black hat
USA 2020

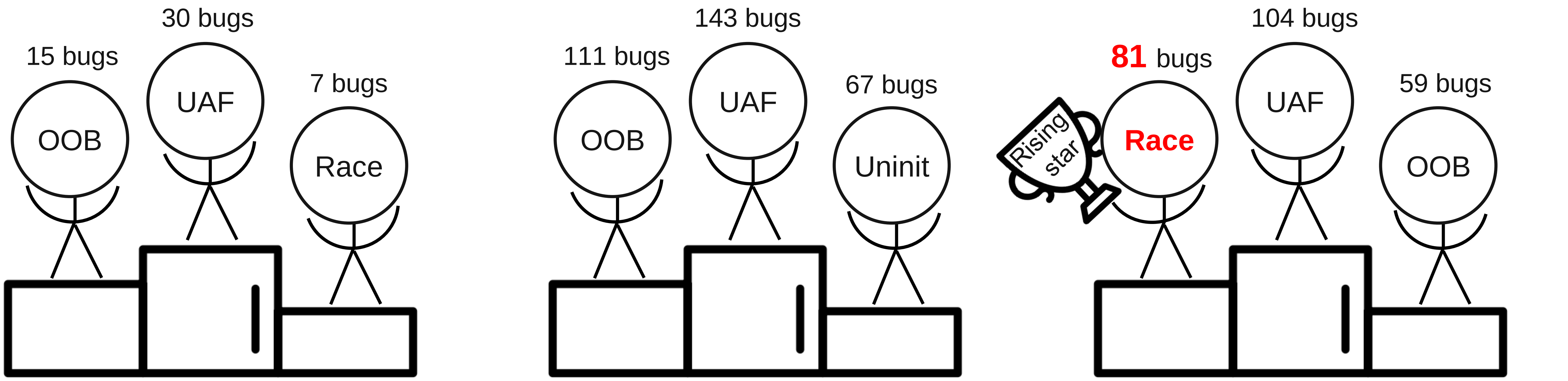
AUGUST 5-6, 2020
BRIEFINGS

Exploting Kernel Races Through Taming Thread Interleaving

Yoochan Lee, Byoungyoung Lee, Chanwoo Min

Seoul National University, Virginia Tech

Race condition is an increasing attack vector



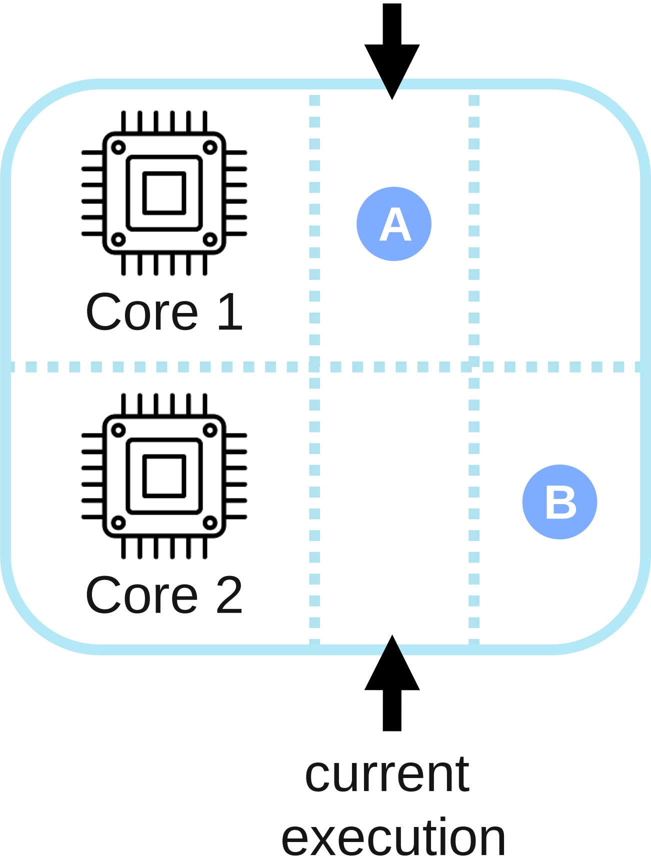
# of fixed bugs that Syzkaller found in **2017**

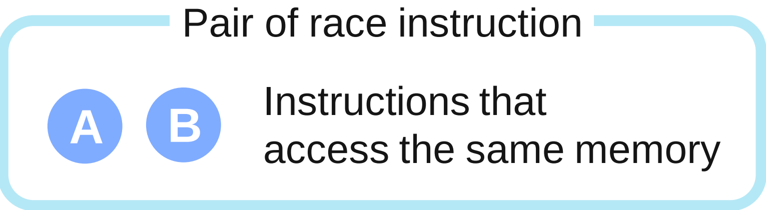
# of fixed bugs that Syzkaller found in **2018**

# of fixed bugs that Syzkaller found in **2019**

- Race Condition is gaining strong attention from the security community.
- Razzer, IEEE S&P 2019, found more than **30 race bugs**.
- KCSAN, developed by Google 2019, found more than **300 race bugs**.

Background : Race condition

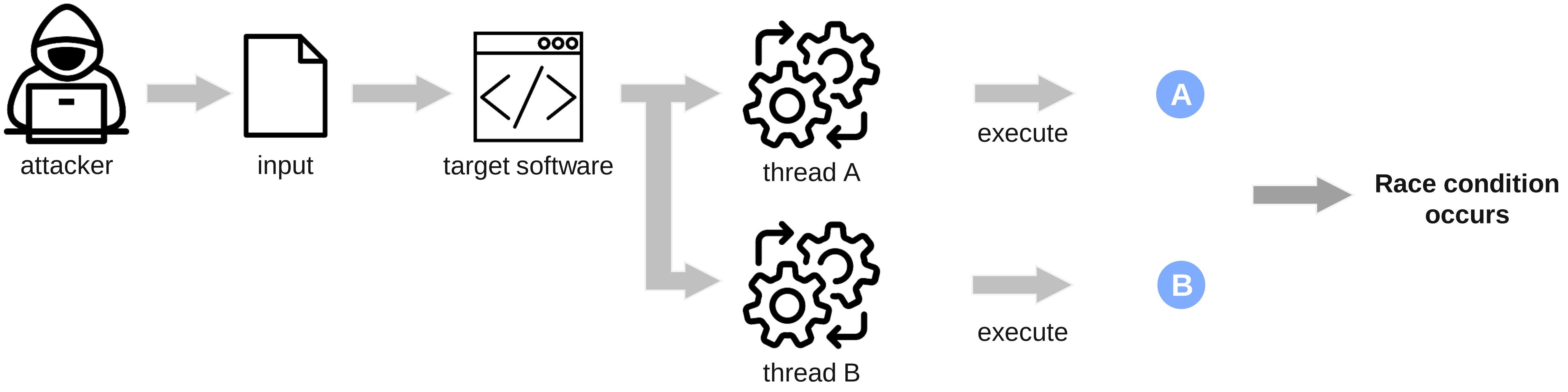




- **Accessing the same memory** location from two processor

➔ **the results are different** according to access order.

Background : Two Conditions for Triggering Race



attacker

input

target software

thread A

thread B

execute

execute

A

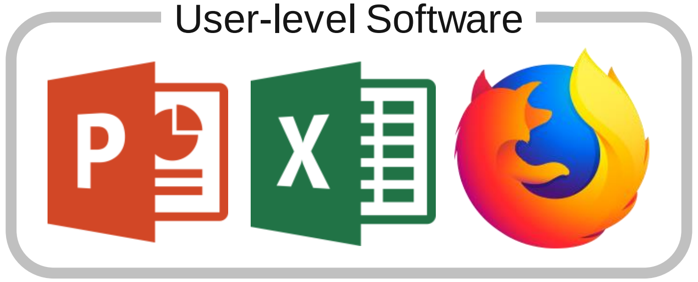
B

Race condition occurs

1. Can **create threads**

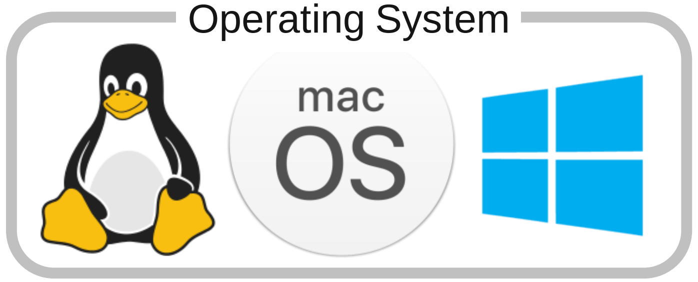
2. Can **execute race instructions** on each threads

User-level Software



Difficult to meet two conditions

Operating System



Create two threads, where each executes syscalls

Background : Race Condition Vulnerability

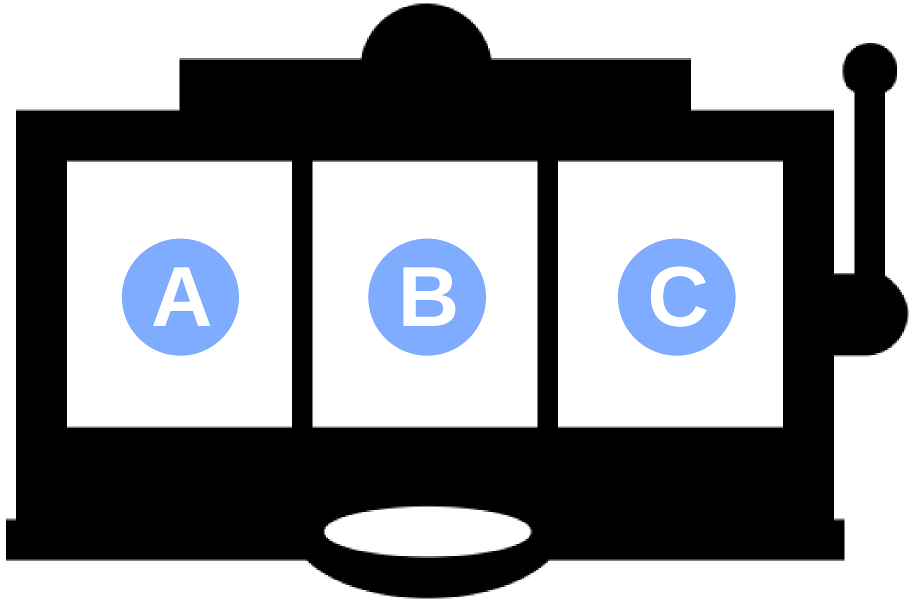
if A >> B >> C , then

Race Condition Vulnerability = **Race Condition** + **Memory Corruption**

Race Condition:
- Race instruction pair A
- Race instruction pair B
- .
- .
- .

Memory Corruption:
- Overflow
- Use-After-Free
- .
- .
- .

Background : to trigger Race Condition Vulnerability

if  , then **memory corruption** occurs.

