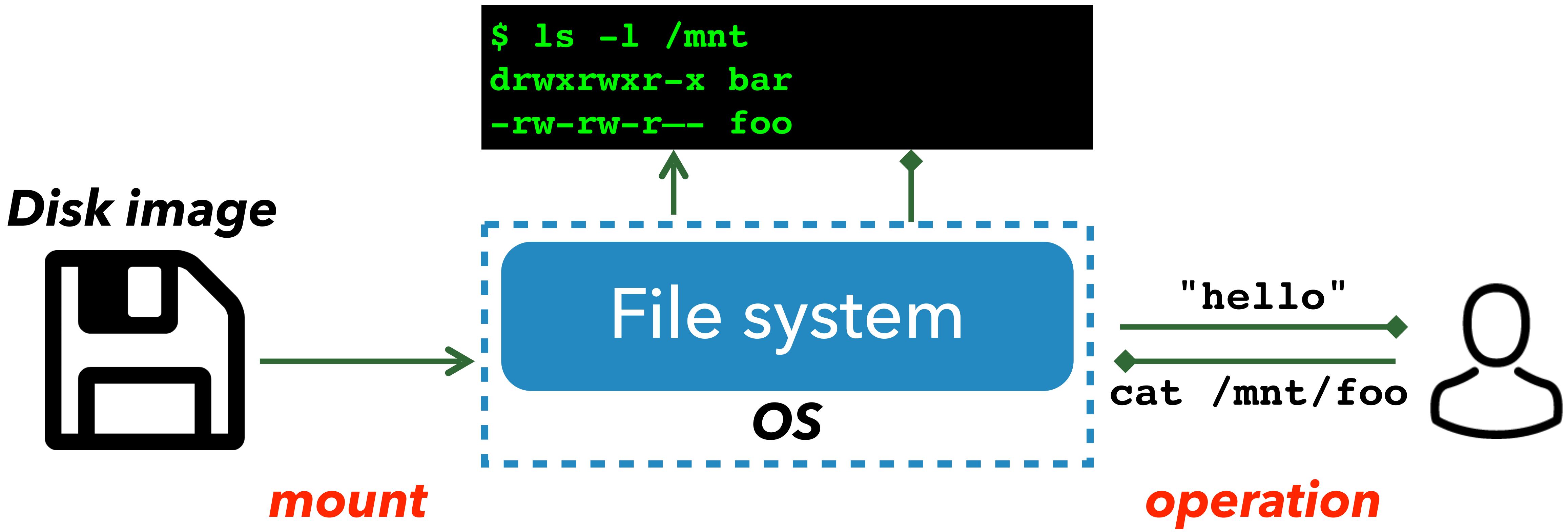
Fuzzing File Systems via Two-Dimensional Input Space Exploration

