Software-based Microarchitectural Attacks

Daniel Gruss

January 23, 2019

Graz University of Technology

Frame Rate of Slide Deck: 11 fps

1337 4242

FOOD CACHE

Revolutionary concept!

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

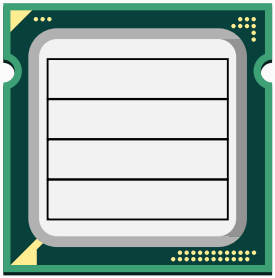
Can store **ALL** kinds of food.

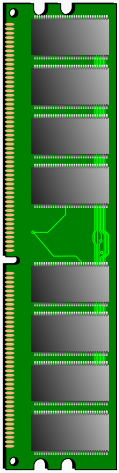
ONLY TODAY INSTEAD OF ~~$1,300~~

$1,299

ORDER VIA PHONE: +555 12345

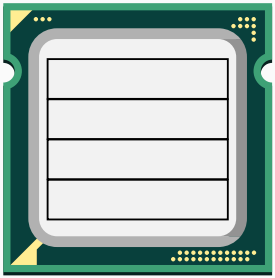
```
printf("%d", i);
printf("%d", i);
```

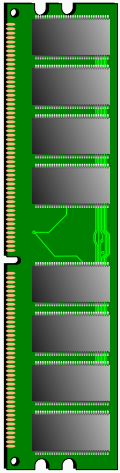




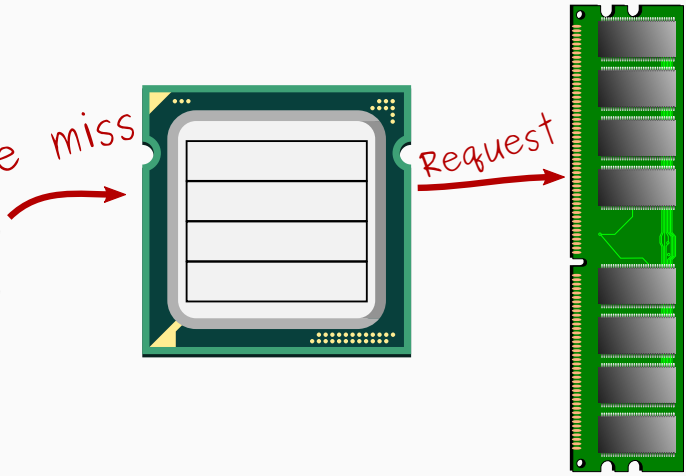
```
printf("%d", i);
printf("%d", i);
```

Cache miss

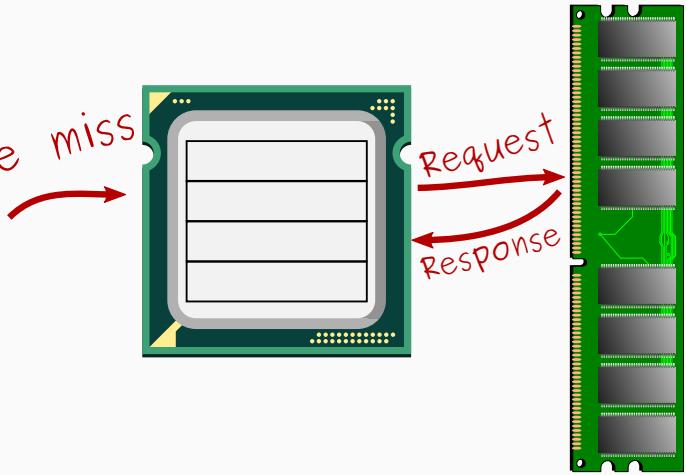




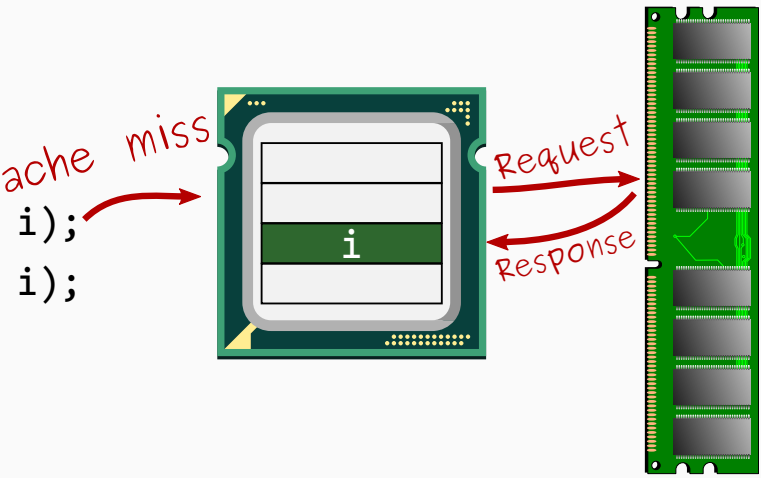
```
printf("%d", i);
printf("%d", i);
```

