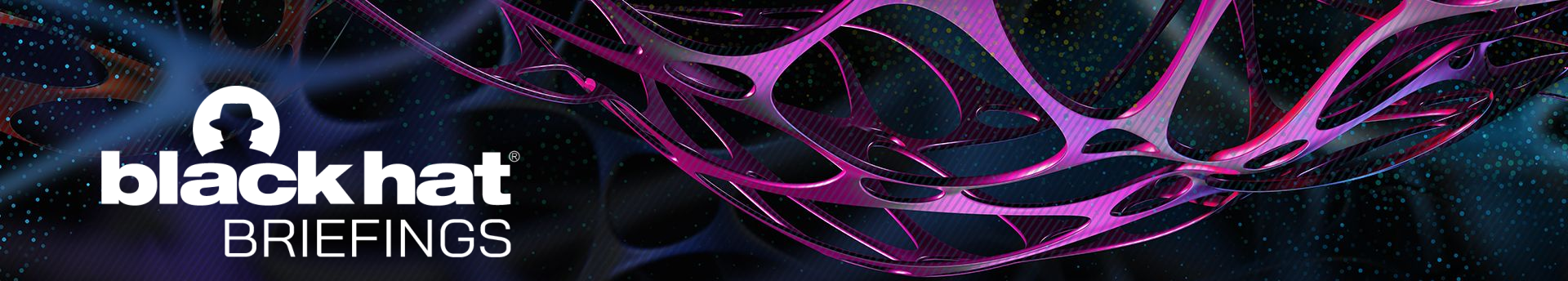


AUGUST 6-7, 2025

MANDALAY BAY / LAS VEGAS

Branch Privilege Injection

Leaking memory on any Intel processor with a microarchitectural race condition

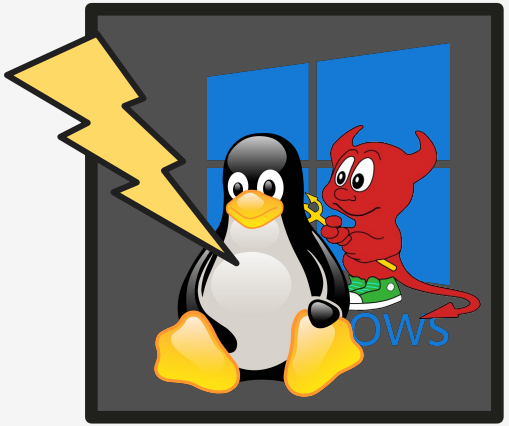
Sandro
PhD student

Kaveh
Professor

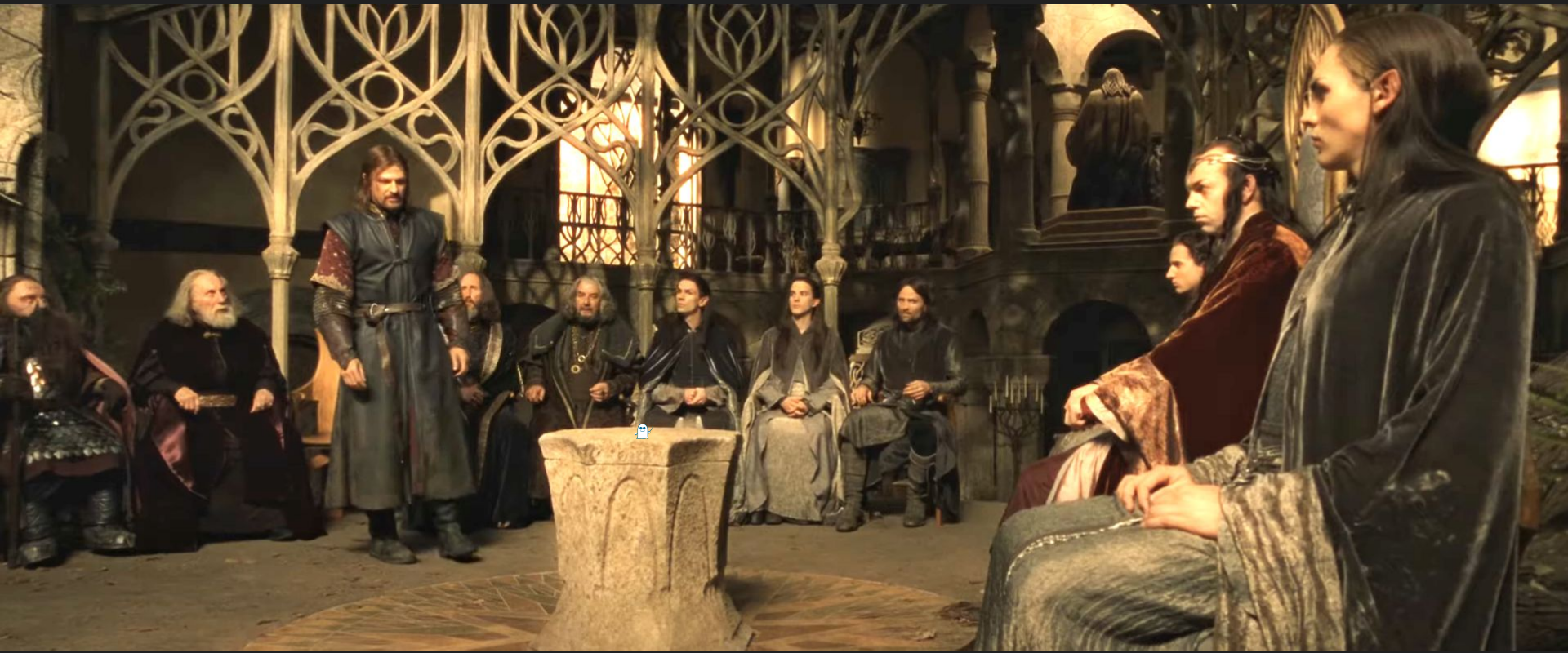
Johannes
Former PhD Student
Speculative execution vulnerabilities.

ETH zürich | COMSEC

back in 2017…


OWS

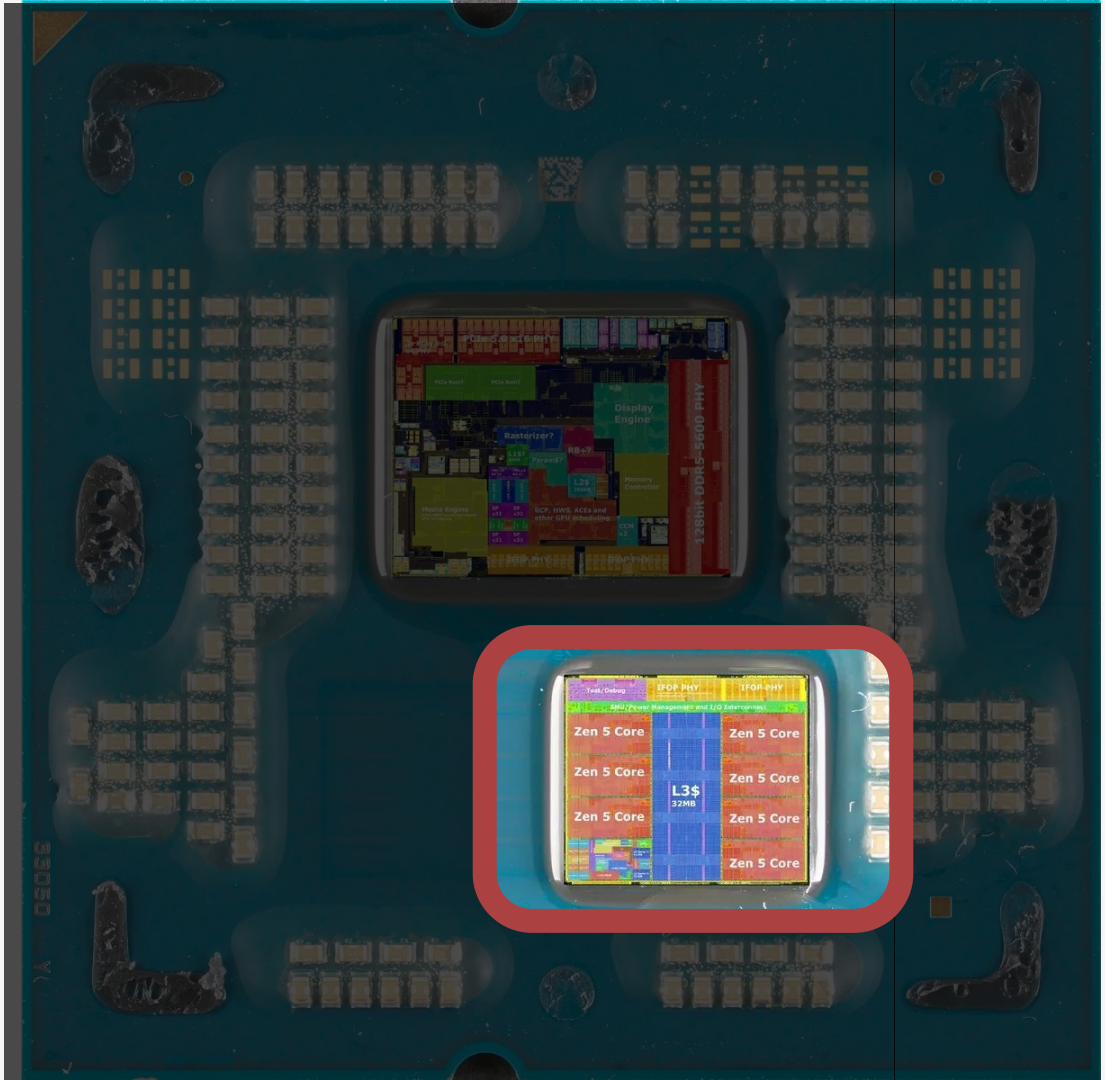

KVM
vmware



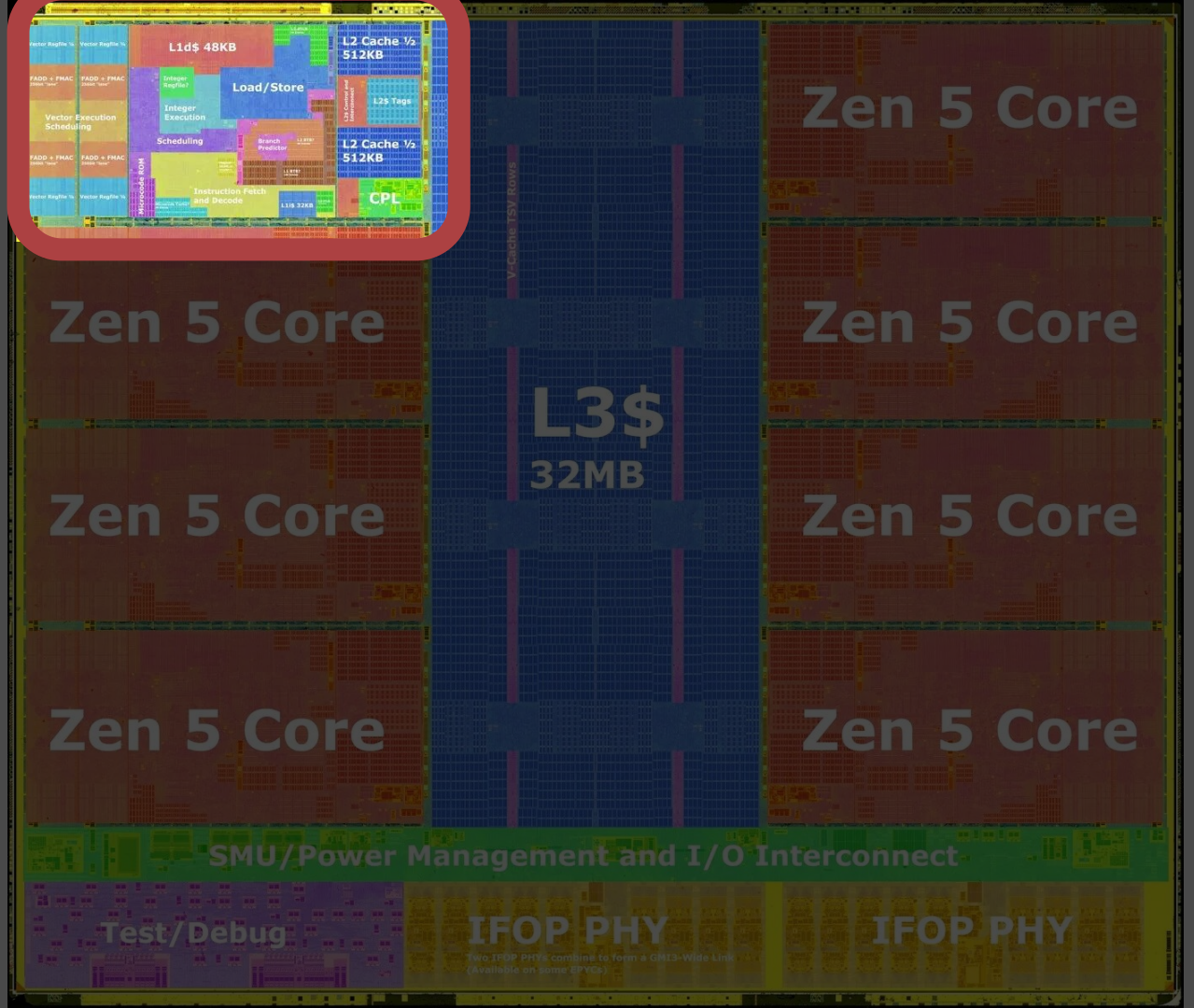

Xen Project
NATIONAL SECURITY AGENCY
UNITED STATES OF AMERICA
AMD
arm
amazon
intel
Microsoft

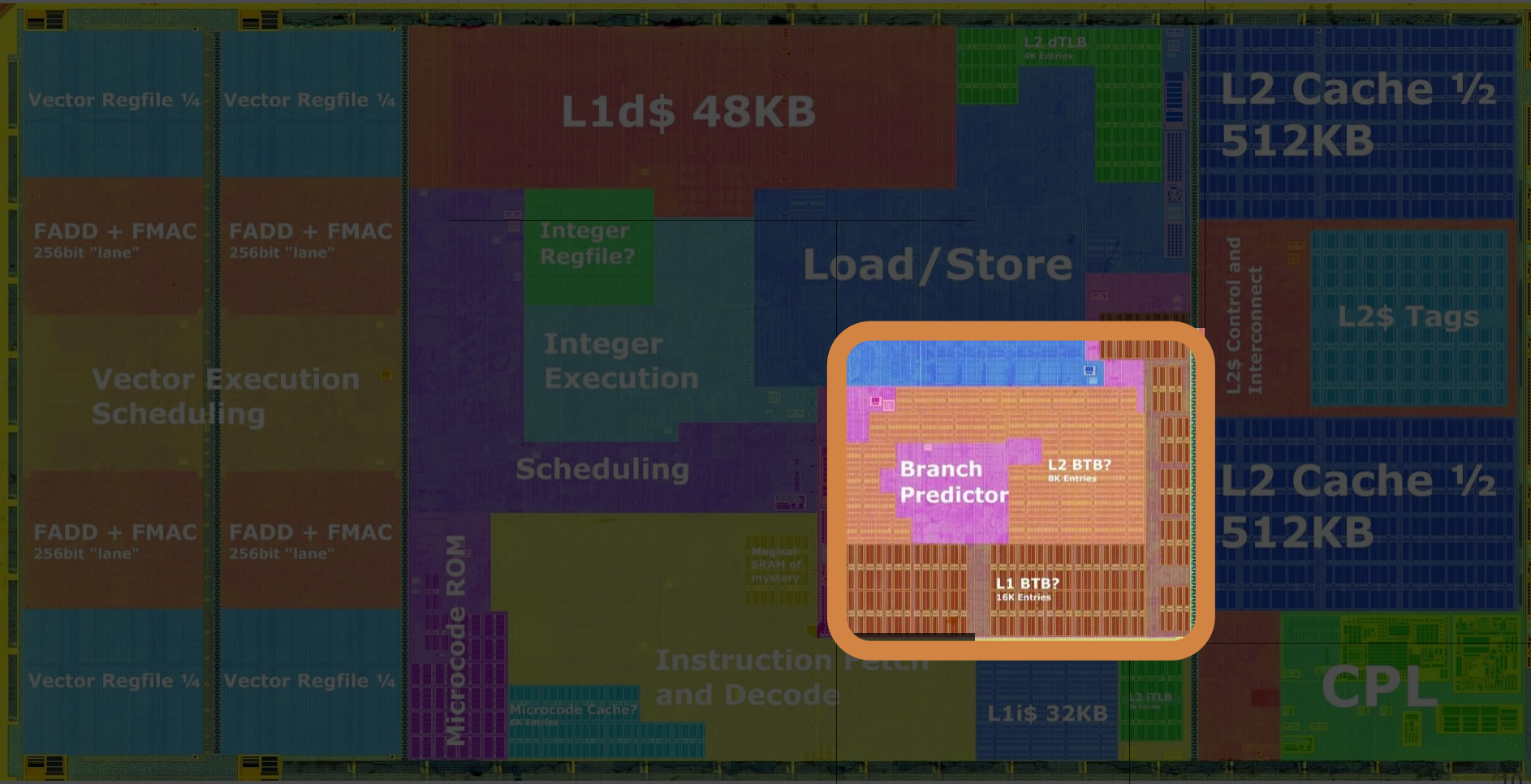

Xen Project
NATIONAL SECURITY AGENCY
UNITED STATES OF AMERICA
AMD
arm
amazon
intel
Microsoft

