



Microarchitectural Side-Channel Attacks

for Privileged Software Adversaries

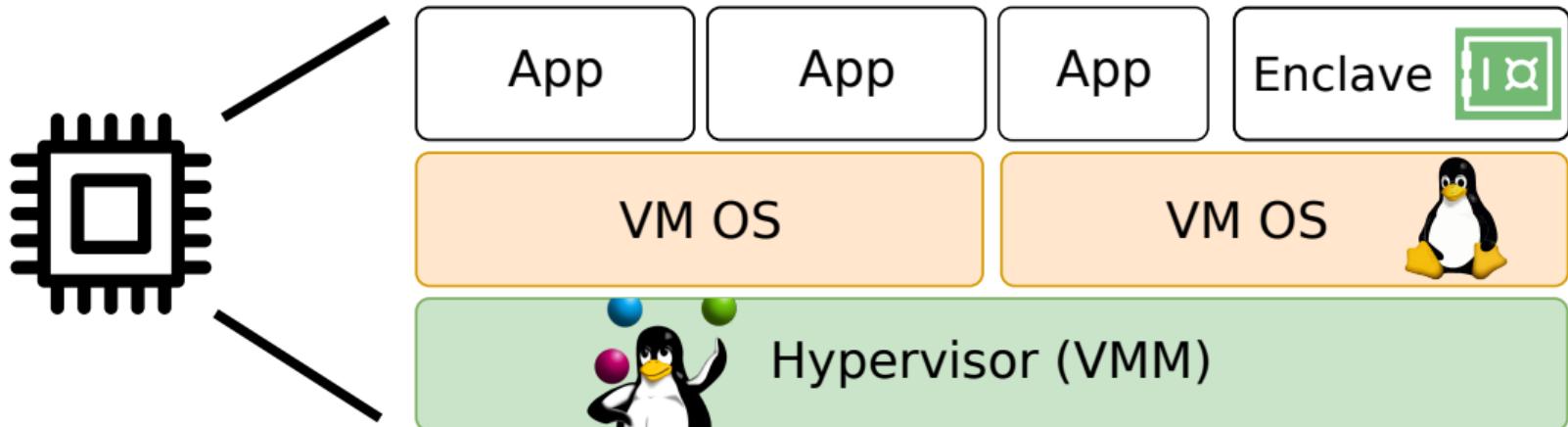
Jo Van Bulck

DistriNet reunion, February 5, 2020

🏡 imec-DistriNet, KU Leuven 📩 jo.vanbulck@cs.kuleuven.be 🐦 [jovanbulck](https://twitter.com/jovanbulck)

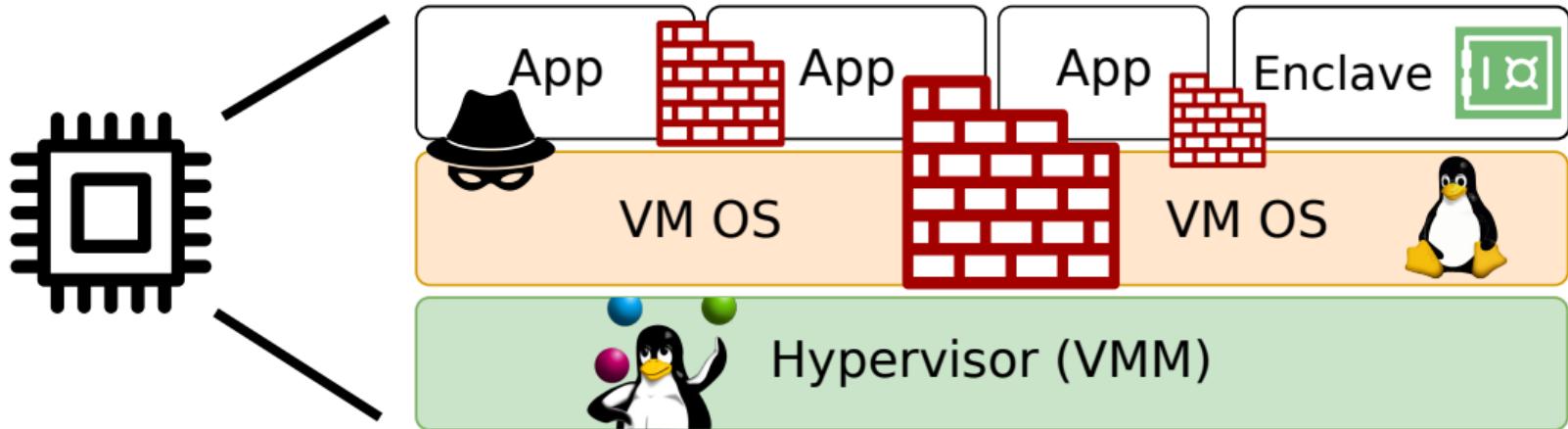


Processor security: Hardware isolation mechanisms



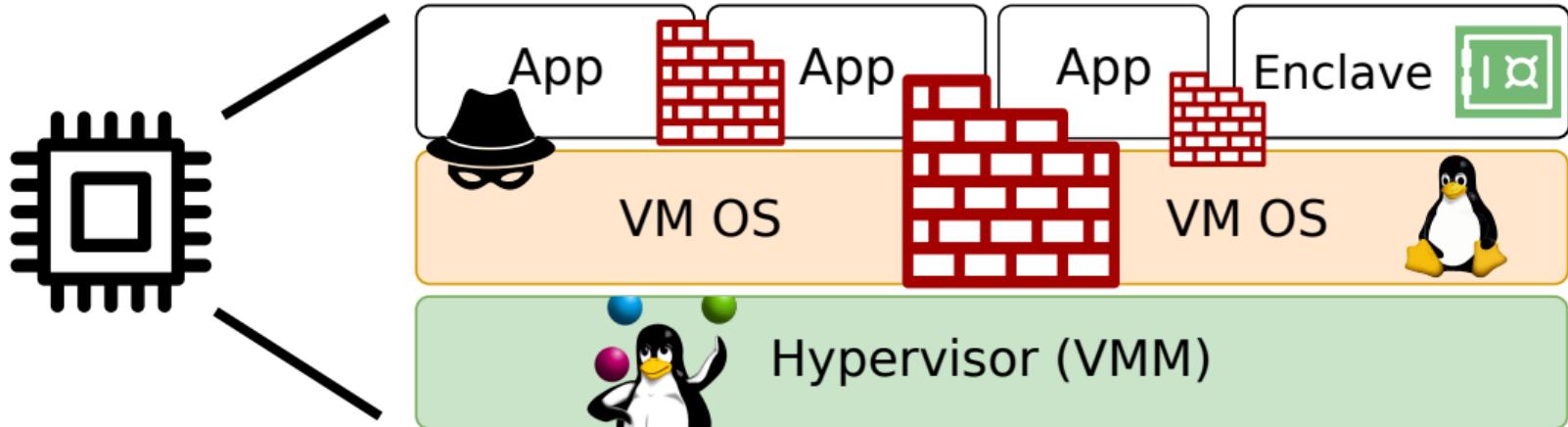
- Different software **protection domains**: user processes, virtual machines, enclaves

Processor security: Hardware isolation mechanisms



- Different software **protection domains**: user processes, virtual machines, enclaves
- CPU builds “walls” for **memory isolation** between applications and privilege levels

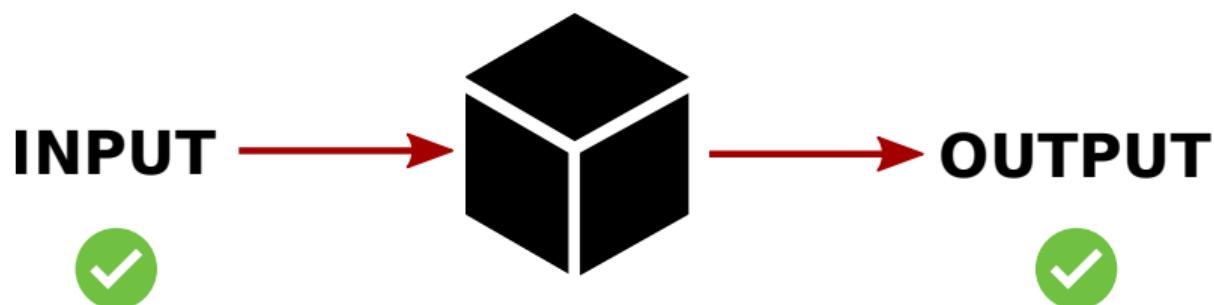
Processor security: Hardware isolation mechanisms



- Different software **protection domains**: user processes, virtual machines, enclaves
- CPU builds “walls” for **memory isolation** between applications and privilege levels
 - ↔ Architectural protection walls permeate **microarchitectural side-channels!**

