Ozz: Identifying Kernel Out-of-Order Concurrency Bugs with In-Vivo Memory Access Reordering

Dae R. Jeong[1,2], Yewon Choi[2], Byoungyoung Lee[3], Insik Shin[2], Youngjin Kwon[2]

[1]Georgia Institute of Technology

[2]Korea Advanced Institute of Science & Technology

[3]Seoul National University

Ozz: Identifying Kernel Out-of-Order Concurrency Bugs with In-Vivo Memory Access Reordering

Dae R. Jeong[1,2], Yewon Choi[2], Byoungyoung Lee[3], Insik Shin[2], Youngjin Kwon[2]

[1]Georgia Institute of Technology

[2]Korea Advanced Institute of Science & Technology

[3]Seoul National University

Concurrency bugs caused by Out-of-order execution

The culprit!

Syscall A	Syscall B
ptr = malloc();	
ready = true;	
	if (***ready***)
	****ptr*** = 1

Syscall A initializes ***ptr*** *then announces it is* ***ready***

If it is ***ready****,* *Syscall B accesses* ***ptr***

Uninitialized access!

Okay, Syscall B seems to access ***ptr*** *only if it is* ***ready***

In Apple Silicon M3, however… ***Why?***

Concurrency bugs caused by Out-of-order execution

In reality…

Syscall A | Syscall B

```
ptr = malloc();
ready = true;
                        if (ready)
                           *ptr = 1
```

*Syscall A announces it is **ready***

Wait… it seems something is missing?

*Okay, it is **ready**,*
*I am going to access **ptr***

Uninitialized access!

*What is the **correct** implementation?*

Memory barrier to prevent out-of-order execution

Syscall A

Syscall B

```
ptr = malloc();
memory_barrier();
ready = true;
```

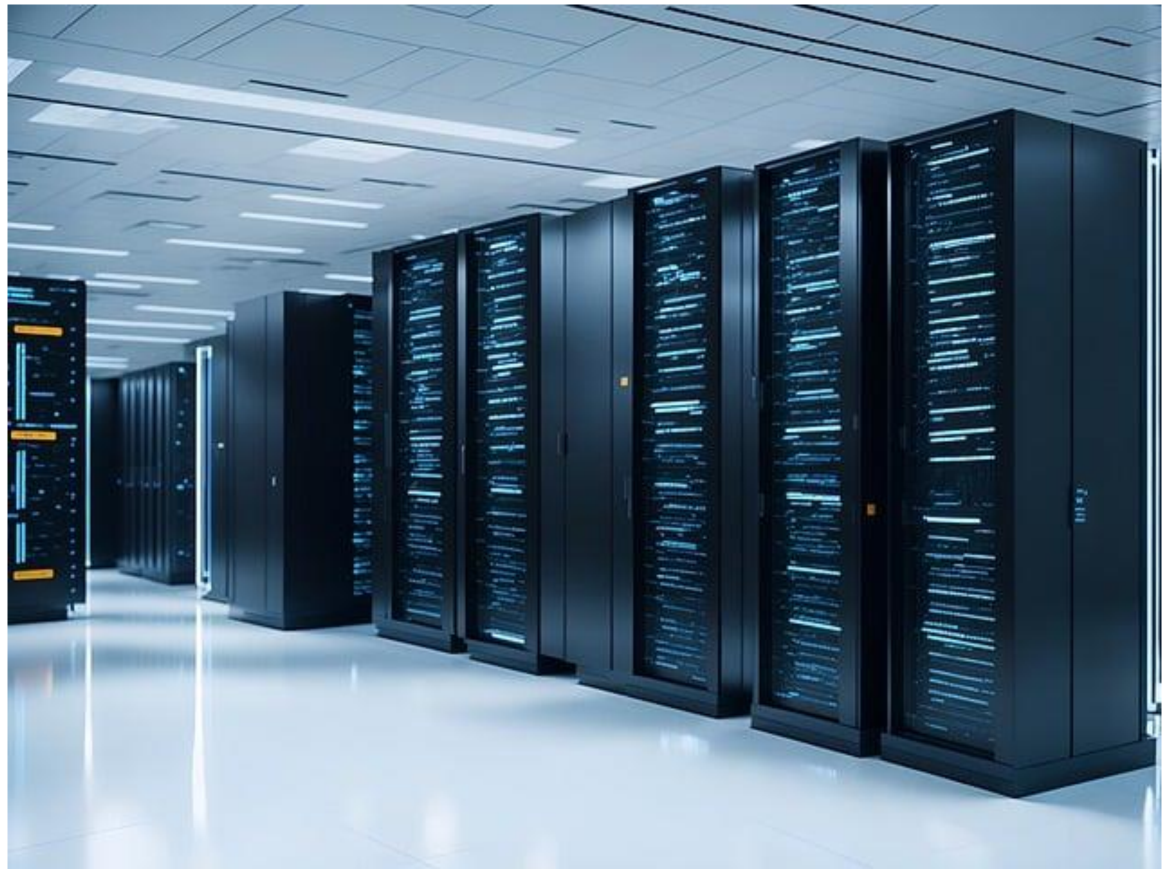
→ The processor *guarantees that*
ptr is initialized
*before setting **ready** to true*

```
if (ready)
```

If developers misses memory barriers,
out-of-order execution causes concurrency bugs

Machines exhibiting this behavior

ARM-based machines are getting more popular these days



*Memory ordering is **hard to think about**,*
*and people **won't even realize that they may be wrong**.*

- Linux developer

Machines exhibiting this behavior

ARM-based machines are getting more popular these days

[SRU,Trusty,1/1] tty: fix stall caused by missing memory barrier in drivers/tty/n

