

ZOMBIELoad ATTACK

DANIEL GRUSS
MORITZ LIPP
MICHAEL SCHWARZ





Michael Schwarz

Post Doc @ Graz University of Technology

🐦 @misc0110

✉ michael.schwarz@iaik.tugraz.at

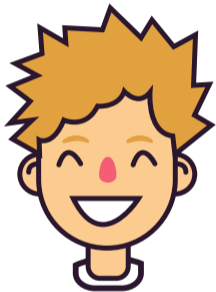


Moritz Lipp

PhD Candidate @ Graz University of Technology

🐦 @mlqxyz

✉ moritz.lipp@iaik.tugraz.at



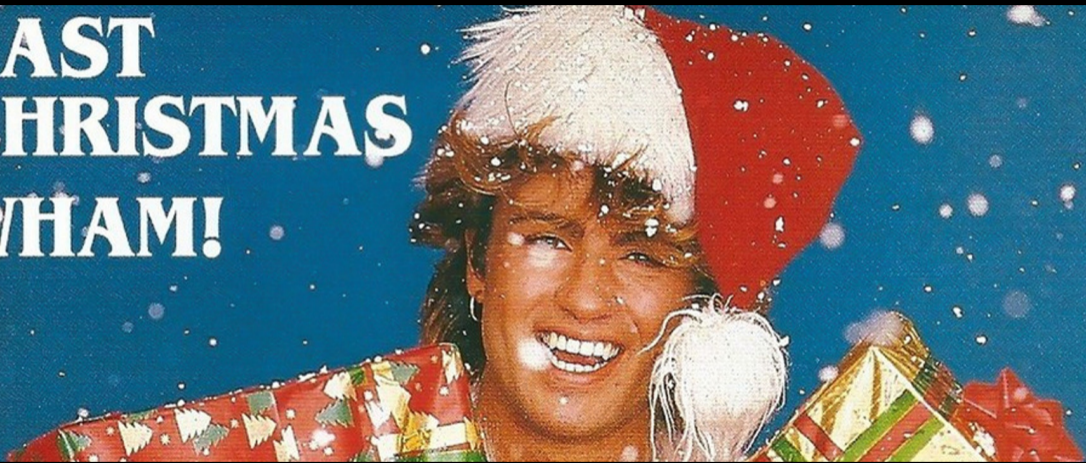
Daniel Gruss

Assistant Professor @ Graz University of Technology

🐦 @lavados

✉ daniel.gruss@iaik.tugraz.at

**LAST
CHRISTMAS
WHAM!**



A CHRISTMAS CAROL

The Spectres of the Past, Present, and Future

Claudio Canella

Moritz Lipp

Daniel Gruss

Michael Schwarz



▶ ⏪ 🔊 3:07 / 1:01:28



35C3 - A Christmas Carol - The Spectres of the Past, Present, and Future

2.766 Aufrufe • 28.12.2018

👍 47 💬 4 ➦ TEILEN ⌵ SPEICHERN ⋮



media.ccc.de
86.900 Abonnenten

ABONNIEREN

DONATE

[MEHR ANSEHEN](#)



Marc Karasek vor 11 Monaten

Ditch the suite dude please.. He looks so uncomfortable...

👍 💬 ANTWORTEN

⬆️ [Antworten ausblenden](#)

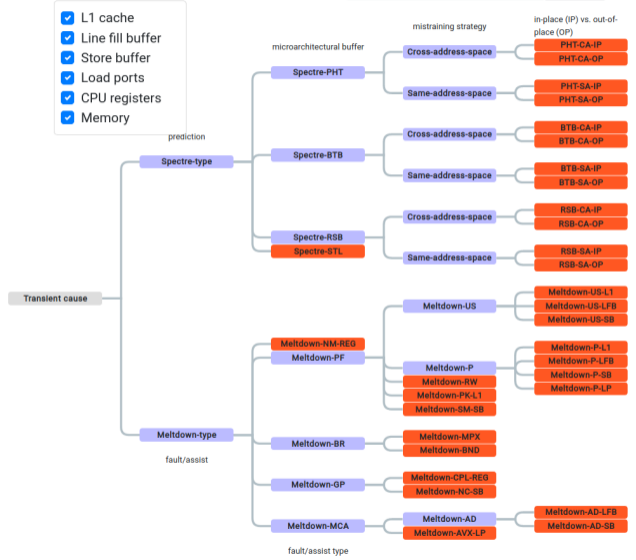


DerUnbekannte vor 11 Monaten

pretty sure that was him playing Ebenezer Scrooge

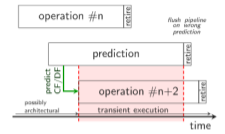
👍 💬 ANTWORTEN

- L1 cache
- Line fill buffer
- Store buffer
- Load ports
- CPU registers
- Memory



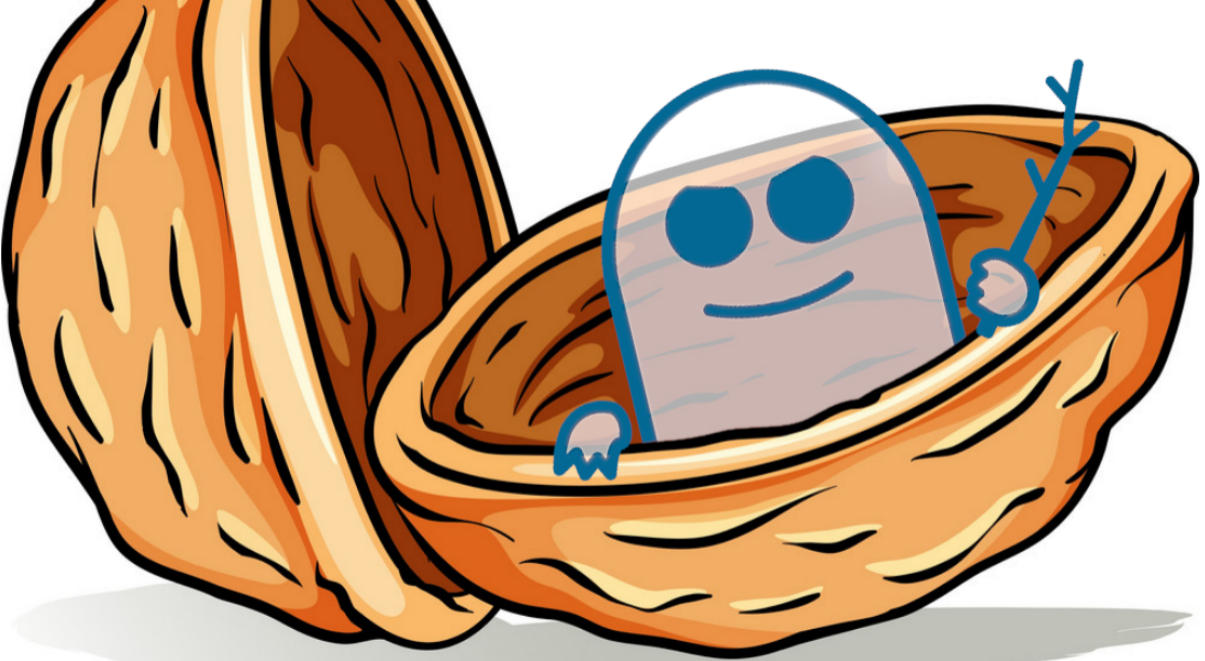
Details

Spectre-type



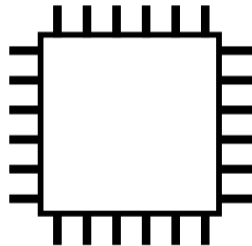
Spectre exploits a performance optimization in modern CPUs. Instead of waiting for the correct resolution of a branch, the CPU tries to predict the most likely outcome of the branch and starts transiently executing along the predicted path. Upon resolving the branch, the CPU discards the results of the transient execution if the prediction was wrong but does not revert changes in the microarchitecture. The prediction is based on events in the past, allowing an attacker to mistrain the predictor to leak data through the microarchitecture that should normally not be accessible to the attacker.

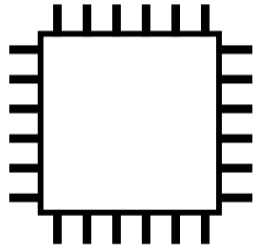
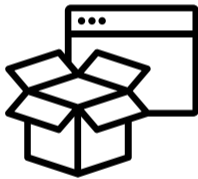
References

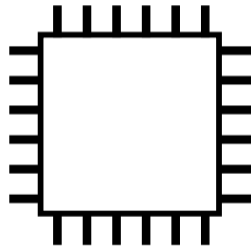
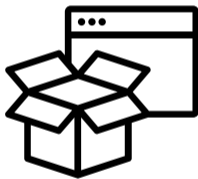


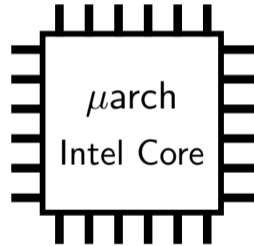
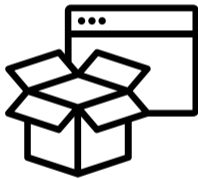


**NEED
BACKGROUND**



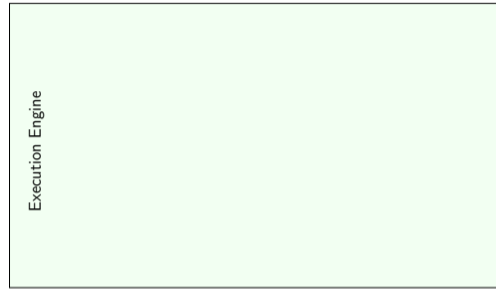
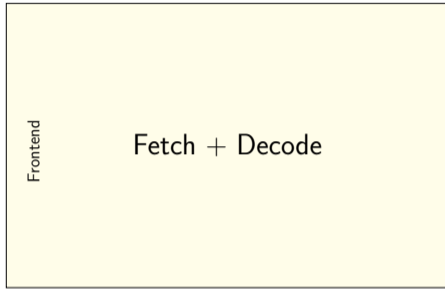


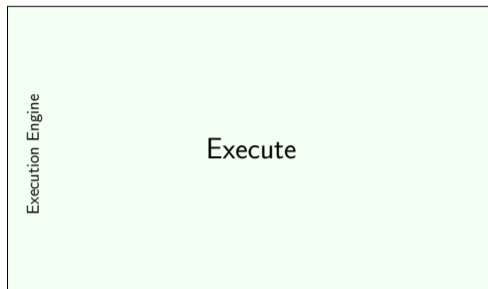
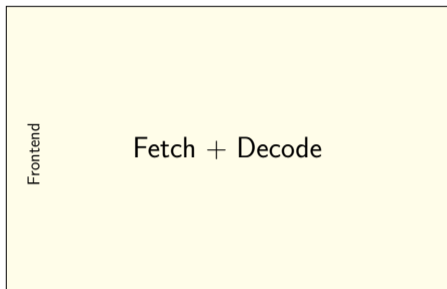


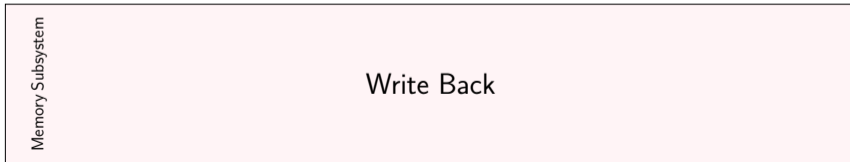
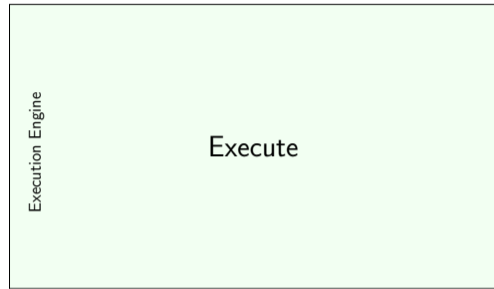
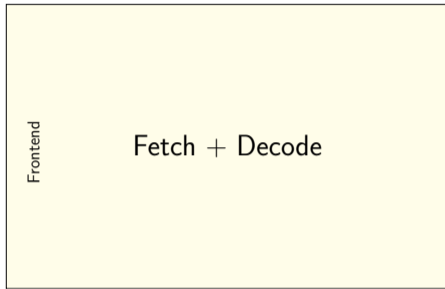




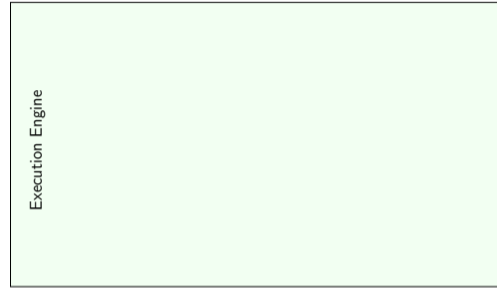
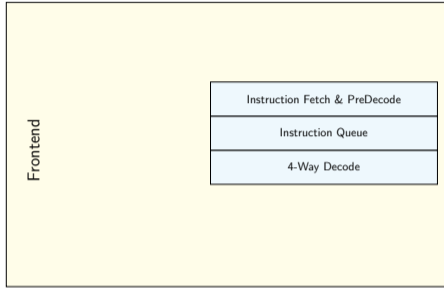


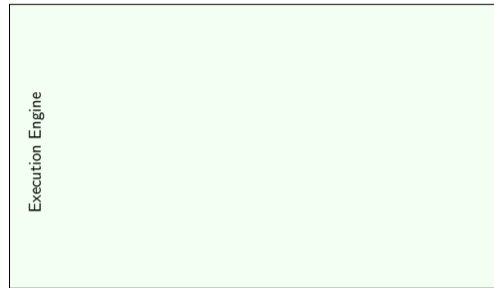
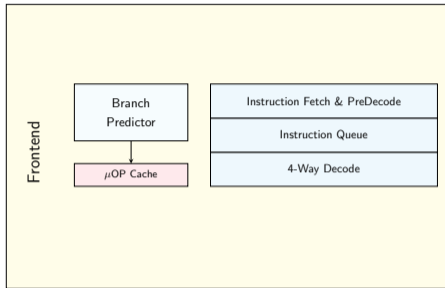


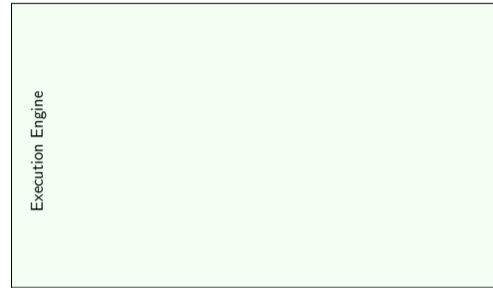
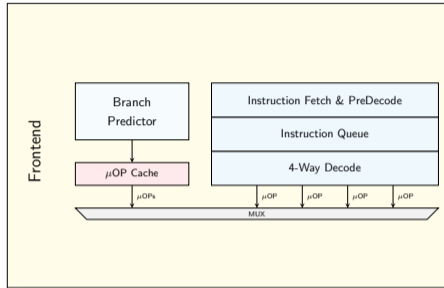


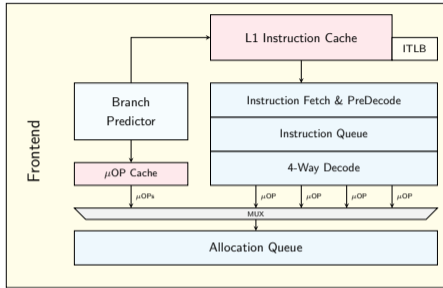


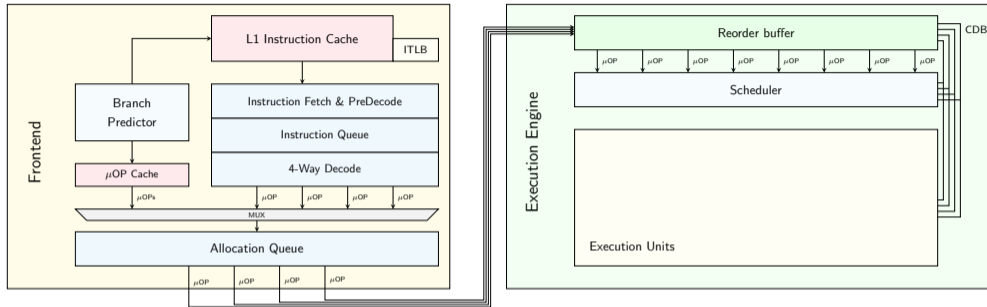


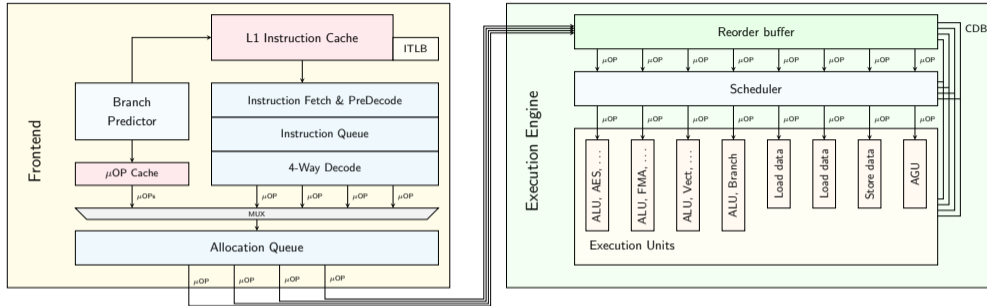


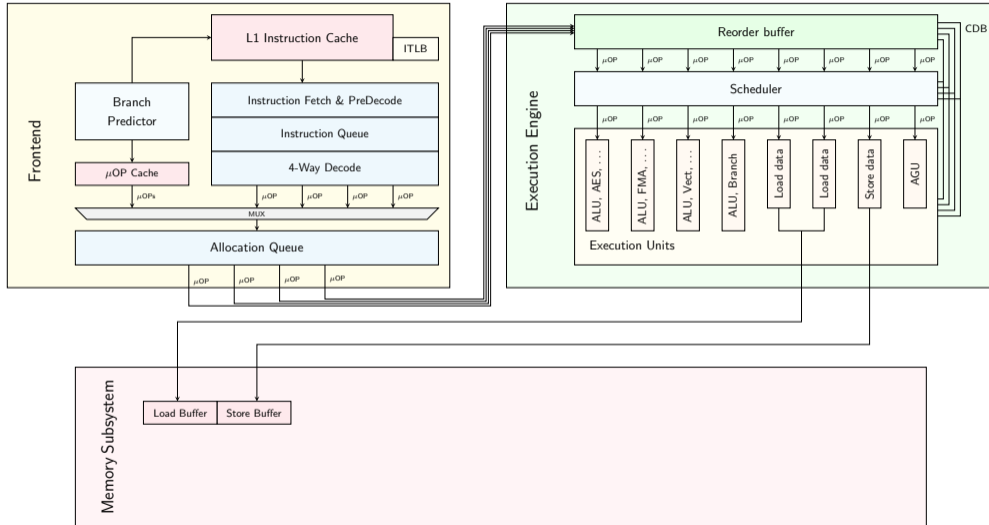


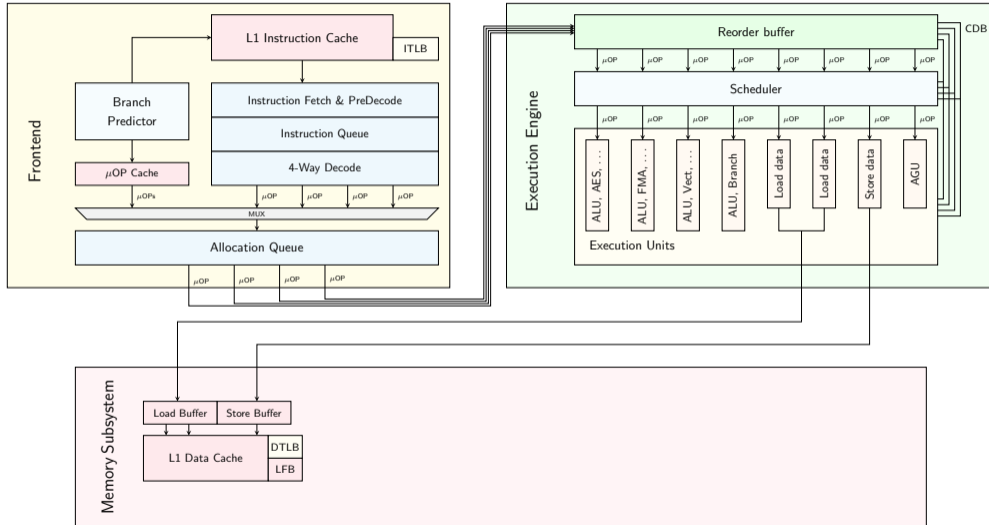


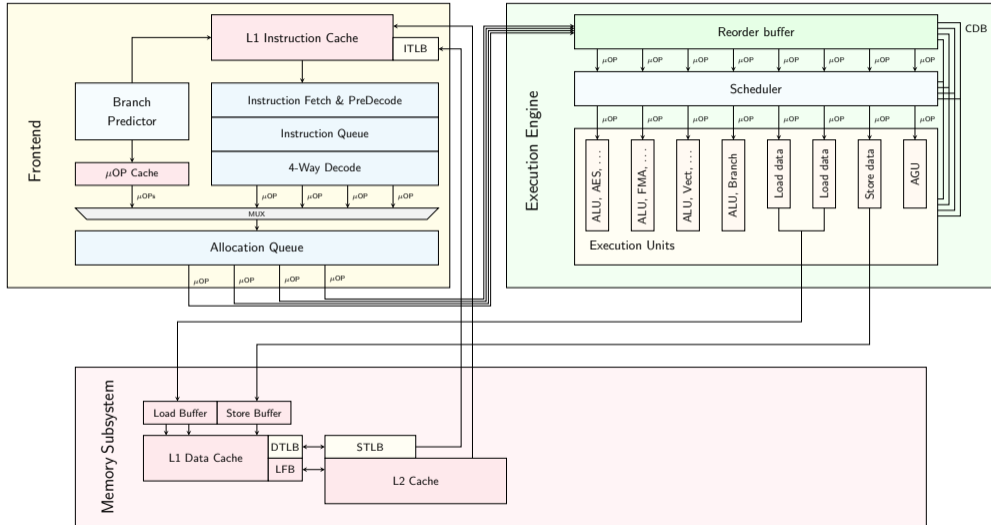


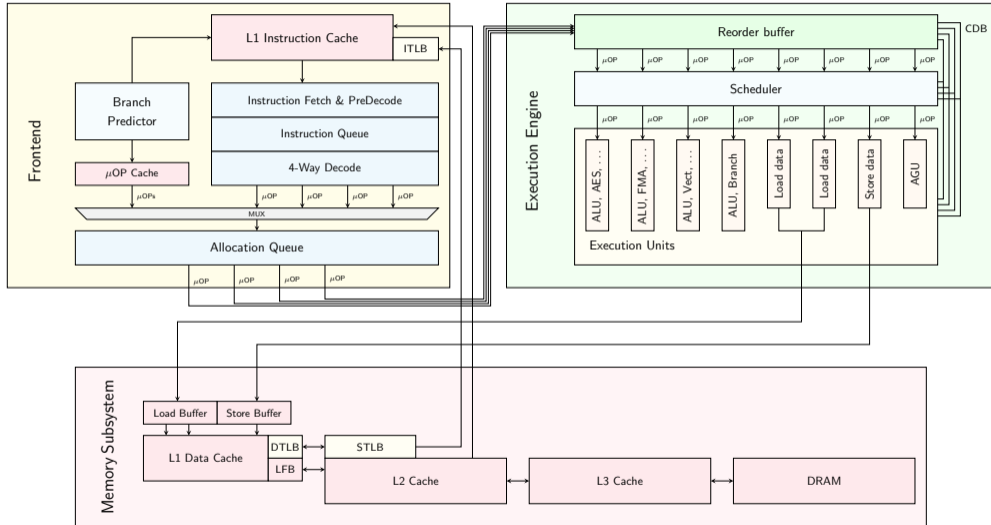




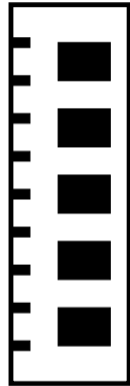
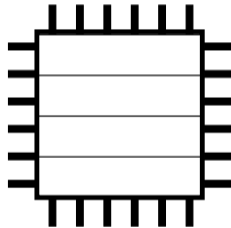






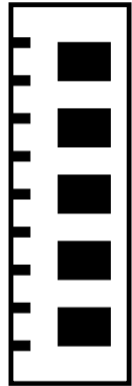
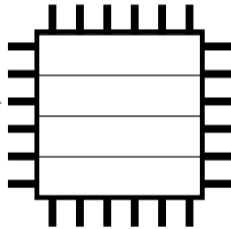


