SEGFUZZ: Segmentizing Thread Interleaving to Discover Kernel Concurrency Bugs through Fuzzing

Dae R. Jeong[1], Byoungyoung Lee[2], Insik Shin[1], Youngjin Kwon[1]

[1]Korea Advanced Institute of Science & Technology
[2]Seoul National University

Kernel concurrency bugs

- Kernel concurrency bugs manifest depending on thread interleavings

Kernel concurrency bugs

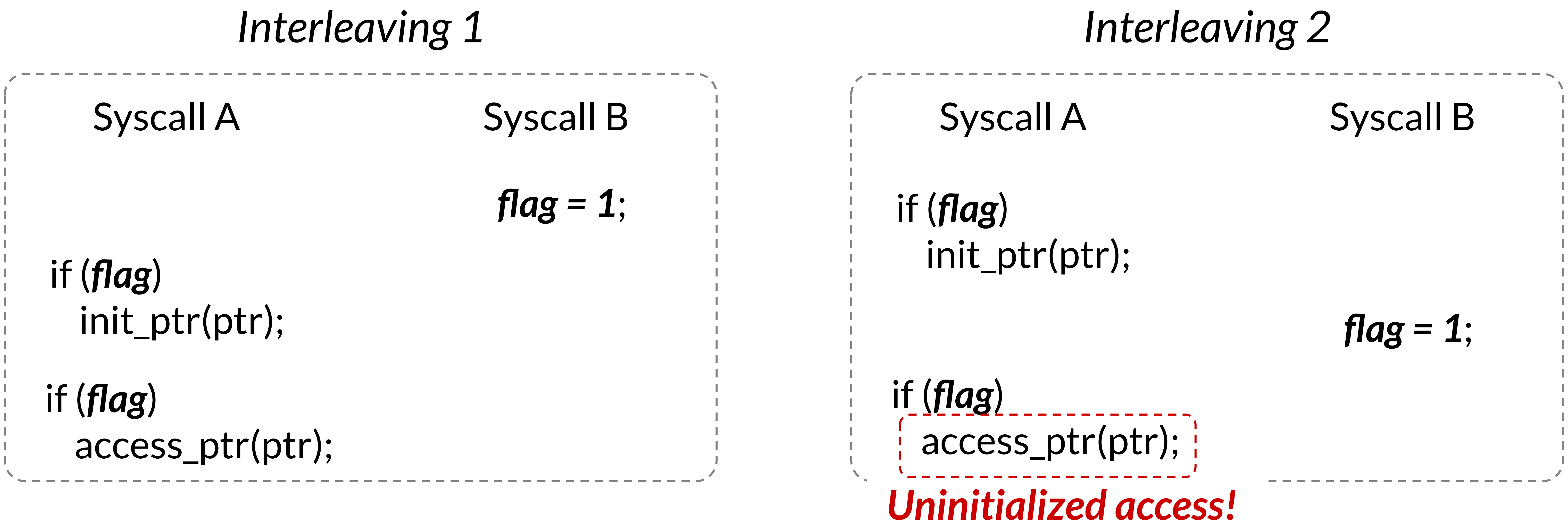
- Kernel concurrency bugs manifest depending on thread interleavings

Interleaving 1

```
   Syscall A                    Syscall B

                                 flag = 1;

 if (flag)
    init_ptr(ptr);

 if (flag)
    access_ptr(ptr);
```

Kernel concurrency bugs

- Kernel concurrency bugs manifest depending on thread interleavings



Interleaving 1

```
Syscall A                    Syscall B

                             flag = 1;

 if (flag)
    init_ptr(ptr);

if (flag)
   access_ptr(ptr);
```

Interleaving 2

```
Syscall A                    Syscall B

 if (flag)
    init_ptr(ptr);

                             flag = 1;

if (flag)
   access_ptr(ptr);
```

Uninitialized access!

Fuzzing

- Fuzzing explores the search space of the program by running random inputs
  - Conventionally focusing on exploring ***execution paths***
    - Symbolic/concolic execution, static analysis, …

Fuzzing

- Fuzzing explores the search space of the program by running random inputs
  - Conventionally focusing on exploring ***execution paths***
    - Symbolic/concolic execution, static analysis, …
- Recent approaches to identify concurrency bugs
  - Exploring ***execution path & thread interleavings***
    - *Razzer* [S&P'19], *Krace*[S&P'20], *Snowboard*[SOSP'21], *Conzzer*[NDSS'22], …
  - ***Controlling thread interleavings*** by overriding the kernel scheduler

Coverage-guided fuzzing

- ***Coverage metric***
  - Expressing the search space of the program
  - Guiding the generation of new test cases

Coverage-guided fuzzing

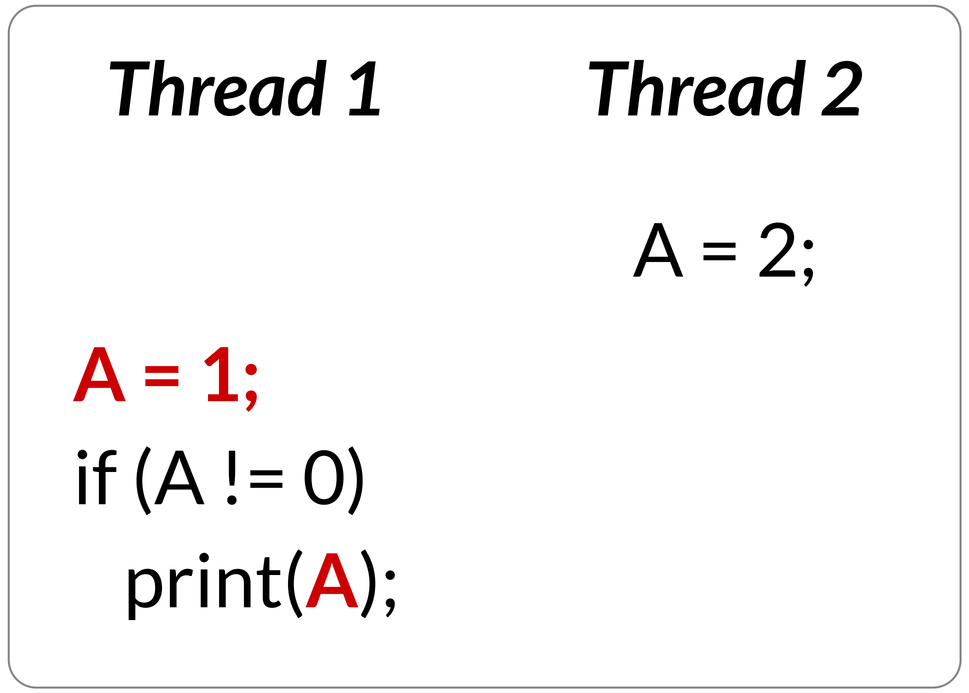
- ***Coverage metric***
  - Expressing the search space of the program
  - Guiding the generation of new test cases

- ***Code coverage***
  - Expressing the search space of ***execution paths***
  - Ex) Branch coverage

Coverage-guided fuzzing

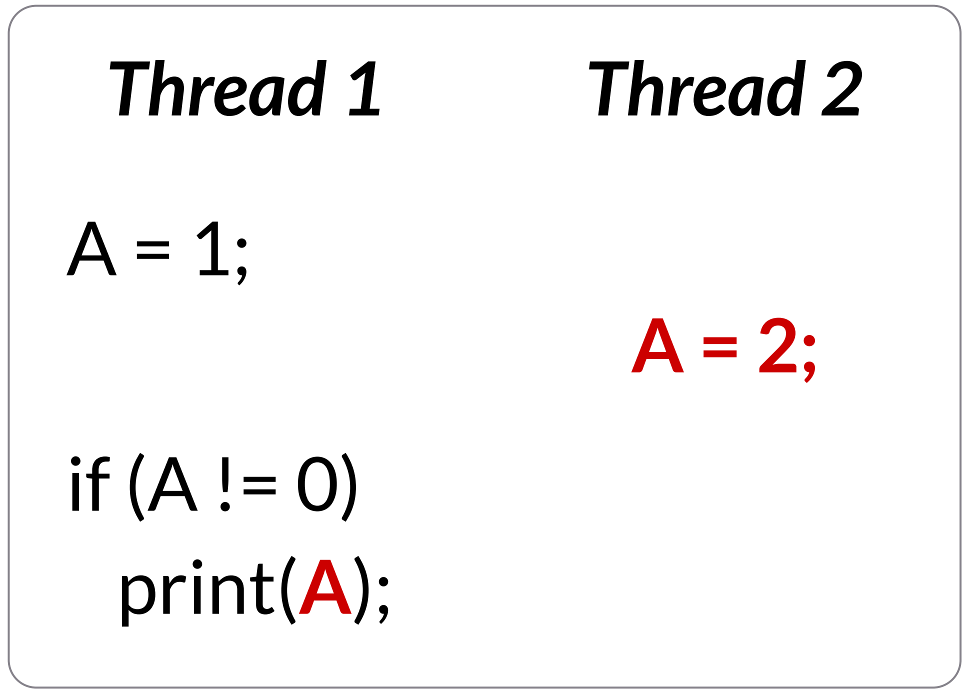
- ***Code coverage***
  - Limited in expressing the search space of thread interleavings

Interleaving 1



Thread 1	***Thread 2***
	A = 2;
A = 1;	
if (A != 0)	
print(**A**);	

Interleaving 2



Thread 1	***Thread 2***
A = 1;	
	A = 2;
if (A != 0)	
print(**A**);	

The same branch coverage but different outcomes

Coverage-guided fuzzing

- ***Coverage metric***
  - Expressing the search space of the program
  - Guiding the generation of new test cases

- ***Code coverage***
  - Expressing the search space for ***execution paths***
  - Ex) Branch coverage

- ***Interleaving coverage***
  - Expressing the search space for ***thread interleavings***
  - ***Not well-studied area***

Coverage-guided fuzzing

- ***Coverage metric***
  - Expressing the search space of the program
  - Guiding the generation of new test cases

We want to design and utilize interleaving coverage

- ***Interleaving coverage***
  - Expressing the search space for ***thread interleavings***
  - ***Not well-studied area***

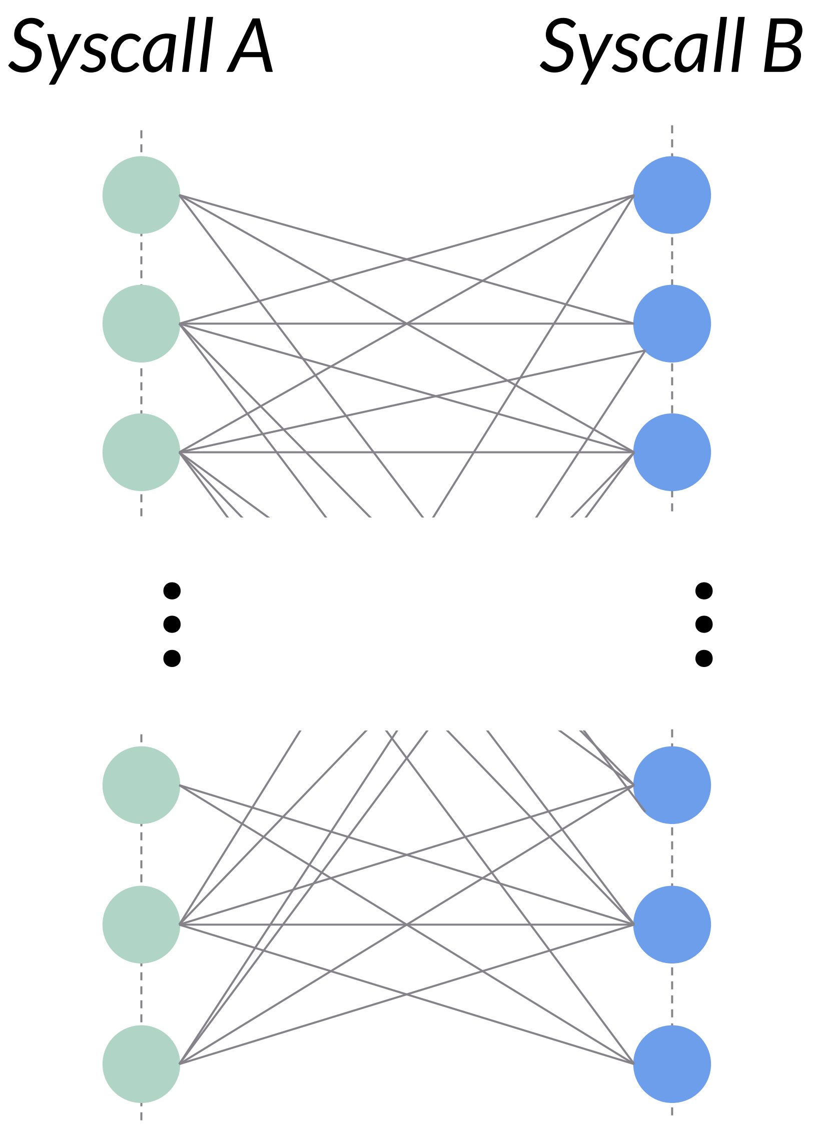
Coverage metric for thread interleavings

- ***Challenge***
  - A large search space of thread interleavings

Coverage metric for thread interleavings

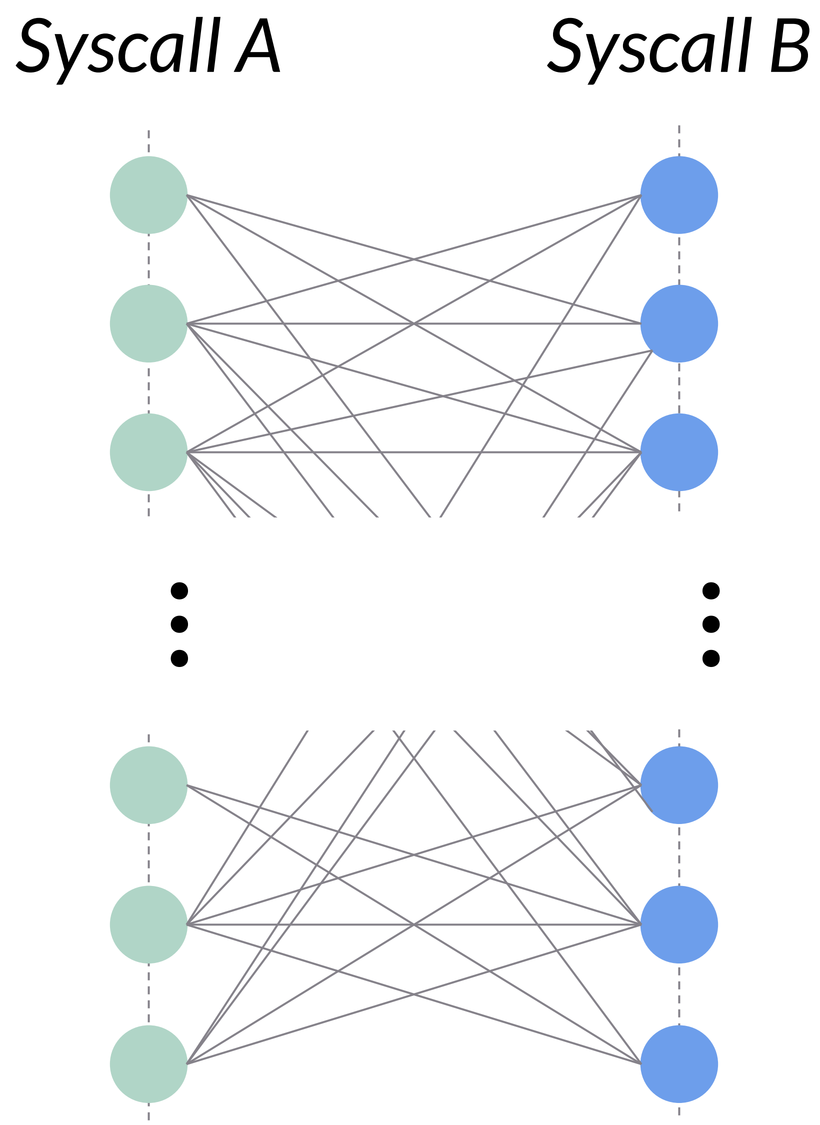
- ***Challenge***
  - A large search space of thread interleavings