Message ID f00642df1c338f1dbe2bc9a58a8aaeef71
State New
Headers show

Commit Message

Joseph Salisbury

```
From: Kosuke Tatsukawa <tatsu@ab.jp.nec.com>

BugLink: http://bugs.launchpad.net/bugs/15128

My colleague ran into a program stall on a x8
n_tty_read() was waiting for data even if the
in the pty.  kernel stack for the stuck proce
 #0 [ffff88303d107b58] __schedule at ffffffff
 #1 [ffff88303d107bd0] schedule at ffffffff81
 #2 [ffff88303d107bf0] schedule_timeout at ff
 #3 [ffff88303d107ca0] wait_woken at ffffffff
 #4 [ffff88303d107ce0] n_tty_read at ffffffff
 #5 [ffff88303d107dd0] tty_read at ffffffff81
 #6 [ffff88303d107e20] __vfs_read at ffffffff
 #7 [ffff88303d107ec0] vfs_read at ffffffff81
 #8 [ffff88303d107f00] sys_read at ffffffff81
 #9 [ffff88303d107f50] entry_SYSCALL_64_fastp
```

Xen Missing memory barriers DoS (XSA-340)

HIGH Nessus Plugin ID 144856

Information | Dependencies | Dependents | Changel

Synopsis

The remote Xen hypervisor installation is missing a security upd

Description

A denial of service (DoS) vulnerability exists in Xen servers whe to a missing memory barrier. An authenticated, local attacker m resulting in a Denial of Service (DoS).

CVE-2021-29650 Detail

MODIFIED

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided.

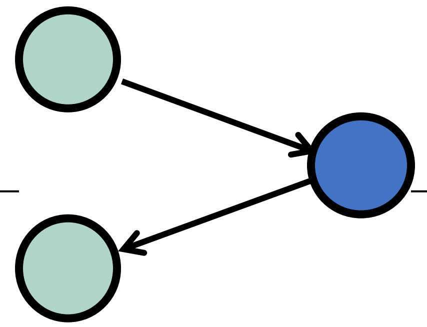
Description

The goal of this work is to identify concurrency bugs that are caused by missing memory barriers

- Linux developer

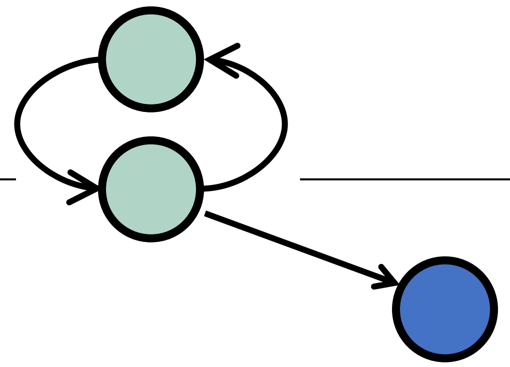
Challenges in identifying out-of-order bugs

OoO bugs manifests depending on ***two types of non-deterministic behaviors***



Thread interleaving

The order of memory accesses
between multiple threads

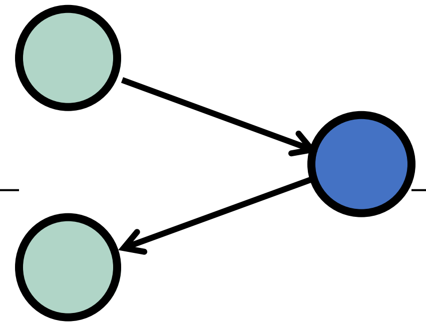


Out-of-order execution

The order of memory accesses
inside a* single *thread

Challenges in identifying out-of-order bugs

OoO bugs manifests depending on ***two types of non-deterministic behaviors***



Thread interleaving

The order of memory accesses
between multiple threads

Previous work:
- *DataCollider [OSDI'10], SKI [OSDI'14], Razzer [S&P'19], Snowboard [SOSP'21], …*

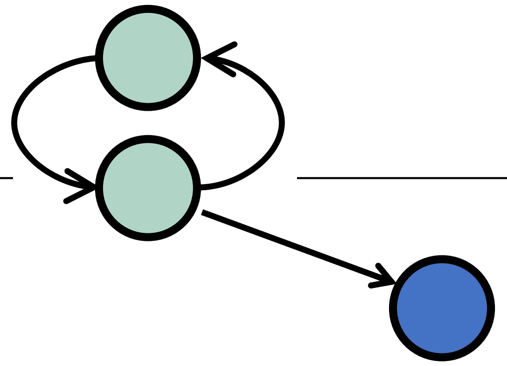
- *Various methods are used (e.g., breakpoints, suspending vCPUs…)*

Challenges in identifying out-of-order bugs

OoO bugs manifests depending on ***two types of non-deterministic behaviors***

No approach has been proposed to control out-of-order execution

Even worse, previous approaches ***obscure*** *the observation of out-of-order execution!*

