not reliable

- Memory-unsafe languages
- State-sharing across clients
- High concurrency
- Fault tolerance

Finding and fixing bugs in DFSes become important

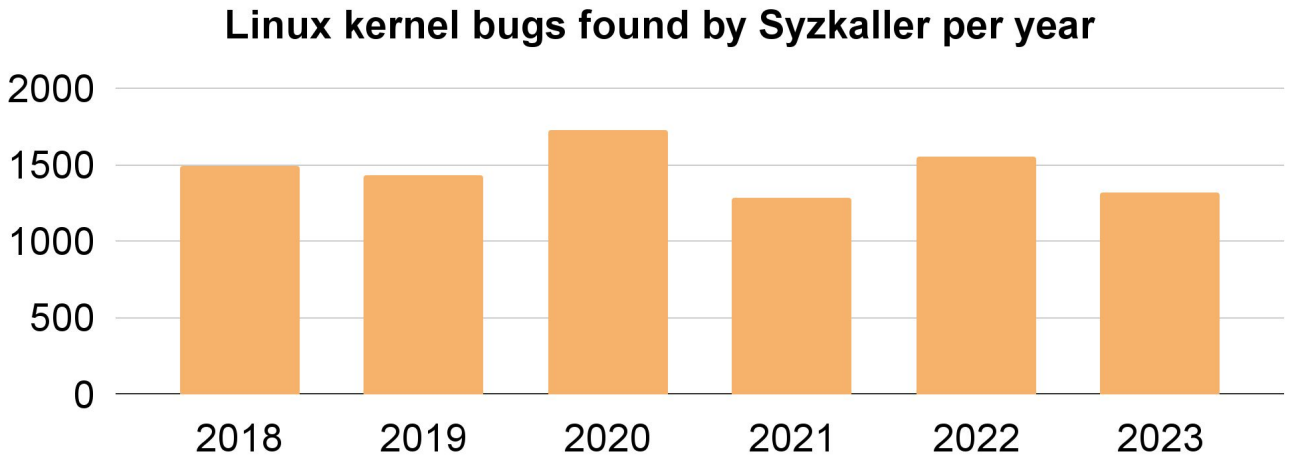
Existing approaches to finding DFSes bugs

Regression Testing	**Model Checking**	**Formal Verification**
Test suit	Modist SAMC	Verdi Ironfleet
Manual expert effort **Limited test cases**	**State explosion**	**Manual expert effort** **Mostly verify critical parts**

Any automatic and scalable bug-finding techniques?

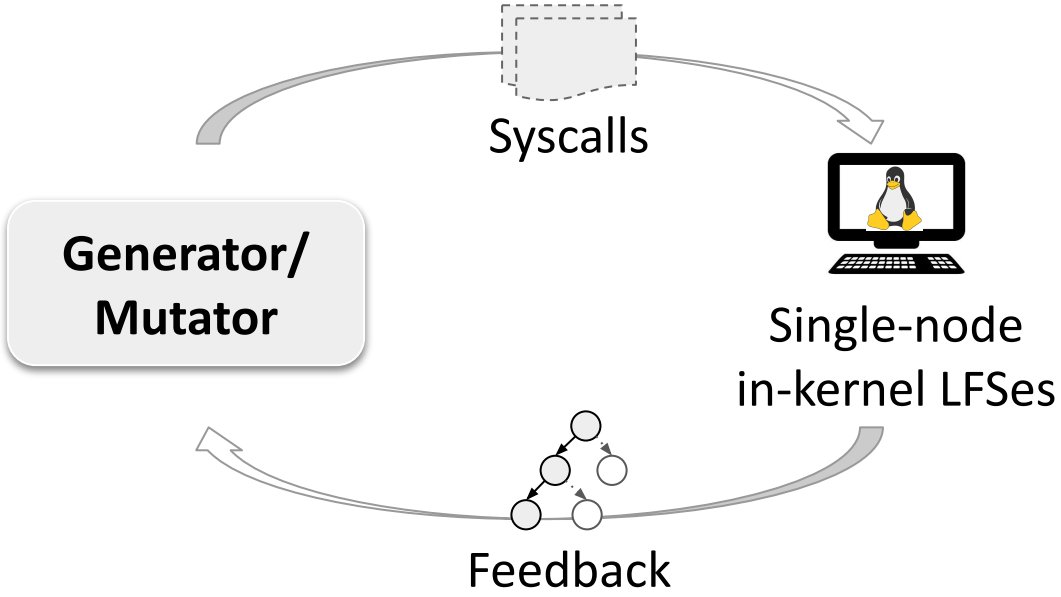
Fuzzing: A practical and impactful approach

- Linux kernel fuzzer, **syzkaller**, detects over **1K bugs every year**[1]
- Specifically, more than **800 bugs in local file systems** over the years