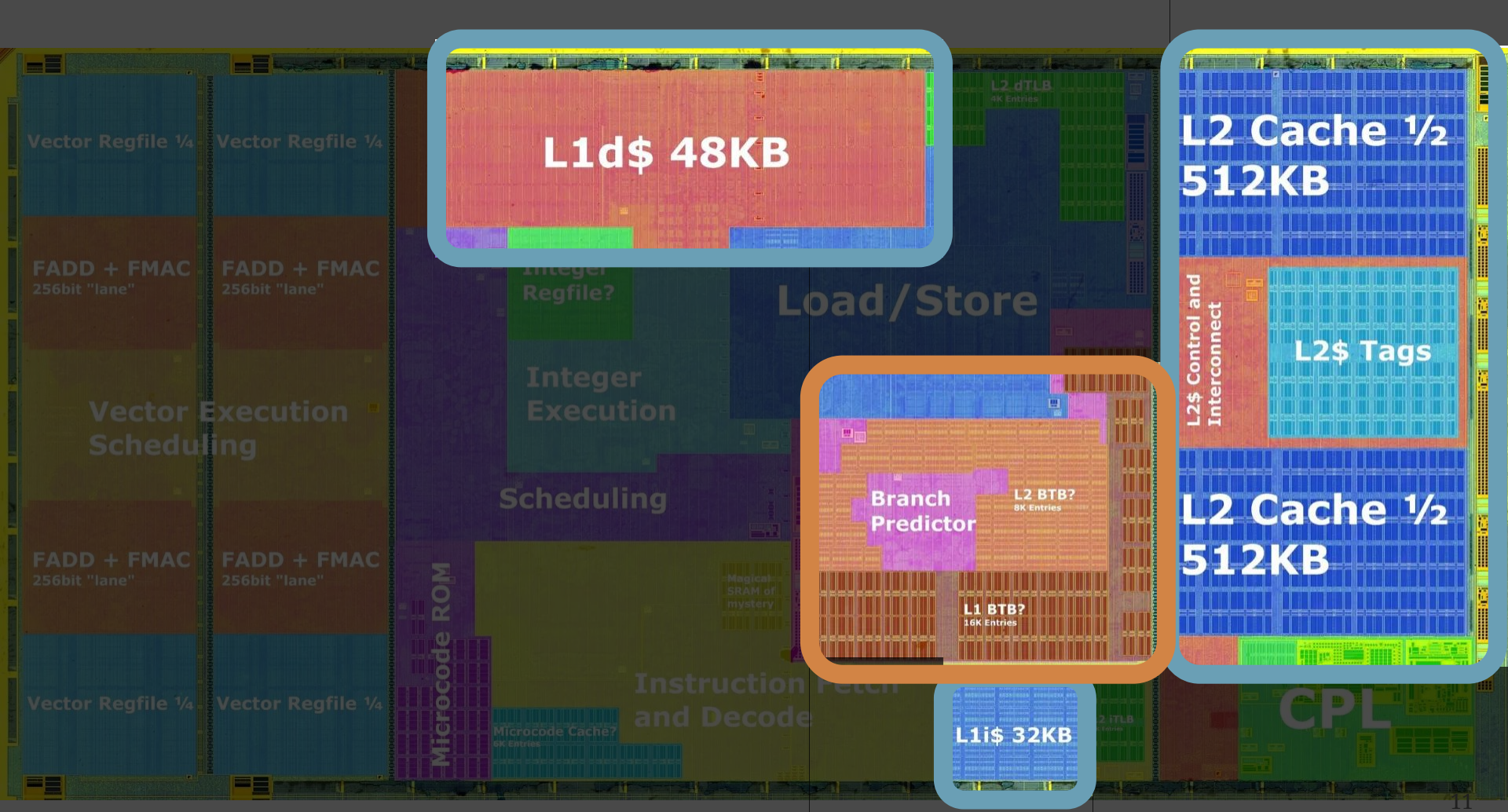
SPECTRE

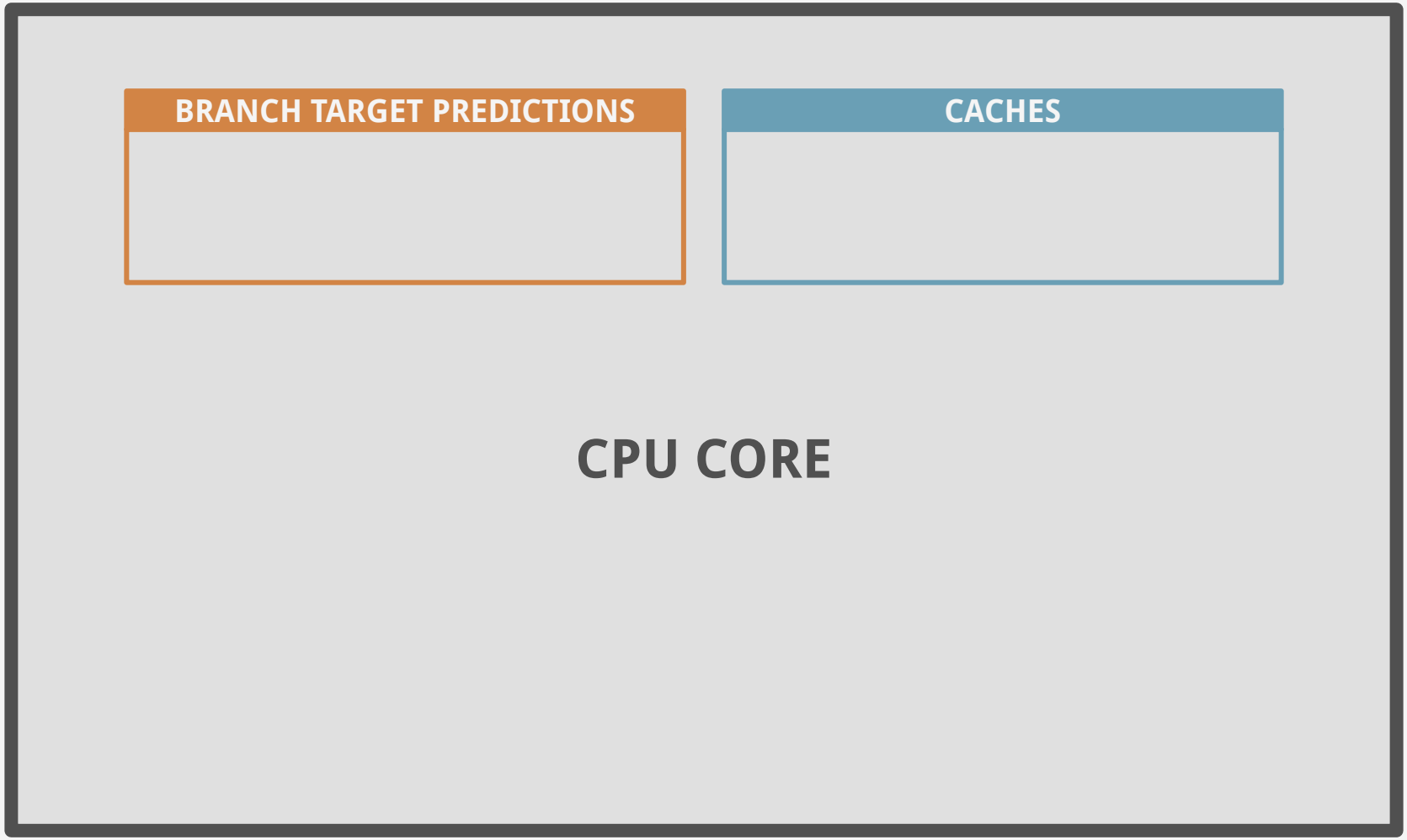


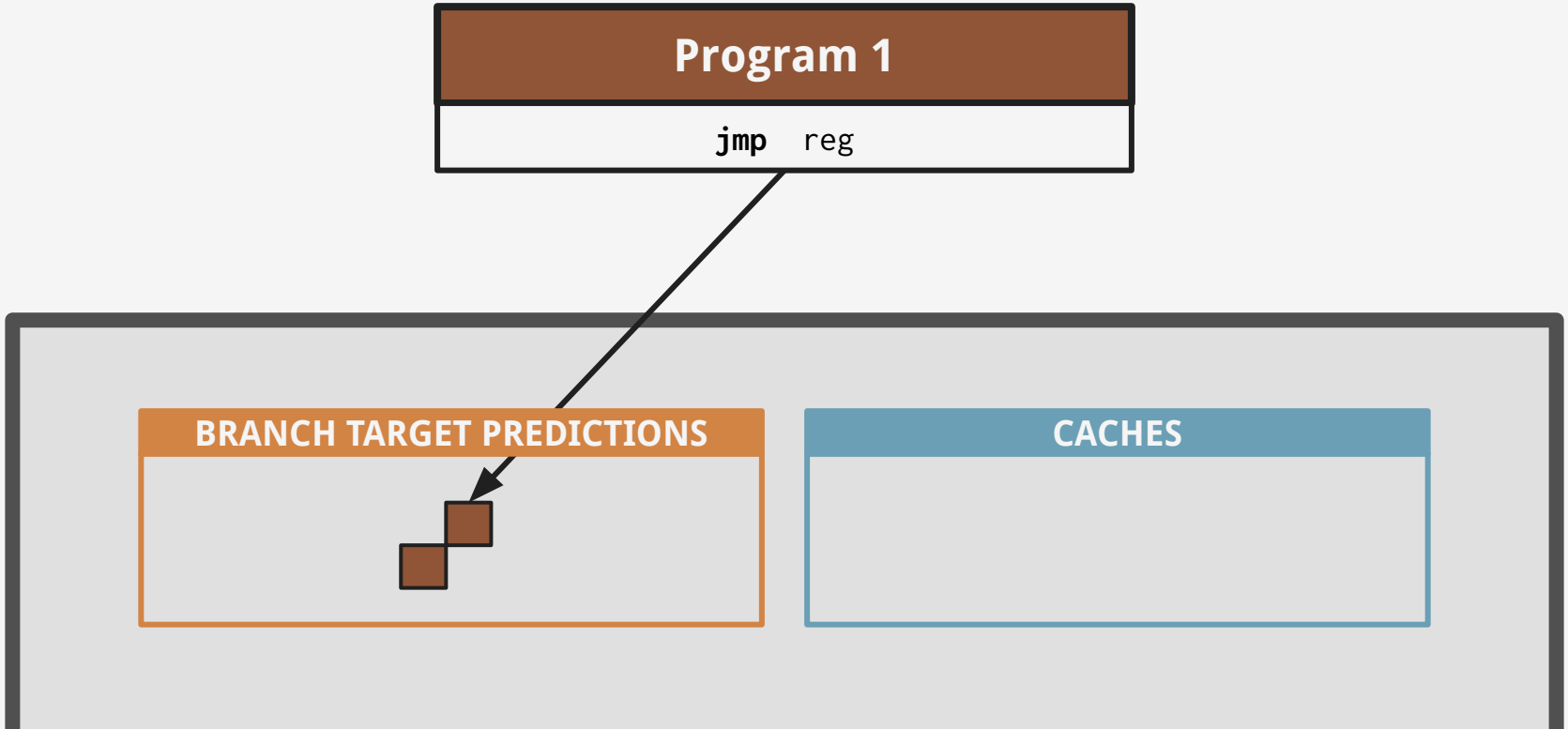
Hilbert Hagedoorn
Oct. 2024
AMD Ryzen 9000 Die Shots gets Annotated In Detail
guru3d.com/story/amd-ryzen-9000-die-shots-gets-annotated-in-detail/

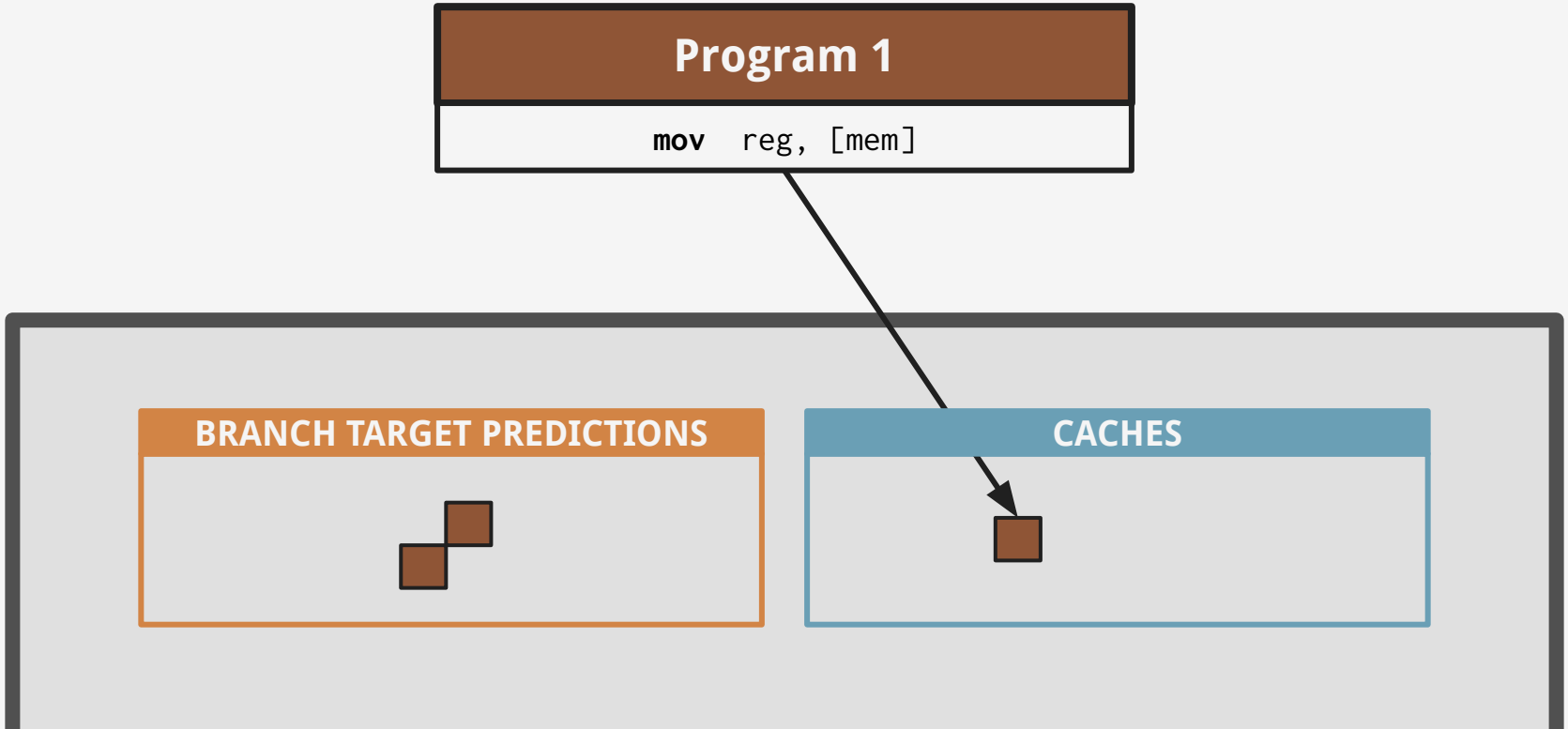

L1d$ 48KB
L2 Cache ½
512KB
Load/Store
Integer Execution
Vector Execution Scheduling
Scheduling
Branch Predictor
L2 Cache ½
512KB
L2$ Tags
CPL
Instruction Fetch and Decode
L1$ 32KB
Zen 5 Core
Zen 5 Core
Zen 5 Core
Zen 5 Core
Zen 5 Core
Zen 5 Core
L3$
32MB
SMU/Power Management and I/O Interconnect
Test/Debug
IFOP PHY
IFOP PHY

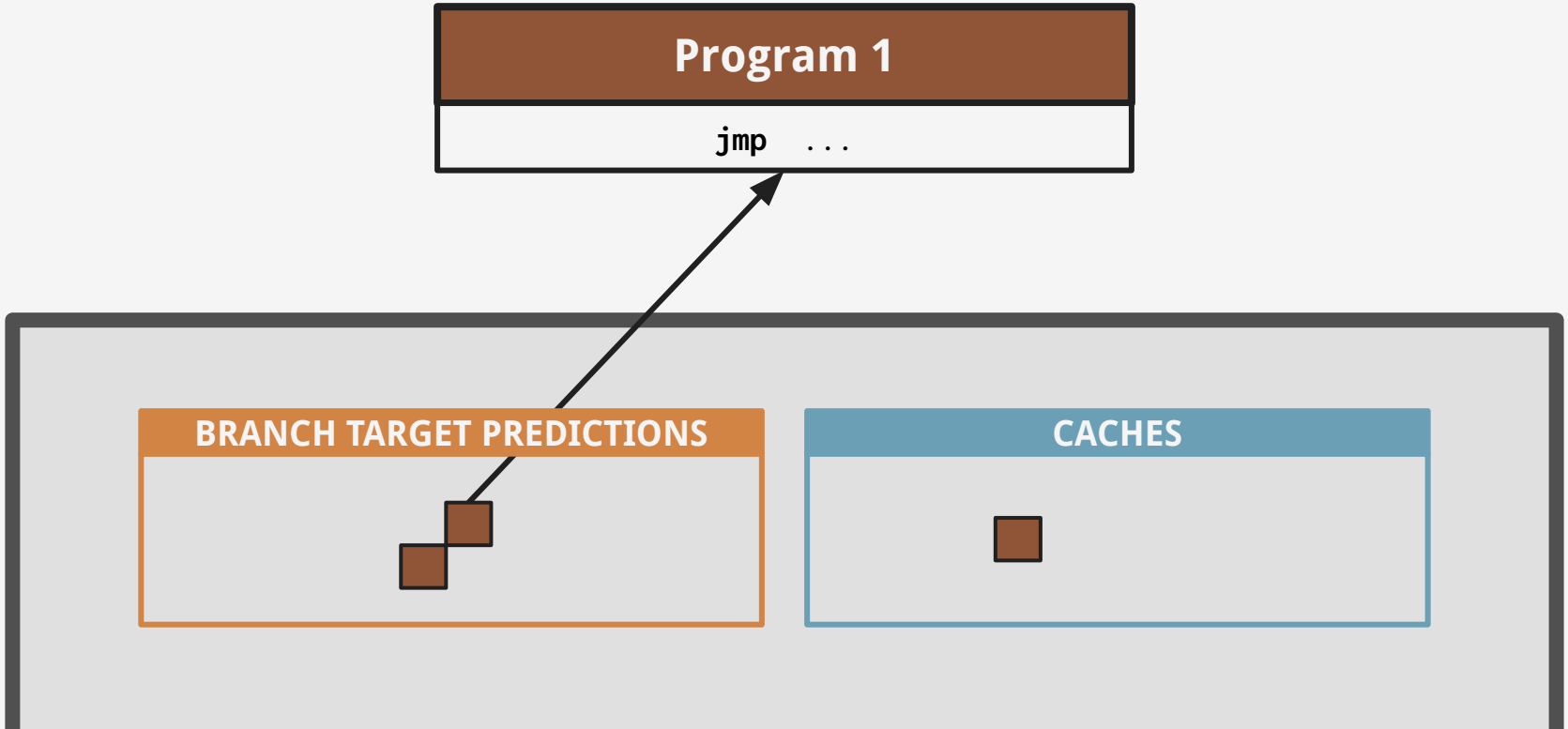

Vector Regfile ¼
Vector Regfile ¼
FADD + FMAC
256bit "lane"
FADD + FMAC
256bit "lane"
Vector Execution
Scheduling
FADD + FMAC
256bit "lane"
FADD + FMAC
256bit "lane"
Vector Regfile ¼
Vector Regfile ¼
L1d$ 48KB
L2 dTLB
4K Entries
L2 Cache ½
512KB
Integer
Regfile?
Load/Store
L2$ Control and
Interconnect
L2$ Tags
Integer
Execution
Scheduling
Branch
Predictor
L2 BTB?
8K Entries
L1 BTB?
16K Entries
L2 Cache ½
512KB
Magical
SRAM of
mystery
Microcode ROM
Instruction Fetch
and Decode
Microcode Cache?
6K Entries
L1i$ 32KB
L2 iTLB
CPL


L1d$ 48KB
L2 Cache ½
512KB
L2$ Control and Interconnect
L2$ Tags
L2 Cache ½
512KB
Branch Predictor
L2 BTB?
8K Entries
L1 BTB?
16K Entries
L1i$ 32KB
Vector Regfile ¼
FADD + FMAC
256bit "lane"
Vector Execution Scheduling
Load/Store
Integer Execution
Scheduling
Microcode ROM
Instruction Fetch and Decode
CPL


BRANCH TARGET PREDICTIONS
CACHES
CPU CORE

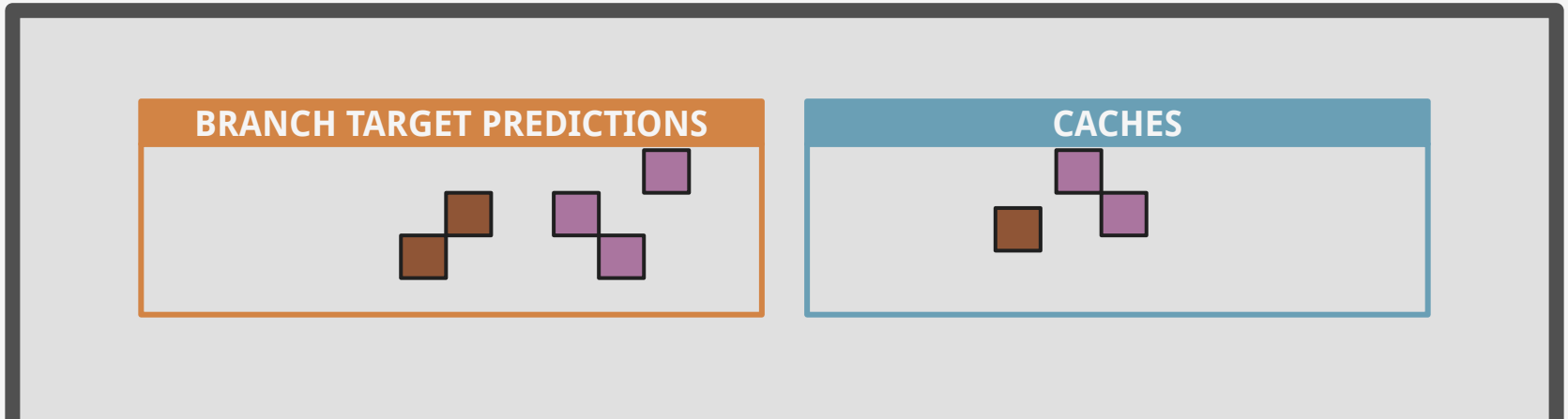

Program 1
jmp reg
BRANCH TARGET PREDICTIONS
CACHES


Program 1
mov reg, [mem]
BRANCH TARGET PREDICTIONS
CACHES


Program 1
jmp ...
BRANCH TARGET PREDICTIONS
CACHES

Program 2

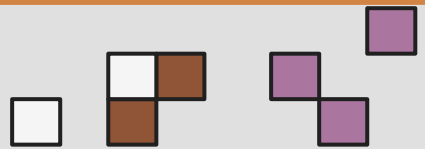
```
jmp… mov… mov… jmp… etc
```



OS/kernel (ring 0)

