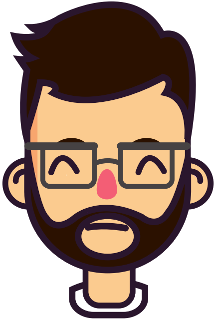
Ghosts in a Nutshell

Moritz Lipp (@mlqxyz)
Claudio Canella (@cc0x1f)

Who am I?



Moritz Lipp

PhD student @ Graz University of Technology

@mlqxyz

moritz.lipp@iaik.tugraz.at

Claudio Canella

PhD student @ Graz University of Technology

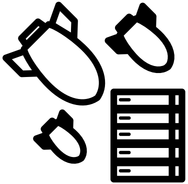
@cc0x1f

claudio.canella@iaik.tugraz.at


FIREEYE CYBER
THREAT MAP
DEVELOPING STORY
COMPUTER CHIP FLAWS IMPACT BILLIONS OF DEVICES
LIVE
CNN
DAX ▲164.69
NEWS STREAM

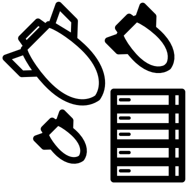

GLOBAL
COMPUTER CHIP SCARE
The bugs are known as 'Spectre' and 'Meltdown'
BBC WORLD NEWS
£:HKS 10.58
EURO:£ 0.891

- Side-Channel Vulnerability Variant 1, 2, 3
    - and Variant 3a
- Meltdown and Spectre have been disclosed



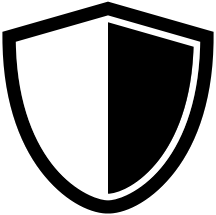
- Variant 1 - Bounds Check Bypass (BCB)
- Variant 2 - Branch Target Injection (BTI)
- Variant 3 - Rogue Data Cache Load (RDCL)
- Variant 3a - Rogue System Register Read (RSRR)
- Variant 4 - Speculative Store Bypass (SSB)
- Variant 1.1 - Bounds Check Bypass Store (BCBS)
- Variant 1.2 - Read-only protection bypass (RPB)
- Lazy FP State Restore

Motivation

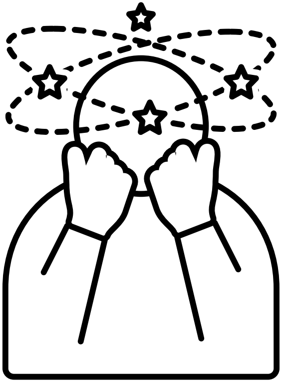


- SpectreRSB - Return Mispredict
- Foreshadow
- L1 Terminal Fault (L1TF)
- Portsmash
- Netspectre
- SMoTherSpectre
- SPOILER

Motivation

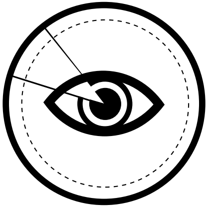


- KAISER patch / KPTI / KVA Shadow
- Microcode Updates
- IBRS / STIPB / IBPB
- Retpoline
- Taint Tracking
- Serialization
- InvisiSpec / SafeSpec / DAWG
- RSB Stuffing
- Site Isolation
- SSBD / SSBB
- ...



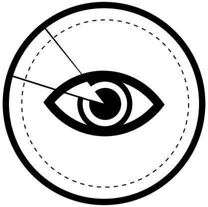
Now you already lost me . . .

What this talk is

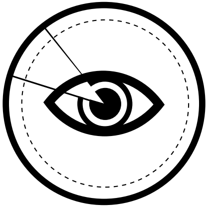


- We want to shed some light and make it less confusing

What this talk is



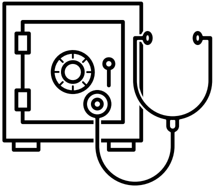
- We want to shed some light and make it less confusing
- Give a comprehensible overview of all attacks and defenses

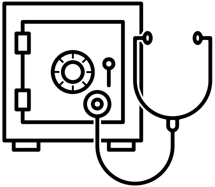


- We want to shed some light and make it less confusing
- Give a comprehensible overview of all attacks and defenses
- Show that systematic analysis allows to find new attacks and circumventions of countermeasures

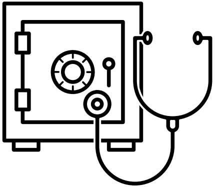
Background

- Bug-free software does not mean safe execution

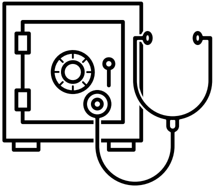




- Bug-free software does not mean safe execution
- Information leaks due to underlying hardware

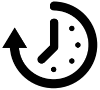


- Bug-free software does not mean safe execution
- Information leaks due to underlying hardware
- Exploit leakage through side-effects



- Bug-free software does not mean safe execution
- Information leaks due to underlying hardware
- Exploit leakage through side-effects

Power consumption

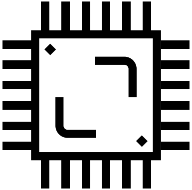


Execution time

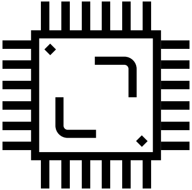
Microarchitectural elements

...

Architecture vs Microarchitecture

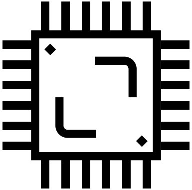


- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)



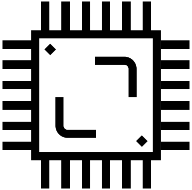
- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)
- Interface between hardware and software

Architecture vs Microarchitecture



- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, ...)
- Interface between hardware and software
- Microarchitecture is an ISA implementation

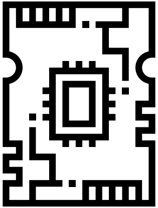
Architecture vs Microarchitecture



- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)
- Interface between hardware and software
- Microarchitecture is an ISA implementation

Moritz Lipp (@mlqxyz), Claudio Canella (@cc0x1f)

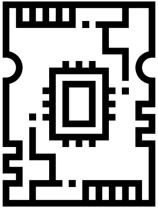
Microarchitectural Elements



- Modern CPUs contain multiple microarchitectural elements

Microarchitectural Elements

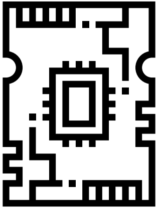
- Modern CPUs contain multiple microarchitectural elements



Caches and buffer

Predictor

...



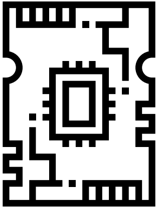
- Modern CPUs contain multiple microarchitectural elements

Caches and buffer

Predictor

...

- Transparent for the programmer



- Modern CPUs contain multiple microarchitectural elements

Caches and buffer

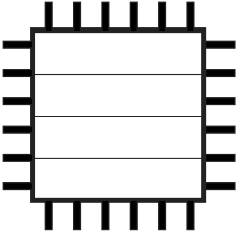
Predictor

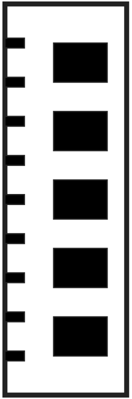
•••

- Transparent for the programmer
- Timing optimizations → side-channel leakage

Caches & Cache Attacks

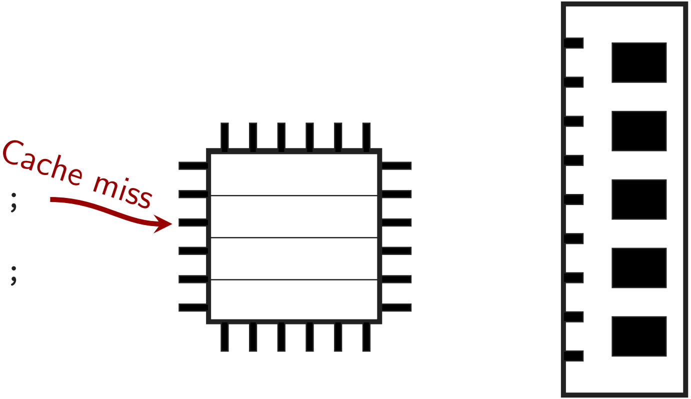
```
printf("%d", i);

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



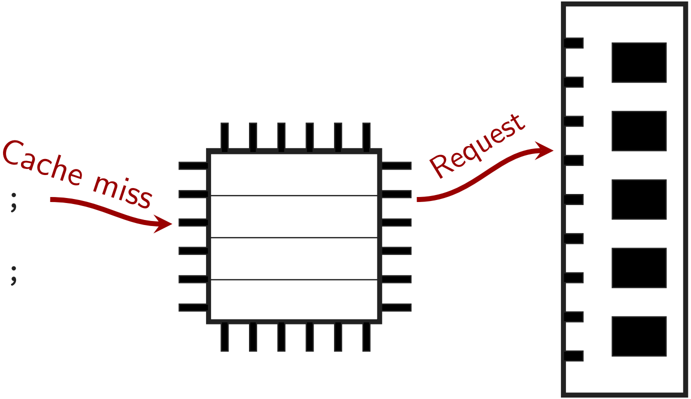


Cache

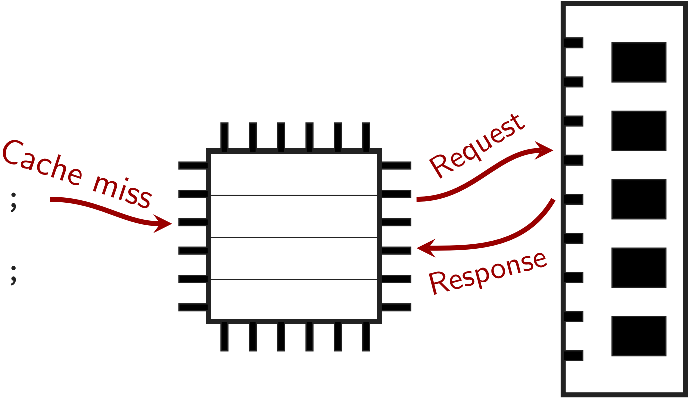
```
printf("%d", i);

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



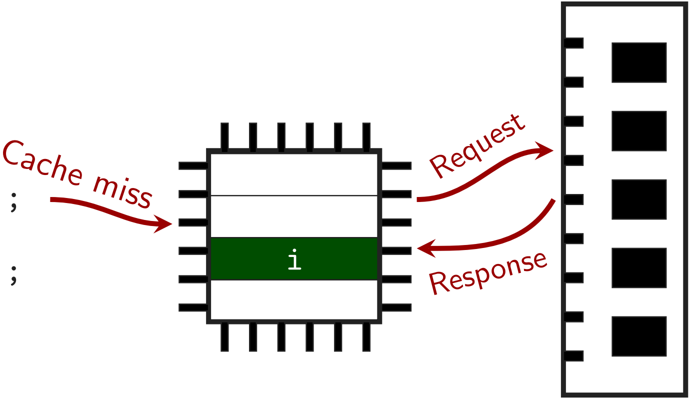
```
printf("%d", i);

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

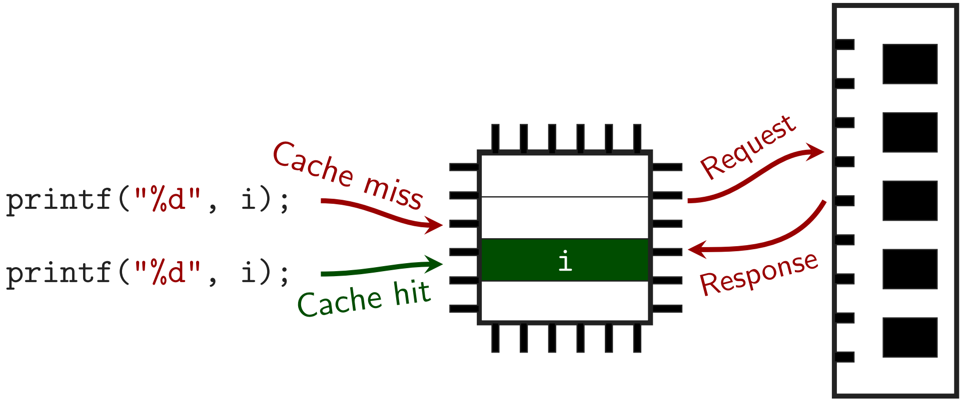


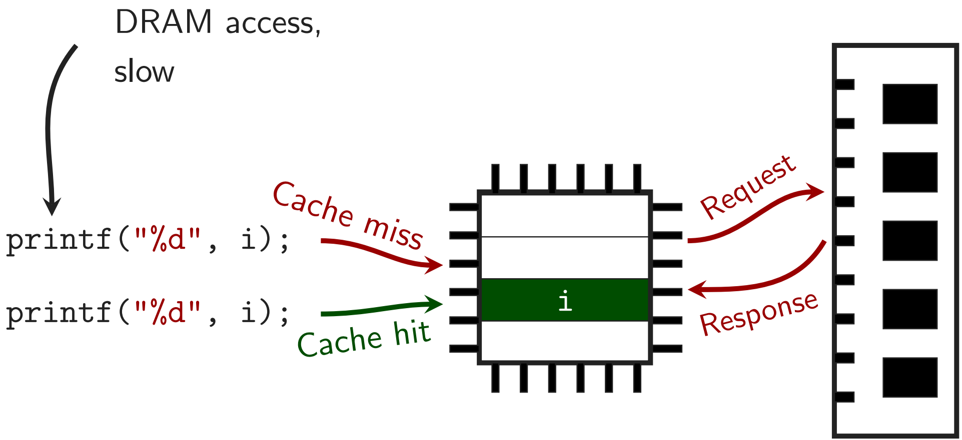
```
printf("%d", i);

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

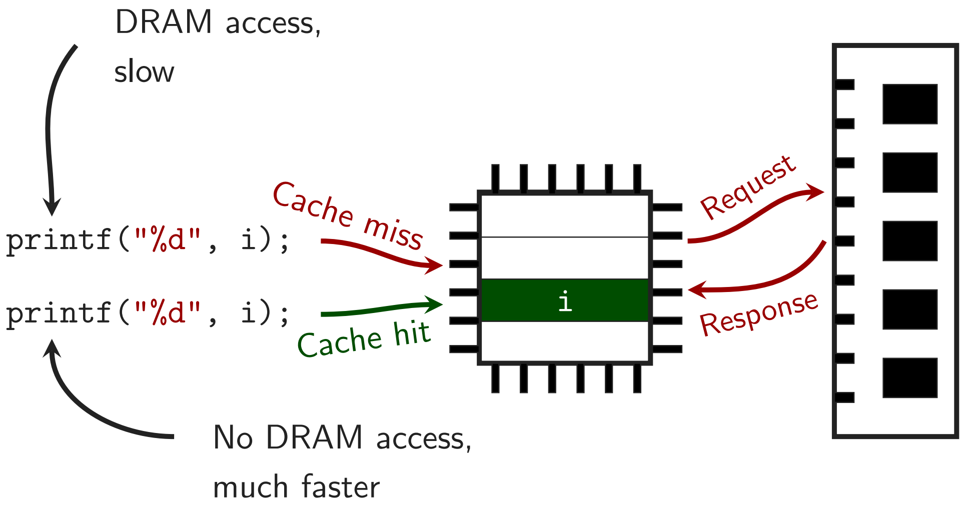


```
printf("%d", i);

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

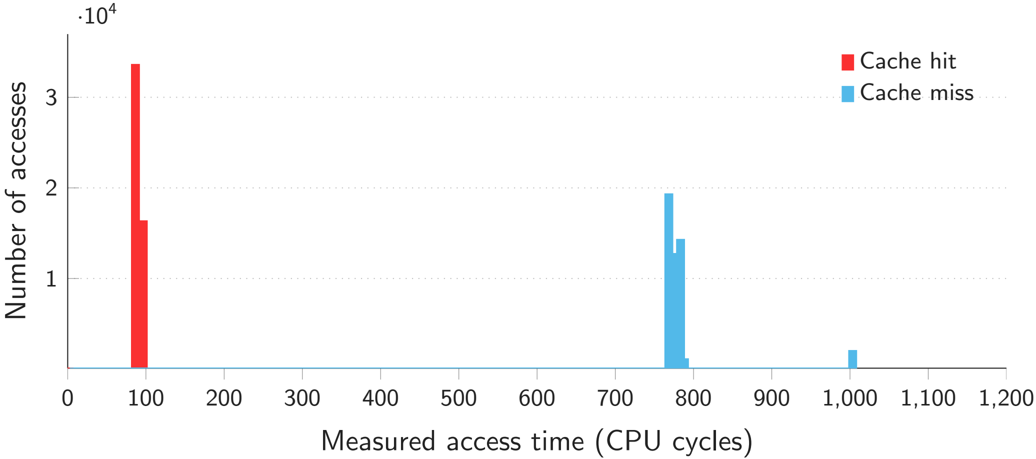


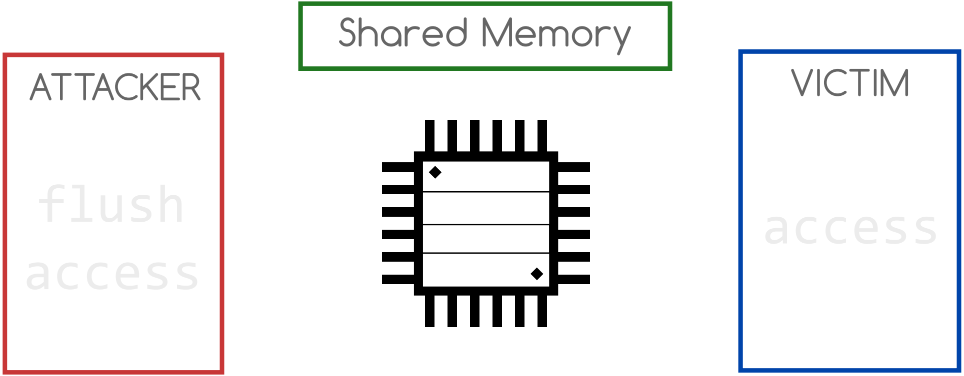

printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response

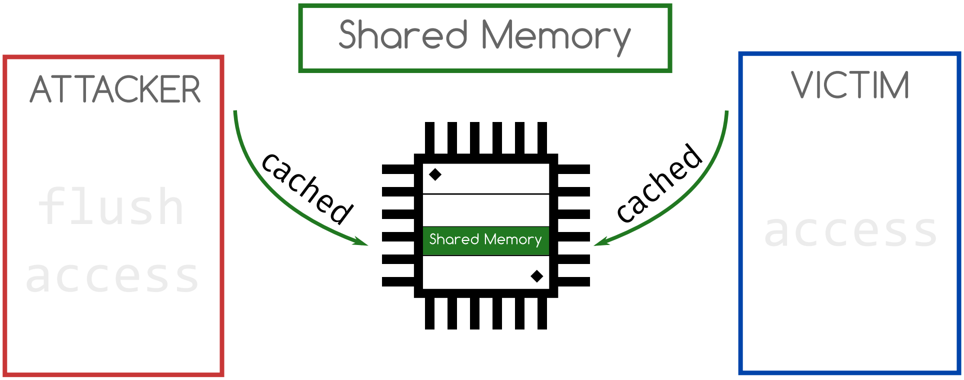

DRAM access,
slow
printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response

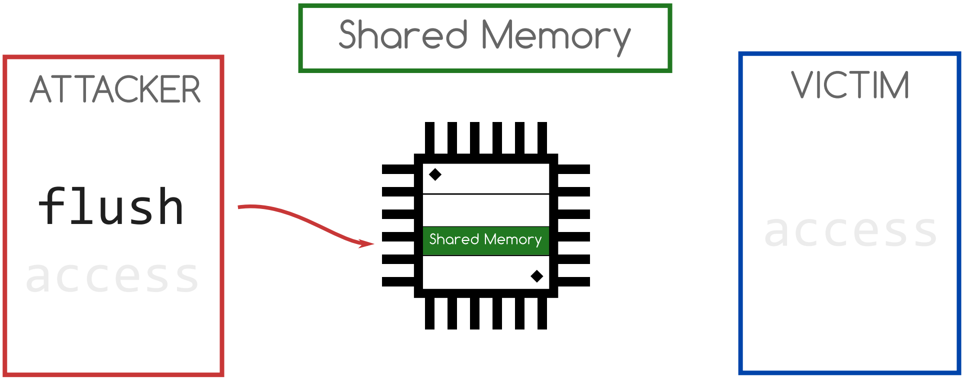

DRAM access,
slow
printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response
No DRAM access,
much faster

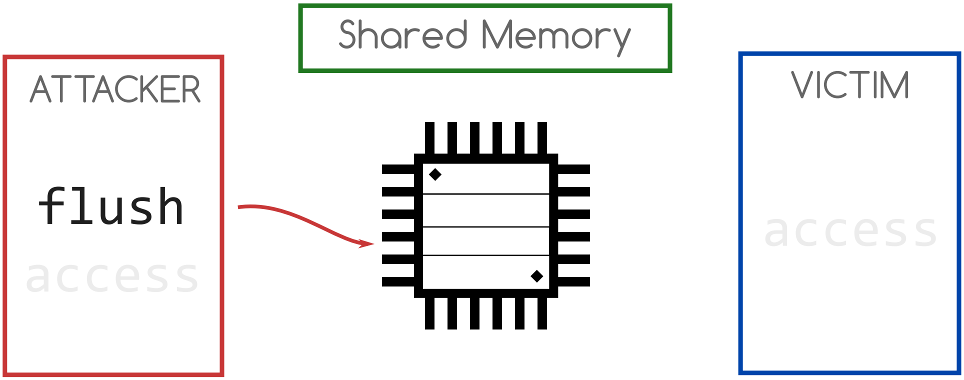
Caching speeds up Memory Accesses

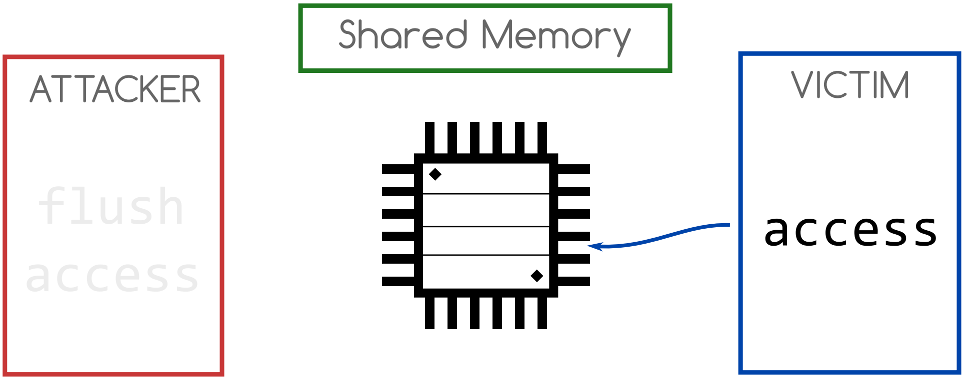


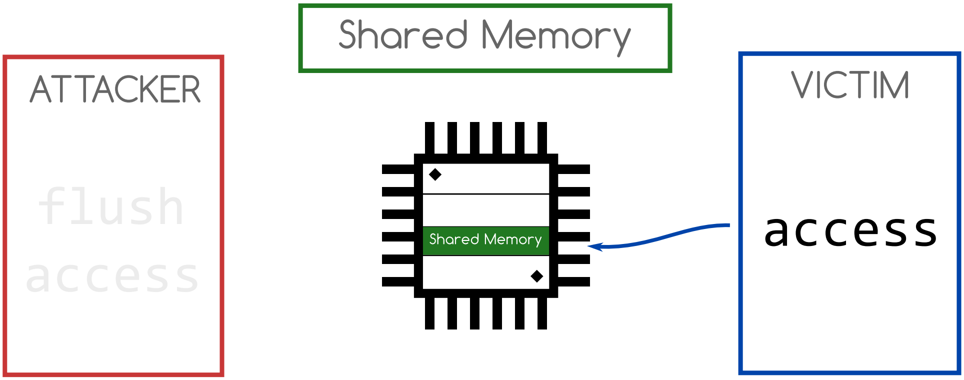

Shared Memory
ATTACKER
flush
access
VICTIM
access


Shared Memory
ATTACKER
flush
access
cached
Shared Memory
cached
VICTIM
access

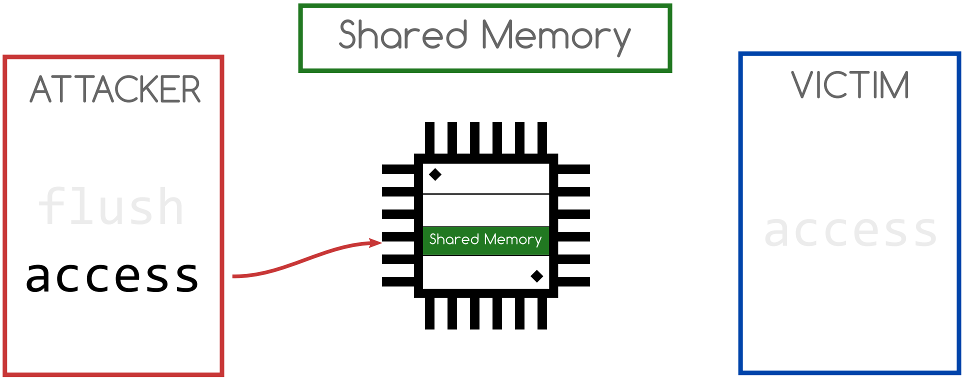

Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access

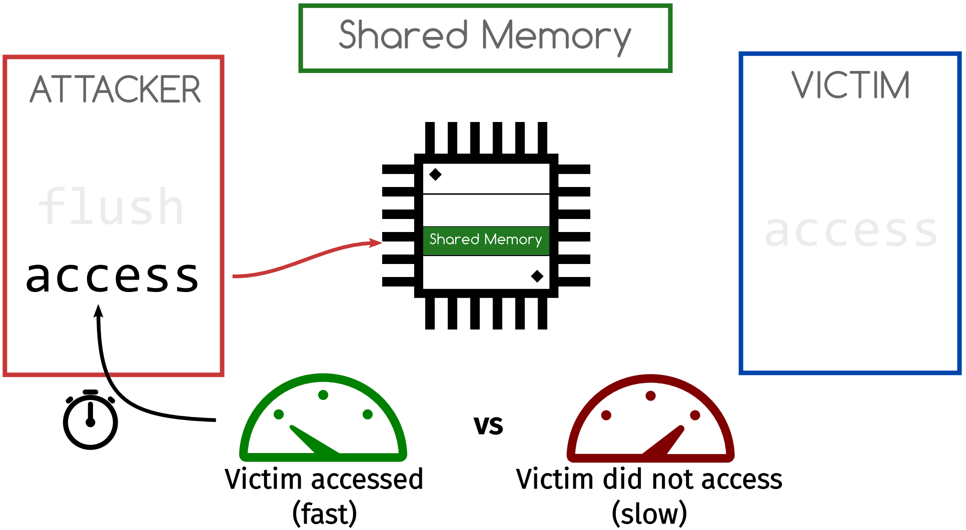

Shared Memory
ATTACKER
flush
access
VICTIM
access


Shared Memory
ATTACKER
flush
access
VICTIM
access

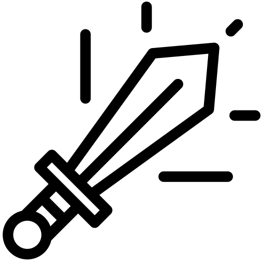

Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access

Flush+Reload




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

- Leak cryptographic keys
- Leak information on co-located virtual machines
- Monitor function calls of other applications
- Build covert communication channels
- ...



- Leak cryptographic keys
- Leak information on co-located virtual machines
- Monitor function calls of other applications
- Build covert communication channels
- ...

Only meta data. Not interesting and not in any threat model.

Out-of-order execution

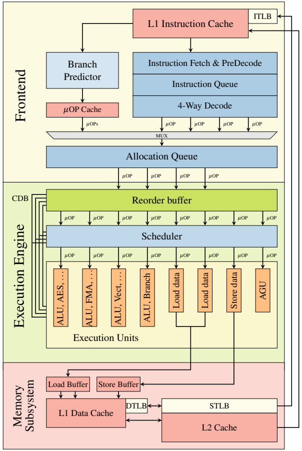
Out-of-order Execution

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

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

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

Parallelize

Dependency

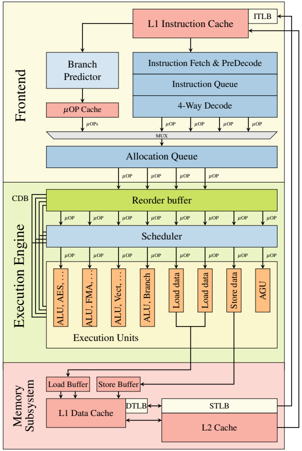
```
int width = 10, height = 5;