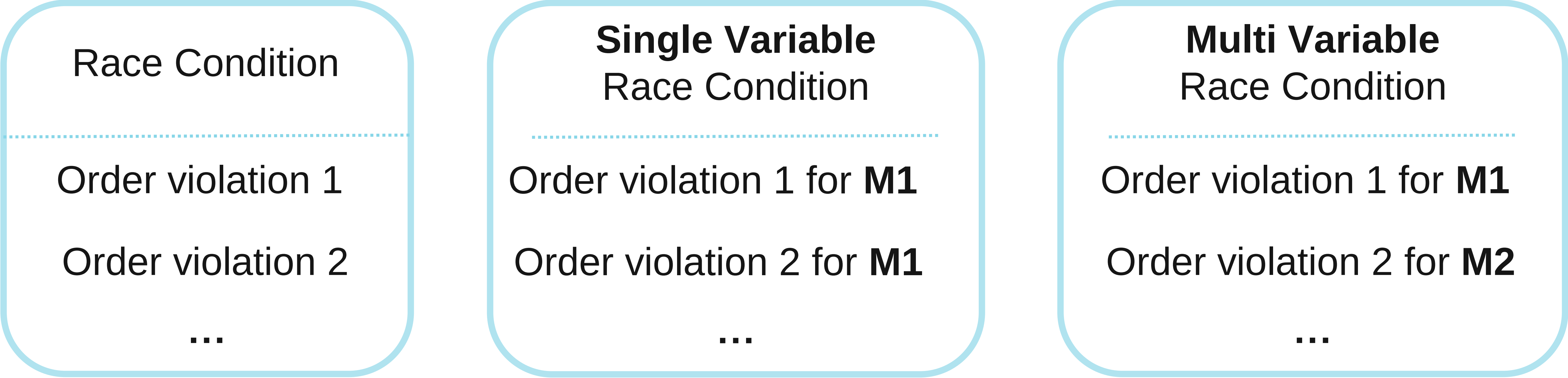
Brute forcing :

Try until success

Background : Exploitability of Race Condition Vulnerability

Is Race Condition Vulnerability Exploitable? = **A very specific memory access order** + Availability of Memory Corruption

Classification of Race Condition Vulnerability



Race Condition

Order violation 1

Order violation 2

...

Single Variable
Race Condition

Order violation 1 for **M1**

Order violation 2 for **M1**

...

Multi Variable
Race Condition

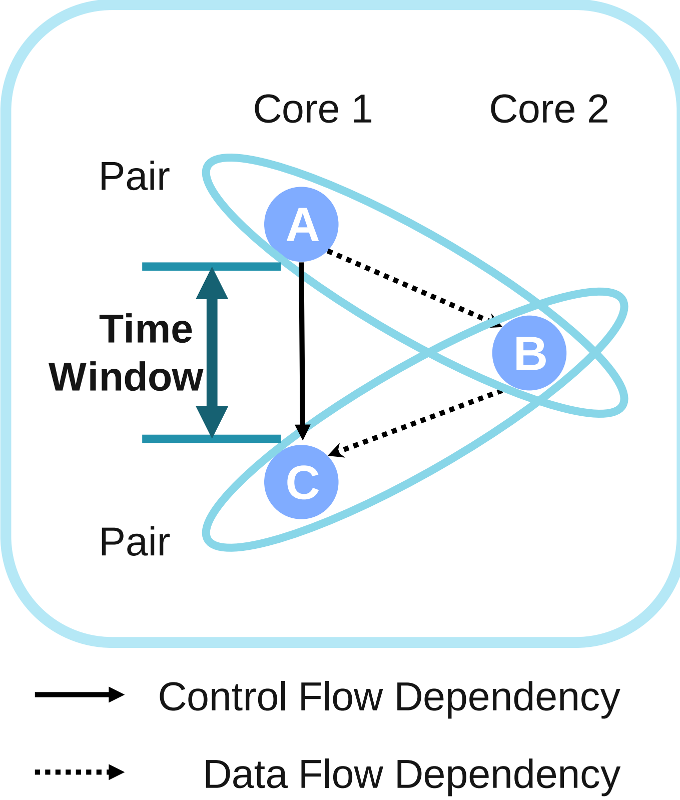
Order violation 1 for **M1**

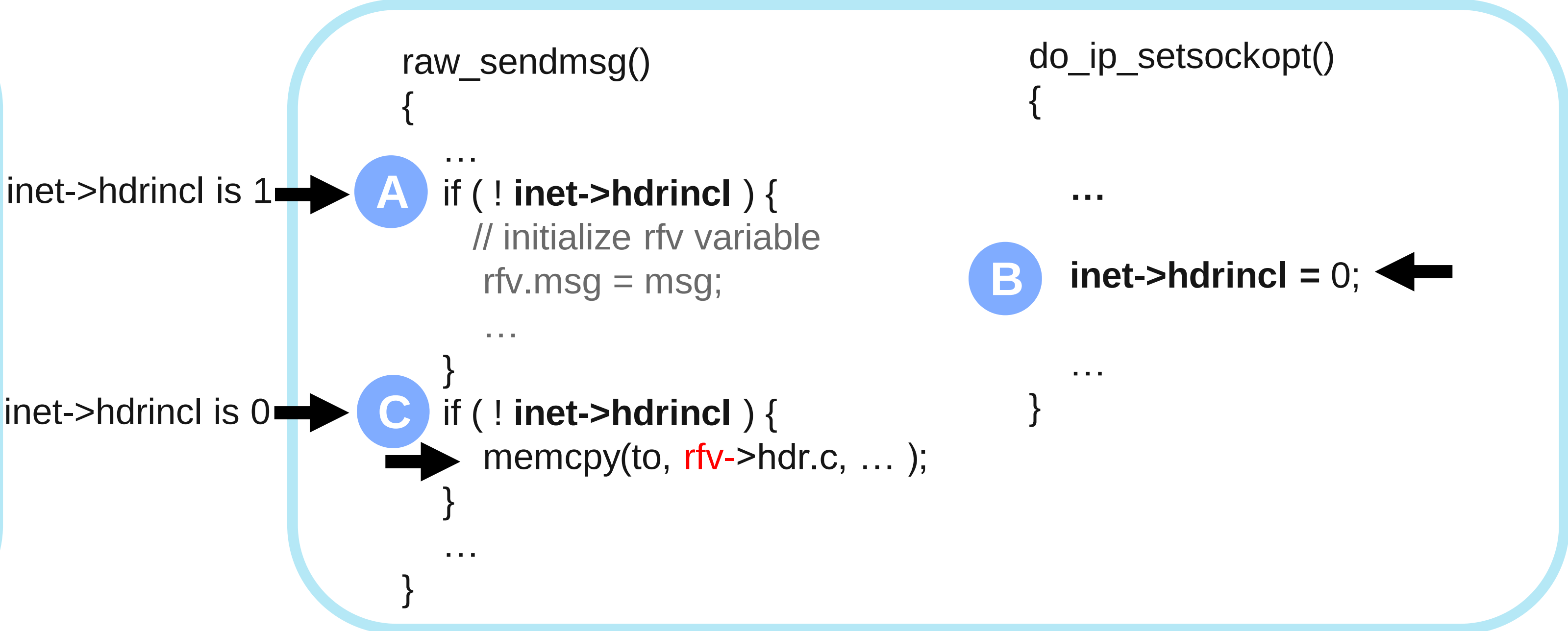
Order violation 2 for **M2**

...

- As mentioned earlier, race conditions consist of **multiple order-violations**.
- Order violations can occur only for **one variable** or **multiple variables**.

Single-variable Race Condition





```
raw_sendmsg()
{
    …
    if ( ! inet->hdrincl ) {
      // initialize rfv variable
       rfv.msg = msg;
       …
    }
    if ( ! inet->hdrincl ) {
       memcpy(to, rfv->hdr.c, … );
    }
    …
}
```

```
do_ip_setsockopt()
{

     …

     inet->hdrincl = 0;

     …
}
```

Case study : CVE-2017-17712

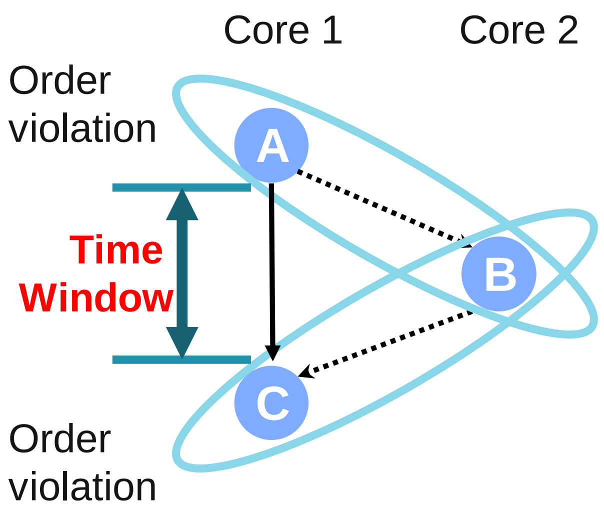
if A >> B >> C , then uninitialized buffer use occurs.

- Single-variable race condition consists of more than one race pairs related to **single variable** (Most of bugs consist of two order violation).

Exploitability of Single-variable Race

No matter how low the probability,

it is **not zero**.

