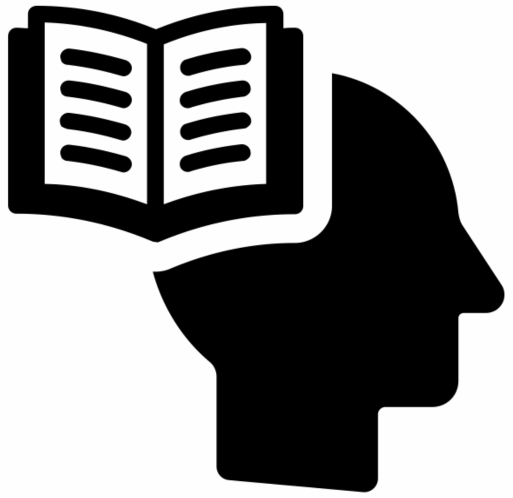
black hat USA 2020

AUGUST 5-6, 2020
BRIEFINGS

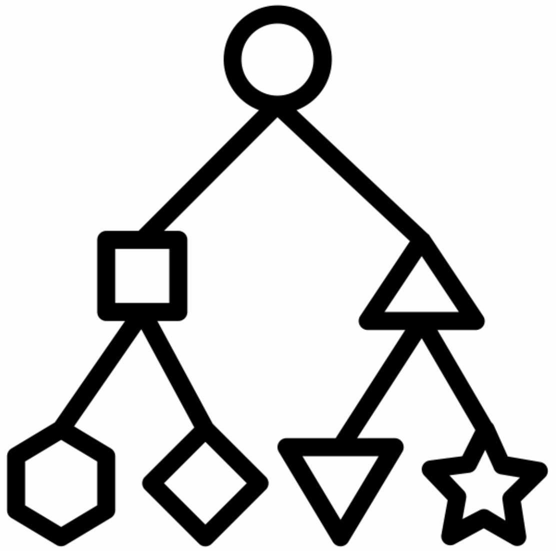
Exploiting Kernel Races Through Taming Thread Interleaving

Yoochan Lee, Byoungyoung Lee, Chanwoo Min
Seoul National University, Virginia Tech

Summary



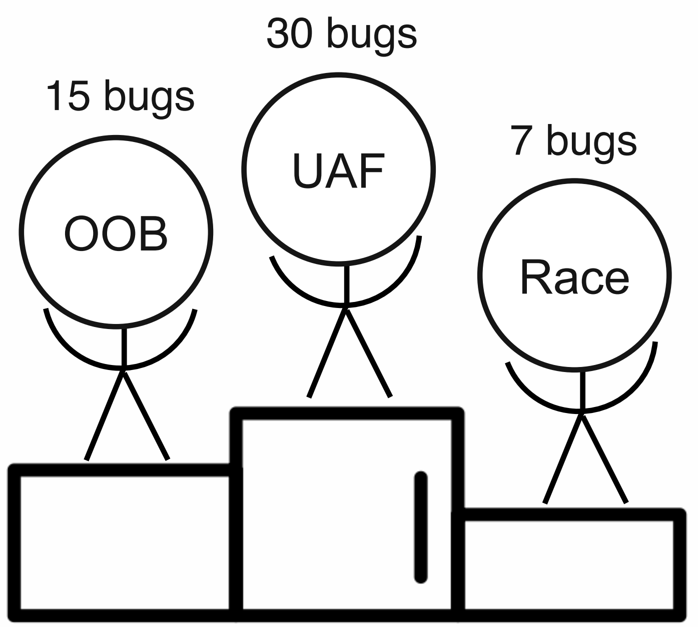
- Background on races



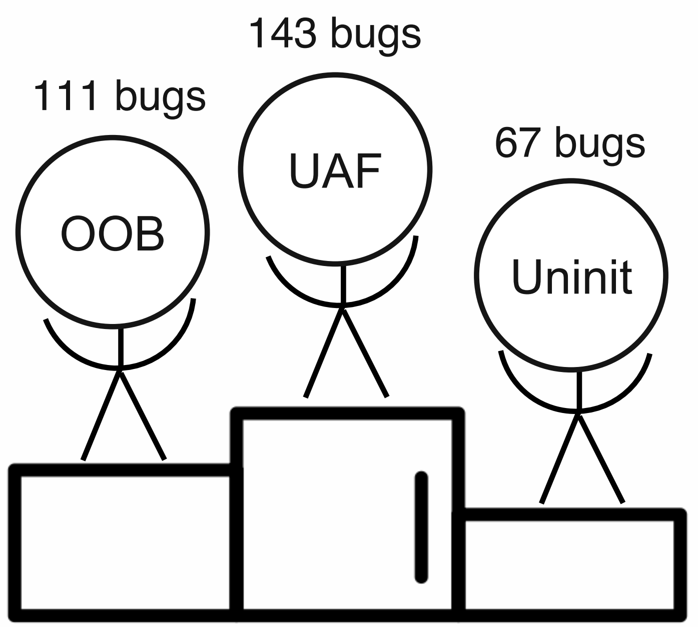
- Classification on races
- **Unexploitable races**

- New technique turning **unexploitable races to exploitable races**

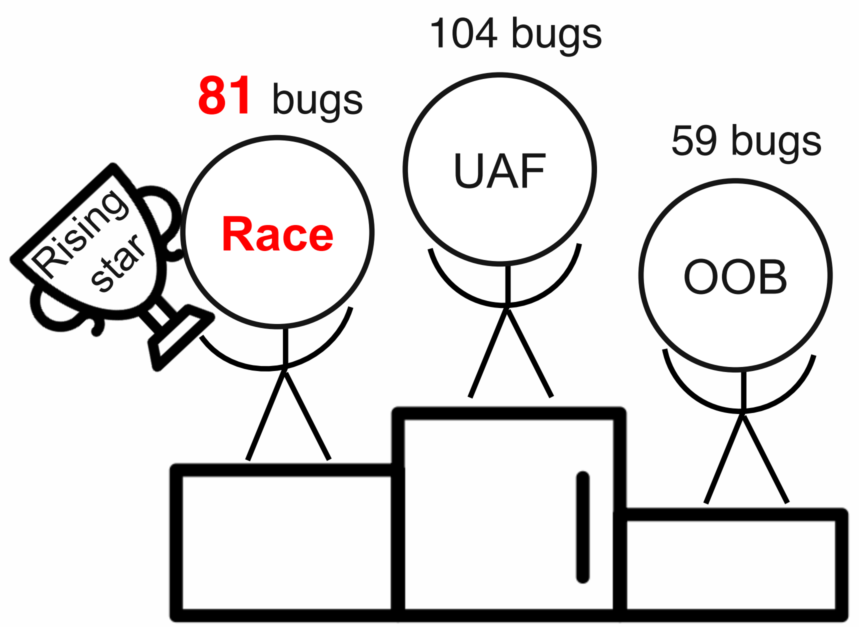
Race condition is an increasing attack vector



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



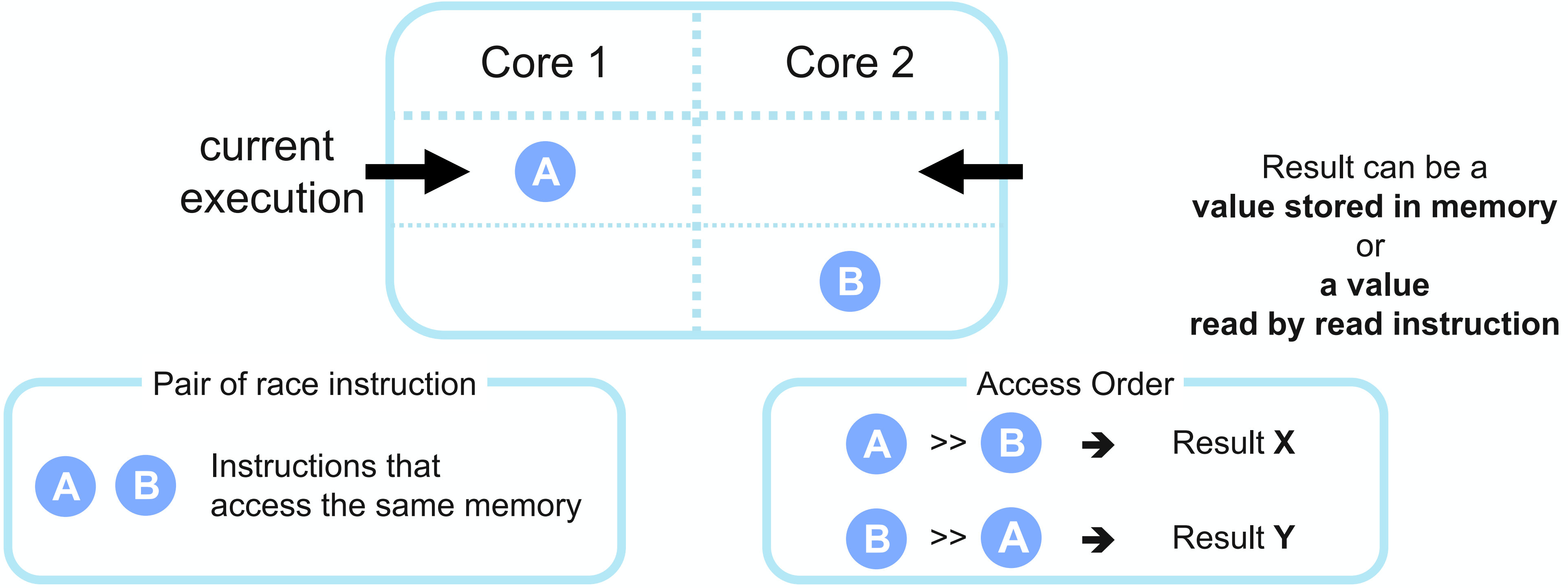
# of fixed bugs that Syzkaller found in **2018**



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

- 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 processors

➔ **Execution results are different** depending on the access order.

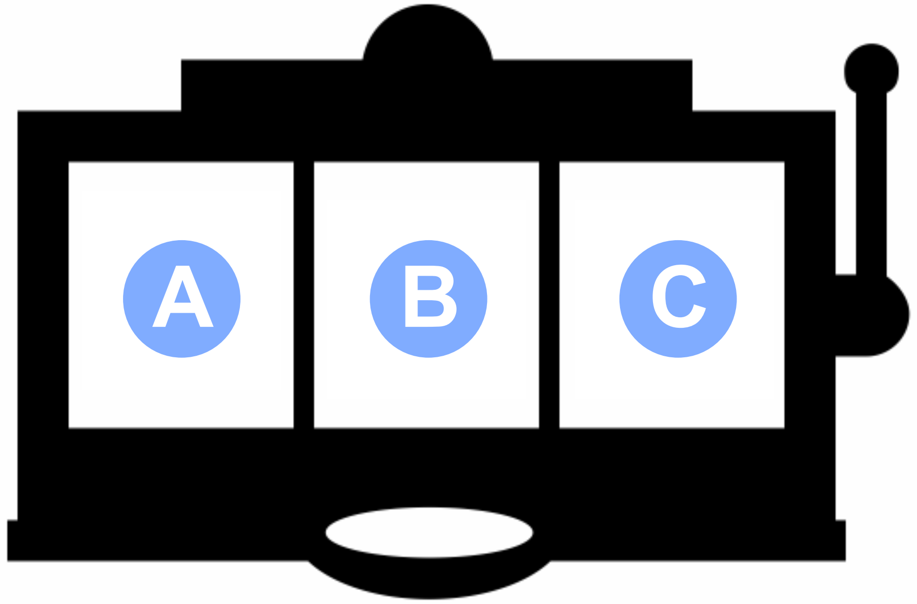
Background : Race Condition Vulnerability

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 :

