

Microarchitectural Inception

Jo Van Bulck, Michael Schwarz, Daniel Gruss, Moritz Lipp

rC3 — Remote Chaos Experience, December 2020

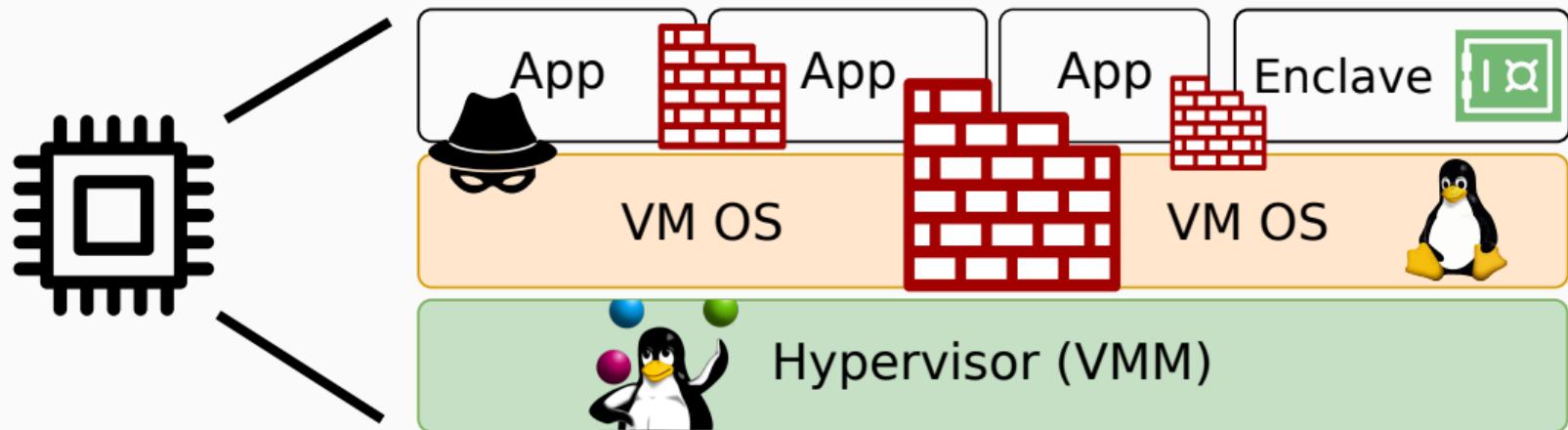


check in obs that picture looks pretty much the same



...Oh, we already started. My name is JVB and I welcome you to our talk: Load Value Injection. Before we start there's something you should know about me. I specialize

Processor security: Hardware isolation mechanisms

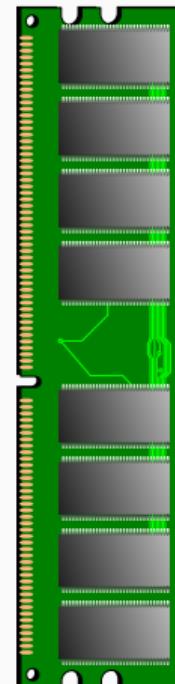
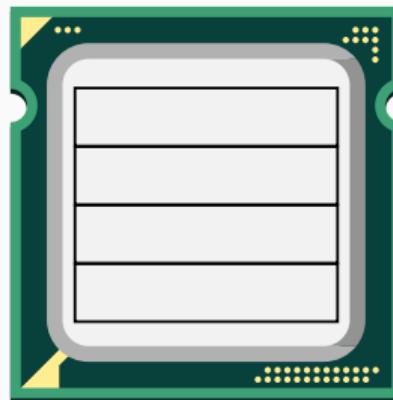




Daniel/Michael: well, have you thought about the uarch foundation under your nice arch walls?

Microarchitectural timing attacks: CPU cache

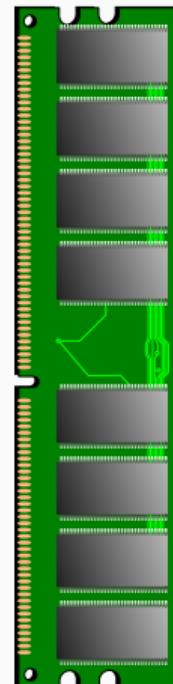
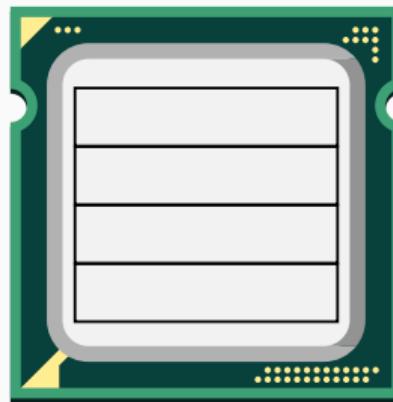
```
printf("%d", i);  
printf("%d", i);
```



Microarchitectural timing attacks: CPU cache

```
printf("%d", i);  
printf("%d", i);
```

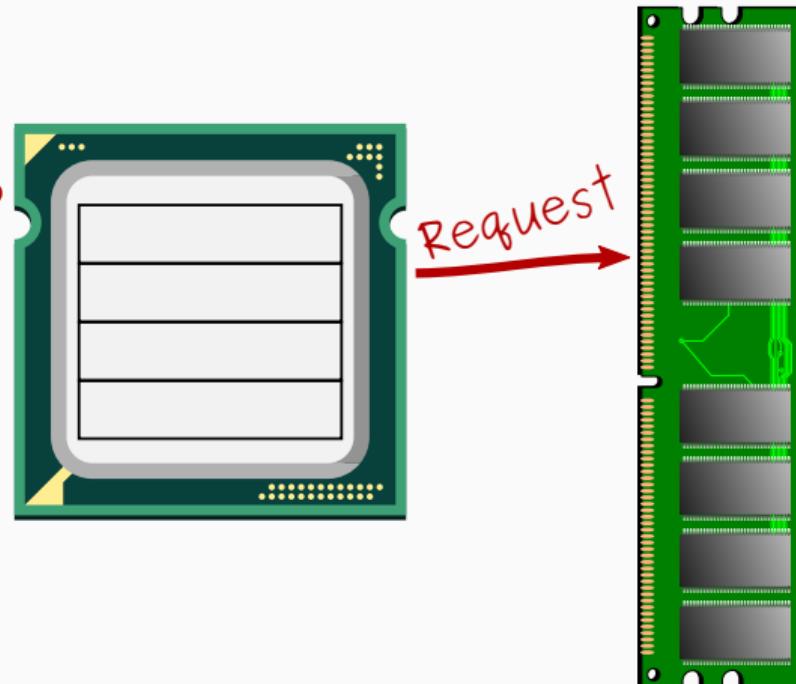
Cache miss



Microarchitectural timing attacks: CPU cache

```
printf("%d", i);  
printf("%d", i);
```

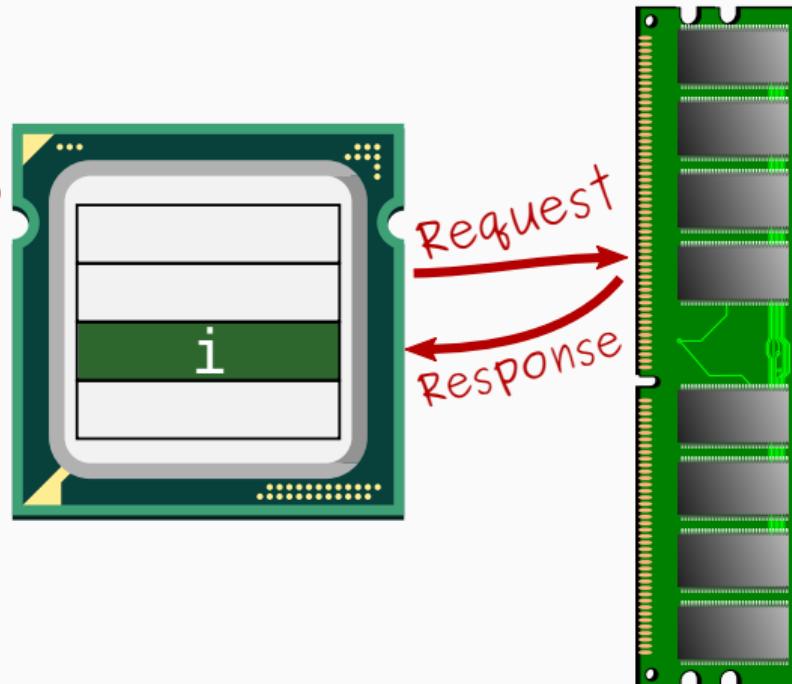
Cache miss



Microarchitectural timing attacks: CPU cache

```
printf("%d", i);  
printf("%d", i);
```

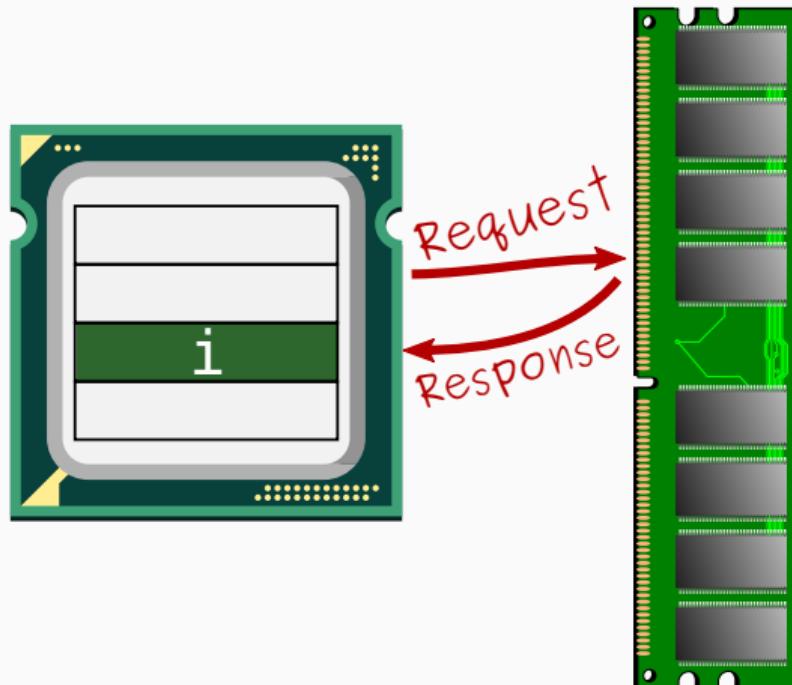
Cache miss



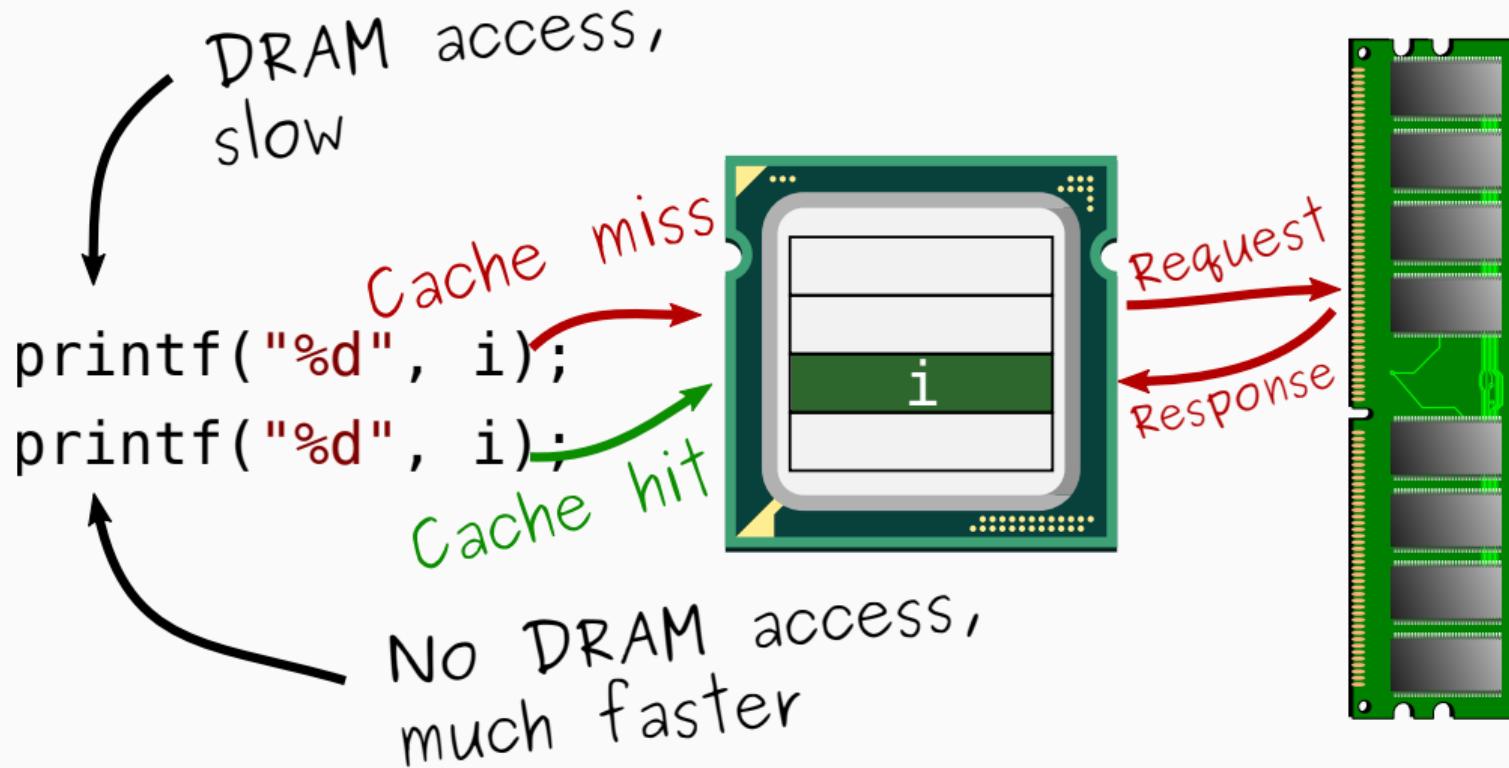
Microarchitectural timing attacks: CPU cache

```
printf("%d", i);  
printf("%d", i);
```