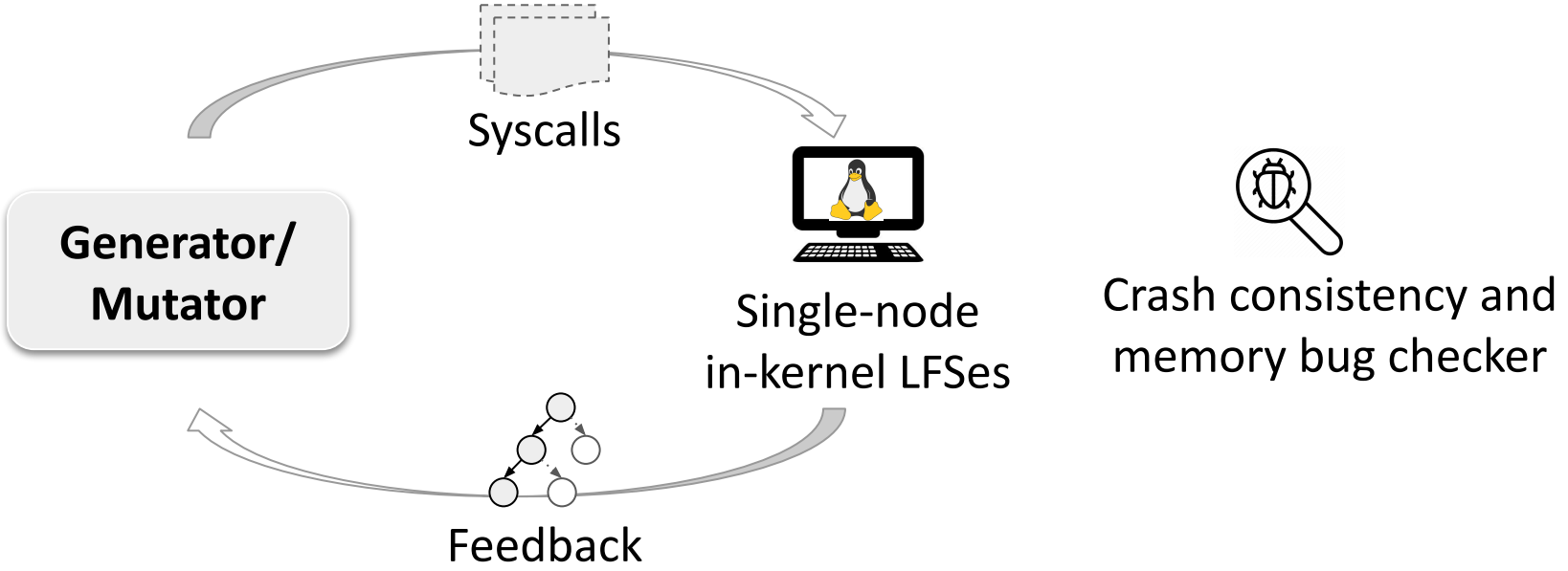
1. Syzbot/Syzkaller https://syzkaller.appspot.com/upstream/graph/found-bugs

How does fuzzing work?



Crash consistency and
memory bug checker

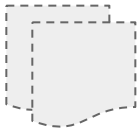
How does fuzzing work?



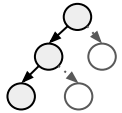
Current LFS fuzzers are not applicable for DFSes

Missing pieces for fuzzing DFSes

Single-node and in-kernel file system



Syscalls



Representation of single-kernel exe state

Crash consistency + memory checker

Multi-node cross-kernel/userspace fuzzing architecture

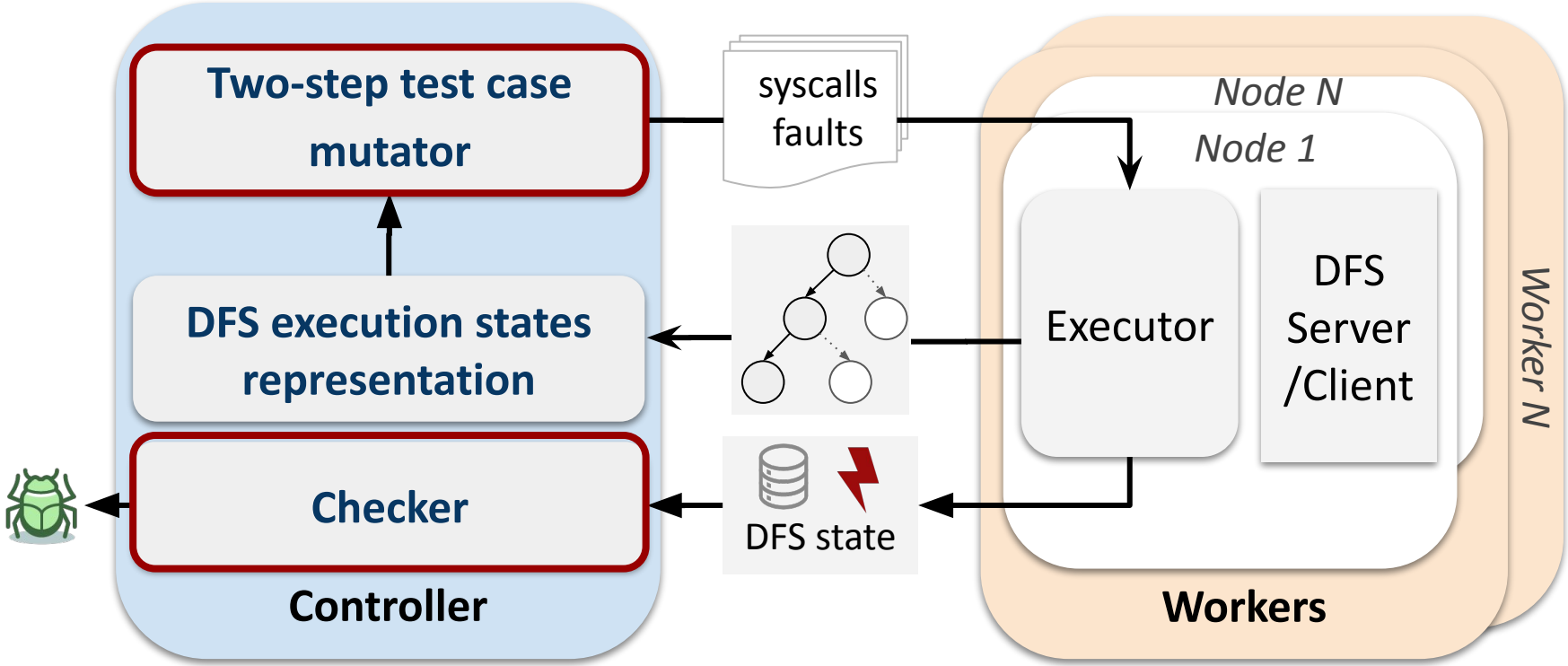
Distributed faults as a testing input space

