

Microarchitectural Attacks

Files for exercise: <https://gruss.cc/fosad.tar.gz>

Daniel Gruss

August 29-30, 2019

Graz University of Technology





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

CHANGE MY MIND



- Profiling cache utilization with performance counters?



- Profiling cache utilization with performance counters? → No





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



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



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



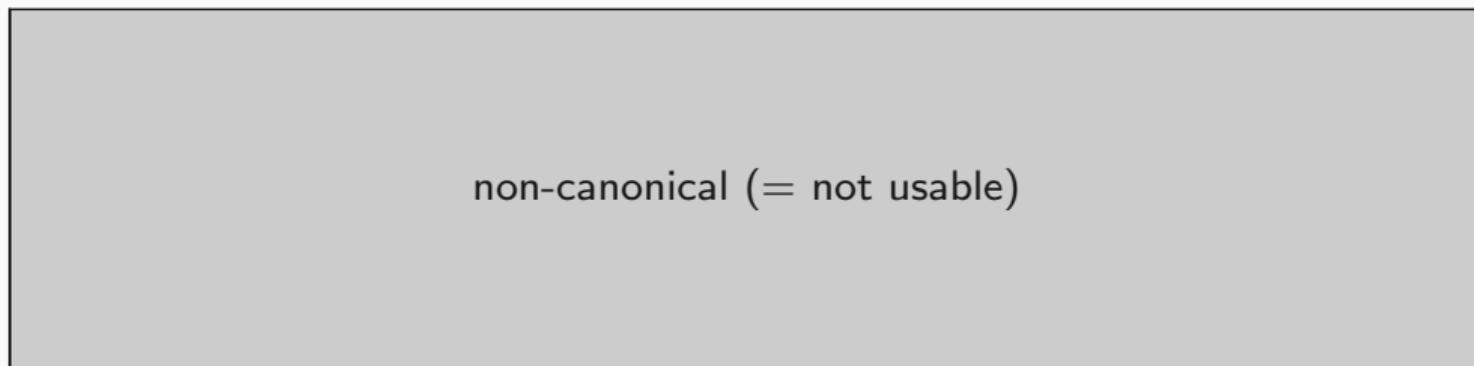
- Profiling cache utilization with performance counters? → No
- Observing cache utilization with performance counters and using it to infer a crypto key? → Yes
- Measuring memory access latency with Flush+Reload? → No



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

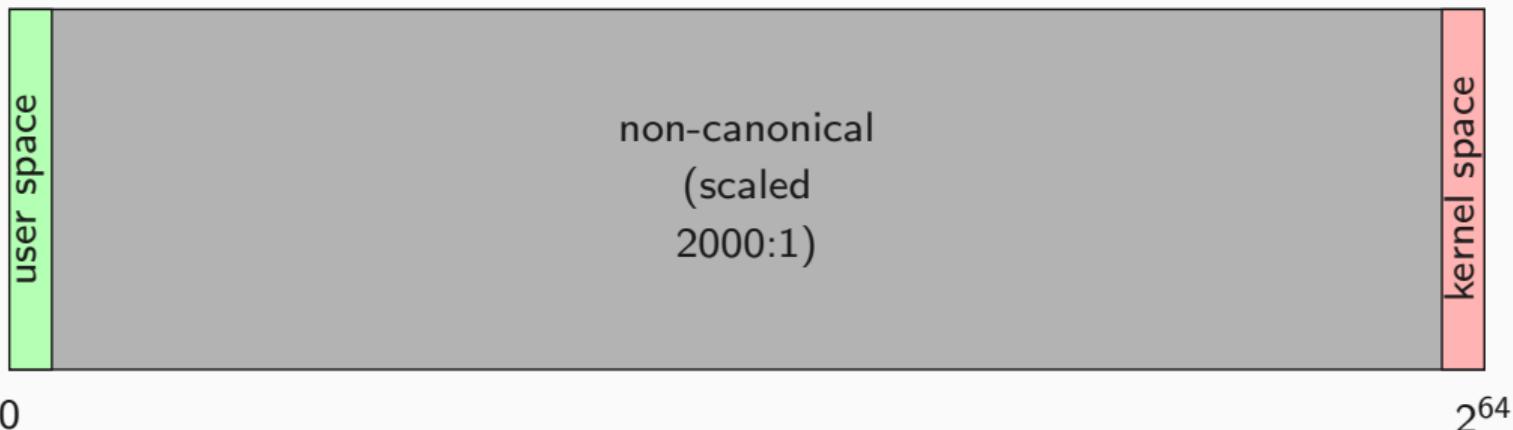


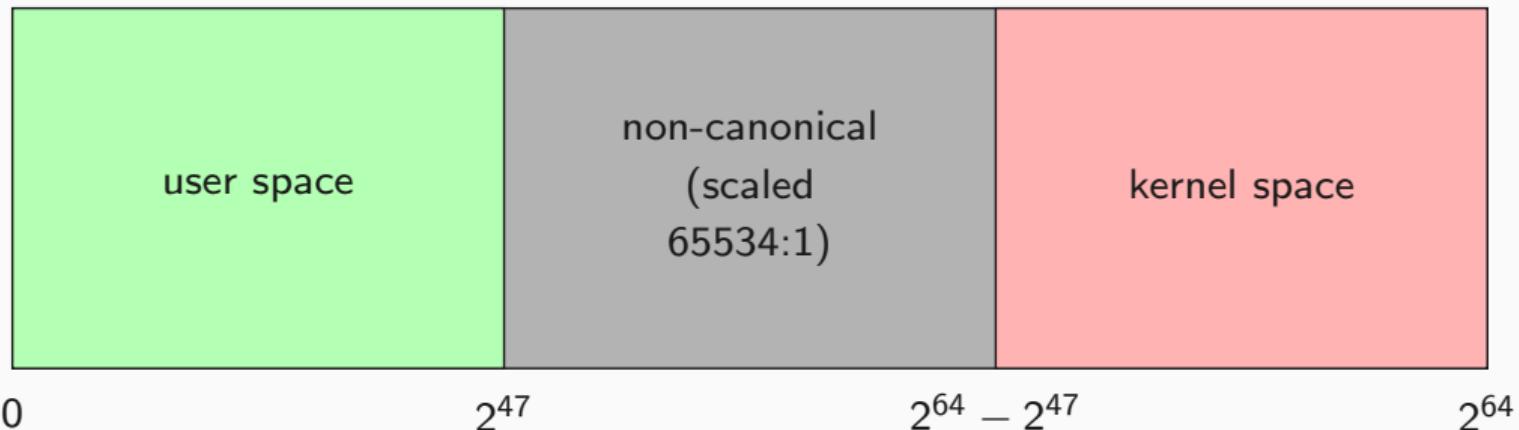
- Profiling cache utilization with performance counters? → No
- Observing cache utilization with performance counters and using it to infer a crypto key? → Yes
- Measuring memory access latency with Flush+Reload? → No
- Measuring memory access latency with Flush+Reload and using it to infer keystroke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



0

2^{64}





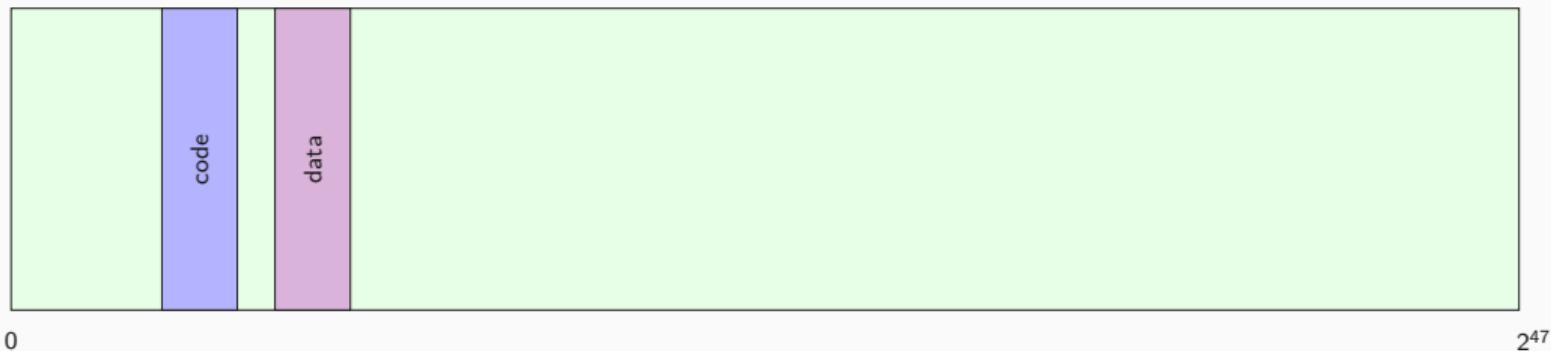


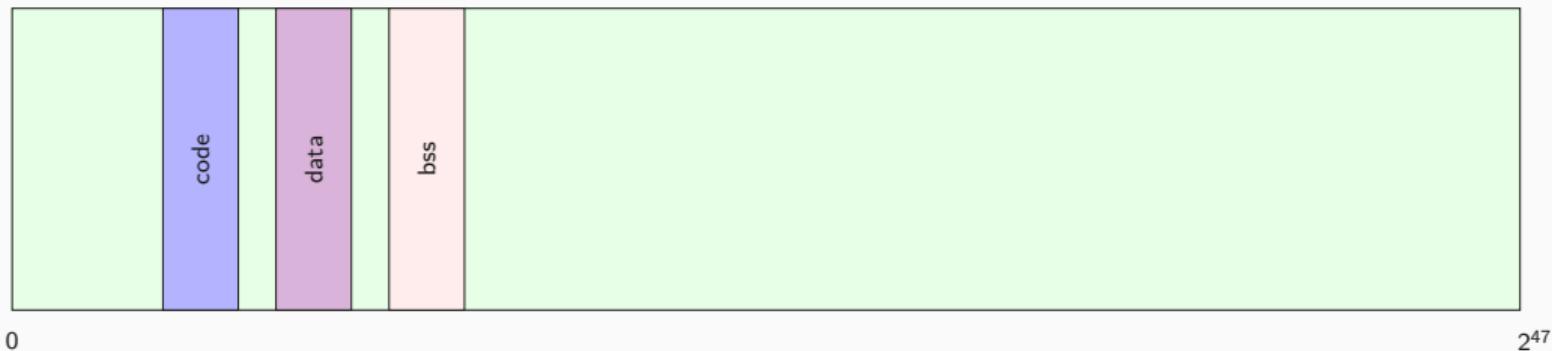


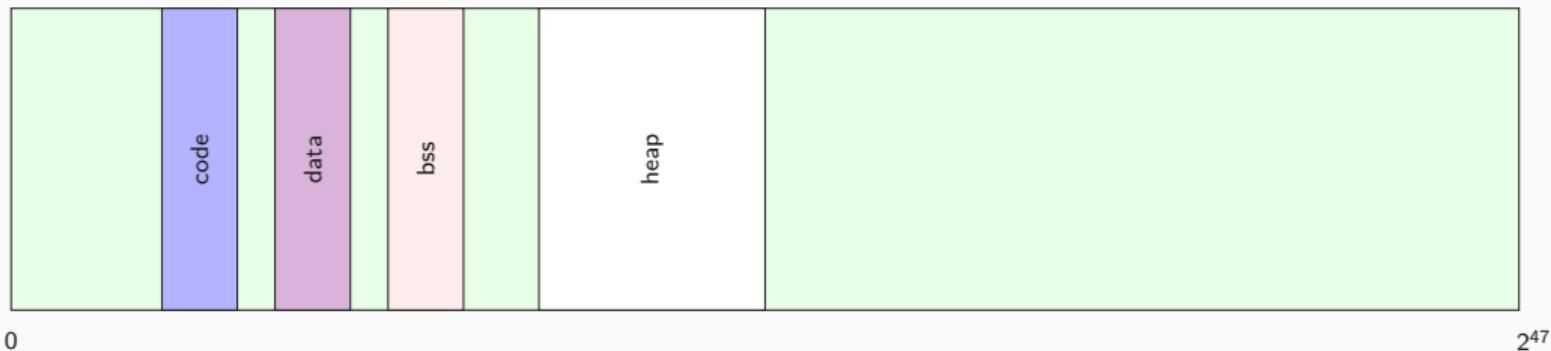
0

2^{47}

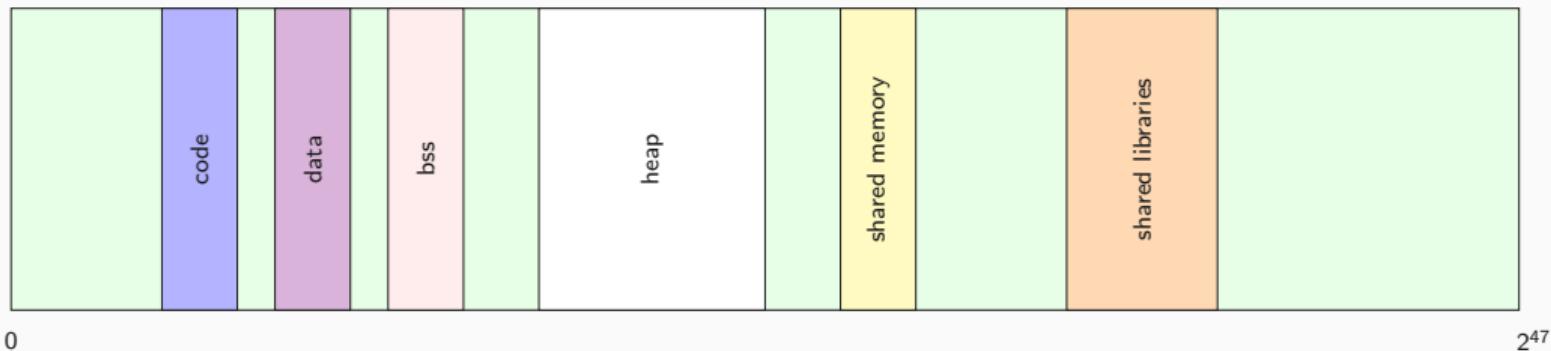


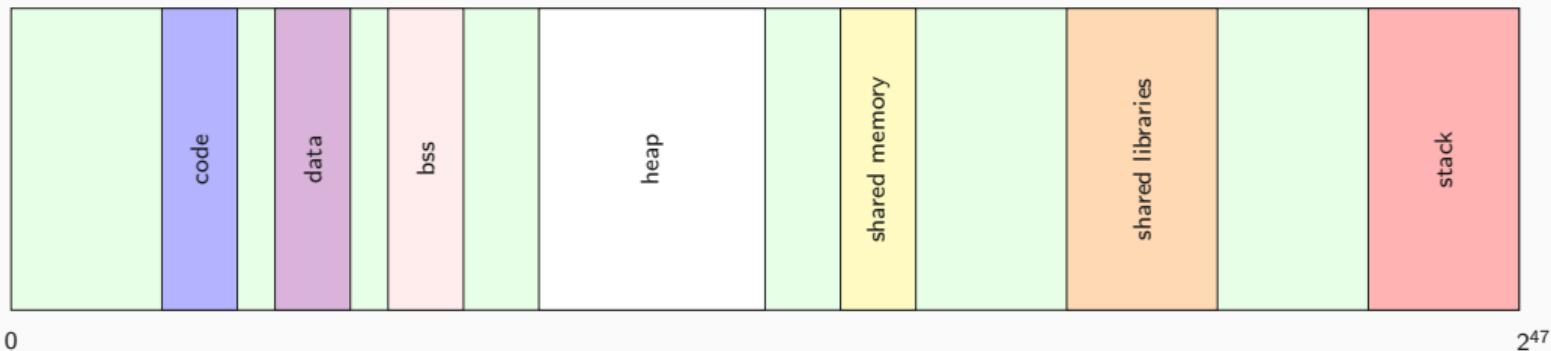


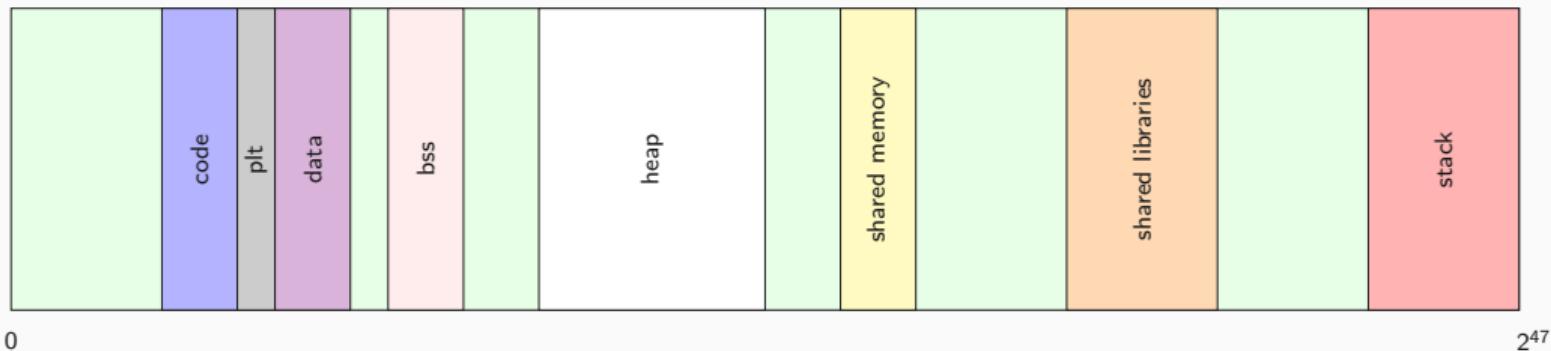


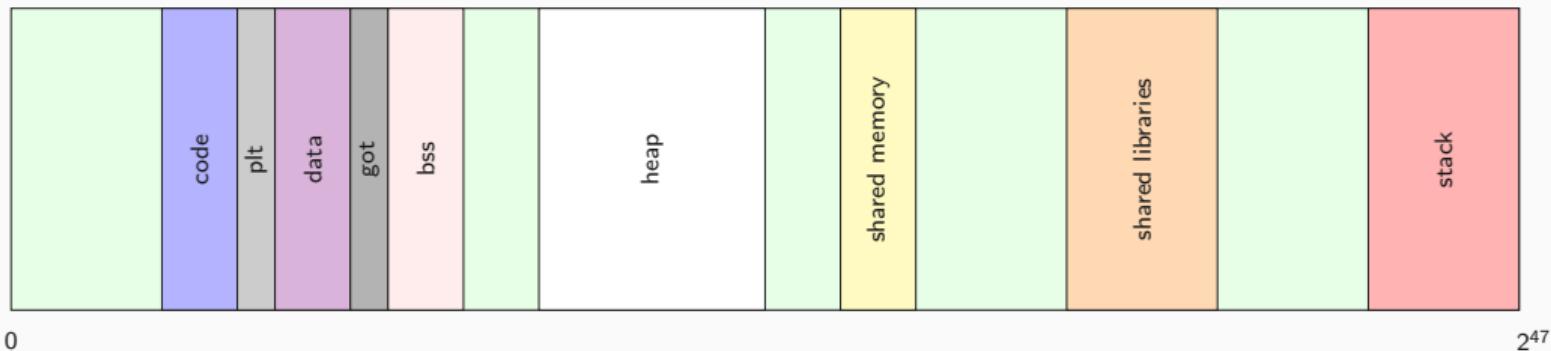












Address Space Layout Randomization (ASLR)...

Address Space Layout Randomization (ASLR)...

- randomizes the memory layout

Address Space Layout Randomization (ASLR)...

- randomizes the memory layout
- moves the base addresses of code/data segment to a random position at startup

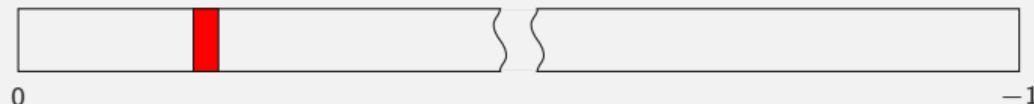
Address Space Layout Randomization (ASLR)...

- randomizes the memory layout
- moves the base addresses of code/data segment to a random position at startup
- requires operating system support

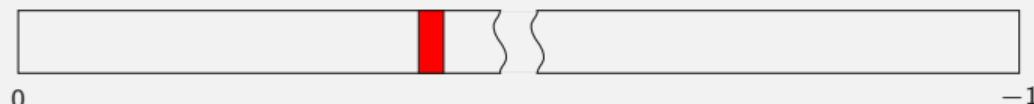
Address Space Layout Randomization (ASLR)...

- randomizes the memory layout
- moves the base addresses of code/data segment to a random position at startup
- requires operating system support
- is quite limited on 32-bit systems

Library A



Library A



Library A



- same library, **different offset** in each process

```
#include <stdio.h>
int main() {
    int x;
    printf("%p\n", &x);
    return 0;
}
```

```
#include <stdio.h>
int main() {
    int x;
    printf("%p\n", &x);
    return 0;
}
```

```
% ./aslr
0x7ffe80fd2da4
% ./aslr
0x7ffd660c6154
% ./aslr
0x7ffeff5545d4
```

- Powerful concept for inter-process communication
- Some are similar to exceptions
- Sometimes used for exploits

```
int main()
{
    signal(SIGSEGV, segfaulthandler)
    ;
    setjmp(buf);
    while (1)
    {
    }
    return 0;
}
```

```
int main()
{
    size_t i = -1ULL;
    signal(SIGSEGV, segfaulthandler)
    ;
    setjmp(buf);
    while (i != -1ULL/4096)
    {
        ++i;
        currentstate = VALID;
        *(volatile size_t*)(i*4096);
    }
    return 0;
}
```

```
// ...
while (i != -1ULL/4096)
{
    printState();
    laststate = currentstate;
    ++i;
    currentstate = VALID;
    *(volatile size_t*)(i*4096);
}
// ...
```

```
static void unblock_signal(int
    signum __attribute__((
    __unused__))) {
    sigset_t sigs;
    sigemptyset(&sigs);
    sigaddset(&sigs, signum);
    sigprocmask(SIG_UNBLOCK, &sigs,
        NULL);
}

void segfaulthandler(int signum) {
    currentstate = INVALID;
    unblock_signal(SIGSEGV);
    longjmp(buf,0);
}

// ...
while (i != -1ULL/4096)
{
    printState();
    laststate = currentstate;
    ++i;
    currentstate = VALID;
    *(volatile size_t*)(i*4096);
}
// ...
```

Demo / Time to code

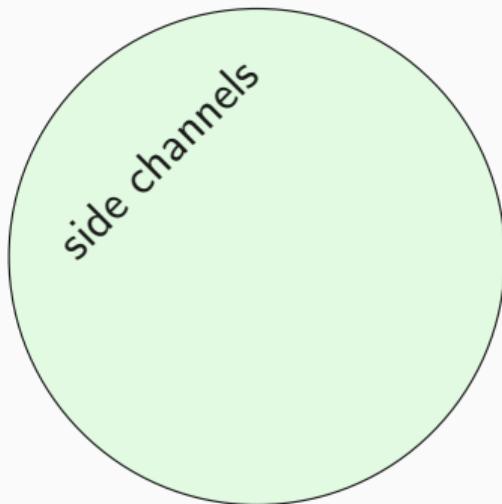
Intel Analysis of Speculative Execution Side Channels

[Download PDF](#)

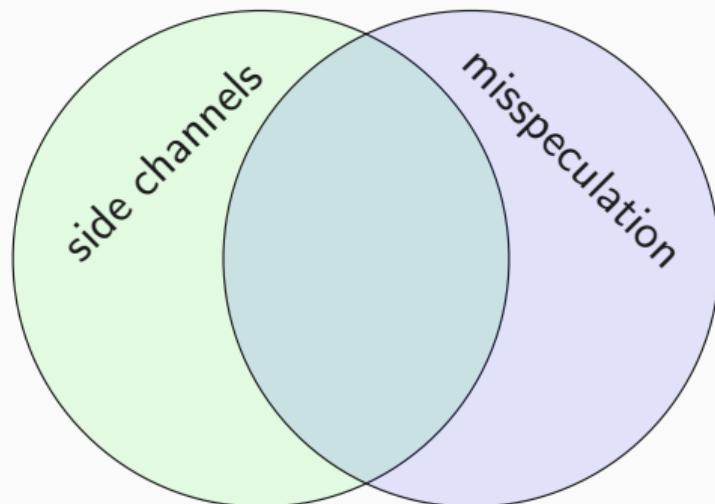


Intel Analysis of Speculative Execution Side Channels

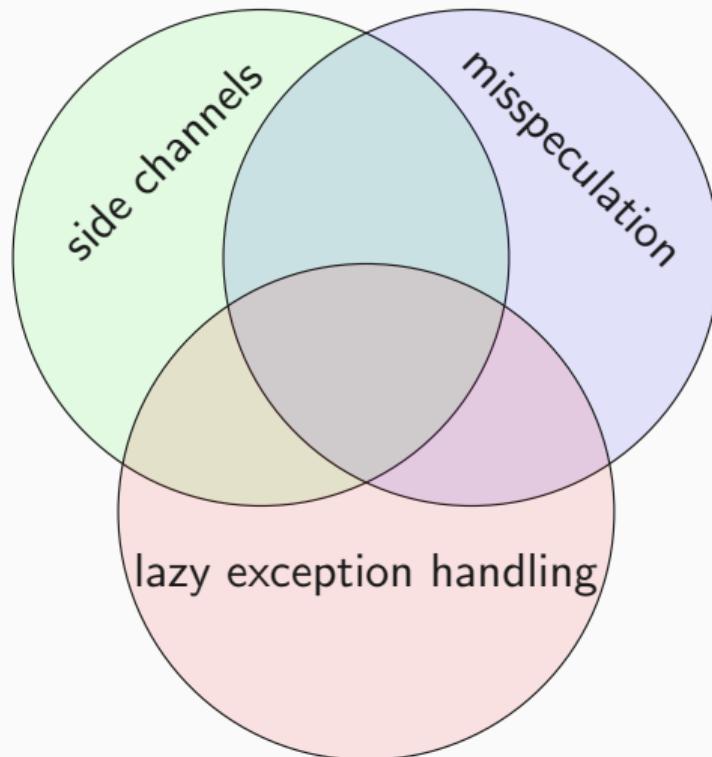
[White Paper](#)



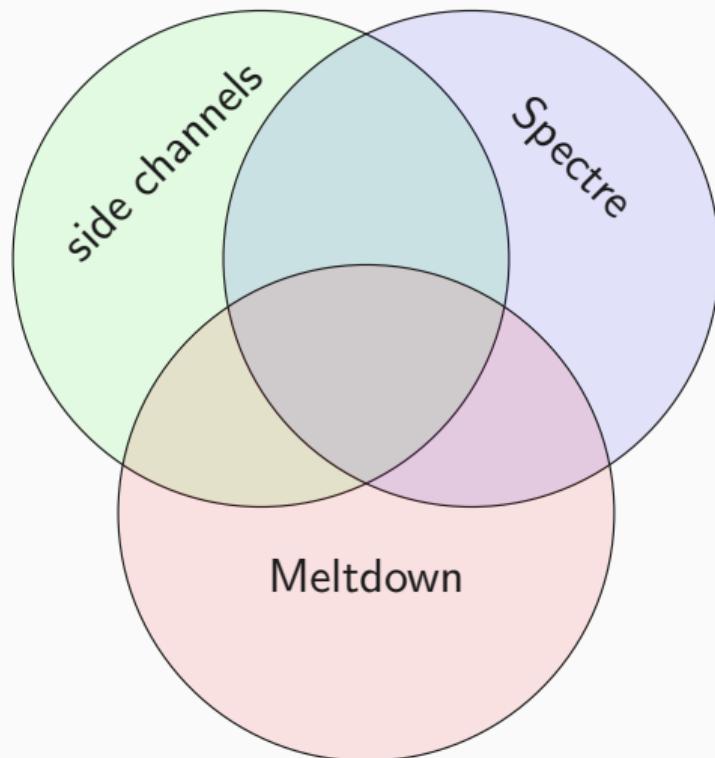
- traditional cache attacks (crypto, keys, etc)



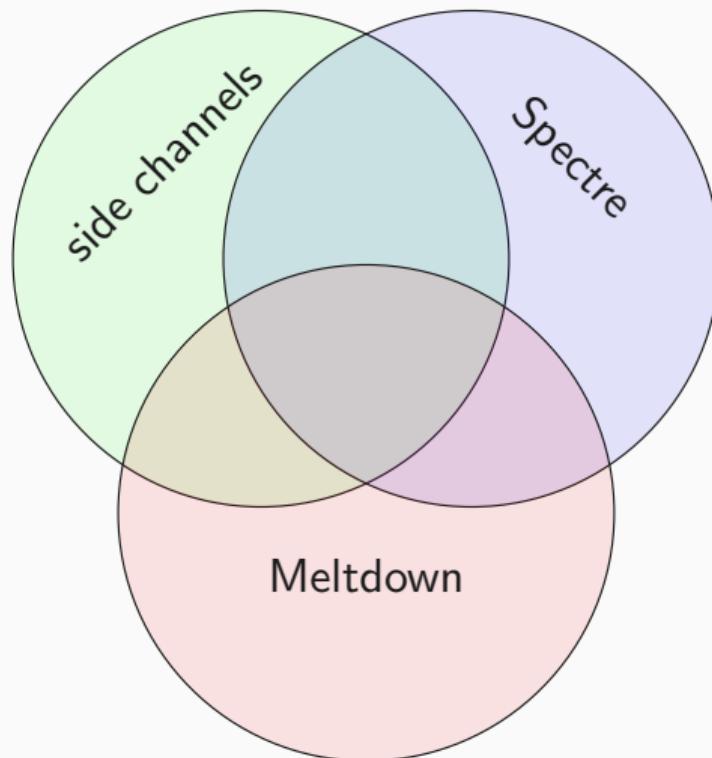
- traditional cache attacks (crypto, keys, etc)
- actual misspeculation (e.g., branch misprediction)



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



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



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









1337 4242

FOOD CACHE

Revolutionary concept!

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

Can store **ALL** kinds of food.

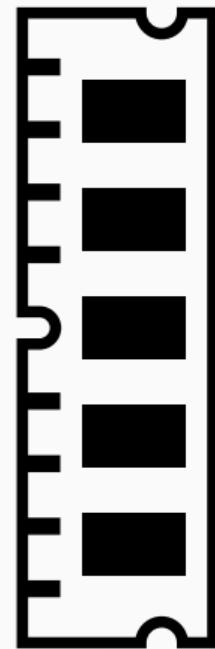
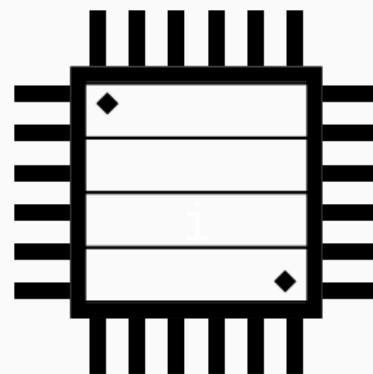
ONLY TODAY INSTEAD OF ~~\$1,300~~

\$1,299

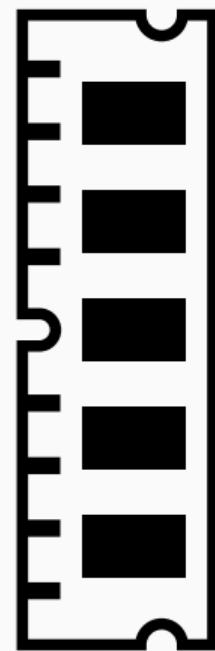
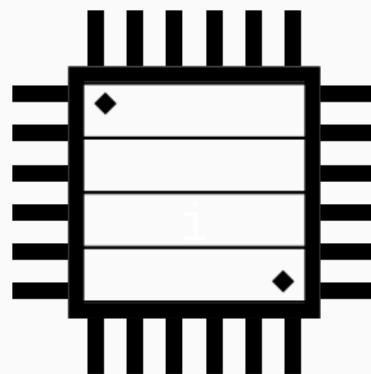
ORDER VIA PHONE: +555 12345



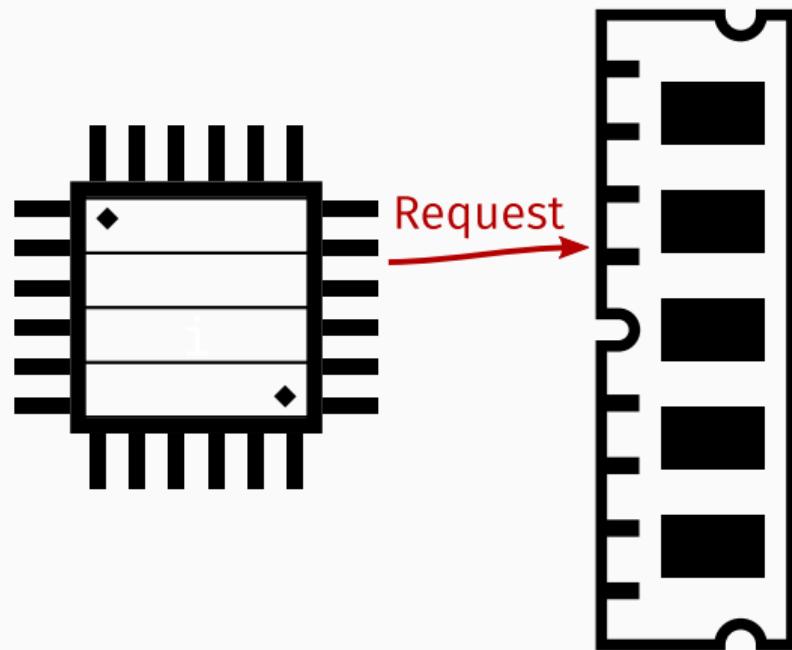
```
printf("%d", i);  
printf("%d", i);
```

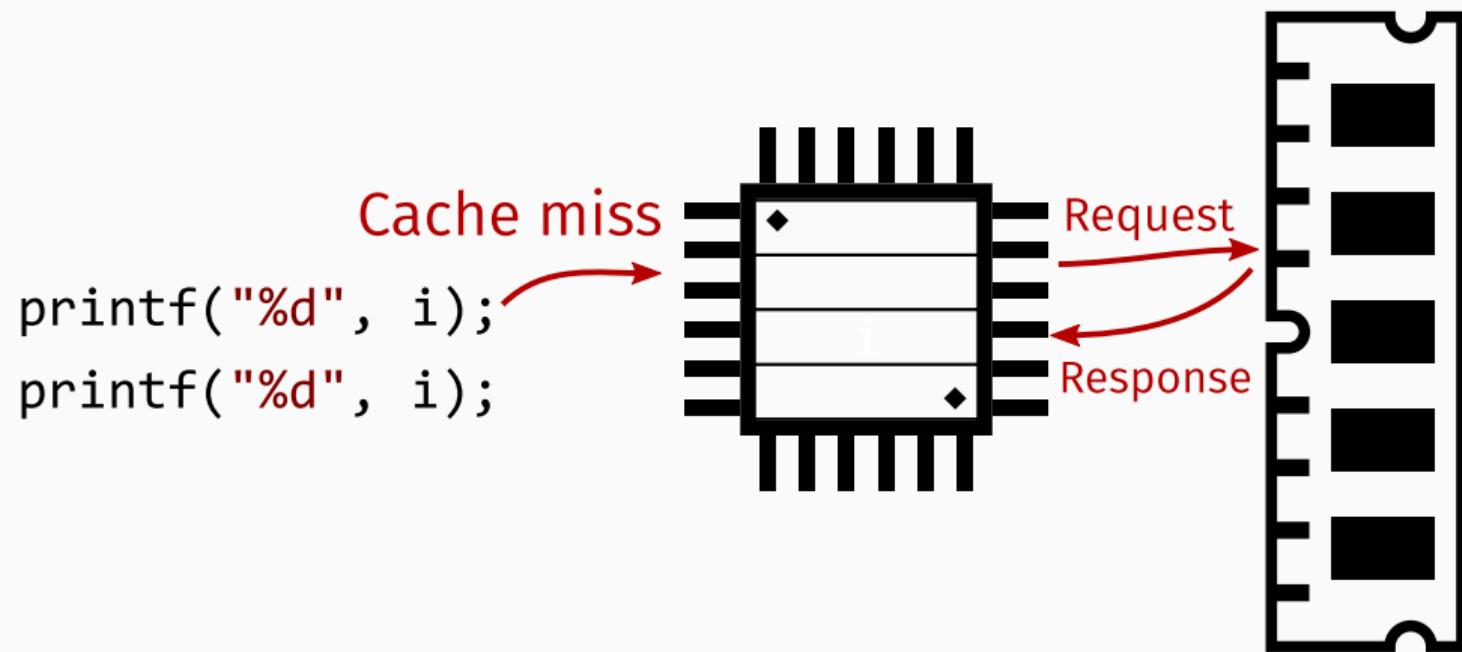


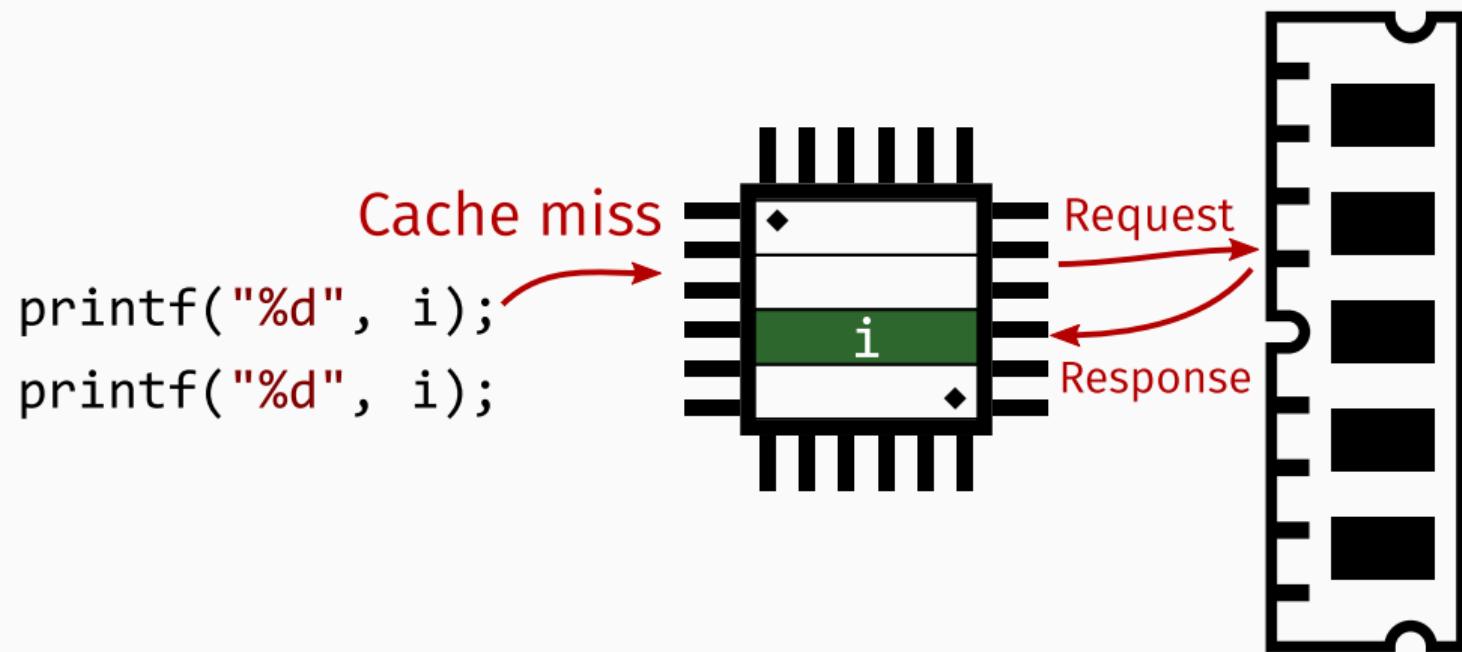
Cache miss
printf("%d", i);
printf("%d", i);

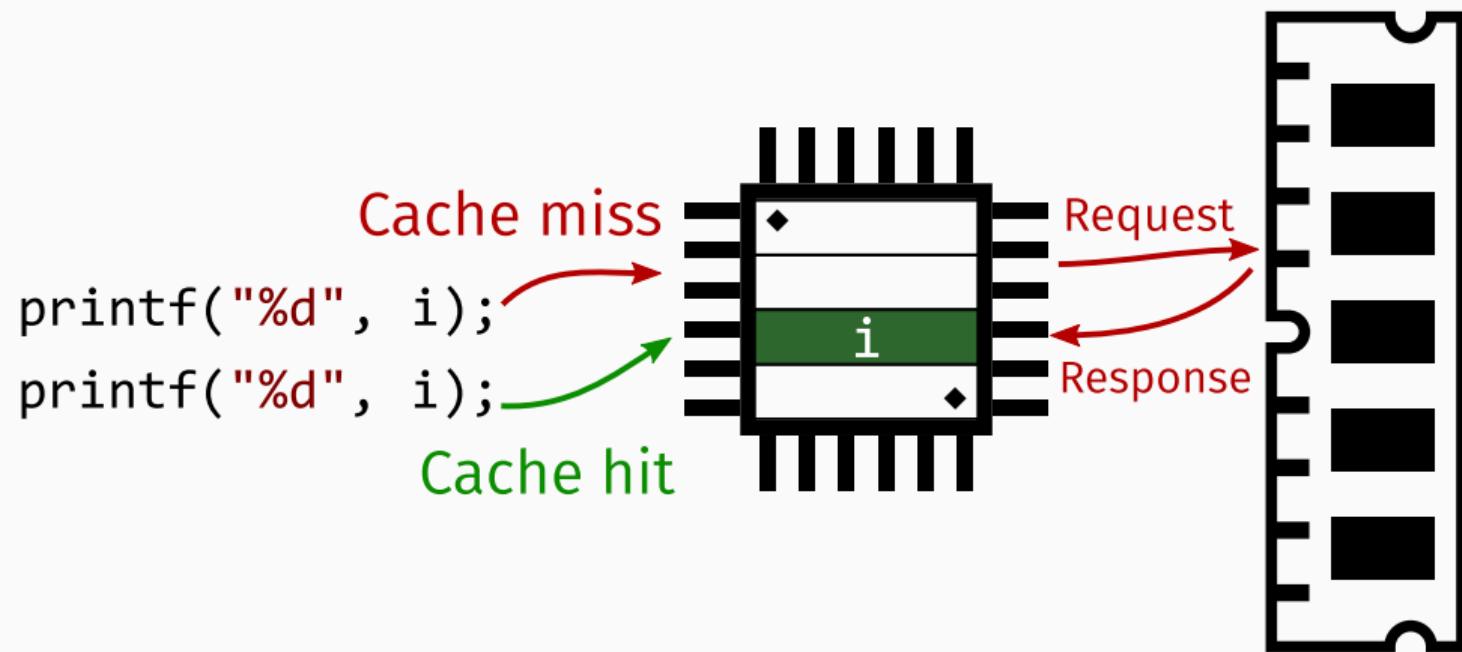


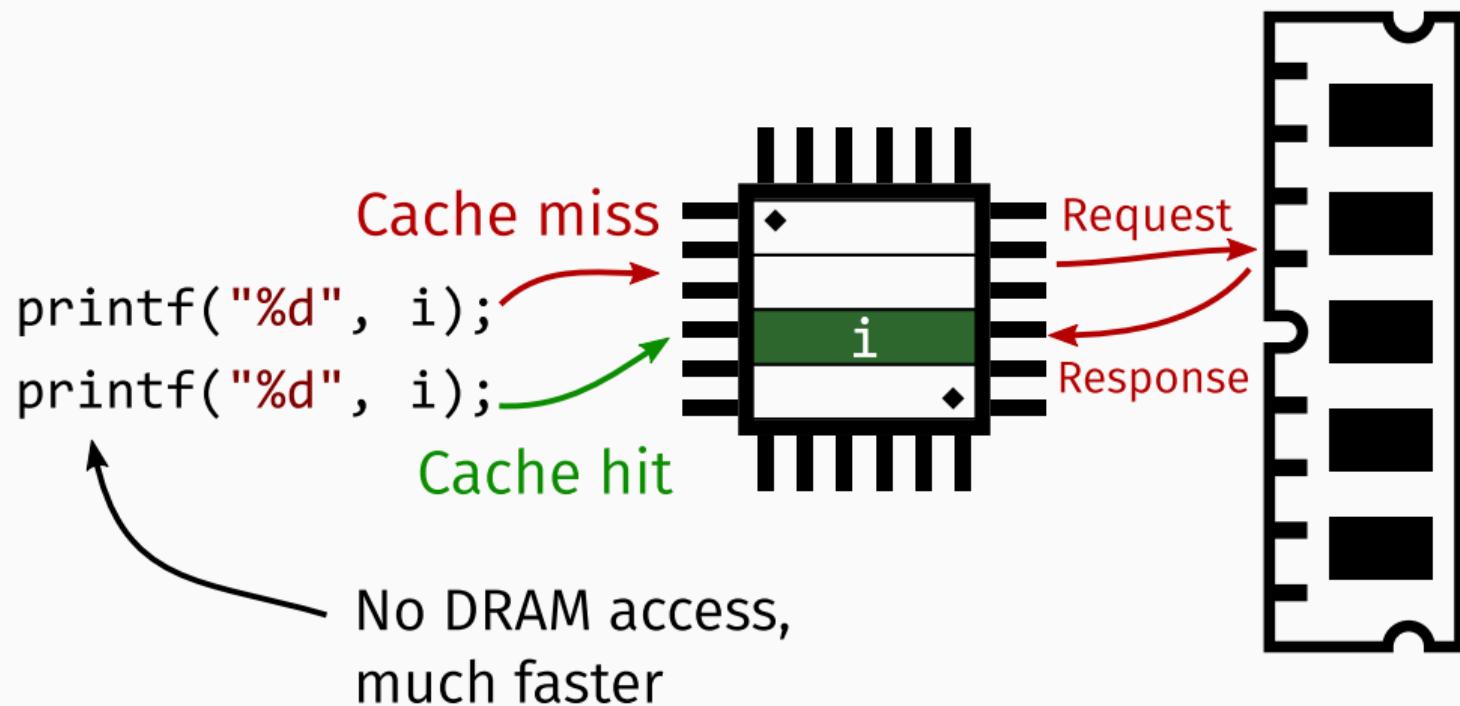
```
Cache miss  
printf("%d", i);  
printf("%d", i);
```