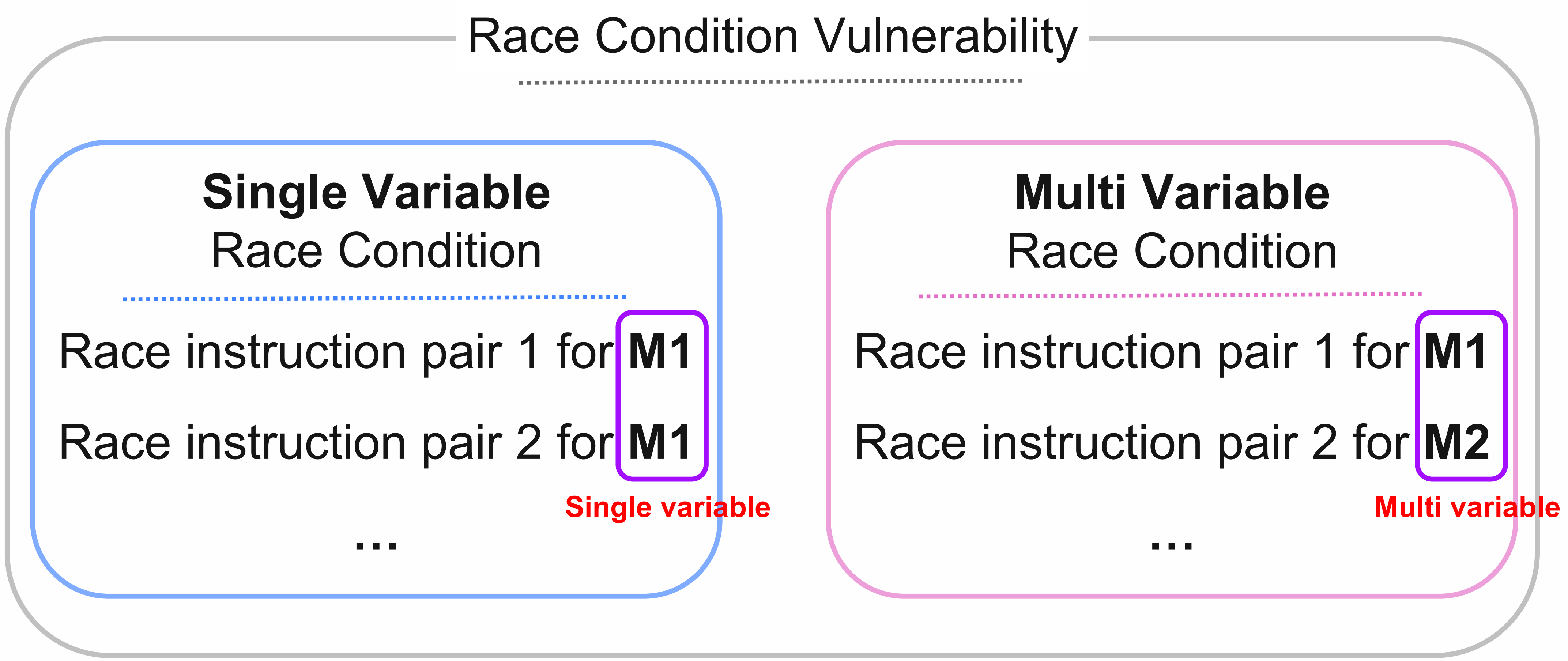
Try until success

Background : Exploitability of Race Condition Vulnerability

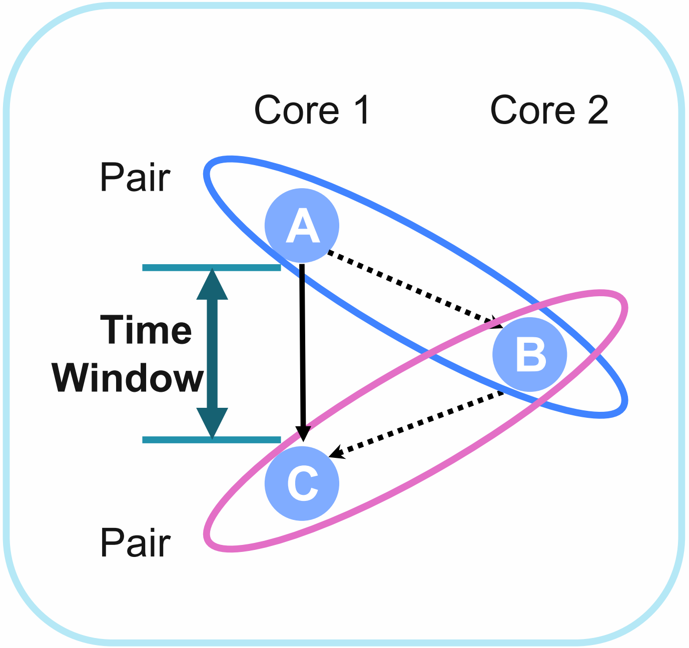
Exploitable Races? = **A very specific memory access order** + Availability of Memory Corruption

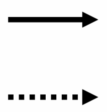
(e.g., if A >> B >> C , then)

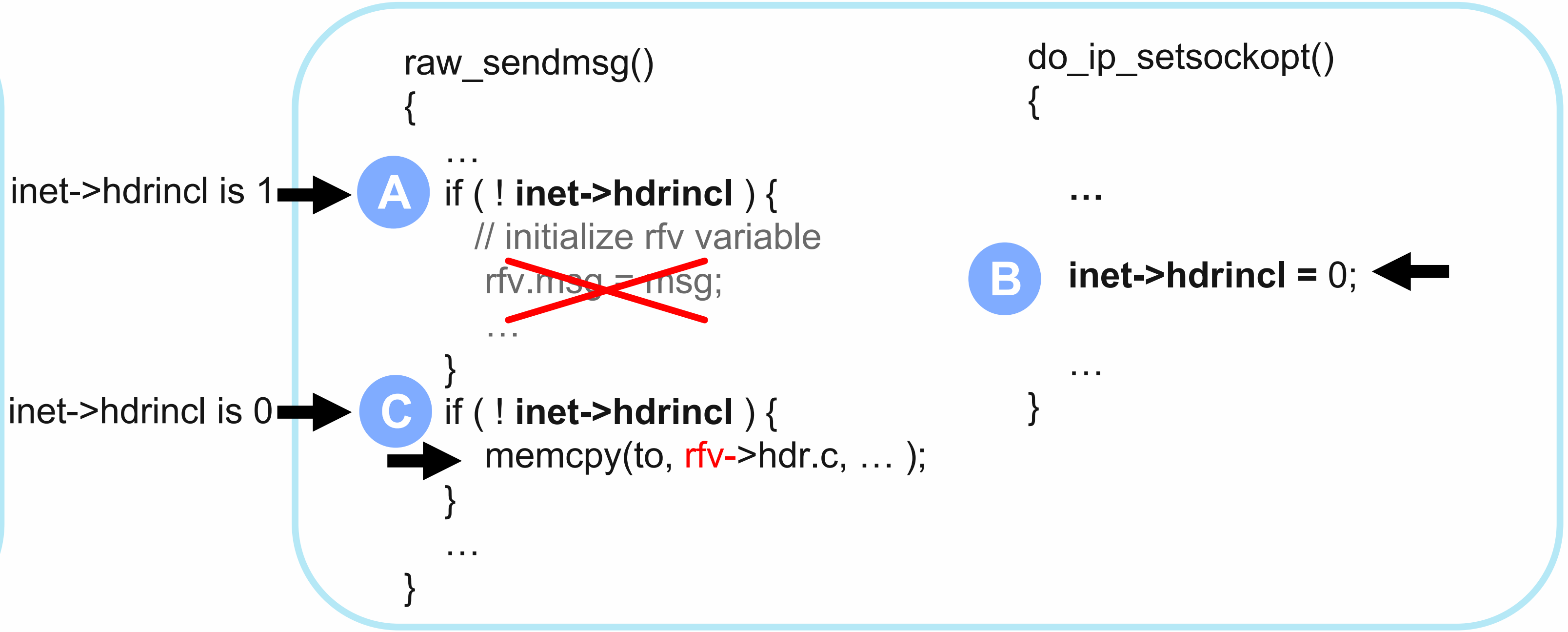
Classification of Race Condition Vulnerability


Race Condition Vulnerability
Single Variable
Race Condition
Race instruction pair 1 for **M1**
Race instruction pair 2 for **M1**
Single variable
…
Multi Variable
Race Condition
Race instruction pair 1 for **M1**
Race instruction pair 2 for **M2**
Multi variable
…

Single-variable Race Condition







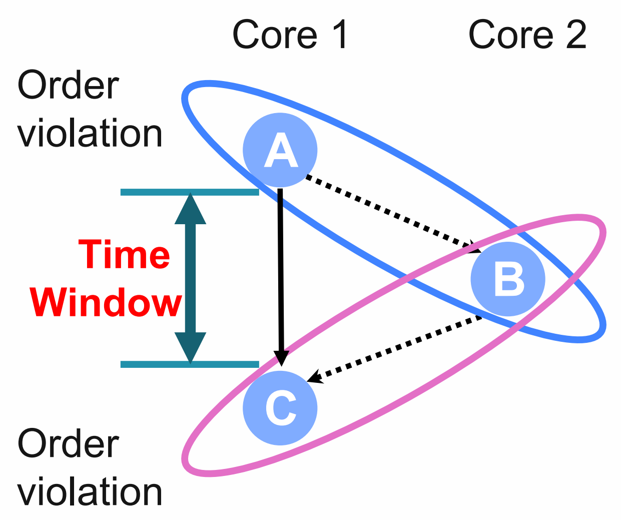
```
raw_sendmsg()
{
    …
A   if ( ! inet->hdrincl ) {
      // initialize rfv variable
      ~~rfv.msg = msg;~~
      …
    }
C   if ( ! inet->hdrincl ) {
      memcpy(to, rfv->hdr.c, … );
    }
    …
}
```

```
do_ip_setsockopt()
{
    …
B   inet->hdrincl = 0;
    …
}
```

Case study : CVE-2017-17712

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

Exploitability of Single-variable Race

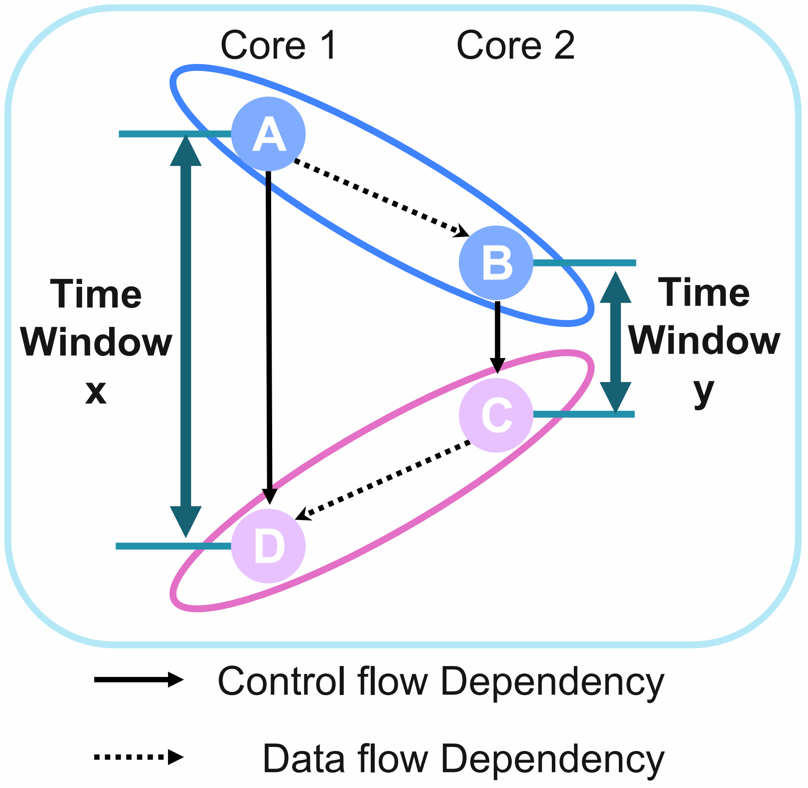


- Brute-forcing would somehow trigger the race

  ➔ if B can be executed within the time window

- The smaller the time window is, the lower the probability of successful races.

