Transient Execution Attacks

Daniel Gruss

June 20, 2019

Graz University of Technology

amazon.com
Prime+Probe

ANOTHER FLIP IN THE ROW

ROWHAMMER.JS

FANTASTIC TIMERS

AND WHERE TO FIND THEM

HIGH-RESOLUTION MICROARCHITECTURAL ATTACKS IN JAVASCRIPT

JavaScript zero

REAL JavaScript AND ZERO SIDE-CHANNEL ATTACKS

Marketing...

Marketing...

- Why do you have a website?

- Why do you have a website?→ Inform journalists and the general public

- Why do you have a website?→ Inform journalists and the general public
  - Otherwise: completely misleading presentation of your work in the media

- Why do you have a website?→ Inform journalists and the general public
  - Otherwise: completely misleading presentation of your work in the media

  → defend yourself against misleading presentations!

Marketing...

- Why do you have a website?→ Inform journalists and the general public
  - Otherwise: completely misleading presentation of your work in the media

  → defend yourself against misleading presentations!
- Why do you have fancy names?

Marketing...

- Why do you have a website?→ Inform journalists and the general public
  - Otherwise: completely misleading presentation of your work in the media

  → defend yourself against misleading presentations!
- Why do you have fancy names? → what was CVE-2017-5754 again?

Marketing...

- Why do you have a website?→ Inform journalists and the general public
  - Otherwise: completely misleading presentation of your work in the media

  → defend yourself against misleading presentations!
- Why do you have fancy names? → what was CVE-2017-5754 again?
  - People will throw things together that don't belong together

- Why do you have a website?→ Inform journalists and the general public
  - Otherwise: completely misleading presentation of your work in the media

  → defend yourself against misleading presentations!
- Why do you have fancy names? → what was CVE-2017-5754 again?
  - People will throw things together that don't belong together

  → Names enable unambiguous communication

Marketing...

- Why do you have a website?→ Inform journalists and the general public
  - Otherwise: completely misleading presentation of your work in the media

  → defend yourself against misleading presentations!
- Why do you have fancy names? → what was CVE-2017-5754 again?
  - People will throw things together that don't belong together

  → Names enable unambiguous communication
- Why do you need a logo?

Marketing...

- Why do you have a website?→ Inform journalists and the general public
  - Otherwise: completely misleading presentation of your work in the media

  → defend yourself against misleading presentations!
- Why do you have fancy names? → what was CVE-2017-5754 again?
  - People will throw things together that don't belong together

  → Names enable unambiguous communication
- Why do you need a logo?
  - Otherwise: media makes their own

Marketing...

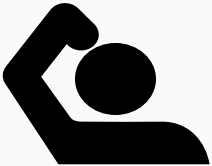
- Why do you have a website?→ Inform journalists and the general public
  - Otherwise: completely misleading presentation of your work in the media

  → defend yourself against misleading presentations!
- Why do you have fancy names? → what was CVE-2017-5754 again?
  - People will throw things together that don't belong together

  → Names enable unambiguous communication
- Why do you need a logo?
  - Otherwise: media makes their own → no control over how inappropriate these are


side channel
= obtaining meta-data and
deriving secrets from it
CHANGE MY MIND

Side Channel or not?

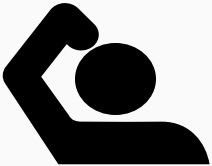


Side Channel or not?

- Profiling cache utilization with performance counters?

Side Channel or not?

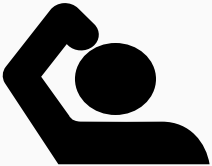
- Profiling cache utilization with performance counters? $\rightarrow$ No



- Profiling cache utilization with performance counters? $\rightarrow$ No
- Observing cache utilization with performance counters and using it to infer a crypto key?

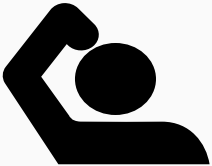
Side Channel or not?

- Profiling cache utilization with performance counters? $\rightarrow$ No
- Observing cache utilization with performance counters and using it to infer a crypto key? $\rightarrow$ Yes

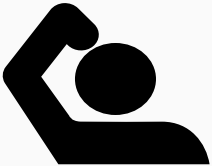


- Profiling cache utilization with performance counters? $\rightarrow$ No
- Observing cache utilization with performance counters and using it to infer a crypto key? $\rightarrow$ Yes
- Measuring memory access latency with Flush+Reload?

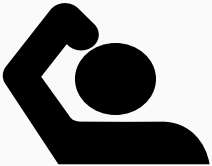
Side Channel or not?



- Profiling cache utilization with performance counters? → No
- Observing cache utilization with performance counters and using it to infer a crypto key? → Yes
- Measuring memory access latency with Flush+Reload? → No



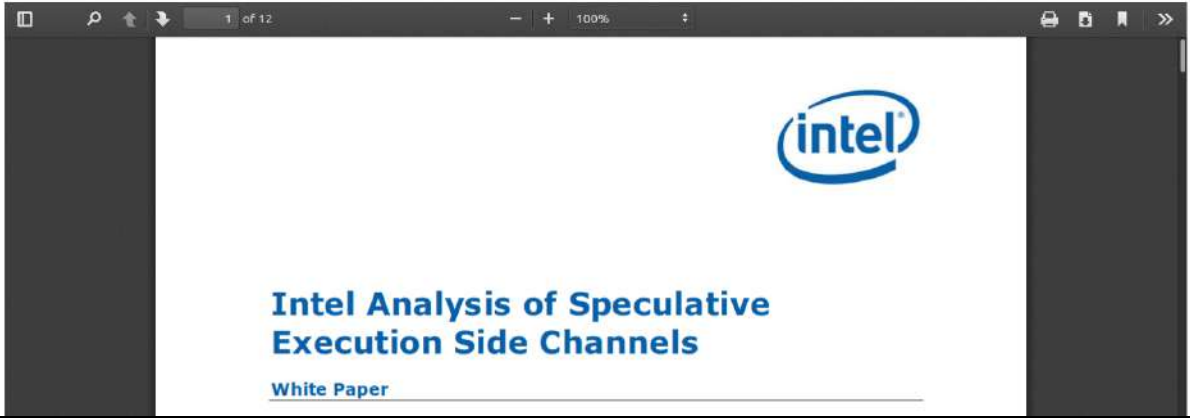
- Profiling cache utilization with performance counters? $\rightarrow$ No
- Observing cache utilization with performance counters and using it to infer a crypto key? $\rightarrow$ Yes
- Measuring memory access latency with Flush+Reload? $\rightarrow$ No
- Measuring memory access latency with Flush+Reload and using it to infer keystroke timings?



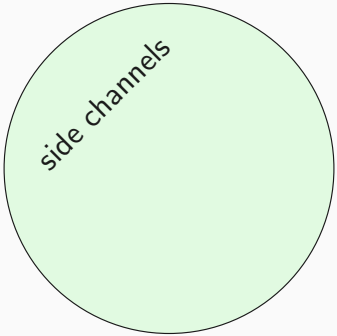
- Profiling cache utilization with performance counters? $\rightarrow$ No
- Observing cache utilization with performance counters and using it to infer a crypto key? $\rightarrow$ Yes
- Measuring memory access latency with Flush+Reload? $\rightarrow$ No
- Measuring memory access latency with Flush+Reload and using it to infer keystroke timings? $\rightarrow$ Yes

Intel Analysis of Speculative Execution Side Channels

Download PDF

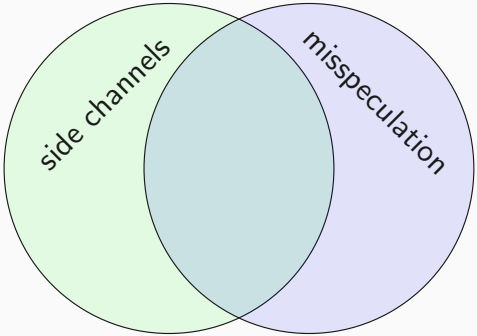


Speculative Side-Channel Attacks?



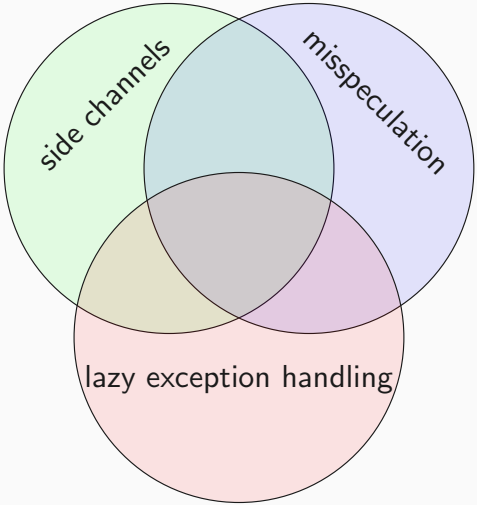
- traditional cache attacks (crypto, keys, etc)

Speculative Side-Channel Attacks?



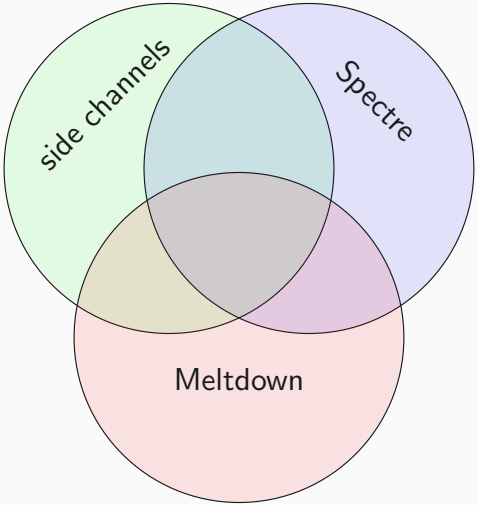
- traditional cache attacks (crypto, keys, etc)
- actual misspeculation (e.g., branch misprediction)

Speculative Side-Channel Attacks?



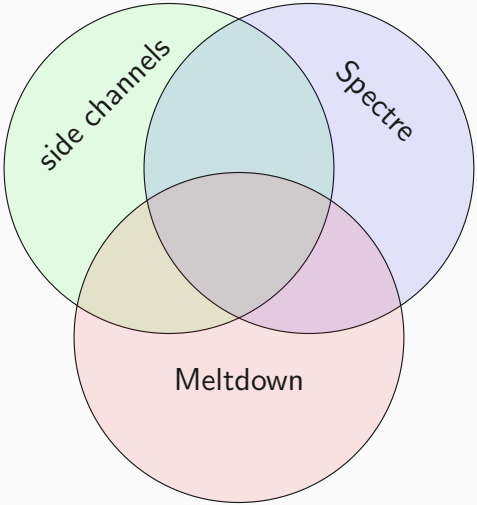
- traditional cache attacks (crypto, keys, etc)
- actual misspeculation (e.g., branch misprediction)
- Meltdown, Foreshadow, ZombieLoad, etc

Speculative Side-Channel Attacks?



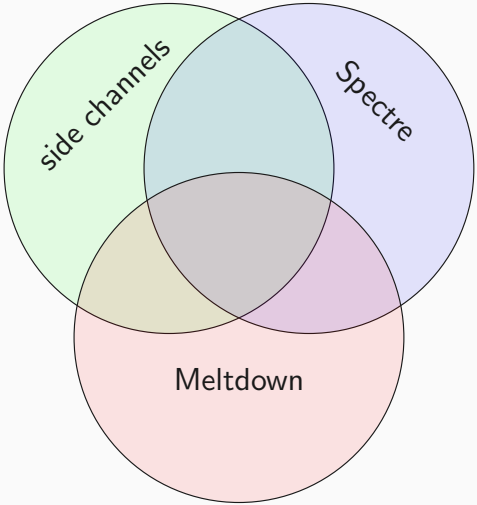
- traditional cache attacks (crypto, keys, etc)
- actual misspeculation (e.g., branch misprediction)
- Meltdown, Foreshadow, ZombieLoad, etc

Speculative Side-Channel Attacks?



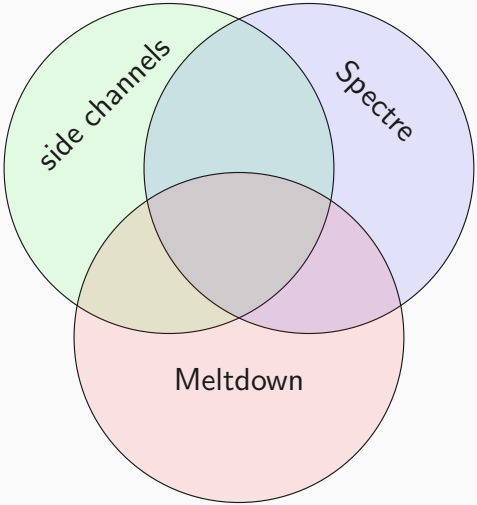
- traditional cache attacks (crypto, keys, etc)
- actual misspeculation (e.g., branch misprediction)
- Meltdown, Foreshadow, ZombieLoad, etc
- **Let's avoid the term Speculative Side-Channel Attacks**

Speculative Side-Channel Attacks?



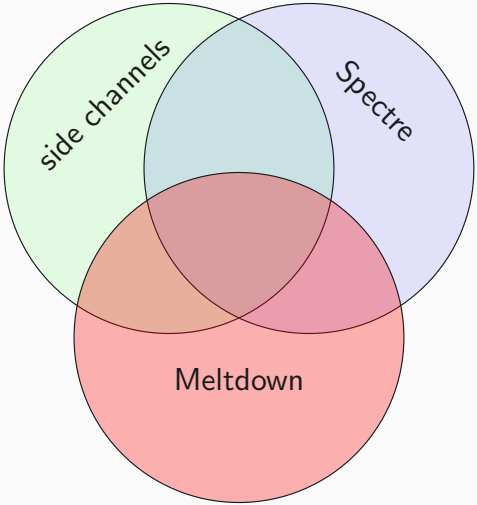
- traditional cache attacks (crypto, keys, etc)
- actual misspeculation (e.g., branch misprediction)
- Meltdown, Foreshadow, ZombieLoad, etc
- **Let's avoid the term Speculative Side-Channel Attacks**
- Let's be more precise

Speculative Side-Channel Attacks?

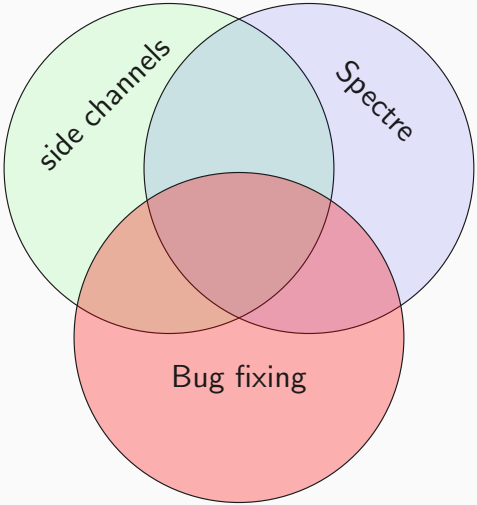


- traditional cache attacks (crypto, keys, etc)
- actual misspeculation (e.g., branch misprediction)
- Meltdown, Foreshadow, ZombieLoad, etc
- **Let's avoid the term Speculative Side-Channel Attacks**
- Let's be more precise
- $\rightarrow$ then we can think about actual mitigations

Speculative Side-Channel Attacks?



Speculative Side-Channel Attacks?



Back to Work

7. Serve with cooked and peeled potatoes

Wait for an hour

Wait for an hour

LATENCY

1. Wash and cut vegetables

2. Pick the basil leaves and set aside

3. Heat 2 tablespoons of oil in a pan

4. Fry vegetables until golden and softened

1. Wash and cut vegetables

2. Pick the basil leaves and set aside

3. Heat 2 tablespoons of oil in a pan

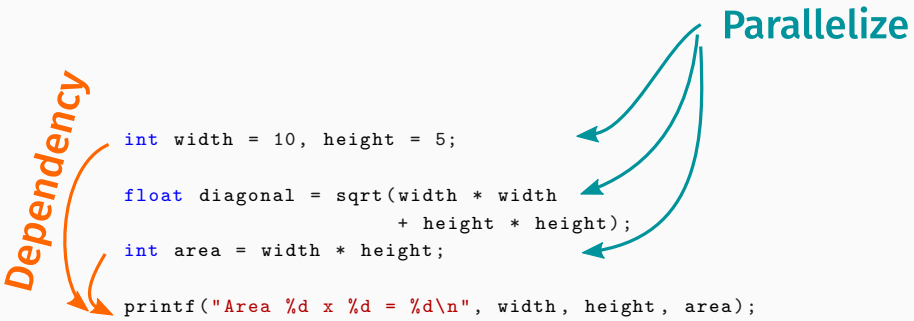
4. Fry vegetables until golden and softened

Dependency

Parallelize

Out-of-order Execution

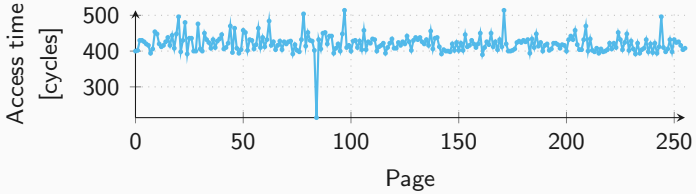
```
int width = 10, height = 5;

float diagonal = sqrt(width * width
                      + height * height);
int area = width * height;

printf("Area %d x %d = %d\n", width, height, area);
```


Parallelize
Dependency
int width = 10, height = 5;
float diagonal = sqrt(width * width
+ height * height);
int area = width * height;
printf("Area %d x %d = %d\n", width, height, area);

```
*(volatile char*) 0;
array[84 * 4096] = 0;
```

- Flush+Reload over all pages of the array

- Flush+Reload over all pages of the array



- "Unreachable" code line was actually executed

- Flush+Reload over all pages of the array

- "Unreachable" code line was actually executed
- Exception was only thrown afterwards

- Out-of-order instructions leave microarchitectural traces

- Out-of-order instructions leave microarchitectural traces
  - We can see them for example through the cache

- Out-of-order instructions leave microarchitectural traces
  - We can see them for example through the cache
- Give such instructions a name: transient instructions

- Out-of-order instructions leave microarchitectural traces
  - We can see them for example through the cache
- Give such instructions a name: transient instructions
- We can indirectly observe the execution of transient instructions

