Brief Overview on Meltdown and Spectre

Daniel Gruss

January 25, 2018

Graz University of Technology

Whoami

- Daniel Gruss
- Post-Doc @ Graz University of Technology
- Twitter: `@lavados`
- Email: `daniel.gruss@iaik.tugraz.at`

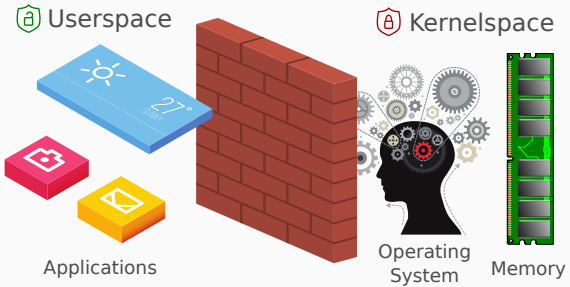
- security and privacy rely on secrets (unknown to attackers)
- secrets can leak through side channels

Software-based Side-Channel Attacks

- security and privacy rely on secrets (unknown to attackers)
- secrets can leak through side channels
- software-based $\rightarrow$ no physical access

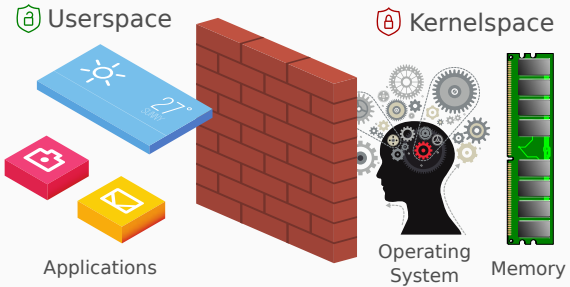
The Core of Meltdown/Spectre

- Kernel is isolated from user space



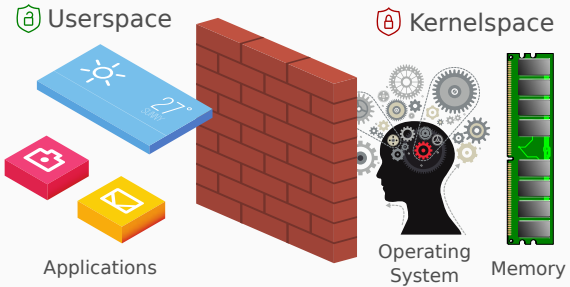
The Core of Meltdown/Spectre

- Kernel is isolated from user space
- This isolation is a combination of hardware and software



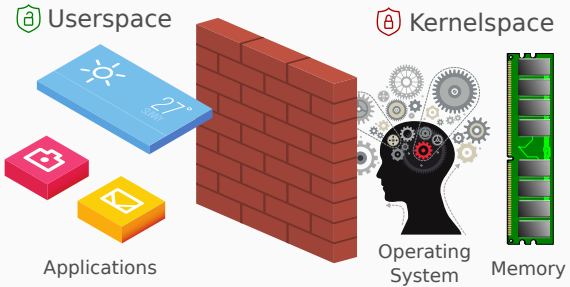
The Core of Meltdown/Spectre

- Kernel is isolated from user space
- This isolation is a combination of hardware and software
- User applications cannot access anything from the kernel

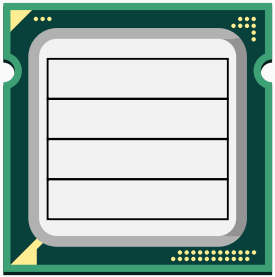


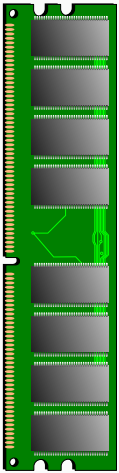
The Core of Meltdown/Spectre

- Kernel is isolated from user space
- This isolation is a combination of hardware and software
- User applications cannot access anything from the kernel
- There is only a well-defined interface → syscalls



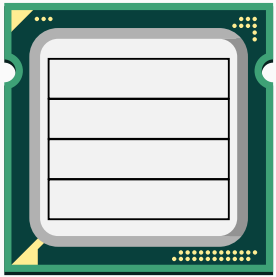
```
printf("%d", i);
printf("%d", i);
```

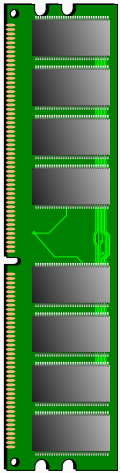




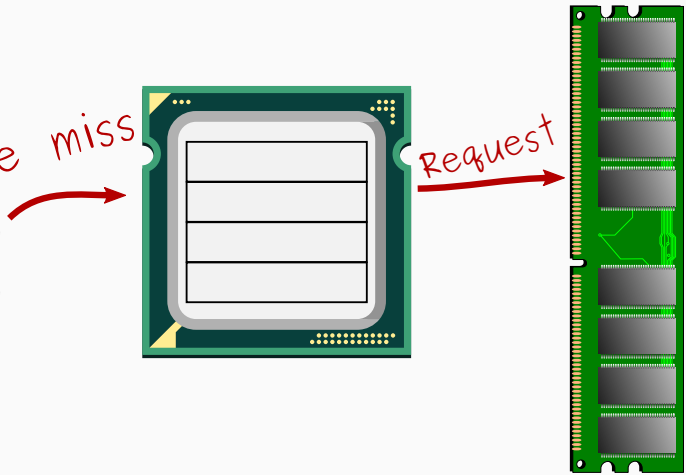
```
printf("%d", i);
printf("%d", i);
```

Cache miss

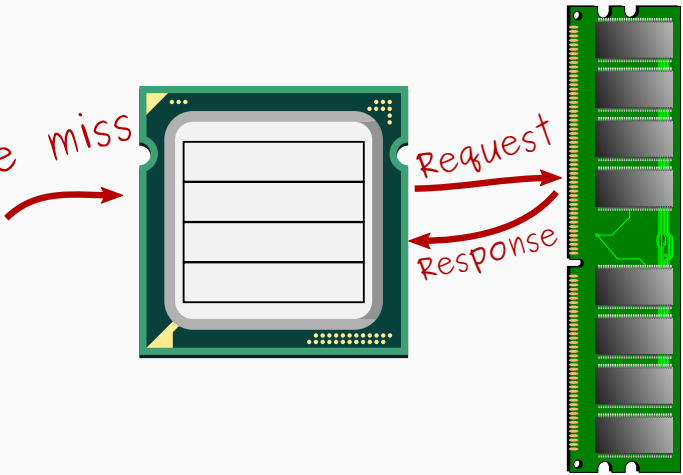




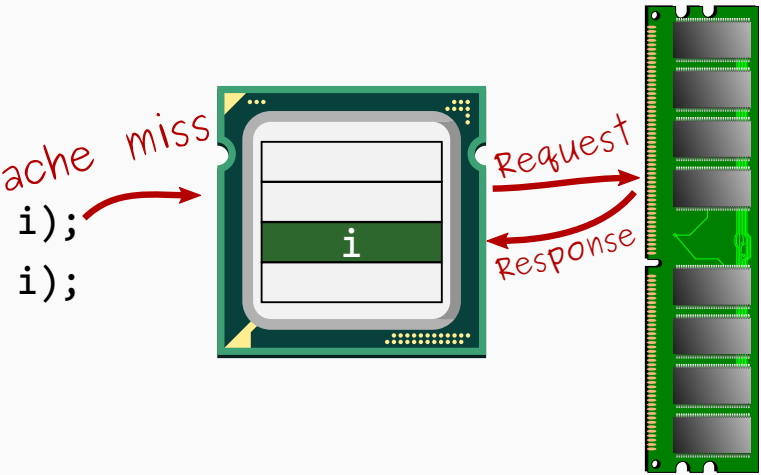
```
printf("%d", i);
printf("%d", i);
```