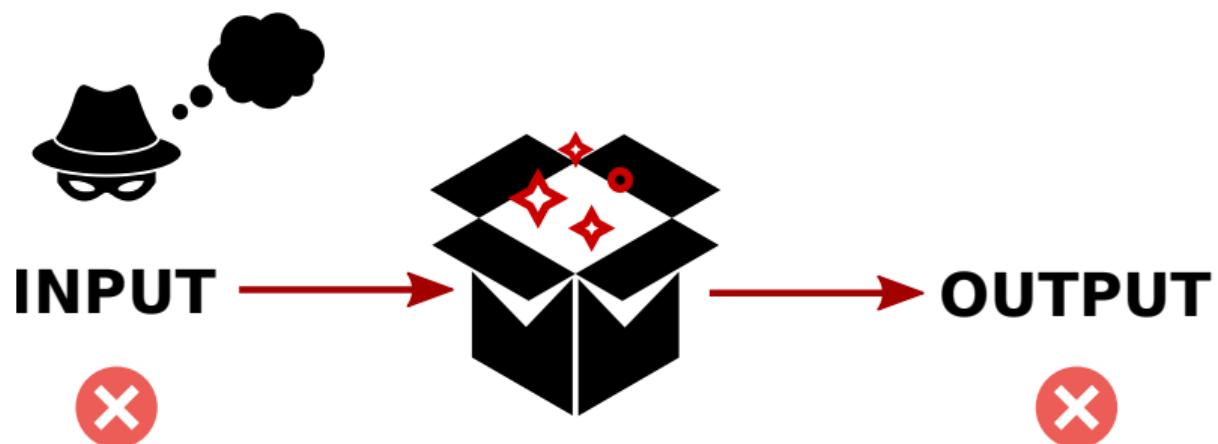
A primer on software security

Secure program: convert all input to *expected output*



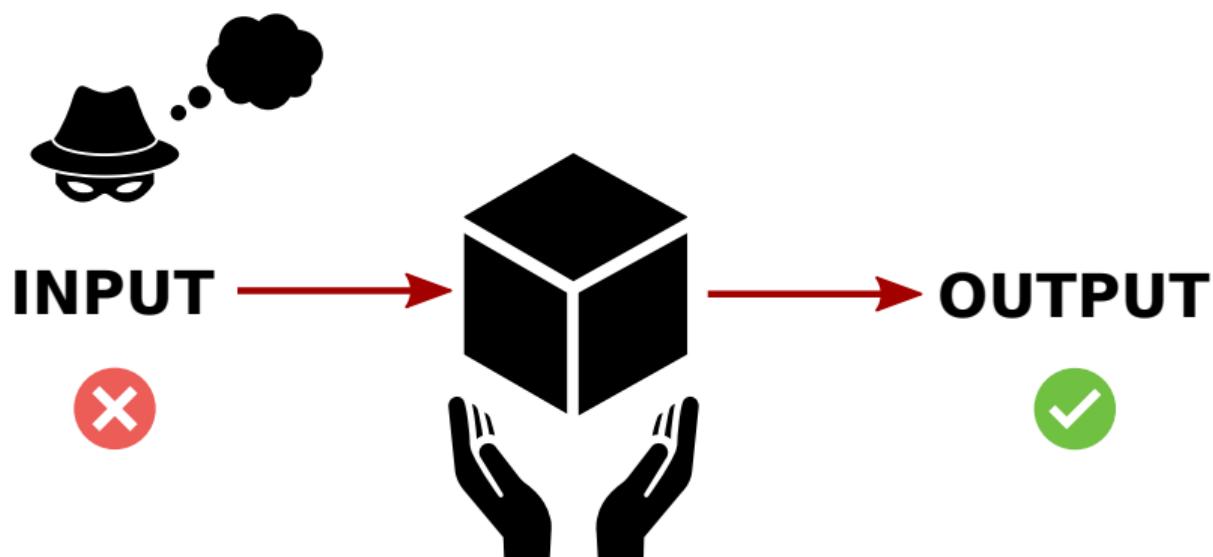
A primer on software security

Buffer overflow vulnerabilities: trigger *unexpected behavior*



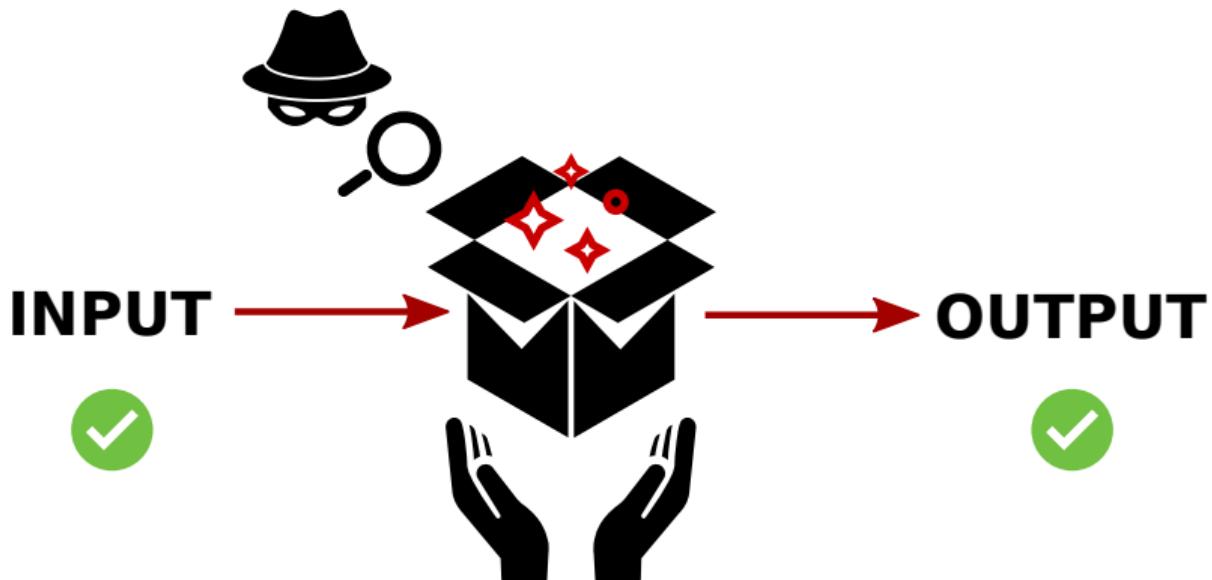
A primer on software security

Safe languages & formal verification: preserve expected behavior



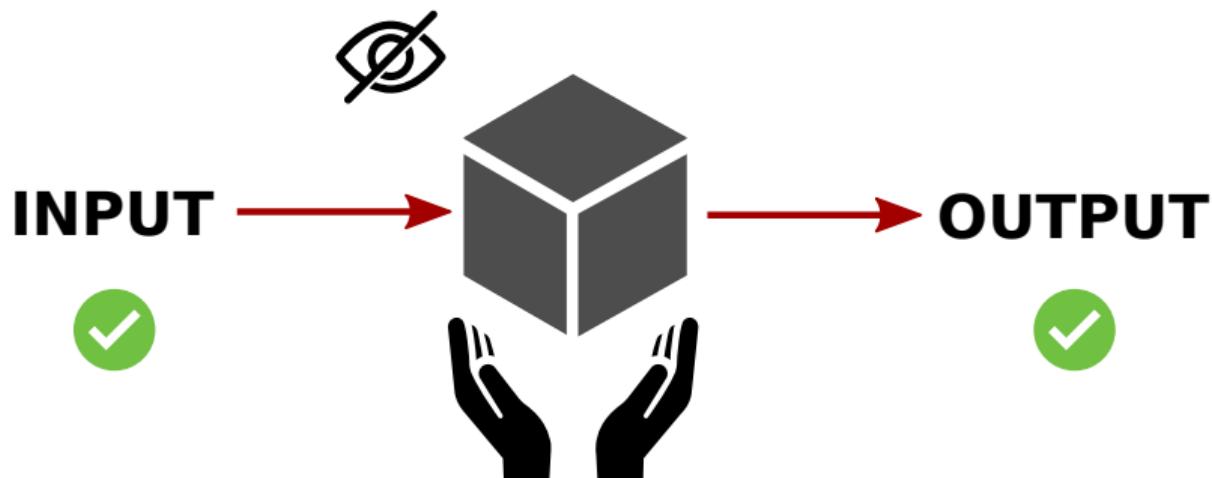
A primer on software security

Side-channels: observe *side-effects* of the computation



A primer on software security

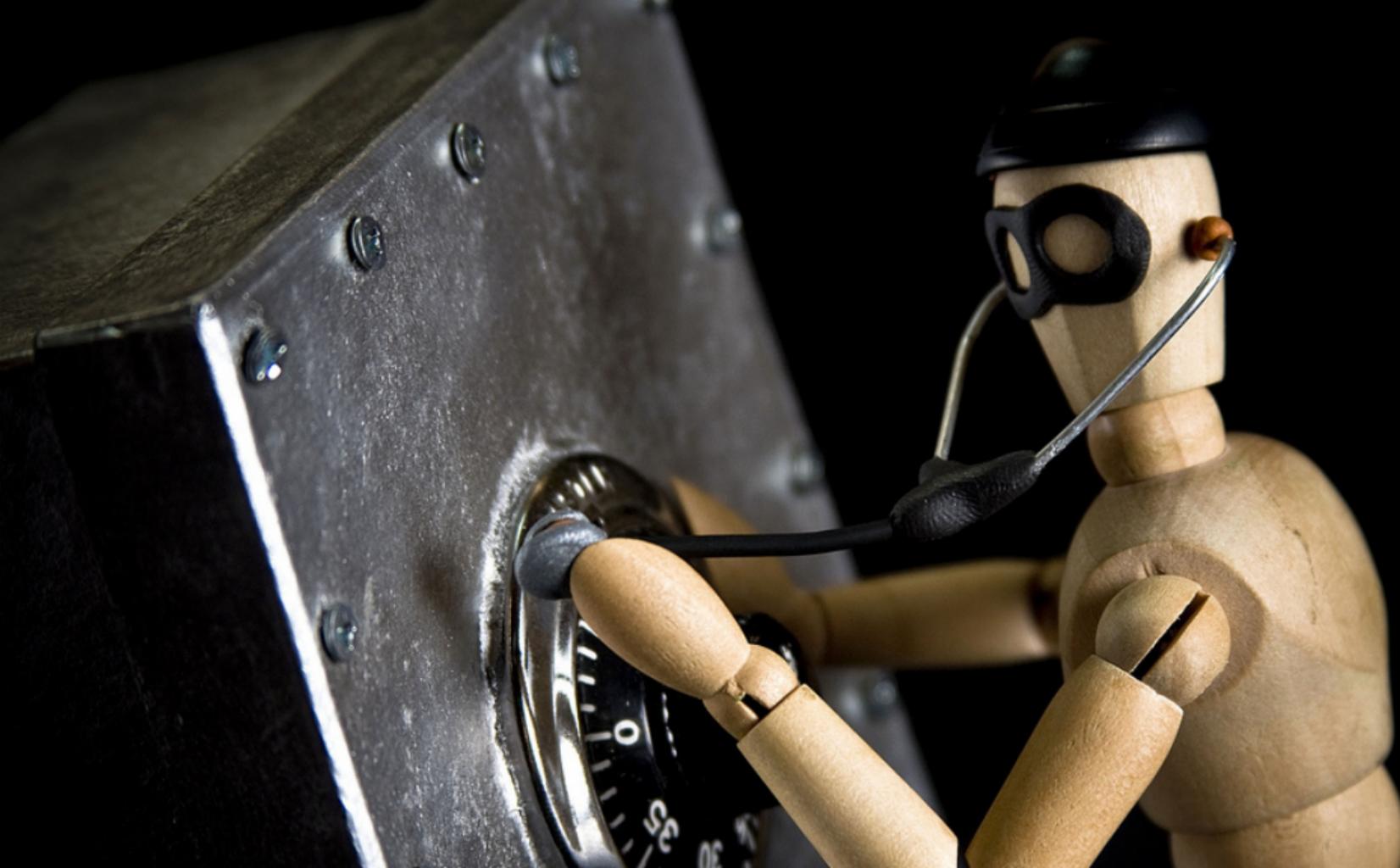
Constant-time code: eliminate *secret-dependent* side-effects





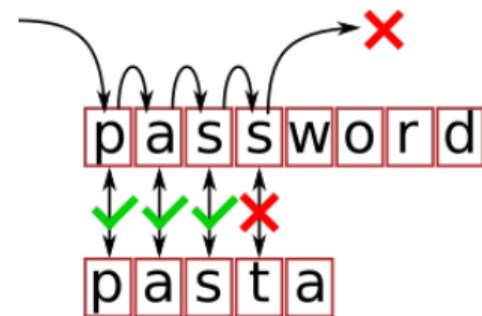
VAULT DOOR

WEIGHT: 22 1/2 Tons
THICKNESS: 22 Inches
STEEL: 11 Layers of Special
Cutting and Drill Resistant
LOCKS: 4 Hamilton Watch
Movements for Time Locks



A vulnerable example program and its constant-time equivalent

```
1 void check_pwd( char *input)
2 {
3     for ( int i=0; i < PWD_LEN; i++)
4         if ( input[ i ] != pwd[ i ])
5             return 0;
6
7     return 1;
8 }
```



Overall execution time reveals correctness of individual password bytes!

→ reduce brute-force attack from an exponential to a linear effort...

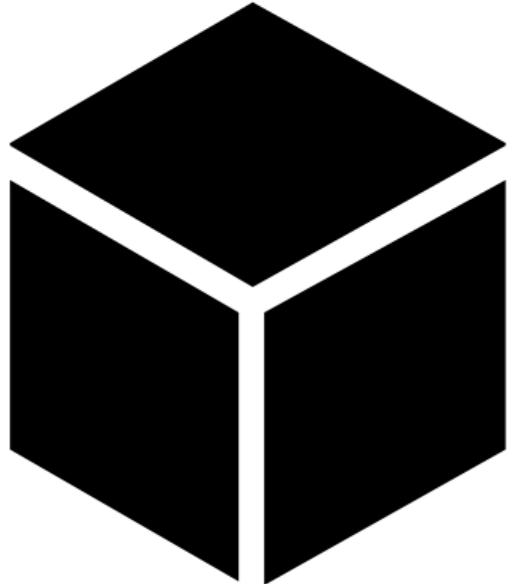
A vulnerable example program and its constant-time equivalent

```
1 void check_pwd(char *input)
2 {
3     for (int i=0; i < PWD_LEN; i++)
4         if (input[i] != pwd[i])
5             return 0;
6
7     return 1;
8 }
```

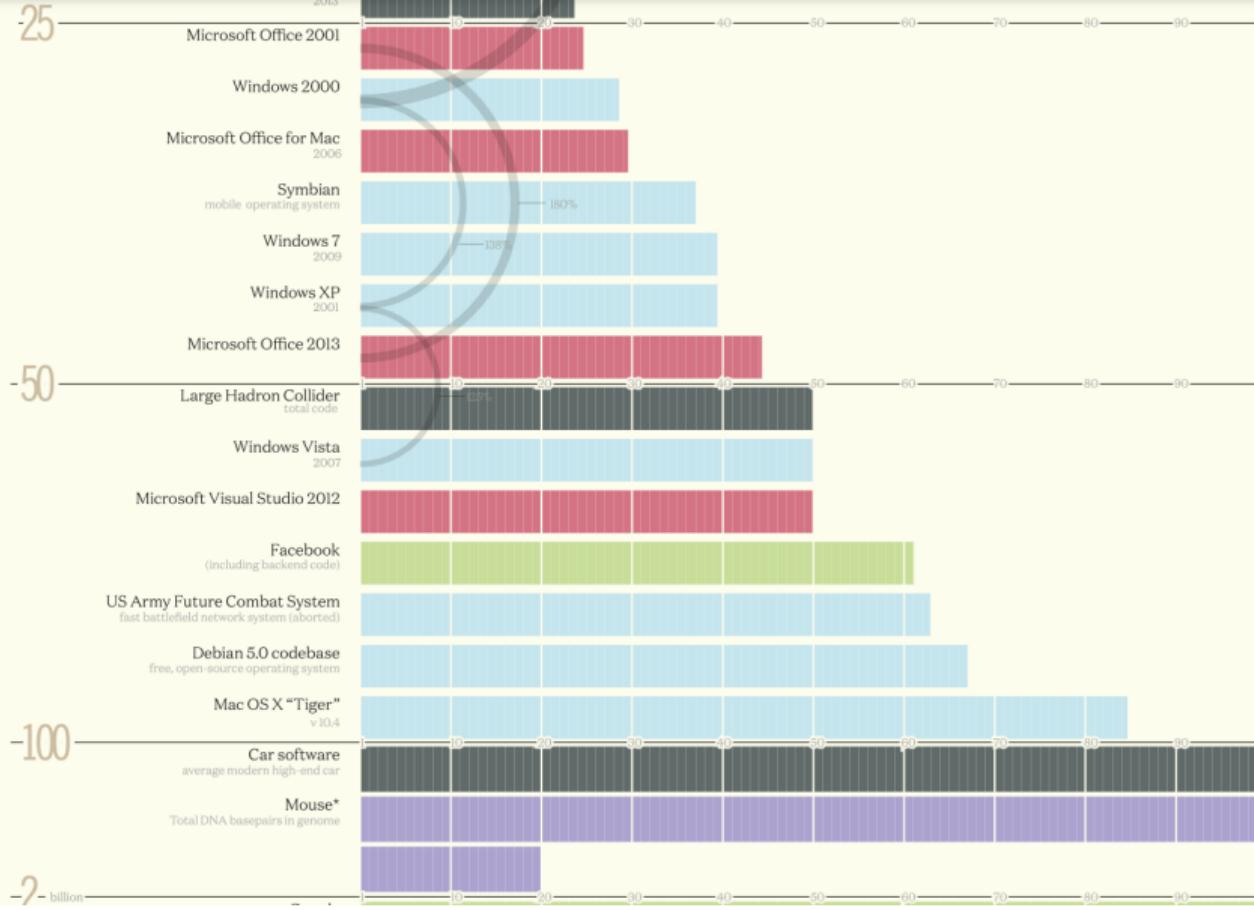
```
1 void check_pwd(char *input)
2 {
3     int rv = 0x0;
4     for (int i=0; i < PWD_LEN; i++)
5         rv |= input[i] ^ pwd[i];
6
7     return (rv == 0);
8 }
```