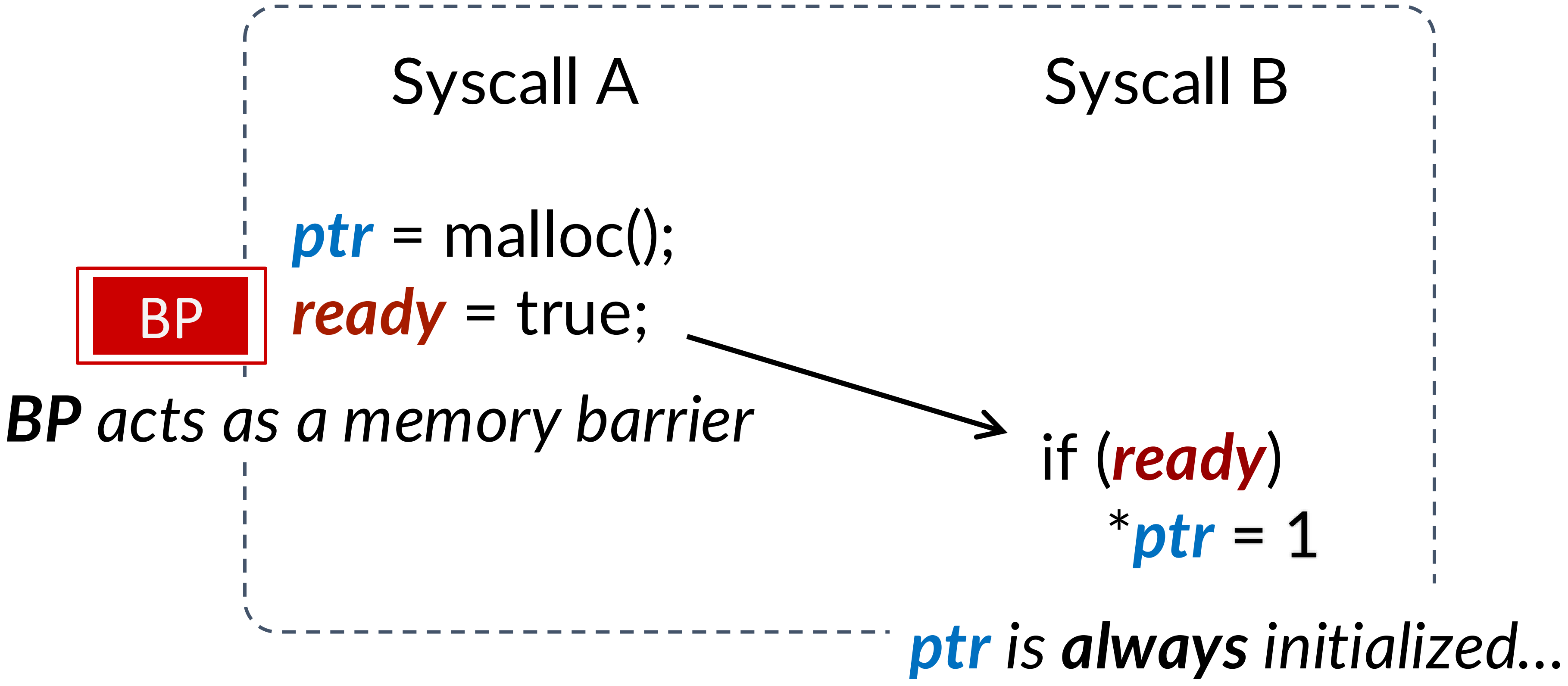


Out-of-order execution

The order of memory accesses
inside a **single** ***thread***

Challenges in identifying out-of-order bugs

Controlling thread interleaving ***obscures the effect of out-of-order execution***



A new method is required to control out-of-order execution

In this work, we introduce…

OEMU

- A mechanism to tame the non-deterministic behavior of out-of-order execution during runtime

Ozz

- A kernel fuzzer tailored to find OoO bugs by deterministically controlling
  - ***Out-of-order execution*** through OEMU, and
  - ***Thread interleaving*** thruugh a custom scheduler from a previous work[1]

1: Jeong, Dae R., Byoungyoung Lee, Insik Shin, and Youngjin Kwon. "Segfuzz: Segmentizing thread interleaving to discover kernel concurrency bugs through fuzzing." In *2023 IEEE Symposium on Security and Privacy (SP).*

In this work, we introduce…

OEMU

- A mechanism to tame the non-deterministic behavior of out-of-order execution during runtime

Ozz

- A kernel fuzzer tailored to find OoO bugs by deterministically controlling
  - ***Out-of-order execution*** through OEMU, and
  - ***Thread interleaving*** through a custom scheduler from a previous work[1]

1: Jeong, Dae R., Byoungyoung Lee, Insik Shin, and Youngjin Kwon. "Segfuzz: Segmentizing thread interleaving to discover kernel concurrency bugs through fuzzing." In *2023 IEEE Symposium on Security and Privacy (SP).*