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

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



Instructions are

- fetched and decoded in the front-end

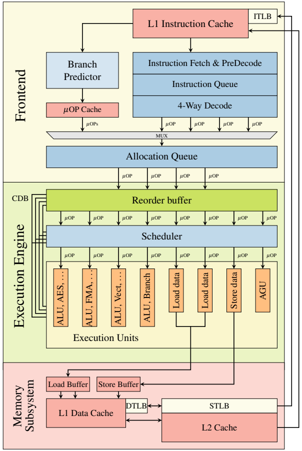
Out-of-Order Execution



Instructions are

- fetched and decoded in the front-end
- dispatched to the backend

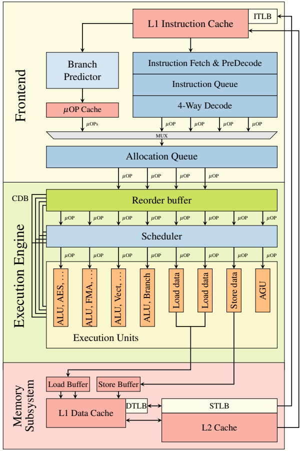
Out-of-Order Execution



Instructions are

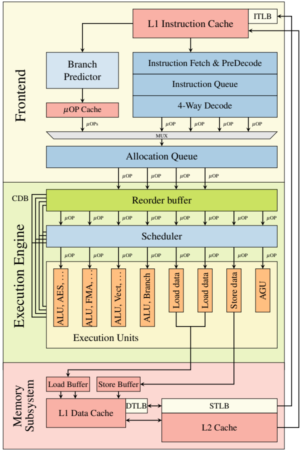
- fetched and decoded in the front-end
- dispatched to the backend
- processed by individual execution units

Out-of-Order Execution



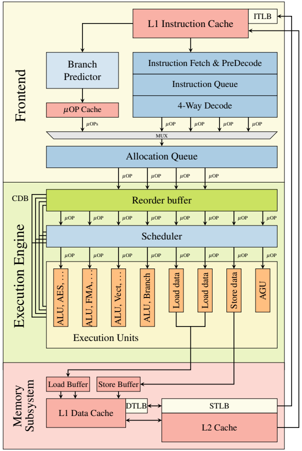
Instructions

- are executed out-of-order



Instructions

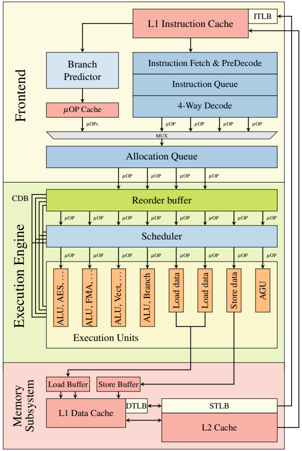
- are executed out-of-order
- wait until their dependencies are ready



Instructions

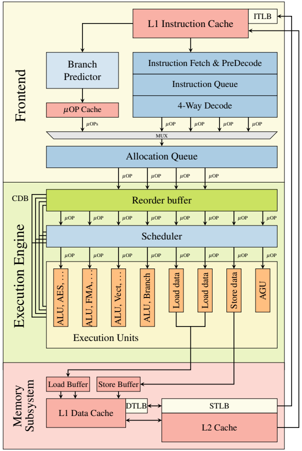
- are executed out-of-order
- wait until their dependencies are ready
  - Later instructions might execute prior earlier instructions

Out-of-Order Execution



Instructions

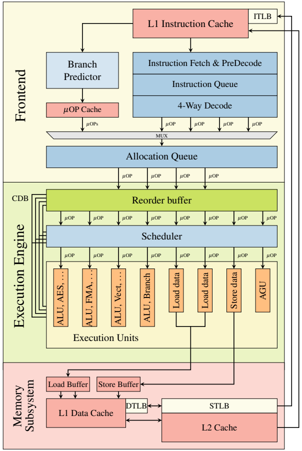
- are executed out-of-order
- wait until their dependencies are ready
  - Later instructions might execute prior earlier instructions
- retire in-order



Instructions

- are executed out-of-order
- wait until their dependencies are ready
  - Later instructions might execute prior earlier instructions
- retire in-order
  - State becomes architecturally visible

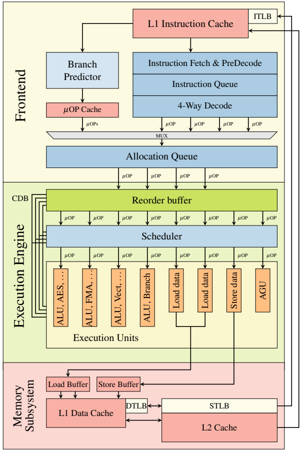
Out-of-Order Execution



Instructions

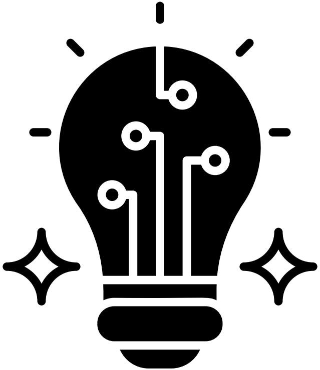
- are executed out-of-order
- wait until their dependencies are ready
  - Later instructions might execute prior earlier instructions
- retire in-order
  - State becomes architecturally visible
- Exceptions are checked during retirement

Out-of-Order Execution

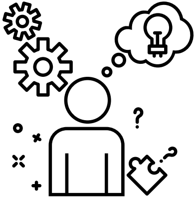


Instructions

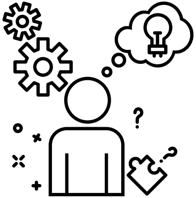
- are executed out-of-order
- wait until their dependencies are ready
  - Later instructions might execute prior earlier instructions
- retire in-order
  - State becomes architecturally visible
- Exceptions are checked during retirement
  - Flush pipeline and recover state



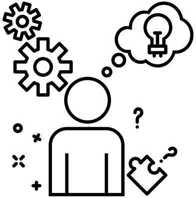
The state does not become architecturally visible but . . .



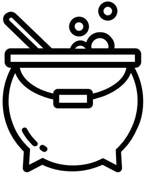
- What happens to the instructions that are dismissed?



- What happens to the instructions that are dismissed?
- Do they leave any side effects?

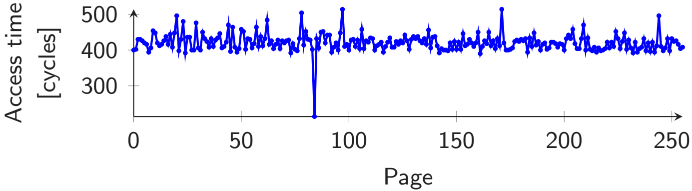


- What happens to the instructions that are dismissed?
- Do they leave any side effects?
- Can we observe them?

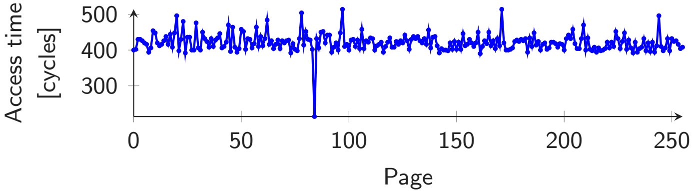


```
*(volatile char*) 0; // raise_exception();
array[84 * 4096] = 0;
```

- Flush+Reload over all pages of the array



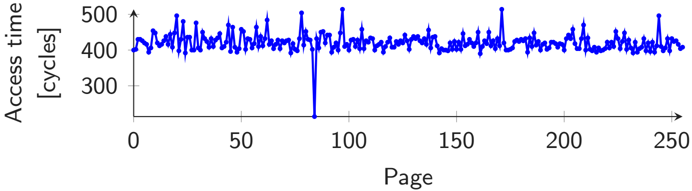
- Flush+Reload over all pages of the array



- “Unreachable” code line was actually executed

Building the Code

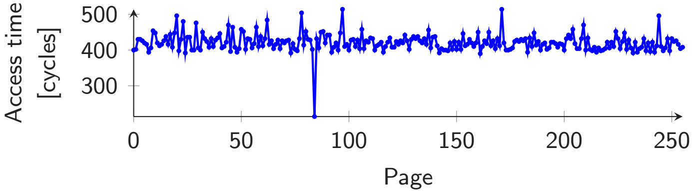
- Flush+Reload over all pages of the array



- "Unreachable" code line was actually executed
- Exception was only thrown afterwards

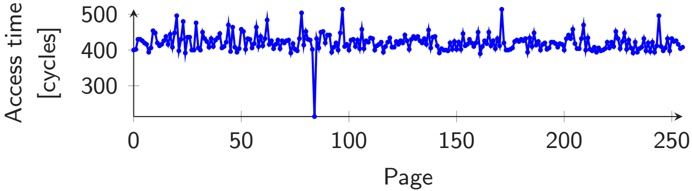
Moritz Lipp (@mlqxyz), Claudio Canella (@cc0x1f)

- Flush+Reload over all pages of the array



- "Unreachable" code line was actually executed
- Exception was only thrown afterwards
- Out-of-order instructions leave microarchitectural traces

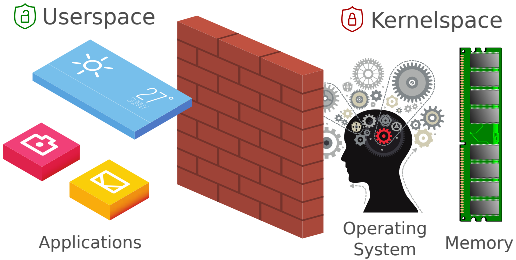
- Flush+Reload over all pages of the array



- "Unreachable" code line was actually executed
- Exception was only thrown afterwards
- Out-of-order instructions leave microarchitectural traces
- Give such instructions a name: transient instructions

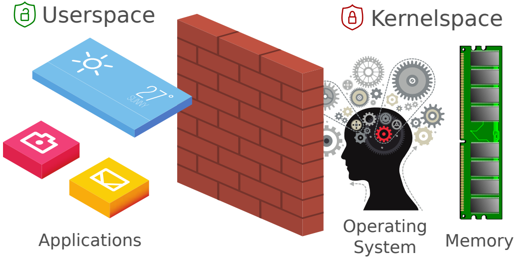
Does the CPU ignore the semantics of exceptions before handling it?

Memory Isolation

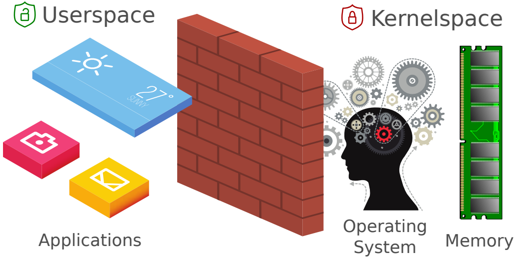


- Kernel is isolated from user space

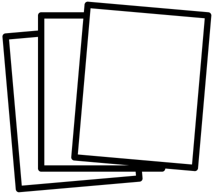
Memory Isolation



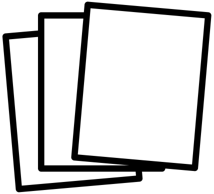
- Kernel is isolated from user space
- This isolation is a combination of hardware and software



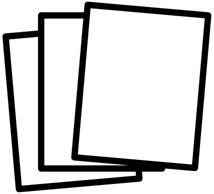
- Kernel is isolated from user space
- This isolation is a combination of hardware and software
- User applications cannot access anything from the kernel



- CPU support virtual address spaces to isolate processes

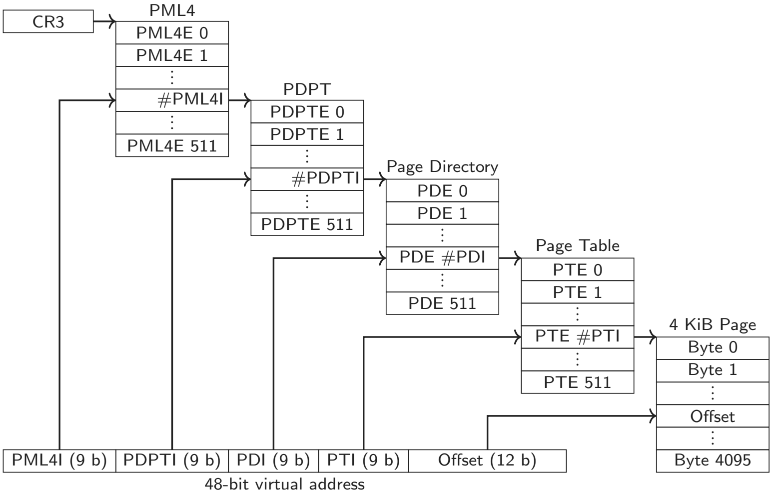


- CPU support virtual address spaces to isolate processes
- Physical memory is organized in page frames

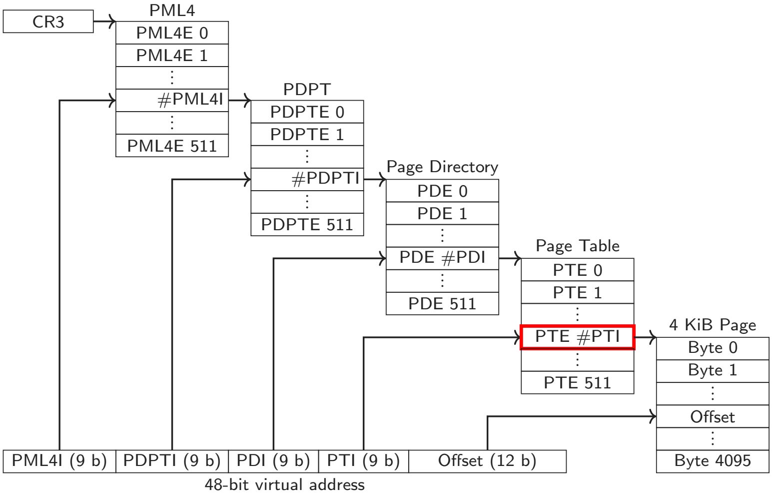


- CPU support virtual address spaces to isolate processes
- Physical memory is organized in page frames
- Virtual memory pages are mapped to page frames using page tables

Address Translation on x86-64



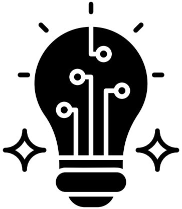
Address Translation on x86-64



Moritz Lipp (@mlqxyz), Claudio Canella (@cc0x1f)

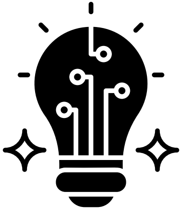
<table>
<tr><td>P</td><td>RW</td><td>US</td><td>WT</td><td>UC</td><td>R</td><td>D</td><td>S</td><td>G</td><td colspan="3">Ignored</td><td colspan="4"></td></tr>
<tr><td colspan="16" rowspan="2">Physical Page Number</td></tr>
<tr></tr>
<tr><td colspan="4"></td><td colspan="7">Ignored</td><td colspan="4">PK</td><td>X</td></tr>
</table>

- User/Supervisor bit defines in which privilege level the page can be accessed



- Add another layer of indirection to test

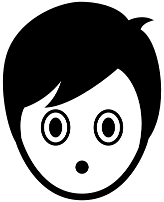
```c
char data = *(char*) 0xffffffff81a000e0; // kernel address
array[data * 4096] = 0;
```



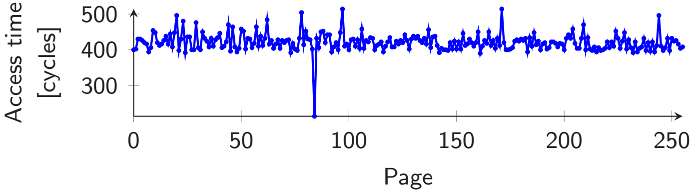
- Add another layer of indirection to test

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

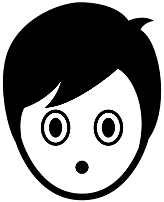
- Then check whether any part of `array` is cached



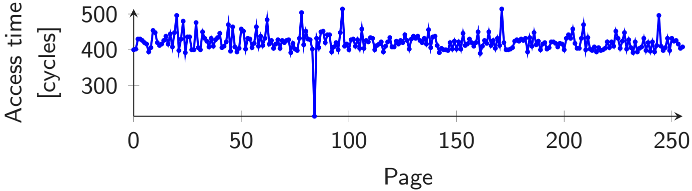
- Flush+Reload over all pages of the array



- Index of cache hit reveals data



- Flush+Reload over all pages of the array



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

MELTDOWN

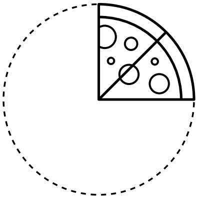
```
e01d8150: 69 6c 69 63 6f 6e 20 47  72 61 70 68 69 63 73 2c |ilicon Graphics,|
e01d8160: 20 49 6e 63 2e 20 20 48  6f 77 65 76 65 72 2c 20 | Inc.  However, |
e01d8170: 74 68 65 20 61 75 74 68  6f 72 73 20 6d 61 6b 65 |the authors make|
e01d8180: 20 6e 6f 20 63 6c 61 69  6d 20 74 68 61 74 20 4d | no claim that M|
e01d8190: 65 73 61 0a 20 69 73 20  69 6e 20 61 6e 79 20 77 |esa. is in any w|
e01d81a0: 61 79 20 61 20 63 6f 6d  70 61 74 69 62 6c 65 20 |ay a compatible |
e01d81b0: 72 65 70 6c 61 63 65 6d  65 6e 74 20 66 6f 72 20 |replacement for |
e01d81c0: 4f 70 65 6e 47 4c 20 6f  72 20 61 73 73 6f 63 69 |OpenGL or associ|
e01d81d0: 61 74 65 64 20 77 69 74  68 0a 20 53 69 6c 69 63 |ated with. Silic|
e01d81e0: 6f 6e 20 47 72 61 70 68  69 63 73 2c 20 49 6e 63 |on Graphics, Inc|
e01d81f0: 2e 0a 20 2e 0a 20 54 68  69 73 20 76 65 72 73 69 |.. .. This versi|
e01d8200: 6f 6e 20 6f 66 20 4d 65  73 61 20 70 72 6f 76 69 |on of Mesa provi|
e01d8210: 64 65 73 20 47 4c 58 20  61 6e 64 20 44 52 49 20 |des GLX and DRI |
e01d8220: 63 61 70 61 62 69 6c 69  74 69 65 73 3a 20 69 74 |capabilities: it|
e01d8230: 20 69 73 20 63 61 70 61  62 6c 65 20 6f 66 0a 20 | is capable of. |
e01d8240: 62 6f 74 68 20 64 69 72  65 63 74 20 61 6e 64 20 |both direct and |
e01d8250: 69 6e 64 69 72 65 63 74  20 72 65 6e 64 65 72 69 |indirect renderi|
e01d8260: 6e 67 2e 20 20 46 6f 72  20 64 69 72 65 63 74 20 |ng.  For direct |
e01d8270: 72 65 6e 64 65 72 69 6e  67 2c 20 69 74 20 63 61 |rendering, it ca|
e01d8280: 6e 20 75 73 65 20 44 52  49 0a 20 6d 6f 64 75 6c |n use DRI. modul|
```