`jmp… mov… mov… jmp… etc`

BRANCH TARGET PREDICTIONS

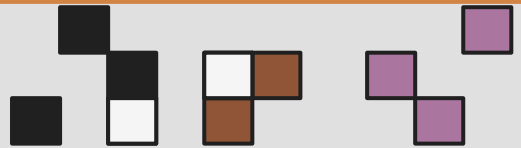


CACHES

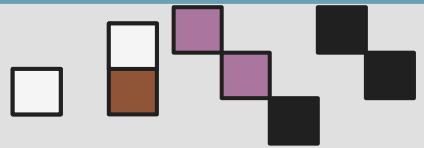
VMM (“ring -1”)

```
jmp… mov… mov… jmp… etc
```

BRANCH TARGET PREDICTIONS

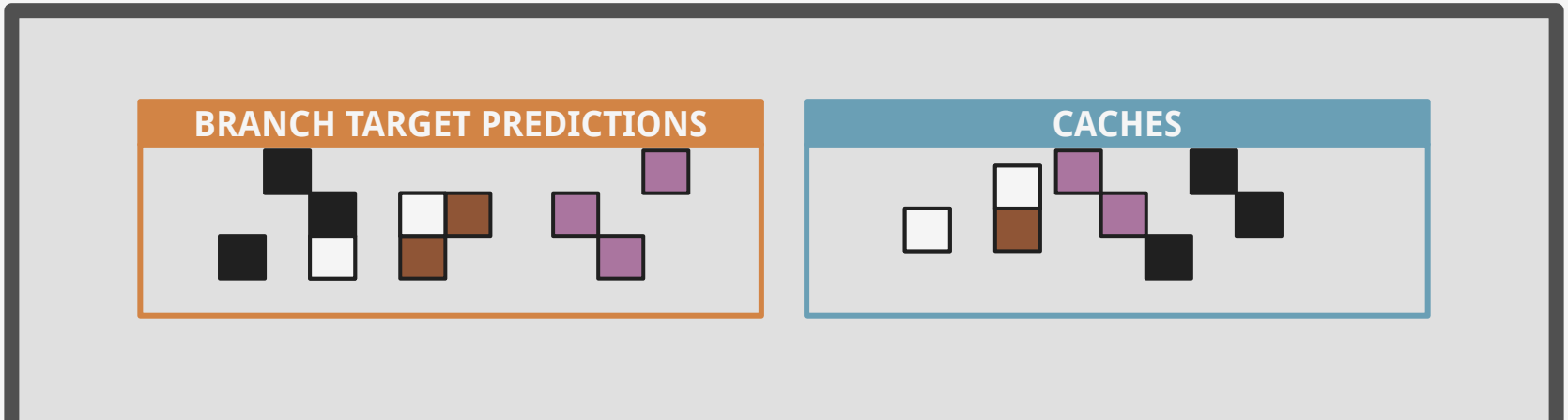


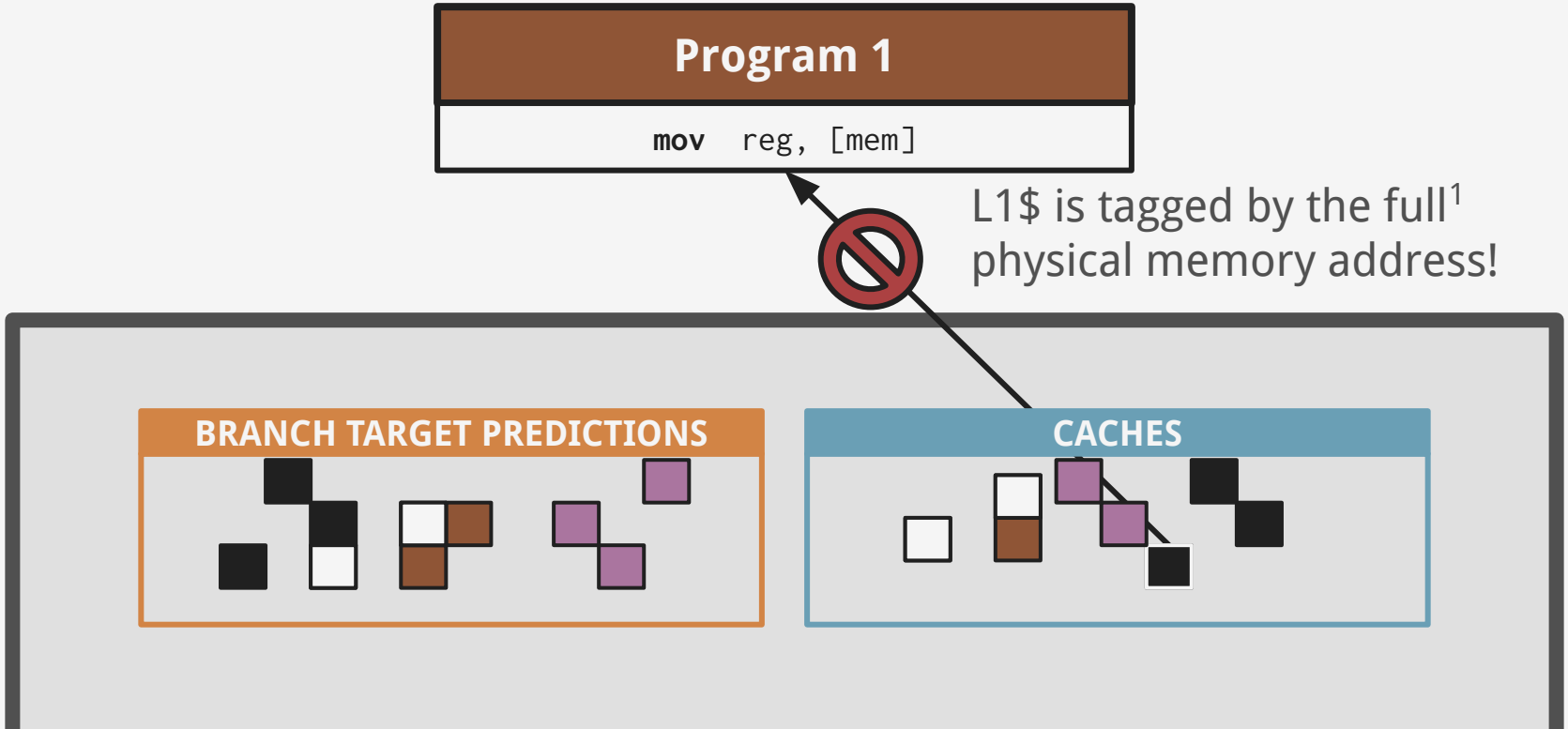
CACHES



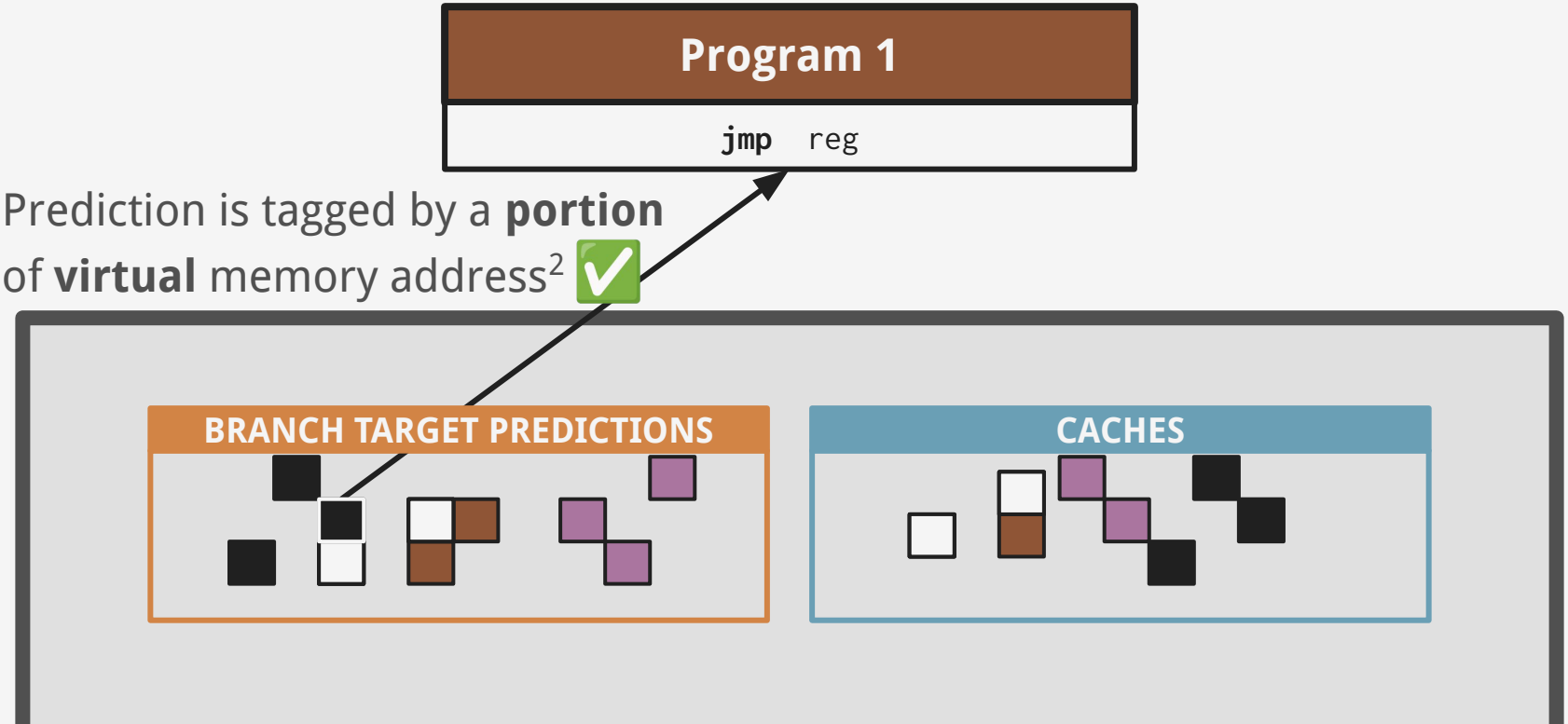
Program 1

```
mov  reg, [mem]
```

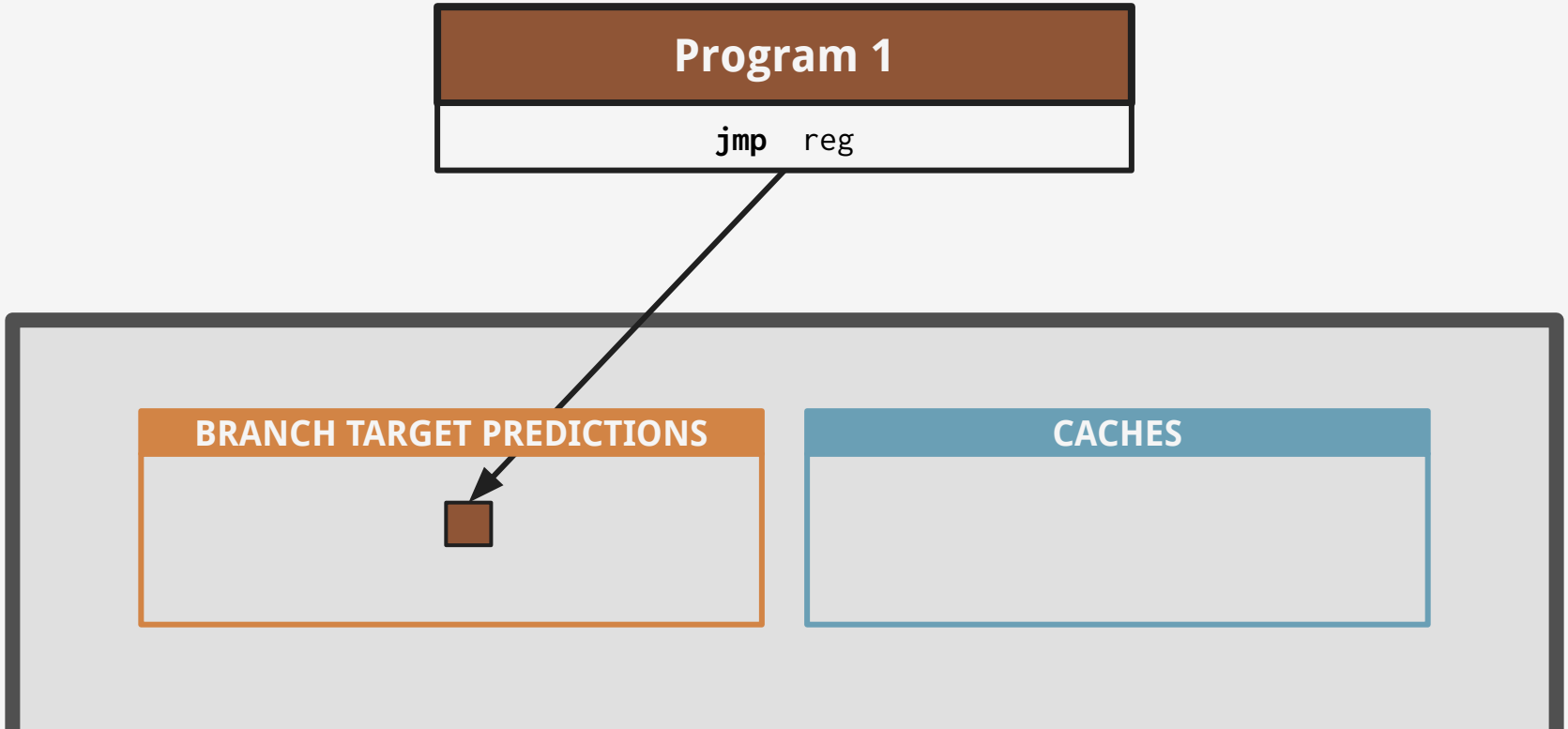



Program 1
mov reg, [mem]
L1$ is tagged by the full[1] physical memory address!
BRANCH TARGET PREDICTIONS
CACHES

So what? It's a design choice.