100 inst. for each syscall



Coverage metric for thread interleavings

- ***Challenge***
  - A large search space of thread interleavings

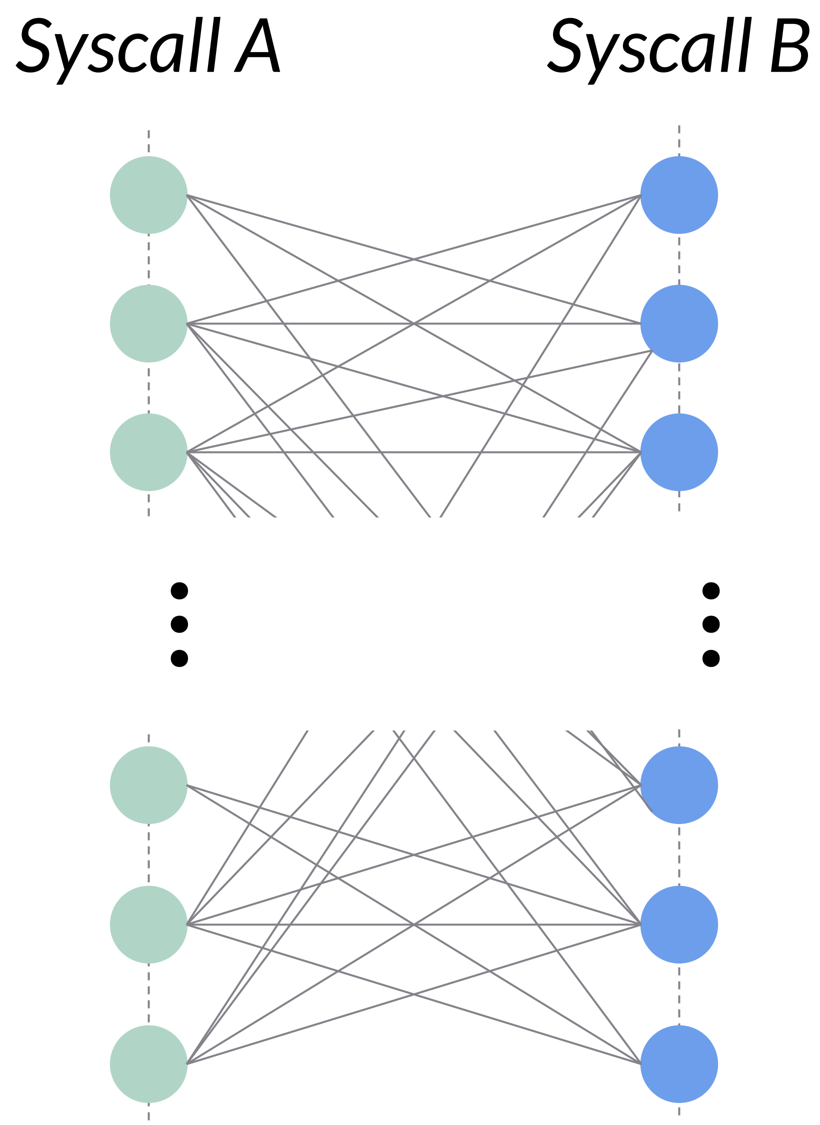


100 inst. for each syscall

There are a huge number of interleavings (e.g., more than 10^{58})

Coverage metric for thread interleavings

- ***Challenge***
  - A large search space of thread interleavings



100 inst. for each syscall

There are a huge number of interleavings (e.g., more than 10^{58})

Only a small number of interleavings cause a concurrency bug.

Coverage metric for thread interleavings

- ***Challenge***
  - A large search space of thread interleavings

- Our interleaving coverage should

  1) reduce the search space

  2) capture *“interesting”* interleavings

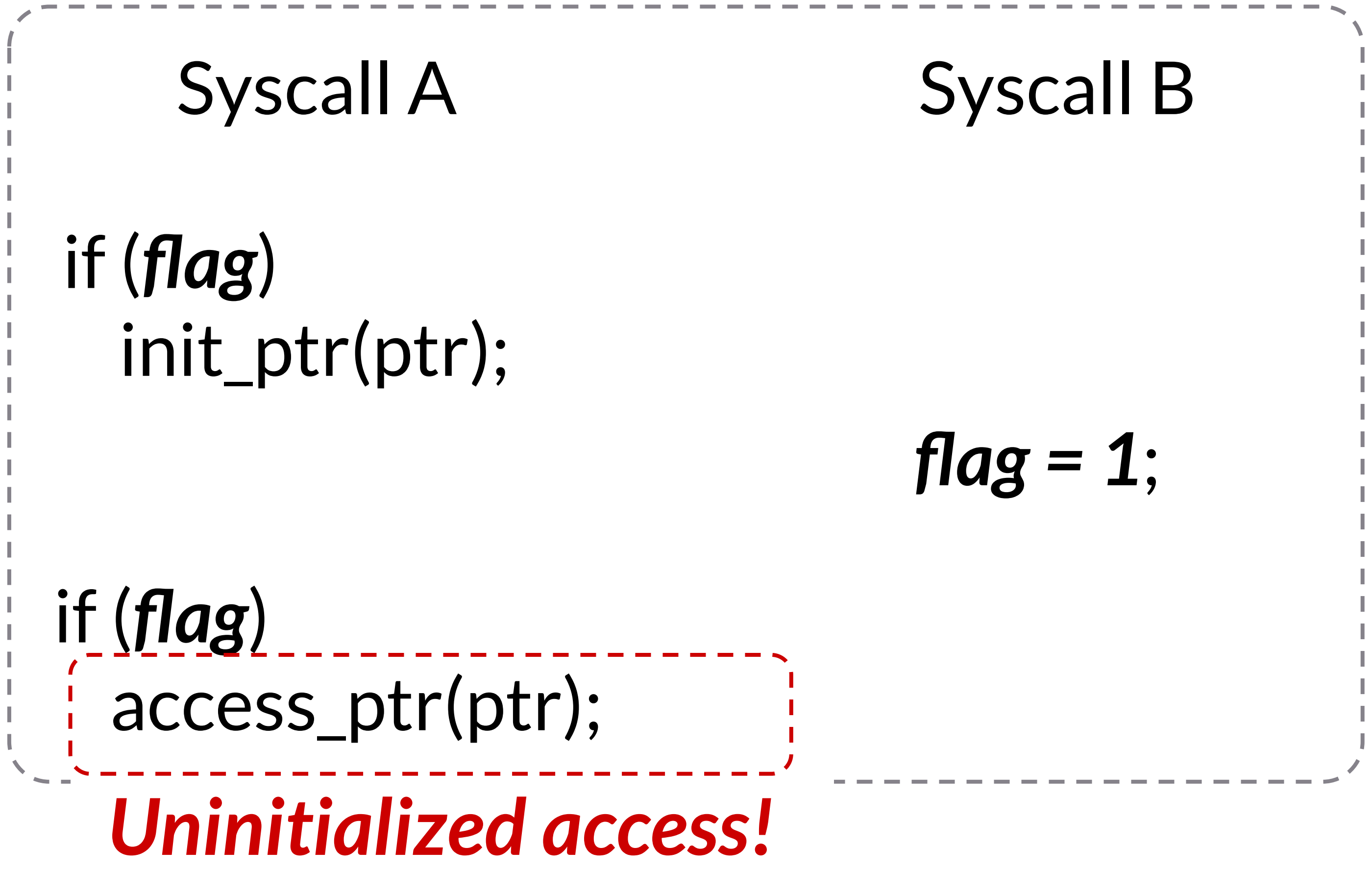
Characteristic of concurrency bugs

- ***Observation from a previous study*** [1]
  - Most of concurrency bugs (97 out of 105) manifest depending on the execution order of ***at most four memory accesses***

[1] Lu, Shan, et al. "Learning from mistakes: a comprehensive study on real world concurrency bug characteristics." *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*. 2008.

Characteristic of concurrency bugs

- ***Observation from a previous study*** [1]
  - Most of concurrency bugs (97 out of 105) manifest depending on the execution order of ***at most four memory accesses***



```
Syscall A                          Syscall B

if (flag)
   init_ptr(ptr);
                                   flag = 1;

if (flag)
   access_ptr(ptr);
```

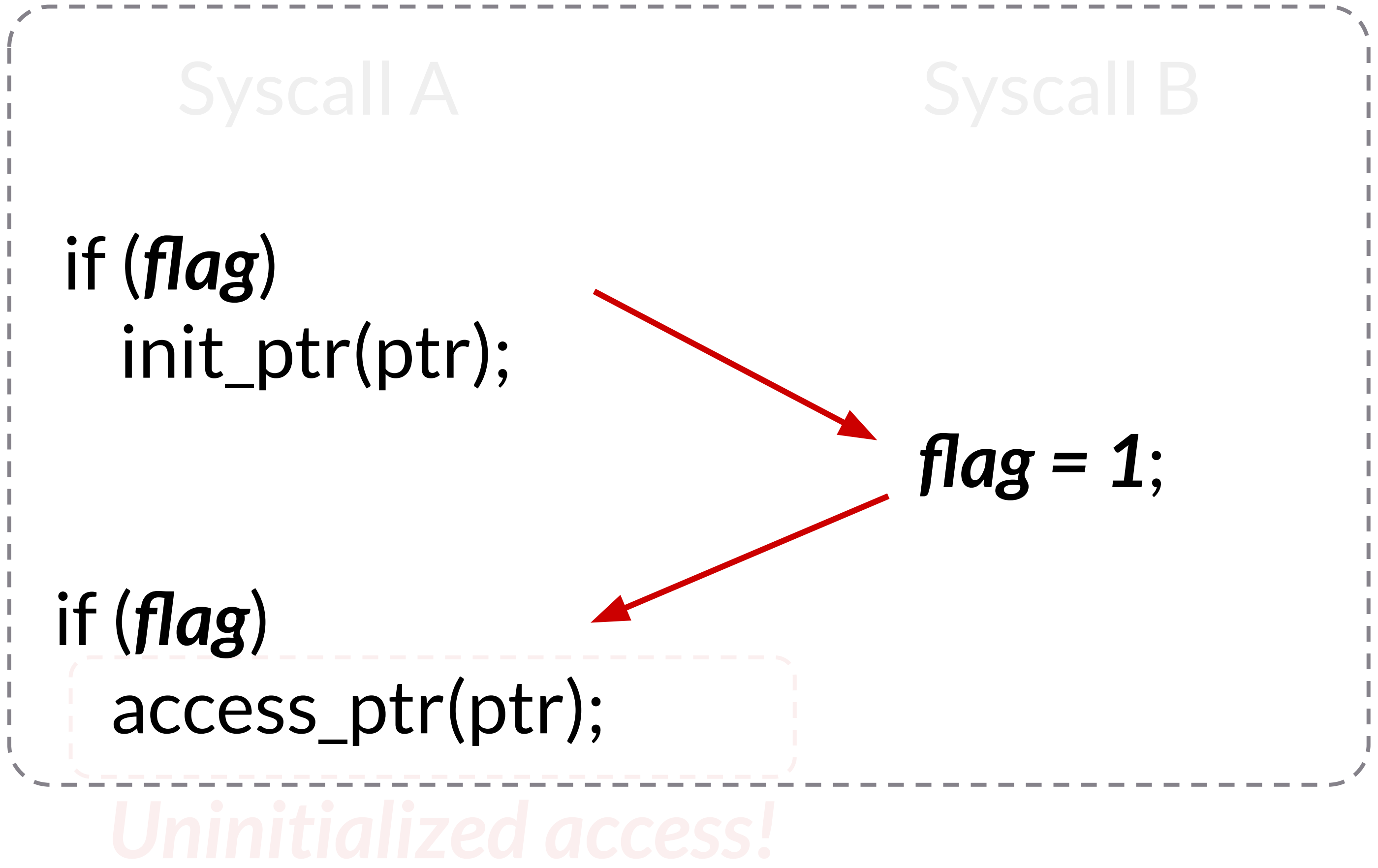
Uninitialized access!

[1] Lu, Shan, et al. "Learning from mistakes: a comprehensive study on real world concurrency bug characteristics." *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*. 2008.

Characteristic of concurrency bugs

- ***Observation from a previous study*** [1]
  - Most of concurrency bugs (97 out of 105) manifest depending on the execution order of ***at most four memory accesses***

*The uninitialized access bug manifests depending **only on three instructions***



Syscall A

Syscall B

```
if (flag)
   init_ptr(ptr);

                         flag = 1;

if (flag)
   access_ptr(ptr);
```

Uninitialized access!

[1] Lu, Shan, et al. "Learning from mistakes: a comprehensive study on real world concurrency bug characteristics." *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*. 2008.

Characteristic of concurrency bugs

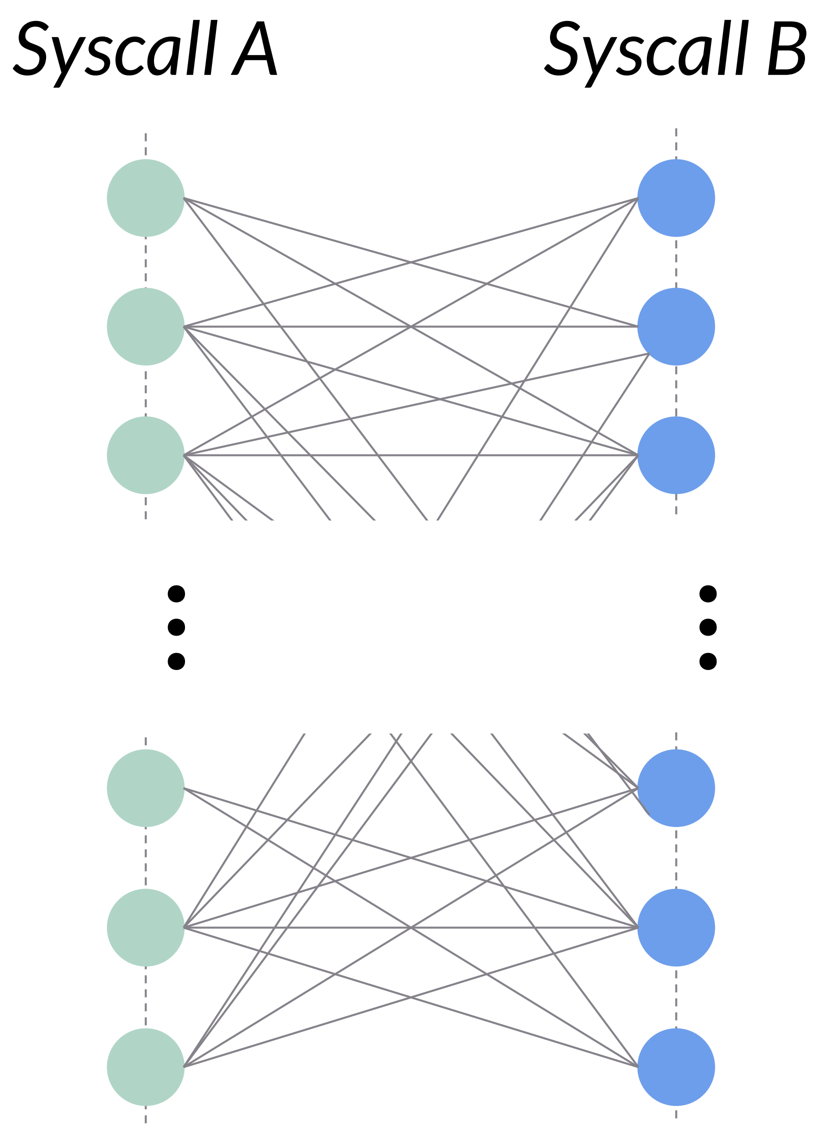
- ***Observation from a previous study*** [1]
  - Most of concurrency bugs (97 out of 105) manifest depending on the execution order of ***at most four memory accesses***

- ***Our strategy: Segmentizing thread interleaving***
  - Decomposing thread interleaving into small interleaving segments that consists of at most four memory accesses

[1] Lu, Shan, et al. "Learning from mistakes: a comprehensive study on real world concurrency bug characteristics." *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*. 2008.