Meltdown

- Using out-of-order execution, we can read data at any address

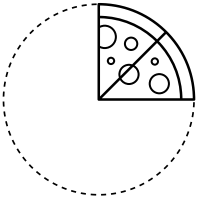
Meltdown

- Using out-of-order execution, we can read data at any address
- Entire physical memory is typically accessible through kernel space



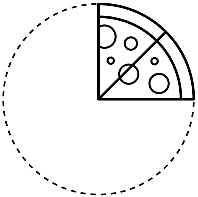
- Assumed that one can only read data stored in the L1 with Meltdown

Uncached memory

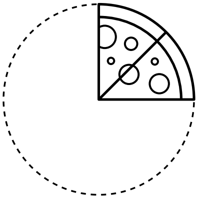


- Assumed that one can only read data stored in the L1 with Meltdown
- Experiment where a thread flushes the value constantly and a thread on a different core reloads the value

Uncached memory

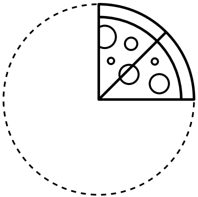


- Assumed that one can only read data stored in the L1 with Meltdown
- Experiment where a thread flushes the value constantly and a thread on a different core reloads the value
  - Target data is not in the L1 cache of the attacking core



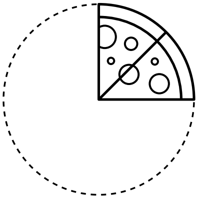
- Assumed that one can only read data stored in the L1 with Meltdown
- Experiment where a thread flushes the value constantly and a thread on a different core reloads the value
  - Target data is not in the L1 cache of the attacking core
- We can still leak the data at a lower reading rate

Uncached memory



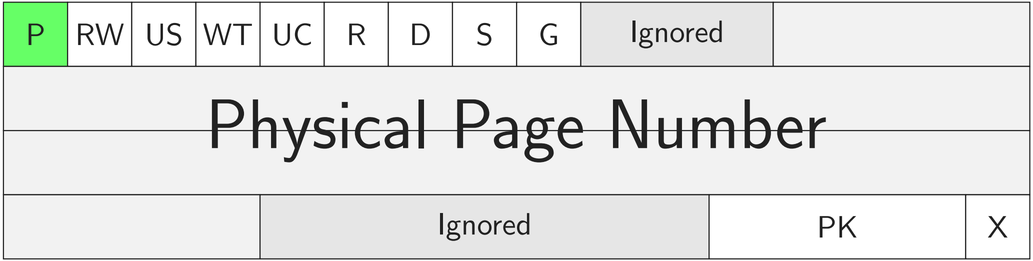
- Assumed that one can only read data stored in the L1 with Meltdown
- Experiment where a thread flushes the value constantly and a thread on a different core reloads the value
  - Target data is not in the L1 cache of the attacking core
- We can still leak the data at a lower reading rate
  - Meltdown might implicitly cache the data

Uncached memory

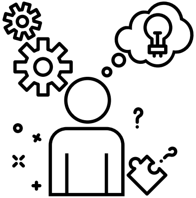


- Assumed that one can only read data stored in the L1 with Meltdown
- Experiment where a thread flushes the value constantly and a thread on a different core reloads the value
  - Target data is not in the L1 cache of the attacking core
- We can still leak the data at a lower reading rate
  - Meltdown might implicitly cache the data

Which bits are left in the PTE?

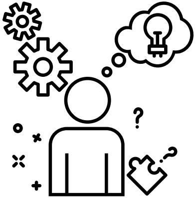


- Present bit defines whether a page is present in physical memory.



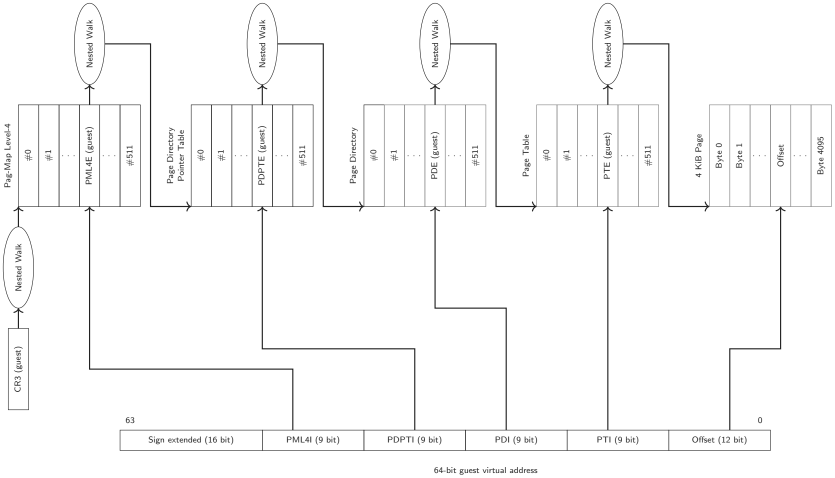
- Can not really manage the PTE from user space

Problem



- Can not really manage the PTE from user space
- Use virtual machine
  - → completely controlled by attacker

Virtual Machine: Extended Page Table



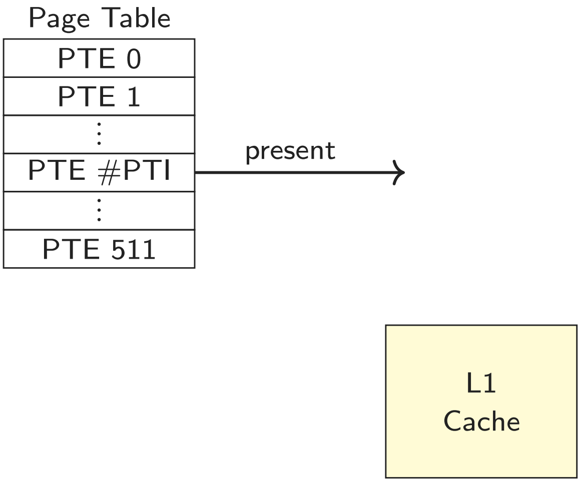
Present Bit

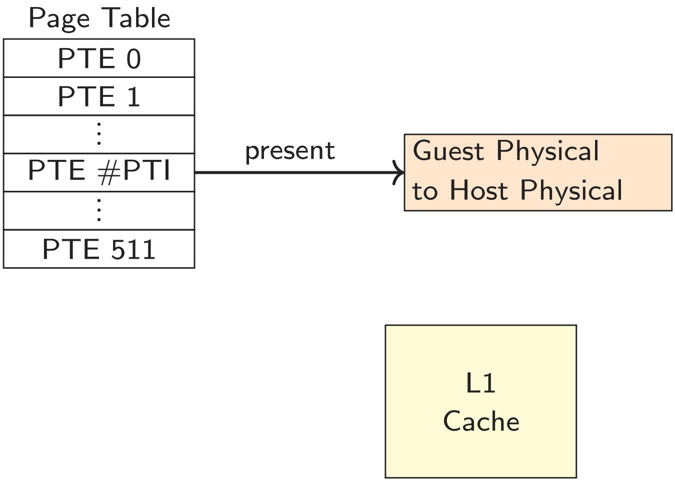
Page Table

PTE 0
PTE 1
⋮
PTE #PTI
⋮
PTE 511

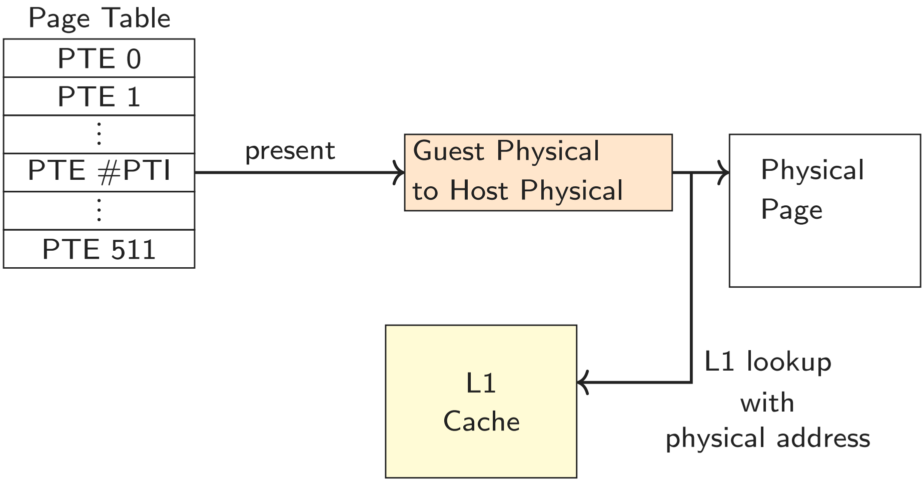
L1
Cache

Present Bit




Page Table
PTE 0
PTE 1
PTE #PTI
PTE 511
present
Guest Physical to Host Physical
L1 Cache

Present Bit


Page Table
PTE 0
PTE 1
PTE #PTI
PTE 511
present
Guest Physical
to Host Physical
Physical
Page
L1
Cache
L1 lookup
with
physical address

Present Bit

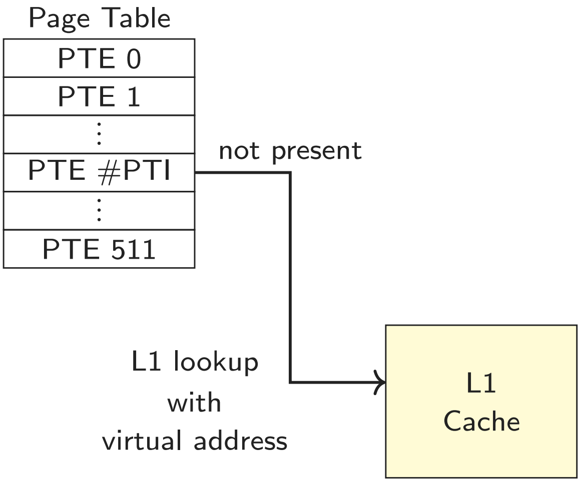
Page Table

PTE 0
PTE 1
⋮
PTE #PTI
⋮
PTE 511

not present

L1
Cache

Present Bit


Page Table
PTE 0
PTE 1
PTE #PTI
PTE 511
not present
L1 lookup
with
virtual address
L1
Cache

Demo

- Foreshadow or L1TF

- Foreshadow or L1TF
- Leak data from L1 data cache

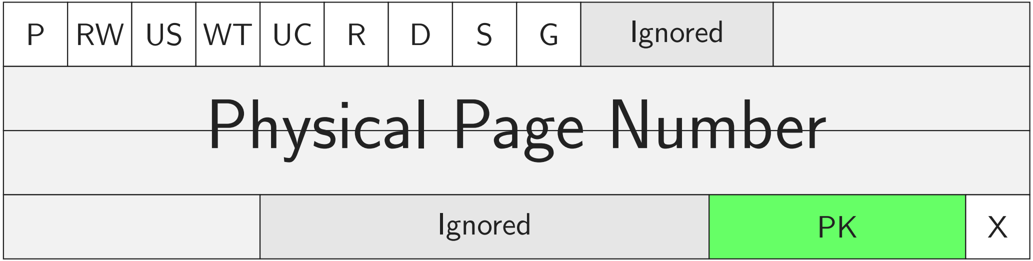
Impact

- Foreshadow or L1TF
- Leak data from L1 data cache
- Affects virtual machines (VM), hypervisors (VMM), operating systems (OS) and system management mode (SMM)

- Foreshadow or L1TF
- Leak data from L1 data cache
- Affects virtual machines (VM), hypervisors (VMM), operating systems (OS) and system management mode (SMM)
- Read SGX-protected memory and leak machine's private attestation key

There are still bits left in the PTE

Page Table Entry



- PK bits define the assigned protection key

Moritz Lipp (@mlqxyz), Claudio Canella (@cc0x1f)

- Protection key for a group of pages

Intel MPK

- Protection key for a group of pages
- 4 bits in PTE identify key for protected memory regions

Intel MPK

- Protection key for a group of pages
- 4 bits in PTE identify key for protected memory regions
- Quick update of access rights

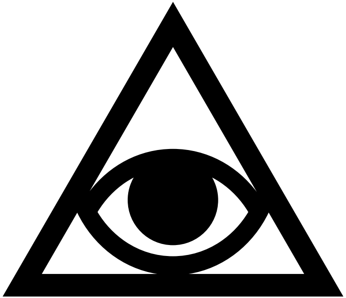
- Protection key for a group of pages
- 4 bits in PTE identify key for protected memory regions
- Quick update of access rights
- Available on Intel Xeon CPUs

Protection Keys

- Protection keys are lazily enforced

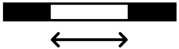
Protection Keys

- Protection keys are lazily enforced
- Protected value is forwarded to transient instructions



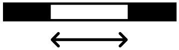
Other exceptions?

Bound Range Exceeded

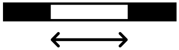


- x86 provides dedicated instruction raising #BR exception if bound-range is exceeded

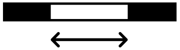
Bound Range Exceeded



- x86 provides dedicated instruction raising #BR exception if bound-range is exceeded
- Data used in transient execution



- x86 provides dedicated instruction raising #BR exception if bound-range is exceeded
- Data used in transient execution
- Attacker can determine data

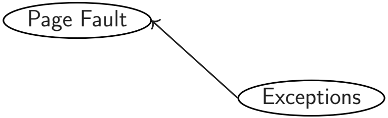


- x86 provides dedicated instruction raising #BR exception if bound-range is exceeded
- Data used in transient execution
- Attacker can determine data
- First Meltdown-type attack on AMD

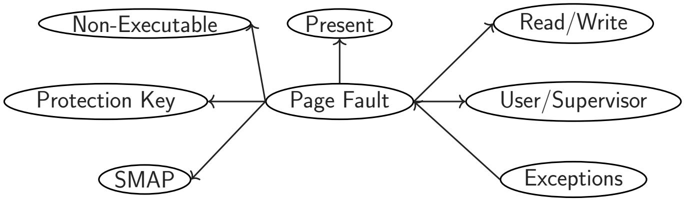
We tested all of them

Take a closer look . . . again

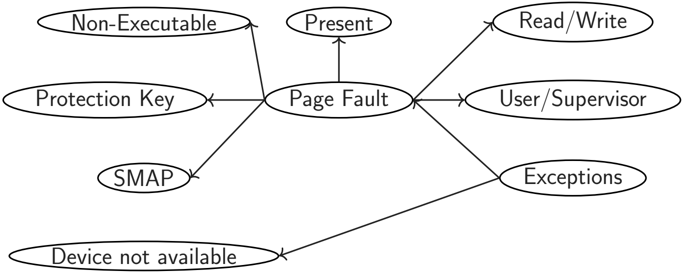
Exceptions


Page Fault
Exceptions

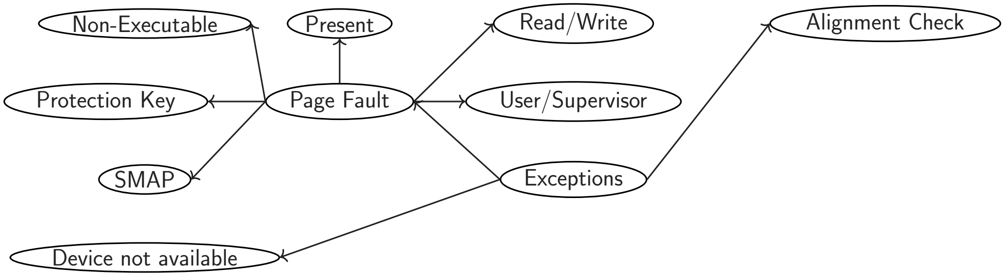
Take a closer look . . . again



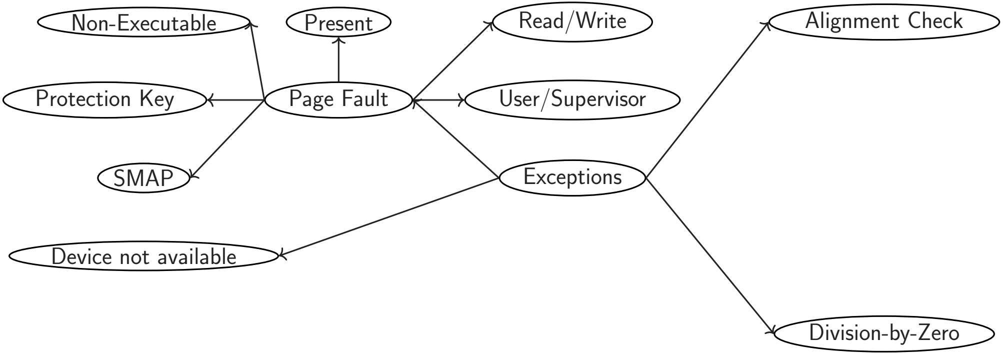
Take a closer look . . . again

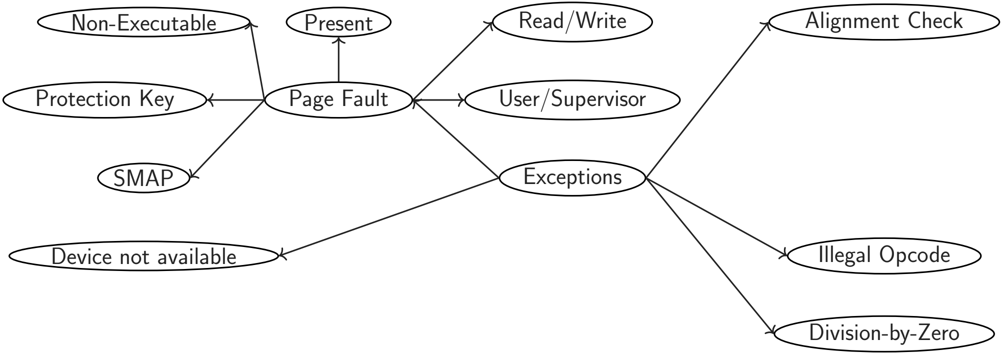


Take a closer look . . . again

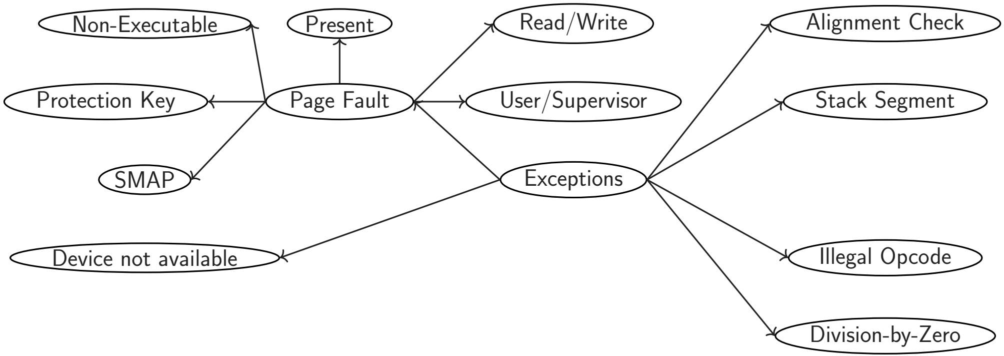


Take a closer look . . . again

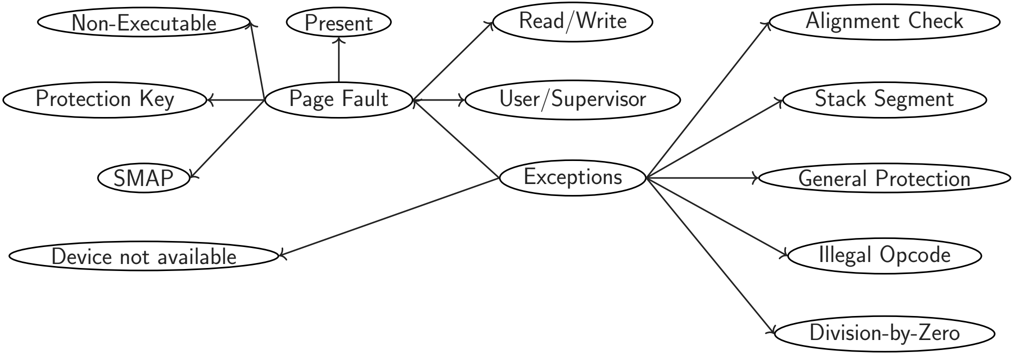



Non-Executable
Present
Read/Write
Alignment Check
Protection Key
Page Fault
User/Supervisor
SMAP
Exceptions
Device not available
Illegal Opcode
Division-by-Zero

Take a closer look . . . again

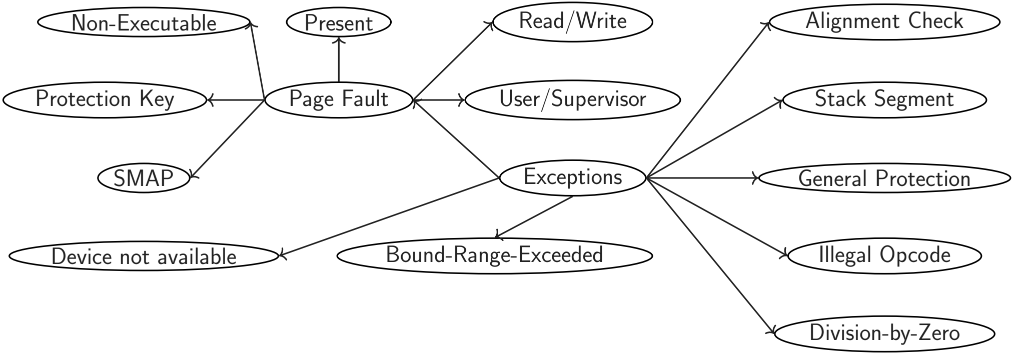


Take a closer look . . . again

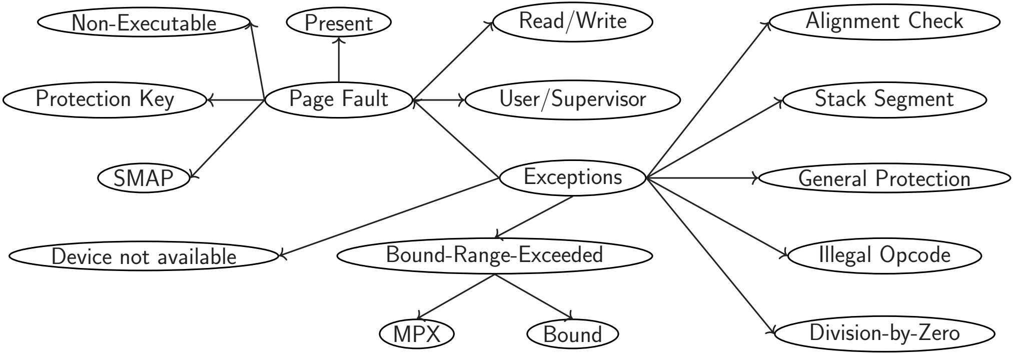


Moritz Lipp (@mlqxyz), Claudio Canella (@cc0x1f)

