

Side Channels and Transient Execution Attacks

Daniel Gruss

July 30, 2019

Graz University of Technology

amazon.com
Prime+Probe

ROWHAMMER IS ANOTHER FLIP IN THE ROW



FANTASTIC TIMERS

AND WHERE
TO FIND THEM

HIGH-RESOLUTION MICROARCHITECTURAL
ATTACKS IN JAVASCRIPT



JavaScript
zero

REAL
JavaScript
AND ZERO
SIDE-CHANNEL
ATTACKS



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

CHANGE MY MIND



- Profiling cache utilization with performance counters?



- Profiling cache utilization with performance counters? → No





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



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



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



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



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



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

Intel Analysis of Speculative Execution Side Channels

[Download PDF](#)



1

of 12

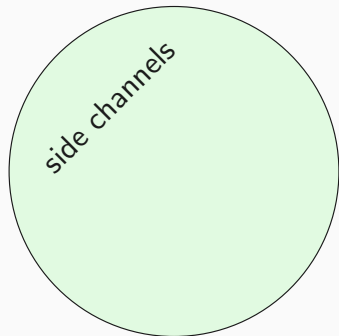


100%

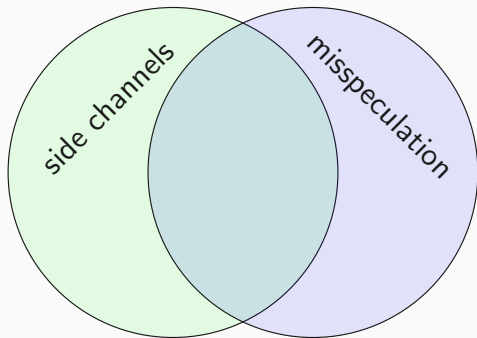


Intel Analysis of Speculative Execution Side Channels

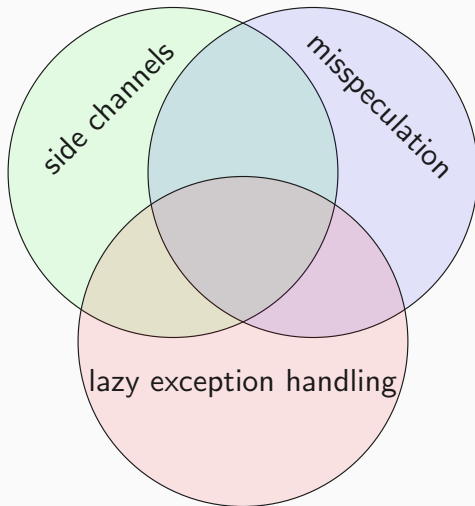
[White Paper](#)



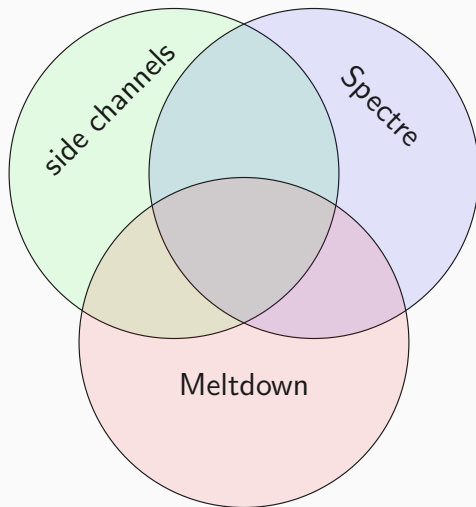
- traditional cache attacks (crypto, keys, etc)



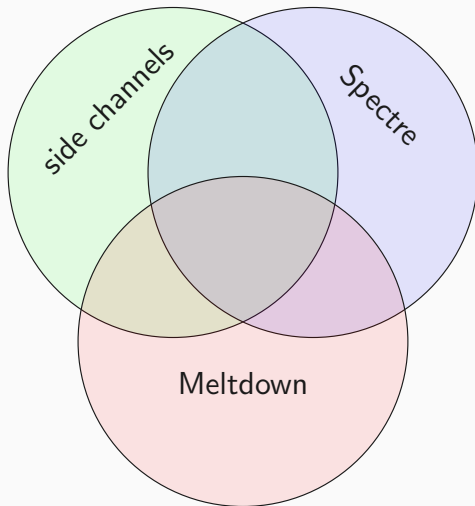
- traditional cache attacks (crypto, keys, etc)
- actual misspeculation (e.g., branch misprediction)



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



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



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









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.

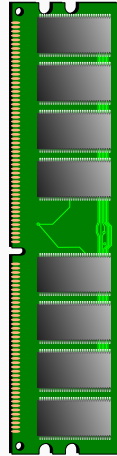
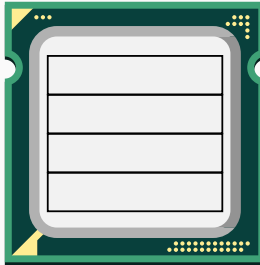
ONLY TODAY INSTEAD OF ~~\$1,300~~

\$1,299

ORDER VIA PHONE: +555 12345

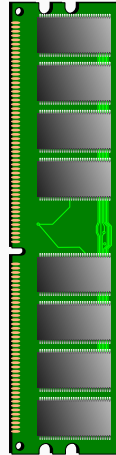
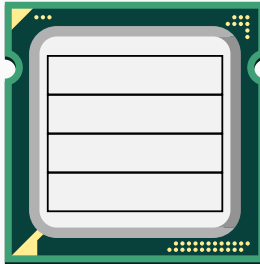


```
printf("%d", i);  
printf("%d", i);
```



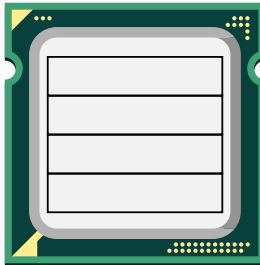
```
printf("%d", i);  
printf("%d", i);
```

Cache miss

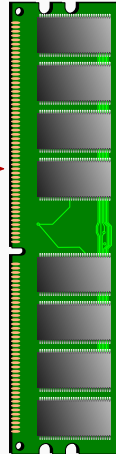


```
printf("%d", i);  
printf("%d", i);
```

Cache miss

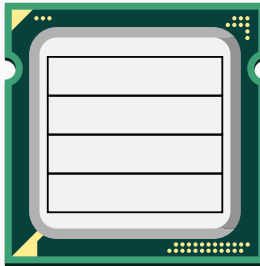


Request



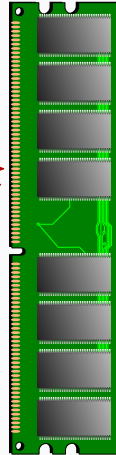
```
printf("%d", i);  
printf("%d", i);
```

Cache miss



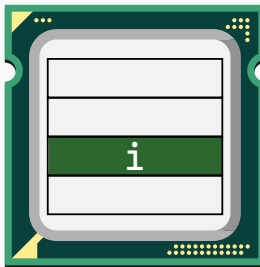
Request

Response

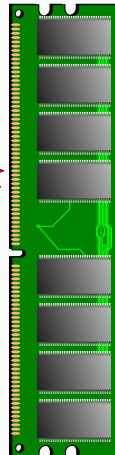


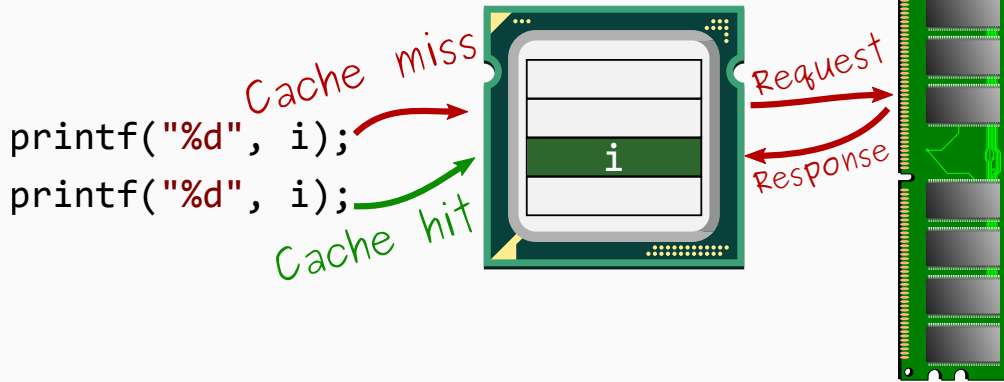
```
printf("%d", i);  
printf("%d", i);
```

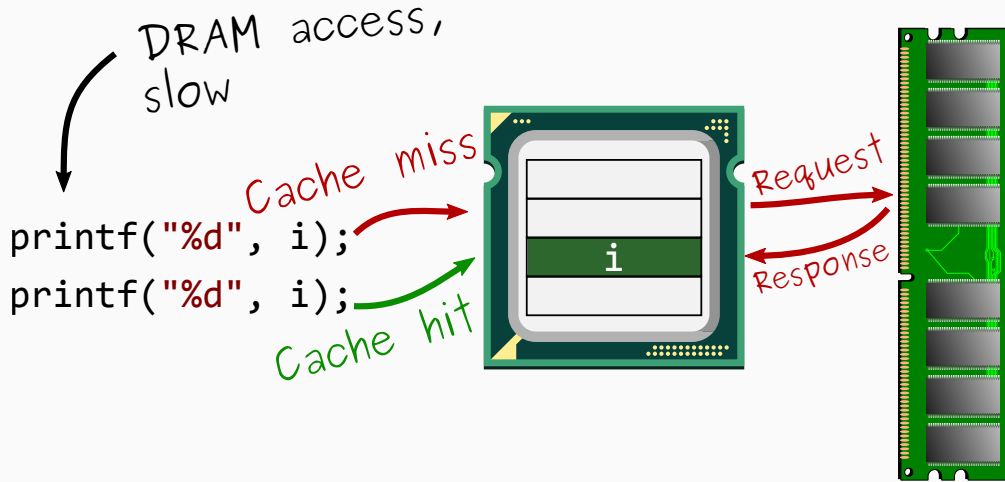
Cache miss

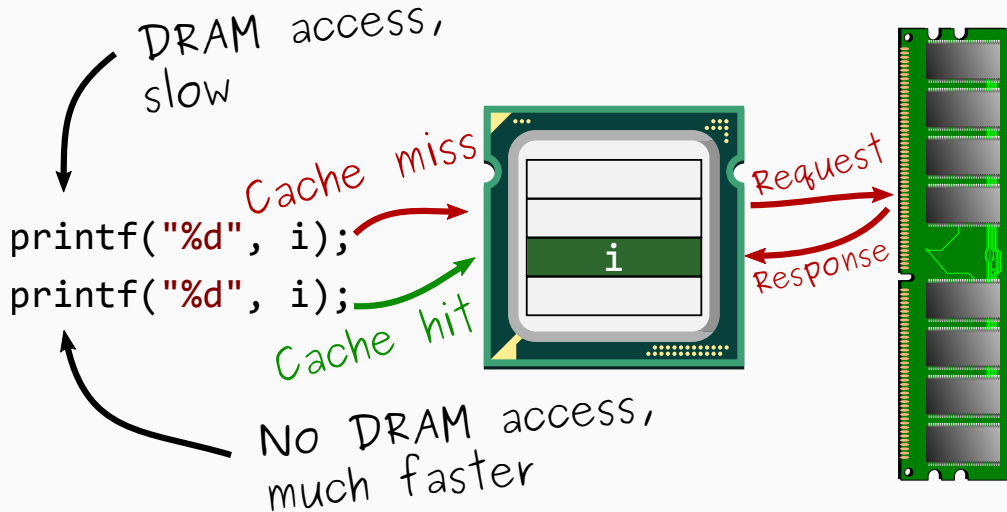


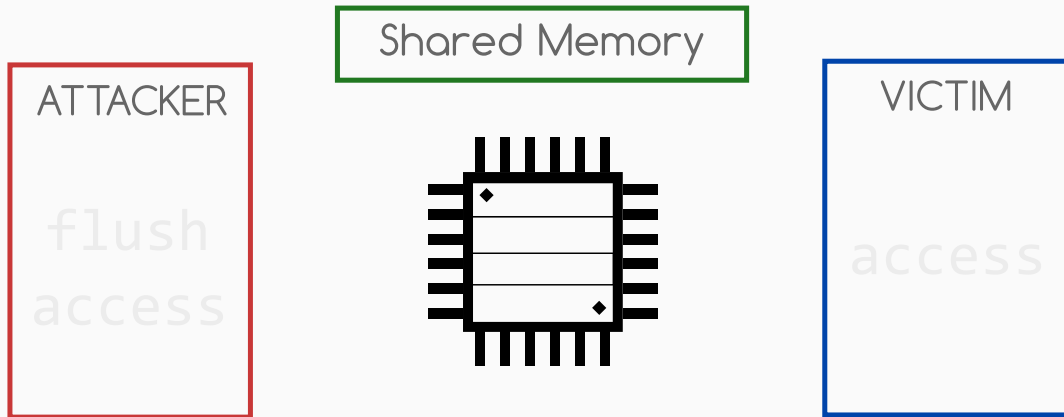
Request
Response

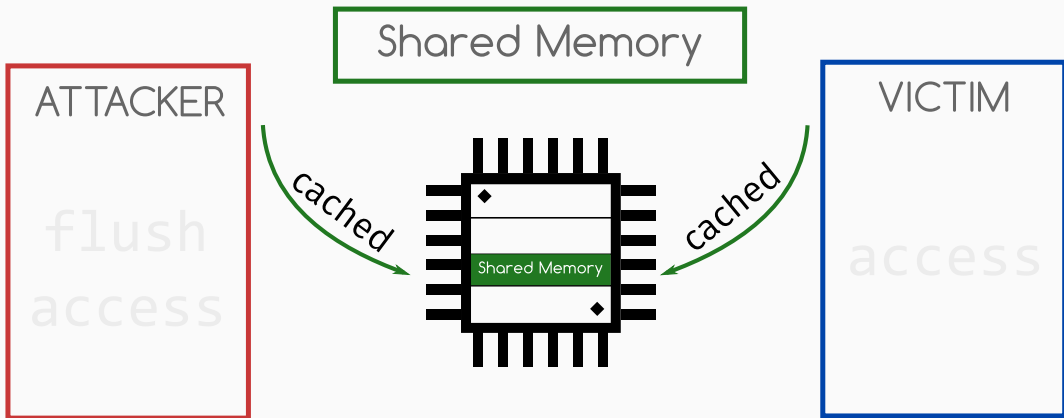


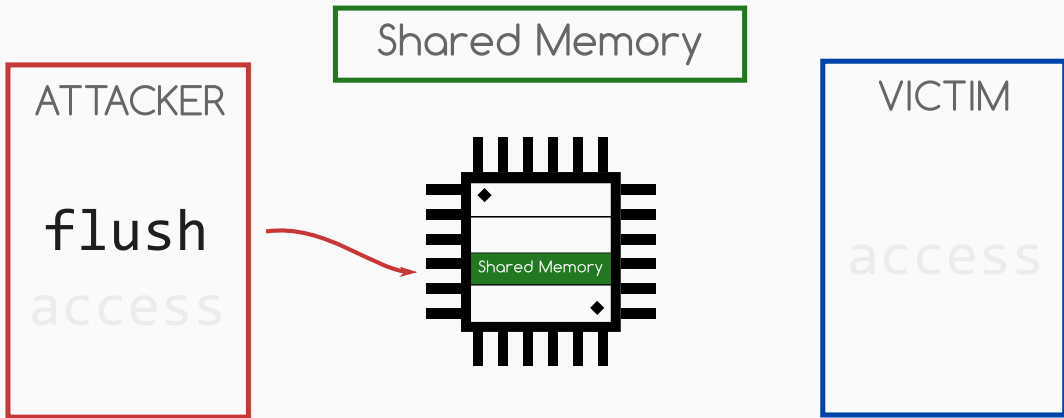


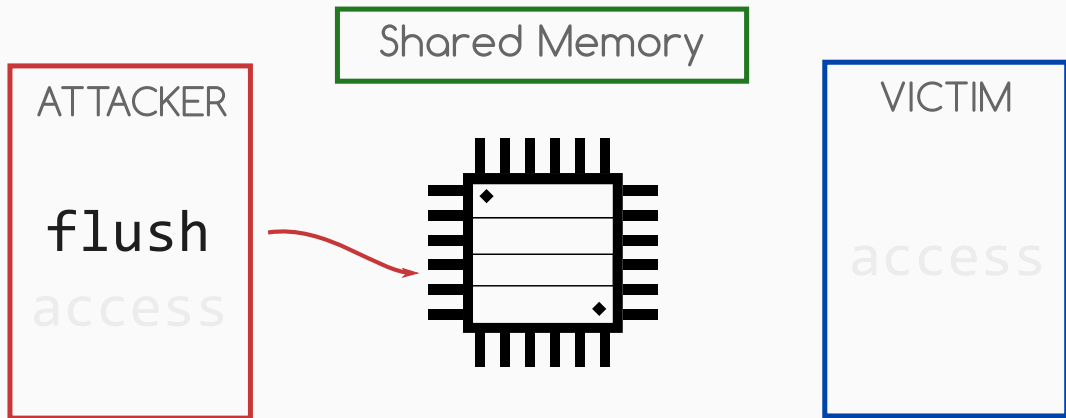


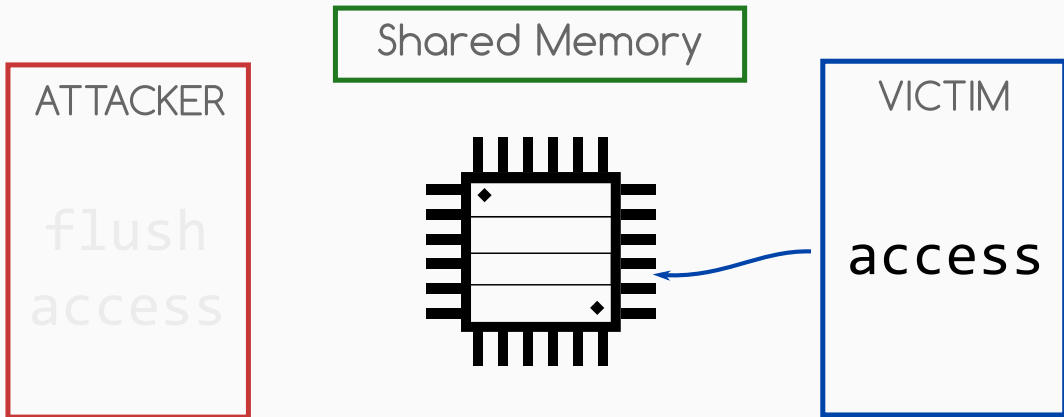


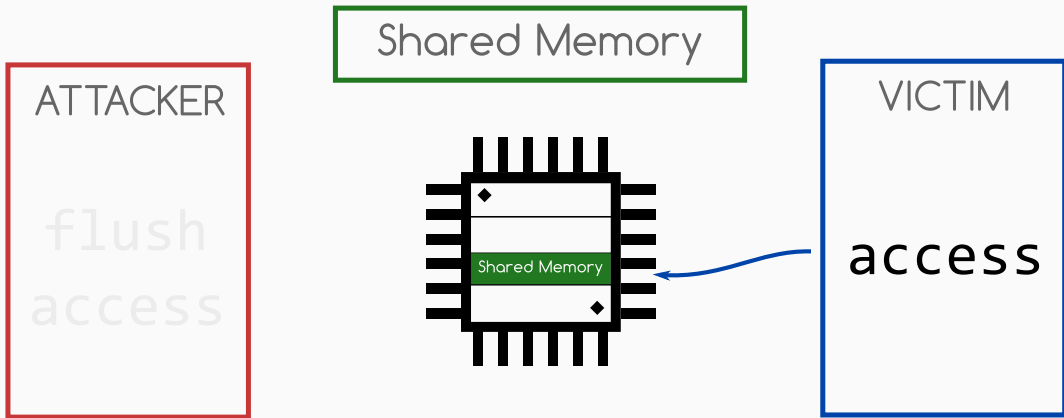


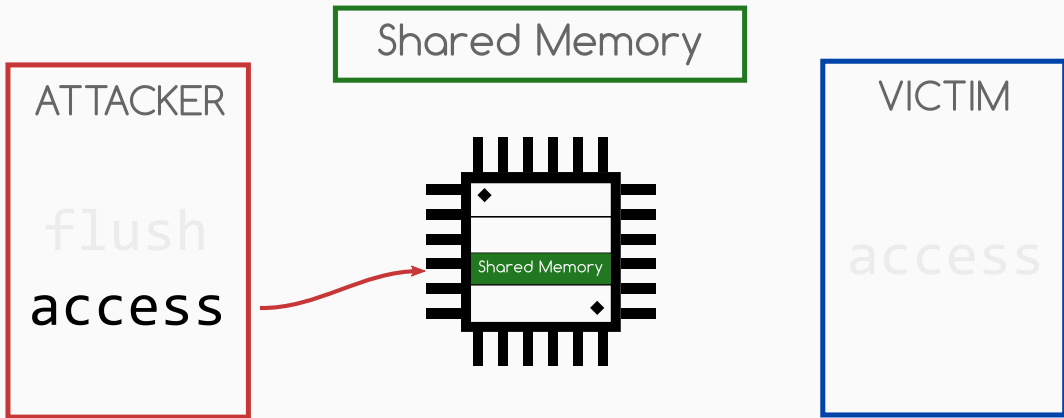


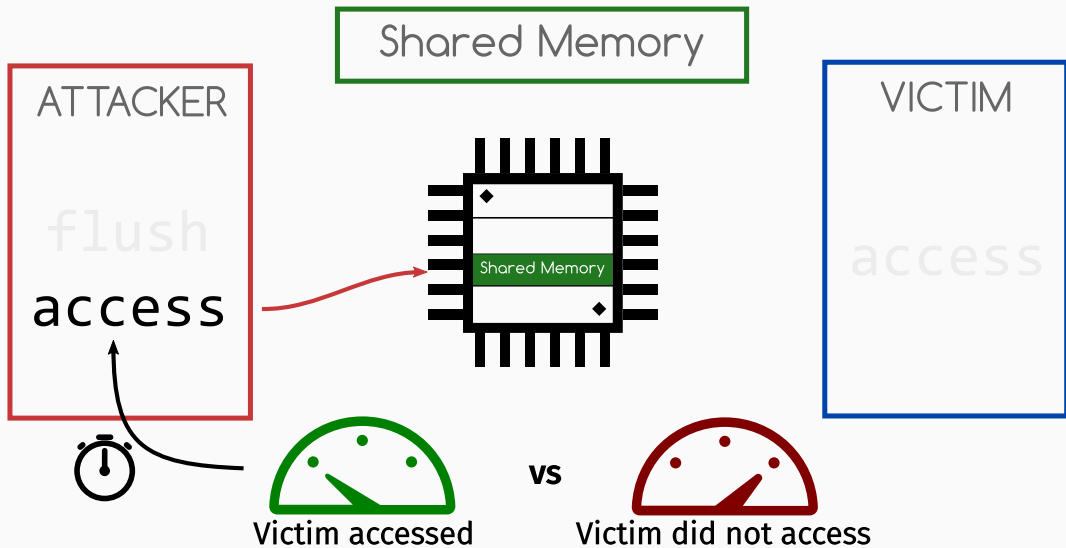












- use pseudo-serializing instruction `rdtscp` (recent CPUs)

- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cuid`

- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cuid`
- and/or use fences like `mfence`

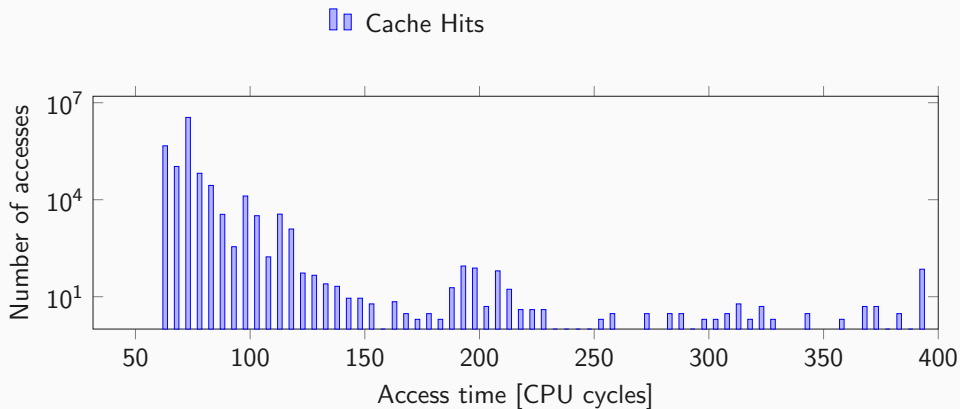
- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cuid`
- and/or use fences like `mfence`

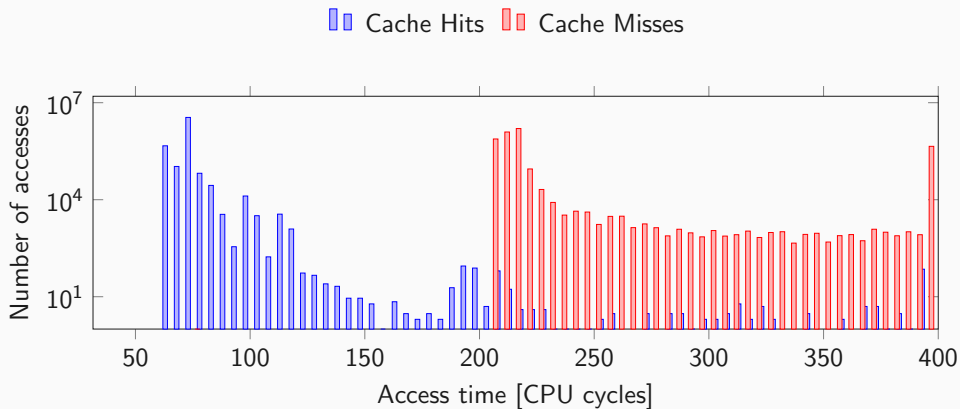
Intel, *How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures White Paper*, December 2010.

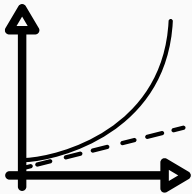
AUGUST 22, 2018 BY BRUCE

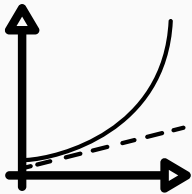
Intel Publishes Microcode Security Patches, No Benchmarking Or Comparison Allowed!

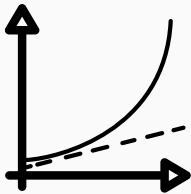
UPDATE: **Intel has resolved their microcode licensing issue which I complained about in this blog post.** The new license text is [here](#).



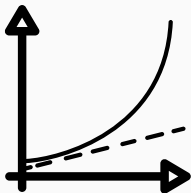




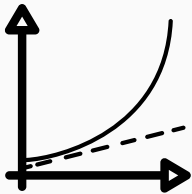




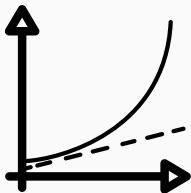
- Flush+Reload had beautifully nice timings, right?



- Flush+Reload had beautifully nice timings, right?
- Well... steps of 2-4 cycles



- Flush+Reload had beautifully nice timings, right?
- Well... steps of 2-4 cycles
 - only 35-70 steps between hits and misses



- Flush+Reload had beautifully nice timings, right?
- Well... steps of 2-4 cycles
 - only 35-70 steps between hits and misses
- On some devices only 1-2 steps!



- We can build our **own timer** [Lip+16; Sch+17]



- We can build our **own timer** [Lip+16; Sch+17]
- Start a thread that continuously increments a global variable



- We can build our **own timer** [Lip+16; Sch+17]
- Start a thread that continuously increments a global variable
- The global variable is our **timestamp**





**ARE YOU REALLY EXPECTING TO
OUTPERFORM THE HARDWARE COUNTER?**

CPU cycles one increment takes

rdtsc  3

```
1 timestamp = rdtsc();
```

CPU cycles one increment takes

rdtsc  3

C

```
1 while(1) {  
2     timestamp++;  
3 }
```

CPU cycles one increment takes

rdtsc  3

C  4.7

```
1 while(1) {  
2     timestamp++;  
3 }
```

CPU cycles one increment takes

rdtsc  3

C  4.7

```
1 while(1) {  
2     timestamp++;  
3 }
```

CPU cycles one increment takes

rdtsc  3

C  4.7

```
1 mov &timestamp, %rcx
2 1: incl (%rcx)
3 jmp 1b
```

Assembly

CPU cycles one increment takes



```
1 mov &timestamp, %rcx  
2 1: incl (%rcx)  
3 jmp 1b
```

CPU cycles one increment takes

rdtsc  3

C  4.7

Assembly  4.67

```
1 mov &timestamp, %rcx
2 1: incl (%rcx)
3 jmp 1b
```

CPU cycles one increment takes

rdtsc 3

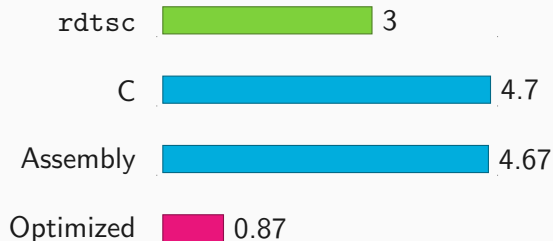
C 4.7

Assembly 4.67

Optimized

```
1 mov &timestamp, %rcx
2 1: inc %rax
3 mov %rax, (%rcx)
4 jmp 1b
```

CPU cycles one increment takes

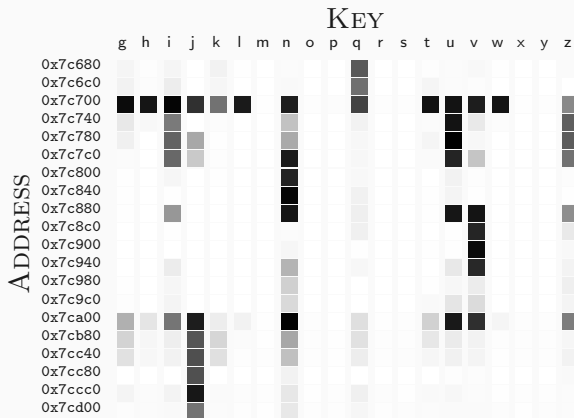


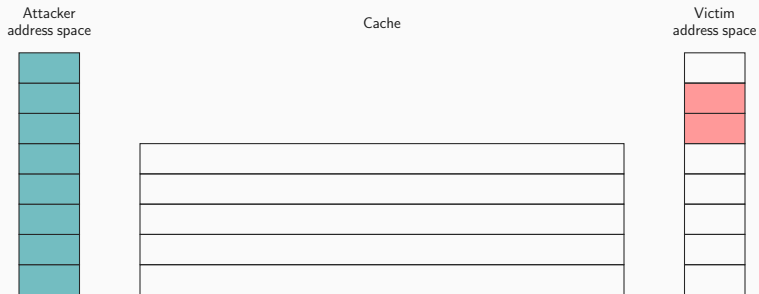
```
1 mov &timestamp, %rcx
2 1: inc %rax
3 mov %rax, (%rcx)
4 jmp 1b
```

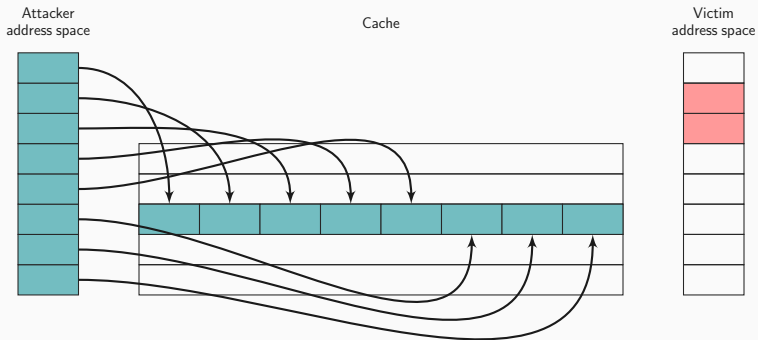
```
Terminal
File Edit View Search Terminal Help
% sleep 2; ./spy 300 7f05140a4000-7f051417b000 r-xp 0x20000 08:02 26
8050 /usr/lib/x86_64-linux-gnu/gedit/libgedit.so
```

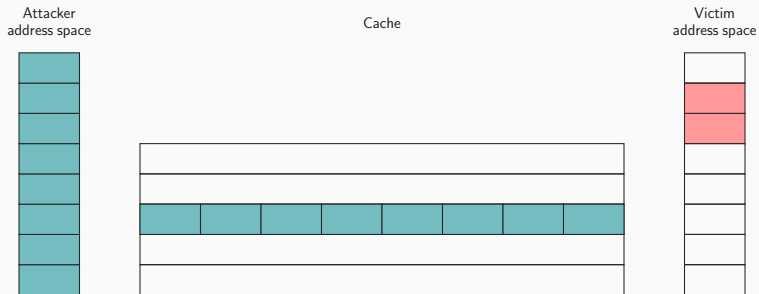
```
Terminal
File Edit View Search Terminal Help
shark% ./spy
```

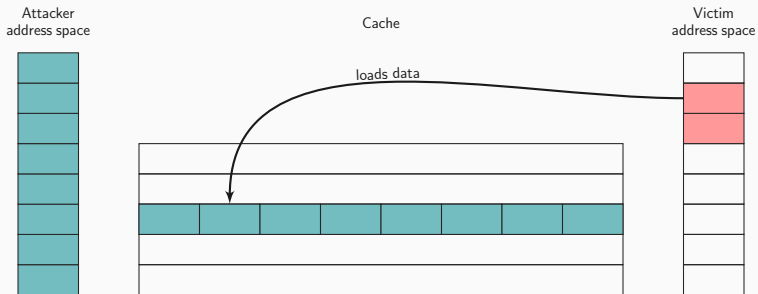
```
Untitled Document 1
Open + Save - + x
1
I
Plain Text Tab Width: 2 Ln 1, Col 1 INS
```

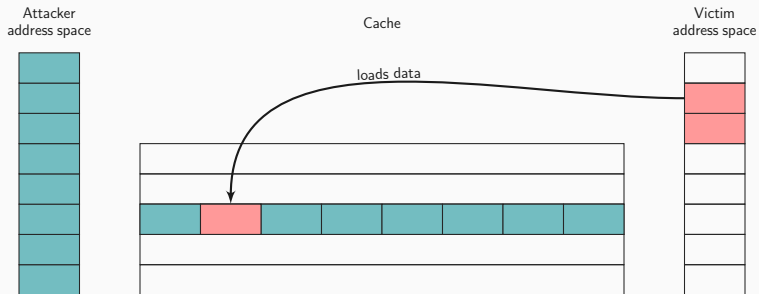


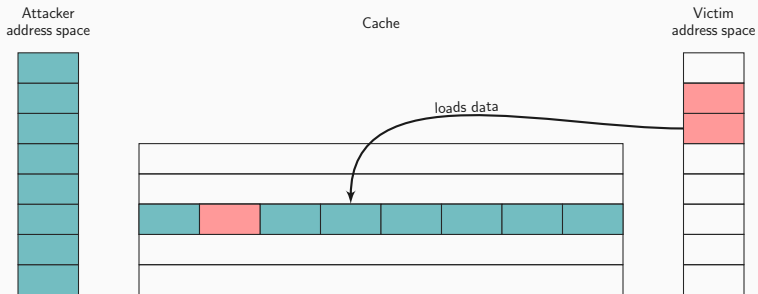


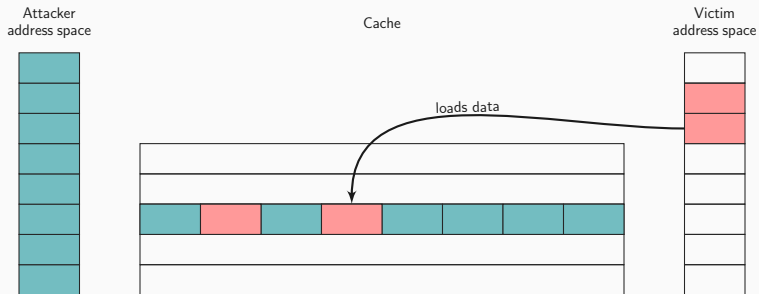


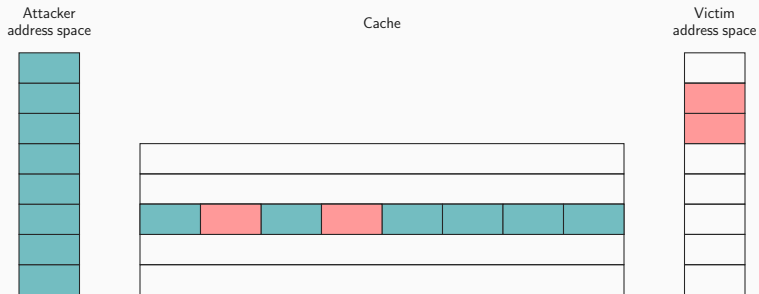


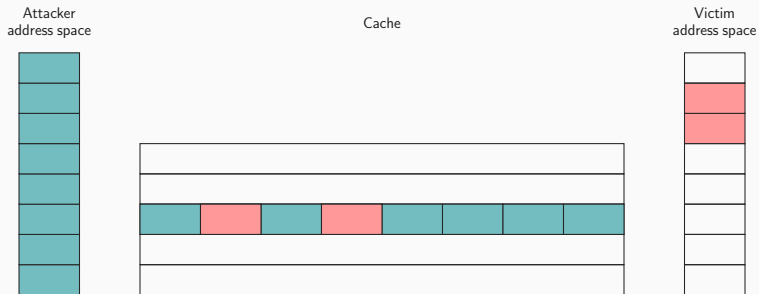


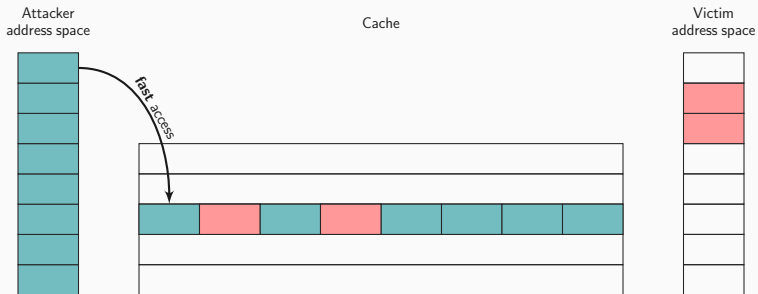


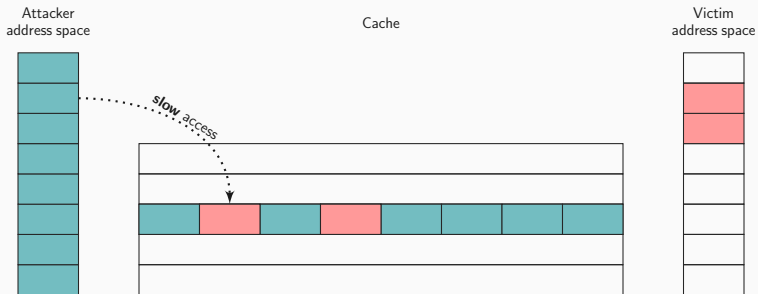












Pros: less restrictive

1. no need for `clflush` instruction (not available e.g., in JS)

Pros: less restrictive

1. no need for `clflush` instruction (not available e.g., in JS)
2. no need for shared memory ($\rightarrow$ cross-VM)

Pros: less restrictive

1. no need for `clflush` instruction (not available e.g., in JS)
2. no need for shared memory ($\rightarrow$ cross-VM)

Pros: less restrictive

1. no need for `clflush` instruction (not available e.g., in JS)
2. no need for shared memory ($\rightarrow$ cross-VM)

Cons: coarser granularity (1 set)

“LRU eviction” memory accesses

cache set



“LRU eviction” memory accesses

cache set



- LRU replacement policy: oldest entry first

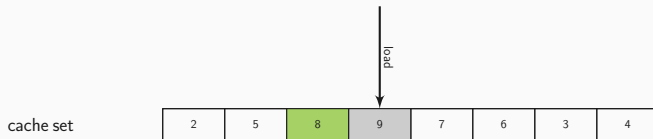
“LRU eviction” memory accesses

cache set

2	5	8	1	7	6	3	4
---	---	---	---	---	---	---	---

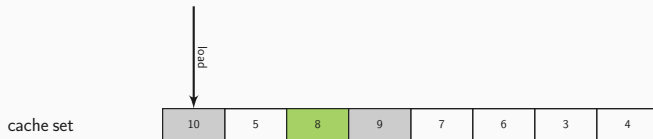
- LRU replacement policy: oldest entry first
- timestamps for every cache line

“LRU eviction” memory accesses



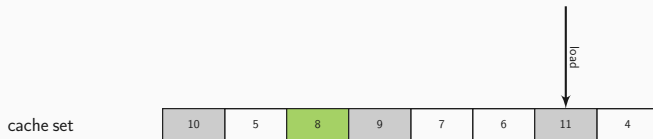
- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

“LRU eviction” memory accesses



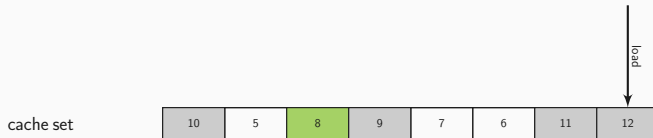
- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

“LRU eviction” memory accesses



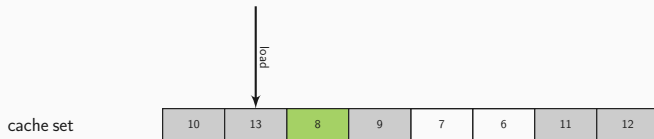
- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

“LRU eviction” memory accesses



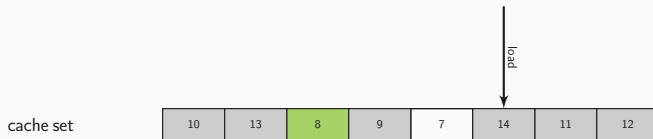
- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

“LRU eviction” memory accesses



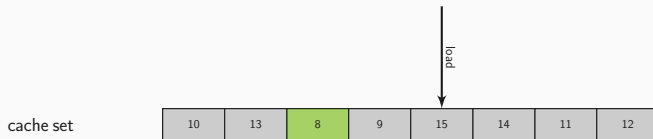
- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

“LRU eviction” memory accesses



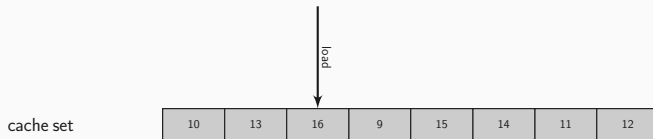
- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

