Side-Channel Lab II

Michael Schwarz

Security Week Graz 2019

Covert channel

What is a covert channel?

- Two programs would like to communicate

Covert channel

What is a covert channel?

- Two programs would like to communicate but are not allowed to do so

Covert channel

What is a covert channel?

- Two programs would like to communicate but are not allowed to do so
  - either because there is no communication channel...

Covert channel

What is a covert channel?

- Two programs would like to communicate but are not allowed to do so
  - either because there is no communication channel...
  - ...or the channels are monitored and programs are stopped on communication attempts

Covert channel

What is a covert channel?

- Two programs would like to communicate but are not allowed to do so
  - either because there is no communication channel...
  - ...or the channels are monitored and programs are stopped on communication attempts
- Use side channels and stay stealthy

Covert channel

Covert channel

State of the art

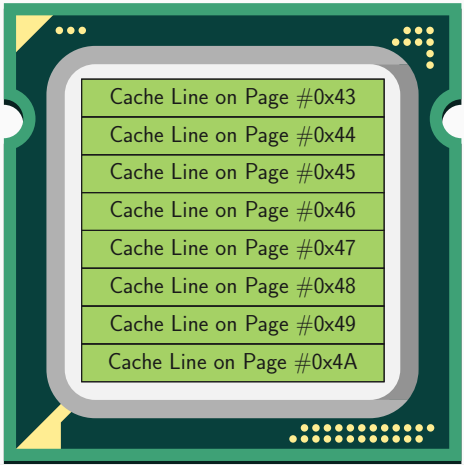
method	raw capacity	err. rate	true capacity	env.
F+F [Gru+16]	3968Kbps	0.840%	3690Kbps	native
F+R [Gru+16]	2384Kbps	0.005%	2382Kbps	native
E+R [Lip+16]	1141Kbps	1.100%	1041Kbps	native
P+P [Mau+17]	601Kbps	0.000%	601Kbps	native
P+P [Liu+15]	600Kbps	1.000%	552Kbps	virt
P+P [Mau+17]	362Kbps	0.000%	362Kbps	native

Sending Data (easy but inefficient)

Sender

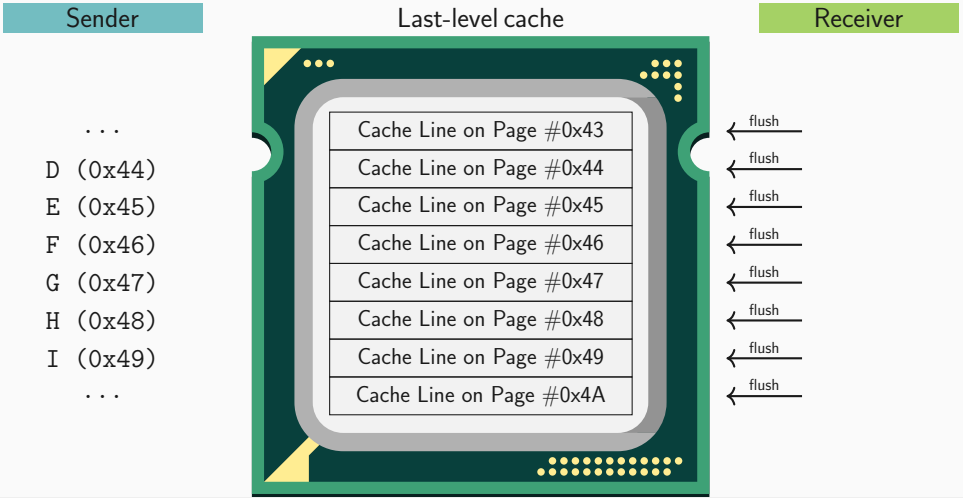
```
...
D (0x44)
E (0x45)
F (0x46)
G (0x47)
H (0x48)
I (0x49)
...
```

Last-level cache



Receiver

Sending Data (easy but inefficient)

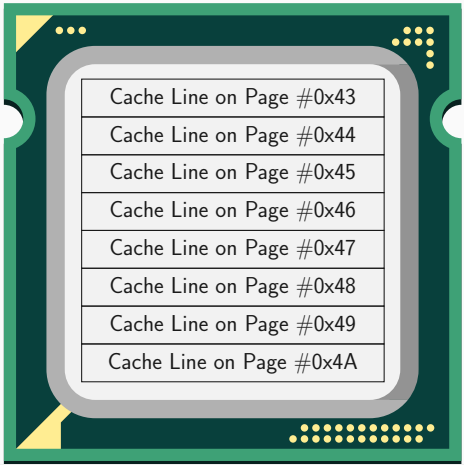


Sending Data (easy but inefficient)

Sender

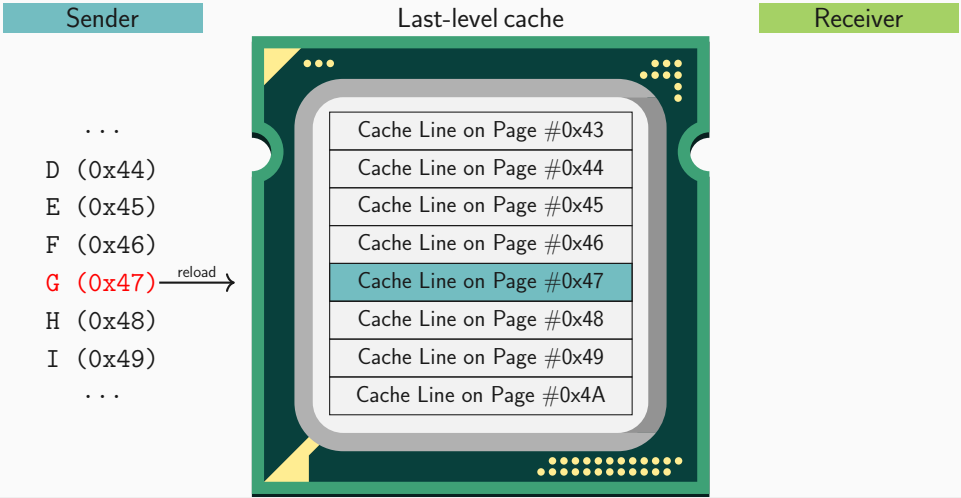
```
...
D (0x44)
E (0x45)
F (0x46)
G (0x47)
H (0x48)
I (0x49)
...
```

Last-level cache



Receiver

Sending Data (easy but inefficient)

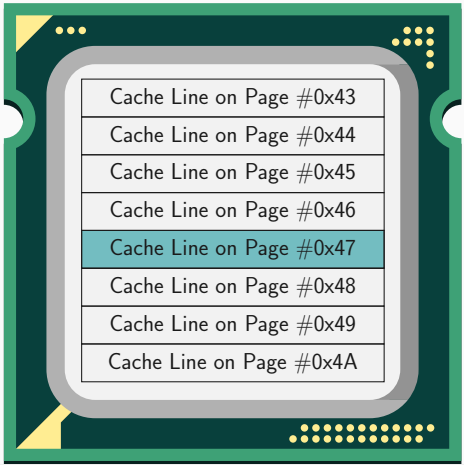


Sending Data (easy but inefficient)

Sender

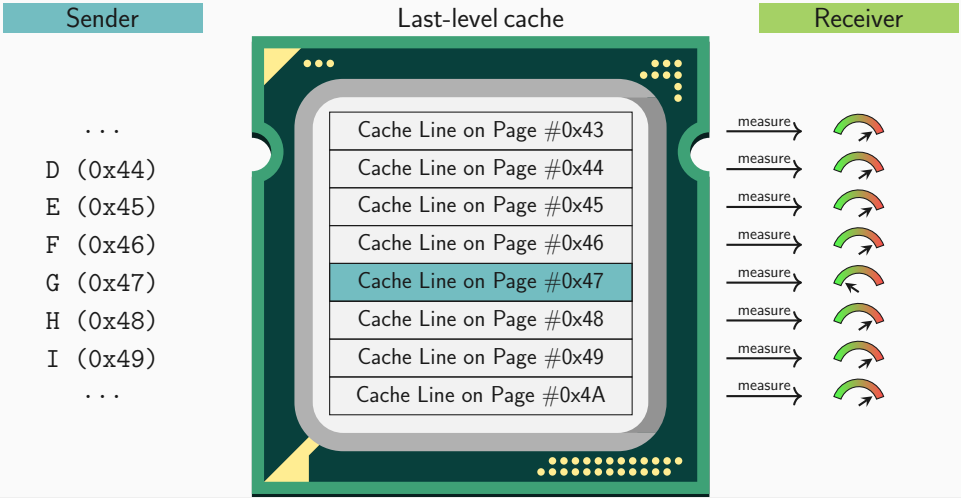
```
...
D (0x44)
E (0x45)
F (0x46)
G (0x47)
H (0x48)
I (0x49)
...
```