Wen Xu, Hyungon Moon, Sanidhya Kashyap, Po-Ning Tseng and Taesoo Kim

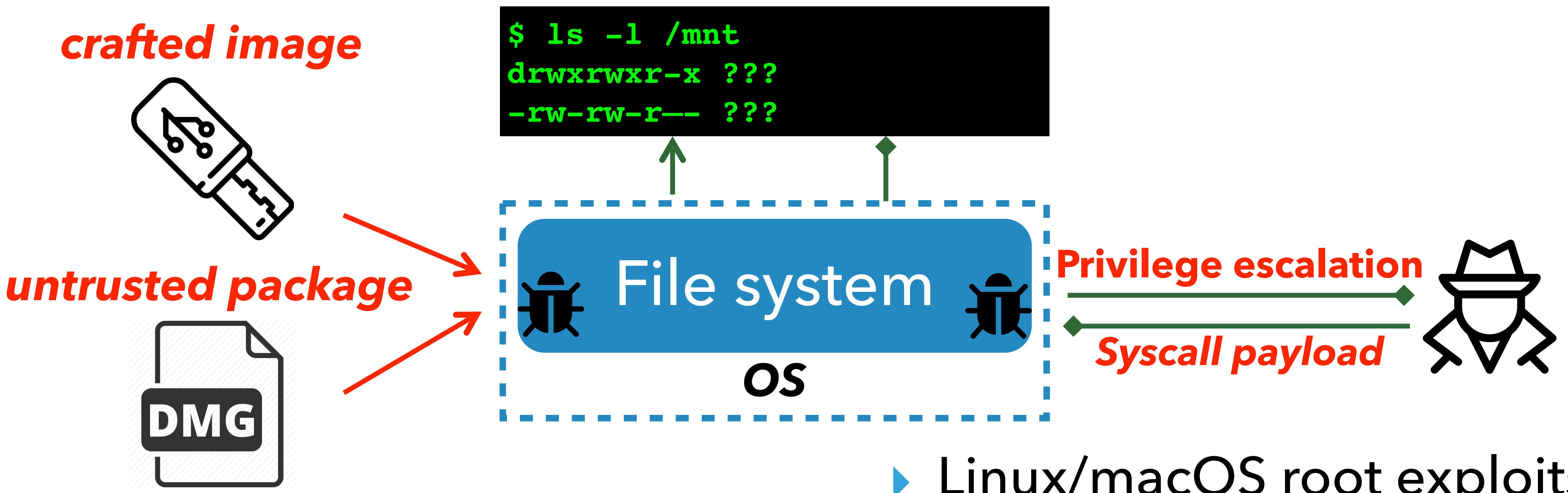
INTRODUCTION

FILE SYSTEMS 101



```
$ ls -l /mnt
drwxrwxr-x bar
-rw-rw-r-- foo
```

FILE SYSTEM ATTACKS



```
$ ls -l /mnt
drwxrwxr-x ???
-rw-rw-r-- ???
```

- Linux/macOS root exploits
- Evil maid attacks
- Air-gapped APT attacks

COMPLEX FILE SYSTEMS

FS	LoC	Active
ext4	50K	✓
XFS	140K	✓
Btrfs	130K	✓

File systems are hard to be *bug-free!*

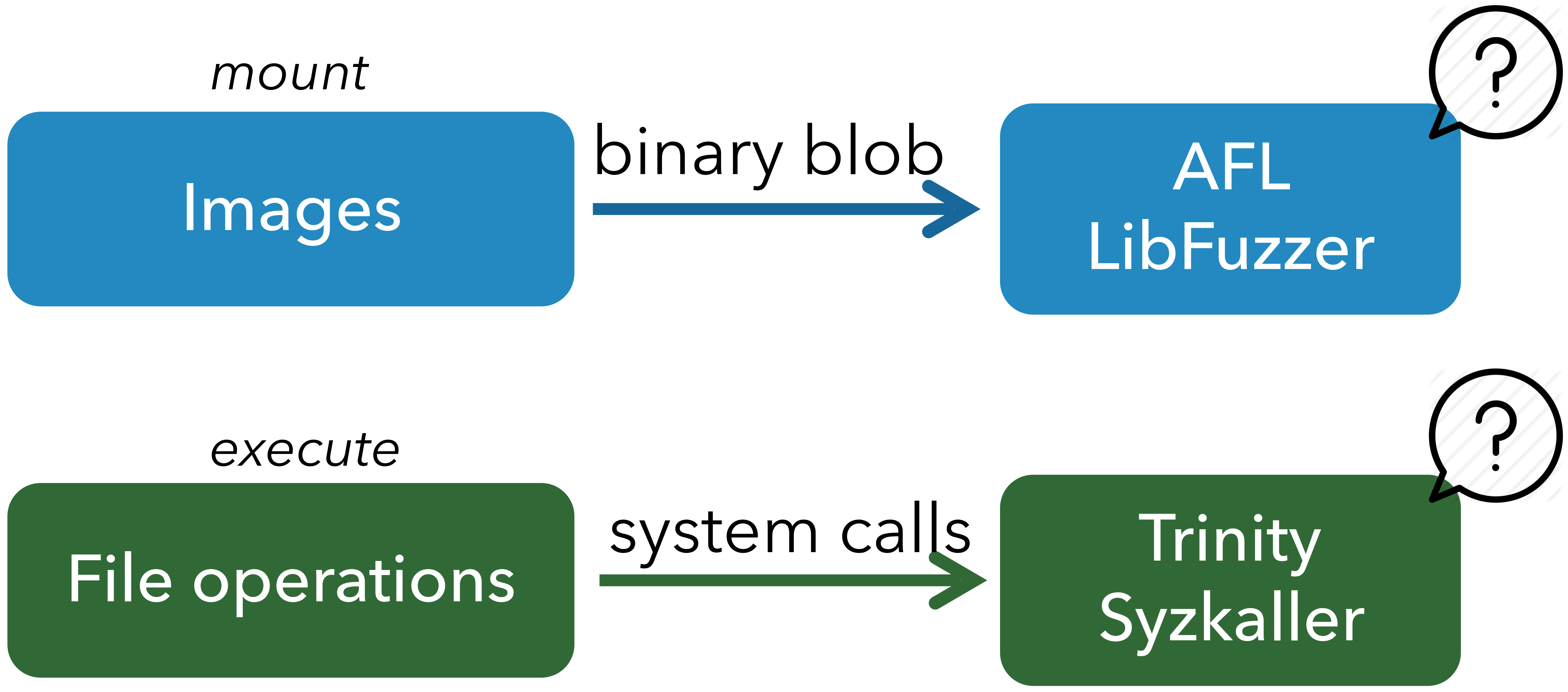
SOLUTION: FUZZING

Efficient

Minimal knowledge

Practical

FUZZING FILE SYSTEMS



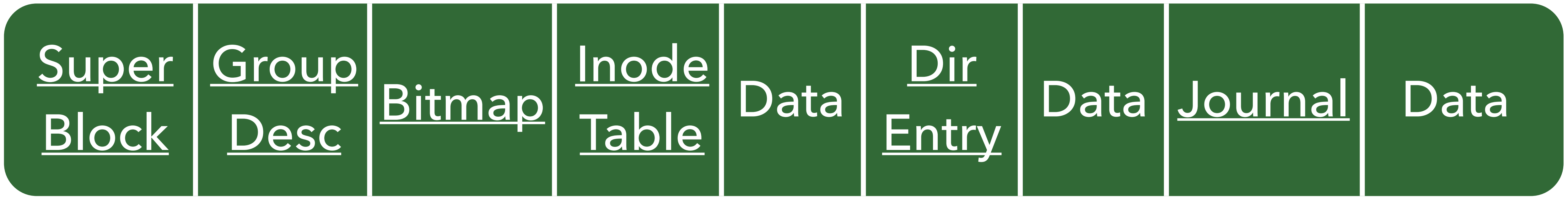
CHALLENGES

FILE SYSTEM IMAGES REVISITED

- Particularly *large*

ext4: 2MB | XFS: 16MB | Btrfs: 100MB

- Highly *structured* (metadata)



ext4 disk layout

- Checksums

[1] FUZZING IMAGES AS BLOBS

- Particularly *large*

Huge IO costs on loading and saving testcases

- Highly *structured*

Metadata is rarely touched

- Checksums

Corrupted after mutation

OUR APPROACH: META-ONLY IMAGE FUZZING

- Locate and extract only metadata blocks for mutation
- Record checksum information for each metadata block

OUR APPROACH

- Particularly *large*

Metadata occupies < 1%

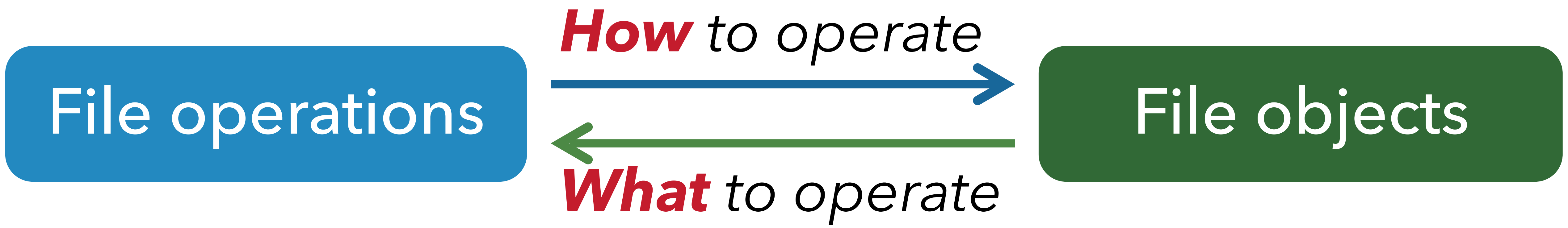
- Highly *structured*

Only metadata is fuzzed

- Checksums

Enough information for fixing

FILE OPERATIONS REVISITED



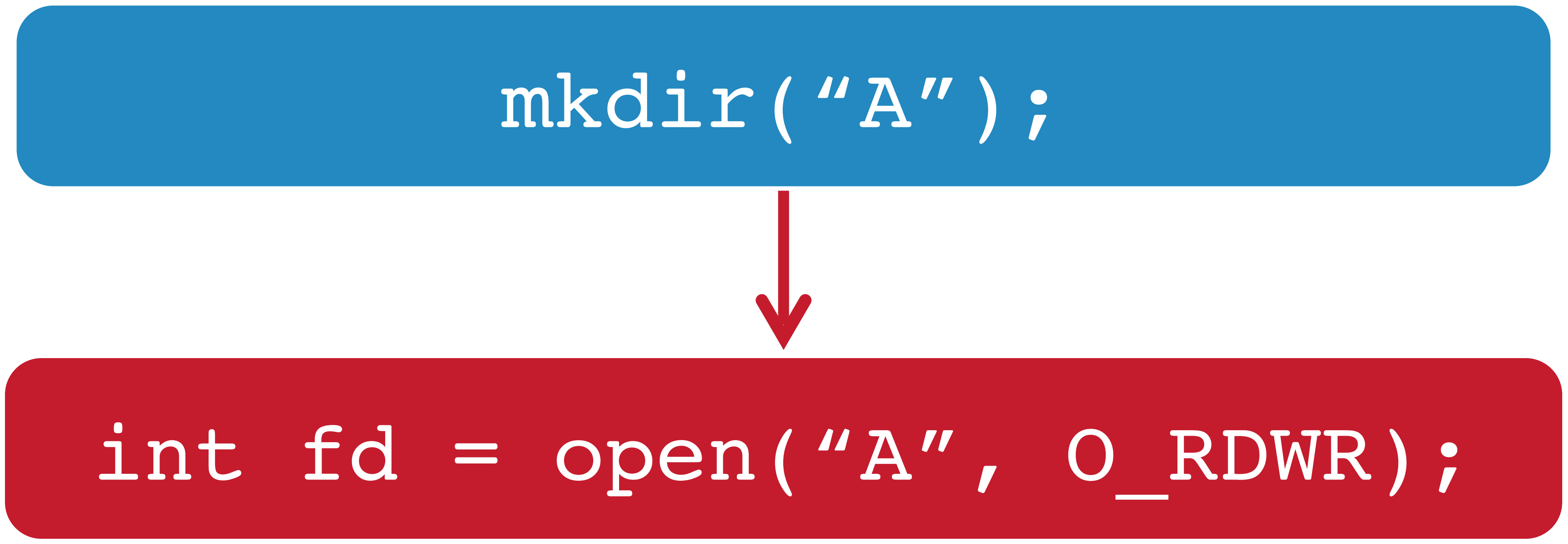
The *inter-dependence* between
file operations and files on an image

[2] GENERATING FILE OPERATIONS BY SPECS

```
* open(filename, flag)
* rename(filename, filename)
* mkdir(filename)
* unlink(filename)
* read(fd, buffer, int)
* write(fd, buffer, int)
```