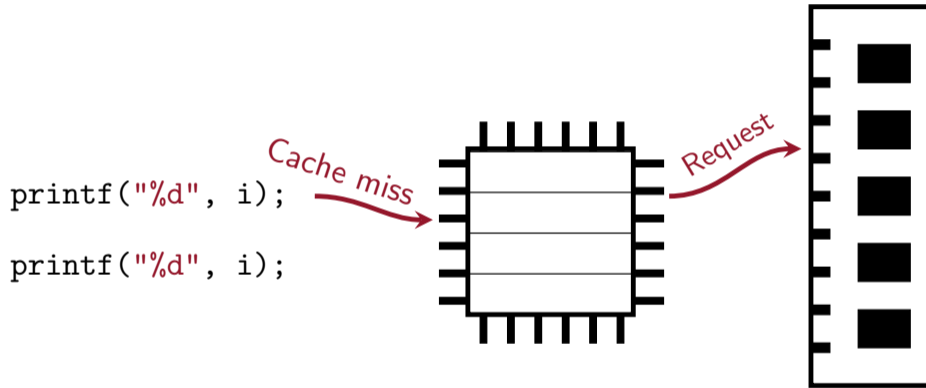
```
printf("%d", i);  
printf("%d", i);
```

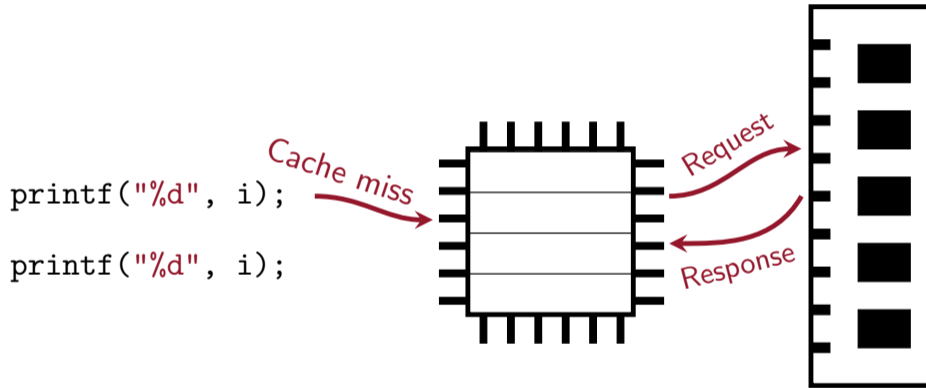


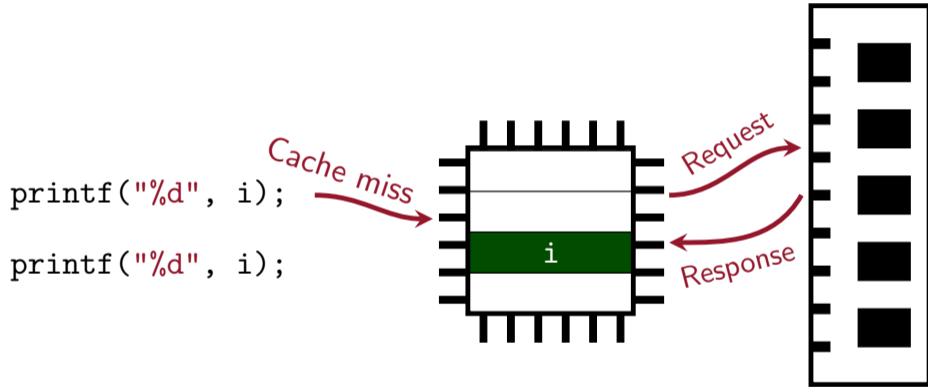
```
printf("%d", i);  
printf("%d", i);
```

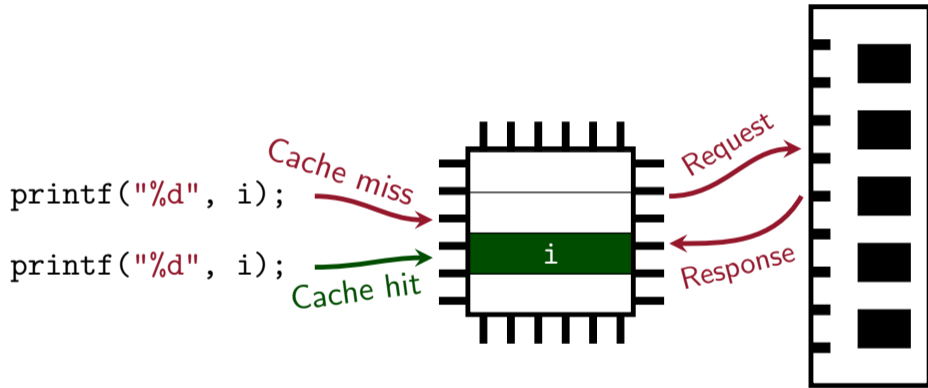
Cache miss

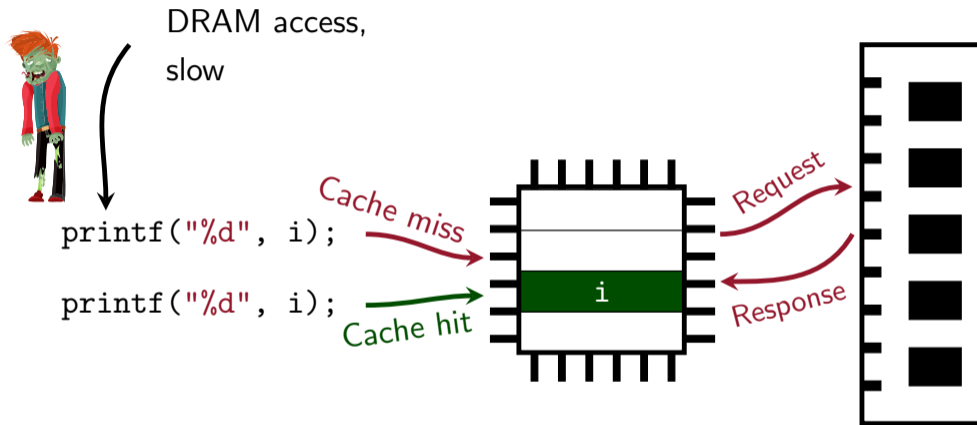


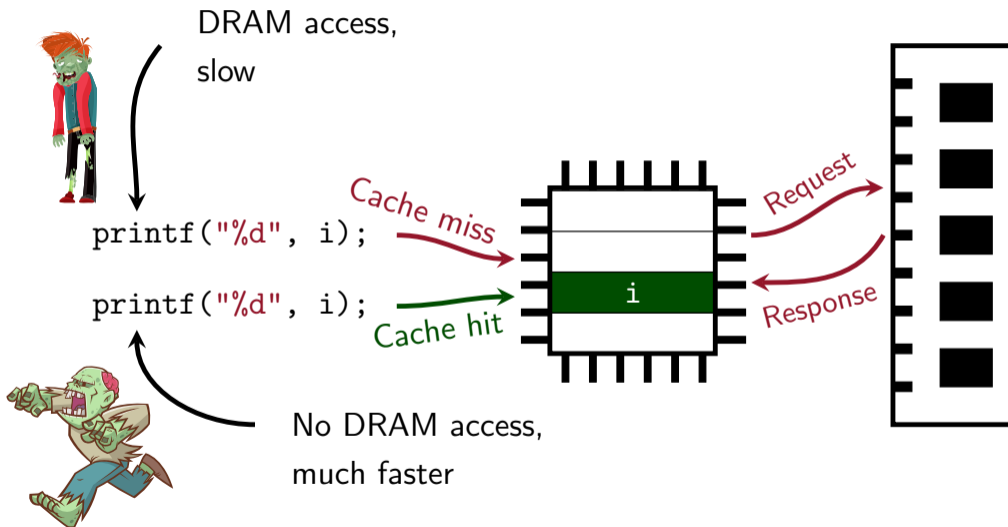


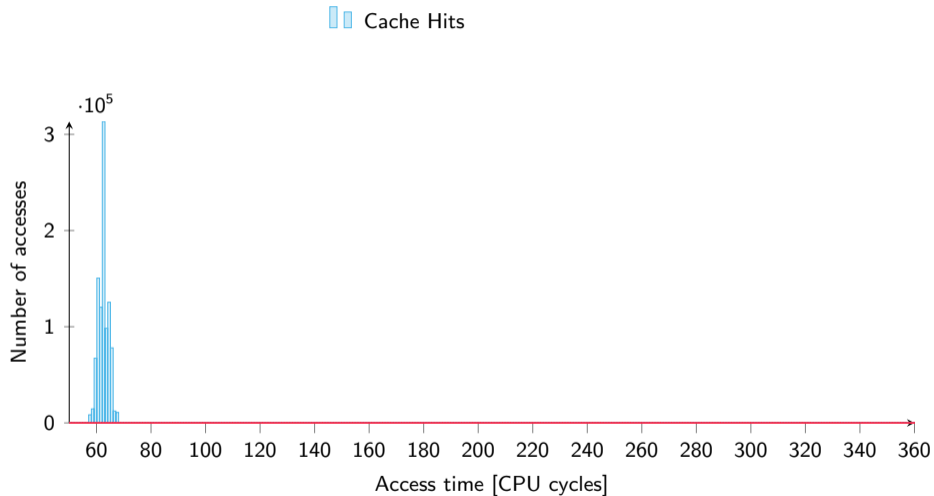


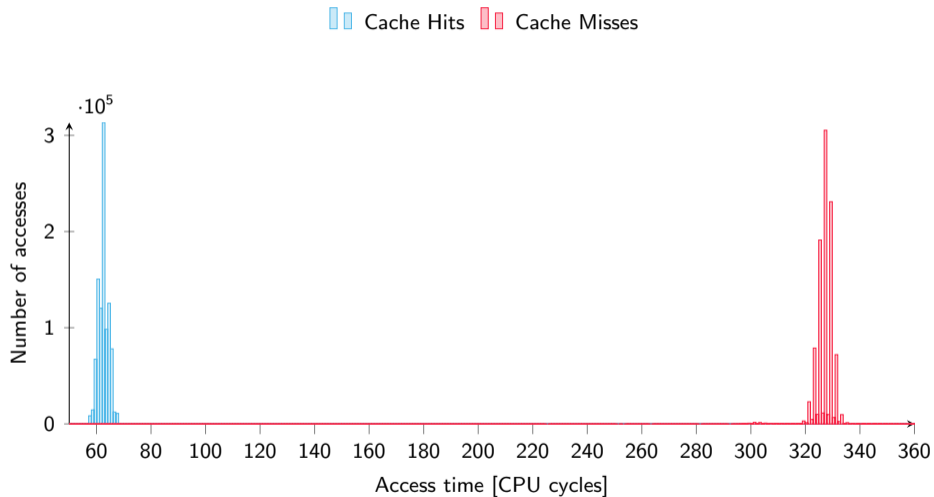


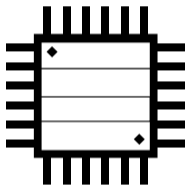
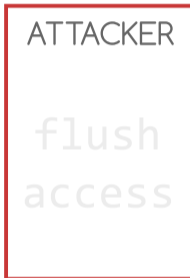


