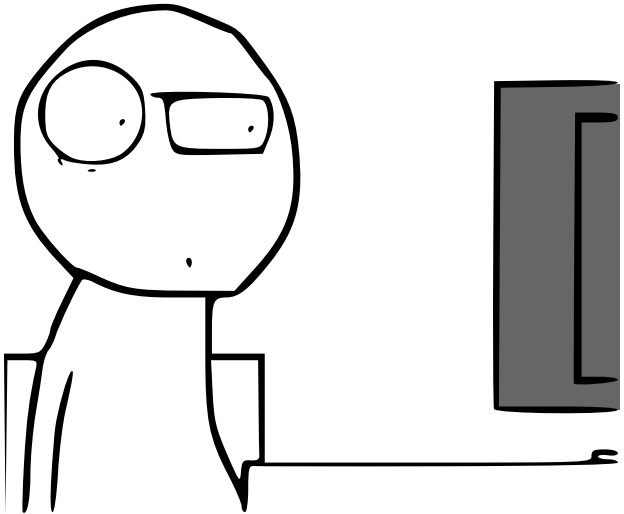
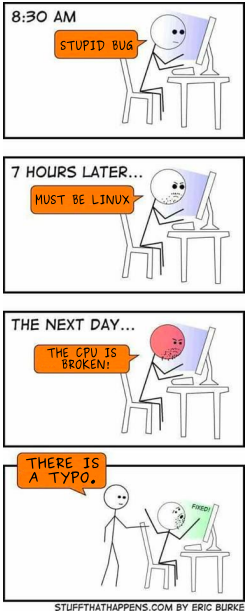
From Random Observations to Automated Leakage Discovery

Michael Schwarz

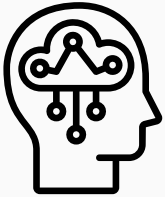
December 2022

CISPA Helmholtz Center for Information Security



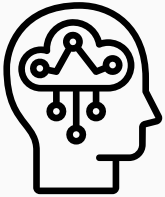

8:30 AM
STUPID BUG
7 HOURS LATER...
MUST BE LINUX
THE NEXT DAY...
THE CPU IS BROKEN!
THERE IS A TYPO.
FIXED!
STUFFTHATHAPPENS.COM BY ERIC BURKE

In-Order Execution



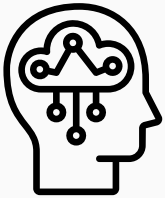
- Mental model of CPU is simple

In-Order Execution



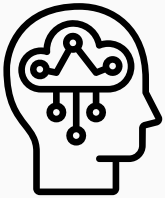
- Mental model of CPU is simple
- Instructions are executed in program order

In-Order Execution

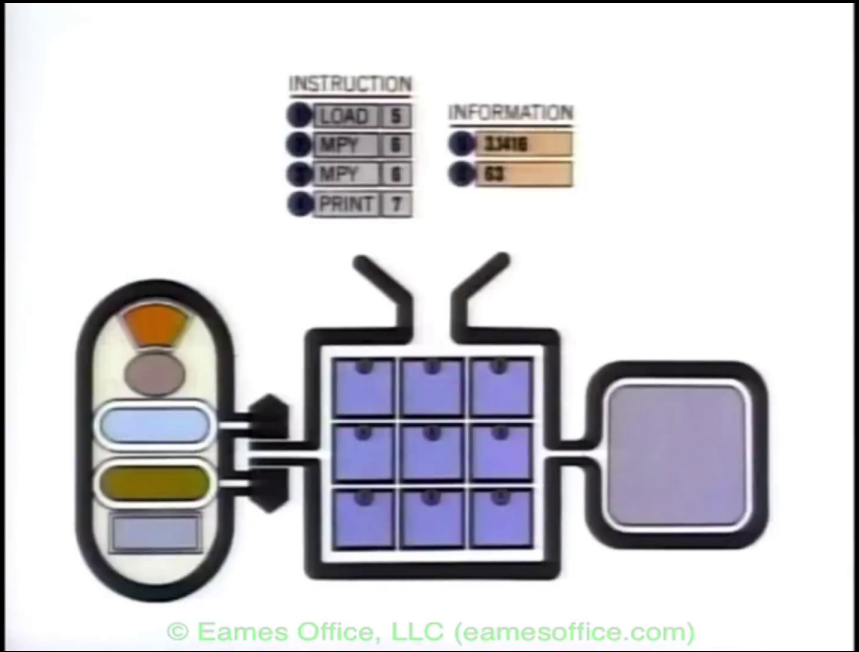


- Mental model of CPU is simple
- Instructions are executed in program order
- Pipeline stalls when stages are not ready

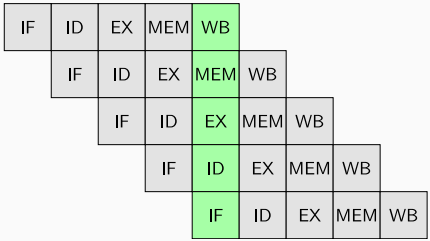
In-Order Execution



- Mental model of CPU is simple
- Instructions are executed in program order
- Pipeline stalls when stages are not ready
- If data is not cached, we need to wait

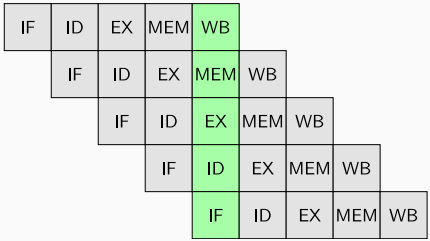


In-Order Execution



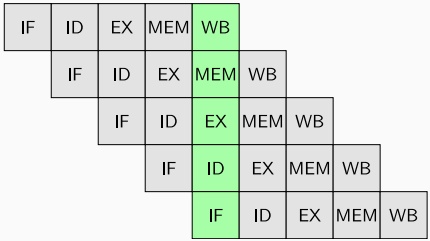
- Instructions are...

In-Order Execution



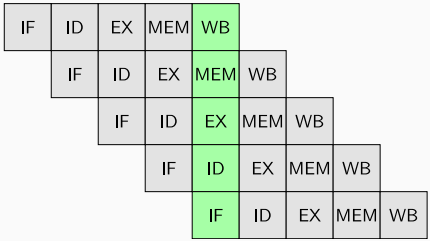
- Instructions are...
  - fetched (`IF`) from the L1 Instruction Cache

In-Order Execution



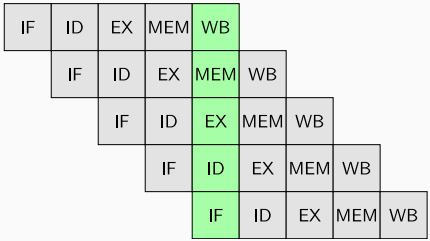
- Instructions are...
  - fetched (`IF`) from the L1 Instruction Cache
  - decoded (`ID`)

In-Order Execution



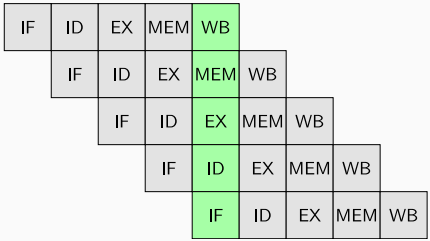
- Instructions are...
  - fetched (`IF`) from the L1 Instruction Cache
  - decoded (`ID`)
  - executed (`EX`) by execution units

In-Order Execution



- Instructions are...
  - fetched (`IF`) from the L1 Instruction Cache
  - decoded (`ID`)
  - executed (`EX`) by execution units
- Memory access is performed (`MEM`)

In-Order Execution



- Instructions are...
  - fetched (`IF`) from the L1 Instruction Cache
  - decoded (`ID`)
  - executed (`EX`) by execution units
- Memory access is performed (`MEM`)
- Architectural register file is updated (`WB`)

Measuring Time

```
x = y + 1
```

Measuring Time

```
start = ⏱
  x = y + 1
end = ⏱
```

Measuring Time

```
start = ⏱
  x = y + 1
end = ⏱
Δ = end - start
```

Measuring Time

```
start = ⏱
  x = y + 1
end = ⏱
Δ = end - start
```

1. run: Δ = 302

Measuring Time

```
start = ⏱
  x = y + 1
end = ⏱
Δ = end - start
```

1. run: $\Delta = 302$
2. run: $\Delta = 54$

Measuring Time

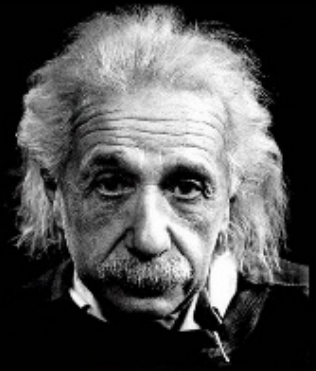
```
start = ⏱
  x = y + 1
end = ⏱
Δ = end - start
```

1. run: Δ = 302
2. run: Δ = 54

Determinism?

Same code with **different** execution time **without** changes

Insanity: doing the same thing over and over again and expecting different results.



Albert Einstein

Measuring Time

```
start = ⏱
```

```
end = ⏱
```

Measuring Time

```
start = ⏱
```

```
end = ⏱
Δ = end - start
```

Measuring Time

```
start = ⏱
```

1. run: Δ = 12

```
end = ⏱
Δ = end - start
```

Measuring Time

```
start = ⏱
```

```
end = ⏱
```

```
Δ = end - start
```

1. run: $\Delta = 12$
2. run: $\Delta = 12$

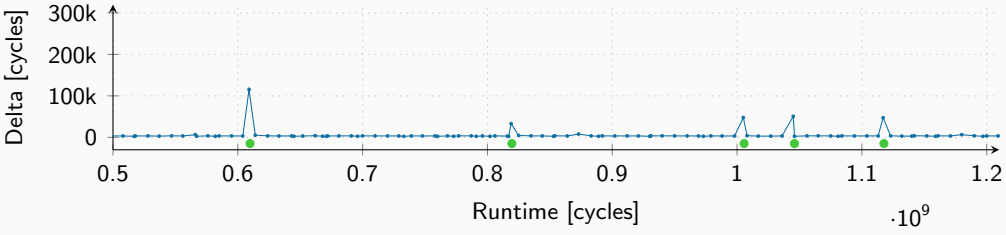
Measuring Time

```
start = ⏱
```

```
end = ⏱
```

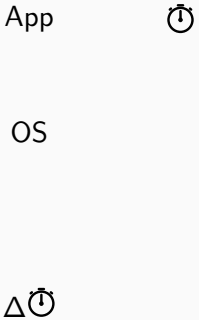
```
Δ = end - start
```

1. run: $\Delta = 12$
2. run: $\Delta = 12$

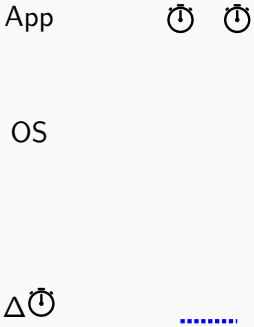



I HAVE NO
IDEA WHAT
I'M DOING

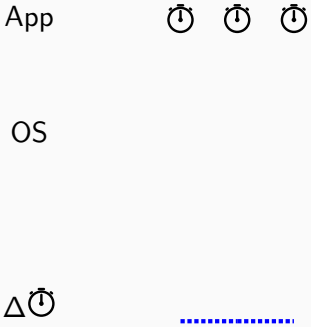
Interrupts!



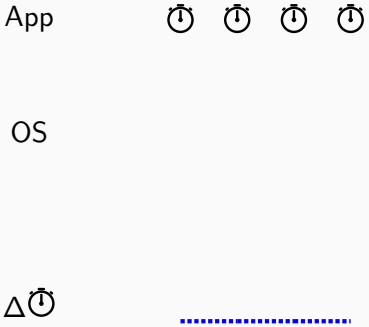
Interrupts!



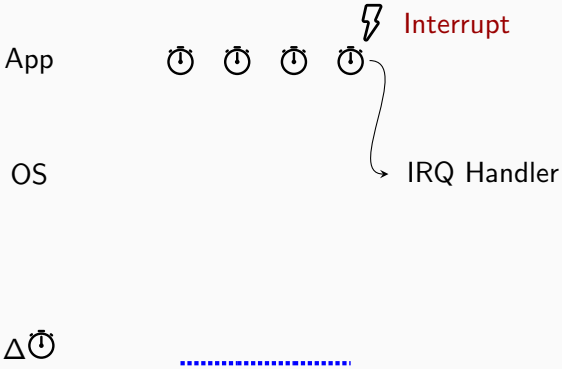
Interrupts!



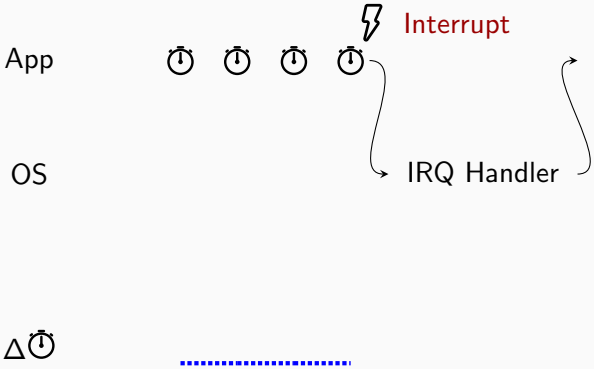
Interrupts!



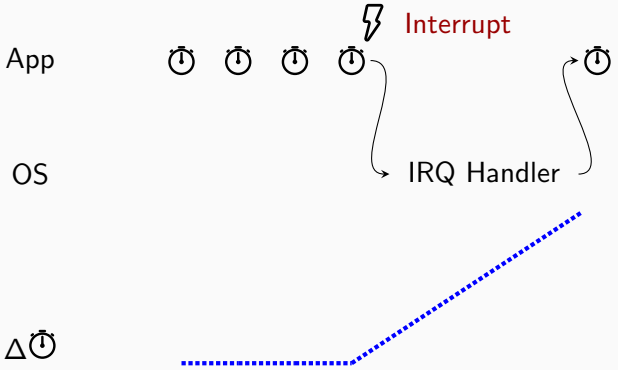
Interrupts!



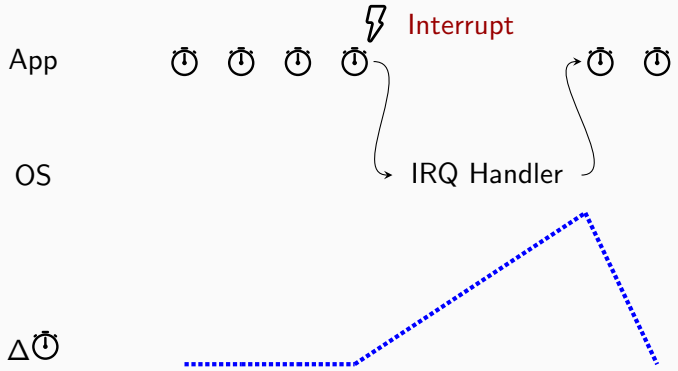
Interrupts!



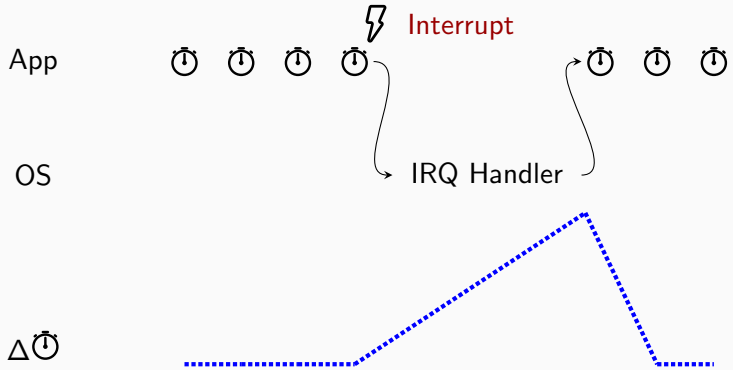
Interrupts!



Interrupts!



Interrupts!




Why?
Why?
Why?
Oh, that's why.

Interrupt-timing Attacks

```
int now = rdtsc();
while (true) {
  int last = now;
  now = rdtsc();
  if ((now - last) > threshold) {
    reportEvent(now, now - last);
  }
}
```

Interrupt-timing Attacks

```
int now = rdtsc();
while (true) {
  int last = now;
  now = rdtsc();
  if ((now - last) > threshold) {
    reportEvent(now, now - last);
  }
}
```

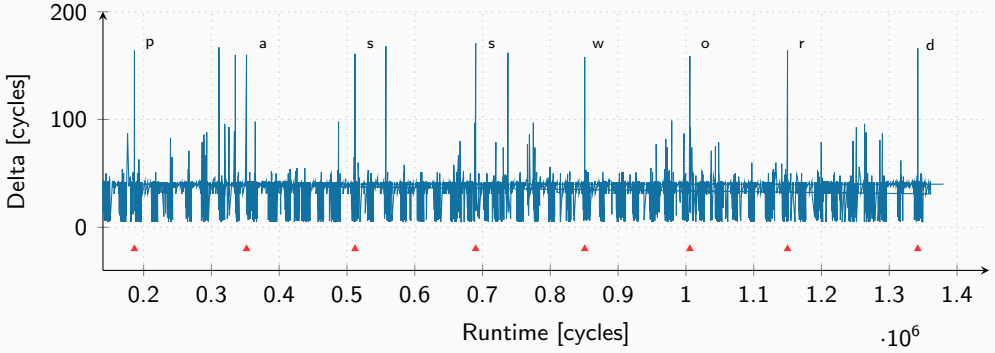
- Continuously acquire high-resolution timestamp

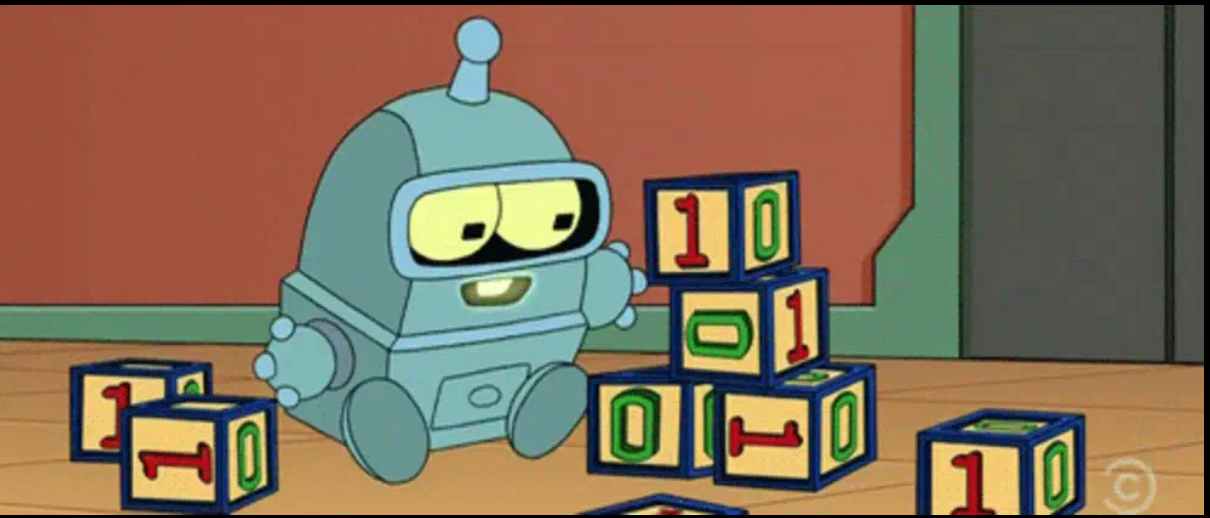
Interrupt-timing Attacks

```
int now = rdtsc();
while (true) {
  int last = now;
  now = rdtsc();
  if ((now - last) > threshold) {
    reportEvent(now, now - last);
  }
}
```

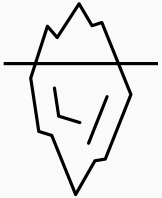
- Continuously acquire high-resolution timestamp
- Interrupt $\rightarrow$ large difference between timestamps

Interrupt-timing Attacks




1 0
0 1
1 0
1 0
1 0

Is that everything?



- Explains last experiment...
- ...but what about the simple calculation?
- Noise?

Data Location

- Memory operations have different runtimes

Data Location

- Memory operations have different runtimes
- Depends where the data is located

Data Location

- Memory operations have different runtimes
- Depends where the data is located
  - → Loading from cache: fast

Data Location

- Memory operations have different runtimes
- Depends where the data is located
  - → Loading from cache: fast
  - → Loading from memory: slow

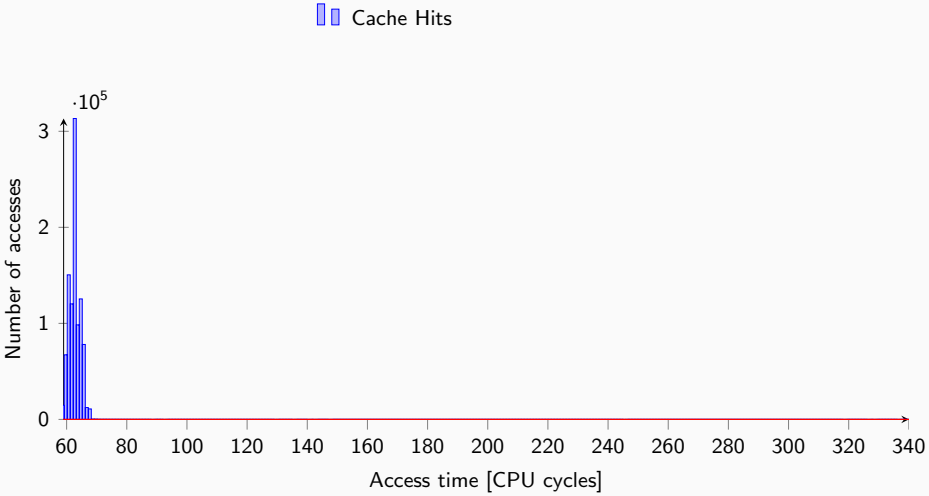
Data Location

- Memory operations have different runtimes
- Depends where the data is located
  - → Loading from cache: fast
  - → Loading from memory: slow
  - → Loading from disk: extremely slow

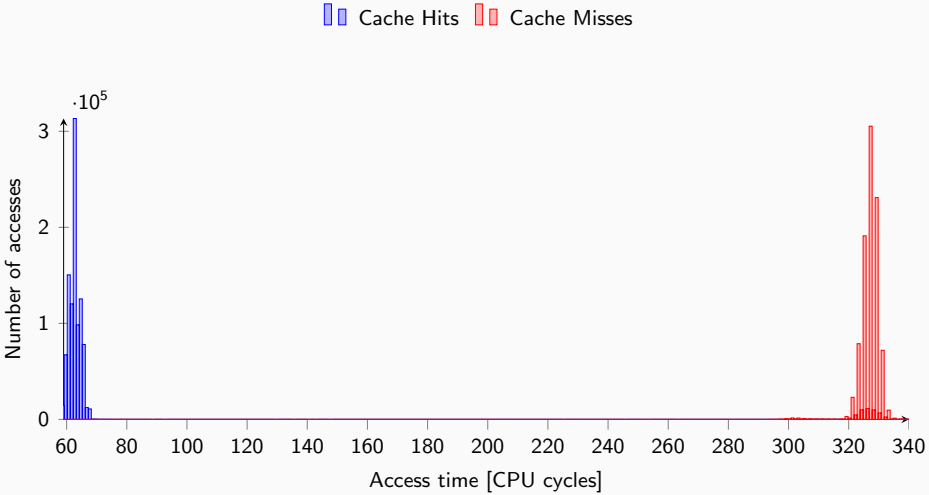
Data Location

- Memory operations have different runtimes
- Depends where the data is located
  - → Loading from cache: fast
  - → Loading from memory: slow
  - → Loading from disk: extremely slow
- Can also be measured

