More Side Channel Defenses: A Cat-and-Mouse Game

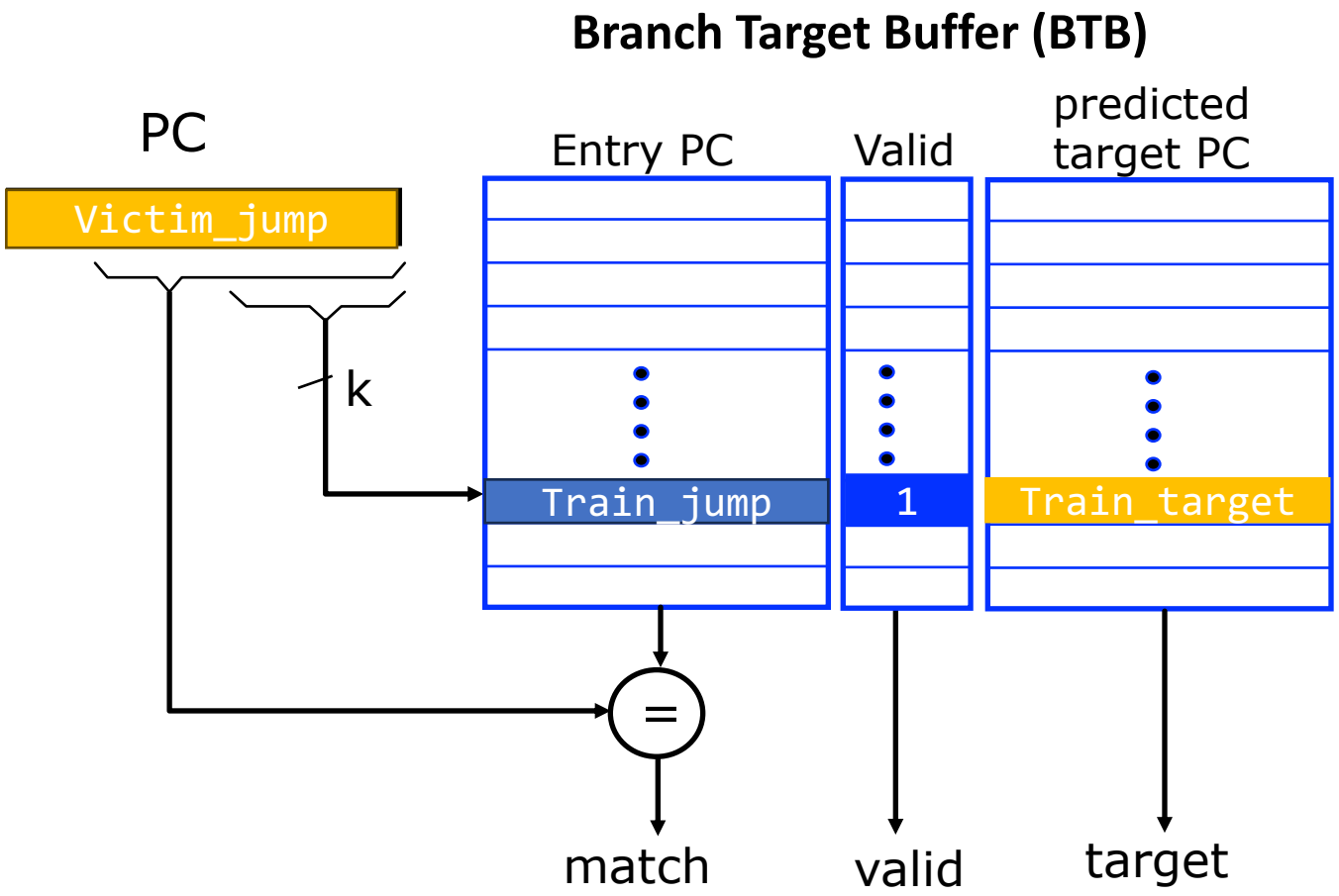
Yuheng Yang

Spring 2025

Based on slides from Prof. Mengjia Yan

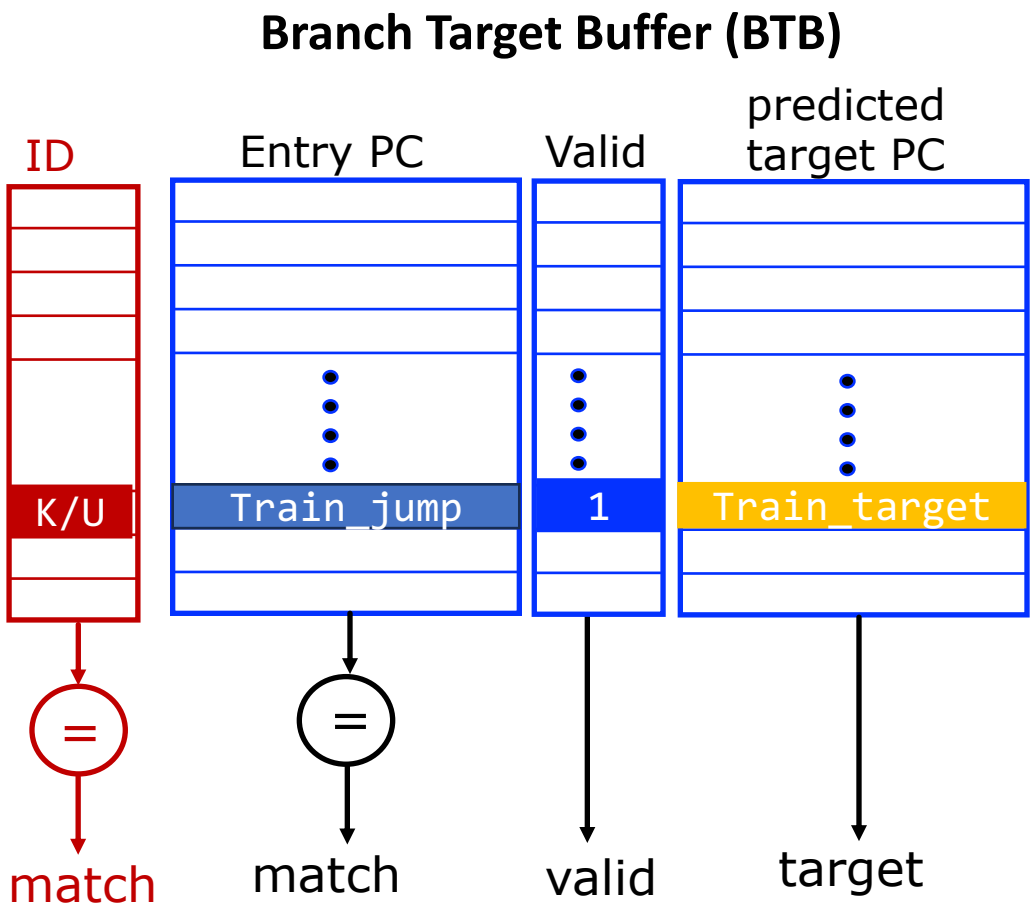
Recall Spectre v2 (BTB Injection)

```
; Attacker code
Train_jump:
    jmp Train_target
    …
; ----CONTEXT SWITCH---
; Victim code
Victim_jump:
    jmp rax
    …
Train_target:
    secret = array1[x]
    y = array2[secret*4096]