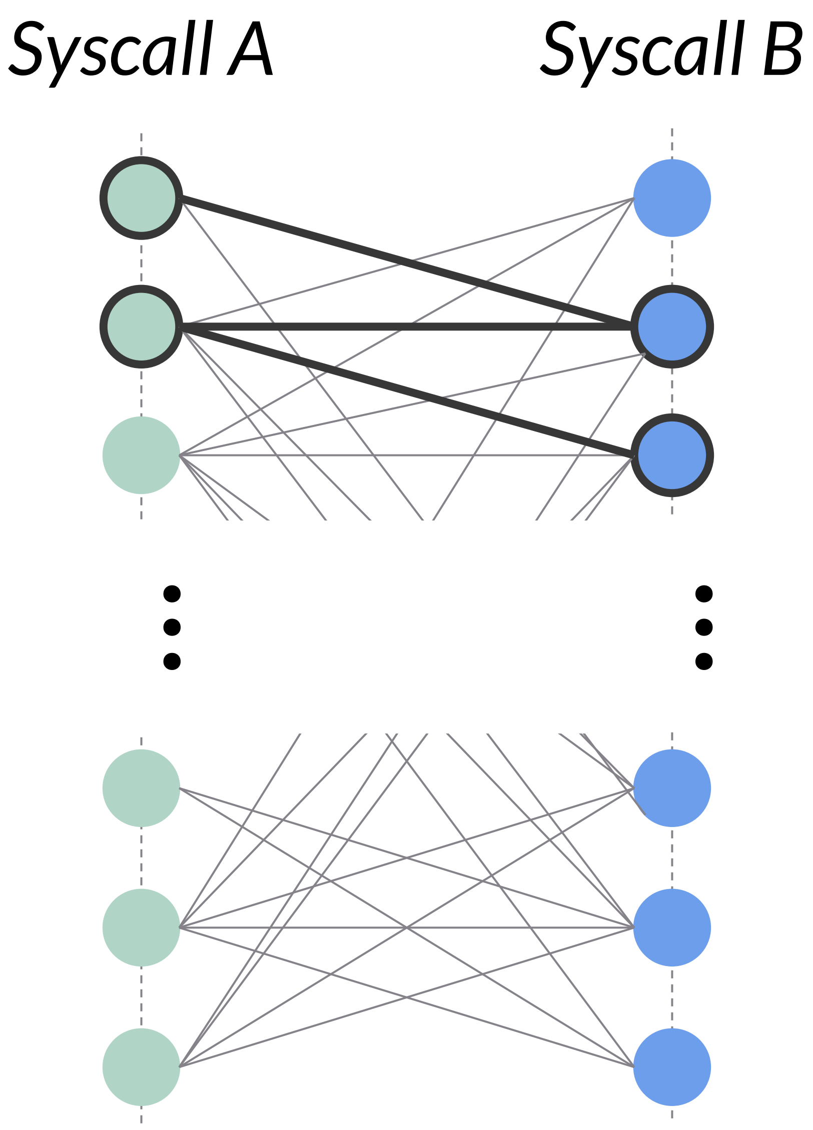
Key idea: decomposing thread interleaving

100 inst. for each syscall

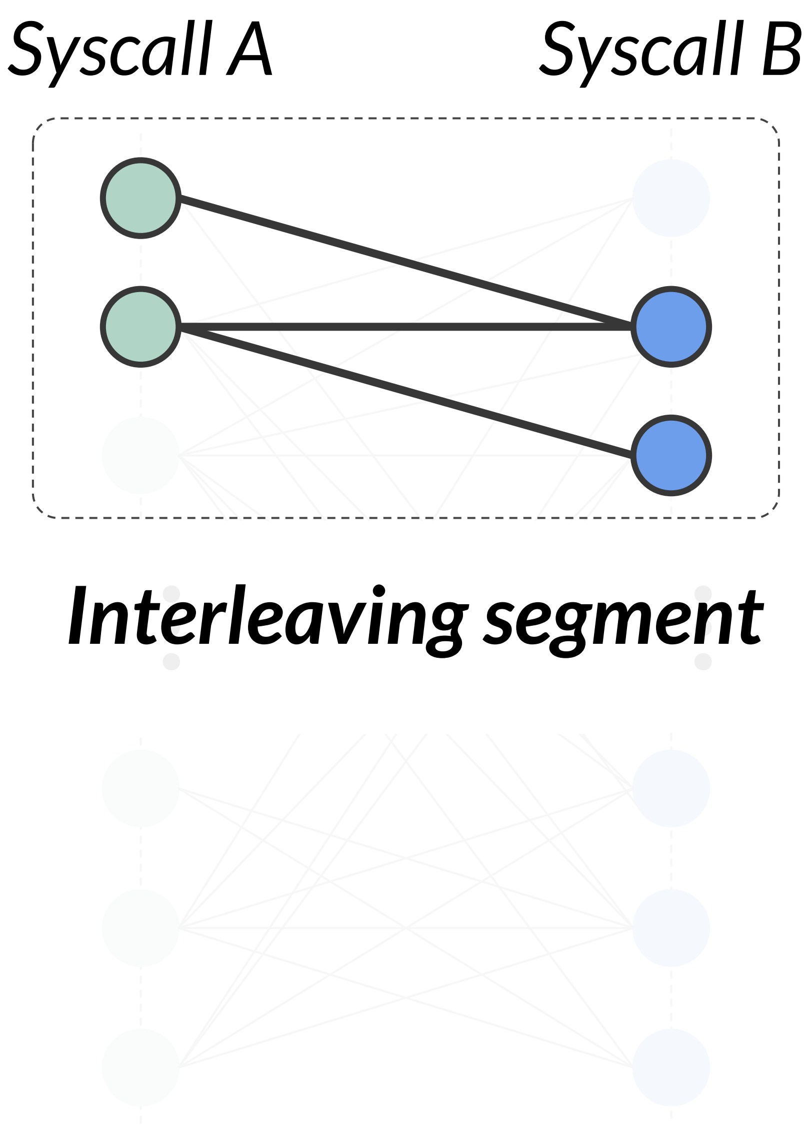


Key idea: decomposing thread interleaving

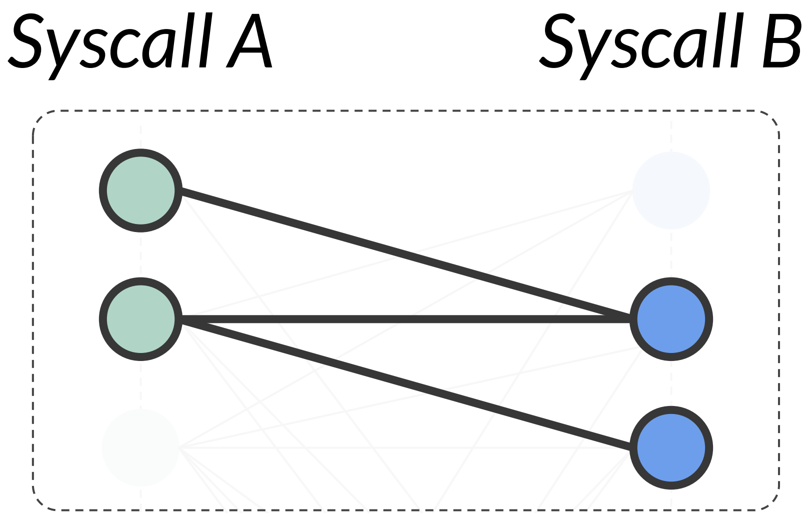
100 inst. for each syscall



Key idea: decomposing thread interleaving



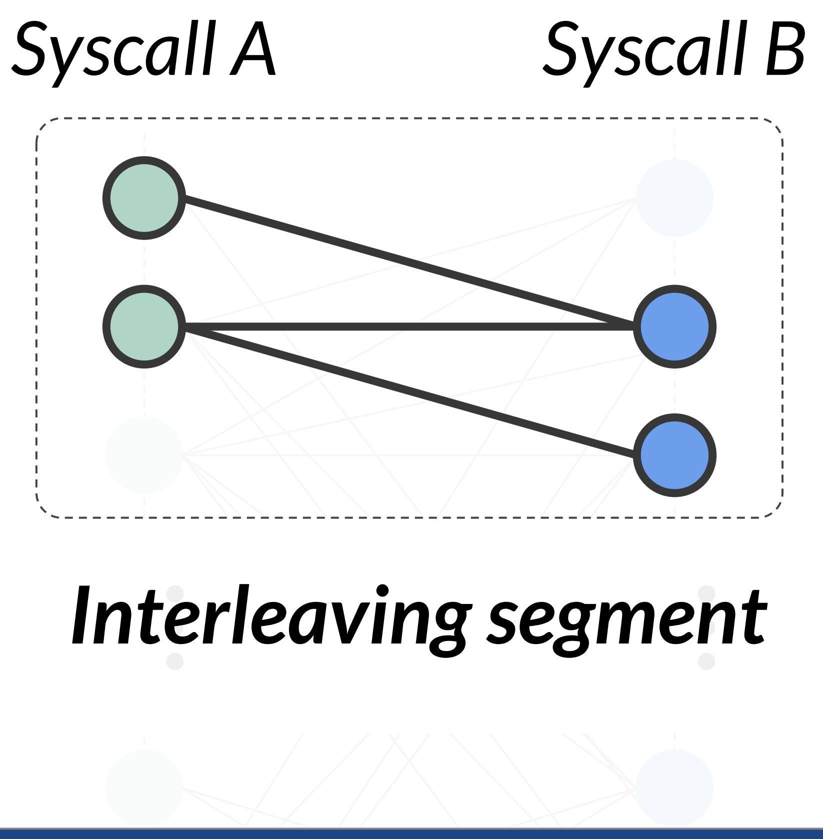
Key idea: decomposing thread interleaving



Interleaving segment

- ***Benefits***
  - Reducing the search space
  - Tracking interesting interleavings

