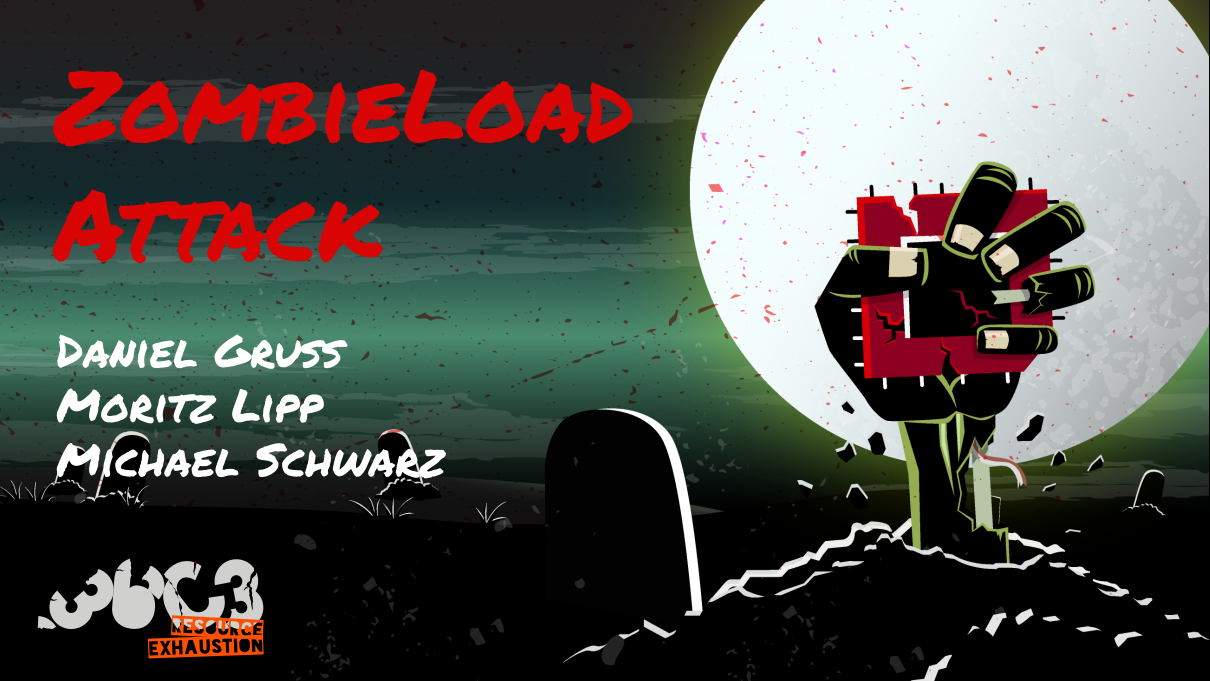
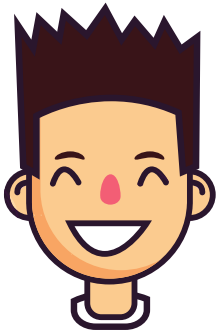
ZombieLoad Attack

Daniel Gruss
Moritz Lipp
Michael Schwarz



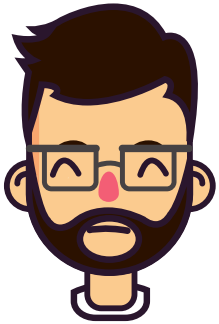


Michael Schwarz

Post Doc @ Graz University of Technology

@misc0110

michael.schwarz@iaik.tugraz.at

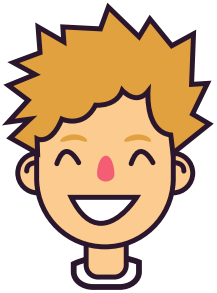


Moritz Lipp

PhD Candidate @ Graz University of Technology

@mlqxyz

moritz.lipp@iaik.tugraz.at



Daniel Gruss

Assistant Professor @ Graz University of Technology

@lavados

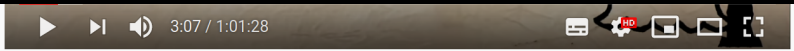
daniel.gruss@iaik.tugraz.at


LAST
CHRISTMAS
WHAM!

A Christmas Carol

The Spectres of the Past, Present, and Future

Claudio Canella
Daniel Gruss
Moritz Lipp
Michael Schwarz



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

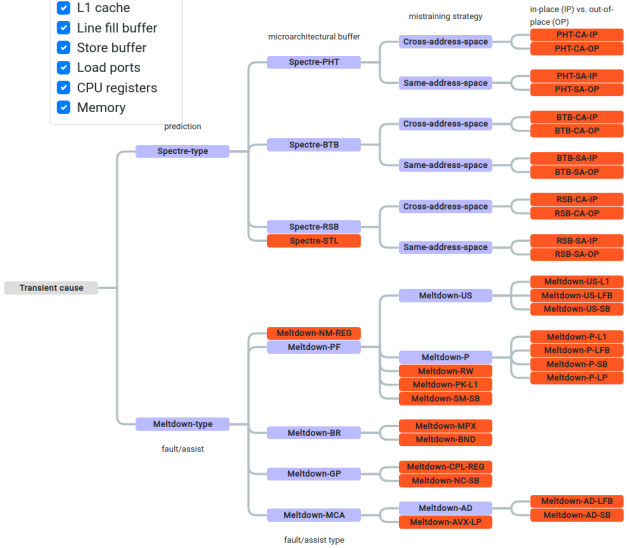
Antworten ausblenden

DerUnbekannte vor 11 Monaten
pretty sure that was him playing Ebenezer Scrooge

ANTWORTEN

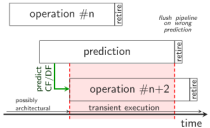
Spectre | Meltdown | Buffer | Vendor | More filters | Search | Search | Export

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



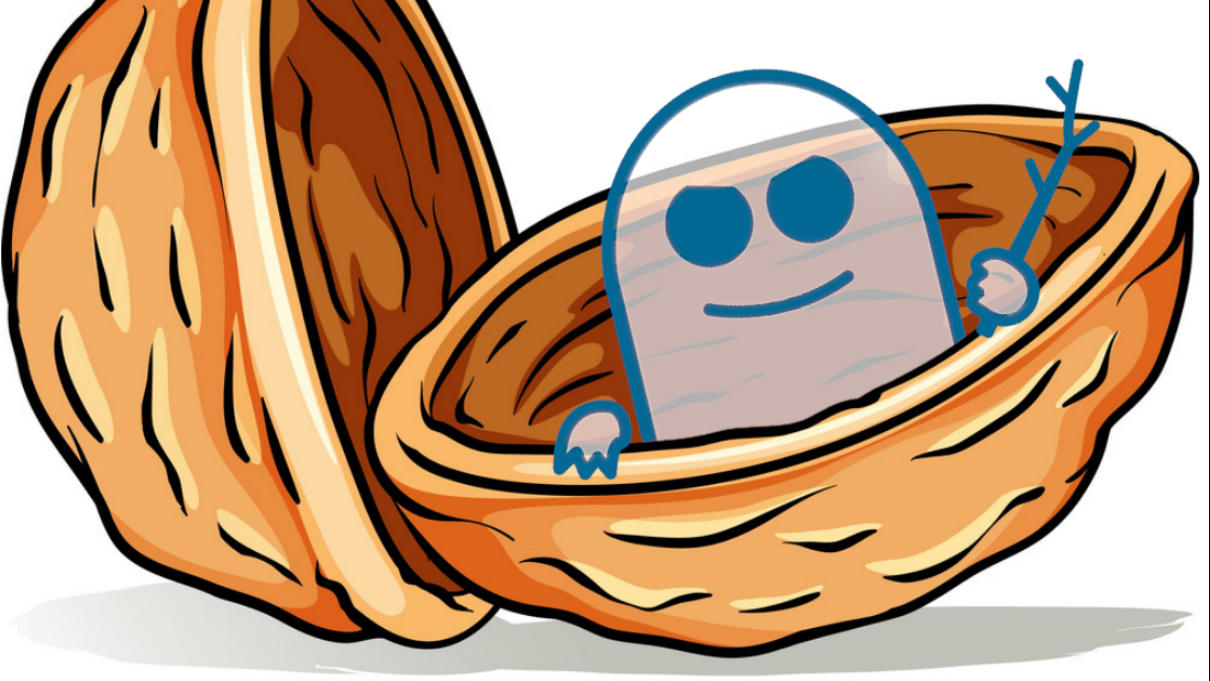
Details

Spectre-type

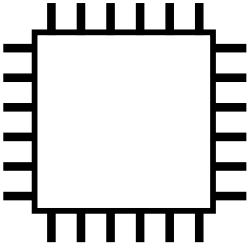


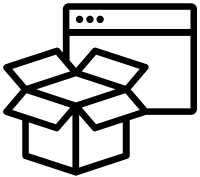
Spectre exploits a performance optimization in modern CPUs. Instead of waiting for the correct resolution of a branch, the CPU tries to predict the most likely outcome of the branch and starts transiently executing along the predicted path. Upon resolving the branch, the CPU discards the results of the 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.

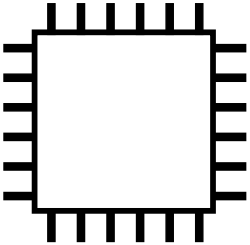
References

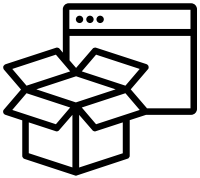



Need
Background

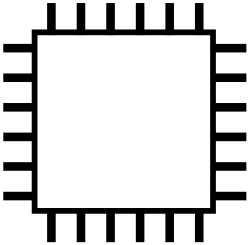


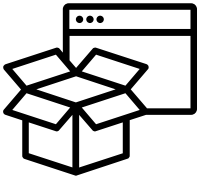




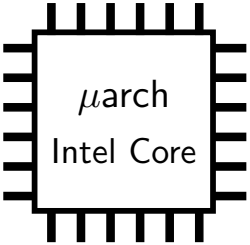


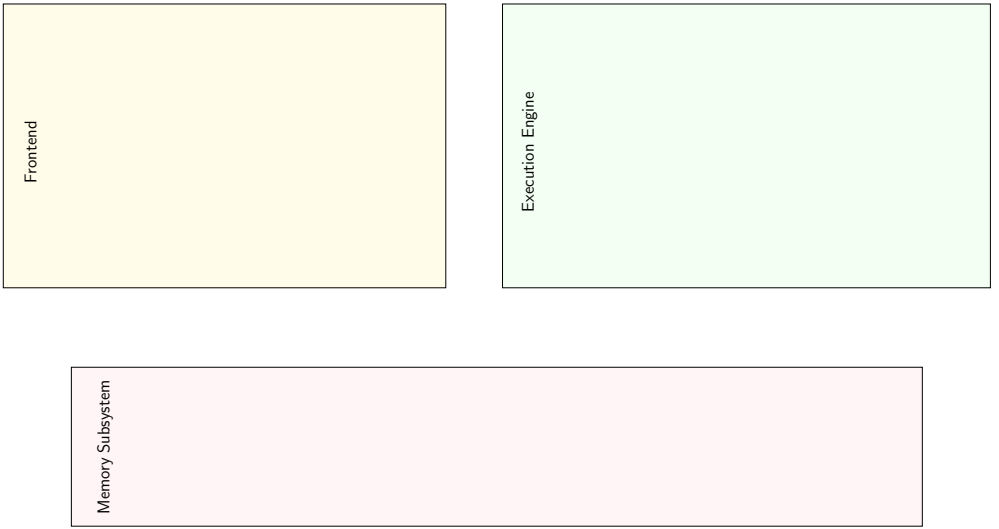

ISA
x86

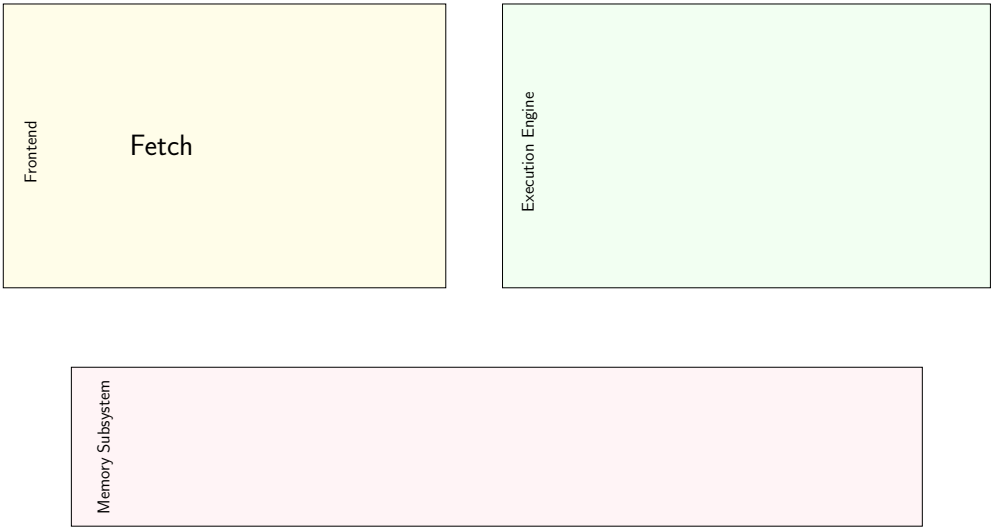


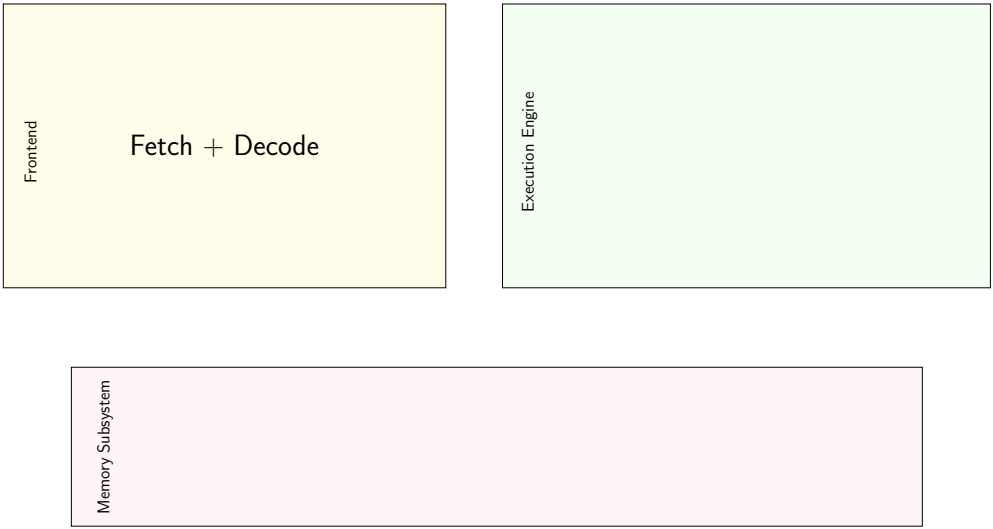


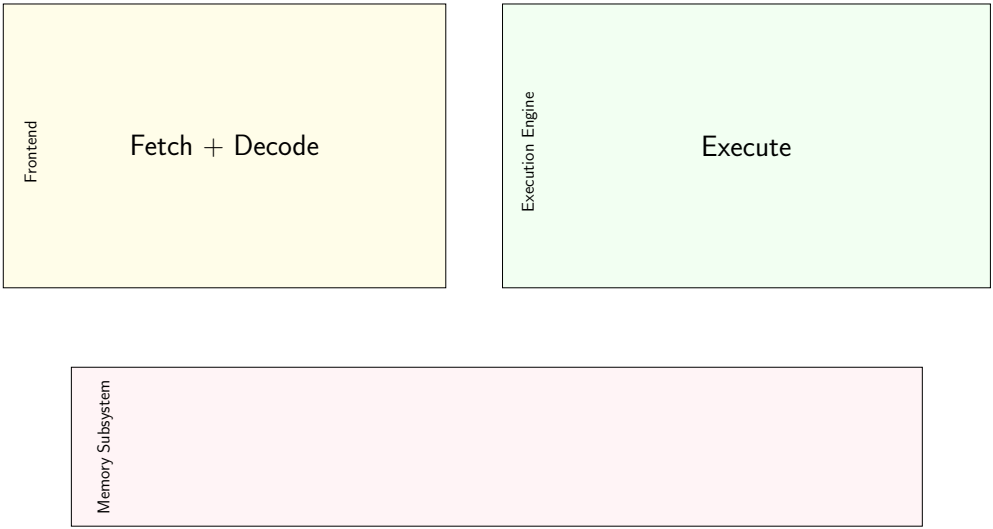

ISA
x86

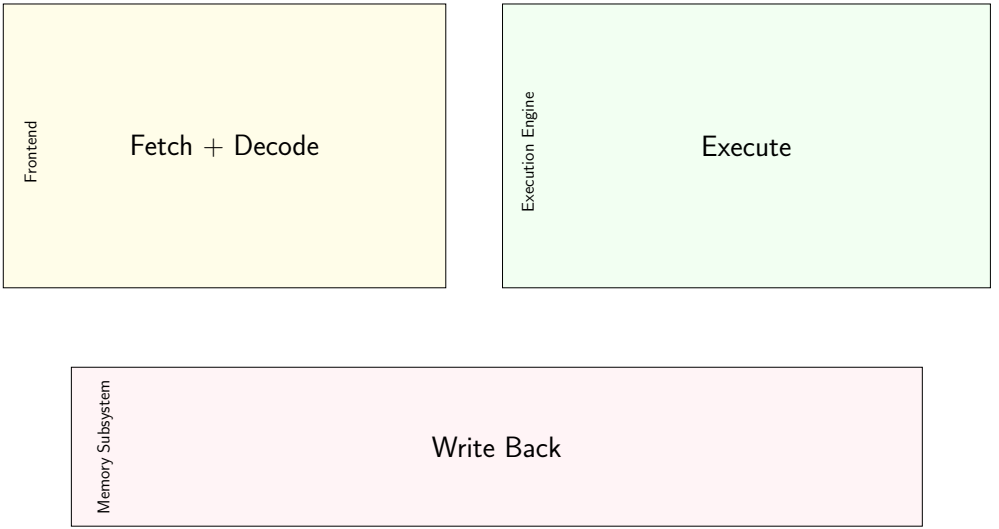

μarch
Intel Core

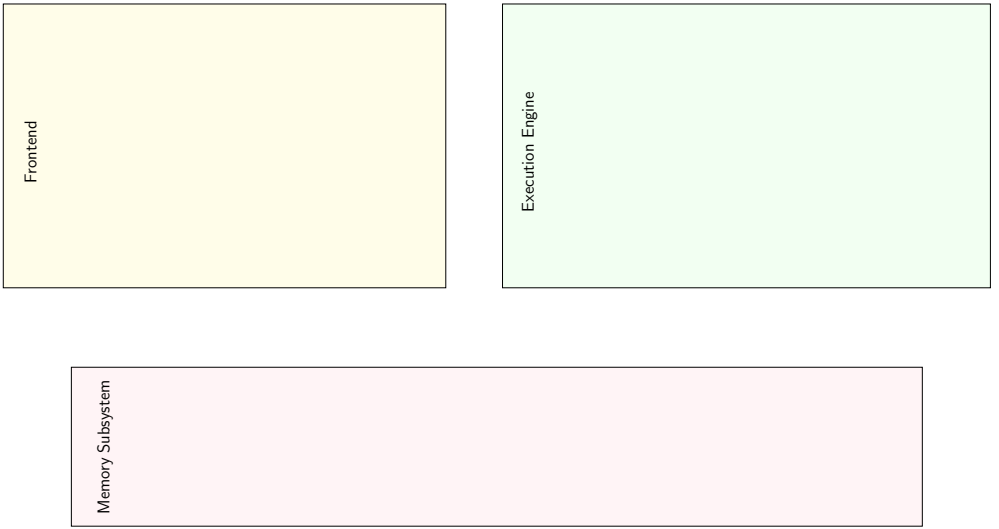

Frontend
Execution Engine
Memory Subsystem

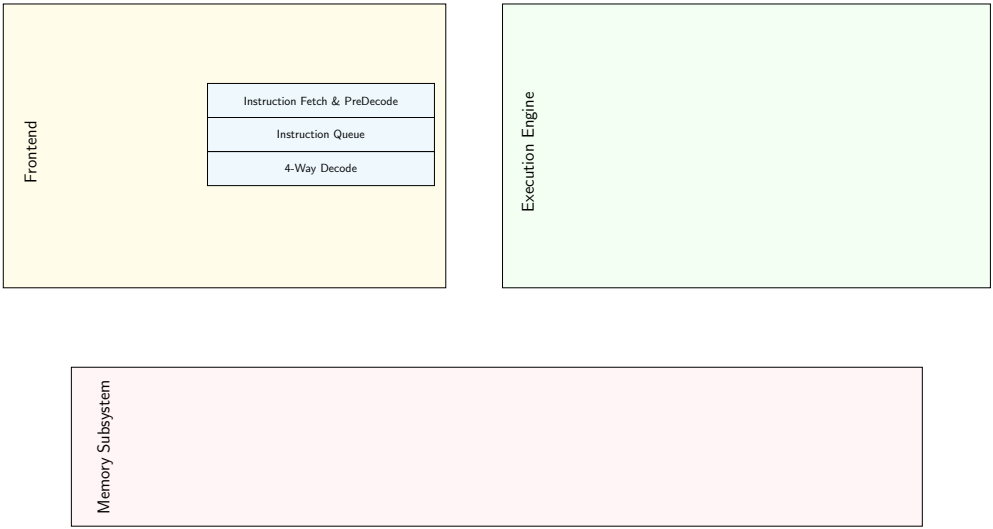

Frontend
Fetch
Execution Engine
Memory Subsystem

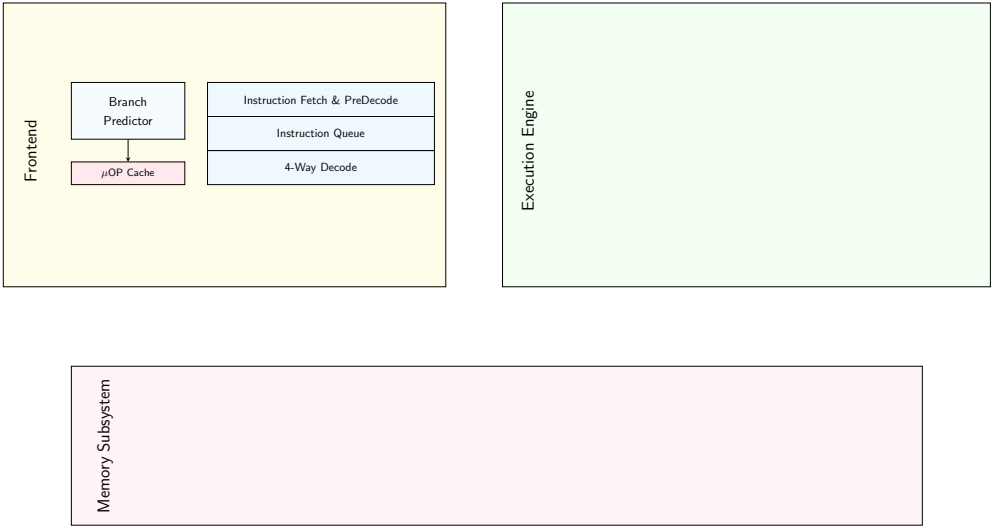

Frontend
Fetch + Decode
Execution Engine
Memory Subsystem

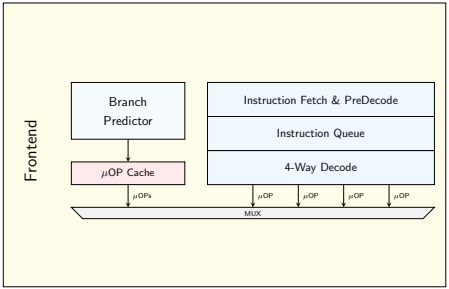

Frontend
Fetch + Decode
Execution Engine
Execute
Memory Subsystem


Frontend
Fetch + Decode
Execution Engine
Execute
Memory Subsystem
Write Back


Frontend
Execution Engine
Memory Subsystem

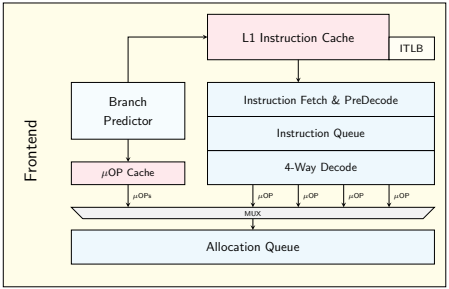

Frontend
Instruction Fetch & PreDecode
Instruction Queue
4-Way Decode
Execution Engine
Memory Subsystem


Frontend
Branch Predictor
μOP Cache
Instruction Fetch & PreDecode
Instruction Queue
4-Way Decode
Execution Engine
Memory Subsystem


Frontend
Branch Predictor
μOP Cache
Instruction Fetch & PreDecode
Instruction Queue
4-Way Decode
μOPs
μOP
μOP
μOP
μOP
MUX

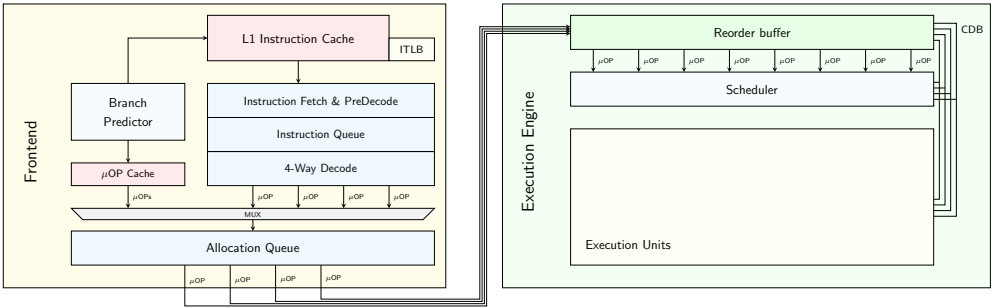

Execution Engine


Memory Subsystem

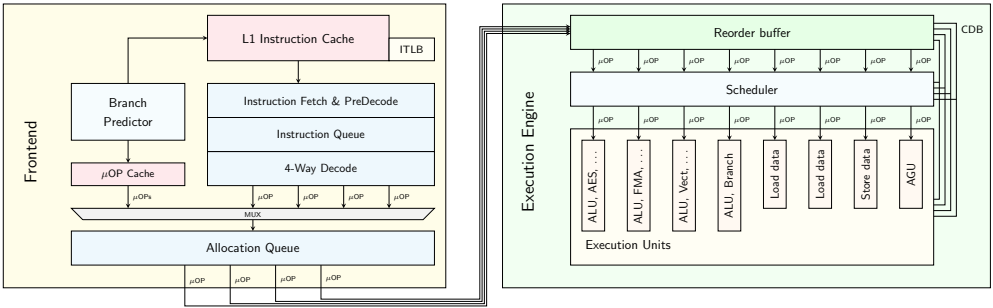

Frontend
L1 Instruction Cache
ITLB
Branch Predictor
Instruction Fetch & PreDecode
Instruction Queue
4-Way Decode
μOP Cache
μOPs
μOP
μOP
μOP
μOP
MUX
Allocation Queue


