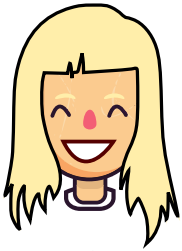
Transient Execution Attacks: Still hARMful?

Barbara Gigerl (@barbarag2112), Claudio Canella (@cc0x1f)

May 17, 2019

Graz University of Technology

Who am I?



Barbara Gigerl

Master student @ Graz University of Technology

@barbarag2112

barbara.gigerl@student.tugraz.at

Who am I?

Claudio Canella

PhD student @ Graz University of Technology

@cc0x1f

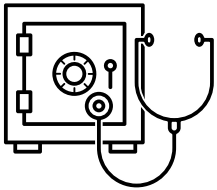
claudio.canella@iaik.tugraz.at


WASHINGTON, D.C.
FOX
BUSINESS
WASHINGTON, D.C.
WINTER STORM
NEWS
ALERT
INTEL REVEALS DESIGN FLAW THAT
COULD ALLOW HACKERS TO ACCESS DATA
FOX
BUSINESS
NETWORK
@FOXBUSINESS

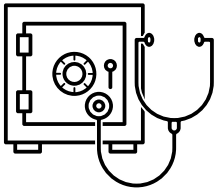

FIREEYE CYBER
THREAT MAP
DEVELOPING STORY
COMPUTER CHIP FLAWS IMPACT BILLIONS OF DEVICES
LIVE
CNN
DAX ▲164.69
NEWS STREAM

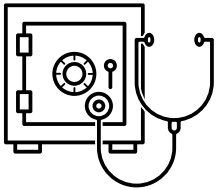

GLOBAL
COMPUTER CHIP SCARE
The bugs are known as 'Spectre' and 'Meltdown'
BBC WORLD NEWS
£:HK$ 10.58
EURO:£ 0.891

- Bug-free software does not mean safe execution

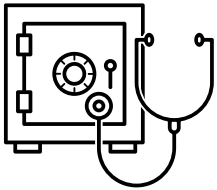


- Bug-free software does not mean safe execution
- Information leaks due to underlying hardware





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

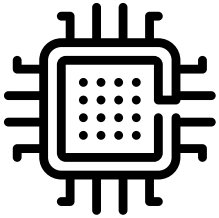


- Bug-free software does not mean safe execution
- Information leaks due to underlying hardware
- Exploit leakage through side-effects

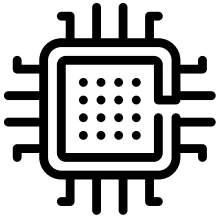
Power consumption

Execution time

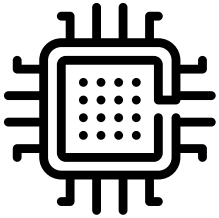
CPU caches



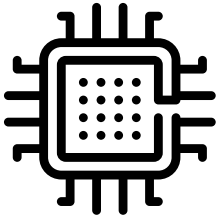
- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)



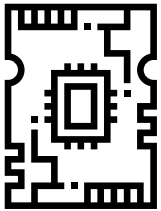
- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)
- Interface between hardware and software



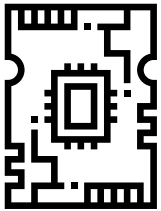
- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)
- Interface between hardware and software
- Microarchitecture is an ISA implementation



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



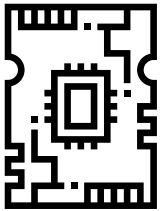
- Modern CPUs contain multiple microarchitectural elements



- Modern CPUs contain multiple microarchitectural elements

Caches and buffers

Predictors

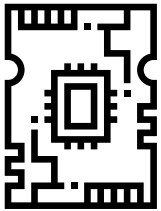


- Modern CPUs contain multiple microarchitectural elements

Caches and buffers

Predictors

- Transparent for the programmer



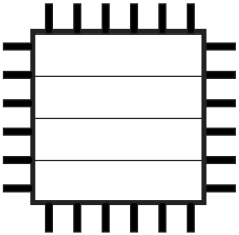
- Modern CPUs contain multiple microarchitectural elements

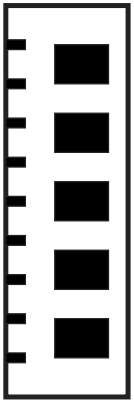
Caches and buffers

Predictors

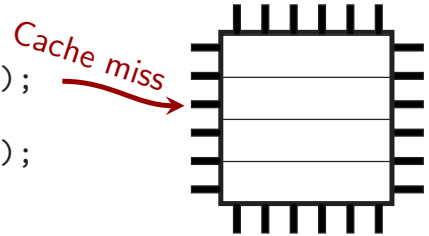
- Transparent for the programmer
- Timing optimizations → side-channel leakage

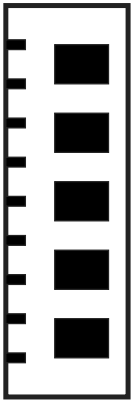
```
printf("%d", i);

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



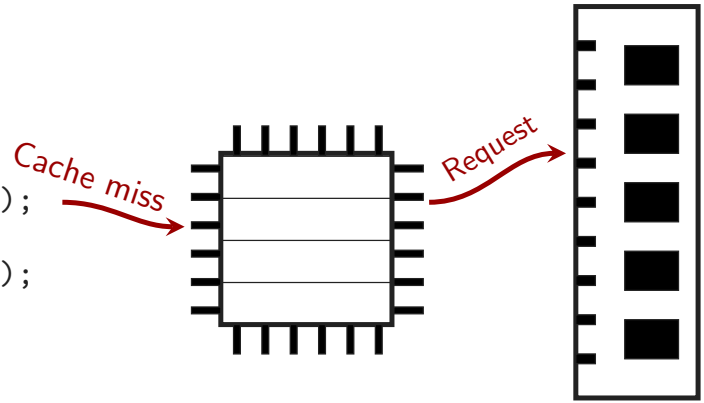


```
printf("%d", i);

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

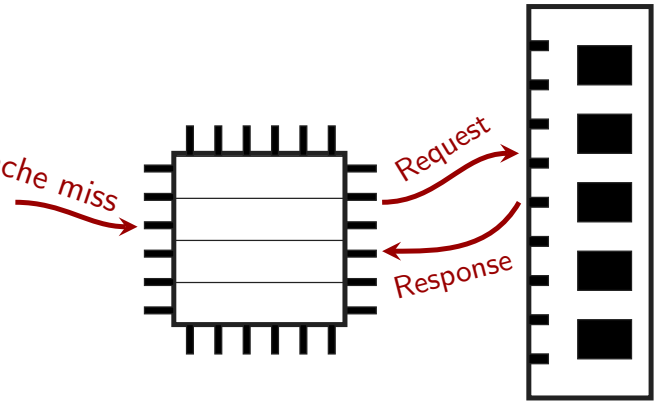


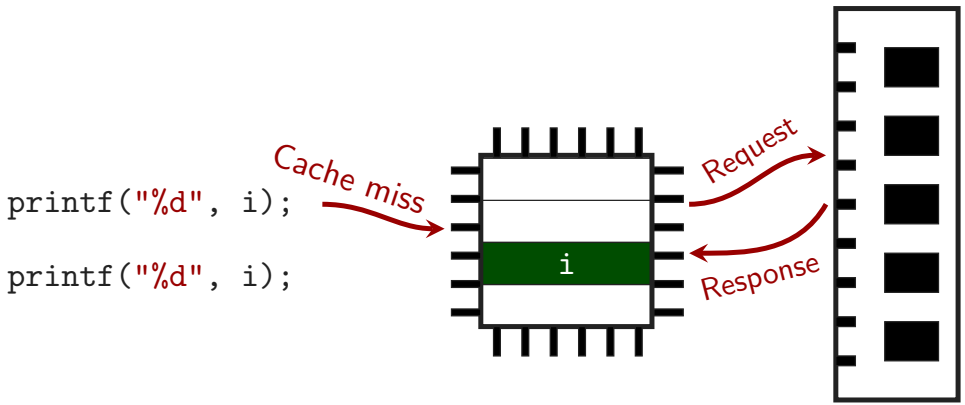


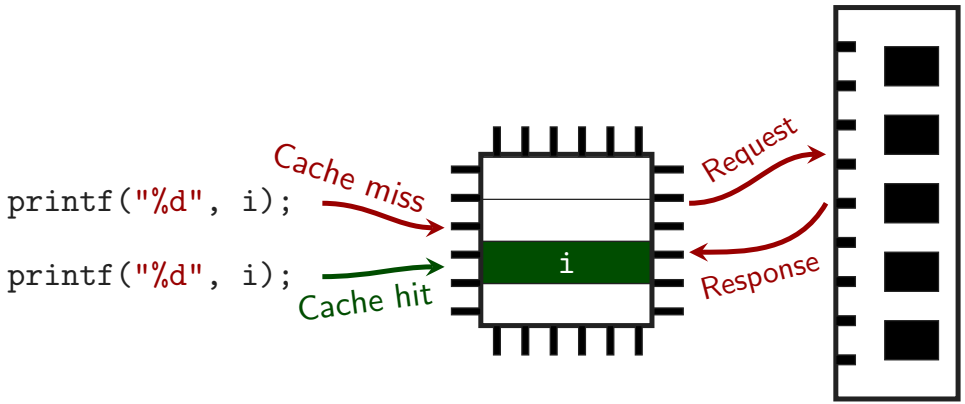
```
printf("%d", i);

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

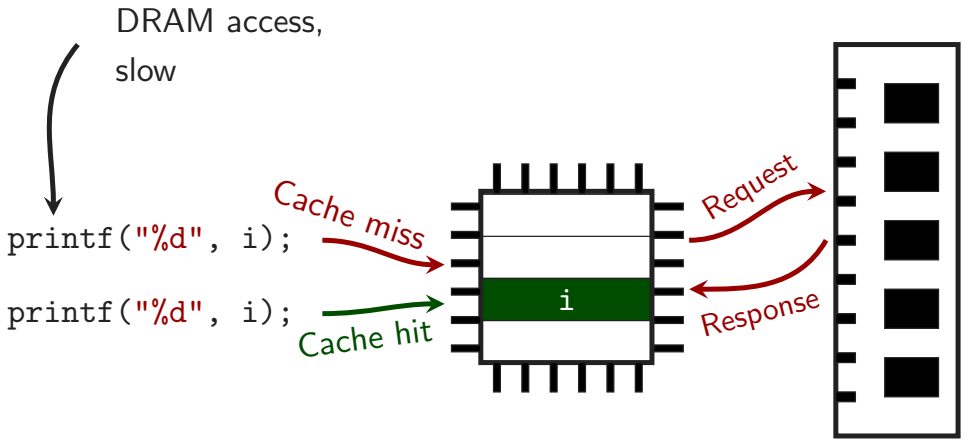


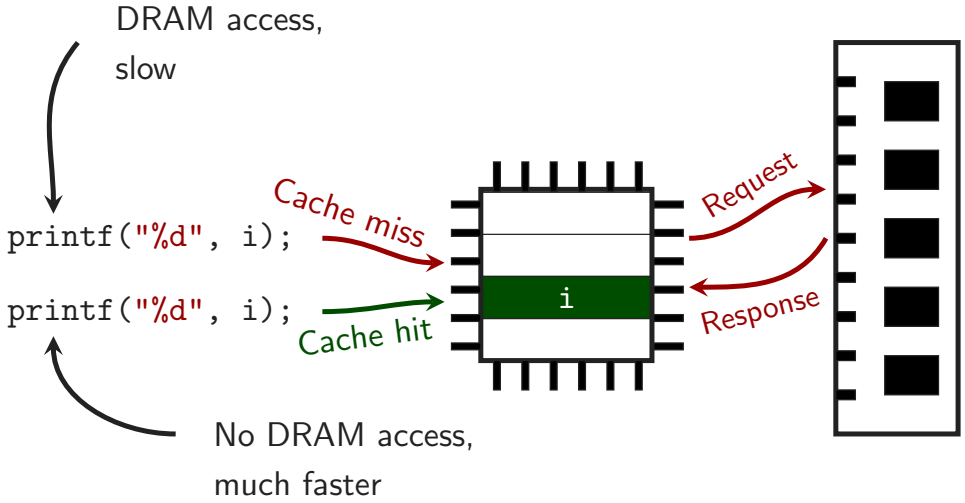
```
printf("%d", i);

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



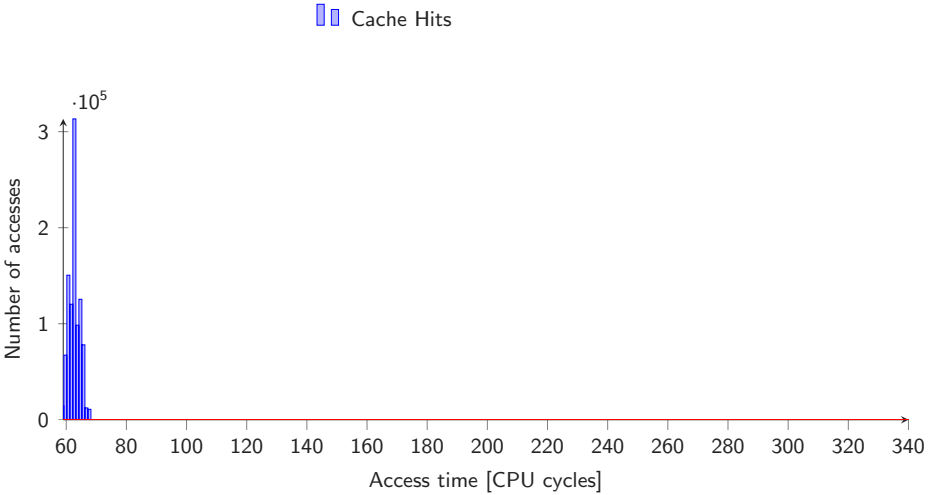

printf("%d", i);
printf("%d", i);
Cache miss
i
Request
Response


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

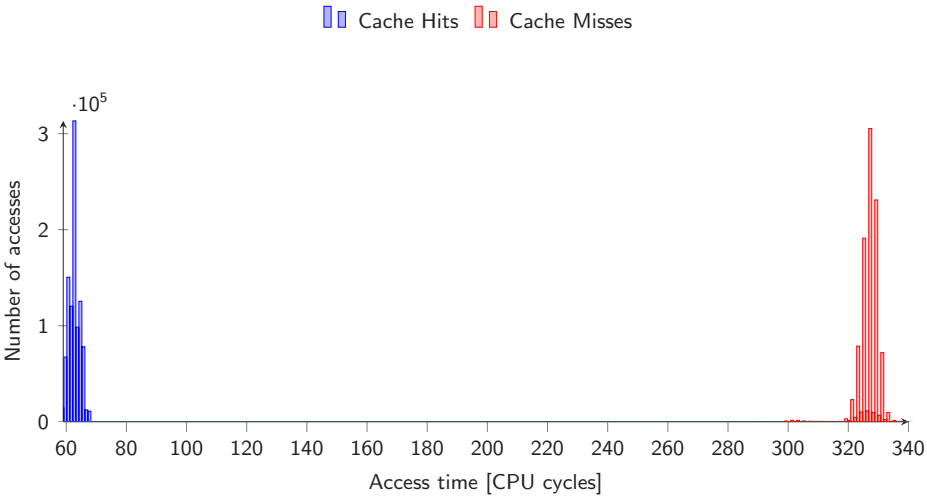

DRAM access,
slow
printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response


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

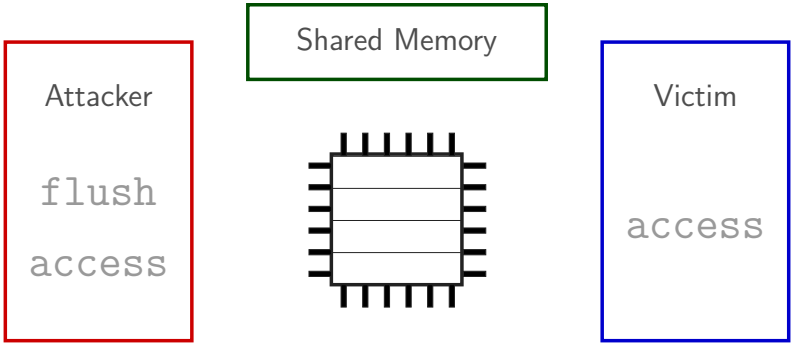
Caching speeds up Memory Accesses

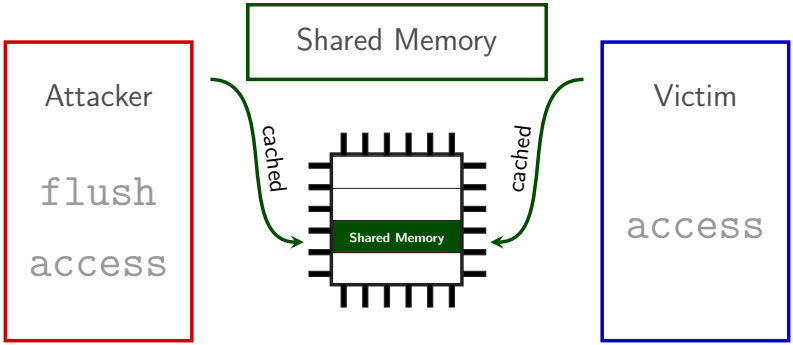


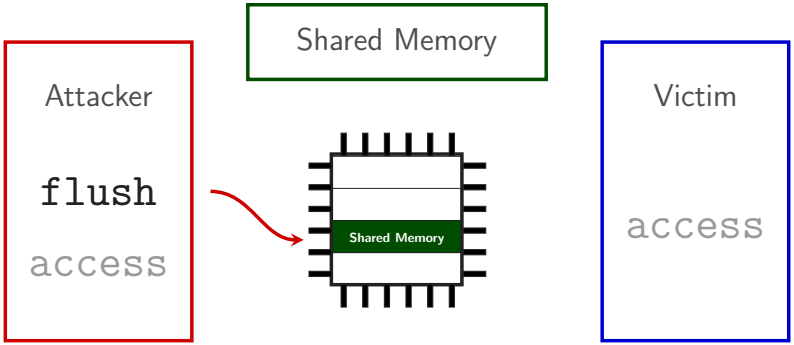
Caching speeds up Memory Accesses

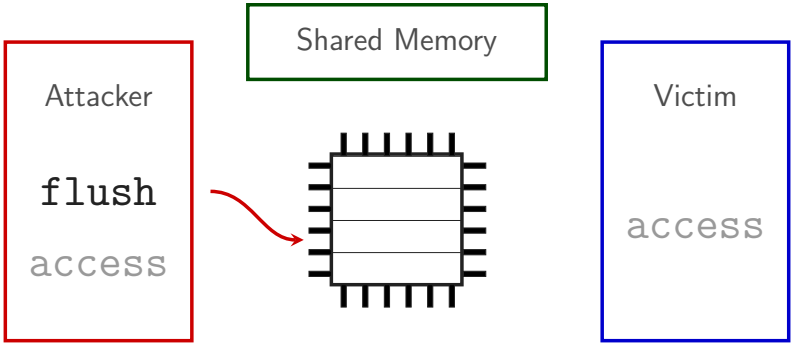


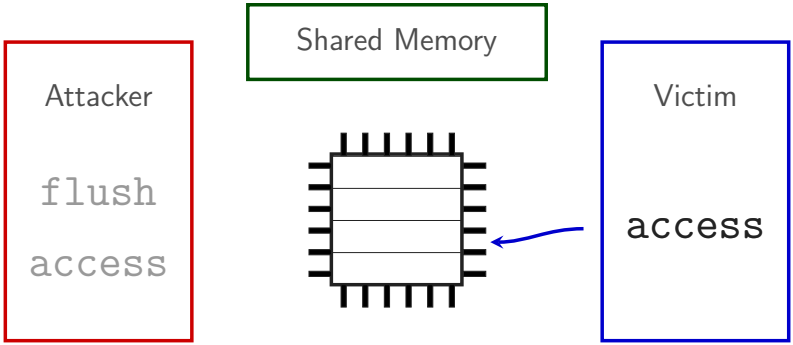
Barbara Gigerl (@barbarag2112), Claudio Canella (@cc0x1f) — Graz University of Technology