“LRU eviction” memory accesses



- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

“LRU eviction” memory accesses



- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

THAT'S NOT HOW THIS WORKS



**THAT'S NOT HOW ANY OF
THIS WORKS**

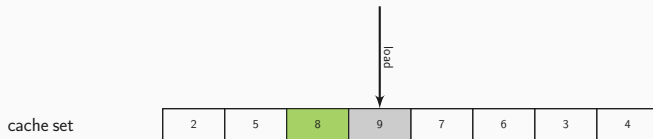
“LRU eviction” memory accesses

cache set

2	5	8	1	7	6	3	4
---	---	---	---	---	---	---	---

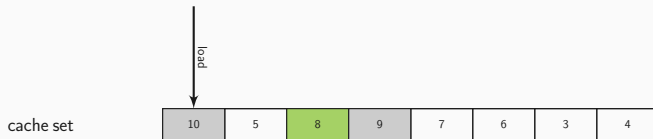
- no LRU replacement

“LRU eviction” memory accesses



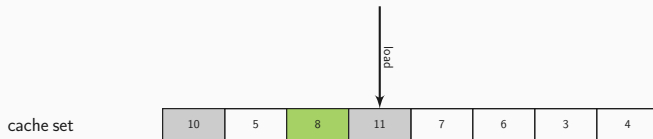
- no LRU replacement

“LRU eviction” memory accesses



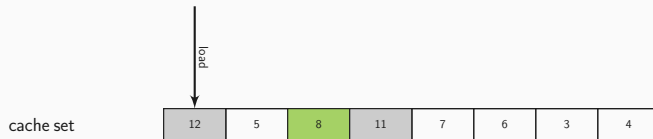
- no LRU replacement

“LRU eviction” memory accesses



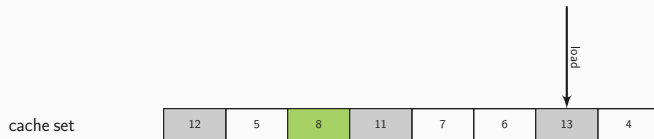
- no LRU replacement

“LRU eviction” memory accesses



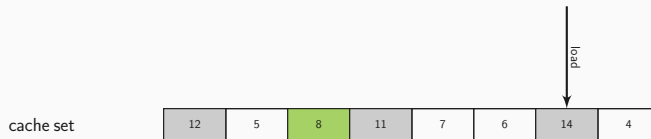
- no LRU replacement

“LRU eviction” memory accesses



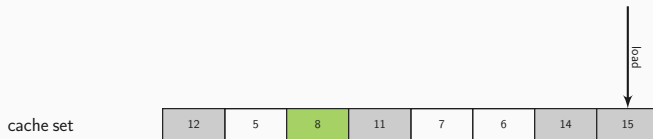
- no LRU replacement

“LRU eviction” memory accesses



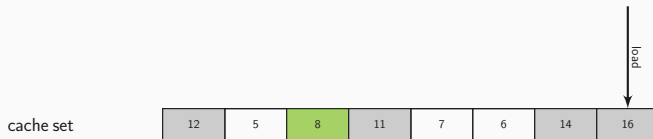
- no LRU replacement

“LRU eviction” memory accesses



- no LRU replacement

“LRU eviction” memory accesses



- no LRU replacement

“LRU eviction” memory accesses

cache set

12	5	8	11	7	6	14	16
----	---	---	----	---	---	----	----

- no LRU replacement
- only 75% success rate on Haswell

“LRU eviction” memory accesses

cache set

12	5	8	11	7	6	14	16
----	---	---	----	---	---	----	----

- no LRU replacement
- only 75% success rate on Haswell
- more accesses → higher success rate, but **too slow**

Write eviction strategies as: *P-C-D-L-S*

```
for (s = 0; s <= S - D; s += L)
  for (c = 0; c <= C; c += 1)
    for (d = 0; d <= D; d += 1)
      *a[s+d];
```

Write eviction strategies as: $P-C-D-L-S$

S : total number of different addresses
(= set size)

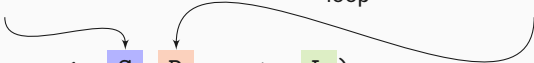


```
for (s = 0; s <= S - D; s += L)
  for (c = 0; c <= C; c += 1)
    for (d = 0; d <= D; d += 1)
      *a[s+d];
```

Write eviction strategies as: $P-C-D-L-S$

S : total number of different addresses
(= set size)

D : different addresses per inner access
loop



```
for (s = 0; s <= S - D; s += L)
  for (c = 0; c <= C; c += 1)
    for (d = 0; d <= D; d += 1)
      *a[s+d];
```

Write eviction strategies as: $P-C-D-L-S$

S : total number of different addresses
(= set size)

D : different addresses per inner access

```
for (s = 0; s <= S - D; s += L)
  for (c = 0; c <= C; c += 1)
    for (d = 0; d <= D; d += 1)
      *a[s+d];
```

L : step size of the inner access
loop

Write eviction strategies as: *P-C-D-L-S*

S: total number of different addresses
(= set size)

D: different addresses per inner access
loop

```
for (s = 0; s <= S - D; s += L)
  for (c = 0; c <= C; c += 1)
    for (d = 0; d <= D; d += 1)
      *a[s+d];
```

L: step size of the inner access
loop

C: number of repetitions of the inner
access loop

We evaluated more than 10000 strategies...¹

strategy	# accesses	eviction rate	loop time
<i>P</i> -1-1-1-17	17		
<i>P</i> -1-1-1-20	20		

¹Executed in a loop, on a Haswell with a 16-way last-level cache

We evaluated more than 10000 strategies...¹

strategy	# accesses	eviction rate	loop time
<i>P</i> -1-1-1-17	17	74.46% ✗	
<i>P</i> -1-1-1-20	20	99.82% ✓	

¹Executed in a loop, on a Haswell with a 16-way last-level cache

We evaluated more than 10000 strategies...¹

strategy	# accesses	eviction rate	loop time
<i>P</i> -1-1-1-17	17	74.46% ✗	307 ns ✓
<i>P</i> -1-1-1-20	20	99.82% ✓	934 ns ✗

¹Executed in a loop, on a Haswell with a 16-way last-level cache

We evaluated more than 10000 strategies...¹

strategy	# accesses	eviction rate	loop time
<i>P</i> -1-1-1-17	17	74.46% ✗	307 ns ✓
<i>P</i> -1-1-1-20	20	99.82% ✓	934 ns ✗
<i>P</i> -2-1-1-17	34		

¹Executed in a loop, on a Haswell with a 16-way last-level cache

We evaluated more than 10000 strategies...¹

strategy	# accesses	eviction rate	loop time
<i>P</i> -1-1-1-17	17	74.46% ✗	307 ns ✓
<i>P</i> -1-1-1-20	20	99.82% ✓	934 ns ✗
<i>P</i> -2-1-1-17	34	99.86% ✓	

¹Executed in a loop, on a Haswell with a 16-way last-level cache

We evaluated more than 10000 strategies...¹

strategy	# accesses	eviction rate	loop time
<i>P</i> -1-1-1-17	17	74.46% ✗	307 ns ✓
<i>P</i> -1-1-1-20	20	99.82% ✓	934 ns ✗
<i>P</i> -2-1-1-17	34	99.86% ✓	191 ns ✓

¹Executed in a loop, on a Haswell with a 16-way last-level cache

We evaluated more than 10000 strategies...¹

strategy	# accesses	eviction rate	loop time
<i>P</i> -1-1-1-17	17	74.46% ✗	307 ns ✓
<i>P</i> -1-1-1-20	20	99.82% ✓	934 ns ✗
<i>P</i> -2-1-1-17	34	99.86% ✓	191 ns ✓
<i>P</i> -2-2-1-17	64		

¹Executed in a loop, on a Haswell with a 16-way last-level cache

We evaluated more than 10000 strategies...¹

strategy	# accesses	eviction rate	loop time
<i>P</i> -1-1-1-17	17	74.46% ✗	307 ns ✓
<i>P</i> -1-1-1-20	20	99.82% ✓	934 ns ✗
<i>P</i> -2-1-1-17	34	99.86% ✓	191 ns ✓
<i>P</i> -2-2-1-17	64	99.98% ✓	

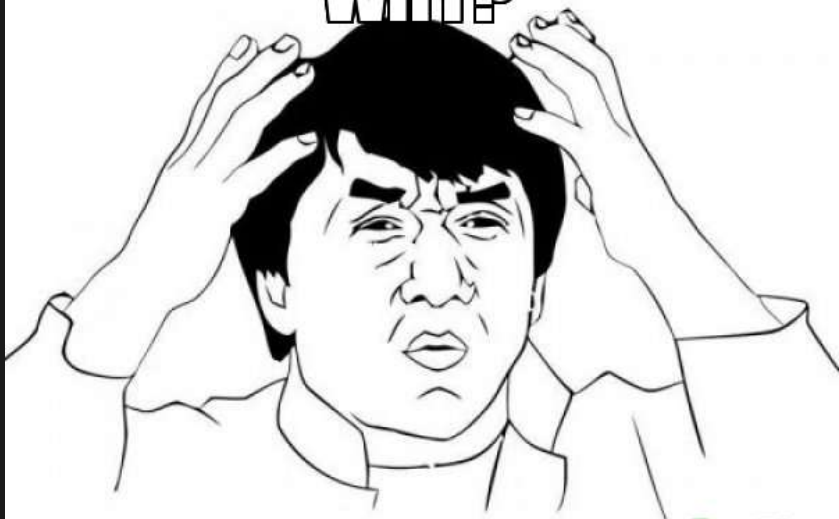
¹Executed in a loop, on a Haswell with a 16-way last-level cache

We evaluated more than 10000 strategies...¹

strategy	# accesses	eviction rate	loop time
<i>P</i> -1-1-1-17	17	74.46% ✗	307 ns ✓
<i>P</i> -1-1-1-20	20	99.82% ✓	934 ns ✗
<i>P</i> -2-1-1-17	34	99.86% ✓	191 ns ✓
<i>P</i> -2-2-1-17	64	99.98% ✓	180 ns ✓

¹Executed in a loop, on a Haswell with a 16-way last-level cache

WHY?



Memes Happen

$P-1-1-1-17$ (17 accesses, 307ns)

$P-2-1-1-17$ (34 accesses, 191ns)

Time in ns



$P-1-1-1-17$ (17 accesses, 307ns)

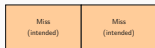


$P-2-1-1-17$ (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)

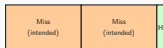


P -2-1-1-17 (34 accesses, 191ns)

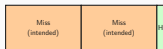


Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)

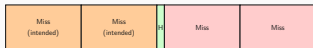


P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)

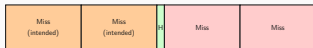


P -2-1-1-17 (34 accesses, 191ns)

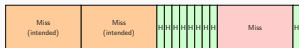


Time in ns

P -1-1-1-17 (17 accesses, 307ns)

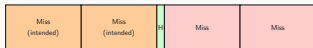


P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)

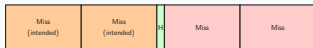


P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)

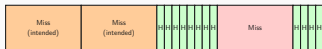


Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)

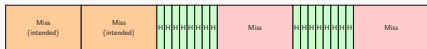


Time in ns

P -1-1-1-17 (17 accesses, 307ns)

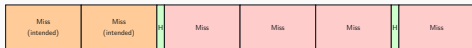


P -2-1-1-17 (34 accesses, 191ns)

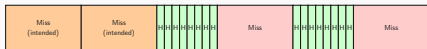


Time in ns

P -1-1-1-17 (17 accesses, 307ns)

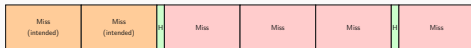


P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)

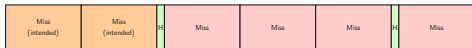


P -2-1-1-17 (34 accesses, 191ns)

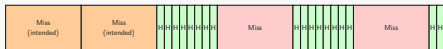


Time in ns

P -1-1-1-17 (17 accesses, 307ns)

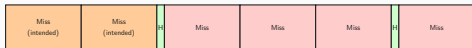


P -2-1-1-17 (34 accesses, 191ns)

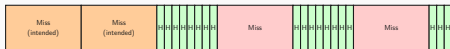


Time in ns

P -1-1-1-17 (17 accesses, 307ns)

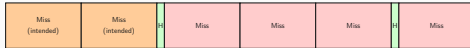


P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)

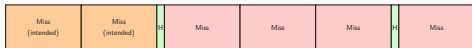


P -2-1-1-17 (34 accesses, 191ns)

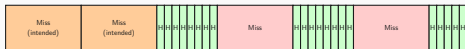


Time in ns

P -1-1-1-17 (17 accesses, 307ns)

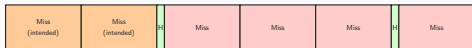


P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)

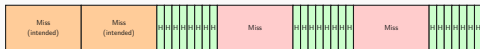


Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)

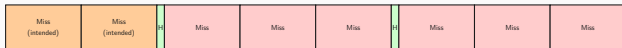


P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)

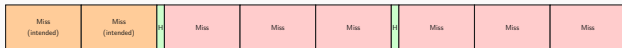


P -2-1-1-17 (34 accesses, 191ns)

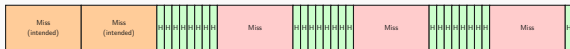


Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)

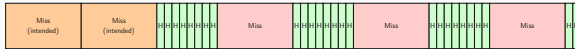


Time in ns

P -1-1-1-17 (17 accesses, 307ns)

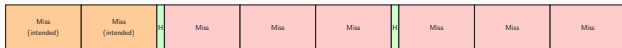


P -2-1-1-17 (34 accesses, 191ns)

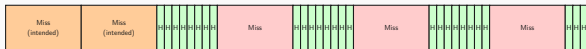


Time in ns

P -1-1-1-17 (17 accesses, 307ns)

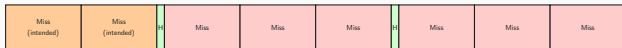


P -2-1-1-17 (34 accesses, 191ns)

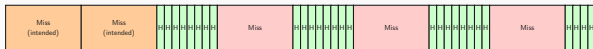


Time in ns

P -1-1-1-17 (17 accesses, 307ns)

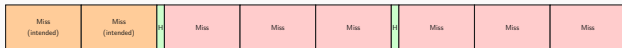


P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)

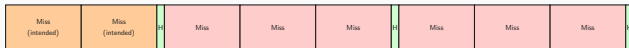


P -2-1-1-17 (34 accesses, 191ns)

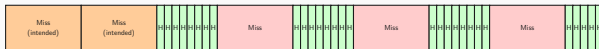


Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)

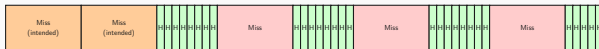


Time in ns

P -1-1-1-17 (17 accesses, 307ns)

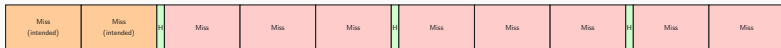


P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)

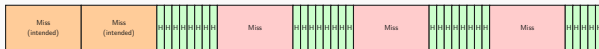


Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)



Time in ns

P -1-1-1-17 (17 accesses, 307ns)



P -2-1-1-17 (34 accesses, 191ns)



Time in ns



```
File Edit View Bookmarks Settings Help
local -- Konsole
ssh -l root 172.31.31.32 -p 22
```

```
File Edit View Bookmarks Settings Help
root@172-31-31-32 ~ %
```

```
File Edit View Bookmarks Settings Help
-- alert 0.4.5 on ip-172-31-31-32 --
[... ASCII art ...]
Active Interface: eth0
Interface Speed: unknown
Current RX Speed: 0.01 KB/s
Graph Top RX Speed: 0.98 KB/s
Overall Top RX Speed: 0.98 KB/s
Received Packets: 64
Bytes Received: 0.045 MB
Errors on Receiving: 0
Current TX Speed: 0.33 KB/s
Graph Top TX Speed: 2.64 KB/s
Overall Top TX Speed: 2.64 KB/s
Transmitted Packets: 61
Bytes Transmitted: 0.039 MB
Errors on Transmission: 0
```

```
File Edit View Bookmarks Settings Help
root@172-31-31-32 ~ %
```

```
File Edit View Bookmarks Settings Help
-- alert 0.4.5 on ip-172-31-31-32 --
[... ASCII art ...]
Active Interface: eth0
Interface Speed: unknown
Current RX Speed: 0.05 KB/s
Graph Top RX Speed: 0.36 KB/s
Overall Top RX Speed: 0.36 KB/s
Received Packets: 36
Bytes Received: 0.003 MB
Errors on Receiving: 0
Current TX Speed: 0.29 KB/s
Graph Top TX Speed: 0.84 KB/s
Overall Top TX Speed: 0.84 KB/s
Transmitted Packets: 26
Bytes Transmitted: 0.010 MB
Errors on Transmission: 0
```

HELLO FROM THE OTHER SIDE (DEMO):
VIDEO STREAMING OVER CACHE COVERT CHANNEL

Protection from Side-Channel Attacks

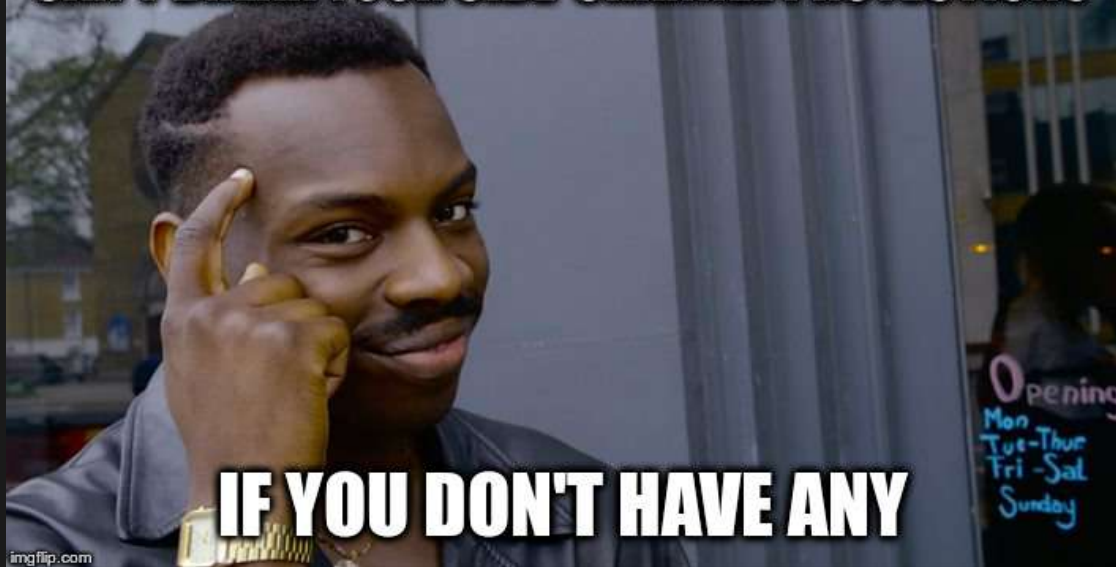
Protection from Side-Channel Attacks

Intel SGX does not provide explicit protection from side-channel attacks.

Protection from Side-Channel Attacks

Intel SGX does not provide explicit protection from side-channel attacks. It is the enclave developer's responsibility to address side-channel attack concerns.

CAN'T BREAK YOUR SIDE-CHANNEL PROTECTIONS



IF YOU DON'T HAVE ANY



- **Ledger SGX Enclave** for blockchain applications
- **BitPay Copay** Bitcoin wallet
- **Teechain** payment channel using SGX



- Ledger SGX Enclave for blockchain applications
- BitPay Copay Bitcoin wallet
- Teechain payment channel using SGX

Teechain

[...] We assume the TEE guarantees to hold



- **Ledger SGX Enclave** for blockchain applications
- **BitPay Copay** Bitcoin wallet
- **Teechain** payment channel using SGX

Teechain

[...] We assume the TEE guarantees to hold and do not consider side-channel attacks [5, 35, 46] on the TEE.



- **Ledger SGX Enclave** for blockchain applications
- **BitPay Copay** Bitcoin wallet
- **Teechain** payment channel using SGX

Teechain

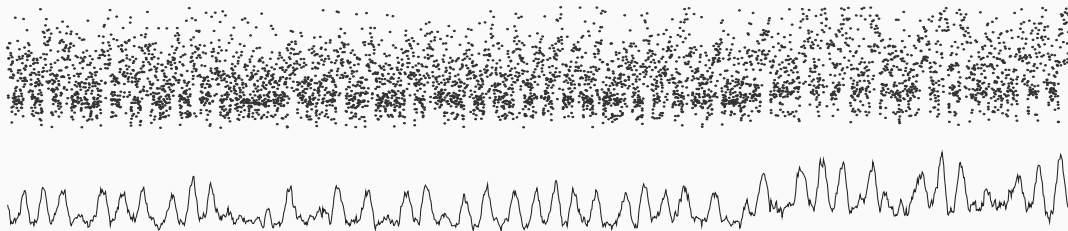
[...] We assume the TEE guarantees to hold and do not consider side-channel attacks [5, 35, 46] on the TEE. Such attacks and their mitigations [36, 43] are outside the scope of this work. [...]

Raw Prime+Probe trace...²



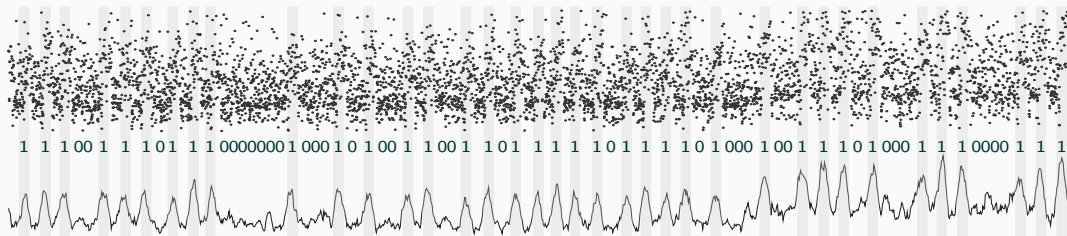
²Michael Schwarz et al. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In: DIMVA. 2017.

...processed with a simple moving average...³



³Michael Schwarz et al. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In: DIMVA. 2017.

...allows to clearly see the bits of the exponent⁴



⁴Michael Schwarz et al. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In: DIMVA. 2017.

YOU CAN'T DO THAT!



THAT'S AGAINST THE RULES!