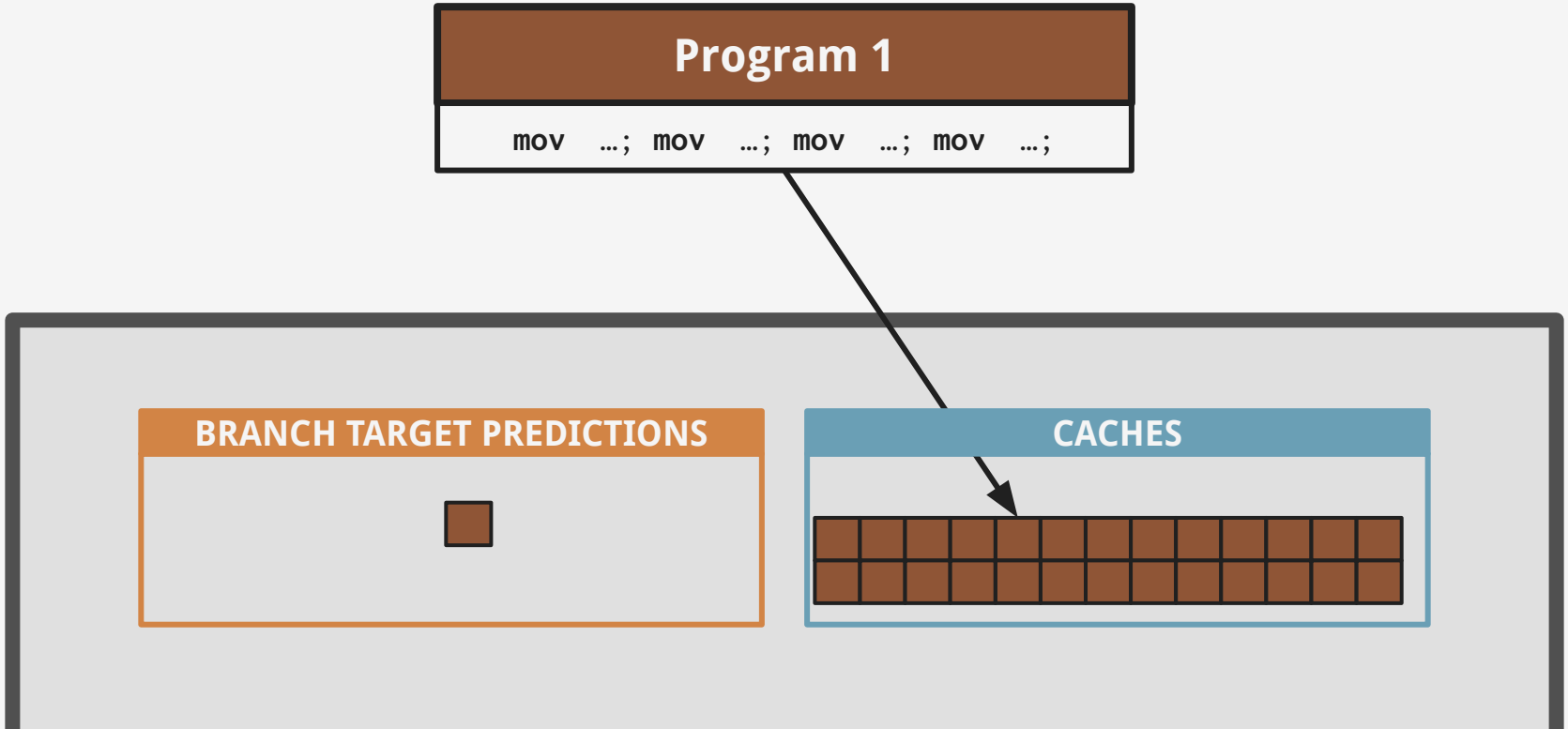


Inject prediction



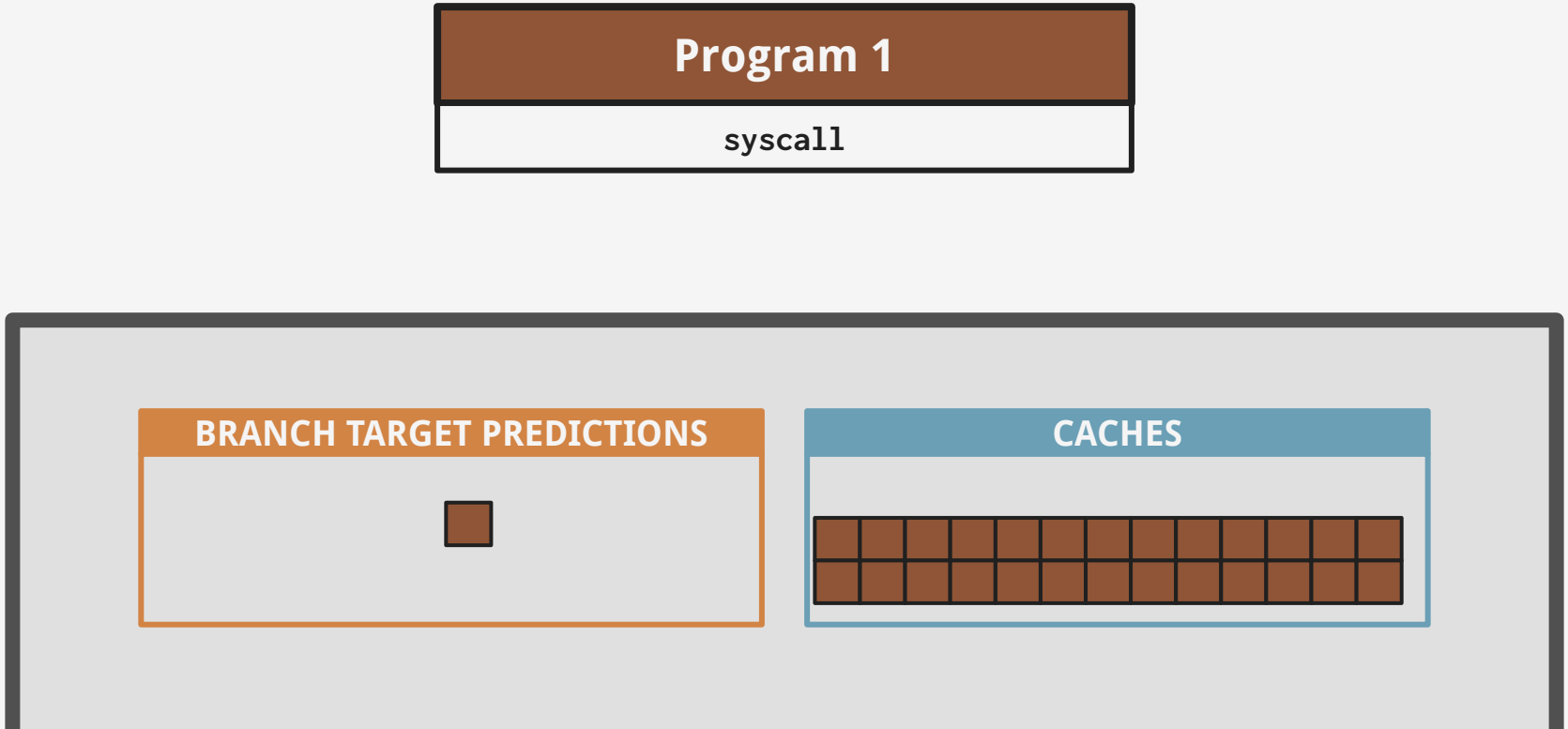
Inject prediction

Prime side channel



Inject prediction

Prime side channel



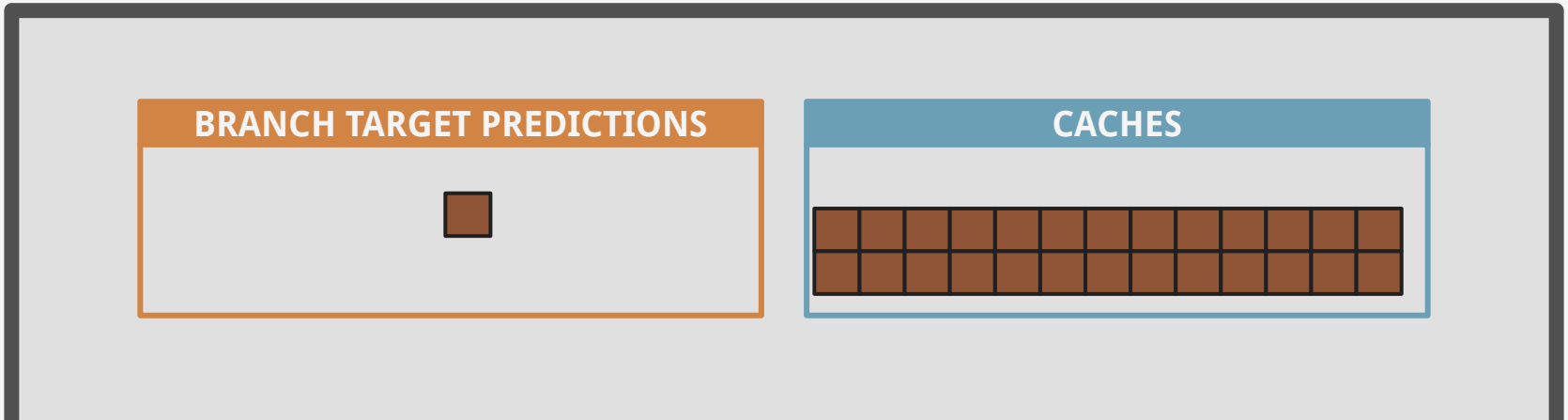
Inject prediction

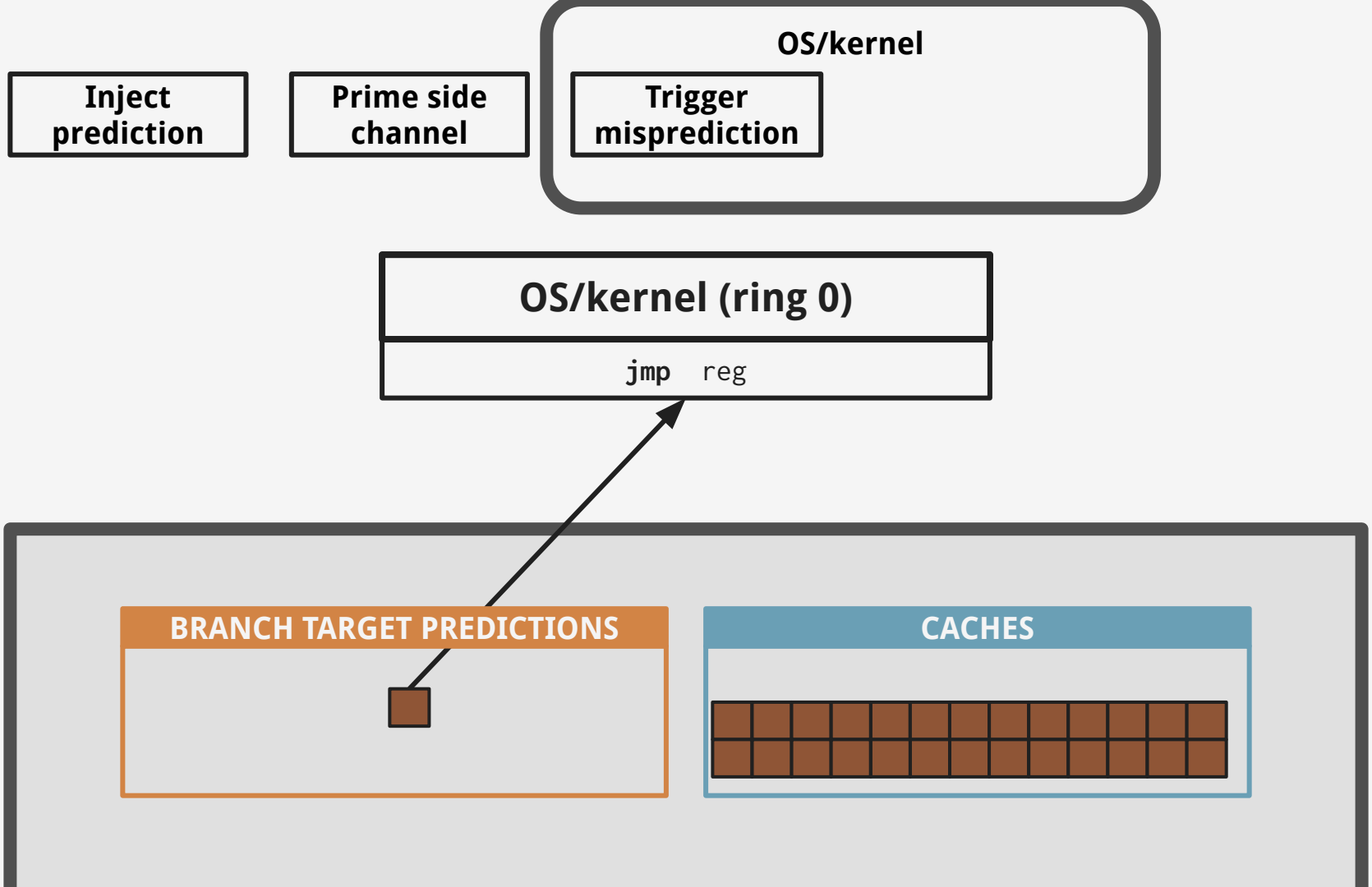
Prime side channel

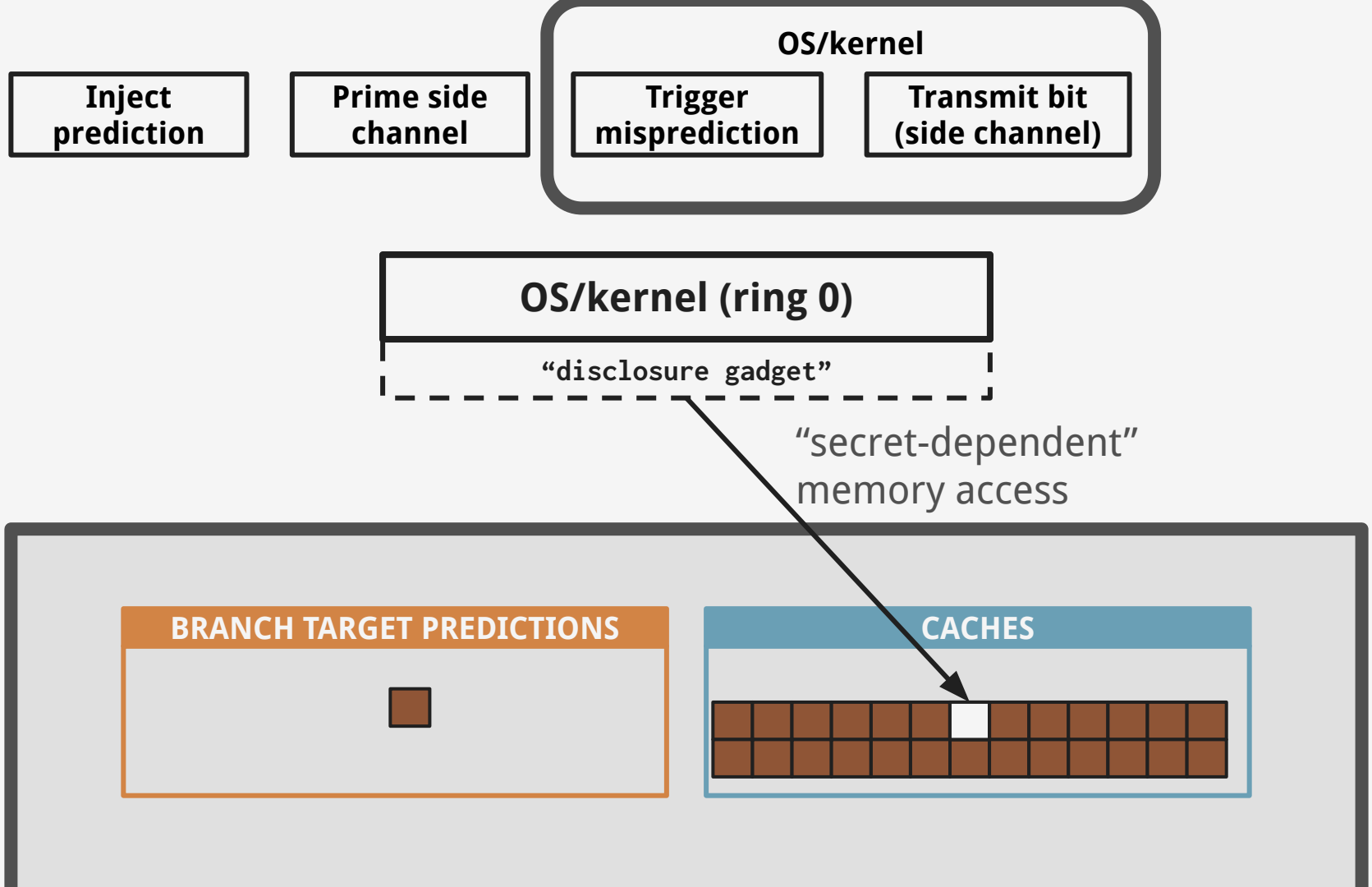
OS/kernel

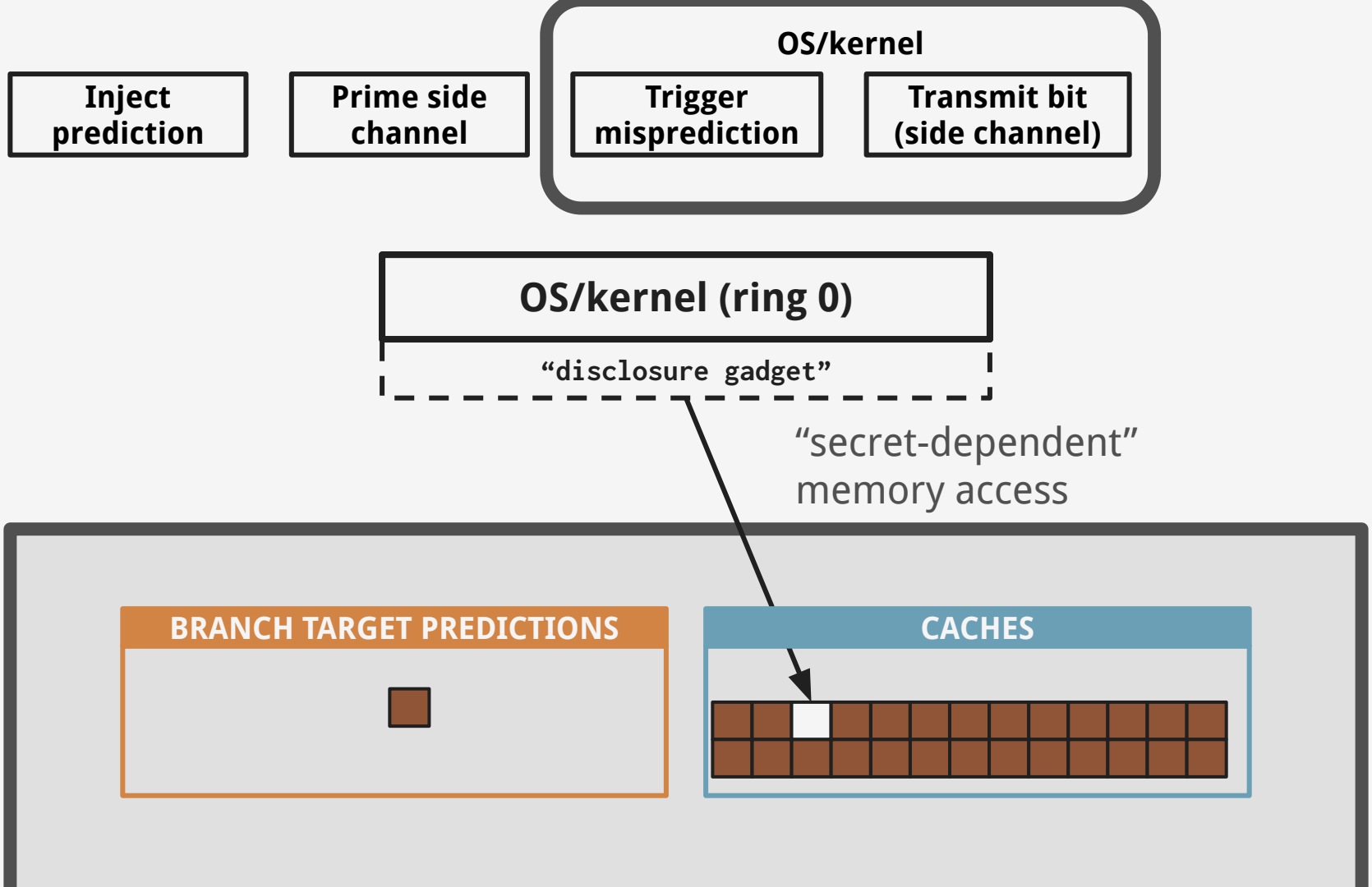
OS/kernel (ring 0)

```
jmp  reg
```




OS/kernel
Inject prediction
Prime side channel
Trigger misprediction
OS/kernel (ring 0)
jmp reg
BRANCH TARGET PREDICTIONS
CACHES


Inject prediction
Prime side channel
OS/kernel
Trigger misprediction
Transmit bit (side channel)
OS/kernel (ring 0)
“disclosure gadget”
“secret-dependent” memory access
BRANCH TARGET PREDICTIONS
CACHES


Inject prediction
Prime side channel
OS/kernel
Trigger misprediction
Transmit bit (side channel)
OS/kernel (ring 0)
“disclosure gadget”
“secret-dependent” memory access
BRANCH TARGET PREDICTIONS
CACHES

Inject prediction

Prime side channel

OS/kernel

Trigger misprediction

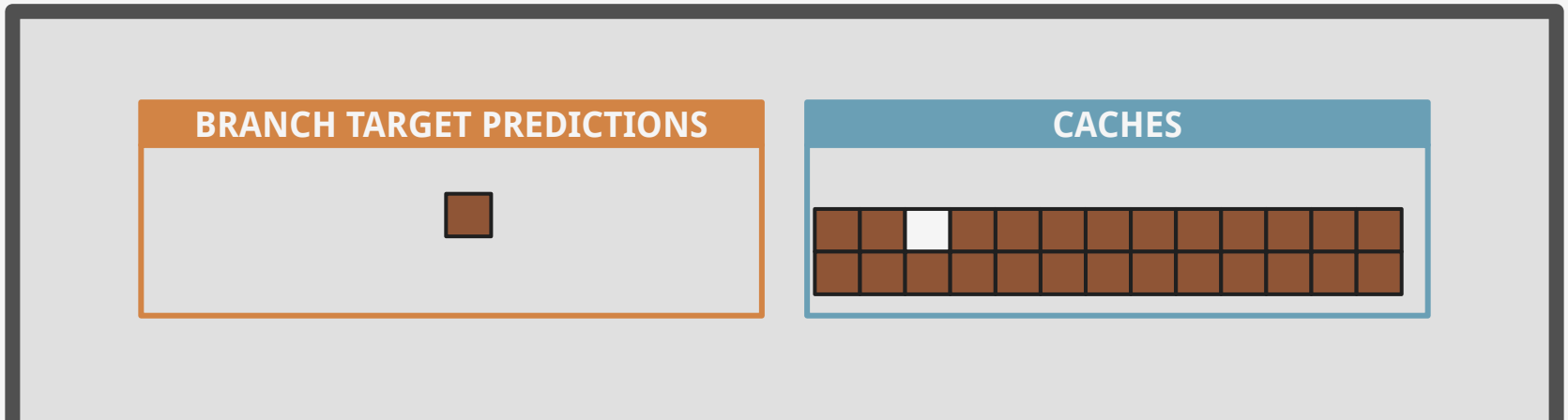
Transmit bit (side channel)

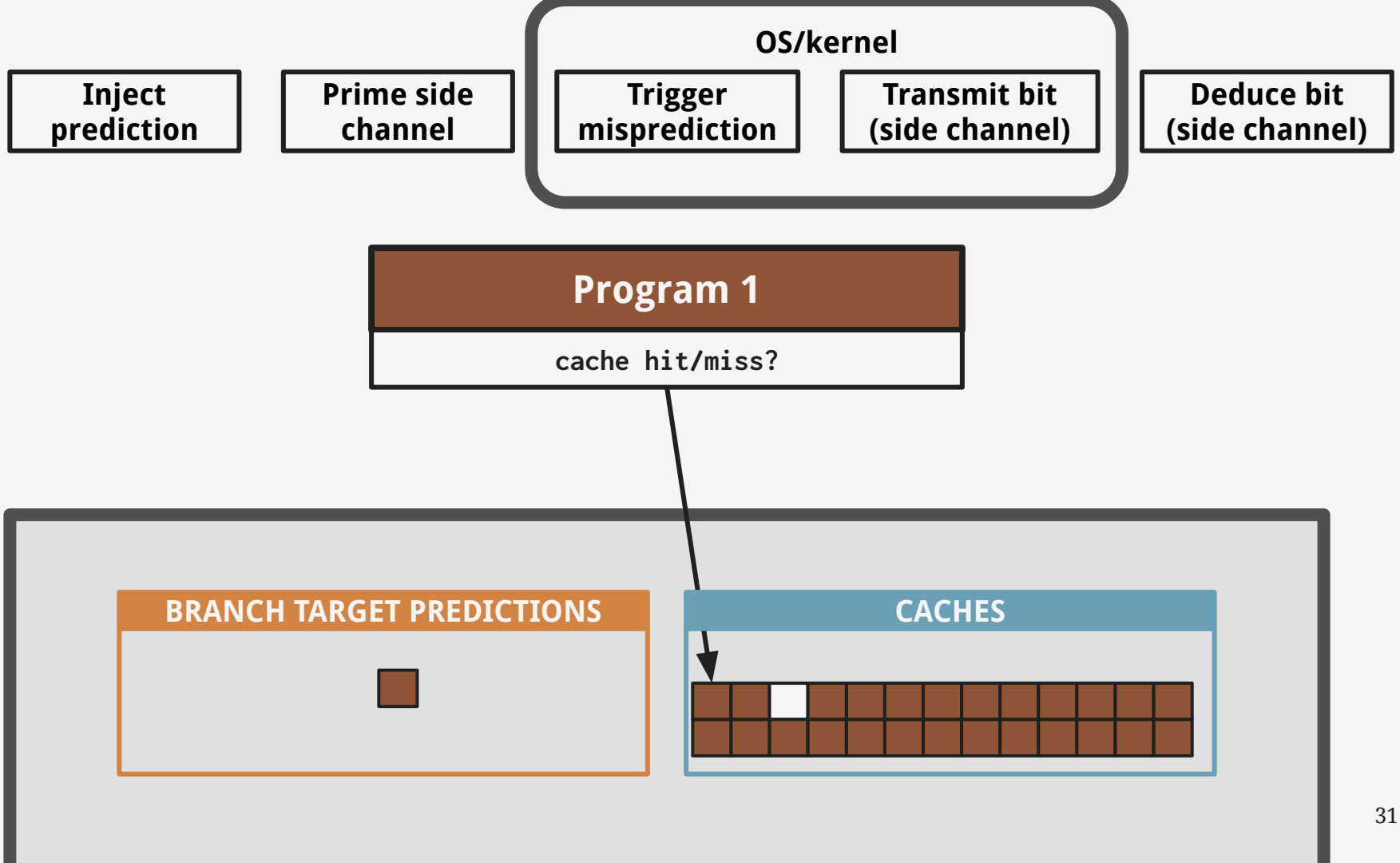
OS/kernel (ring 0)