Caching Speeds-up Memory Accesses

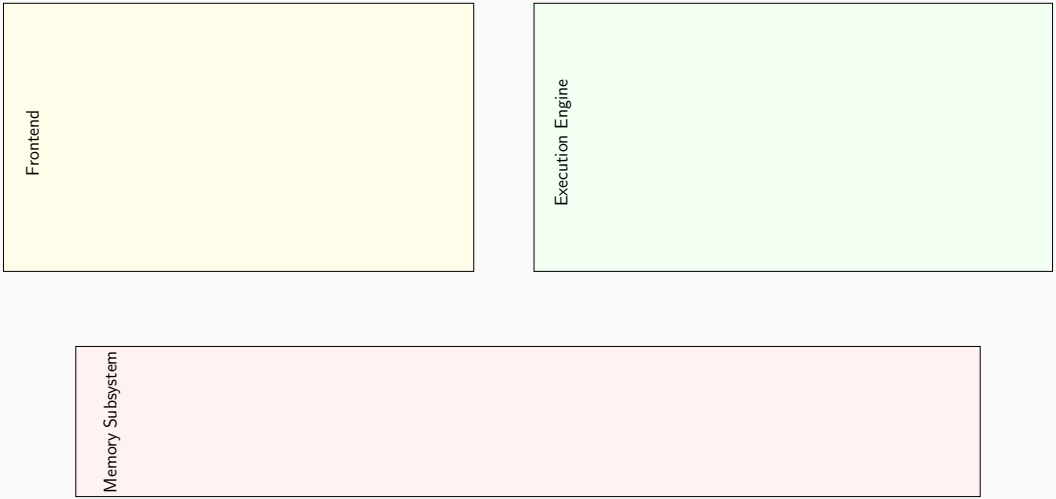


Caching Speeds-up Memory Accesses

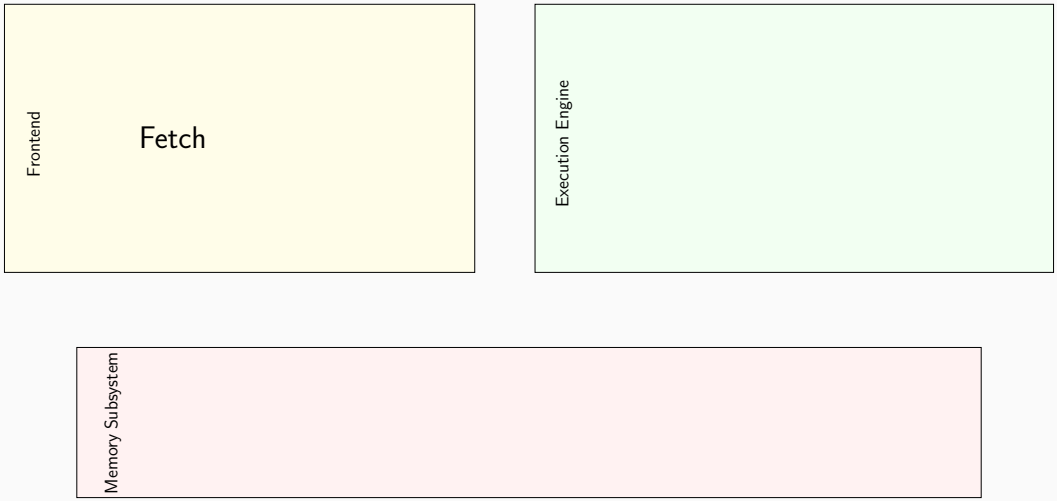



IS MY MENTAL MODEL OF THE CPU WRONG?
NO, IT MUST BE THE MEASUREMENTS THAT ARE WRONG
imgflip.com

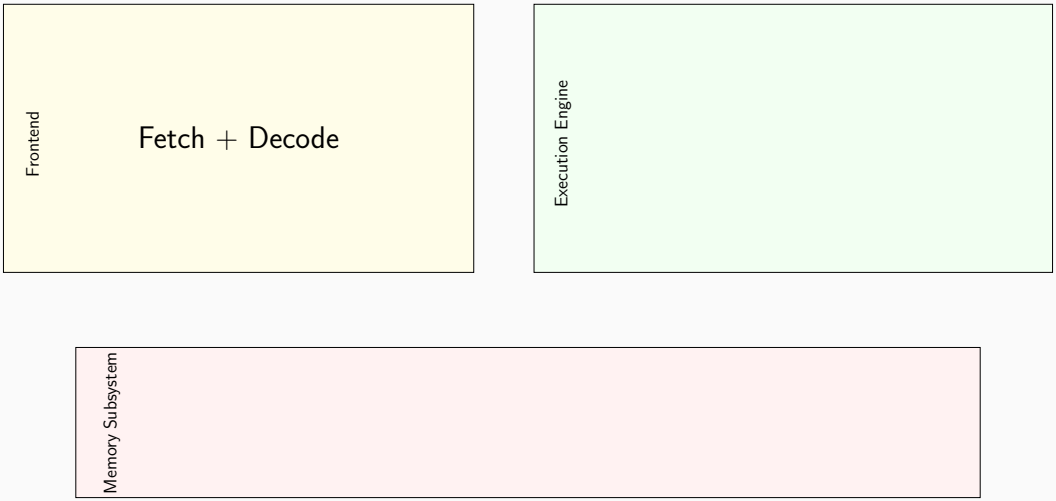
Reality: (Simplified) Modern CPU



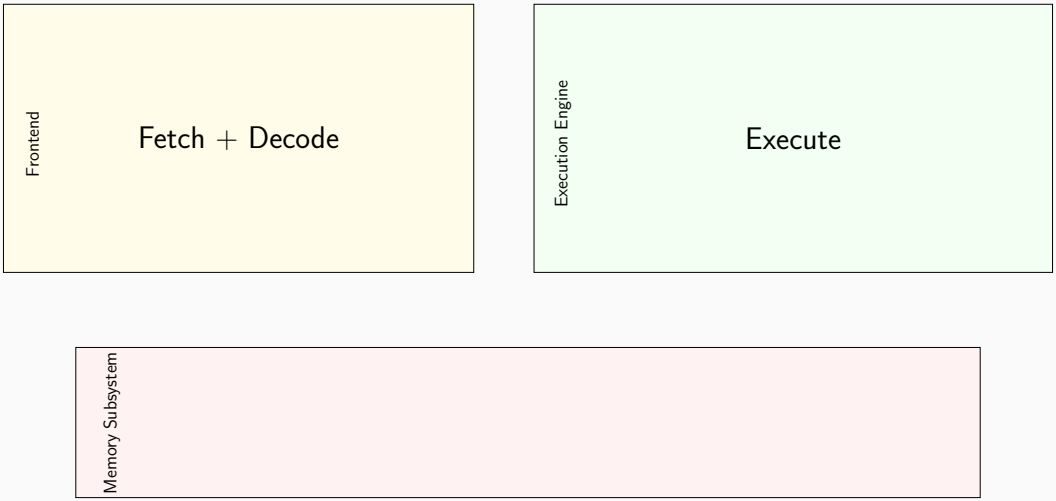
Reality: (Simplified) Modern CPU



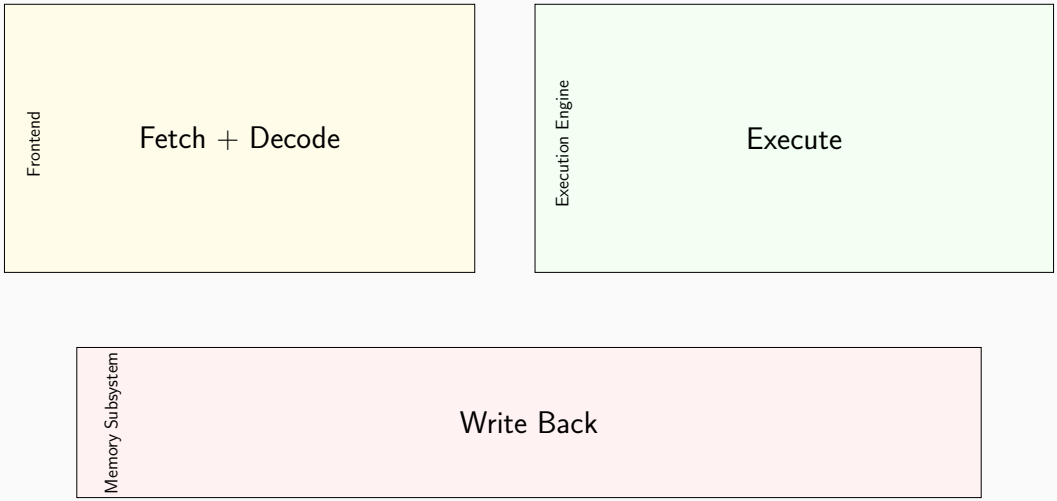
Reality: (Simplified) Modern CPU



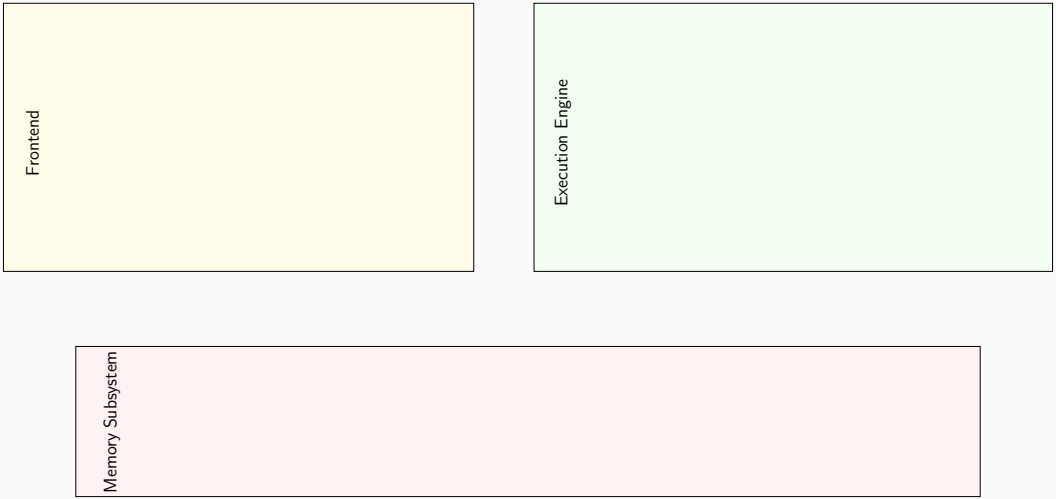
Reality: (Simplified) Modern CPU



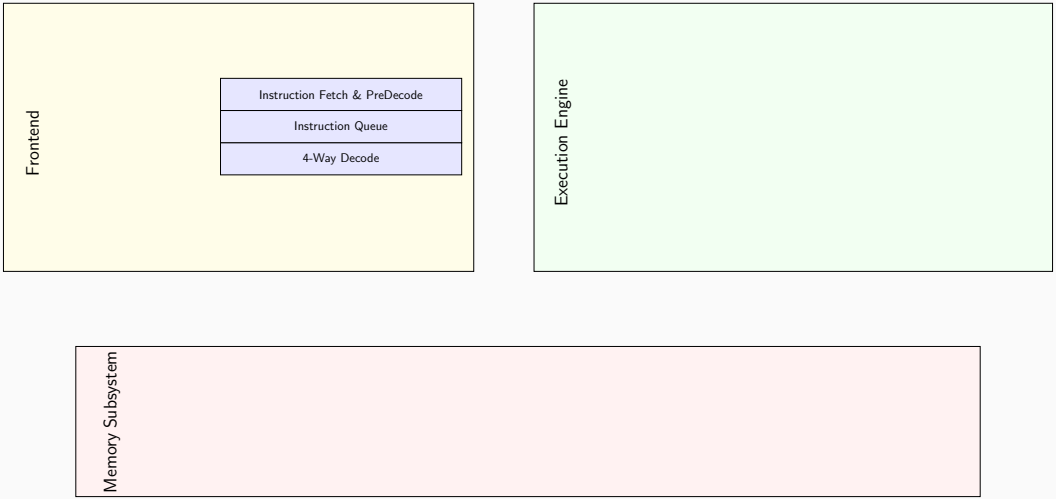
Reality: (Simplified) Modern CPU



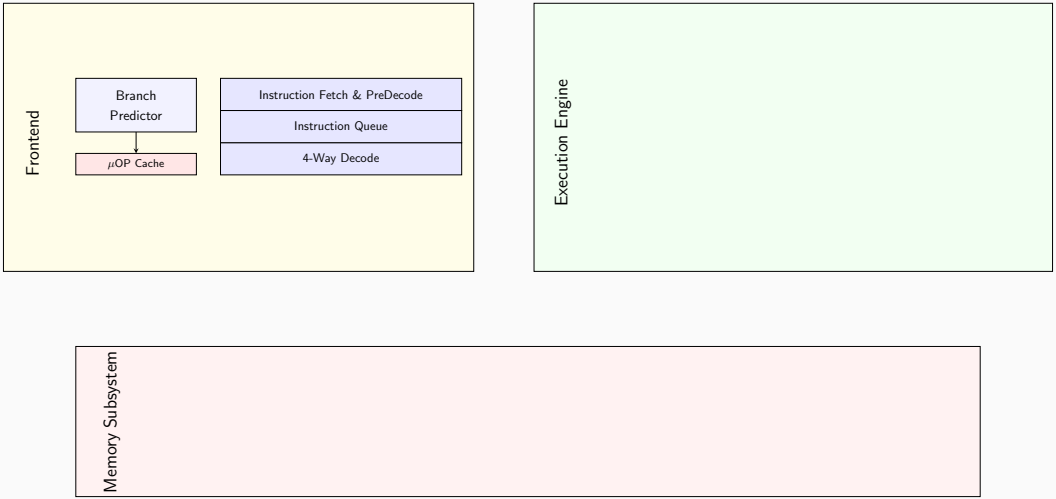
Reality: (Simplified) Modern CPU



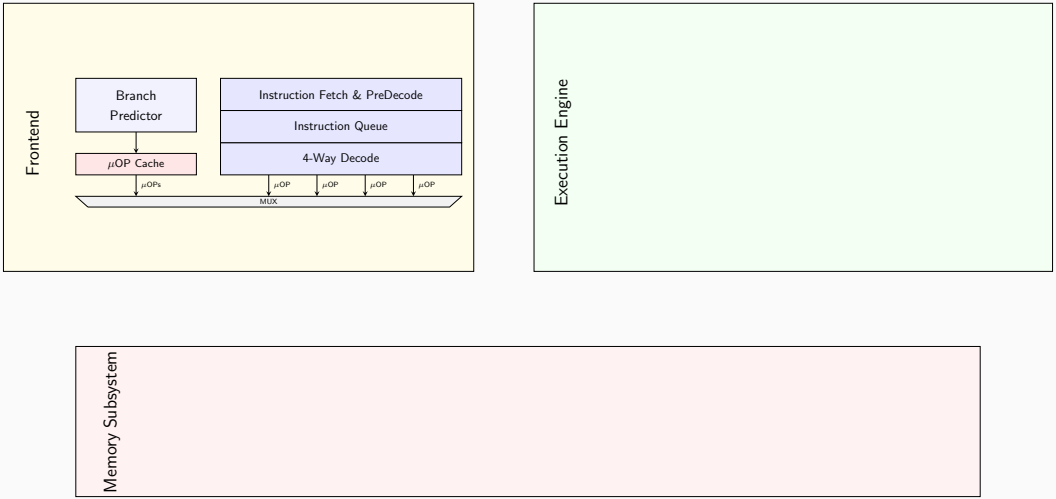
Reality: (Simplified) Modern CPU



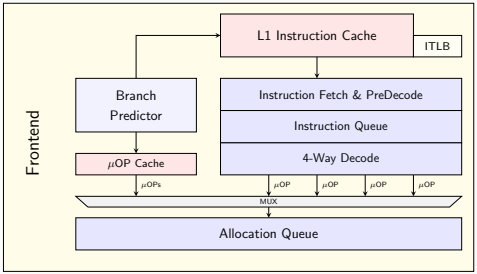
Reality: (Simplified) Modern CPU

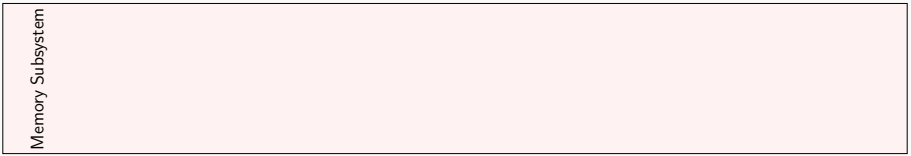


Reality: (Simplified) Modern CPU

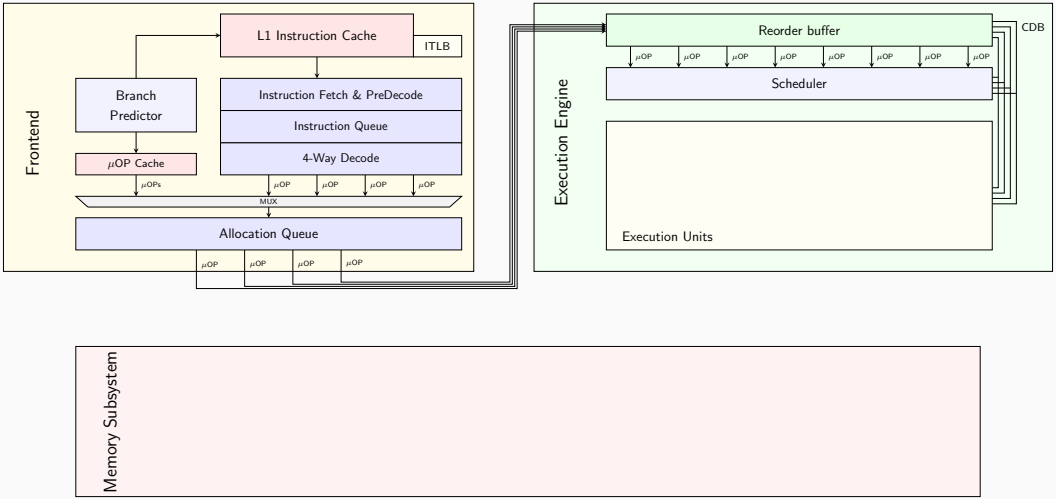


Reality: (Simplified) Modern CPU

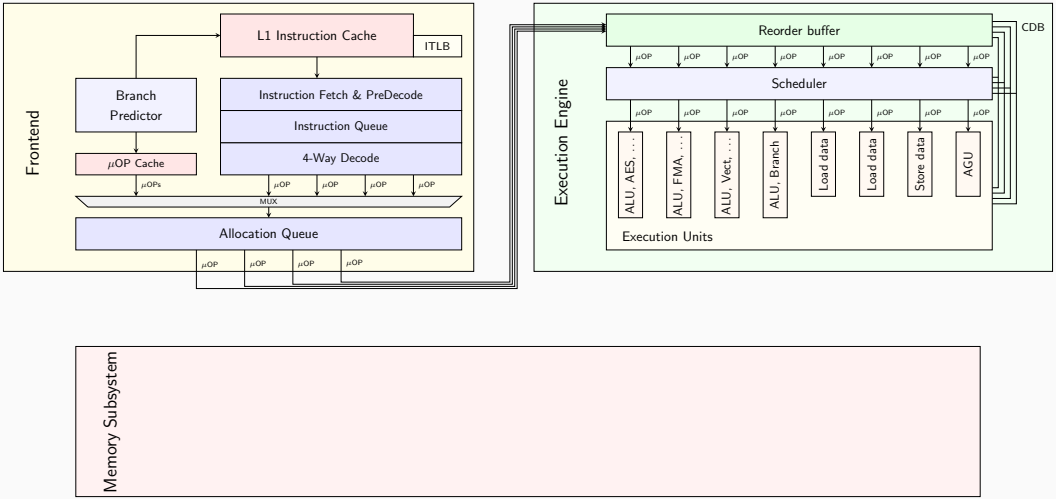




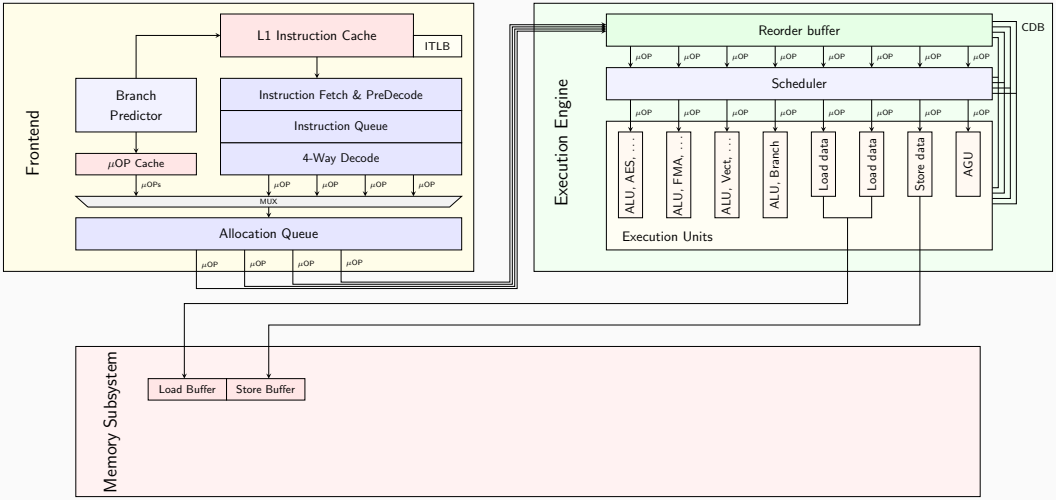
Reality: (Simplified) Modern CPU



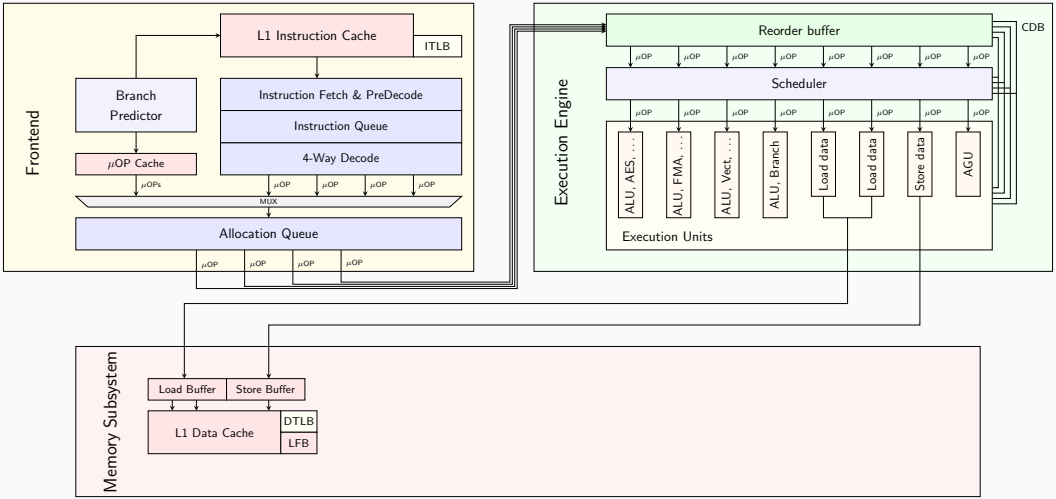
Reality: (Simplified) Modern CPU



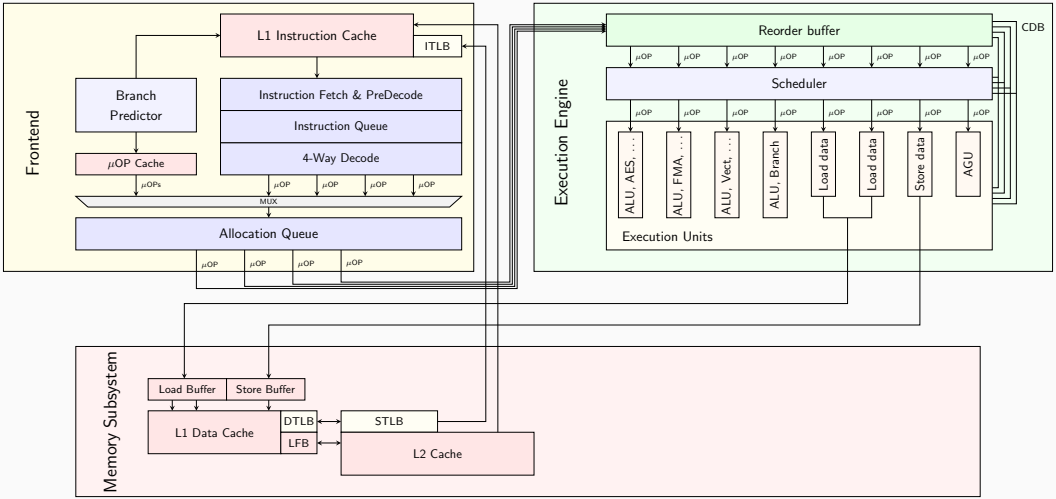
Reality: (Simplified) Modern CPU



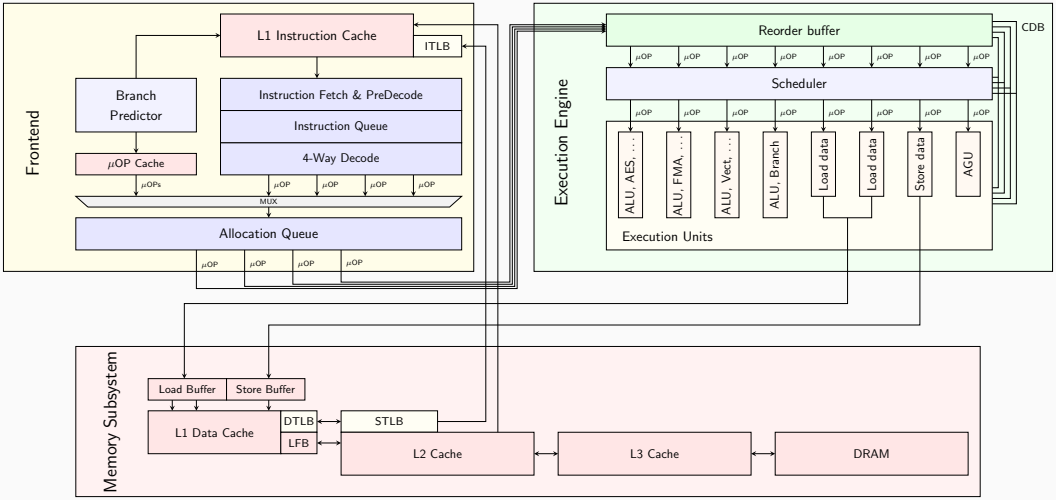
Reality: (Simplified) Modern CPU



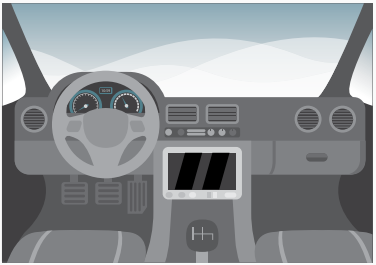
Reality: (Simplified) Modern CPU



Reality: (Simplified) Modern CPU



Intermezzo: CPU Architecture



- Cars all have the same interface (= architecture)

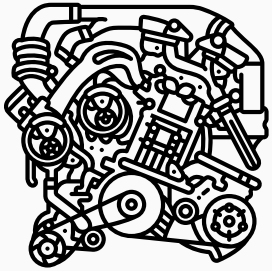
→ Steering wheel, pedals, gear stick, ...