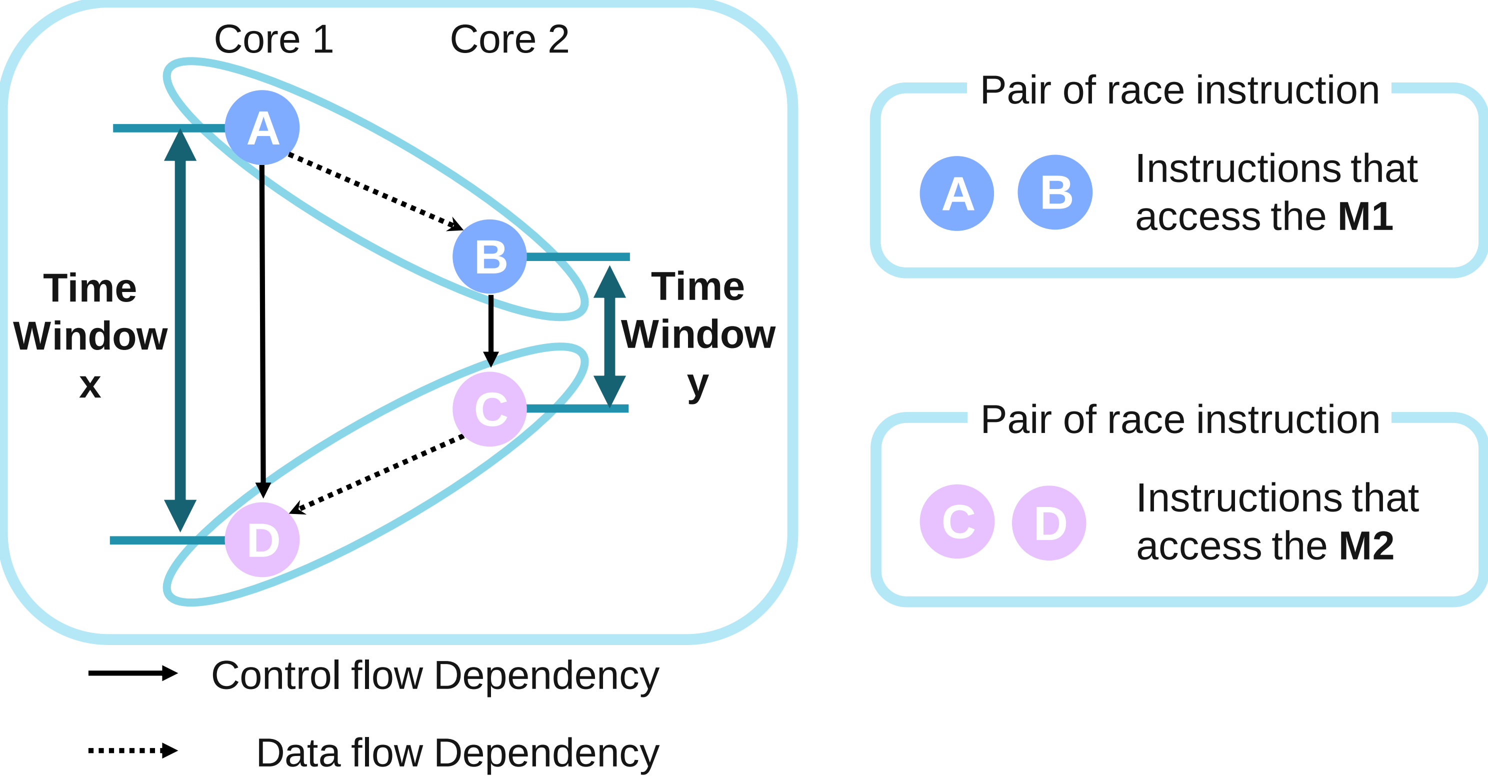


Only consider the

availability of memory corruption

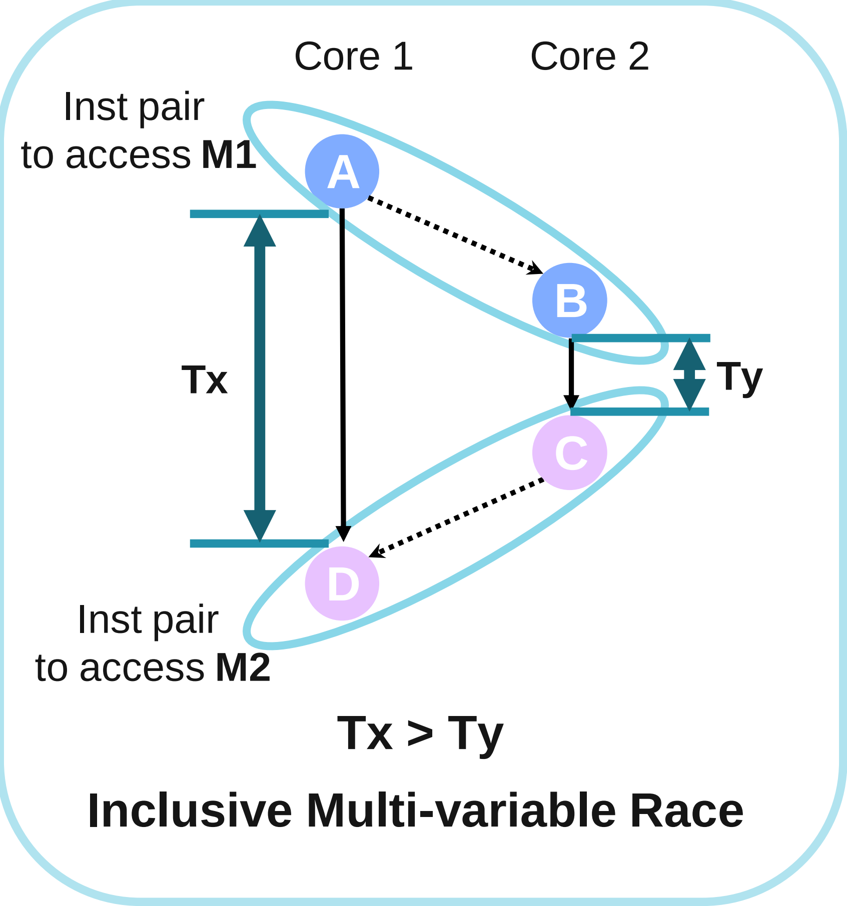
- The smaller the time window is, the lower the probability of race condition occurring.

Multi-variable Race Condition



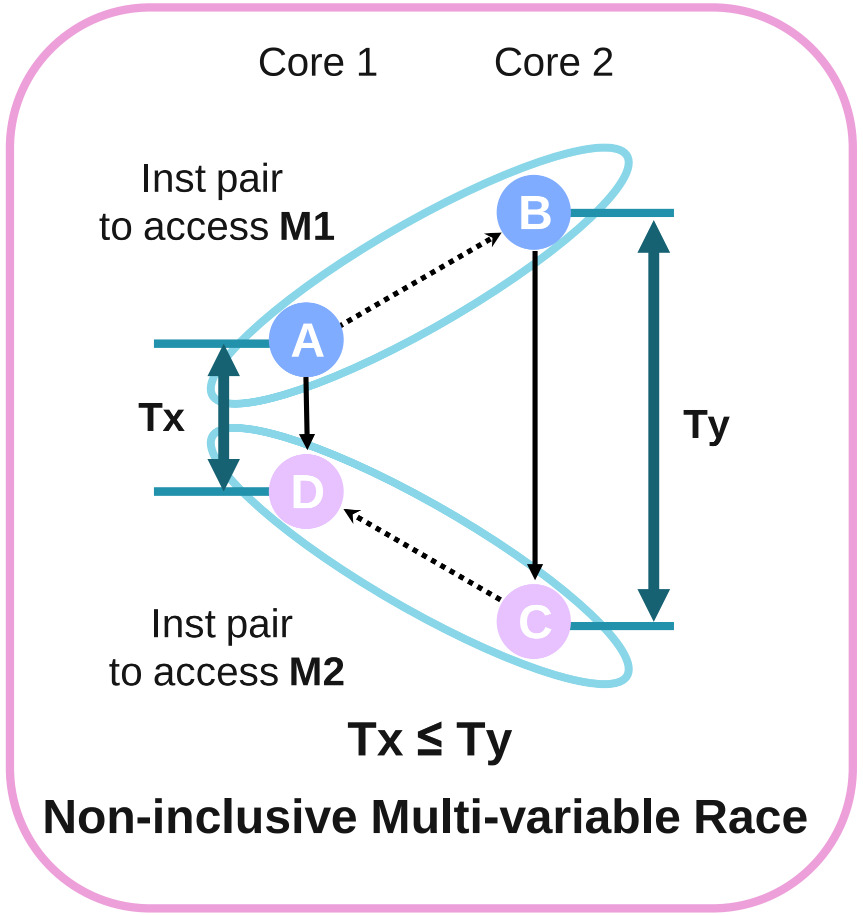
- Multi-Variable race condition consists of more than one race pairs, each race pair is related to a **different variable**.

Multi-variable Race Condition



Tx > Ty

Inclusive Multi-variable Race



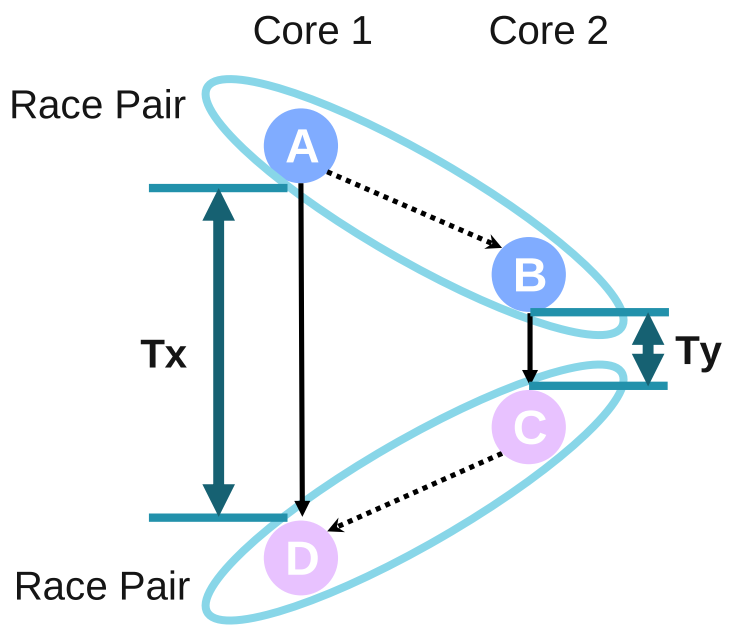
Tx ≤ Ty

Non-inclusive Multi-variable Race

Exploitability of Inclusive Multi-variable Race

No matter how low the probability,

it is **not zero**.

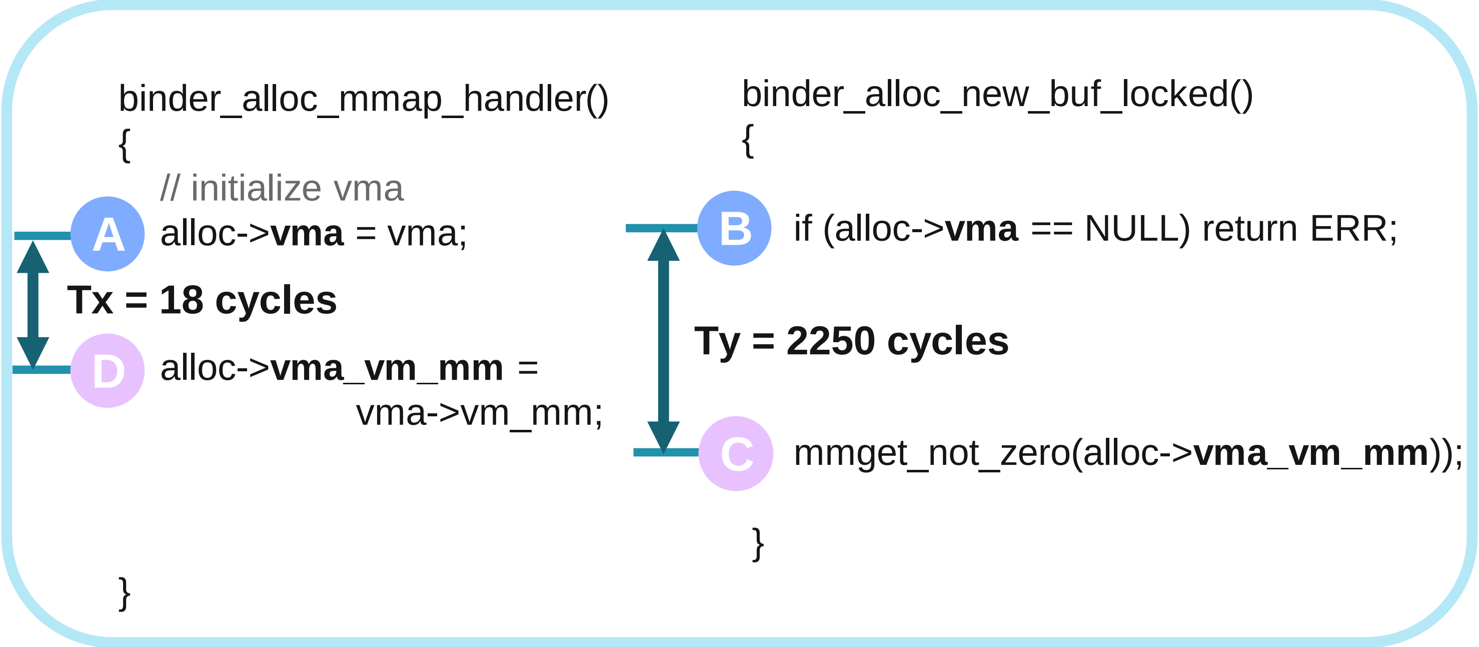


Only consider the

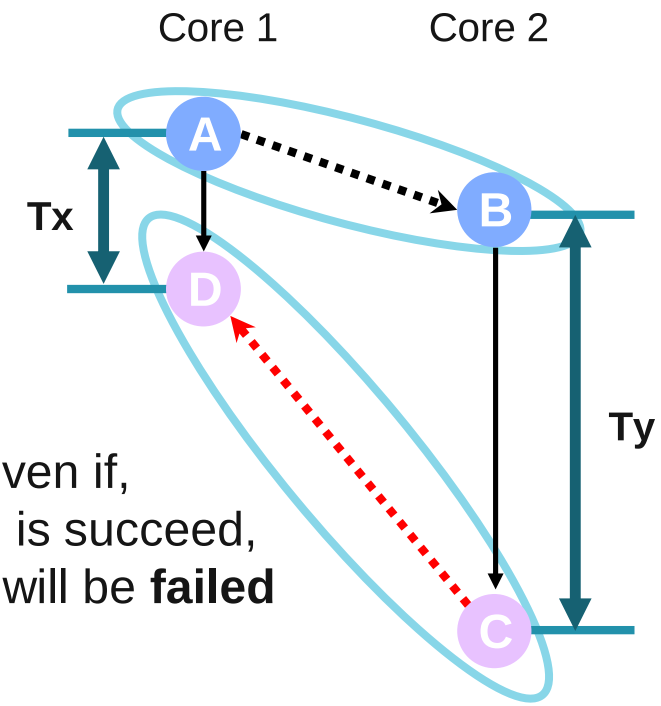
availability of memory corruption

- The more similar the two time windows are, the lower the probability that a race will occur.