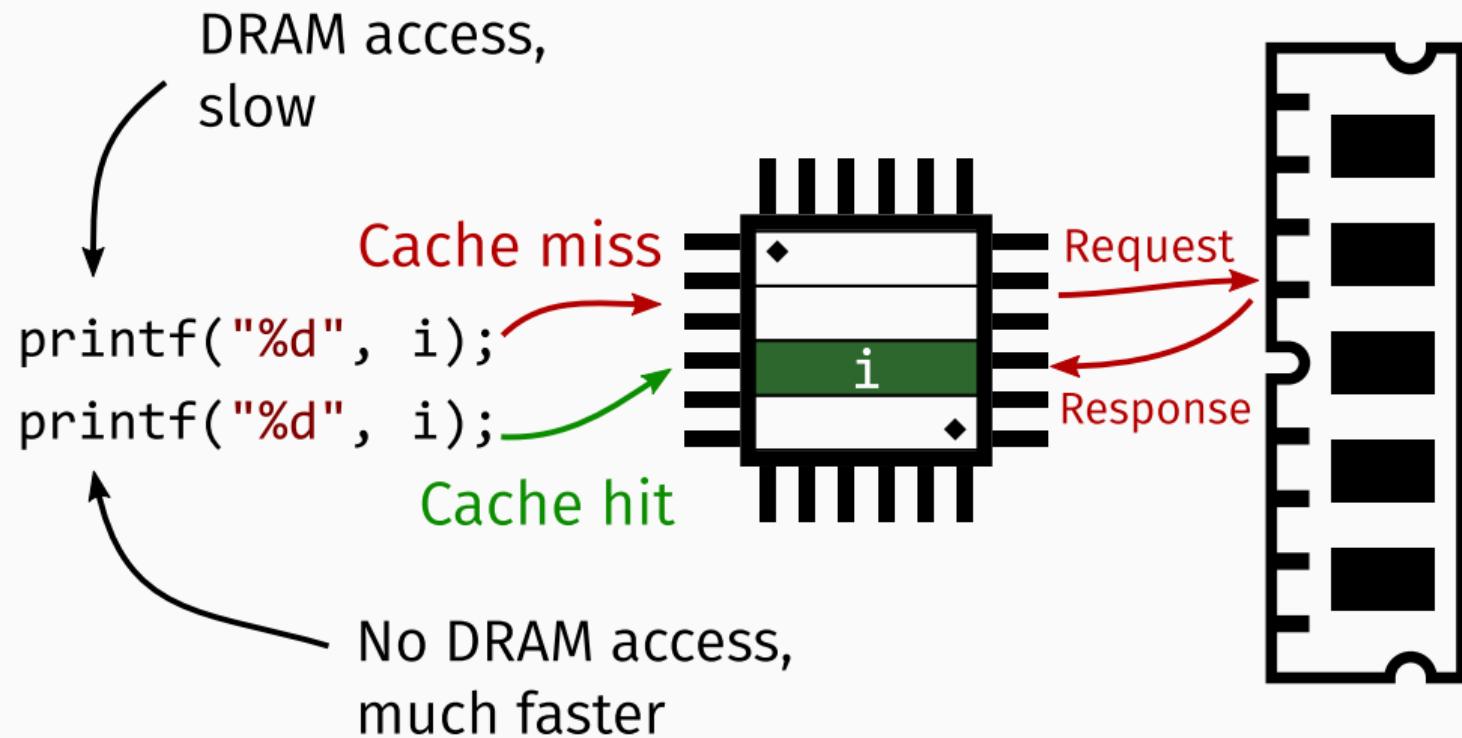


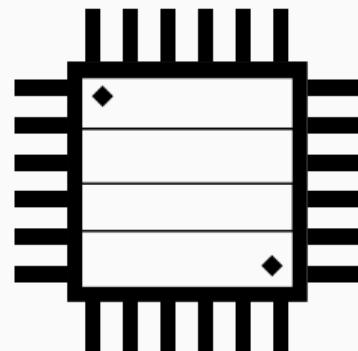


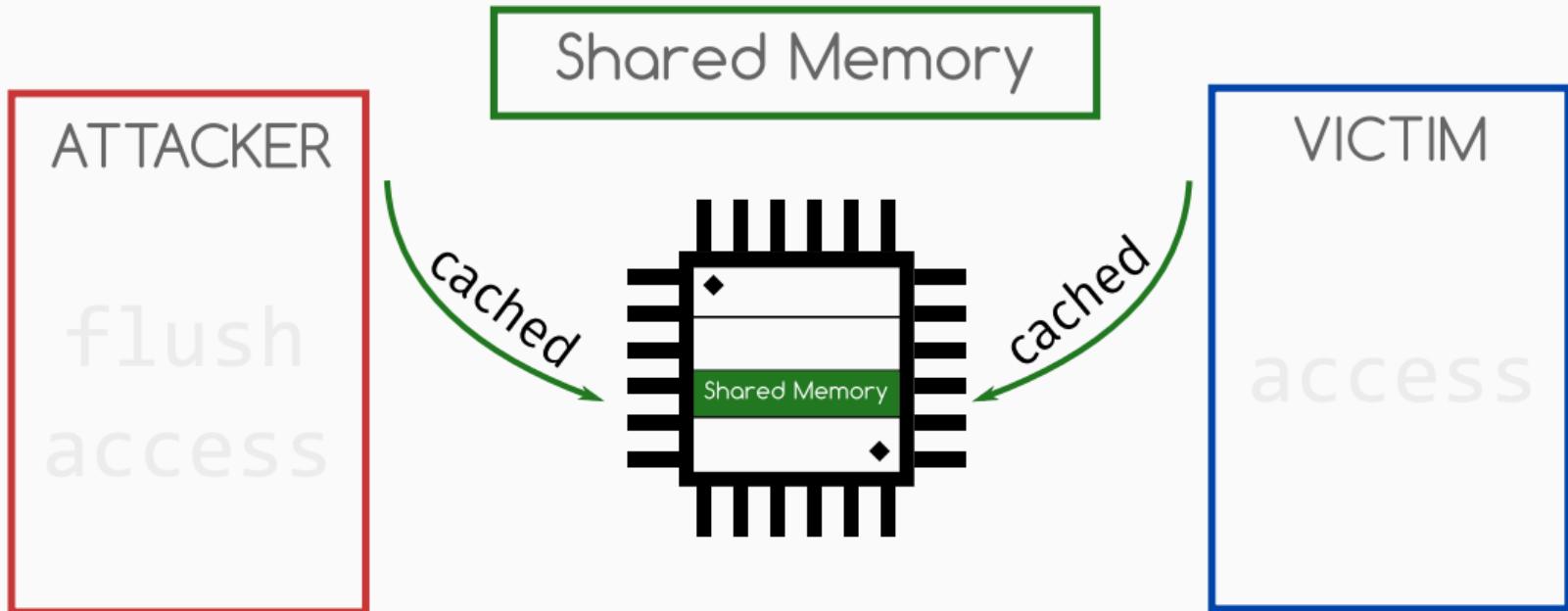


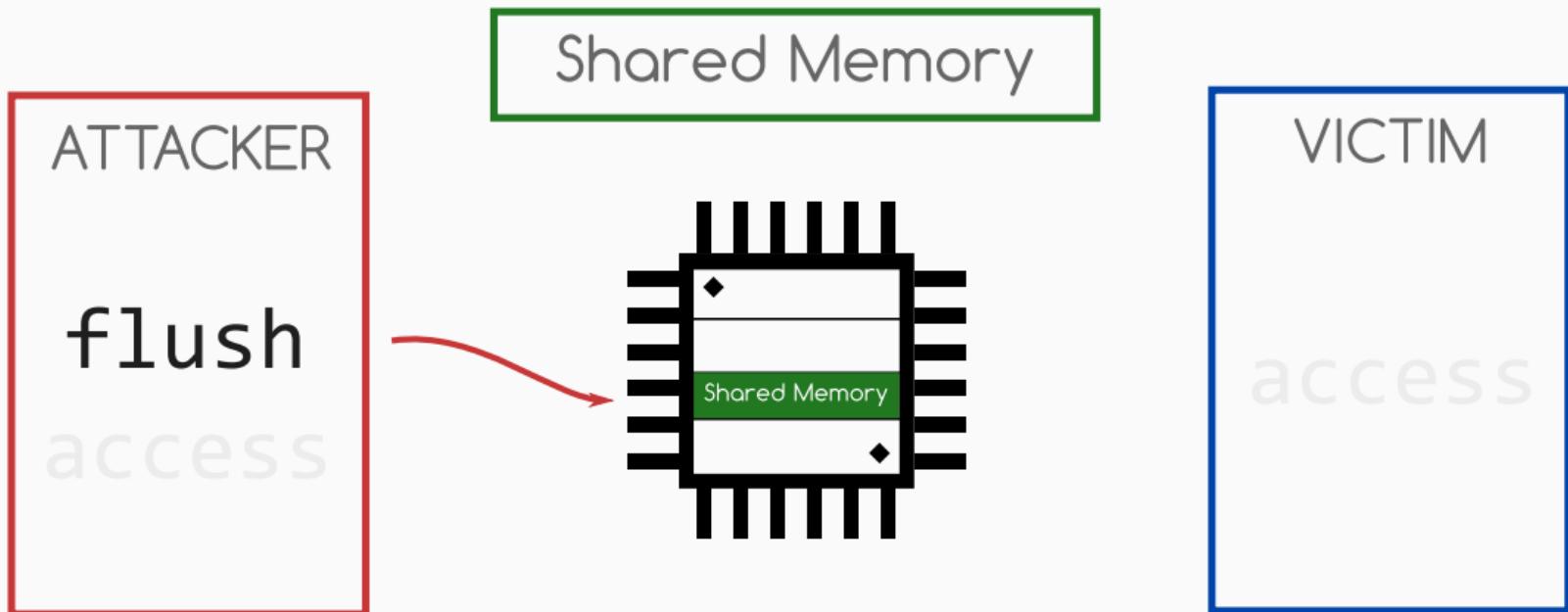


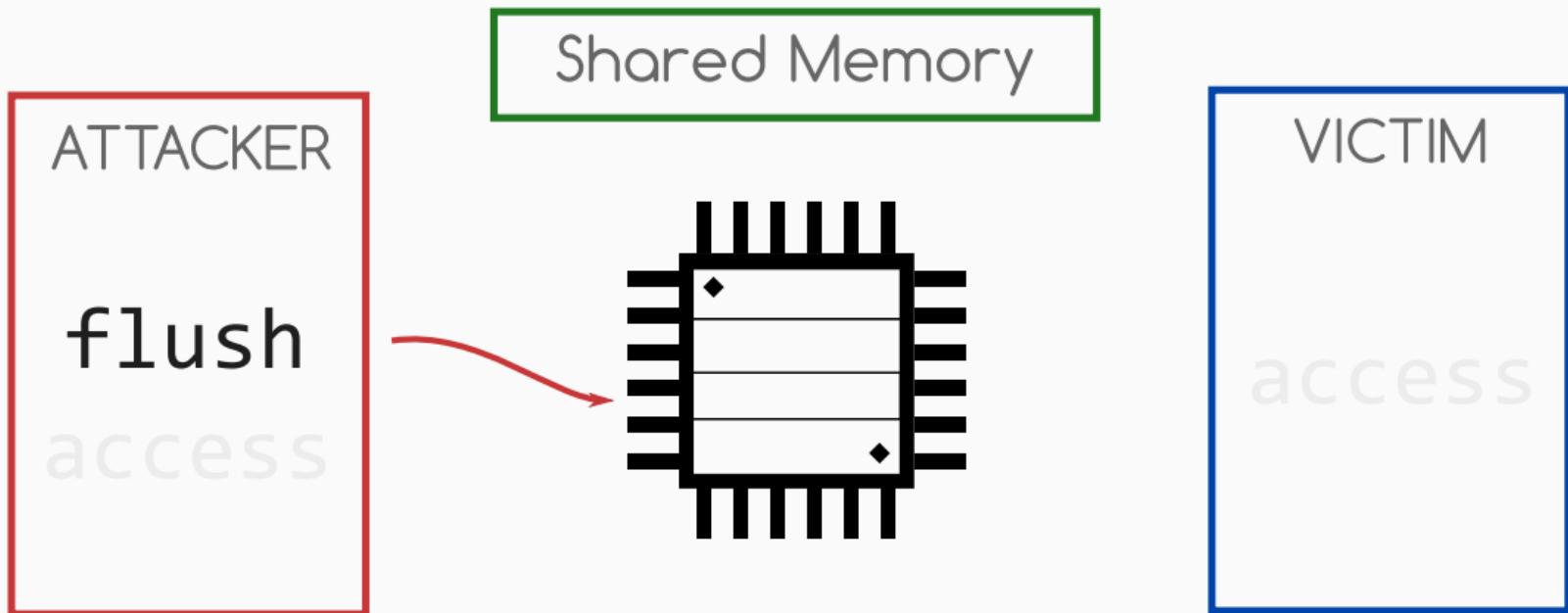












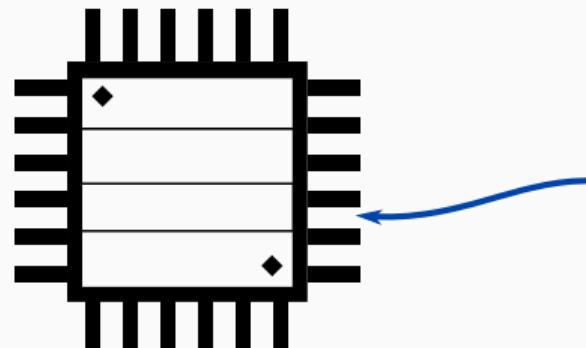
ATTACKER

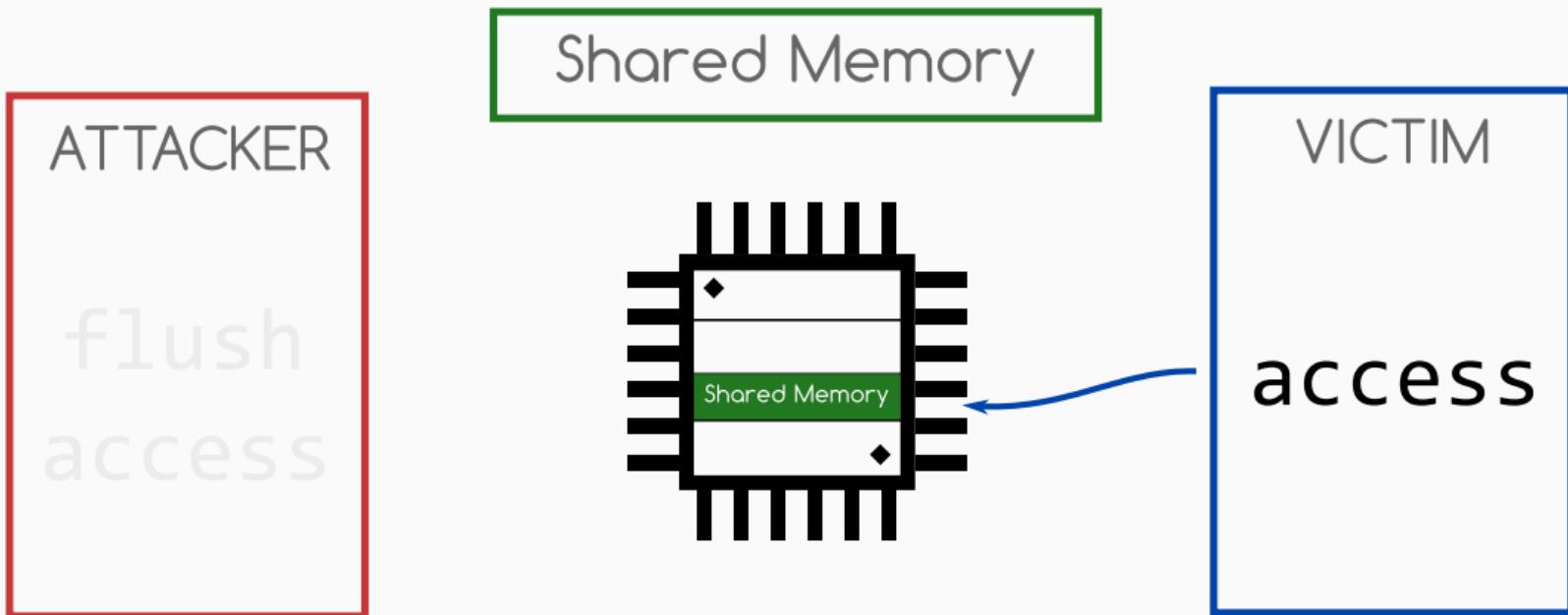
flush
access

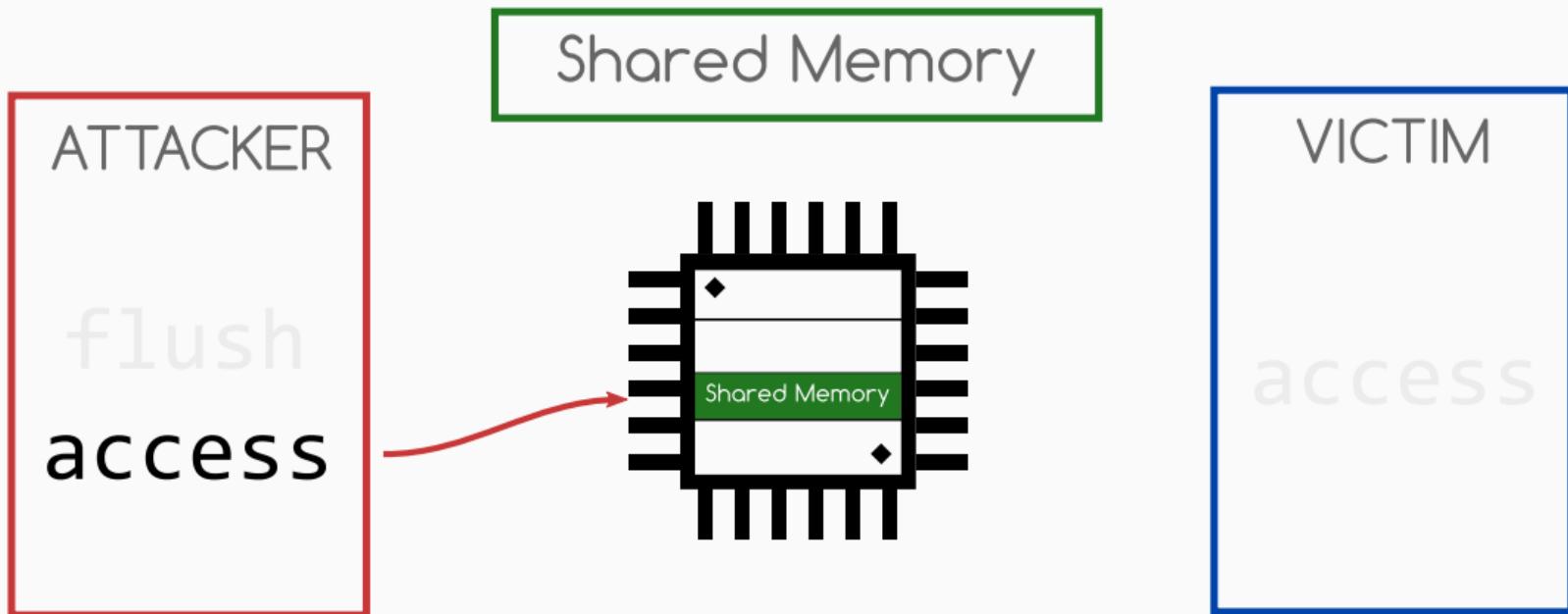
Shared Memory

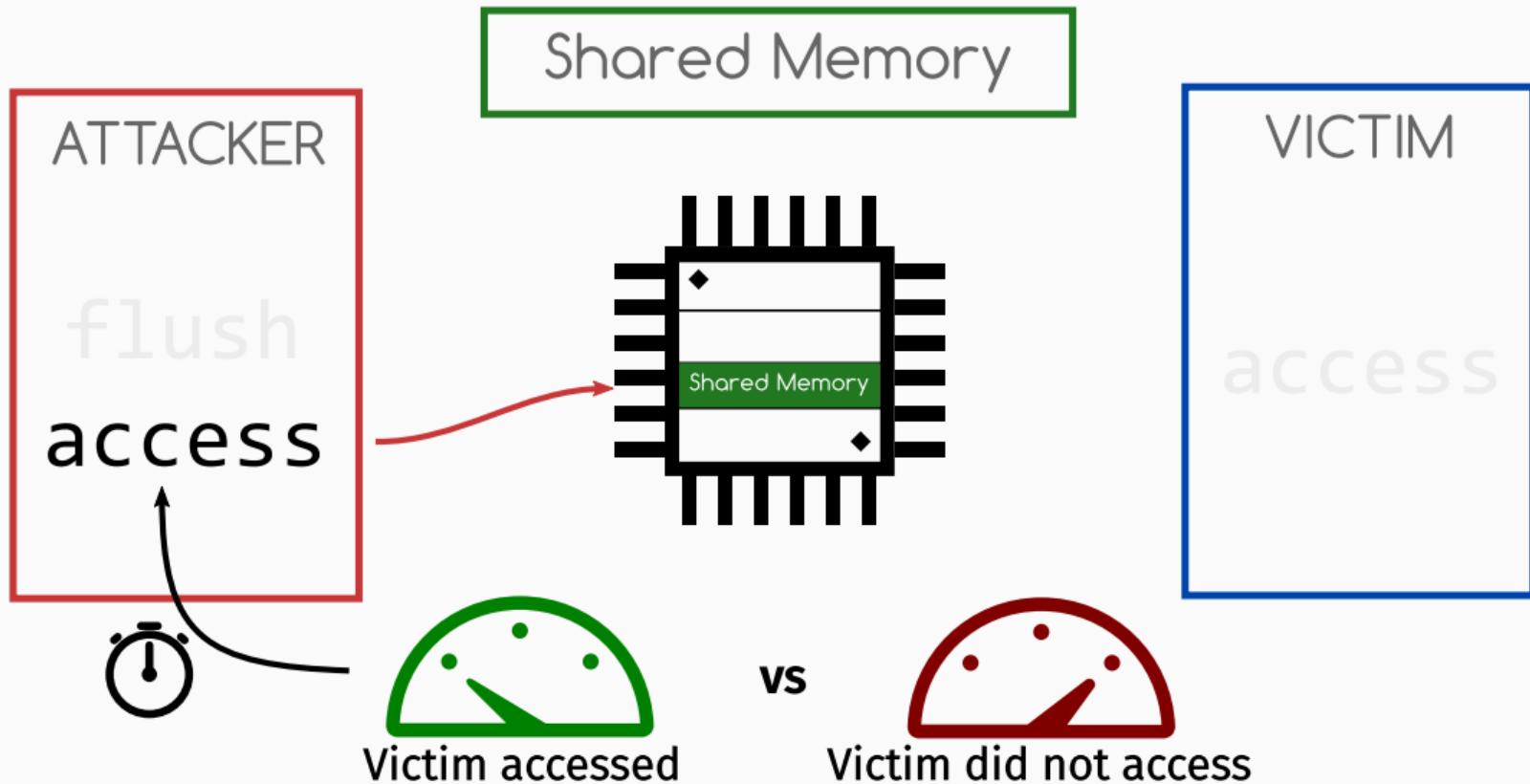
VICTIM

access









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

- 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`

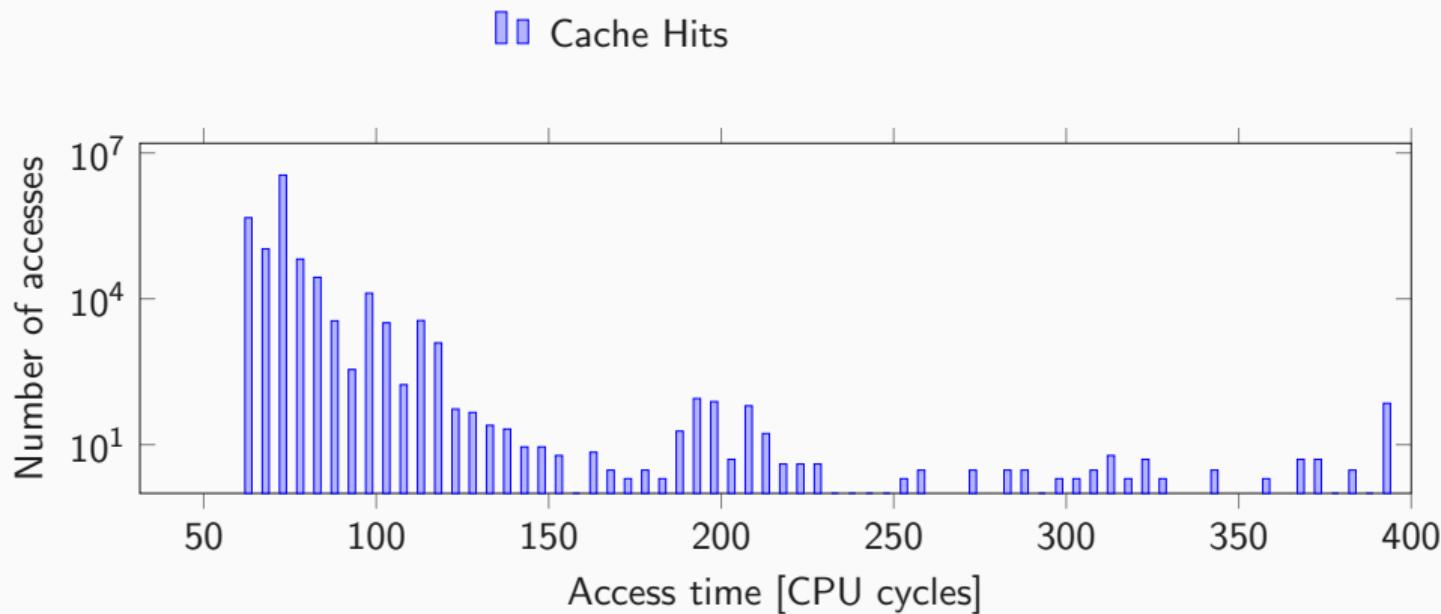
- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cpuid`
- and/or use fences like `mfence`

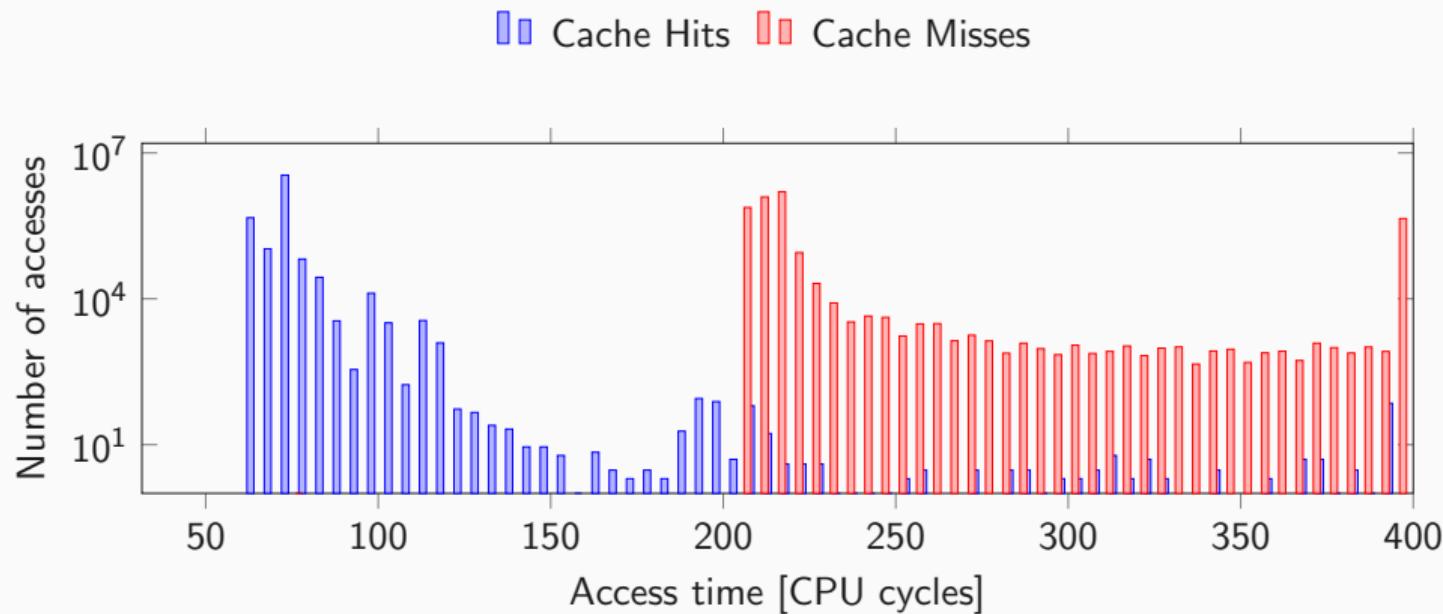
Intel, *How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures White Paper*, December 2010.

AUGUST 22, 2018 BY BRUCE

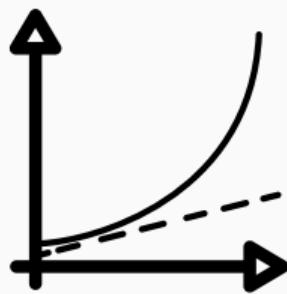
Intel Publishes Microcode Security Patches, No Benchmarking Or Comparison Allowed!

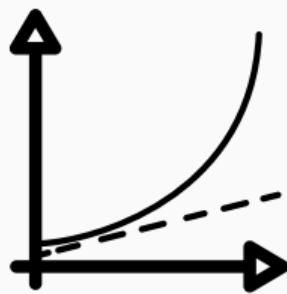
UPDATE: **Intel has resolved their microcode licensing issue which I complained about in this blog post.** The new license text is [here](#).

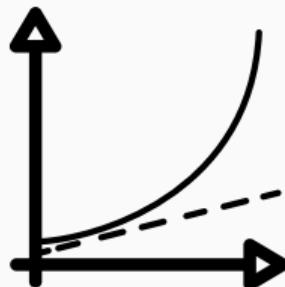




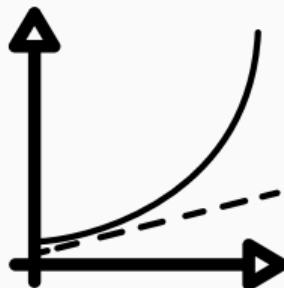
Demo / Time to code



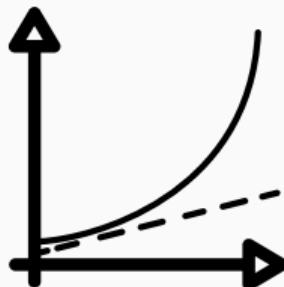




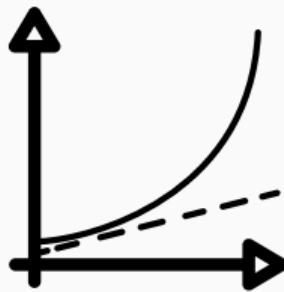
- Flush+Reload had beautifully nice timings, right?



- 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



- We can build our own timer
- Start a thread that continuously increments a global variable



- We can build our own timer
- Start a thread that continuously increments a global variable
- The global variable is our timestamp





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

CPU cycles one increment takes

```
rdtsc [REDACTED] 3           1 timestamp = rdtsc();
```

CPU cycles one increment takes

rdtsc [REDACTED] 3

C

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

CPU cycles one increment takes

rdtsc  3

C  4.7

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

CPU cycles one increment takes

rdtsc  3

C  4.7

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

CPU cycles one increment takes

rdtsc  3

C  4.7

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

Assembly

CPU cycles one increment takes

rdtsc  3

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

C  4.7

Assembly  4.67

CPU cycles one increment takes

rdtsc  3

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

C  4.7

Assembly  4.67

CPU cycles one increment takes

rdtsc  3

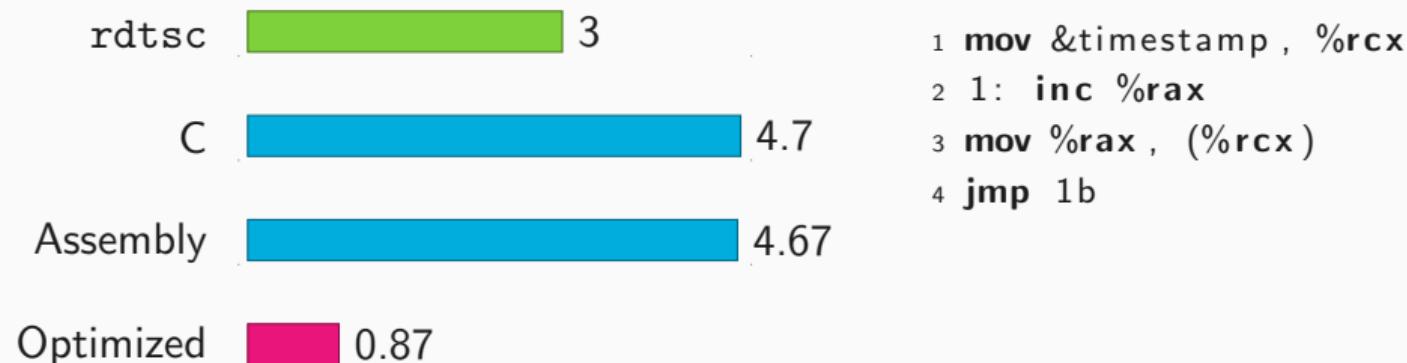
C  4.7

Assembly  4.67

Optimized

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

CPU cycles one increment takes



- locate **key-dependent** memory accesses

- locate **key-dependent** memory accesses
- How to locate key-dependent memory accesses?

- It's complicated:

- It's complicated:
 - Large binaries and libraries (third-party code)

- It's complicated:
 - Large binaries and libraries (third-party code)
 - Many libraries (gedit: 60MB)

- It's complicated:
 - Large binaries and libraries (third-party code)
 - Many libraries (gedit: 60MB)
 - Closed-source / unknown binaries

- It's complicated:
 - Large binaries and libraries (third-party code)
 - Many libraries (gedit: 60MB)
 - Closed-source / unknown binaries
 - Self-compiled binaries

- It's complicated:
 - Large binaries and libraries (third-party code)
 - Many libraries (gedit: 60MB)
 - Closed-source / unknown binaries
 - Self-compiled binaries
- Difficult to find **all** exploitable addresses

Profiling Phase

- Preprocessing step to find exploitable addresses automatically

Exploitation Phase

Profiling Phase

- Preprocessing step to find exploitable addresses automatically
 - w.r.t. “events” (keystrokes, encryptions, ...)

Exploitation Phase

Profiling Phase

- Preprocessing step to find exploitable addresses automatically
 - w.r.t. “events” (keystrokes, encryptions, ...)
 - called “Cache Template”

Exploitation Phase

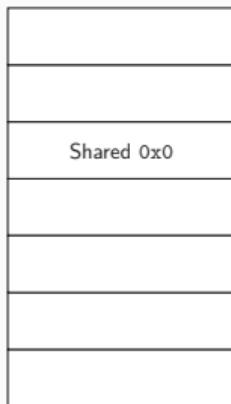
Profiling Phase

- Preprocessing step to find exploitable addresses automatically
 - w.r.t. “events” (keystrokes, encryptions, ...)
 - called “Cache Template”

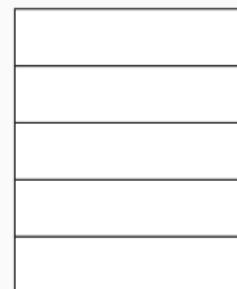
Exploitation Phase

- Monitor exploitable addresses

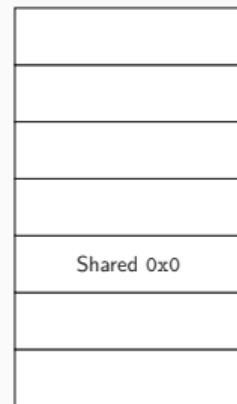
Attacker address space



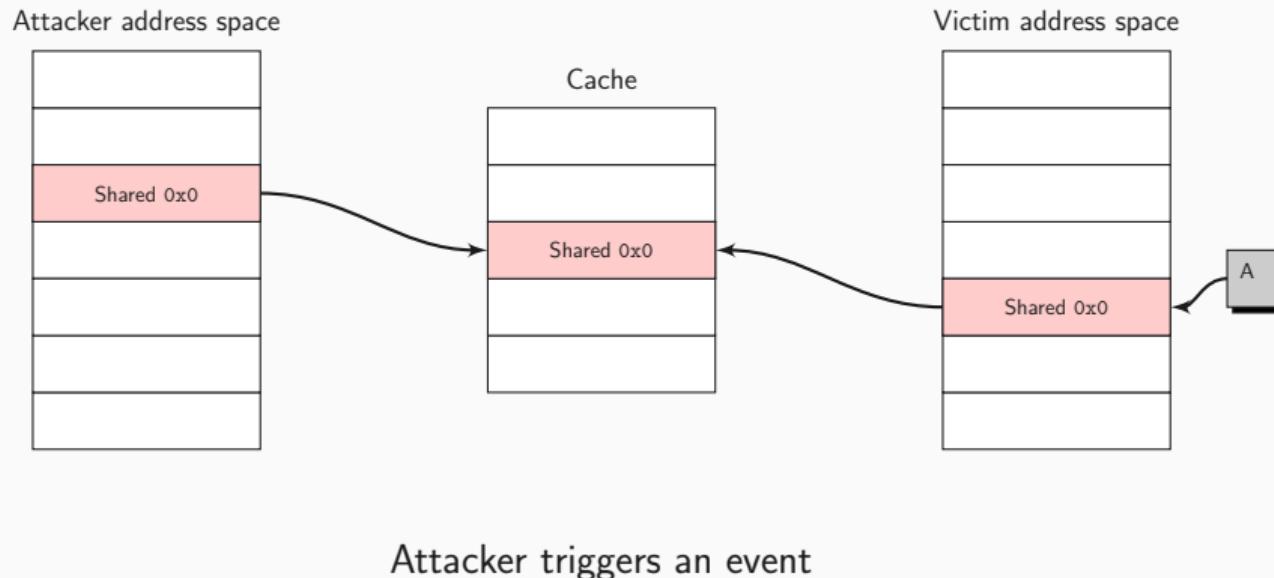
Cache

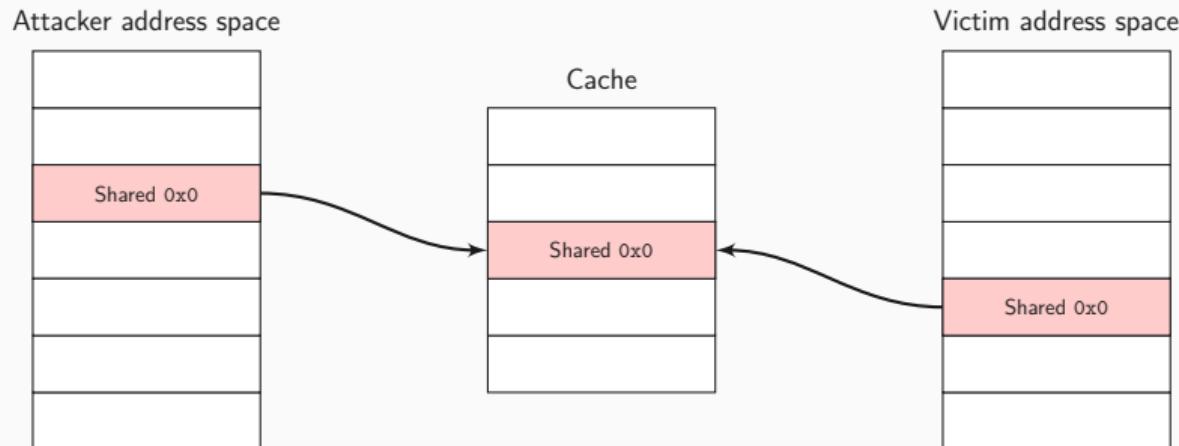


Victim address space

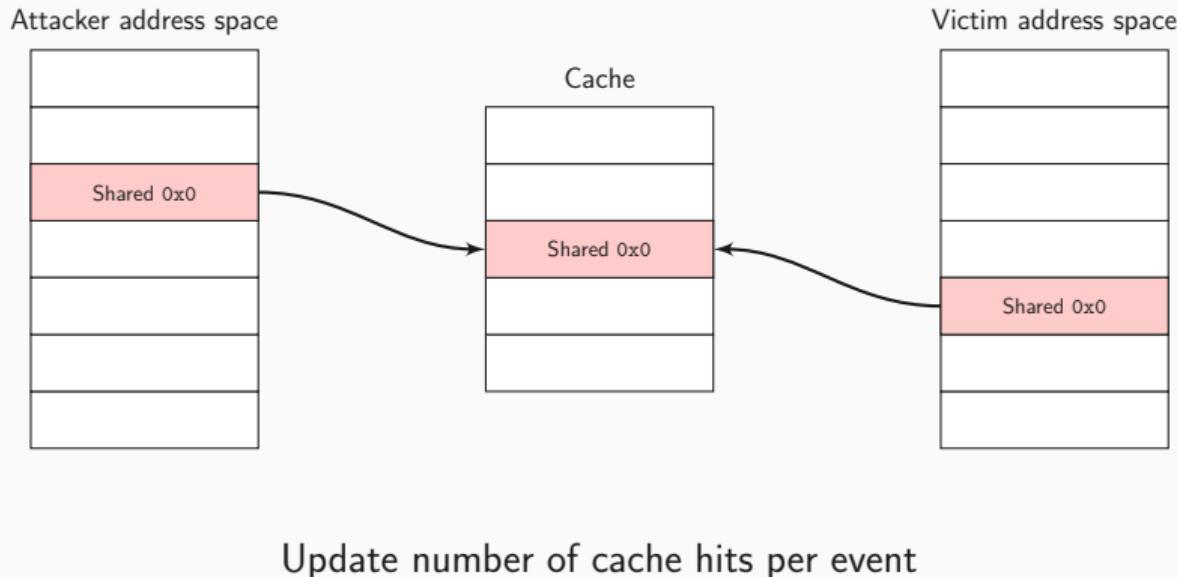


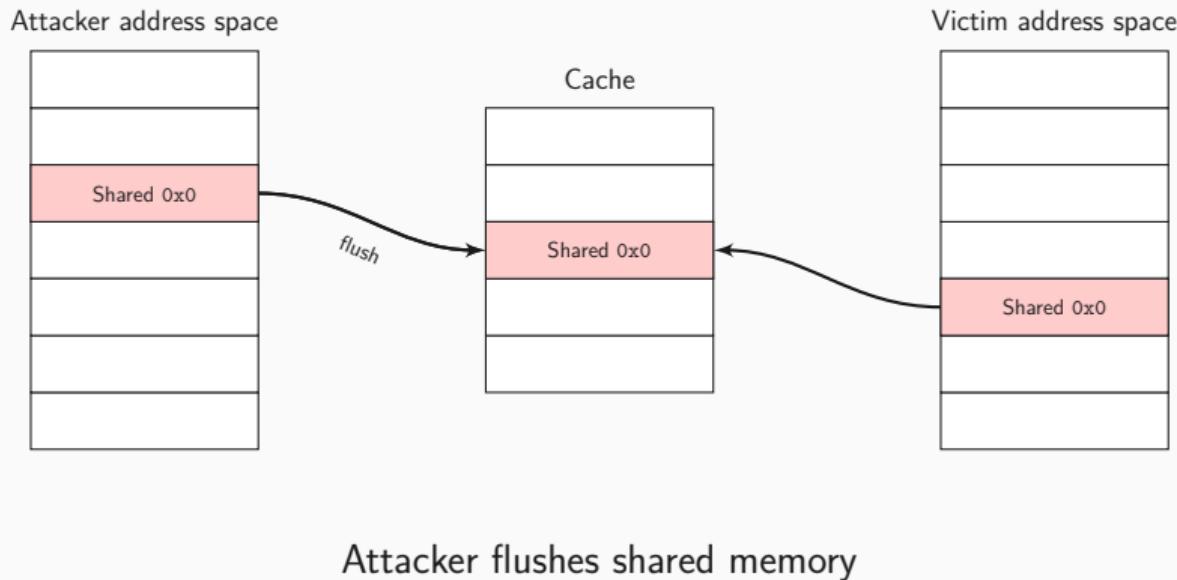
Cache is empty



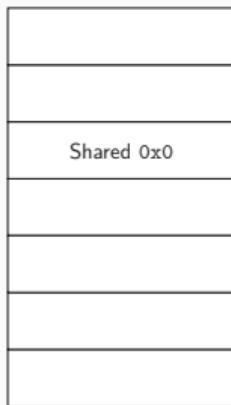


Attacker checks one address for cache hits ("Reload")

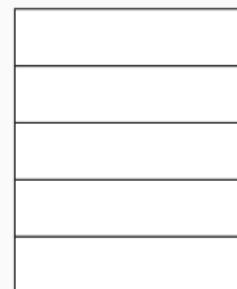




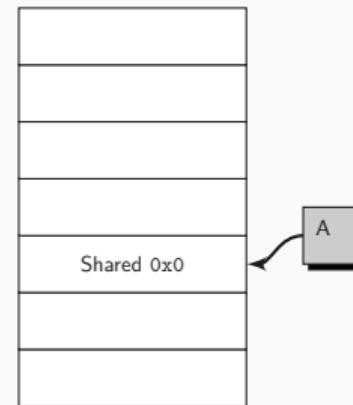
Attacker address space



Cache



Victim address space



Repeat for higher accuracy

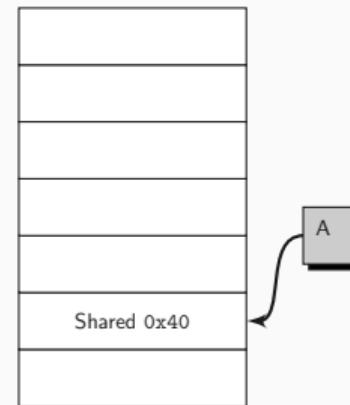
Attacker address space



Cache

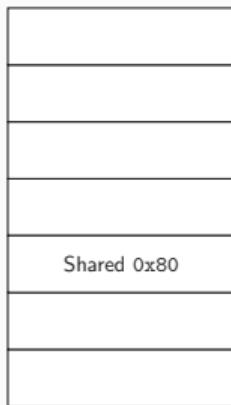


Victim address space

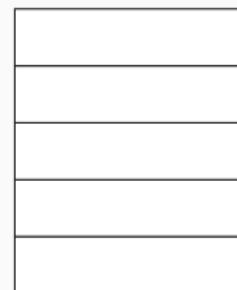


Continue with next address

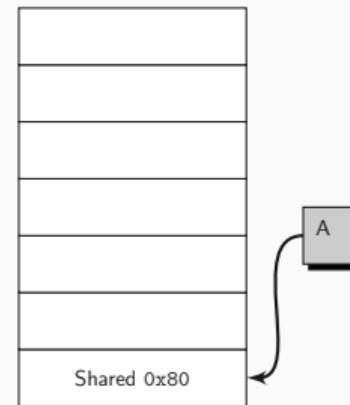
Attacker address space



Cache



Victim address space



Continue with next address

```
# ps -A | grep gedit
# cat /proc/pid/maps
00400000-00489000 r-xp 00000000 08:11 396356
/usr/bin/gedit
7f5a96991000-7f5a96a51000 r-xp 00000000 08:11 399365
/usr/lib/x86_64-linux-gnu/libgdk-3.so.0.1400.14
...
.
```

memory range, access rights, offset, –, –, file name

```
cd ../profiling/generic_low_frequency_example
# put the threshold into spy.c (MIN_CACHE_MISS_CYCLES)
make
./spy
# start the targeted program
sleep 2; ./spy 200          400000-489000  --      20000
-- -- /usr/bin/gedit
```

... and hold down a key in the target program

```
cd ../profiling/generic_low_frequency_example
# put the threshold into spy.c (MIN_CACHE_MISS_CYCLES)
make
./spy
# start the targeted program
sleep 2; ./spy 200          400000-489000  --
-- -- /usr/bin/gedit
```

... and hold down a key in the target program
save addresses with peaks!

```
cd ../exploitation/generic
# put the threshold into spy.c (MIN_CACHE_MISS_CYCLES)
make
./spy file offset
```

Demo / Time to code

The screenshot shows a Linux desktop environment with three windows:

- Terminal Window (Top Left):** The title bar says "Terminal". The menu bar includes "File", "Edit", "View", "Search", "Terminal", and "Help". The command entered is:

```
% sleep 2; ./spy 300 7f05140a4000-7f051417b000 r-xp 0x20000 08:02 26  
8050 /usr/lib/x86_64-linux-gnu/gedit/libgedit.so
```
- Gedit Document Window (Top Right):** The title bar says "Untitled Document 1". The menu bar includes "Save", "Open", and "New". The document content is a single character "I".
- Nautilus File Browser Window (Bottom Left):** The title bar says "Terminal". The menu bar includes "File", "Edit", "View", "Search", "Terminal", and "Help". The sidebar shows a tree view with "Desktop", "Downloads", and "Trolltech". The main pane shows a directory listing for "DIR" dated "14.03.2017 21:44:26". The command entered is:

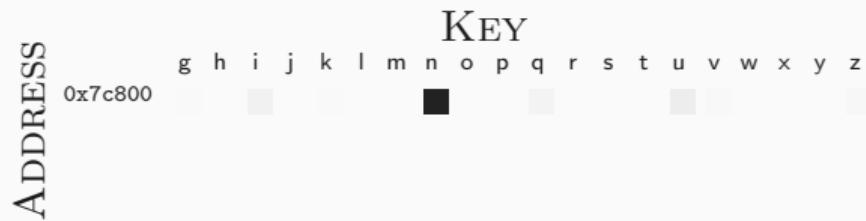
```
shark% ./spy []
```

The status bar at the bottom of the desktop indicates the user is "gnome/gamer" and shows "Plain Text", "Tab Width: 2", "Ln 1, Col 1", and "INS".





Example: Cache Hit Ratio for (0x7c800, n): 200 / 200

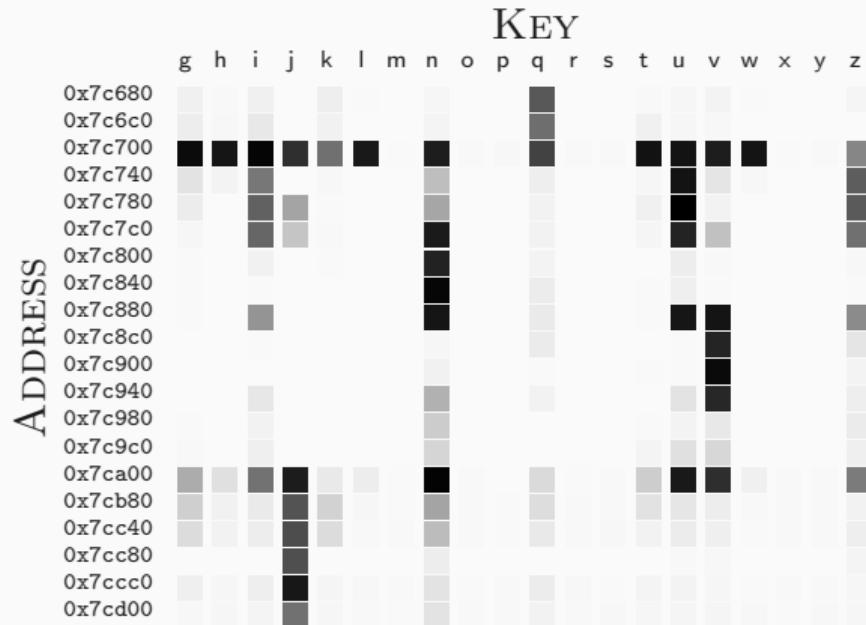




Example: Cache Hit Ratio for (0x7c800, u): 13 / 200



Distinguish **n** from other keys by monitoring 0x7c800



AES uses T-Tables (precomputed from S-Boxes)

- 4 T-Tables

AES uses T-Tables (precomputed from S-Boxes)

- 4 T-Tables
-

$$T_0 [k_{\{0,4,8,12\}} \oplus p_{\{0,4,8,12\}}]$$

$$T_1 [k_{\{1,5,9,13\}} \oplus p_{\{1,5,9,13\}}]$$

$$T_2 [k_{\{2,6,10,14\}} \oplus p_{\{2,6,10,14\}}]$$

$$T_3 [k_{\{3,7,11,15\}} \oplus p_{\{3,7,11,15\}}]$$

AES uses T-Tables (precomputed from S-Boxes)

- 4 T-Tables
-

$$T_0 [k_{\{0,4,8,12\}} \oplus p_{\{0,4,8,12\}}]$$

$$T_1 [k_{\{1,5,9,13\}} \oplus p_{\{1,5,9,13\}}]$$

$$T_2 [k_{\{2,6,10,14\}} \oplus p_{\{2,6,10,14\}}]$$

$$T_3 [k_{\{3,7,11,15\}} \oplus p_{\{3,7,11,15\}}]$$

- If we know which entry of T is accessed, we know the result of $k_i \oplus p_i$.