Rewrite program such that execution time does not depend on secrets

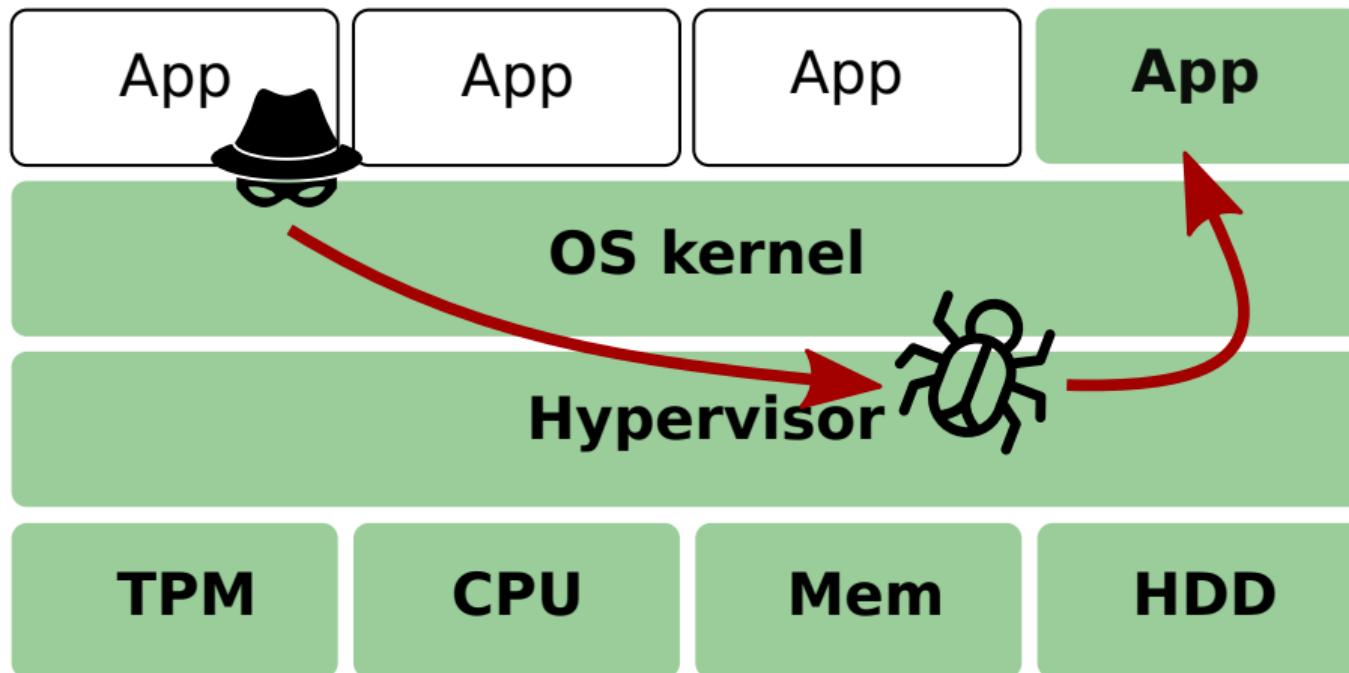
→ manual, error-prone solution; side-channels are likely here to stay...



What's inside the black box?

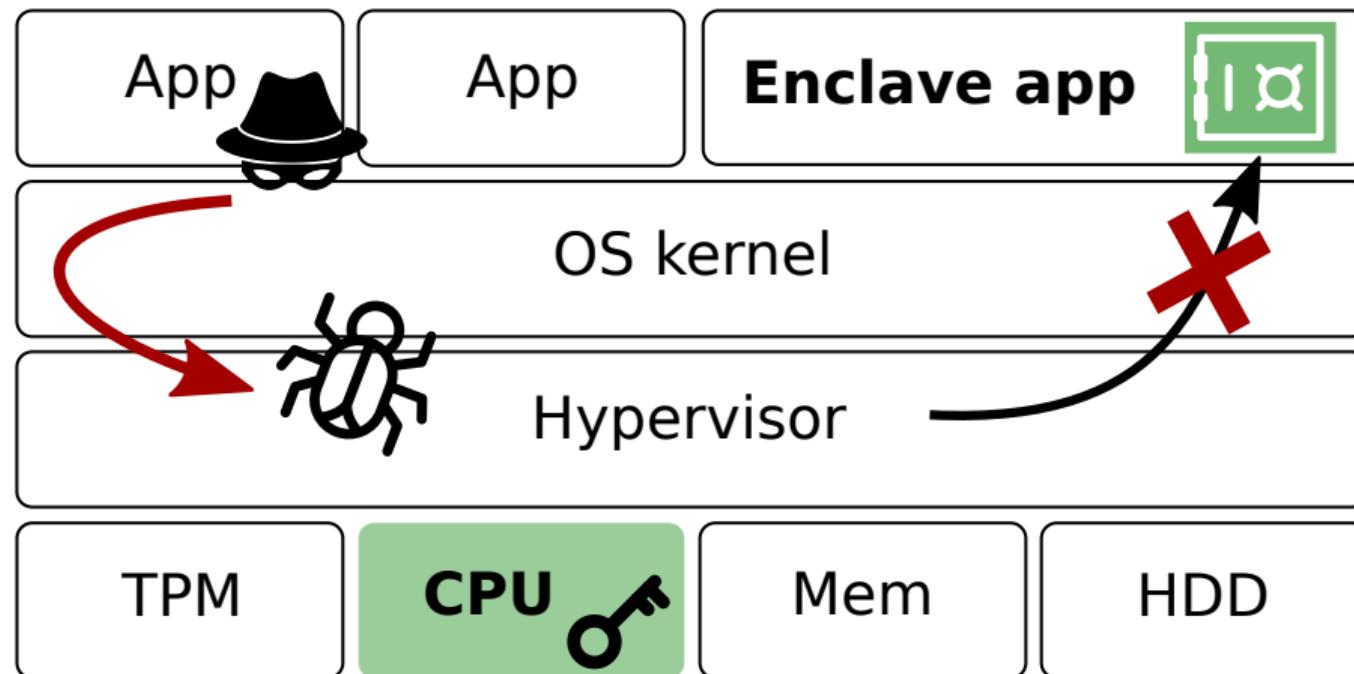


Enclaved execution: Reducing attack surface



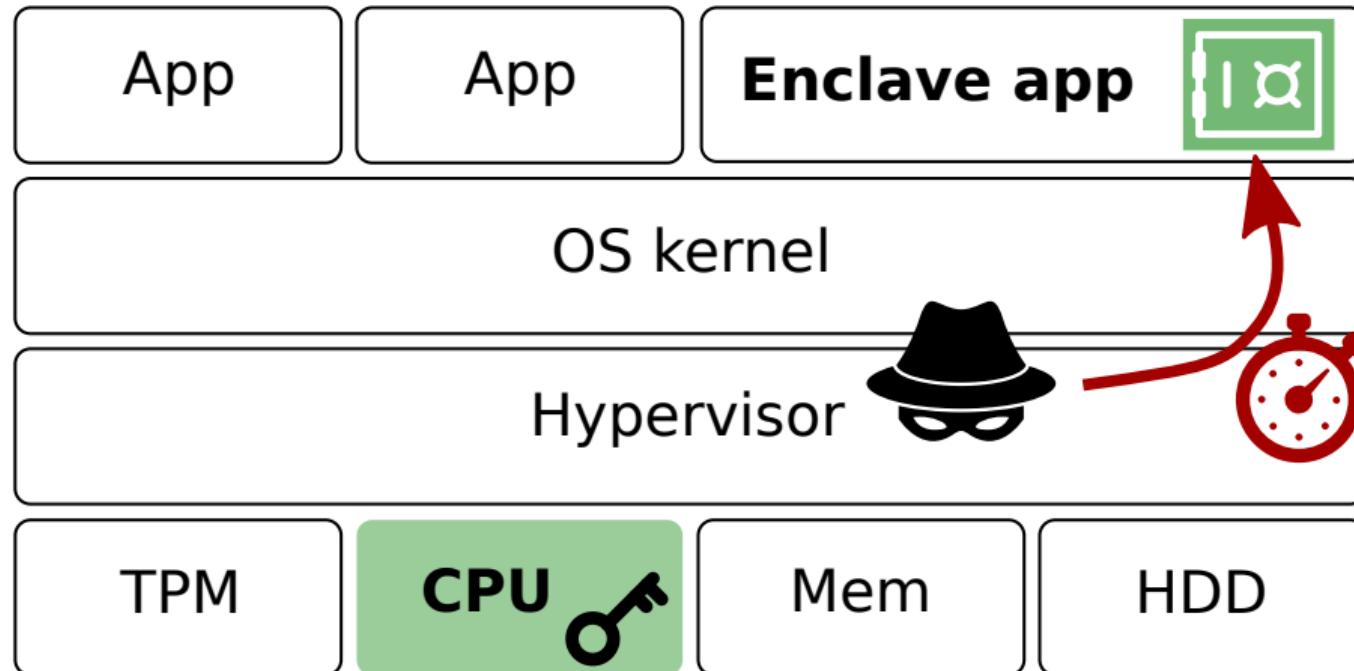
Traditional layered designs: large trusted computing base

Enclaved execution: Reducing attack surface



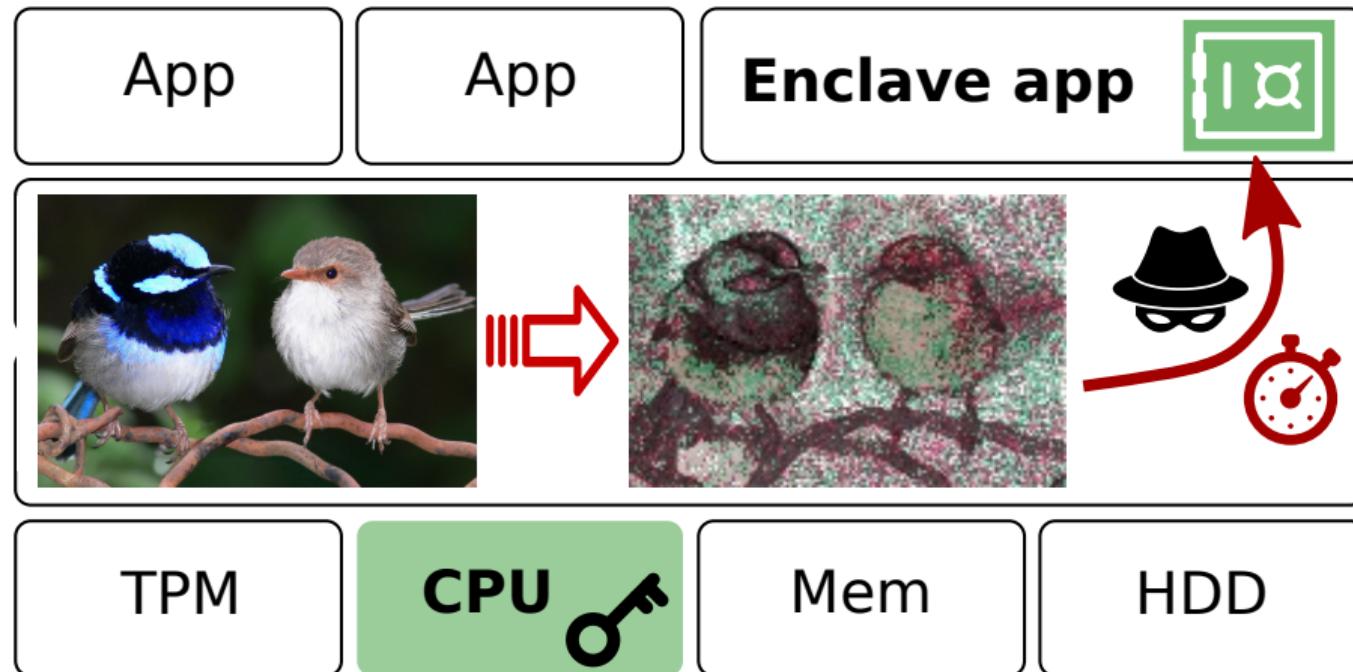
Intel SGX promise: hardware-level **isolation** and **attestation**