Representation of cross-node and cross-kernel/userspace execution states

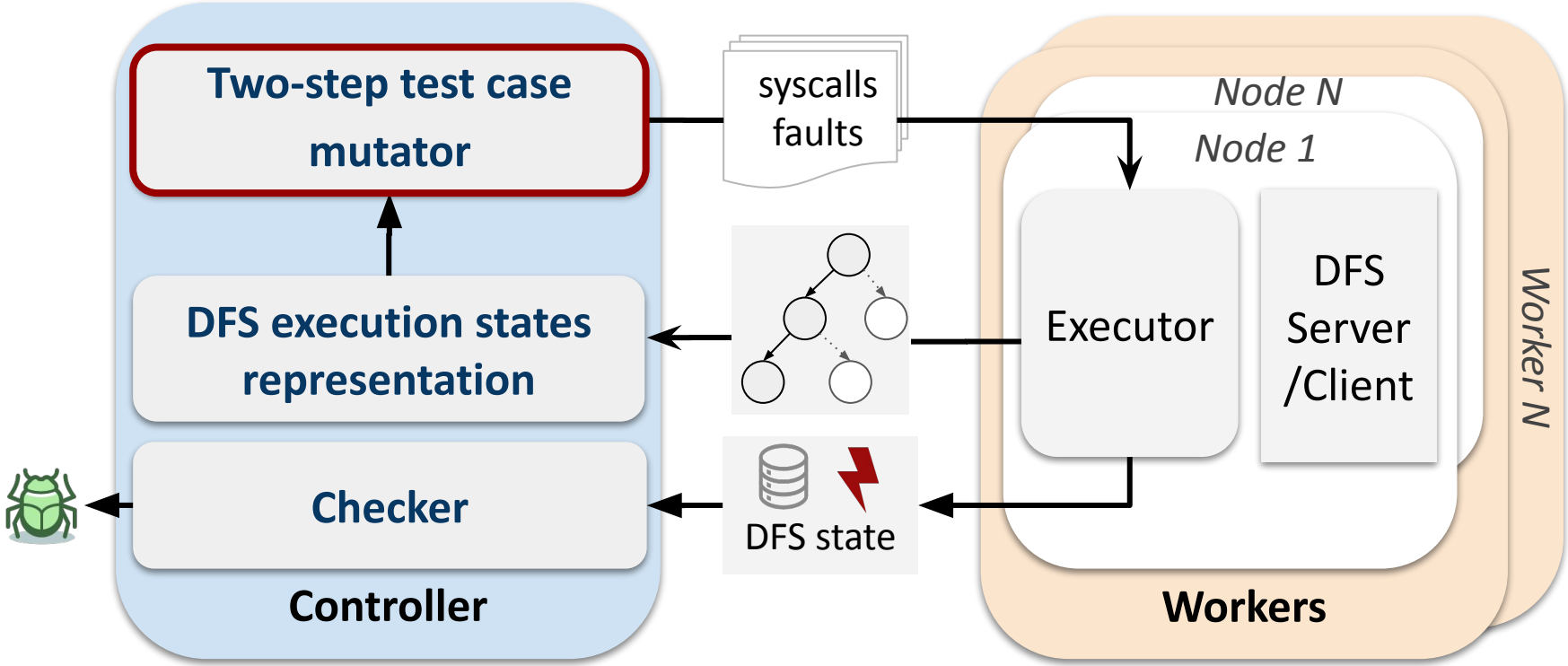
A systematic DFS semantic checker

MONARCH
A distributed file system fuzzer

Monarch architecture



Monarch architecture



Two-step test case mutator

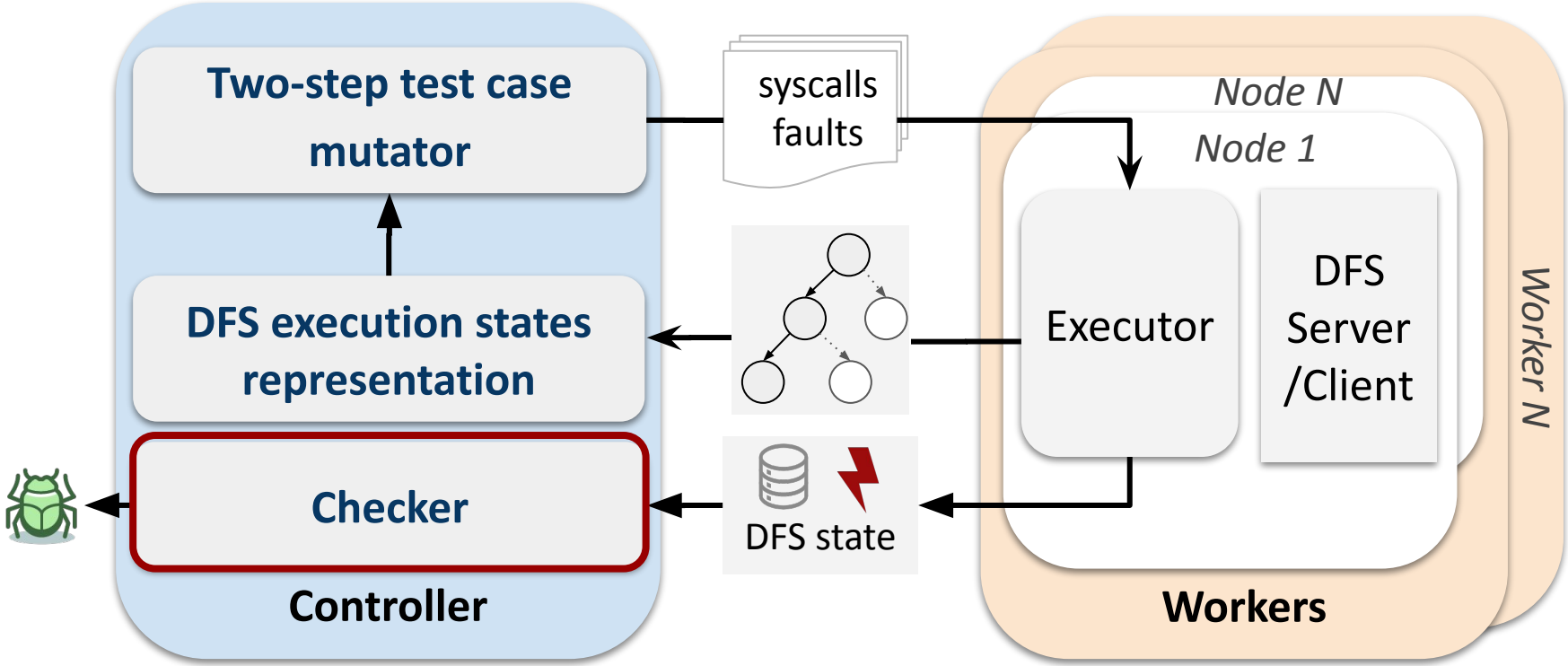
Non-fault mode: Testing with file system syscalls only (e.g., open, read)

