Cross-checking Semantic Correctness: The Case of Finding File System Bugs

Changwoo Min, Sanidhya Kashyap, Byoungyoung Lee,
Chengyu Song, Taesoo Kim

Georgia Institute of Technology
School of Computer Science

Two promising approaches to make bug-free software

- Formal proof → require “proof”
  - Guarantee high-level invariants (e.g., functional correctness)
- Model checking → require “model”
  - Check if code fits with domain model (e.g., locking rules)

Two promising approaches to make bug-free software

- Formal proof → require “proof”
  - Guarantee high-level invariants (e.g., functional correctness)

- Model checking → require “model”
  - Check if code fits with domain model (e.g., locking rules)

In practice, many software are (already) built without such theories

There exist many similar implementations of a program

- File systems: >50 implementations in Linux
- JavaScript: ECMAScript, V8, SpiderMonkey, etc
- POSIX C Library: Gnu Libc, FreeBSD, eLibc, etc

Without proof or model,
can we leverage
these existing implementations?

There exist many similar implementations of a program

- File systems: >50 implementations in Linux
- JavaScript: ECMAScript, V8, SpiderMonkey, etc
- POSIX C Library: Gnu Libc, FreeBSD, eLibc, etc

Without proof or model,
can we leverage
these existing implementations?

File system bugs are critical

Ubuntu
linux package

2013-01-07

Overview | Code | **Bugs** | Blueprints | Translations | Answers

Risk of filesystem corruption with ext3 in lucid

Bug #1097042 reported by lemonsqueeze on 2013-01-07

This bug affects 1 person

Affects	Status	Importance	Assigned to
linux (Ubuntu)	Expired	Medium	Unassigned

Also affects project | Also affects distribution/package | Nominate for series

Bug Description

```
On my system, a default ext3 mount (no fstab entry) results in:
$ cat /proc/mounts
/dev/sda6 /media/space ext3 rw,nosuid,nodev,relatime,errors=continue,
user_xattr,acl,data=ordered 0 0

We can see the "barrier=1" option is missing by default, which can cause
severe filesystem corruption in case of power failure (i've been hit
recently). Quoting mount(1):
```

File system bugs are critical

Ubuntu
linux package **2013-01-07**

Overview Code **Bugs** Blueprints Translations Answers

Risk of filesystem corruption with ext3 in lucid

Bug #1097042 report

This bug affects 1

Affects

linux (U

Also affects pr

Bug Descripti

```
On my syste
$ cat /proc
/dev/sda6 /
user_xattr,

We can see
severe file
recently).
```

Ubuntu
linux package **2014-10-17**

Overview Code **Bugs** Blueprints Translations Answers

XFS: memory allocation deadlock in kmem_alloc (mode:0x8250)

Bug #1382333 reported by Rafael David Tinoco on 2014-10-17

This bug affects 3 people

Affects	Status	Importance	Assigned to	Milestone
linux (Ubuntu)	Fix Released	Undecided	Unassigned	
Trusty	Fix Released	Undecided	Rafael David Tinoco	
Utopic	Fix Released	Undecided	Unassigned	

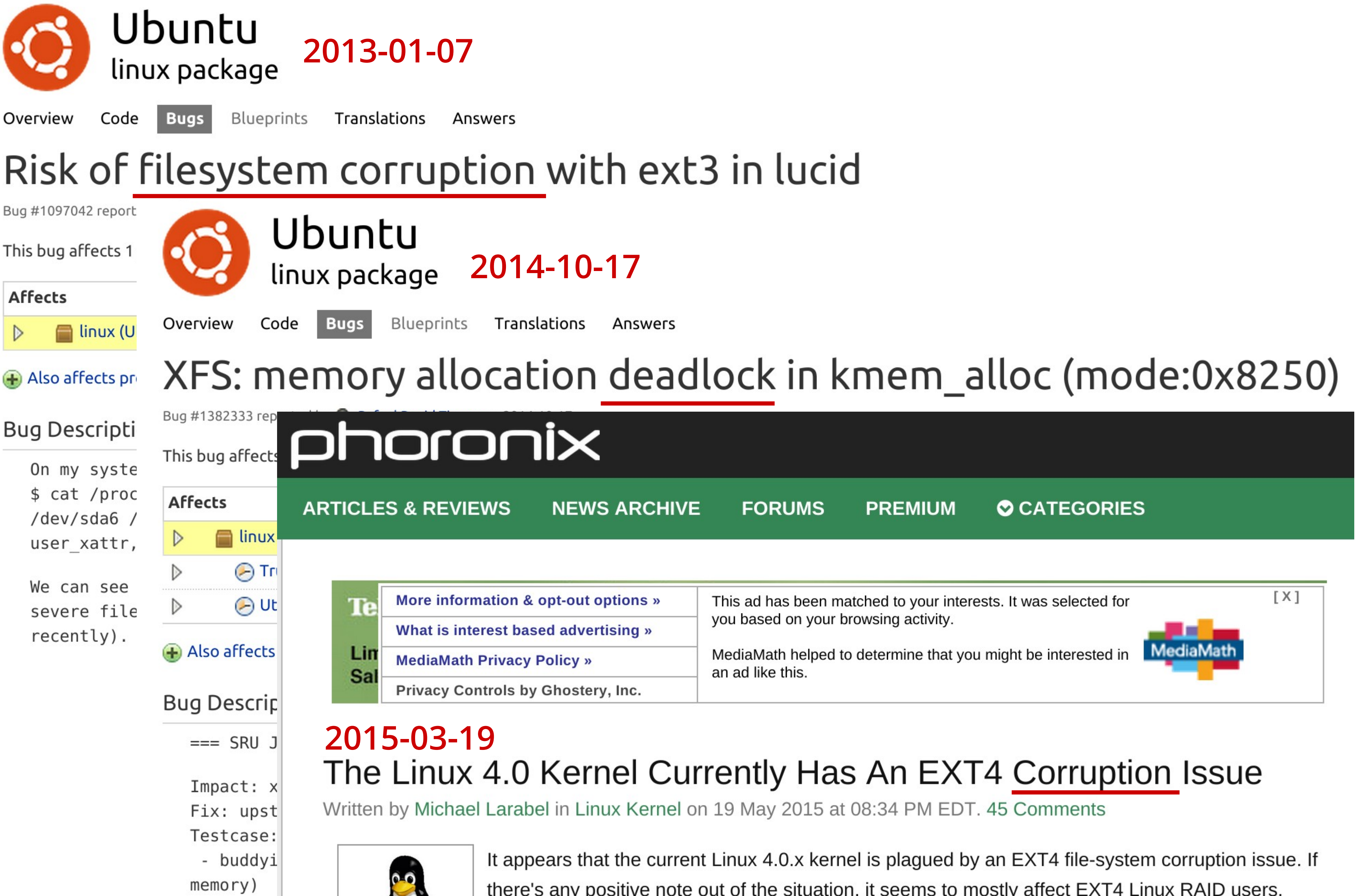
Also affects project Also affects distribution/package Nominate for series

Bug Description

```
=== SRU Justification ===

Impact: xfs can hang on lack of contiguous memory page to be allocated.
Fix: upstream patch (b3f03bac8132207a20286d5602eda64500c19724).
Testcase:
 - buddyinfo showing lack of contiguous blocks to be allocated (fragmented
memory)
```

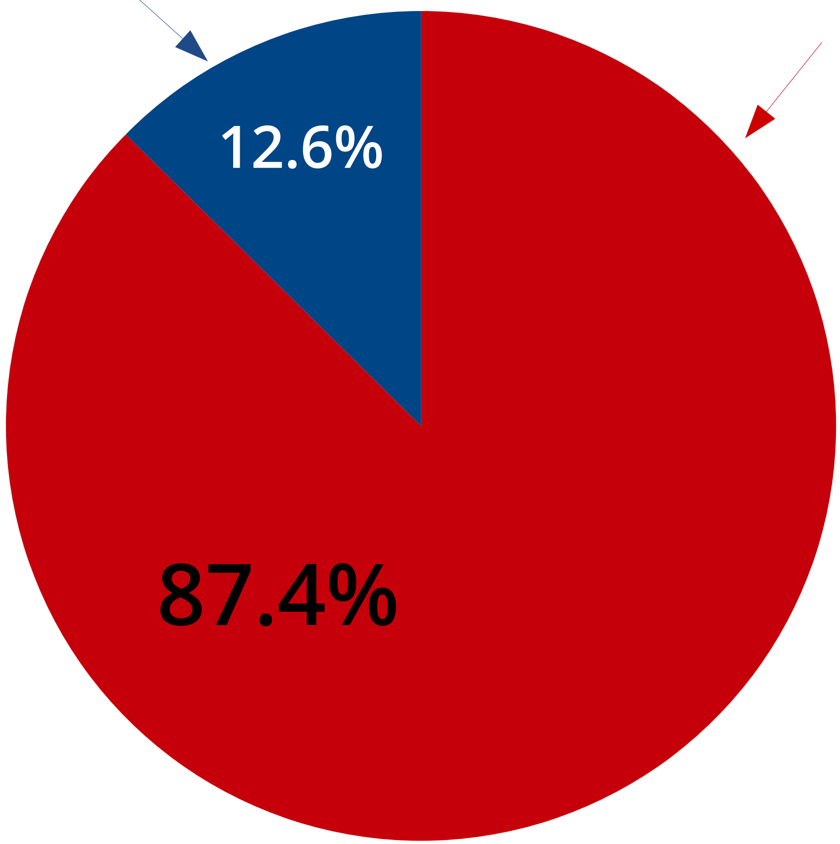
File system bugs are critical



A majority of bugs in file systems are hard to detect

Memory bugs:

NULL dereference
Use-after-free
…



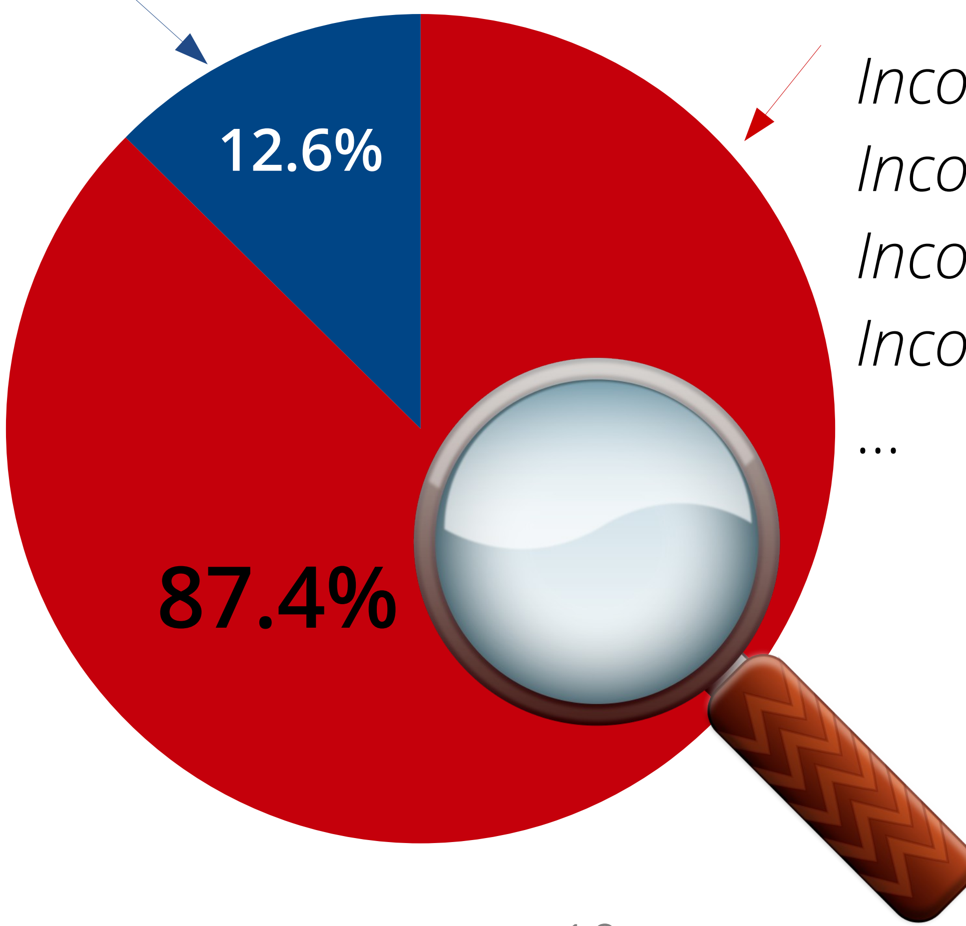
Semantic bugs:

Incorrect condition check
Incorrect statue update
Incorrect argument
Incorrect error code
…

A majority of bugs in file systems are hard to detect

Memory bugs:

NULL dereference
Use-after-free
…



Semantic bugs:

Incorrect condition check
Incorrect statue update
Incorrect argument
Incorrect error code
…

Example of semantic bug: Missing capability check in OCFS2

ocfs2: trusted xattr missing CAP_SYS_ADMIN check
Signed-off-by: Sanidhya Kashyap <sanidhya@gatech.edu>

...

@@ static size_t **ocfs2_xattr_trusted_list**

```
+     if (!capable(CAP_SYS_ADMIN))
+             return 0;
```

Example of semantic bug: Missing capability check in OCFS2

ocfs2: trusted xattr missing CAP_SYS_ADMIN check
Signed-off-by: Sanidhya Kashyap <sanidhya@gatech.edu>
...