OEMU

A mechanism to ***control out-of-order execution*** during runtime

- Consisting of a compiler pass and callback functions

```
X = 1;
r1 = Y;
```

Source code

Compile

```
store_value(&X, 1);
r1 = load_value(&Y);
```

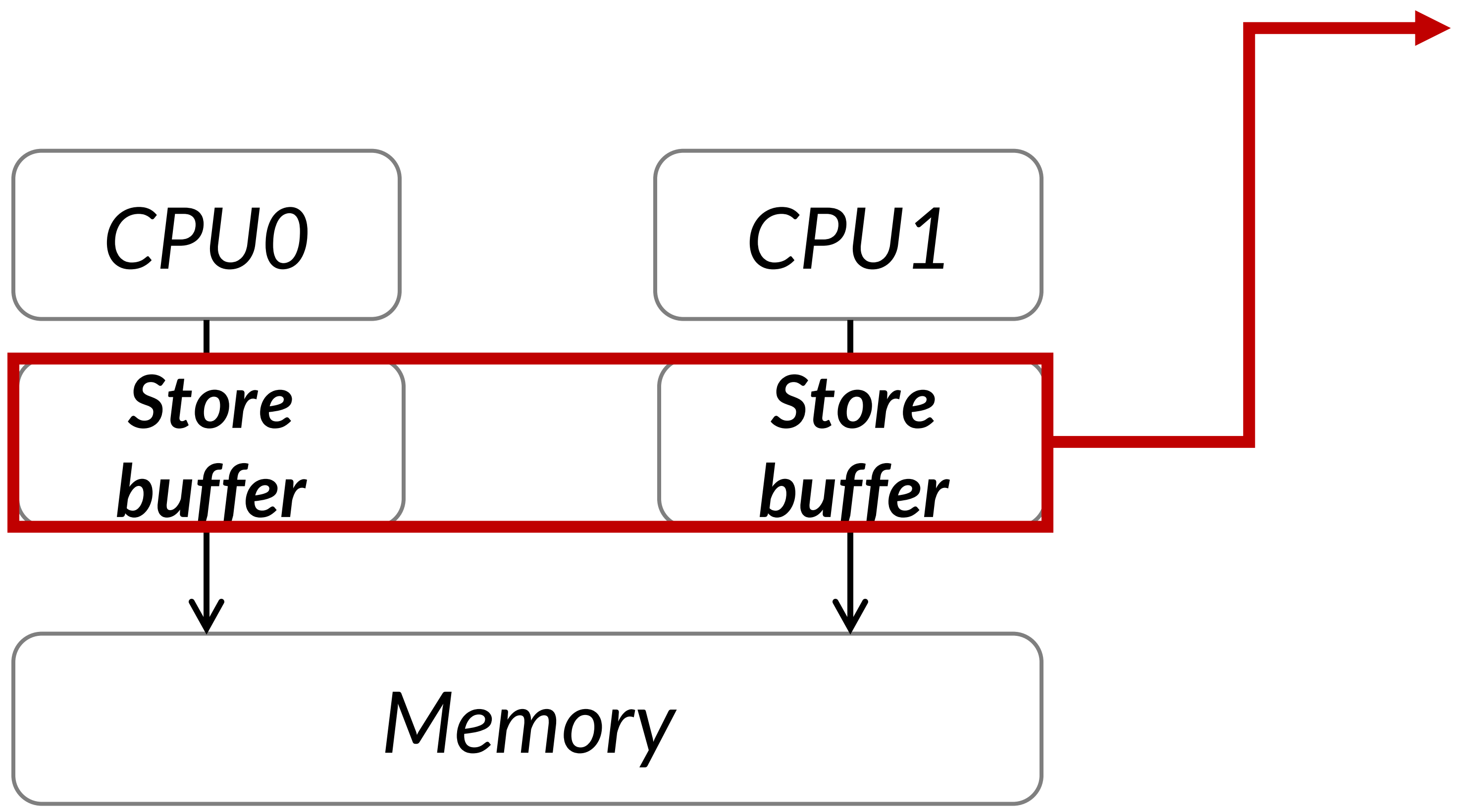
Compiled binary

Providing two primitive operations

- Delayed store operation
- Versioned load operation

Delayed store operation

*Emulating how hardware reorders **store** operations*



Store buffer

- A small hardware component that temporary holds the results of store operations
- ***It may change the order*** in which the results of store operations are written to memory

→ ***OEMU emulates the store buffer!***

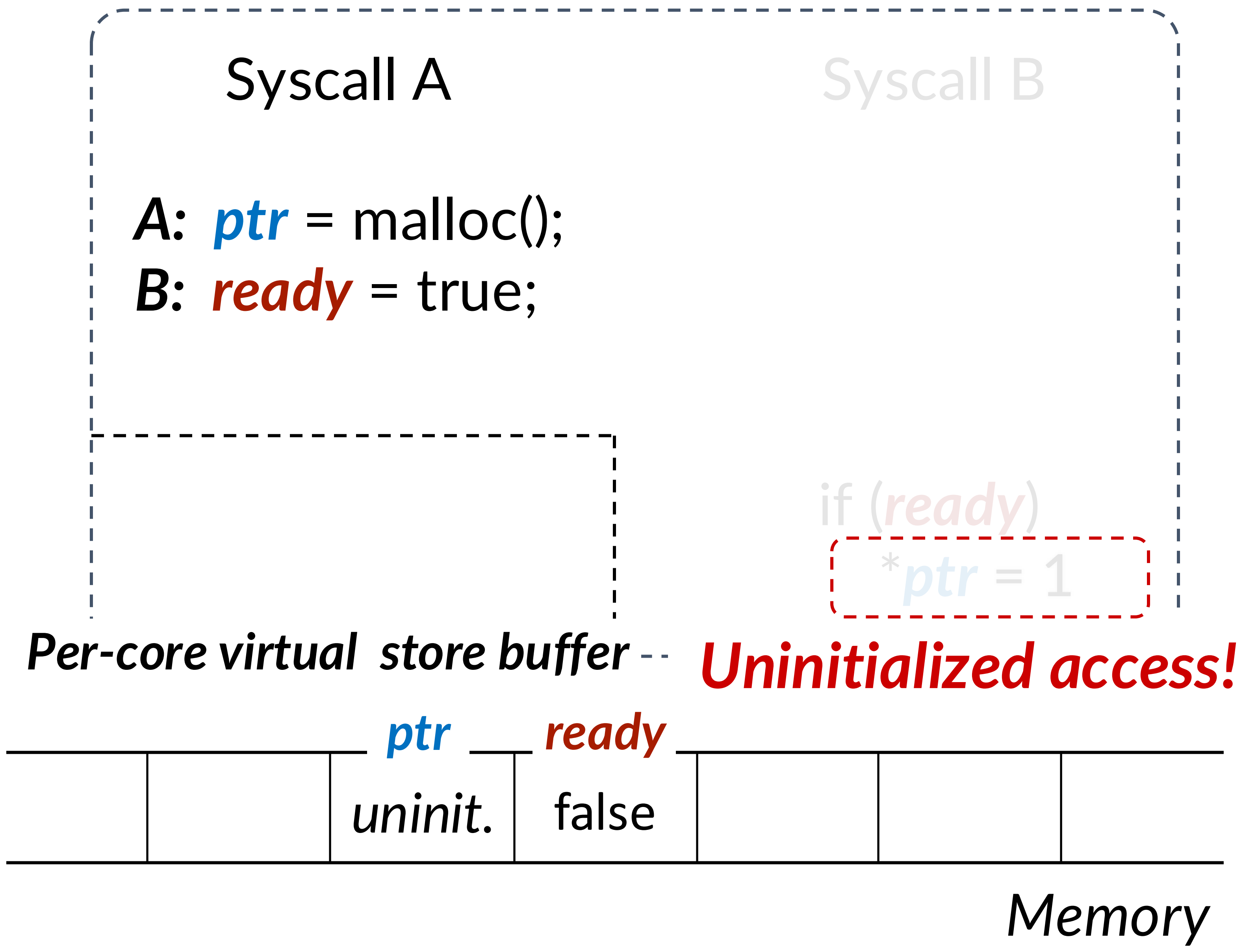
Delayed store operation

through emulating the store buffer

We want to reorder
the exec. order ***B*** **➔** ***A***

We instruct the store buffer to
1) hold the value of ***ptr***
2) flush the value of ***ready***