Problem : Exploitability of Non-inclusive Race



```
binder_alloc_mmap_handler()
{
    // initialize vma
A   alloc->vma = vma;
    Tx = 18 cycles
D   alloc->vma_vm_mm =
                 vma->vm_mm;

}

binder_alloc_new_buf_locked()
{
B   if (alloc->vma == NULL) return ERR;

    Ty = 2250 cycles

C   mmget_not_zero(alloc->vma_vm_mm));

}
```



Even if,
A >> B is succeed,
C >> D will be **failed**

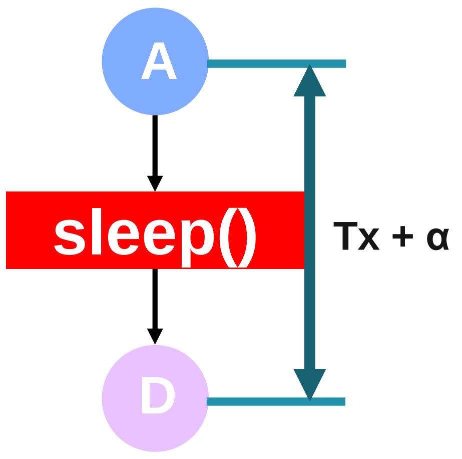
Case study : Patch #987393

if A >> B && C >> D , then uninitialized buffer use occurs in C.

- **impossible to physically execute** this type of race condition in the order of A >> B and C >> D.

Previous Approach : Using Debugging Feature

```
void race_function1()
{
```



```
}
```

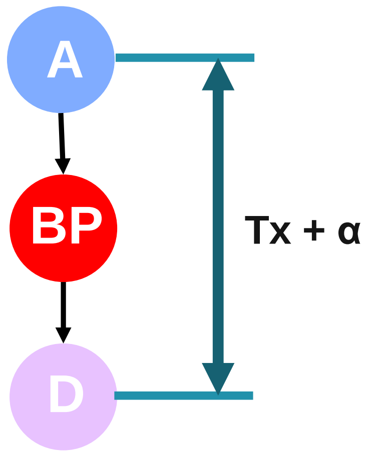
Insert sleep function

Modifying the kernel

GDB

Using debugger

```
void race_function1()
{
```

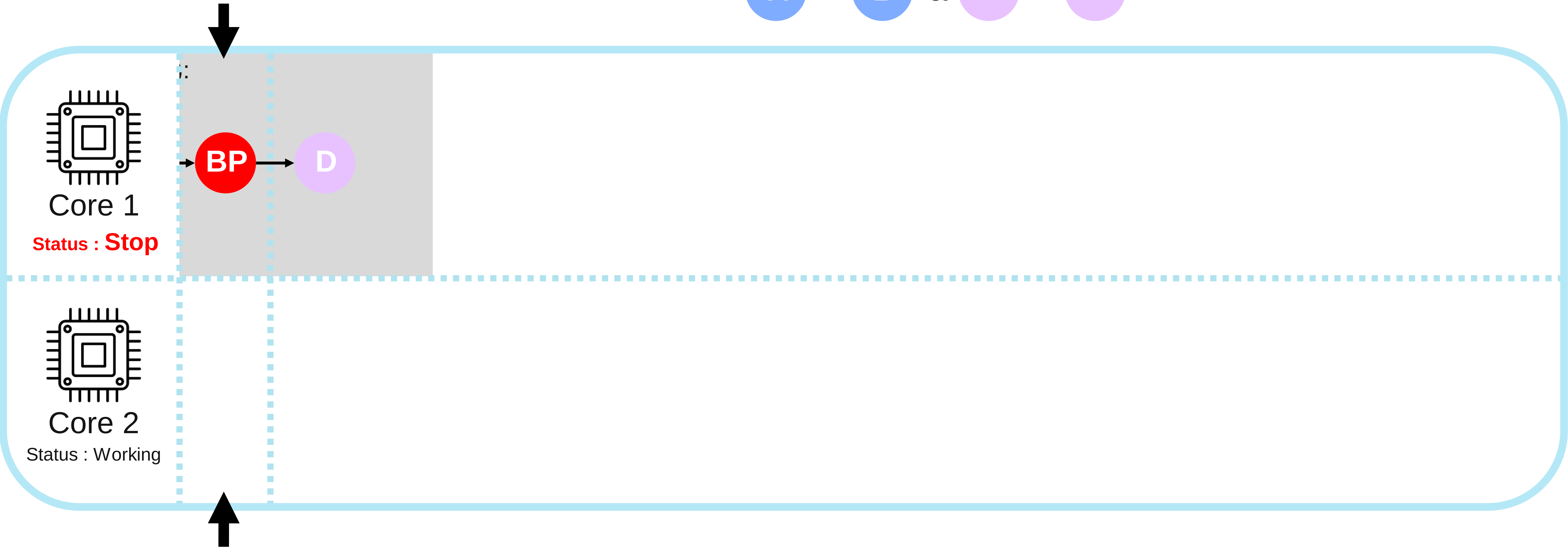


```
}
```

Insert breakpoint

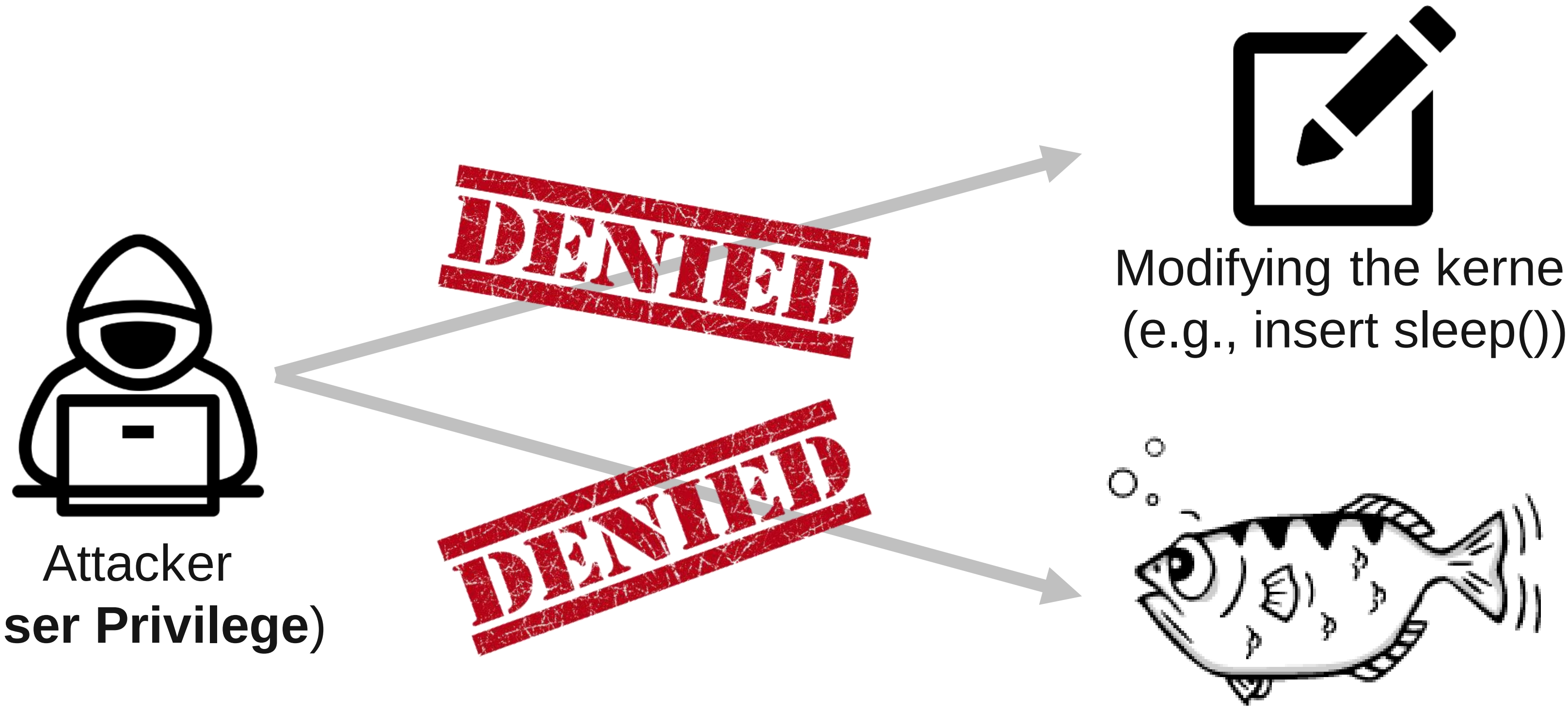
Previous Approach : Using Debugging Feature

Execution Order : A >> B & C >> D



current execution

Limitation of Using Debugging Feature



Attacker
(**User Privilege**)

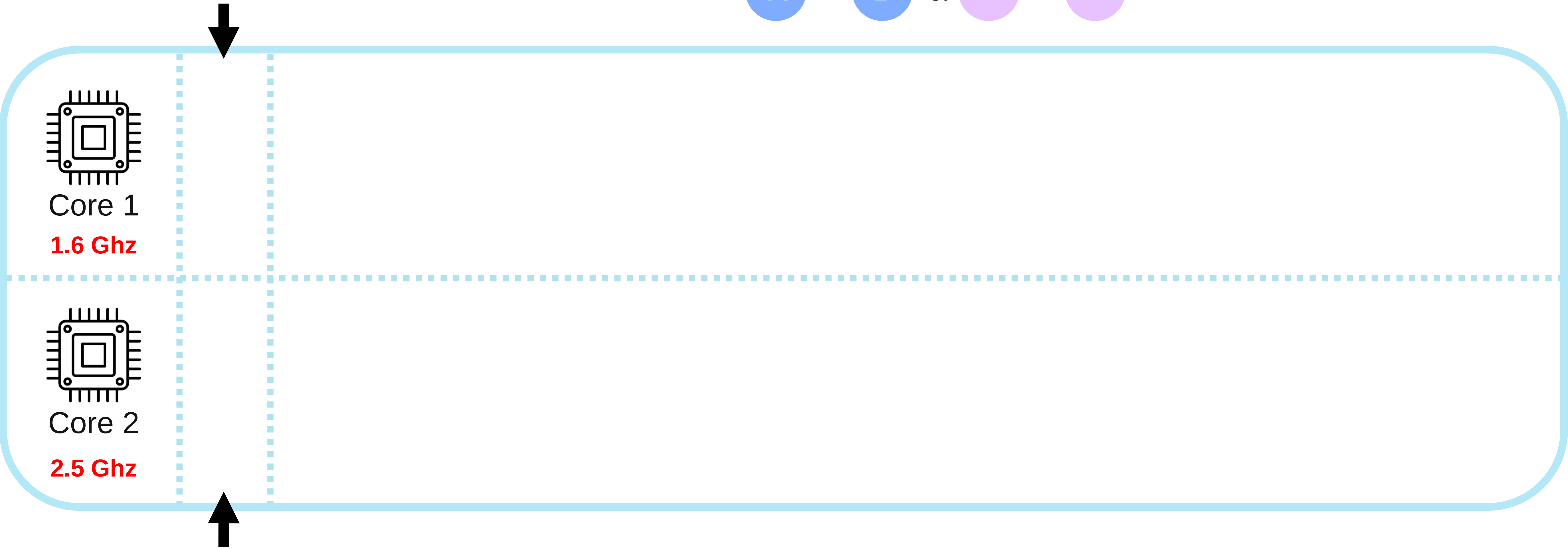
Modifying the kernel
(e.g., insert sleep())