- Some have special extensions

→ Air conditioning, cruise control, ...

- Driving skills are “compatible” with all cars

Intermezzo: CPU Microarchitecture



- Cars are implemented differently (= microarchitecture)

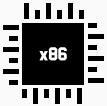
→ engine, fuel, motor control, ...

- Same car ("architecture") with different engines

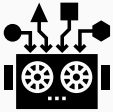
→ stronger or more efficient "microarchitecture"

- Drivers don't need to now anything about internals

No thorough Description



Intel manual
(full architecture)

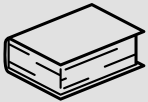


Intel optimization manual
(microarchitecture parts)

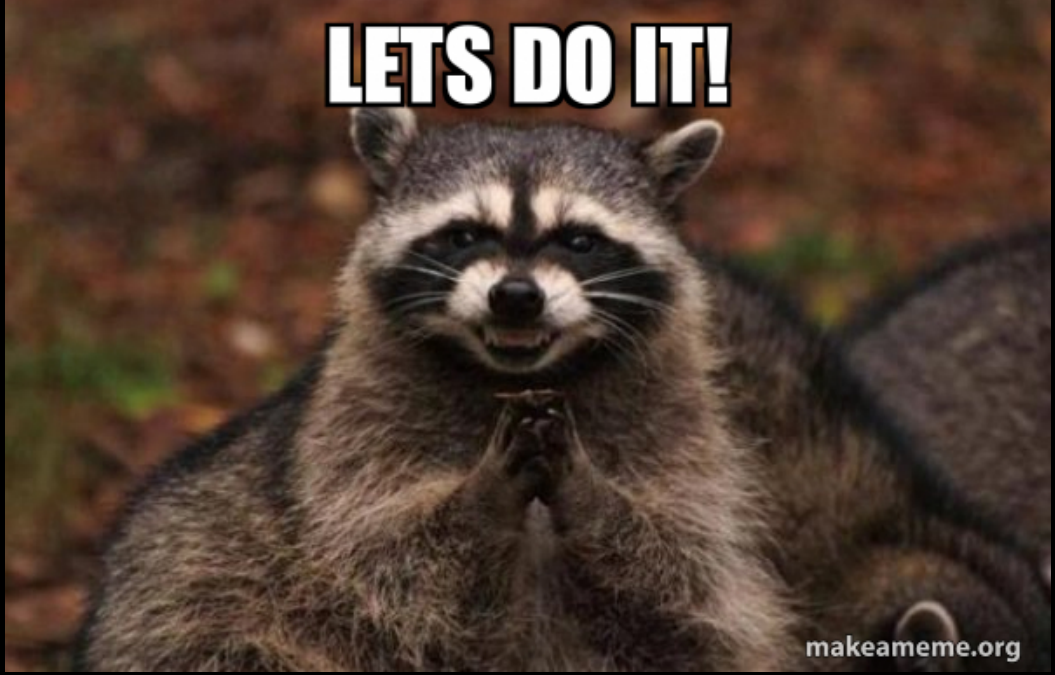
4778 pages

868 pages

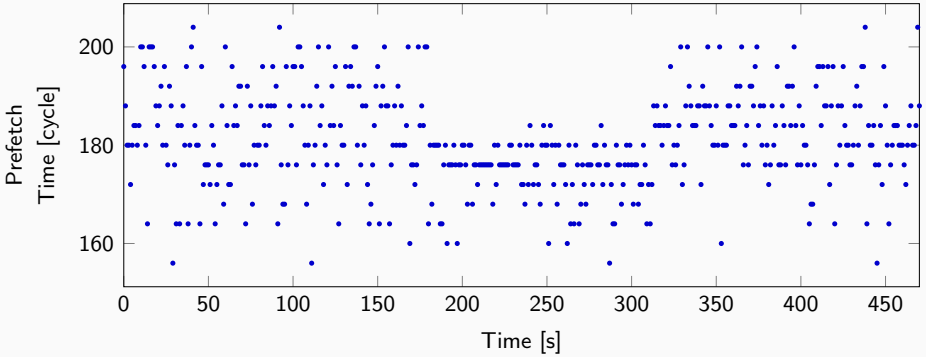
The `prefetch` Instruction



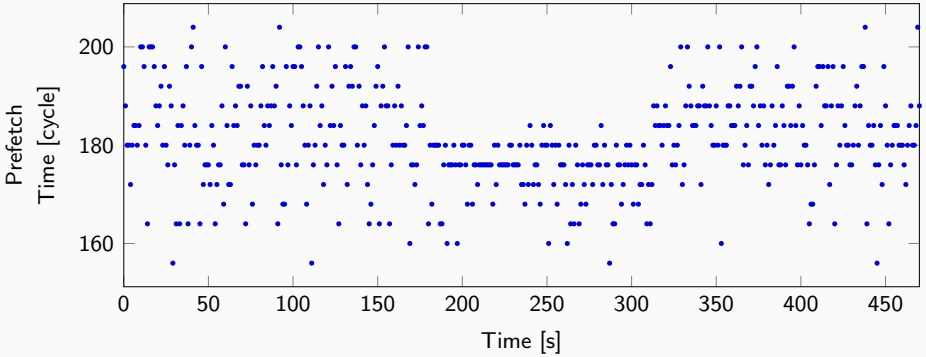
Use of software prefetch should be limited to memory addresses that are managed or owned within the application context. Prefetching to addresses that are not mapped to physical pages can experience non-deterministic performance penalty.


LETS DO IT!
makeameme.org

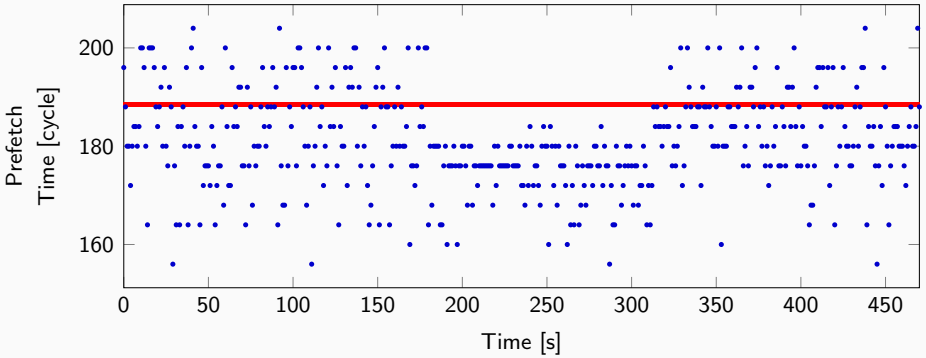
Prefetch Timings on the Operating System



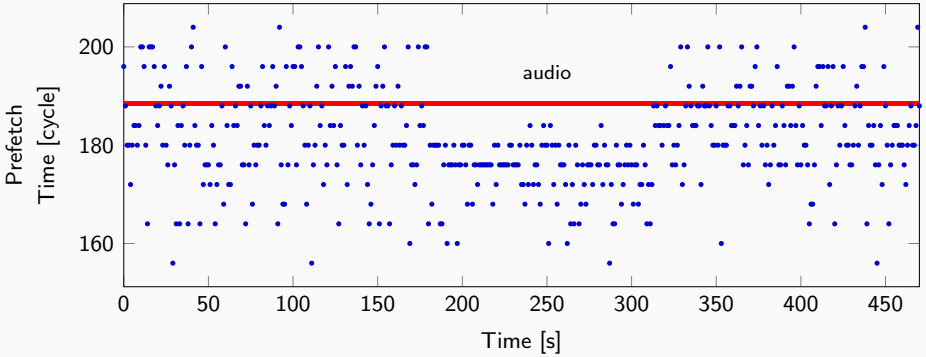
Prefetch Timings on the Operating System

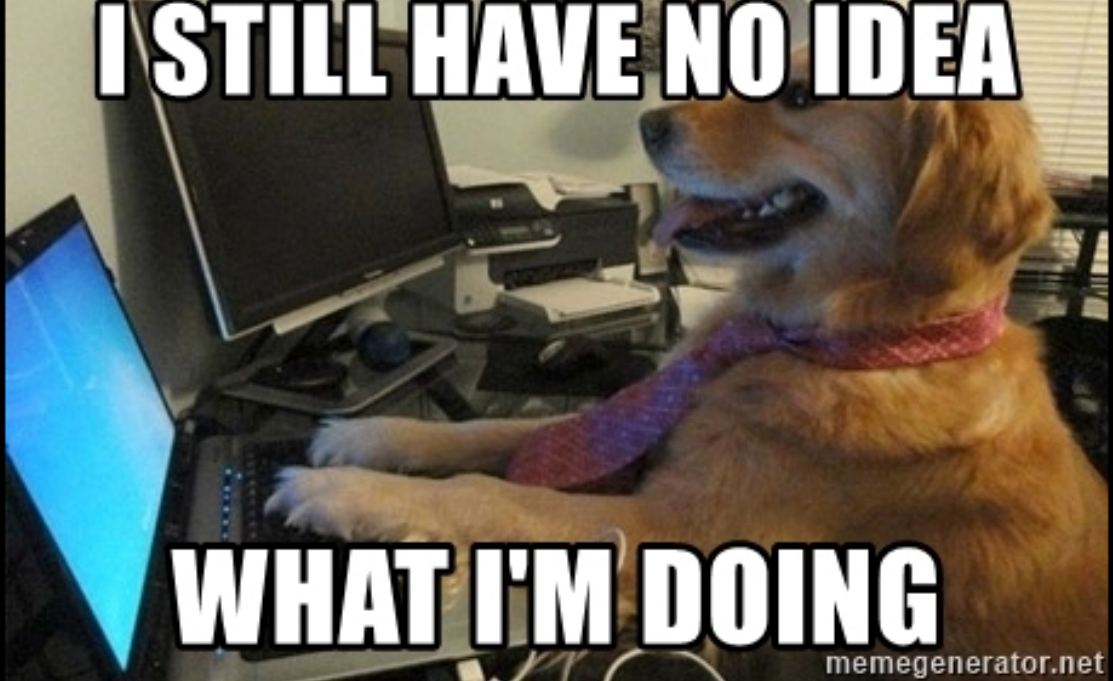


Prefetch Timings on the Operating System



Prefetch Timings on the Operating System




I STILL HAVE NO IDEA
WHAT I'M DOING
memegenerator.net

Systematic Approach

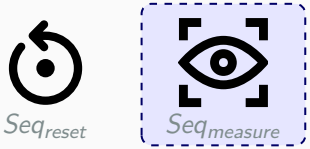
Reset instruction

Instruction to measure

Instruction with possible side effect

Testing A Sequence Triple

Testing A Sequence Triple



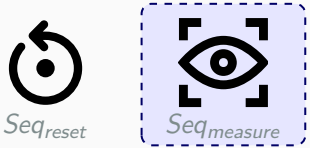
Testing A Sequence Triple

Seq_{reset}

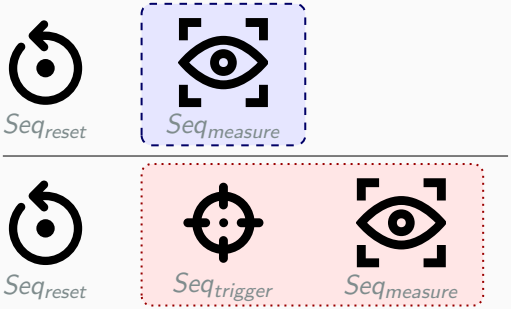
$Seq_{measure}$

Cold path S0

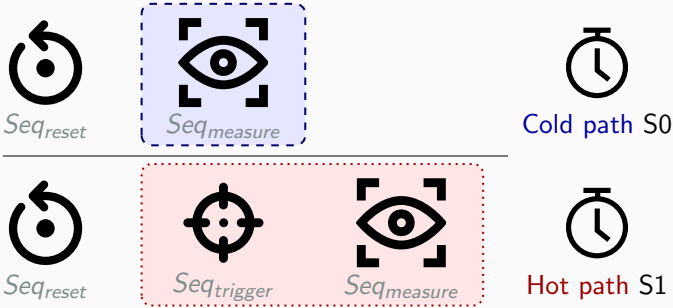
Testing A Sequence Triple



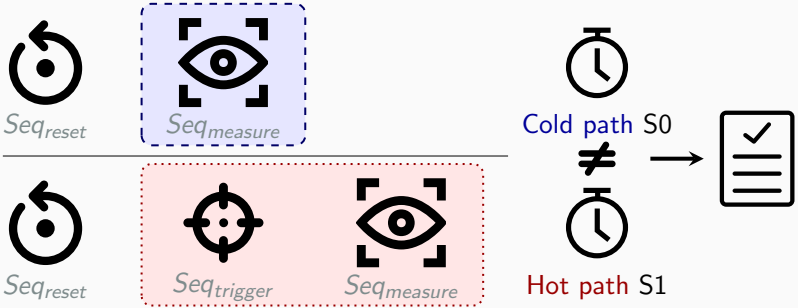
Testing A Sequence Triple



Testing A Sequence Triple

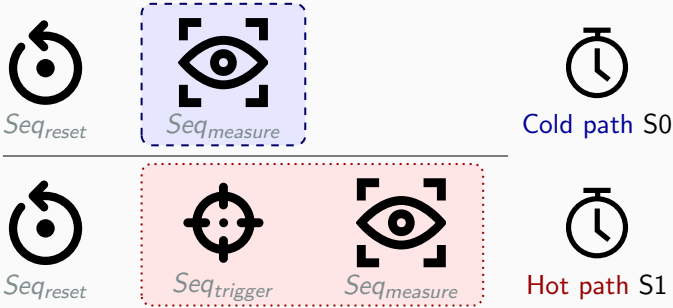


Testing A Sequence Triple



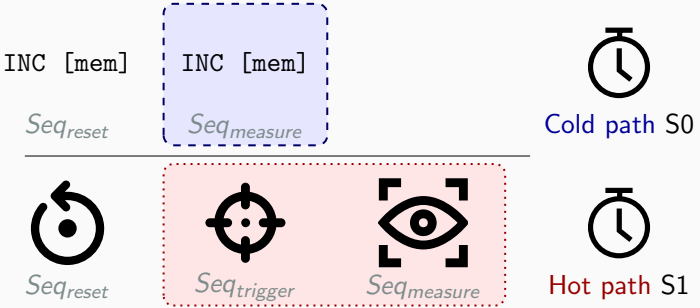
Testing A Sequence Triple

Example 1: $Seq_{measure} = Seq_{trigger} = Seq_{reset} =$ `INC [mem]`



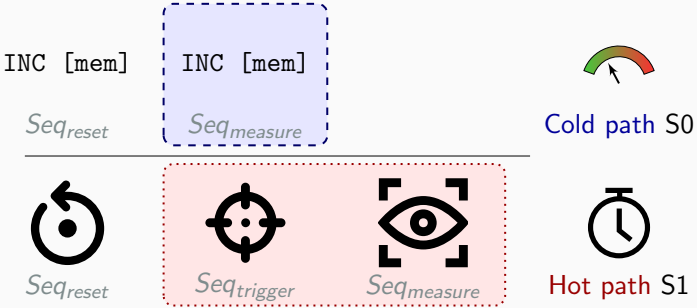
Testing A Sequence Triple

Example 1: $Seq_{measure} = Seq_{trigger} = Seq_{reset} =$ `INC [mem]`



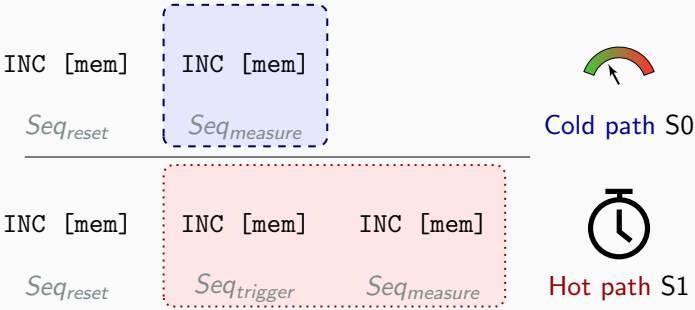
Testing A Sequence Triple

Example 1: $Seq_{measure} = Seq_{trigger} = Seq_{reset} =$ `INC [mem]`



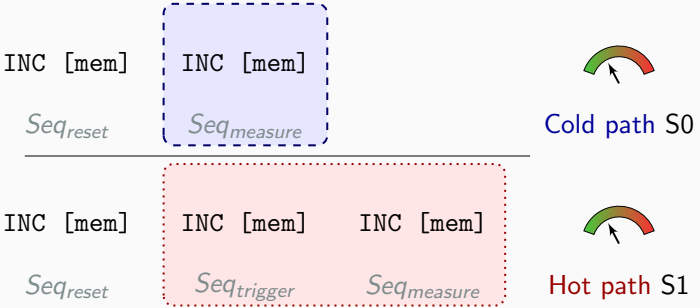
Testing A Sequence Triple

Example 1: $Seq_{measure} = Seq_{trigger} = Seq_{reset} =$ `INC [mem]`



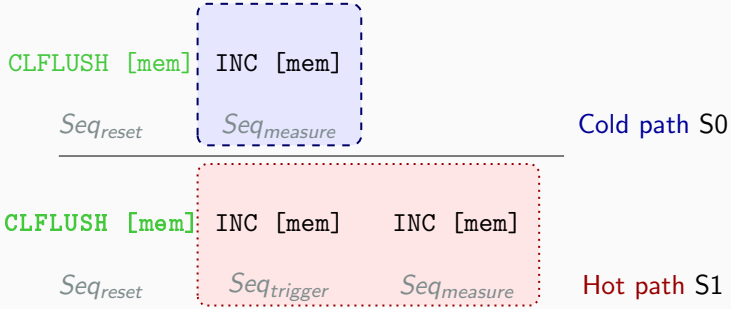
Testing A Sequence Triple

Example 1: $Seq_{measure} = Seq_{trigger} = Seq_{reset} =$ `INC [mem]`



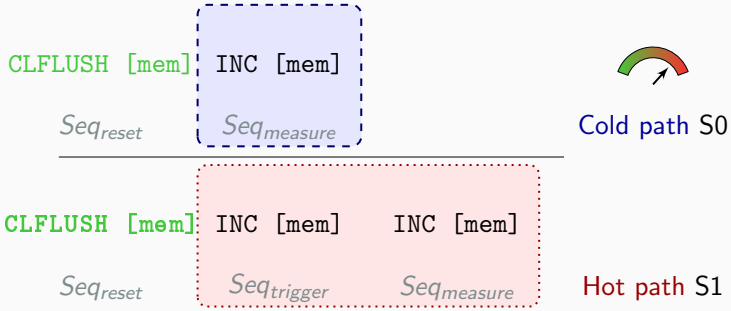
Testing A Sequence Triple

Example 2: $Seq_{measure} = Seq_{trigger} =$ `INC [mem]`;
$Seq_{reset} =$ `CLFLUSH [mem]`



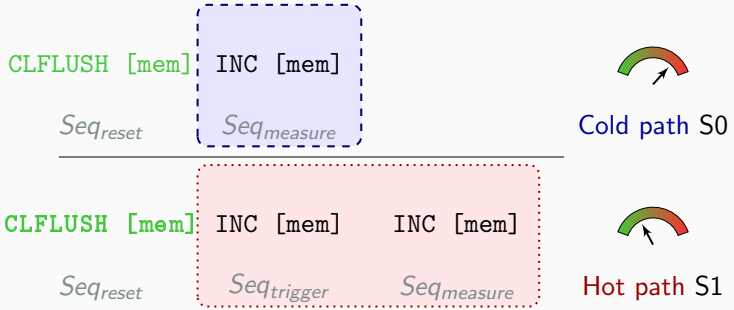
Testing A Sequence Triple

Example 2: $Seq_{measure} = Seq_{trigger} =$ `INC [mem]`;
$Seq_{reset} =$ `CLFLUSH [mem]`



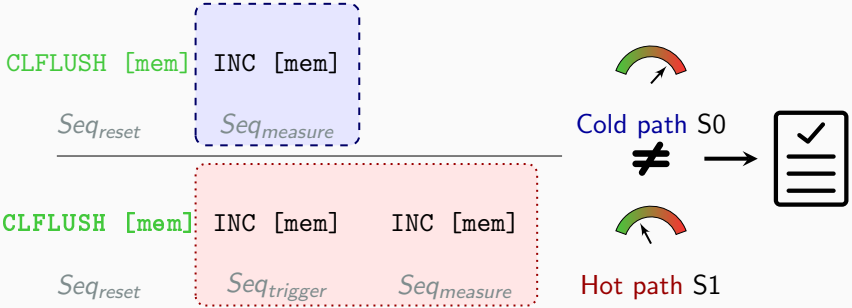
Testing A Sequence Triple

Example 2: $Seq_{measure} = Seq_{trigger} =$ `INC [mem]`;
$Seq_{reset} =$ `CLFLUSH [mem]`



Testing A Sequence Triple

Example 2: $Seq_{measure}$ = $Seq_{trigger}$ = `INC [mem]`;
Seq_{reset} = `CLFLUSH [mem]`



Recap: Measuring Time

```
start = ⏱
  x = y + 1
end = ⏱
Δ = end - start
```

Recap: Measuring Time

```
start = ⏱
  x = y + 1
end = ⏱
Δ = end - start
```

1. run: Δ = 302 $\rightarrow$ cache miss
2. run: Δ = 54 $\rightarrow$ cache hit

Recap: Measuring Time

```
clflush [y]
start = ⏱
  x = y + 1
end = ⏱
Δ = end - start
```

1. run: Δ = 302 $\rightarrow$ cache miss
2. run: Δ = 302 $\rightarrow$ cache miss

Recap: Measuring Time

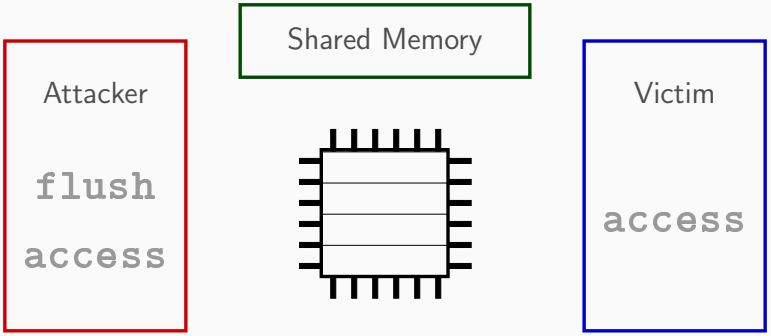
```
clflush [y]
start = ⏱
  x = y + 1
end = ⏱
Δ = end - start
```

1. run: Δ = 302 $\rightarrow$ cache miss
2. run: Δ = 302 $\rightarrow$ cache miss

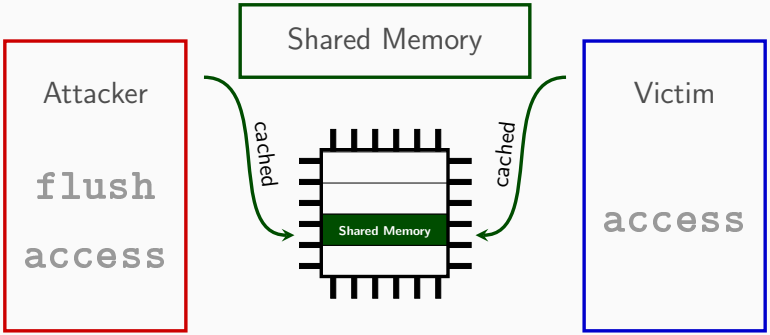
Determinism!