Last-level cache

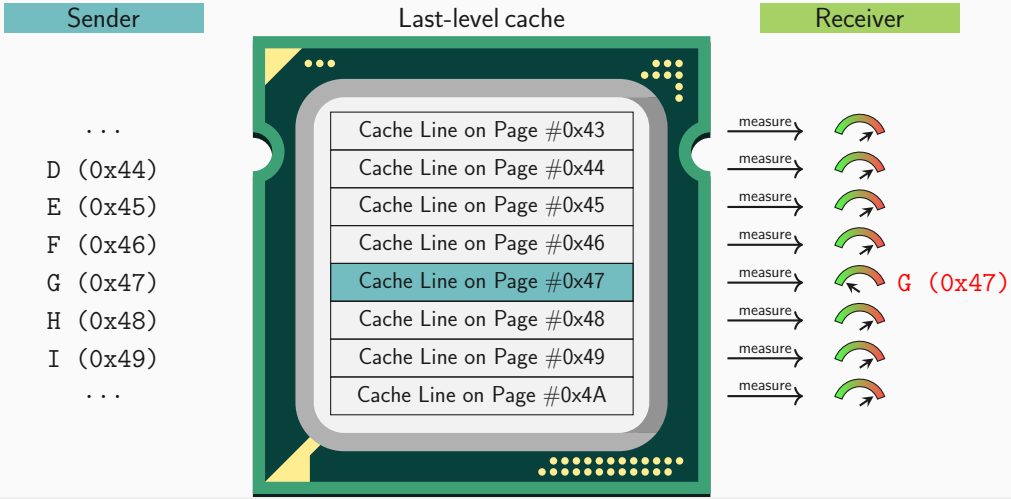


Receiver

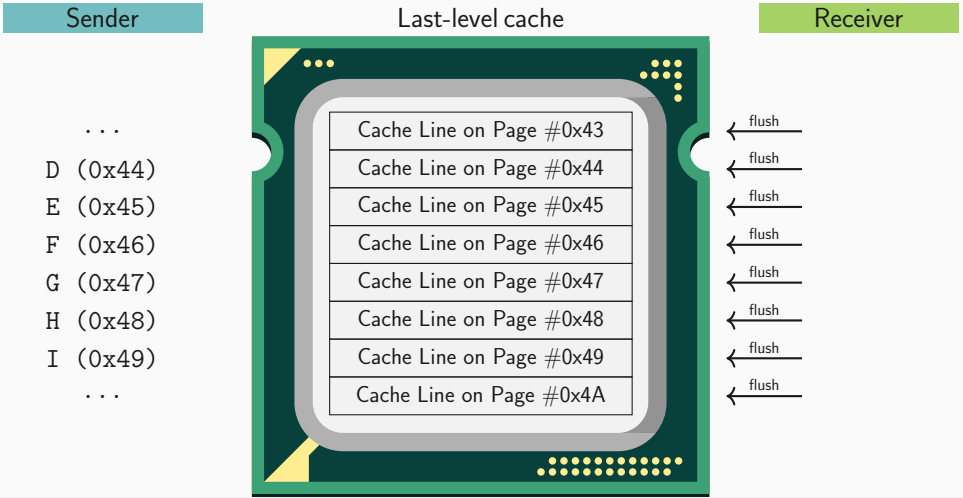
Sending Data (easy but inefficient)



Sending Data (easy but inefficient)



Sending Data (easy but inefficient)

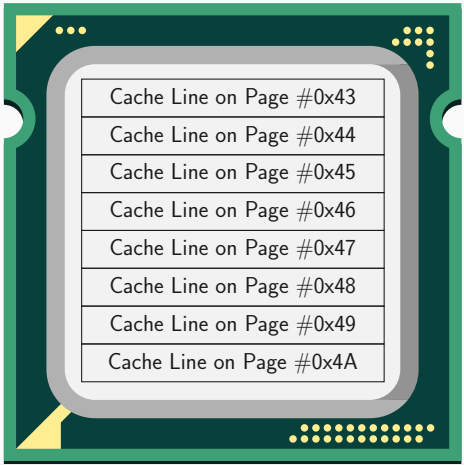


Sending Data (easy but inefficient)

Sender

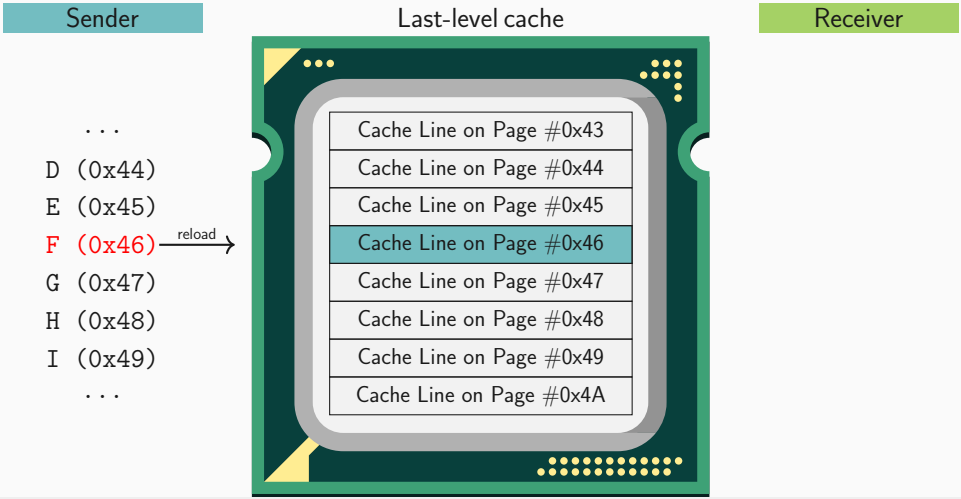
```
...
D (0x44)
E (0x45)
F (0x46)
G (0x47)
H (0x48)
I (0x49)
...
```

Last-level cache

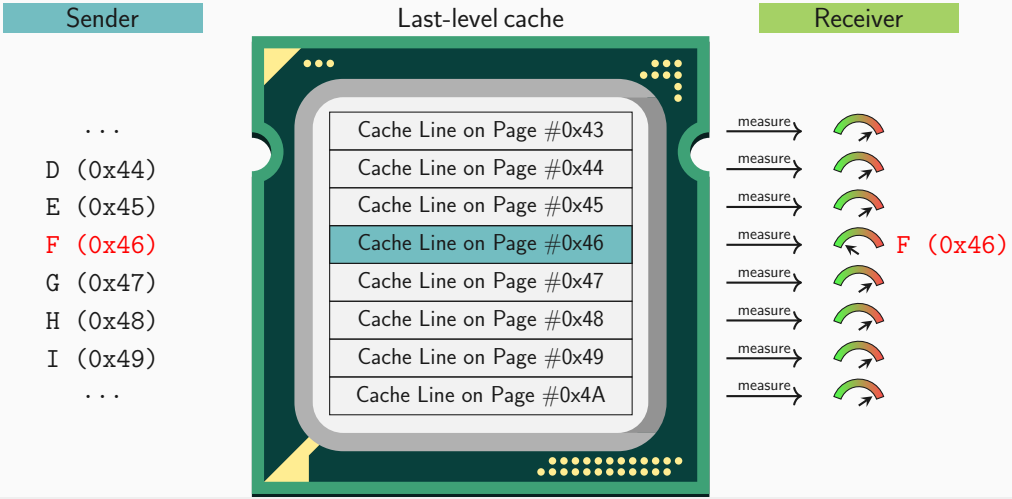


Receiver

Sending Data (easy but inefficient)



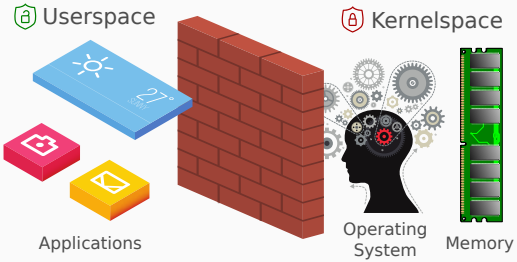
Sending Data (easy but inefficient)



Time to code

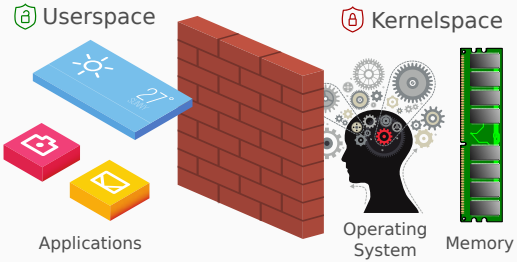
Operating Systems 101

Memory Isolation



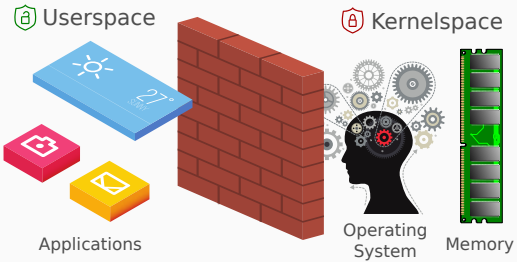
- Kernel is isolated from user space

Memory Isolation



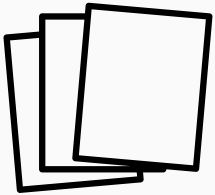
- Kernel is isolated from user space
- This isolation is a combination of hardware and software

Memory Isolation



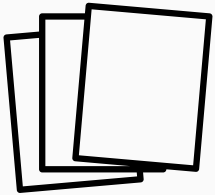
- Kernel is isolated from user space
- This isolation is a combination of hardware and software
- User applications cannot access anything from the kernel

Paging



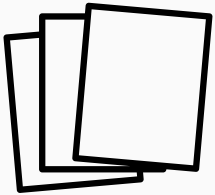
- CPU support virtual address spaces to isolate processes

Paging



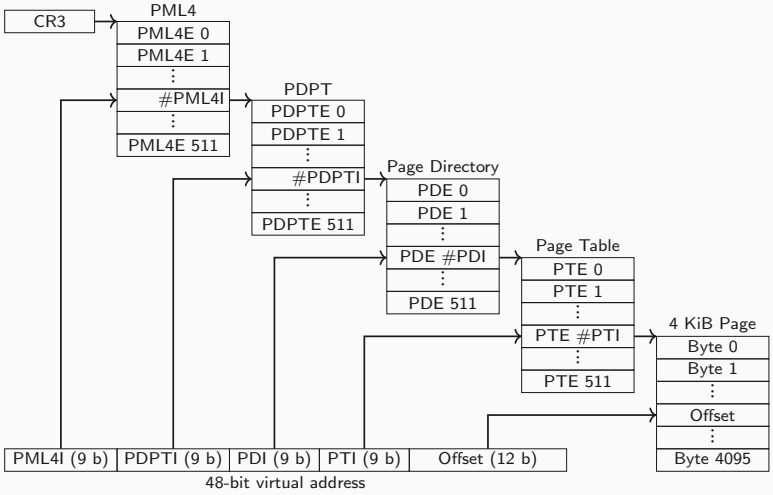
- CPU support virtual address spaces to isolate processes
- Physical memory is organized in page frames