Cache miss
Cache hit



Microarchitectural timing attacks: CPU cache

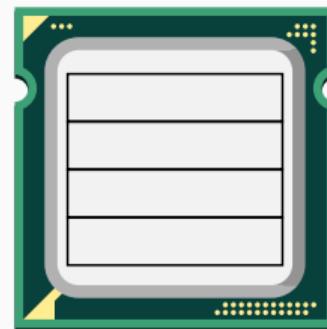


Flush+Reload

ATTACKER

flush
access

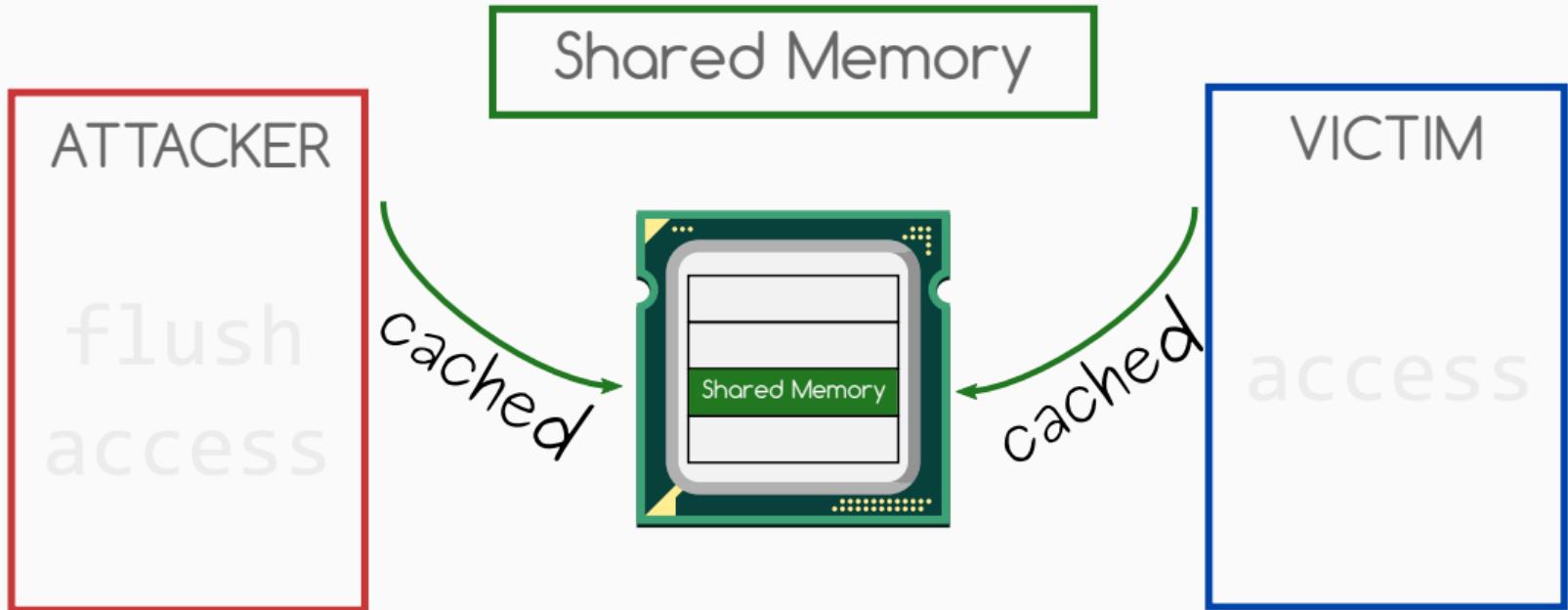
Shared Memory



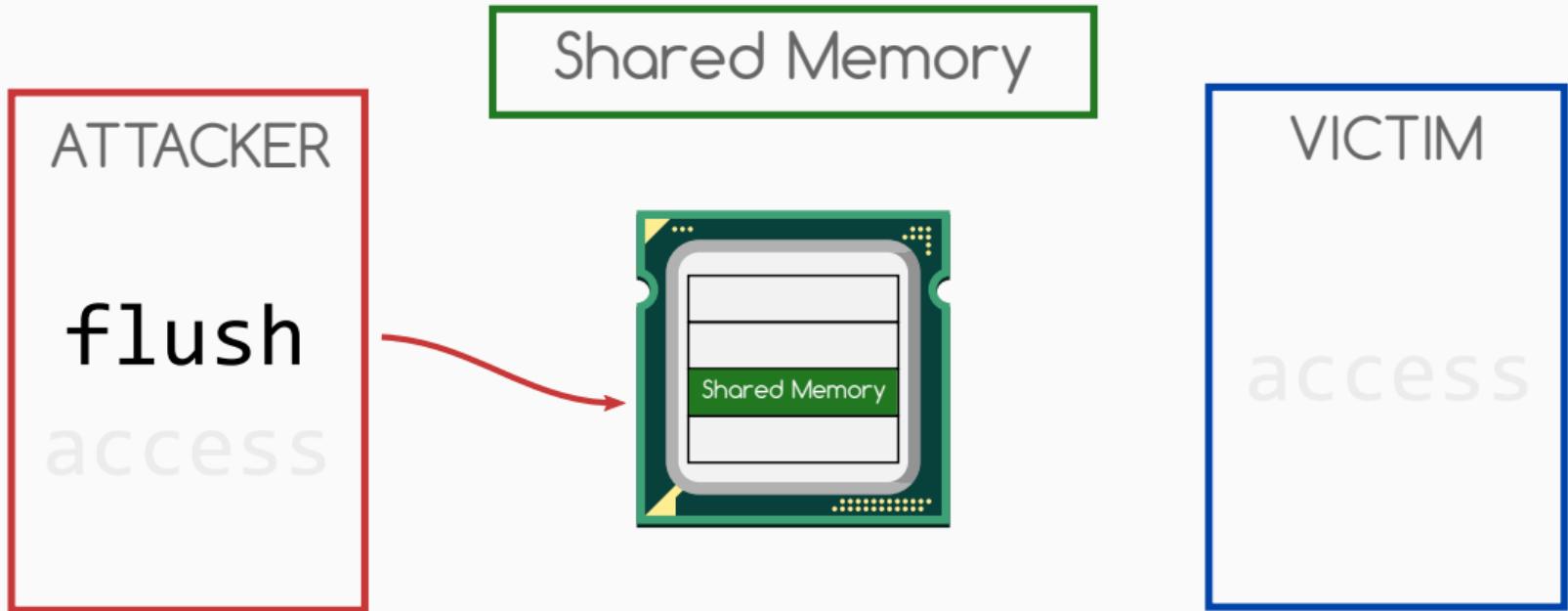
VICTIM

access

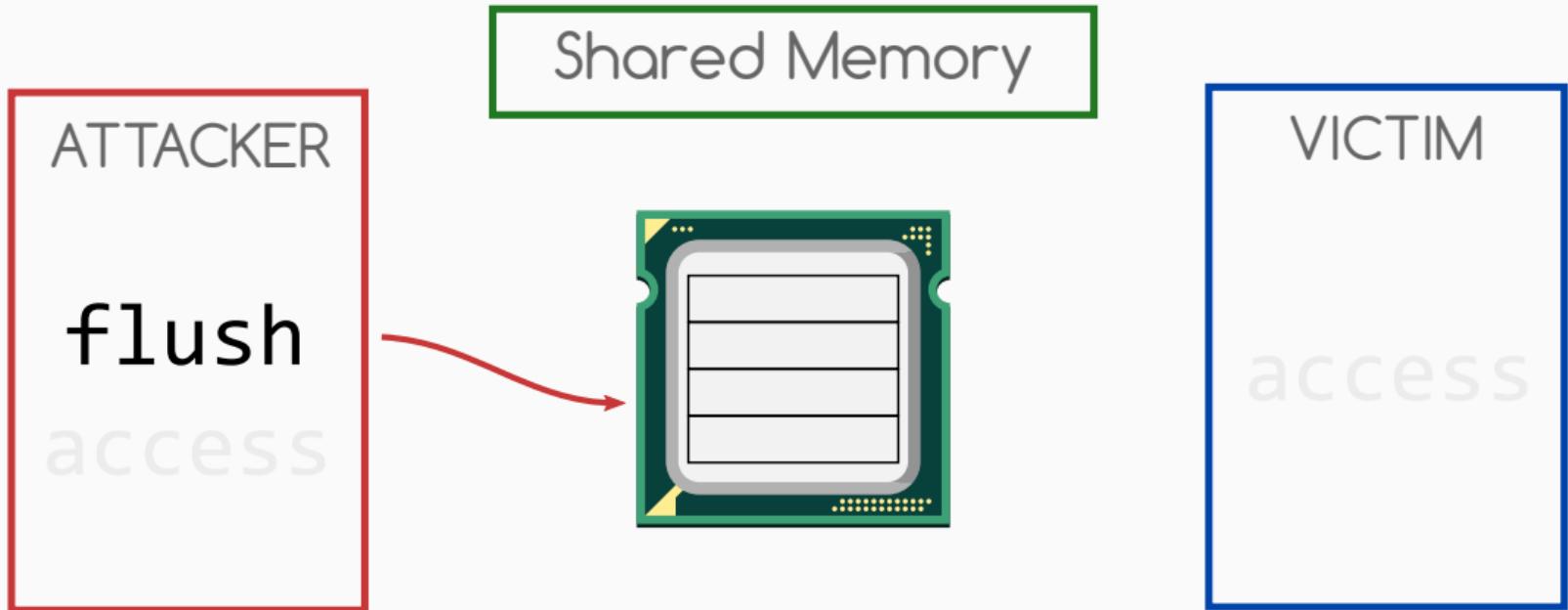
Flush+Reload



Flush+Reload



Flush+Reload

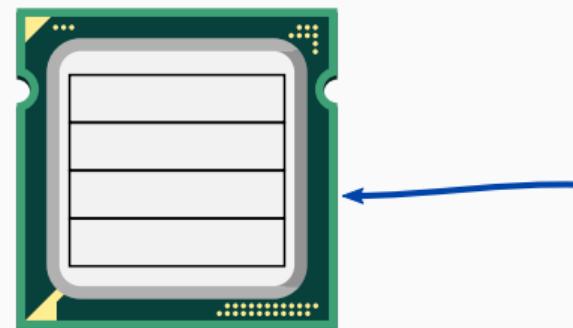


Flush+Reload

ATTACKER

flush
access

Shared Memory

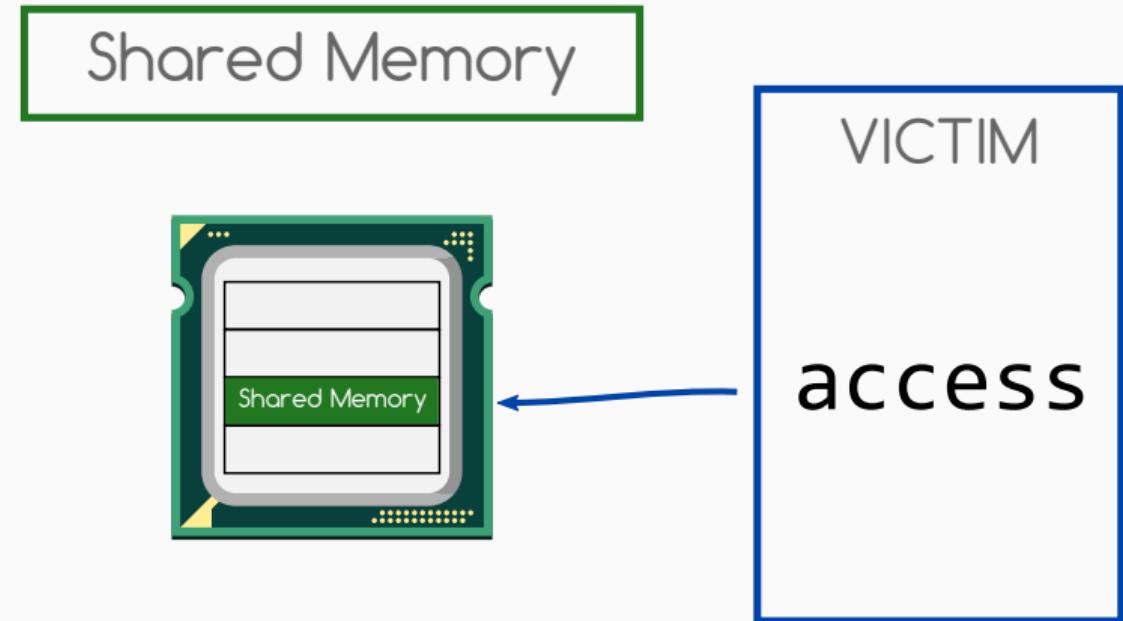


VICTIM

access

Flush+Reload

ATTACKER
flush
access

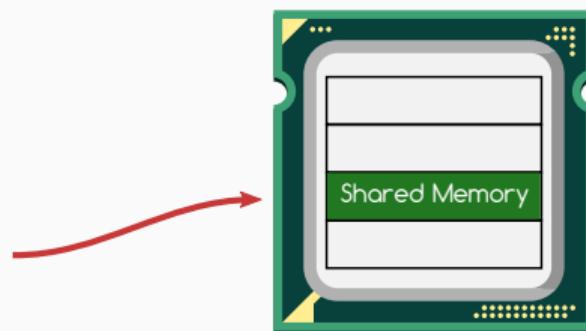


Flush+Reload

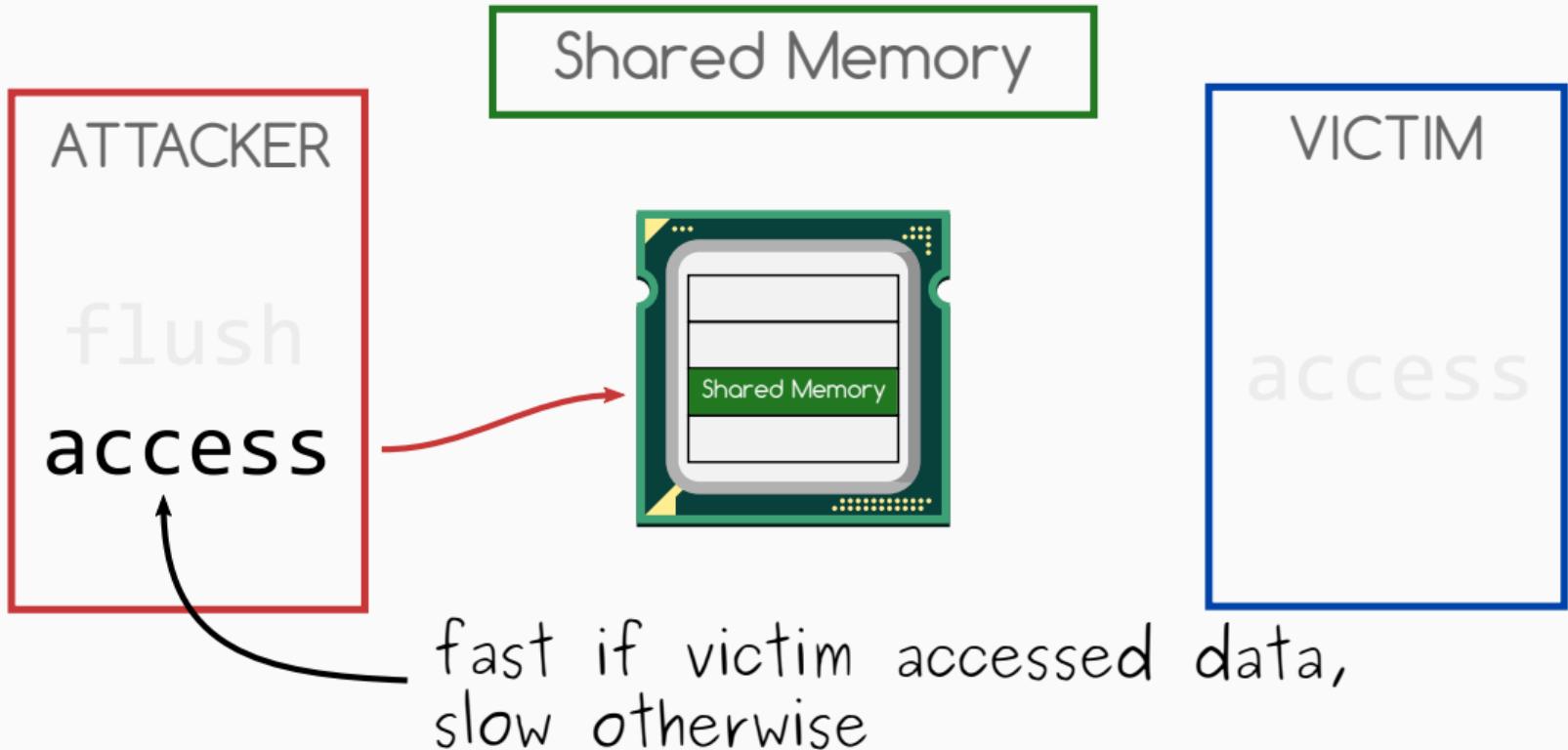
ATTACKER
flush
access

Shared Memory

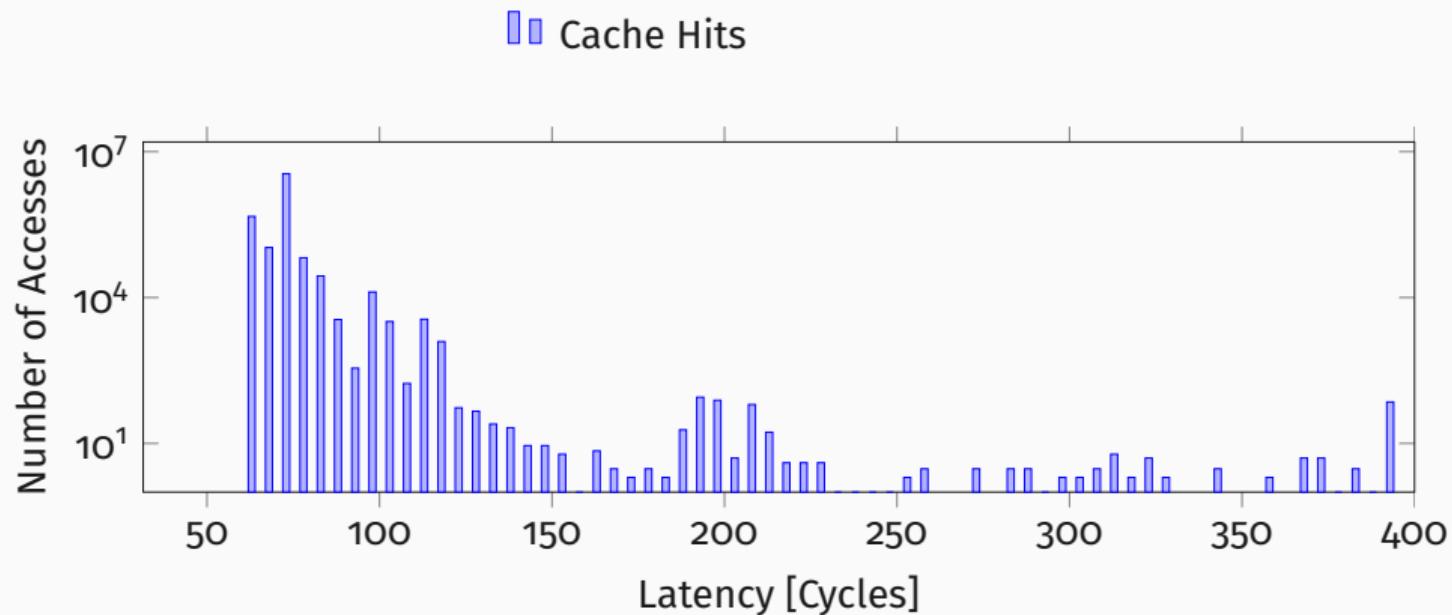
VICTIM



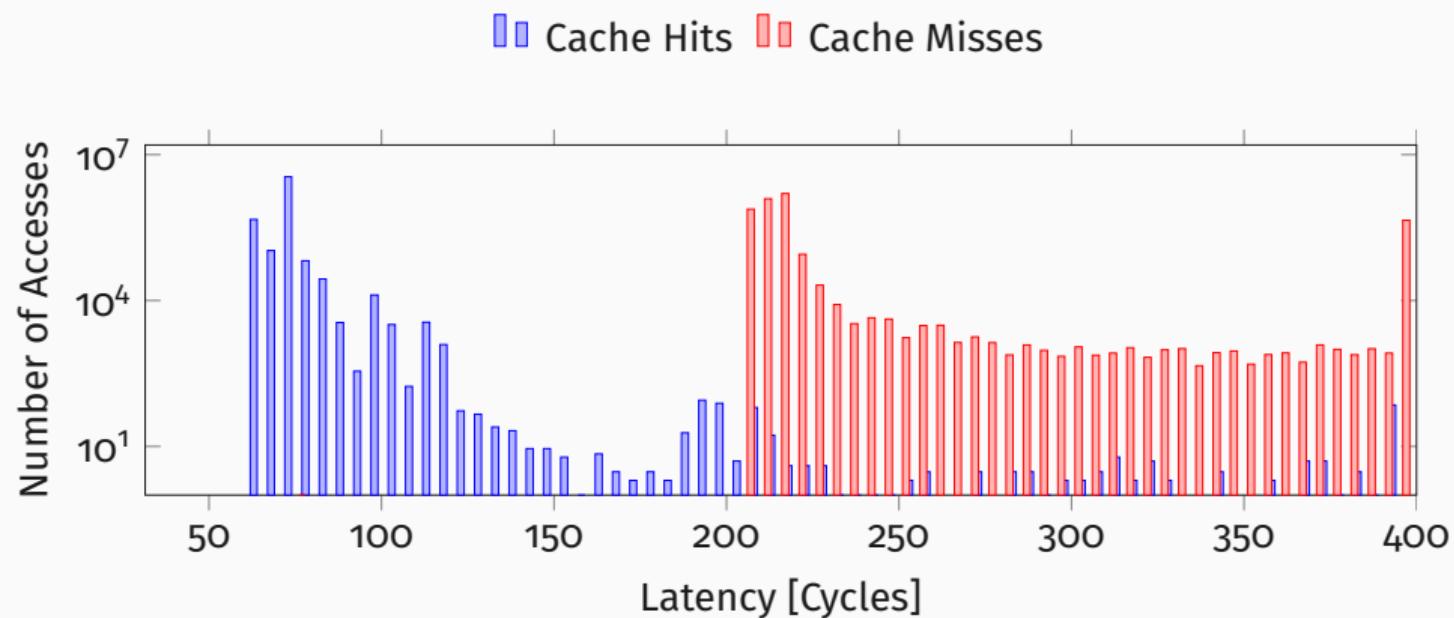
Flush+Reload



Memory Access Latency



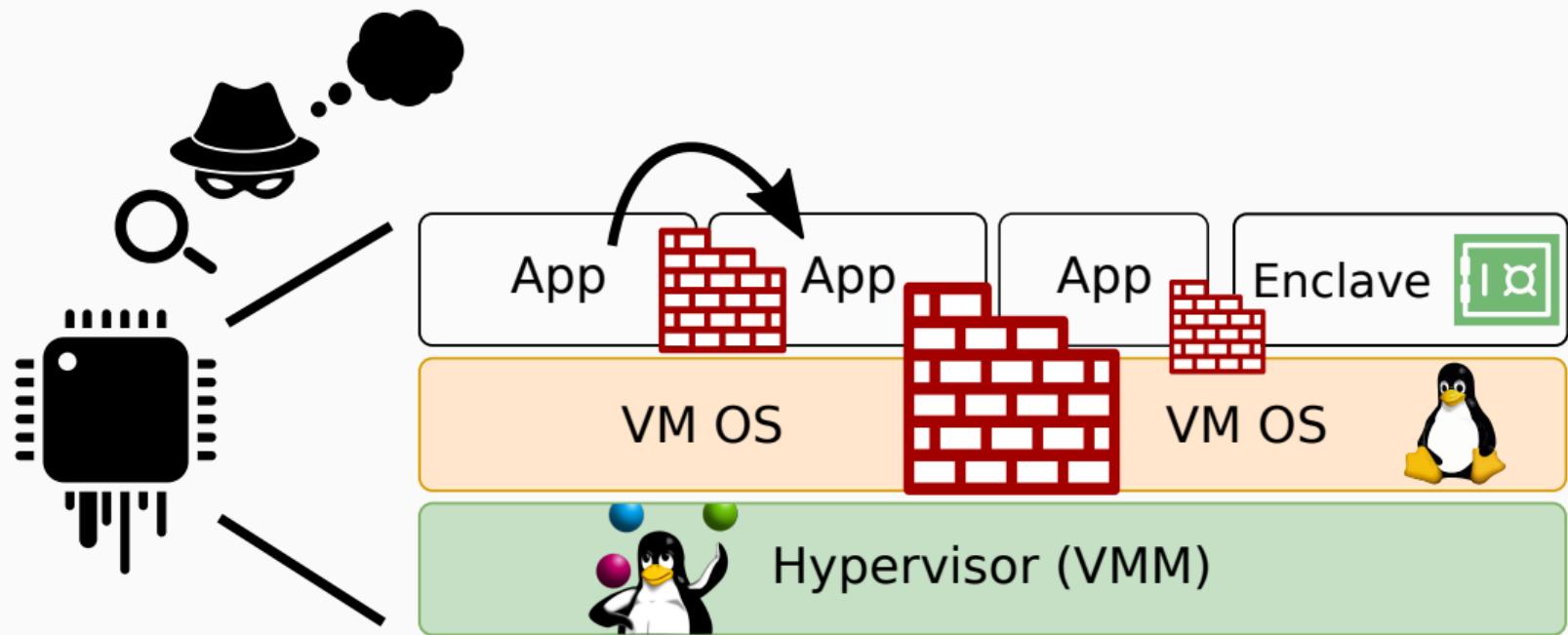
Memory Access Latency





We can communicate across protection walls
using microarchitectural side channels!

Leaky processors: Jumping over protection walls with side-channels





2:03:02



Inception



check in obs that picture looks pretty much the same



Jo: "I have a test for you" Michael: "You're not gonna tell me anything about this first?" Jo: "Before I describe the project, I have to know you can do it" Michael: "Why?"



2:03:04



Inception



check in obs that picture looks pretty much the same