No randomness or non-determinism – just behavior we did not understand

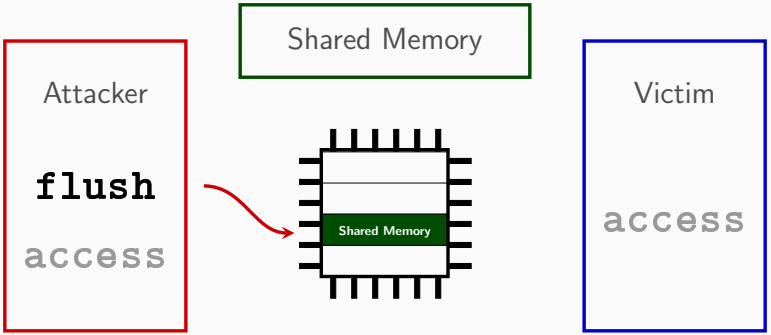
Flush+Reload



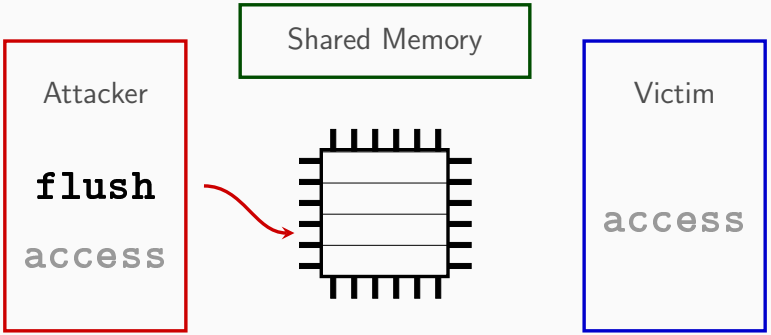
Flush+Reload



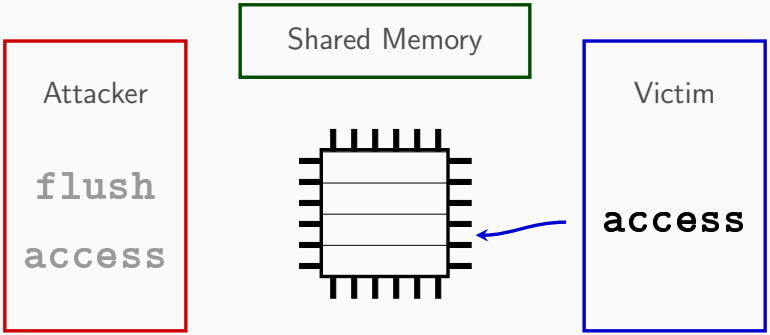
Flush+Reload



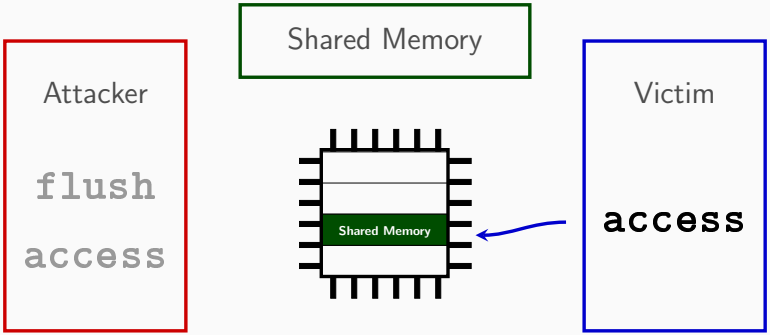
Flush+Reload



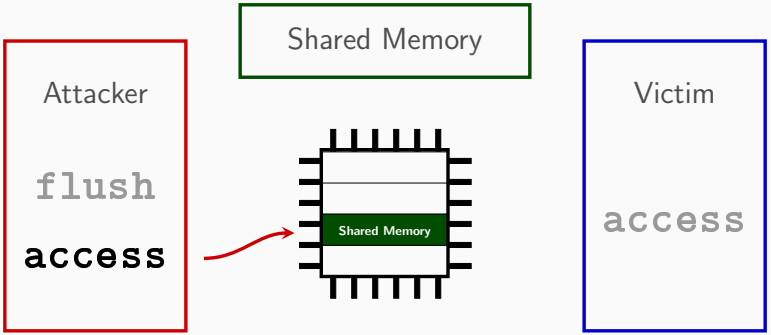
Flush+Reload



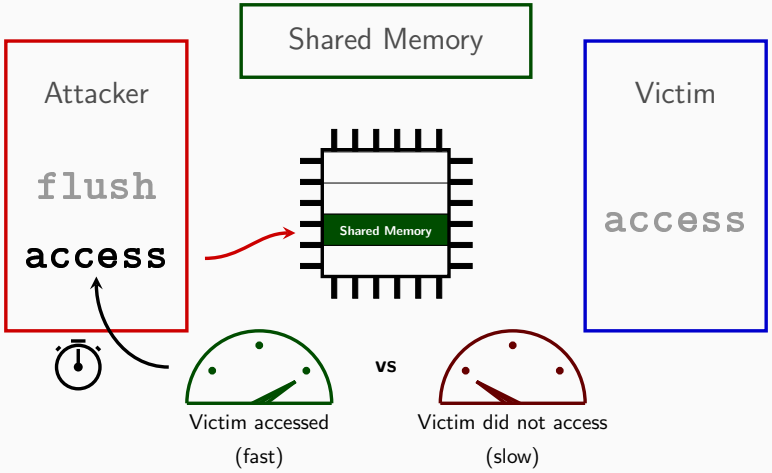
Flush+Reload



Flush+Reload



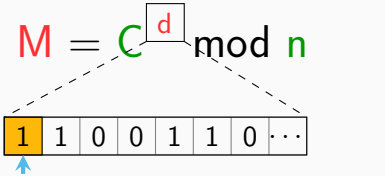
Flush+Reload



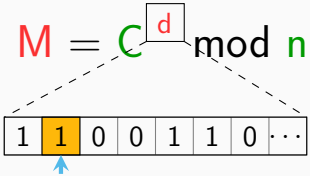
Flush+Reload on Square-and-Multiply

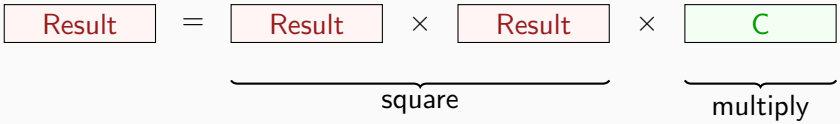
$$M = C^{d} \bmod n$$

Flush+Reload on Square-and-Multiply

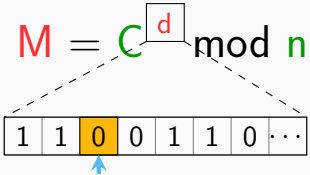


Flush+Reload on Square-and-Multiply





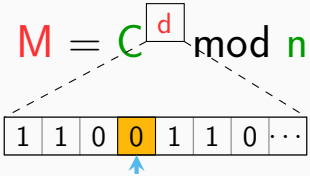
Flush+Reload on Square-and-Multiply



Result = Result × Result

square

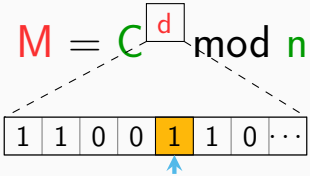
Flush+Reload on Square-and-Multiply

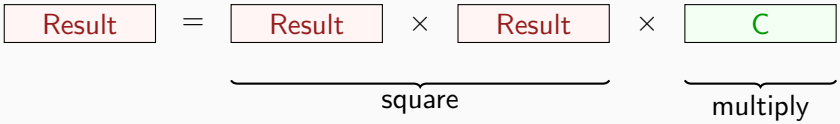


Result = Result × Result

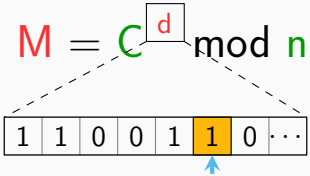
square

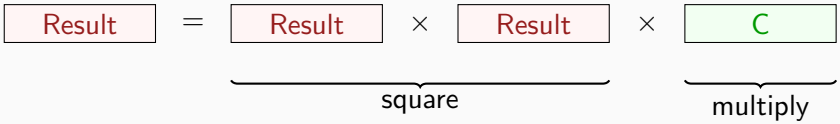
Flush+Reload on Square-and-Multiply



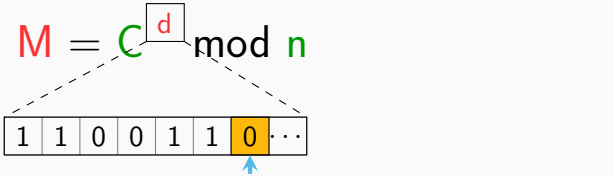


Flush+Reload on Square-and-Multiply





Flush+Reload on Square-and-Multiply



Motivation

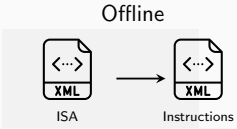
Problem

Finding side channels is a **complex** and **time-consuming** process

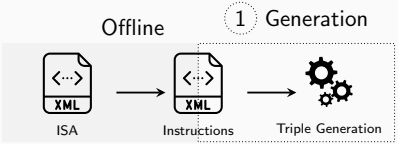
MEASURE

ALL THE THINGS

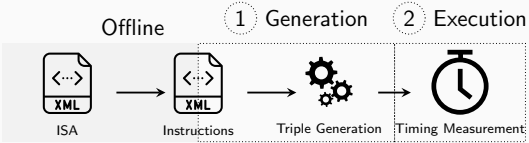
Osiris – Fuzzing x86 CPUs for Side Channels



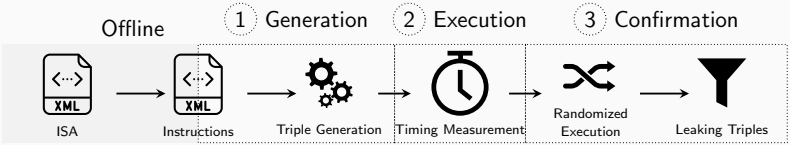
Osiris – Fuzzing x86 CPUs for Side Channels



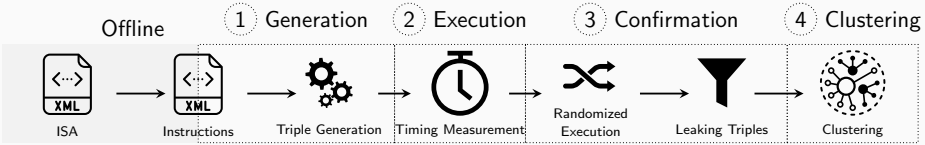
Osiris – Fuzzing x86 CPUs for Side Channels



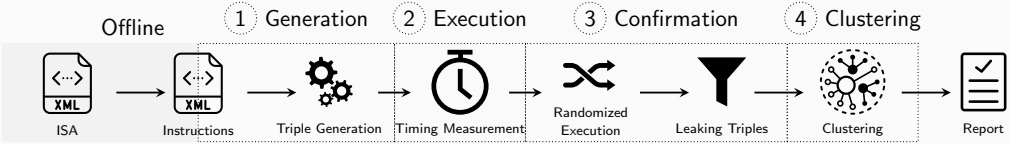
Osiris – Fuzzing x86 CPUs for Side Channels



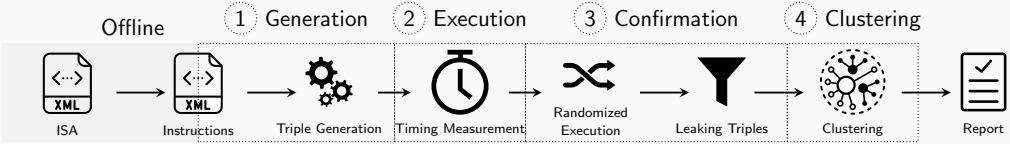
Osiris – Fuzzing x86 CPUs for Side Channels



Osiris – Fuzzing x86 CPUs for Side Channels



Osiris – Fuzzing x86 CPUs for Side Channels

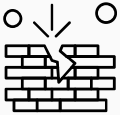


- Fuzzed on 5 different CPUs
- AMD and Intel

Osiris Results

~4 days per CPU

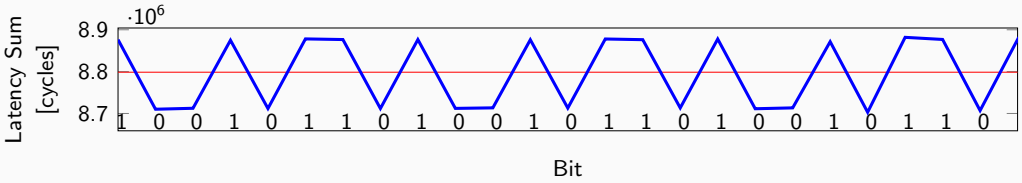
2 side channels rediscovered



4 new side channels

2 new attacks

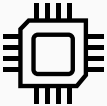
Cross-VM Interference



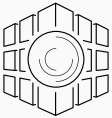
- Random-number generator `RDRAND`
- VM in the cloud (e.g., AWS) sees usage of other VMs

$\rightarrow$ Breaks VM isolation

RDRAND Covert Channel - Properties



AMD and Intel



VM and native

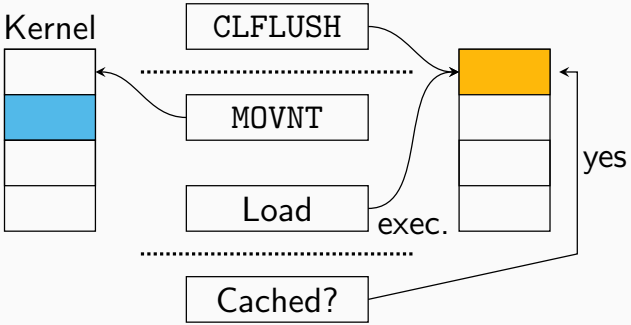
1000 bit/s

No memory

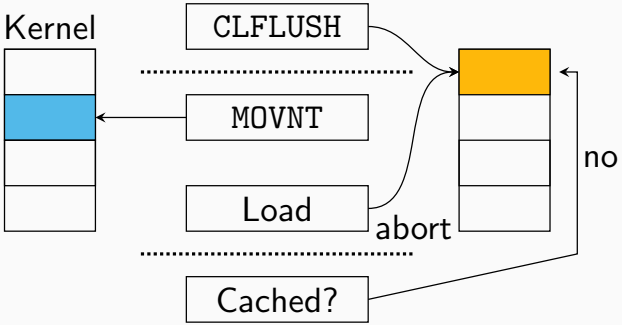
No detection

No mitigation

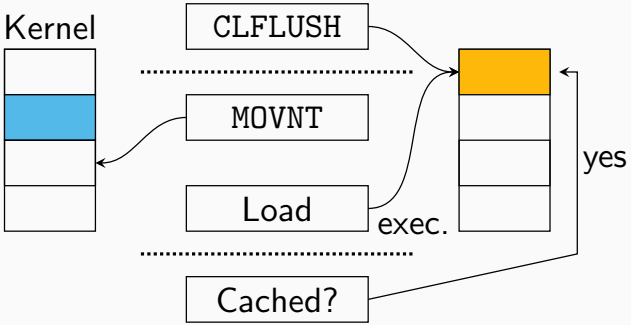
KASLR Break

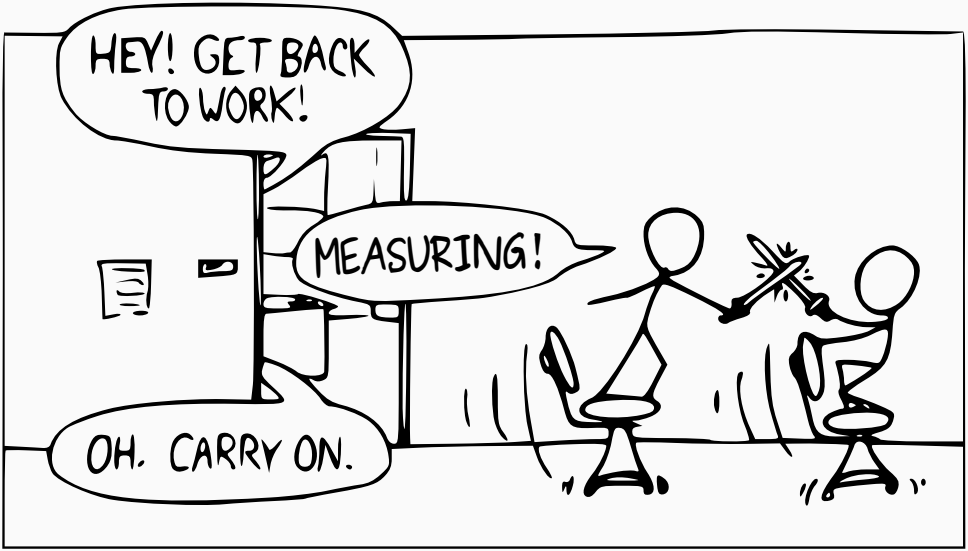


KASLR Break



KASLR Break




HEY! GET BACK TO WORK!
MEASURING!
OH. CARRY ON.

Continue Measuring

```
x = y + 1


start = ⏱
  x = y + 1
end = ⏱
Δ = end - start
```

Continue Measuring

```
x = y + 1
```

```
start = ⏱
  x = y + 1
end = ⏱
Δ = end - start
```

1. run: Δ = 54 $\rightarrow$ cache hit
2. run: Δ = 54 $\rightarrow$ cache hit

Continue Measuring

<crash>

x = y + 1 (never executed architecturally)

<restart>

start = ⏱

x = y + 1

end = ⏱

Δ = end - start

1. run:
2. run:

Continue Measuring


```
<crash>
x = y + 1  (never executed architecturally)
<restart>
start = ⏱
  x = y + 1
end = ⏱
Δ = end - start
```

1. run: Δ = 54 $\rightarrow$ cache hit
2. run: Δ = 54 $\rightarrow$ cache hit

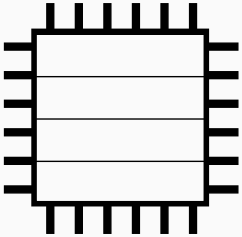
(Im)possible

Microarchitecture can do things impossible for the architecture

User Memory

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

```
char value = kernel[0]
```

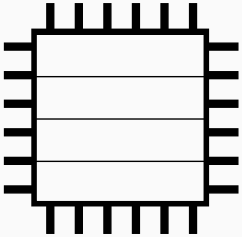


User Memory

	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)



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

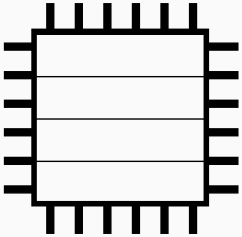
```
char value = kernel[0]
```

Page fault (Exception)

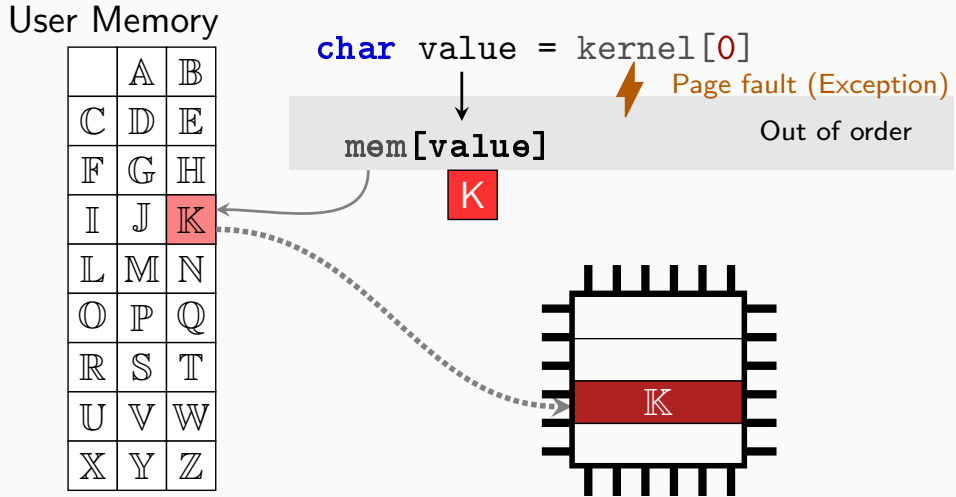
Out of order

```
mem[value]
```

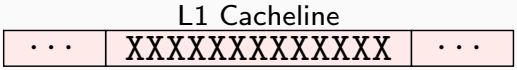
K



Meltdown

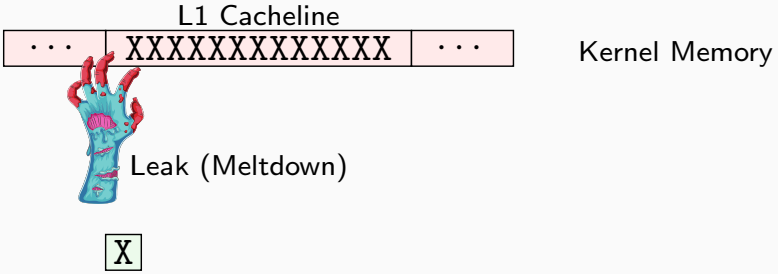

User Memory
char value = kernel[0]
Page fault (Exception)
Out of order
mem[value]
K
A B
C D E
F G H
I J K
L M N
O P Q
R S T
U V W
X Y Z
K