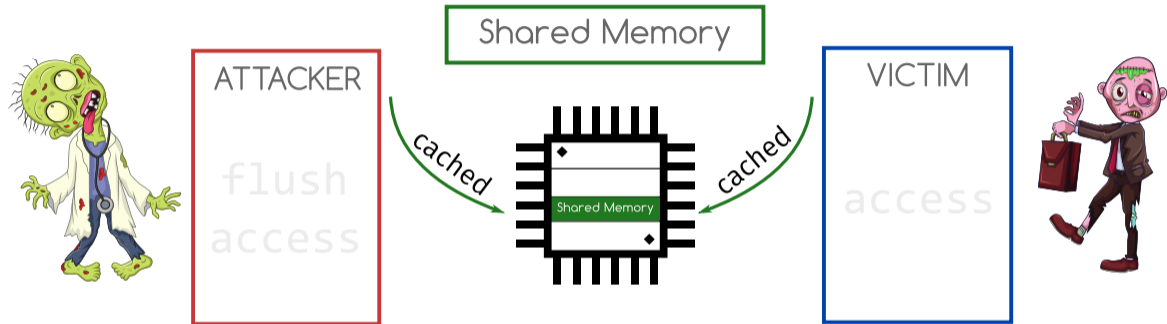


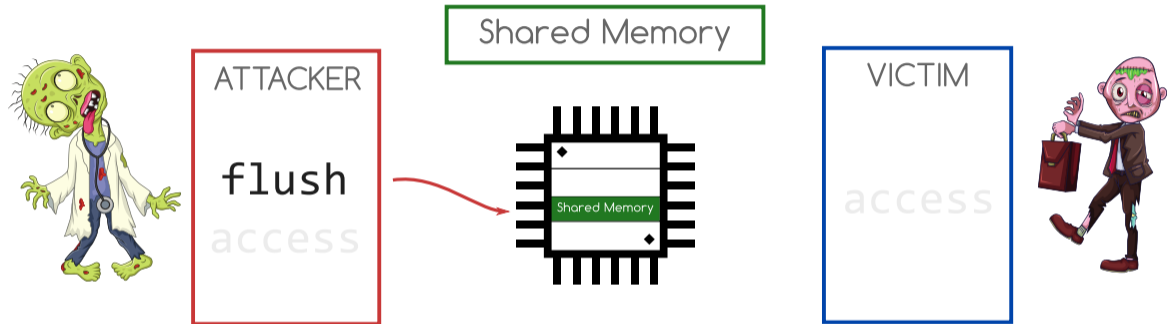


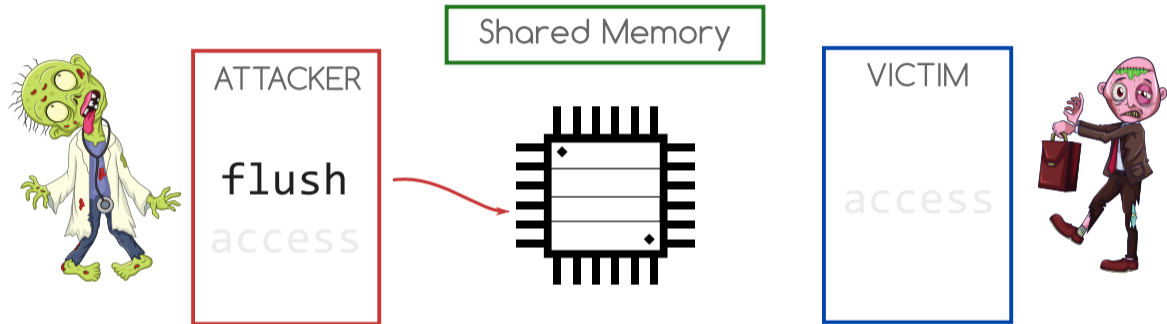


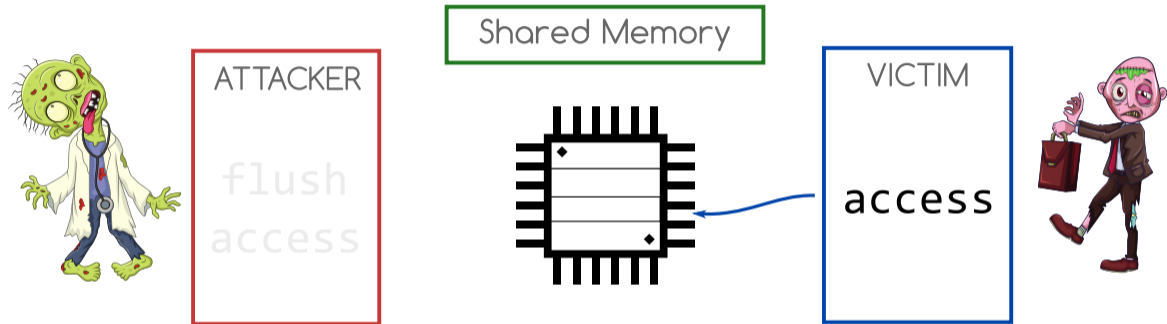


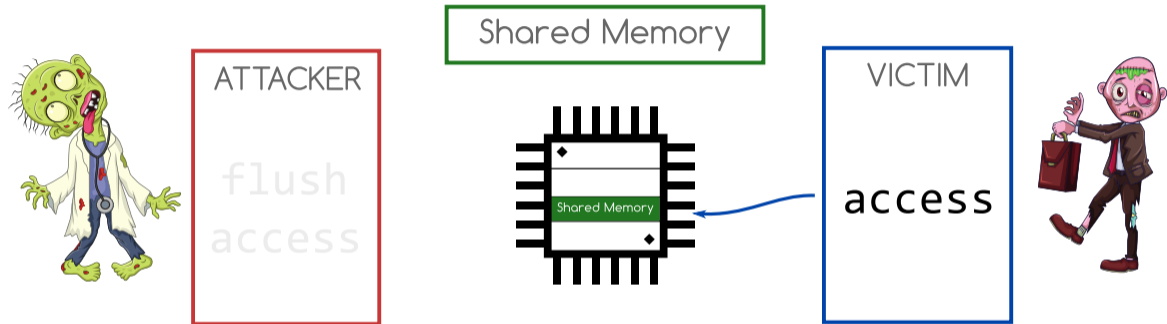


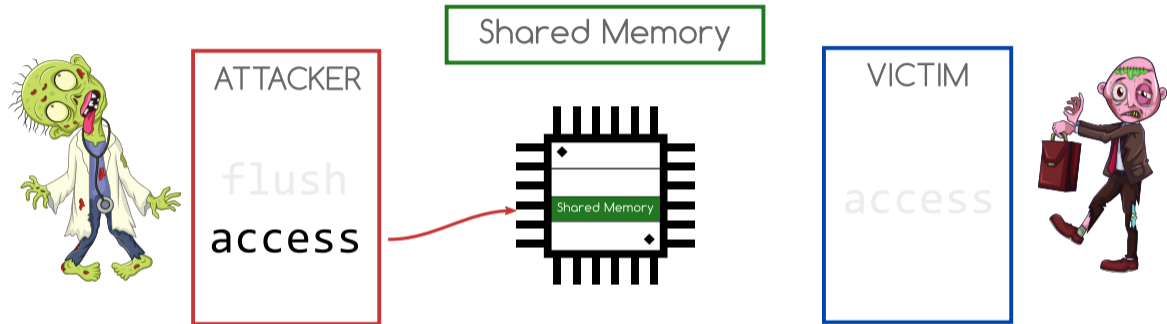


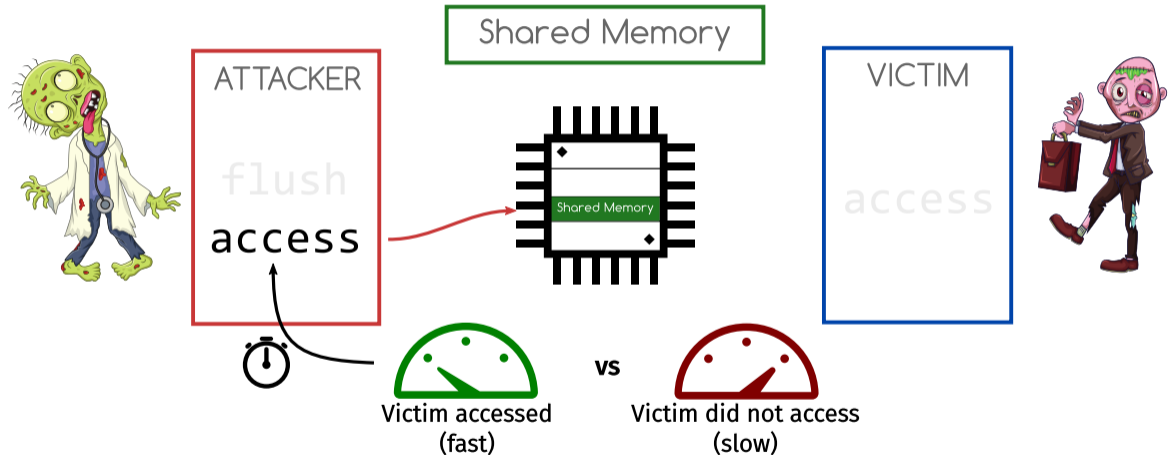


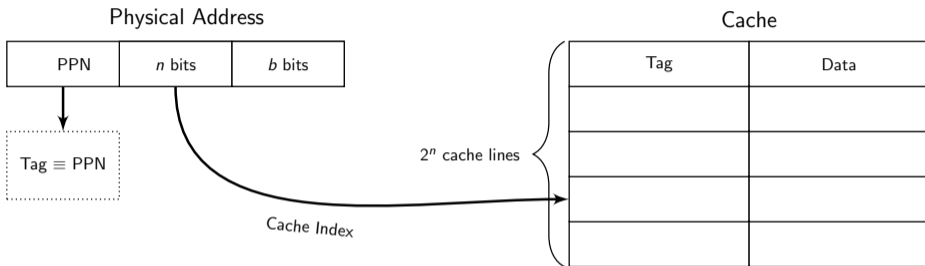


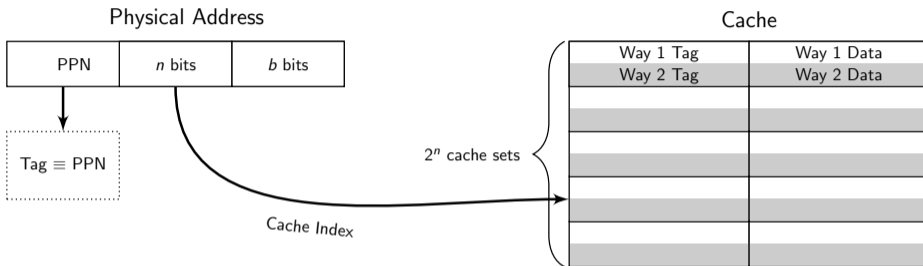


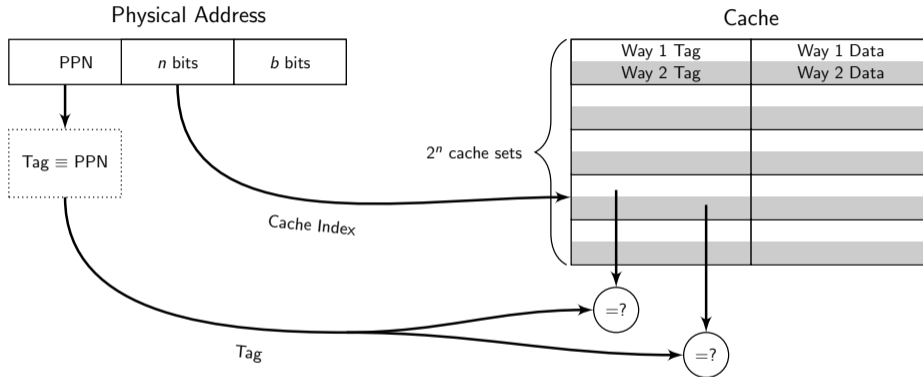


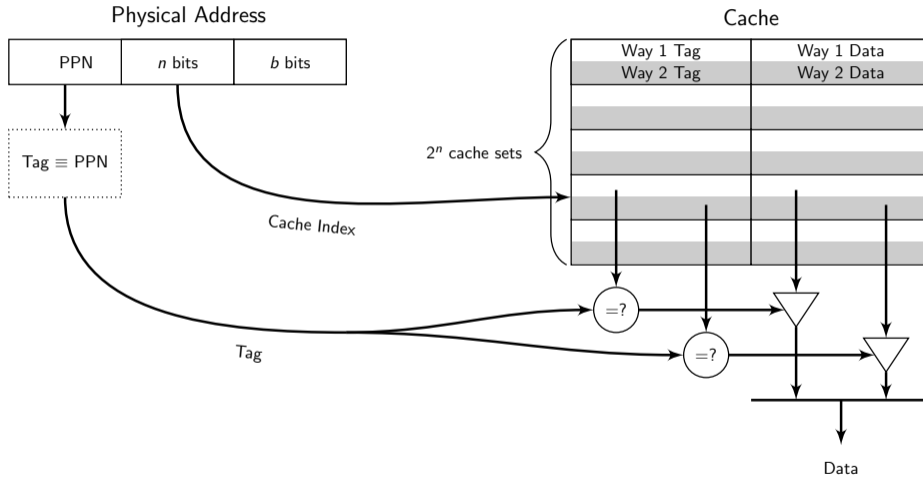










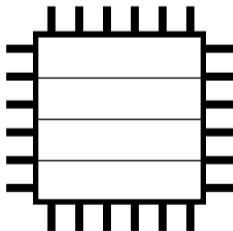




User Memory

	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]
```

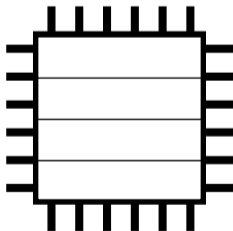


User Memory

	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)



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

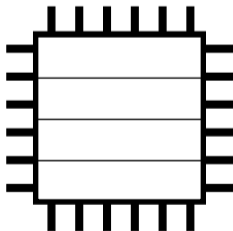
```
char value = kernel[0]
```

mem[value]

K

Page fault (Exception)

Out of order



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

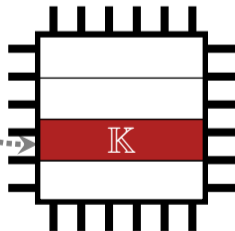
```
char value = kernel[0]
```

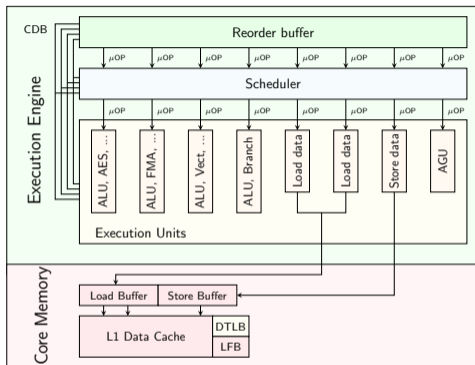
```
mem[value]
```

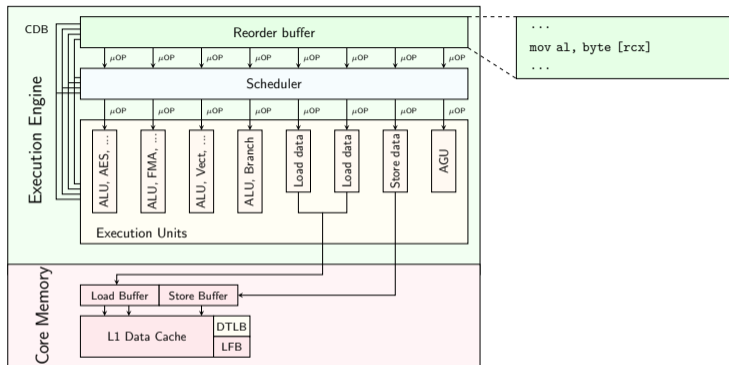
K

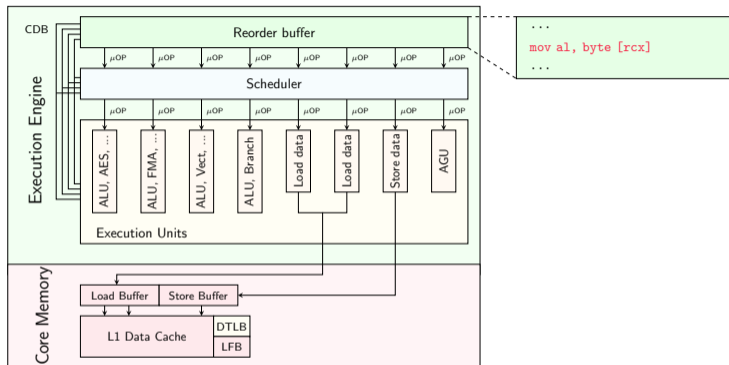
⚡ Page fault (Exception)

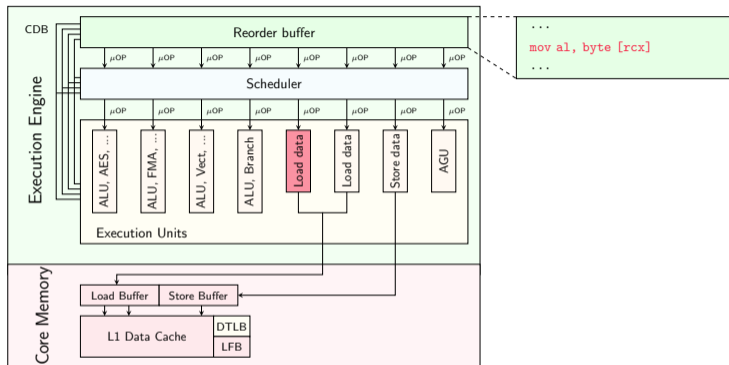
Out of order

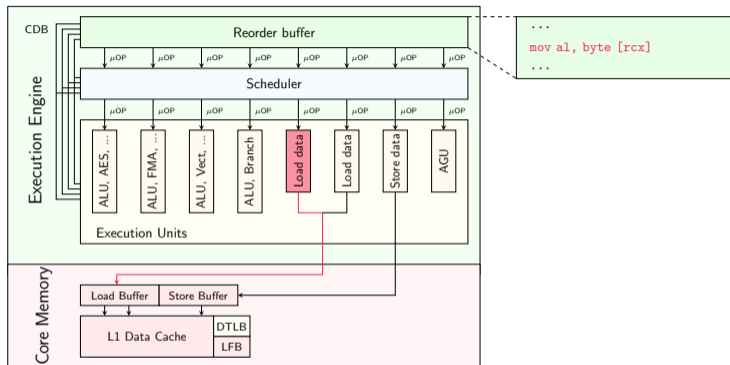


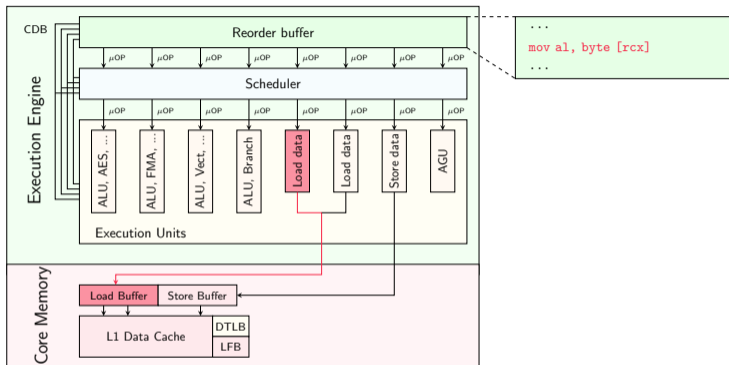


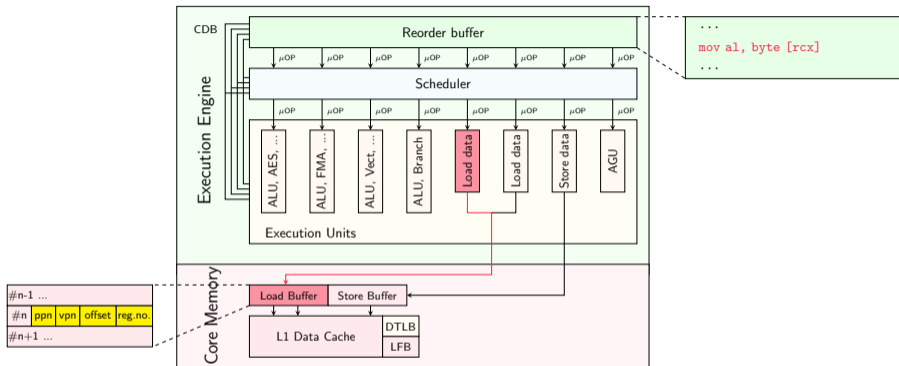


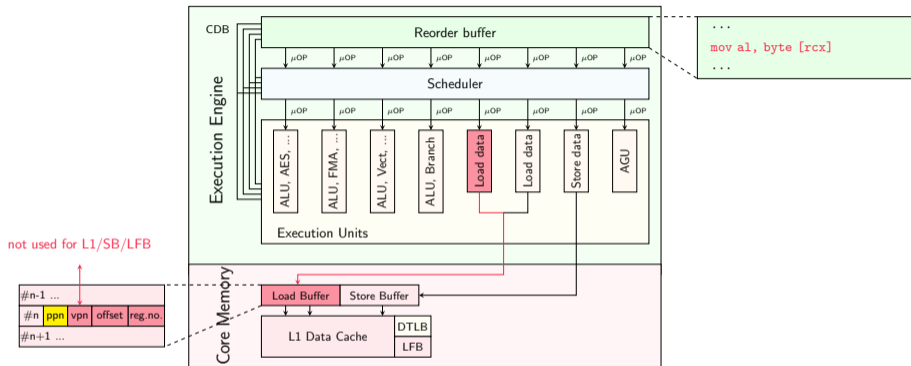


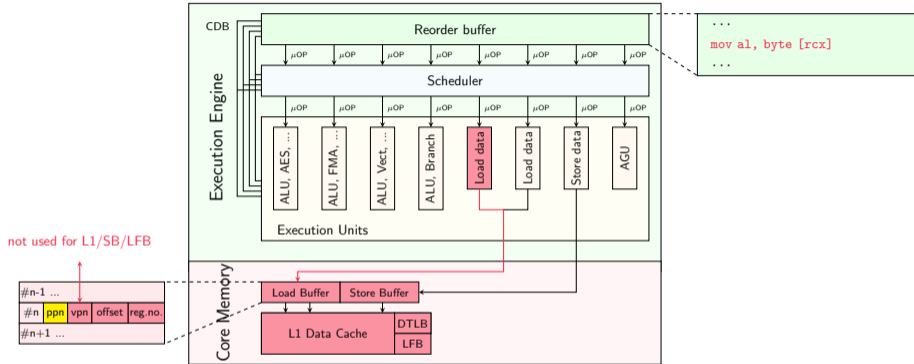


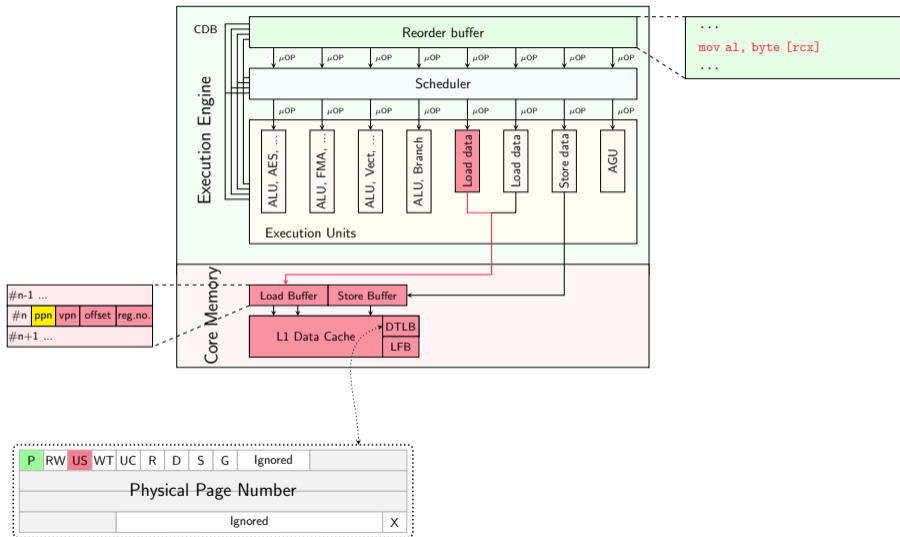


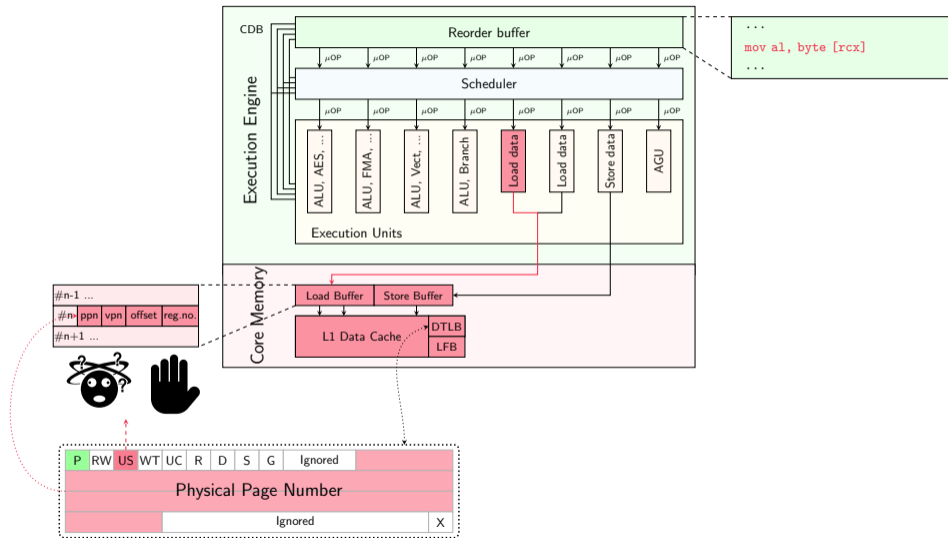


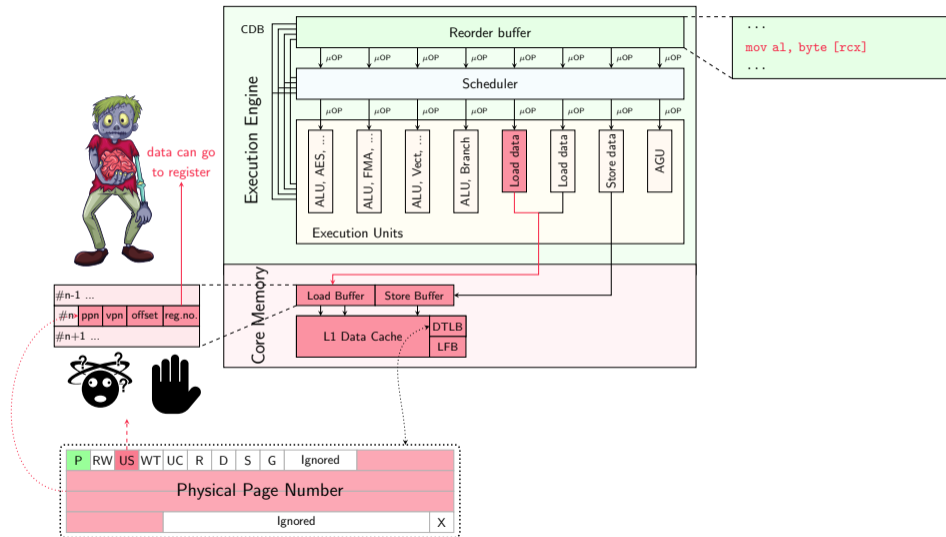














fault occurs load operation completed? "intel corp"



Sign in

All

News

Images

Videos

Shopping

More

Settings

Tools

About 48.600 results (0,29 seconds)

US5724536A - Method and apparatus for blocking execution ...

<https://www.google.com> > patents

The present invention halts the execution of the **load operation** when a dependency ... When the dependency no longer exists, the present invention redispatches the **load operation** so that it **completes**. ... Application filed by **Intel Corp** If a **fault occurs** with respect to the **load operation**, it is marked as valid and **completed**.



Meltdown

Leakage Rate

552.4 kB/s



Meltdown

Leakage Rate

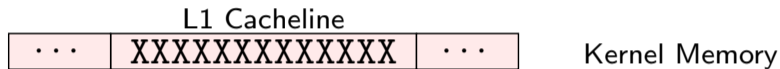
552.4 kB/s

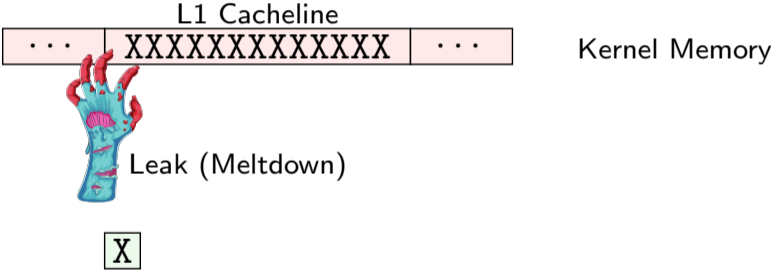


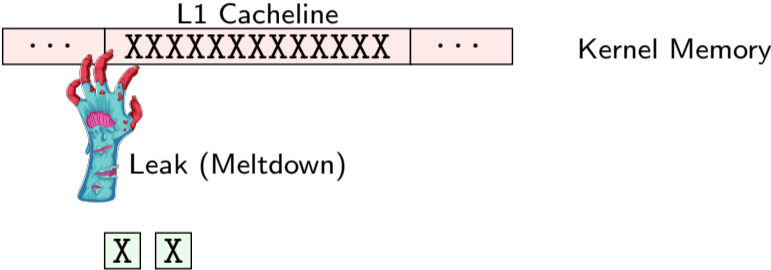
Meltdown

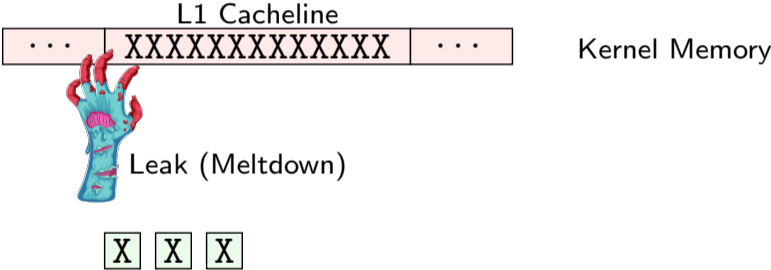
Error Rate

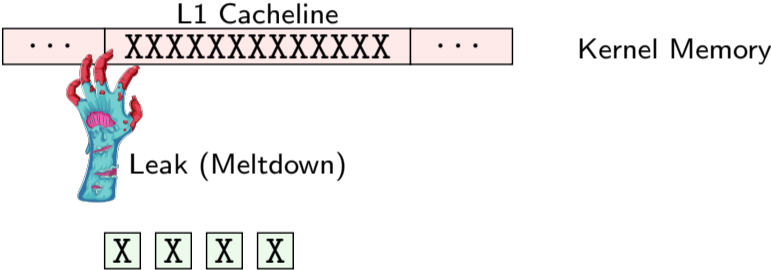
0.003 %

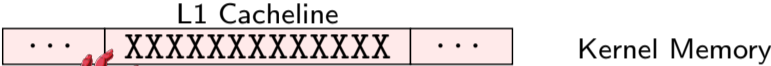






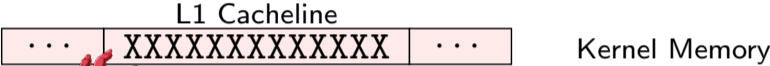






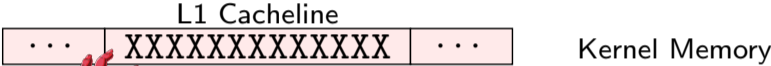
Leak (Meltdown)





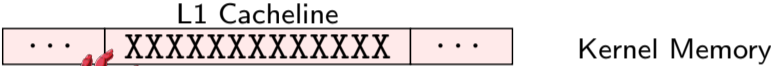
Leak (Meltdown)



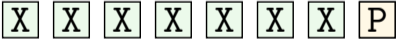


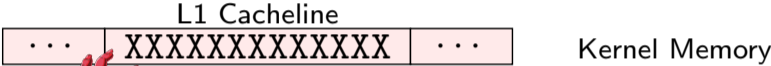
Leak (Meltdown)





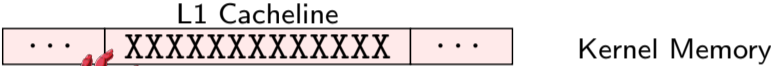
Leak (Meltdown)





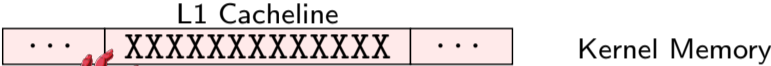
Leak (Meltdown)





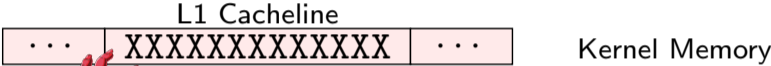
Leak (Meltdown)





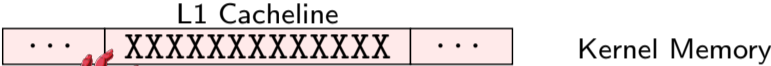
Leak (Meltdown)



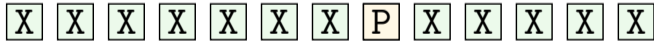


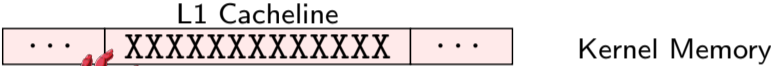
Leak (Meltdown)





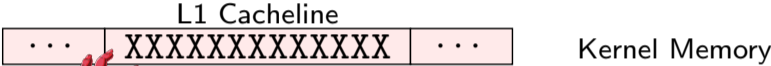
Leak (Meltdown)





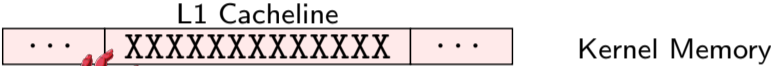
Leak (Meltdown)



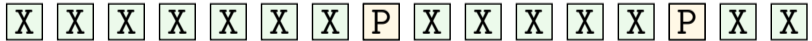


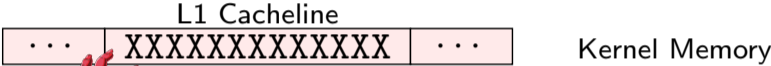
Leak (Meltdown)





Leak (Meltdown)





Leak (Meltdown)





Intel SA-00088

For instance, on some implementations speculatively probing memory will **only pass data** on to subsequent operations if the data is resident in the **lowest level data cache (L1)**. This can allow the data in question to be queried by the malicious application, leading to a side channel that **reveals supervisor data**.



Intel SA-00088

For instance, on some implementations speculatively probing memory will **only pass data** on to subsequent operations if the data is resident in the **lowest level data cache (L1)**. This can allow the data in question to be queried by the malicious application, leading to a side channel that **reveals supervisor data**.

- Works on L3 too ...



Meltdown L3

Leakage Rate

10.4 kB/s



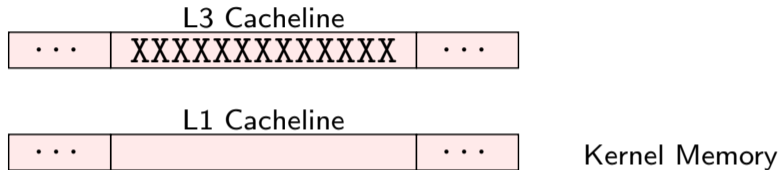
Meltdown L3
Leakage Rate

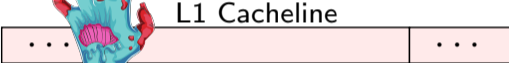
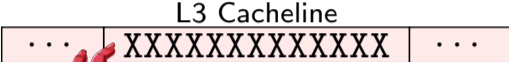
10.4 kB/s



Meltdown L3
Error Rate

0.02 %



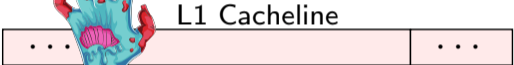
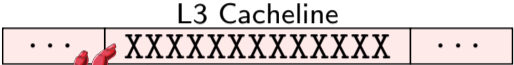


Kernel Memory



Leak (Meltdown)



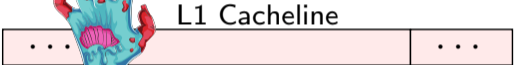
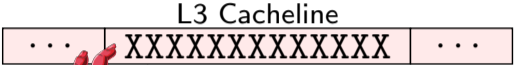


Kernel Memory



Leak (Meltdown)



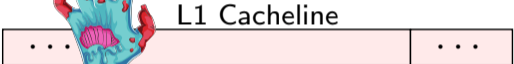
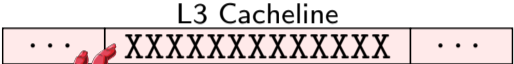


Kernel Memory



Leak (Meltdown)

X	X	P
---	---	---

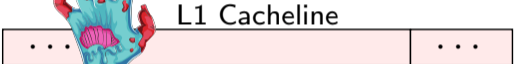
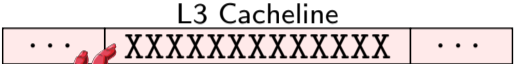


Kernel Memory



Leak (Meltdown)



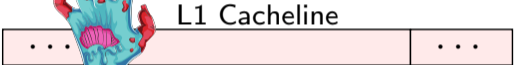
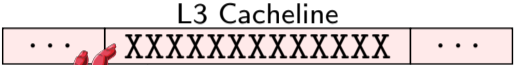


Kernel Memory



Leak (Meltdown)

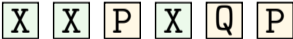


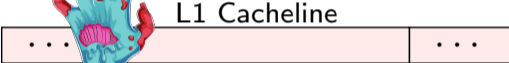
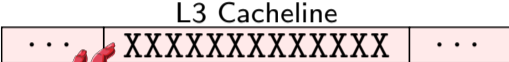


Kernel Memory



Leak (Meltdown)