AES uses T-Tables (precomputed from S-Boxes)

- 4 T-Tables
-

$$T_0 [k_{\{0,4,8,12\}} \oplus p_{\{0,4,8,12\}}]$$

$$T_1 [k_{\{1,5,9,13\}} \oplus p_{\{1,5,9,13\}}]$$

$$T_2 [k_{\{2,6,10,14\}} \oplus p_{\{2,6,10,14\}}]$$

$$T_3 [k_{\{3,7,11,15\}} \oplus p_{\{3,7,11,15\}}]$$

- If we know which entry of T is accessed, we know the result of $k_i \oplus p_i$.
- Known-plaintext attack (p_i is known) $\rightarrow k_i$ can be determined

- Just to see whether the attack works

- Just to see whether the attack works
- Recovers 64 out of 128 bits

- Just to see whether the attack works
- Recovers 64 out of 128 bits
- Cache hits depend on key value and plaintext value

- Just to see whether the attack works
- Recovers 64 out of 128 bits
- Cache hits depend on key value and plaintext value
- Random key/data → cache hit ratio of 92%

1. Choose a cache line of T_0

1. Choose a cache line of T_0
2. Fix first key byte to a value $\in 0, 16, 32, \dots, 256$ (other bytes random)

1. Choose a cache line of T_0
2. Fix first key byte to a value $\in 0, 16, 32, \dots, 256$ (other bytes random)
 - or plaintext byte, it's symmetric

1. Choose a cache line of T_0
2. Fix first key byte to a value $\in 0, 16, 32, \dots, 256$ (other bytes random)
 - or plaintext byte, it's symmetric
3. Encrypt

1. Choose a cache line of T_0
2. Fix first key byte to a value $\in 0, 16, 32, \dots, 256$ (other bytes random)
 - or plaintext byte, it's symmetric
3. Encrypt
4. Check whether the cache line is cached

1. Choose a cache line of T_0
2. Fix first key byte to a value $\in 0, 16, 32, \dots, 256$ (other bytes random)
 - or plaintext byte, it's symmetric
3. Encrypt
4. Check whether the cache line is cached
5. Repeat

1. Choose a cache line of T_0
2. Fix first key byte to a value $\in 0, 16, 32, \dots, 256$ (other bytes random)
 - or plaintext byte, it's symmetric
3. Encrypt
4. Check whether the cache line is cached
5. Repeat
 - Cache hit ratio 100% $\rightarrow k_0 \oplus p_0$ fell into that cache line

1. Choose a cache line of T_0
2. Fix first key byte to a value $\in 0, 16, 32, \dots, 256$ (other bytes random)
 - or plaintext byte, it's symmetric
3. Encrypt
4. Check whether the cache line is cached
5. Repeat
 - Cache hit ratio 100% $\rightarrow k_0 \oplus p_0$ fell into that cache line
 - Otherwise: cache hit ratio will be around 92%

Demo / Time to code

- Oh no! The compiler didn't align the T-Table to a 1 KB boundary!

- Oh no! The compiler didn't align the T-Table to a 1 KB boundary!
- Is that a problem?

- Oh no! The compiler didn't align the T-Table to a 1 KB boundary!
- Is that a problem?
- $T_0[255]$, $T_1[255]$, $T_2[255]$, and $T_3[255]$ will be on separate cache lines

- Oh no! The compiler didn't align the T-Table to a 1 KB boundary!
- Is that a problem?
- $T_0[255]$, $T_1[255]$, $T_2[255]$, and $T_3[255]$ will be on separate cache lines
- if $k_i \oplus p_i = 255$ we will see an access on that corresponding cache line

For all 16 key bytes:

1. Choose the corresponding cache line ($T_0[255]$, $T_1[255]$, $T_2[255]$, or $T_3[255]$)

For all 16 key bytes:

1. Choose the corresponding cache line ($T_0[255]$, $T_1[255]$, $T_2[255]$, or $T_3[255]$)
2. Fix the corresponding plaintext byte to a value $\in 0, 1, 2, \dots, 256$ (other bytes random)

For all 16 key bytes:

1. Choose the corresponding cache line ($T_0[255]$, $T_1[255]$, $T_2[255]$, or $T_3[255]$)
2. Fix the corresponding plaintext byte to a value $\in 0, 1, 2, \dots, 256$ (other bytes random)
3. Encrypt

For all 16 key bytes:

1. Choose the corresponding cache line ($T_0[255]$, $T_1[255]$, $T_2[255]$, or $T_3[255]$)
2. Fix the corresponding plaintext byte to a value $\in 0, 1, 2, \dots, 256$ (other bytes random)
3. Encrypt
4. Check whether the cache line is cached

For all 16 key bytes:

1. Choose the corresponding cache line ($T_0[255]$, $T_1[255]$, $T_2[255]$, or $T_3[255]$)
2. Fix the corresponding plaintext byte to a value $\in 0, 1, 2, \dots, 256$ (other bytes random)
3. Encrypt
4. Check whether the cache line is cached
5. Repeat

For all 16 key bytes:

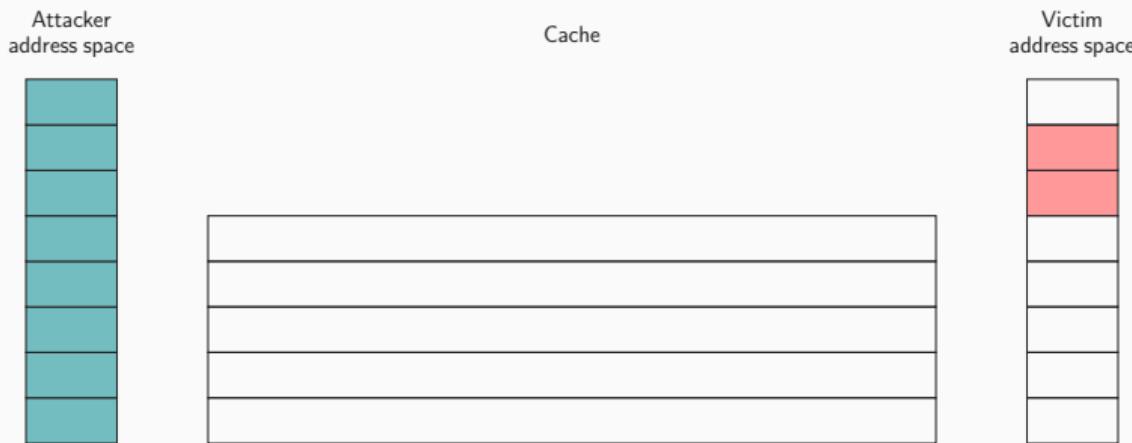
1. Choose the corresponding cache line ($T_0[255]$, $T_1[255]$, $T_2[255]$, or $T_3[255]$)
2. Fix the corresponding plaintext byte to a value $\in 0, 1, 2, \dots, 256$ (other bytes random)
3. Encrypt
4. Check whether the cache line is cached
5. Repeat

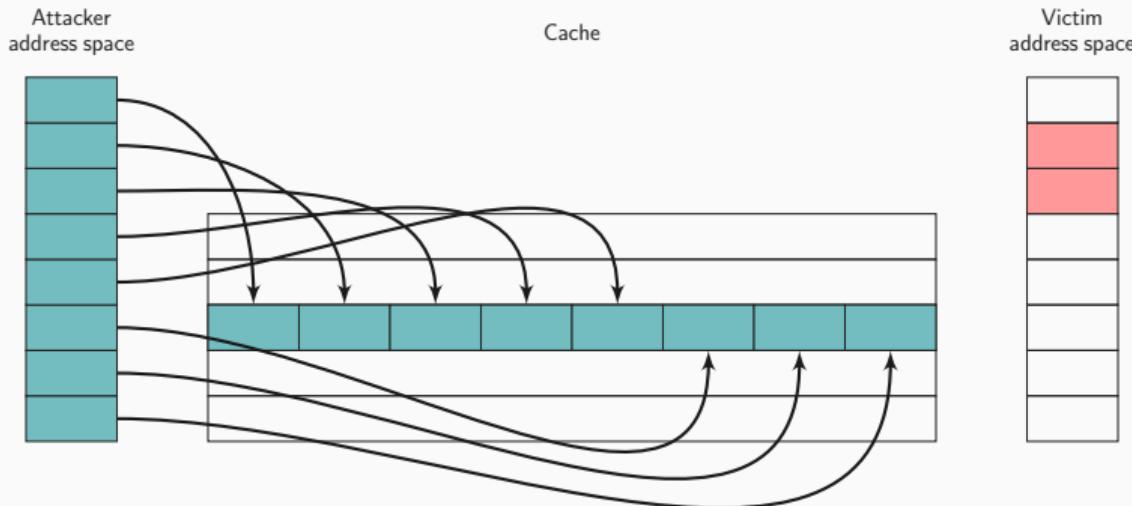
For all 16 key bytes:

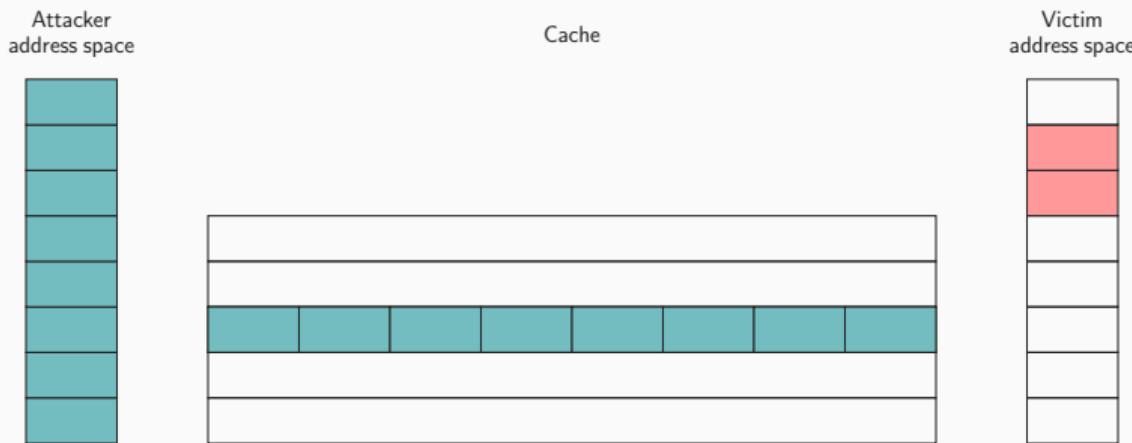
1. Choose the corresponding cache line ($T_0[255]$, $T_1[255]$, $T_2[255]$, or $T_3[255]$)
2. Fix the corresponding plaintext byte to a value $\in 0, 1, 2, \dots, 256$ (other bytes random)
3. Encrypt
4. Check whether the cache line is cached
5. Repeat

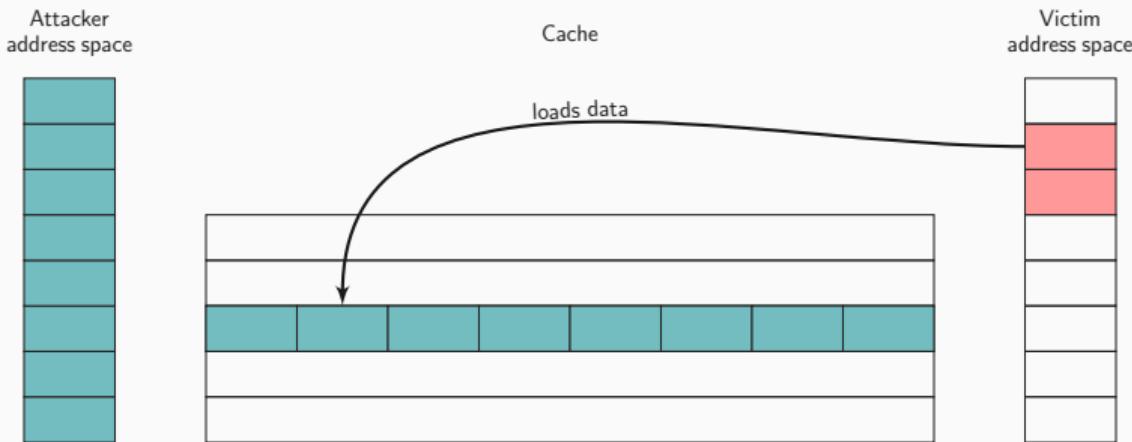
→ Obtain the key: $k_i \oplus p_i = 255 \rightarrow k_i = p_i \oplus 255$

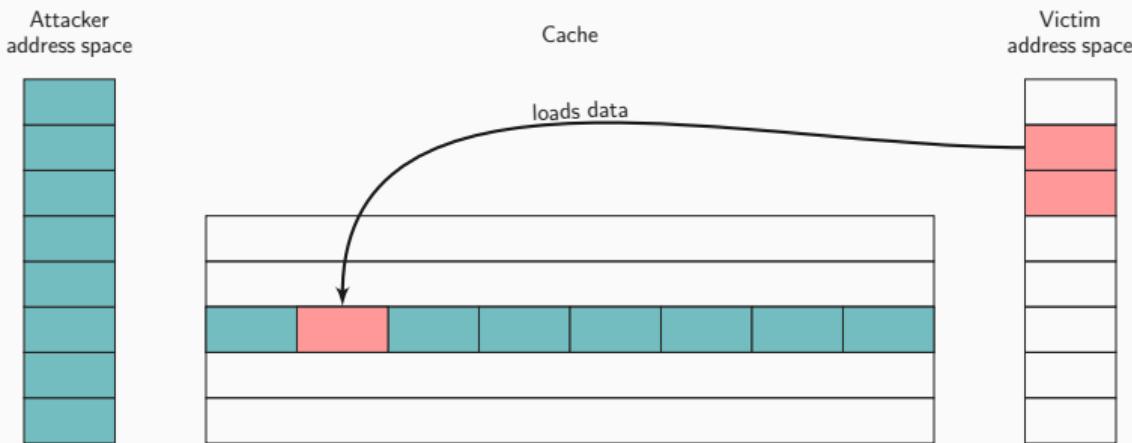
Demo / Time to code

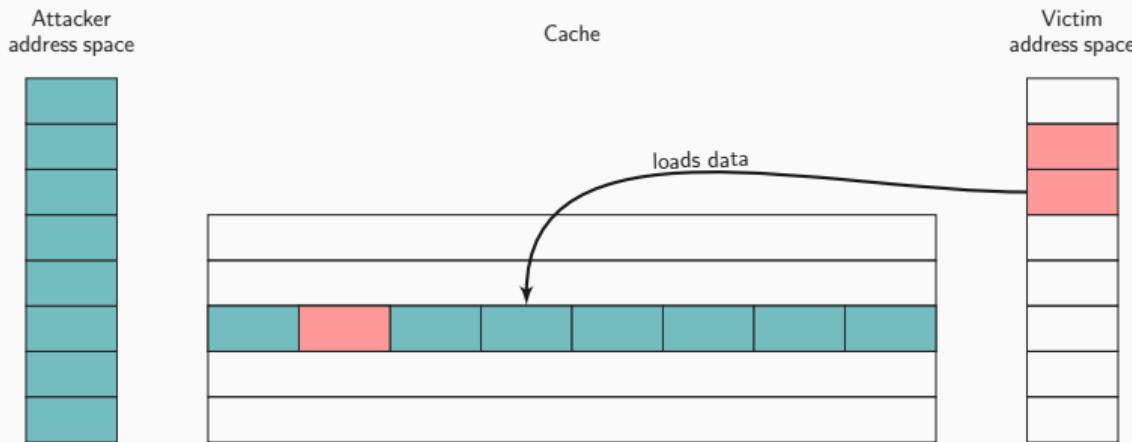


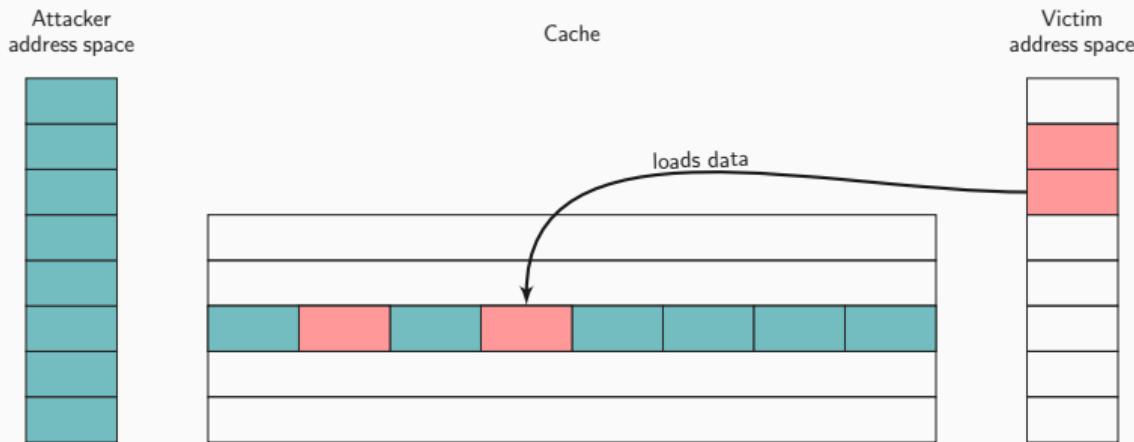


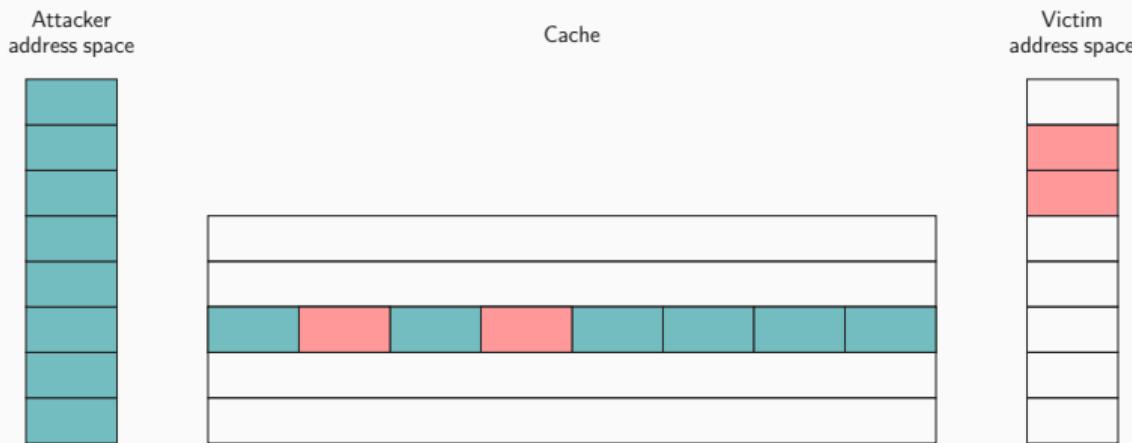


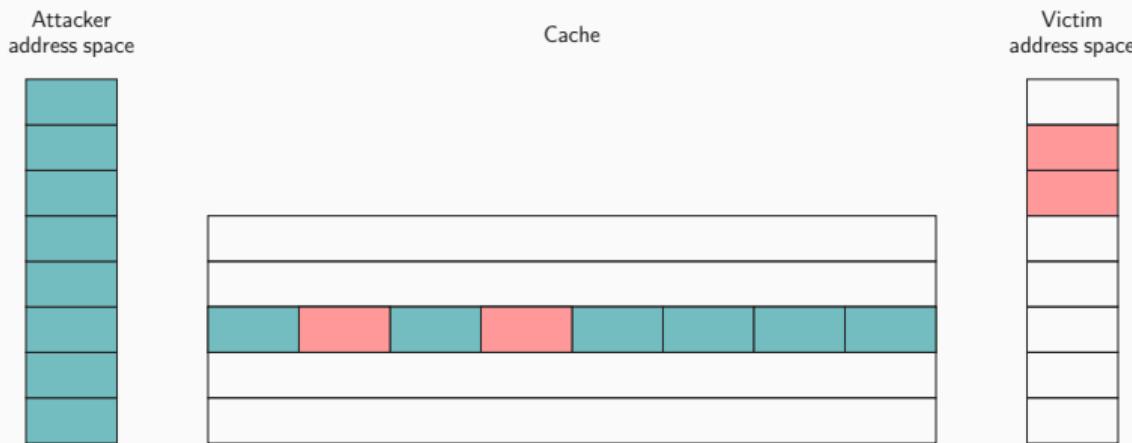


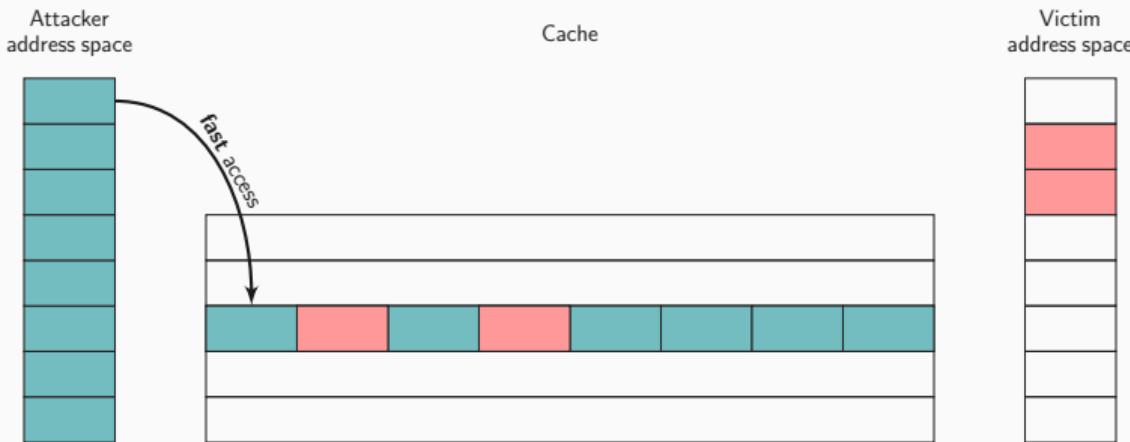


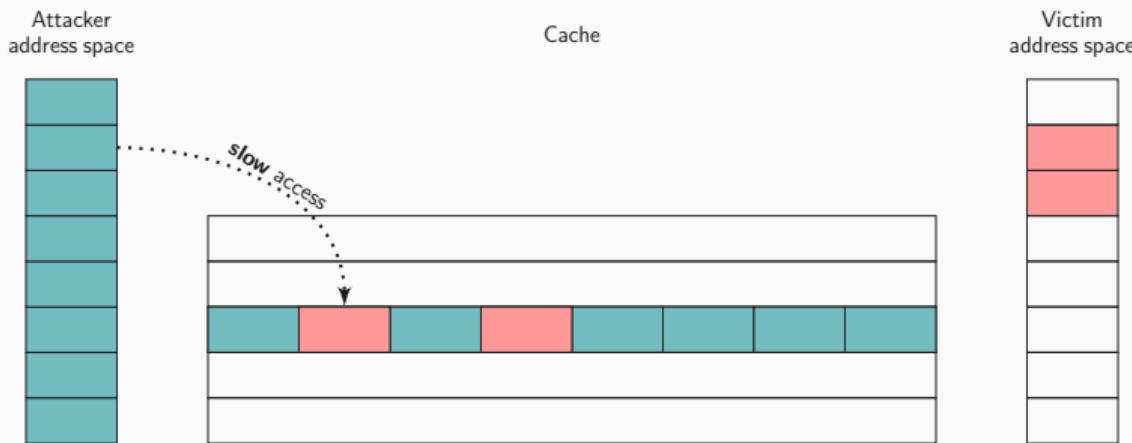












Pros: less restrictive

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

Pros: less restrictive

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

Pros: less restrictive

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

Pros: less restrictive

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

Cons: coarser granularity (1 set)

“LRU eviction” memory accesses

cache set



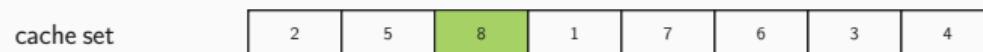
“LRU eviction” memory accesses

cache set



- LRU replacement policy: oldest entry first

“LRU eviction” memory accesses



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

“LRU eviction” memory accesses



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

“LRU eviction” memory accesses



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

“LRU eviction” memory accesses



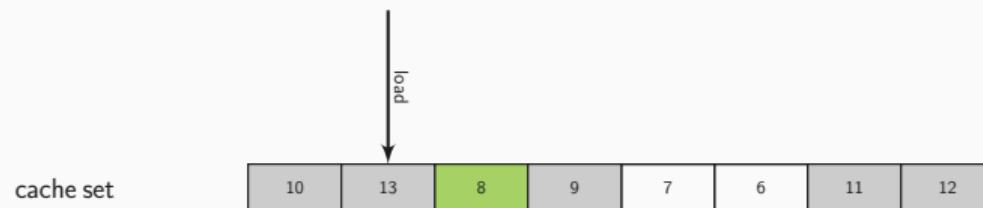
- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

“LRU eviction” memory accesses



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

“LRU eviction” memory accesses



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

“LRU eviction” memory accesses



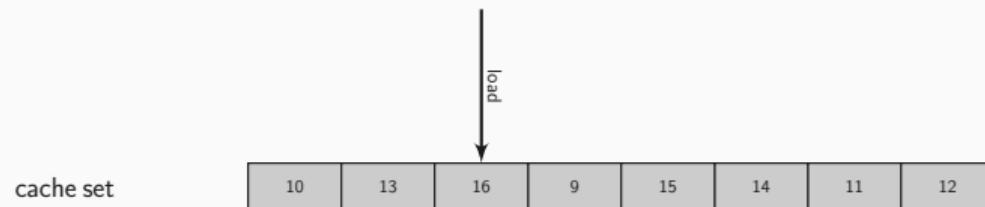
- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

“LRU eviction” memory accesses



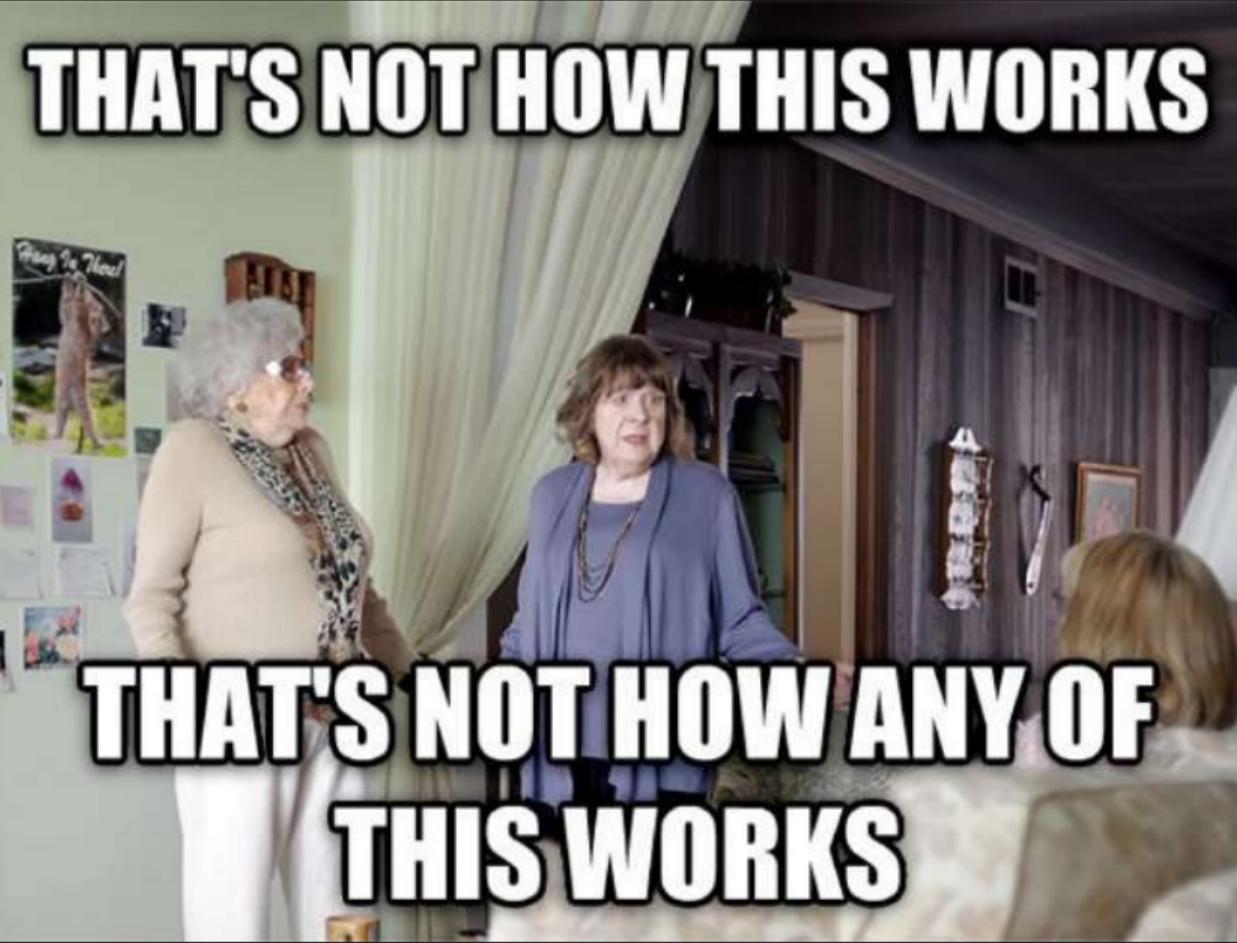
- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

“LRU eviction” memory accesses



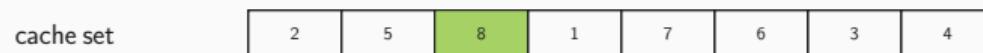
- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

THAT'S NOT HOW THIS WORKS

A photograph of two women in a living room. On the left, an older woman with grey hair, wearing a light beige sweater over a patterned scarf, looks towards the right. In the center, another woman with brown hair, wearing a blue top and a necklace, also looks towards the right. In the bottom right corner, the back of a third person's head is visible, showing blonde hair. The background features a window with green curtains, a wooden-paneled wall, and various decorations like a small framed picture and a hanging basket.

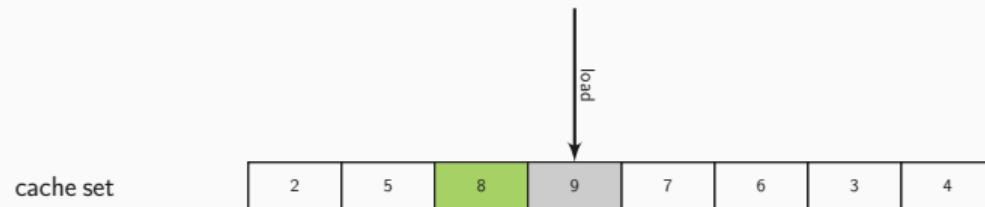
**THAT'S NOT HOW ANY OF
THIS WORKS**

“LRU eviction” memory accesses



- no LRU replacement

“LRU eviction” memory accesses



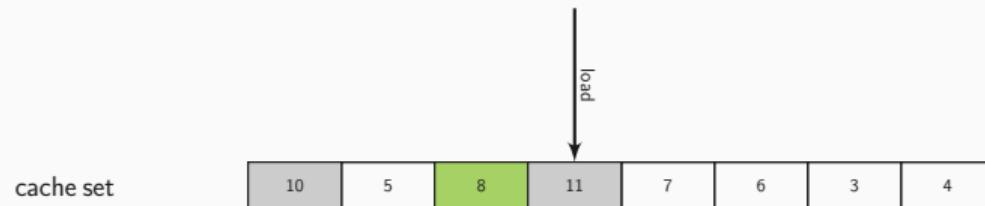
- no LRU replacement

“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



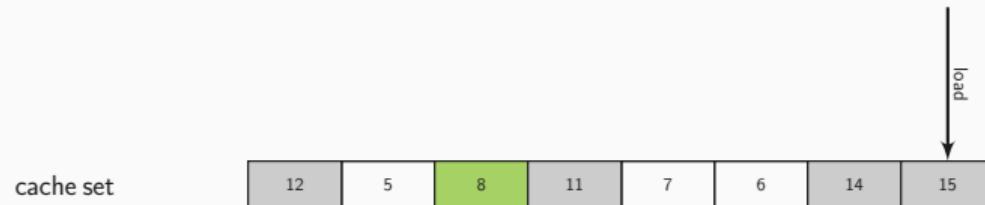
- no LRU replacement

“LRU eviction” memory accesses



- no LRU replacement

“LRU eviction” memory accesses



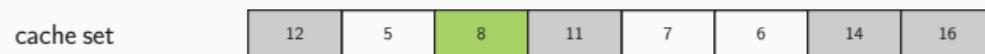
- no LRU replacement

“LRU eviction” memory accesses



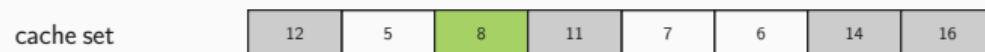
- no LRU replacement

“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

```
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];
```

The diagram illustrates a nested loop structure. A curly brace above the first two loops is labeled 'loop'. Inside the first loop, the expression $S - D$ is highlighted in orange. Inside the second loop, the variable C is highlighted in purple. Inside the third loop, the variable D is highlighted in orange. The step value L is highlighted in green. The entire outermost loop is bracketed by a large curly brace labeled 'loop'.

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

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

D: different addresses per inner access

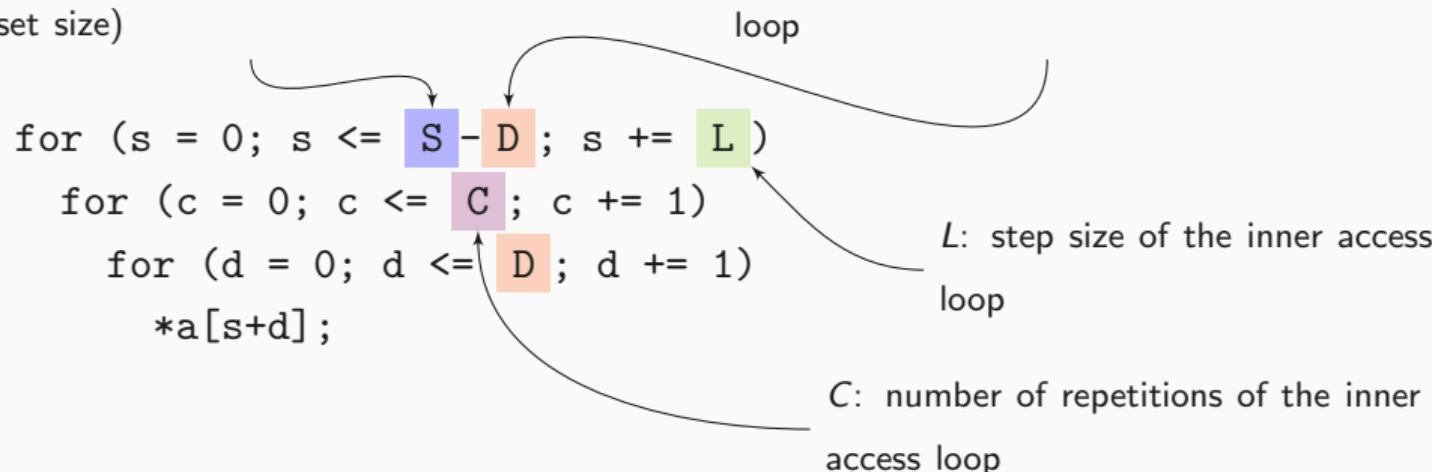
```
for (s = 0; s <= S - D ; s += L)
    for (c = 0; c <= C ; c += 1)
        for (d = 0; d <= D ; d += 1)
            *a[s+d] ;
```

loop

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)



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		

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

We evaluated more than 10000 strategies...¹

strategy	# accesses	eviction rate	loop time
$P-1-1-1-17$	17	74.46% X	
$P-1-1-1-20$	20	99.82% ✓	

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

We evaluated more than 10000 strategies...¹

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

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

We evaluated more than 10000 strategies...¹

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

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

We evaluated more than 10000 strategies...¹

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

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

We evaluated more than 10000 strategies...¹

strategy	# accesses	eviction rate	loop time
$P\text{-}1\text{-}1\text{-}1\text{-}17$	17	74.46% ✗	307 ns ✓
$P\text{-}1\text{-}1\text{-}1\text{-}20$	20	99.82% ✓	934 ns ✗
$P\text{-}2\text{-}1\text{-}1\text{-}17$	34	99.86% ✓	191 ns ✓

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

We evaluated more than 10000 strategies...¹

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

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

We evaluated more than 10000 strategies...¹

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

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

We evaluated more than 10000 strategies...¹

strategy	# accesses	eviction rate	loop time
$P\text{-}1\text{-}1\text{-}1\text{-}17$	17	74.46% ✗	307 ns ✓
$P\text{-}1\text{-}1\text{-}1\text{-}20$	20	99.82% ✓	934 ns ✗
$P\text{-}2\text{-}1\text{-}1\text{-}17$	34	99.86% ✓	191 ns ✓
$P\text{-}2\text{-}2\text{-}1\text{-}17$	64	99.98% ✓	180 ns ✓

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

WHY?

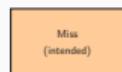


P-1-1-1-17 (17 accesses, 307ns)

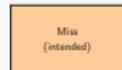
P-2-1-1-17 (34 accesses, 191ns)

Time in ns

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

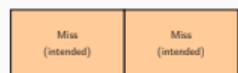


P-2-1-1-17 (34 accesses, 191ns)

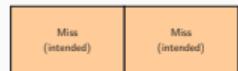


Time in ns →

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

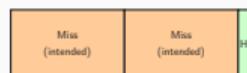


P-2-1-1-17 (34 accesses, 191ns)

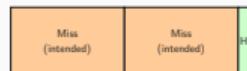


Time in ns →

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



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



Time in ns →

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



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



Time in ns →

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



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



Time in ns →

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



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



Time in ns

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



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



Time in ns →

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



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



Time in ns

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



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



Time in ns →

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



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

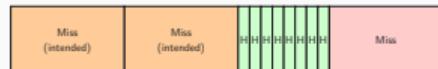


Time in ns →

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



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

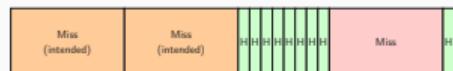


Time in ns →

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

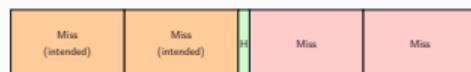


P-2-1-1-17 (34 accesses, 191ns)

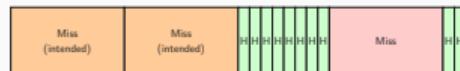


Time in ns

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



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

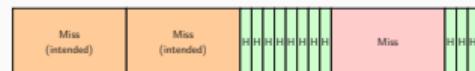


Time in ns →

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



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

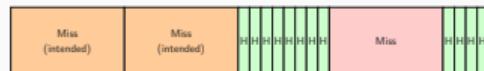


Time in ns

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



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

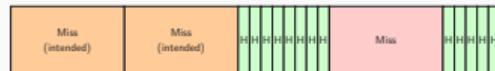


Time in ns →

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



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

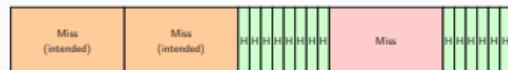


Time in ns →

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



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

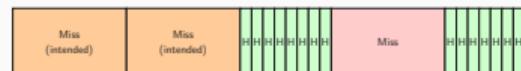


Time in ns →

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



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

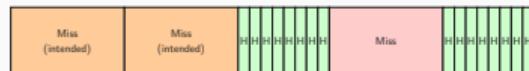


Time in ns →

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

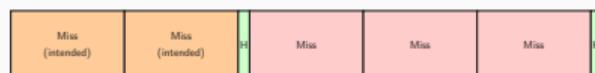


P-2-1-1-17 (34 accesses, 191ns)

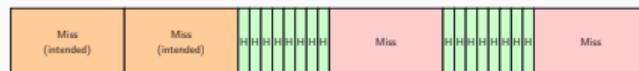


Time in ns

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

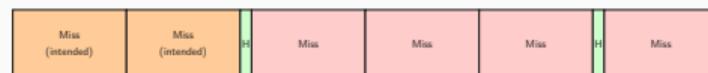


P-2-1-1-17 (34 accesses, 191ns)



Time in ns →

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

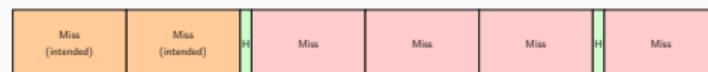


P-2-1-1-17 (34 accesses, 191ns)

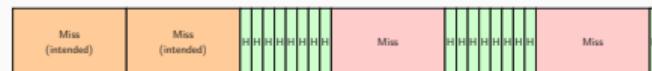


Time in ns →

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

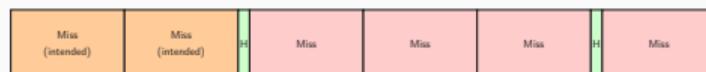


P-2-1-1-17 (34 accesses, 191ns)

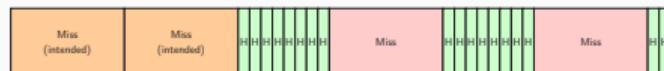


Time in ns

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

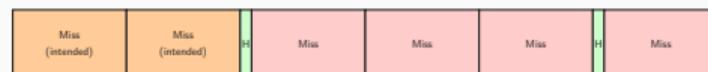


P-2-1-1-17 (34 accesses, 191ns)

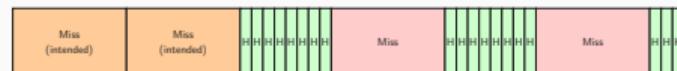


Time in ns

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

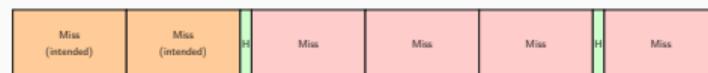


P-2-1-1-17 (34 accesses, 191ns)

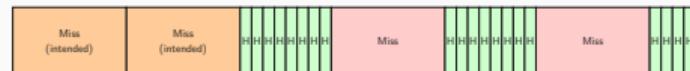


Time in ns

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

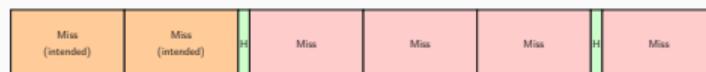


P-2-1-1-17 (34 accesses, 191ns)

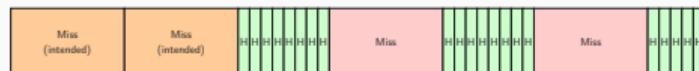


Time in ns →

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

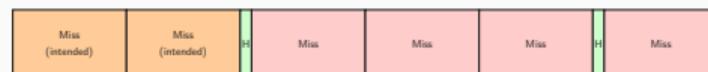


P-2-1-1-17 (34 accesses, 191ns)



Time in ns →

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



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

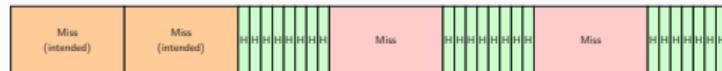


Time in ns →

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



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

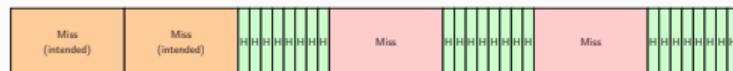


Time in ns →

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

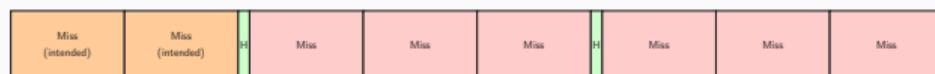


P-2-1-1-17 (34 accesses, 191ns)



Time in ns →

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

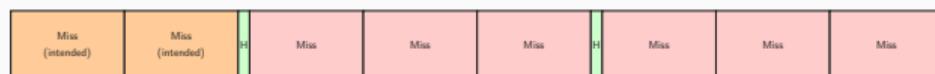


P-2-1-1-17 (34 accesses, 191ns)

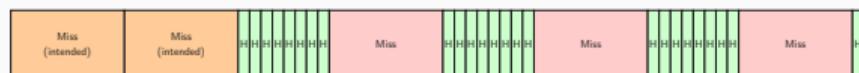


Time in ns →

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

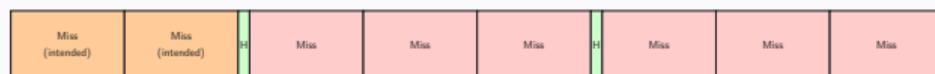


P-2-1-1-17 (34 accesses, 191ns)

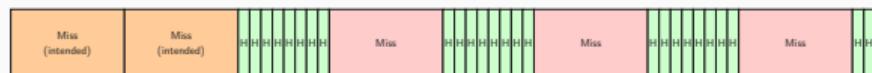


Time in ns →

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

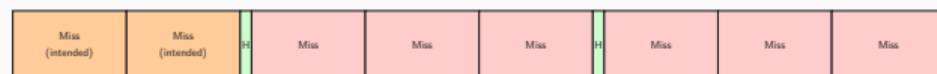


P-2-1-1-17 (34 accesses, 191ns)

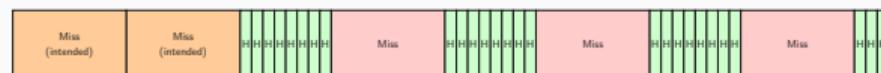


Time in ns →

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

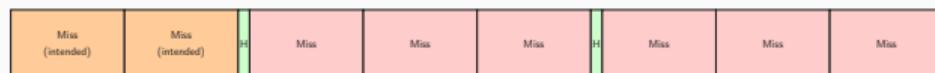


P-2-1-1-17 (34 accesses, 191ns)

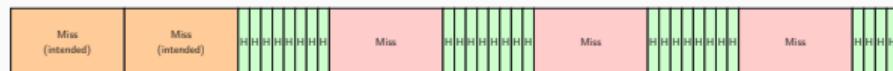


Time in ns →

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

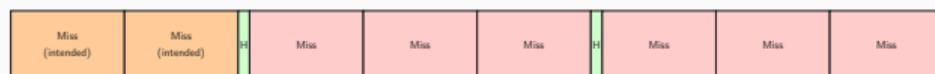


P-2-1-1-17 (34 accesses, 191ns)

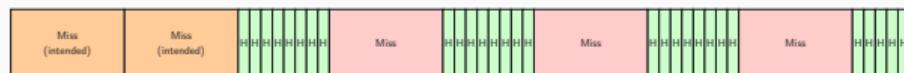


Time in ns →

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

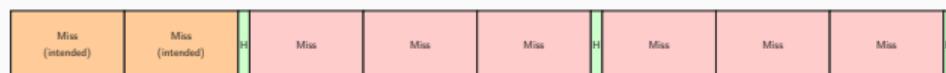


P-2-1-1-17 (34 accesses, 191ns)

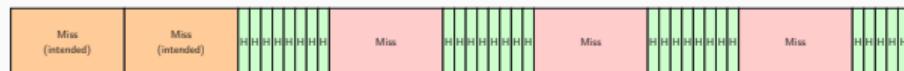


Time in ns →

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



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

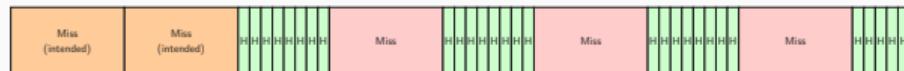


Time in ns →

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

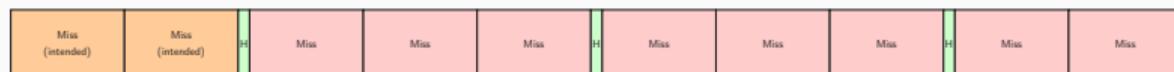


P-2-1-1-17 (34 accesses, 191ns)