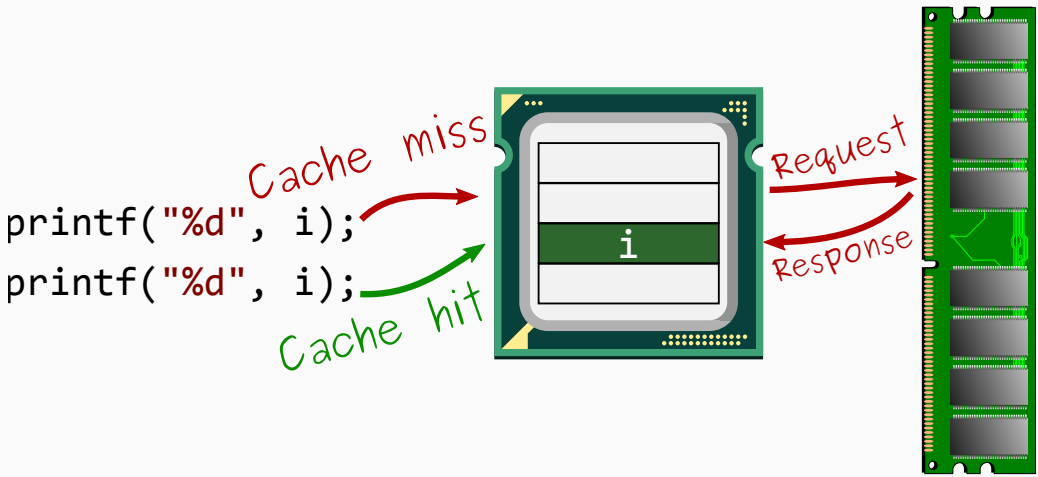


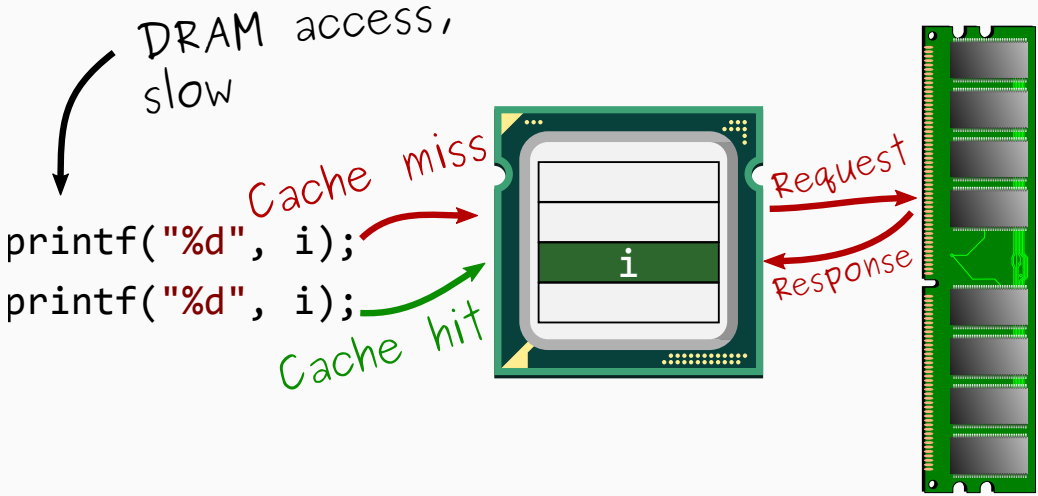
```
printf("%d", i);
printf("%d", i);
```

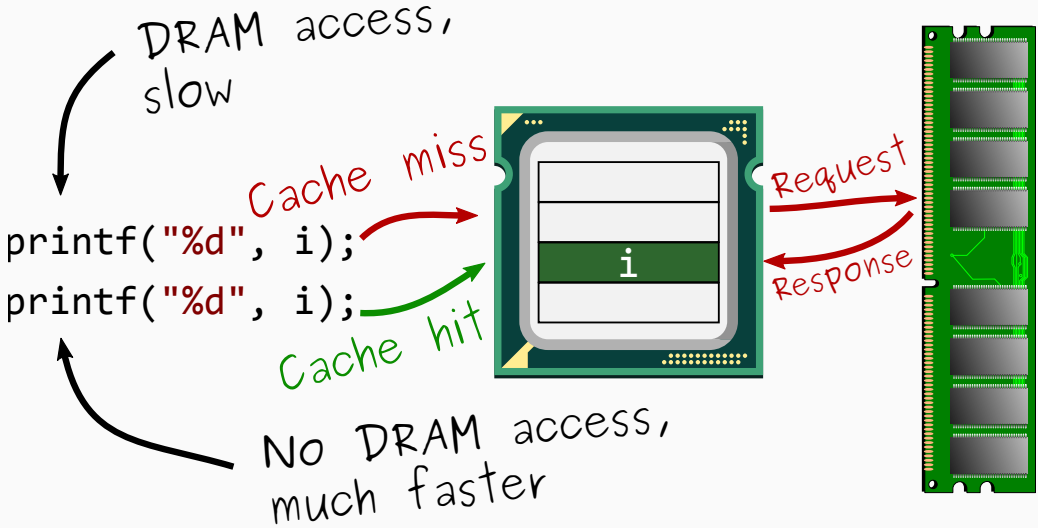


```
printf("%d", i);
printf("%d", i);
```




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

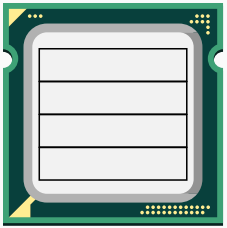

DRAM access, slow
printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response
No DRAM access, much faster

