Transient Execution Attacks

Daniel Gruss

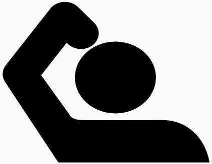
July 11, 2019

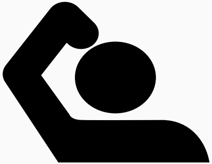
Graz University of Technology


amazon.com
Prime+Probe
ROWHAMMER.JS
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

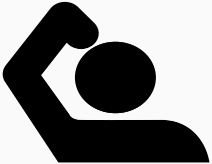

side channel
= obtaining meta-data and
deriving secrets from it
CHANGE MY MIND

Side Channel or not?



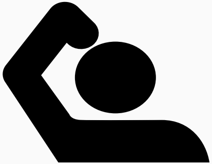


- Profiling cache utilization with performance counters?

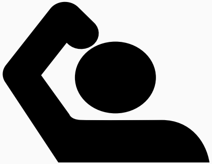


- Profiling cache utilization with performance counters? $\rightarrow$ No

Side Channel or not?

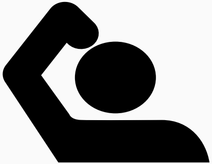


- Profiling cache utilization with performance counters? → No
- Observing cache utilization with performance counters and using it to infer a crypto key?

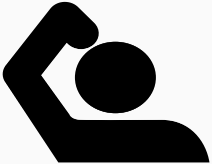


- Profiling cache utilization with performance counters? $\rightarrow$ No
- Observing cache utilization with performance counters and using it to infer a crypto key? $\rightarrow$ Yes

Side Channel or not?

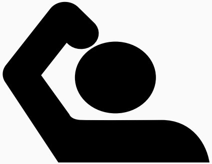


- Profiling cache utilization with performance counters? $\rightarrow$ No
- Observing cache utilization with performance counters and using it to infer a crypto key? $\rightarrow$ Yes
- Measuring memory access latency with Flush+Reload?



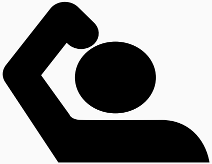
- Profiling cache utilization with performance counters? $\rightarrow$ No
- Observing cache utilization with performance counters and using it to infer a crypto key? $\rightarrow$ Yes
- Measuring memory access latency with Flush+Reload? $\rightarrow$ No

Side Channel or not?



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

Side Channel or not?



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

Intel Analysis of Speculative Execution Side Channels

Download PDF

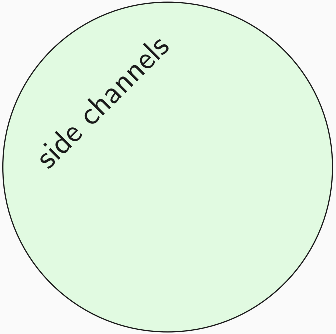
intel

Intel Analysis of Speculative Execution Side Channels

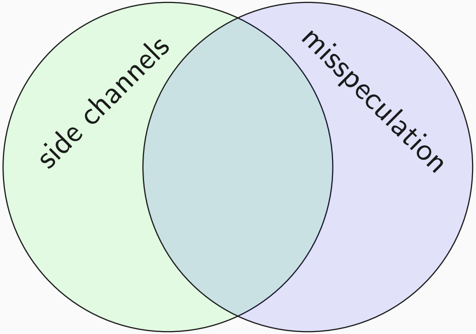
White Paper

Speculative Side-Channel Attacks?

Speculative Side-Channel Attacks?

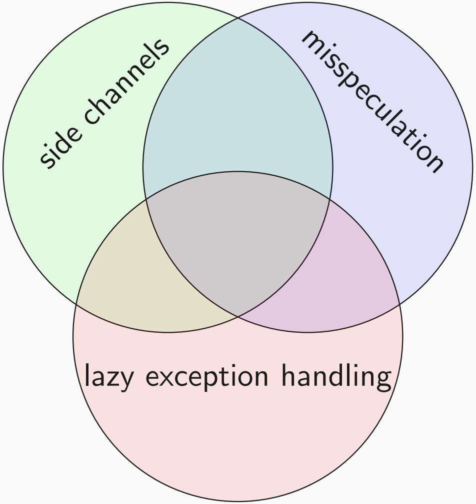


- traditional cache attacks (crypto, keys, etc)

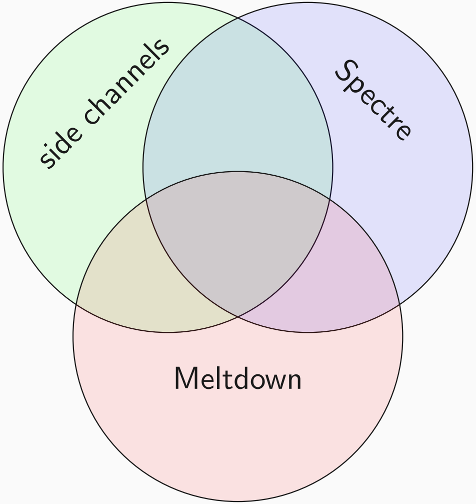


- traditional cache attacks (crypto, keys, etc)
- actual misspeculation (e.g., branch misprediction)

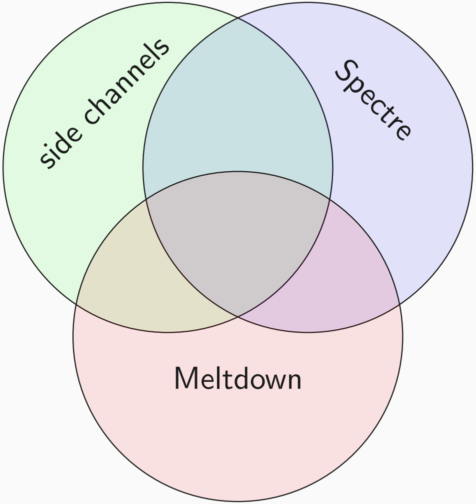
Speculative Side-Channel Attacks?



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



- 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.

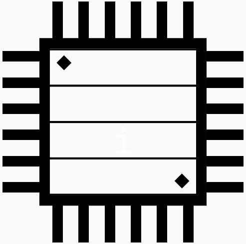
Can store **ALL** kinds of food.

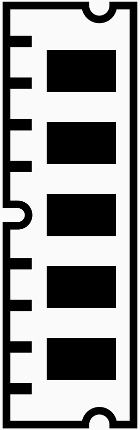
ONLY TODAY INSTEAD OF ~~$1,300~~

$1,299

ORDER VIA PHONE: +555 12345

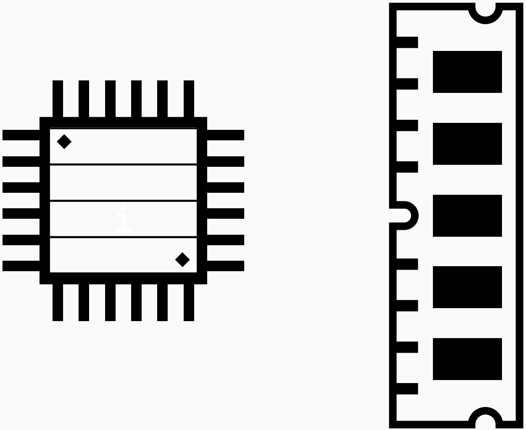
```
printf("%d", i);
printf("%d", i);
```





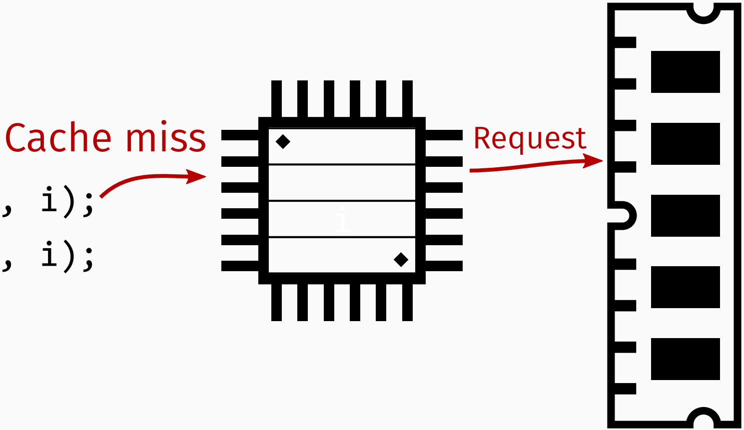
```
printf("%d", i);
printf("%d", i);
```

Cache miss

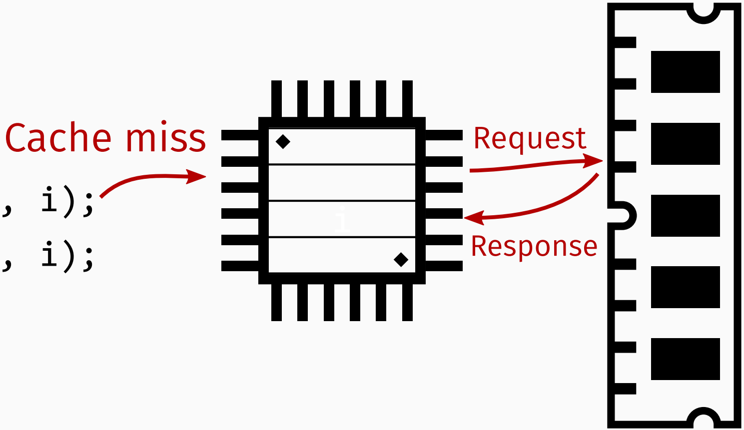


CPU Cache

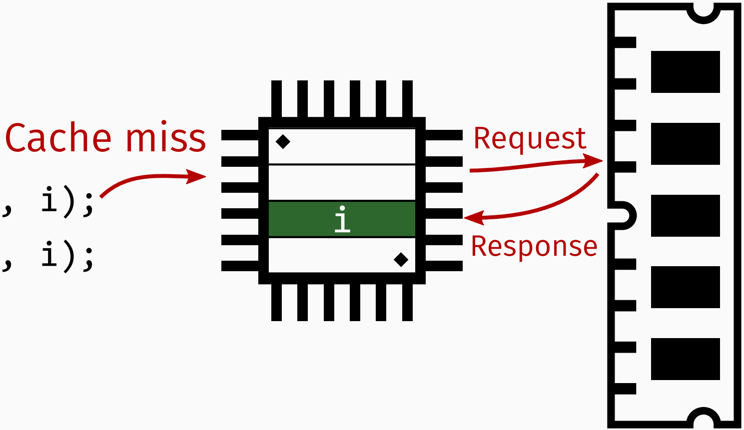
```
printf("%d", i);
printf("%d", i);
```

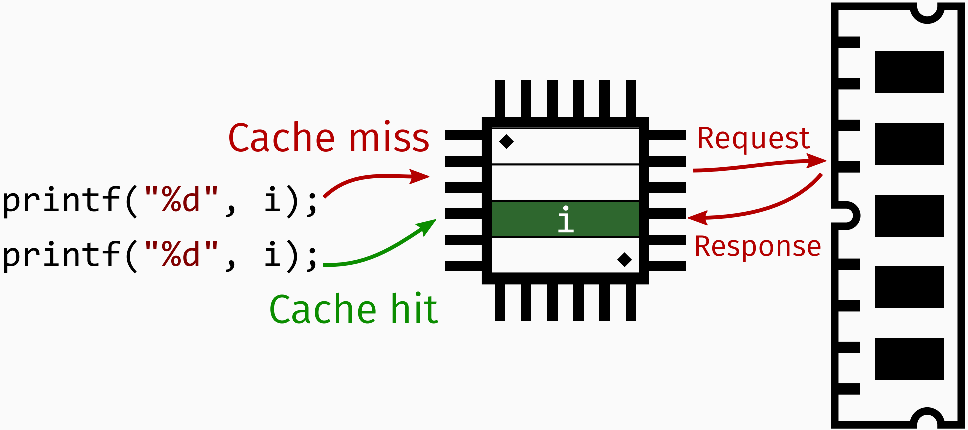


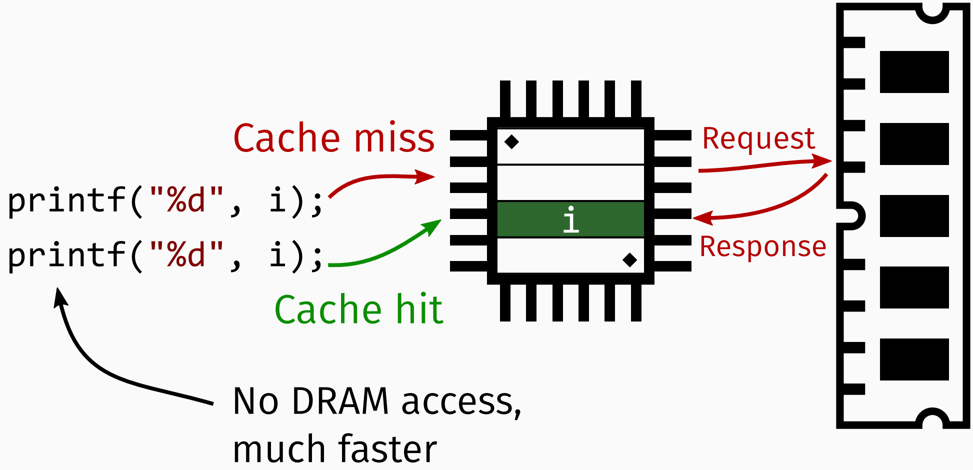
```
printf("%d", i);
printf("%d", i);
```

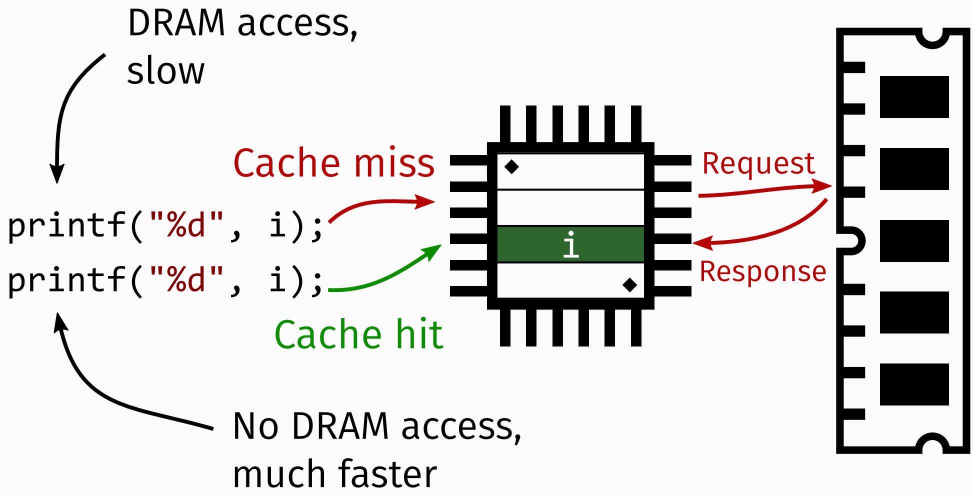


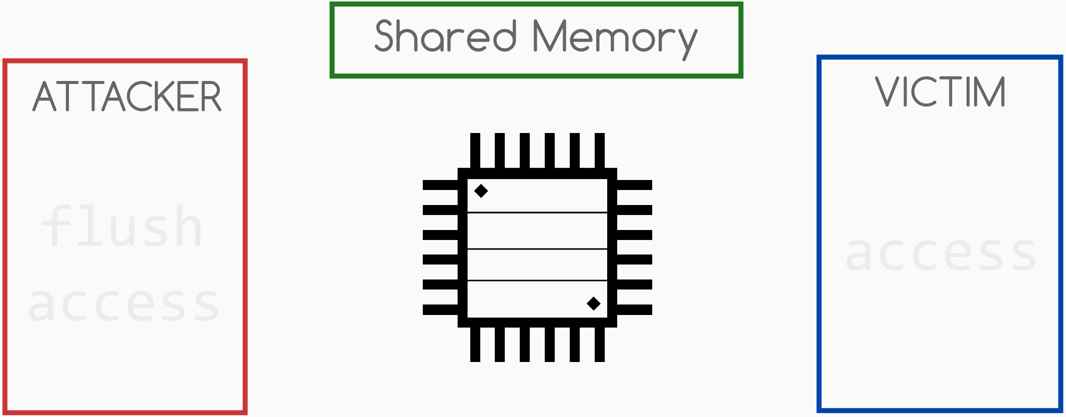
```
printf("%d", i);
printf("%d", i);
```



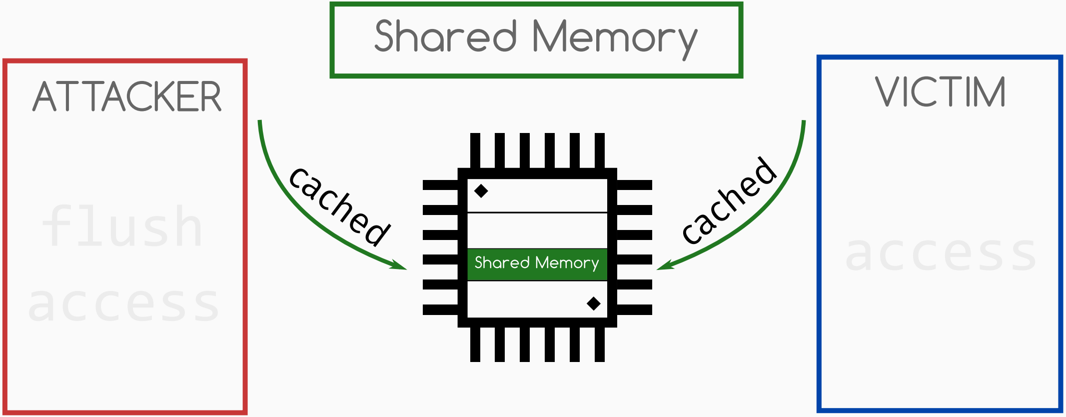

printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response


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

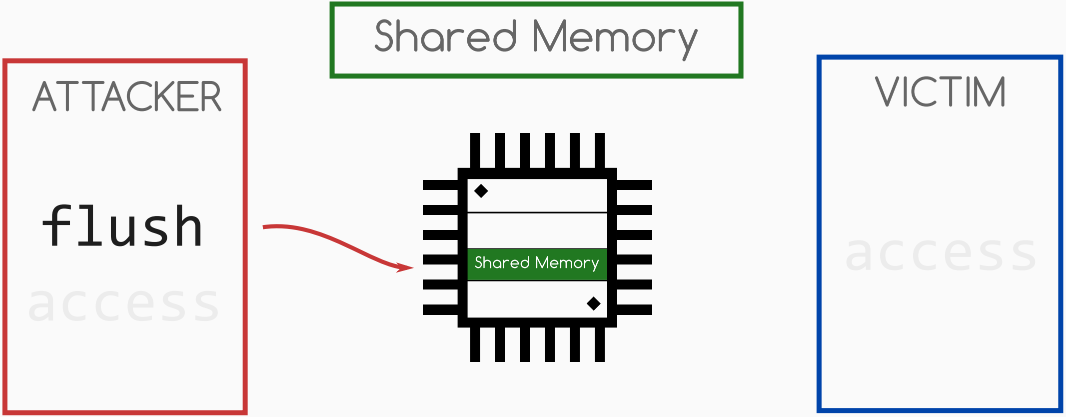

DRAM access,
slow
Cache miss
printf("%d", i);
printf("%d", i);
Cache hit
i
Request
Response
No DRAM access,
much faster


Shared Memory
ATTACKER
flush
access
VICTIM
access

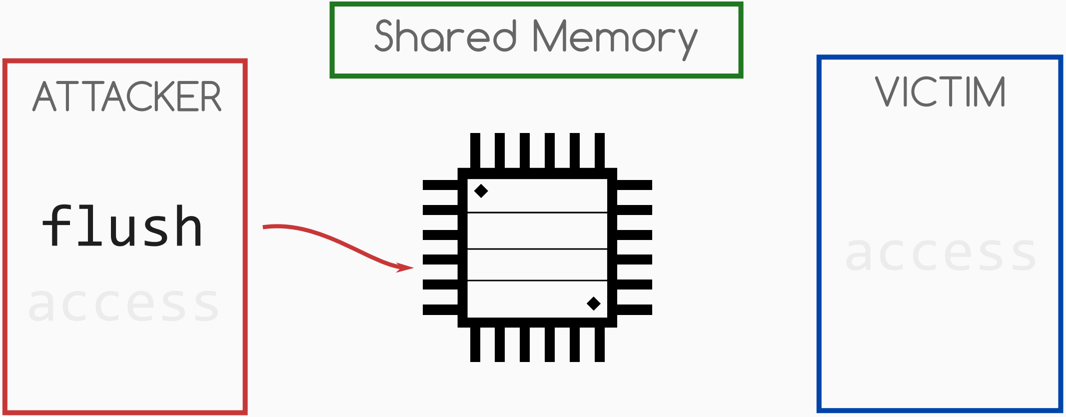
Flush+Reload

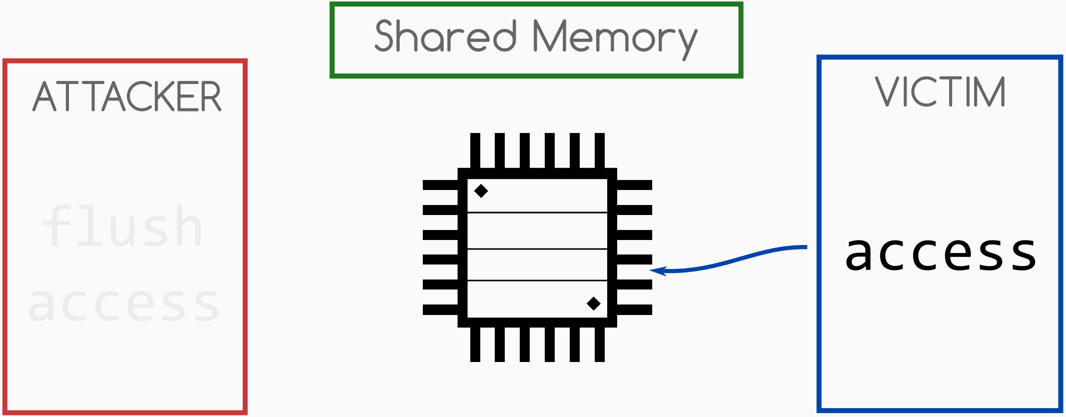


Flush+Reload

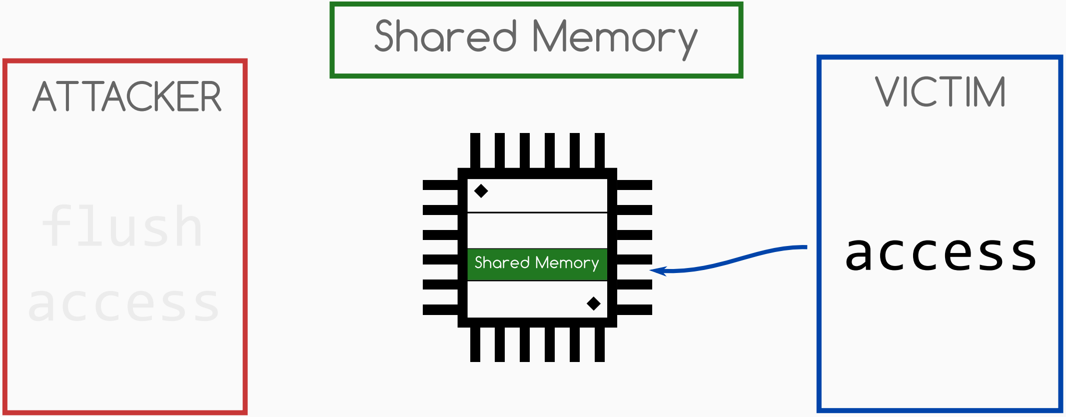


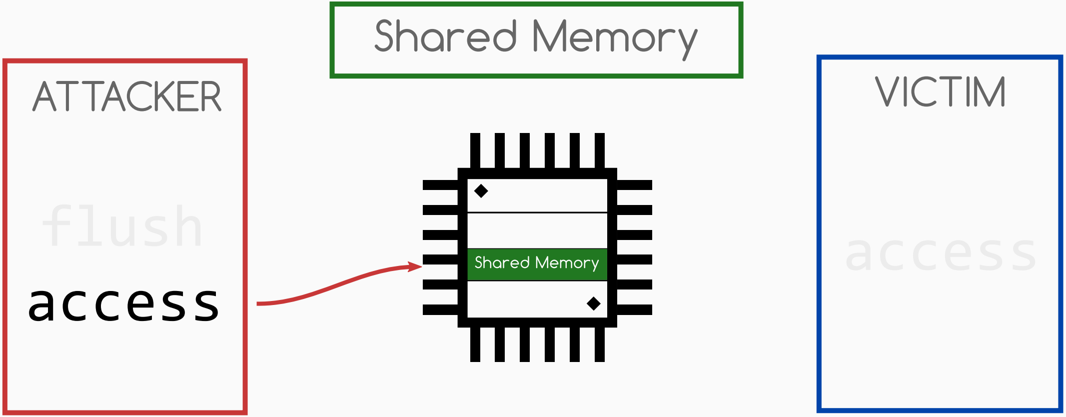
Flush+Reload

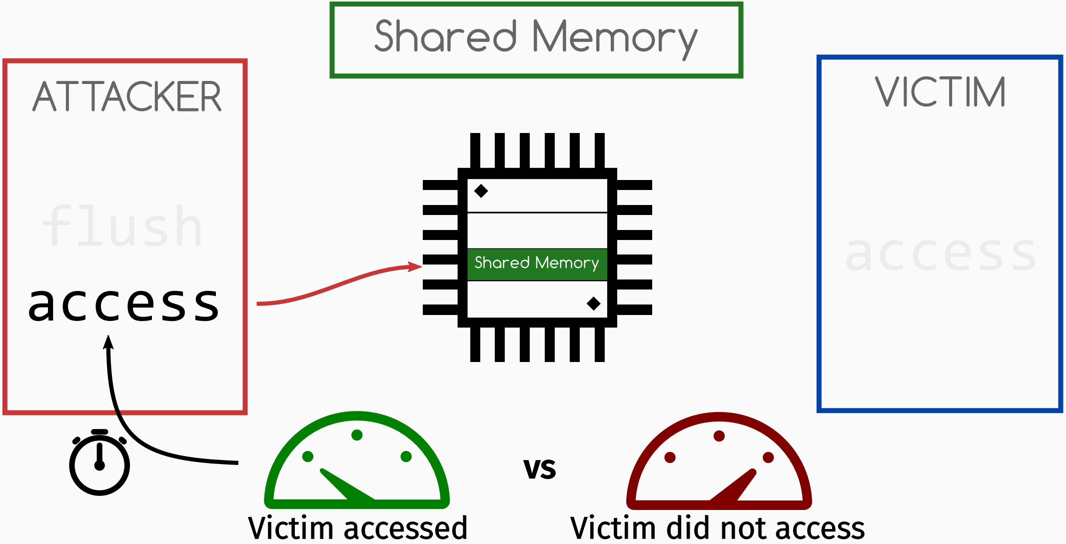



Shared Memory
ATTACKER
flush
access
VICTIM
access

Flush+Reload




Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access


Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access
vs
Victim accessed
Victim did not access

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

Accurate Microarchitecture Timing

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

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

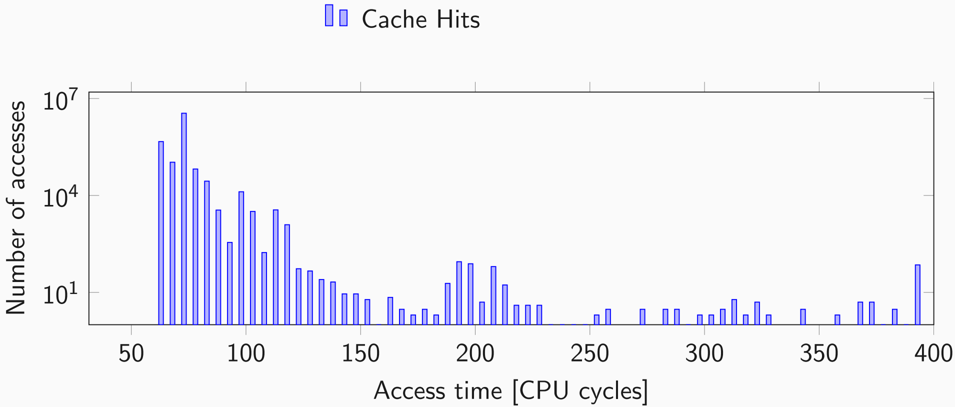
- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cpuid`
- and/or use fences like `mfence`

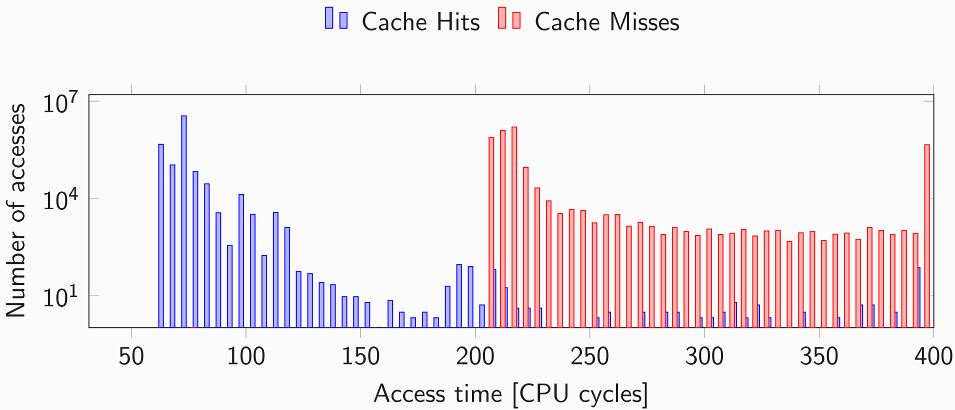
Intel, *How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures White Paper*, December 2010.

