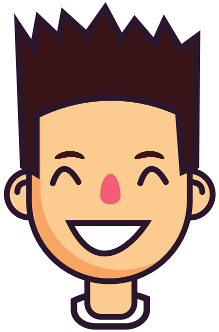
ZombieLoad Attack

Michael Schwarz
Moritz Lipp

Who am I?



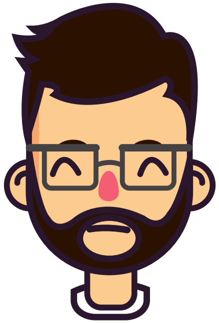
Michael Schwarz

Faculty @ CISPA Helmholtz Center for Information Security

@misc0110

michael.schwarz@cispa.saarland

Who am I?



Moritz Lipp

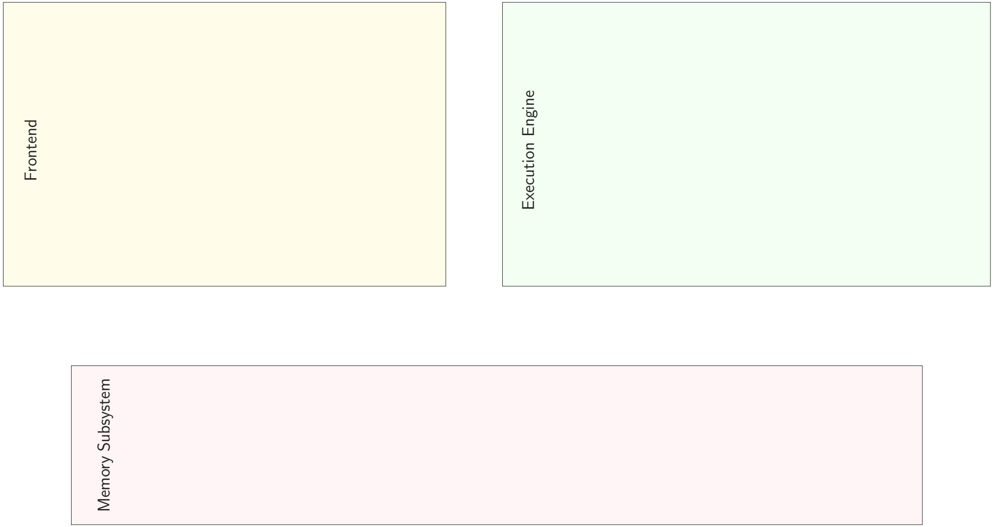
PhD Candidate @ Graz University of Technology