Meltdown Experiment

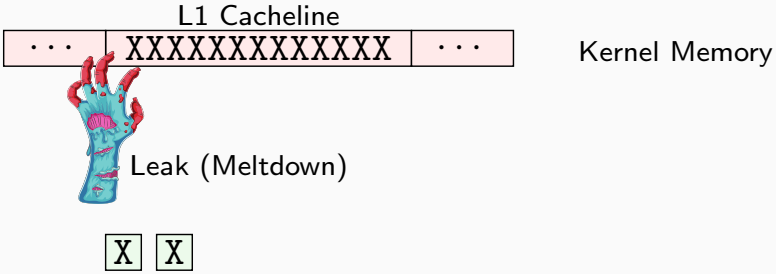


Kernel Memory

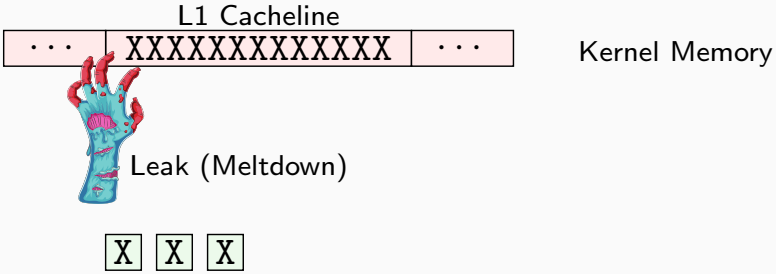
Meltdown Experiment



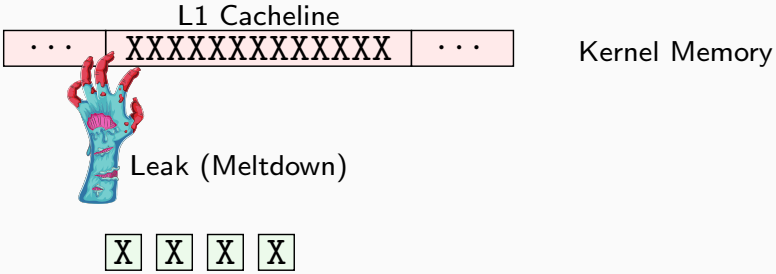
Meltdown Experiment



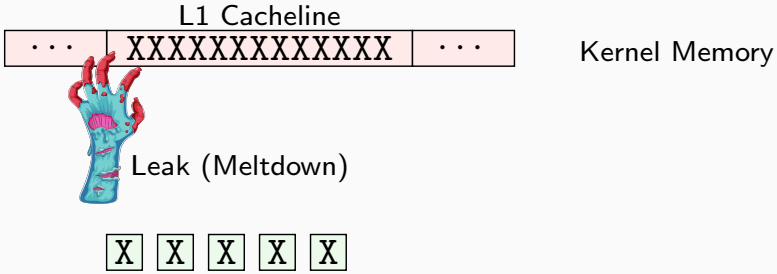
Meltdown Experiment



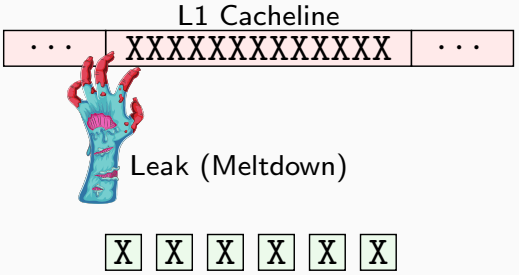
Meltdown Experiment



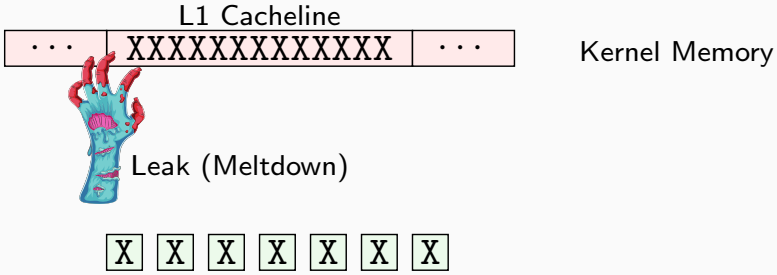
Meltdown Experiment



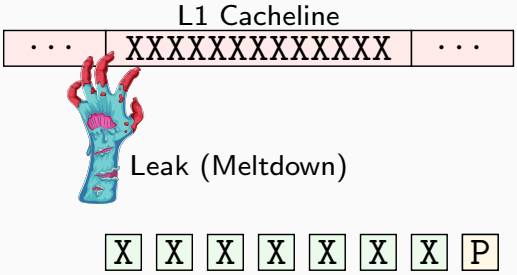
Meltdown Experiment



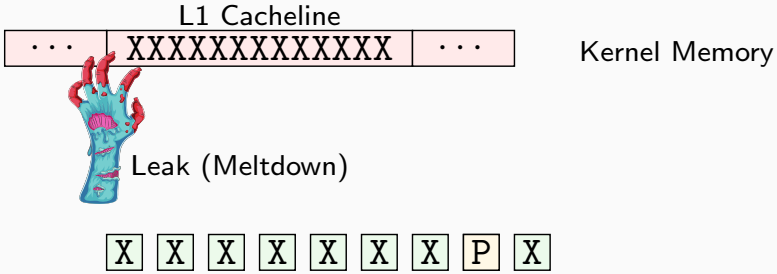
Meltdown Experiment



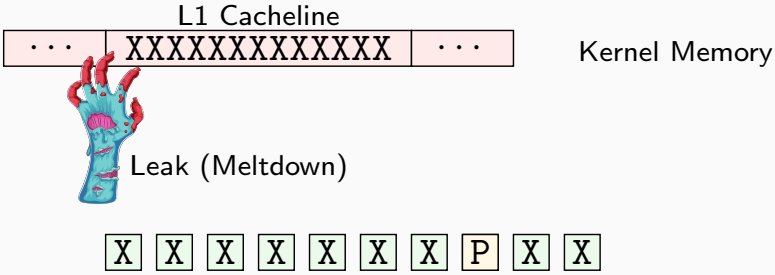
Meltdown Experiment



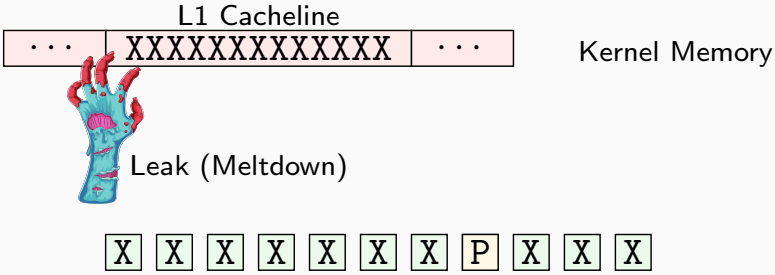
Meltdown Experiment



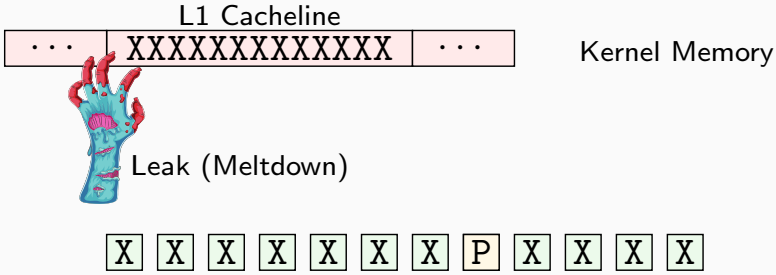
Meltdown Experiment



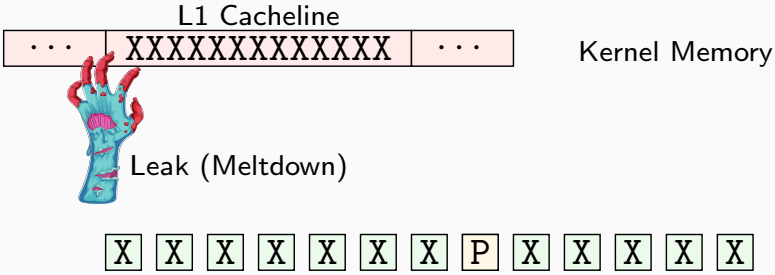
Meltdown Experiment



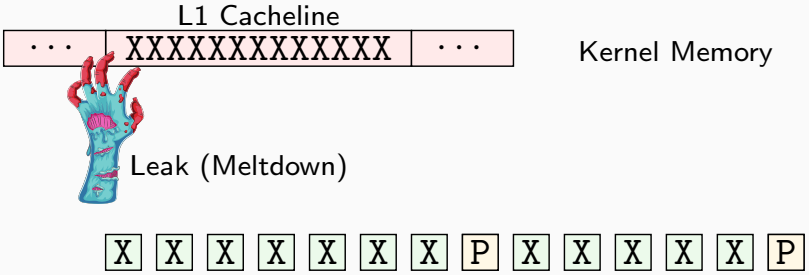
Meltdown Experiment



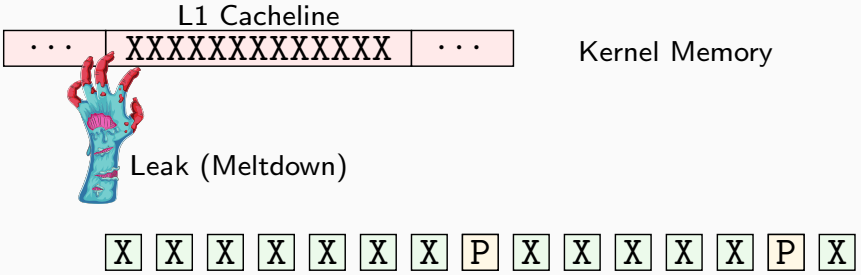
Meltdown Experiment



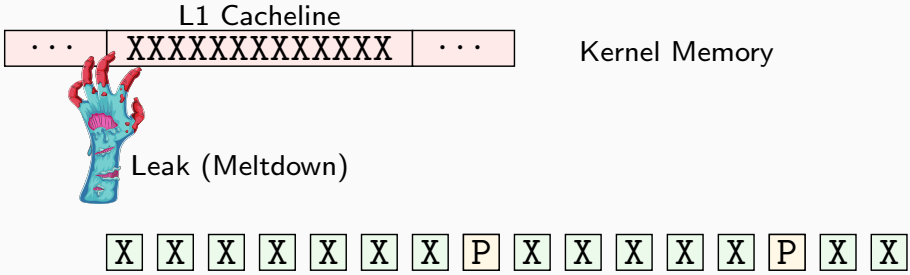
Meltdown Experiment



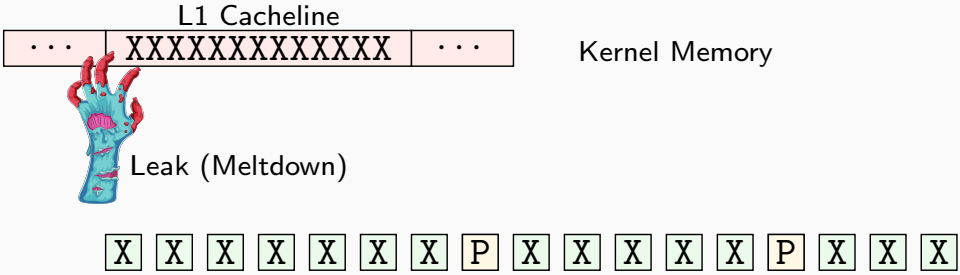
Meltdown Experiment



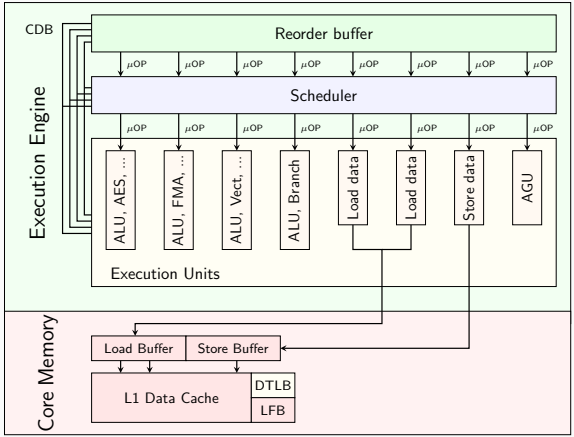
Meltdown Experiment



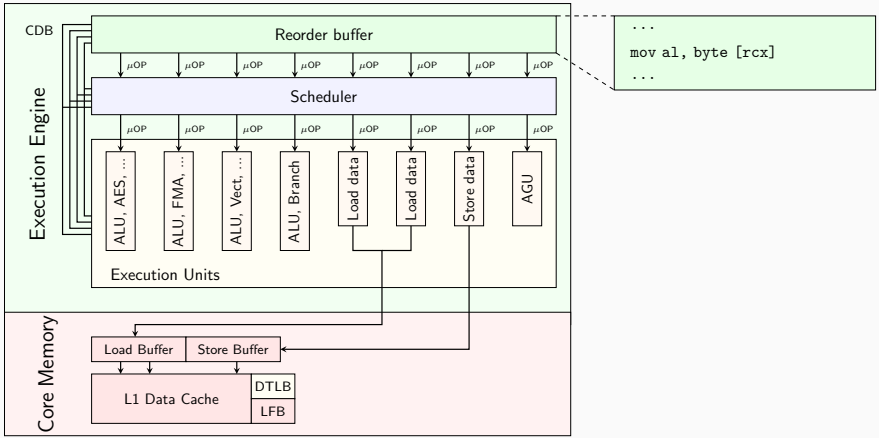
Meltdown Experiment



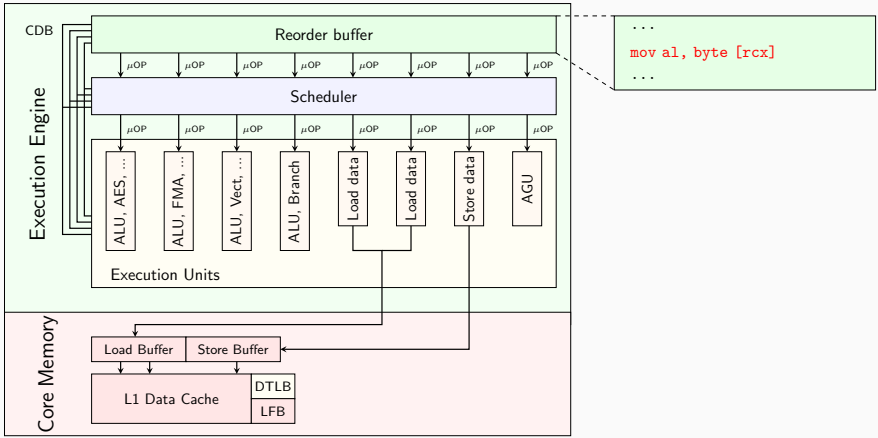
Complex Load Situations



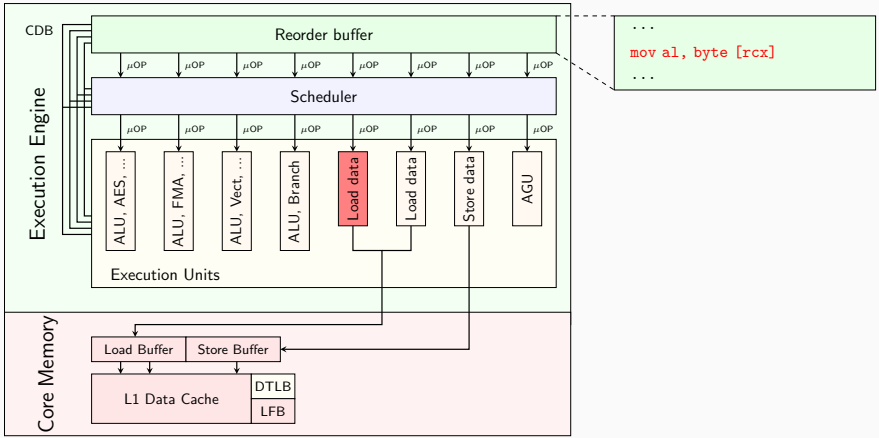
Complex Load Situations



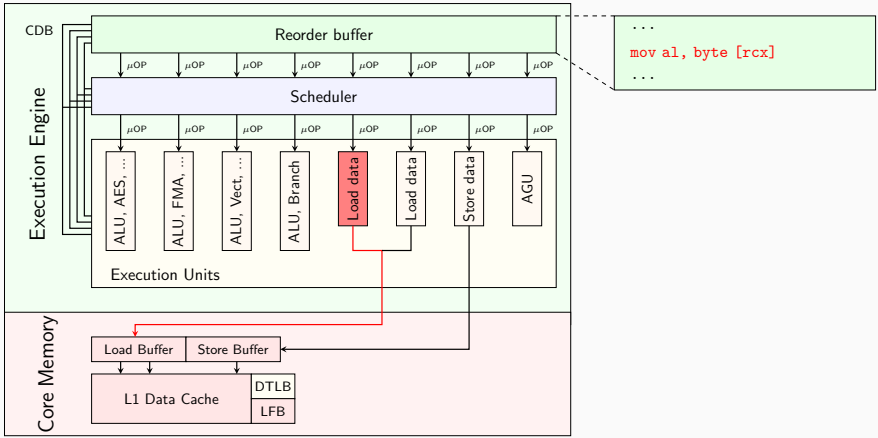
Complex Load Situations



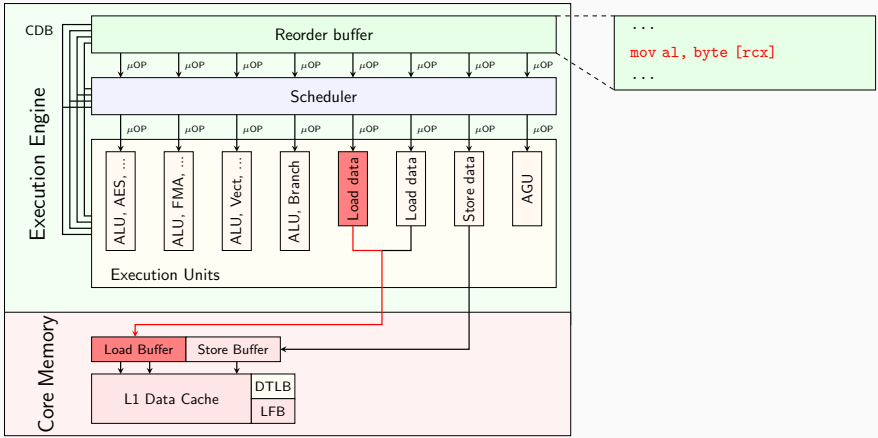
Complex Load Situations



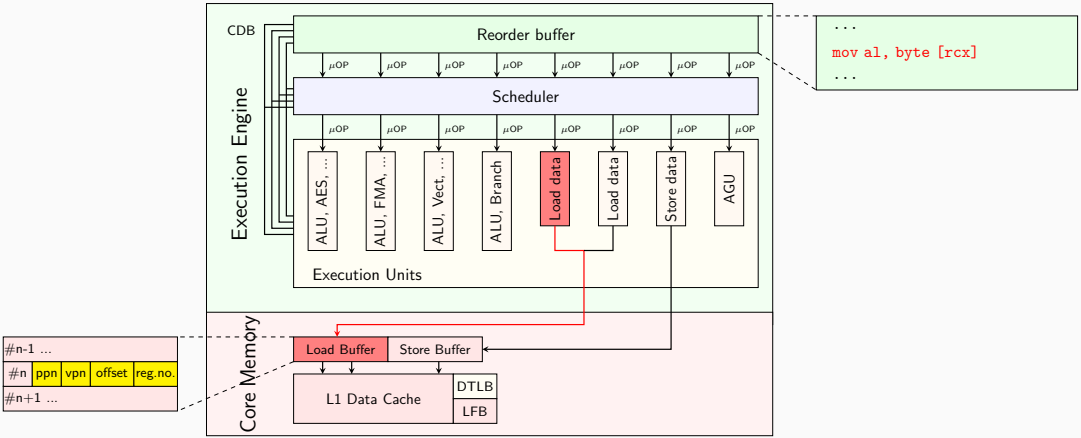
Complex Load Situations



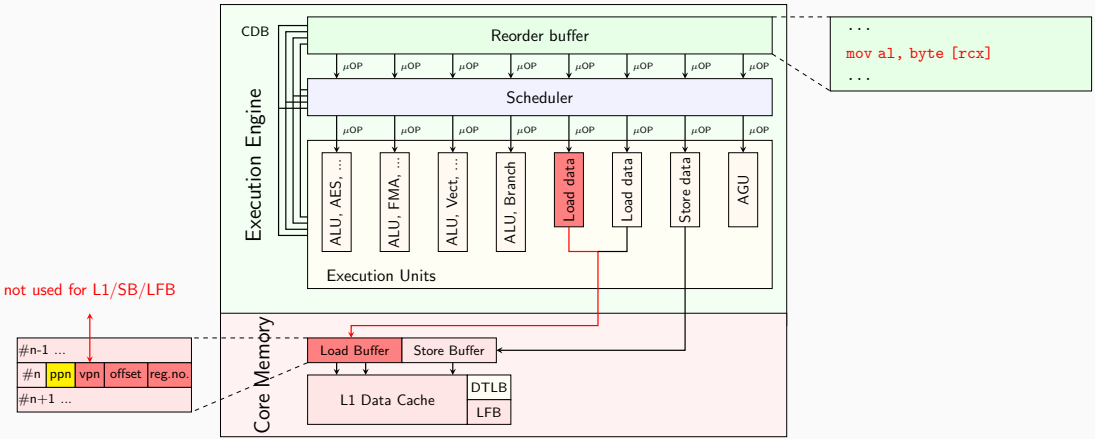
Complex Load Situations



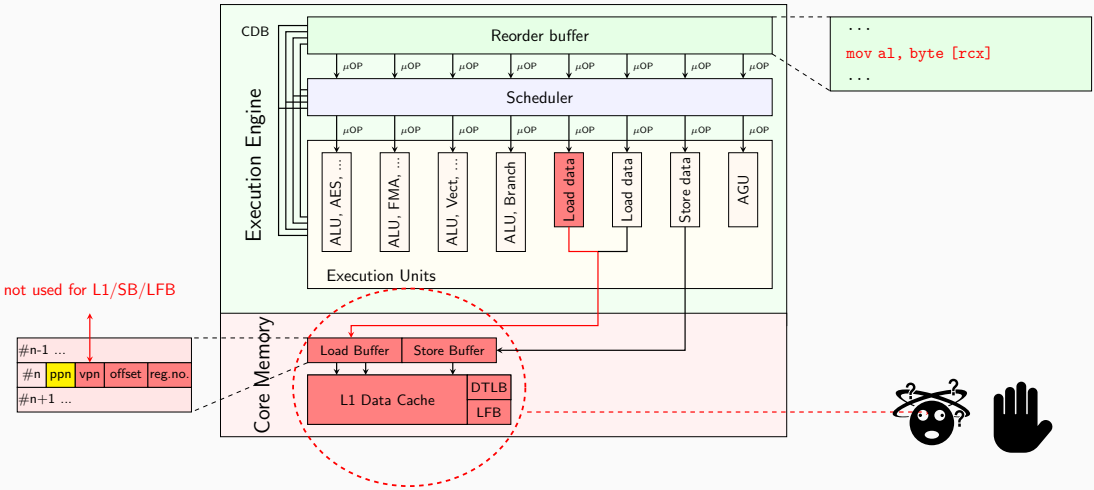
Complex Load Situations



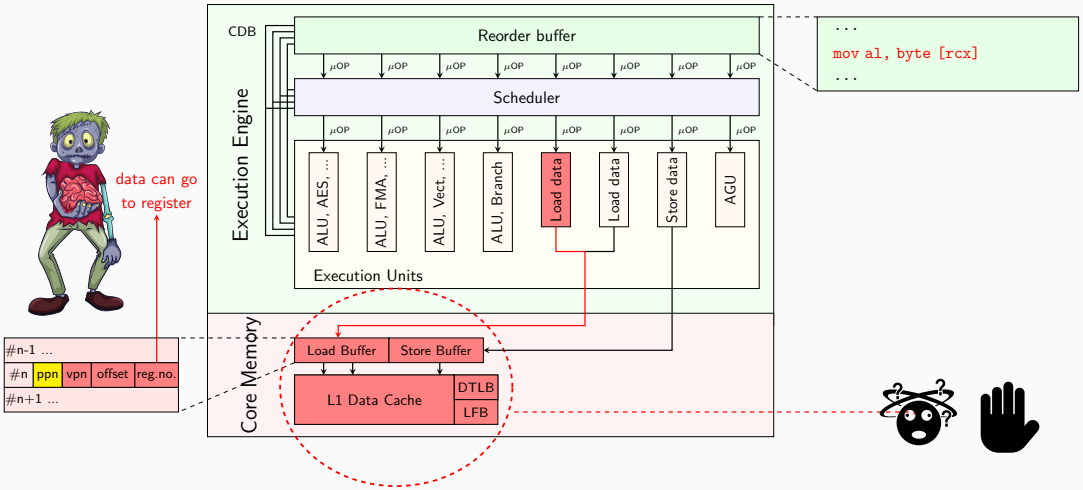
Complex Load Situations

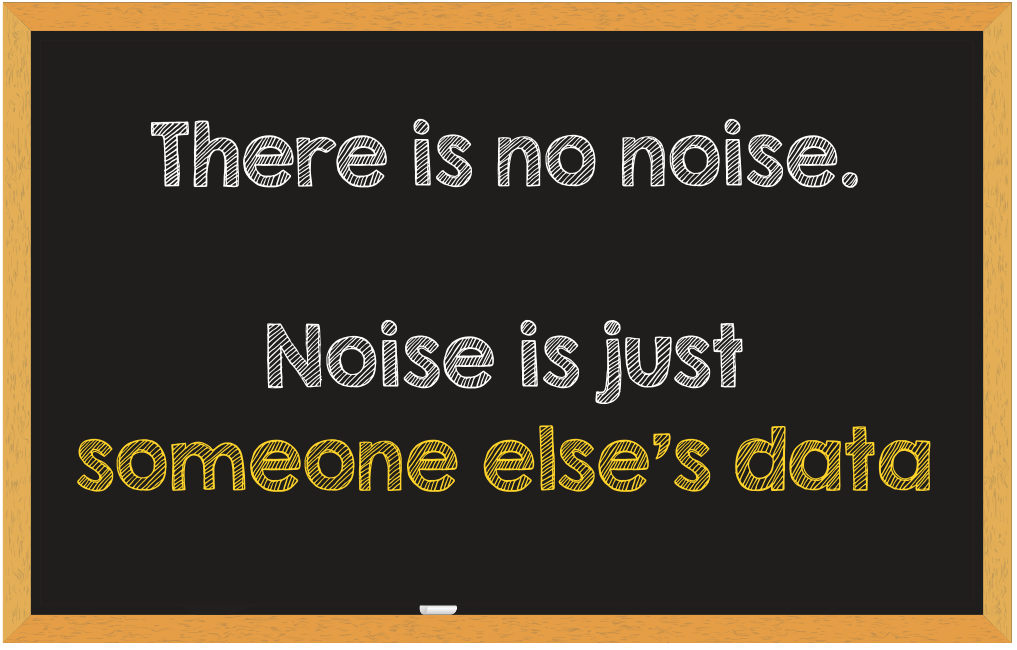


Complex Load Situations

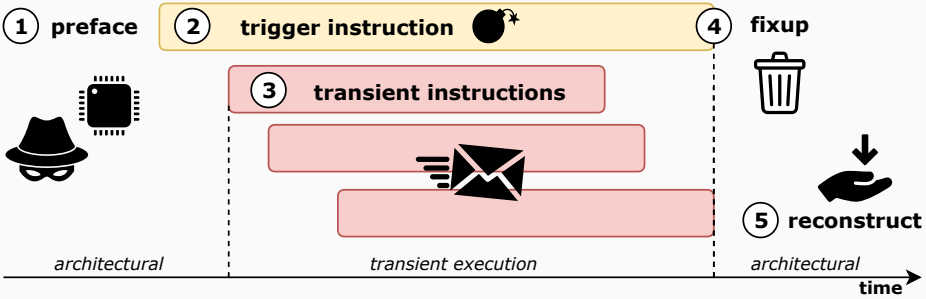


Complex Load Situations




There is no noise.
Noise is just
someone else's data

Systematic Analysis



BBC

Intel Zombieload bug fix to slow data centre computers

THE VERGE

ZombieLoad attack lets hackers steal data from Intel chips

FORTUNE

'Zombieload' Flaw Lets Hackers Crack Almost Every Intel Chip Back to 2011. Why's It Being Downplayed?