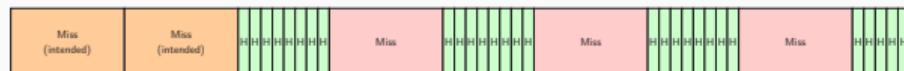


Time in ns →

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



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

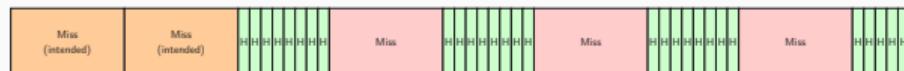


Time in ns →

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



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

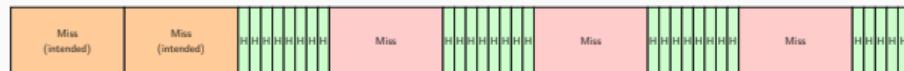


Time in ns →

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

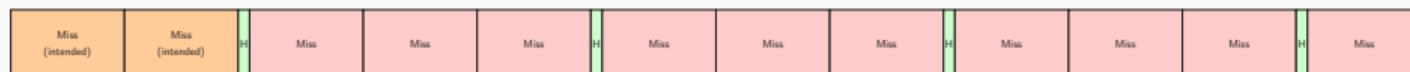


P-2-1-1-17 (34 accesses, 191ns)

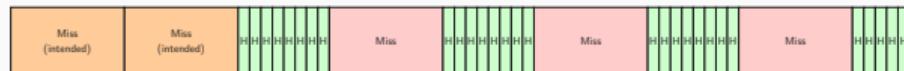


Time in ns →

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



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

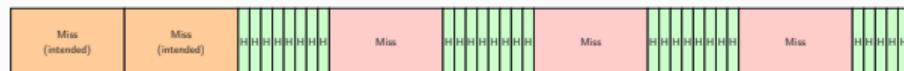


Time in ns →

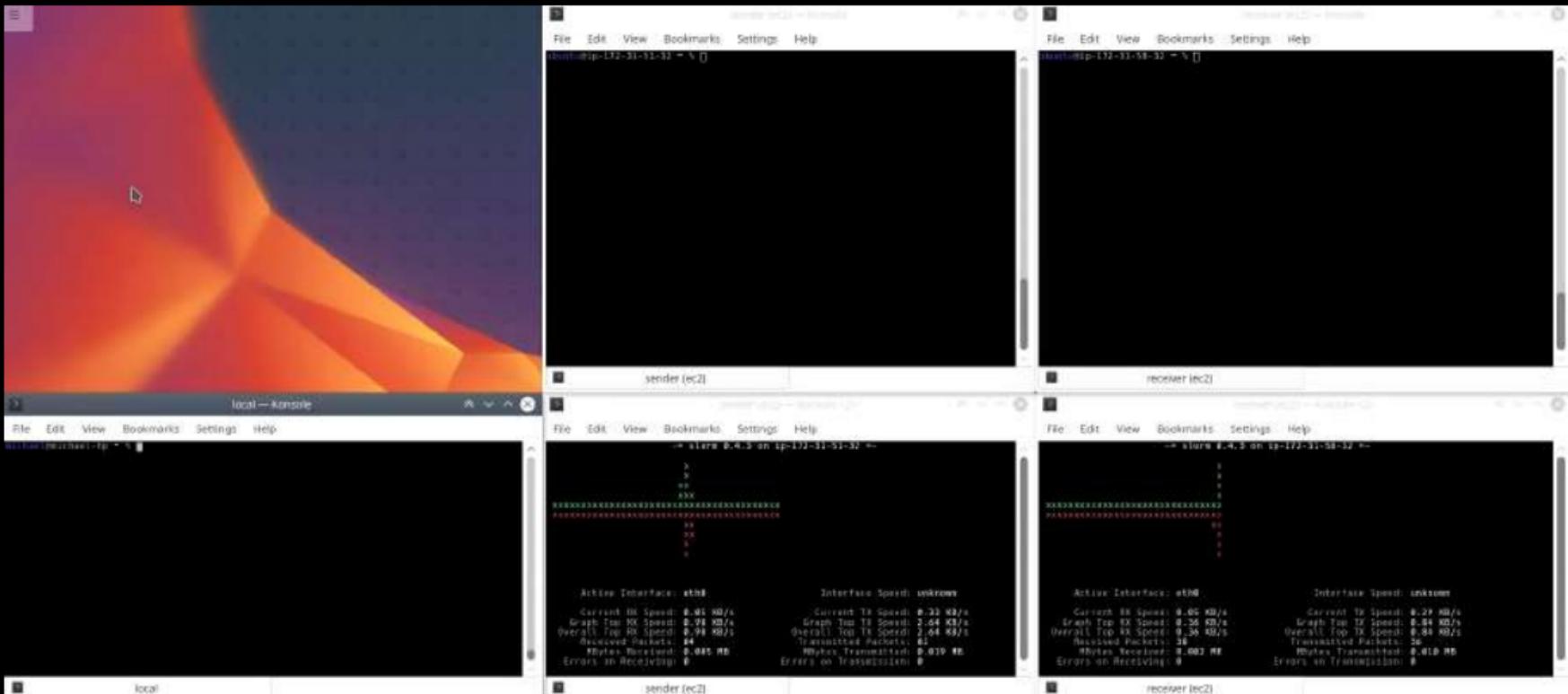
P-1-1-1-17 (17 accesses, 307ns)



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



Time in ns →



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

Protection from Side-Channel Attacks

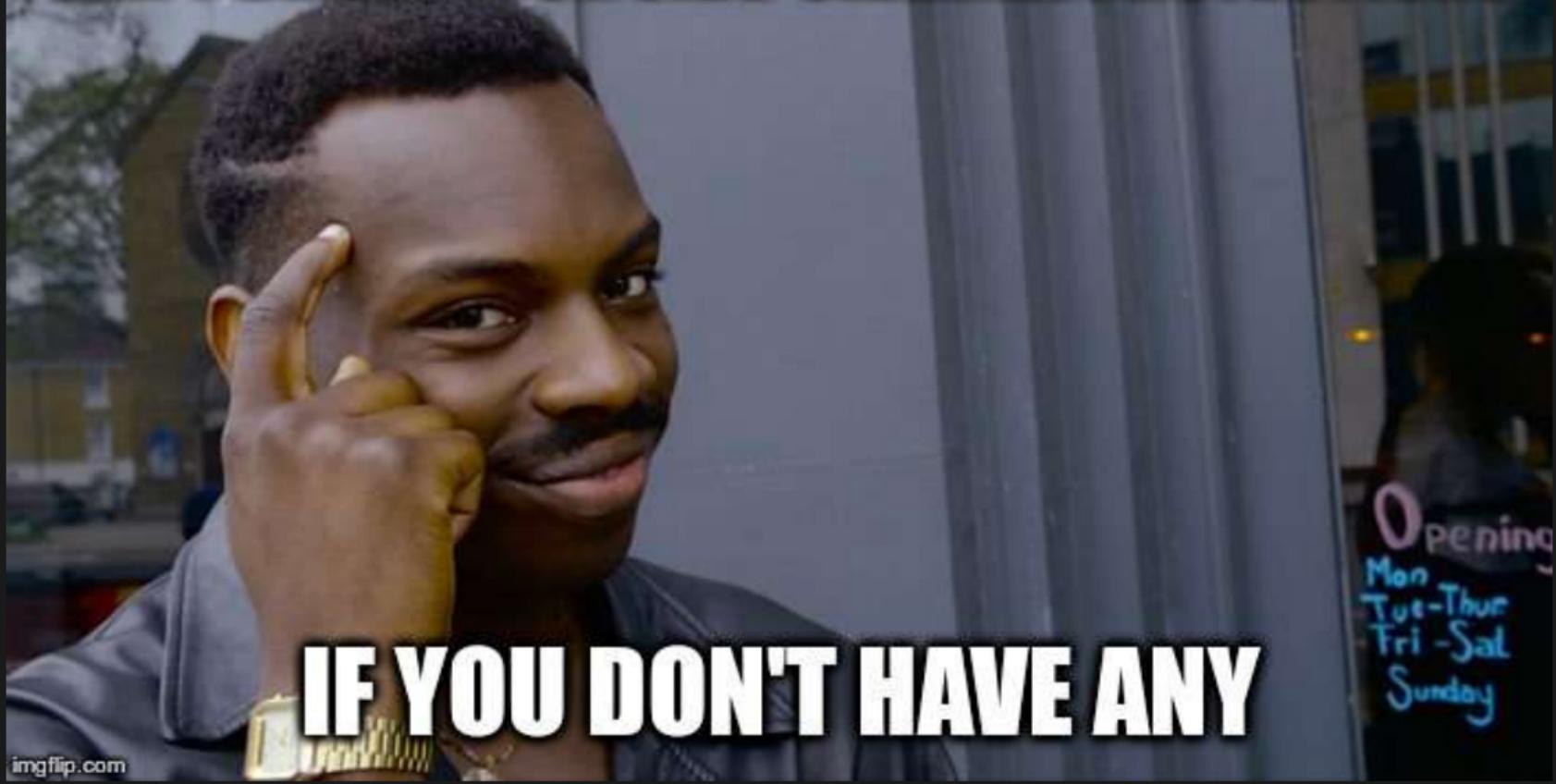
Protection from Side-Channel Attacks

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

Protection from Side-Channel Attacks

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

CAN'T BREAK YOUR SIDE-CHANNEL PROTECTIONS



IF YOU DON'T HAVE ANY

Opening
Mon
Tue-Thur
Tri -Sal
Sunday



- 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.

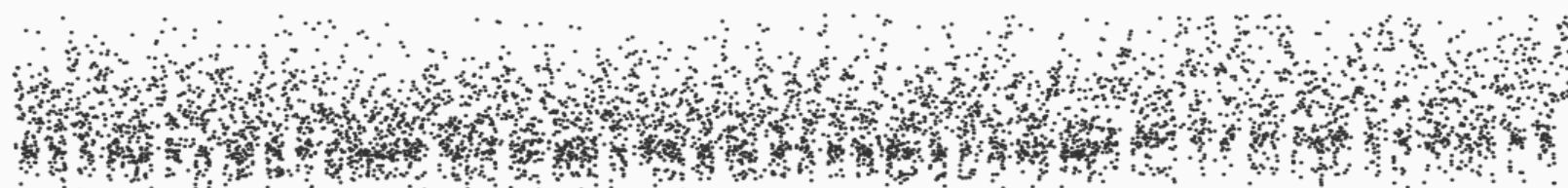


- Ledger SGX Enclave for blockchain applications
- BitPay Copay Bitcoin wallet
- Teechain payment channel using SGX

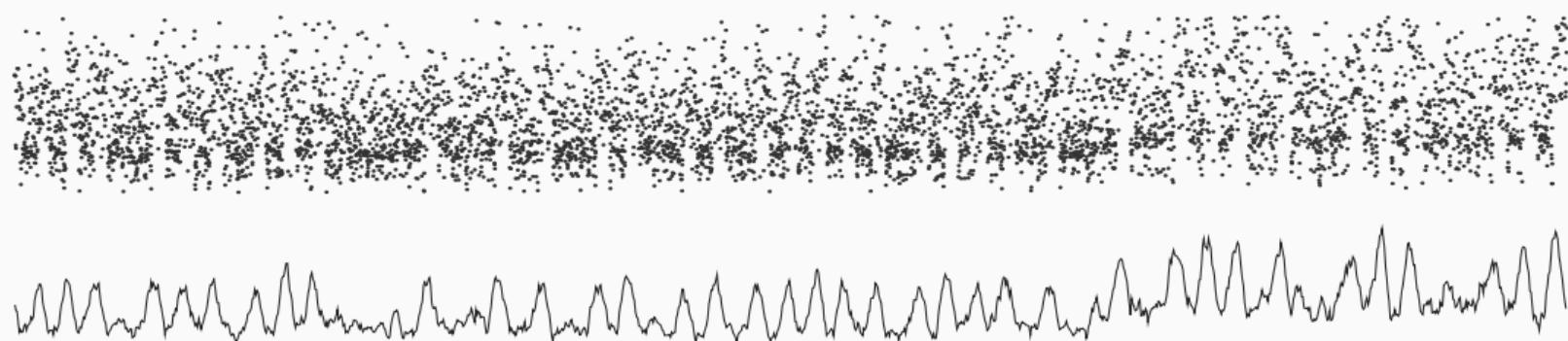
Teechain

[...] 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. [...]

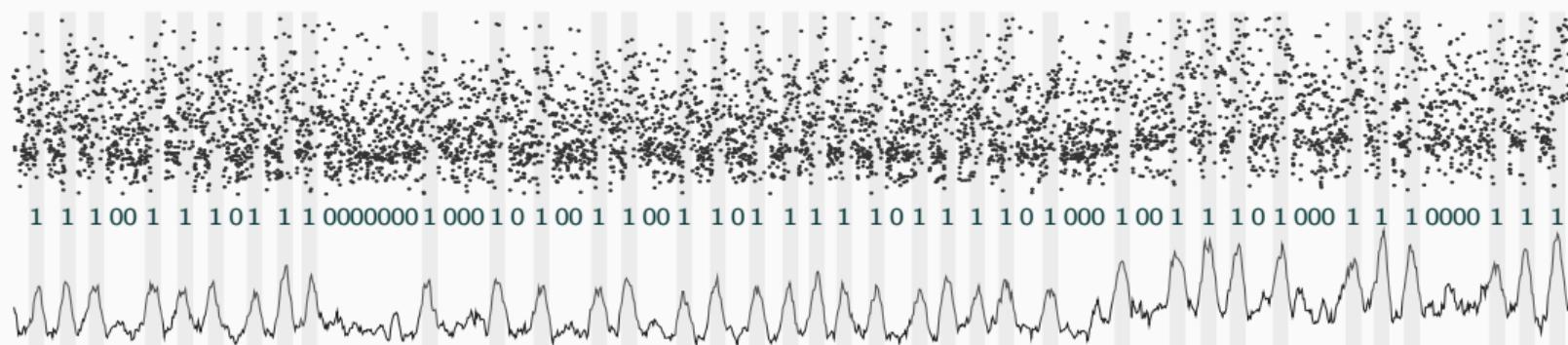
Raw Prime+Probe trace...



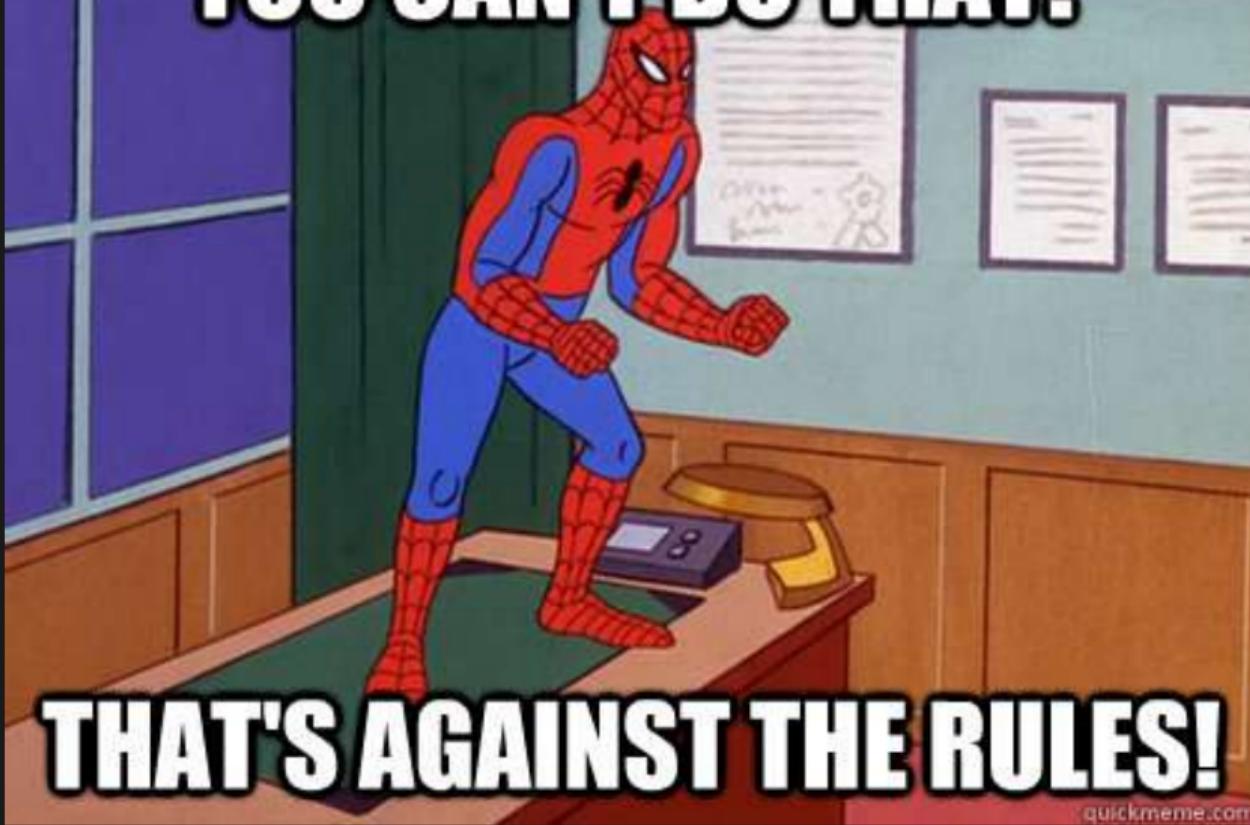
...processed with a simple moving average...



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



YOU CAN'T DO THAT!



THAT'S AGAINST THE RULES!



Back to Work

*7. Serve with cooked
and peeled potatoes*





Wait for an hour





Wait for an hour

LATENCY

*1. Wash and cut
vegetables*

*2. Pick the basil leaves
and set aside*

*3. Heat 2 tablespoons of
oil in a pan*

*4. Fry vegetables until
golden and softened*



Dependency

1. Wash and cut vegetables

2. Pick the basil leaves and set aside

3. Heat 2 tablespoons of oil in a pan

4. Fry vegetables until golden and softened



Parallelize



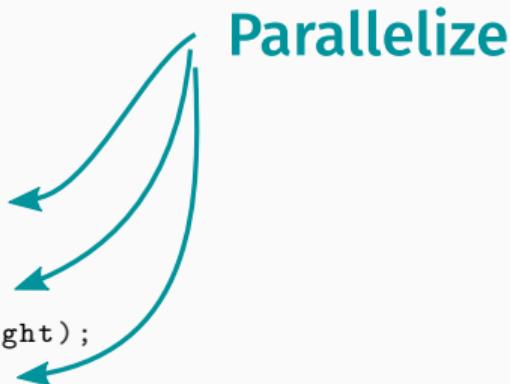
```
int width = 10, height = 5;

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

printf("Area %d x %d = %d\n", width, height, area);
```