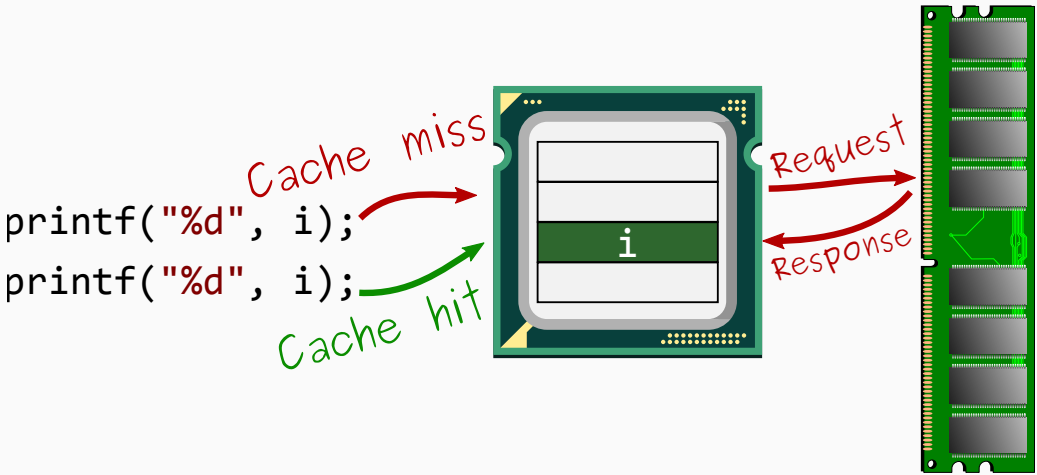


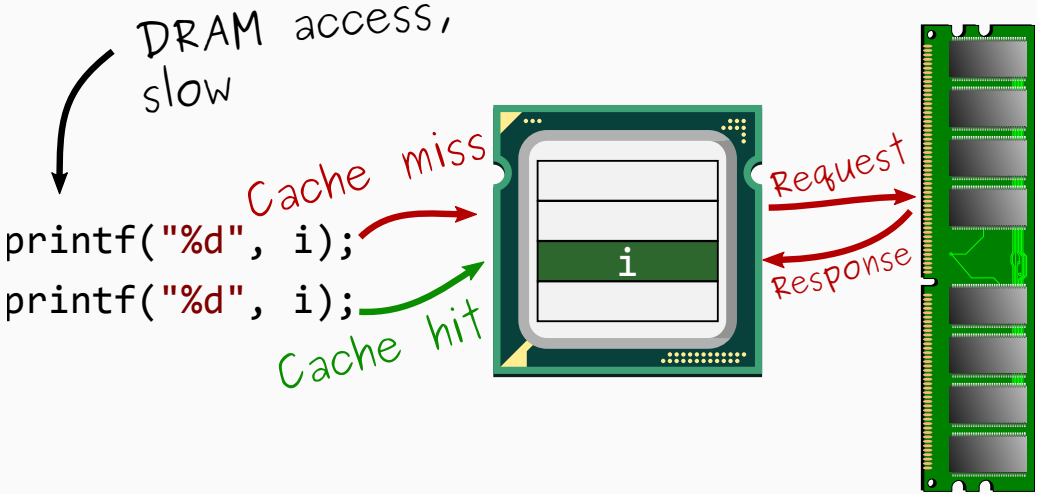
```
printf("%d", i);
printf("%d", i);
```

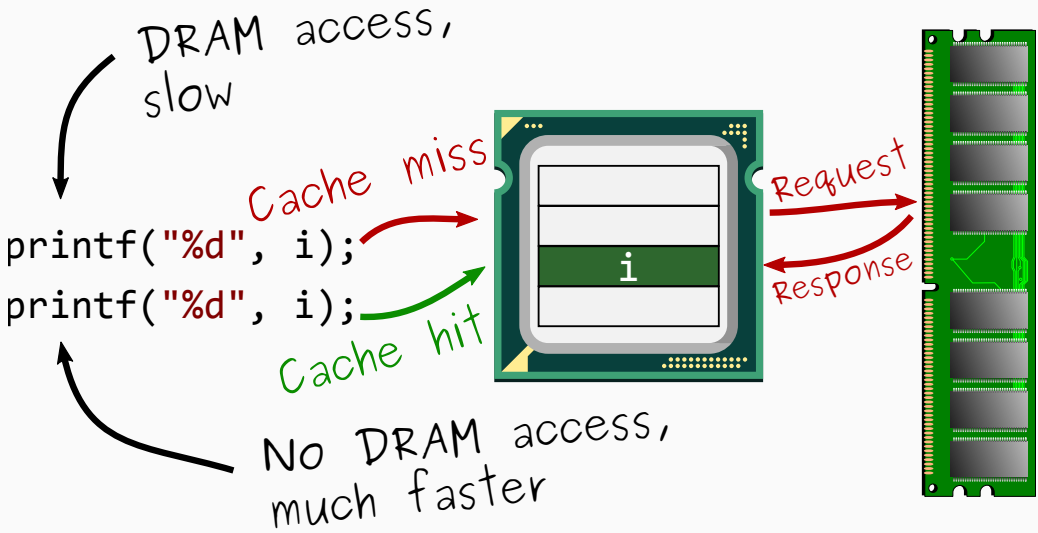


```
printf("%d", i);
printf("%d", i);
```




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

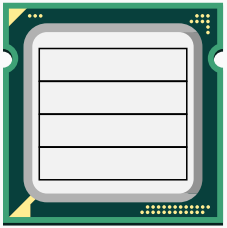

DRAM access,
slow
printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response
No DRAM access,
much faster