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.

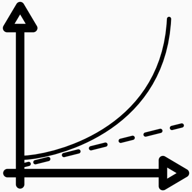

Cache Hits
Number of accesses
10^7
10^4
10^1
50
100
150
200
250
300
350
400
Access time [CPU cycles]



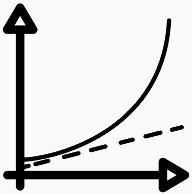
Cache Hits
Cache Misses
Number of accesses
10^7
10^4
10^1
50
100
150
200
250
300
350
400
Access time [CPU cycles]

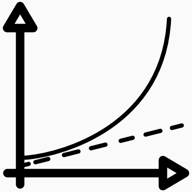
Talking about Accuracy

Talking about Accuracy

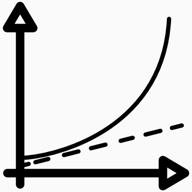


Talking about Accuracy

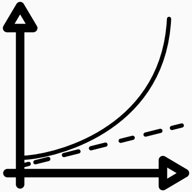




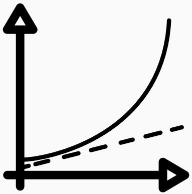
- Flush+Reload had beautifully nice timings, right?



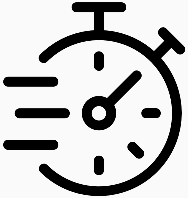
- Flush+Reload had beautifully nice timings, right?
- Well... steps of 2-4 cycles



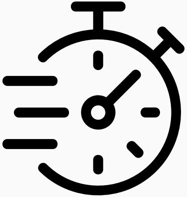
- Flush+Reload had beautifully nice timings, right?
- Well... steps of 2-4 cycles
    - only 35-70 steps between hits and misses



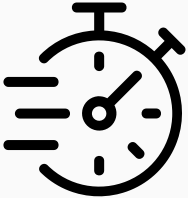
- Flush+Reload had beautifully nice timings, right?
- Well... steps of 2-4 cycles
    - only 35-70 steps between hits and misses
- On some devices only 1-2 steps!



- 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

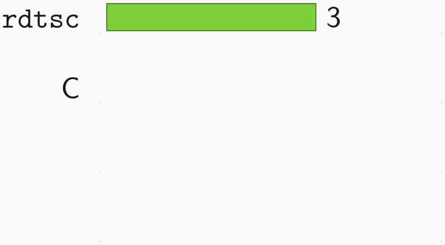

ARE YOU REALLY EXPECTING TO
OUTPERFORM THE HARDWARE COUNTER?
imgflip.com

CPU cycles one increment takes


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

Self-built Timer

CPU cycles one increment takes




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

CPU cycles one increment takes

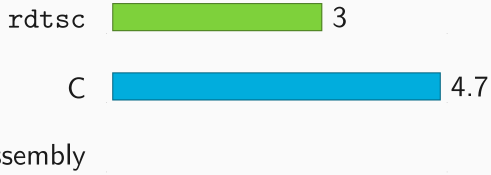

```
while(1) {
  timestamp++;
}
```

CPU cycles one increment takes


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

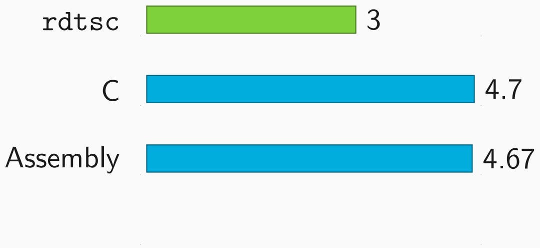
Self-built Timer

CPU cycles one increment takes




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

CPU cycles one increment takes

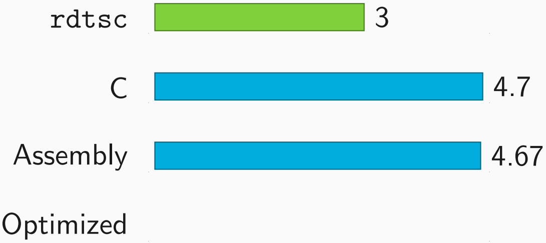



```
mov &timestamp, %rcx
1: incl (%rcx)
jmp 1b
```

CPU cycles one increment takes


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

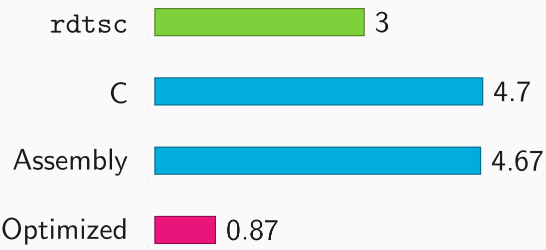
CPU cycles one increment takes



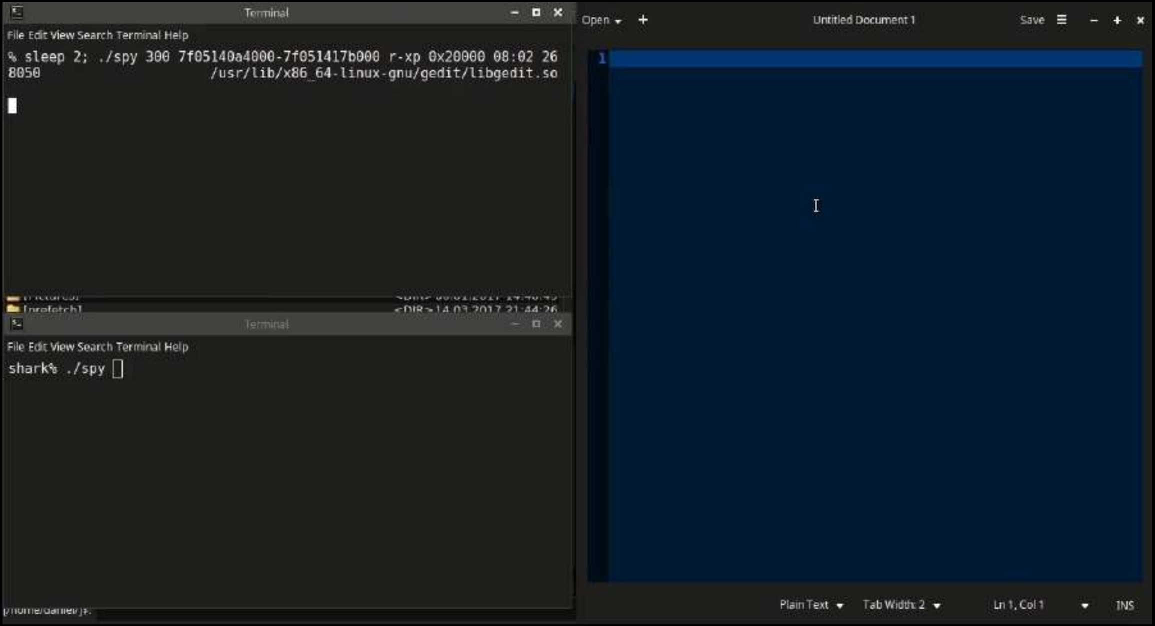

```
mov &timestamp, %rcx
1: inc %rax
mov %rax, (%rcx)
jmp 1b
```

Self-built Timer

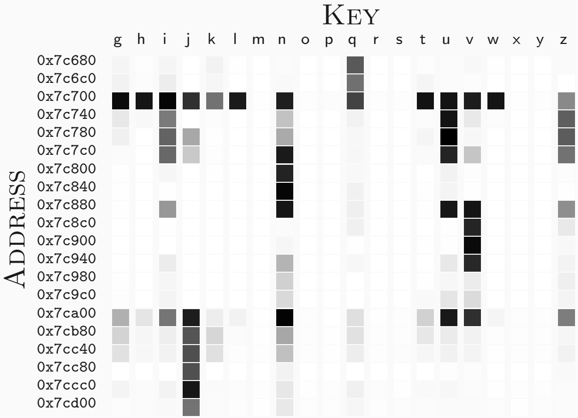
CPU cycles one increment takes

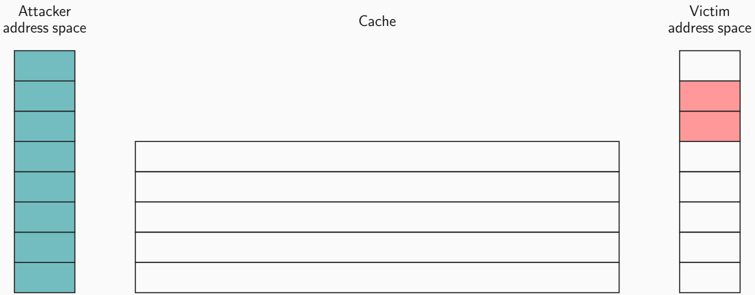



```
mov &timestamp, %rcx
1: inc %rax
mov %rax, (%rcx)
jmp 1b
```

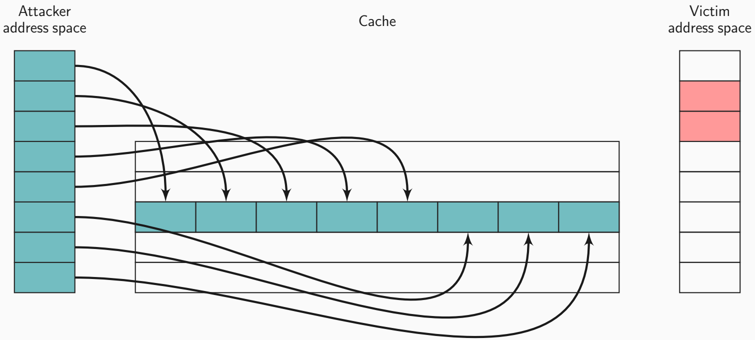


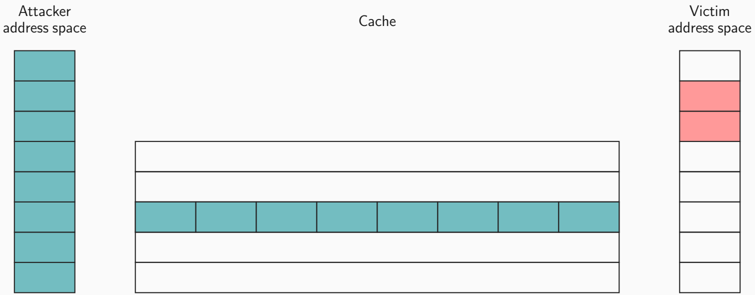
Cache Template Attack Demo

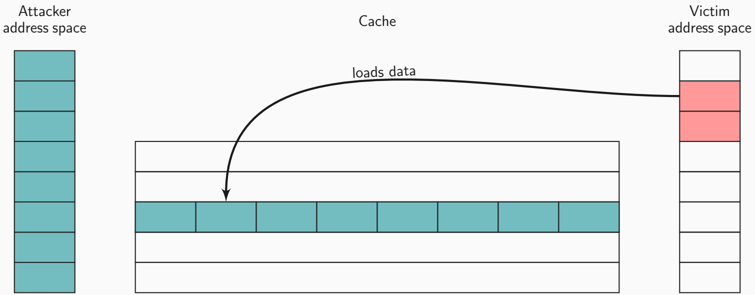

Key
g h i j k l m n o p q r s t u v w x y z
Address
0x7c680
0x7c6c0
0x7c700
0x7c740
0x7c780
0x7c7c0
0x7c800
0x7c840
0x7c880
0x7c8c0
0x7c900
0x7c940
0x7c980
0x7c9c0
0x7ca00
0x7cb80
0x7cc40
0x7cc80
0x7ccc0
0x7cd00

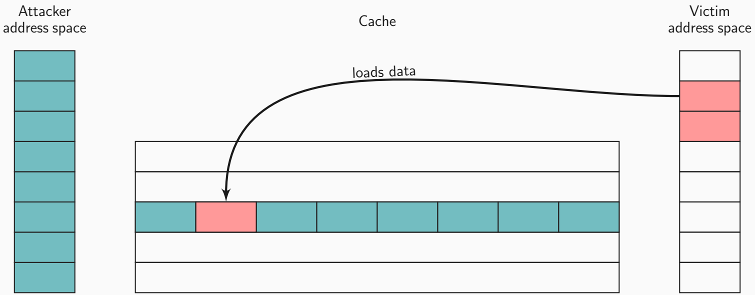

Attacker
address space
Cache
Victim
address space

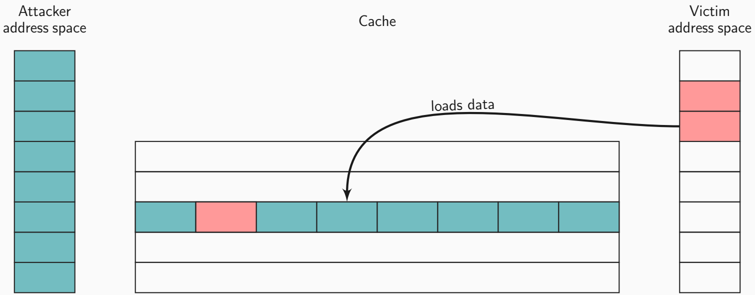
Prime+Probe



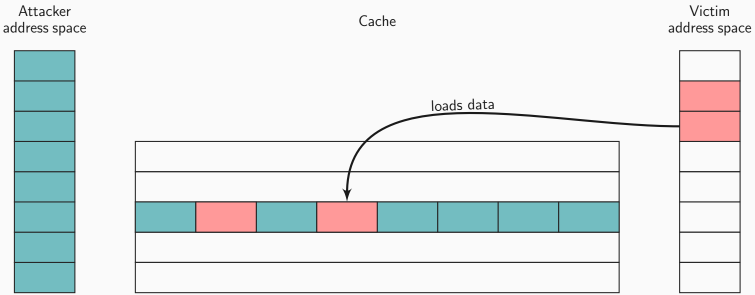

Attacker
address space
Cache
Victim
address space

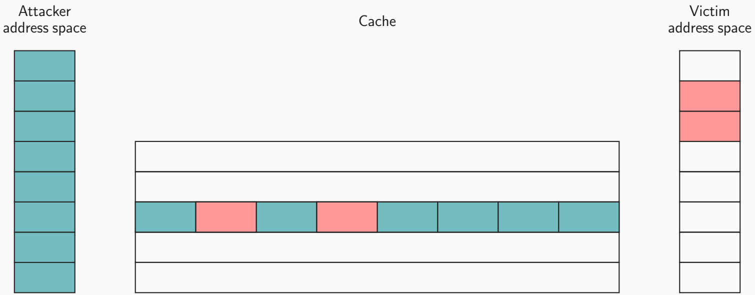

Attacker
address space
Cache
Victim
address space
loads data

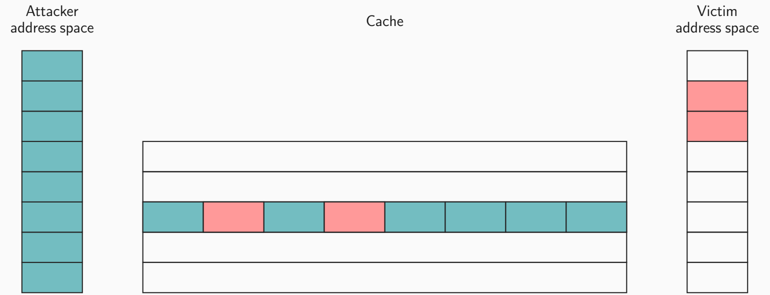

Attacker
address space
Cache
Victim
address space
loads data


Attacker
address space
Cache
Victim
address space
loads data

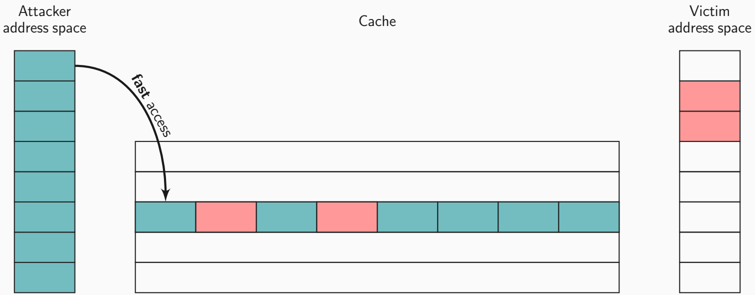
Prime+Probe

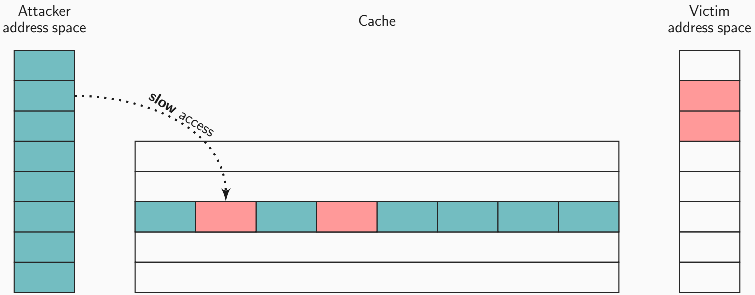

Attacker
address space
Cache
Victim
address space
loads data


Attacker
address space
Cache
Victim
address space


Attacker
address space
Cache
Victim
address space

Prime+Probe


Attacker
address space
Cache
Victim
address space
fast access


Attacker
address space
Cache
Victim
address space
slow access

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

"LRU eviction" memory accesses

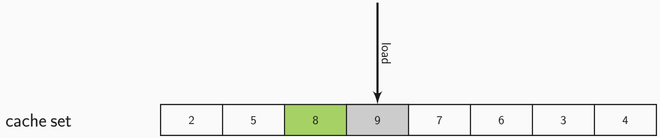
- LRU replacement policy: oldest entry first

"LRU eviction" memory accesses

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

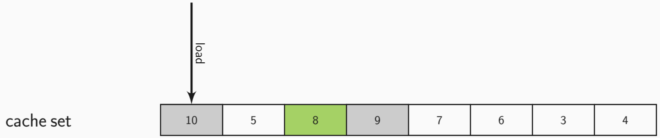
Replacement policy on older CPUs

"LRU eviction" memory accesses



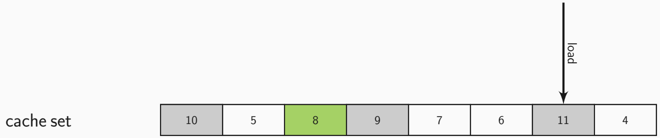
- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

"LRU eviction" memory accesses



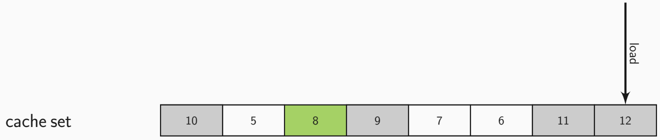
- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

"LRU eviction" memory accesses



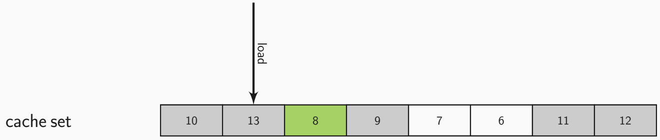
- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

"LRU eviction" memory accesses



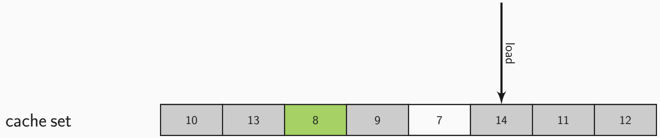
- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

"LRU eviction" memory accesses



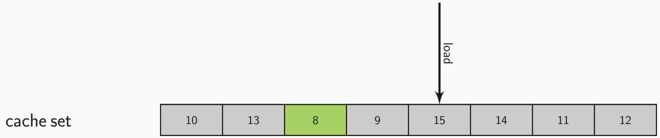
- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

“LRU eviction” memory accesses



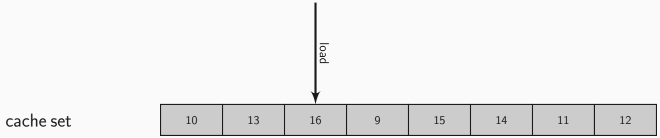
- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

"LRU eviction" memory accesses



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

"LRU eviction" memory accesses

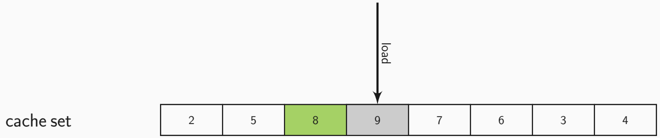


- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

“LRU eviction” memory accesses

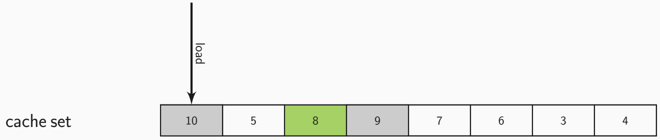
- no LRU replacement

"LRU eviction" memory accesses



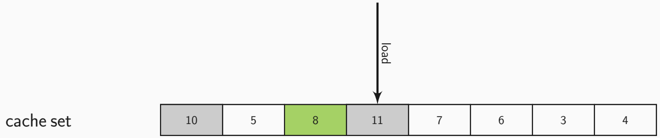
- no LRU replacement

"LRU eviction" memory accesses



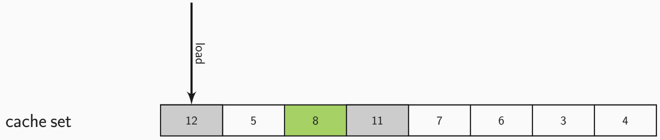
- no LRU replacement

"LRU eviction" memory accesses



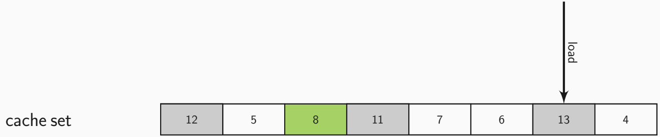
- no LRU replacement

“LRU eviction” memory accesses



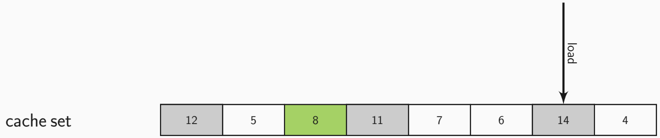
- no LRU replacement

"LRU eviction" memory accesses



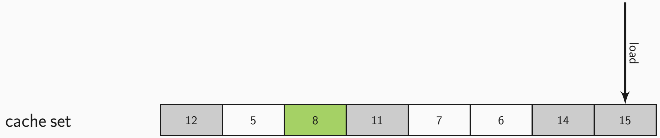
- no LRU replacement

"LRU eviction" memory accesses



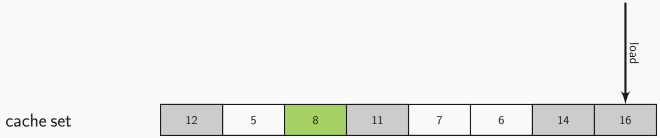
- no LRU replacement

"LRU eviction" memory accesses



- no LRU replacement

"LRU eviction" memory accesses



- no LRU replacement

"LRU eviction" memory accesses

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

"LRU eviction" memory accesses

- 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 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