Multi-variable Race Condition



Pair of race instruction

● A ● B Instructions that access the **M1**

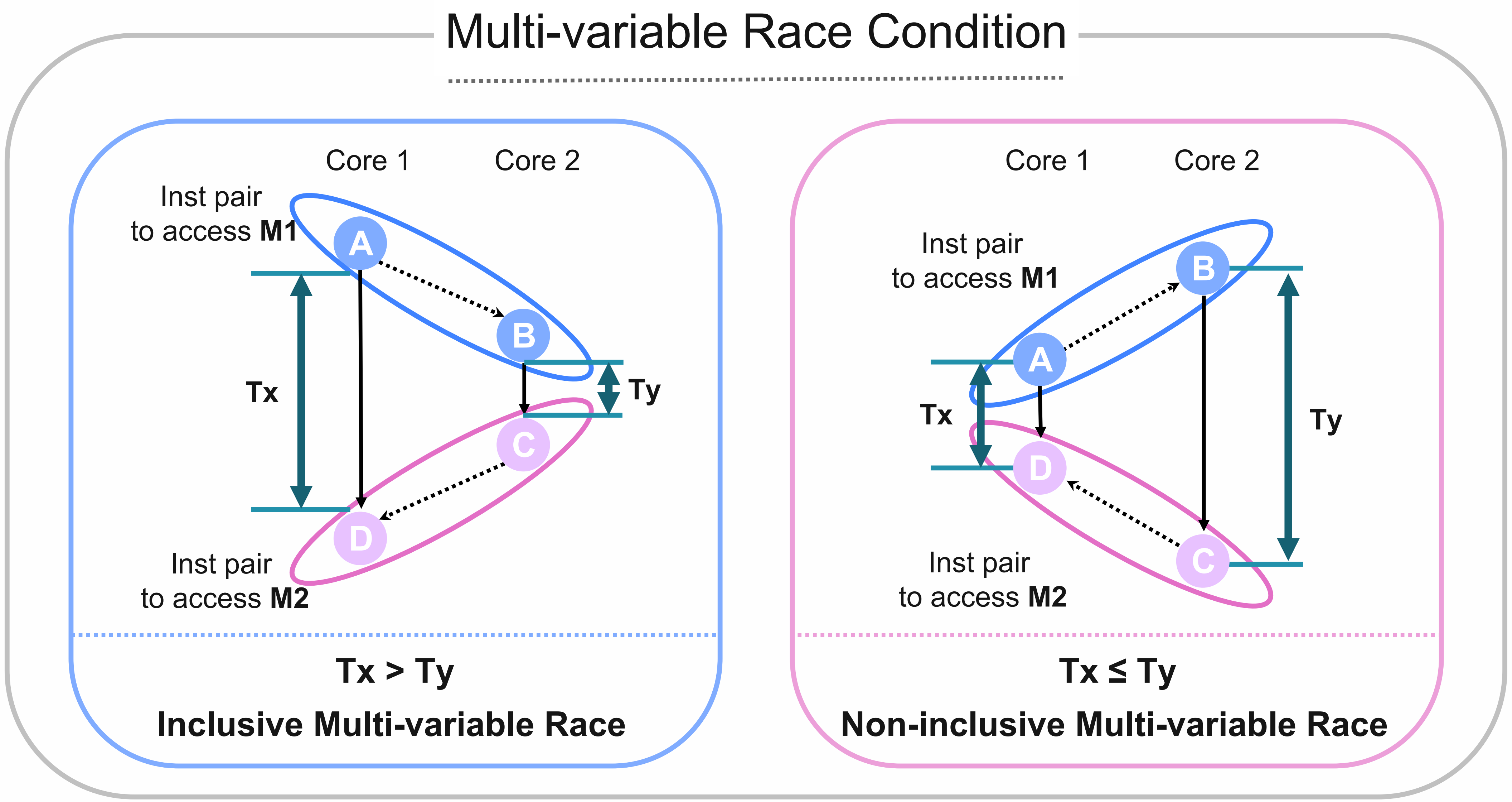
Pair of race instruction

● C ● D Instructions that access the **M2**

if A >> B && C >> D ,

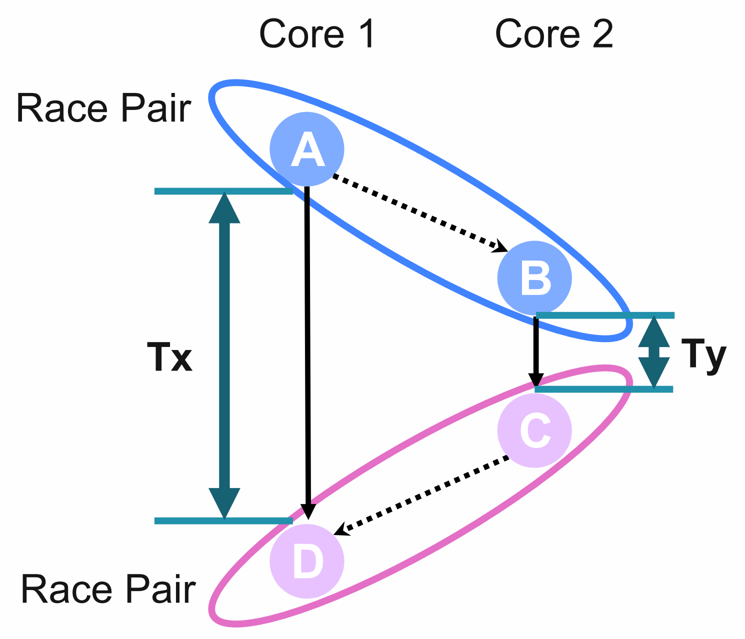
then memory corruption occurs.

Multi-variable Race Condition



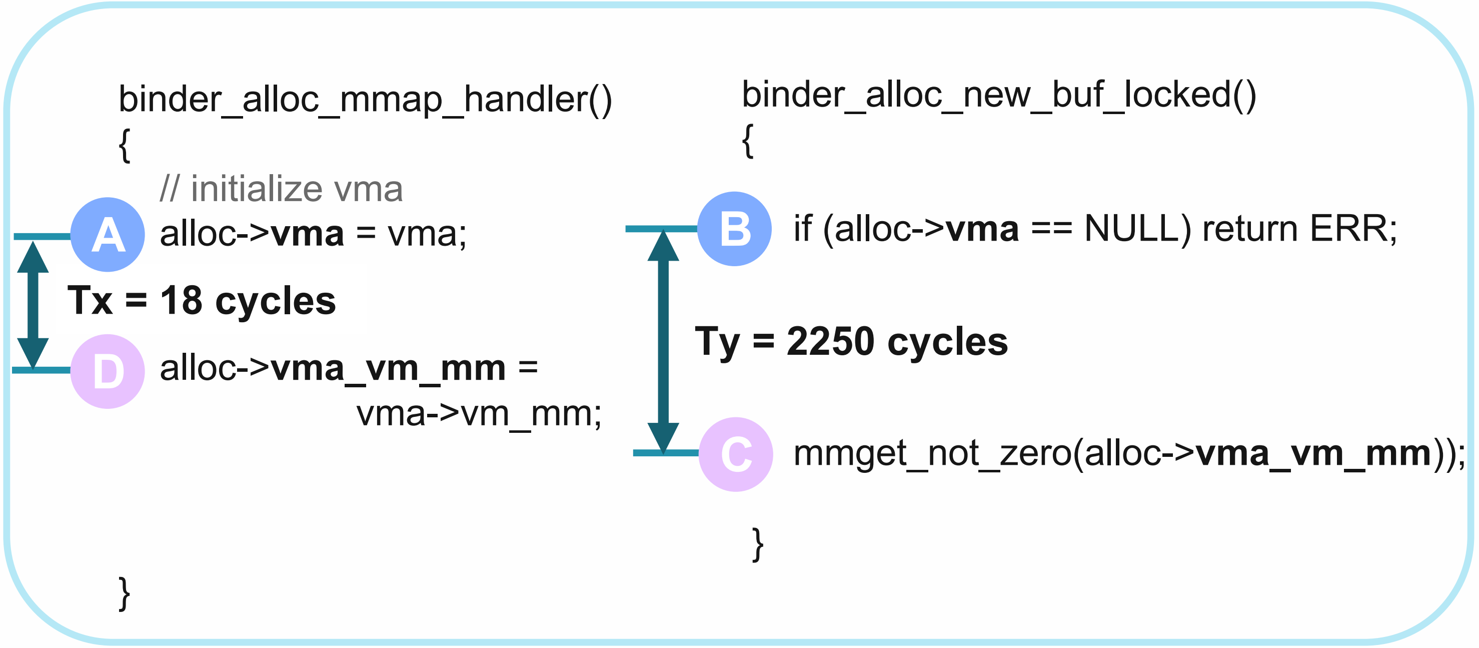
#BHUSA @BLACKHATEVENTS

Exploitability of Inclusive Multi-variable Race



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

Problem : Exploitability of Non-inclusive Race

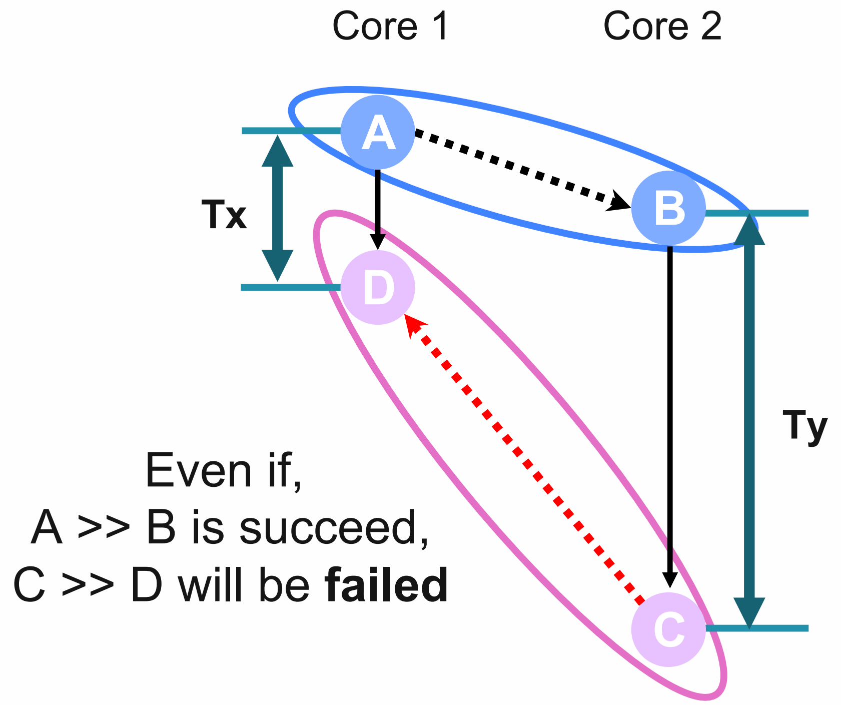


```
binder_alloc_mmap_handler()
{
    // initialize vma
(A) alloc->vma = vma;
    Tx = 18 cycles
(D) alloc->vma_vm_mm =
                  vma->vm_mm;

}
```

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

     Ty = 2250 cycles

(C)  mmget_not_zero(alloc->vma_vm_mm));