GDB

Using debugger
Insert breakpoint

Previous method : Using Different Core Latency

Execution Order : A >> B & C >> D



- e.g., Qualcomm Snapdragon 845 4x 2.5GHz, 4x 1.6GHz

Limitations of Use Different Core Latency

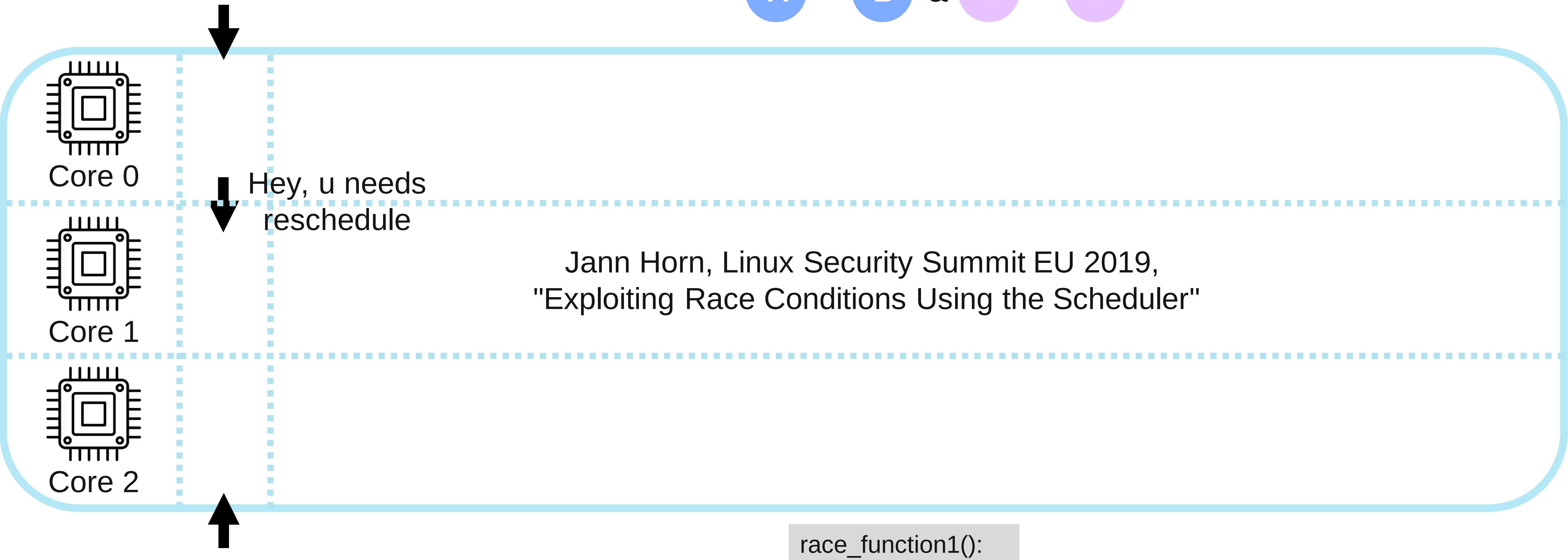
CPU

CPU dependency

- **must use the CPU** that latency between the cores are different.
- Not applicable to vulnerabilities with large time window differences

Previous Approach : Using scheduler (CONFIG_PREEMPT)

Execution Order : A >> B & C >> D



Core 0

Core 1

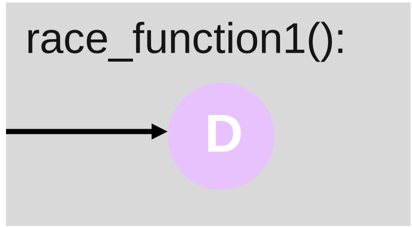
Core 2

Hey, u needs reschedule

Jann Horn, Linux Security Summit EU 2019, "Exploiting Race Conditions Using the Scheduler"

current execution

Wait queue :



race_function1():

Limitation of Using scheduler

Configuration dependency

- Can be used when COFIG_PREEMPT option is applied.
- Linux apply **CONFIG_PREEMPT_VOLUTARY** option **as default.**

Each of methods has obvious limitations

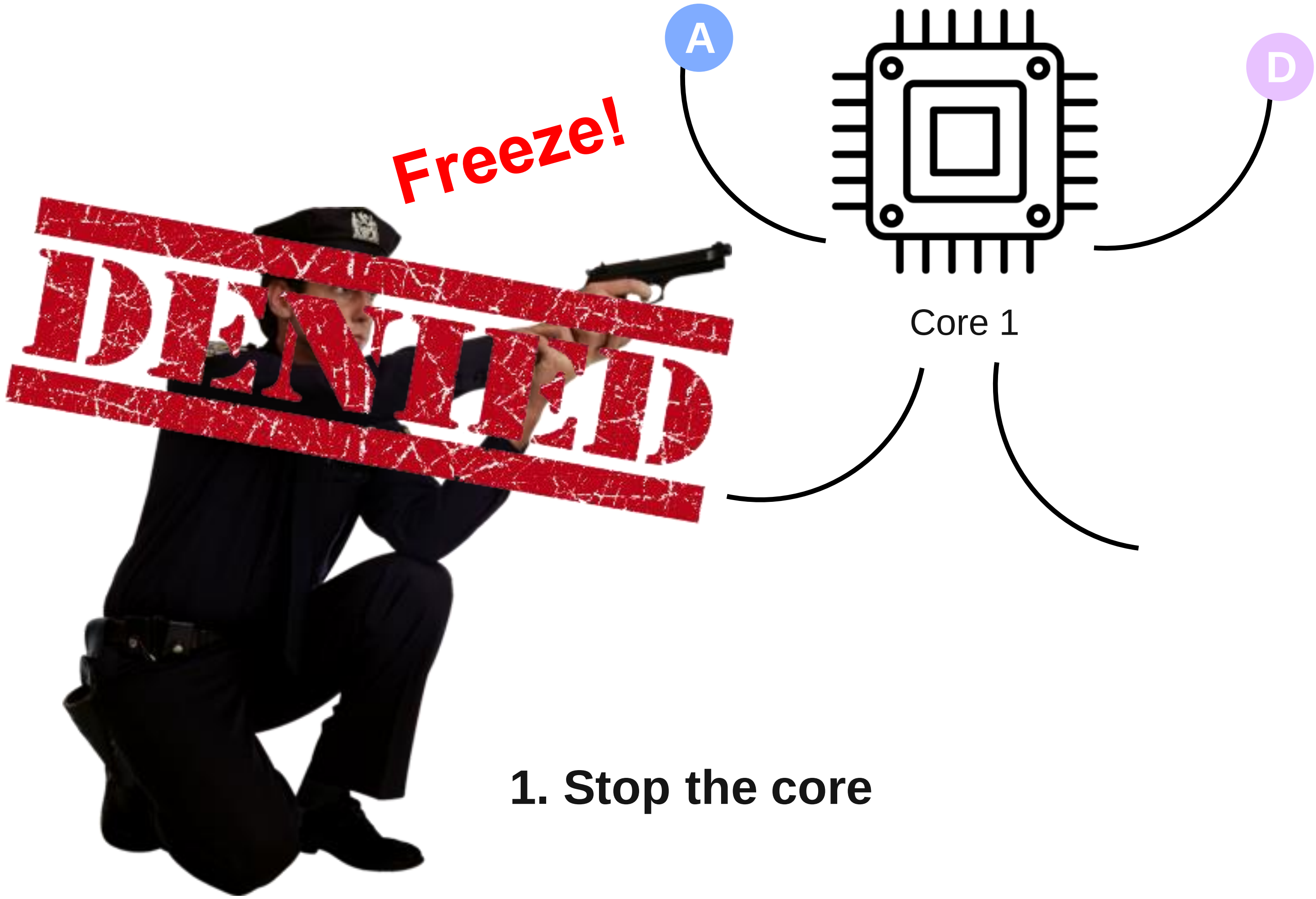
Attacker
(**User Privilege**)

CPU dependency

Configuration dependency

- All of the methods are hard to applied in general.
- We needs **a new method** that extend the race window and can be used **in general.**

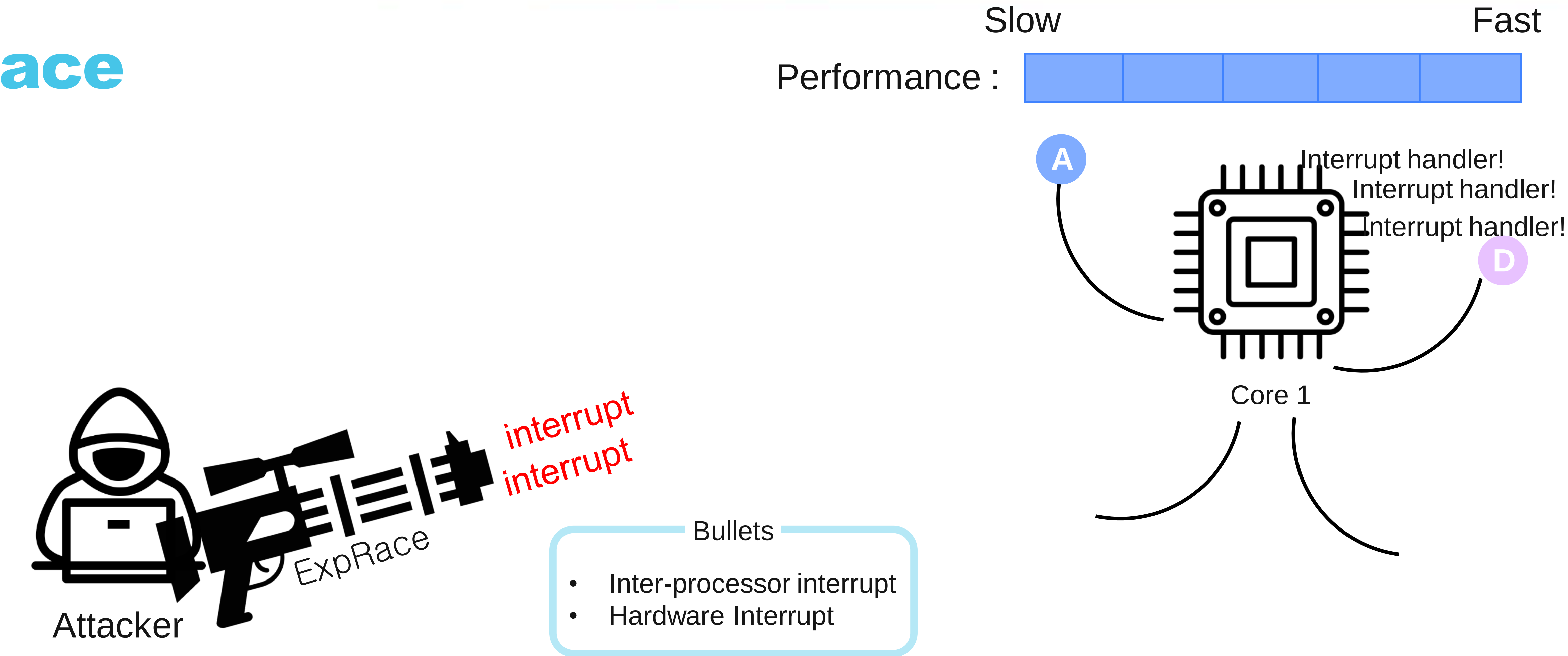
How to extend the race window?



