Spectre Mitigations: A Cat-and-Mouse Game

Mengjia Yan

Spring 2026

Last Lecture: Constant-time Programming

- **In theory: the non-interference property**
  - For any secret values, a program always takes the same amount of time for the same input when executing on the same machine, and this holds for arbitrary public inputs.
- **In practice:**
  - No secret-dependent branch
  - No secret-dependent memory accesses
  - No secret-dependent non-constant-time arithmetic operations

- Given a software mitigation, understand the associated hardware assumptions

By the end of last lecture,
we have not discussed how to deal with speculation…

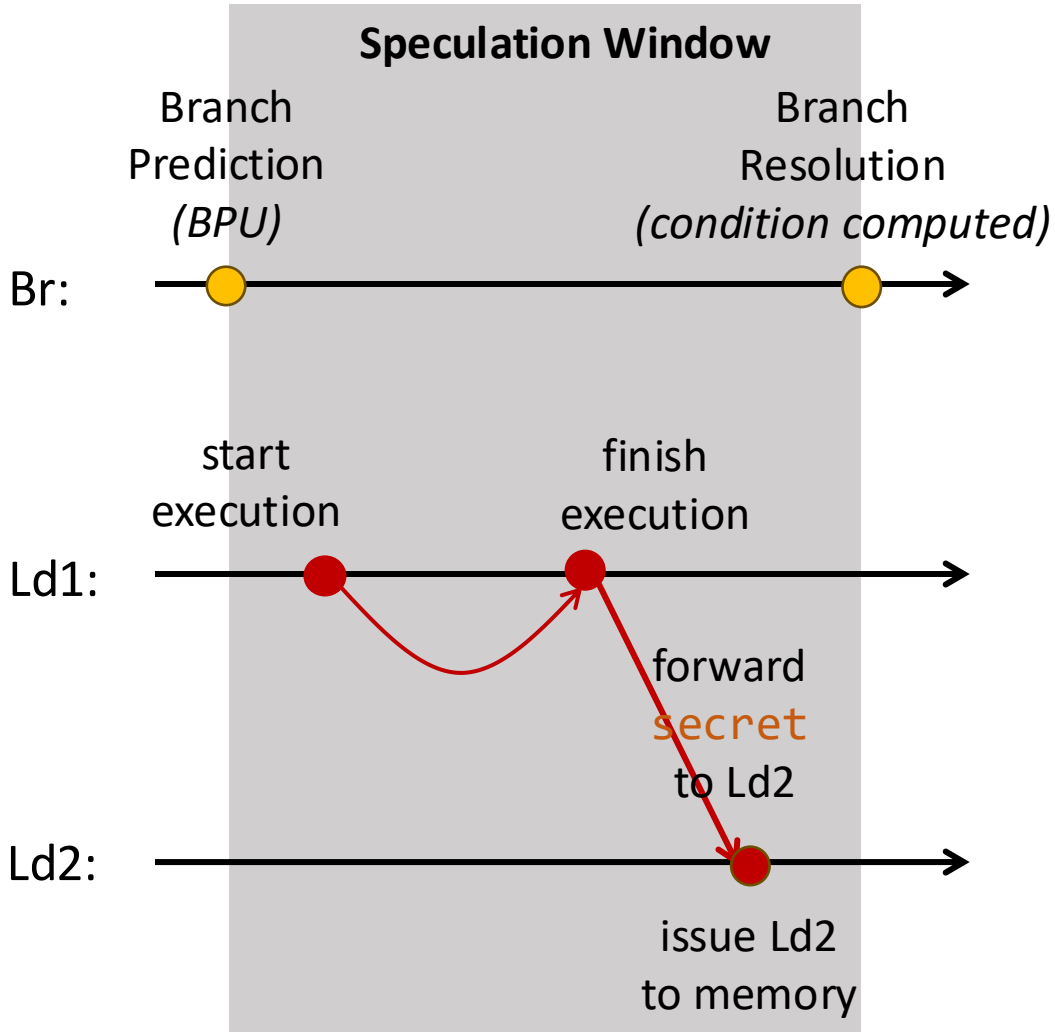
Spectre v1
Timing and Mitigations

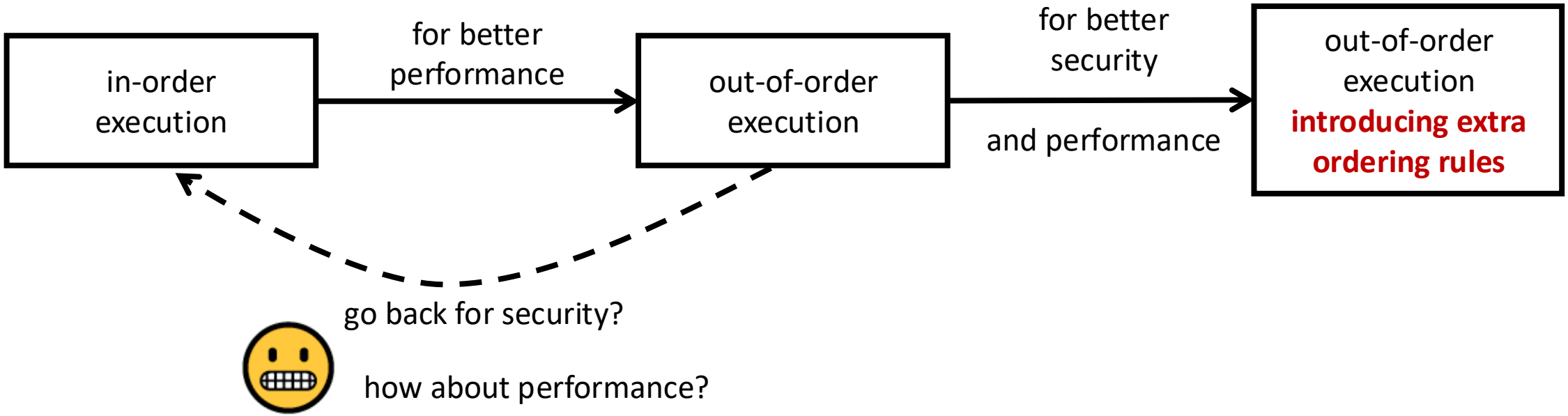
Spectre v1 Timing

```
Br:  if (x < size_array1) {
Ld1:       secret = array1[x]
Ld2:       y = array2[secret*64]
     }
```