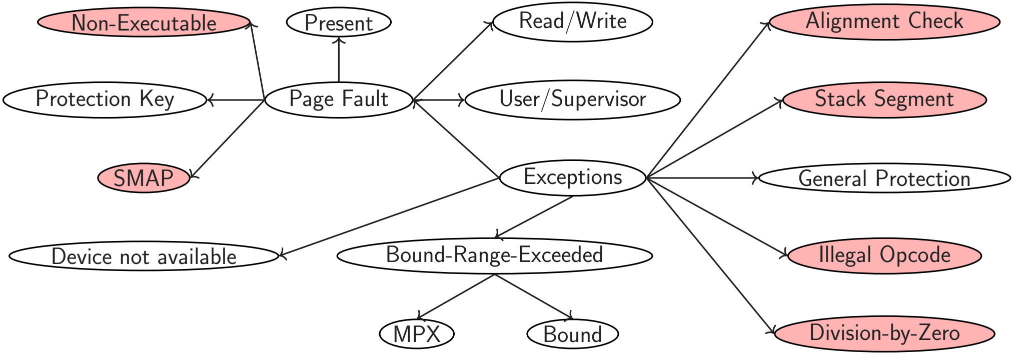
Take a closer look . . . again



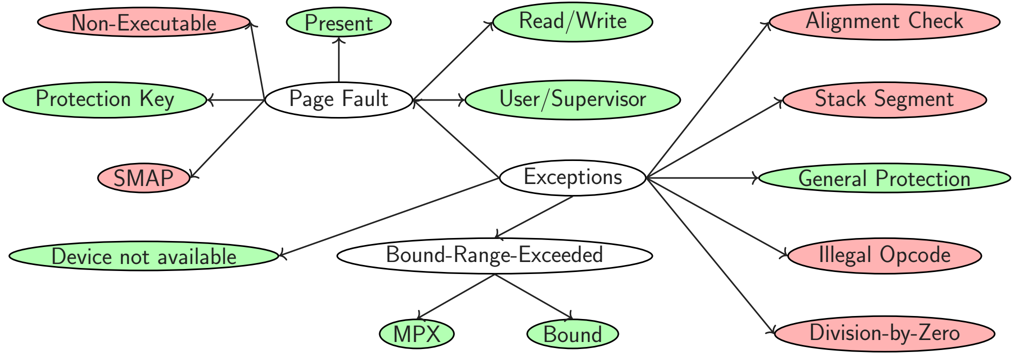
Take a closer look . . . again

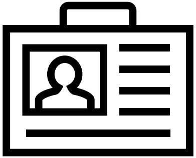


Take a closer look . . . again



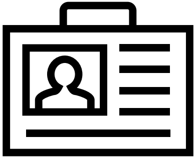
Take a closer look . . . again





- Names are still confusing

New Naming Scheme

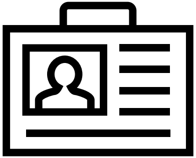


- Names are still confusing
- Propose: use exception/bit for naming attack

New Naming Scheme

- Names are still confusing
- Propose: use exception/bit for naming attack
  - $\rightarrow$ Meltdown = Meltdown-US

New Naming Scheme

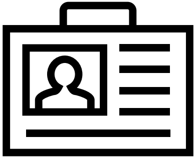


- Names are still confusing
- Propose: use exception/bit for naming attack
  - → Meltdown = Meltdown-US
  - → Foreshadow = Meltdown-P

New Naming Scheme

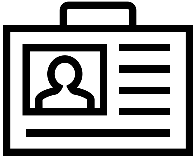
- Names are still confusing
- Propose: use exception/bit for naming attack
  - → Meltdown = Meltdown-US
  - → Foreshadow = Meltdown-P
  - → Meltdown 3a = Meltdown-GP

New Naming Scheme

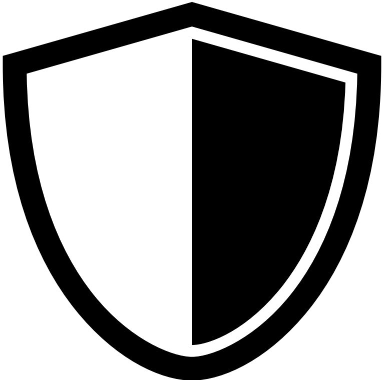


- Names are still confusing
- Propose: use exception/bit for naming attack
  - → Meltdown = Meltdown-US
  - → Foreshadow = Meltdown-P
  - → Meltdown 3a = Meltdown-GP
  - → Protection Keys = Meltdown-PK

New Naming Scheme



- Names are still confusing
- Propose: use exception/bit for naming attack
  - → Meltdown = Meltdown-US
  - → Foreshadow = Meltdown-P
  - → Meltdown 3a = Meltdown-GP
  - → Protection Keys = Meltdown-PK
  - → ...



How can we fix this?

Meltdown Defense Categorization

Meltdown defenses in 2 categories:

Meltdown Defense Categorization

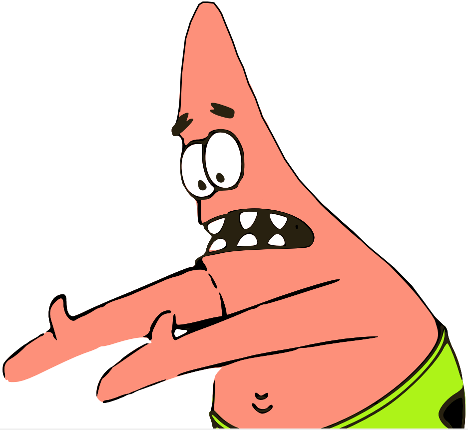
Meltdown defenses in 2 categories:

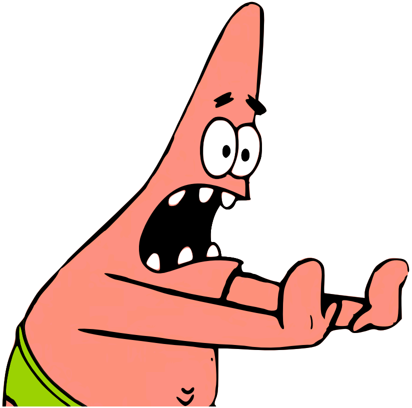
D1 Architecturally inaccessible data is also microarchitecturally inaccessible

Take the kernel addresses...

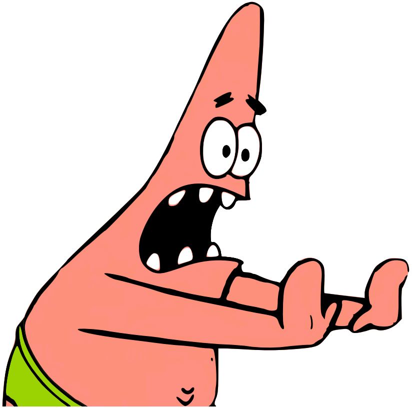
- Kernel addresses in user space are a problem

- Kernel addresses in user space are a problem
- Why don't we take the kernel addresses...



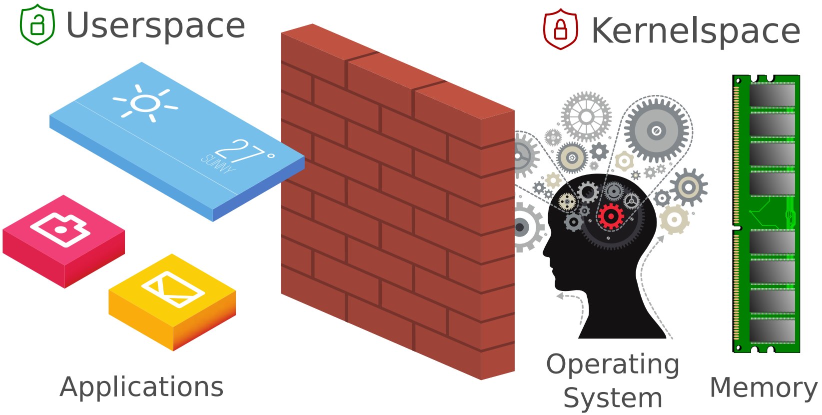


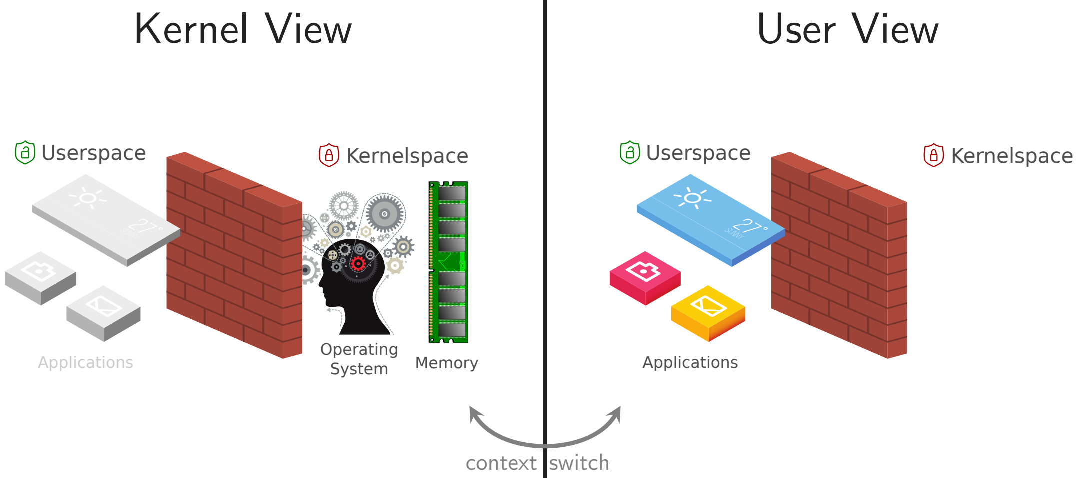
- ...and remove them if not needed?

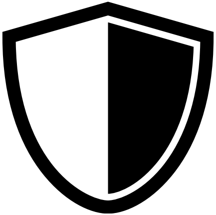


- ...and remove them if not needed?
- User accessible check in hardware is not reliable

Meltdown-US Mitigation: KAISER

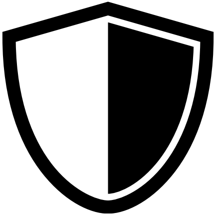



Kernel View
Userspace
Kernelspace
Applications
Operating System
Memory
User View
Userspace
Kernelspace
Applications
context switch

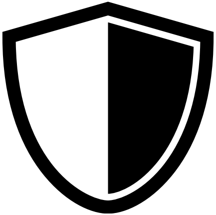


- **Linux**: Kernel Page-table Isolation (KPTI)

Mitigations



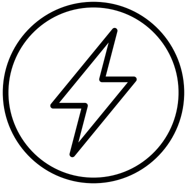
- **Linux**: Kernel Page-table Isolation (KPTI)
- **Apple**: Released updates



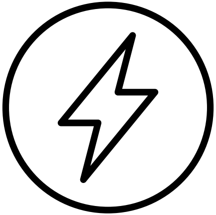
- **Linux**: Kernel Page-table Isolation (KPTI)
- **Apple**: Released updates
- **Windows**: Kernel Virtual Address (KVA) Shadow

Meltdown defenses in 2 categories:

D1 Architecturally inaccessible data is also microarchitecturally inaccessible

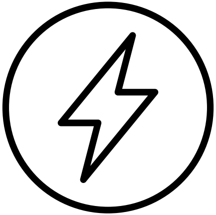


D2 Preventing occurrence of faults

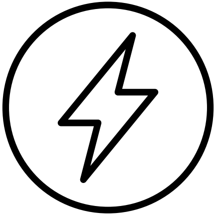


- Prevent the occurrence of faults in the first place

D2: Preventing occurrence of faults

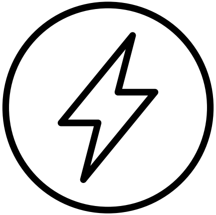


- Prevent the occurrence of faults in the first place
  - Accesses which would normally fault → become architecturally and microarchitecturally valid accesses



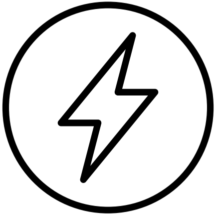
- Prevent the occurrence of faults in the first place
  - Accesses which would normally fault $\rightarrow$ become architecturally and microarchitecturally valid accesses
  - But do not leak data

D2: Preventing occurrence of faults



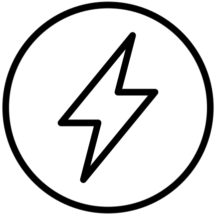
- Prevent the occurrence of faults in the first place
  - Accesses which would normally fault → become architecturally and microarchitecturally valid accesses
  - But do not leak data
- SGX Page Abort Semantics

D2: Preventing occurrence of faults



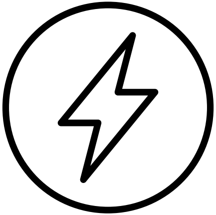
- Prevent the occurrence of faults in the first place
  - Accesses which would normally fault → become architecturally and microarchitecturally valid accesses
  - But do not leak data
- SGX Page Abort Semantics
  - Returns −1 when enclave memory is accessed from the outside world

D2: Preventing occurrence of faults



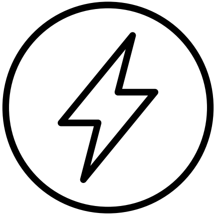
- Prevent the occurrence of faults in the first place
  - Accesses which would normally fault $\rightarrow$ become architecturally and microarchitecturally valid accesses
  - But do not leak data
- SGX Page Abort Semantics
  - Returns -1 when enclave memory is accessed from the outside world
- Meltdown-NM mitigation: Use eager switching instead of lazy switching in the FPU

D2: Preventing occurrence of faults



- Prevent the occurrence of faults in the first place
  - Accesses which would normally fault $\rightarrow$ become architecturally and microarchitecturally valid accesses
  - But do not leak data
- SGX Page Abort Semantics
  - Returns −1 when enclave memory is accessed from the outside world
- Meltdown-NM mitigation: Use eager switching instead of lazy switching in the FPU
  - FPU is always available $\rightarrow$ never faults

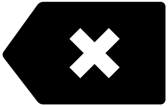
D2: Preventing occurrence of faults



- Prevent the occurrence of faults in the first place
  - Accesses which would normally fault → become architecturally and microarchitecturally valid accesses
  - But do not leak data
- SGX Page Abort Semantics
  - Returns −1 when enclave memory is accessed from the outside world
- Meltdown-NM mitigation: Use eager switching instead of lazy switching in the FPU
  - FPU is always available → never faults

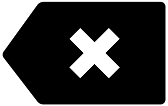
Meltdown-P Mitigation

Meltdown-P Mitigation



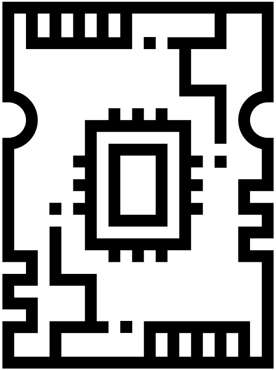
Clear phyiscal address field of unmapped PTEs

Meltdown-P Mitigation



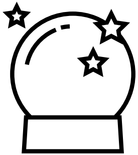
Clear phyiscal address field of unmapped PTEs

Flush L1 upon switching protection domains

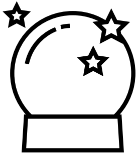


What about other microarchitectural elements?

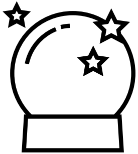
Branch Prediction



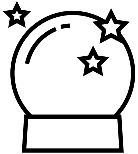
- CPU tries to predict the future, . . .



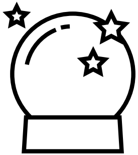
- CPU tries to predict the future, . . .
    - . . . based on what happened in the past



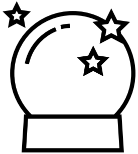
- CPU tries to predict the future, ...
  - ...based on what happened in the past
- Speculative execution of instructions



- CPU tries to predict the future, ...
  - ... based on what happened in the past
- Speculative execution of instructions
- Correct prediction, ...



- CPU tries to predict the future, ...
  - ...based on what happened in the past
- Speculative execution of instructions
- Correct prediction, ...
  - ...very fast



- CPU tries to predict the future, . . .
  - . . . based on what happened in the past
- Speculative execution of instructions
- Correct prediction, . . .
  - . . . very fast
  - otherwise: Discard results

- CPU has many different predictors . . .

- CPU has many different predictors ...

Pattern History Table
(PHT)

Branch Prediction

- CPU has many different predictors ...

Pattern History Table (PHT)

Branch Target Buffer (BTB)

Branch Prediction

- CPU has many different predictors . . .

Pattern History Table (PHT)

Branch Target Buffer (BTB)

Return Stack Buffer (RSB)

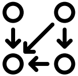
Branch Prediction

- CPU has many different predictors . . .

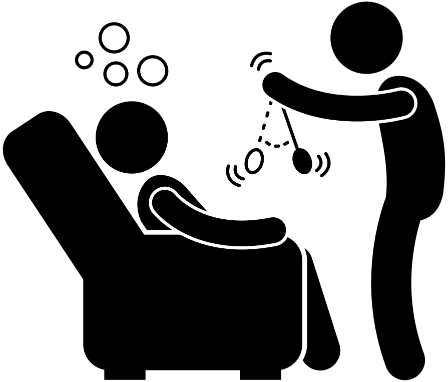
Pattern History Table (PHT)

Branch Target Buffer (BTB)

Return Stack Buffer (RSB)

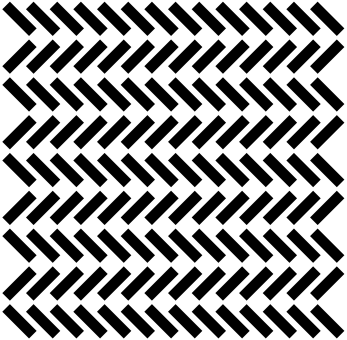


Memory Disambiguation (STL)



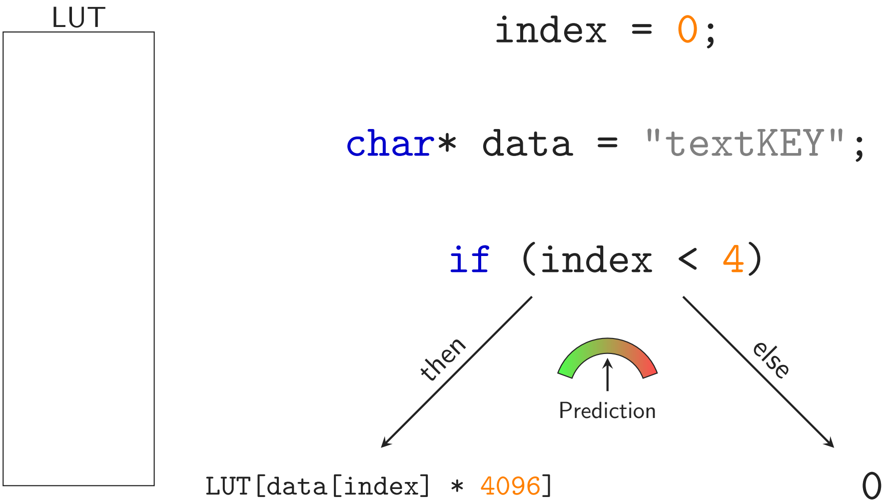
Can we trick the CPU into executing code speculatively?

And exploit that?



Pattern History Table (PHT)

Pattern History Table (PHT)



Moritz Lipp (@mlqxyz), Claudio Canella (@cc0x1f)

LUT

```
index = 0;

char* data = "textKEY";

if (index < 4)
```

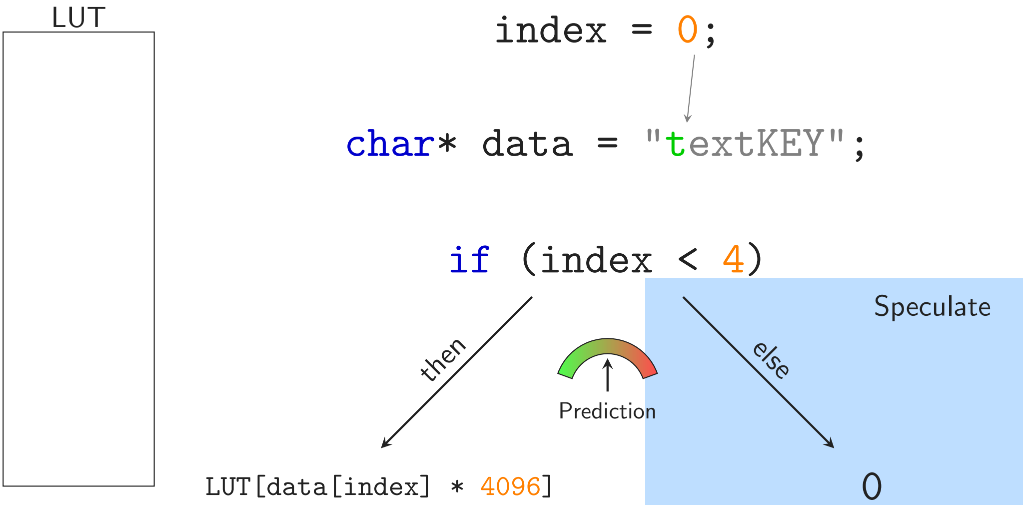
then

Prediction

else

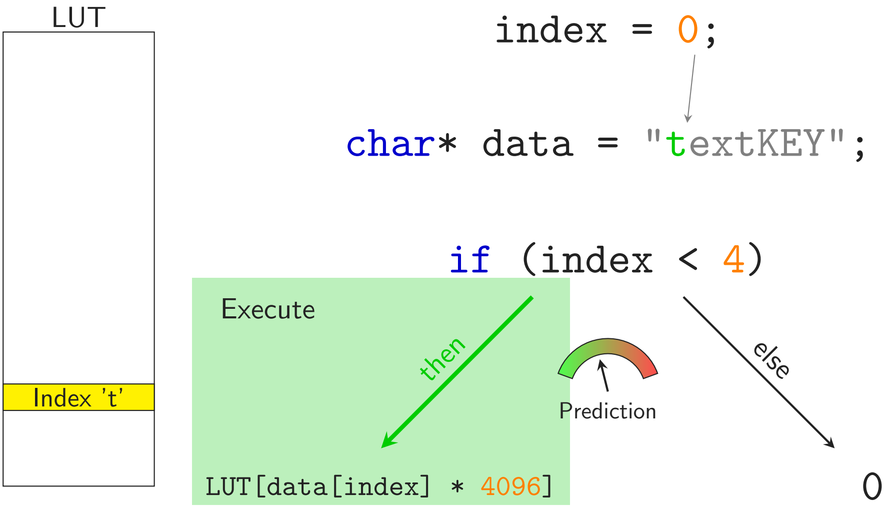
```
LUT[data[index] * 4096]
```

0



Moritz Lipp (@mlqxyz), Claudio Canella (@cc0x1f)

Pattern History Table (PHT)



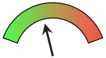
Pattern History Table (PHT)