}
```

Case study : Patch #987393

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

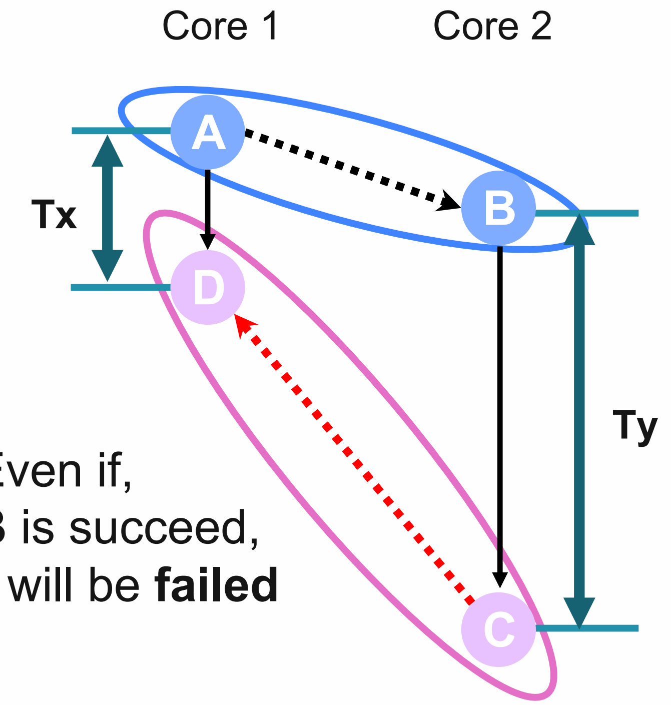


- Brute-force never works.
- **impossible to execute** with the order of A >> B && C >> D .

Problem : Exploitability of Non-inclusive Race

Non-inclusive race vulnerabilities found in linux kernel

	Tx	Ty
CVE-2017-15265	*35*	*450*
CVE-2019-1999	*150*	*1,800*
CVE-2019-2025	*50*	*600*
CVE-2019-6974	*18*	*1,210*
#1035566	*1,153*	*13,121*
#987393	*18*	*2,250*
#759959	*120*	*730*
.		
.		
.		

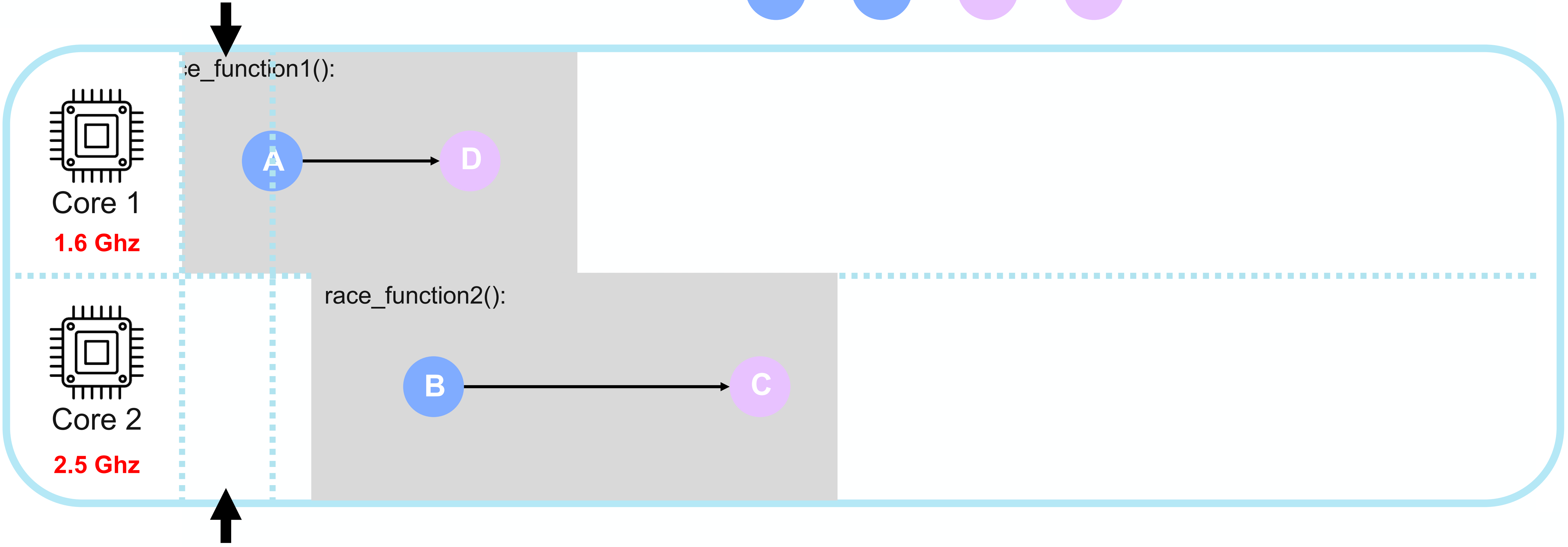


Even if,
A >> B is succeed,
C >> D will be **failed**

- Brute-force never works.
- **impossible to execute** with the order of A >> B && C >> D .

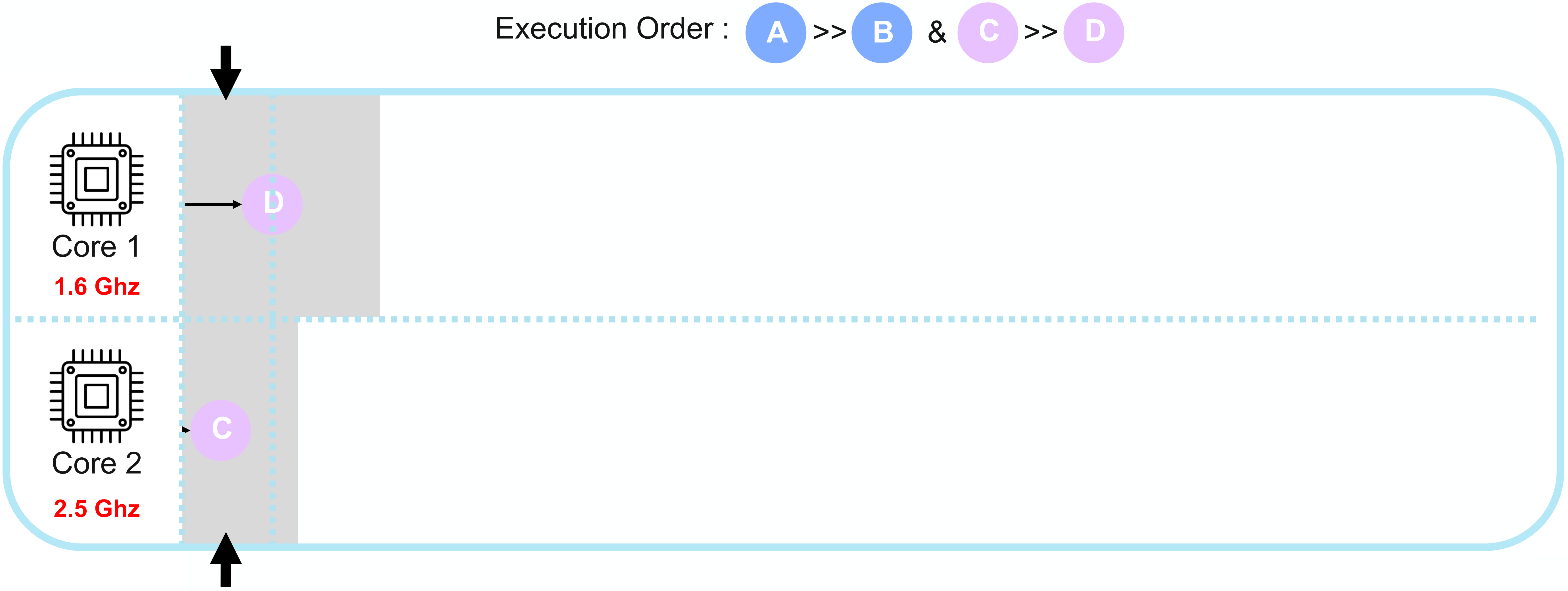
Previous method : Using Different Core Latency

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



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

Previous method : Using Different Core Latency



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

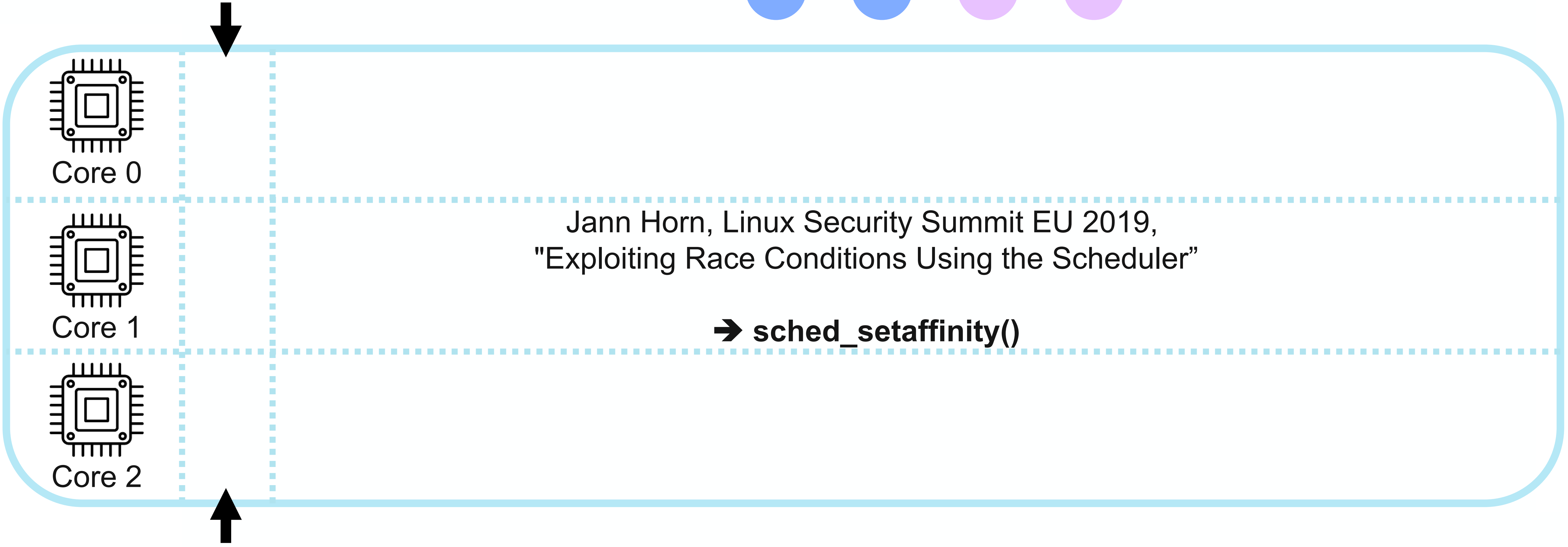
Limitations of Use Different Core Latency

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



