Enhanced File System Testing Through Input and Output Coverage

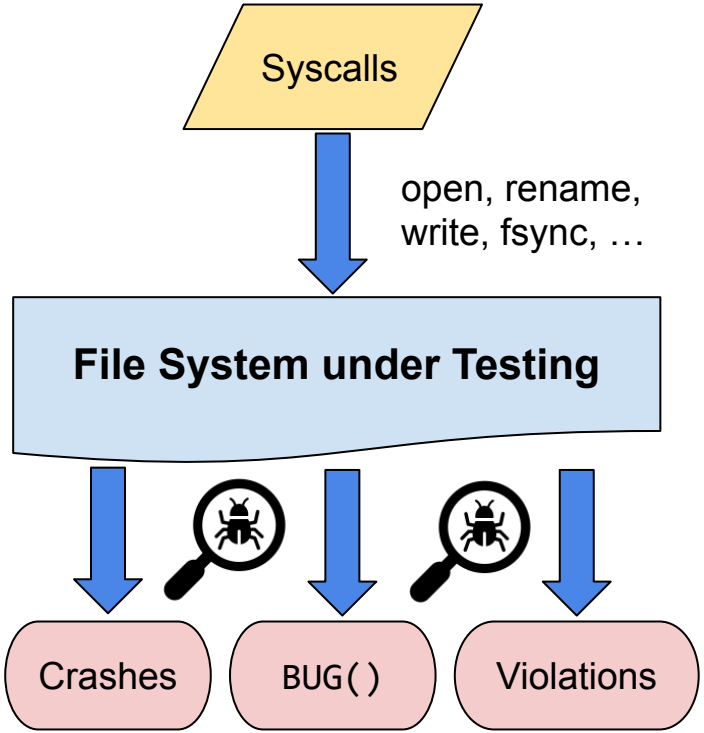
18th ACM International Systems and Storage Conference (SYSTOR '25)

Yifei Liu[1], Geoff Kuenning[2], Md. Kamal Parvez[1], Scott A. Smolka[1], and Erez Zadok[1]

[1] Stony Brook University; [2] Harvey Mudd College

File System Testing

- Bugs emerge even in well-tested file systems[1]
- Coverage matters: bugs hide in corner cases
- Coverage metrics guide testing
  - Code coverage is most common
  - Evaluate tests, then improve them
  - **Effective metrics**: higher coverage leads to more bugs found



[1] Kim, Seulbae, et al. “Finding Semantic Bugs in File Systems with an Extensible Fuzzing Framework”, SOSP, 2019.

Code Coverage

- Measures how much source code is exercised
  - **Levels**: lines, functions, branches, etc.
  - Assess test completeness; finds untested code
- Limitations of code coverage in file system testing
  - **Weak link**: test inputs ↔ file system code
  - Large effort to instrument kernel code
- **Unclear correlation: coverage vs. test effectiveness**

