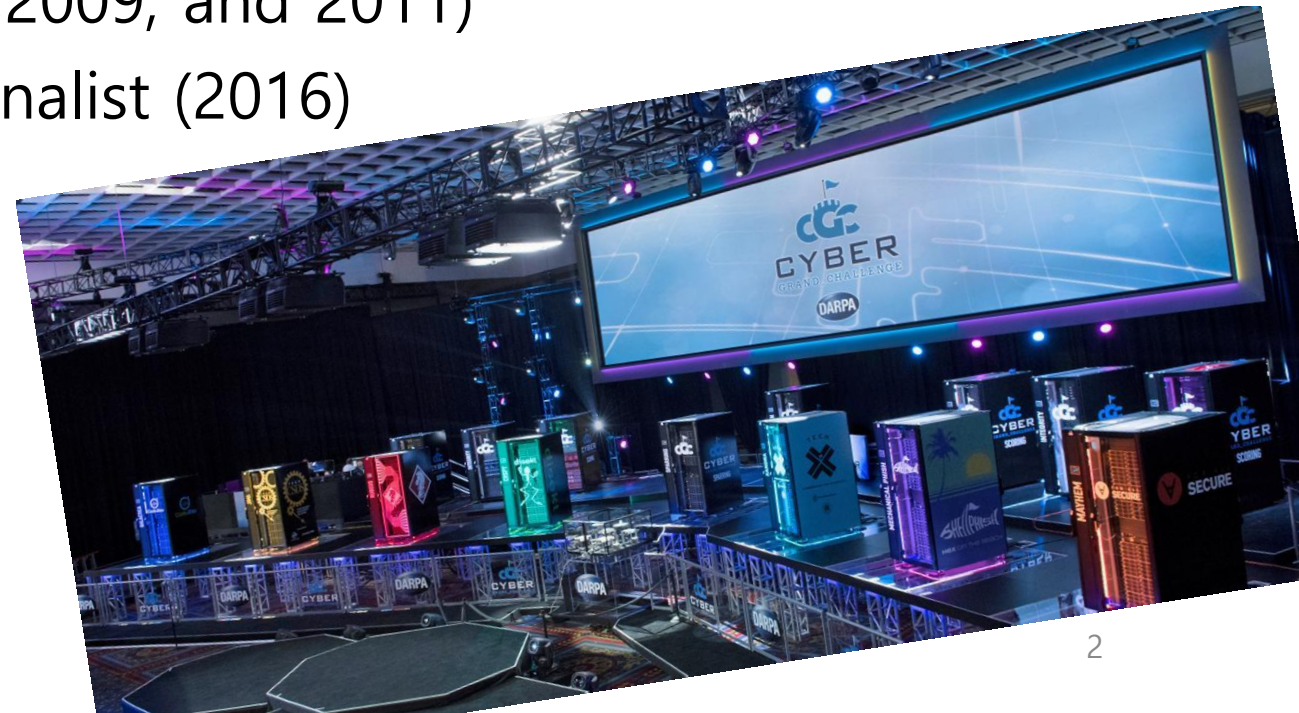


System Security Overview with an Emphasis on Security Issues for Storage and Emerging NVM (Part 1)

Byoungyoung Lee (이병영)
byoungyoung@snu.ac.kr
Seoul National University

Speaker: Byoungyoung Lee (이병영)

- Research areas: Hacking, Systems Security, Software Security
- Found 100++ vulnerabilities from Windows kernel, Linux kernel, Chrome, Firefox, etc.
- Internet Defense Prize by Facebook and USENIX (2015)
- Three times DEFCON CTF Finalist (2007,2009, and 2011)
- DARPA Cyber Grand Challenge (CGC) Finalist (2016)



My Research Areas: Protecting Commodity Systems

Apps



DangNull [NDSS 15]
Expector [WWW 15]
TrackMeOrNot [WWW 16]
MEDS [NDSS 18]



CaVer [USENIX Sec 15]
HexType [CCS 17]



ASLR-Guard [CCS 15]

OS



Juxta [SOSP 15]
Kenali [NDSS 16]
KUP [ATC 16]
Razzer [S&P 19]



Morula [S&P 14]



CAB-Fuzz [ATC 17]



SGX-ASLR [NDSS 17]
Obliviate [NDSS 18]

HW



HDFI [S&P 16]



Minion [NDSS 18]

My Research Areas: Protecting Commodity Systems

Apps



DangNull [NDSS 15]
Expector [WWW 15]
TrackMeOrNot [WWW 16]
MEDS [NDSS 18]



CaVer [USENIX Sec 15]
HexType [CCS 17]

NGINX

ASLR-Guard [CCS 15]

Attack
Mitigation

OS



Juxta [SOSP 15]
Kenali [NDSS 16]
KUP [ATC 16]
Razzer [S&P 19]



Morula [S&P 14]



CAB-Fuzz [ATC 17]

Intel® SGX

SGX-ASLR [NDSS 17]
Obliviate [NDSS 18]

HW



HDFI [S&P 16]



Minion [NDSS 18]

My Research Areas: Protecting Commodity Systems

Apps



DangNull [NDSS 15]
Expector [WWW 15]
TrackMeOrNot [WWW 16]
MEDS [NDSS 18]



CaVer [USENIX Sec 15]
HexType [CCS 17]

NGINX

ASLR-Guard [CCS 15]

Attack
Mitigation

OS



Juxta [SOSP 15]
Kenali [NDSS 16]
KUP [ATC 16]
Razzer [S&P 19]



Morula [S&P 14]



CAB-Fuzz [ATC 17]

Intel® SGX

SGX-ASLR [NDSS 17]
Obliviate [NDSS 18]

Vulnerability
Finding

HW



HDFI [S&P 16]



Minion [NDSS 18]

My Research Areas: Protecting Commodity Systems

Apps



DangNull [NDSS 15]
Expector [WWW 15]
TrackMeOrNot [WWW 16]
MEDS [NDSS 18]



CaVer [USENIX Sec 15]
HexType [CCS 17]

NGINX

ASLR-Guard [CCS 15]

**Attack
Mitigation**

OS



Juxta [SOSP 15]
Kenali [NDSS 16]
KUP [ATC 16]
Razzer [S&P 19]



Morula [S&P 14]



CAB-Fuzz [ATC 17]

Intel® SGX

SGX-ASLR [NDSS 17]
Obliviate [NDSS 18]

**Vulnerability
Finding**

HW



HDFI [S&P 16]



Minion [NDSS 18]

**Secure
Trusted
Computing**

Outline

Part1. Bugs in File Systems

Semantic inconsistency inference
Fuzzing

Part2. Attacks and Defenses

Ransomware
Cold boot attacks
Side-channels

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

Linux Kernel Plagued by an EXT4 Data Corruption Issue, Patch Available

Overview Code **Bugs** Blueprint It is related to RAID setups only and a patch exists

Risk of filesystem

May 20, 2015 02:37 GMT · By Marius Nestor · Share: [Twitter](#) [Reddit](#) [Facebook](#) [G+](#) [F](#)

Bug #1097042 reported by [lemonsqueeze](#)

Apparently, there's a data corruption issue in the EXT4 file system for multiple Linux kernel branches, affecting several mainstream distributions, including Arch Linux, Fedora, and Debian GNU/Linux.

This bug affects 1 person

Affects	Status
linux (Ubuntu)	Expired

Several [Arch Linux](#) users have already [reported data loss](#) after upgrading to Linux kernel 4.0.2, the latest version available in the main software repositories of the respective operating system, which had already been flagged as out of date since May 17, 2015.

[+ Also affects project](#) [?](#) [+ Also af](#)

Bug Description

```
On my system, a default ext4
$ cat /proc/mounts
/dev/sda6 /media/space ext3
user_xattr,acl,data=ordered
```

We can see the "barrier=1" ,
severe filesystem corruption
recently). Quoting mount(1)

However, the EXT4 data corruption issue has not been fixed in Linux kernel 4.0.3, nor Linux kernel 4.0.4, according to an [Arch Linux](#) user who reported problems with a RAID (redundant array of independent disks) configuration a couple of days ago on the official forums of the distribution.

The issue is also [confirmed by a Phoronix reader](#), who says, "several people were affected by an ext4 data corruption bug in Linux 4.0.2. The bug is reported to be unpatched even in the most recent stable version, Linux 4.0.4. We are not sure what exactly triggers the problem, using a RAID setup seems to have something to do with it. It is reported to affect multiple distributions, including Arch, [Debian](#) and [Fedora](#), so it seems to be an upstream problem."

The EXT4 data corruption issue has been confirmed by Debian users too

File system bugs are critical



Ubuntu
linux package

Linux Kernel Plagued by an EXT4 Issue, Patch Available

Overview Code **Bugs** Blueprint It is related to RAID setups only and a patch exists

Risk of filesystem

May 20, 2015 02:37 GMT · By Marius Nestor · Share: [Twitter](#) [Reddit](#) [Facebook](#)

Bug #1097042 reported by [lemonsqueeze](#)

This bug affects 1 person

Affects	Status
linux (Ubuntu)	Expired

[+ Also affects project](#) [?](#) [+ Also af](#)

Bug Description

```
On my system, a default ext4
$ cat /proc/mounts
/dev/sda6 /media/space ext4
user_xattr,acl,data=ordered

We can see the "barrier=1"
severe filesystem corruption
recently). Quoting mount(1)
```

Apparently, there's a data corruption issue in the EXT4 multiple Linux kernel branches, affecting several including Arch Linux, Fedora, and Debian GNU/Linux.

Several Arch Linux users have already reported data loss after upgrading to Linux 4.0.2, the latest version available in the main software repository. The issue was reported on the Arch Linux forums, which had already been flagged as out of date.

However, the EXT4 data corruption issue has not been fixed in kernel 4.0.4, according to an Arch Linux user who reported the problem on the Arch Linux forums of the distribution.

The issue is also confirmed by a Phoronix reader, who says, "I've been using Linux 4.0.2 for a while now. The bug is reported by an ext4 data corruption bug in Linux 4.0.2. The bug is reported by the most recent stable version, Linux 4.0.4. We are not sure if the problem, using a RAID setup seems to have something to do with it. The issue is also confirmed by a Phoronix reader, who says, "I've been using Linux 4.0.2 for a while now. The bug is reported by the most recent stable version, Linux 4.0.4. We are not sure if the problem, using a RAID setup seems to have something to do with it. The issue is also confirmed by a Phoronix reader, who says, "I've been using Linux 4.0.2 for a while now. The bug is reported by the most recent stable version, Linux 4.0.4. We are not sure if the problem, using a RAID setup seems to have something to do with it."

The EXT4 data corruption issue has been confirmed by D

Data Centre ▶ Storage

'Urgent data corruption issue' destroys filesystems in Linux 4.14

Using bcache to speed Linux 4.14? Stop if you want your data to live

By Simon Sharwood 22 Nov 2017 at 08:35

54 [SHARE](#) ▼

A filesystem-eating bug has been found in Linux 4.14.

First reported last week by developer Pavel Goran, the problem struck bcache, a tool that lets one use a solid state disk drive as a read/write cache for another drive. bcache is often used to store data from a slow disk on faster media.

Goran noticed the problem after trying to upgrade Gentoo from version 4.13 of the Linux kernel to version 4.14. During that effort he noticed "reads from the bcache device produce different data in 4.14 and 4.13."

After plenty of analysis, he concluded that "this looks like a very serious bug that can corrupt or completely destroy the data on bcache devices."

Others agreed with Goran, as his report quickly made its way onto the Gentoo bugs list and was later identified as having the following cause:

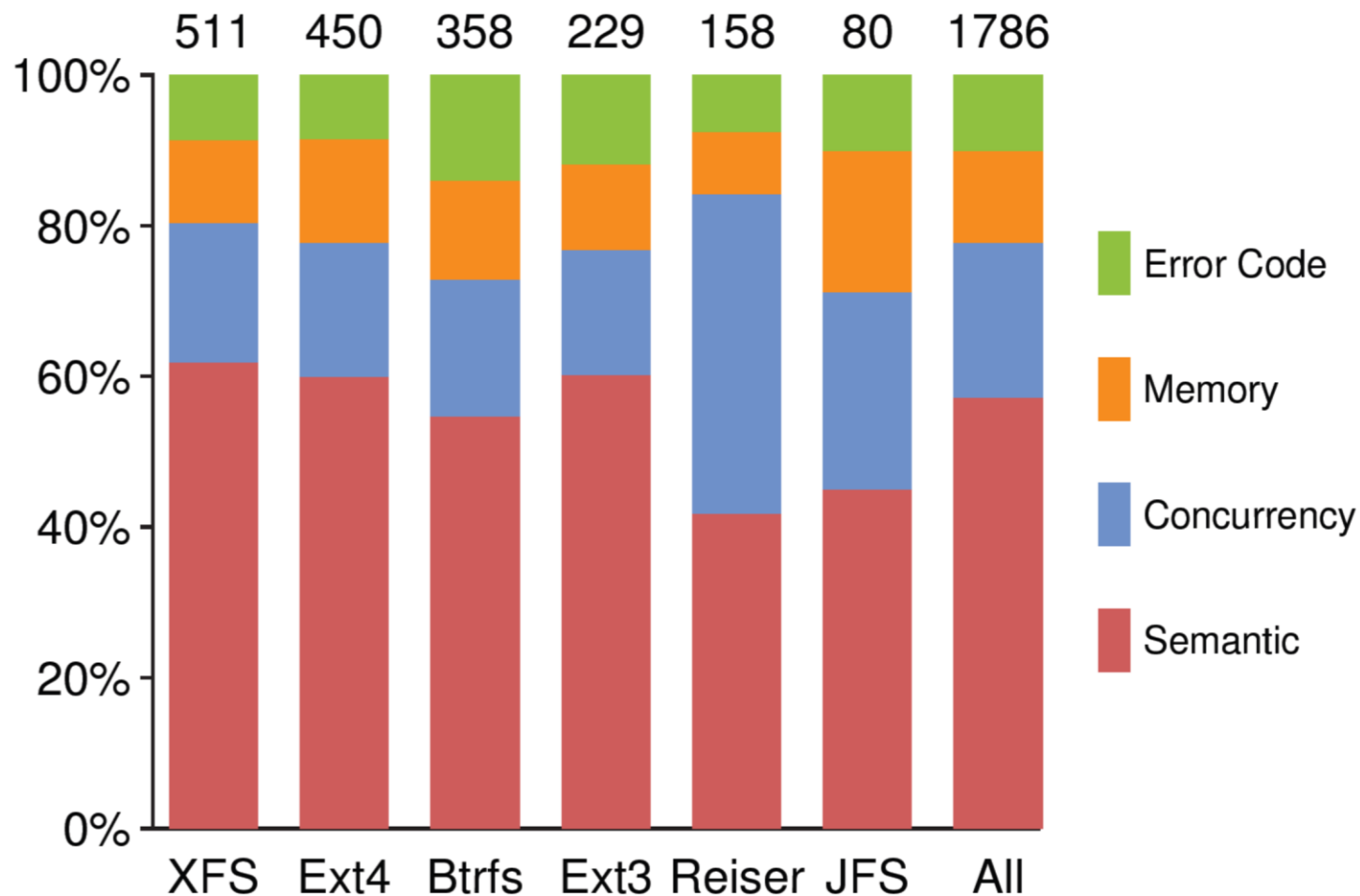
A new field was introduced in 74d46992e0d9, bi_partno, instead of using bdev->bd_contains and encoding the partition information in the bi_bdev field. __bio_clone_fast was changed to copy the disk information, but not the partition information. At minimum, this regressed bcache and caused data corruption.

Bug Patterns in File Systems

Bug pattern	Description
Semantic	<ul style="list-style-type: none">- Incorrect design or implementation- e.g., incorrect state update, wrong design
Concurrency	<ul style="list-style-type: none">- Incorrect concurrent behavior- e.g., miss unlock, deadlock
Memory	<ul style="list-style-type: none">- Incorrect handling of memory objects- e.g., resource leak, null dereference
Error code	<ul style="list-style-type: none">- Missing or wrong error code handling- e.g., return wrong error code

“A study of Linux File System Evolution [FAST 13]”

Bug Patterns in File Systems



“A study of Linux File System Evolution [FAST 13]”

How to stop bleeding from bugs

- **Detection**
 - Static analysis
 - Dynamic analysis
- Prevention
 - Formal proof-assisted development (FSCQ [SOSP15])
 - Use type-safe languages
- Minimizing negative security impacts
 - Isolation
 - A principle of least privileges
 - Data backup

Bug Detection Techniques in General

- Static analysis
 - Analyzing an implementation without running
 - Low false negative: complete coverage can be guaranteed
 - High false positives: difficult to determine if it's truly a bug
- Dynamic analysis
 - Analyzing an implementation while running
 - Low false positive: capable of generating evidence of bugs
 - High false negatives: bug detection capability is limited by its testing coverage

Detection Methods per Bug Pattern

Bug pattern	Detection Methods
Semantic	Model inference from specification Semantic inconsistency inference [PLDI 07, SOSP 15]
Concurrency	Model checking Random thread interleaving: SKI [OSDI 14] Fuzzing: Razzar [S&P 19]
Memory	Static program analysis (Points-to analysis) Symbolic execution: KLEE [OSDI 08] Fuzzing
Error code	Model inference from specification Semantic inconsistency inference

Outline

Part1. Bugs in File Systems

Semantic inconsistency inference

Fuzzing

Part2. Attacks and Defenses

Ransomware

Cold boot attacks

Side-channels

Semantic Bugs

- Semantic bugs
 - Difficult to define violation conditions
 - Most semantic bugs are discovered by manual auditing
- Semantic bugs include
 - Incorrect condition check
 - Incorrect status 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;
```

Can we find this bug
by leveraging
other implementations?

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;
```

Deviant implementations
→ 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;

...

