

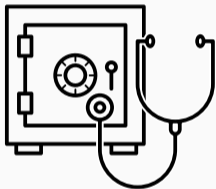
# Microarchitectural Attacks

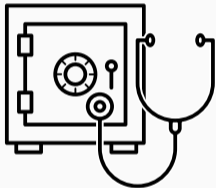
**Michael Schwarz**

November 22, 2019

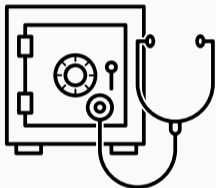
[www.iaik.tugraz.at](http://www.iaik.tugraz.at)

- Safe software infrastructure does not mean safe execution

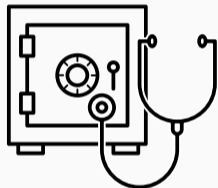




- Safe software infrastructure does not mean safe execution
- Information leaks because of the **underlying hardware**



- Safe software infrastructure does not mean safe execution
- Information leaks because of the **underlying hardware**
- Exploit **unintentional information leakage by side-effects**



- Safe software infrastructure does not mean safe execution
- Information leaks because of the **underlying hardware**
- Exploit **unintentional information leakage by side-effects**



Power consumption

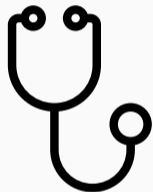


Execution time

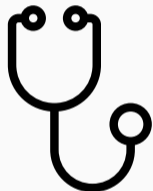


CPU caches

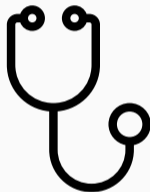




- Side channels also exist in **software**



- Side channels also exist in **software**
- Can be used for attacks



- Side channels also exist in **software**
- Can be used for attacks
- Usually **timing differences**



- Trivial approach: Compare each digit until a difference

```
int check_pin(char* input) {
    const char* correct = "1234";
    for(int i = 0; i < 4; i++) {
        if(correct[i] != input[i]) {
            // digit differs, abort
            return ERROR;
        }
    }
    // PIN is correct
    return OK;
}
```

- Measuring the **execution times** for different PINs

PIN Time




- Measuring the **execution times** for different PINs

PIN	Time
0000	





- Measuring the **execution times** for different PINs

PIN	Time
0000	
1000	





- Measuring the **execution times** for different PINs

PIN	Time
0000	
1000	
2000	





- Measuring the **execution times** for different PINs

PIN	Time
0000	
1000	
2000	
3000	

- Measuring the **execution times** for different PINs

PIN	Time
0000	
1000	
2000	
3000	
...	...





- Measuring the **execution times** for different PINs

PIN	Time
0000	
<b>1000</b>	
2000	
3000	
...	...

- If digit is **correct**, next digit is checked → **longer** execution time



- Measuring the **execution times** for different PINs

PIN	Time
0000	
1000	
2000	
3000	
...	...

- If digit is **correct**, next digit is checked → **longer** execution time
- 10 tries (maximum) to get a digit


- Measuring the execution times for different PINs

PIN Time

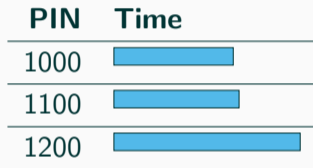
- Measuring the execution times for different PINs

PIN	Time
1000	

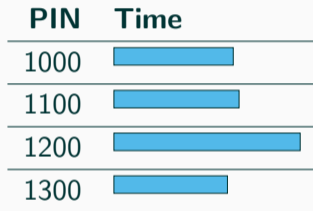
- Measuring the execution times for different PINs

PIN	Time
1000	
1100	

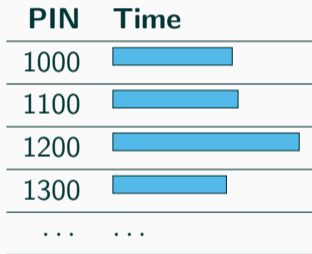
- Measuring the execution times for different PINs







- Measuring the execution times for different PINs



- Measuring the execution times for different PINs







- Measuring the execution times for different PINs

PIN	Time
1000	
1100	
1200	
1300	
...	...