Back to Work

*6. Cook everything until
vegetables are soft*

*6. Cook everything until
vegetables are soft*

*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;

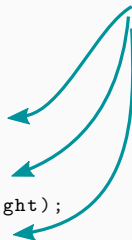
float diagonal = sqrt(width * width
                      + height * height);
int area = width * height;

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);
```





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



- Flush+Reload over all pages of the array





- Flush+Reload over all pages of the array



- “Unreachable” code line was **actually executed**



- Flush+Reload over all pages of the array



- “Unreachable” code line was **actually executed**
- Exception was only thrown **afterwards**



- Out-of-order instructions **leave microarchitectural traces**



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



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



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



- Add another **layer of indirection** to test

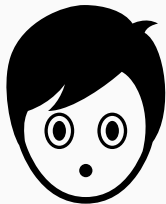
```
char data = *(char*) 0xffffffff81a000e0;  
array[data * 4096] = 0;
```



- Add another **layer of indirection** to test

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

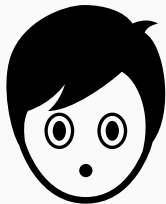
- Then check whether any part of array is **cached**



- Flush+Reload over all pages of the array



- Index of cache hit reveals data



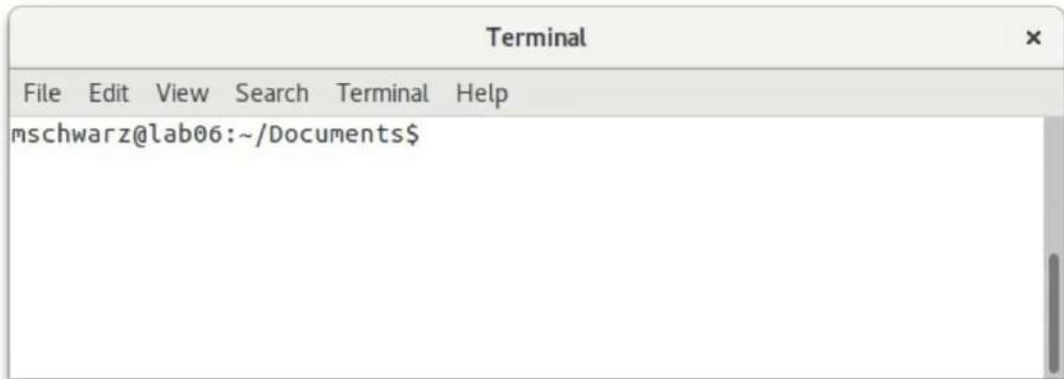
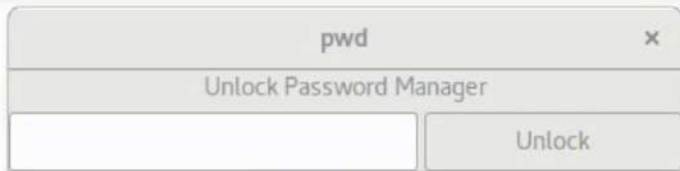
- Flush+Reload over all pages of the array



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

I SHIT YOU NOT

**THERE WAS KERNEL MEMORY ALL
OVER THE TERMINAL**



attacker@meltdown ~/exploit %

victim@meltdown ~ %





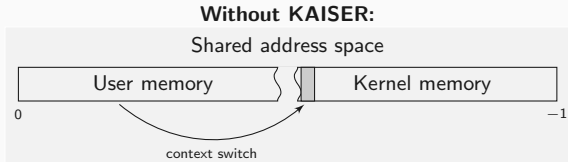
Kernel **A**ddress **I**solation to have **S**ide channels **E**fficiently **R**emoved

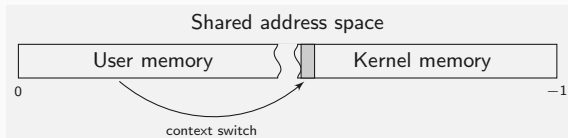
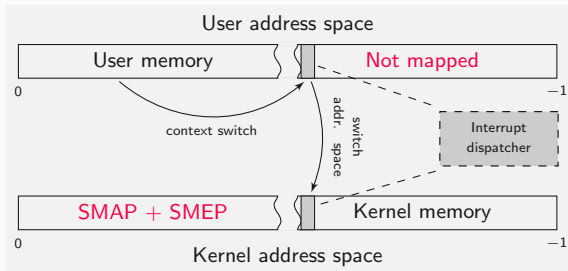
KAISER /'kAɪzə/

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

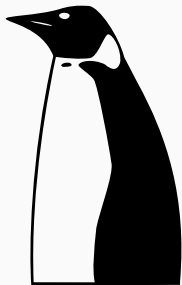


Kernel **A**ddress **I**solation to have **S**ide channels **E**fficiently **R**emoved

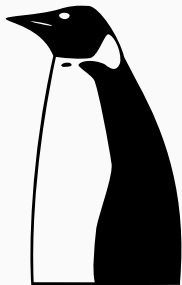


Without KAISER:**With KAISER:**

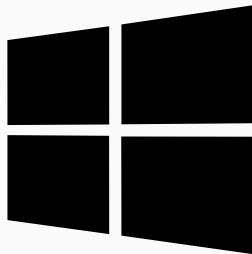




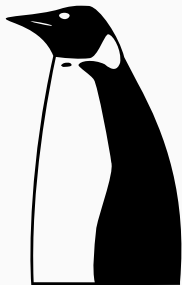
- Our patch
- Adopted in Linux



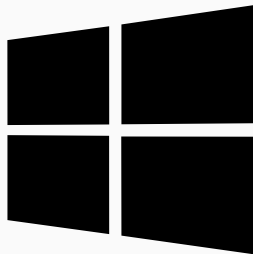
- Our patch
- Adopted in Linux



- Adopted in Windows



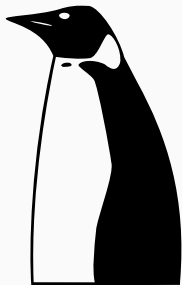
- Our patch
- Adopted in Linux



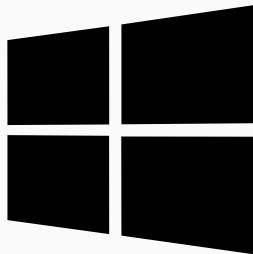