During runtime…

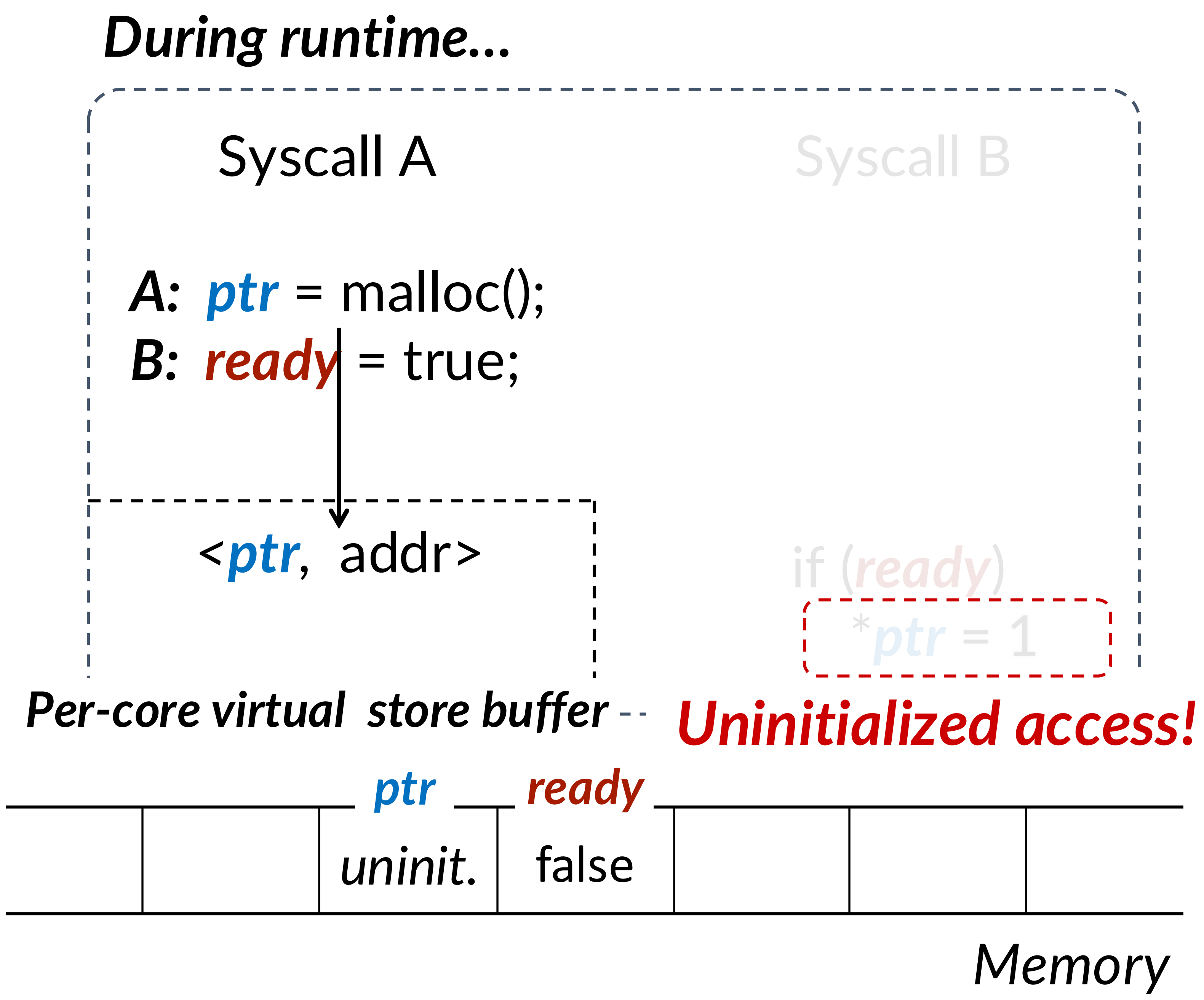


Delayed store operation

through emulating the store buffer

We want to reorder the exec. order ***B ➔ A***

We instruct the store buffer to
1) hold the value of ***ptr***
2) flush the value of ***ready***