Real-World Bug Study

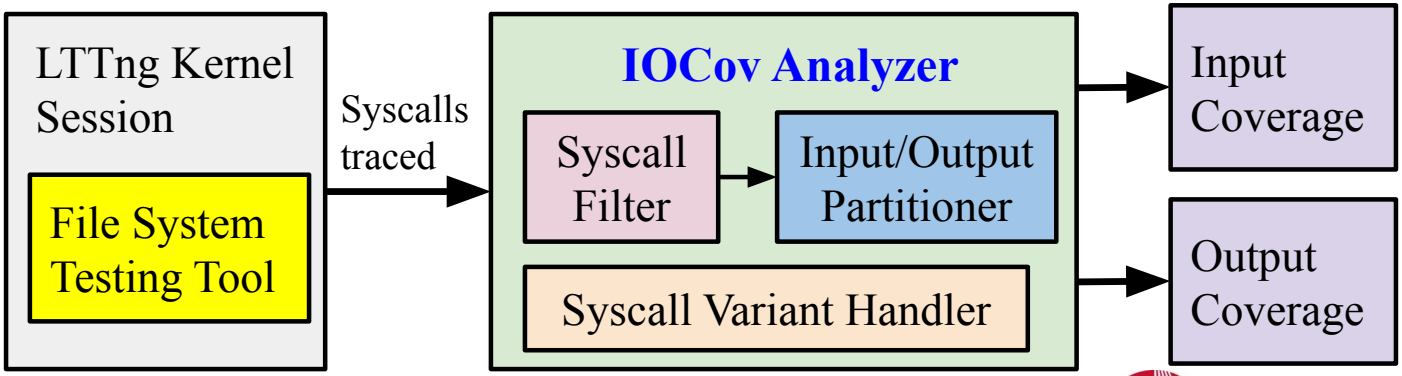
- Analyzed 70 recently reported Ext4 and Btrfs bugs
  - Ran xfstests to check bug detection and code coverage
  - Does code coverage imply bug detection?
- xfstests missed bugs despite line, function, and branch coverage
- 71% of bugs depend on specific syscall inputs (input bugs)
- 59% occur on exit paths affecting syscall returns (output bugs)
- **Takeaways**
  1. Code coverage is ***not strongly correlated*** with the test effectiveness of file system testing
  2. Covering both ***syscall inputs and outputs*** is essential for file system testing

Input and Output Partitioning

- Syscall input and output space is massive
  - **Linux:** ~400 syscalls, dozens of them for file systems
  - **Input/Output space:** various arguments, arbitrary values, error codes
- Input space partitioning
  - **Bitmasks**: Partitioned by bit flags (e.g., `open` flags)
  - **Numeric**: Partitioned by powers of 2 numbers (e.g., `write` size)
  - **Categorical**: Partitioned by individual categories (e.g., `lseek` whence)
- Output space partitioning
  - Success or failure; Error codes; Powers of 2 for bytes
- **Input/output coverage:** coverage of input/output partitions

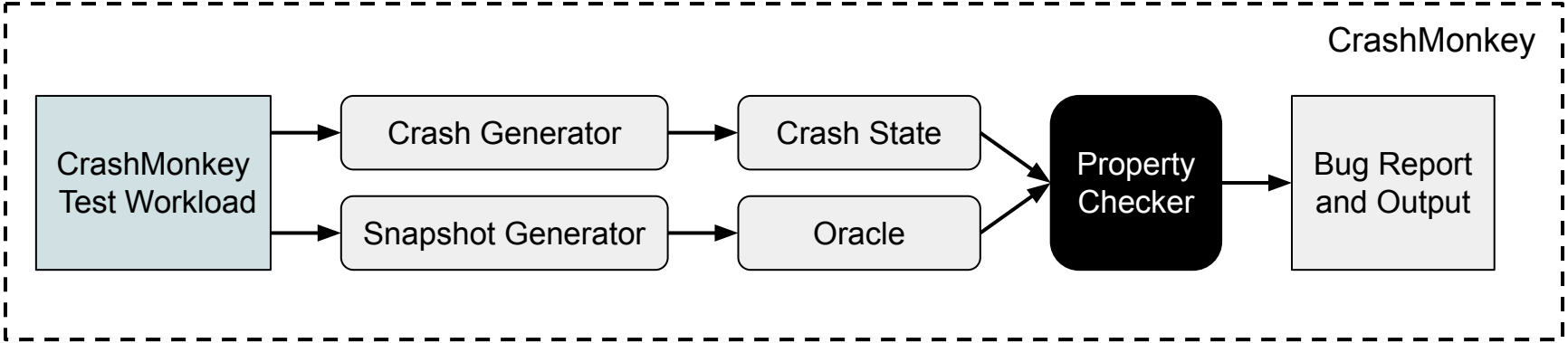
IOCov Framework

- **IOCov:** computing input and output coverage for file system testing tools
  - Syscall filter
    - Filter out irrelevant syscalls not used for testing
  - Syscall variant handler
    - Merge coverage of syscall variants
  - Input/Output partitioner
    - Partition syscall Input/Output space to obtain coverage