1. Stop the core

2. Degrade the performance

ExpRace

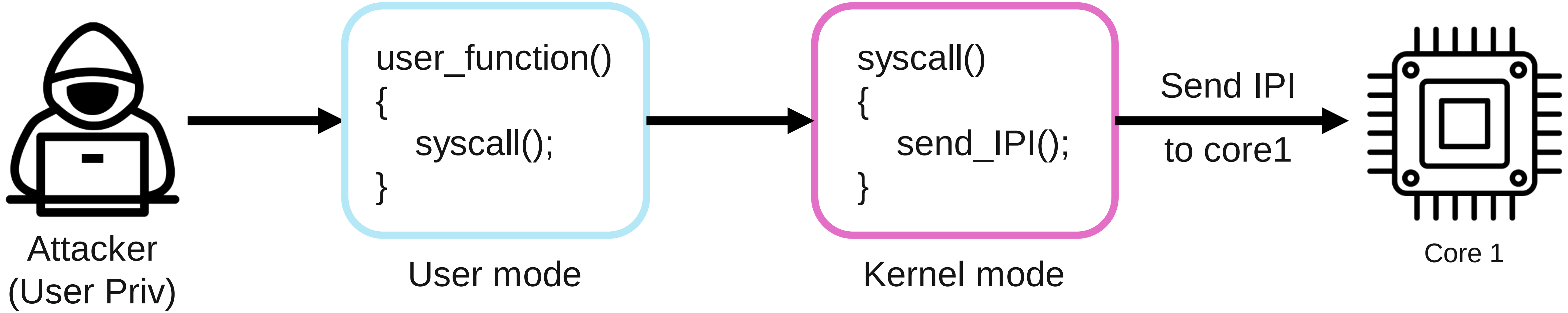


Bullets

- Inter-processor interrupt
- Hardware Interrupt

- The key idea of EXPRACE is **to keep raising interrupts** to indirectly alter kernel thread’s interleaving.

ExpRace : How to send IPI & IRQ with user priv



Attacker (User Priv)

```
user_function()
{
    syscall();
}
```

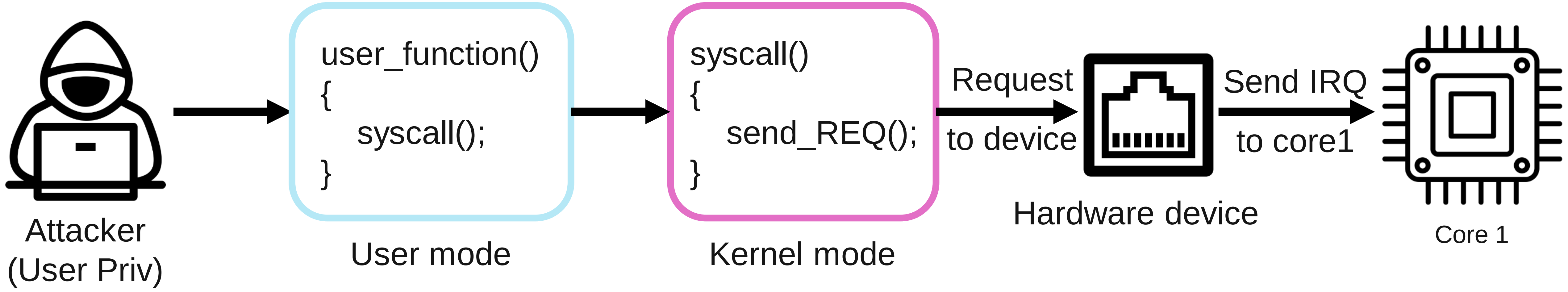
User mode

```
syscall()
{
    send_IPI();
}
```

Kernel mode

Send IPI to core1

Core 1



Attacker (User Priv)

```
user_function()
{
    syscall();
}
```

User mode

```
syscall()
{
    send_REQ();
}
```

Kernel mode

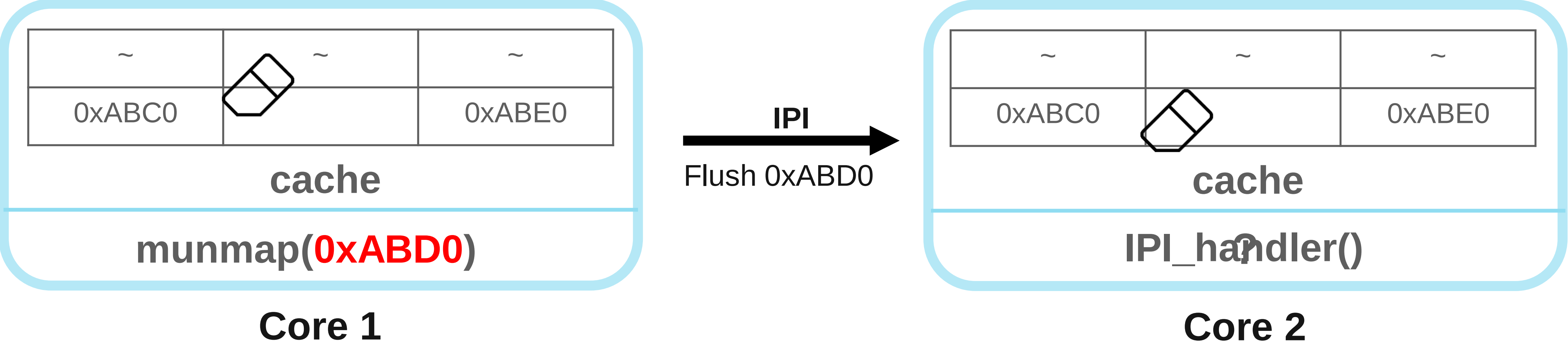
Request to device

Hardware device

Send IRQ to core1

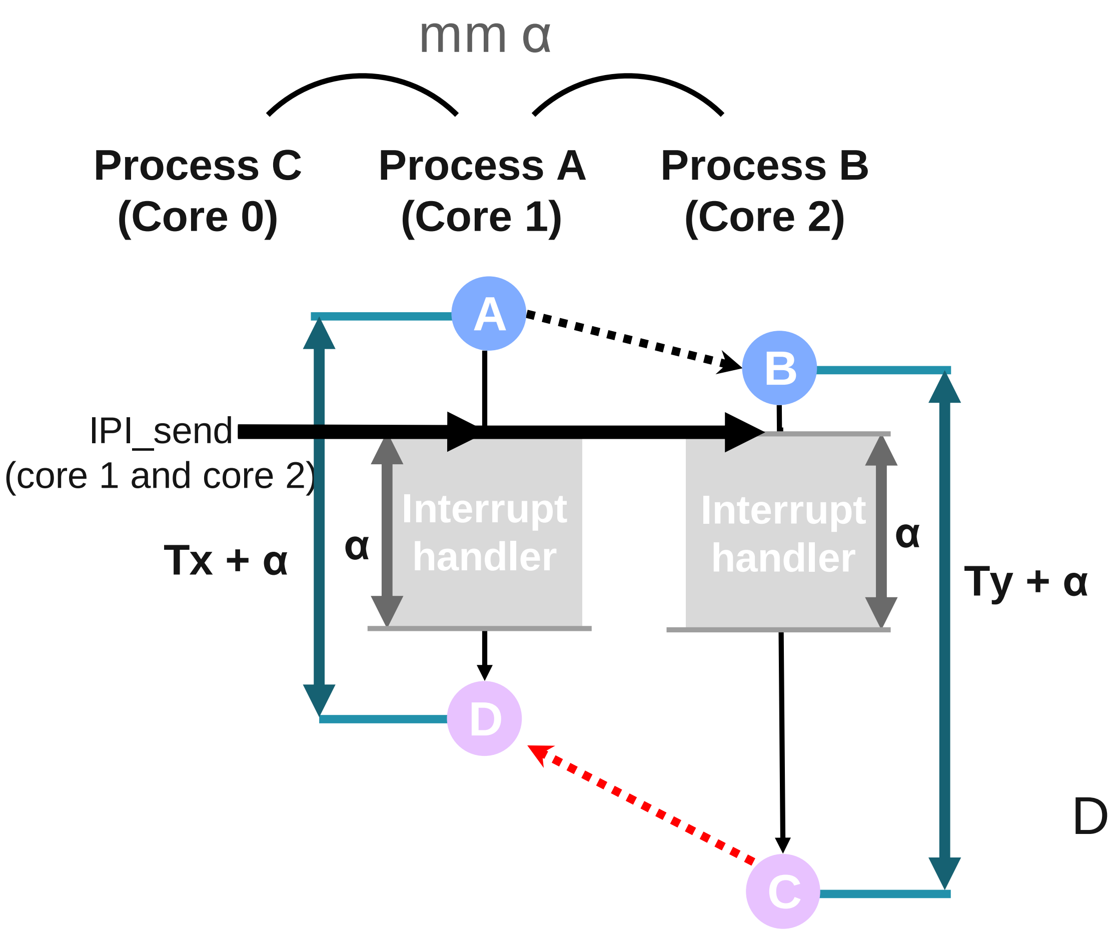
Core 1

ExpRace : TLB Shootdown



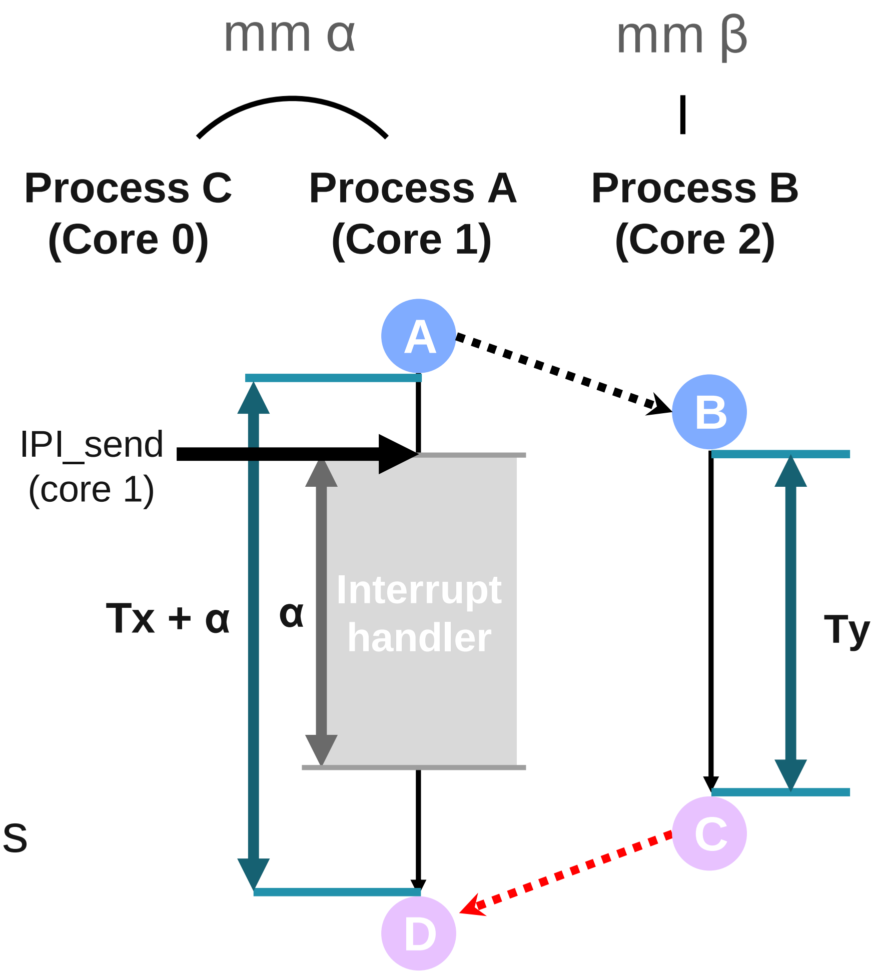
- Modern OS implement a TLB shootdown mechanism to ensure that TLB entries are synchronized across different cores.
- Syscalls that either modify the permission of the page (e.g., mprotect()) or unmap (e.g., munmap()) the page use IPI for TLB shootdown.