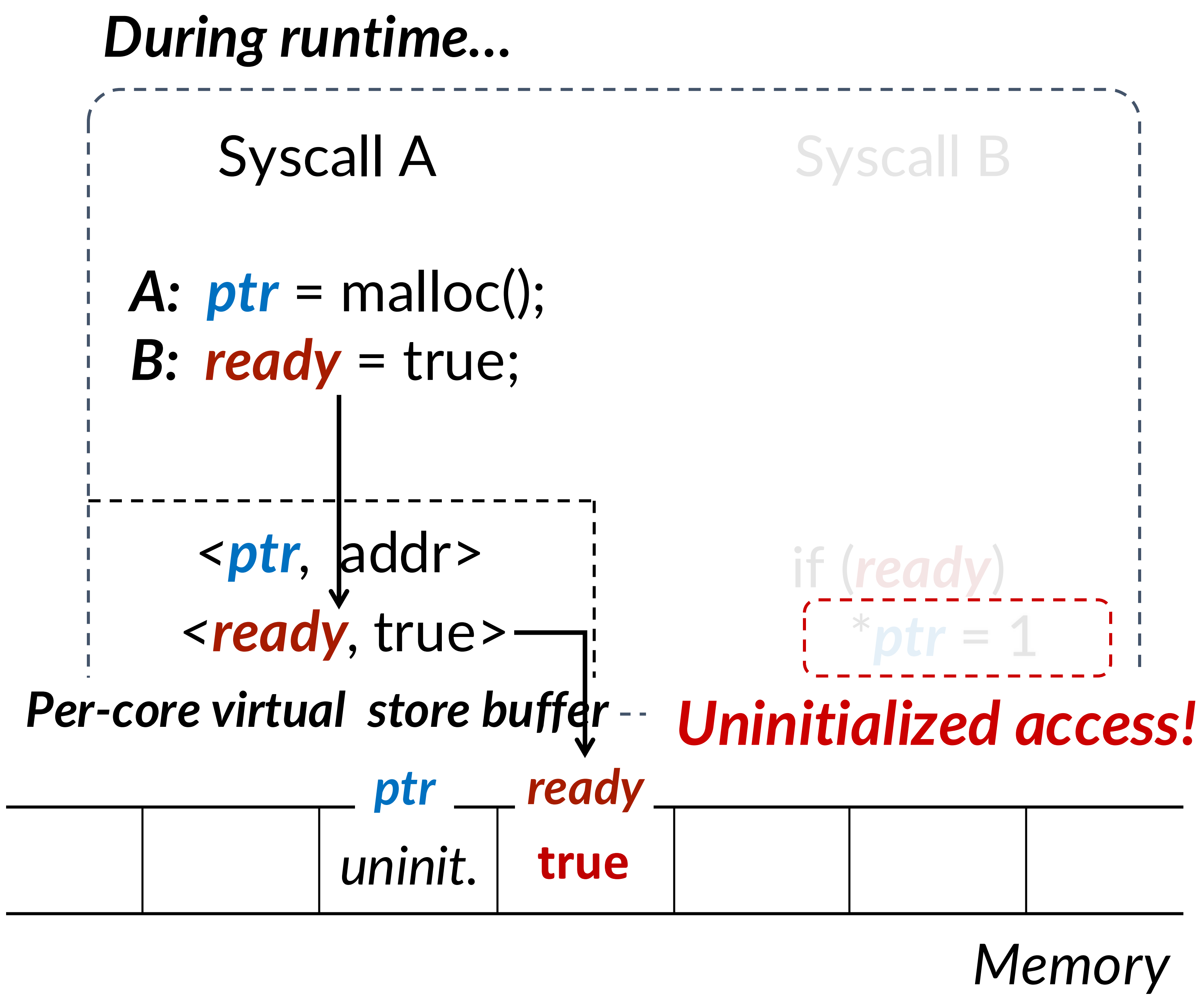


Delayed store operation

through emulating the store buffer

*We want to reorder the exec. order **B ➔ A***

We instruct the store buffer to
*1) hold the value of **ptr***
*2) flush the value of **ready***

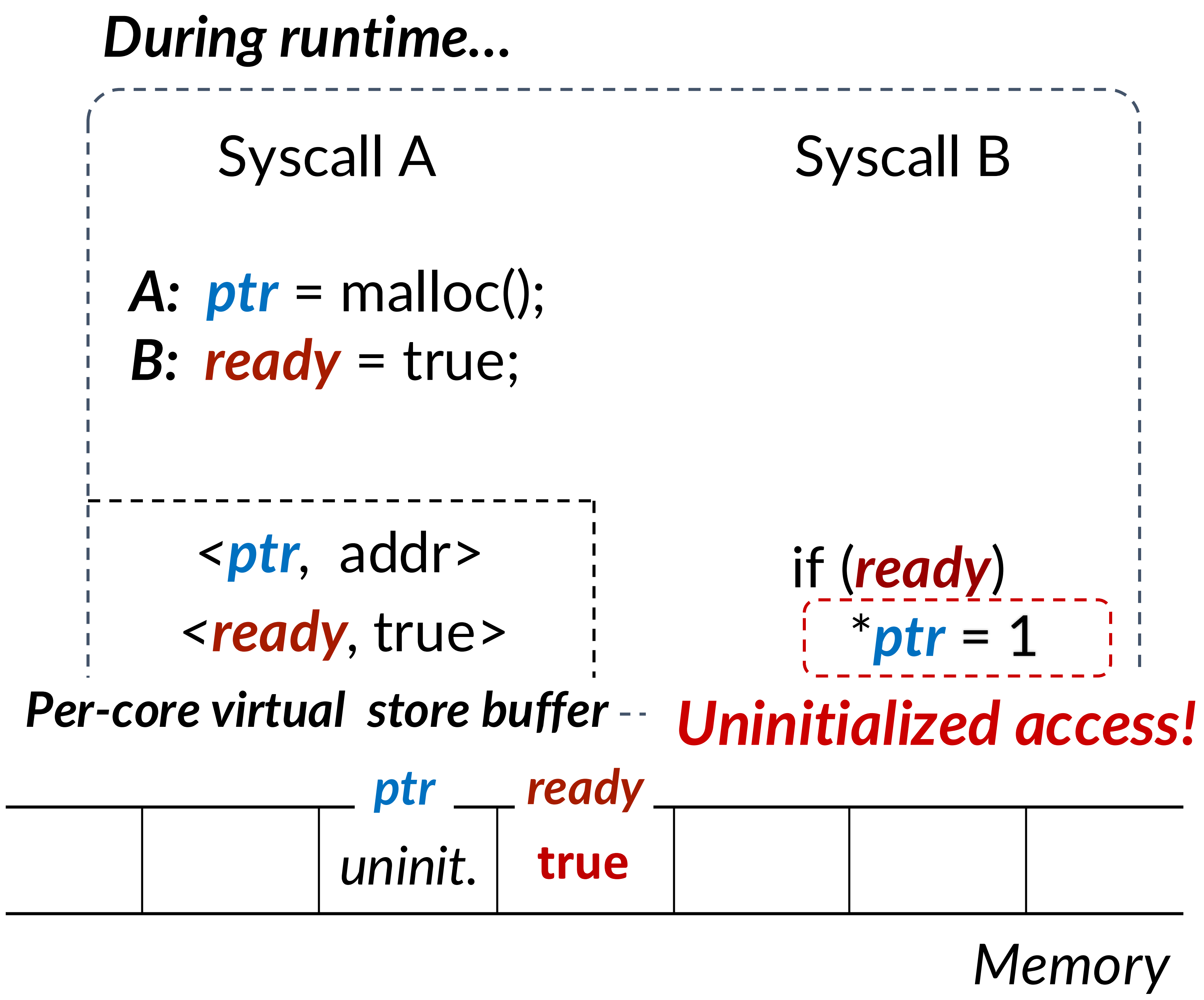


Delayed store operation

through emulating the store buffer

*We want to reorder the exec. order **B → A***

We instruct the store buffer to
*1) hold the value of **ptr***
*2) flush the value of **ready***



Versioned load operation

Genuine architectural behavior

A processor may read memory *ahead of previous instructions*

Ex) reading the address of ***ptr*** before ***ready*** in Syscall B

Syscall A

Syscall B

*Read the invalid addr of **ptr***

```
ptr = malloc();
memory_barrier();
ready = true;
```

```
if (ready)
  *ptr = 1
```

Another memory barrier is needed here!