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);
```



Parallelize



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

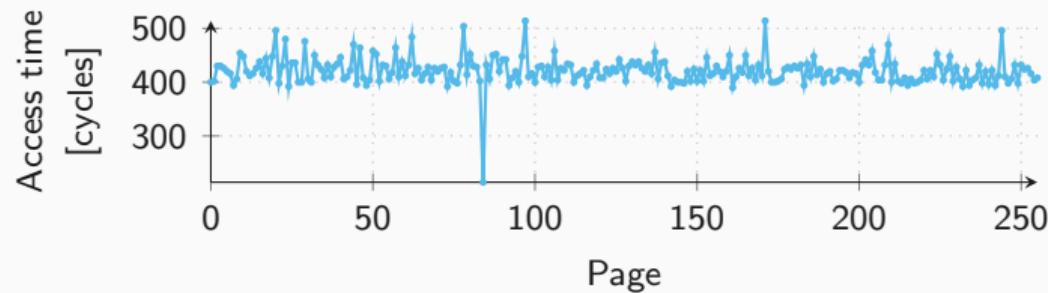


- Flush+Reload over all pages of the array





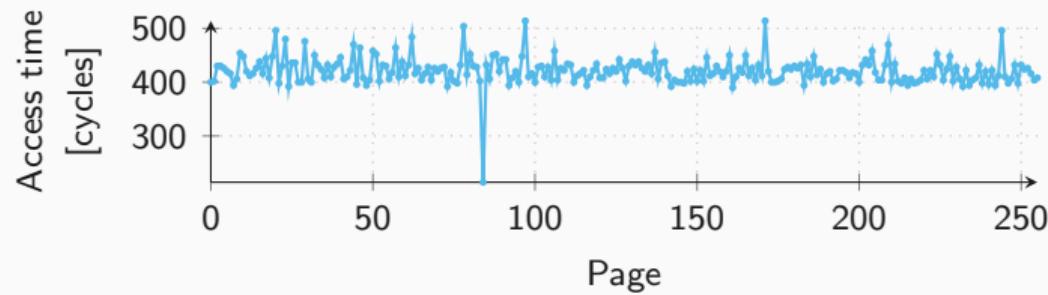
- Flush+Reload over all pages of the array



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



- Flush+Reload over all pages of the array



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



- Out-of-order instructions leave microarchitectural traces



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



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



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



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

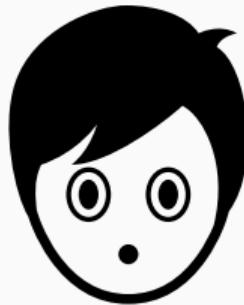
```
char data = *(char*) 0xffffffff81a000e0;  
array[data * 4096] = 0;
```



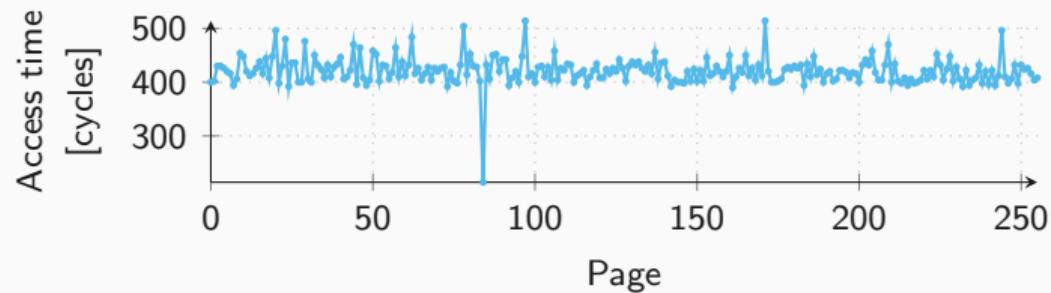
- Add another **layer of indirection** to test

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

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



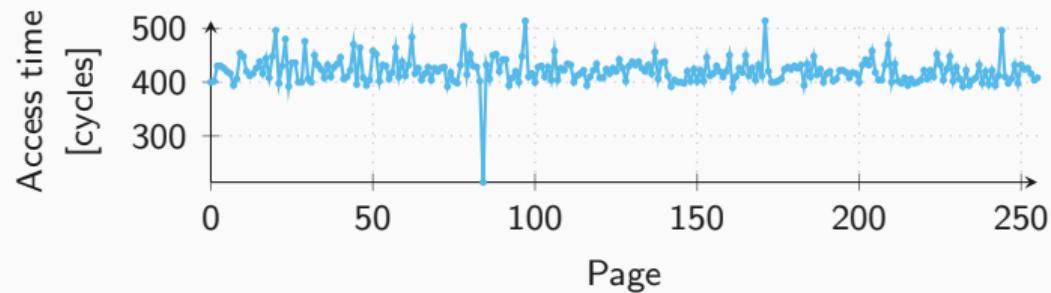
- Flush+Reload over all pages of the array



- Index of cache hit reveals **data**



- Flush+Reload over all pages of the array

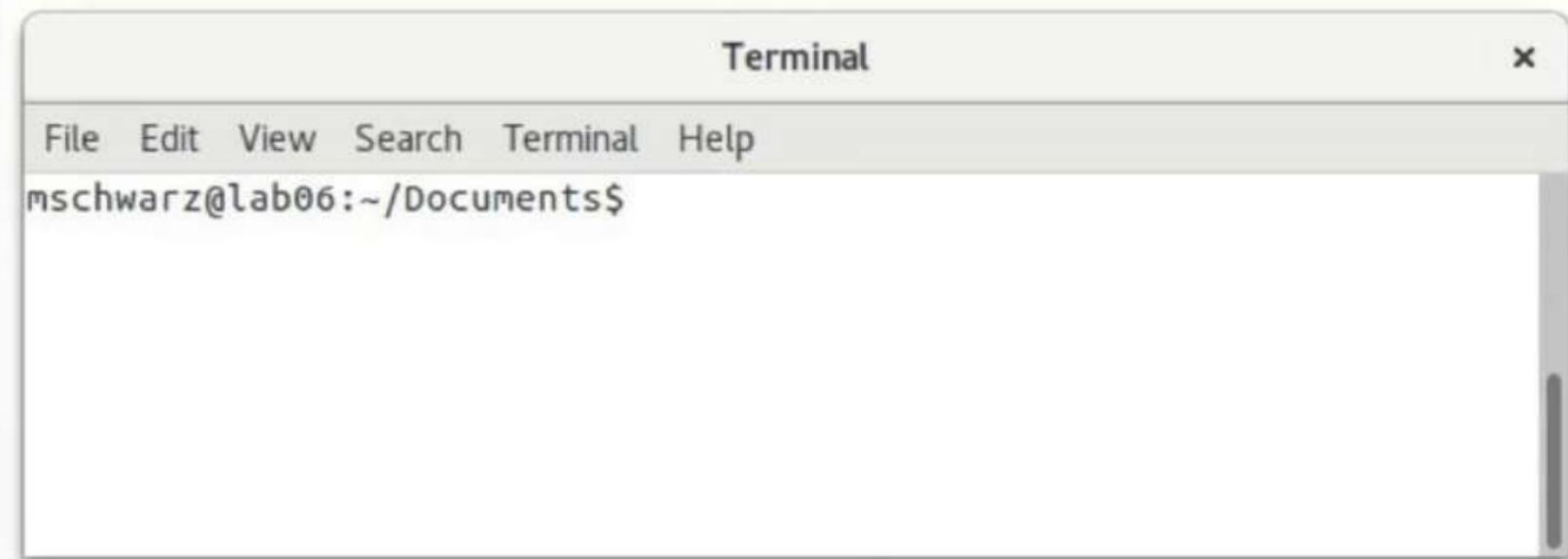
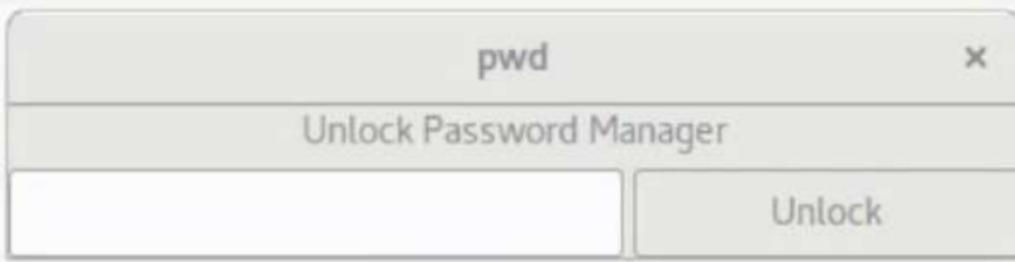


- Index of cache hit reveals **data**
- Permission check is in some cases **not fast enough**

I SHIT YOU NOT

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





File Edit View Search Terminal Help

attacker@meltdown ~/exploit %

File Edit View Search Terminal Help

victim@meltdown ~ %





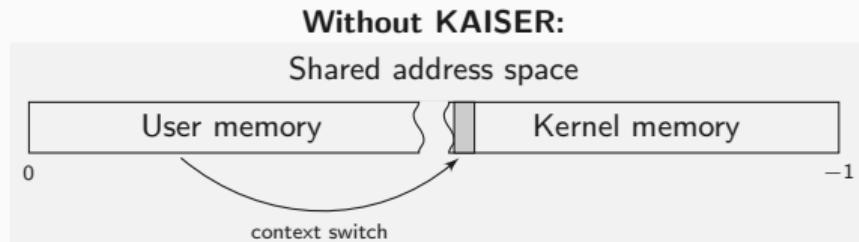
K_{er}nel A_{dd}ress I_{sol}ation to have S_{ide} channels E_{fficiently} R_{emoved}

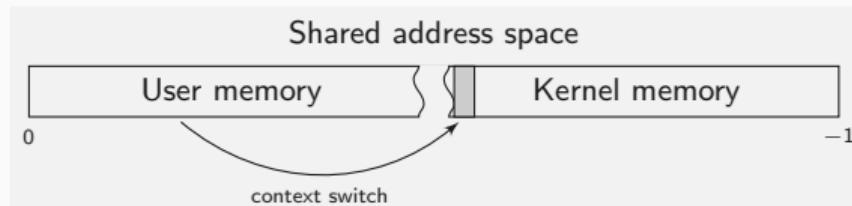
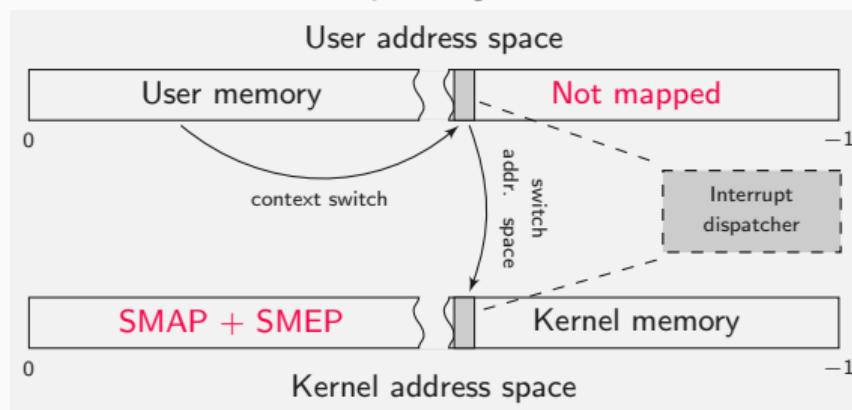
KAISER /'kʌɪzə/

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



K_{er}nel A_{dd}ress I_{sol}ation to have S_{ide} channels E_{fficiently} R_{emoved}



Without KAISER:**With KAISER:**

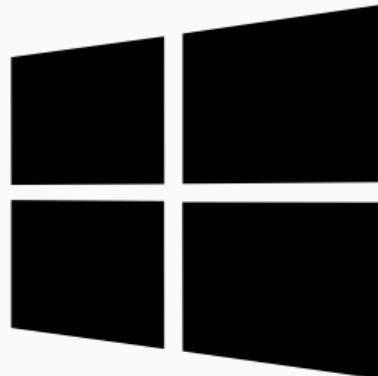




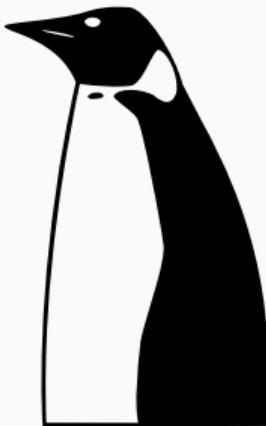
- Our patch
- Adopted in
Linux



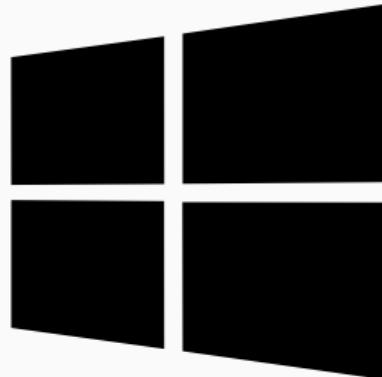
- Our patch
- Adopted in Linux



- Adopted in Windows



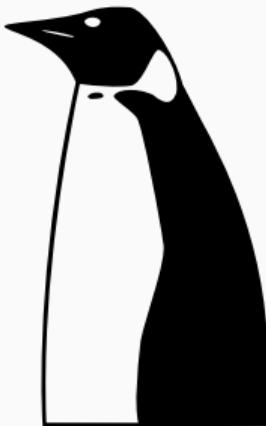
- Our patch
- Adopted in Linux



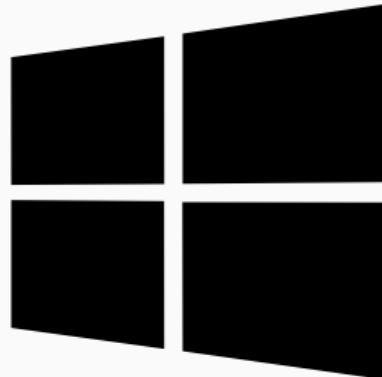
- Adopted in Windows



- Adopted in OSX/iOS



- Our patch
- Adopted in Linux



- Adopted in Windows



- Adopted in OSX/iOS

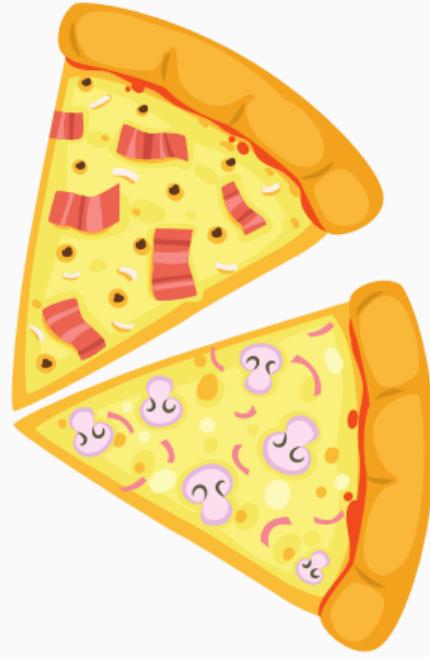
→ now in every computer

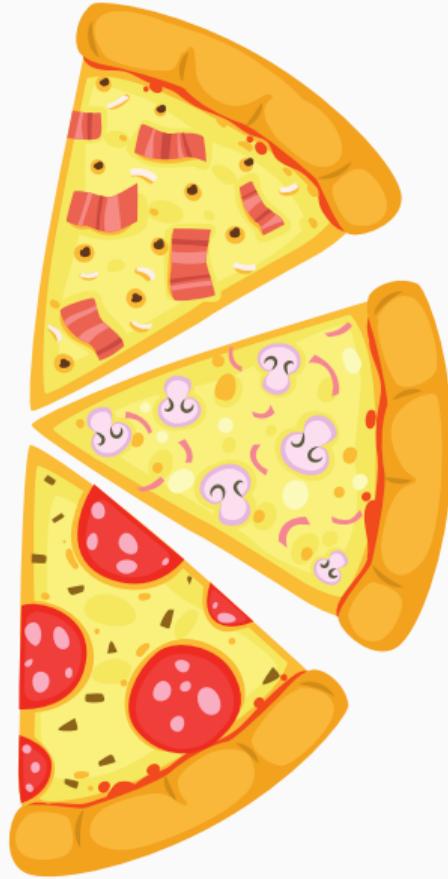


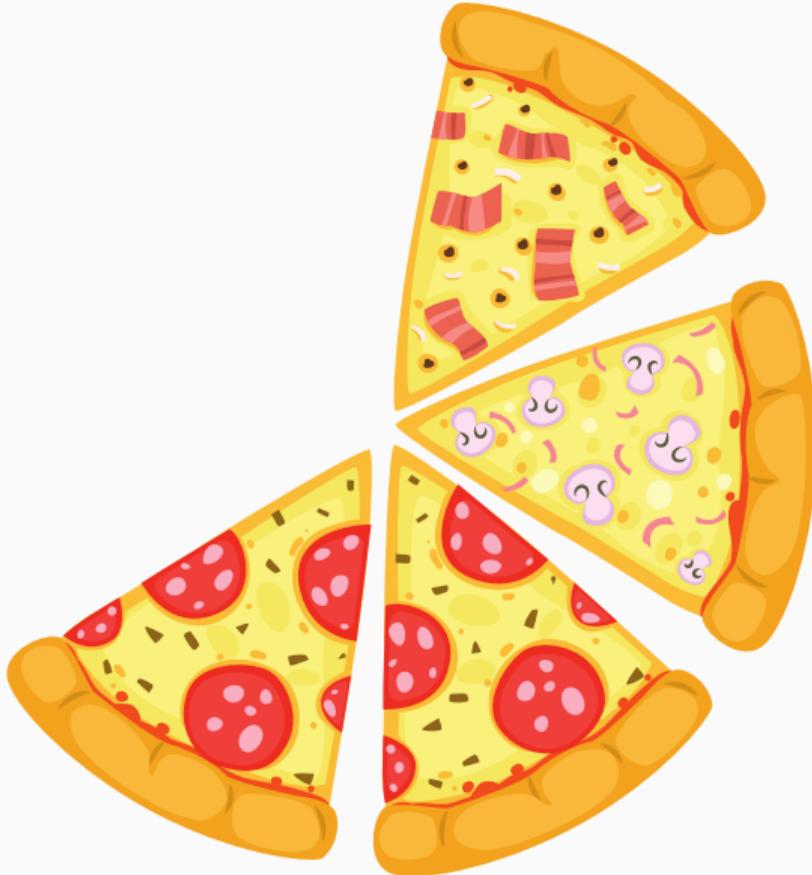
PIZZA

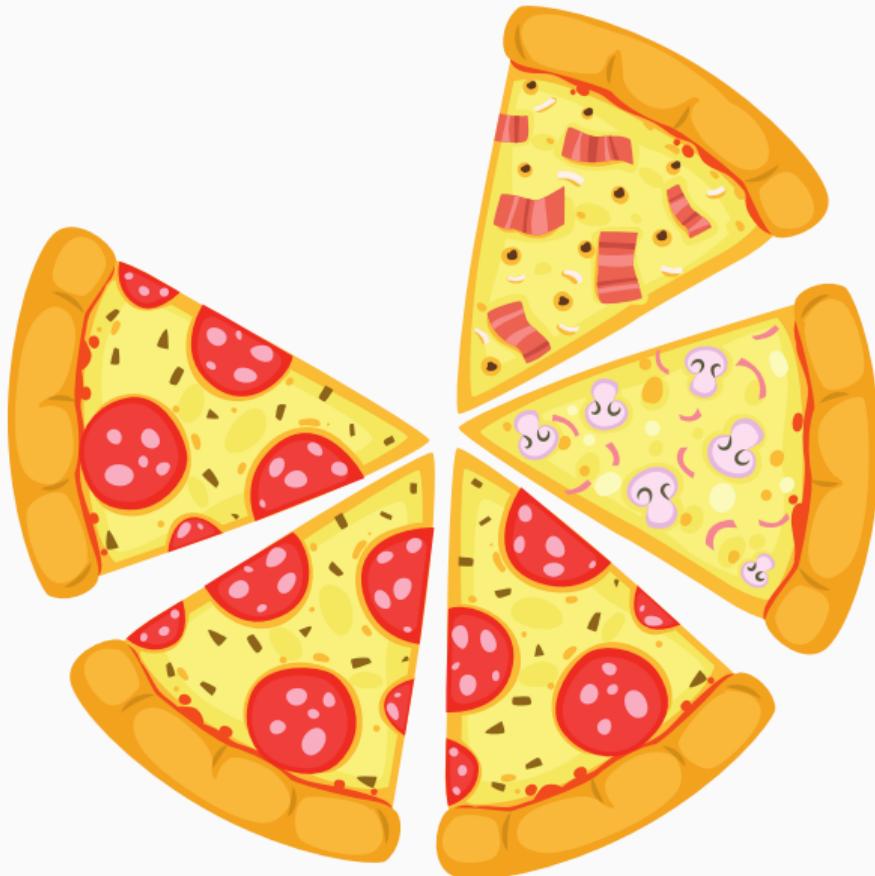
SPECIAL RECIPES















›A table for 6 please‹



Speculative Cooking





A table for 6 please





PIZZA

SPECIAL RECIPES



PIZZA

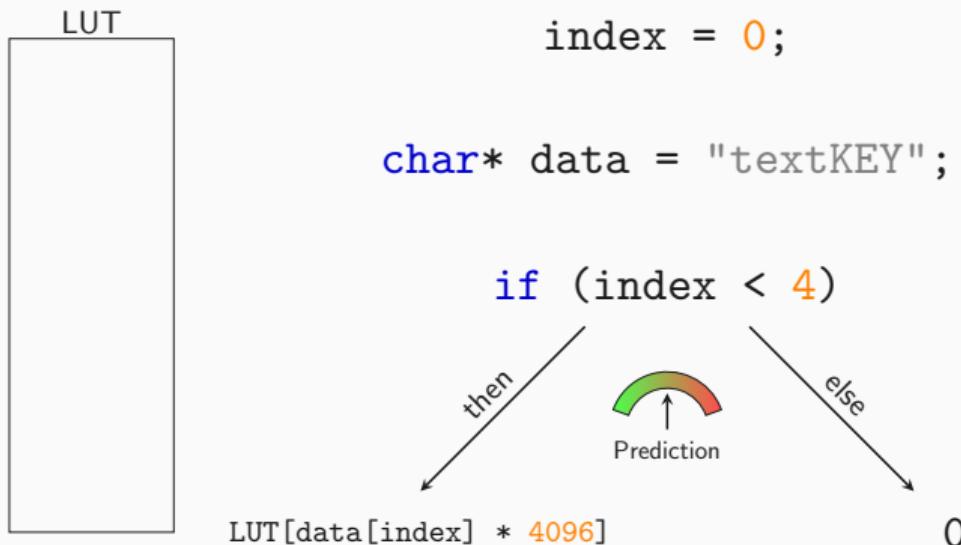
SPECIAL RECIPES

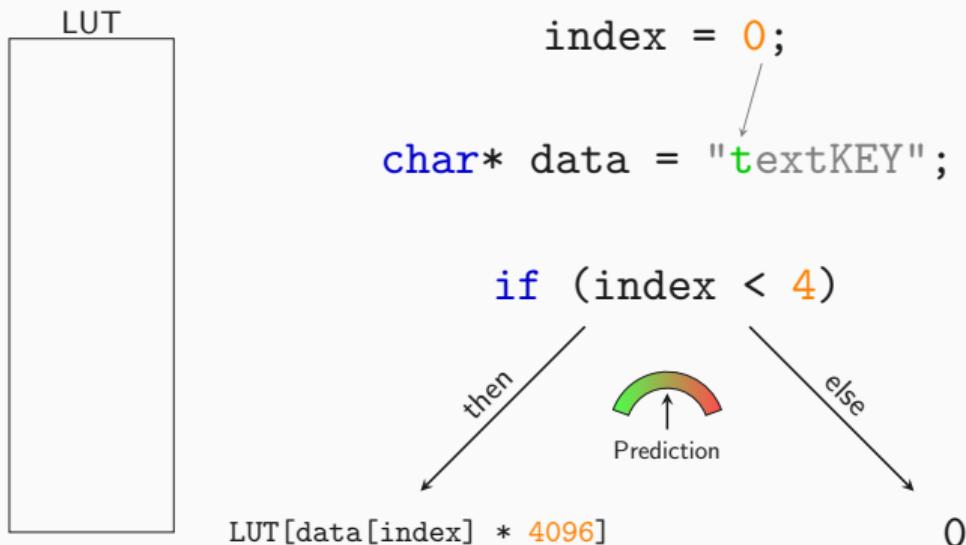
PIZZA

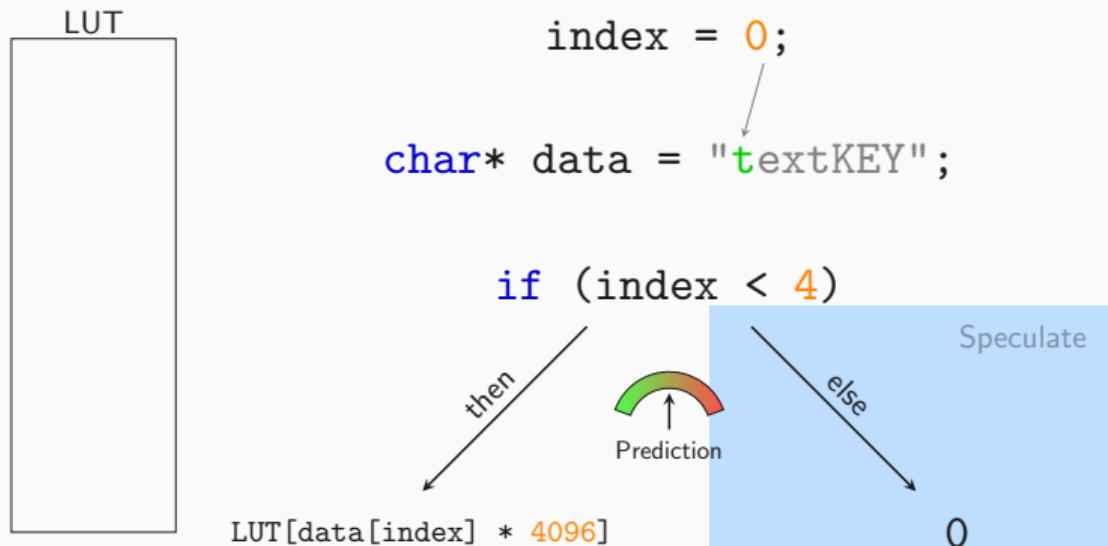


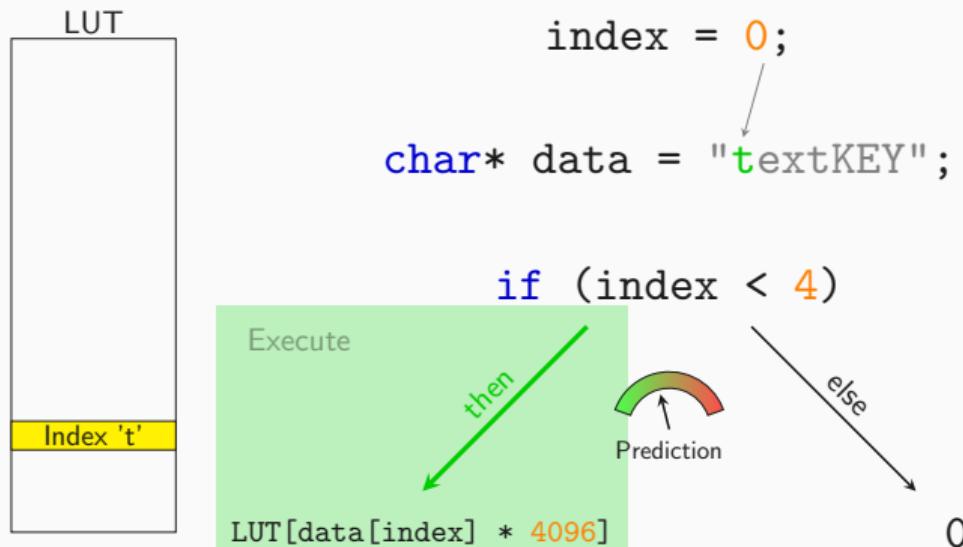


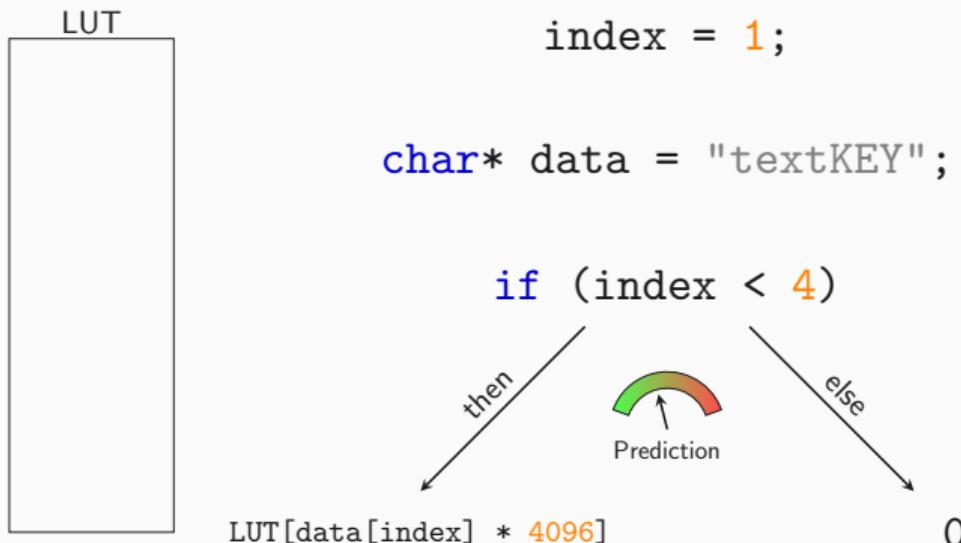


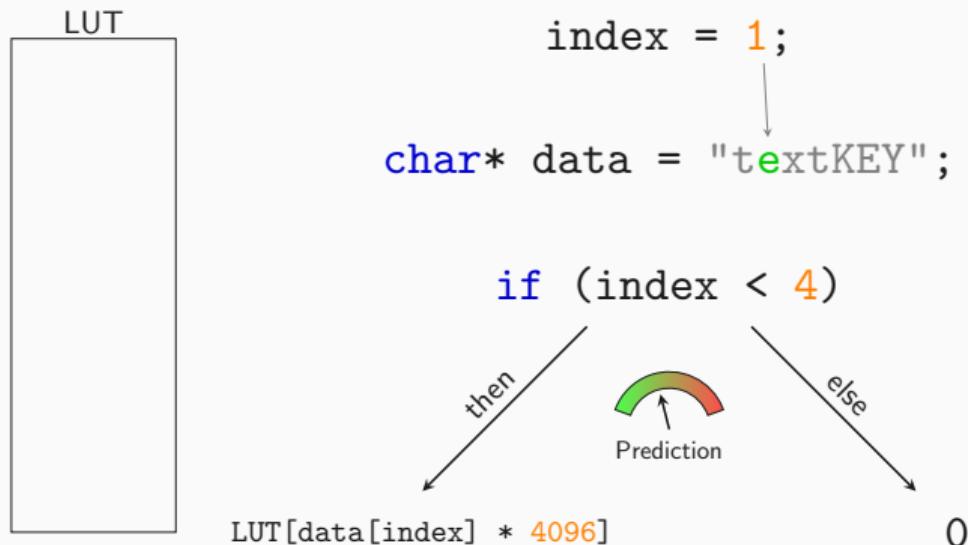


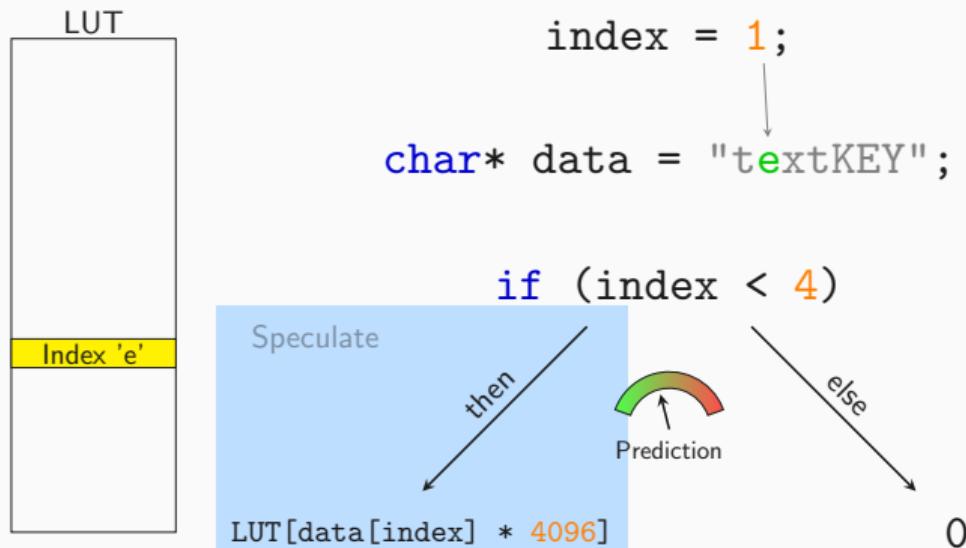


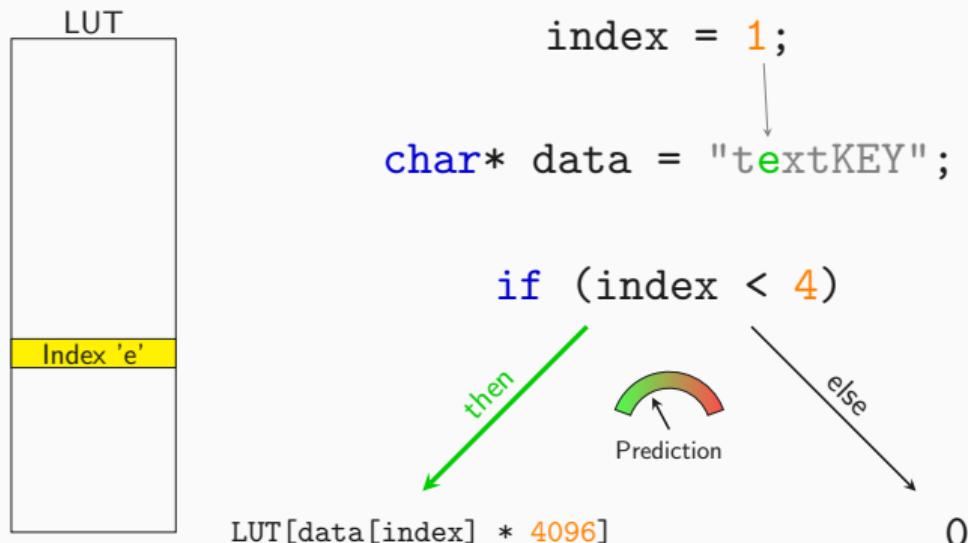


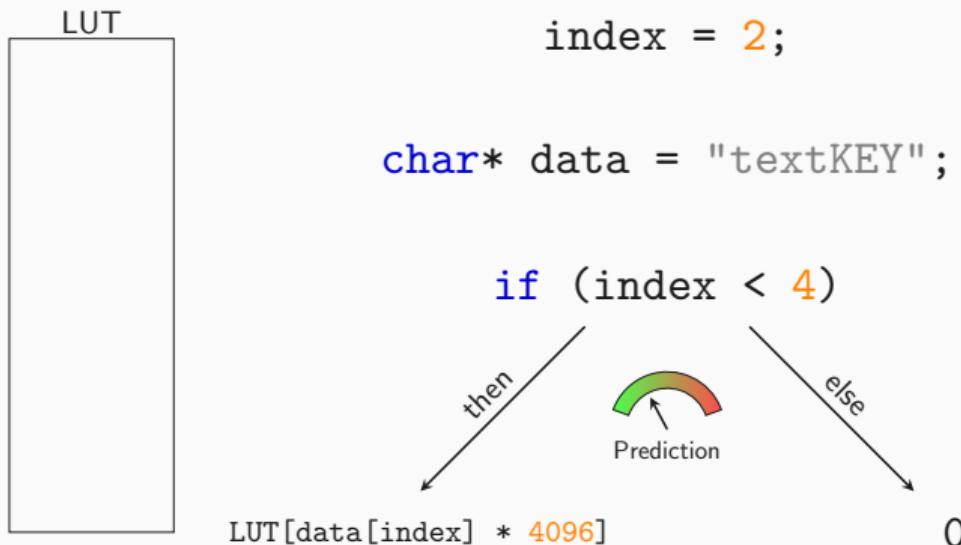


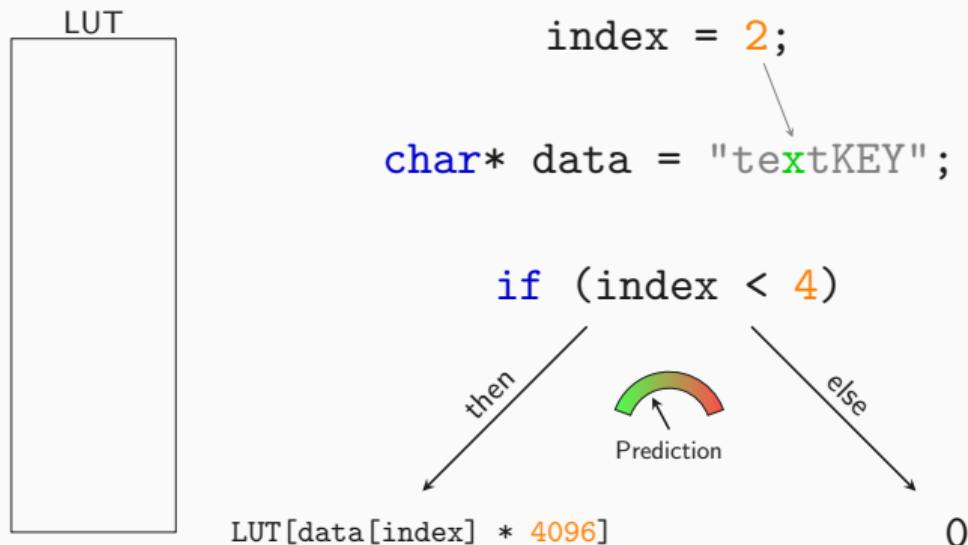


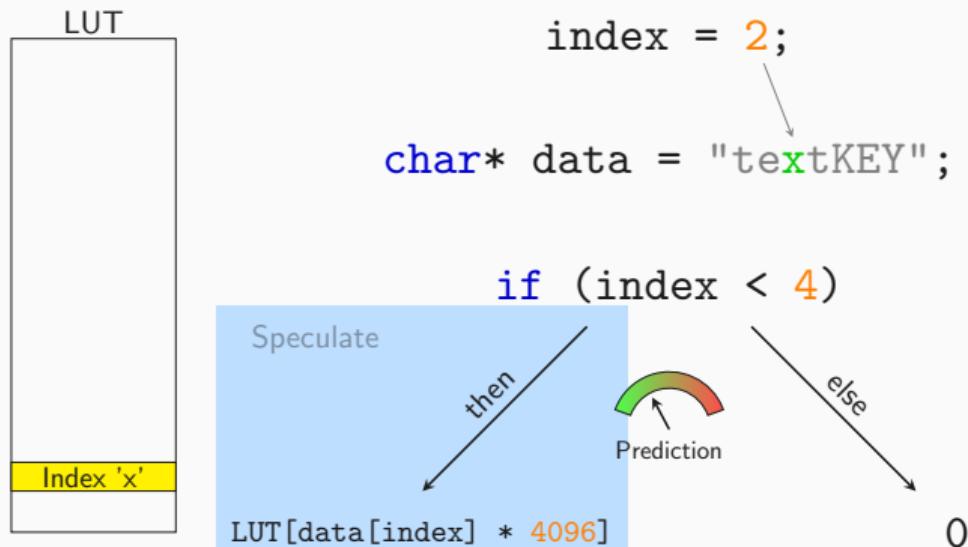


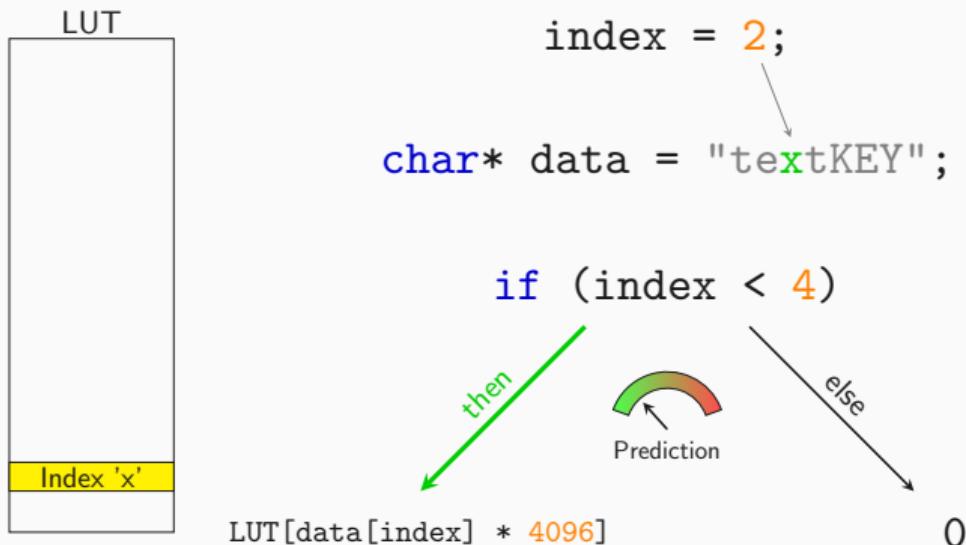


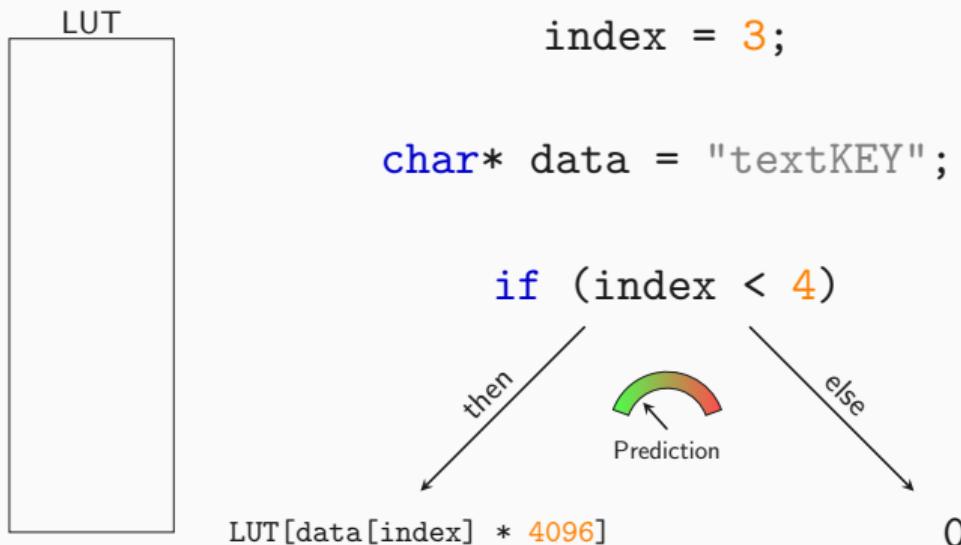


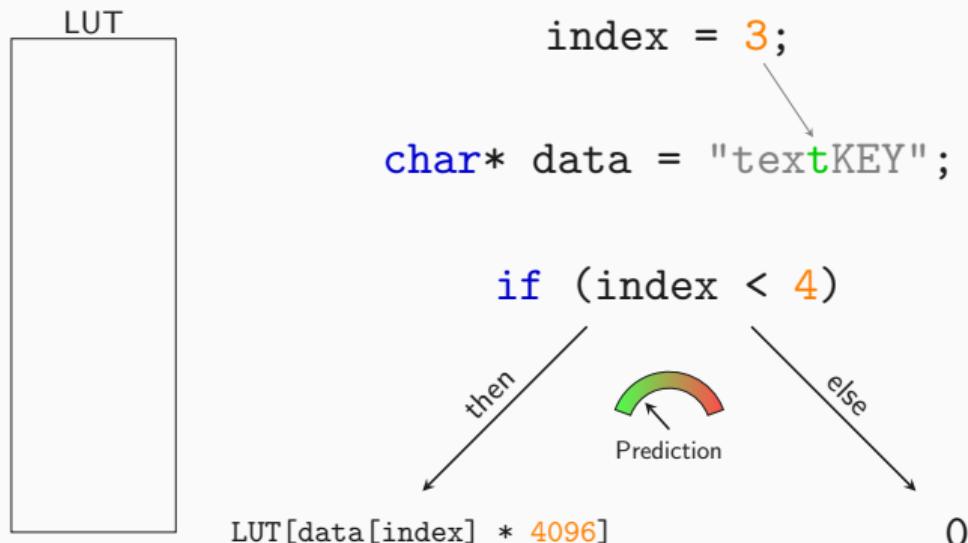


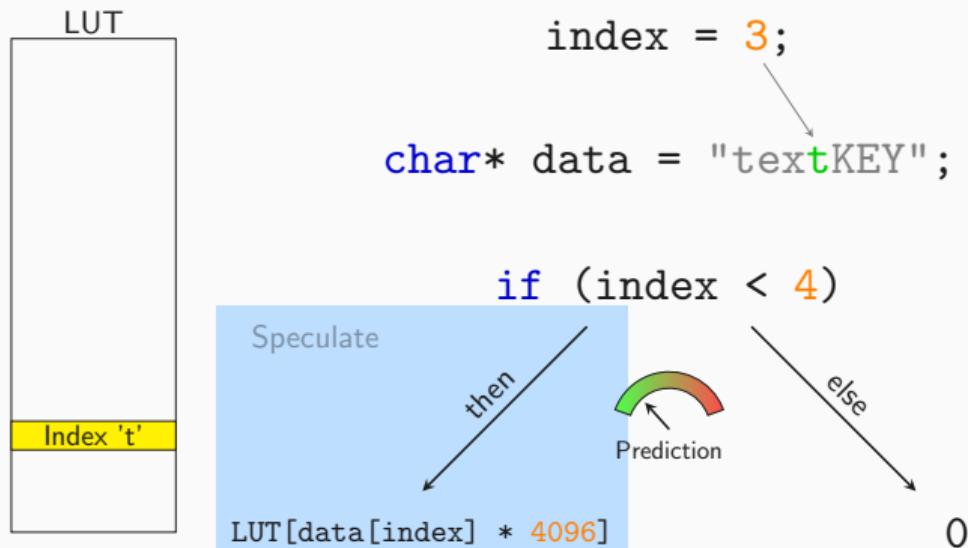


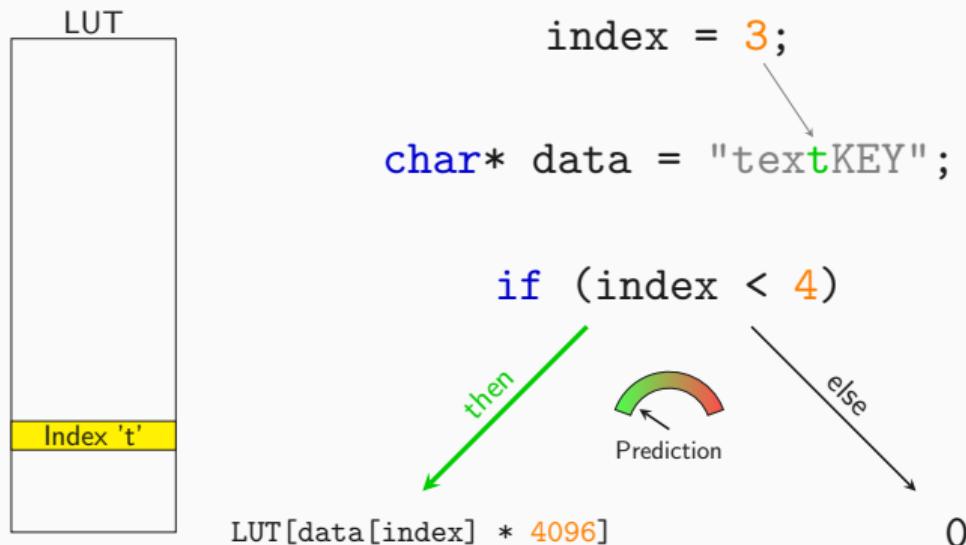


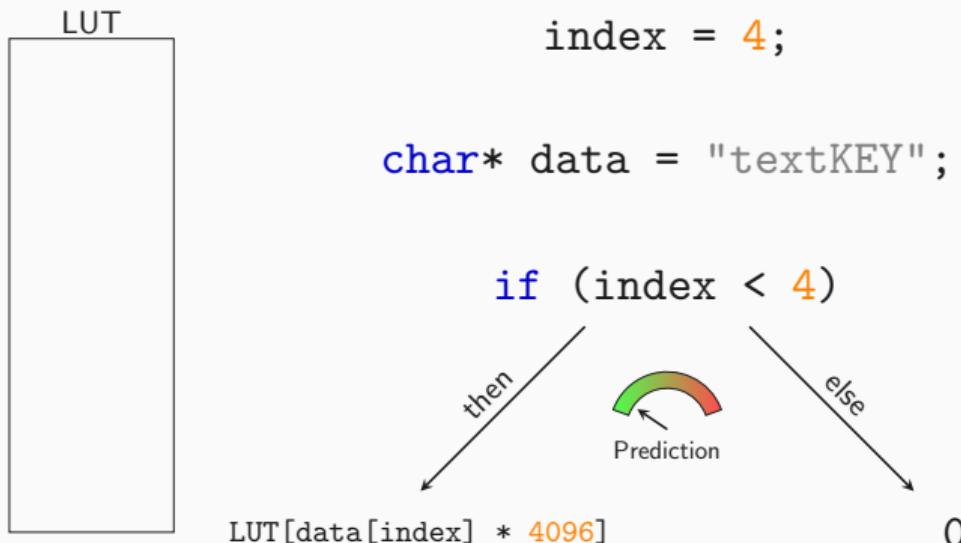


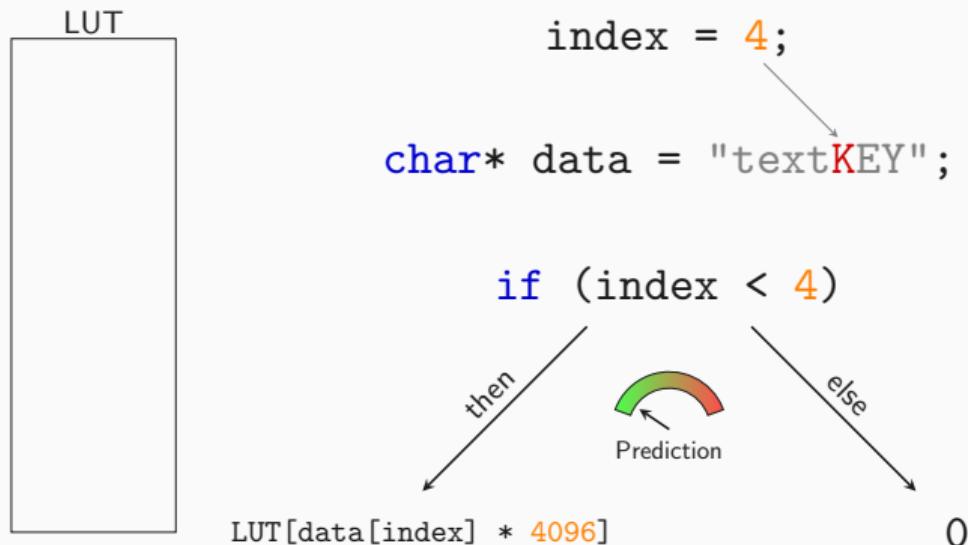


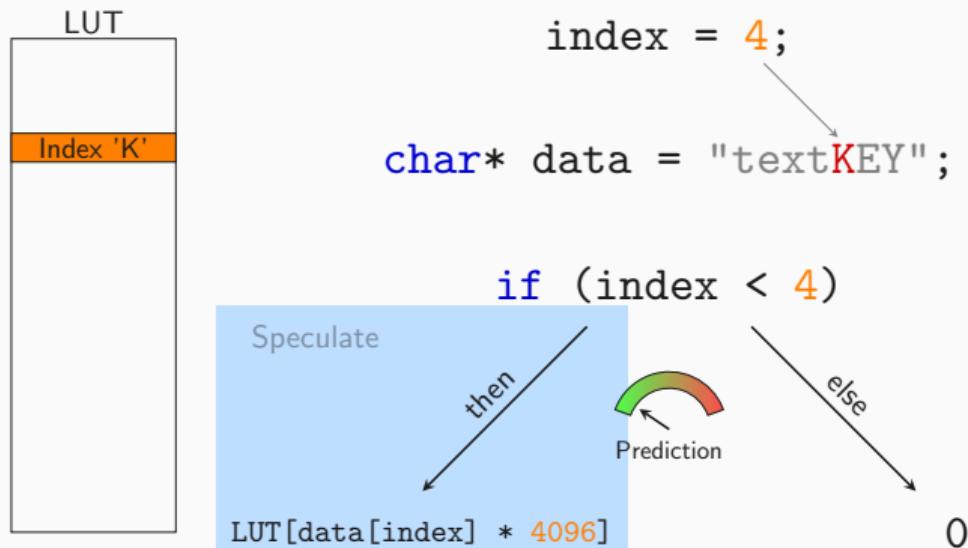


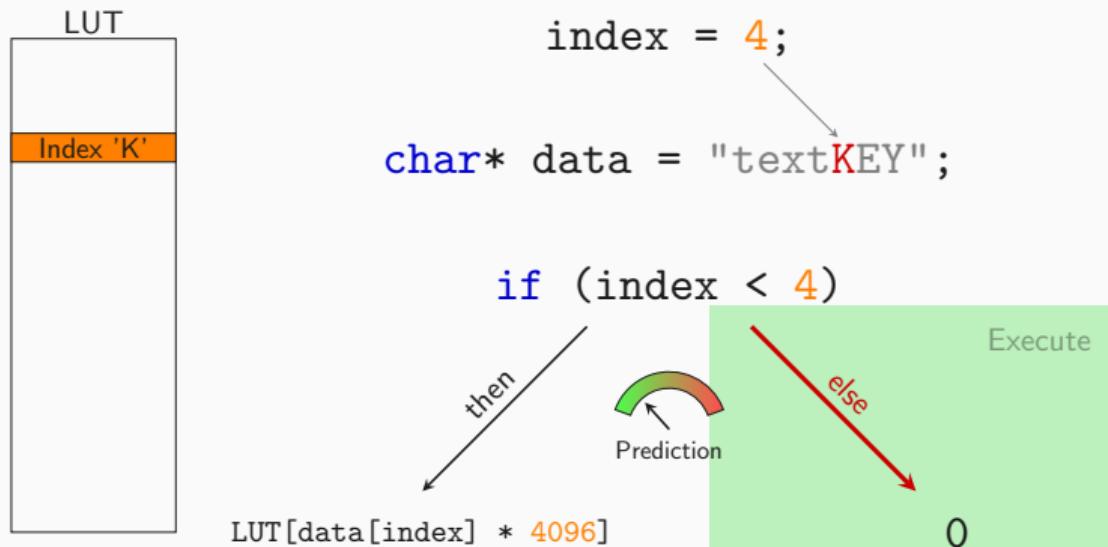


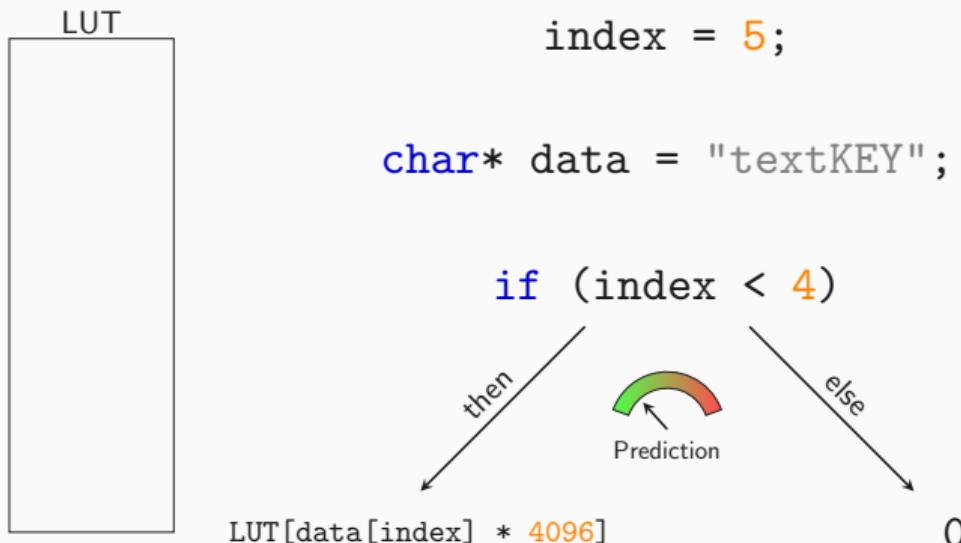


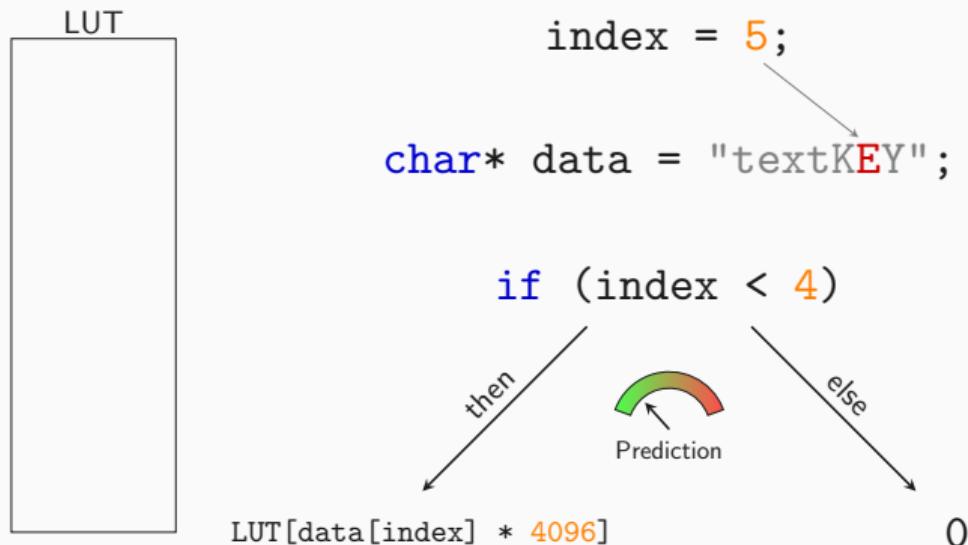


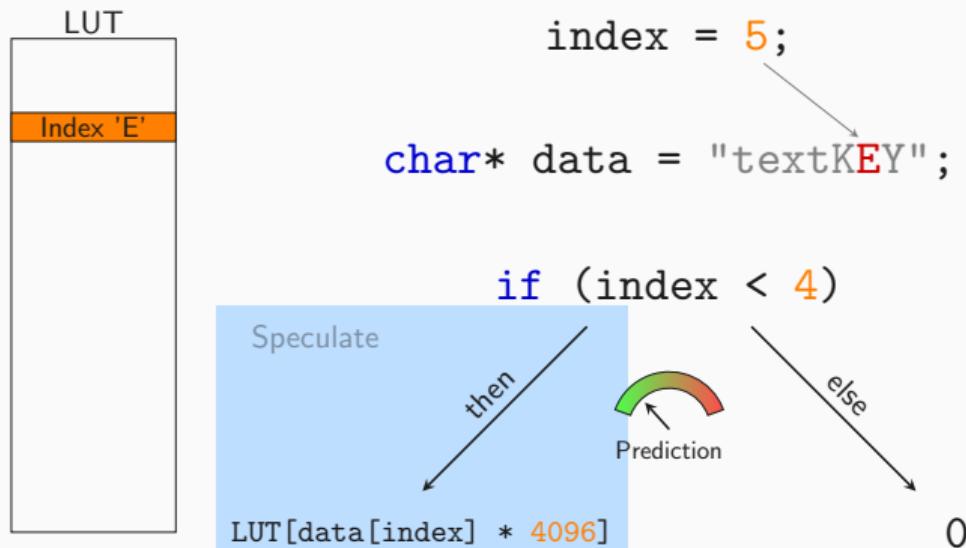


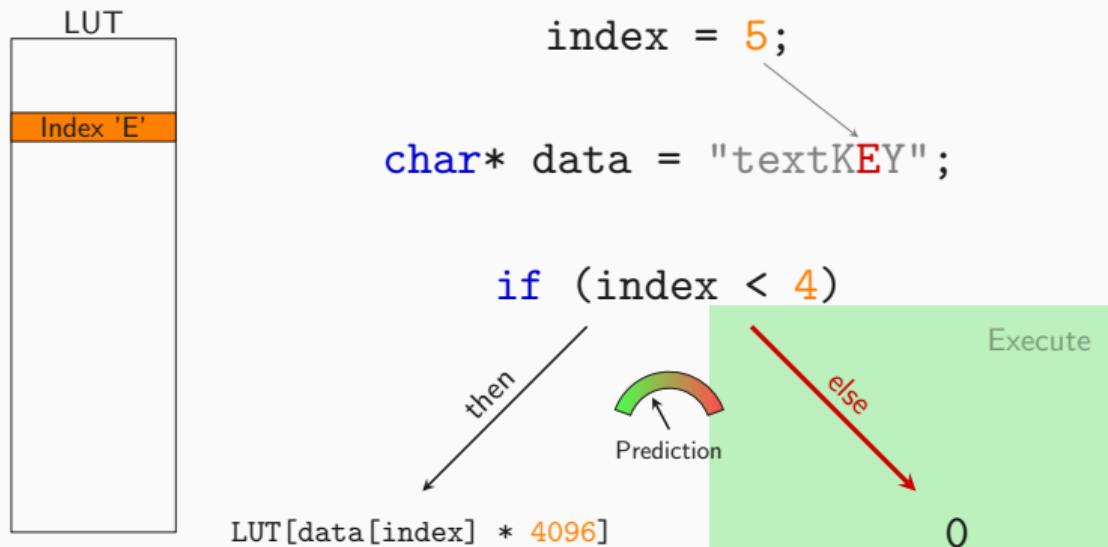


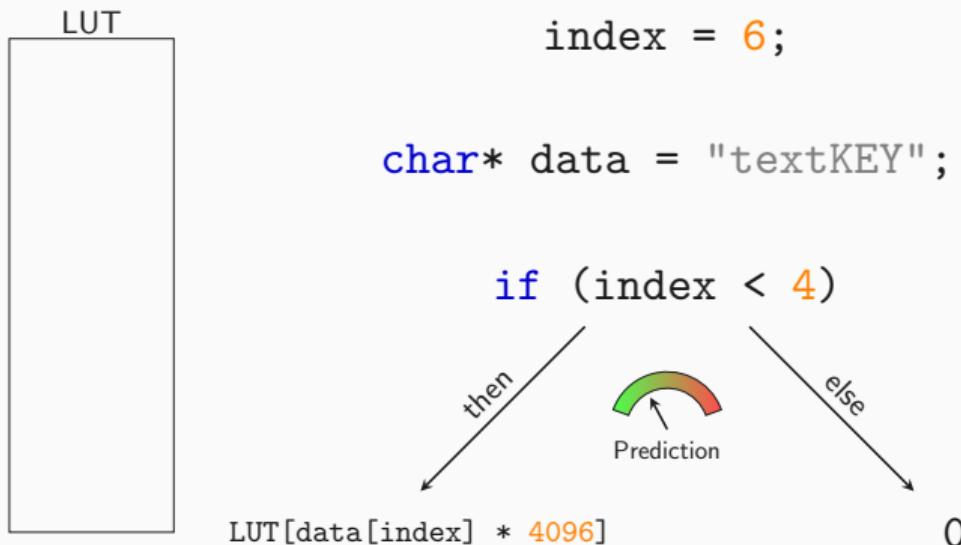


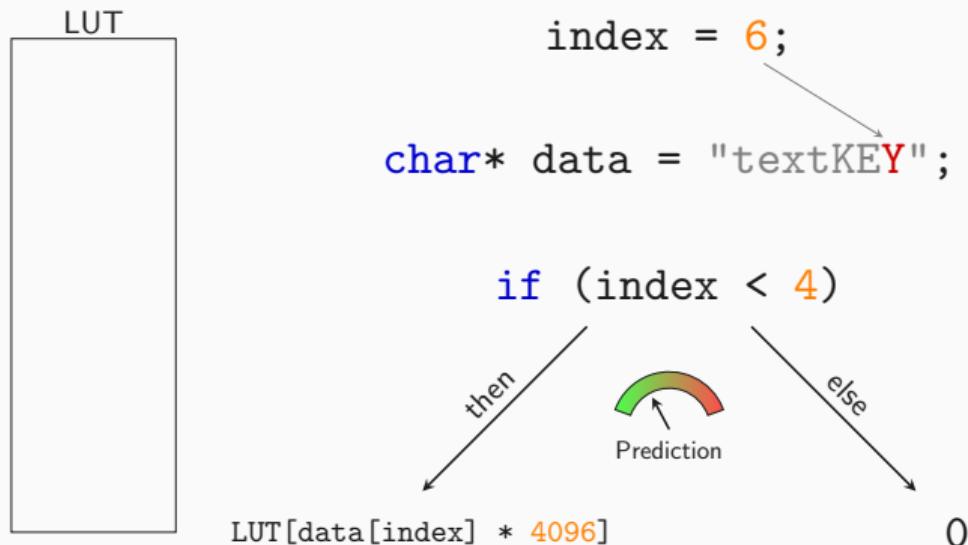


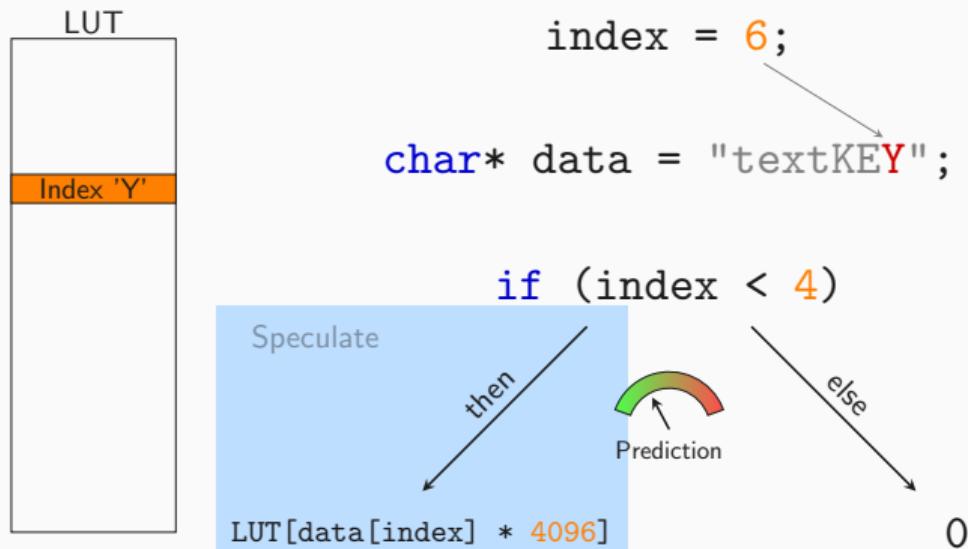


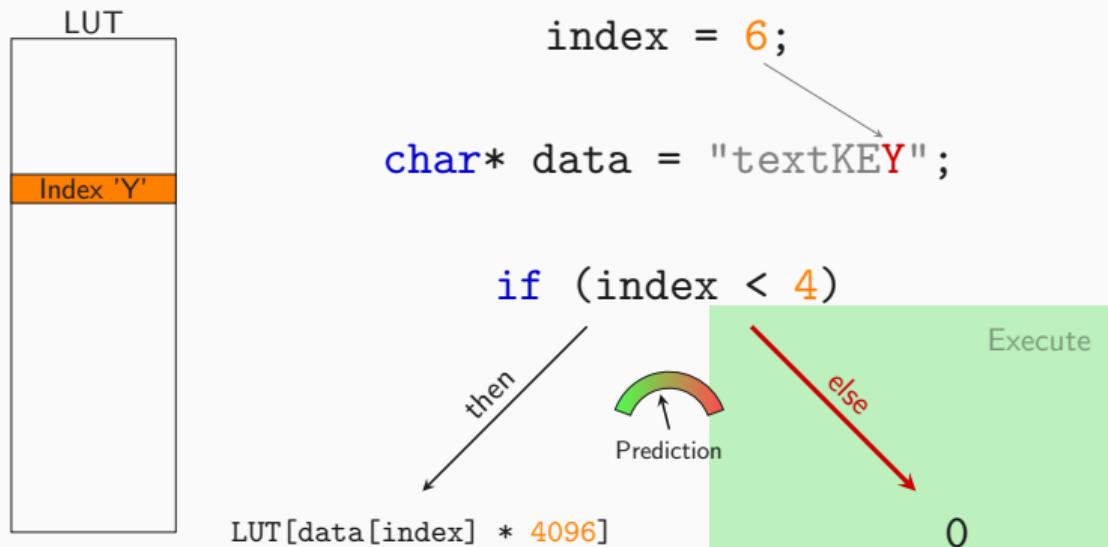




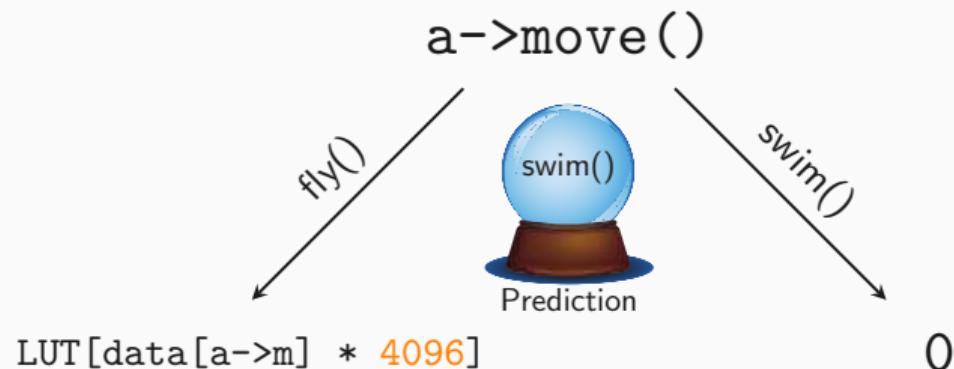




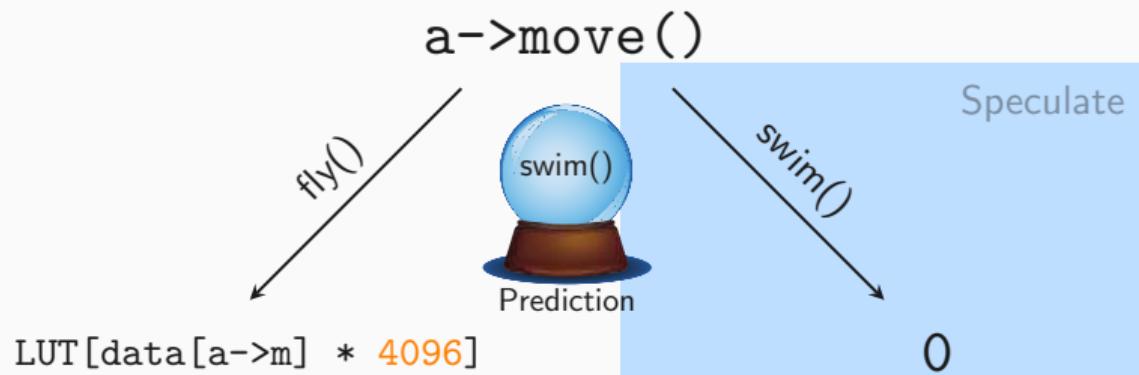




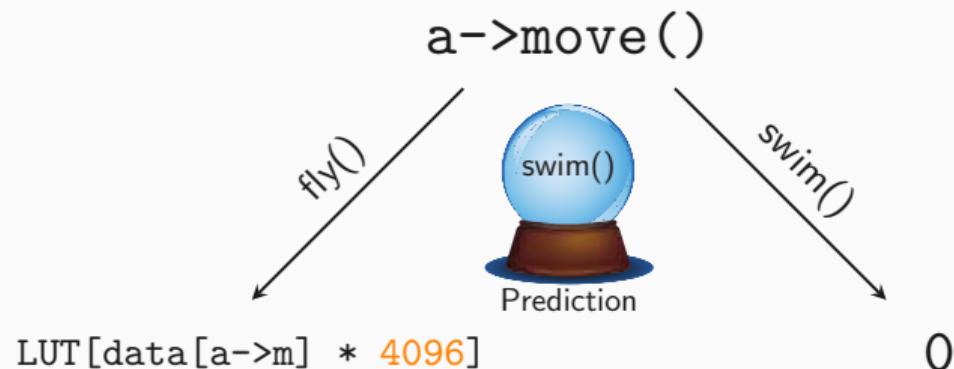
```
Animal* a = bird;
```



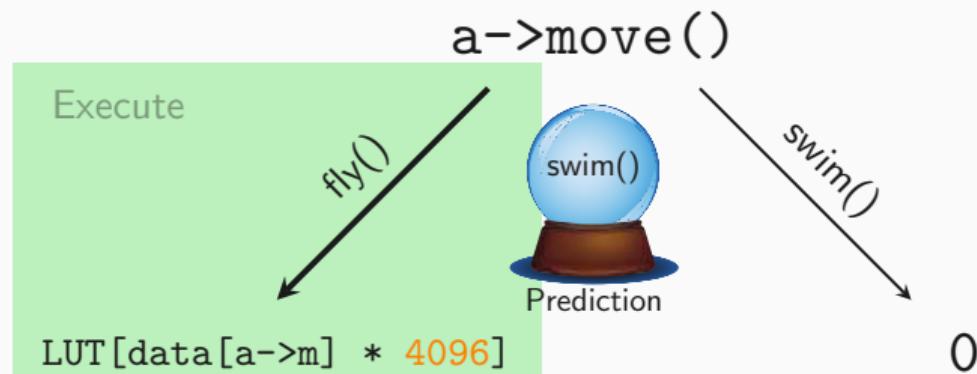
```
Animal* a = bird;
```



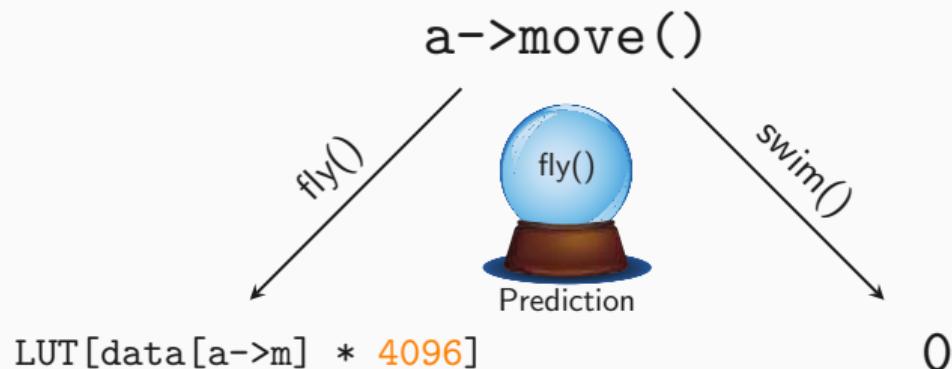
```
Animal* a = bird;
```



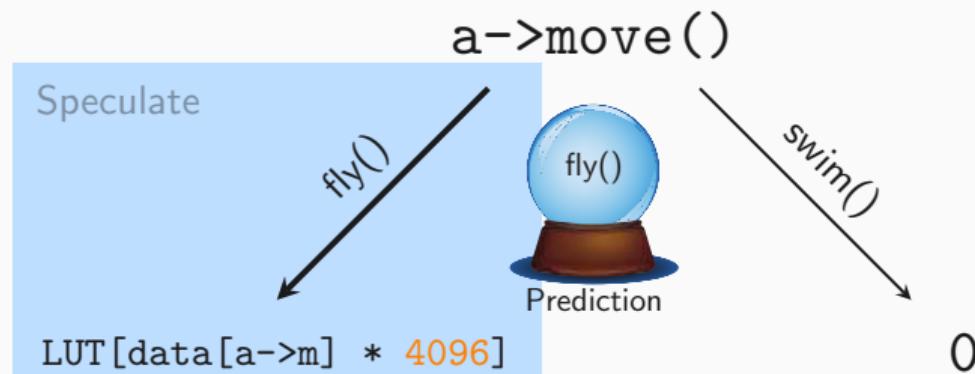
```
Animal* a = bird;
```



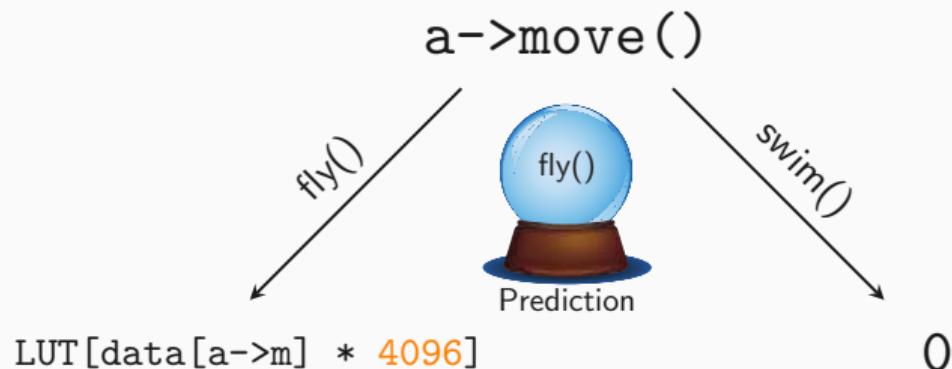
```
Animal* a = bird;
```



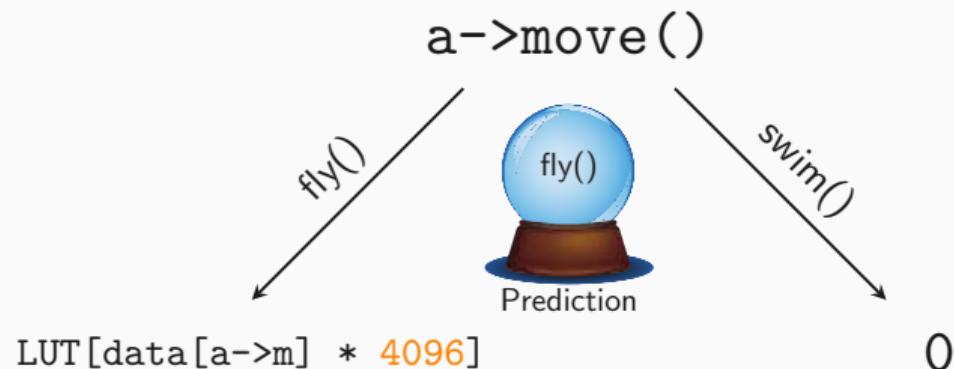
```
Animal* a = bird;
```



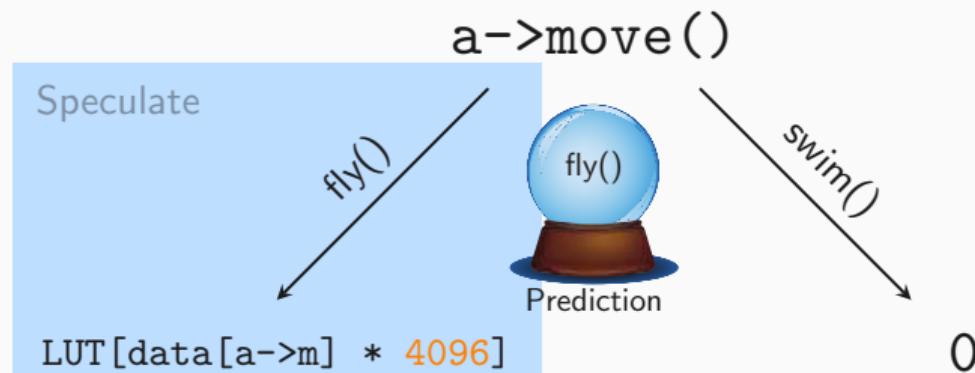
```
Animal* a = bird;
```



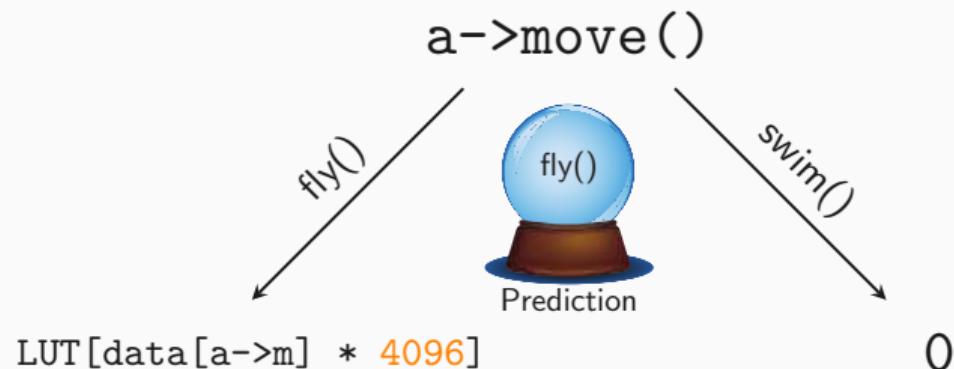
```
Animal* a = fish;
```



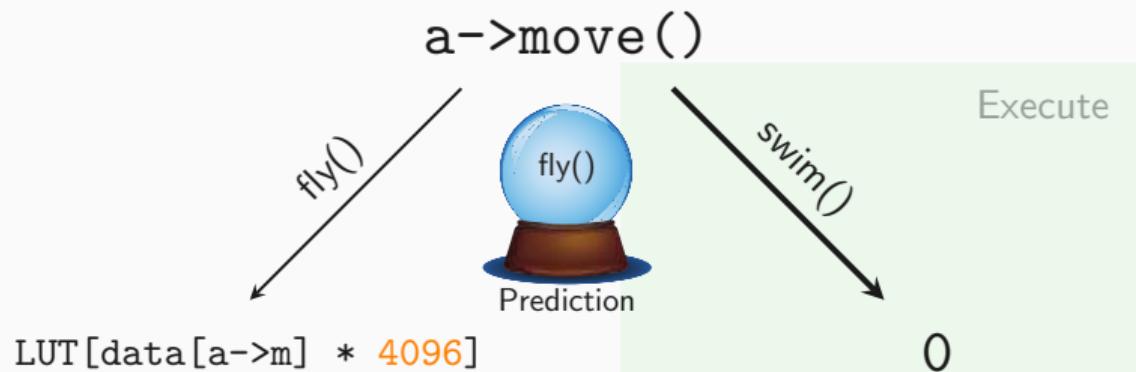
```
Animal* a = fish;
```



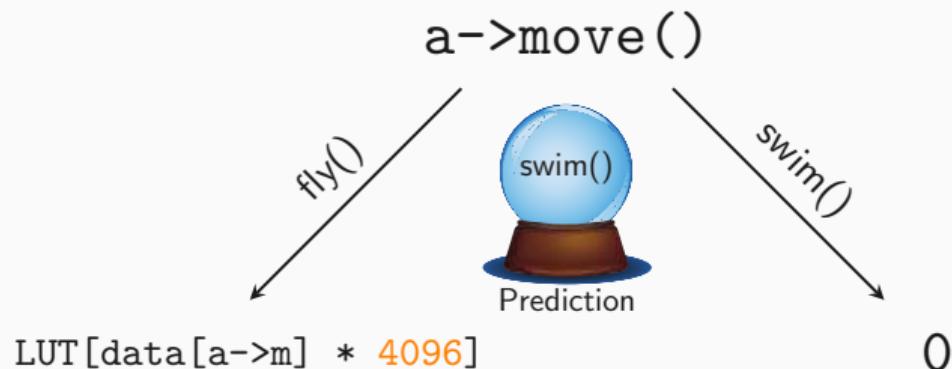
```
Animal* a = fish;
```

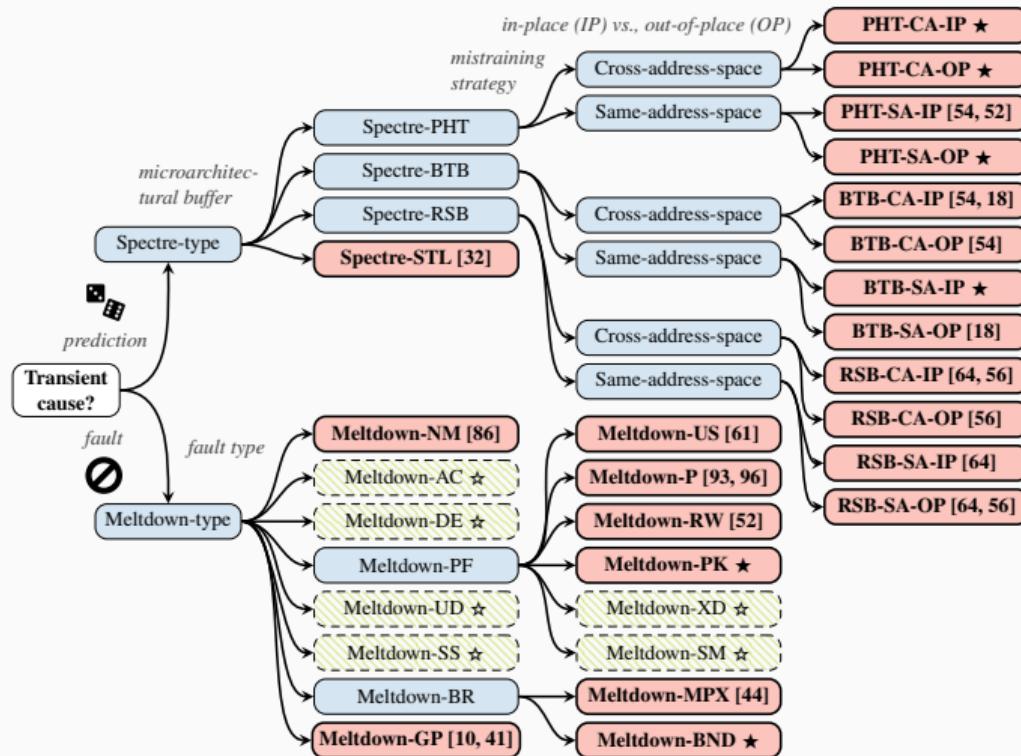