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
P-1-1-1-17	17		
P-1-1-1-20	20		

[1] Executed 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
P-1-1-1-17	17	74.46% ✗	
P-1-1-1-20	20	99.82% ✓	

[1] Executed 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
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗

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

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...[1]

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

[1] Executed 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
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34	99.86% ✓	

[1] Executed 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
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34	99.86% ✓	191 ns ✓

[1] Executed 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
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34	99.86% ✓	191 ns ✓
P-2-2-1-17	64		

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

We evaluated more than 10000 strategies...[1]

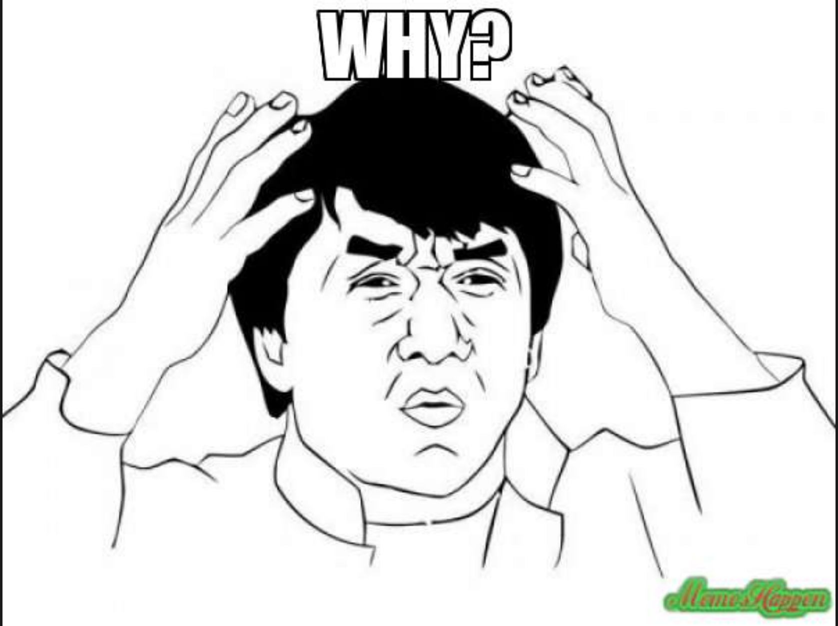
strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34	99.86% ✓	191 ns ✓
P-2-2-1-17	64	99.98% ✓	

[1] Executed 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
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34	99.86% ✓	191 ns ✓
P-2-2-1-17	64	99.98% ✓	180 ns ✓

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

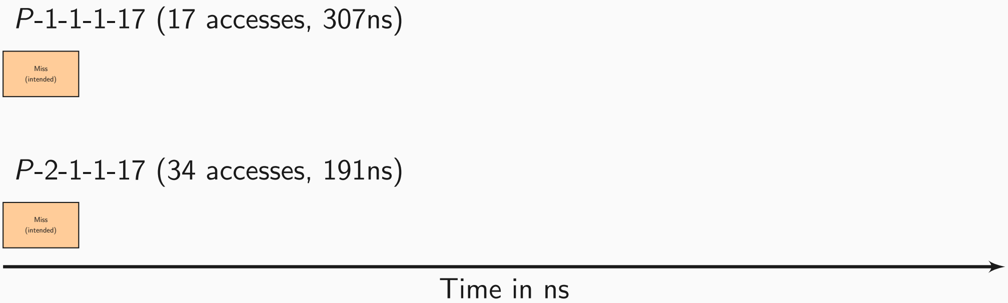

WHY?

Cache eviction strategies: Illustration

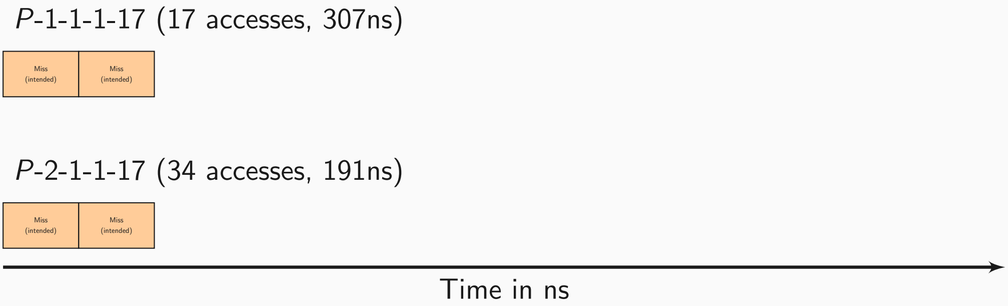
P-1-1-1-17 (17 accesses, 307ns)

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

Time in ns

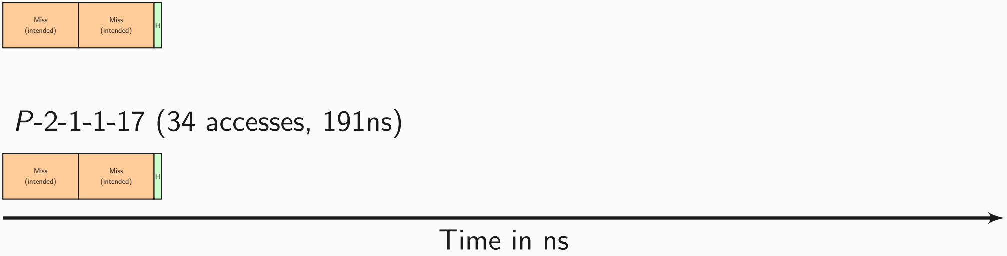

P-1-1-1-17 (17 accesses, 307ns)
Miss (intended)
P-2-1-1-17 (34 accesses, 191ns)
Miss (intended)
Time in ns

Cache eviction strategies: Illustration



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

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

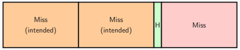


Cache eviction strategies: Illustration

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

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

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

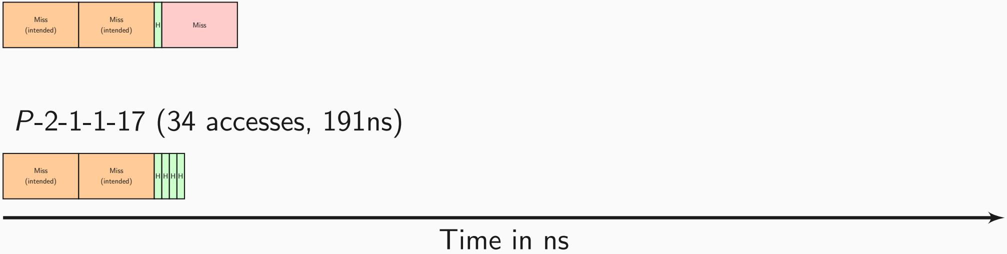


P-2-1-1-17 (34 accesses, 191ns)

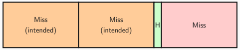
Cache eviction strategies: Illustration

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

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

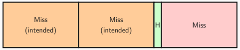


P-1-1-1-17 (17 accesses, 307ns)



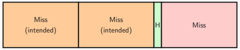
P-2-1-1-17 (34 accesses, 191ns)

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



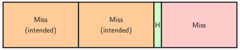
P-2-1-1-17 (34 accesses, 191ns)

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



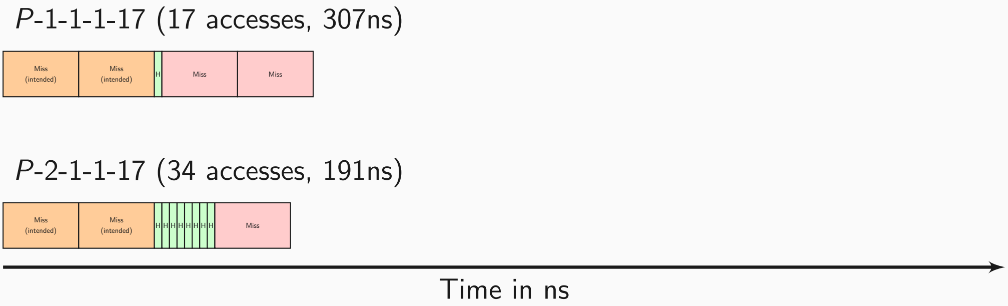
P-2-1-1-17 (34 accesses, 191ns)

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

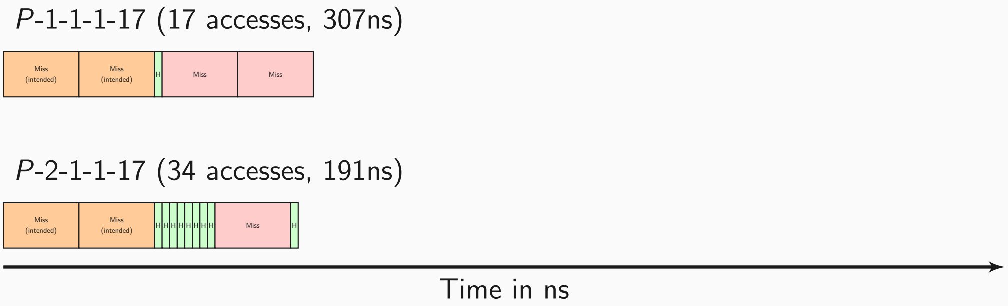


P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

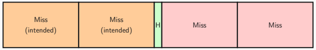


Cache eviction strategies: Illustration



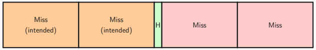
Cache eviction strategies: Illustration

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



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

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



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

Cache eviction strategies: Illustration

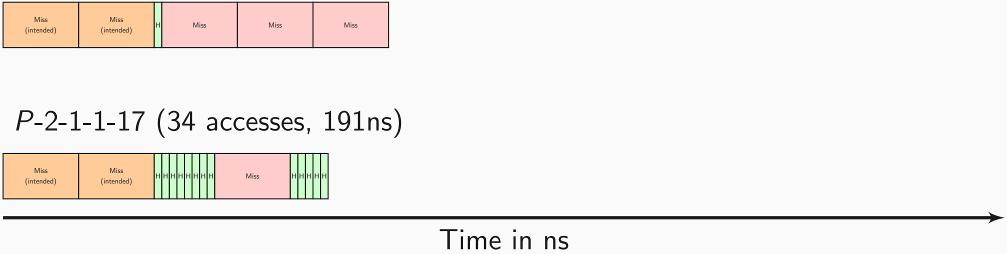
P-1-1-1-17 (17 accesses, 307ns)

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

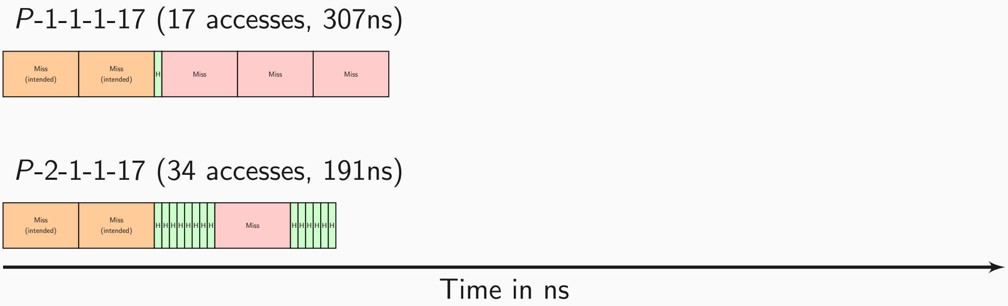
Cache eviction strategies: Illustration

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

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



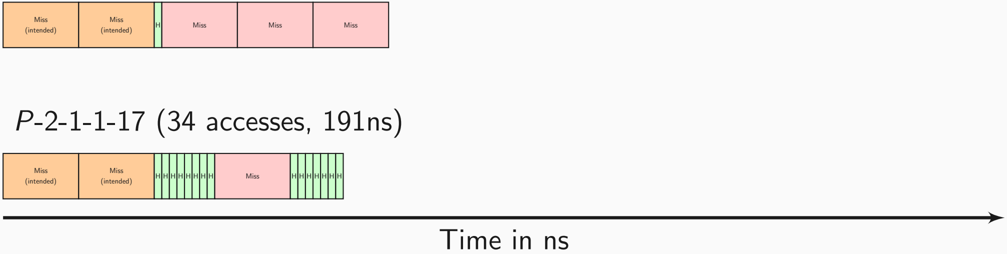
Cache eviction strategies: Illustration



Cache eviction strategies: Illustration

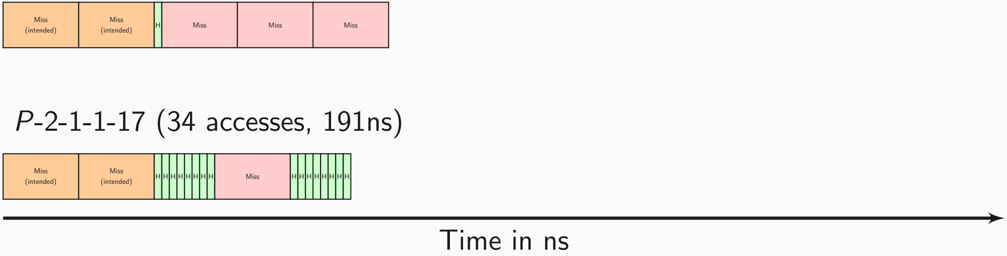
P-1-1-1-17 (17 accesses, 307ns)

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



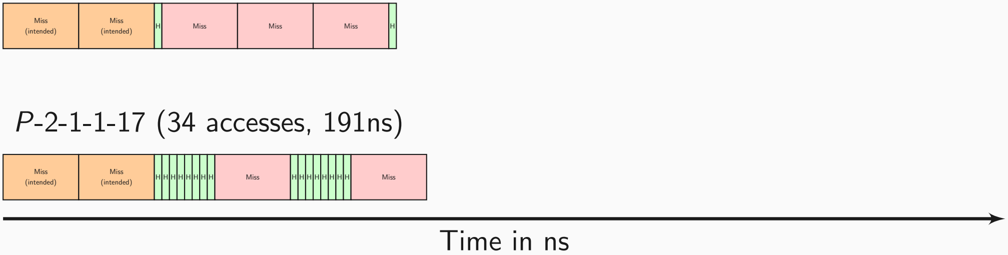
P-1-1-1-17 (17 accesses, 307ns)

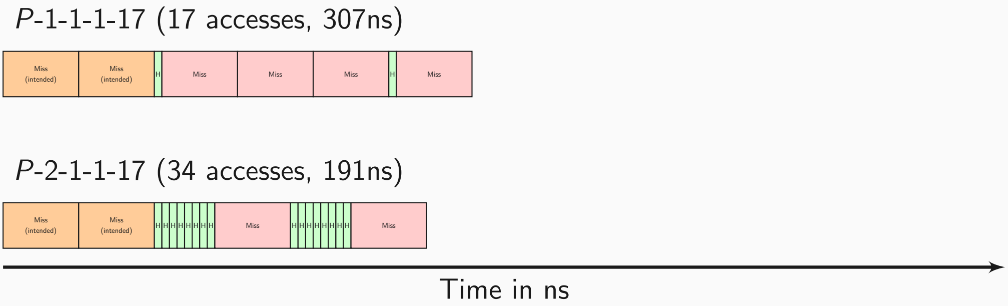
P-2-1-1-17 (34 accesses, 191ns)



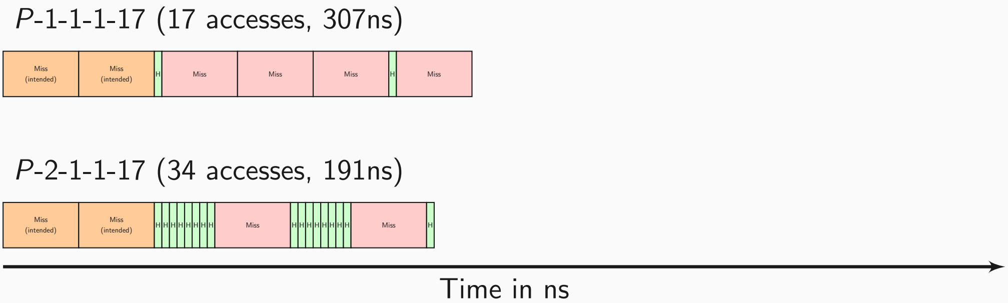
P-1-1-1-17 (17 accesses, 307ns)

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

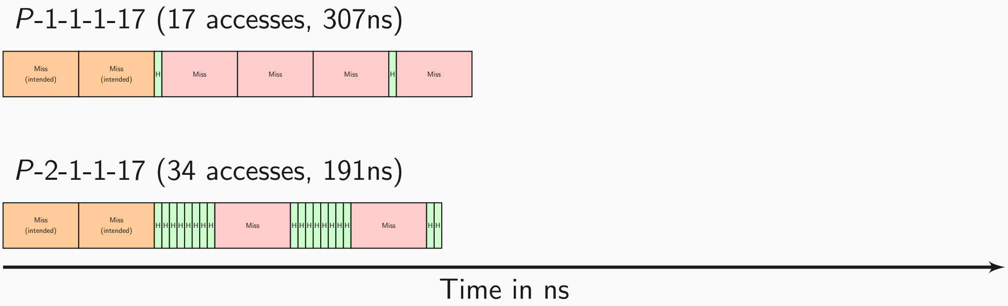


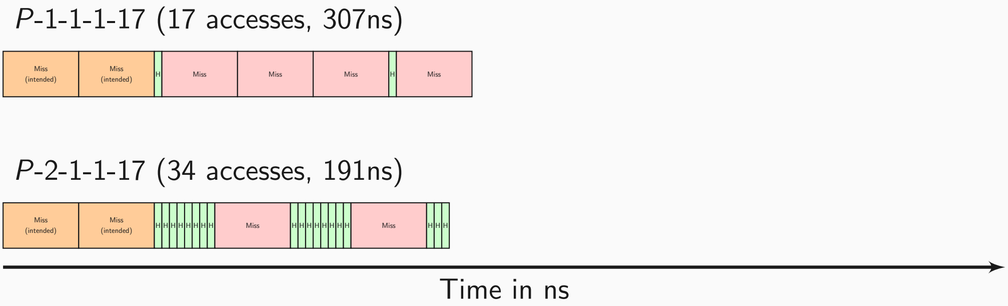

P-1-1-1-17 (17 accesses, 307ns)
Miss (intended)
Miss (intended)
H
Miss
Miss
Miss
H
Miss
P-2-1-1-17 (34 accesses, 191ns)
Miss (intended)
Miss (intended)
H H H H H H H H
Miss
H H H H H H H H
Miss
Time in ns

Cache eviction strategies: Illustration



Cache eviction strategies: Illustration

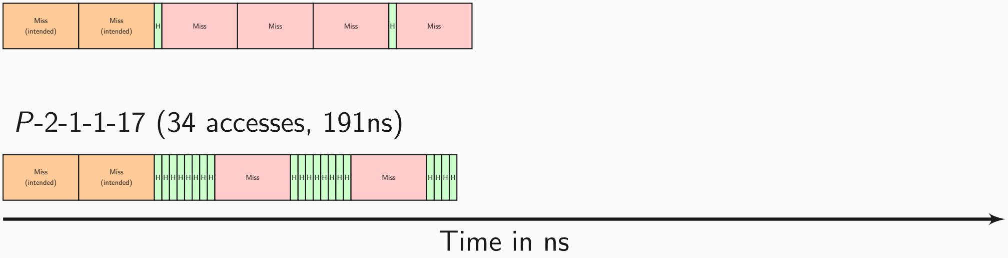




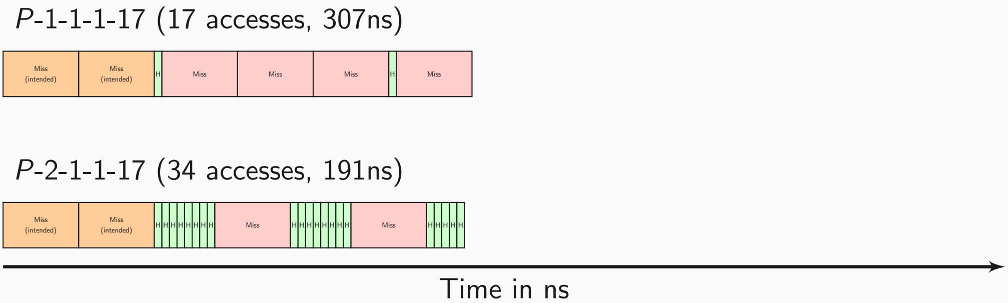
P-1-1-1-17 (17 accesses, 307ns)
Miss (intended)
Miss (intended)
H
Miss
Miss
Miss
H
Miss
P-2-1-1-17 (34 accesses, 191ns)
Miss (intended)
Miss (intended)
H H H H H H H H
Miss
H H H H H H H H
Miss
H H H
Time in ns

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

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



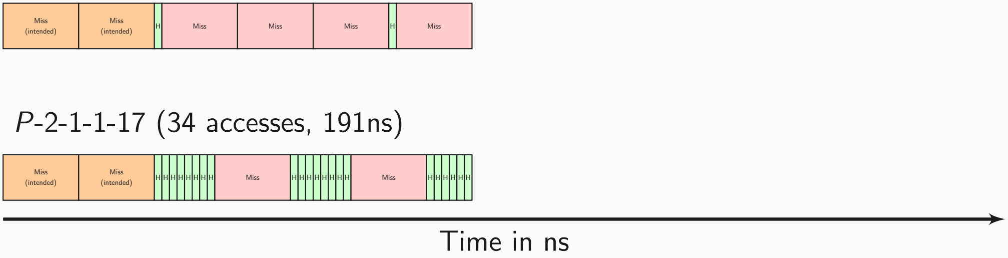
Cache eviction strategies: Illustration



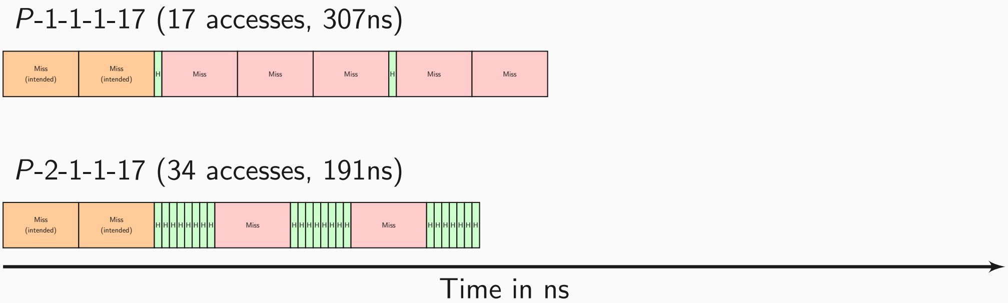
Cache eviction strategies: Illustration

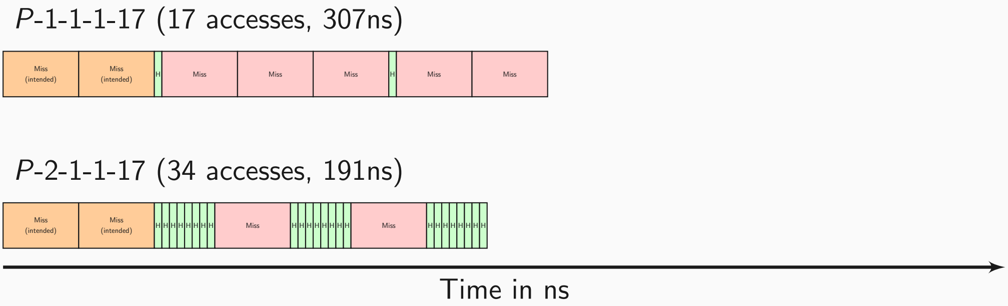
P-1-1-1-17 (17 accesses, 307ns)

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

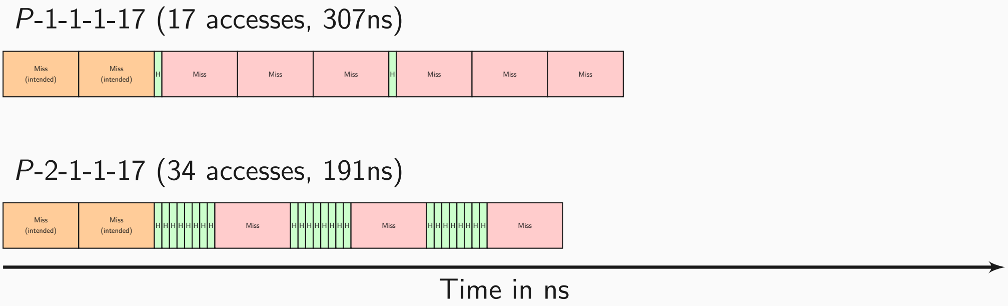


Cache eviction strategies: Illustration

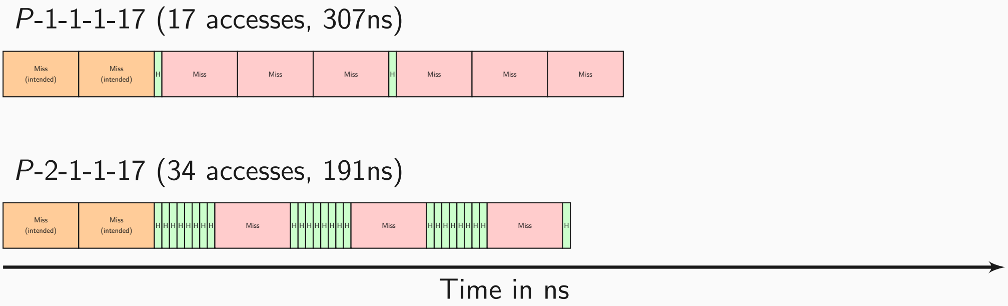



P-1-1-1-17 (17 accesses, 307ns)
Miss (intended)
Miss (intended)
H
Miss
Miss
Miss
H
Miss
Miss
P-2-1-1-17 (34 accesses, 191ns)
Miss (intended)
Miss (intended)
H H H H H H H H
Miss
H H H H H H H H
Miss
H H H H H H H H
Time in ns