- Repeat for every digit



- Measuring the execution times for different PINs

PIN	Time
1000	
1100	
1200	
1300	
...	...

- Repeat for every digit
- Longest** execution time reveals correct digit



- Maximum 10 measurements per digit



- Maximum 10 measurements per digit
- 4-digit PIN: 40 tries



- Maximum 10 measurements per digit
- 4-digit PIN: 40 tries
- Brute force: 10 000 tries



- Maximum 10 measurements per digit
- 4-digit PIN: 40 tries
- Brute force: 10 000 tries
- Simple side channel reduces tries by **factor 250**

- Many functions can be implemented with **constant runtime**

- Many functions can be implemented with **constant runtime**

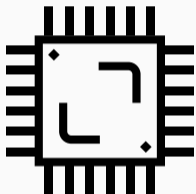
```
int check_pin(char* input) {
    const char* correct = "1234";
    int same = 0;
    for(int i = 0; i < 4; i++) {
        same |= correct[i] - input[i];
    }
    return (same == 0);
}
```

- Many functions can be implemented with **constant runtime**

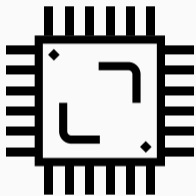
```
int check_pin(char* input) {
    const char* correct = "1234";
    int same = 0;
    for(int i = 0; i < 4; i++) {
        same |= correct[i] - input[i];
    }
    return (same == 0);
}
```

- Sometimes, there is still a side channel in the **hardware**

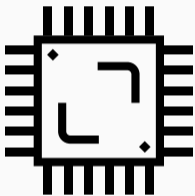




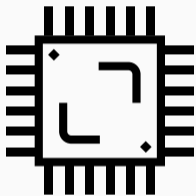
- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, ...)



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



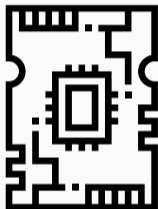
- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, ...)
- Serves as the **interface** between hardware and software
- Microarchitecture is an **actual implementation** of the ISA



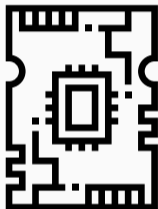
- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, ...)
- Serves as the **interface** between hardware and software
- Microarchitecture is an **actual implementation** of the ISA



- Modern CPUs contain multiple **microarchitectural elements**



- Modern CPUs contain multiple **microarchitectural elements**



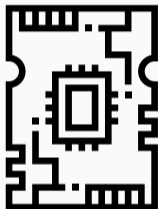
Caches and buffer



Predictor



- Modern CPUs contain multiple **microarchitectural elements**



Caches and buffer

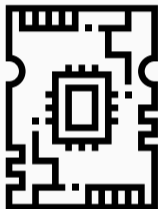


Predictor



- **Transparent** for the programmer

- Modern CPUs contain multiple **microarchitectural elements**



Caches and buffer



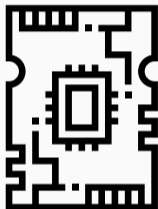
Predictor



- **Transparent** for the programmer
- **Optimize** program execution



- Modern CPUs contain multiple **microarchitectural elements**



Caches and buffer

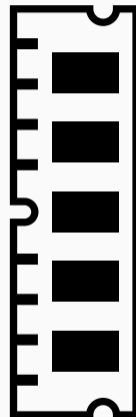
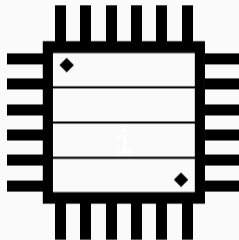


Predictor



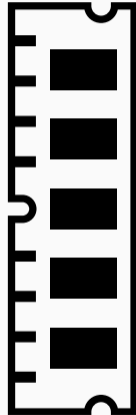
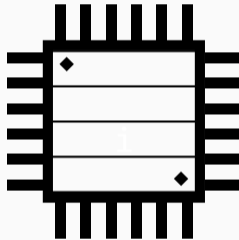
- **Transparent** for the programmer
- **Optimize** program execution
- Timing differences → side-channel leakage