Case Study: Write a page

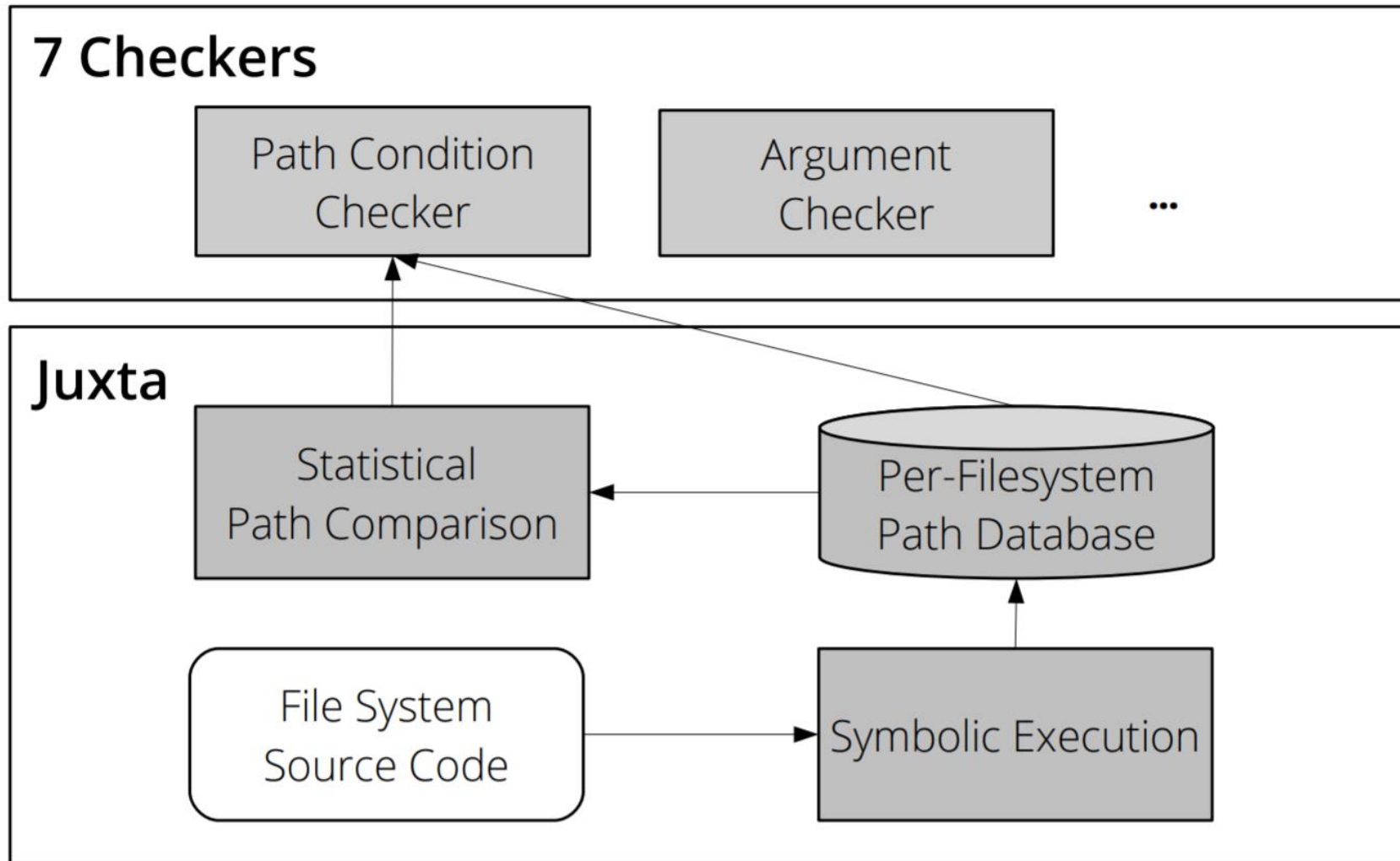
- Each file system defines how to write a page
- Semantic of writepage()
 - Success → return locked page
 - Failure → return unlocked page
- Document/filesystem/vfs specifies such a rule
 - Hard to detect semantic violations without domain knowledge

**What if 99% file systems follow above pattern,
But not one file system?
Will it be a bug?**

Juxta [SOSP 15]: Approaches

- Intuition
 - Bugs are rare
 - Majority of implementations is correct
- Idea
 - Find deviant ones as potential bugs

Juxta: overview



Juxta: Results Summary

- New semantic 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.

Outline

Part1. Bugs in File Systems

Semantic inconsistency inference

Fuzzing

Part2. Attacks and Defenses

Ransomware

Cold boot attacks

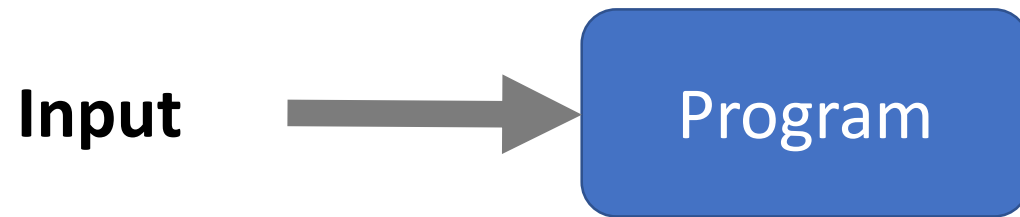
Side-channels

What programs do

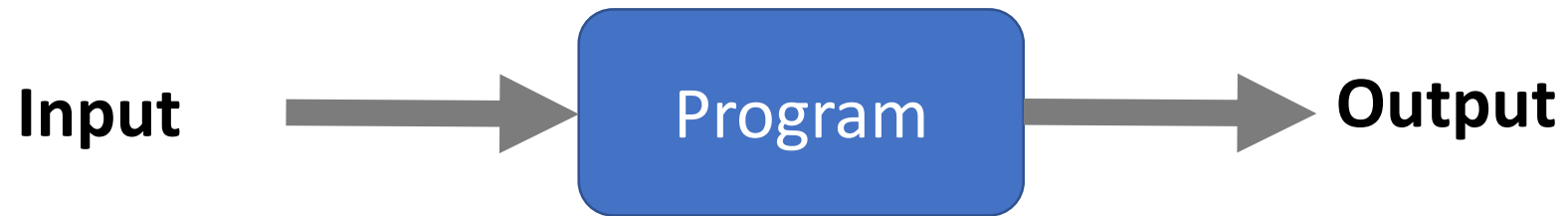


Program

What programs do



What programs do

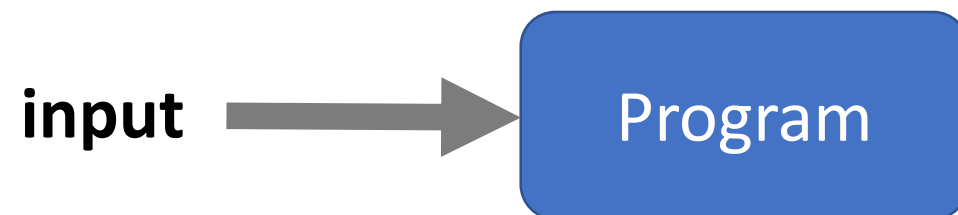


Vulnerability

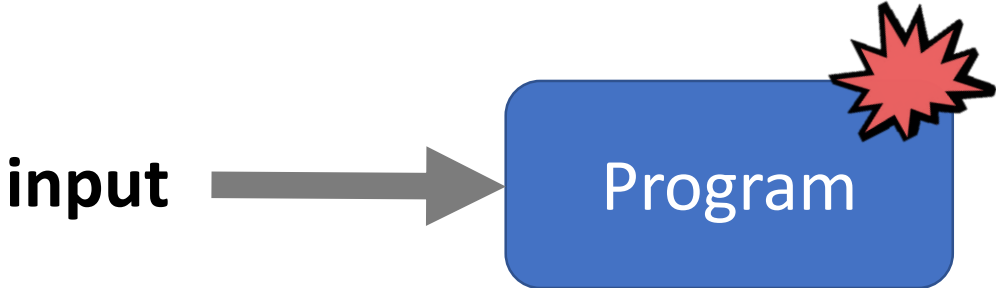


Program

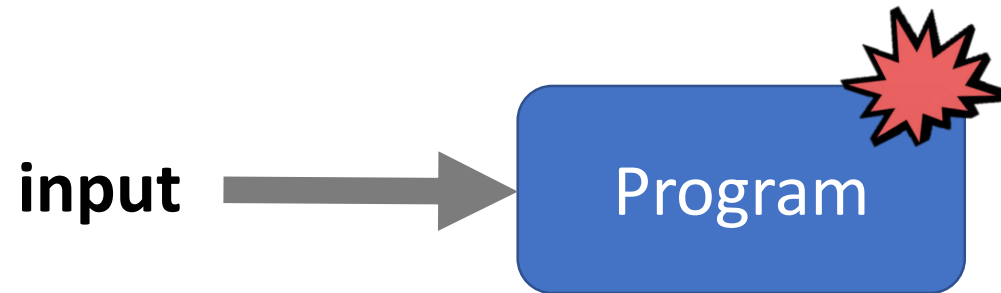
Vulnerability



Vulnerability

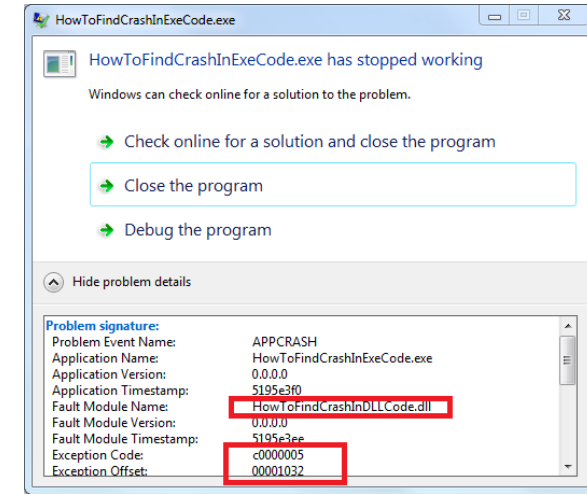
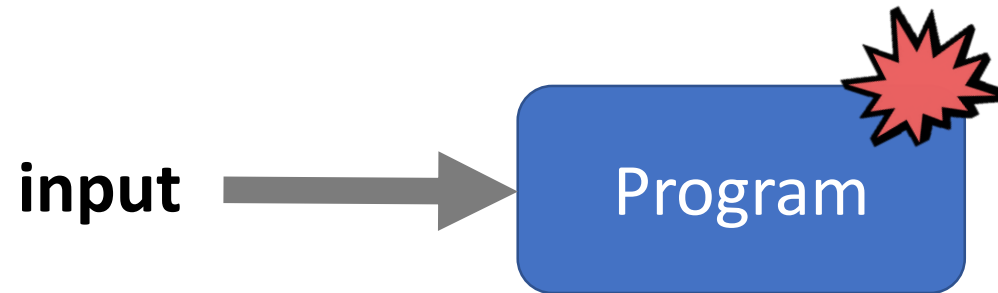


Vulnerability



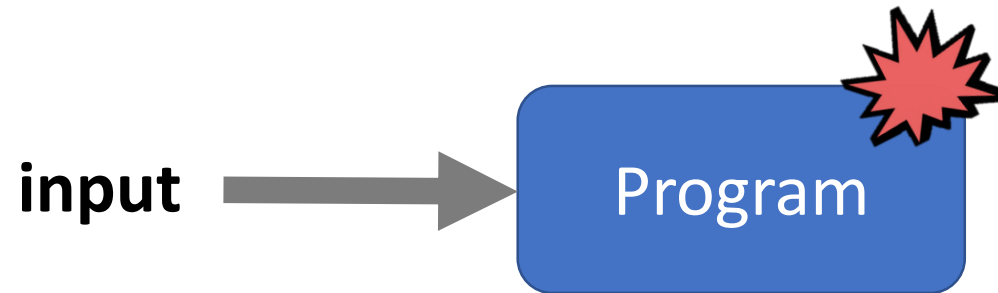
**For some input, the program acts weird
(e.g., crash, hanging, etc.)**

Vulnerability

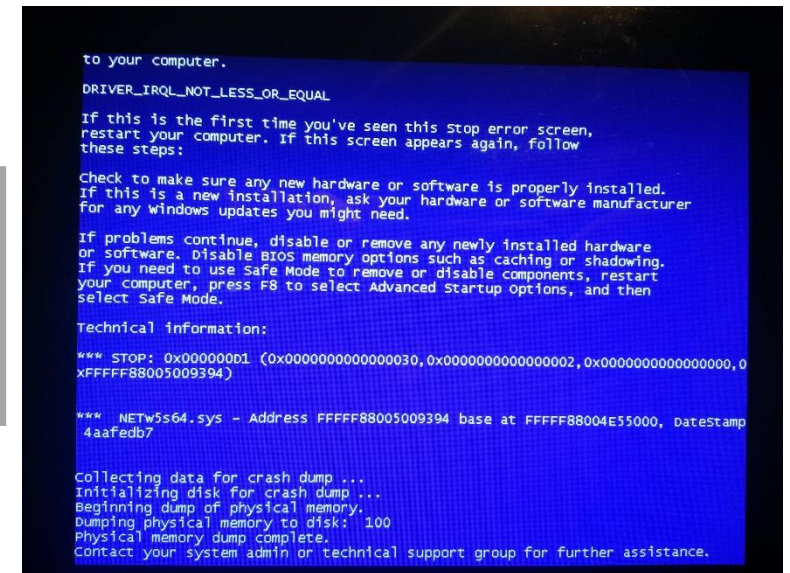
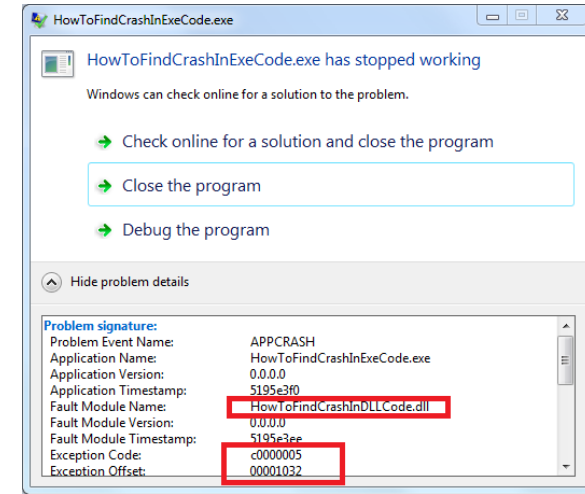


For some input, the program acts weird (e.g., crash, hanging, etc.)

Vulnerability



For some input, the program acts weird (e.g., crash, hanging, etc.)



Vulnerability: Example (Stack Overflow)

```
char input[8];  
char output[4];
```

```
i=0;  
while (1) {  
    input[i] = output[i];  
    if (input[i] == 0)  
        break;  
    i++;  
}
```

Vulnerability: Example (Stack Overflow)

```
char input[8];  
char output[4];
```

Vulnerability: Example (Stack Overflow)

```
char input[8];  
char output[4];
```

```
strcpy(output, input);
```

Vulnerability: Example (Stack Overflow)

```
char input[8];  
char output[4];
```

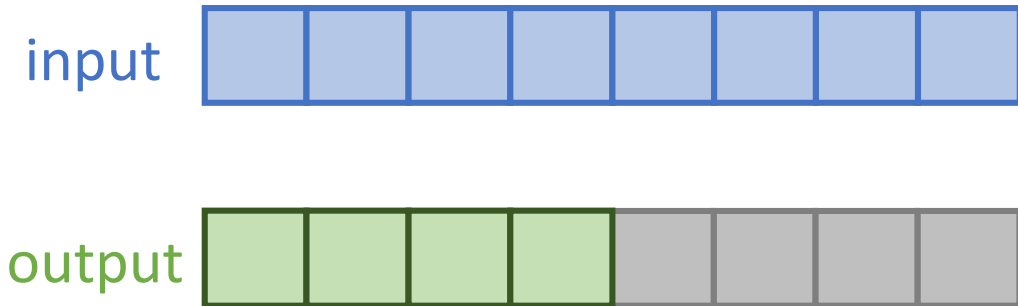
```
strcpy(output, input);
```



Vulnerability: Example (Stack Overflow)

```
char input[8];  
char output[4];
```

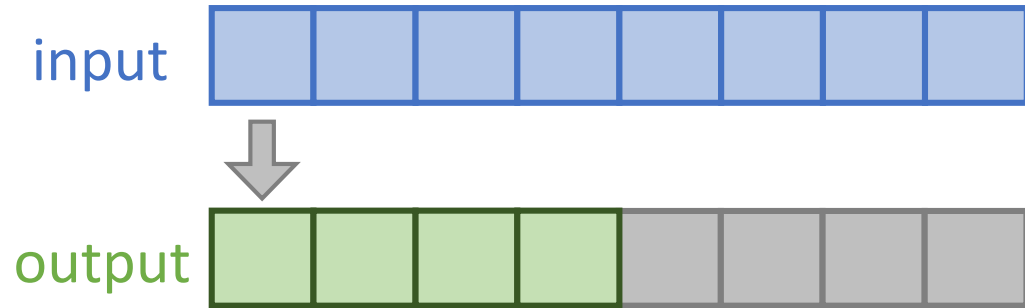
```
strcpy(output, input);
```



Vulnerability: Example (Stack Overflow)

```
char input[8];  
char output[4];
```

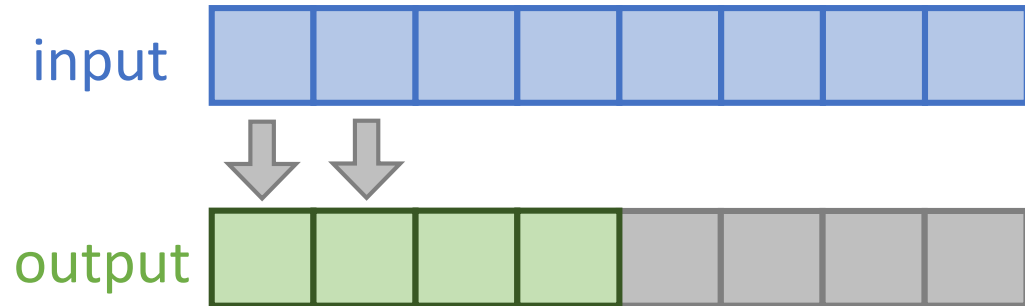
```
strcpy(output, input);
```



Vulnerability: Example (Stack Overflow)

```
char input[8];  
char output[4];
```

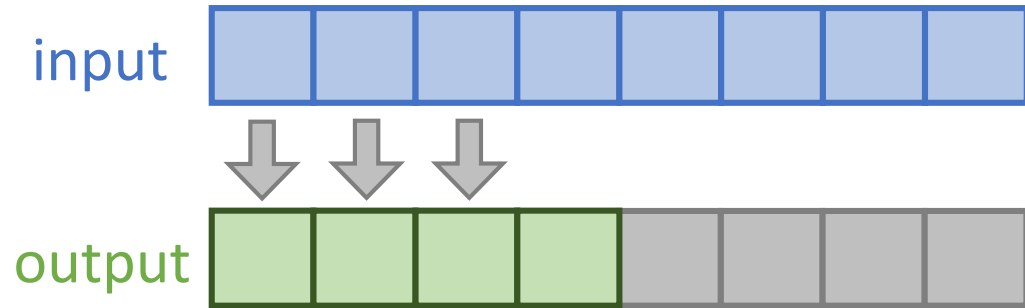
```
strcpy(output, input);
```



Vulnerability: Example (Stack Overflow)

```
char input[8];  
char output[4];
```

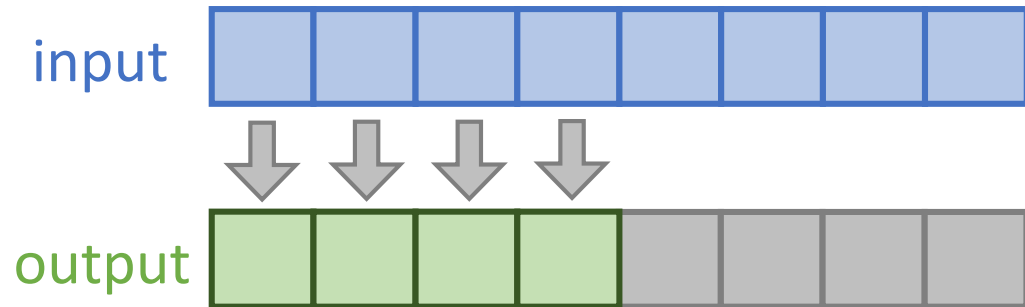
```
strcpy(output, input);
```



Vulnerability: Example (Stack Overflow)

```
char input[8];  
char output[4];
```

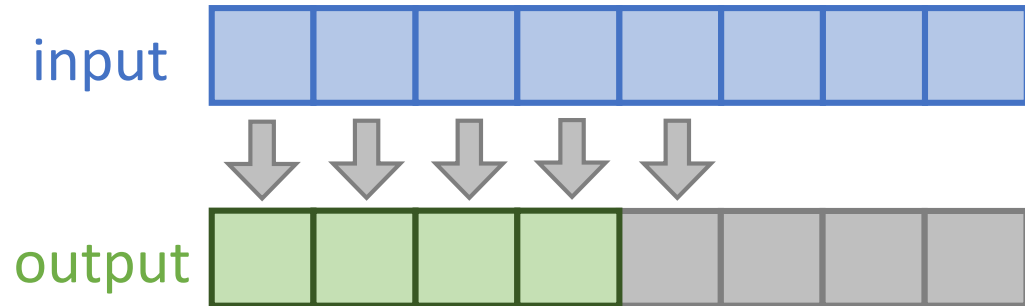
```
strcpy(output, input);
```



Vulnerability: Example (Stack Overflow)

```
char input[8];  
char output[4];
```