- Syscall templates and mutation rules

Fault mode: Testing with syscalls + injected faults

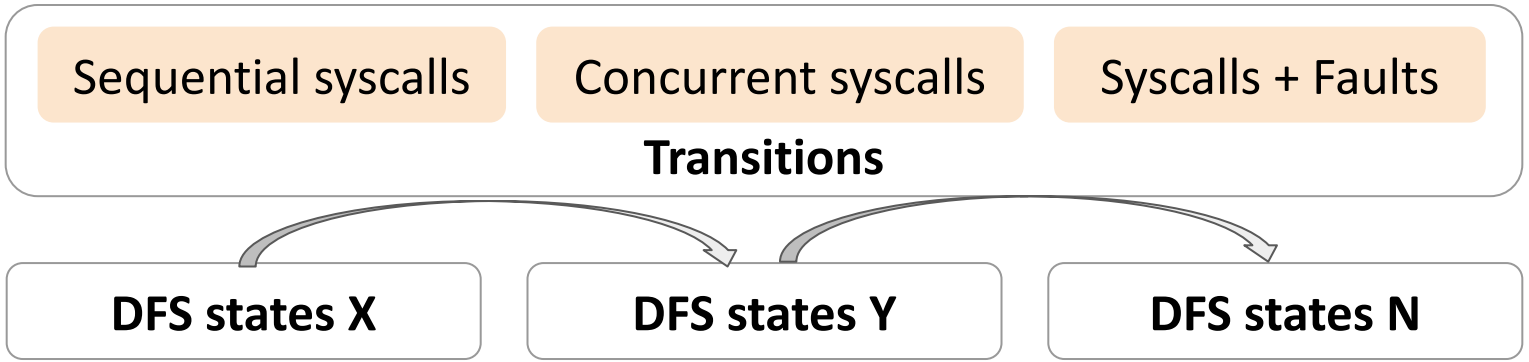
- **Distributed faults** → Network partitions, node crashes
- **How to deterministically inject faults?** → Synchronization primitives
- **At which granularity to inject faults?** → Syscall granularity
- **When to inject faults?** → After a test case triggers new execution state

Monarch architecture



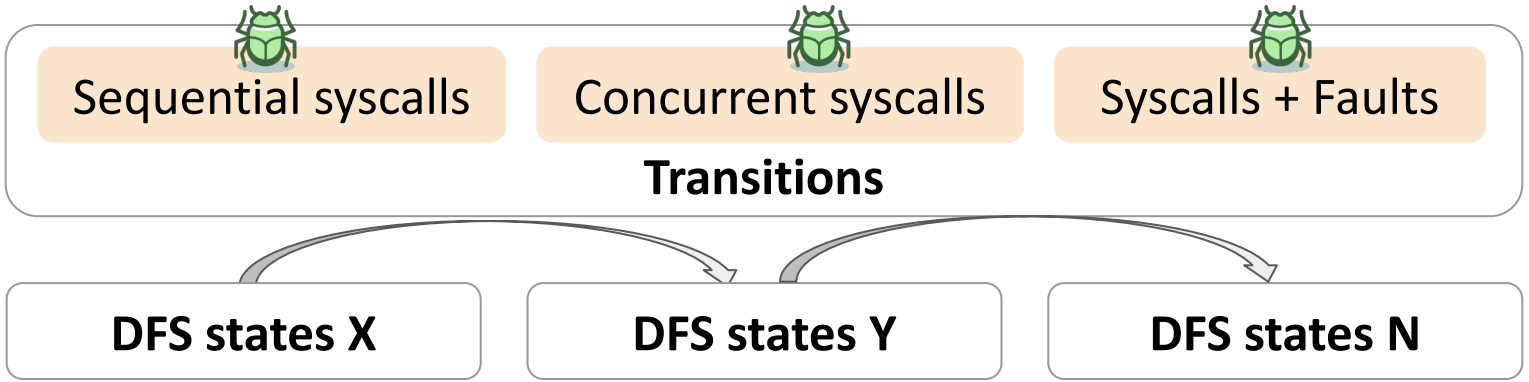
What are the semantic bugs in DFS?