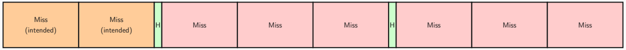
Cache eviction strategies: Illustration



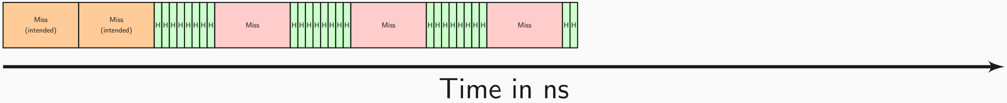
Cache eviction strategies: Illustration



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

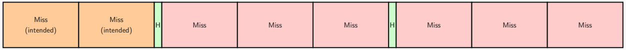


P-2-1-1-17 (34 accesses, 191ns)

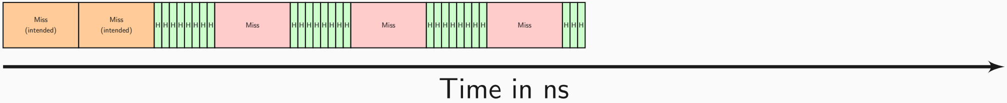


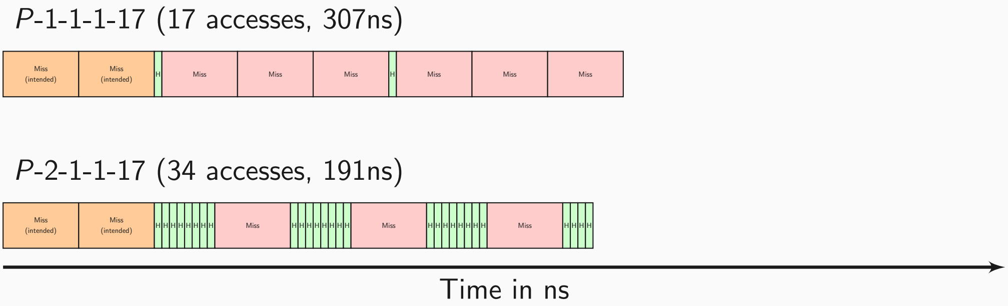
Cache eviction strategies: Illustration

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



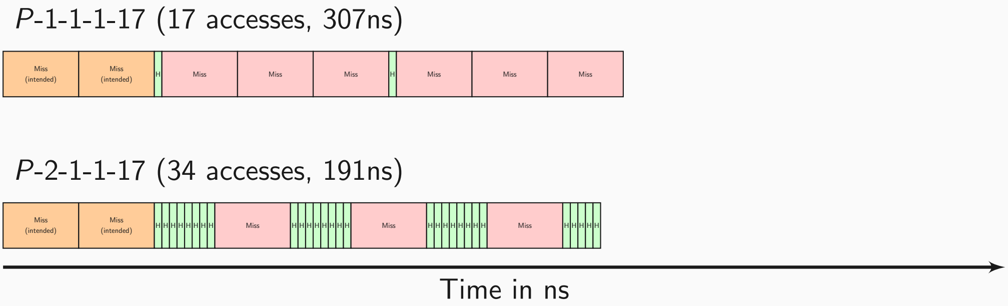
P-2-1-1-17 (34 accesses, 191ns)



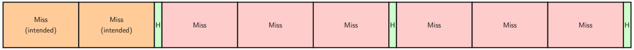


P-1-1-1-17 (17 accesses, 307ns)
Miss (intended)
Miss (intended)
H
Miss
Miss
Miss
H
Miss
Miss
Miss
P-2-1-1-17 (34 accesses, 191ns)
Miss (intended)
Miss (intended)
H H H H H H H H
Miss
H H H H H H H H
Miss
H H H H H H H H
Miss
H H H H
Time in ns

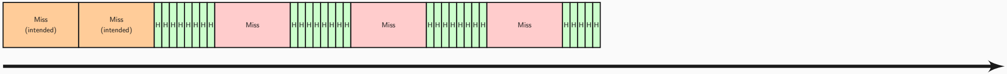
Cache eviction strategies: Illustration



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

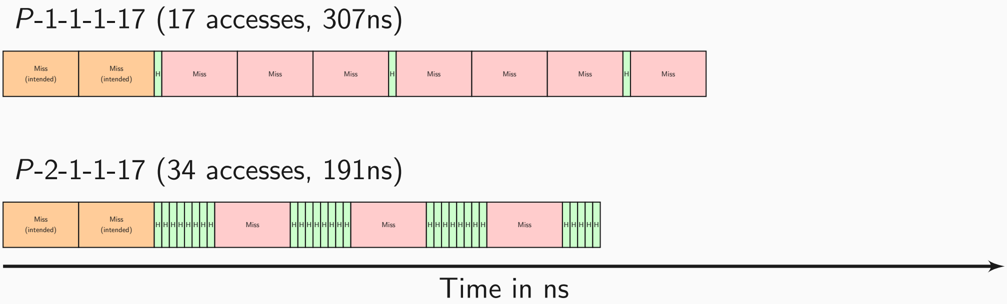


P-2-1-1-17 (34 accesses, 191ns)



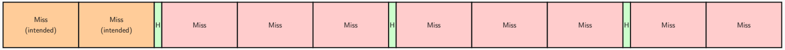
Time in ns

Cache eviction strategies: Illustration

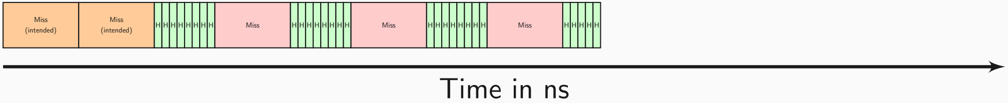


Cache eviction strategies: Illustration

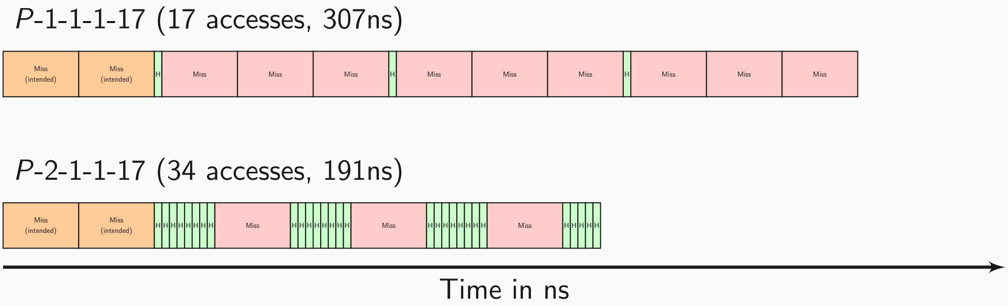
P-1-1-1-17 (17 accesses, 307ns)



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

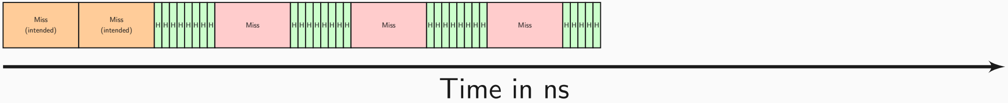


Cache eviction strategies: Illustration



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

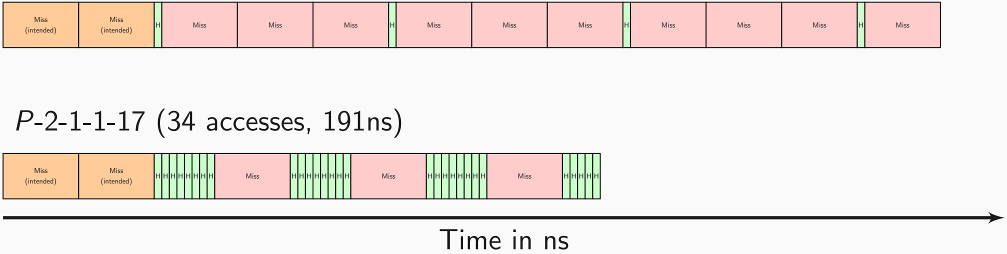
P-2-1-1-17 (34 accesses, 191ns)



Cache eviction strategies: Illustration

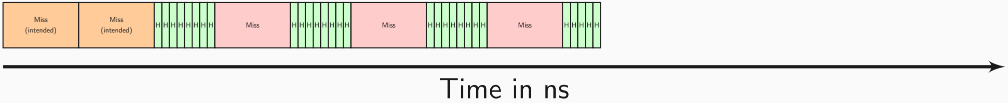
P-1-1-1-17 (17 accesses, 307ns)

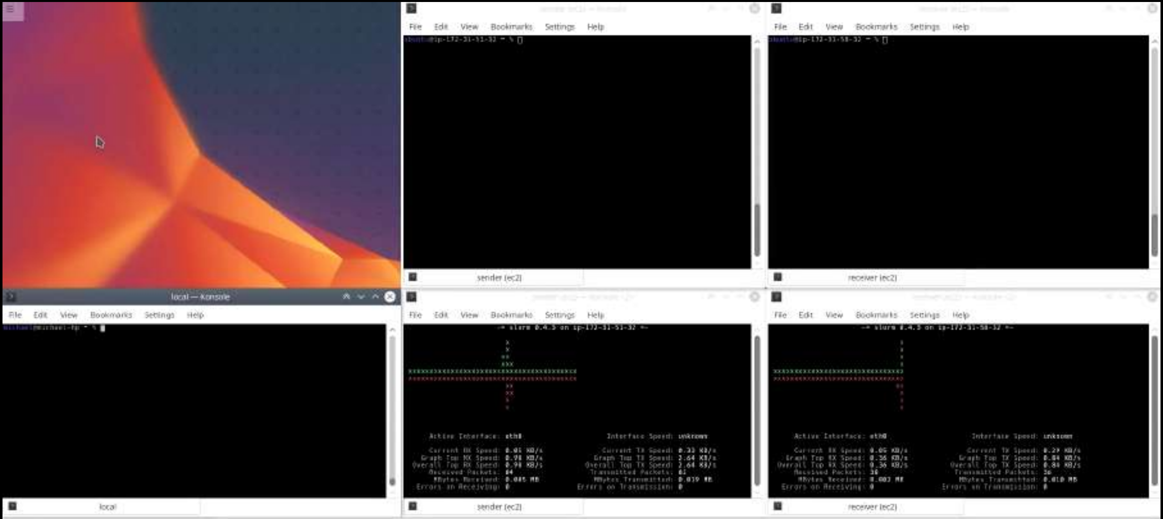
P-2-1-1-17 (34 accesses, 191ns)



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

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





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

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

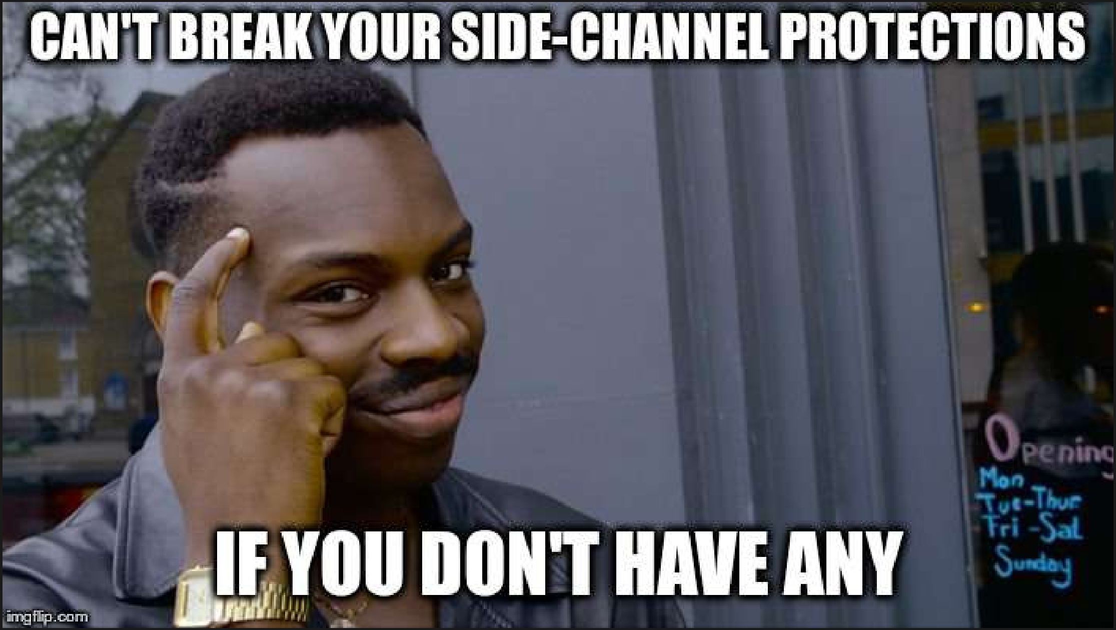
Protection from Side-Channel Attacks

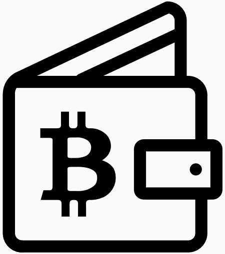
Protection from Side-Channel Attacks

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

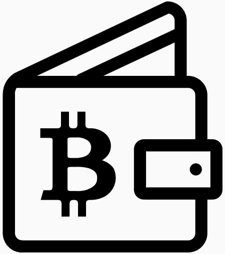
Protection from Side-Channel Attacks

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
Mon
Tue-Thur
Fri -Sat
Sunday
imgflip.com



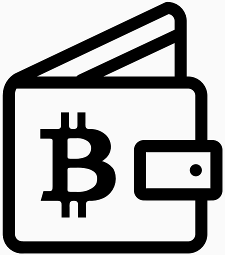
- Ledger SGX Enclave for blockchain applications
- BitPay Copay Bitcoin wallet
- Teechain payment channel using SGX



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

Teechain

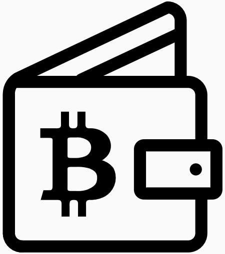
[...] We assume the TEE guarantees to hold



- 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.

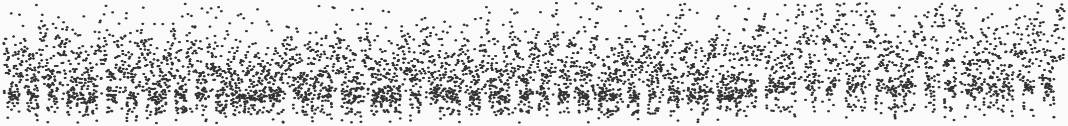


- Ledger SGX Enclave for blockchain applications
- BitPay Copay Bitcoin wallet
- Teechain payment channel using SGX

Teechain

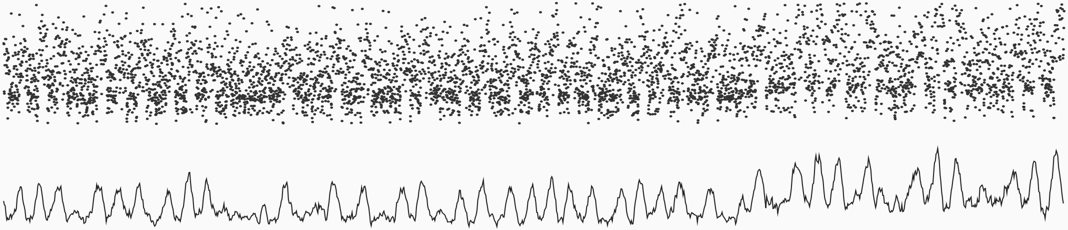
[...] We assume the TEE guarantees to hold and do not consider side-channel attacks [5, 35, 46] on the TEE. Such attacks and their mitigations [36, 43] are outside the scope of this work. [...]

Raw Prime+Probe trace...[2]



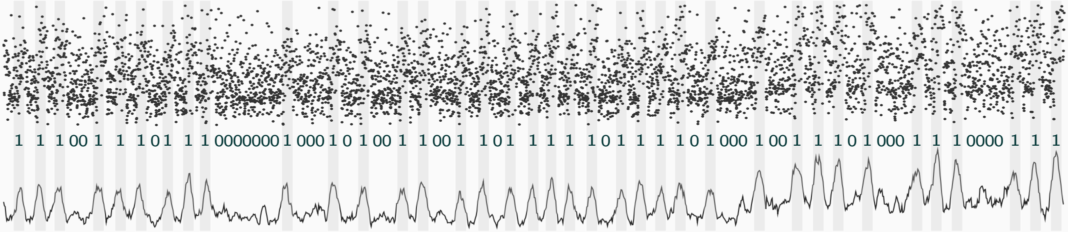
[2] Michael Schwarz et al. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In: DIMVA. 2017.

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

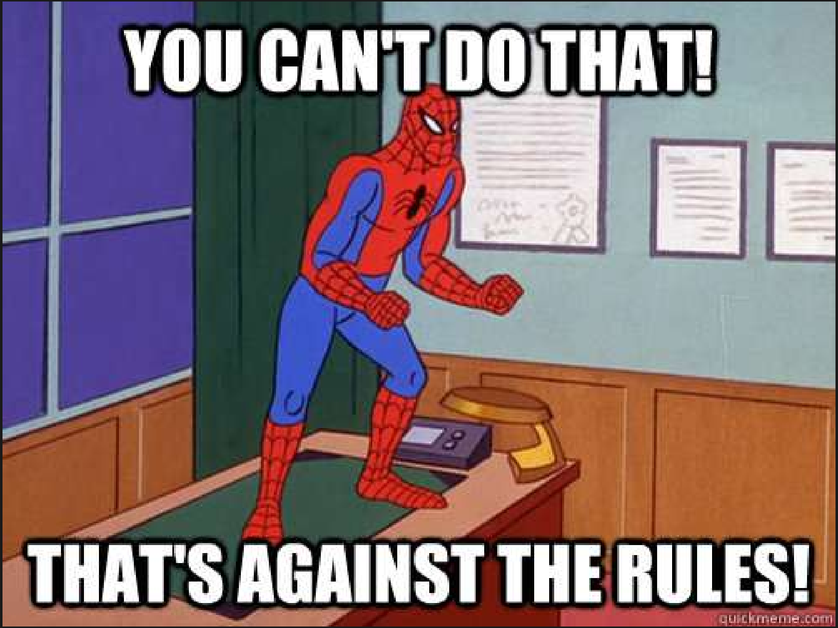


[3] Michael Schwarz et al. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In: DIMVA. 2017.

...allows to clearly see the bits of the exponent[4]



[4] Michael 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!
quickmeme.com


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

Parallelize

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

```c
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);
```

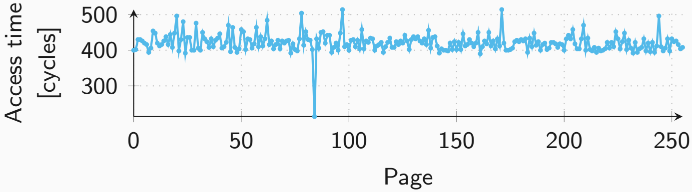
Parallelize

Dependency

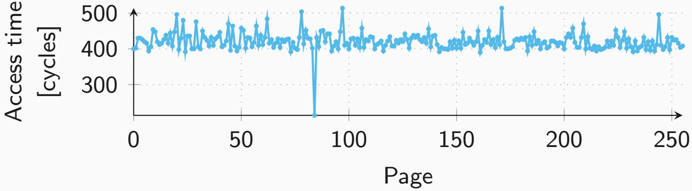
```
int width = 10, height = 5;

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

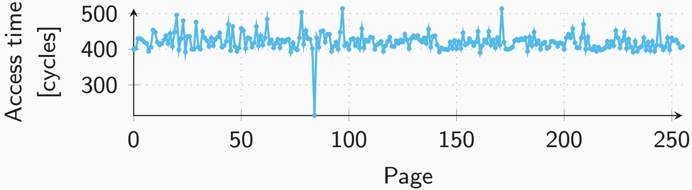


- Flush+Reload over all pages of the array



- "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

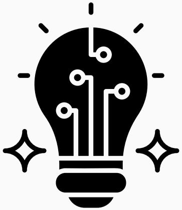
- Out-of-order instructions leave microarchitectural traces
  - We can see them for example through the cache

- 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

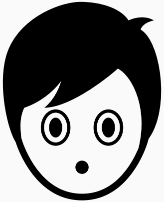
```
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```



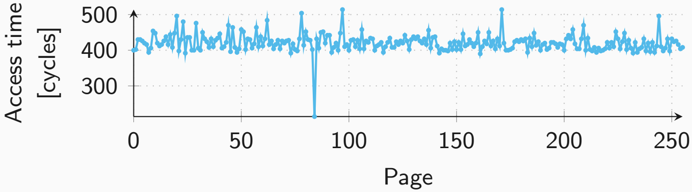
- Add another layer of indirection to test

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

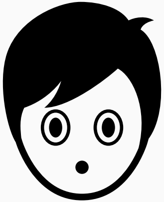
- Then check whether any part of array is cached



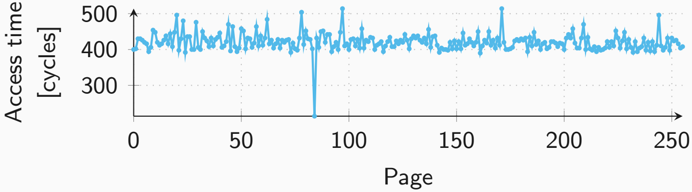
- Flush+Reload over all pages of the array



- Index of cache hit reveals data

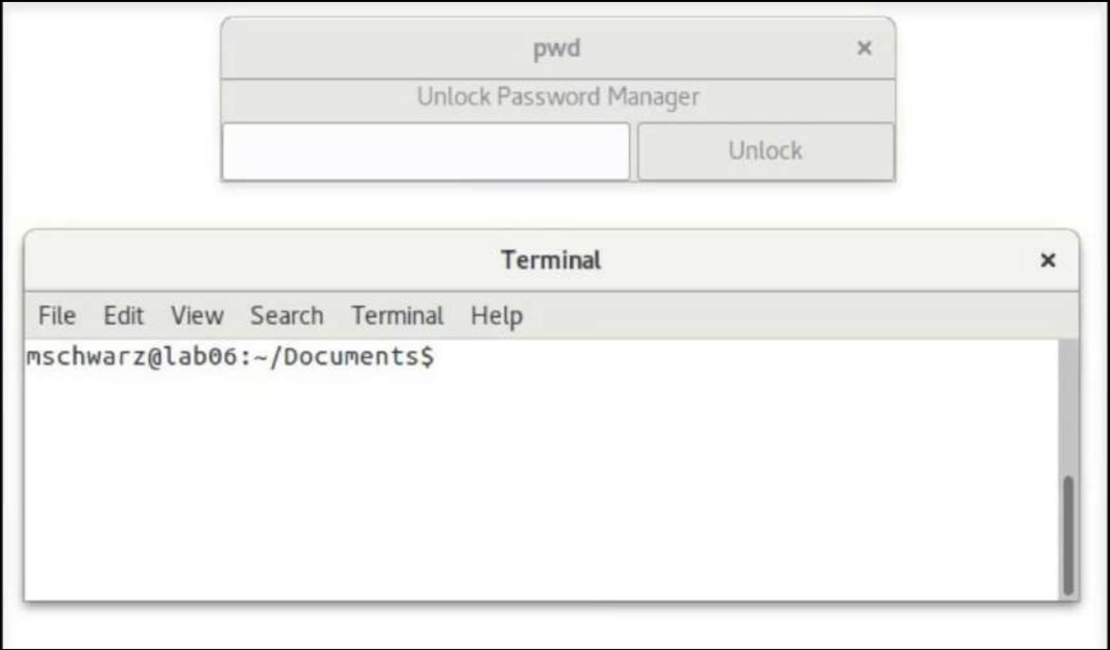


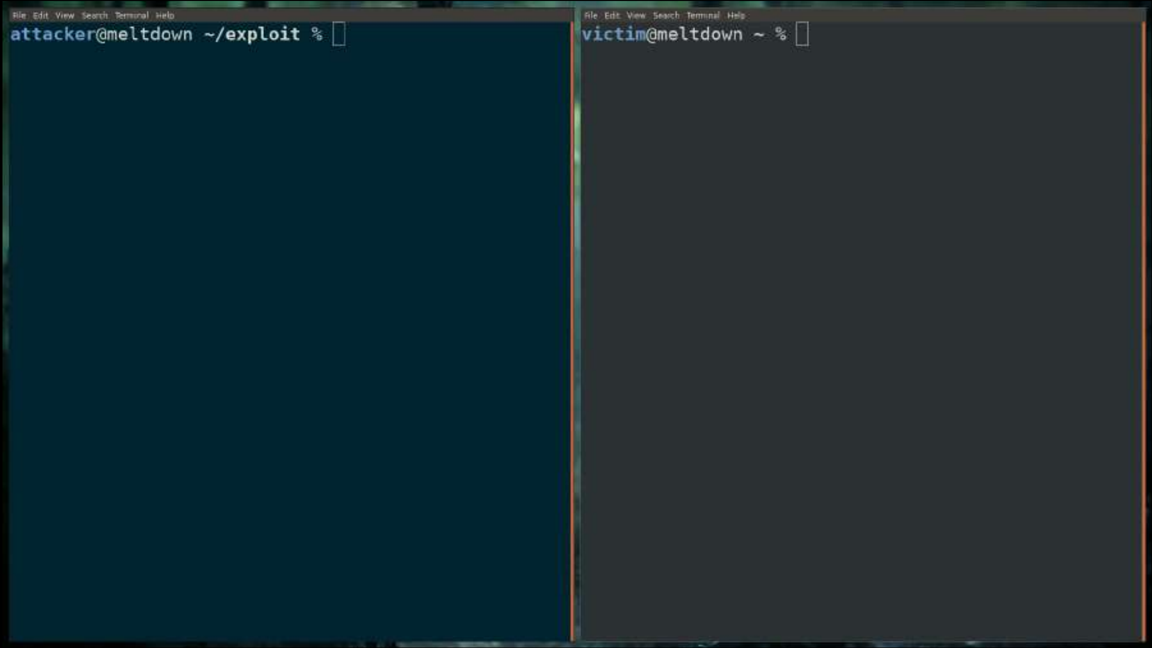
- Flush+Reload over all pages of the array