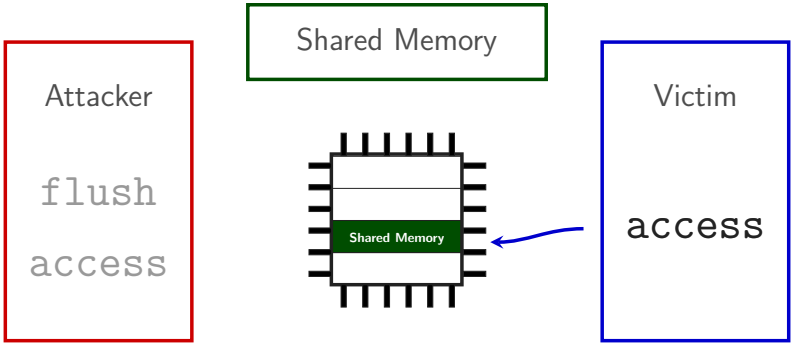

Shared Memory
Attacker
flush
access
Victim
access

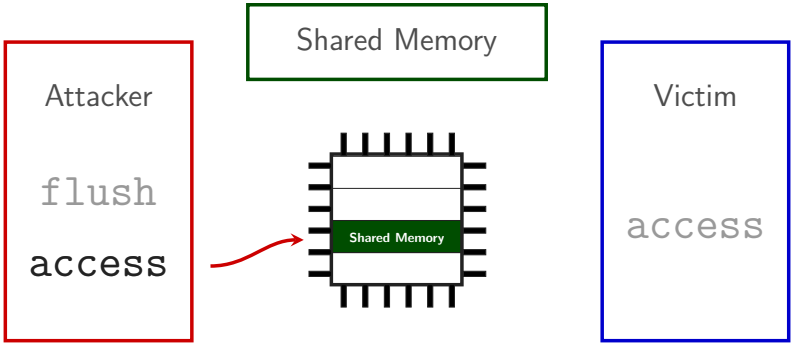

Shared Memory
Attacker
flush
access
cached
cached
Shared Memory
Victim
access

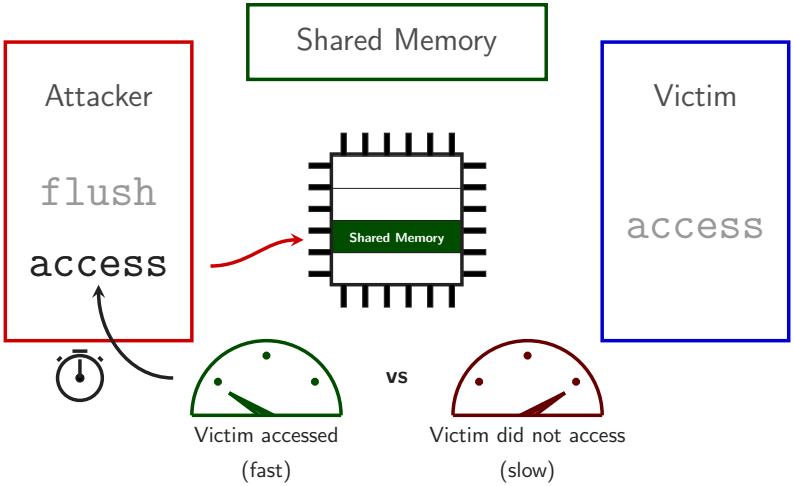

Shared Memory
Attacker
flush
access
Shared Memory
Victim
access


Shared Memory
Attacker
flush
access
Victim
access


Shared Memory
Attacker
flush
access
Victim
access


Shared Memory
Attacker
flush
access
Shared Memory
Victim
access


Shared Memory
Attacker
flush
access
Shared Memory
Victim
access


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

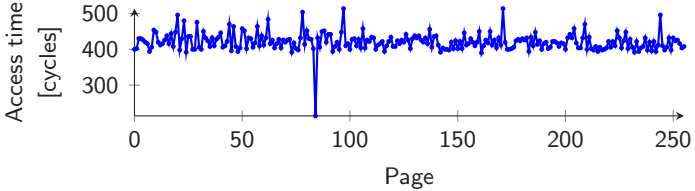
Toy example

```
char array[256 * 4096]; // 256 pages of memory
```

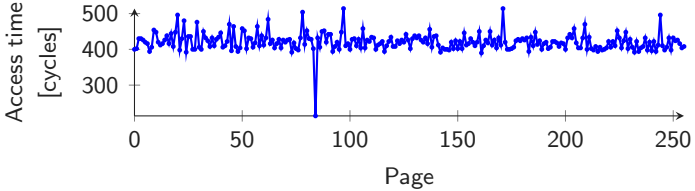
Toy example

```
char array[256 * 4096]; // 256 pages of memory

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

- Flush+Reload over all pages of the array

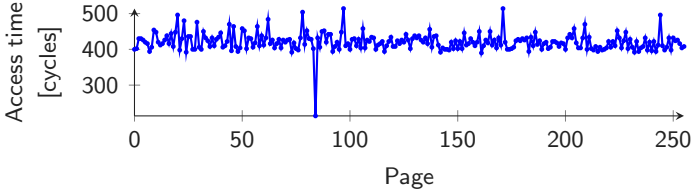


- Flush+Reload over all pages of the array



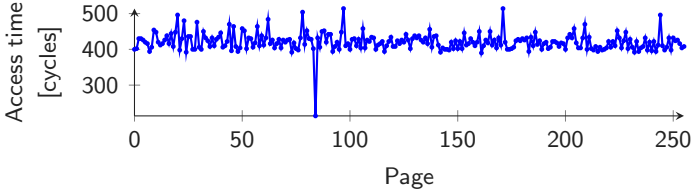
- "Unreachable" code line was actually executed

- Flush+Reload over all pages of the array



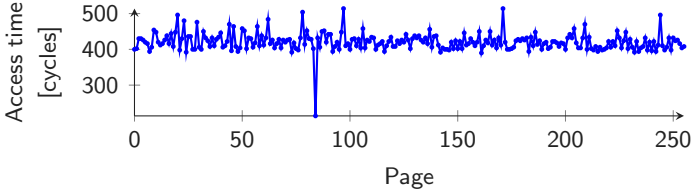
- "Unreachable" code line was actually executed
- Exception was only thrown afterwards

- 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

- Add another layer of indirection to test

```
char array[256 * 4096]; // 256 pages of memory
```

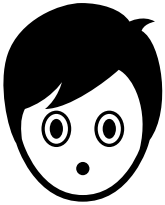
- Add another layer of indirection to test