@mlqxyz

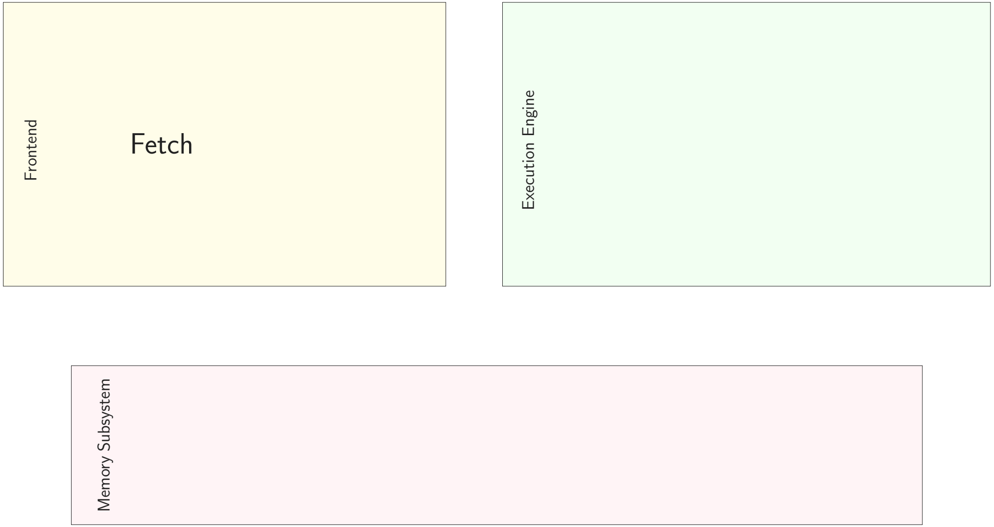
moritz.lipp@iaik.tugraz.at

NEED
BACKGROUND

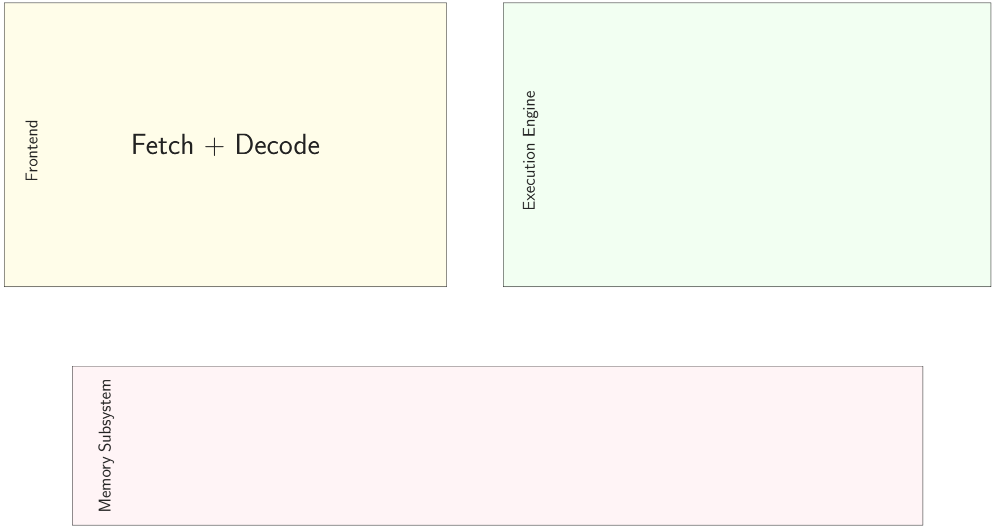
Microarchitecture



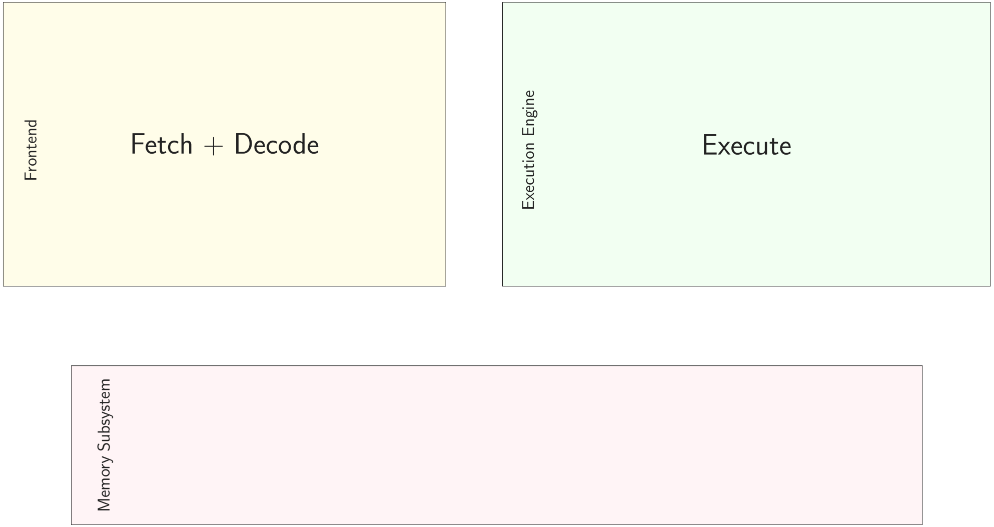
Microarchitecture



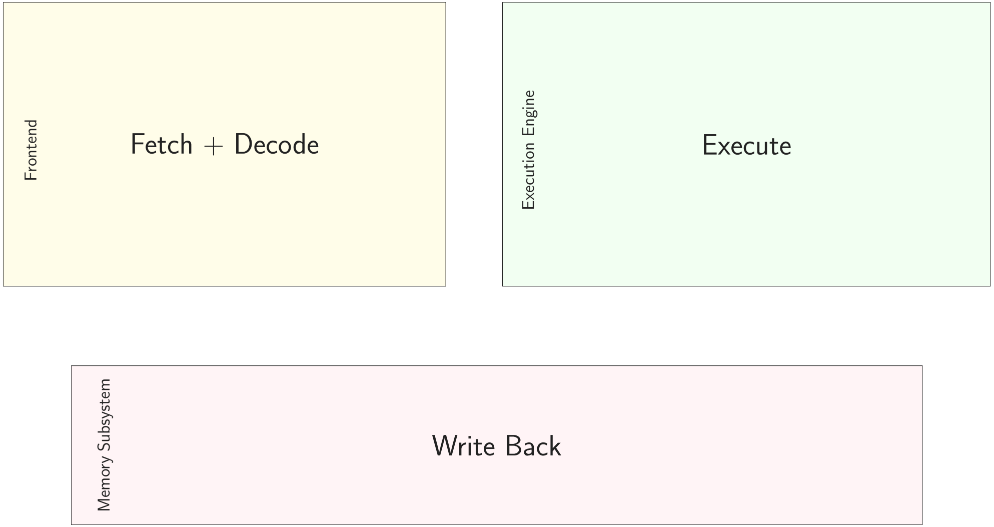
Microarchitecture



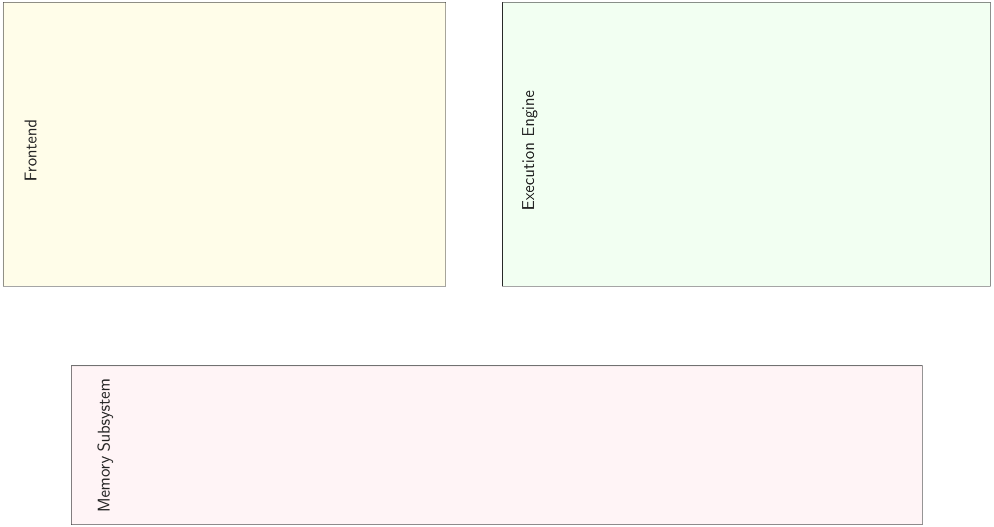
Microarchitecture



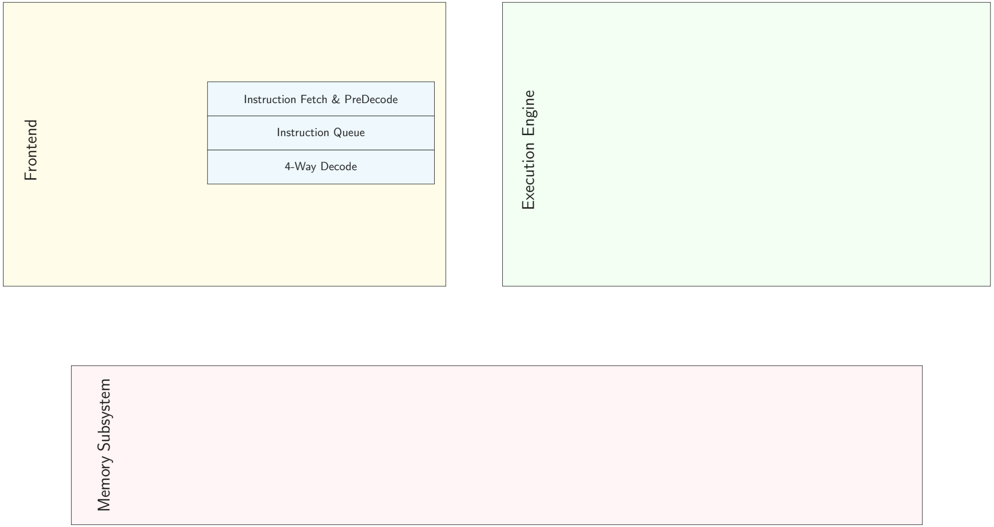
Microarchitecture



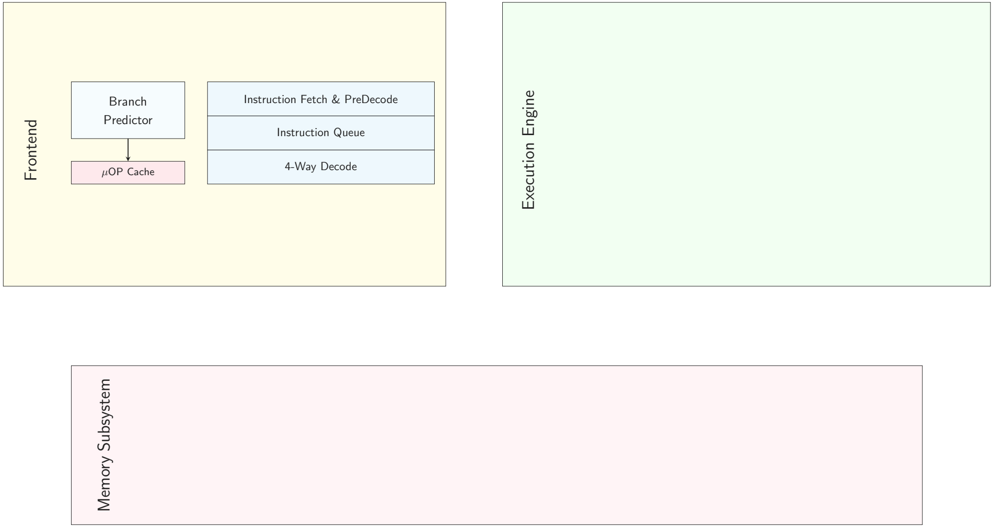
Microarchitecture



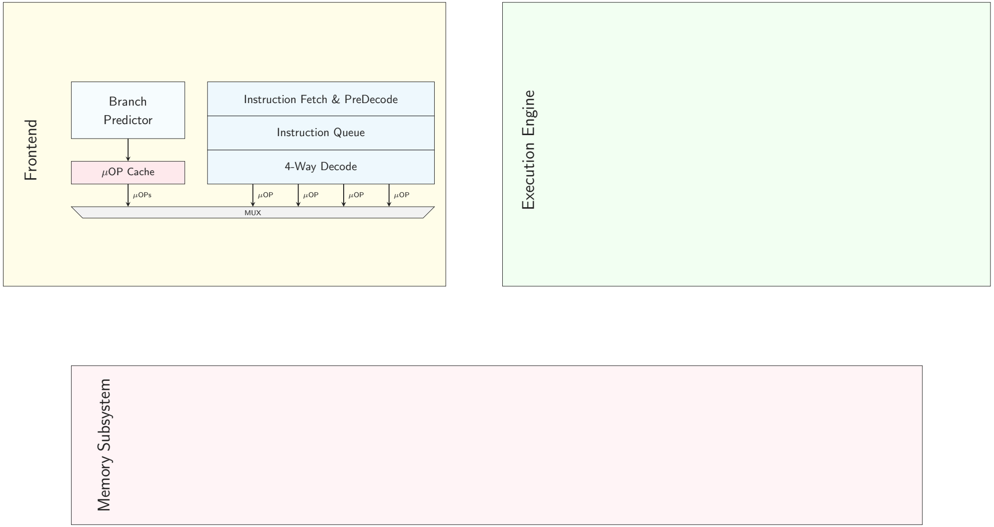
Microarchitecture



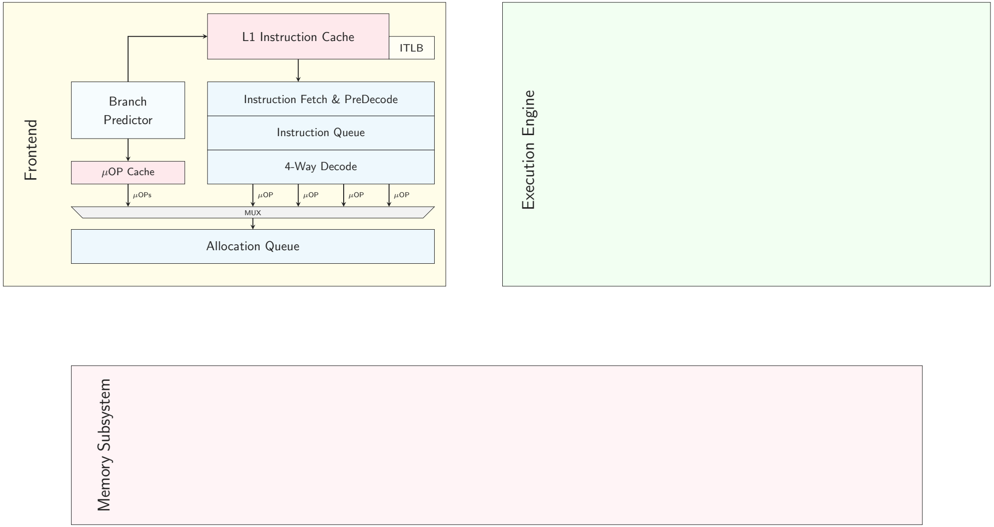
Microarchitecture



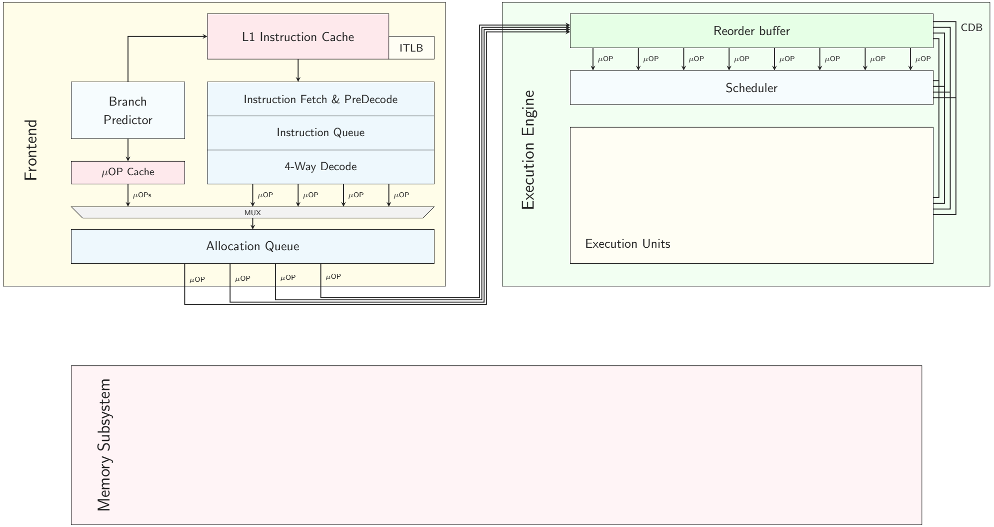
Microarchitecture



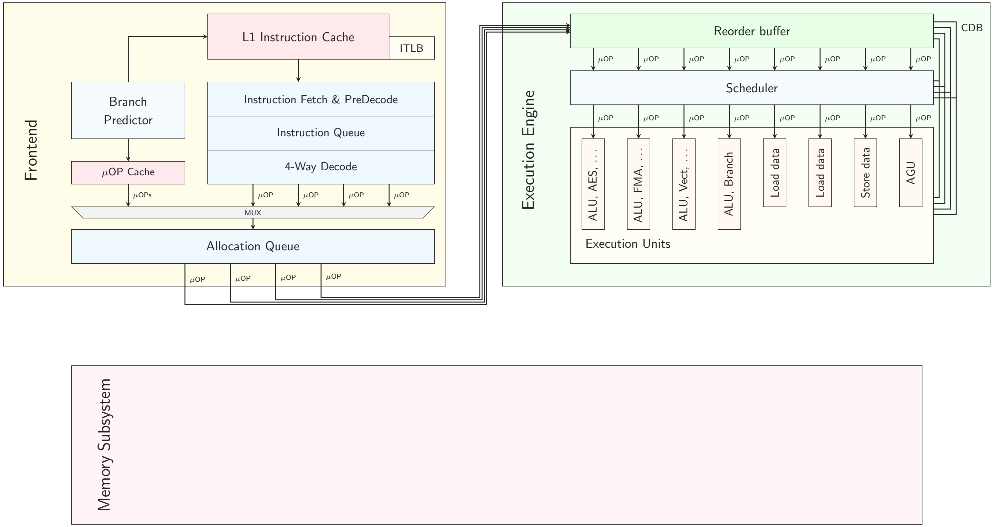
Microarchitecture



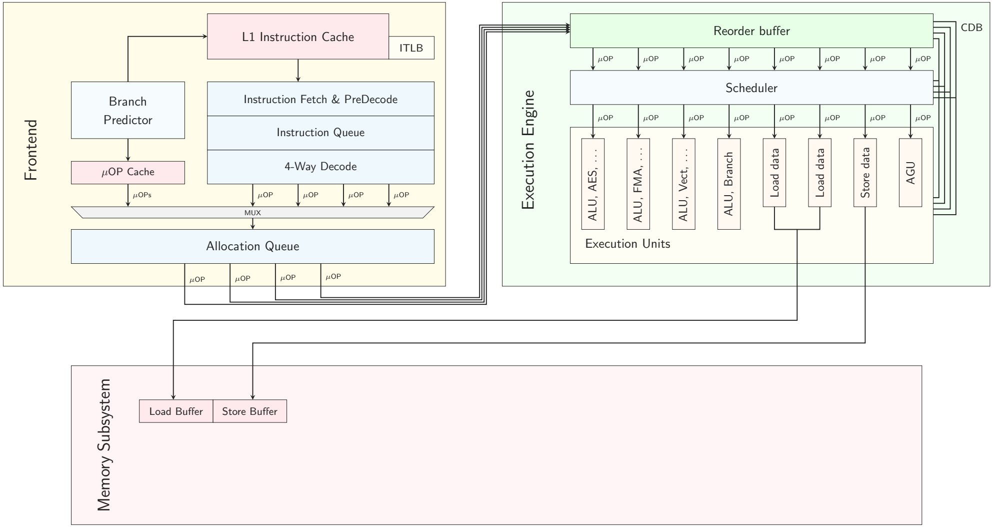
Microarchitecture



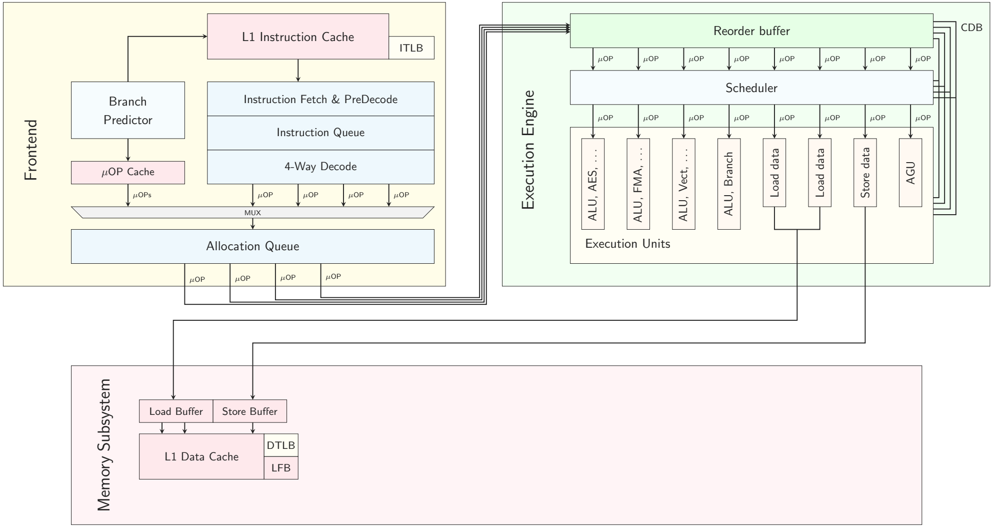
Microarchitecture



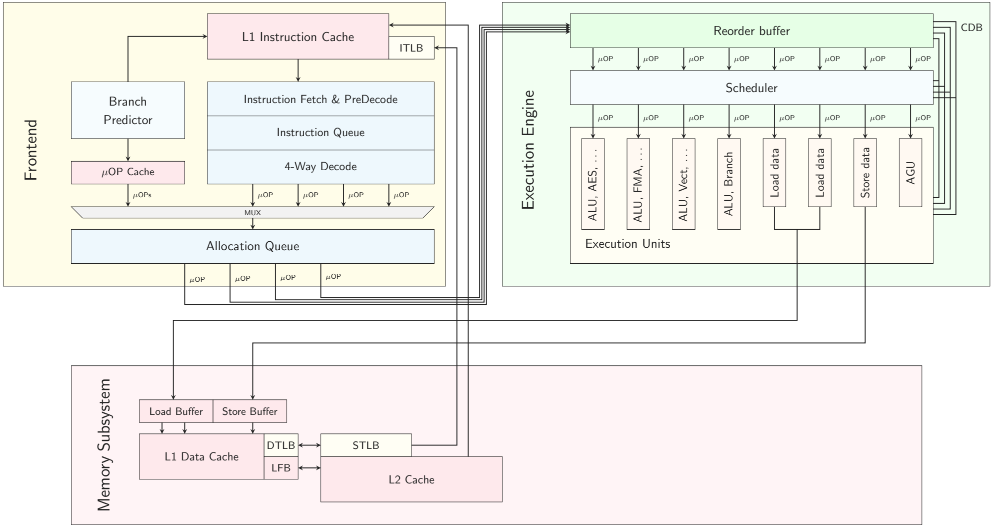
Microarchitecture



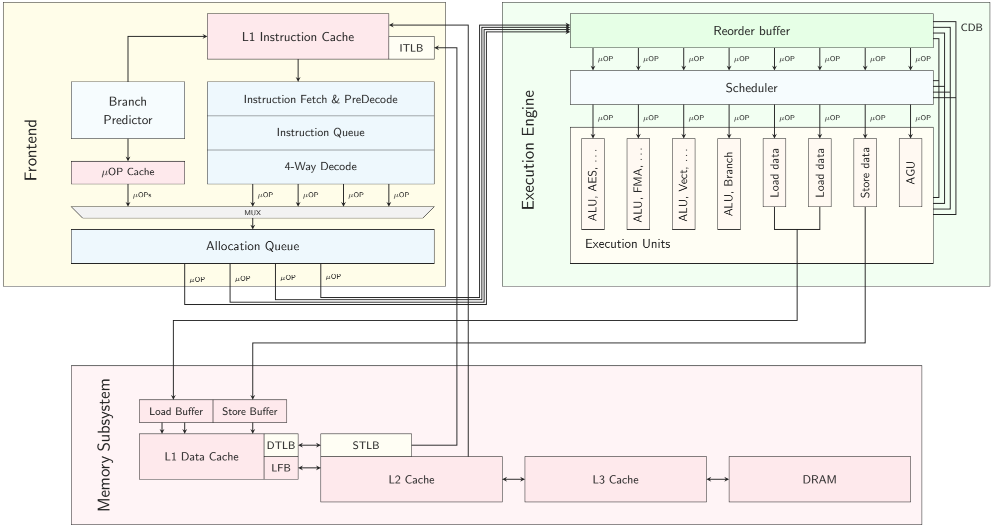
Microarchitecture



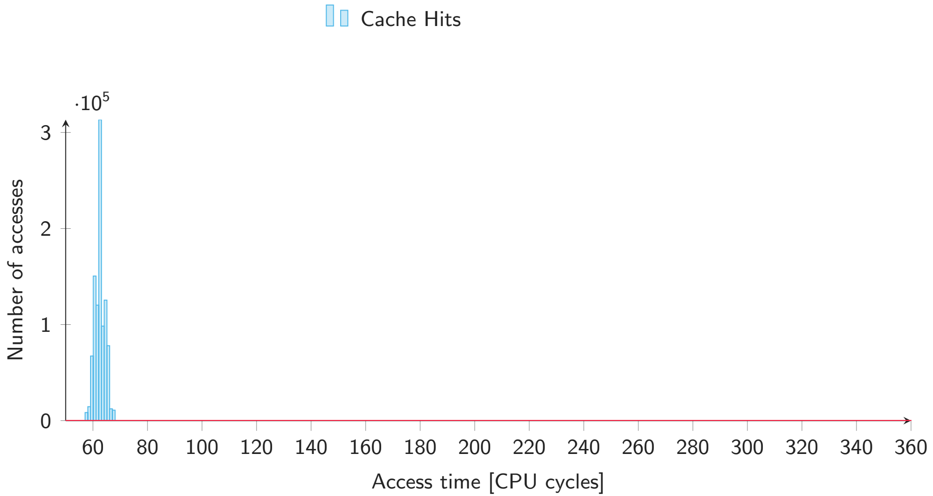
Microarchitecture



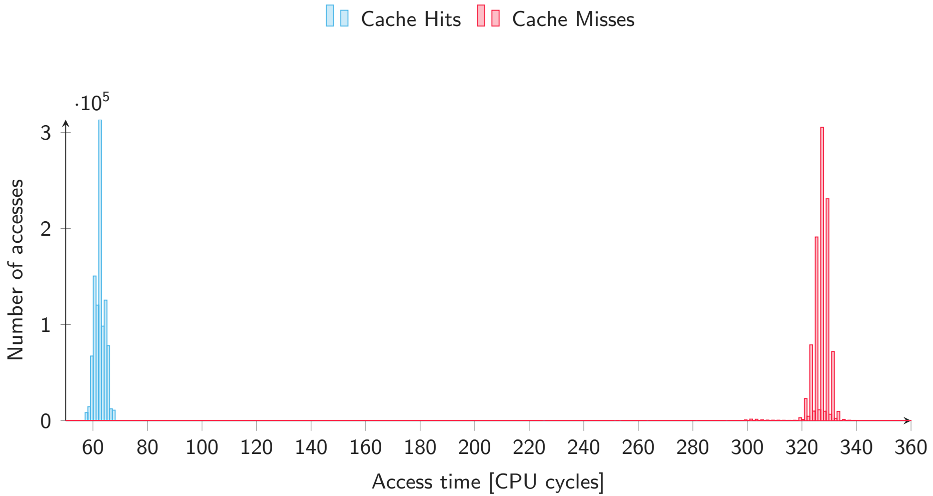
Microarchitecture



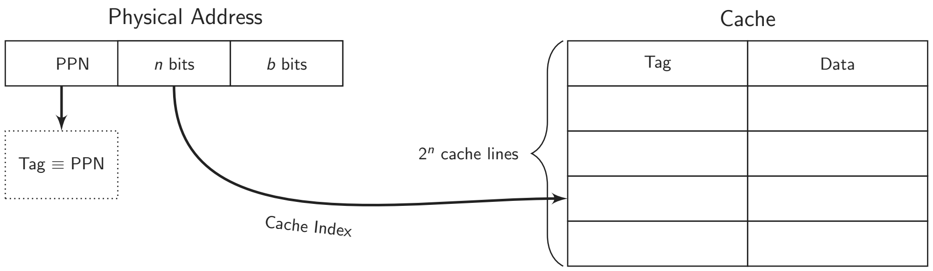