Uninitialized access!

Versioned load operation

Emulating the architectural behavior

Syscall A

Syscall B

*Record the invalid addr of **ptr***

```
ptr = malloc();
memory_barrier();
ready = true;
```

```
if (ready)
    *ptr = 1
```

*may read **the old and invalid** address of **ptr***

A processor may read memory *ahead of previous instructions*

Ex) reading the address of ***ptr*** before ***ready*** in Syscall B

A versioned load operation emulates this hardware behavior

- It allows a load operation to read an ***old version*** of the value

OEMU manages multiple versions of a value

Please check the paper for detail!

In this work, we introduce…

OEMU

- A mechanism to tame the non-deterministic behavior of out-of-order execution during runtime

Ozz

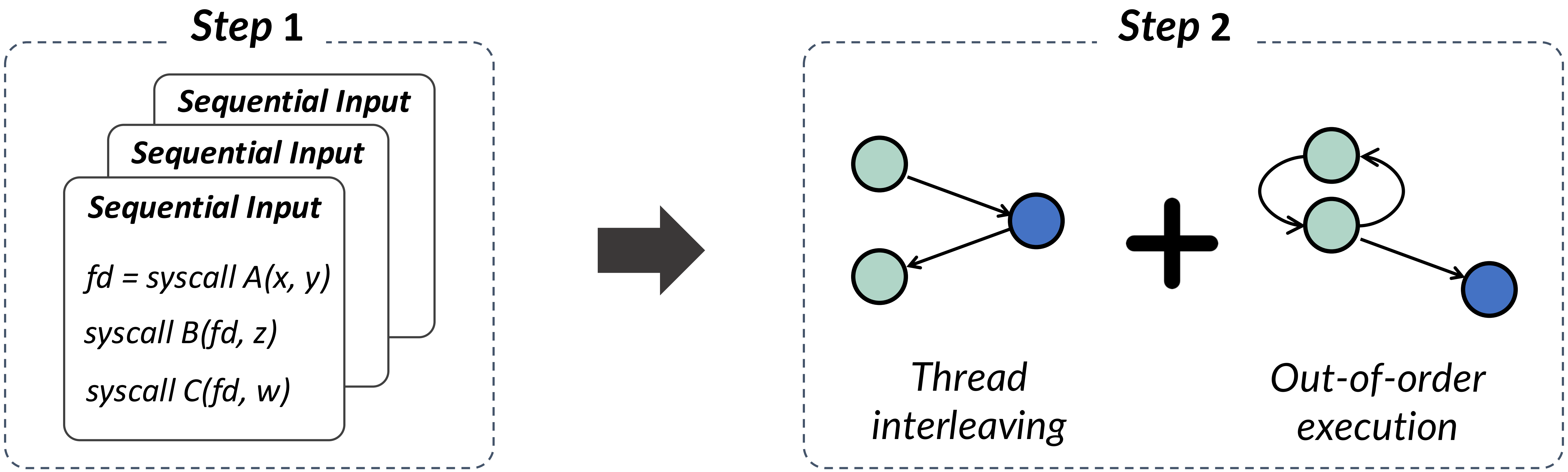
- A kernel fuzzer tailored to find OoO bugs by deterministically controlling
  - ***Out-of-order execution*** through OEMU, and
  - ***Thread interleaving*** through a custom scheduler from a previous work[1]

1: Jeong, Dae R., Byoungyoung Lee, Insik Shin, and Youngjin Kwon. "Segfuzz: Segmentizing thread interleaving to discover kernel concurrency bugs through fuzzing." In *2023 IEEE Symposium on Security and Privacy (SP).*