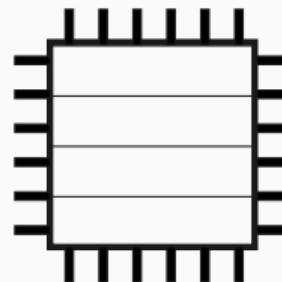


Meltdown: Transiently encoding unauthorized memory

User Memory

	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

char value = kernel[0]



Meltdown: Transiently encoding unauthorized memory

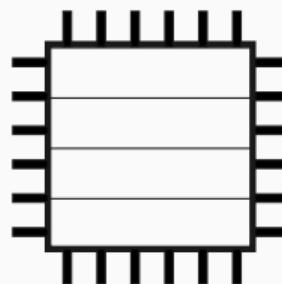
User Memory

	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

`char value = kernel[0]`



Page fault (Exception)

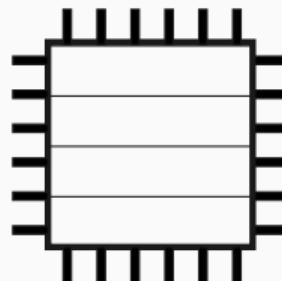


Meltdown: Transiently encoding unauthorized memory

User Memory

	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

```
char value = kernel[0]  
          ↓  
mem[value]  
          K  
          ⚡ Page fault (Exception)  
          Out of order
```



Meltdown: Transiently encoding unauthorized memory

User Memory

	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

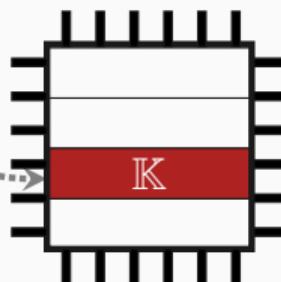
`char value = kernel[0]`

`mem[value]`

Page fault (Exception)

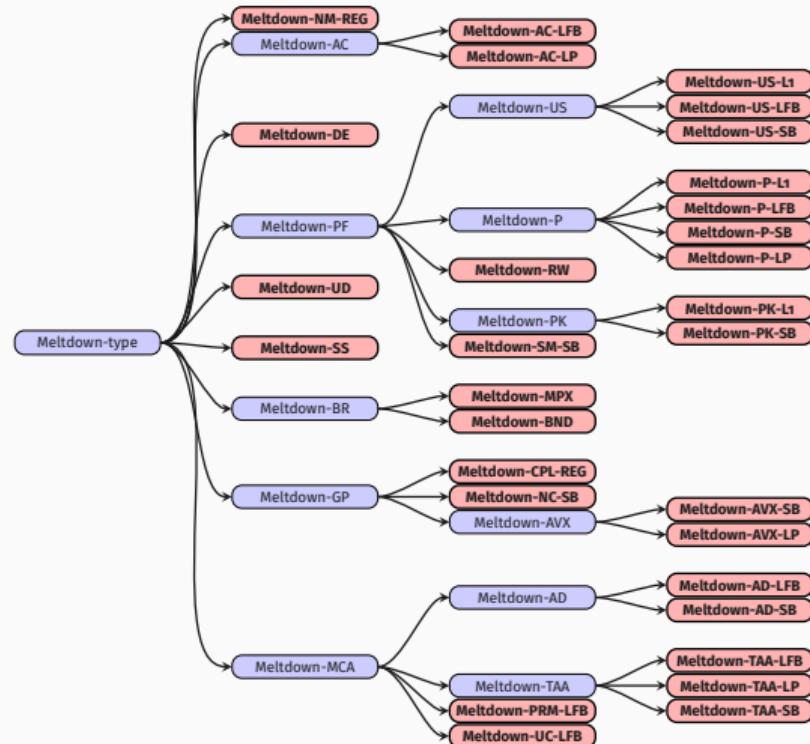
Out of order

K



Meltdown variants: The transient-execution zoo

<https://transient.fail>



Meltdown variants: Address dependencies

Meltdown	Page Number		Page Offset	
	51	47	Physical Virtual	12 12
			11	0

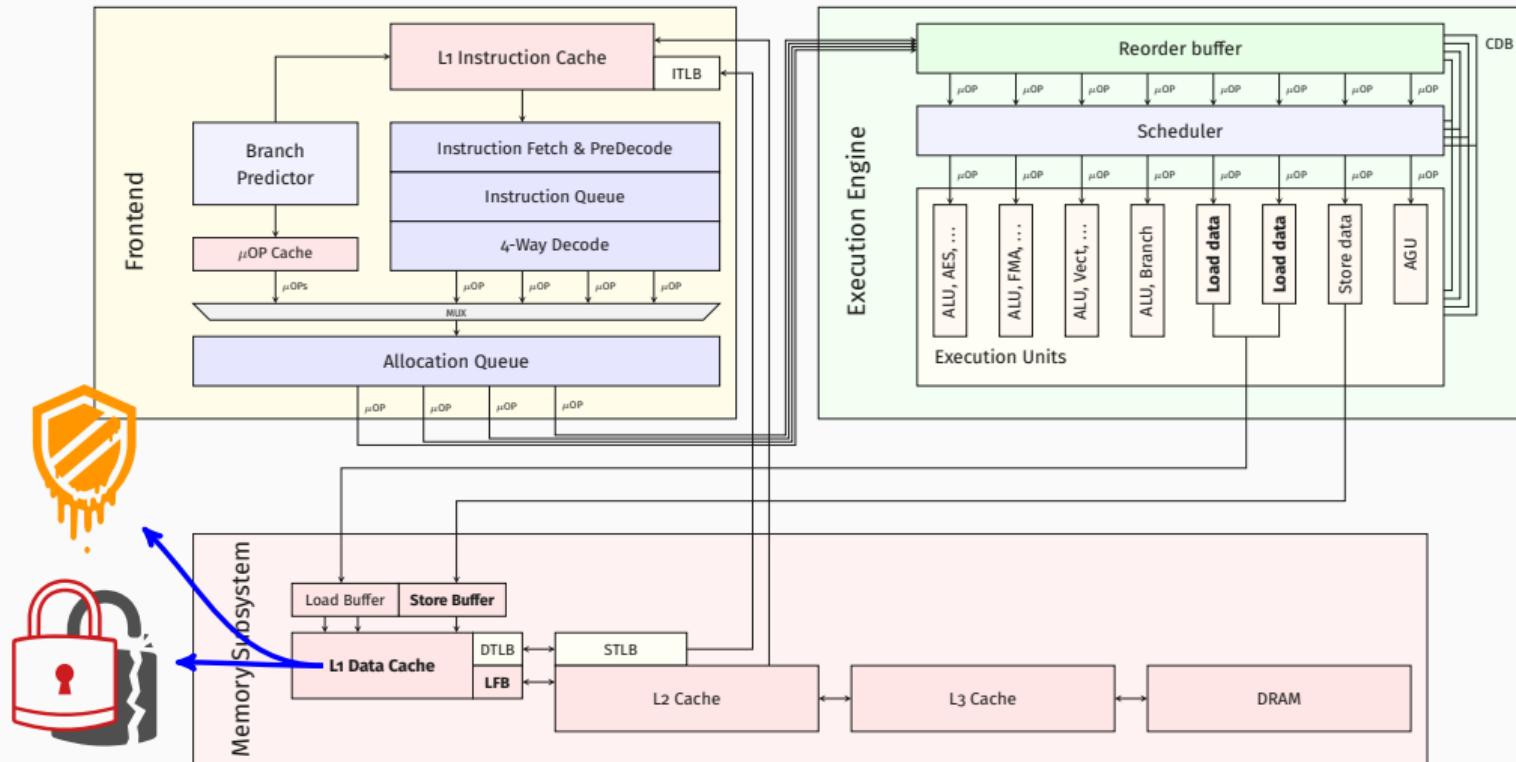
Meltdown variants: Address dependencies

	Page Number	Page Offset
Meltdown	51	Physical
	47	Virtual
Foreshadow	51	Physical
	47	Virtual