Static rules *(definitions of syscalls) used by Syzkaller*

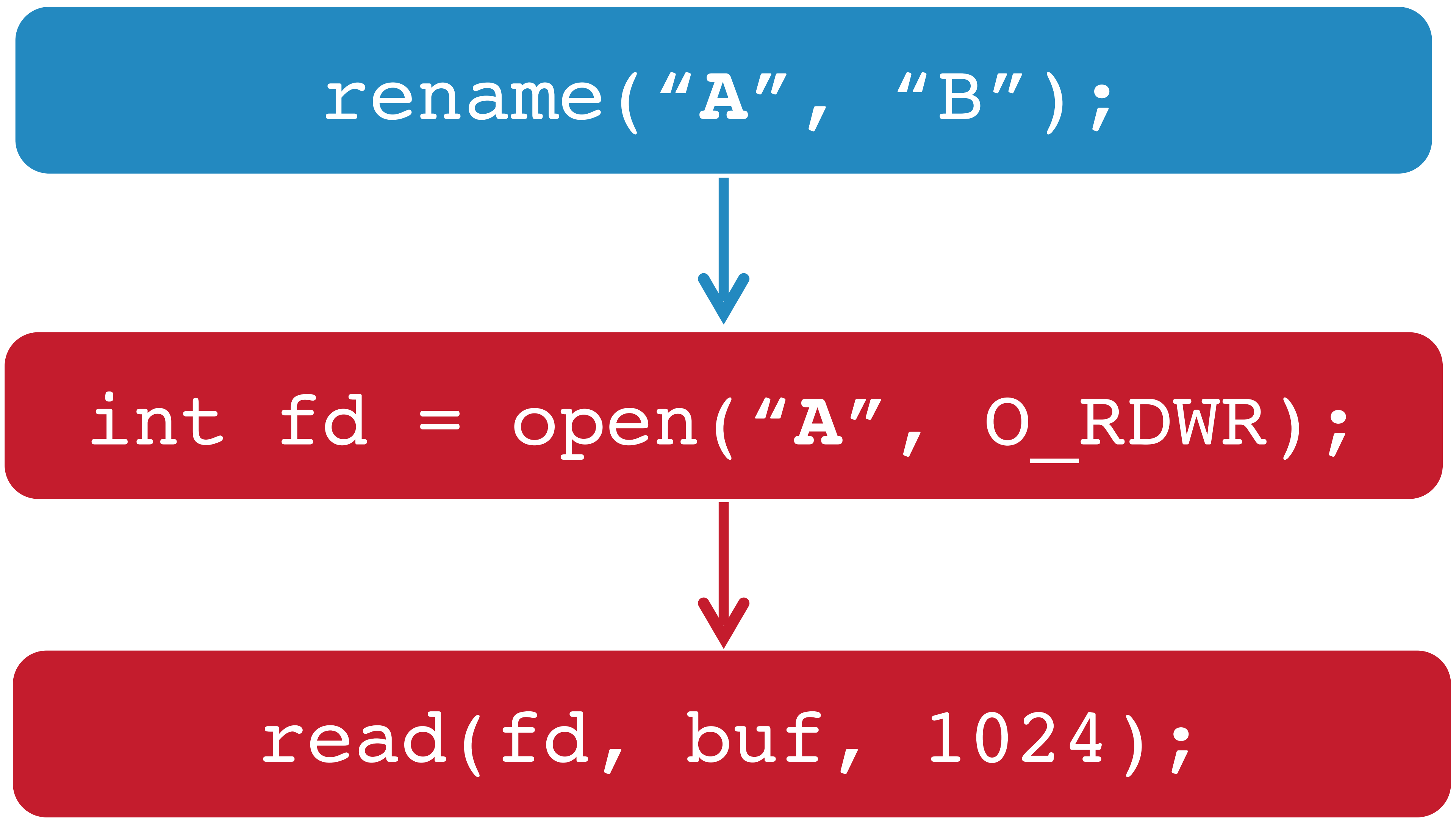
COUNTER EXAMPLE 1



```
mkdir(“A”);
```

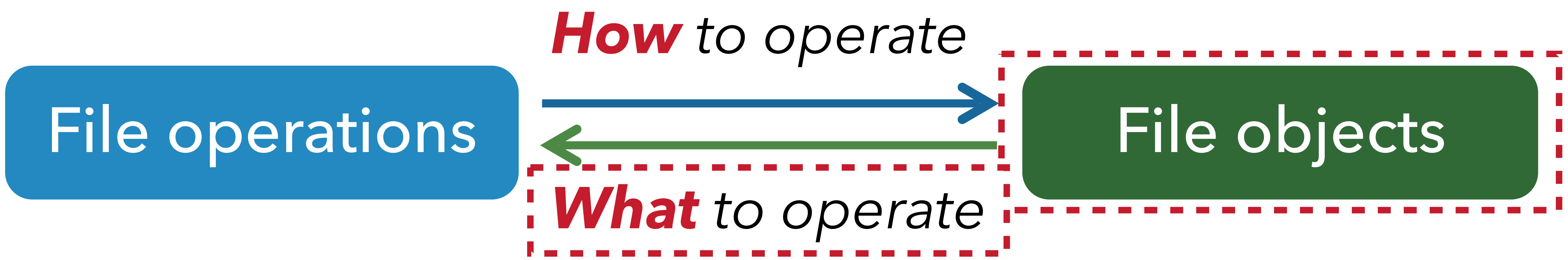
```
int fd = open(“A”, O_RDWR);
```

COUNTER EXAMPLE 2



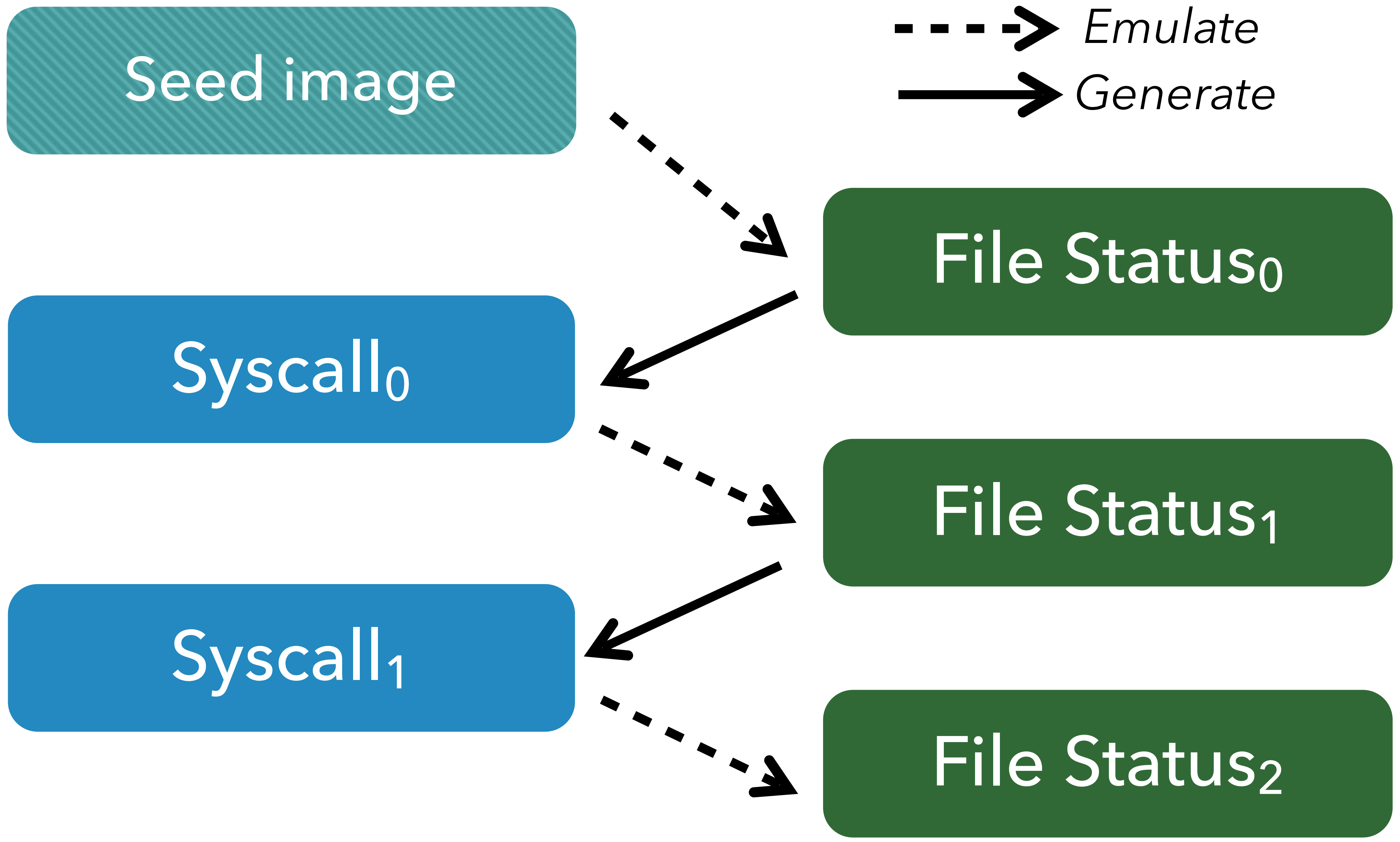
```
rename("A", "B");
int fd = open("A", O_RDWR);
read(fd, buf, 1024);
```