- **Semantic**: DFS state transitions specified in the spec
  - Syscalls: sequential and concurrent
  - Syscalls + faults



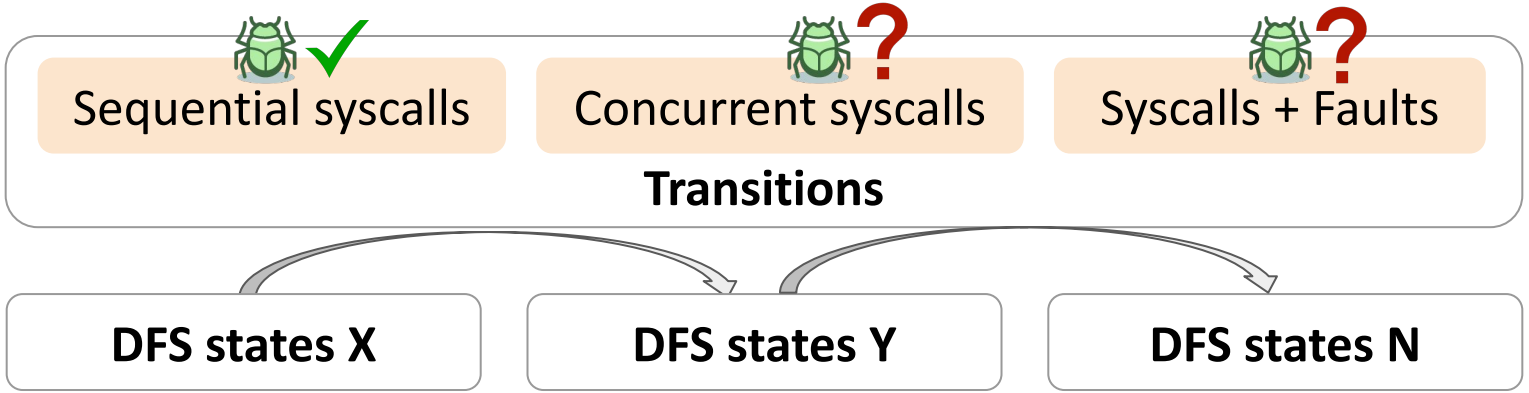
What are the semantic bugs in DFS?

- **Semantic**: DFS state transitions specified in the spec
  - Syscalls: sequential and concurrent
  - Syscalls + faults
- **Semantic bugs**: implementations violate the specified semantic

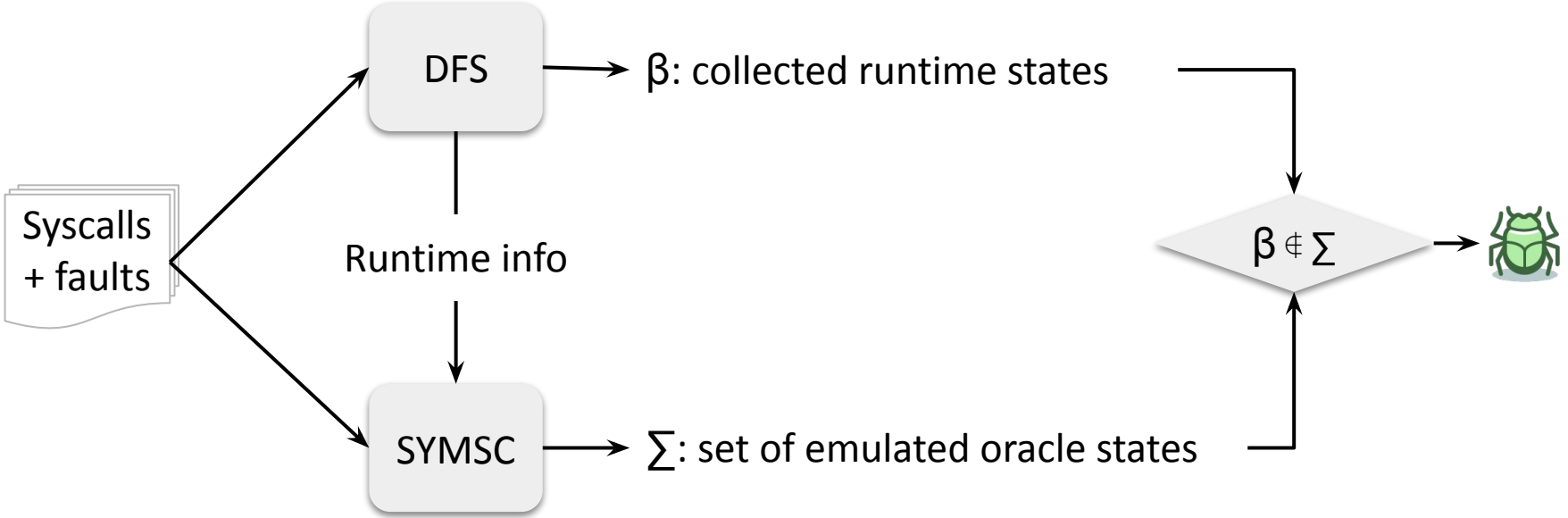


Which semantic bug types are missing?

- **Semantic**: DFS state transitions specified in the spec
  - Syscalls: sequential and concurrent
  - Syscalls + faults
- **Semantic bugs**: implementations violate the specified semantic



DFS semantic checker: SYMSC



Emulate sequential syscall executions

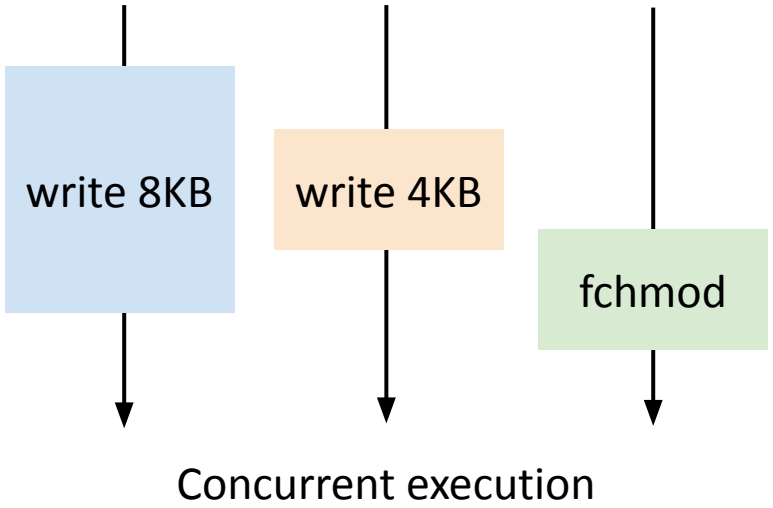
- Emulate syscalls one by one according to the specification