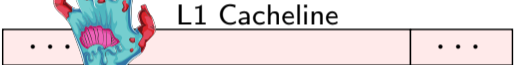
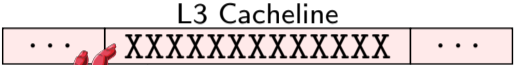


Kernel Memory



Leak (Meltdown)



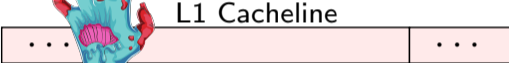
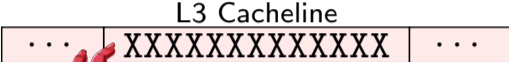


Kernel Memory



Leak (Meltdown)





Kernel Memory

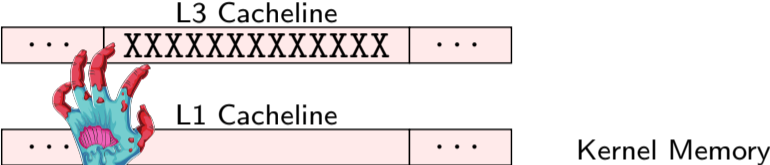


Leak (Meltdown)



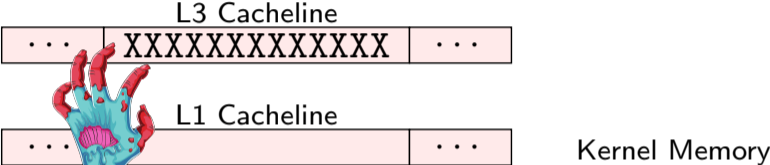


X X P X Q P X X X N



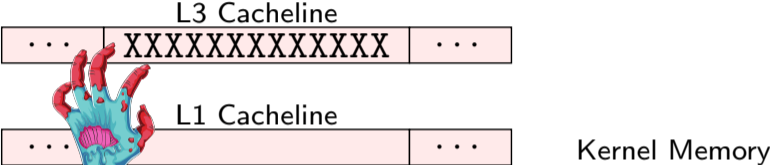
Leak (Meltdown)

X X P X Q P X X X N P



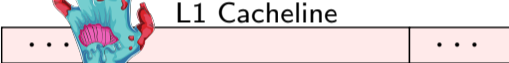
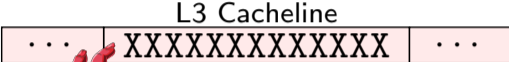
Leak (Meltdown)

X X P X Q P X X X N P N



Leak (Meltdown)

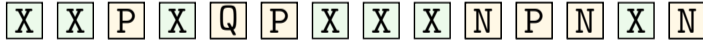
X	X	P	X	Q	P	X	X	X	N	P	N	X
---	---	---	---	---	---	---	---	---	---	---	---	---



Kernel Memory



Leak (Meltdown)

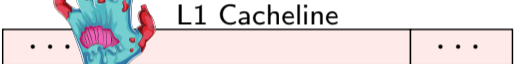
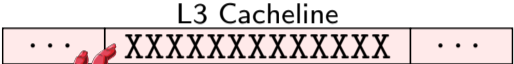




X X P X Q P X X X N P N X N X



X X P X Q P X X X N P N X N X X



Kernel Memory



Leak (Meltdown)





How to get rid of the noise?

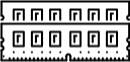



Just repeat the experiment, noise averages out

- BlackHat USA: “Meltdown: Basics, Details, Consequences” (📅 August 9, 2018)

- BlackHat USA: “Meltdown: Basics, Details, Consequences” (📅 August 9, 2018)

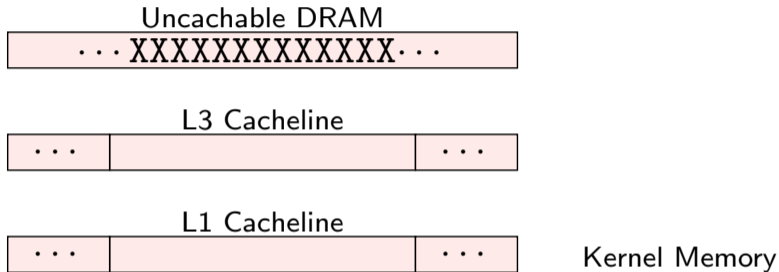
Uncachable memory

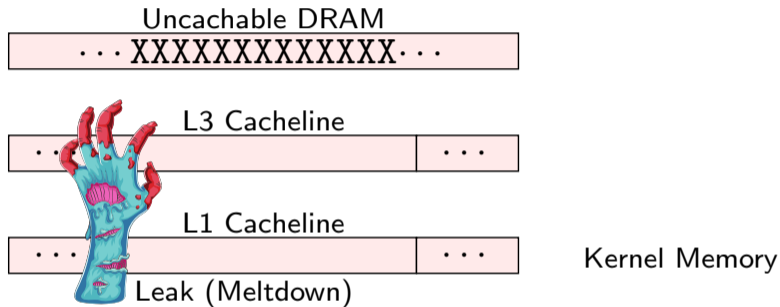


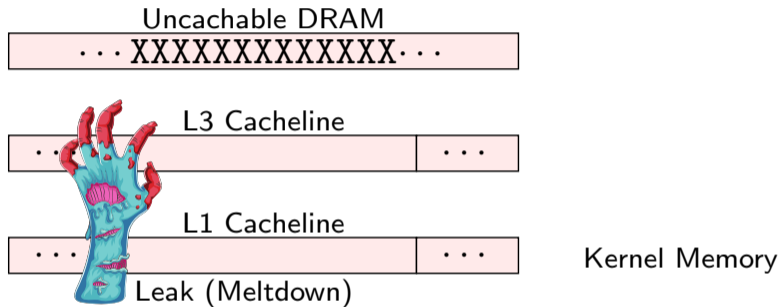
- Mark pages in page tables as UC (uncachable)
 - Every read or write operation will go to main memory
- If the attacker can trigger a legitimate load (system call, ...)
on the same CPU core, the data still can be leaked
- Meltdown might read the data from one of the fill buffers
 - as they are shared between threads running on the same core

39

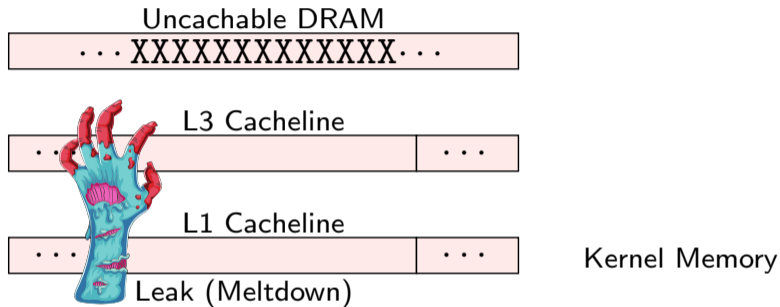
Moritz Lipp, Michael Schwarz, Daniel Gruss | Graz University of Technology



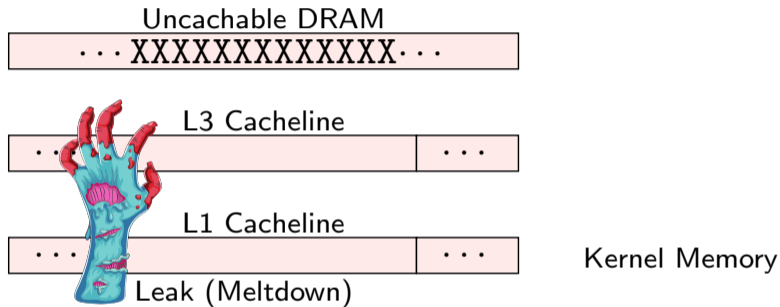




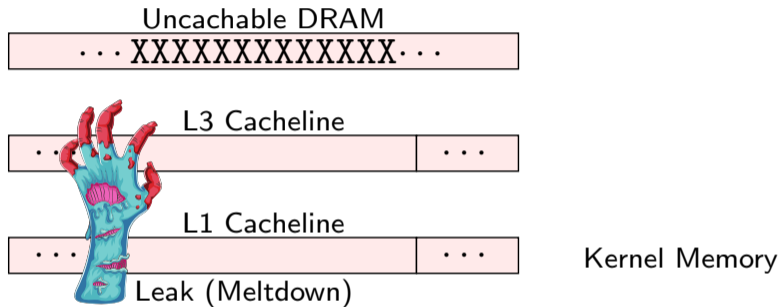
S N



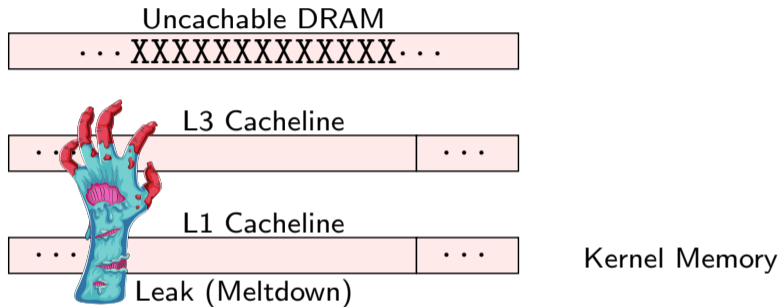
S N P



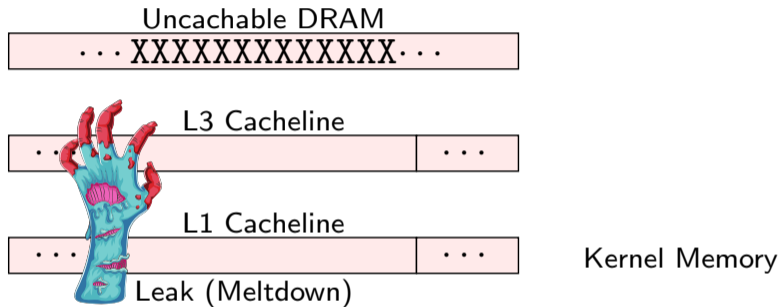
S N P P



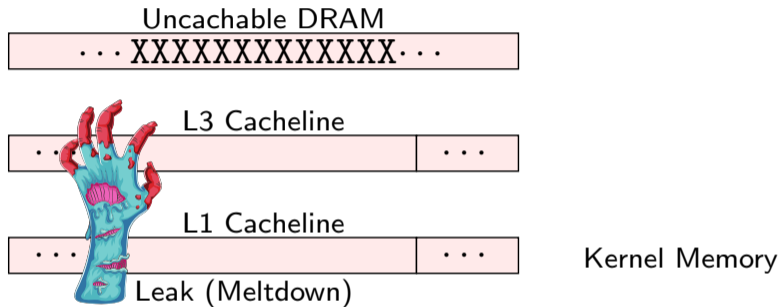
S N P P Q



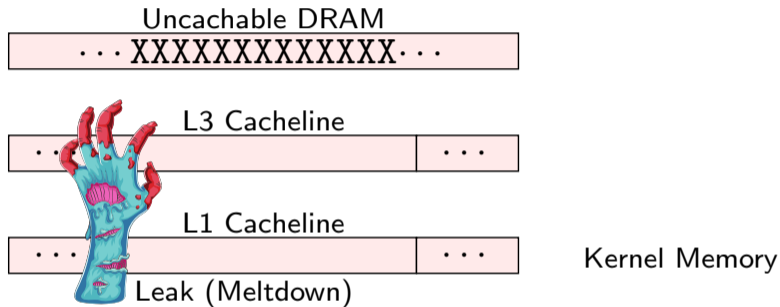
S N P P Q P



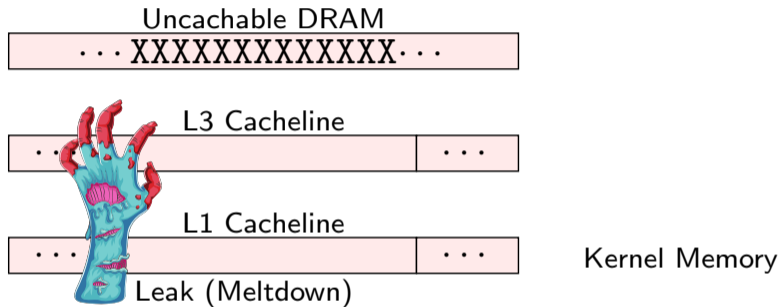
S N P P Q P S



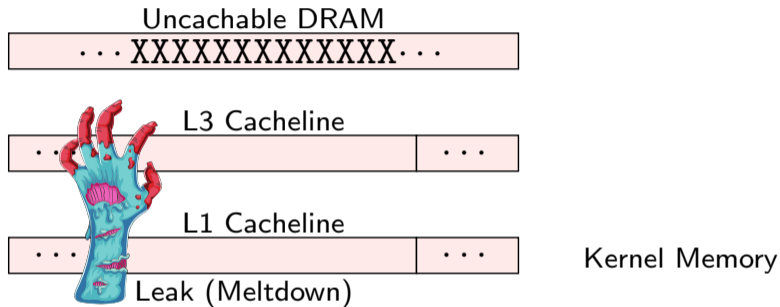
S N P P Q P S N



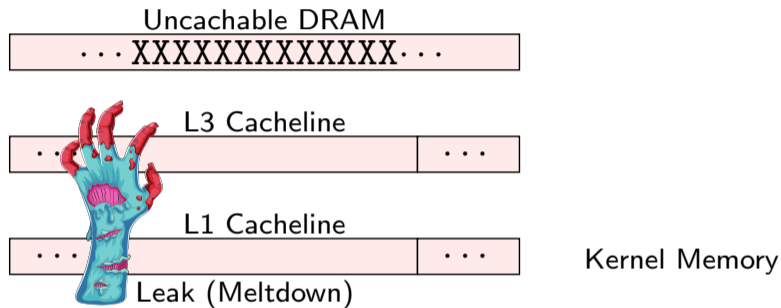
S N P P Q P S N P



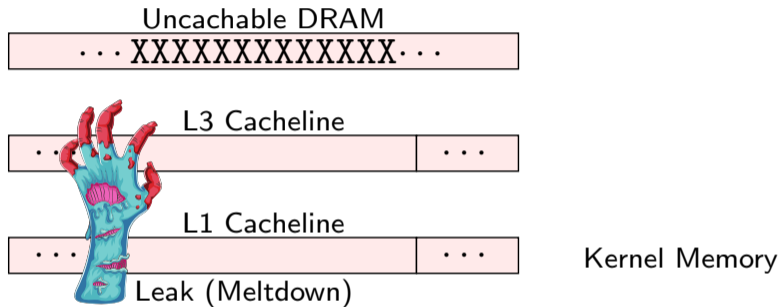
S N P P Q P S N P N



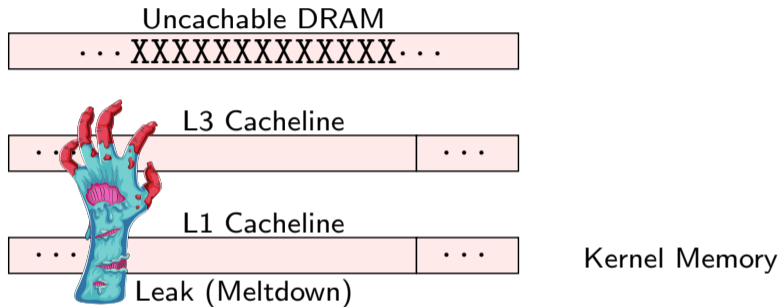
S N P P Q P S N P N P



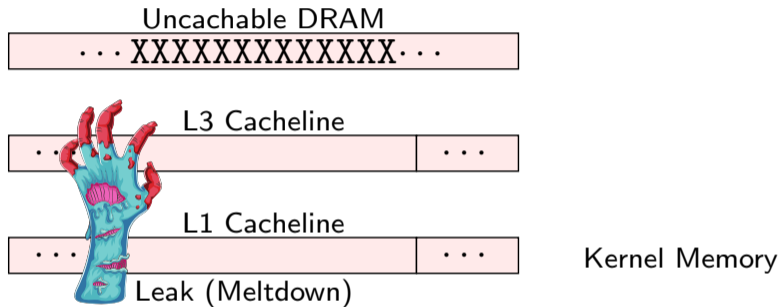
S N P P Q P S N P N P N



S N P P Q P S N P N P N Q

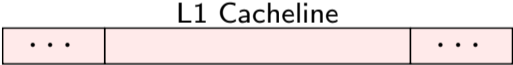
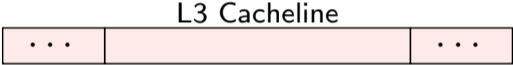
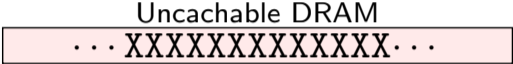


S N P P Q P S N P N P N Q N

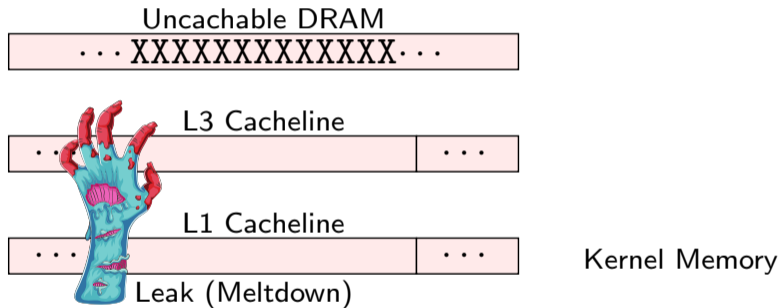


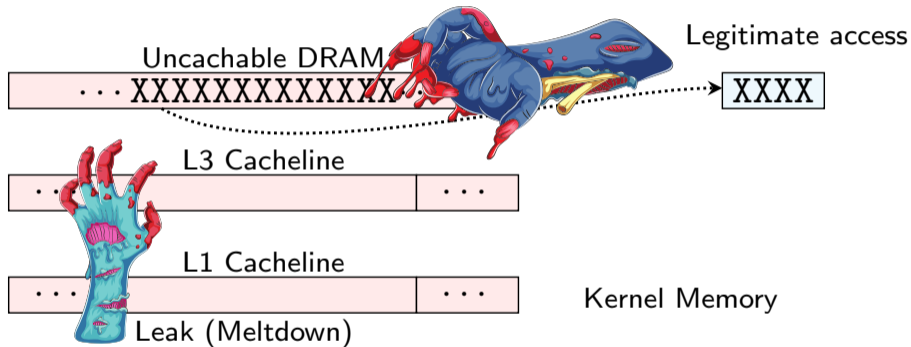
S N P P Q P S N P N P N Q N N



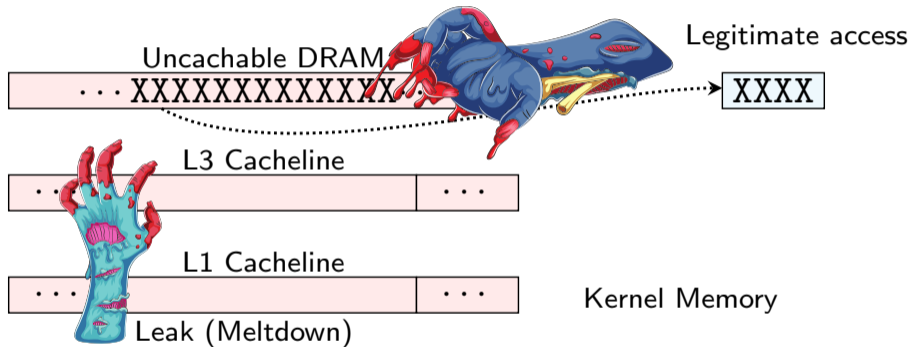


Kernel Memory

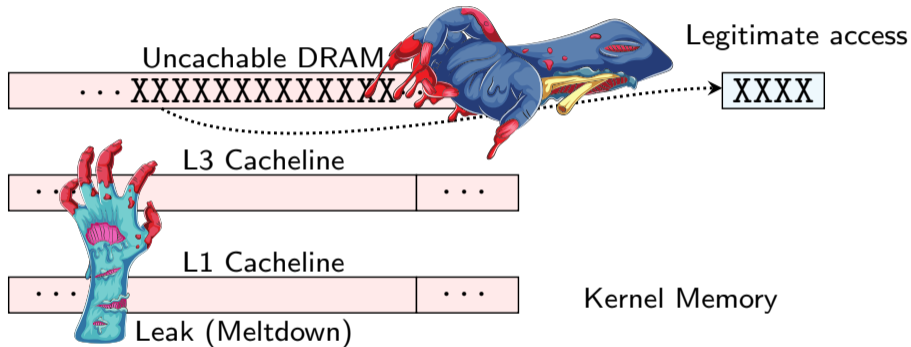




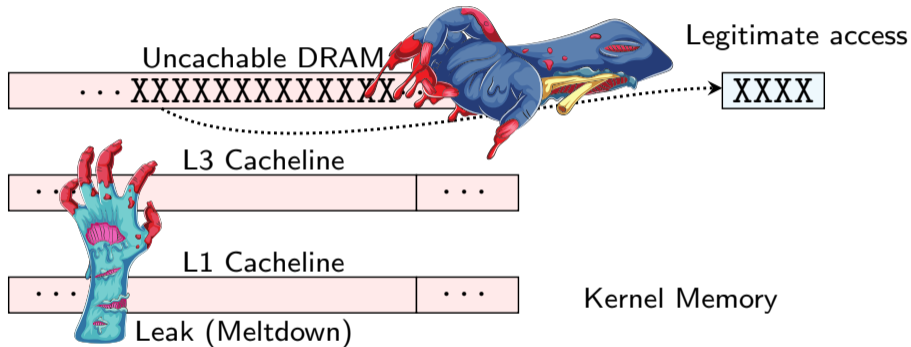
S X



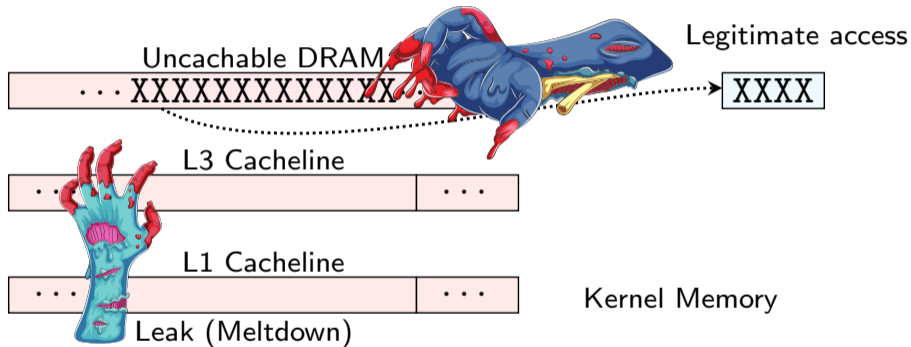
S X X



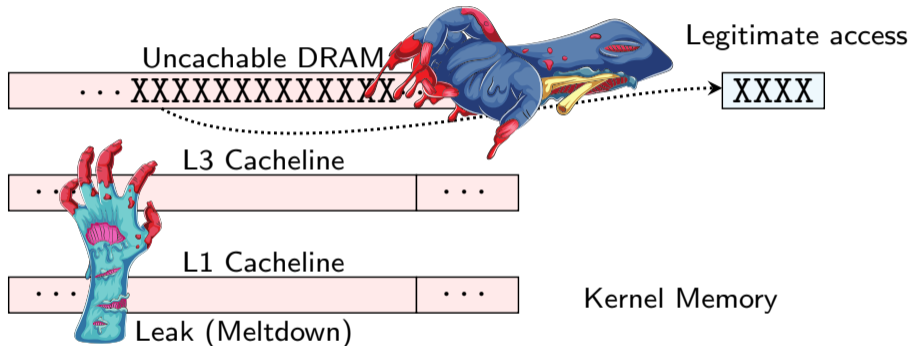
S X X X



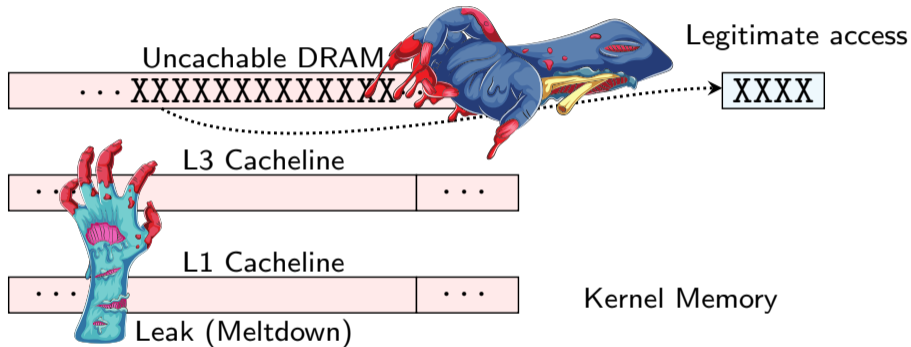
S X X X X



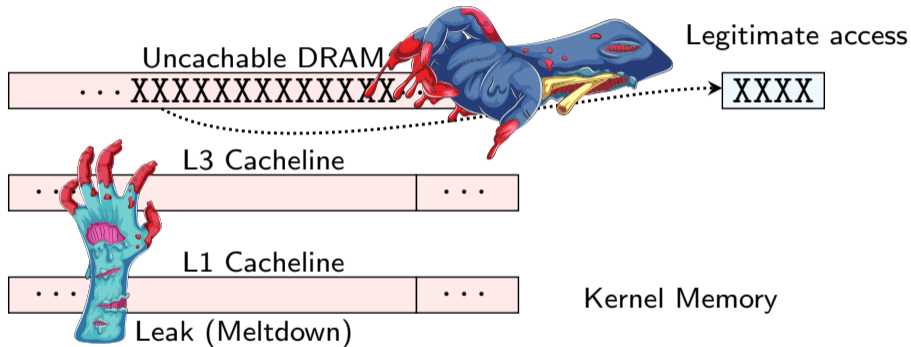
S X X X X P



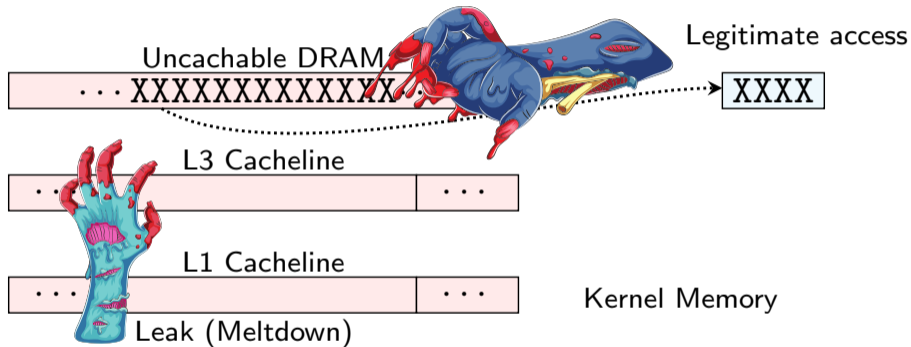
S X X X X P X



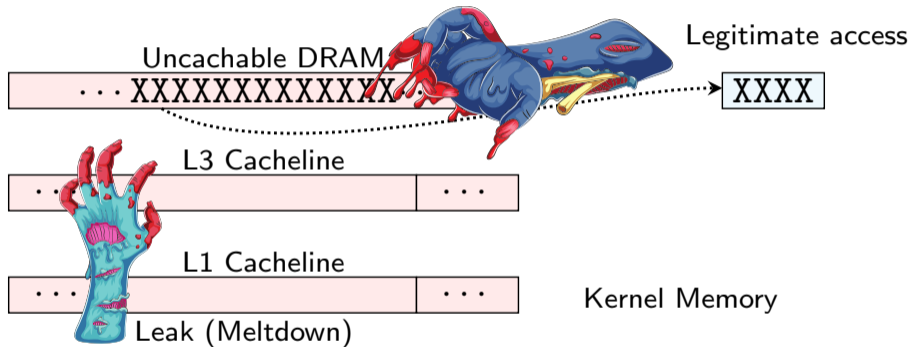
S X X X X P X N



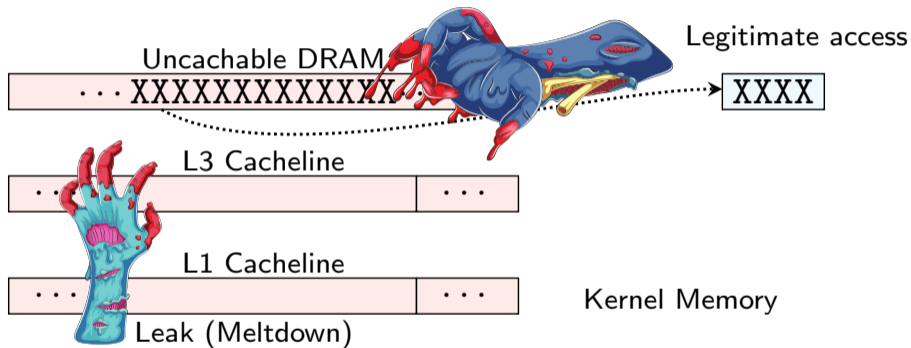
S X X X X P X N P



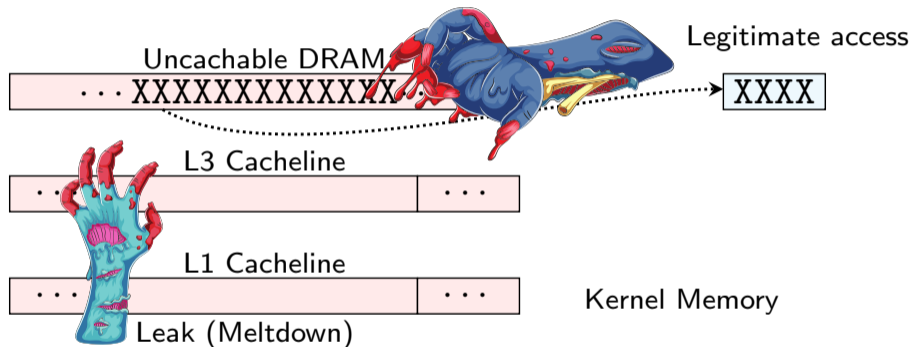
S X X X X P X N P X



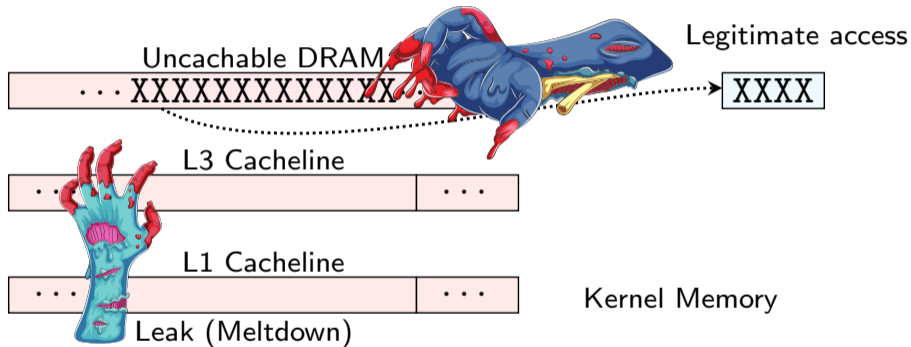
S X X X X P X N P X X



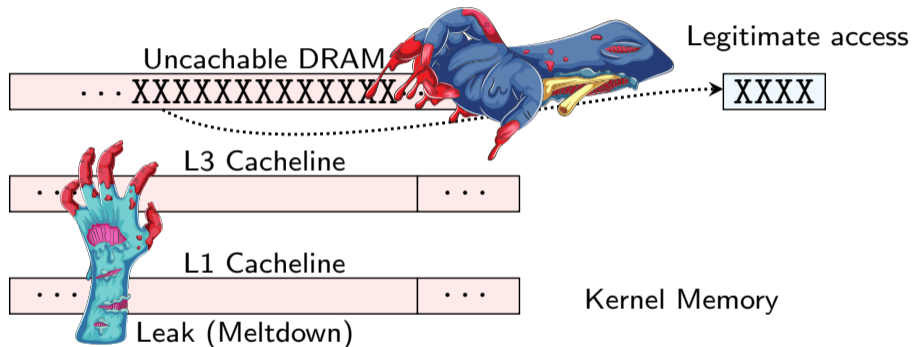
S X X X X P X N P X X N



S X X X X P X N P X X N Q



S X X X X P X N P X X N Q X



S X X X X P X N P X X N Q X X



Meltdown: Reading Kernel Memory from User Space

We suspect that Meltdown reads the value from the **line fill buffers**.



Meltdown: Reading Kernel Memory from User Space

We suspect that Meltdown reads the value from the **line fill buffers**. As the fill buffers are shared between threads running on the same core, the read to the same address within the Meltdown attack could be **served from one of the fill buffers** allowing the attack to succeed.



Meltdown: Reading Kernel Memory from User Space

We suspect that Meltdown reads the value from the **line fill buffers**. As the fill buffers are shared between threads running on the same core, the read to the same address within the Meltdown attack could be **served from one of the fill buffers** allowing the attack to succeed. However, we leave further investigations on this matter open for **future work**.



You always leave
the shitty jobs
to "future you".

Yeah!
Fuck future me.

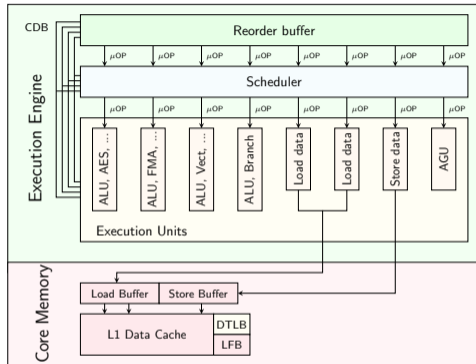


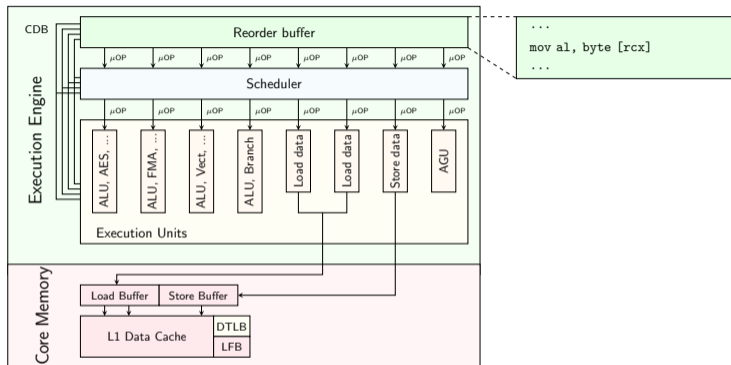
You always leave
the shitty jobs
to "future you".

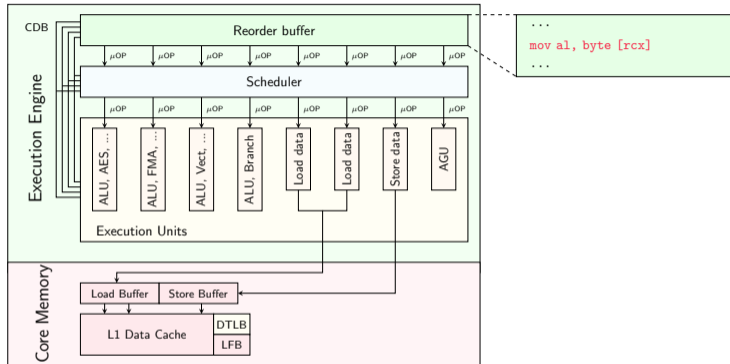
MENTAL

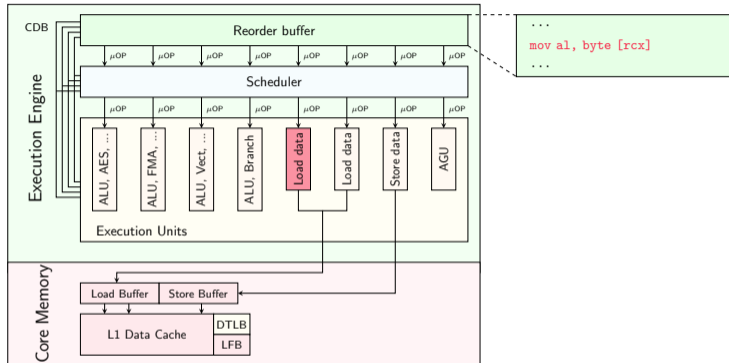
RESOURCE

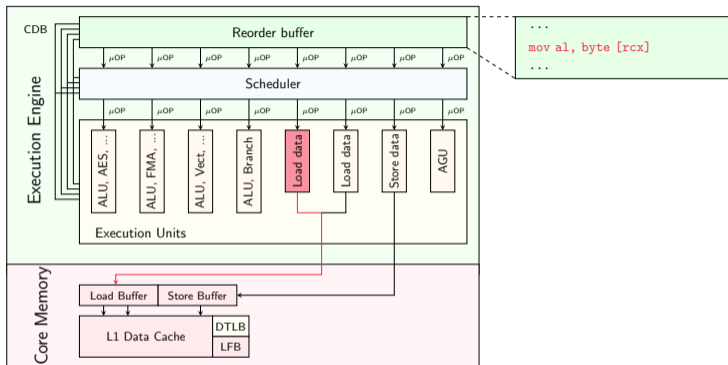
EXHAUSTION

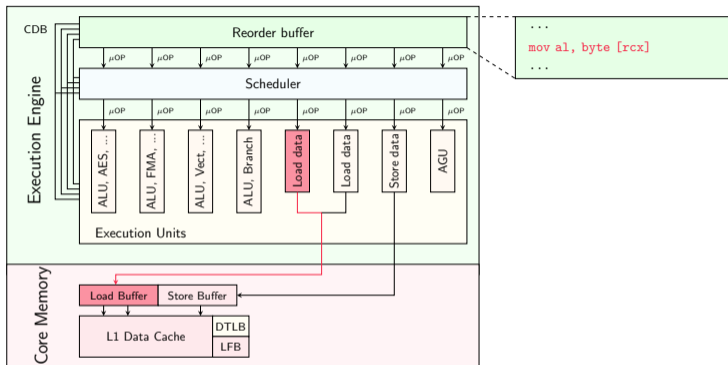


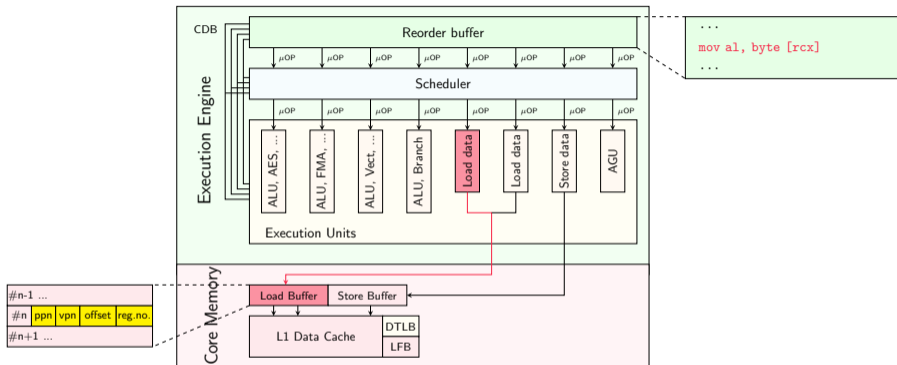


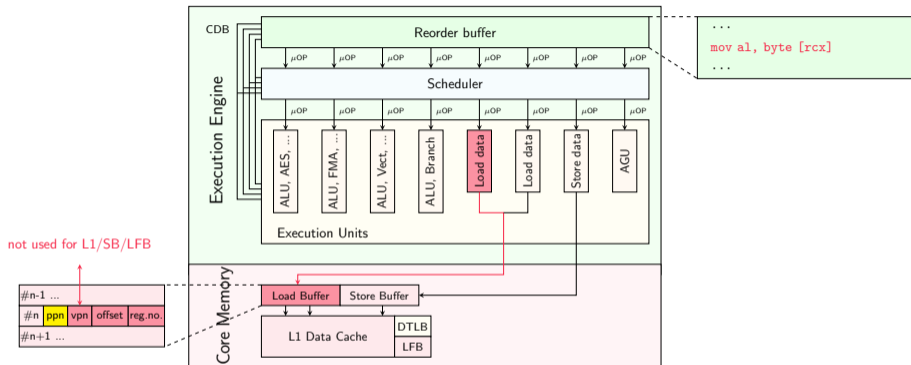


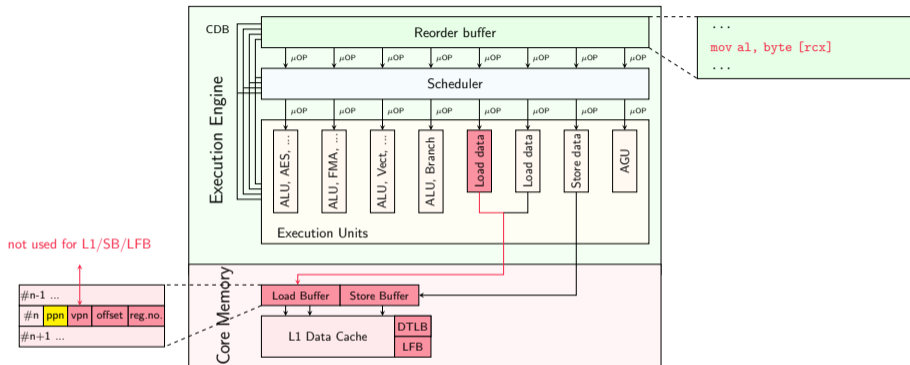


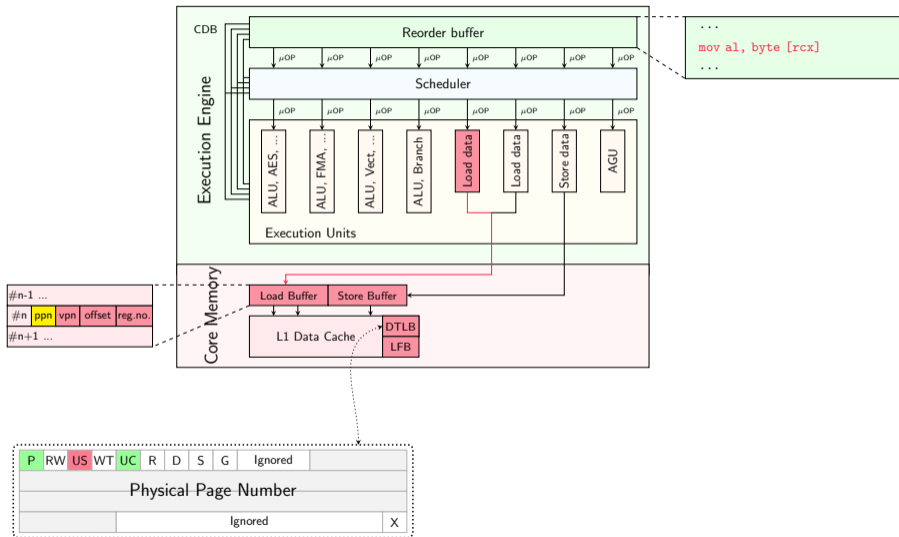


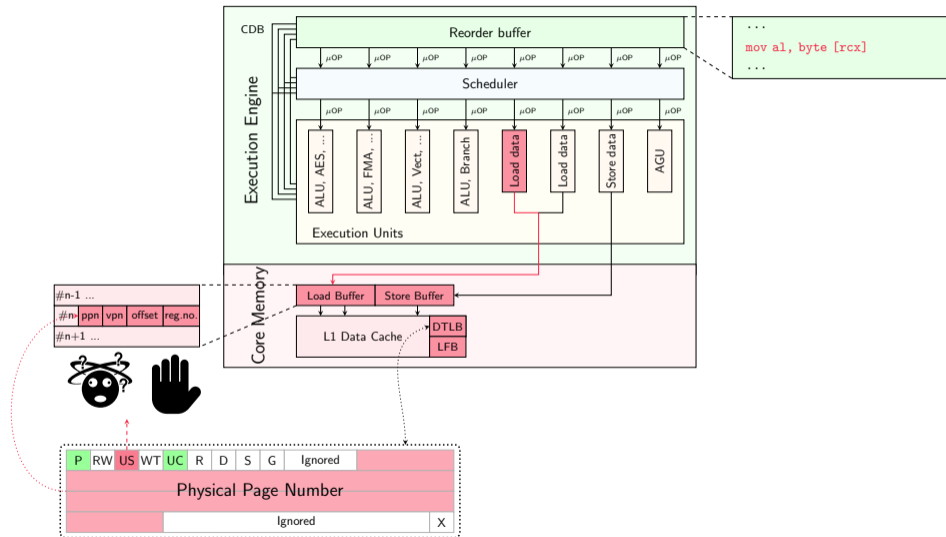






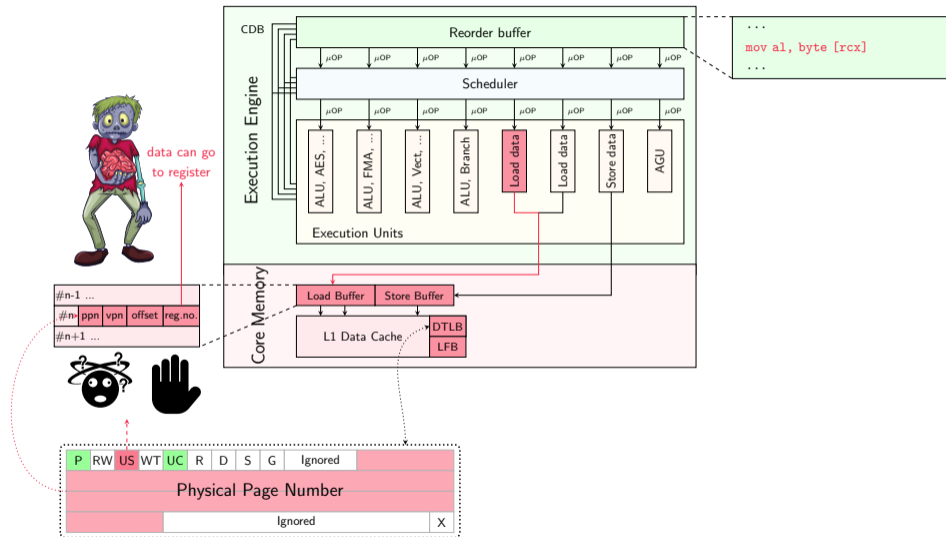