Application and CrashMonkey Architecture

- **IOCov application:** evaluate and improve coverage for better testing
- **CrashMonkey**[2]: simulates crashes to test file system crash consistency
- **CM-IOCov**: improves CrashMonkey's input coverage to detect more crash consistency bugs



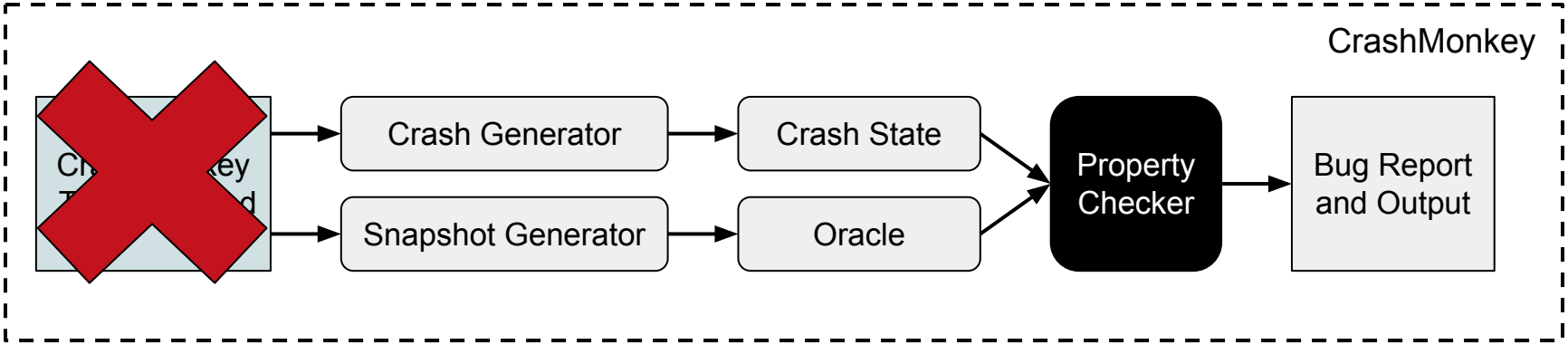
[2] Mohan, Jayashree, et al. "Finding Crash-Consistency Bugs with Bounded Black-Box Crash Testing", OSDI, 2018.

CM-IOCov Architecture

CM-IOCov Input Driver:
generates workloads covering more input partitions than original CrashMonkey

Examples of newly-supported inputs

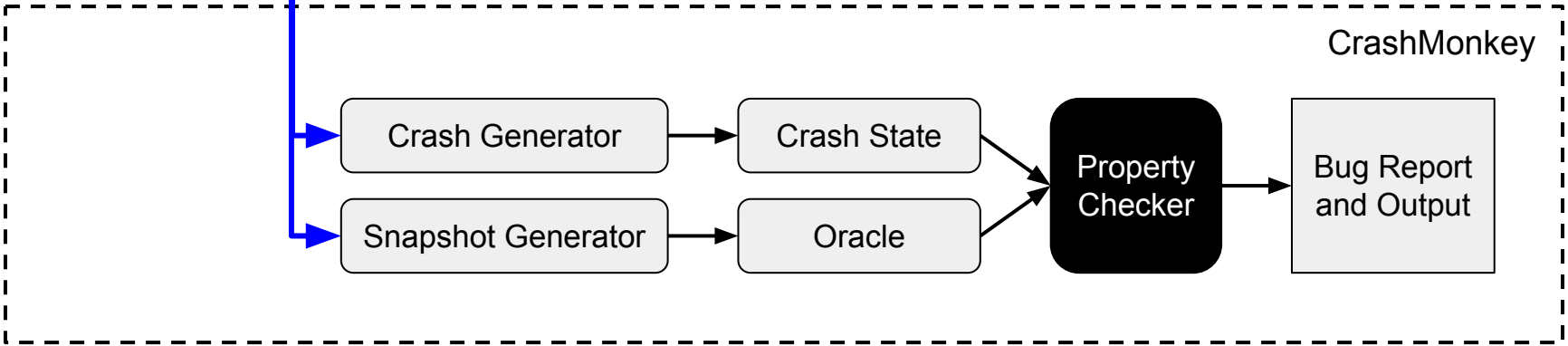
- More `open` flags
- More `open/mkdir` mode
- More `write/offset/fallocate` bytes
- …