Shared Memory

ATTACKER

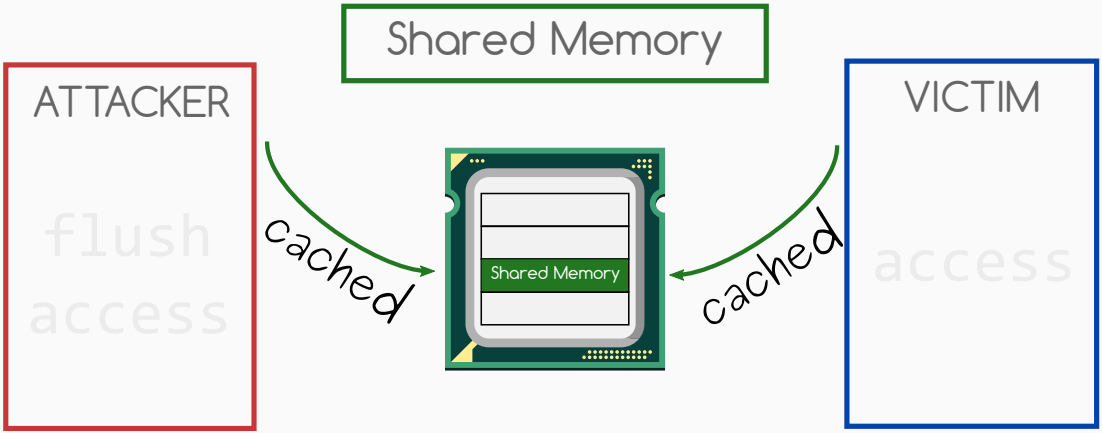
flush

access



VICTIM

access

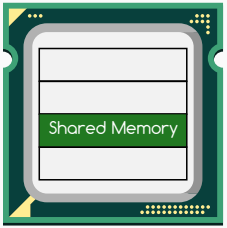

Shared Memory
ATTACKER
flush
access
cached
Shared Memory
cached
VICTIM
access

Shared Memory

ATTACKER

flush

access



VICTIM

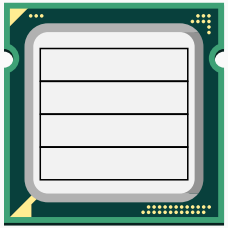
access

Shared Memory

ATTACKER

flush

access



VICTIM

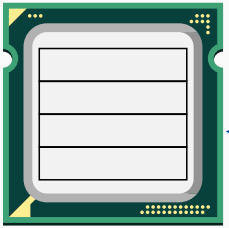
access

Shared Memory

ATTACKER

flush

access



VICTIM

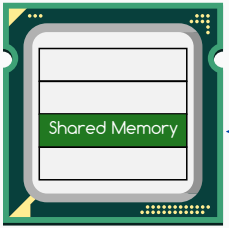
access

Shared Memory

ATTACKER

flush

access



VICTIM

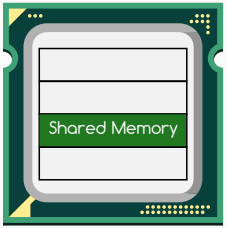
access

Shared Memory

ATTACKER

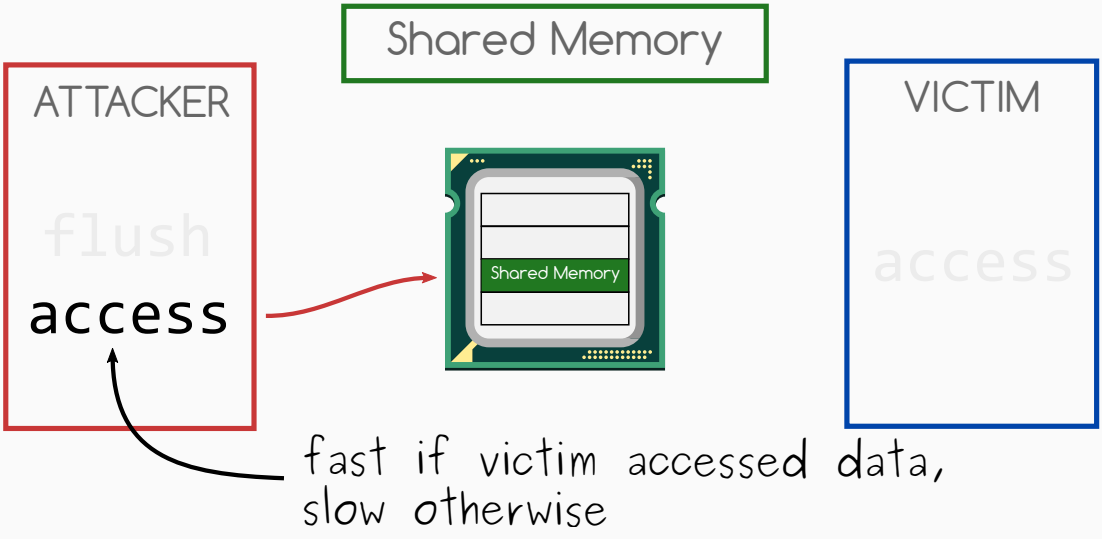
flush

access

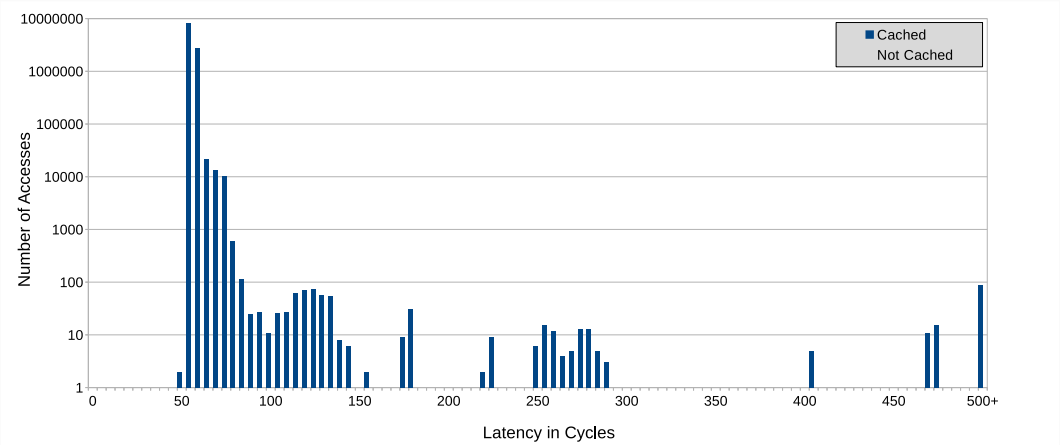


VICTIM

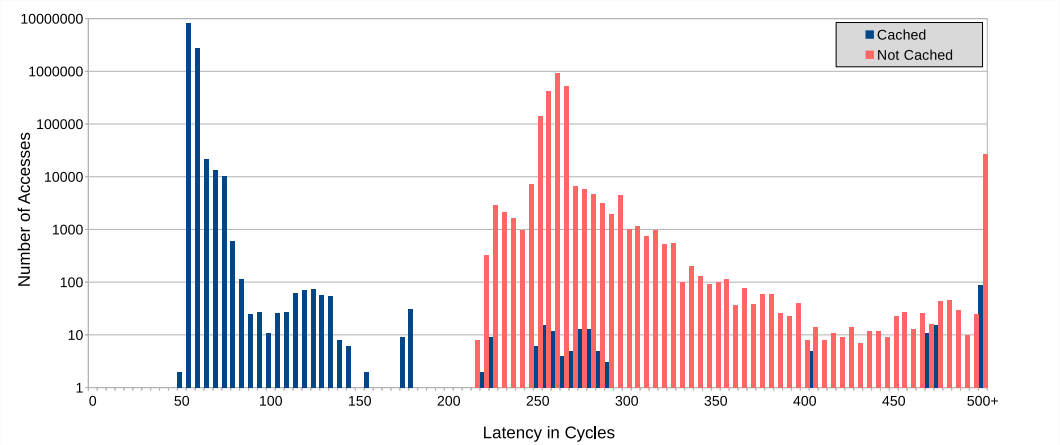
access

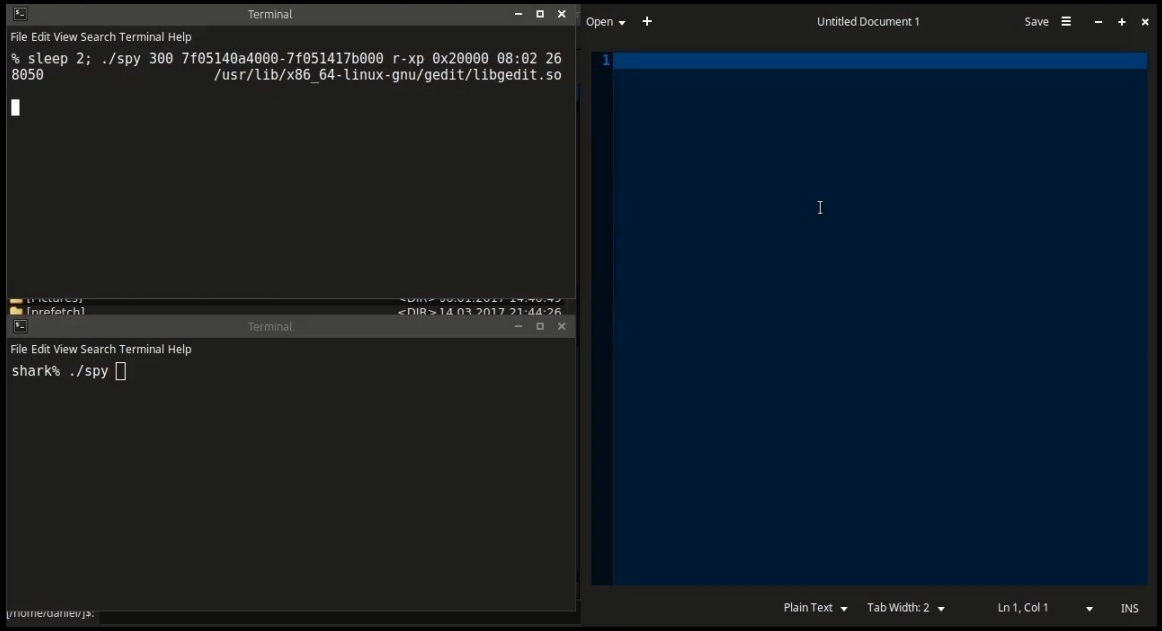

Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access
fast if victim accessed data,
slow otherwise

Memory Access Latency



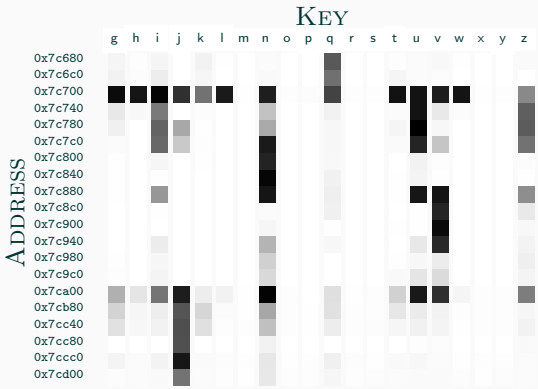
Memory Access Latency





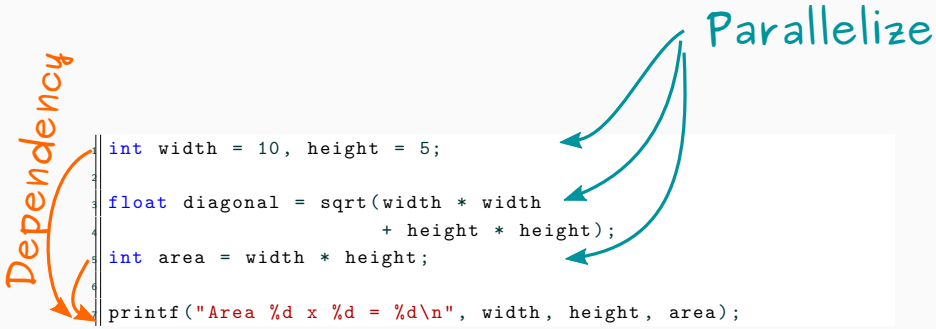
Cache Template Attack Demo

Cache Template



Out-of-order Execution

```
1 int width = 10, height = 5;
2
3 float diagonal = sqrt(width * width
4                       + height * height);
5 int area = width * height;
6
7 printf("Area %d x %d = %d\n", width, height, area);
```


Parallelize
Dependency
1 int width = 10, height = 5;
2
3 float diagonal = sqrt(width * width
4 + height * height);
5 int area = width * height;
6
7 printf("Area %d x %d = %d\n", width, height, area);

- Out-of-order instructions leave microarchitectural traces

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

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

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

- Maybe there is no permission check in transient instructions...