ExpRace : IPI Environment setting



If 3 processes have **same mm**

Same mm == thread
Different mm == process

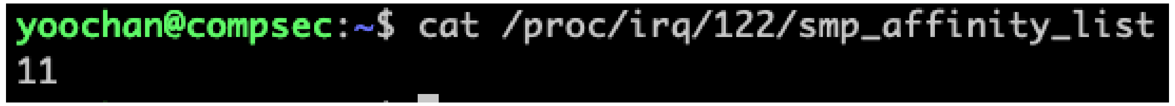


If process A and C have **same mm,**
and B have **different mm**

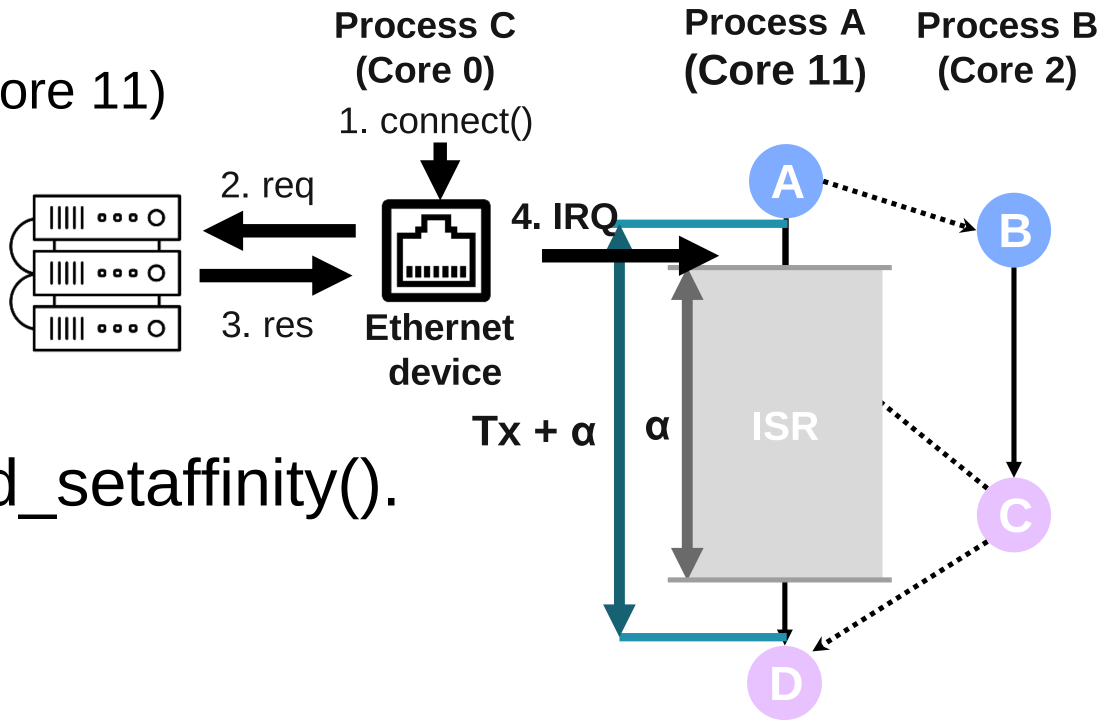
ExpRace : Hardware Interrupt Environment Setting

1. Check **irq's core affinity**.

   (In our environment, ethernet device (IRQ 122) have affinity to core 11)

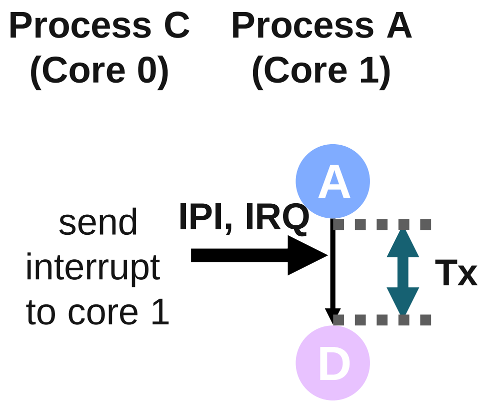


```
yoochan@compsec:~$ cat /proc/irq/122/smp_affinity_list
11
```



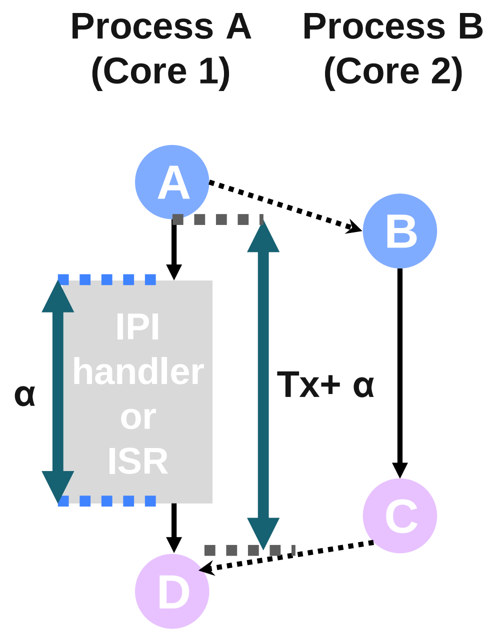
2. **Pin the thread to corresponding core** using sched_setaffinity().

ExpRace : Two conditions must be satisfied for succeed



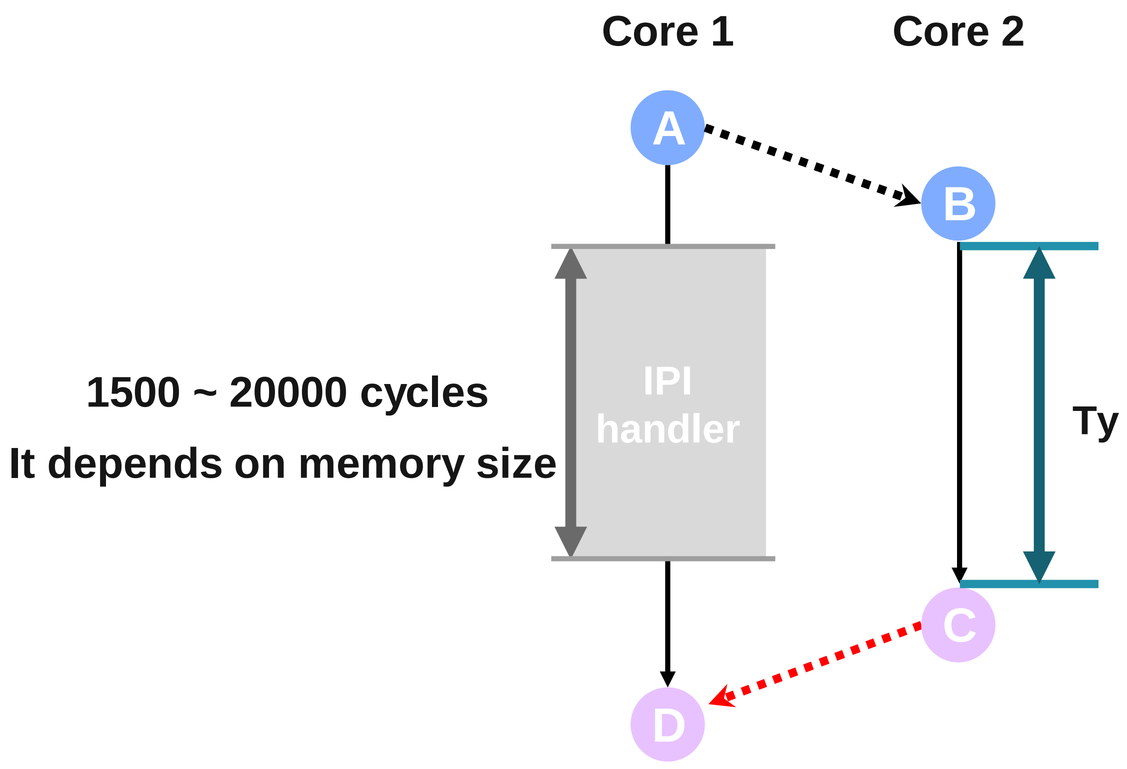
Interrupt is received **within Tx**

&&

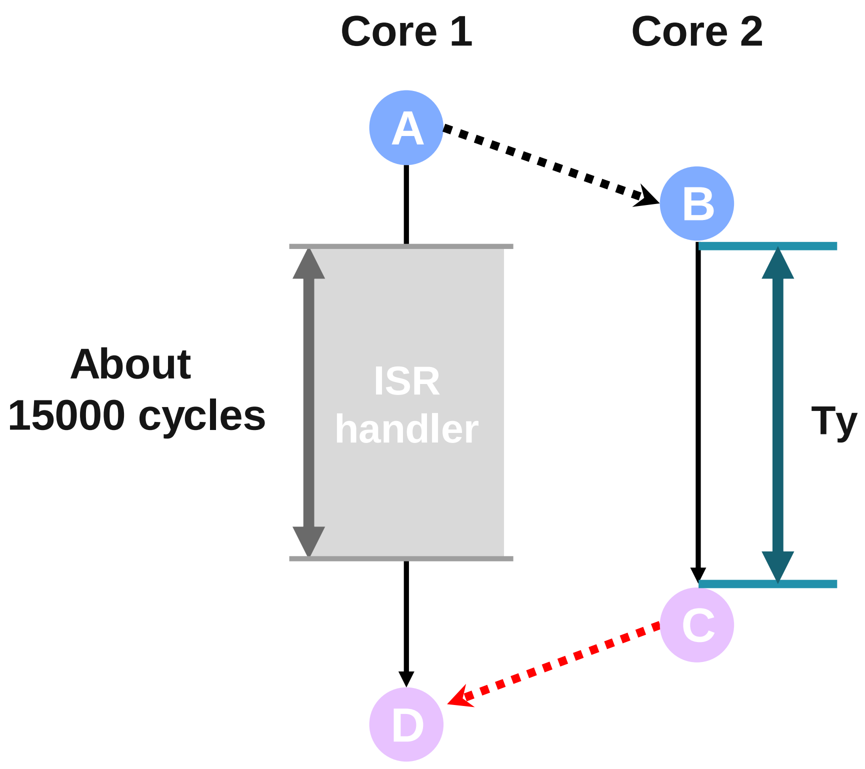


B and C is executed **within Tx + α**

ExpRace : How many cycles are extended?

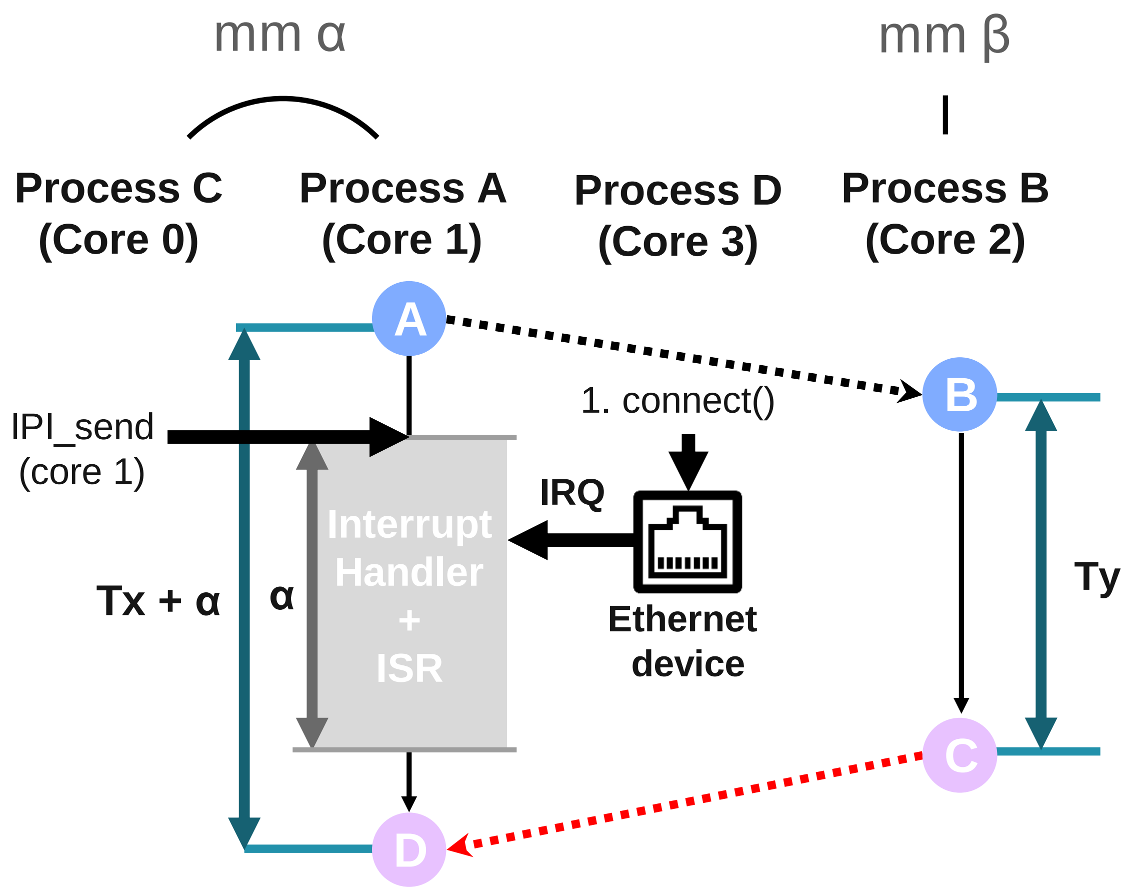


TLB Shootdown



Hardware Interrupt

ExpRace : Advanced Technique



- IPI and IRQ can be used simultaneously.
- The time window is extended up to 200,000 cycles