CM-IOCov Architecture

CM-IOCov Input Driver: generates workloads covering more input partitions than original CrashMonkey

- Seamlessly replace the original CrashMonkey input driver
- Reuse CrashMonkey's crash simulation and checker modules

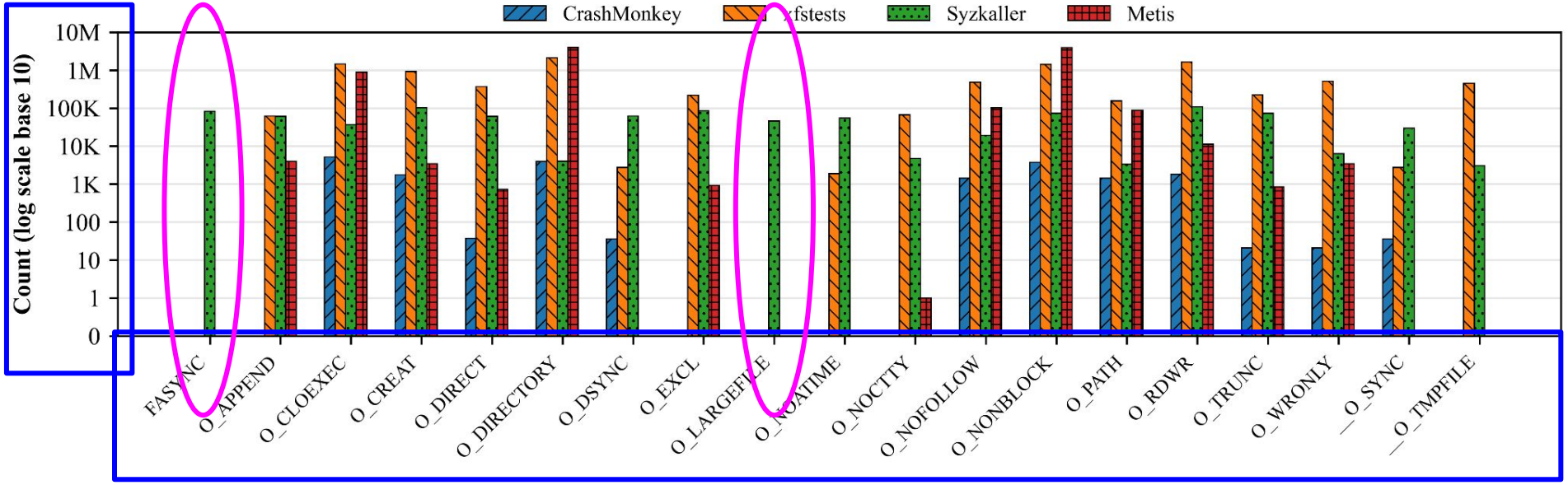


IOCov Evaluation Setup

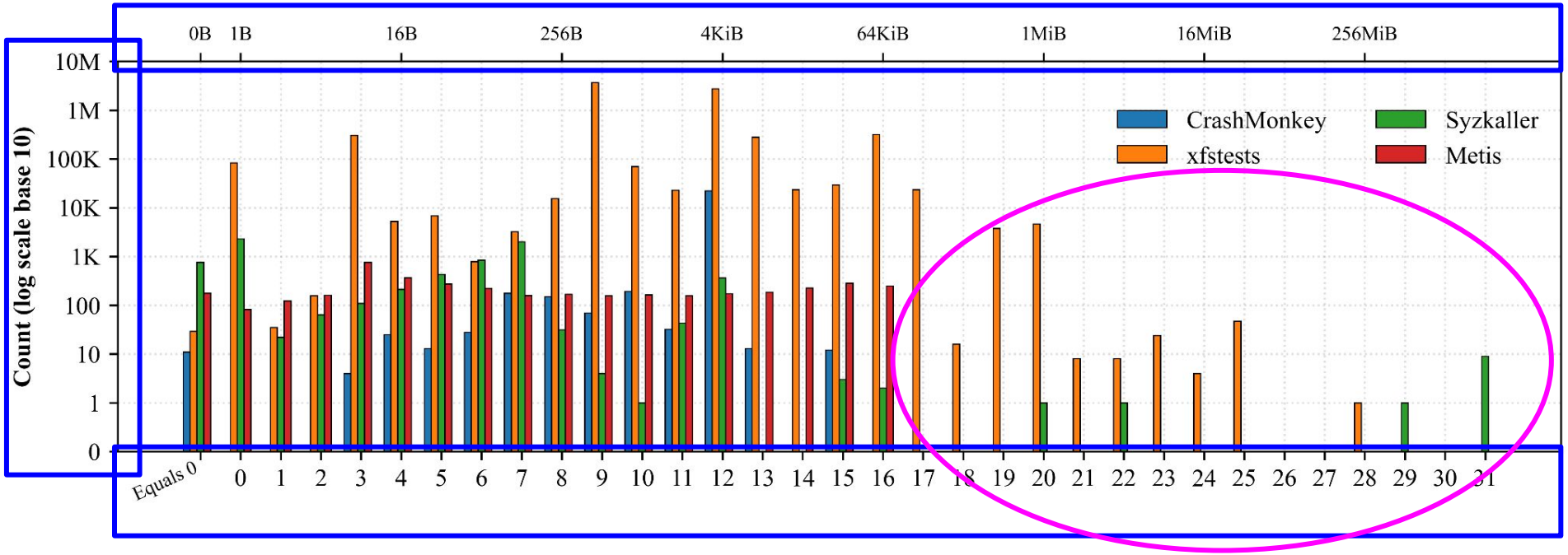
- IOCov supports input/output coverage for 27 file system calls, including 11 base syscalls
- Four representative testing tools for the **Ext4** file system
  - **CrashMonkey**: automatic test generation
  - **xfstests**: regression test suite
  - **Syzkaller**: fuzzing
  - **Metis**: model checking
- Measured input/output coverage over equal time

Base Syscall	Variants	Arguments (inputs) Captured
open	openat creat openat2	flags mode

Input Coverage: `open()` flags



Input Coverage: `write()` sizes



Bug Detection: CM-IOCov vs. CrashMonkey

- Ran same workloads on CM-IOCov and CrashMonkey
  - **Exclusive test failures**: detected by only one tool (CM-IOCov or CrashMonkey)
  - One bug → multiple failing tests
- **Kernel 5.6**, total **426K** workloads
  - Exclusive test failures — **CM-IOCov: 400** vs. **CrashMonkey: 31**
- **Kernel 6.12**, total **379K** workloads
  - Exclusive test failures — **CM-IOCov: 322** vs. **CrashMonkey: 115**
- **CM-IOCov**: significantly more exclusive failures than CrashMonkey

No.	Bug Consequence	System Call Sequence
1	Allocated blocks lost after `fsync`	open, write, falloc
2	File content did not match after `fsync`	open, write, mmapwrite
3	Data block missing after `rename`	open, write, falloc, rename
4	Rename not persisted by `fsync`	opendir, close, rename, mkdir
5	Incorrect number of file hard links after `fsync`	mkdir, open, link, rename

Conclusions

- Code coverage is not strongly correlated with test effectiveness in file system testing
- File system testing requires input and output coverage alongside code coverage
- **IOCov**: measures input/output coverage to identify under- and over-testing and offers insights to improve testing
- **CM-IOCov**: improves input coverage to find more crash consistency bugs in file systems

Enhanced File System Testing through Input and Output Coverage

Thank You

Q&A

IOCov

yifeliu@cs.stonybrook.edu

CM-IOCov