Execution Engine

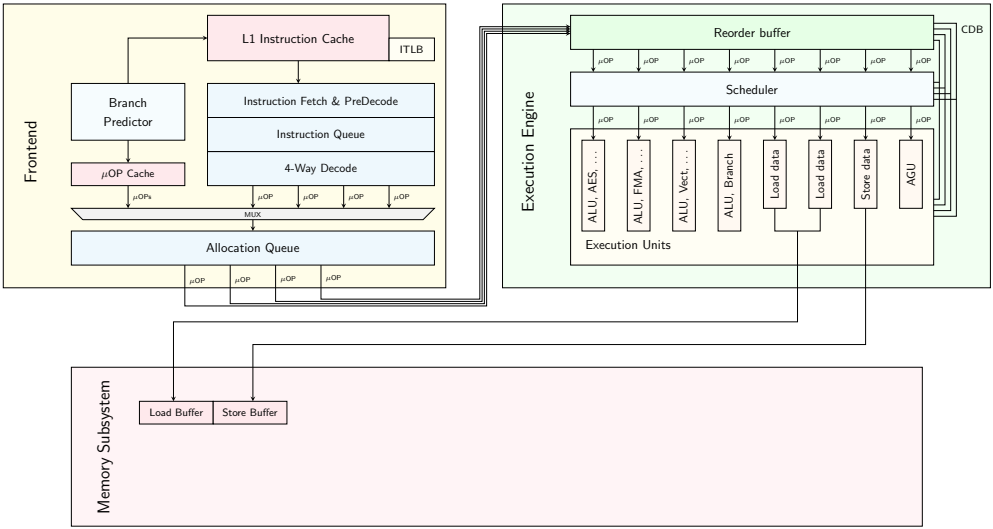

Memory Subsystem



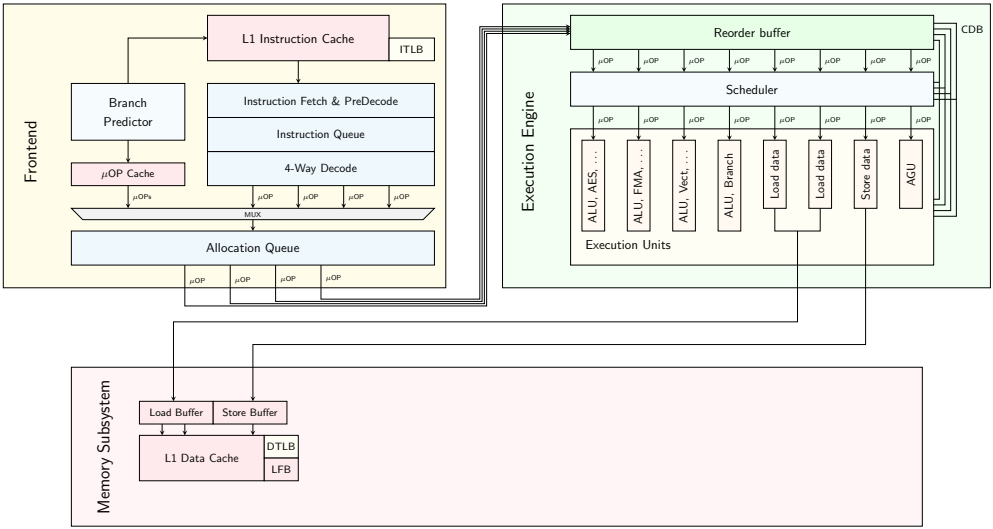
Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz), Daniel Gruss (@lavados)

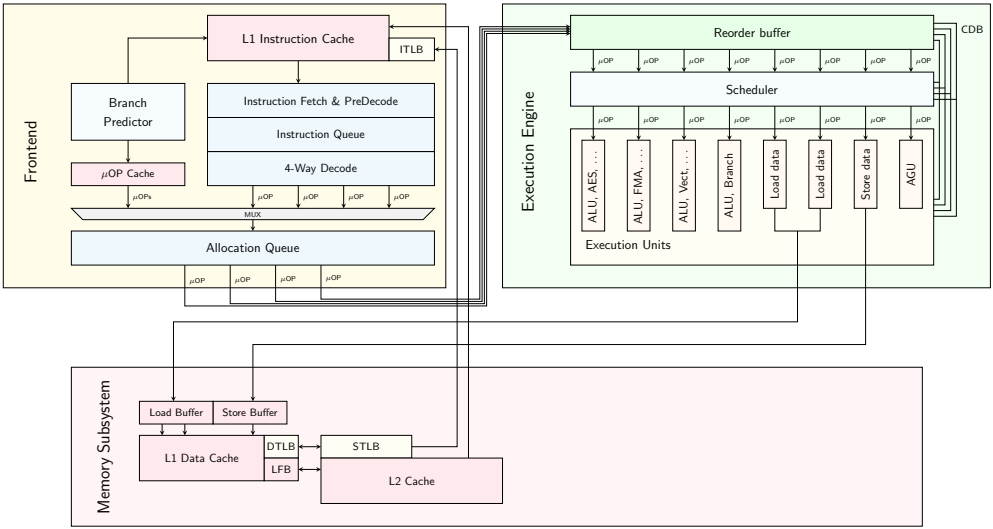

Frontend
L1 Instruction Cache
ITLB
Branch Predictor
Instruction Fetch & PreDecode
Instruction Queue
4-Way Decode
μOP Cache
μOPs
μOP
MUX
Allocation Queue
Execution Engine
Reorder buffer
CDB
Scheduler
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units

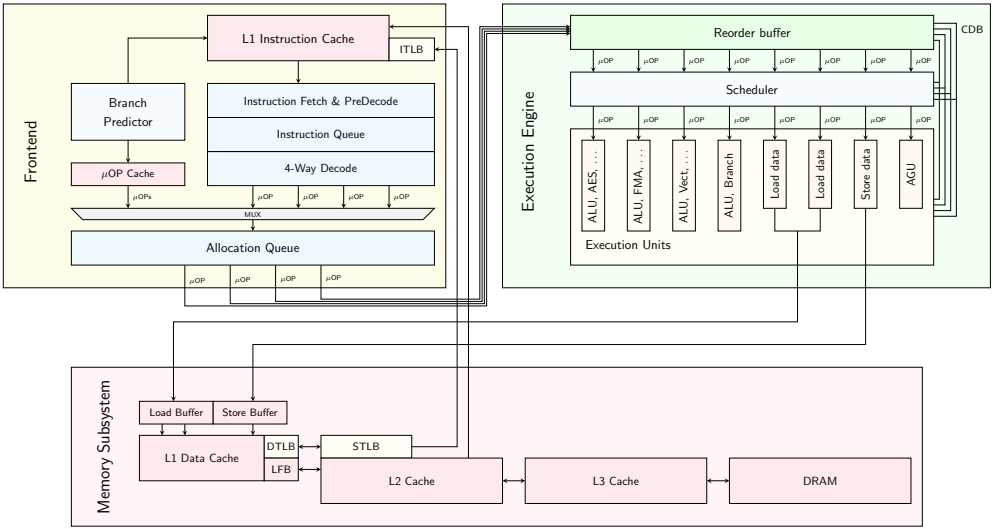

Memory Subsystem



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

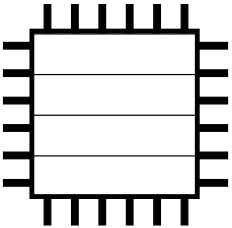

Frontend
L1 Instruction Cache
ITLB
Branch Predictor
Instruction Fetch & PreDecode
Instruction Queue
4-Way Decode
μOP Cache
μOPs
μOP
MUX
Allocation Queue
Execution Engine
Reorder buffer
CDB
Scheduler
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Memory Subsystem
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB

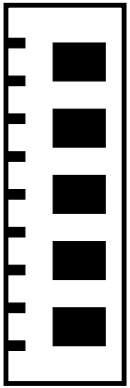

Frontend
L1 Instruction Cache
ITLB
Branch Predictor
Instruction Fetch & PreDecode
Instruction Queue
4-Way Decode
μOP Cache
μOPs
μOP
MUX
Allocation Queue
Execution Engine
Reorder buffer
CDB
Scheduler
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Memory Subsystem
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB
STLB
L2 Cache



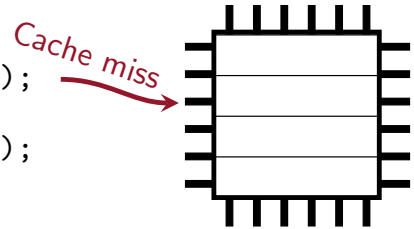
Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz), Daniel Gruss (@lavados)

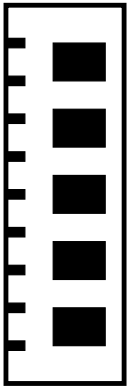
```
printf("%d", i);

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



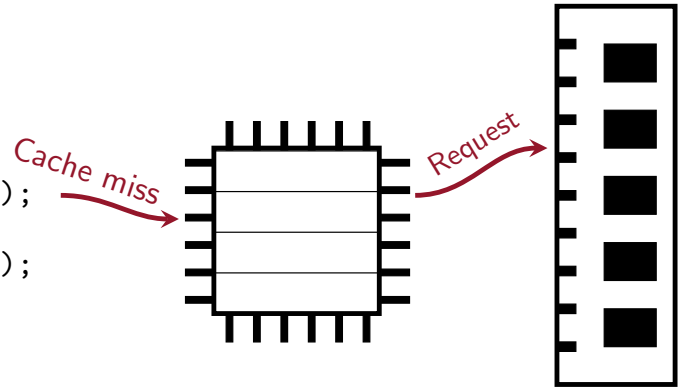


```
printf("%d", i);

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

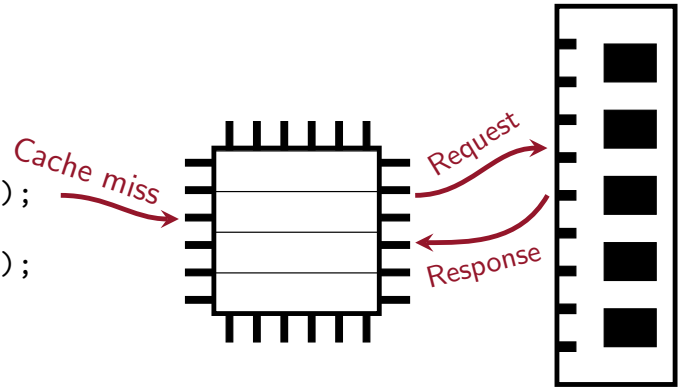


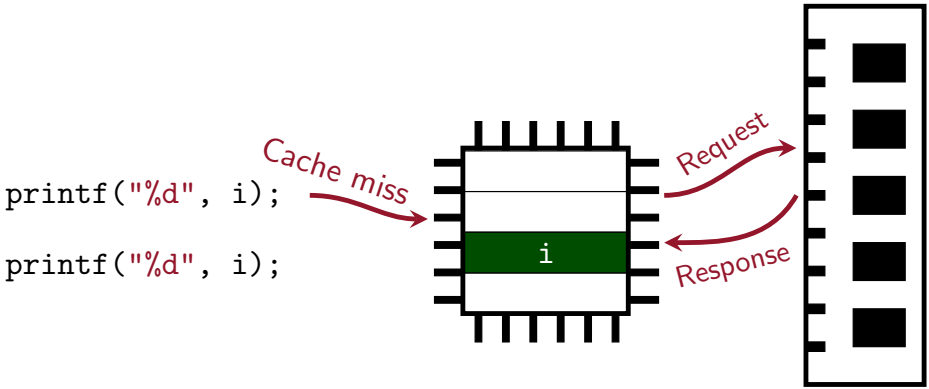


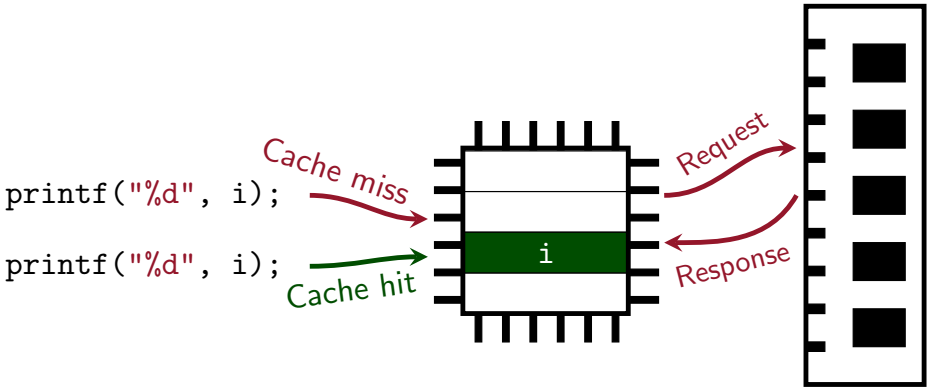
```
printf("%d", i);

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

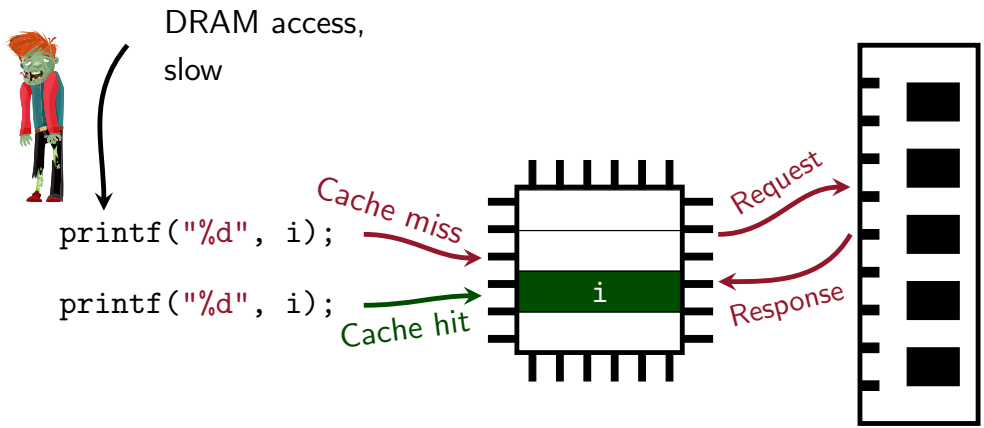


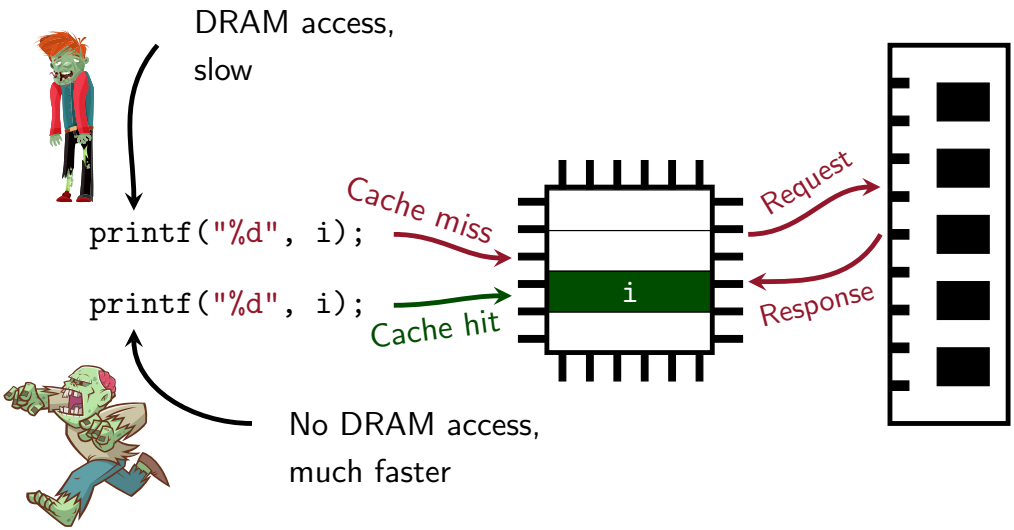
```
printf("%d", i);

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

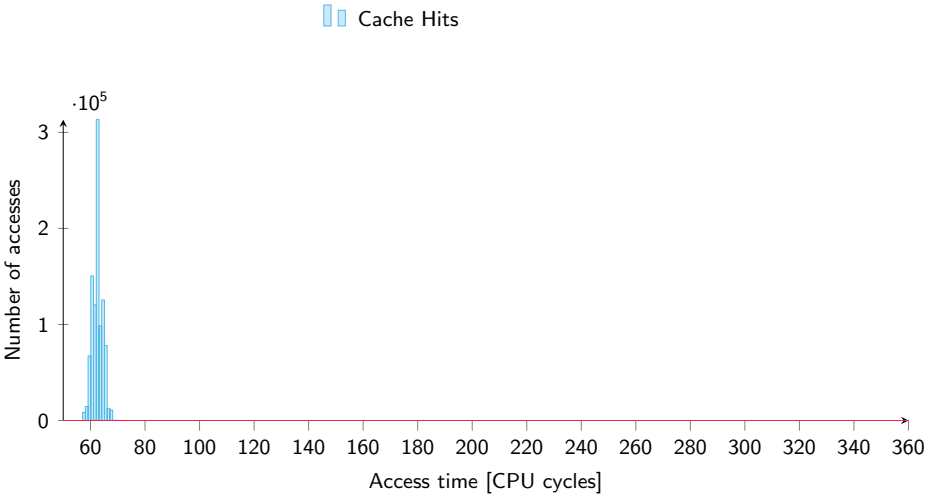


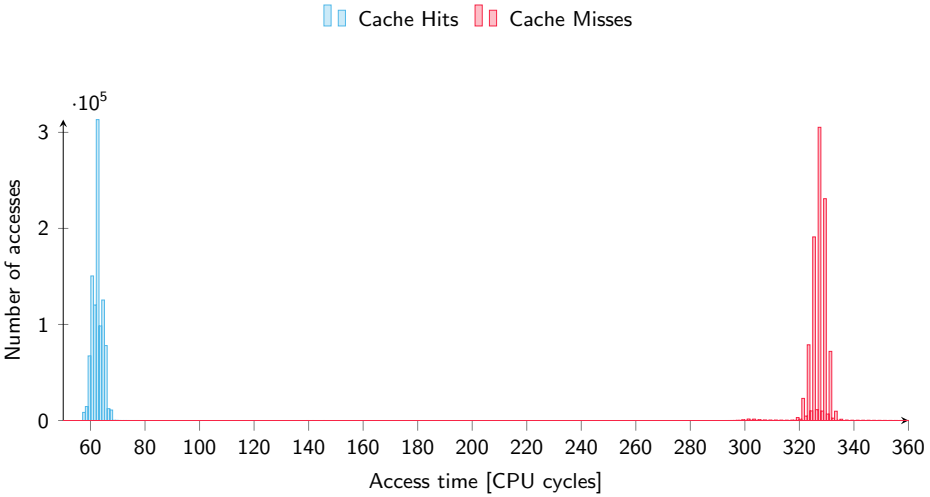

printf("%d", i);
printf("%d", i);
Cache miss
i
Request
Response


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


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


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


Cache Hits
$\cdot 10^5$
3
2
1
0
Number of accesses
60
80
100
120
140
160
180
200
220
240
260
280
300
320
340
360
Access time [CPU cycles]

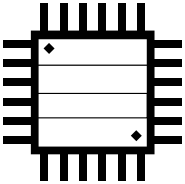

Cache Hits
Cache Misses
$\cdot 10^5$
3
2
1
0
Number of accesses
60
80
100
120
140
160
180
200
220
240
260
280
300
320
340
360
Access time [CPU cycles]

ATTACKER

flush

access

Shared Memory



VICTIM

access

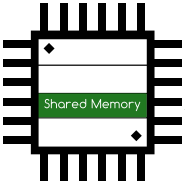
ATTACKER

flush

access

Shared Memory

cached



cached

VICTIM

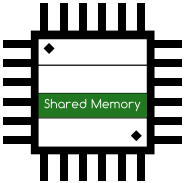
access

ATTACKER

flush

access

Shared Memory



VICTIM

access

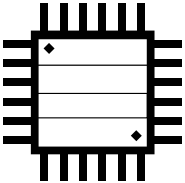
Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz), Daniel Gruss (@lavados)

ATTACKER

flush

access

Shared Memory



VICTIM

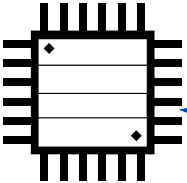
access

ATTACKER

flush

access

Shared Memory



VICTIM

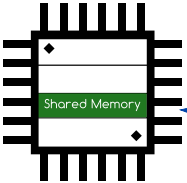
access

ATTACKER

flush

access

Shared Memory



VICTIM

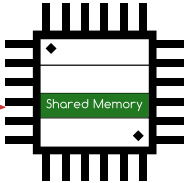
access

ATTACKER

flush

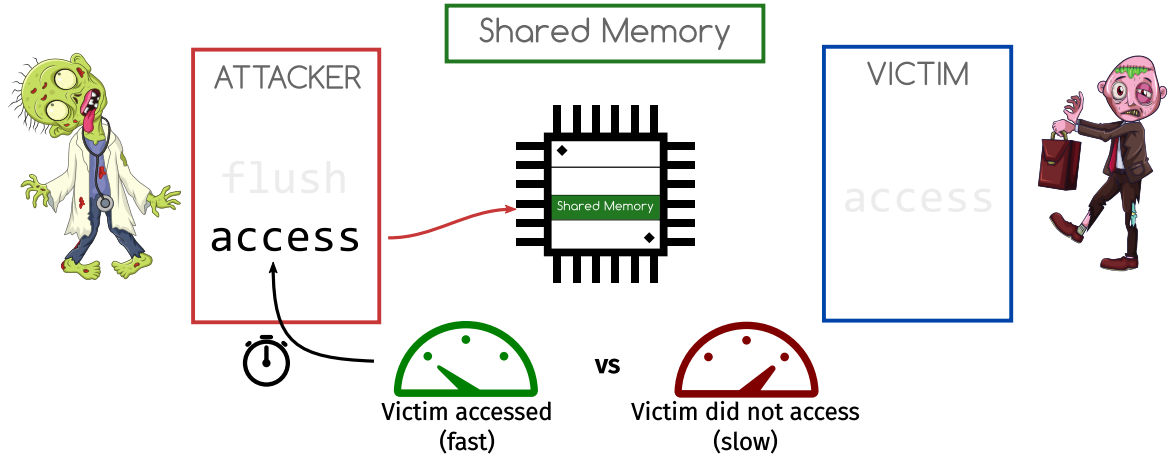
access

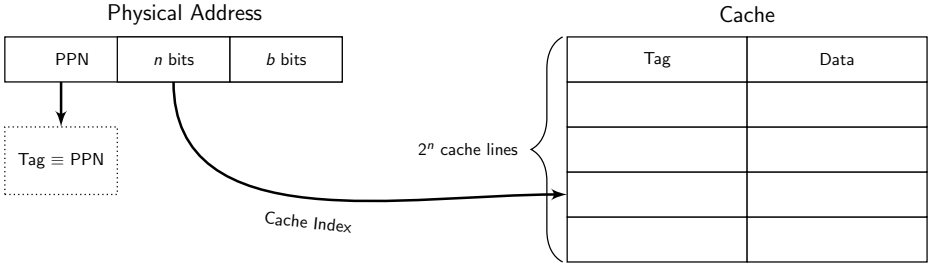
Shared Memory

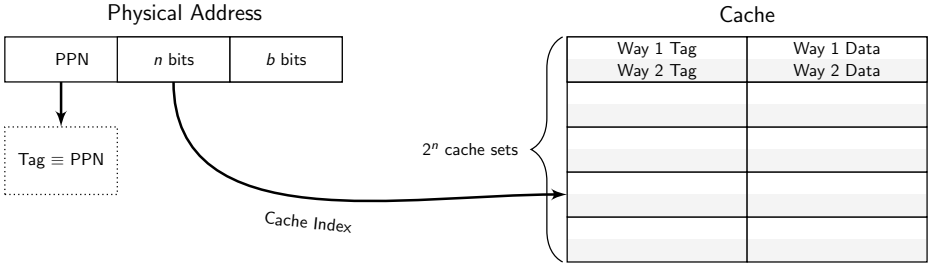


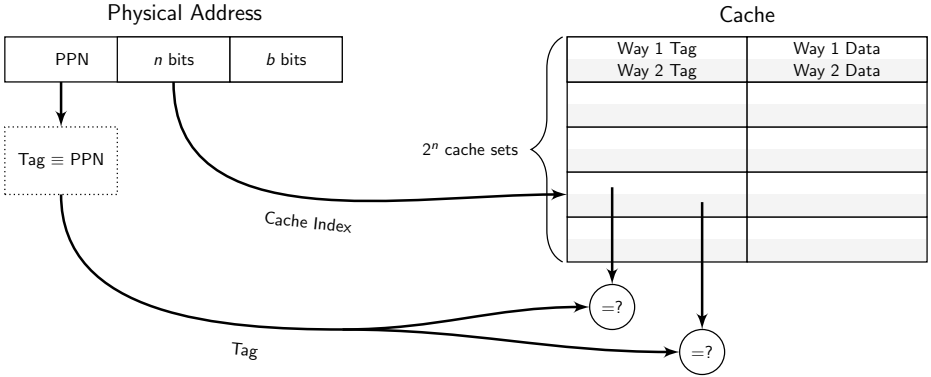
VICTIM

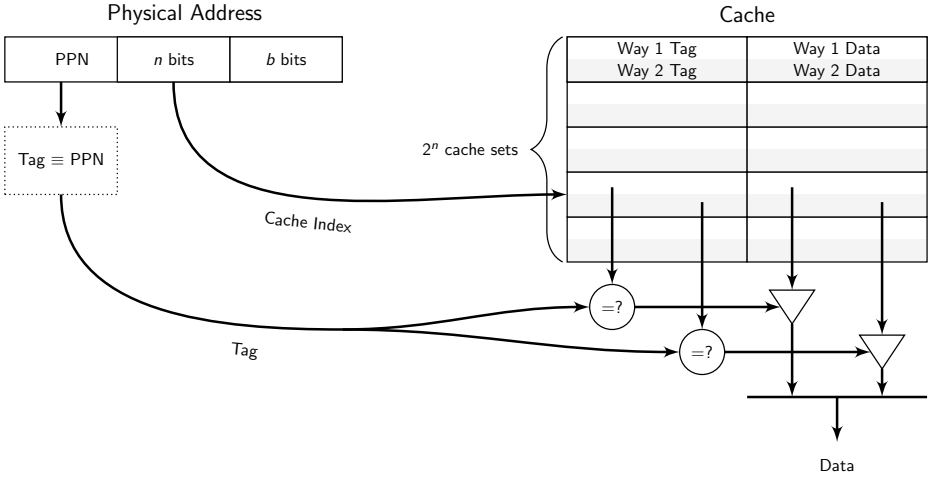
access


Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access
vs
Victim accessed
(fast)
Victim did not access
(slow)


Physical Address
PPN
n bits
b bits
Tag ≡ PPN
Cache Index
Cache
Tag
Data
2^n cache lines


Physical Address
PPN
n bits
b bits
Tag ≡ PPN
Cache Index
Cache
Way 1 Tag
Way 2 Tag
Way 1 Data
Way 2 Data
2^n cache sets


Physical Address
PPN
n bits
b bits
Tag ≡ PPN
Cache Index
Tag
Cache
Way 1 Tag
Way 2 Tag
Way 1 Data
Way 2 Data
2^n cache sets
=?
=?