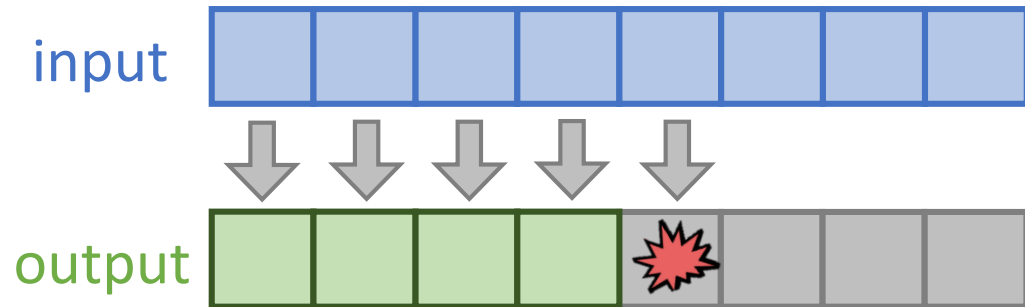
```
strcpy(output, input);
```



Vulnerability: Example (Stack Overflow)

```
char input[8];  
char output[4];
```

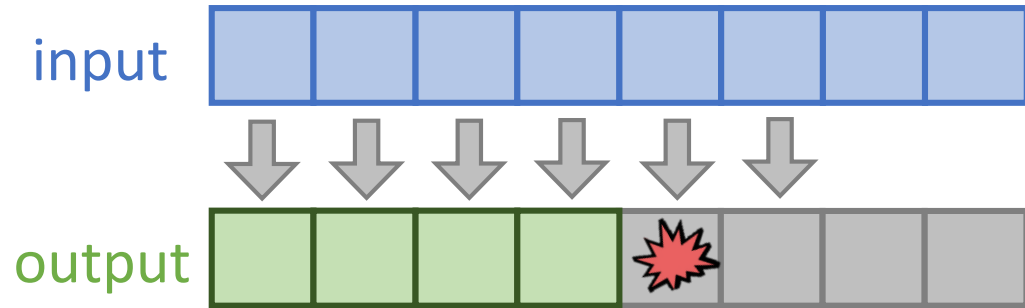
```
strcpy(output, input);
```



Vulnerability: Example (Stack Overflow)

```
char input[8];  
char output[4];
```

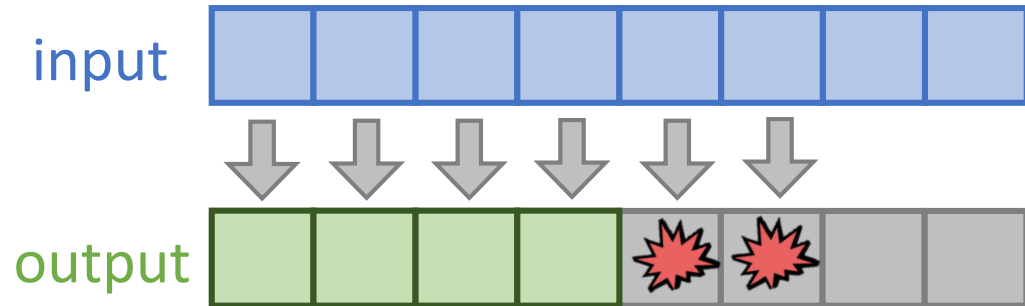
```
strcpy(output, input);
```



Vulnerability: Example (Stack Overflow)

```
char input[8];  
char output[4];
```

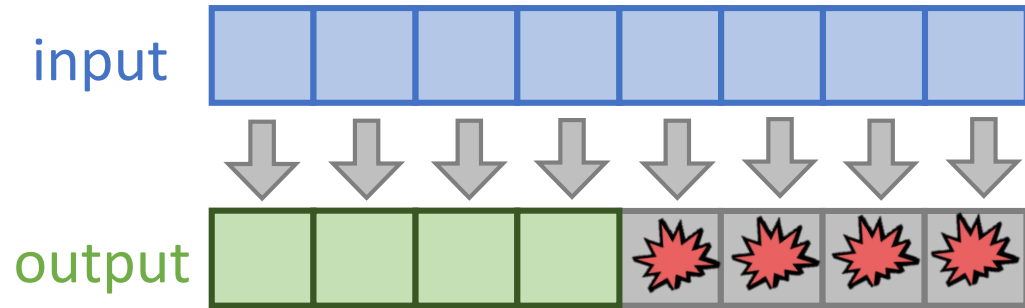
```
strcpy(output, input);
```



Vulnerability: Example (Stack Overflow)

```
char input[8];  
char output[4];
```

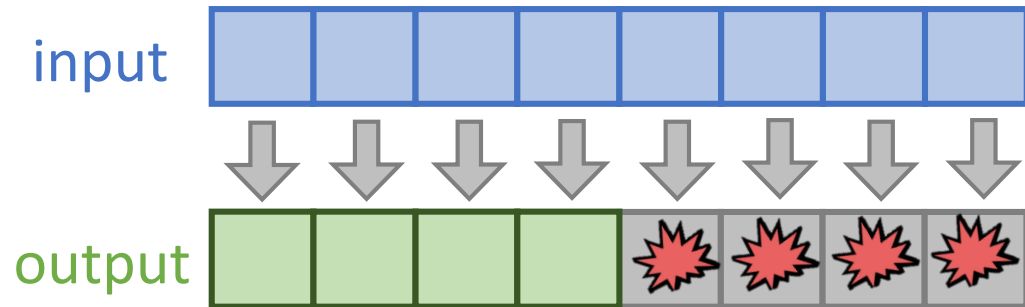
```
strcpy(output, input);
```



Vulnerability: Example (Stack Overflow)

```
char input[8];  
char output[4];
```

```
strcpy(output, input);
```

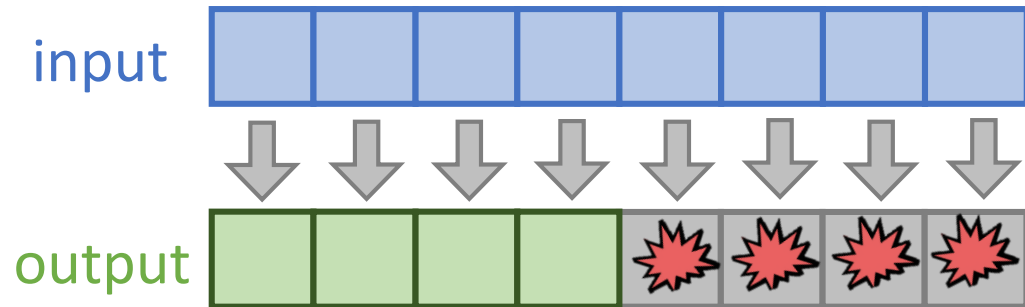


If `strlen(input) > 4`, a vulnerability occurs.

Vulnerability: Example (Stack Overflow)

```
char input[8];  
char output[4];
```

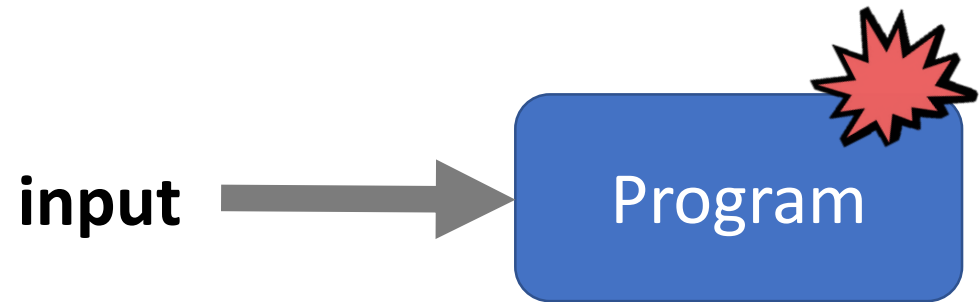
```
strcpy(output, input);
```



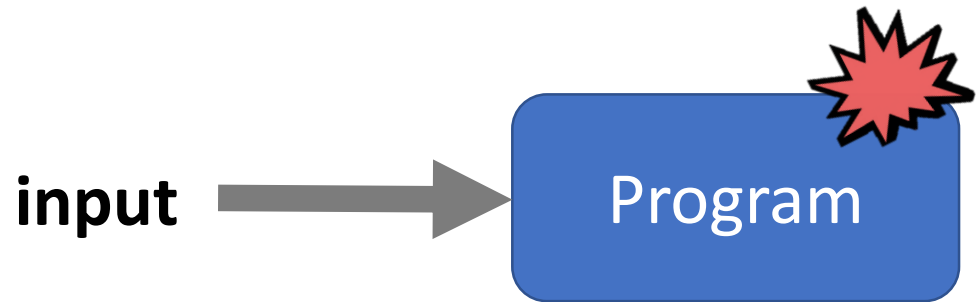
If $\text{strlen}(\text{input}) > 4$, a vulnerability occurs.

Q. How to **automatically know** the program will act bad if $\text{strlen}(\text{input}) > 4$?

Fuzzing: Random Testing!

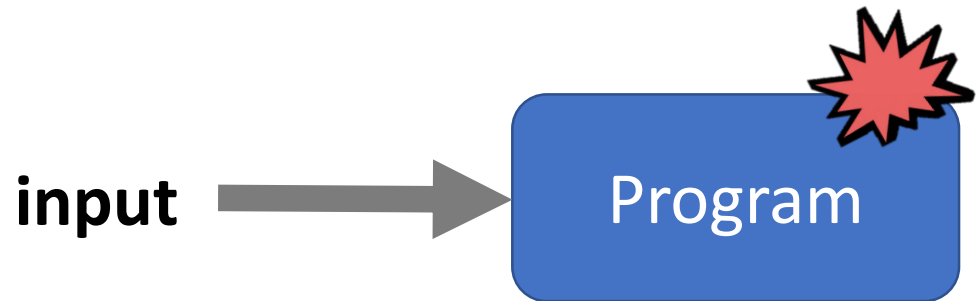


Fuzzing: Random Testing!



Fuzzing algorithm

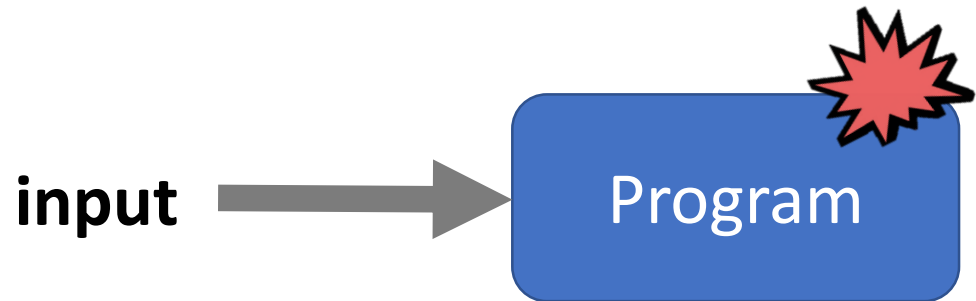
Fuzzing: Random Testing!



Fuzzing algorithm

#1. Randomly pick input $x^R \in X$

Fuzzing: Random Testing!

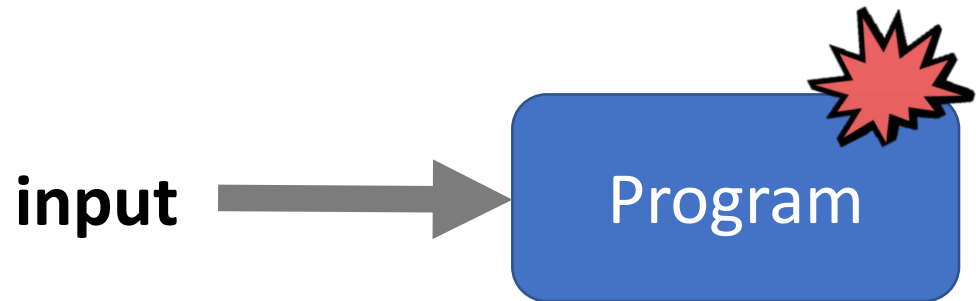


Fuzzing algorithm

#1. Randomly pick input $x^R \in X$

#2. Run the program using x^R
- If something bad occurs,
 x^R induces a vulnerability

Fuzzing: Random Testing!



Fuzzing algorithm

#1. Randomly pick input $x^R \in X$

#2. Run the program using x^R
- If something bad occurs,
 x^R induces a vulnerability

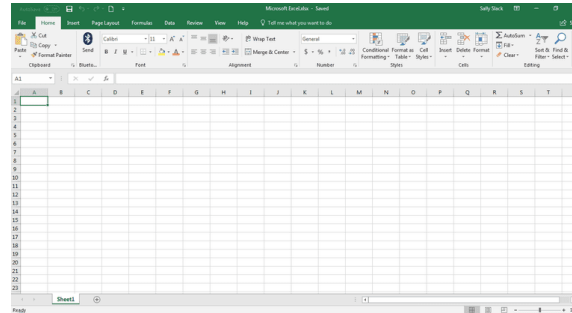
#3. Go back to step #1

Fuzzing in the old days: Excel example



**Randomly
Generate/mutate
an Excel file**

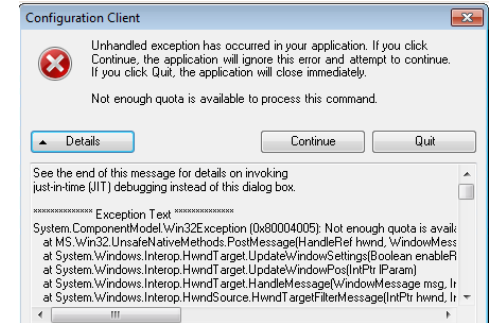
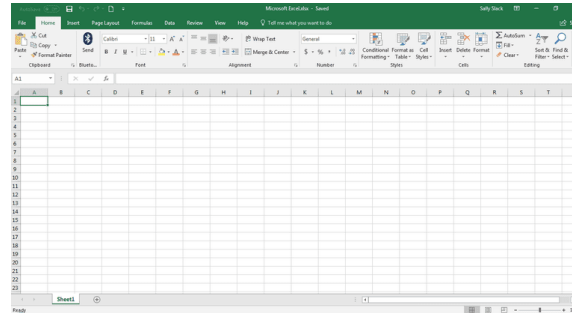
Fuzzing in the old days: Excel example



**Randomly
Generate/mutate
an Excel file**

Run Excel

Fuzzing in the old days: Excel example

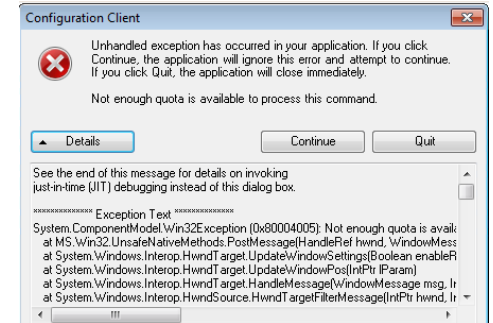
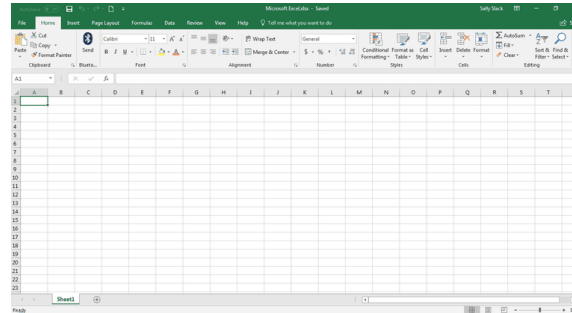


**Randomly
Generate/mutate
an Excel file**

Run Excel

Crash!

Fuzzing in the old days: Excel example



**Randomly
Generate/mutate
an Excel file**

Run Excel

Crash!

Finding a crashing input gets harder and harder ☹️

Fuzzing: Desired Features

- Fuzzing attempts to
 - Detect explicit violation
 - While exploring all different program states in a given implementation
- Desired features
 - Good violation detection capability
 - Exploring all different program states

Fuzzing: Good Violation Detection Capability

- Memory bug
 - Violation conditions are clear
 - Per-pointer capability w.r.t. accessing memory space
- Concurrency bug
 - Violation conditions are clear
 - If race occurs, it is undefined behavior
 - Challenges: False positive issues in concurrency bug detection
 - Undefined behavior can be implementation-defined behavior
 - Developers may intentionally introduce races
 - Difficult to differentiate benign VS harmful races
- Semantic bug
 - Challenges: Violation conditions are **unclear**
 - We don't even know violation conditions
 - Only assertions (manually inserted by developers) may help
 - Only a few research have been done

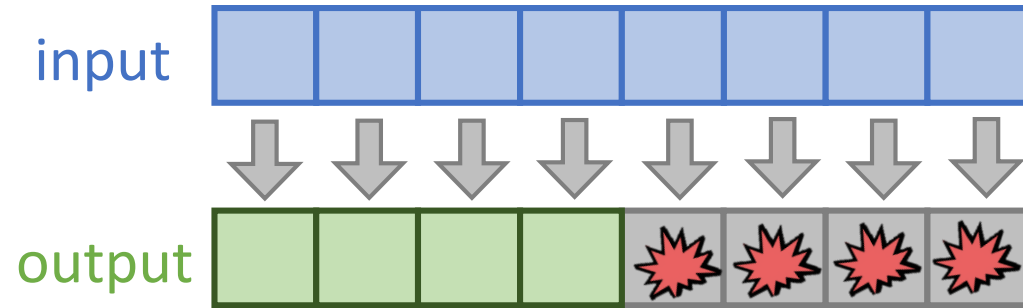
Fuzzing: Exploring all different program states

- State exploration and bug finding
 - More code coverage → More complete testing → More bugs
 - Most fuzzing techniques take coverage-oriented genetic algorithm (proposed by AFL)
- Coverage-oriented genetic algorithm for fuzzing
 - Design an object function f to evaluate goodness of input's coverage
 - Algorithm
 - Initialize a corpus C
 - Corpus: a set of inputs to be mutated
 - Generate an input i
 - Generation: Randomly generate i from scratch
 - Mutation: Pick c from C , and then mutate c
 - If $f(i) \geq f(c)$ for all c in C
 - Add input i to the corpus C

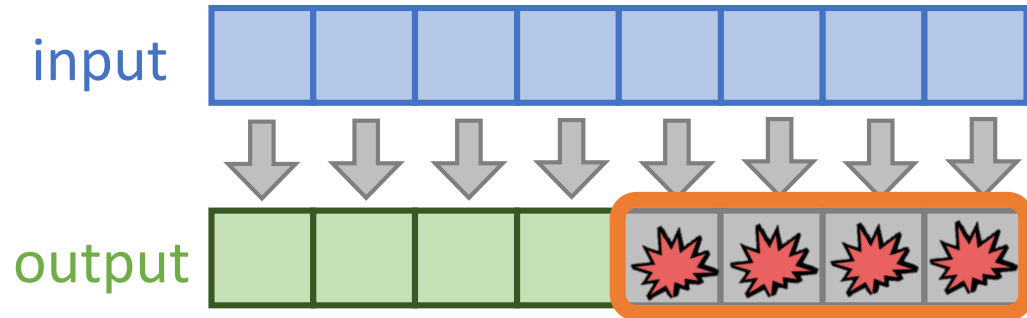
Fuzzing: Exploring all different program states

- Covering complete context sensitivity is challenging
- Concurrency bugs are much more complex
 - Scheduler mechanisms can be seen as non-deterministic
 - For fuzzing, input definition is complicated
 - Should consider thread interleaving
 - Related to hardness of race bug reproduction

Challenges in Fuzzing: Violation Detection



Challenges in Fuzzing: Violation Detection



This crash may not be observable in practice.

It may simply overwrite some other variables, and it won't involve critically bad behaviors.