    …
```



Deployed Hardware Fixes: eIBRS

eIBRS stands for Enhanced Indirect Branch Restricted Speculation => Isolate BTB entries across privilege levels.



Intel. Indirect Branch Restricted Speculation. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/indirect-branch-restricted-speculation.html

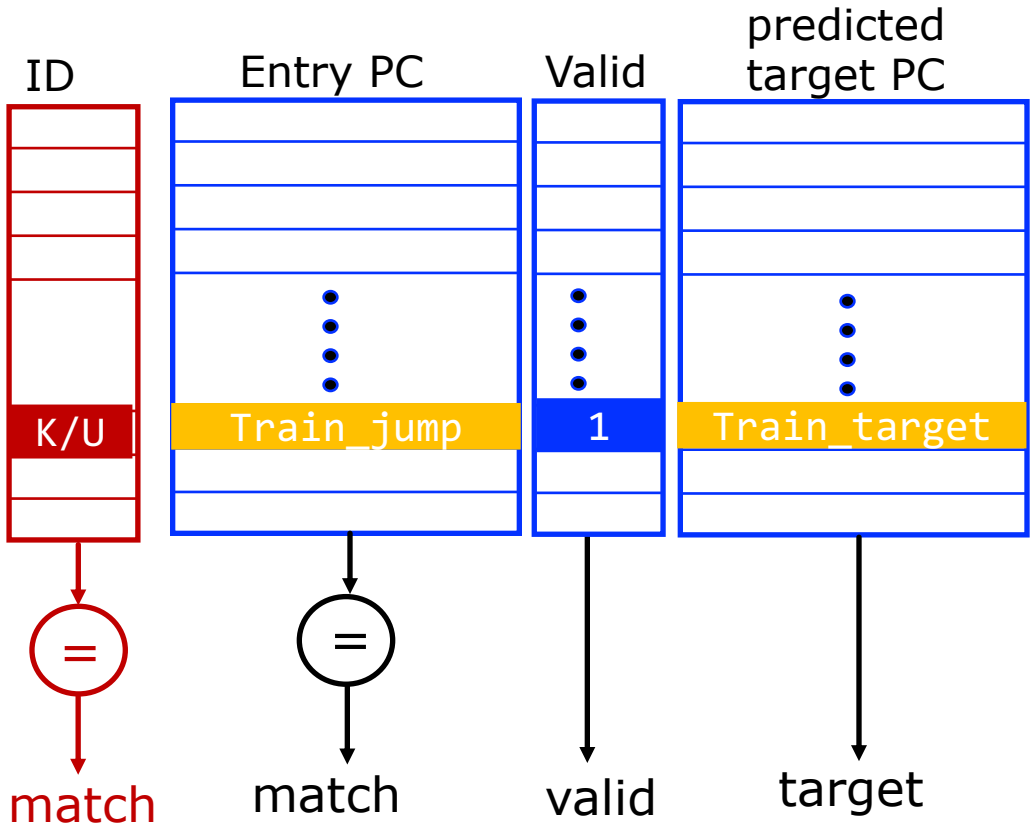
Examine the Security Property

What do we mean by isolation?