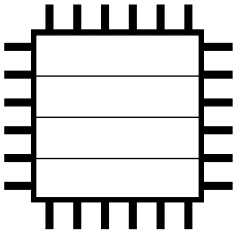

Physical Address
PPN
n bits
b bits
Tag ≡ PPN
Cache Index
Tag
Cache
Way 1 Tag
Way 2 Tag
Way 1 Data
Way 2 Data
2^n cache sets
=?
=?
Data


Attack

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]
```

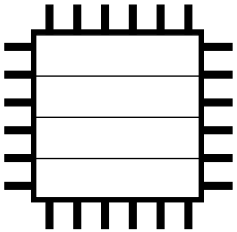


User Memory

	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

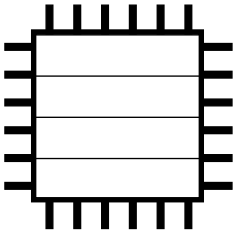
```
char value = kernel[0]
```

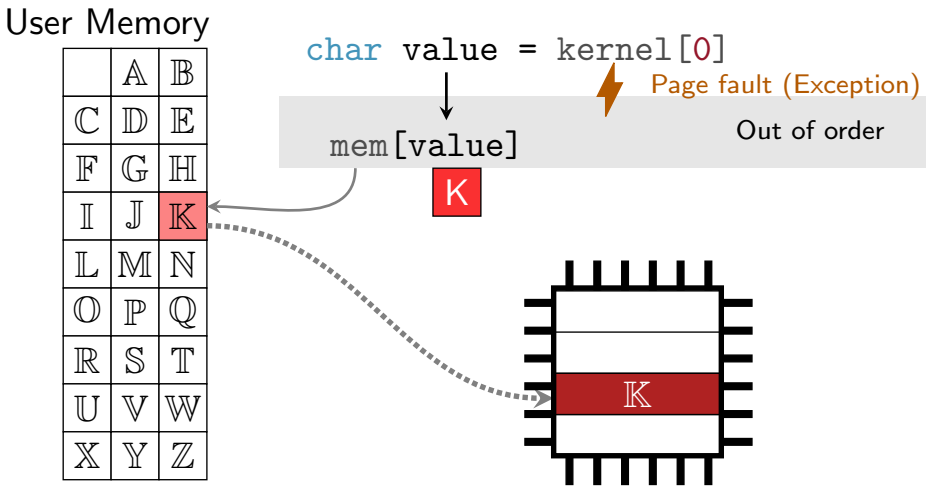
Page fault (Exception)

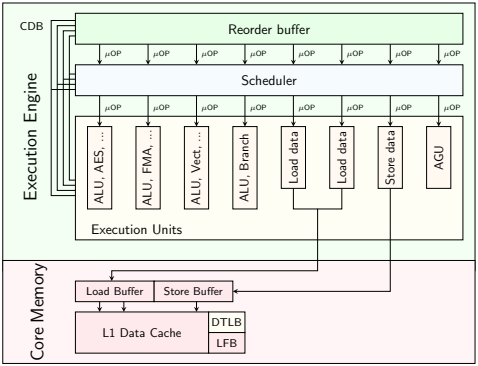
```
mem[value]
```

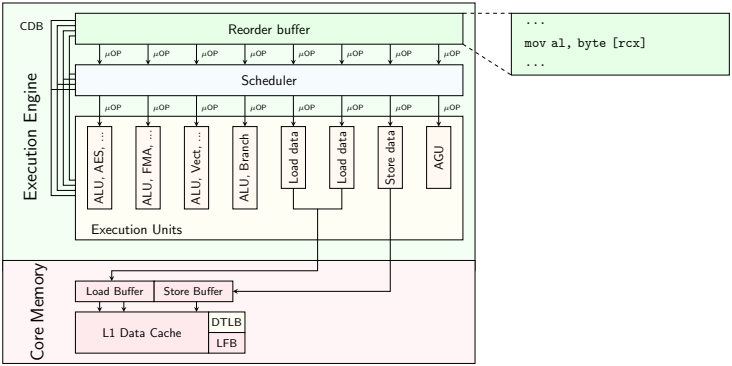
Out of order

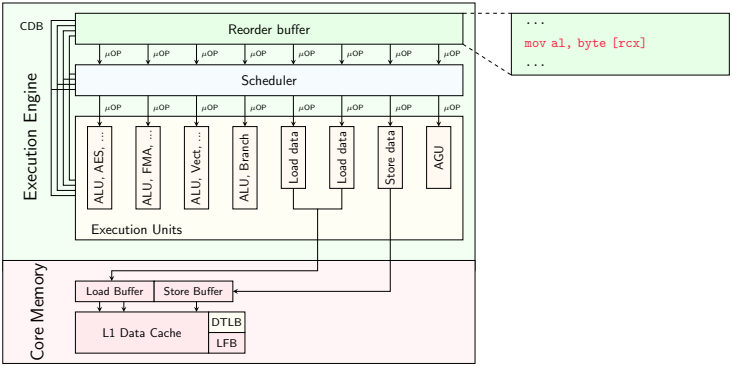
K

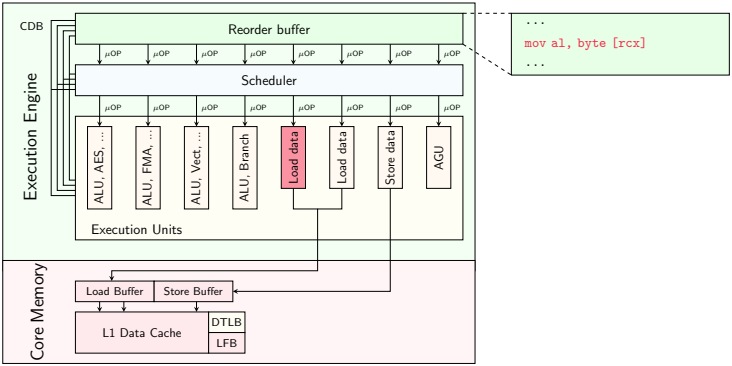


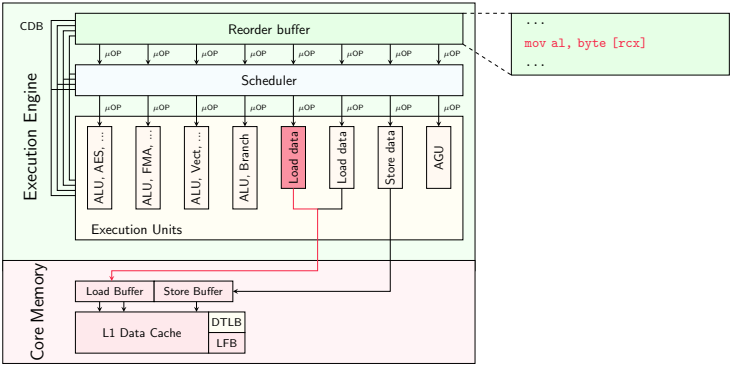

User Memory
A B
C D E
F G H
I J K
L M N
O P Q
R S T
U V W
X Y Z
char value = kernel[0]
Page fault (Exception)
Out of order
mem[value]
K
K

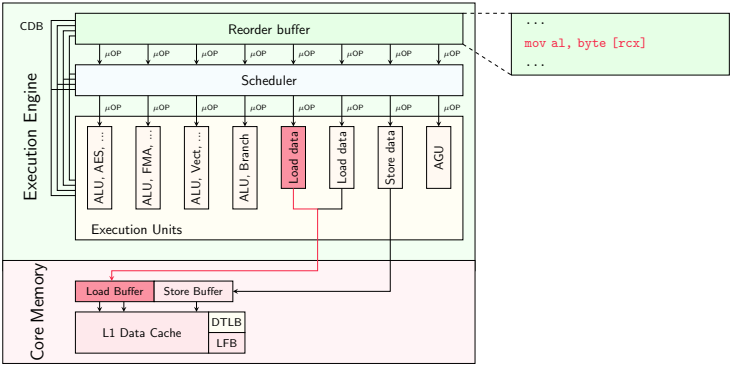

CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect., ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB

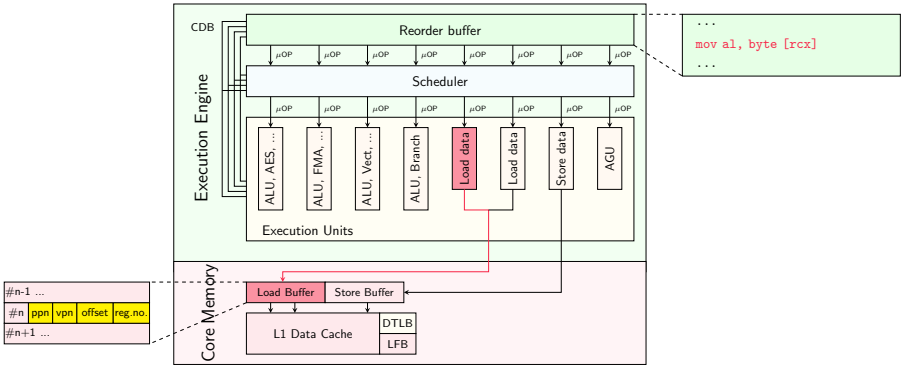

CDB
Reorder buffer
...
mov al, byte [rcx]
...
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect., ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB

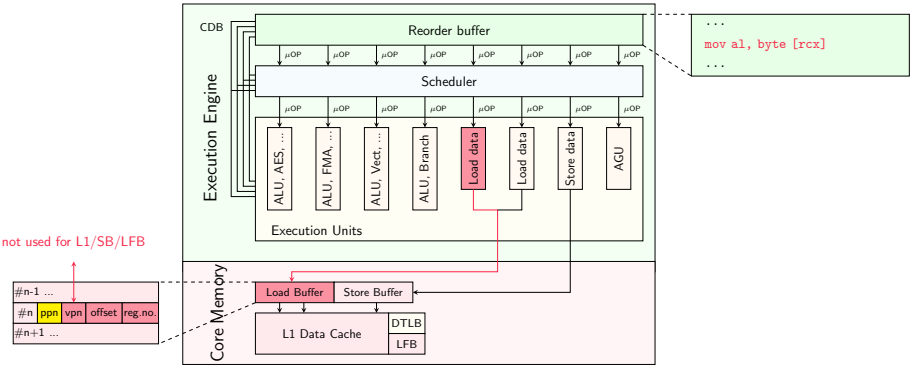

CDB
Reorder buffer
...
mov al, byte [rcx]
...
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect., ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB

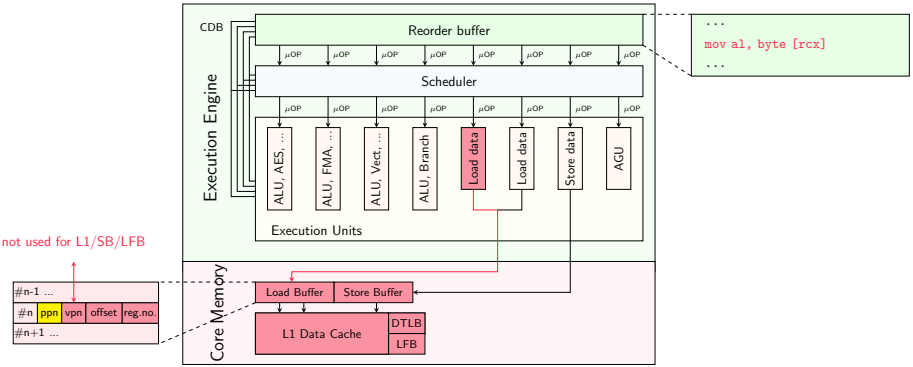

CDB
Reorder buffer
...
mov al, byte [rcx]
...
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect., ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB

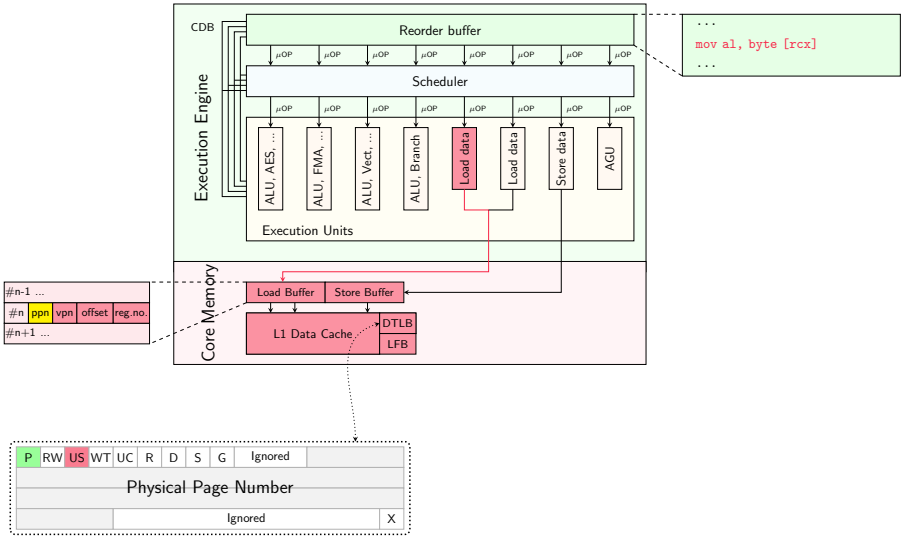

CDB
Reorder buffer
...
mov al, byte [rcx]
...
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect., ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB


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

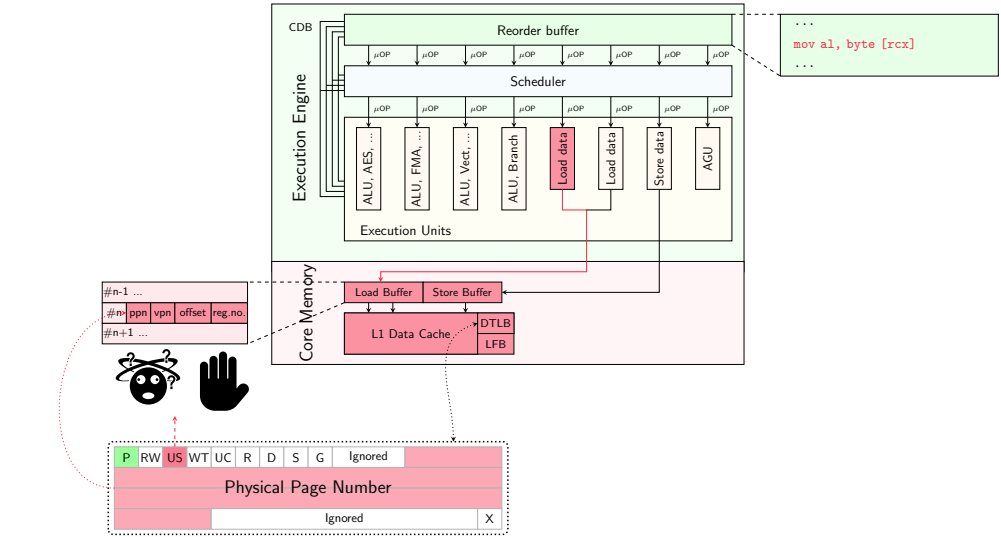

CDB
Reorder buffer
...
mov al, byte [rcx]
...
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect., ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB


CDB
Reorder buffer
...
mov al, byte [rcx]
...
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect., ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
not used for L1/SB/LFB
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB

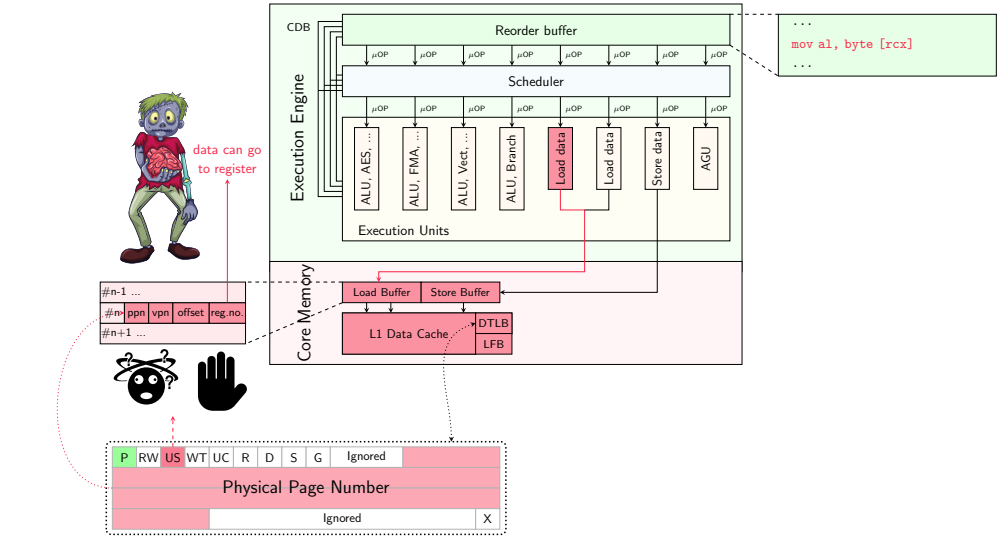

CDB
Reorder buffer
...
mov al, byte [rcx]
...
Scheduler
Execution Engine
μOP
ALU, AES, ...
ALU, FMA, ...
ALU, Vect., ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
not used for L1/SB/LFB
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB



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



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



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

Google

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

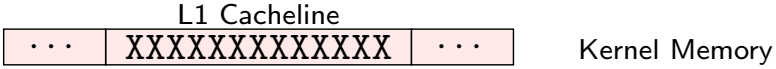
552.4 kB/s

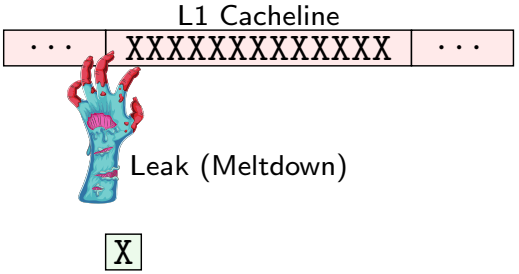
Meltdown
Leakage Rate

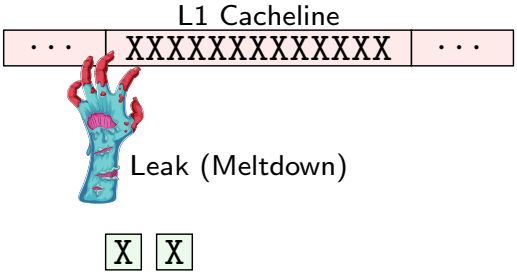
552.4 kB/s

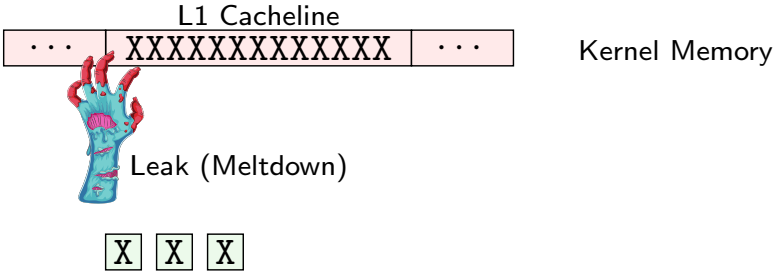
Meltdown
Error Rate

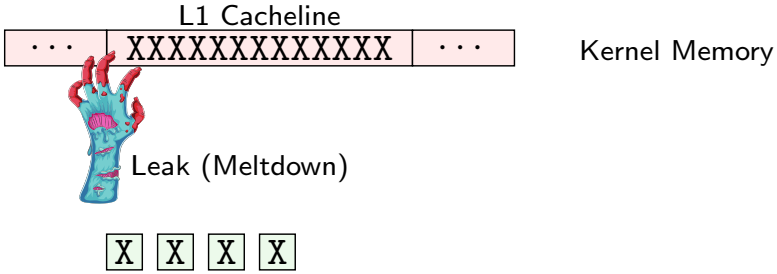
0.003 %

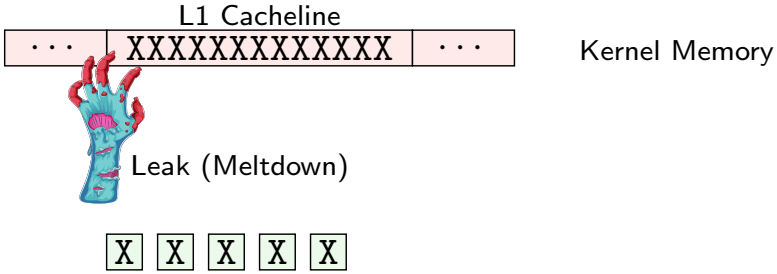

L1 Cacheline
· · ·
XXXXXXXXXXXXX
· · ·
Kernel Memory

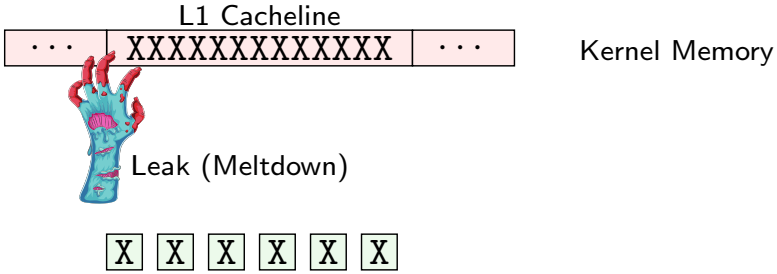

L1 Cacheline
· · ·
XXXXXXXXXXXXX
· · ·
Kernel Memory
Leak (Meltdown)
X

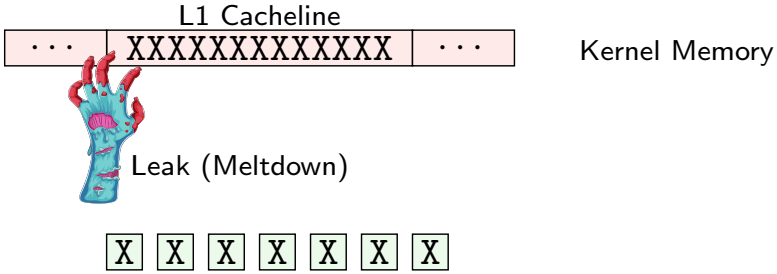

L1 Cacheline
· · ·
XXXXXXXXXXXXX
· · ·
Kernel Memory
Leak (Meltdown)
X
X

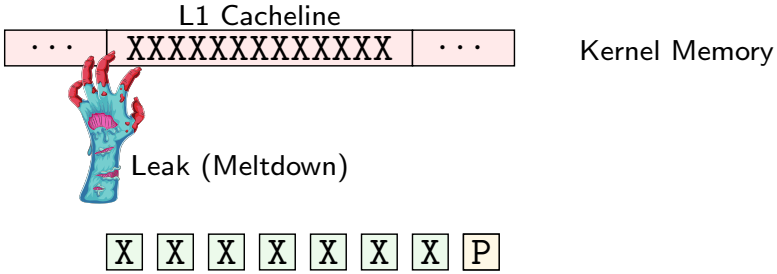

L1 Cacheline
· · ·
XXXXXXXXXXXXX
· · ·
Kernel Memory
Leak (Meltdown)
X
X
X

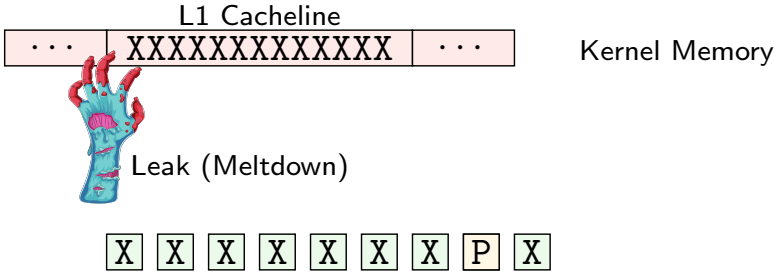

L1 Cacheline
· · ·
XXXXXXXXXXXXX
· · ·
Kernel Memory
Leak (Meltdown)
X
X
X
X

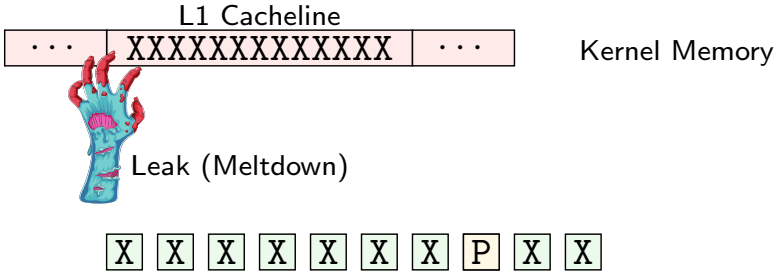

L1 Cacheline
· · ·
XXXXXXXXXXXXX
· · ·
Kernel Memory
Leak (Meltdown)
X X X X X

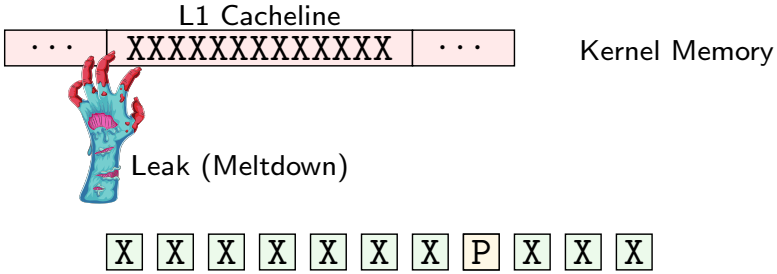

L1 Cacheline
· · ·
XXXXXXXXXXXXX
· · ·
Kernel Memory
Leak (Meltdown)
X X X X X X

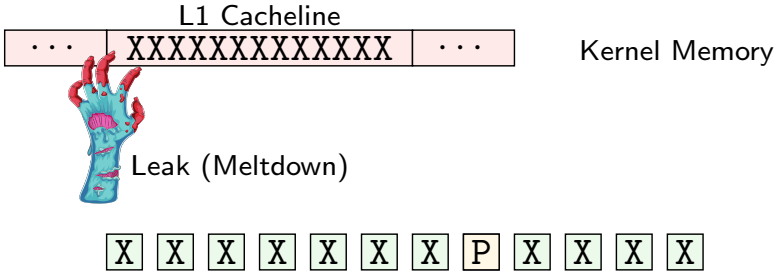

L1 Cacheline
· · ·
XXXXXXXXXXXXX
· · ·
Kernel Memory
Leak (Meltdown)
X X X X X X X

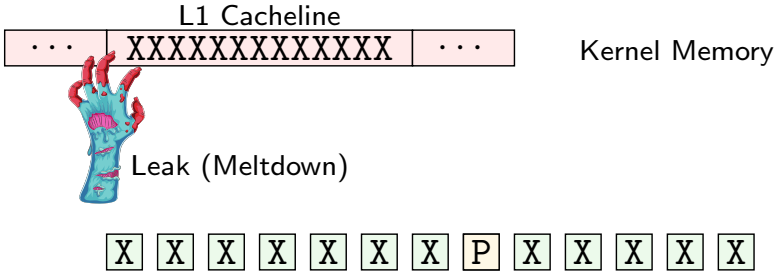

L1 Cacheline
· · ·
XXXXXXXXXXXXX
· · ·
Kernel Memory
Leak (Meltdown)
X X X X X X X P

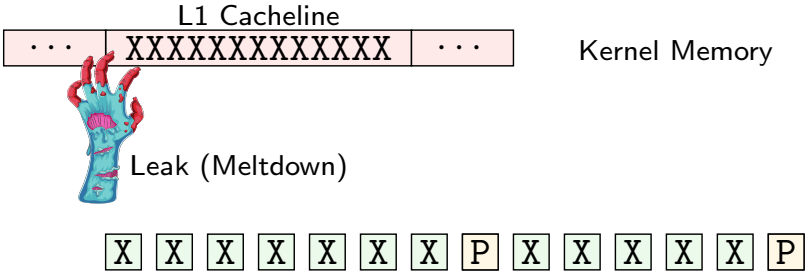

L1 Cacheline
· · ·
XXXXXXXXXXXXX
· · ·
Kernel Memory
Leak (Meltdown)
X X X X X X X P X

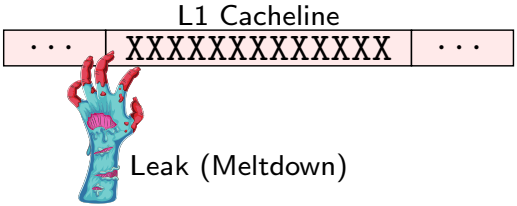

L1 Cacheline
· · ·
XXXXXXXXXXXXX
· · ·
Kernel Memory
Leak (Meltdown)
X X X X X X X P X X

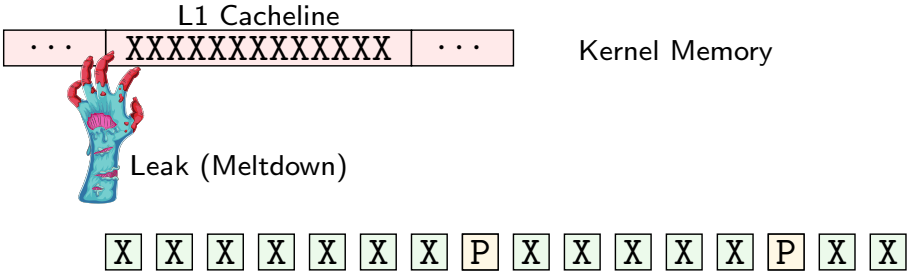

L1 Cacheline
· · ·
XXXXXXXXXXXXX
· · ·
Kernel Memory
Leak (Meltdown)
X X X X X X X P X X X

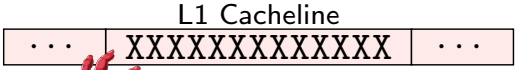

L1 Cacheline
· · ·
XXXXXXXXXXXXX
· · ·
Kernel Memory
Leak (Meltdown)
X X X X X X X P X X X X


L1 Cacheline
· · ·
XXXXXXXXXXXXX
· · ·
Kernel Memory
Leak (Meltdown)
X X X X X X X P X X X X X


L1 Cacheline
· · ·
XXXXXXXXXXXXX
· · ·
Kernel Memory
Leak (Meltdown)
X X X X X X X P X X X X X P


L1 Cacheline
· · ·
XXXXXXXXXXXXX
· · ·
Kernel Memory
Leak (Meltdown)
X X X X X X X P X X X X X P X


L1 Cacheline
· · ·
XXXXXXXXXXXXX
· · ·
Kernel Memory
Leak (Meltdown)
X X X X X X X P X X X X X P X X



X	X	X	X	X	X	X	P	X	X	X	X	X	P	X	X	X

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

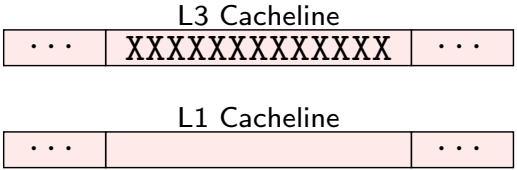
10.4 kB/s

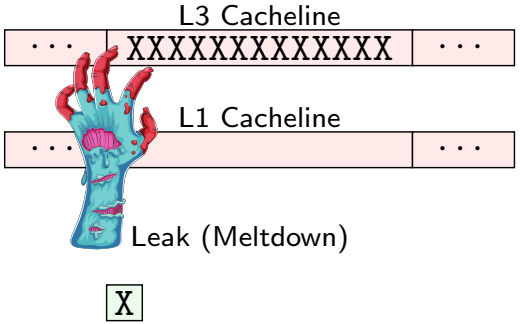
Meltdown L3
Leakage Rate

10.4 kB/s

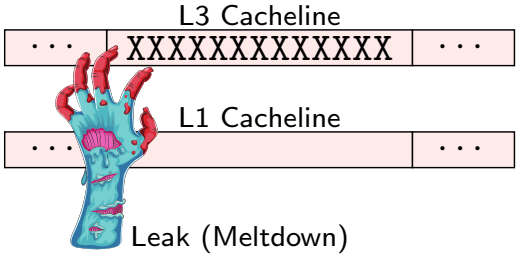
Meltdown L3
Error Rate

0.02 %

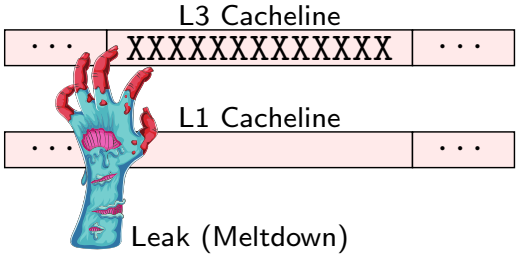

L3 Cacheline
· · ·
XXXXXXXXXXXXX
· · ·
L1 Cacheline
· · ·
· · ·
Kernel Memory

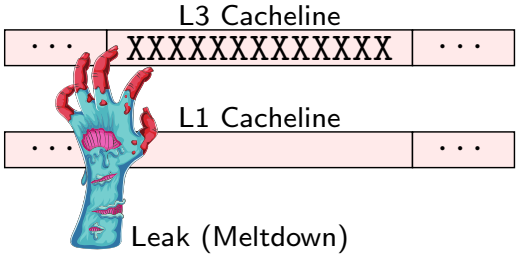


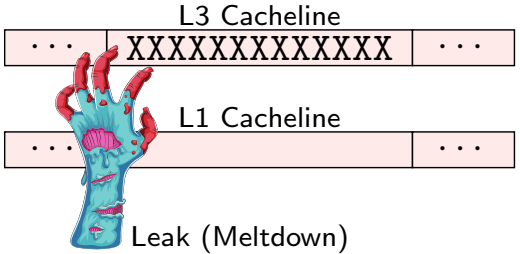
Kernel Memory

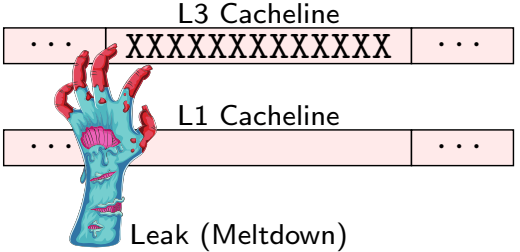