LUT

```
index = 1;

char* data = "textKEY";

if (index < 4)
```

then

else

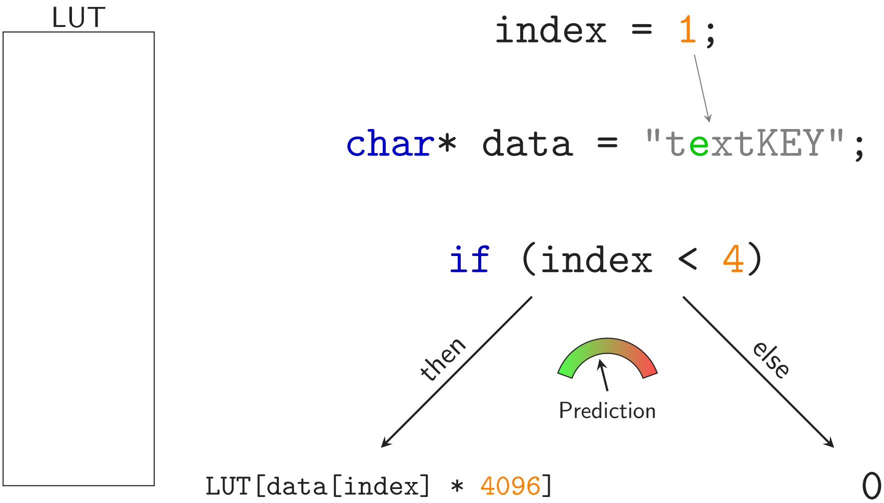


Prediction

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

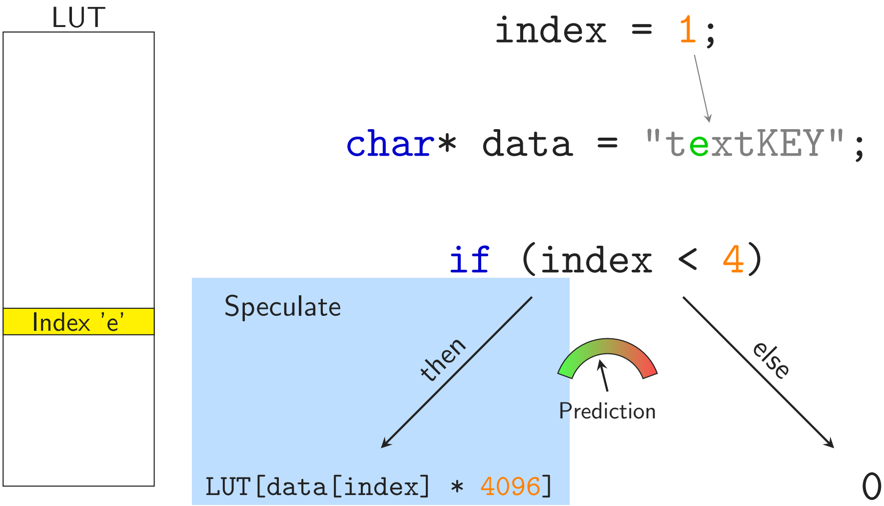
0

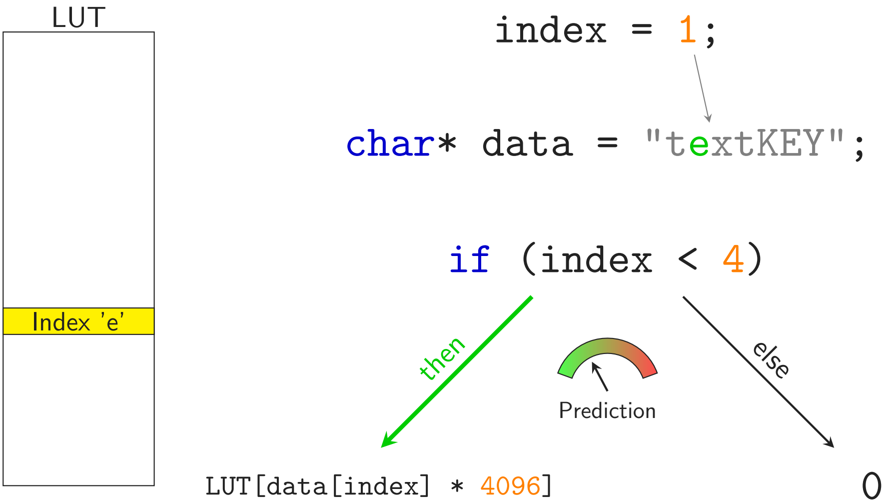
Pattern History Table (PHT)



Moritz Lipp (@mlqxyz), Claudio Canella (@cc0x1f)

Pattern History Table (PHT)



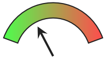

LUT
Index 'e'
index = 1;
char* data = "textKEY";
if (index < 4)
then
else
Prediction
LUT[data[index] * 4096]
0

Pattern History Table (PHT)

LUT

```
index = 2;

char* data = "textKEY";

if (index < 4)
```

then



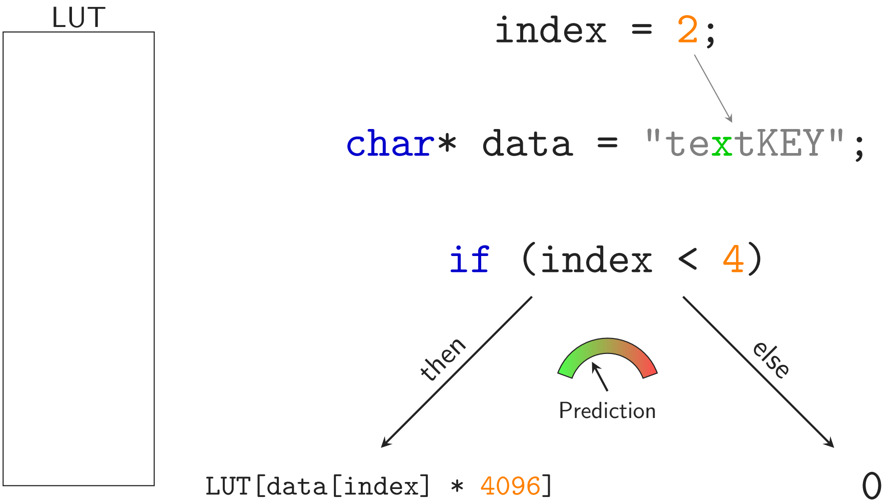
Prediction

else

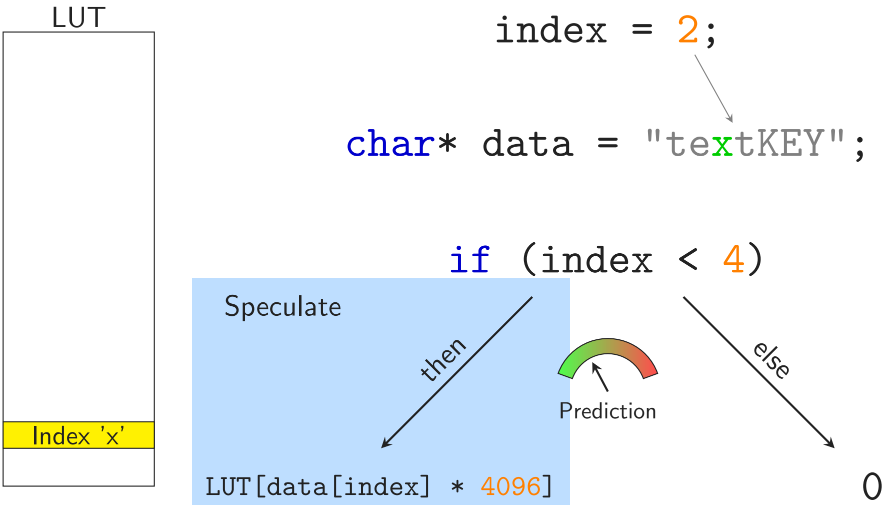
```
LUT[data[index] * 4096]
```

0

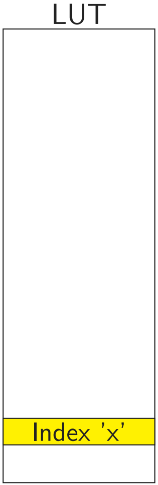
Pattern History Table (PHT)



Moritz Lipp (@mlqxyz), Claudio Canella (@cc0x1f)


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

Pattern History Table (PHT)




```
index = 2;

char* data = "textKEY";

if (index < 4)
```

then

else

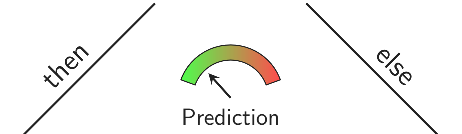
```
LUT[data[index] * 4096]
```

0

Moritz Lipp (@mlqxyz), Claudio Canella (@cc0x1f)

LUT

```
index = 3;

char* data = "textKEY";

if (index < 4)
```




```
LUT[data[index] * 4096]                 0
```

Pattern History Table (PHT)

LUT

```
index = 3;

char* data = "textKEY";

if (index < 4)
```

then

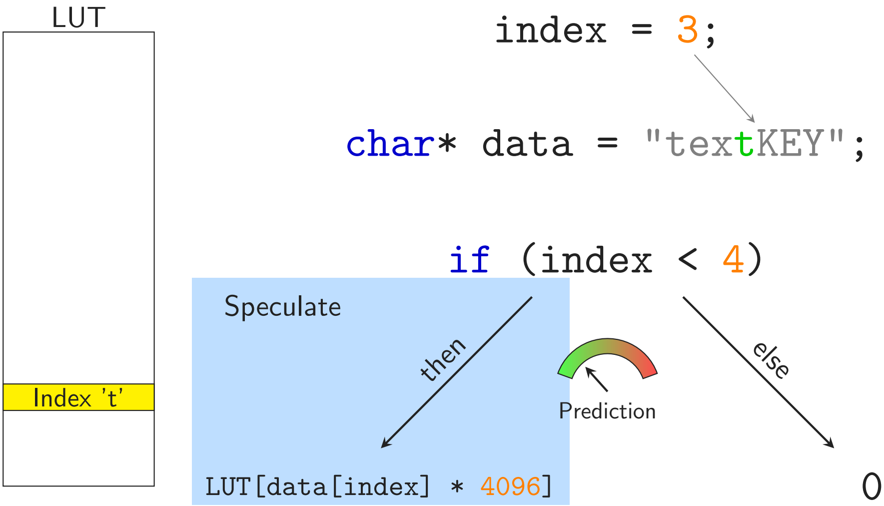
else

Prediction

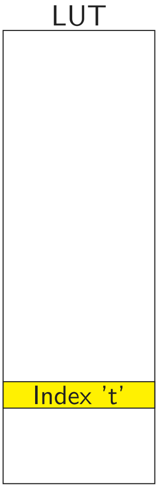
```
LUT[data[index] * 4096]
```

0

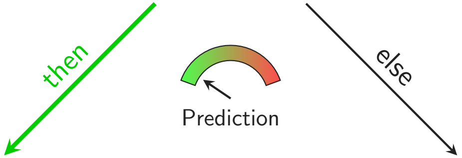
Pattern History Table (PHT)



Pattern History Table (PHT)




```
index = 3;

char* data = "textKEY";

if (index < 4)
```




```
LUT[data[index] * 4096]                0
```

LUT

```
index = 4;

char* data = "textKEY";

if (index < 4)
```

then

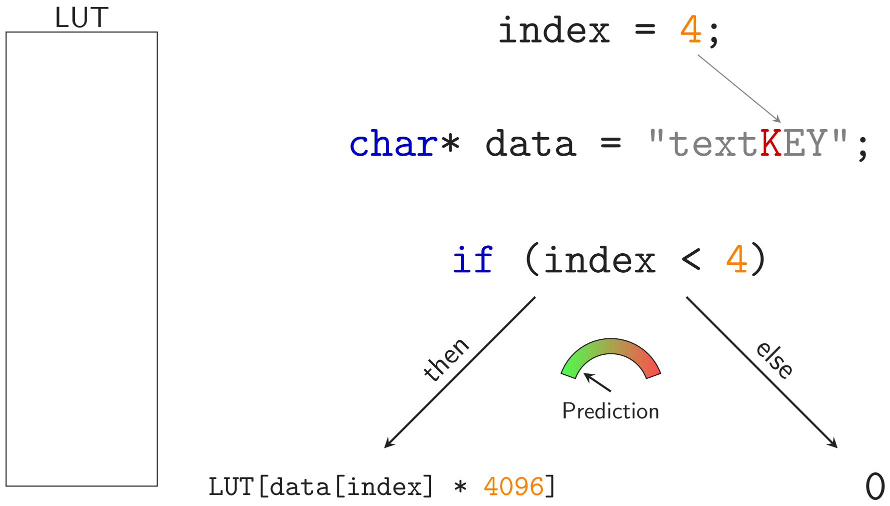
else

Prediction

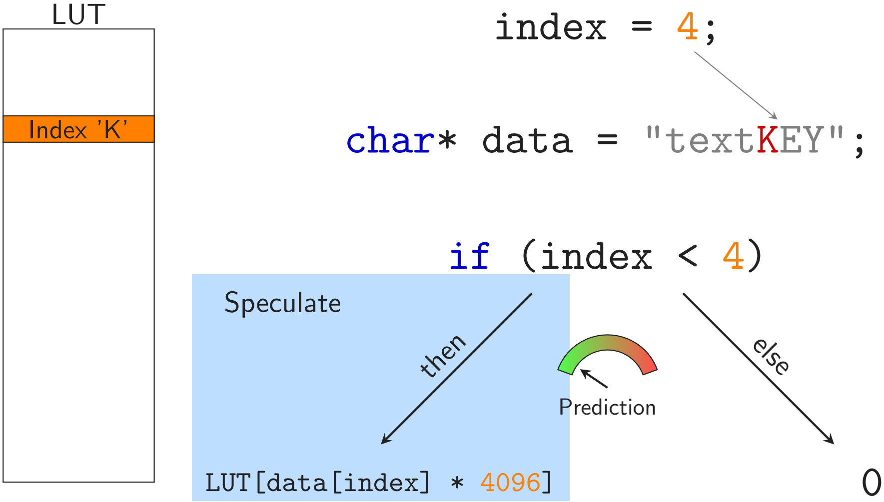
```
LUT[data[index] * 4096]
```

0

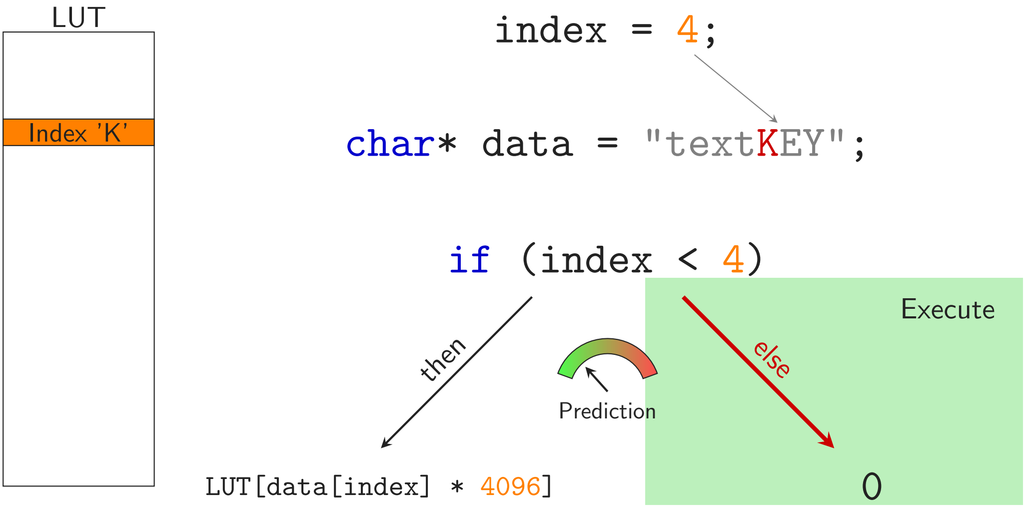
Pattern History Table (PHT)



Pattern History Table (PHT)



Pattern History Table (PHT)



Pattern History Table (PHT)

LUT

```
index = 5;

char* data = "textKEY";

if (index < 4)
```

then

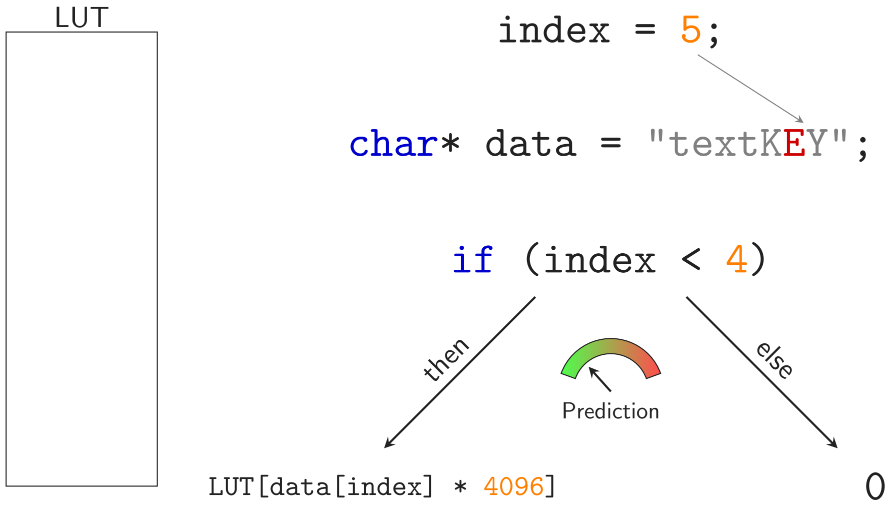
Prediction

else

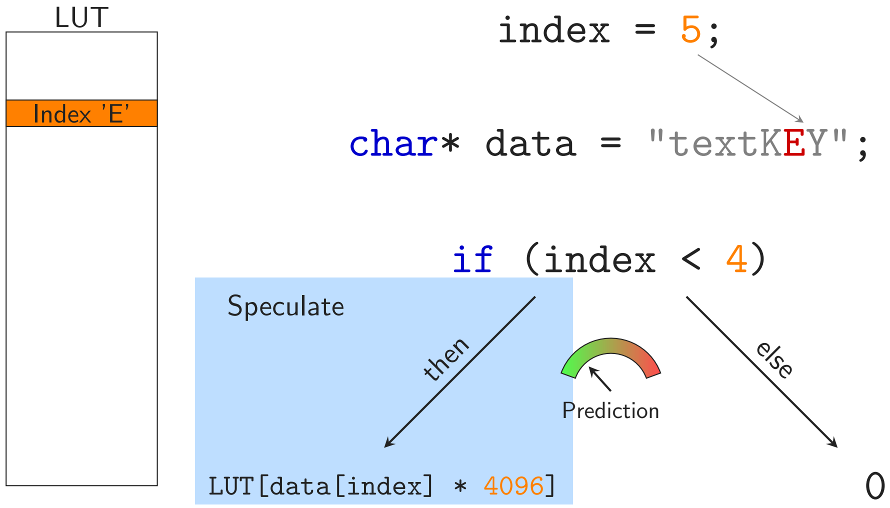
```
LUT[data[index] * 4096]
```

0

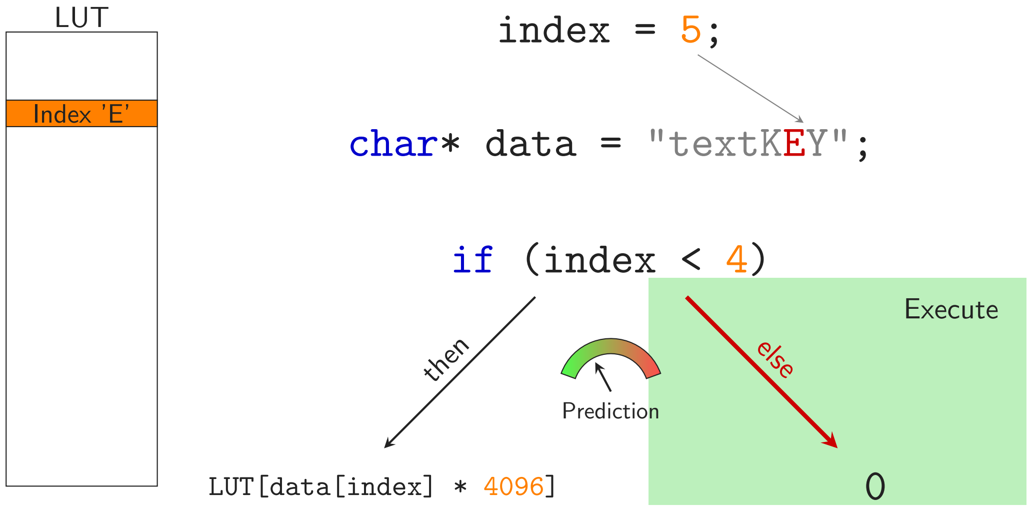
Pattern History Table (PHT)



Pattern History Table (PHT)



Pattern History Table (PHT)



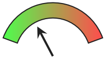
Pattern History Table (PHT)

LUT

```
index = 6;

char* data = "textKEY";

if (index < 4)
```

then

else

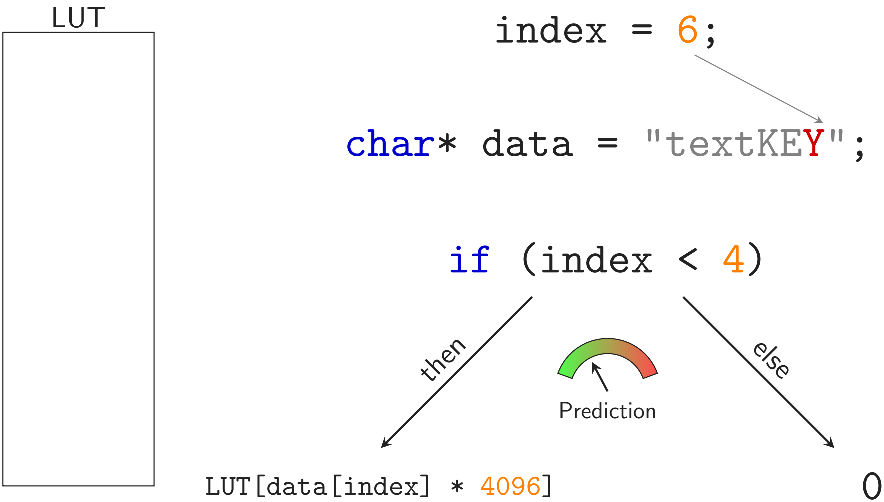


Prediction

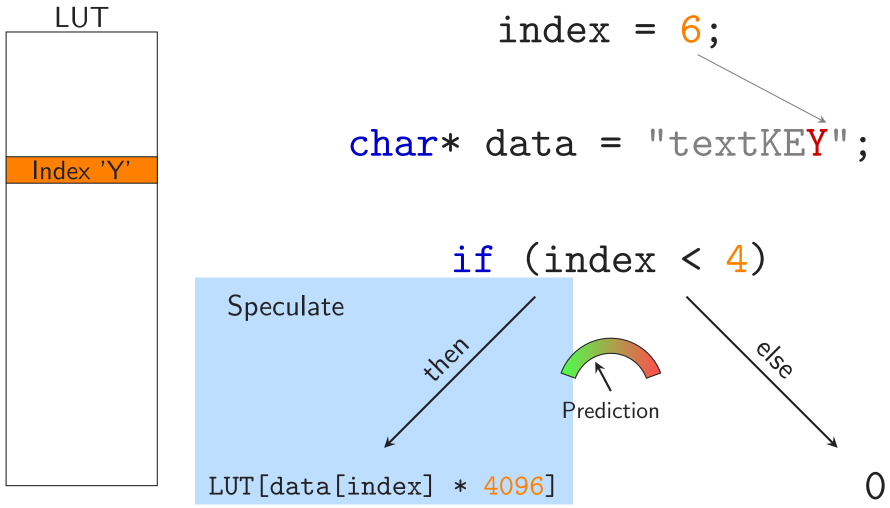
```
LUT[data[index] * 4096]
```

0

Pattern History Table (PHT)

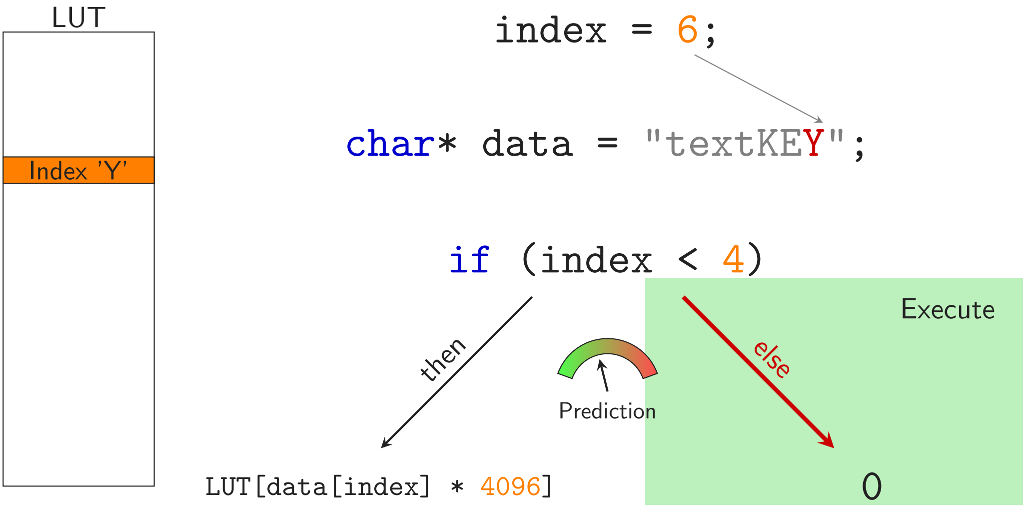


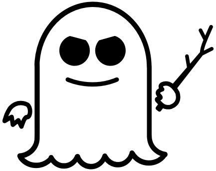
Pattern History Table (PHT)