ExpRace : Other OSs

Windows 10

✓ TLB shootdown

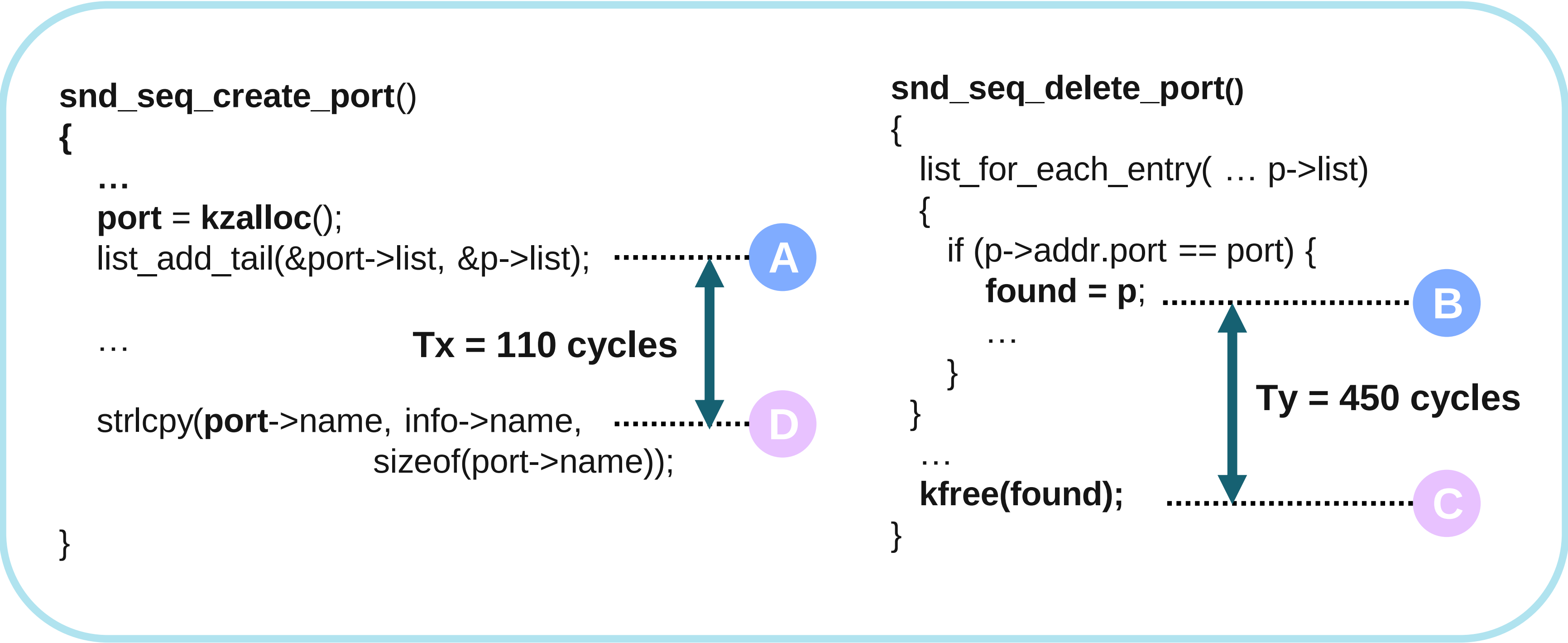
✓ Hardware Interrupt
(*#Device Parameters Interrupt* registry)

Mac OS

✓ TLB shootdown

✗ Hardware Interrupt

Case Study : CVE-2017-15265



```
snd_seq_create_port()
{
    …
    port = kzalloc();
    list_add_tail(&port->list, &p->list);

    …

    strlcpy(port->name, info->name,
                          sizeof(port->name));

}
```

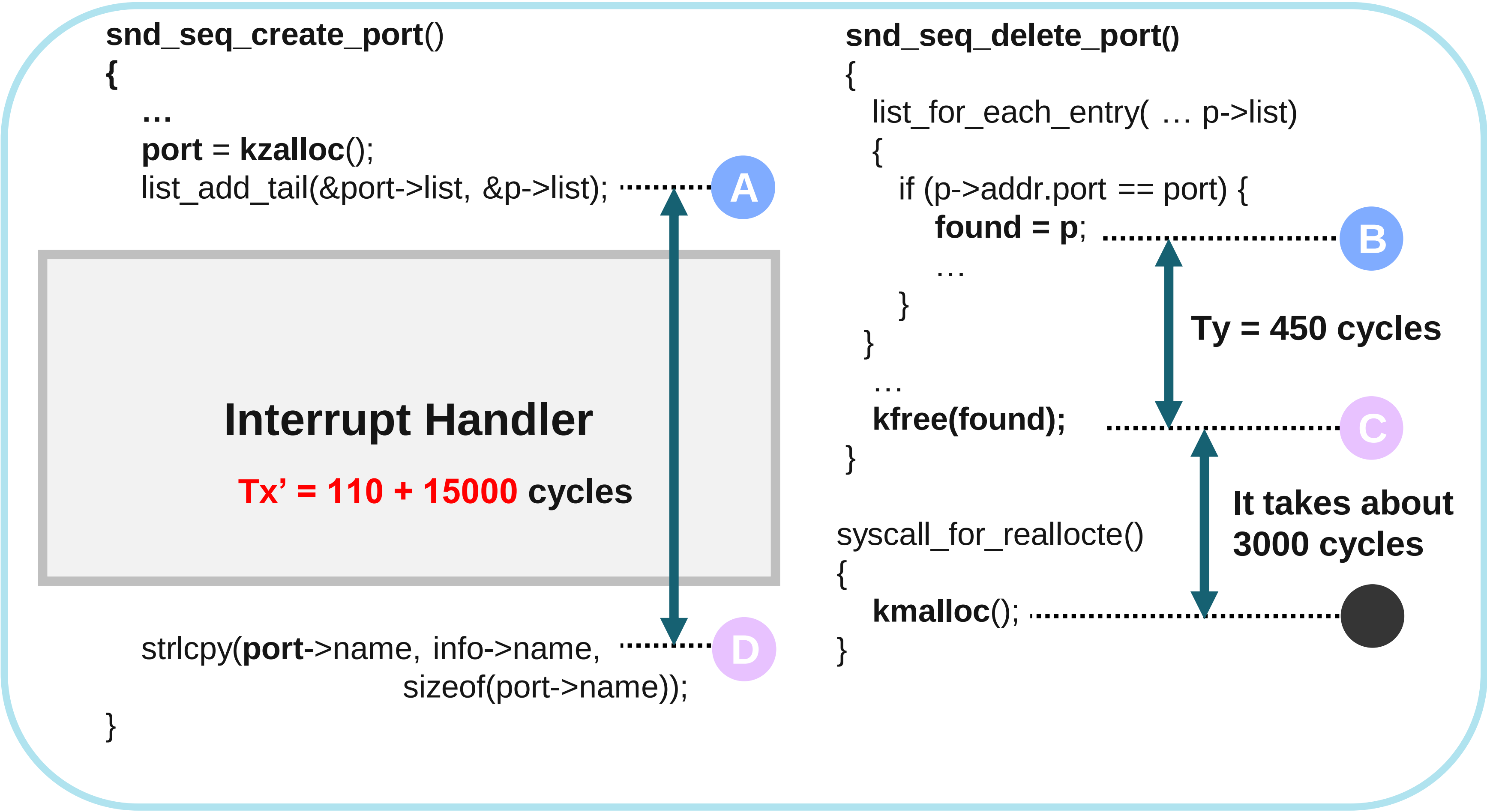
```
snd_seq_delete_port()
{
   list_for_each_entry( … p->list)
   {
      if (p->addr.port == port) {
          found = p;
          …
      }
   }
   …
   kfree(found);
}
```

Problems to exploit

1. **Non-inclusive Multi-variable Race**
2. **No time to reallocate**

if A >> B && C >> D , then **Use-After-Free Write occurs**.

ExpRace can solve two problems at once



```
snd_seq_create_port()
{
   …
   port = kzalloc();
   list_add_tail(&port->list, &p->list);

   strlcpy(port->name, info->name,
                      sizeof(port->name));
}
```

Interrupt Handler

Tx’ = 110 + 15000 cycles

```
snd_seq_delete_port()
{
  list_for_each_entry( … p->list)
  {
    if (p->addr.port == port) {
      found = p;
      …
    }
  }
  …
  kfree(found);
}

syscall_for_reallocte()
{
  kmalloc();
}
```

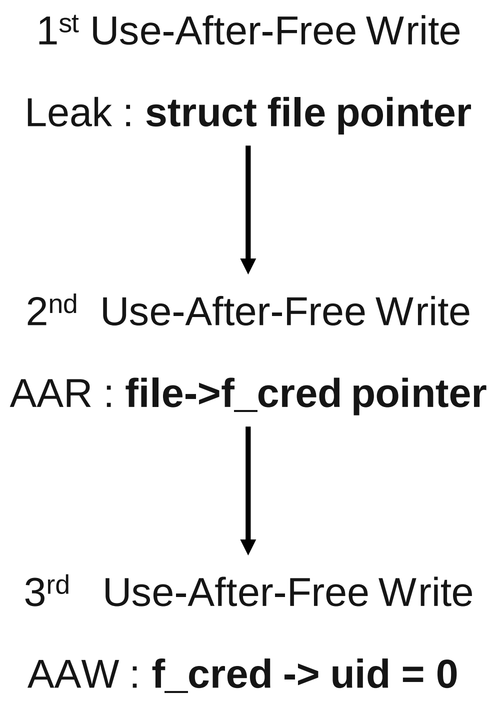
Ty = 450 cycles

It takes about 3000 cycles

if A >> B && C >> D , then **Use-After-Free Write occurs**.

Brief introduction about memory corruption exploit

- Spray struct file pointer using SCM_RIGHT
- Partially overwrite the pointer in reallocated structure for kernel address leak.
- Use iovec structure for AAR, AAW.



1st Use-After-Free Write

Leak : **struct file pointer**

2nd Use-After-Free Write

AAR : **file->f_cred pointer**

3rd Use-After-Free Write

AAW : **f_cred -> uid = 0**

We totally trigger the vulnerability **3 times**

DEMO

```
yoochan@snu-hostname:~$ 
```

Conclusion

- Some type of race condition vulnerabilities are impossible to exploit.
- ExpRace can make unexploitable race to exploitable race.
- ExpRace is the only method that can be used in general.