- Property #1: ✓
  - Kernelspace indirect branches **do not use** branch target inserted by userspace code.
- Property #2 (non-interference): ✗
  - Userspace code **does not interfere** with Kernelspace indirect branch predictions.

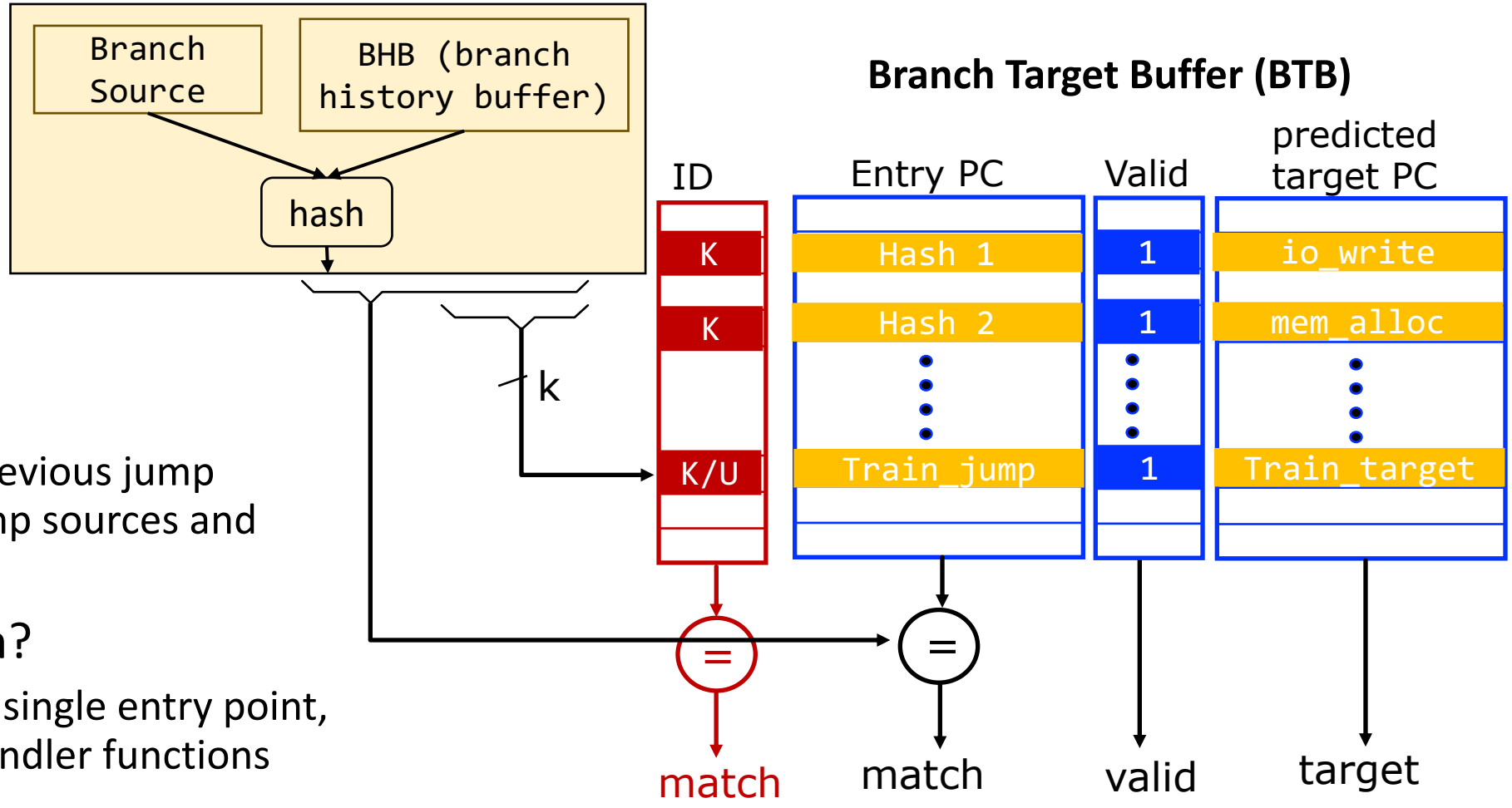
Does eIBRS achieve property #2? If not, counterexamples?