Sequential syscalls	Emulated states
mkdir A	Inode0.dents = [., A] Inode1.dents = []
setxattr A user.key:val	Inode1.xattr = [user.key:val] …

But the spec does not specify semantics under concurrency and faults

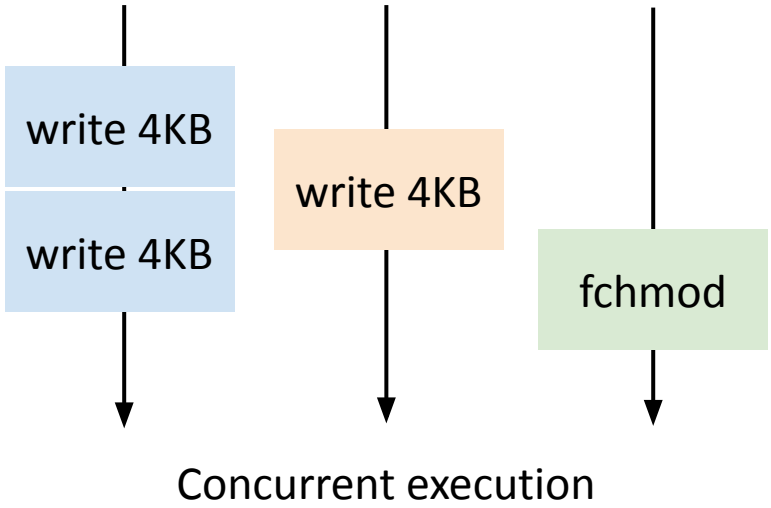
How to emulate semantic under concurrency?