- 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


pwd
Unlock Password Manager
Unlock
Terminal
File Edit View Search Terminal Help
mschwarz@lab06:~/Documents$


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

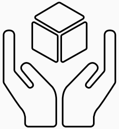
Details: Exception Handling

- Basic Meltdown code leads to a crash (segfault)

Details: Exception Handling

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

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



Fault Handling

Fault Suppression

Fault Prevention

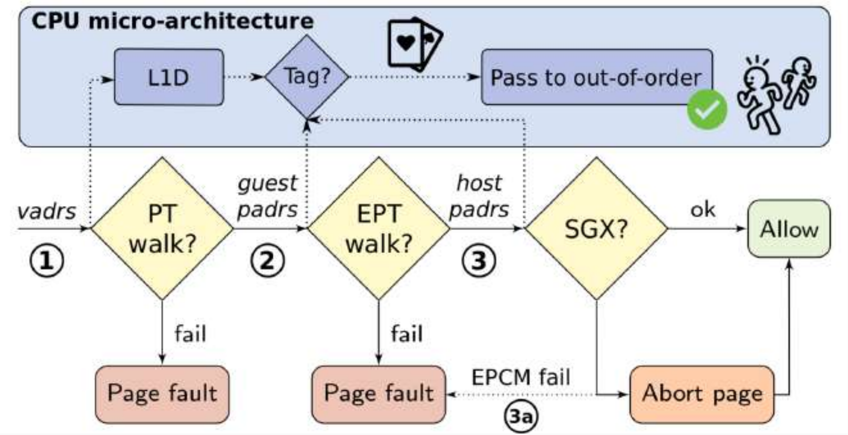
Meltdown with Fault Suppression

- Intel TSX to suppress exceptions instead of signal handler

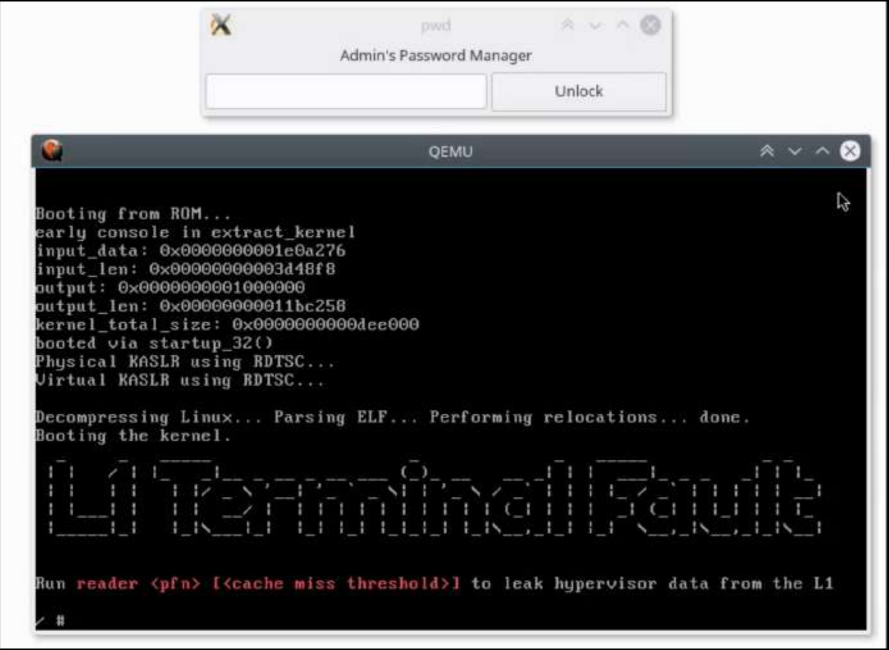
```
if(xbegin() == XBEGIN_STARTED) {
  char secret = *(char*) 0xffffffff81a000e0;
  array[secret * 4096] = 0;
  xend();
}

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

- Speculative execution to prevent exceptions

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

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



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


pwd
Admin's Password Manager
Unlock
QEMU
Booting from ROM...
early console in extract_kernel
input_data: 0x0000000001e0a276
input_len: 0x00000000003d48f8
output: 0x0000000001000000
output_len: 0x00000000011bc258
kernel_total_size: 0x0000000000dee000
booted via startup_32()
Physical KASLR using RDTSC...
Virtual KASLR using RDTSC...
Decompressing Linux... Parsing ELF... Performing relocations... done.
Booting the kernel.
Run reader <pfn> [<cache miss threshold>] to leak hypervisor data from the L1
/ #


Kernel Address Isolation to have Side channels Efficiently Removed

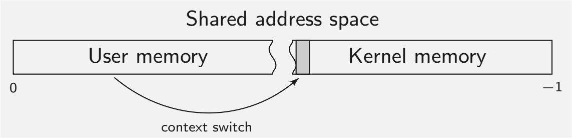
KAISER /ˈkʌɪzə/

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

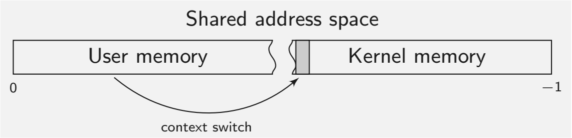
Kernel **A**ddress **I**solation to have **S**ide channels **E**fficiently **R**emoved

KAISER Illustration

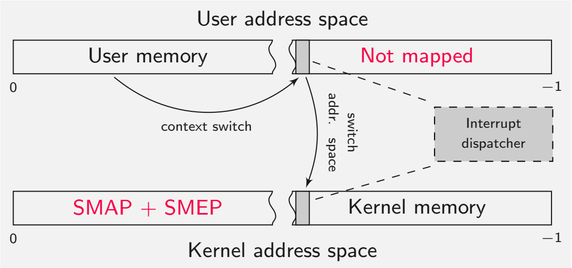
Without KAISER:



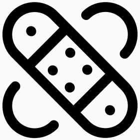
Without KAISER:

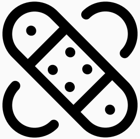


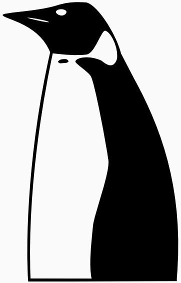
With KAISER:



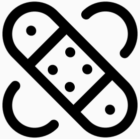
KAISER (Stronger Kernel Isolation) Patches

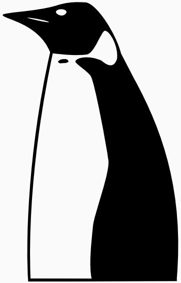






- Our patch
- Adopted in Linux





- Our patch
- Adopted in Linux

- Adopted in Windows

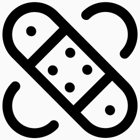
KAISER (Stronger Kernel Isolation) Patches

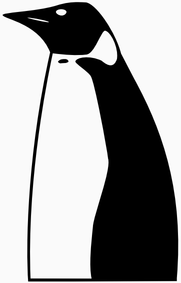
- Our patch
- Adopted in Linux

- Adopted in Windows

- Adopted in OSX/iOS

KAISER (Stronger Kernel Isolation) Patches





- Our patch
- Adopted in Linux

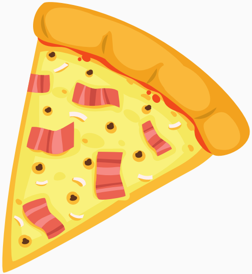
- Adopted in Windows

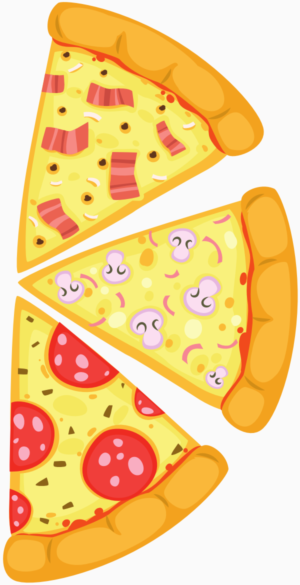
- Adopted in OSX/iOS

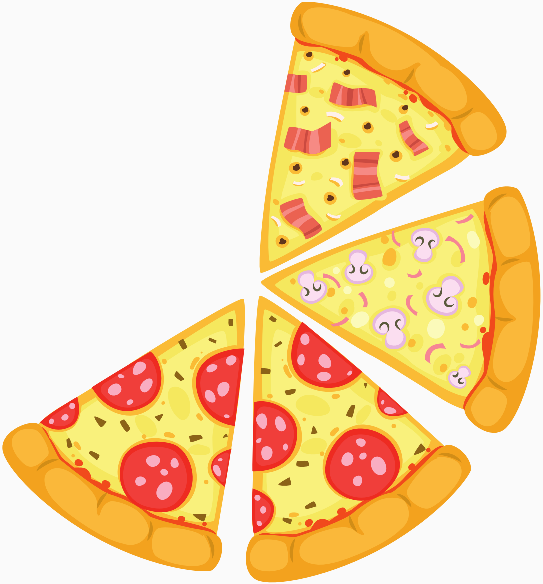
→ **now in every computer**

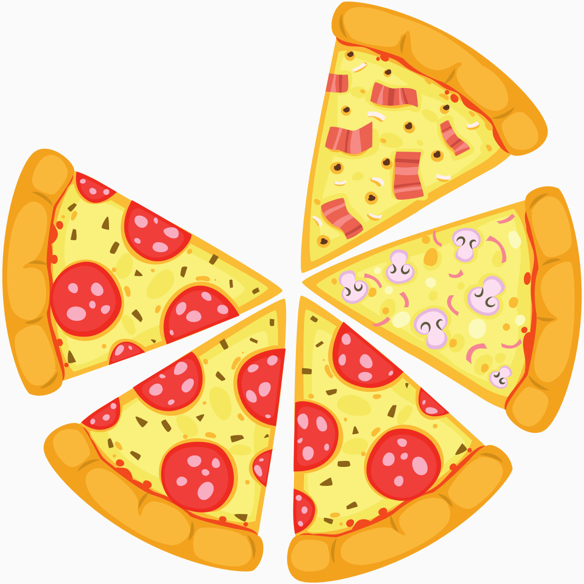
PIZZA

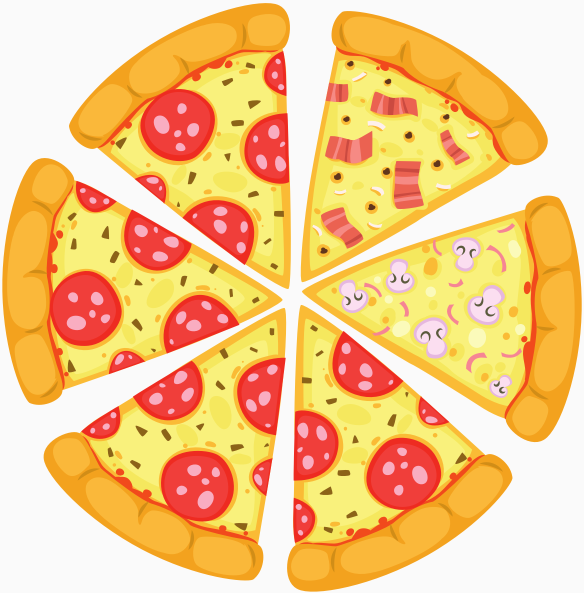
SPECIAL RECIPES












»A table for 6 please«

Speculative Cooking


»A table for 6 please«

PIZZA

SPECIAL RECIPES

PIZZA

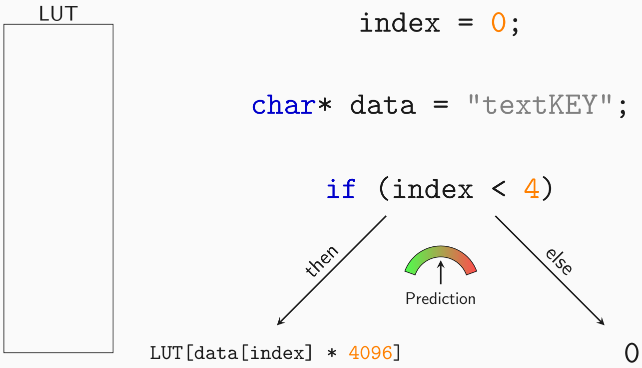
SPECIAL RECIPES

Pizza


PIZZA
SPECIAL RECIPES


Pizza


PIZZA
SPECIAL RECIPES
Pizza


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

Spectre-PHT (v1)

LUT

```
index = 0;

char* data = "textKEY";

if (index < 4)
```

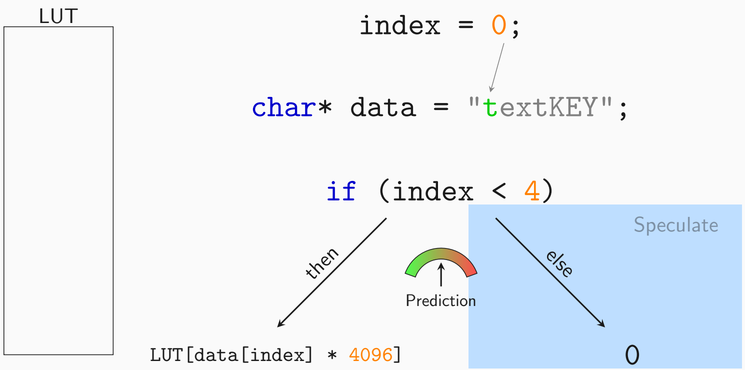
then

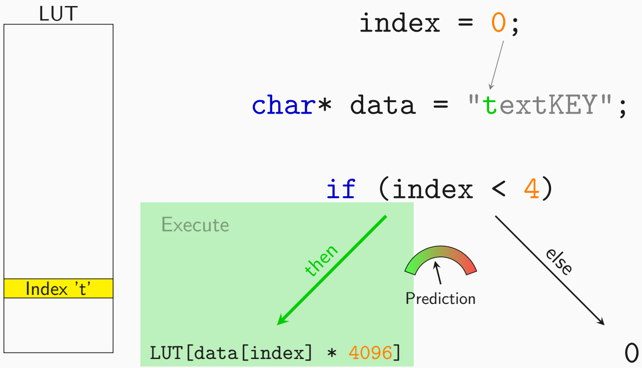
else

Prediction

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

0


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


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

LUT

```
index = 1;

char* data = "textKEY";

if (index < 4)
```

then

Prediction

else

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

0

Spectre-PHT (v1)

LUT

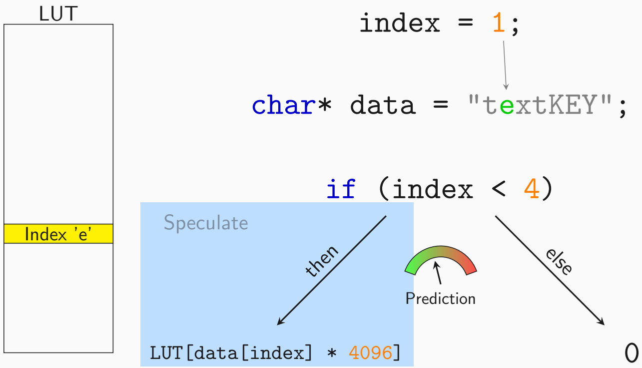
```
index = 1;

char* data = "textKEY";

if (index < 4)
```

then

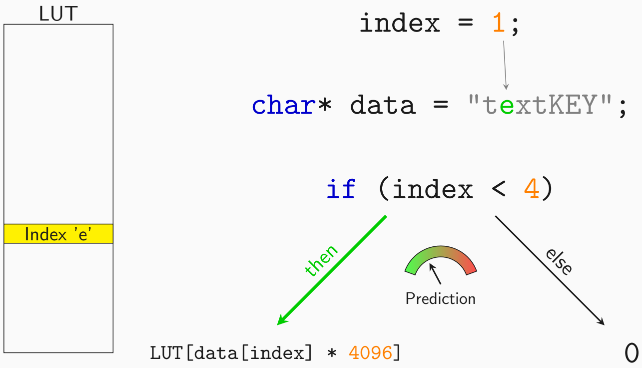
else

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

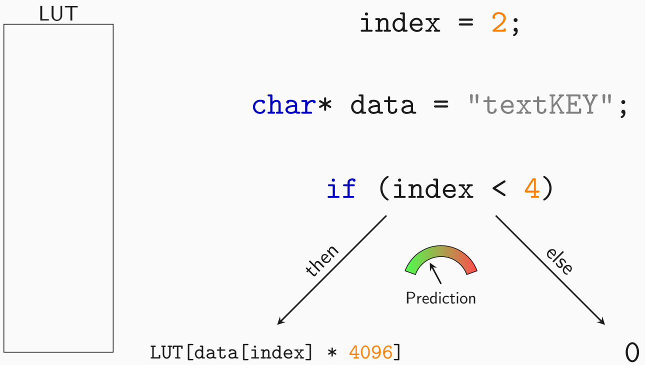
0

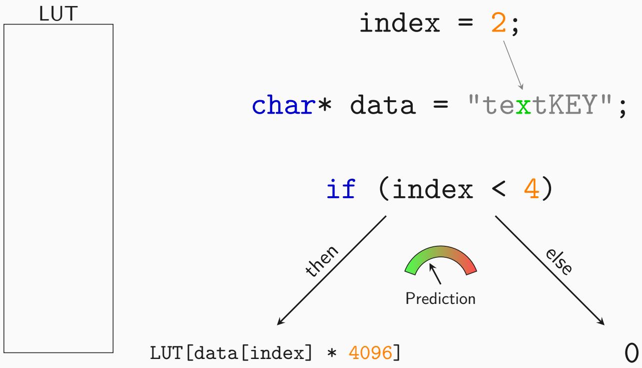

LUT
Index 'e'
index = 1;
char* data = "textKEY";
if (index < 4)
Speculate
then
else
Prediction
LUT[data[index] * 4096]
0

Spectre-PHT (v1)

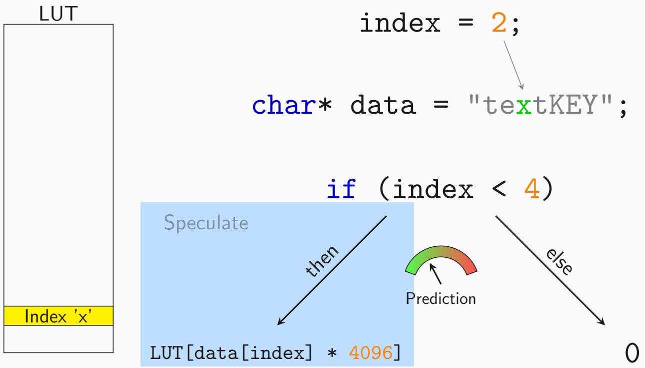


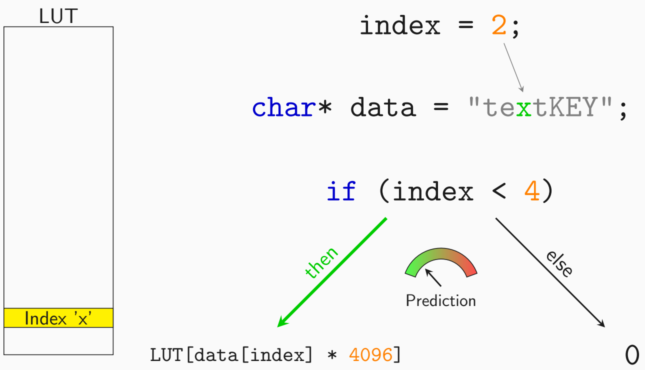
Spectre-PHT (v1)



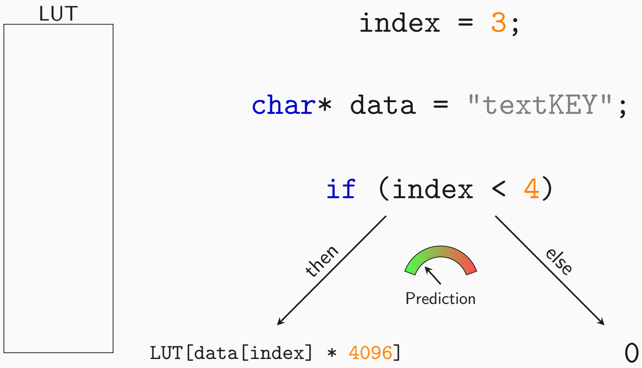

LUT
index = 2;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

Spectre-PHT (v1)

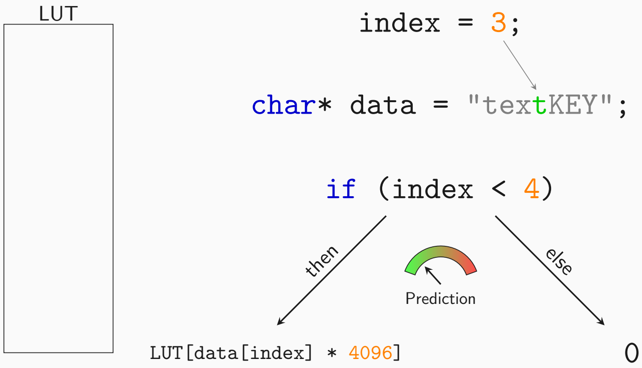




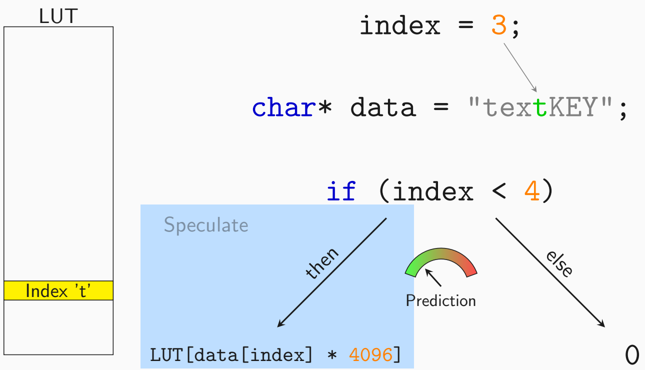
LUT
Index 'x'
index = 2;
char* data = "textKEY";
if (index < 4)
then
Prediction
else
LUT[data[index] * 4096]
0


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

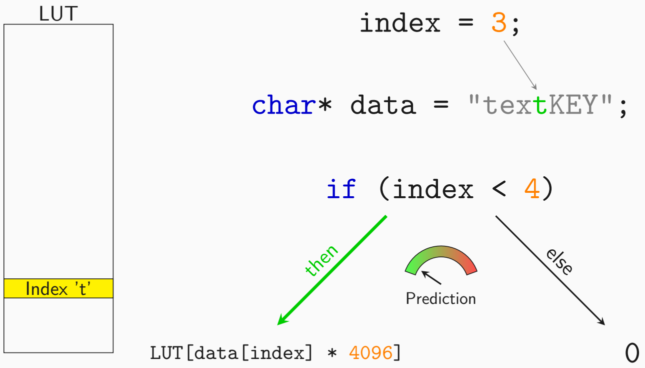
Spectre-PHT (v1)



Spectre-PHT (v1)



Spectre-PHT (v1)



LUT