- Adopted in Windows



- Adopted in OSX/iOS



- Our patch
- Adopted in Linux



- Adopted in Windows



- Adopted in OSX/iOS

→ **now in every computer**

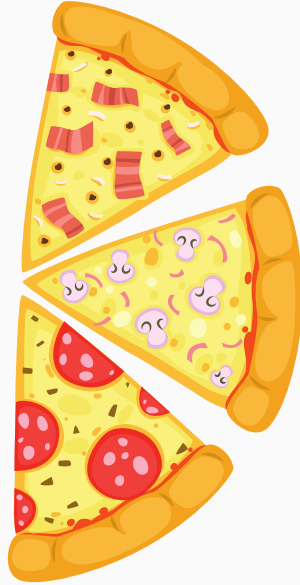


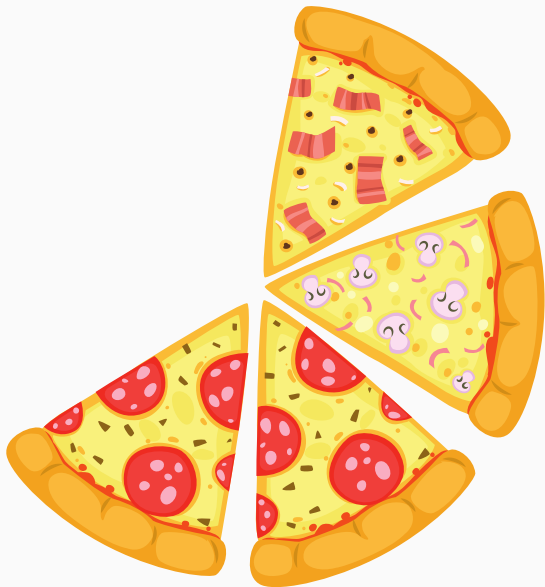
PIZZA

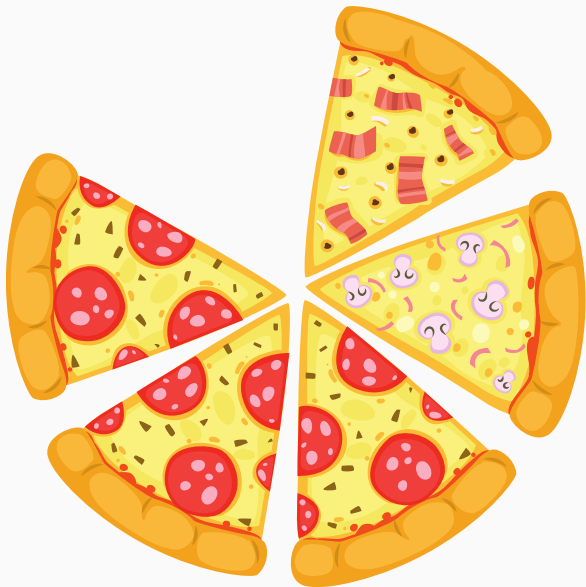
SPECIAL RECIPES

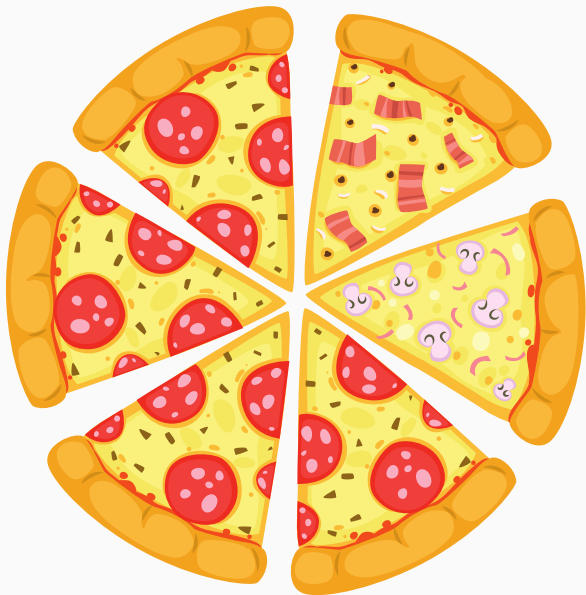




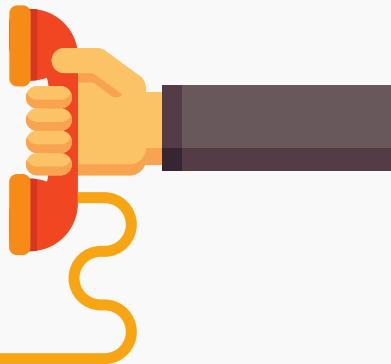








»A table for 6 please«





Speculative Cooking



»A table for 6 please«





PIZZA

SPECIAL RECIPES

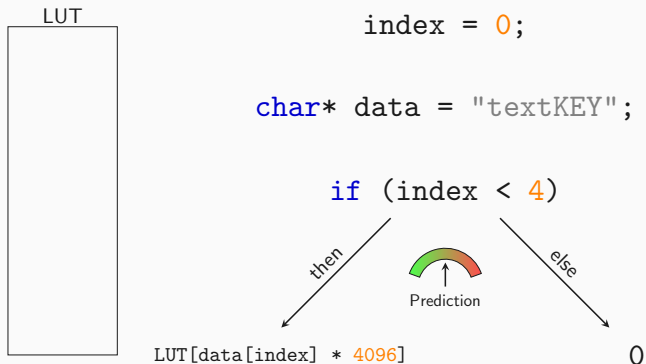


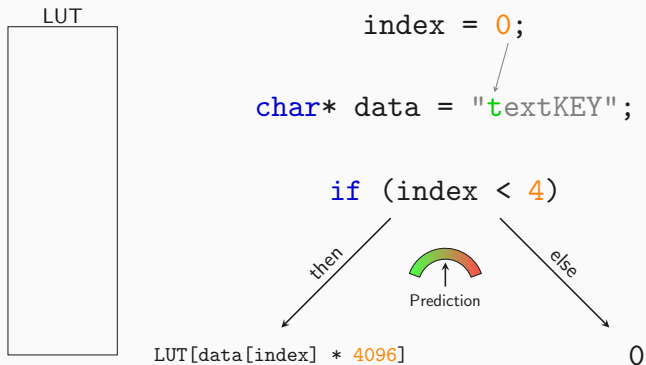
SPECIAL RECIPES

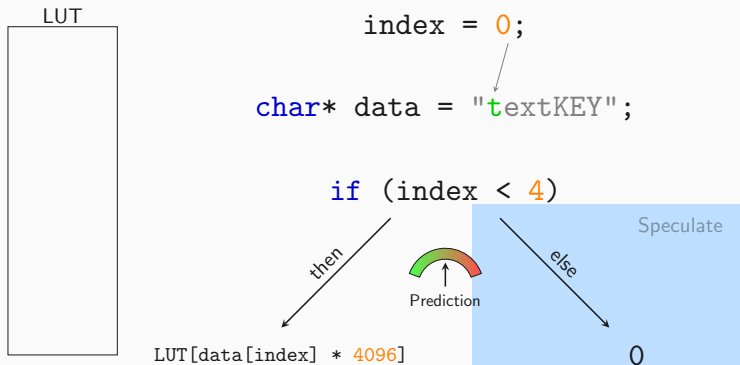


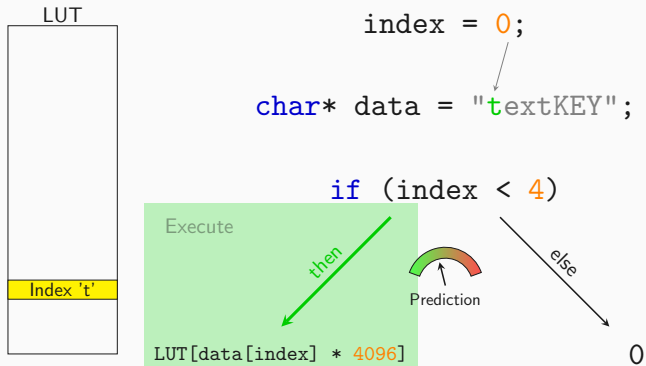


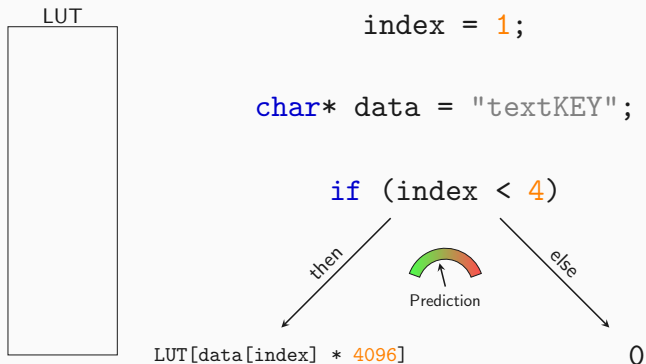


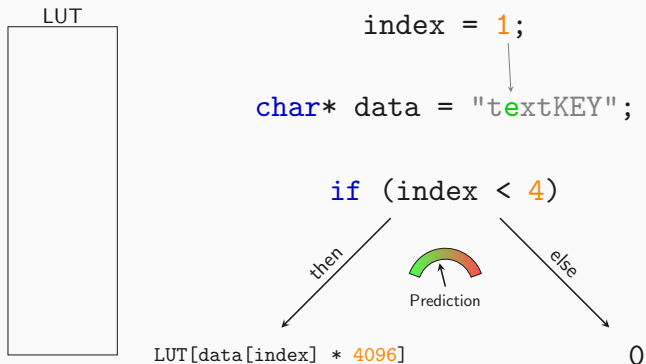


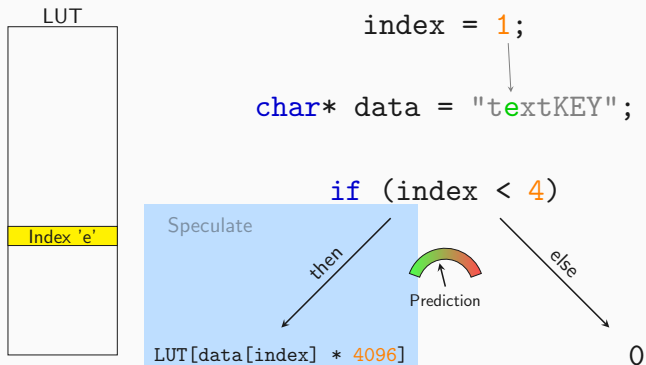


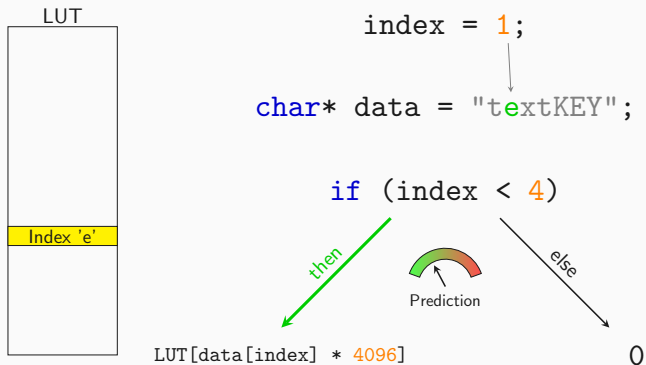


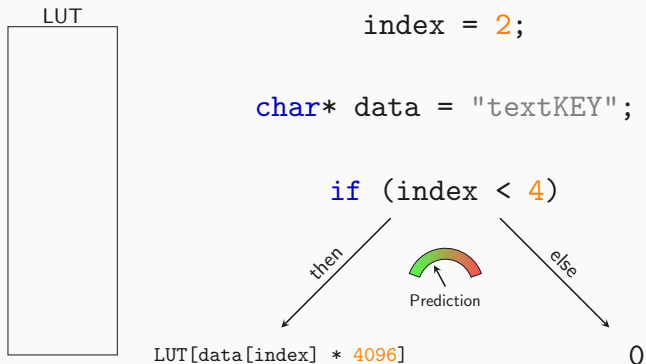


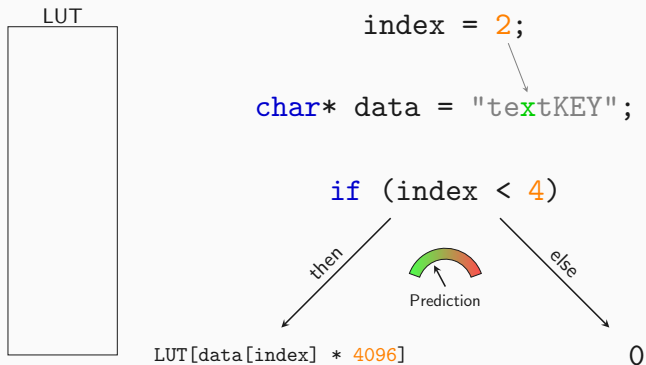


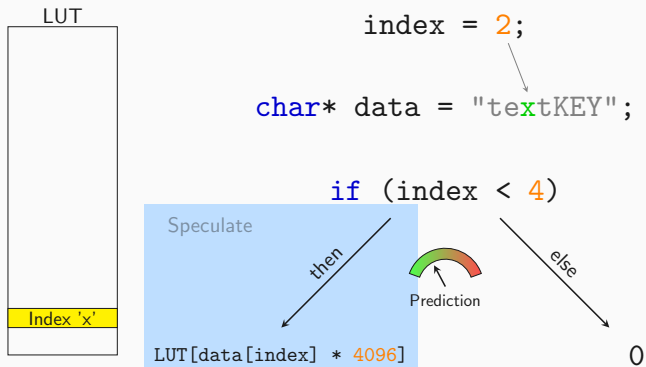


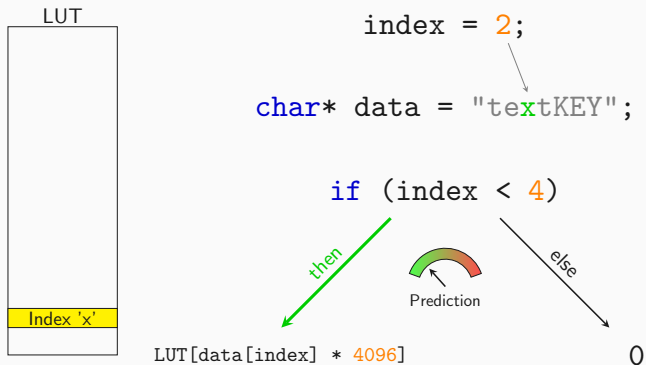


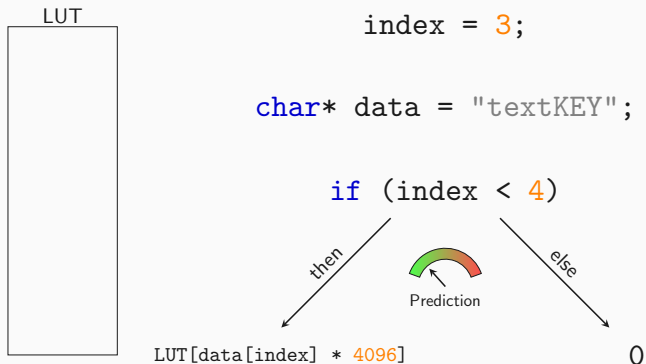


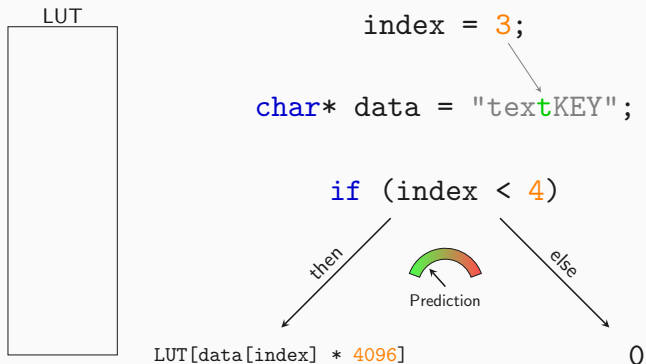


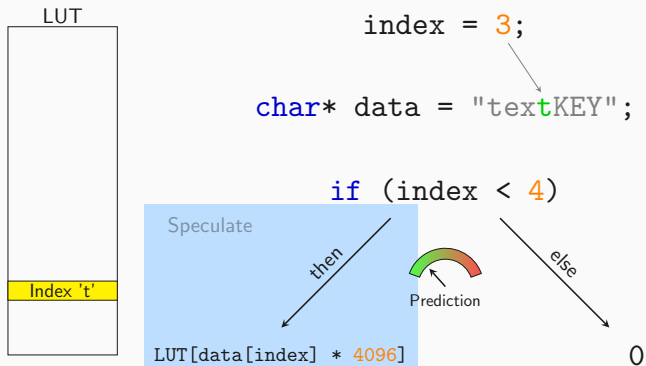


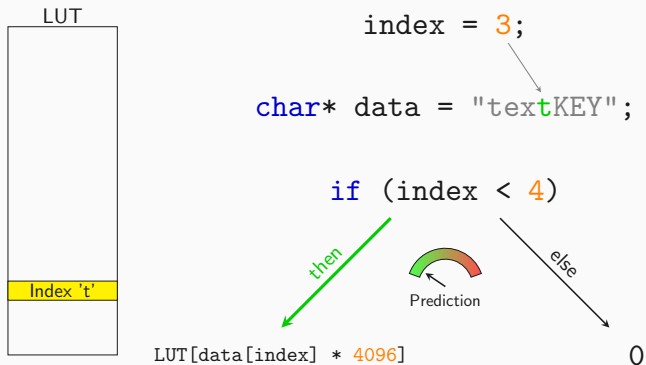


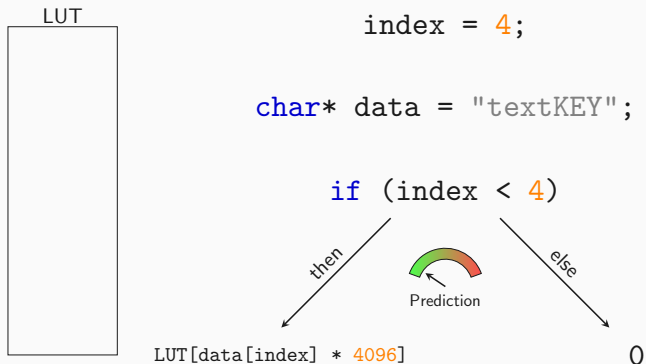


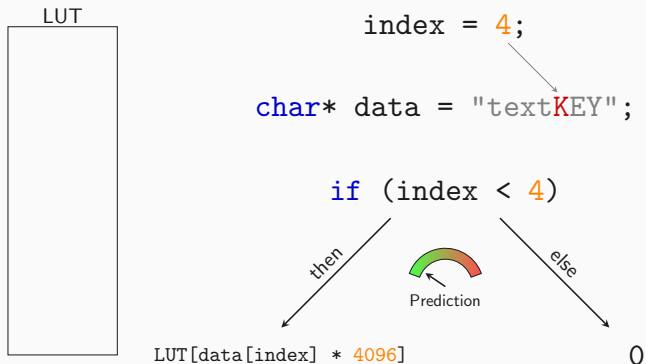


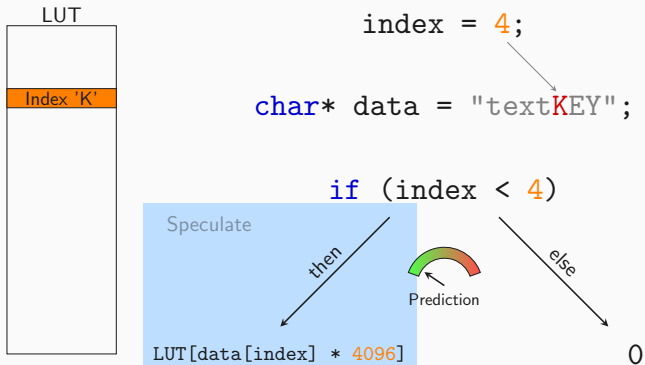


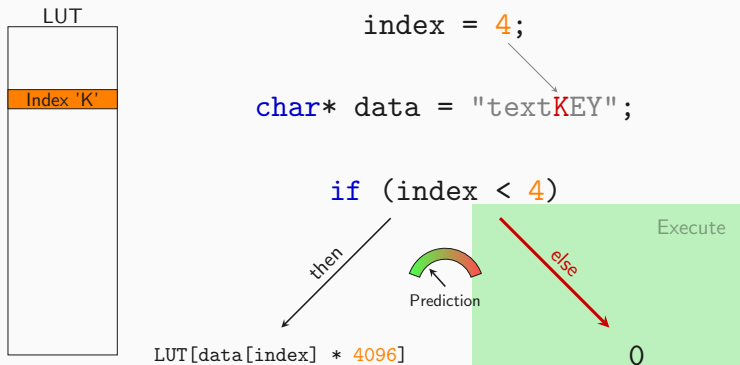


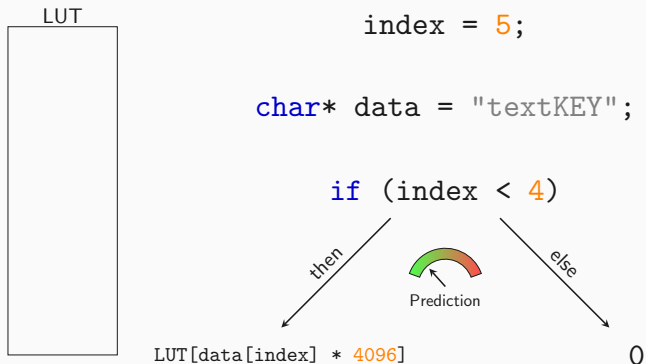


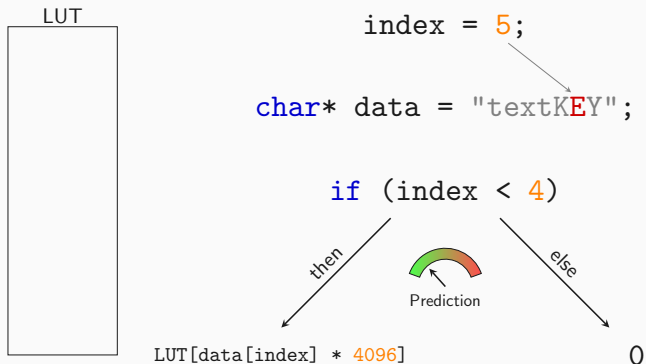


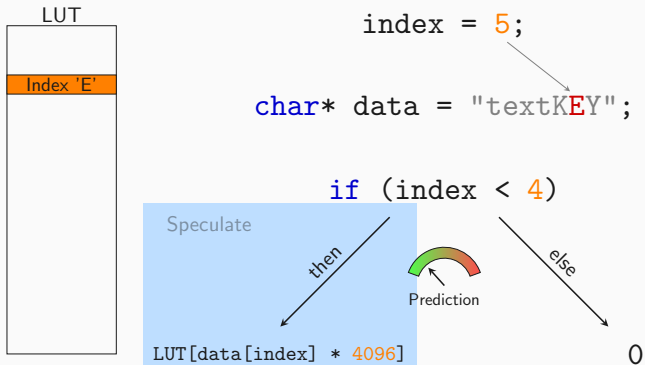


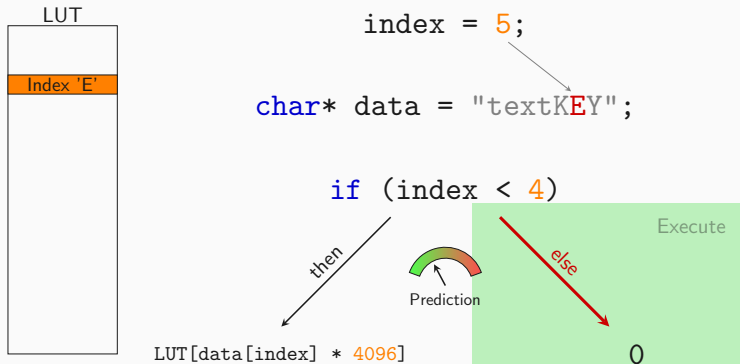


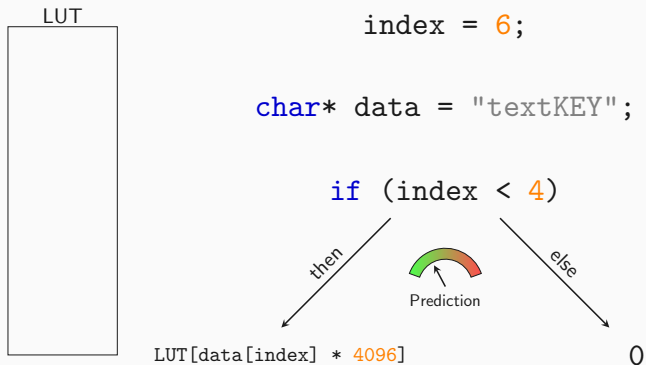


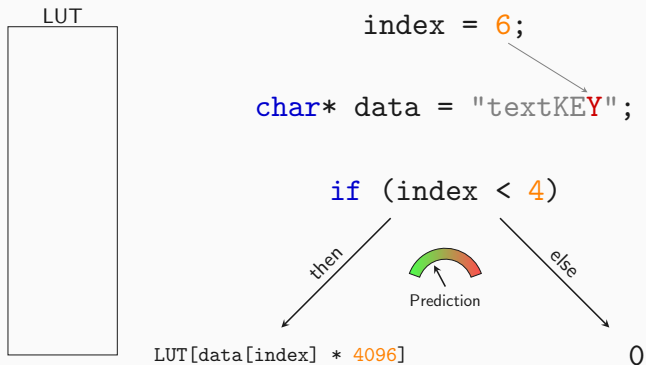


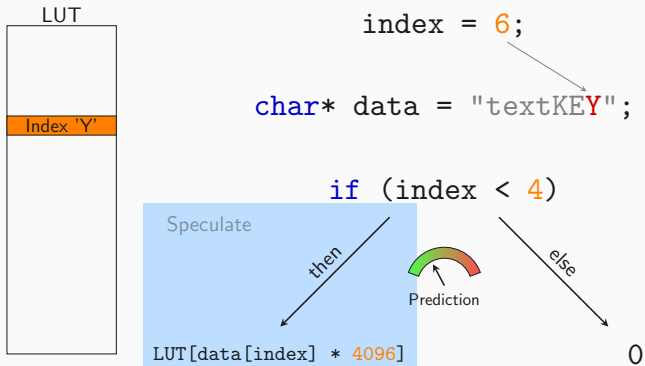


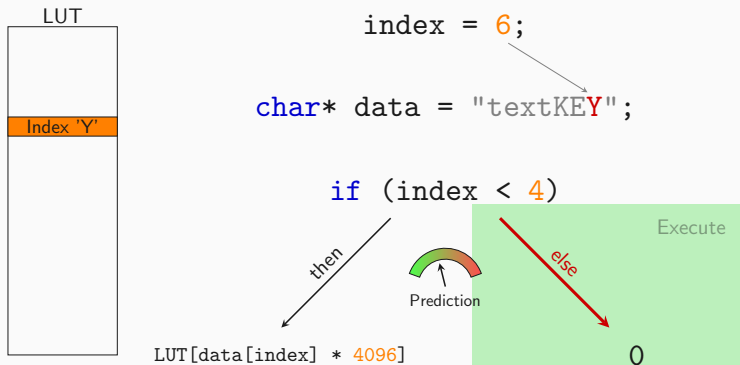




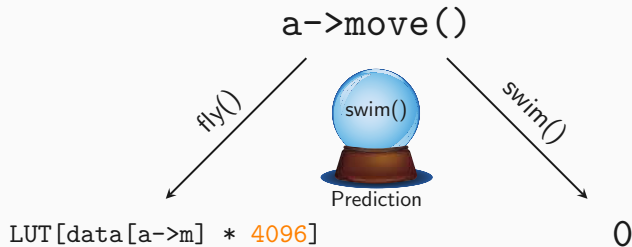




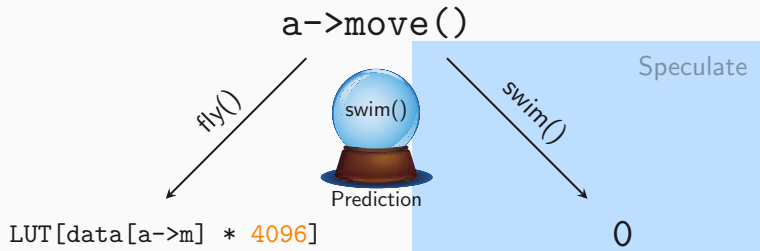




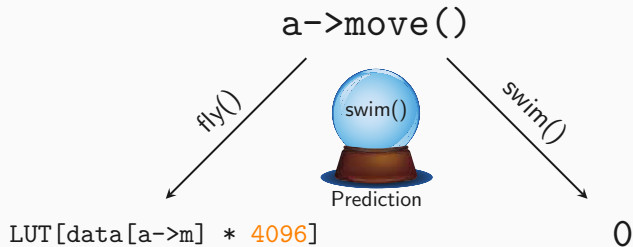
```
Animal* a = bird;
```



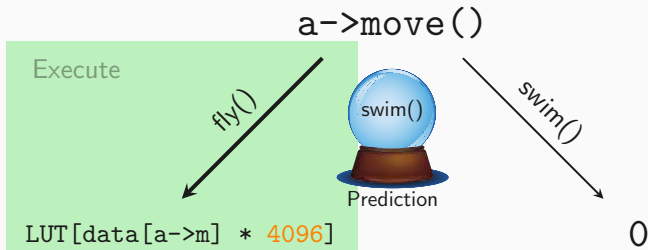
```
Animal* a = bird;
```



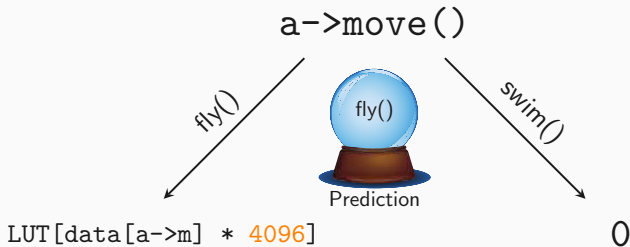
```
Animal* a = bird;
```



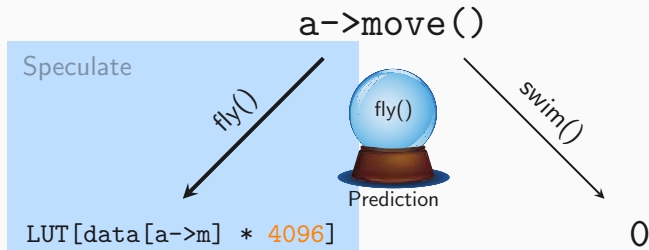
```
Animal* a = bird;
```



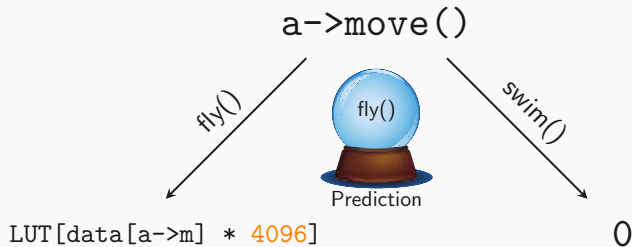
```
Animal* a = bird;
```



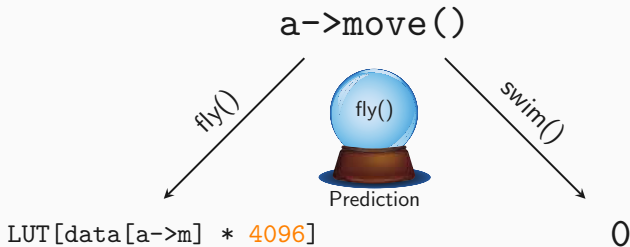
```
Animal* a = bird;
```



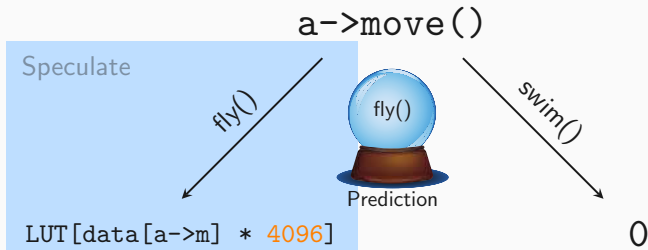
```
Animal* a = bird;
```



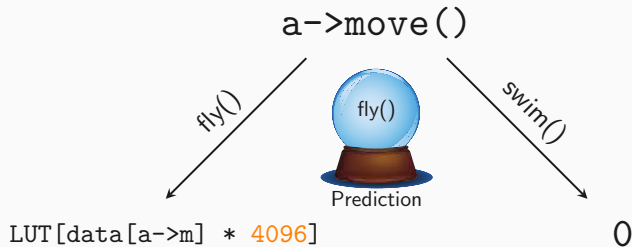
```
Animal* a = fish;
```



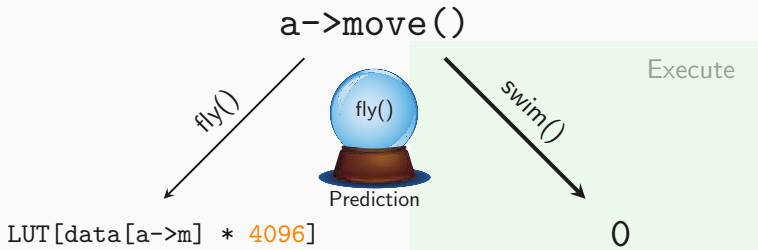
```
Animal* a = fish;
```



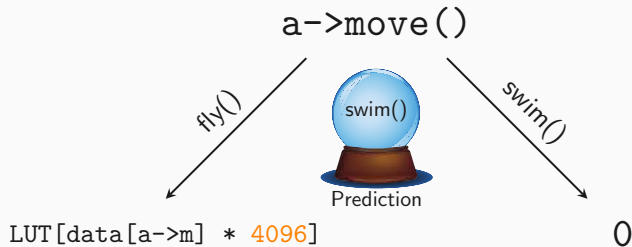
```
Animal* a = fish;
```

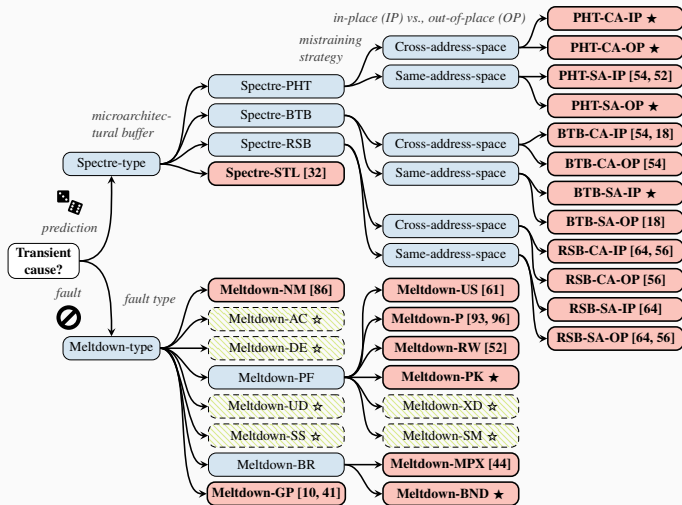