Building the Code

- Maybe there is no permission check in transient instructions...
- ...or it is only done when commiting them

- Maybe there is no permission check in transient instructions...
- ...or it is only done when commiting them
- Add another layer of indirection to test

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

- Maybe there is no permission check in transient instructions...
- ...or it is only done when commiting them
- Add another layer of indirection to test

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

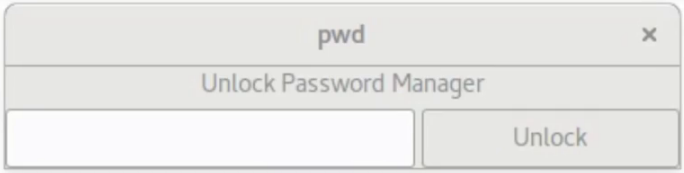
- Then check whether any part of `array` is cached

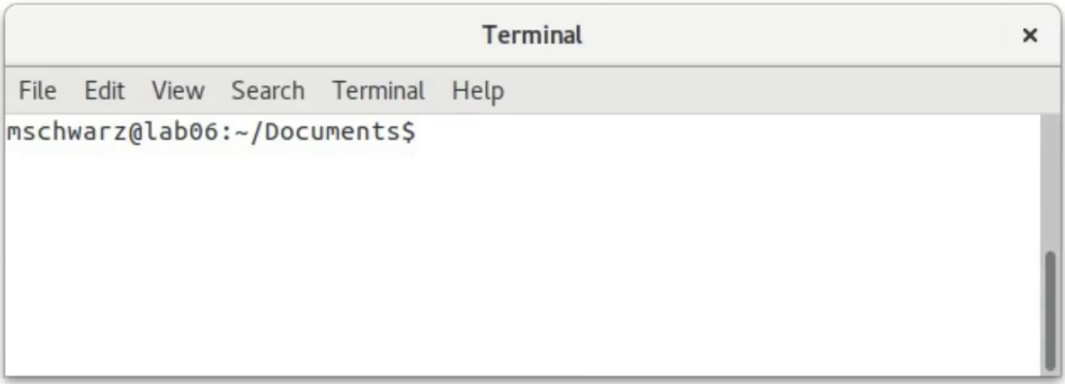
- Flush+Reload over all pages of the array

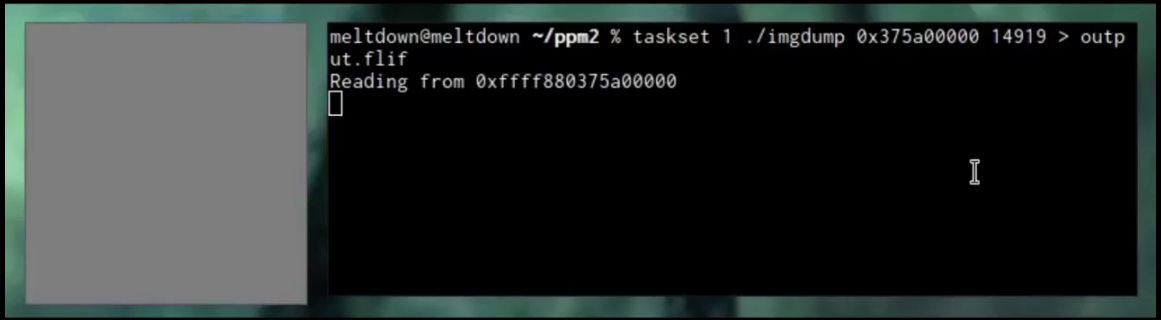
- Index of cache hit reveals data

- Flush+Reload over all pages of the array

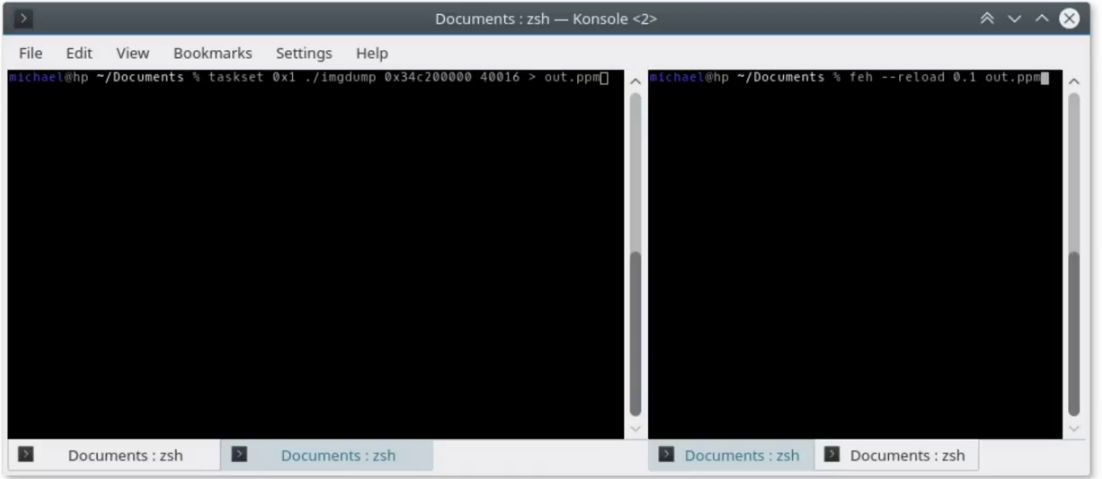
- Index of cache hit reveals data
- Permission check is in some cases not fast enough


pwd
Unlock Password Manager
Unlock

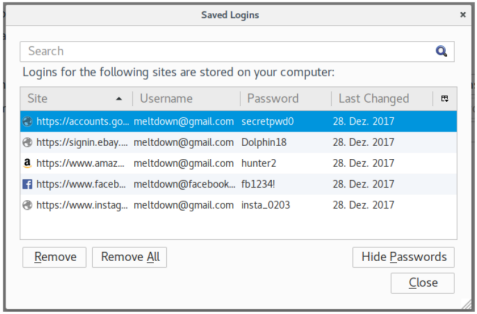

Terminal
File Edit View Search Terminal Help
mschwarz@lab06:~/Documents$



Leaking a picture like in CSI Cyber

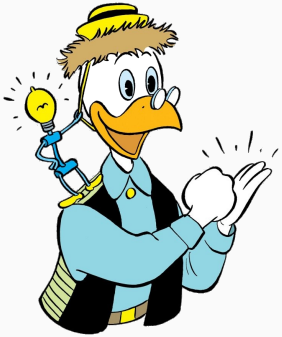

Documents : zsh — Konsole <2>
File Edit View Bookmarks Settings Help
michael@hp ~/Documents % taskset 0x1 ./imgdump 0x34c200000 40016 > out.ppm
michael@hp ~/Documents % feh --reload 0.1 out.ppm
Documents : zsh
Documents : zsh
Documents : zsh
Documents : zsh

Leaking Passwords from your Password Manager



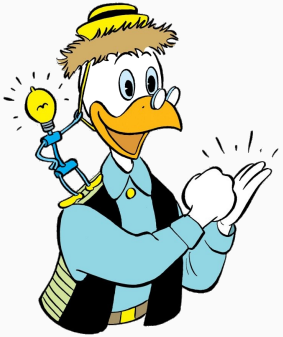

```
f94b7690: e5 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 e5 |................|
f94b76a0: e5 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 e5 |................|
f94b76b0: 70 52 b8 6b 96 7f XX XX  XX XX XX XX XX XX XX XX |pR.k............|
f94b76c0: 09 XX XX XX XX XX XX XX  XX XX XX XX XX XX XX XX |................|
f94b76d0: XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX XX |................|
f94b76e0: XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX 81 |................|
f94b76f0: 12 XX e0 81 19 XX e0 81  44 6f 6c 70 68 69 6e 31 |........Dolphin1|
f94b7700: 38 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 e5 |8...............|
f94b7710: 70 52 b8 6b 96 7f XX XX  XX XX XX XX XX XX XX XX |pR.k............|
f94b7720: XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX XX |................|
f94b7730: XX XX XX XX 4a XX XX XX  XX XX XX XX XX XX XX XX |....J...........|
f94b7740: XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX XX |................|
f94b7750: XX XX XX XX XX XX XX XX  XX e0 81 69 6e 73 74 |...........inst|
f94b7760: 61 5f 30 32 30 33 e5 e5  e5 e5 e5 e5 e5 e5 e5 e5 |a_0203..........|
f94b7770: 70 52 18 7d 28 7f XX XX  XX XX XX XX XX XX XX XX |pR.}(...........|
f94b7780: XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX XX |................|
f94b7790: XX XX XX XX 54 XX XX XX  XX XX XX XX XX XX XX XX |....T...........|
f94b77a0: XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX XX |................|
f94b77b0: XX XX XX XX XX XX XX XX  XX XX XX 73 65 63 72 |...........secr|
f94b77c0: 65 74 70 77 64 30 e5 e5  e5 e5 e5 e5 e5 e5 e5 e5 |etpwd0..........|
f94b77d0: 30 b4 18 7d 28 7f XX XX  XX XX XX XX XX XX XX XX |0..}(...........|
f94b77e0: XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX XX |................|
f94b77f0: XX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX XX |................|
f94b7800: e5 e5 e5 e5 e5 e5 e5 e5  e5 e5 e5 e5 e5 e5 e5 e5 |................|
f94b7810: 68 74 74 70 73 3a 2f 2f  61 64 64 6f 6e 73 2e 63 |https://addons.c|
f94b7820: 64 6e 2e 6d 6f 7a 69 6c  6c 61 2e 6e 65 74 2f 75 |dn.mozilla.net/u|
f94b7830: 73 65 72 2d 6d 65 64 69  61 2f 61 64 64 6f 6e 5f |ser-media/addon_|
f94b7840: 69 63 6f 6e 73 2f 33 35  34 2f 33 35 34 33 39 30 |icons/354/354390|
f94b7850: 2d 36 34 2e 70 6e 67 3f  6d 6f 64 69 66 69 65 64 |-64.png?modified|
f94b7860: 3d 31 34 35 32 32 34 34  38 31 35 XX XX XX XX XX |=1452244815.....|
```

How to stop a Meltdown?



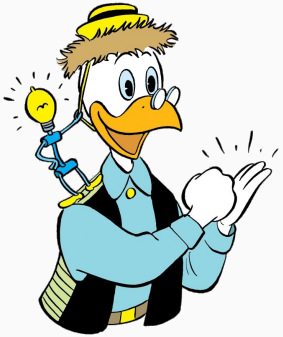
- Kernel addresses in user space are a problem

How to stop a Meltdown?