Enclaved execution: Privileged side-channel attacks



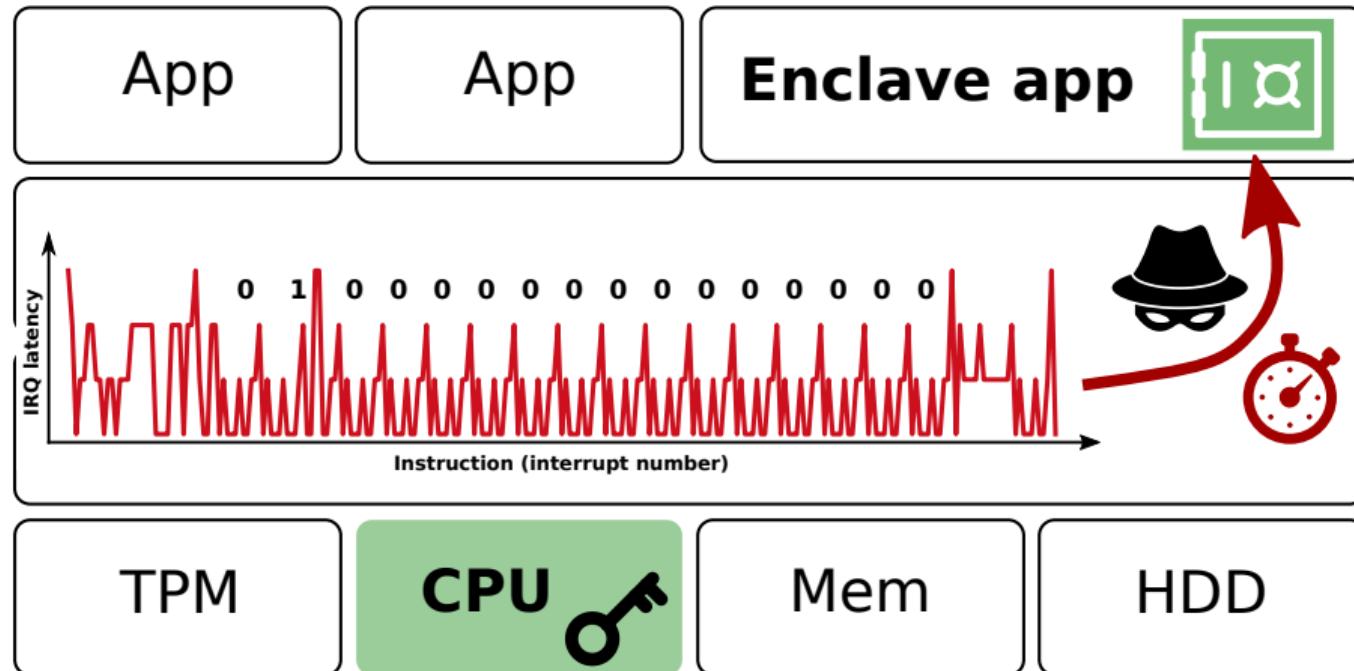
Game-changer: Untrusted OS → new class of powerful **side-channels**

Enclaved execution: Privileged side-channel attacks



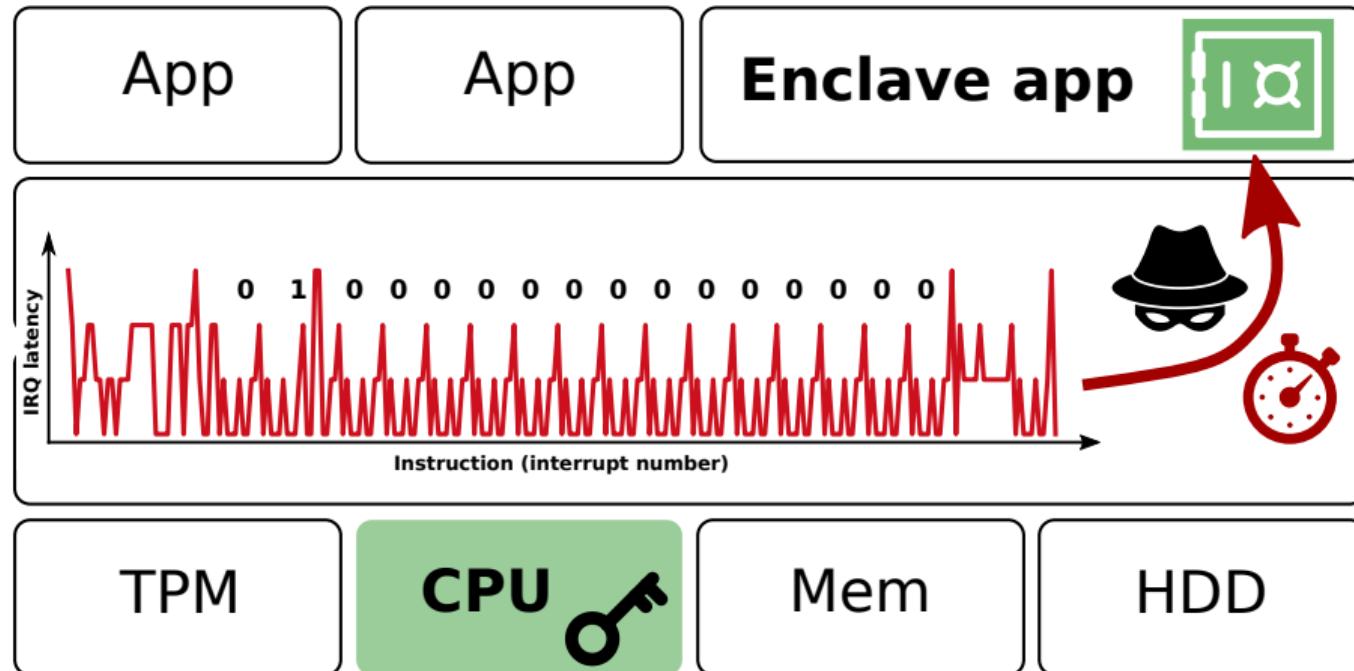
Game-changer: Untrusted OS → new class of powerful **side-channels**

Enclaved execution: Privileged side-channel attacks



Game-changer: Untrusted OS → new class of powerful **side-channels**

Enclaved execution: Privileged side-channel attacks

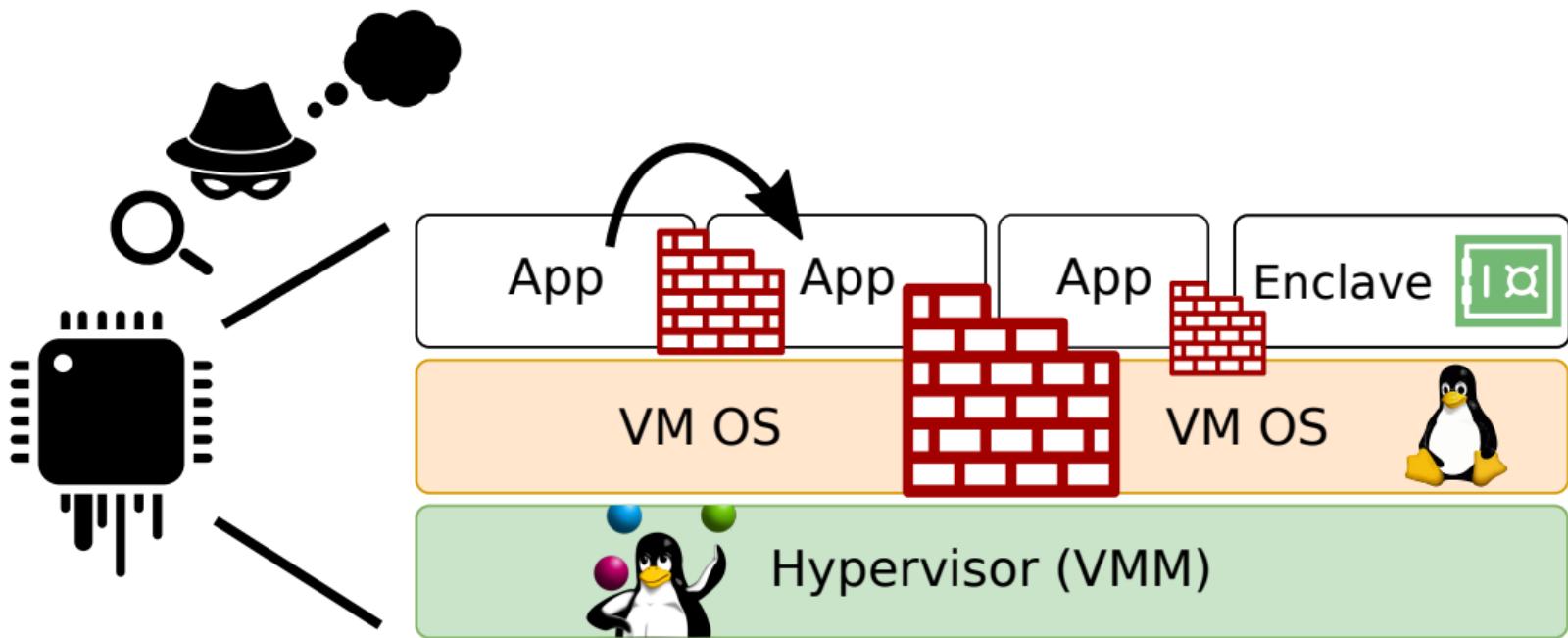


Game-changer: Untrusted OS → new class of powerful **side-channels**



We can communicate across protection walls
using microarchitectural side-channels!

Leaky processors: Jumping over protection walls with side-channels





SHARING IS NOT CARING

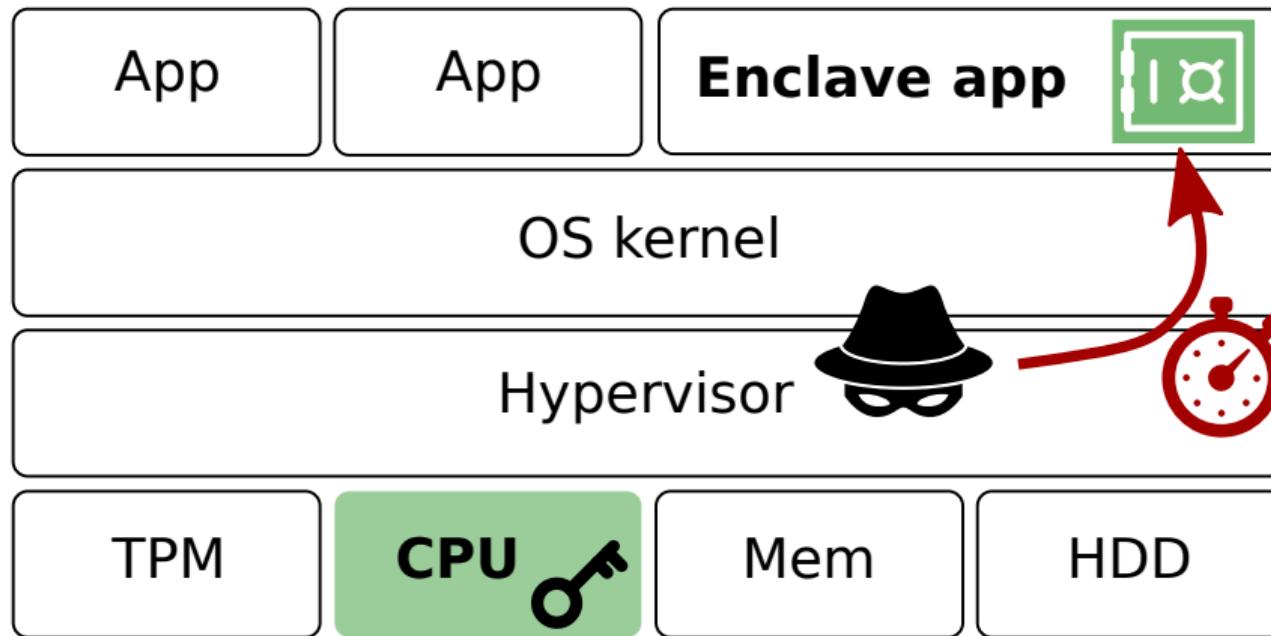
SHARING IS LOSING YOUR STUFF TO OTHERS



Can we do better? Can we demolish architectural protection walls instead of just peaking over?