```
char array[256 * 4096]; // 256 pages of memory


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

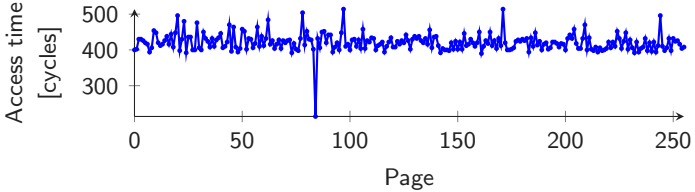
- Add another layer of indirection to test

```
char array[256 * 4096]; // 256 pages of memory


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

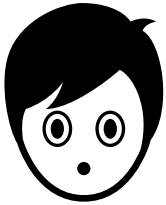
- Then check whether any part of `array` is cached



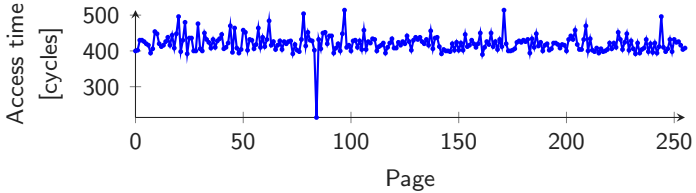
- Flush+Reload over all pages of the array



- Index of cache hit reveals data



- Flush+Reload over all pages of the array



- Index of cache hit reveals data
- Permission check is in some cases too late

- CPU uses data in out-of-order execution before permission check

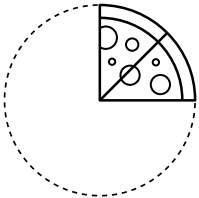
- CPU uses data in out-of-order execution before permission check
- Meltdown can read any kernel address

- CPU uses data in out-of-order execution before permission check
- Meltdown can read any kernel address
- Physical memory is usually mapped in kernel

- CPU uses data in out-of-order execution before permission check
- Meltdown can read any kernel address
- Physical memory is usually mapped in kernel

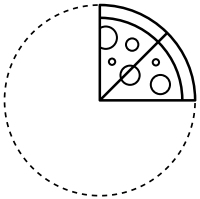
→ Read arbitrary memory

Uncached and uncachable memory



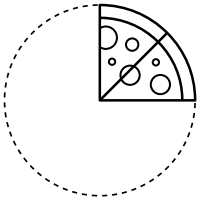
- Assumed Meltdown can one only read data from the L1

Uncached and uncachable memory



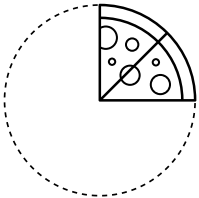
- Assumed Meltdown can one only read data from the L1
- Leakage from L3 or memory is possible, just slower

Uncached and uncachable memory



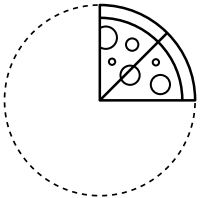
- Assumed Meltdown can one only read data from the L1
- Leakage from L3 or memory is possible, just slower
- Even leakage of UC (uncachable) memory regions...

Uncached and uncachable memory



- Assumed Meltdown can one only read data from the L1
- Leakage from L3 or memory is possible, just slower
- Even leakage of UC (uncachable) memory regions...
  - ...if other hyperthread (legally) accesses the data

Uncached and uncachable memory



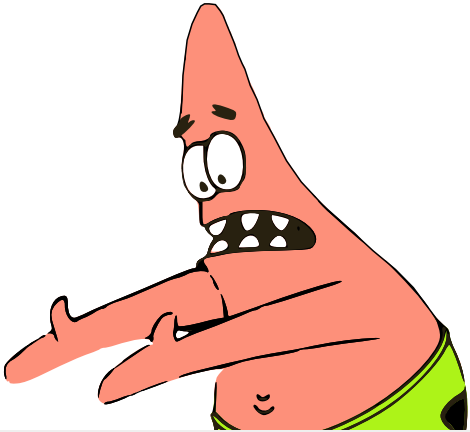
- Assumed Meltdown can one only read data from the L1
- Leakage from L3 or memory is possible, just slower
- Even leakage of UC (uncachable) memory regions...
  - ...if other hyperthread (legally) accesses the data

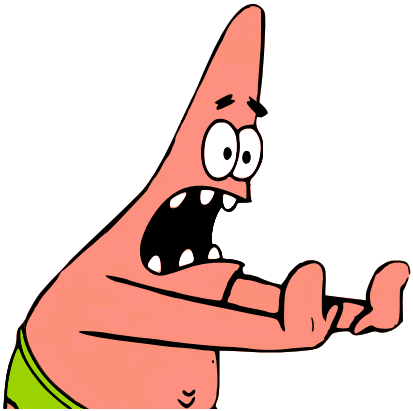
  → ...leaks from line fill buffer

- Kernel addresses in user space are a problem

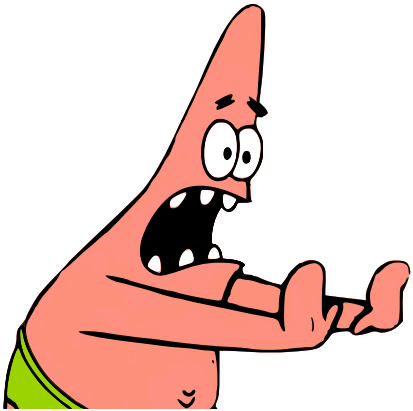
Take the kernel addresses...

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

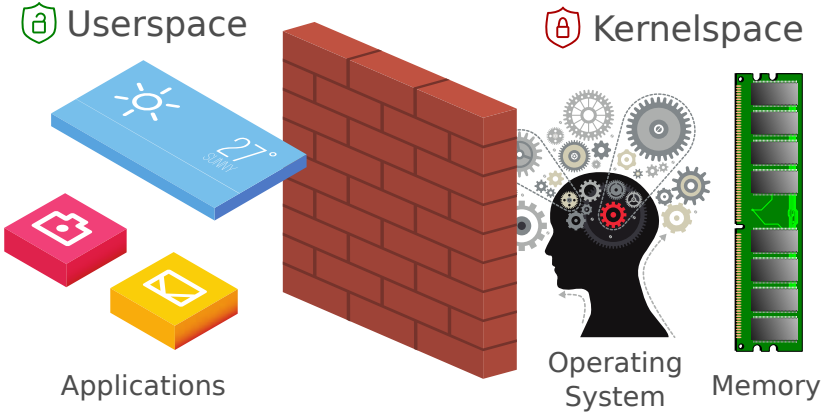




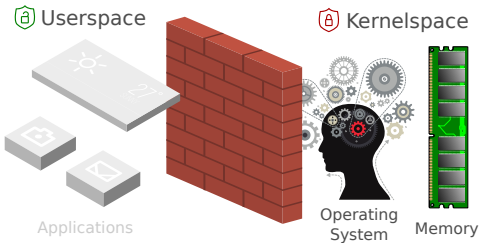
- ...and remove them if not needed?



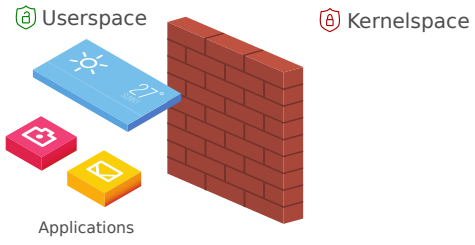
- ...and remove them if not needed?
- User accessible check in hardware is not reliable


Userspace
Kernelspace
27°
Applications
Operating System
Memory

Kernel View



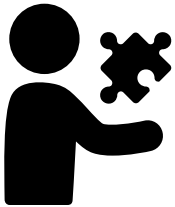
User View



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

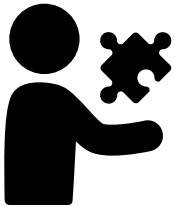
- **Linux**: Kernel Page-table Isolation (KPTI)
- **Apple**: Released updates

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



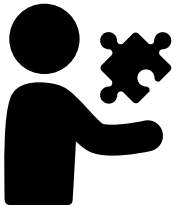
- Meltdown fully mitigated in software

Problem Solved?



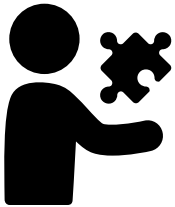
- Meltdown fully mitigated in software
- Problem seemed to be solved

Problem Solved?



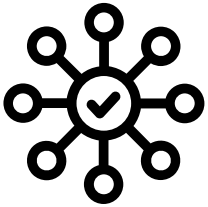
- Meltdown fully mitigated in software
- Problem seemed to be solved
- No attack surface left

Problem Solved?



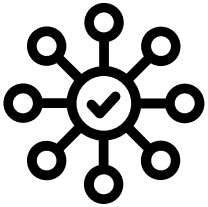
- Meltdown fully mitigated in software
- Problem seemed to be solved
- No attack surface left
- That is what everyone thought


There are no bugs,
just happy little accidents

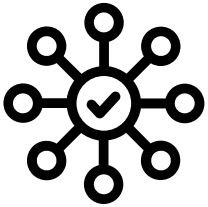


- Meltdown is a whole category of vulnerabilities

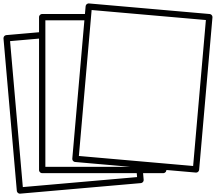
Generalization



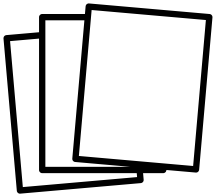
- Meltdown is a whole category of vulnerabilities
- Not only the user-accessible check



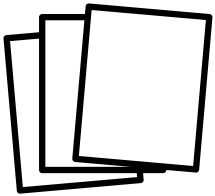
- Meltdown is a whole category of vulnerabilities
- Not only the user-accessible check
- Looking closer at the check...



- CPU uses virtual address spaces to isolate processes

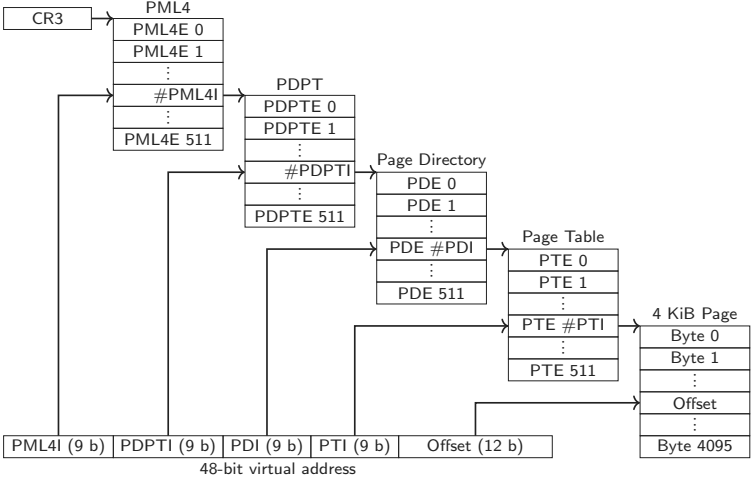


- CPU uses virtual address spaces to isolate processes
- Physical memory is organized in page frames



- CPU uses 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



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


P
RW
US
WT
UC
R
D
S
G
Ignored
Physical Page Number
Ignored
X

- Present bit is the next obvious bit

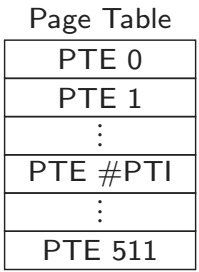
- An even worse bug → Foreshadow-NG/L1TF

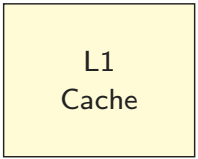
- An even worse bug → Foreshadow-NG/L1TF
- Exploitable from VMs

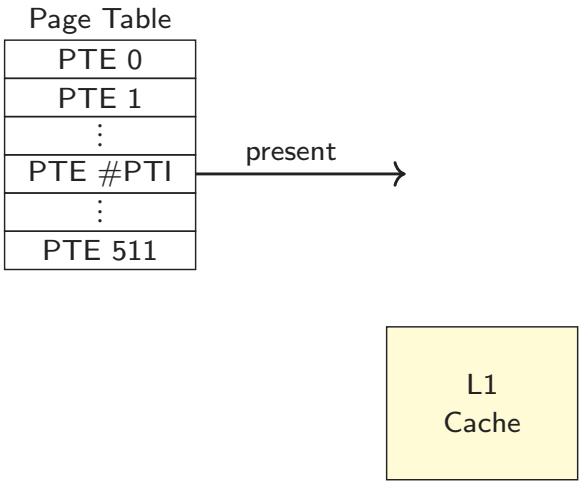
- An even worse bug → Foreshadow-NG/L1TF
- Exploitable from VMs
- Allows leaking data from the L1 cache

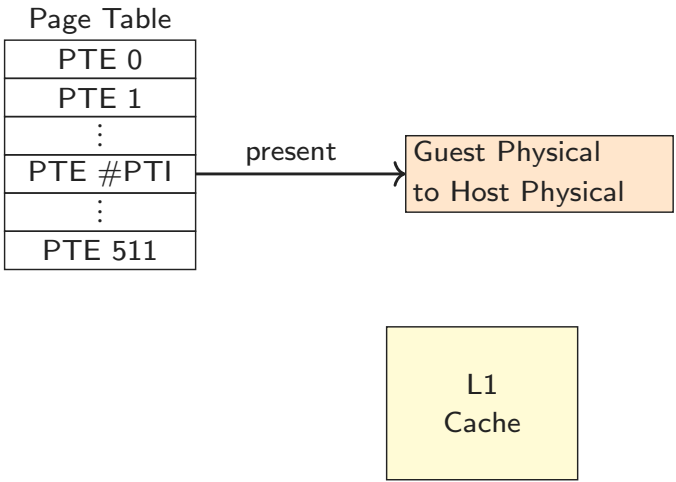
- An even worse bug → Foreshadow-NG/L1TF
- Exploitable from VMs
- Allows leaking data from the L1 cache
- Same mechanism as Meltdown

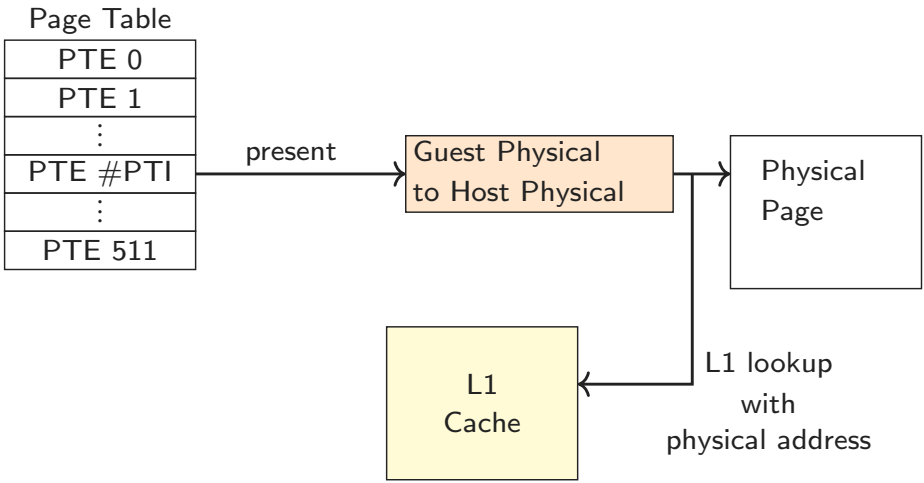
- An even worse bug → Foreshadow-NG/L1TF
- Exploitable from VMs
- Allows leaking data from the L1 cache
- Same mechanism as Meltdown
- Just a different bit in the PTE

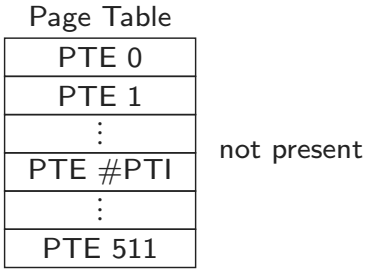

Page Table
PTE 0
PTE 1
PTE #PTI
PTE 511

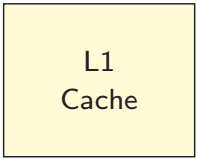

L1
Cache

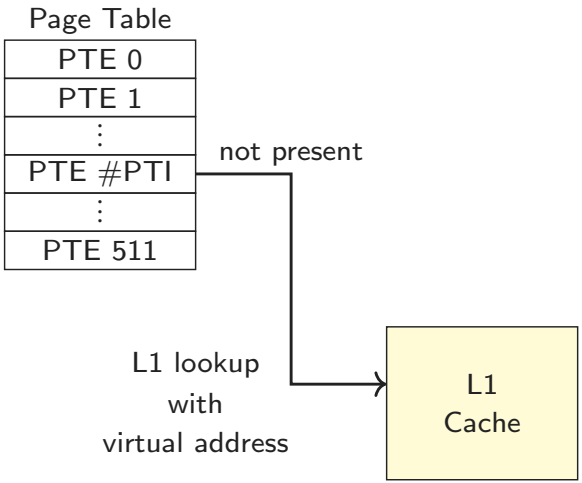

Page Table
PTE 0
PTE 1
PTE #PTI
PTE 511
present
L1
Cache


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


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


Page Table
PTE 0
PTE 1
PTE #PTI
PTE 511
not present


L1
Cache


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

- KAISER/KPTI/KVA does not help

- KAISER/KPTI/KVA does not help
- Only software workarounds

- KAISER/KPTI/KVA does not help
- Only software workarounds
  → Flush L1 on VM entry

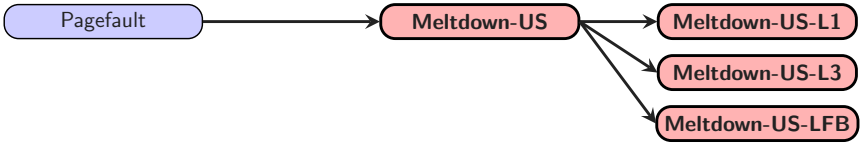
- KAISER/KPTI/KVA does not help
- Only software workarounds
  - → Flush L1 on VM entry
  - → Disable HyperThreading

- KAISER/KPTI/KVA does not help
- Only software workarounds
  - → Flush L1 on VM entry
  - → Disable HyperThreading
- Workarounds might not be complete

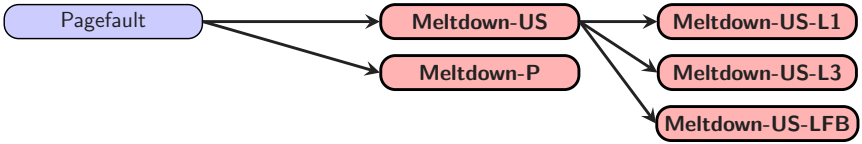
Pagefault


Pagefault
Meltdown-US

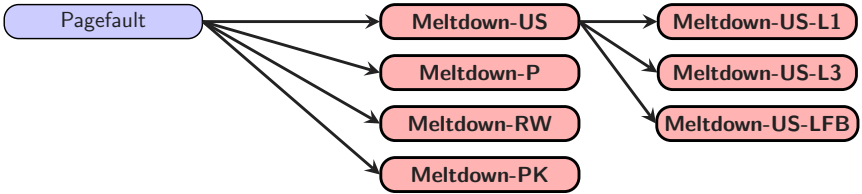
Meltdown Variants



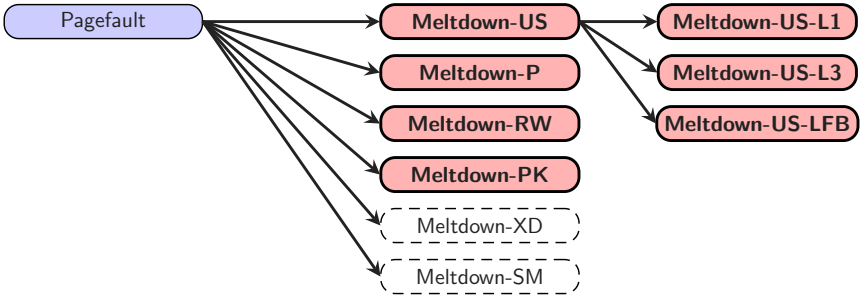
Meltdown Variants

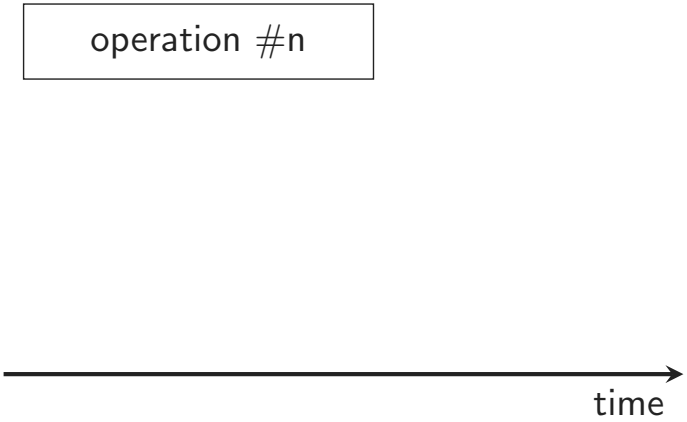


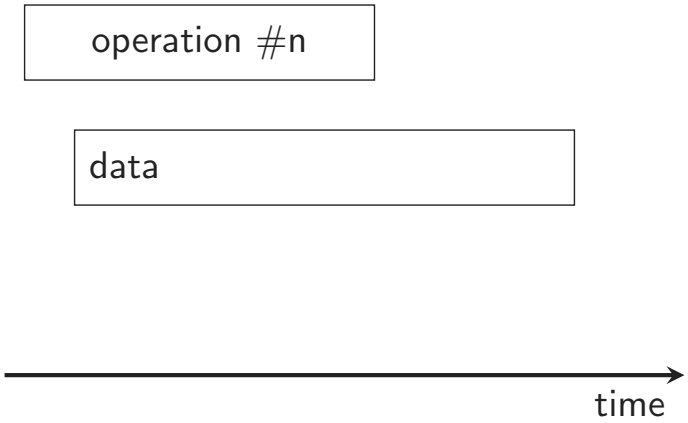
Meltdown Variants

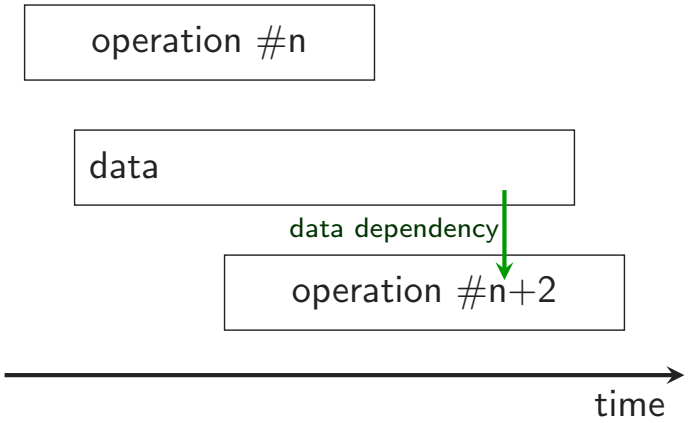


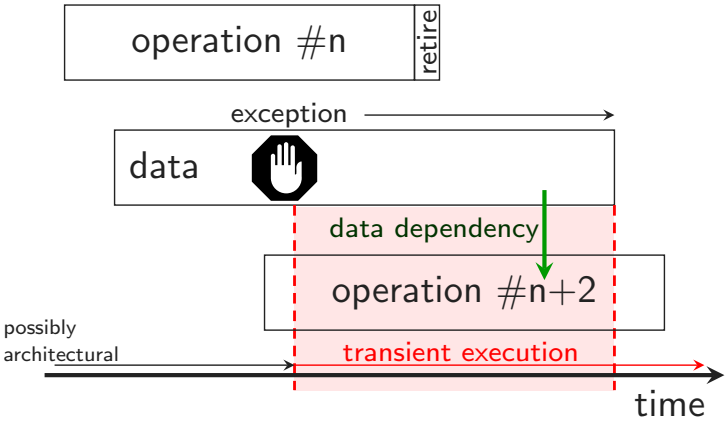
Meltdown Variants

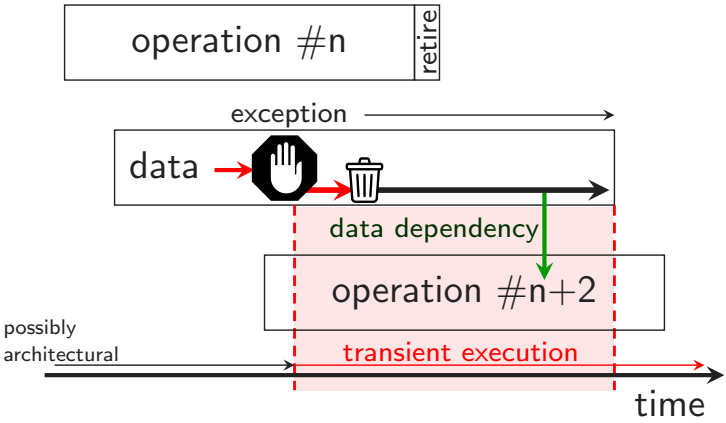


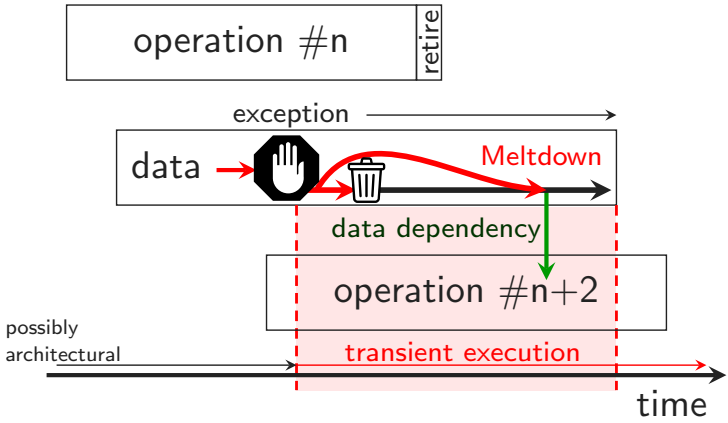

operation #n
time

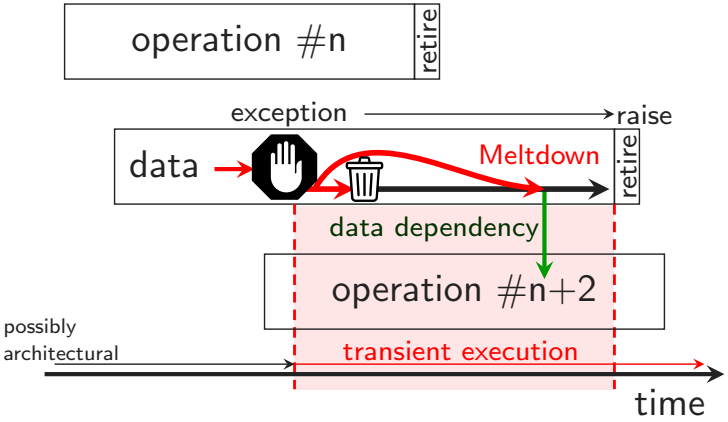

operation #n
data
time


operation #n
data
data dependency
operation #n+2
time

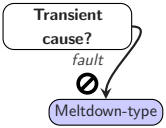

operation #n
retire
exception
data
data dependency
operation #n+2
possibly
architectural
transient execution
time


operation #n
retire
exception
data
data dependency
operation #n+2
possibly
architectural
transient execution
time

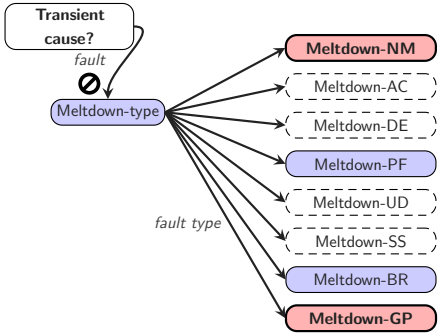

operation #n
retire
exception
data
Meltdown
data dependency
operation #n+2
possibly
architectural
transient execution
time


operation #n
retire
exception
raise
data
Meltdown
retire
data dependency
operation #n+2
possibly
architectural
transient execution
time

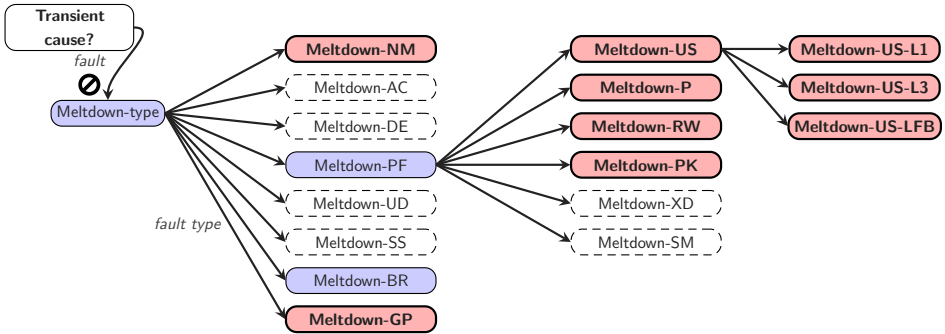
Transient cause?


Transient cause?
fault
Meltdown-type

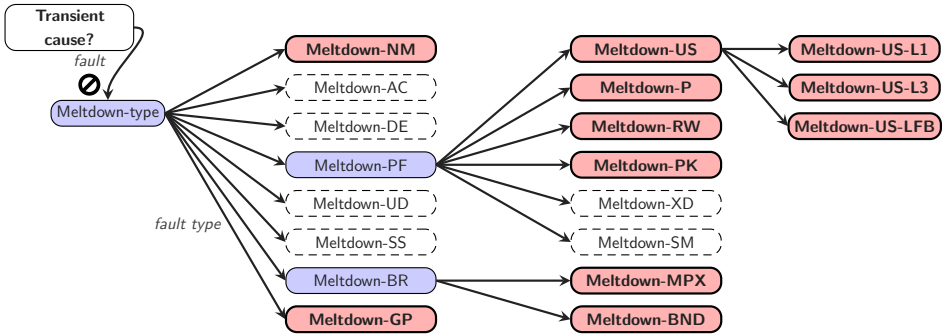
Meltdown Tree



Meltdown Tree



Meltdown Tree



- Meltdown is not a fully solved issue

- Meltdown is not a fully solved issue
- The tree is extensible

- Meltdown is not a fully solved issue
- The tree is extensible
- Silicon fixes might not be complete

- Meltdown not the only transient execution attacks

- Meltdown not the only transient execution attacks
- Spectre is a second class of transient execution attacks

- Meltdown not the only transient execution attacks
- Spectre is a second class of transient execution attacks
- Instead of faults, exploit control (or data) flow predictions

- CPU tries to predict the future (branch predictor), ...

- CPU tries to predict the future (branch predictor), ...
  - ... based on events learned in the past

- CPU tries to predict the future (branch predictor), ...
  - ... based on events learned in the past
- Speculative execution of instructions

- CPU tries to predict the future (branch predictor), ...
  - ... based on events learned in the past
- Speculative execution of instructions
- If the prediction was correct, ...

- CPU tries to predict the future (branch predictor), ...
  - ... based on events learned in the past
- Speculative execution of instructions
- If the prediction was correct, ...
  - ... very fast

- CPU tries to predict the future (branch predictor), ...
  - ... based on events learned in the past
- Speculative execution of instructions
- If the prediction was correct, ...
  - ... very fast
  - otherwise: Discard results