Branch Target Buffer (BTB)



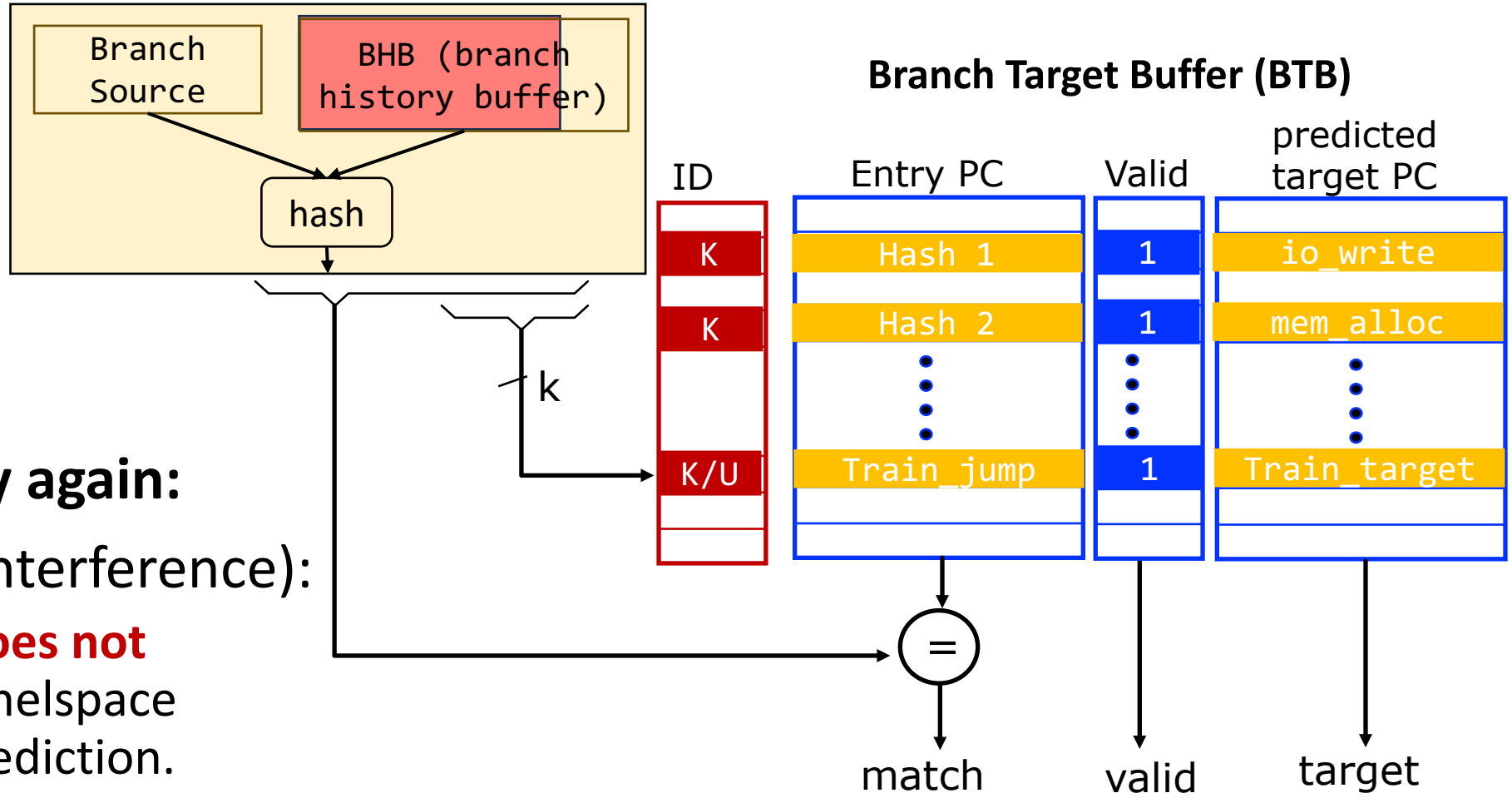
Same-mode misprediction

Surprise 1: How Does BTB Actually Work?



- BHB
  - History information of previous jump instruction, including jump sources and targets
- Why put BHB into hash?
  - E.g., System calls share a single entry point, but will jump to many handler functions

Branch History Injection



Look at the property again:

- Property #2 (non-interference):
  - Userspace code **does not interfere** with Kernelspace indirect branch prediction.