Kernel Memory

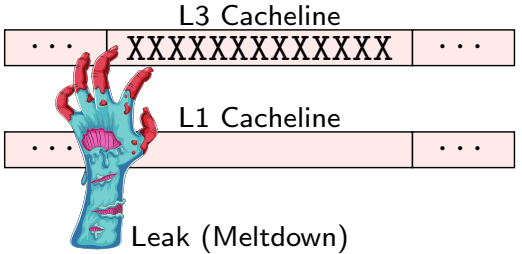

L3 Cacheline
XXXXXXXXXXXXX
L1 Cacheline
Leak (Meltdown)
X
X
P
Kernel Memory


L3 Cacheline
XXXXXXXXXXXXX
L1 Cacheline
Leak (Meltdown)
X X P X
Kernel Memory

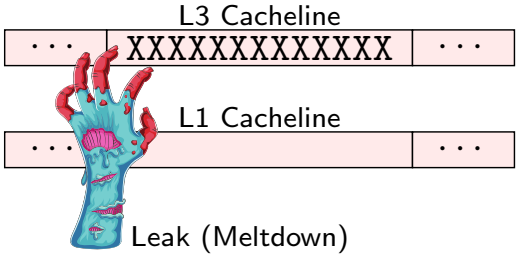

L3 Cacheline
XXXXXXXXXXXXX
L1 Cacheline
Leak (Meltdown)
X X P X Q
Kernel Memory



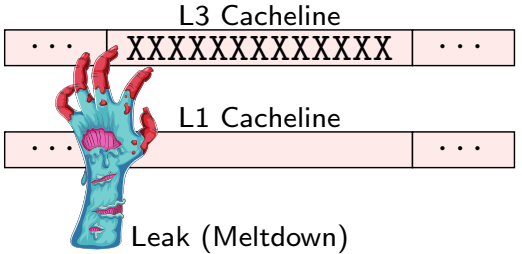
Kernel Memory

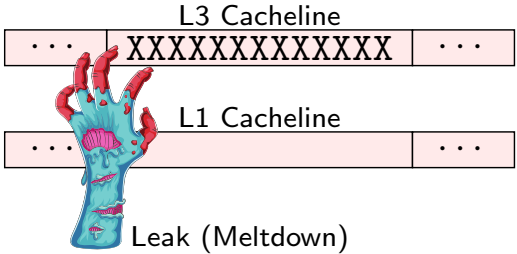


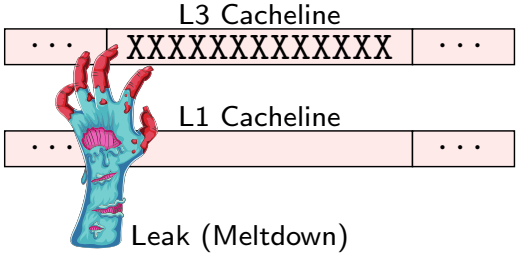
Kernel Memory

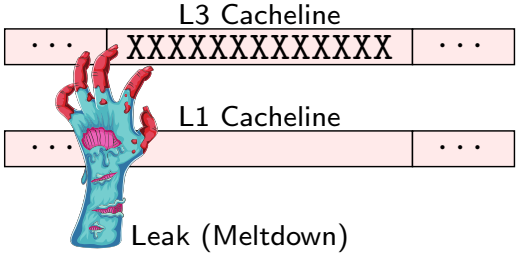


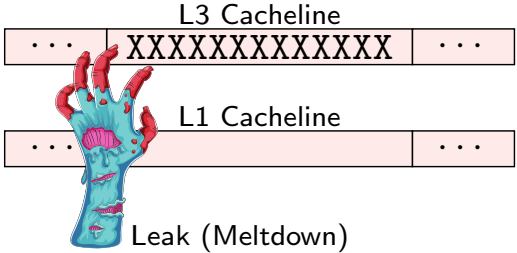
Kernel Memory

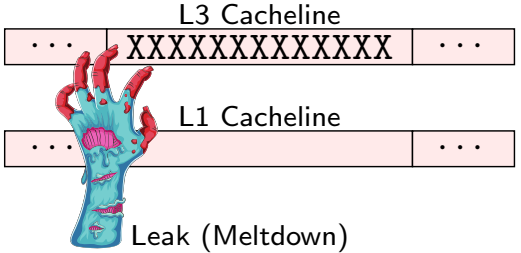

L3 Cacheline
XXXXXXXXXXXXX
L1 Cacheline
Kernel Memory
Leak (Meltdown)
X X P X Q P X X X

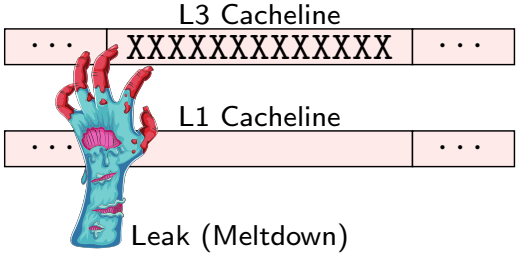

L3 Cacheline
XXXXXXXXXXXXX
L1 Cacheline
Kernel Memory
Leak (Meltdown)
X X P X Q P X X X N

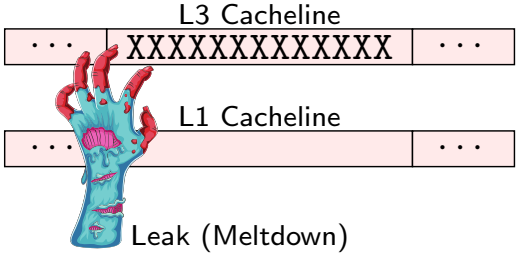

L3 Cacheline
XXXXXXXXXXXXX
L1 Cacheline
Kernel Memory
Leak (Meltdown)
X X P X Q P X X X N P

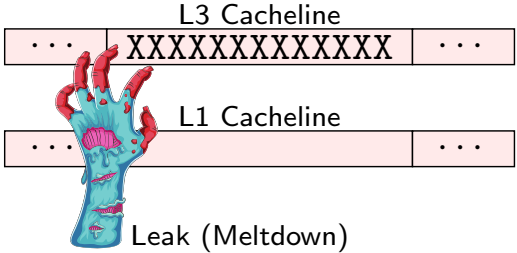

L3 Cacheline
XXXXXXXXXXXXX
L1 Cacheline
Kernel Memory
Leak (Meltdown)
X X P X Q P X X X N P N


L3 Cacheline
XXXXXXXXXXXXX
L1 Cacheline
Kernel Memory
Leak (Meltdown)
X X P X Q P X X X N P N X


L3 Cacheline
XXXXXXXXXXXXX
L1 Cacheline
Kernel Memory
Leak (Meltdown)
X X P X Q P X X X N P N X N


L3 Cacheline
XXXXXXXXXXXXX
L1 Cacheline
Kernel Memory
Leak (Meltdown)
X X P X Q P X X X N P N X N X


L3 Cacheline
XXXXXXXXXXXXX
L1 Cacheline
Kernel Memory
Leak (Meltdown)
X X P X Q P X X X N P N X N X X


L3 Cacheline
XXXXXXXXXXXXX
L1 Cacheline
Kernel Memory
Leak (Meltdown)
X X P X Q P X X X N P N X N X X X

How to get rid of the noise?

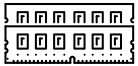
Just repeat the experiment, noise averages out

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

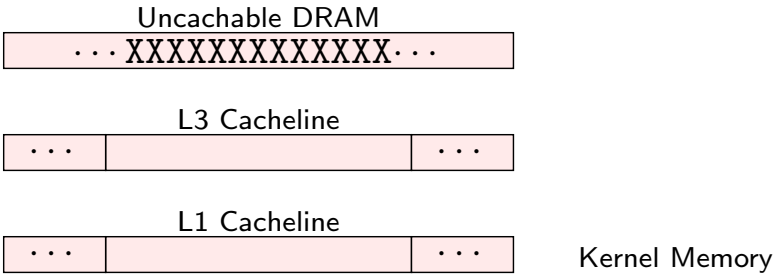
- BlackHat USA: “Meltdown: Basics, Details, Consequences” (📅 August 9, 2018)

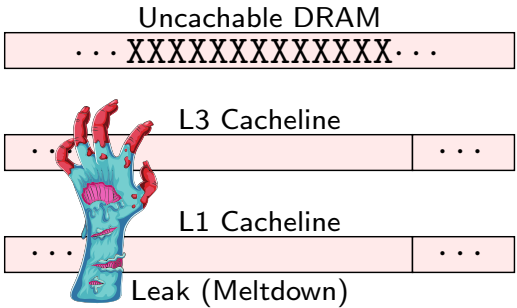
Uncachable memory

black hat



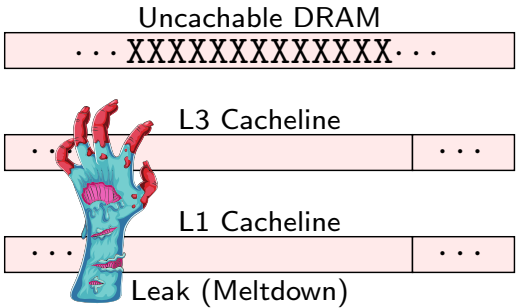
- Mark pages in page tables as UC (uncachable)
  - Every read or write operation will go to main memory
- If the attacker can trigger a legitimate load (system call, ...) on the same CPU core, the data still can be leaked
- Meltdown might read the data from one of the fill buffers
  - as they are shared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


Uncachable DRAM
· · · XXXXXXXXXXXXX · · ·
L3 Cacheline
· · ·
· · ·
L1 Cacheline
· · ·
· · ·
Kernel Memory



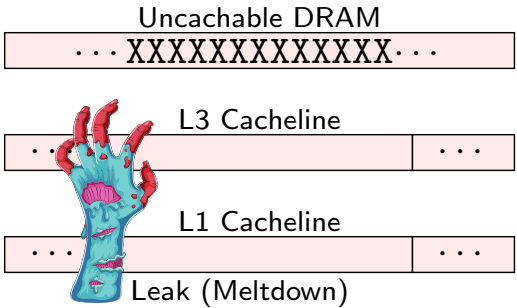
Kernel Memory

S



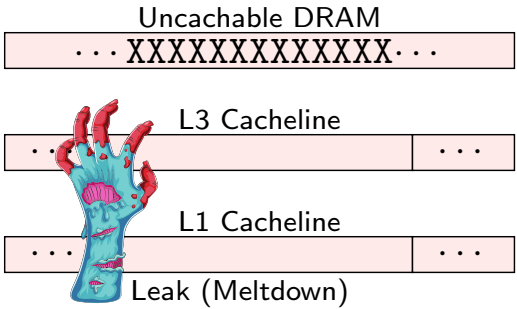
Kernel Memory

S N



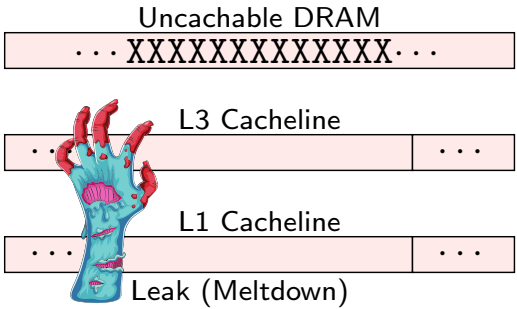
Kernel Memory

S N P



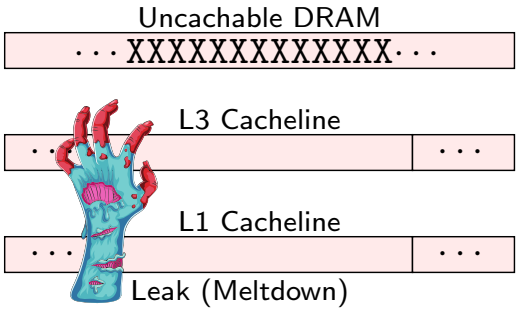
Kernel Memory

S N P P



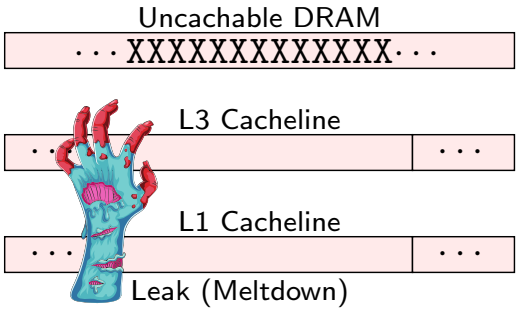
Kernel Memory

S N P P Q



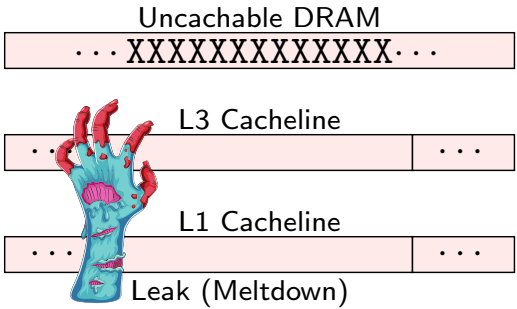
Kernel Memory

S N P P Q P



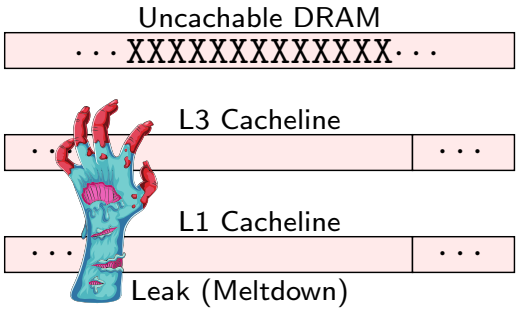
Kernel Memory

S N P P Q P S



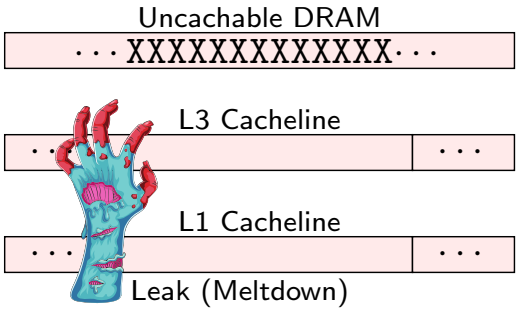
Kernel Memory

S N P P Q P S N



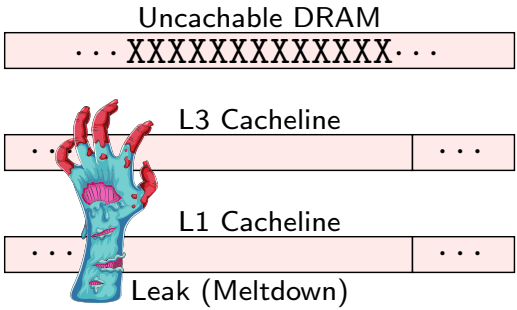
Kernel Memory

S N P P Q P S N P



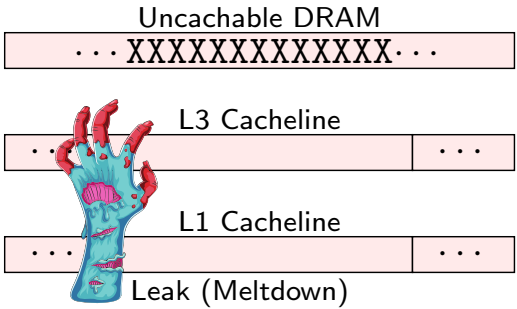
Kernel Memory

S	N	P	P	Q	P	S	N	P	N



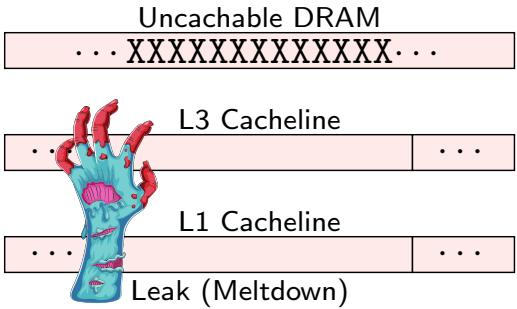
Kernel Memory

S	N	P	P	Q	P	S	N	P	N	P



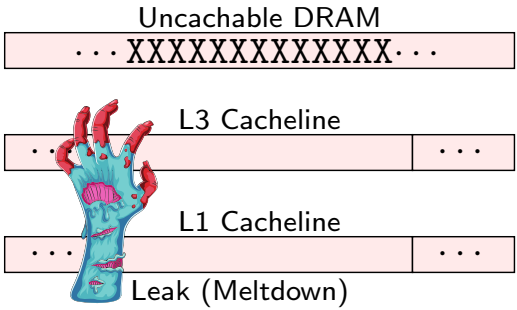
Kernel Memory

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

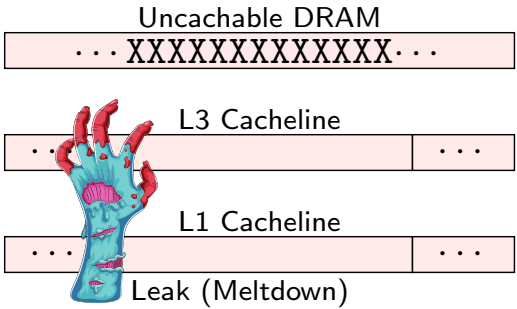


Kernel Memory

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

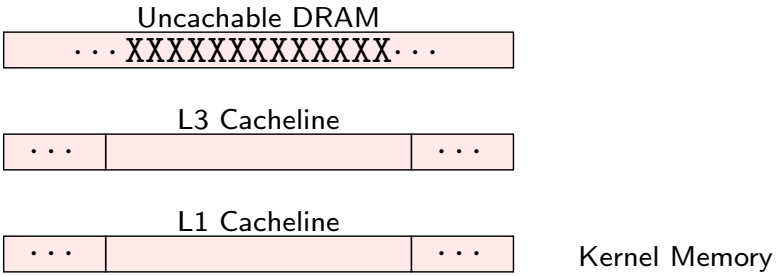


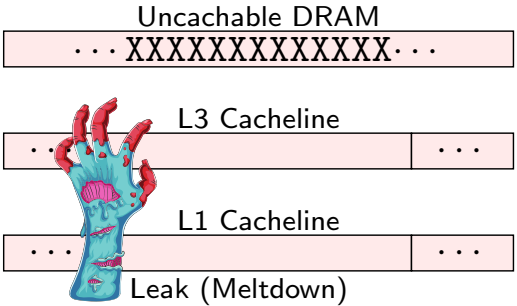
S N P P Q P S N P N P N Q N



Kernel Memory

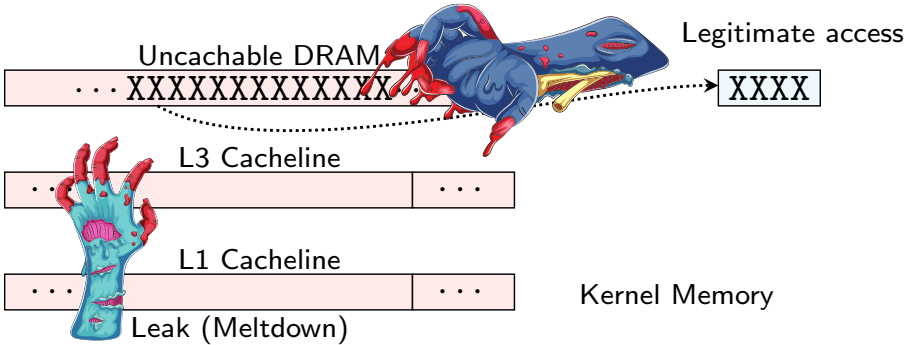
S N P P Q P S N P N P N Q N N


Uncachable DRAM
· · · XXXXXXXXXXXXX · · ·
L3 Cacheline
· · ·
· · ·
L1 Cacheline
· · ·
· · ·
Kernel Memory

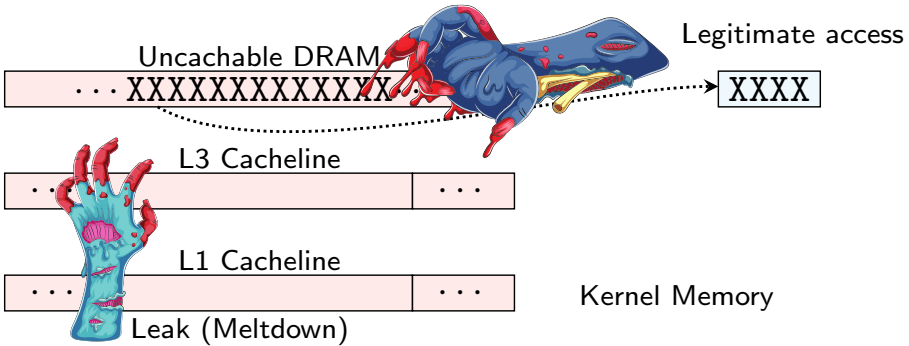


Kernel Memory

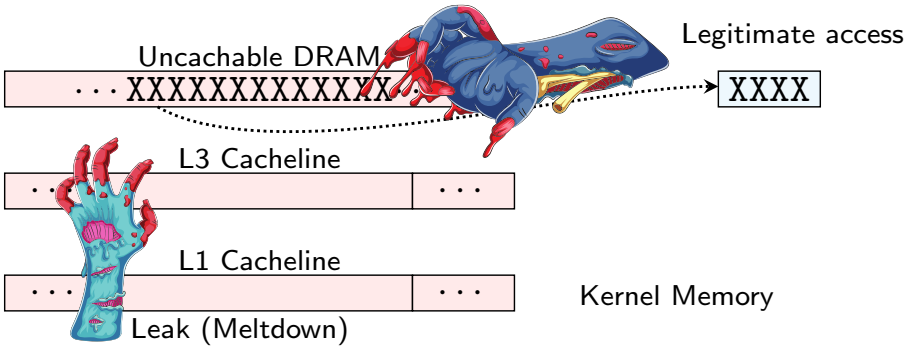
S

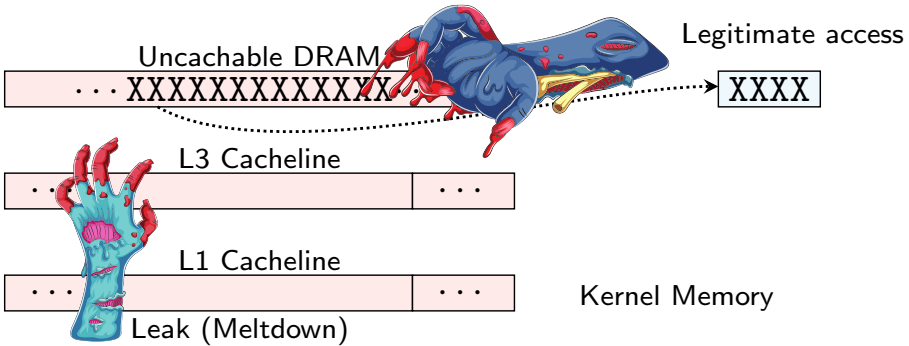

Uncachable DRAM
XXXXXXXXXXXXXX
Legitimate access
XXXX
L3 Cacheline
L1 Cacheline
Kernel Memory
Leak (Meltdown)

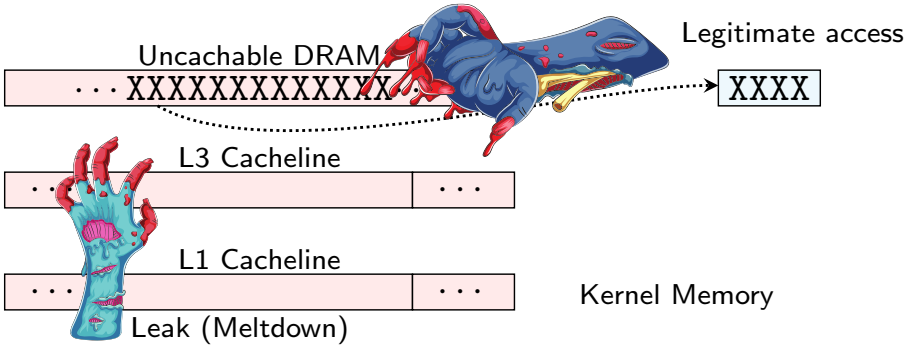
S X


Uncachable DRAM
XXXXXXXXXXXXXX
Legitimate access
XXXX
L3 Cacheline
L1 Cacheline
Kernel Memory
Leak (Meltdown)

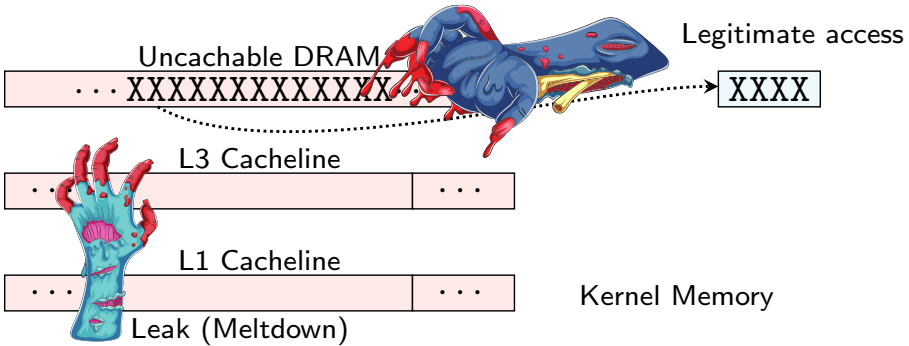
S X X


Uncachable DRAM
XXXXXXXXXXXXXX
Legitimate access
XXXX
L3 Cacheline
L1 Cacheline
Kernel Memory
Leak (Meltdown)
S
X
X
X

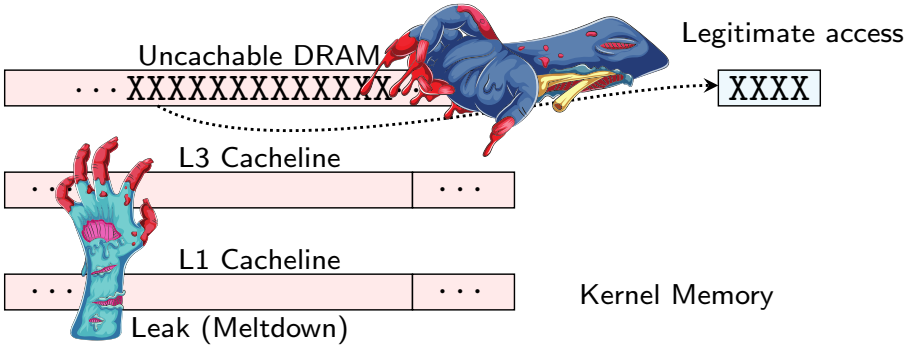

Uncachable DRAM
XXXXXXXXXXXXXX
Legitimate access
XXXX
L3 Cacheline
L1 Cacheline
Kernel Memory
Leak (Meltdown)
S X X X X


Uncachable DRAM
XXXXXXXXXXXXXX
Legitimate access
XXXX
L3 Cacheline
L1 Cacheline
Kernel Memory
Leak (Meltdown)

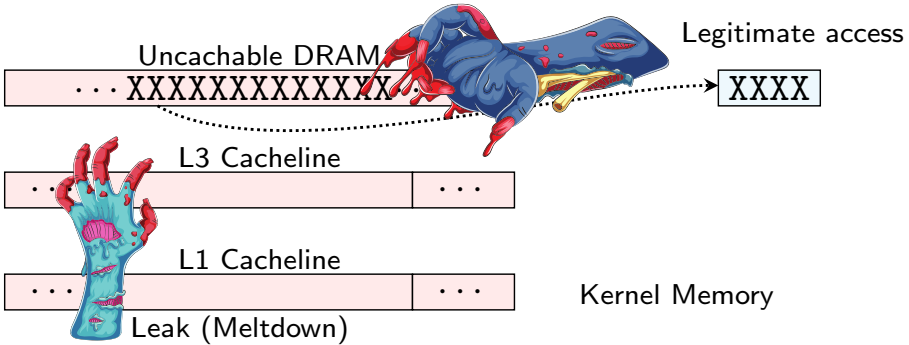
S X X X X P


Uncachable DRAM
XXXXXXXXXXXXXX
Legitimate access
XXXX
L3 Cacheline
L1 Cacheline
Kernel Memory
Leak (Meltdown)

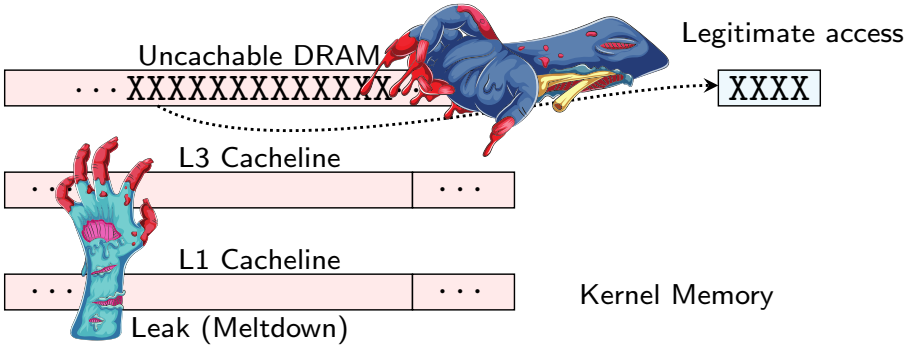
S X X X X P X


Uncachable DRAM
XXXXXXXXXXXXXX
Legitimate access
XXXX
L3 Cacheline
L1 Cacheline
Kernel Memory
Leak (Meltdown)

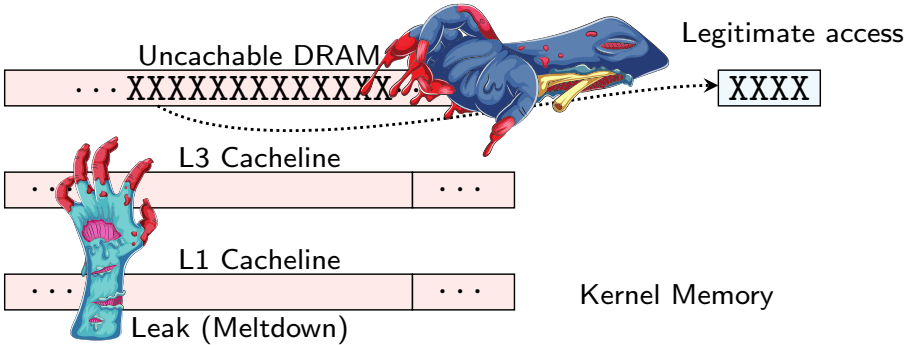
S X X X X P X N


Uncachable DRAM
XXXXXXXXXXXXXX
Legitimate access
XXXX
L3 Cacheline
L1 Cacheline
Kernel Memory
Leak (Meltdown)

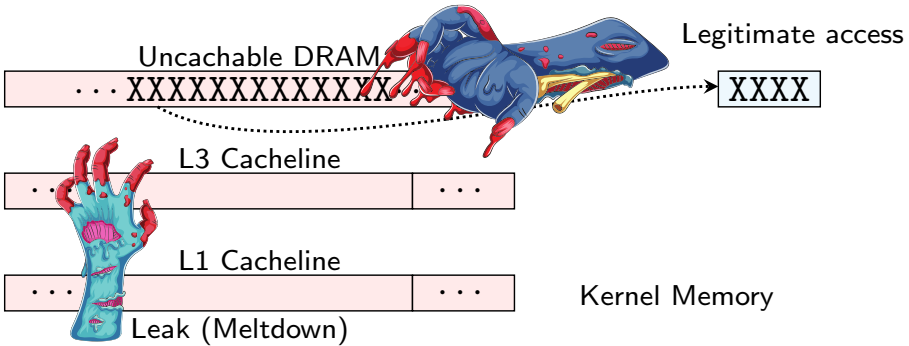
S	X	X	X	X	P	X	N	P


Uncachable DRAM
Legitimate access
XXXXXXXXXXXXXX
XXXX
L3 Cacheline
L1 Cacheline
Kernel Memory
Leak (Meltdown)

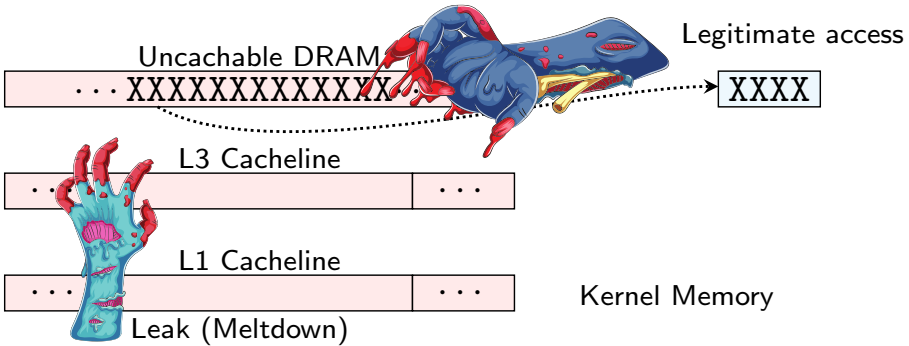
S	X	X	X	X	P	X	N	P	X



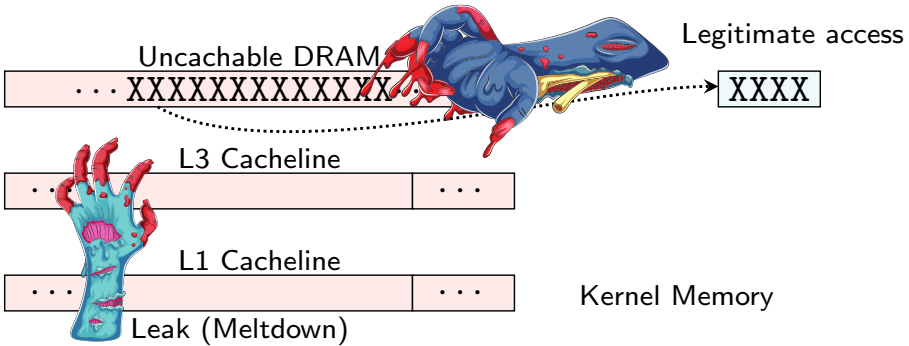
S	X	X	X	X	P	X	N	P	X	X



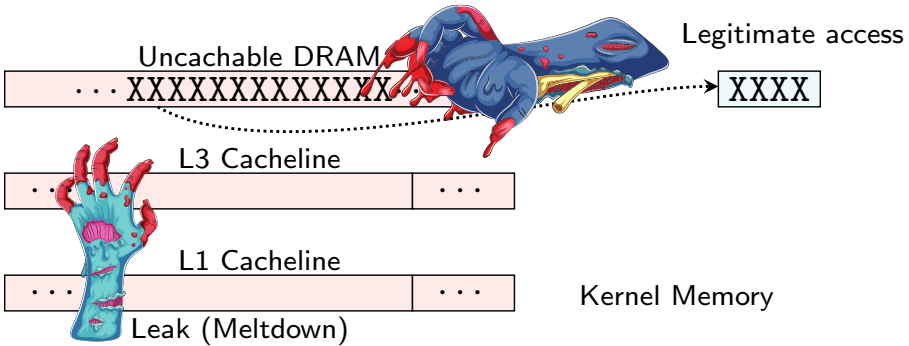
S	X	X	X	X	P	X	N	P	X	X	N



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

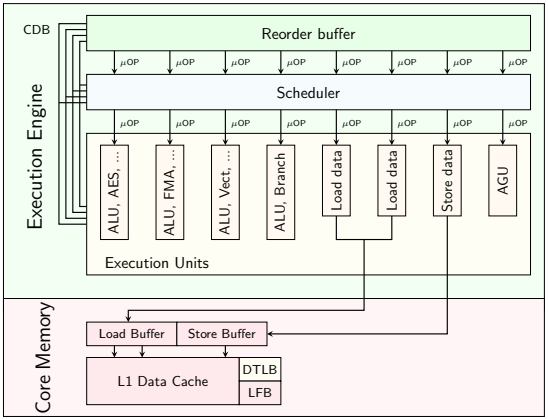