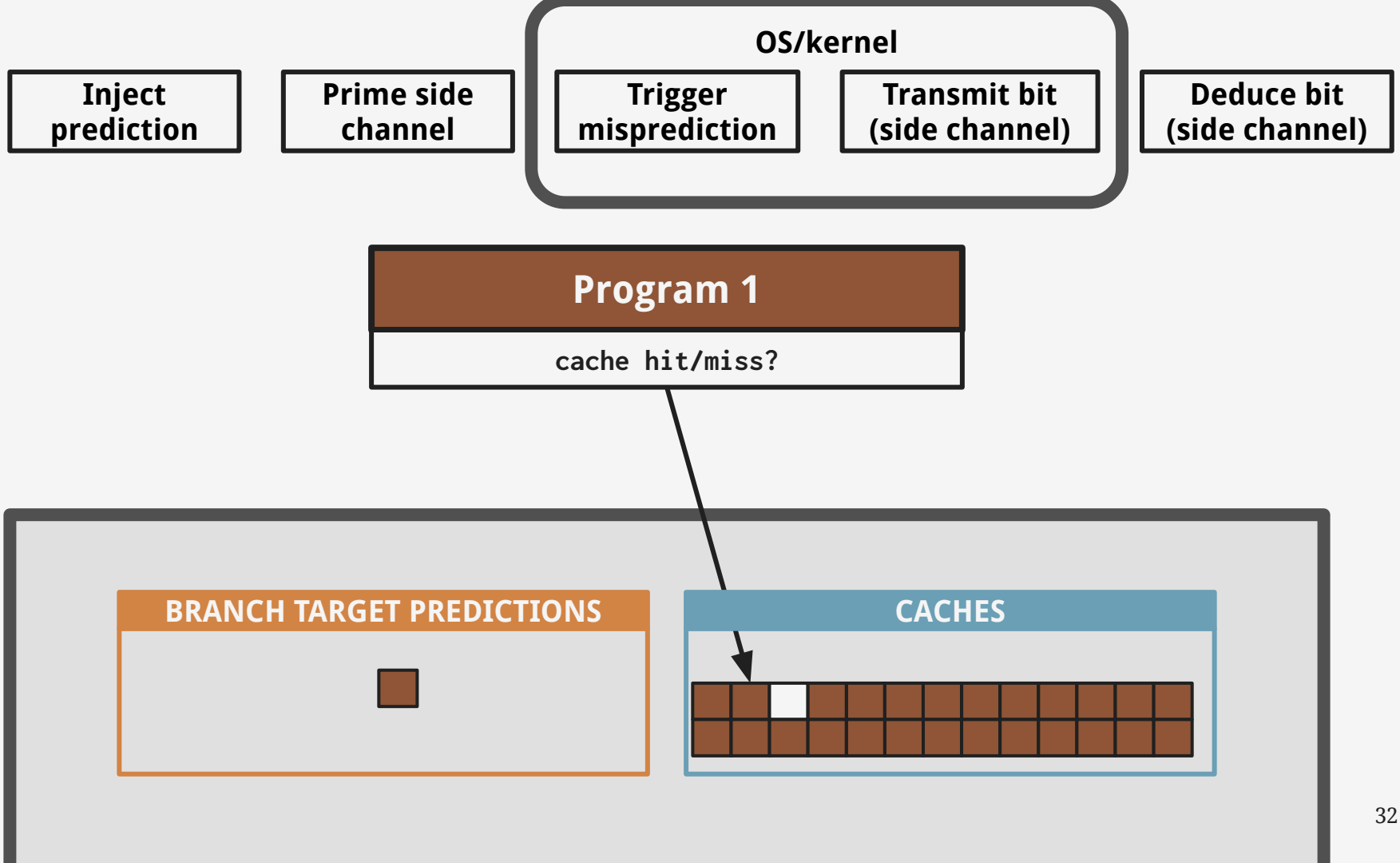
“disclosure gadget”

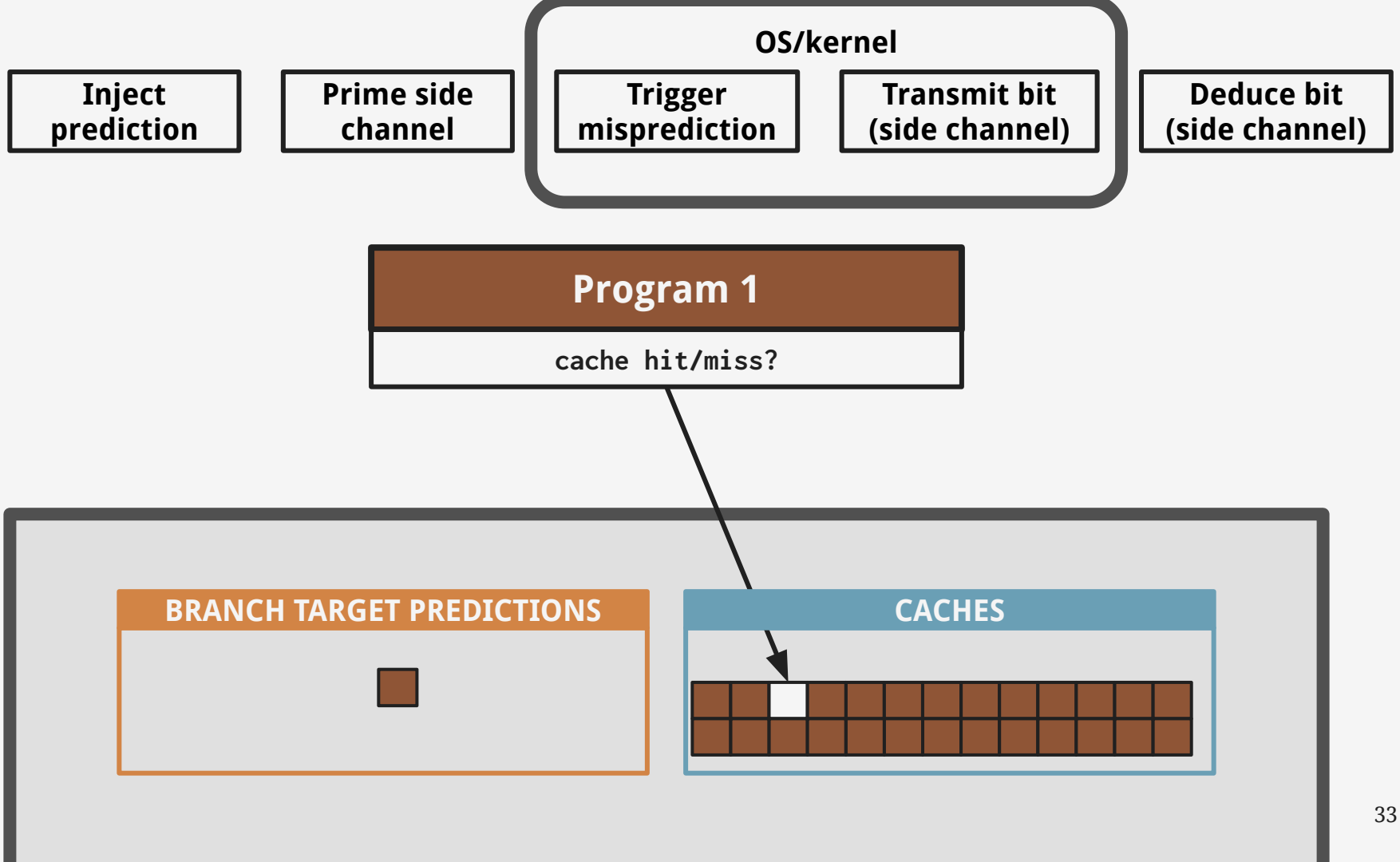
```
movzx   R_sec, byte  ptr [R1]
mov     R_sink, qword ptr [R2 + R_sec*8]
```

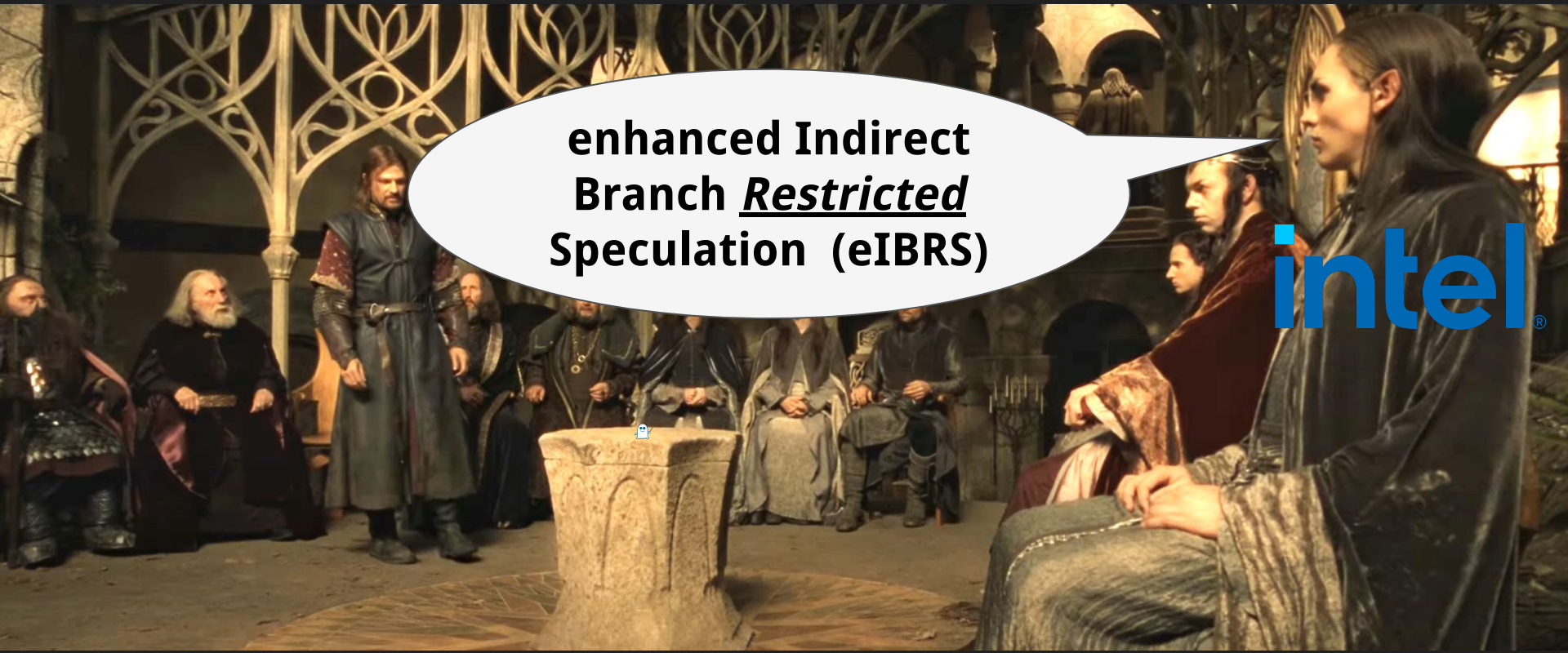
R1: *&secret*
R2: *shared memory*




OS/kernel
Inject prediction
Prime side channel
Trigger misprediction
Transmit bit (side channel)
Deduce bit (side channel)
Program 1
cache hit/miss?
BRANCH TARGET PREDICTIONS
CACHES


Inject prediction
Prime side channel
OS/kernel
Trigger misprediction
Transmit bit (side channel)
Deduce bit (side channel)
Program 1
cache hit/miss?
BRANCH TARGET PREDICTIONS
CACHES


Inject prediction
Prime side channel
OS/kernel
Trigger misprediction
Transmit bit (side channel)
Deduce bit (side channel)
Program 1
cache hit/miss?
BRANCH TARGET PREDICTIONS
CACHES


enhanced Indirect
Branch *Restricted*
Speculation (eIBRS)
intel

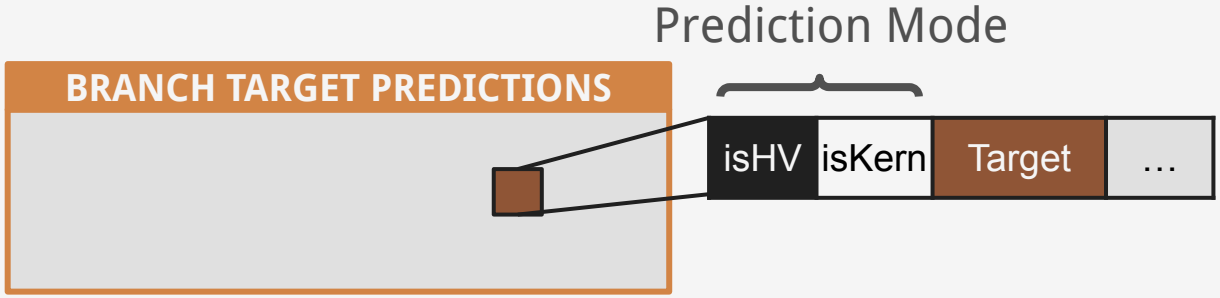

Love the vibe in the room rn
Great meeting everyone
Great job team
Can’t wait for the kick-off

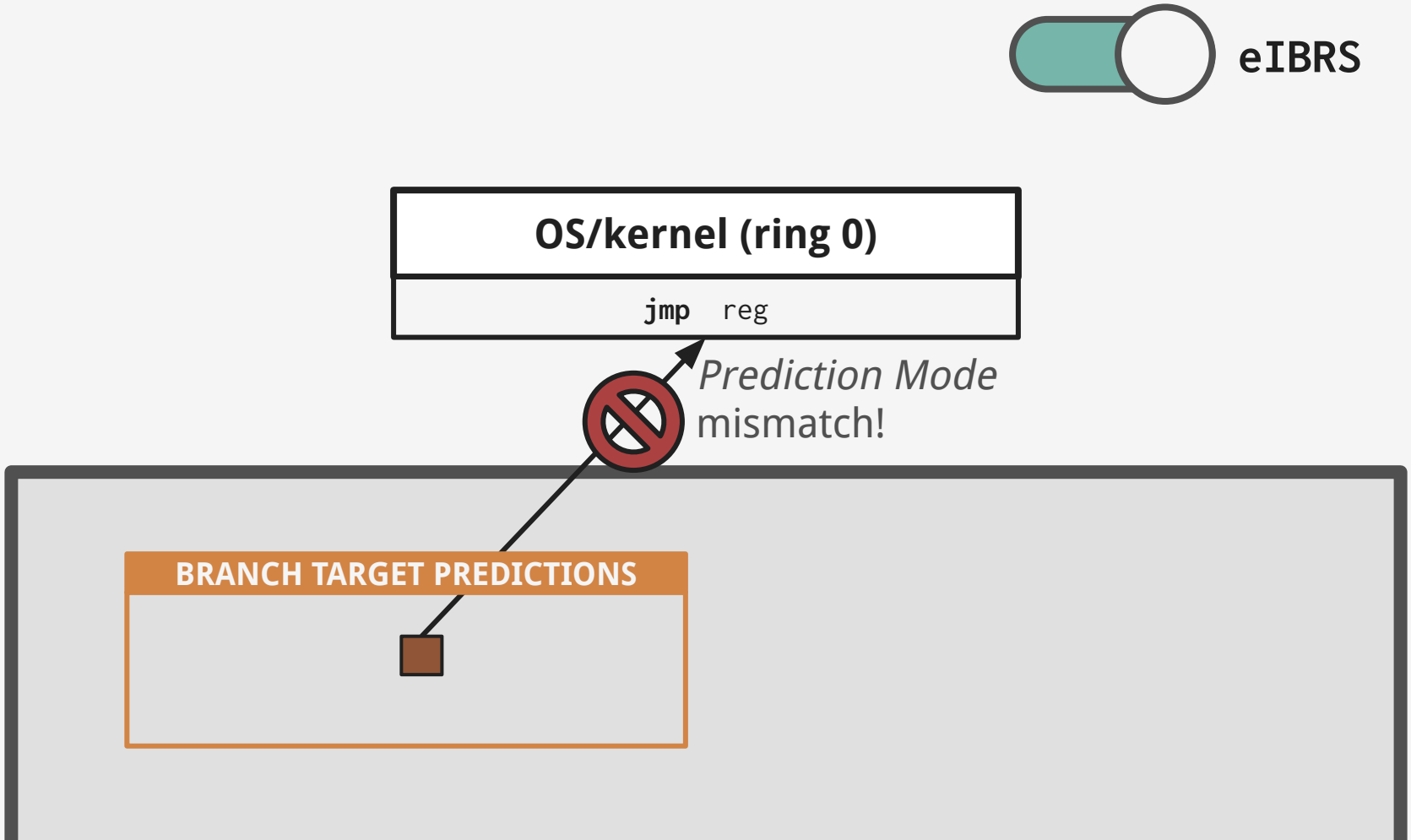
eIBRS

- `asm("wrmsr" :: "c"(SPEC_CTRL), "a"(1), "d"(0));`

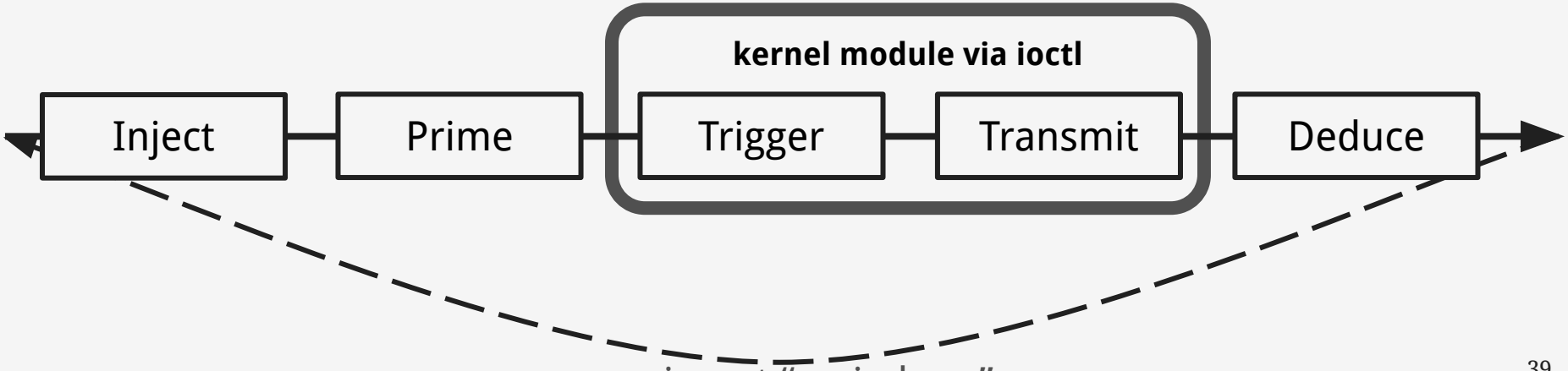
eIBRS

- `asm("wrmsr" :: "c"(SPEC_CTRL), "a"(1), "d"(0));`
- Stays on forever

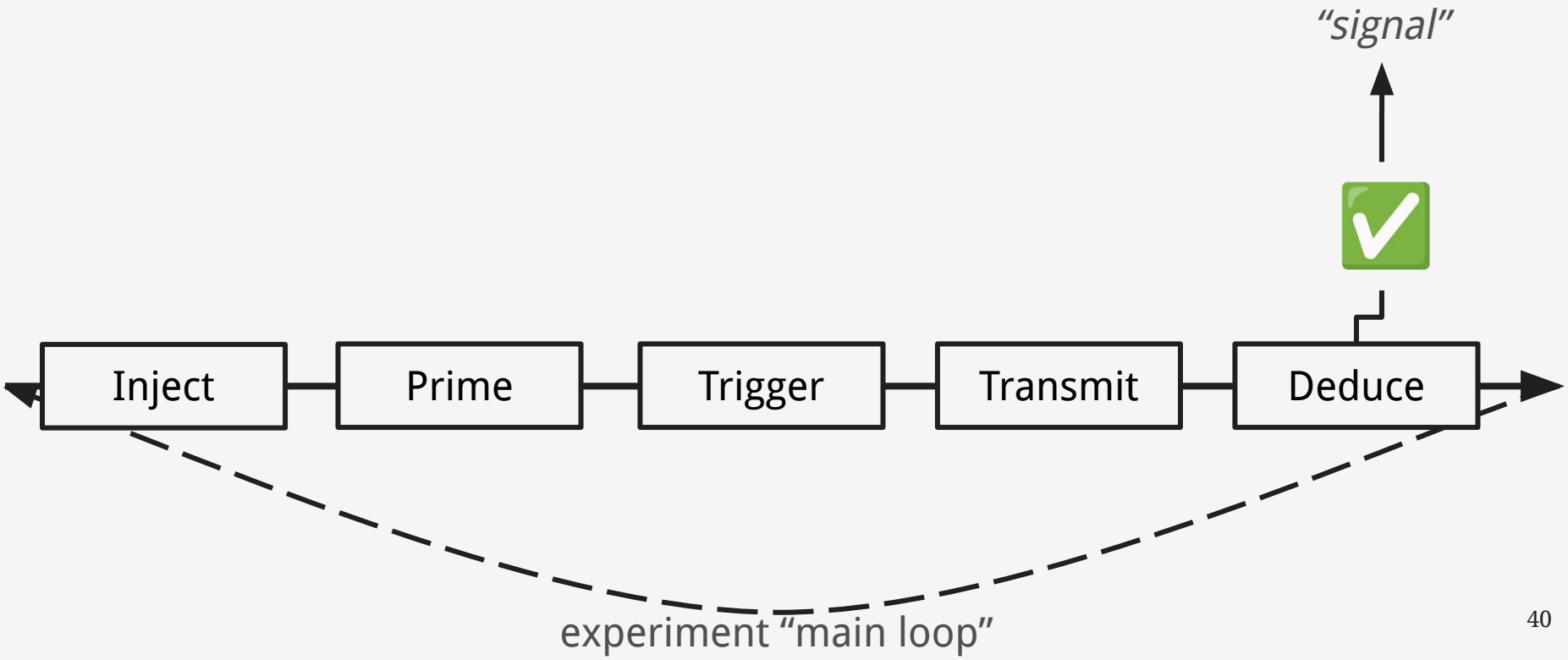



eIBRS
OS/kernel (ring 0)
jmp reg
Prediction Mode
mismatch!
BRANCH TARGET PREDICTIONS