```
...
mov al, byte [rcx]
...
```





BACK
TO **FUTURE**
THE



There is no noise.

Noise is just
someone else's data

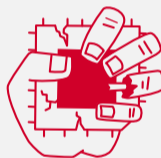
Lemma 1: Noise is someone else's data



Lemma 1: Noise is someone else's data



=



Deep Dive: Intel Analysis of Microarchitectural Data Sampling

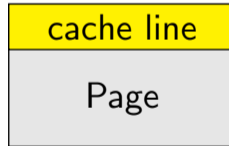
Fill buffers may retain stale data from **prior memory requests** until a new memory request overwrites the fill buffer.

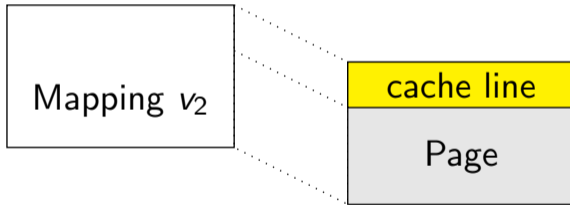
Deep Dive: Intel Analysis of Microarchitectural Data Sampling

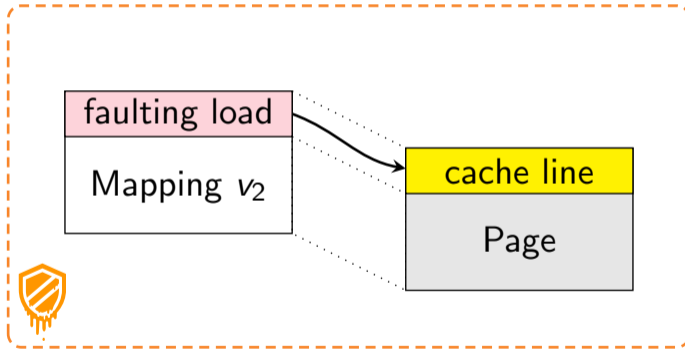
Fill buffers may retain stale data from **prior memory requests** until a new memory request overwrites the fill buffer. Under **certain conditions**, the fill buffer may speculatively **forward data**, including stale data,

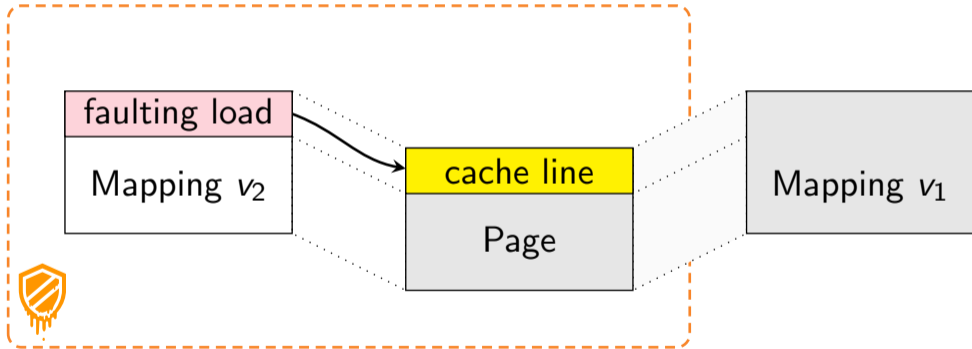
Deep Dive: Intel Analysis of Microarchitectural Data Sampling

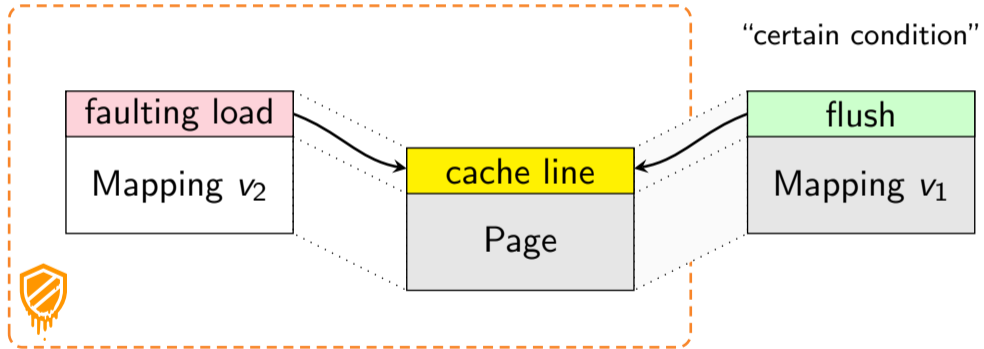
Fill buffers may retain stale data from **prior memory requests** until a new memory request overwrites the fill buffer. Under **certain conditions**, the fill buffer may speculatively **forward data**, including stale data, to a load operation that will cause a **fault/assist**.

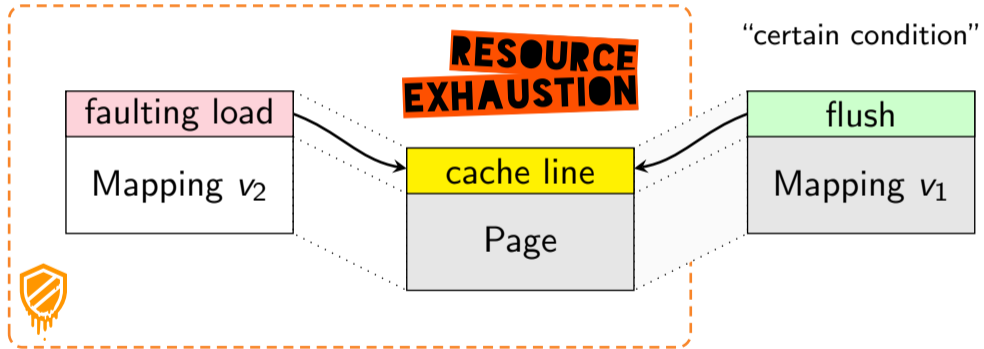


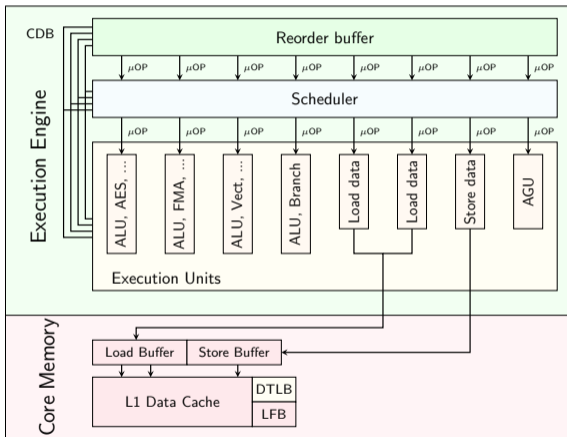


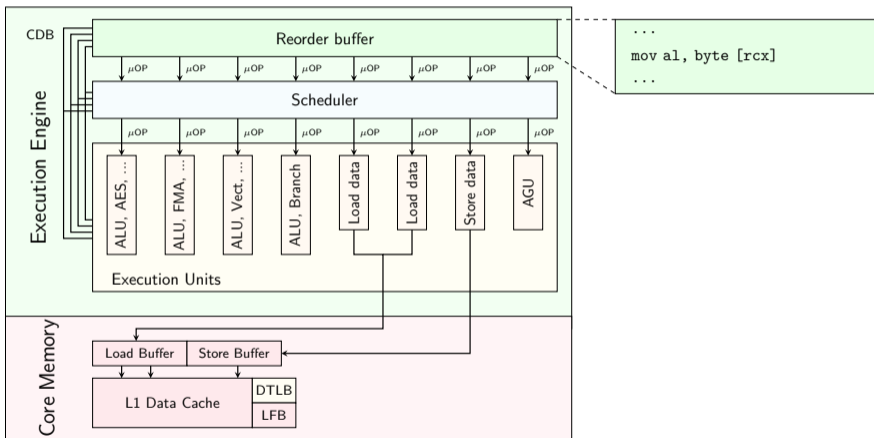


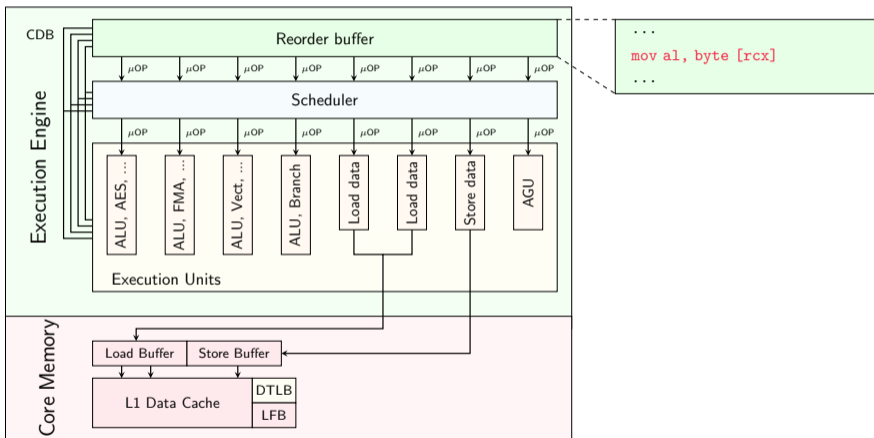


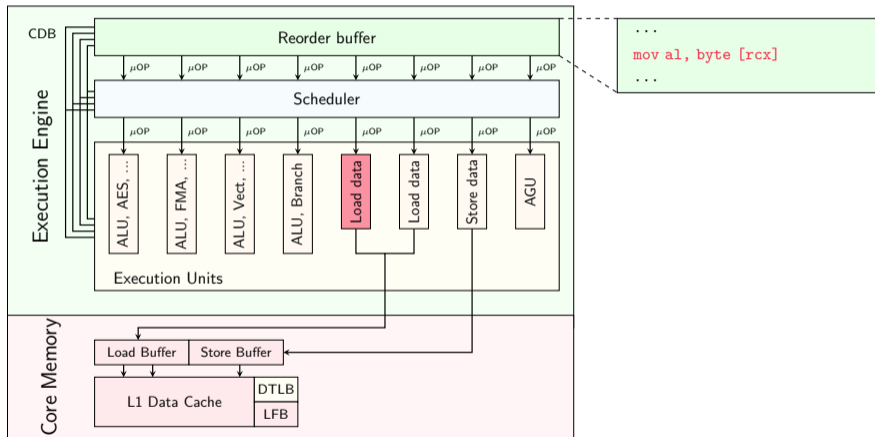


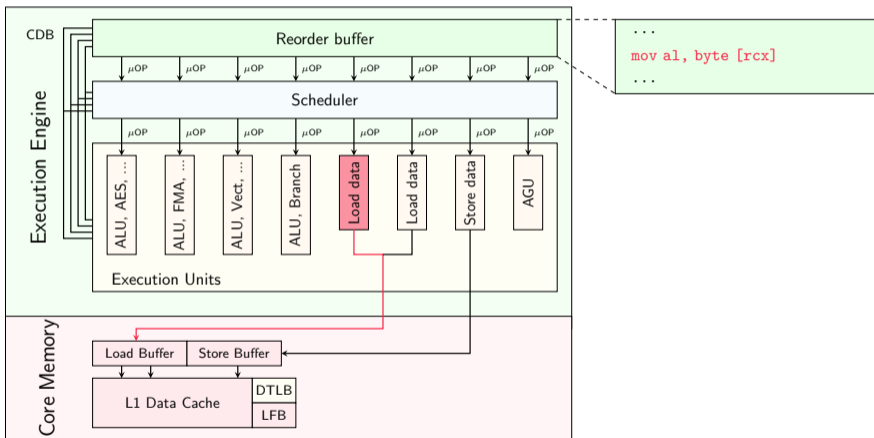


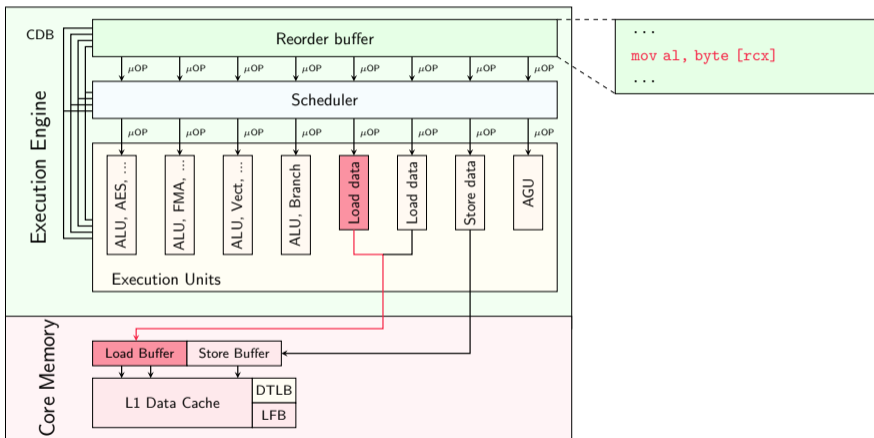


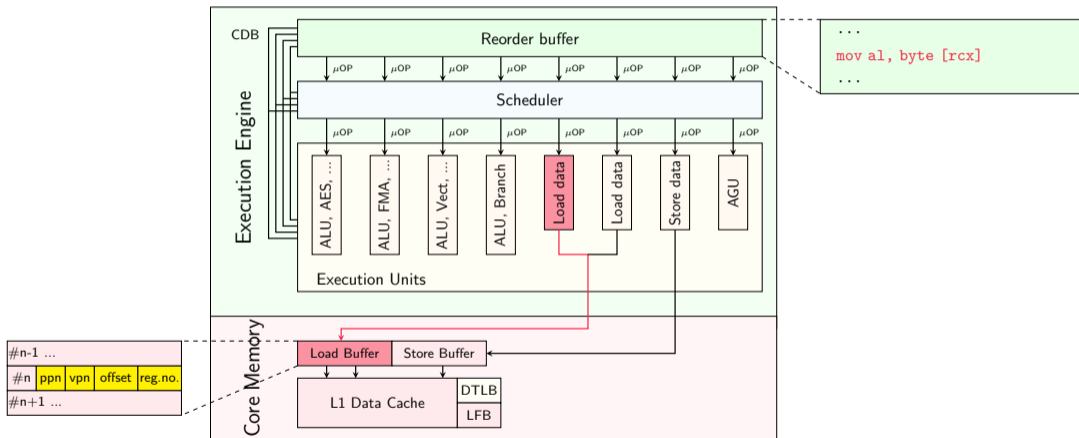


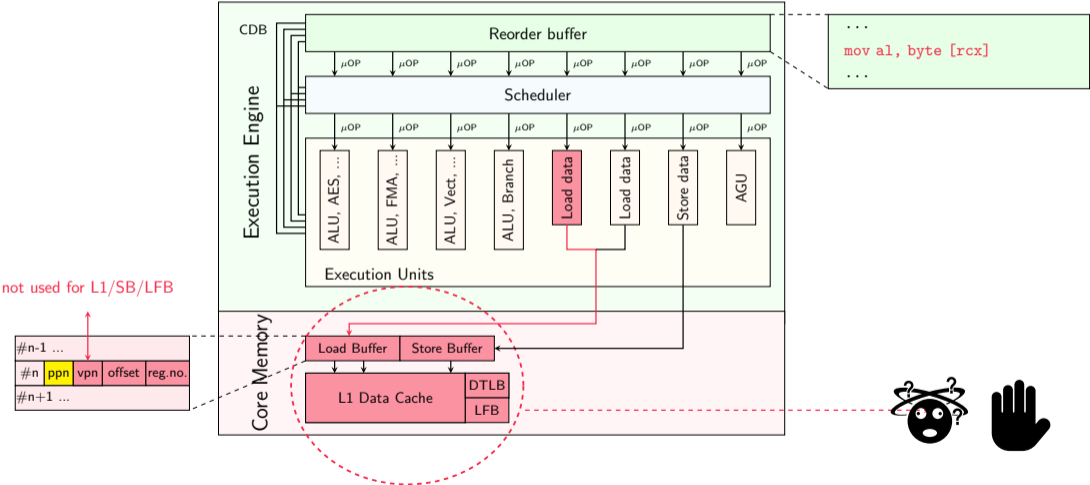


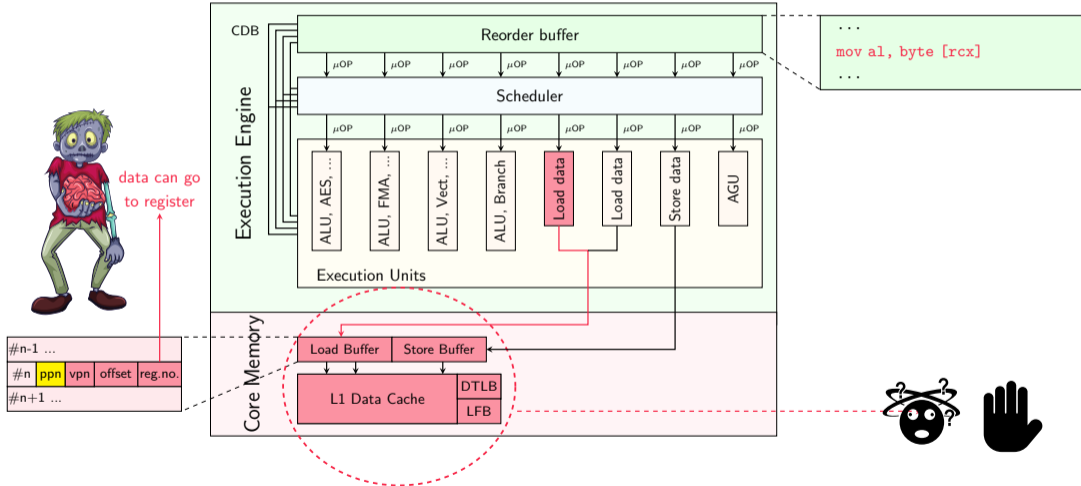








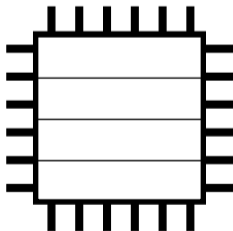




User Memory

	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

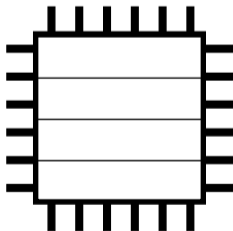
```
char value = faulting[0]
```



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

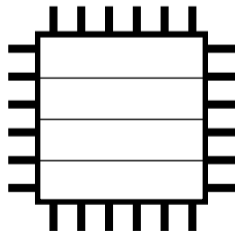
```
char value = faulting[0]
```



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]
```