Caching Speeds Up Memory Accesses



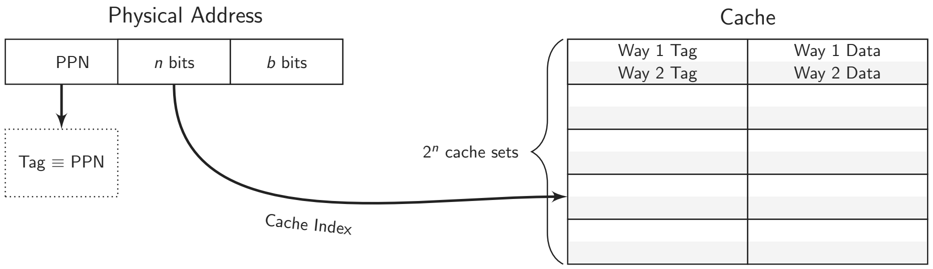
Caching Speeds Up Memory Accesses



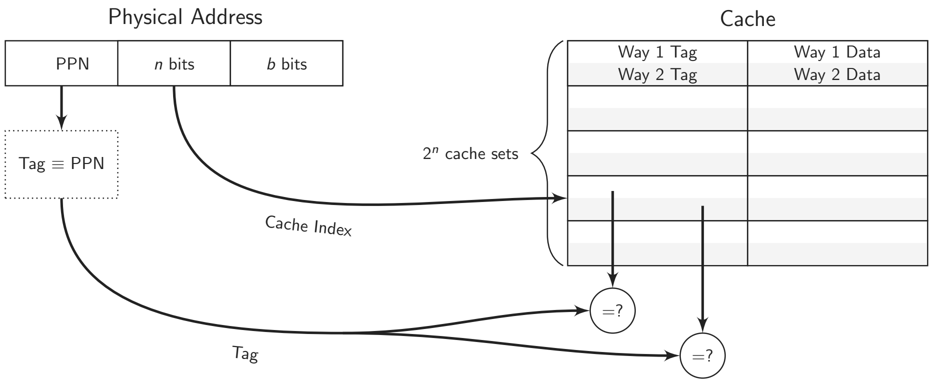
How Do Caches Actually Work?



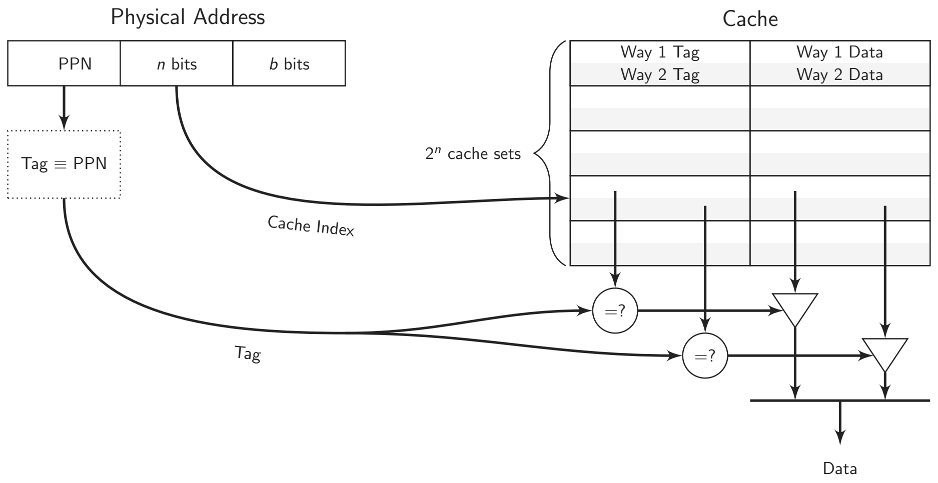
How Do Caches Actually Work?



How Do Caches Actually Work?



How Do Caches Actually Work?



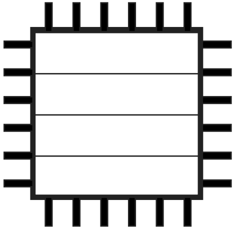
ATTACK

Meltdown

User Memory

	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

```
char value = kernel[0]
```



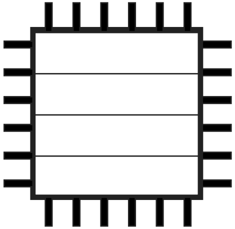
Meltdown

User Memory

	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

```
char value = kernel[0]
```

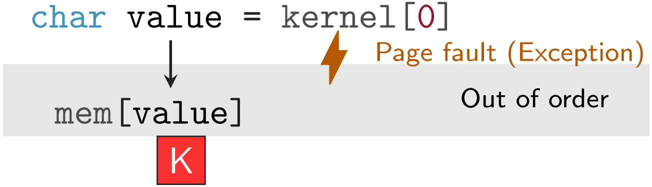
Page fault (Exception)

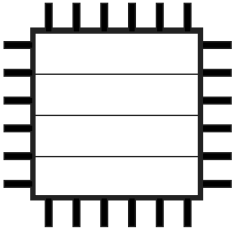


Meltdown

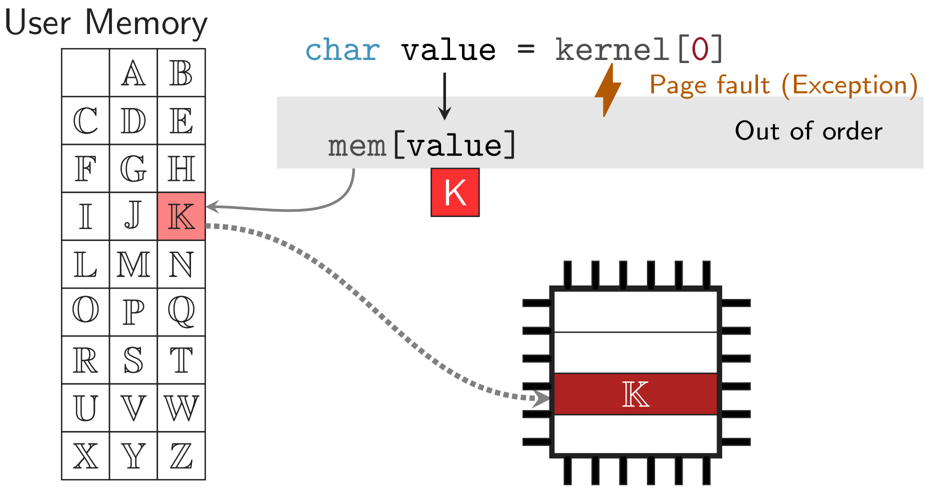
User Memory

	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

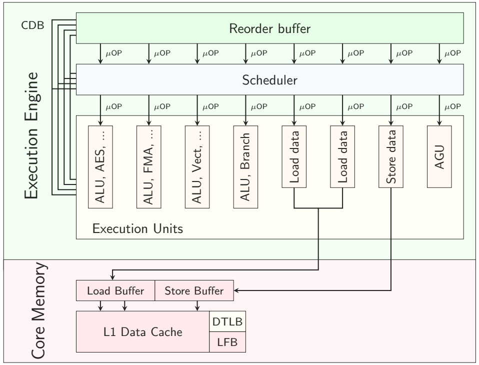




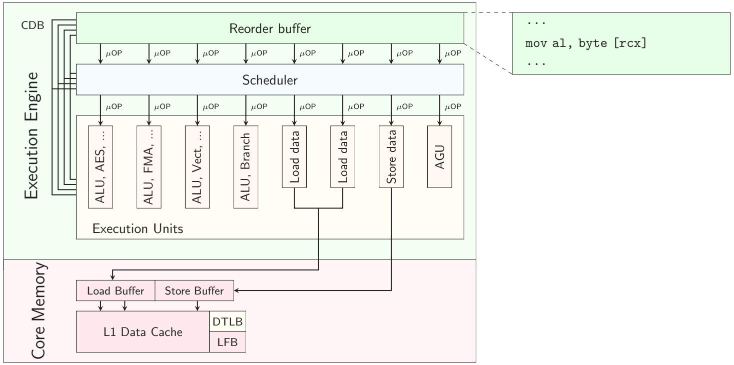
Meltdown


User Memory
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
char value = kernel[0]
Page fault (Exception)
Out of order
mem[value]
K
K