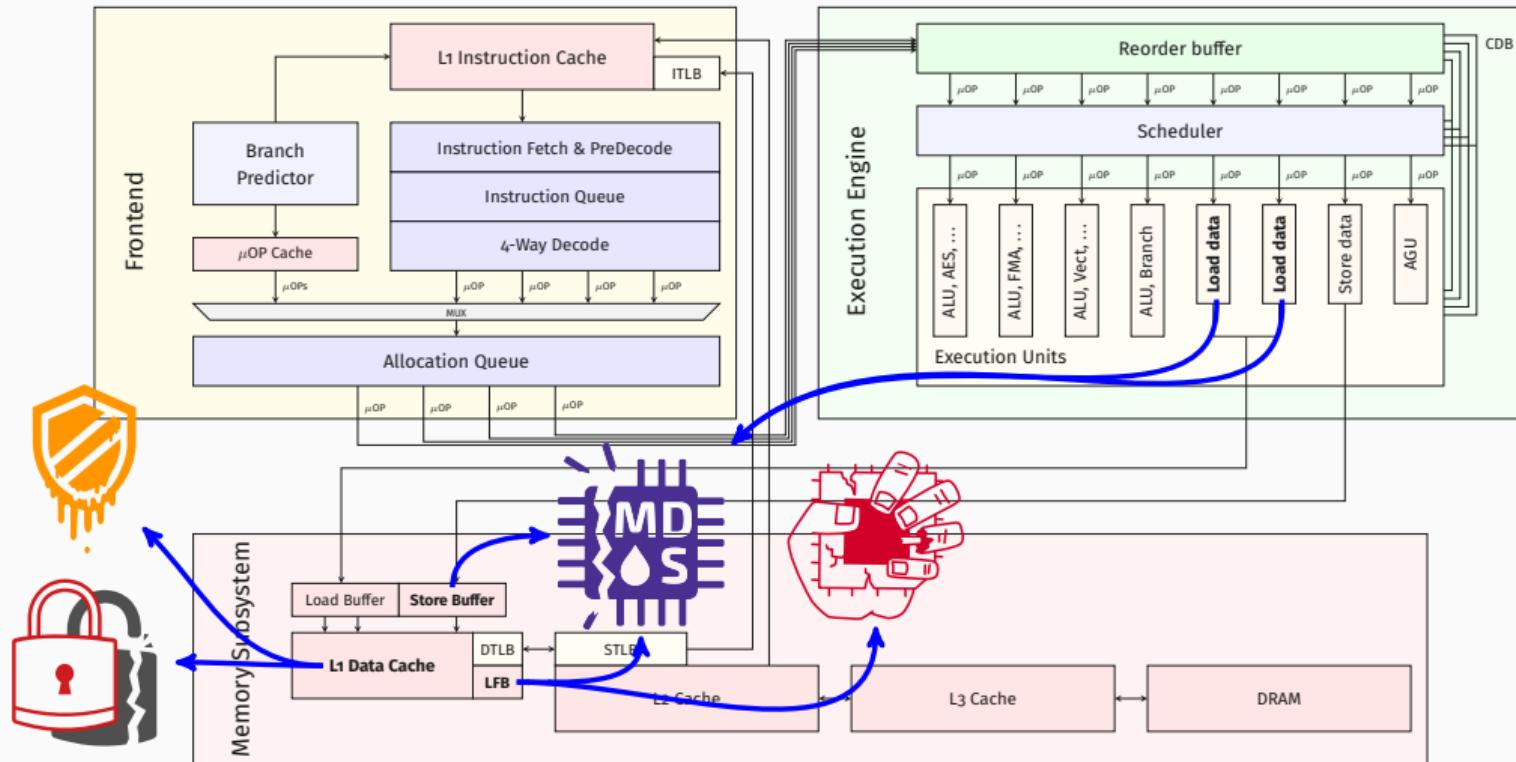
Meltdown variants: Address dependencies

		Page Number		Page Offset
Meltdown	51	Physical	12	11
	47	Virtual	12	0
Foreshadow	51	Physical	12	11
	47	Virtual	12	0
Fallout	51	Physical	12	11
	47	Virtual	12	0
ZombieLoad/ RIDL	51	Physical	12	11
	47	Virtual	12	6 5 0

Meltdown variants: Microarchitectural buffers



Meltdown variants: Microarchitectural buffers





Meltdown take-away

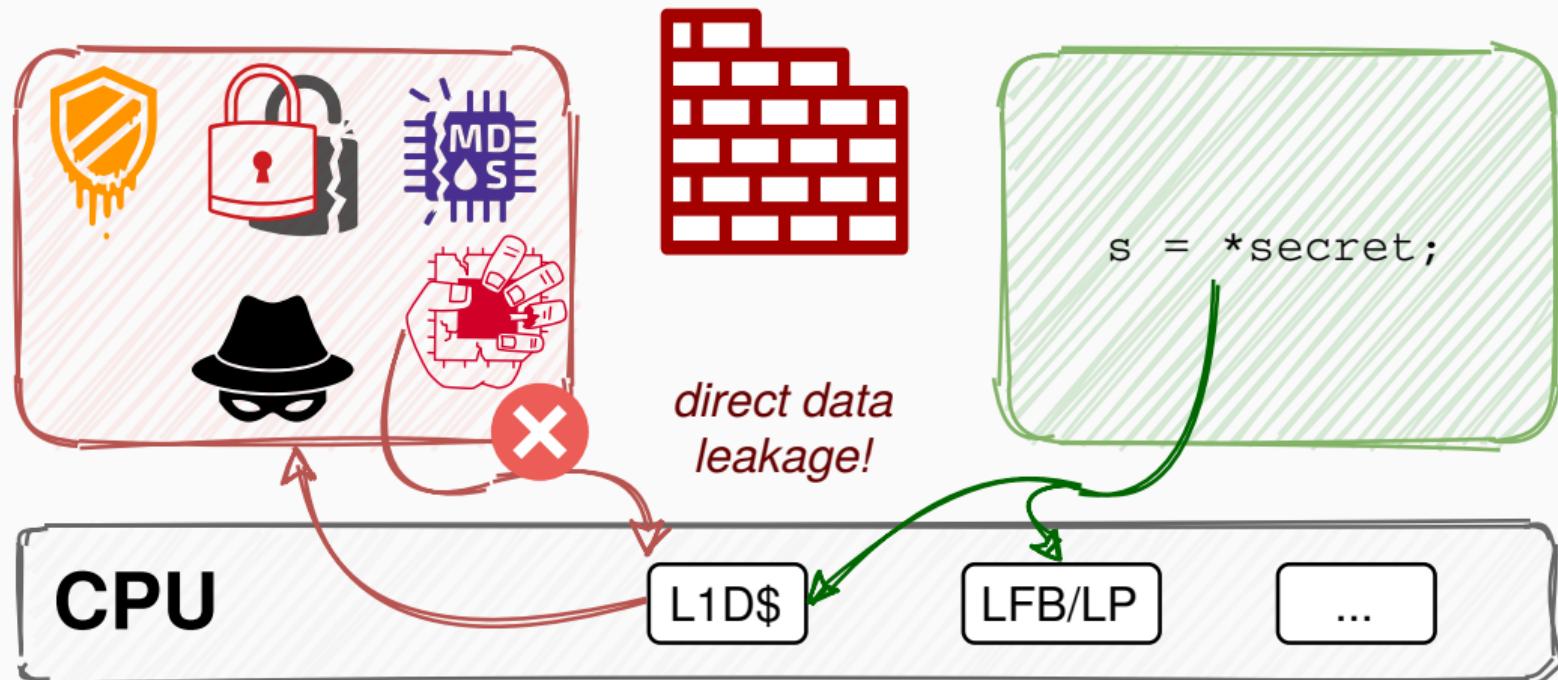
Faulting (or assisted) loads transiently forward **unrelated data** from various microarchitectural buffers



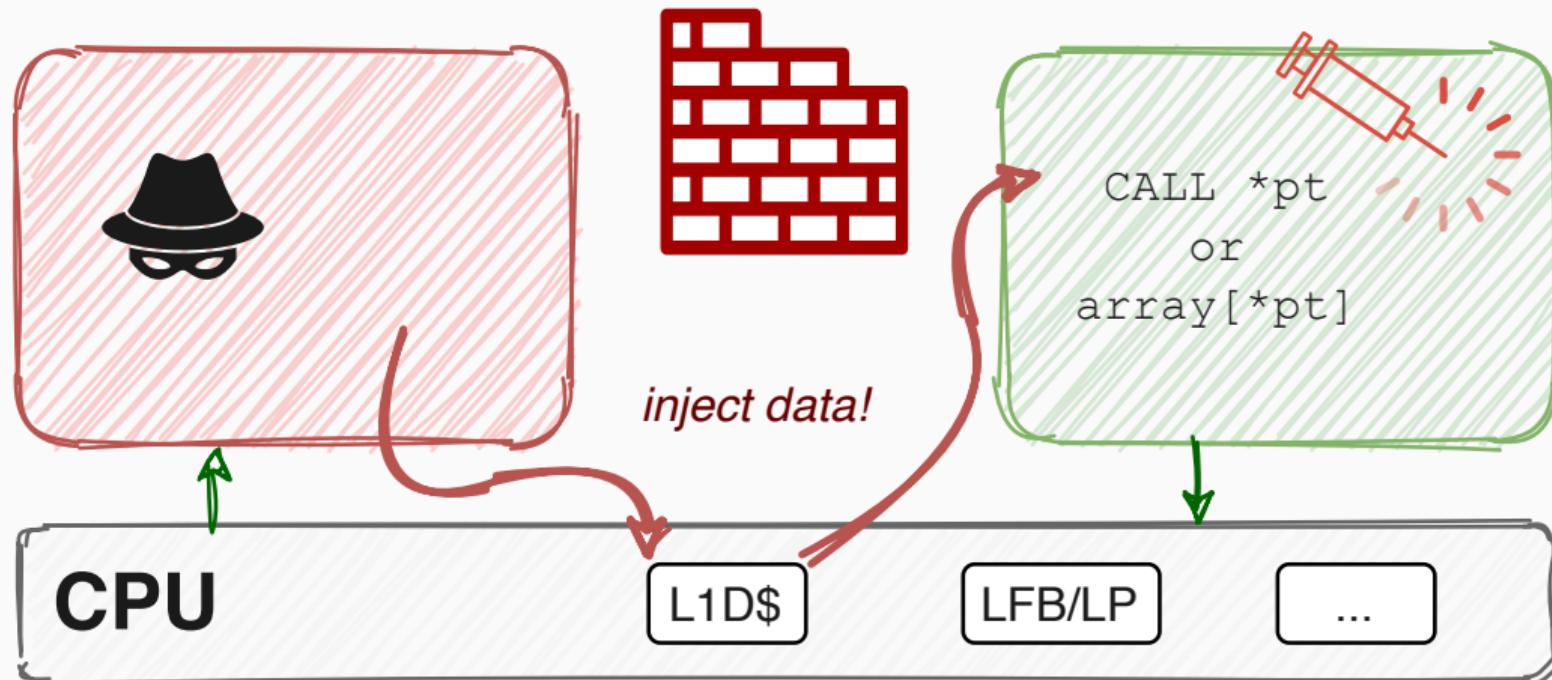


If we take a look at these transient execution attacks we have basically a figure like this (show 4 empty squares - explain and build up the figure except the LVI square)

Load Value Injection (LVI): Turning Meltdown around



Load Value Injection (LVI): Turning Meltdown around





Then you break in and leak it

Then you break in a leak it. Well it's not, strictly speaking, leaking. After all: it's called load value *injection*. (exactly as in the trailer XD)



check in obs that picture looks pretty much the same





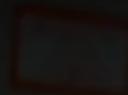
2:08:30



Inception



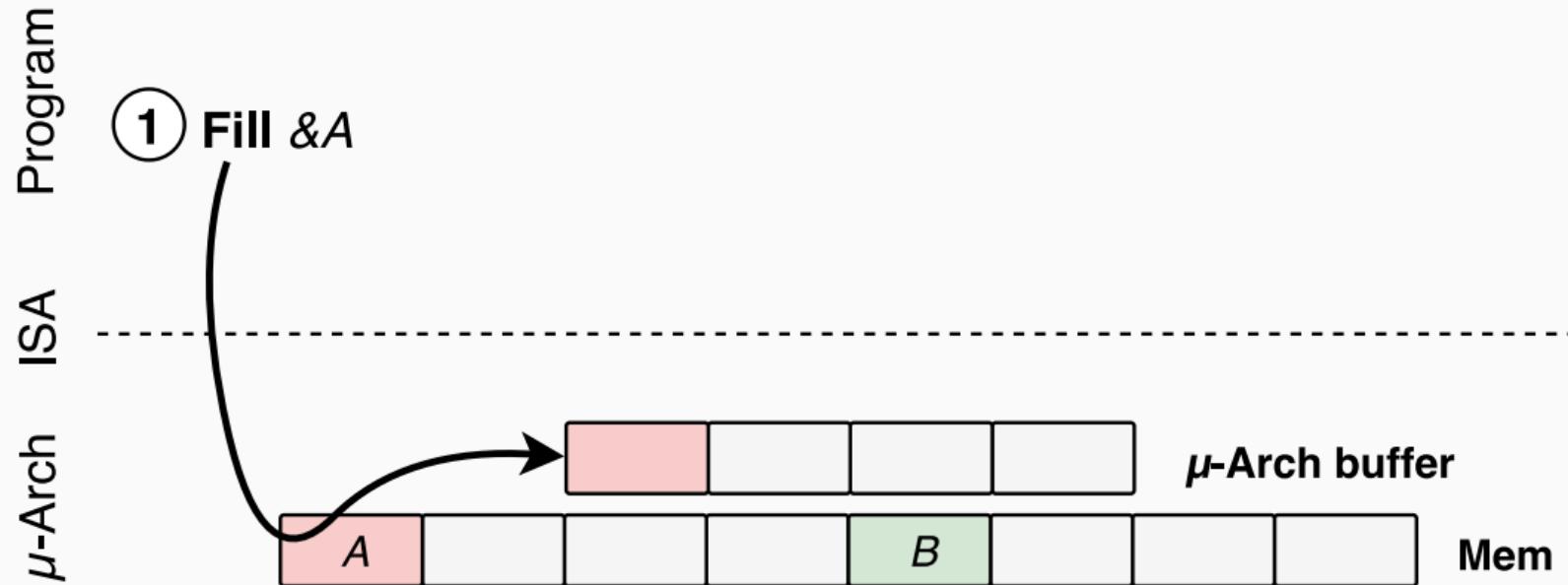
check in obs that picture looks pretty much the same