Shared Memory

ATTACKER

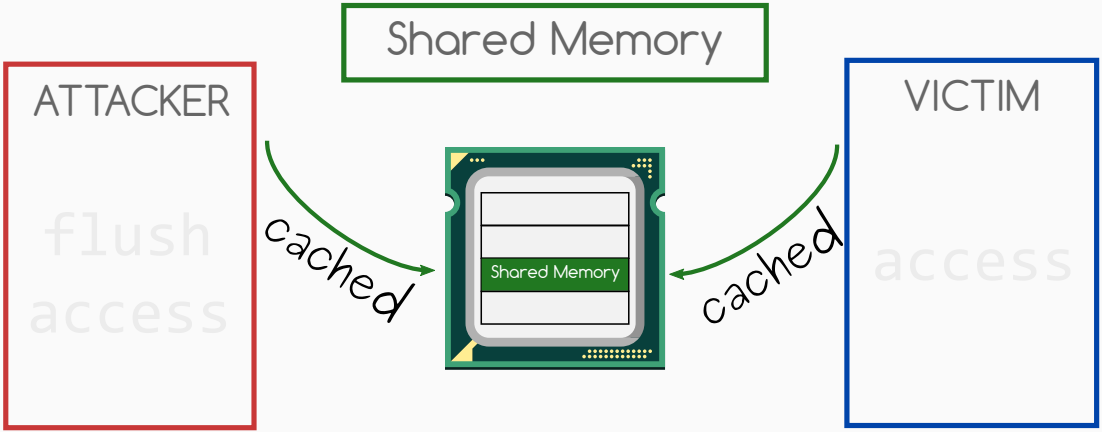
flush

access



VICTIM

access

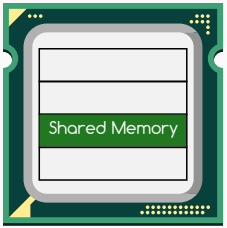

Shared Memory
ATTACKER
flush
access
cached
Shared Memory
cached
VICTIM
access

Shared Memory

ATTACKER

flush

access



VICTIM

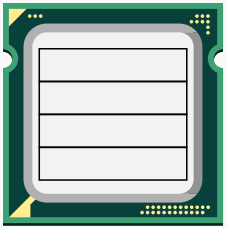
access

Shared Memory

ATTACKER

flush

access



VICTIM

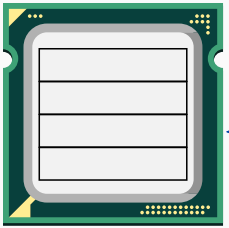
access

Shared Memory

ATTACKER

flush

access



VICTIM

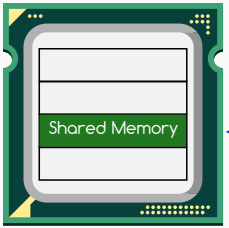
access

Shared Memory

ATTACKER

flush

access



VICTIM