How-To Geek

Only New CPUs Can Truly Fix ZombieLoad and Spectre

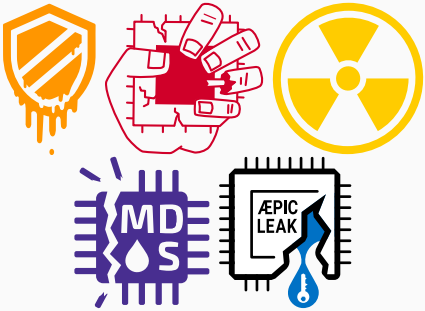
That Escalated Quickly

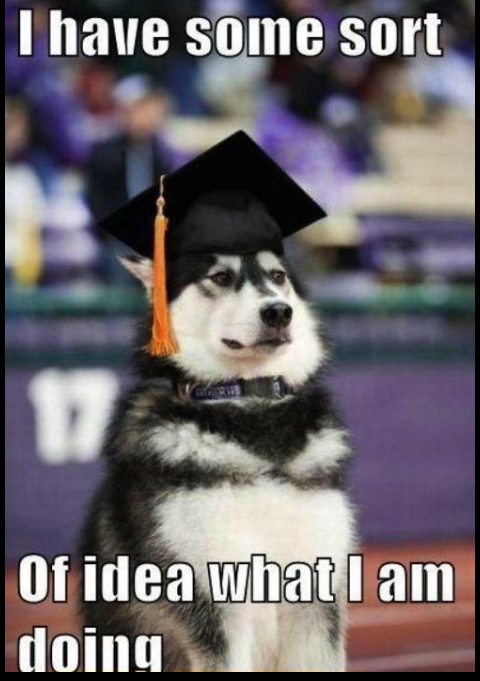
How it started

That Escalated Quickly

How it started

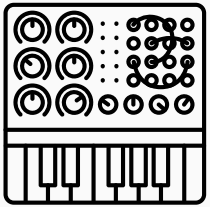
How it's going




I have some sort
Of idea what I am doing

WITH GREAT INSIGHT
COMES GREAT AUTOMATION

MDS Analysis

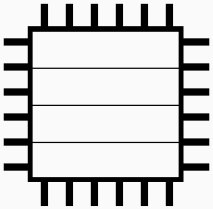


- Many Microarchitectural Data Sampling (MDS) attacks
  → ZombieLoad, RIDL, Fallout, Meltdown-UC
- Different variants and leakage targets
- Complex to reproduce and test all variations
- Common: require a fault or microcode assist

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 = faulting[0]
```

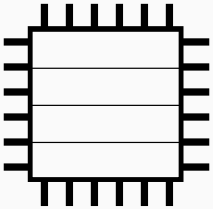


User Memory

	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 = faulting[0]
```

Fault



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

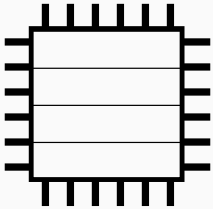
```
char value = faulting[0]
```

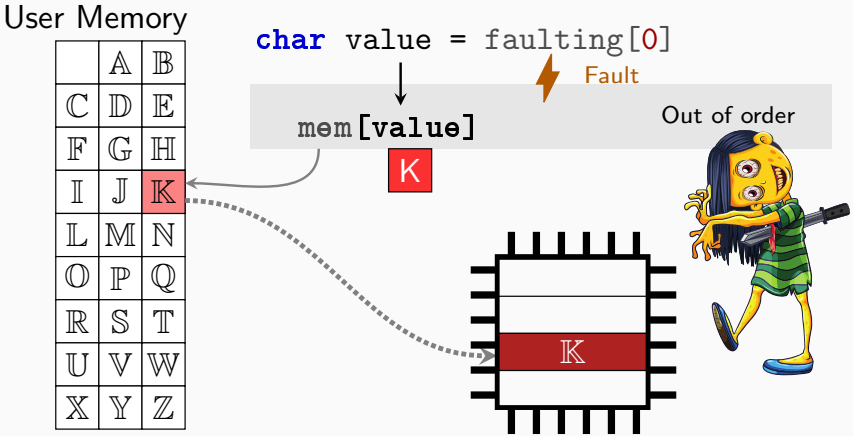
Fault

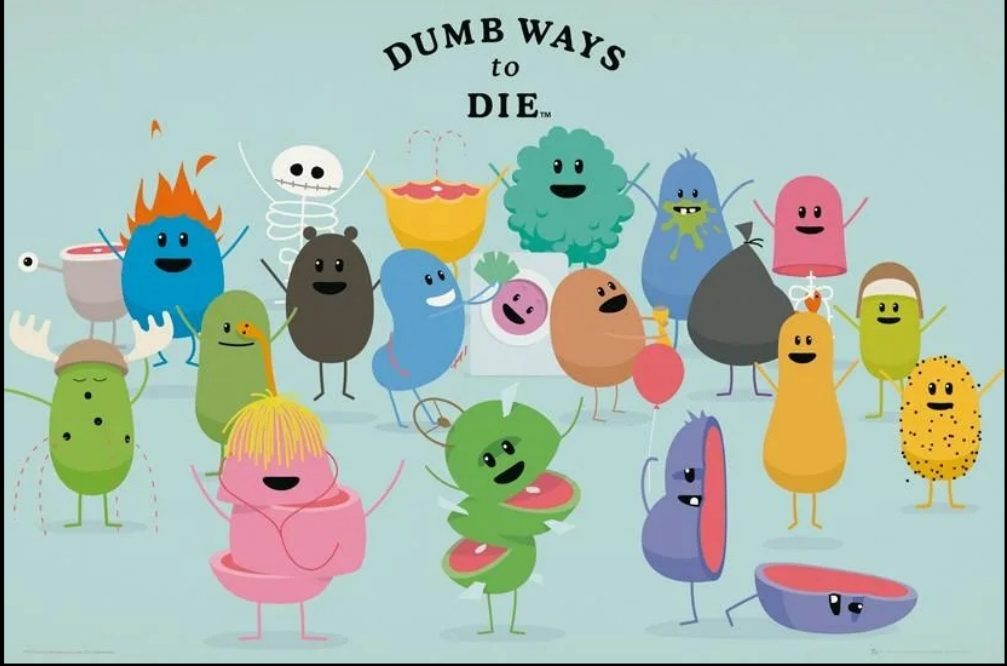
Out of order

```
mem[value]
```

K




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 = faulting[0]
Fault
mem[value]
K
Out of order
K

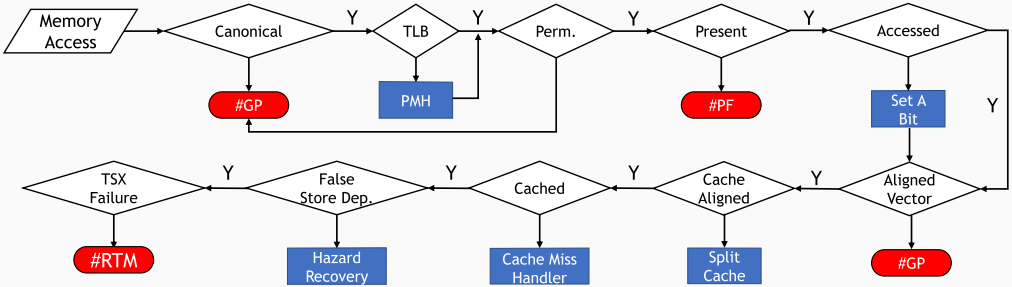

DUMB WAYS
to
DIE™

Memory Access Checks (simplified)

- Many possibilities for faults

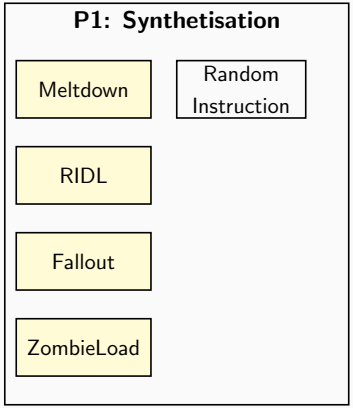
Memory Access Checks (simplified)

- Many possibilities for faults

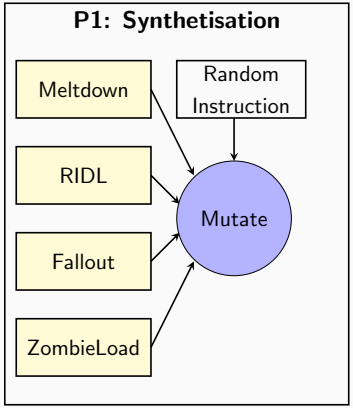


- Idea: mutation fuzzing for new variants

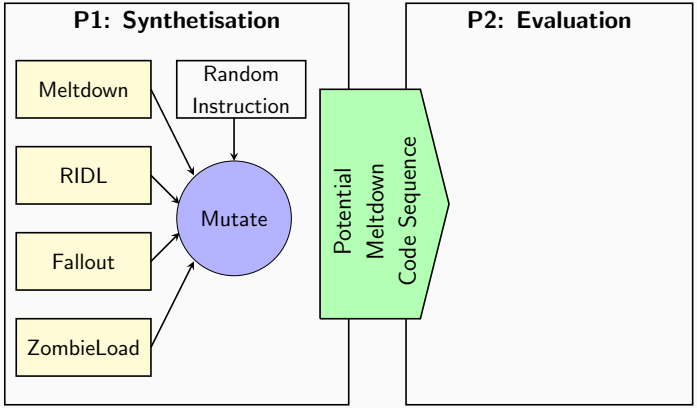
Transynther



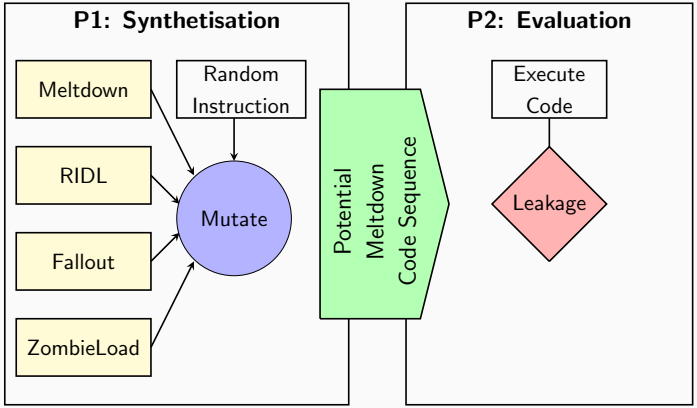
Transynther



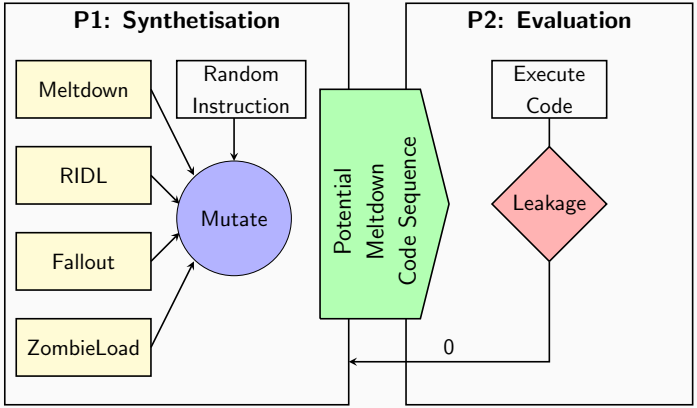
Transynther



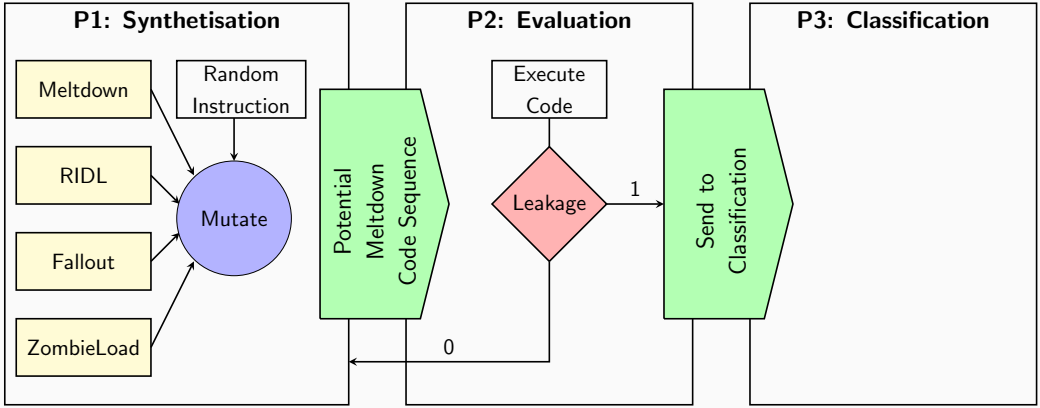
Transynther



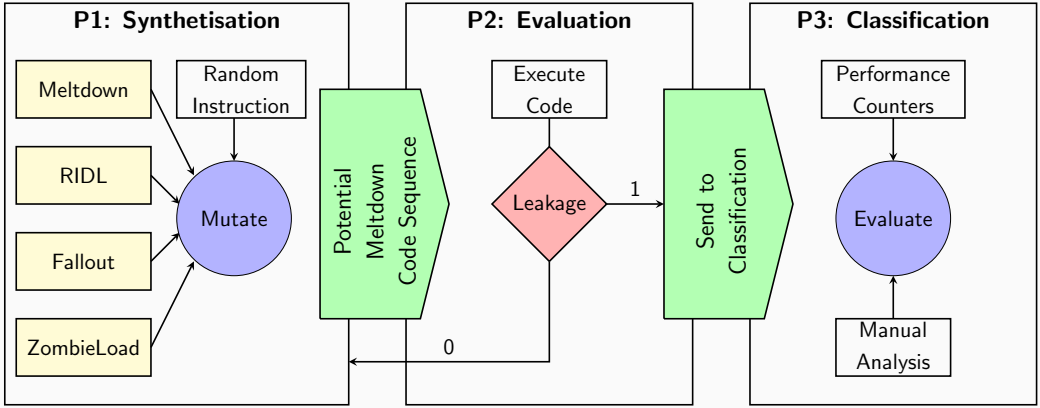
Transynther



Transynther

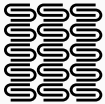


Transynther



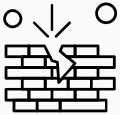
Transynther Results

26 hours runtime



100 unique leakage patterns

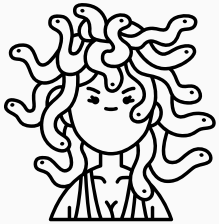
7 attacks reproduced



1 new vulnerability

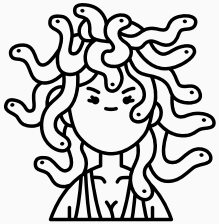
1 regression

Medusa



- Medusa: new variant of ZombieLoad

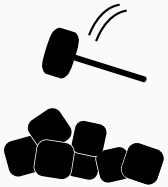
Medusa



- Medusa: new variant of ZombieLoad
- Leaks from write-combining buffer, i.e., `REP MOV`
- Used for fast memory copy, e.g., in OpenSSL or kernel

$\rightarrow$ Leaked RSA key while decoding in OpenSSL

Ice Lake Regression



- Ice Lake microarchitecture reported no vulnerabilities
- Transynther found a regression via a small mutation

→ Re-enabled a “mitigated” variant

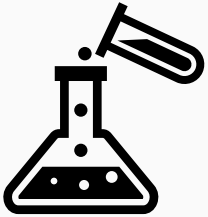
- Fixed via microcode update


ARCHITECTURE
STRANGER THINGS
MICROARCHITECTURE
CONTENTION FAULTS
INTERRUPT CONFLICT

WHY SHOULDN'T I LOOK
AT ARCHITECTURAL INTERFACES?

Playing with MSRs

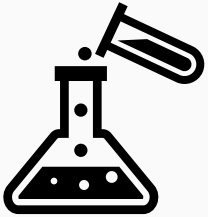
- Goal: disable a prefetcher for an experiment



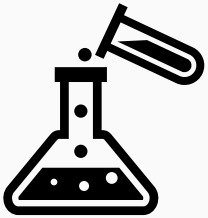
Playing with MSRs

- Goal: disable a prefetcher for an experiment

```
% sudo wrmsr -a 320 0x2
```



Playing with MSRs

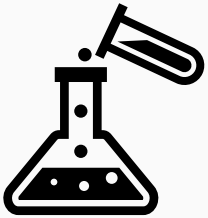


- Goal: disable a prefetcher for an experiment

```
% sudo wrmsr -a 320 0x2
```

- Typo: should be 420, not 320

Playing with MSRs



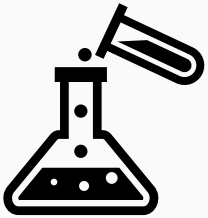
- Goal: disable a prefetcher for an experiment

```
% sudo wrmsr -a 320 0x2
```

- Typo: should be 420, not 320

```
% sudo wrmsr -a 420 0x2
Segmentation fault (core dumped)
% top
Segmentation fault (core dumped)
% sudo reboot
Segmentation fault (core dumped)
```

Playing with MSRs



- Goal: disable a prefetcher for an experiment

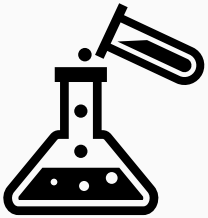
```
% sudo wrmsr -a 320 0x2
```

- Typo: should be 420, not 320

```
% sudo wrmsr -a 420 0x2
Segmentation fault (core dumped)
% top
Segmentation fault (core dumped)
% sudo reboot
Segmentation fault (core dumped)
```

→ Every application crashed on startup

Playing with MSRs



- Goal: disable a prefetcher for an experiment

```
% sudo wrmsr -a 320 0x2
```

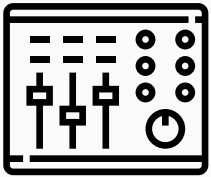
- Typo: should be 420, not 320

```
% sudo wrmsr -a 420 0x2
Segmentation fault (core dumped)
% top
Segmentation fault (core dumped)
% sudo reboot
Segmentation fault (core dumped)
```

→ Every application crashed on startup

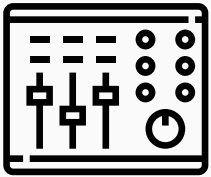
- No information about this MSR in the Intel manual

Model-Specific Registers and their Hidden Secrets



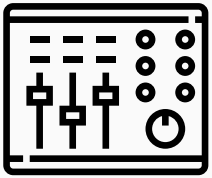
- Model-Specific Registers (MSRs): CPU control registers

Model-Specific Registers and their Hidden Secrets



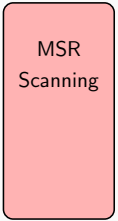
- Model-Specific Registers (MSRs): CPU control registers
- AMD K8: Undocumented debug mode via MSR

Model-Specific Registers and their Hidden Secrets

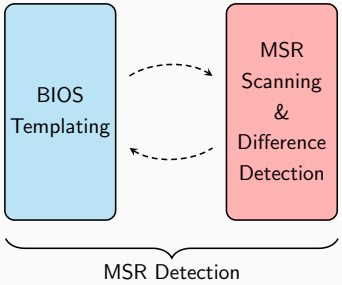


- Model-Specific Registers (MSRs): CPU control registers
- AMD K8: Undocumented debug mode via MSR
- VIA C3 CPU: backdoor access via undocumented MSR bit

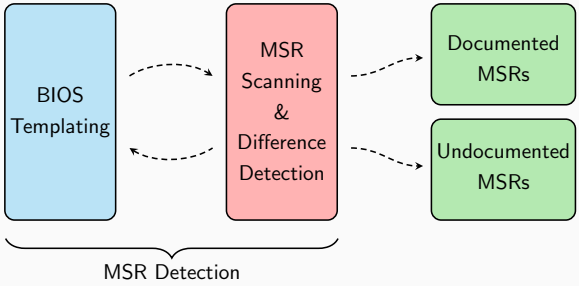
MSRevelio – Overview



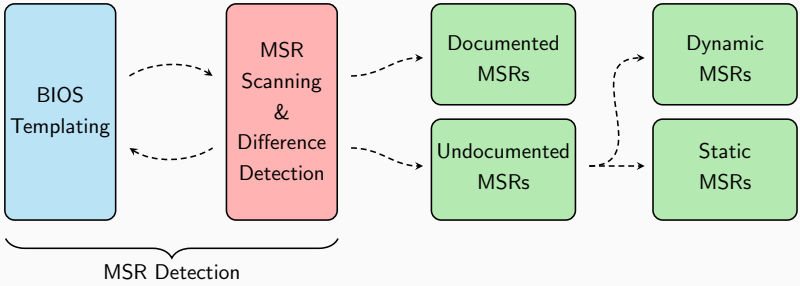
MSRevelio – Overview



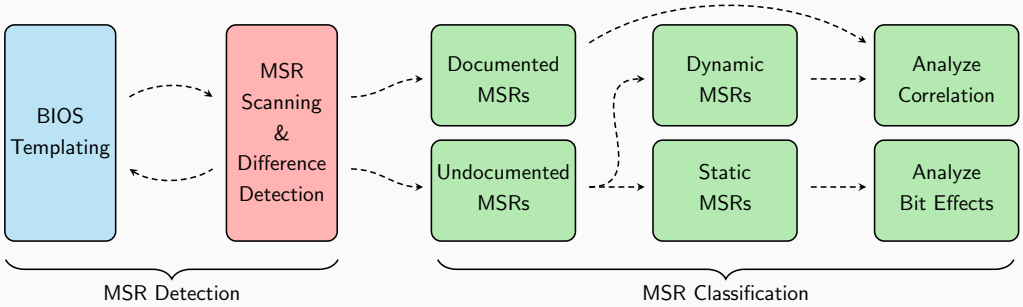
MSRevelio – Overview



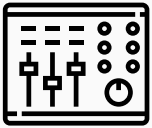
MSRevelio – Overview



MSRevelio – Overview

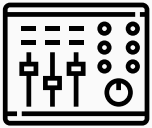


MSRevelio – Results Overview

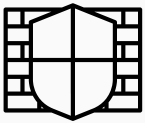


5890 undocumented MSRs

MSRevelio – Results Overview

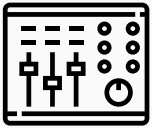


5890 undocumented MSRs

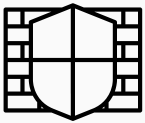


3 MSRs as attack mitigation

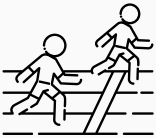
MSRevelio – Results Overview



5890 undocumented MSRs

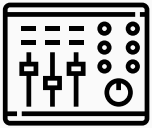


3 MSRs as attack mitigation

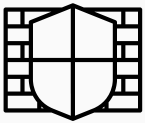


1 MSR allows TOCTOU vulnerability

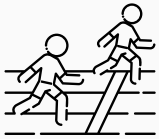
MSRevelio – Results Overview



5890 undocumented MSRs



3 MSRs as attack mitigation



1 MSR allows TOCTOU vulnerability

New MSRs hint towards vulnerabilities

Findings – Undisclosed Attacks

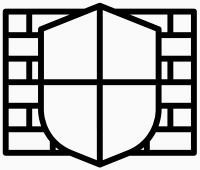
- MSRs often introduce vulnerability fixes