@@ static size_t **ocfs2_xattr_trusted_list**

```
+     if (!capable(CAP_SYS_ADMIN))
+             return 0;
```

Can we find this bug by leveraging other implementations?

A majority of file system already implemented capability check

ocfs2: trusted xattr missing CAP_SYS_ADMIN check
Signed-off-by: Sanidhya Kashyap <sanidhya@gatech.edu>

```
...
@@ static size_t ocfs2_xattr_trusted_list

+       if (!capable(CAP_SYS_ADMIN))
+               return 0;
```

- **ext2**

```
static size_t ext2_xattr_trusted_list()
        if (!capable(CAP_SYS_ADMIN))
                return 0;
```

- **ext4**

```
static size_t ext4_xattr_trusted_list()
        if (!capable(CAP_SYS_ADMIN))
                return 0;
```

- **XFS**

```
static size_t xfs_xattr_put_listent()
        if ((flags & XFS_ATTR_ROOT) &&
            !capable(CAP_SYS_ADMIN))
                return 0;
```

…

A majority of file system already implemented capability check

ocfs2: trusted xattr missing CAP_SYS_ADMIN check
Signed-off-by: Sanidhya Kashyap <sanidhya@gatech.edu>
...

```
@@ static size_t ocfs2_xattr_trusted_list

+     if (!capable(CAP_SYS_ADMIN))
+             return 0;
```

Deviant implementation → potential bugs?

- **ext2**

```
static size_t ext2_xattr_trusted_list()
        if (!capable(CAP_SYS_ADMIN))
                return 0;
```

- **ext4**

```
static size_t ext4_xattr_trusted_list()
        if (!capable(CAP_SYS_ADMIN))
                return 0;
```

- **XFS**

```
static size_t xfs_xattr_put_listent()
        if ((flags & XFS_ATTR_ROOT) &&
            !capable(CAP_SYS_ADMIN))
                return 0;
```

...

A majority of file system already implemented capability check

ocfs2: trusted xattr missing CAP_SYS_ADMIN check
Signed-off-by: Sanidhya Kashyap <sanidhya@gatech.edu>
...
@@ static size_t **ocfs2_xattr_trusted_list**

```
+     if (!capable(CAP_SYS_ADMIN))
+             return 0;
```

Deviant implementation → potential bugs?

A new bug we found
It has been hidden for 6 years

- **ext2**

static size_t **ext2_xattr_trusted_list()**

```
    if (!capable(CAP_SYS_ADMIN))
        return 0;
```

- **ext4**

static size_t **ext4_xattr_trusted_list()**

```
    if (!capable(CAP_SYS_ADMIN))
        return 0;
```

- **XFS**

static size_t **xfs_xattr_put_listent()**

```
    if ((flags & XFS_ATTR_ROOT) &&
        !capable(CAP_SYS_ADMIN))
        return 0;
```

...

Case study: Write a page

- Each file system defines how to write a page
- Semantic of writepage()
  - Success → return locked page
  - Failure → return unlocked page
- Document/filesystems/vfs.txt specifies such rule
  - Hard to detect without domain knowledge

What if 99% file systems follow above pattern, but not one file system? bug?

Our approach can reveal such bugs without domain specific knowledge

- 52 file systems follow the locking rules
- But 2 file systems don't (Ceph and AFFS)

```
-------------------------------- fs/ceph/addr.c --------------------------------
index fd5599d..e723482 100644
@@ static int ceph_write_begin
+     if (r < 0)
+             page_cache_release(page);
+     else
+             *pagep = page;
```

Our approach can reveal such bugs without domain specific knowledge

- 52 file systems follow the locking rules
- But 2 file systems don't (Ceph and AFFS)

```
------------------------------- fs/ceph/addr.c -------------------------------
index fd5599d..e723482 100644
@@ static int ceph_write_begin
+     if (r < 0)
+             page_cache_release(page);
+     else
+             *pagep = page;
```

We found 3 bugs in 2 file systems
Hidden for over 5 years

Our approach in finding bugs

Intuition:

Bugs are rare

Majority of implementations is correct

Idea:

Find deviant ones as potential bugs

Our approach is promising in finding semantic bugs (Example: file systems)

- New semantics bugs
  - 118 new bugs in 54 file systems
- Critical bugs
  - System crash, data corruption, deadlock, etc
- Bugs difficult to find
  - Bugs were hidden for 6.2 years on average
- Various kinds of bugs
  - Condition check, argument use, return value, locking, etc

Technical challenges

- All software are different one way or another
  - e.g., disk layout in file system

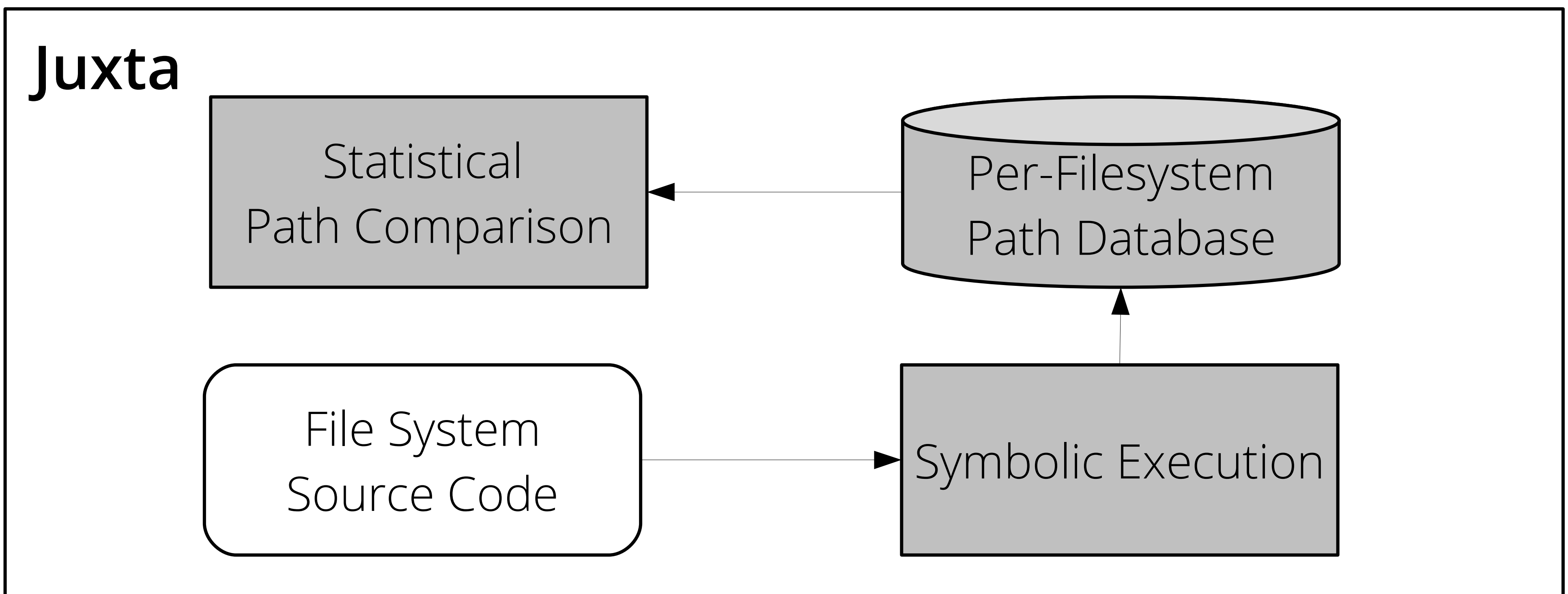
- How to compare different implementation?
  - **Q1:** Where to start?
  - **Q2:** What to compare?
  - **Q3:** How to compare?

Juxta: the case of file system

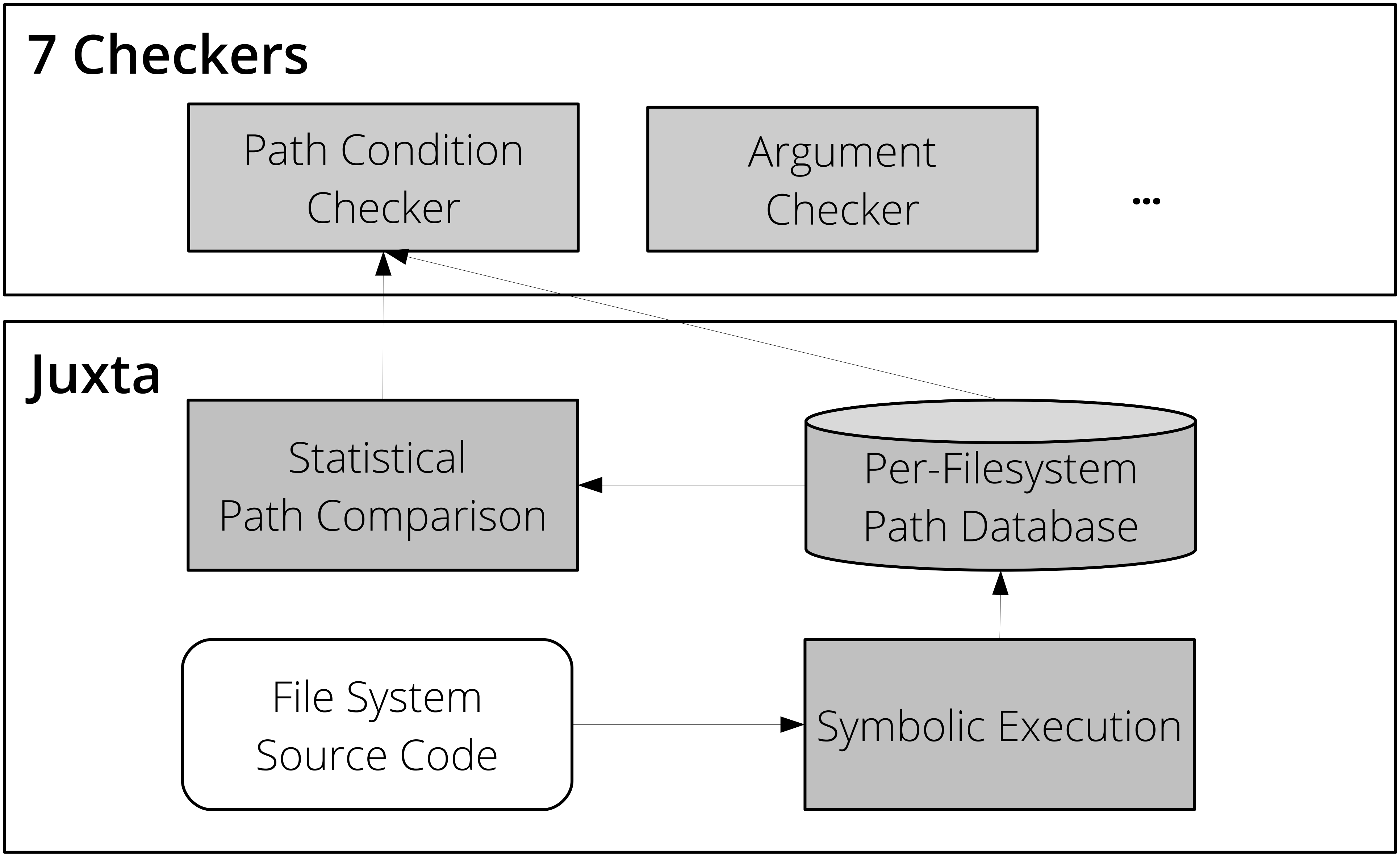
- All file systems should follow VFS API in Linux
  - e.g., vfs_rename() in each file system

- How to compare different file systems?
  - **Q1:** Where to start? → VFS entries in file system
  - **Q2:** What to compare? → symbolic environment
  - **Q3:** How to compare? → statistical comparison

Juxta overview



Juxta overview



Comparing multiple file systems

- Q1: Where to start?
  - Identifying semantically similar entry points
- Q2: What to compare?
  - Building per-path symbolic environment
- Q3: How to compare?
  - Statistically comparing each path

Comparing multiple file systems

- Q1: Where to start?
  - Identifying semantically similar entry points
- Q2: What to compare?
  - Building per-path symbolic environment
- Q3: How to compare?
  - Statistically comparing each path

Identifying semantically similar entry points

- Linux Virtual File System (VFS)
  - Use common data structures and behavior (e.g., inode and page cache)
  - Define filesystem-specific interfaces (e.g., open, rename)

Example: inode_operations→rename()

```
struct inode_operations {
        int (*rename) (struct inode *, ...);
        int (*create) (struct inode *,...);
        int (*unlink) (struct inode *,..);
        int (*mkdir) (struct inode *,...);
};
```

Compare ***_rename()**
to find deviant **rename()** implementations.

Example: inode_operations→rename()

```
struct inode_operations {
    int (*rename) (struct inode *, ...);
    int (*create) (struct inode *,...);
    int (*unlink) (struct inode *,..);
    int (*mkdir) (struct inode *,...);
};
```

```
btrfs_rename(...);
ext4_rename(...);
xfs_vn_rename(...);
...
```

Compare ***_rename()**
to find deviant **rename()** implementations.