We suspect that Meltdown reads the value from the line fill buffers. As the fill buffers are shared between threads running on the same core, the read to the same address within the Meltdown attack could be served from one of the fill buffers allowing the attack to succeed.

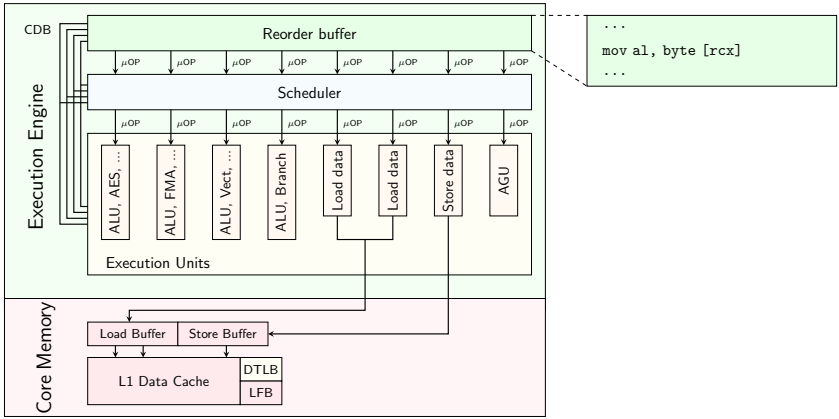
Meltdown: Reading Kernel Memory from User Space

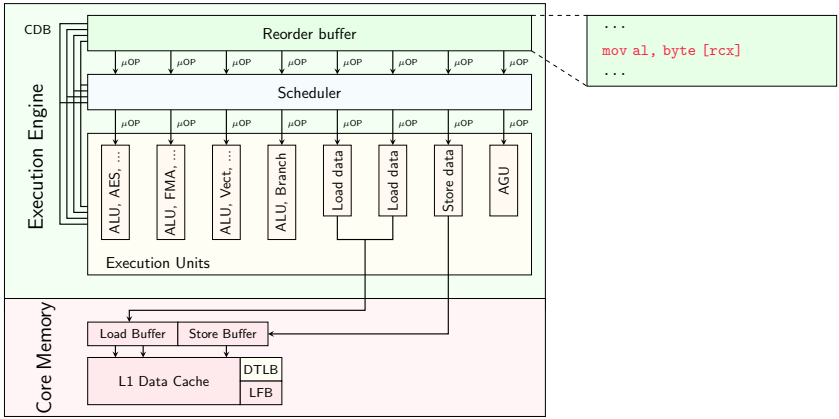
We suspect that Meltdown reads the value from the line fill buffers. As the fill buffers are shared between threads running on the same core, the read to the same address within the Meltdown attack could be served from one of the fill buffers allowing the attack to succeed. However, we leave further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

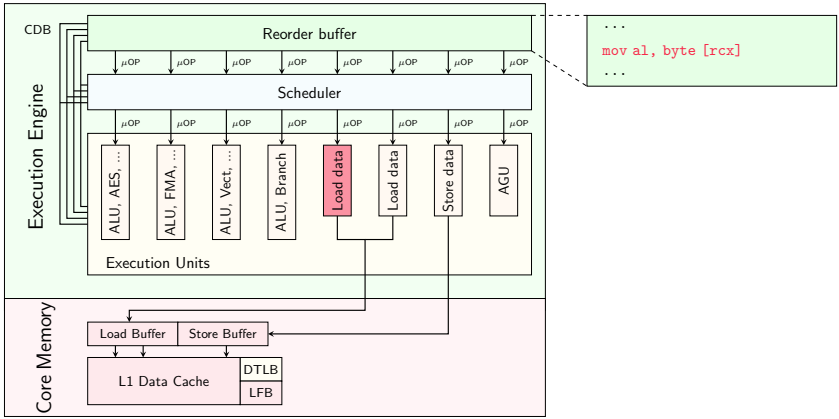

You always leave the shitty jobs to "future you".
Yeah! Fuck future me.

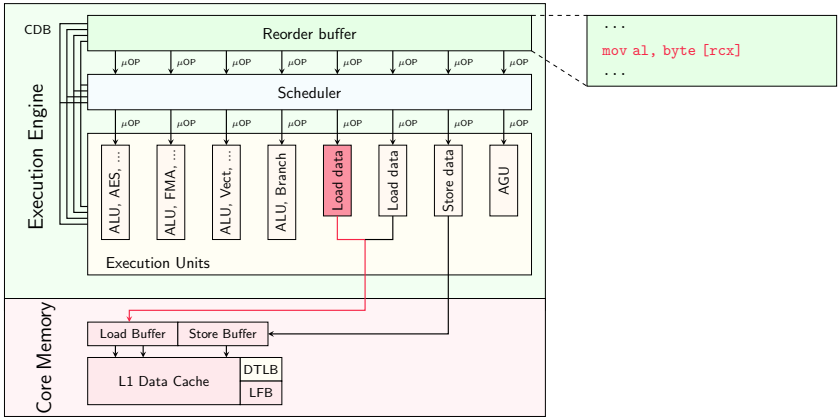

You always leave the shitty jobs to "future you".
MENTAL RESOURCE EXHAUSTION

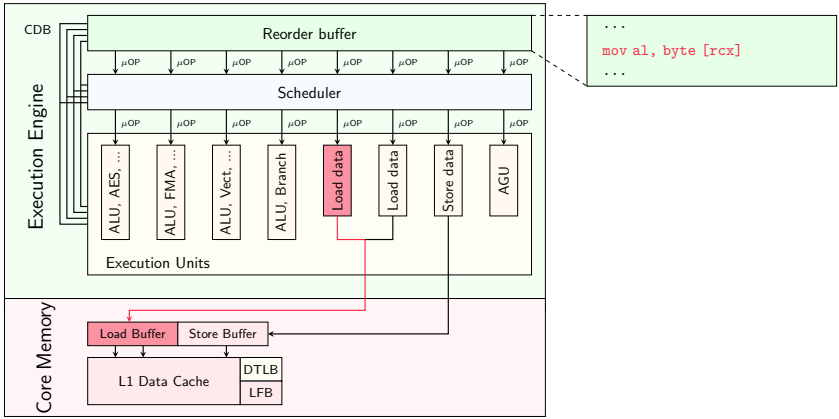

CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB

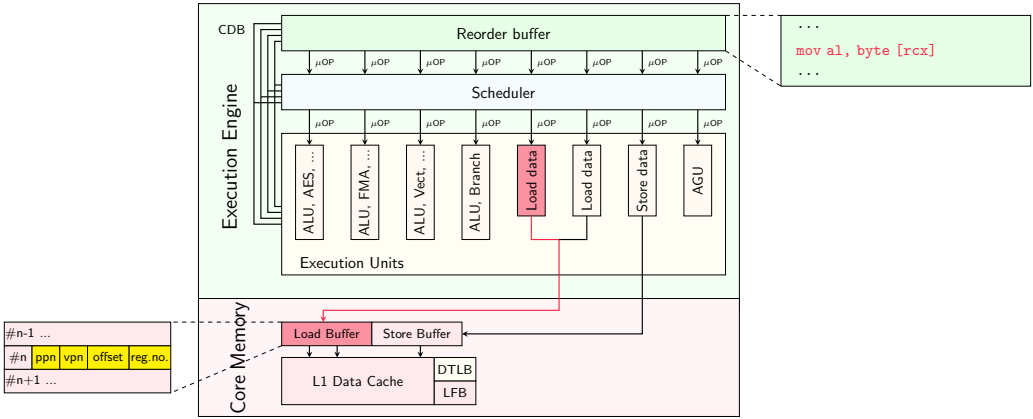

CDB
Reorder buffer
...
mov al, byte [rcx]
...
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB

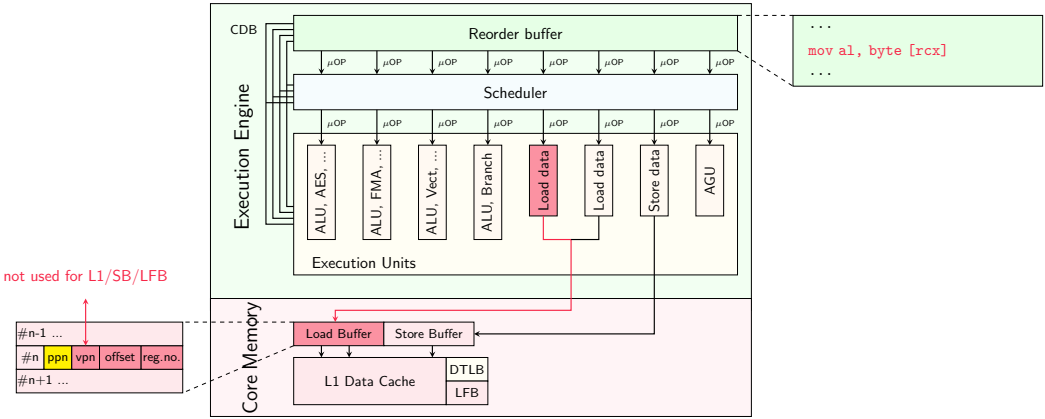

CDB
Reorder buffer
...
mov al, byte [rcx]
...
Execution Engine
μOP
Scheduler
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB

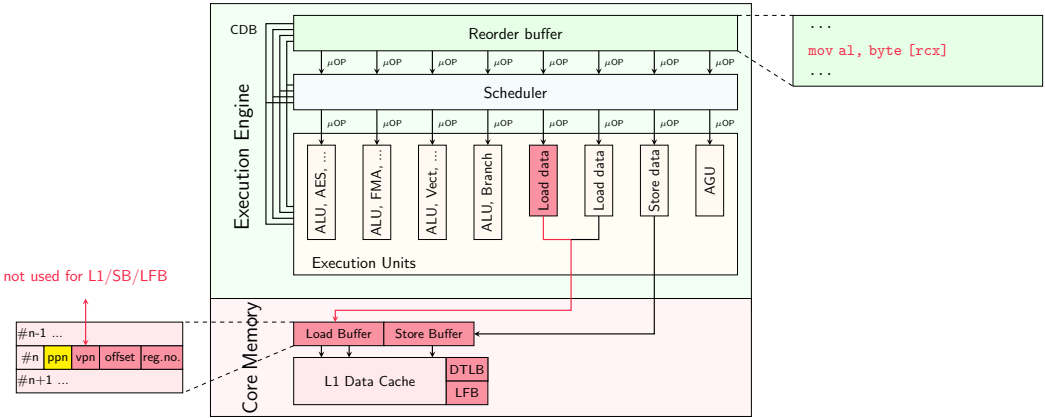

CDB
Reorder buffer
...
mov al, byte [rcx]
...
Scheduler
Execution Engine
μOP
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB


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

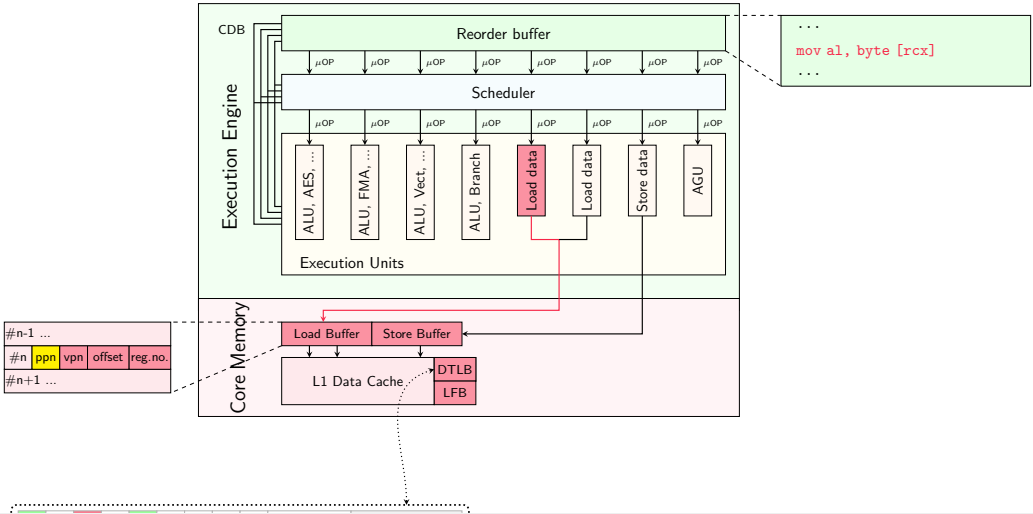

CDB
Reorder buffer
...
mov al, byte [rcx]
...
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB

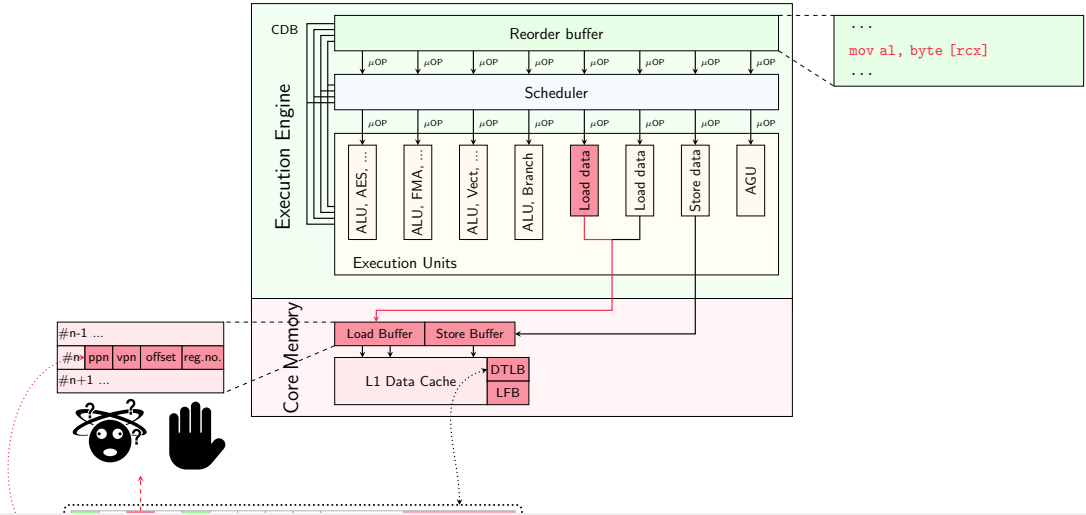

CDB
Reorder buffer
Scheduler
Execution Engine
μOP
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
...
mov al, byte [rcx]
...
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB

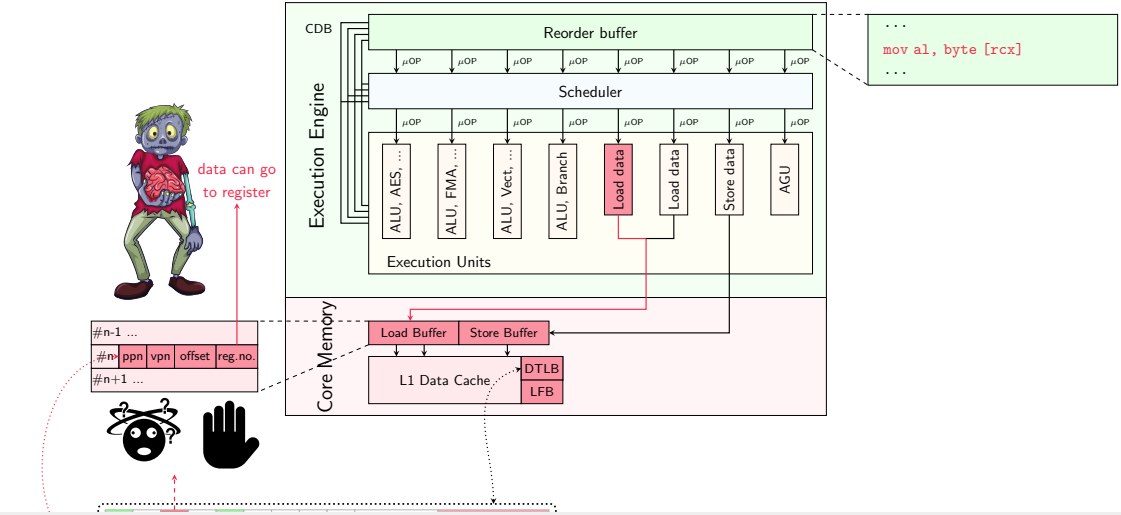

CDB
Reorder buffer
Scheduler
Execution Engine
μOP
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
...
mov al, byte [rcx]
...
not used for L1/SB/LFB
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB



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


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


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


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


BACK
TO
THE FUTURE

There is no noise.

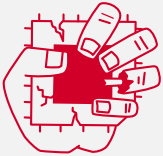
Noise is just

someone else's data

Lemma 1: Noise is someone else's data

$$\lim_{\text{  } \to \varnothing} \text{  } =$$

Lemma 1: Noise is someone else's data

$$\lim_{\text{  } \to \varnothing} \text{  } = \text{  }$$

Deep Dive: Intel Analysis of Microarchitectural Data Sampling

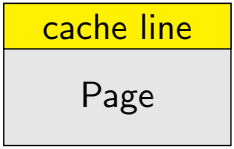
Fill buffers may retain stale data from prior memory requests until a new memory request overwrites the fill buffer.

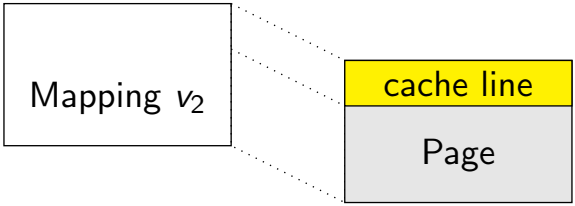
Deep Dive: Intel Analysis of Microarchitectural Data Sampling

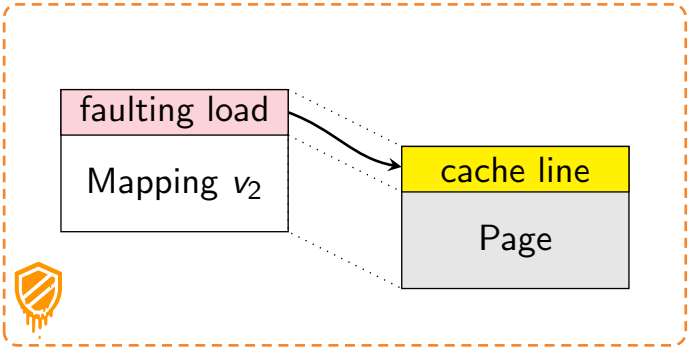
Fill buffers may retain stale data from prior memory requests until a new memory request overwrites the fill buffer. Under certain conditions, the fill buffer may speculatively forward data, including stale data,

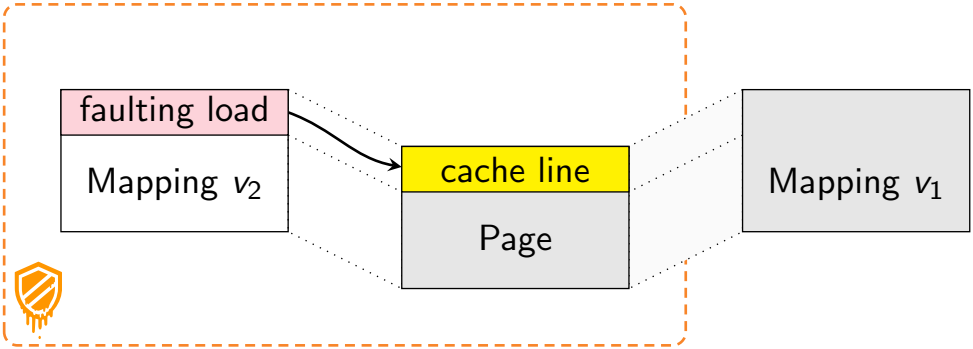
Deep Dive: Intel Analysis of Microarchitectural Data Sampling

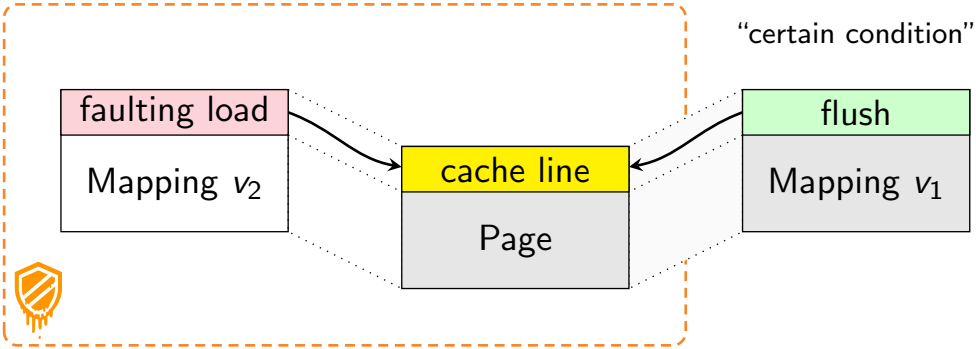
Fill buffers may retain stale data from prior memory requests until a new memory request overwrites the fill buffer. Under certain conditions, the fill buffer may speculatively forward data, including stale data, to a load operation that will cause a fault/assist.

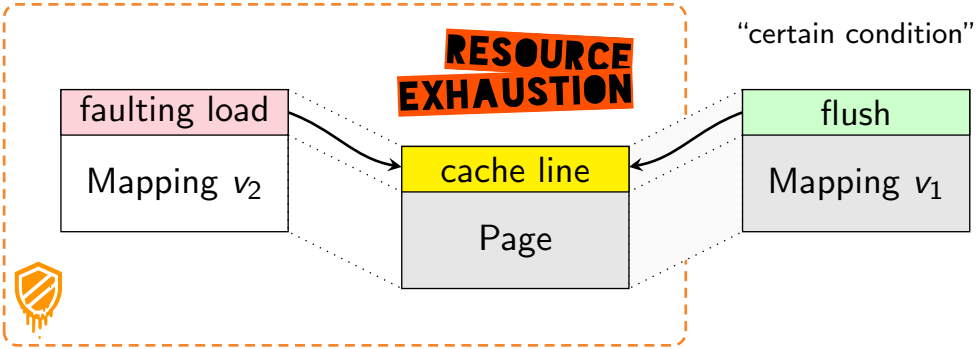

cache line
Page

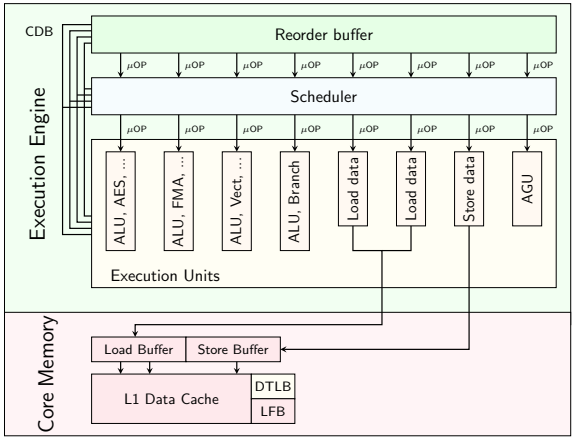

Mapping v_2
cache line
Page

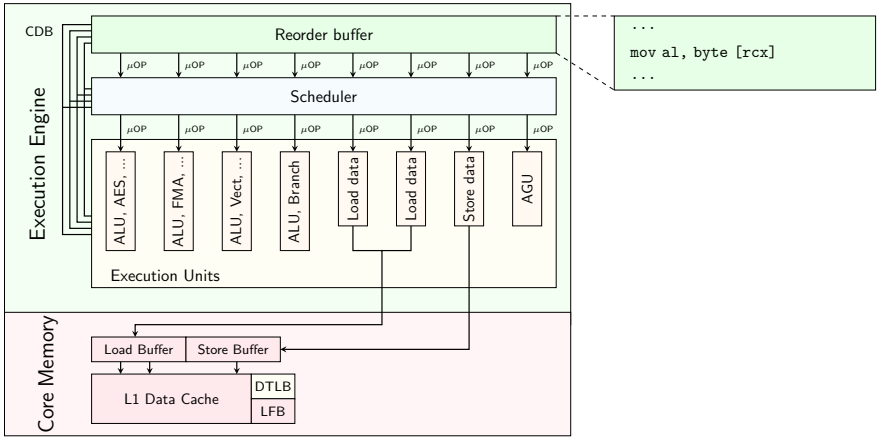

faulting load
Mapping v_2
cache line
Page

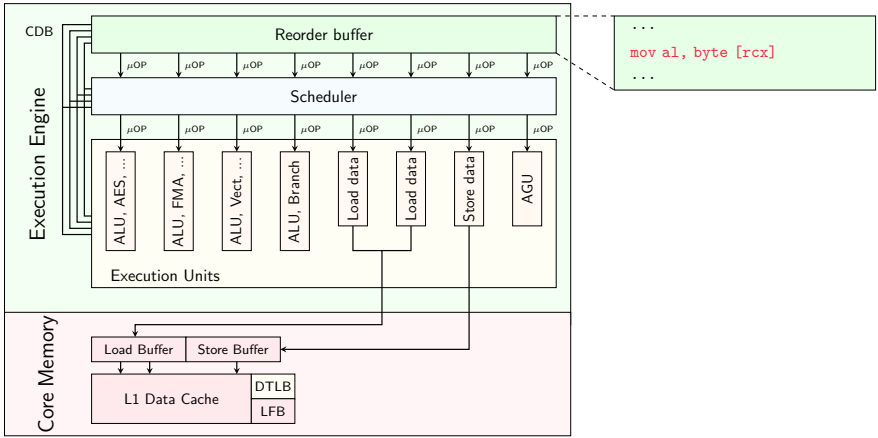

faulting load
Mapping v_2
cache line
Page
Mapping v_1

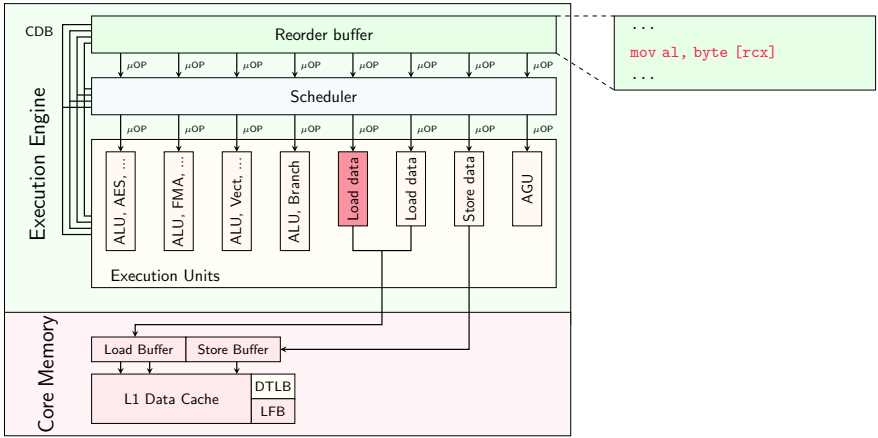

"certain condition"
faulting load
Mapping v_2
cache line
Page
flush
Mapping v_1

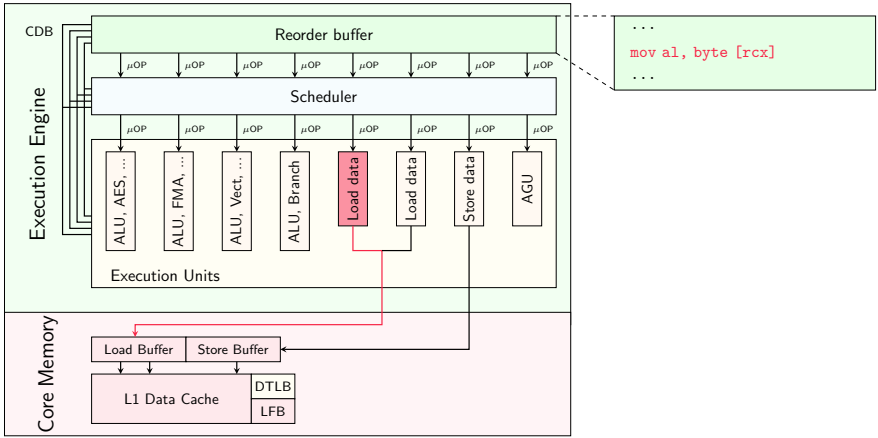

RESOURCE EXHAUSTION
"certain condition"
faulting load
Mapping v_2
cache line
Page
flush
Mapping v_1

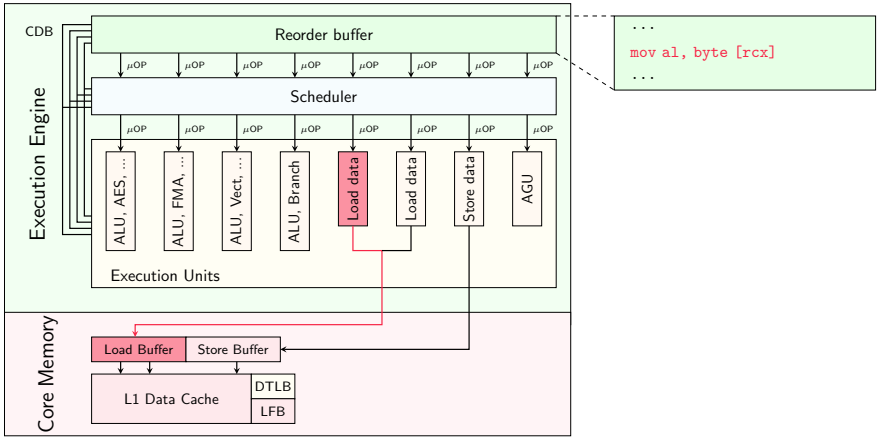

CDB
Reorder buffer
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB

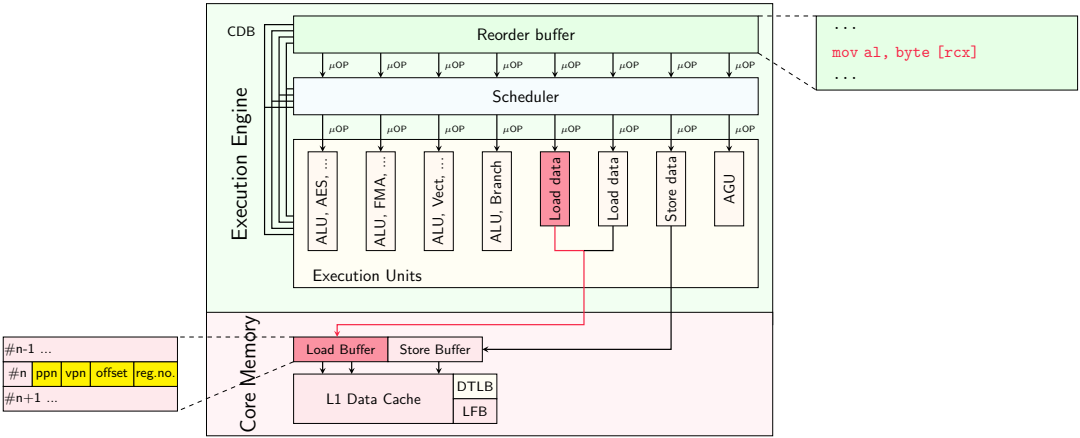

CDB
Reorder buffer
...
mov al, byte [rcx]
...
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB

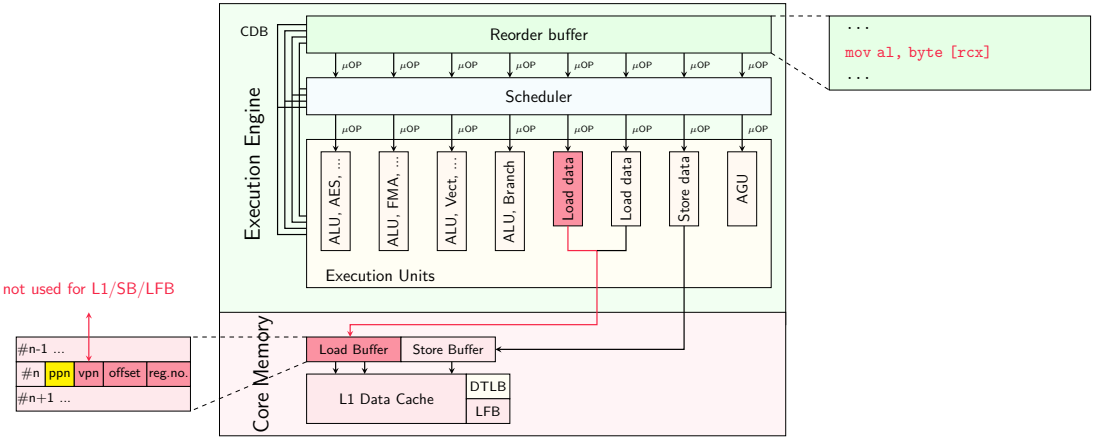

CDB
Reorder buffer
...
mov al, byte [rcx]
...
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB

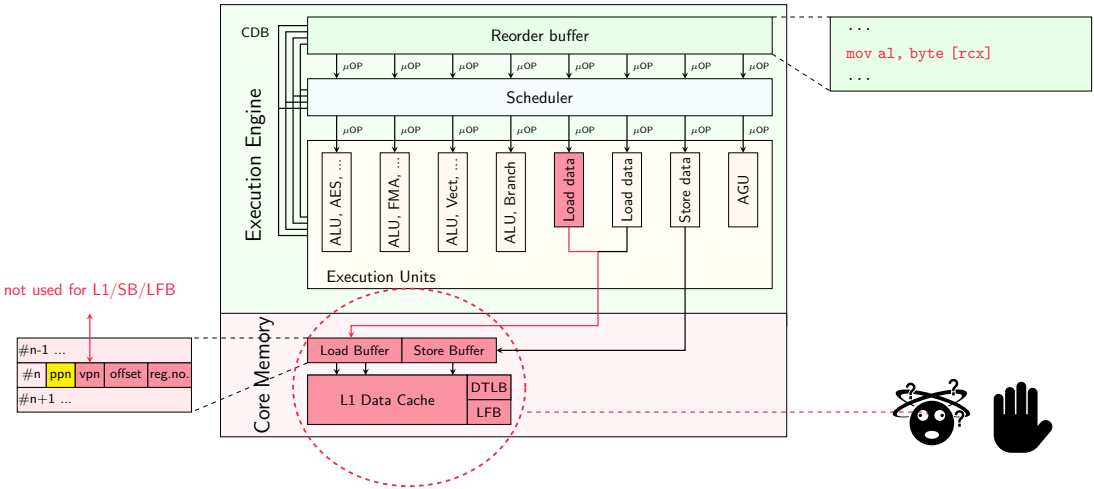

CDB
Reorder buffer
...
mov al, byte [rcx]
...
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB


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

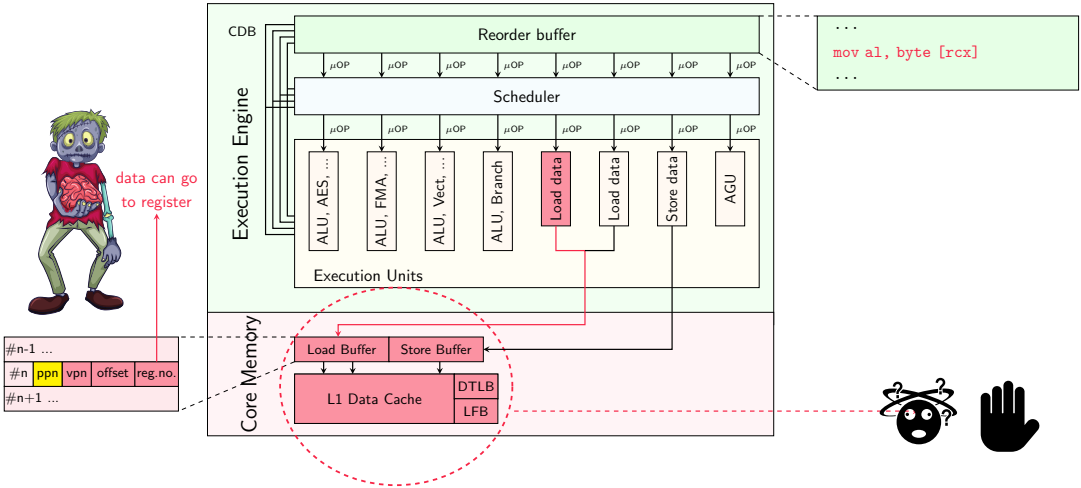

CDB
Reorder buffer
...
mov al, byte [rcx]
...
μOP
Scheduler
Execution Engine
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB


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


CDB
Reorder buffer
...
mov al, byte [rcx]
...
Scheduler
Execution Engine
μOP
ALU, AES, ...
ALU, FMA, ...
ALU, Vect, ...
ALU, Branch
Load data
Load data
Store data
AGU
Execution Units
not used for L1/SB/LFB
#n-1 ...
#n
ppn
vpn
offset
reg.no.
#n+1 ...
Core Memory
Load Buffer
Store Buffer
L1 Data Cache
DTLB
LFB



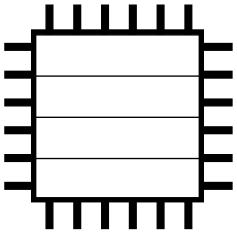
Michael Schwarz (@misc0110), Moritz Lipp (@mlqxyz), Daniel Gruss (@lavados)


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

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]
```