Meltdown in the Microarchitecture

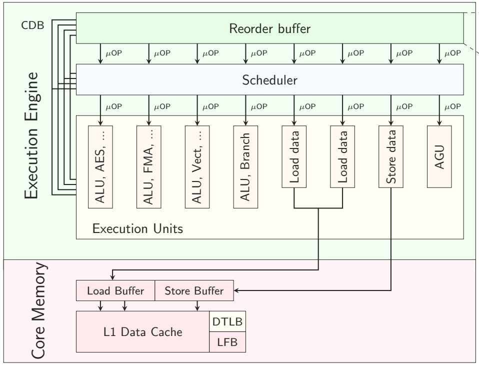


Meltdown in the Microarchitecture



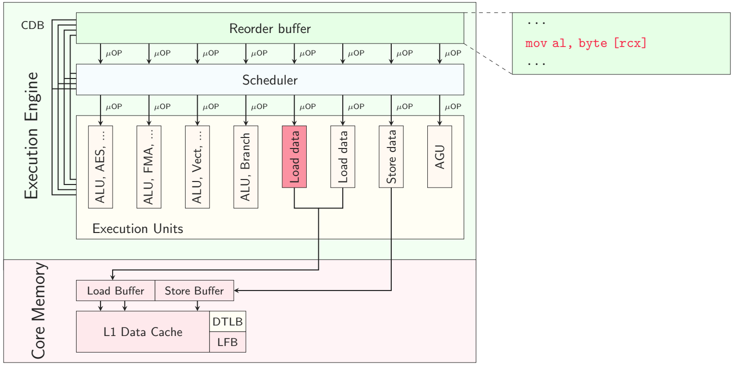
```
...
mov al, byte [rcx]
...
```

Meltdown in the Microarchitecture



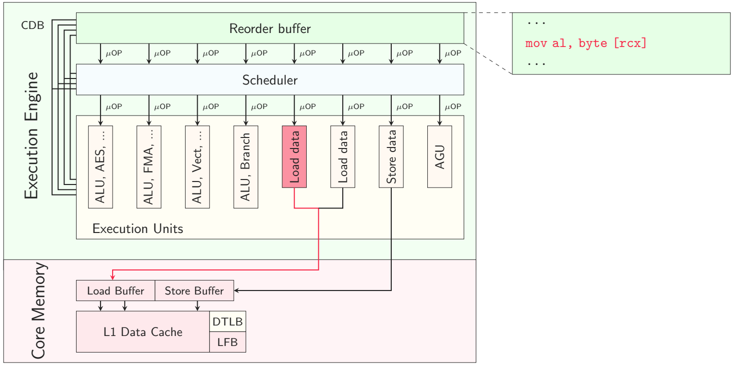
```
...
mov al, byte [rcx]
...
```

Meltdown in the Microarchitecture



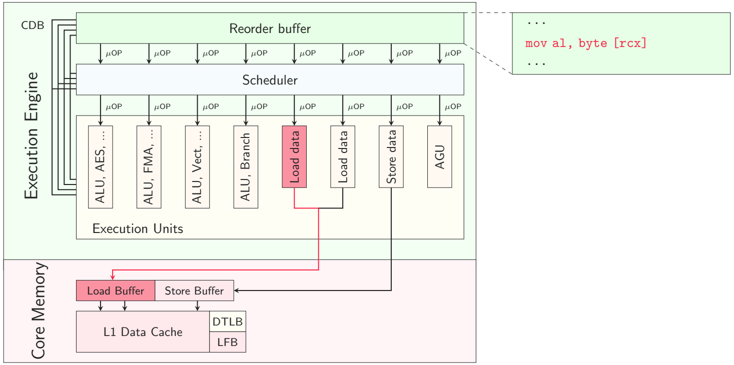
```
...
mov al, byte [rcx]
...
```

Meltdown in the Microarchitecture



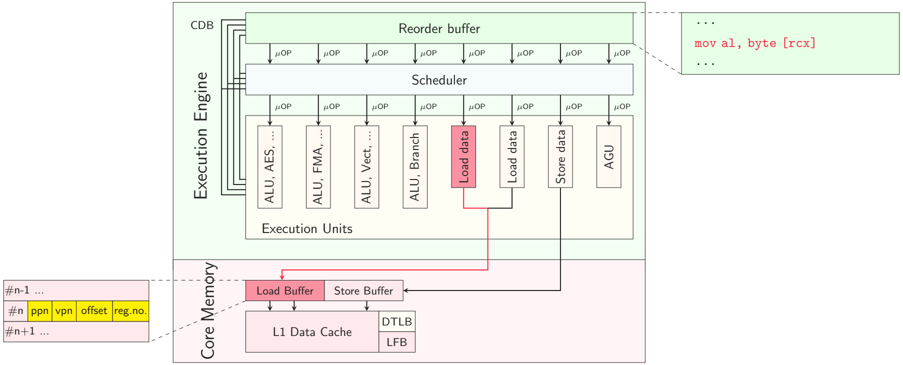
```
...
mov al, byte [rcx]
...
```

Meltdown in the Microarchitecture



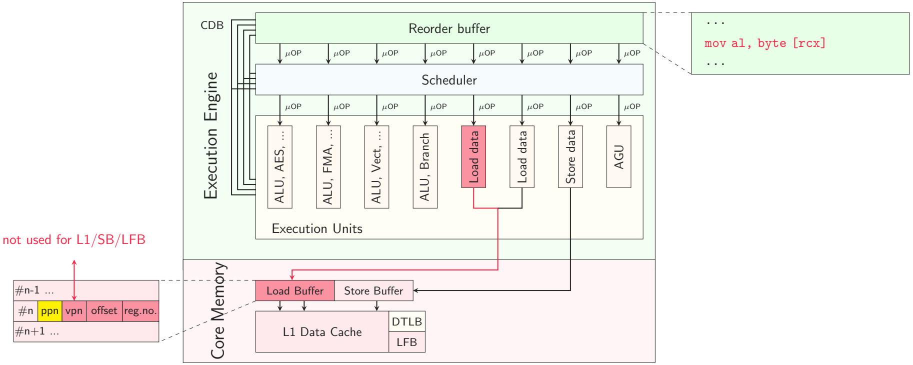
```
...
mov al, byte [rcx]
...
```

Meltdown in the Microarchitecture

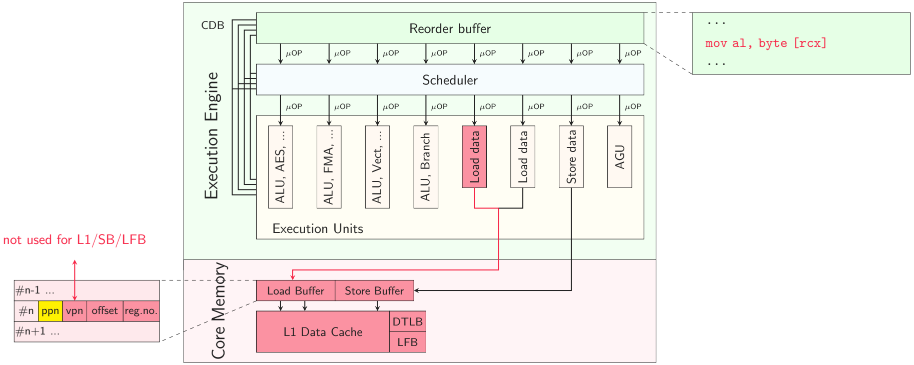


```
...
mov al, byte [rcx]
...
```

Meltdown in the Microarchitecture

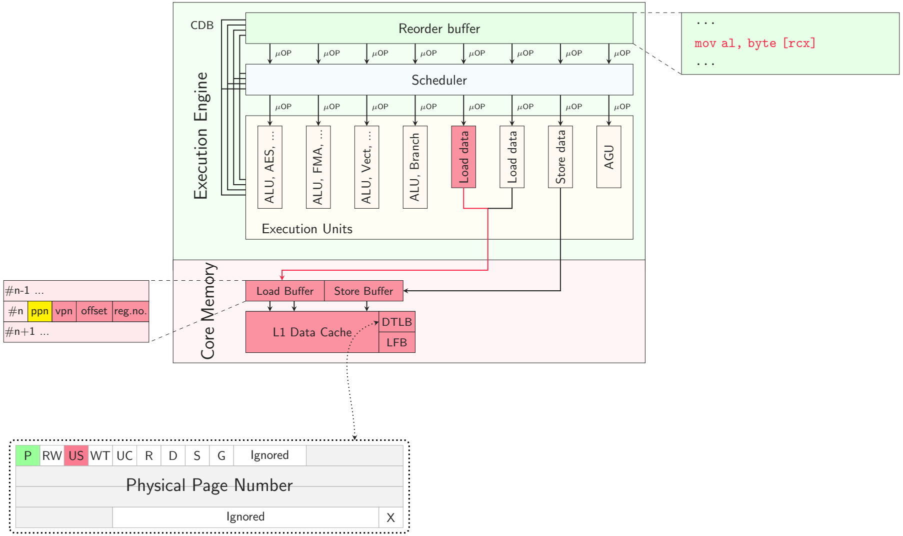


Meltdown in the Microarchitecture



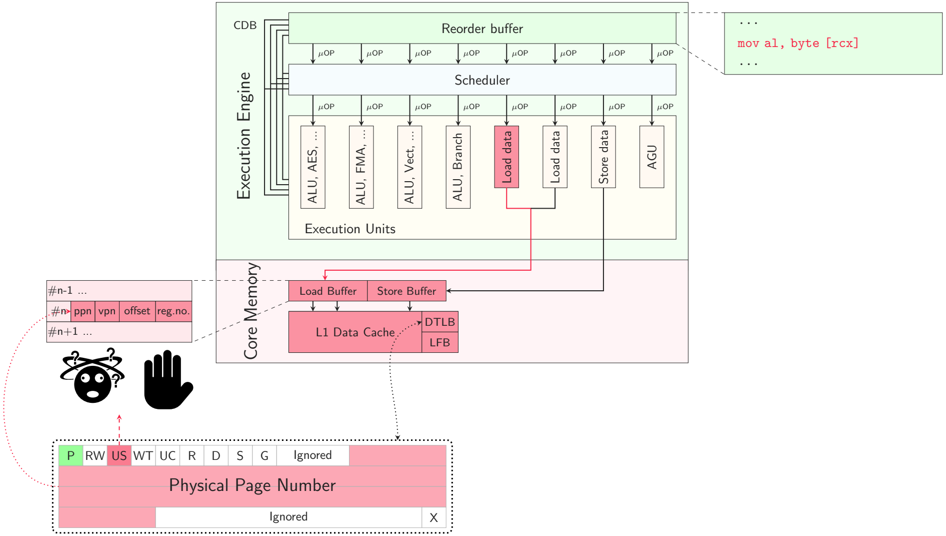
```
...
mov al, byte [rcx]
...
```

Meltdown in the Microarchitecture

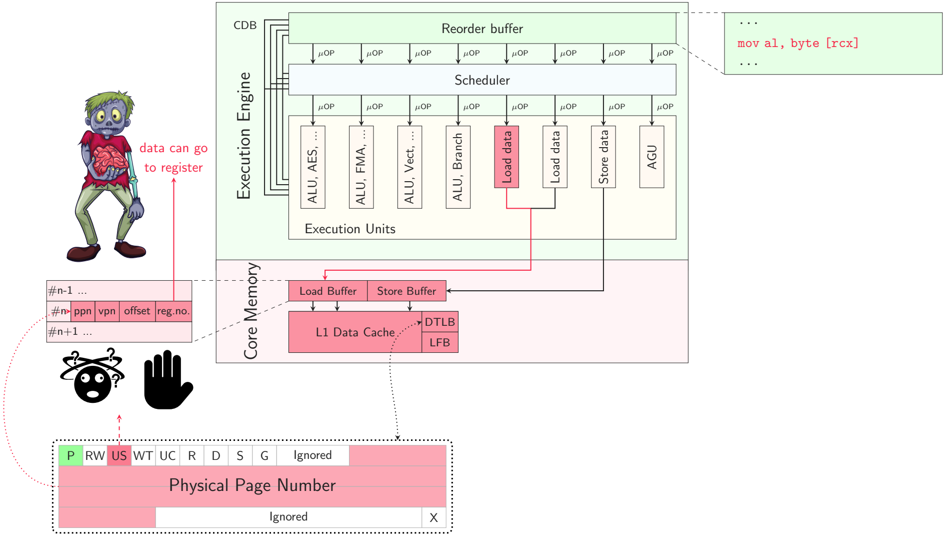


```
...
mov al, byte [rcx]
...
```

Meltdown in the Microarchitecture



Meltdown in the Microarchitecture



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

Performance

Meltdown

Leakage Rate

552.4 kB/s

Performance

Meltdown

Leakage Rate

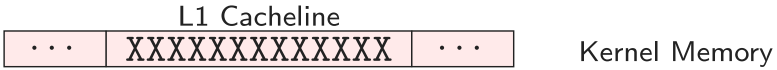
552.4 kB/s

Meltdown

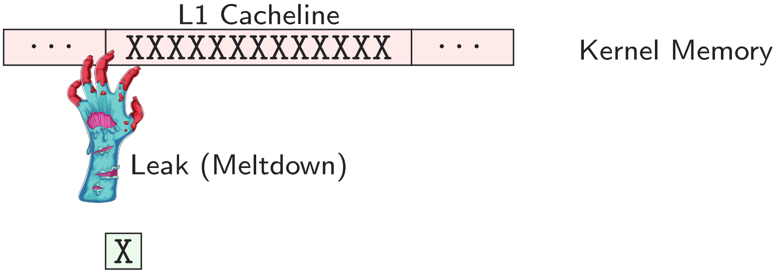
Error Rate

0.003 %

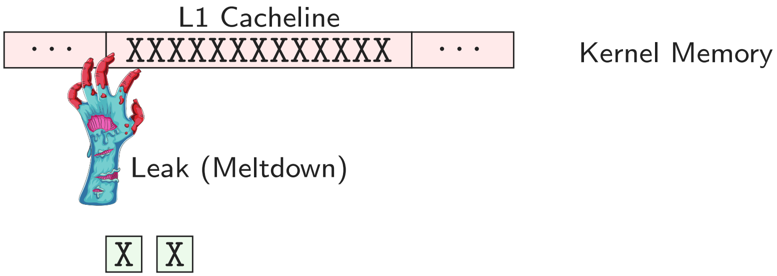
Meltdown Experiment



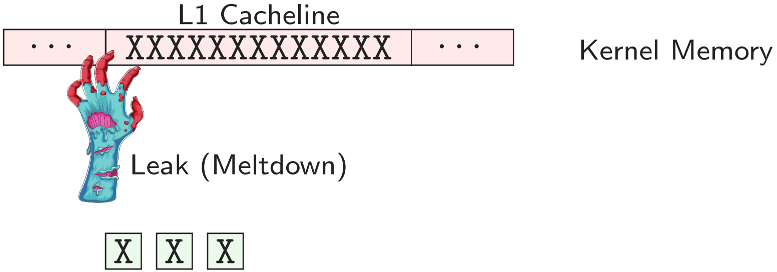
Meltdown Experiment



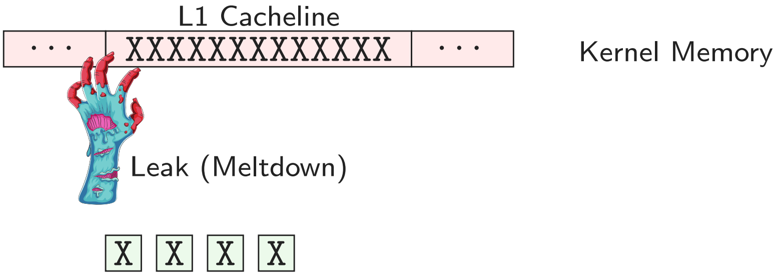
Meltdown Experiment



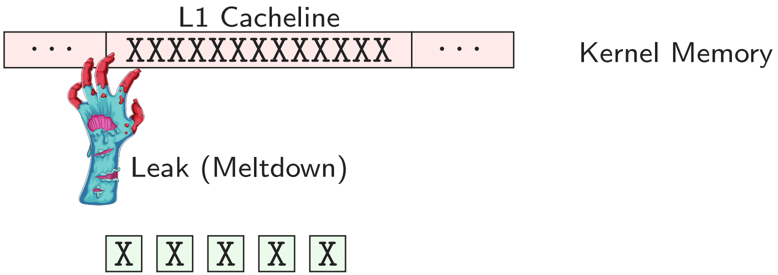
Meltdown Experiment



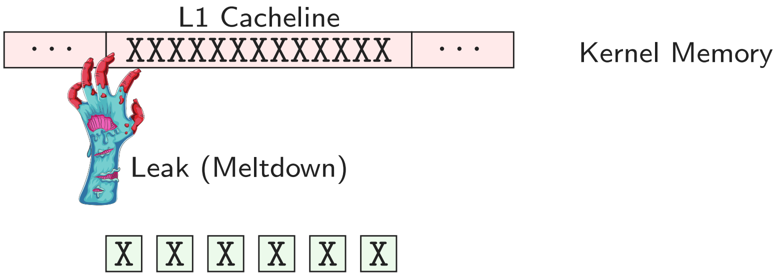
Meltdown Experiment



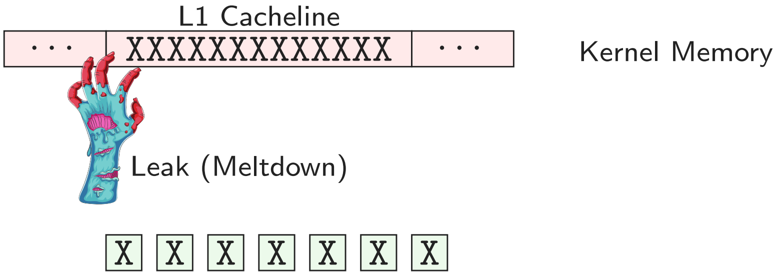
Meltdown Experiment



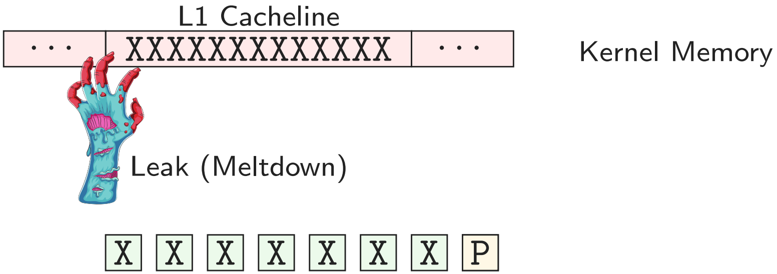
Meltdown Experiment



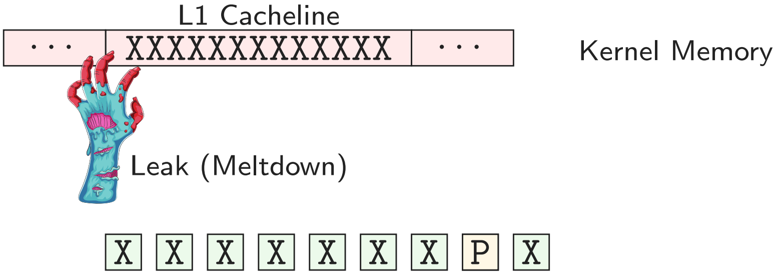
Meltdown Experiment



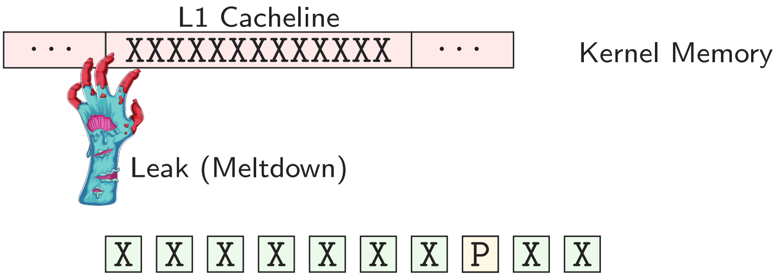
Meltdown Experiment



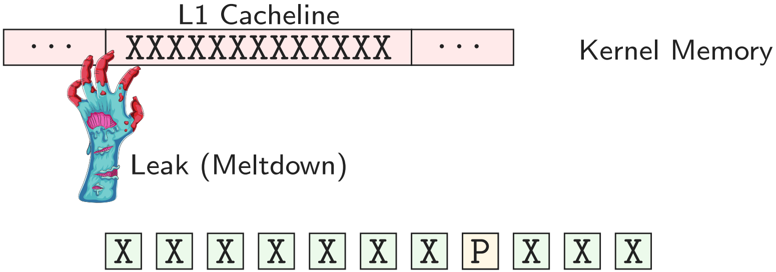
Meltdown Experiment



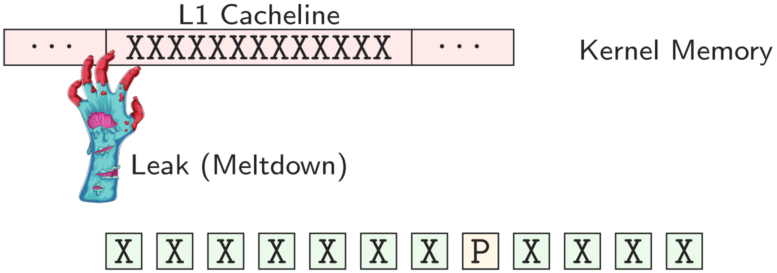
Meltdown Experiment



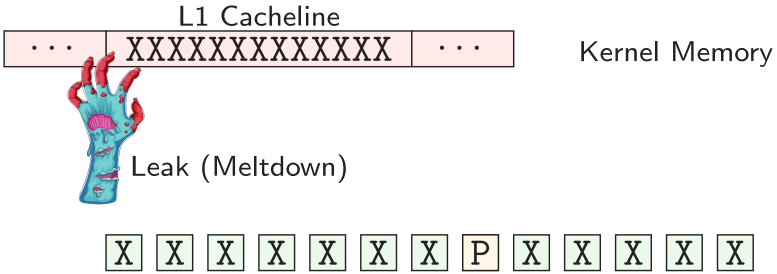
Meltdown Experiment



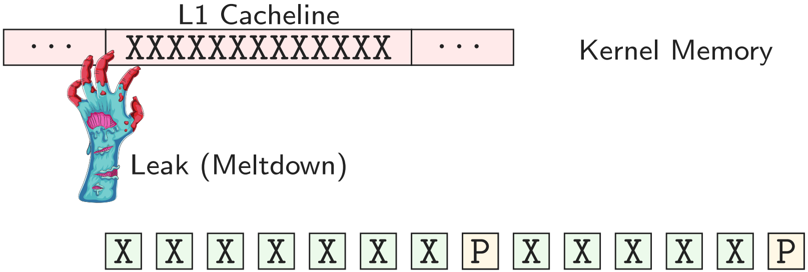
Meltdown Experiment



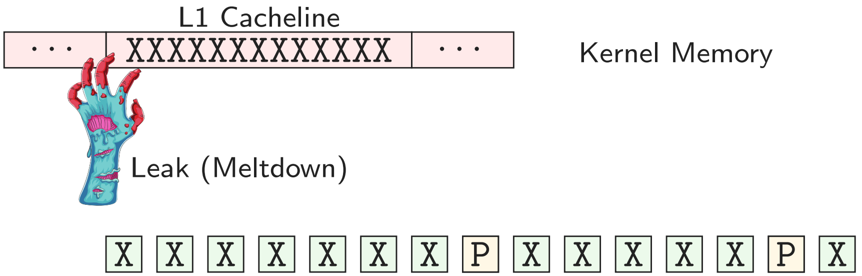
Meltdown Experiment



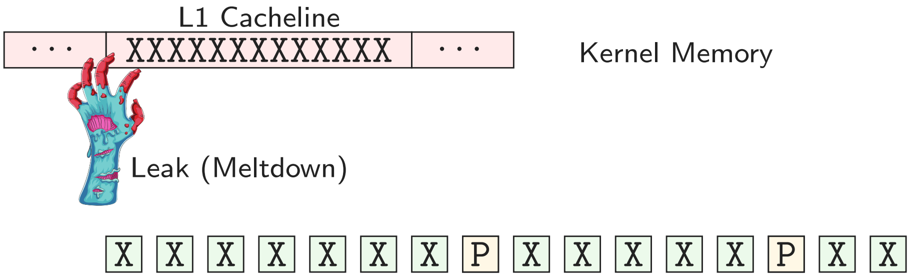
Meltdown Experiment



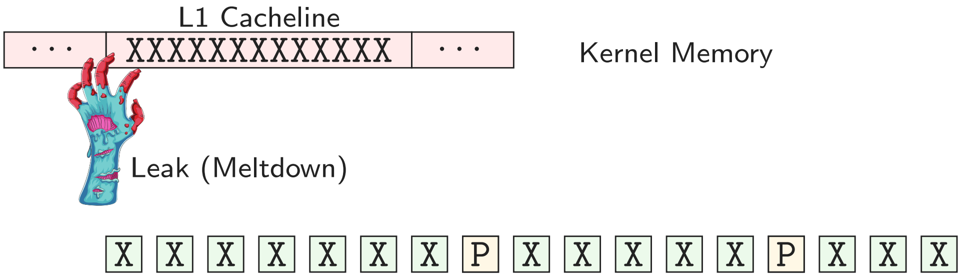
Meltdown Experiment



Meltdown Experiment



Meltdown Experiment



How to get rid of the noise?

There is no noise.

Noise is just someone else's data

Analyse the Noise

Analyse the Noise

Lemma 1: Noise is someone else's data

$$\lim_{\;\to \varnothing} \quad =$$

Analyse the Noise

Lemma 1: Noise is someone else's data

ZombieLoad

Deep Dive: Intel Analysis of Microarchitectural Data Sampling

Fill buffers may retain stale data from prior memory requests until a new memory request overwrites the fill buffer.

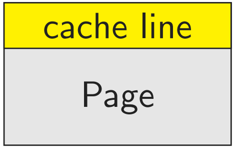
Deep Dive: Intel Analysis of Microarchitectural Data Sampling

Fill buffers may retain stale data from prior memory requests until a new memory request overwrites the fill buffer. Under certain conditions, the fill buffer may speculatively forward data, including stale data,

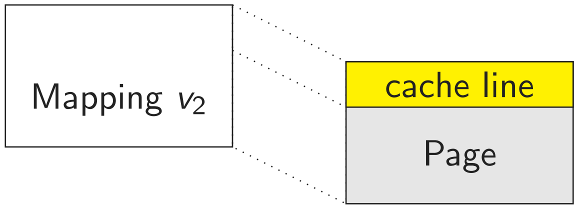
Deep Dive: Intel Analysis of Microarchitectural Data Sampling

Fill buffers may retain stale data from prior memory requests until a new memory request overwrites the fill buffer. Under certain conditions, the fill buffer may speculatively forward data, including stale data, to a load operation that will cause a fault/assist.