Comparing multiple file systems

- Q1: Where to start?
  - Identifying semantically similar entry points
- Q2: What to compare?
  - Building per-path symbolic environment
- Q3: How to compare?
  - Statistically comparing each path

Building per-path symbolic environment

- Context/flow-sensitive symbolic execution
  - C language level
  - Build symbolic environment per path

    (e.g., path cond, return values, side-effect, function calls)

- Key idea: return-oriented comparison
  - Error codes represent per-path semantics

    (e.g., comparing all paths returning EACCES in rename() implementations)

Example: Per-path symbolic environment

```
int foo_rename(int flag) {
  if (flag == RO)
    return -EACCES;

  inode→flag = flag;
  kmalloc(..., GFP_NOFS)
  return SUCCESS;
}
```

Execution Path Information

Example: Per-path symbolic environment

```
int foo_rename(int flag) {
  if (flag == RO)
    return -EACCES;

  inode→flag = flag;
  kmalloc(..., GFP_NOFS)
  return SUCCESS;
}
```

Execution Path Information

Example: Per-path symbolic environment

```
int foo_rename(int flag) {
  if (flag == RO)
    return -EACCES;

  inode→flag = flag;
  kmalloc(..., GFP_NOFS)
  return SUCCESS;
}
```

Execution Path Information

Condition	flag: !RO

Example: Per-path symbolic environment

```
int foo_rename(int flag) {
  if (flag == RO)
    return -EACCES;

  inode→flag = flag;
▶ kmalloc(..., GFP_NOFS)
  return SUCCESS;
}
```

Execution Path Information

Condition	flag: !RO
Side-effect	inode→flag = flag

Example: Per-path symbolic environment

```
int foo_rename(int flag) {
  if (flag == RO)
    return -EACCES;

  inode→flag = flag;
  kmalloc(..., GFP_NOFS)
▶ return SUCCESS;
}
```

Execution Path Information

Condition	flag: !RO
Side-effect	inode→flag = flag
Call	kmalloc(..., GFP_NOFS)

Example: Per-path symbolic environment

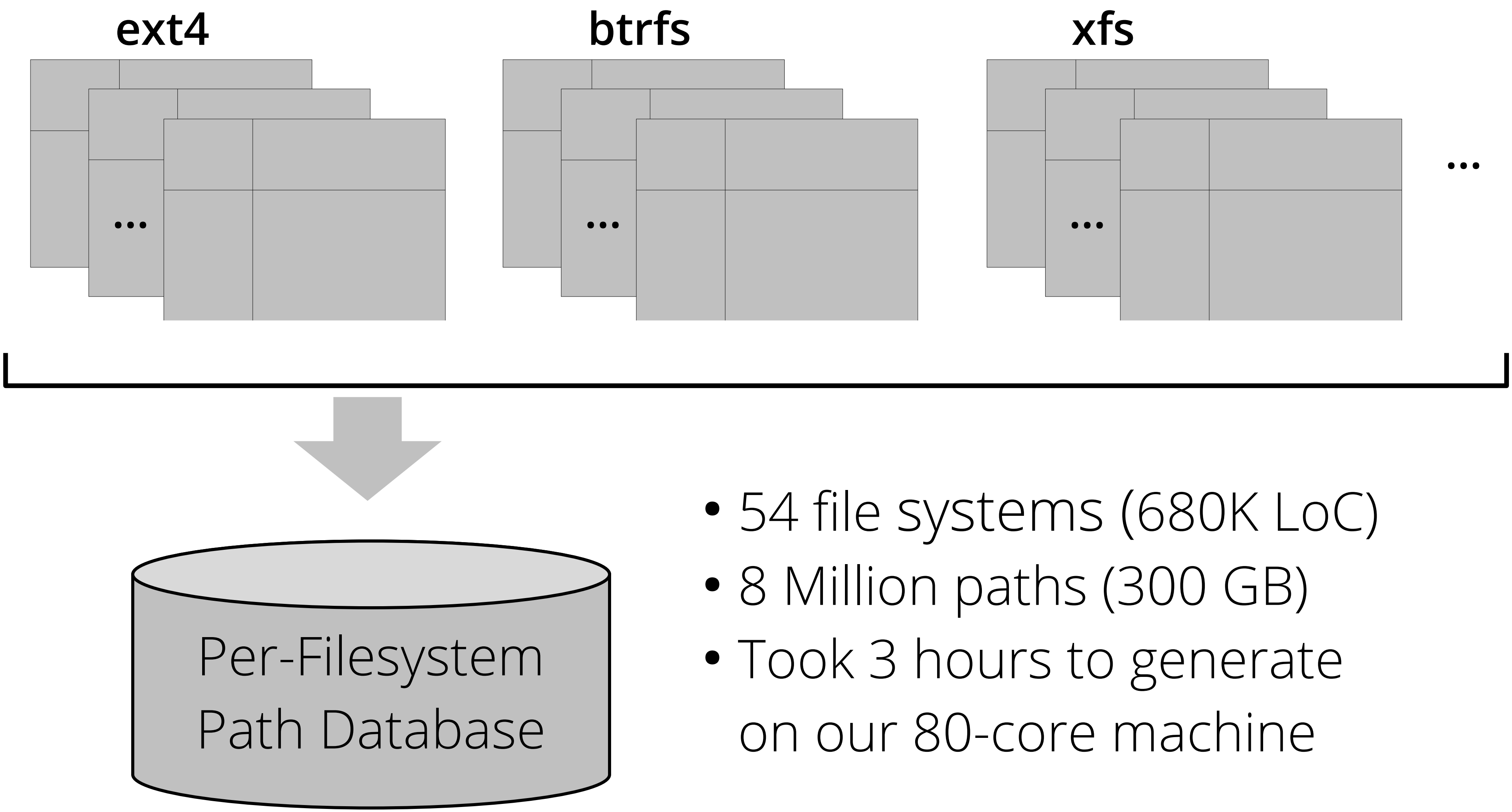
```
int foo_rename(int flag) {
  if (flag == RO)
    return -EACCES;

  inode→flag = flag;
  kmalloc(..., GFP_NOFS)
▶ return SUCCESS;
}
```

Execution Path Information

Condition	flag: !RO
Side-effect	inode→flag = flag
Call	kmalloc(..., GFP_NOFS)
Return	SUCCESS

Constructing path database

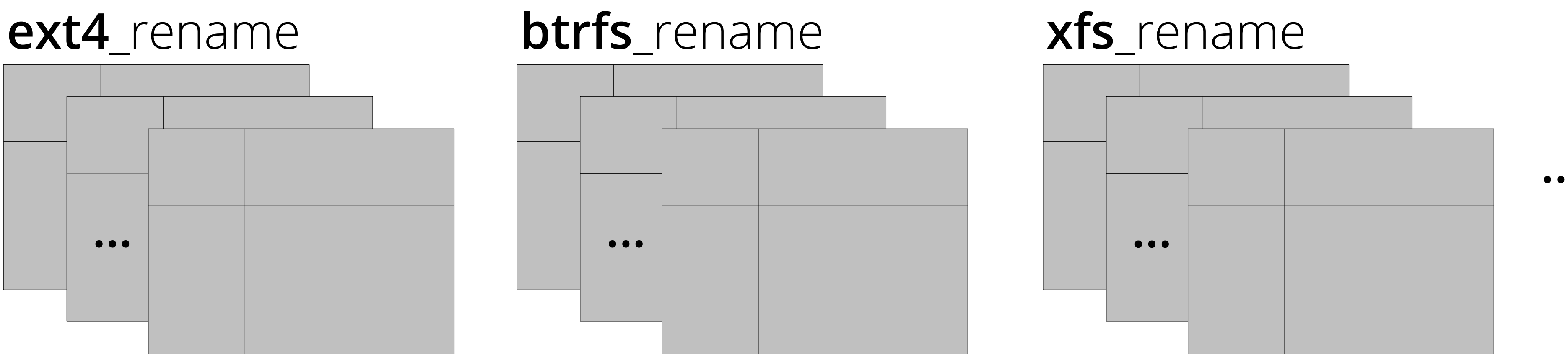


- 54 file systems (680K LoC)
- 8 Million paths (300 GB)
- Took 3 hours to generate on our 80-core machine

Comparing multiple file systems

- Q1: Where to start?
  - Identifying semantically similar entry points
- Q2: What to compare?
  - Building per-path symbolic environment
- Q3: How to compare?
  - Statistically comparing each path

Two types of per-path symbolic data

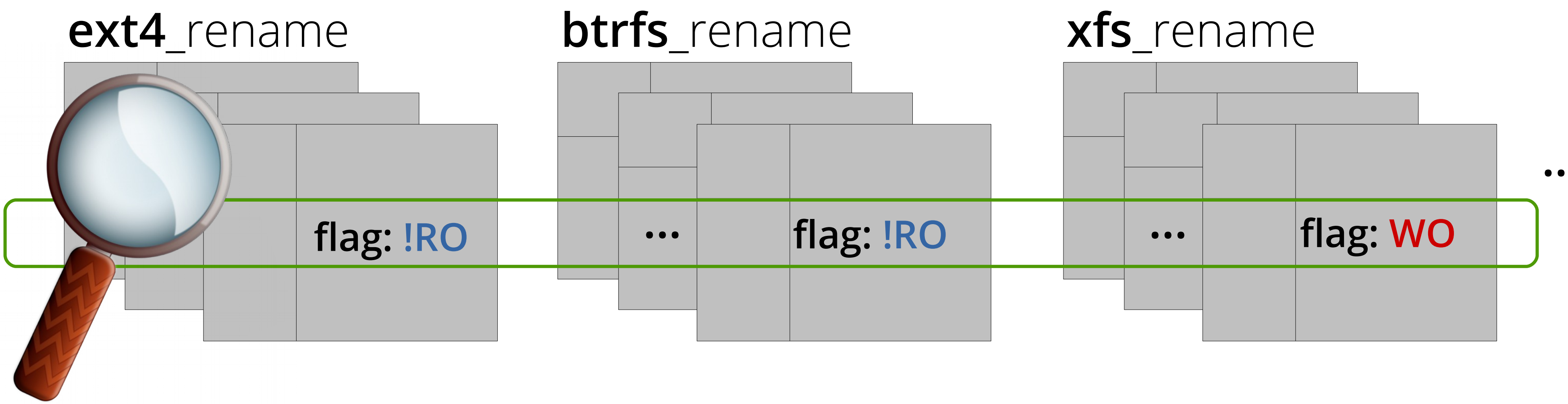


- Range data (or symbolic constraint)
  - *What is the* ***range of argument*** *for this execution path?*

    e.g., path condition, return value, etc.
- Occurrences
  - ***How many times a particular API flag*** *is used?*

    e.g., API argument usage, error handling, etc.

Two types of per-path symbolic data

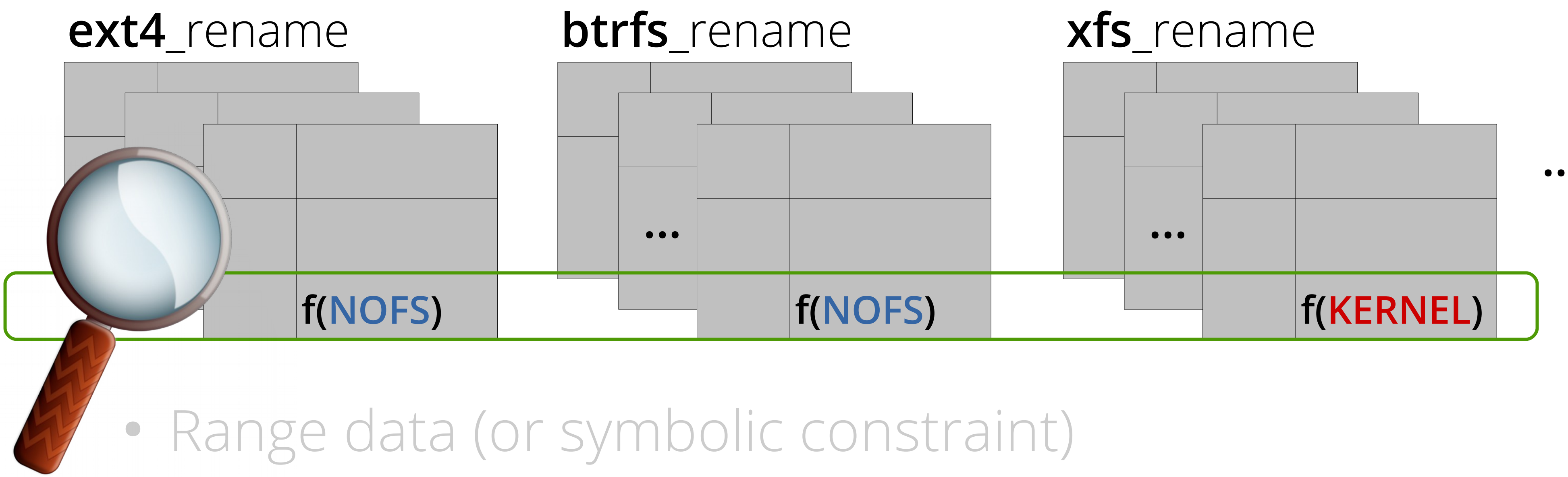


- Range data (or symbolic constraint)
  - *What is the **range of argument** for this execution path?*

    e.g., path condition, return value, etc.
- Occurrences
  - ***How many times a particular API flag** is used?*

    e.g., API argument usage, error handling, etc.

Two types of per-path symbolic data



- Range data (or symbolic constraint)
  - *What is the **range of argument** for this execution path?*
    e.g., path condition, return value, etc.
- Occurrences
  - ***How many times a particular API flag** is used?*
    e.g., API argument usage, error handling, etc.