Barberis et al. Branch History Injection: On the Effectiveness of Hardware Mitigations Against Cross-Privilege Spectre-v2 Attacks. USENIX'22 https://www.vusec.net/projects/bhi-spectre-bhb/

Surprise 2: Consequences due to Retpoline

Before retpoline	`jmp *%rax`

After retpoline	`call set_up_target` *(1)*
	
	`capture_spec:` *(4)*
	`pause`
	`lfence`
	`jmp capture_spec`
	
	`set_up_target:`
	`mov %rax, (%rsp)` *(2)*
	`ret` *(3)*

Listing 3 Linux implementation for the Spectre v2 mitigation before version 5.14 on Intel processors depending on eIBRS hardware support. The shown example is taken from the indirect jump in charge to execute the correct syscall handler stored in the `sys_call_table`.

```
do_syscall_64:
    ;...
    mov     rax, [sys_call_table + rax*8]
    call    __x86_indirect_thunk_rax
```

```
;with eIBRS support
__x86_indirect_thunk_rax:
    jmp     rax
```

Perfect victim branch for BTB attack

```
;without eIBRS support (retpoline)
__x86_indirect_thunk_rax:
    call    B
A:  pause
    lfence
    jmp     A
B:  mov     [rsp], rax
    ret
```

Google. Retpoline: a software construct for preventing branch-target-injection https://support.google.com/faqs/answer/7625886

Summary: The Cat-and-Mouse Game

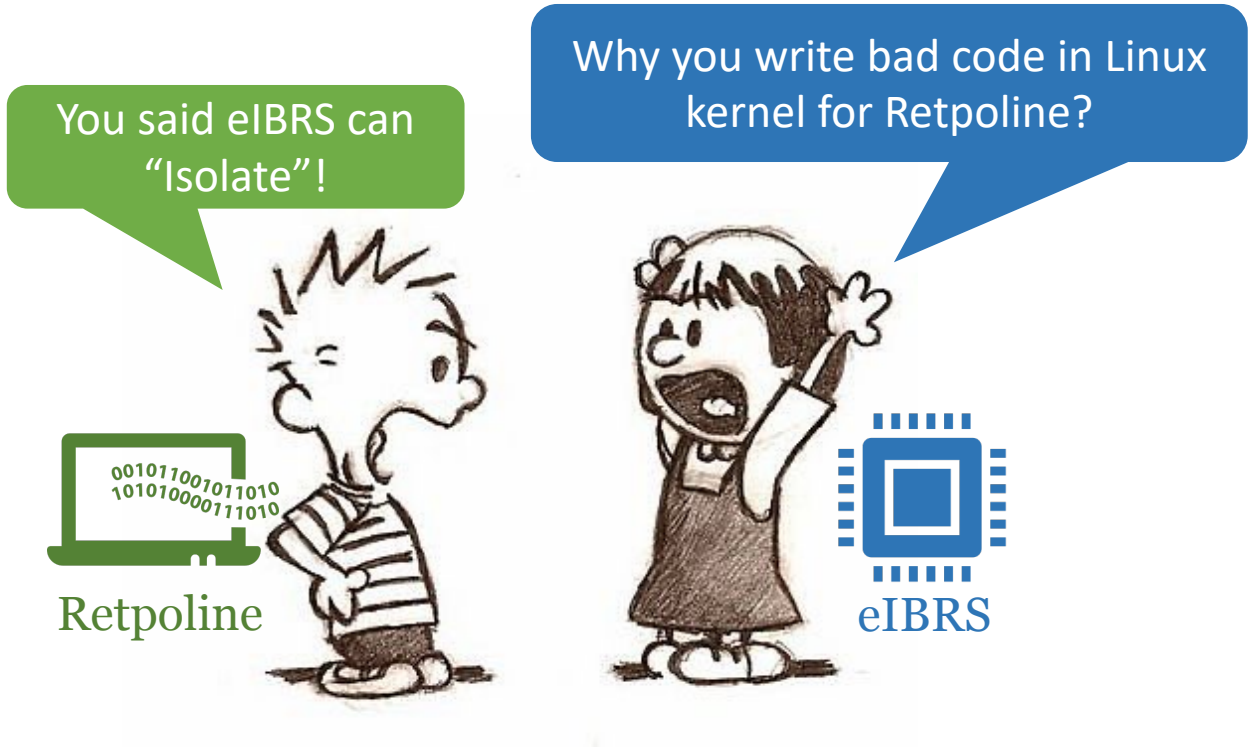
Spectre v2 (BTB Injection)

eIBRS

Branch History Injection

Consequences due to Retpoline

Why hardware designers fail to make eIBRS secure?

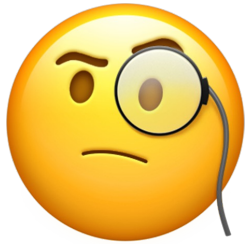


Solution to the Fight

- Goal: communicate security property achieved by hardware defenses
  - The bad example: eIBRS -> unclear what exactly “isolation” mean…

- Alternative approaches:
  - Approach 1: Show SW people all the HW implementation details

  - Approach 2: define new SW-HW contracts



SW-HW Contracts for Secure Speculation

Contract #1: Make Speculation Invisible

- Idea: make speculative executed instructions' microarchitecture effects invisible by the attacker
- Examine program examples

```
if (false)
  sec = ld x
  dummy = ld sec
```

```
sec = ld x

if (false)
  dummy = ld sec
```

```
sec = ld x
dummy = ld sec

if (false)
  ……
```

Secure if using invisible speculation?