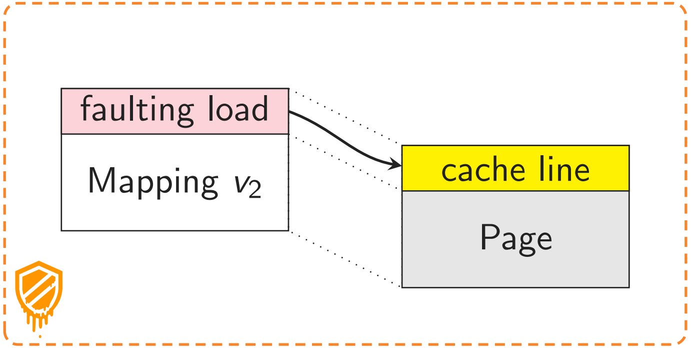
ZombieLoad Cache-line Conflicts



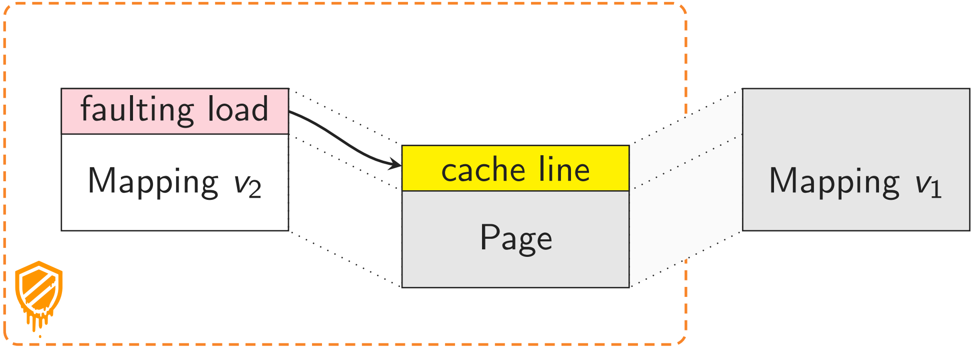
ZombieLoad Cache-line Conflicts



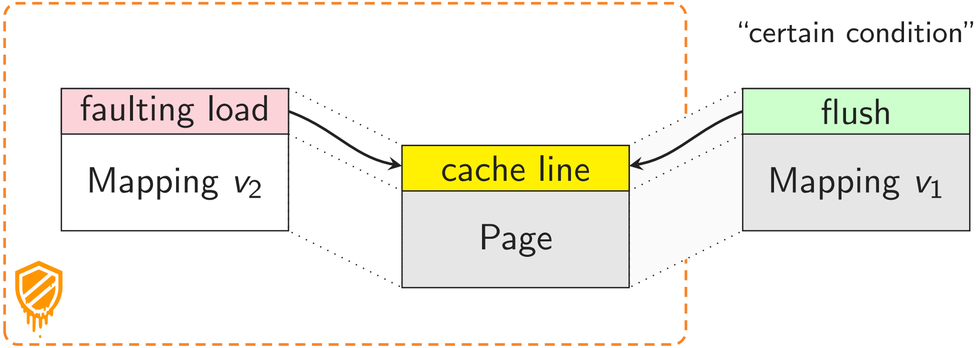
ZombieLoad Cache-line Conflicts



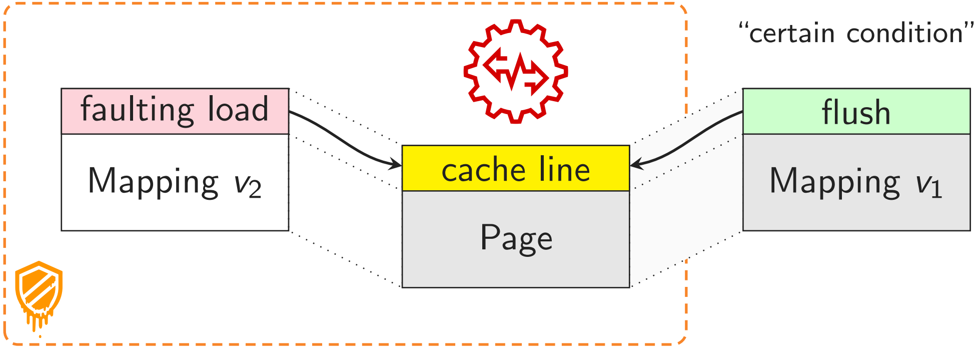
ZombieLoad Cache-line Conflicts



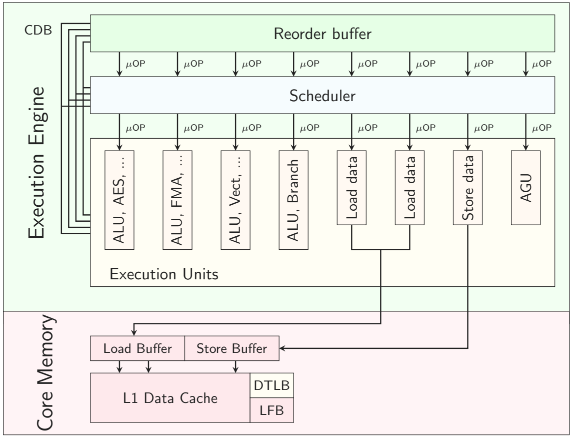
ZombieLoad Cache-line Conflicts



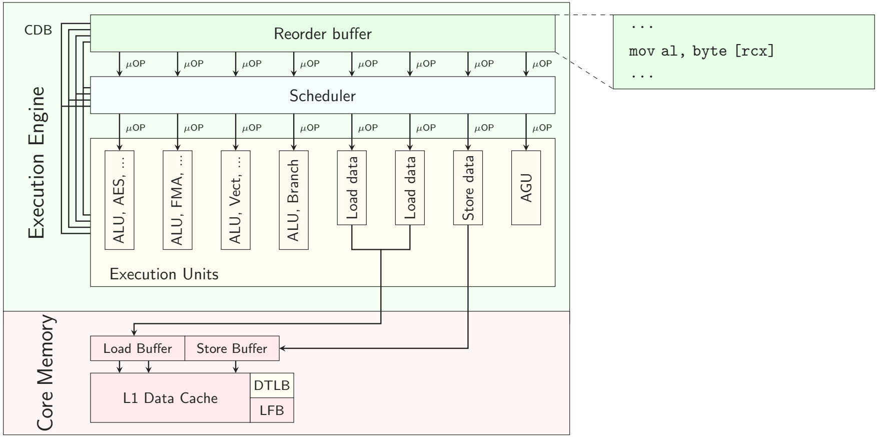
ZombieLoad Cache-line Conflicts



Complex Load Situations

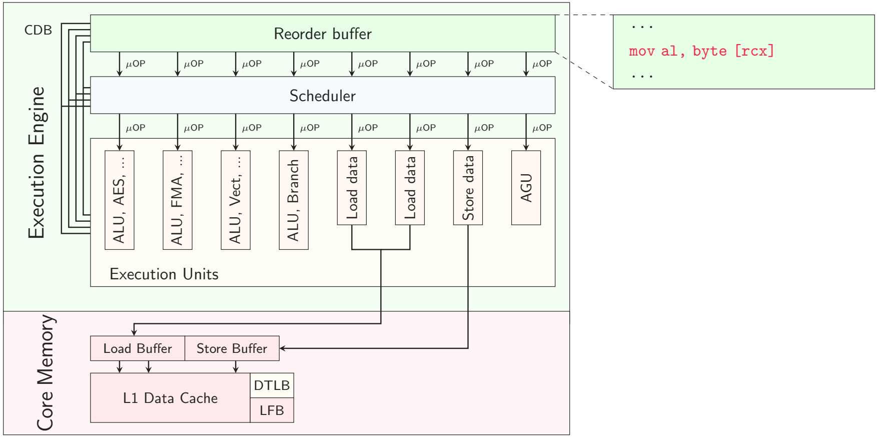


Complex Load Situations



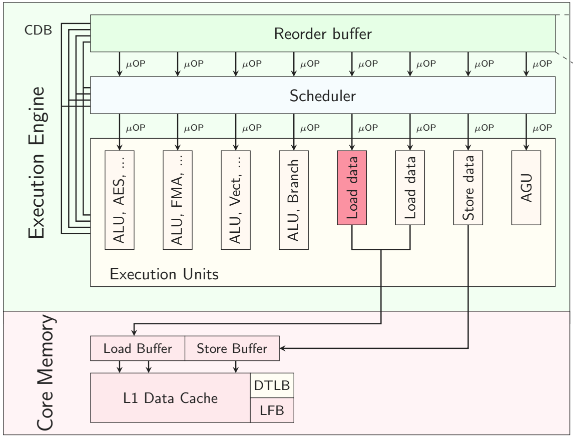
```
...
mov al, byte [rcx]
...
```

Complex Load Situations



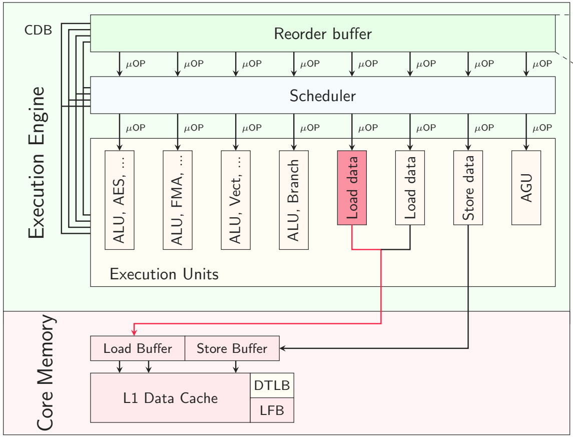
```
...
mov al, byte [rcx]
...
```

Complex Load Situations



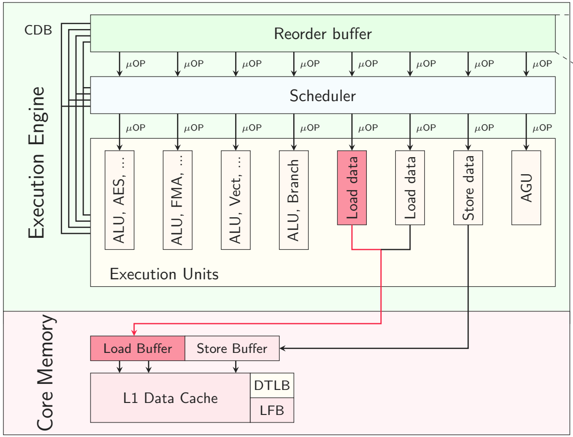
```
...
mov al, byte [rcx]
...
```

Complex Load Situations



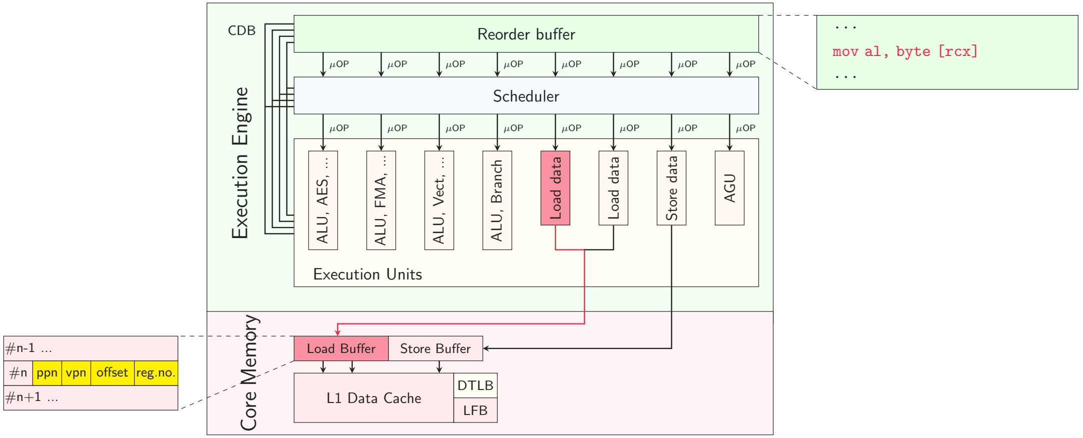
```
...
mov al, byte [rcx]
...
```

Complex Load Situations



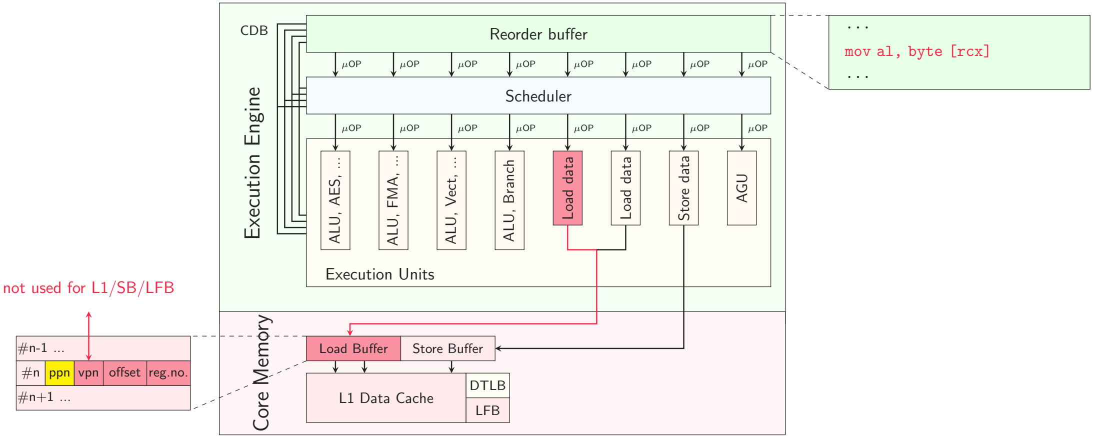
```
...
mov al, byte [rcx]
...
```

Complex Load Situations



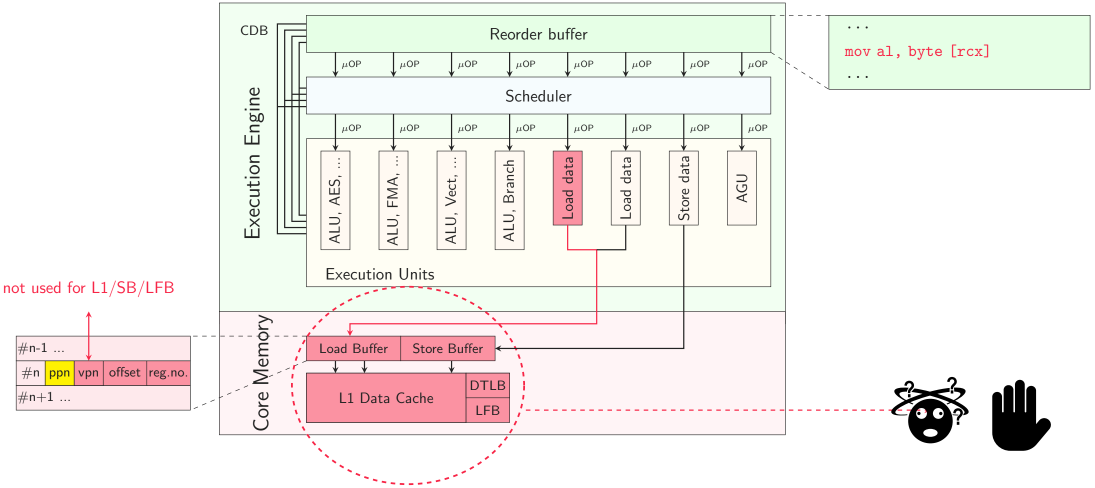
```
...
mov al, byte [rcx]
...
```

Complex Load Situations

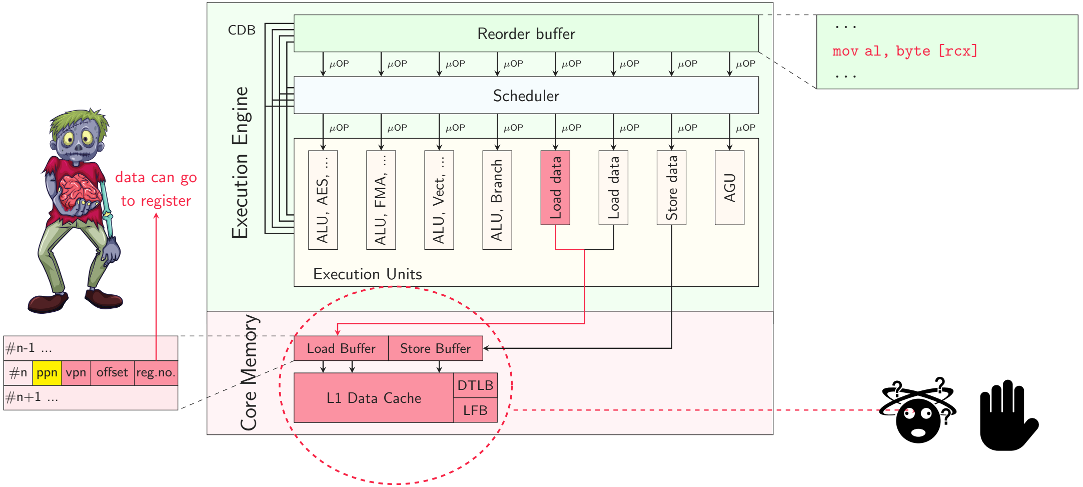


```
...
mov al, byte [rcx]
...
```

Complex Load Situations



Complex Load Situations

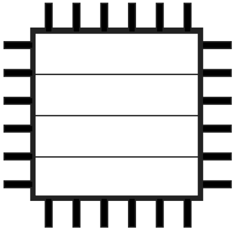


Data Encoding

User Memory

	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

```
char value = faulting[0]
```



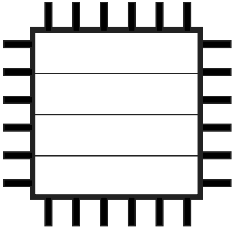
Data Encoding

User Memory

	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

```
char value = faulting[0]
```

Fault

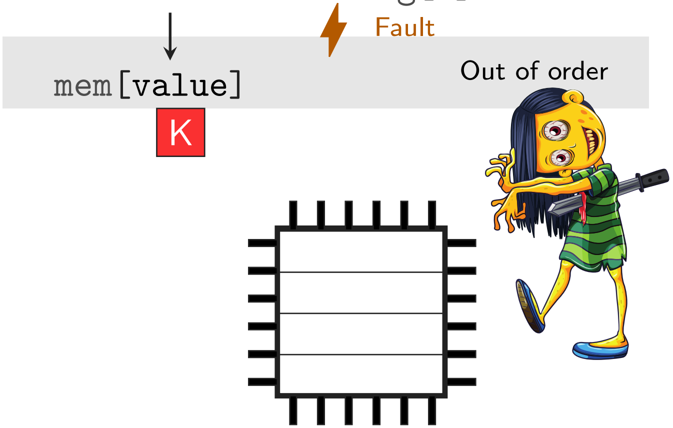


Data Encoding

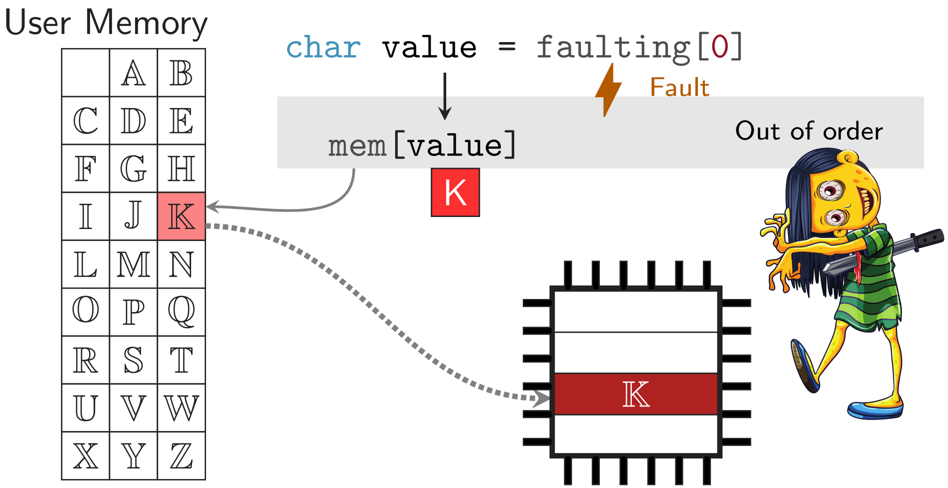
User Memory

	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

```
char value = faulting[0]
```

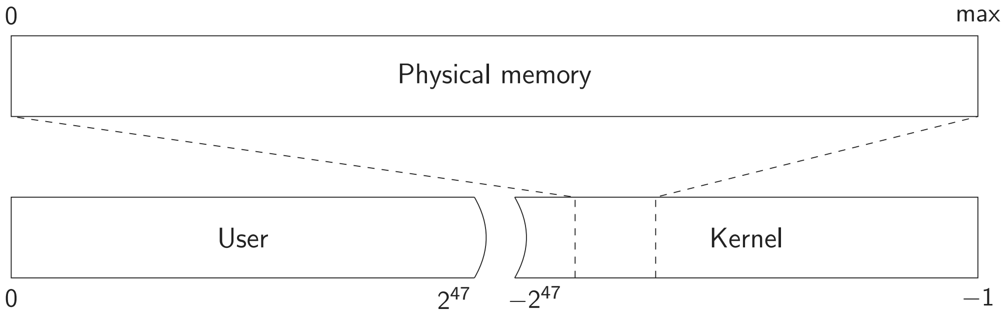


Data Encoding

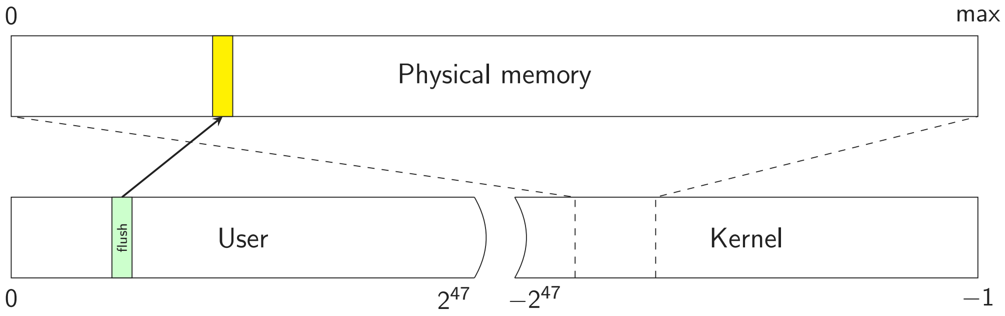


```
char value = faulting[0]
mem[value]
```

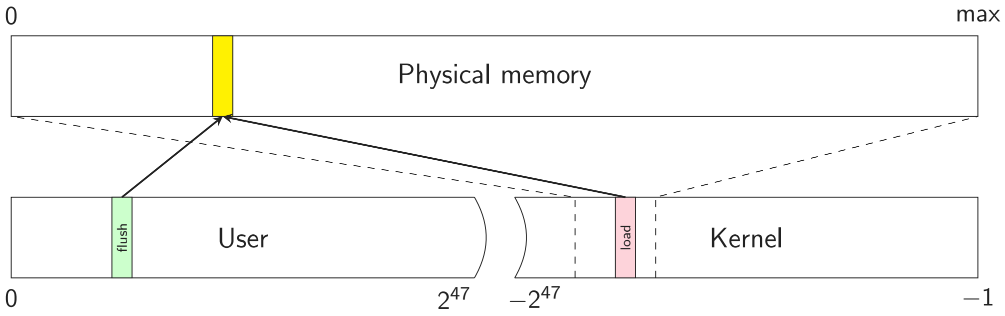
ZombieLoad Variant 1



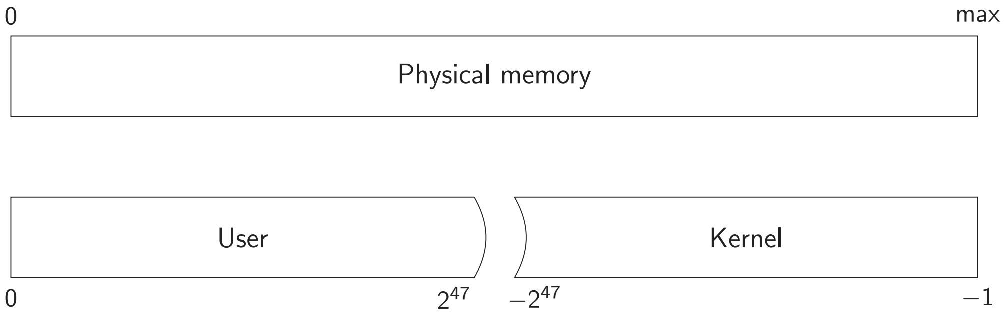
ZombieLoad Variant 1



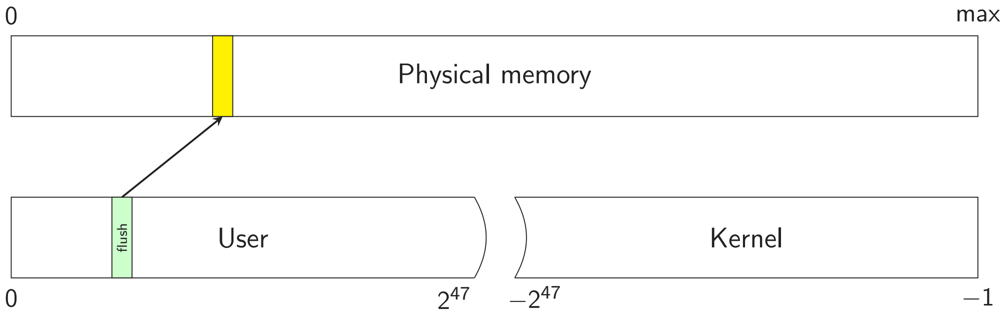
ZombieLoad Variant 1



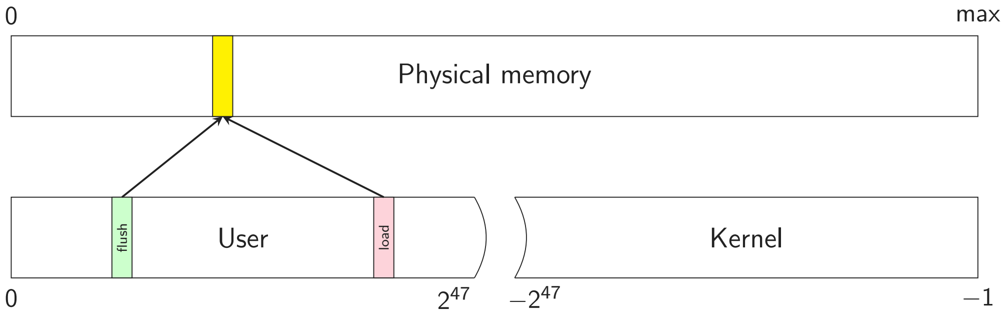
ZombieLoad Variant 3



ZombieLoad Variant 3



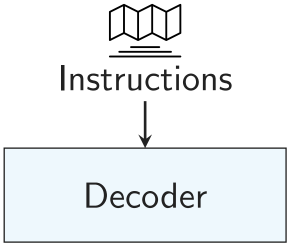
ZombieLoad Variant 3



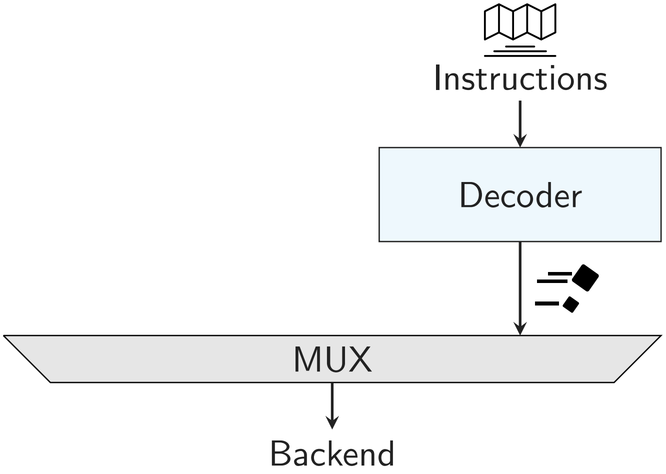
Microcode Assist (Variant 3)

Instructions

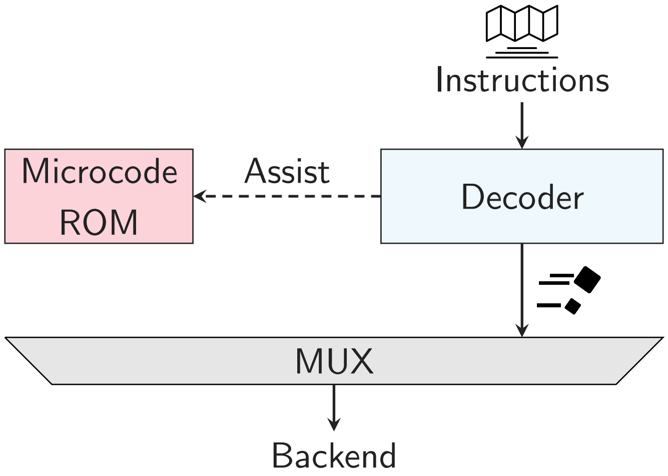
Microcode Assist (Variant 3)



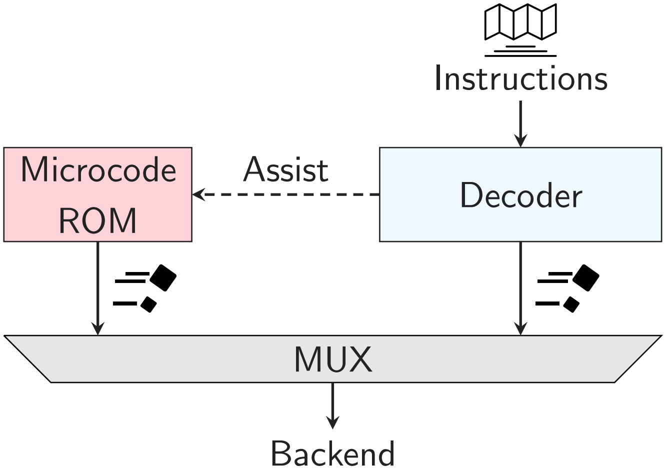
Microcode Assist (Variant 3)



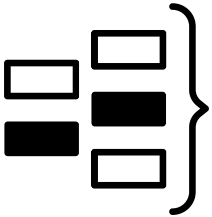
Microcode Assist (Variant 3)



Microcode Assist (Variant 3)

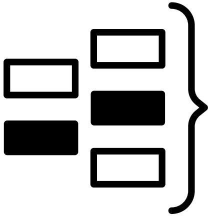


Microcode Assist (Variant 3)



- Microcode assist handles rare cases

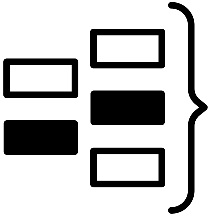
Microcode Assist (Variant 3)



- Microcode assist handles rare cases

→ Microarchitectural fault

Microcode Assist (Variant 3)

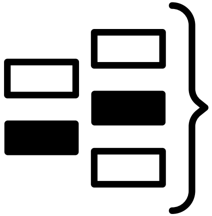


- Microcode assist handles rare cases

→ Microarchitectural fault

- Setting accessed/dirty bit in page table

Microcode Assist (Variant 3)