FILE OPERATIONS REVISITED



The *inter-dependence* between
file operations and files on an image

OUR APPROACH: CONTEXT-AWARE GENERATION



[3] FUZZING OS MODULES WITH VMS

- Conventional file systems are in-kernel modules
- OS fuzzers fuzz with VMs
  - Never reboot until a VM crashes

Performance

Aging kernel

Unstable executions

Hard-to-reproduce bugs

OUR APPROACH: LIBOS-BASED OS FUZZING

- We use library OS to fuzz OS.
  - A user application linked with a library OS invokes syscalls in user space.

Run on the same host

- Coverage monitoring
- Testcase sharing

Fast reboot
~10ms

- Non-aging OSes and modules
- Stable executions
- PoCs debugging

[4] FUZZING BOTH IMAGES AND SYSCALLS

No existing fuzzing platforms supports jointly fuzzing *binary blobs* and *API calls!*

We propose Janus, which co-ordinates fuzzing in two dimensions.

JANUS FOUND BUGS

- We run Janus for 4 months against 8 file systems on 1 workstation.
  - **90** unique bugs in total
  - **62** confirmed unknown bugs
  - **32** assigned CVEs
- During the period, Syzkaller found and fixed *8* bugs, and only one of them is missed by Janus.

SELECTED BUGS

FS	#0days/#critical	#mount-only
ext4 [*]	16 (12)	1
XFS	7 (2)	0
Btrfs	8 (2)	5
F2FS	11 (5)	5
Overall	42	11

* ext4 developers responded most actively to our bug reports.

JANUS

- A coverage-driven fuzzers that efficiently and effectively test images and file operations in a joint manner.