Findings – Undisclosed Attacks

- MSRs often introduce vulnerability fixes
- MSRs exist before public disclosure

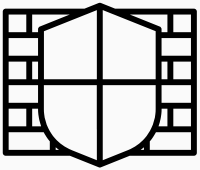
→ Useful for 1-Day Exploits

Findings – Attack Mitigation



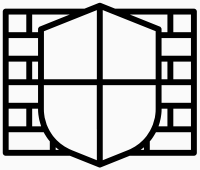
- Mitigate prefetch side-channel attacks

Findings – Attack Mitigation



- Mitigate prefetch side-channel attacks
- Reduce CrossTalk leakage

Findings – Attack Mitigation



- Mitigate prefetch side-channel attacks
- Reduce CrossTalk leakage
- Reduce Medusa leakage

MSRevelio – Lessons Learned

- Searching for unknown behavior is hard

MSRevelio – Lessons Learned

- Searching for unknown behavior is hard
- We can automate the search for undocumented MSR behavior

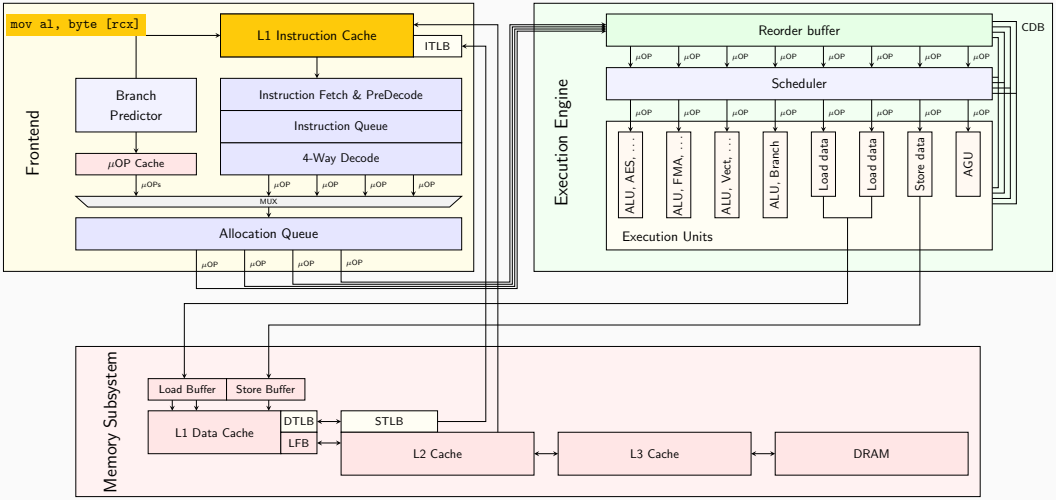
MSRevelio – Lessons Learned

- Searching for unknown behavior is hard
- We can automate the search for undocumented MSR behavior
- Automation allows tracing changes between releases

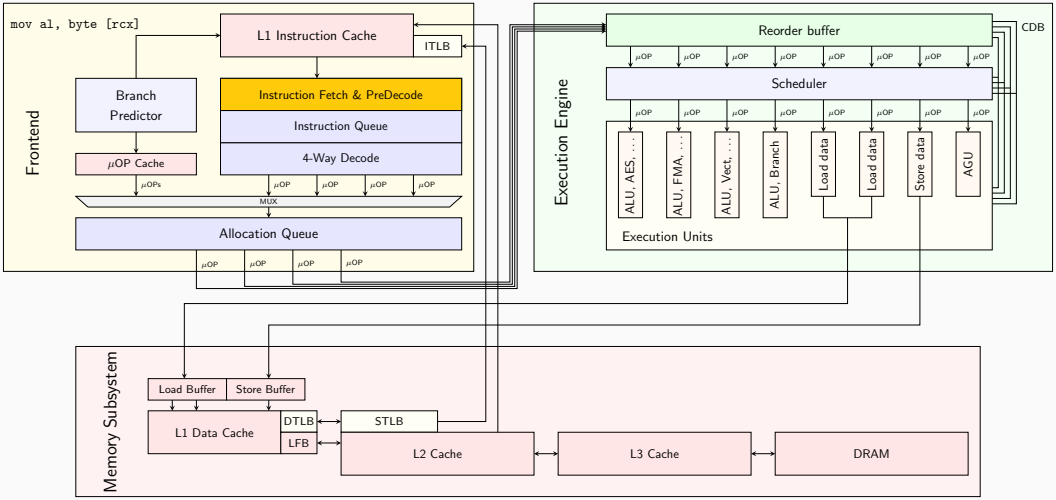
MSRevelio – Lessons Learned

- Searching for unknown behavior is hard
- We can automate the search for undocumented MSR behavior
- Automation allows tracing changes between releases
- Architectural interfaces not fully explored

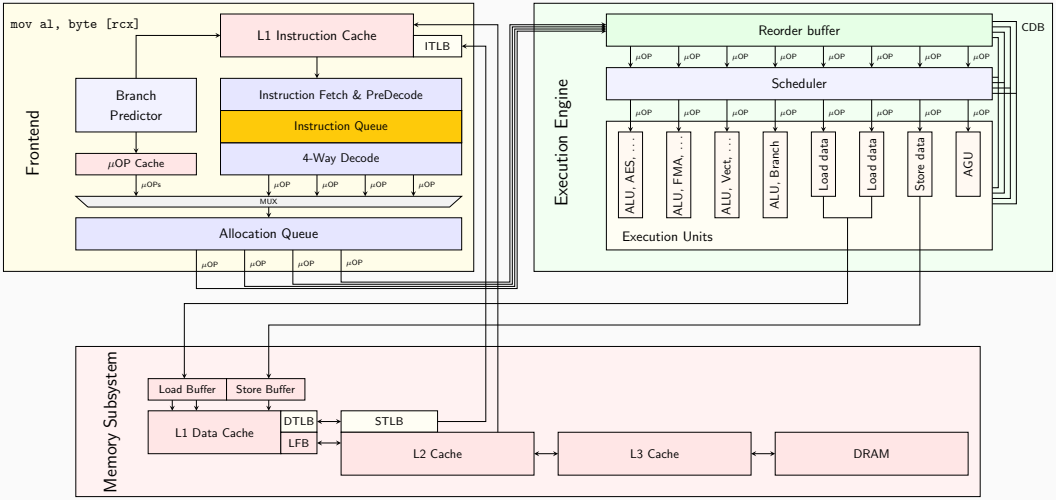
Loads are Complex



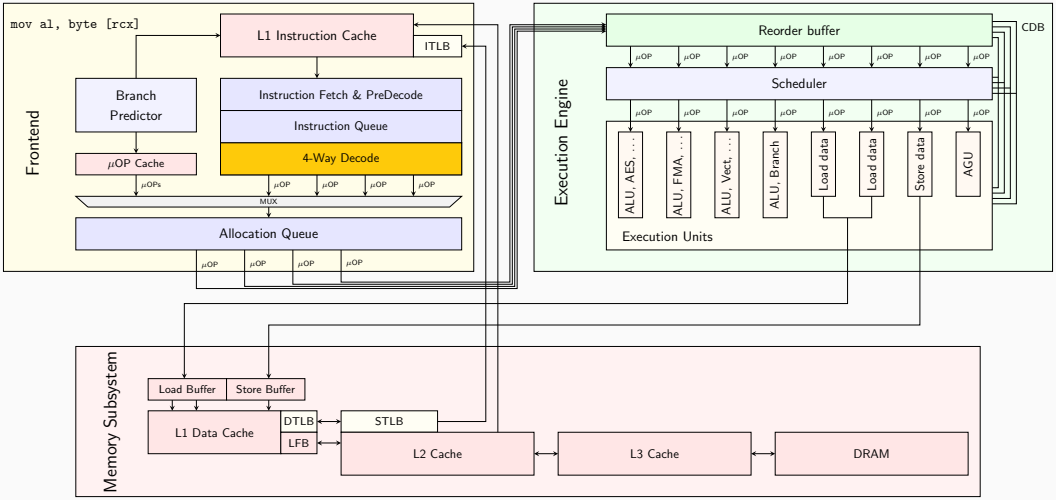
Loads are Complex



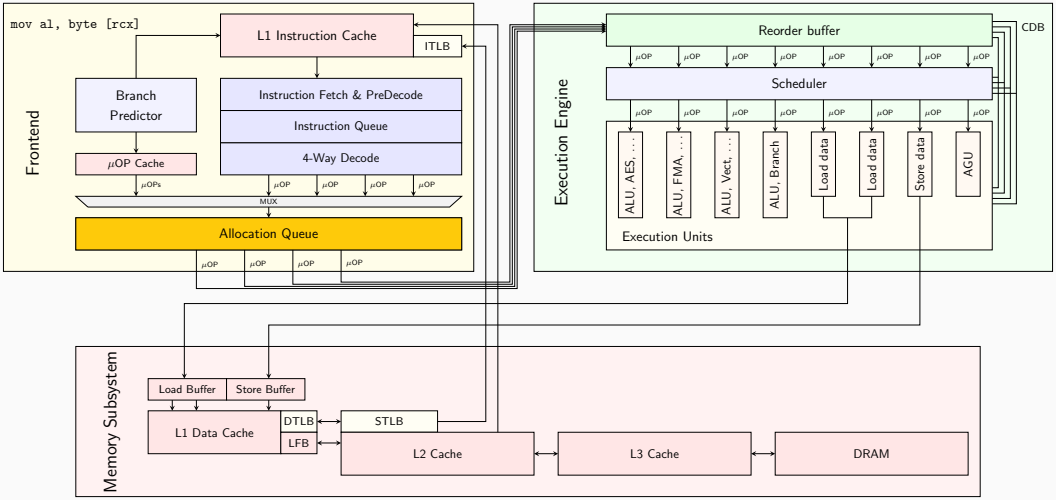
Loads are Complex



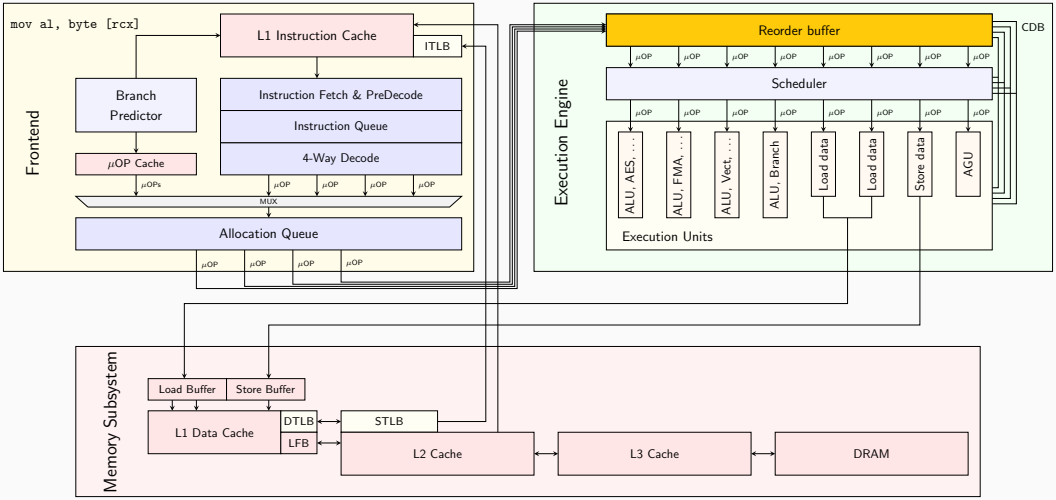
Loads are Complex



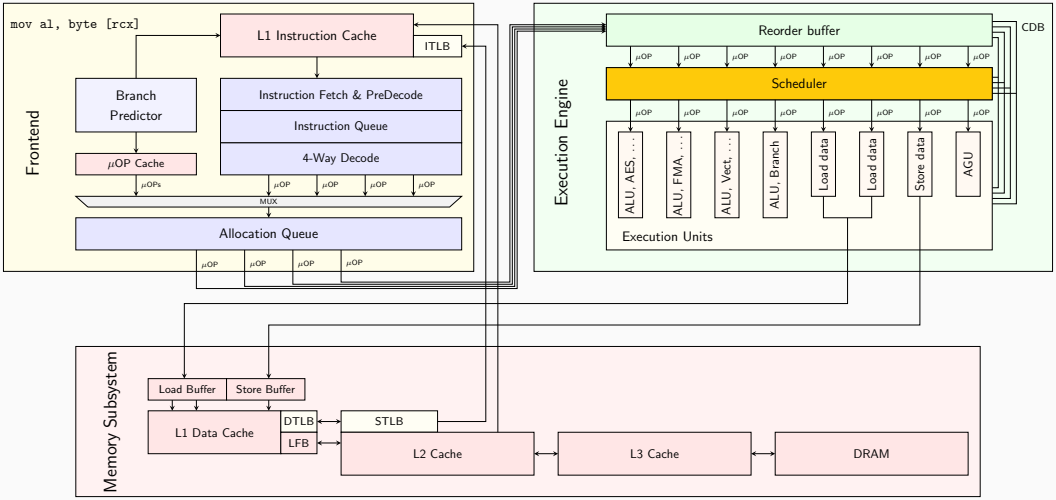
Loads are Complex



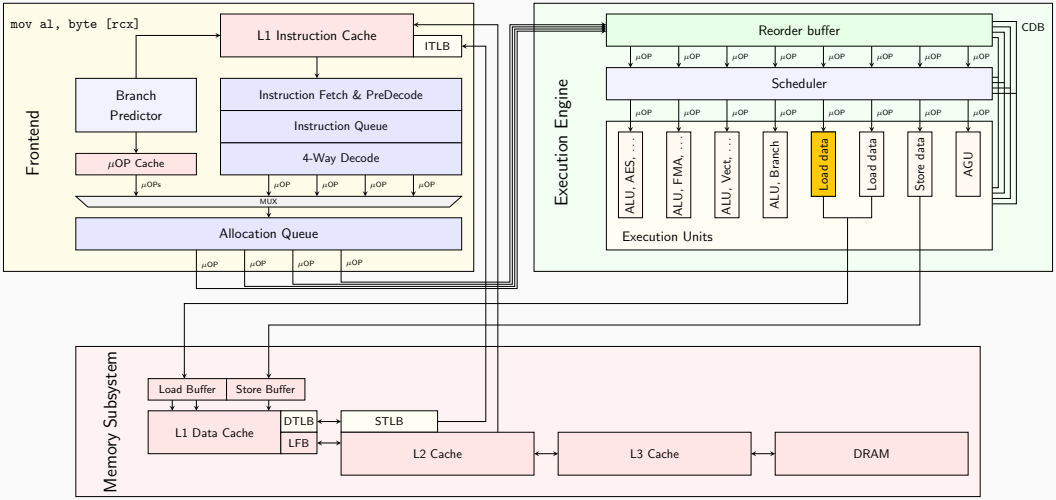
Loads are Complex



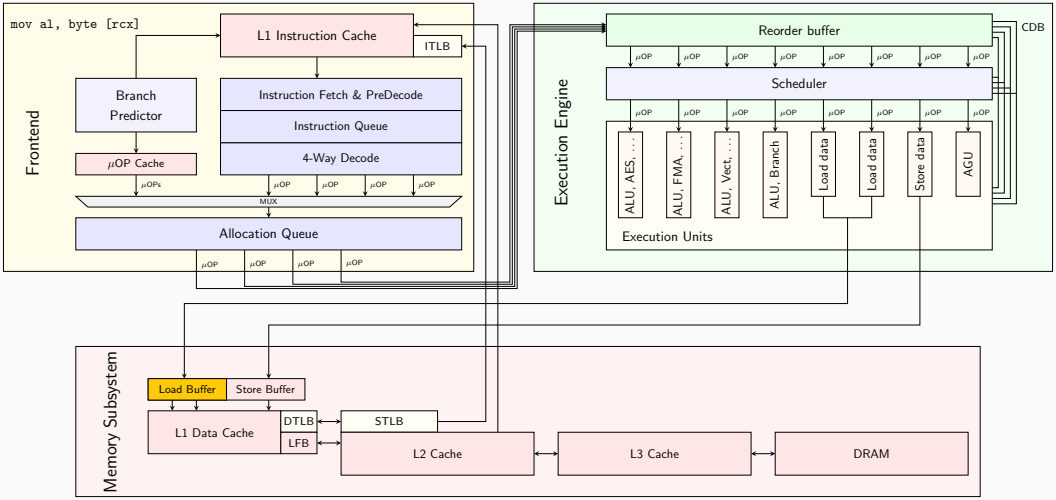
Loads are Complex



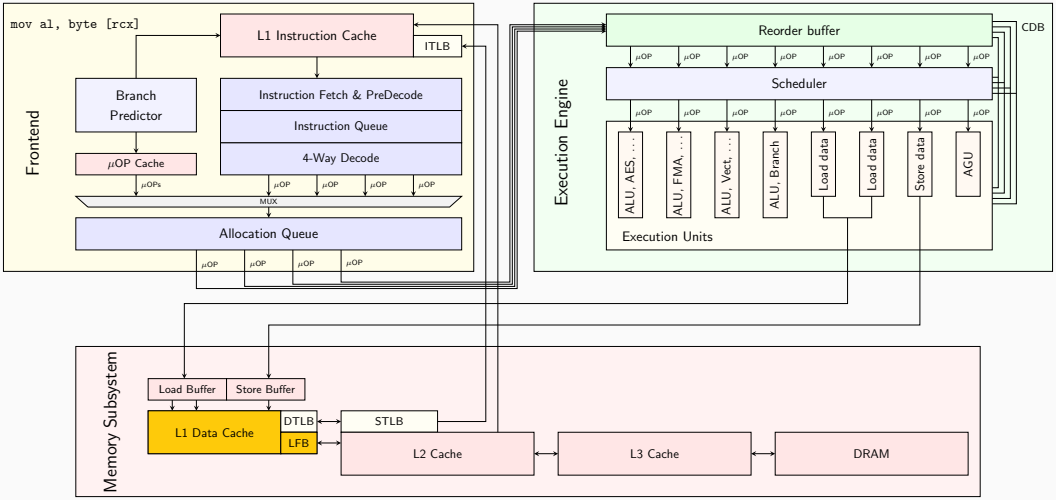
Loads are Complex



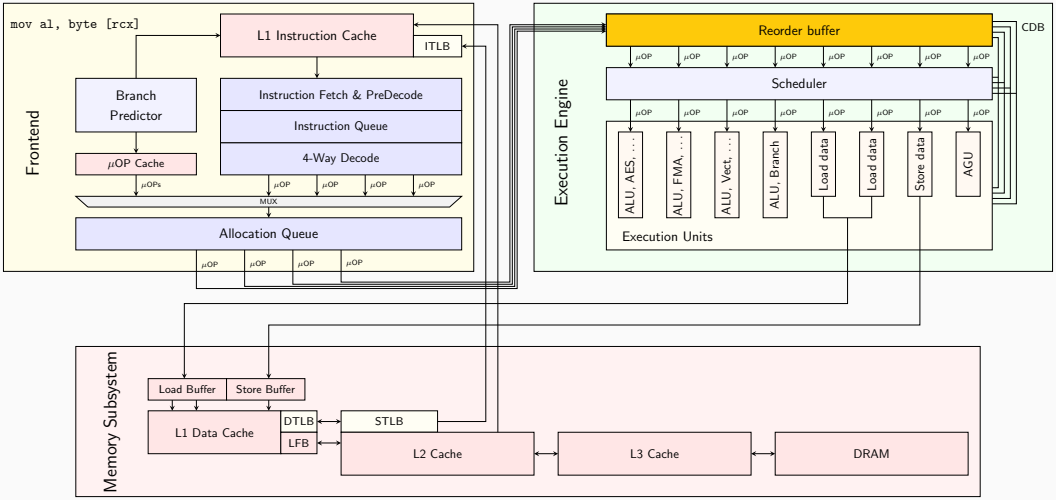
Loads are Complex



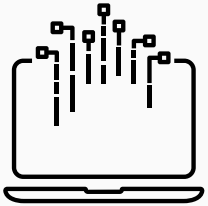
Loads are Complex



Loads are Complex

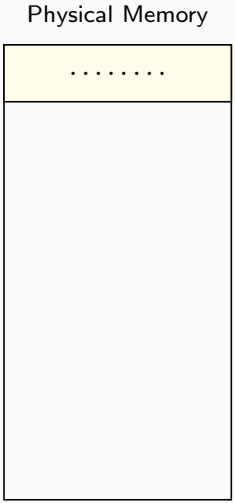


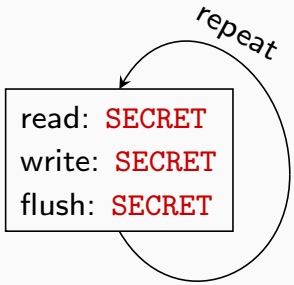
Bunch of Stuff Mapped



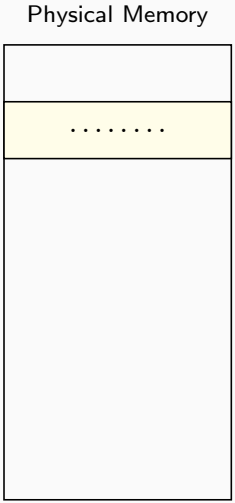
```
00001000-0009efff : System RAM
0009f000-000fffff : Reserved
  000a0000-000bffff : PCI Bus 0000:00
00100000-412f6017 : System RAM
45cba000-45cbafff : ACPI Non-volatile Storage
47f00000-64bfffff : Reserved
  61000000-64bfffff : Graphics Stolen Memory
64c00000-bfffffff : PCI Bus 0000:00
  66000000-721fffff : PCI Bus 0000:01
fee00000-fee00fff : Local APIC
  fee00000-fee00fff : Reserved
ff000000-ffffffff : Reserved
  ff000000-ffffffff : pnp 00:04
100000000-49b3fffff : System RAM
  1ad600000-1ae400df0 : Kernel code
  1ae400df1-1af25687f : Kernel data
  1af52a000-1af9fffff : Kernel bss
49b400000-49bffffff : RAM buffer
[...]
```