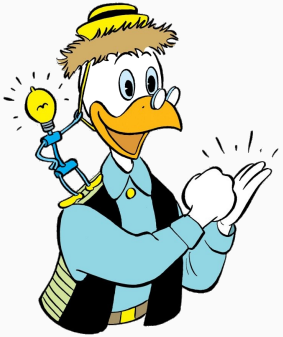
- Kernel addresses in user space are a problem
- Let's just unmap the kernel in user space

How to stop a Meltdown?



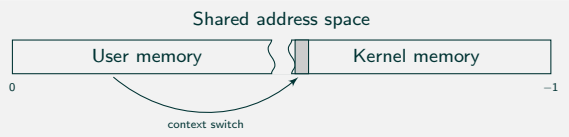
- Kernel addresses in user space are a problem
- Let's just unmap the kernel in user space
- Kernel addresses are then no longer present

How to stop a Meltdown?

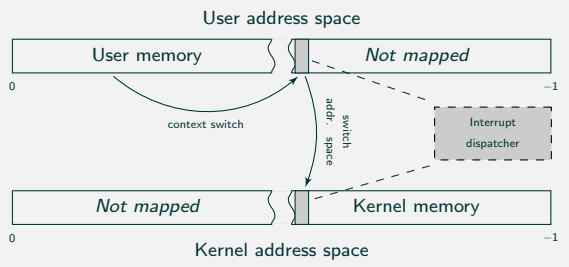


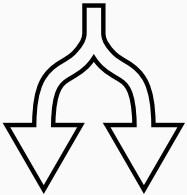
- Kernel addresses in user space are a problem
- Let's just unmap the kernel in user space
- Kernel addresses are then no longer present
- Memory which is not mapped cannot be accessed at all

Today's operating systems:

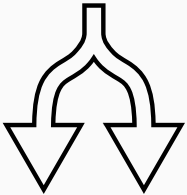


Stronger kernel isolation:

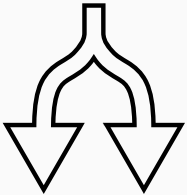




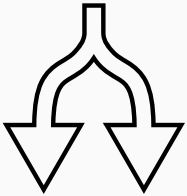
- We published KAISER in July 2017



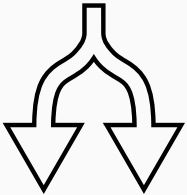
- We published KAISER in July 2017
- Intel and others improved and merged it into Linux as KPTI (Kernel Page Table Isolation)



- We published KAISER in July 2017
- Intel and others improved and merged it into Linux as KPTI (Kernel Page Table Isolation)
- Microsoft implemented similar concept in Windows 10



- We published KAISER in July 2017
- Intel and others improved and merged it into Linux as KPTI (Kernel Page Table Isolation)
- Microsoft implemented similar concept in Windows 10
- Apple implemented it in macOS 10.13.2 and called it "Double Map"



- We published KAISER in July 2017
- Intel and others improved and merged it into Linux as KPTI (Kernel Page Table Isolation)
- Microsoft implemented similar concept in Windows 10
- Apple implemented it in macOS 10.13.2 and called it "Double Map"
- All share the same idea: switching address spaces on context switch

- Depends on how often you need to switch between kernel and user space

Performance

- Depends on how often you need to switch between kernel and user space
- Can be slow, 40% or more on old hardware

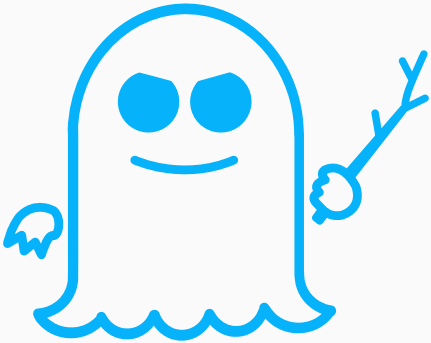
Performance

- Depends on how often you need to switch between kernel and user space
- Can be slow, 40% or more on old hardware
- But modern CPUs have additional features

Performance

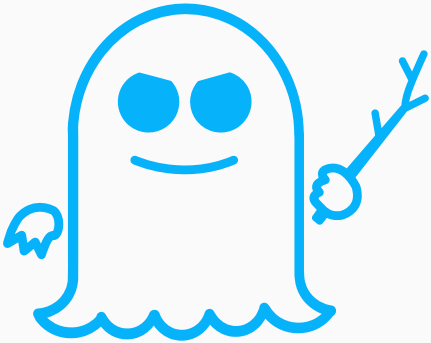
- Depends on how often you need to switch between kernel and user space
- Can be slow, 40% or more on old hardware
- But modern CPUs have additional features
- $\Rightarrow$ Performance overhead on average below 2%

MELTDOWN



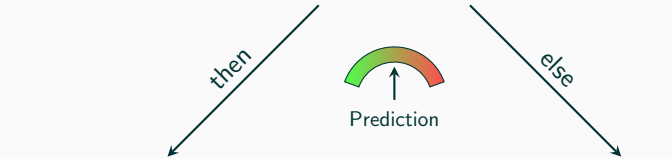
SPECTRE

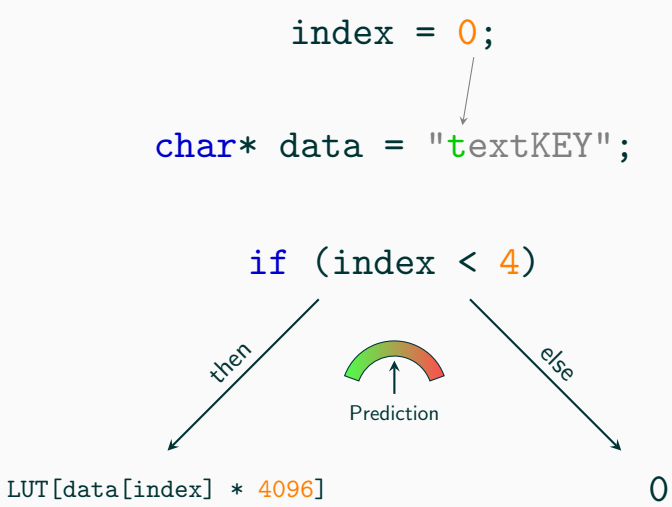
MELTDOWN



SPECTRE

```
index = 0;

char* data = "textKEY";

if (index < 4)
```

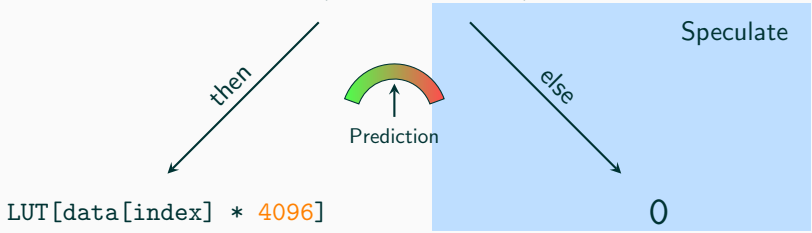



index = 0;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

```
index = 0;
```

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

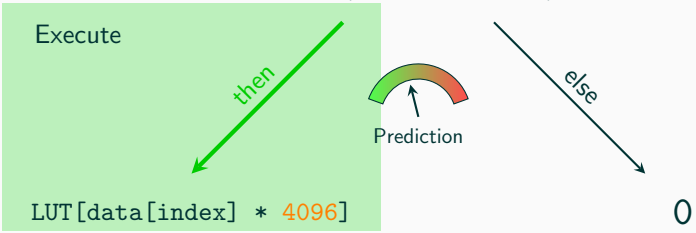
```
if (index < 4)
```



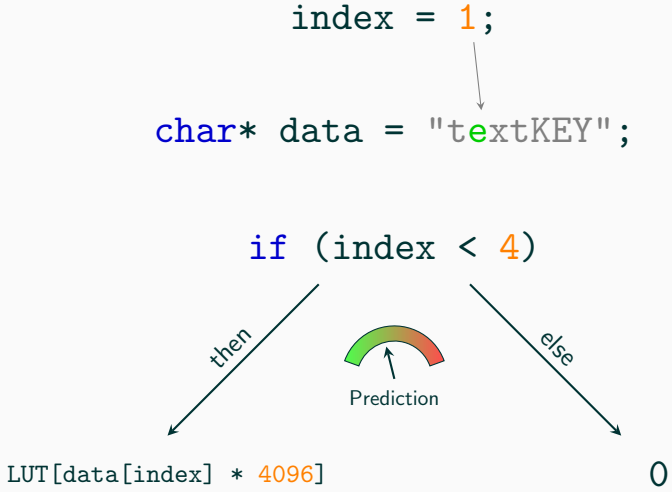
```
index = 0;
```

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

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



```
index = 1;

char* data = "textKEY";

if (index < 4)
```