```
index = 0
```

```
if (index < 4)
```

then

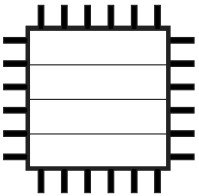
else

Shared Memory

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

```
glyph[data[index]]
```

```
{}
```



Memory

D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

Spectre-PHT (aka Spectre Variant 1)

```
index = 0
```

Shared Memory

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

```
if (index < 4)
```

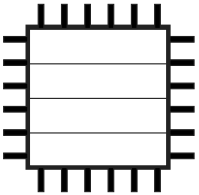
then

```
glyph[data[index]]
```

else

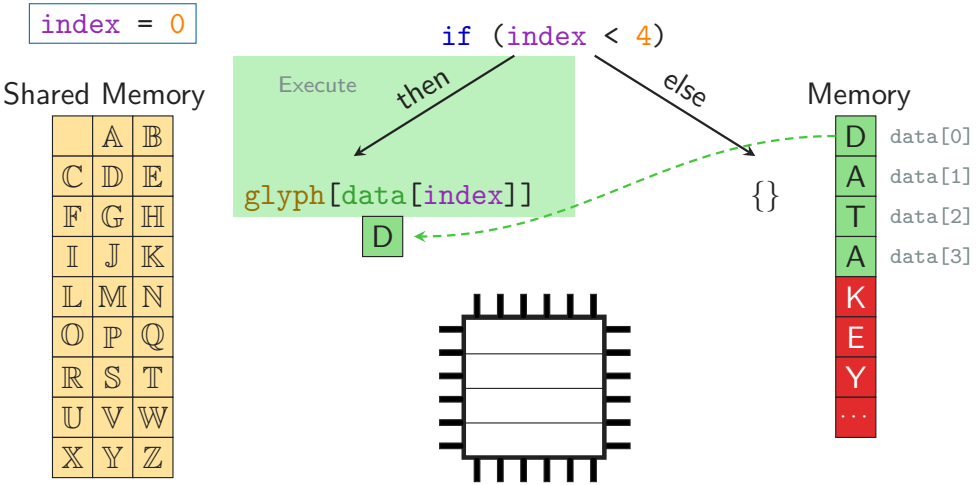
Speculate

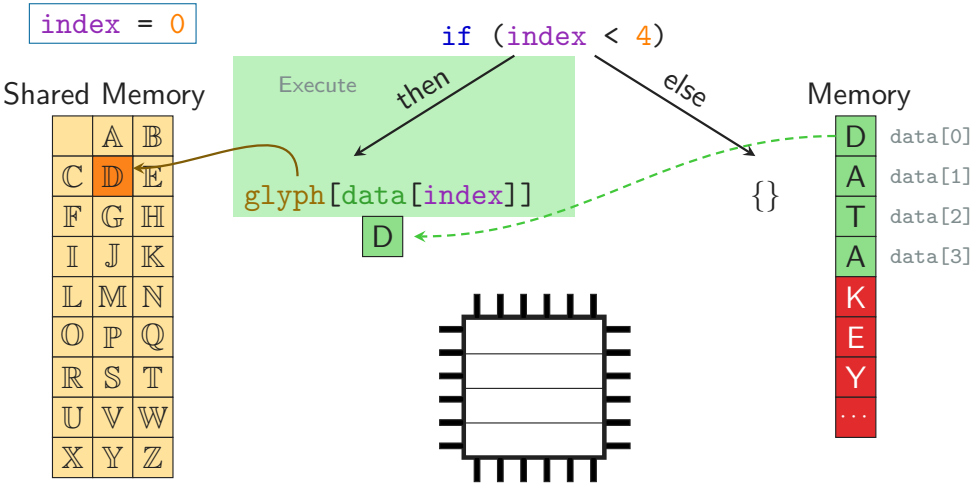
{}

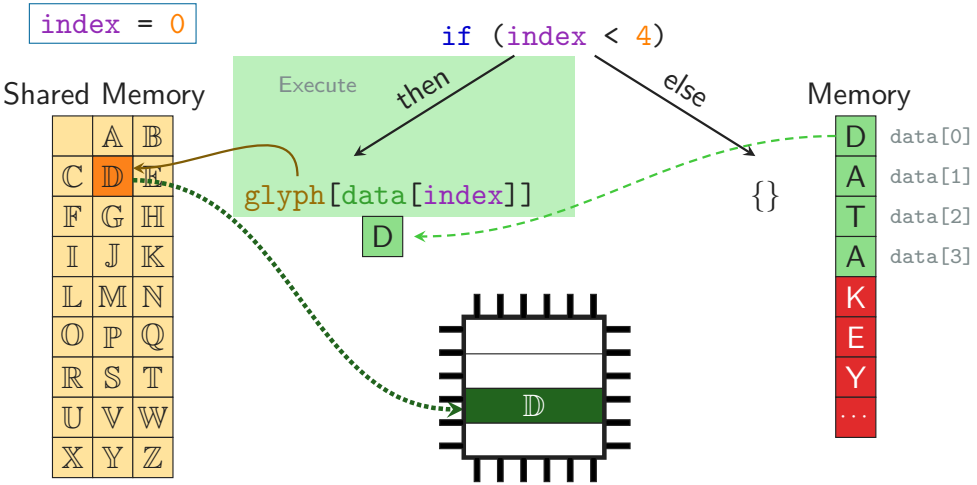


Memory

D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	


index = 0
if (index < 4)
then
else
Execute
glyph[data[index]]
D
{}
Shared Memory
A B
C D E
F G H
I J K
L M N
O P Q
R S T
U V W
X Y Z
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]


index = 0
if (index < 4)
then
else
Execute
Shared Memory
A B
C D E
F G H
I J K
L M N
O P Q
R S T
U V W
X Y Z
glyph[data[index]]
D
{}
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]


index = 0
if (index < 4)
then
else
Execute
Shared Memory
Memory
glyph[data[index]]
D
{}
data[0]
data[1]
data[2]
data[3]
D
A
T
A
K
E
Y
...

```
index = 1
```

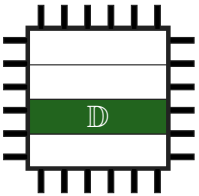
Shared Memory

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

```
if (index < 4)
```

then → `glyph[data[index]]`

else → `{}`



Memory

D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

Spectre-PHT (aka Spectre Variant 1)

```
index = 1
```

```
if (index < 4)
```

Speculate

then

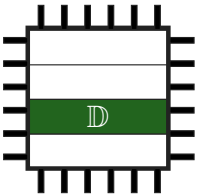
```
glyph[data[index]]
```

else

{}

Shared Memory

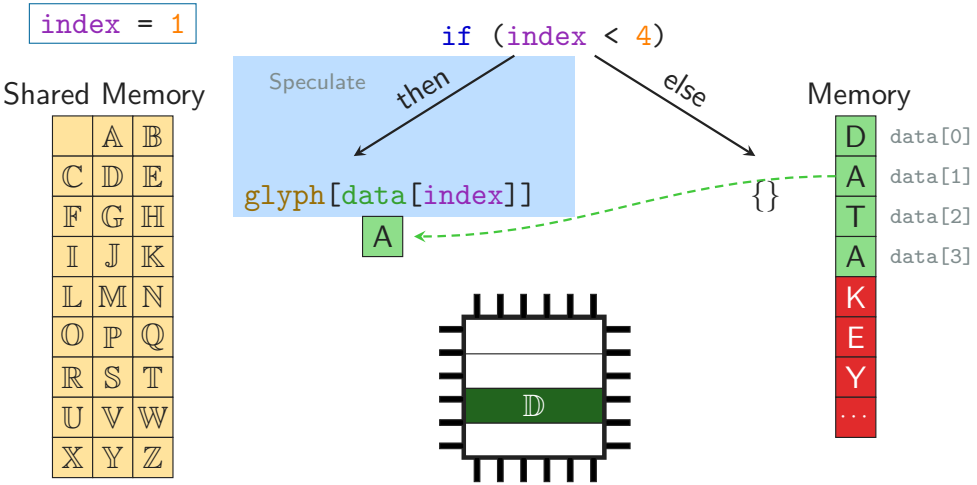
	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

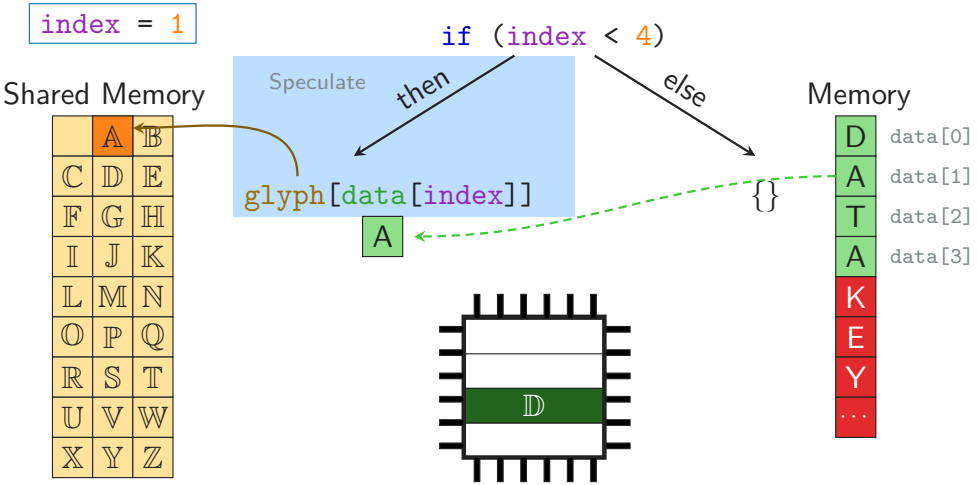


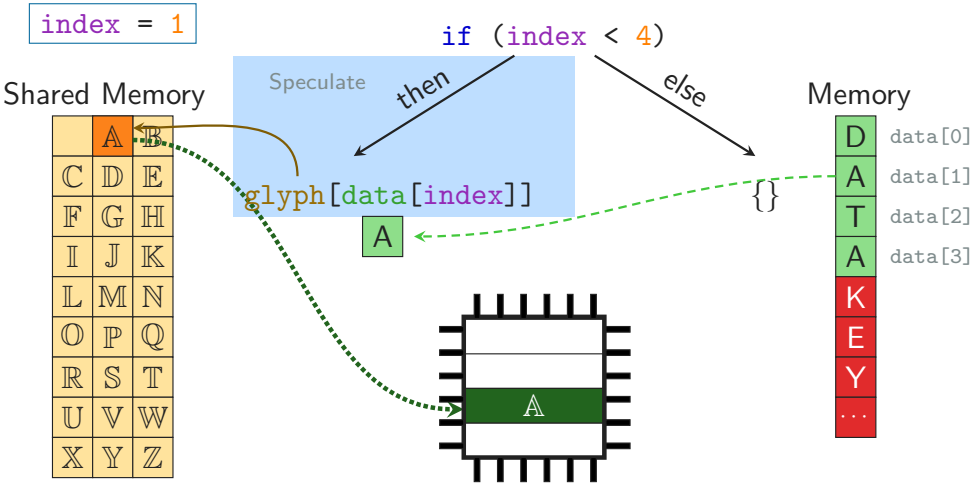
Memory

D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

Spectre-PHT (aka Spectre Variant 1)




index = 1
if (index < 4)
then
else
Speculate
Shared Memory
Memory
glyph[data[index]]
A
{}
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]
A B
C D E
F G H
I J K
L M N
O P Q
R S T
U V W
X Y Z
D


index = 1
if (index < 4)
then
else
Speculate
Shared Memory
Memory
glyph[data[index]]
A
{}
data[0]
data[1]
data[2]
data[3]
D
A
T
A
K
E
Y
...

```
index = 1
```

```
if (index < 4)
```

then

Execute

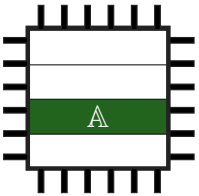
```
glyph[data[index]]
```

else

{}

Shared Memory

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



Memory

D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

```
index = 2
```

```
if (index < 4)
```

Speculate

then

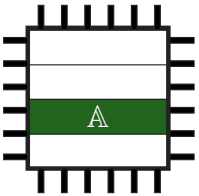
```
glyph[data[index]]
```

else