- Microcode assist handles rare cases

→ Microarchitectural fault

- Setting accessed/dirty bit in page table

→ Regularly reset on Windows

Attack Targets

- Leak data on same and sibling hyperthread

Attack Targets

- Leak data on same and sibling hyperthread

Applications

Attack Targets

- Leak data on same and sibling hyperthread

Applications

Operating System

Attack Targets

- Leak data on same and sibling hyperthread

Applications

Operating System

SGX Enclave

Attack Targets

- Leak data on same and sibling hyperthread

Applications

Operating System

SGX Enclave

Virtual Machine

Attack Targets

- Leak data on same and sibling hyperthread

Applications

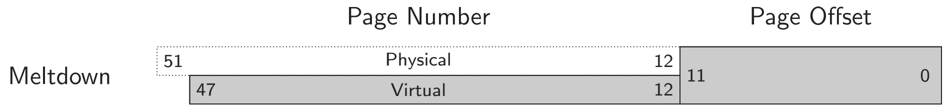
Operating System

SGX Enclave

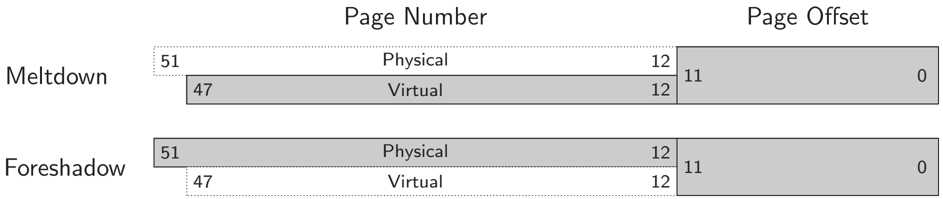
Virtual Machine

Hypervisor

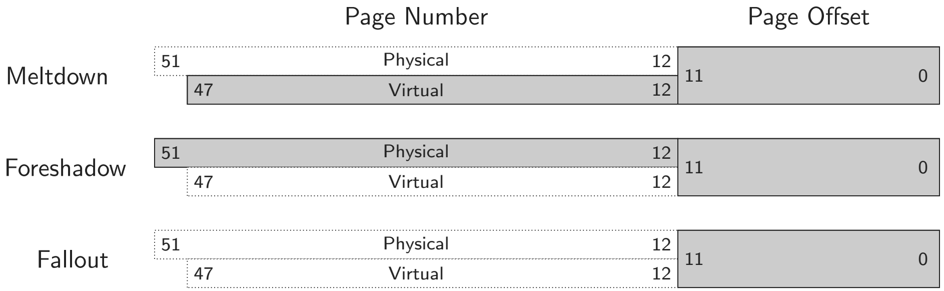
Control


Page Number
Page Offset
Meltdown
51
Physical
12
47
Virtual
12
11
0

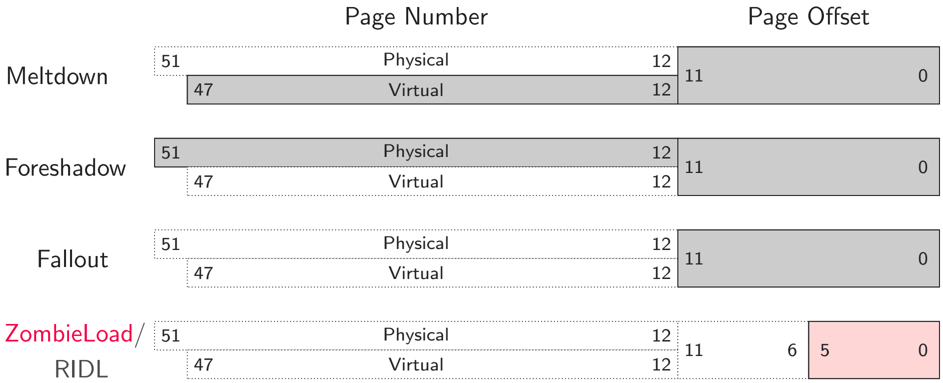
Control


Page Number
Page Offset
Meltdown
51
Physical
12
47
Virtual
12
11
0
Foreshadow
51
Physical
12
47
Virtual
12
11
0

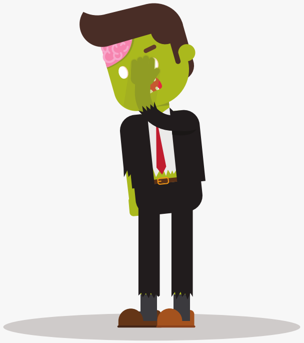
Control


Page Number
Page Offset
Meltdown
51
Physical
12
47
Virtual
12
11
0
Foreshadow
51
Physical
12
47
Virtual
12
11
0
Fallout
51
Physical
12
47
Virtual
12
11
0

Control


Page Number
Page Offset
Meltdown
51 Physical 12
47 Virtual 12
11 0
Foreshadow
51 Physical 12
47 Virtual 12
11 0
Fallout
51 Physical 12
47 Virtual 12
11 0
ZombieLoad / RIDL
51 Physical 12
47 Virtual 12
11 6
5 0

IMPOSSIBLE



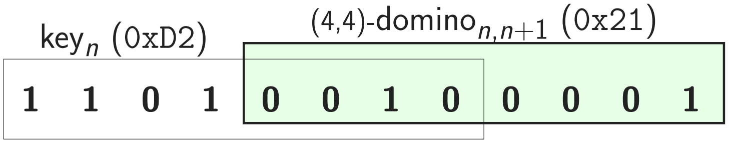
IMPOSSIBLE

Control - Domino Attack

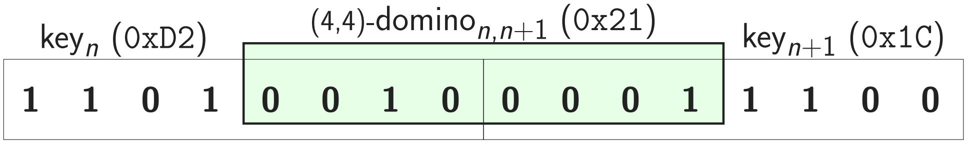
key$_n$ (0xD2)

$$\boxed{1\quad 1\quad 0\quad 1\quad 0\quad 0\quad 1\quad 0}$$

Control - Domino Attack



Control - Domino Attack



DEMO

Attacker Models

Variant 1

Kernel Mapping

	🪟	🐧
👤	○	◐
⚙	●	●

Variant 3

Microcode-Assisted Page-Table Walk

	🪟	🐧
👤	●	○
⚙	●	●

● works ○ does not work ◐ can be prevented

Performance

Variant 1

Kernel Mapping

5.30 kB/s

Performance

Variant 1

Kernel Mapping

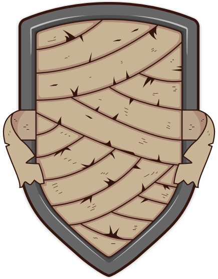
5.30 kB/s

Variant 3

Microcode-Assisted Page-Table Walk

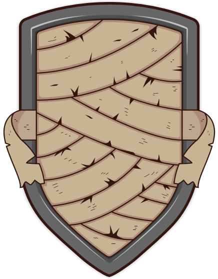
7.73 kB/s

Intel Mitigations



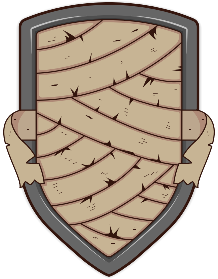
- Disable hyperthreading or group scheduling

Intel Mitigations



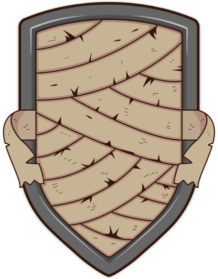
- Disable hyperthreading or group scheduling
- Overwrite microarchitectural buffers

Intel Mitigations



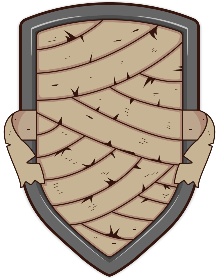
- Disable hyperthreading or group scheduling
- Overwrite microarchitectural buffers
    - `VERW` instruction (microcode update)

Intel Mitigations



- Disable hyperthreading or group scheduling
- Overwrite microarchitectural buffers
  - `VERW` instruction (microcode update)
  - Software sequences

Intel Mitigations

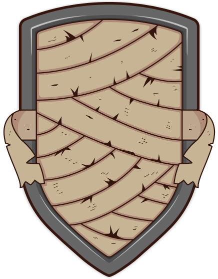


- Disable hyperthreading or group scheduling
- Overwrite microarchitectural buffers
  - `VERW` instruction (microcode update)
  - Software sequences
- New CPUs which are not affected

CPU	Meltdown	Foreshadow	RIDL	Fallout	MLPDS	MDSUM
8th/9th gen. Intel Core Coffee Lake	⚒	⚒	⚒	⚒	⚒	⚒
Intel Xeon Cascade Lake	⚒	⚒	⚒	⚒	⚒	⚒

`https://www.intel.com/content/www/us/en/architecture-and-technology/engineering-new-protections-into-hardware.html`

Intel Mitigations



- Disable hyperthreading or group scheduling
- Overwrite microarchitectural buffers
  - `VERW` instruction (microcode update)
  - Software sequences
- New CPUs which are not affected

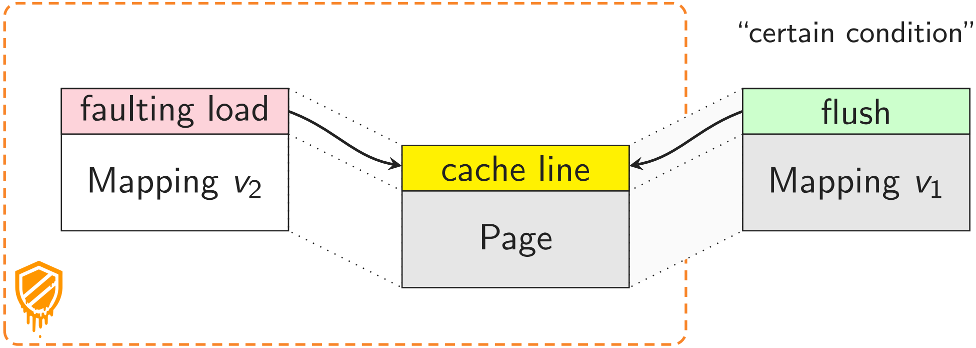
CPU	Meltdown	Foreshadow	RIDL	Fallout	MLPDS	MDSUM	ZombieLoad
8th/9th gen. Intel Core Coffee Lake	⚒	⚒	⚒	⚒	⚒	⚒	???
Intel Xeon Cascade Lake	⚒	⚒	⚒	⚒	⚒	⚒	???

https://www.intel.com/content/www/us/en/architecture-and-technology/engineering-new-protections-into-hardware.html

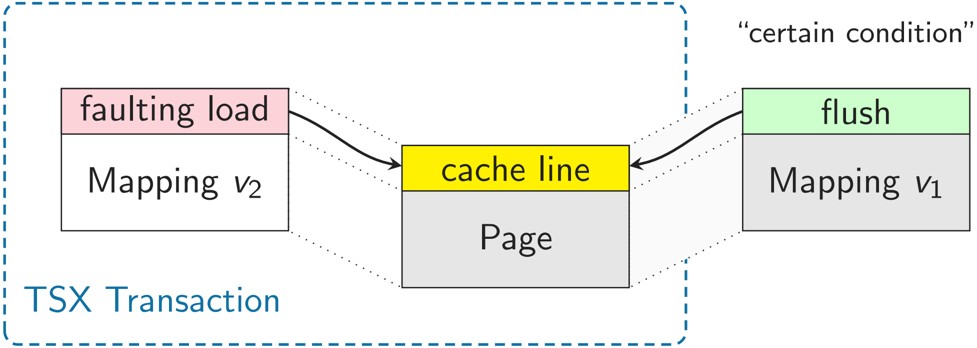
RISE AGAIN

What about second variant?

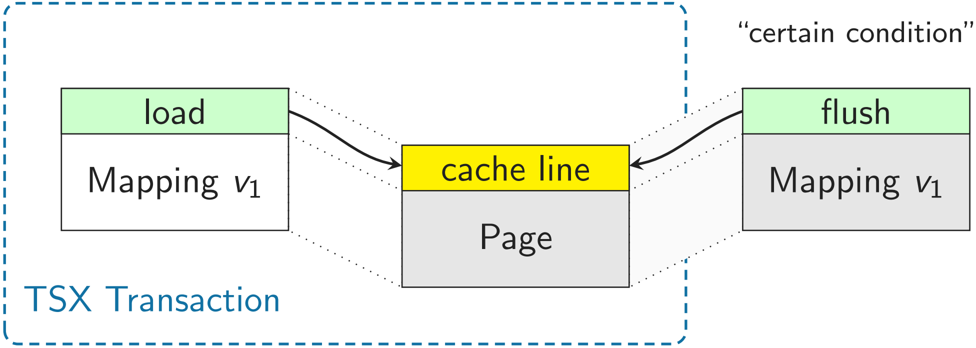
ZombieLoad Variant 2

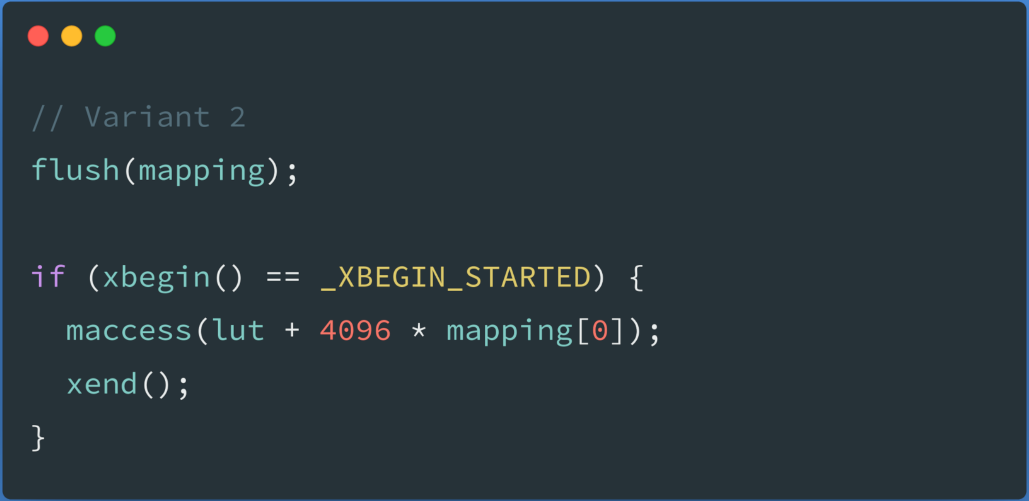


ZombieLoad Variant 2



ZombieLoad Variant 2





```
// Variant 2
flush(mapping);

if (xbegin() == _XBEGIN_STARTED) {
  maccess(lut + 4096 * mapping[0]);
  xend();
}
```

Abort Transactions

- **Data Conflicts**

Abort Transactions

- Data Conflicts
- Limited Transactional Resources

Abort Transactions

- **Data Conflicts**
- **Limited Transactional Resources**
- **Certain Instructions**
    - IO instructions, syscall, ...

Abort Transactions

- **Data Conflicts**
- **Limited Transactional Resources**
- **Certain Instructions**
  - IO instructions, syscall, ...
- **Synchronous Exception Events**
  - #BR, #PF, #DB, #BP/INT3, ...

TSX Intel Manual

12.2.4.5 Miscellaneous Transactional Aborts

Asynchronous events (NMI, SMI, INTR, IPI, PMI, etc.) occurring during transactional execution may cause the transactional execution to abort and transition to a non-transactional execution.

TSX Intel Manual

12.2.4.5 Miscellaneous Transactional Aborts

Asynchronous events (NMI, SMI, INTR, IPI, PMI, etc.) occurring during transactional execution may cause the transactional execution to abort and transition to a non-transactional execution. [...] For example, operating systems with timer ticks generate interrupts that can cause transactional aborts.