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

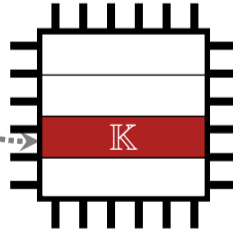
```
char value = faulting[0]
```

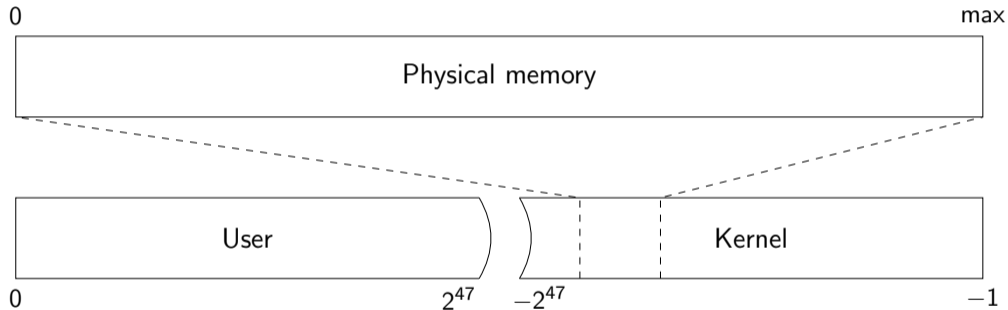
```
mem[value]
```

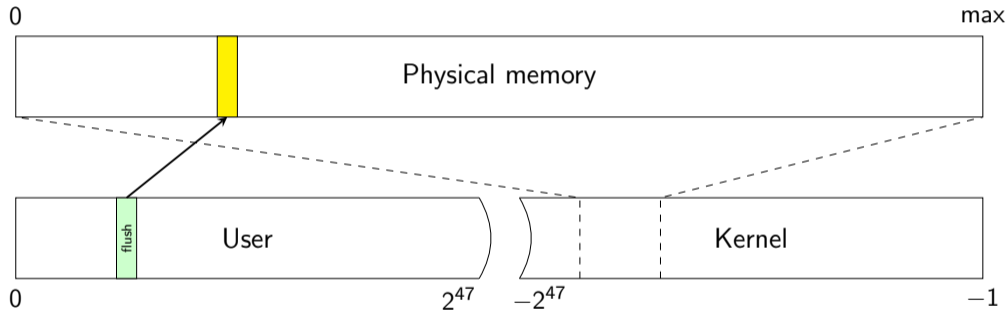
K

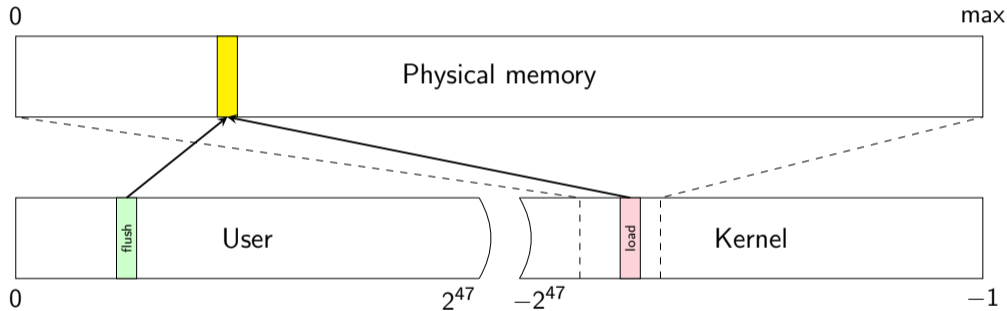
⚡ Fault

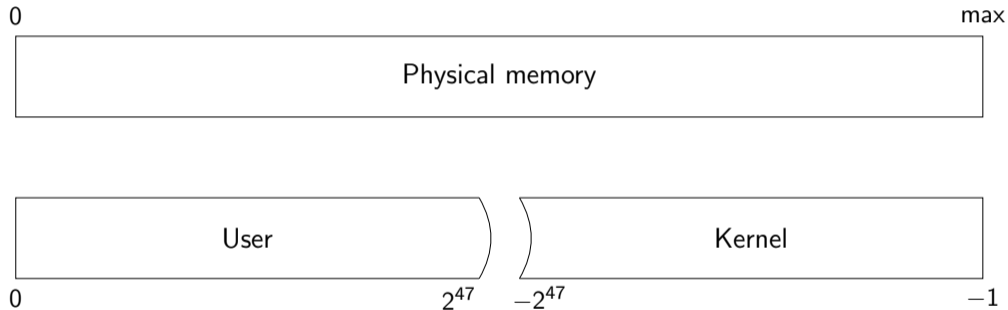
Out of order

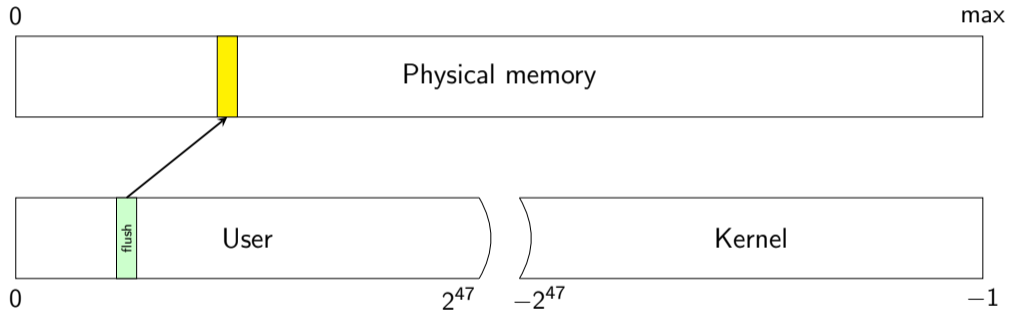


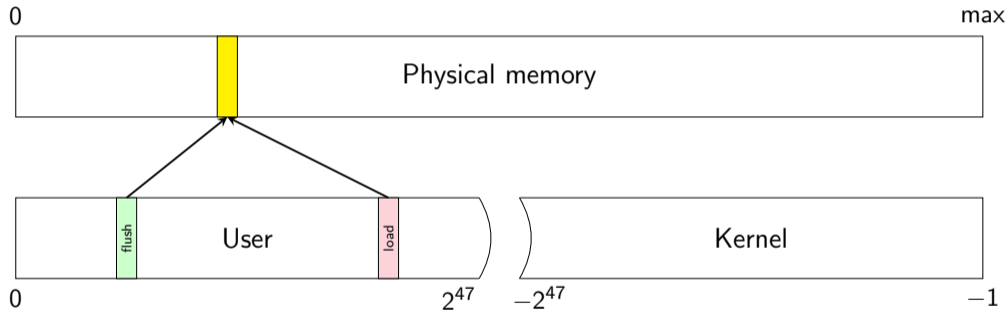




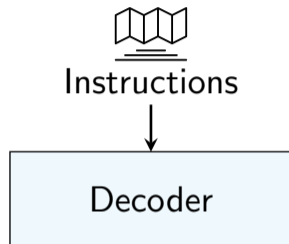


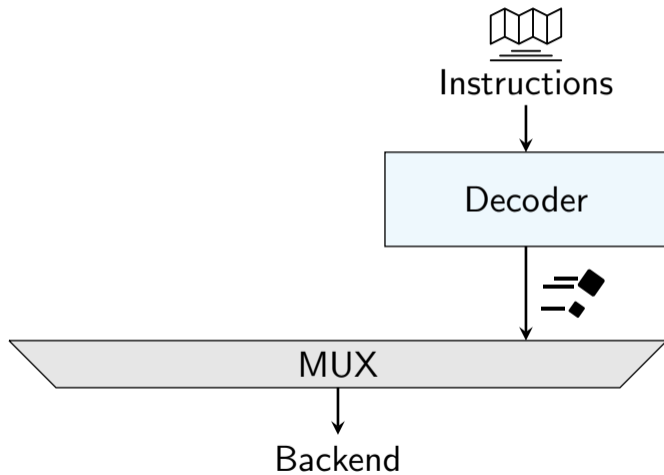


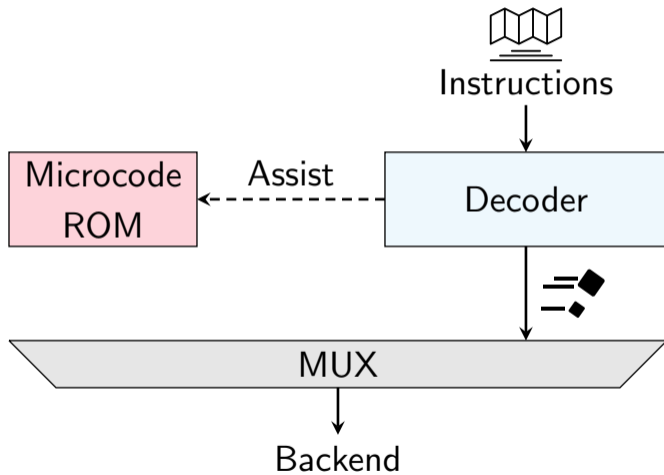


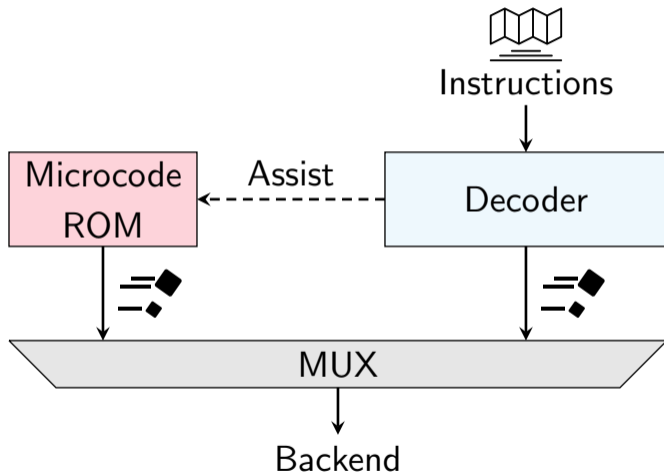


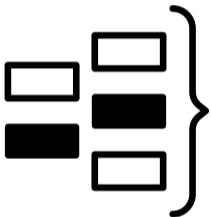




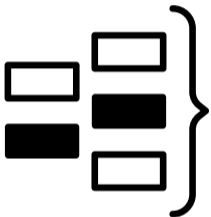




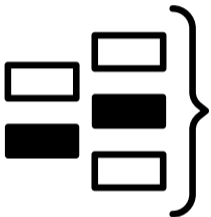




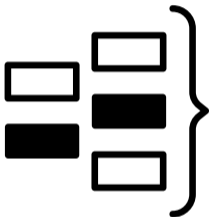
- Microcode assist handles **rare cases**



- Microcode assist handles **rare cases**
- Microarchitectural fault

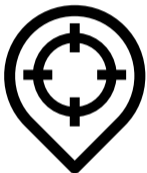


- Microcode assist handles **rare cases**
- Microarchitectural fault
- Setting **accessed**/dirty **bit** in page table



- Microcode assist handles **rare cases**
- Microarchitectural fault
- Setting **accessed/dirty bit** in page table
- Regularly reset on Windows