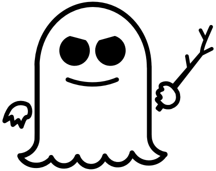
Moritz Lipp (@mlqxyz), Claudio Canella (@cc0x1f)

Pattern History Table (PHT)

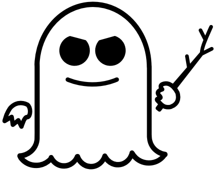




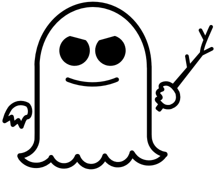
- Spectre Variant 1: Bounds Check Bypass on Load



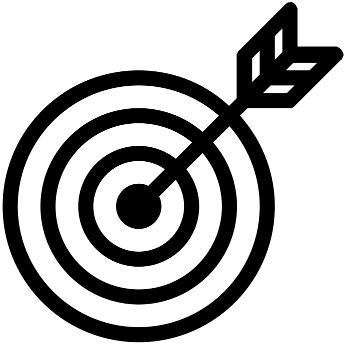
- Spectre Variant 1: Bounds Check Bypass on Load
- Spectre Variant 1.1: Bounds Check Bypass on Stores



- Spectre Variant 1: Bounds Check Bypass on Load
- Spectre Variant 1.1: Bounds Check Bypass on Stores
- Intel: Side-Channel Vulnerability 1



- Spectre Variant 1: Bounds Check Bypass on Load
- Spectre Variant 1.1: Bounds Check Bypass on Stores
- Intel: Side-Channel Vulnerability 1
- Propose: Spectre-PHT



Branch Target Buffer (BTB)

Branch Target Buffer (BTB)

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

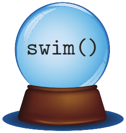
```
a->move()
```

fly()

swim()

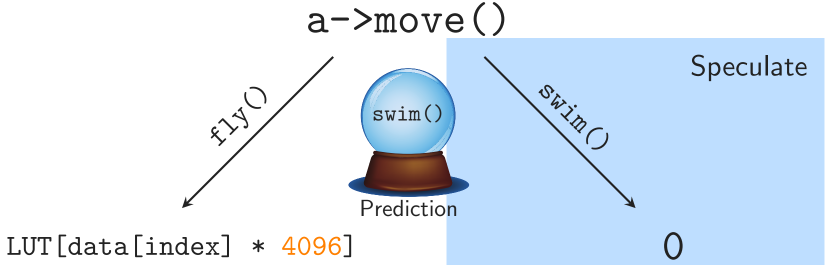
LUT[data[index] * 4096]

0



Prediction

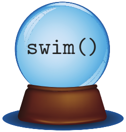
```
Animal* a = bird;
```



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

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

fly()



Prediction

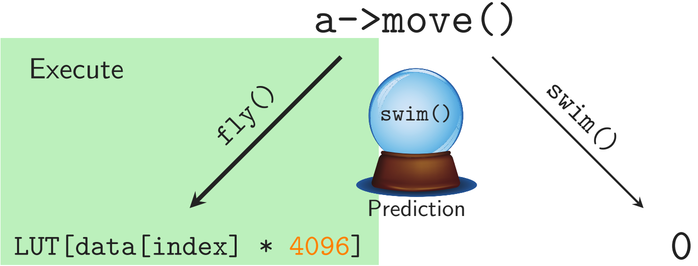
swim()

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

0

Branch Target Buffer (BTB)

`Animal* a = bird;`



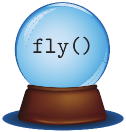
Branch Target Buffer (BTB)

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

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

fly()

swim()



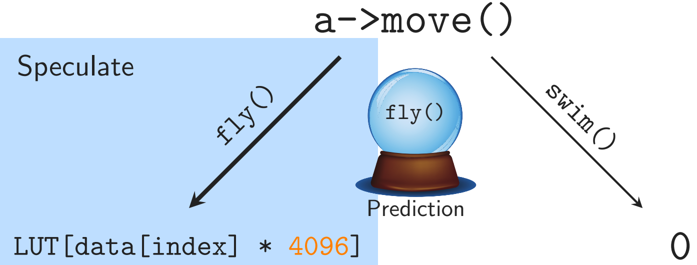
Prediction

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

0

Branch Target Buffer (BTB)

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



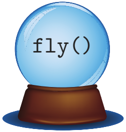
Moritz Lipp (@mlqxyz), Claudio Canella (@cc0x1f)

Branch Target Buffer (BTB)

`Animal* a = bird;`

`a->move()`

`fly()` `swim()`

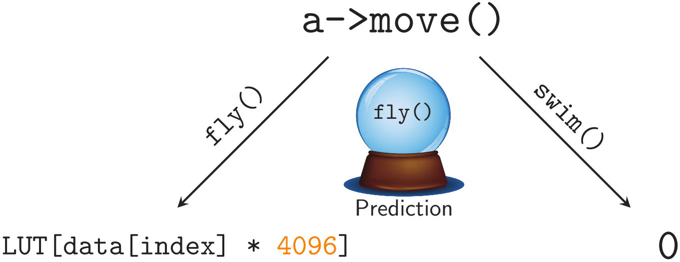


Prediction

`LUT[data[index] * 4096]`

0

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



Branch Target Buffer (BTB)

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

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

Speculate

fly()

Prediction

swim()

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

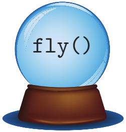
0

Branch Target Buffer (BTB)

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

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

fly() swim()

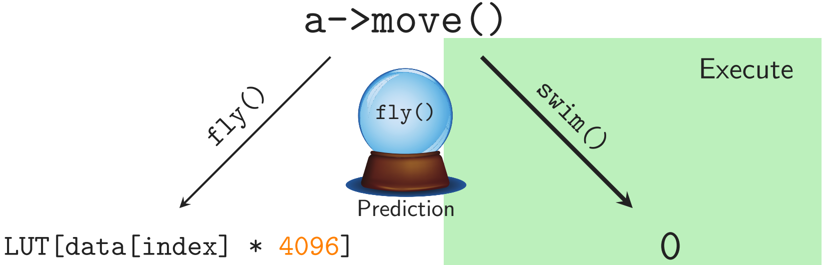


Prediction

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

0

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

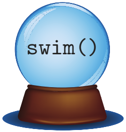


Branch Target Buffer (BTB)

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

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

fly() swim()

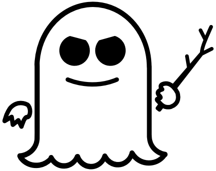


Prediction

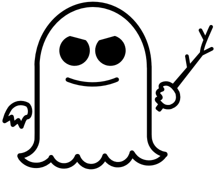
```
LUT[data[index] * 4096]
```

0

Naming

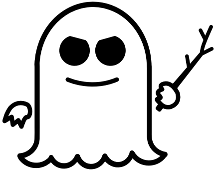


- Spectre Variant 2: Branch Target Injection



- Spectre Variant 2: Branch Target Injection
- Intel: Side-Channel Vulnerability 2

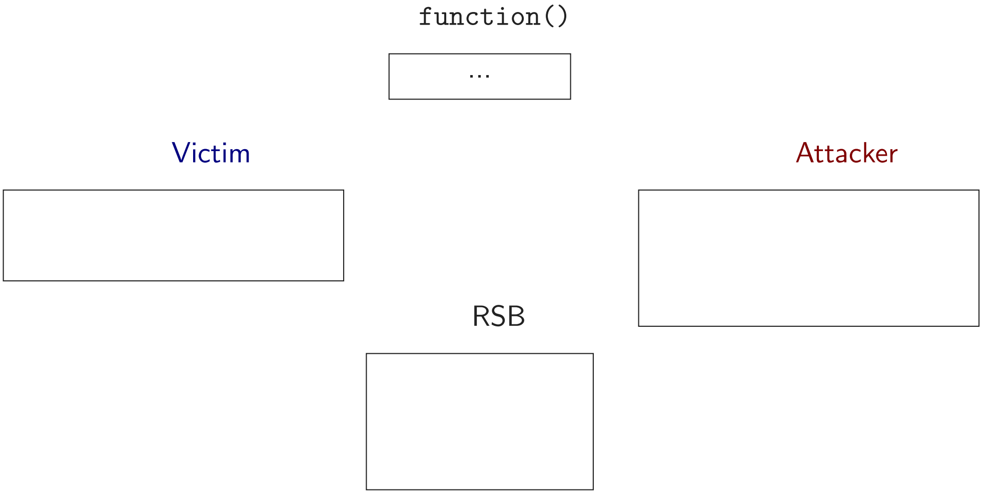
Naming



- Spectre Variant 2: Branch Target Injection
- Intel: Side-Channel Vulnerability 2
- Propose: Spectre-BTB

Return Stack Buffer (RSB)

Return Stack Buffer (RSB)



Return Stack Buffer (RSB)

`function()`

...

Victim

```
reg = secret
```

Attacker

```
reg = dummy
```

RSB

Return Stack Buffer (RSB)

function()

```
...
```

Victim

```
reg = secret
call function(SHORT)
```

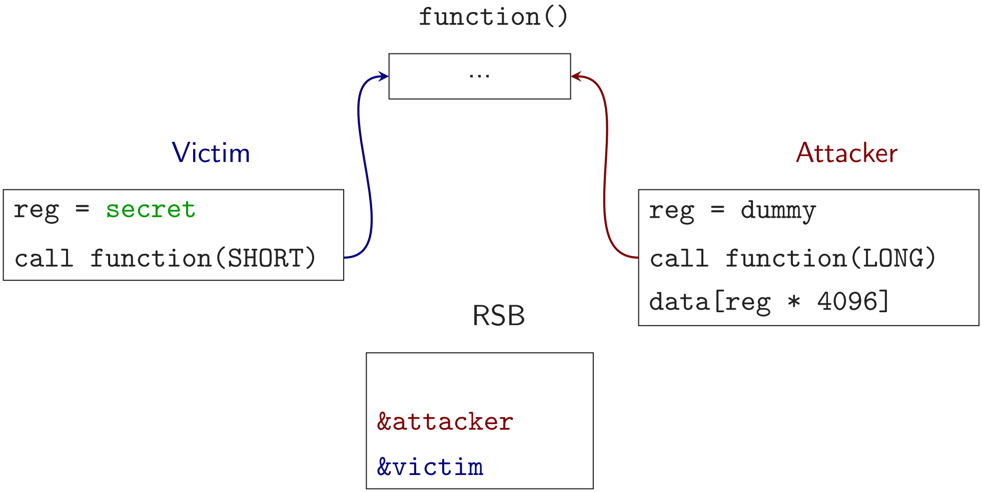
Attacker

```
reg = dummy
```

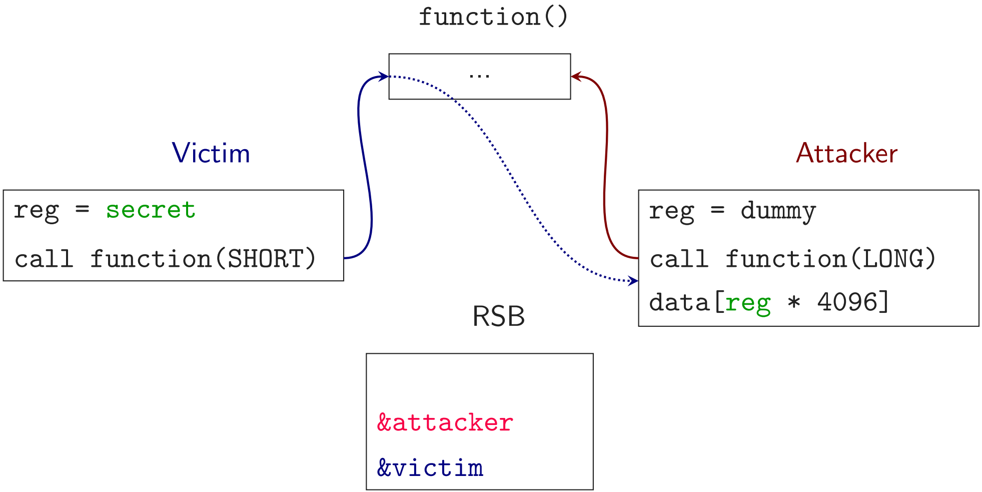
RSB

```
&victim
```

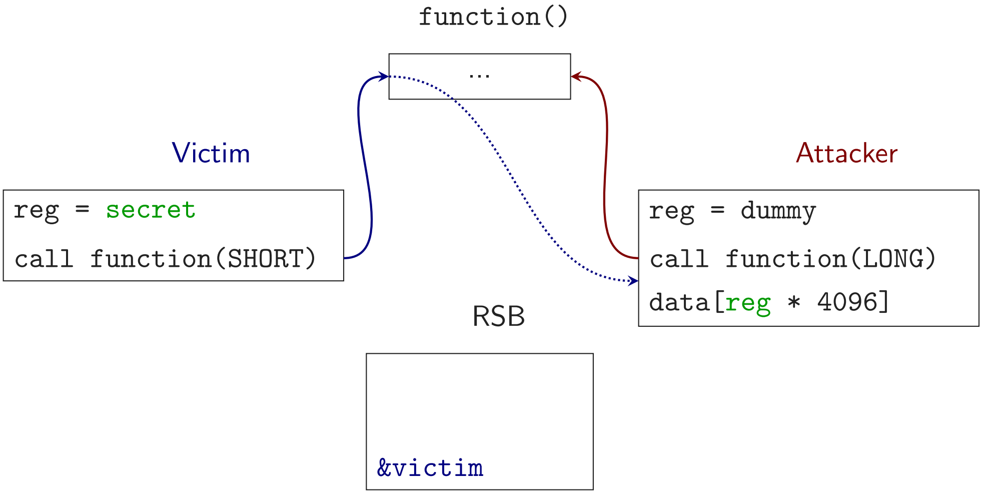
Return Stack Buffer (RSB)



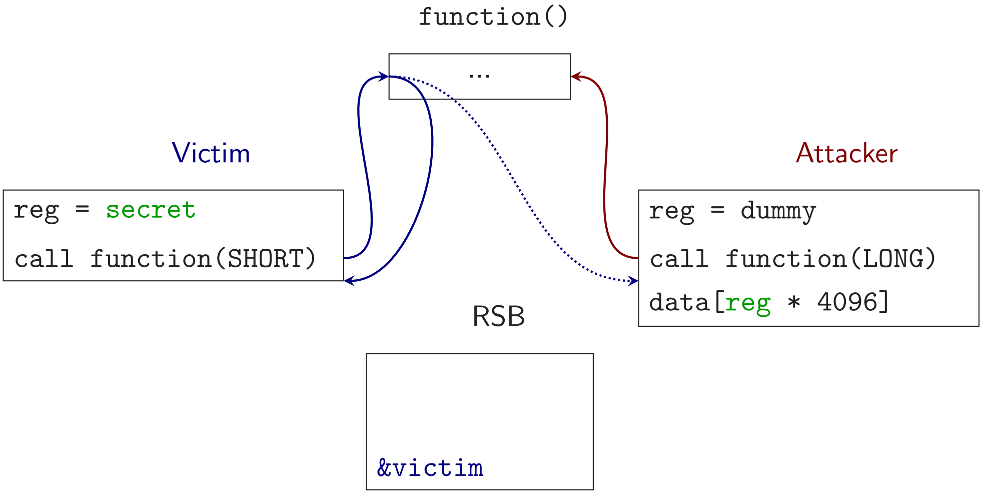
Return Stack Buffer (RSB)



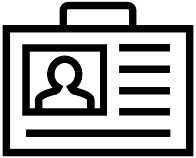
Return Stack Buffer (RSB)



Return Stack Buffer (RSB)

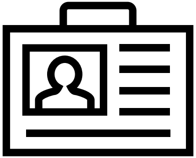


Naming

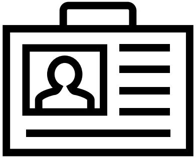


- Spectre-NG

Naming

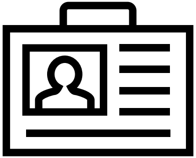


- Spectre-NG
- SpectreRSB

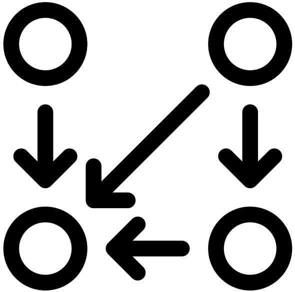


- Spectre-NG
- SpectreRSB
- ret2spec

Naming

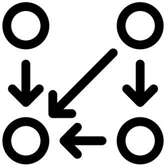


- Spectre-NG
- SpectreRSB
- ret2spec
- Propose: Spectre-RSB

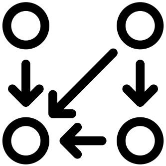


Memory Disambiguation (STL)

Memory Disambiguation (STL)

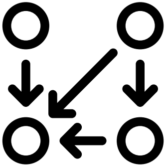


- Loads can be executed out-of-order



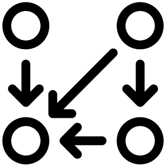
- Loads can be executed out-of-order $\rightarrow$ need to check for previous stores

Memory Disambiguation (STL)



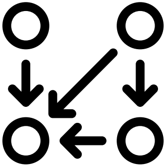
- Loads can be executed out-of-order → need to check for previous stores
- Check is time consuming

Memory Disambiguation (STL)



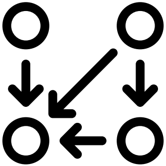
- Loads can be executed out-of-order $\rightarrow$ need to check for previous stores
- Check is time consuming
- Optimization: Speculate whether a store happened or not

Memory Disambiguation (STL)



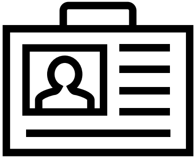
- Loads can be executed out-of-order $\rightarrow$ need to check for previous stores
- Check is time consuming
- Optimization: Speculate whether a store happened or not
  - no store: bypass check

Memory Disambiguation (STL)



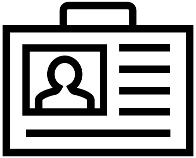
- Loads can be executed out-of-order $\rightarrow$ need to check for previous stores
- Check is time consuming
- Optimization: Speculate whether a store happened or not
    - no store: bypass check
    - stall

Naming

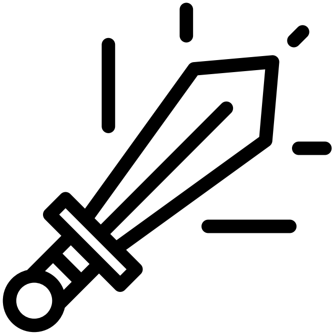


- Spectre Variant 4: Speculative Store Bypass

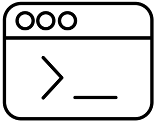
Naming



- Spectre Variant 4: Speculative Store Bypass
- Propose: Spectre-STL

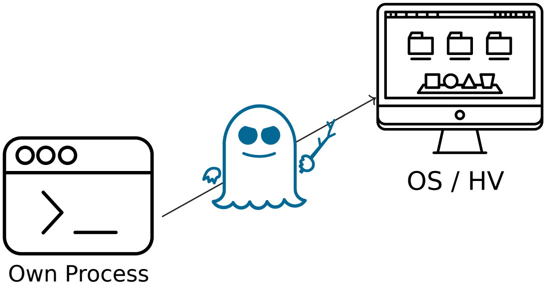


What can you attack?

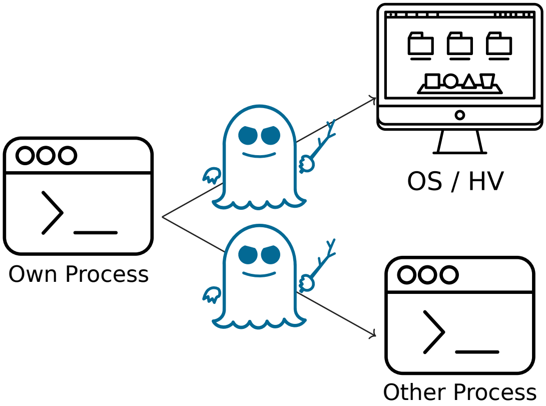


Own Process

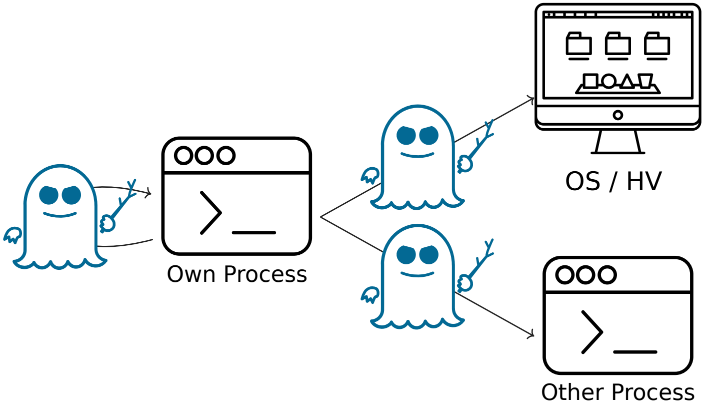
What can you attack?

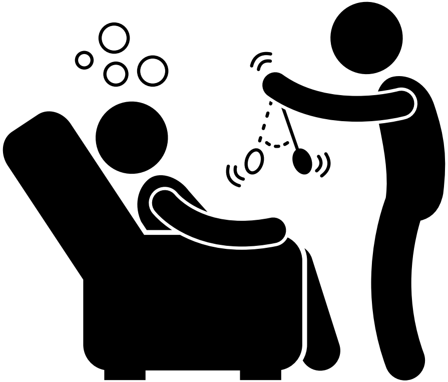


What can you attack?



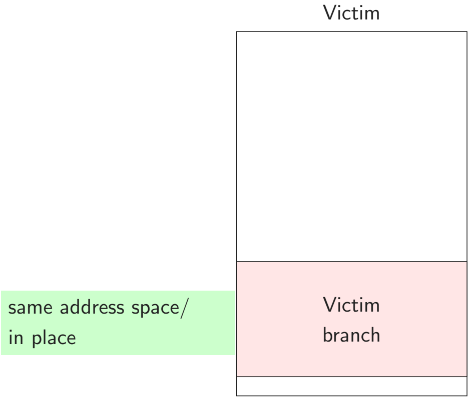
What can you attack?



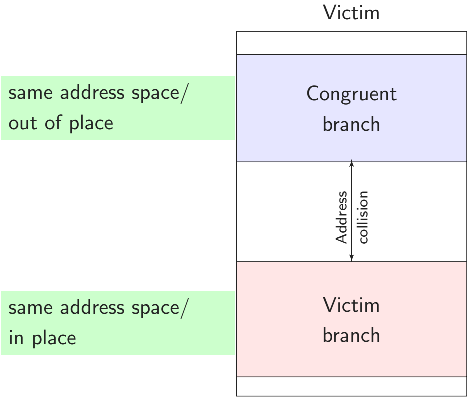


How can we mistrain the CPU?