Assessing eIBRS

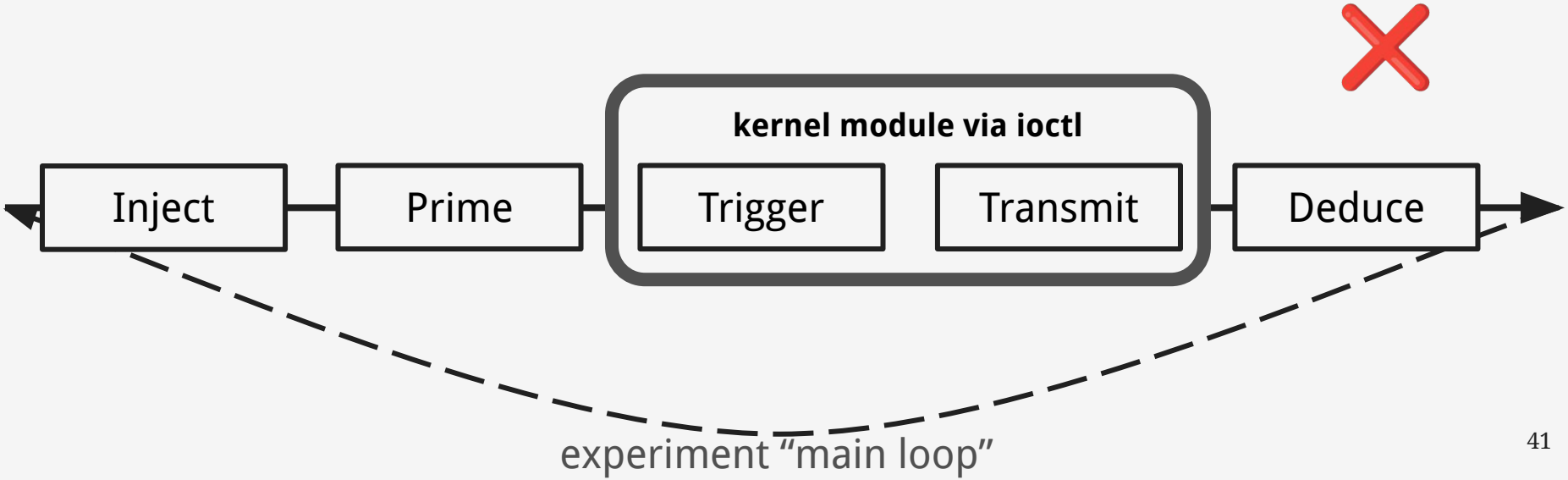


Assessing eIBRS



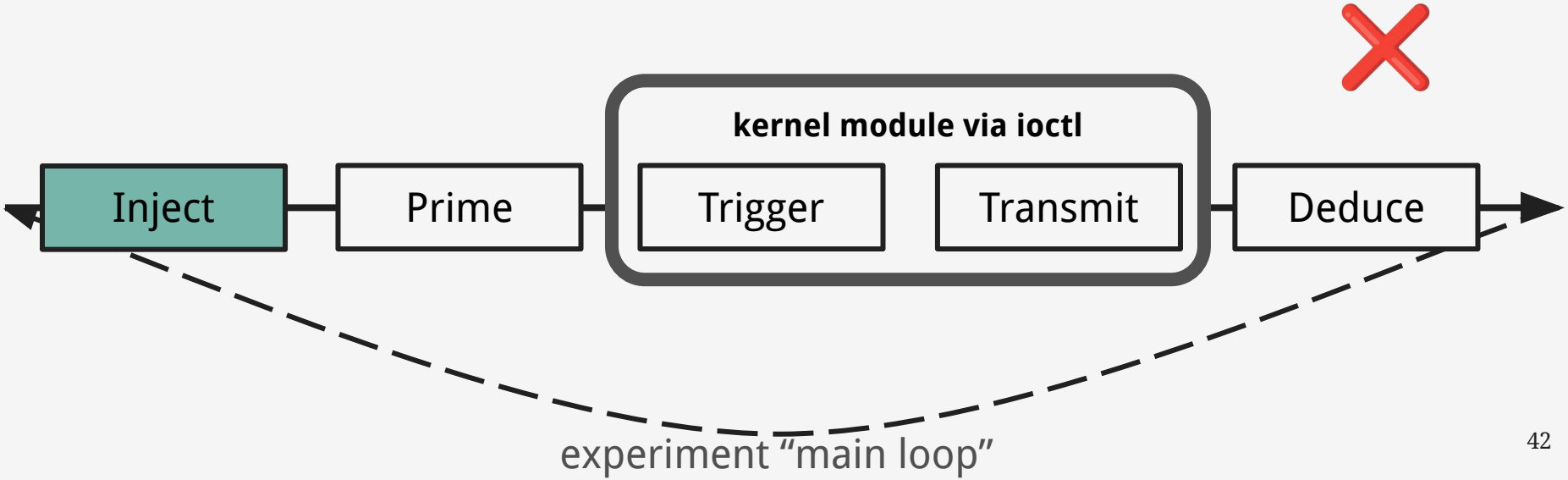
Assessing eIBRS

- Is eIBRS secretly turned on by firmware?
- **Inconclusive:** Does it really fail because of eIBRS?



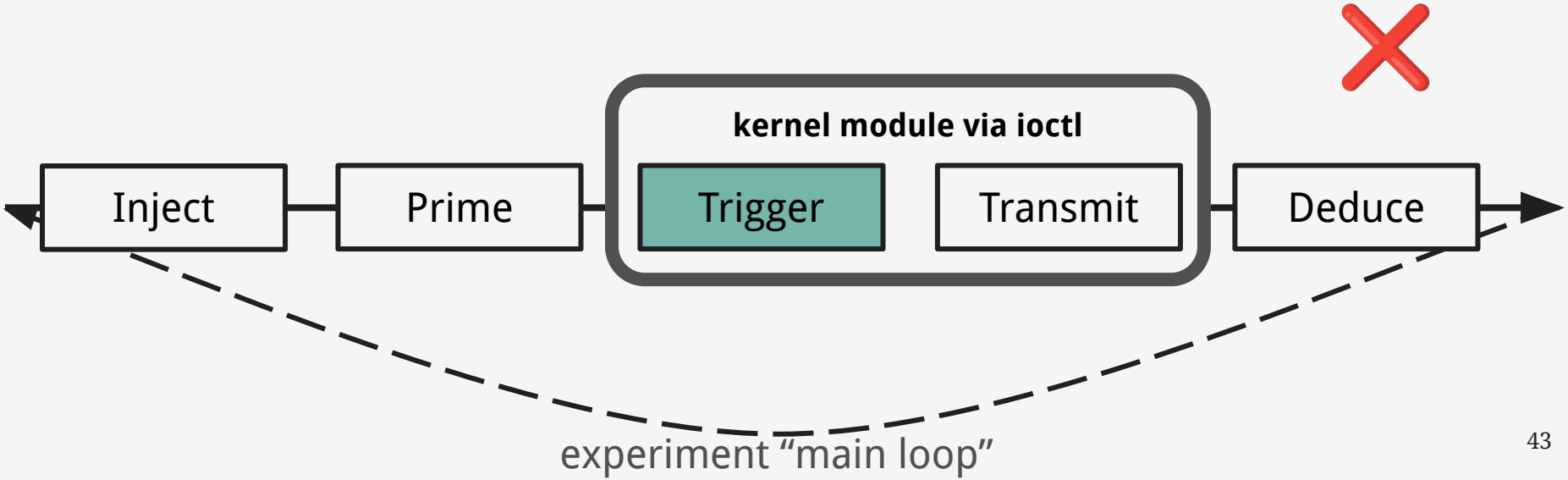
Assessing eIBRS

- Is eIBRS secretly turned on by firmware?
- **Inconclusive:** Does it really fail because of eIBRS?



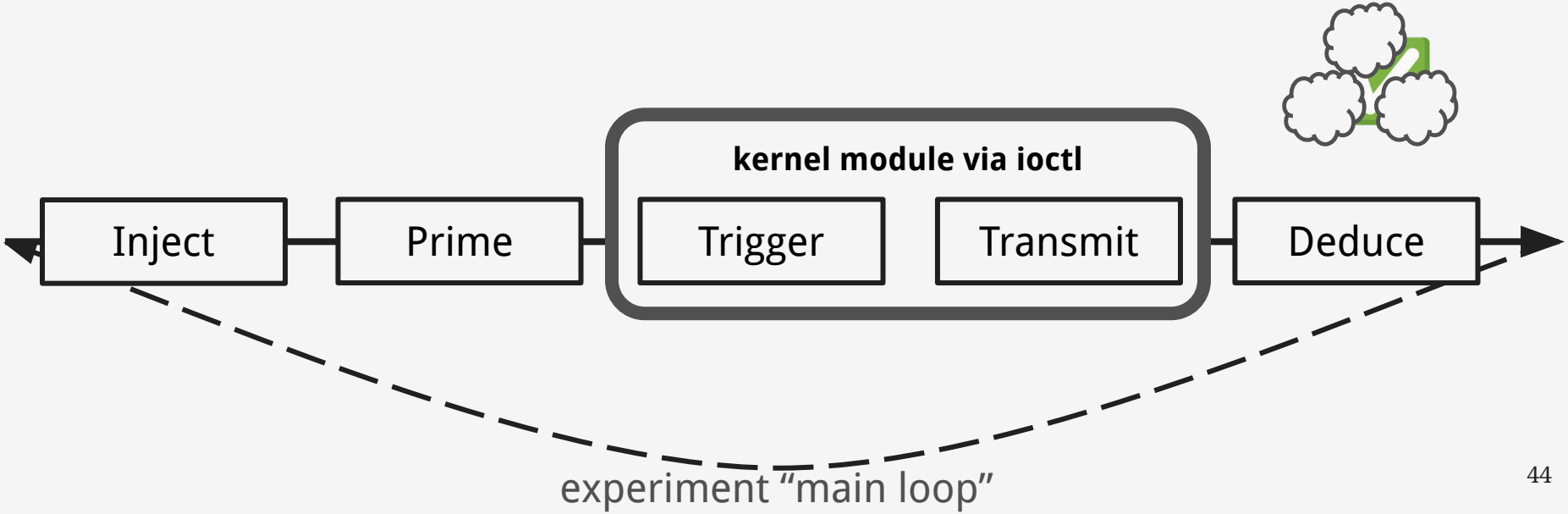
Assessing eIBRS

- Is eIBRS secretly turned on by firmware?
- **Inconclusive:** Does it really fail because of eIBRS?

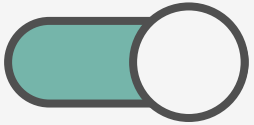


Golden Cove

- **Surprise:** Success, but success rate 0.000001!
- **Hypothesis:** eIBRS got turned off.
- **Reality:** the experiment succeeded regardless of eIBRS setting.

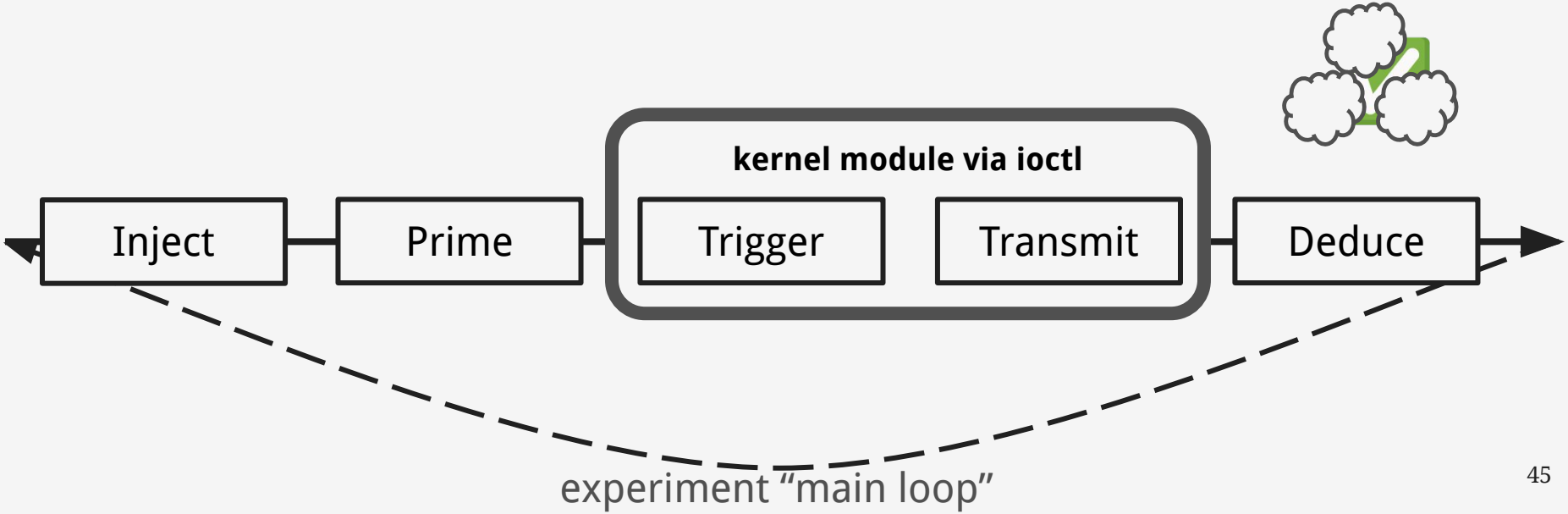


Golden Cove

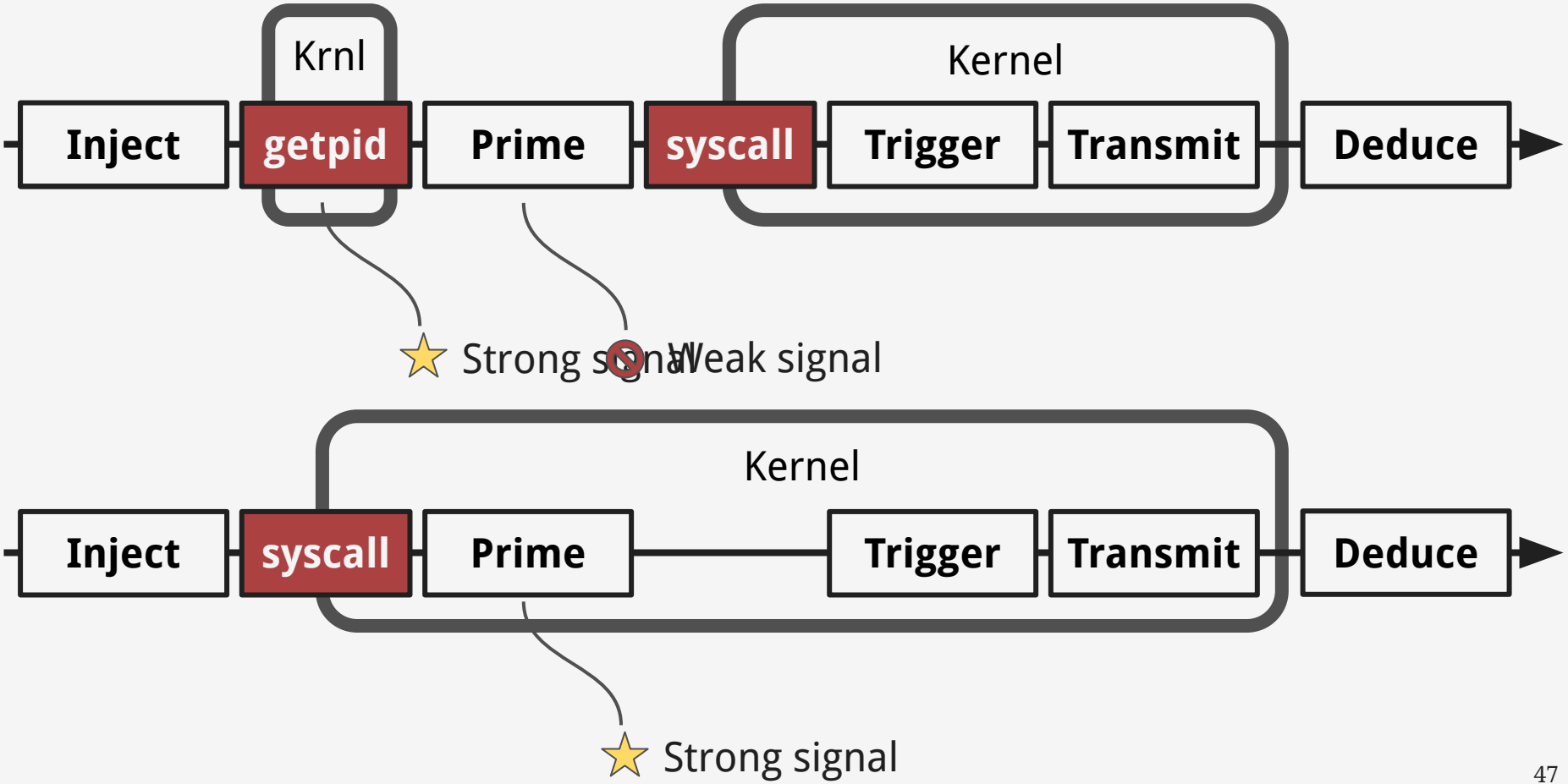


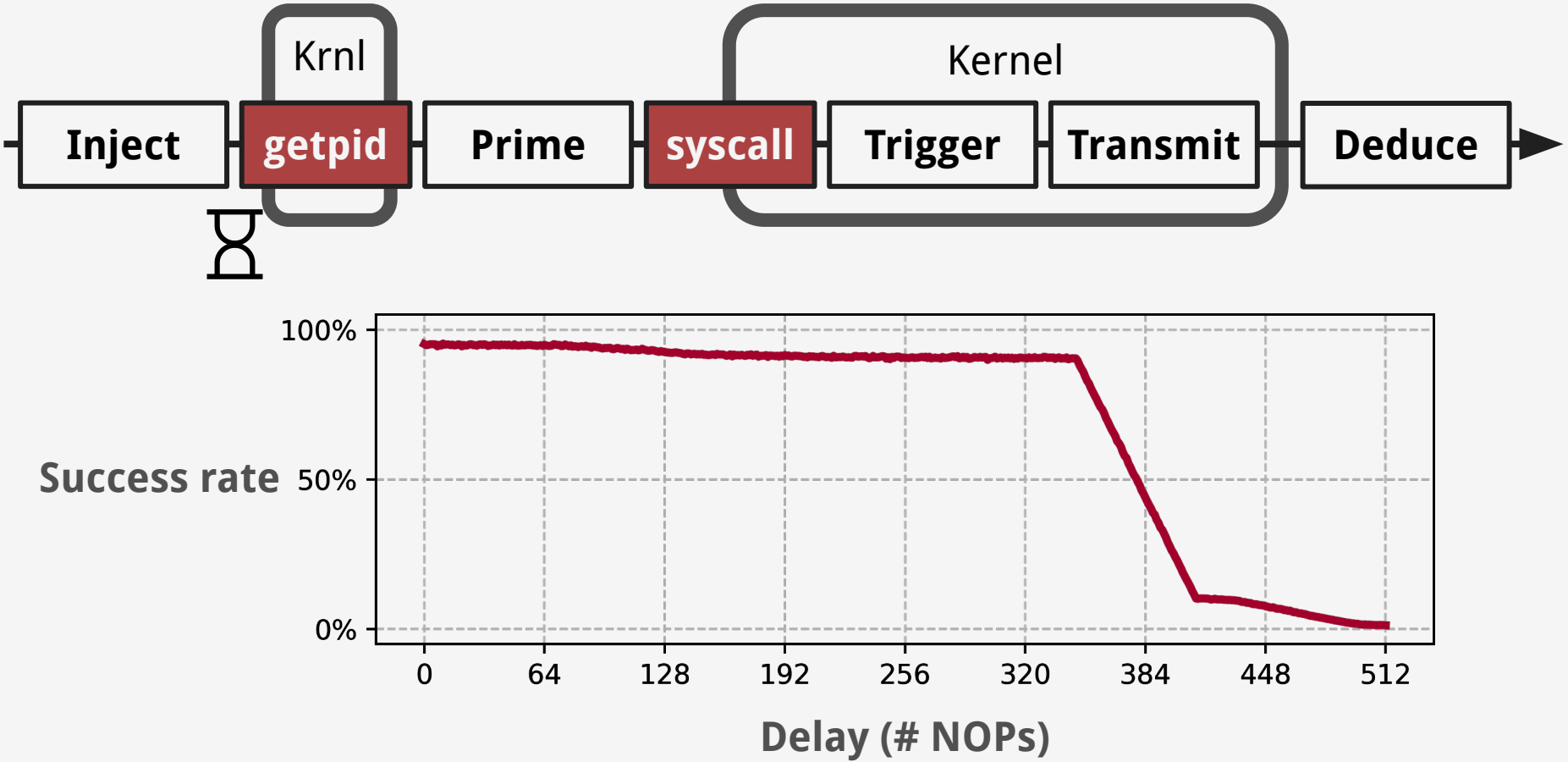
eIBRS

- **Surprise:** Success, but success rate 0.000001!
- **Hypothesis:** eIBRS got turned off.
- **Reality:** the experiment succeeded regardless of eIBRS setting.