Paging

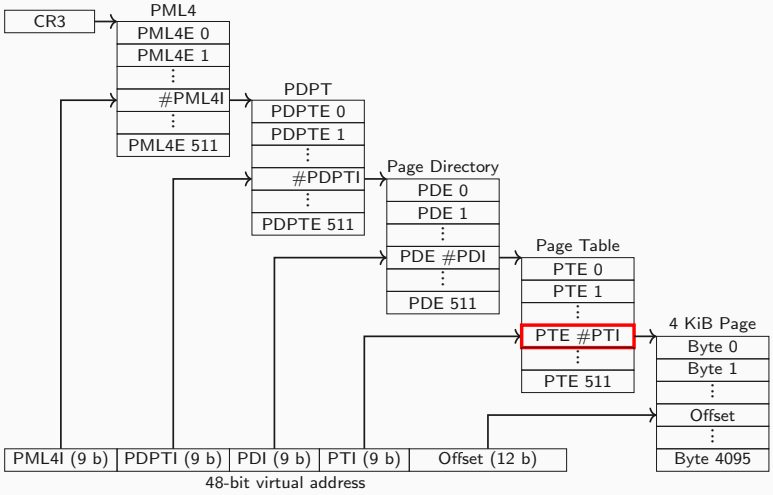


- CPU support virtual address spaces to isolate processes
- Physical memory is organized in page frames
- Virtual memory pages are mapped to page frames using page tables

Address Translation on x86-64



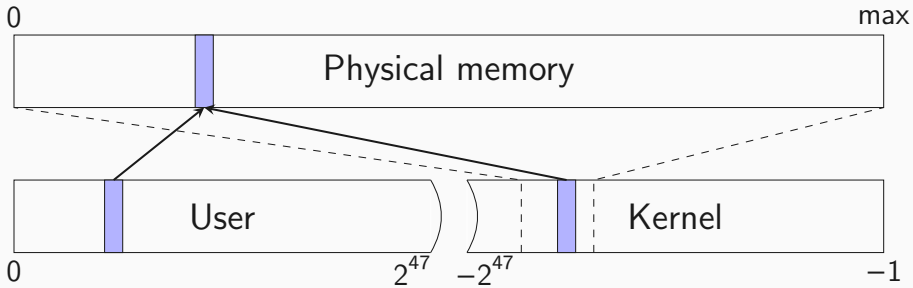
Address Translation on x86-64



Page Table Entry

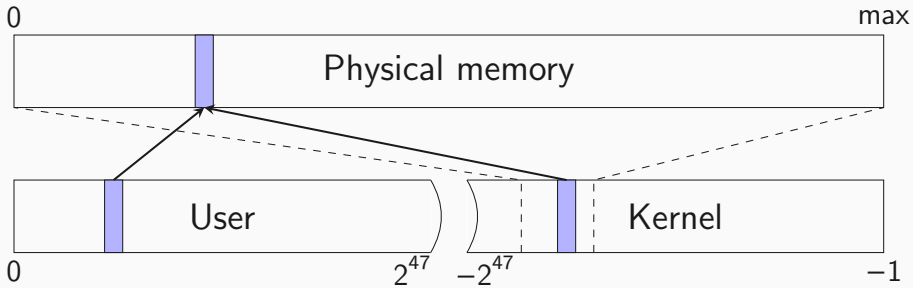
- User/Supervisor bit defines in which privilege level the page can be accessed

Direct-physical map



- Kernel is typically mapped into every address space

Direct-physical map

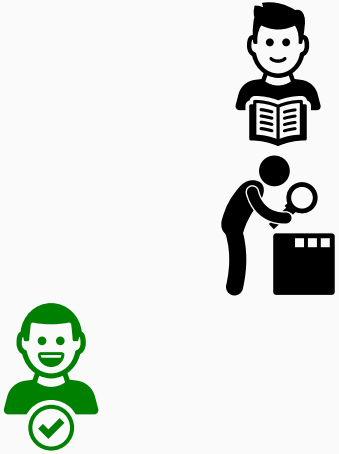


- Kernel is typically mapped into every address space
- Entire physical memory is mapped in the kernel

Loading an address

Loading an address

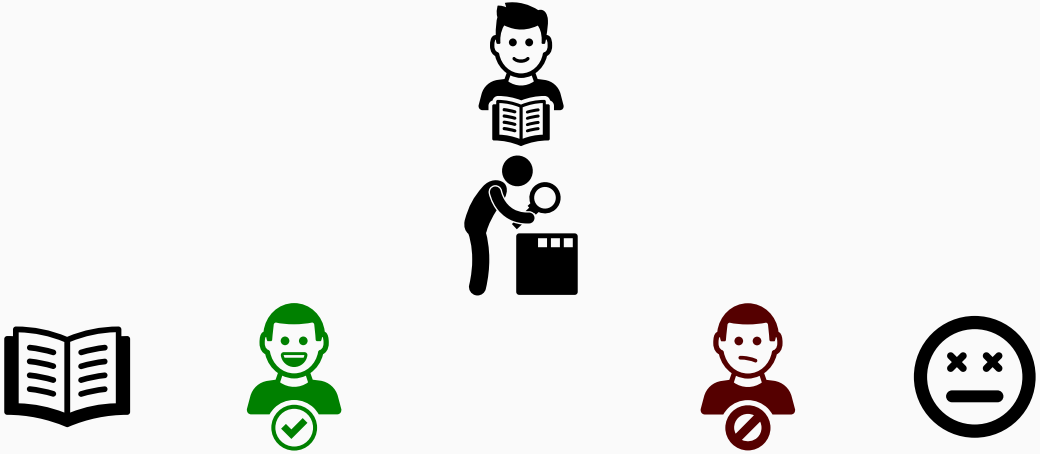
Loading an address



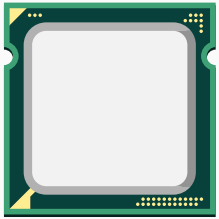
Loading an address

Loading an address

Loading an address

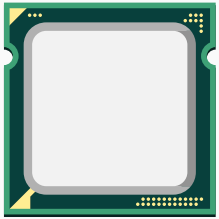


Architecture and Microarchitecture



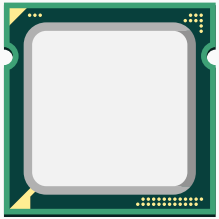
- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)

Architecture and Microarchitecture



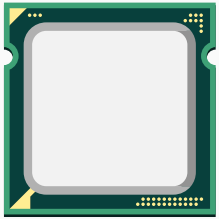
- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)
- Serves as the interface between hardware and software

Architecture and Microarchitecture



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

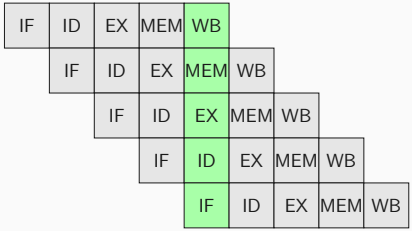
Architecture and Microarchitecture



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

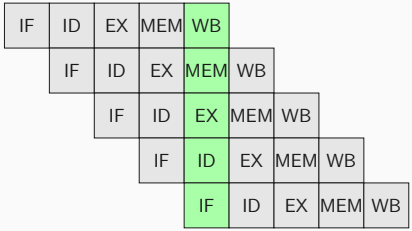
●●●

In-Order Execution



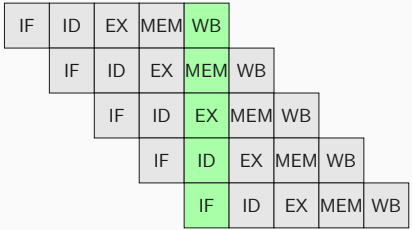
- Instructions are...
  - fetched (`IF`) from the L1 Instruction Cache

In-Order Execution



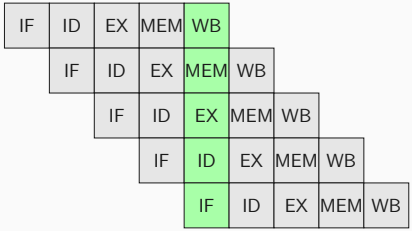
- Instructions are...
  - fetched (`IF`) from the L1 Instruction Cache
  - decoded (`ID`)

In-Order Execution



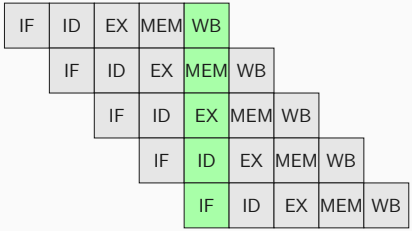
- Instructions are...
  - fetched (`IF`) from the L1 Instruction Cache
  - decoded (`ID`)
  - executed (`EX`) by execution units

In-Order Execution



- Instructions are...
  - fetched (`IF`) from the L1 Instruction Cache
  - decoded (`ID`)
  - executed (`EX`) by execution units
- Memory access is performed (`MEM`)

In-Order Execution



- Instructions are...
  - fetched (`IF`) from the L1 Instruction Cache
  - decoded (`ID`)
  - executed (`EX`) by execution units
- Memory access is performed (`MEM`)
- Architectural register file is updated (`WB`)

In-Order Execution

- Instructions are executed in-order

In-Order Execution

- Instructions are executed in-order
- Pipeline stalls when stages are not ready

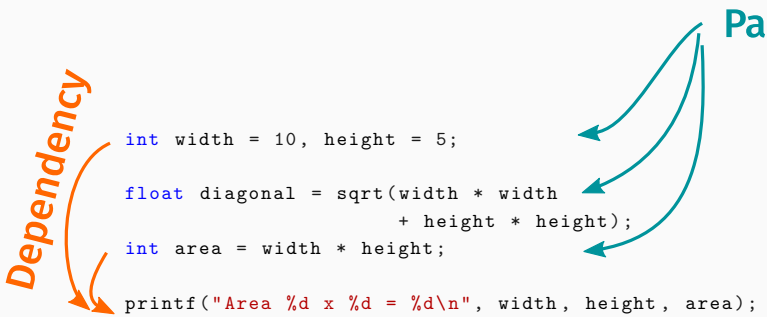
In-Order Execution

- Instructions are executed in-order
- Pipeline stalls when stages are not ready
- If data is not cached, we need to wait