Two statistical comparison methods

- For range data → Histogram-based comparison
  - Compare range data and find deviant sub-ranges
- For occurrences → Entropy-based comparison
  - Find deviation in event occurrences

Histogram-based comparison

1. Represent range data → histogram (see our paper)

2. Build a representative histogram → average histograms
   - High rank frequently used common patterns (e.g., VFS)
   - Low rank specific implementations of file systems

3. Measure distance between histograms
   - Sum up the sizes of non-overlapping area

Example: Path condition checker

foo

```
int foo_rename(flag) {
  if (flag == RO)
    return -EACCES;
}
```

bar

```
int bar_rename(flag) {
  if (flag == RO)
    return -EACCES;
}
```

cad

```
int cad_rename(flag) {
  if (flag == WO)
    return -EACCES;
}1
```

Let's compare ***_rename()** on **-EACCES** path

Example: Path condition checker

foo

```
int foo_rename(flag) {
  if (flag == RO)
    return -EACCES;
}
```

bar

```
int bar_rename(flag) {
  if (flag == RO)
    return -EACCES;
}
```

cad

```
int cad_rename(flag) {
  if (flag == WO)
    return -EACCES;
}1
```

Let's compare ***_rename()** on **-EACCES** path

Represent range data → histogram

foo

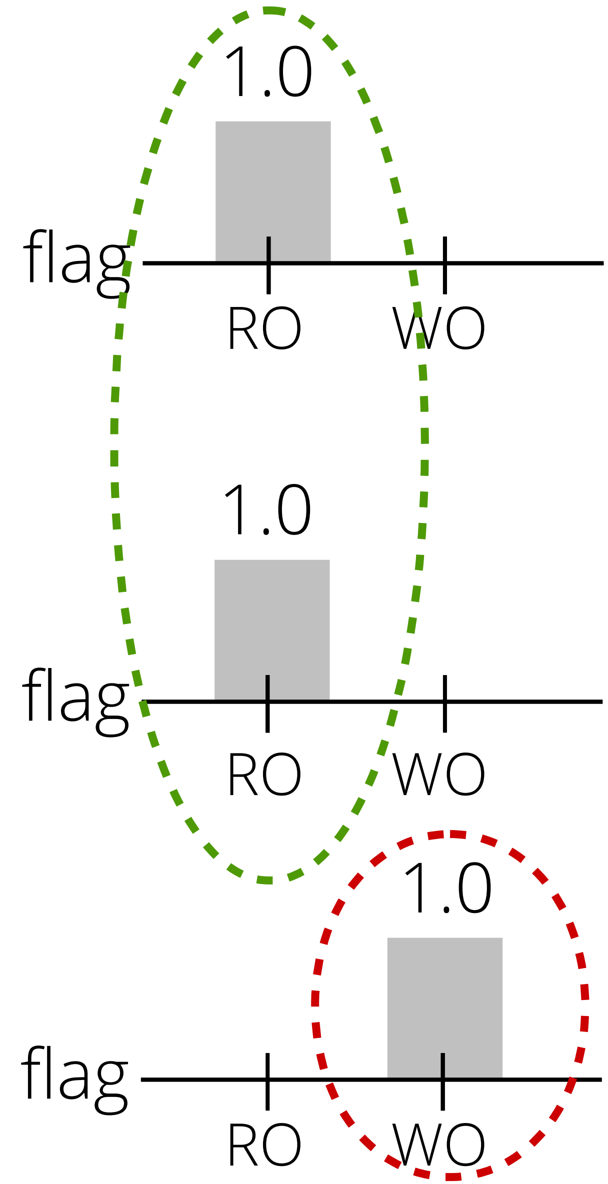
```
int foo_rename(flag) {
  if (flag == RO)
    return -EACCES;
}
```

bar

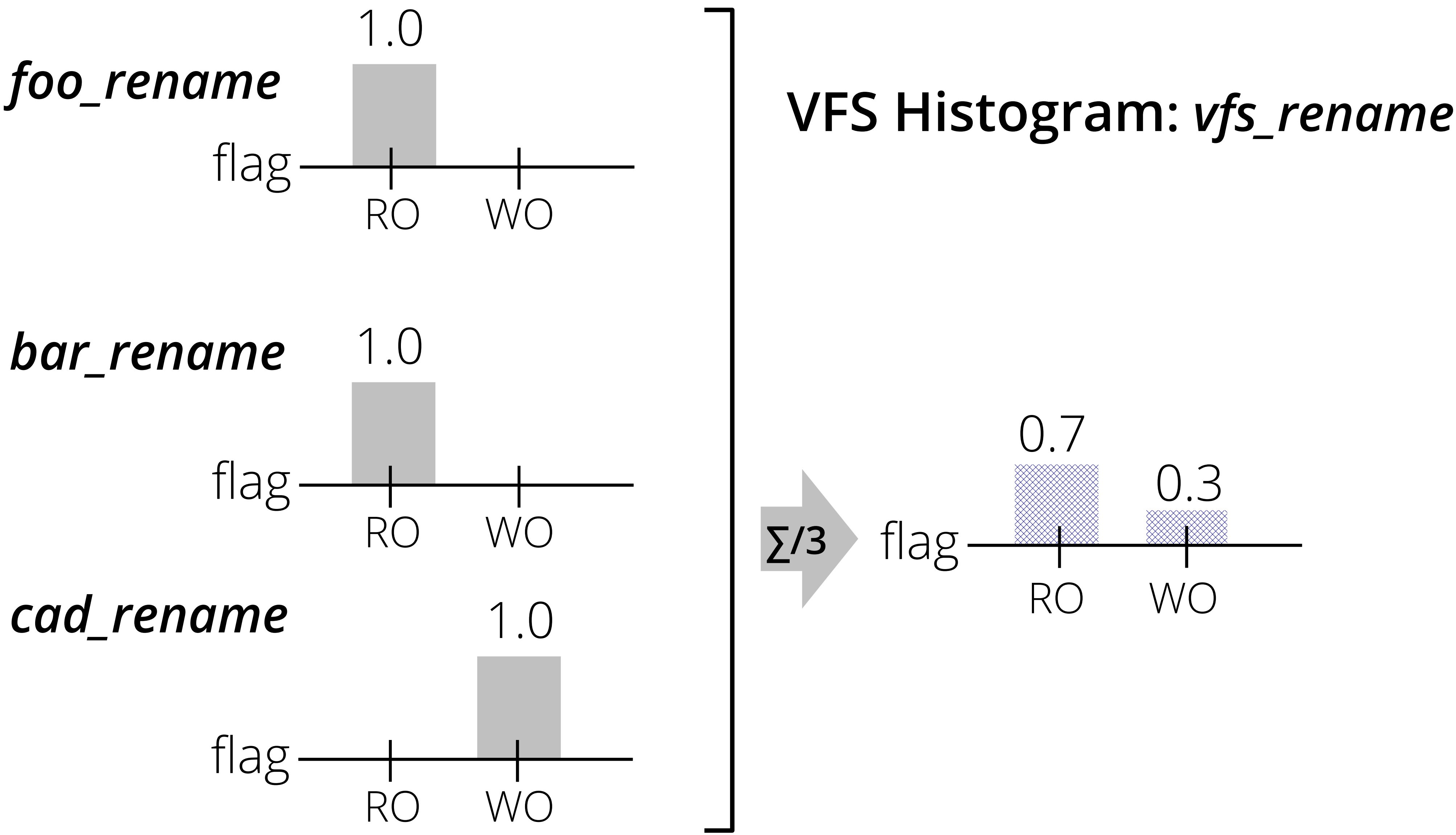
```
int bar_rename(flag) {
  if (flag == RO)
    return -EACCES;
}
```

cad

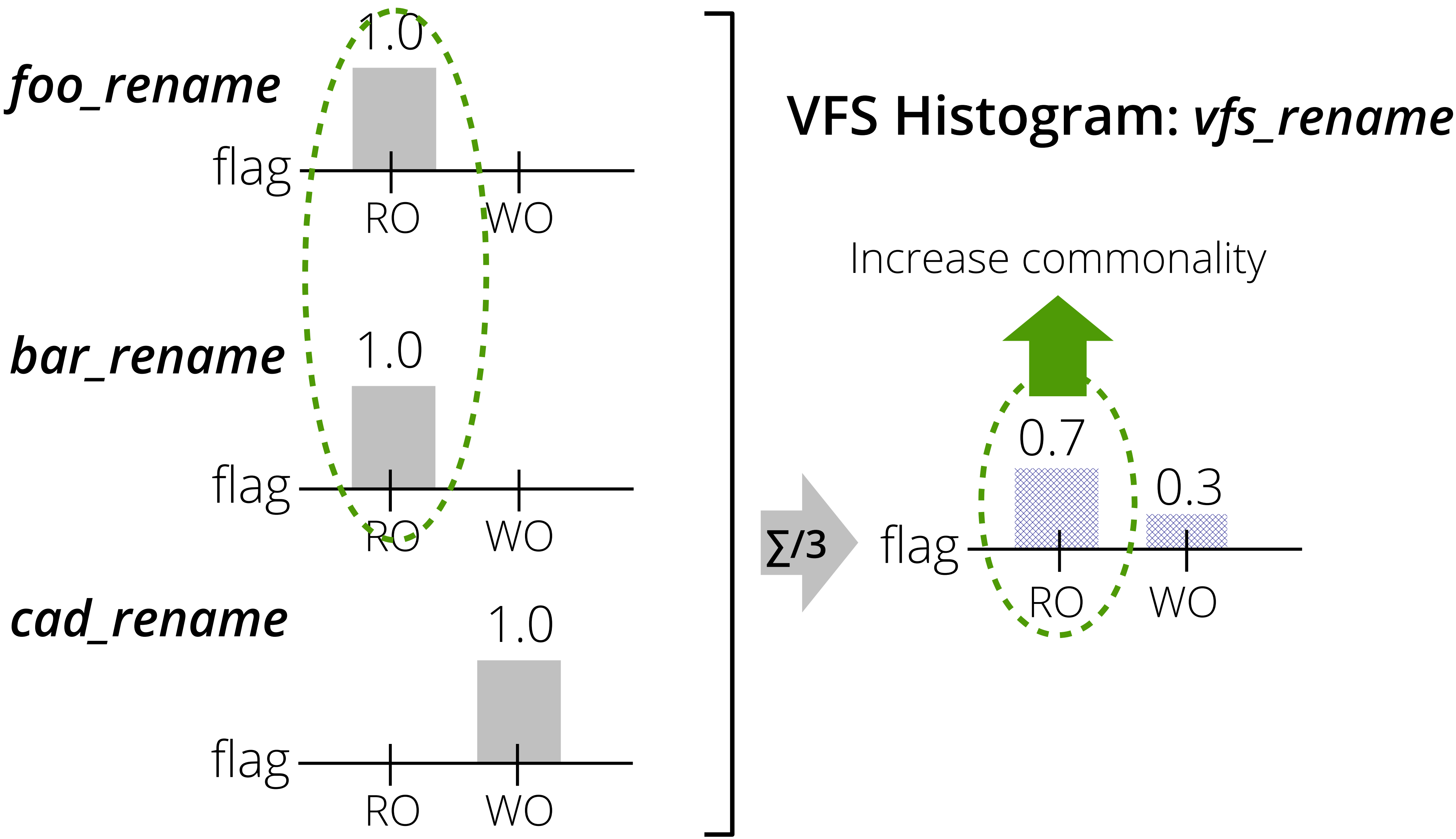
```
int cad_rename(flag) {
  if (flag == WO)
    return -EACCES;
}
```