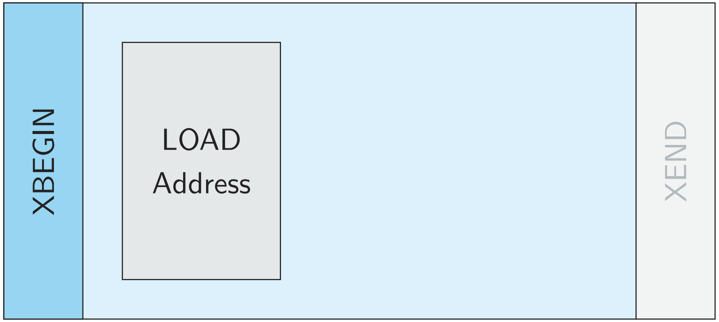
TAA

XBEGIN
XEND

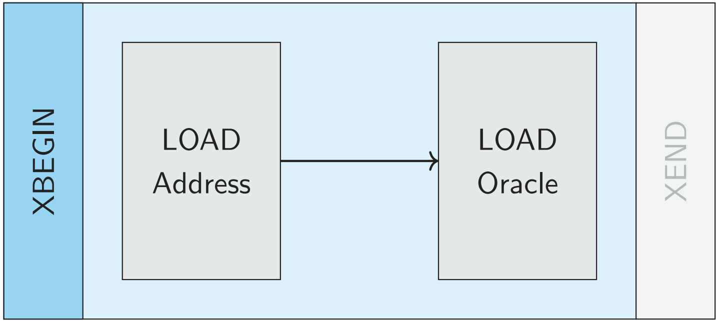
TAA

XBEGIN
XEND

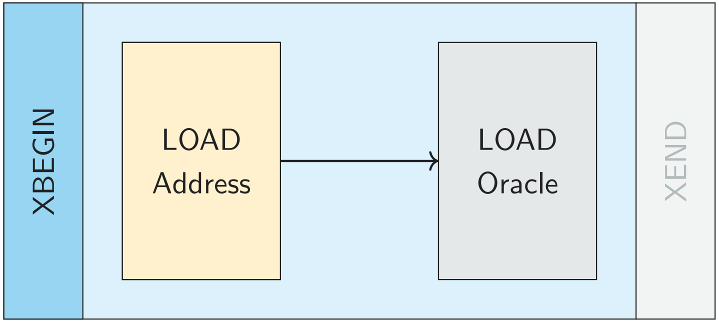
TAA



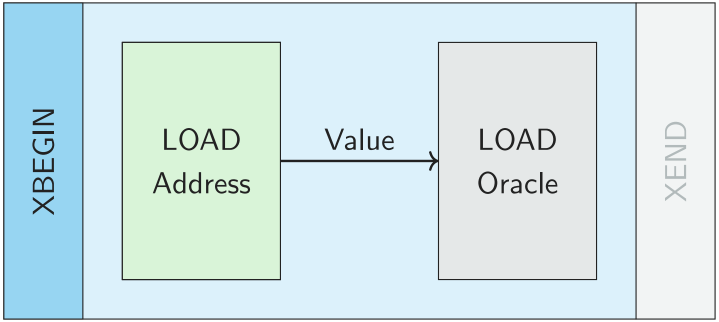
TAA



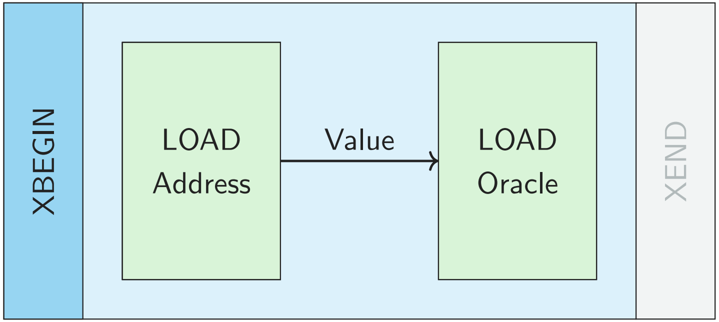
TAA



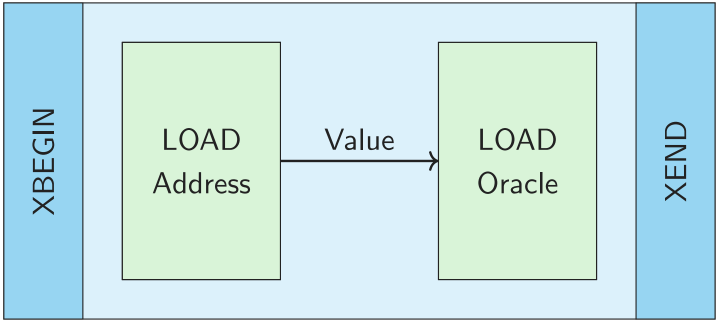
TAA



TAA



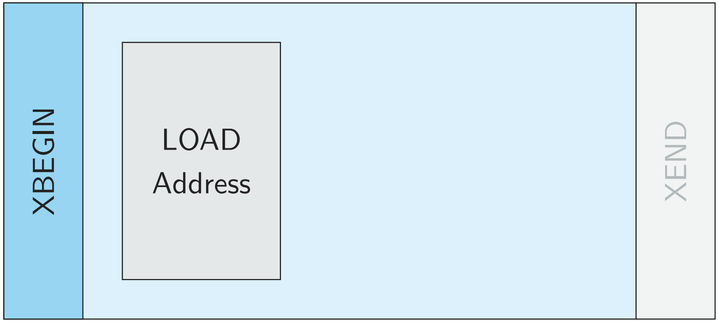
TAA



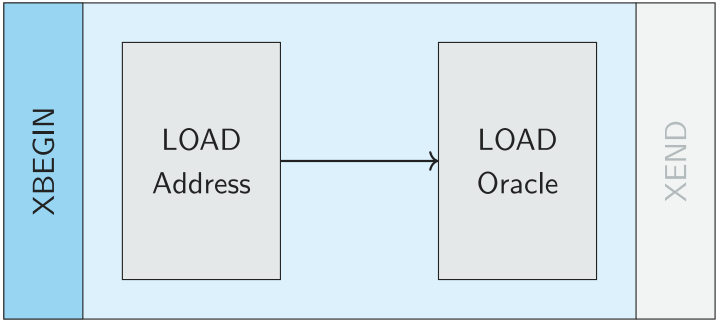
TAA

XBEGIN
XEND

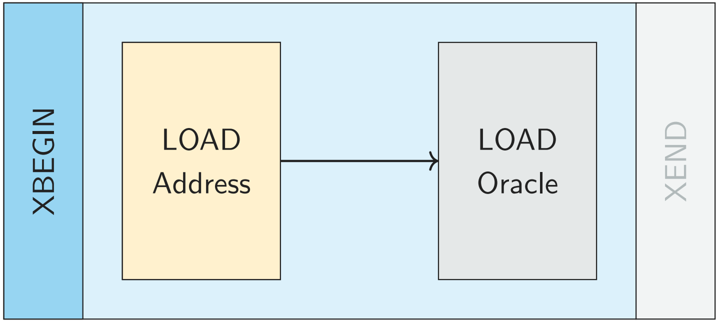
TAA



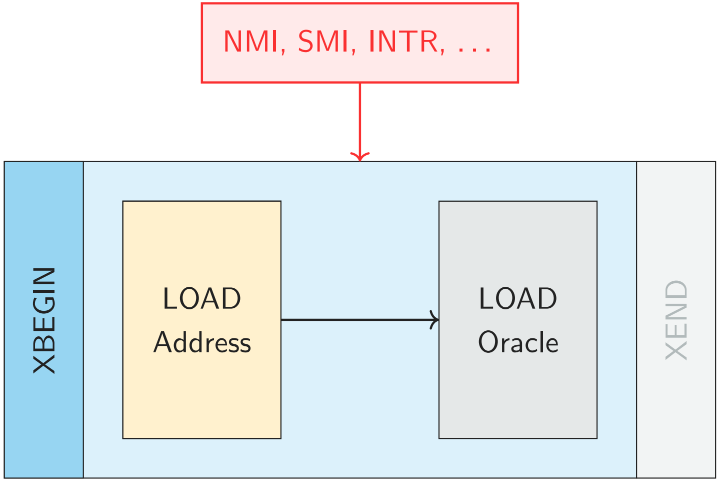
TAA



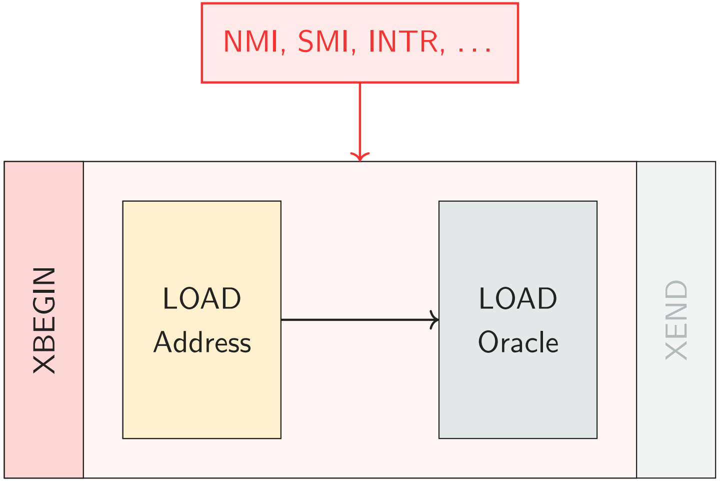
TAA



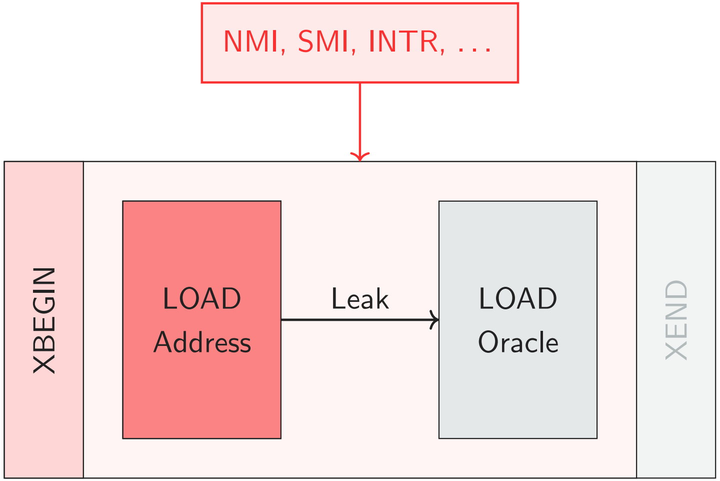
TAA



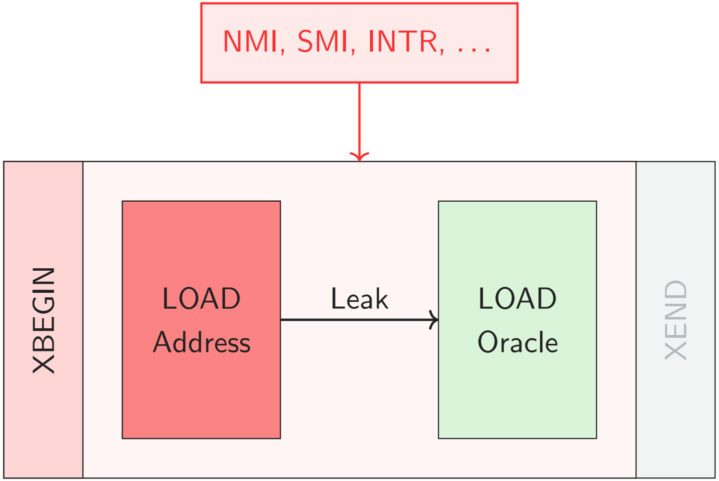
TAA



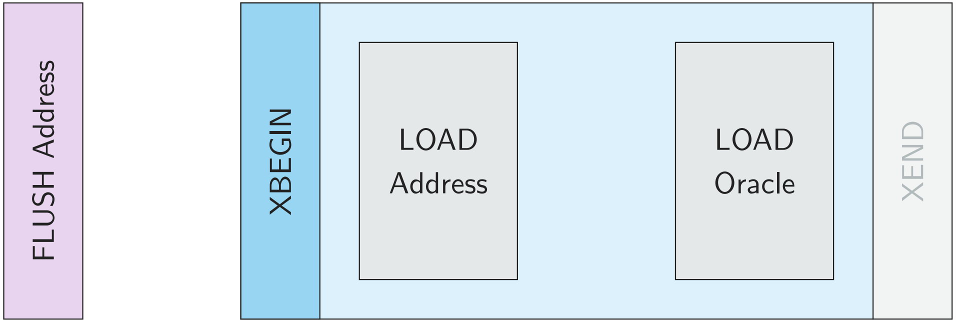
TAA



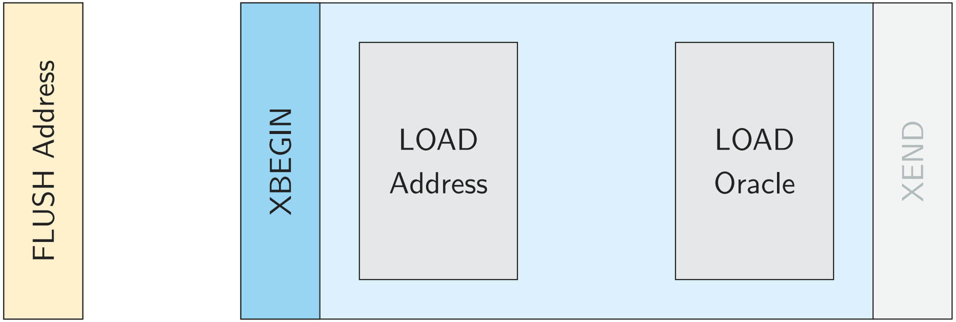
TAA



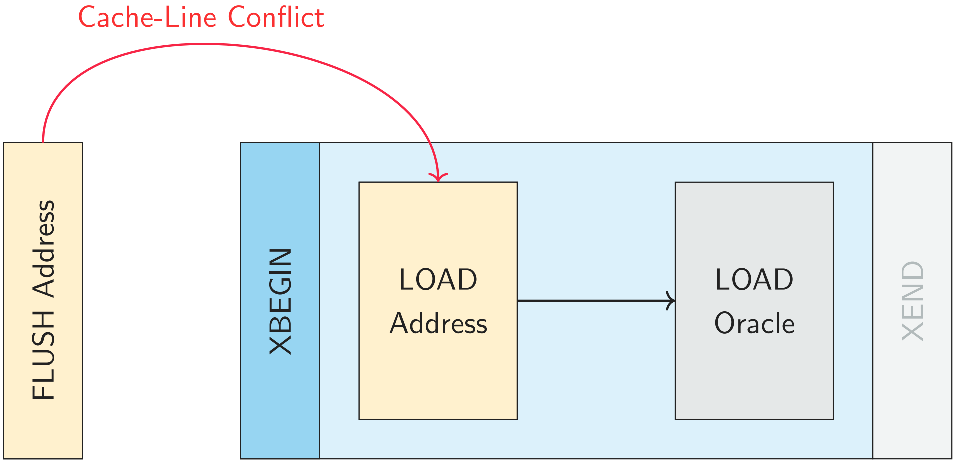
TAA



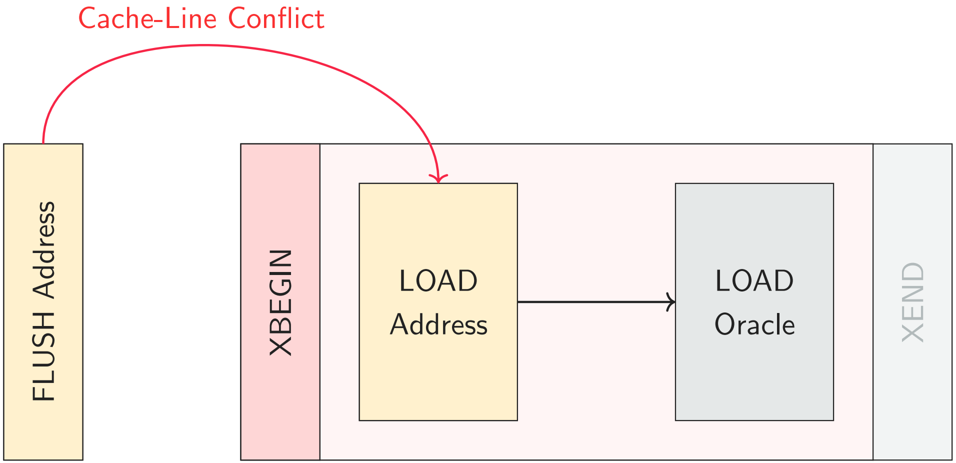
TAA



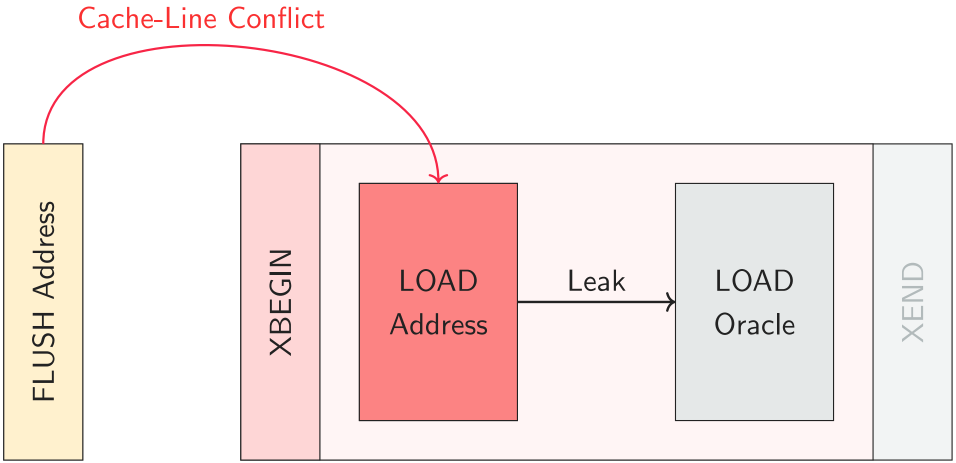
TAA


Cache-Line Conflict
FLUSH Address
XBEGIN
LOAD Address
LOAD Oracle
XEND

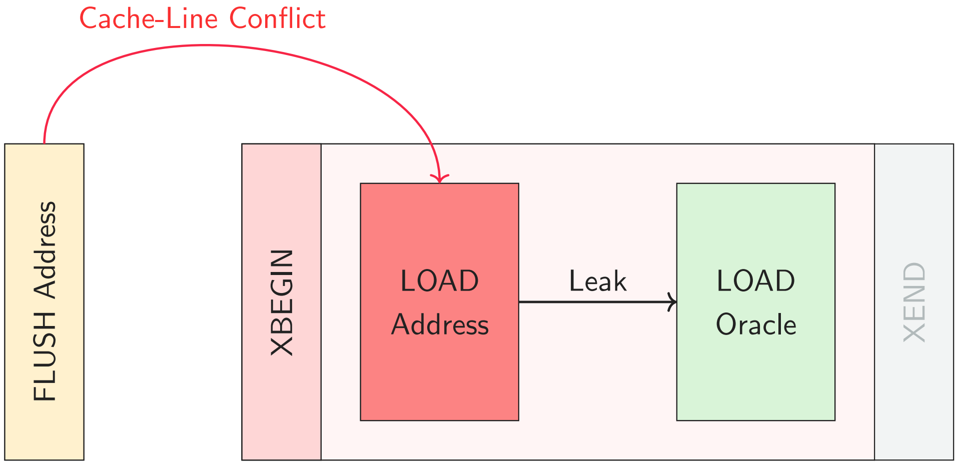
TAA


Cache-Line Conflict
FLUSH Address
XBEGIN
LOAD Address
LOAD Oracle
XEND

TAA


Cache-Line Conflict
FLUSH Address
XBEGIN
LOAD Address
Leak
LOAD Oracle
XEND

TAA


Cache-Line Conflict
FLUSH Address
XBEGIN
LOAD Address
Leak
LOAD Oracle
XEND

Attacker Models

Variant 1

Kernel Mapping

	🪟	🐧
👤	○	◐
⚙	●	●

Variant 3

Microcode-Assisted Page-Table Walk

	🪟	🐧
👤	●	○
⚙	●	●

● works ○ does not work ◐ can be prevented

Attacker Models

Variant 1
Kernel Mapping

	🪟	🐧
👤	○	◐
⚙	●	●

Variant 2
Transactional Asynchronous Abort

	🪟	🐧
👤	◐	◐
⚙	◐	◐

Variant 3
Microcode-Assisted Page-Table Walk

	🪟	🐧
👤	●	○
⚙	●	●

● works ○ does not work ◐ can be prevented

Performance

Variant 1

Kernel Mapping

5.30 kB/s

Variant 3

Microcode-Assisted Page-Table Walk

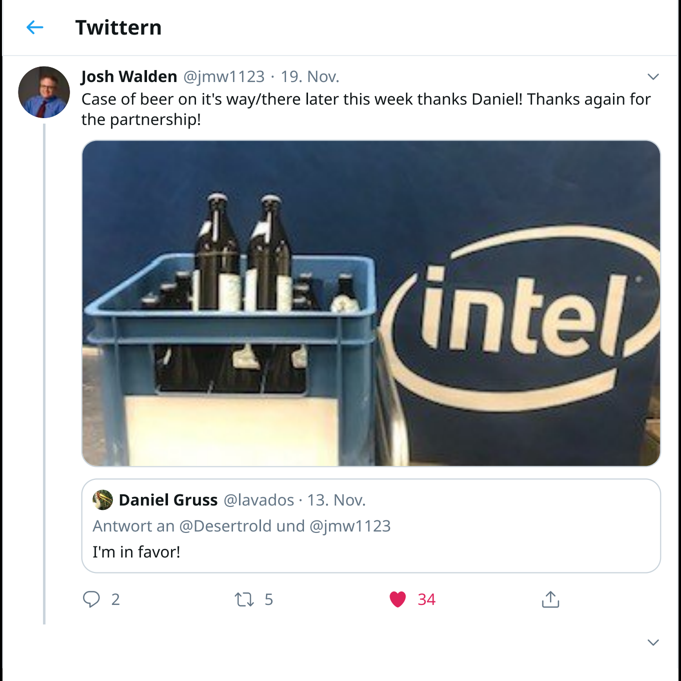
7.73 kB/s

Performance

Variant 1	Variant 2	Variant 3
Kernel Mapping	Transactional Asynchronous Abort	Microcode-Assisted Page-Table Walk
5.30 kB/s	39.66 kB/s	7.73 kB/s

DEMO

MORITZ LIPP
MICHAEL SCHWARZ
DANIEL MOGHIMI
JO VAN BULCK
ZOMBIELOAD
GRAZ UNIVERSITY OF TECHNOLOGY PRESENTS IN COLLABORATION WITH
WORCESTER POLYTECHNIC INSTITUTE, KU LEUVEN, AND CYBERUS TECHNOLOGY
AN ACM CCS 2019 PAPER "ZOMBIELOAD: CROSS-PRIVILEGE-BOUNDARY DATA SAMPLING"
WRITTEN BY MICHAEL SCHWARZ, MORITZ LIPP, DANIEL MOGHIMI, JO VAN BULCK, JULIAN STECKLINA, THOMAS PRESCHER, DANIEL GRUSS



Twittern

Josh Walden @jmw1123 · 19. Nov.

Case of beer on it's way/there later this week thanks Daniel! Thanks again for the partnership!

Daniel Gruss @lavados · 13. Nov.

Antwort an @Desertrold und @jmw1123

I'm in favor!

2 5 34

Daniel Gruss
@lavados

Antwort an @jmw1123

Thanks again Josh!

We already received the case a month ago but only found time this weekend to sit together and enjoy some!