Key idea: decomposing thread interleaving

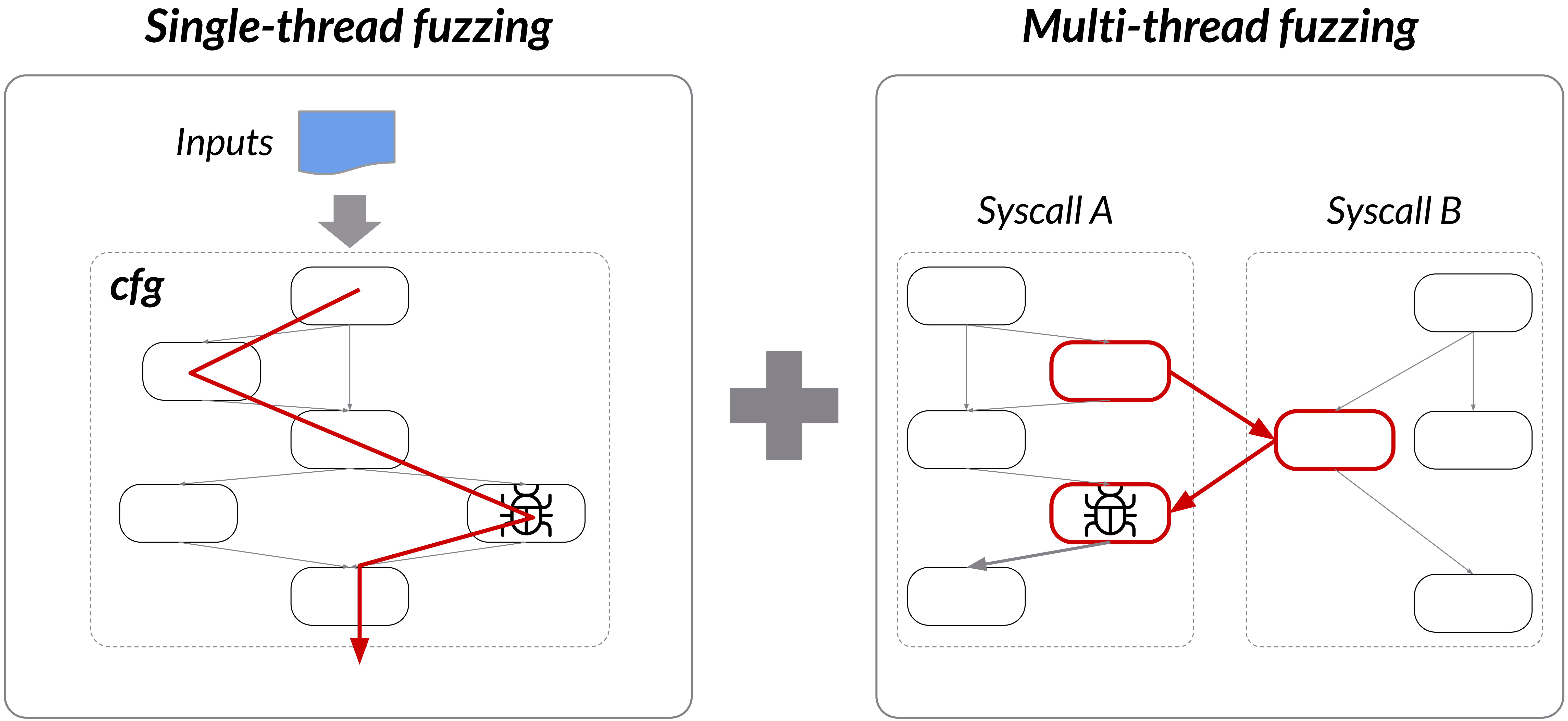


Interleaving segment

Our interleaving coverage is based on interleaving segments

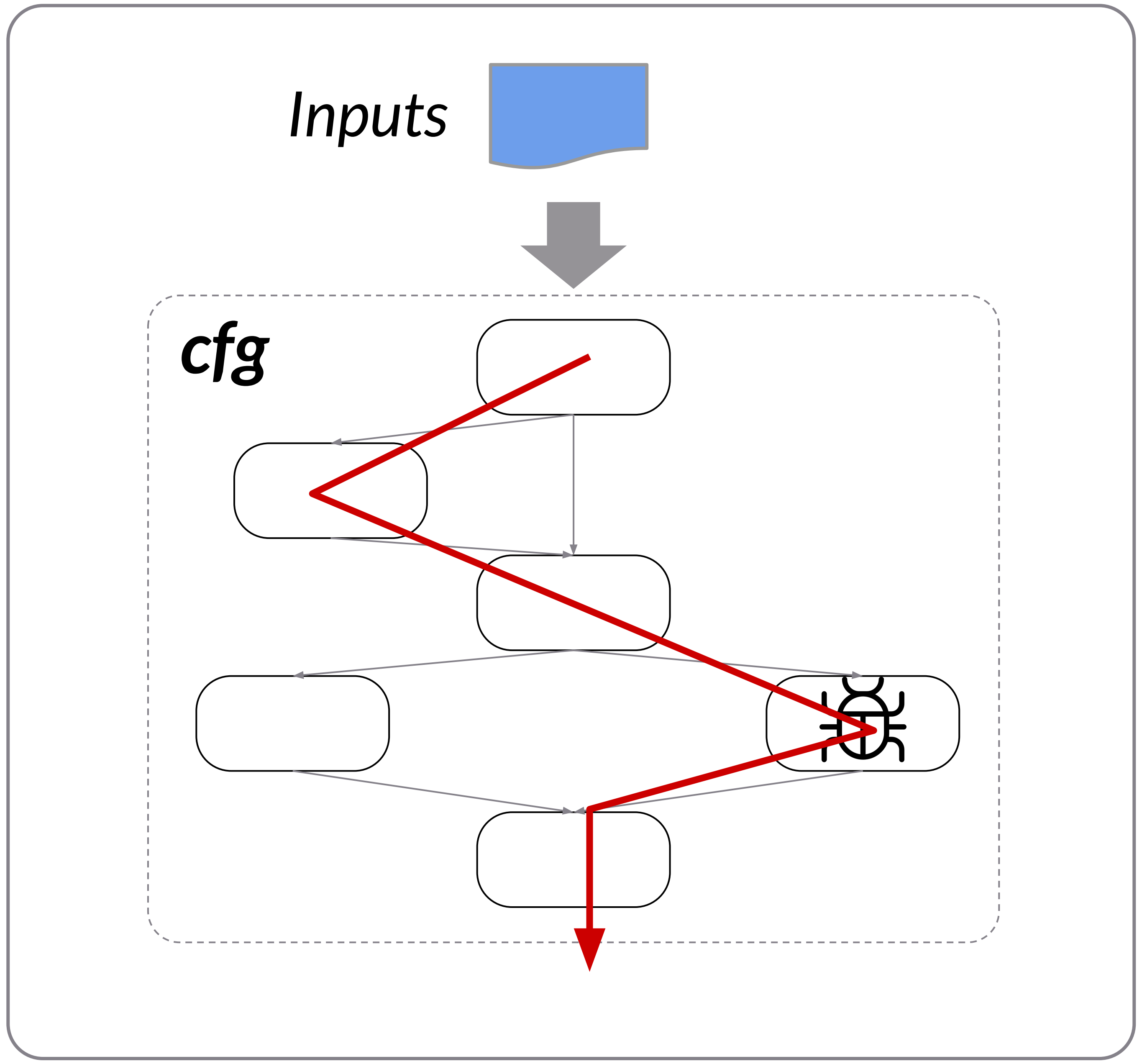
SegFuzz

SegFuzz


Single-thread fuzzing
Inputs
cfg
Multi-thread fuzzing
Syscall A
Syscall B

SegFuzz

Single-thread fuzzing



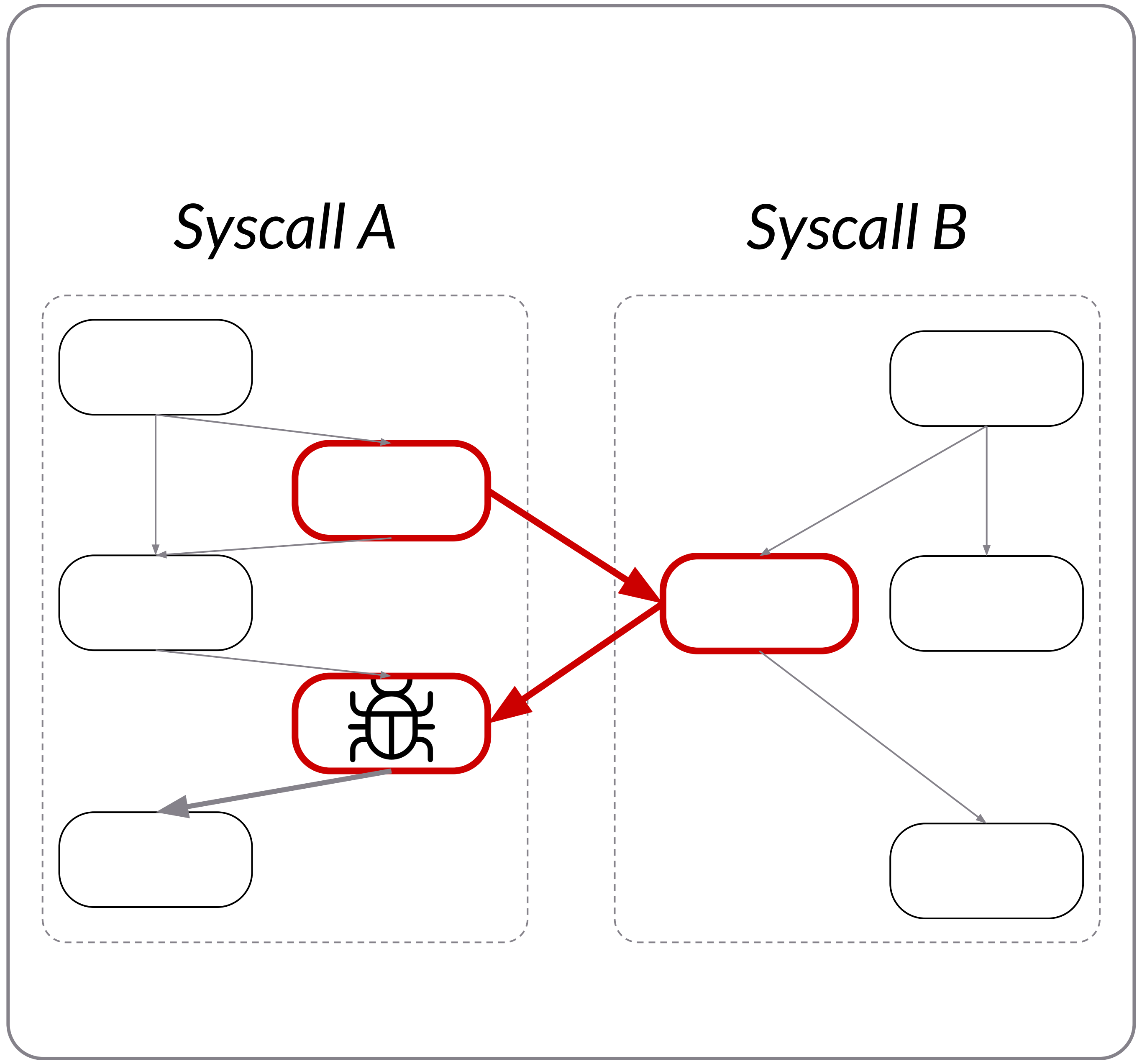
- ***Single-thread fuzzing***
  - Explore execution paths
  - Identify two system calls that potentially cause a concurrency bug

Please check our paper!

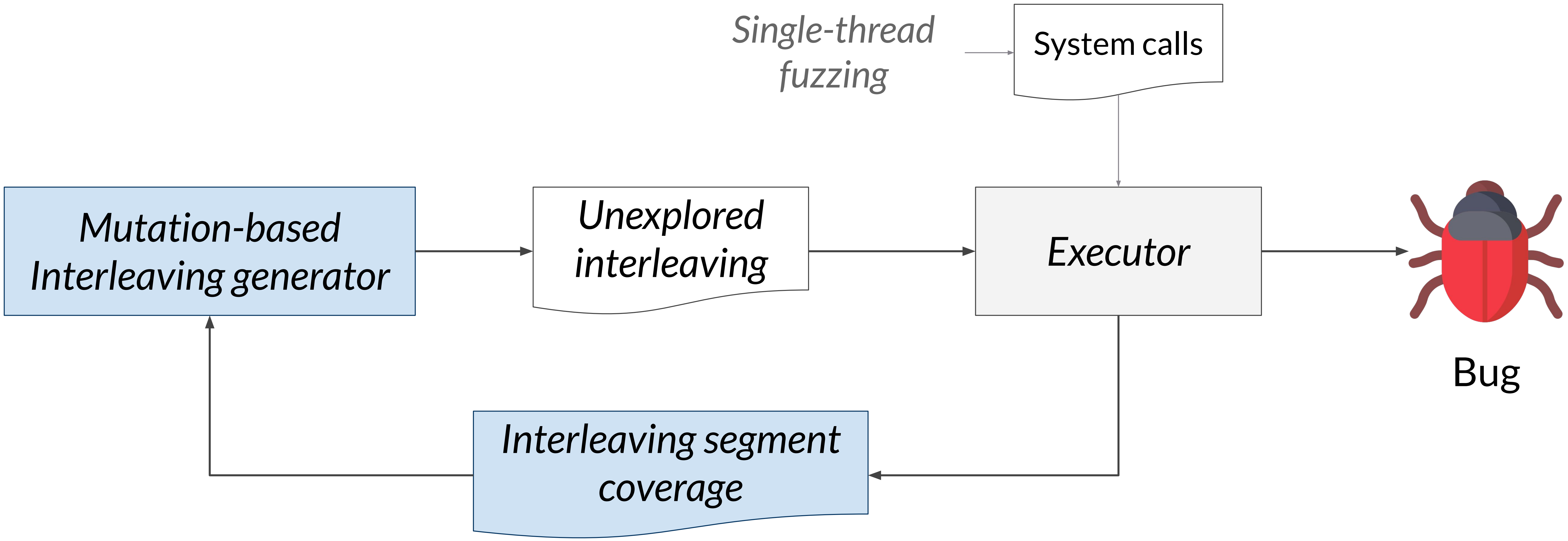
Our approach: SegFuzz

- ***Multi-thread fuzzing***
  - Explore thread interleavings
  - ***Utilizing interleaving coverage***
    - called ***interleaving segment coverage***

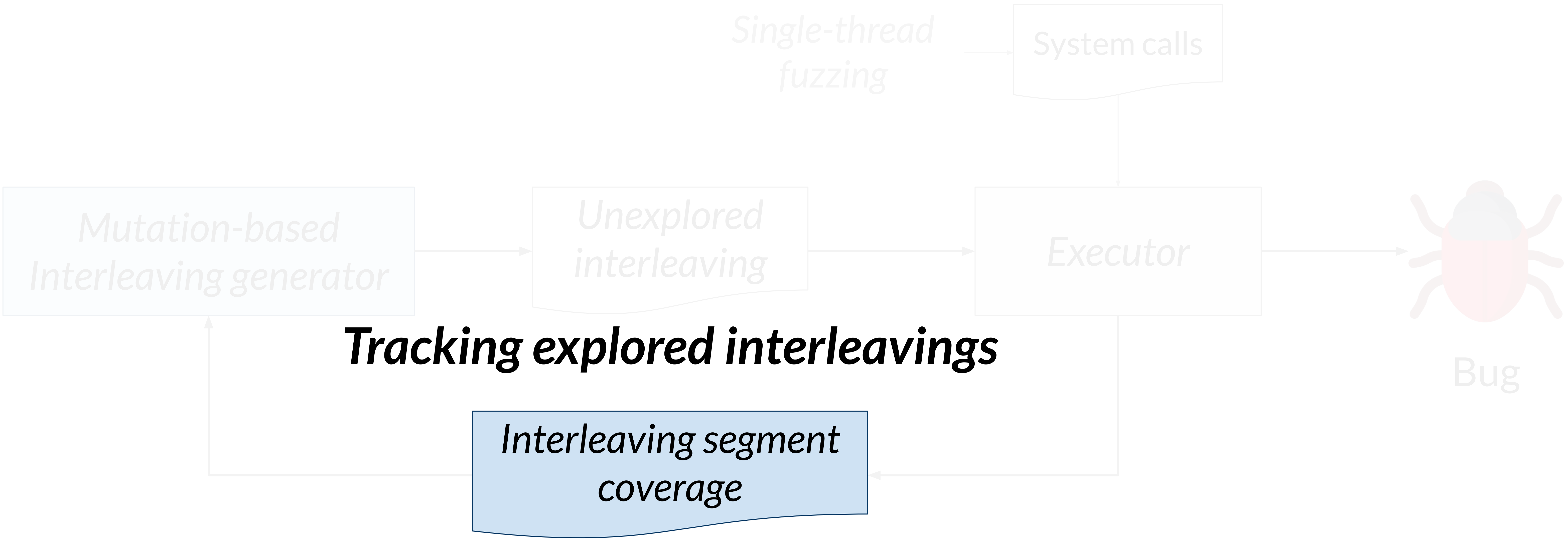
Multi-thread fuzzing



Multi-thread fuzzing of SegFuzz


Single-thread fuzzing
System calls
Mutation-based Interleaving generator
Unexplored interleaving
Executor
Bug
Interleaving segment coverage

Multi-thread fuzzing of SegFuzz

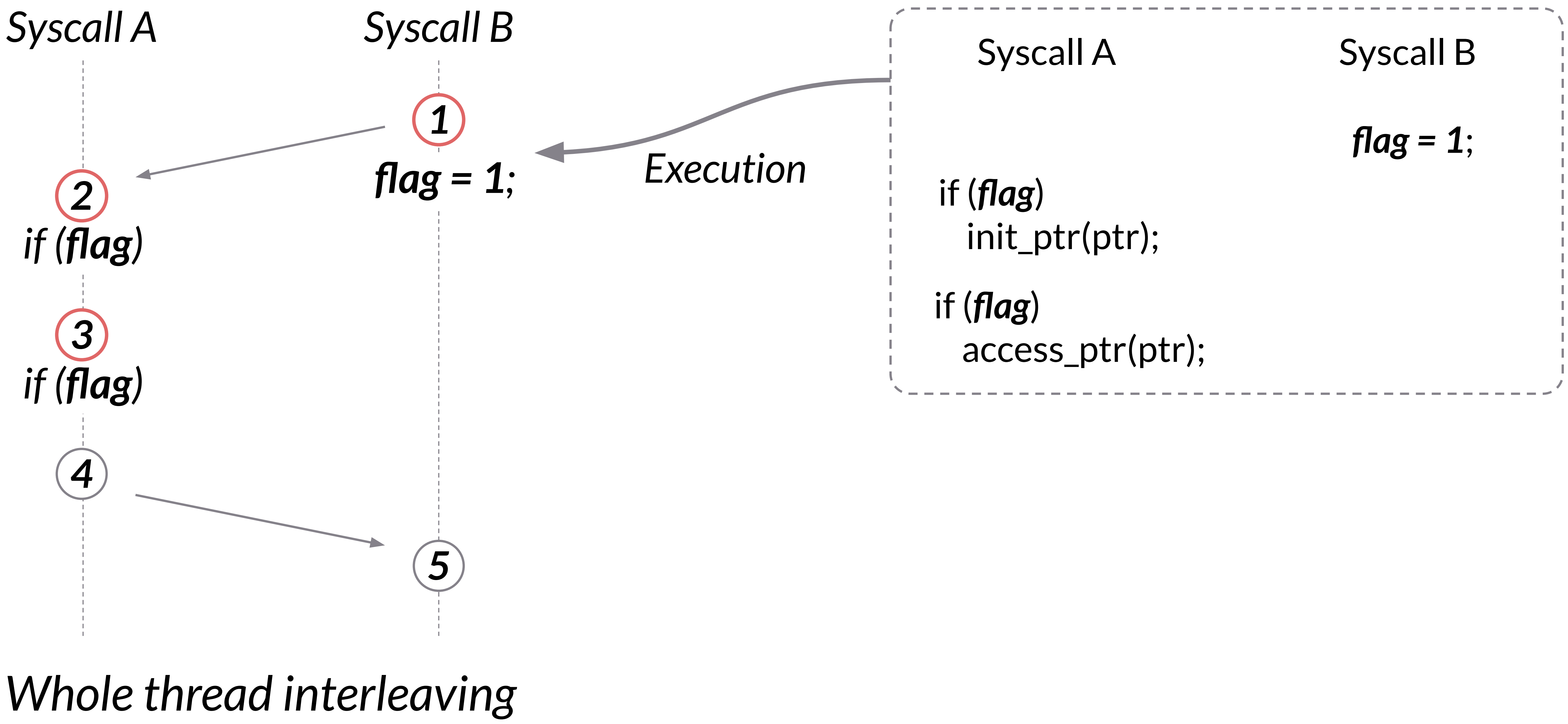


Tracking explored interleavings

Interleaving segment coverage

```
   Syscall A                  Syscall B

                               flag = 1;

 if (flag)
    init_ptr(ptr);

 if (flag)
    access_ptr(ptr);
```