```
Animal* a = fish;
```



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





Mitigations?





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.

	Defense	InvisISpec	SafeSpec	DAM/G	RSB Stuffing	Retpoline	Poison Value	Index Masking	Site Isolation	SLH	YSNB	IBRS	STPB	IBPB	Serialization	Taint Tracking	Slosh Reduction	SSBD/SSBB
Attack \ Defense																		
Intel	Spectre-PHT	□	□	□	◊	◊	●	○	●	○	◊	◊	●	■	●	□	◊	
	Spectre-BTB	□	□	□	◊	●	◊	◊	○	◊	◊	●	●	●	◊	■	●	◊
	Spectre-RSB	□	□	□	◊	●	◊	◊	◊	○	◊	◊	◊	◊	◊	■	●	◊
	Spectre-STL	□	□	□	◊	◊	◊	◊	●	◊	◊	◊	◊	◊	◊	■	●	●
ARM	Spectre-PHT	□	□	□	◊	◊	●	○	●	○	◊	◊	◊	●	■	●	□	◊
	Spectre-BTB	□	□	□	◊	●	◊	◊	○	◊	◊	◊	◊	◊	◊	■	●	◊
	Spectre-RSB	□	□	□	◊	◊	◊	◊	●	◊	◊	◊	◊	◊	◊	■	●	◊
	Spectre-STL	□	□	□	◊	◊	◊	◊	●	◊	◊	◊	◊	◊	◊	■	●	●
AMD	Spectre-PHT	□	□	□	◊	◊	●	○	●	○	◊	◊	◊	●	■	●	□	◊
	Spectre-BTB	□	□	□	◊	●	◊	◊	○	◊	◊	■	■	■	◊	■	●	◊
	Spectre-RSB	□	□	□	◊	◊	◊	◊	●	◊	◊	◊	◊	◊	◊	■	●	◊
	Spectre-STL	□	□	□	◊	◊	◊	◊	●	◊	◊	◊	◊	◊	◊	■	●	●

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

Table 2: Reported performance impacts of countermeasures

Defense \ Impact	Performance Loss	Benchmark
Defense		
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%	SYStmark®2014 SE & SPEC integer
KAISER/KPTI	0–2.6%	system call rates
L1TF mitigations	-3–31%	various SPEC

How to find the next big thing ;)

they become the target of one anti-masker's possessed creation, Arnaudine.

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

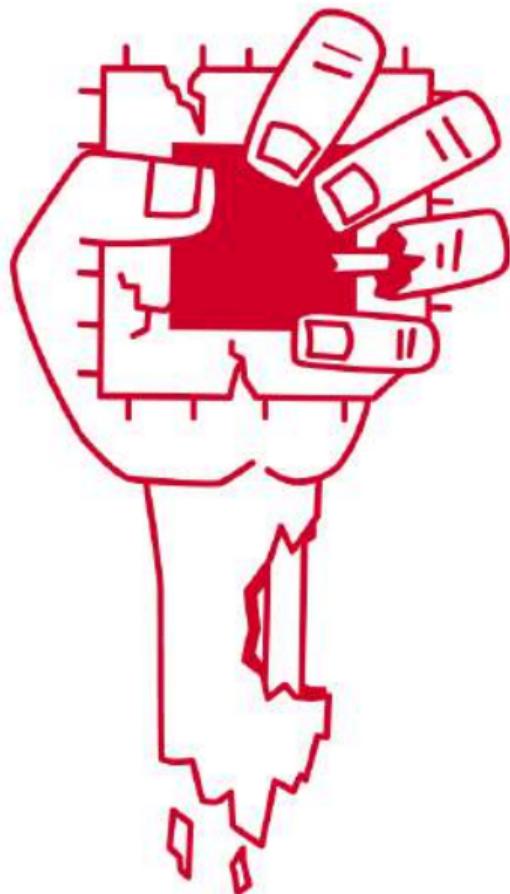
WOODY
HARRELSON JESSE
EISENBERG EMMA
STONE BRIGITTE
BRESLIN

ZOMBIELAND

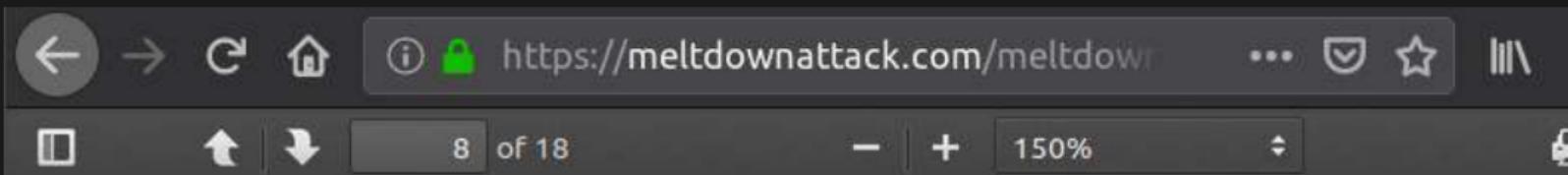
NUT UP OR SHUT UP



IN THEATERS OCTOBER 9
Follow us @Zombieland on Twitter



ZOMBIELOAD ATTACK



determine the kernel address stored at the memory locations, including the

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



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

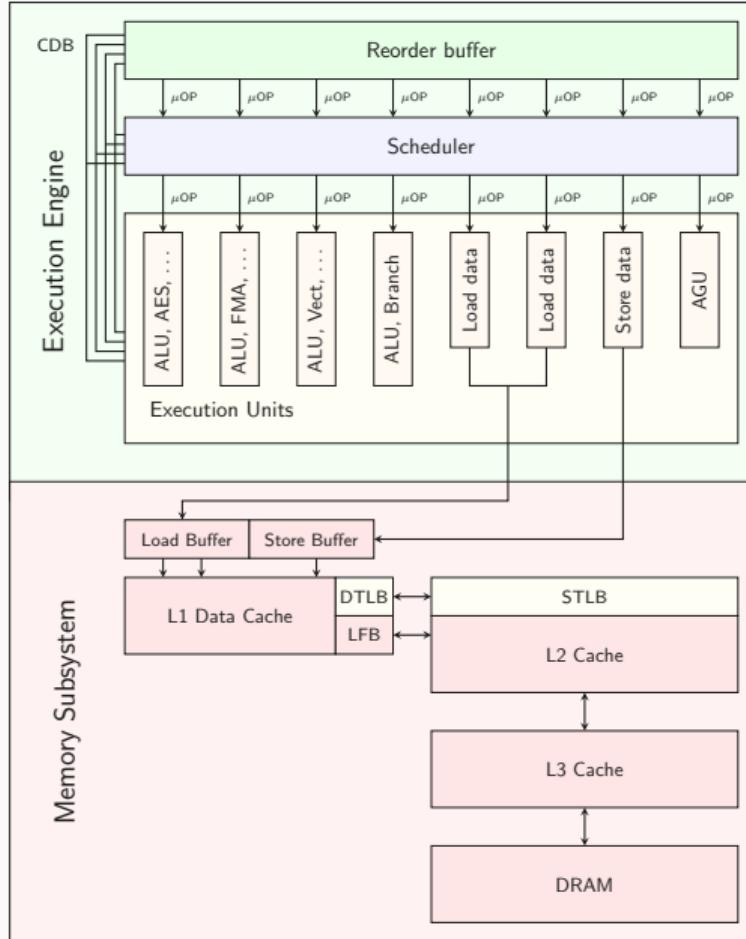
<https://www.techrepublic.com/.../toshiba-boot-error/> ▾ Diese Seite übersetzen

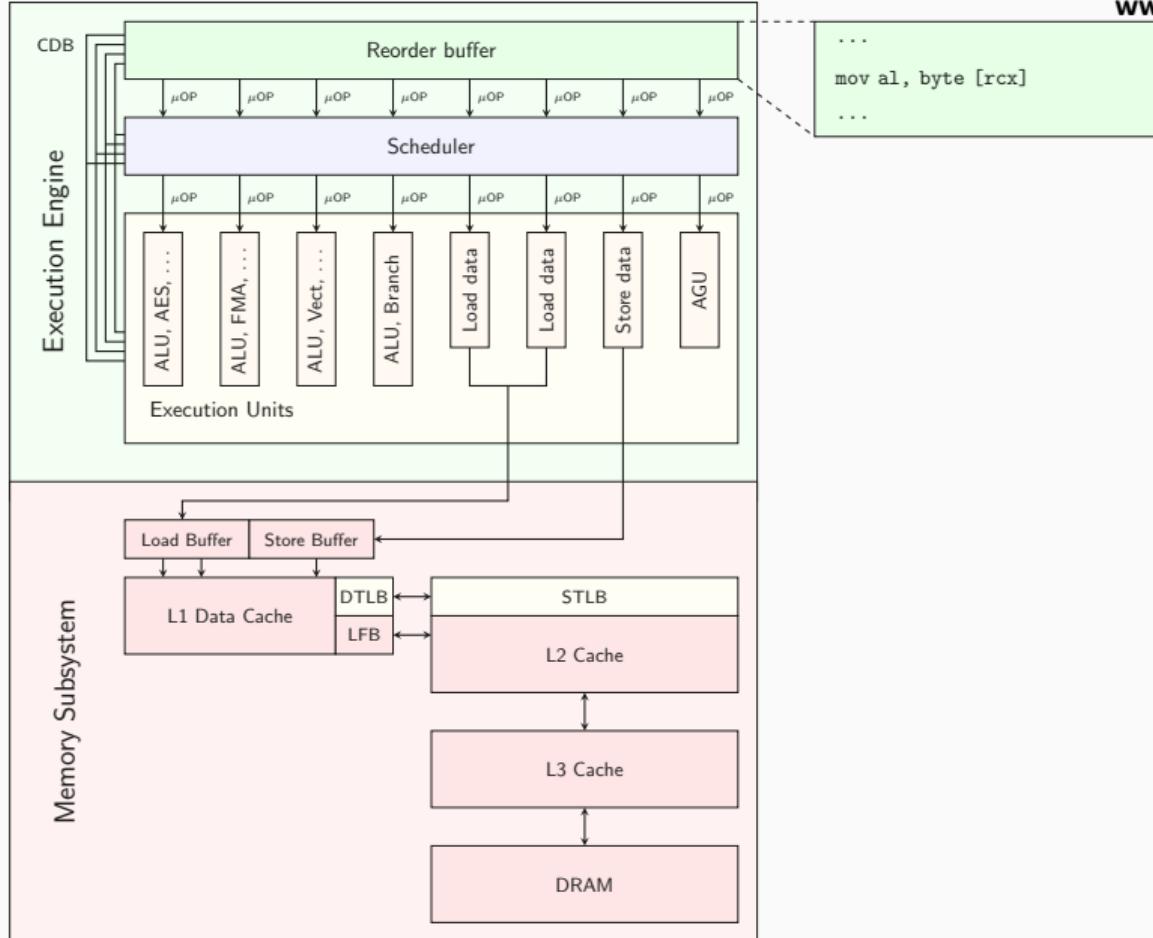
19.05.2007 - by CaptBilly1Eye · 12 years ago In reply to Toshiba Boot Error ... partition on the floppy disk, hard drive or a CD ROM to load the operating system. ... prior to this situation starting to occur, or if you find that the boot sequence already has the ... Leave the notebook plugged in and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

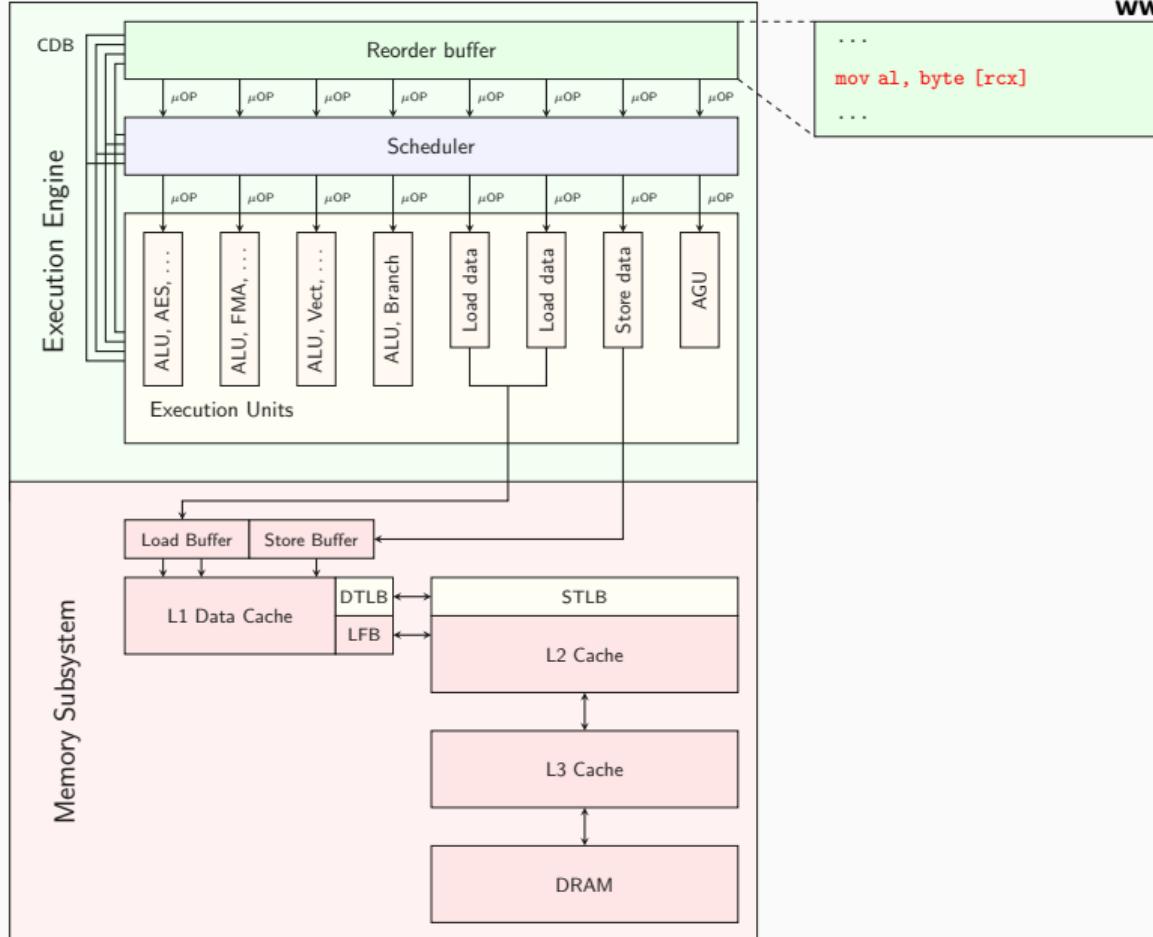
US5751983A - Out-of-order processor with a memory ...