```
{}
```

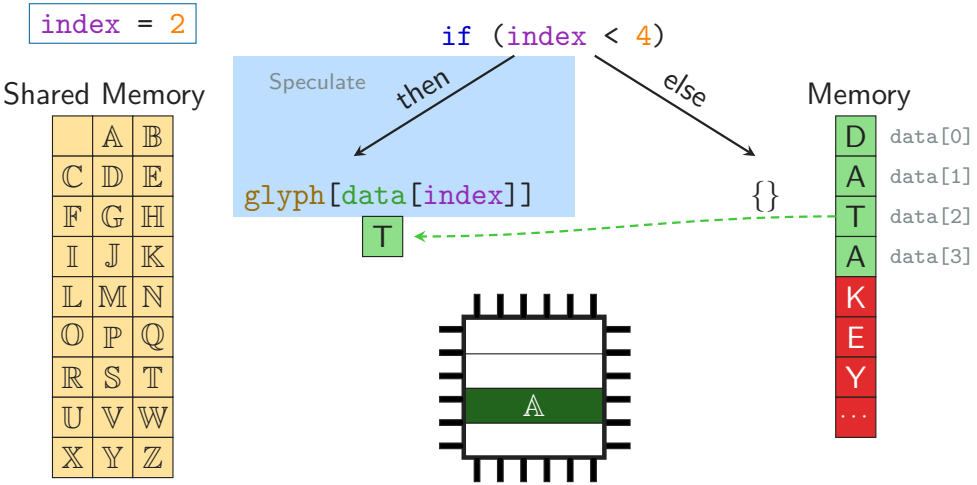
Shared Memory

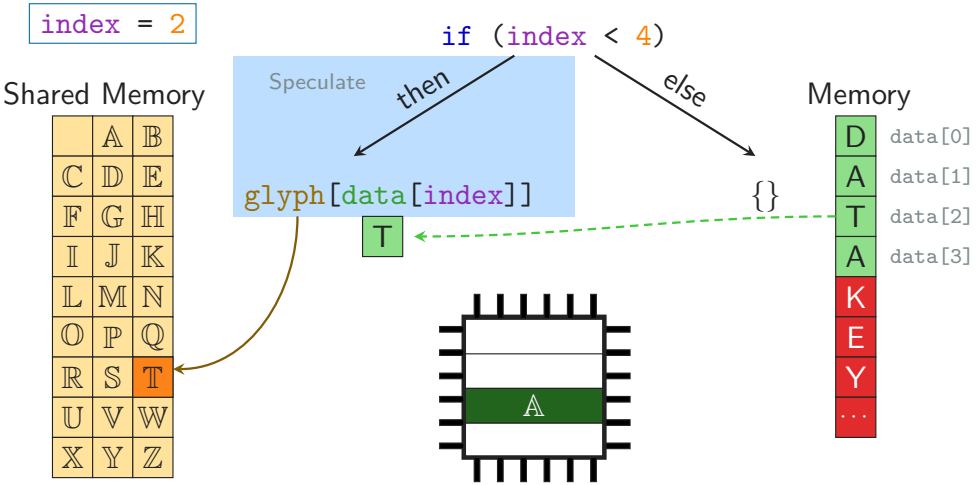
	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

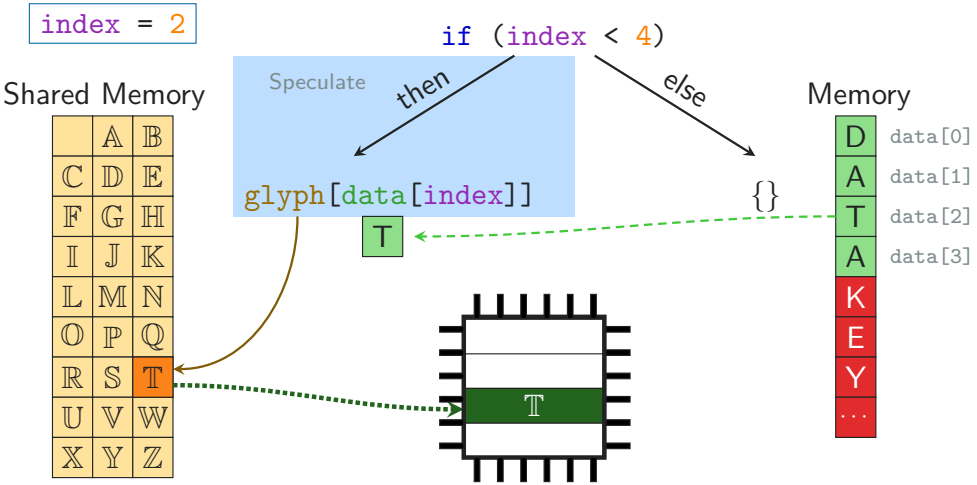


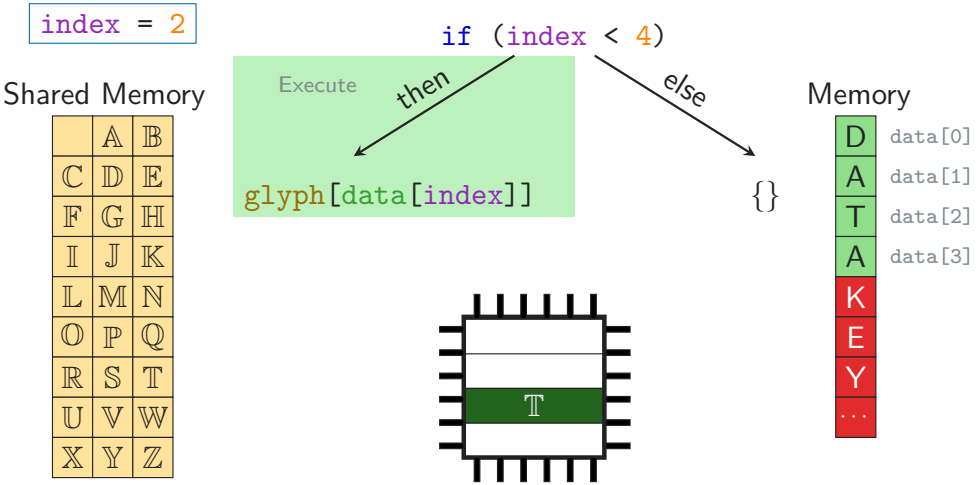
Memory

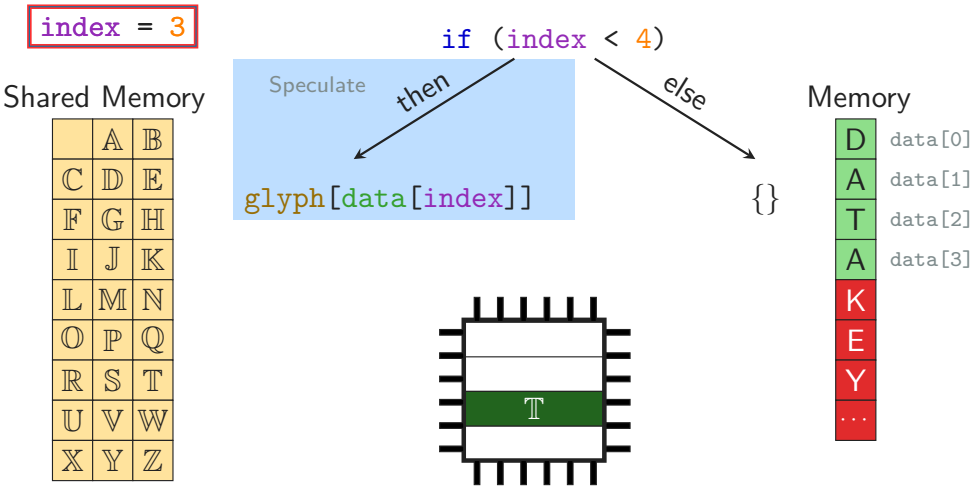
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

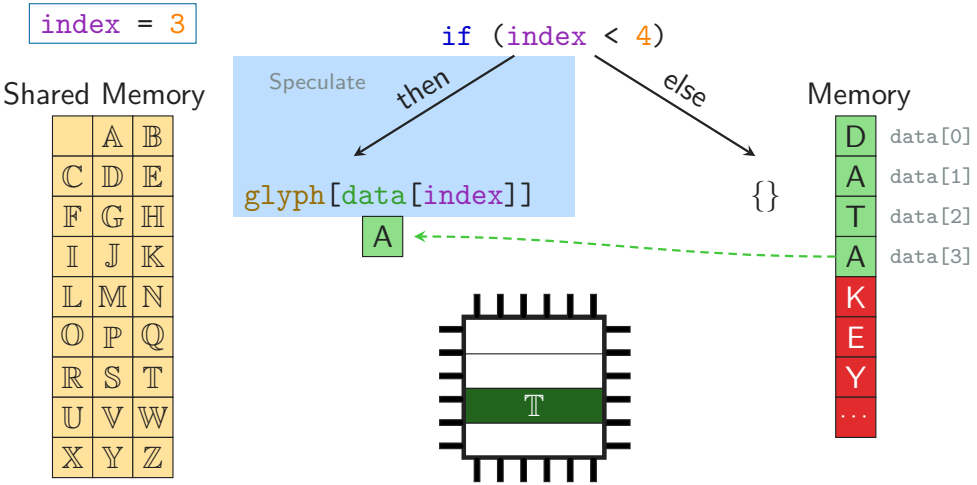

index = 2
if (index < 4)
Speculate
then
else
Shared Memory
Memory
glyph[data[index]]
{}
T
A
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]

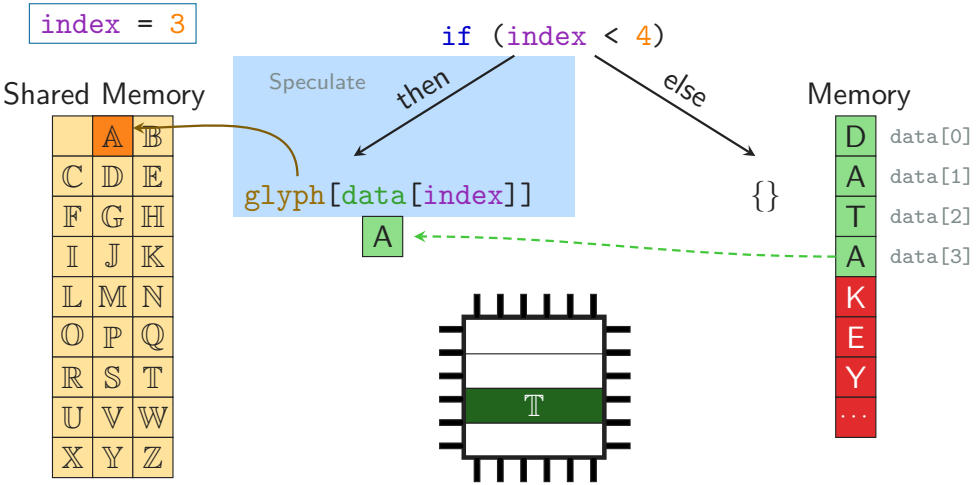

index = 2
if (index < 4)
then
else
Speculate
glyph[data[index]]
T
{}
Shared Memory
Memory
data[0]
data[1]
data[2]
data[3]
A

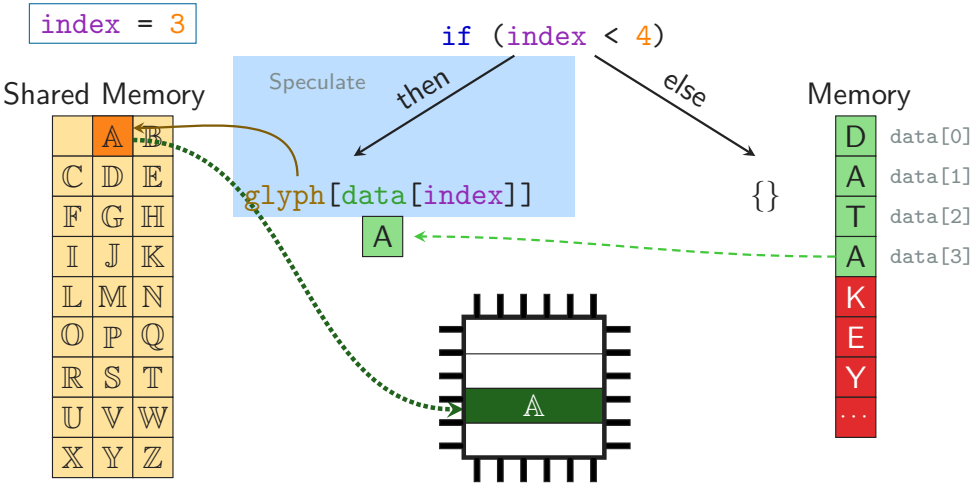

index = 2
if (index < 4)
then
else
Speculate
glyph[data[index]]
T
{}
Shared Memory
Memory
data[0]
data[1]
data[2]
data[3]
T

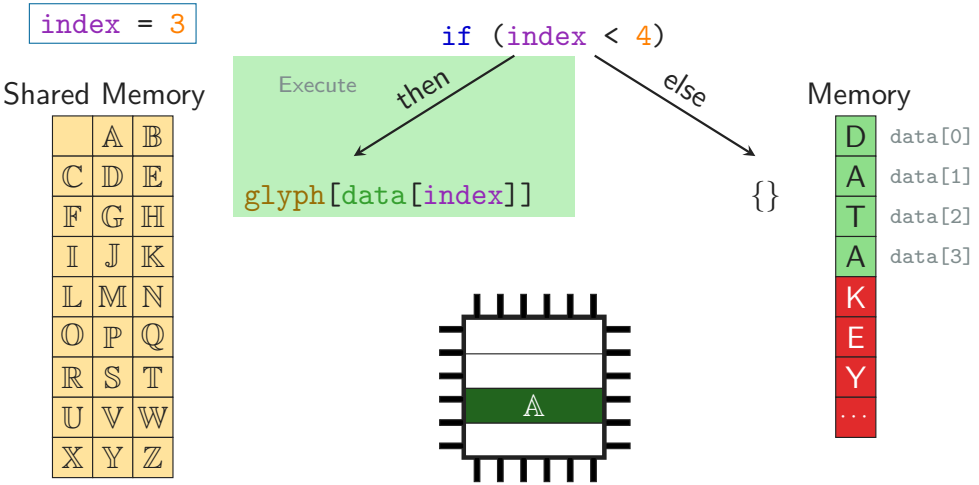

index = 2
if (index < 4)
Execute
then
else
glyph[data[index]]
{}
Shared Memory
Memory
data[0]
data[1]
data[2]
data[3]
T


index = 3
if (index < 4)
Speculate
then
else
glyph[data[index]]
{}
Shared Memory
Memory
data[0]
data[1]
data[2]
data[3]


index = 3
if (index < 4)
then
else
Speculate
glyph[data[index]]
{}
A
Shared Memory
Memory
data[0]
data[1]
data[2]
data[3]
T


index = 3
if (index < 4)
then
else
Speculate
Shared Memory
glyph[data[index]]
A
{}
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]


index = 3
if (index < 4)
Speculate
then
else
Shared Memory
A B
C D E
F G H
I J K
L M N
O P Q
R S T
U V W
X Y Z
glyph[data[index]]
A
{}
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]
A


index = 3
if (index < 4)
Execute
then
else
glyph[data[index]]
{}
Shared Memory
Memory
data[0]
data[1]
data[2]
data[3]
A

```
index = 4
```

```
if (index < 4)
```

Speculate

then

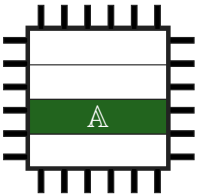
```
glyph[data[index]]
```

else

```
{}
```

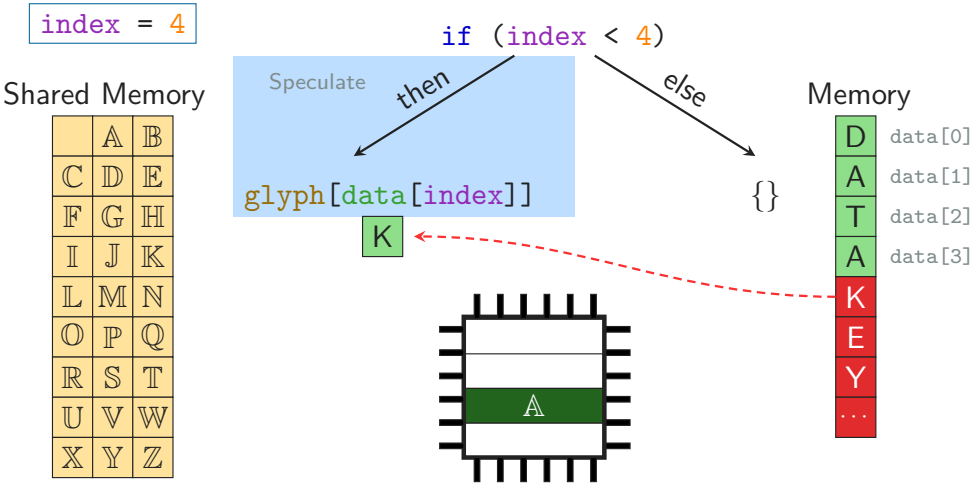
Shared Memory

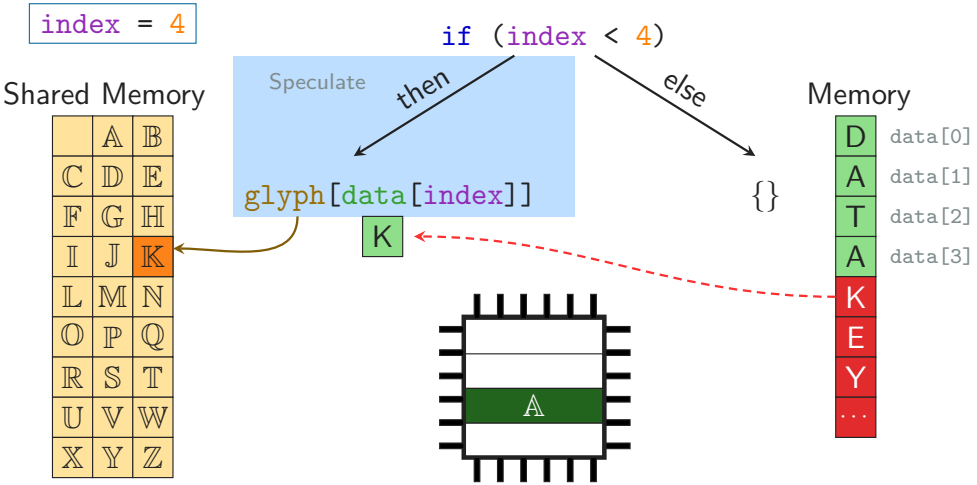
	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

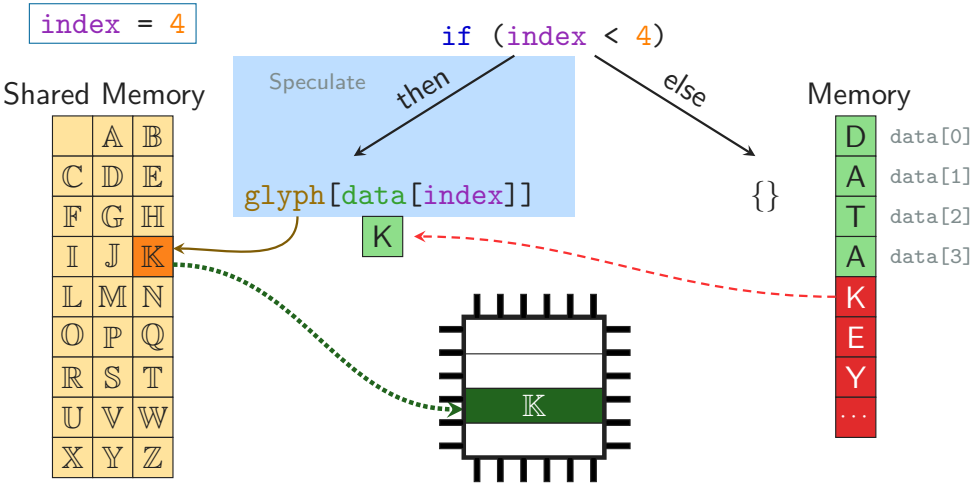


Memory

D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	


index = 4
if (index < 4)
Speculate
then
else
glyph[data[index]]
{}
K
Shared Memory
Memory
data[0]
data[1]
data[2]
data[3]
D
A
T
A
K
E
Y
...
A


index = 4
if (index < 4)
Speculate
then
else
Shared Memory
glyph[data[index]]
K
{}
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]
A


index = 4
if (index < 4)
Speculate
then
else
Shared Memory
glyph[data[index]]
K
{}
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]
K

```
index = 4
```

Shared Memory

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

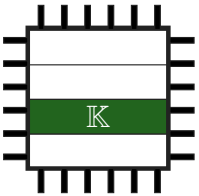
```
if (index < 4)
```

then