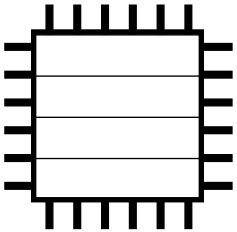


User Memory

	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]
```

Fault



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

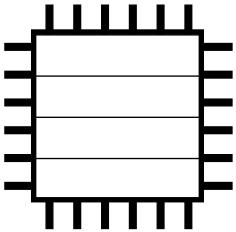
```
char value = faulting[0]
```

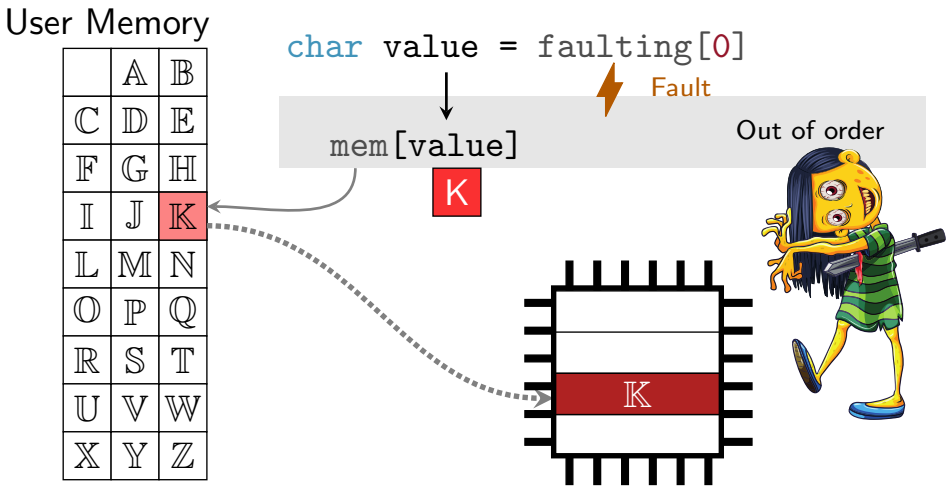
Fault

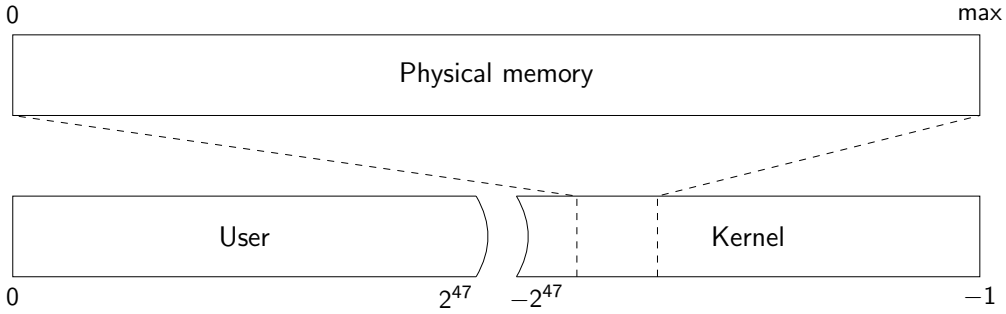
```
mem[value]
```

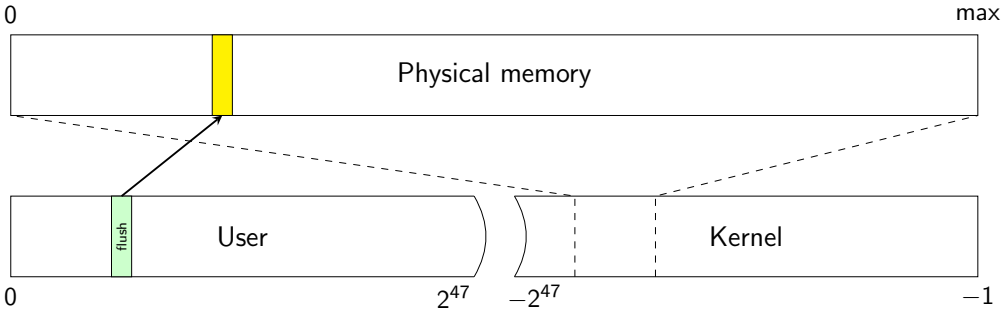
Out of order

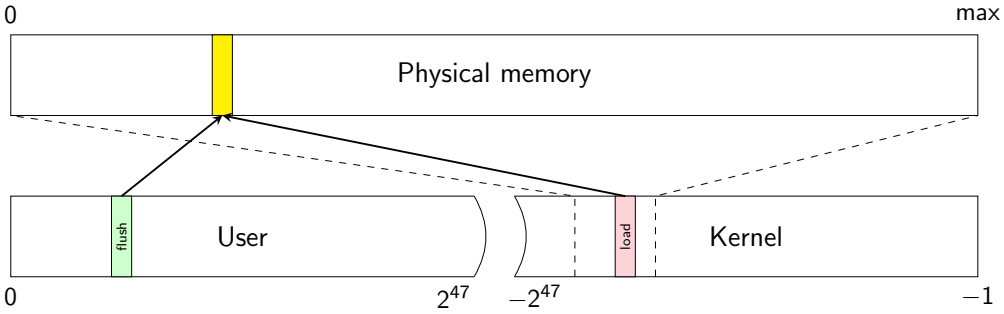
K

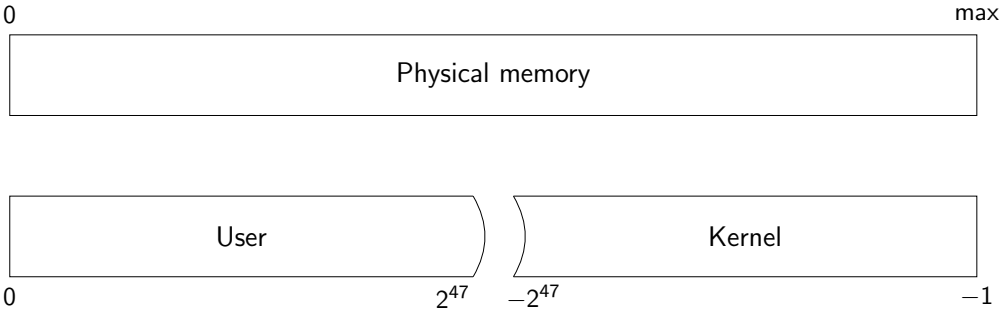


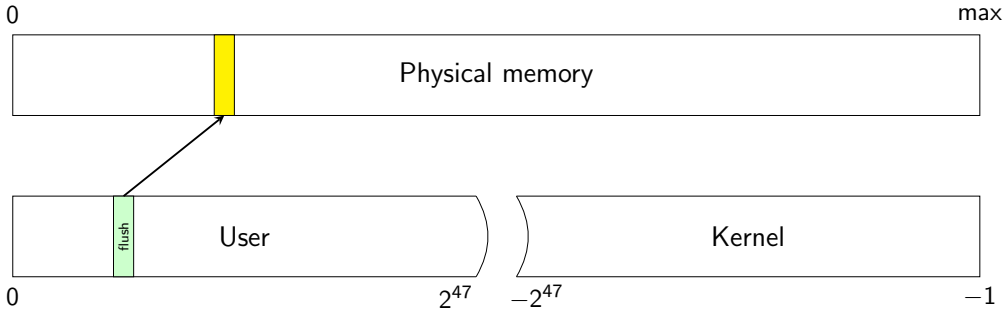

User Memory
A B
C D E
F G H
I J K
L M N
O P Q
R S T
U V W
X Y Z
char value = faulting[0]
Fault
mem[value]
K
Out of order
K

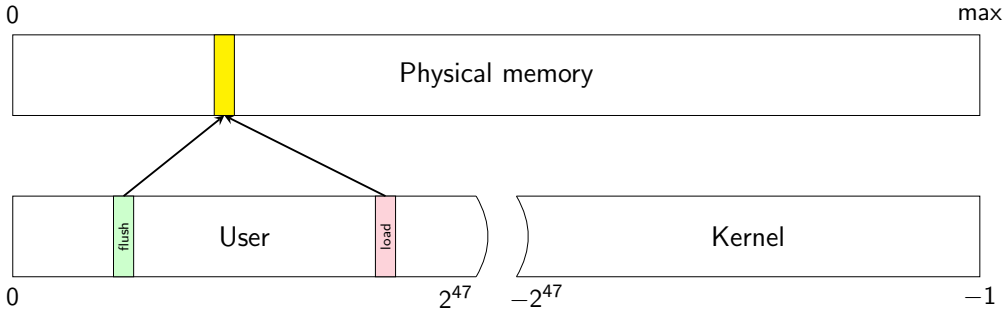

0
max
Physical memory
User
Kernel
0
2^{47}
-2^{47}
-1


0
max
Physical memory
flush
User
Kernel
0
2^{47}
-2^{47}
-1

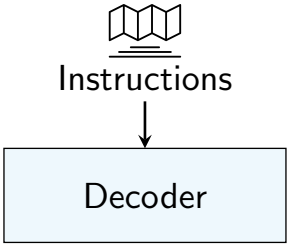

0
max
Physical memory
flush
User
load
Kernel
0
2^{47}
-2^{47}
-1

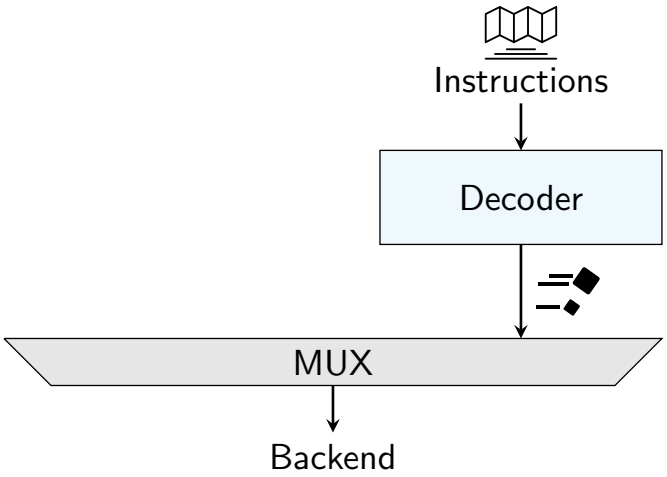

0
max
Physical memory
User
Kernel
0
2^{47}
-2^{47}
-1

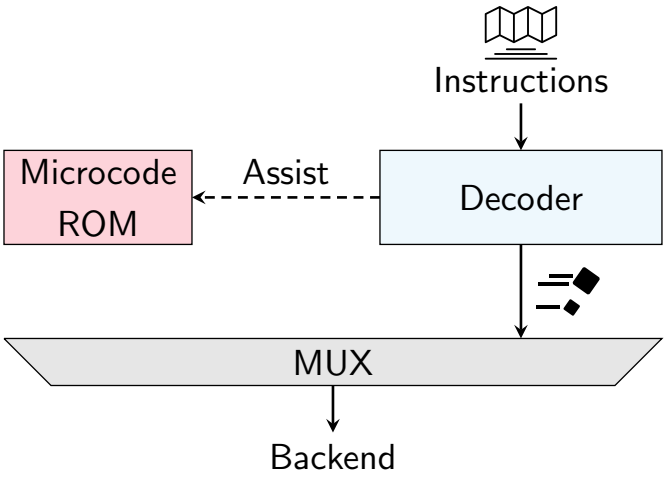

0
max
Physical memory
flush
User
Kernel
0
2^{47}
-2^{47}
-1

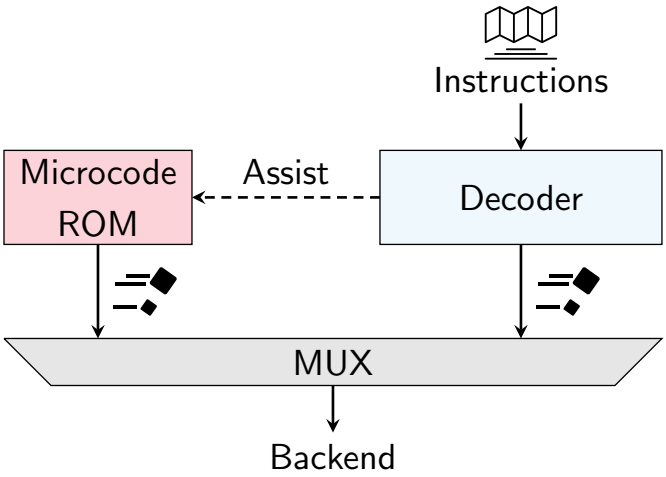

0
max
Physical memory
flush
User
load
Kernel
0
2^{47}
-2^{47}
-1

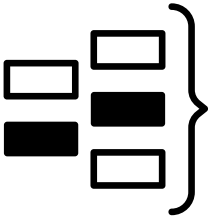
Instructions


Instructions
Decoder

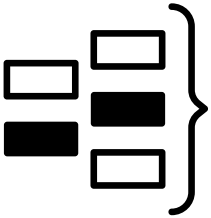

Instructions
Decoder
MUX
Backend


Instructions
Microcode ROM
Assist
Decoder
MUX
Backend


Instructions
Microcode ROM
Assist
Decoder
MUX
Backend

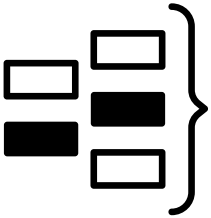


- Microcode assist handles rare cases



- Microcode assist handles rare cases

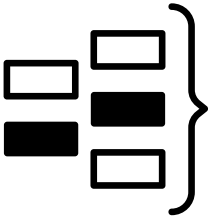
$\rightarrow$ Microarchitectural fault



- 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

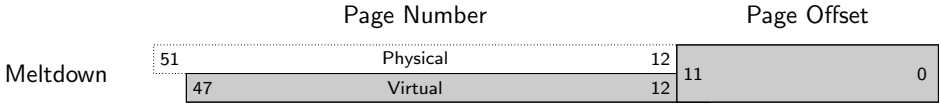
Applications

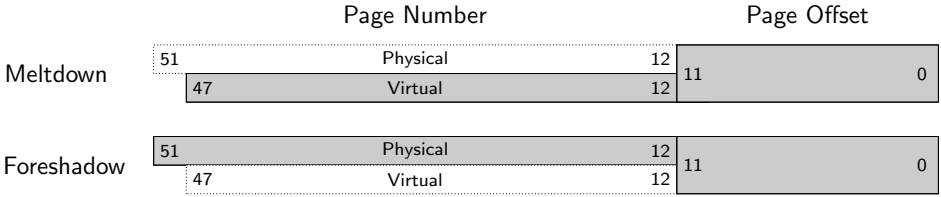
Operating System

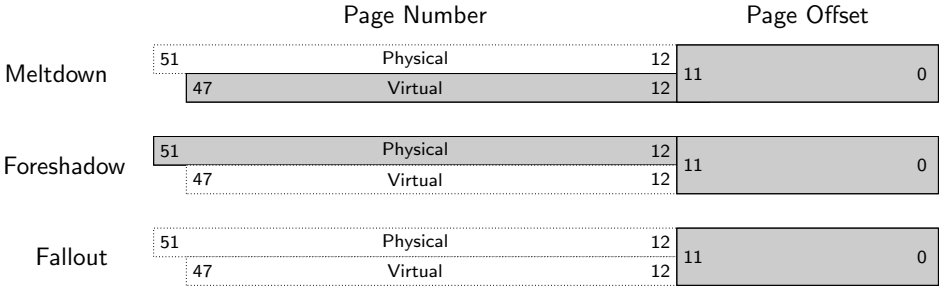
SGX Enclave

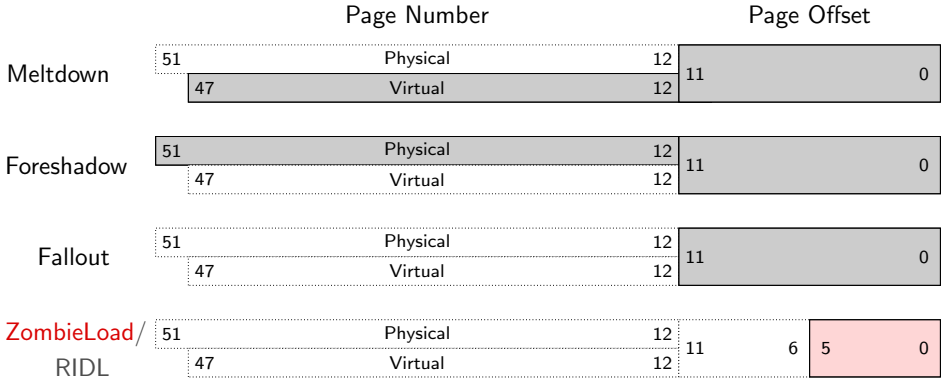
Virtual Machine

Hypervisor

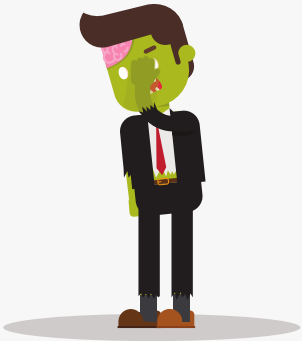

Page Number
Page Offset
Meltdown
51
Physical
12
11
0
47
Virtual
12


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

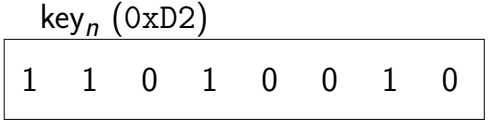

Page Number
Page Offset
Meltdown
51
Physical
12
47
Virtual
12
11
0
Foreshadow
51
Physical
12
47
Virtual
12
11
0
Fallout
51
Physical
12
47
Virtual
12
11
0

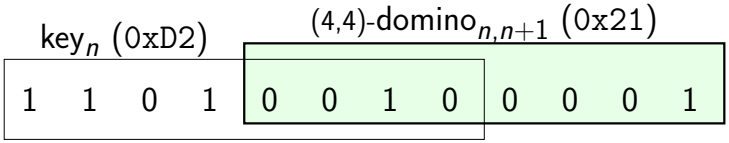

Page Number
Page Offset
Meltdown
51
Physical
12
47
Virtual
12
11
0
Foreshadow
51
Physical
12
47
Virtual
12
11
0
Fallout
51
Physical
12
47
Virtual
12
11
0
ZombieLoad/
RIDL
51
Physical
12
47
Virtual
12
11
6
5
0

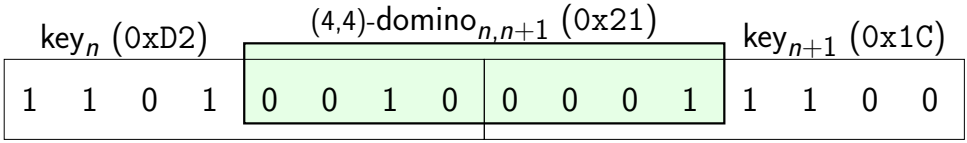
IMPOSSIBLE

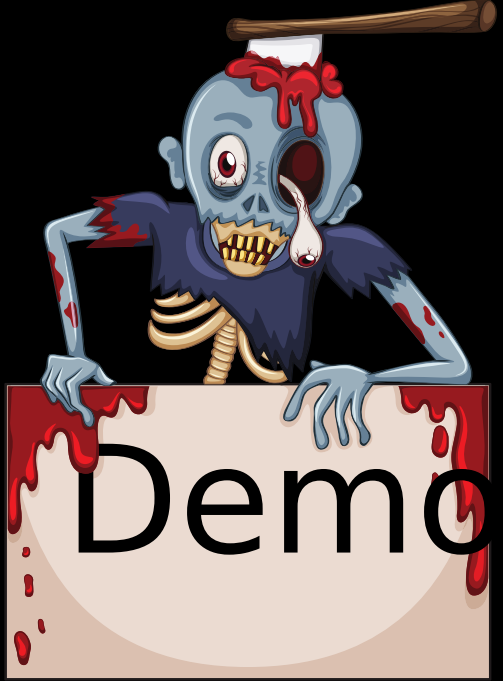


IMPOSSIBLE


key$_n$ (0xD2)
1 1 0 1 0 0 1 0

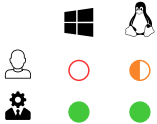

key$_n$ (0xD2)
(4,4)-domino$_{n,n+1}$ (0x21)
1 1 0 1 0 0 1 0 0 0 0 1


key$_n$ (0xD2)
(4,4)-domino$_{n,n+1}$ (0x21)
key$_{n+1}$ (0x1C)
1 1 0 1 0 0 1 0 0 0 0 1 1 1 0 0


Demo

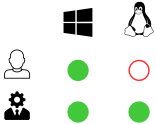
Variant 1

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

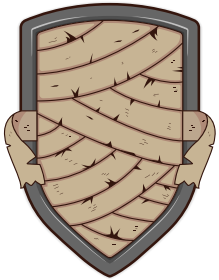
Kernel Mapping

5.30 kB/s

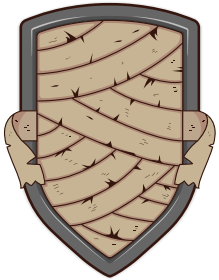
Variant 3

Microcode-Assisted
Page-Table Walk

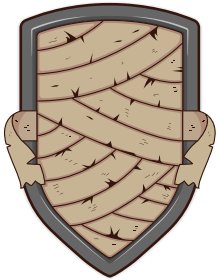
7.73 kB/s



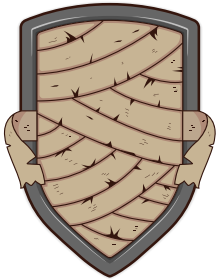
- Disable hyperthreading or group scheduling



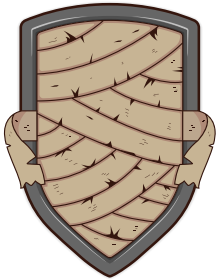
- Disable hyperthreading or group scheduling
- Overwrite microarchitectural buffers



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



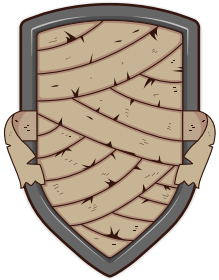
- Disable hyperthreading or group scheduling
- Overwrite microarchitectural buffers
  - `VERW` instruction (microcode update)
  - Software sequences



- 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	✗	✗	✗	✗	✗	✗

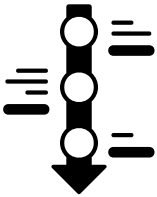
`https://www.intel.com/content/www/us/en/architecture-and-technology/engineering-new-protections-into-hardware.html`



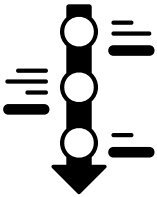
- Disable hyperthreading or group scheduling
- Overwrite microarchitectural buffers
  - `VERW` instruction (microcode update)
  - Software sequences
- New CPUs which are not affected

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

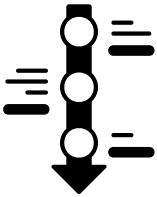



2017



2017

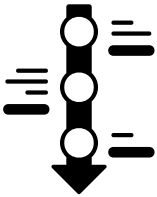
May 4 — KAISER patch published



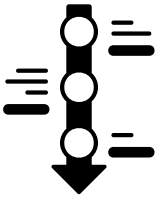
2017

May 4 — KAISER patch published

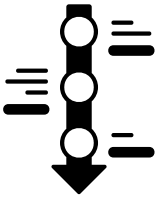
June 22 — Jann Horn reports Meltdown



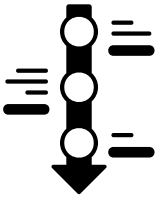
2017	
May 4	KAISER patch published
June 22	Jann Horn reports Meltdown
December 12	We report Meltdown



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)

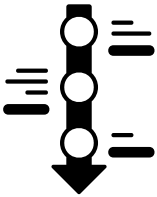


2017	
May 4	KAISER patch published
June 22	Jann Horn reports Meltdown
December 12	We report Meltdown
2018	
February 15	We report Meltdown-UC (MDSUM)
March 28	We send a cleaned up Meltdown-UC PoC (MDSUM)

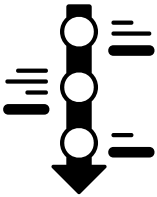


2017	
May 4	KAISER patch published
June 22	Jann Horn reports Meltdown
December 12	We report Meltdown
2018	
February 15	We report Meltdown-UC (MDSUM)
March 28	We send a cleaned up Meltdown-UC PoC (MDSUM)
May 30	We report LFB leakage with Meltdown-UC

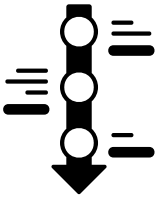
Timeline



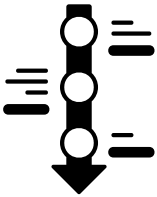
2017	
May 4	KAISER patch published
June 22	Jann Horn reports Meltdown
December 12	We report Meltdown
2018	
February 15	We report Meltdown-UC (MDSUM)
March 28	We send a cleaned up Meltdown-UC PoC (MDSUM)
May 30	We report LFB leakage with Meltdown-UC
2019	



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



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