ARCHITECTURE OVERVIEW

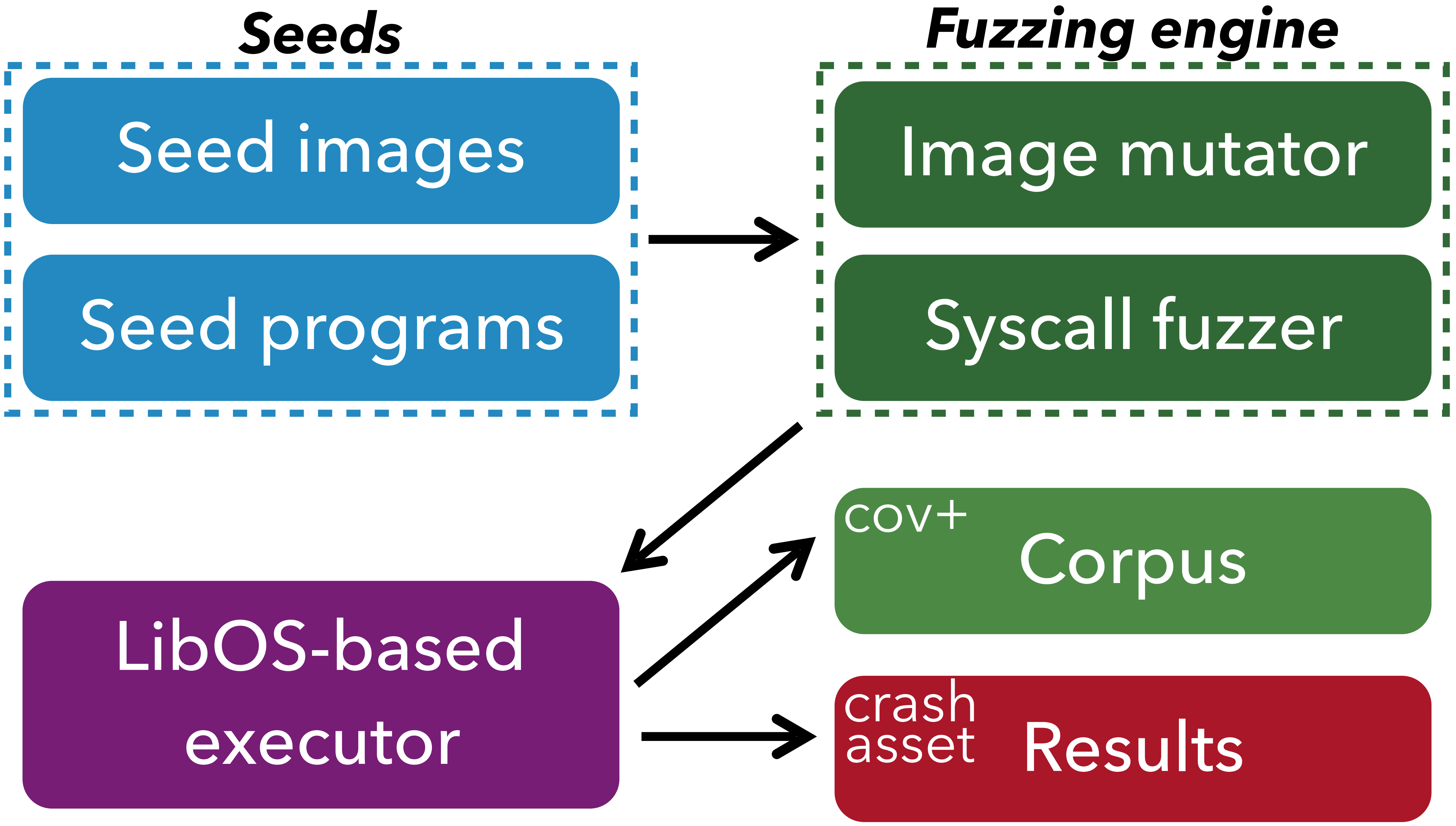
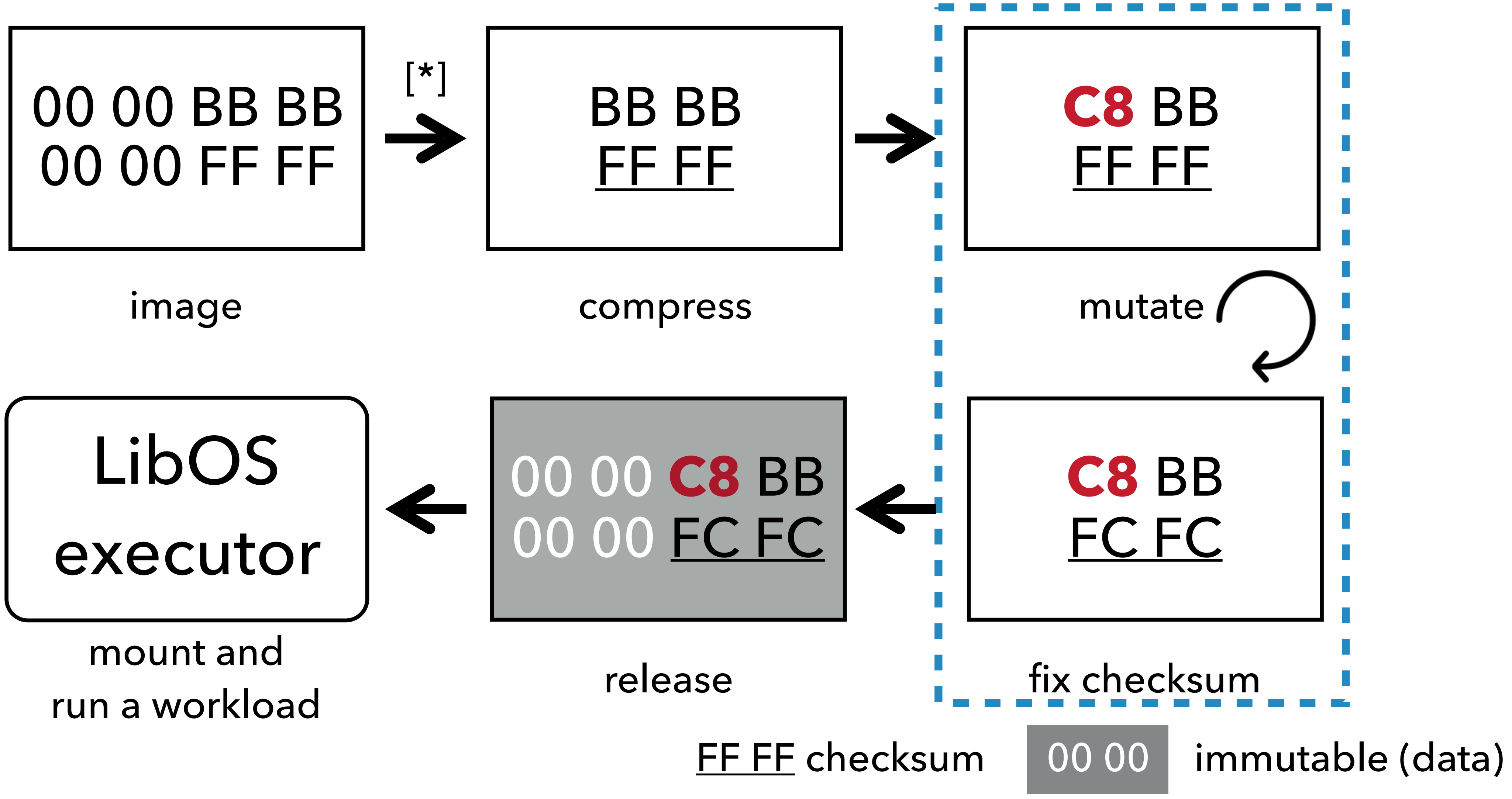
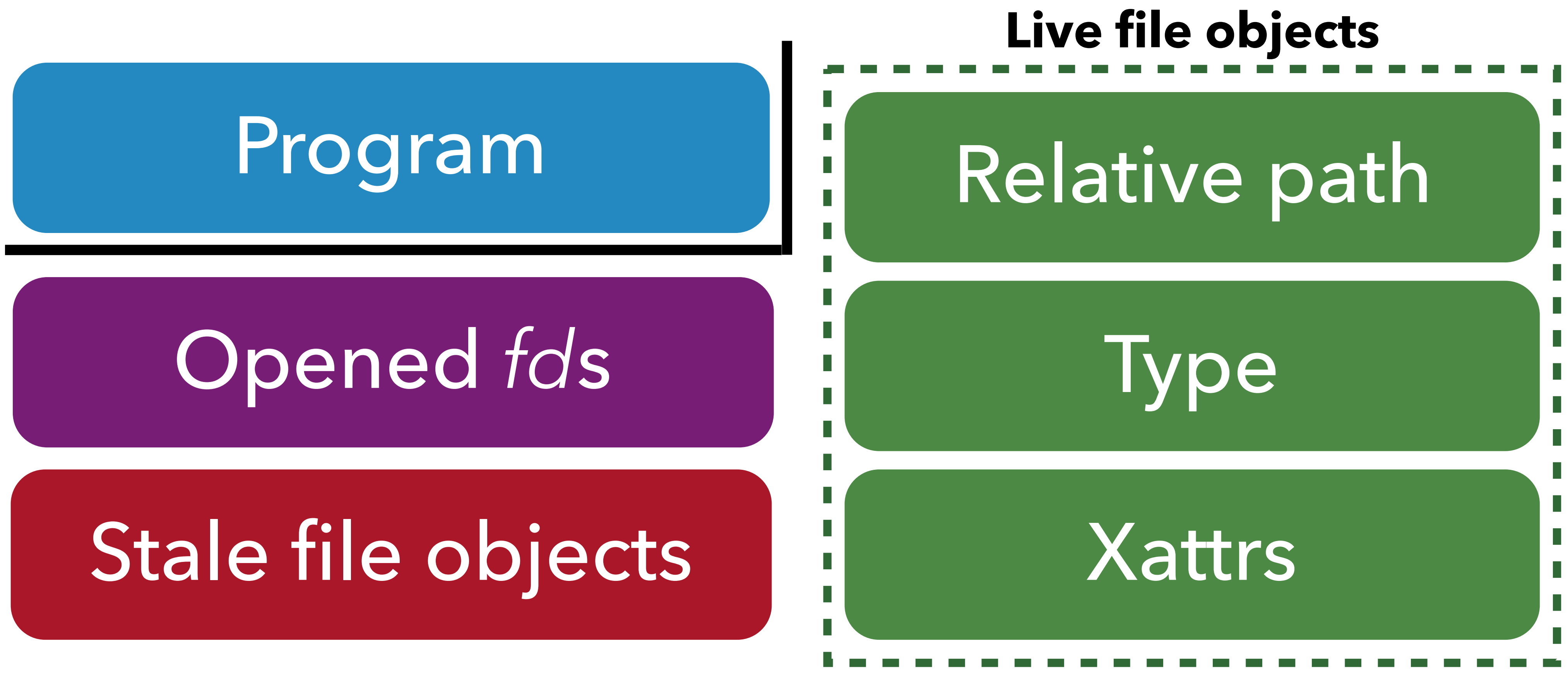


IMAGE MUTATOR



* We develop a specific image parser for each target file system.