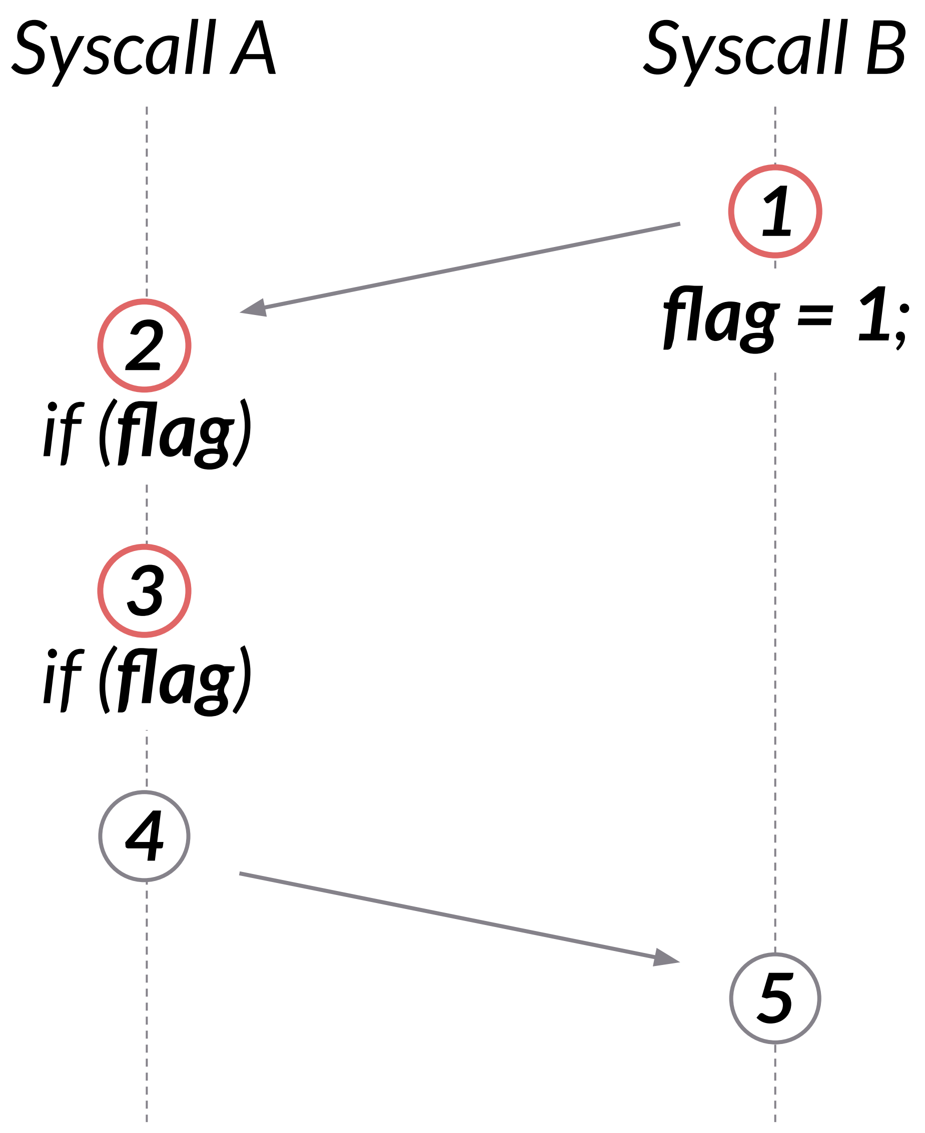
Interleaving segment coverage



```
Syscall A                          Syscall B

                                   flag = 1;

if (flag)
   init_ptr(ptr);

if (flag)
   access_ptr(ptr);
```

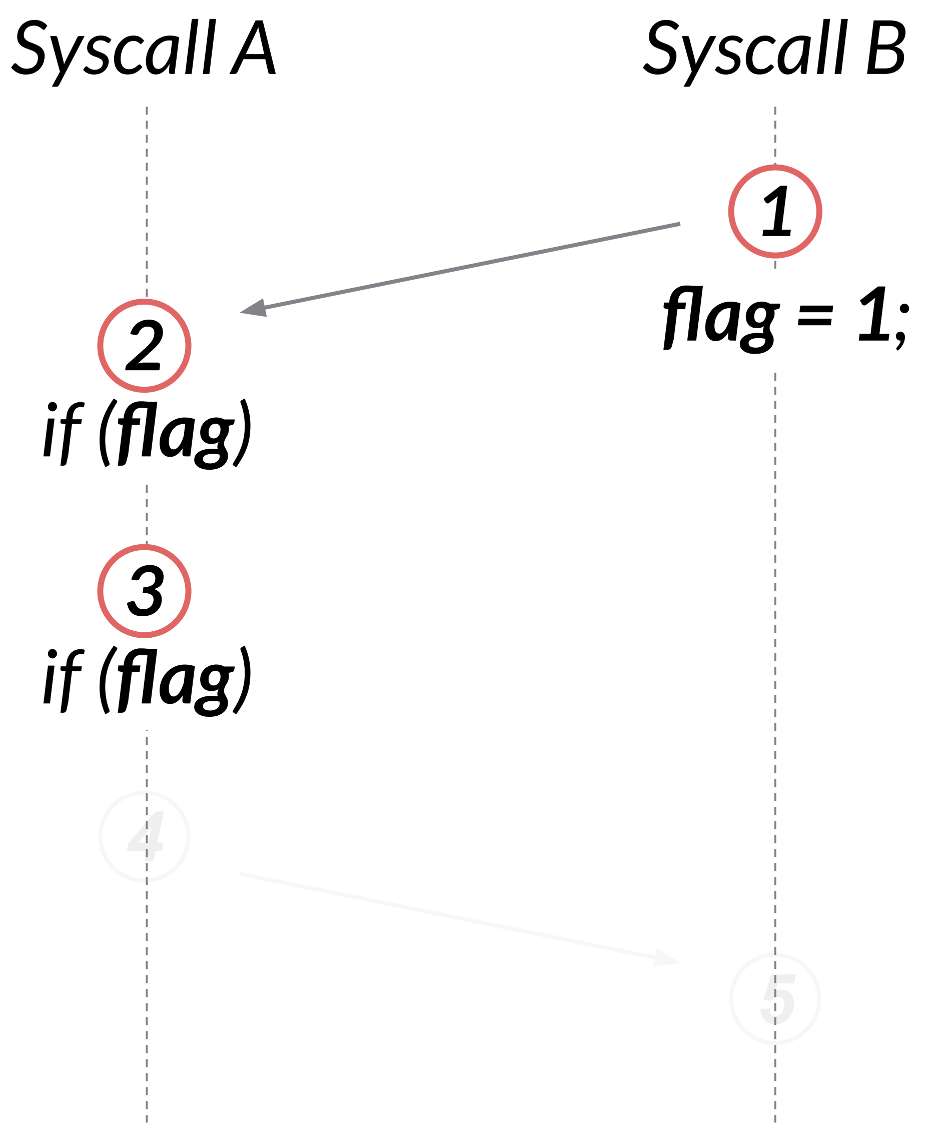
Whole thread interleaving

Interleaving segment coverage

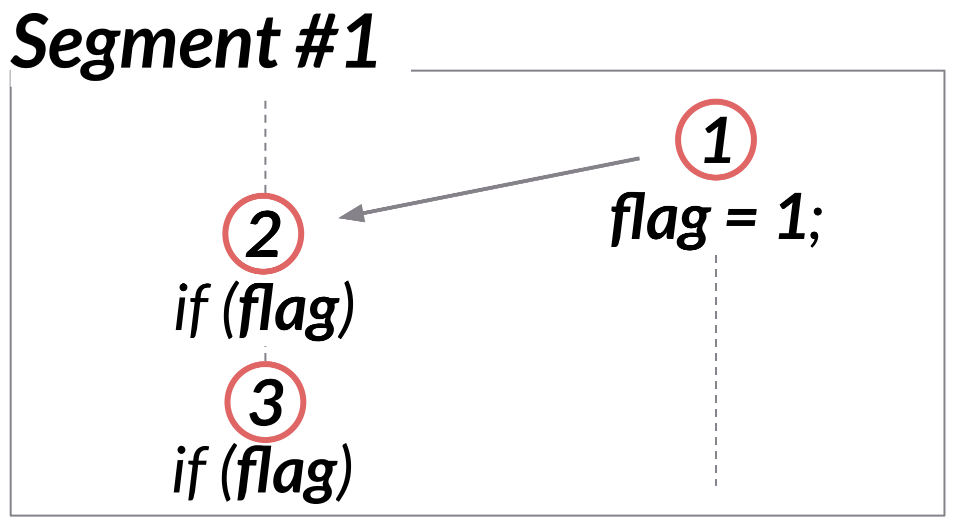


Whole thread interleaving

Interleaving segment coverage

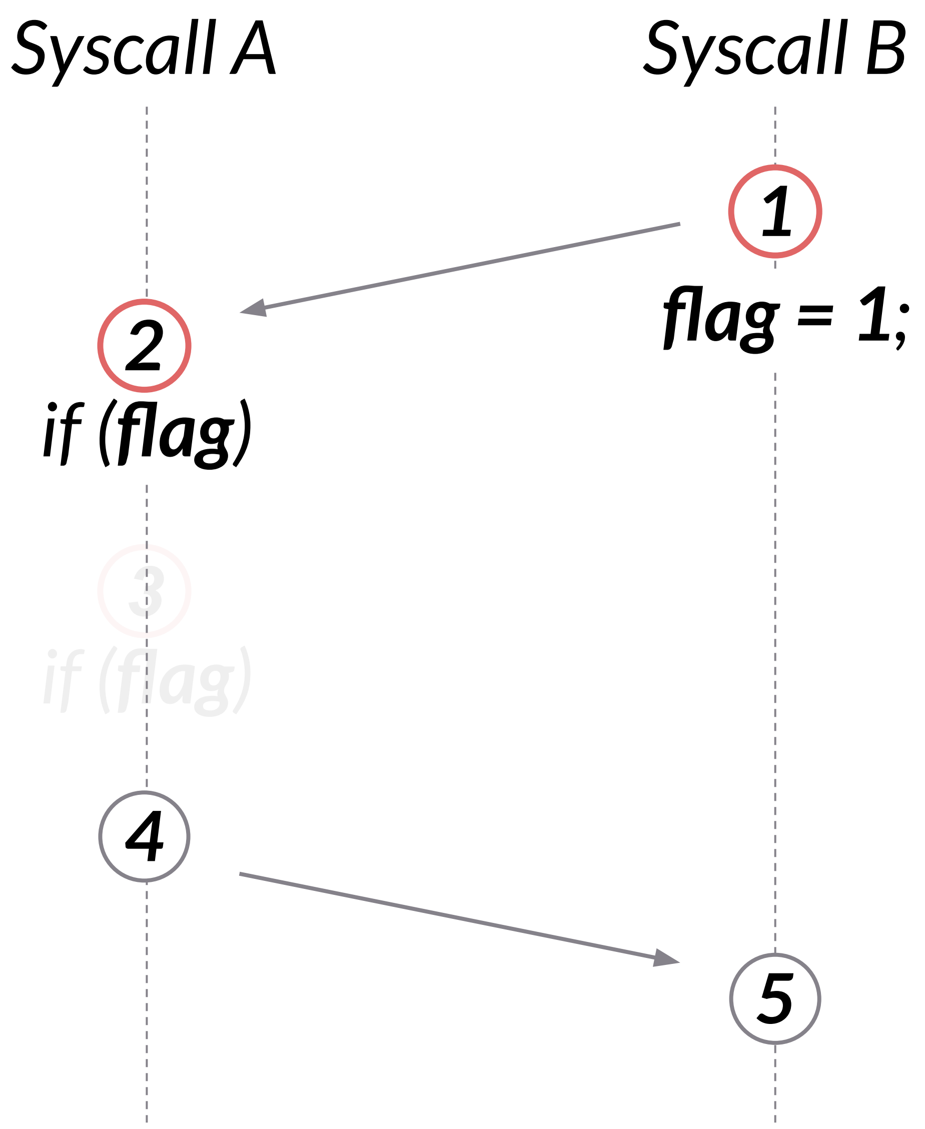


Whole thread interleaving

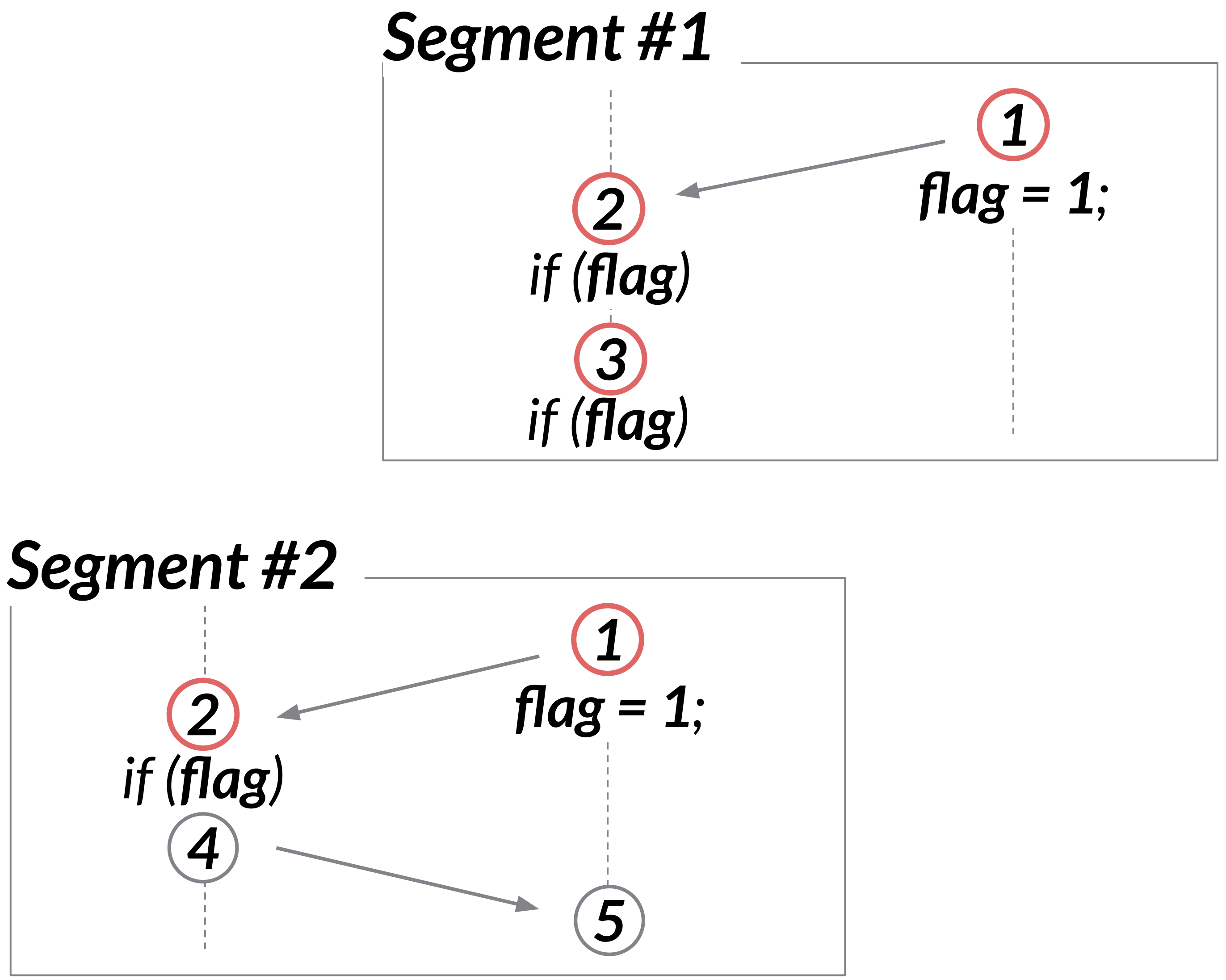


Interleaving segments
(each contains at most 4 inst.)