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

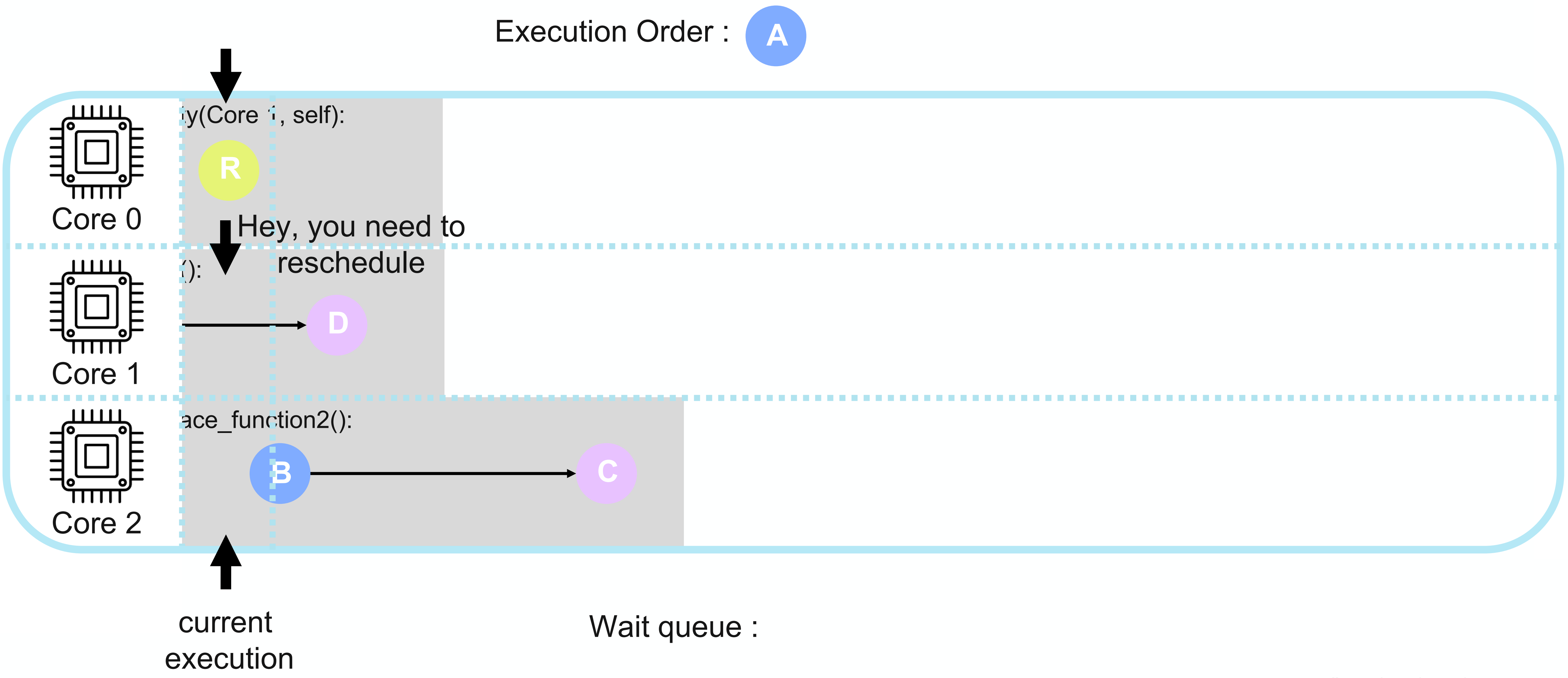
➔ **sched_setaffinity()**

current execution

Wait queue :

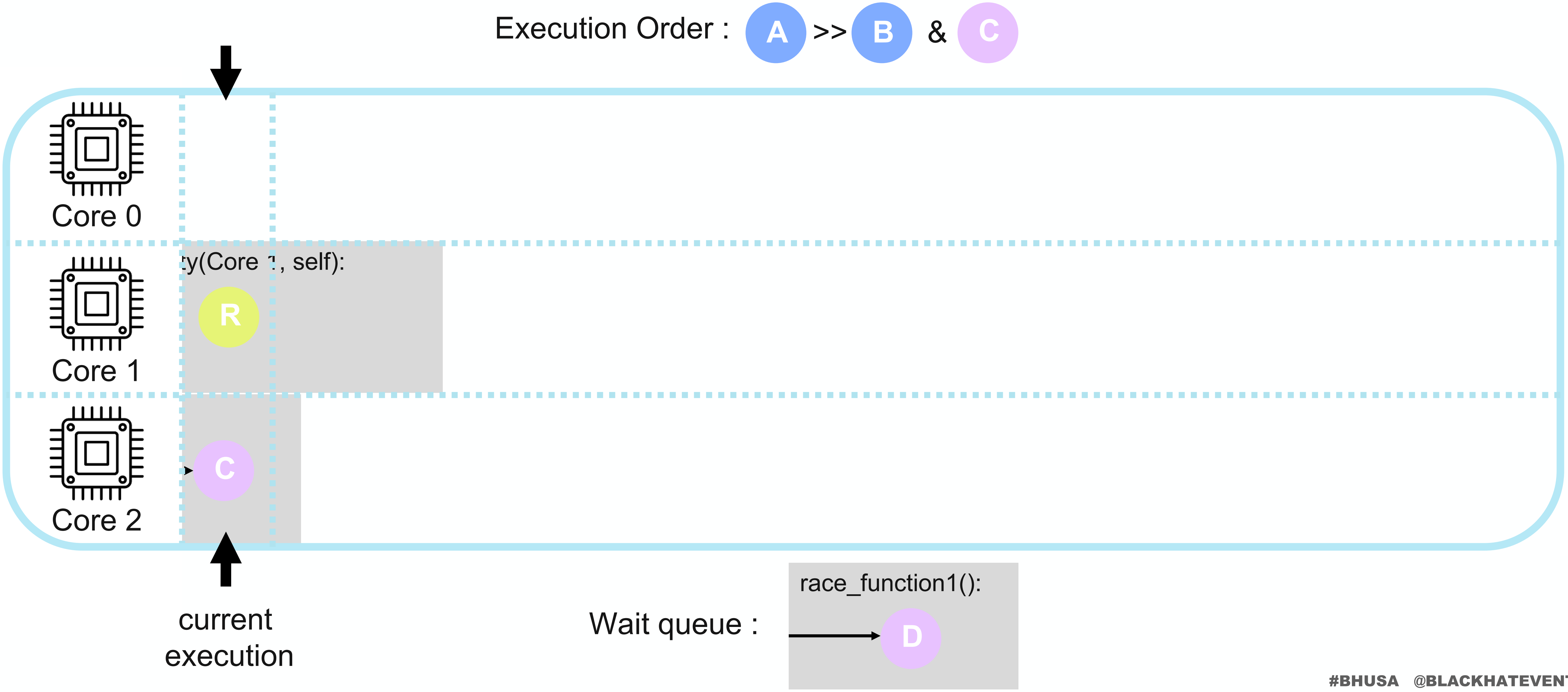
Previous Approach : Using scheduler (CONFIG_PREEMPT)

Execution Order :



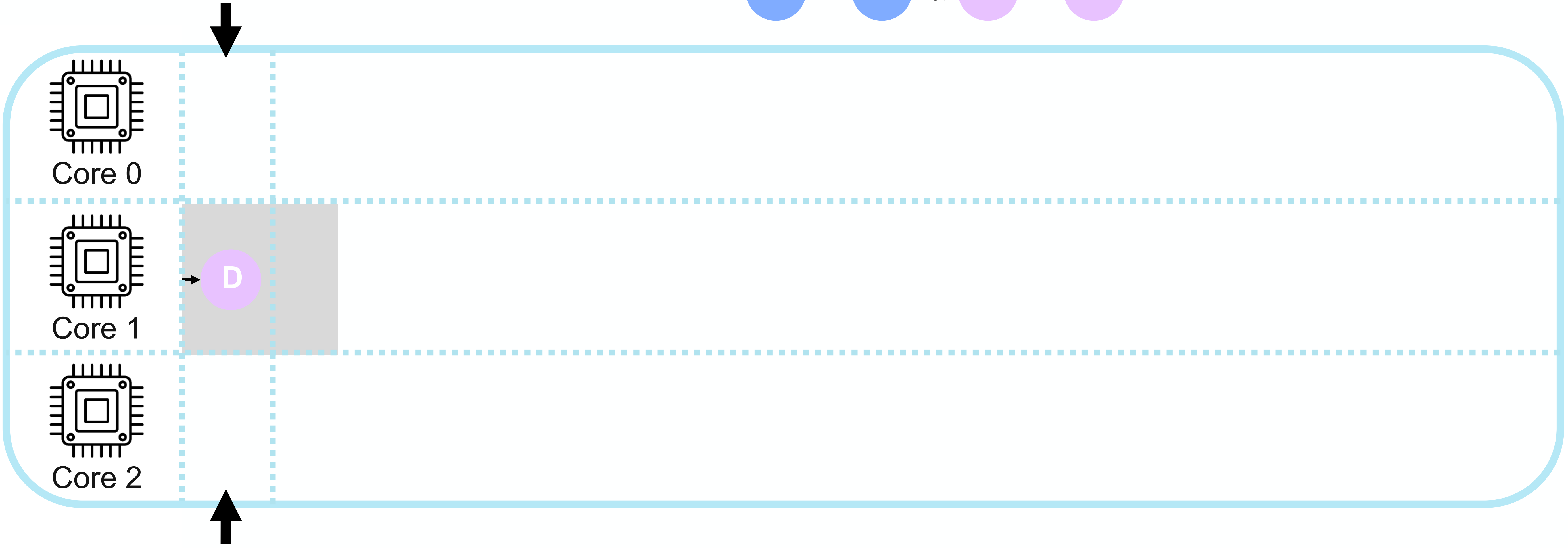
Wait queue :

Previous Approach : Using scheduler (CONFIG_PREEMPT)



Previous Approach : Using scheduler (CONFIG_PREEMPT)

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



current execution

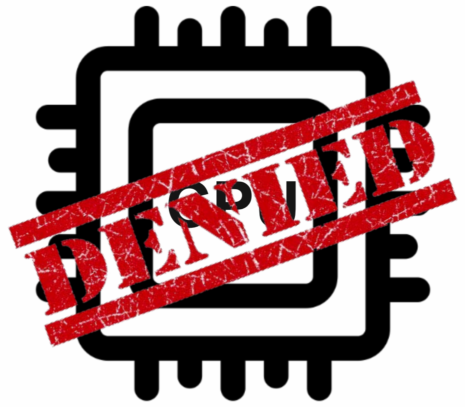
Wait queue :

Limitation of Using scheduler

Configuration dependency

- Can be used when COFIG_PREEMPT option is applied.
- Linux uses **CONFIG_PREEMPT_VOLUTARY** option **by default.**

Each of methods has obvious limitations

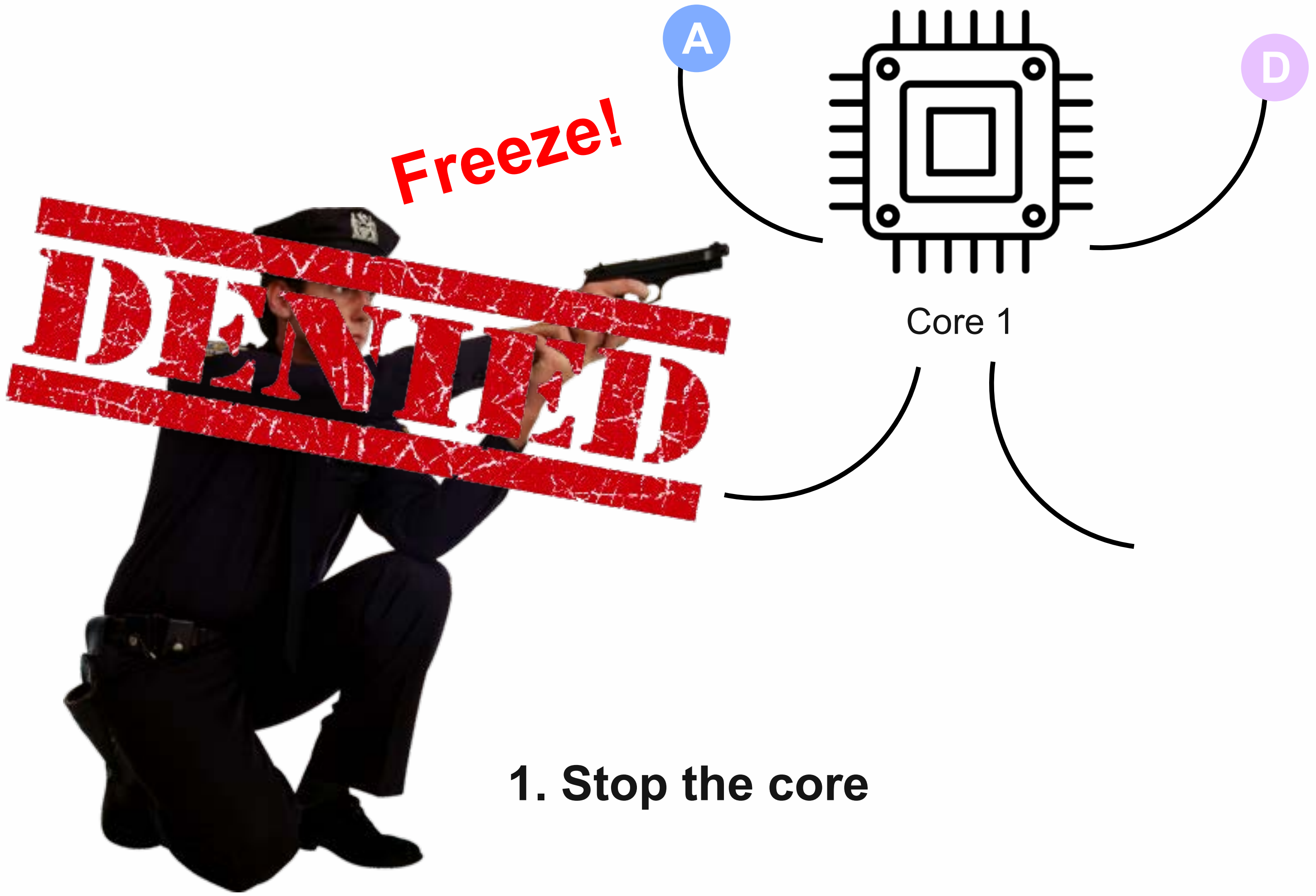


CPU dependency

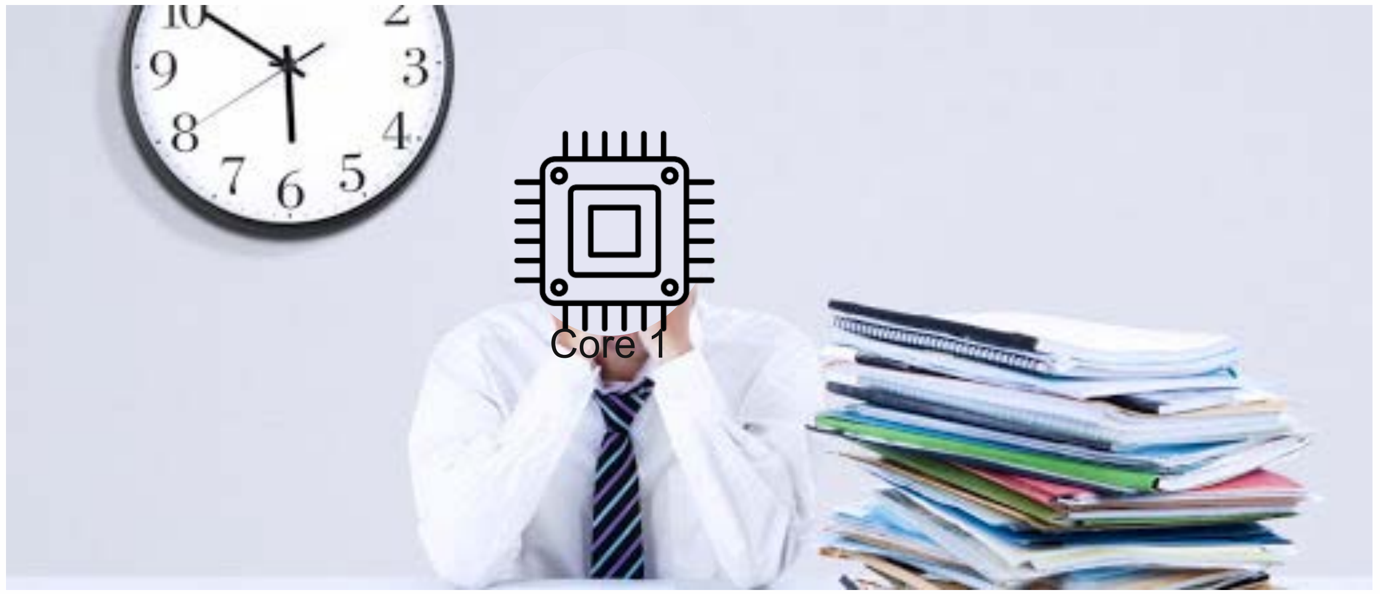
Configuration dependency

- All previous methods are hard to be used in general.
- We need **a new method** that extends the time window.

How to extend the time window?

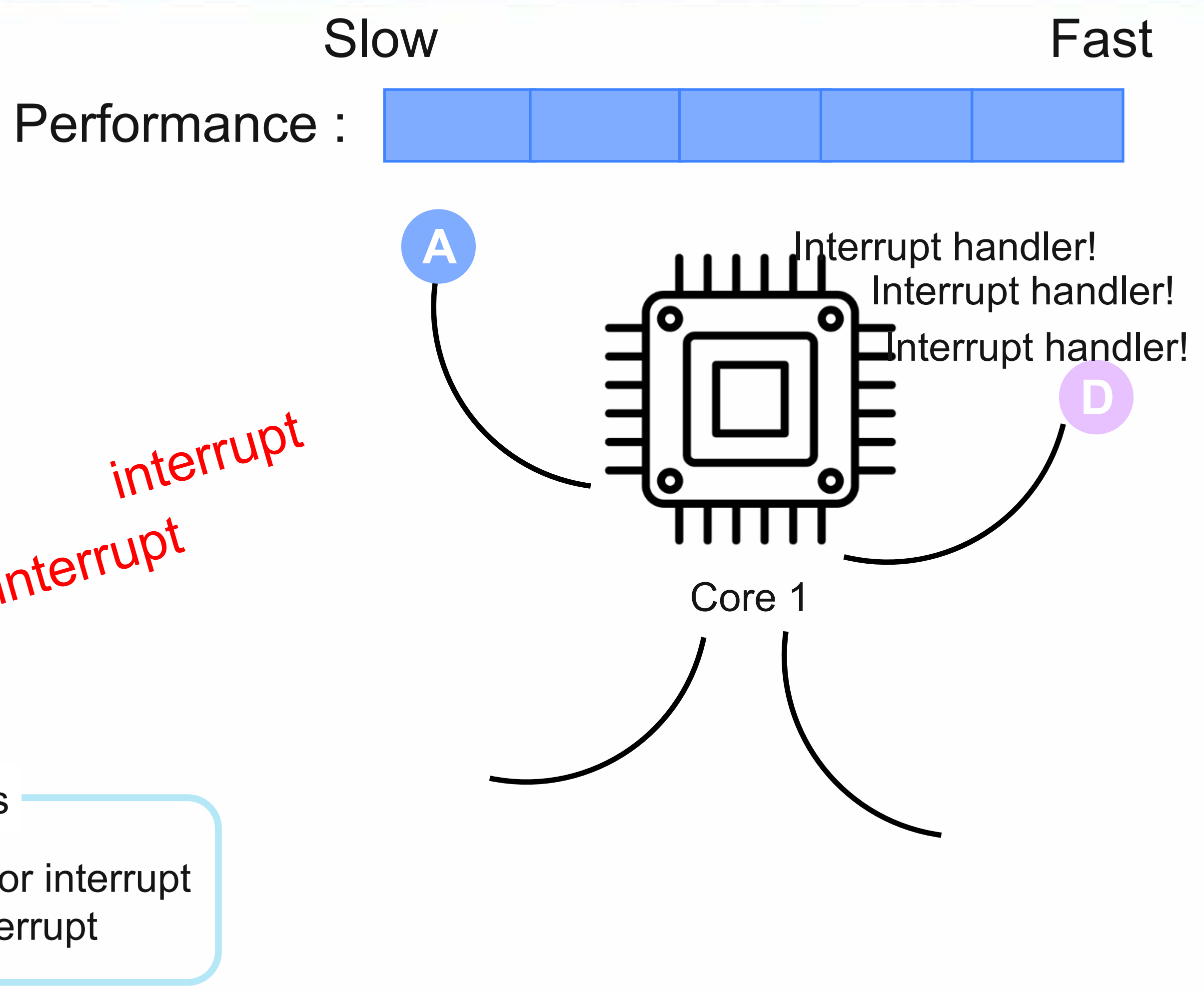


1. Stop the core



2. Degrade the performance

ExpRace