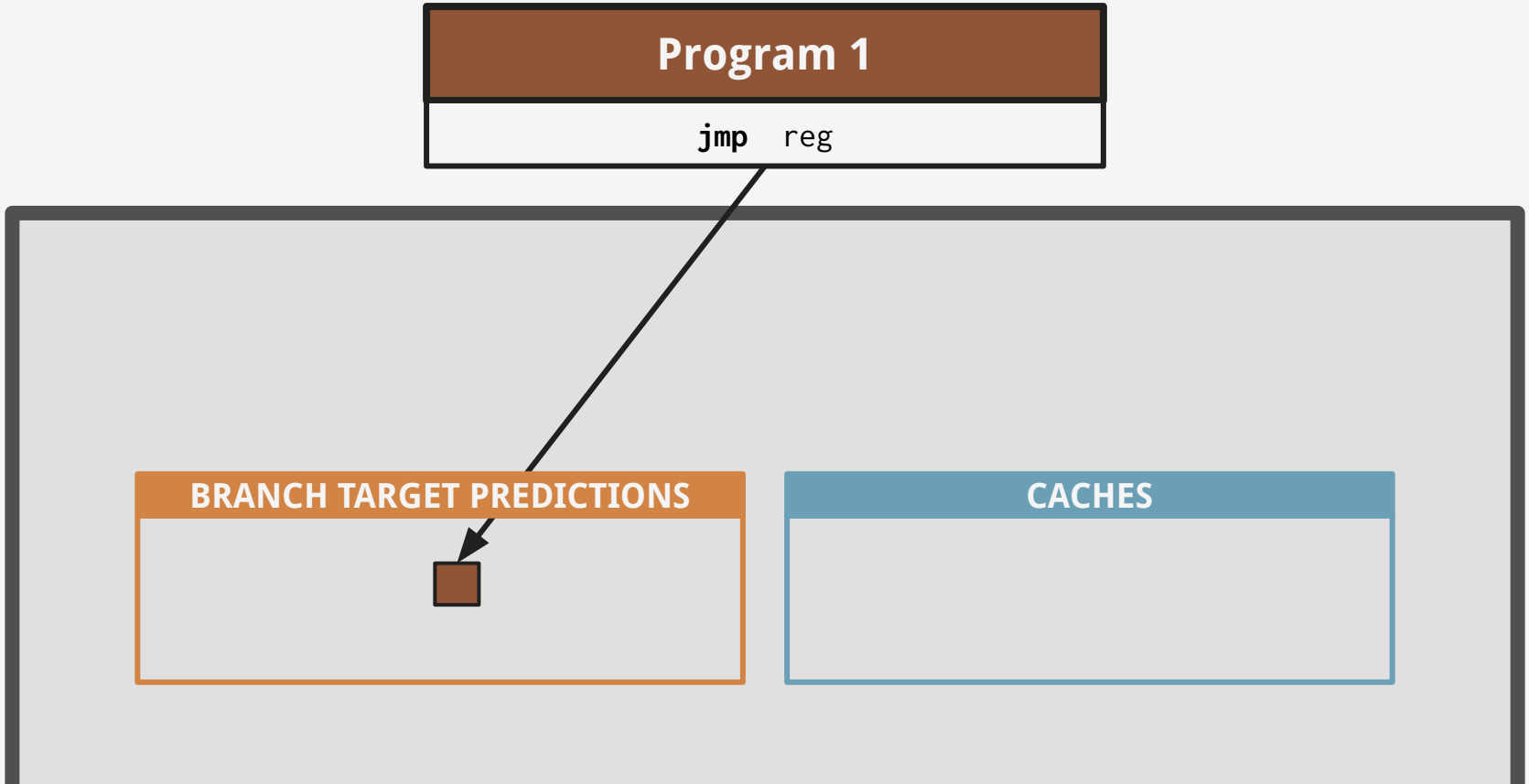
What is going on?


Krnl
Kernel
Inject
getpid
Prime
syscall
Trigger
Transmit
Deduce
Strong signal
Weak signal
Kernel
Inject
syscall
Prime
Trigger
Transmit
Deduce
Strong signal

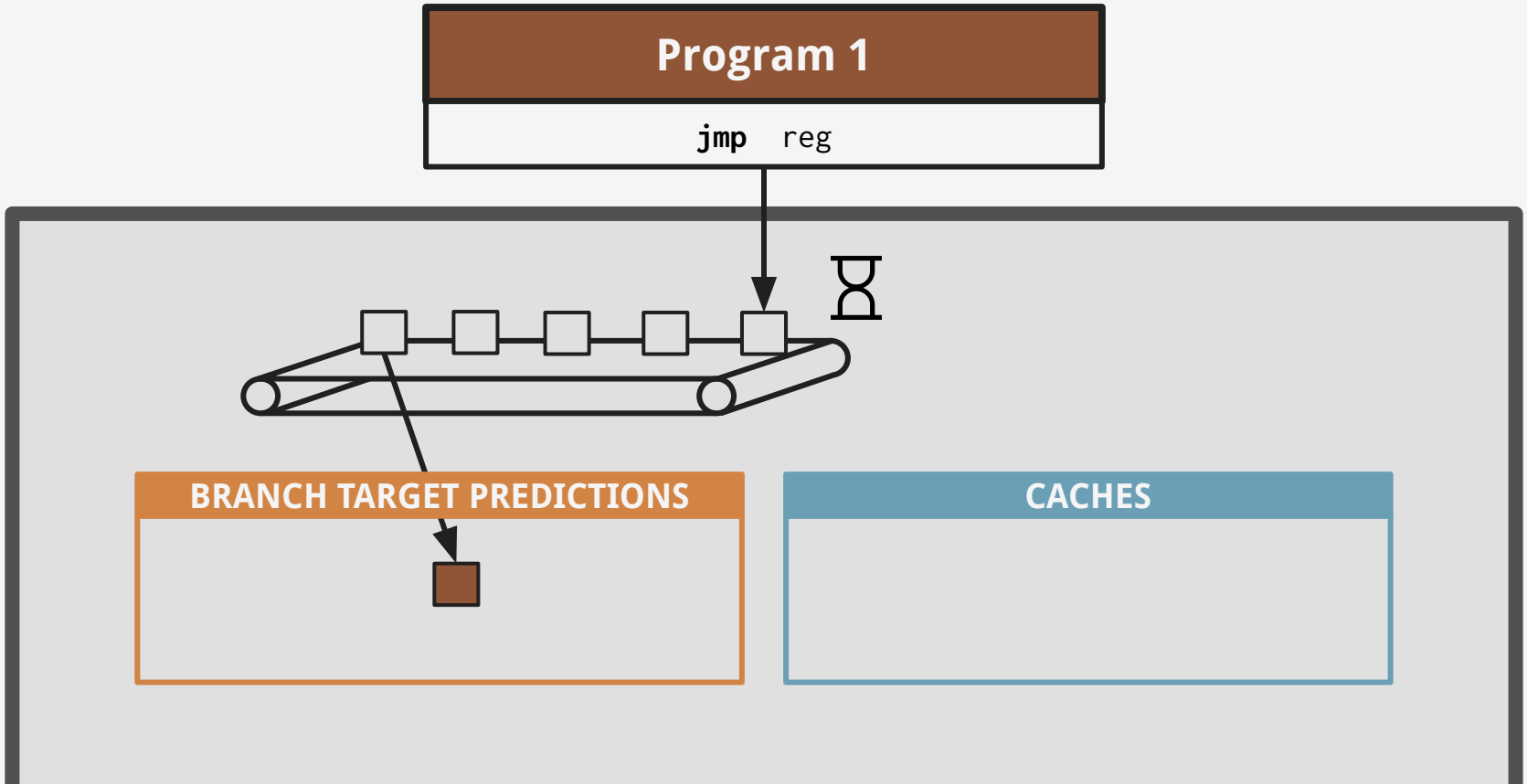

Krnl
Kernel
Inject
getpid
Prime
syscall
Trigger
Transmit
Deduce
100%
50%
0%
Success rate
0
64
128
192
256
320
384
448
512
Delay (# NOPs)

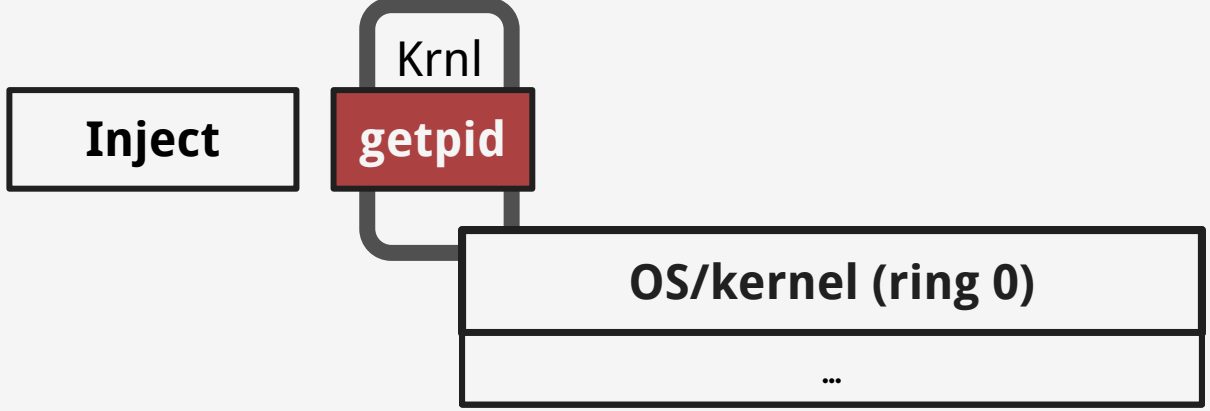
What could cause this?

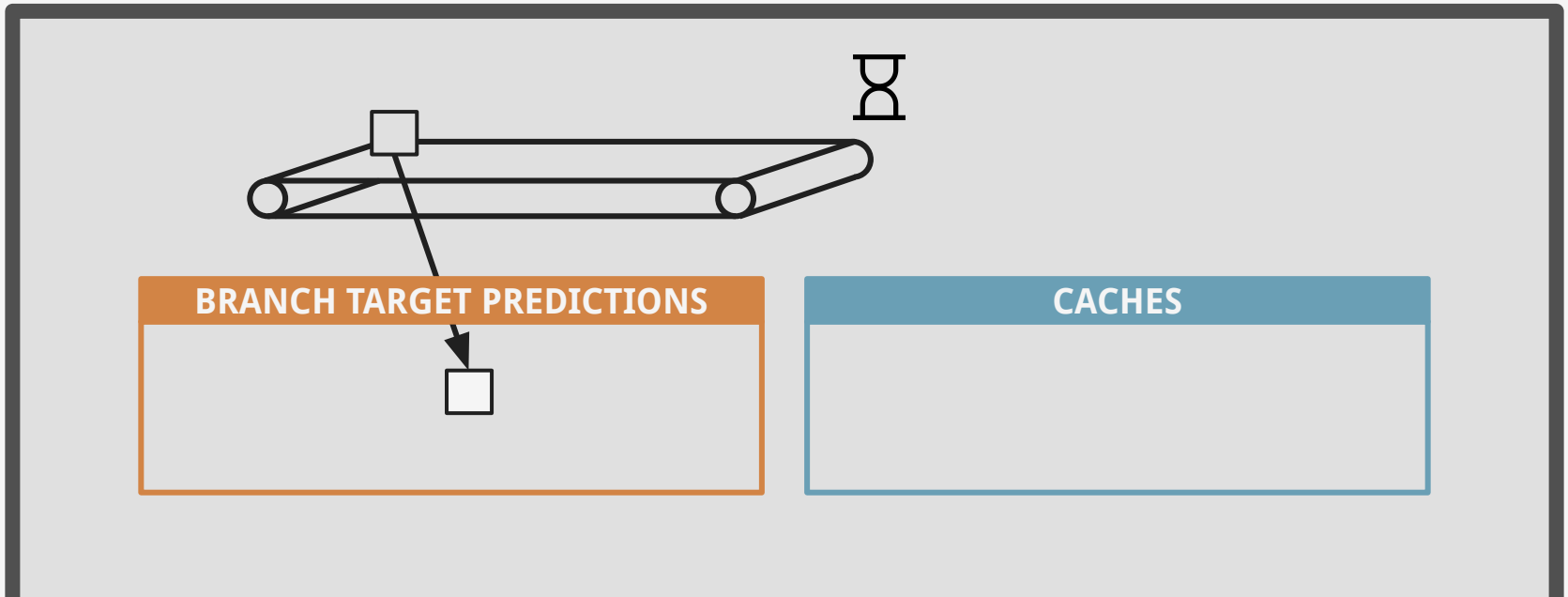
Inject



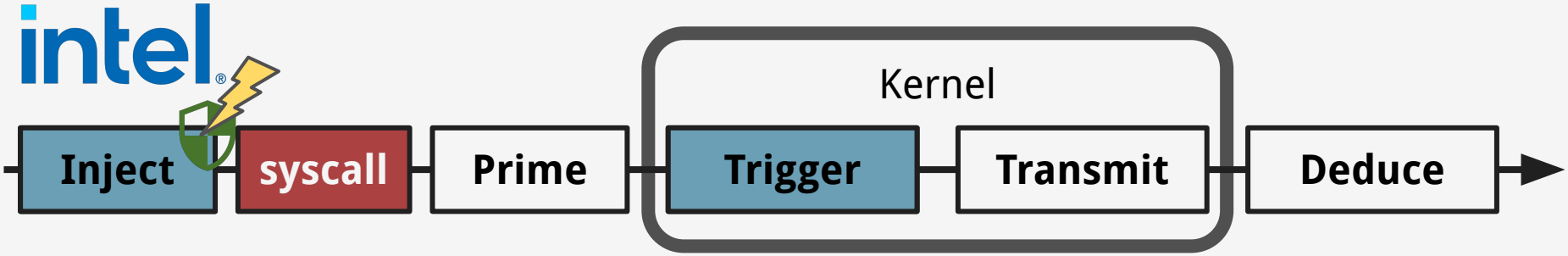
Inject

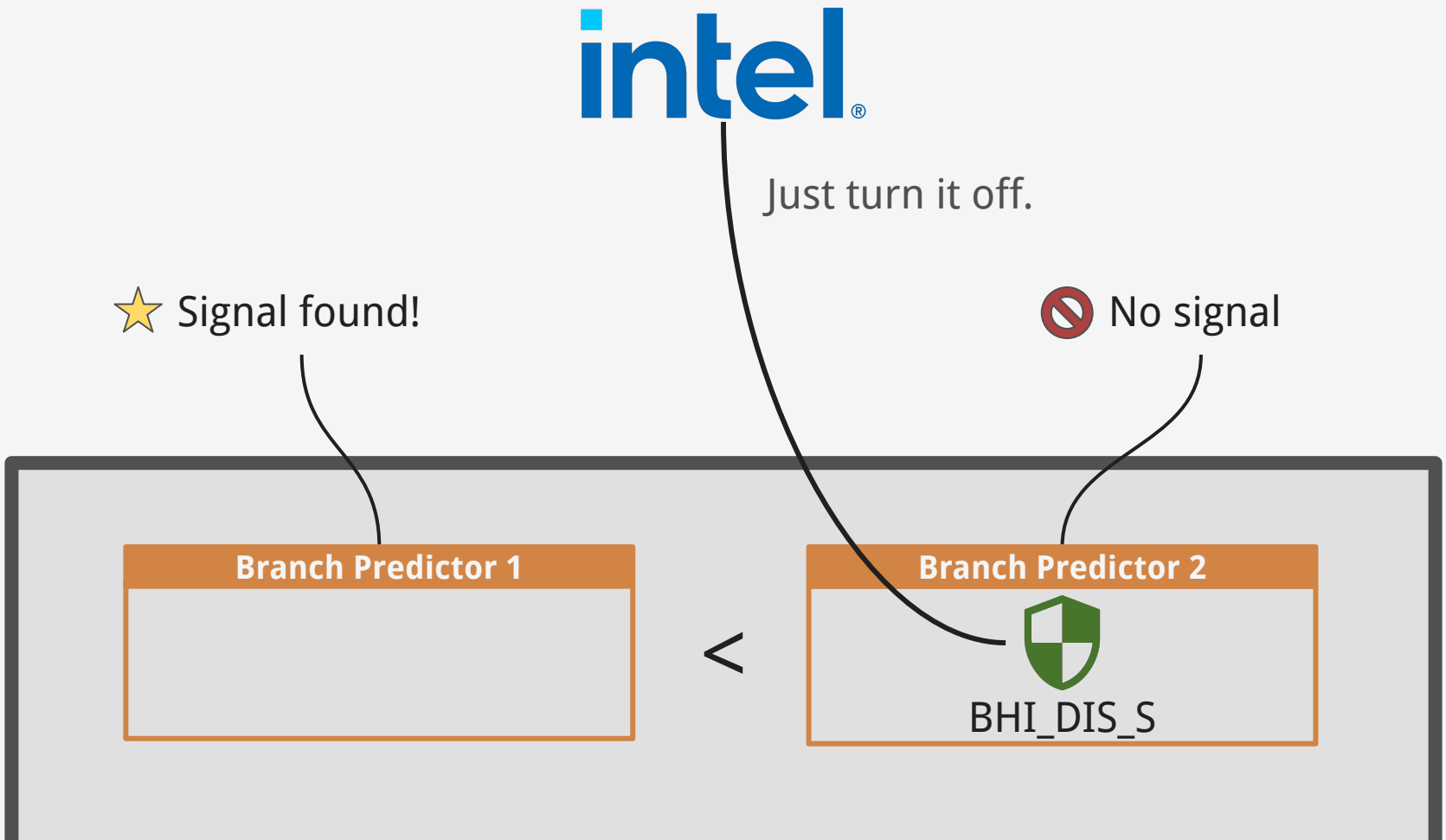



Krnl
Inject
getpid
OS/kernel (ring 0)
...