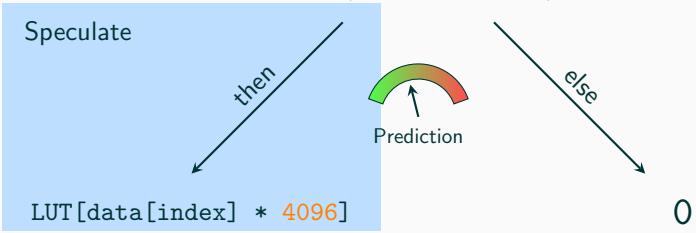

index = 1;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

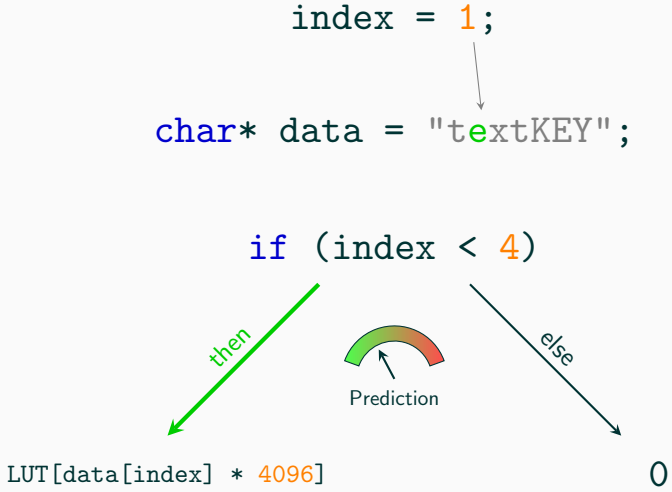
Spectre (variant 1)

```
index = 1;
```

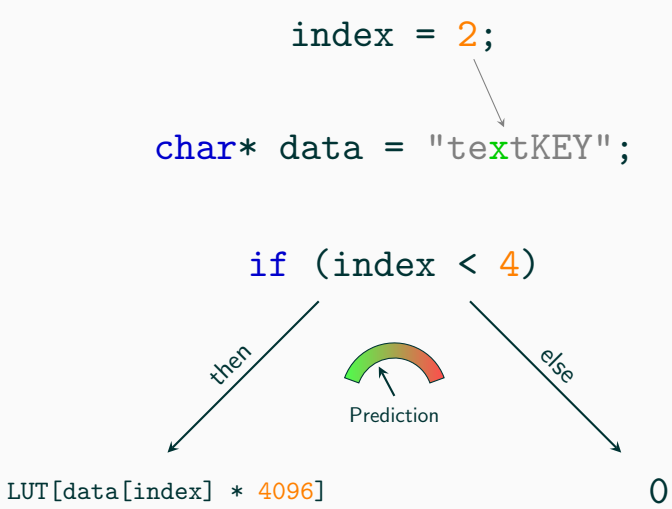
```
char* data = "textKEY";
```

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




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

```
index = 2;

char* data = "textKEY";

if (index < 4)
```

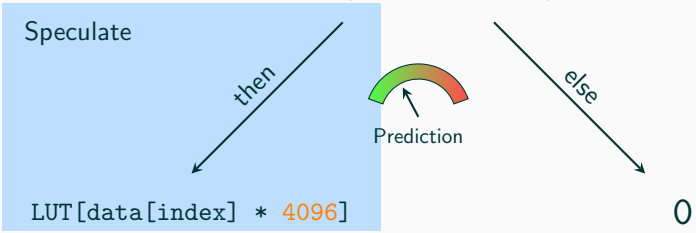

index = 2;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

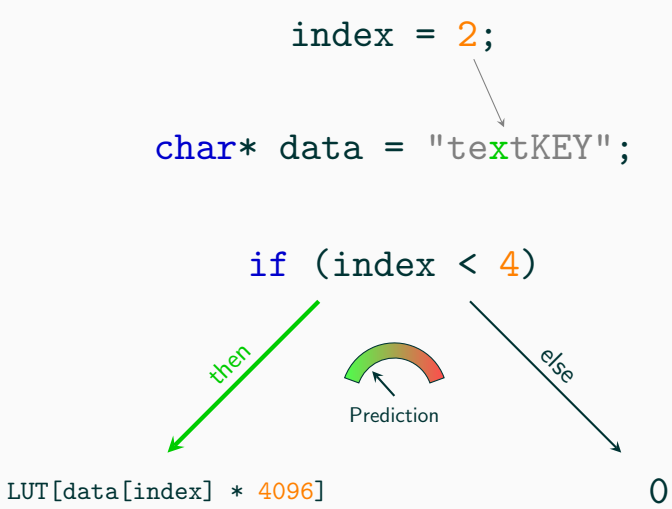
Spectre (variant 1)

```
index = 2;
```

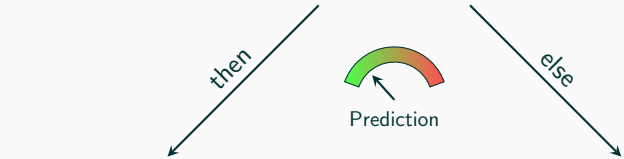
```
char* data = "textKEY";
```

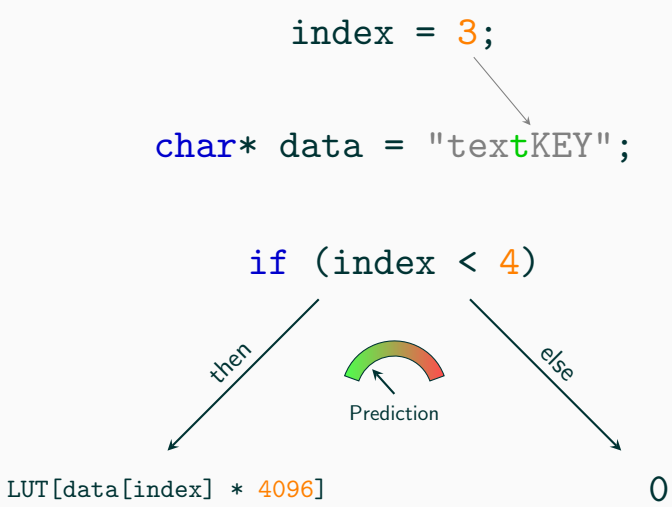
```
if (index < 4)
```




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

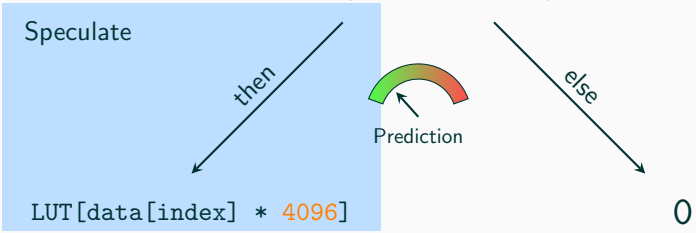
```
index = 3;

char* data = "textKEY";

if (index < 4)
```

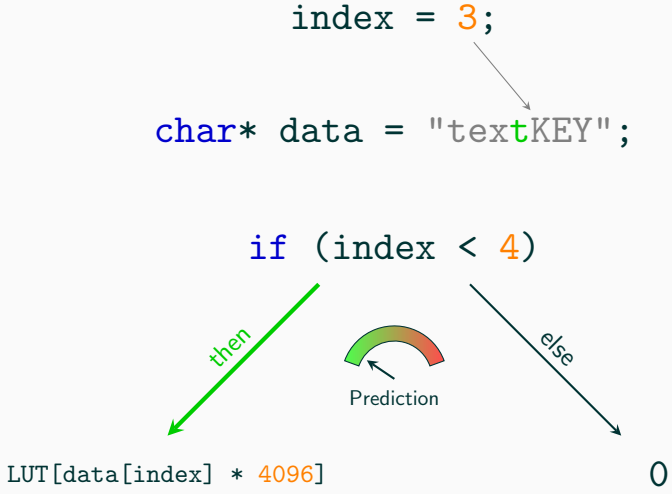



index = 3;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

Spectre (variant 1)

```
index = 3;
char* data = "textKEY";
if (index < 4)
```




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

```
index = 4;

char* data = "textKEY";

if (index < 4)
```

Spectre (variant 1)

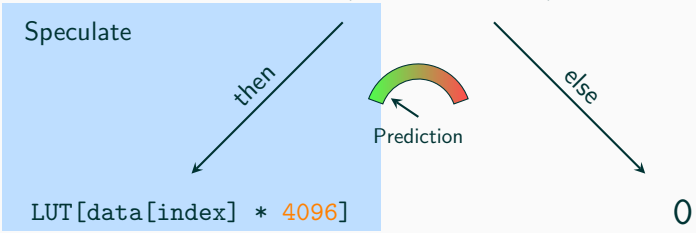
```
index = 4;

char* data = "textKEY";

if (index < 4)
```

`LUT[data[index] * 4096]`

0

Spectre (variant 1)

```
index = 4;

char* data = "textKEY";

if (index < 4)
```

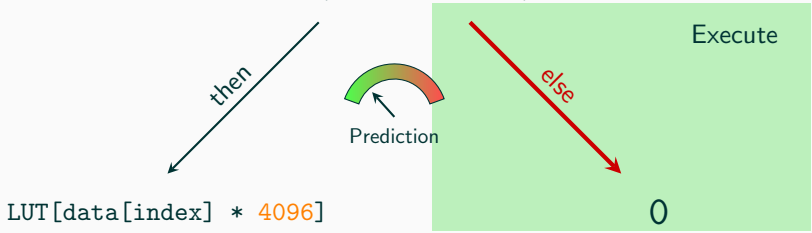


Spectre (variant 1)