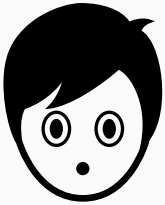
- Add another layer of indirection to test

```
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```

- Add another layer of indirection to test

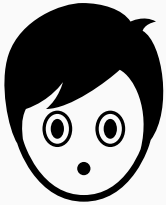
```
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```

- Then check whether any part of `array` is cached



- Flush+Reload over all pages of the array

- Index of cache hit reveals data



- Flush+Reload over all pages of the array

- Index of cache hit reveals data
- Permission check is in some cases not fast enough

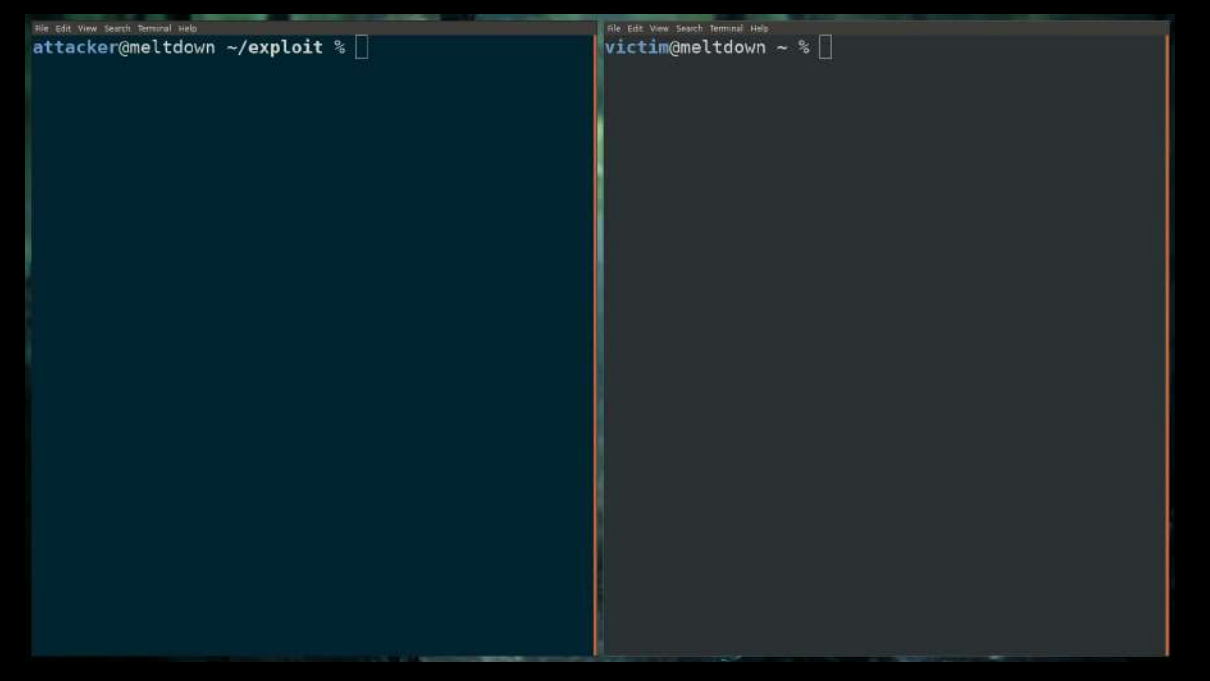

I SHIT YOU NOT
THERE WAS KERNEL MEMORY ALL OVER THE TERMINAL

```
e01d8130: 20 75 73 65 64 20 77 69  74 68 20 61 75 74 68 6f | used with autho|
e01d8140: 72 69 7a 61 74 69 6f 6e  20 66 72 6f 6d 0a 20 53 |rization from. S|
e01d8150: 69 6c 69 63 6f 6e 20 47  72 61 70 68 69 63 73 2c |ilicon Graphics,|
e01d8160: 20 49 6e 63 2e 20 20 48  6f 77 65 76 65 72 2c 20 | Inc.  However, |
e01d8170: 74 68 65 20 61 75 74 68  6f 72 73 20 6d 61 6b 65 |the authors make|
e01d8180: 20 6e 6f 20 63 6c 61 69  6d 20 74 68 61 74 20 4d | no claim that M|
e01d8190: 65 73 61 0a 20 69 73 20  69 6e 20 61 6e 79 20 77 |esa. is in any w|
e01d81a0: 61 79 20 61 20 63 6f 6d  70 61 74 69 62 6c 65 20 |ay a compatible |
e01d81b0: 72 65 70 6c 61 63 65 6d  65 6e 74 20 66 6f 72 20 |replacement for |
e01d81c0: 4f 70 65 6e 47 4c 20 6f  72 20 61 73 73 6f 63 69 |OpenGL or associ|
e01d81d0: 61 74 65 64 20 77 69 74  68 0a 20 53 69 6c 69 63 |ated with. Silic|
e01d81e0: 6f 6e 20 47 72 61 70 68  69 63 73 2c 20 49 6e 63 |on Graphics, Inc|
e01d81f0: 2e 0a 20 2e 0a 20 54 68  69 73 20 76 65 72 73 69 |.. .. This versi|
e01d8200: 6f 6e 20 6f 66 20 4d 65  73 61 20 70 72 6f 76 69 |on of Mesa provi|
e01d8210: 64 65 73 20 47 4c 58 20  61 6e 64 20 44 52 49 20 |des GLX and DRI |
e01d8220: 63 61 70 61 62 69 6c 69  74 69 65 73 3a 20 69 74 |capabilities: it|
e01d8230: 20 69 73 20 63 61 70 61  62 6c 65 20 6f 66 20 0a | is capable of. |
e01d8240: 62 6f 74 68 20 64 69 72  65 63 74 20 61 6e 64 20 |both direct and |
e01d8250: 69 6e 64 69 72 65 63 74  20 72 65 6e 64 65 72 69 |indirect renderi|
e01d8260: 6e 67 2e 20 20 46 6f 72  20 64 69 72 65 63 74 20 |ng.  For direct |
e01d8270: 72 65 6e 64 65 72 69 6e  67 2c 20 69 74 20 63 61 |rendering, it ca|
e01d8280: 6e 20 75 73 65 20 44 52  49 0a 20 6d 6f 64 75 6c |n use DRI. modul|
e01d8290: 65 73 20 66 72 6f 6d 20  74 68 65 20 6c 69 62 67 |es from the libg|
```

Terminal

File Edit View Search Terminal Help

```
mschwarz@lab06:~/Documents$
```


File Edit View Search Terminal Help
attacker@meltdown ~/exploit %
File Edit View Search Terminal Help
victim@meltdown ~ %

- Basic Meltdown code leads to a crash (segfault)

- Basic Meltdown code leads to a crash (segfault)
- How to prevent the crash?

- Basic Meltdown code leads to a crash (segfault)
- How to prevent the crash?

Fault Handling

Fault Suppression

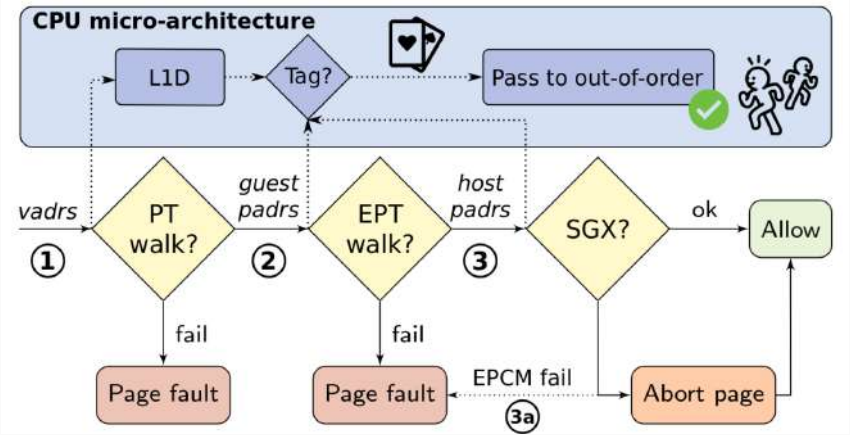
Fault Prevention

- Intel TSX to suppress exceptions instead of signal handler

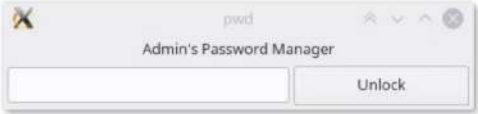
```
if(xbegin() == XBEGIN_STARTED) {
  char secret = *(char*) 0xffffffff81a000e0;
  array[secret * 4096] = 0;
  xend();
}

for (size_t i = 0; i < 256; i++) {
  if (flush_and_reload(array + i * 4096) == CACHE_HIT) {
    printf("%c\n", i);
  }
}
```

- Speculative execution to prevent exceptions

```
int speculate = rand() % 2;
size_t address = (0xffffffff81a000e0 * speculate) +
                 ((size_t)&zero * (1 - speculate));
if(!speculate) {
  char secret = *(char*) address;
  array[secret * 4096] = 0;
}

for (size_t i = 0; i < 256; i++) {
  if (flush_and_reload(array + i * 4096) == CACHE_HIT) {
    printf("%c\n", i);
  }
}
```



[1] Jo Van Bulck et al. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In: USENIX Security Symposium. 2018.



QEMU

```
Booting from ROM...
early console in extract_kernel
input_data: 0x0000000001e0a276
input_len: 0x00000000003d48f8
output: 0x0000000001000000
output_len: 0x00000000011bc258
kernel_total_size: 0x0000000000dee000
booted via startup_32()
Physical KASLR using RDTSC...
Virtual KASLR using RDTSC...

Decompressing Linux... Parsing ELF... Performing relocations... done.
Booting the kernel.
```

```
Run reader <pfn> [<cache miss threshold>] to leak hypervisor data from the L1

/ #
```

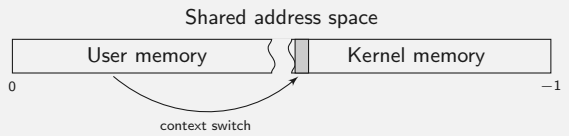
Kernel **A**ddress **I**solation to have **S**ide channels **E**fficiently **R**emoved

KAISER /ˈkʌɪzə/

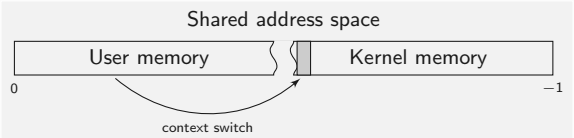
1. [german] Emperor, ruler of an empire
2. largest penguin, emperor penguin

Kernel **A**ddress **I**solation to have **S**ide channels **E**fficiently **R**emoved

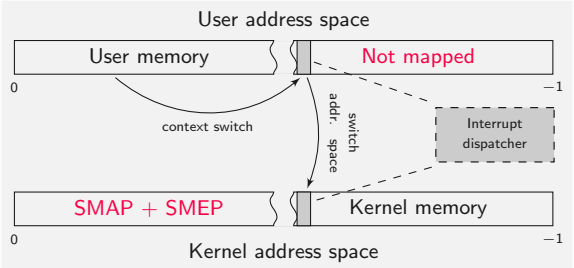
Without KAISER:

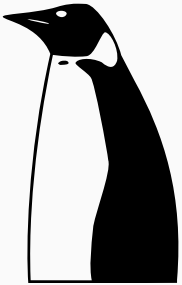


Without KAISER:

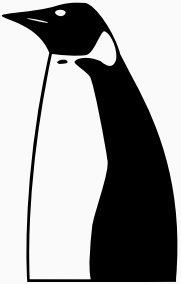


With KAISER:

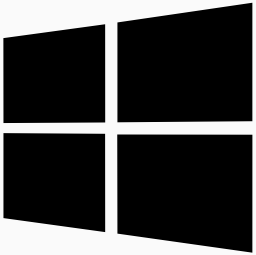




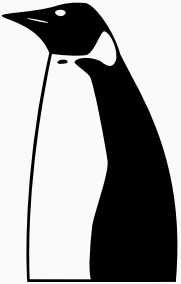
- Our patch
- Adopted in Linux



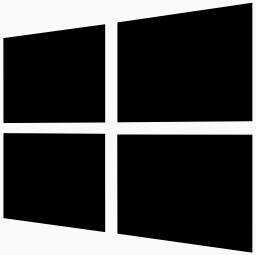
- Our patch
- Adopted in Linux



- Adopted in Windows

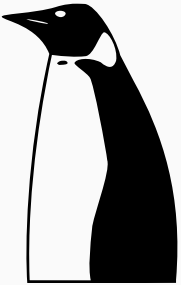


- Our patch
- Adopted in Linux

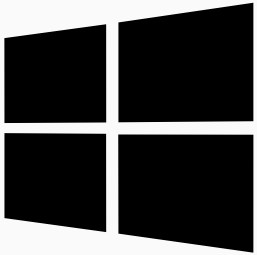


- Adopted in Windows

- Adopted in OSX/iOS



- Our patch
- Adopted in Linux

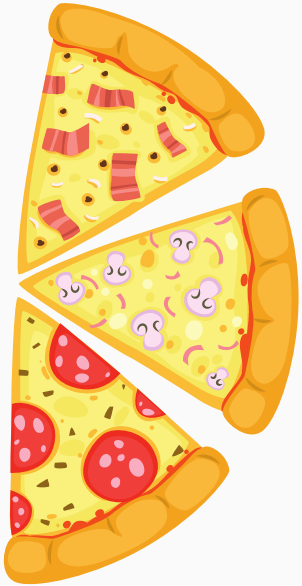


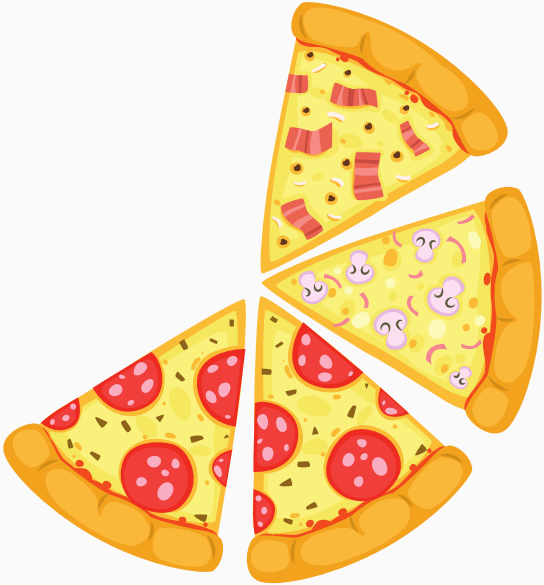
- Adopted in Windows

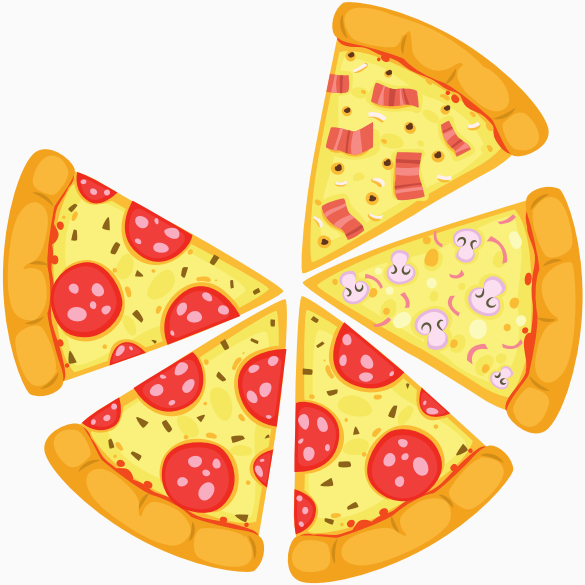
- Adopted in OSX/iOS

$\rightarrow$ **now in every computer**


PIZZA
SPECIAL RECIPES








»A table for 6 please«

Speculative Cooking


»A table for 6 please«


PIZZA
SPECIAL RECIPES


PIZZA
SPECIAL RECIPES

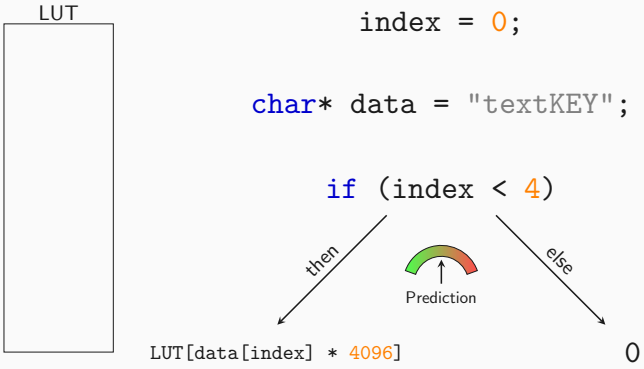

Pizza

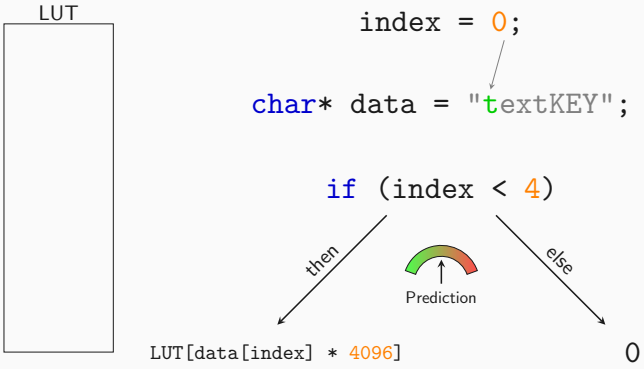

PIZZA
SPECIAL RECIPES

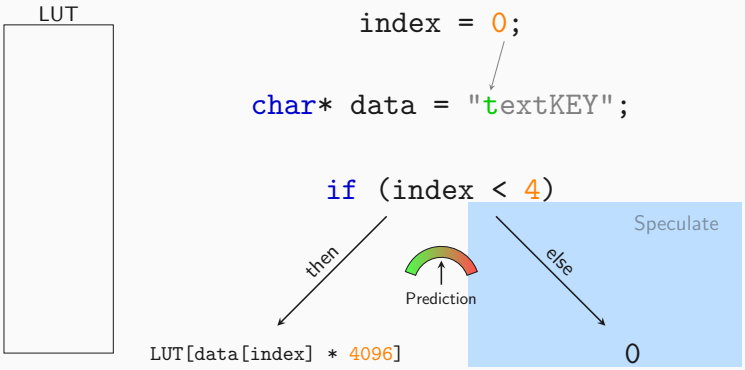

Pizza

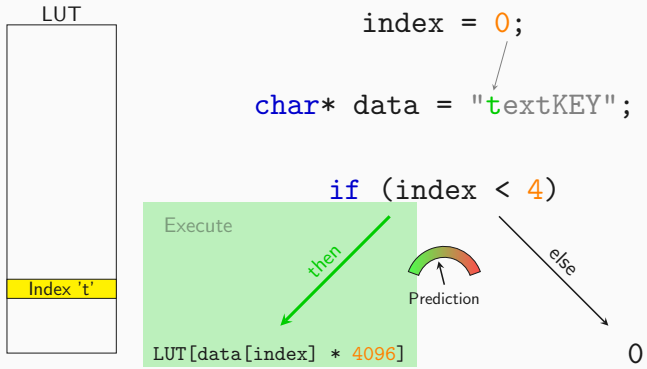

PIZZA
SPECIAL RECIPES

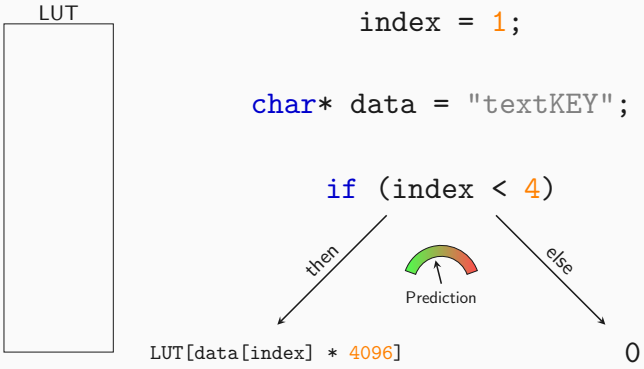

Pizza

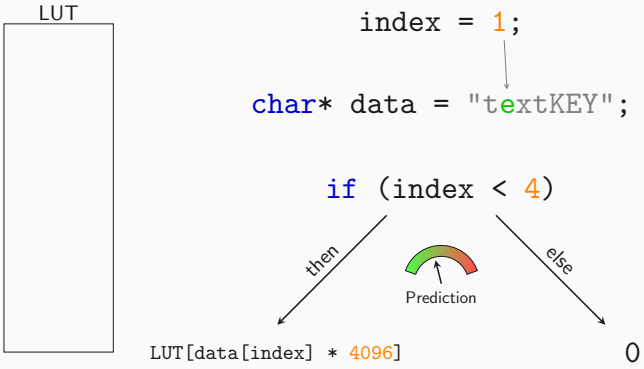

LUT
index = 0;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

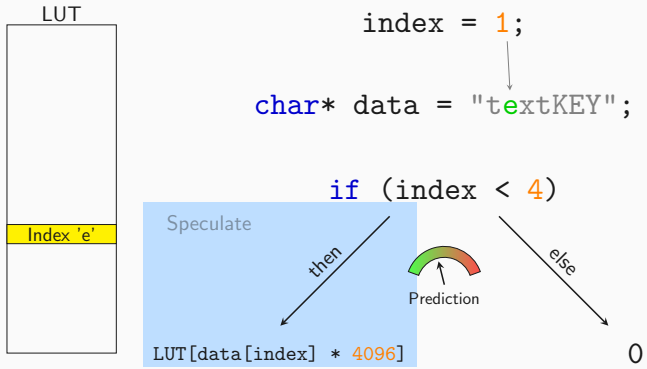

LUT
index = 0;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

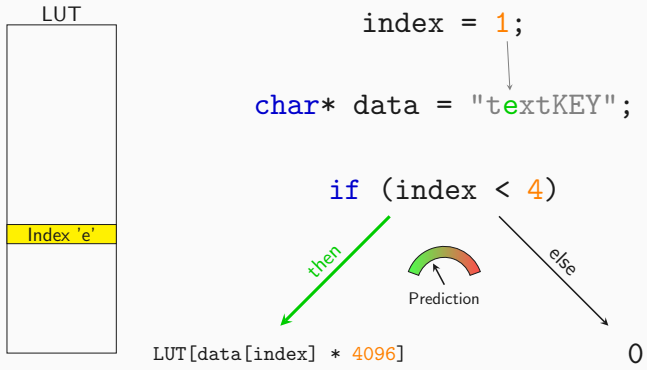

LUT
index = 0;
char* data = "textKEY";
if (index < 4)
Speculate
then
else
Prediction
LUT[data[index] * 4096]
0

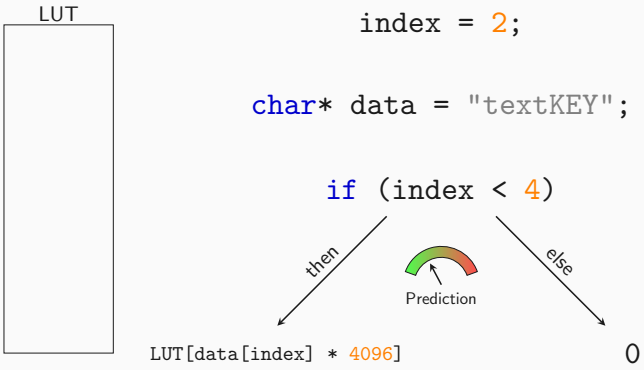

LUT
Index 't'
index = 0;
char* data = "textKEY";
if (index < 4)
Execute
then
Prediction
else
LUT[data[index] * 4096]
0

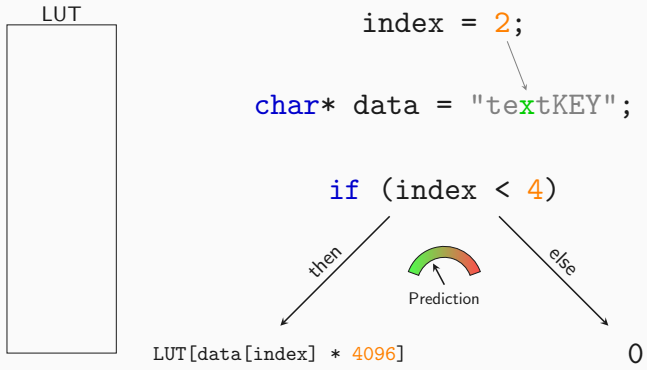

LUT
index = 1;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

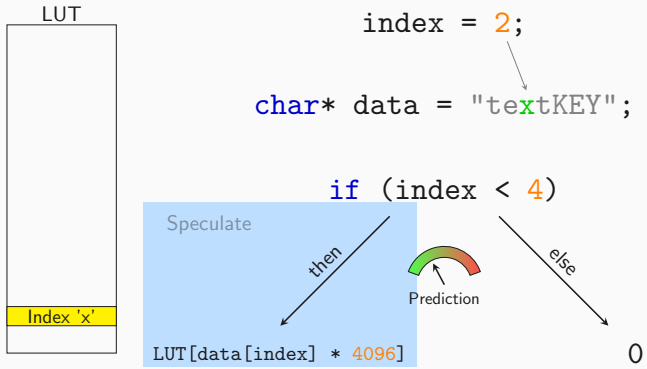

LUT
index = 1;
char* data = "textKEY";
if (index < 4)
then
Prediction
else
LUT[data[index] * 4096]
0

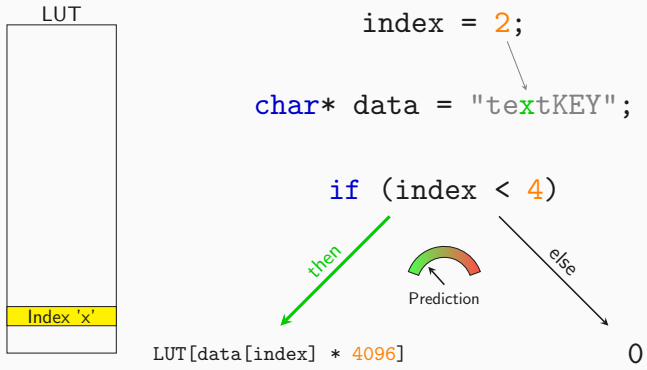

LUT
Index 'e'
index = 1;
char* data = "textKEY";
if (index < 4)
Speculate
then
Prediction
else
LUT[data[index] * 4096]
0

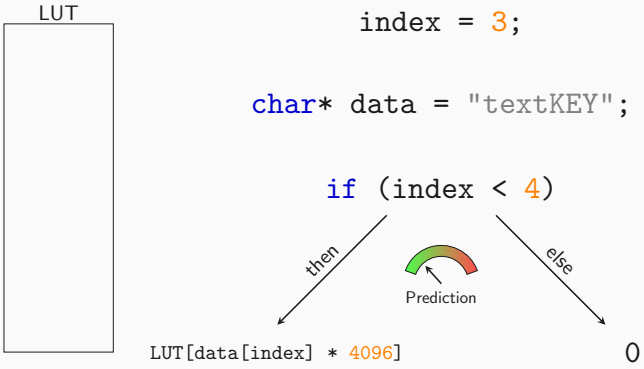

LUT
Index 'e'
index = 1;
char* data = "textKEY";
if (index < 4)
then
Prediction
else
LUT[data[index] * 4096]
0

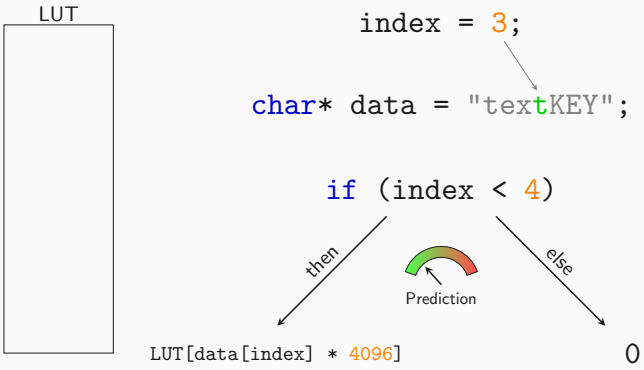

LUT
index = 2;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

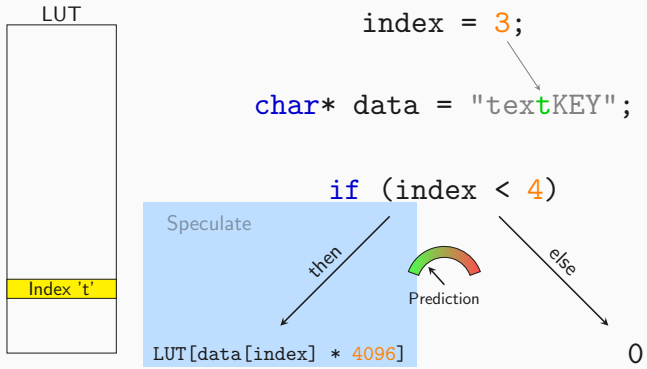

LUT
index = 2;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

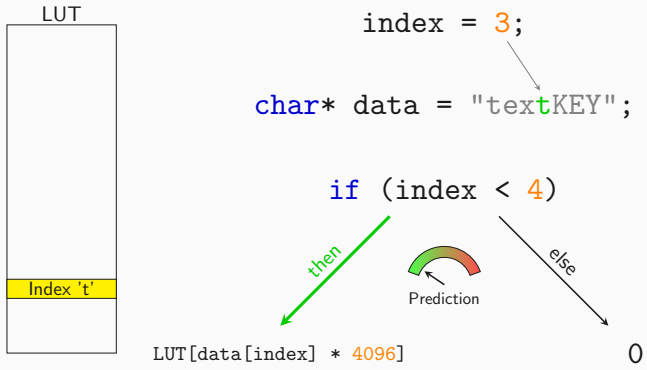

LUT
Index 'x'
index = 2;
char* data = "textKEY";
if (index < 4)
Speculate
then
else
Prediction
LUT[data[index] * 4096]
0

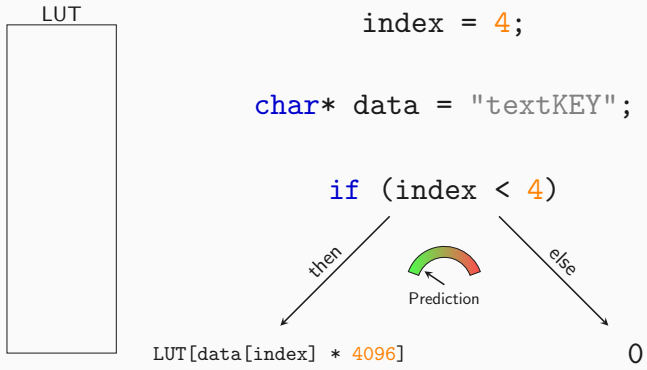

LUT
Index 'x'
index = 2;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

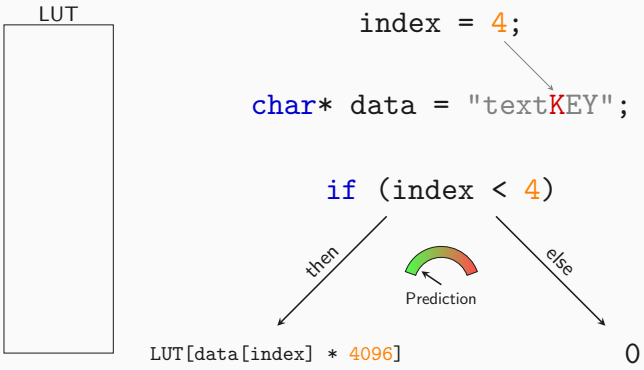

LUT
index = 3;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

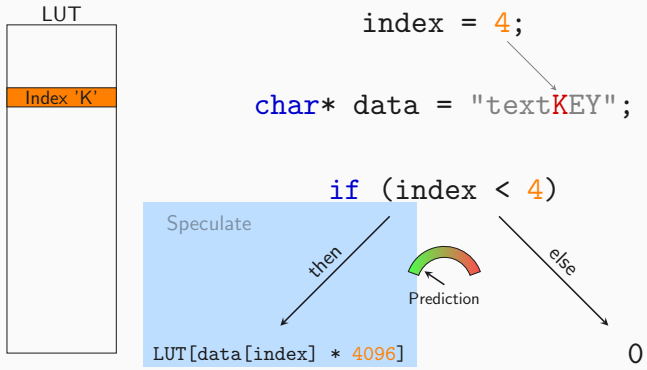

LUT
index = 3;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

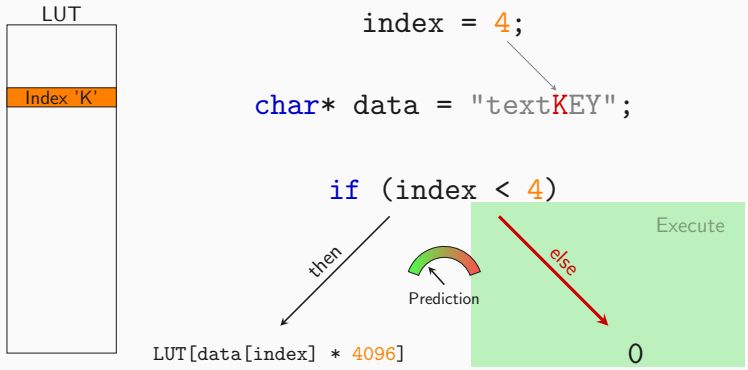

LUT
Index 't'
index = 3;
char* data = "textKEY";
if (index < 4)
Speculate
then
else
Prediction
LUT[data[index] * 4096]
0

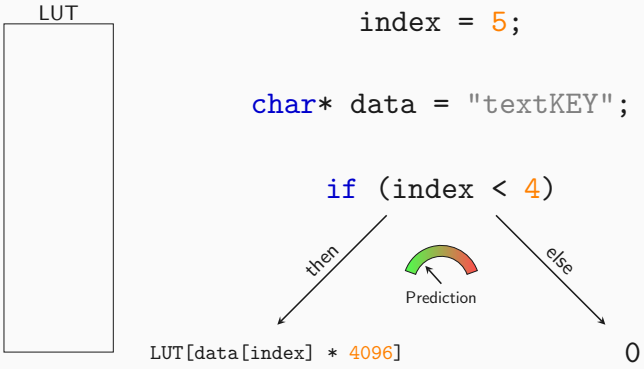

LUT
Index 't'
index = 3;
char* data = "textKEY";
if (index < 4)
then
Prediction
else
LUT[data[index] * 4096]
0

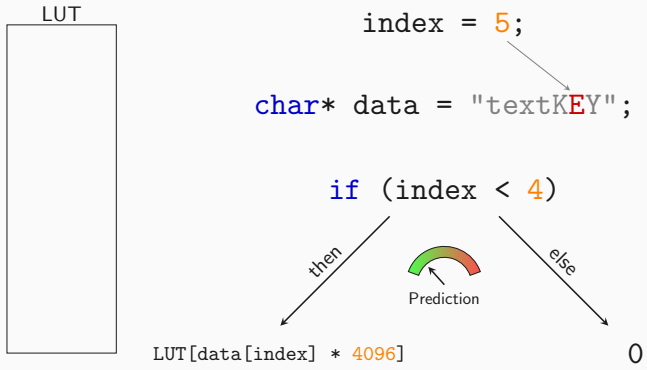

LUT
index = 4;
char* data = "textKEY";
if (index < 4)
then
Prediction
else
LUT[data[index] * 4096]
0

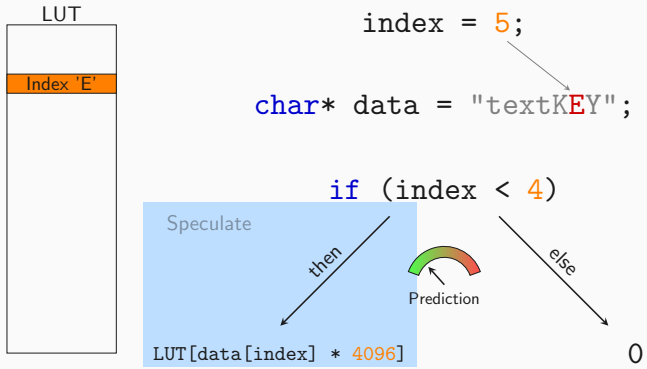

LUT
index = 4;
char* data = "textKEY";
if (index < 4)
then
Prediction
else
LUT[data[index] * 4096]
0