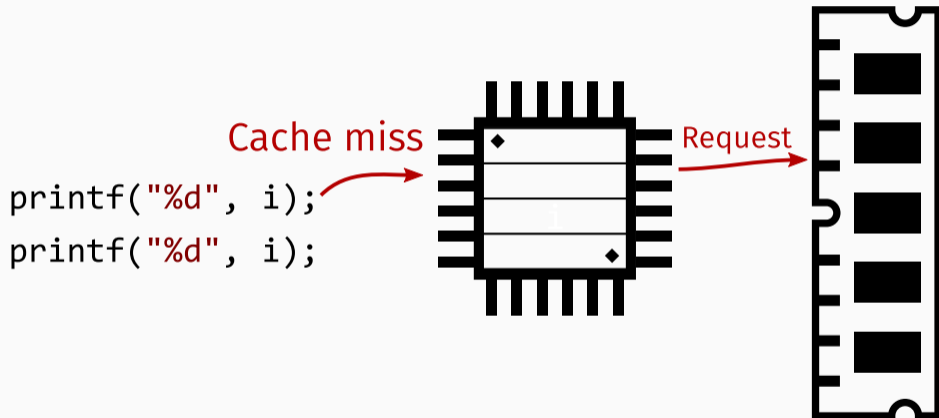
```
printf("%d", i);  
printf("%d", i);
```

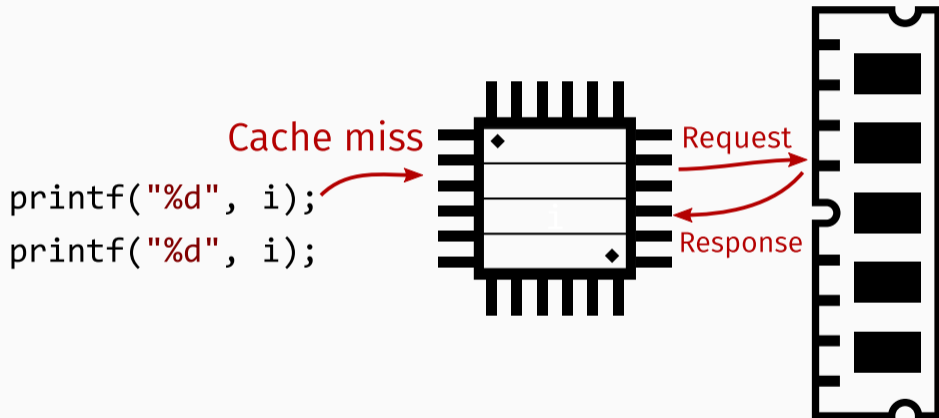


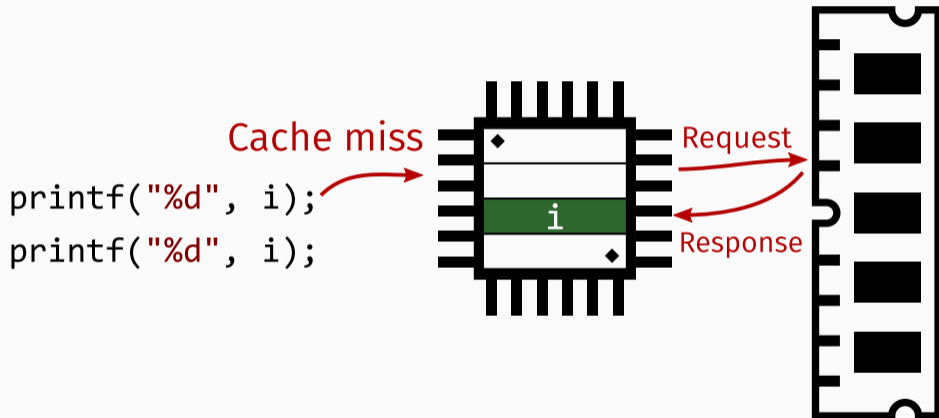
```
printf("%d", i);  
printf("%d", i);
```