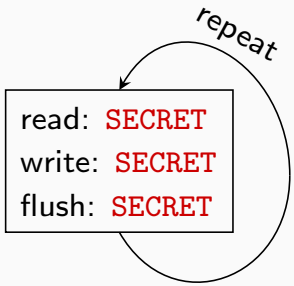
Scanning Memory



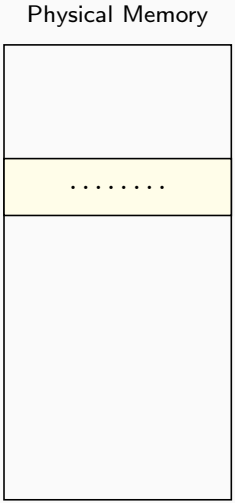


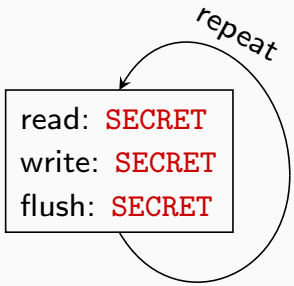
Scanning Memory



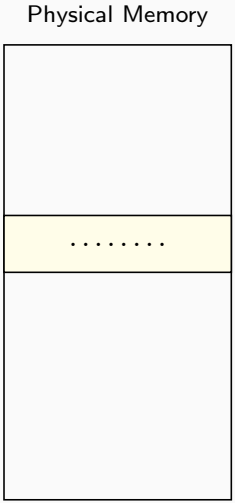


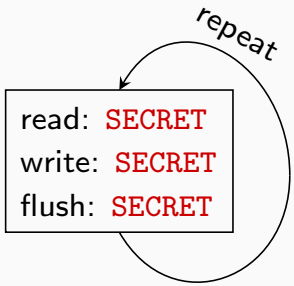
Scanning Memory



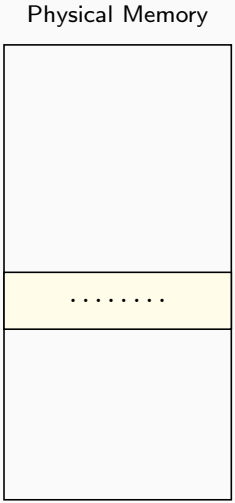


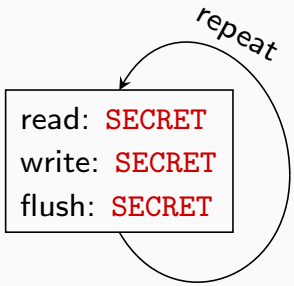
Scanning Memory



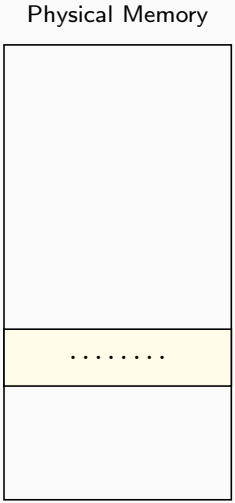


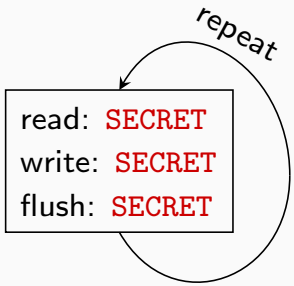
Scanning Memory



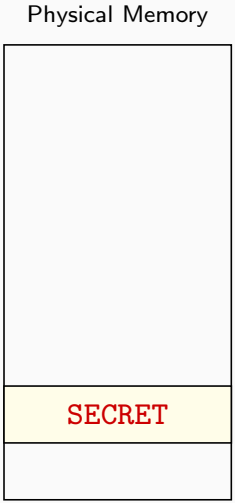


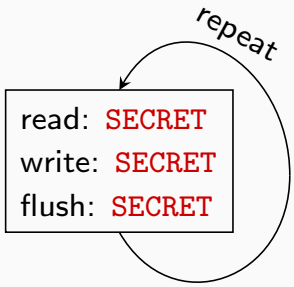
Scanning Memory



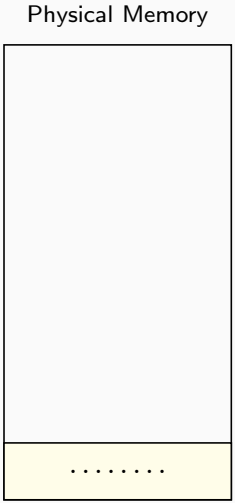


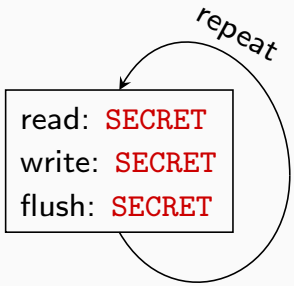
Scanning Memory

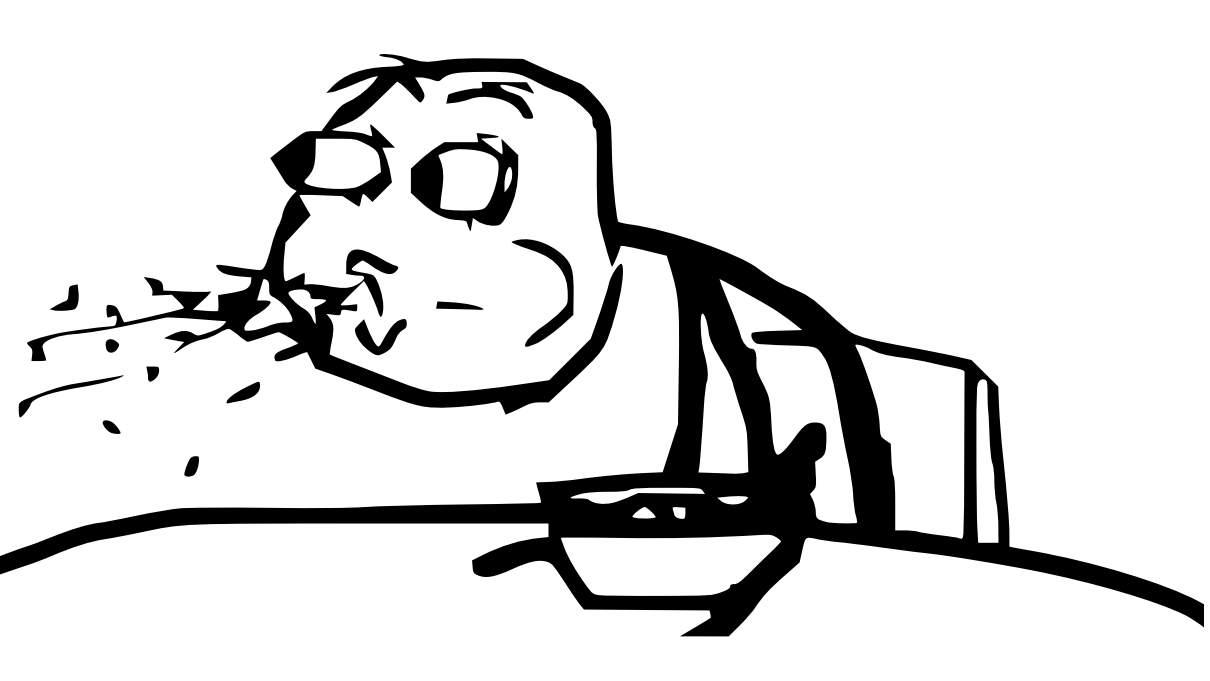




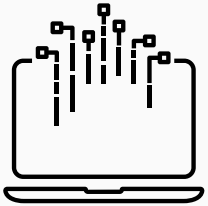
Scanning Memory





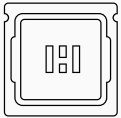


Leaking Address



```
00001000-0009efff : System RAM
0009f000-000fffff : Reserved
  000a0000-000bffff : PCI Bus 0000:00
00100000-412f6017 : System RAM
45cba000-45cbafff : ACPI Non-volatile Storage
47f00000-64bfffff : Reserved
  61000000-64bfffff : Graphics Stolen Memory
64c00000-bfffffff : PCI Bus 0000:00
  66000000-721fffff : PCI Bus 0000:01
fee00000-fee00fff :  Local APIC
  fee00000-fee00fff : Reserved
ff000000-ffffffff : Reserved
  ff000000-ffffffff : pnp 00:04
100000000-49b3fffff : System RAM
  1ad600000-1ae400df0 : Kernel code
  1ae400df1-1af25687f : Kernel data
  1af52a000-1af9fffff : Kernel bss
49b400000-49bffffff : RAM buffer
[...]
```

Advanced Programmable Interrupt Controller (APIC)

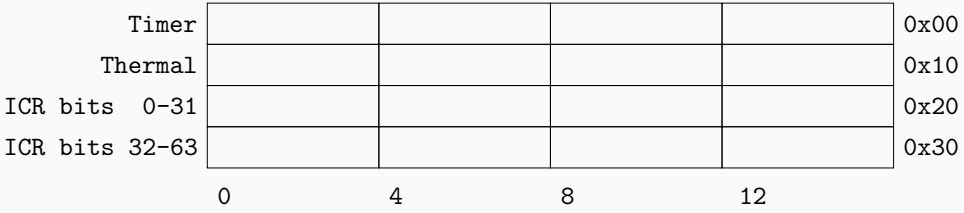


Generate, receive and forward interrupts in modern CPUs.

- Local APIC for each CPU
- I/O APIC towards external devices
- Exposes registers

APIC MMIO

- **Memory-mapped** APIC registers

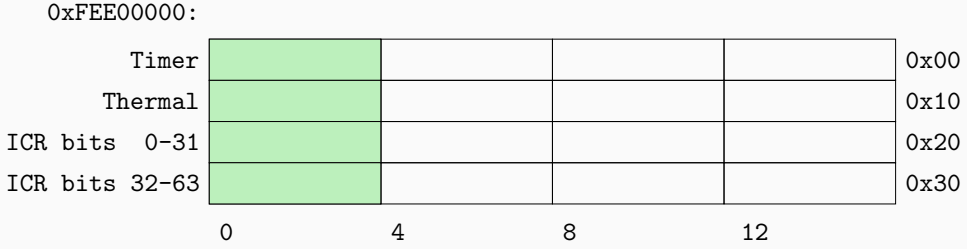


APIC MMIO

- **Memory-mapped** APIC registers
  - Controlled by MSR `IA32_APIC_BASE` (default `0xFEE00000`)

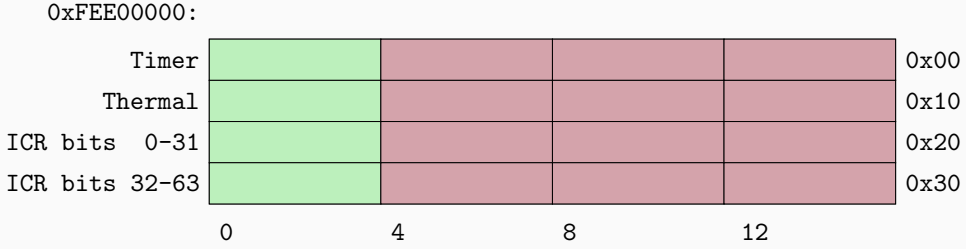
APIC MMIO

- **Memory-mapped** APIC registers
  - Controlled by MSR `IA32_APIC_BASE` (default `0xFEE00000`)
  - Mapped as **32bit** values, **aligned** to **16 bytes**



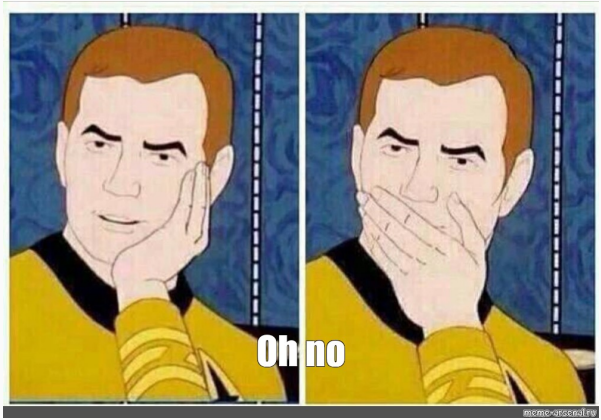
APIC MMIO

- **Memory-mapped** APIC registers
  - Controlled by MSR `IA32_APIC_BASE` (default `0xFEE00000`)
  - Mapped as **32bit** values, **aligned** to **16 bytes**
  - **Should not** be accessed at bytes 4 through 15.

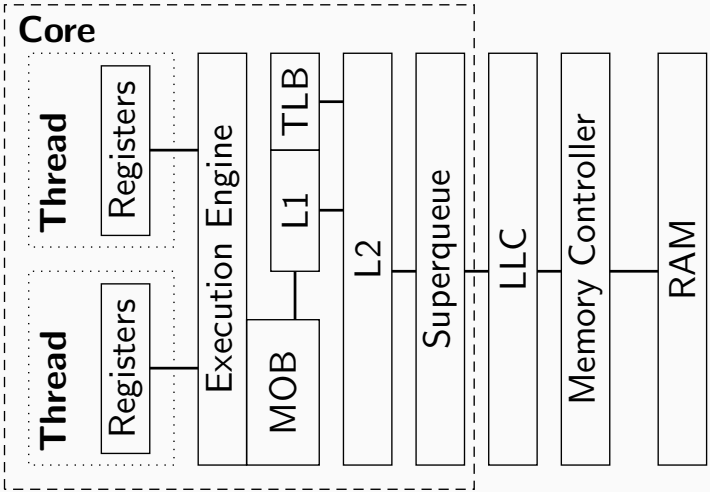


Any access that touches bytes 4 through 15 of an APIC register may cause undefined behavior and must not be executed. This undefined behavior could include hangs, incorrect results, or unexpected exceptions.

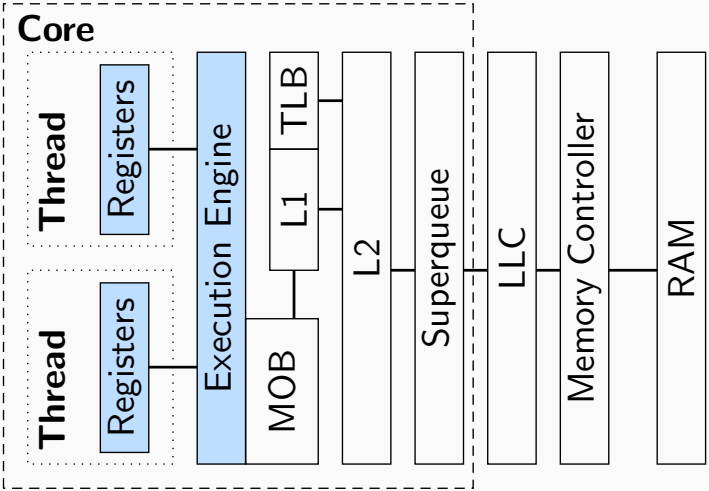
Any access that touches bytes 4 through 15 of an APIC register may cause undefined behavior and must not be executed. This undefined behavior could include hangs, incorrect results, or unexpected exceptions.



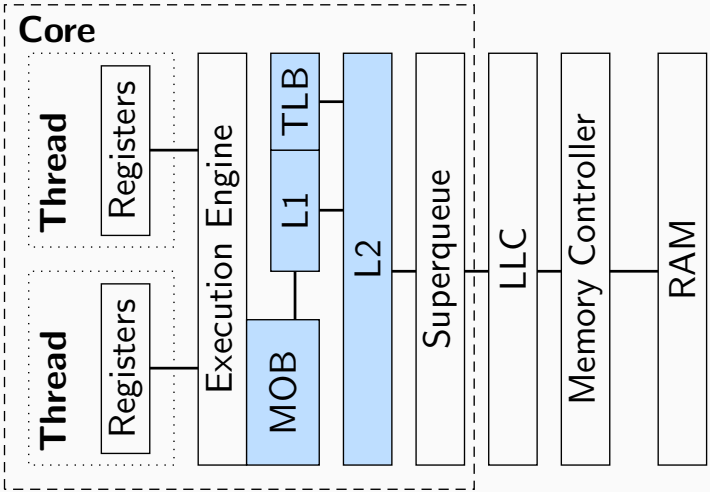
Ruling out Microarchitectural Elements



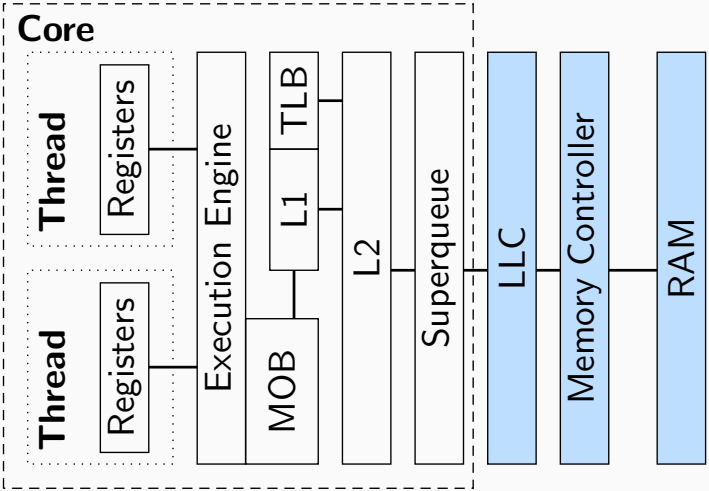
Ruling out Microarchitectural Elements



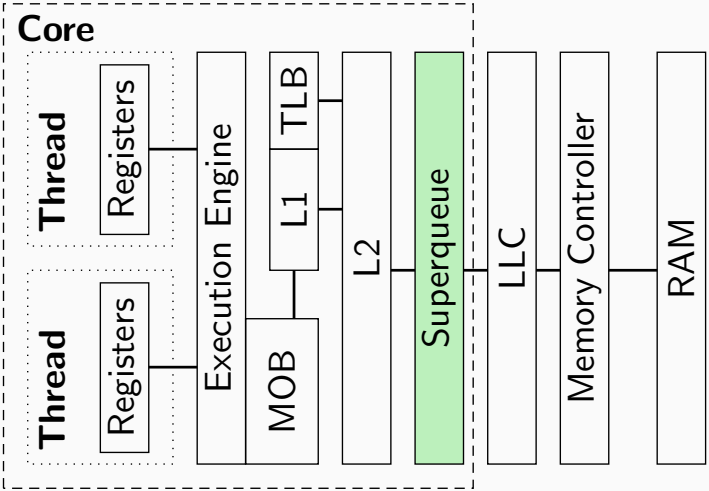
Ruling out Microarchitectural Elements



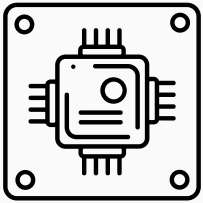
Ruling out Microarchitectural Elements



Ruling out Microarchitectural Elements

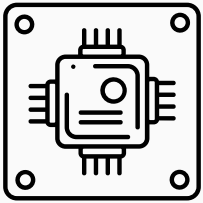


The Superqueue



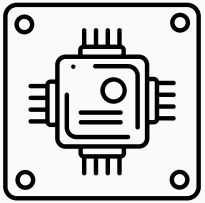
- It's the decoupling buffer between L2 and LLC

The Superqueue



- It's the decoupling buffer between L2 and LLC
- Contains data passed between L2 and LLC

The Superqueue



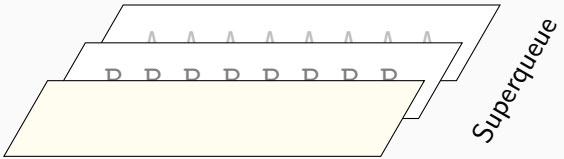
- It's the decoupling buffer between L2 and LLC
- Contains data passed between L2 and LLC
- Like Line Fill Buffers for L1 and L2

Leaking from the Superqueue

APIC

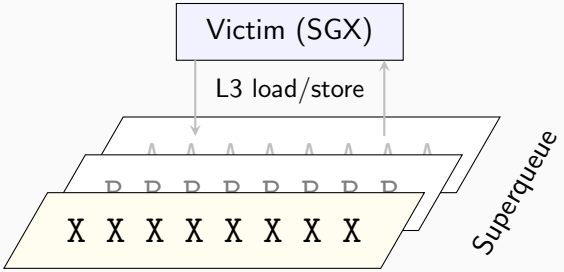
EOI	???
ISR	???
IRR	???

Victim (SGX)



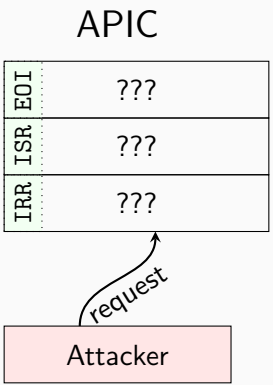
Attacker

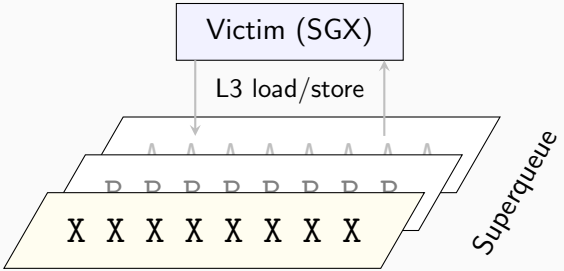
Leaking from the Superqueue



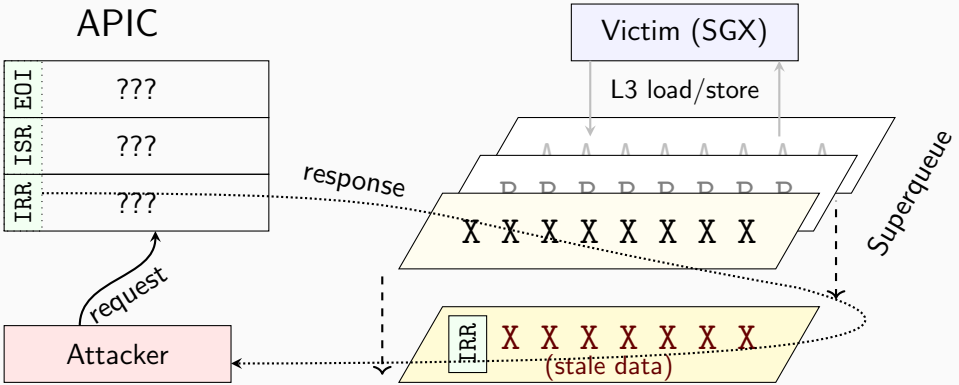
Attacker

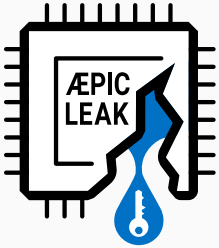
Leaking from the Superqueue





Leaking from the Superqueue





- First architectural CPU vulnerability
- Deterministically leak data from SGX
- Does not require hyperthreading

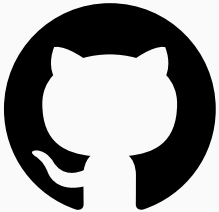
Software vs. Hardware Fuzzing

- All low-hanging fruit
- Approximately as sophisticated as software fuzzing in 1990
- Majority of fuzzers does not use any guidance
- More research on feedback necessary

Summary

- Simple models are sufficient to find leakage
- Dumb fuzzers find leakage within hours
  - New vulnerability variants
  - New side channels
  - Regression in new CPUs
  - New architectural vulnerabilities
- Prediction: smarter fuzzers → more vulnerabilities

Open Source



https://github.com/CISPA/Osiris

USENIX'21

D. Weber, A. Ibrahim, H. Nemati, M. Schwarz, C. Rossow.
Osiris: Automated Discovery of Microarchitectural Side Channels.

https://github.com/vernamlab/Medusa

USENIX'20

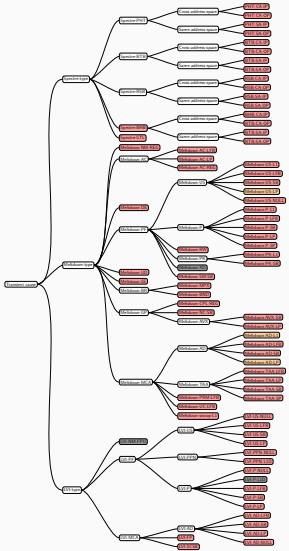
D. Moghimi, M. Lipp, B. Sunar, M. Schwarz.
Medusa: Microarchitectural Data Leakage via Automated Attack Synthesis.

https://github.com/IAIK/MSRevelio

USENIX'22

A. Kogler, D. Weber, M. Haubenwallner, M. Lipp, D. Gruss, M. Schwarz.
Finding and Exploiting CPU Features using MSR Templating.

Spectre, Meltdown, and LVI Variants



Twittern

Josh Walden @jmw1123 · 19. Nov.

Case of beer on it's way/there later this week thanks Daniel! Thanks again for the partnership!

> **Daniel Gruss** @lavados · 13. Nov.
> Antwort an @Desertrold und @jmw1123
> I'm in favor!

2 5 34

Daniel Gruss
@lavados

Antwort an @jmw1123

Thanks again Josh!

We already received the case a month ago but only found time this weekend to sit together and enjoy some!

We wish you a merry Christmas and look forward to continue working with Intel next year.

cc @cc0x1f @mlqxyz @misc0110 @tugraz_csbme #tugraz

Tweet übersetzen

Du und Claudio Canella

5:45 nachm. · 24. Dez. 2019 · Twitter Web App

23 „Gefällt mir"-Angaben


FUZZ
ALL THE THINGS

From Random Observations to Automated Leakage Discovery

Michael Schwarz

December 2022

CISPA Helmholtz Center for Information Security