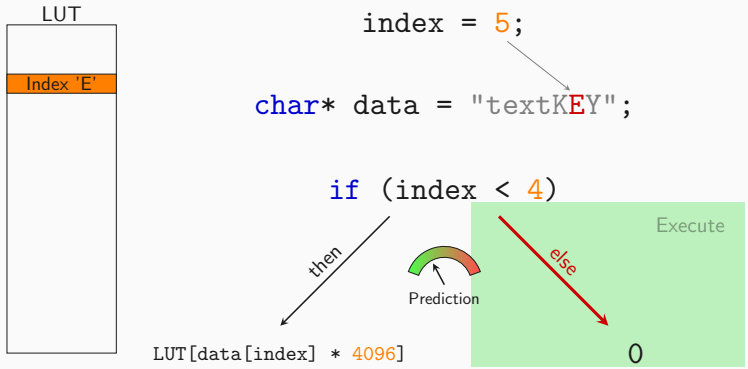

LUT
Index 'K'
index = 4;
char* data = "textKEY";
if (index < 4)
Speculate
then
else
Prediction
LUT[data[index] * 4096]
0

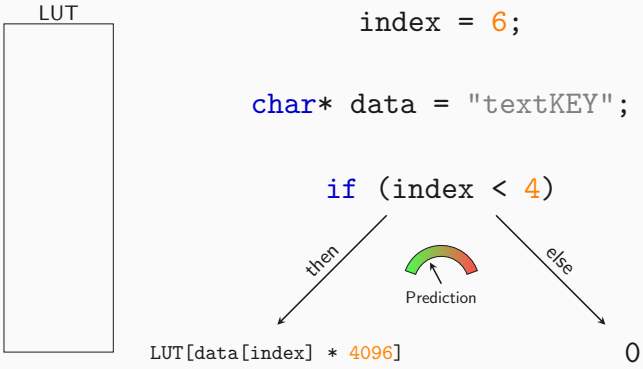

LUT
Index 'K'
index = 4;
char* data = "textKEY";
if (index < 4)
then
Prediction
else
Execute
LUT[data[index] * 4096]
0

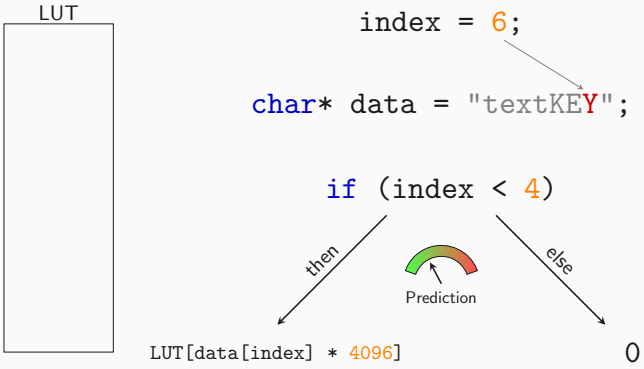

LUT
index = 5;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

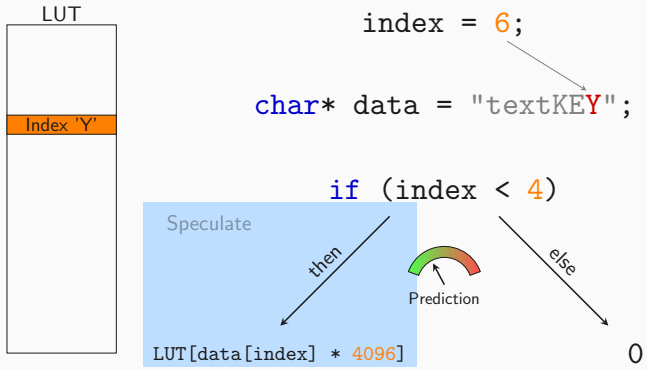

LUT
index = 5;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

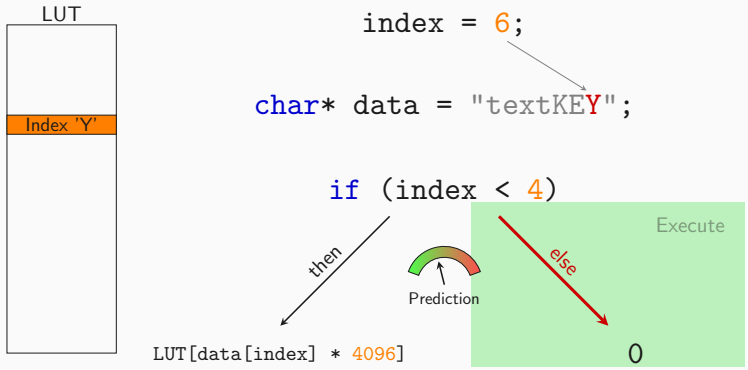

LUT
Index 'E'
index = 5;
char* data = "textKEY";
if (index < 4)
Speculate
then
else
Prediction
LUT[data[index] * 4096]
0

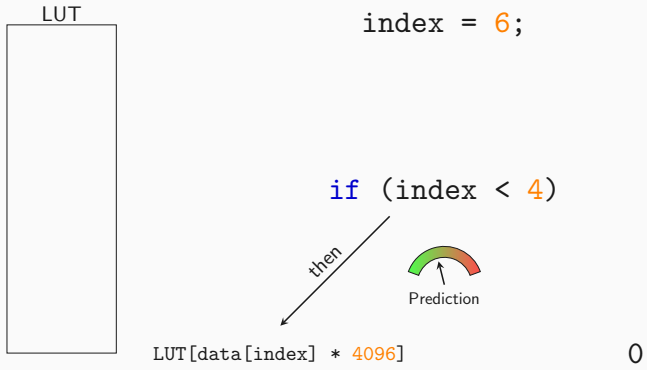

LUT
Index 'E'
index = 5;
char* data = "textKEY";
if (index < 4)
then
Prediction
else
Execute
LUT[data[index] * 4096]
0


LUT
index = 6;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0


LUT
index = 6;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

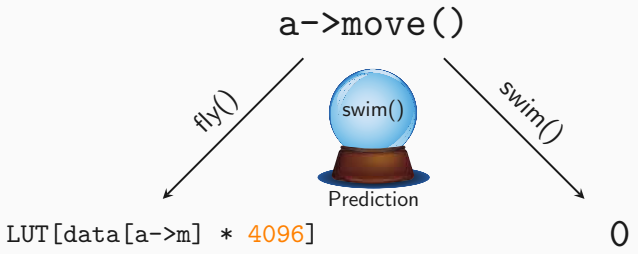

LUT
Index 'Y'
index = 6;
char* data = "textKEY";
if (index < 4)
Speculate
then
else
Prediction
LUT[data[index] * 4096]
0


LUT
Index 'Y'
index = 6;
char* data = "textKEY";
if (index < 4)
then
Prediction
else
Execute
LUT[data[index] * 4096]
0

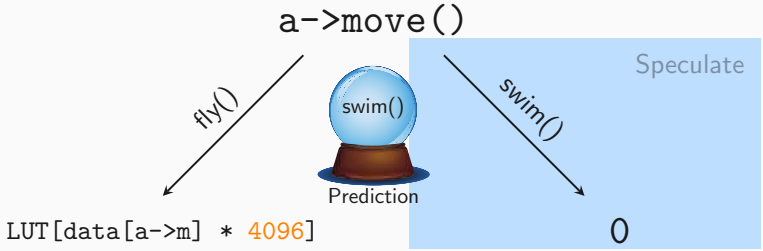


Spectre-STL (v4): Ignore sanitizing write access and use unsanitized old value instead

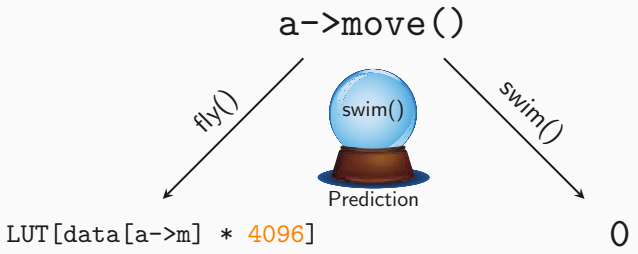
```
Animal* a = bird;
```



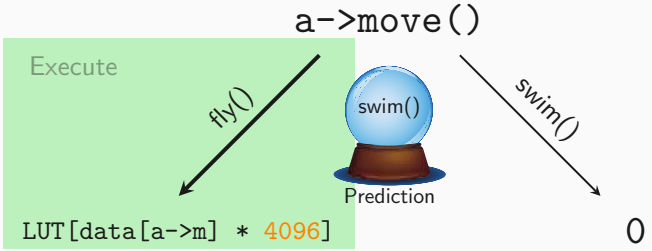
```
Animal* a = bird;
```



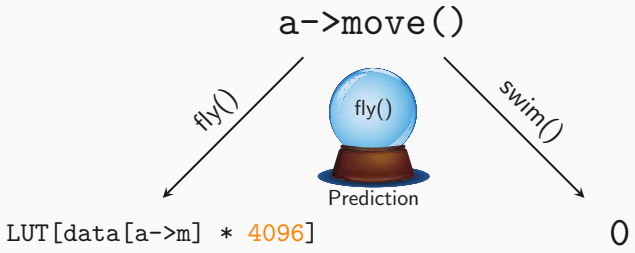
```
Animal* a = bird;
```



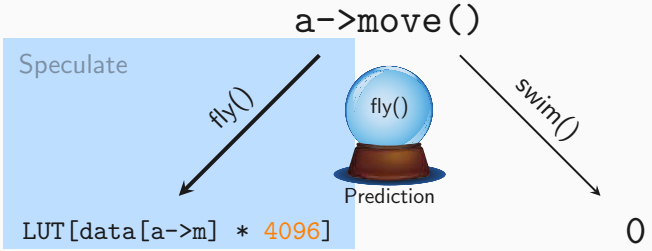
```
Animal* a = bird;
```



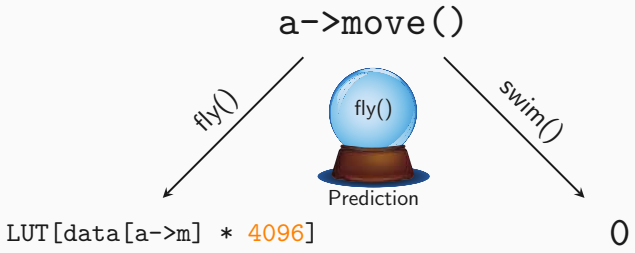
```
Animal* a = bird;
```



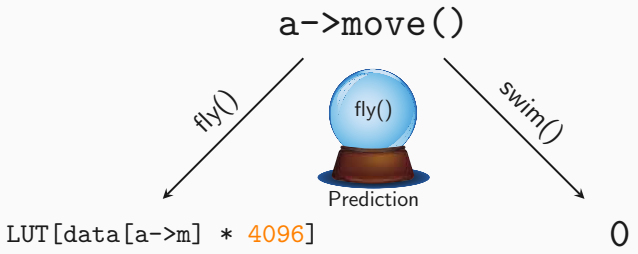
```
Animal* a = bird;
```



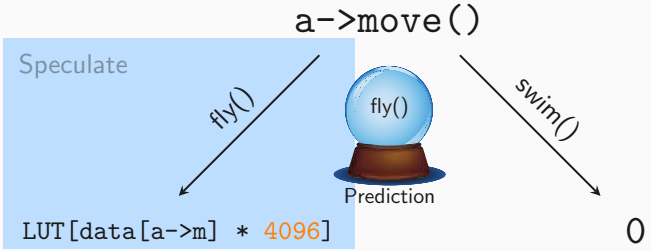
```
Animal* a = bird;
```



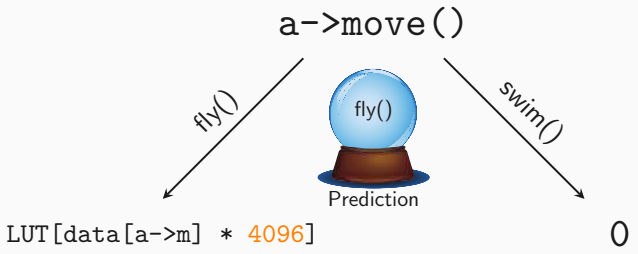
```
Animal* a = fish;
```



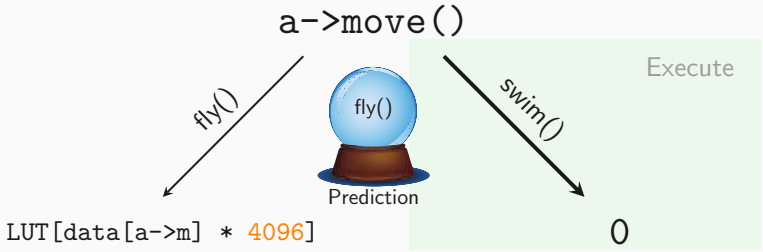
```
Animal* a = fish;
```



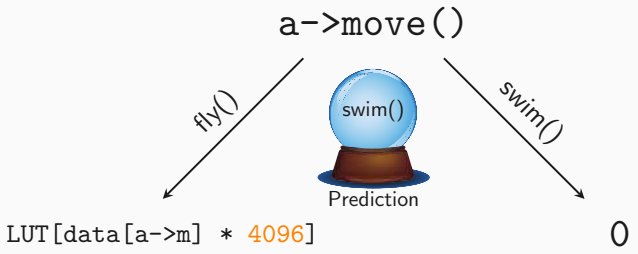
```
Animal* a = fish;
```



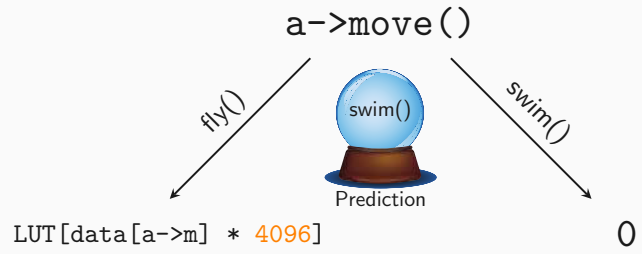
```
Animal* a = fish;
```



```
Animal* a = fish;
```

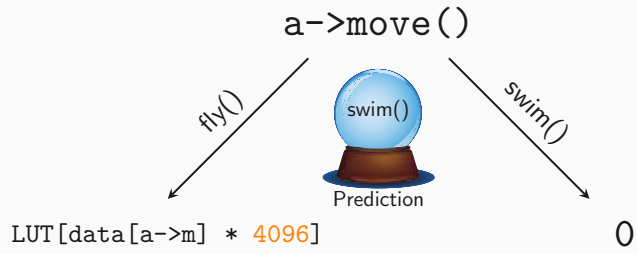


```
Animal* a = fish;
```



Spectre-BTB (v2): mistrain BTB → mispredict indirect jump/call

```
Animal* a = fish;
```



Spectre-BTB (v2): mistrain BTB → mispredict indirect jump/call

Spectre-RSB (v5): mistrain RSB → mispredict return

- v1.1: Speculatively write to memory locations

[2] Vladimir Kiriansky et al. Speculative Buffer Overflows: Attacks and Defenses. In: arXiv:1807.03757 (2018).

- v1.1: Speculatively write to memory locations

→ Many more gadgets than previously anticipated n

[2]Vladimir Kiriansky et al. Speculative Buffer Overflows: Attacks and Defenses. In: arXiv:1807.03757 (2018).

- v1.1: Speculatively write to memory locations

→ Many more gadgets than previously anticipated n

- v1.2: Ignore writable bit

[2]Vladimir Kiriansky et al. Speculative Buffer Overflows: Attacks and Defenses. In: arXiv:1807.03757 (2018).

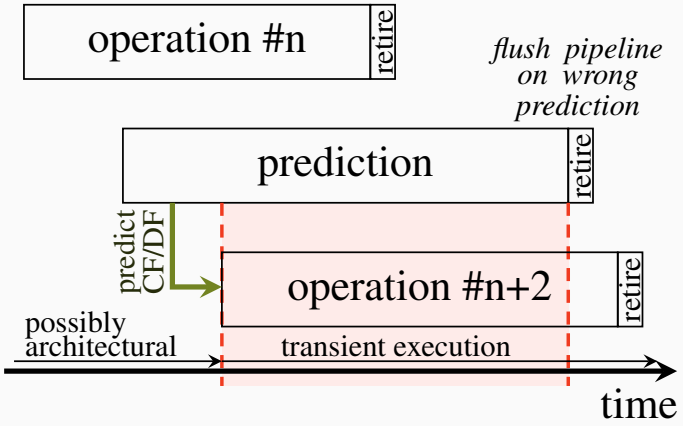
- v1.1: Speculatively write to memory locations

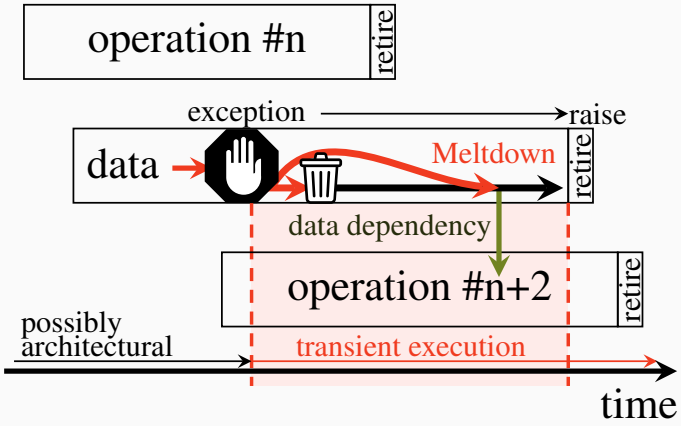
→ Many more gadgets than previously anticipated n