```
index = 4;

char* data = "textKEY";

if (index < 4)
```

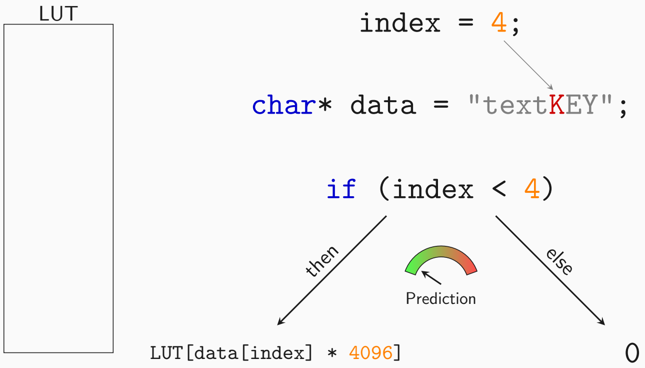
then

else

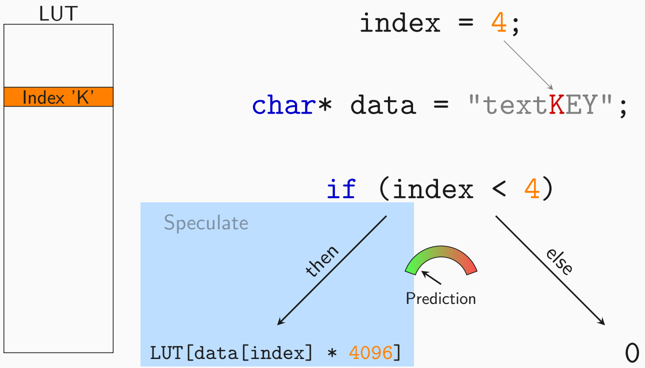

Prediction

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

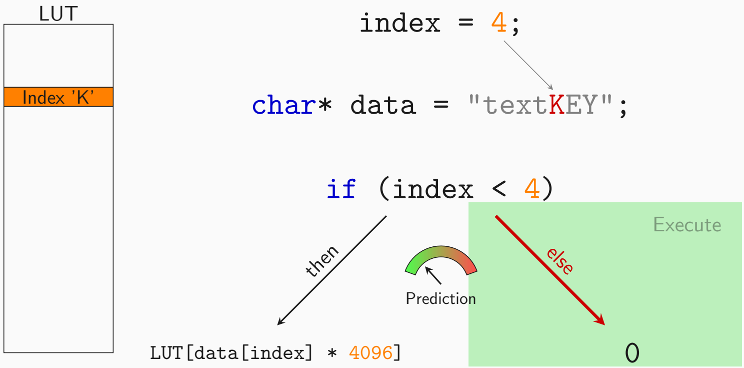
0


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

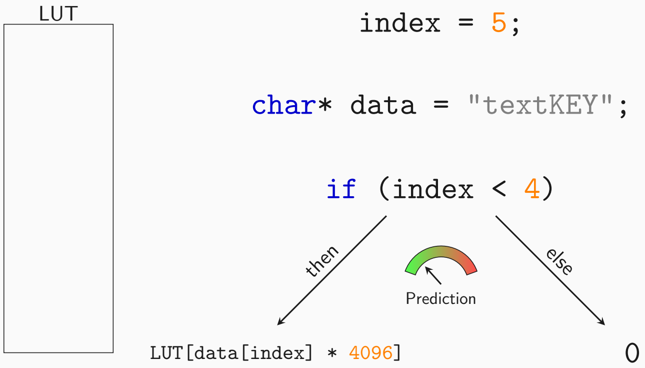
Spectre-PHT (v1)



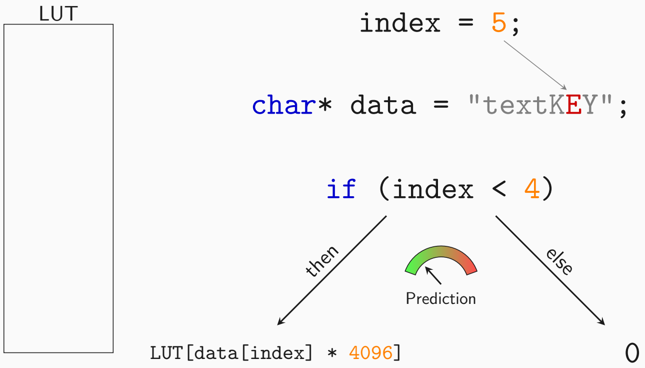
Spectre-PHT (v1)



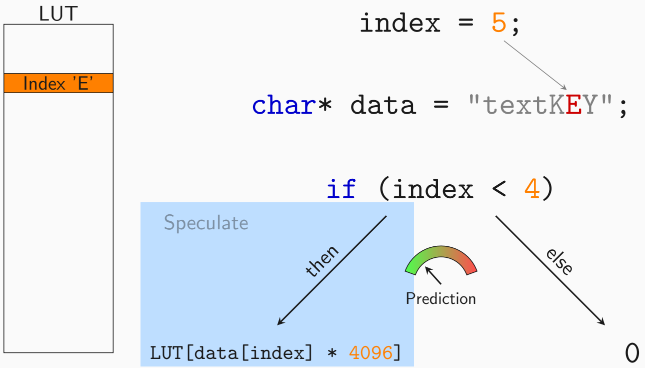
Spectre-PHT (v1)



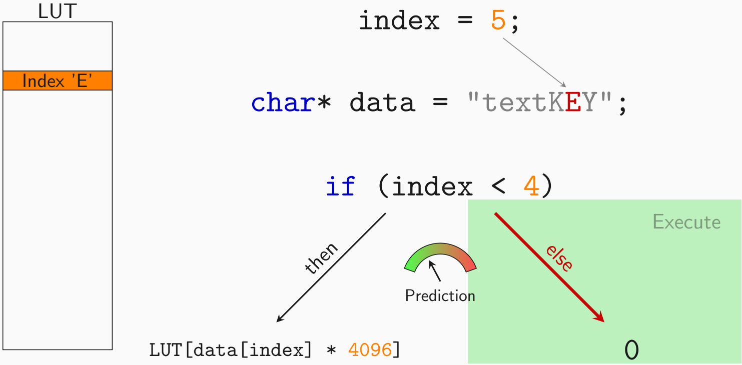
Spectre-PHT (v1)



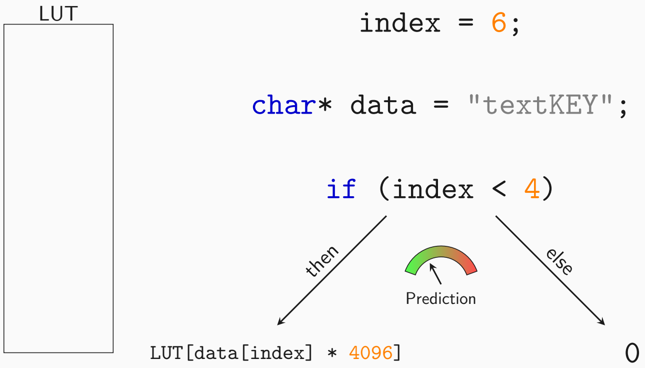
Spectre-PHT (v1)

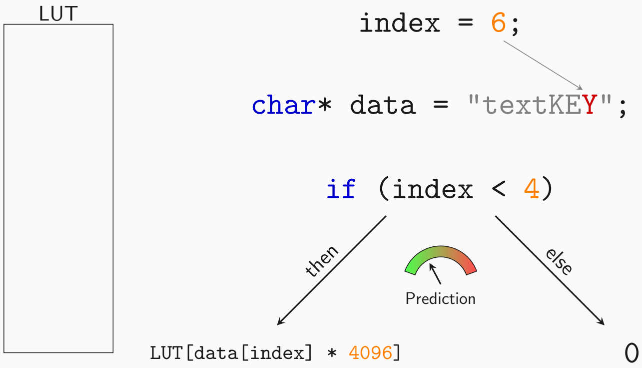


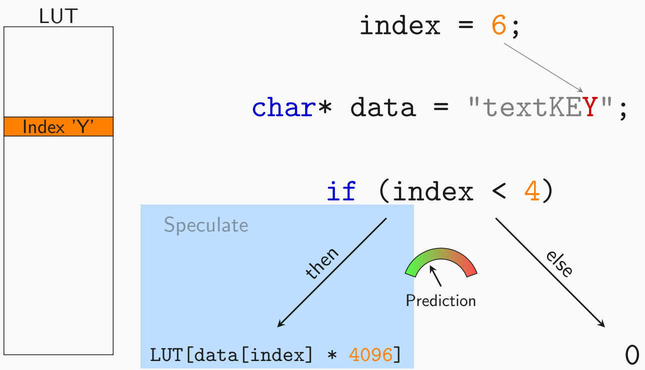
Spectre-PHT (v1)



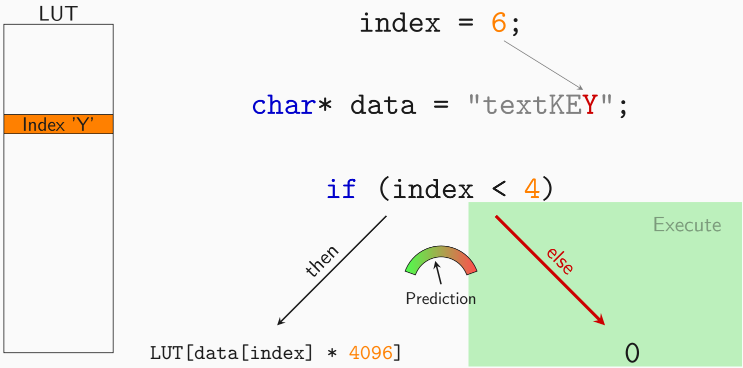
```
index = 6;

char* data = "textKEY";

if (index < 4)
```

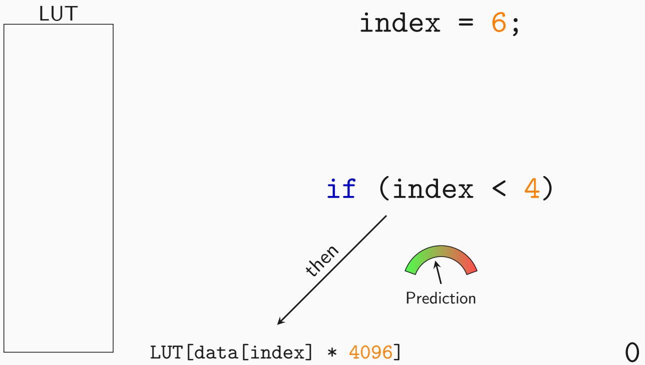



LUT
index = 6;
char* data = "textKEY";
if (index < 4)
then
Prediction
else
LUT[data[index] * 4096]
0


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

Spectre-PHT (v1)





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

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

```
a->move()
```

fly()

Prediction

swim()

```
LUT[data[a->m] * 4096]
```

0

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

```
a->move()
```

fly()

swim()

Prediction

Speculate

```
LUT[data[a->m] * 4096]
```

0

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

```
a->move()
```

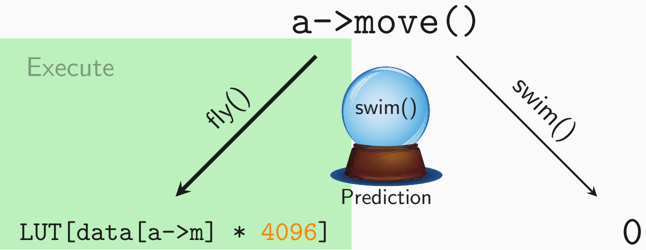
fly() swim()

Prediction

```
LUT[data[a->m] * 4096]
```

0

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



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

```
a->move()
```

fly()

swim()

Prediction

```
LUT[data[a->m] * 4096]
```

0

Spectre v2

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

```
a->move()
```

Speculate

fly()

swim()

Prediction

```
LUT[data[a->m] * 4096]
```

0

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

```
a->move()
```

fly()

swim()

Prediction

```
LUT[data[a->m] * 4096]
```

0

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

```
a->move()
```

fly()

swim()

Prediction

```
LUT[data[a->m] * 4096]
```

0

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

```
a->move()
```

Speculate

fly()

Prediction

swim()

```
LUT[data[a->m] * 4096]
```

0

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

```
a->move()
```

fly() swim()

Prediction

```
LUT[data[a->m] * 4096]
```

0

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

```
a->move()
```

fly()

Prediction

swim()

Execute

```
LUT[data[a->m] * 4096]
```

0

Spectre v2

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

```
a->move()
```

fly()

swim()

Prediction

```
LUT[data[a->m] * 4096]
```

0

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

```
a->move()
```

Prediction

```
LUT[data[a->m] * 4096]
```

0

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

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

```
a->move()
```

fly() → `LUT[data[a->m] * 4096]`

swim() → 0

Prediction

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

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

“Speculative Buffer Overflows”[6]

- v1.1: Speculatively write to memory locations

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

"Speculative Buffer Overflows"[6]

- v1.1: Speculatively write to memory locations

→ Many more gadgets than previously anticipated n

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

“Speculative Buffer Overflows”[6]

- v1.1: Speculatively write to memory locations

→ Many more gadgets than previously anticipated n

- v1.2: Ignore writable bit

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

"Speculative Buffer Overflows"[6]

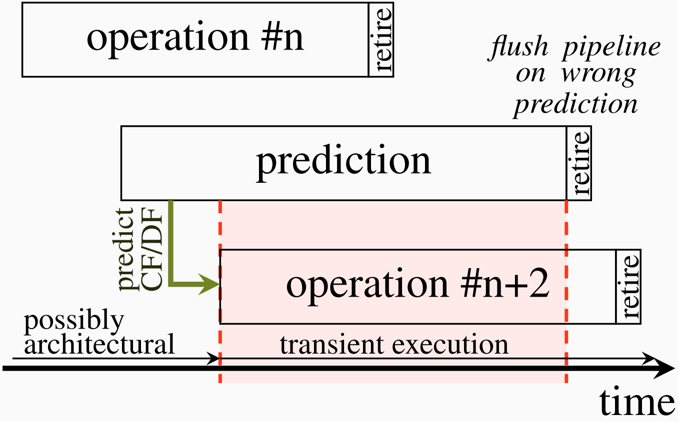
- v1.1: Speculatively write to memory locations

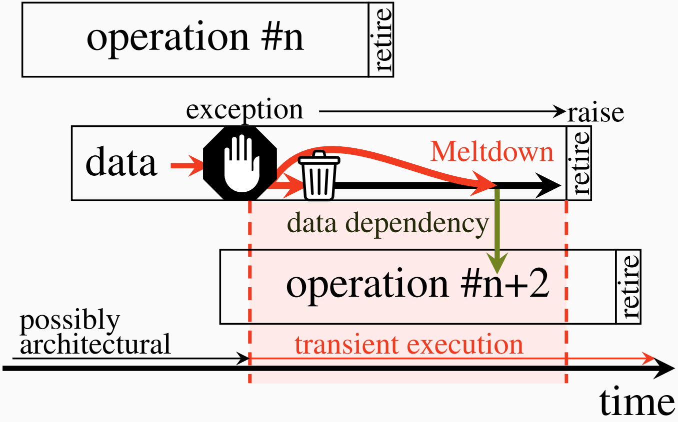
→ Many more gadgets than previously anticipated n

- v1.2: Ignore writable bit

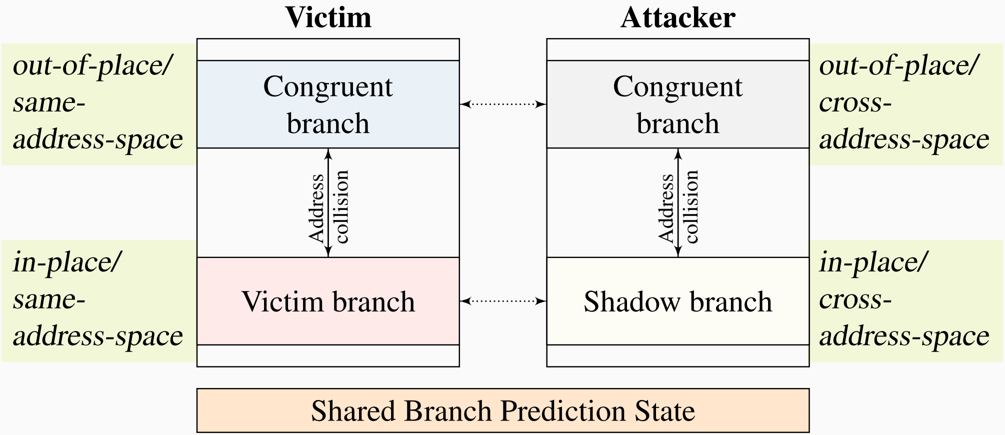
→ = Meltdown-RW

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

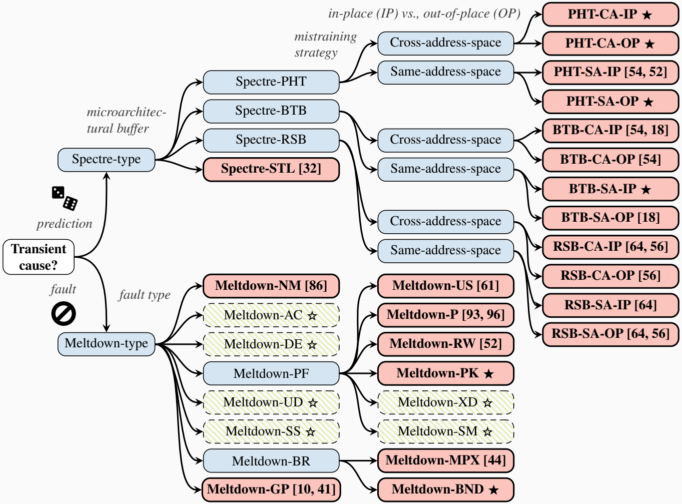

operation #n
retire
flush pipeline on wrong prediction
prediction
retire
predict CF/DF
operation #n+2
retire
possibly architectural
transient execution
time


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

Mistraining Location



Classification Tree



Mitigations?


BLOCKCHAIN
HD
HISTORY.COM

ACM SIGARCH

JOIN BENEFIT CONTRIBUTE DISCOVER BLOG

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	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	□	□	□	◇	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	■	●

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

Impact / **Defense**	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 ;)

https://www.imdb.com/search/title/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=b9121fa8-b7bb-4a3e-8887-aab822e0b5a7&pf_rd_r=QWCGAY

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
ABIGAIL
BRESLIN
ZOMBIELAND
NUT UP OR SHUT UP
IN THEATERS OCTOBER 9
Follow us @Zombieland on Twitter

ZombieLoad Attack

When the kernel address is loaded in line 4,
that the CPU already issued the subsequent i
as part of the out-of-order execution, and tha
responding μOPs wait in the reservation stat
content of the kernel address to arrive. As s

Google

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

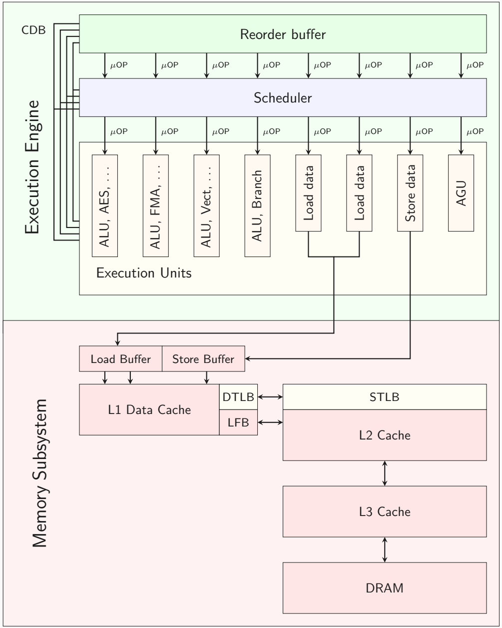
19.05.2007 - by CaptBilly1Eye · 12 years ago In reply to Toshiba Boot **Error** ... partition on the floppy disk, hard drive or a CD ROM to **load** the **operating** system. ... prior to this situation starting to **occur**, or if you find that the boot sequence already has the ... Leave the notebook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

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

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

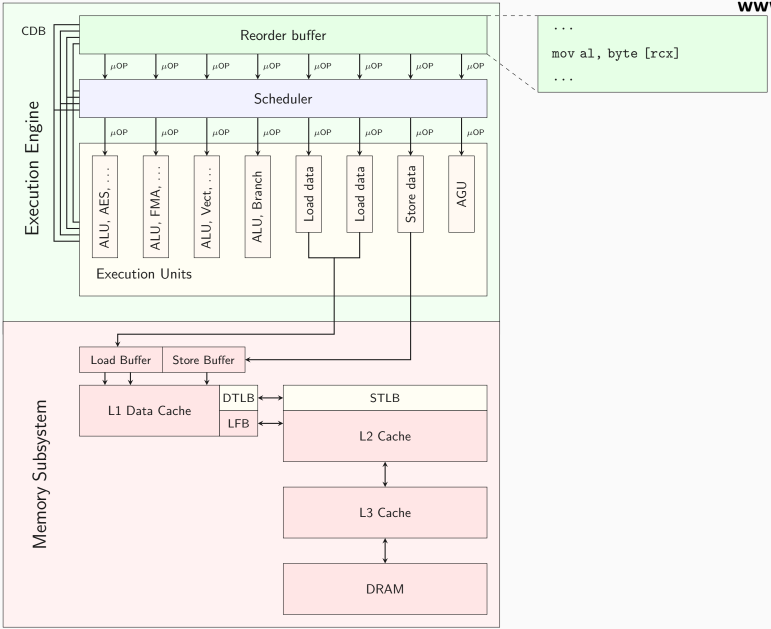
Application filed by **Intel Corp** ... Hence, a functional unit may often **complete** a first instruction (which logically precedes a second instruction in the ..... If a **fault occurs** with respect to the **LOAD operation**, it is marked as valid and **completed**.

Meltdown



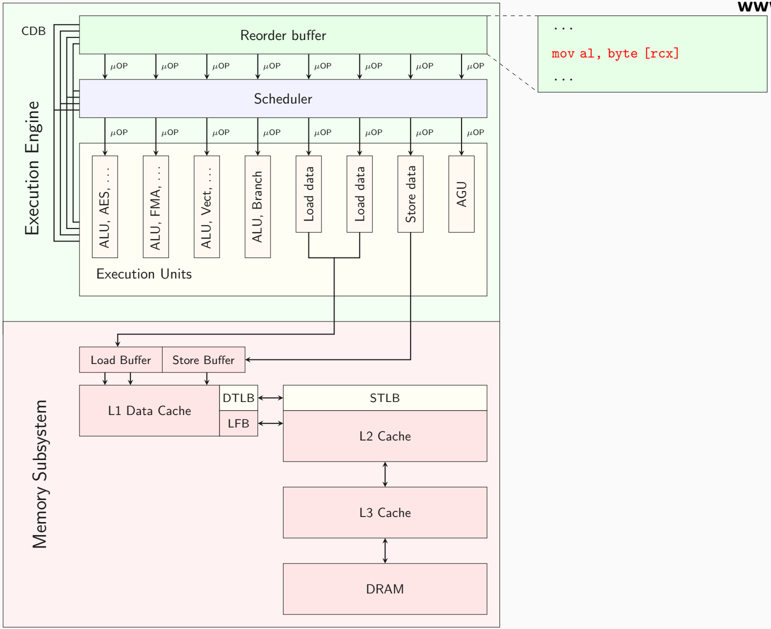
CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Memory Subsystem
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM

Meltdown

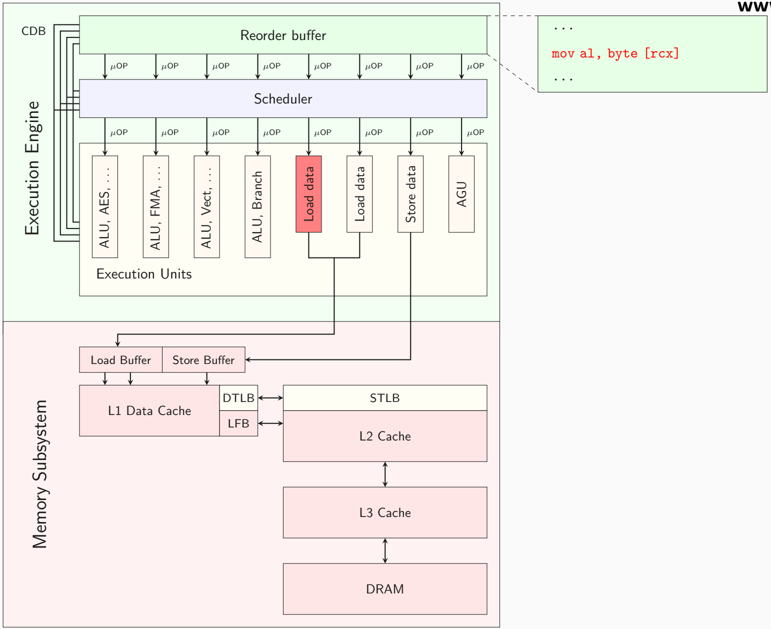


CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Memory Subsystem
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
...
mov al, byte [rcx]
...

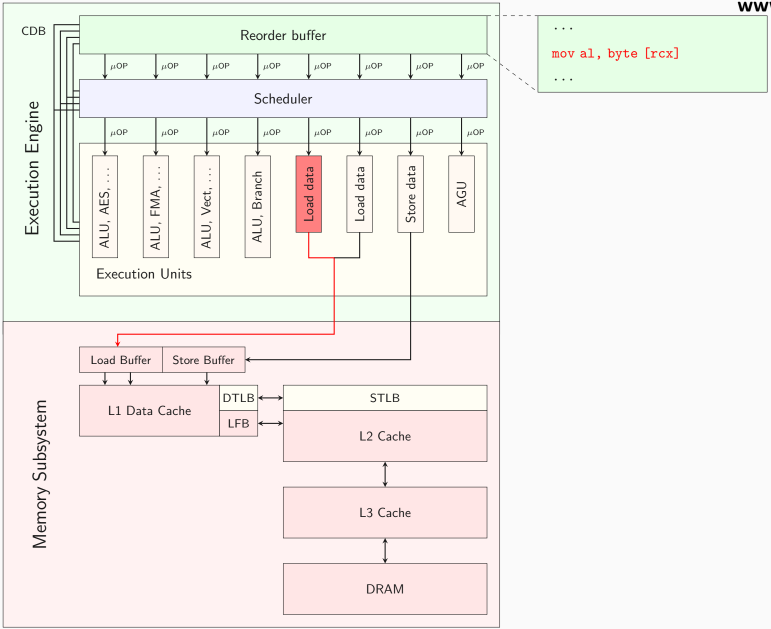
Meltdown


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

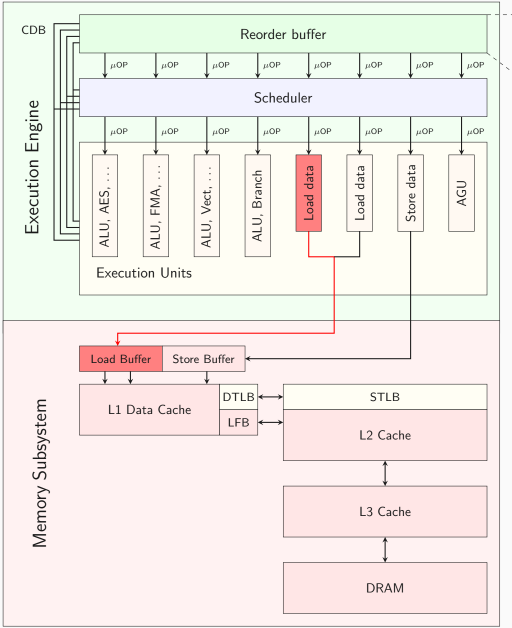
Meltdown


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

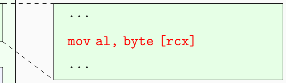
Meltdown


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

Meltdown

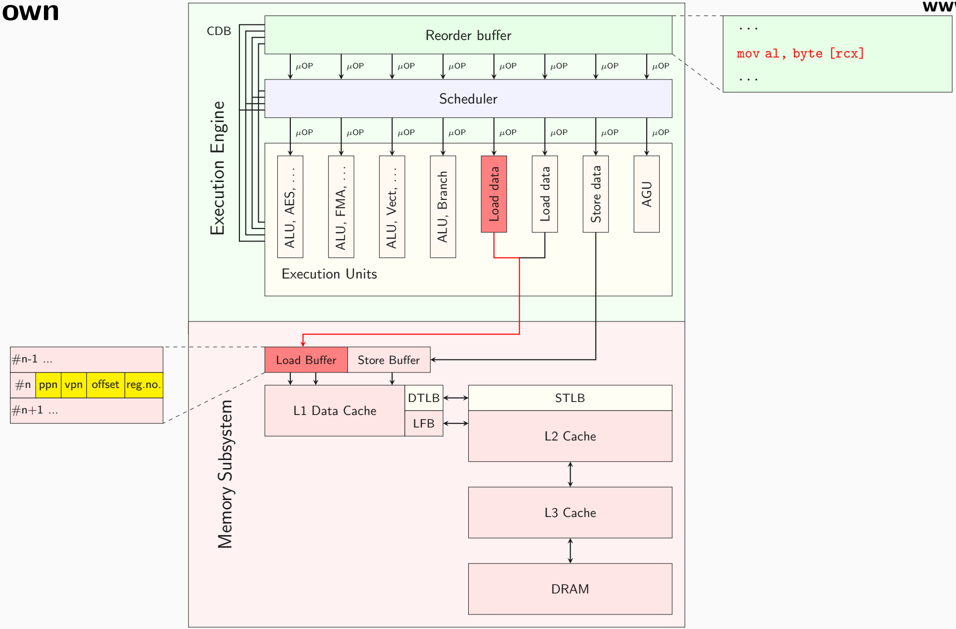


CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem

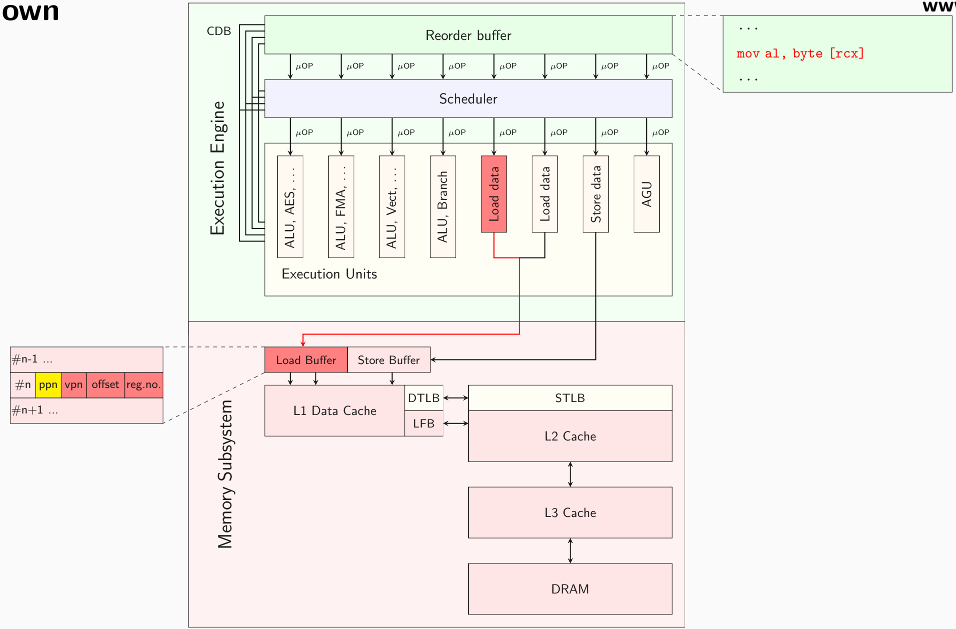


...
mov al, byte [rcx]
...

Meltdown

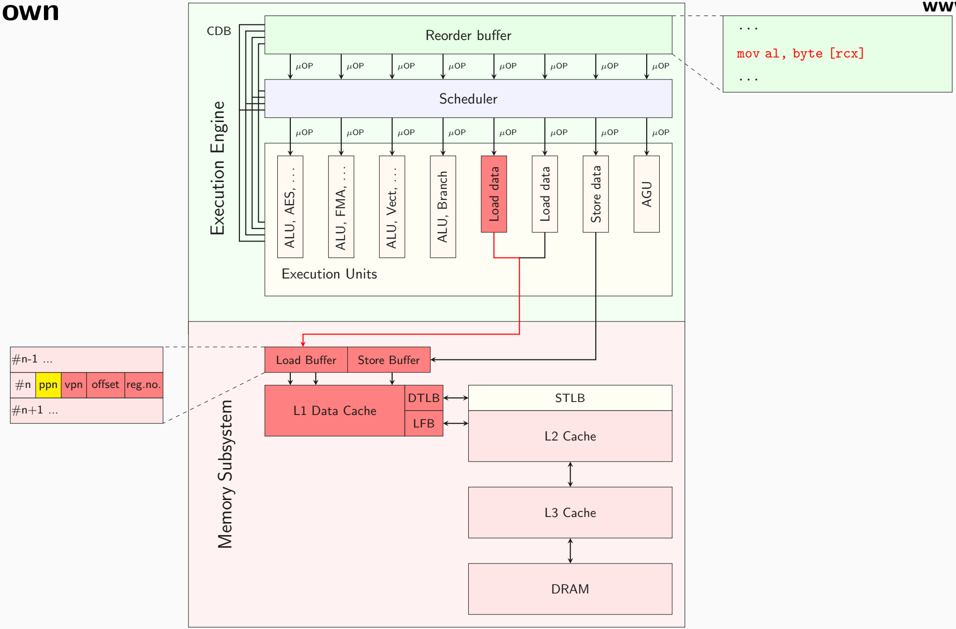

CDB
Reorder buffer
...
mov al, byte [rcx]
...
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM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

Meltdown



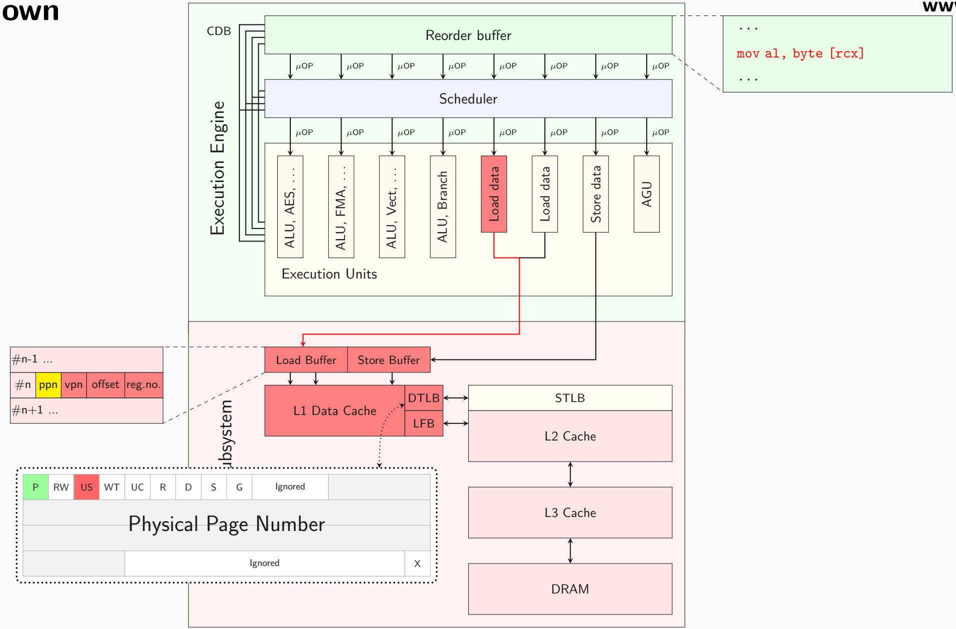
CDB
Reorder buffer
µOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
...
mov al, byte [rcx]
...
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM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

Meltdown



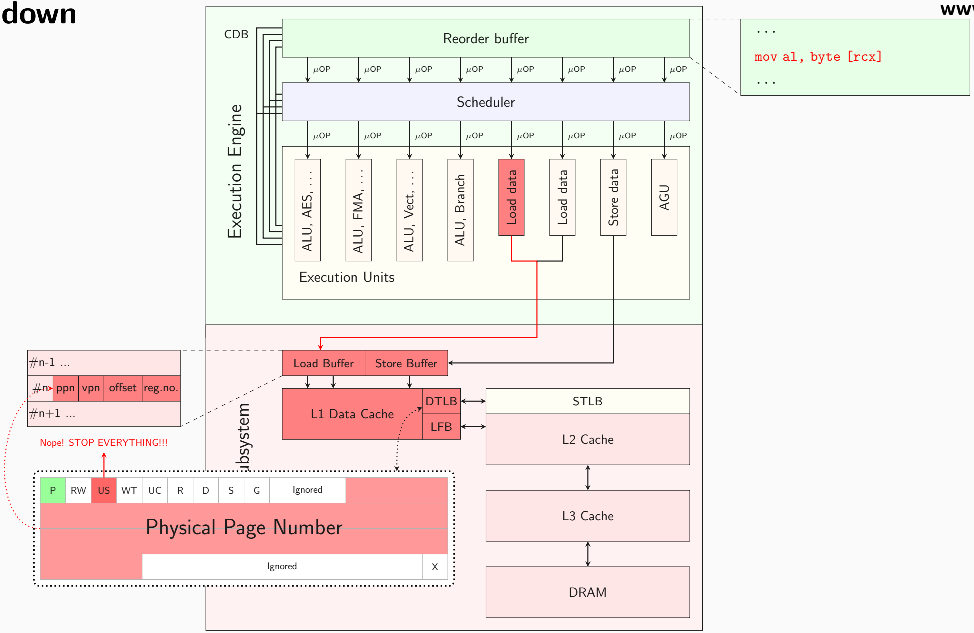
CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem
...
mov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

Meltdown

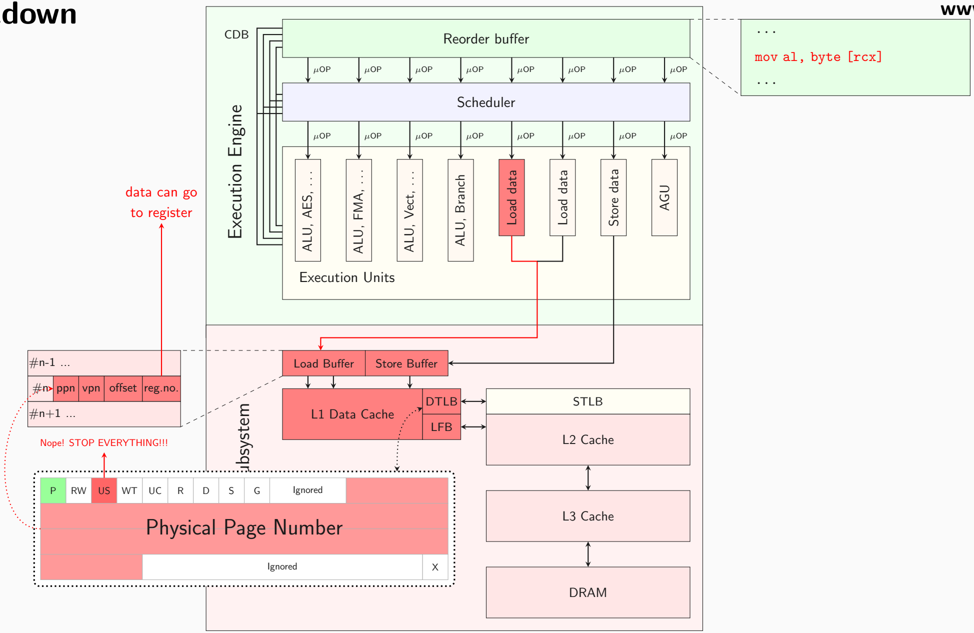


CDB
Reorder buffer
...
mov al, byte [rcx]
...
µOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM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

Meltdown

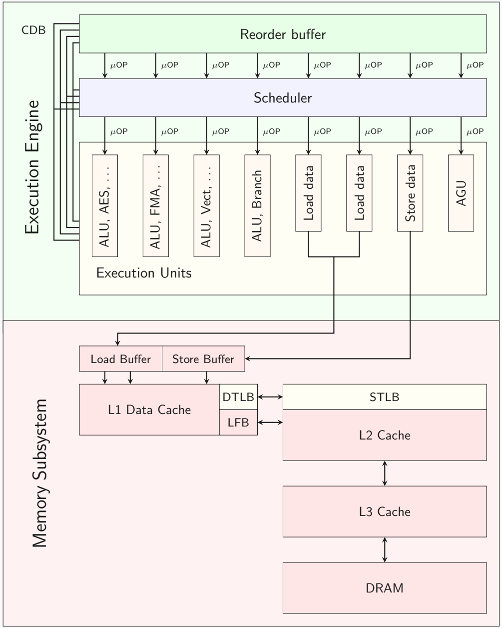

CDB
Reorder buffer
µOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
...
mov al, byte [rcx]
...
#n-1 ...
#n ppn vpn offset reg.no.
#n+1 ...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM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

Meltdown

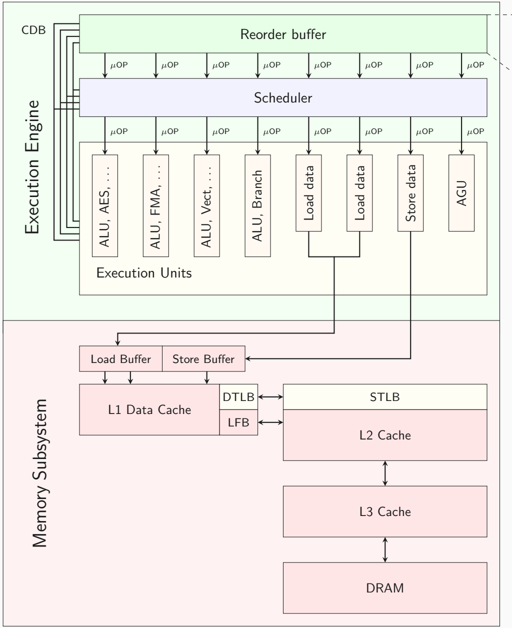


...
mov al, byte [rcx]
...
CDB
Reorder buffer
µOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
data can go
to register
Load Buffer
Store Buffer
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
L1 Data Cache
DTLB
LFB
STLB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

Foreshadow-VMM

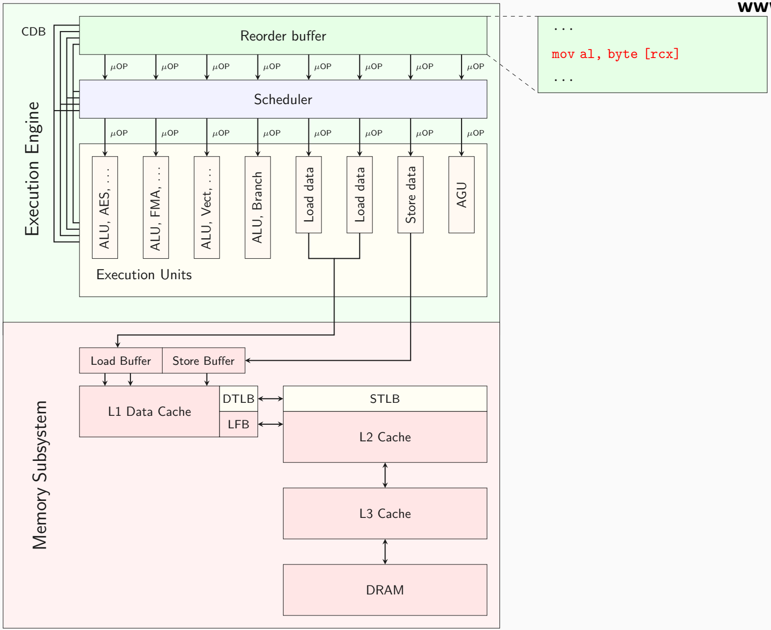


Foreshadow-VMM

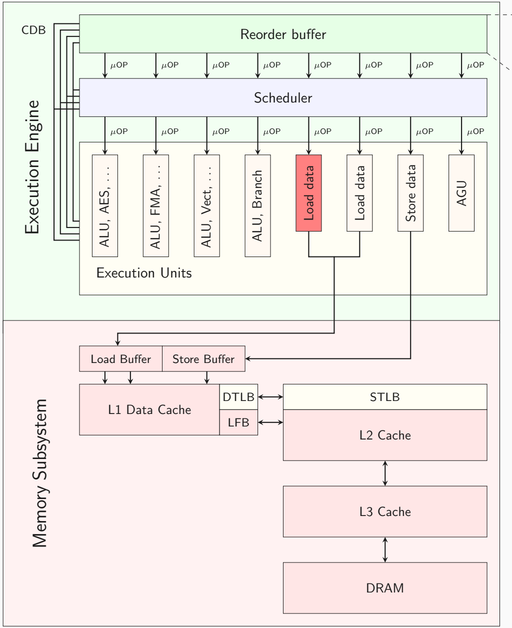