Memory error bugs

- Spatial safety violation
 - When accessing beyond allocated memory regions
 - e.g., stack overflow, heap overflow
- Temporal safety violation
 - When accessing deallocated memory regions
 - e.g., use-after-free, double-free

Memory Error: Detection Approaches

- Pointer-based detection
 - Keep track of each pointer's capability w.r.t. memory accesses
 - Software: SoftBound [PLDI 09], CETS [ISMM 10]
 - Hardware: Watch Dog [ISCA 12], Intel Memory Protection Extension (MPX)
- Redzone-based detection
 - Keep track of global memory space accessibility
 - Software: AddressSanitizer [ATC 12]
 - Hardware: REST [ISCA 18]

SoftBound [PLDI 09]

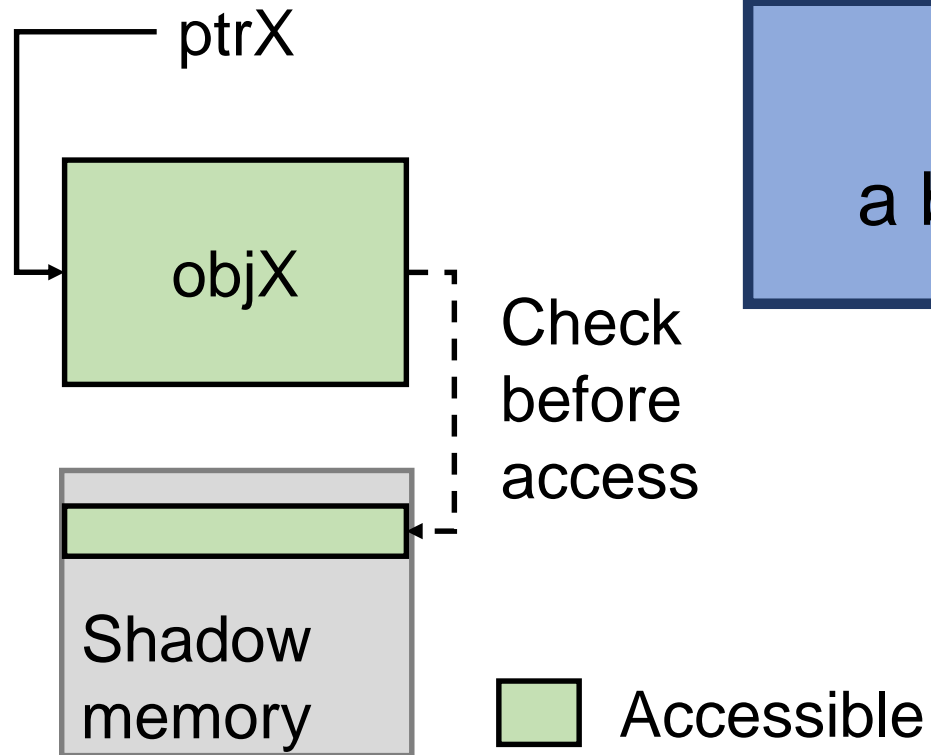
- Maintain per-pointer metadata
 - (base, bound) per pointer
- Runtime safety enforcement
 - All memory access should be within the range (base, base+bound)
- Metadata propagation
 - Per-pointer metadata is propagated when performing pointer arithmetic operations
- Disjoint metadata scheme
 - For better compatibility, metadata space is isolated from original program's memory space

Intel MPX

- Hardware-assisted version of SoftBound
 - Hardware-assisted metadata management
 - Metadata management: Bound registers + Bound table
 - Bound registers
 - (base, bound) information is now stored in CPU registers
 - Fast bound check: a dedicated instruction is supported by MPX
 - Bound table
 - Used if bound registers are not available
- Limitations
 - Compatibility issues (arbitrary type casting, external libraries, multi-threading)
 - More details: Intel MPX Explained [SIGMETRICS 18]

AddressSanitizer [ATC 12]

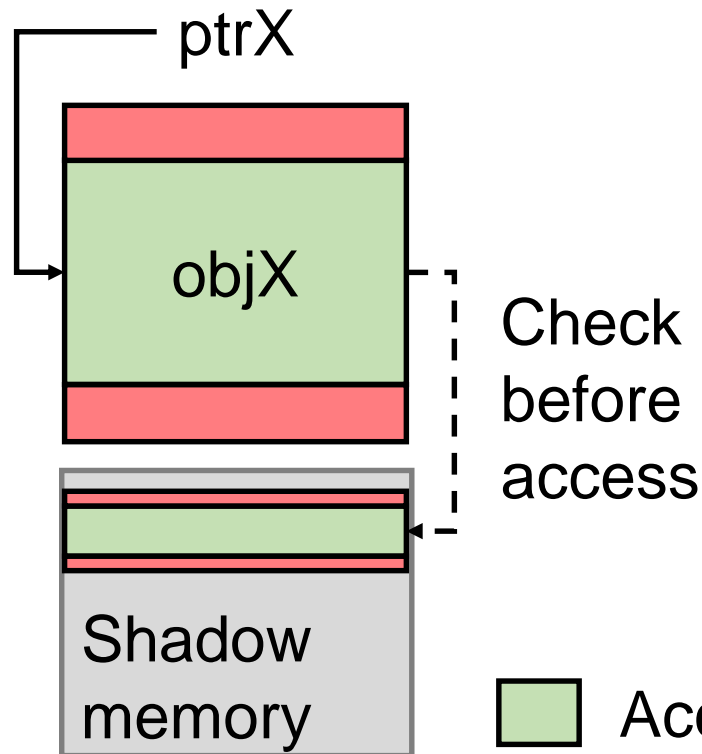
- Buffer overflow (spatial memory errors)



Shadow memory:
a bitmap to validate all addresses



AddressSanitizer [ATC 12]

- Buffer overflow (spatial memory errors)



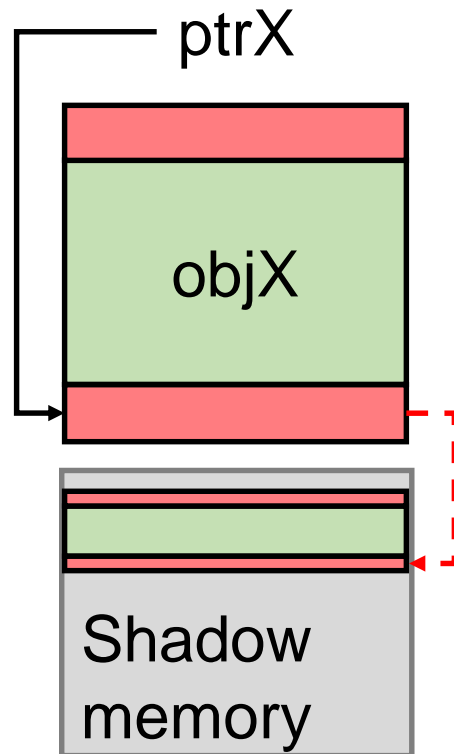
Shadow memory:
a bitmap to validate all addresses

Redzone:
inaccessible region between objects

-  Accessible
-  Inaccessible (redzone)

AddressSanitizer [ATC 12]

- Buffer overflow (spatial memory errors)



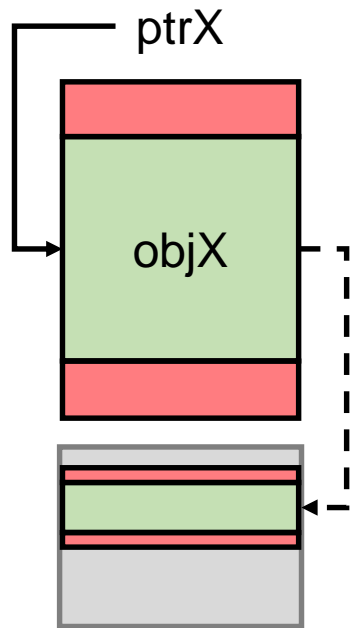
Shadow memory:
a bitmap to validate all addresses

Redzone:
inaccessible region between objects

- Accessible
- Inaccessible (redzone)

AddressSanitizer [ATC 12]

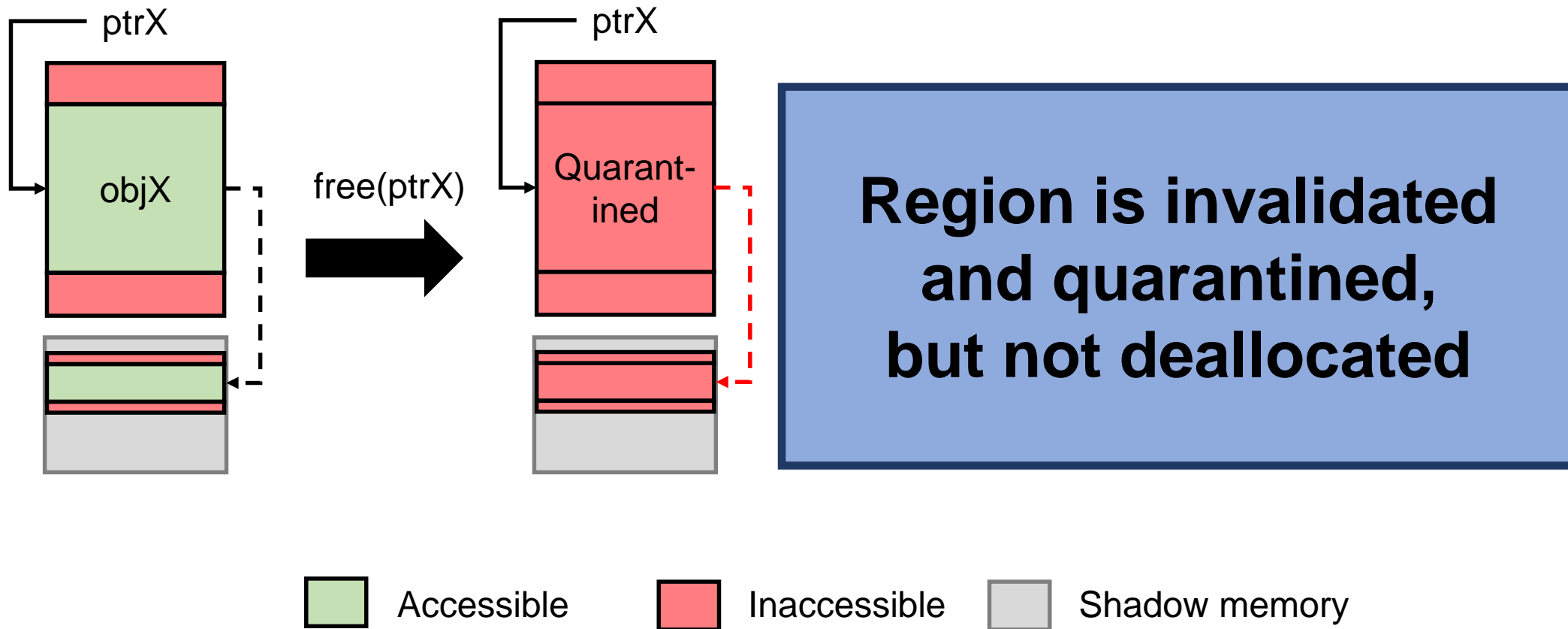
- Use-after-free (Temporal memory errors)



 Accessible  Inaccessible  Shadow memory

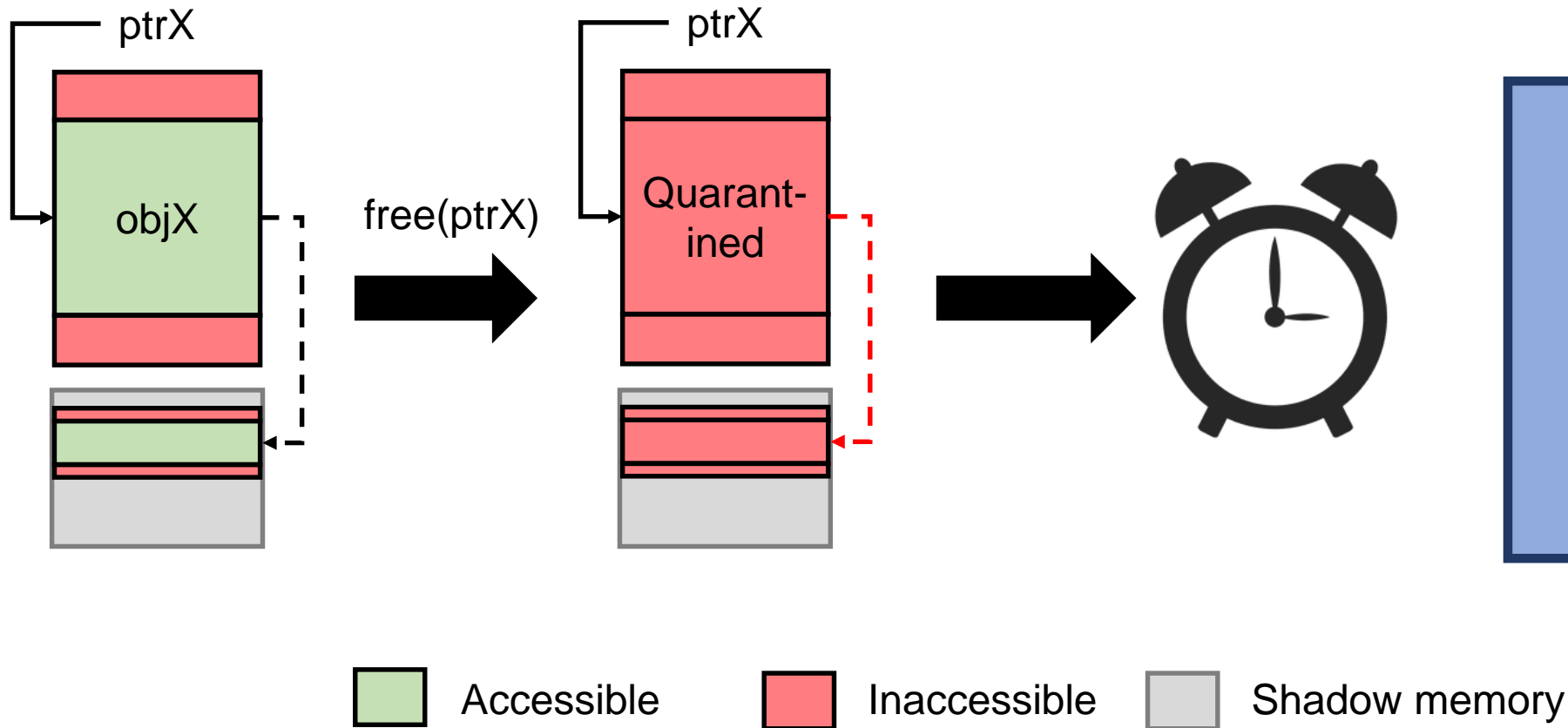
AddressSanitizer [ATC 12]

- Use-after-free (Temporal memory errors)



AddressSanitizer [ATC 12]

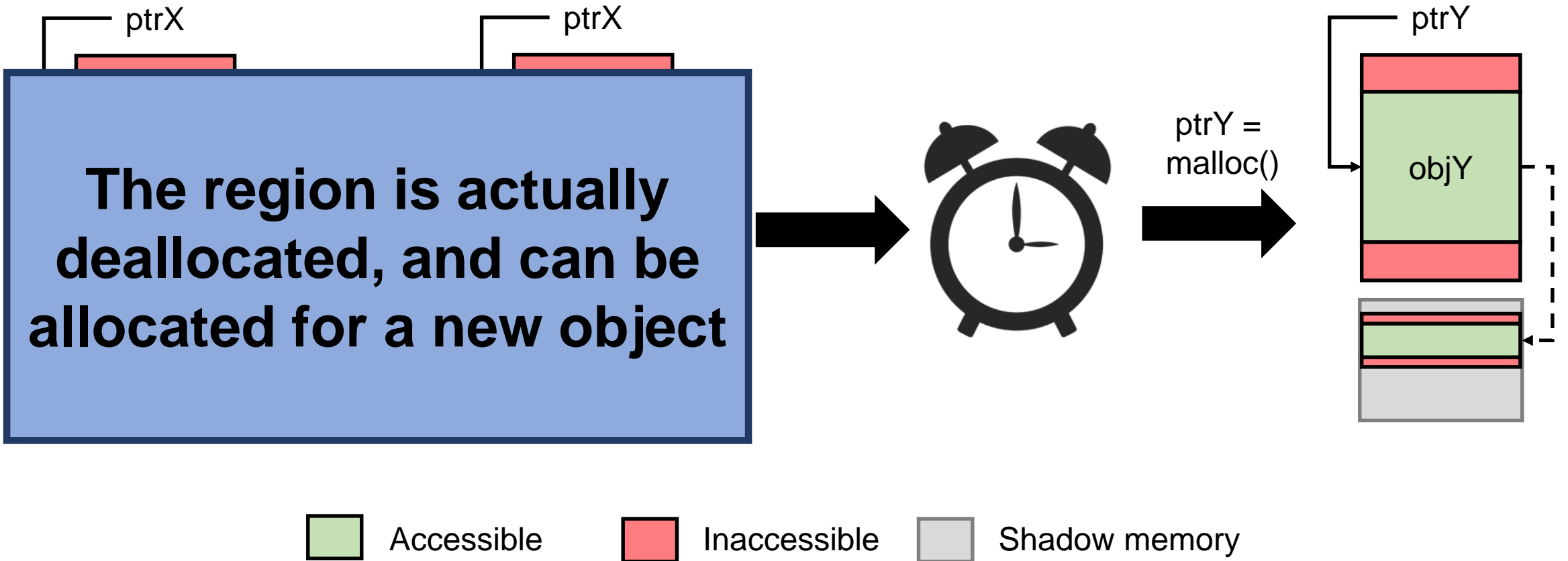
- Use-after-free (Temporal memory errors)



Hold the region until quarantine zone is full (FIFO)

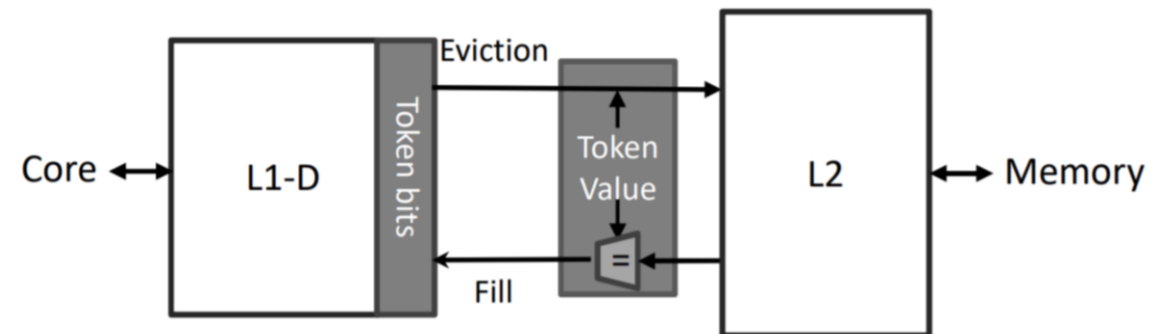
AddressSanitizer [ATC 12]

- Use-after-free (Temporal memory errors)



REST [ISCA 18]

- Hardware-assisted version of AddressSanitizer
- A fixed token value in redzone
 - A pre-determined random value (in 64-byte space) is reserved for representing an invalid memory region
 - No need to maintain shadow memory
- Validity checks on memory read/write can be very efficient
 - Simply check if the token bit (in cache) is set



Race condition

- Race conditions
 - If following three conditions meet
 - Two instructions access the same memory location
 - At least one of two is a write instruction
 - These two are executed concurrently
- If a race condition occurs, the computational results may vary depending on the execution order.