```
Animal* a = fish;
```



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





Mitigations?



BLOCKCHAIN

HHD
HISTORY.COM



Computer Architecture Today

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

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

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



CONTRIBUTE

Editor: Alvin R. Lebeck

Associate Editor: Vijay Janapa Reddi

Contribute to Computer
Architecture Today

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

Attack \ Defense																					
		InvisiSpec	SafeSpec	DAWG	RSB	Stuffing	Retpoline	Poison Value	Index Masking	Site Isolation	SLH	YSNB	IBRS	STIPB	IBPB	Serialization	Taint Tracking	Timer Reduction	Slosh	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																				

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

Table 2: Reported performance impacts of countermeasures

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

How to find the next big thing ;)

they become the target of the doll-maker's possessed creation, Annabelle.

Director: [David F. Sandberg](#) | Stars: [Anthony LaPaglia](#), [Samara Lee](#), [Miranda Otto](#), [Brad Greenquist](#)

Votes: **92,806** | Gross: **\$102.09M**



29. **Zombieland: Double Tap** (2019)

Action, Comedy, Horror | **Post-production**

Columbus, Tallahassee, Wichita, and Little Rock move to the American heartland as they face off against evolved zombies, fellow survivors, and the growing pains of the snarky makeshift family.

Director: [Ruben Fleischer](#) | Stars: [Emma Stone](#), [Zoey Deutch](#), [Woody Harrelson](#), [Abigail Breslin](#)



30. **Love, Death & Robots** (2019-)

TV-MA | 15 min | Animation, Short, Comedy

★ **8.7** ☆ [Rate this](#)

A collection of animated short stories that span various genres including science fiction, fantasy, horror and comedy.

Stars: [Scott Whyte](#), [Nolan North](#), [Matthew Yang King](#), [Michael Benyaer](#)

Votes: **58,780**



31. **iZombie** (2015-)

TV-14 | 42 min | Comedy, Crime, Drama

★ **7.9** ☆ [Rate this](#)

A medical resident finds that being a zombie has its perks, which she uses to assist the police.

Stars: [Rose McIver](#), [Malcolm Goodwin](#), [Rahul Kohli](#), [Robert Buckley](#)

Votes: **54,215**

WOODY HARRELSON

**JESSE
EISENBERG**

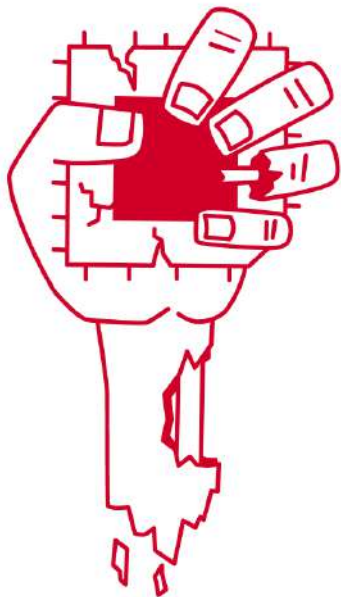
EMMA
STONE

**ANIGAIL
BRESLIN**

WOODY HARRELSON EISENBERG
ZOMBIELAND
NUT UP OR SHUT UP

NUT UP OR SHUT UP

[illegible]



ZOMBIELOAD ATTACK

https://meltdownattack.com/meltdown

8 of 18 150%

When the kernel address is loaded in line 4, that the CPU already issued the subsequent instructions as part of the out-of-order execution, and that the corresponding μ OPs wait in the reservation station for the content of the kernel address to arrive. As s



fault occurs load operation completed? "intel corp"



Alle



News



Bilder



Shopping



Videos



Mehr

Einstellungen

Tools

Ungefähr 111 000 Ergebnisse (0,42 Sekunden)

Toshiba Boot Error - TechRepublic

<https://www.techrepublic.com/.../toshiba-boot-error/> ▾ Diese Seite übersetzen

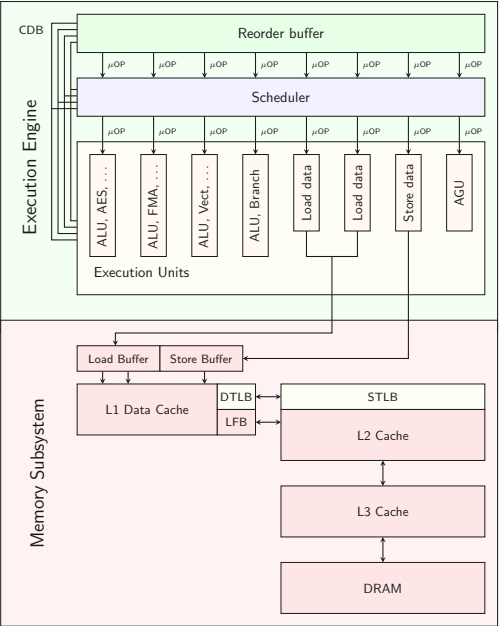
19.05.2007 - by CaptBilly1Eye · 12 years ago In reply to Toshiba Boot Error ... partition on the floppy disk, hard drive or a CD ROM to load the **operating** system. ... prior to this situation starting to **occur**, or if you find that the boot sequence already has the ... Leave the notebook plugged in and undisturbed until **completed**.