- Concurrency relation built from syscall's start and end timestamp



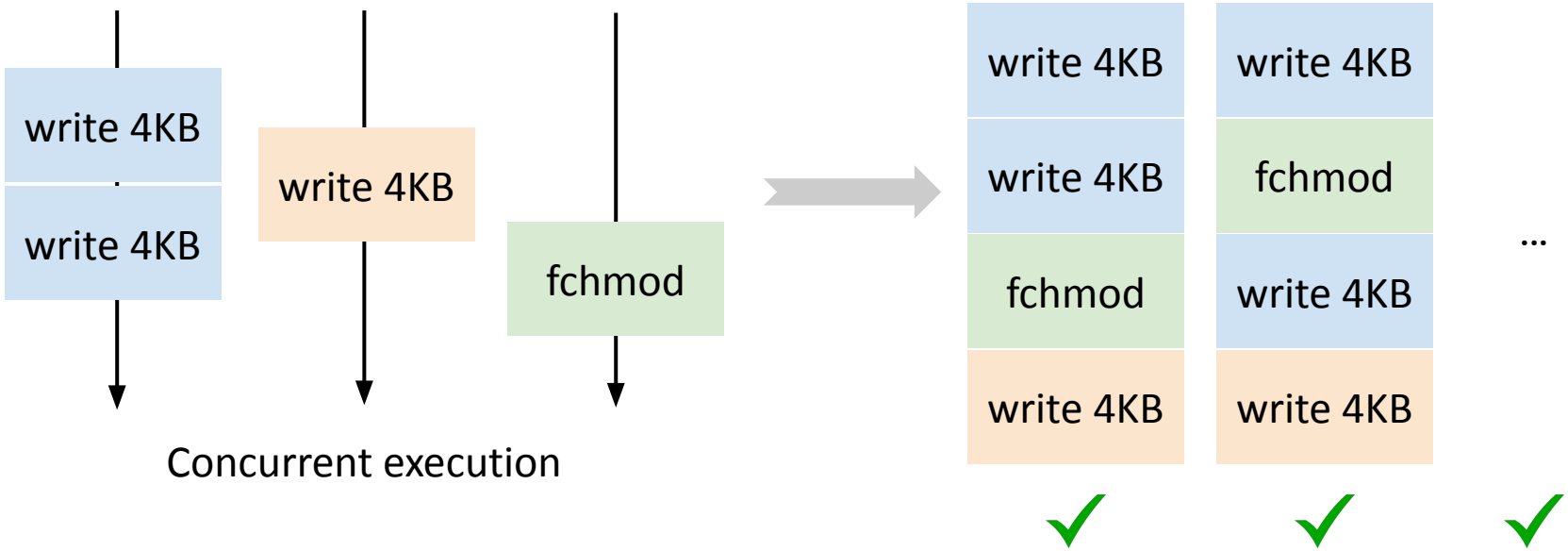
Syscalls to atomic operations

- Syscalls are splitted into one or more atomic operations

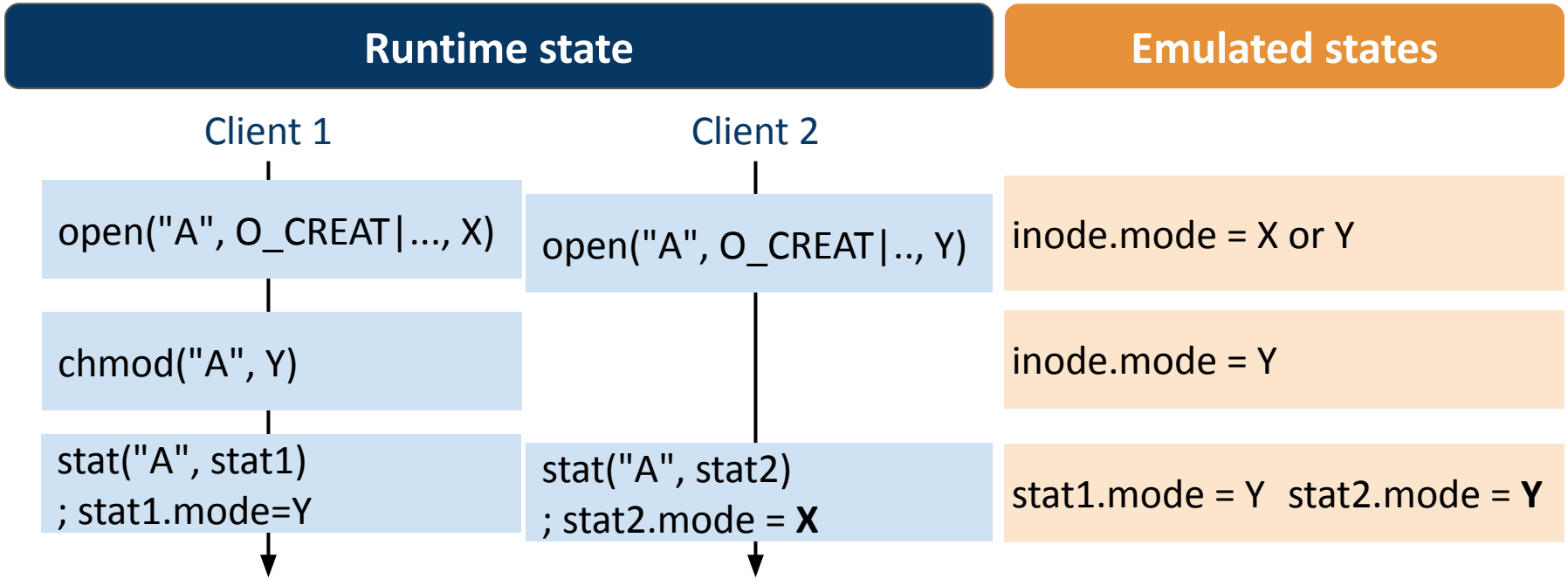


Serialize concurrent atomic operations

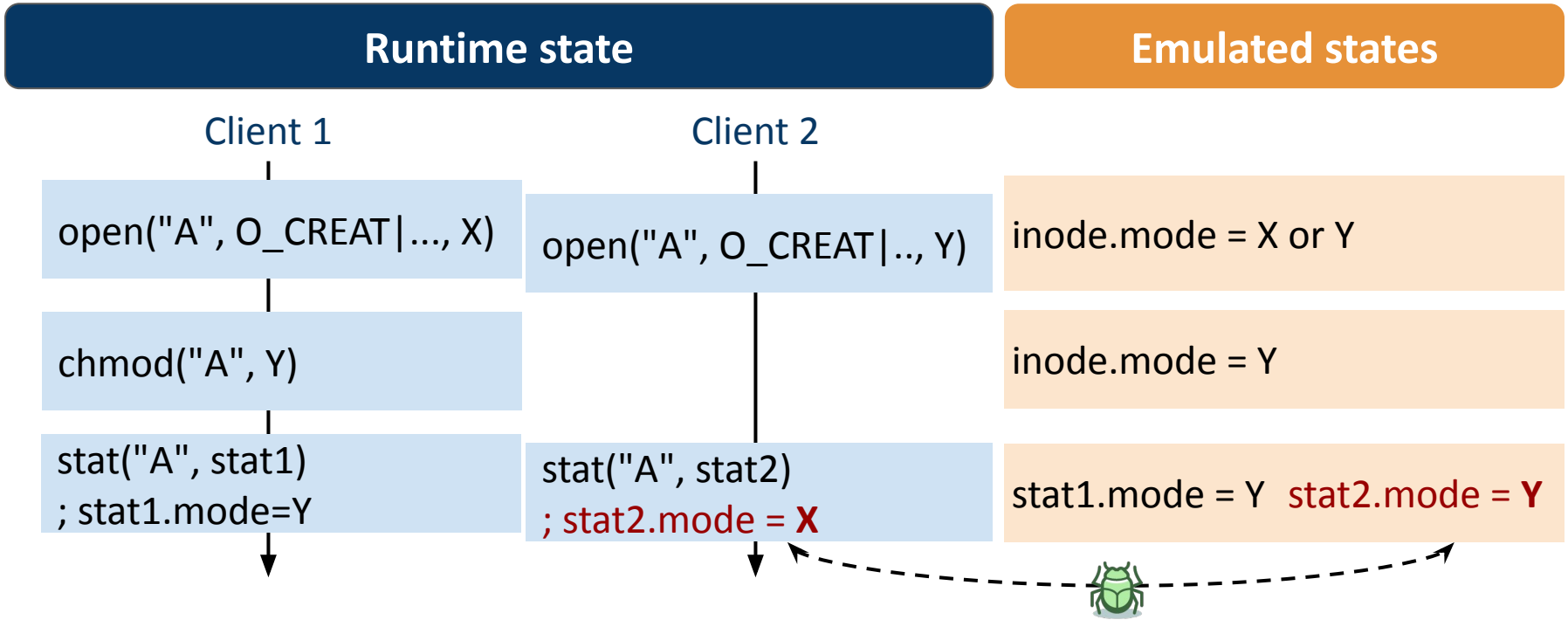
- Syscalls are splitted into one or more atomic operations
- **Oracle states**: emulate all interleavings among concurrent atomic operations