Race condition: A simple example

```
a = 0;
```

```
if (a) {  
    b = 0;  
} else {  
    b = 1;  
}
```

Thread 1

```
a = 1;
```

Thread 2

Race condition: A simple example

```
a = 0;
```

```
If (a) {  
    b = 0;  
} else {  
    b = 1;  
}
```

Thread 1

```
a = 1;
```

Thread 2

b will be 1

Race condition: A simple example

```
a = 0;
```

```
If (a) {  
    b = 0;  
} else {  
    b = 1;  
}
```

Thread 1

```
a = 1;
```

Thread 2

b will be 1

```
a = 0;
```

```
If (a) {  
    b = 0;  
} else {  
    b = 1;  
}
```

Thread 1

```
a = 1;
```

Thread 2

Race condition: A simple example

```
a = 0;
```

```
If (a) {  
    b = 0;  
} else {  
    b = 1;  
}
```

Thread 1

```
a = 1;
```

Thread 2

b will be 1

```
a = 0;
```

```
If (a) {  
    b = 0;  
} else {  
    b = 1;  
}
```

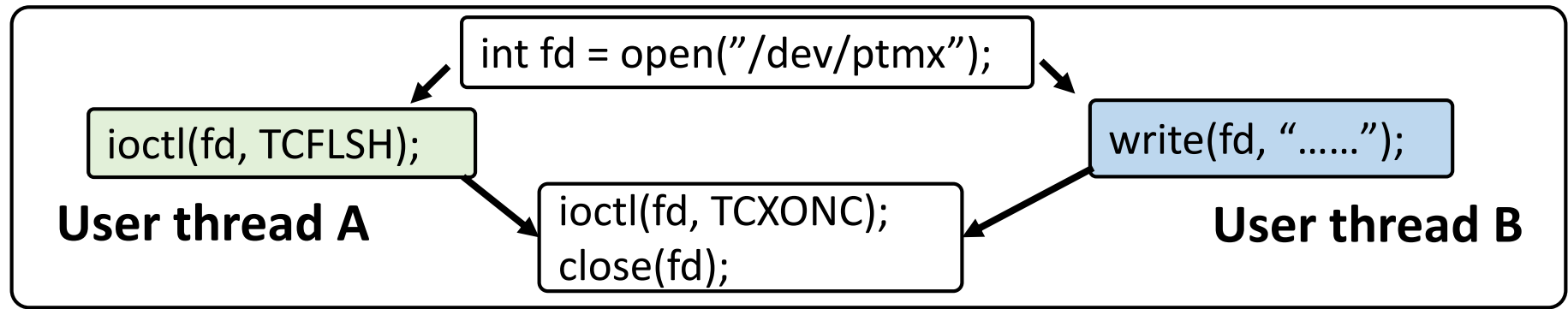
Thread 1

```
a = 1;
```

Thread 2

b will be 0

**Real-world
race example:
CVE-2017-2636
(Linux Kernel)**



User

```
431: if (n_hdlc->tbuf) {
432:   push_back(free_list, n_hdlc->tbuf);
-----
440:   n_hdlc->tbuf = NULL;
-----
441: }
```

Kernel thread A

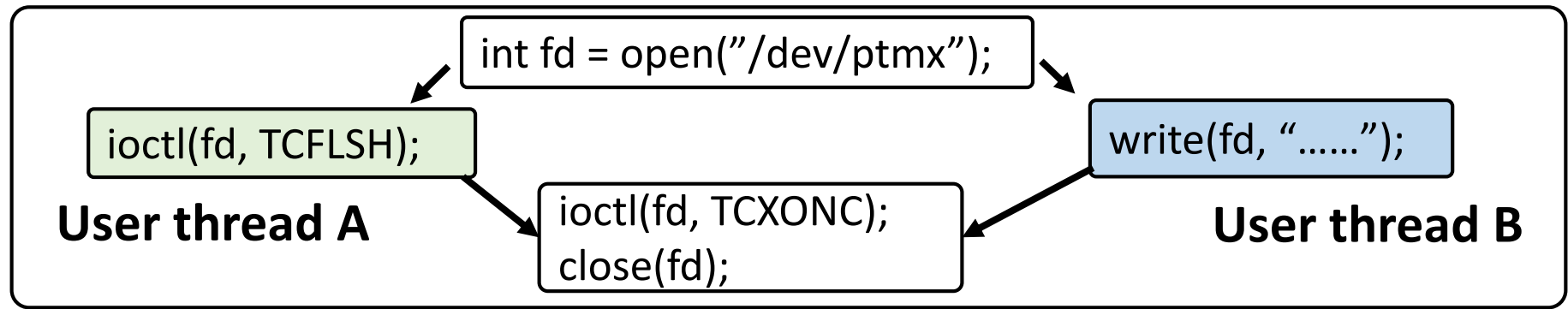
```
216: tbuf = n_hdlc->tbuf;
-----
217: if (tbuf)
218:   push_back(free_list, tbuf);
```

Kernel thread B

Kernel

```
266: if (n_hdlc->flag & TCXONC)
267:   while (list_empty(free_list)) {
268:     buf = pop_front(free_list);
269:     kfree(buf);
270: }
```

**Real-world
race example:
CVE-2017-2636
(Linux Kernel)**



User

```
431: if (n_hdlc->tbuf) {
432:   push_back(free_list, n_hdlc->tbuf);
-----
440:   n_hdlc->tbuf = NULL;
-----
441: }
```

Kernel thread A

```
216: tbuf = n_hdlc->tbuf;
-----
217: if (tbuf)
218:   push_back(free_list, tbuf);
```

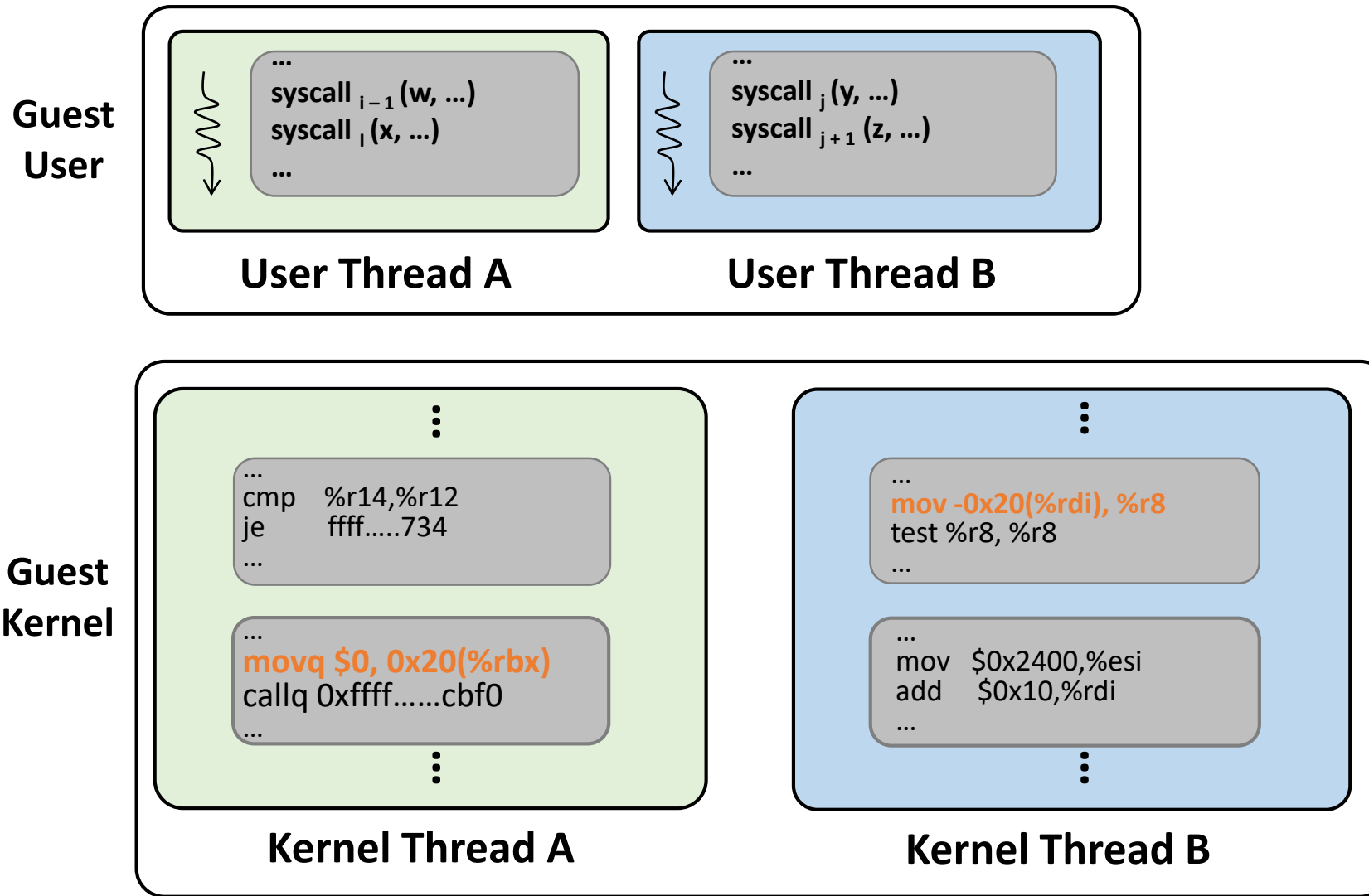
Kernel thread B

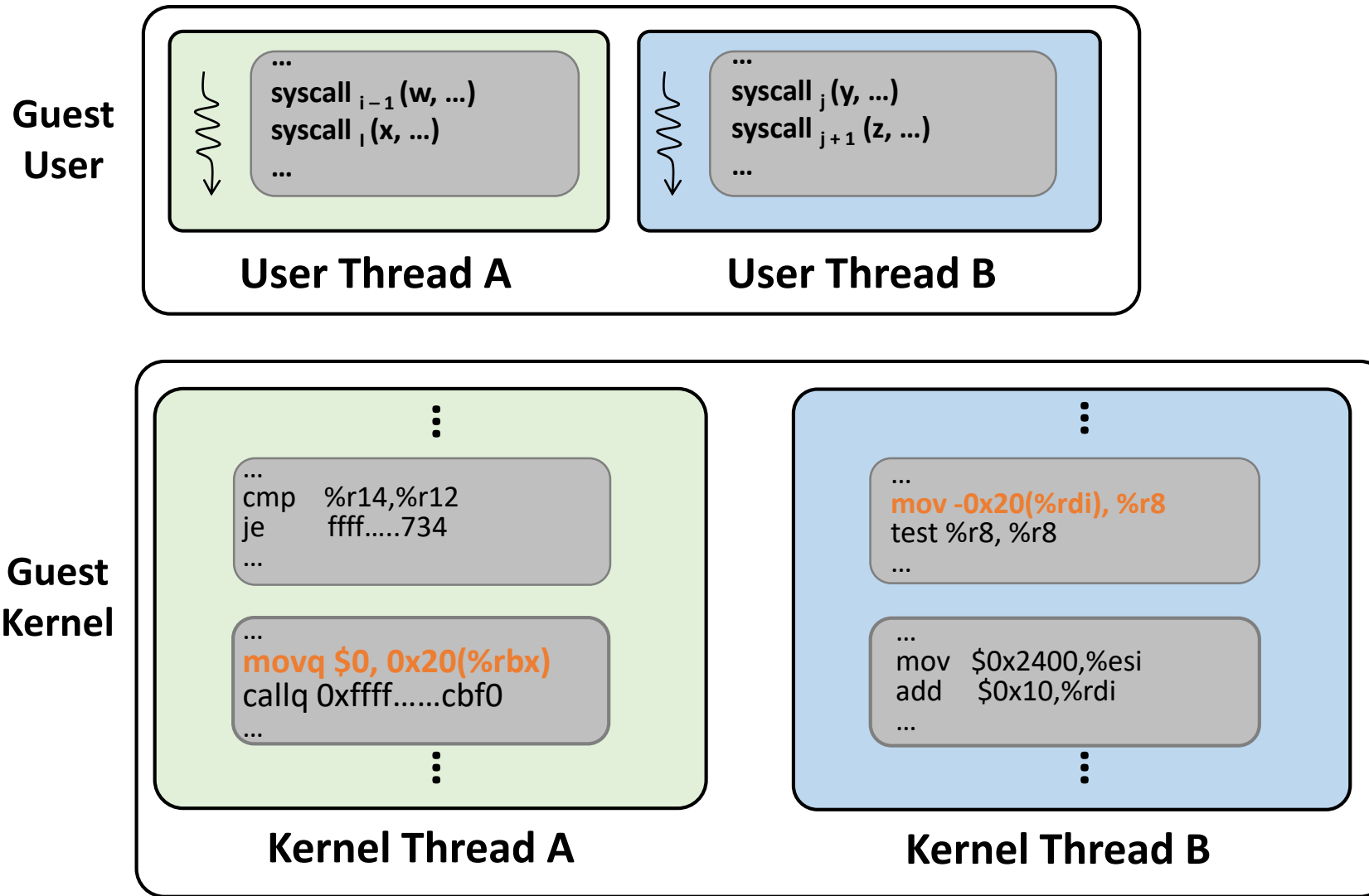
Kernel

**This will turn into “double-free”,
which in turn allows a privilege escalation**

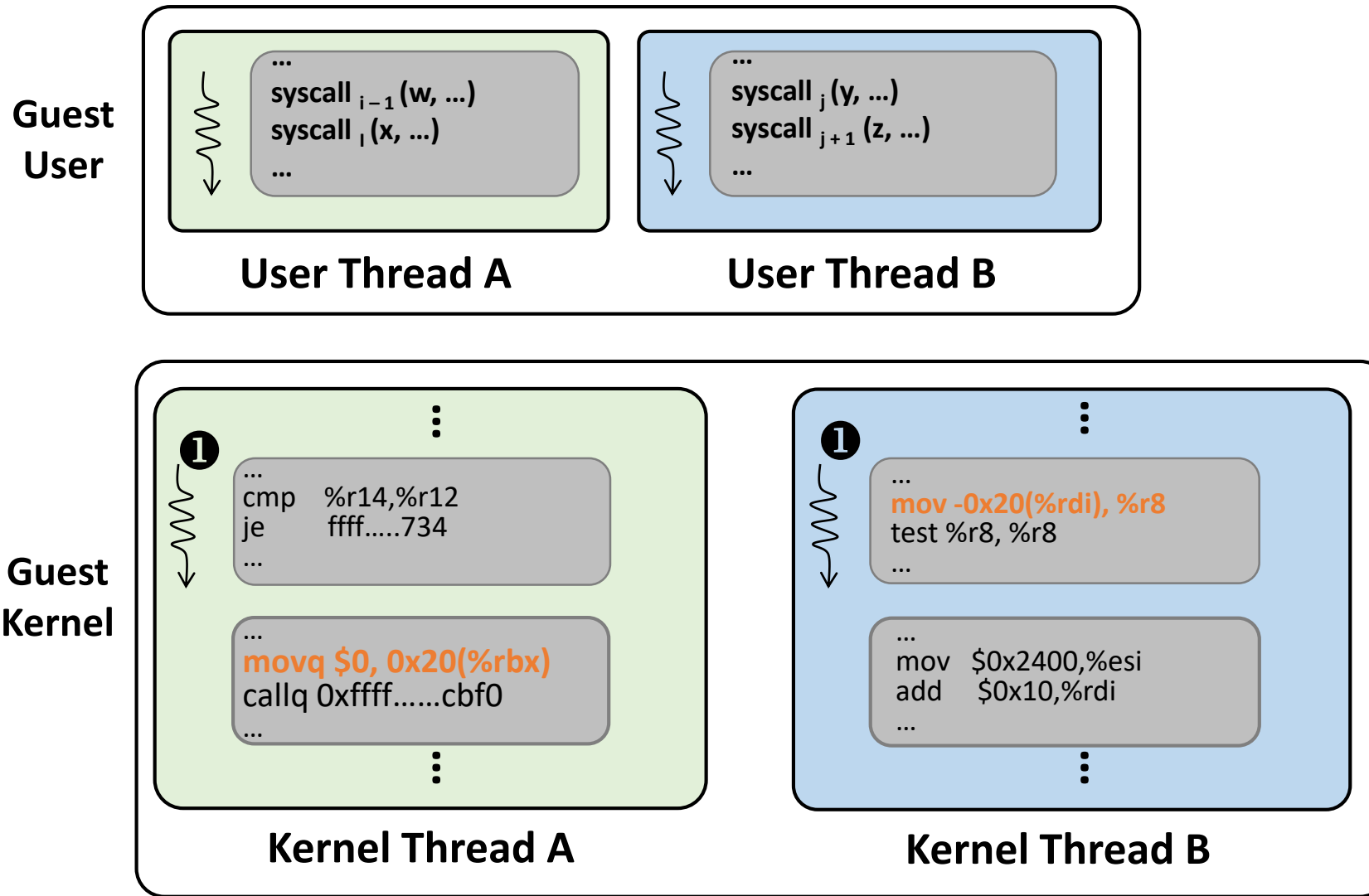
Race condition and fuzzing

- Finding race conditions through fuzzing is challenging
 - How to execution racing pair instructions **concurrently**?
 - All the execution interleaving (scheduling) is “randomly” determined
- Case study: Previous techniques
 - Syzkaller: a system call fuzzer for the Linux kernel
 - SKI: an academic prototype to find a race [OSDI 14]