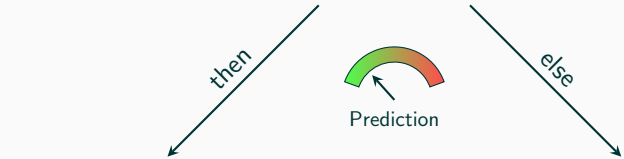
```
index = 4;
```

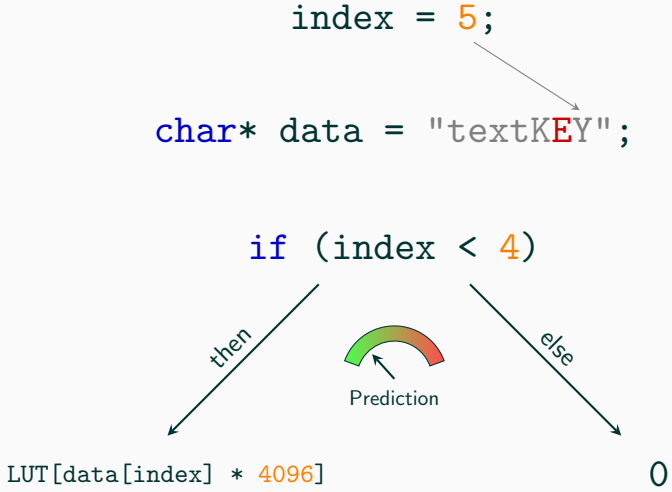
```
char* data = "textKEY";
```

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



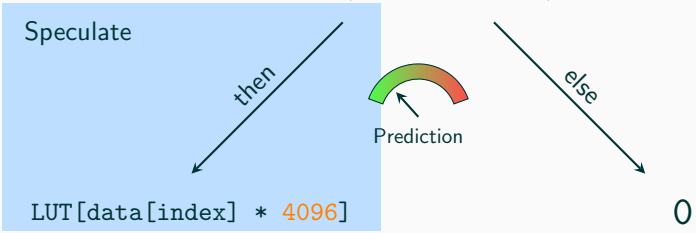
```
index = 5;

char* data = "textKEY";

if (index < 4)
```




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

Spectre (variant 1)

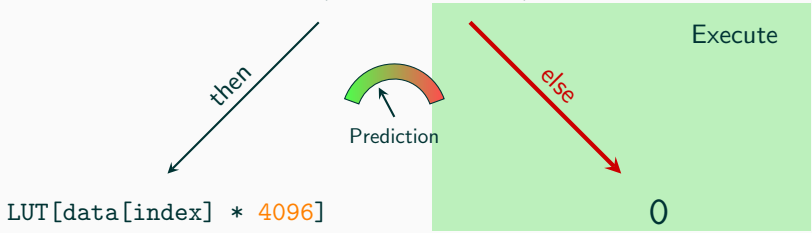
```
index = 5;

char* data = "textKEY";

if (index < 4)
```



```
index = 5;
```

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

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



```
index = 6;

char* data = "textKEY";

if (index < 4)
```

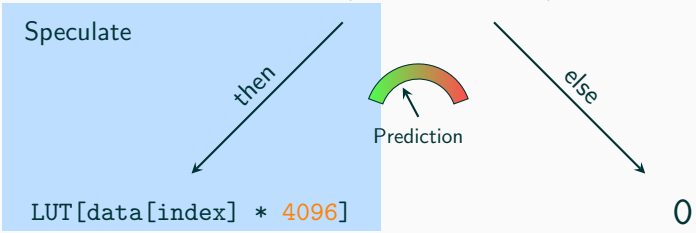
```
index = 6;

char* data = "textKEY";

if (index < 4)
```


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

0

Spectre (variant 1)

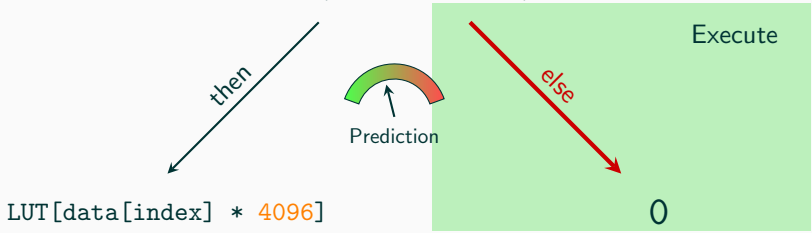
```
index = 6;

char* data = "textKEY";

if (index < 4)
```