Interleaving segment coverage

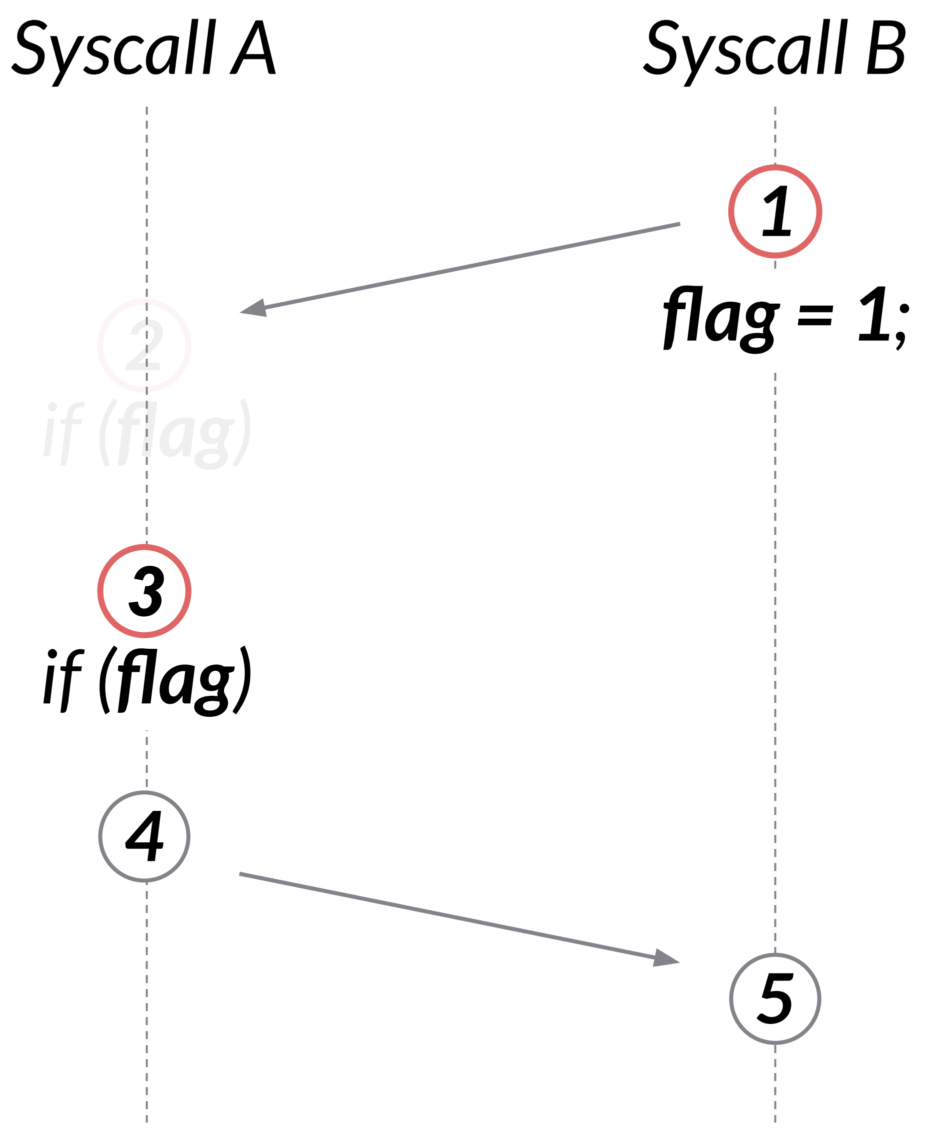


Whole thread interleaving



Interleaving segments
(each contains at most 4 inst.)

Interleaving segment coverage



Syscall A

Syscall B

1

flag = 1;

2

if (flag)

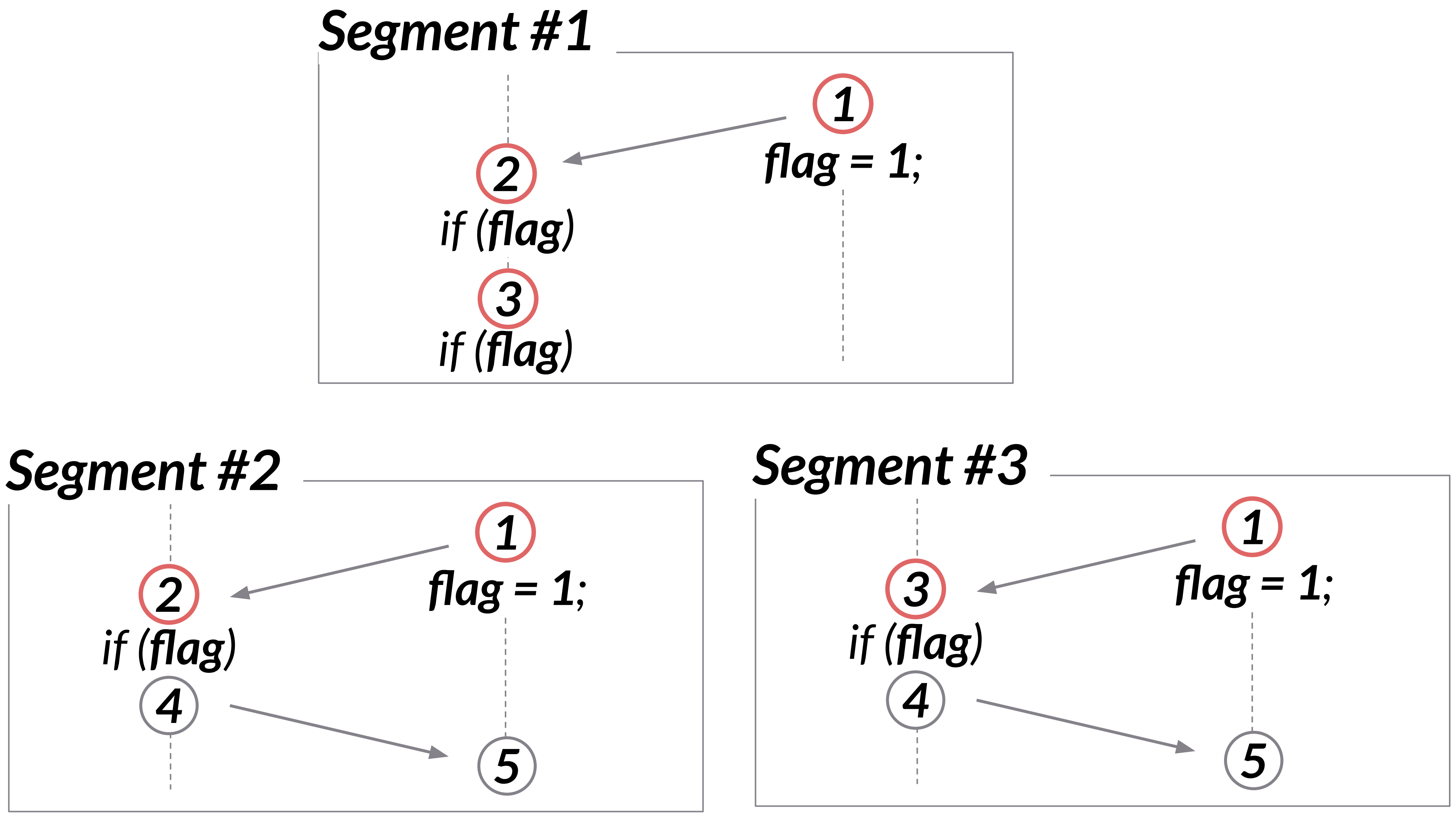
3

if (flag)

4

5

Whole thread interleaving



Segment #1

1

flag = 1;

2

if (flag)

3

if (flag)

Segment #2

1

flag = 1;

2

if (flag)

4

5

Segment #3

1

flag = 1;

3

if (flag)

4

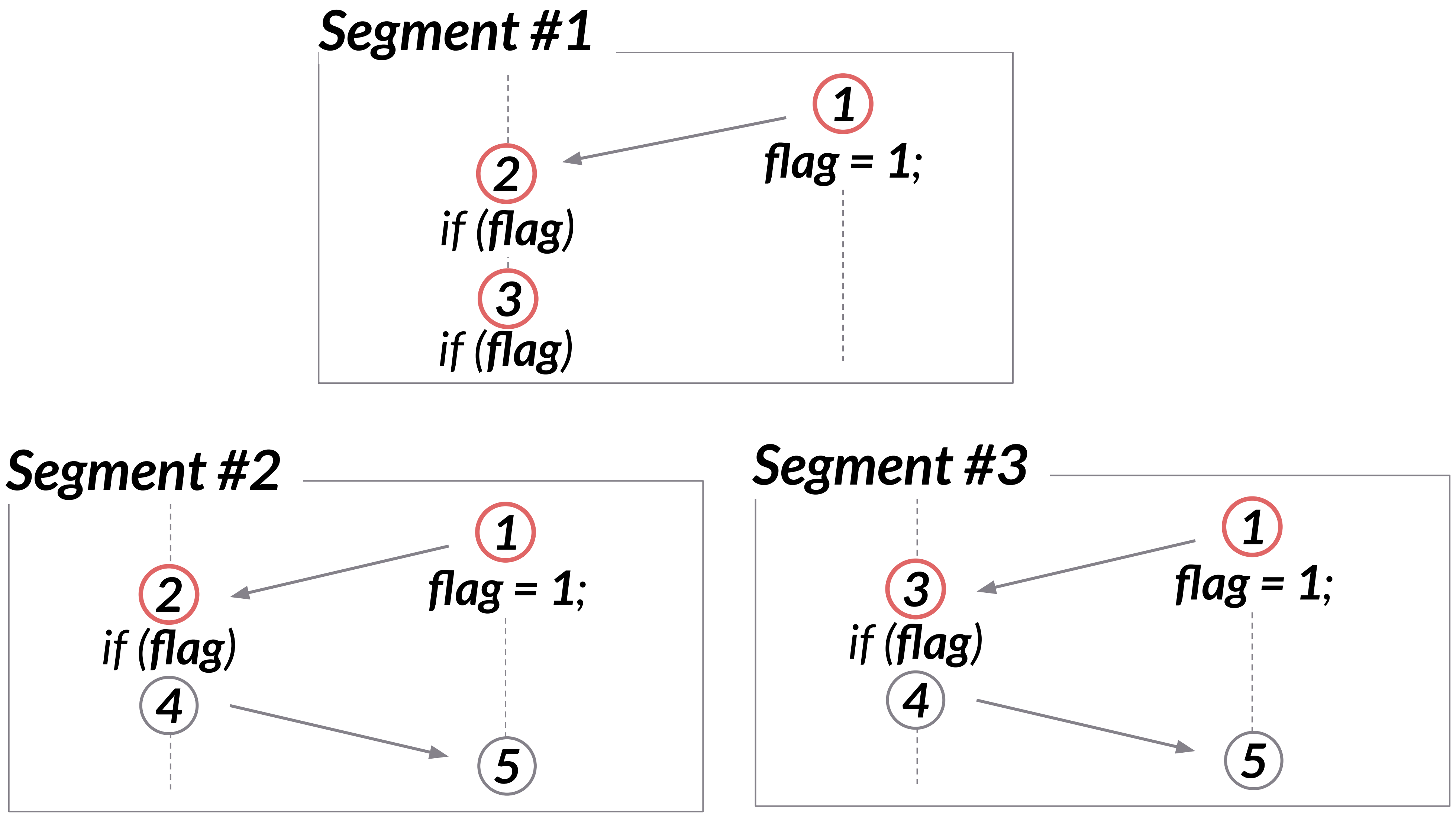
5

Interleaving segments

(each contains at most 4 inst.)

Interleaving segment coverage

- ***Interleaving segment coverage***
  - Collection of segments

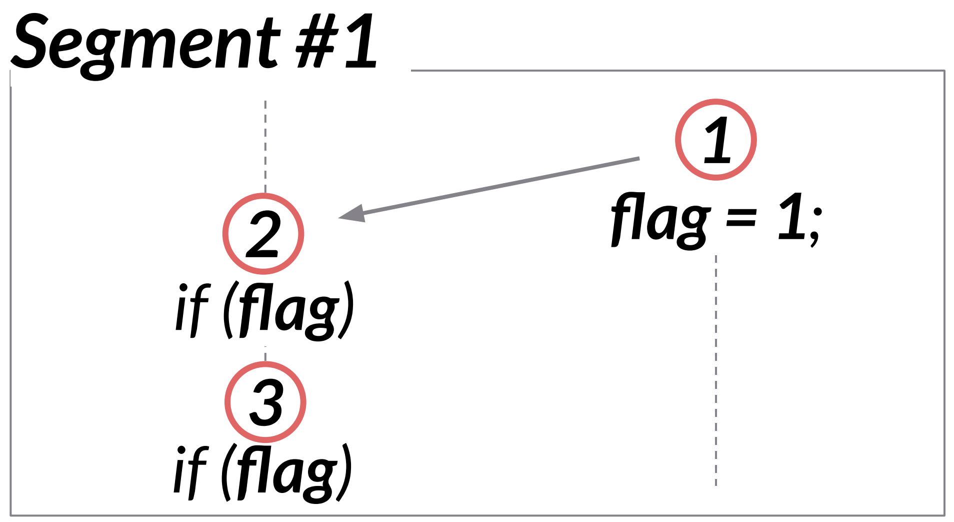


Interleaving segments

(each contains at most 4 inst.)