```
glyph[data[index]]
```

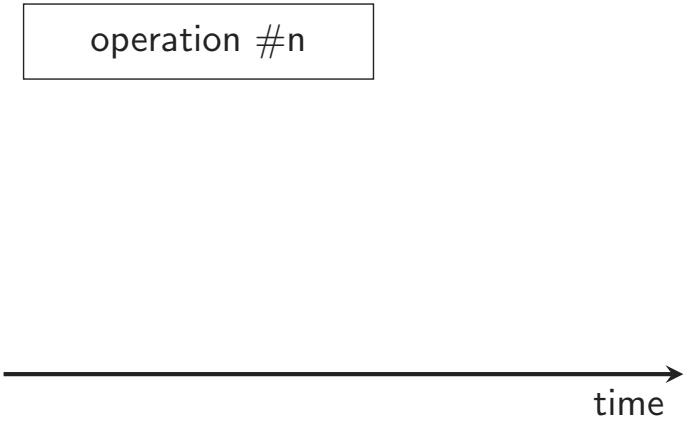
else

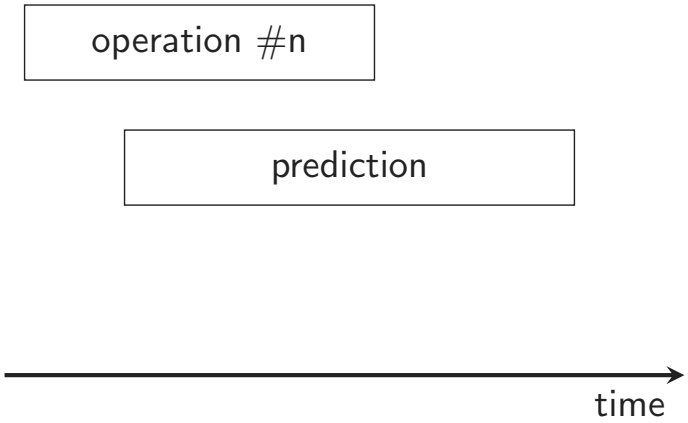
Execute {}

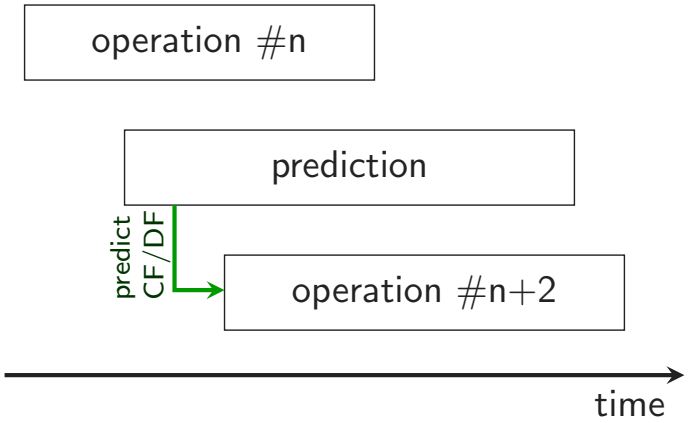


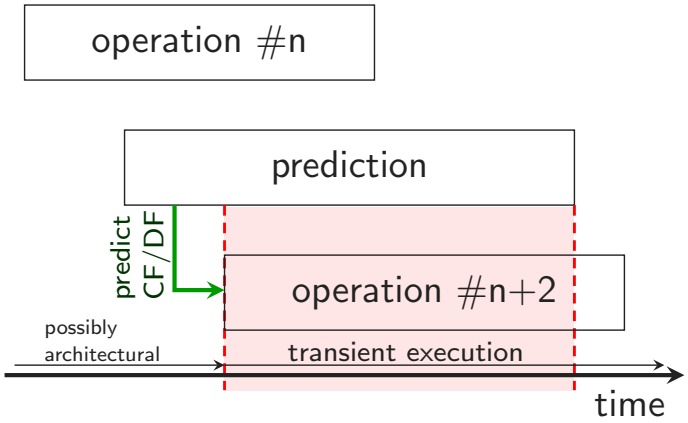
Memory

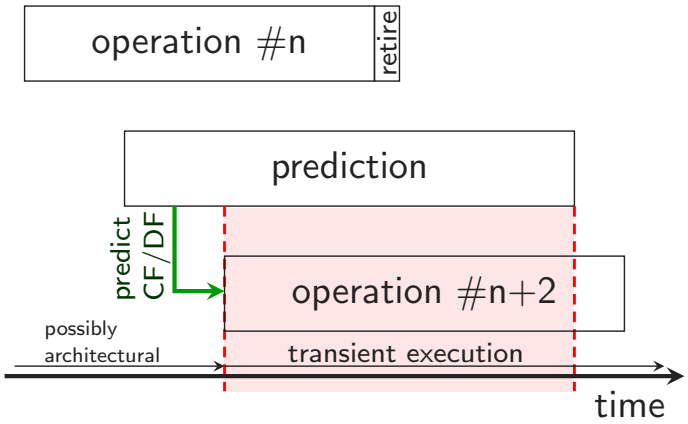
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

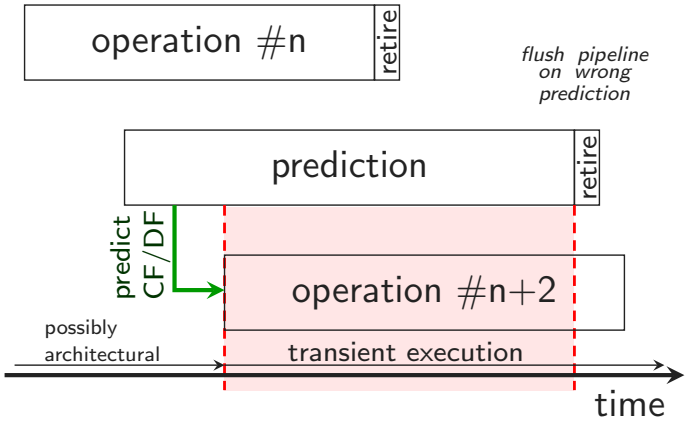

operation #n
time

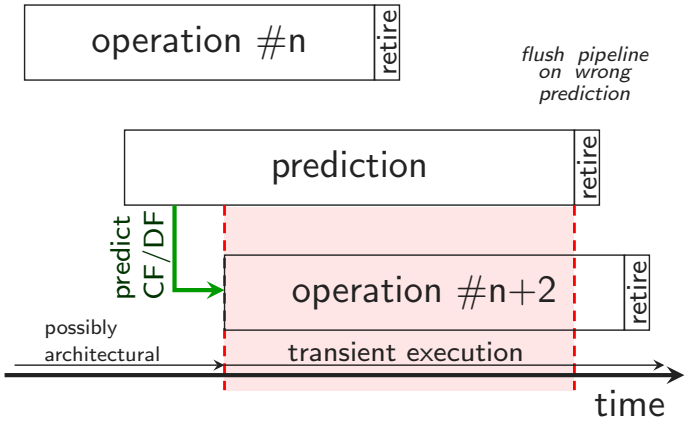

operation #n
prediction
time

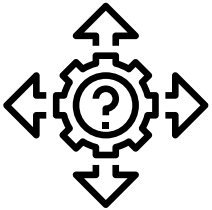

operation #n
prediction
predict
CF/DF
operation #n+2
time


operation #n
prediction
predict CF/DF
operation #n+2
possibly architectural
transient execution
time

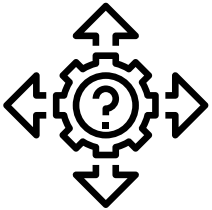

operation #n
retire
prediction
predict CF/DF
operation #n+2
possibly architectural
transient execution
time


operation #n
retire
flush pipeline on wrong prediction
prediction
retire
predict CF/DF
operation #n+2
possibly architectural
transient execution
time

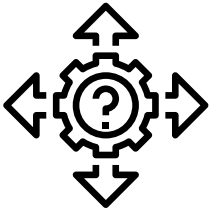

operation #n
retire
flush pipeline on wrong prediction
prediction
retire
predict CF/DF
operation #n+2
retire
possibly architectural
transient execution
time



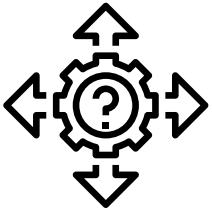
- Many predictors in modern CPUs



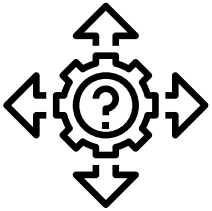
- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)



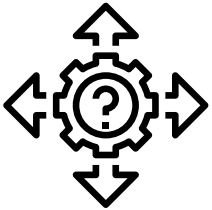
- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)
  - Call/Jump destination (BTB)



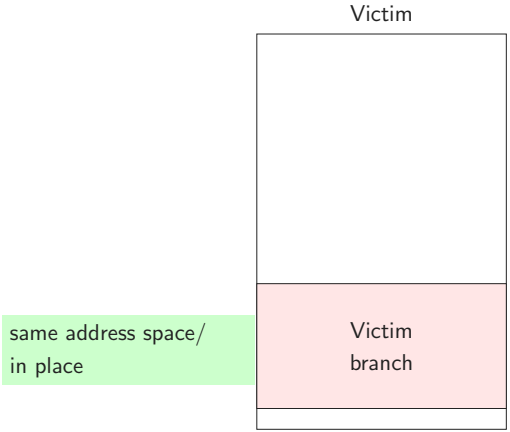
- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)
  - Call/Jump destination (BTB)
  - Function return destination (RSB)

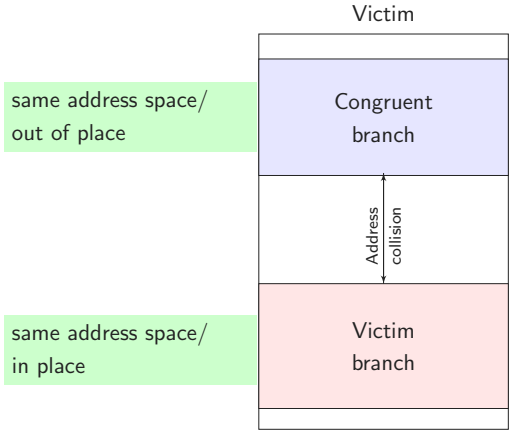


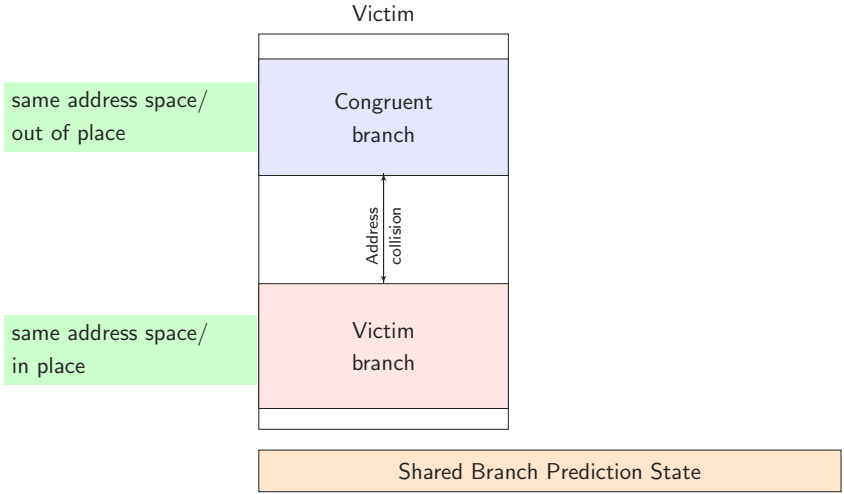
- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)
  - Call/Jump destination (BTB)
  - Function return destination (RSB)
  - Load matches previous store (STL)



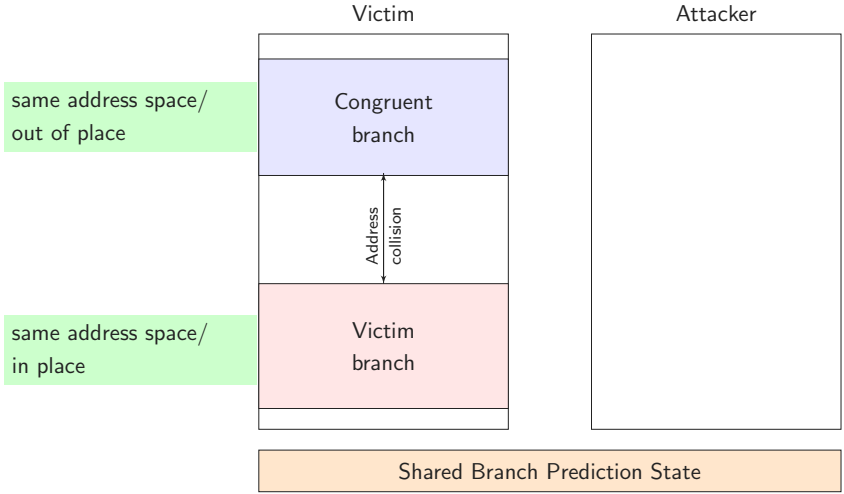
- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)
  - Call/Jump destination (BTB)
  - Function return destination (RSB)
  - Load matches previous store (STL)
- Most are even shared among processes

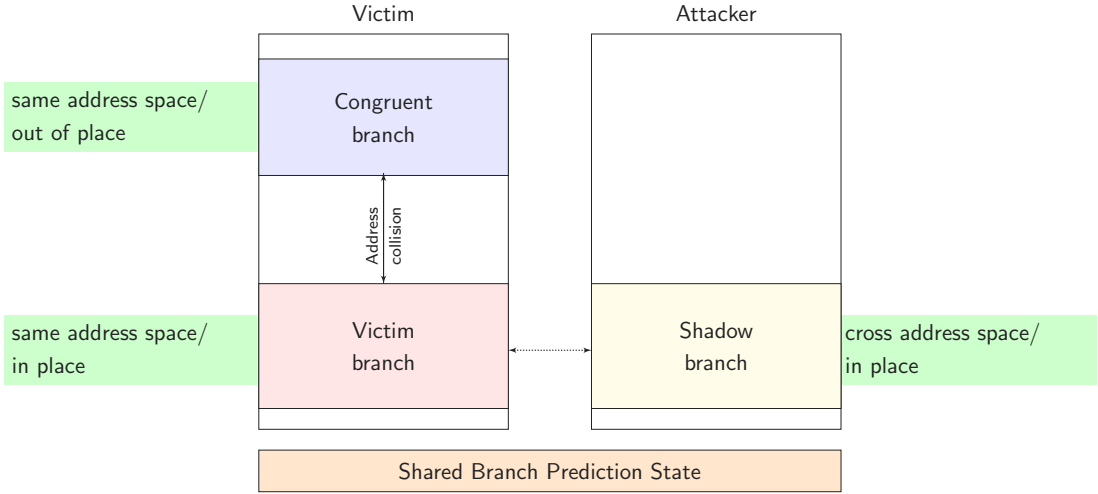

Victim
same address space/
in place
Victim
branch

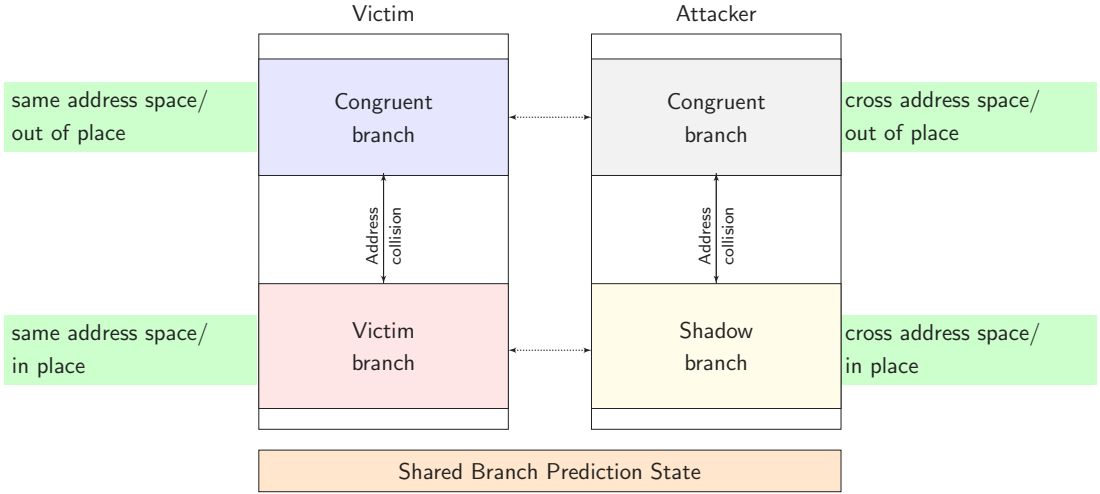

Victim
same address space/
out of place
Congruent
branch
Address
collision
same address space/
in place
Victim
branch


Victim
same address space/
out of place
Congruent
branch
Address
collision
same address space/
in place
Victim
branch
Shared Branch Prediction State

Spectre Mistraining




Victim
Attacker
same address space/
out of place
Congruent
branch
Address
collision
same address space/
in place
Victim
branch
Shadow
branch
cross address space/
in place
Shared Branch Prediction State


Victim
Attacker
same address space/
out of place
Congruent
branch
Congruent
branch
cross address space/
out of place
Address
collision
Address
collision
same address space/
in place
Victim
branch
Shadow
branch
cross address space/
in place
Shared Branch Prediction State

- Questions:

- Questions:
  - Size of speculation window?

- Questions:
  - Size of speculation window?
  - Best working gadget?

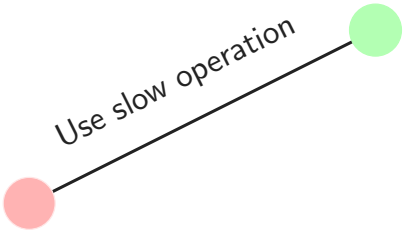
- Questions:
  - Size of speculation window?
  - Best working gadget?
- Experiment Setup:

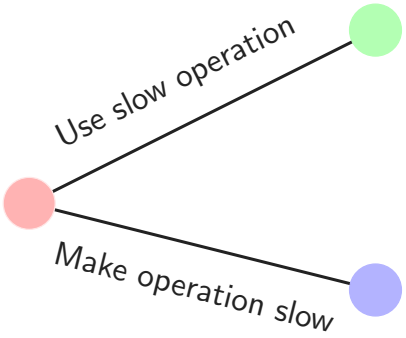
- Questions:
  - Size of speculation window?
  - Best working gadget?
- Experiment Setup:
  - ARM Cortex A-57

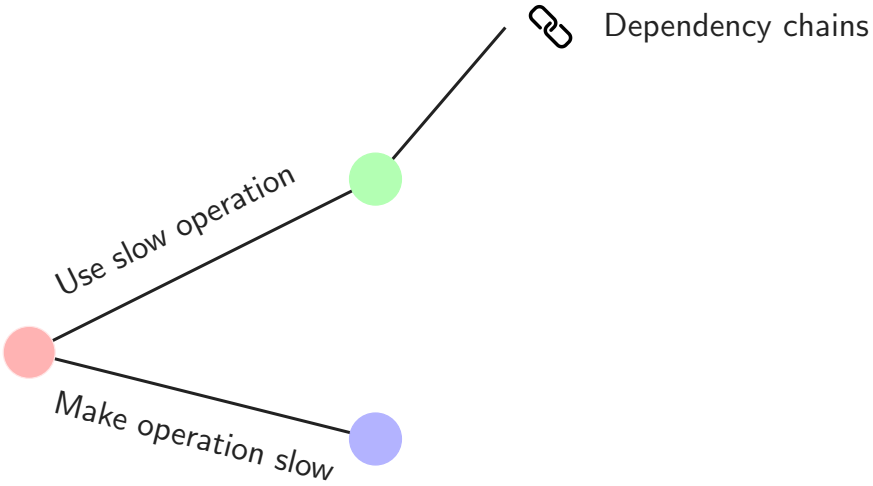
- Questions:
  - Size of speculation window?
  - Best working gadget?
- Experiment Setup:
  - ARM Cortex A-57
  - Spectre-PHT with F+R

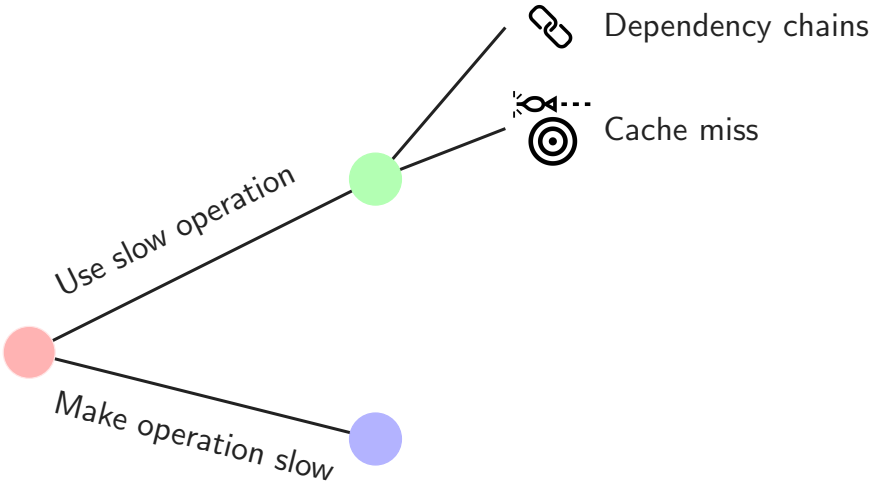
- Questions:
  - Size of speculation window?
  - Best working gadget?
- Experiment Setup:
  - ARM Cortex A-57
  - Spectre-PHT with F+R
  - 1 run: leak 1 byte

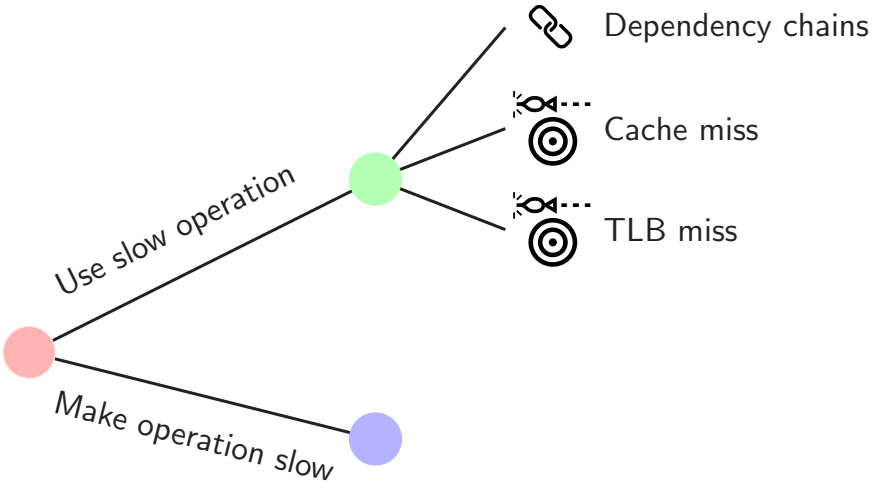
- Questions:
  - Size of speculation window?
  - Best working gadget?
- Experiment Setup:
  - ARM Cortex A-57
  - Spectre-PHT with F+R
  - 1 run: leak 1 byte
  - 1 test: do *n* runs in *m* different processes

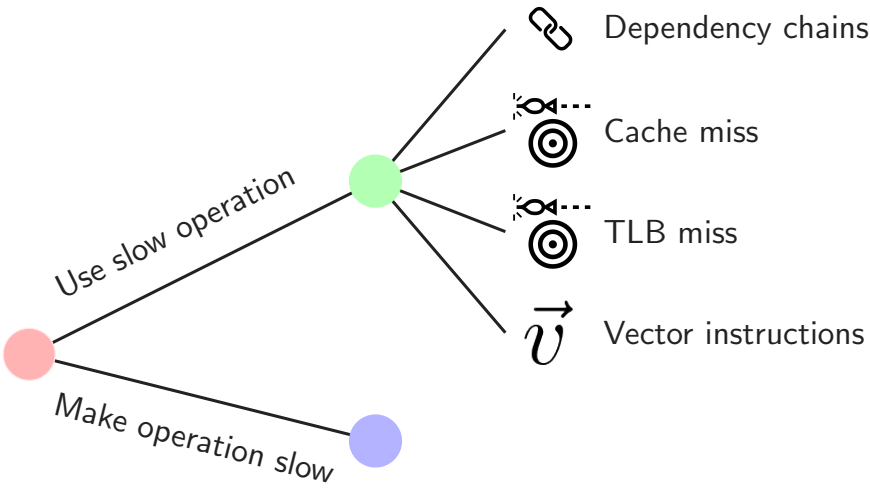

Use slow operation

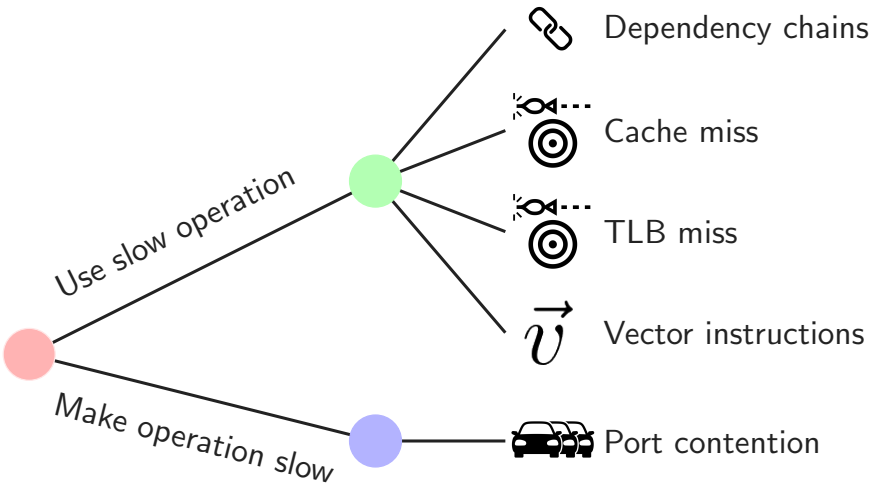

Use slow operation
Make operation slow

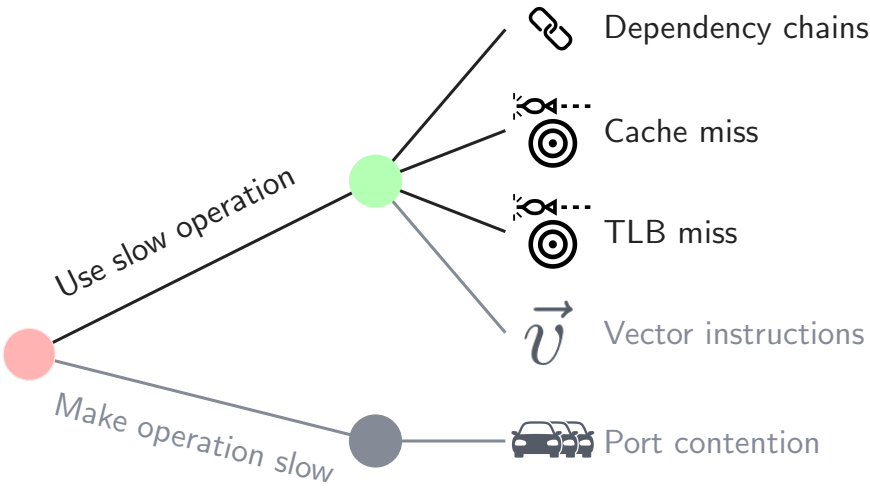

Dependency chains
Use slow operation
Make operation slow


Dependency chains
Cache miss
Use slow operation
Make operation slow

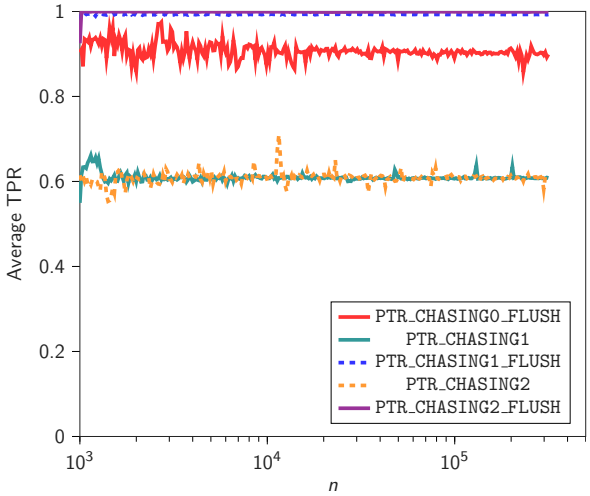

Dependency chains
Cache miss
TLB miss
Use slow operation
Make operation slow


Dependency chains
Cache miss
TLB miss
Vector instructions
Use slow operation
Make operation slow

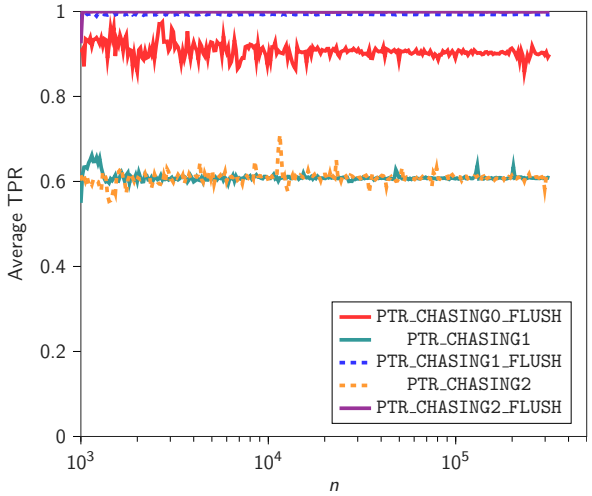

Use slow operation
Make operation slow
Dependency chains
Cache miss
TLB miss
Vector instructions
Port contention


Use slow operation
Make operation slow
Dependency chains
Cache miss
TLB miss
Vector instructions
Port contention

Preliminary study ($m = 10$)



Preliminary study ($m = 10$)



$\rightarrow n = 100000$

Results: Dependency chains