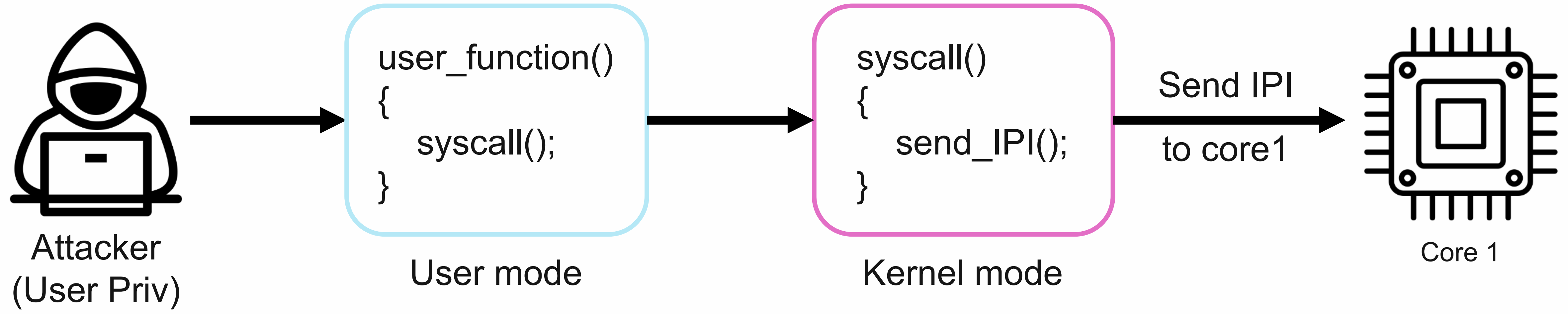


Bullets

- Inter-processor interrupt
- Hardware Interrupt

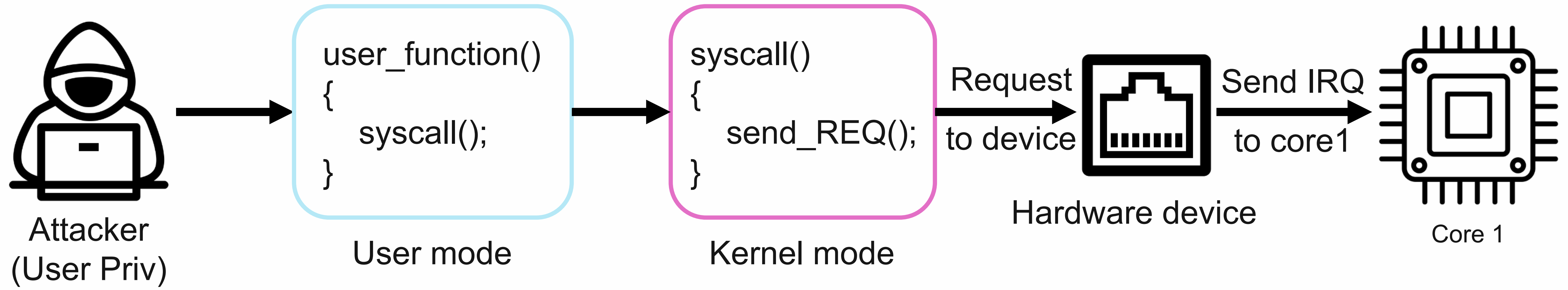
- The key idea of ExpRace is **to keep raising interrupts** to indirectly alter kernel thread's interleaving.

ExpRace : How to send IPI & IRQ with user priv



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

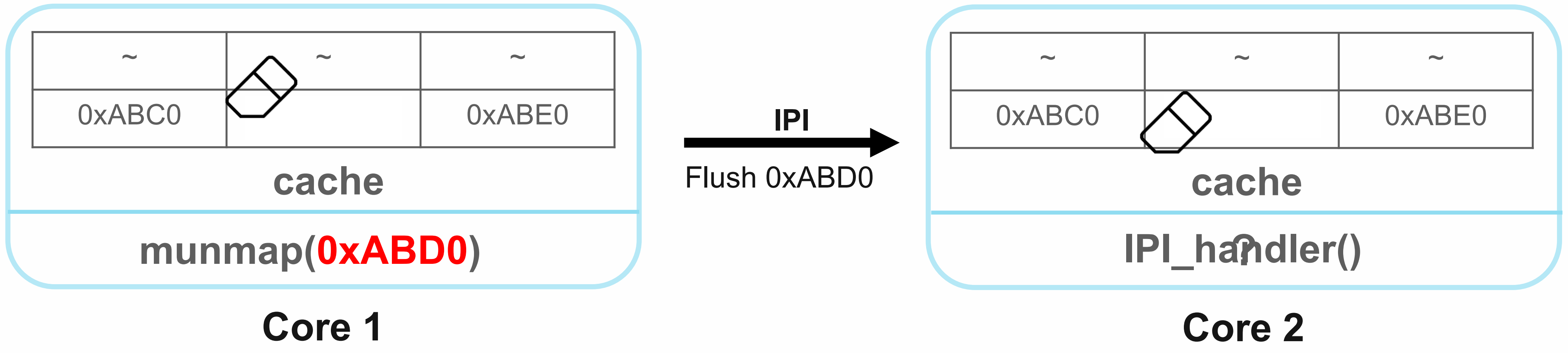
```
syscall()
{
    send_IPI();
}
```



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

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

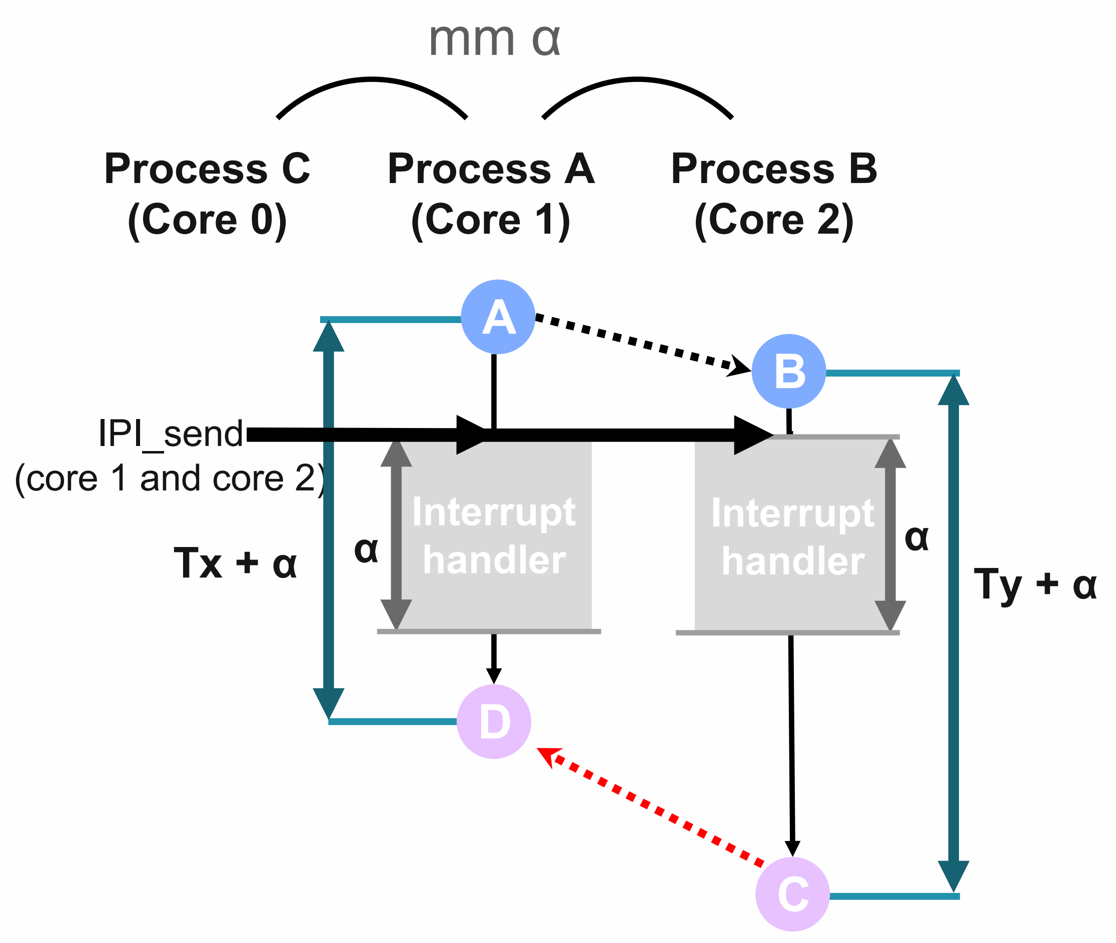
ExpRace : TLB Shootdown



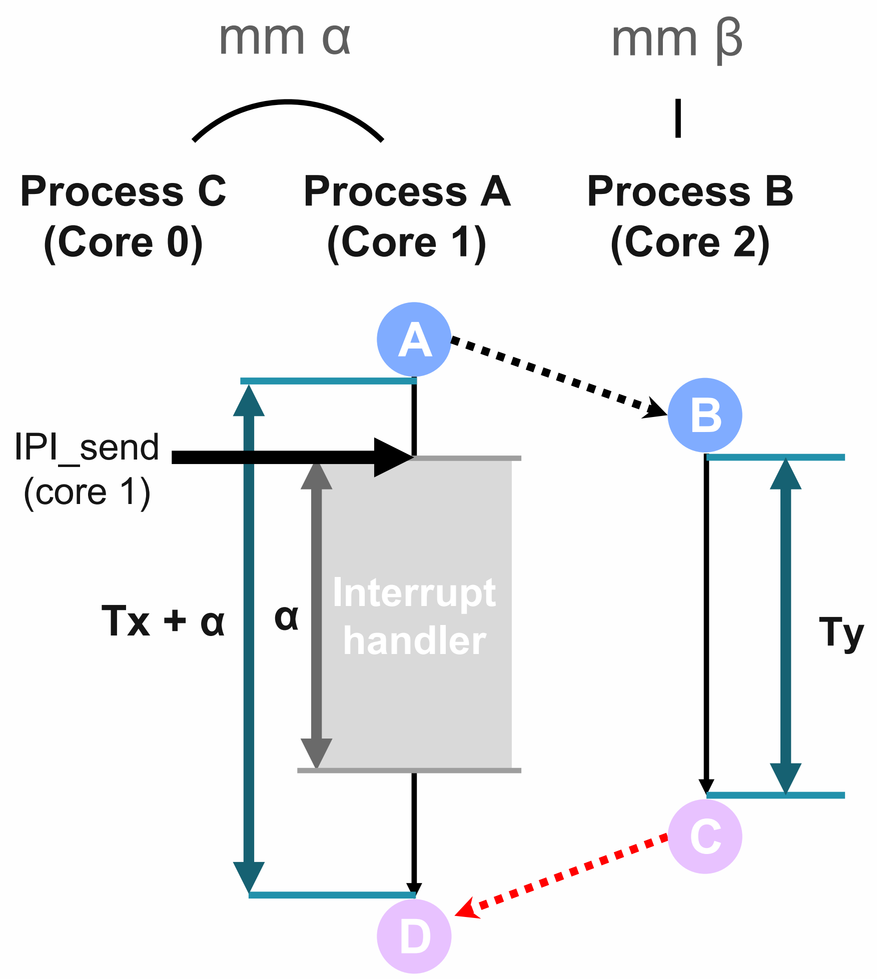
- Modern OSs implement a TLB shootdown mechanism to ensure that TLB entries are synchronized across different cores.

- 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**

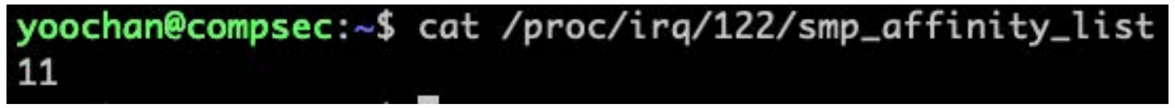


If process A and C have **same mm,**
and B have **different mm**

ExpRace : Hardware Interrupt Environment Setting

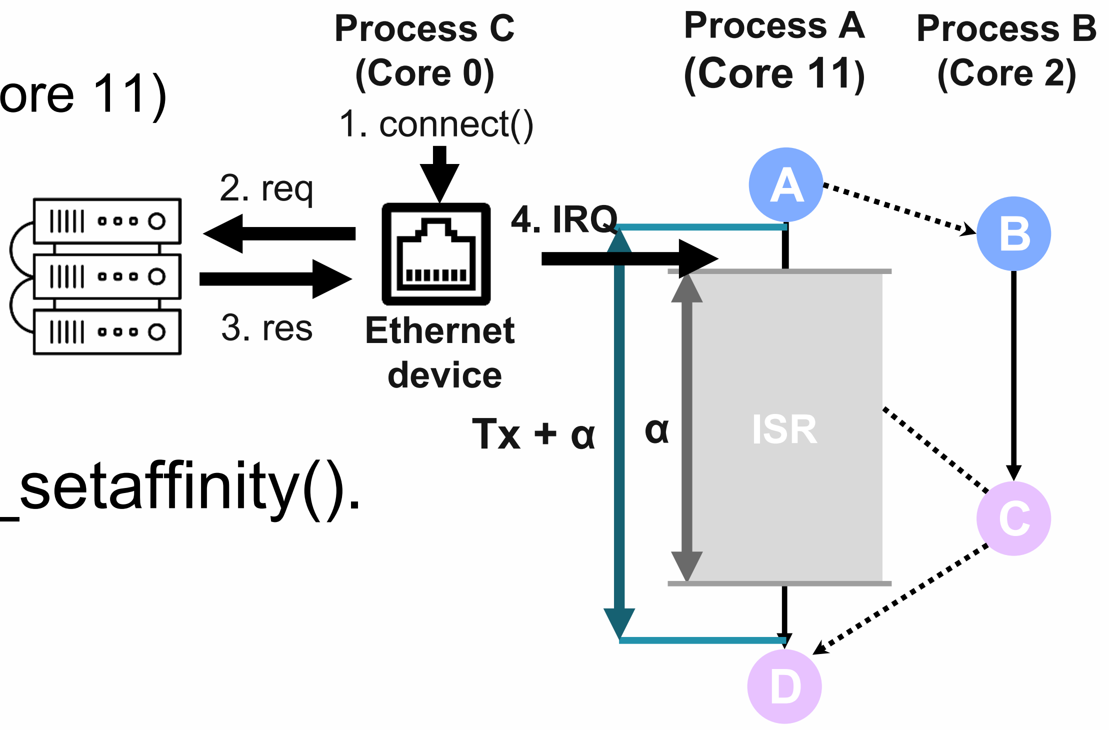
1. Check **irq's core affinity**.

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

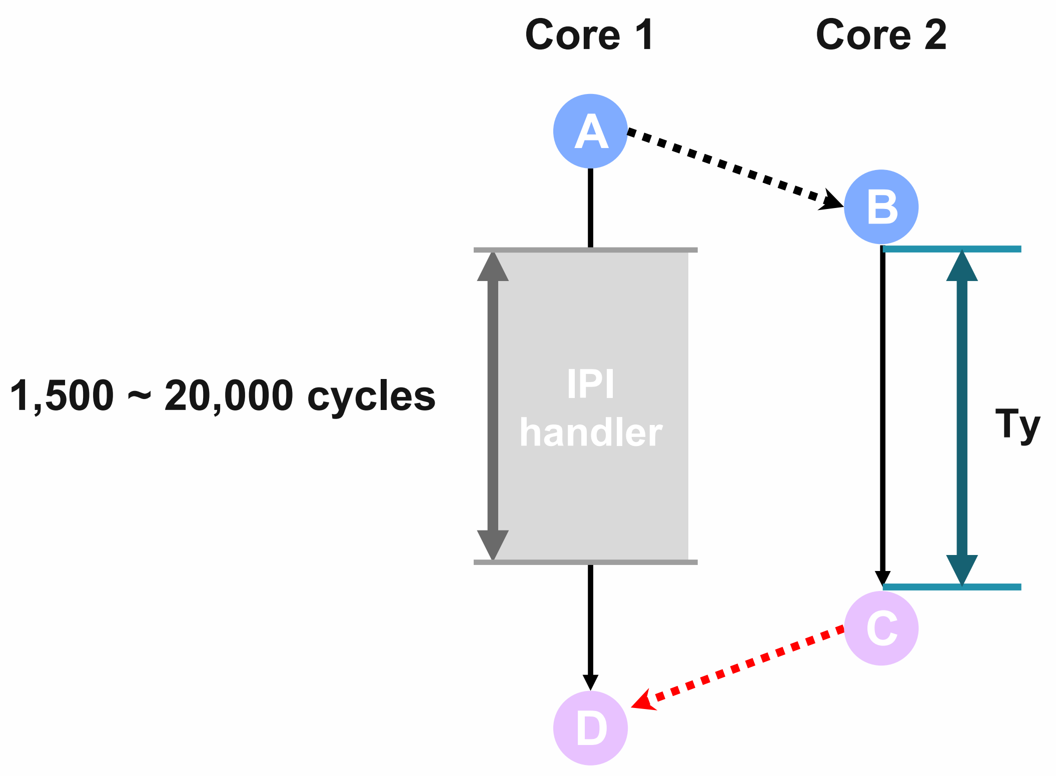