- Leak data on **same** and **sibling** hyperthread





- Leak data on **same** and **sibling** hyperthread



Applications

- Leak data on **same** and **sibling** hyperthread



Applications



Operating System

- Leak data on **same** and **sibling** hyperthread



Applications



Operating System



SGX Enclave

- Leak data on **same** and **sibling** hyperthread



Applications



Operating System



SGX Enclave



Virtual Machine

- Leak data on **same** and **sibling** hyperthread



Applications



Operating System



SGX Enclave



Virtual Machine



Hypervisor

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

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

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

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

IMPOSSIBLE

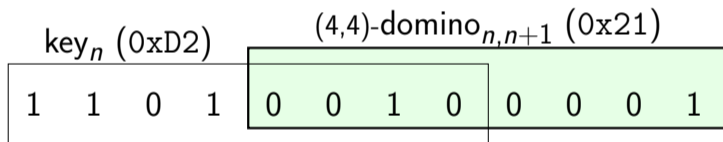


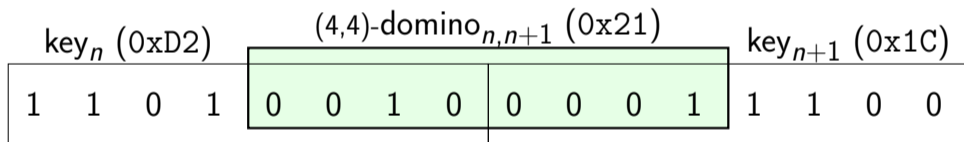
IMPOSSIBLE



key_n (0xD2)

1	1	0	1	0	0	1	0
---	---	---	---	---	---	---	---







DEMO



Variant 1

Kernel Mapping



Variant 3

Microcode-Assisted Page-Table Walk



works
 does not work
 can be prevented



Variant 1
Kernel Mapping

5.30 kB/s



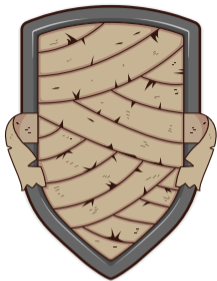
Variant 1
Kernel Mapping

5.30 kB/s

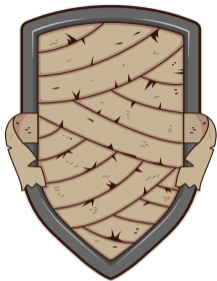


Variant 3
Microcode-Assisted
Page-Table Walk

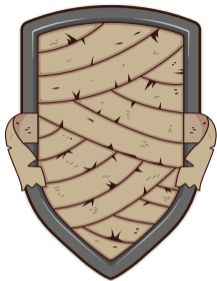
7.73 kB/s



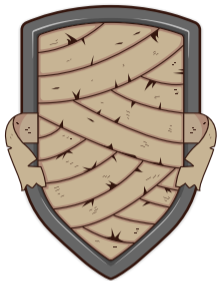
- **Disable hyperthreading** or group scheduling



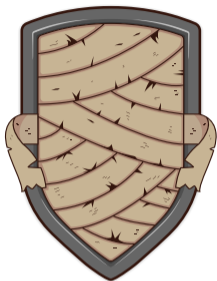
- **Disable hyperthreading** or group scheduling
- **Overwrite** microarchitectural buffers



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



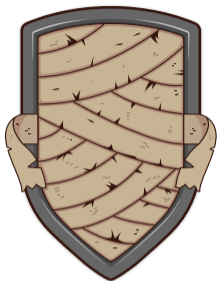
- **Disable hyperthreading** or group scheduling
- **Overwrite** microarchitectural buffers
 - VERW instruction (microcode update)
 - Software sequences



- **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	✘	✘	✘	✘	✘	✘

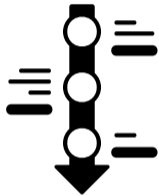
<https://www.intel.com/content/www/us/en/architecture-and-technology/engineering-new-protections-into-hardware.html>



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

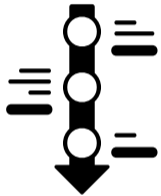
CPU	Meltdown	Foreshadow	RIDL	Fallout	MLPDS	MDSUM	ZombieLoad
8th/9th gen. Intel Core Coffee Lake	✘	✘	✘	✘	✘	✘	???
Intel Xeon Cascade Lake	✘	✘	✘	✘	✘	✘	???

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



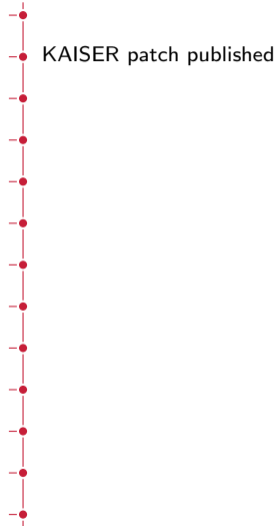
2017

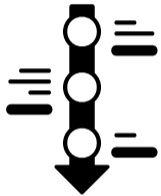




2017

May 4





2017

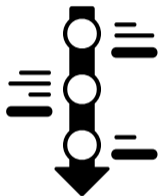
May 4

June 22



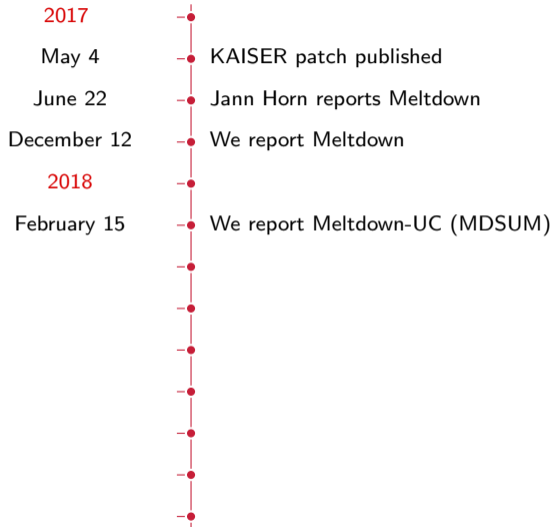
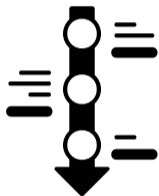
KAISER patch published

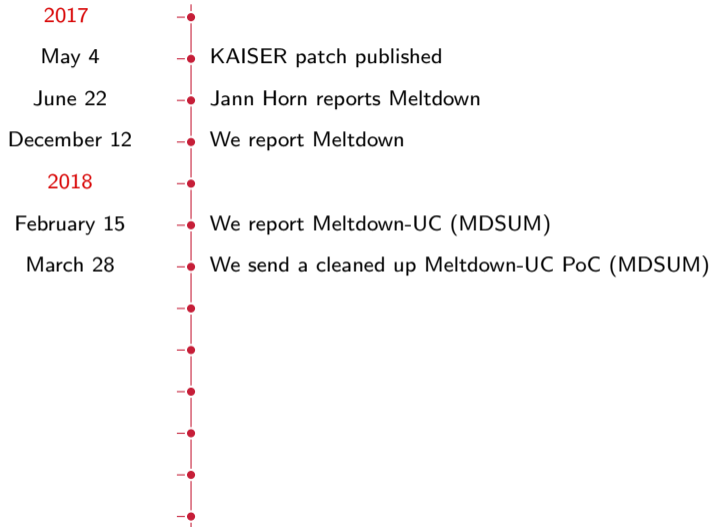
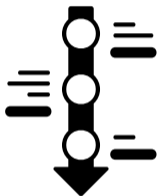
Jann Horn reports Meltdown

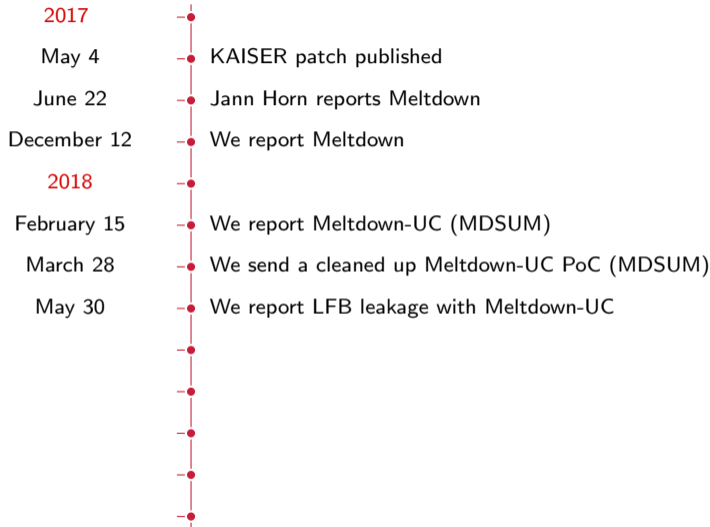
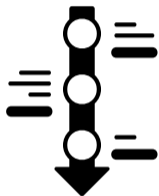


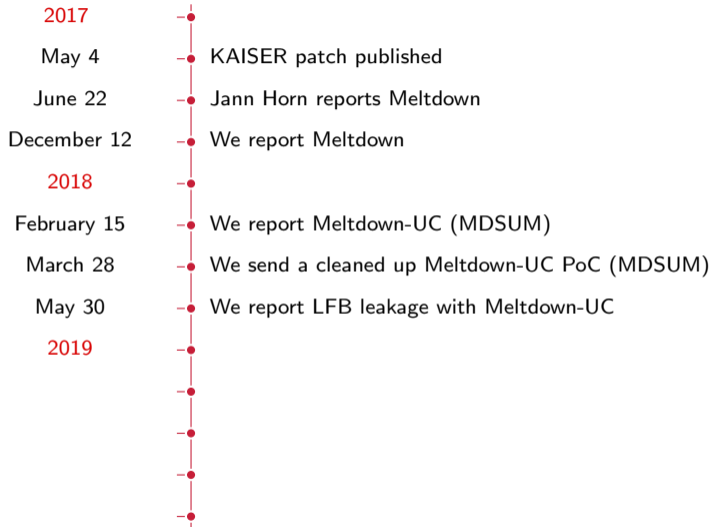
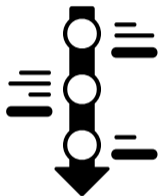
- 2017
- May 4 KAISER patch published
- June 22 Jann Horn reports Meltdown
- December 12 We report Meltdown

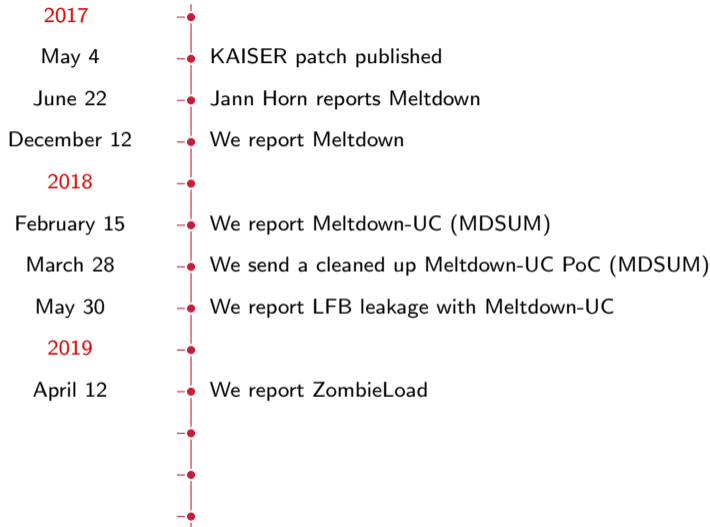
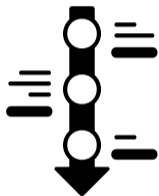


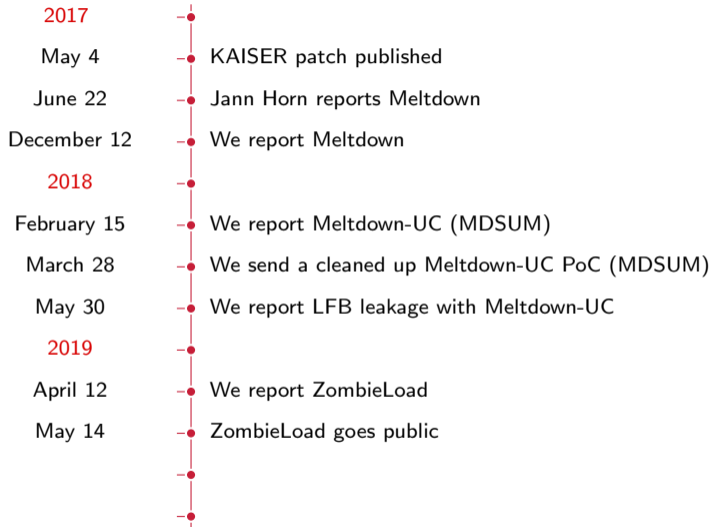
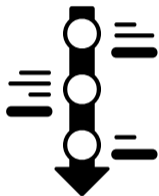


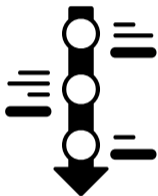












2017	-●
May 4	-● KAISER patch published
June 22	-● Jann Horn reports Meltdown
December 12	-● We report Meltdown
2018	-●
February 15	-● We report Meltdown-UC (MDSUM)
March 28	-● We send a cleaned up Meltdown-UC PoC (MDSUM)
May 30	-● We report LFB leakage with Meltdown-UC
2019	-●
April 12	-● We report ZombieLoad
May 14	-● ZombieLoad goes public
May 14	-● ZombieLoad-resistant CPUs announced
	-●









Which ZombieLoad Variant works despite MDS mitigations?

◆ A:

◆ B:

◆ C:

◆ D:

Which ZombieLoad Variant works despite MDS mitigations?

◆ **A:** Variant 1

◆ **B:**

◆ **C:**

◆ **D:**

Which ZombieLoad Variant works despite MDS mitigations?

◆ **A:** Variant 1

◆ **B:** Variant 3

◆ **C:**

◆ **D:**

Which ZombieLoad Variant works despite MDS mitigations?

◆ **A:** Variant 1

◆ **B:** Variant 3

◆ **C:** Variant 2

◆ **D:**

Which ZombieLoad Variant works despite MDS mitigations?

◆ **A:** Variant 1

◆ **B:** Variant 3

◆ **C:** Variant 2

◆ **D:** None

Which ZombieLoad Variant works despite MDS mitigations?

◆ **A:** Variant 1

◆ **B:** Variant 3

◆ **C:** Variant 2

◆ **D:** None

Which ZombieLoad Variant works despite MDS mitigations?

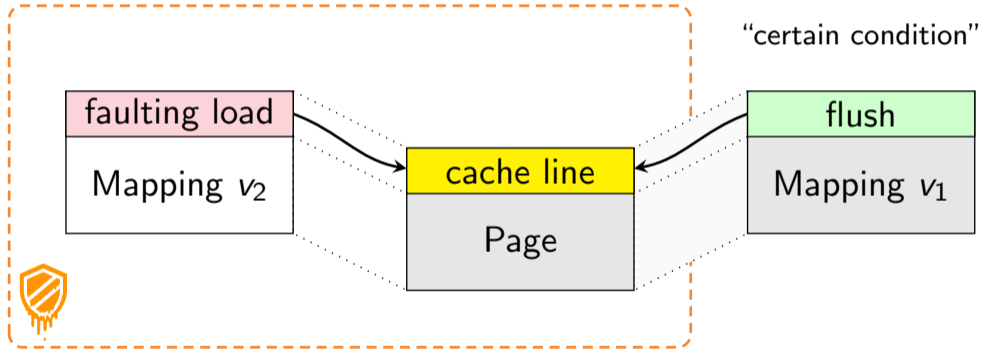
◆ **A:** Variant 1

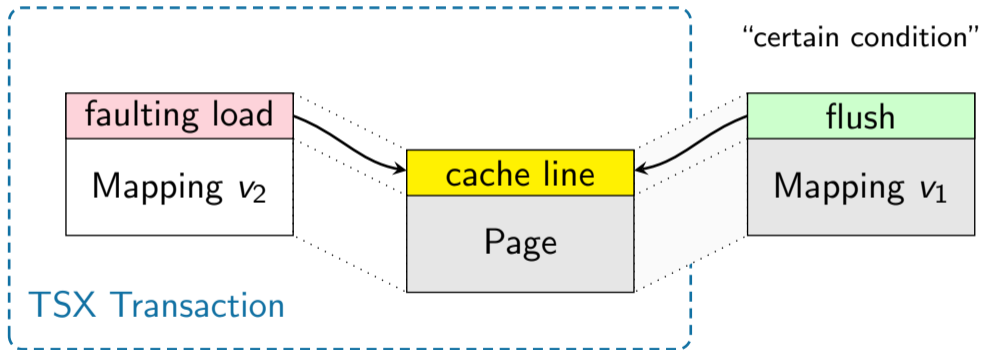
◆ **B:** Variant 3

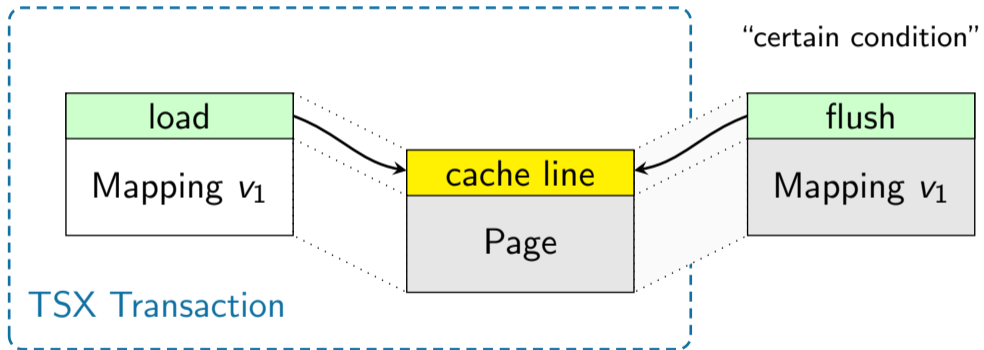
◆ **C:** Variant 2

◆ **D:** None











```
// Variant 2
```

```
flush(mapping);
```

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

- Data Conflicts





- Data Conflicts
- Limited Transactional Resources

**RESOURCE
EXHAUSTION**



- Data Conflicts
- Limited Transactional Resources

RESOURCE EXHAUSTION

- Certain Instructions
 - IO instructions, syscall, ...



- Data Conflicts
- Limited Transactional Resources

RESOURCE EXHAUSTION

- Certain Instructions
 - IO instructions, syscall, ...
- Synchronous Exception Events
 - #BR, #PF, #DB, #BP/INT3, ...



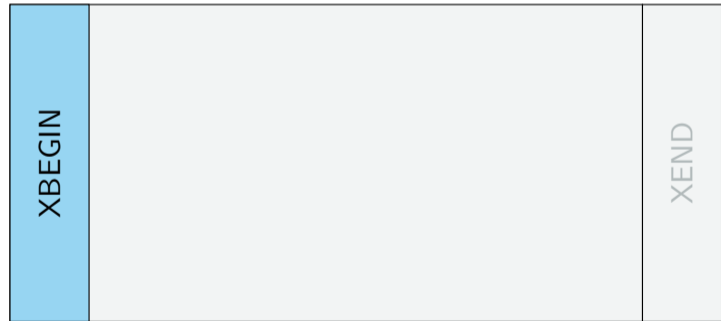
12.2.4.5 Miscellaneous Transactional Aborts

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.

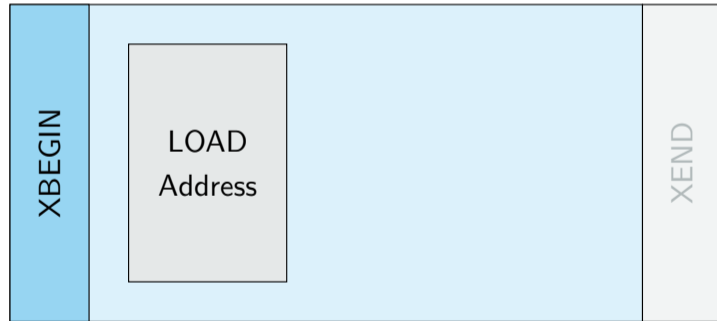


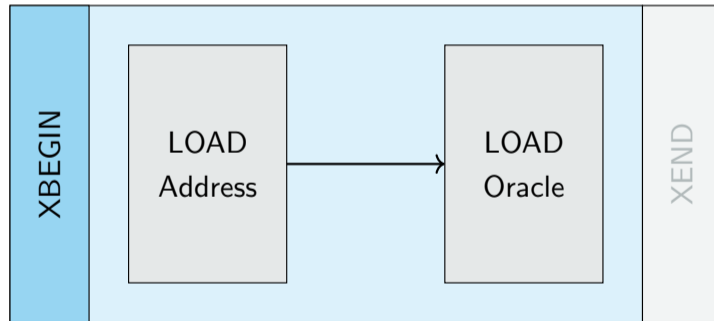
12.2.4.5 Miscellaneous Transactional Aborts

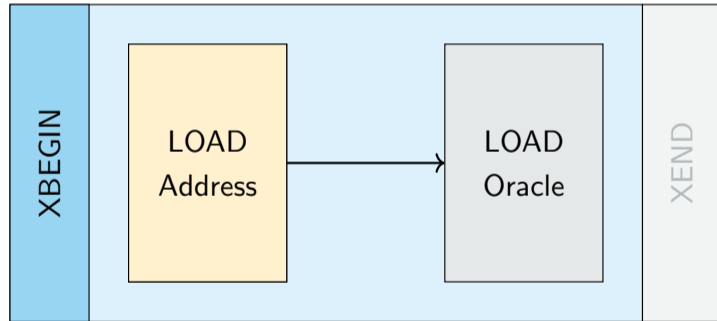
Asynchronous events (NMI, SMI, INTR, IPI, PMI, etc.) occurring during transactional execution may cause the transactional execution to abort and transition to a non-transactional execution. [...] For example, operating systems with timer ticks generate interrupts that can **cause transactional aborts**.