US5751983A - Out-of-order processor with a memory ...

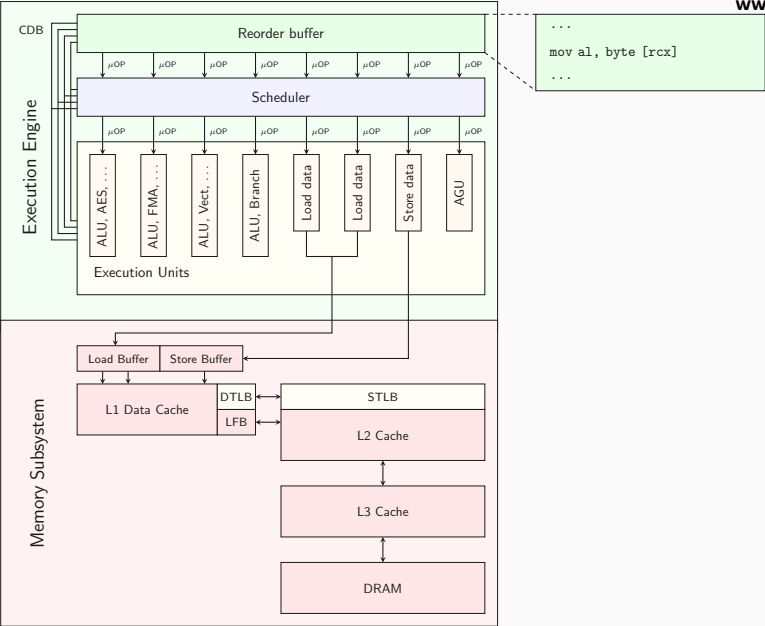
www.google.com/patents/US5751983 - Diese Seite übersetzen

Application filed by Intel Corp ... Hence, a functional unit may often **complete** a first instruction (which logically precedes a second instruction in the If a **fault occurs** with respect to the **LOAD operation**, it is marked as valid and **completed**.

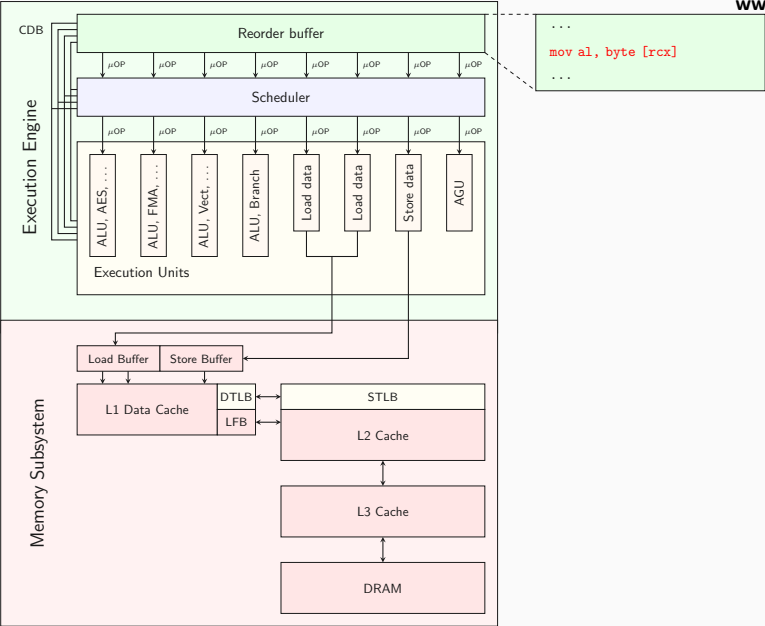
Meltdown



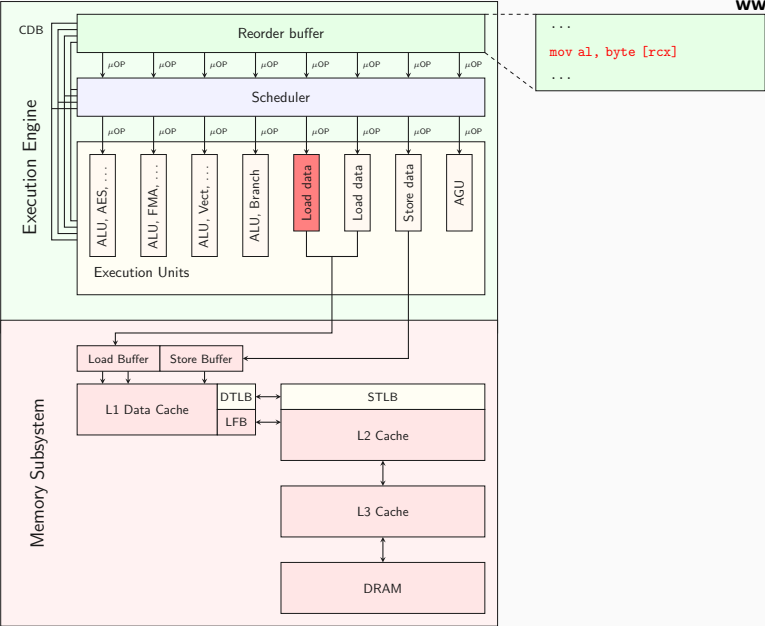
Meltdown



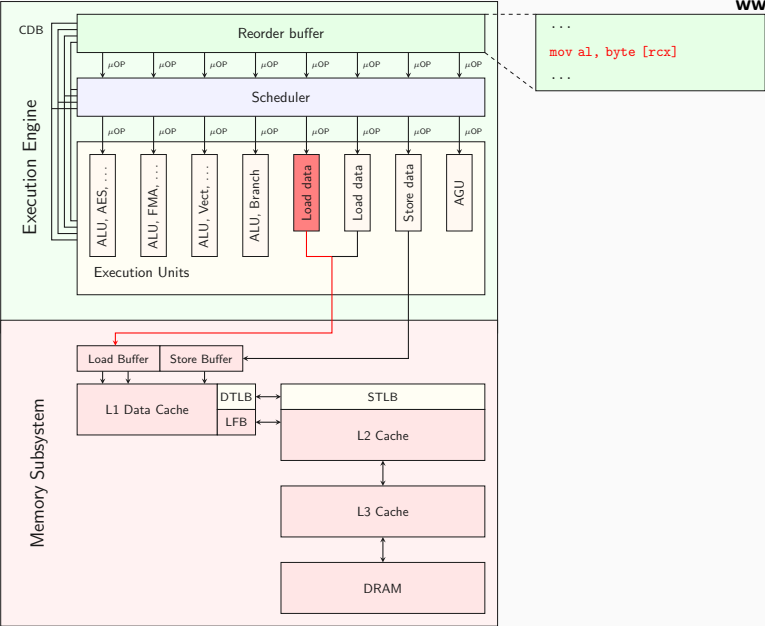
Meltdown



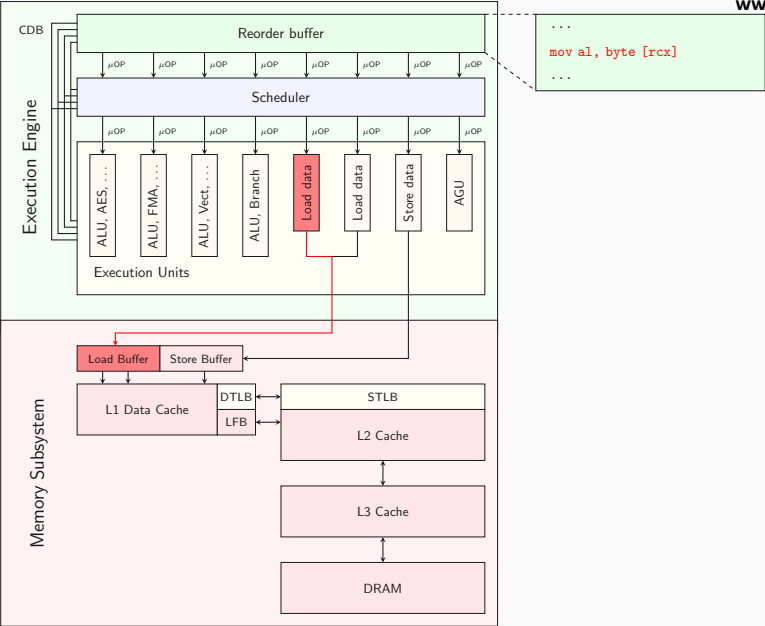
Meltdown



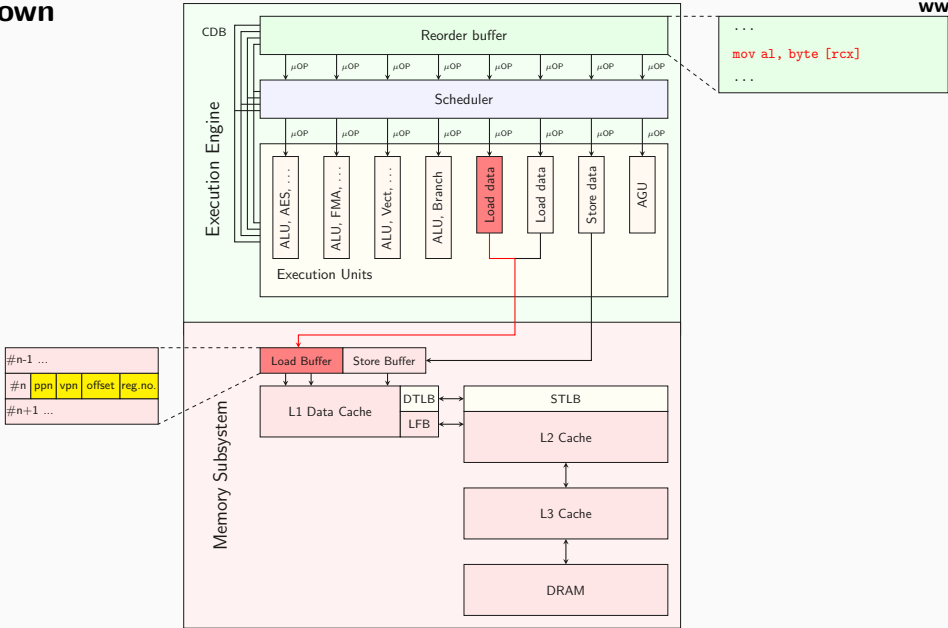
Meltdown



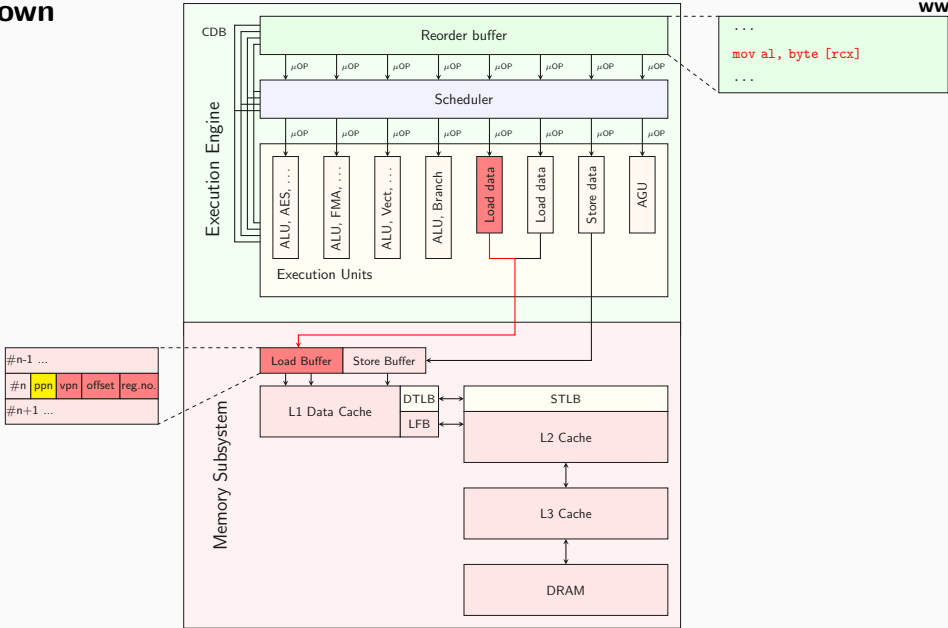
Meltdown



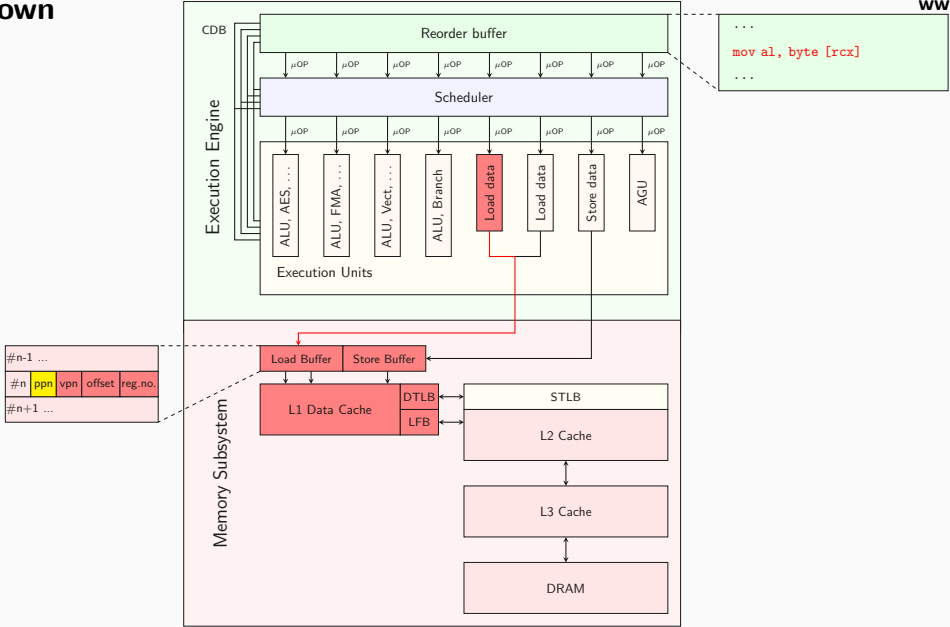
Meltdown



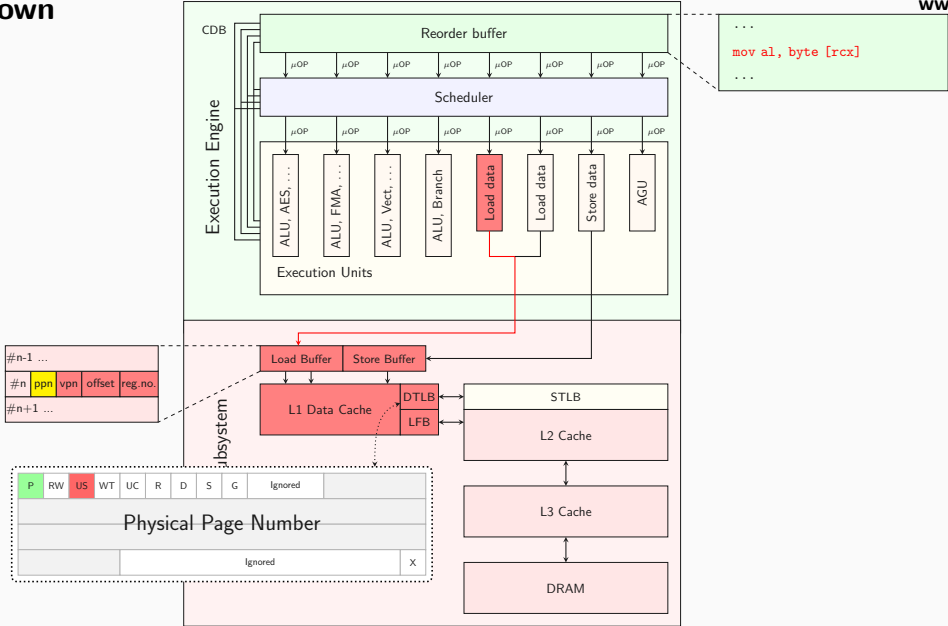
Meltdown



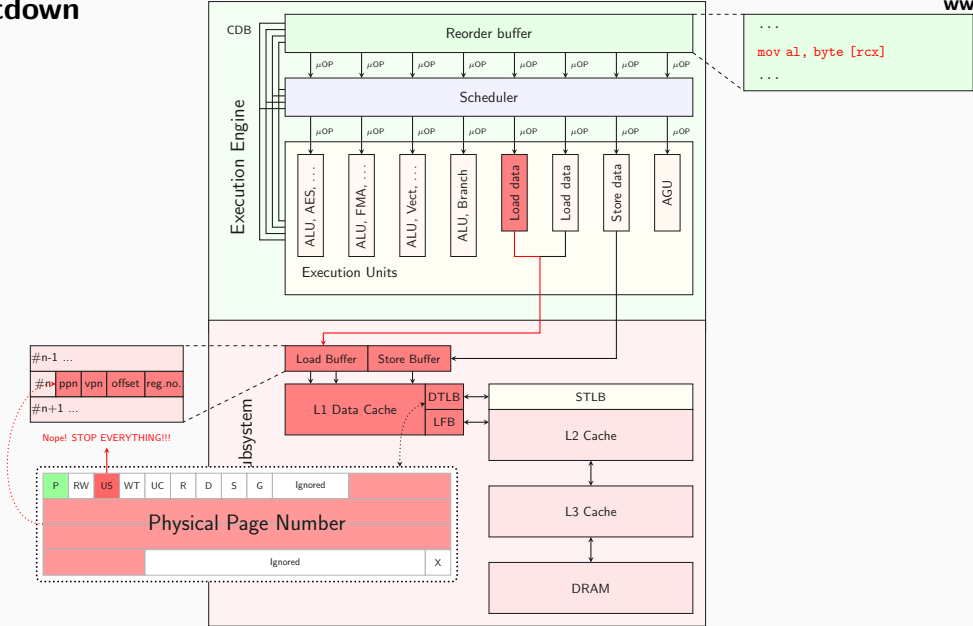
Meltdown



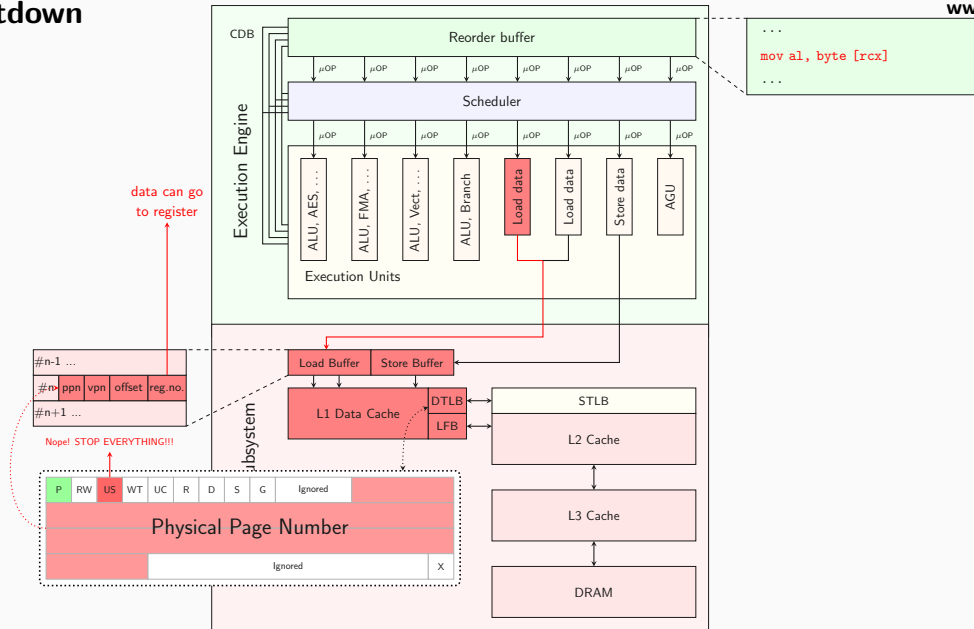
Meltdown

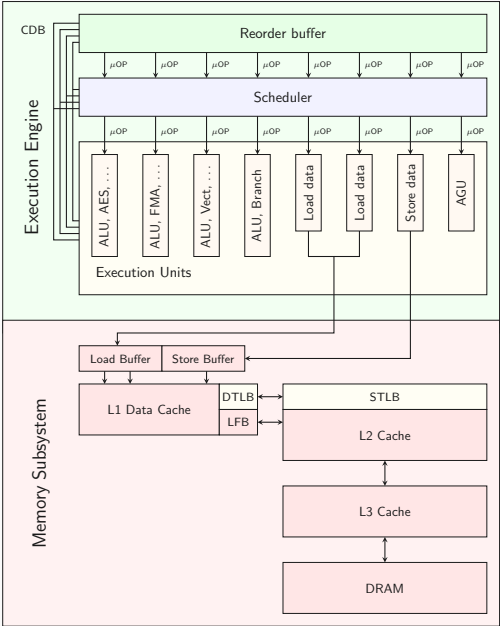


Meltdown

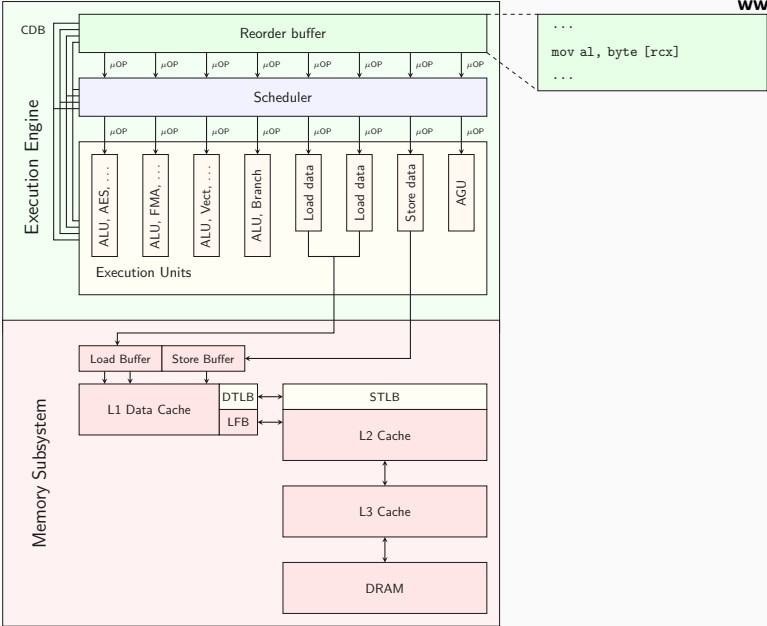


www.tugraz.at ■

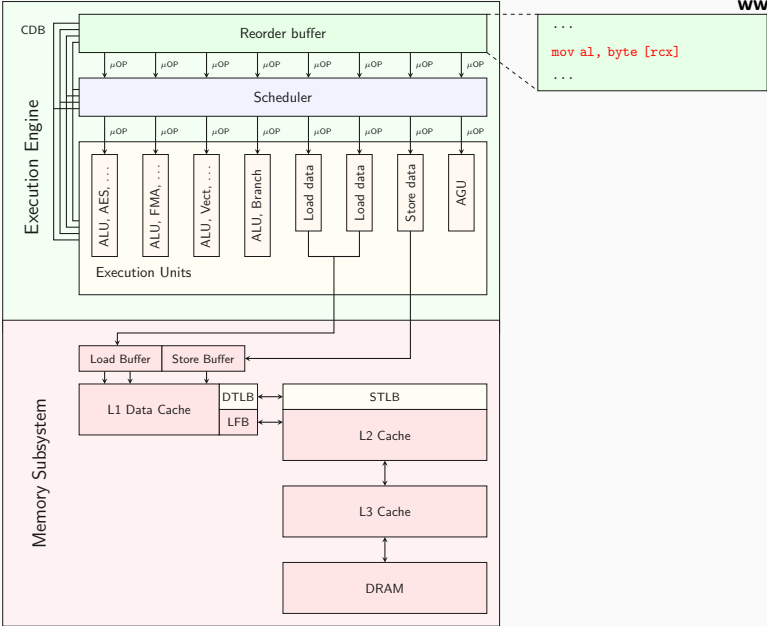




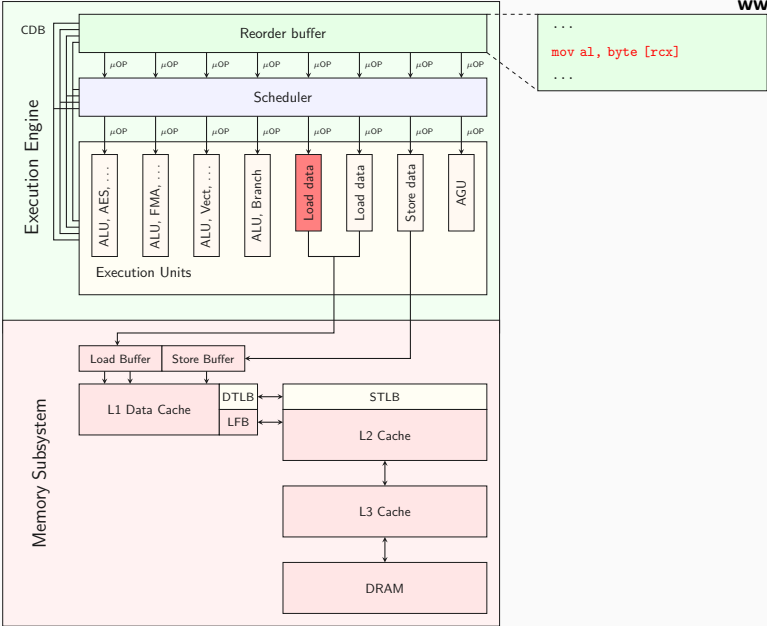
Foreshadow-VMM



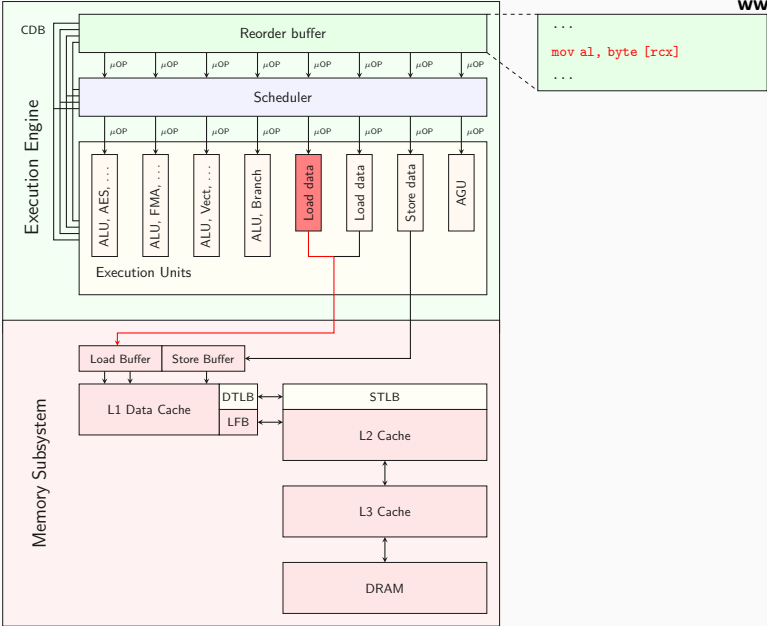
Foreshadow-VMM



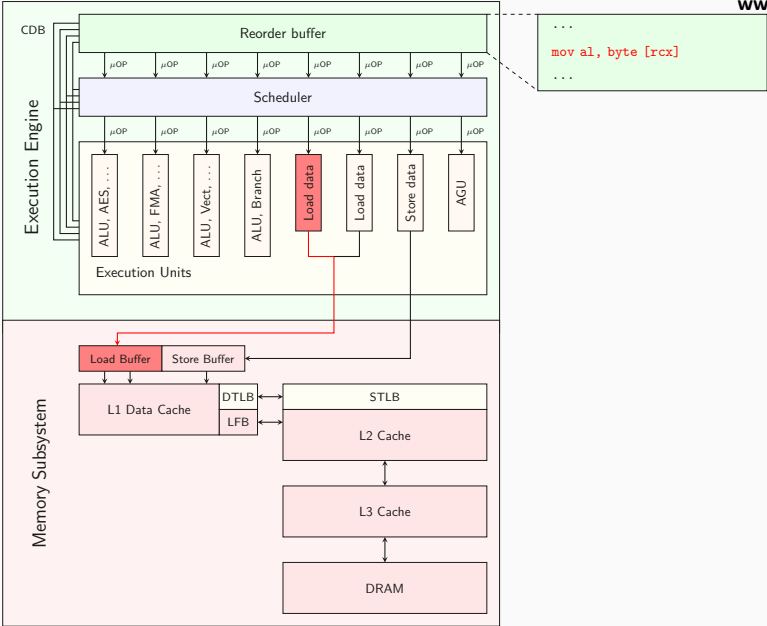
Foreshadow-VMM



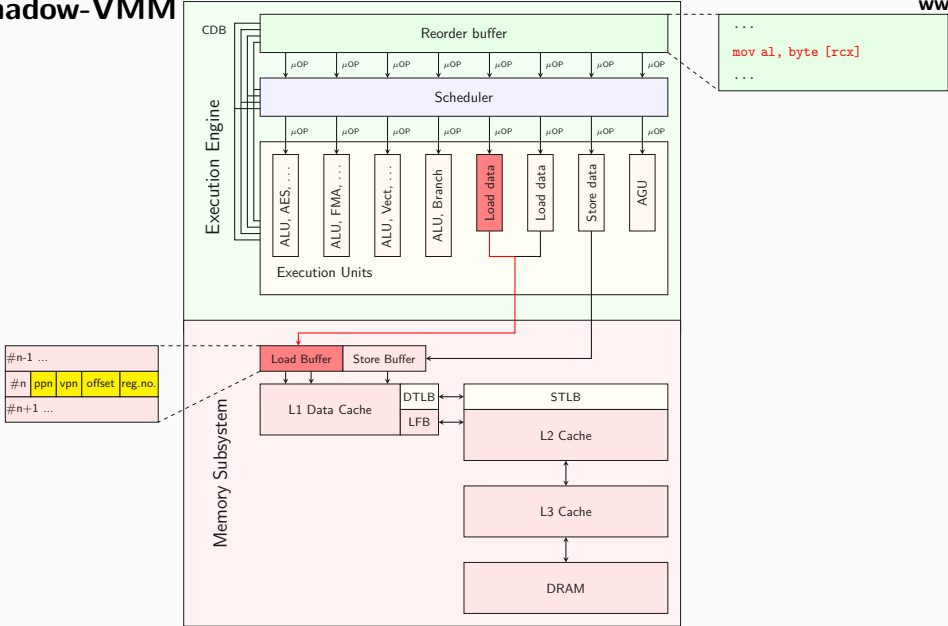
Foreshadow-VMM



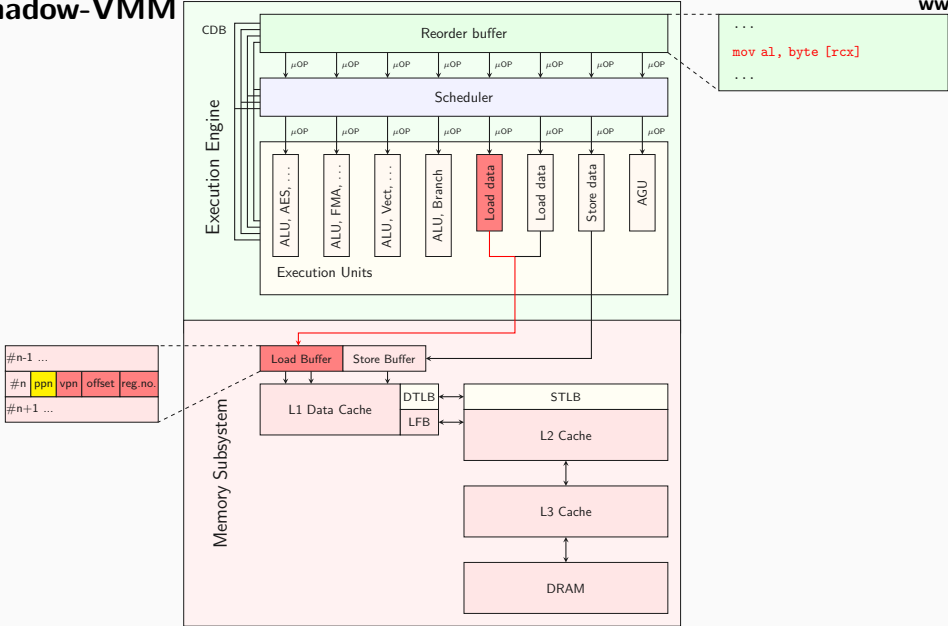
Foreshadow-VMM



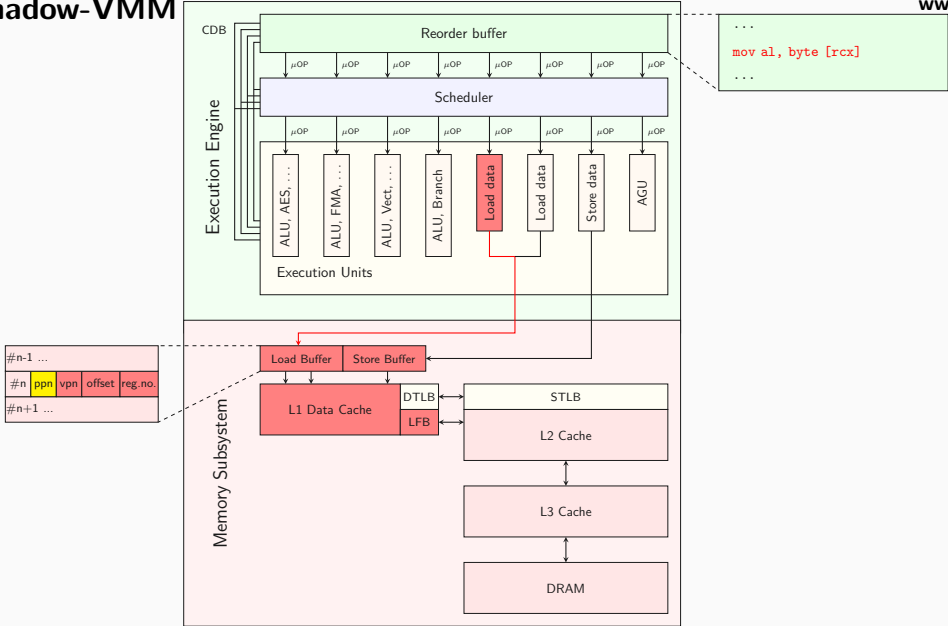
Foreshadow-VMM



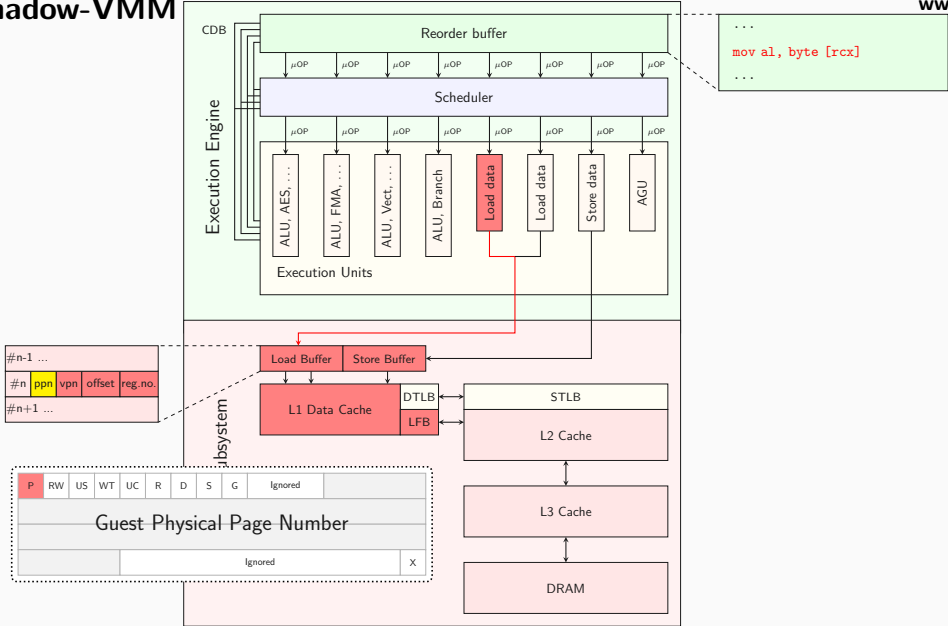
Foreshadow-VMM



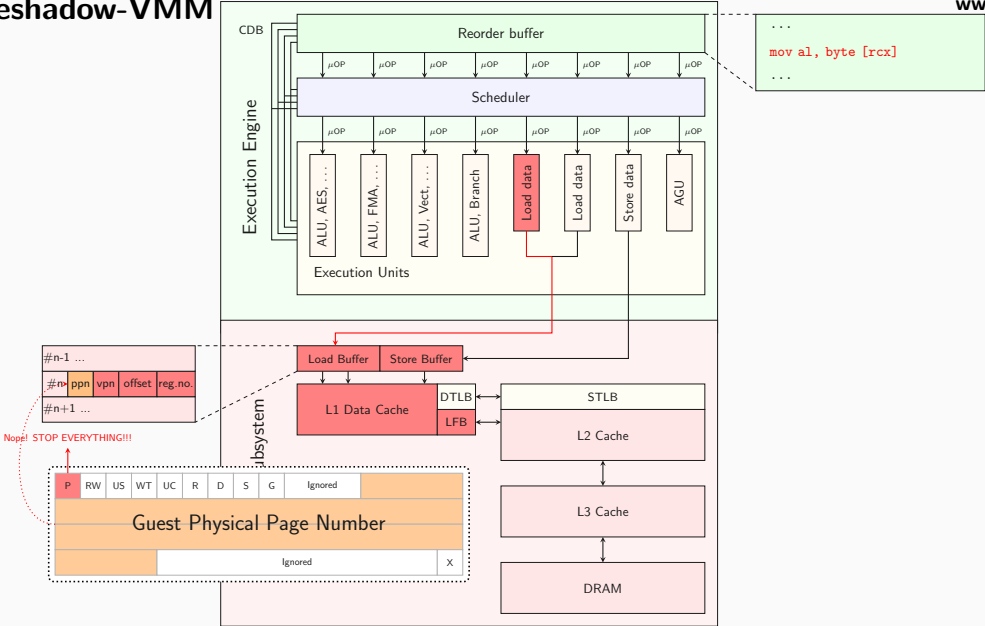
Foreshadow-VMM



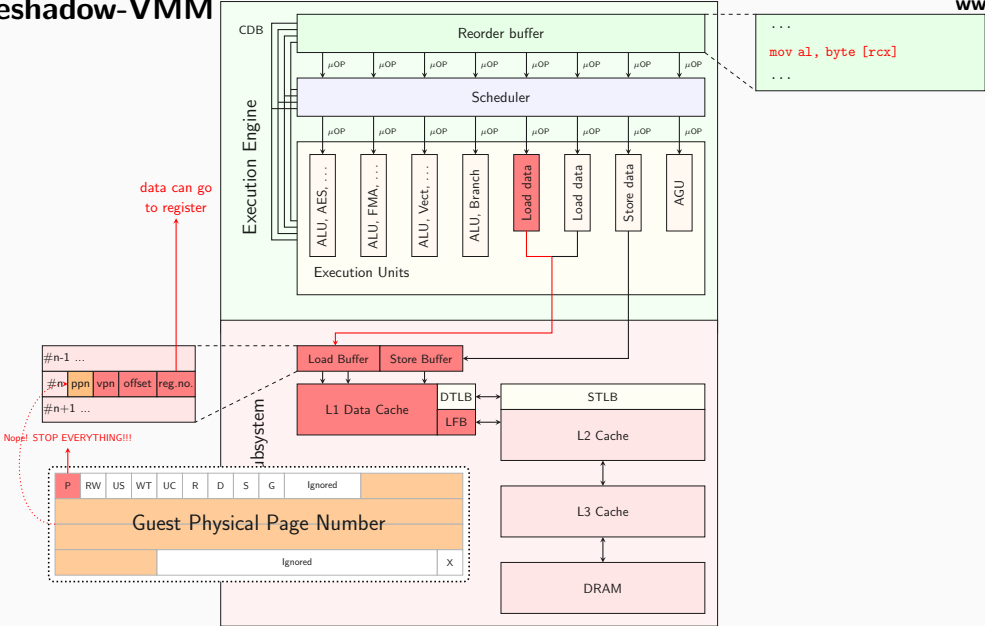
Foreshadow-VMM

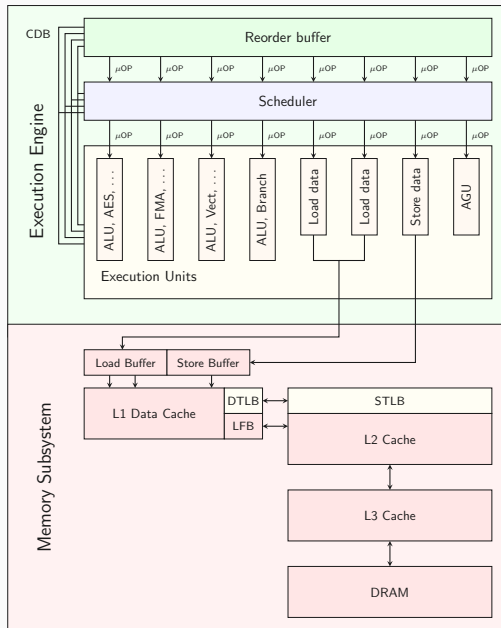


Foreshadow-VMM

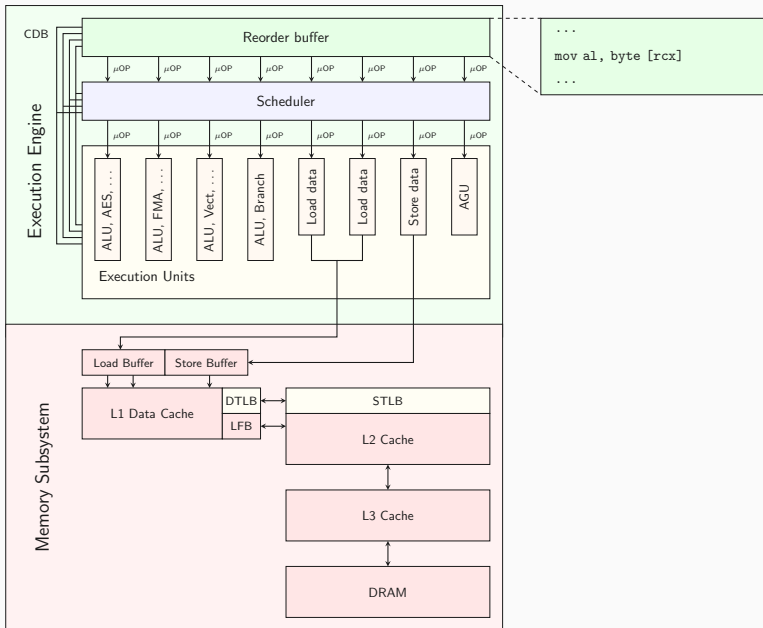


Foreshadow-VMM

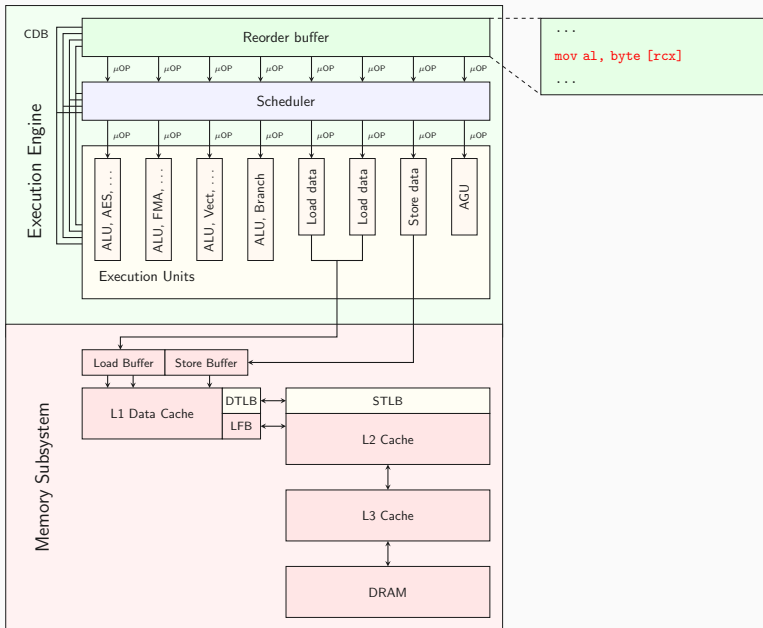




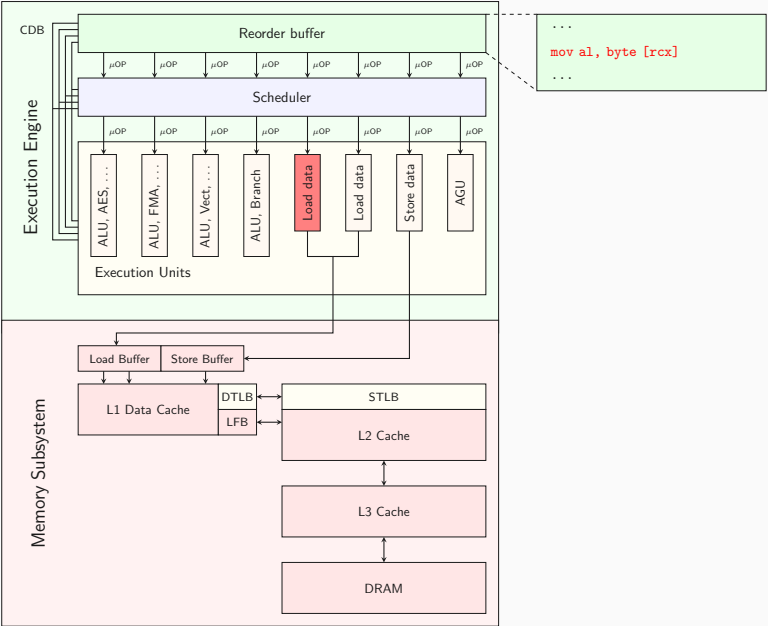
ZombieLoad



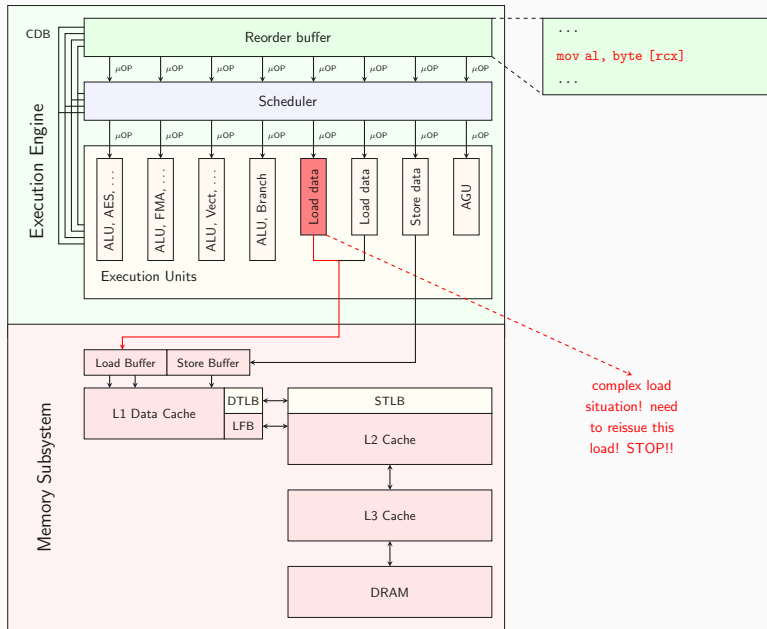
ZombieLoad



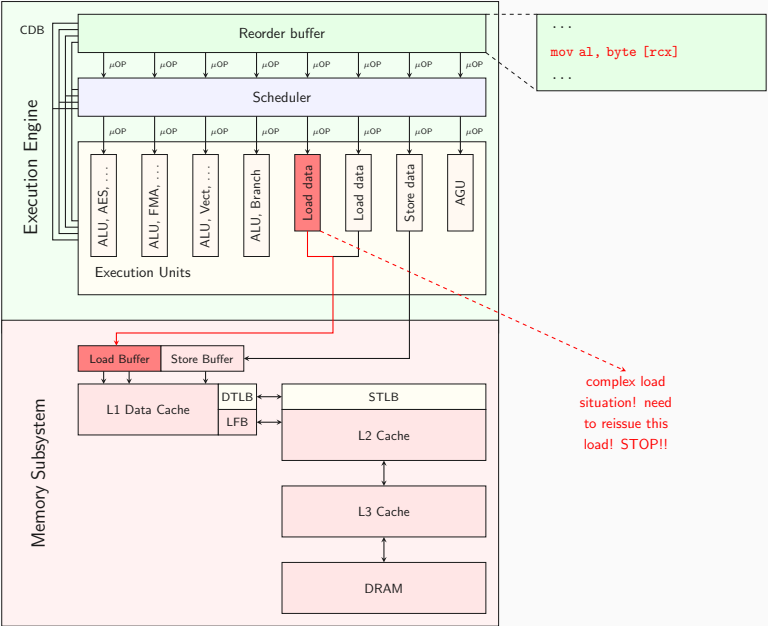
ZombieLoad



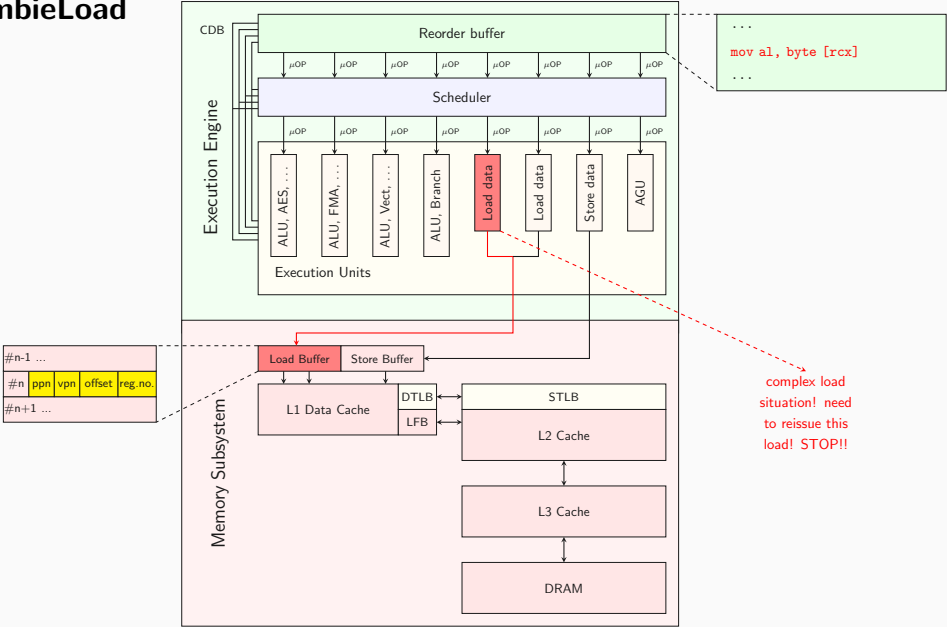
ZombieLoad



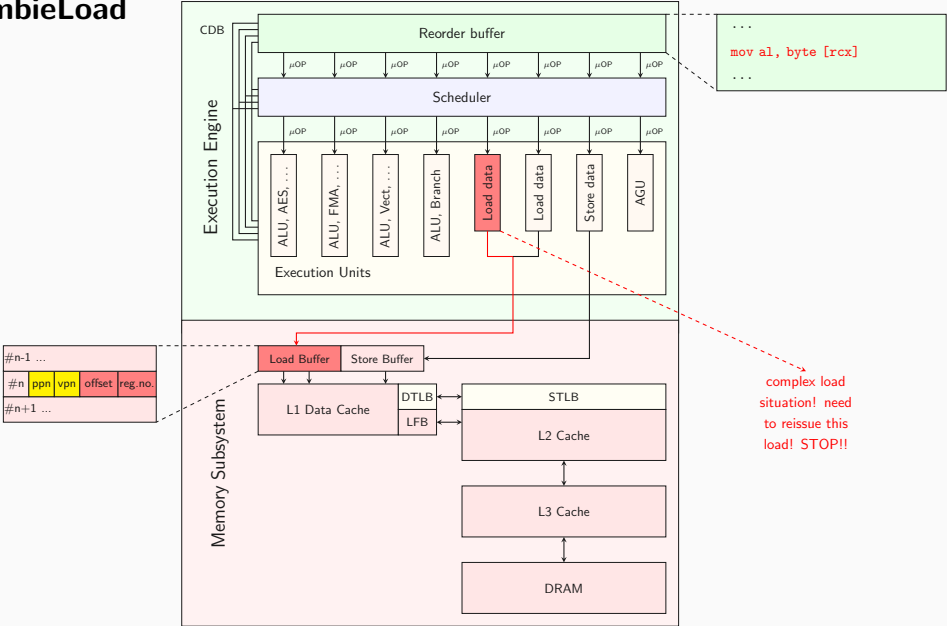
ZombieLoad



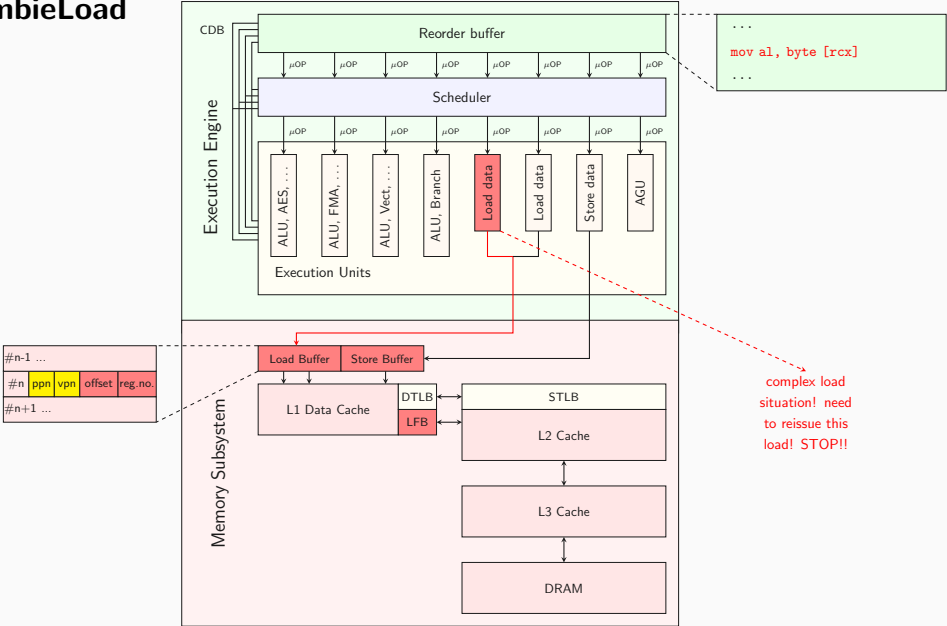
ZombieLoad



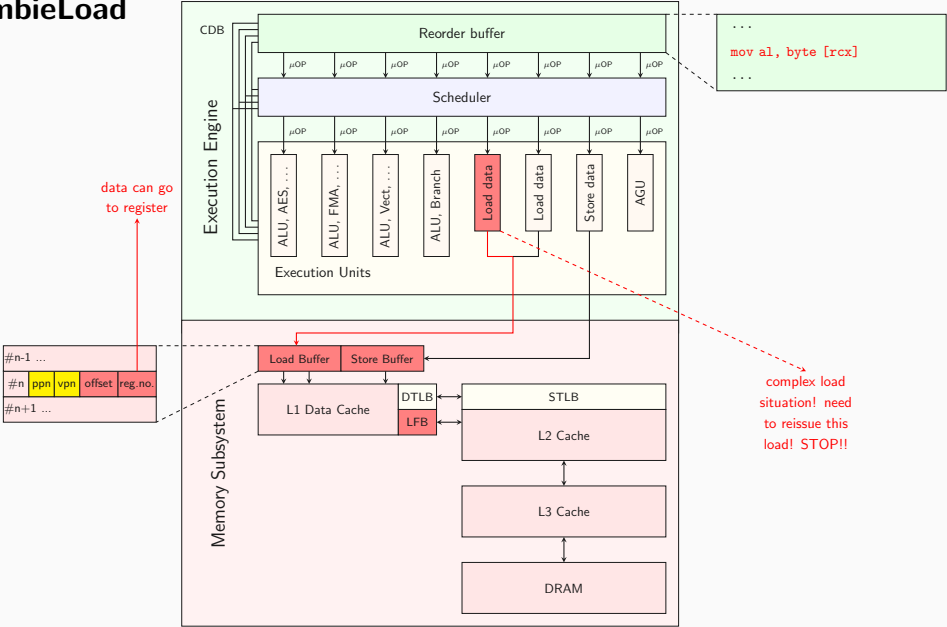
ZombieLoad



ZombieLoad



ZombieLoad



1

zombieload : zsh — Konsole #2



File Edit View Bookmarks Settings Help

michael@hp /tmp/zombieload %

1

zombieload : zsh

1

zombieload : zsh — Konsole #2



File Edit View Bookmarks Settings Help

michael@hp /tmp/zombieload %

1

zombieload : zsh





- “You should work more on defenses than on attacks”



- “You should work more on defenses than on attacks”
- **dangerous**: we overlooked Meltdown and Spectre for decades



- “You should work more on defenses than on attacks”
- **dangerous**: we overlooked Meltdown and Spectre for decades
- do we even know the most important problems?



- “You should work more on defenses than on attacks”
- **dangerous**: we overlooked Meltdown and Spectre for decades
- do we even know the most important problems?
- are we *hammering* on solutions and miss the bigger picture?







Computer science:



Computer science:

- computer engineering



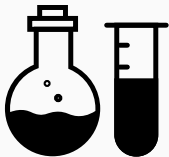
Computer science:

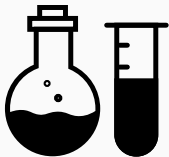
- computer engineering
- philosophy