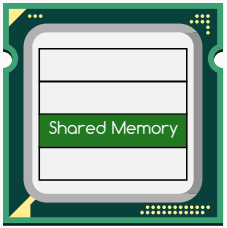
access

Shared Memory

ATTACKER

flush

access



VICTIM

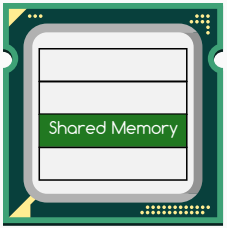
access

Shared Memory

ATTACKER

flush

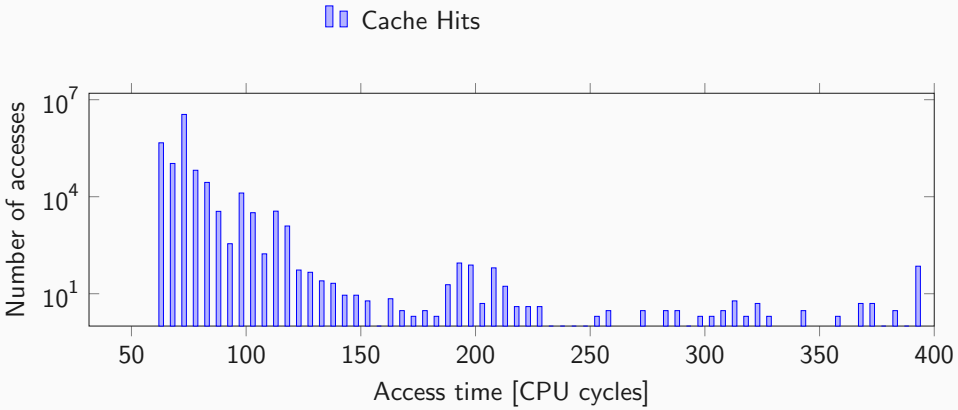
access

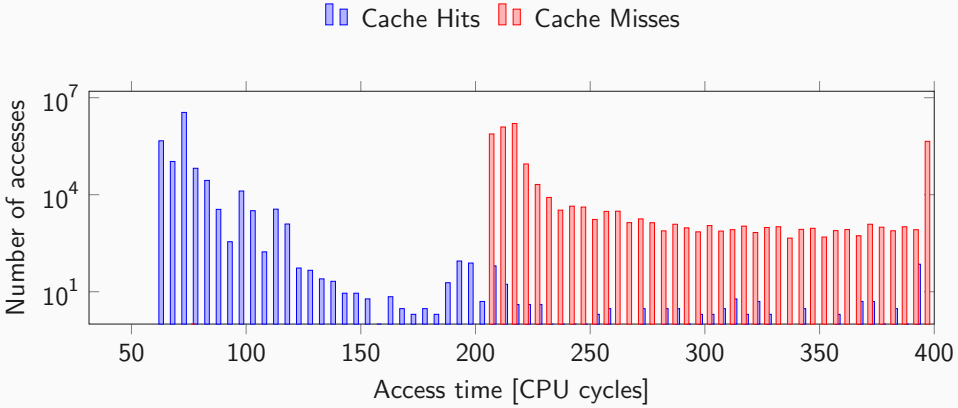


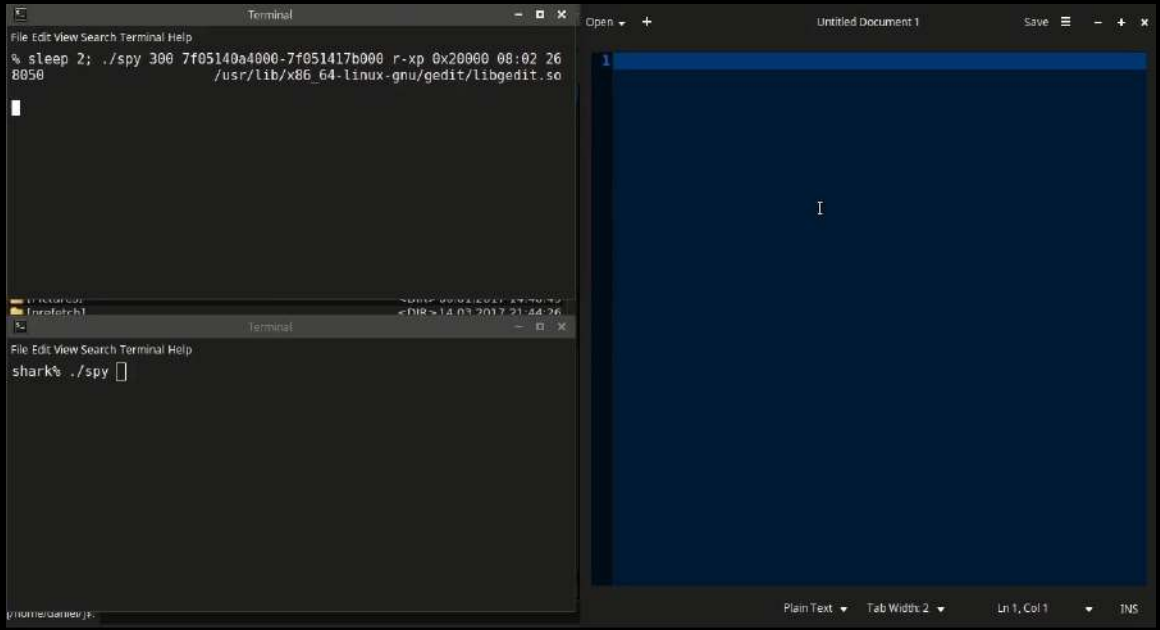
VICTIM

access

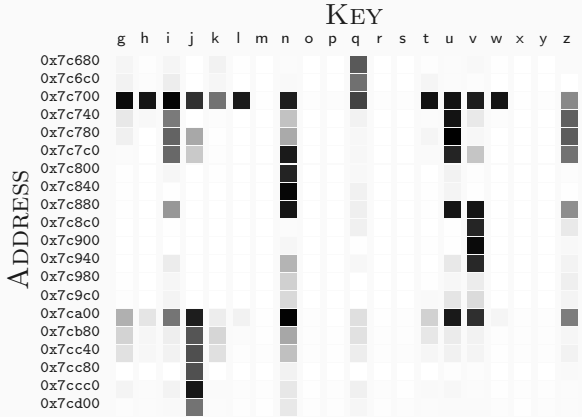
fast if victim accessed data,
slow otherwise

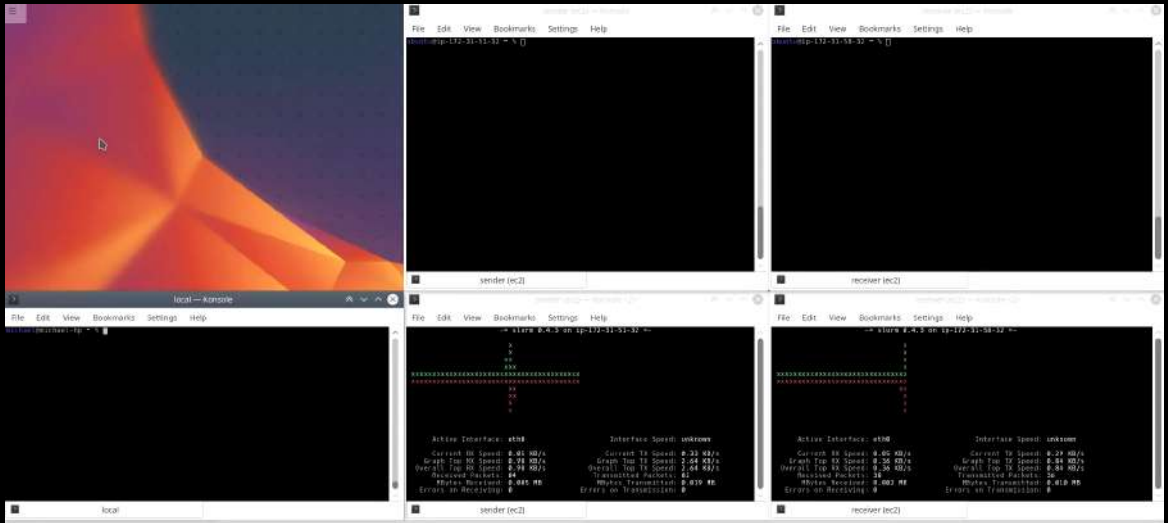

Cache Hits
Number of accesses
10^7
10^4
10^1
50
100
150
200
250
300
350
400
Access time [CPU cycles]