SYSCALL FUZZER

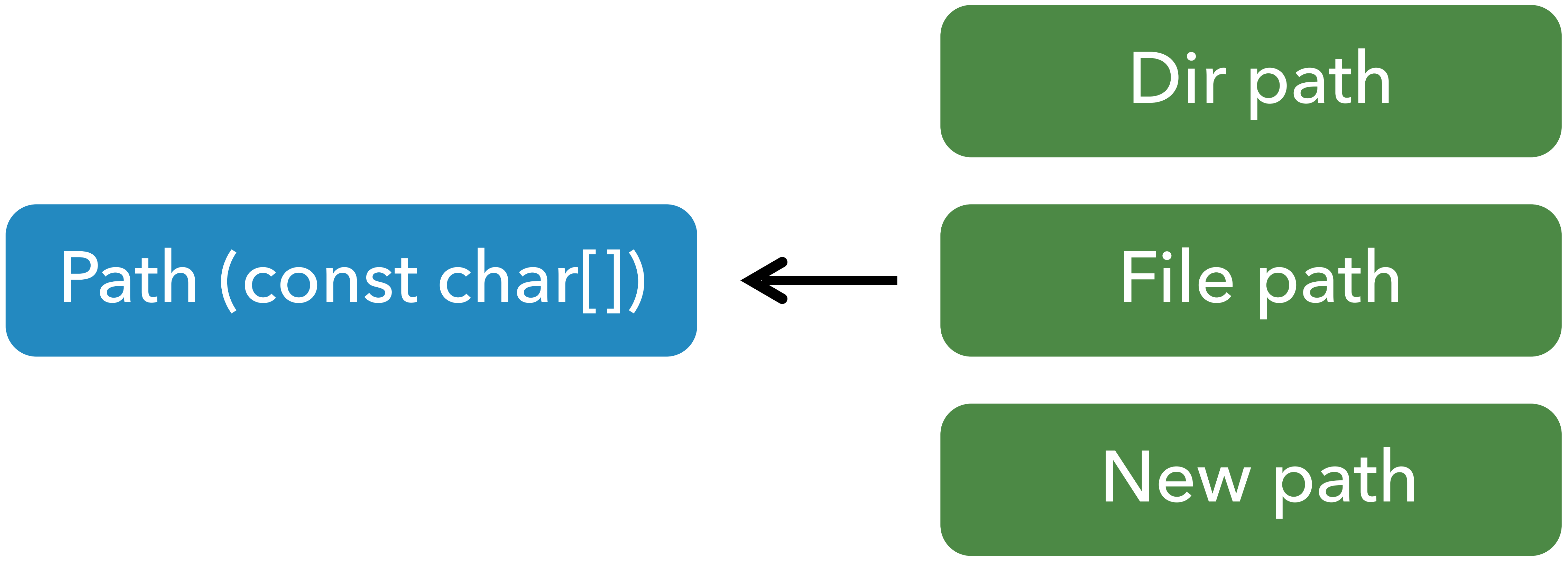


A testcase of Janus' syscall fuzzer

SYSCALL FUZZER

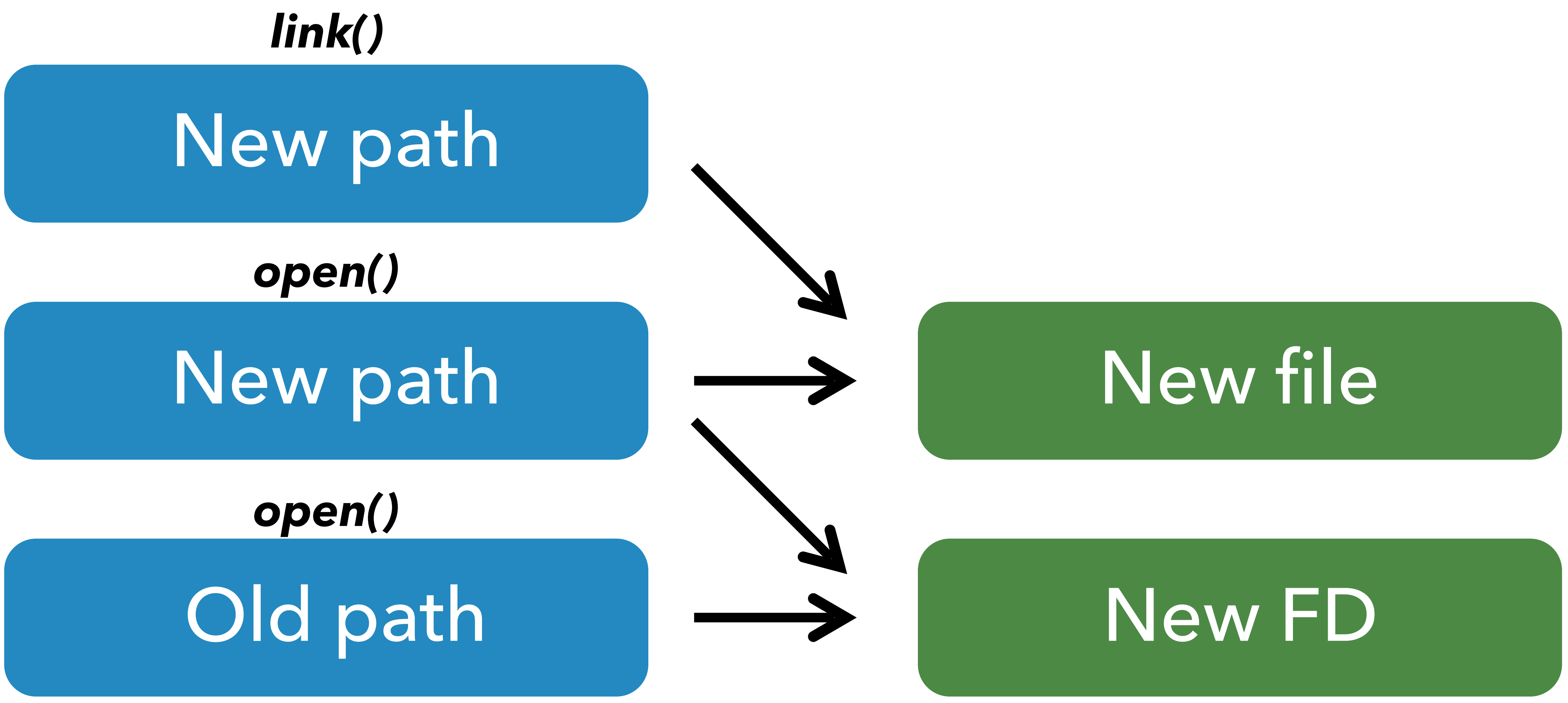
- *Phase 1:* Generate based on the context
  - **Mutating** the argument of an existing syscall
  - or **Appending** a newly generated syscall
- *Phase 2:* Emulate
  - **Updating** the corresponding context

SYSCALL FUZZER



Argument generation

SYSCALL FUZZER



Context update

CO-ORDINATE TWO FUZZERS

- First, Janus mutates images.

The image indicates the *initial state* of a file system, and its impact on file operations gradually decreases.

- Second, Janus launches its syscall fuzzer *without new coverage*.

Introducing new syscalls quickly increases the mutation space and erase the changes from past syscalls.

IMPLEMENTATION

IMPLEMENTATION OVERVIEW

- Janus is a variant of AFL.
    - Image parsers (8 FSes) 5,229 lines of C++
    - Syscall fuzzing 4,300 lines of C++
- Janus selects Linux Kernel Library as its LibOS solution.
    - Syscall executor 851 lines of C++
    - KASAN support 804 lines of C
    - Instrumentation for coverage 360 lines of C++
- Janus supports fuzzing **8** file systems on Linux.
    - ext4, XFS, btrfs, F2FS, GFS2, HFS+, ReiserFS, and vFAT
- Janus supports fuzzing **34** system calls for file operation.

EVALUATION

- We compared with the state-of-the-art OS fuzzer, *Syzkaller.*
- We used the same machine, seed images and starting programs to fuzz 8 file systems.