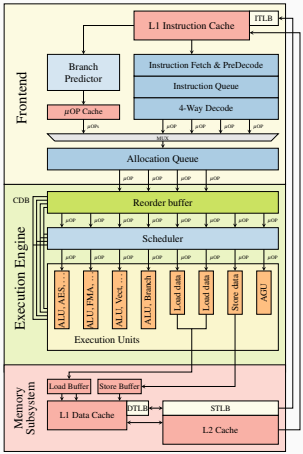
Out-of-order Execution


```
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);
```

Out-of-order Execution



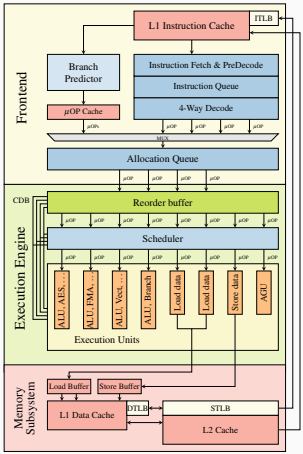
Out-of-Order Execution



Instructions are

- fetched and decoded in the front-end

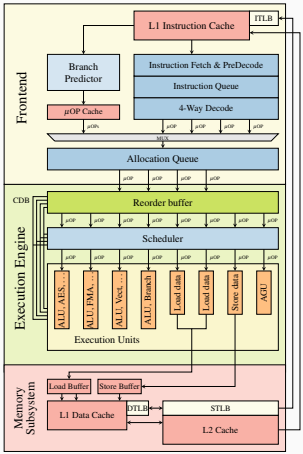
Out-of-Order Execution



Instructions are

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

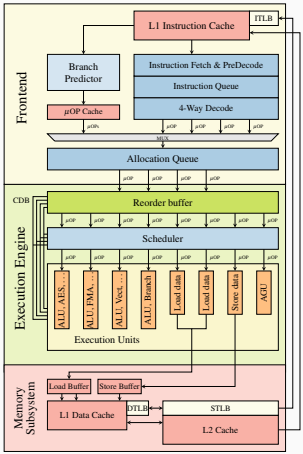
Out-of-Order Execution



Instructions are

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

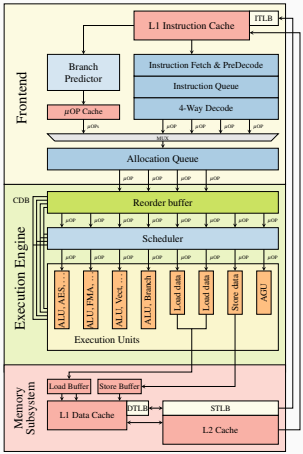
Out-of-Order Execution



Instructions

- are executed out-of-order

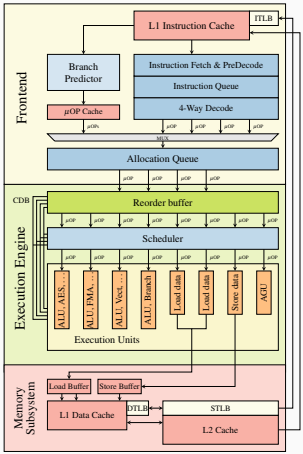
Out-of-Order Execution



Instructions

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

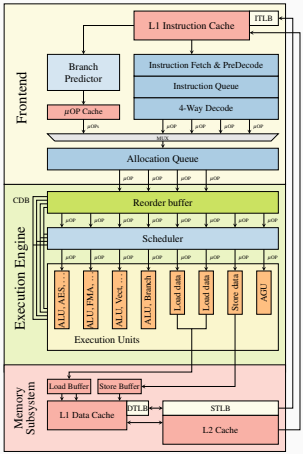
Out-of-Order Execution



Instructions

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

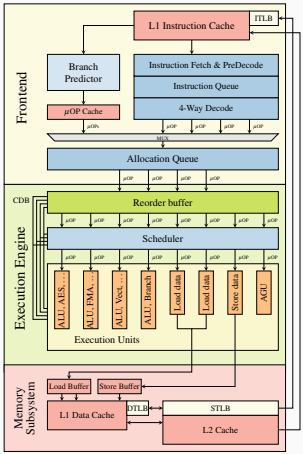
Out-of-Order Execution



Instructions

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

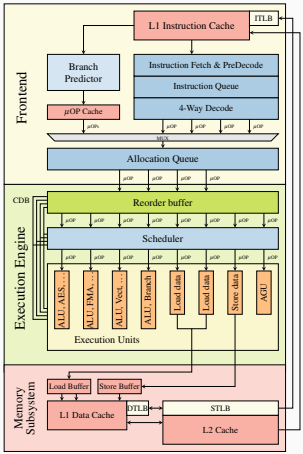
Out-of-Order Execution



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

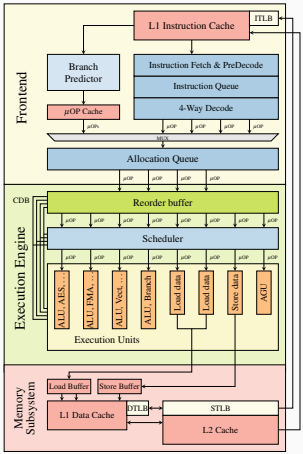
Out-of-Order Execution



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 . . .

The state does not become architecturally visible

but . . .

Getting started...

Getting started...

- New code

```
char data = 'S'; // a "secret" value
// ...
*(volatile char*) 0;
array[data * 4096] = 0;
```

Getting started...

- New code

```
char data = 'S'; // a "secret" value
// ...
*(volatile char*) 0;
array[data * 4096] = 0;
```

- Luckily we know how to catch a segfault

Getting started...

- New code

```
char data = 'S'; // a "secret" value
// ...
*(volatile char*) 0;
array[data * 4096] = 0;
```

- Luckily we know how to catch a segfault
- Then check whether any part of `array` is cached

Checking the array

- Flush+Reload over all pages of the array

Time to code

Meltdown

- Add another layer of indirection to test

```
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```

Meltdown

- Add another layer of indirection to test

```
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```

Which address?

- Check /proc/kallsyms

```
sudo cat /proc/kallsyms | grep banner
```

Which address?

- Check /proc/kallsyms

```
sudo cat /proc/kallsyms | grep banner
```

- or check /proc/pid/pagemap and print address

```
printf("target: %p\n",
    libsc_get_physical_address(ctx, vaddr));
```

Which address?

- Check /proc/kallsyms

```
sudo cat /proc/kallsyms | grep banner
```

- or check /proc/pid/pagemap and print address

```
printf("target: %p\n",
    libsc_get_physical_address(ctx, vaddr));
```

- or start at a random address and iterate

Time to code

Spectre-PHT (aka Spectre Variant 1)

```
index = 0
```

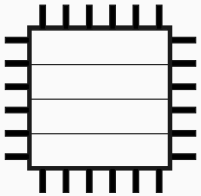
Shared Memory

	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 → `{}`



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)
```

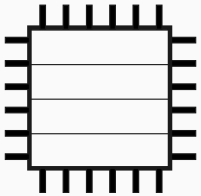
then

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

else

Speculate

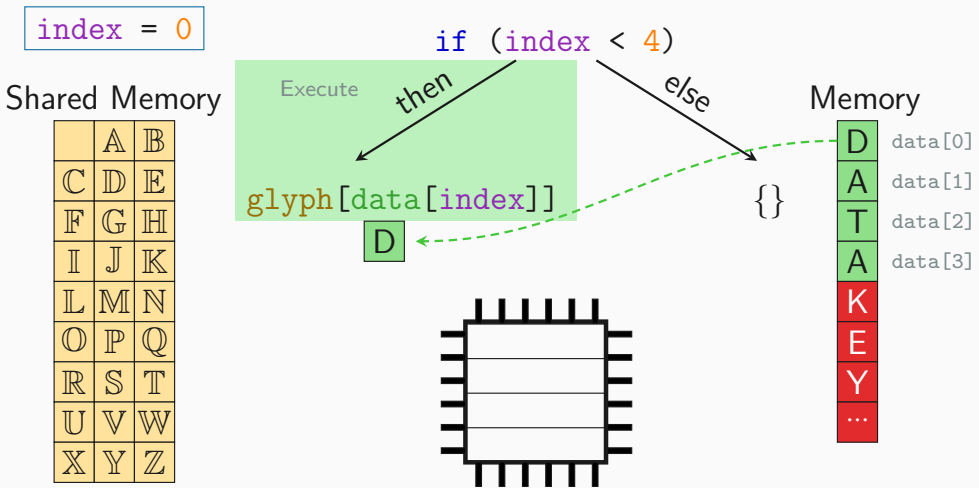
{}



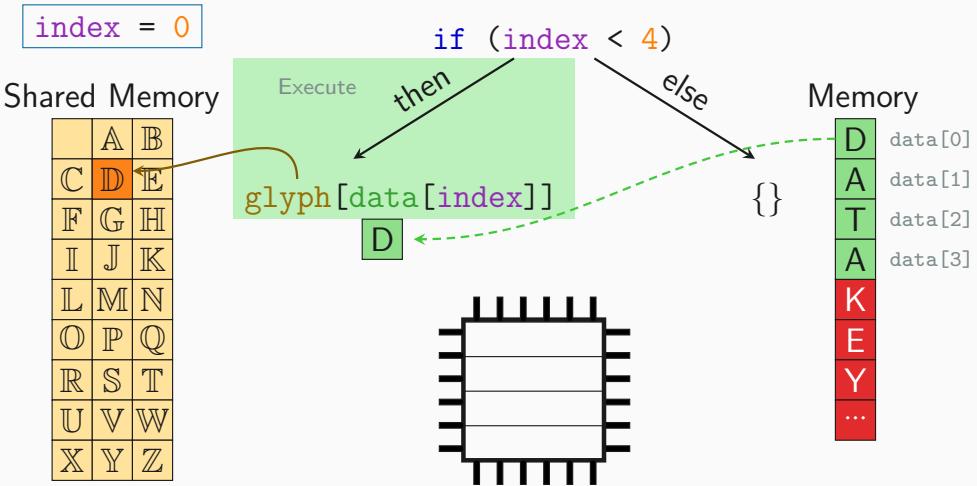
Memory

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

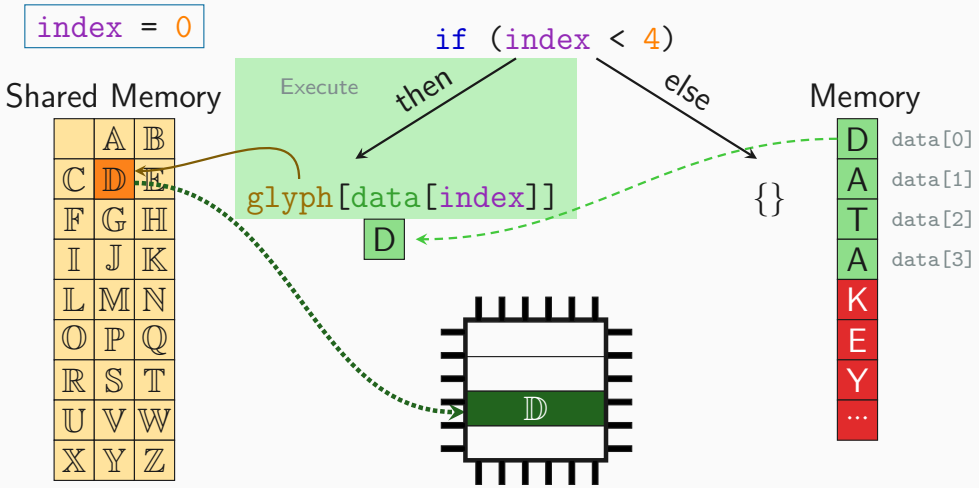
Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)