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


THIS IS FINE.


THIS IS FINE.


THIS IS FINE.


RISE AGAIN

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

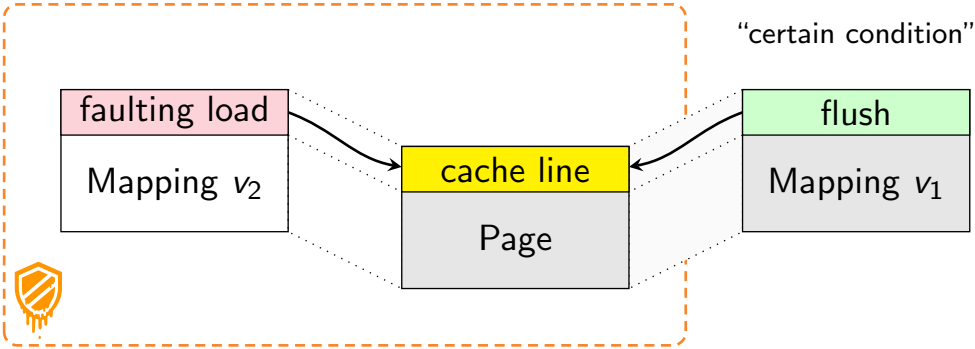
Which ZombieLoad Variant works despite MDS mitigations?

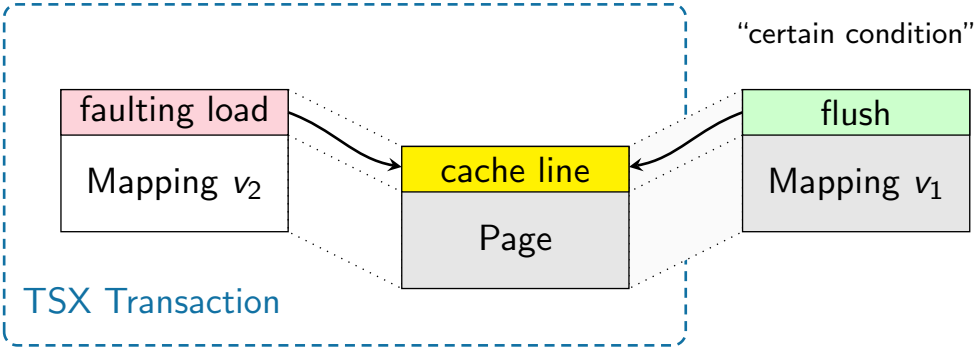
A: Variant 1

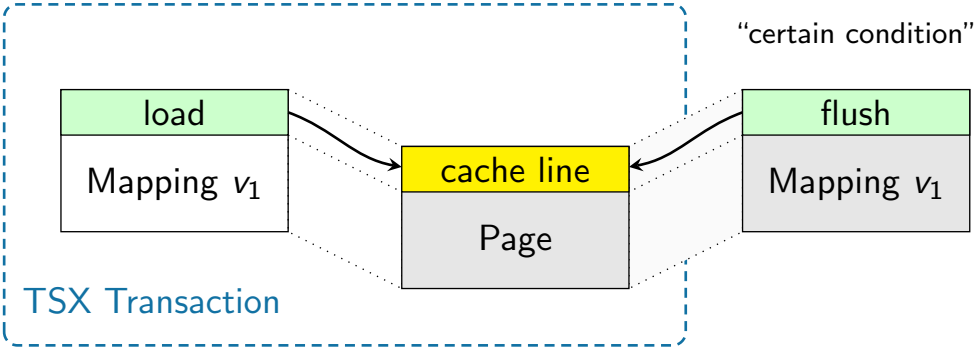
B: Variant 3

C: Variant 2

D: None


"certain condition"
faulting load
Mapping v_2
cache line
Page
flush
Mapping v_1


"certain condition"
faulting load
Mapping v_2
cache line
Page
flush
Mapping v_1
TSX Transaction


"certain condition"
load
Mapping v_1
cache line
Page
flush
Mapping v_1
TSX Transaction


// Variant 2
flush(mapping);
if (xbegin() == _XBEGIN_STARTED) {
maccess(lut + 4096 * mapping[0]);
xend();
}

- Data Conflicts

- Data Conflicts
- Limited Transactional Resources

- Data Conflicts
- Limited Transactional Resources

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

- Data Conflicts
- Limited Transactional Resources

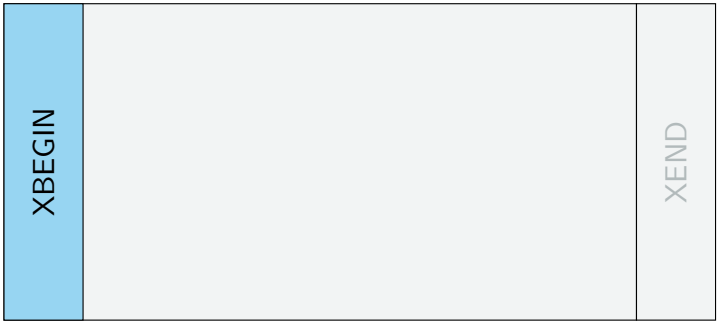
- Certain Instructions
  - IO instructions, syscall, ...
- Synchronous Exception Events
  - #BR, #PF, #DB, #BP/INT3, ...

12.2.4.5 Miscellaneous Transactional Aborts

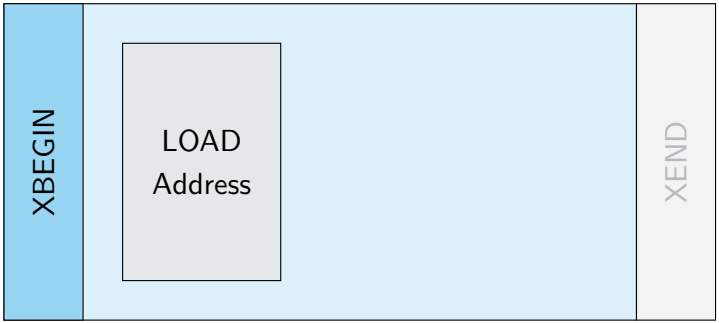
Asynchronous events (NMI, SMI, INTR, IPI, PMI, etc.) occurring during transactional execution may cause the transactional execution to abort and transition to a non-transactional execution.

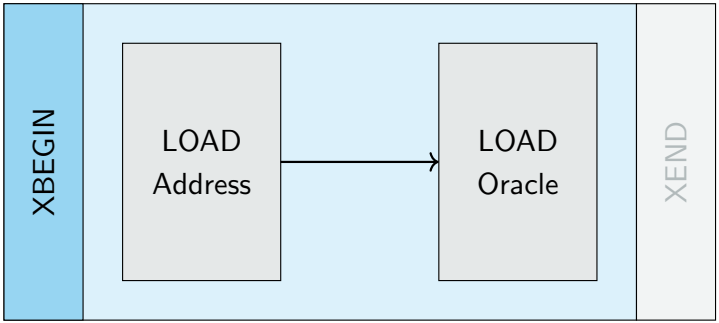
12.2.4.5 Miscellaneous Transactional Aborts

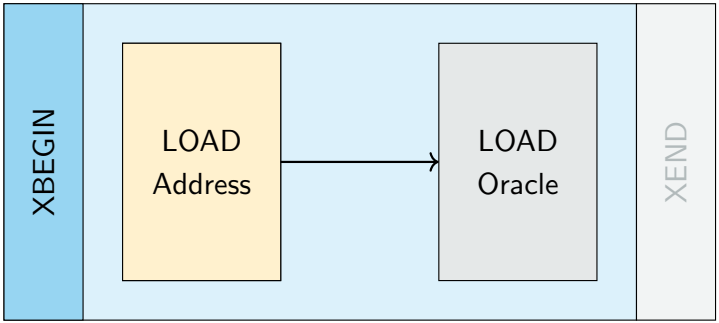
Asynchronous events (NMI, SMI, INTR, IPI, PMI, etc.) occurring during transactional execution may cause the transactional execution to abort and transition to a non-transactional execution. [...] For example, operating systems with timer ticks generate interrupts that can cause transactional aborts.

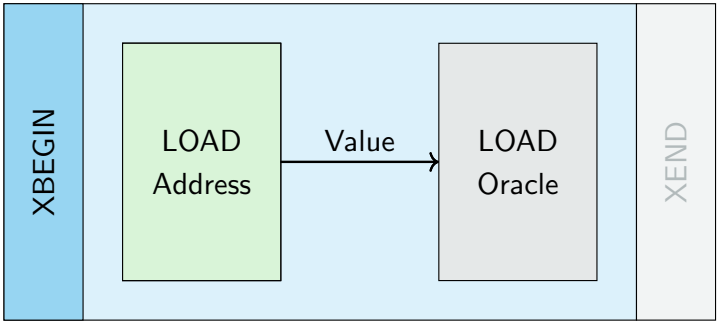

XBEGIN
XEND

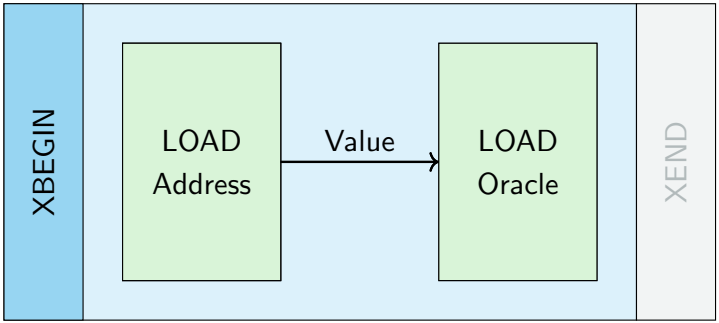

XBEGIN
XEND

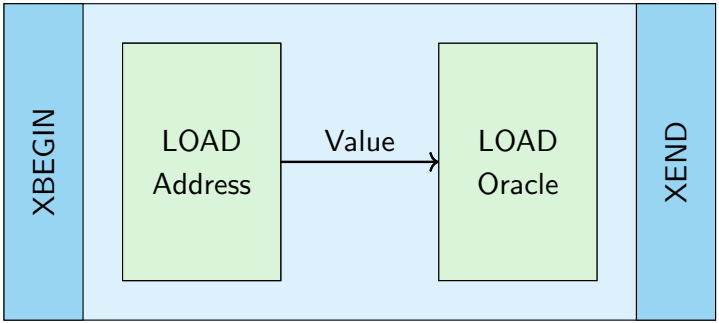

XBEGIN
LOAD
Address
XEND


XBEGIN
LOAD
Address
LOAD
Oracle
XEND

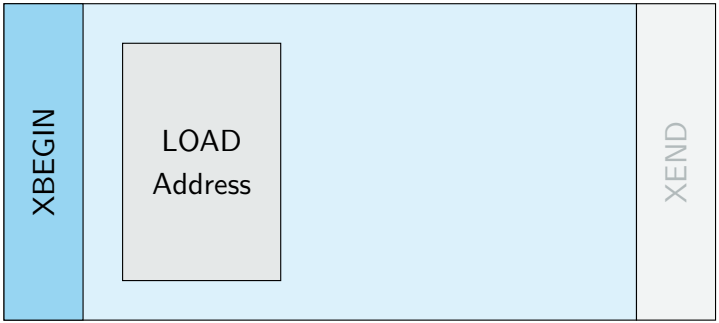

XBEGIN
LOAD
Address
LOAD
Oracle
XEND

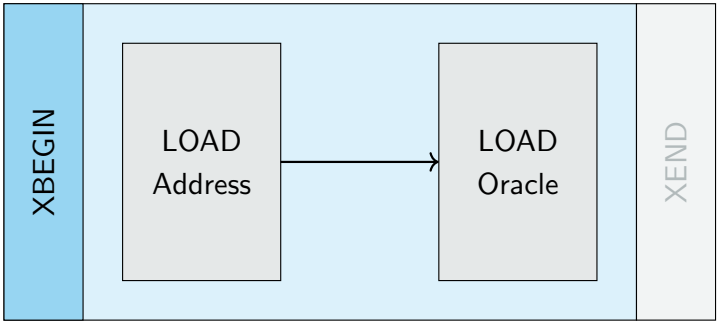

XBEGIN
LOAD
Address
Value
LOAD
Oracle
XEND

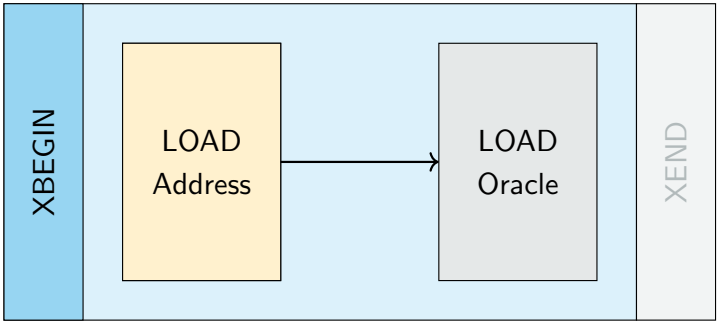

XBEGIN
LOAD
Address
Value
LOAD
Oracle
XEND

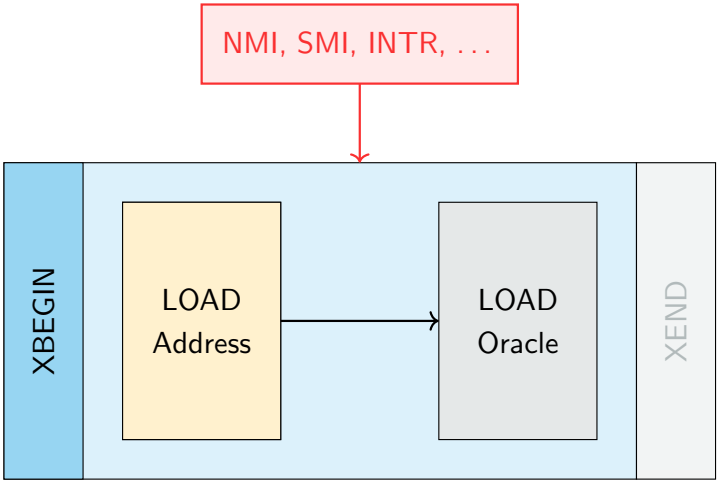

XBEGIN
LOAD
Address
Value
LOAD
Oracle
XEND

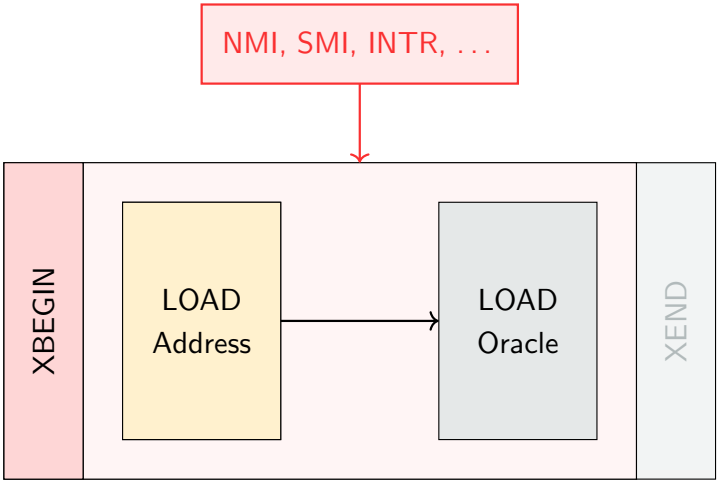

XBEGIN
XEND

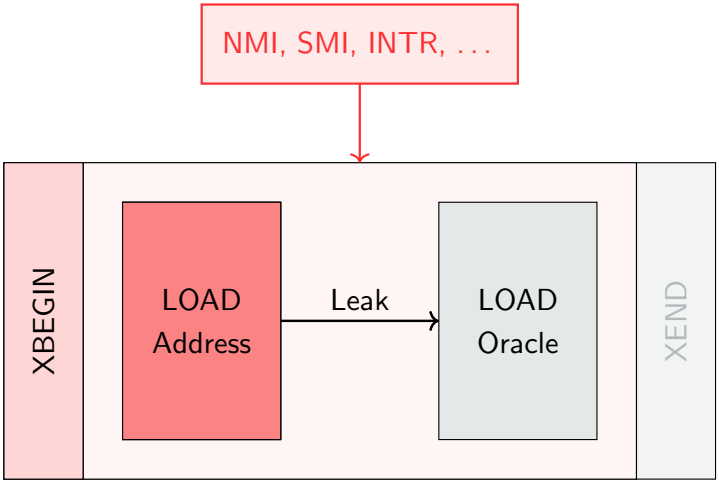

XBEGIN
LOAD
Address
XEND

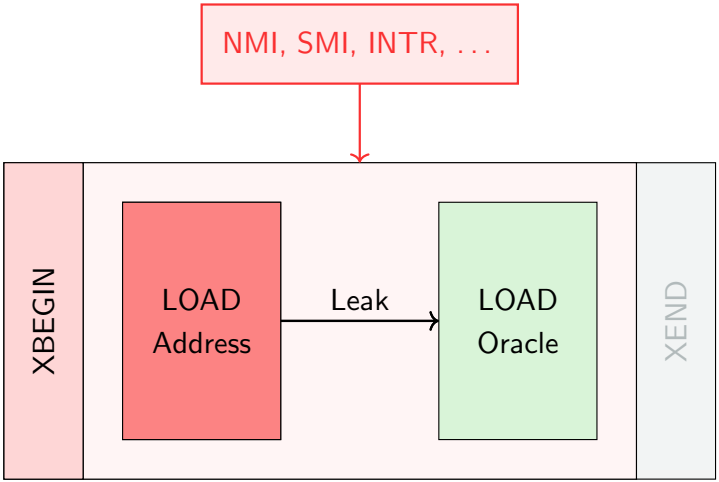

XBEGIN
LOAD
Address
LOAD
Oracle
XEND

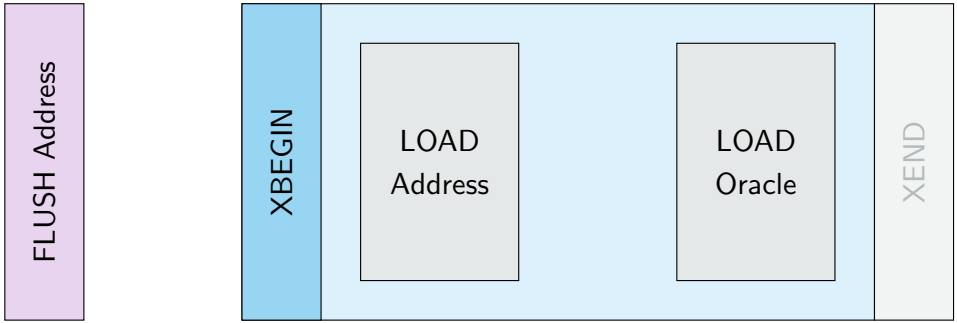

XBEGIN
LOAD
Address
LOAD
Oracle
XEND

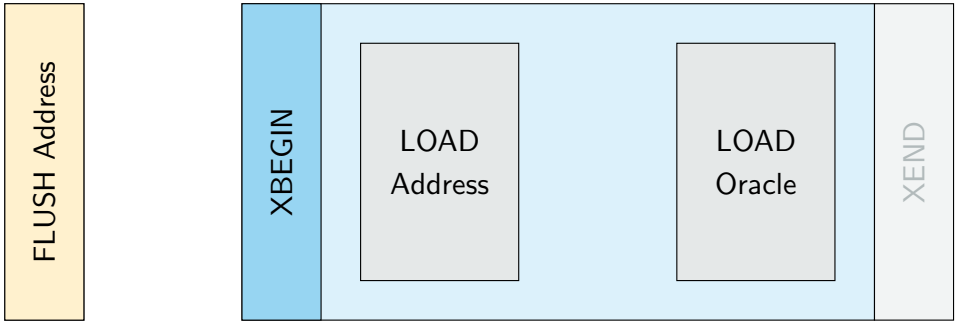

NMI, SMI, INTR, ...
XBEGIN
LOAD
Address
LOAD
Oracle
XEND

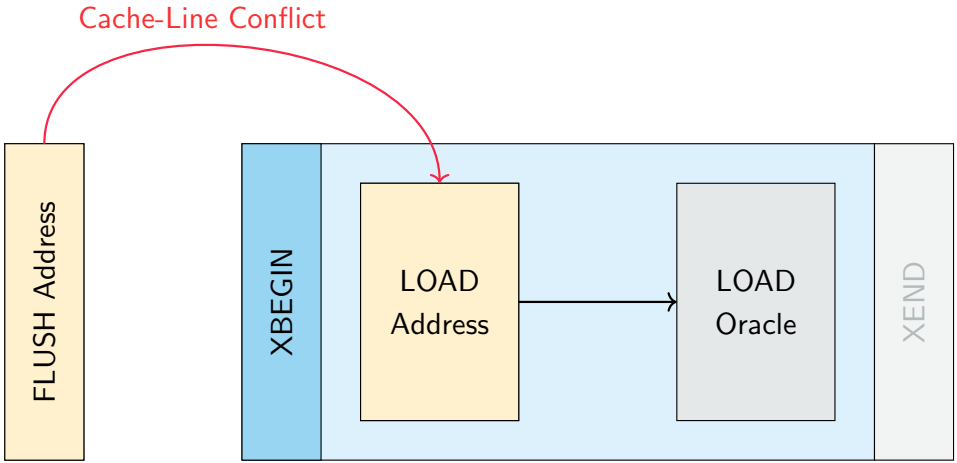

NMI, SMI, INTR, ...
XBEGIN
LOAD
Address
LOAD
Oracle
XEND

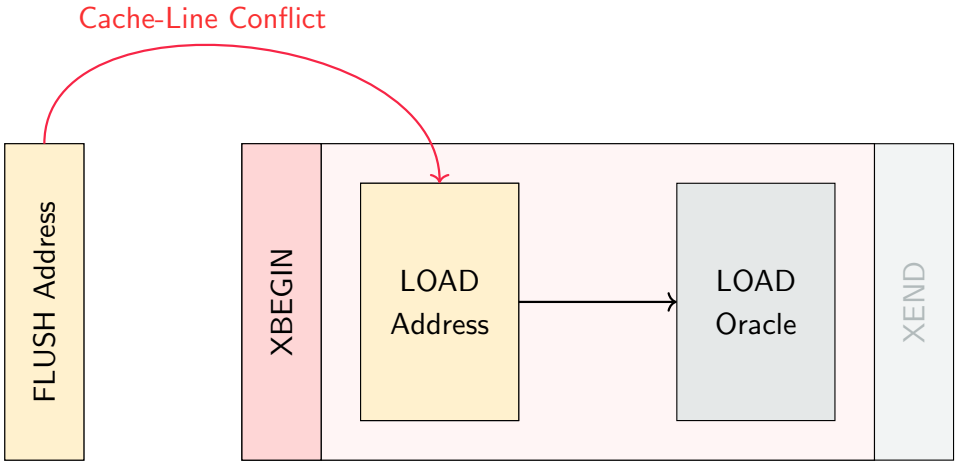

NMI, SMI, INTR, ...
XBEGIN
LOAD
Address
Leak
LOAD
Oracle
XEND

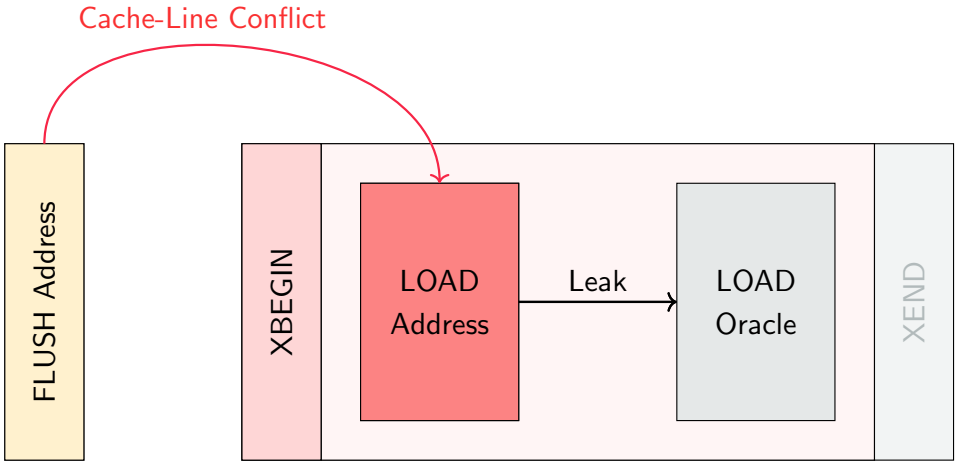

NMI, SMI, INTR, ...
XBEGIN
LOAD
Address
Leak
LOAD
Oracle
XEND

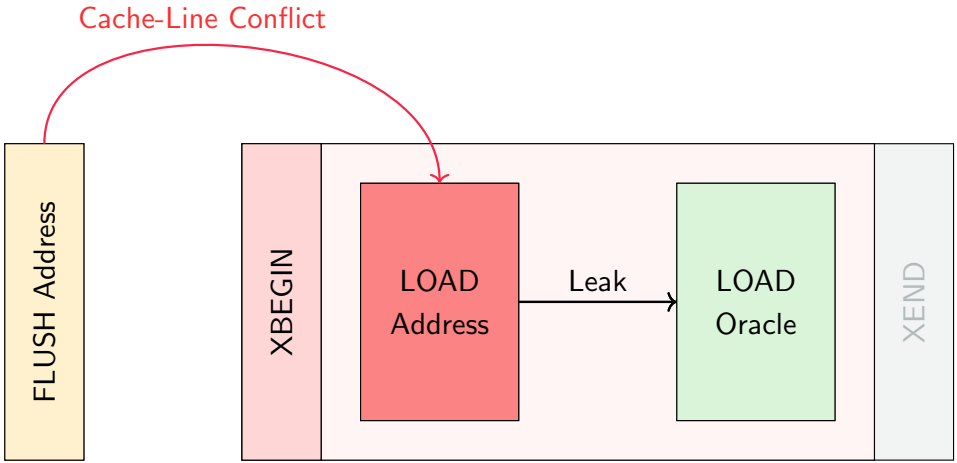

FLUSH Address
XBEGIN
LOAD
Address
LOAD
Oracle
XEND


FLUSH Address
XBEGIN
LOAD
Address
LOAD
Oracle
XEND


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

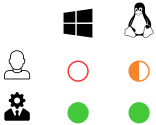

Cache-Line Conflict
FLUSH Address
XBEGIN
LOAD
Address
LOAD
Oracle
XEND


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


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

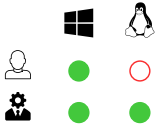
Variant 1

Kernel Mapping



Variant 3

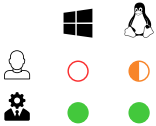
Microcode-Assisted Page-Table Walk



● works ○ does not work ◐ can be prevented

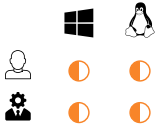
Variant 1

Kernel Mapping



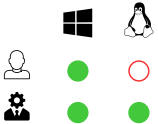
Variant 2

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

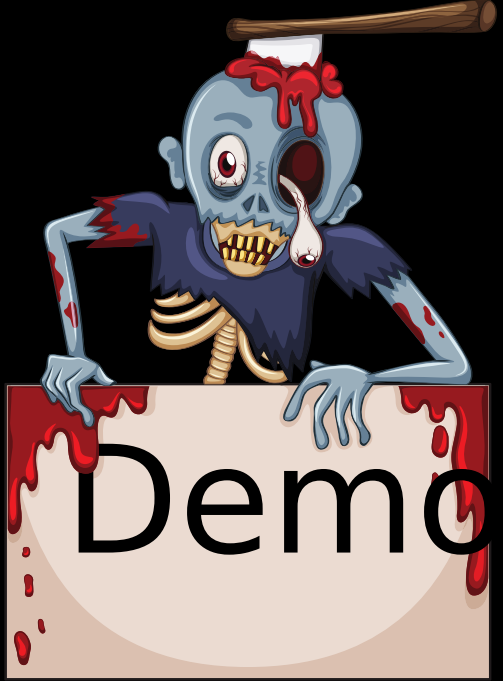
Transactional Asynchronous Abort

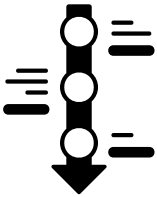
39.66 kB/s

Variant 3

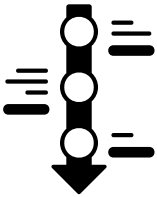
Microcode-Assisted Page-Table Walk

7.73 kB/s


Demo

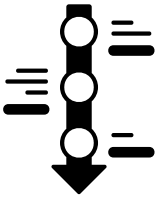


2019



2019

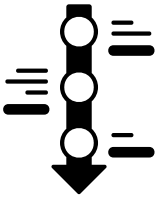
April 12 — We report ZombieLoad



2019

April 12 — We report ZombieLoad

April 24 — Report Variant 2

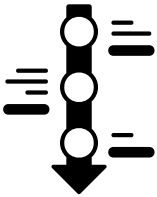


2019

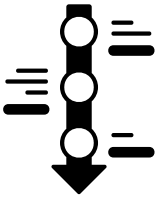
April 12 — We report ZombieLoad

April 24 — Report Variant 2

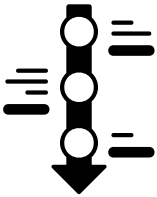
May 10 — Report TAA on Cascade Lake



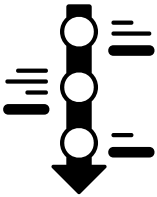
2019	
April 12	We report ZombieLoad
April 24	Report Variant 2
May 10	Report TAA on Cascade Lake
May 11	Call with Intel + Embargo



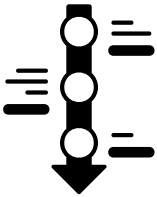
2019	
April 12	We report ZombieLoad
April 24	Report Variant 2
May 10	Report TAA on Cascade Lake
May 11	Call with Intel + Embargo
May 14	Disclosure of ZombieLoad (without Variant 2)



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

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

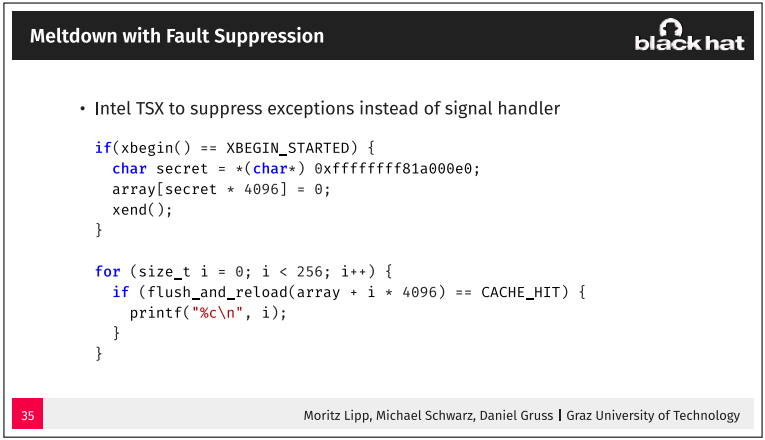
5:45 nachm. · 24. Dez. 2019 · Twitter Web App

23 „Gefällt mir"-Angaben


WAIT A MINUTE...
TAA IS TSX OVER A LEAK GADGET?
imgflip.com

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

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



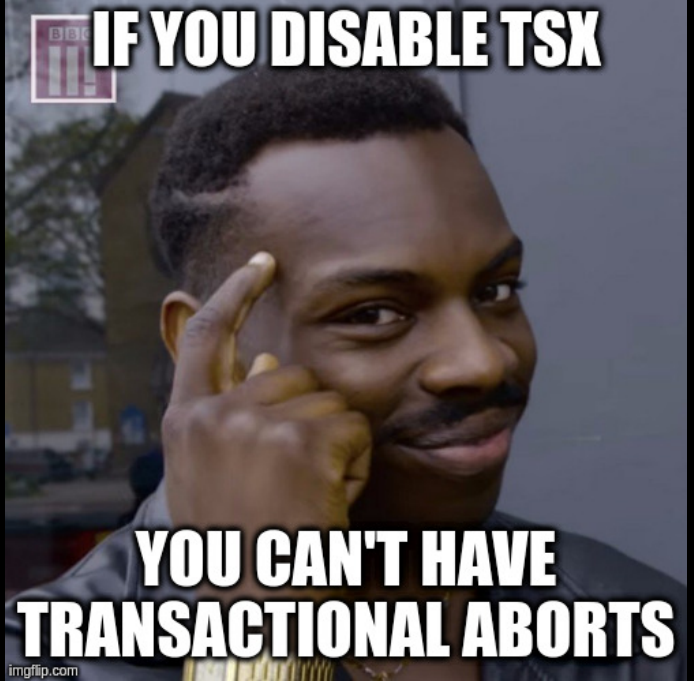
Meltdown with Fault Suppression

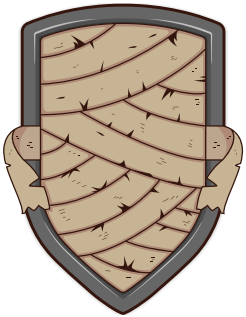
black hat

- Intel TSX to suppress exceptions instead of signal handler

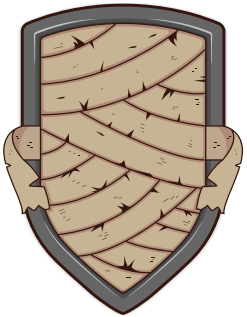
```
if(xbegin() == XBEGIN_STARTED) {
  char secret = *(char*) 0xffffffff81a000e0;
  array[secret * 4096] = 0;
  xend();
}