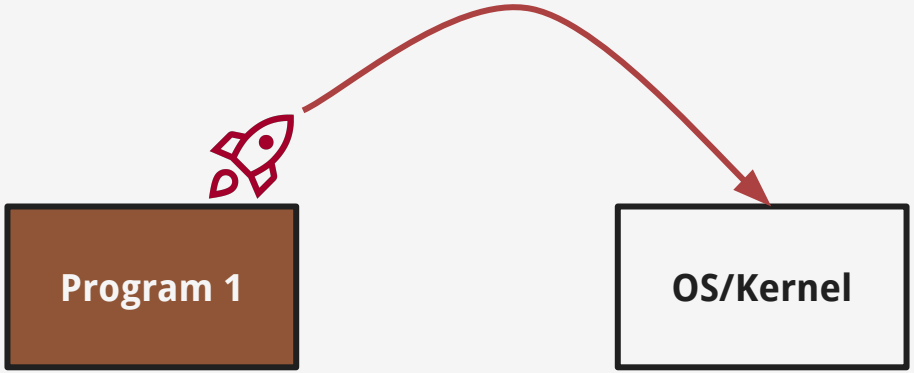

BRANCH TARGET PREDICTIONS
CACHES

Is it this easy?

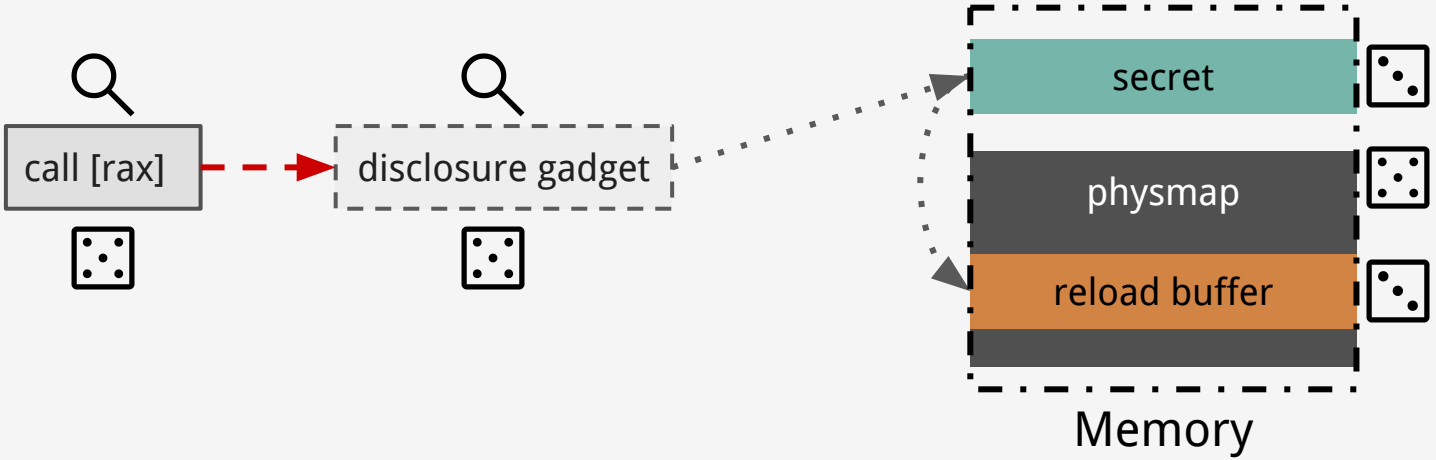

Kernel
Inject
syscall
Prime
Trigger
Transmit
Deduce

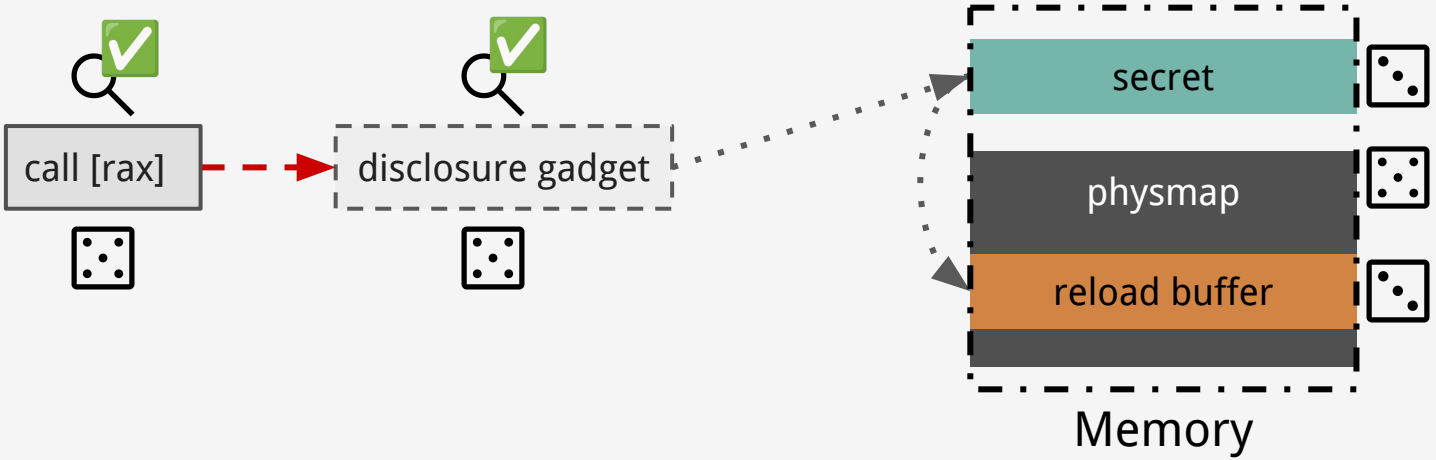

intel
Just turn it off.
Signal found!
No signal
Branch Predictor 1
<
Branch Predictor 2
BHI_DIS_S

Let’s build an attack!



- Local code execution
- Know kernel image


call [rax]
disclosure gadget
secret
physmap
reload buffer
Memory



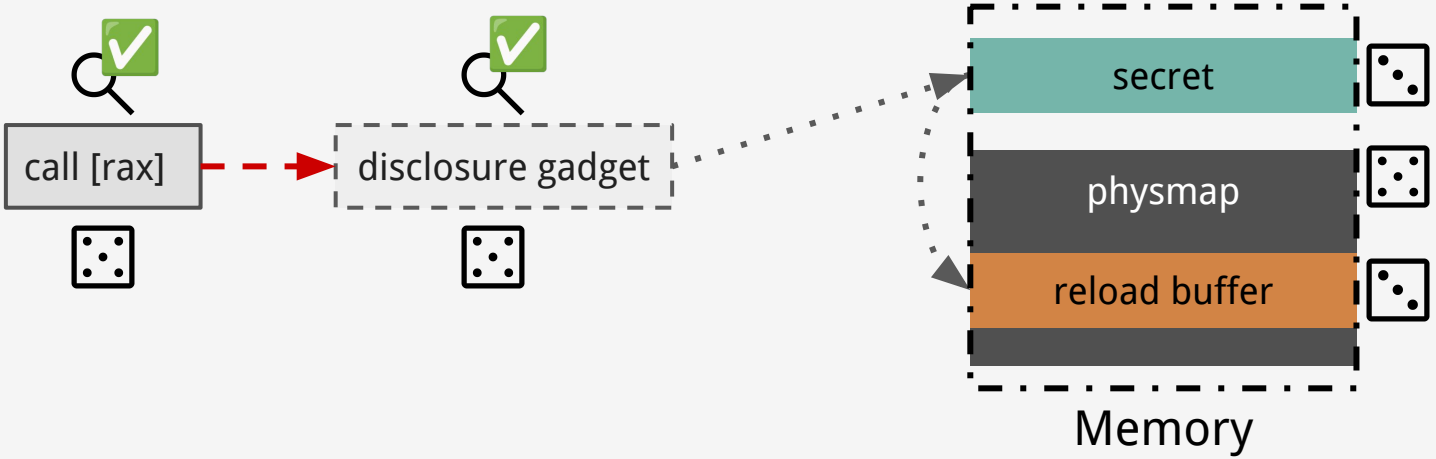
Analyze Kernel

```
key->type->read(key, buffer, buflen);
```

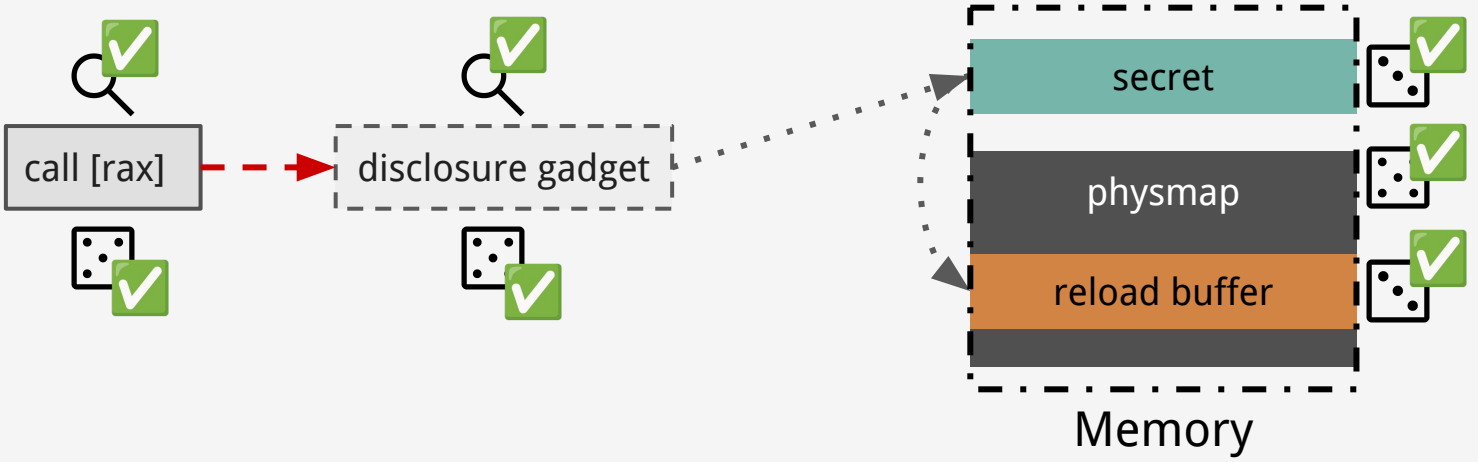
```
call [rcx]
```

```
secret = *(uint8_t *) buffer;
*(uint64_t *) (buflen + 8 * secret);
```

```
movzx edx, byte  ptr [r12]
mov   rbx, qword ptr [r13 + rdx*8]
```



Analyze Kernel



Analyze Kernel

Locate Gadgets

Locate Shared Mem

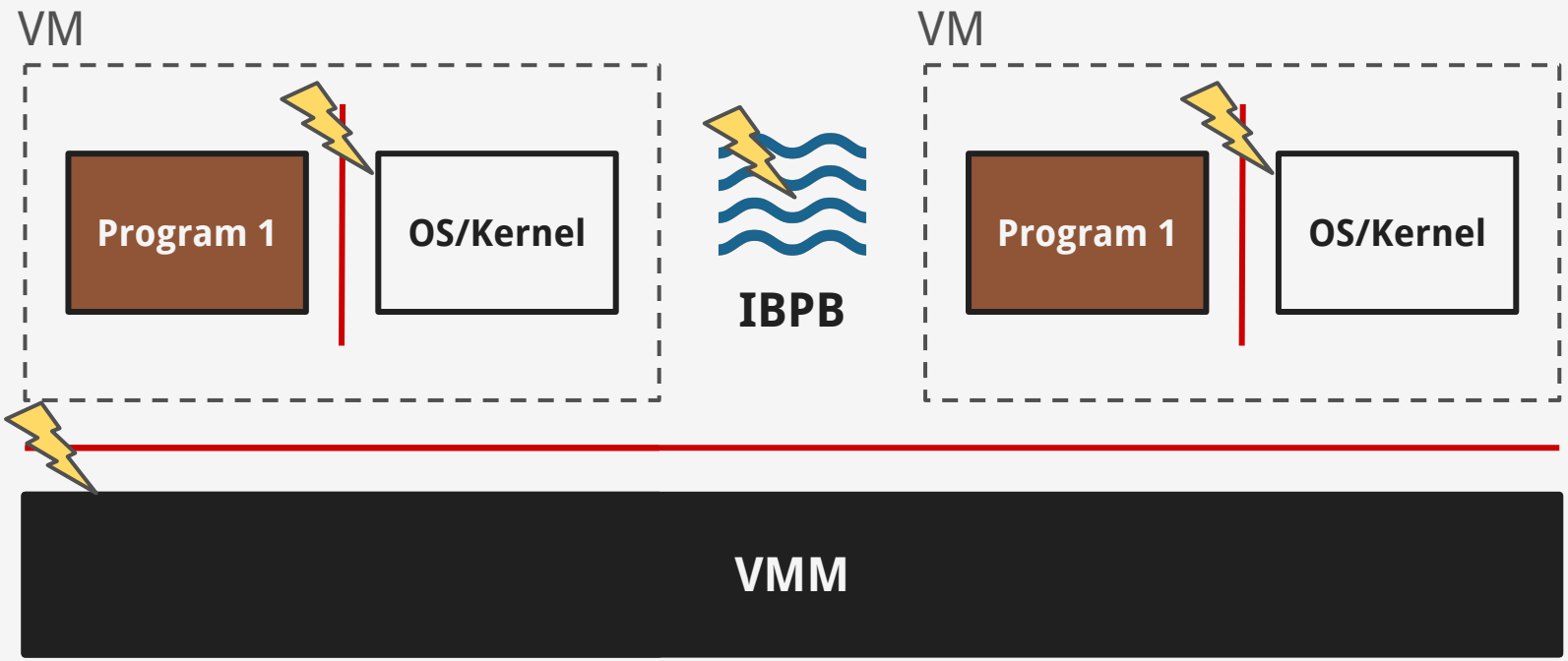
Locate `/etc/shadow`

Leak `/etc/shadow`

`root:$6$`

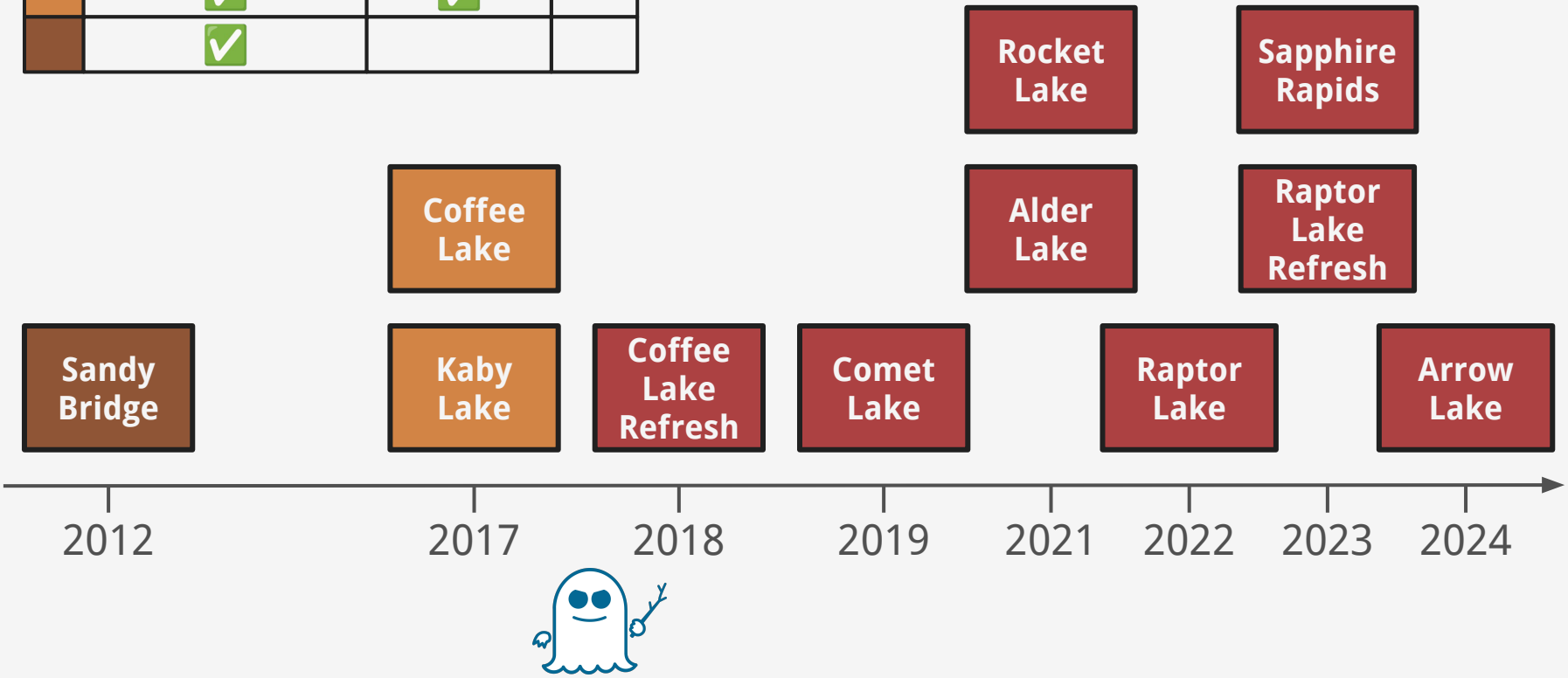
DEMO TIME!

What is affected?

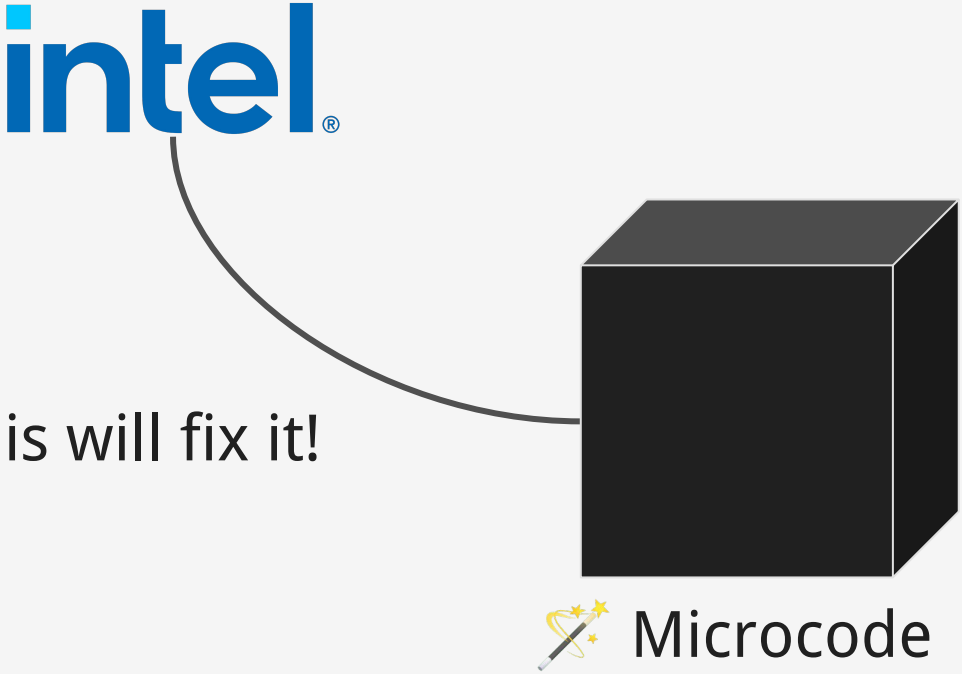

VM
Program 1
OS/Kernel
IBPB
VM
Program 1
OS/Kernel
VMM

How far back does this go?

	Async Updates	IBPB Race	BPI
	✅	✅	✅
	✅	✅	
	✅		



How do we fix it?



What does it do?

Conclusion

- An overlooked race condition undermines the long-trusted eIBRS mitigation.
- Spectre: a cat n’ mouse game between offense/defense.
- We must assess “black box” hw mitigations.
  - We’ve found shortcomings multiple times!
    - Type confusions
    - Race conditions
    - Use of uninitialized memory (uarch buffers)
    - … Classic software-style issues emerging in the hardware domain

Black Hat Sound Bytes

1. 8 years in, and we keep finding way to exploit Spectre.
2. Hardware defenses (like eIBRS) need to independently assessed.
3. Hardware vendors must take more responsibility.
    - Stop relying on > 9-months embargoes.
    - Adhere to standard 90 day responsible disclosure period.
    - Consider “chicken bits” for all potentially exploitable CPU features.