```
index = 1
```

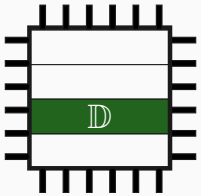
Shared Memory

	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)
```

then

Speculate

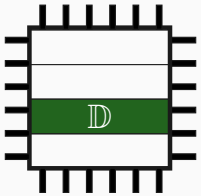
```
glyph[data[index]]
```

else

```
{}
```

Shared Memory

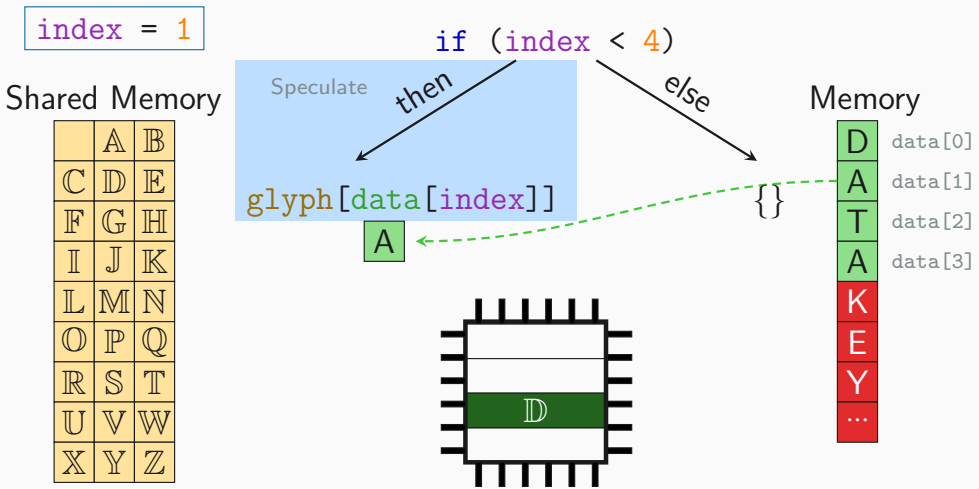
	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z



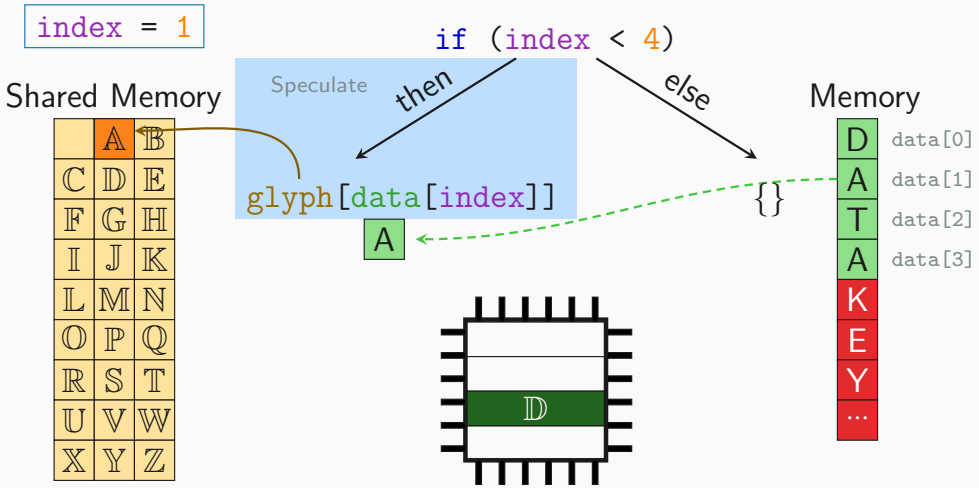
Memory

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

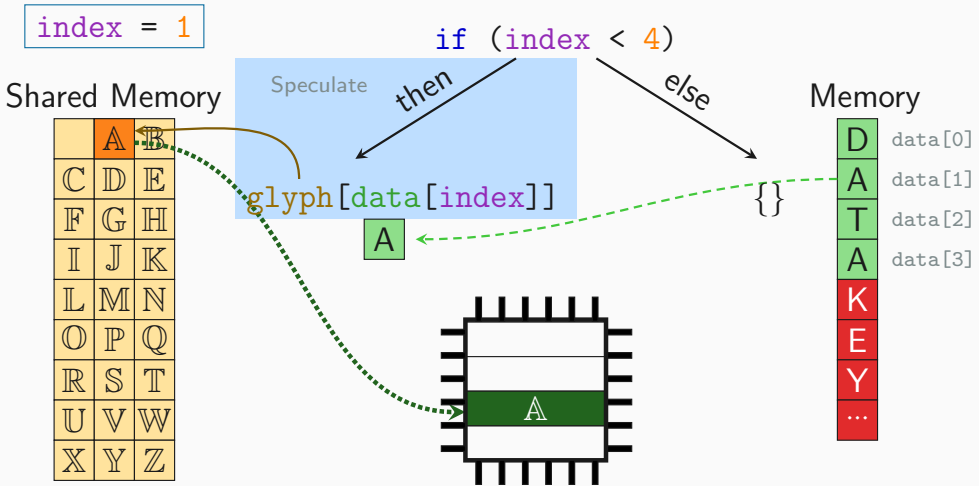
Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)


```
index = 1
```

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

then

Execute

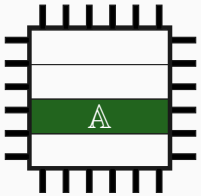
```
glyph[data[index]]
```

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	
...	

Spectre-PHT (aka Spectre Variant 1)

```
index = 2
```

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

Speculate

then

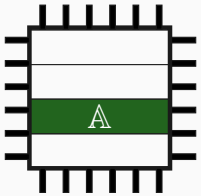
```
glyph[data[index]]
```

else

{}

Shared Memory

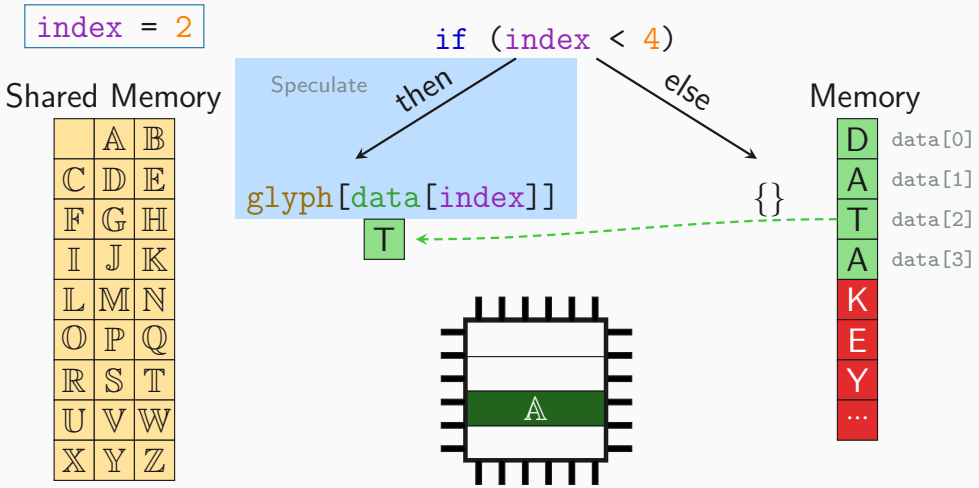
	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z