Do they follow constant-time programming?

Speculative Non-interference

- Some notations
  - P: a deterministic program
  - M_{pub}: public memory and inputs
  - M_{sec} : secret memory and inputs
  - O: microarchitecture observation (traces)
- Property:
  - **if** the SW does not leak under the constant-time programming model
  - **then** the HW should ensure no more secrets leaked under speculation

$$\forall\ P, M_{pub}, M_{sec}, M'_{sec},$$
$$\text{IF} \quad O_{seq}(P, M_{pub}, M_{sec}) = O_{seq}(P, M_{pub}, M'_{sec})$$
$$\text{THEN} \quad O_{spec}(P, M_{pub}, M_{sec}) = O_{spec}(P, M_{pub}, M'_{sec})$$

Execute program **sequentially**, monitor memory addresses.

Execute program **speculatively**, monitor memory addresses.

Hardware-Software Contracts for Secure Speculation; Guarnieri et al; S&P'19

Is Speculative Non-interference Achieved?

We prepared 3 out-of-order processors in our Visualized Simulator

ALU has 3 ports, operating in parallel

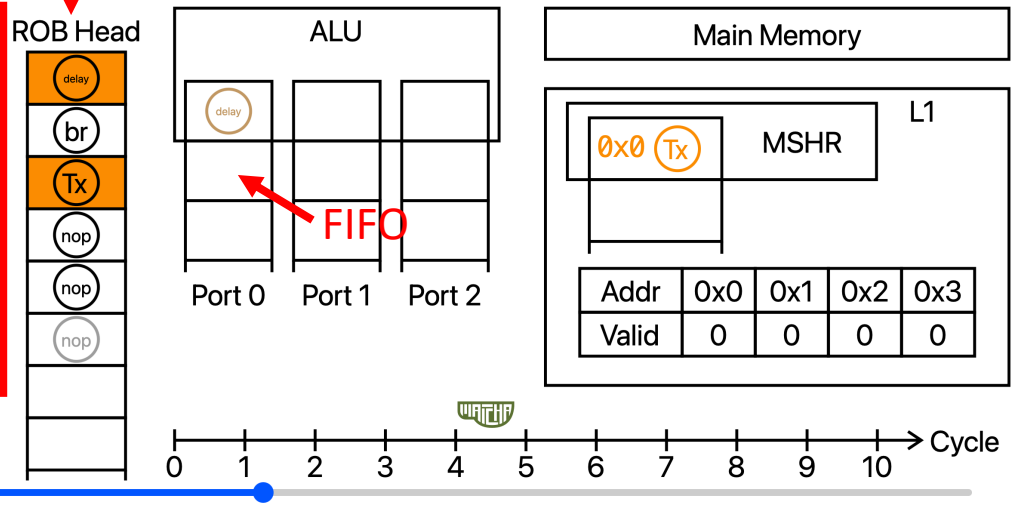
ROB has infinite entries

Memory has 4 entries

You provide a program with:

- 4 types of instructions: ALU, Branch, Load, NOP
- 8 registers: r0-r7
  - r0 is constant 0
  - r7 is also named as rSec

```
r1 <- ALU-0(r0, lat=4)
IF (r1==0) PC += 6
__ <- LOAD(rSec)
NOP
NOP
NOP
```