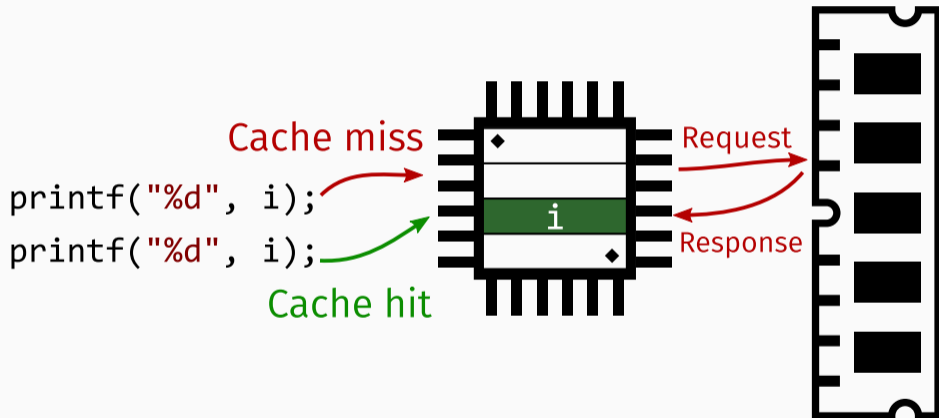
Cache miss

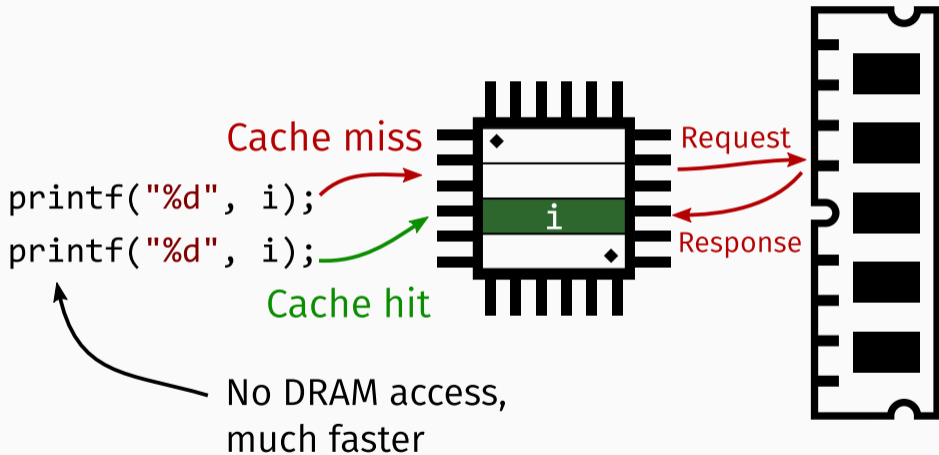




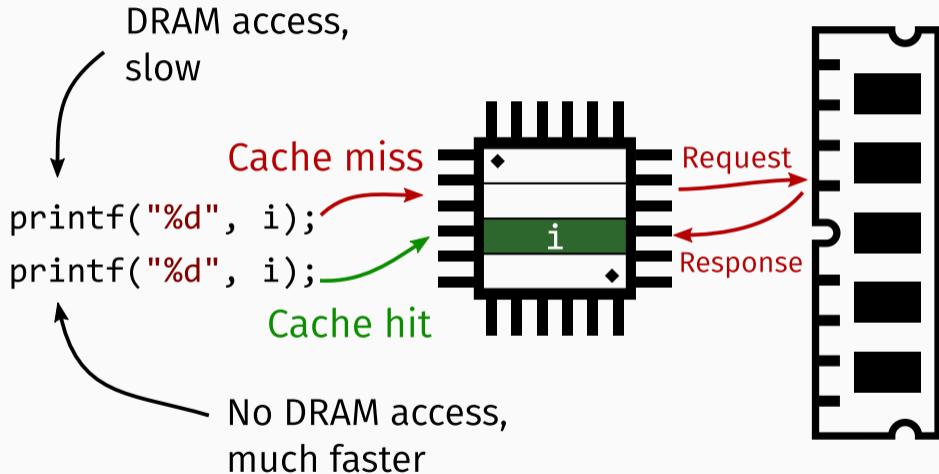


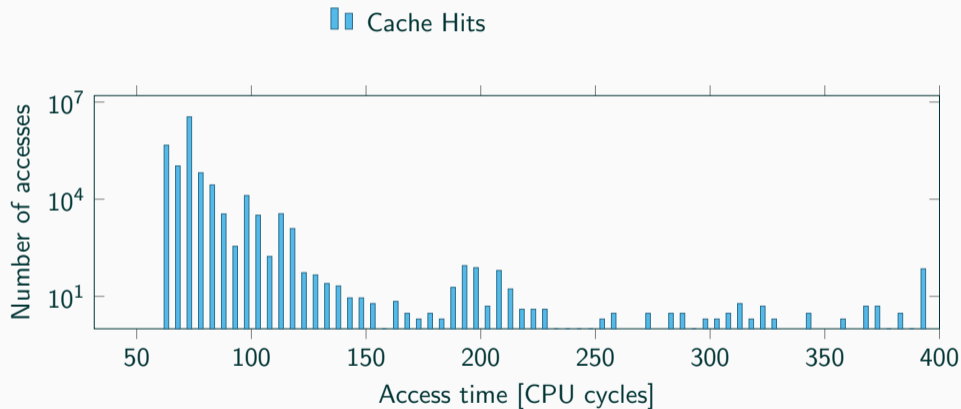


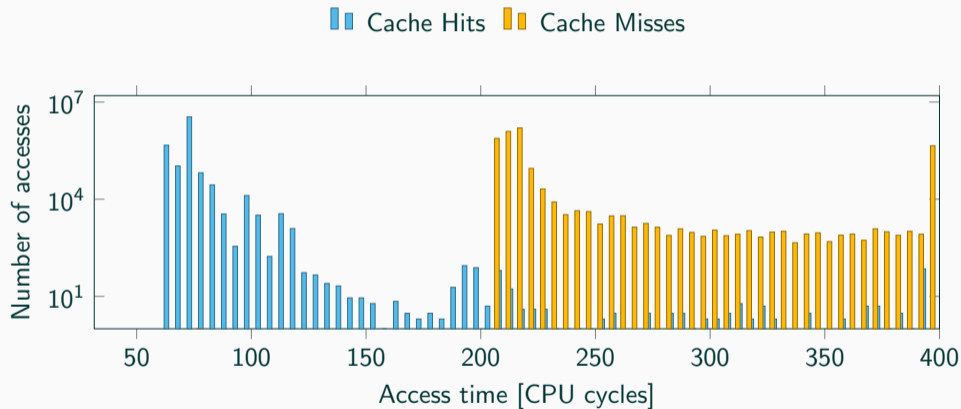


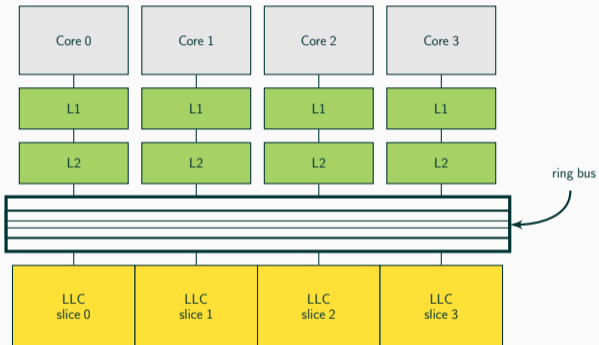




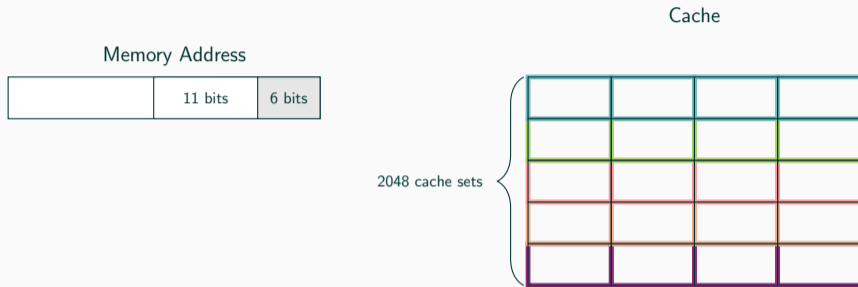