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



Hello from the other side (DEMO): Video streaming over cache covert channel

Line 17, Column 1 INSERT Soft Tabs: 4(8) UTF-8 Normal

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

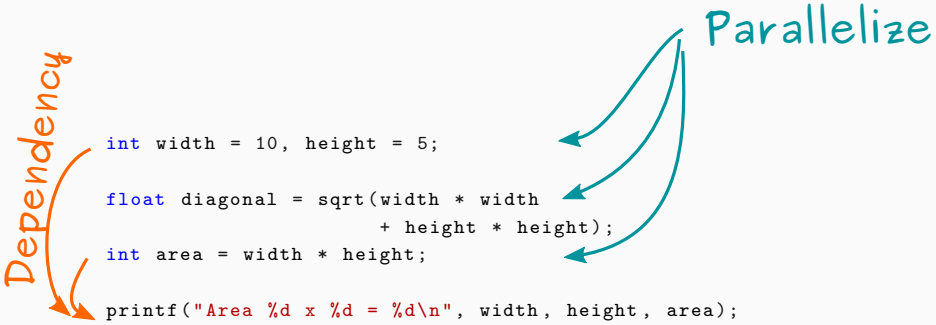
4. Fry vegetables until golden and softened

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);

```
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

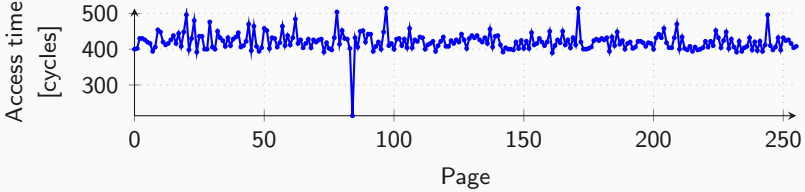
Flush+Reload over all pages of the array

This also works on AMD and ARM!

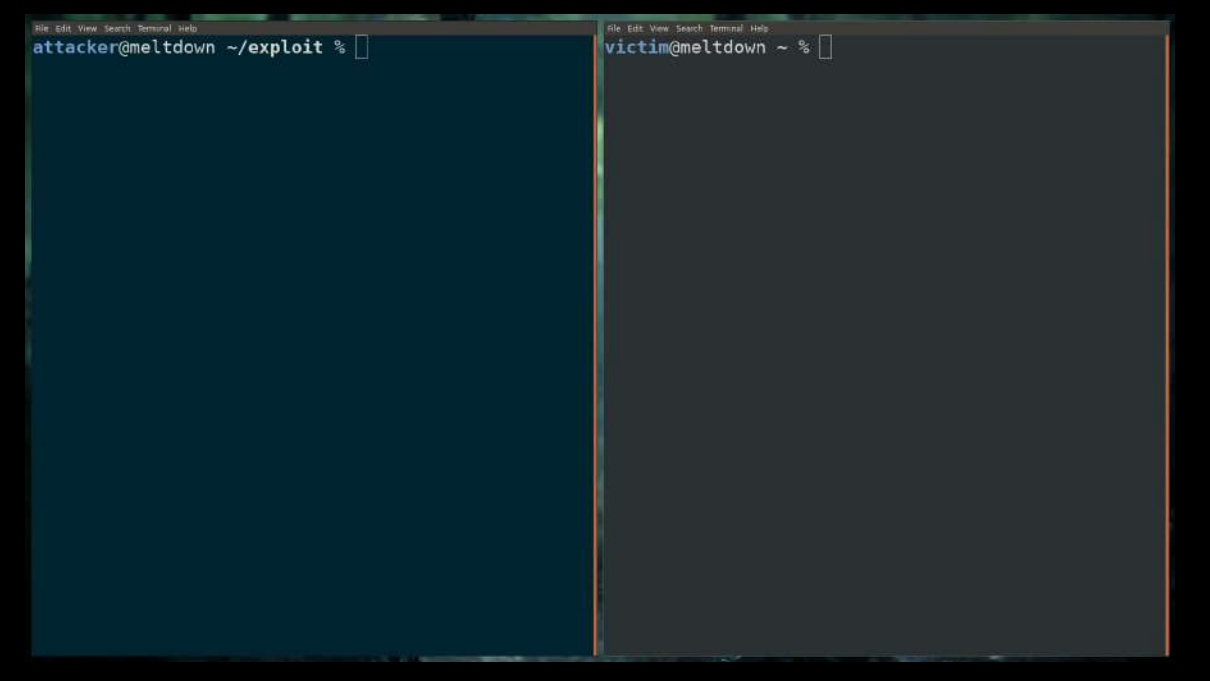
- Combine the two things

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

Flush+Reload again...



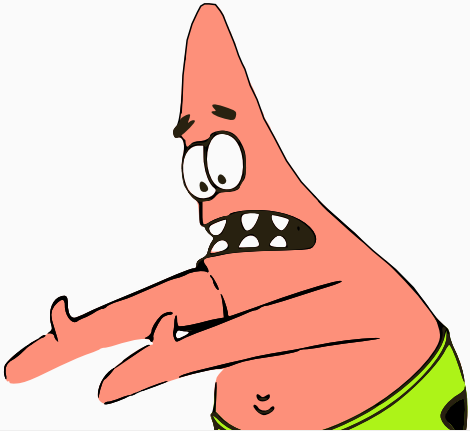
... Meltdown actually works.

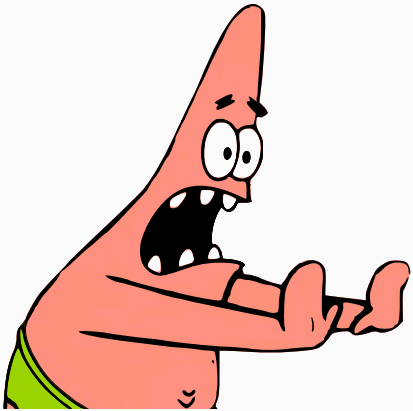

File Edit View Search Terminal Help
attacker@meltdown ~/exploit %
File Edit View Search Terminal Help
victim@meltdown ~ %

Take the kernel addresses...

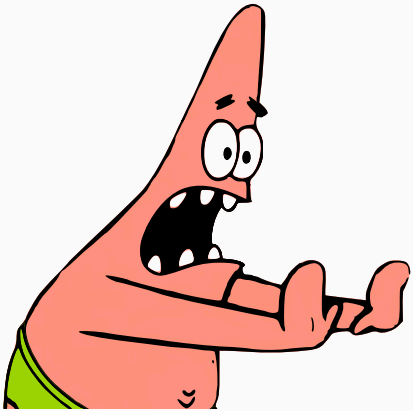
- Kernel addresses in user space are a problem (the wall does not work)

- Kernel addresses in user space are a problem (the wall does not work)