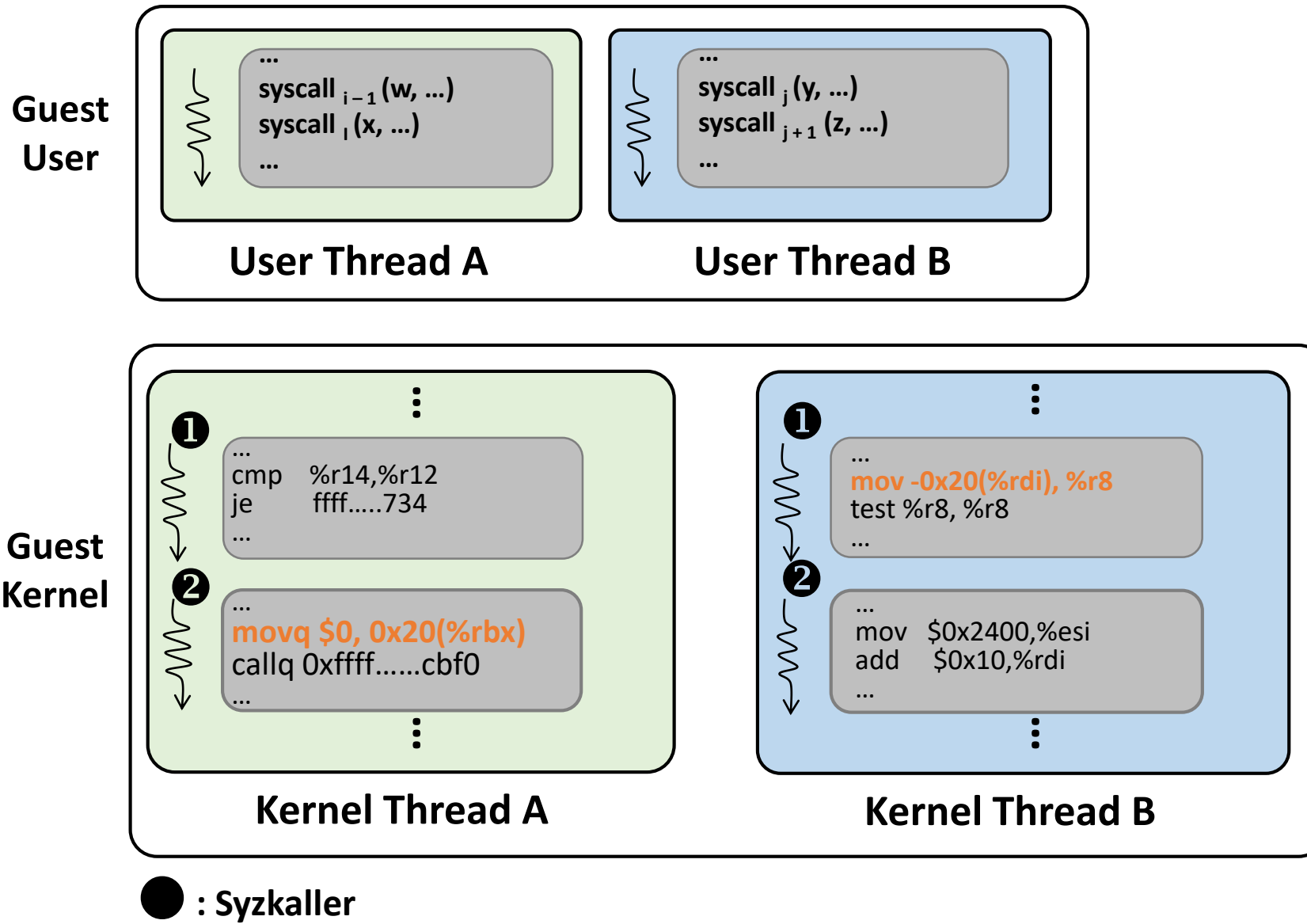


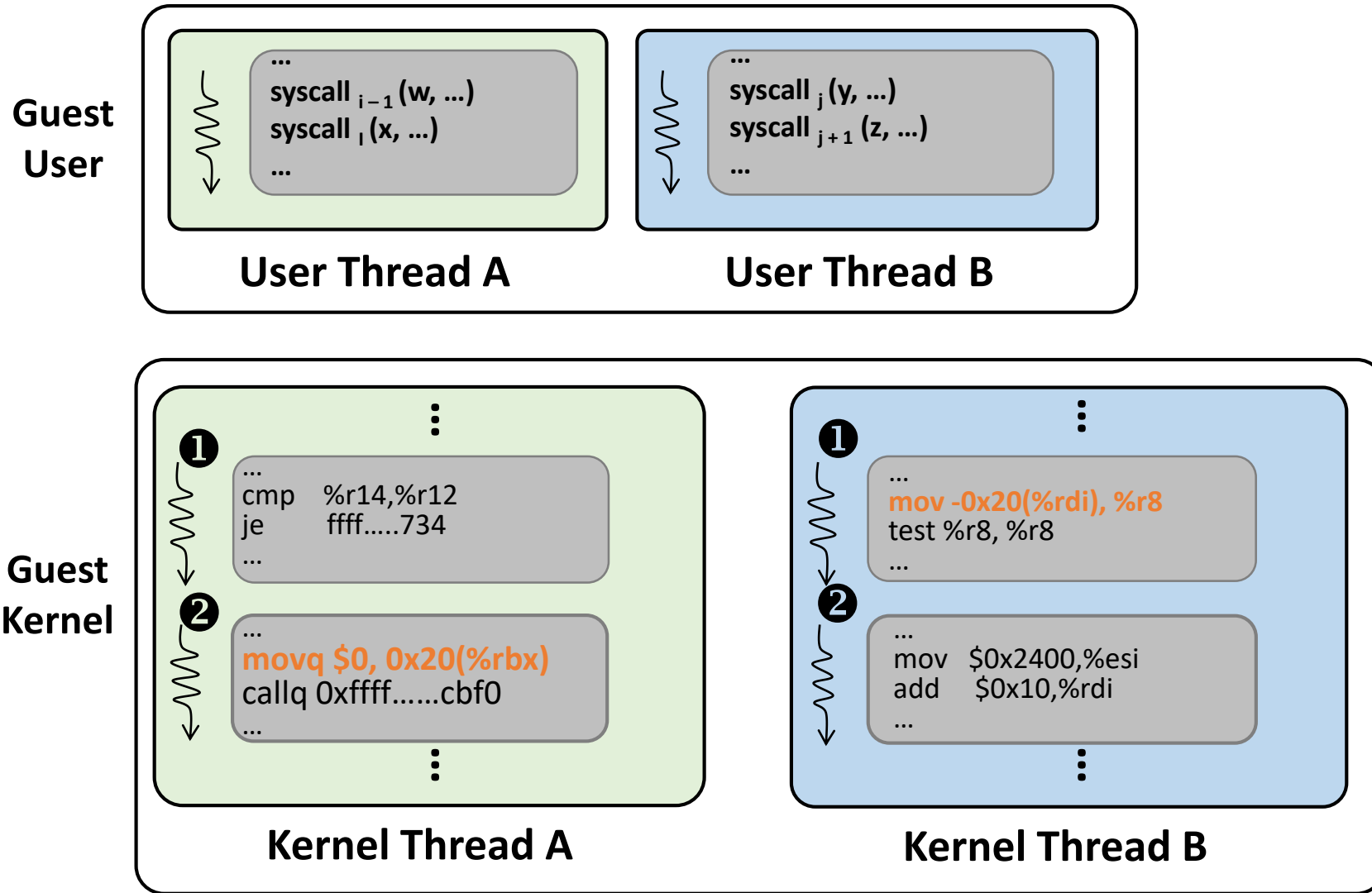


● : Syzkaller



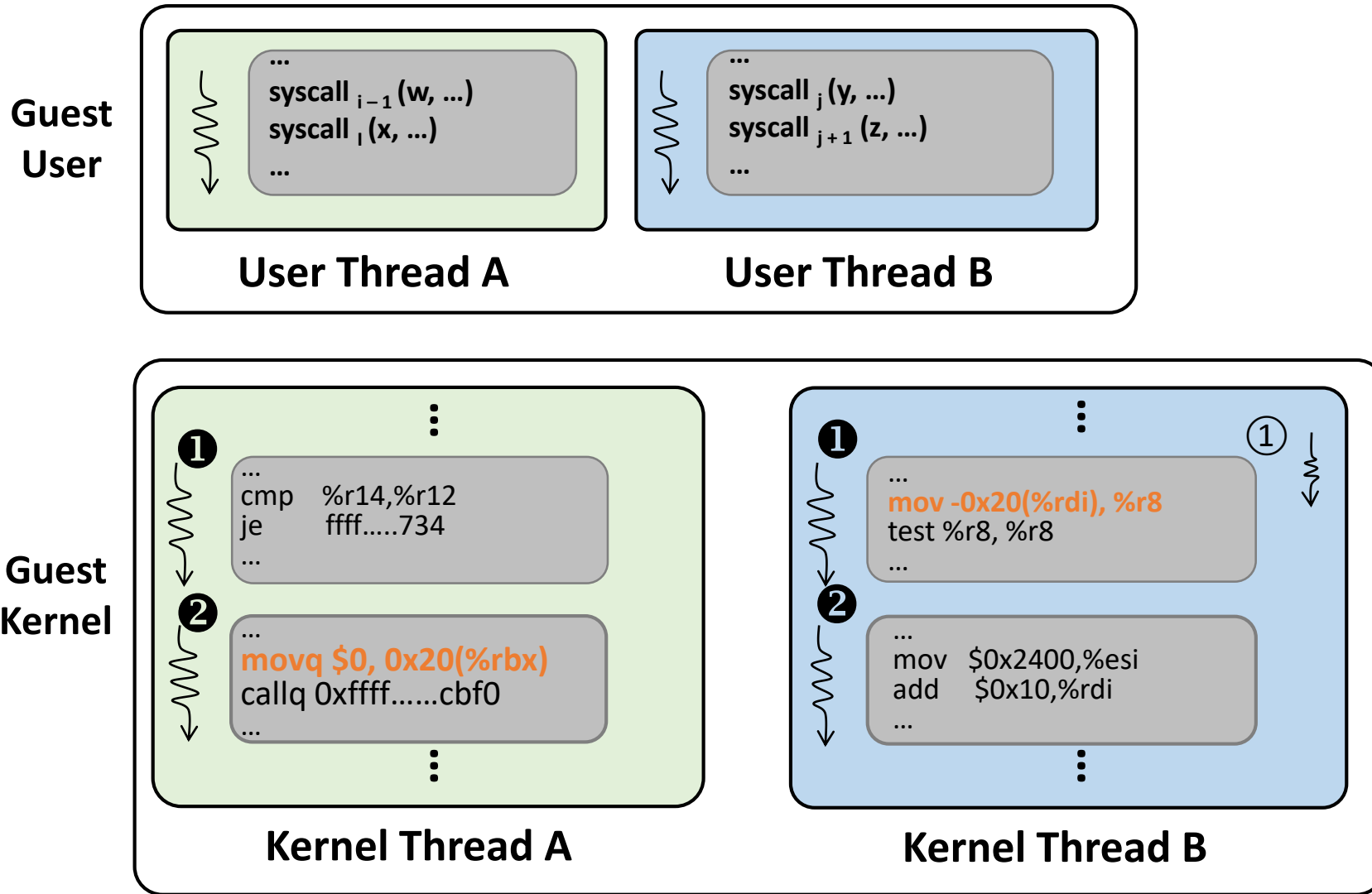
● : Syzkaller





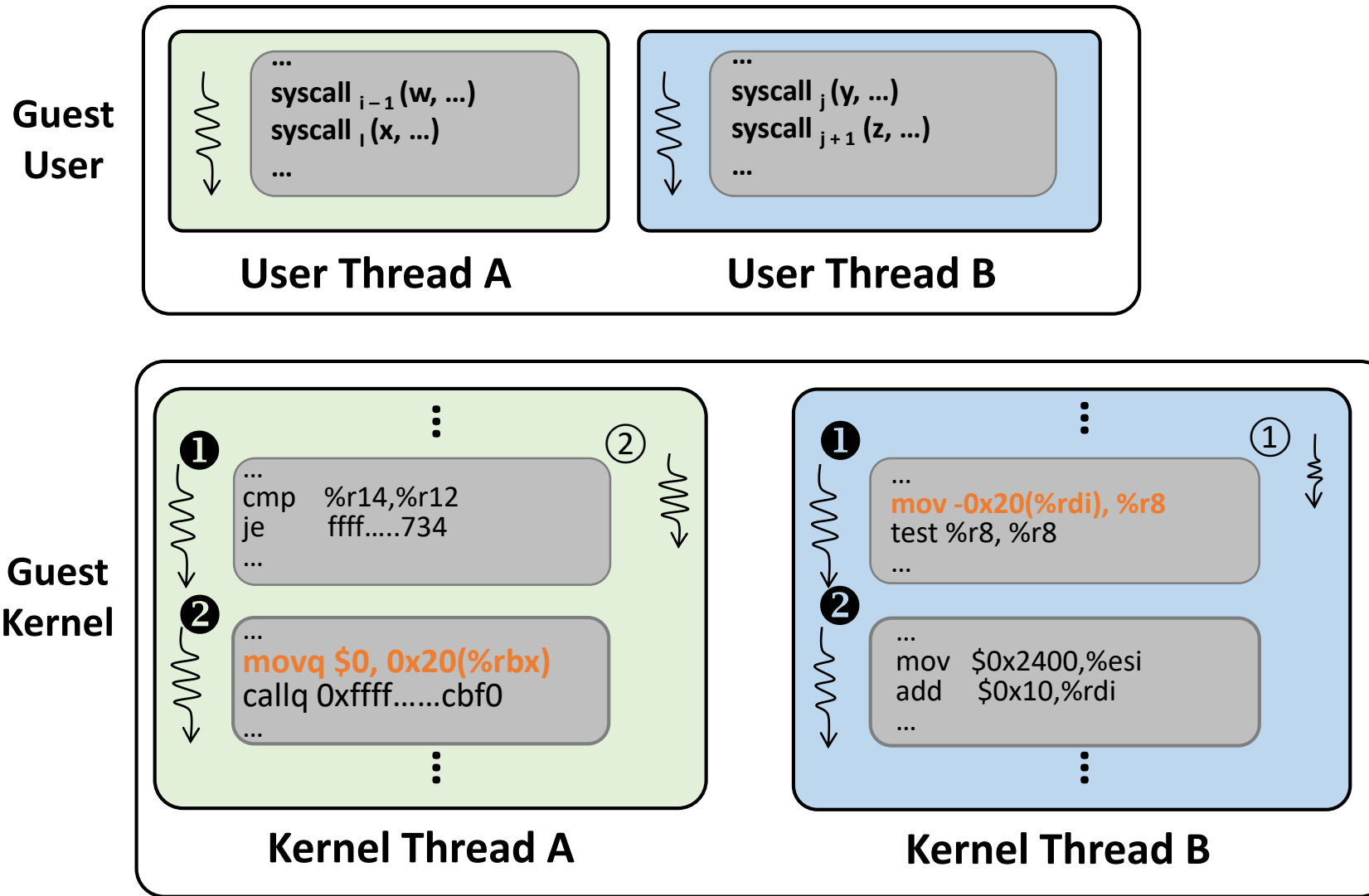
● : Syzkaller

○ : SKI



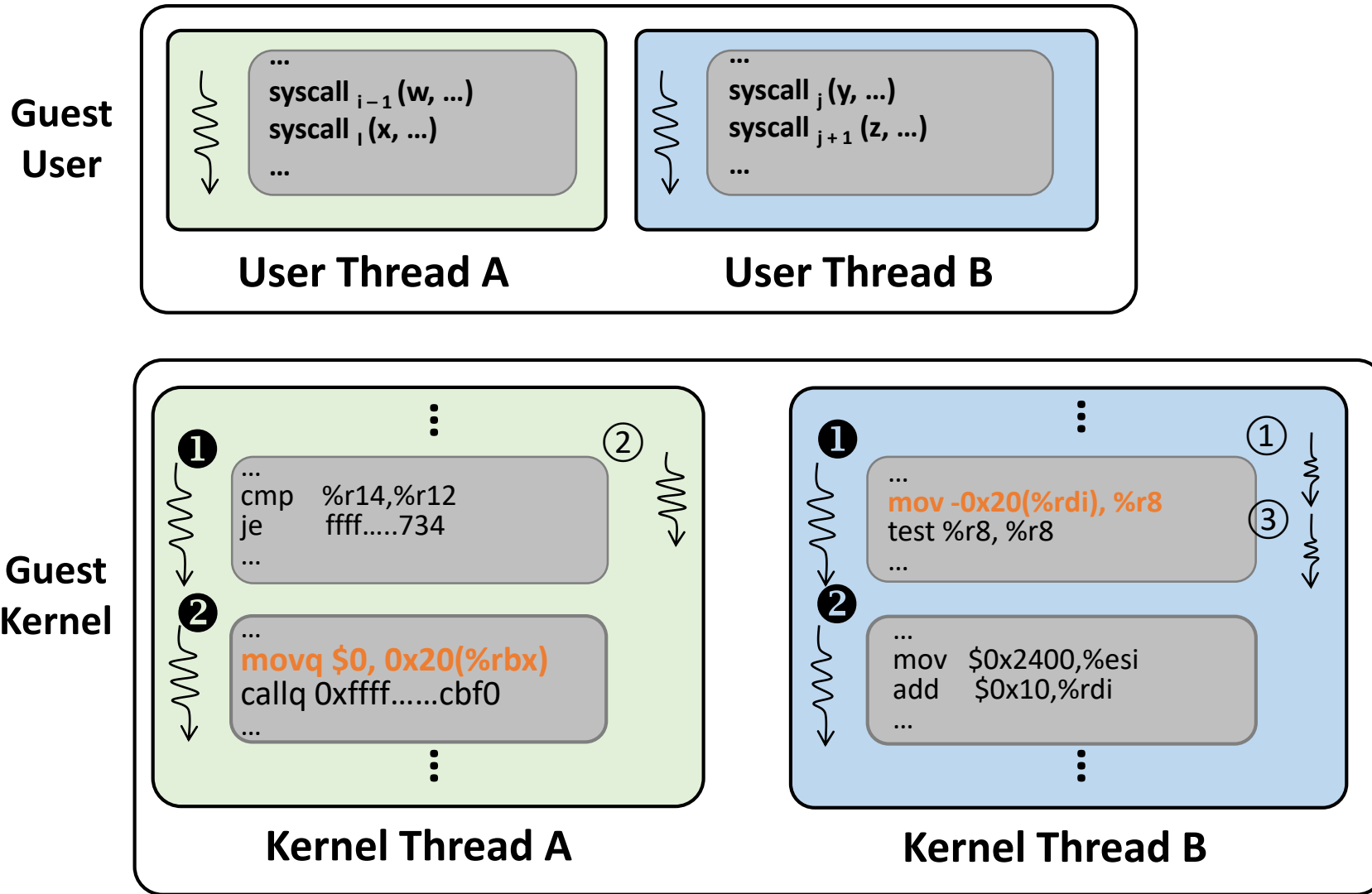
● : Syzkaller

○ : SKI



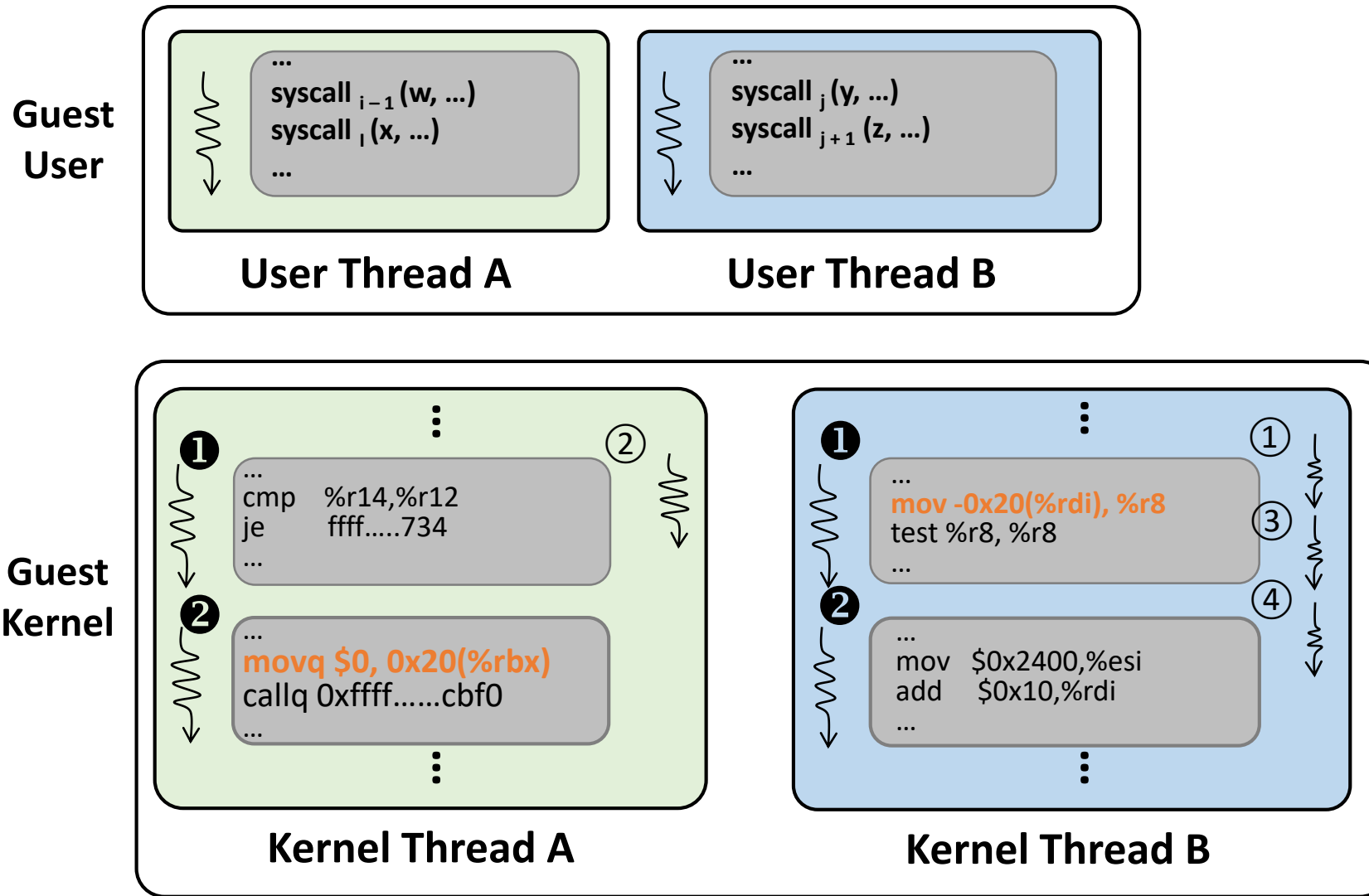
● : Syzkaller

○ : SKI