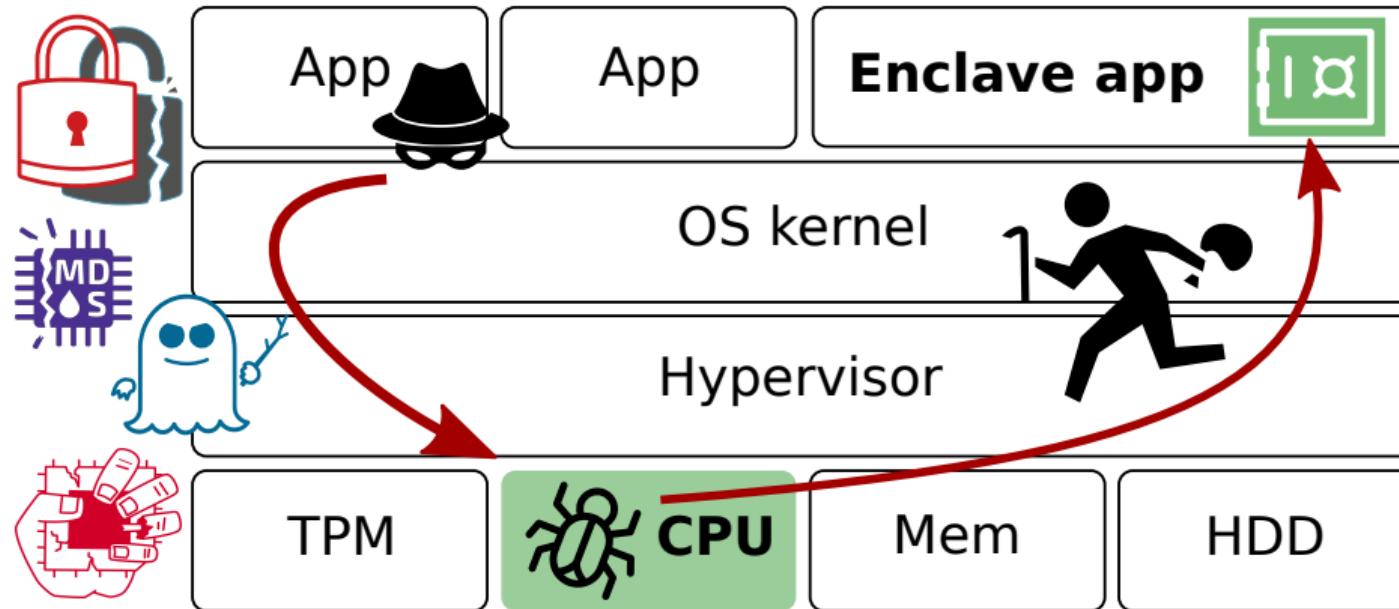


Enclaved execution: Side-channel attacks



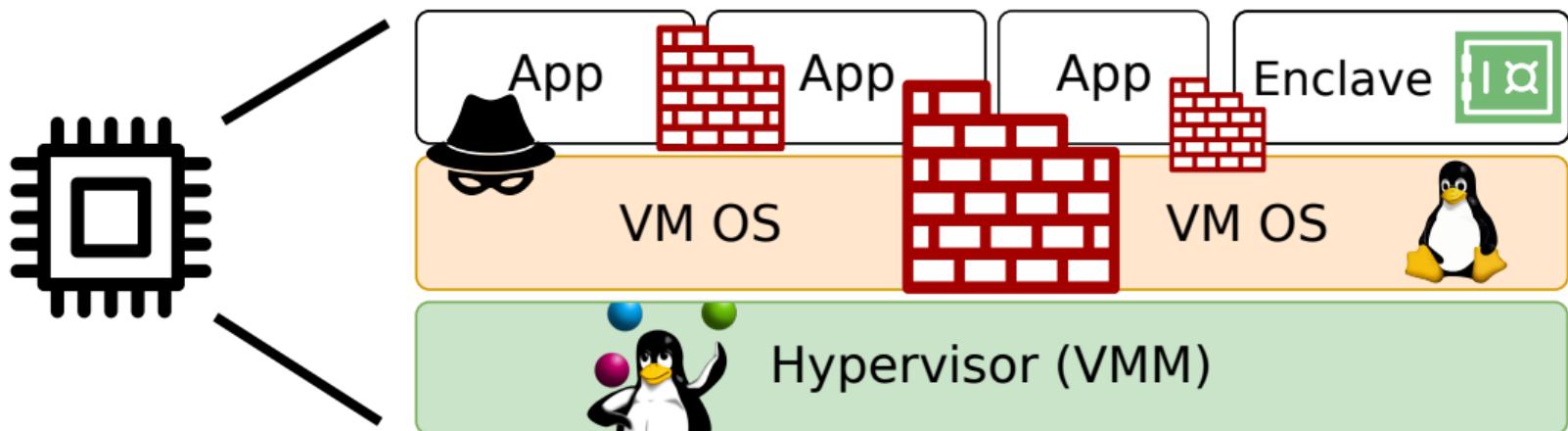
Untrusted OS → new class of powerful **side-channels**

Enclaved execution: Transient-execution attacks

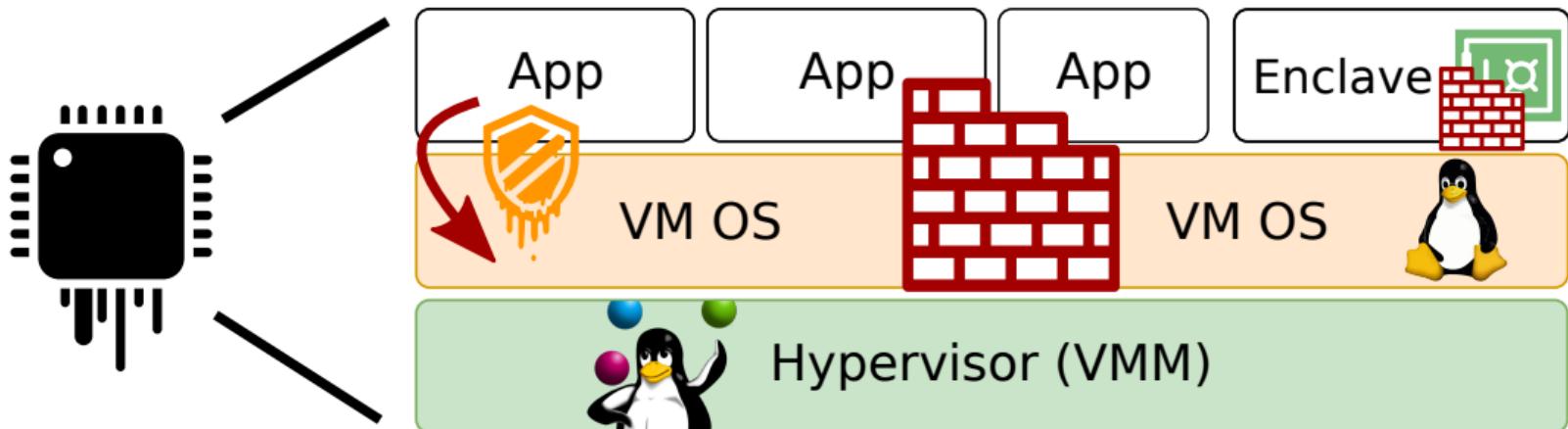


Trusted CPU → exploit **microarchitectural bugs/design flaws**

Leaky processors: Breaking isolation mechanisms

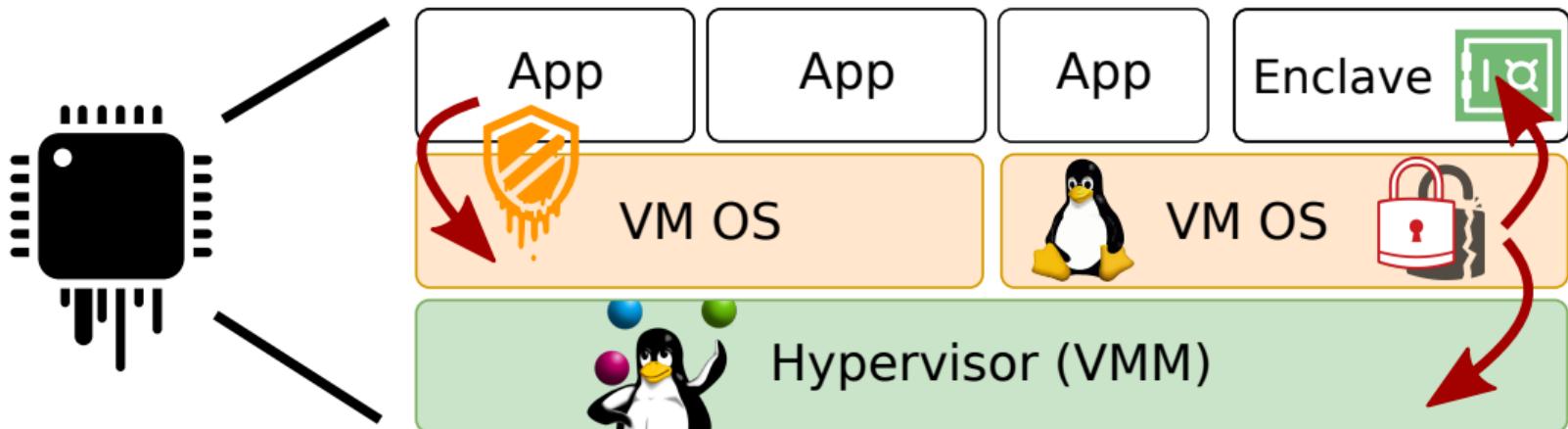


Leaky processors: Breaking isolation mechanisms



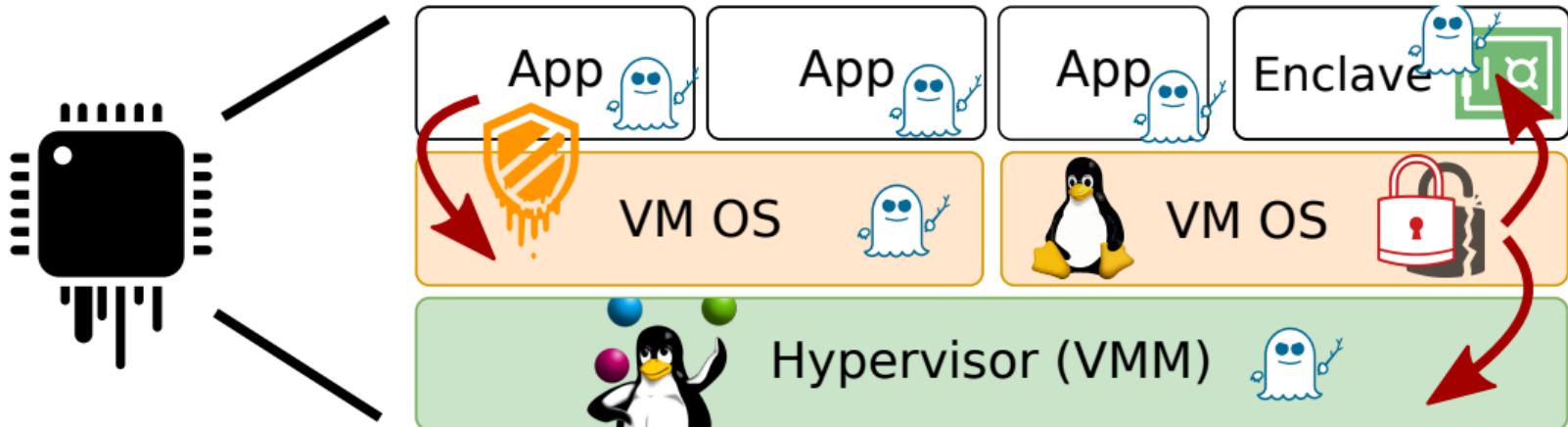
- **Meltdown** breaks user/kernel isolation

Leaky processors: Breaking isolation mechanisms



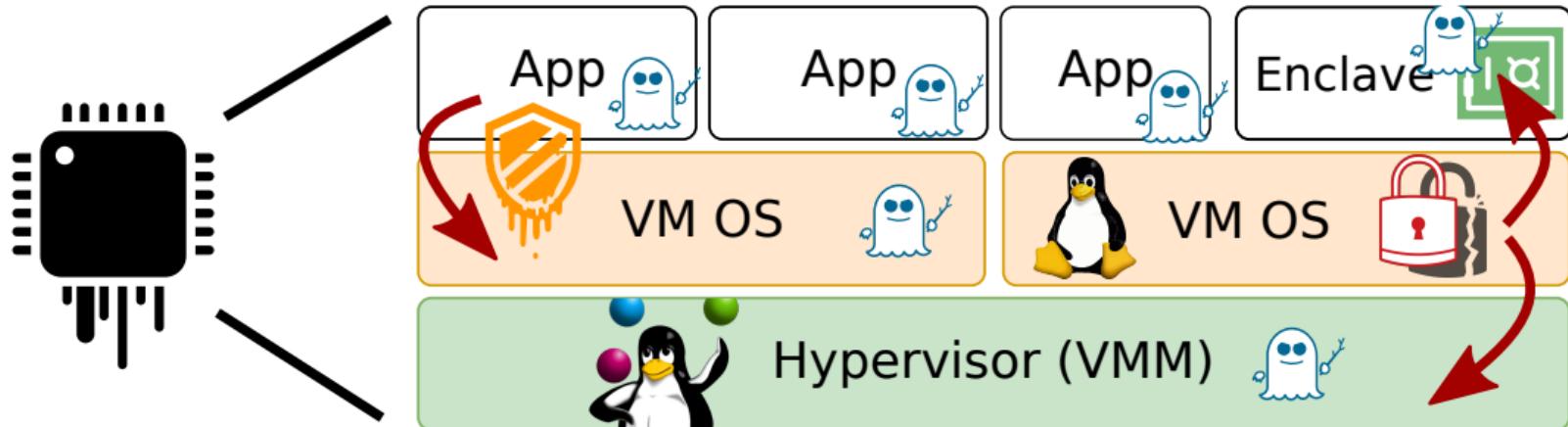
- **Meltdown** breaks user/kernel isolation
- **Foreshadow** breaks SGX enclave and virtual machine isolation