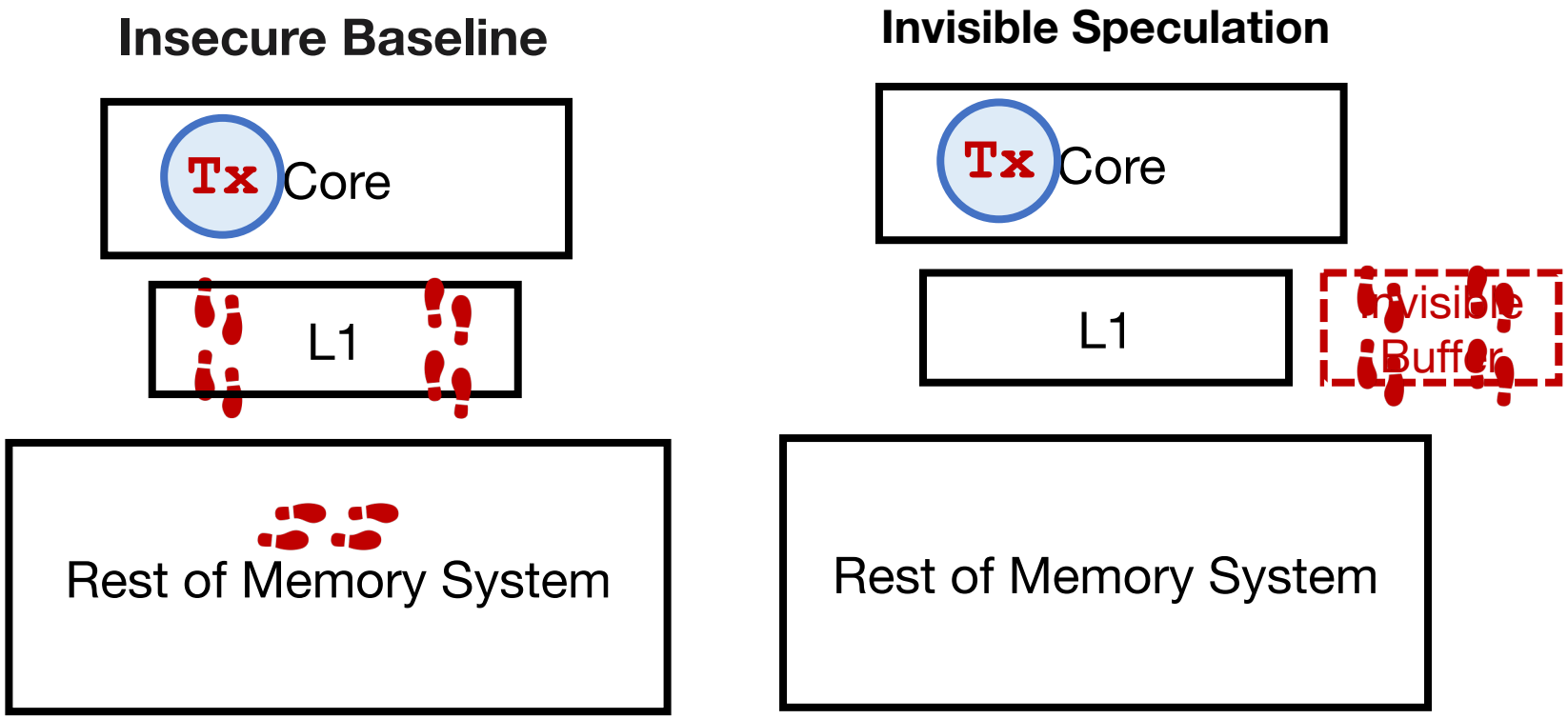
On L1 miss, wait 3 cycles in MSHR

L1 can cache all 4 entries
Hit takes 1 cycle

(Jupyter Notebook link: https://mybinder.org/v2/gh/yuhengy/SHD-SpectreDemo/HEAD?urlpath=%2Fdoc%2Ftree%2F2-attackProcessors.ipynb)