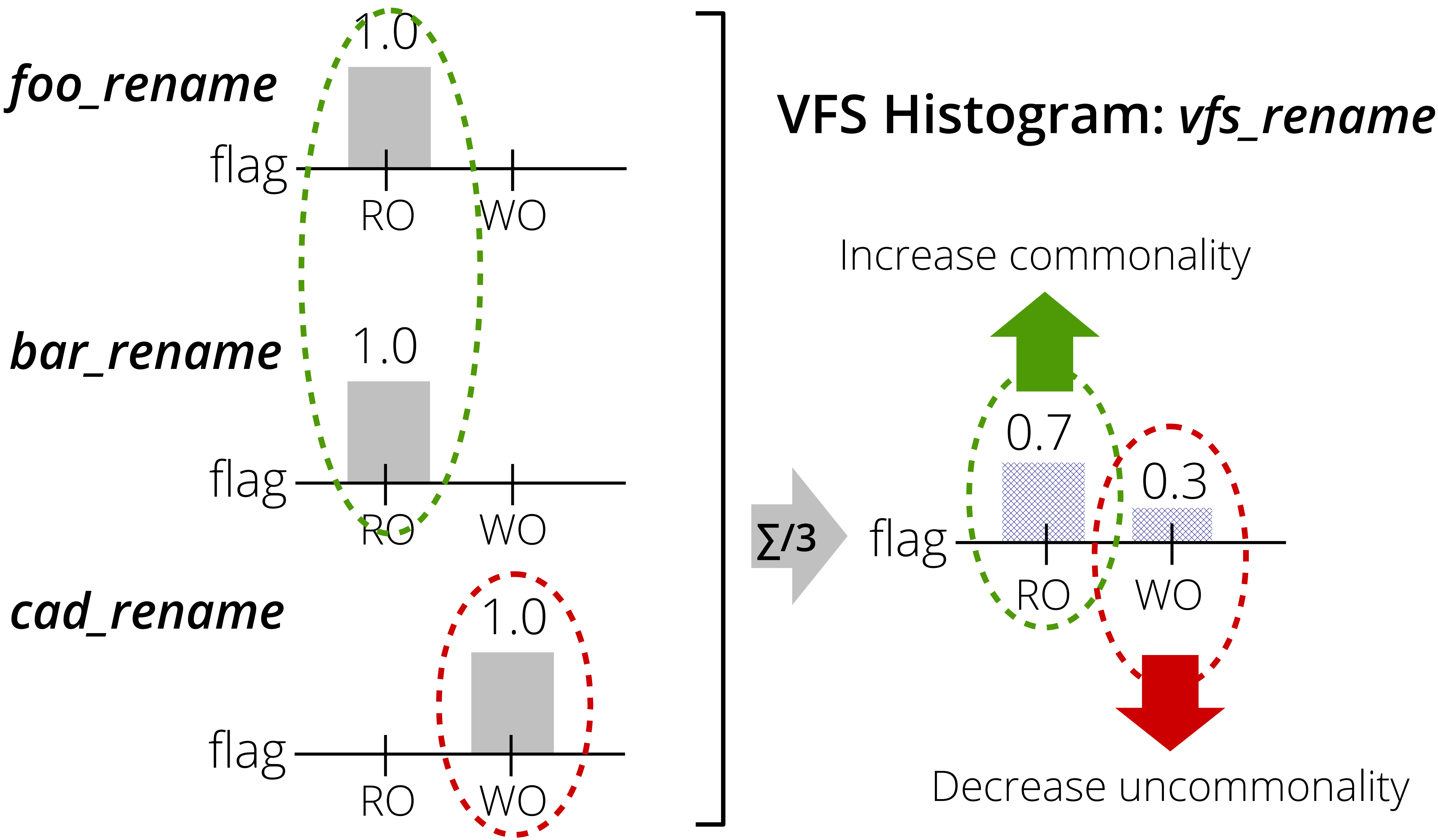
Build a representative histogram



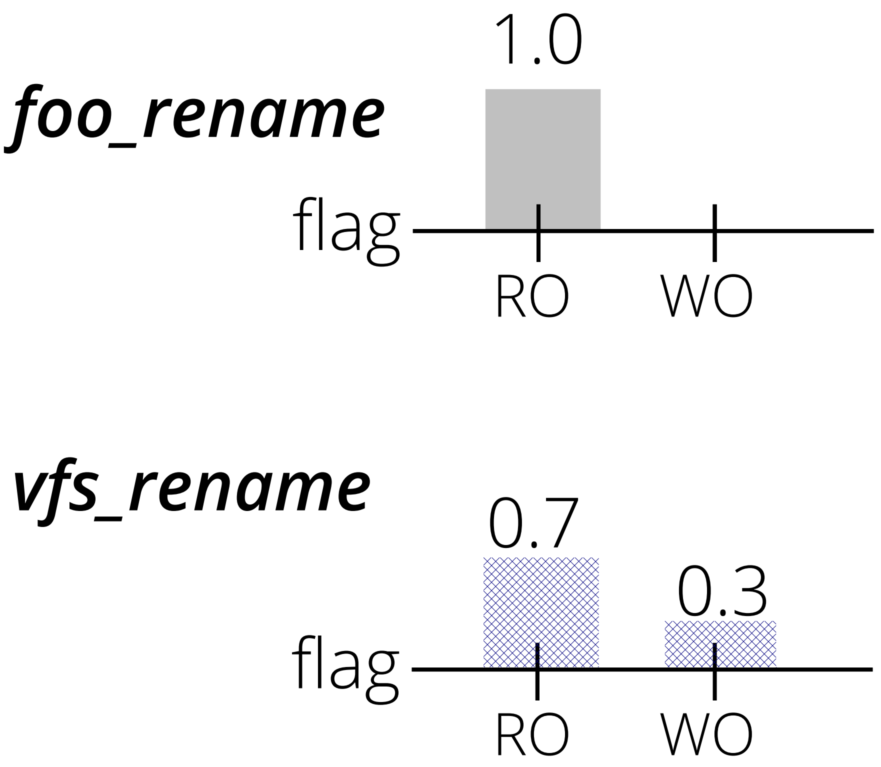
Build a representative histogram



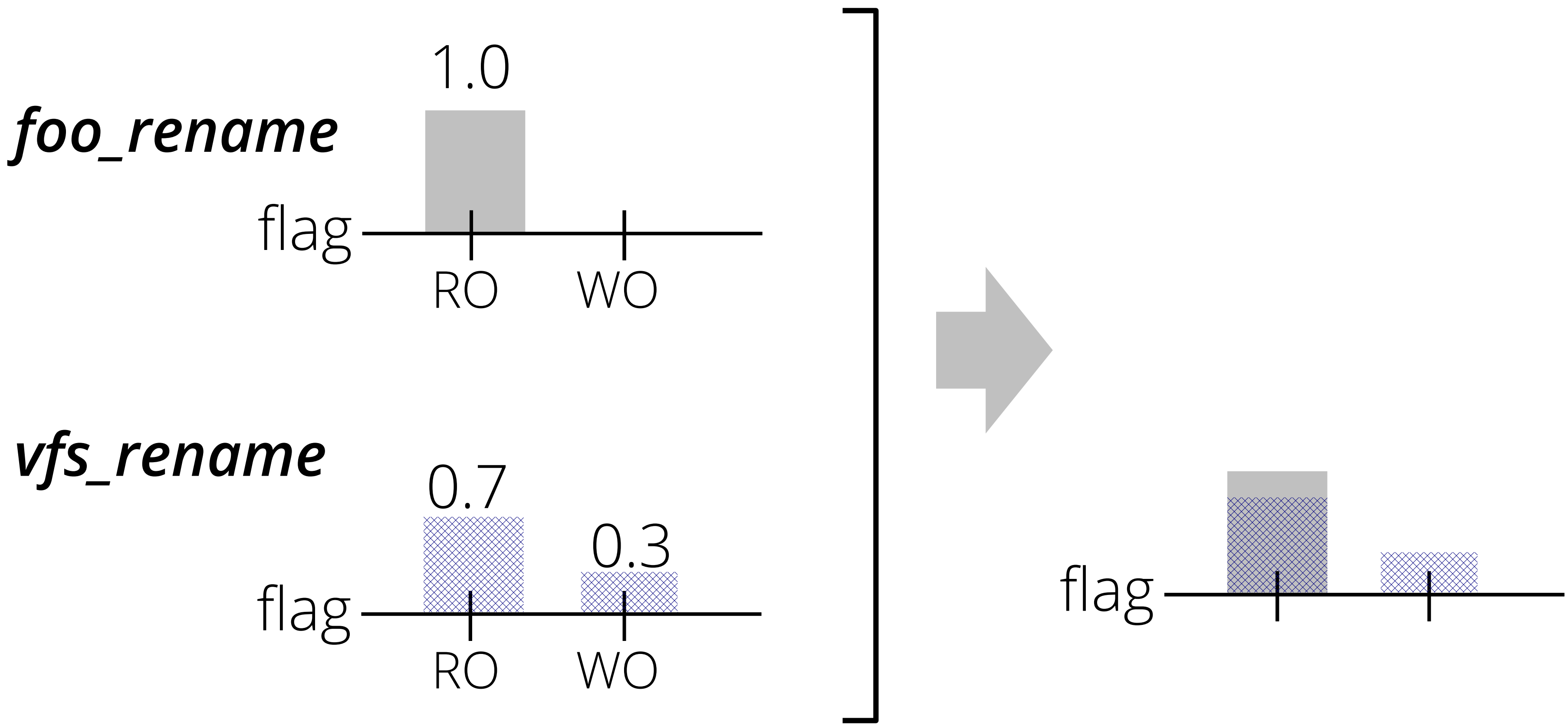
Build a representative histogram



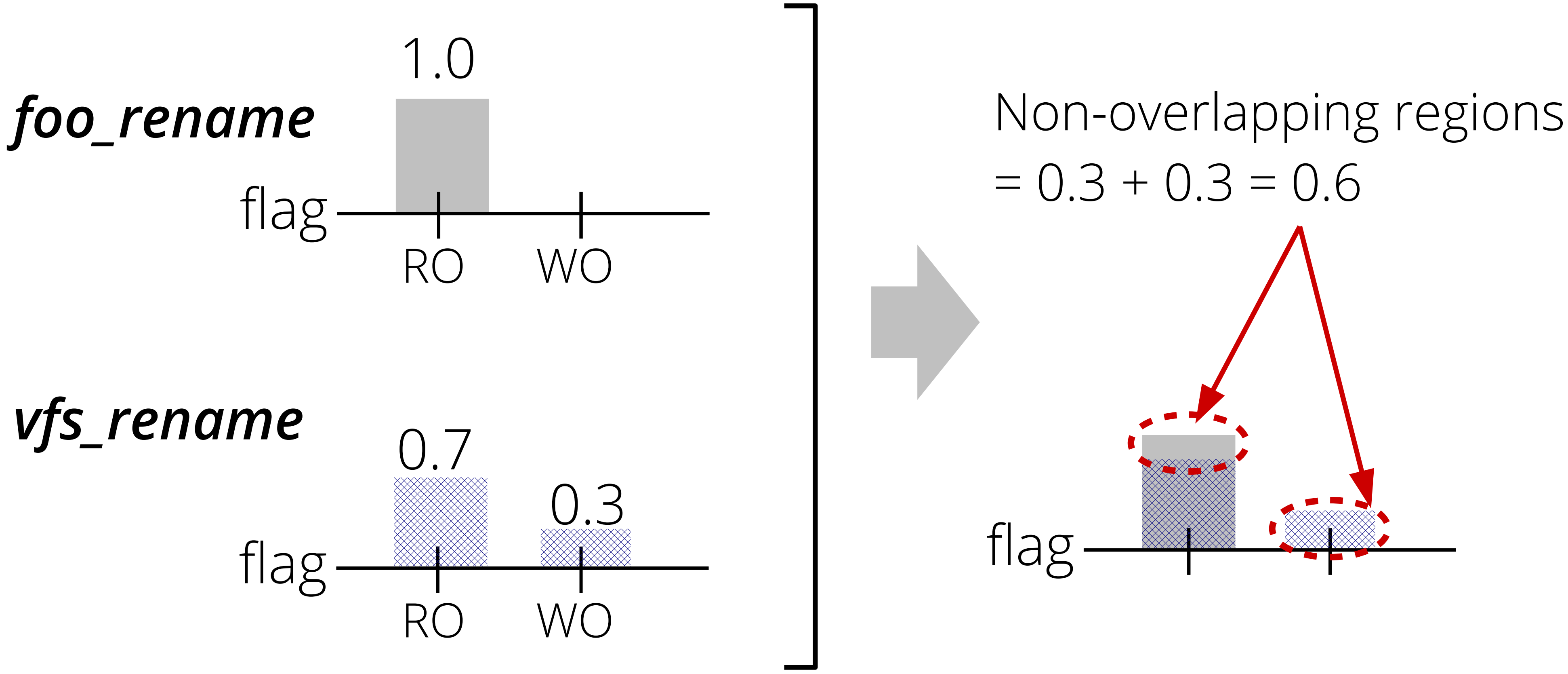
Measure distance between histograms



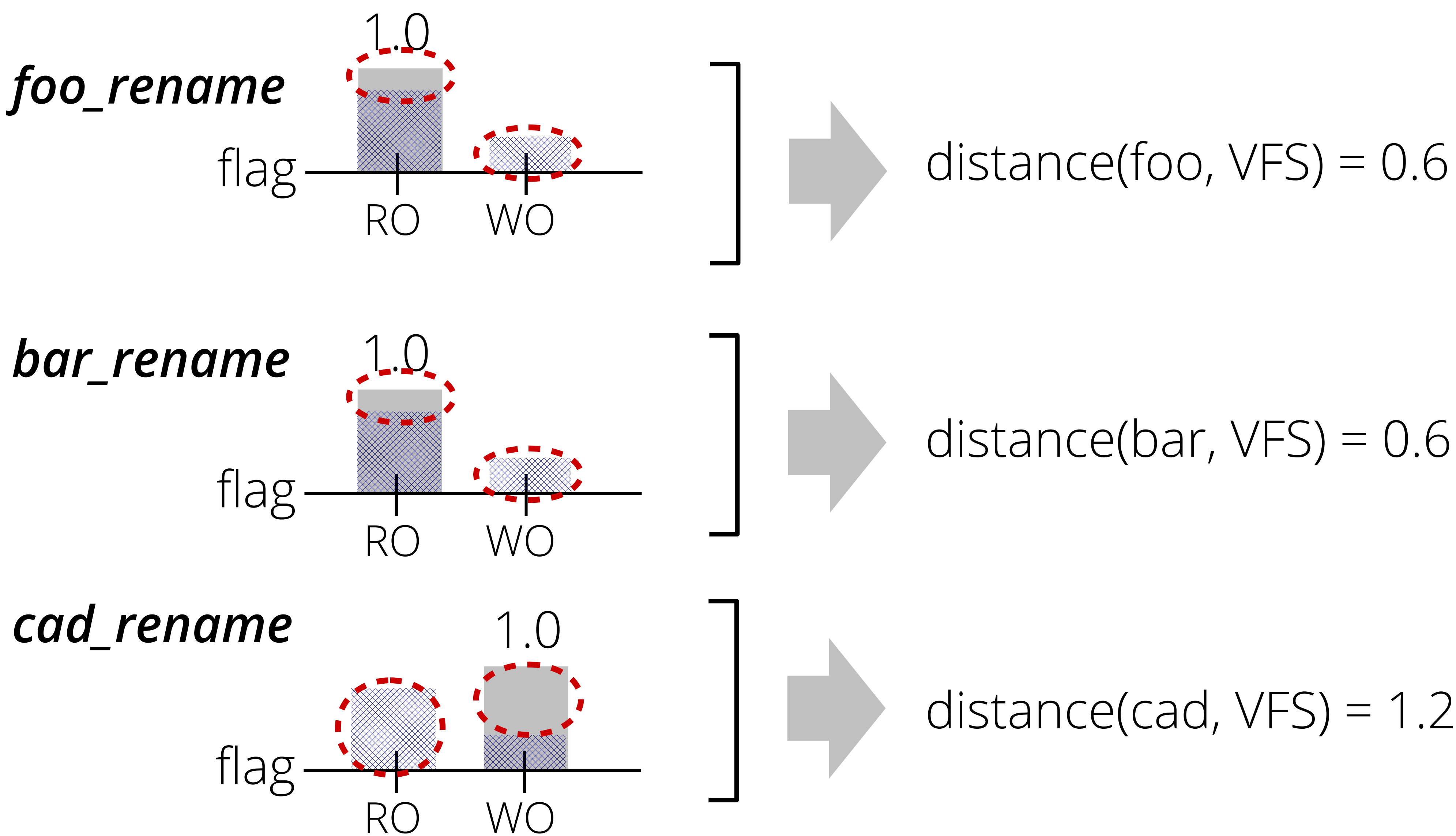
Measure distance between histograms



Measure distance between histograms



Histogram distance



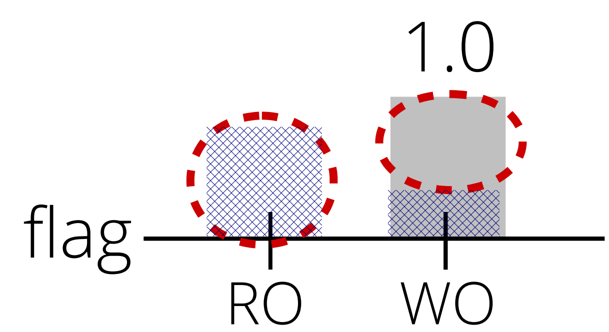
Ranking based on distance

		Distance	Reason

cad	`int foo_rename(flag) {`
	`  if (flag == RO)`
	`    return -EACCES;`
	`}`
		1.2	

foo	`int bar_rename(flag) {`
	`  if (flag == RO)`
	`    return -EACCES;`
	`}`
		0.6	Missing check: flag == RO

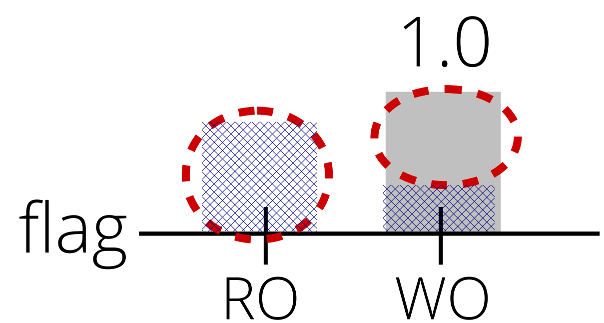
bar	`int cad_rename(flag) {`
	`  if (flag == WO)`
	`    return -EACCES;`
	`}1`
		0.6	



Missing check: flag == RO

Ranking based on distance

		Distance	Reason

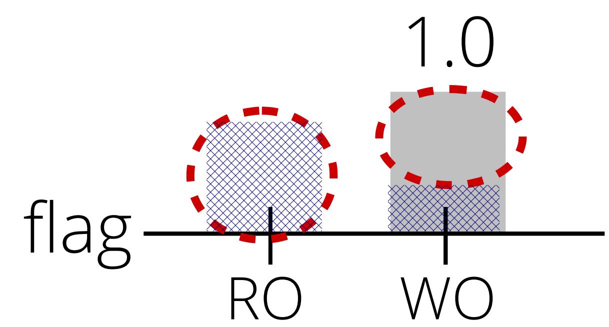
cad	`int foo_rename(flag) {`
	`  if (flag == RO)`
	`    return -EACCES;`
	`}`
		1.2	

foo	`int bar_rename(flag) {`	***0.6***	Missing check: flag == RO

bar	`int cad_rename(flag) {`