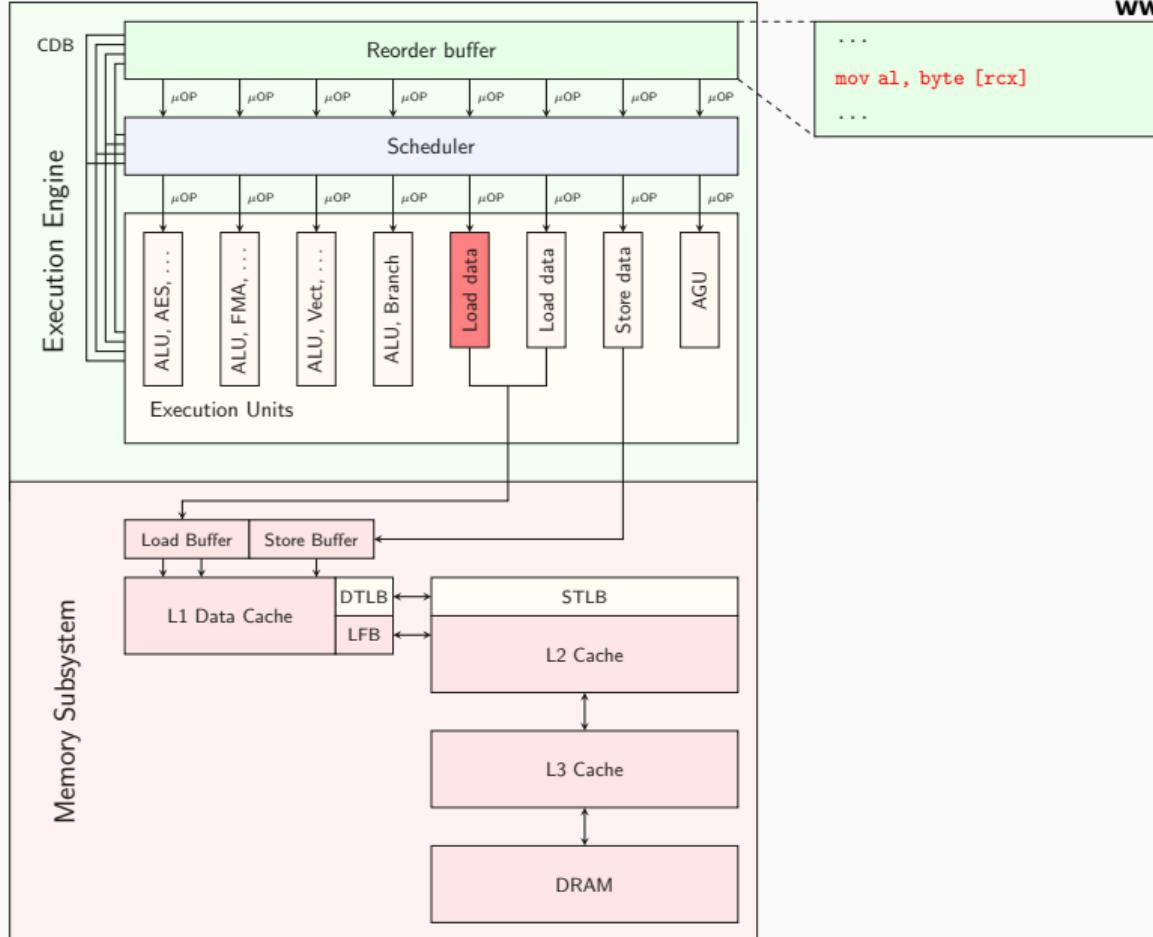
www.google.com/patents/US5751983 ▾ Diese Seite übersetzen

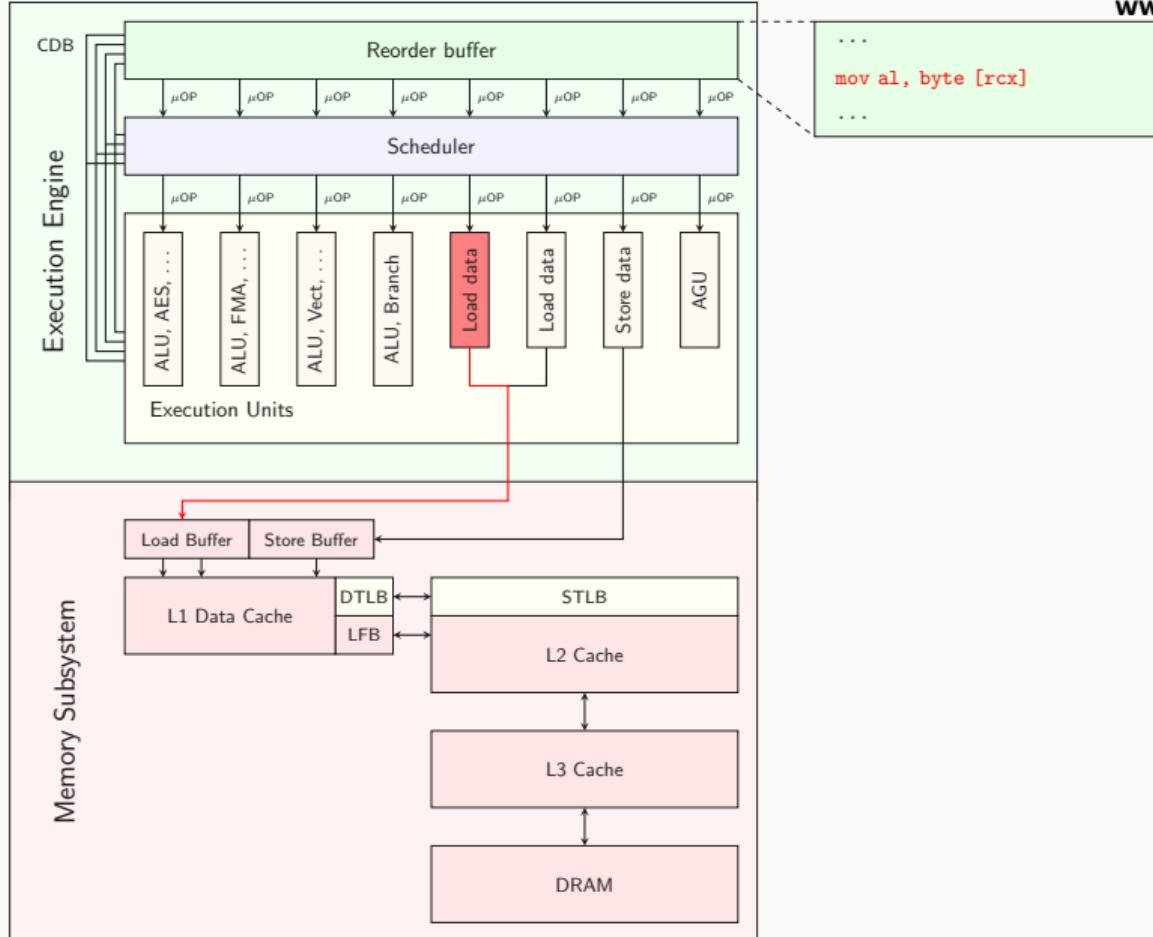
Application filed by Intel Corp ... Hence, a functional unit may often complete a first instruction (which logically precedes a second instruction in the If a fault occurs with respect to the LOAD operation, it is marked as valid and completed.

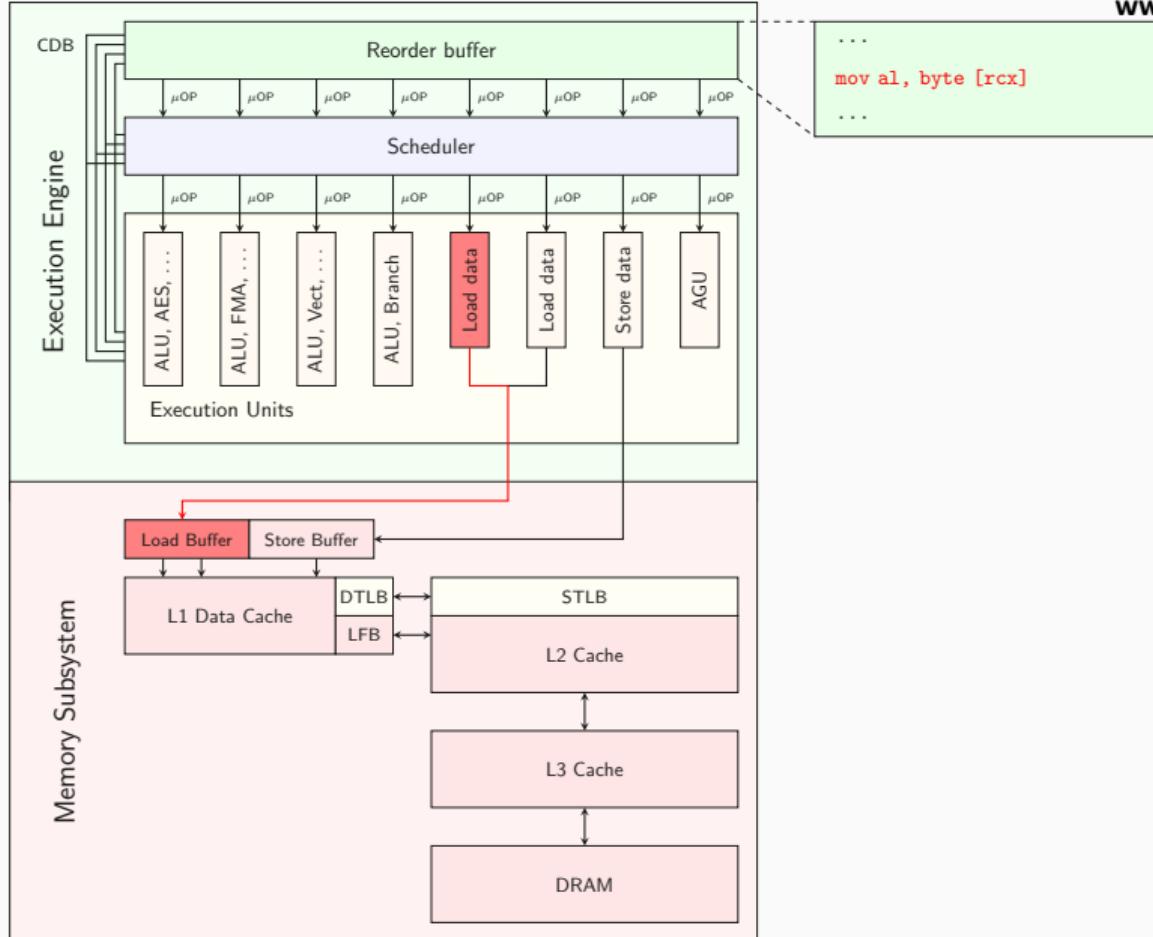


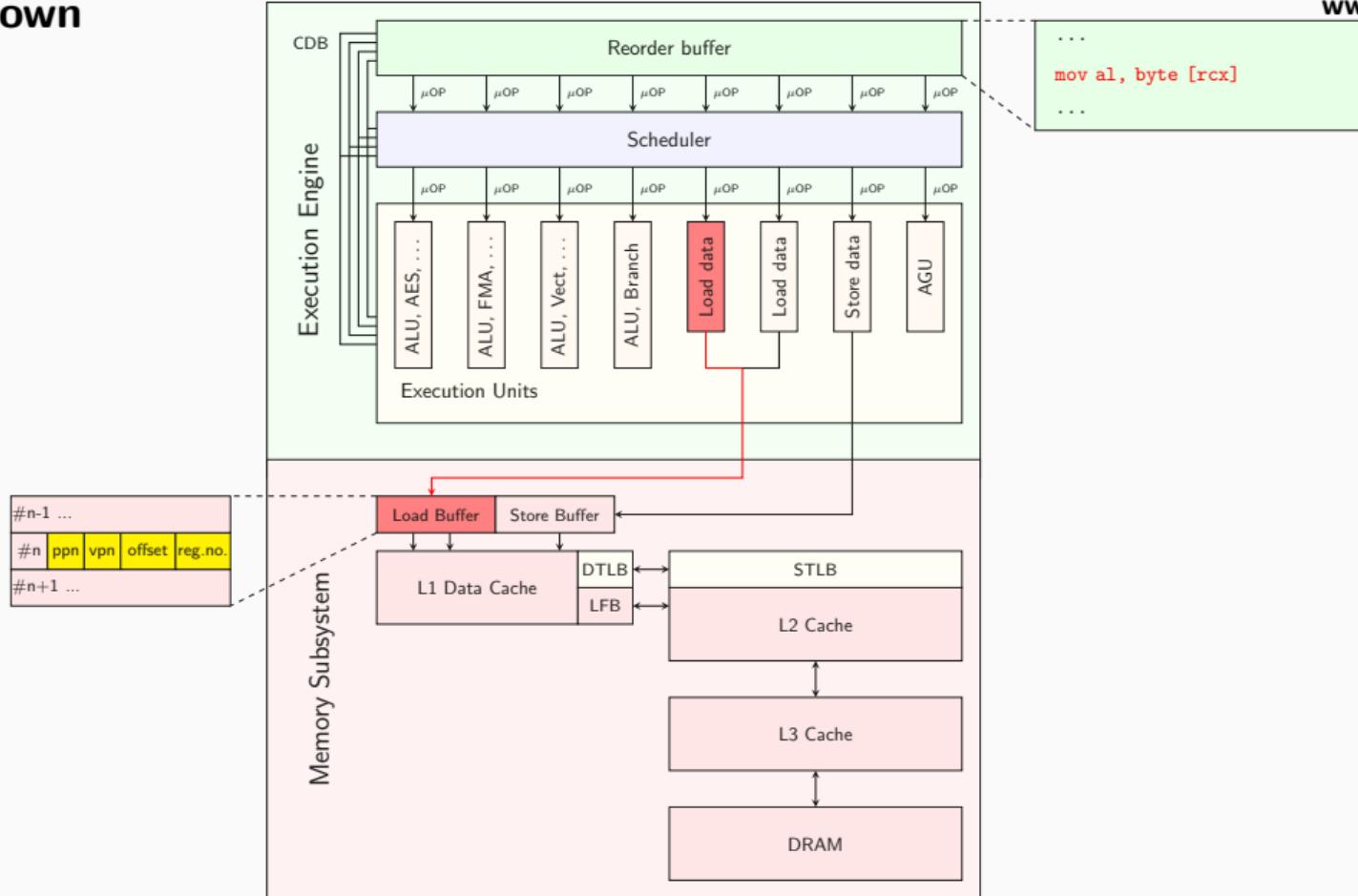


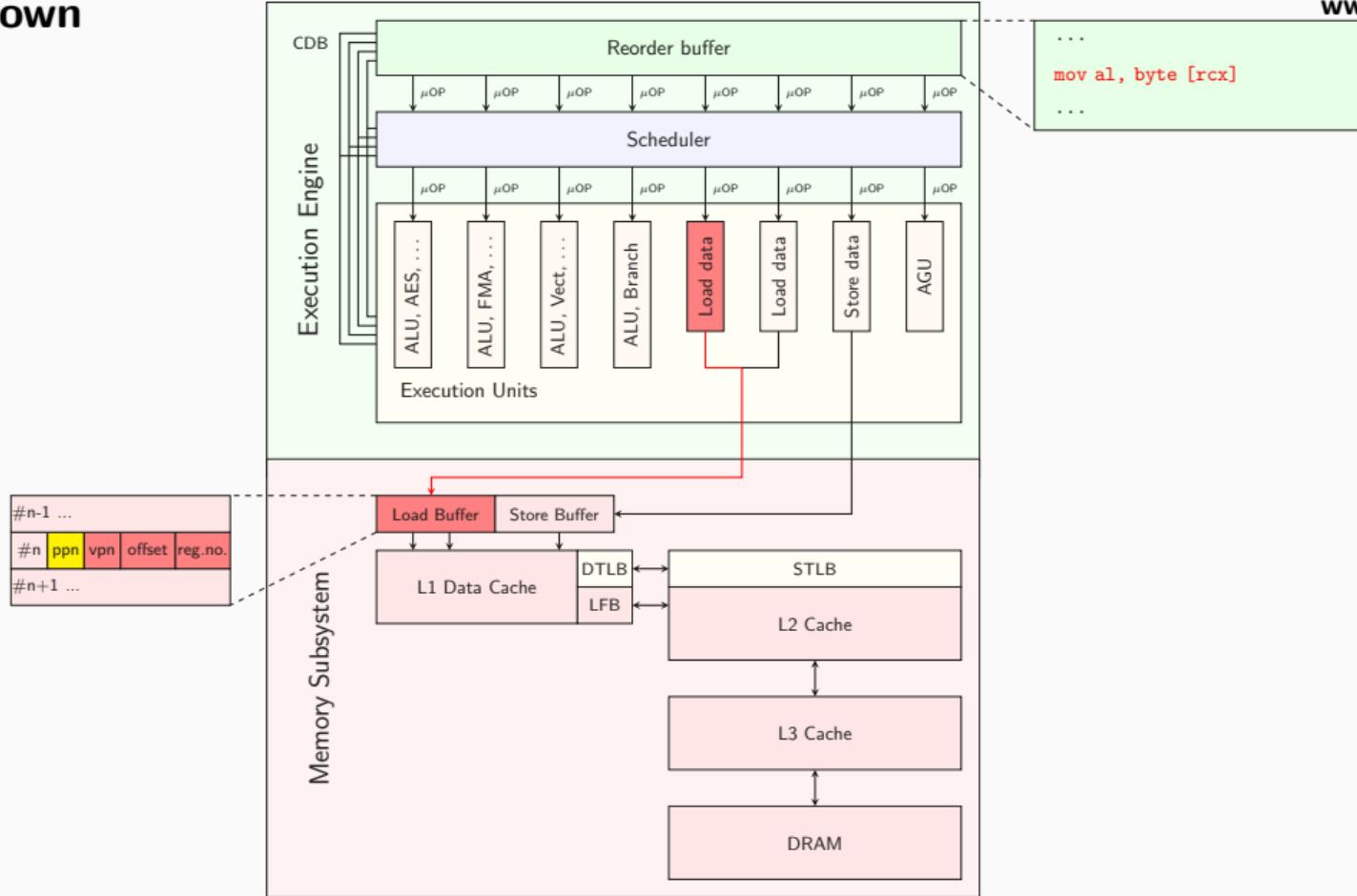


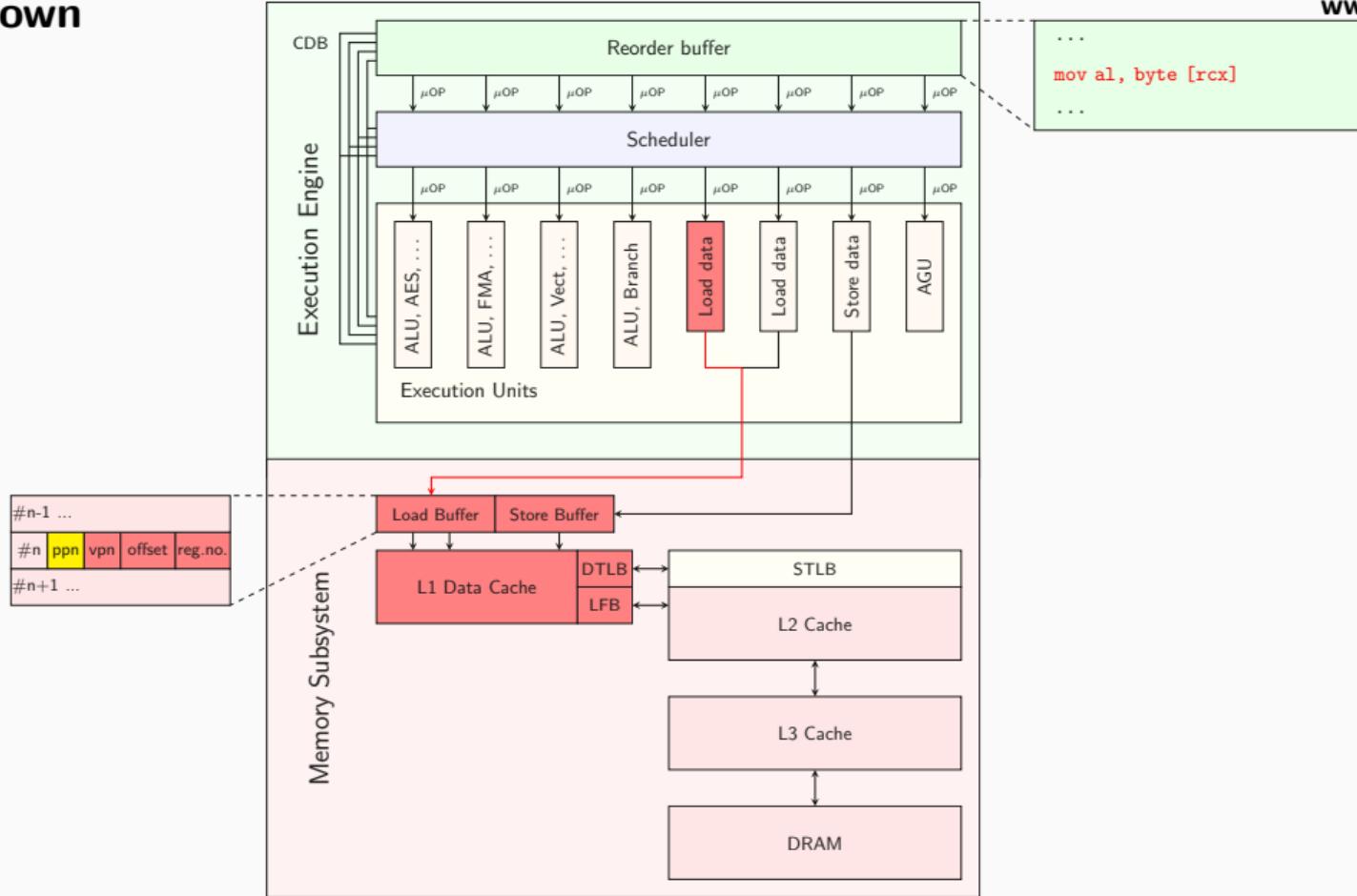


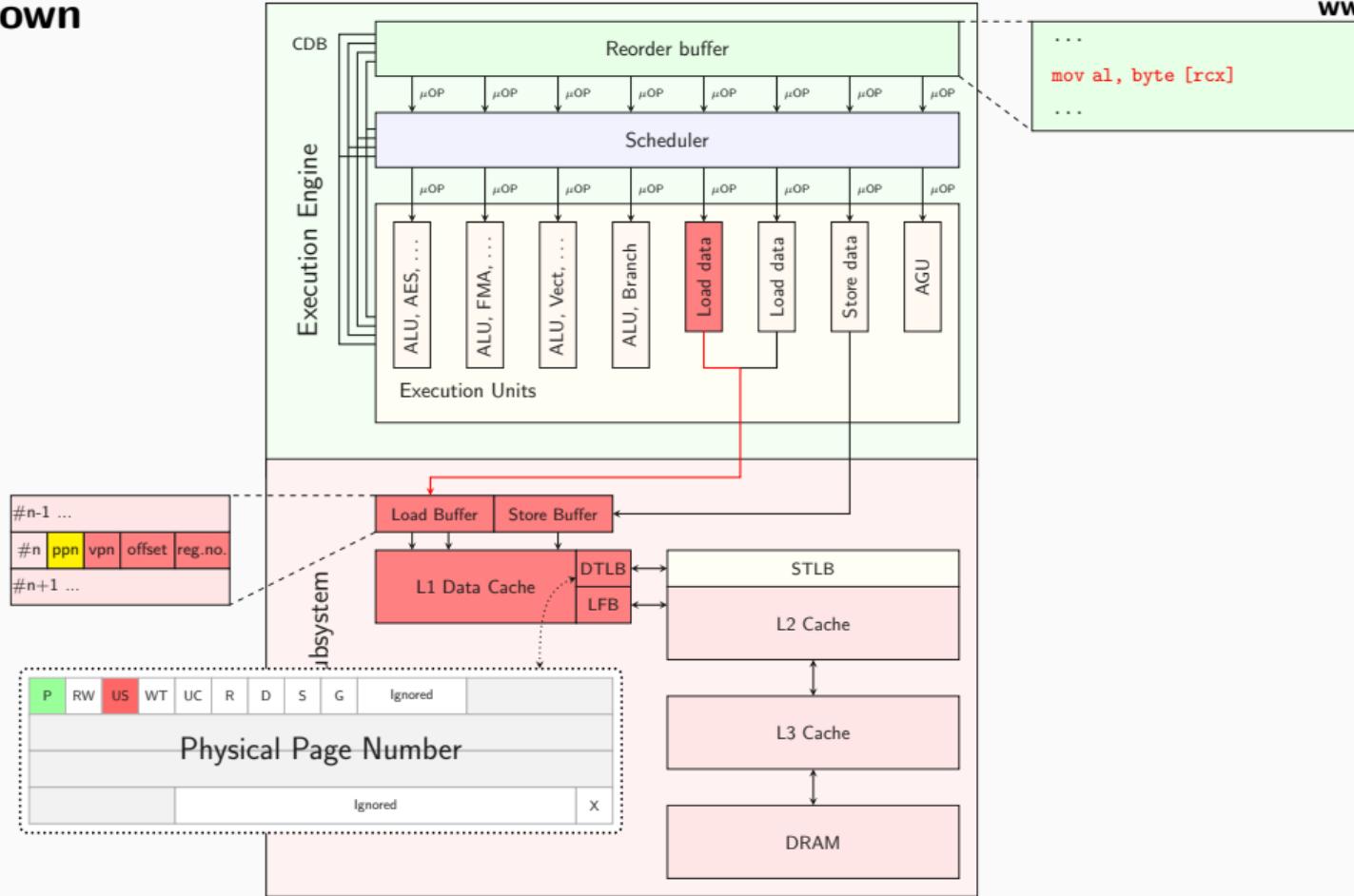


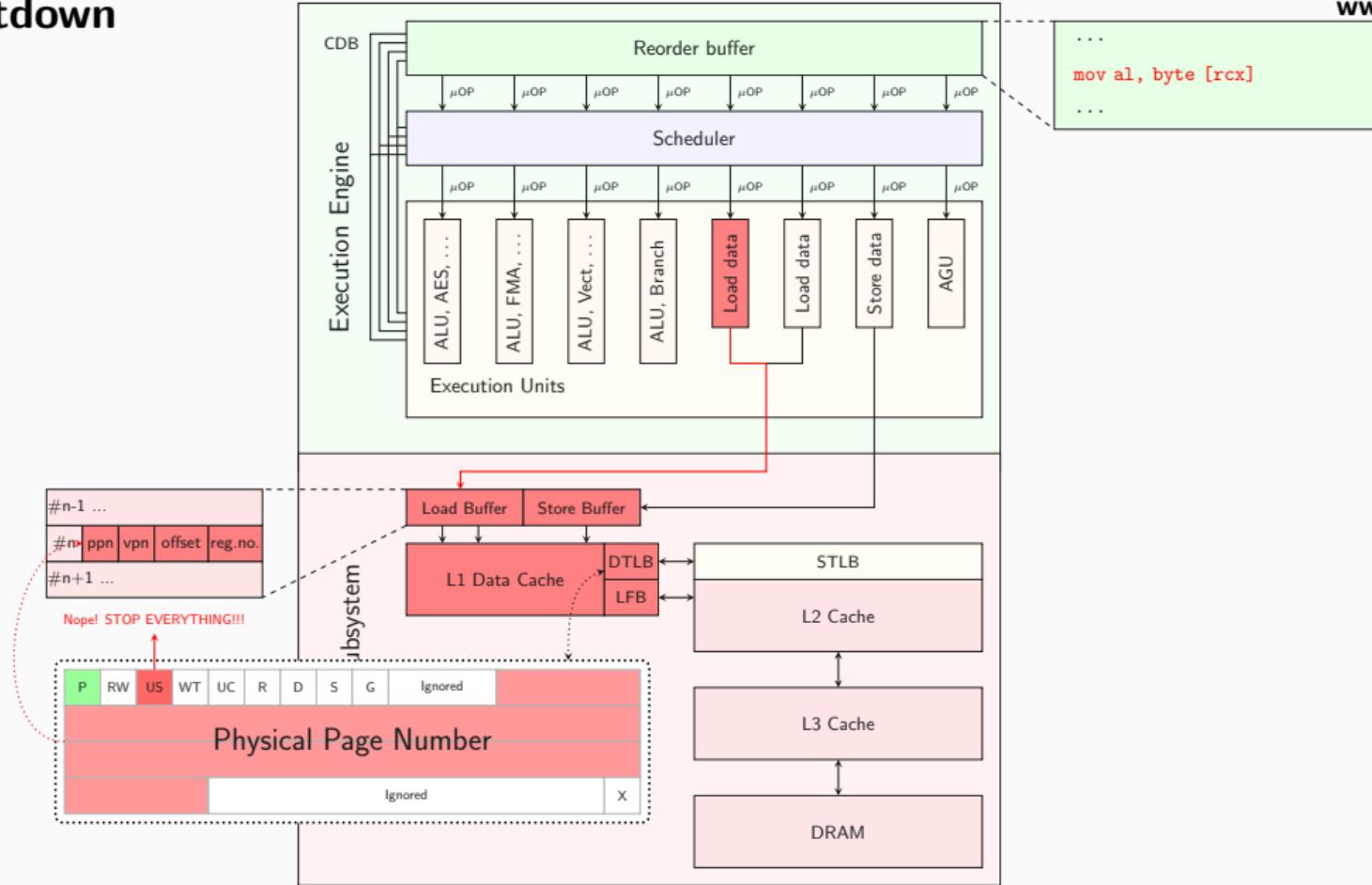


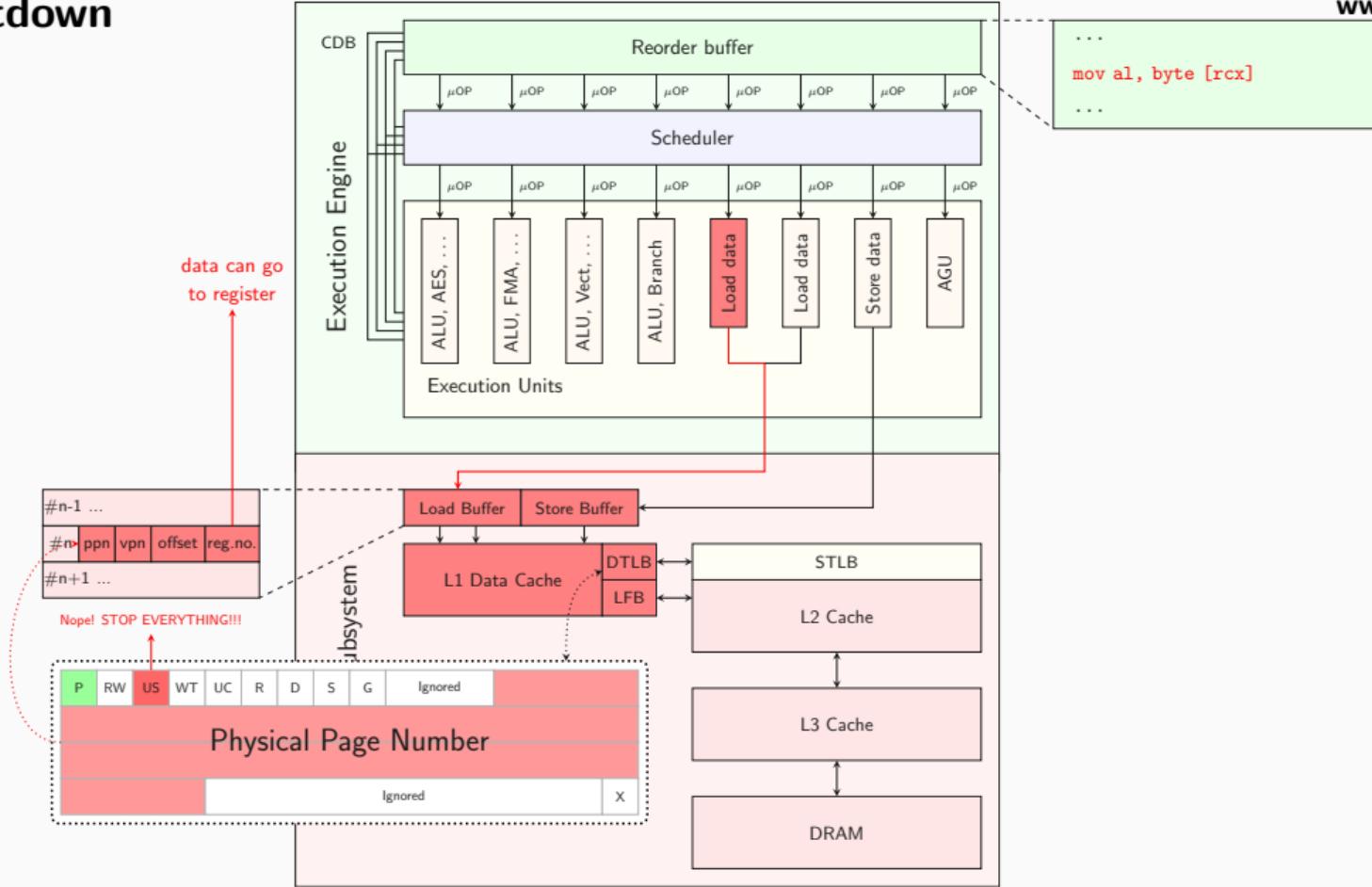


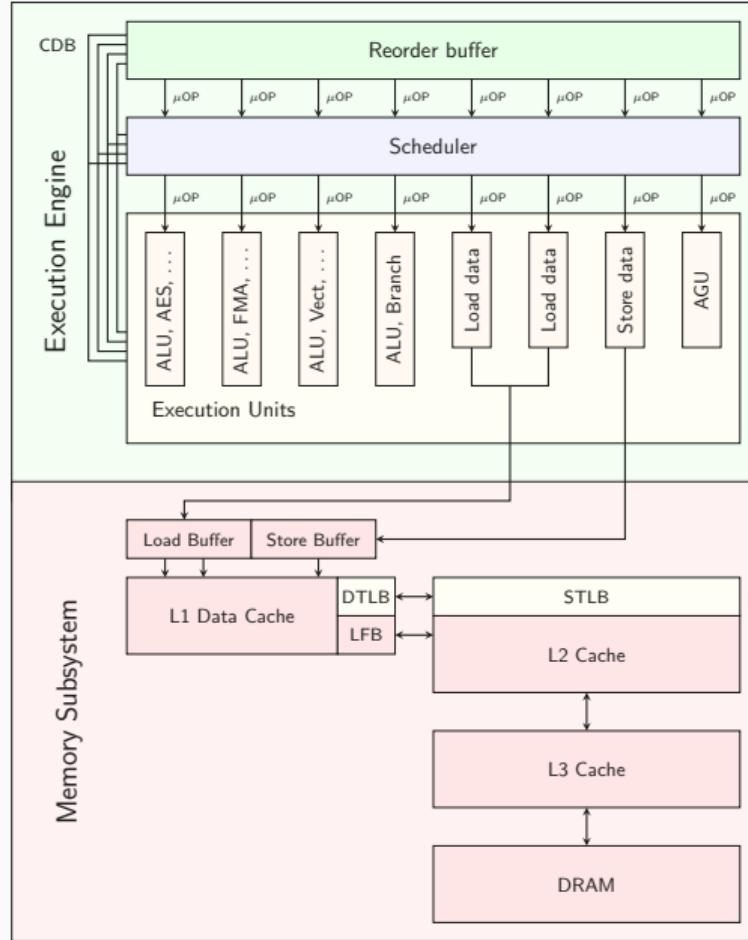


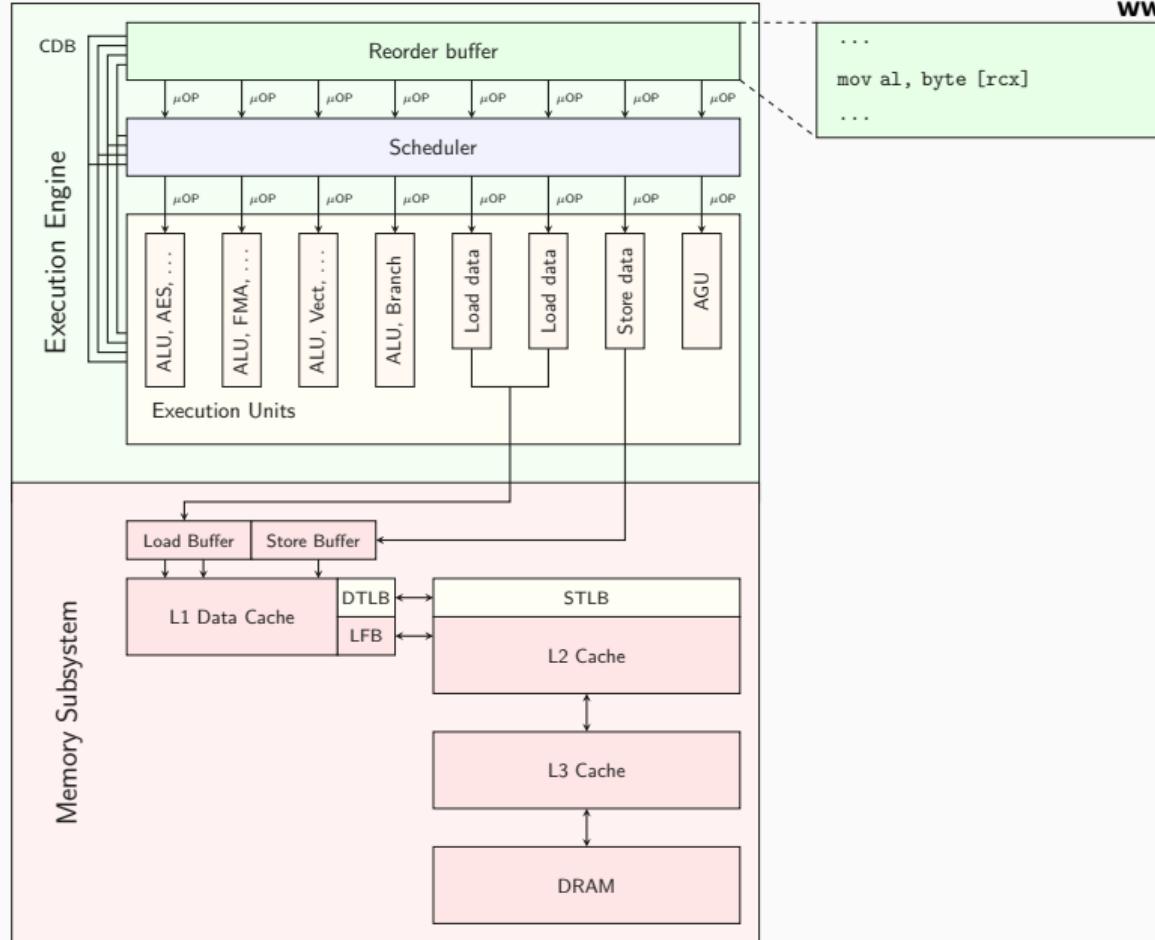


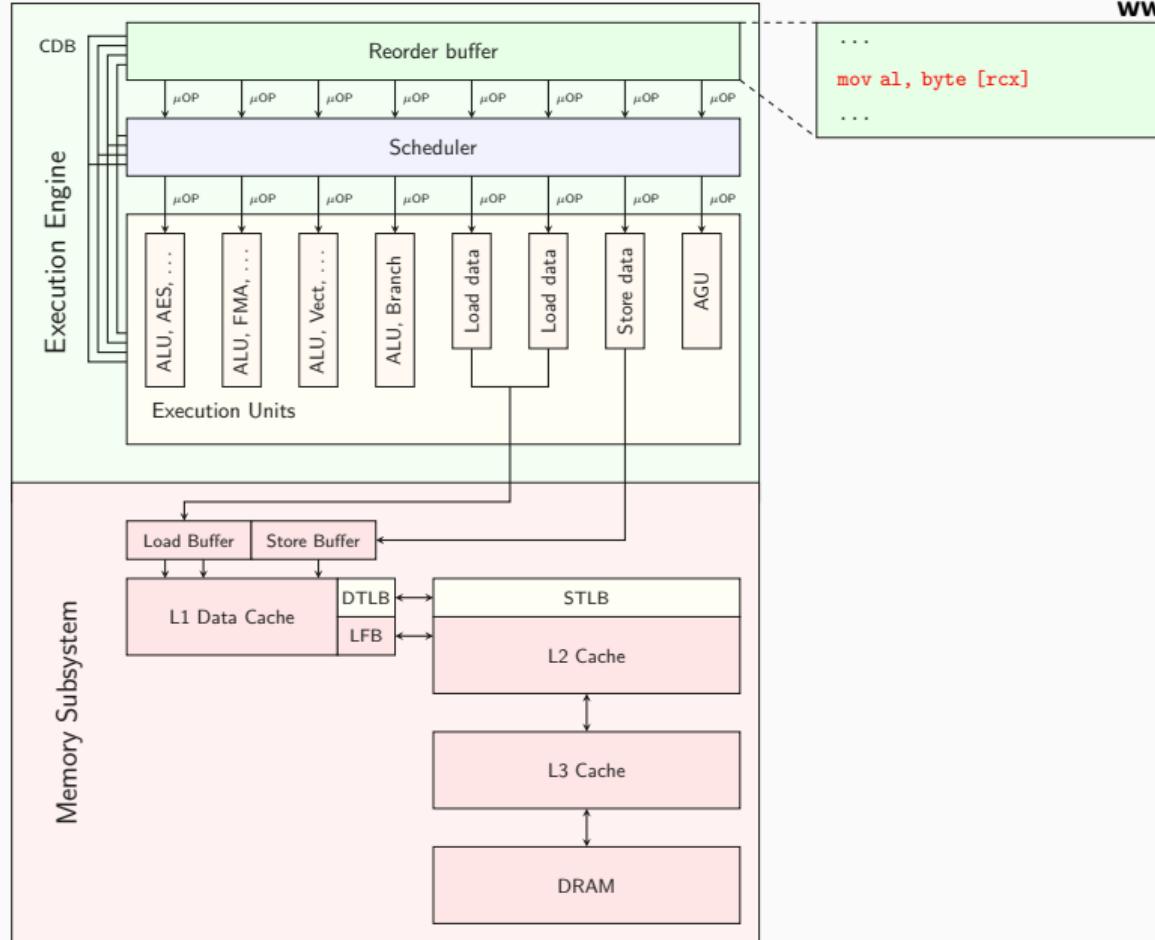


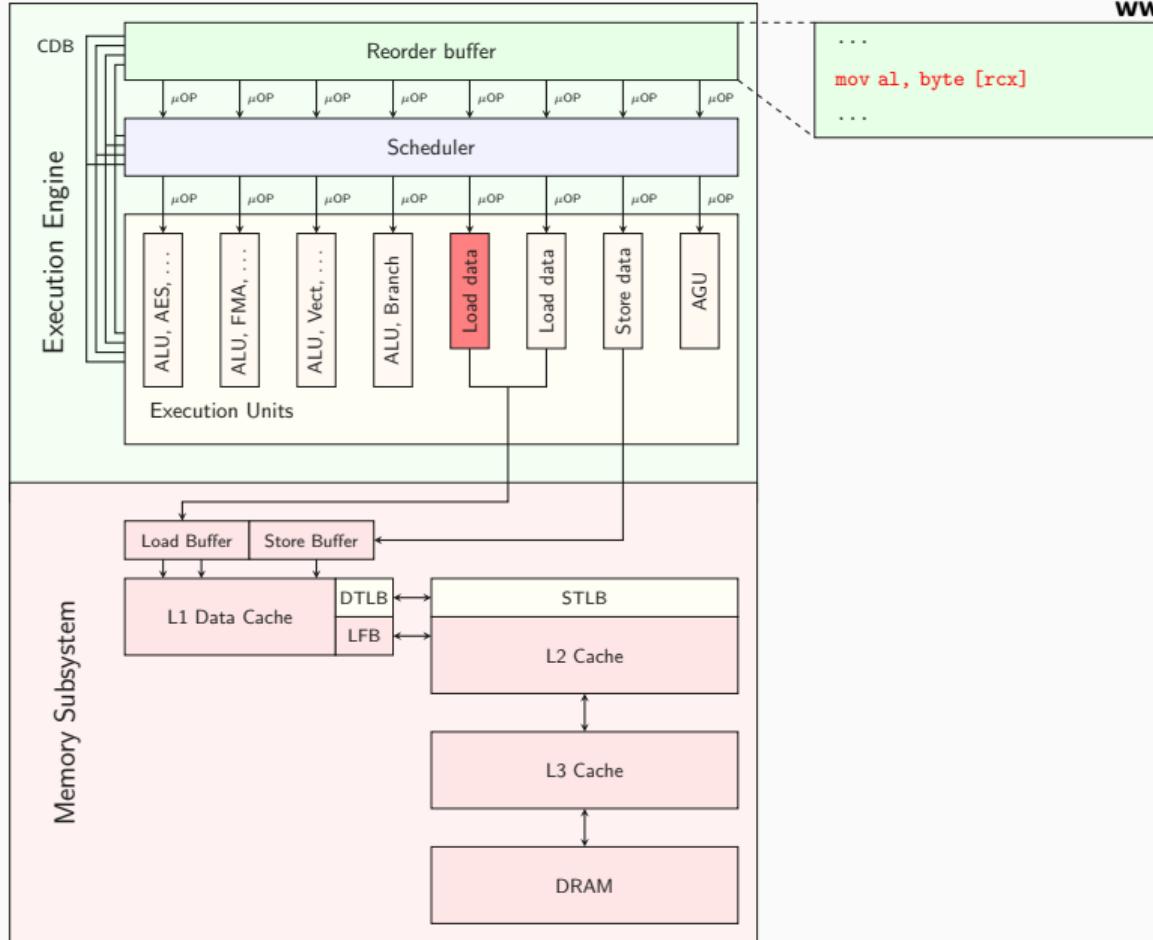


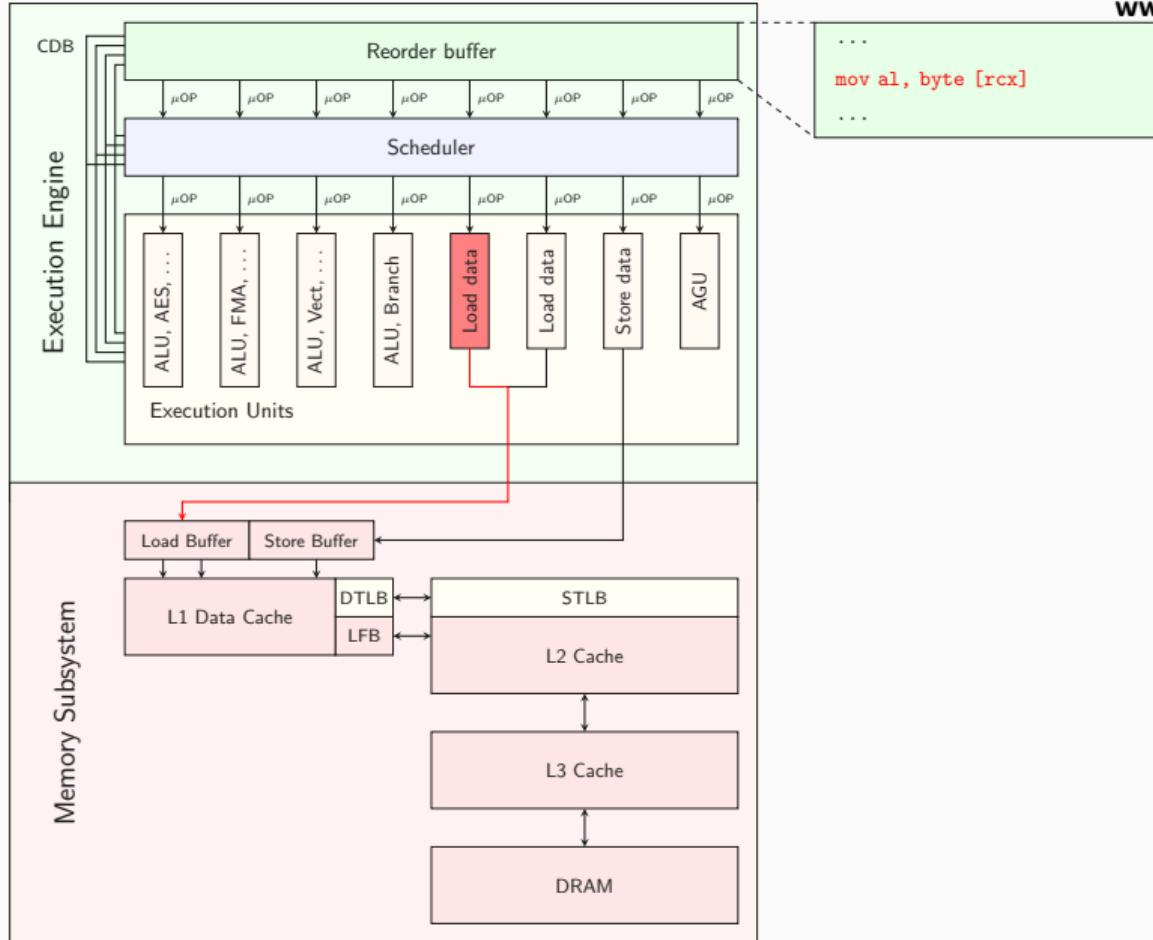


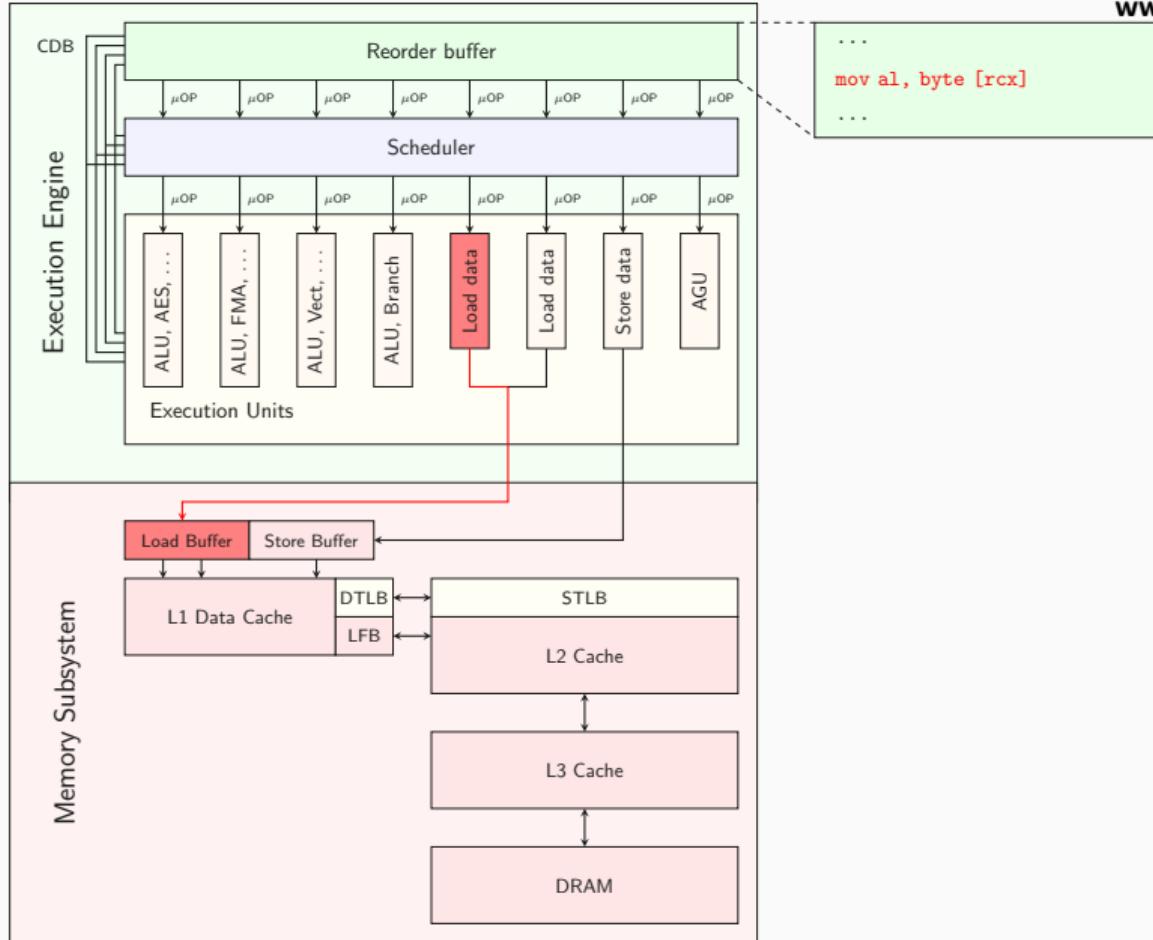


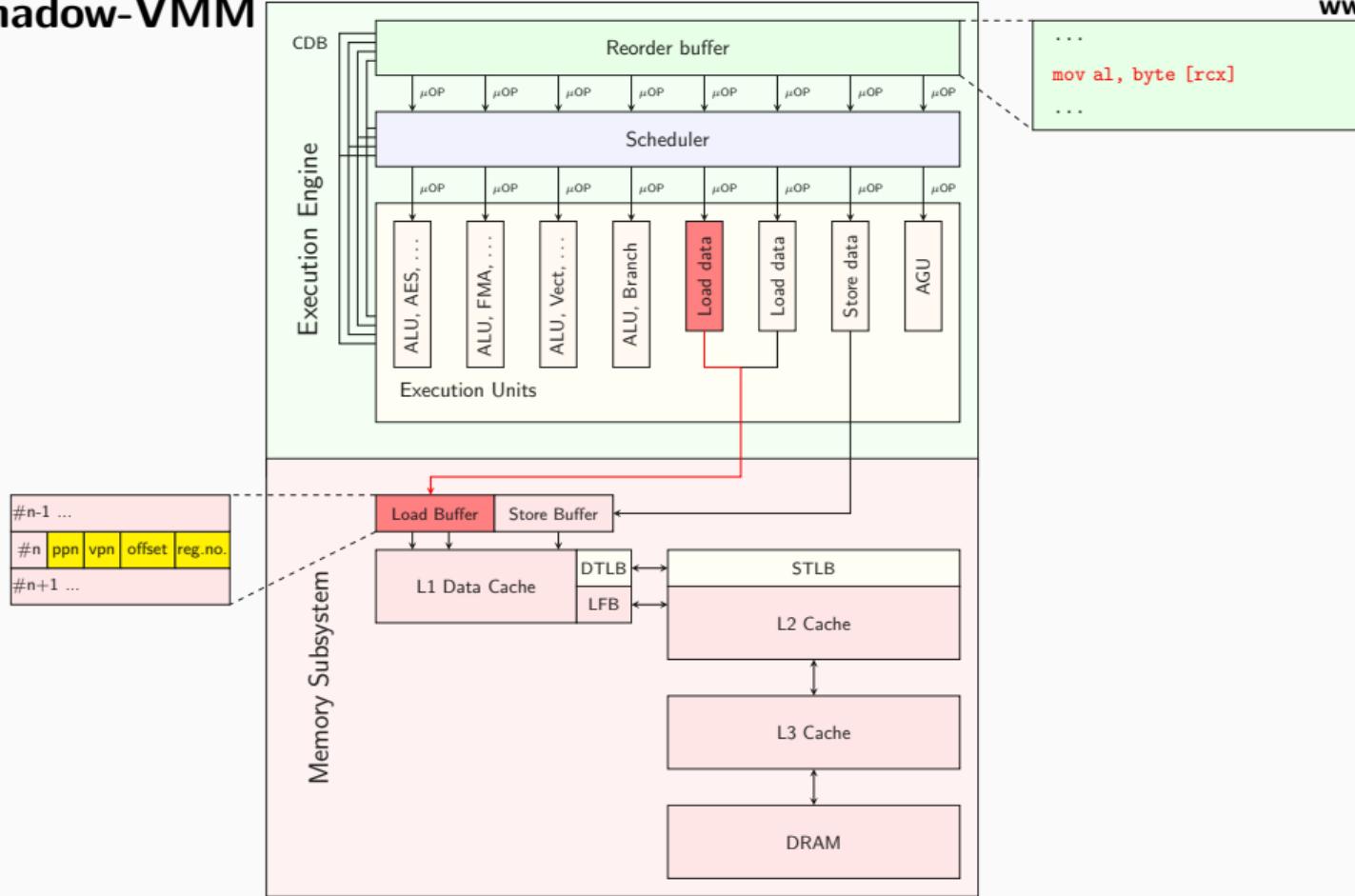


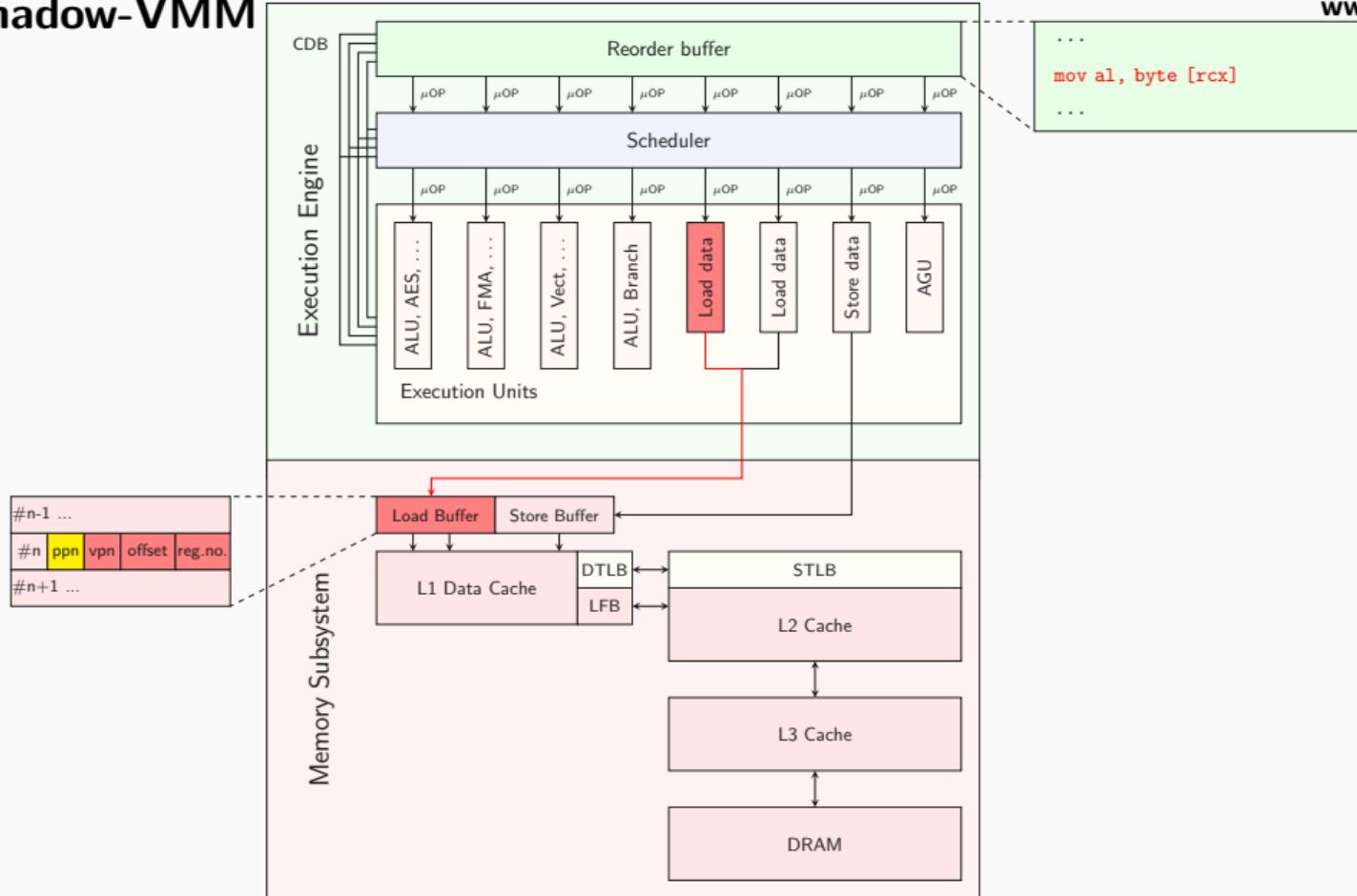


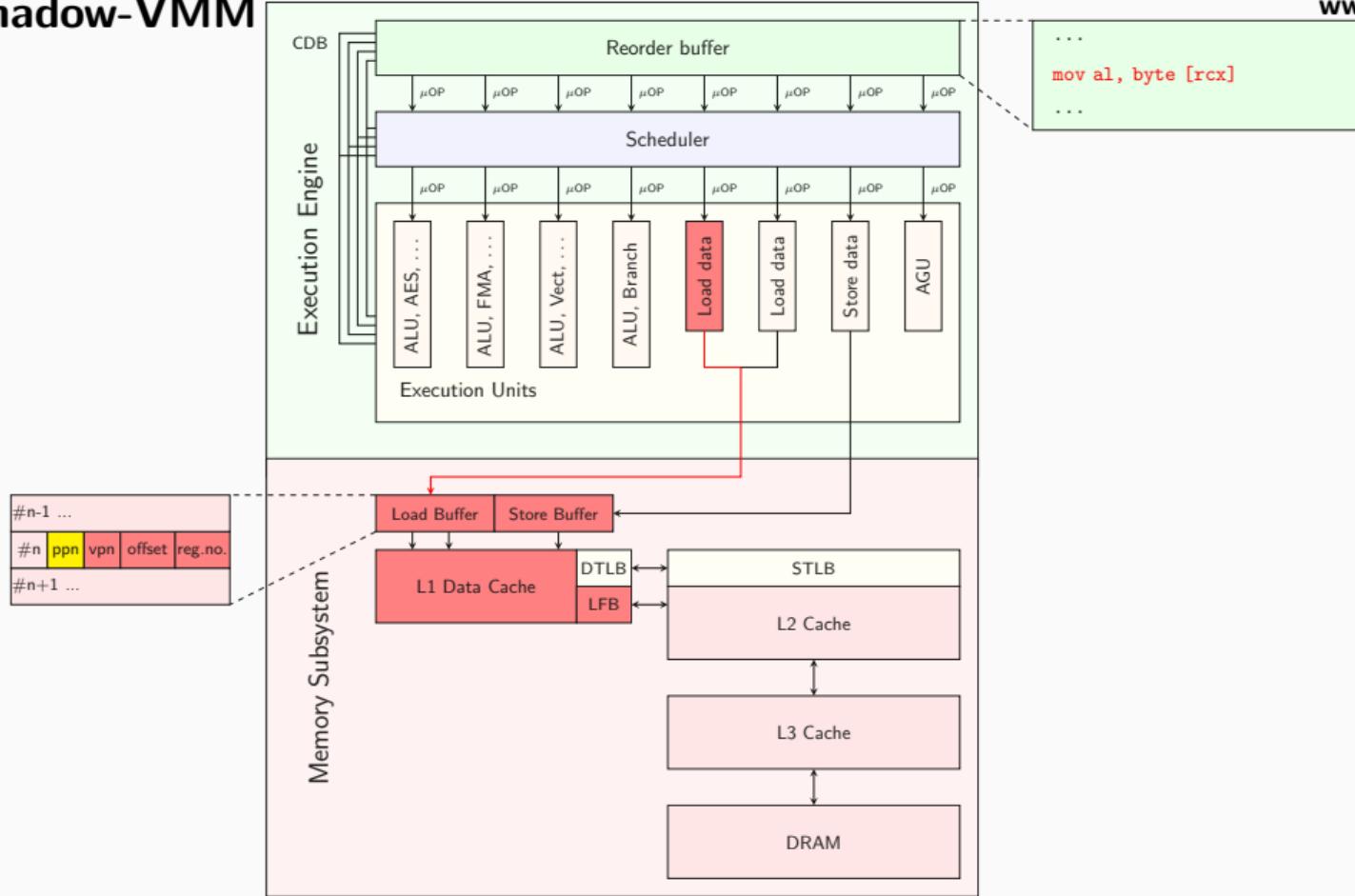


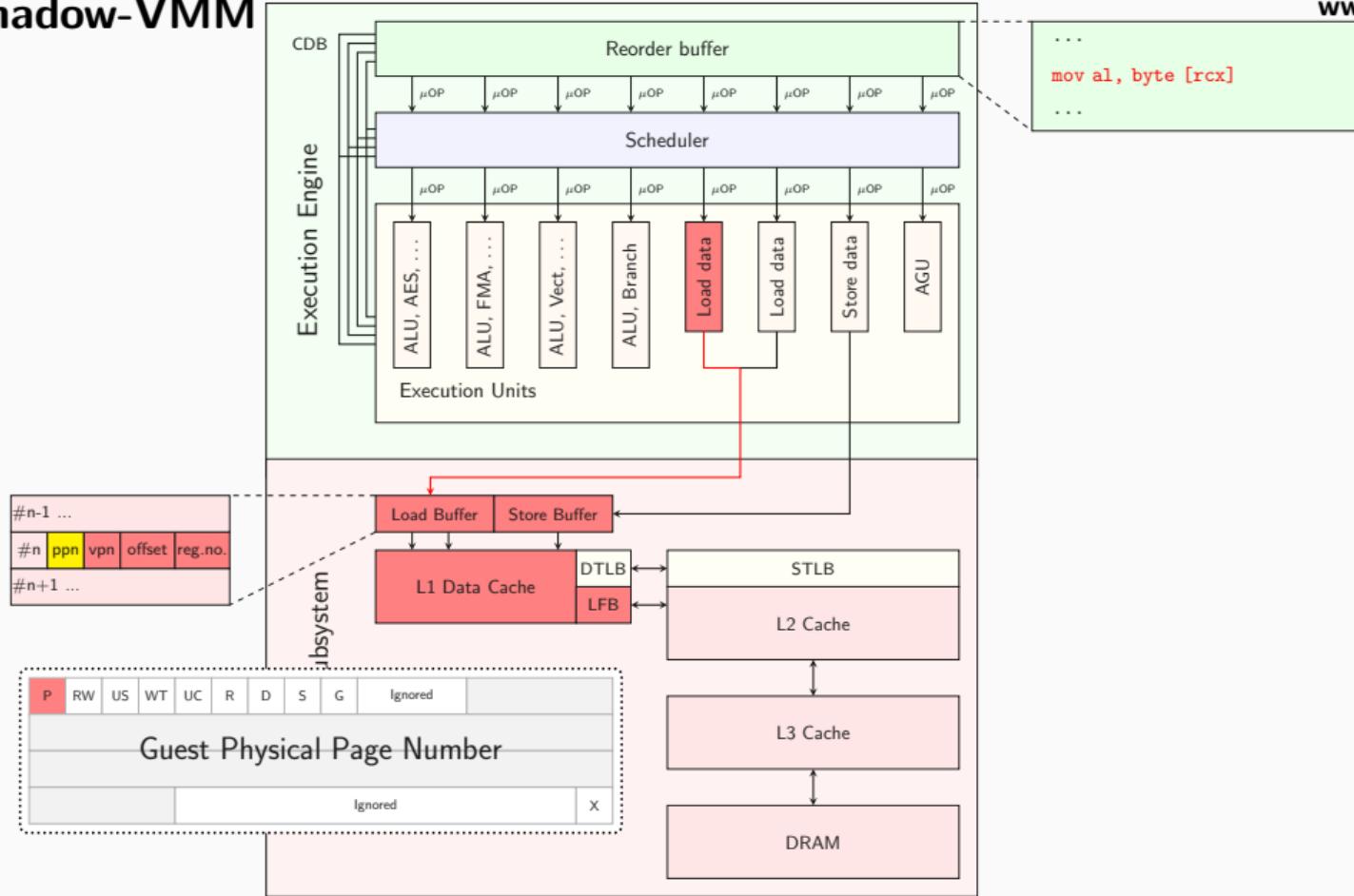


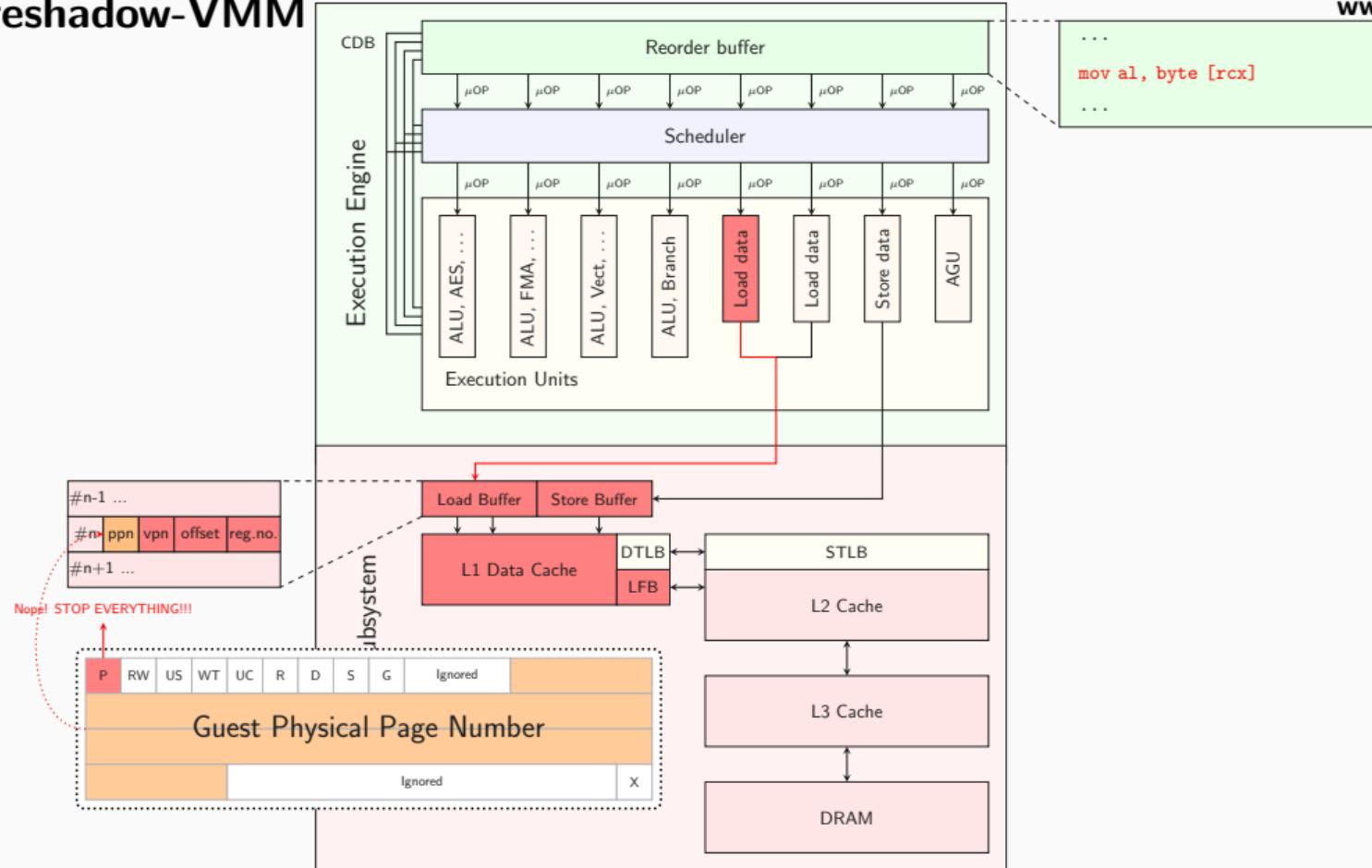


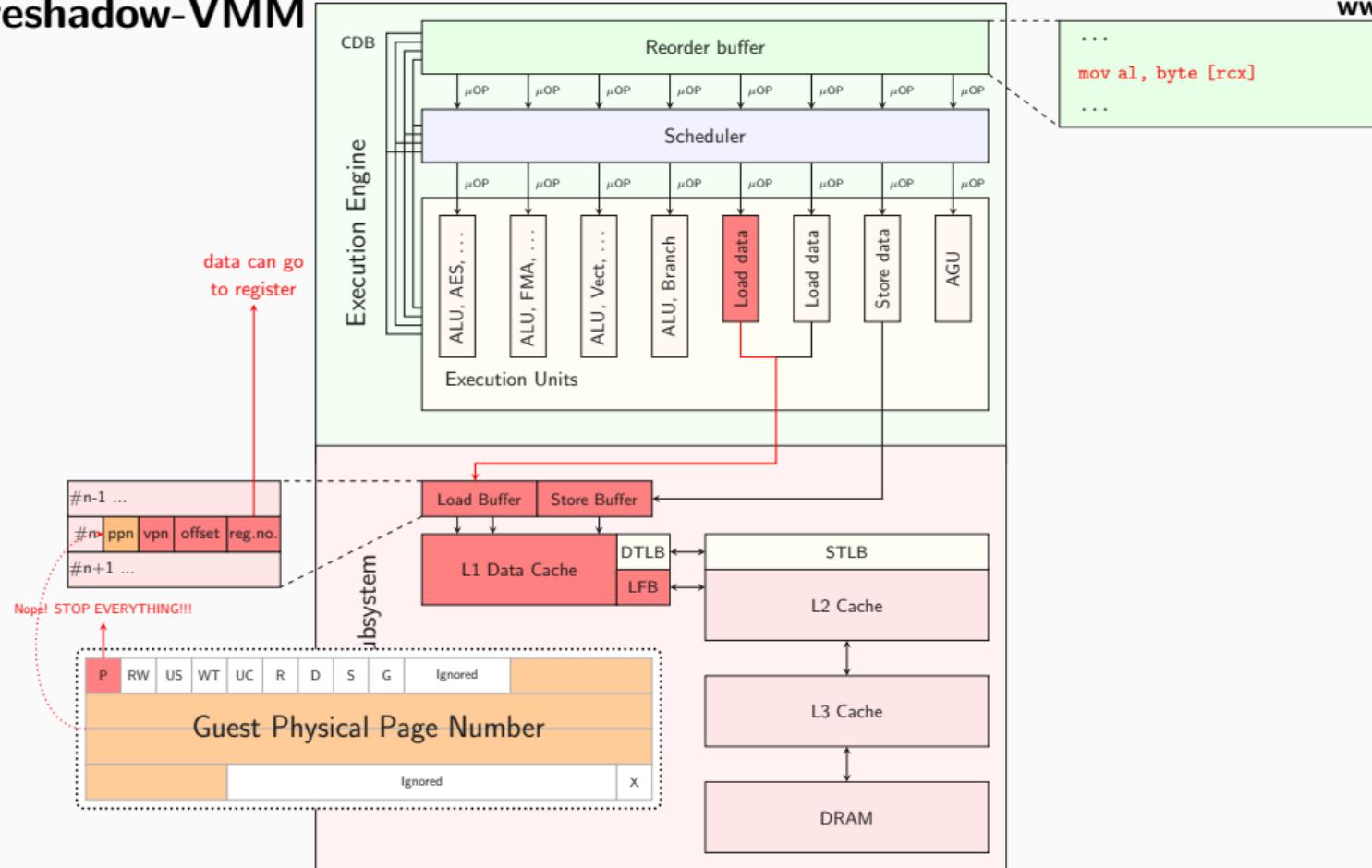


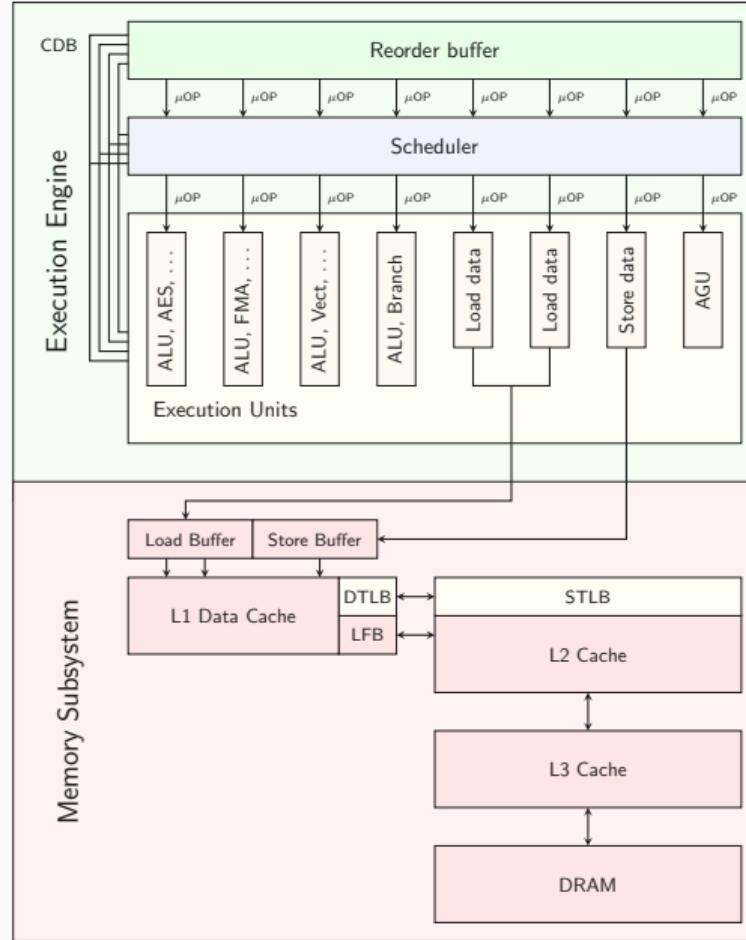


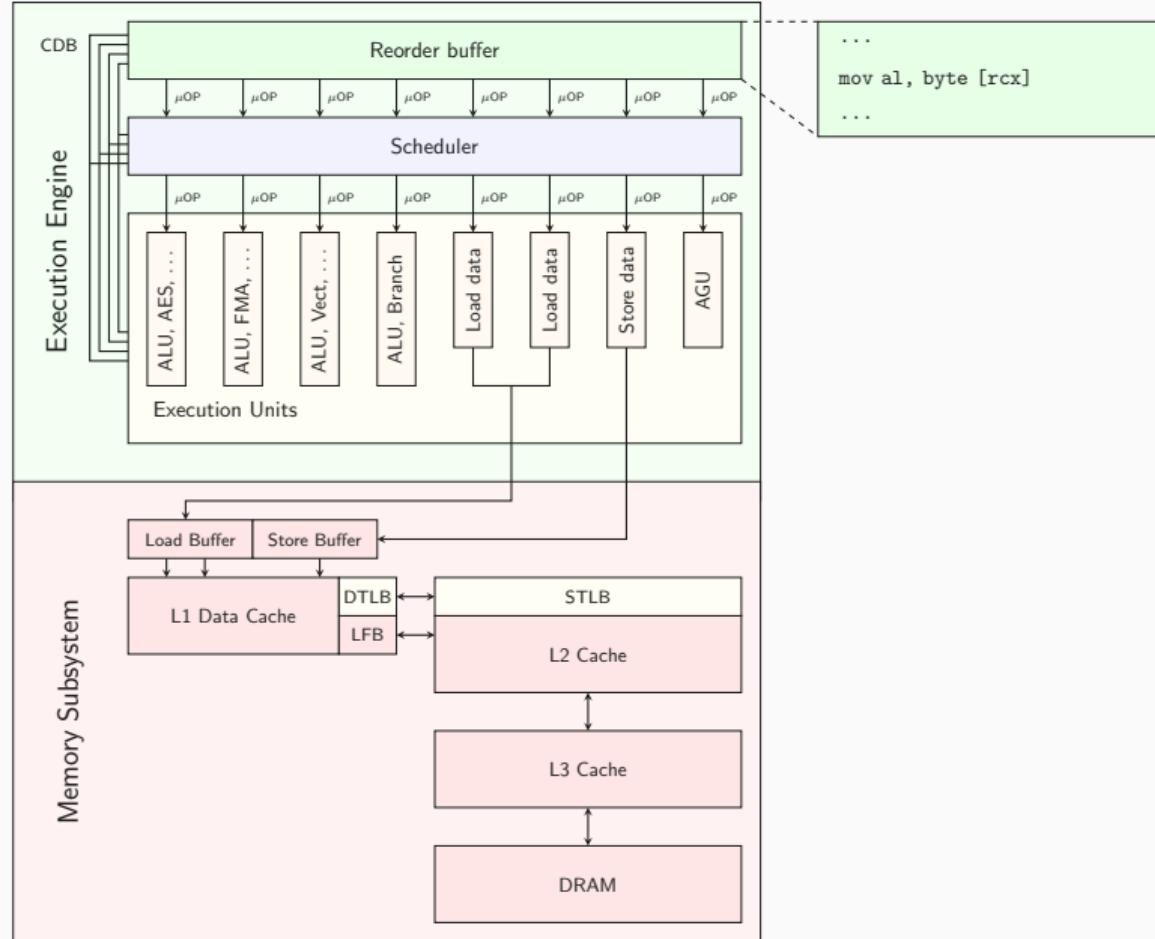


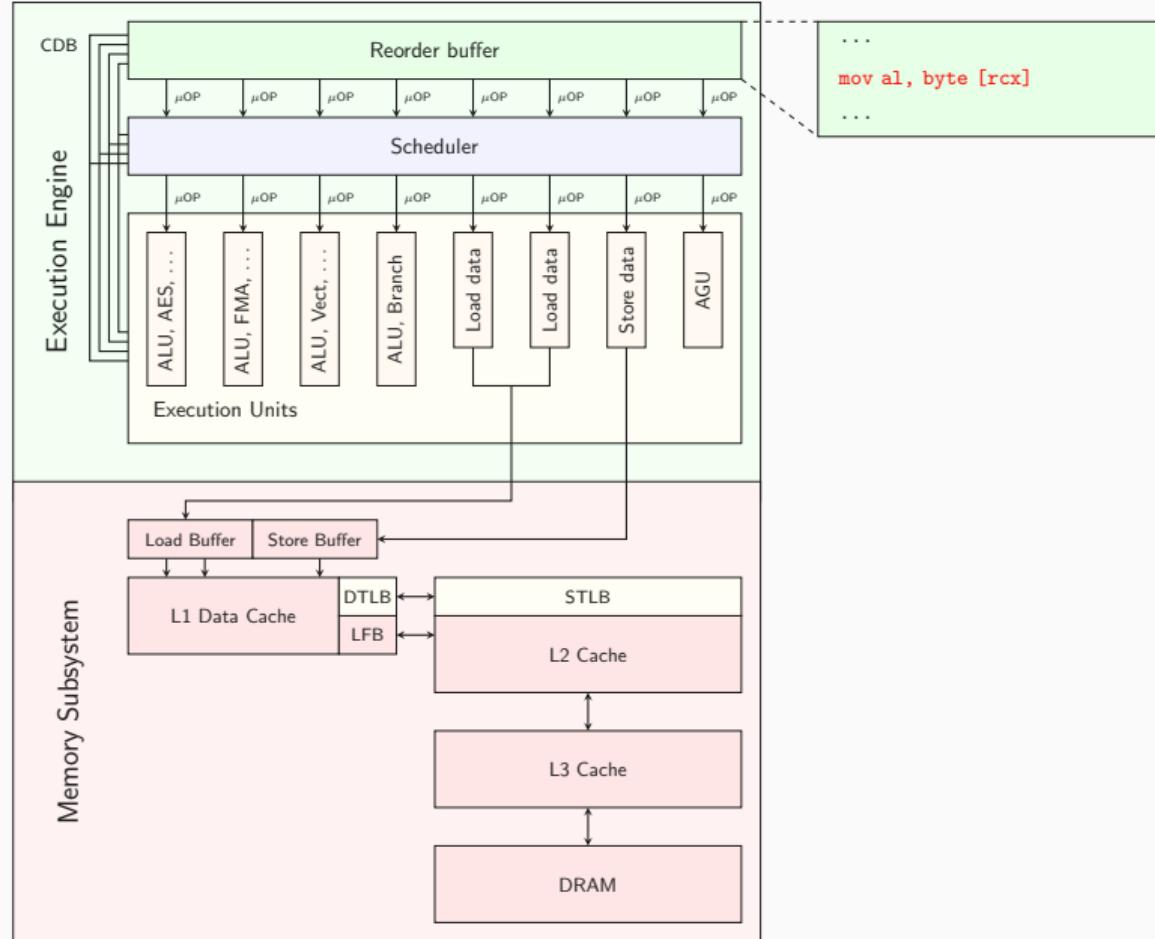


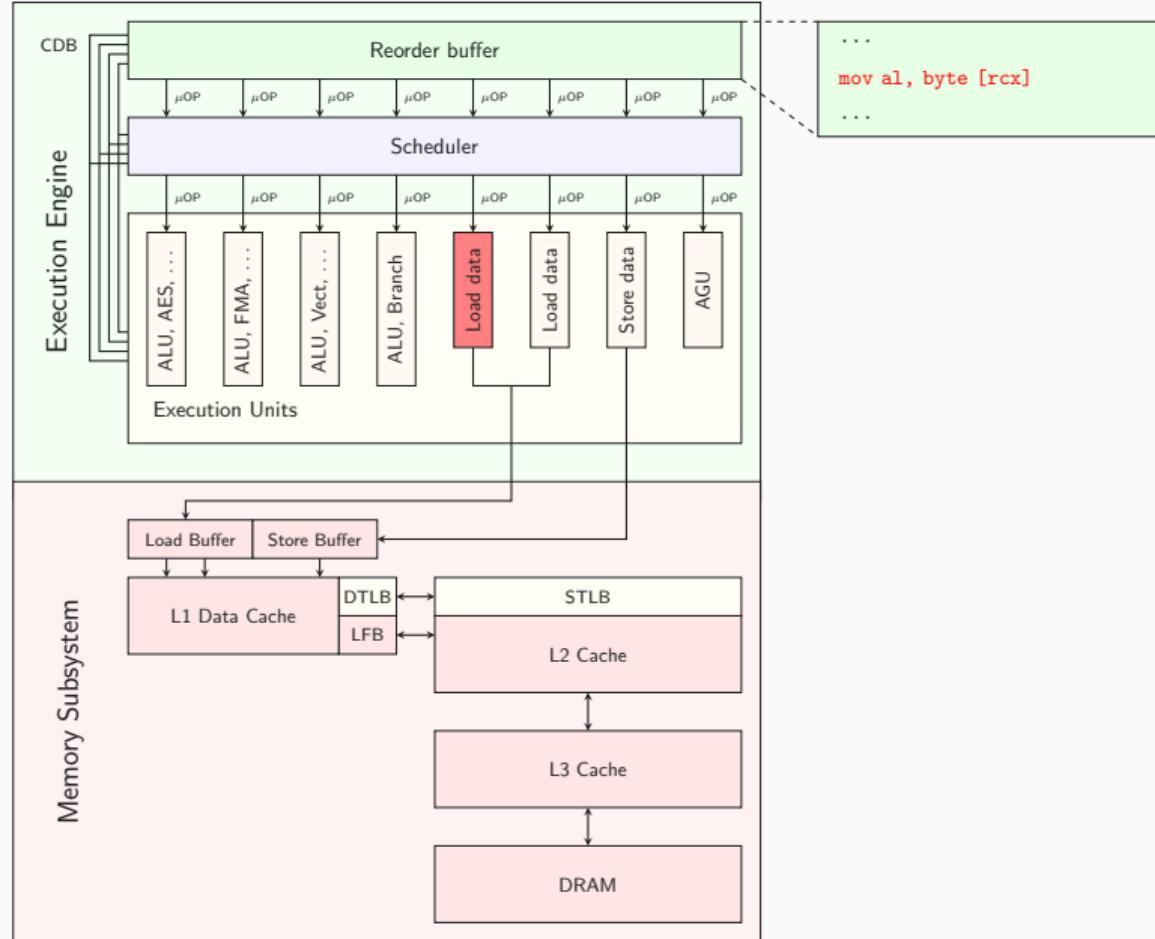


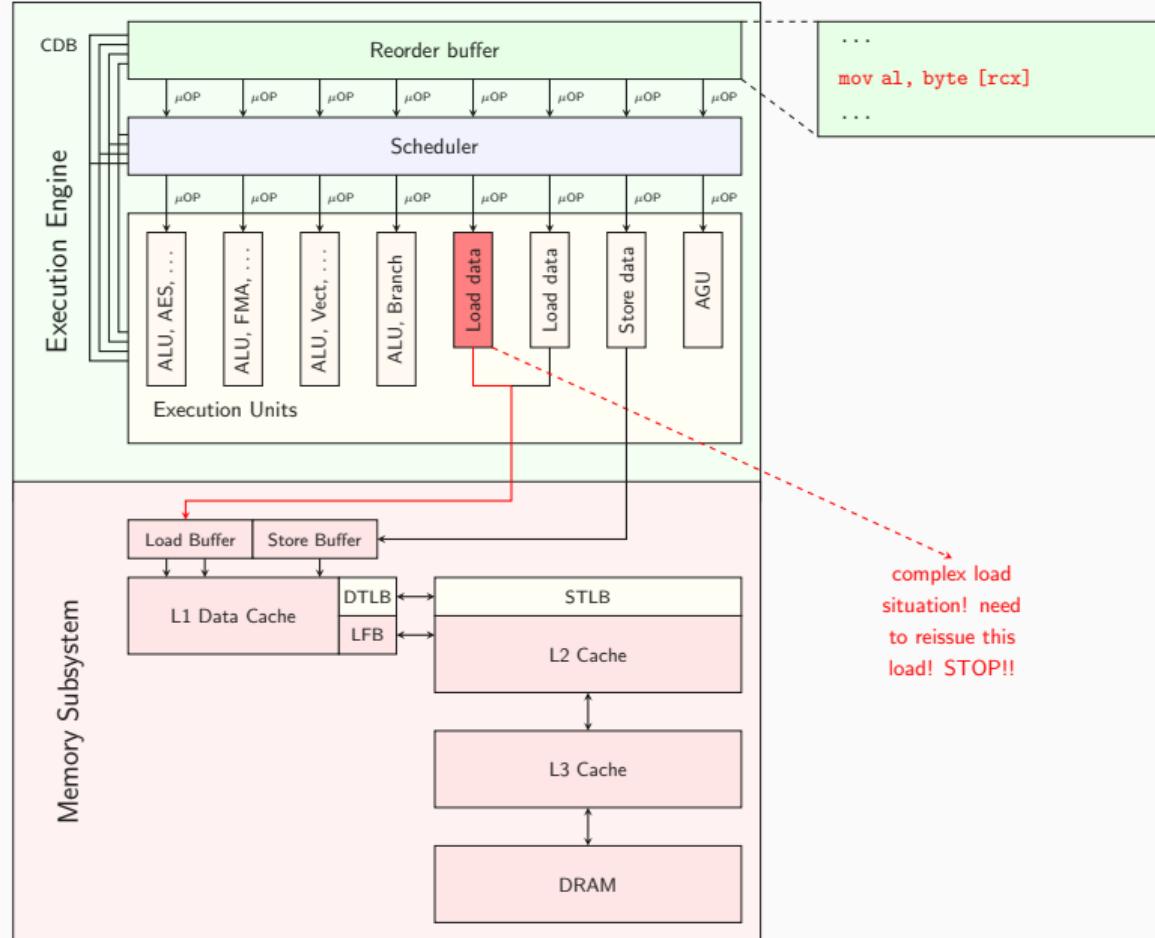


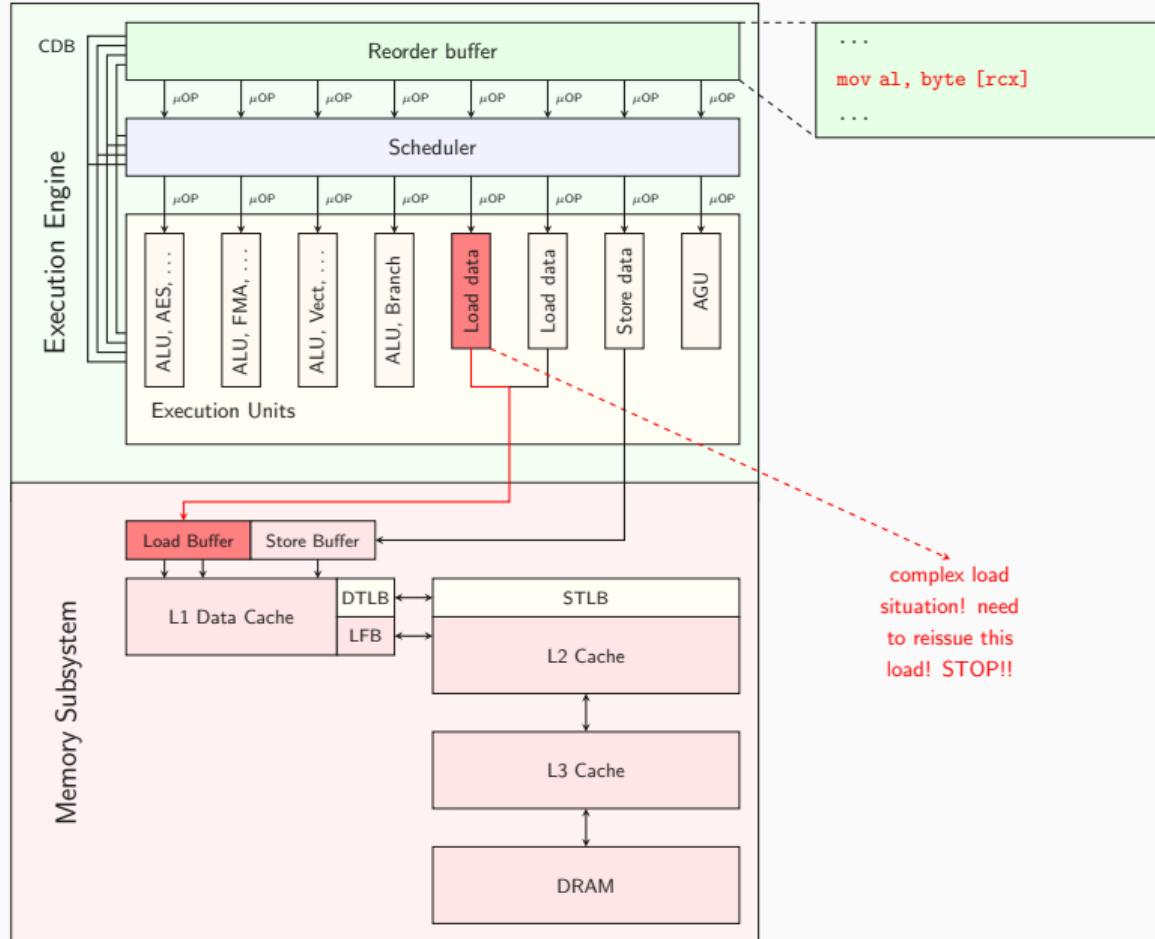


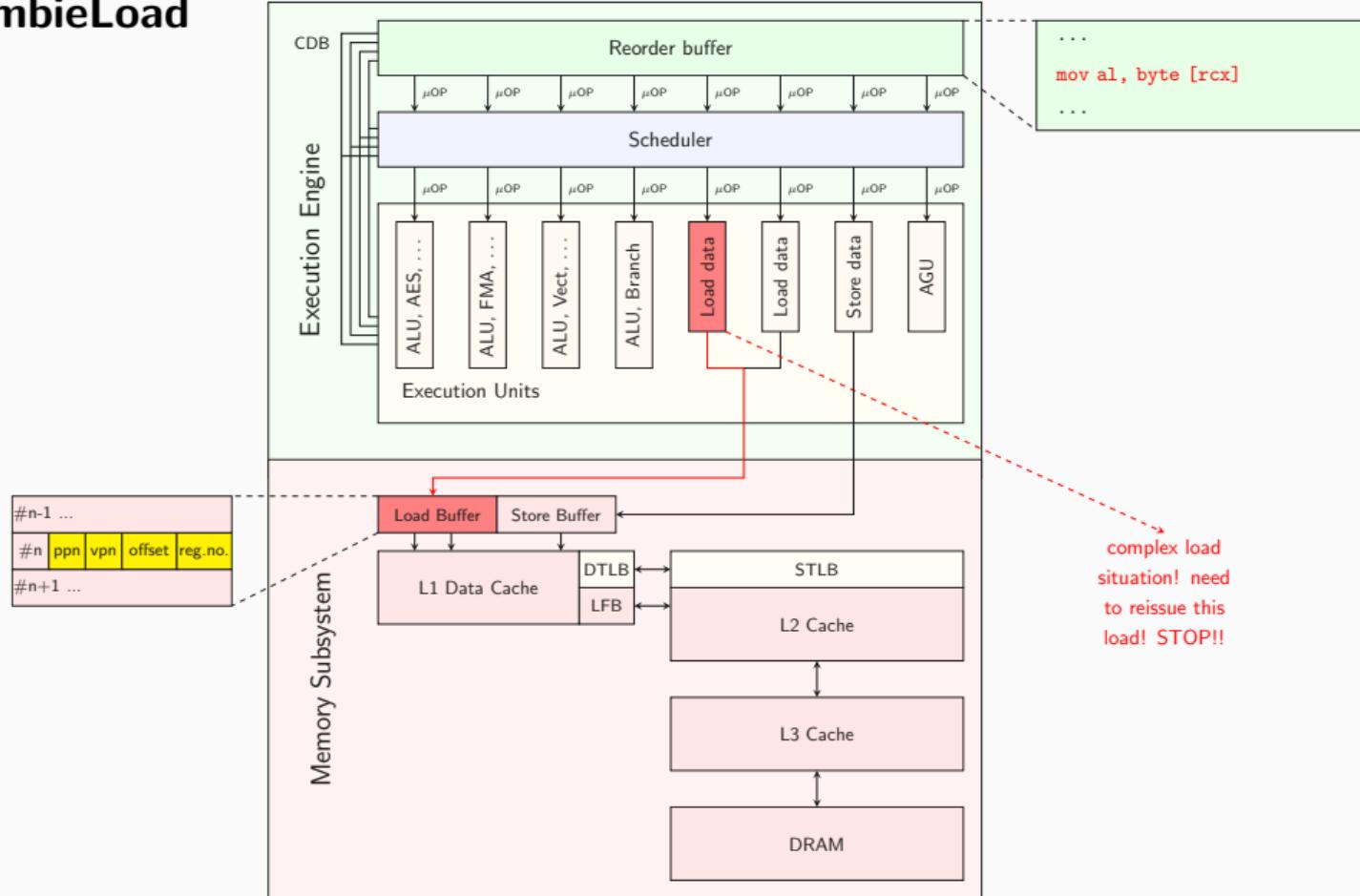


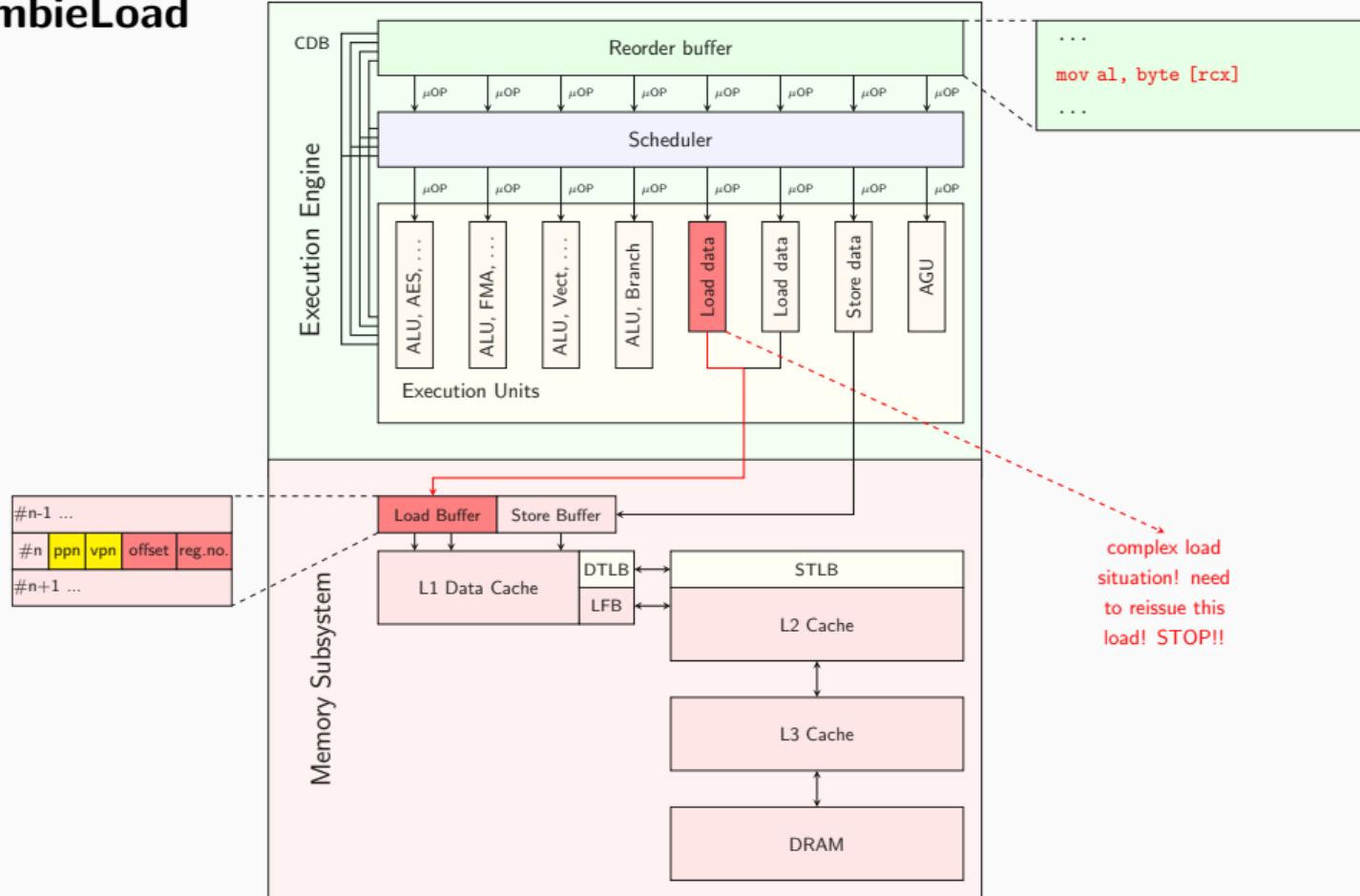


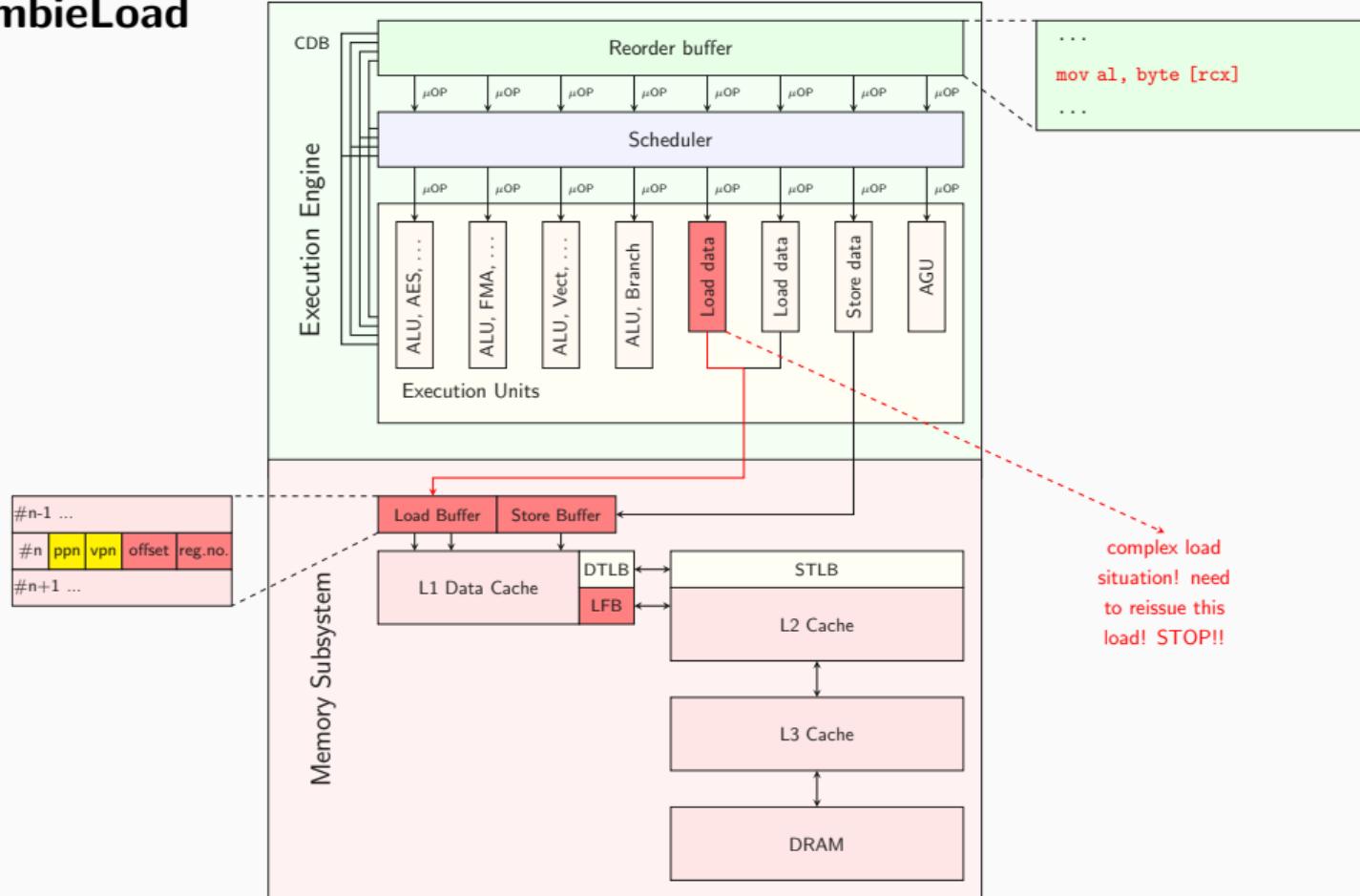


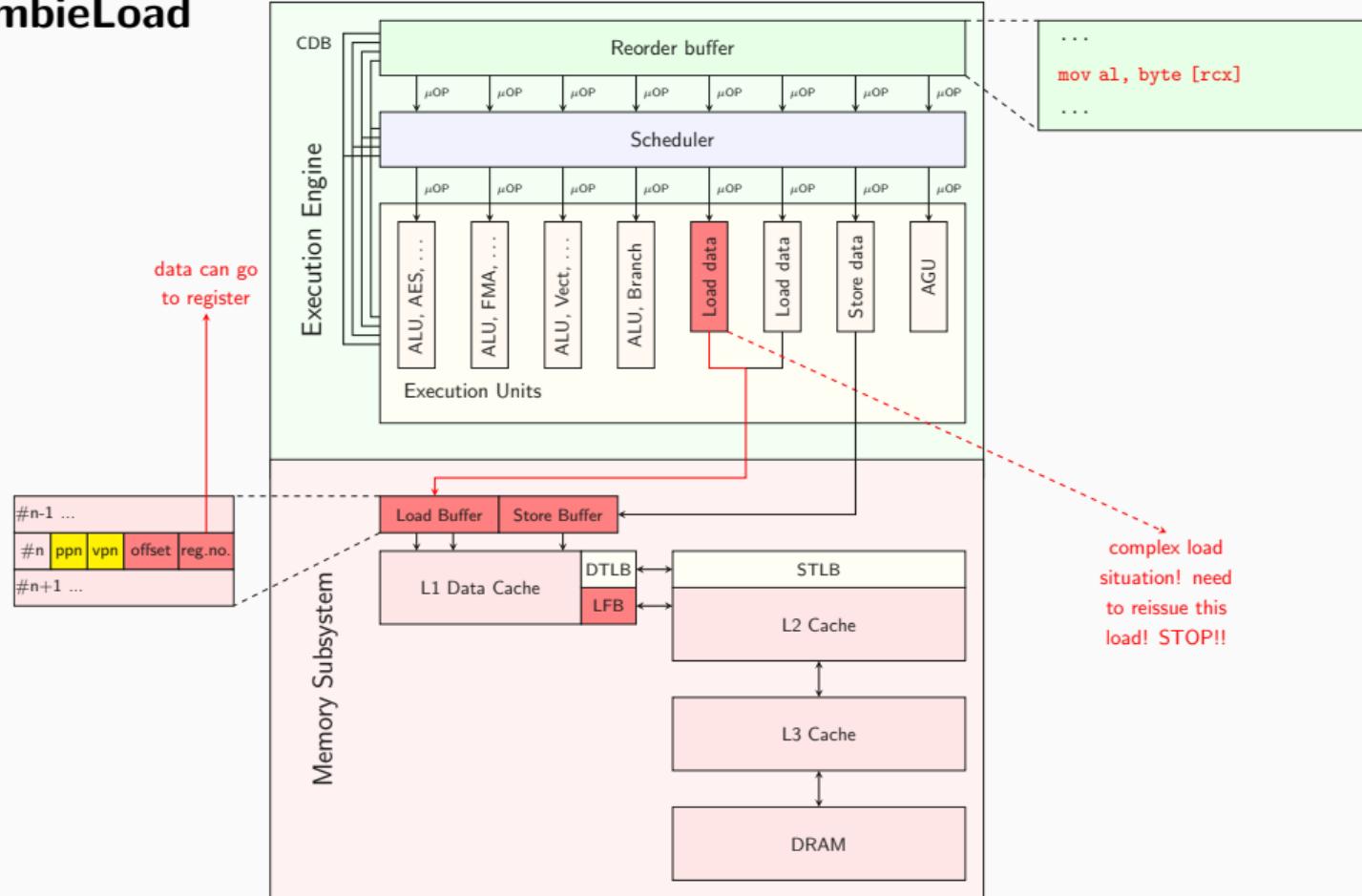














zombieload:zsh — Konsole <2>

File Edit View Bookmarks Settings Help

michael@hp /tmp/zombieload %



zombieload:zsh



zombieLoad : zsh — Konsole <2>

File Edit View Bookmarks Settings Help

michael@hp /tmp/zombieLoad %



zombieLoad : zsh





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



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



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







Computer science:



Computer science:

- computer engineering



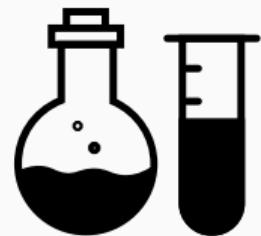
Computer science:

- computer engineering
- philosophy

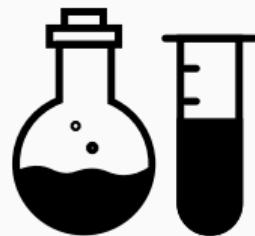


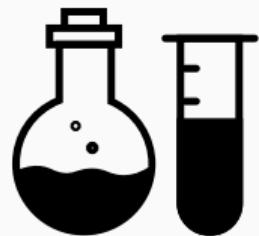
Computer science:

- computer engineering
- philosophy
- artificial science



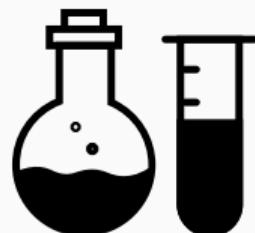
“The Sciences of the Artificial” (Herbert A. Simon, 1969)





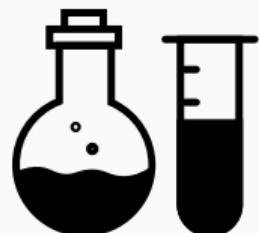