Leaky processors: Breaking isolation mechanisms

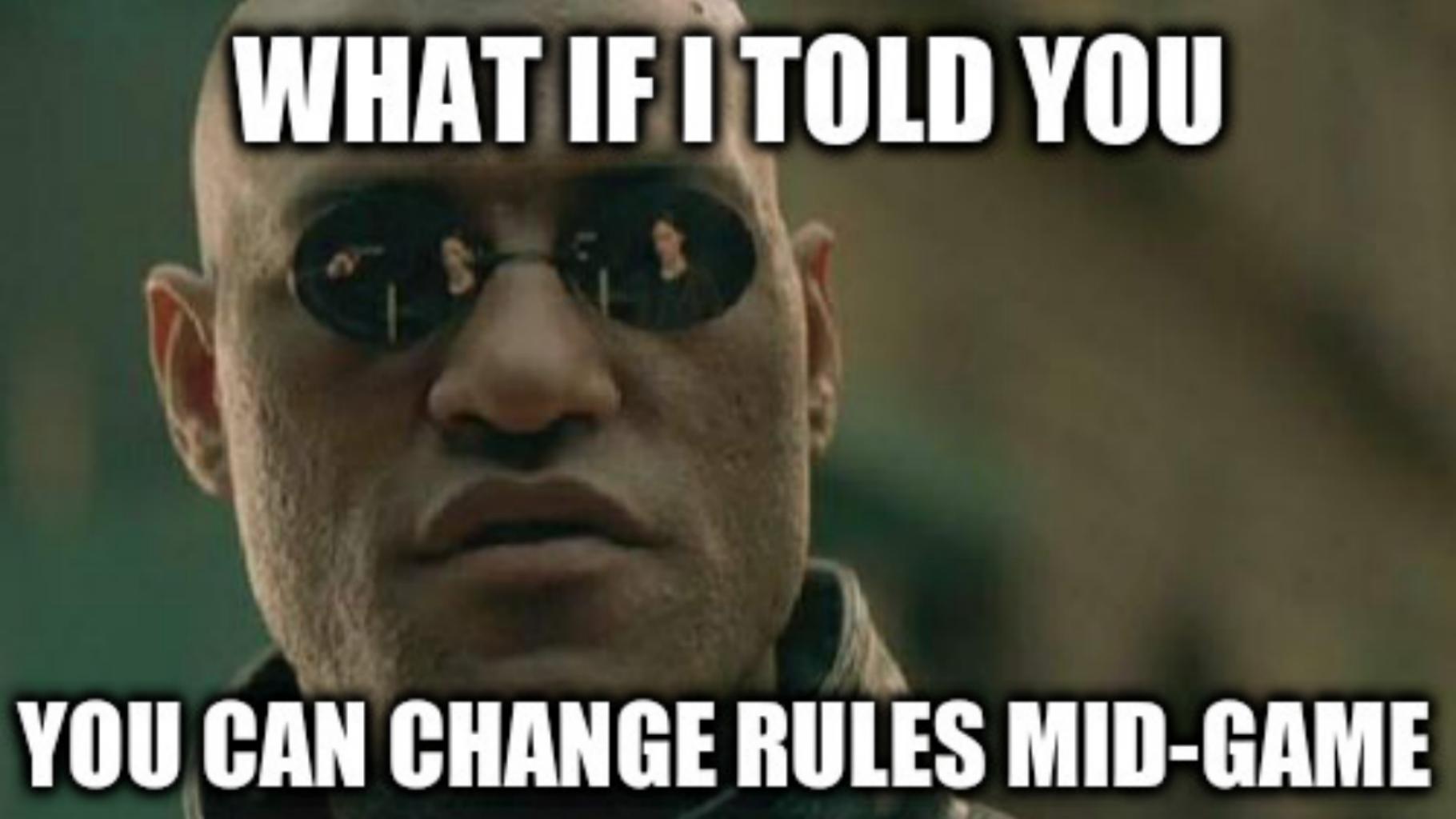


- **Meltdown** breaks user/kernel isolation
- **Foreshadow** breaks SGX enclave and virtual machine isolation
- **Spectre** breaks software-defined isolation on various levels

Leaky processors: Breaking isolation mechanisms



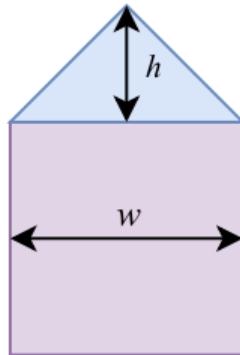
- **Meltdown** breaks user/kernel isolation
- **Foreshadow** breaks SGX enclave and virtual machine isolation
- **Spectre** breaks software-defined isolation on various levels
- ... many more – but all exploit the same underlying insights!

A close-up portrait of Agent Smith from The Matrix. He has his signature bald head, dark sunglasses, and a neutral, slightly smug expression. His skin is pale and textured. The background is a soft-focus green and brown.

WHAT IF I TOLD YOU

YOU CAN CHANGE RULES MID-GAME

Out-of-order and speculative execution

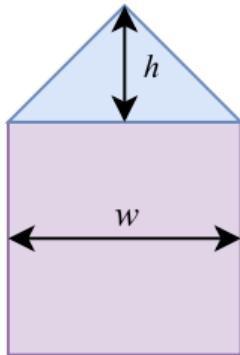


Key **discrepancy**:

- Programmers write **sequential** instructions

```
int area(int h, int w)
{
    int triangle = (w*h)/2;
    int square   = (w*w);
    return triangle + square;
}
```

Out-of-order and speculative execution



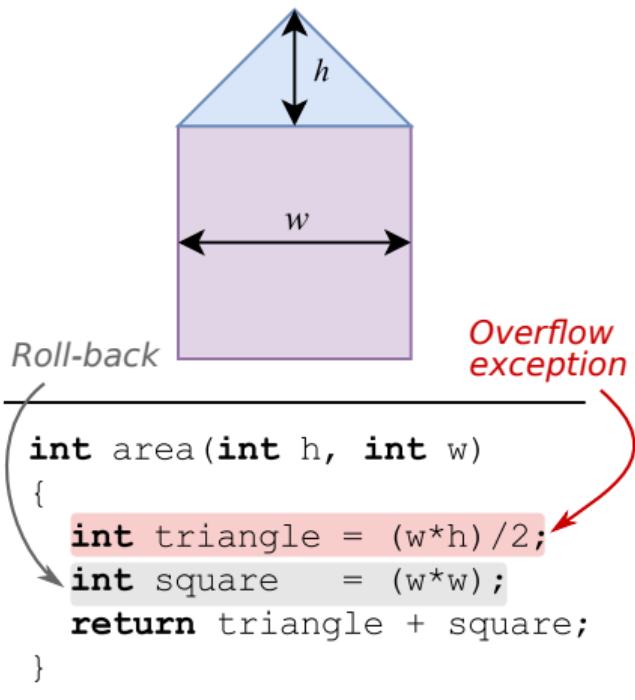
Key **discrepancy**:

- Programmers write **sequential** instructions
- Modern CPUs are inherently **parallel**

⇒ *Execute instructions ahead of time*

```
int area(int h, int w)
{
    int triangle = (w*h)/2;
    int square   = (w*w);
    return triangle + square;
}
```

Out-of-order and speculative execution



Key **discrepancy**:

- Programmers write **sequential** instructions
- Modern CPUs are inherently **parallel**

⇒ *Execute instructions ahead of time*

Best-effort: What if triangle fails?

- Commit in-order, **roll-back** square
- ... But **side-channels** may leave traces (!)

Transient-execution attacks: Welcome to the world of fun!

CPU executes ahead of time in transient world

- Success → *commit* results to normal world 😊
- Fail → *discard* results, compute again in normal world 😞



Transient-execution attacks: Welcome to the world of fun!

Key finding of 2018

⇒ *Transmit secrets from transient to normal world*



Transient-execution attacks: Welcome to the world of fun!

Key finding of 2018

⇒ *Transmit secrets from transient to normal world*



Transient world (microarchitecture) may temp bypass architectural software intentions:



Delayed exception handling



Control flow prediction

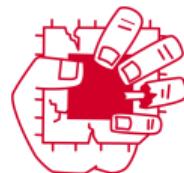
Transient-execution attacks: Welcome to the world of fun!

Key finding of 2018

⇒ *Transmit secrets from transient to normal world*



Transient world (microarchitecture) may temp bypass architectural software intentions:

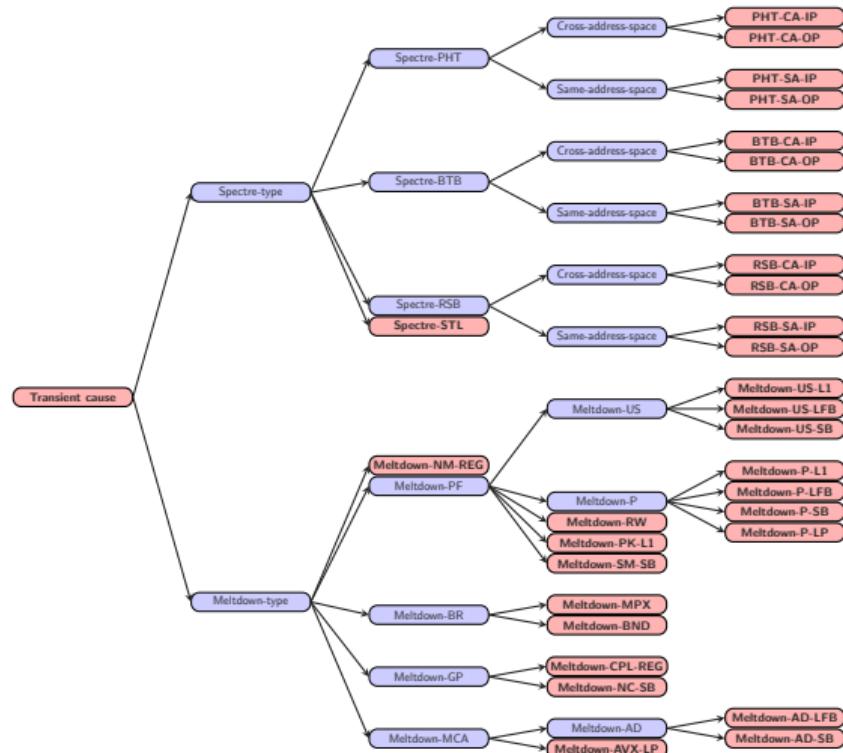


CPU access control bypass

Speculative buffer overflow/ROP

The transient-execution zoo

<https://transient.fail>

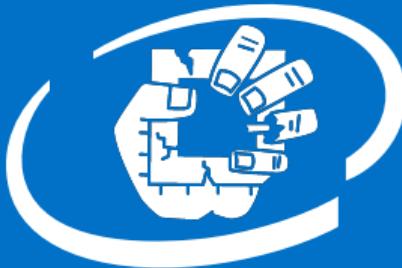




inside™



inside™

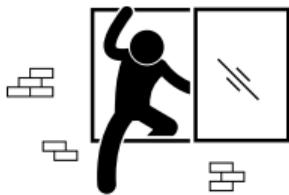


inside™



inside™

Meltdown: Transiently encoding unauthorized memory



Unauthorized access

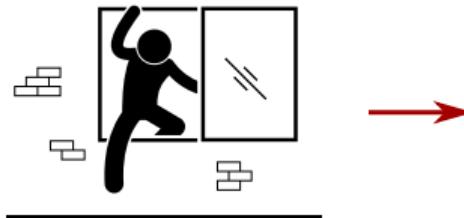
Listing 1: x86 assembly

```
1 meltdown:  
2     // %rdi: oracle  
3     // %rsi: secret_ptr  
4  
5     movb (%rsi), %al  
6     shl $0xc, %rax  
7     movq (%rdi, %rax), %rdi  
8     retq
```

Listing 2: C code.

```
1 void meltdown(  
2         uint8_t *oracle,  
3         uint8_t *secret_ptr)  
4 {  
5     uint8_t v = *secret_ptr;  
6     v = v * 0x1000;  
7     uint64_t o = oracle[v];  
8 }
```

Meltdown: Transiently encoding unauthorized memory



Unauthorized access



Transient out-of-order window

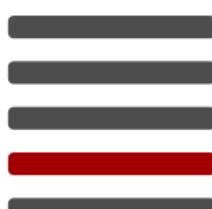
Listing 1: x86 assembly.

```
1 meltdown:  
2 // %rdi: oracle  
3 // %rsi: secret_ptr  
4  
5 movb (%rsi), %al  
6 shl $0xc, %rax  
7 movq (%rdi, %rax), %rdi  
8 retq
```

Listing 2: C code.