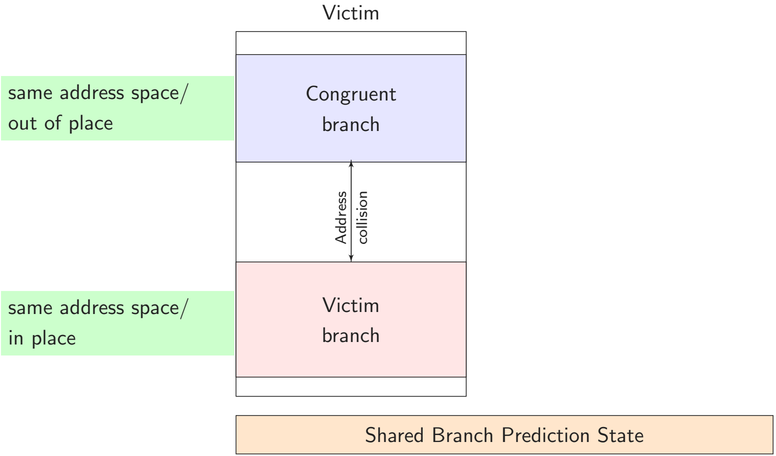
How can we mistrain?



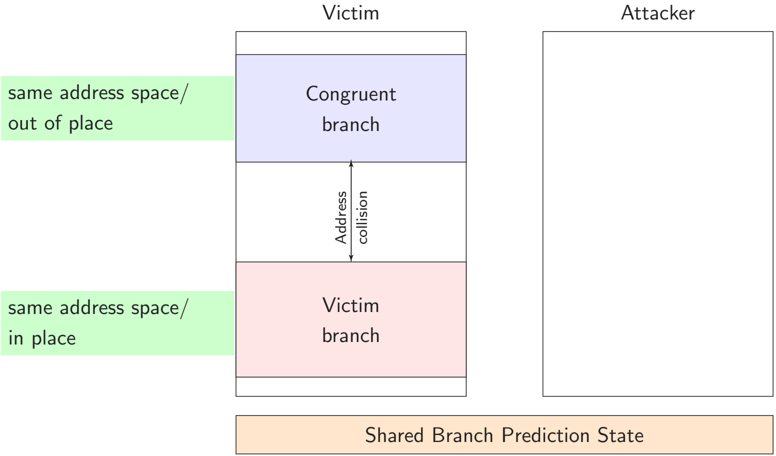
How can we mistrain?



How can we mistrain?

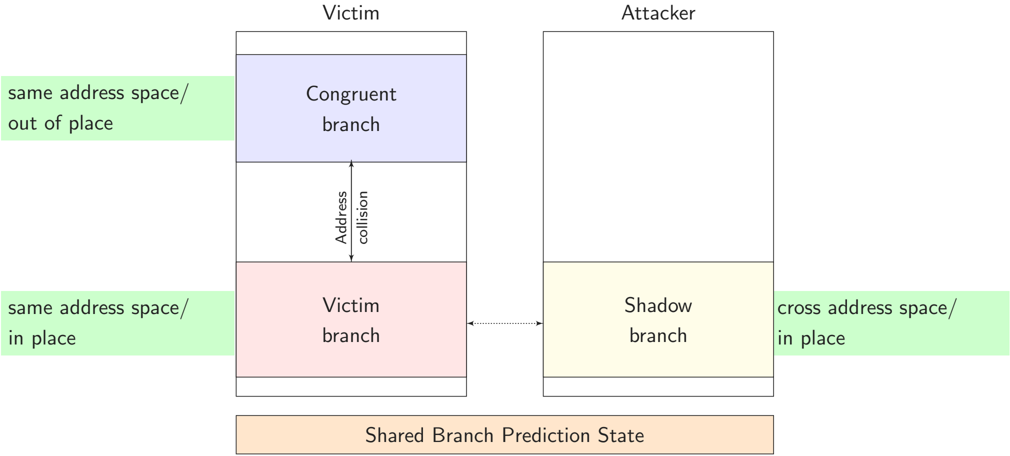


How can we mistrain?

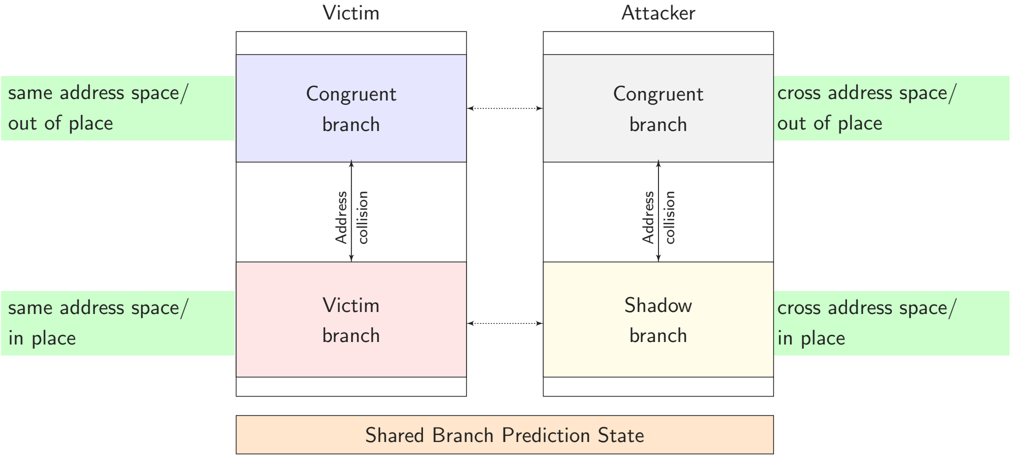


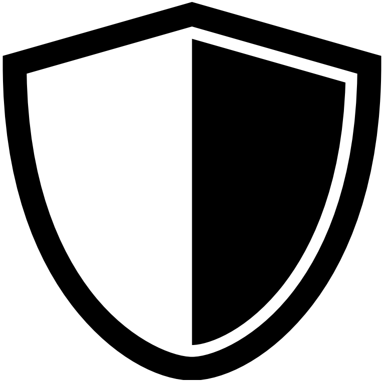
Moritz Lipp (@mlqxyz), Claudio Canella (@cc0x1f)

How can we mistrain?



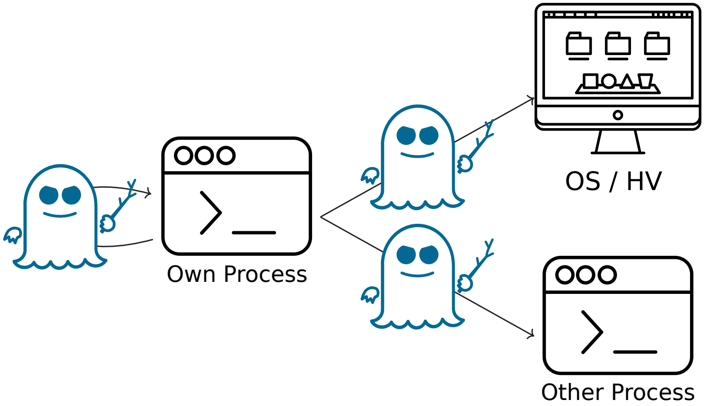
How can we mistrain?



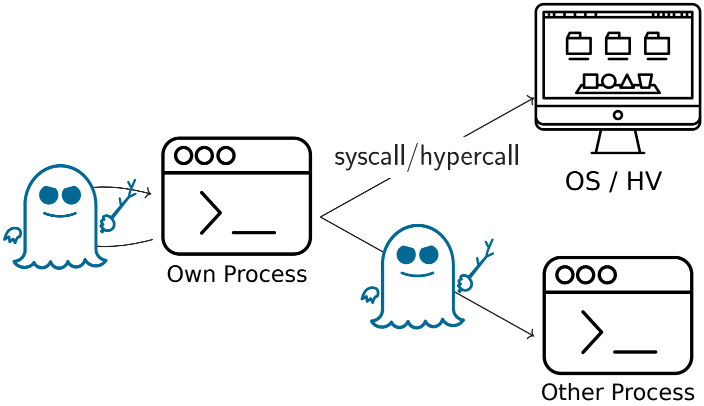


How can we fix this?

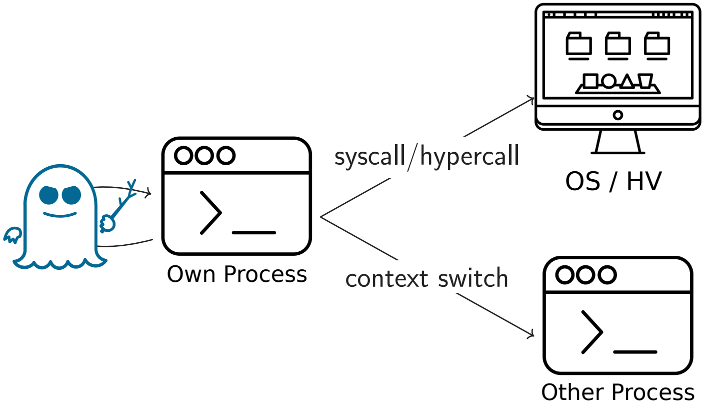
How can we fix this?



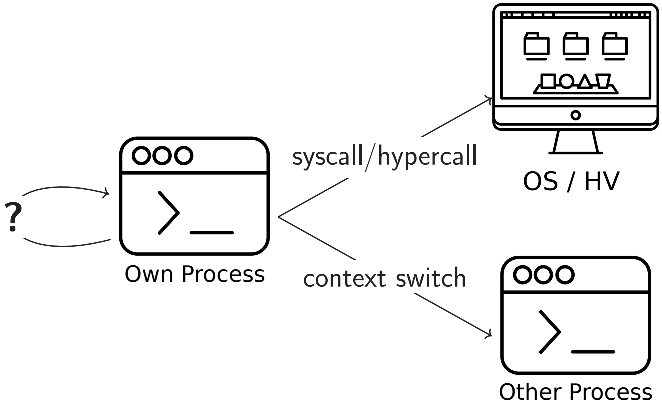
How can we fix this?



How can we fix this?



How can we fix this?

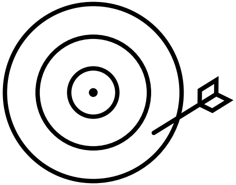


Spectre Defense Categorization

Spectre defenses in 3 categories:

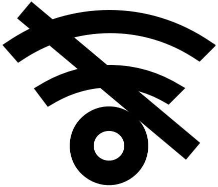
Spectre Defense Categorization

Spectre defenses in 3 categories:



C1 Mitigating or reducing the accuracy of covert channels

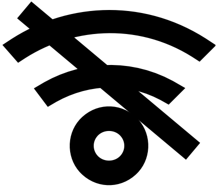
C1: Mitigating covert channels

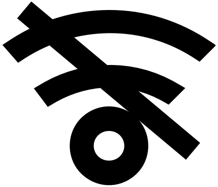


- **Deactivate the cache**

C1: Mitigating covert channels

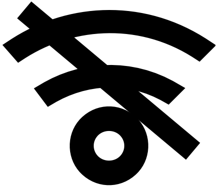
- **Deactivate the cache** → System would be too slow





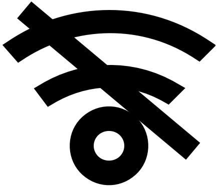
- **Deactivate the cache** → System would be too slow
- **Remove flush instructions**

C1: Mitigating covert channels



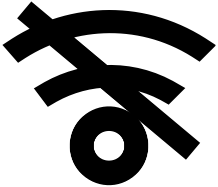
- **Deactivate the cache** → System would be too slow
- **Remove flush instructions** → Use eviction

C1: Mitigating covert channels



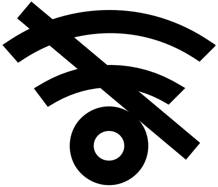
- **Deactivate the cache** → System would be too slow
- **Remove flush instructions** → Use eviction
- **Remove rdtsc**

C1: Mitigating covert channels



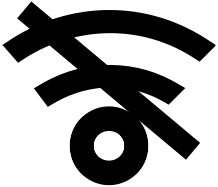
- **Deactivate the cache** → System would be too slow
- **Remove flush instructions** → Use eviction
- **Remove rdtsc** → Many alternative high-resolution timers

C1: Mitigating covert channels



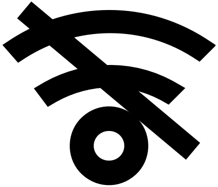
- **Deactivate the cache** → System would be too slow
- **Remove flush instructions** → Use eviction
- **Remove rdtsc** → Many alternative high-resolution timers
- **Build new caches**: DAWG, InvisiSpec, SafeSpec

C1: Mitigating covert channels



- **Deactivate the cache** → System would be too slow
- **Remove flush instructions** → Use eviction
- **Remove rdtsc** → Many alternative high-resolution timers
- **Build new caches**: DAWG, InvisiSpec, SafeSpec → Require hardware changes

C1: Mitigating covert channels

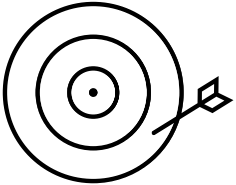


- **Deactivate the cache** → System would be too slow
- **Remove flush instructions** → Use eviction
- **Remove rdtsc** → Many alternative high-resolution timers
- **Build new caches**: DAWG, InvisiSpec, SafeSpec → Require hardware changes

- **Other covert channels** besides the cache:
    - PortSmash
    - SMoTHERSpectre
    - AVX

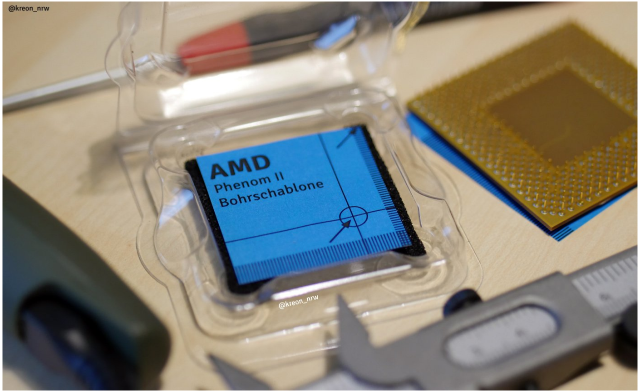
Spectre Defense Categorization

Spectre defenses in 3 categories:

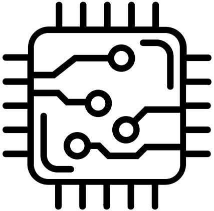


C1 Mitigating or reducing the accuracy of covert channels

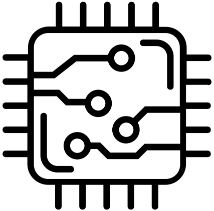
C2 Mitigating or aborting speculation



Drilling template (@kreon_nrw)

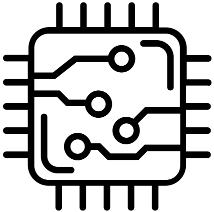


- Three new controls for ISA:
  - Indirect Branch Restricted Speculation (**IBRS**)



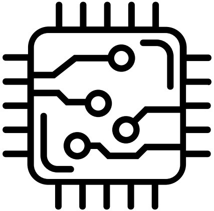
- Three new controls for ISA:
    - Indirect Branch Restricted Speculation (**IBRS**)
    - Single Thread Indirect Branch Prediction (**STIBP**)

C2: Mitigating or aborting speculation in hardware



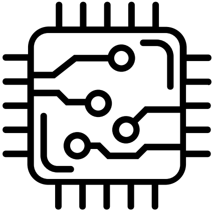
- Three new controls for ISA:
  - Indirect Branch Restricted Speculation (**IBRS**)
  - Single Thread Indirect Branch Prediction (**STIBP**)
  - Indirect Branch Predictor Barrier (**IBPB**)

C2: Mitigating or aborting speculation in hardware

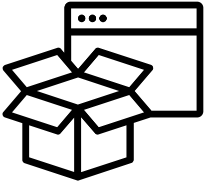


- Three new controls for ISA:
  - Indirect Branch Restricted Speculation (**IBRS**)
  - Single Thread Indirect Branch Prediction (**STIBP**)
  - Indirect Branch Predictor Barrier (**IBPB**)
  - ARMv8.5-A: New barrier (`sb`)

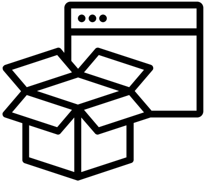
C2: Mitigating or aborting speculation in hardware



- Three new controls for ISA:
  - Indirect Branch Restricted Speculation (**IBRS**)
  - Single Thread Indirect Branch Prediction (**STIBP**)
  - Indirect Branch Predictor Barrier (**IBPB**)
  - ARMv8.5-A: New barrier (`sb`)
- Speculative Store Bypass Safe/Disable (**SSBS/SSBD**)

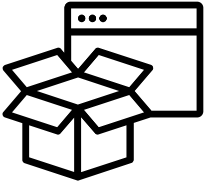


- Use **serializing instructions** on branch outcomes (`lfence`, `dsb sy`)

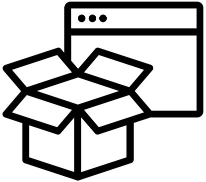


- Use **serializing instructions** on branch outcomes (`lfence`, `dsb sy`)
  - Developers need to know where to put them

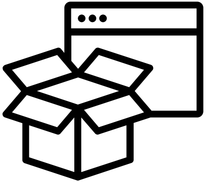
C2: Mitigating or aborting speculation in software



- Use **serializing instructions** on branch outcomes (`lfence`, `dsb sy`)
  - Developers need to know where to put them
- **Retpoline**: replacing indirect branches with return instructions



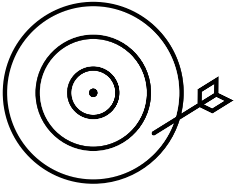
- Use **serializing instructions** on branch outcomes (`lfence`, `dsb sy`)
  - Developers need to know where to put them
- **Retpoline**: replacing indirect branches with return instructions
- **RSB Stuffing**



- Use **serializing instructions** on branch outcomes (`lfence`, `dsb sy`)
    - Developers need to know where to put them
- **Retpoline**: replacing indirect branches with return instructions
- **RSB Stuffing**
    - Fill RSB with benign addresses on every context switch

Spectre Defense Categorization

Spectre defenses in 3 categories:



C1 Mitigating or reducing the accuracy of covert channels

C2 Mitigating or aborting speculation

C3 Ensuring secret data cannot be reached

- What to do if an attacker attacks his own process (sandboxing)?

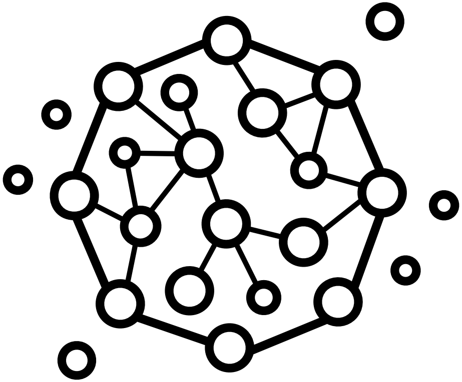
- What to do if an attacker attacks his own process (sandboxing)?
- Webkit: Index masking instead of array bounds checks

C3: Ensuring secret data cannot be reached

- What to do if an attacker attacks his own process (sandboxing)?
- Webkit: Index masking instead of array bounds checks
- Webkit: Poison values to protect pointers

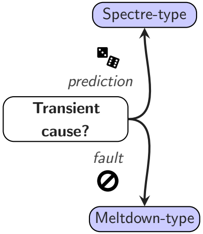
C3: Ensuring secret data cannot be reached

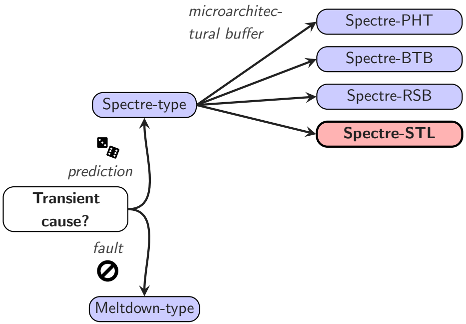
- What to do if an attacker attacks his own process (sandboxing)?
- Webkit: Index masking instead of array bounds checks
- Webkit: Poison values to protect pointers
- Chrome: Site Isolation

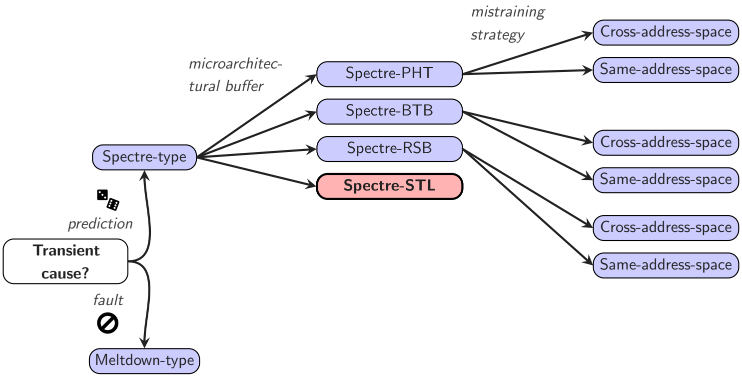


Systematization

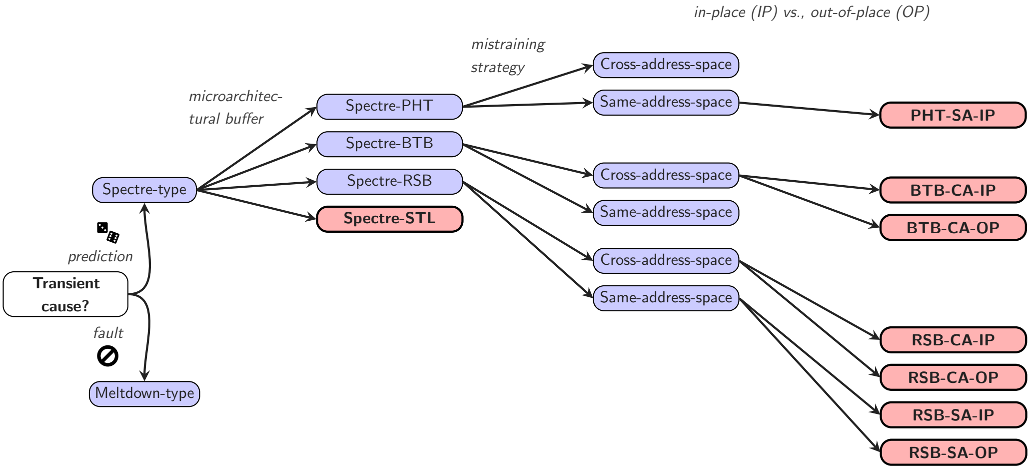
Transient cause?