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

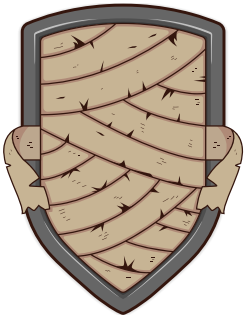

IF YOU DISABLE TSX
YOU CAN'T HAVE
TRANSACTIONAL ABORTS
imgflip.com



- Disable Intel TSX



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



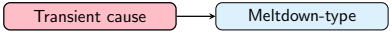
- Disable Intel TSX
  - Deactivated by default with new microcode updates on CPUS enumerating `MDS_NO`
- VERW to overwrite affected buffers

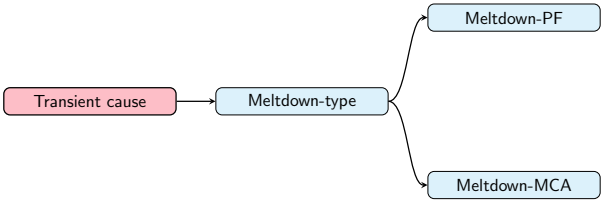
- Software-sequences and VERW do not work reliably

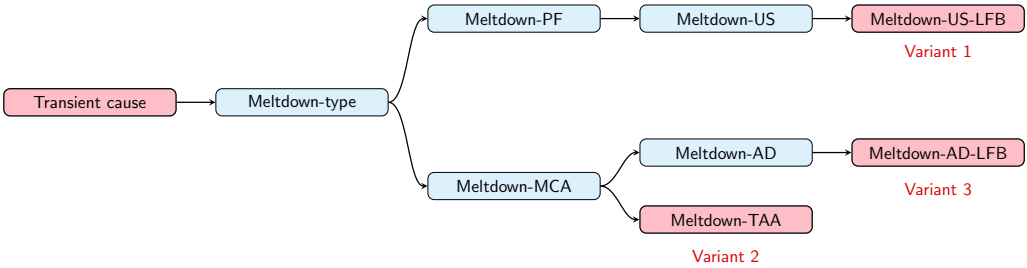
- Software-sequences and VERW do not work reliably
  - Cases where leakage is still visible

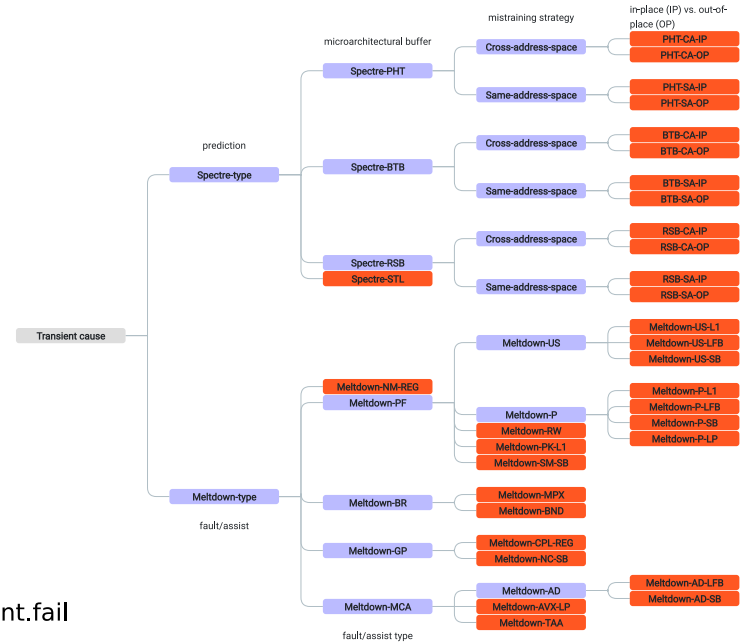

Insights


Transient cause


Transient cause
Meltdown-type

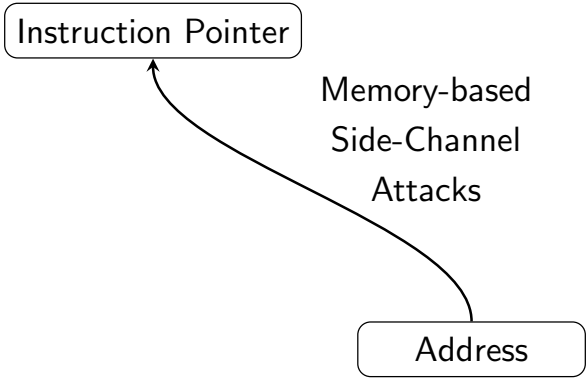

Transient cause
Meltdown-type
Meltdown-PF
Meltdown-MCA

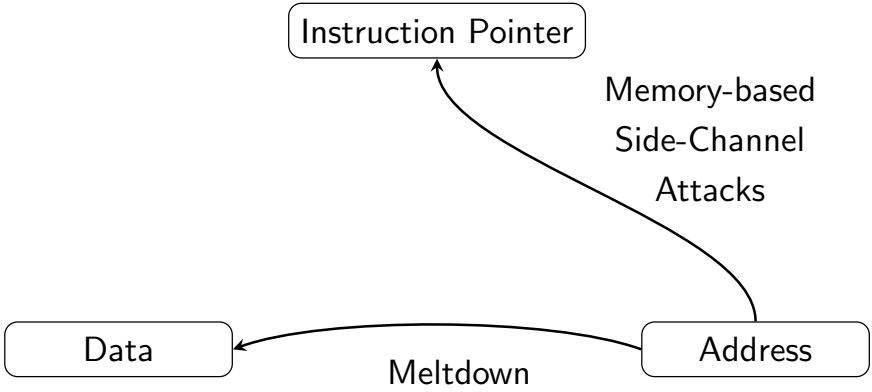

Transient cause
Meltdown-type
Meltdown-PF
Meltdown-US
Meltdown-US-LFB
Variant 1
Meltdown-MCA
Meltdown-AD
Meltdown-AD-LFB
Variant 3
Meltdown-TAA
Variant 2

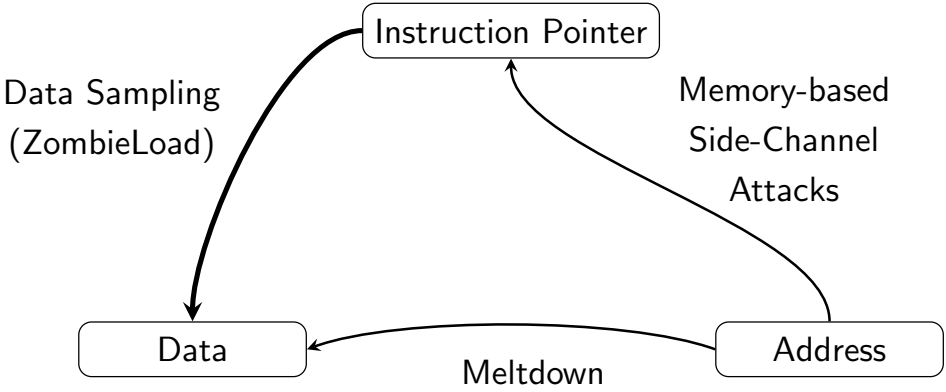


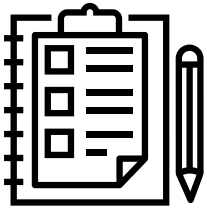
https://transient.fail

Address


Instruction Pointer
Memory-based
Side-Channel
Attacks
Address


Instruction Pointer
Memory-based
Side-Channel
Attacks
Data
Meltdown
Address


Instruction Pointer
Data Sampling
(ZombieLoad)
Memory-based
Side-Channel
Attacks
Data
Address
Meltdown



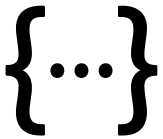
- Spectre is here to stay
- More Meltdown-type than Spectre-type attacks

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

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