```
yoochan@compsec:~$ cat /proc/irq/122/smp_affinity_list
11
```

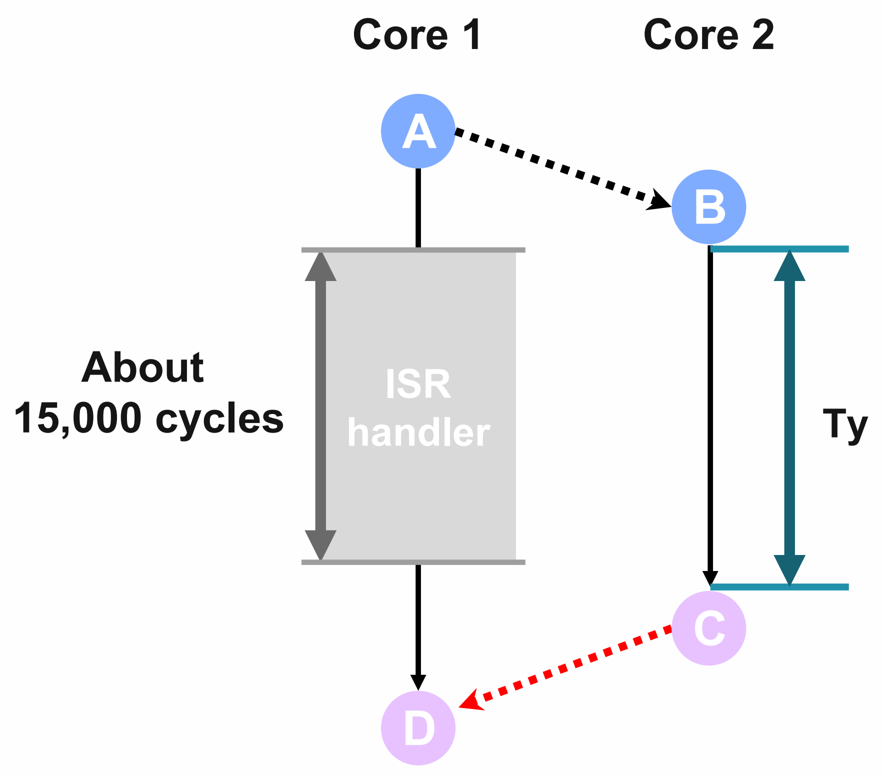
2. Pin the thread to **corresponding core** using sched_setaffinity().



ExpRace : How many cycles are extended?

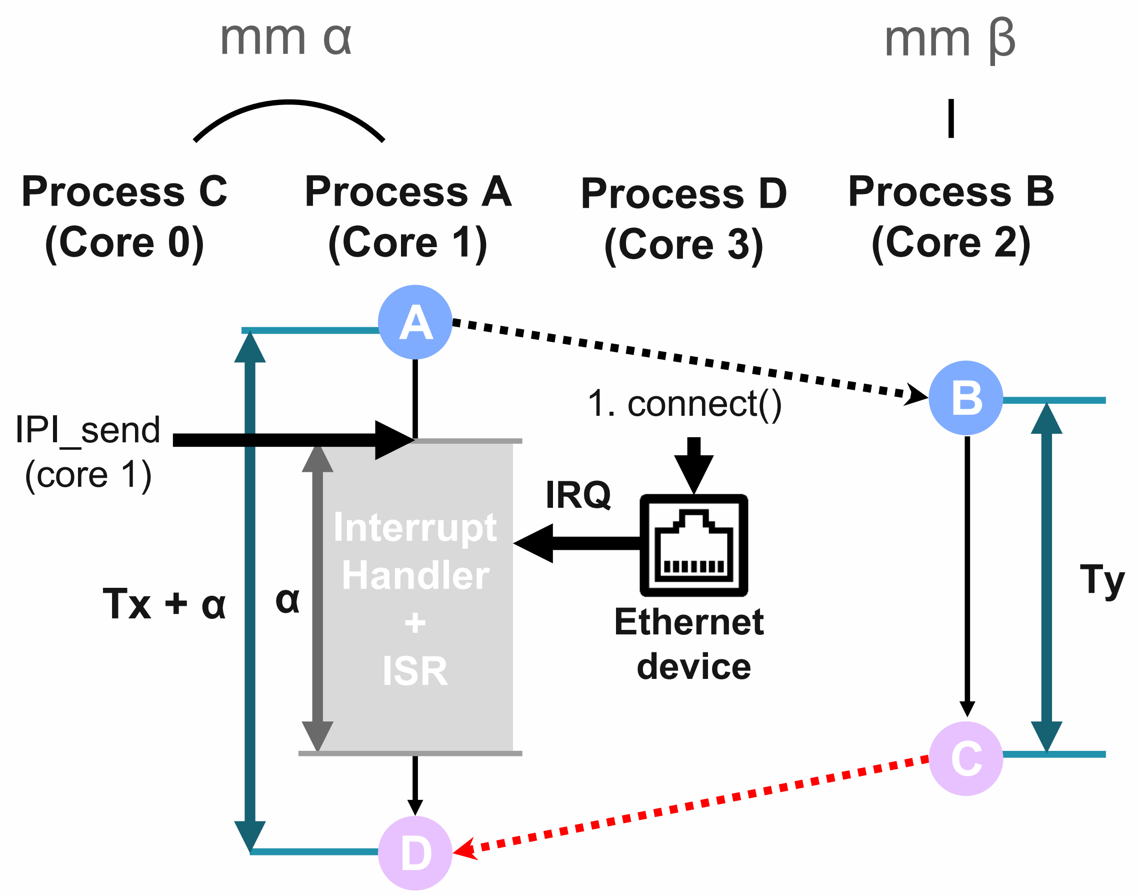


TLB Shootdown



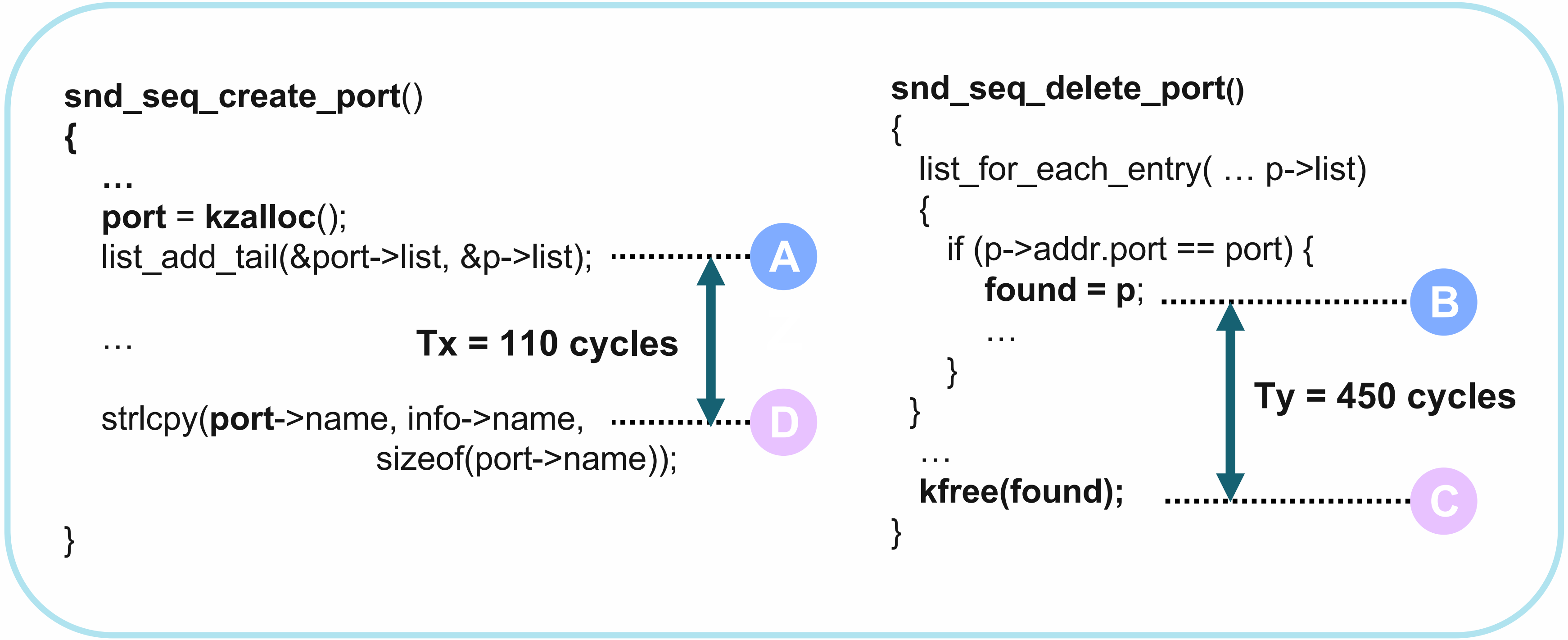
Hardware Interrupt

ExpRace : Advanced Technique



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

Case Study : CVE-2017-15265



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

   …

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

}
```