Among all the events listed in the figure, which "happens-before" relationship is essential for the attack to succeed?

Ld2's mem stage -> Br's resolution time
(-> means "happens before")




in-order execution
for better performance
out-of-order execution
for better security
and performance
out-of-order execution
introducing extra ordering rules
go back for security?
how about performance?

Inserting Fence

```
Br:  if (x < size_array1) {
Ld1:       secret = array1[x]
Ld2:       y = array2[secret*64]
     }
```

```
cmp rsi, rdx

jae .OOB ; jump if out-of-bounds
fence
mov rax, [array+rsi] ; sensitive load
```

- Recall from the cache attack lecture

  `fence`: Memory Fence
  Performs a serializing operation on all memory instructions

- Question: where to insert fence?

Inserting Fence

```
Br:  if (x < size_array1) {
Ld1:       secret = array1[x]
Ld2:       y = array2[secret*64]
     }
```

```
cmp rsi, rdx
ja .OOB_taken ; if taken

……    ;fall-through, other code

.OOB_taken:
fence
mov rax, [array+rsi] ; sensitive load
```

- Recall from the cache attack lecture

  `fence`: Memory Fence
  Performs a serializing operation on all memory instructions

- Question: where to insert fence?

Introducing Dependency

```
Br:  if (x < size_array) {
Ld1:       secret = array[x]
……
}
```

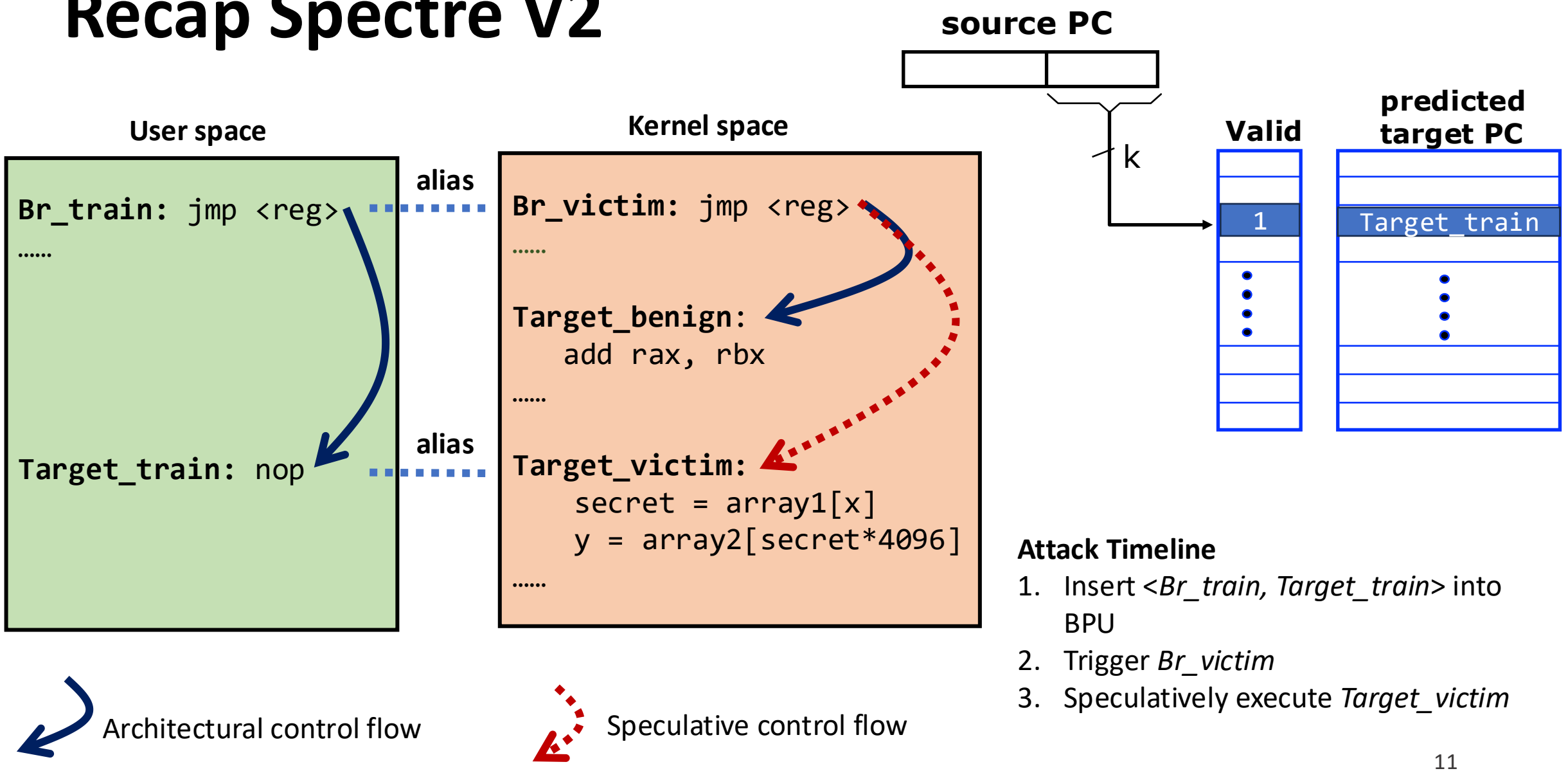
- Two types of dependencies:
  - Data dependency — Usually do not speculate
  - Control-flow dependency — speculate

```
mask=(unsigned long)(-(x < size_array));
//mask is 0xFFFFFFFF.. or 0x0
secret = array[x];
secret &= mask
//misspeculation: secret becomes 0
```