```
...
mov al, byte [rcx]
...
```

Foreshadow-VMM

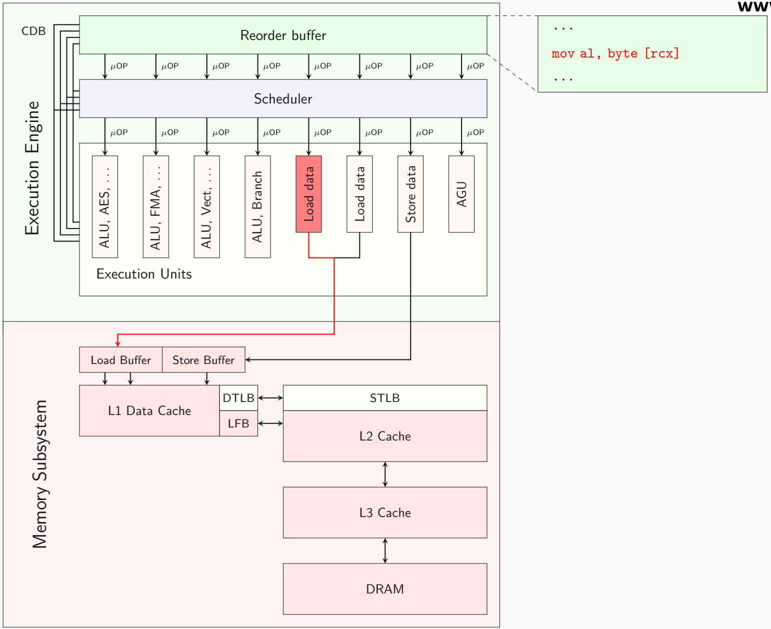


Foreshadow-VMM

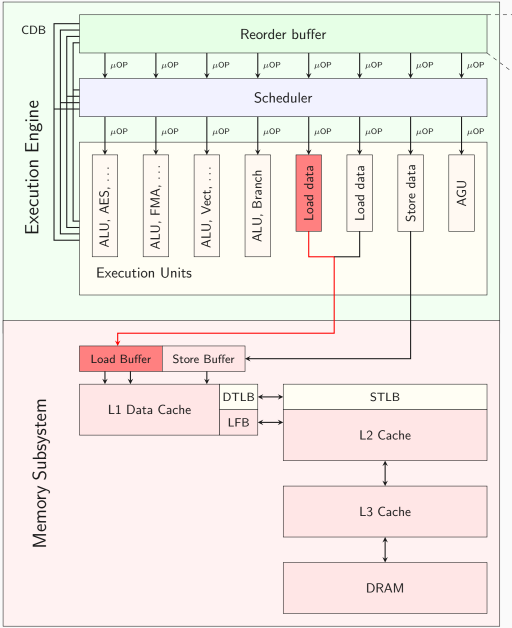



```
...
mov al, byte [rcx]
...
```

Foreshadow-VMM

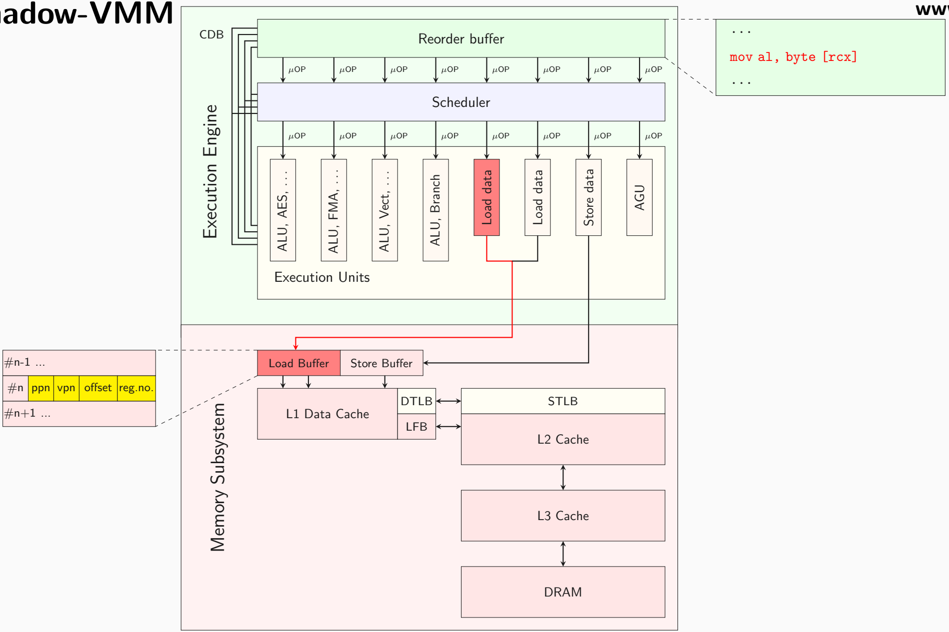


Foreshadow-VMM

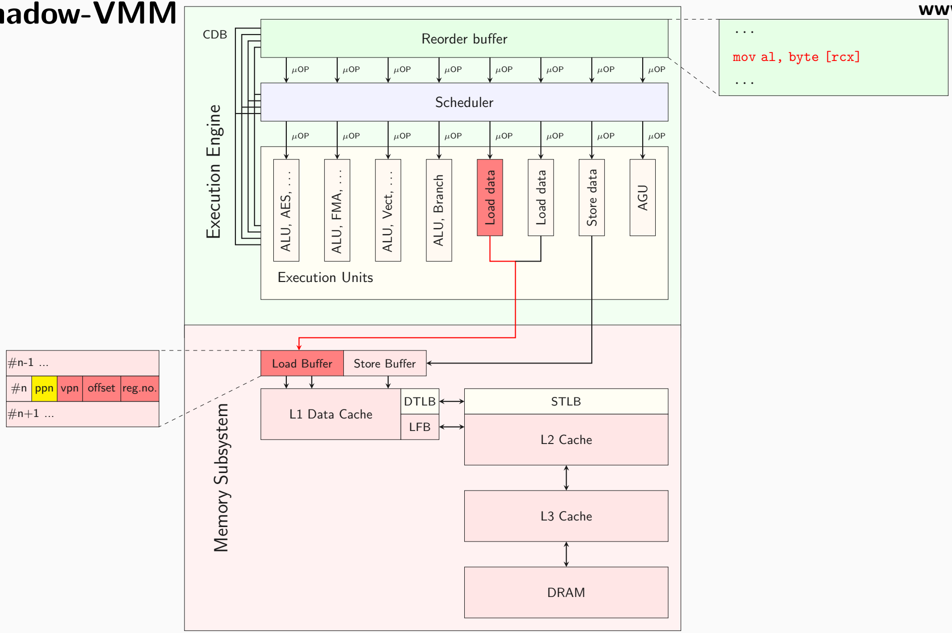