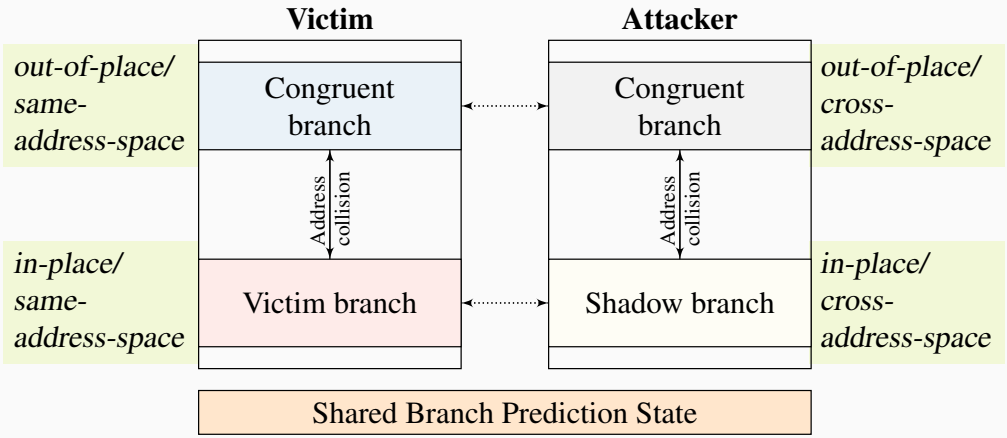
- v1.2: Ignore writable bit

→ = Meltdown-RW

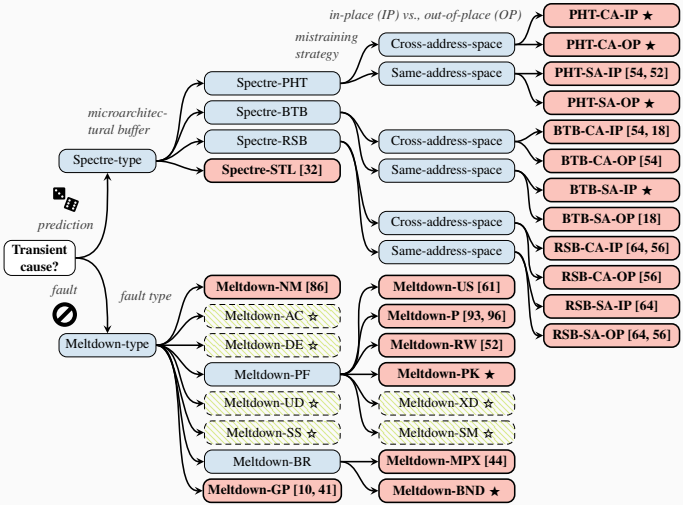
[2] Vladimir Kiriansky et al. Speculative Buffer Overflows: Attacks and Defenses. In: arXiv:1807.03757 (2018).


operation #n
retire
flush pipeline
on wrong
prediction
prediction
retire
predict
CF/DF
operation #n+2
retire
possibly
architectural
transient execution
time


operation #n
retire
exception
raise
data
Meltdown
retire
data dependency
operation #n+2
retire
possibly
architectural
transient execution
time


Victim
Attacker
out-of-place/ same-address-space
Congruent branch
Congruent branch
out-of-place/ cross-address-space
Address collision
Address collision
in-place/ same-address-space
Victim branch
Shadow branch
in-place/ cross-address-space
Shared Branch Prediction State

Classification Tree




BLOCKCHAIN
HD
HISTORY.COM

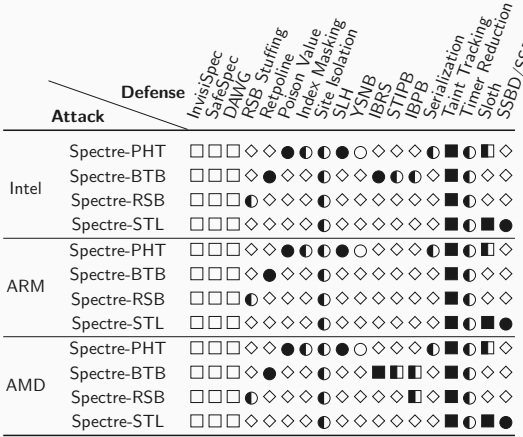
Computer Architecture Today

Informing the broad computing community about current activities, advances and future directions in computer architecture.

Let's Keep it to Ourselves: Don't Disclose Vulnerabilities

by Gus Uht on Jan 31, 2019 | Tags: Opinion, Security

Table 1: Spectre-type defenses and what they mitigate.

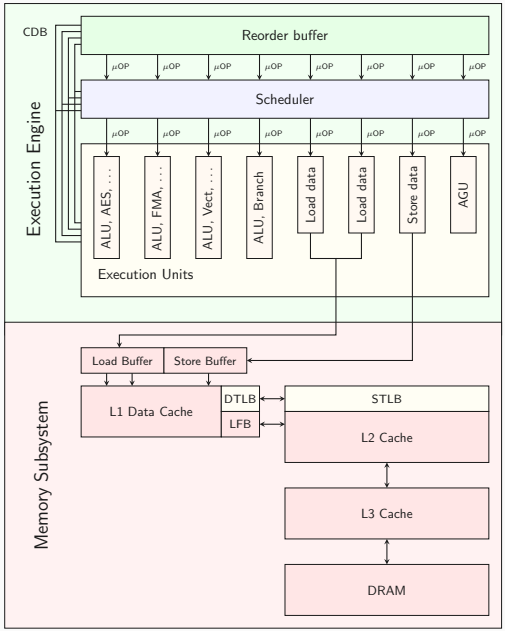


Symbols show if an attack is mitigated (●), partially mitigated (◐), not mitigated (○), theoretically mitigated (■), theoretically impeded (◧), not theoretically impeded (□), or out of scope (◇)

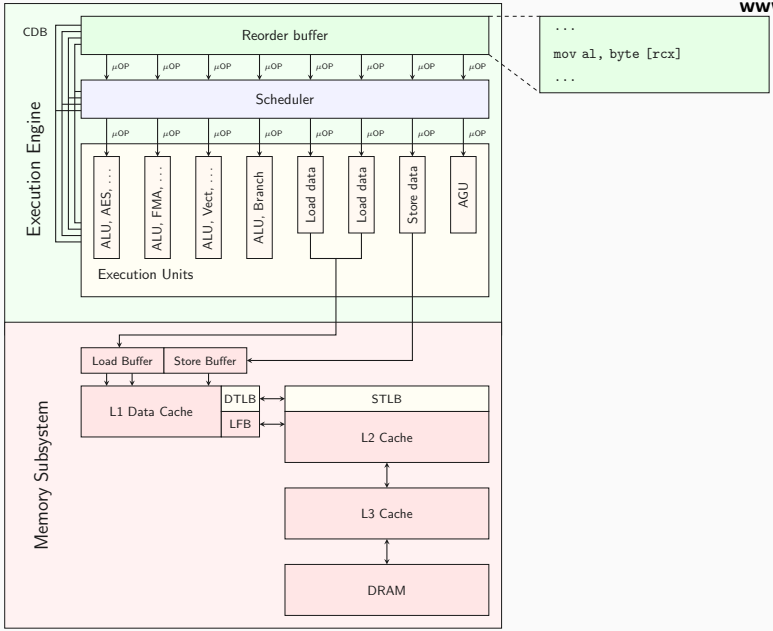
Table 2: Reported performance impacts of countermeasures

Defense \ Impact	Performance Loss	Benchmark
InvisiSpec	22%	SPEC
SafeSpec	3% (improvement)	SPEC2017 on MARSSx86
DAWG	2–12%, 1–15%	PARSEC, GAPBS
RSB Stuffing	no reports	
Retpoline	5–10%	real-world workload servers
Site Isolation	only memory overhead	
SLH	36.4%, 29%	Google microbenchmark suite
YSNB	60%	Phoenix
IBRS	20–30%	two sysbench 1.0.11 benchmarks
STIPB	30– 50%	Rodinia OpenMP, DaCapo
IBPB	no individual reports	
Serialization	62%, 74.8%	Google microbenchmark suite
SSBD/SSBB	2–8%	SYSmark®2014 SE & SPEC integer
KAISER/KPTI	0–2.6%	system call rates
L1TF mitigations	-3–31%	various SPEC

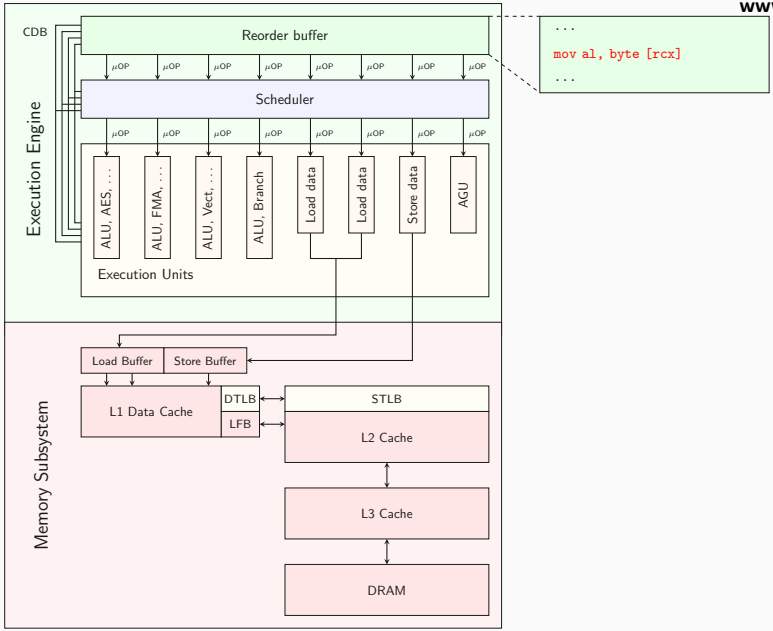
Meltdown


CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem

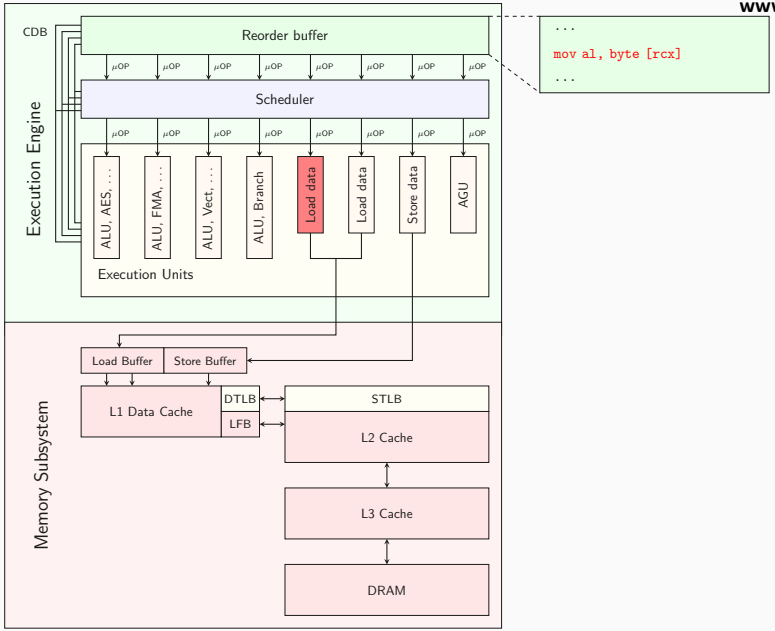
Meltdown


...
mov al, byte [rcx]
...
CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem

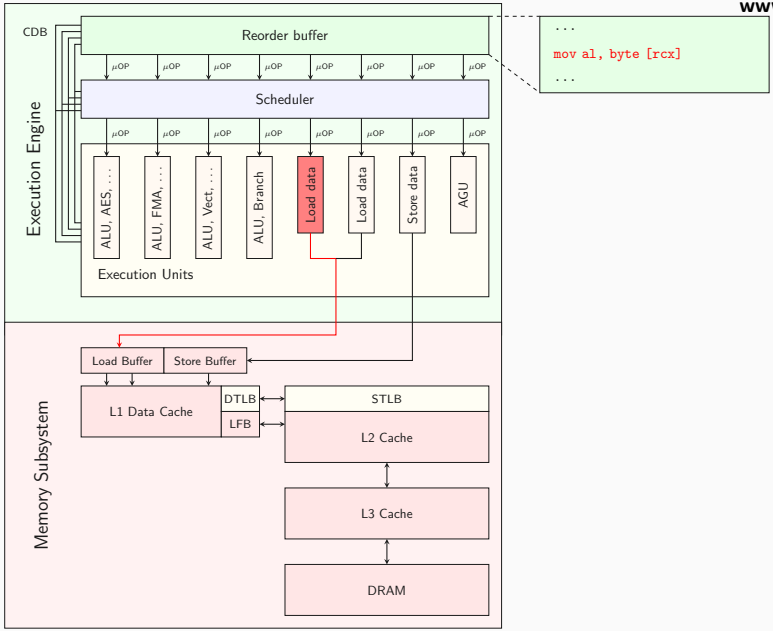
Meltdown


...
mov al, byte [rcx]
...
CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem

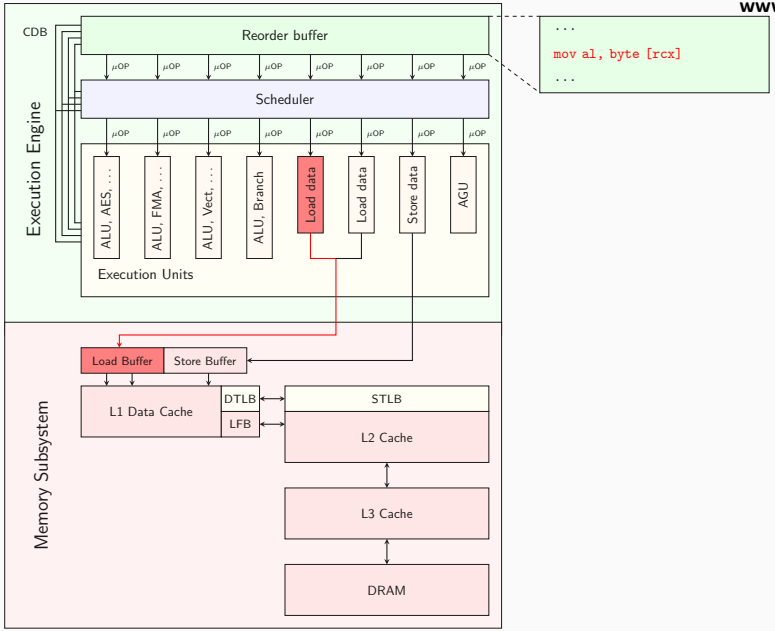
Meltdown


...
mov al, byte [rcx]
...
CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem

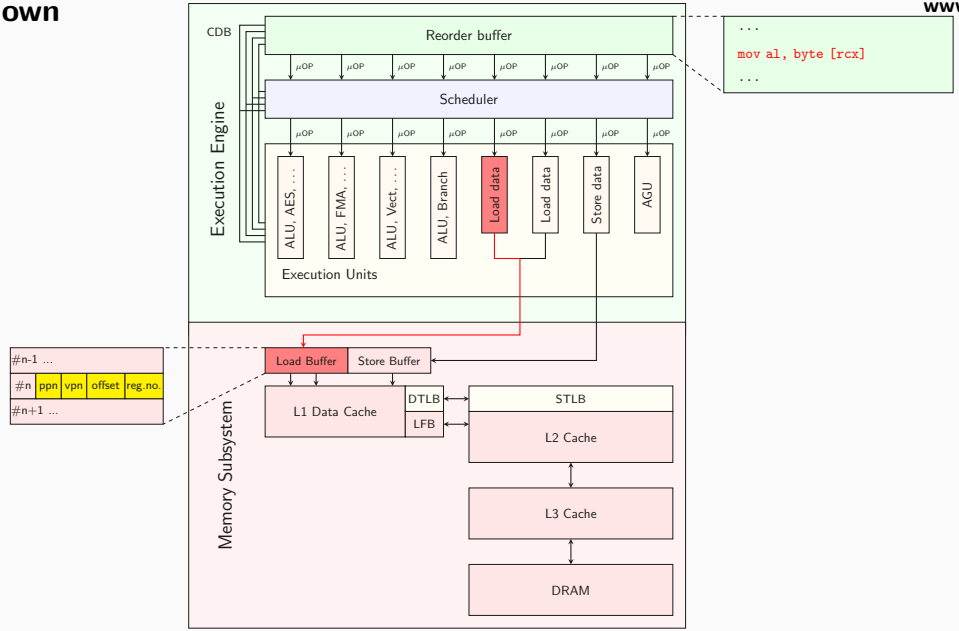
Meltdown


...
mov al, byte [rcx]
...
CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem

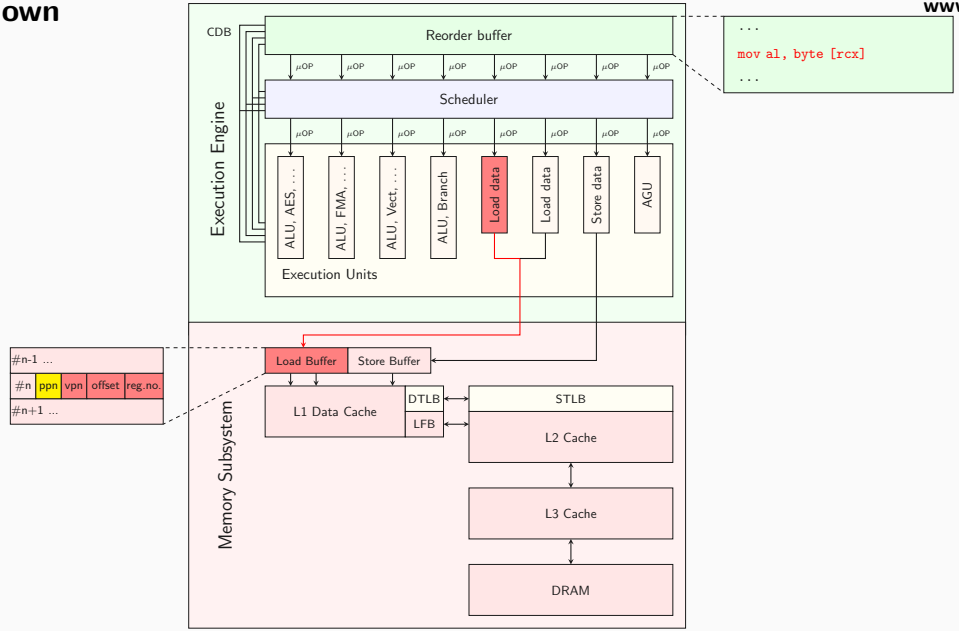
Meltdown


...
mov al, byte [rcx]
...
CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem

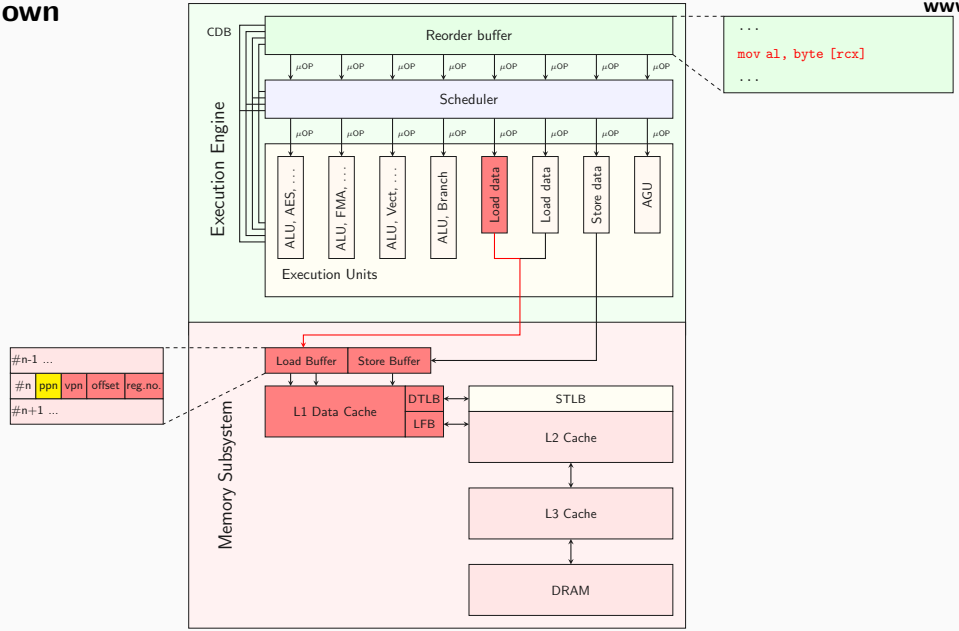
Meltdown


CDB
Reorder buffer
...
mov al, byte [rcx]
...
µOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
#n-1 ...
#n
ppn
vpn
offset
reg. no.
#n+1 ...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem

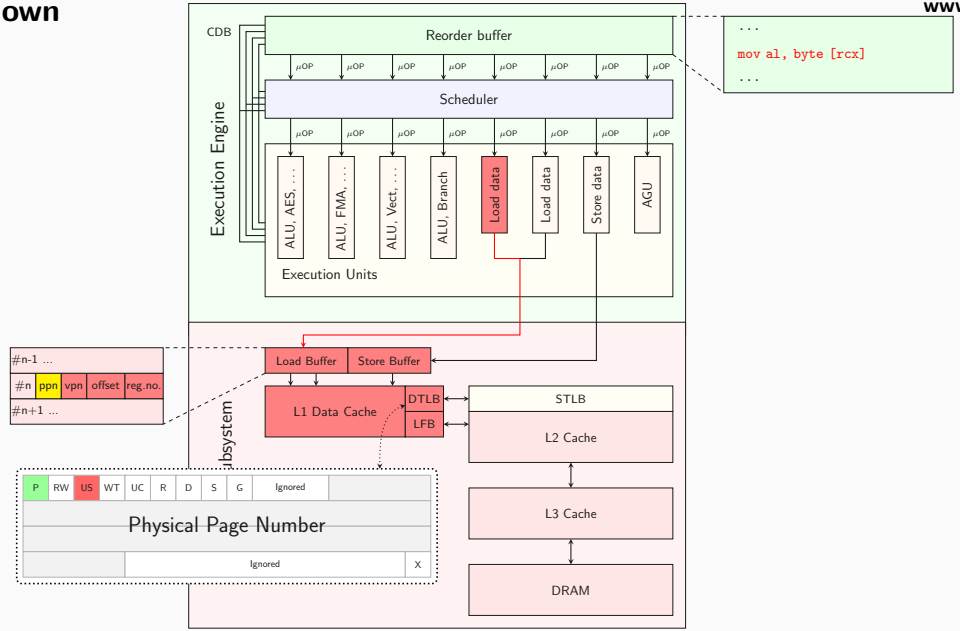
Meltdown


...
mov al, byte [rcx]
...
CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem

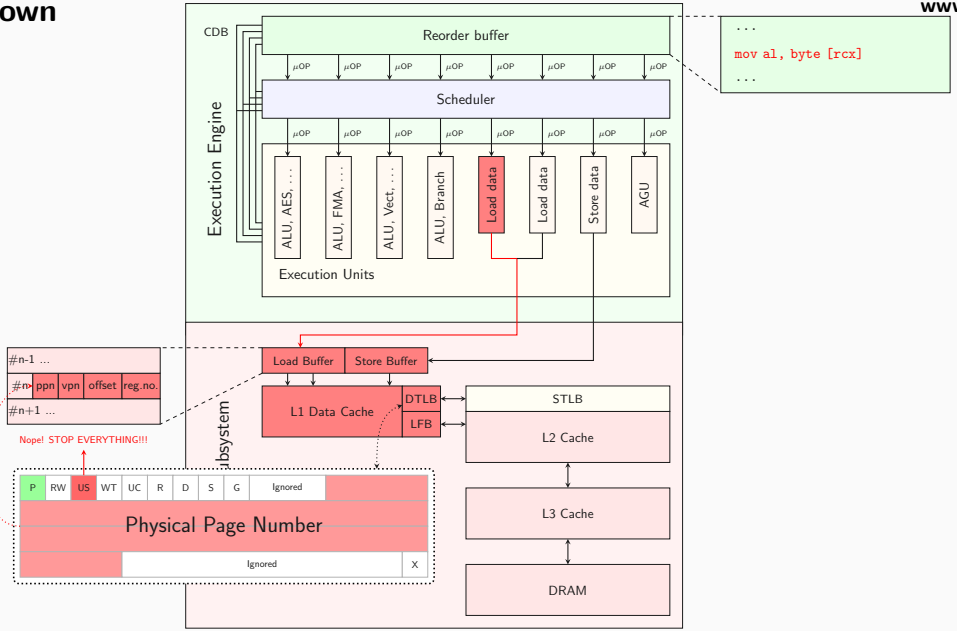
Meltdown


...
mov al, byte [rcx]
...
CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem

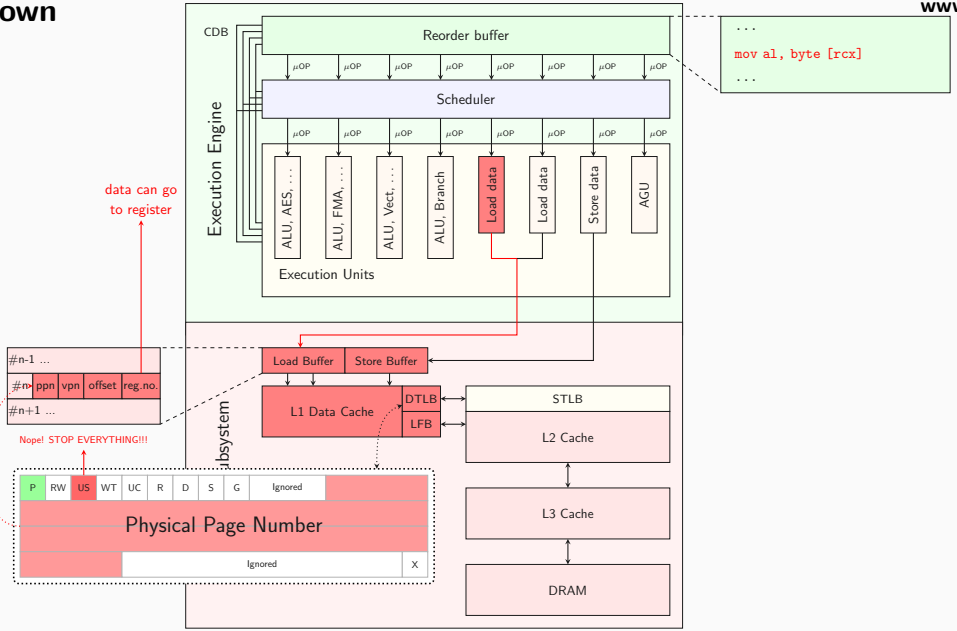
Meltdown


...
mov al, byte [rcx]
...
CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
bsystem
P
RW
US
WT
UC
R
D
S
G
Ignored
Physical Page Number
Ignored
X

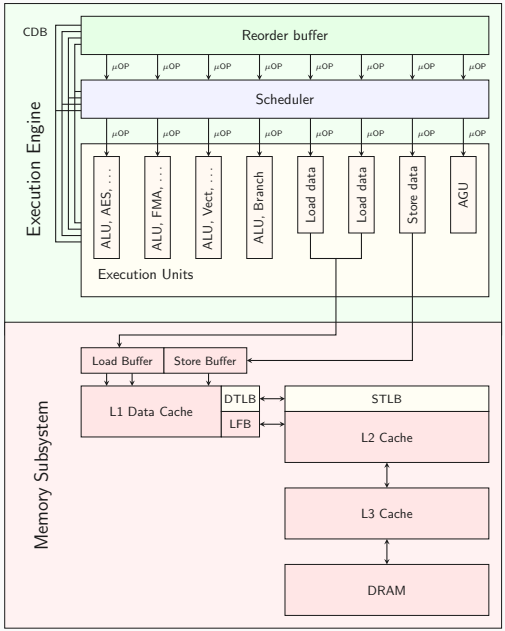
Meltdown


CDB
Reorder buffer
µOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
...
mov al, byte [rcx]
...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
#n-1 ...
#n ppn vpn offset reg.no.
#n+1 ...
Nope! STOP EVERYTHING!!!
P RW US WT UC R D S G Ignored
Physical Page Number
Ignored X

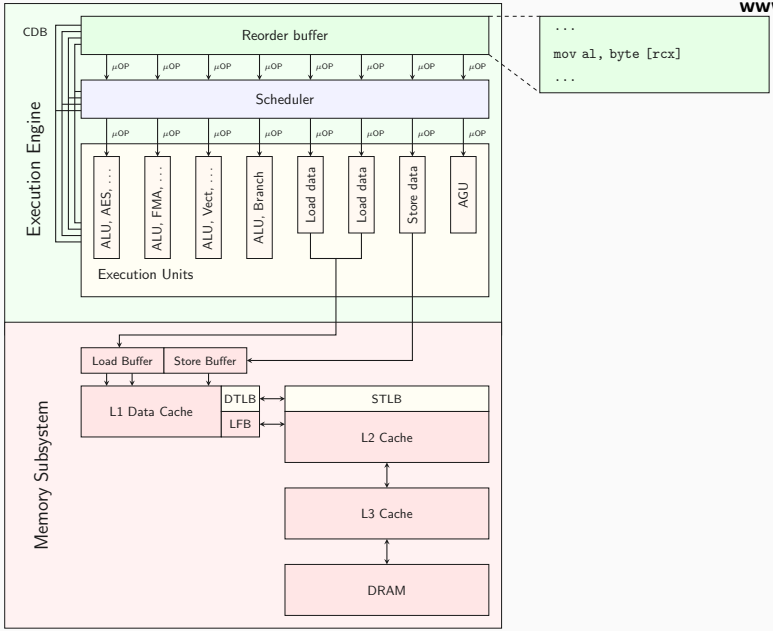
Meltdown


...
mov al, byte [rcx]
...
CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
data can go to register
#n-1 ...
#n ppn vpn offset reg.no.
#n+1 ...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Nope! STOP EVERYTHING!!!
P RW US WT UC R D S G Ignored
Physical Page Number
Ignored X