- **Speculative Load Hardening (SLH)** introduces data dependency between Br and Ld
  - Supported by clang/llvm, compile with flag `-mspeculative-load-hardening`

Spectre v2 Mitigations
A cat and mouse game

Recap Spectre V2



Attack Timeline

1. Insert <*Br_train, Target_train*> into BPU
2. Trigger *Br_victim*
3. Speculatively execute *Target_victim*

Retpoline

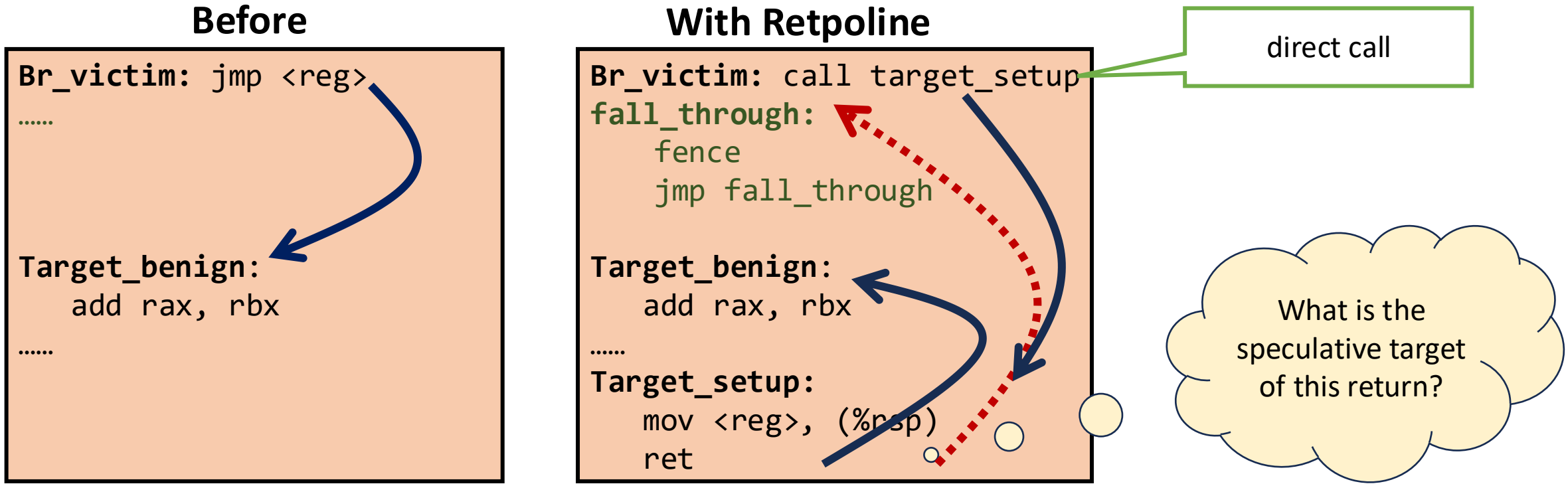
- Idea: convert an indirect jump to a direct jump and a return

- How “call” and “return” work?
  - rsp in x86 == ra in RISCV
  - call label/<reg>: put <PC+4> into %rsp, then jump to label/<reg>
  - return: jump to %rsp

- How to predict?
  - Return stack buffer (RSB)

Retpoline

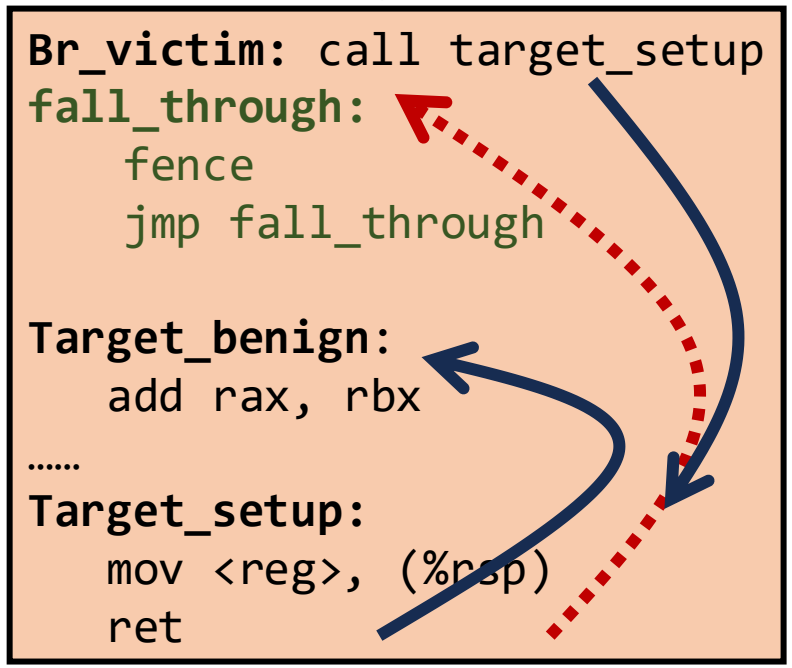
- Idea: convert an indirect jump to a direct jump and a return