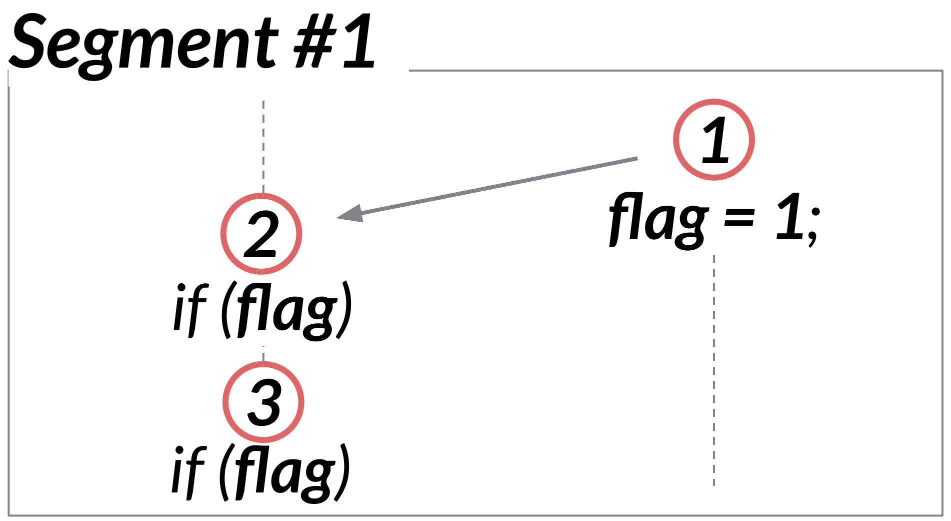
Interleaving segment coverage

- ***Interleaving segment coverage***
    - Collection of segments



Interleaving segment coverage

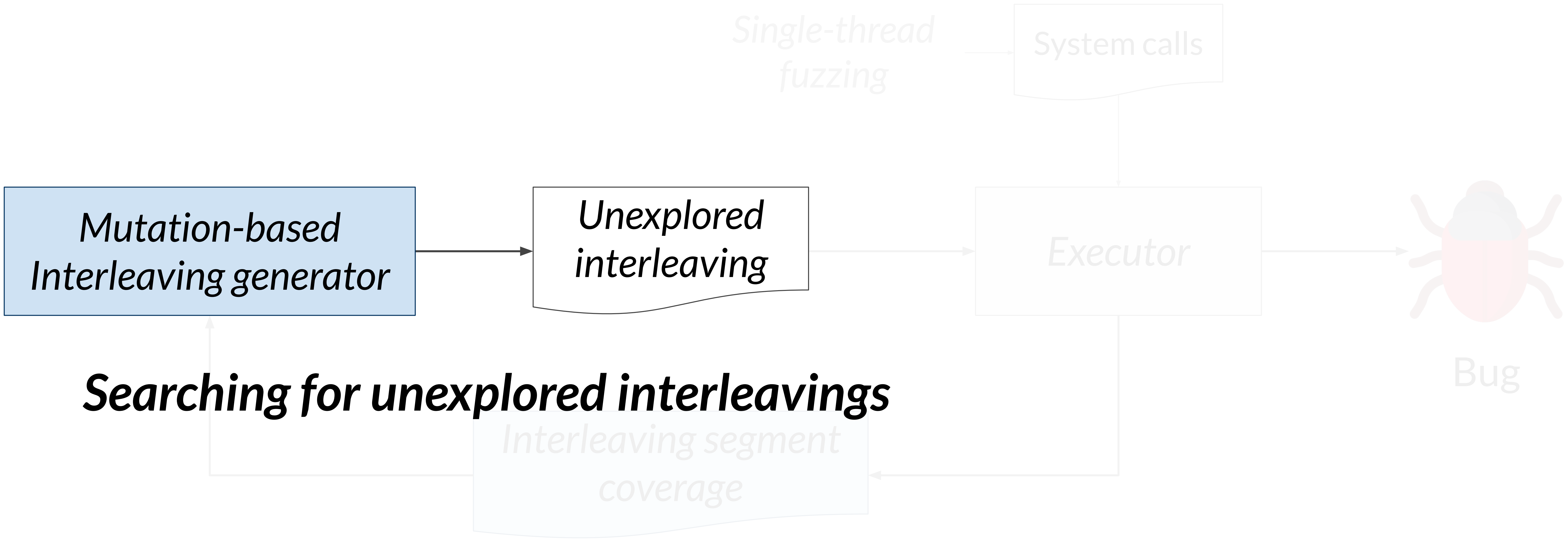
- ***Interleaving segment coverage***
  - Collection of segments



There are ***more interleavings*** *of these instructions that* ***we have not explored***

(including the offending interleaving)

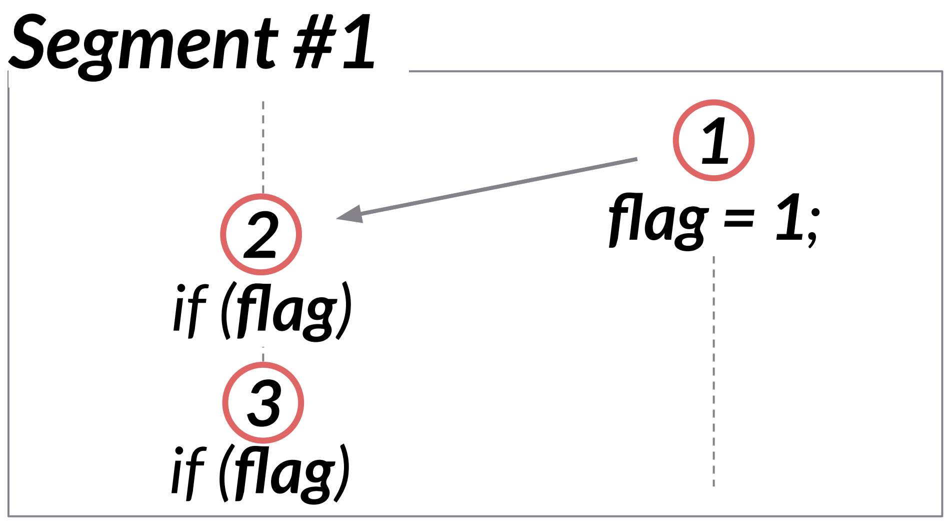
Multi-thread fuzzing of SegFuzz



Searching for unexplored interleavings

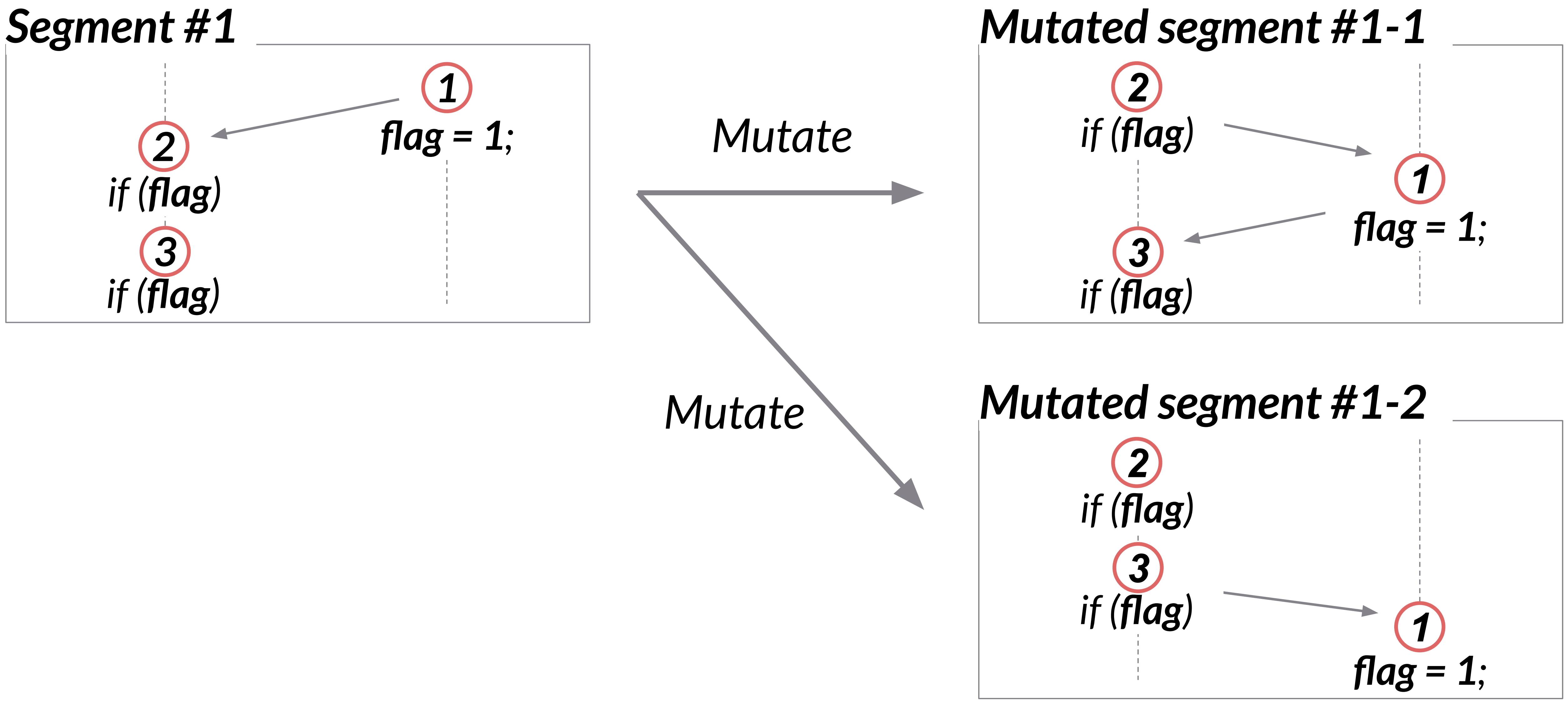
Mutation-based interleaving generator

- Mutating interleavings within segments to ***generate unexplored interleavings***



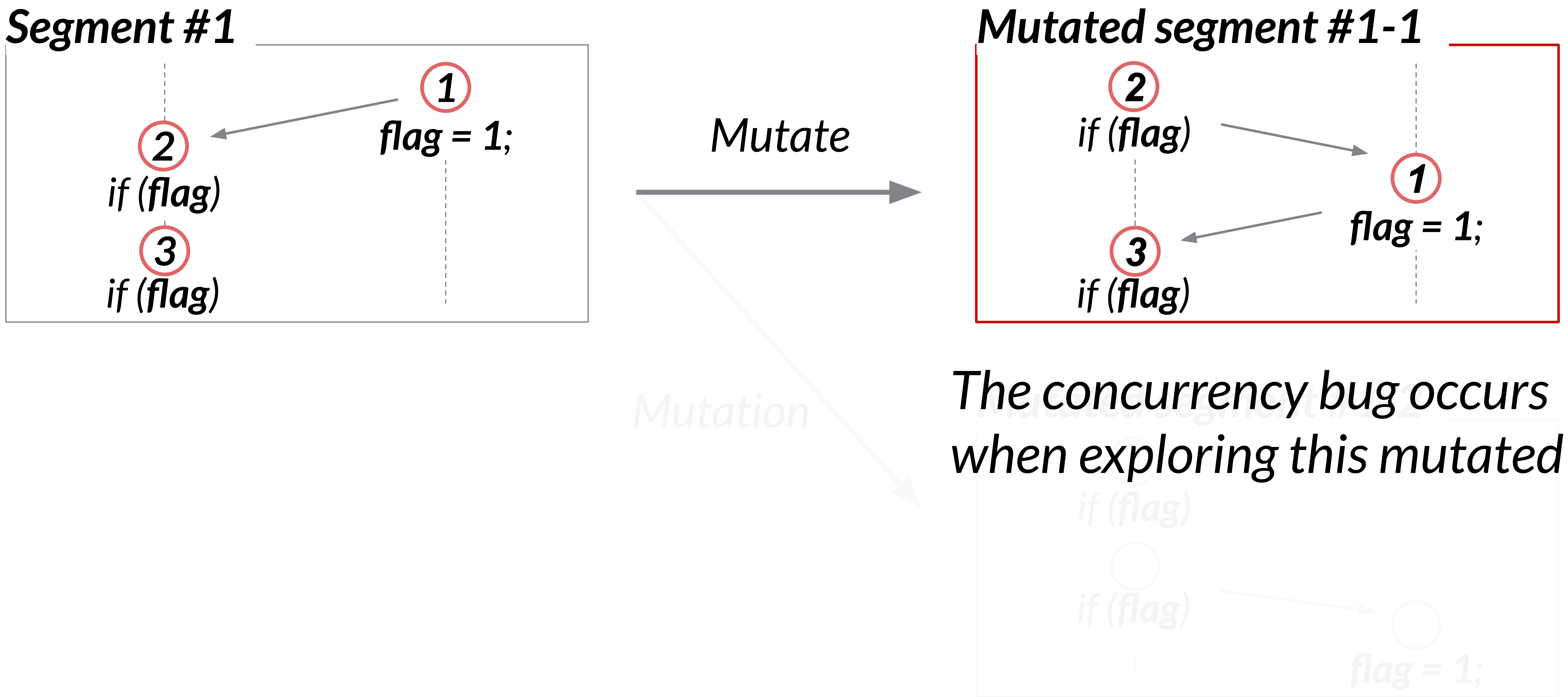
Mutation-based interleaving generator

- Mutating interleavings within segments to ***generate unexplored interleavings***



Mutation-based interleaving generator

- Mutating interleavings within segments to ***generate unexplored interleavings***



Segment #1

Mutated segment #1-1

The concurrency bug occurs when exploring this mutated segment

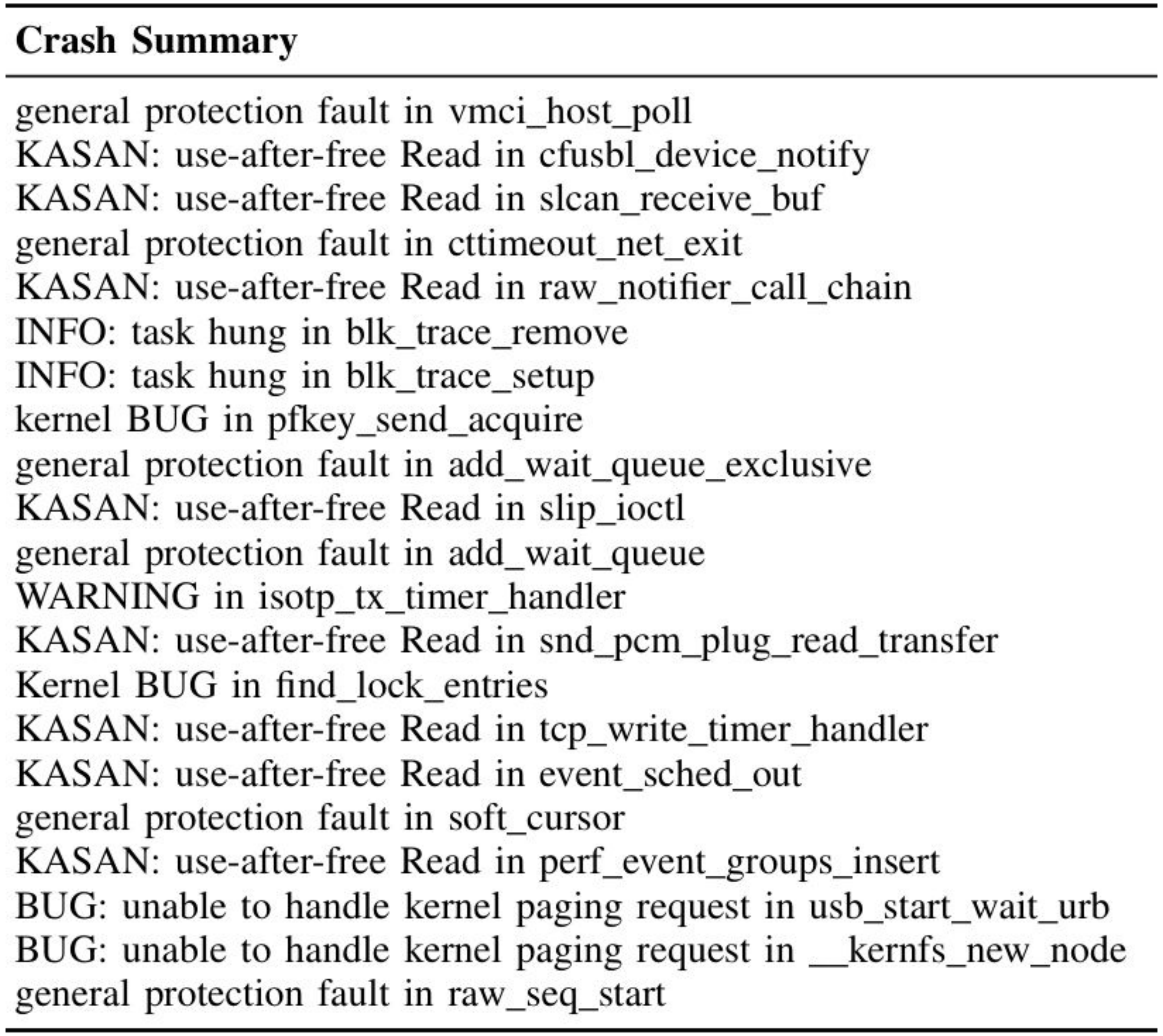
Mutation-based interleaving generator

- Mutating interleavings within segments to ***generate unexplored interleavings***