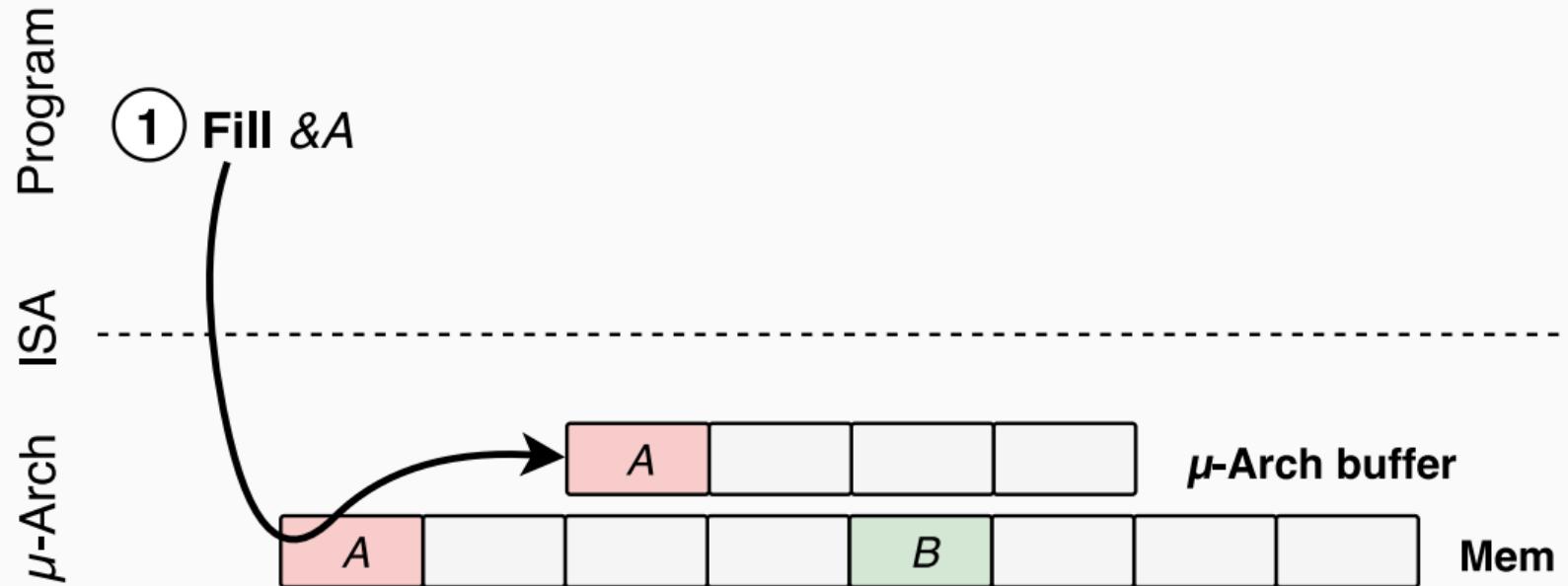
EXIT



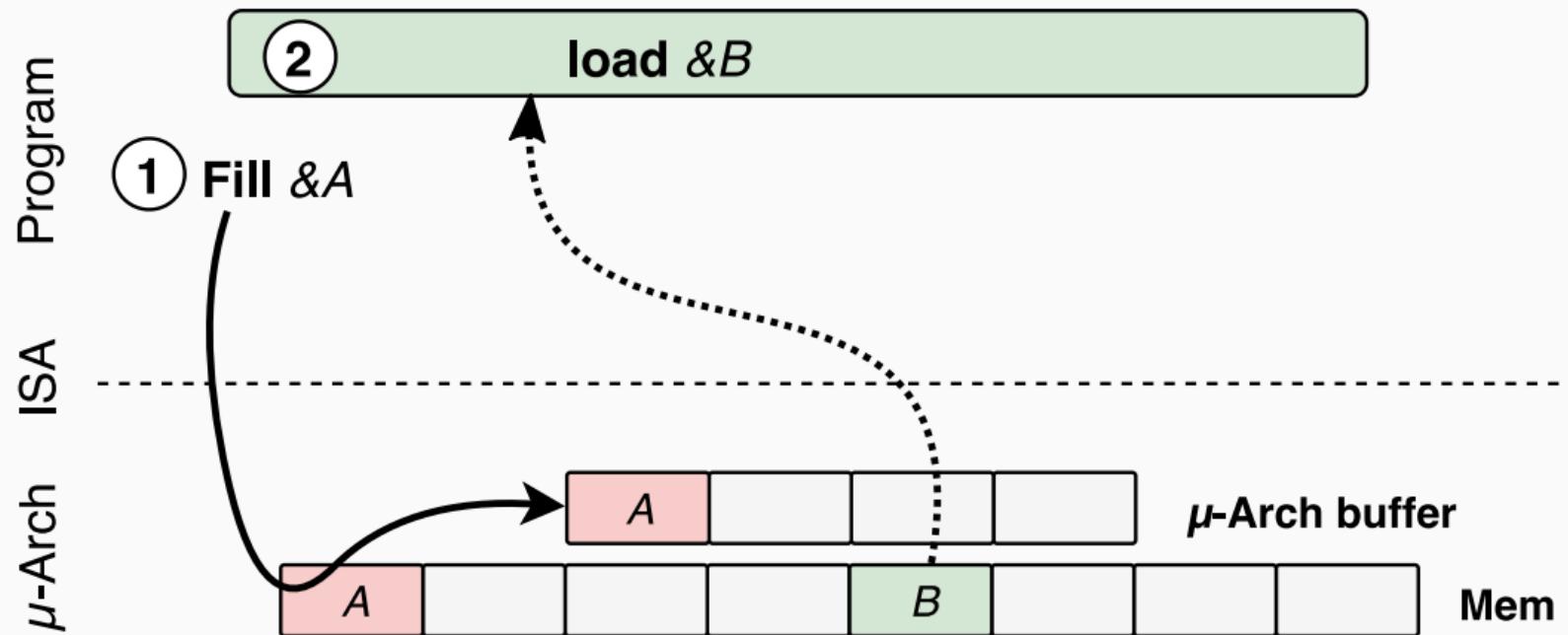
LVI: Microarchitectural inception



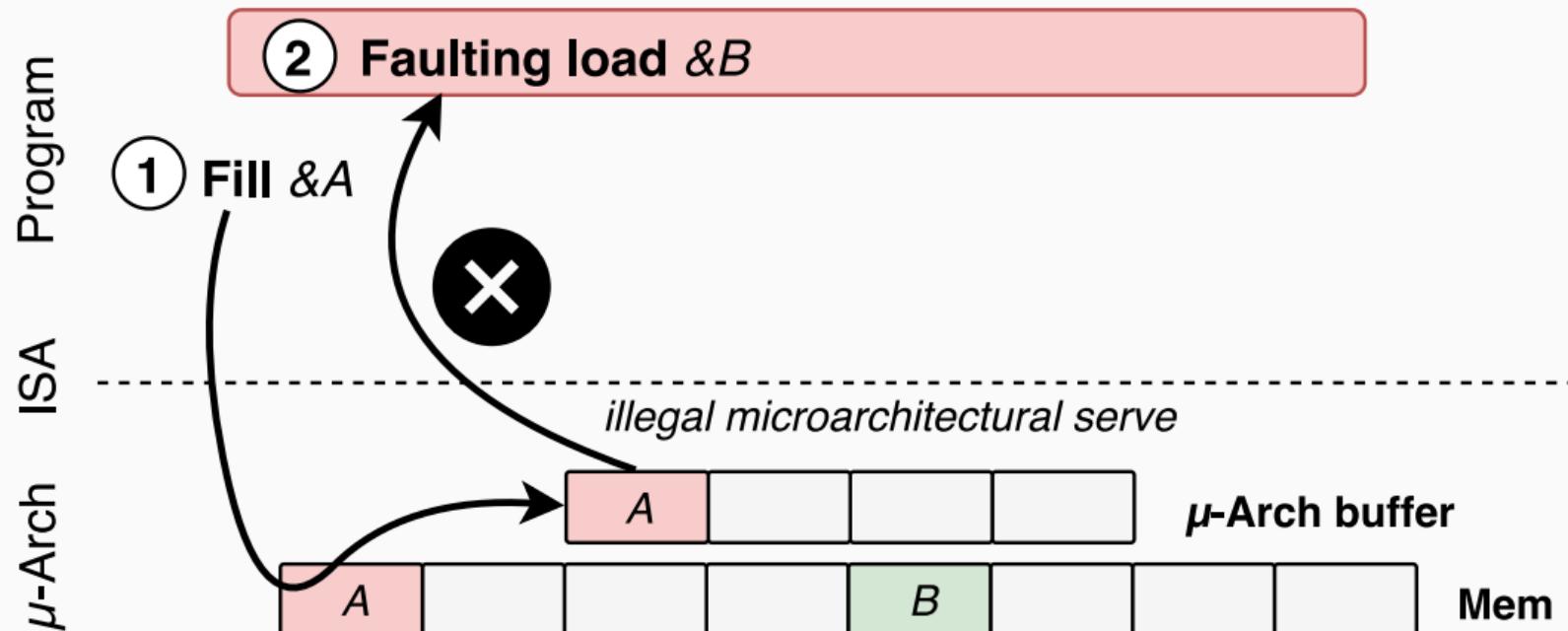
LVI: Microarchitectural inception



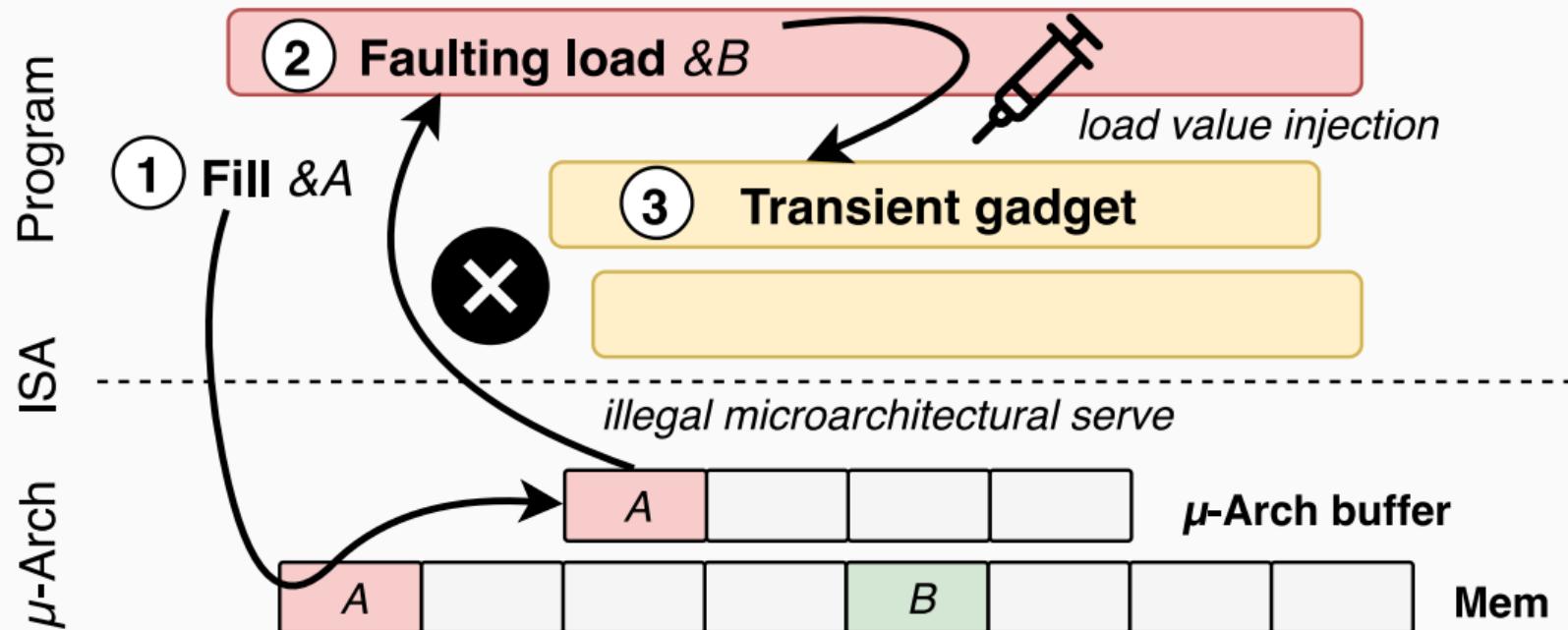
LVI: Microarchitectural inception



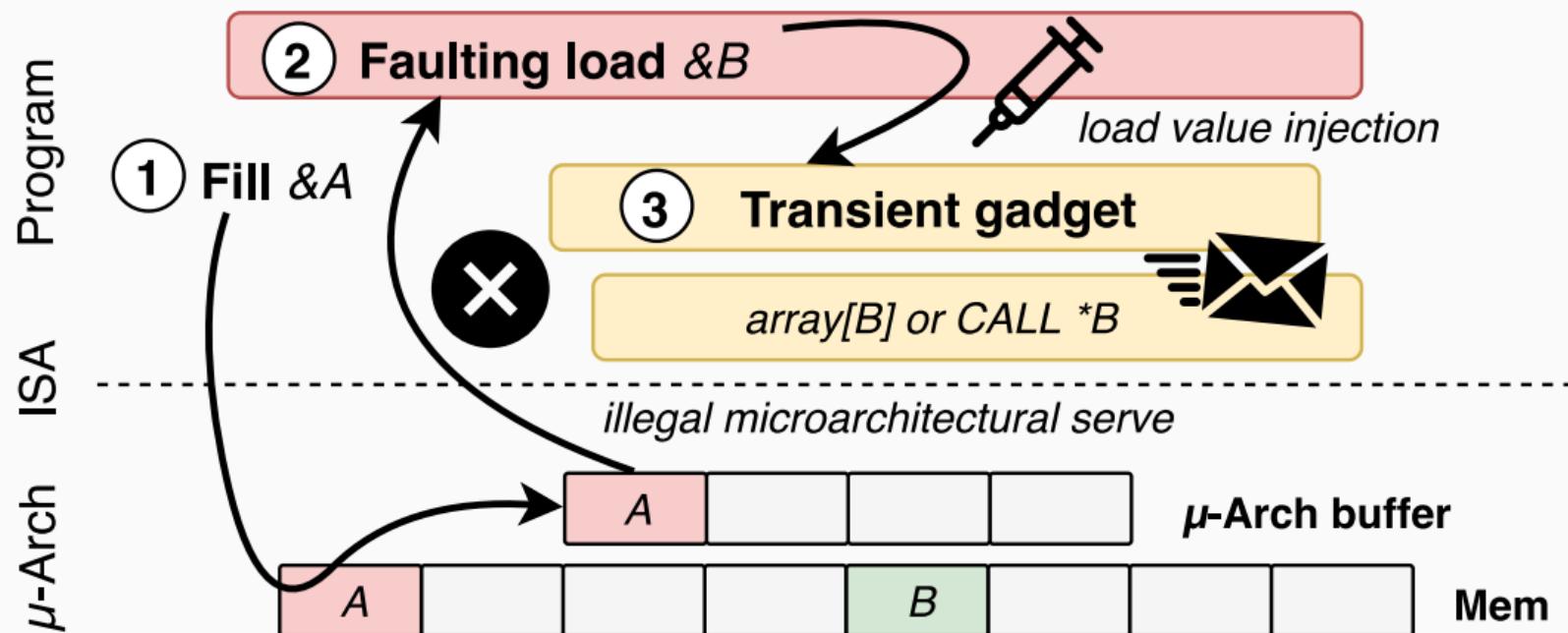
LVI: Microarchitectural inception



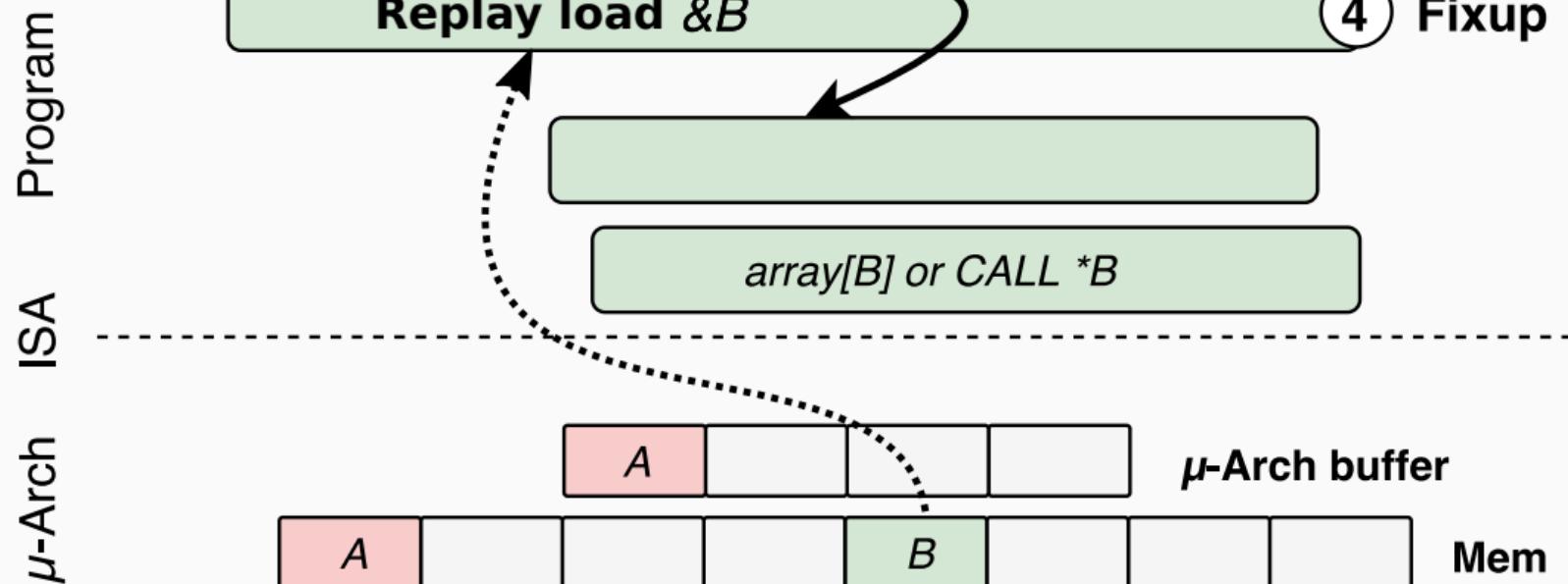
LVI: Microarchitectural inception



LVI: Microarchitectural inception



LVI: Microarchitectural inception





1:52:28



Inception



check in obs that picture looks pretty much the same



Jo: "Load Value Injection" Michael's glass stops halfway to his mouth. Jo: "Don't bother telling me it's impossible." Michael: "It's perfectly possible. Just bloody difficult."



check in obs that picture looks pretty much the same



Jo records himself on this slide



check in obs that picture looks pretty much the same



Michael on this slide

Vulnerable platforms: Intel Software Guard Extensions (SGX)



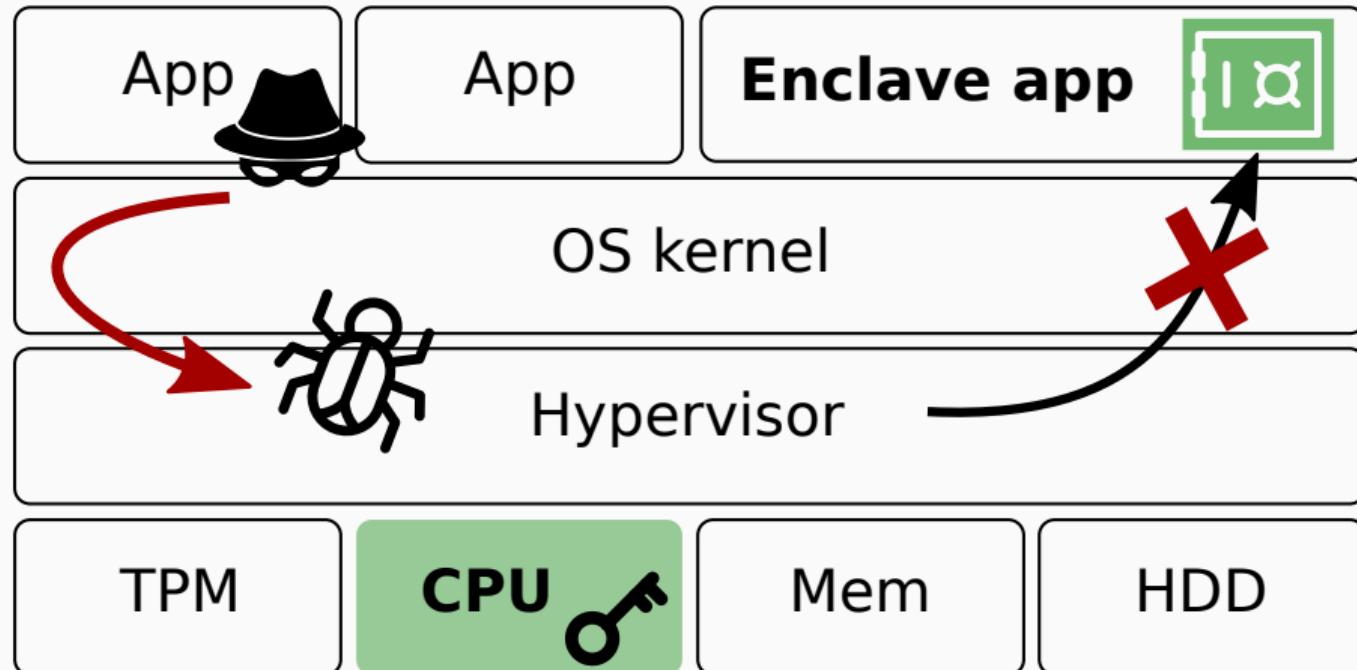