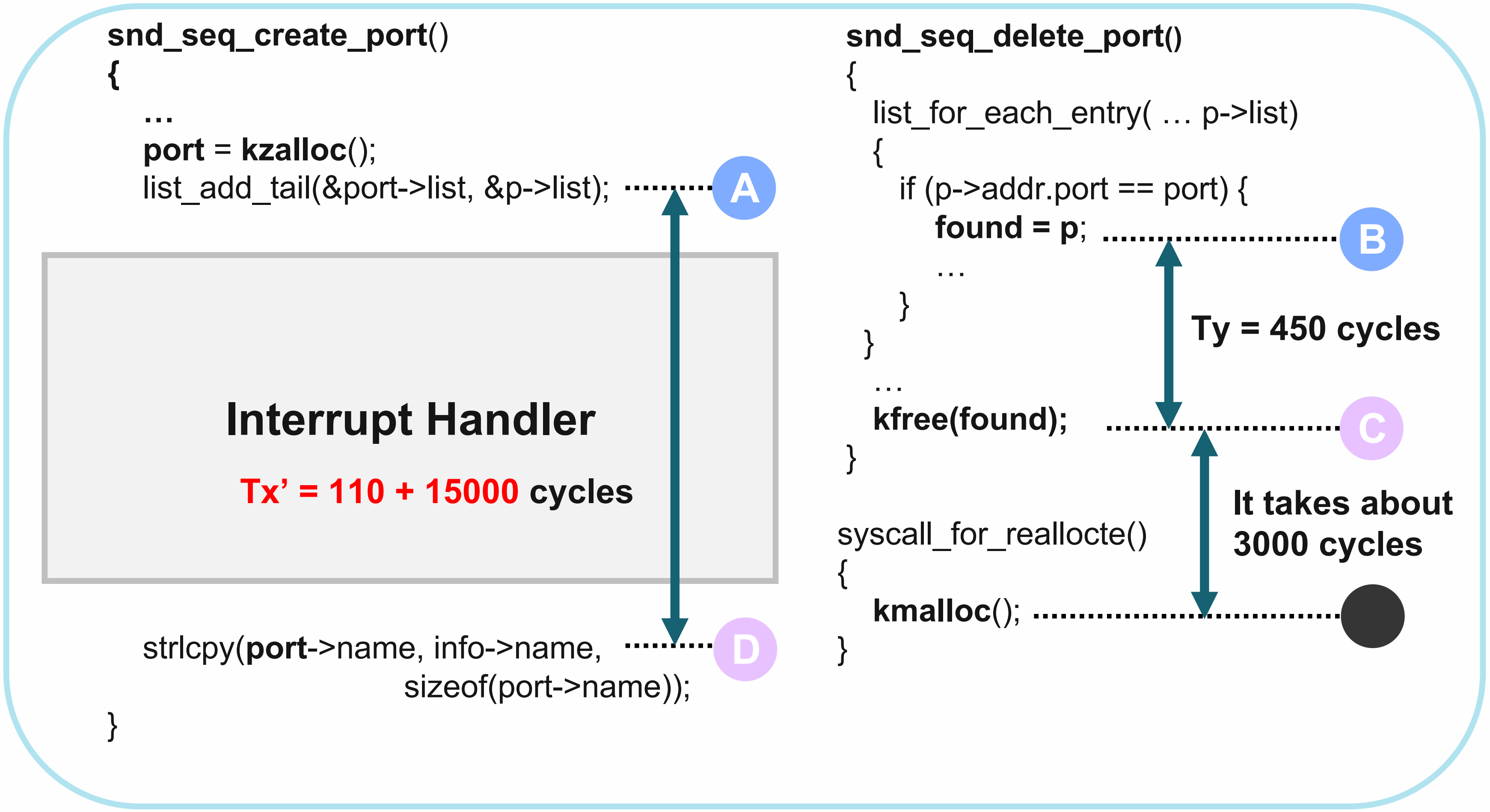
```
snd_seq_delete_port()
{
  list_for_each_entry( … p->list)
  {
    if (p->addr.port == port) {
       found = p;   ........ B
       …
    }
  }
  …
  kfree(found);   ........ C
}
```

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));
}

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

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

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 arbitrary memory write and read.

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

Conclusion

- Introduced **unexploitable** race types.

- ExpRace can turn **unexploitable** races into **exploitable** races.