https://support.google.com/faqs/answer/7625886

Retpoline Security Vulnerabilities

With Retpoline



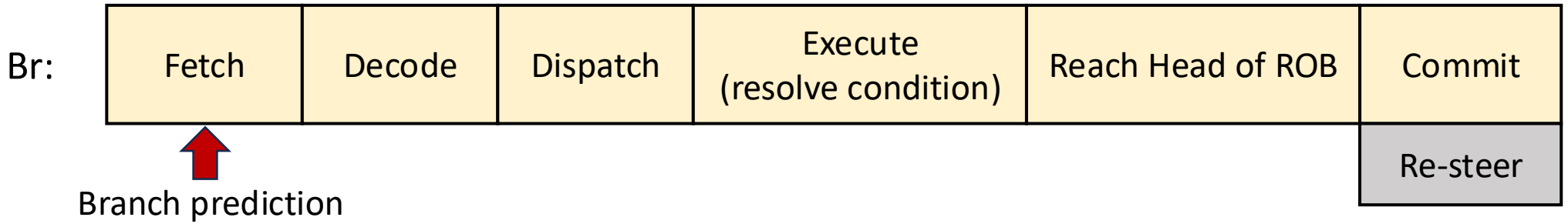
- Two hardware assumptions:
  1. Direct call/jmp will not have misprediction
  2. Return will only use RSB for prediction
- Phantom breaks assumption 1
  - Prediction happens before instruction decoding
- Retbleed breaks assumption 2
  - When RSB underflow, BTB is used for predicting return

Wikner et. al., Phantom: Exploiting decoder-detectable mispredictions. MICRO'23
Wikner et. al. RETBLEED: Arbitrary speculative code execution with return instructions. USENIX'22

Phantom

Q: Where does branch prediction happen?

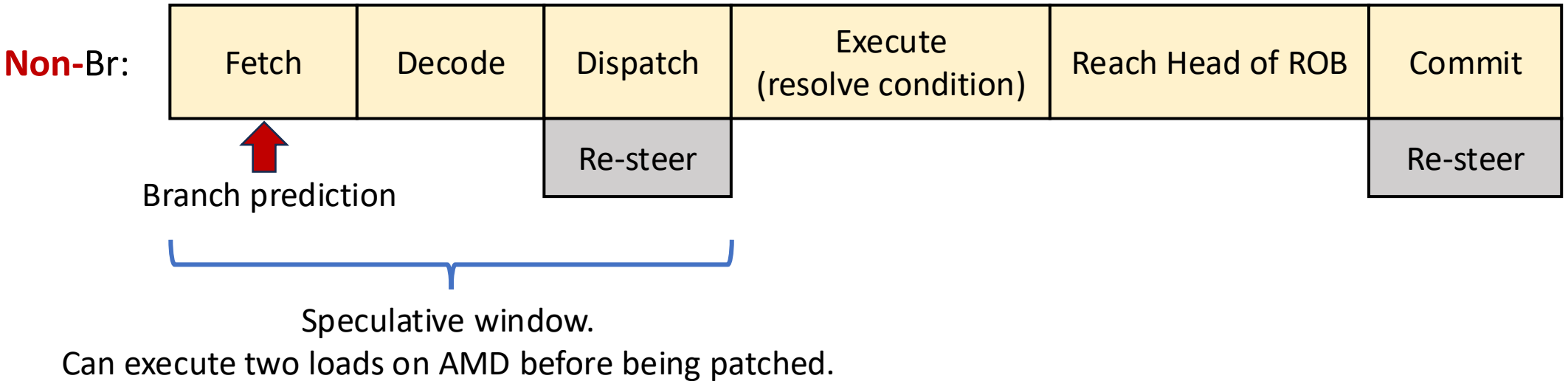
Fetch. BTB is indexed by PC, and a valid bit indicates whether this PC holds a branch.



Phantom

Q: Where does branch prediction happen?