“The Sciences of the Artificial” (Herbert A. Simon, 1969)

- Natural sciences: studying something natural



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

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

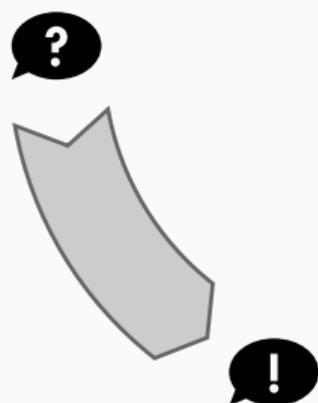


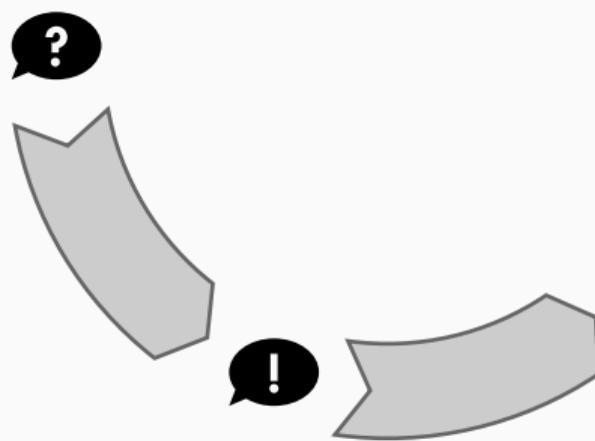
“The Sciences of the Artificial” (Herbert A. Simon, 1969)

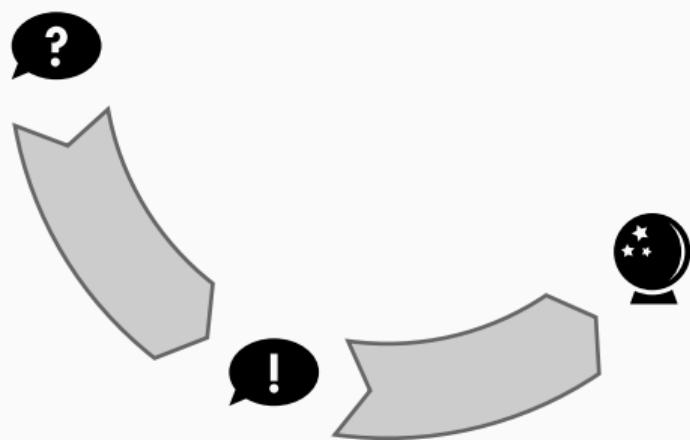
- Natural sciences: studying something natural
 - Artificial science: studying something artificial (something human-made) **as if it was something natural**
- A consequence of **complexity**

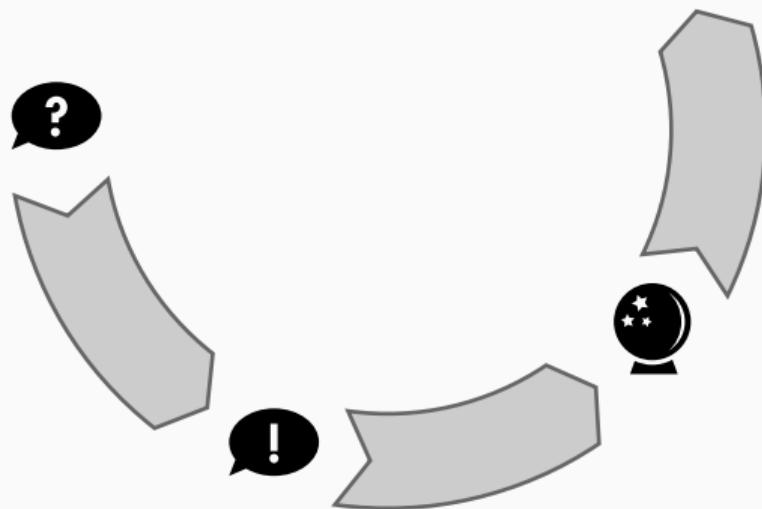


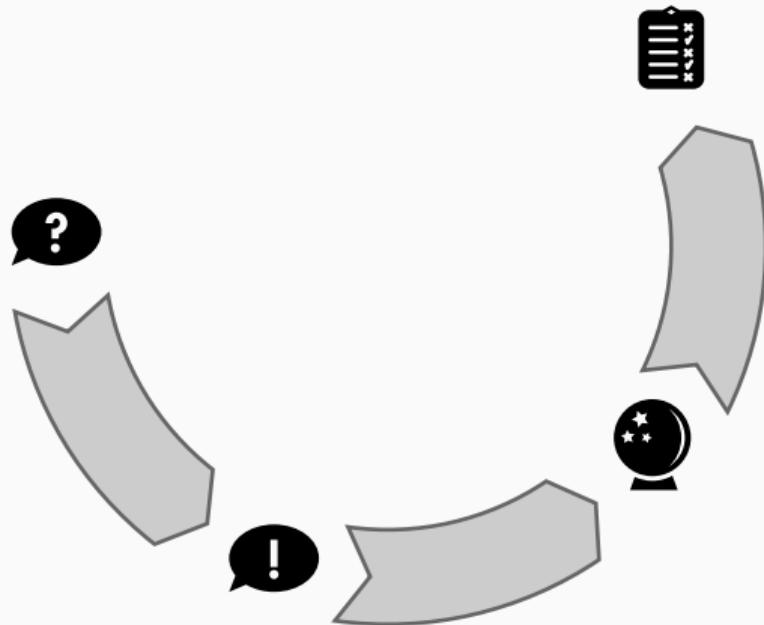


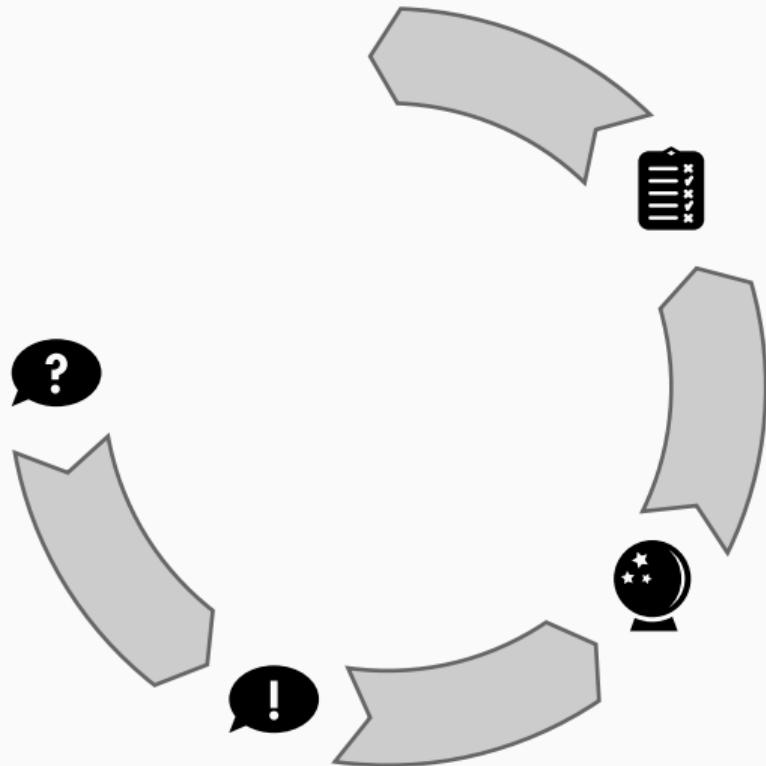


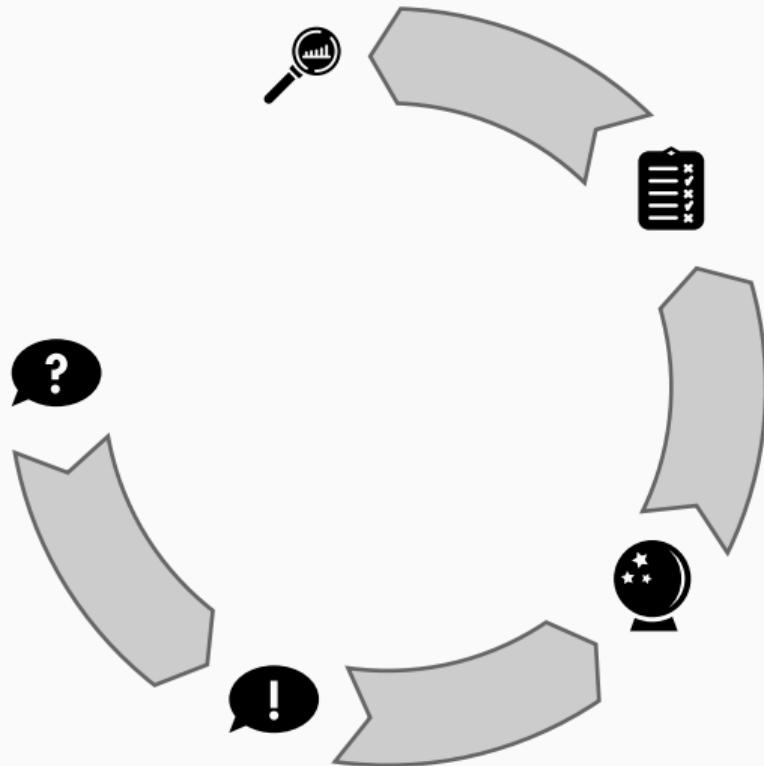


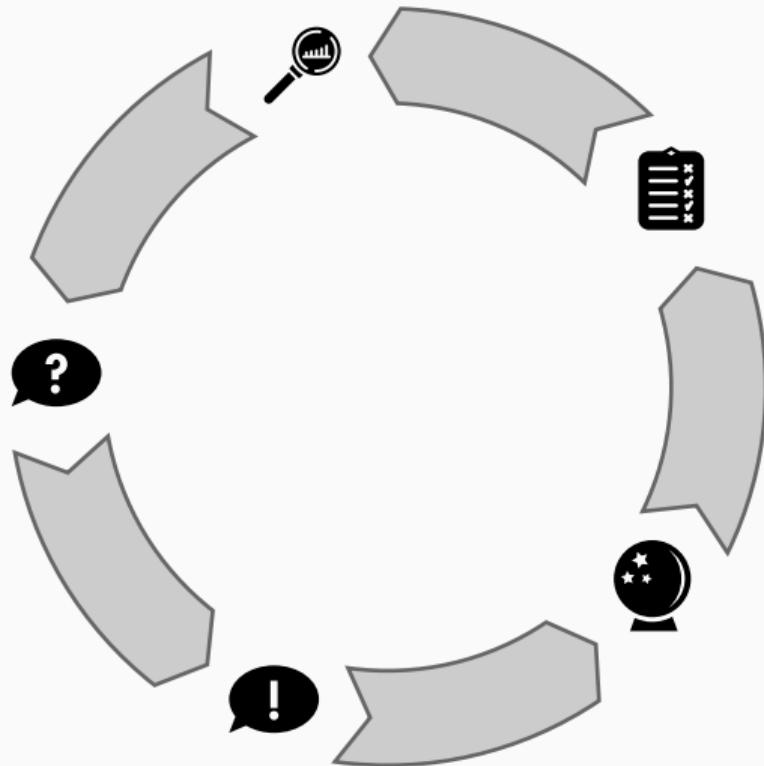


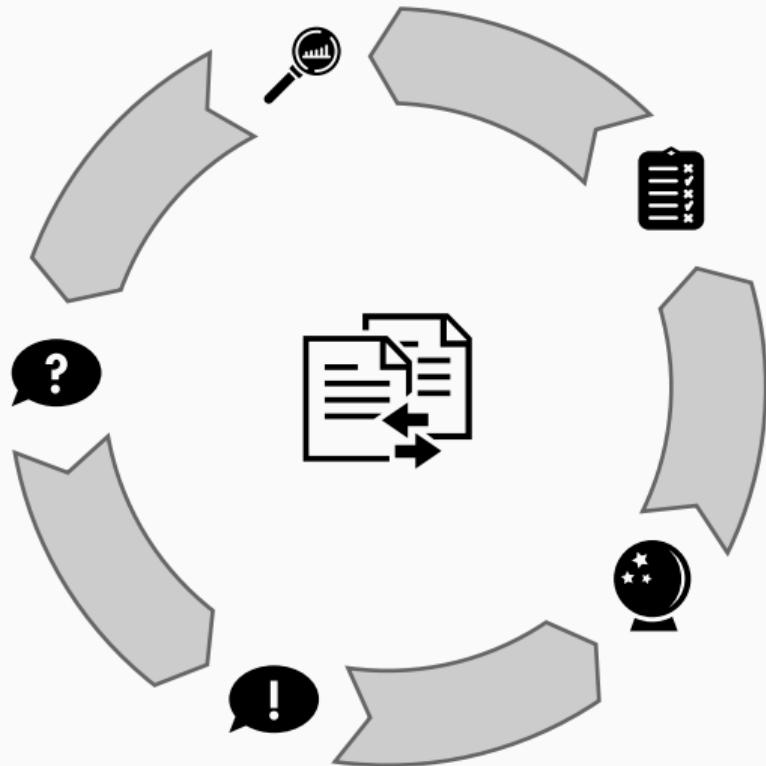












We have ignored microarchitectural attacks for many years:



We have ignored microarchitectural attacks for many years:



- attacks on crypto

We have ignored microarchitectural attacks for many years:



- attacks on crypto → “software should be fixed”

We have ignored microarchitectural attacks for many years:



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

We have ignored microarchitectural attacks for many years:



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

We have ignored microarchitectural attacks for many years:



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

We have ignored microarchitectural attacks for many years:



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

We have ignored microarchitectural attacks for many years:



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

We have ignored microarchitectural attacks for many years:



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

We have ignored microarchitectural attacks for many years:



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



- lower refresh rate = lower energy but more bit flips



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



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



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



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



- new class of software-based attacks



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



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

Microarchitectural Attacks

Files for exercise: <https://gruss.cc/fosad.tar.gz>

Daniel Gruss

August 29-30, 2019

Graz University of Technology