```
asm volatile("mov x3, #0"
```

Results: Dependency chains

```
asm volatile("mov x3, #0"
"mul x3, x3, x3"
"mul x3, x3, x3"
...
```

Results: Dependency chains

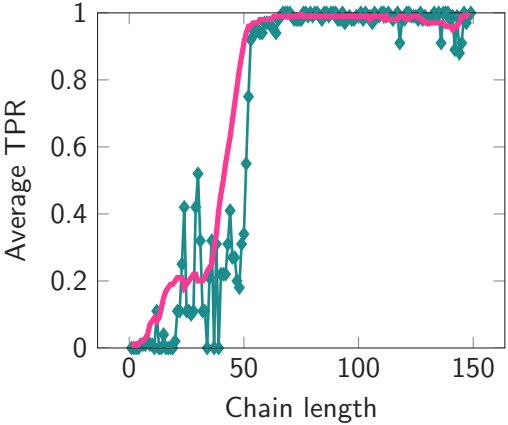
```
asm volatile("mov x3, #0"
"mul x3, x3, x3"
"mul x3, x3, x3"
...
"mov %0, x3"
 : "=r"(res));
```

Results: Dependency chains

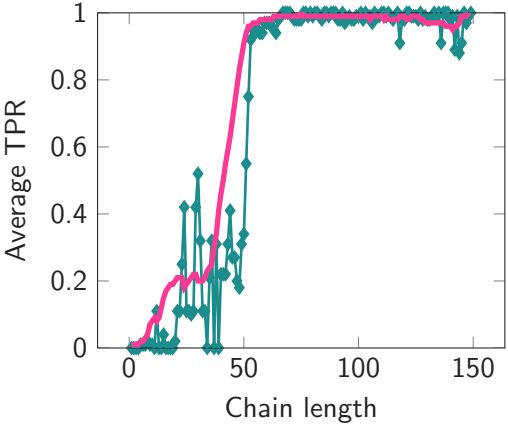
```
asm volatile("mov x3, #0"
"mul x3, x3, x3"
"mul x3, x3, x3"
...
"mov %0, x3"
 : "=r"(res));
if((x + res) < len)
  oracle[data[x]*4096];
```

Results: Dependency chains

```
asm volatile("mov x3, #0"
"mul x3, x3, x3"
"mul x3, x3, x3"
...
"mov %0, x3"
 : "=r"(res));
if((x + res) < len)
  oracle[data[x]*4096];
```



```
asm volatile("mov x3, #0"
"mul x3, x3, x3"
"mul x3, x3, x3"
...
"mov %0, x3"
 : "=r"(res));
if((x + res) < len)
  oracle[data[x]*4096];
```



Maximal speculation window size: 57 instructions at a chain length of 22

Results: Cache miss

```
unsigned char value = 0;
char* ptr = &value;
char** ptr2 = &ptr;
char*** ptr3 = &ptr2;
...
```

Results: Cache miss

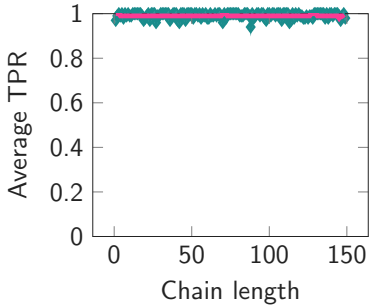
```
unsigned char value = 0;
char* ptr = &value;
char** ptr2 = &ptr;
char*** ptr3 = &ptr2;
...

flush(ptr);
flush(ptr2);
flush(ptr3);
...
```

Results: Cache miss

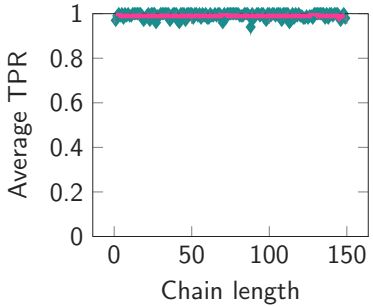
```
unsigned char value = 0;
char* ptr = &value;
char** ptr2 = &ptr;
char*** ptr3 = &ptr2;
...

flush(ptr);
flush(ptr2);
flush(ptr3);
...

if((x + ***ptr3) < len)
  oracle[data[x]*4096];
```

Results: Cache miss

```
unsigned char value = 0;
char* ptr = &value;
char** ptr2 = &ptr;
char*** ptr3 = &ptr2;
...

flush(ptr);
flush(ptr2);
flush(ptr3);
...

if((x + ***ptr3) < len)
  oracle[data[x]*4096];
```



```
unsigned char value = 0;
char* ptr = &value;
char** ptr2 = &ptr;
char*** ptr3 = &ptr2;
...

flush(ptr);
flush(ptr2);
flush(ptr3);
...

if((x + ***ptr3) < len)
  oracle[data[x]*4096];
```



Maximal speculation window size: 57 instructions at a chain length of 1

Results: TLB miss

```
char* addr = malloc(...);
*addr = 0;
...
```

Results: TLB miss

```
char* addr = malloc(...);
*addr = 0;
...
flush_tlb();
```

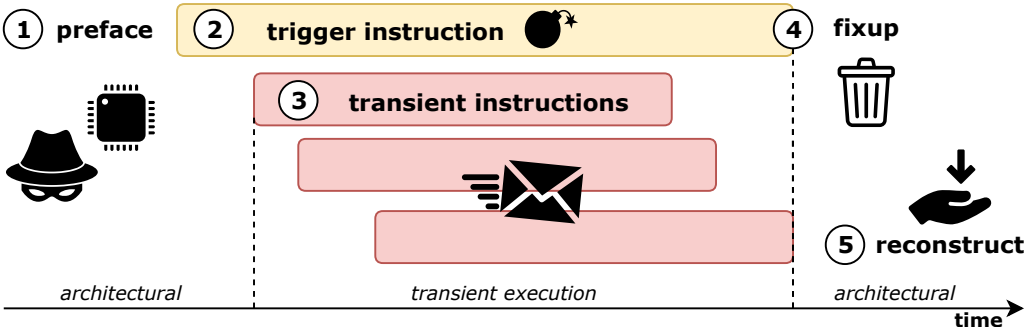
Results: TLB miss

```
char* addr = malloc(...);
*addr = 0;
...
flush_tlb();
if((x + *addr) < len)
 oracle[data[x] * 4096];
```

Results: TLB miss