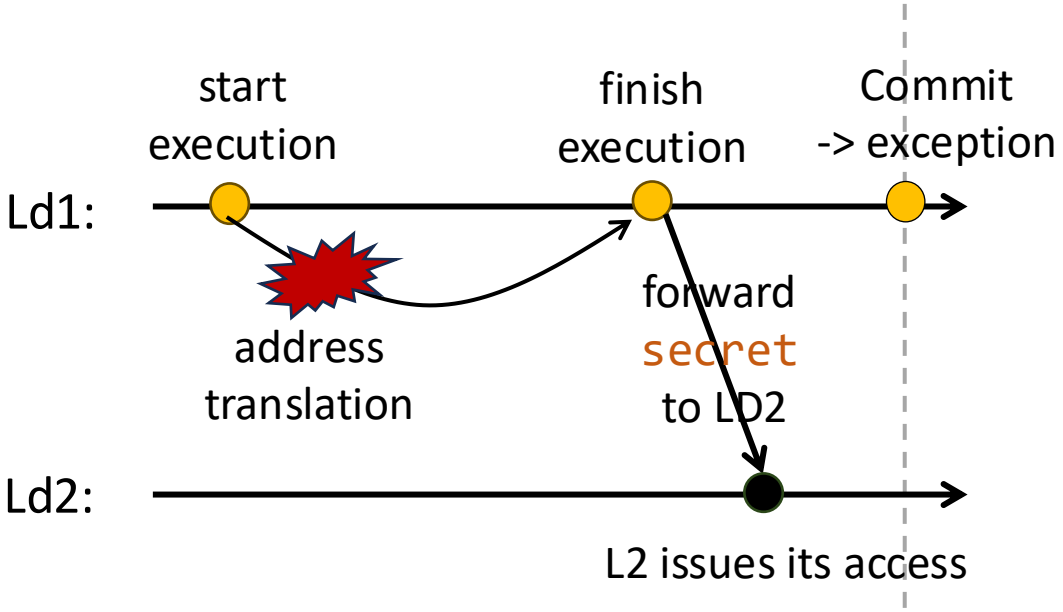
Fetch. BTB is indexed by PC, and a valid bit indicates whether this PC holds a branch.

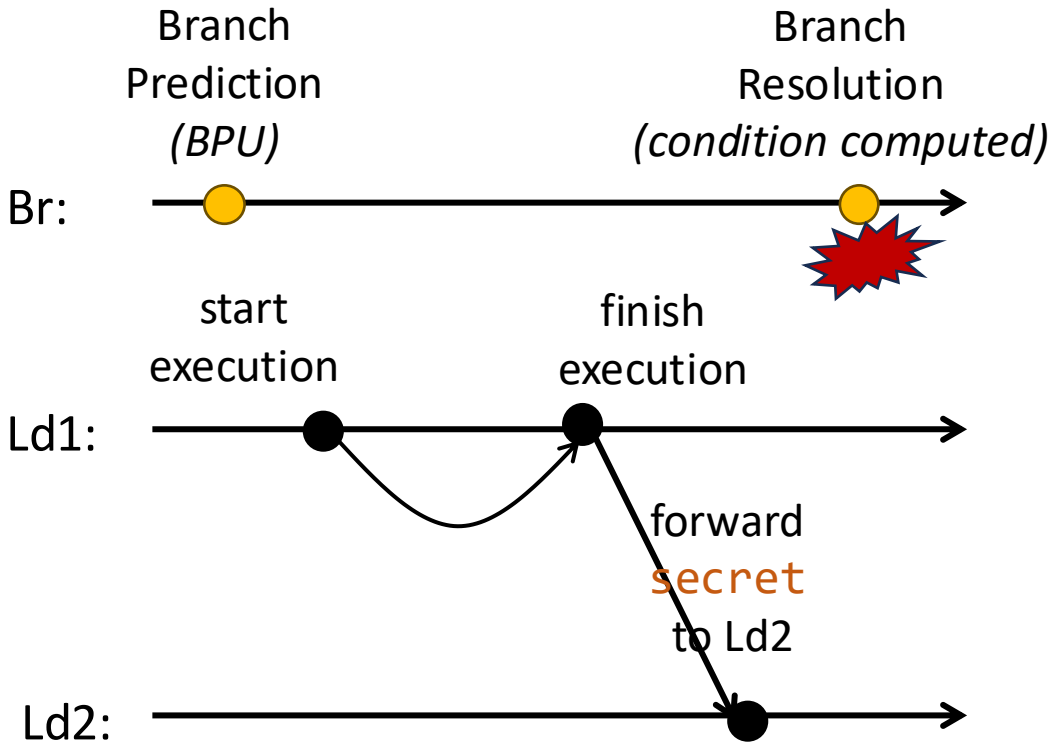


Can execute two loads on AMD before being patched.

Why more difficult to mitigate Spectre?

- The key is when a misspeculation is determined
- Stall until confirming prediction correctness → performance loss