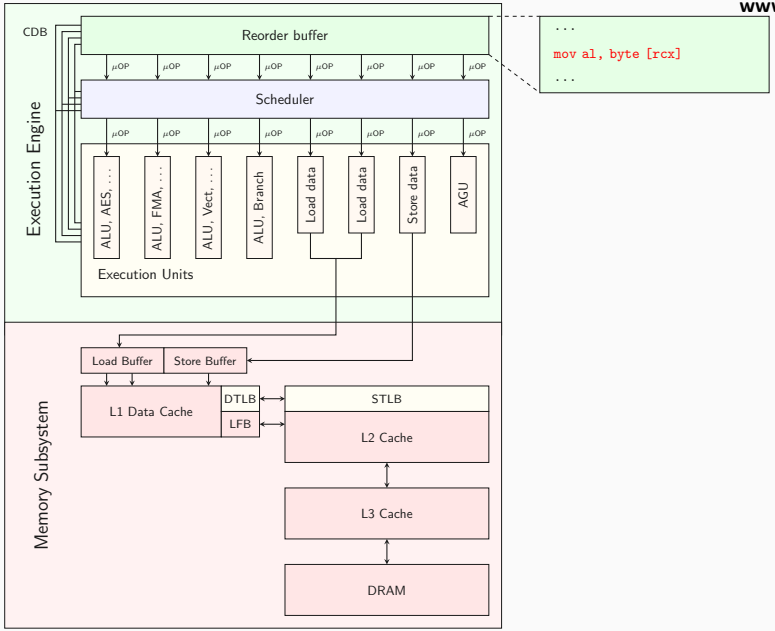
Foreshadow-VMM



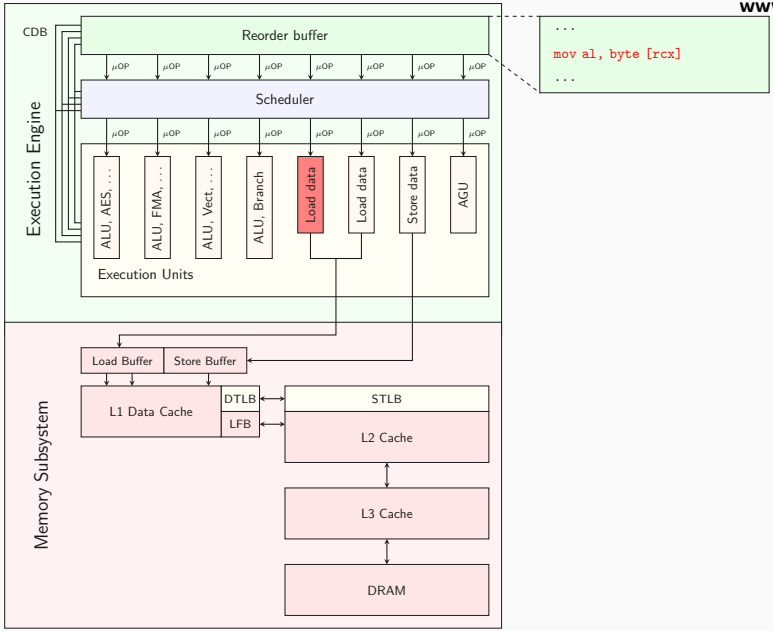
Foreshadow-VMM



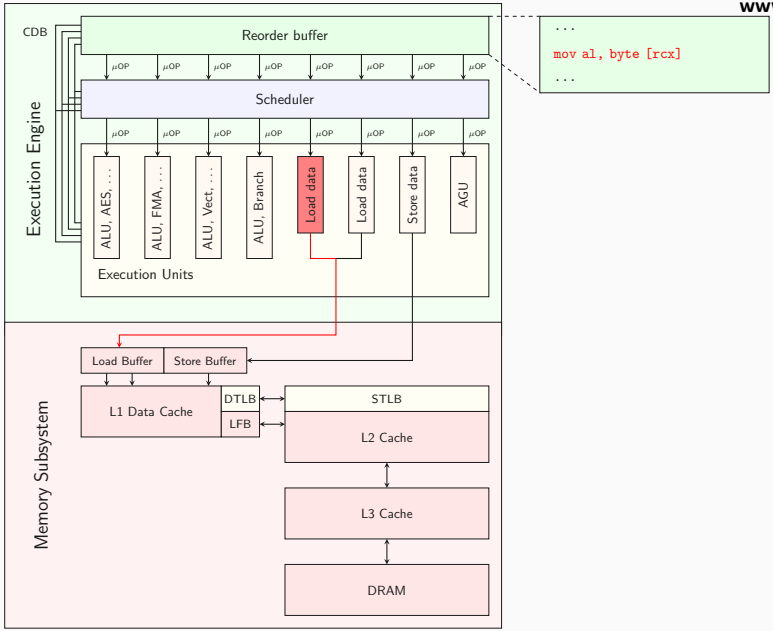
Foreshadow-VMM



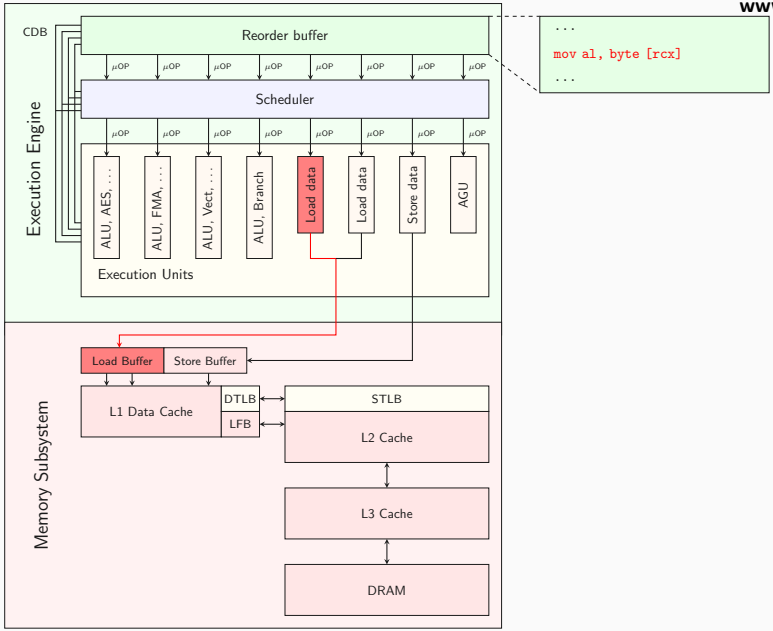
Foreshadow-VMM



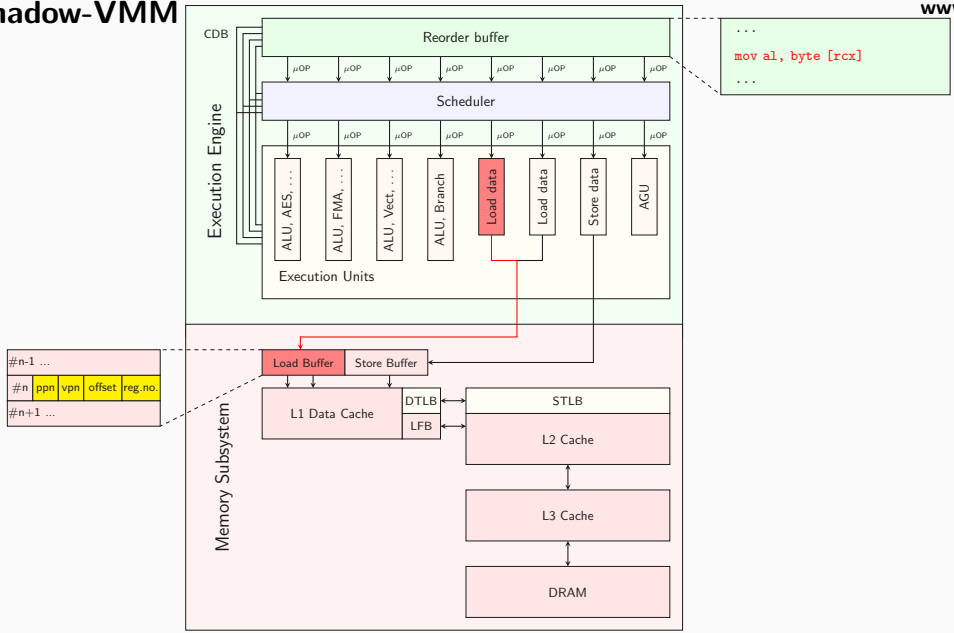
Foreshadow-VMM



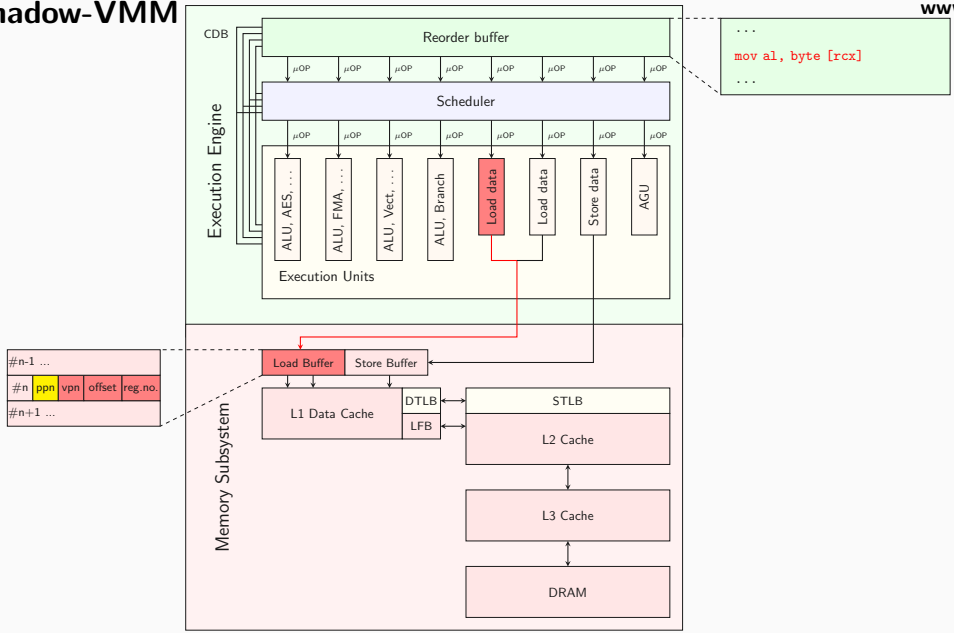
Foreshadow-VMM



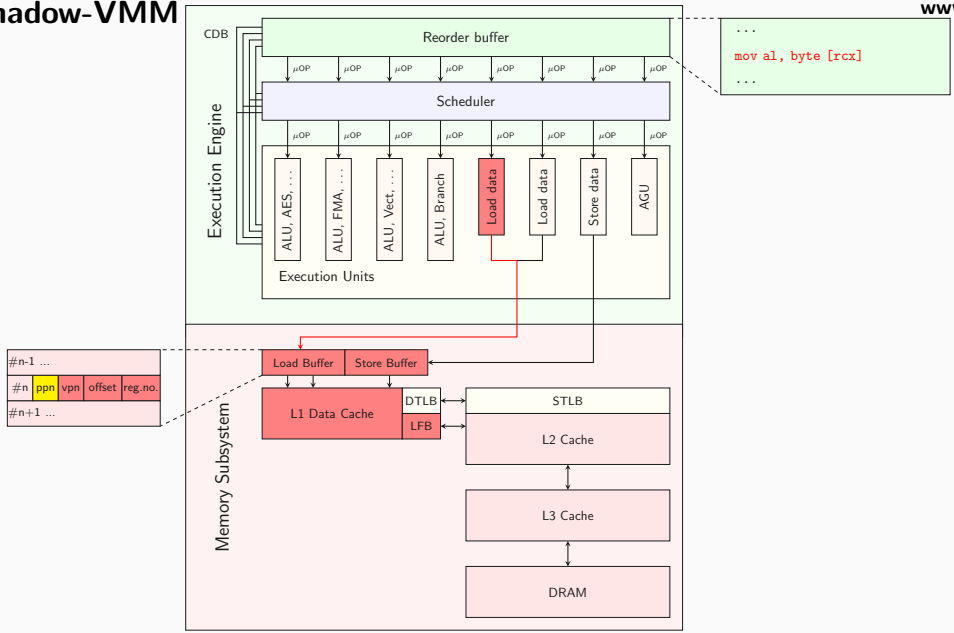
Foreshadow-VMM



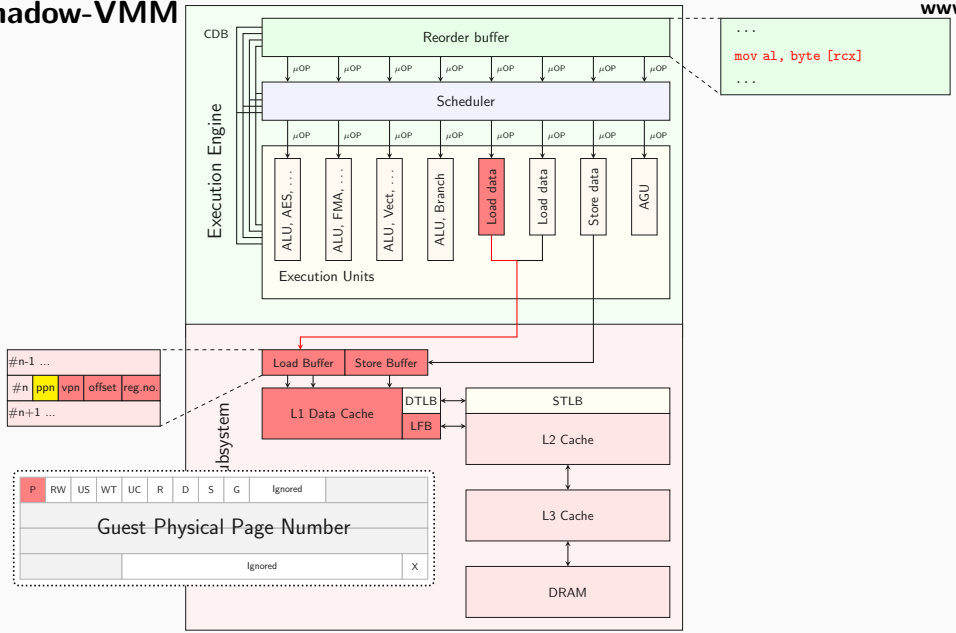
Foreshadow-VMM



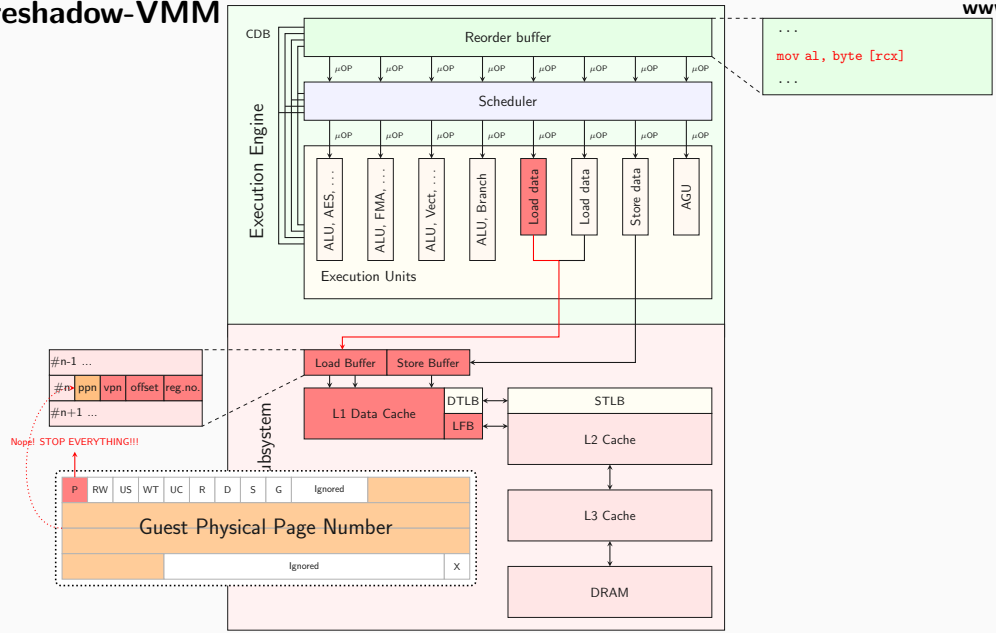
Foreshadow-VMM



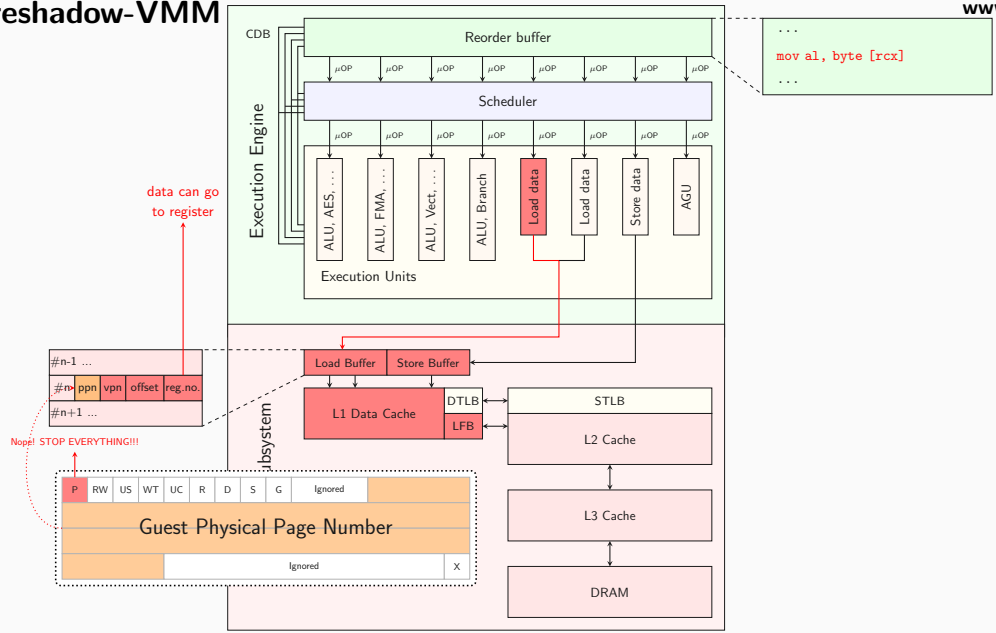
Foreshadow-VMM

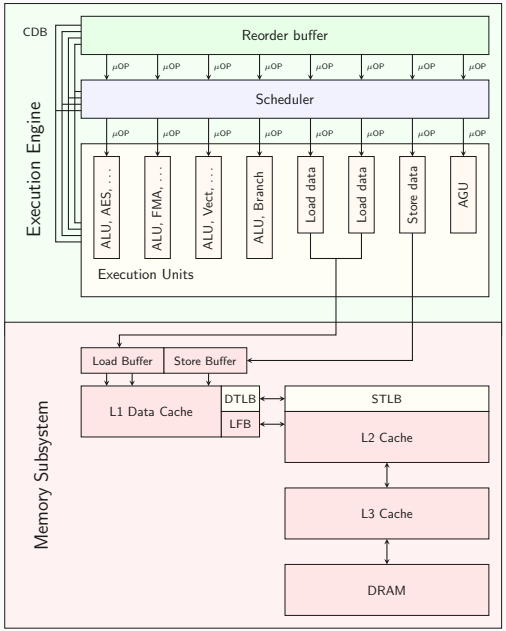


Foreshadow-VMM

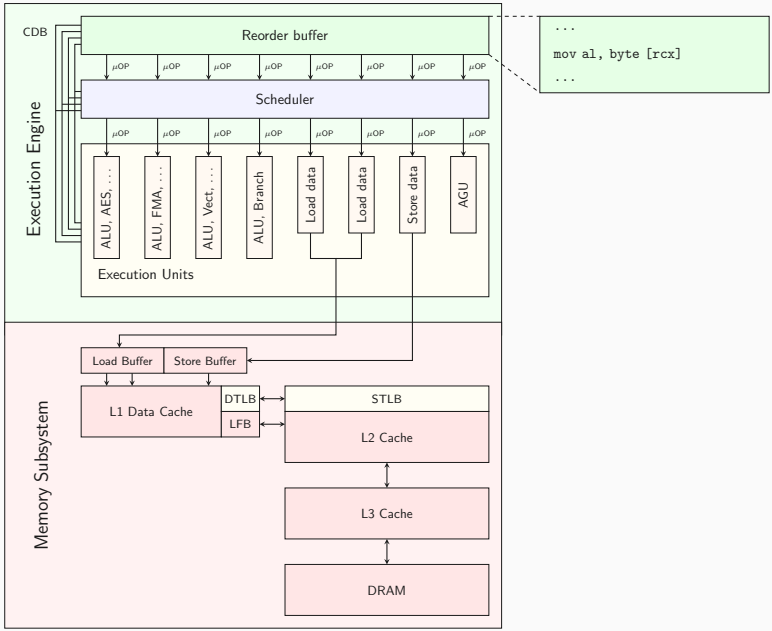


Foreshadow-VMM

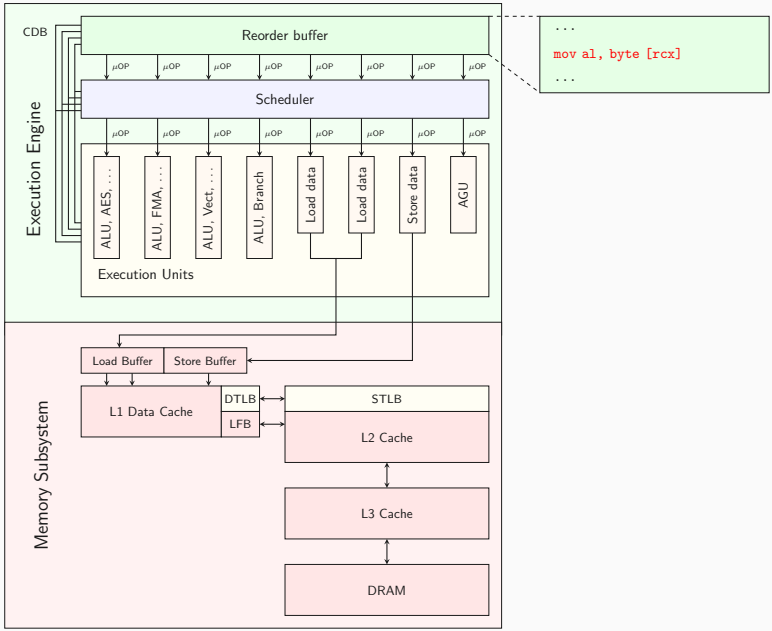



CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Memory Subsystem
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM

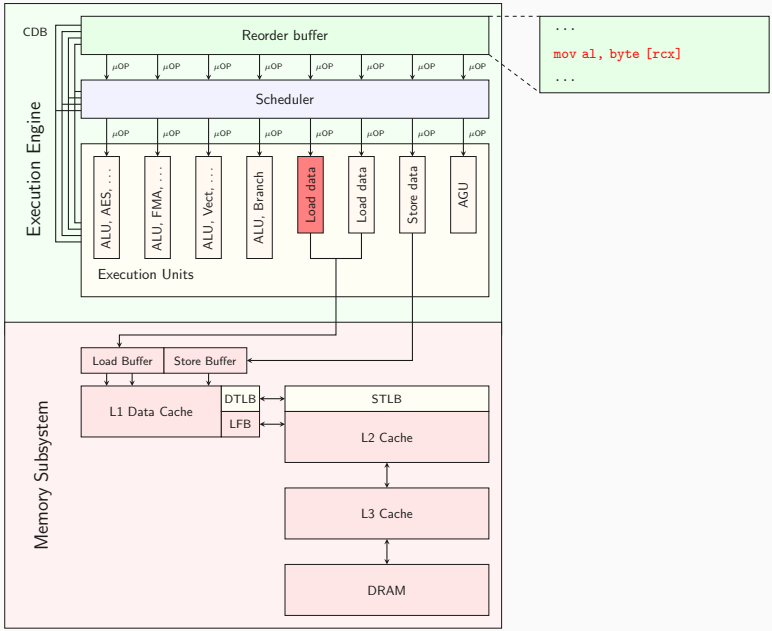
ZombieLoad


CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
...
mov al, byte [rcx]
...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem

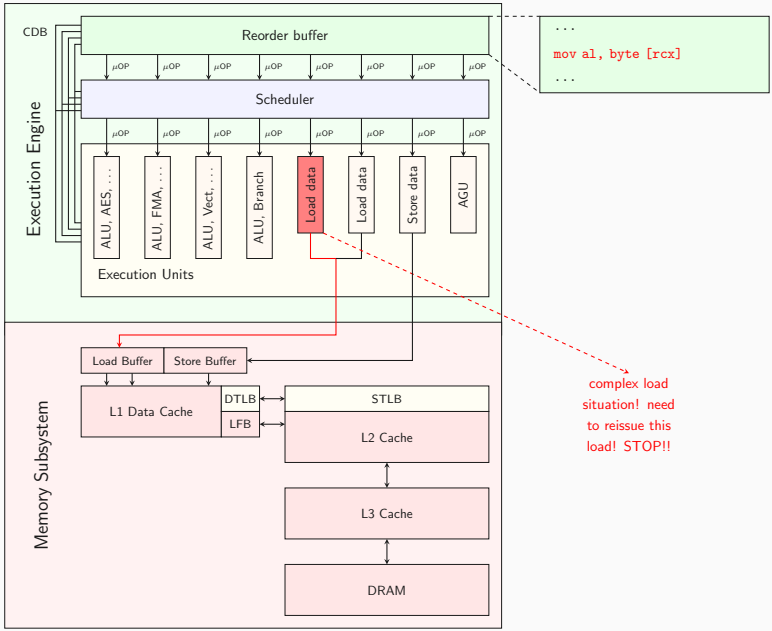
ZombieLoad


CDB
Reorder buffer
...
mov al, byte [rcx]
...
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem

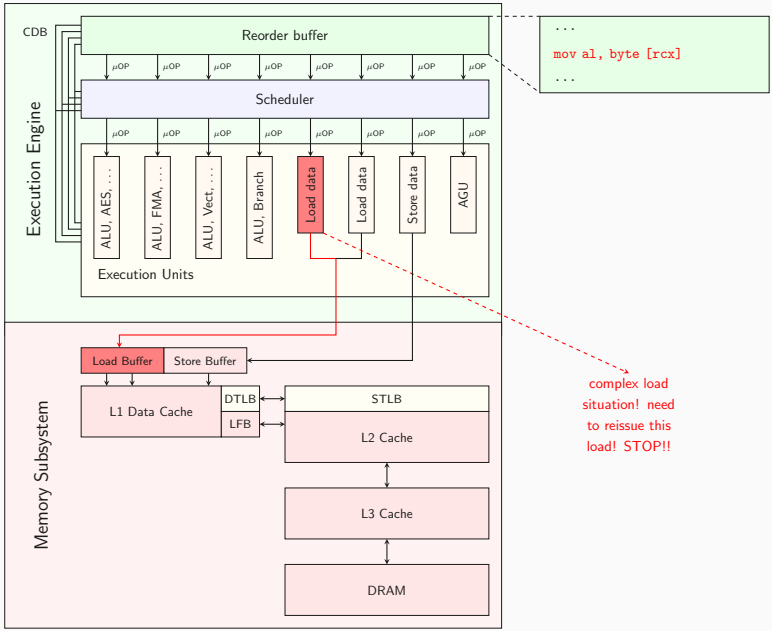
ZombieLoad


CDB
Reorder buffer
...
mov al, byte [rcx]
...
µOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem

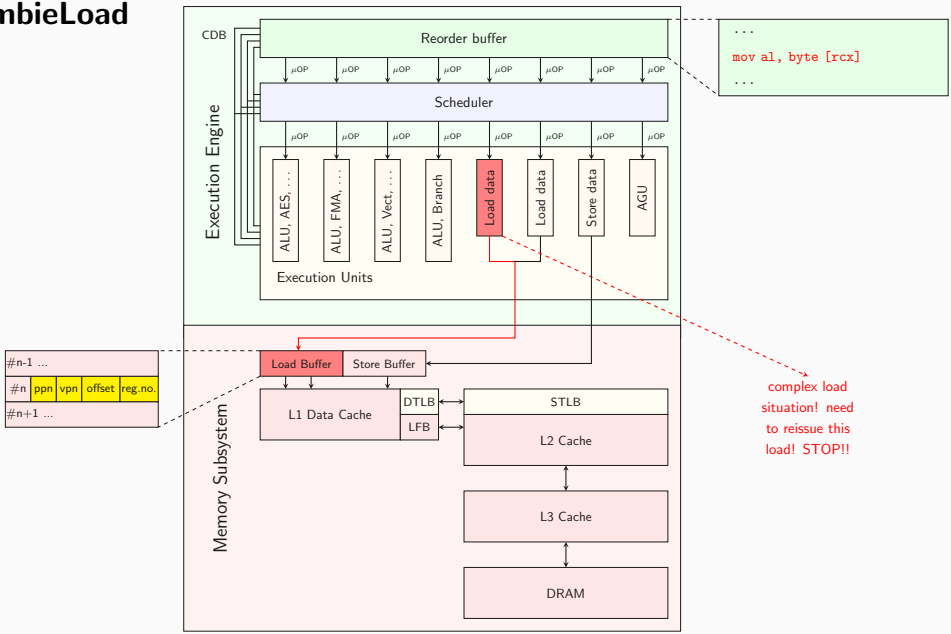
ZombieLoad


CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
...
mov al, byte [rcx]
...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem
complex load situation! need to reissue this load! STOP!!

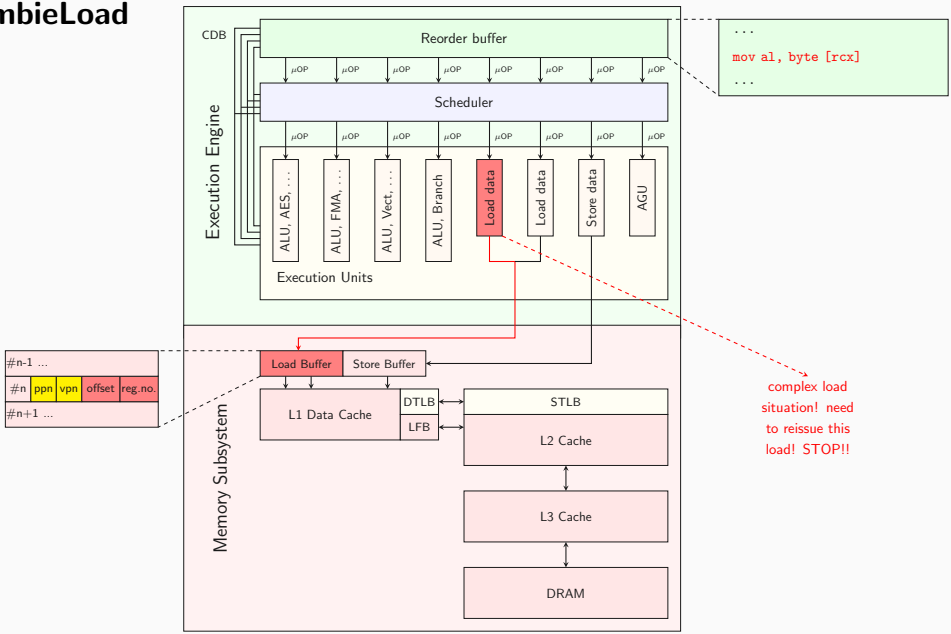
ZombieLoad


CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
...
mov al, byte [rcx]
...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem
complex load situation! need to reissue this load! STOP!!

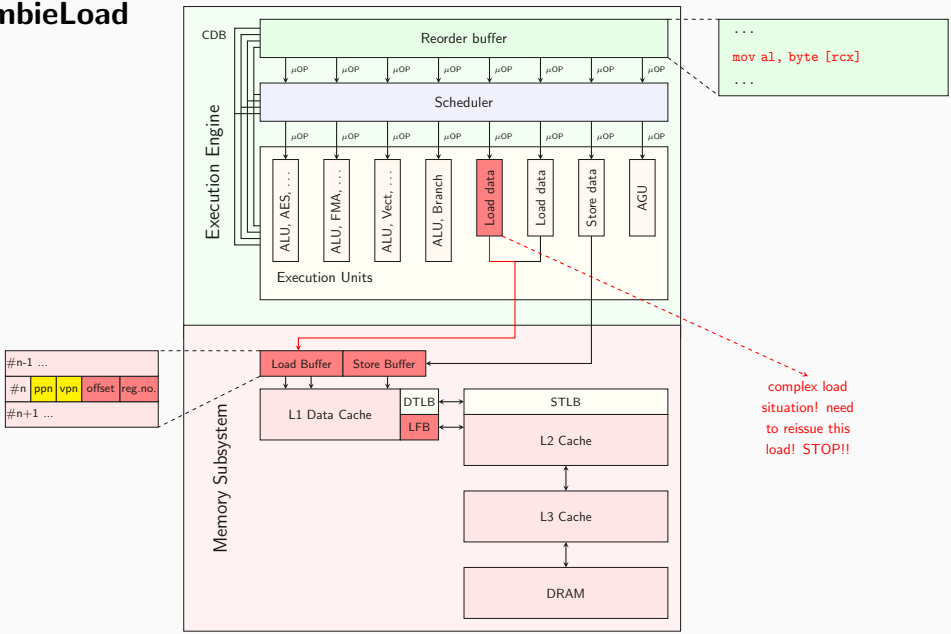
ZombieLoad


CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
...
mov al, byte [rcx]
...
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem
complex load situation! need to reissue this load! STOP!!

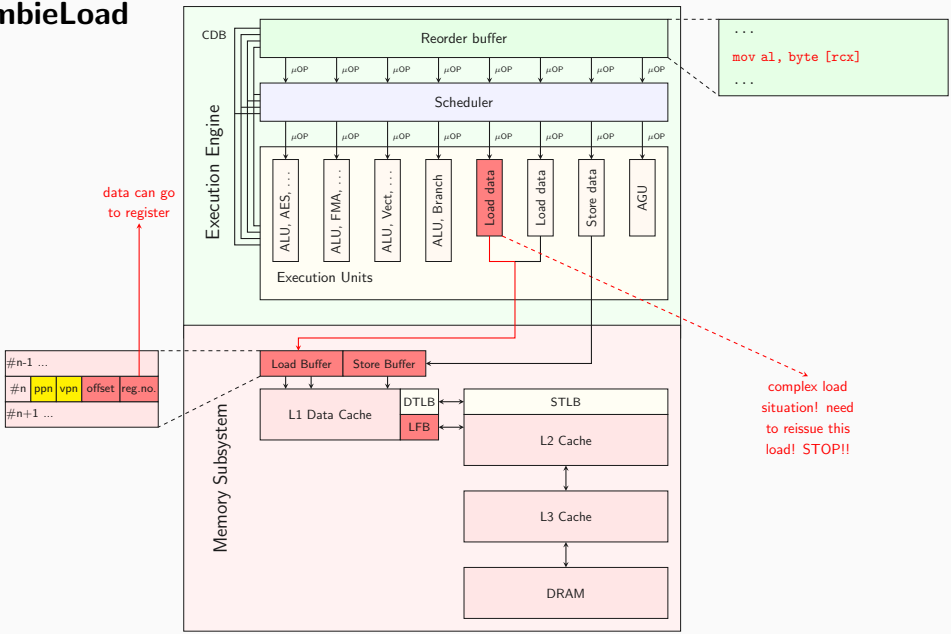
ZombieLoad


CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
...
mov al, byte [rcx]
...
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem
complex load situation! need to reissue this load! STOP!!

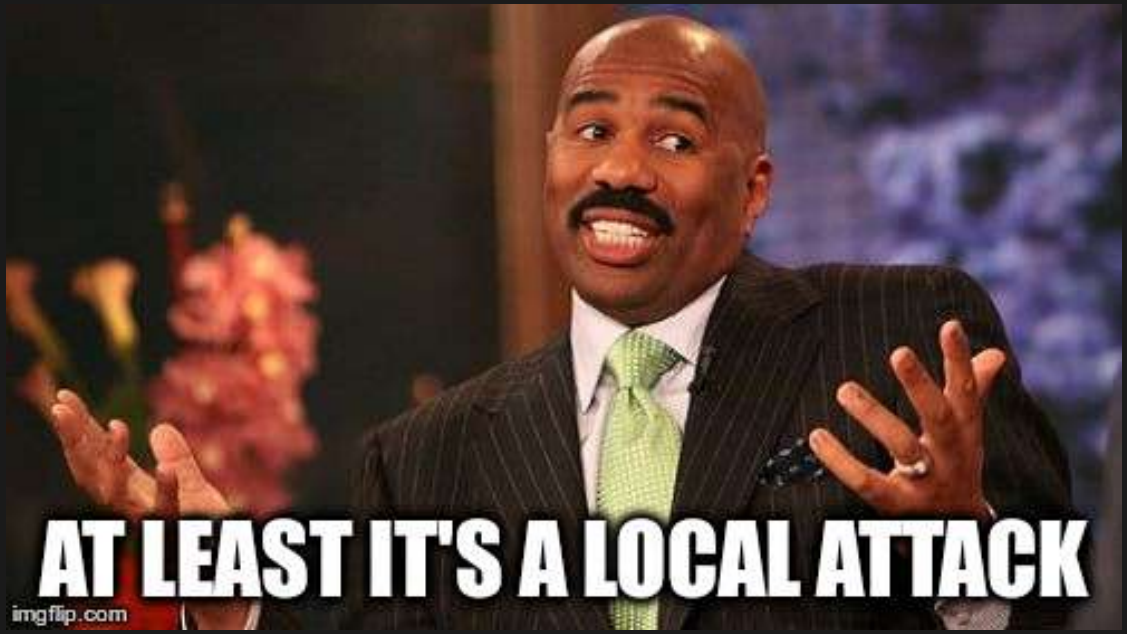
ZombieLoad


...
mov al, byte [rcx]
...
CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem
complex load situation! need to reissue this load! STOP!!

ZombieLoad


...
mov al, byte [rcx]
...
CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
data can go to register
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem
complex load situation! need to reissue this load! STOP!!


zombieload : zsh — Konsole <2>
File Edit View Bookmarks Settings Help
michael@hp /tmp/zombieload %
zombieload : zsh


AT LEAST IT'S A LOCAL ATTACK
imgflip.com

Just a few examples:

Just a few examples:

- Remote timing attacks on crypto ([Ber04; BB05] and many more)

Just a few examples:

- Remote timing attacks on crypto ([Ber04; BB05] and many more)
- ThrowHammer and NetHammer

Just a few examples:

- Remote timing attacks on crypto ([Ber04; BB05] and many more)
- ThrowHammer and NetHammer
- NetSpectre

We have ignored microarchitectural attacks for many years:

We have ignored microarchitectural attacks for many years:

- attacks on crypto

We have ignored microarchitectural attacks for many years:

- attacks on crypto → "software should be fixed"

We have ignored microarchitectural attacks for many years:

- attacks on crypto → "software should be fixed"
- attacks on ASLR

We have ignored microarchitectural attacks for many years:

- attacks on crypto → "software should be fixed"
- attacks on ASLR → "ASLR is broken anyway"

We have ignored microarchitectural attacks for many years:

- attacks on crypto → "software should be fixed"
- attacks on ASLR → "ASLR is broken anyway"
- attacks on SGX and TrustZone

We have ignored microarchitectural attacks for many years:

- attacks on crypto → "software should be fixed"
- attacks on ASLR → "ASLR is broken anyway"
- attacks on SGX and TrustZone → "not part of the threat model"

We have ignored microarchitectural attacks for many years:

- attacks on crypto → "software should be fixed"
- attacks on ASLR → "ASLR is broken anyway"
- attacks on SGX and TrustZone → "not part of the threat model"
- Rowhammer

We have ignored microarchitectural attacks for many years:

- attacks on crypto → "software should be fixed"
- attacks on ASLR → "ASLR is broken anyway"
- attacks on SGX and TrustZone → "not part of the threat model"
- Rowhammer → "only affects cheap sub-standard modules"

We have ignored microarchitectural attacks for many years:

- attacks on crypto → "software should be fixed"
- attacks on ASLR → "ASLR is broken anyway"
- attacks on SGX and TrustZone → "not part of the threat model"
- Rowhammer → "only affects cheap sub-standard modules"

→ for years we solely optimized for performance

... and we're still optimizing for performance

- lower refresh rate = lower energy but more bit flips

- lower refresh rate = lower energy but more bit flips
- ECC memory → fewer bit flips

- lower refresh rate = lower energy but more bit flips
- ECC memory → fewer bit flips

→ it's an optimization problem

- lower refresh rate = lower energy but more bit flips
- ECC memory → fewer bit flips

→ it's an optimization problem

  - what if "too aggressive" changes over time?

- lower refresh rate = lower energy but more bit flips
- ECC memory → fewer bit flips

→ it's an optimization problem

  - what if "too aggressive" changes over time?

  → difficult to optimize with an intelligent adversary

- new class of software-based attacks

- new class of software-based attacks
- many problems to solve around microarchitectural attacks and especially transient execution attacks

- new class of software-based attacks
- many problems to solve around microarchitectural attacks and especially transient execution attacks
- dedicate more time into identifying problems and not solely in mitigating known problems

Transient Execution Attacks

Daniel Gruss

June 20, 2019

Graz University of Technology

References

Michael Backes et al. Acoustic Side-Channel Attacks on Printers. In: USENIX Security. 2010.

David Brumley et al. Remote timing attacks are practical. In: Computer Networks 48.5 (2005), pp. 701–716.

Daniel J. Bernstein. Cache-Timing Attacks on AES. 2004. URL: http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

Elad Carmon et al. Photonic Side Channel Attacks Against RSA. In: HOST'17. 2017.

Daniel Gruss et al. Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. In: DIMVA. 2016.


Daniel Gruss et al. Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches. In: USENIX Security Symposium. 2015.


J. Alex Halderman et al. Lest we remember: cold-boot attacks on encryption keys. In: Communications of the ACM (May 2009).


Michael Hutter et al. The temperature side channel and heating fault attacks. In: International Conference on Smart Card Research and Advanced Applications. Springer. 2013, pp. 219–235.


Paul Kocher et al. Differential power analysis. In: Annual International Cryptology Conference. Springer. 1999, pp. 388–397.


Paul Kocher et al. Spectre Attacks: Exploiting Speculative Execution. In: S&P. 2019.


Emilia Käsper et al. Faster and Timing-Attack Resistant AES-GCM. In: Cryptographic Hardware and Embedded Systems (CHES). 2009, pp. 1–17.


Vladimir Kiriansky et al. Speculative Buffer Overflows: Attacks and Defenses. In: arXiv:1807.03757 (2018).


Moritz Lipp et al. ARMageddon: Cache Attacks on Mobile Devices. In: USENIX Security Symposium. 2016.


Moritz Lipp et al. Nethammer: Inducing Rowhammer Faults through Network Requests. In: arXiv:1711.08002 (2017).


Moritz Lipp et al. Meltdown: Reading Kernel Memory from User Space. In: USENIX Security Symposium. 2018.


Stefan Mangard et al. Power analysis attacks: Revealing the secrets of smart cards. Vol. 31. Springer Science & Business Media, 2008.


Yossef Oren et al. The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications. In: CCS. 2015.


Josyula R Rao et al. EMpowering Side-Channel Attacks. In: IACR Cryptology ePrint Archive 2001 (2001), p. 37.


Alexander Schlösser et al. Simple Photonic Emission Analysis of AES. In: CHES'12. 2012.


Michael Schwarz et al. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In: DIMVA. 2017.


Michael Schwarz et al. Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features. In: AsiaCCS (2018).


Michael Schwarz et al. NetSpectre: Read Arbitrary Memory over Network. In: arXiv:1807.10535 (2018).


Andrei Tatar et al. Throwhammer: Rowhammer Attacks over the Network and Defenses. In: USENIX ATC. 2018.


Jo Van Bulck et al. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In: USENIX Security Symposium. 2018.


Ofir Weisse et al. Foreshadow-NG: Breaking the Virtual Memory Abstraction with Transient Out-of-Order Execution. In: Technical report (2018).