	`  if (flag == WO)`
	`    return -EACCES;`
	`}1`
		0.6	

Larger distance → more deviant

Ranking based on distance

		Distance	**Reason**
cad	`int foo_rename(flag) { if (flag == RO) return -EACCES; }`	***1.2***	
foo	`int bar_rename(flag) {`	***0.6***	Missing check: flag == RO

Larger distance → more deviant

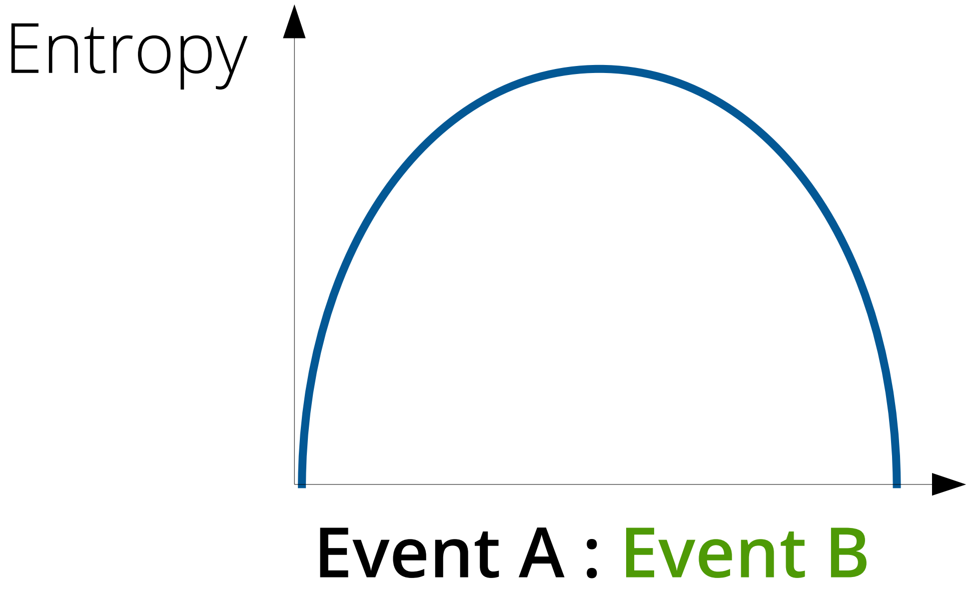
We found 59 new semantic bugs using histogram-based comparison

Two statistical comparison methods

- For range data → Histogram-based comparison
  - Compare range data and find deviant sub-ranges
- For occurrences → Entropy-based comparison
  - Find deviation in event occurrences

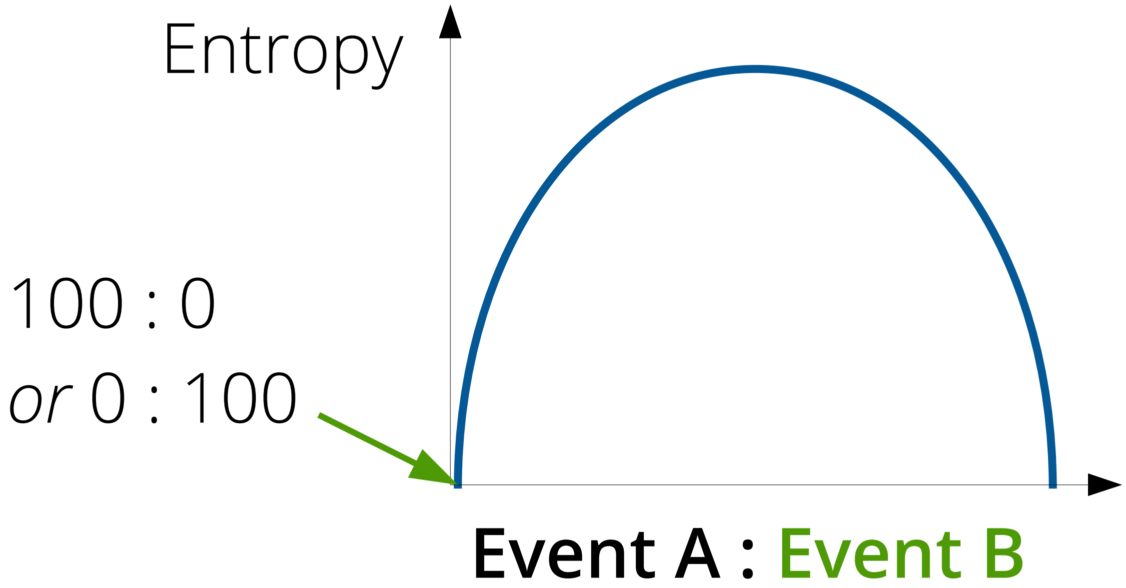
Entropy-based comparison

- Find deviation in event occurrence
  - Function argument, return value handling, etc.
- Shannon Entropy



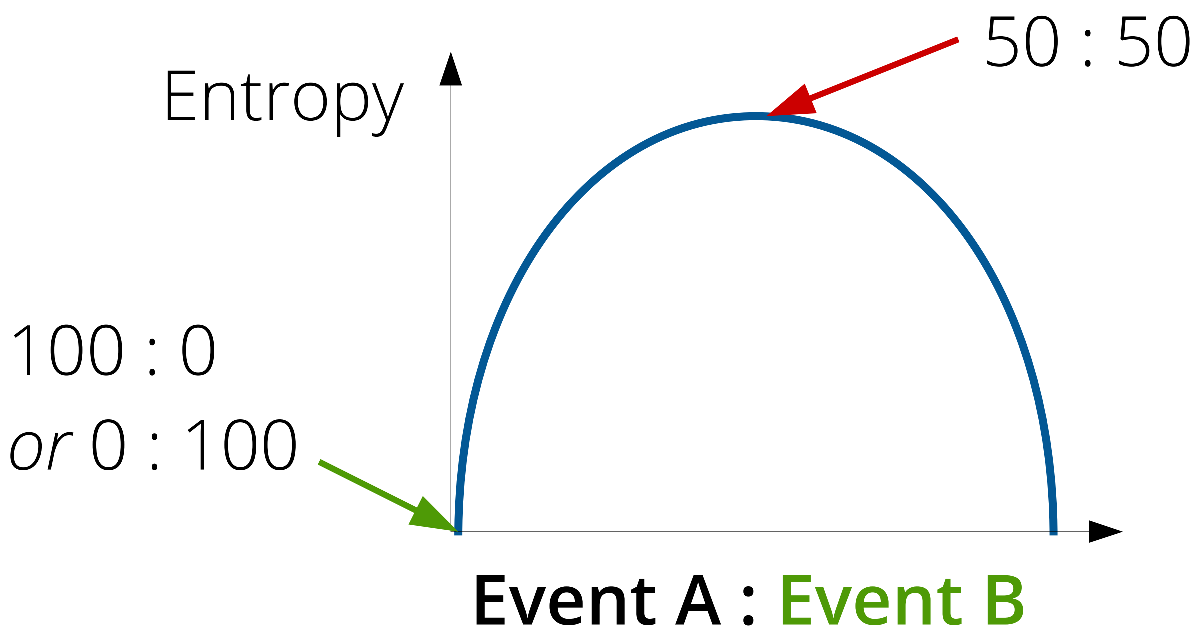
Entropy-based comparison

- Find deviation in event occurrence
  - Function argument, return value handling, etc.
- Shannon Entropy



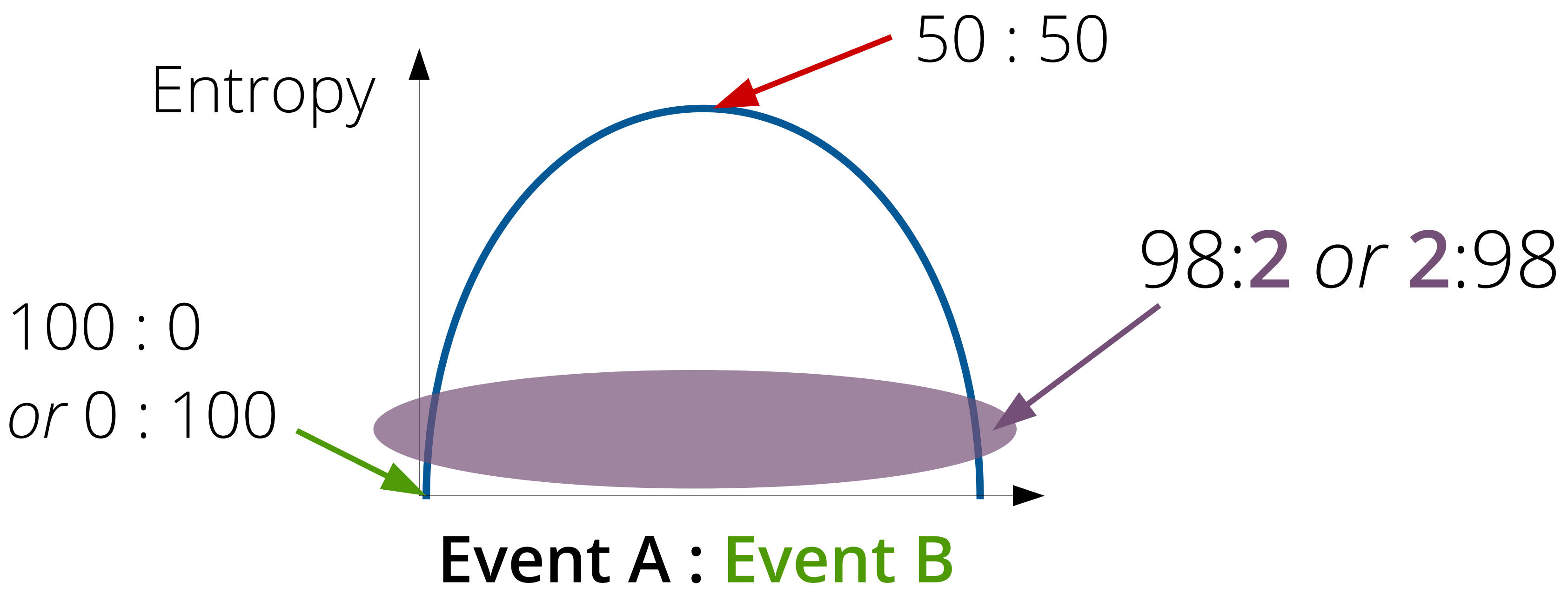
Entropy-based comparison

- Find deviation in event occurrence
  - Function argument, return value handling, etc.
- Shannon Entropy