We wish you a merry Christmas and look forward to continue working with Intel next year.

cc @cc0x1f @mlqxyz @misc0110 @tugraz_csbme #tugraz

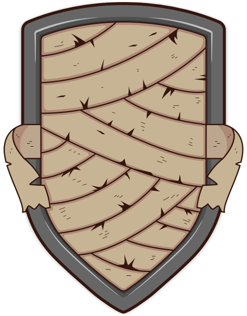
Tweet übersetzen

Du und Claudio Canella

5:45 nachm. · 24. Dez. 2019 · Twitter Web App

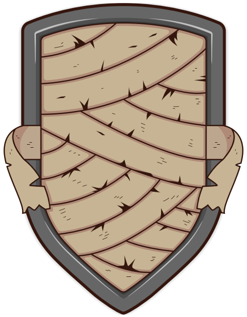
23 „Gefällt mir"-Angaben

Intel Mitigations



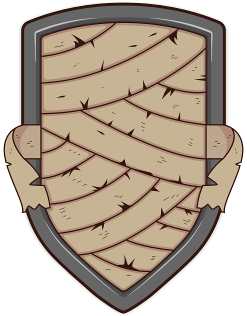
- Disable Intel TSX

Intel Mitigations



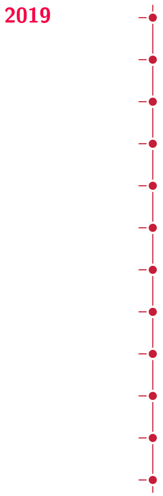
- Disable Intel TSX
  - Deactivated by default with new microcode updates on CPUs enumerating `MDS_NO`

Intel Mitigations



- Disable Intel TSX
  - Deactivated by default with new microcode updates on CPUs enumerating `MDS_NO`
- VERW to overwrite affected buffers

Timeline



Timeline

2019

April 12 — We report ZombieLoad

Timeline

2019	
April 12	We report ZombieLoad
April 24	Report Variant 2

Timeline

2019	
April 12	We report ZombieLoad
April 24	Report Variant 2
May 10	Report TAA on Cascade Lake

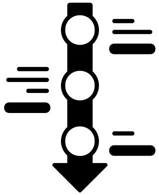
Timeline

2019	
April 12	We report ZombieLoad
April 24	Report Variant 2
May 10	Report TAA on Cascade Lake
May 11	Call with Intel + Embargo

Timeline

2019	
April 12	We report ZombieLoad
April 24	Report Variant 2
May 10	Report TAA on Cascade Lake
May 11	Call with Intel + Embargo
May 14	Disclosure of ZombieLoad (without Variant 2)

Timeline



2019	
April 12	We report ZombieLoad
April 24	Report Variant 2
May 10	Report TAA on Cascade Lake
May 11	Call with Intel + Embargo
May 14	Disclosure of ZombieLoad (without Variant 2)
May 14	MDS-resistant CPUs and Mitigations available

Timeline

2019	
April 12	We report ZombieLoad
April 24	Report Variant 2
May 10	Report TAA on Cascade Lake
May 11	Call with Intel + Embargo
May 14	Disclosure of ZombieLoad (without Variant 2)
May 14	MDS-resistant CPUs and Mitigations available
May 16	Report VERW/Software-Sequences are insufficient

Timeline

2019	
April 12	We report ZombieLoad
April 24	Report Variant 2
May 10	Report TAA on Cascade Lake
May 11	Call with Intel + Embargo
May 14	Disclosure of ZombieLoad (without Variant 2)
May 14	MDS-resistant CPUs and Mitigations available
May 16	Report VERW/Software-Sequences are insufficient
Nov 14	Public Disclosure of Variant 2

Timeline

2019	
April 12	We report ZombieLoad
April 24	Report Variant 2
May 10	Report TAA on Cascade Lake
May 11	Call with Intel + Embargo
May 14	Disclosure of ZombieLoad (without Variant 2)
May 14	MDS-resistant CPUs and Mitigations available
May 16	Report VERW/Software-Sequences are insufficient
Nov 14	Public Disclosure of Variant 2
2020	
January 27	Public Disclosure of L1DES

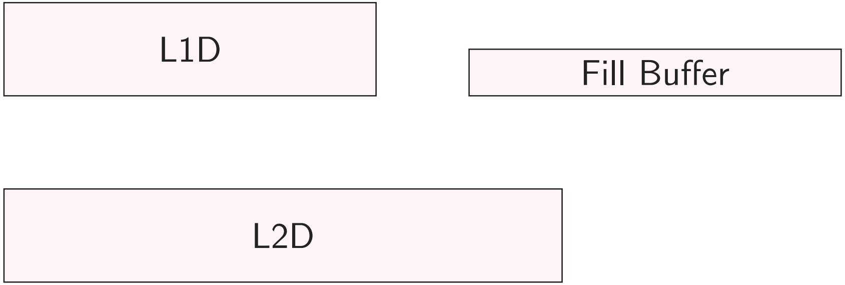
Intel Mitigations

- Software-sequences and VERW do not work reliably

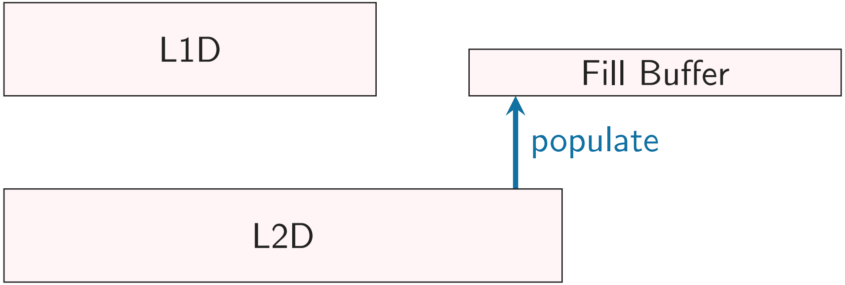
Intel Mitigations

- Software-sequences and VERW do not work reliably
  - Cases where leakage is still visible

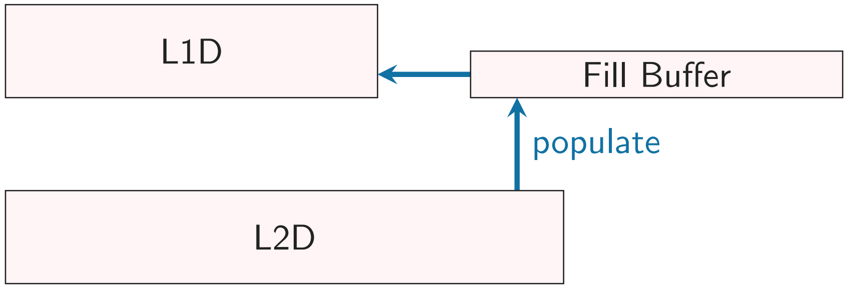
L1 Data Eviction Sampling (L1DES)



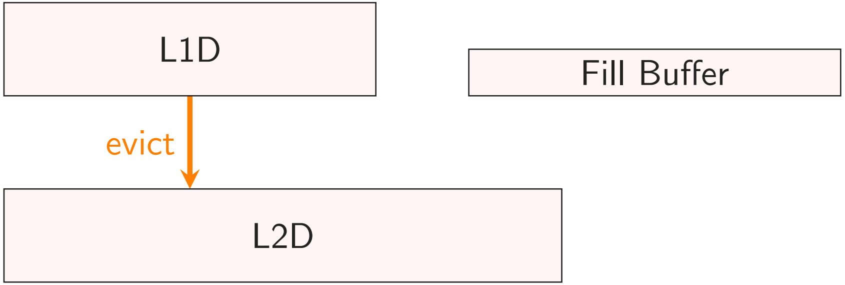
L1 Data Eviction Sampling (L1DES)



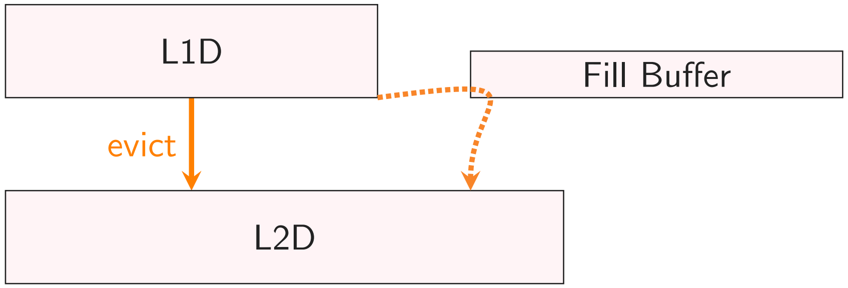
L1 Data Eviction Sampling (L1DES)



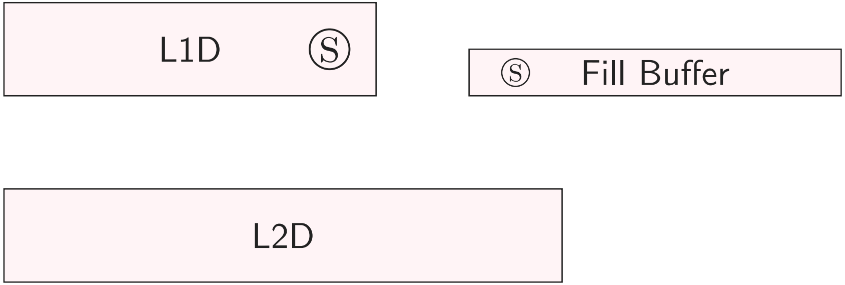
L1 Data Eviction Sampling (L1DES)



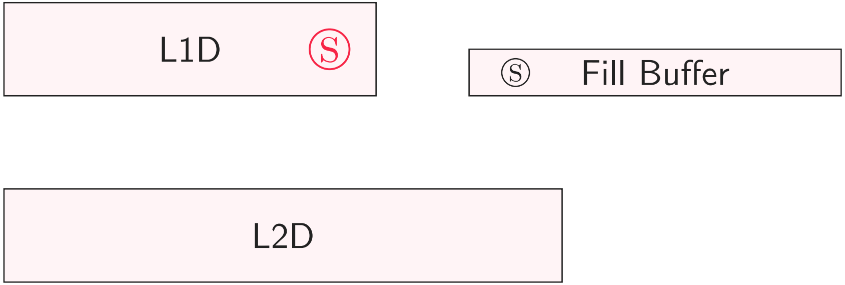
L1 Data Eviction Sampling (L1DES)



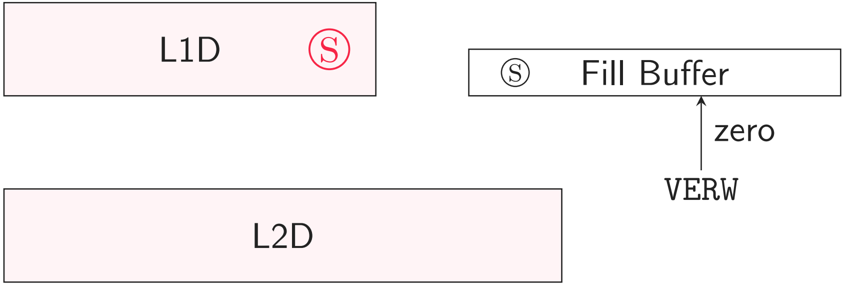
L1 Data Eviction Sampling (L1DES)



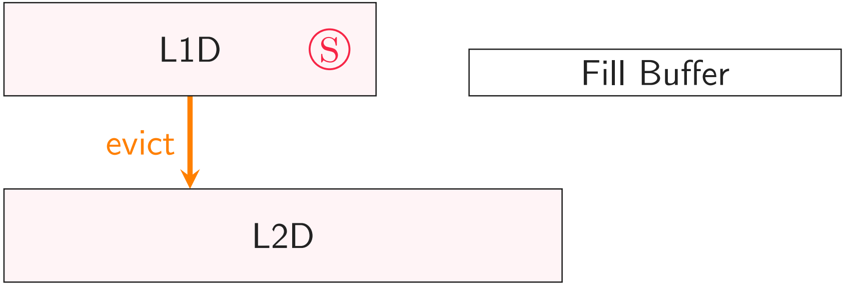
L1 Data Eviction Sampling (L1DES)



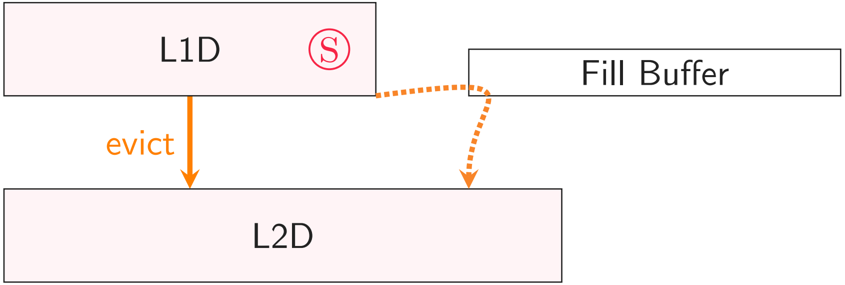
L1 Data Eviction Sampling (L1DES)



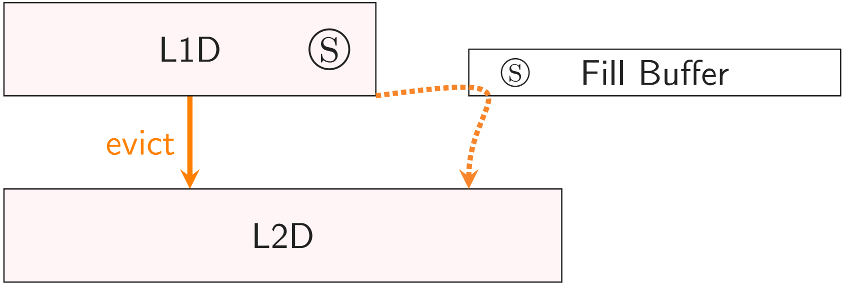
L1 Data Eviction Sampling (L1DES)



L1 Data Eviction Sampling (L1DES)



L1 Data Eviction Sampling (L1DES)

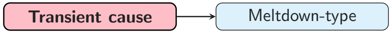


INSIGHTS

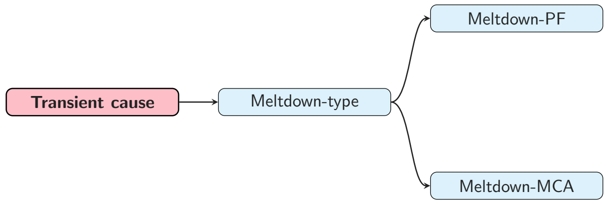
Transient Execution Attack Tree

Transient cause

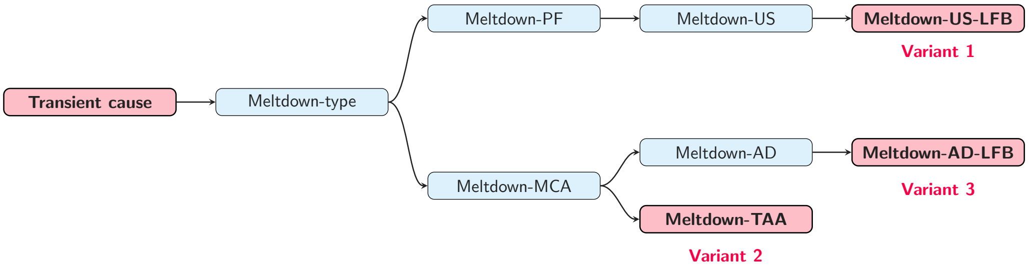
Transient Execution Attack Tree

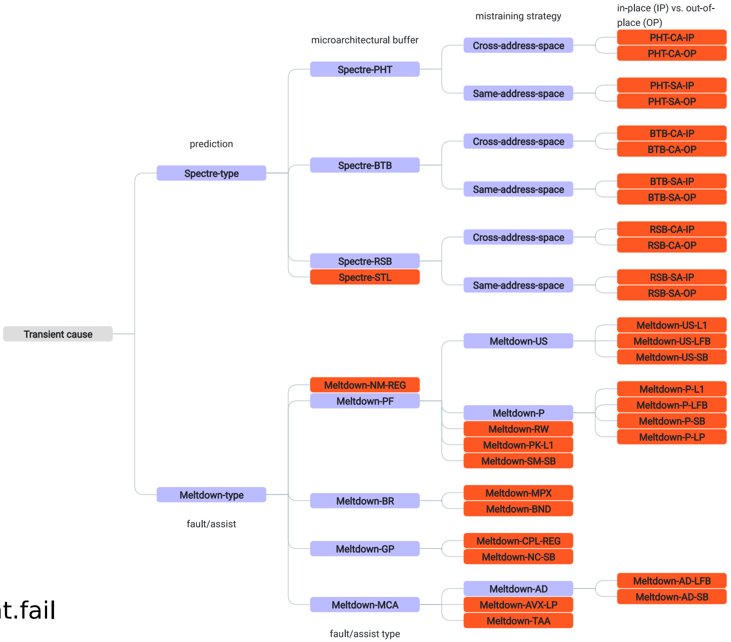


Transient Execution Attack Tree



Transient Execution Attack Tree



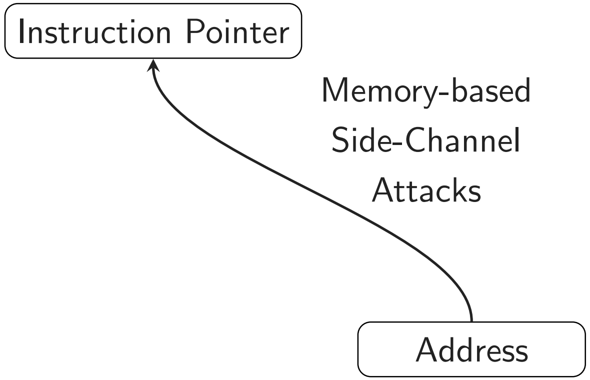


https://transient.fail

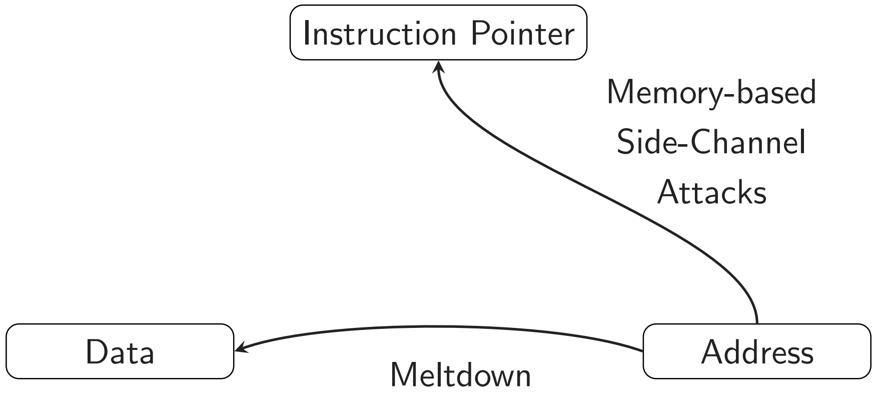
ZombieLoad Insights

Address

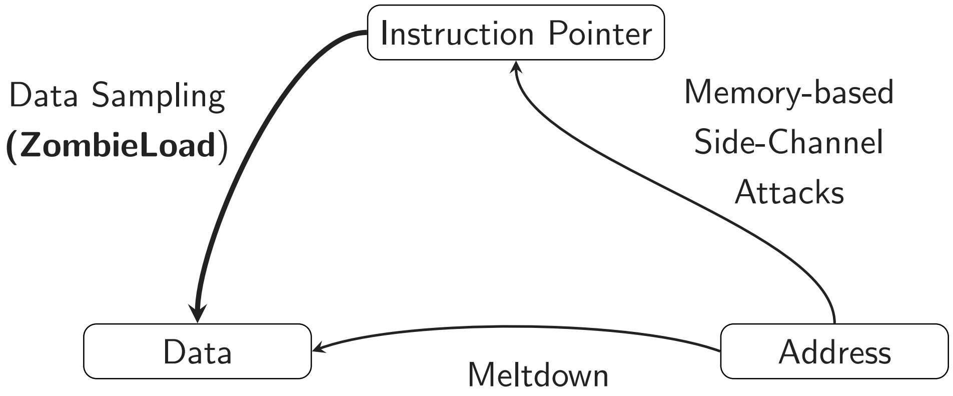
ZombieLoad Insights



ZombieLoad Insights



ZombieLoad Insights



GitHub

You can find our proof-of-concept implementation on:

- `https://github.com/IAIK/ZombieLoad`

Conclusion

- Transient-execution attacks: the gift that keeps on giving

Conclusion

- Transient-execution attacks: the gift that keeps on giving
- Class of Meltdown attacks is larger than expected

Conclusion

- Transient-execution attacks: the gift that keeps on giving
- Class of Meltdown attacks is larger than expected
- CPUs are deterministic - there is no noise

ZombieLoad: Leaking Data on Intel CPUs

https://github.com/IAIK/ZombieLoad

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz)

October 2, 2020

References

- Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. "ZombieLoad: Cross-Privilege-Boundary Data Sampling". In: *CCS*. 2019
- Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. "Meltdown: Reading Kernel Memory from User Space". In: *USENIX Security Symposium*. 2018