Enarx (Red Hat)



Asylo (Google)



Enclaves to the rescue!



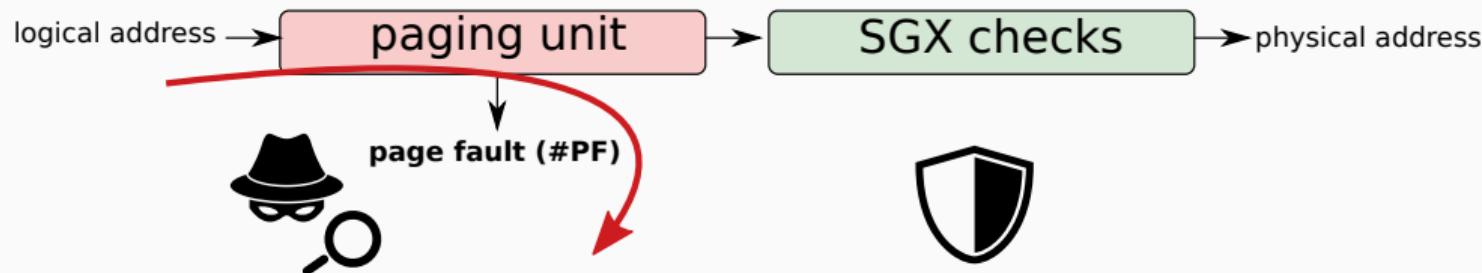
Intel SGX promise: hardware-level **isolation and attestation**

Intel SGX: A look under the hood



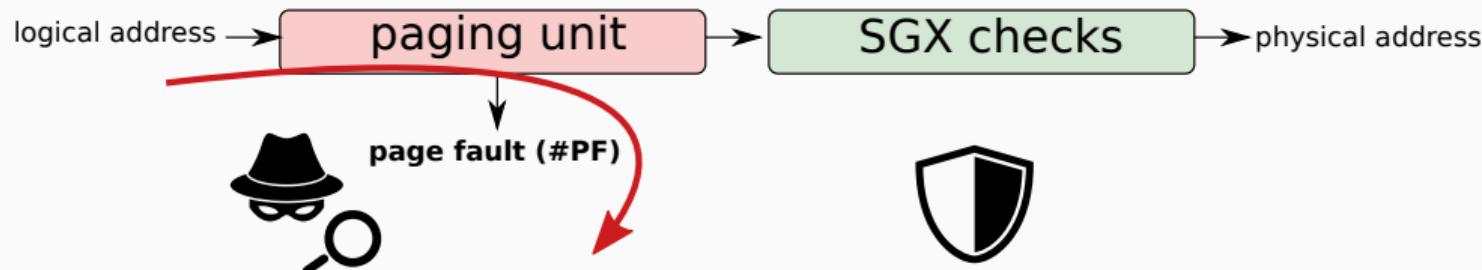
- **SGX machinery** protects against direct address remapping attacks

Intel SGX: A look under the hood



- **SGX machinery** protects against direct address remapping attacks
- ...but untrusted address translation may **fault** during enclaved execution (!)

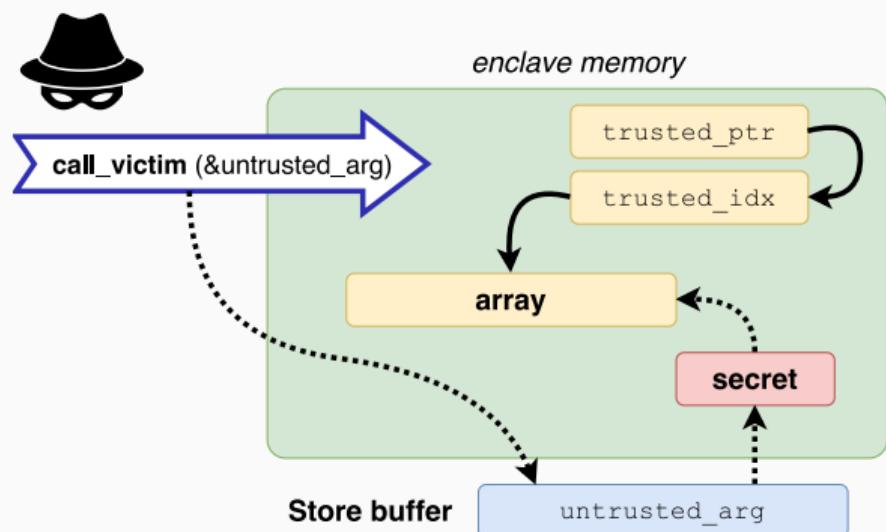
Intel SGX: A look under the hood



We can arbitrarily provoke page faults for trusted enclave loads!

LVI toy example: Hijacking transient data flow

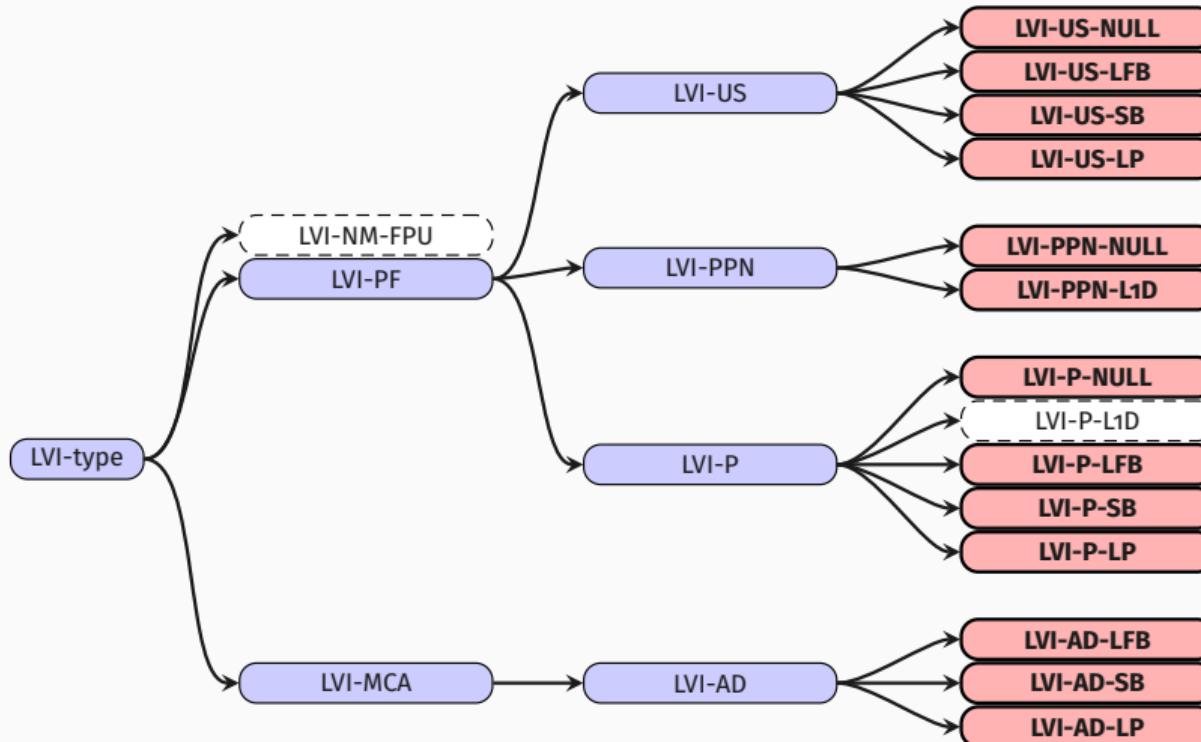
```
1 void call_victim(size_t  
                  untrusted_arg)  
2 {  
3     *arg_copy = untrusted_arg;  
4     array[**trusted_ptr * 4096];  
5 }
```



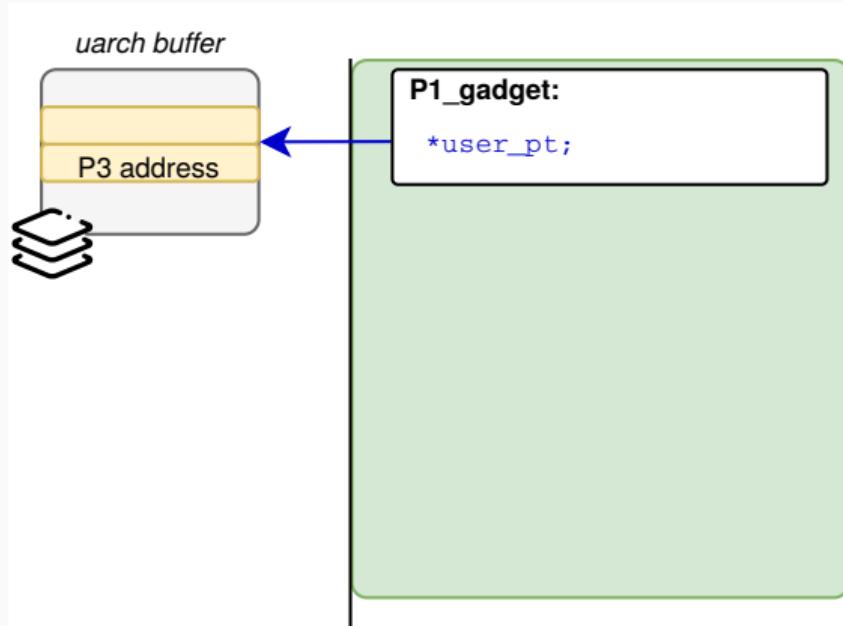
LVI toy example: Recovering arbitrary secrets



Taxonomy of LVI variants: Many buffers, many faults...