Entropy-based comparison

- Find deviation in event occurrence
  - Function argument, return value handling, etc.
- Shannon Entropy



Example: Argument checker

- Inferring API usage patterns
  - e.g., **kmalloc()** in file system
    → **GFP_NOFS** to avoid deadlock

- Without any special knowledge, the argument checker can statically identify incorrect uses of API flags in file systems

Calculating entropy of GFP flag usages in file systems

VFS entry	GFP_KERNEL	GFP_NOFS	Entropy
inode→set_acl()	**60**	40	**0.97**
file→read()	**40**	60	**0.97**
file→write()	**2**	98	**0.14**

Calculating entropy of GFP flag usages in file systems

VFS entry	GFP_KERNEL	GFP_NOFS	Entropy
inode→set_acl()	**60**	40	**0.97**
file→read()	**40**	60	**0.97**
file→write()	**2**	98	**0.14**

Ranking based on entropy

VFS entry	GFP_KERNEL	GFP_NOFS	Entropy
file→write()	**2**	**98**	**0.14**
inode→set_acl()	**60**	**40**	**0.97**
file→read()	**40**	**60**	**0.97**

Ranking based on entropy

VFS entry	GFP_KERNEL	GFP_NOFS	Entropy
file→write()	**2**	98	**0.14**
file→read()	**40**	60	**0.97**

Smaller entropy → more deviant

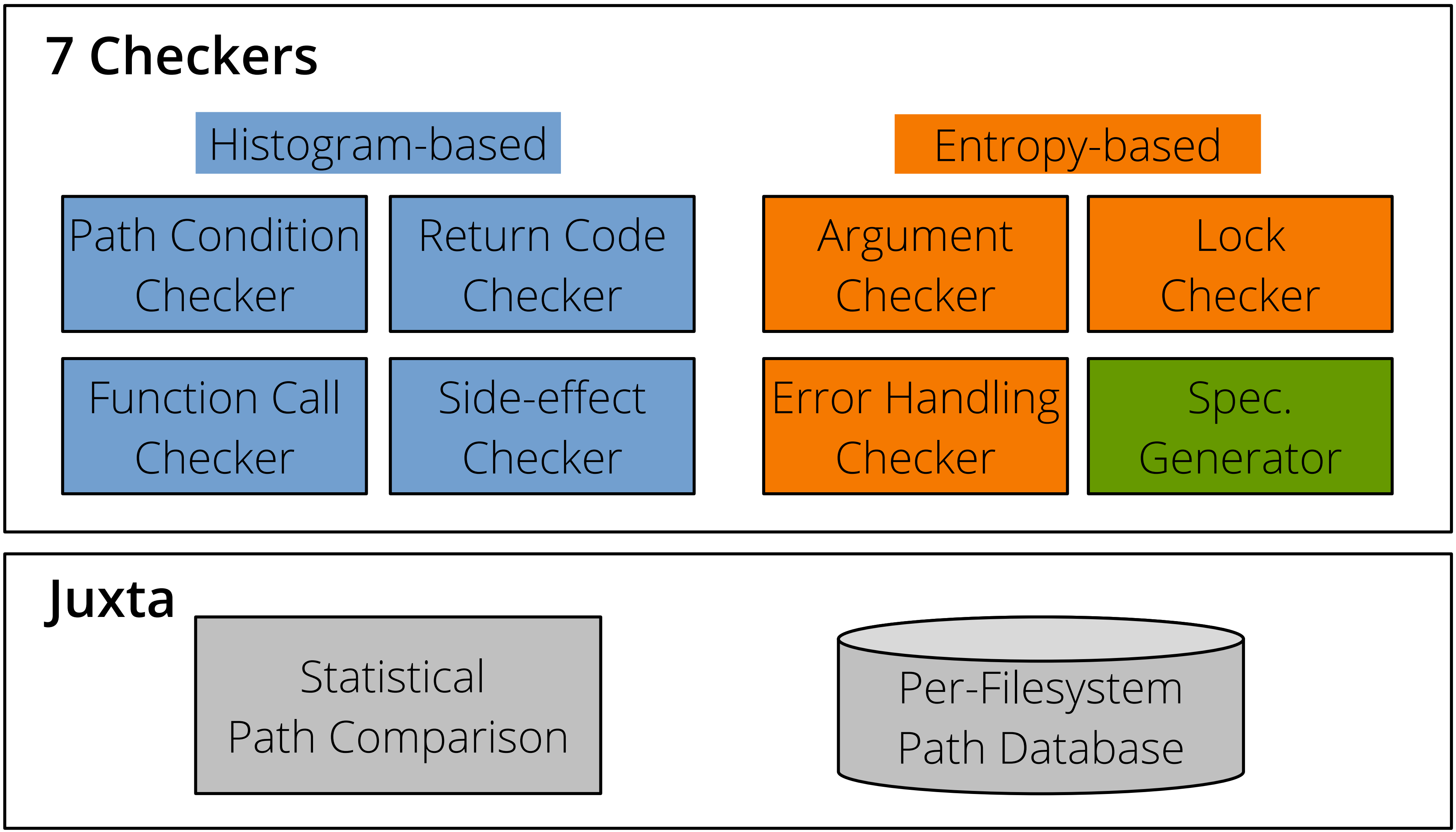
Ranking based on entropy

VFS entry	GFP_KERNEL	GFP_NOFS	Entropy
file→write()	**2**	**98**	**0.14**

Smaller entropy → more deviant

We found 59 new semantic bugs using entropy-based comparison

Specialized Checkers for Specific Types of Semantic Bugs



Implementation of **Juxta**

- 12K LoC in total
  - Symbolic path explorer → 6K lines of C/C++ (Clang 3.6)
  - Tools and library → 3K lines of Python
  - Checkers → 3K lines of Python

- VFS entry database → Linux kernel 4.0-rc2

Evaluation questions

- How effective is Juxta in finding new bugs?
- What types of semantic bugs can Juxta find?
- How complete is Juxta's approach?
- How effective is Juxta's ranking scheme?

Juxta found 118 bugs in 54 file systems

Checker	# reports	# manually verified reports	New bugs
Return code	573	150	2
Side-effect	389	150	6
Function call	521	100	5
Path condition	470	150	46
Argument	56	10	4
Error handling	242	100	47
Lock	131	50	8
Total	**2,382**	**710**	**118**

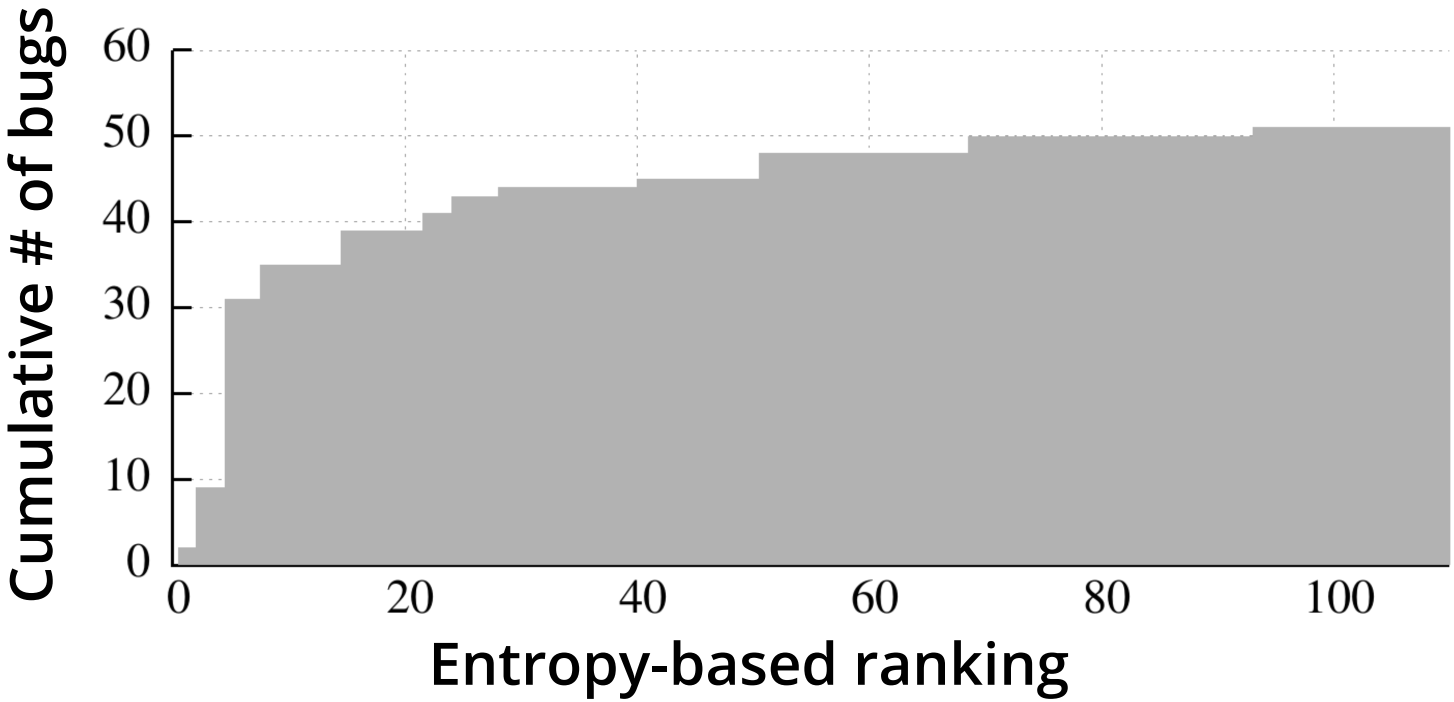
Juxta found 7 types of new semantic bugs

Checker	# reports	# manually verified reports	New bugs
Return code	573	150	2
Side-effect	389	150	6
Function call	521	100	5
Path condition	470	150	46
Argument	56	10	4
Error handling	242	100	47
Lock	131	50	8
Total	**2,382**	**710**	**118**

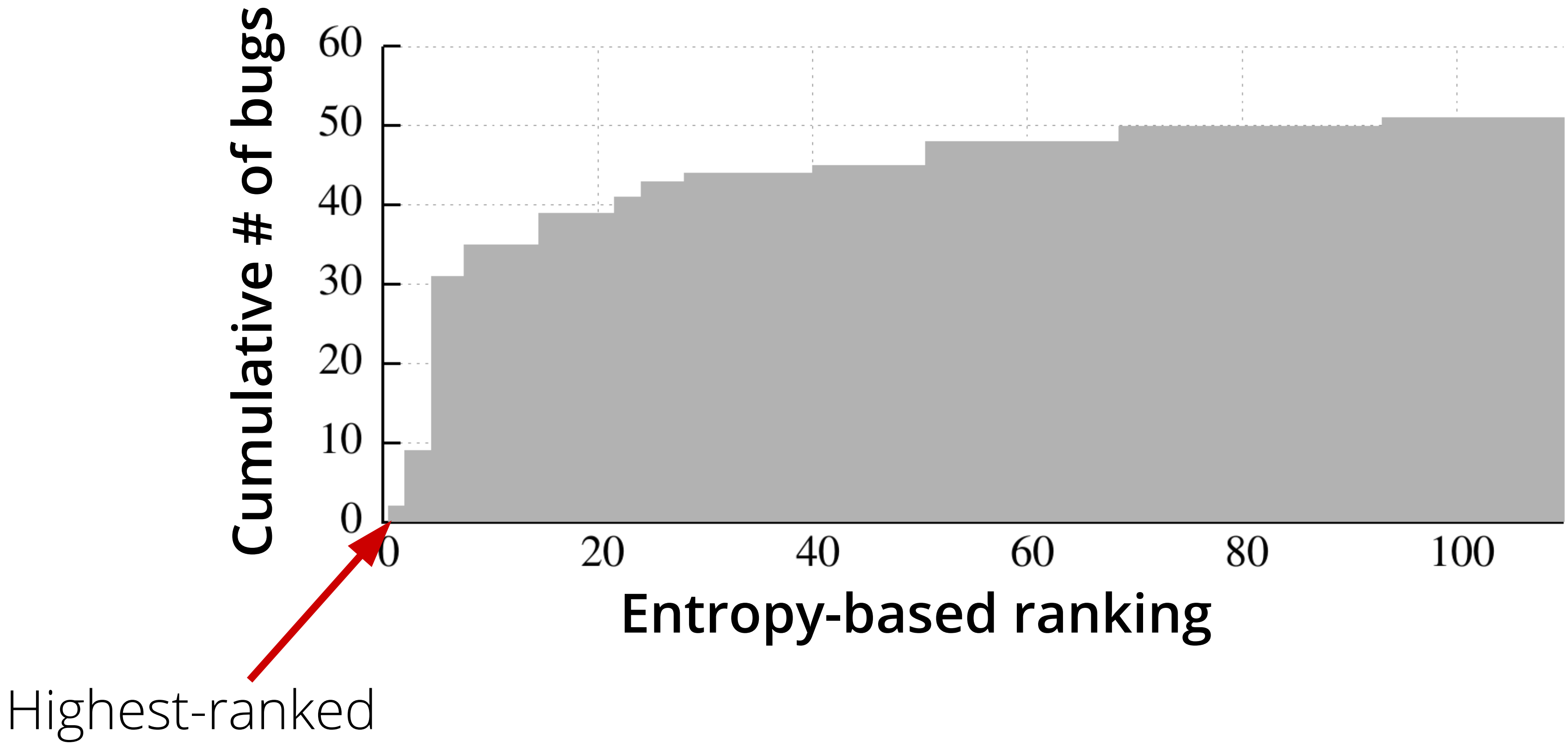
Juxta found most known bugs

- Test case
  - 21 known file system semantic bugs from PatchDB [Lu:FAST12]
  - Synthesize them to the Linux Kernel 4.0-rc2
- Juxta found 19 out of 21 bugs
- 2 missing bugs ← incomplete symbolic execution
  - state explosion
  - limited inter-procedural analysis