- 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

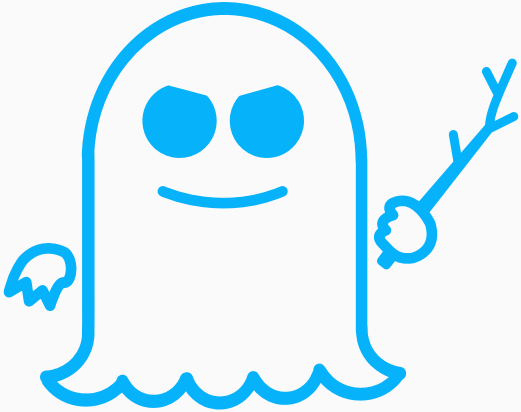
Let us get rid of bottlenecks

Use the
naughty/nice list
of last year

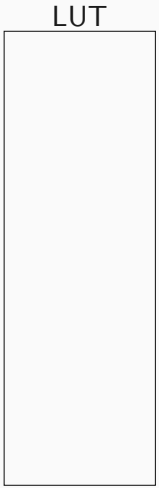
Finally, check predictions with list of this year

Throwing away wrongly manufactured presents

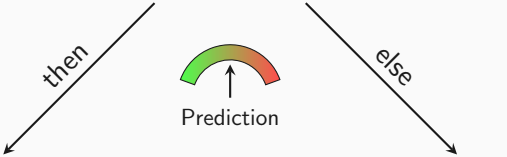
Correct predictions result in free time



SPECTRE

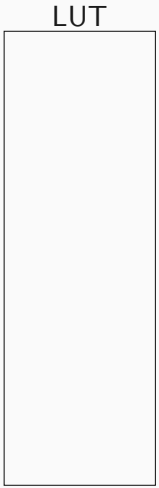



```
index = 0;

char* data = "textKEY";

if (index < 4)
```




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

0




```
index = 0;

char* data = "textKEY";

if (index < 4)
```

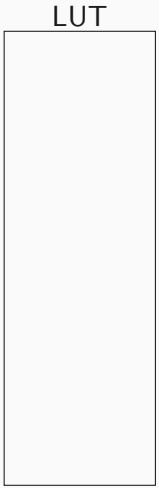
then

Prediction

else

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

```
0
```




```
index = 0;
char* data = "textKEY";
if (index < 4)
```

then

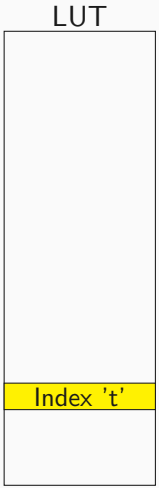
else

Prediction

Speculate

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

0




```
index = 0;

char* data = "textKEY";

if (index < 4)
```

Execute

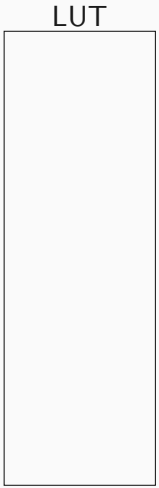
then

Prediction

else

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

0



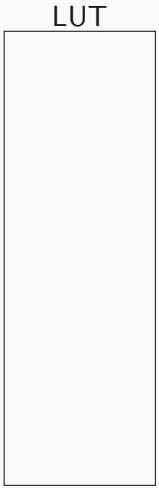