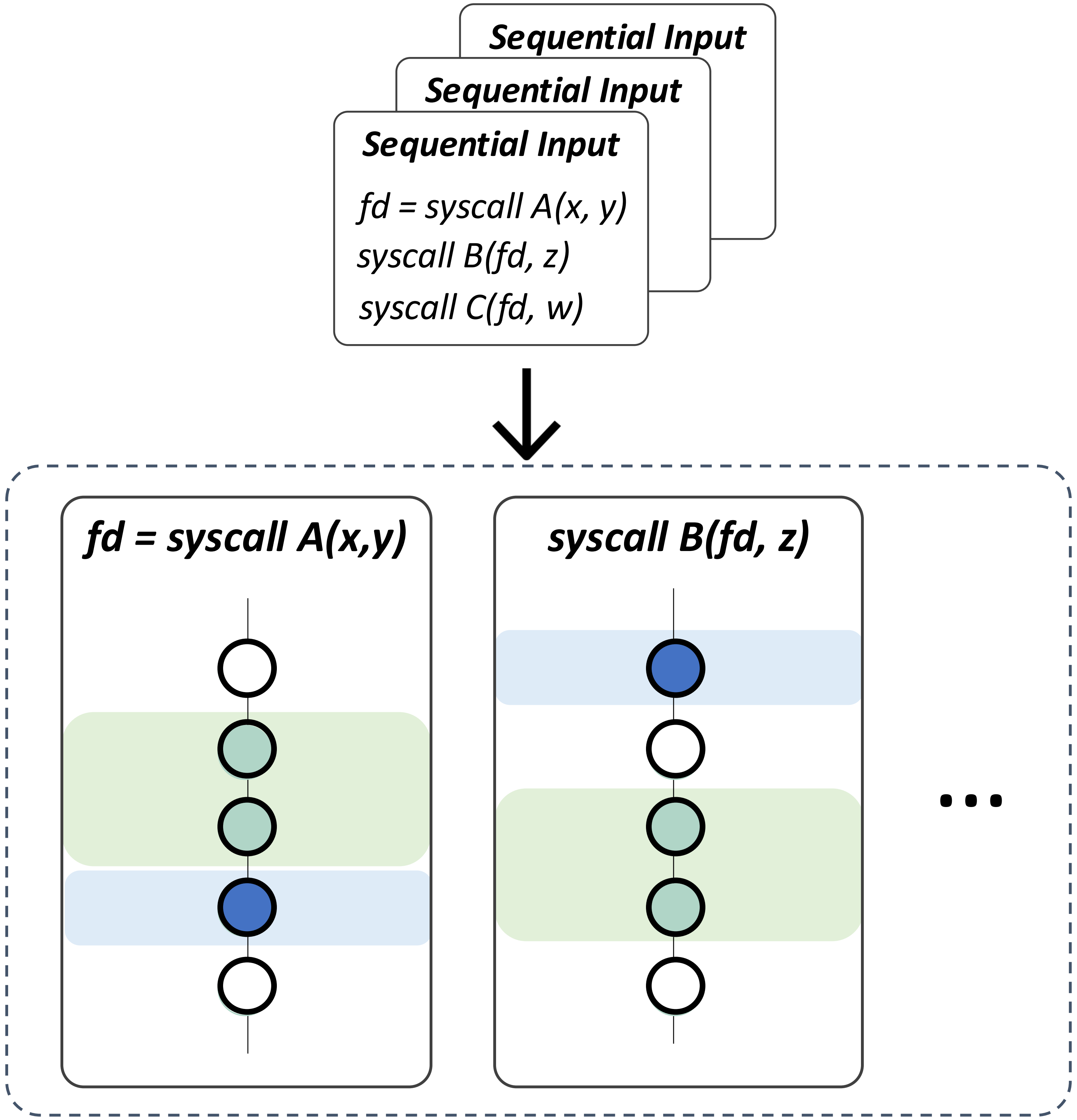
Ozz

A kernel fuzzer tailored to identify OoO bugs through two steps

- **Step 1:** Running single-threaded inputs to dynamically profile memory accesses
- **Step 2:** Running multi-threaded inputs to find OoO bugs



Step 1: Profiling memory accesses



Ozz utilizes fuzzing to generate sequential inputs

- Exploring execution paths as much as possible
- Dynamically tracing memory accesses of system calls

Select system call pairs accessing shared memory objects

- Ozz will run them concurrently in Step 2

Step 2: Finding OoO bugs

Syscall A

Syscall B

ptr = malloc();
memory_barrier();
ready = true;

If a memory barrier existed,
the store to **ptr** would not be delayed

if (***ready***)
 ****ptr*** = 1

Uninitialized access!

1. *Guess where a memory barrier is missing*

How?

2. *Execute instructions in a way that would* ***not happen*** *if the memory barrier existed*

3. *Observe whether the kernel malfunctions*

Step 2: Finding OoO bugs

Guess where a memory barrier is missing

Maximizing the number of reordered memory accesses

Syscall A

Case 1 → (between `ptr->len = len;` and `ptr->ops = &ops;`)

Case 2 → (between `ptr->ops = &ops;` and `ready = true;`)

```
ptr->len = len;
ptr->ops = &ops;
ready = true;
```

Syscall B

```
if (ready) {
    len = ptr->len;
    ptr->ops->confirm(len);
}
```

The more execution deviates from a sequential order, the harder it becomes to reason about

Step 2: Finding OoO bugs

Guess where a memory barrier is missing

Maximizing the number of reordered memory accesses

Syscall A

Case 1 →

```
ptr->len = len;
ptr->ops = &ops;
ready = true;
```

Delayed by OEMU

Enforced by the custom scheduler

Syscall B

```
if (ready) {
    len = ptr->len;
    ptr->ops->confirm(len);
}
```

Step 2: Finding OoO bugs

Guess where a memory barrier is missing

Maximizing the number of reordered memory accesses

Syscall A

```
ptr->len = len;
ptr->ops = &ops;
ready = true;
```

Case 2 →

Delayed by OEMU

Enforced by the custom scheduler

Syscall B

```
if (ready) {
    len = ptr->len;
    ptr->ops->confirm(len);
}
```

In Case2, more memory accesses are reordered than in Case 1
Ozz prioritizes Case 2 as it is harder for developers to reason about

Evaluation

Finding unknown bugs / reproducing known bugs

We found 11 new OoO bugs in the Linux kernel

- Some were found in popular subsystemssuch as TLS or eBPF
- We reported all of them, and they were accordingly patched by the kernel developers

Subsystem	Summary
RDS	KASAN: slab-out-of-bounds Read in rds_loop_xmit
watchqueue	BUG: unable to handle kernel NULL pointer dereference in _find_first_bit
VMCI	general protection fault in add_wait_queue
XDP	BUG: unable to handle kernel NULL pointer dereference in xsk_poll
TLS	BUG: unable to handle kernel NULL pointer dereference in tls_getsockopt
BPF	BUG: unable to handle kernel NULL pointer dereference in sk_psock_verdict_data_ready
XDP	BUG: unable to handle kernel NULL pointer dereference in xsk_generic_xmit
SMC	BUG: unable to handle kernel NULL pointer dereference in connect
TLS	BUG: unable to handle kernel NULL pointer dereference in tls_setsockopt
SMC	KASAN: null-ptr-deref Write in fput
GSM	BUG: unable to handle kernel NULL pointer dereference in gsm_dlci_config

Evaluation

Finding unknown bugs / reproducing known bugs

We found 11 new OoO bugs in the Linux kernel

- Some were found in popular subsystemssuch as TLS or eBPF
- We reported all of them, and they were accordingly patched by the kernel developers

We show OMEU/Ozz can reproduce 8 out of 9 known OoO bugs

- The one failing case involves another non-deterministic behavior, *thread migration*

Please check our paper for more evaluation

Conclusion

Our work introduces

- **OEMU**
  - A mechanism to tame the non-deterministic behavior of out-of-order execution during runtime

- **Ozz**
  - A kernel fuzzer tailored to find OoO bugs by deterministically controlling
    - ***Out-of-order execution*** through OEMU, and
    - ***Thread interleaving*** through a custom scheduler from a previous work

Ozz finds 11 new out-of-order concurrency bugs in the Linux kernel

Ozz: Identifying Kernel Out-of-Order Concurrency Bugs with In-Vivo Memory Access Reordering

Q&A