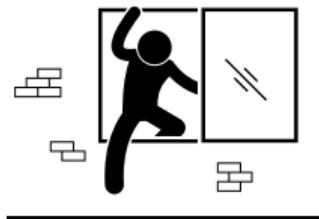
```
1 void meltdown(  
2     uint8_t *oracle,  
3     uint8_t *secret_ptr)  
4 {  
5     uint8_t v = *secret_ptr;  
6     v = v * 0x1000;  
7     uint64_t o = oracle[v];  
8 }
```

oracle array



secret idx

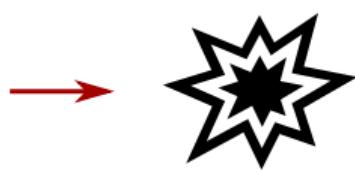
Meltdown: Transiently encoding unauthorized memory



Unauthorized access



Transient out-of-order window



Exception

(discard architectural state)

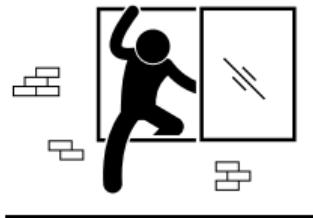
Listing 1: x86 assembly.

```
1 meltdown:  
2 // %rdi: oracle  
3 // %rsi: secret_ptr  
4  
5 movb (%rsi), %al  
6 shr $0xc, %rax  
7 movq (%rdi,%rax),%rdi  
8 retq
```

Listing 2: C code.

```
1 void meltdown(  
2     uint8_t *oracle,  
3     uint8_t *secret_ptr)  
4 {  
5     uint8_t v = *secret_ptr;  
6     v = v * 0x1000;  
7     uint64_t o = oracle[v];  
8 }
```

Meltdown: Transiently encoding unauthorized memory



Unauthorized access



Transient out-of-order window



Exception handler

Listing 1: x86 assembly.

```
1 meltdown:  
2     // %rdi: oracle  
3     // %rsi: secret_ptr  
4  
5     movb (%rsi), %al  
6     shl $0xc, %rax  
7     movq (%rdi), %rax  
8     movq (%rdi, %rax), %rdi  
9     retq
```

Listing 2: C code.

```
1 void meltdown(  
2         uint8_t *oracle,  
3         uint8_t *secret_ptr)  
4 {  
5     uint8_t v = *secret_ptr;  
6     v = v * 0x1000;  
7     uint64_t o = oracle[v];  
8 }
```

oracle array



cache hit



inside™



inside™

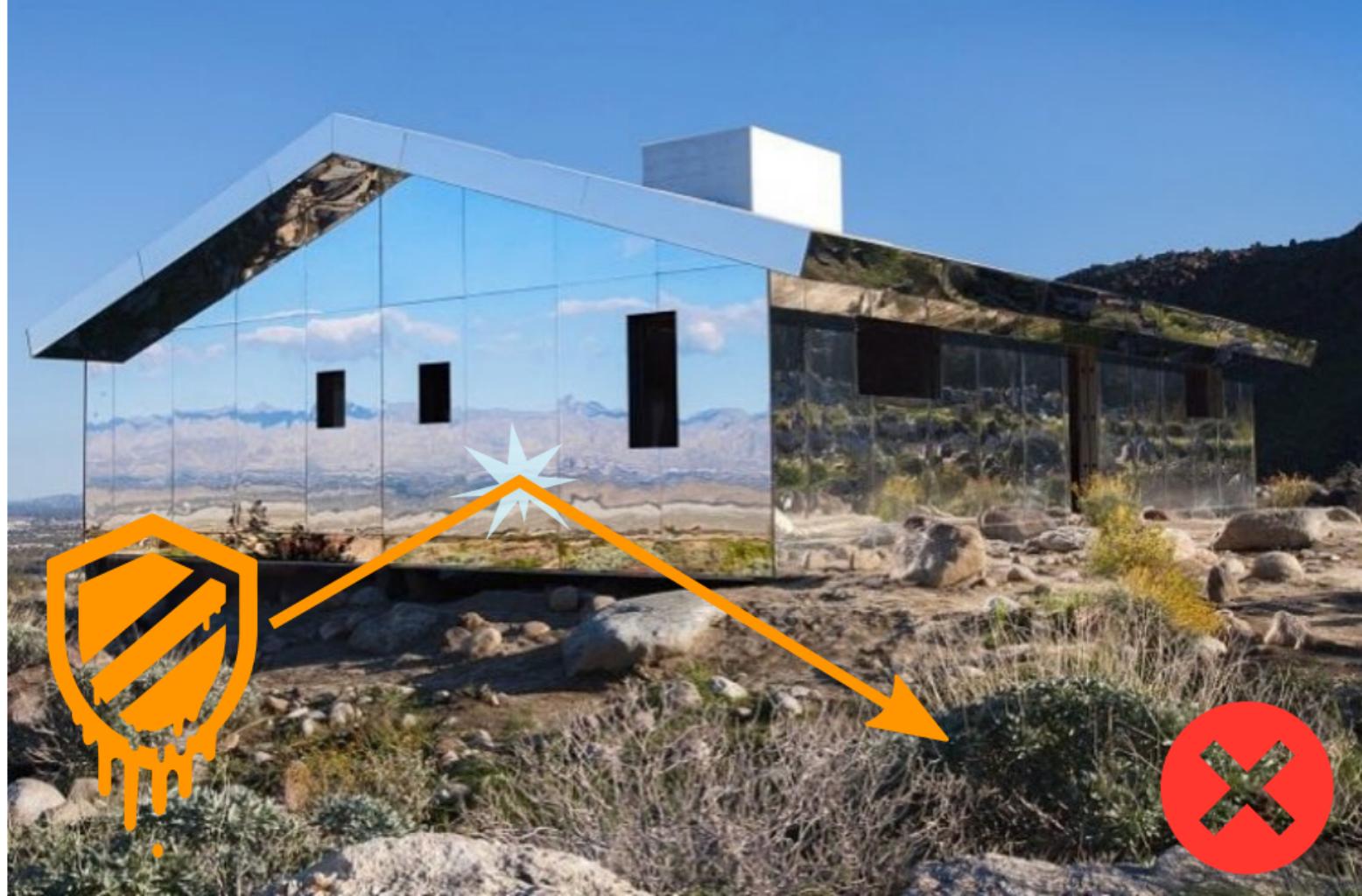


inside™



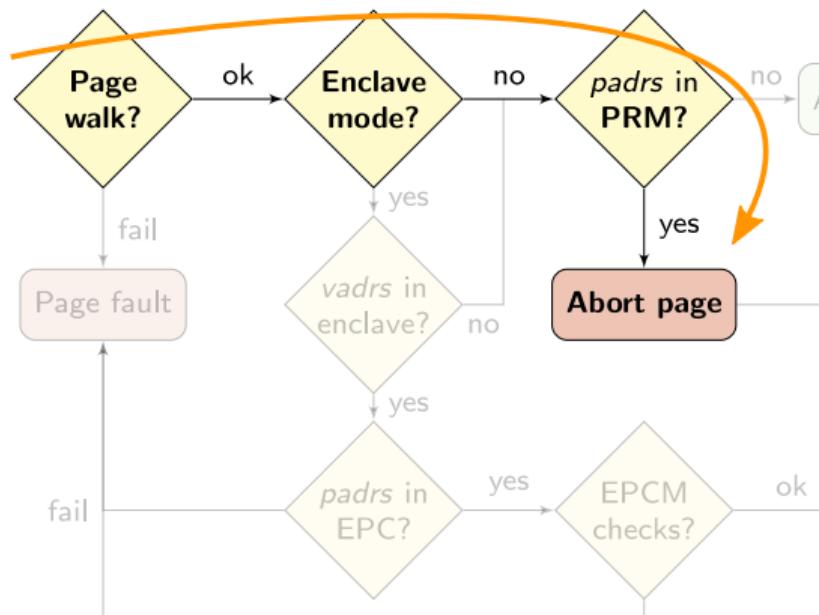
inside™





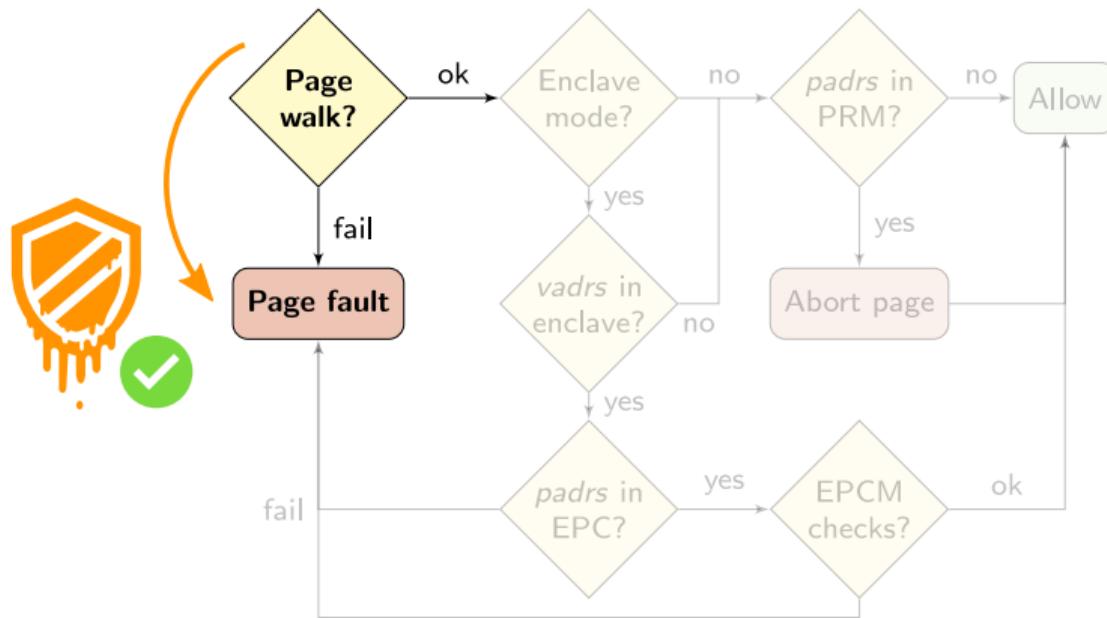
Building Foreshadow: Evade the abort page

Straw man: (Speculative) accesses in non-enclave mode are dropped



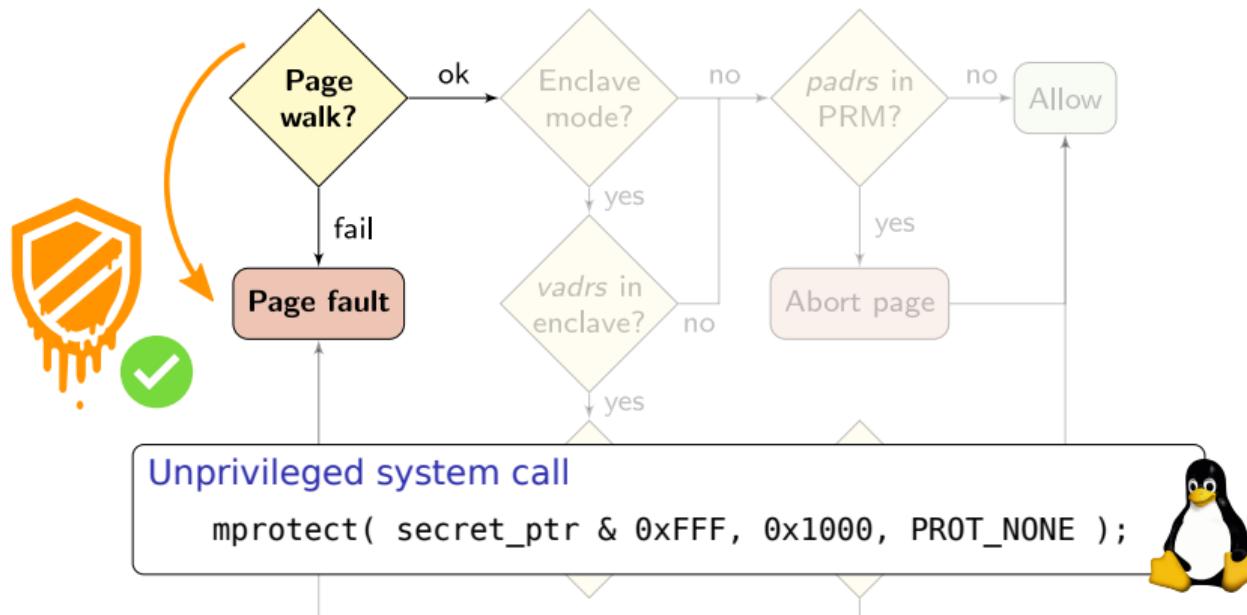
Building Foreshadow: Evade the abort page