```
...
mov al, byte [rcx]
...
```

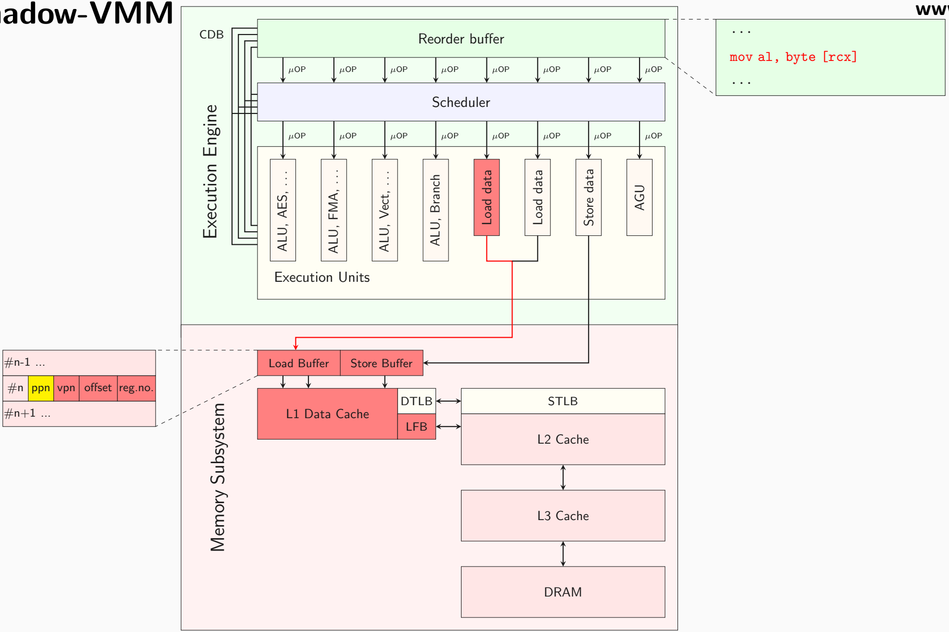
Foreshadow-VMM



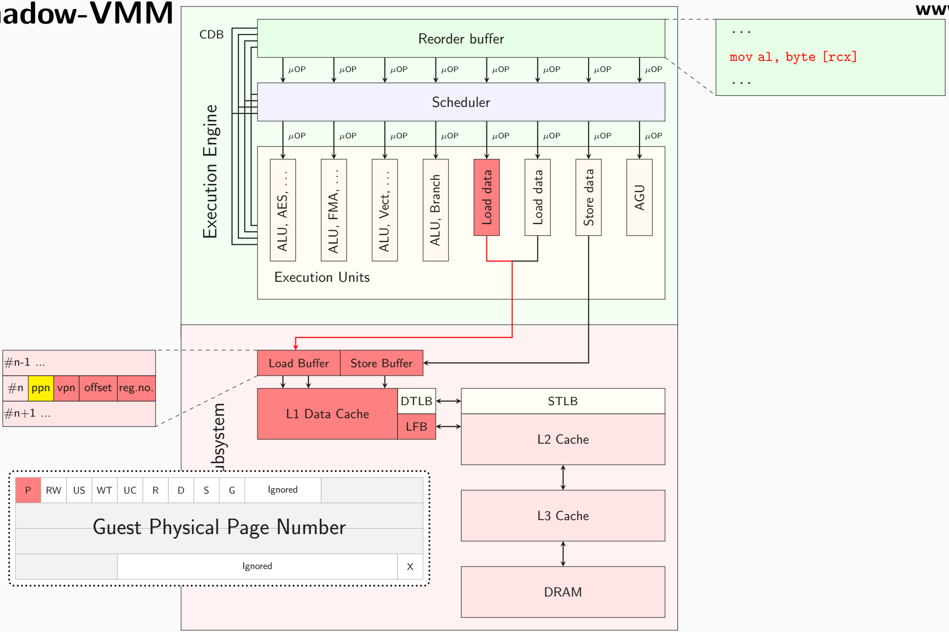
Foreshadow-VMM



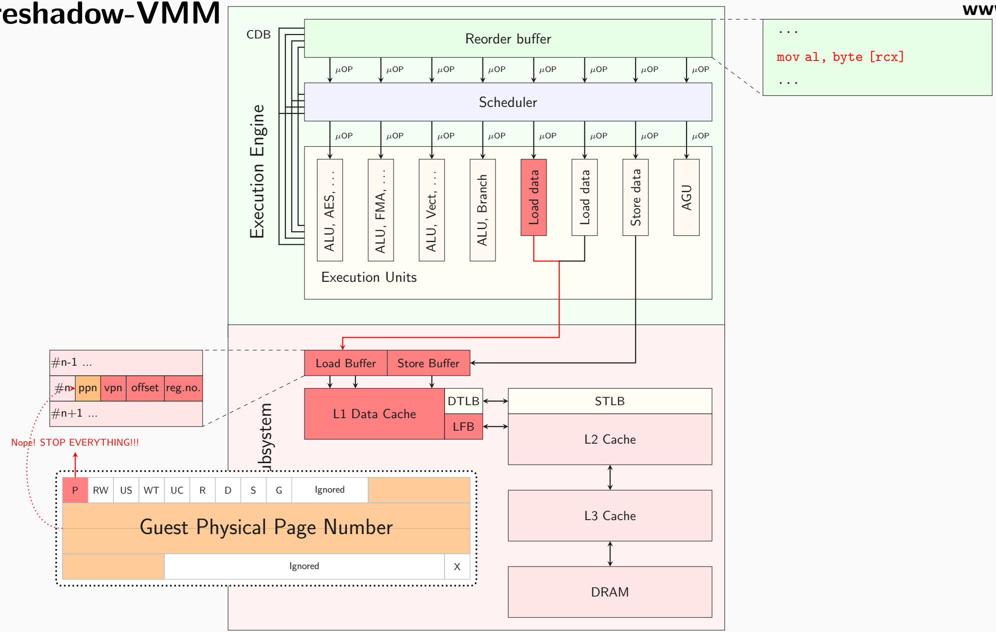
Foreshadow-VMM



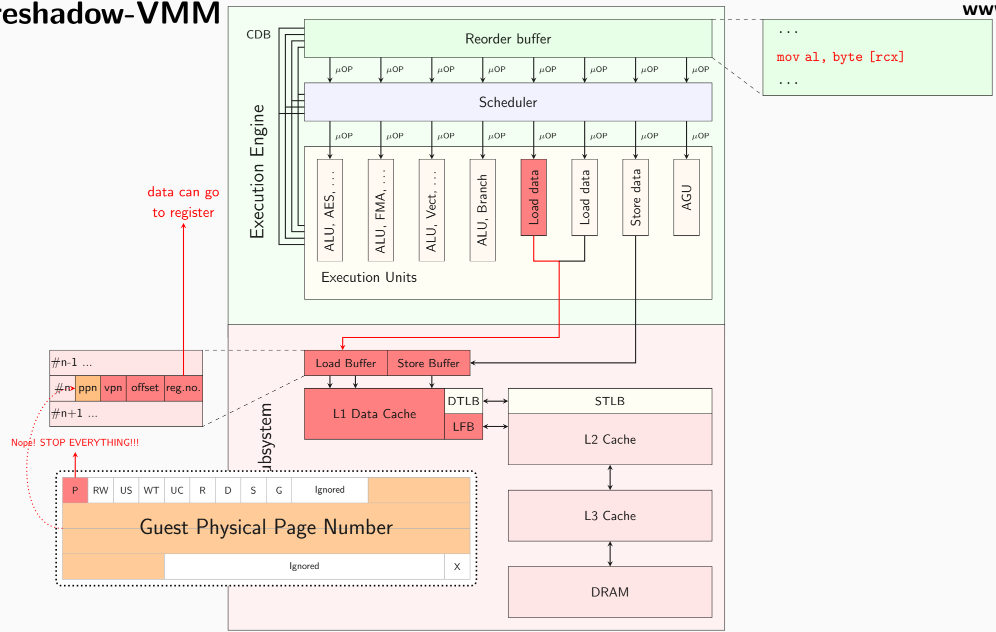
Foreshadow-VMM



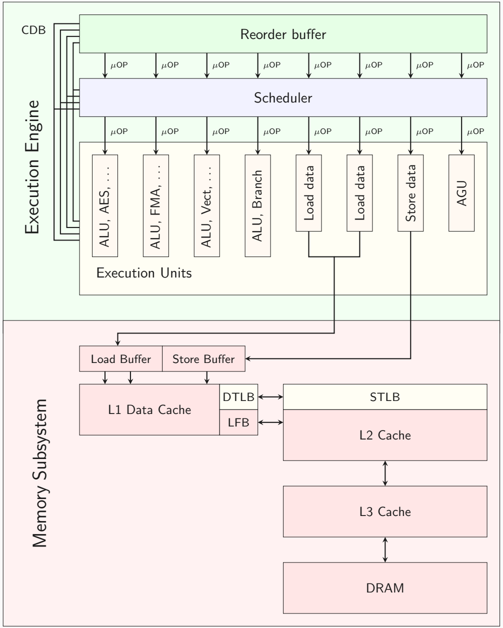
Foreshadow-VMM



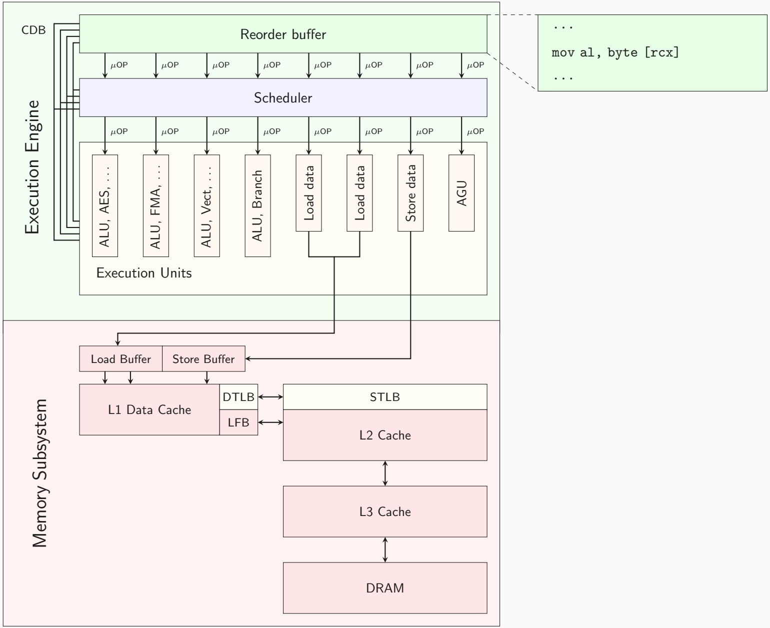
Foreshadow-VMM



ZombieLoad

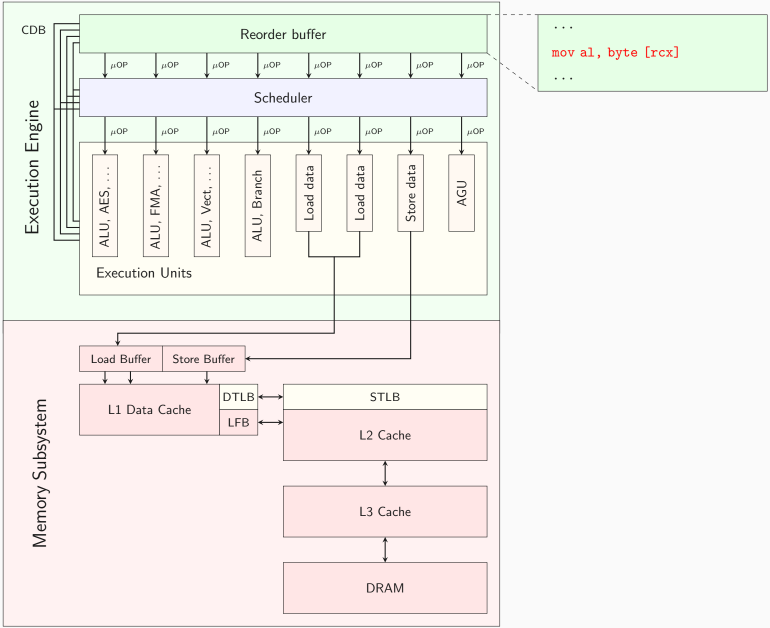

CDB
Reorder buffer
µOP
Scheduler
µOP
Execution Engine
ALU, AES, . . .
ALU, FMA, . . .
ALU, Vect, . . .
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem

ZombieLoad



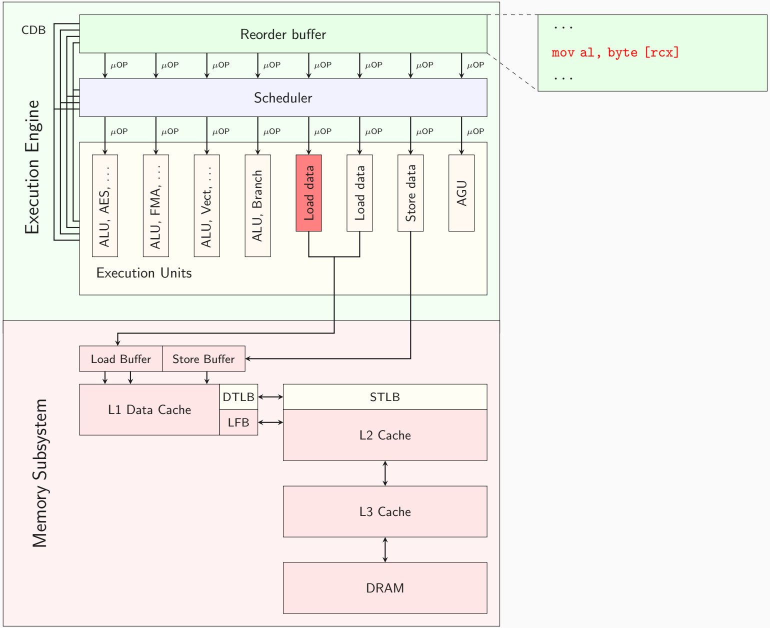
CDB
Reorder buffer
µOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Memory Subsystem
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
...
mov al, byte [rcx]
...

ZombieLoad



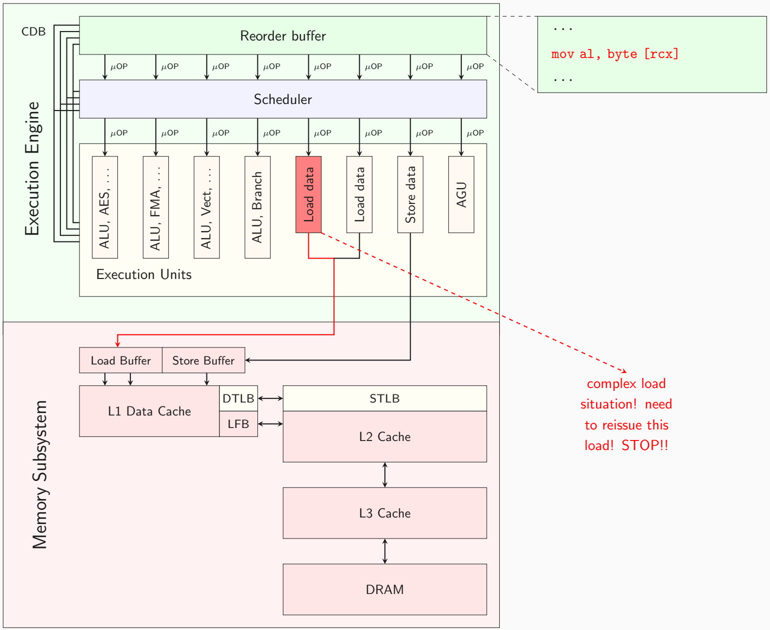
CDB
Reorder buffer
µOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem
...
mov al, byte [rcx]
...

ZombieLoad

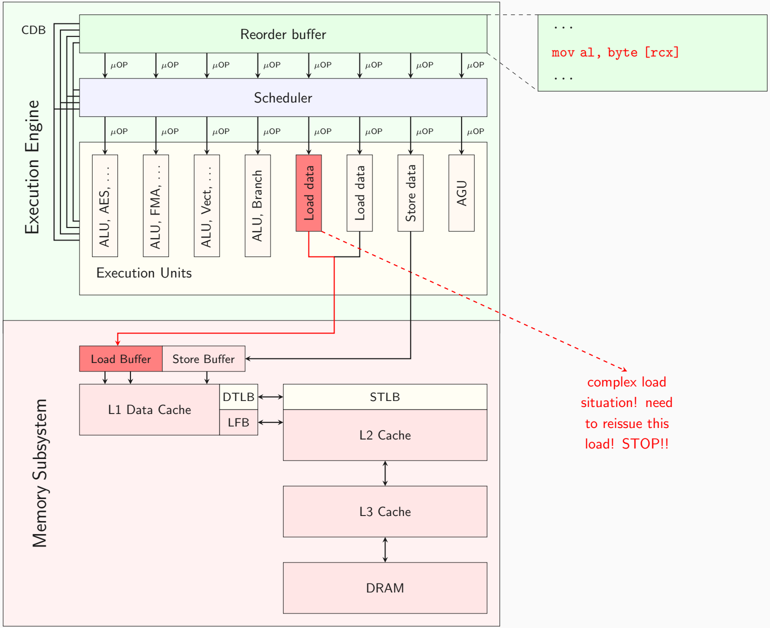


CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, . . .
ALU, FMA, . . .
ALU, Vect, . . .
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Memory Subsystem
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
. . .
mov al, byte [rcx]
. . .

ZombieLoad

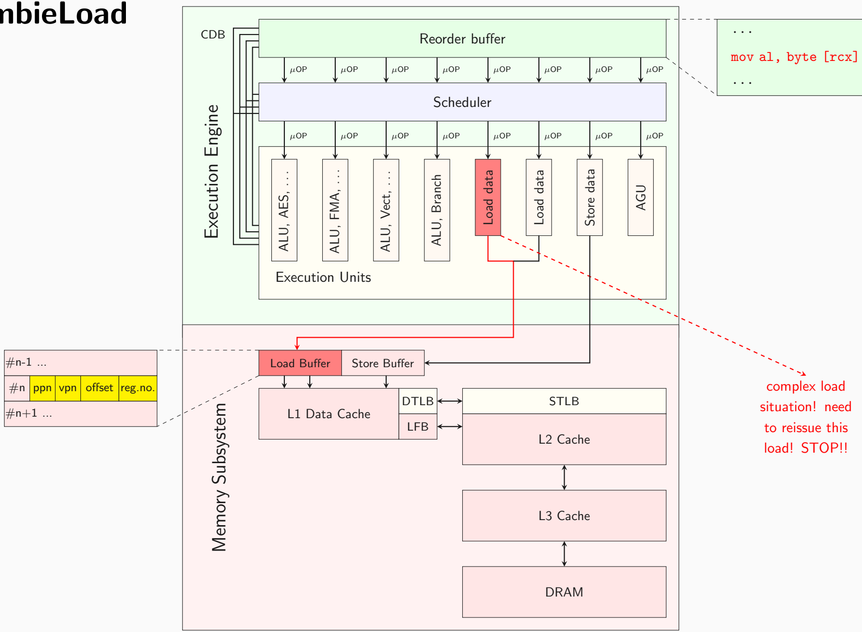

CDB
Reorder buffer
µOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Memory Subsystem
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
...
mov al, byte [rcx]
...
complex load situation! need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

ZombieLoad



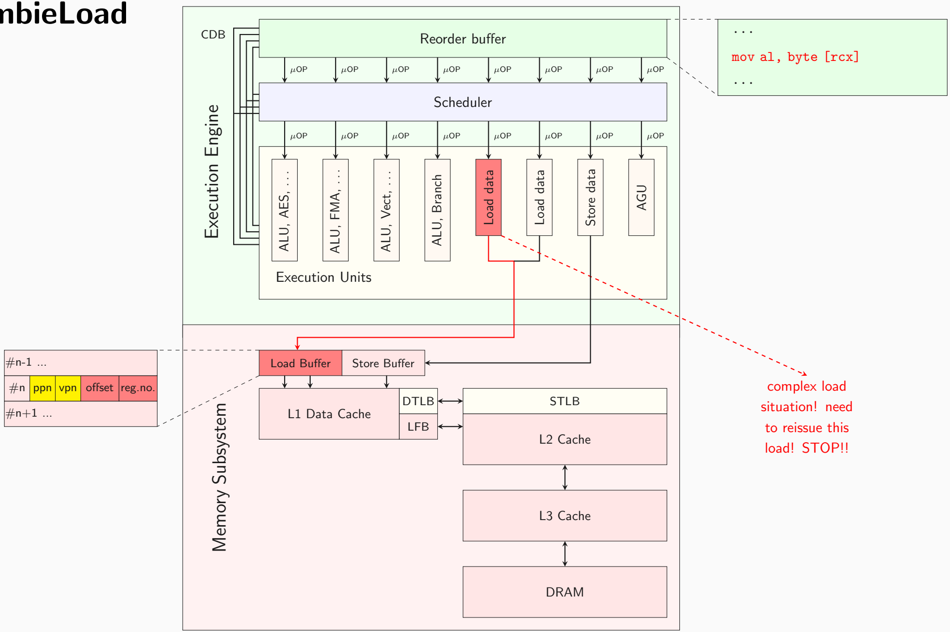
CDB
Reorder buffer
µOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Memory Subsystem
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
...
mov al, byte [rcx]
...
complex load situation! need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

ZombieLoad

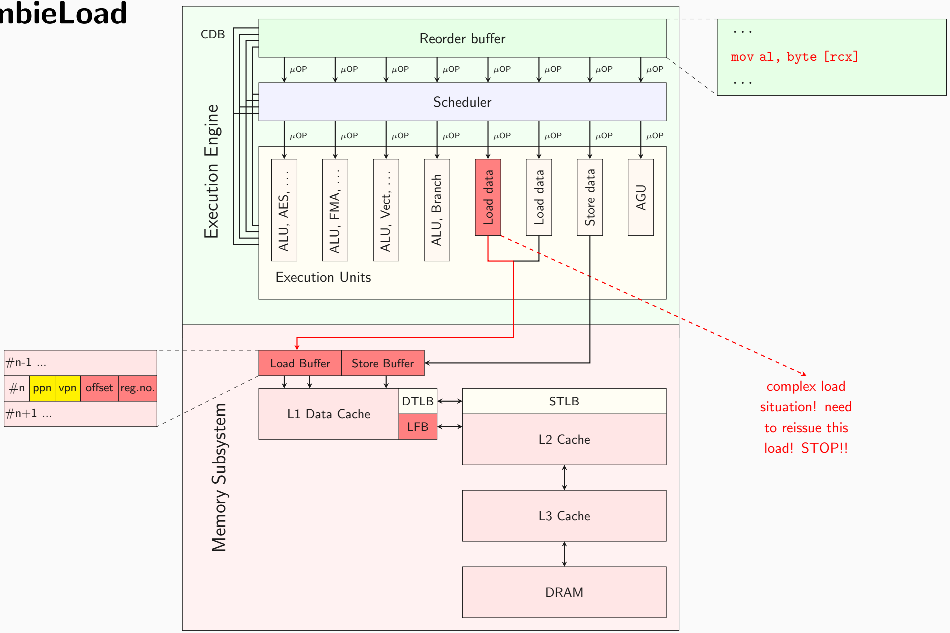



```
...
mov al, byte [rcx]
...
```

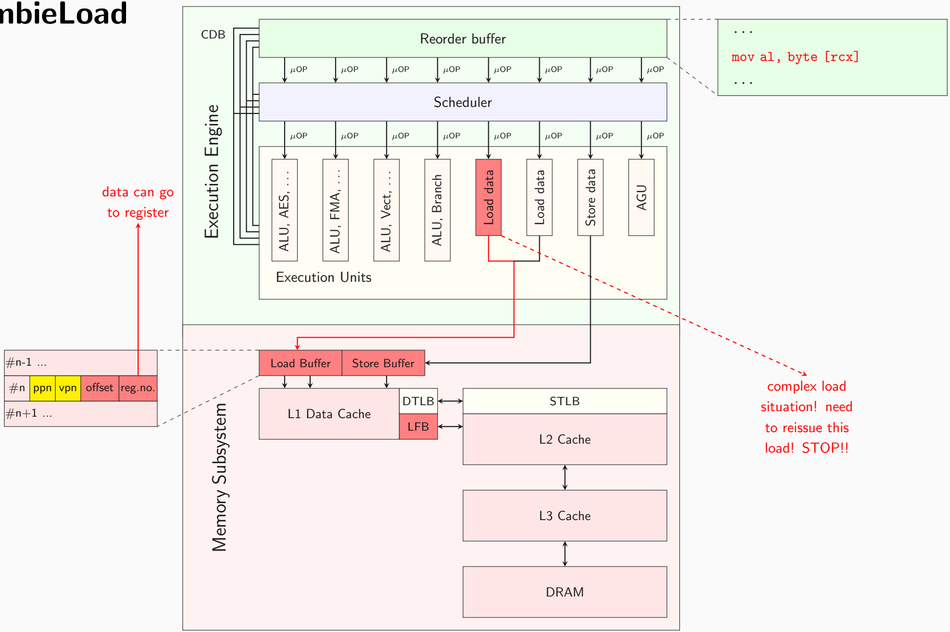
ZombieLoad


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

ZombieLoad

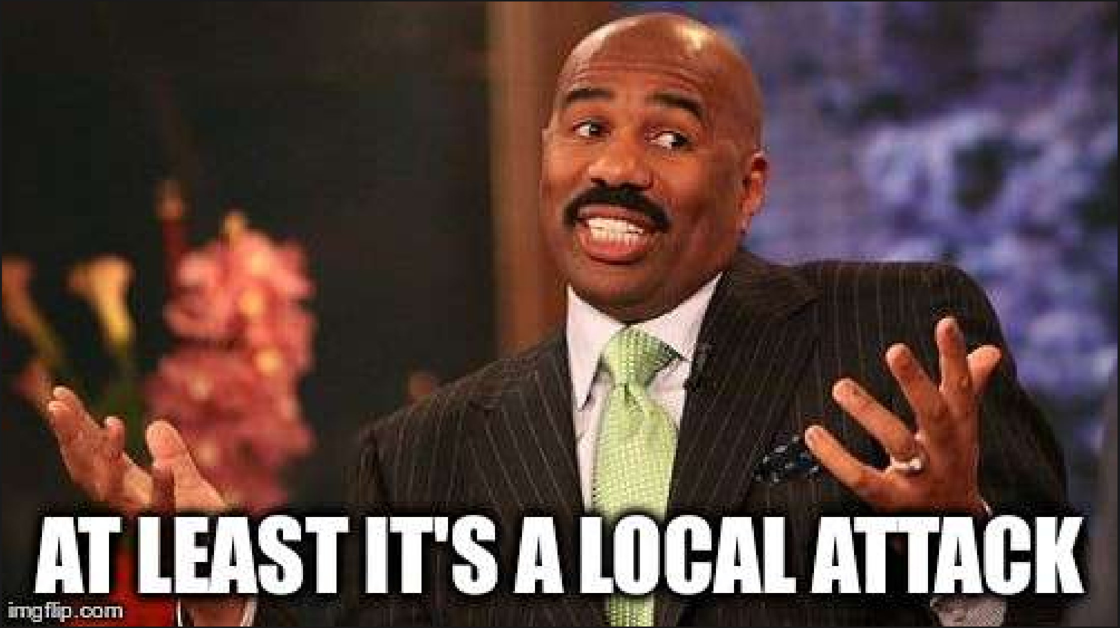

CDB
Reorder buffer
µOP
Scheduler
µOP
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Execution Engine
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache
L3 Cache
DRAM
Memory Subsystem
...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

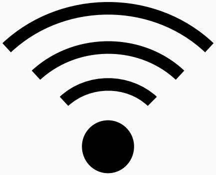
ZombieLoad



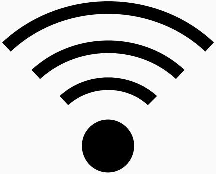
CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
...
mov al, byte [rcx]
...
data can go
to register
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB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


File Edit View Bookmarks Settings Help
michael@hp /tmp/zombieload %
zombieload : zsh


AT LEAST IT'S A LOCAL ATTACK
imgflip.com

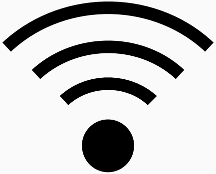


Just a few examples:



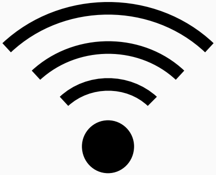
Just a few examples:

- Remote timing attacks on crypto



Just a few examples:

- Remote timing attacks on crypto
- ThrowHammer and NetHammer



Just a few examples:

- Remote timing attacks on crypto
- ThrowHammer and NetHammer
- NetSpectre

We have ignored microarchitectural attacks for many years:

We have ignored microarchitectural attacks for many years:

- attacks on crypto

How did we get here?

We have ignored microarchitectural attacks for many years:

- attacks on crypto → “software should be fixed”

How did we get here?

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

How did we get here?

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"

How did we get here?

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

... and we're still optimizing for performance

- 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

... and we're still optimizing for performance

- 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?

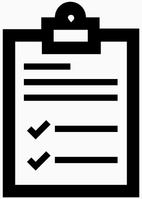
... and we're still optimizing for performance

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

→ it's an optimization problem

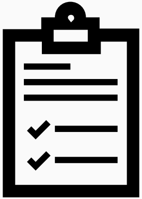
  - what if "too aggressive" changes over time?

  → difficult to optimize with an intelligent adversary

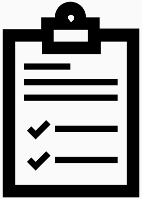


- new class of software-based attacks

Conclusions



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



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

Transient Execution Attacks

Daniel Gruss

July 11, 2019

Graz University of Technology

References

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

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

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

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