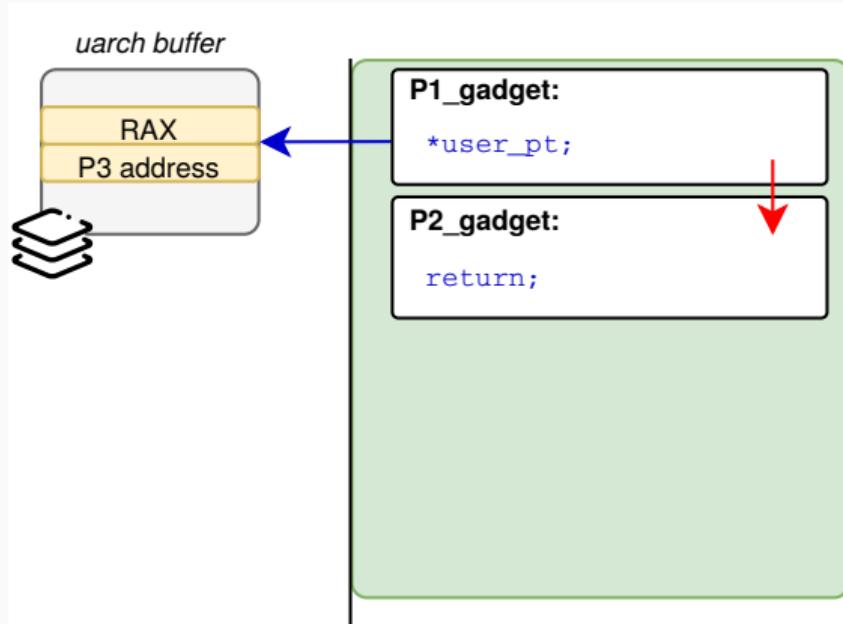


LVI-based transient control-flow hijacking



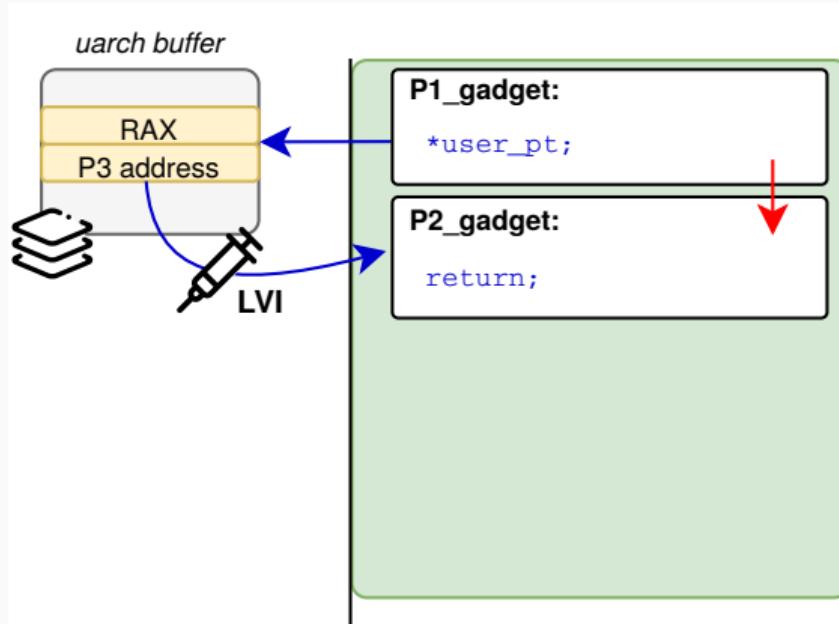
1. Victim **fills μ -arch buffer** with attacker-controlled data

LVI-based transient control-flow hijacking



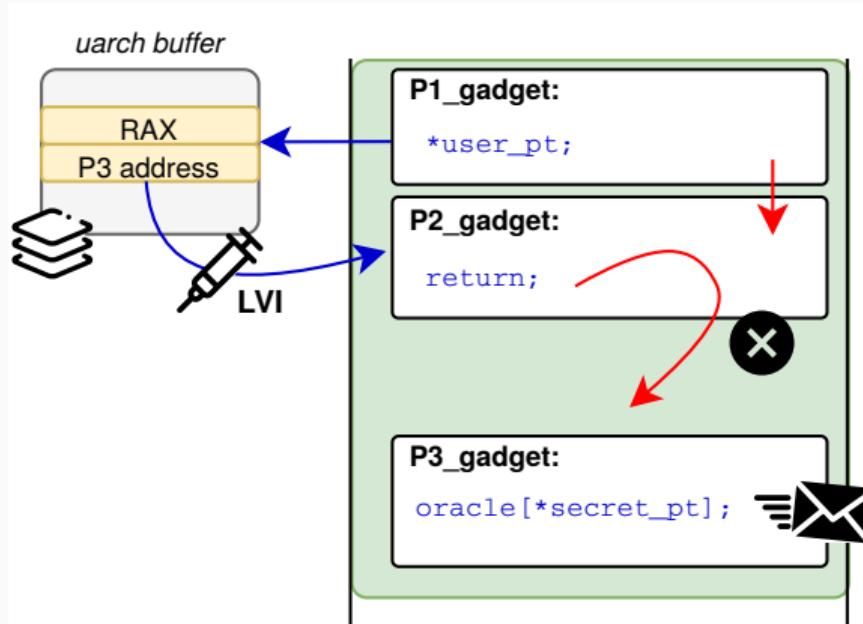
1. Victim **fills μ -arch buffer** with attacker-controlled data
2. Victim executes **indirect branch** (JMP/CALL/RET)

LVI-based transient control-flow hijacking



1. Victim **fills μ -arch buffer** with attacker-controlled data
2. Victim executes indirect branch (JMP/CALL/RET)
3. **Faulting load** → inject incorrect attacker values(!)

LVI-based transient control-flow hijacking



1. Victim **fills μ -arch buffer** with attacker-controlled data
2. Victim executes **indirect branch** (JMP/CALL/RET)
3. **Faulting load** → inject incorrect attacker values(!)
4. Redirect **transient control flow**



```
E/asm.S _main.c
28     .global ecall_lvi_sb_rop
29     # %rdi store_pt
30     # %rsi oracle_pt
31 ecall_lvi_sb_rop:
32     mov %rsp, rsp_backup(%rip)
33     lea page_b(%rip), %rsp
34     add $OFFSET, %rsp
35
36     /* transient delay */
37     clflush dummy(%rip)
38     mov dummy(%rip), %rax
39
40     /* STORE TO USER ADRS */
41     movq $R', (%rdi)
42     lea ret_gadget(%rip), %rax
43     movq %rax, 8(%rdi)
44
45     /* HIJACK TRUSTED LOAD FROM ENCLAVE STACK */
46     /* should go to do_real_ret; will transiently go to ret_gadget if we fault on the stack loads */
47     pop %rax
48 #if LFENCE
49     notq (%rsp)
50     notq (%rsp)
51     lfence
52     ret
53 #else
54     ret
55 #endif
56
57 1:  jmp 1b
58     mfence
59
60 do_real_ret:
61     mov rsp_backup(%rip), %rsp
62     ret
63
```

Real-world LVI-ROP gadget in Intel SGX SDK

```
1 ; %rbx: user-controlled argument ptr (outside enclave)
2 sgx_my_sum_bridge:
3 ...
4 call my_sum           ; compute 0x10(%rbx) + 0x8(%rbx)
5 mov %rax,(%rbx)       ; P1: store sum to user address
6 xor %eax,%eax
7 pop %rbx
8 ret                   ; P2: load from trusted stack
9
```

Real-world LVI-ROP gadget in Intel SGX SDK

```
1 ; %rbx: user-controlled argument ptr (outside enclave)
2 sgx_my_sum_bridge:
3 ...
4 call my_sum           ; compute 0x10(%rbx) + 0x8(%rbx)
5 mov %rax,(%rbx)       ; P1: store sum to user address
6 xor %eax,%eax
7 pop %rbx
8 ret                   ; P2: load from trusted stack
9
```



We can setup a fake transient stack in the store buffer or L1D!

Real-world LVI-ROP gadget in Quoting Enclave

```
1 __intel_avx_rep_memcpy: ; libirc_2.4/efi2/libirc.a
2   ...
3   vmovups %xmm0,-0x10(%rdi,%rcx,1)
4   ...
5   pop    %r12          ; P2: load from trusted stack
6   ret
7
```



56:40



Inception



check in obs that picture looks pretty much the same



Daniel: "Instead of actual data values, it should be safe to just return 0 values, right?" Jo: "I wouldn't do that if I were a CPU manufacturer. I believe an attacker could"



56:34



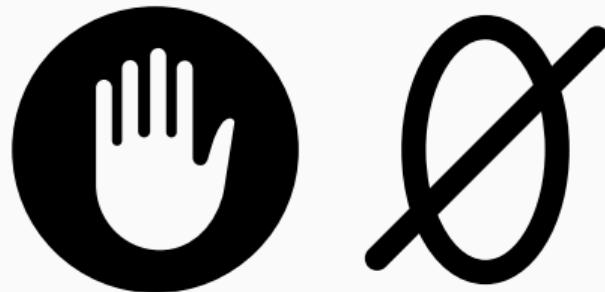
Inception



check in obs that picture looks pretty much the same

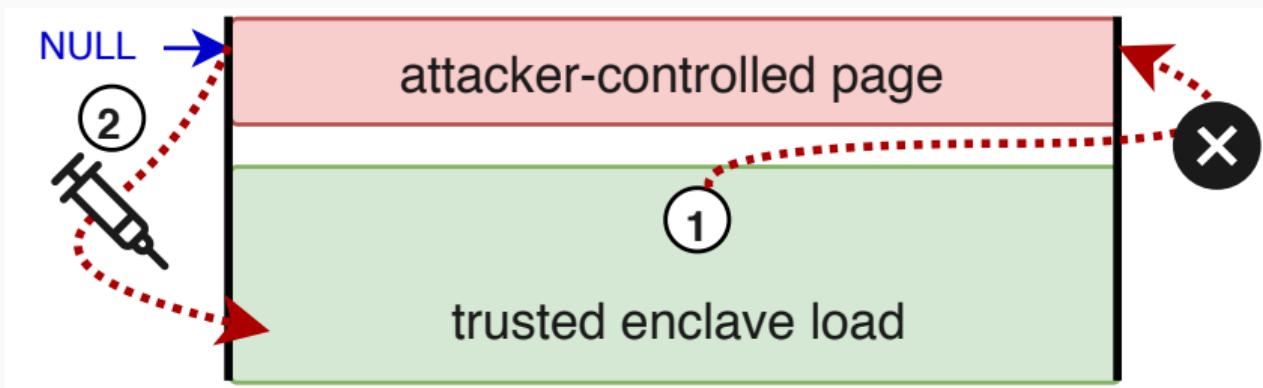


LVI-NUL: Why `0x00` is *not* a safe value



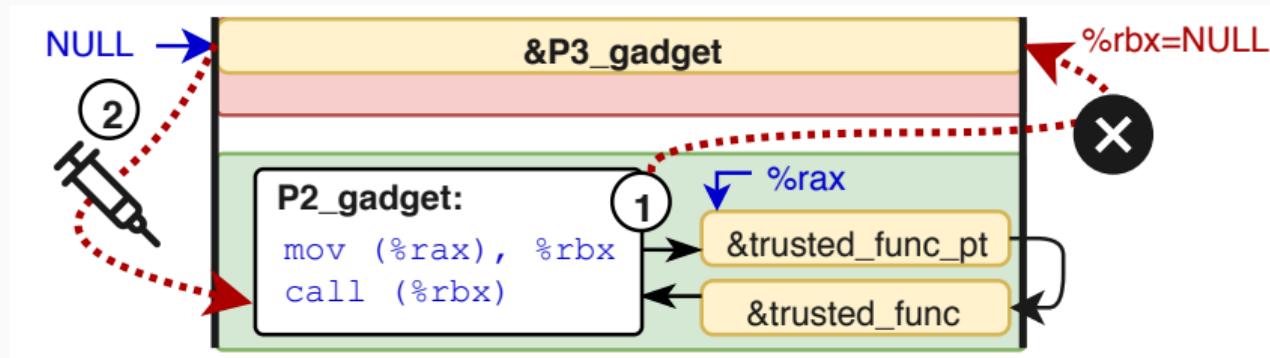
- Recent Intel CPUs forward **0x00 dummy values** for faulting loads

LVI-NULL: Why `0x00` is *not* a safe value



- Recent Intel CPUs forward **`0x00` dummy values** for faulting loads
- ...but NULL is a **valid virtual memory address**, under attacker control

LVI-NULL: Why `0x00` is *not* a safe value



- Recent Intel CPUs forward **0x00 dummy values** for faulting loads
- ...but NULL is a **valid virtual memory address**, under attacker control
- ...hijack **pointer values** (e.g., function pointer-to-pointer)

Real-world LVI-NUL stack hijacking gadget

```
1 asm_oret: ; (linux-sgx/sdk/trts/linux/trts_pic.S#L454)
2 ...
3 mov    0x58(%rsp),%rbp          ; %rbp <- NULL
4 ...
5 mov    %rbp,%rsp              ; %rsp <- NULL
6 pop    %rbp                  ; %rbp <- *(NULL)
7 ret                ; %rip <- *(NULL+8)
8
```