```
index = 1;

char* data = "textKEY";

if (index < 4)
```

then

else

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

0



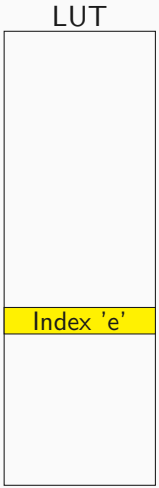

```
index = 1;

char* data = "textKEY";

if (index < 4)
```

then

else

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

0




```
index = 1;

char* data = "textKEY";

if (index < 4)
```

Speculate

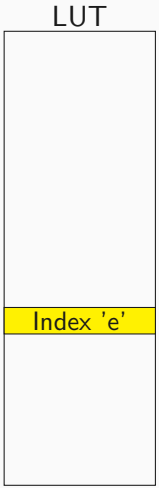
then

Prediction

else

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

0



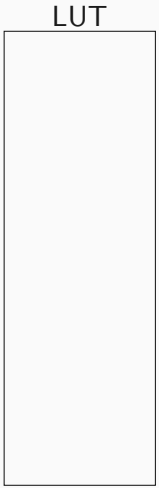

```
index = 1;

char* data = "textKEY";

if (index < 4)
```

then

else

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

0



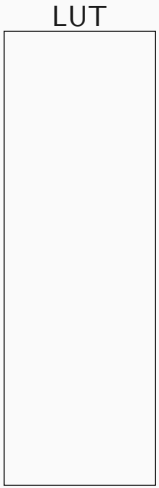

```
index = 2;

char* data = "textKEY";

if (index < 4)
```

then

else

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

```
0
```




```
index = 2;

char* data = "textKEY";

if (index < 4)
```

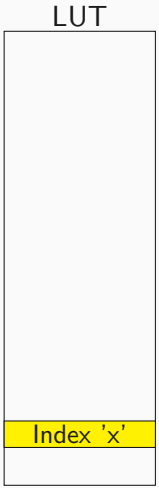
then

Prediction

else

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

0




```
index = 2;

char* data = "textKEY";

if (index < 4)
```

Speculate

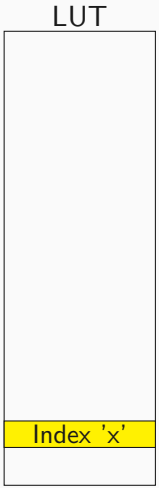
then

Prediction

else

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

0




```
index = 2;

char* data = "textKEY";

if (index < 4)
```

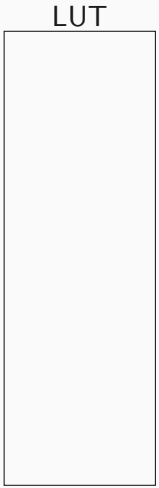
then

Prediction

else

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

0



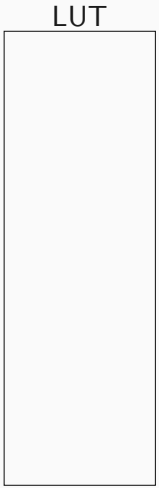

```
index = 3;

char* data = "textKEY";

if (index < 4)
```

then

else

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

0




```
index = 3;

char* data = "textKEY";

if (index < 4)
```

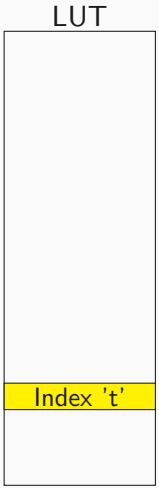
then

Prediction

else

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

0




```
index = 3;

char* data = "textKEY";

if (index < 4)
```

Speculate

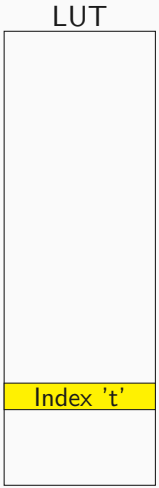
then

else

Prediction

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

0



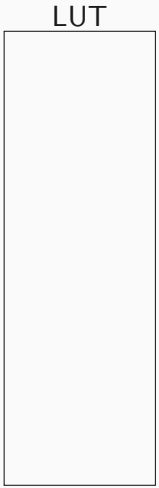

```
index = 3;

char* data = "textKEY";

if (index < 4)
```

then

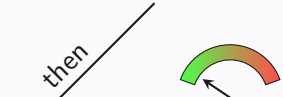
else

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

0

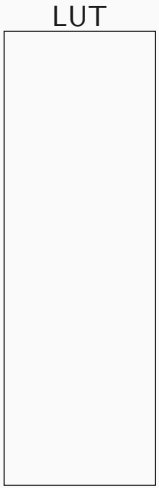