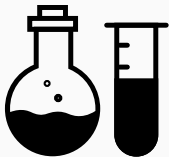
Computer science:

- computer engineering
- philosophy
- artificial science



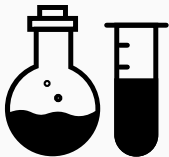


“The Sciences of the Artificial” (Herbert A. Simon, 1969)



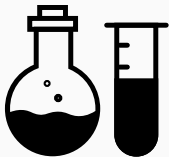
“The Sciences of the Artificial” (Herbert A. Simon, 1969)

- Natural sciences: studying something natural



“The Sciences of the Artificial” (Herbert A. Simon, 1969)

- Natural sciences: studying something natural
- Artificial science: studying something artificial (something human-made) **as if it was something natural**

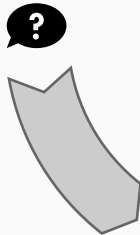


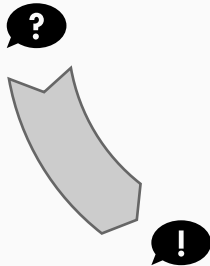
“The Sciences of the Artificial” (Herbert A. Simon, 1969)

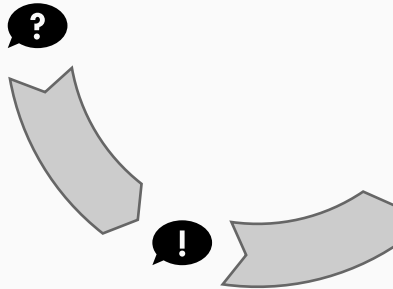
- Natural sciences: studying something natural
- Artificial science: studying something artificial (something human-made) **as if it was something natural**

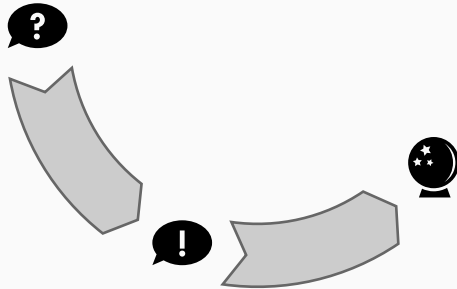
→ A consequence of **complexity**

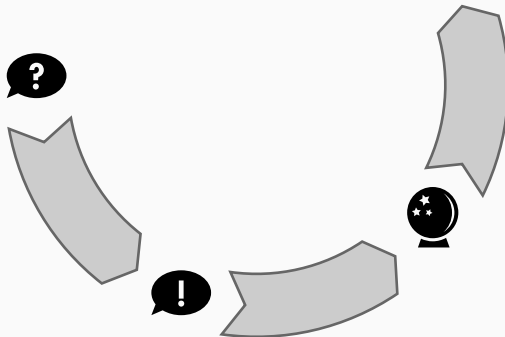


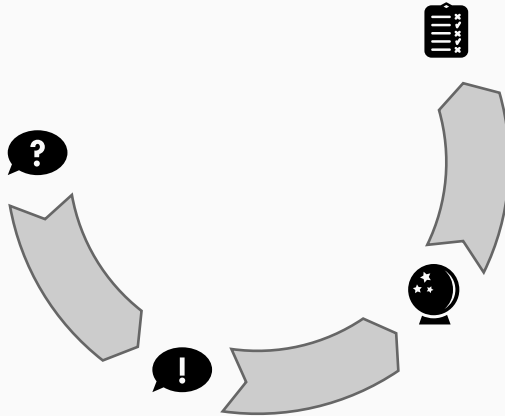


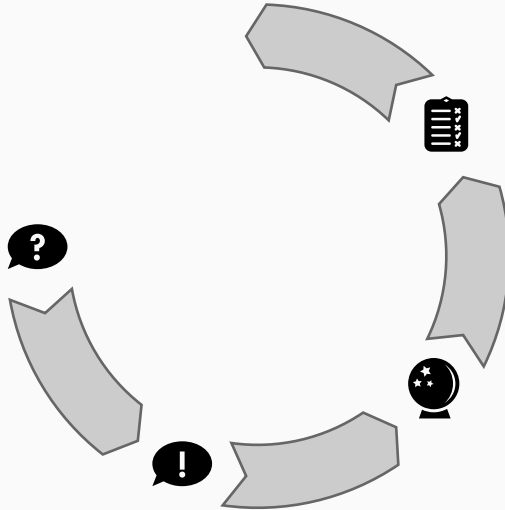


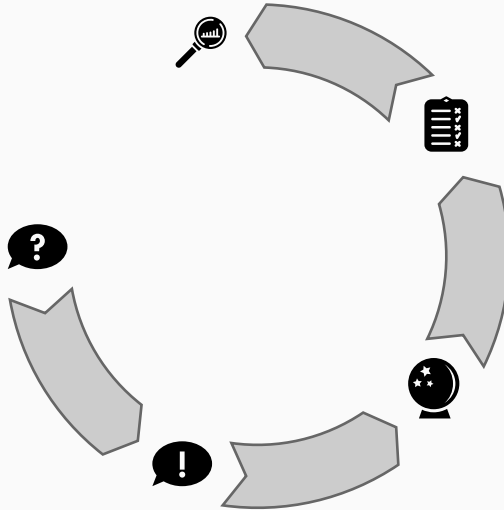


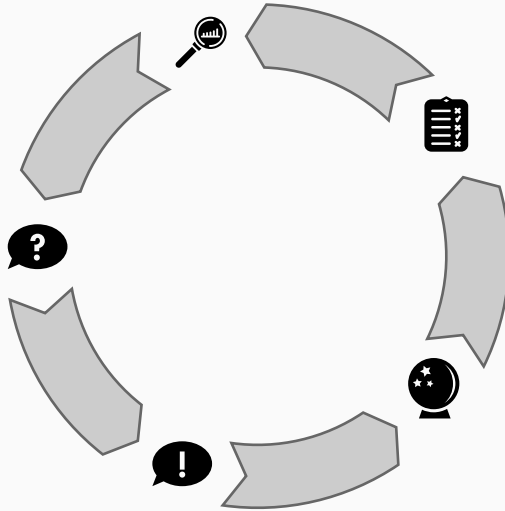


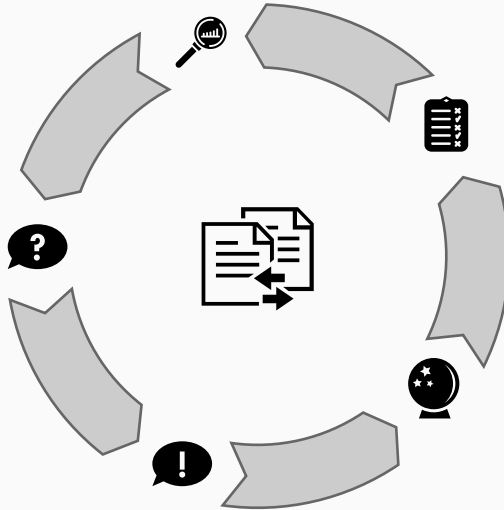












We have ignored microarchitectural attacks for many years:





We have ignored microarchitectural attacks for many years:

- attacks on crypto



We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”



We have ignored microarchitectural attacks for many years:

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



We have ignored microarchitectural attacks for many years:

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



We have ignored microarchitectural attacks for many years:

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



We have ignored microarchitectural attacks for many years:

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



We have ignored microarchitectural attacks for many years:

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



We have ignored microarchitectural attacks for many years:

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



We have ignored microarchitectural attacks for many years:

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

→ for years we solely optimized for performance



- lower refresh rate = lower energy but more bit flips



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



- lower refresh rate = lower energy but more bit flips
 - ECC memory → fewer bit flips
- it's an optimization problem



- lower refresh rate = lower energy but more bit flips
 - ECC memory → fewer bit flips
- it's an optimization problem
- what if “too aggressive” changes over time?



- lower refresh rate = lower energy but more bit flips
 - ECC memory → fewer bit flips
- it's an optimization problem
- what if “too aggressive” changes over time?
- difficult to optimize with an intelligent adversary



- new class of software-based attacks



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



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

Side Channels and Transient Execution Attacks

Daniel Gruss

July 30, 2019

Graz University of Technology

References



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



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