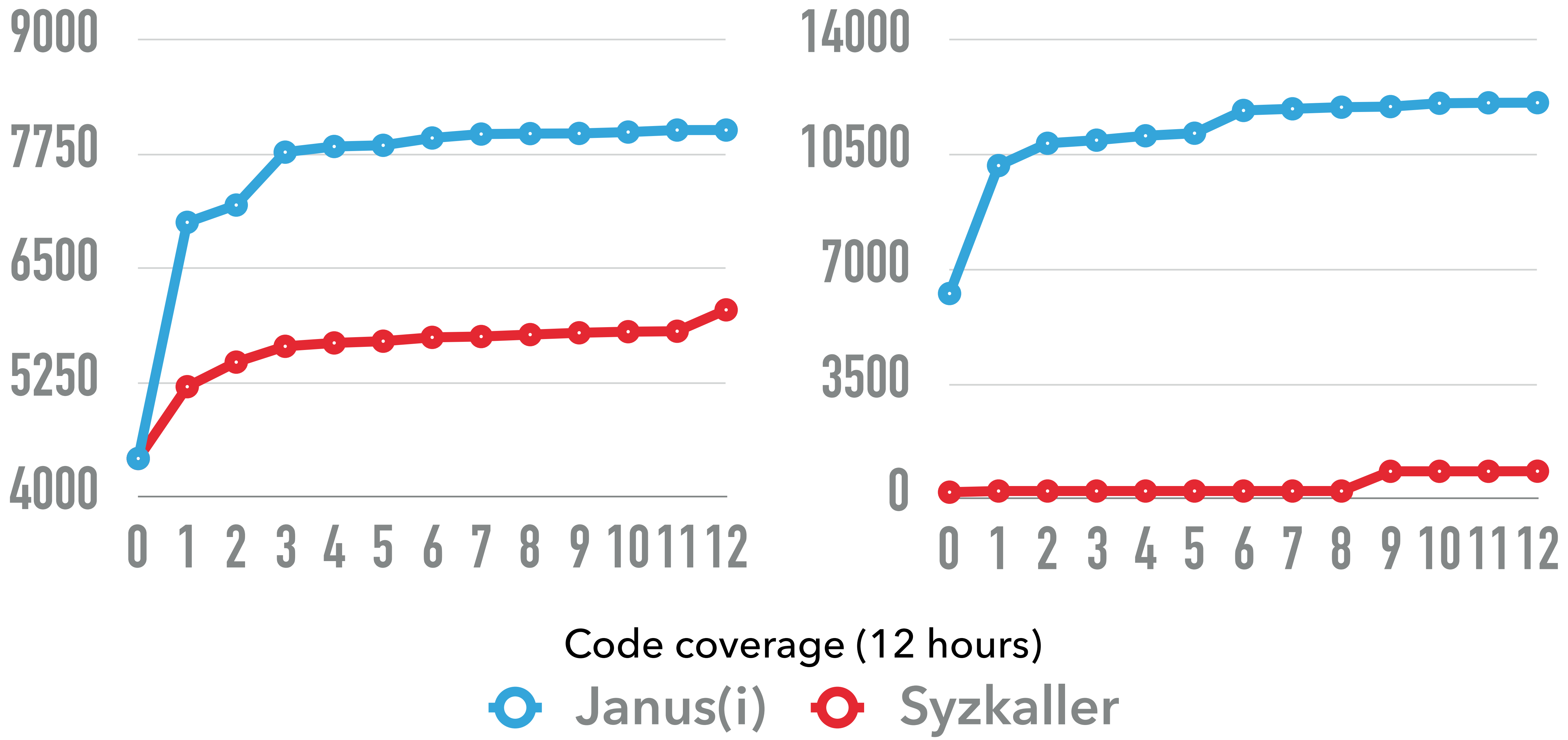
LIBOS REPRODUCE MORE BUGS

FS	Syzkaller (KVM)	Janus
ext4	0/3	196/196 (8)
XFS v5	0/6	24/24 (2)
Btrfs	0/0	1793/2054 (18)
F2FS	0/1288	2390/2458 (28)
Overall	0%	88% - 100%