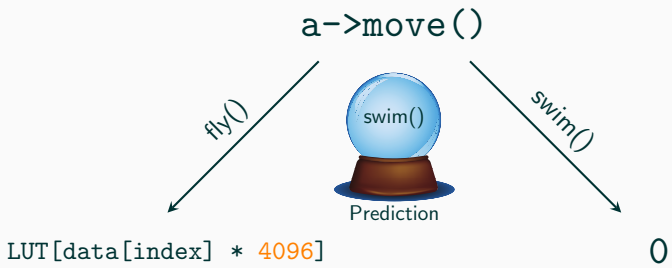
```
index = 6;
```

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

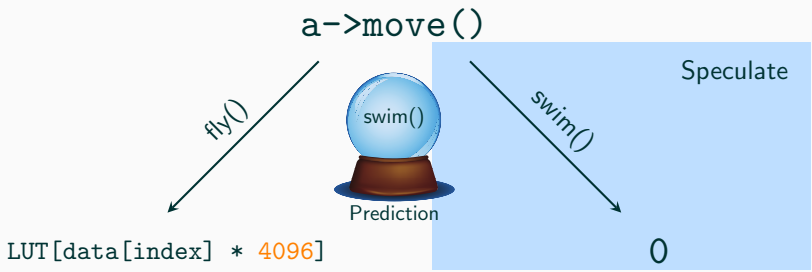
```
if (index < 4)
```



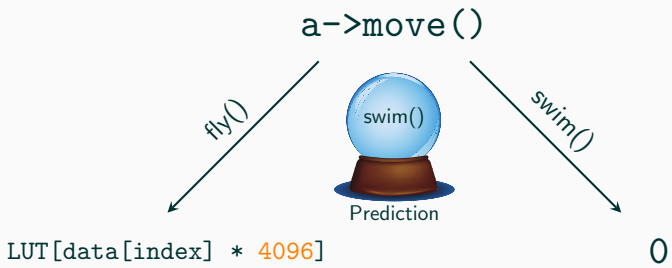
```
Animal* a = bird;
```



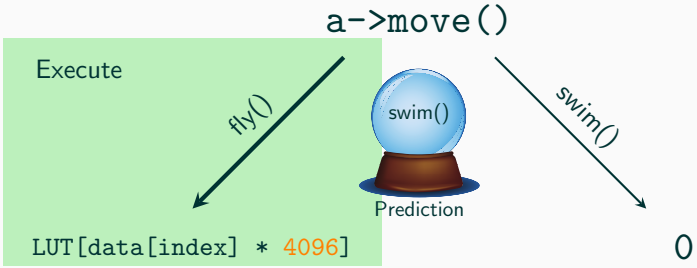
```
Animal* a = bird;
```



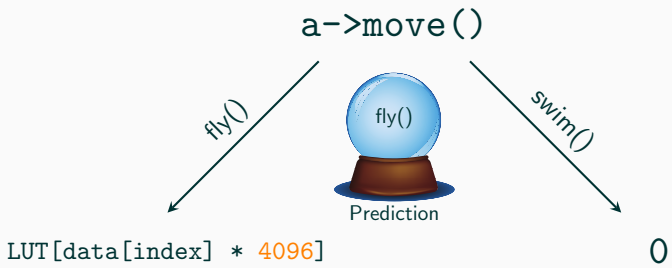
```
Animal* a = bird;
```



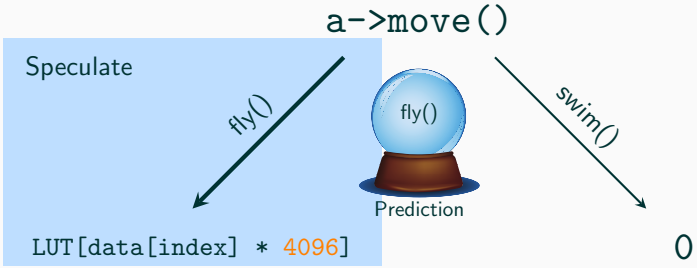
```
Animal* a = bird;
```



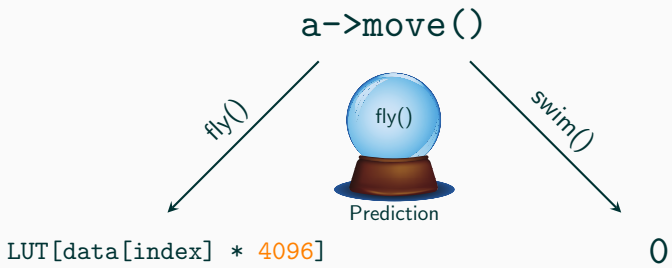
```
Animal* a = bird;
```



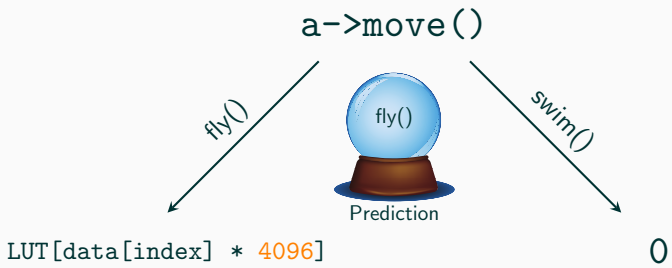
```
Animal* a = bird;
```



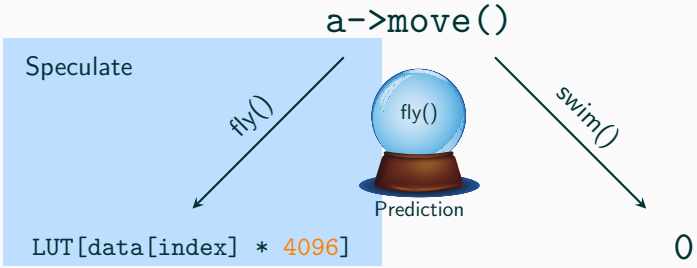
```
Animal* a = bird;
```



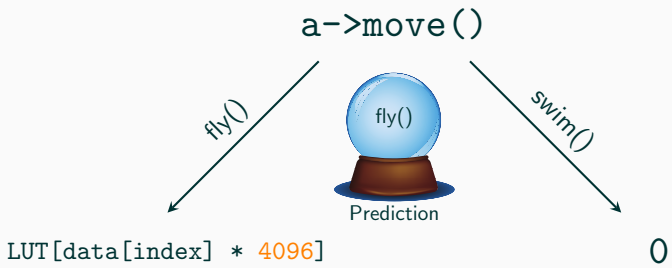
```
Animal* a = fish;
```



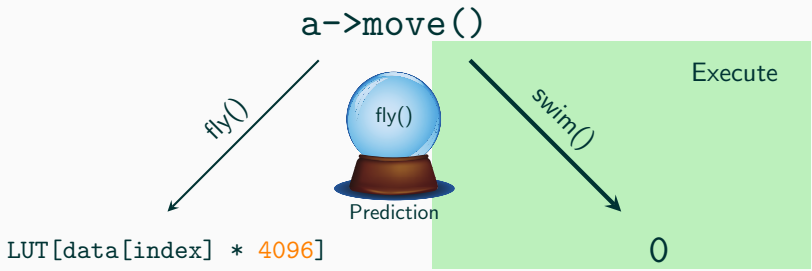
```
Animal* a = fish;
```



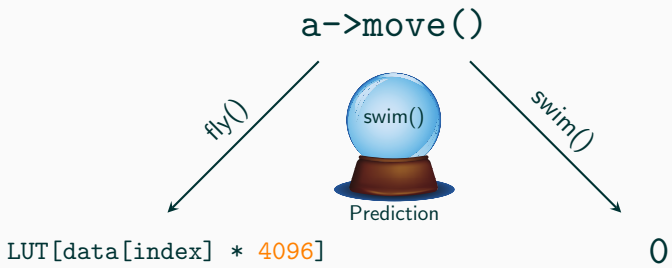
```
Animal* a = fish;
```



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



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



- Read own memory (e.g., sandbox escape)

- Read own memory (e.g., sandbox escape)
- “Convince” other programs to reveal their secrets

- Read own memory (e.g., sandbox escape)
- “Convince” other programs to reveal their secrets
- Again, a cache attack (Flush+Reload) is used to read the secret

- Read own memory (e.g., sandbox escape)
- “Convince” other programs to reveal their secrets
- Again, a cache attack (Flush+Reload) is used to read the secret
- Much harder to fix, KAISER does not help

- Read own memory (e.g., sandbox escape)
- “Convince” other programs to reveal their secrets
- Again, a cache attack (Flush+Reload) is used to read the secret
- Much harder to fix, KAISER does not help
- Ongoing effort to patch via microcode update and compiler extensions


GHOSTBUSTERS

- LFENCE

- LFENCE

→ speculation barrier to insert after every bounds check

- LFENCE

→ speculation barrier to insert after every bounds check

- implemented as a compiler extension

- Indirect Branch Restricted Speculation (IBRS):

Spectre Variant 2 Mitigations (Microcode/MSRs)

- Indirect Branch Restricted Speculation (IBRS):
  - do not speculate based on anything before entering or outside IBRS mode

Spectre Variant 2 Mitigations (Microcode/MSRs)

- Indirect Branch Restricted Speculation (IBRS):
  - do not speculate based on anything before entering or outside IBRS mode
- Single Thread Indirect Branch Predictors (STIBP)

Spectre Variant 2 Mitigations (Microcode/MSRs)

- Indirect Branch Restricted Speculation (IBRS):
  - do not speculate based on anything before entering or outside IBRS mode
- Single Thread Indirect Branch Predictors (STIBP)
  - do not speculate based on anything the other hyperthread does

Spectre Variant 2 Mitigations (Microcode/MSRs)

- Indirect Branch Restricted Speculation (IBRS):
  - do not speculate based on anything before entering or outside IBRS mode
- Single Thread Indirect Branch Predictors (STIBP)
  - do not speculate based on anything the other hyperthread does
- Indirect Branch Predictor Barrier (IBPB):

Spectre Variant 2 Mitigations (Microcode/MSRs)

- Indirect Branch Restricted Speculation (IBRS):
  - do not speculate based on anything before entering or outside IBRS mode
- Single Thread Indirect Branch Predictors (STIBP)
  - do not speculate based on anything the other hyperthread does
- Indirect Branch Predictor Barrier (IBPB):
  - flush branch-target buffer

retpoline

retpoline


```
1       push <call_target>
2       call 1f
3     2:                    ; speculation will continue here
4       lfence              ; speculation barrier
5       jmp 2b              ; endless loop
6     1:
7       lea 8(%rsp), %rsp   ; restore stack pointer
8       ret                 ; the actual call to <call_target>
```

→ always predict to enter an endless loop

retpoline


```
1       push <call_target>
2       call 1f
3     2:                    ; speculation will continue here
4       lfence              ; speculation barrier
5       jmp 2b              ; endless loop
6     1:
7       lea 8(%rsp), %rsp   ; restore stack pointer
8       ret                 ; the actual call to <call_target>
```

→ always predict to enter an endless loop

- instead of the correct (or wrong) target function

retpoline


```
1       push <call_target>
2       call 1f
3     2:                    ; speculation will continue here
4       lfence              ; speculation barrier
5       jmp 2b              ; endless loop
6     1:
7       lea 8(%rsp), %rsp   ; restore stack pointer
8       ret                 ; the actual call to <call_target>
```

→ always predict to enter an endless loop

- instead of the correct (or wrong) target function

What do we learn from it?

We have ignored software side-channels for many many years:

What do we learn from it?

We have ignored software side-channels for many many years:

- attacks on crypto

What do we learn from it?

We have ignored software side-channels for many many years:

- attacks on crypto $\rightarrow$ "software should be fixed"

What do we learn from it?

We have ignored software side-channels for many many years:

- attacks on crypto $\rightarrow$ "software should be fixed"
- attacks on ASLR

What do we learn from it?

We have ignored software side-channels for many many years:

- attacks on crypto $\rightarrow$ "software should be fixed"
- attacks on ASLR $\rightarrow$ "ASLR is broken anyway"

What do we learn from it?

We have ignored software side-channels for many many years:

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

What do we learn from it?

We have ignored software side-channels 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 software side-channels for many many years:

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

$\rightarrow$ for years we solely optimized for performance

When you read the Intel manuals...

After learning about a side channel you realize:

When you read the Intel manuals...

After learning about a side channel you realize:

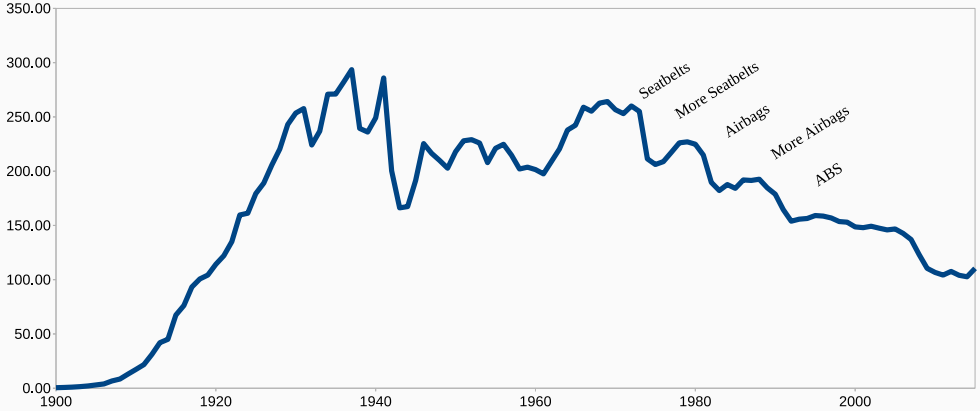
- the side channels were documented in the Intel manual

When you read the Intel manuals...

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

What do we learn from it?

A unique chance to

- rethink processor design
- grow up, like other fields (car industry, construction industry)
- find good trade-offs between security and performance

Brief Overview on Meltdown and Spectre

Daniel Gruss

January 25, 2018

Graz University of Technology