```
char* addr = malloc(...);
*addr = 0;
...
flush_tlb();
if((x + *addr) < len)
 oracle[data[x] * 4096];
```

Average TPR	0.97
Speculation window size	57 instructions


1 preface
2 trigger instruction
3 transient instructions
4 fixup
5 reconstruct
architectural
transient execution
architectural
time

Attack
Covert channel
Meltdown-US/RW/GP/NM/PK
Meltdown-P
Meltdown-BR
Spectre-PHT
Spectre-BTB/RSB
Spectre-STL
NetSpectre

Gadget Classification

Attack	1. Preface
Covert channel	◐ Flush/Prime/Evict
Meltdown-US/RW/GP/NM/PK	● (Exception suppression)
Meltdown-P	○ (L1 prefetch)
Meltdown-BR	-
Spectre-PHT	◐ PHT poisoning
Spectre-BTB/RSB	◐ BTB/RSB poisoning
Spectre-STL	-
NetSpectre	○ Thrash/reset

Gadget Classification

Attack	1. Preface	2. Trigger example
Covert channel	◐ Flush/Prime/Evict	-
Meltdown-US/RW/GP/NM/PK	● (Exception suppression)	● *mov/rdmsr*/FPU
Meltdown-P	○ (L1 prefetch)	● *mov*
Meltdown-BR	-	○ *bound/bndclu*
Spectre-PHT	◐ PHT poisoning	○ *jz*
Spectre-BTB/RSB	◐ BTB/RSB poisoning	○ *call/jmp/ret*
Spectre-STL	-	○ *mov*
NetSpectre	○ Thrash/reset	○ *jz*

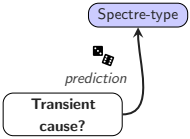
Gadget Classification

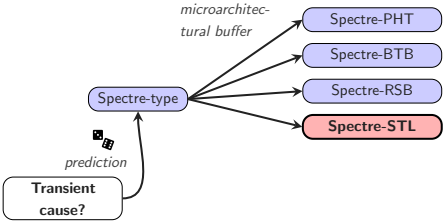
Attack	1. Preface	2. Trigger example	3. Transient
Covert channel	◐ Flush/Prime/Evict	-	◐ Load/AVX/Port/...
Meltdown-US/RW/GP/NM/PK	● (Exception suppression)	● *mov/rdmsr*/FPU	● Controlled encode
Meltdown-P	○ (L1 prefetch)	● *mov*	● Controlled encode
Meltdown-BR	-	○ *bound/bndclu*	○ Inadvertent leak
Spectre-PHT	◐ PHT poisoning	○ *jz*	○ Inadvertent leak
Spectre-BTB/RSB	◐ BTB/RSB poisoning	○ *call/jmp/ret*	○ ROP-style encode
Spectre-STL	-	○ *mov*	○ Inadvertent leak
NetSpectre	○ Thrash/reset	○ *jz*	○ Inadvertent leak

Gadget Classification

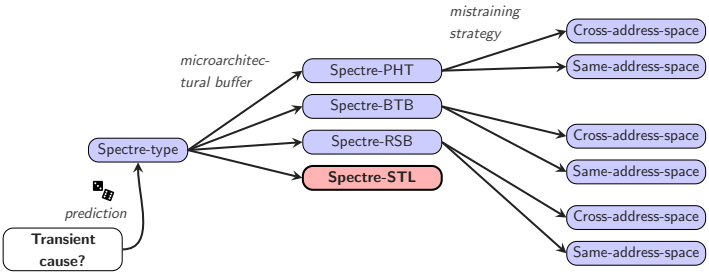
Attack	1. Preface	2. Trigger example	3. Transient	5. Reconstruction
Covert channel	◐ Flush/Prime/Evict	-	◐ Load/AVX/Port/...	◐ Reload/Probe/Time
Meltdown-US/RW/GP/NM/PK	● (Exception suppression)	● *mov/rdmsr/FPU*	● Controlled encode	● Exception handling
Meltdown-P	○ (L1 prefetch)	● *mov*	● Controlled encode	& controlled decode
Meltdown-BR	-	○ *bound/bndclu*	○ Inadvertent leak	*same as above*
Spectre-PHT	◐ PHT poisoning	○ *jz*	○ Inadvertent leak	● Controlled decode
Spectre-BTB/RSB	◐ BTB/RSB poisoning	○ *call/jmp/ret*	○ ROP-style encode	● Controlled decode
Spectre-STL	-	○ *mov*	○ Inadvertent leak	● Controlled decode
NetSpectre	○ Thrash/reset	○ *jz*	○ Inadvertent leak	○ Inadvertent transmit

Transient cause?

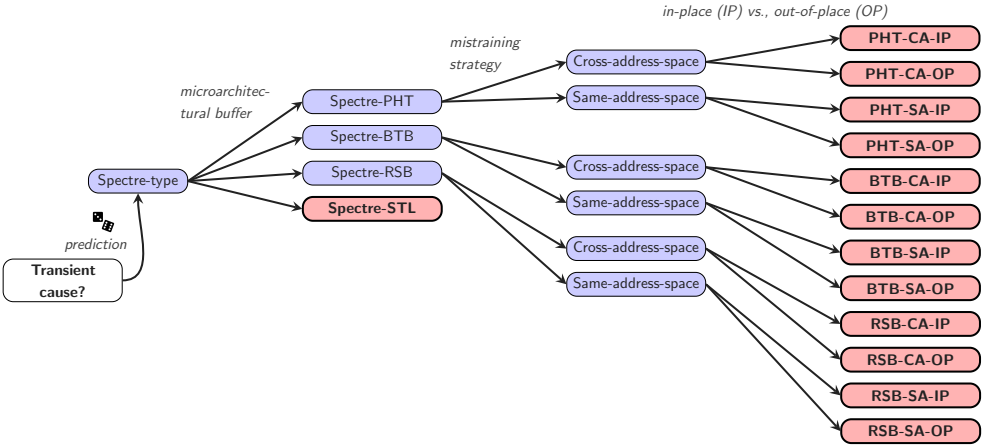

Spectre-type
prediction
Transient cause?

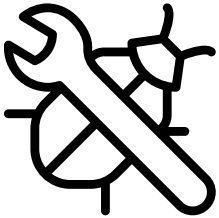

microarchitec-
tural buffer
Spectre-PHT
Spectre-BTB
Spectre-RSB
Spectre-STL
Spectre-type
prediction
Transient
cause?

Spectre Variants

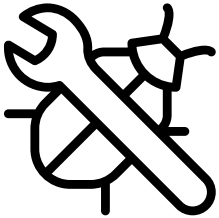


Spectre Variants

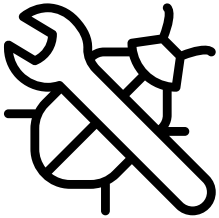




- Spectre is not a bug

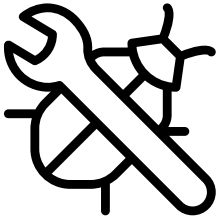


- Spectre is not a bug
- It is an useful optimization



- Spectre is not a bug
- It is an useful optimization

→ Cannot simply fix it (as with Meltdown)

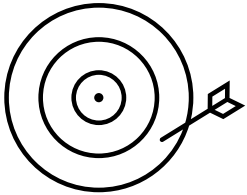


- Spectre is not a bug
- It is an useful optimization

→ Cannot simply fix it (as with Meltdown)

- Workarounds for critical code parts

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

	Attack \ Defense	*InvisiSpec*	*SafeSpec*	*DAWG*	*RSB Stuffing*	*Retpoline*	*Poison Value*	*Index Masking*	*Site Isolation*	*SLH*	*YSNB*	*IBRS*	*STIPB*	*IBPB*	*Serialization*	*Taint Tracking*	*Timer Reduction*	*Sloth*	*SSBD/SSBB*
ARM	Spectre-PHT																		
	Spectre-BTB																		
	Spectre-RSB																		
	Spectre-STL																		

Attack is mitigated (●), partially mitigated (◐), not mitigated (○), theoretically mitigated (■), theoretically impeded (◧), not theoretically impeded (□), or out of scope (◇).

	Attack \ Defense	InvisiSpec	SafeSpec	DAWG	RSB Stuffing	Retpoline	Poison Value	Index Masking	Site Isolation	SLH	YSNB	IBRS	STIPB	IBPB	Serialization	Taint Tracking	Timer Reduction	Sloth	SSBD/SSBB
ARM	Spectre-PHT						●			●									
	Spectre-BTB					●													
	Spectre-RSB																		
	Spectre-STL																		●

Attack is mitigated (●), partially mitigated (◐), not mitigated (○), theoretically mitigated (■), theoretically impeded (◧), not theoretically impeded (□), or out of scope (◇).

	Attack \ Defense	InvisiSpec	SafeSpec	DAWG	RSB Stuffing	Retpoline	Poison Value	Index Masking	Site Isolation	SLH	YSNB	IBRS	STIPB	IBPB	Serialization	Taint Tracking	Timer Reduction	Sloth	SSBD/SSBB
ARM	Spectre-PHT						●	◐	◐	●					◐		◐		
	Spectre-BTB					●			◐								◐		
	Spectre-RSB				◐				◐								◐		
	Spectre-STL								◐								◐		●

Attack is mitigated (●), partially mitigated (◐), not mitigated (○), theoretically mitigated (■), theoretically impeded (◧), not theoretically impeded (□), or out of scope (◇).

	Attack \ Defense	InvisiSpec	SafeSpec	DAWG	RSB Stuffing	Retpoline	Poison Value	Index Masking	Site Isolation	SLH	YSNB	IBRS	STIPB	IBPB	Serialization	Taint Tracking	Timer Reduction	Sloth	SSBD/SSBB
ARM	Spectre-PHT						●	◐	◐	●	○				◐		◐		
	Spectre-BTB					●			◐								◐		
	Spectre-RSB				◐				◐								◐		
	Spectre-STL								◐								◐		●

Attack is mitigated (●), partially mitigated (◐), not mitigated (○), theoretically mitigated (■), theoretically impeded (◧), not theoretically impeded (□), or out of scope (◇).

	Attack \ Defense	InvisiSpec	SafeSpec	DAWG	RSB Stuffing	Retpoline	Poison Value	Index Masking	Site Isolation	SLH	YSNB	IBRS	STIPB	IBPB	Serialization	Taint Tracking	Timer Reduction	Sloth	SSBD/SSBB
ARM	Spectre-PHT						●	◐	◐	●	○				◐	■	◐		
	Spectre-BTB					●			◐							■	◐		
	Spectre-RSB				◐				◐							■	◐		
	Spectre-STL								◐							■	◐	■	●

Attack is mitigated (●), partially mitigated (◐), not mitigated (○), theoretically mitigated (■), theoretically impeded (◧), not theoretically impeded (□), or out of scope (◇).

	Attack \ Defense	InvisiSpec	SafeSpec	DAWG	RSB Stuffing	Retpoline	Poison Value	Index Masking	Site Isolation	SLH	YSNB	IBRS	STIPB	IBPB	Serialization	Taint Tracking	Timer Reduction	Sloth	SSBD/SSBB
ARM	Spectre-PHT						●	◐	◐	●	○				◐	■	◐	◧	
	Spectre-BTB					●			◐							■	◐		
	Spectre-RSB				◐				◐							■	◐		
	Spectre-STL								◐							■	◐	■	●

Attack is mitigated (●), partially mitigated (◐), not mitigated (○), theoretically mitigated (■), theoretically impeded (◧), not theoretically impeded (□), or out of scope (◇).

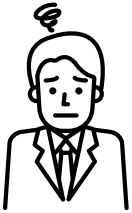
Spectre: Defense Analysis

	Attack \ Defense	InvisiSpec	SafeSpec	DAWG	RSB Stuffing	Retpoline	Poison Value	Index Masking	Site Isolation	SLH	YSNB	IBRS	STIPB	IBPB	Serialization	Taint Tracking	Timer Reduction	Sloth	SSBD/SSBB
ARM	Spectre-PHT	□	□	□			●	◐	◐	●	○				◐	■	◐	◧	
	Spectre-BTB	□	□	□		●			◐							■	◐		
	Spectre-RSB	□	□	□	◐				◐							■	◐		
	Spectre-STL	□	□	□					◐							■	◐	■	●

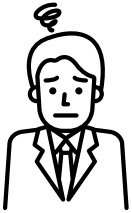
Attack is mitigated (●), partially mitigated (◐), not mitigated (○), theoretically mitigated (■), theoretically impeded (◧), not theoretically impeded (□), or out of scope (◇).

	Attack \ Defense	InvisiSpec	SafeSpec	DAWG	RSB Stuffing	Retpoline	Poison Value	Index Masking	Site Isolation	SLH	YSNB	IBRS	STIPB	IBPB	Serialization	Taint Tracking	Timer Reduction	Sloth	SSBD/SSBB
ARM	Spectre-PHT	□	□	□	◇	◇	●	◐	◐	●	○	◇	◇	◇	◐	■	◐	◧	◇
	Spectre-BTB	□	□	□	◇	●	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	◇	◇
	Spectre-RSB	□	□	□	◐	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	◇	◇
	Spectre-STL	□	□	□	◇	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	■	●

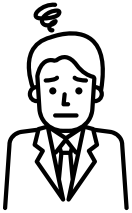
Attack is mitigated (●), partially mitigated (◐), not mitigated (○), theoretically mitigated (■), theoretically impeded (◧), not theoretically impeded (□), or out of scope (◇).



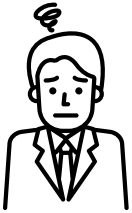
- Many countermeasures only consider the cache to get data...



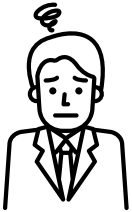
- Many countermeasures only consider the cache to get data...
- ...but there are other possibilities, e.g.,



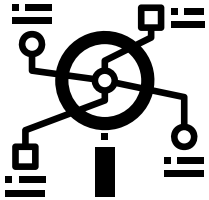
- Many countermeasures only consider the cache to get data...
- ...but there are other possibilities, e.g.,
  - Port contention (SMoTherSpectre)



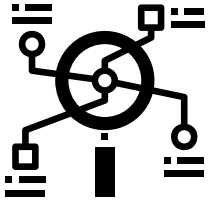
- Many countermeasures only consider the cache to get data...
- ...but there are other possibilities, e.g.,
  - Port contention (SMoTherSpectre)
  - AVX (NetSpectre)



- Many countermeasures only consider the cache to get data...
- ...but there are other possibilities, e.g.,
  - Port contention (SMoTherSpectre)
  - AVX (NetSpectre)
- Cache is just the easiest

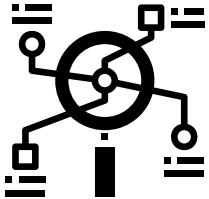


- Current mitigations are either incomplete or cost performance



- Current mitigations are either incomplete or cost performance

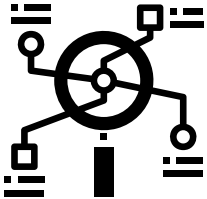
→ More research required



- Current mitigations are either incomplete or cost performance

→ More research required

- Both on attacks and defenses

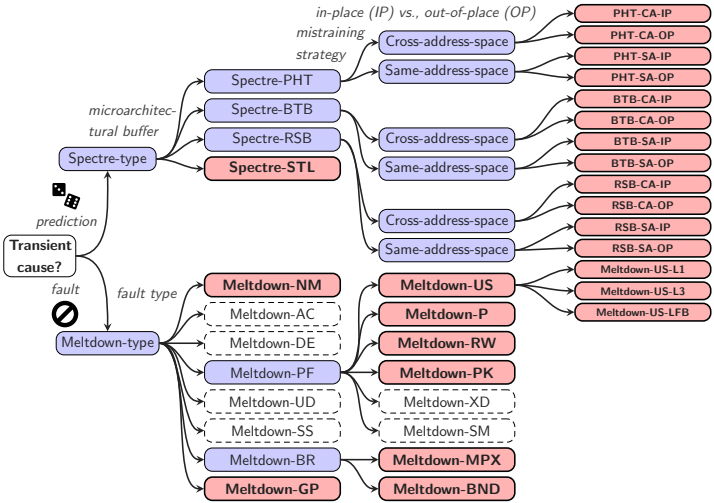


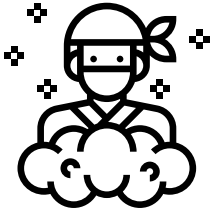
- Current mitigations are either incomplete or cost performance

→ More research required

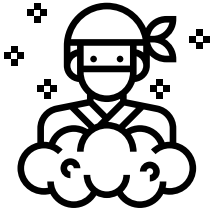
- Both on attacks and defenses

→ Efficient defenses only possible when attacks are known

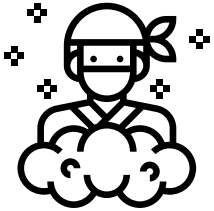

Transient cause?
prediction
fault
Spectre-type
microarchitec-tural buffer
Spectre-PHT
Spectre-BTB
Spectre-RSB
Spectre-STL
mistraining strategy
in-place (IP) vs., out-of-place (OP)
Cross-address-space
Same-address-space
PHT-CA-IP
PHT-CA-OP
PHT-SA-IP
PHT-SA-OP
BTB-CA-IP
BTB-CA-OP
BTB-SA-IP
BTB-SA-OP
RSB-CA-IP
RSB-CA-OP
RSB-SA-IP
RSB-SA-OP
Meltdown-type
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-PK
Meltdown-XD
Meltdown-SM
Meltdown-MPX
Meltdown-BND
Meltdown-US-L1
Meltdown-US-L3
Meltdown-US-LFB



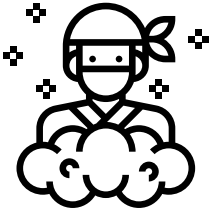
- Transient Execution Attacks are...



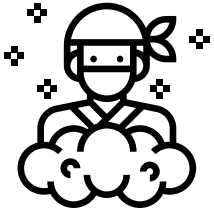
- Transient Execution Attacks are...
  - ...a novel class of attacks



- Transient Execution Attacks are...
  - ...a novel class of attacks
  - ...extremely powerful



- Transient Execution Attacks are...
  - ...a novel class of attacks
  - ...extremely powerful
  - ...only at the beginning



- Transient Execution Attacks are...
  - ...a novel class of attacks
  - ...extremely powerful
  - ...only at the beginning
- Many optimizations introduce side channels → now exploitable

Transient Execution Attacks: Still hARMful?

Barbara Gigerl (@barbarag2112), Claudio Canella (@cc0x1f)

May 17, 2019

Graz University of Technology