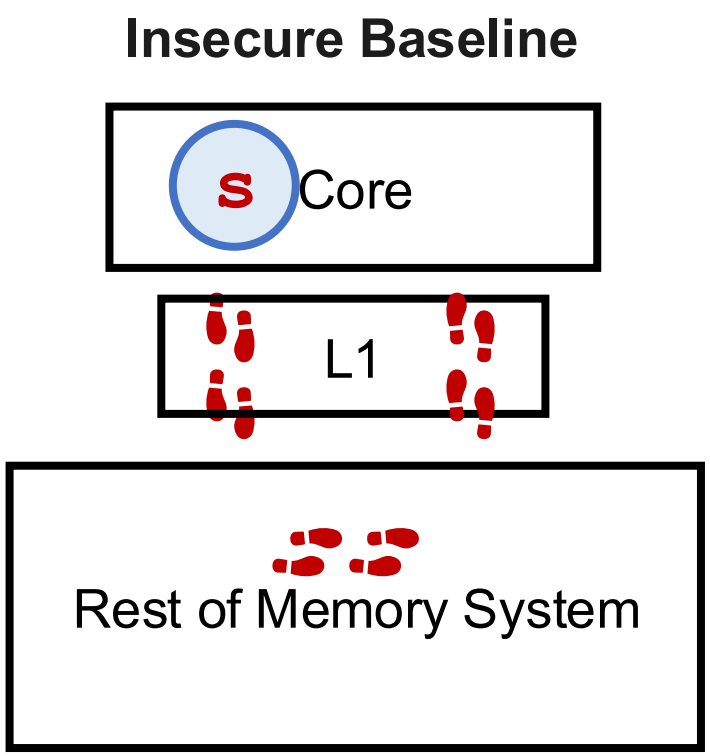


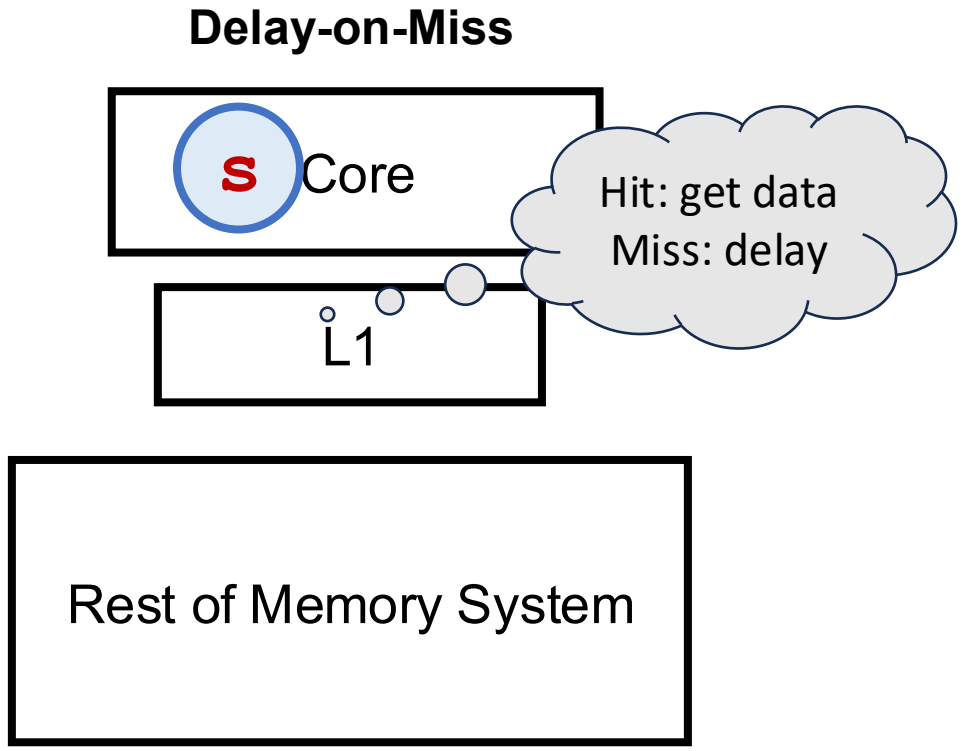
Hardware solutions from academia

- Goal: try to securely perform some execution before speculation is resolved, so we gain some performance back
- Challenge: again a cat and mouse game
- Useful resource:
  - visualization tool to see speculative execution and mitigations in action
  - Github repo: https://github.com/yuhengy/SHD-SpectreDemo
- Assume in the following example the attacker can measure latency of every committed instruction