Spectre-type
prediction
Transient cause?
fault
Meltdown-type

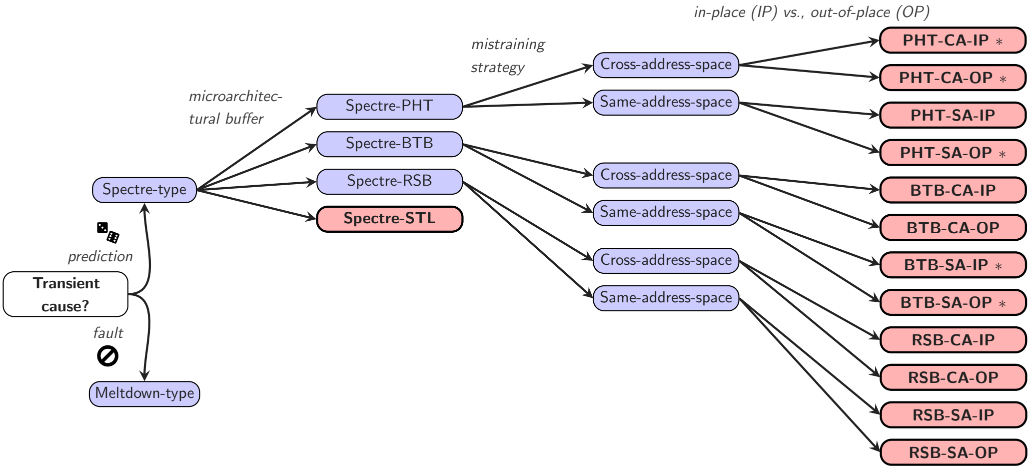

microarchitec-
tural buffer
Spectre-PHT
Spectre-BTB
Spectre-RSB
Spectre-STL
Spectre-type
prediction
Transient cause?
fault
Meltdown-type



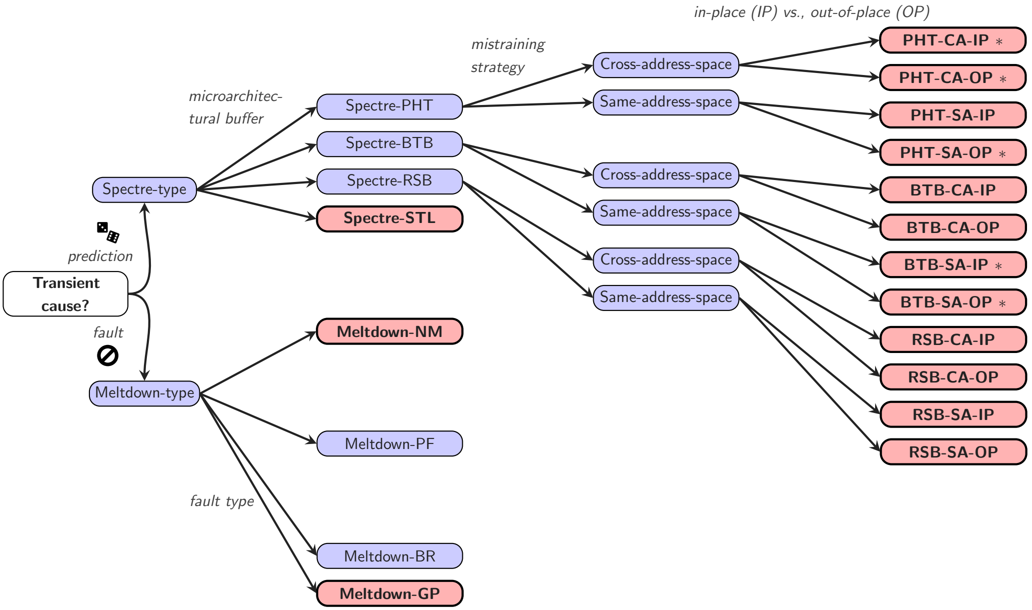
Transient cause?
prediction
fault
Spectre-type
Meltdown-type
microarchitec-tural buffer
Spectre-PHT
Spectre-BTB
Spectre-RSB
Spectre-STL
mistraining strategy
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space

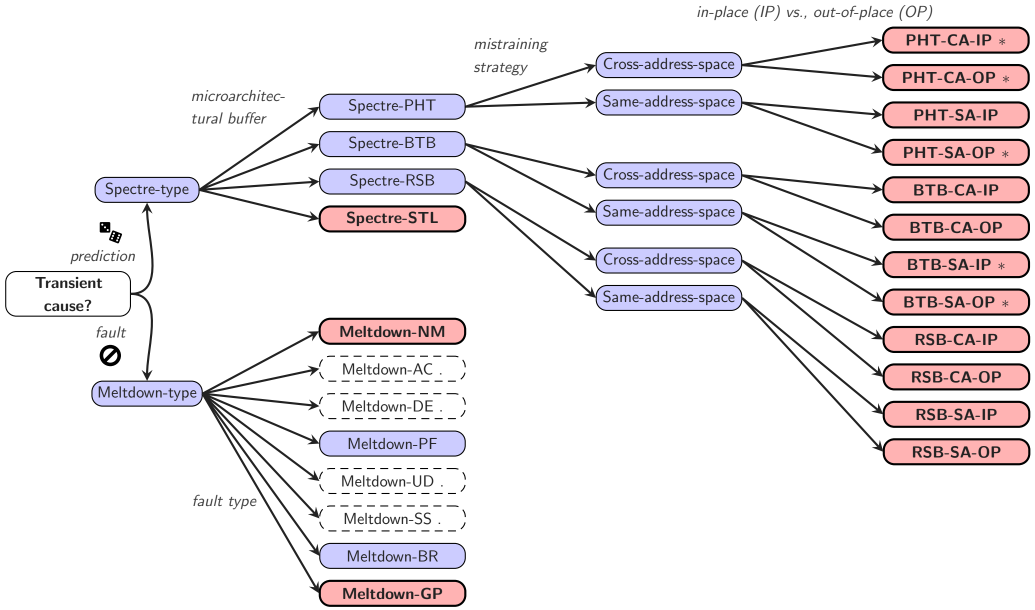


in-place (IP) vs., out-of-place (OP)
mistraining strategy
microarchitec-tural buffer
Transient cause?
prediction
fault
Spectre-type
Meltdown-type
Spectre-PHT
Spectre-BTB
Spectre-RSB
Spectre-STL
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
PHT-SA-IP
BTB-CA-IP
BTB-CA-OP
RSB-CA-IP
RSB-CA-OP
RSB-SA-IP
RSB-SA-OP

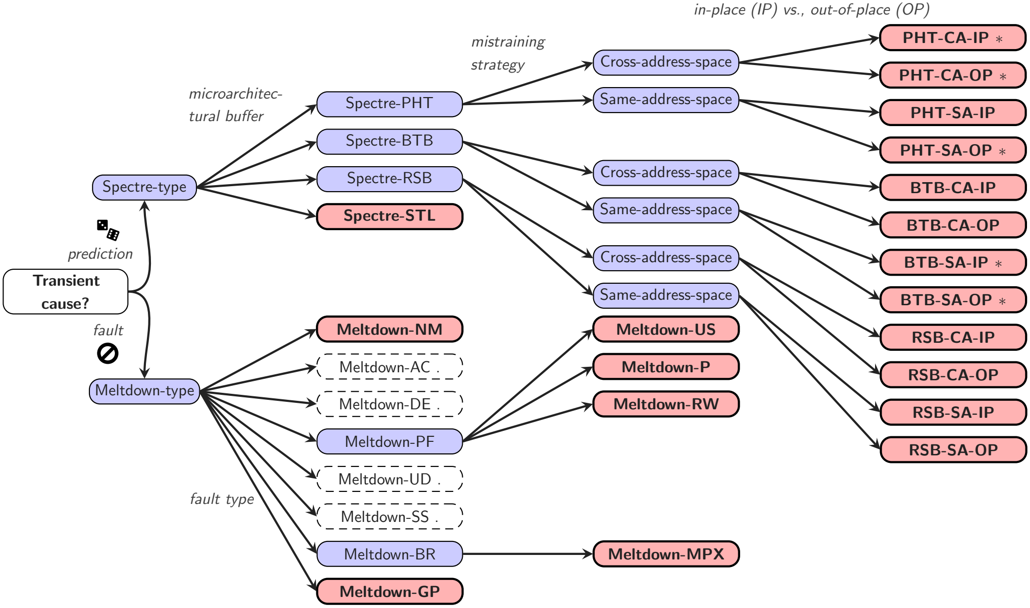


in-place (IP) vs., out-of-place (OP)
mistraining strategy
microarchitectural buffer
Transient cause?
prediction
fault
Spectre-type
Meltdown-type
Spectre-PHT
Spectre-BTB
Spectre-RSB
Spectre-STL
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
PHT-CA-IP ∗
PHT-CA-OP ∗
PHT-SA-IP
PHT-SA-OP ∗
BTB-CA-IP
BTB-CA-OP
BTB-SA-IP ∗
BTB-SA-OP ∗
RSB-CA-IP
RSB-CA-OP
RSB-SA-IP
RSB-SA-OP

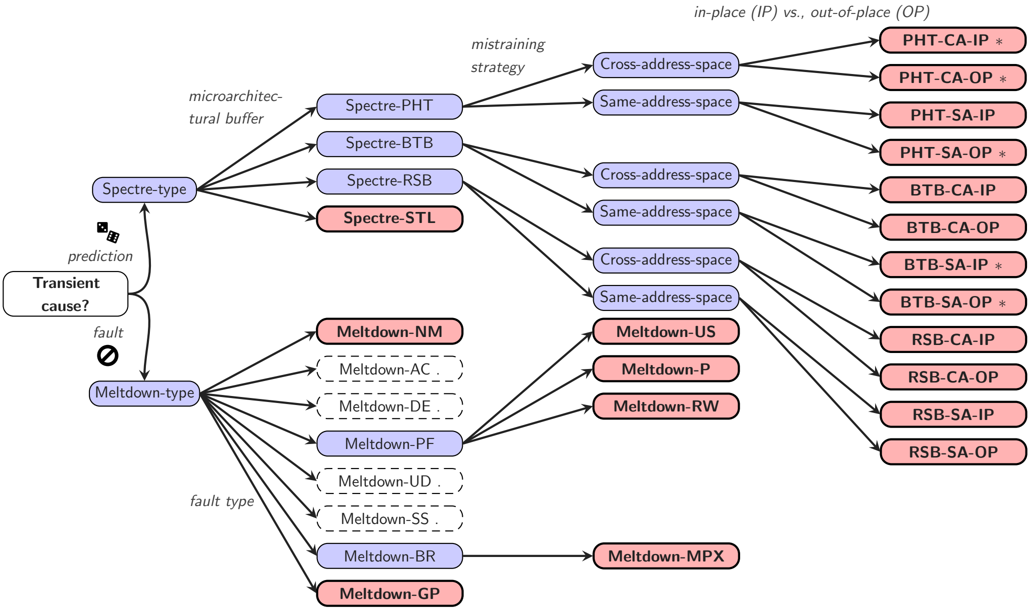

in-place (IP) vs., out-of-place (OP)
mistraining strategy
microarchitectural buffer
Transient cause?
prediction
fault
Spectre-type
Meltdown-type
Spectre-PHT
Spectre-BTB
Spectre-RSB
Spectre-STL
Meltdown-NM
Meltdown-PF
Meltdown-BR
Meltdown-GP
fault type
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
PHT-CA-IP *
PHT-CA-OP *
PHT-SA-IP
PHT-SA-OP *
BTB-CA-IP
BTB-CA-OP
BTB-SA-IP *
BTB-SA-OP *
RSB-CA-IP
RSB-CA-OP
RSB-SA-IP
RSB-SA-OP



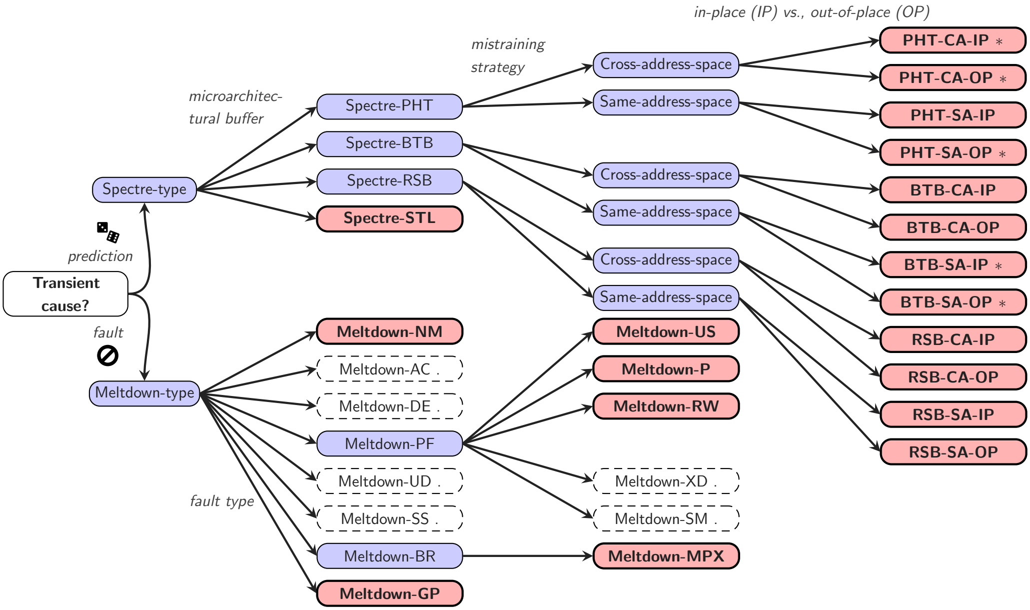
in-place (IP) vs., out-of-place (OP)
mistraining strategy
microarchitectural buffer
prediction
fault
fault type
Transient cause?
Spectre-type
Meltdown-type
Spectre-PHT
Spectre-BTB
Spectre-RSB
Spectre-STL
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
PHT-CA-IP *
PHT-CA-OP *
PHT-SA-IP
PHT-SA-OP *
BTB-CA-IP
BTB-CA-OP
BTB-SA-IP *
BTB-SA-OP *
RSB-CA-IP
RSB-CA-OP
RSB-SA-IP
RSB-SA-OP
Meltdown-NM
Meltdown-AC .
Meltdown-DE .
Meltdown-PF
Meltdown-UD .
Meltdown-SS .
Meltdown-BR
Meltdown-GP



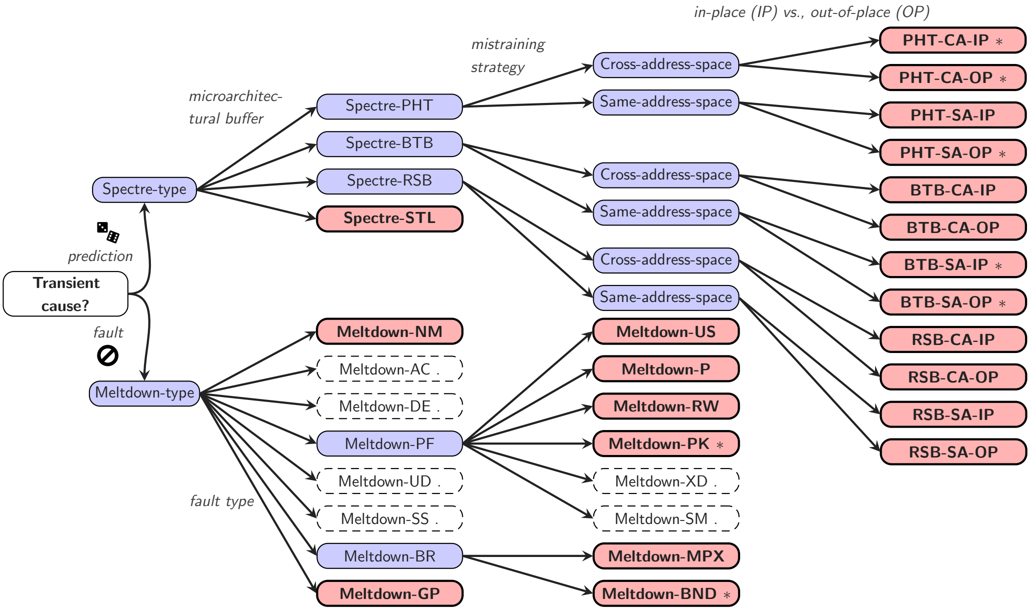
in-place (IP) vs., out-of-place (OP)
mistraining strategy
microarchitectural buffer
prediction
fault
fault type
Transient cause?
Spectre-type
Meltdown-type
Spectre-PHT
Spectre-BTB
Spectre-RSB
Spectre-STL
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
PHT-CA-IP ∗
PHT-CA-OP ∗
PHT-SA-IP
PHT-SA-OP ∗
BTB-CA-IP
BTB-CA-OP
BTB-SA-IP ∗
BTB-SA-OP ∗
RSB-CA-IP
RSB-CA-OP
RSB-SA-IP
RSB-SA-OP
Meltdown-NM
Meltdown-AC .
Meltdown-DE .
Meltdown-PF
Meltdown-UD .
Meltdown-SS .
Meltdown-BR
Meltdown-GP
Meltdown-US
Meltdown-P
Meltdown-RW
Meltdown-MPX



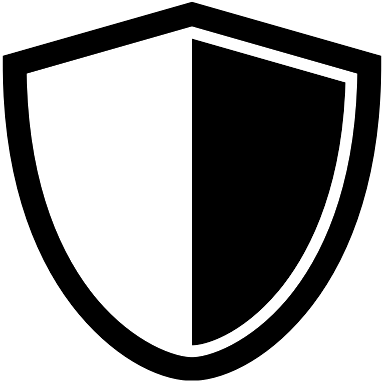
in-place (IP) vs., out-of-place (OP)
mistraining strategy
microarchitectural buffer
Transient cause?
prediction
fault
fault type
Spectre-type
Meltdown-type
Spectre-PHT
Spectre-BTB
Spectre-RSB
Spectre-STL
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
PHT-CA-IP ∗
PHT-CA-OP ∗
PHT-SA-IP
PHT-SA-OP ∗
BTB-CA-IP
BTB-CA-OP
BTB-SA-IP ∗
BTB-SA-OP ∗
RSB-CA-IP
RSB-CA-OP
RSB-SA-IP
RSB-SA-OP
Meltdown-NM
Meltdown-AC .
Meltdown-DE .
Meltdown-PF
Meltdown-UD .
Meltdown-SS .
Meltdown-BR
Meltdown-GP
Meltdown-US
Meltdown-P
Meltdown-RW
Meltdown-MPX



Transient cause?
prediction
fault
Spectre-type
Meltdown-type
microarchitectural buffer
Spectre-PHT
Spectre-BTB
Spectre-RSB
Spectre-STL
mistraining strategy
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
in-place (IP) vs., out-of-place (OP)
PHT-CA-IP ∗
PHT-CA-OP ∗
PHT-SA-IP
PHT-SA-OP ∗
BTB-CA-IP
BTB-CA-OP
BTB-SA-IP ∗
BTB-SA-OP ∗
RSB-CA-IP
RSB-CA-OP
RSB-SA-IP
RSB-SA-OP
fault type
Meltdown-NM
Meltdown-AC .
Meltdown-DE .
Meltdown-PF
Meltdown-UD .
Meltdown-SS .
Meltdown-BR
Meltdown-GP
Meltdown-US
Meltdown-P
Meltdown-RW
Meltdown-XD .
Meltdown-SM .
Meltdown-MPX

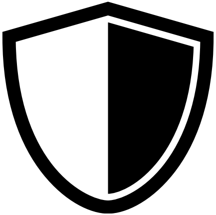


in-place (IP) vs., out-of-place (OP)
mistraining strategy
microarchitectural buffer
prediction
fault
fault type
Transient cause?
Spectre-type
Meltdown-type
Spectre-PHT
Spectre-BTB
Spectre-RSB
Spectre-STL
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
Cross-address-space
Same-address-space
PHT-CA-IP ∗
PHT-CA-OP ∗
PHT-SA-IP
PHT-SA-OP ∗
BTB-CA-IP
BTB-CA-OP
BTB-SA-IP ∗
BTB-SA-OP ∗
RSB-CA-IP
RSB-CA-OP
RSB-SA-IP
RSB-SA-OP
Meltdown-NM
Meltdown-AC .
Meltdown-DE .
Meltdown-PF
Meltdown-UD .
Meltdown-SS .
Meltdown-BR
Meltdown-GP
Meltdown-US
Meltdown-P
Meltdown-RW
Meltdown-PK ∗
Meltdown-XD .
Meltdown-SM .
Meltdown-MPX
Meltdown-BND ∗



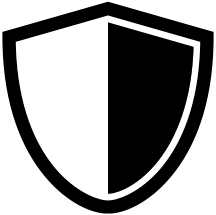
Are the defenses working?

Defenses working?



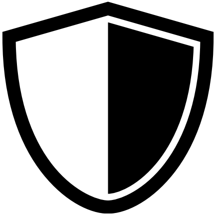
- Evaluated all mitigations without hardware changes
  - **Serialization:** still leaks through TLB, AVX

Defenses working?



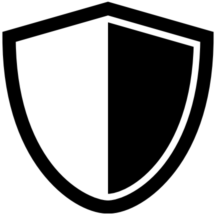
- Evaluated all mitigations without hardware changes
  - **Serialization:** still leaks through TLB, AVX
  - **STIPB, IBPB, SSBD:** not enabled by default

Defenses working?



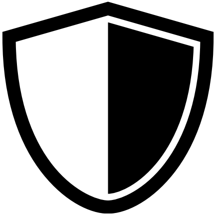
- Evaluated all mitigations without hardware changes
  - **Serialization:** still leaks through TLB, AVX
  - **STIPB, IBPB, SSBD:** not enabled by default
  - **Site Isolation:** can still leak data of own process

Defenses working?



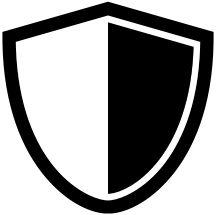
- Evaluated all mitigations without hardware changes
  - **Serialization:** still leaks through TLB, AVX
  - **STIPB, IBPB, SSBD:** not enabled by default
  - **Site Isolation:** can still leak data of own process
  - **Index Masking:** only limits how far beyond leakage can occur

Defenses working?



- Evaluated all mitigations without hardware changes
  - **Serialization:** still leaks through TLB, AVX
  - **STIPB, IBPB, SSBD:** not enabled by default
  - **Site Isolation:** can still leak data of own process
  - **Index Masking:** only limits how far beyond leakage can occur
  - **Speculative Load Hardening:** Spectrum showed potential leakage

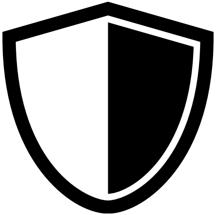
Defenses working?



- Evaluated all mitigations without hardware changes
  - **Serialization:** still leaks through TLB, AVX
  - **STIPB, IBPB, SSBD:** not enabled by default
  - **Site Isolation:** can still leak data of own process
  - **Index Masking:** only limits how far beyond leakage can occur
  - **Speculative Load Hardening:** Spectrum showed potential leakage
  - **Timer reduction:** can build new timers

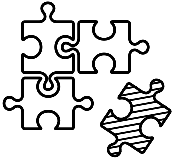
- Evaluated all mitigations without hardware changes
  - **Serialization:** still leaks through TLB, AVX
  - **STIPB, IBPB, SSBD:** not enabled by default
  - **Site Isolation:** can still leak data of own process
  - **Index Masking:** only limits how far beyond leakage can occur
  - **Speculative Load Hardening:** Spectrum showed potential leakage
  - **Timer reduction:** can build new timers
  - **Poison Value:** works

Defenses working?

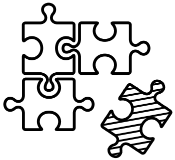


- Evaluated all mitigations without hardware changes
  - **Serialization:** still leaks through TLB, AVX
  - **STIPB, IBPB, SSBD:** not enabled by default
  - **Site Isolation:** can still leak data of own process
  - **Index Masking:** only limits how far beyond leakage can occur
  - **Speculative Load Hardening:** Spectrum showed potential leakage
  - **Timer reduction:** can build new timers
  - **Poison Value:** works
- Meltdown defenses

Conclusion

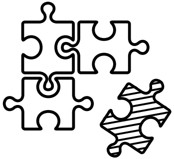


- Optimizations always come at a cost



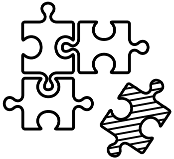
- Optimizations always come at a cost
- Some mitigations cost more than gained by the feature they defend

Conclusion



- Optimizations always come at a cost
- Some mitigations cost more than gained by the feature they defend
- Transient-execution attacks will keep us busy for a while

Conclusion



- Optimizations always come at a cost
- Some mitigations cost more than gained by the feature they defend
- Transient-execution attacks will keep us busy for a while
- There are still many other microarchitectural elements left to explore