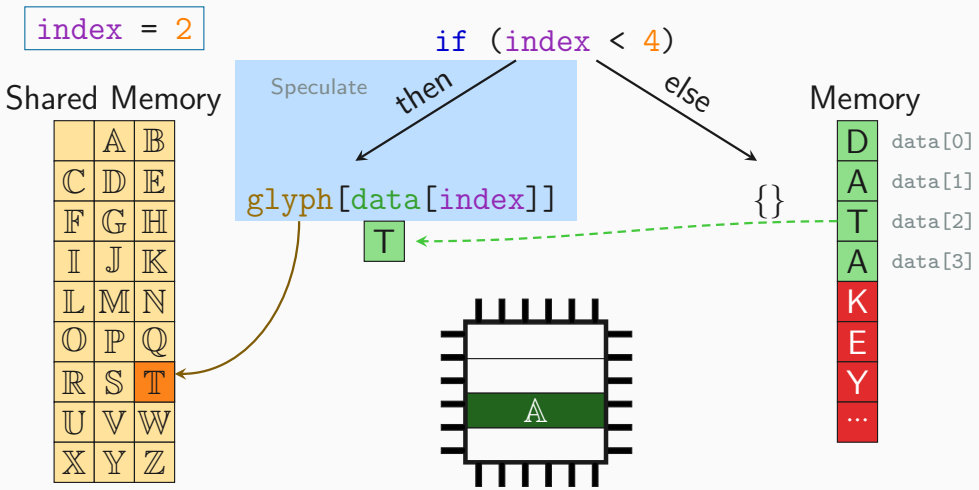
Memory

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

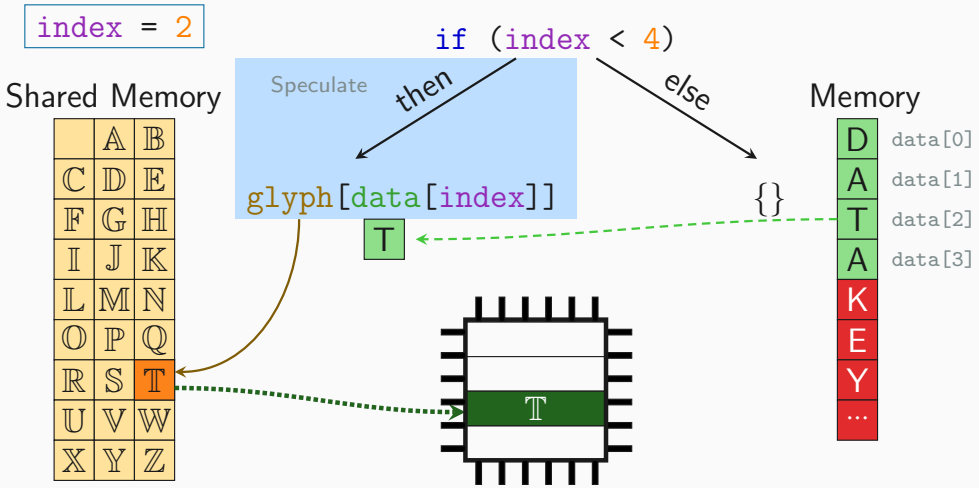
Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)

```
index = 2
```

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

then

Execute

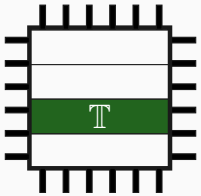
```
glyph[data[index]]
```

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	
...	

Spectre-PHT (aka Spectre Variant 1)

```
index = 3
```

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

Speculate

then

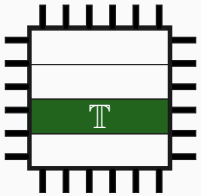
```
glyph[data[index]]
```

else

```
{}
```

Shared Memory

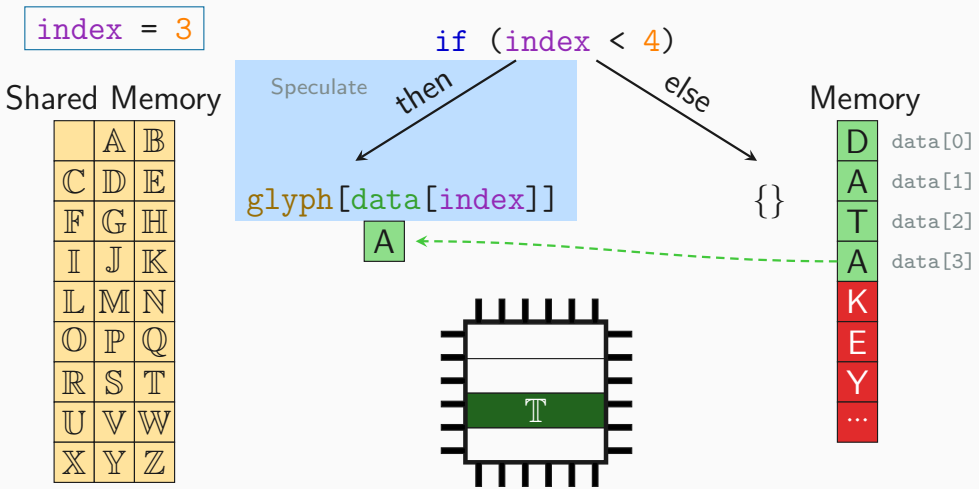
	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z



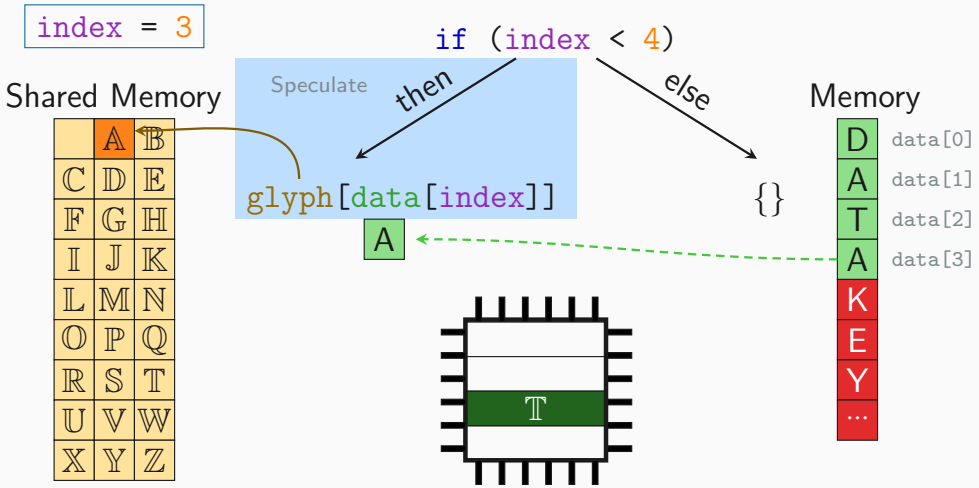
Memory

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

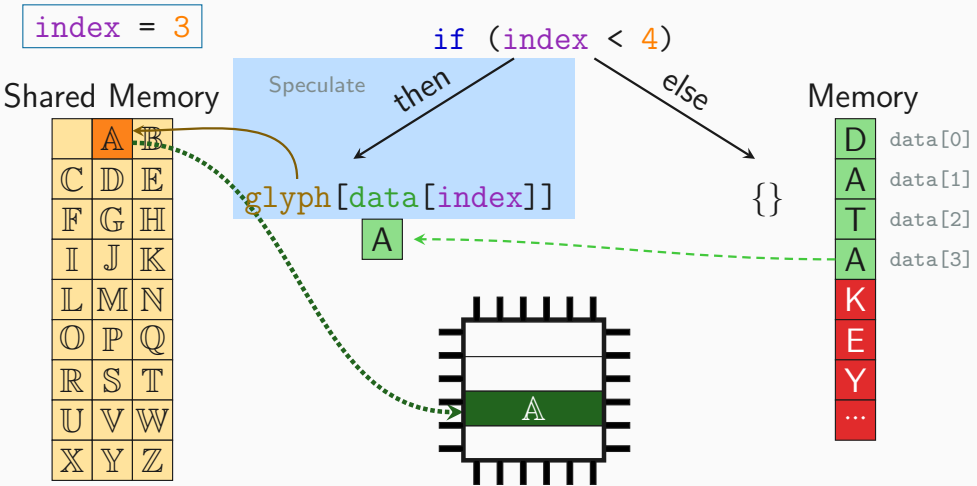
Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)


```
index = 3
```

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

then

Execute

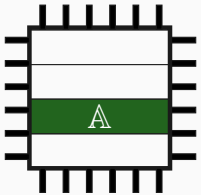
```
glyph[data[index]]
```

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	
...	

Spectre-PHT (aka Spectre Variant 1)

```
index = 4
```

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

then

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

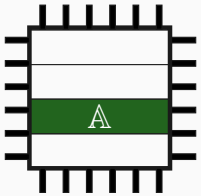
Speculate

else

```
{}
```

Shared Memory

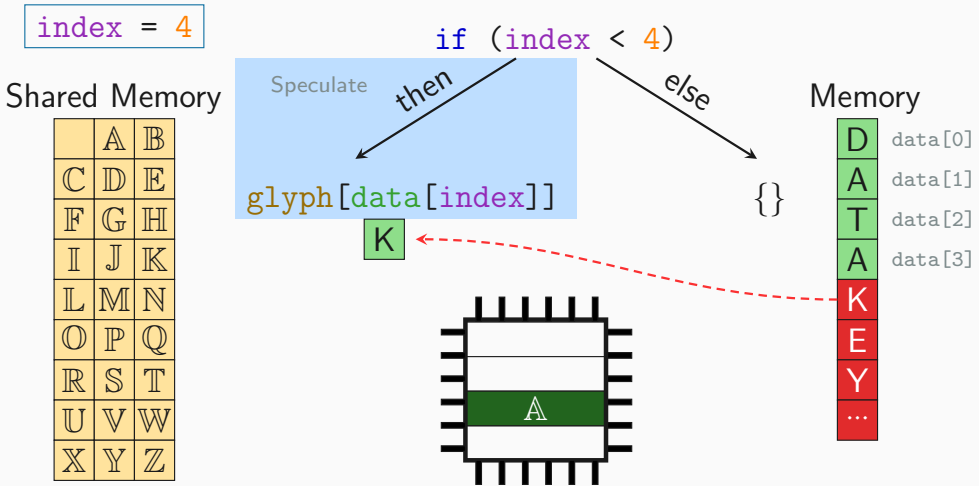
	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z