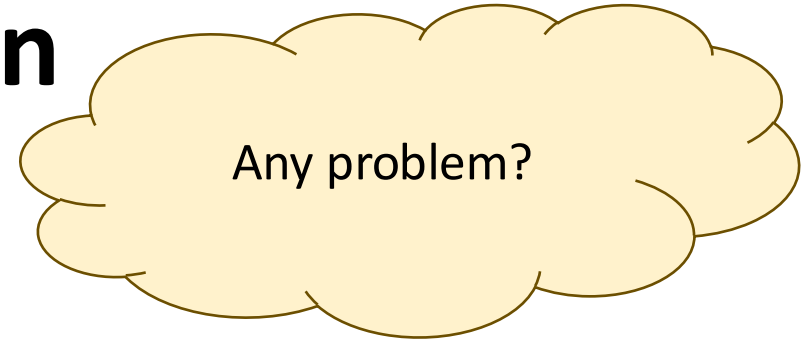
Scheme #1: DoM

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

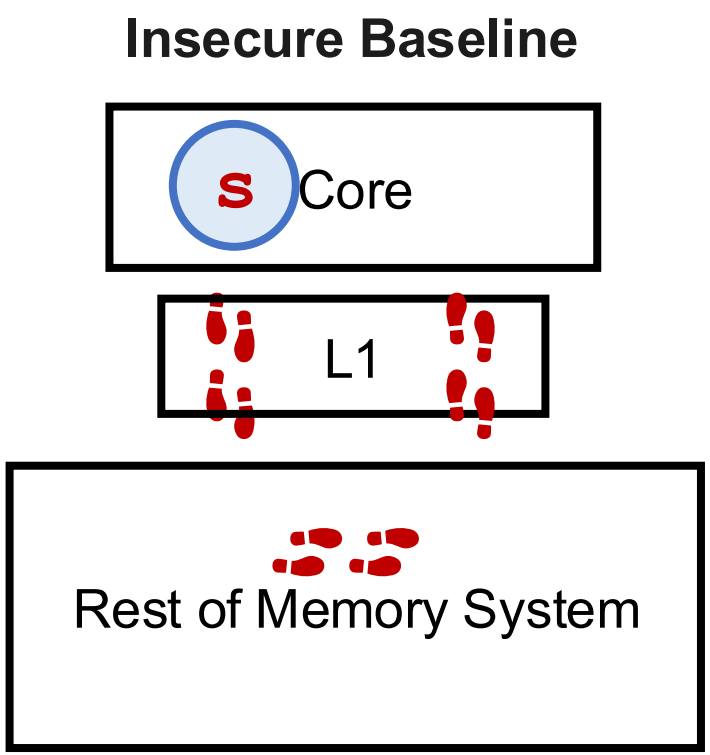


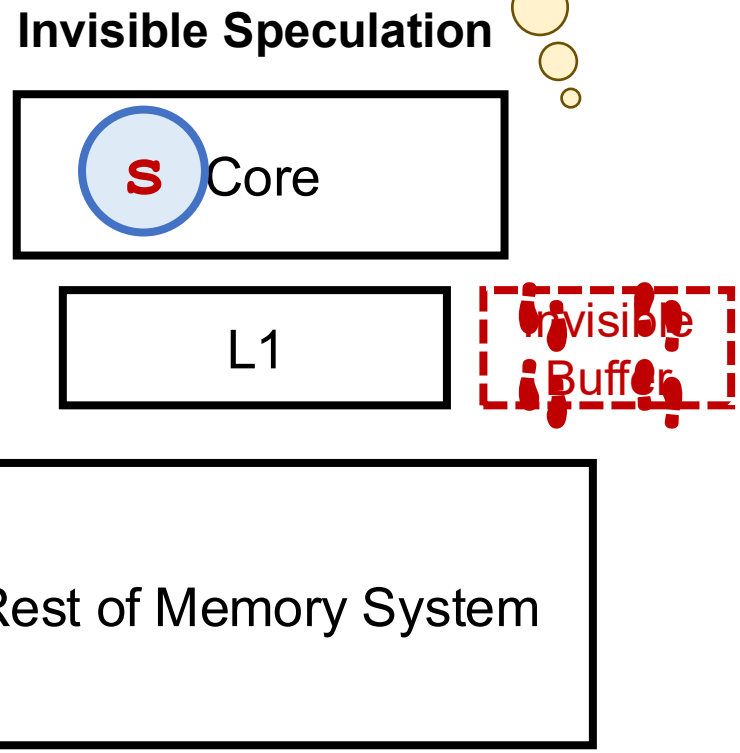


Scheme #2: Invisible Speculation




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



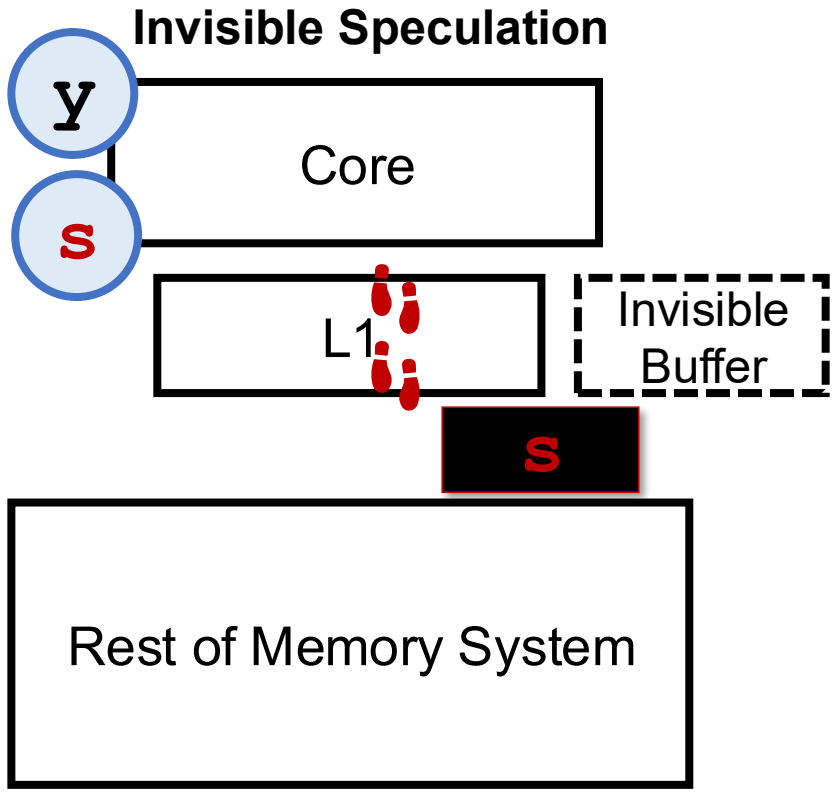


InvisiSpec: Making Speculative Execution Invisible in the Cache Hierarchy; Yan et al; MICRO'18

Speculative Interference Attack

- **Younger** speculative loads interfere with **older** bound-to-commit loads.
- Many other contention structures: non-pipelined ALU, cache port, bank contention, network-on-chip, etc.

```
y = …… //delay
ld y //transmitter
if (false)
    ld sec //interfere
```