#reproduced/#crashes (#unique) in 12 hours

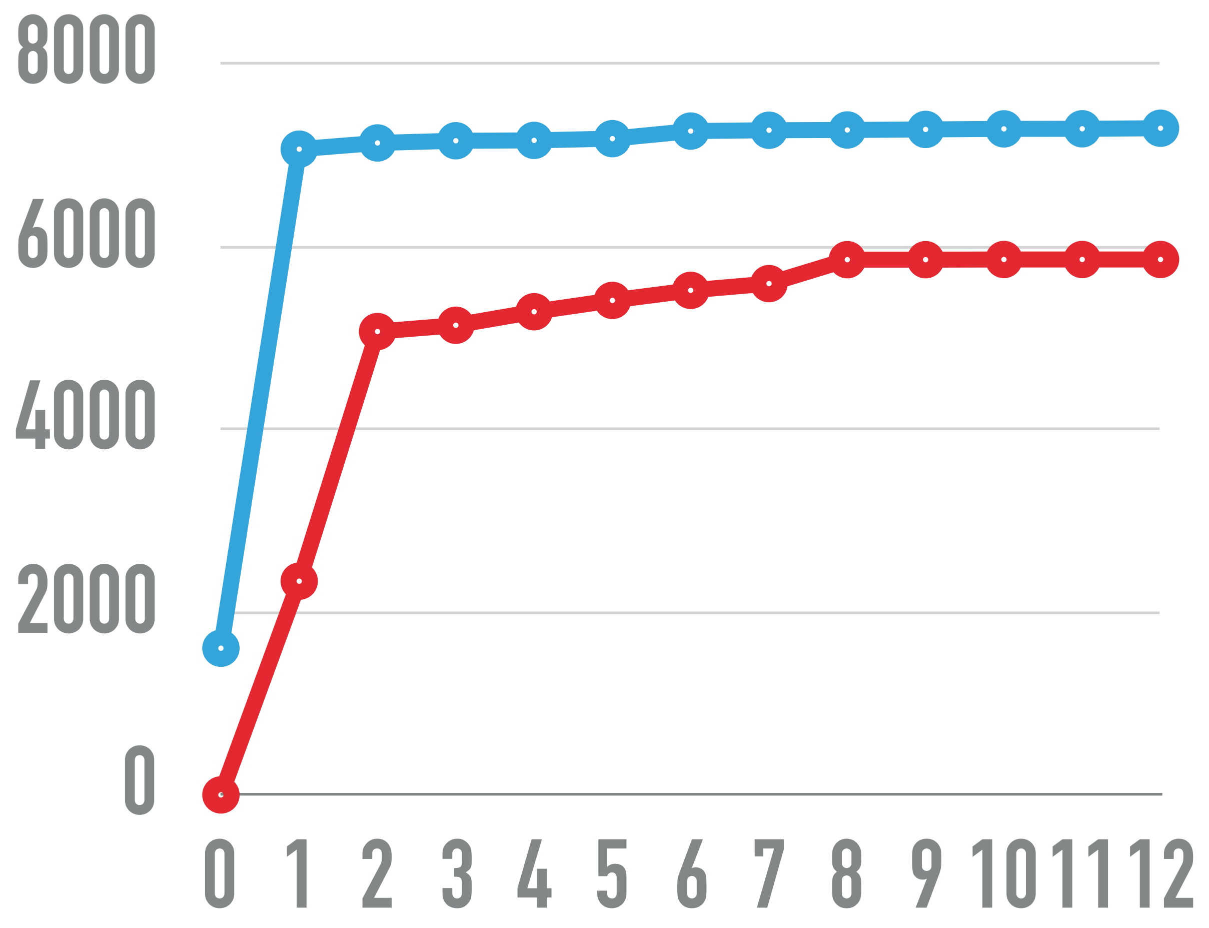
JANUS FUZZES IMAGES BETTER

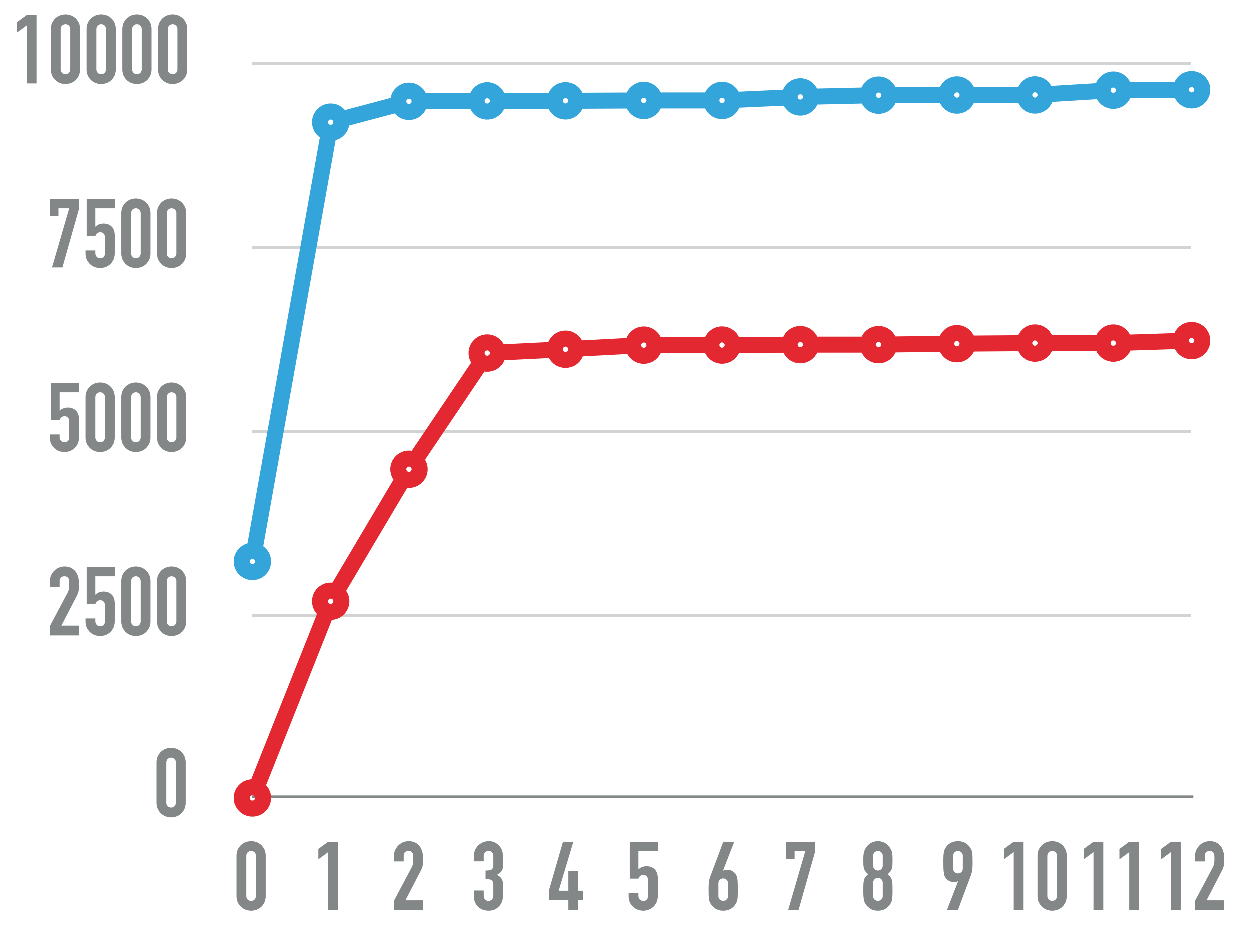
- ext4 (16MB seed): 1.5x
- XFS (16MB seed, checksum): 14.3x



JANUS FUZZES SYSCALLS BETTER

- ext4: 1.2x
- XFS: 1.5x

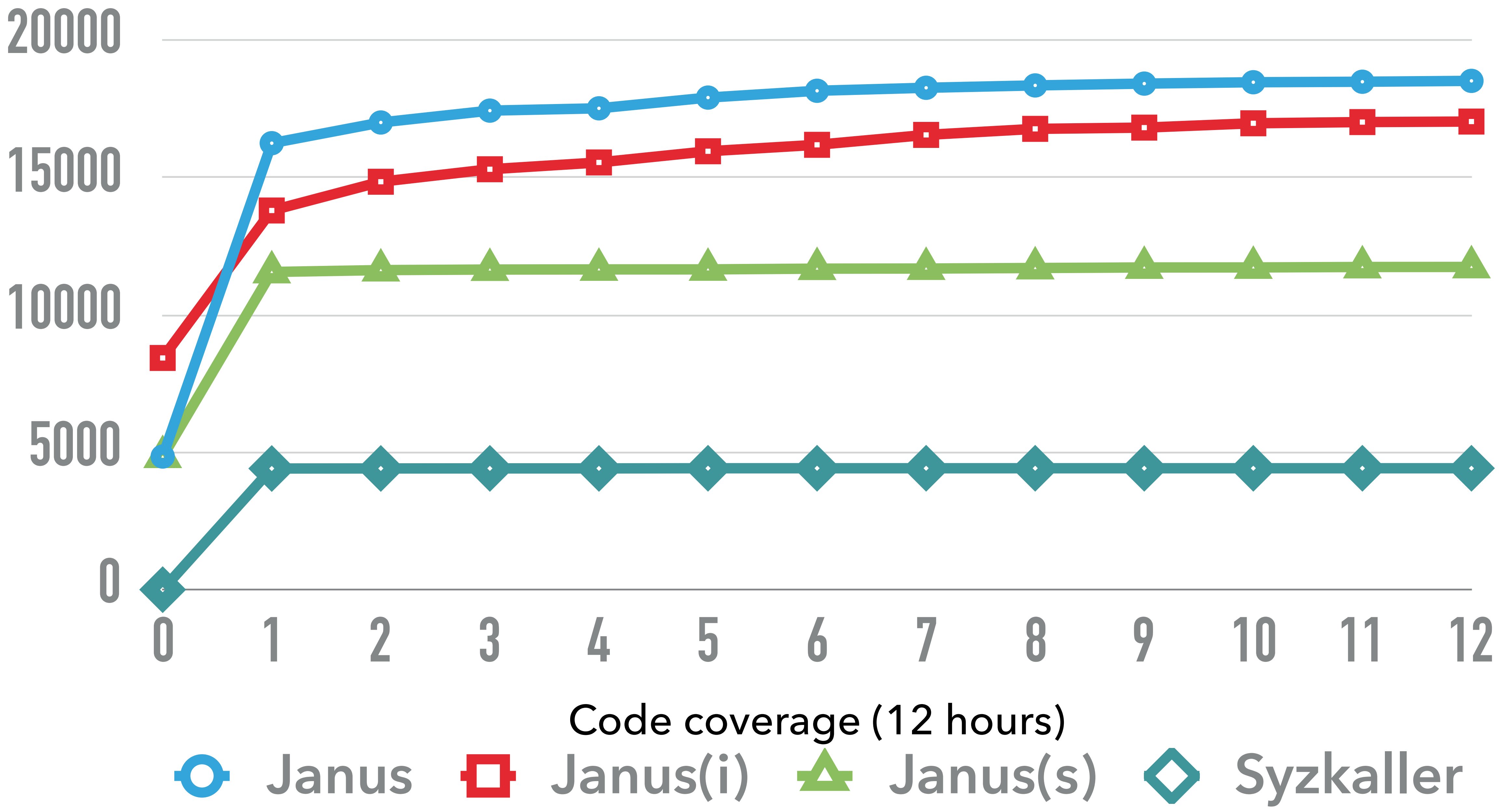




Code coverage (12 hours)

FUZZING BOTH IS MORE EFFECTIVE

- Btrfs (128MB seed): 4.2x



NOT ONLY MEMORY SAFETY BUGS ON LINUX

- *We believe Janus is a practical one-stop solution for all kinds of file system or even OS testing in the future.*
- Janus is easy to be extended for
  - Testing other types of file systems on other OSes
    - FUSE
    - Verified file systems
  - Finding other types of bugs
    - Crash consistency
    - Semantic correctness
- Further work is supported by **Google Faculty Research Award**.

THANKS

We will open source at https://github.com/sslab-gatech/janus