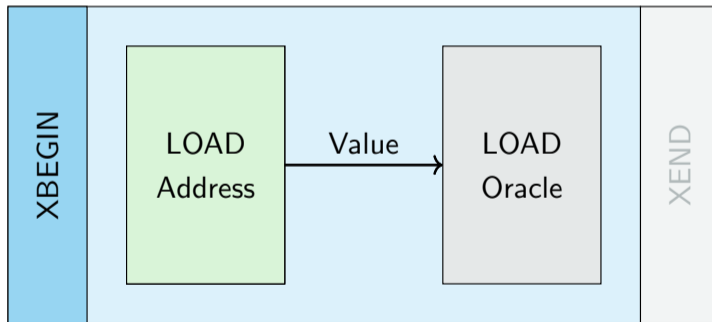


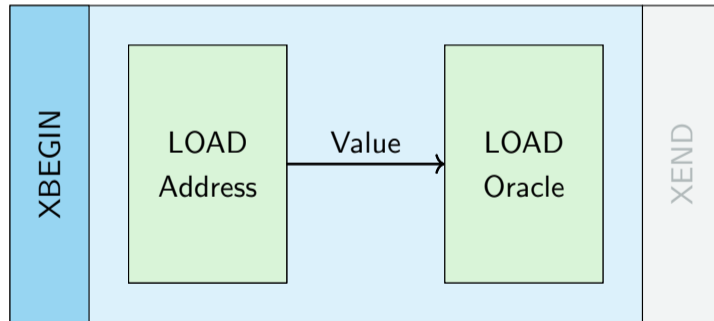


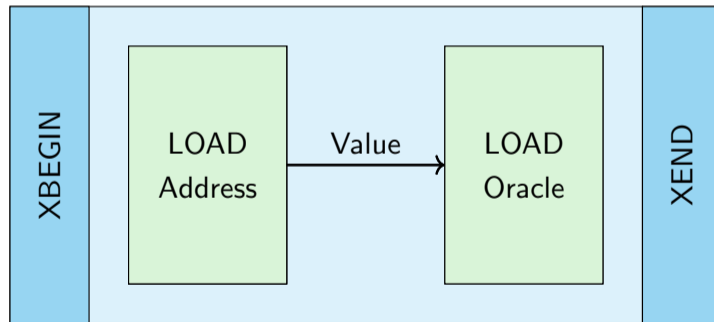




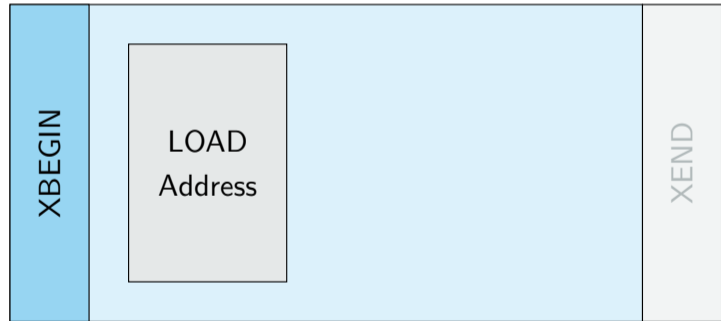


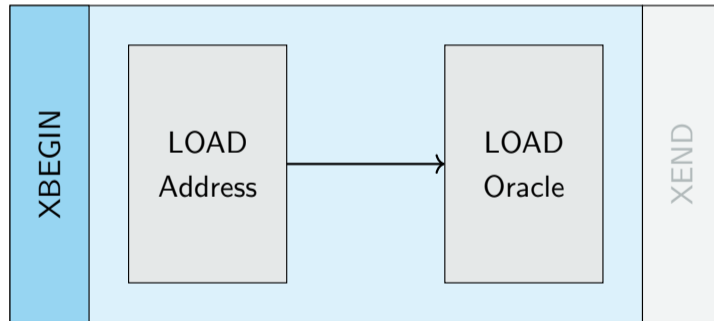


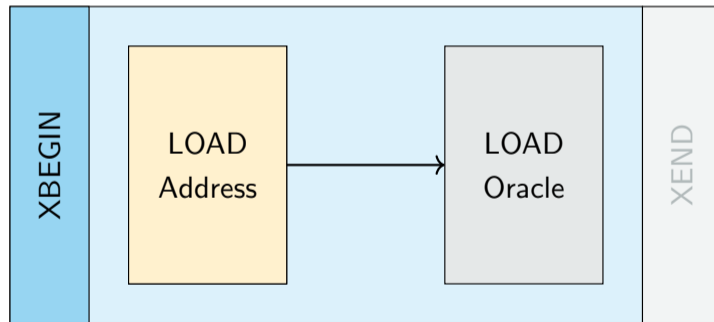


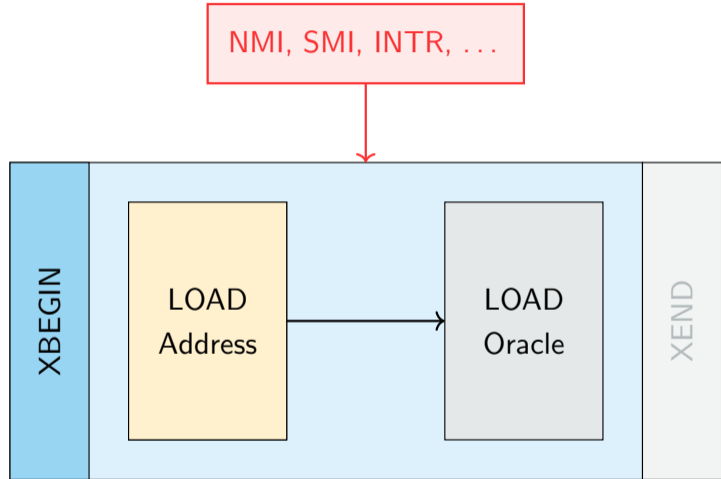


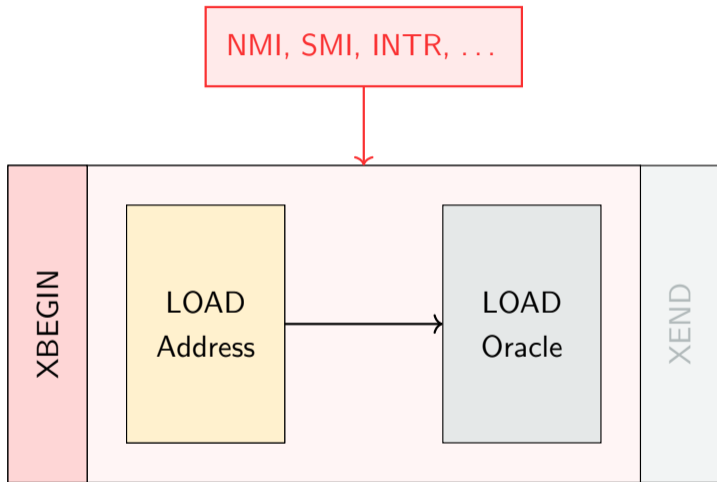


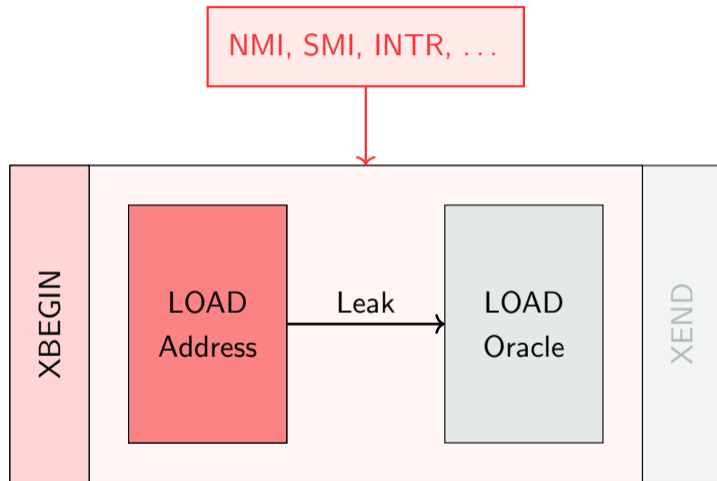


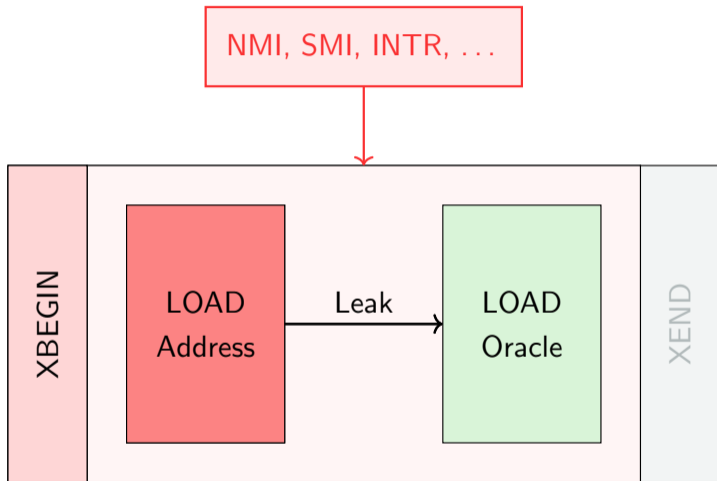


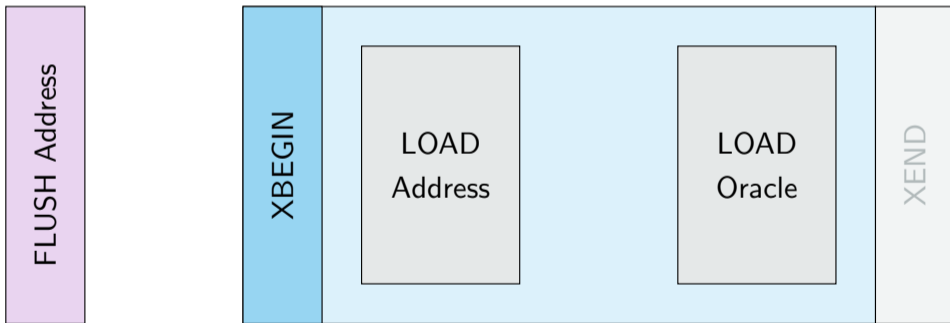


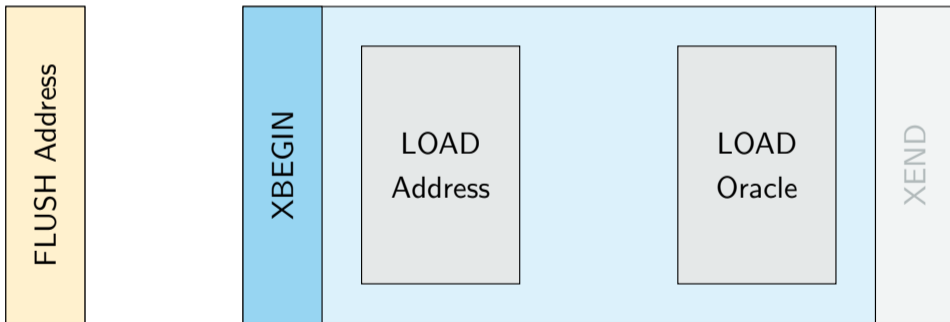


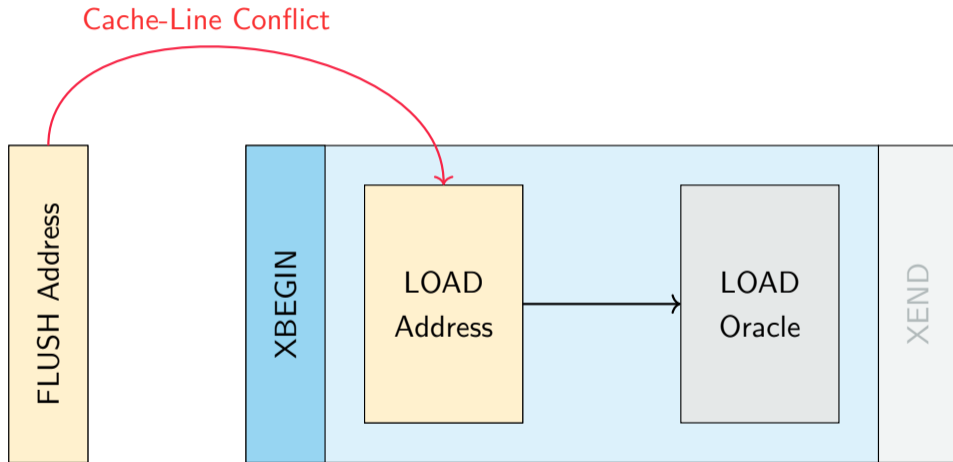


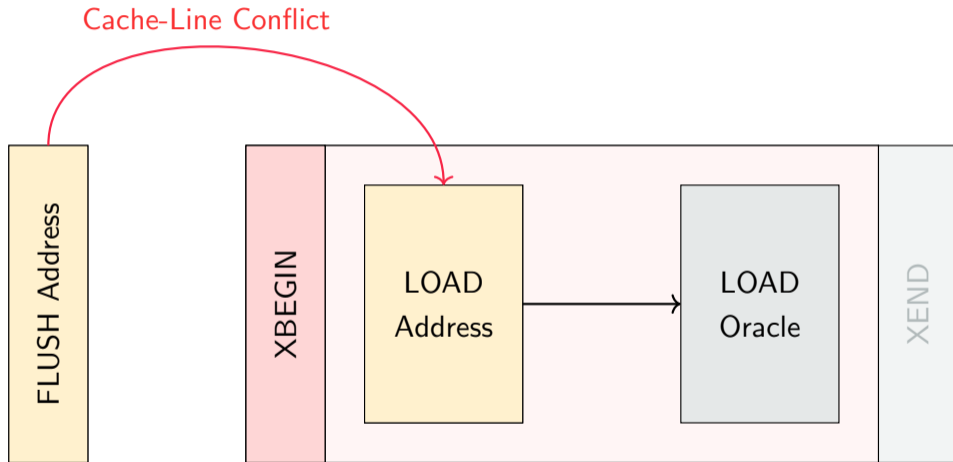


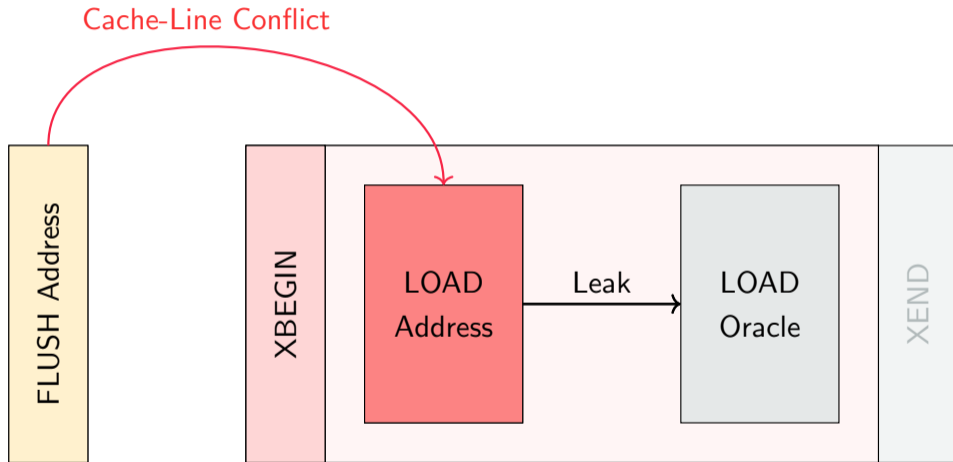


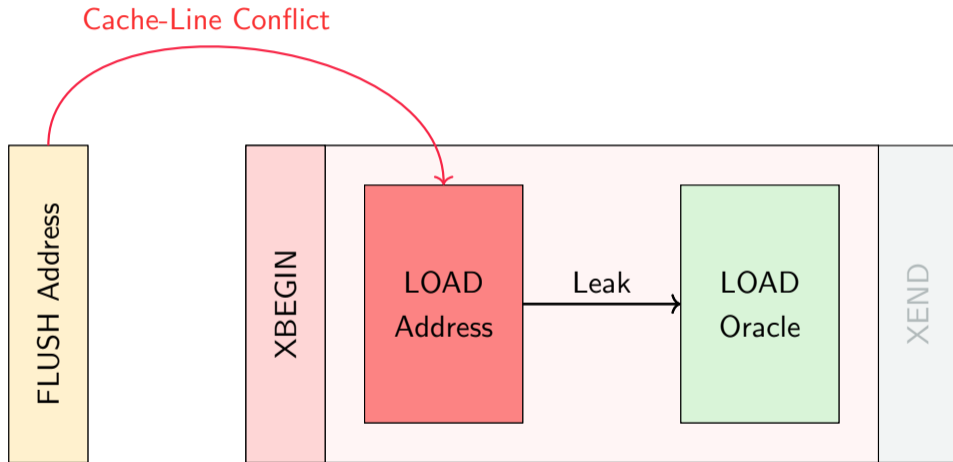














Variant 1

Kernel Mapping



Variant 3

Microcode-Assisted Page-Table Walk



works does not work can be prevented



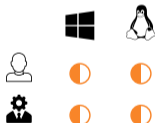
Variant 1

Kernel Mapping



Variant 2

Transactional Asynchronous Abort



Variant 3

Microcode-Assisted Page-Table Walk



works does not work can be prevented



Variant 1

Kernel Mapping

5.30 kB/s



Variant 3

Microcode-Assisted
Page-Table Walk

7.73 kB/s



Variant 1

Kernel Mapping

5.30 kB/s



Variant 2

Transactional
Asynchronous Abort

39.66 kB/s

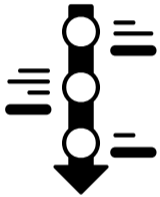


Variant 3

Microcode-Assisted
Page-Table Walk

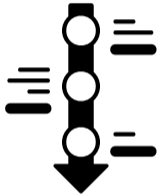
7.73 kB/s





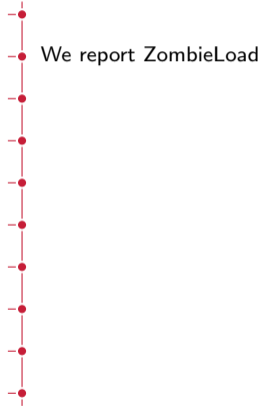
2019

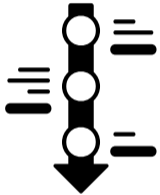




2019

April 12

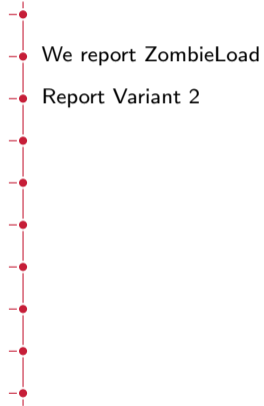


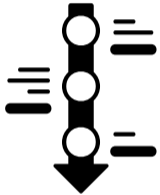


2019

April 12

April 24



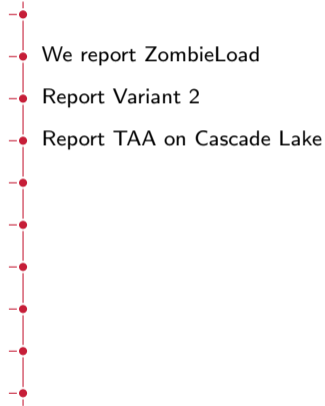


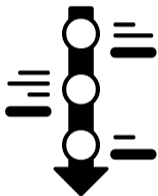
2019

April 12

April 24

May 10





2019

April 12

April 24

May 10

May 11

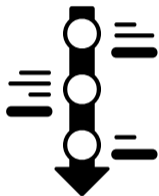


We report ZombieLoad

Report Variant 2

Report TAA on Cascade Lake

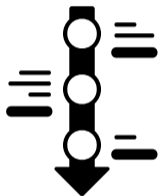
Call with Intel + Embargo



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)

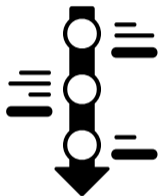




2019

- 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

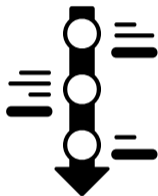




2019

- 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





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

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



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



WAIT A MINUTE...



TAA IS TSX OVER A LEAK GADGET?

- BlackHat USA: “Meltdown: Basics, Details, Consequences” (📅 August 9, 2018)

- BlackHat USA: “Meltdown: Basics, Details, Consequences” (📅 August 9, 2018)

Meltdown with Fault Suppression

- Intel TSX to suppress exceptions instead of signal handler

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

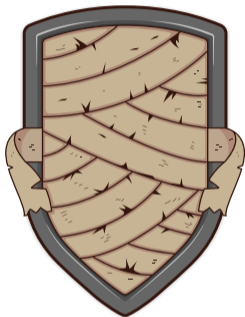
for (size_t i = 0; i < 256; i++) {
    if (flush_and_reload(array + i * 4096) == CACHE_HIT) {
        printf("%c\n", i);
    }
}
```

35 Moritz Lipp, Michael Schwarz, Daniel Gruss | Graz University of Technology

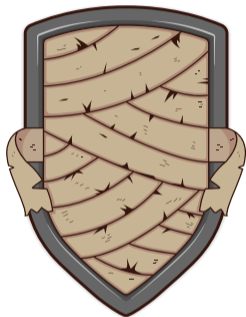


IF YOU DISABLE TSX

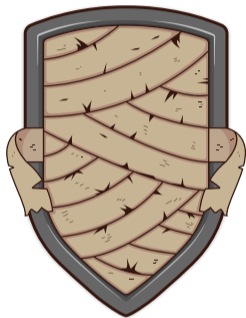
**YOU CAN'T HAVE
TRANSACTIONAL ABORTS**



- **Disable** Intel TSX



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



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



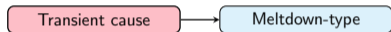
- Software-sequences and VERW do **not work reliably**

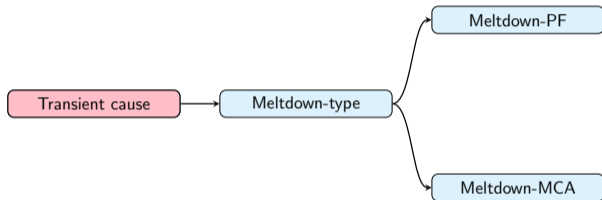


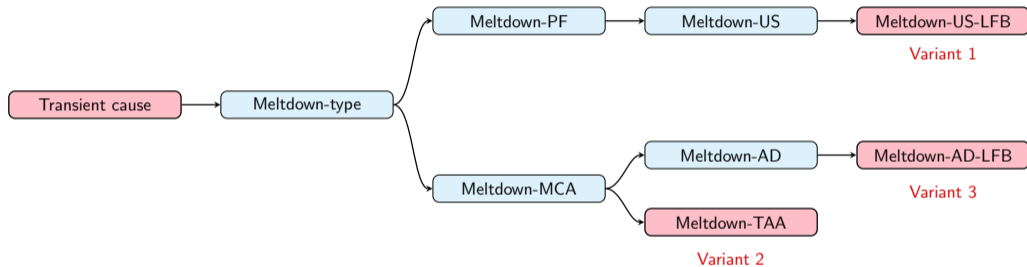
- Software-sequences and VERW do **not work reliably**
 - Cases where leakage is still visible

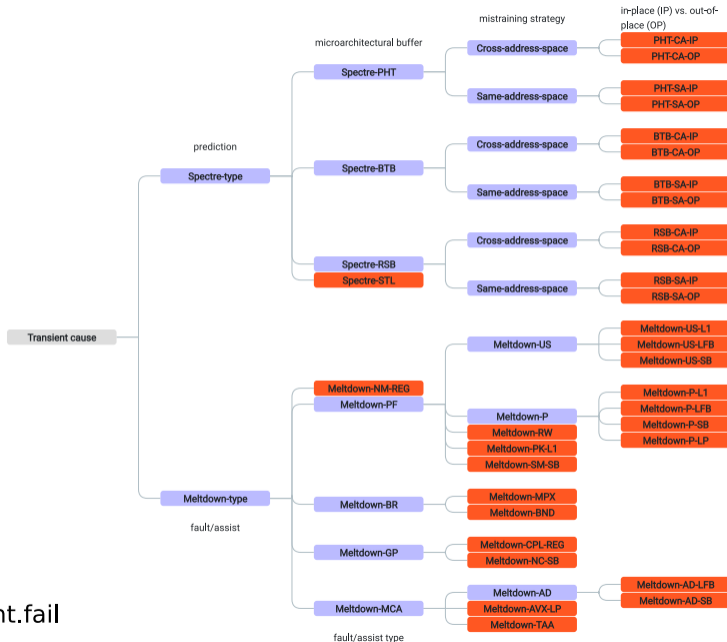


Transient cause



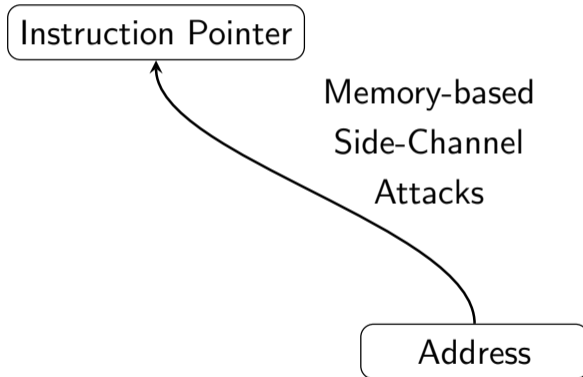


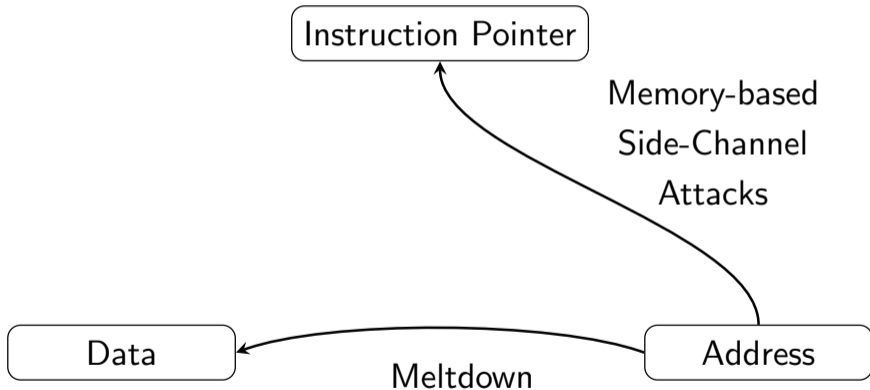


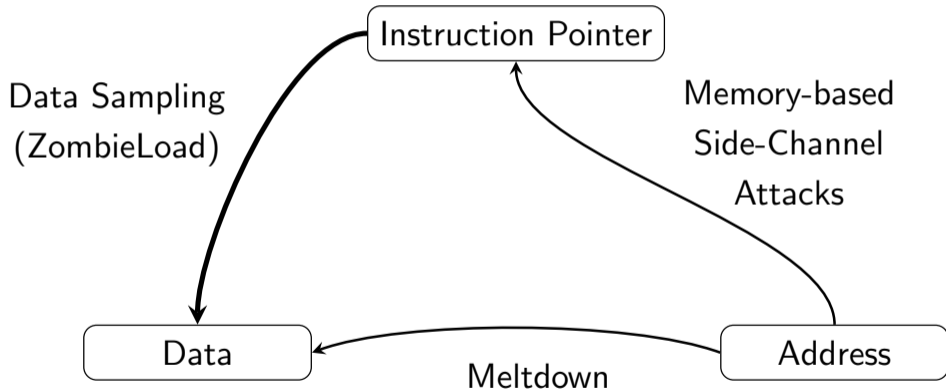


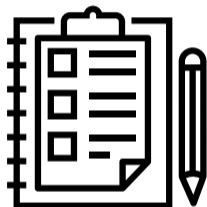
<https://transient.fail>

Address









- Spectre is here to stay
- **More Meltdown**-type than Spectre-type attacks



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

- <https://github.com/IAIK/ZombieLoad>



Other 2019 papers in the same space:

- Fallout: Leaking Data on Meltdown-resistant CPUs
- RIDL: Rogue In-Flight Data Load



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




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



- **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

 <https://github.com/IAIK/ZombieLoad>

Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz), Daniel Gruss (@lavados)

December 28, 2019

IAIK, Graz University of Technology

- 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