Stone man: Bypass abort page via *untrusted* page table



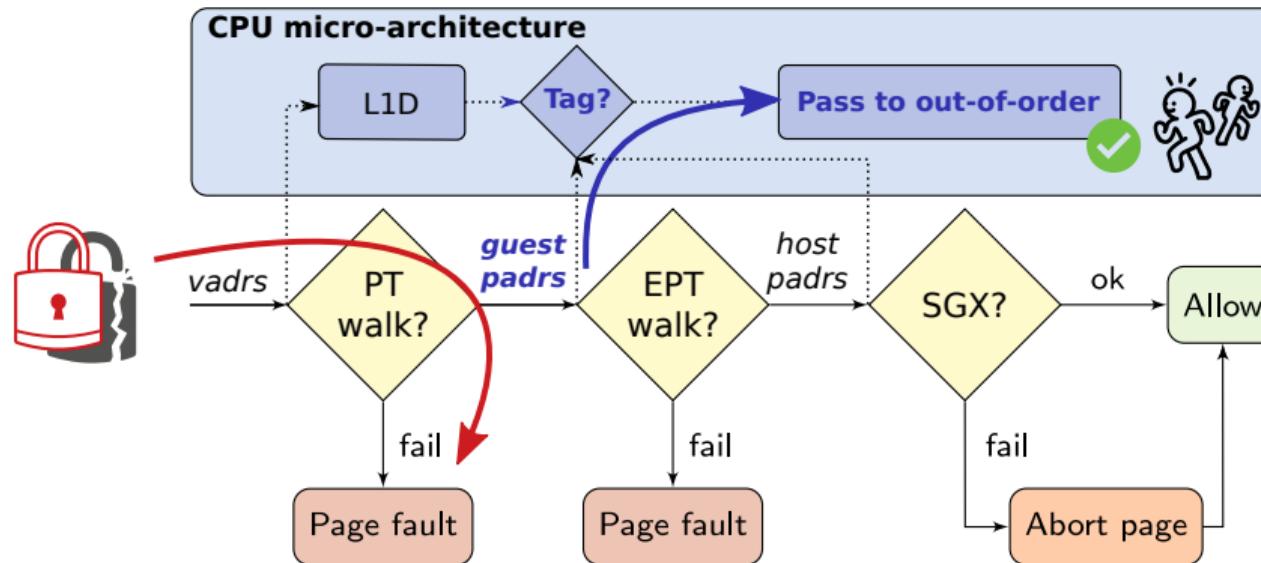
Building Foreshadow: Evade the abort page

Stone man: Bypass abort page via *untrusted* page table

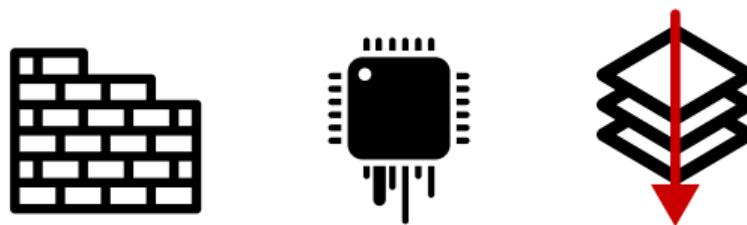


Foresight-NG: Breaking the virtual memory abstraction

L1-Terminal Fault: match *unmapped physical address* (!)



- ⇒ New emerging and powerful class of **transient-execution** attacks
- ⇒ Importance of fundamental **side-channel research**; no silver-bullet defenses
- ⇒ Security **cross-cuts** the system stack: hardware, OS, VMM, compiler, application



Appendix

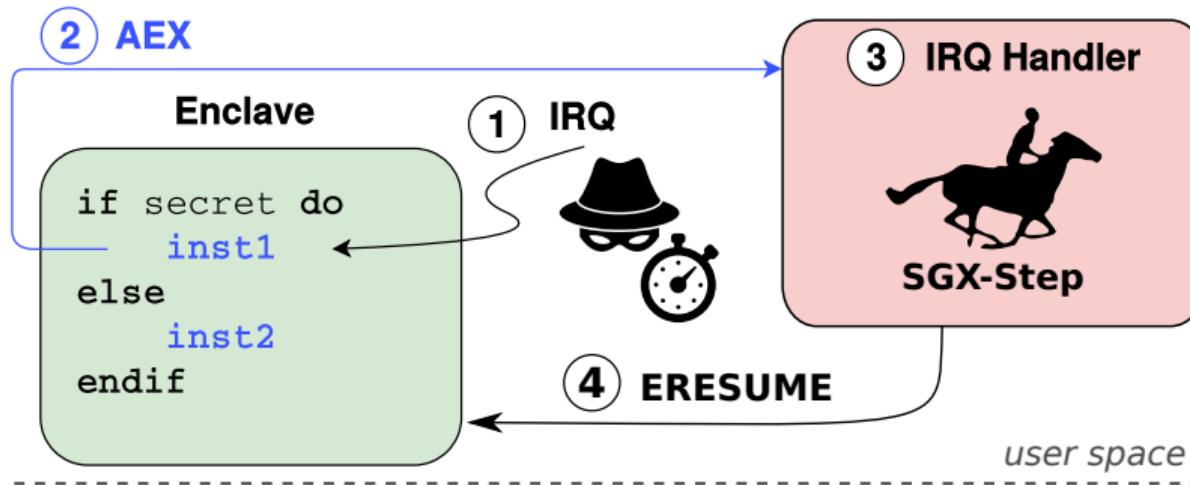
References i

-  C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss.
A Systematic Evaluation of Transient Execution Attacks and Defenses.
In *Proceedings of the 28th USENIX Security Symposium*, 2019.
-  J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx.
Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution.
In *Proceedings of the 27th USENIX Security Symposium*. USENIX Association, August 2018.

-  J. Van Bulck, F. Piessens, and R. Strackx.
SGX-Step: A practical attack framework for precise enclave execution control.
In *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, SysTEX'17, pp. 4:1–4:6. ACM, 2017.
-  J. Van Bulck, F. Piessens, and R. Strackx.
Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic.
In *Proceedings of the 25th ACM Conference on Computer and Communications Security (CCS'18)*. ACM, October 2018.

-  J. Van Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx.
Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution.
In *Proceedings of the 26th USENIX Security Symposium*. USENIX Association, August 2017.

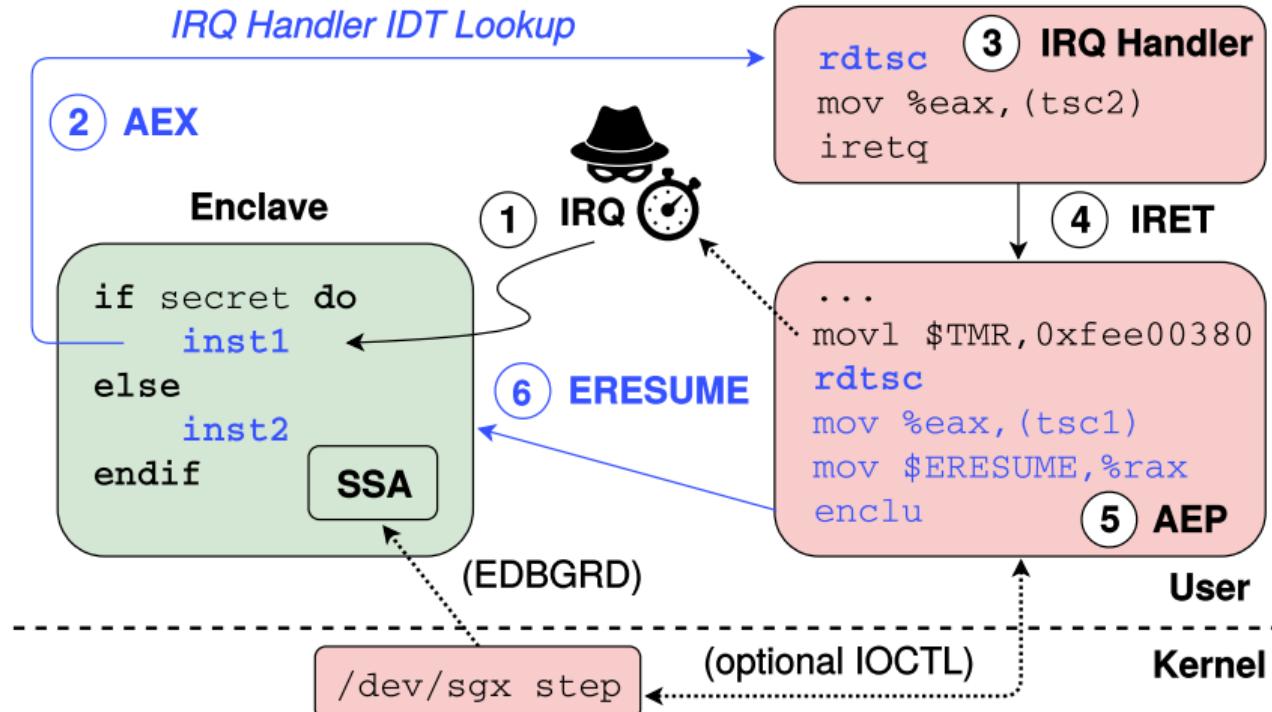
SGX-Step: Executing enclaves one instruction at a time



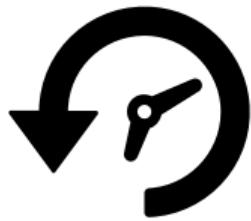
Van Bulck et al. "SGX-Step: A practical attack framework for precise enclave execution control", SysTEX 2017

Van Bulck et al. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic", CCS 2018

SGX-Step: Executing enclaves one instruction at a time



Mitigating Foreshadow



1. Cache secrets in L1



2. Unmap [page table](#) entry



3. Execute [Meltdown](#)

Mitigating Foreshadow



1. Cache secrets in L1



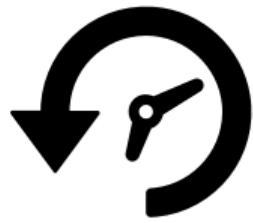
2. Unmap page table entry



3. Execute Meltdown

Future CPUs
(silicon-based changes)

Mitigating Foreshadow



1. Cache secrets in L1



2. Unmap page table entry



3. Execute Meltdown

OS kernel updates
(sanitize page frame bits)

Mitigating Foreshadow



1. Cache secrets in L1



2. Unmap page table entry

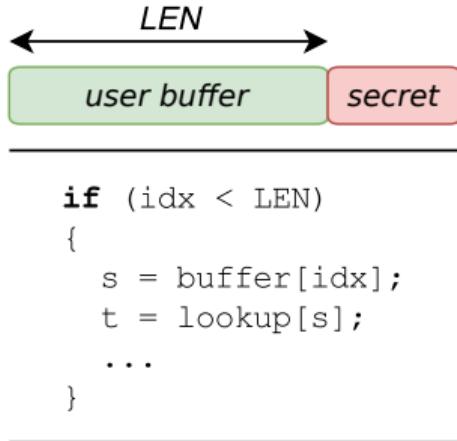


3. Execute Meltdown

Intel microcode updates

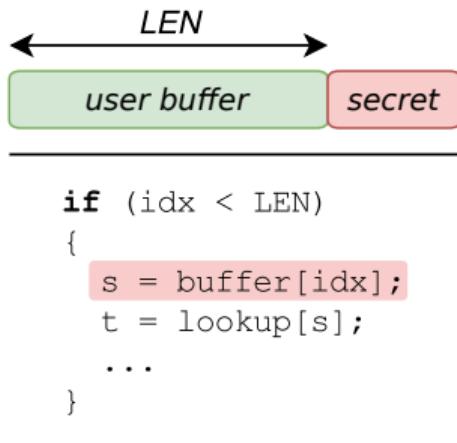
⇒ **Flush L1** cache on enclave/VMM exit + **disable HyperThreading**

Spectre v1: Speculative buffer over-read



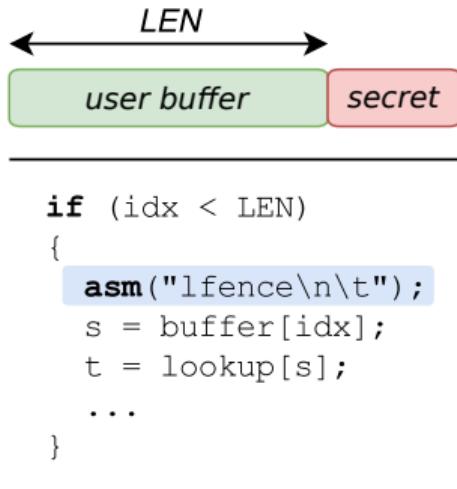
- Programmer *intention*: never access out-of-bounds memory

Spectre v1: Speculative buffer over-read



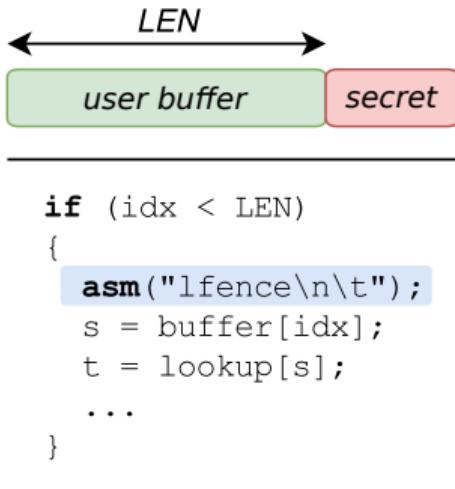
- Programmer *intention*: never access out-of-bounds memory
- Branch can be mistrained to **speculatively** (i.e., ahead of time) execute with $idx \geq LEN$ in the **transient world**

Spectre v1: Speculative buffer over-read



- Programmer *intention*: never access out-of-bounds memory
- Branch can be mistrained to **speculatively** (i.e., ahead of time) execute with $idx \geq LEN$ in the **transient world**
- Insert explicit **speculation barriers** to tell the CPU to halt the transient world...

Spectre v1: Speculative buffer over-read



- Programmer *intention*: never access out-of-bounds memory
- Branch can be mistrained to **speculatively** (i.e., ahead of time) execute with $idx \geq LEN$ in the **transient world**
- Insert explicit **speculation barriers** to tell the CPU to halt the transient world...
- Huge manual, error-prone effort...