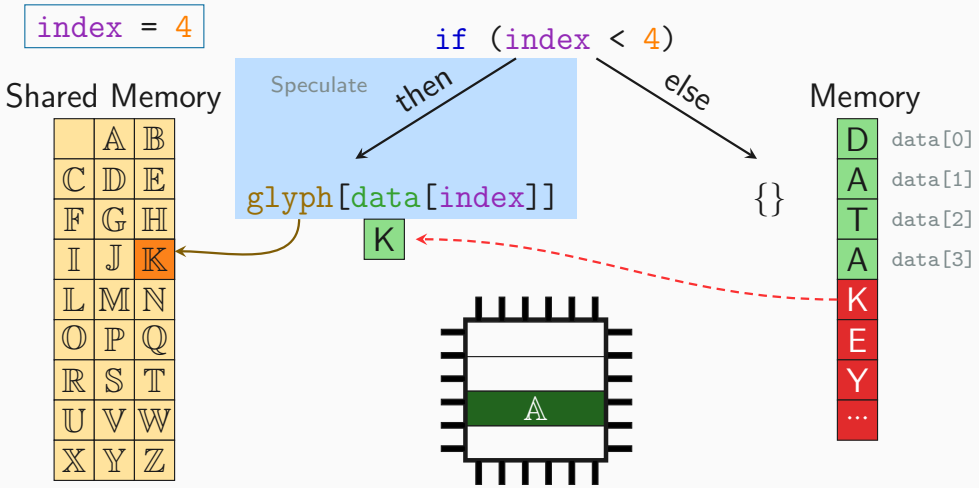
Memory

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

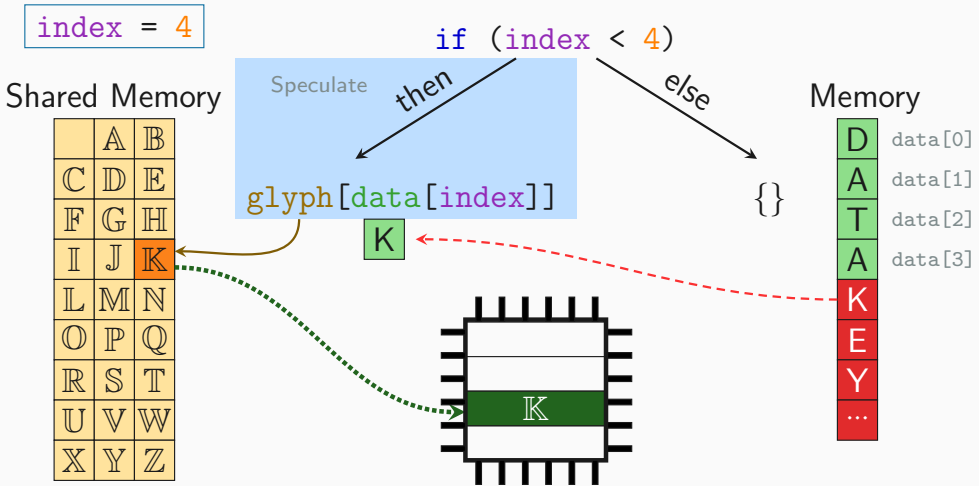
Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)

```
index = 4
```

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

then

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

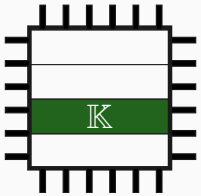
else

Execute

{}

Shared Memory

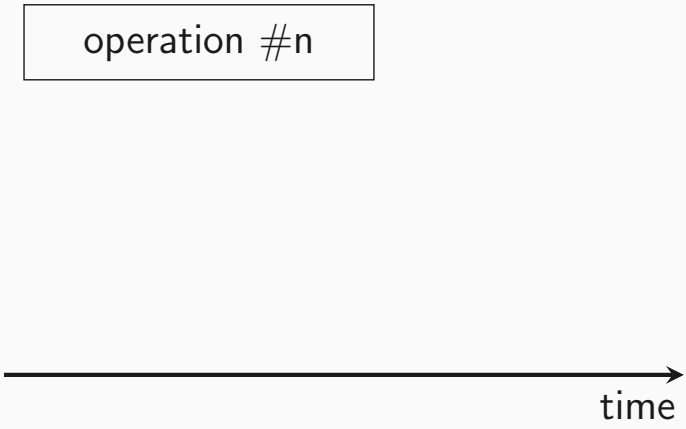
	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z



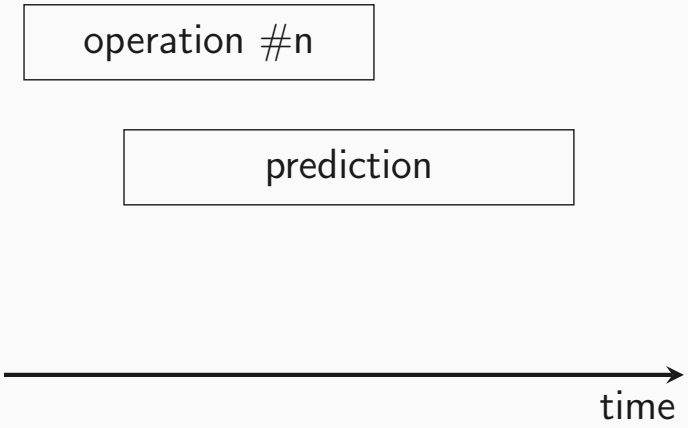
Memory

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

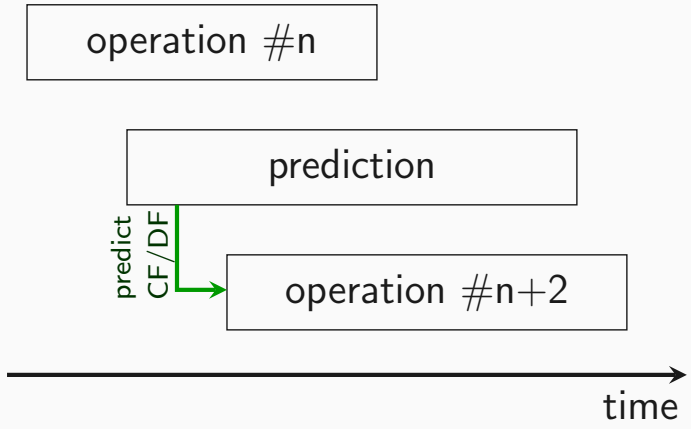
Spectre Root Cause



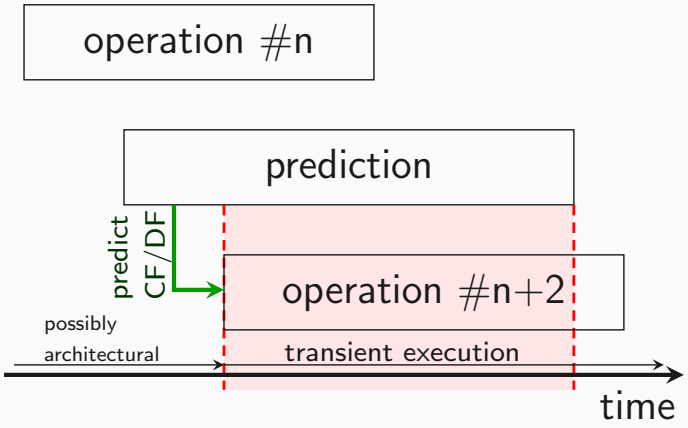
Spectre Root Cause



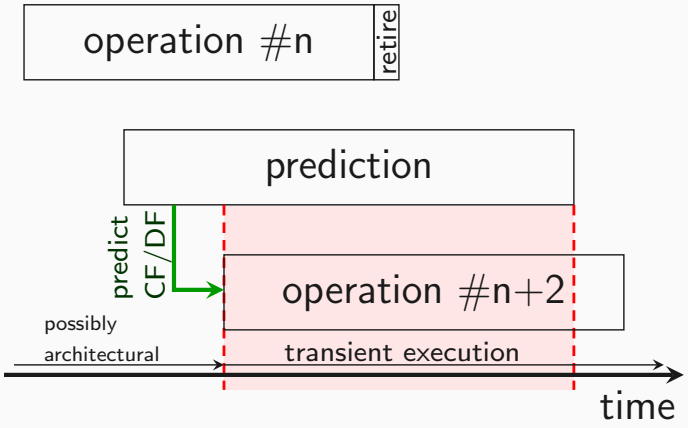
Spectre Root Cause



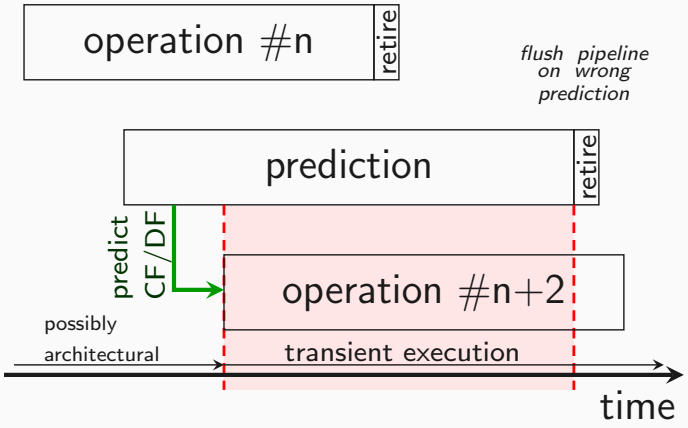
Spectre Root Cause



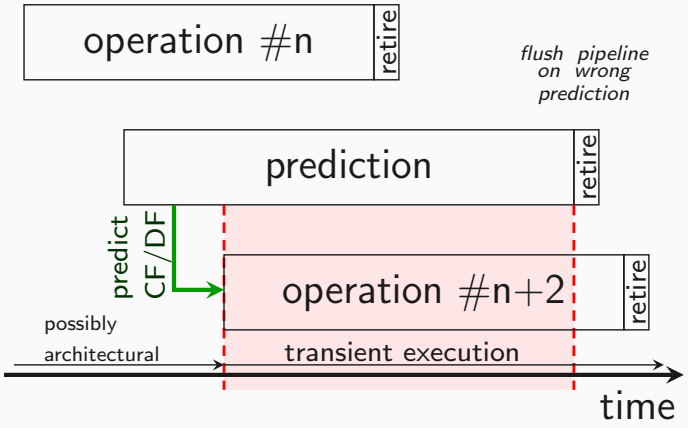
Spectre Root Cause



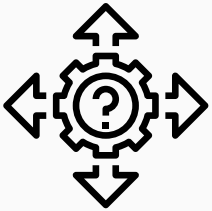
Spectre Root Cause



Spectre Root Cause

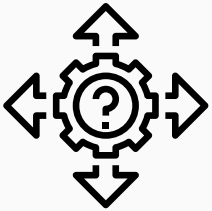


Spectre Root Cause



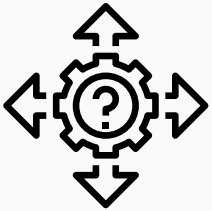
- Many predictors in modern CPUs

Spectre Root Cause



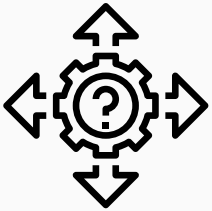
- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)