Case study of a CephFS bug

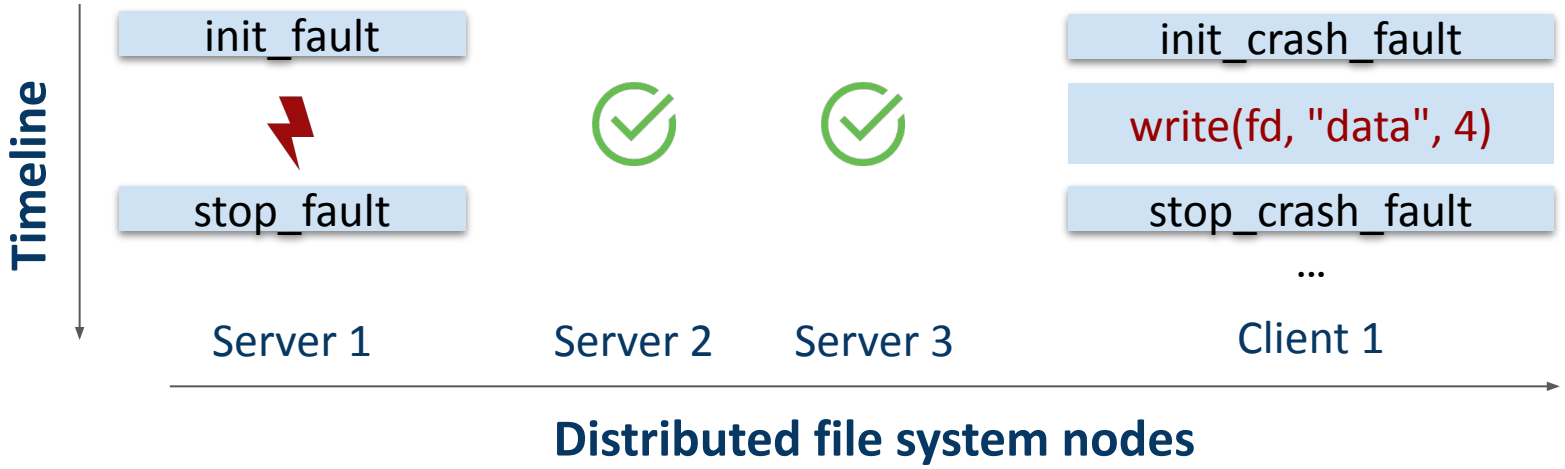


Case study of a CephFS bug



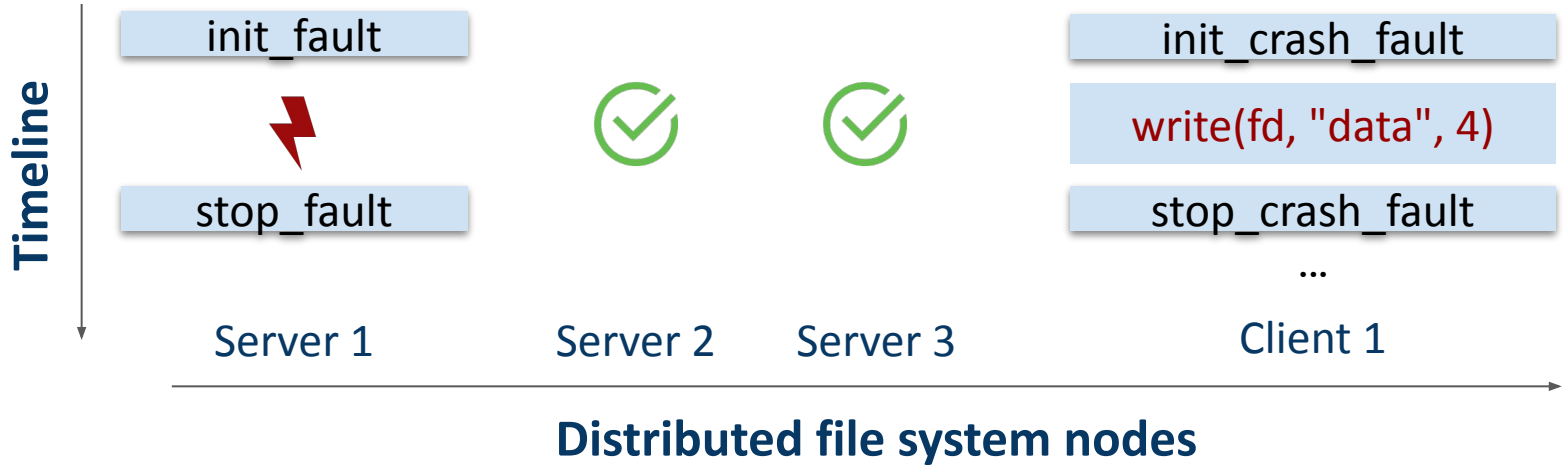
What is the semantic under faults?

What's the semantic of a write under faults?



What is the semantic under faults?

- Apply the original semantic if the DFS is available to this syscall
- Return errors otherwise



Implementation and evaluation setup

- Two 64-core machines
- Ran intermittently for two months on six targeted DFSes

How effective is Monarch in finding bugs?

- Monarch found bugs in all DFSes and categories, with a total of 48 bugs

DFS	Memory bugs	Semantic bugs
Lustre	8	0
GlusterFS	17	5
OrangeFS	3	0
BeeGFS	0	2
CephFS	4	1
NFS	8	0
Total	**40**	**8**

What are the characteristics of DFS bugs?

Faults play a critical role in exposing these bugs

→ 14/40 memory bugs and 3/8 semantic bugs are exposed under faults

Vulnerable code is scattered in both servers and clients

→ 17 bugs (servers) vs 31 bugs (clients)

→ The root causes of semantics bugs are mostly in DFS servers

Bug exposure might depend on specific DFS configurations

Conclusion

- **Problem:** Automatic and scalable bug-finding tool for DFSes
- **Our solution Monarch**: The first DFS fuzzing framework
  - Multi-node and cross-context fuzzing architecture
  - A two-step mutator to test DFSes with syscalls and faults
  - DFS semantic checker
- **Takeway**: Fuzzing is effective on distributed systems as well
- **Artifact**: https://github.com/rs3lab/Monarch

Thank you!