- ***Testing multiple mutated segments*** at one execution
  - Recomposing mutated segments to determine how to schedule instructions
  - ***Please check our paper!***

Evaluation

- *21 new concurrency bugs in the Linux kernel*

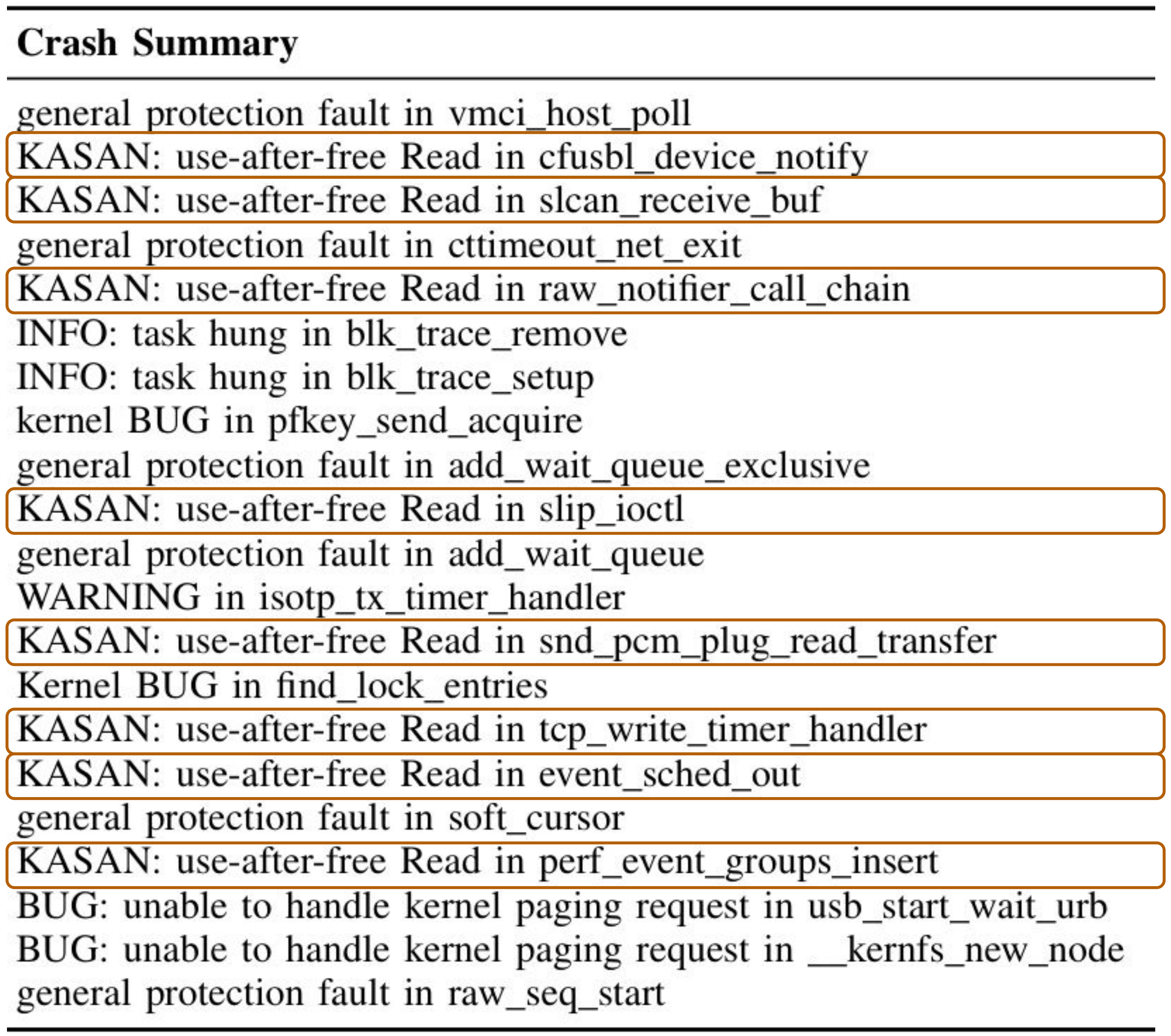


Crash Summary
general protection fault in vmci_host_poll
KASAN: use-after-free Read in cfusbl_device_notify
KASAN: use-after-free Read in slcan_receive_buf
general protection fault in cttimeout_net_exit
KASAN: use-after-free Read in raw_notifier_call_chain
INFO: task hung in blk_trace_remove
INFO: task hung in blk_trace_setup
kernel BUG in pfkey_send_acquire
general protection fault in add_wait_queue_exclusive
KASAN: use-after-free Read in slip_ioctl
general protection fault in add_wait_queue
WARNING in isotp_tx_timer_handler
KASAN: use-after-free Read in snd_pcm_plug_read_transfer
Kernel BUG in find_lock_entries
KASAN: use-after-free Read in tcp_write_timer_handler
KASAN: use-after-free Read in event_sched_out
general protection fault in soft_cursor
KASAN: use-after-free Read in perf_event_groups_insert
BUG: unable to handle kernel paging request in usb_start_wait_urb
BUG: unable to handle kernel paging request in __kernfs_new_node
general protection fault in raw_seq_start

Evaluation

- *21 new concurrency bugs in the Linux kernel*

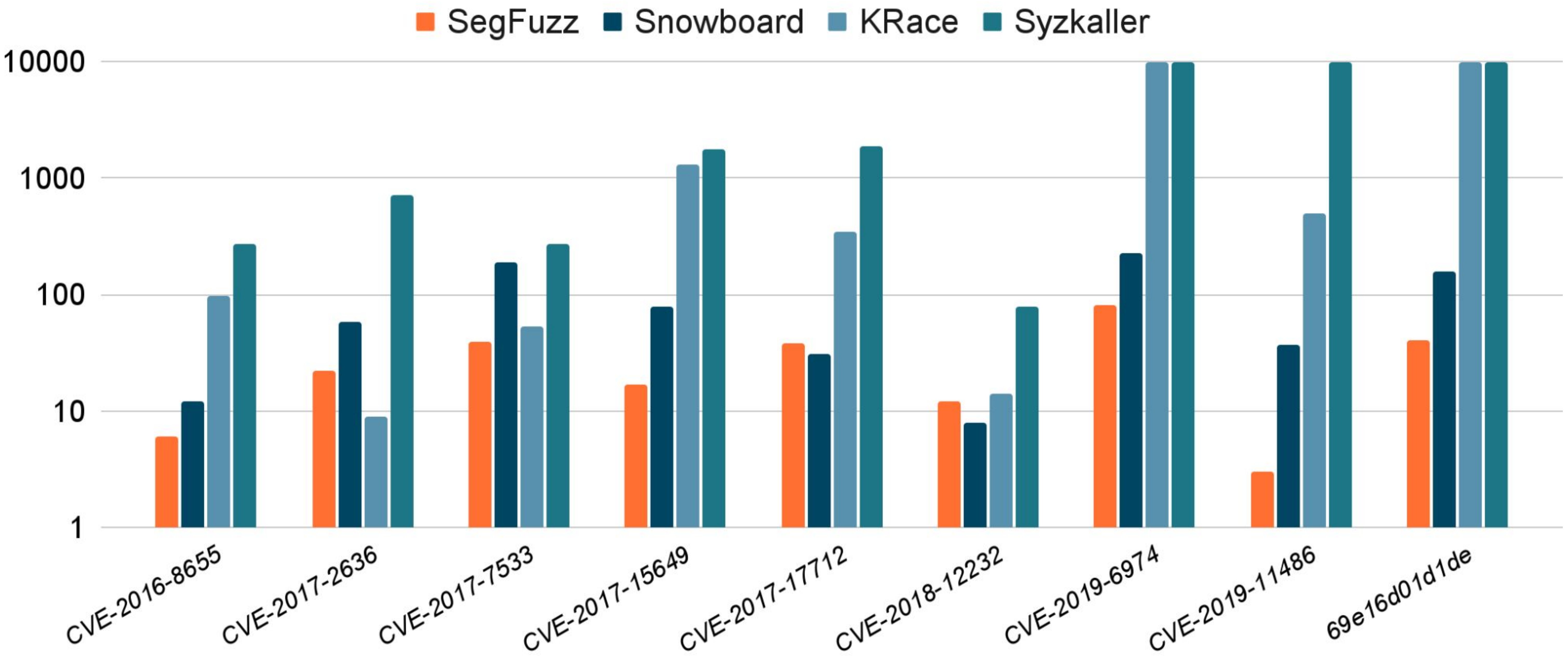
Use-after-free



Crash Summary
general protection fault in vmci_host_poll
KASAN: use-after-free Read in cfusbl_device_notify
KASAN: use-after-free Read in slcan_receive_buf
general protection fault in cttimeout_net_exit
KASAN: use-after-free Read in raw_notifier_call_chain
INFO: task hung in blk_trace_remove
INFO: task hung in blk_trace_setup
kernel BUG in pfkey_send_acquire
general protection fault in add_wait_queue_exclusive
KASAN: use-after-free Read in slip_ioctl
general protection fault in add_wait_queue
WARNING in isotp_tx_timer_handler
KASAN: use-after-free Read in snd_pcm_plug_read_transfer
Kernel BUG in find_lock_entries
KASAN: use-after-free Read in tcp_write_timer_handler
KASAN: use-after-free Read in event_sched_out
general protection fault in soft_cursor
KASAN: use-after-free Read in perf_event_groups_insert
BUG: unable to handle kernel paging request in usb_start_wait_urb
BUG: unable to handle kernel paging request in __kernfs_new_node
general protection fault in raw_seq_start

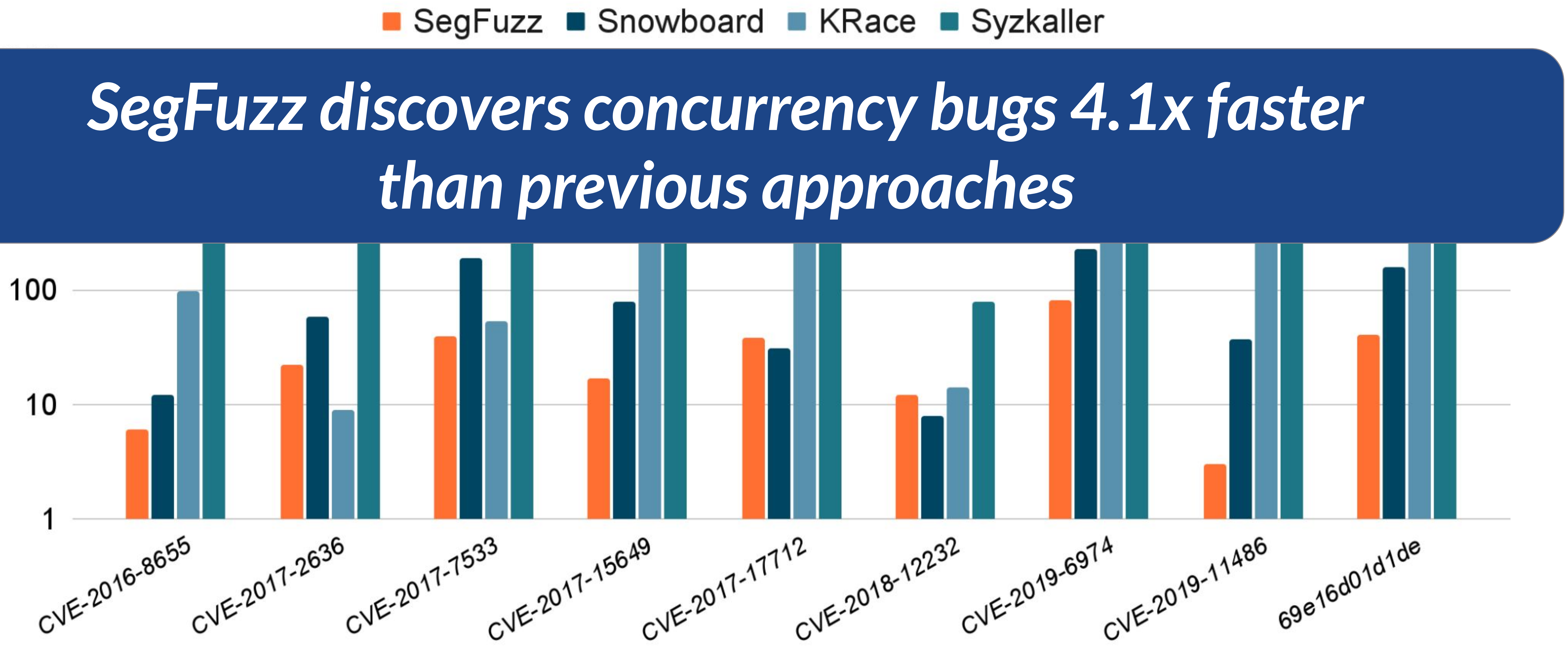
Evaluation - Comparison study

- Compare against Snowboard, KRace, and Syzkaller with 9 kernel concurrency bugs



Evaluation - Comparison study

- Compare against Snowboard, KRace, and Syzkaller with 9 kernel concurrency bugs



SegFuzz discovers concurrency bugs 4.1x faster than previous approaches

Conclusion

- ***SegFuzz***, a fuzzing framework to effectively discover kernel concurrency bugs
  - Applying the problem decomposition strategy based on the previous finding
- A novel thread interleaving coverage called ***interleaving segment coverage***
  - *Tracking explored thread interleavings*
  - *Efficiently exploring unexplored thread interleavings*
- Discovered 21 new concurrency bugs in the Linux kernel

SEGFUZZ: Segmentizing Thread Interleaving to Discover Kernel Concurrency Bugs through Fuzzing

Thank You!