Spectre Root Cause



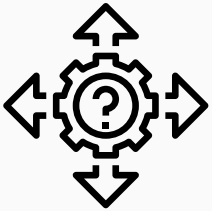
- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)
  - Call/Jump destination (BTB)

Spectre Root Cause



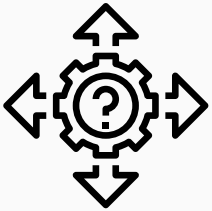
- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)
  - Call/Jump destination (BTB)
  - Function return destination (RSB)

Spectre Root Cause



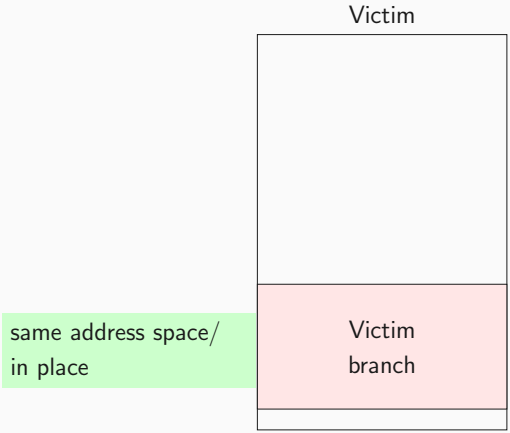
- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)
  - Call/Jump destination (BTB)
  - Function return destination (RSB)
  - Load matches previous store (STL)

Spectre Root Cause

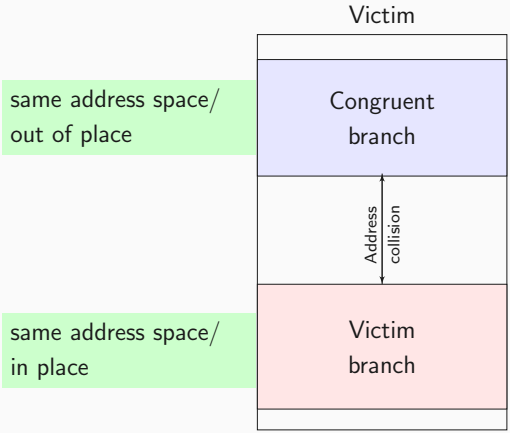


- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)
  - Call/Jump destination (BTB)
  - Function return destination (RSB)
  - Load matches previous store (STL)
- Most are even shared among processes

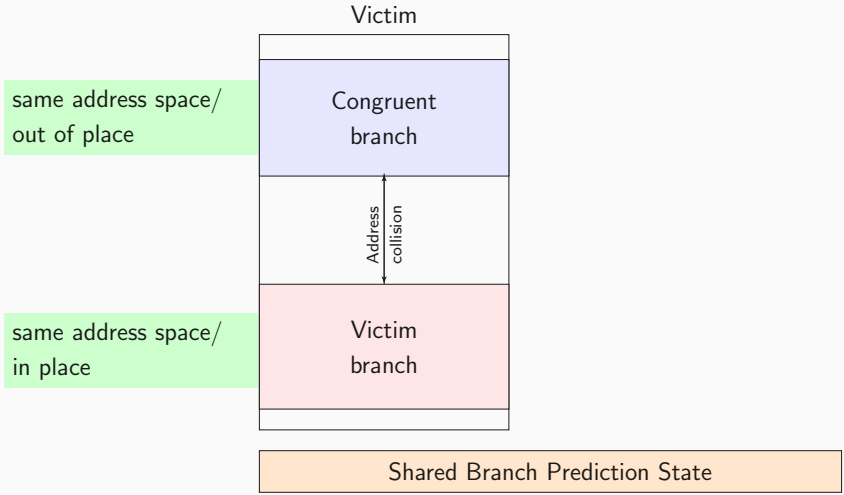
Spectre Mistraining



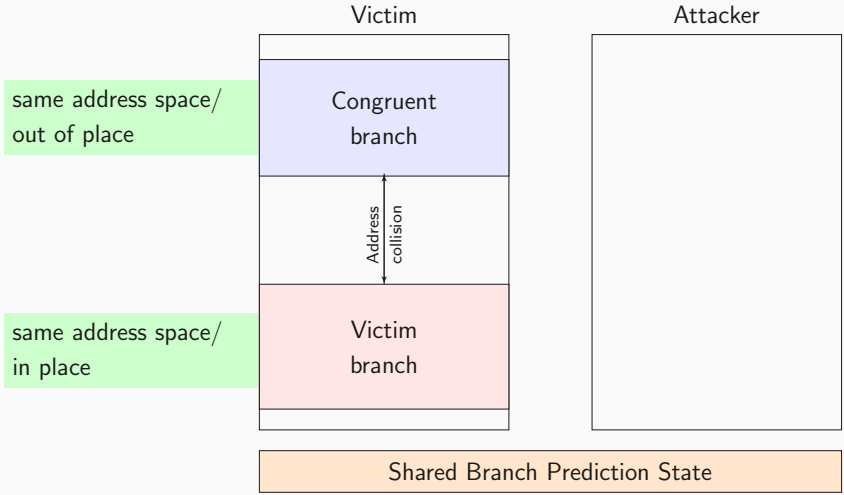
Spectre Mistraining



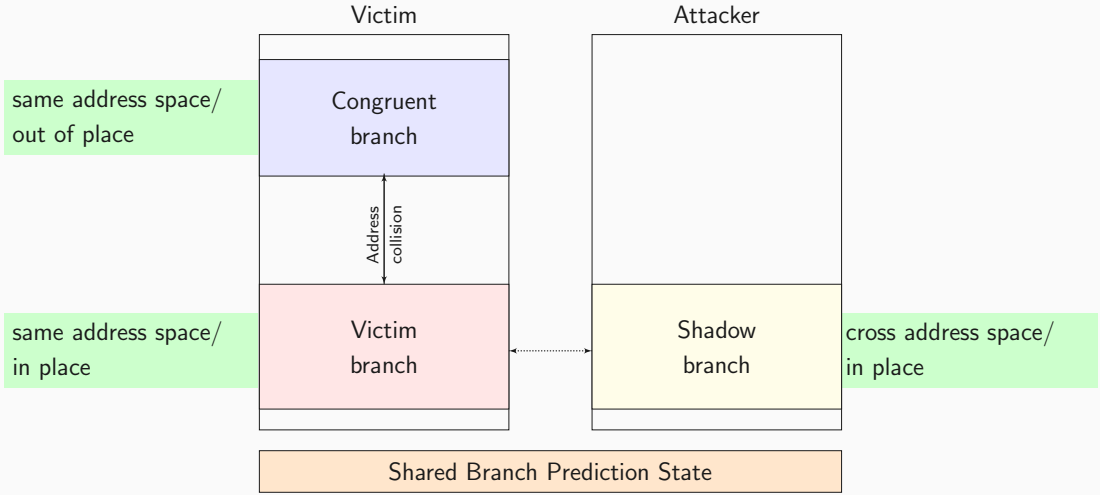
Spectre Mistraining



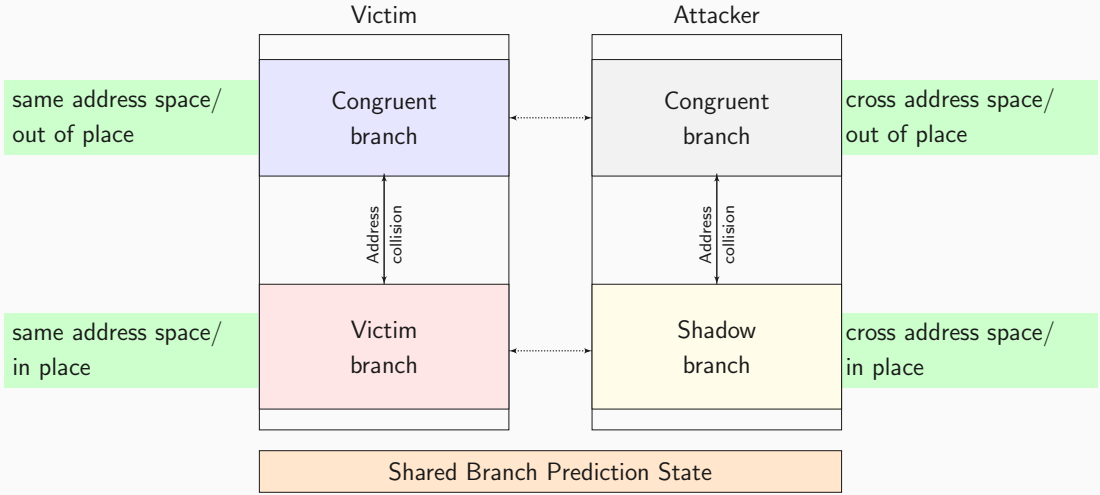
Spectre Mistraining



Spectre Mistraining



Spectre Mistraining



Time to code

Side-Channel Lab II

Michael Schwarz

Security Week Graz 2019


D. Gruss, C. Maurice, K. Wagner, and S. Mangard. Flush+Flush: A Fast and Stealthy Cache Attack. In: DIMVA. 2016.


M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard. ARMageddon: Cache Attacks on Mobile Devices. In: USENIX Security Symposium. 2016.


F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. Last-Level Cache Side-Channel Attacks are Practical. In: S&P. 2015.


C. Maurice, M. Weber, M. Schwarz, L. Giner, D. Gruss, C. Alberto Boano, S. Mangard, and K. Römer. Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud. In: NDSS. 2017.