```
index = 4;

char* data = "textKEY";

if (index < 4)
```




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

0




```
index = 4;

char* data = "textKEY";

if (index < 4)
```

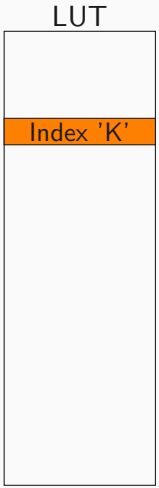
then

Prediction

else

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

0




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

Speculate

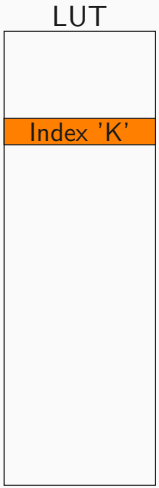
then

else

Prediction

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

0




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

then

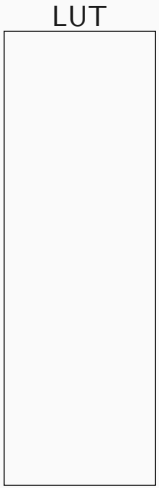
else

Prediction

Execute

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

0




```
index = 5;

char* data = "textKEY";

if (index < 4)
```

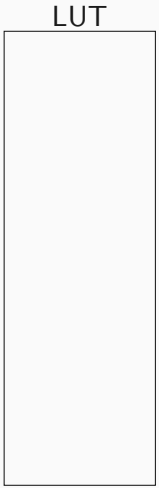
then

Prediction

else

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

0



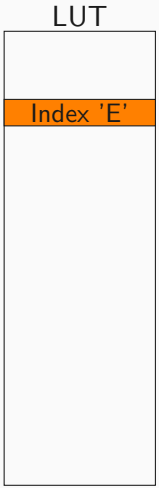

```
index = 5;

char* data = "textKEY";

if (index < 4)
```

then

else

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

0




```
index = 5;

char* data = "textKEY";

if (index < 4)
```

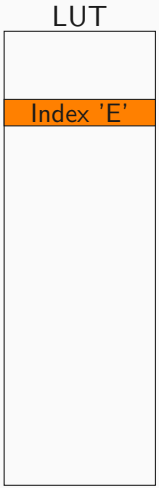
Speculate

then

else


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

0




```
index = 5;

char* data = "textKEY";

if (index < 4)
```

then

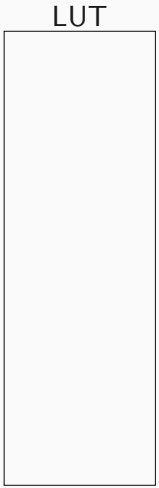
Prediction

else

Execute

0

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



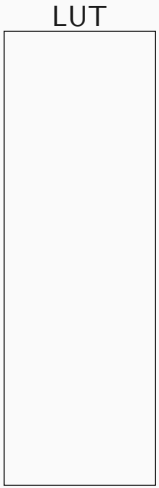

```
index = 6;

char* data = "textKEY";

if (index < 4)
```

then

else

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

0




```
index = 6;

char* data = "textKEY";

if (index < 4)
```

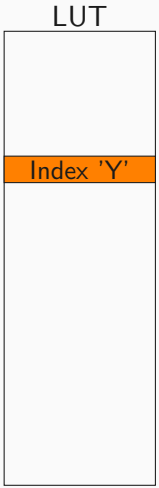
then

Prediction

else

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

0




```
index = 6;

char* data = "textKEY";

if (index < 4)
```

Speculate

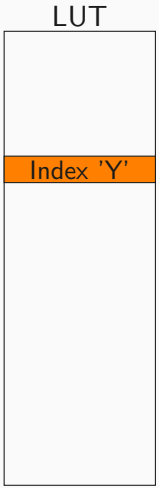
then

Prediction

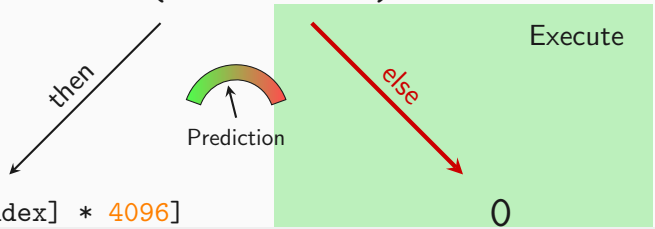
else

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

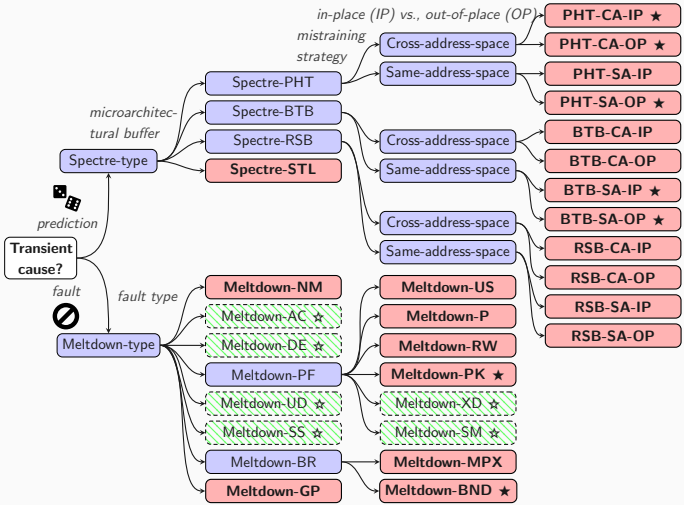
0




```
index = 6;

char* data = "textKEY";

if (index < 4)
```



Systematization Tree




BLOCKCHAIN
HD
HISTORY.COM

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 (◇).

- new class of attacks

Conclusions

- 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

Software-based Microarchitectural Attacks

Daniel Gruss

January 23, 2019

Graz University of Technology