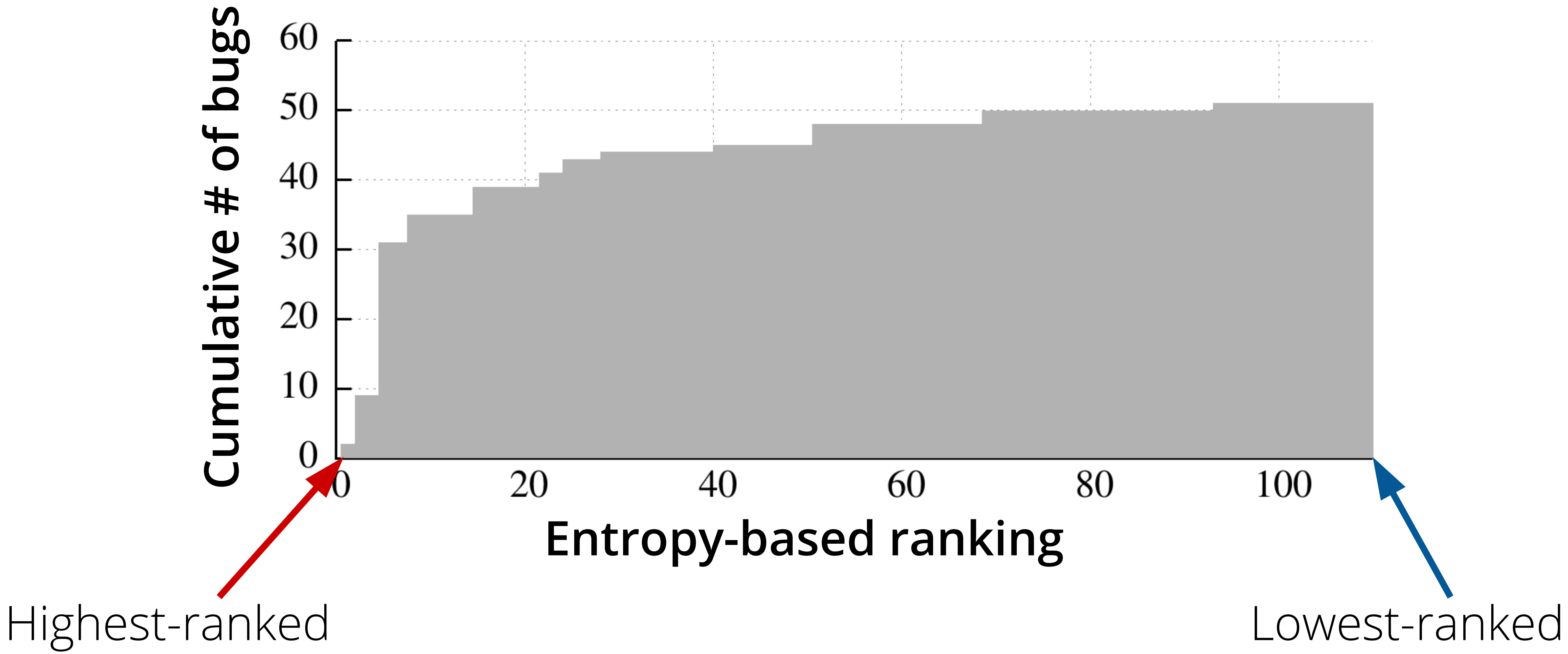
Juxta's ranking scheme is effective



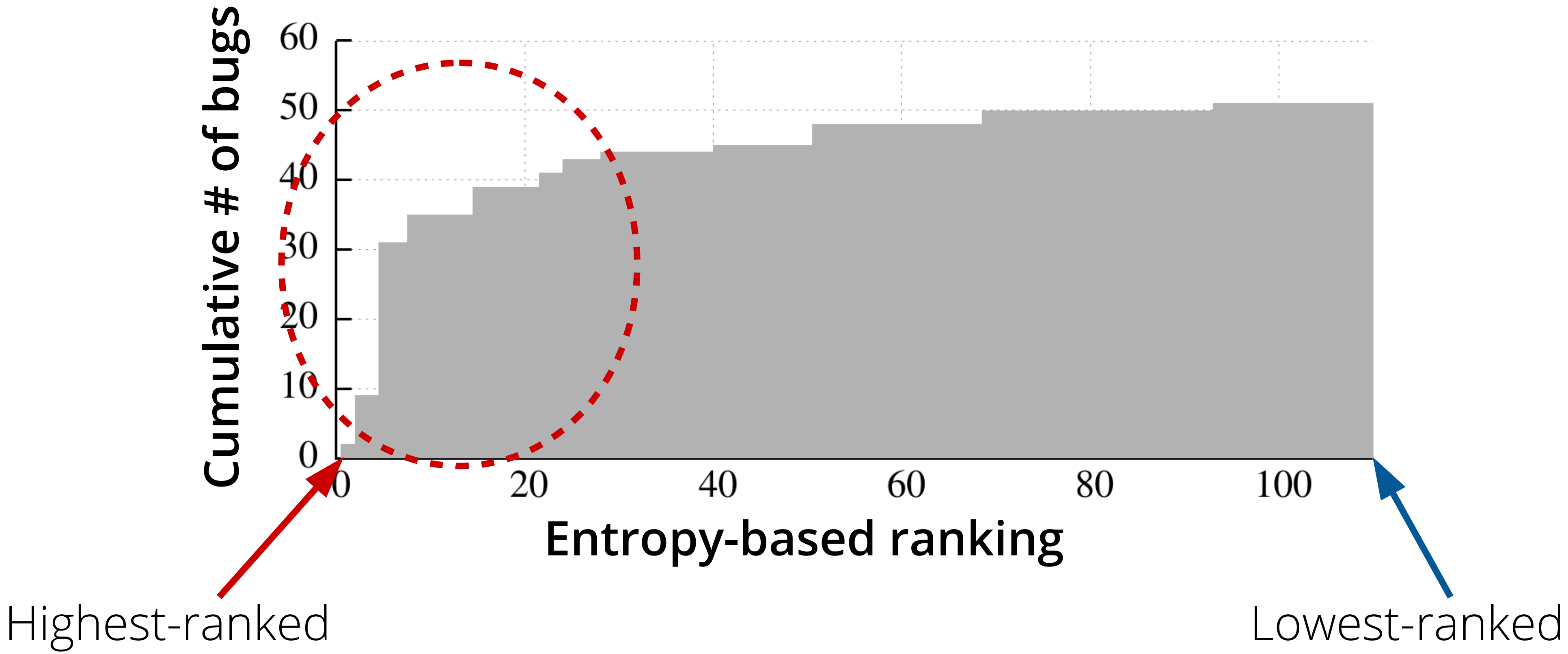
Juxta's ranking scheme is effective



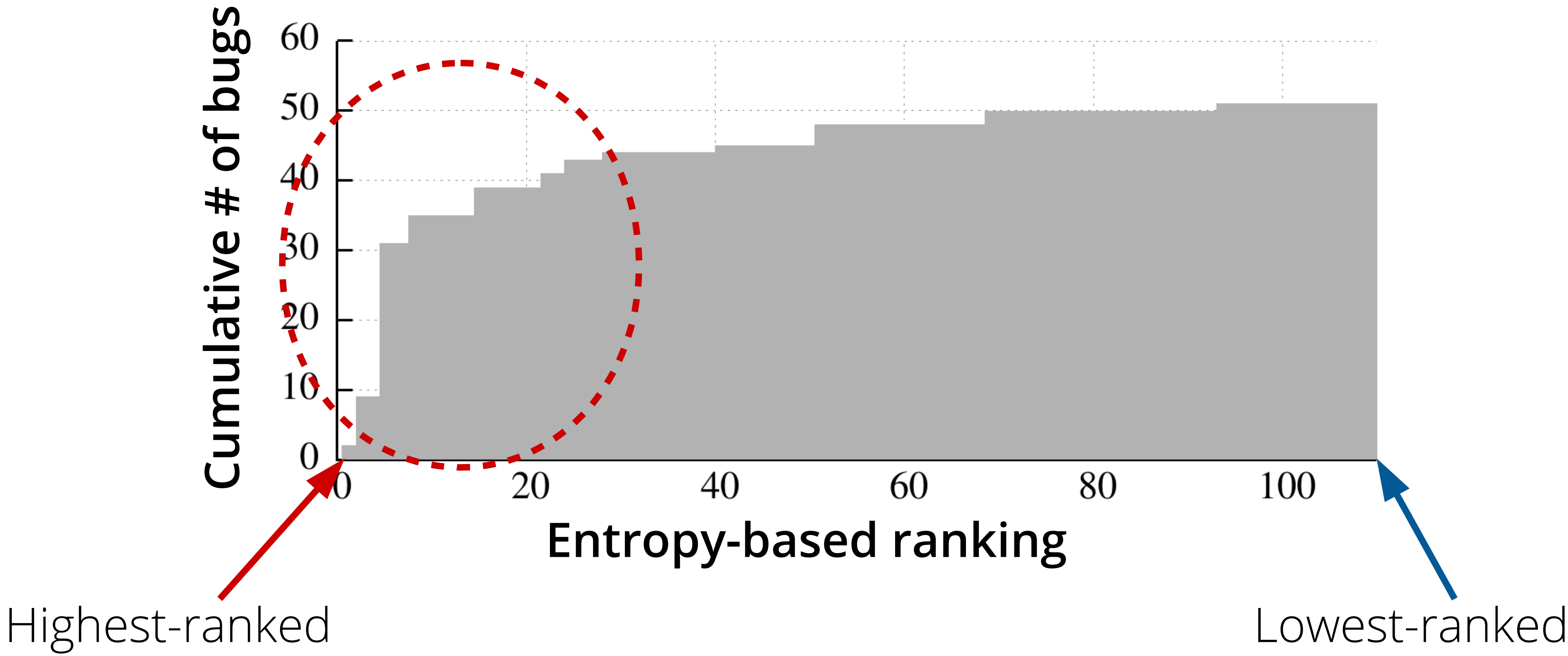
Juxta's ranking scheme is effective



Juxta's ranking scheme is effective



Juxta's ranking scheme is effective



> 50% of real bugs were found in top 100

Limitation

- Deviations do not always mean bugs
  - e.g., 24 patches are rejected after developers' review
- Not universally applicable
  - e.g., requirement: multiple existing implementations
- Symbolic execution is not complete
  - e.g., state explosion, limited inter-procedural analysis

Discussion

- Self-regression
  - e.g., comparing between subsequent versions
- Cross-layer refactoring
  - promoting common code to VFS in Linux file systems
  - e.g., if all file systems need the same capability check, shall we move such check to the VFS?
- Potential programs to be checked
  - e.g., C libs, SCSI device drivers, JavaScript engines, etc.

Conclusion

- Cross-checking semantic correctness by comparing and contrasting multiple implementations
- Juxta: a static tool to find bugs in file systems
  - Seven specialized checkers were developed
  - 118 new semantic bugs found (e.g., ext4, XFS, Ceph, etc.)
- Our code and database will be released soon

Thank you!

Changwoo Min
changwoo@gatech.edu

Sanidhya Kashyap, Byoungyoung Lee,
Chengyu Song, Taesoo Kim

Georgia Institute of Technology
School of Computer Science

Questions?

Case study: Rename a file

- Rename() has complex semantics
  - e.g., rename(old_dir/a, new_dir/b) requires 3x3x3x3 combinations for update (e.g., mtime of dir and file)

- POSIX specification defines subset of such combinations
  - **e.g., ctime** and **mtime** of **old_dir** and **new_dir**

Compare rename() of existing file systems in Linux

- **Majority follows the POSIX spec**
  - **Found 6 incorrect implementation (e.g., HPFS)**
- Found inconsistency of undocumented combinations
  - Found 6 potential bugs (e.g., HFS)

	Attribute	# Updated FS	# Not updated FS
old_dir	ctime	53	1
	mtime	53	1
new_dir	ctime	52	2
	mtime	52	2
file	ctime	48	6

Hidden Spec.

Bugs