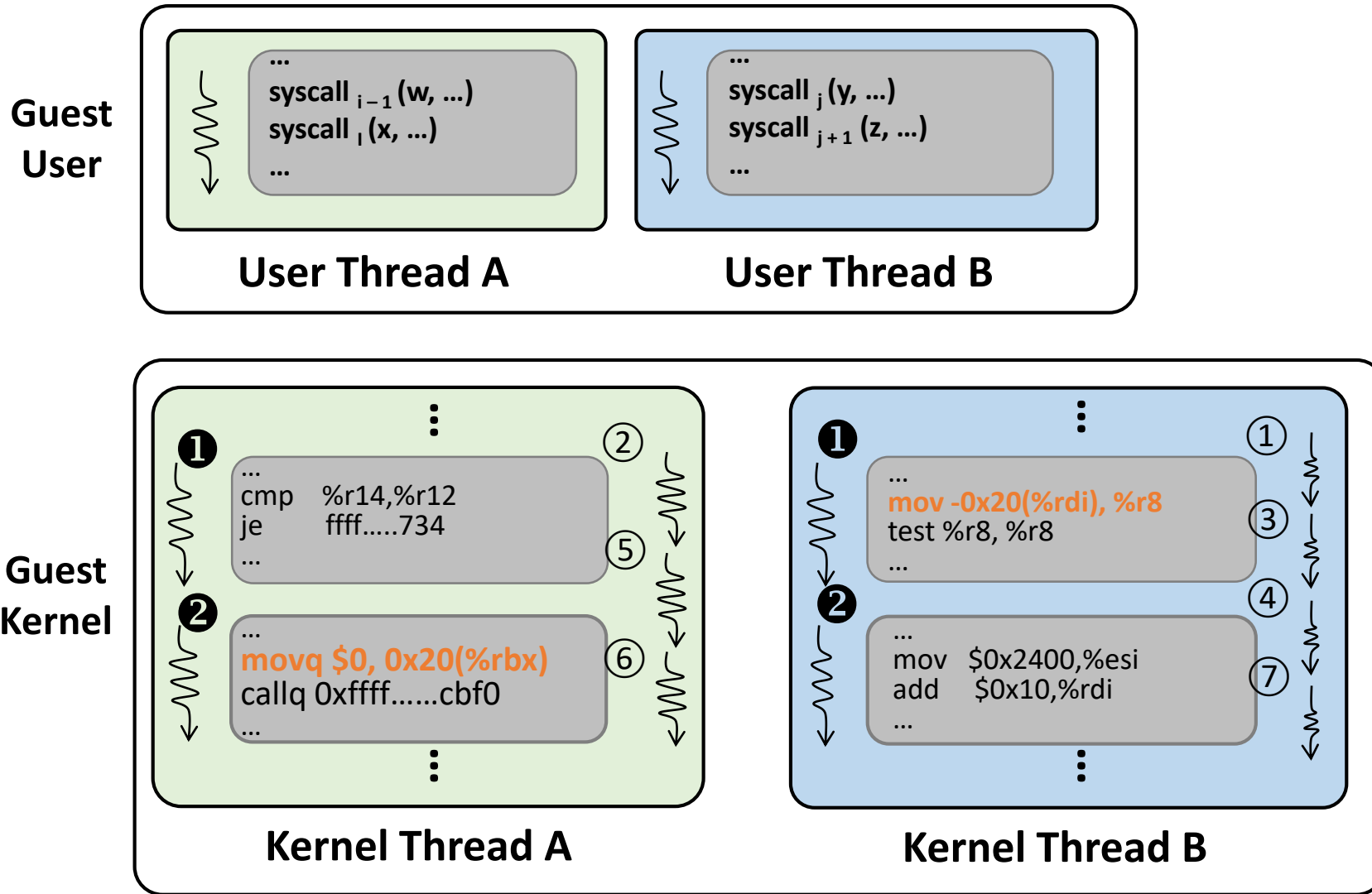
● : Syzkaller

○ : SKI



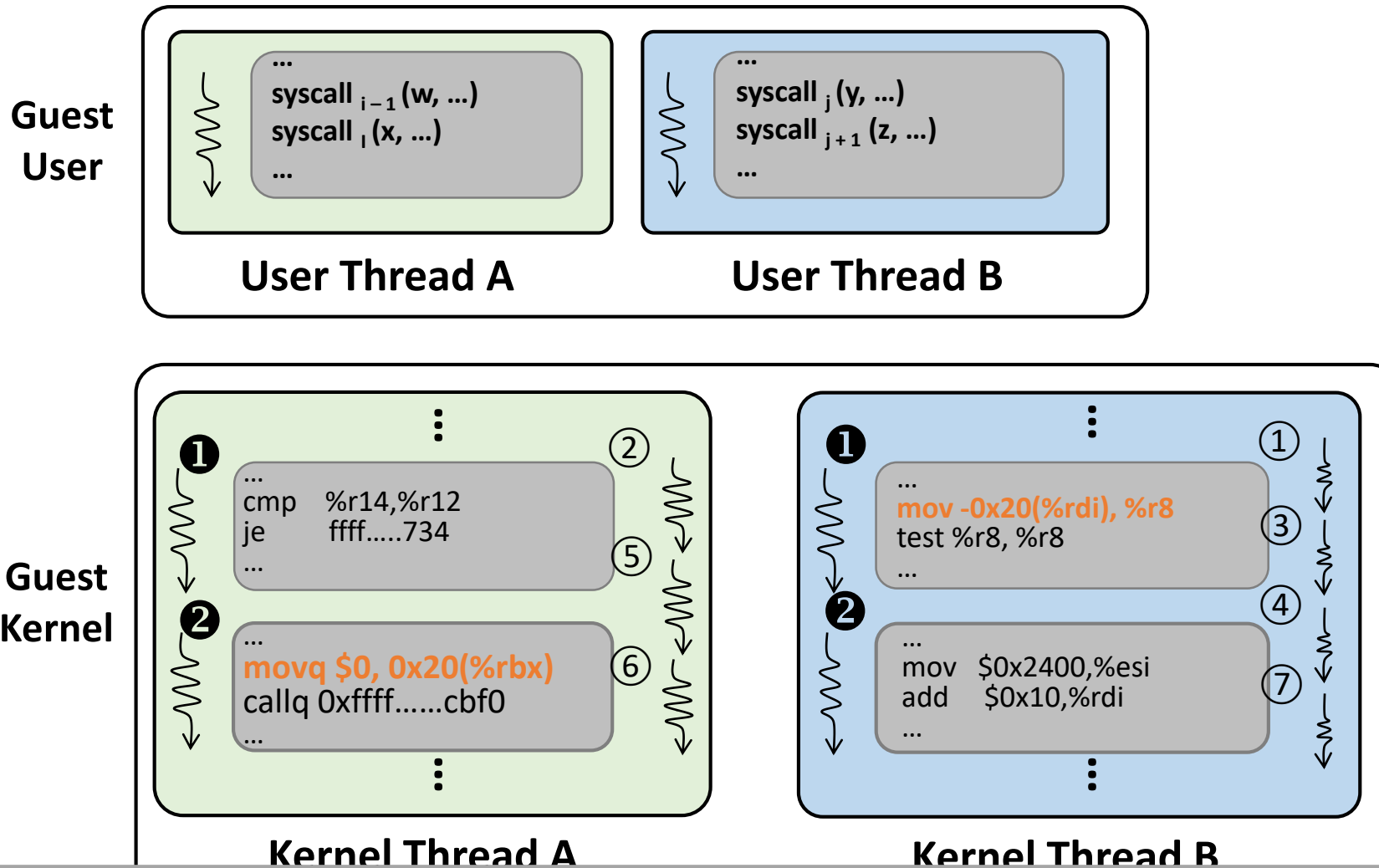
● : Syzkaller

○ : SKI



● : Syzkaller

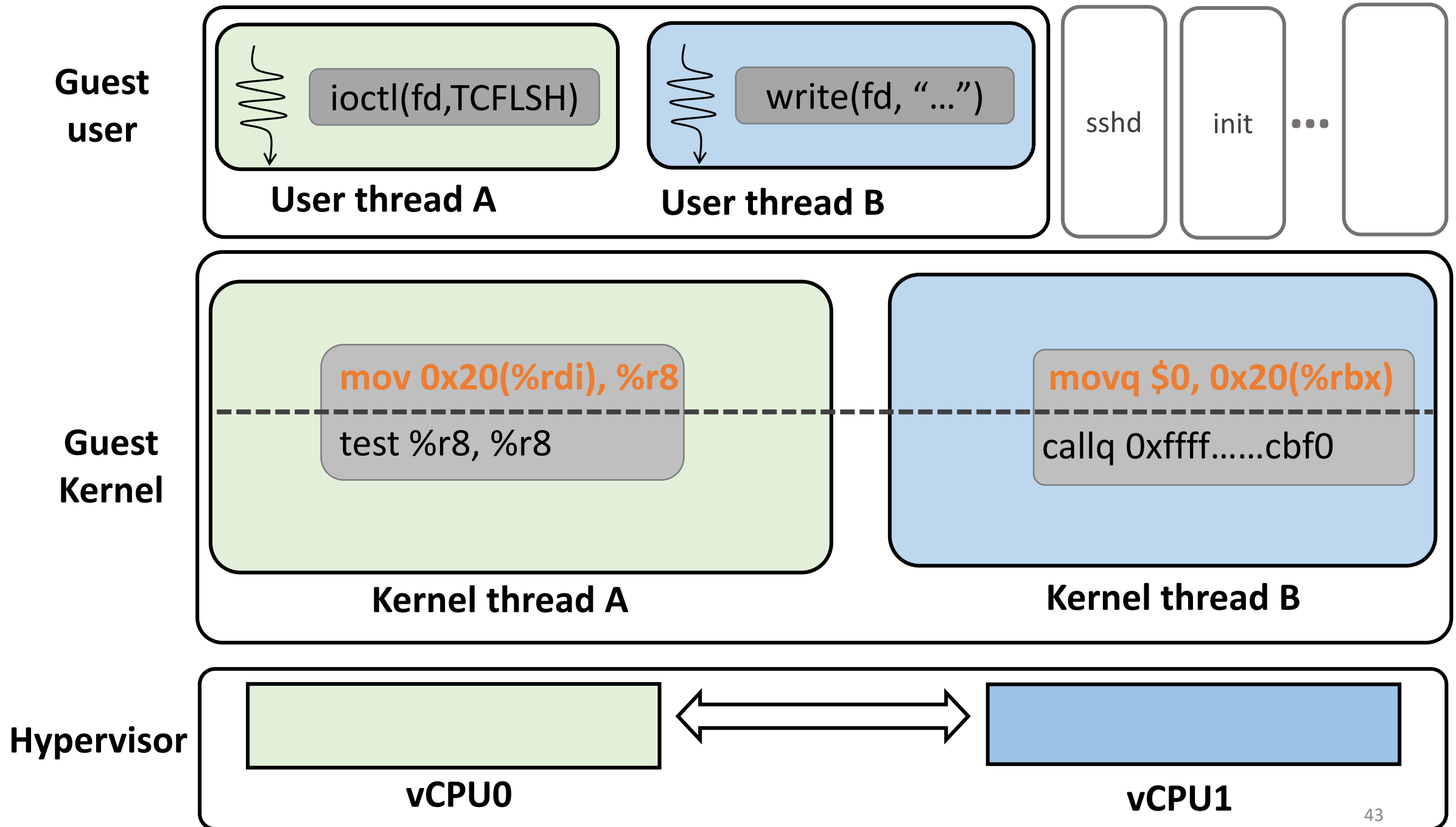
○ : SKI

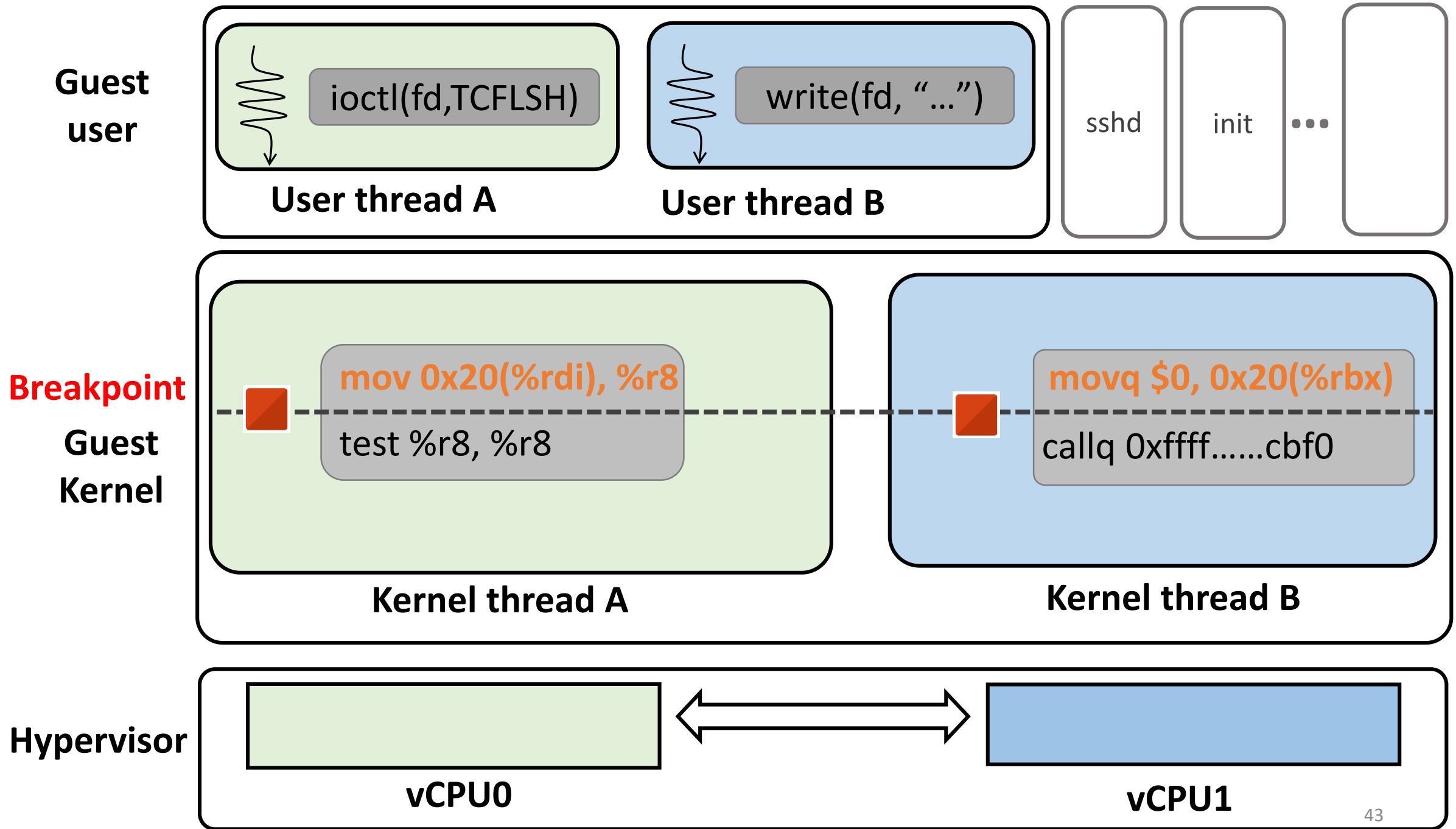


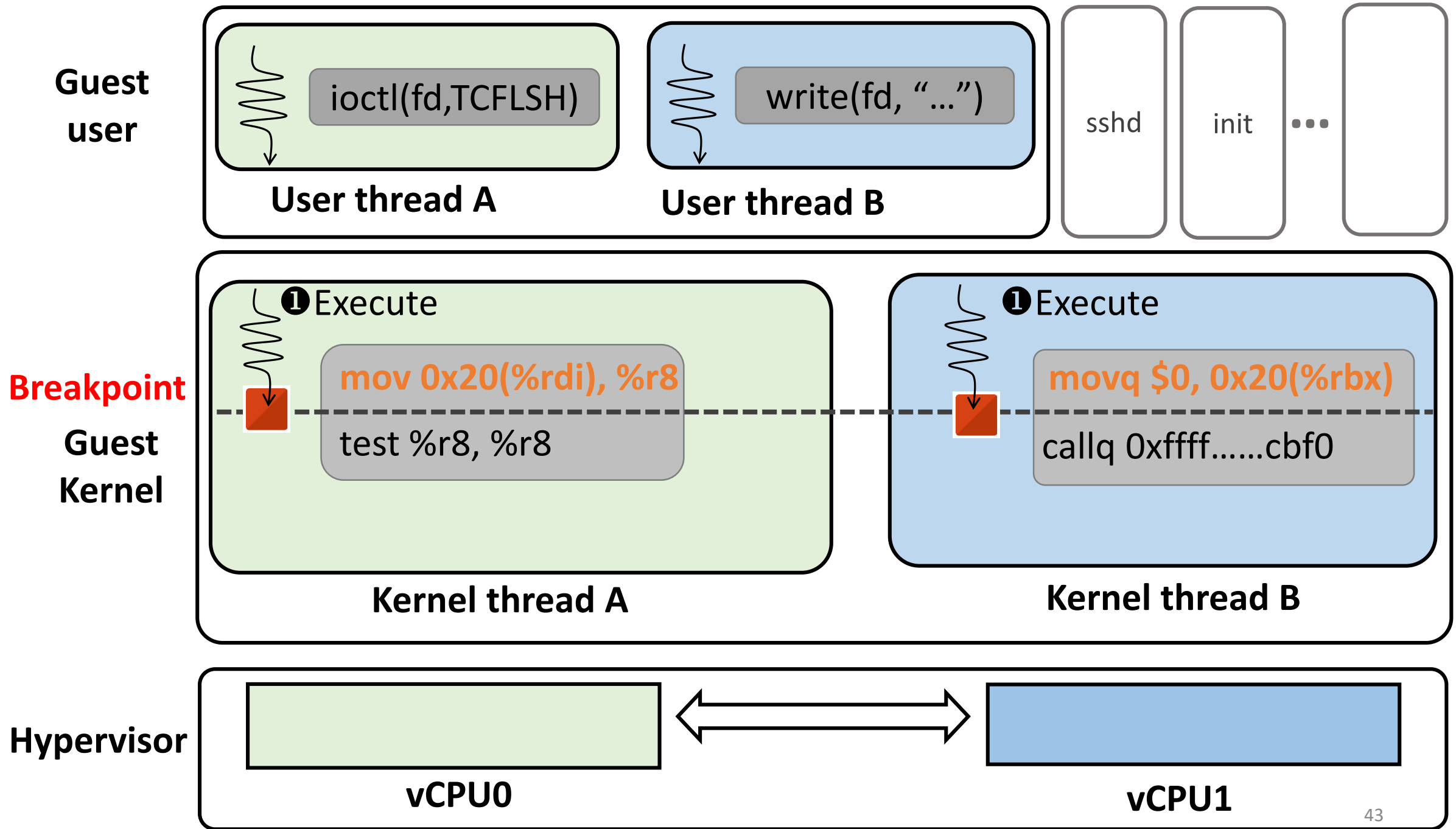
Very unlikely to be executed concurrently
→ Very unlikely to trigger a race!