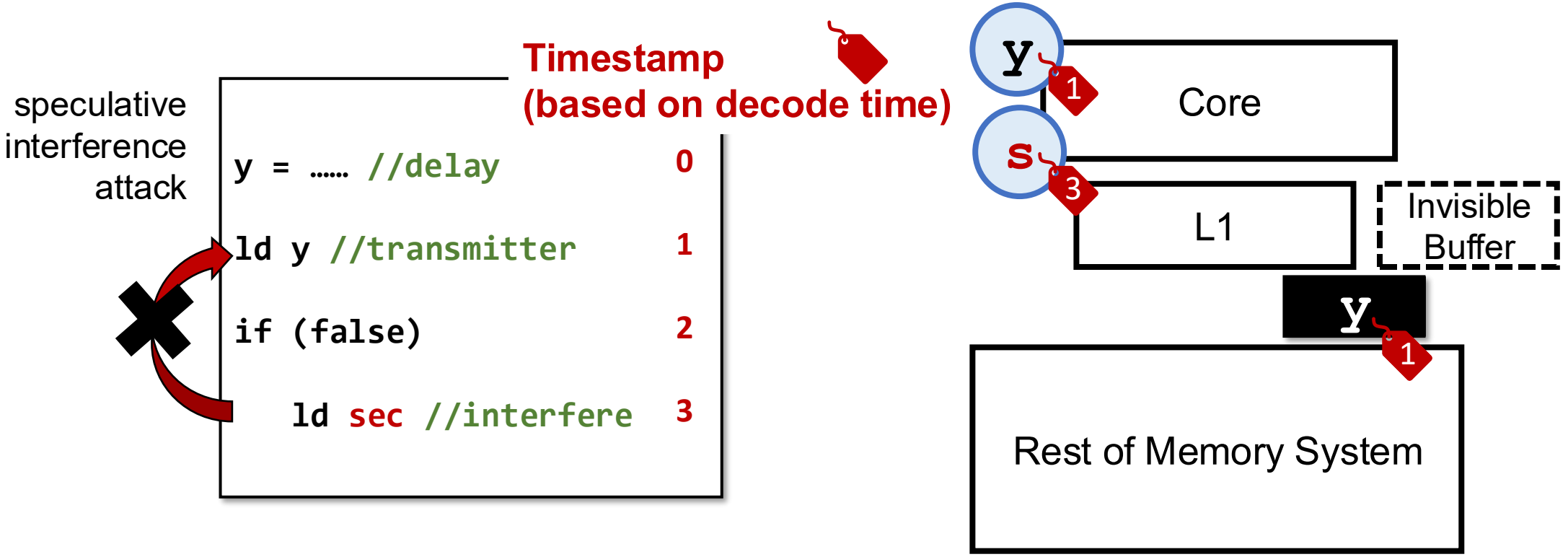


Speculative interference attacks: breaking invisible speculation schemes; Behnia et al; ASPLOS'21

GhostMinion

#1: Invisible Speculation

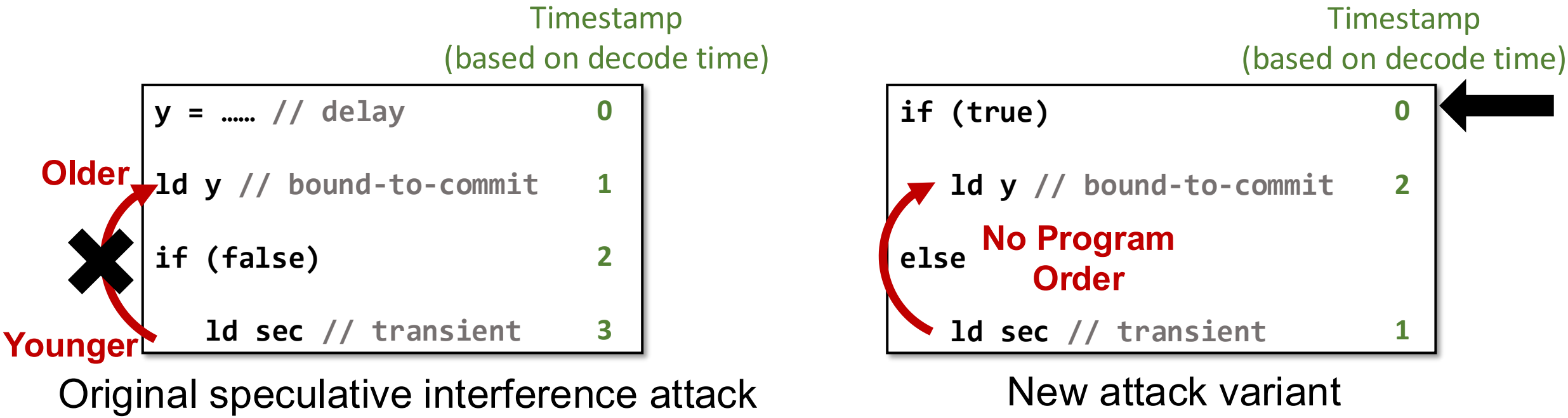
#2: Prioritize Older Instructions through Timestamps



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

New Attack Variant

GhostMinion prioritizes **smaller** timestamps



Pensieve: Microarchitectural Modeling for Security Evaluation; Yang et al; ISCA'23

Summary

- Spectre exploits the gap between *early microarchitectural effects* and *late speculation resolution*.

- Security is about precise ordering of microarchitectural
- Mitigations encode hardware assumptions; when those assumptions fail, attacks resurface.

- Performance vs. strict ordering remains the fundamental tension.

Next:
Physical Attacks