Defense #1: InvisiSpec

```
if (false)
  dummy = ld sec //Tx
```

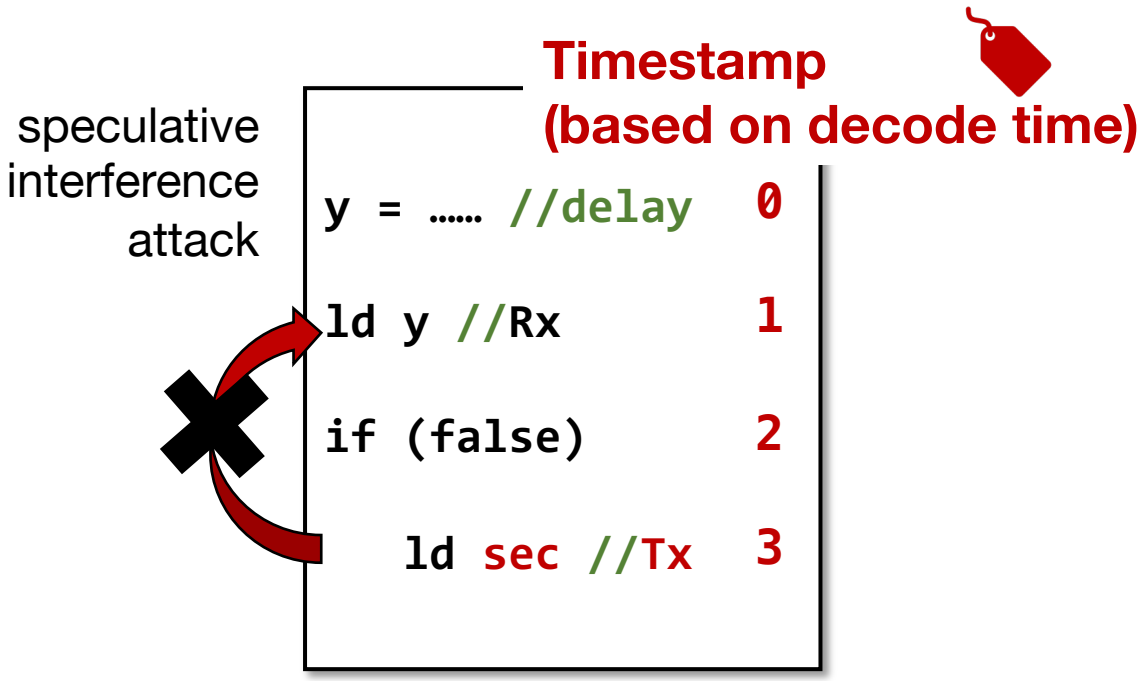


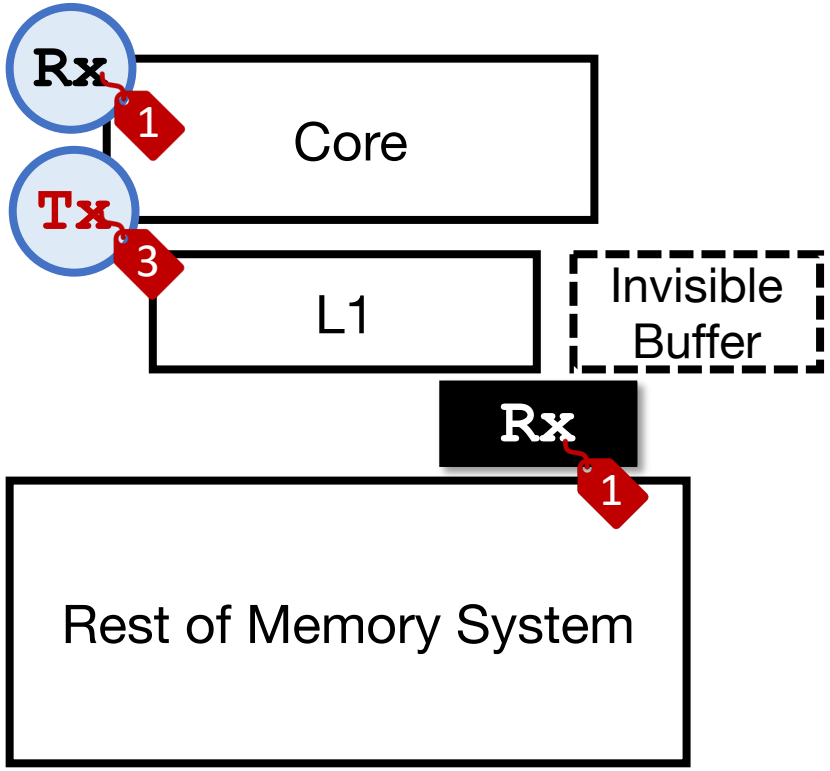
InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy; Yan et al; MICRO'18

Defense #2: GhostMinion

#1: Invisible Speculation

#2: Prioritize Older Instructions through Timestamps





GhostMinion: A Strictness-Ordered Cache System for Spectre Mitigation; Ainsworth; MICRO'21

Summary: The Cat-and-Mouse Game

2018 Spectre

2019 InvisiSpec
Delay-on-Miss …

2020 Speculative interference attack

2021 GhostMinion

2023 New variant of speculative interference attack

Need tools for automatically discovering vulnerabilities

More Contracts

Contract #2: Relax the Security Property

- Idea: only protect speculatively loaded data

Spectre v1

```
if (false)
  sec = ld x
  dummy = ld sec
```

```
sec = ld x

if (false)
  dummy = ld sec
```

```
sec = ld x
dummy = ld sec

if (false)
  ……
```

Secure if using invisible speculation?

Secure if only protecting speculatively loaded data?

SW needs to follow constant-time programming

Software sandboxing ?

STT and NDA Designs

- Draw on the board

Speculative Taint Tracking (STT): A Comprehensive Protection for Speculatively Accessed Data; Jiyong Yu, Mengjia Yan, et al; MICRO, 2019
NDA: Preventing Speculative Execution Attacks at Their Source; Ofir Weisse, et al; MICRO, 2019

Understand the Property/Contract

Speculative non-interference: HW that can protect constant-time programs.

$$\forall\ P, M_{pub}, M_{sec}, M'_{sec},$$
$$\text{IF} \quad O_{seq}(P, M_{pub}, M_{sec}) = O_{seq}(P, M_{pub}, M'_{sec})$$
$$\text{THEN} \quad O_{spec}(P, M_{pub}, M_{sec}) = O_{spec}(P, M_{pub}, M'_{sec})$$

Execute program **sequentially**,
Monitor architecture registers

Execute program **speculatively**,
monitor memory addresses.

Can also be used to describe the case for protecting software sandboxing…

Summary of SW-HW Contracts

$$\forall\ P, M_{pub}, M_{sec}, M'_{sec},$$
$$\text{IF} \quad O_{seq}(P, M_{pub}, M_{sec}) = O_{seq}(P, M_{pub}, M'_{sec})$$
$$\text{THEN} \quad O_{spec}(P, M_{pub}, M_{sec}) = O_{spec}(P, M_{pub}, M'_{sec})$$

Describe what SW needs to achieve

Describe what HW needs to achieve for **only** the SW that satisfies the IF statement

- The payoff: we can check security properties for SW and HW independently
- Ongoing research: How to check and design according to these properties?