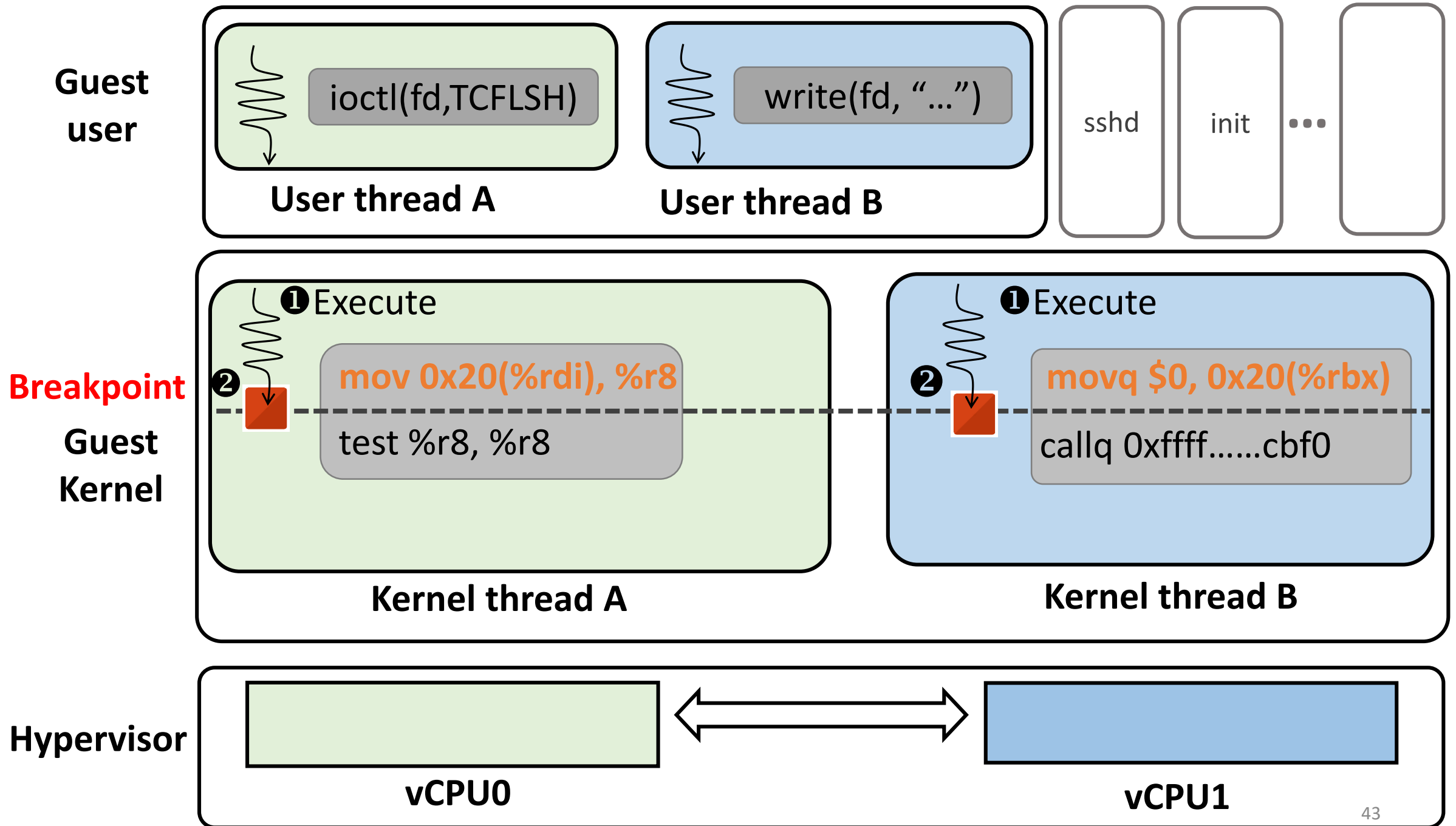
Finding Kernel Races through Fuzzing

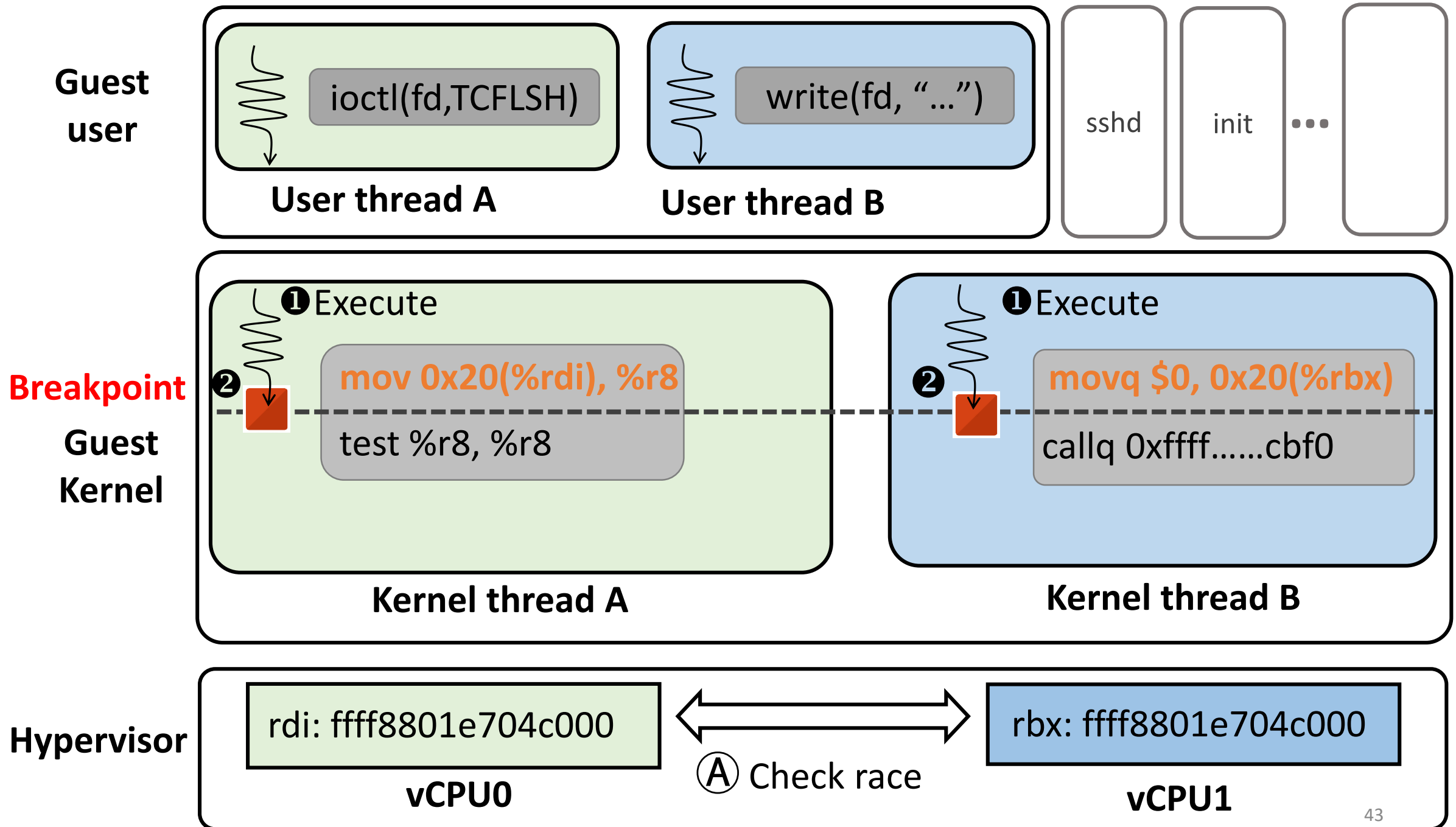
- Razer [S&P 19]
 - #1. Static analysis to find potential race spots
 - Points-to analysis over an entire kernel
 - #2. For each potential race spots, setup per-core breakpoint
 - Run the kernel on modified hypervisor (QEMU/KVM)
 - #3. Check if a race truly occurs
 - Prioritize a truly racing pair to be further fuzzed later

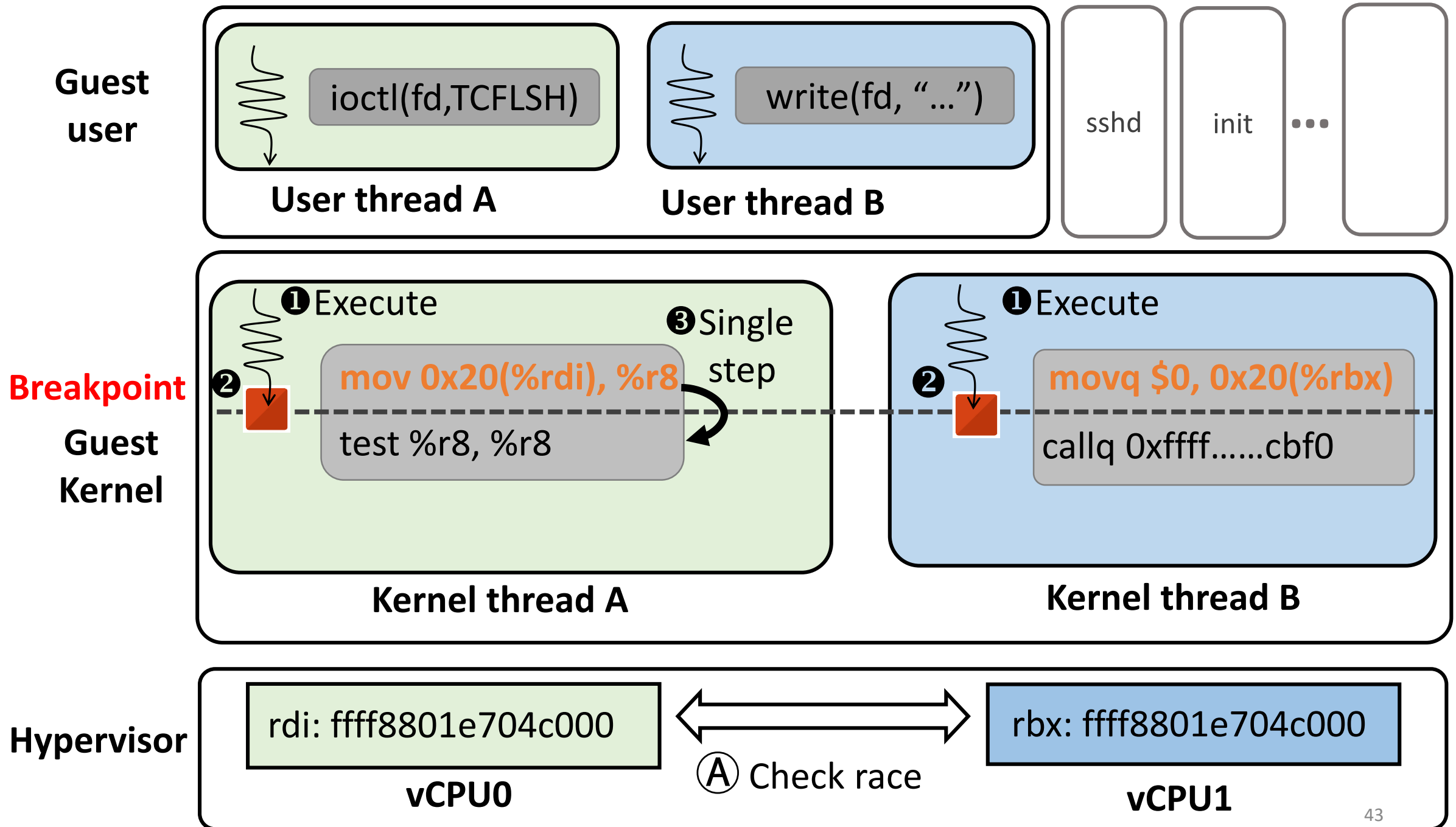


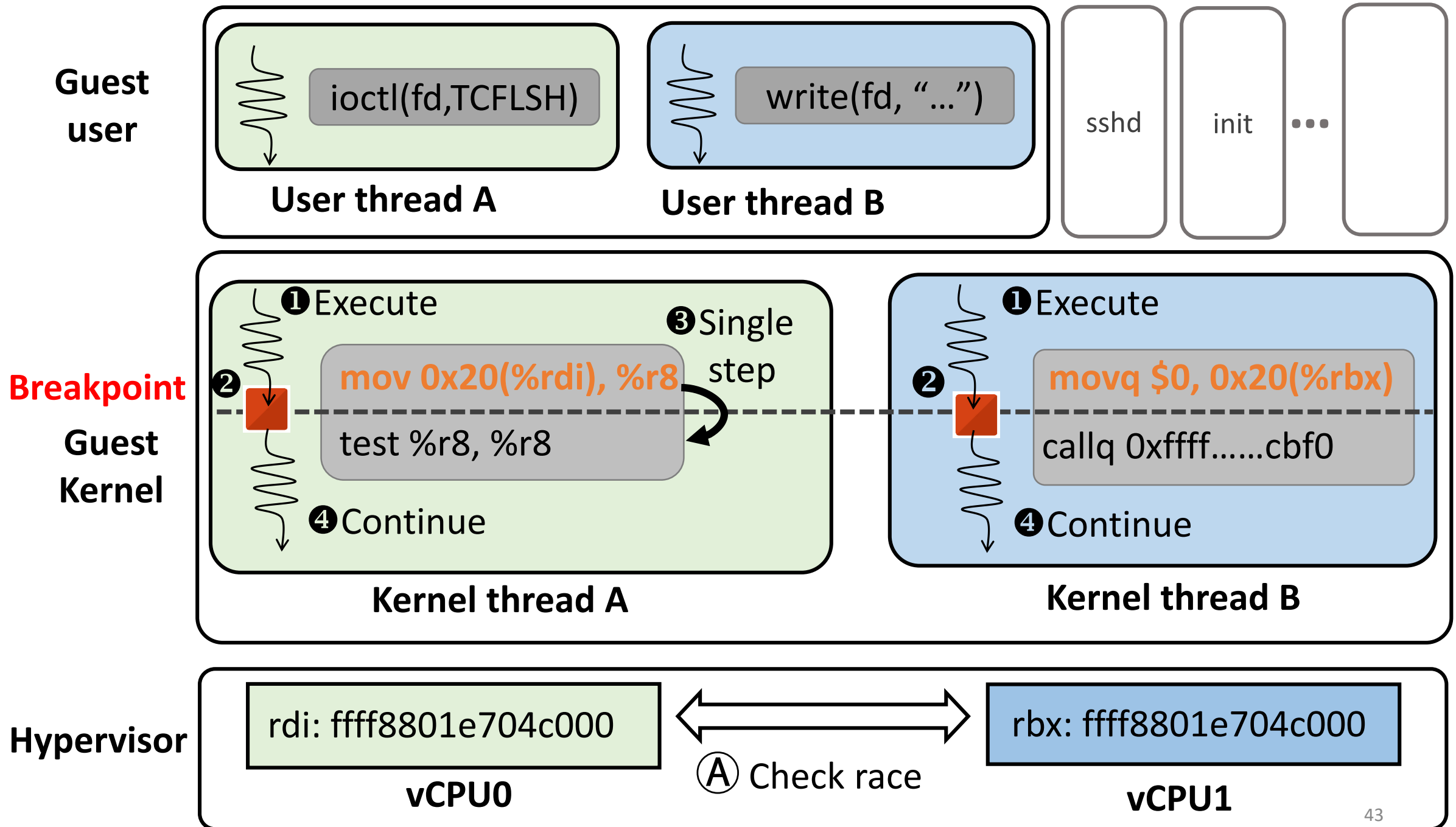












• Results

- 30 new races
- 16 are fixed

Kernel crash summary	Crash type	Kernel version	Kernel subsystem	Confirmed	Patch submitted	Fixed
KASAN: slab-out-of-bounds write in tty_insert_flip_string_flag	Use-After-Free	v4.8	drivers/tty/	✓	✓	✓
WARNING in __static_key_slow_dec	Reachable Warning	v4.8	net/	✓		
Kernel BUG at net/packet/af_packet.c:LINE!	Reachable Assertion	v4.16-rc3	net/packet/	✓	✓	✓
WARNING in refcount_dec	Reachable Warning	v4.16-rc3	net/packet/	✓	✓	✓
unable to handle kernel paging request in snd_seq_oss_readq_puts	Page Fault	v4.16	sound/core/seq/oss/	✓	✓	✓
KASAN: use-after-free Read in loopback_active_get	Use-After-Free	v4.16	sound/drivers/	✓	✓	✓
KASAN: null-ptr-deref Read in rds_ib_get_mr	Null ptr deref	v4.17-rc1	net/rdma/	✓ (assisted Syzkaller)	✓	✓
KASAN: null-ptr-deref Read in list_lru_del	Null ptr deref	v4.17-rc1	fs/			
BUG: unable to handle kernel NULL ptr dereference in corrupted	Null ptr deref	v4.17-rc1	net/sctp/			
KASAN: use-after-free Read in nd_jump_root	Use-After-Free	v4.17-rc1	fs/	✓	✓	✓
KASAN: use-after-free Read in link_path_walk	Use-After-Free	v4.17-rc1	fs/	✓	✓	✓
BUG: unable to handle kernel paging request in __inet_check_established	Page Fault	v4.17-rc1	net/ipv4/			
KASAN: null-ptr-deref Read in ata_pio_sector	Null ptr deref	v4.17-rc1	net/drivers/ata/			
WARNING in ip_recv_error	Reachable Warning	v4.17-rc1	net/	✓	✓	✓
WARNING in remove_proc_entry	Reachable Warning	v4.17-rc1	net/sunrpc/			
KASAN: null-ptr-deref Read in ip6gre_exit_batch_net	Null ptr deref	v4.17-rc1	net/ipv6/			
KASAN: slab-out-of-bounds Write in __register_sysctl_table	Heap overflow	v4.17-rc1	net/ipv6/			
KASAN: use-after-free Write in skb_release_data	Use-After-Free	v4.17-rc1	net/core/			
KASAN: invalid-free in ptlock_free	Double free	v4.17-rc1	mm/			
Kernel BUG at lib/list_debug.c:LINE!	Reachable Assertion	v4.17-rc1	drivers/infiniband/			
INFO: trying to register non-static key in __handle_mm_fault	Reachable INFO	v4.17-rc1	mm/			
KASAN: use-after-free Read in vhost_chr_write_iter	Use-After-Free	v4.17-rc1	drivers/vhost/	✓	✓	✓
BUG: soft lockup in vmemdup_user	Soft lockup	v4.17-rc1	net/			
KASAN: use-after-free Read in rds_tcp_accept_one	Use-After-Free	v4.17-rc1	net/rds/			
WARNING in sg_rq_end_io	Reachable Warning	v4.17-rc1	drivers/scsi/			
BUG: soft lockup in snd_virmidi_output_trigger	Soft lockup	v4.18-rc3	sound/core/seq/	✓ (assisted Syzkaller)	✓	✓
KASAN: null-ptr-deref Read in smc_ioctl	Null ptr deref	v4.18-rc3	net/smc/	✓	✓	✓
KASAN: null-ptr-deref Write in binderf_update_page_range	Null ptr deref	v4.18-rc3	drivers/android/	✓	✓	
WARNING in port_delete	Reachable Warning	v4.18-rc3	sound/core/seq/	✓ (assisted Syzkaller)		
KASAN: null-ptr-deref in inode_permission	Null ptr def	v4.18-rc3	fs/	✓	✓	✓

Discussion

- Semantic bugs
 - Need to automatically synthesize safety assertions
- Concurrency bugs
 - How to better reproduce races? (or find a root cause of races?)
 - How to better test?
- Memory bugs
 - Enforcing memory access permissions by firmware
 - Granularity: Page-granularity? Byte-granularity?
 - Metadata: MMU like designs? MPU like designs?
 - Principals: Bisected principals (e.g., kernel/user)? Multiple principals (e.g., multi-users)?

감사합니다.

이병영
서울대학교 전기정보공학부
byoungyoung@snu.ac.kr