1:43:25



Inception



check in obs that picture looks pretty much the same



Michael on this slide! Michael: "What's the mitigation against LVI?" Jo: "Vendors might say that's none of our concern" Michael: "Yeah, but this isn't the usual sid



check in obs that picture looks pretty much the same



jo on this slide



1:42:45

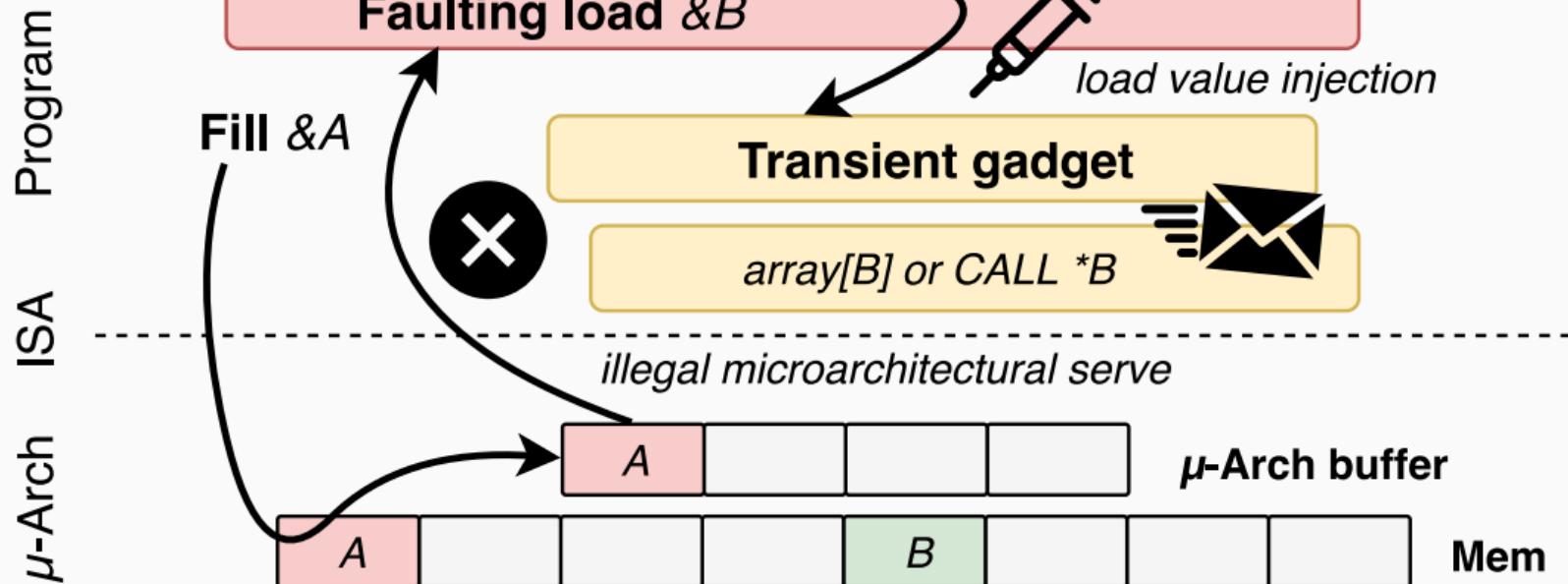


Inception

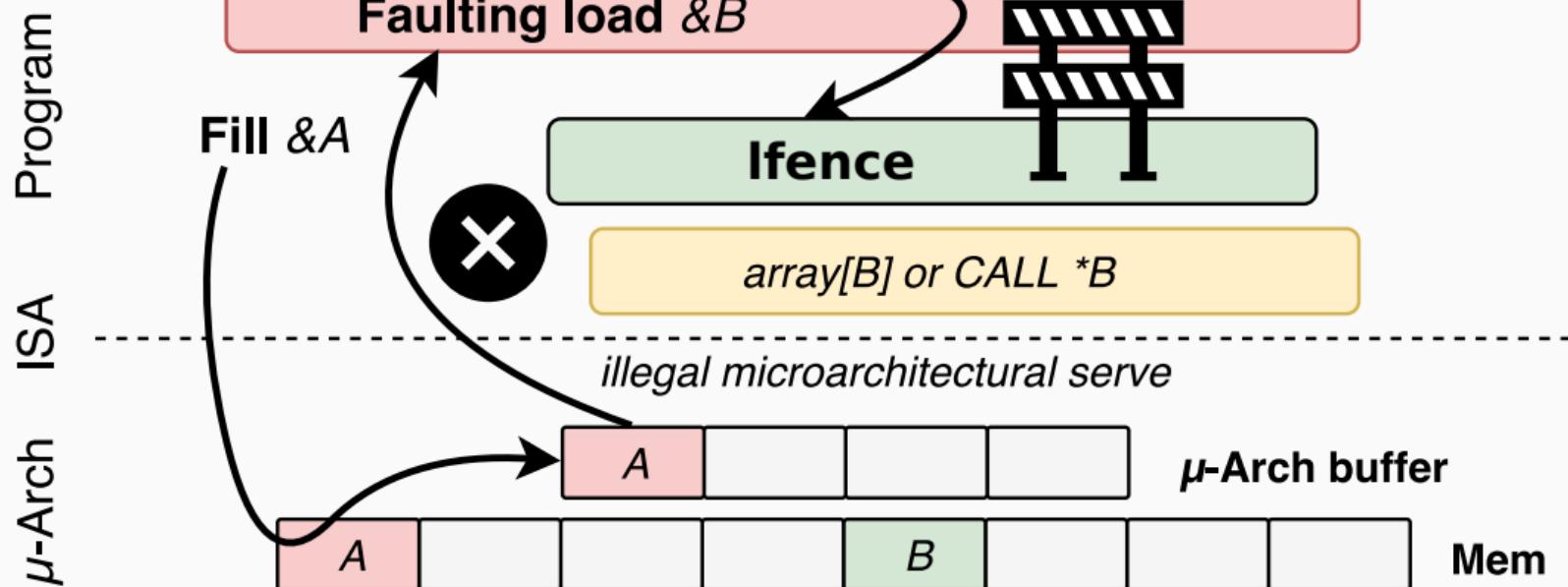


check in obs that picture looks pretty much the same

Mitigation idea: Fencing vulnerable load instructions



Mitigation idea: Fencing vulnerable load instructions



Mitigating LVI: Compiler and assembler support



-mlfence-after-load

GNU Assembler Adds New Options For Mitigating Load Value Injection Attack

Written by Michael Larabel in [GNU](#) on 11 March 2020 at 02:55 PM EDT. [14 Comments](#)



-mlvi-hardening

LLVM Lands Performance-Hitting Mitigation For Intel LVI Vulnerability

Written by Michael Larabel in [Software](#) on 3 April 2020. [Page 1 of 3.](#) [20 Comments](#)



-Qspectre-load

More Spectre Mitigations in [MSVC](#)

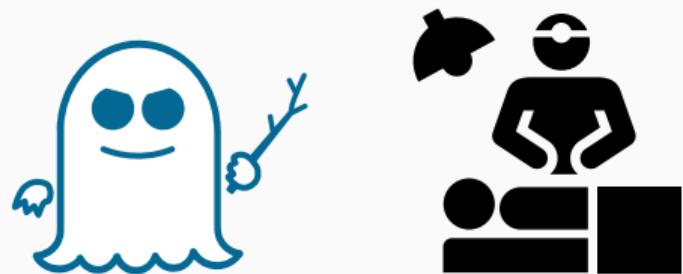
March 13th, 2020

Serializing indirect branches



LVI ↔ Spectre: no control-flow prediction; **every load can be hijacked**

Instruction	Possible emulation	Clobber-free
ret	pop %reg; lfence ; jmp *%reg	X
ret	not (%rsp); not (%rsp); lfence ; ret	✓
jmp (mem)	mov (mem),%reg; lfence ; jmp *%reg	X
call (mem)	mov (mem),%reg; lfence ; call *%reg	X



23 fences

October 2019—“surgical precision”



23 fences

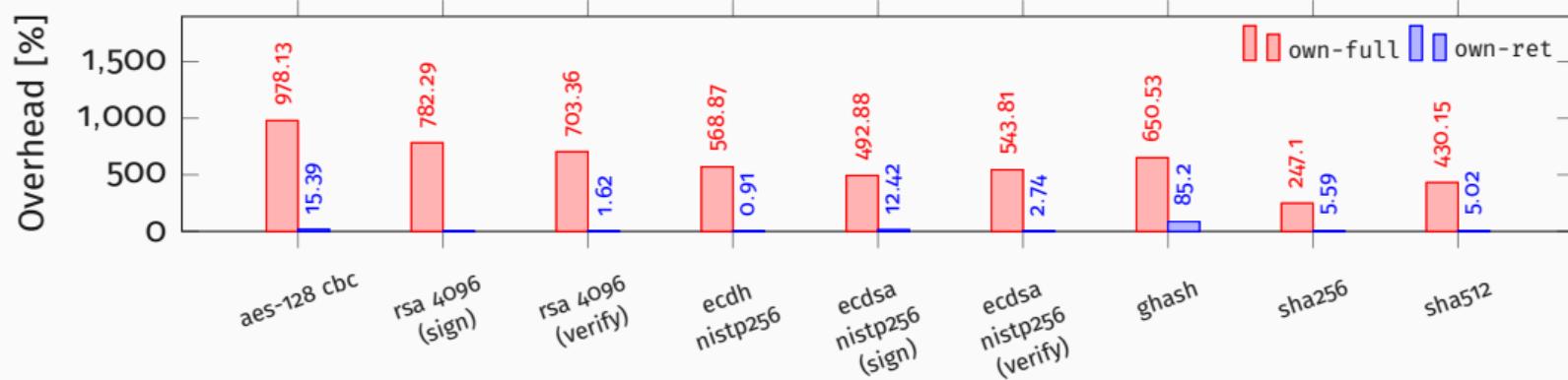
October 2019—“surgical precision”

49,315 fences

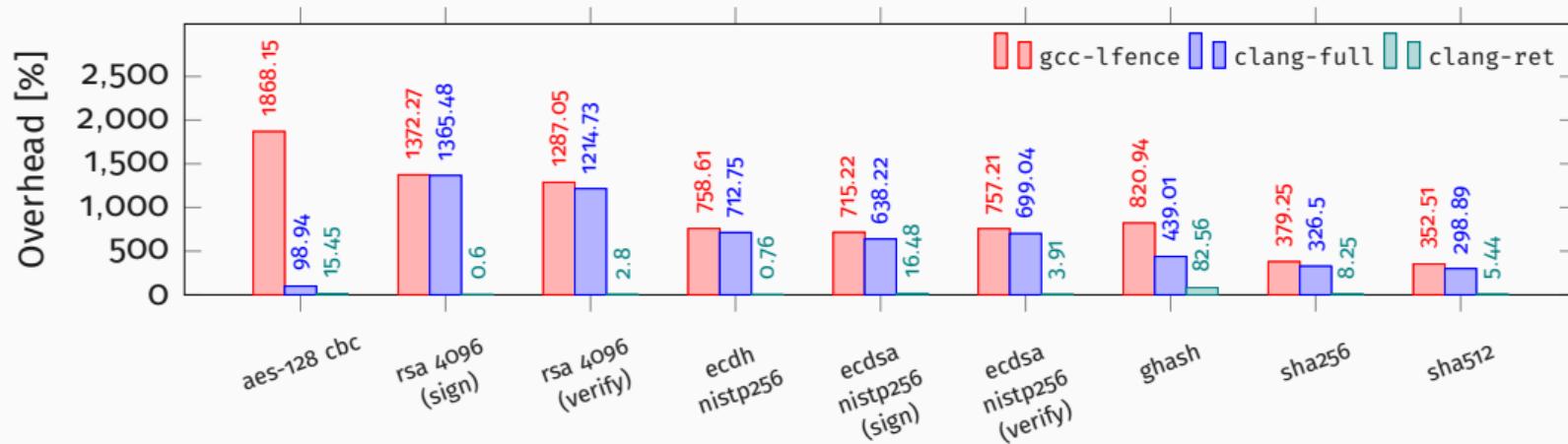
March 2020—“big hammer”



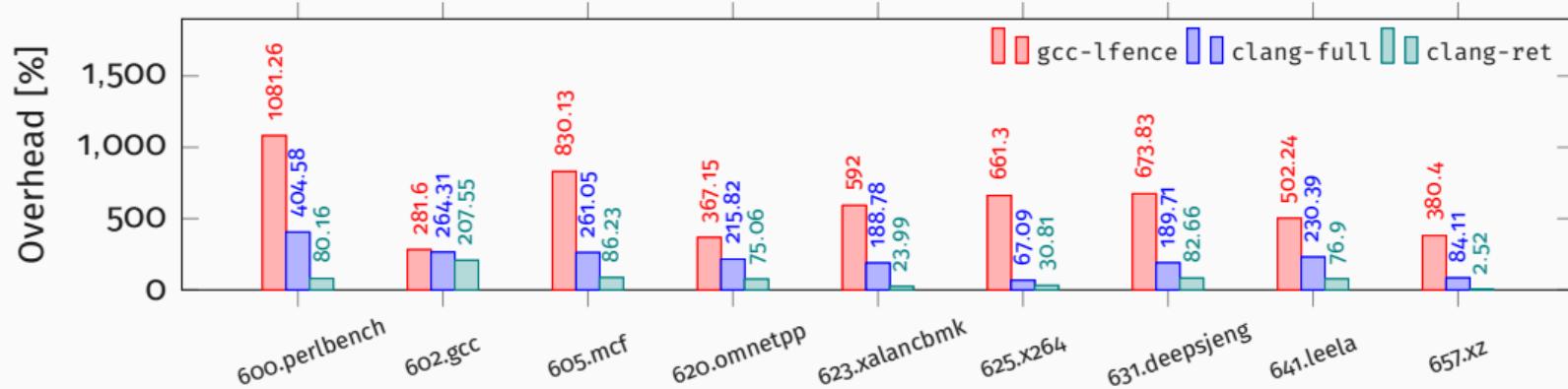
Performance overheads: OpenSSL (our prototype mitigation)



Performance overheads: OpenSSL (Intel's mitigation)



Performance overheads: SPEC (Intel's mitigation)



GNU Assembler Adds New Options For Mitigating Load Value Injection Attack

Written by Michael Larabel in [GNU](#) on 11 March 2020 at 02:55 PM EDT. [14 Comments](#)

The Brutal Performance Impact From Mitigating The LVI Vulnerability

Written by Michael Larabel in [Software](#) on 12 March 2020. [Page 1 of 6.](#) [76 Comments](#)

LLVM Lands Performance-Hitting Mitigation For Intel LVI Vulnerability

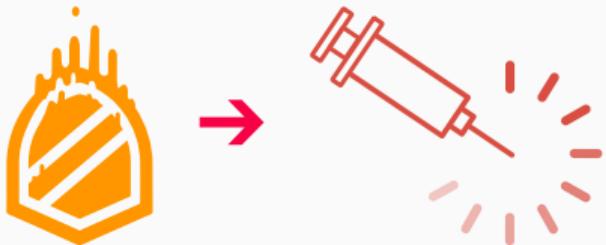
Written by Michael Larabel in [Software](#) on 3 April 2020. [Page 1 of 3.](#) [20 Comments](#)

Looking At The LVI Mitigation Impact On Intel Cascade Lake Refresh

Written by Michael Larabel in [Software](#) on 5 April 2020. [Page 1 of 5.](#) [10 Comments](#)

- ⇒ **LVI gadgets** reversely exploit Meltdown-type effects
- ⇒ **Short-term:** extensive `lfence` compiler mitigations for Intel SGX enclaves
- ⇒ **Long-term:** improved `silicon patches` in new CPUs





Microarchitectural Inception

Jo Van Bulck, Michael Schwarz, Daniel Gruss, Moritz Lipp

rC3 — Remote Chaos Experience, December 2020