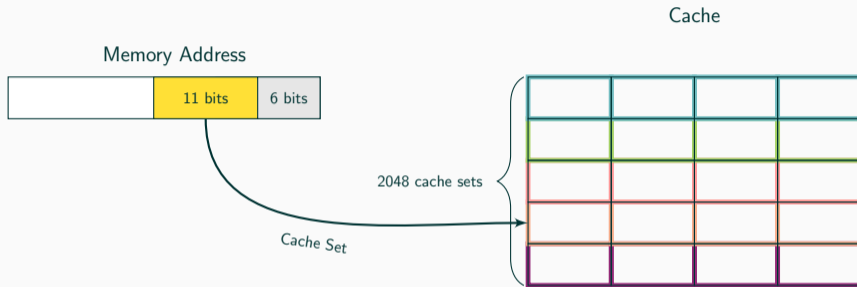




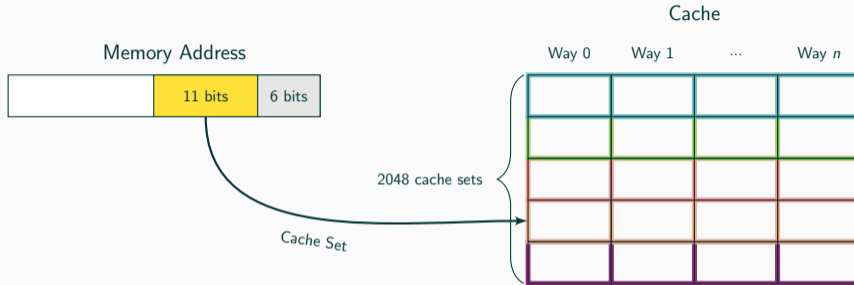
- L1 and L2 are private
- Last-level cache is
  - divided into slices
  - shared across cores
  - inclusive



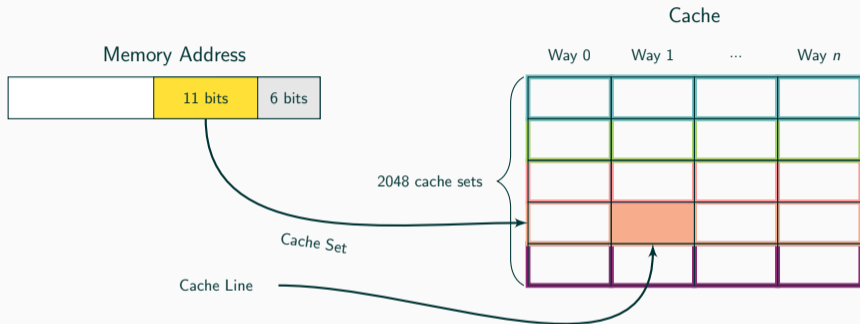
- Location in cache depends on the physical address of data



- Location in cache depends on the physical address of data
- Bits 6 to 16 determine the **cache set**



- Location in cache depends on the physical address of data
- Bits 6 to 16 determine the **cache set**
- A cache set has multiple **ways** to store the data

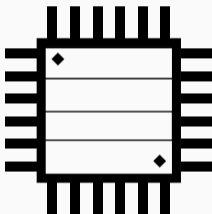


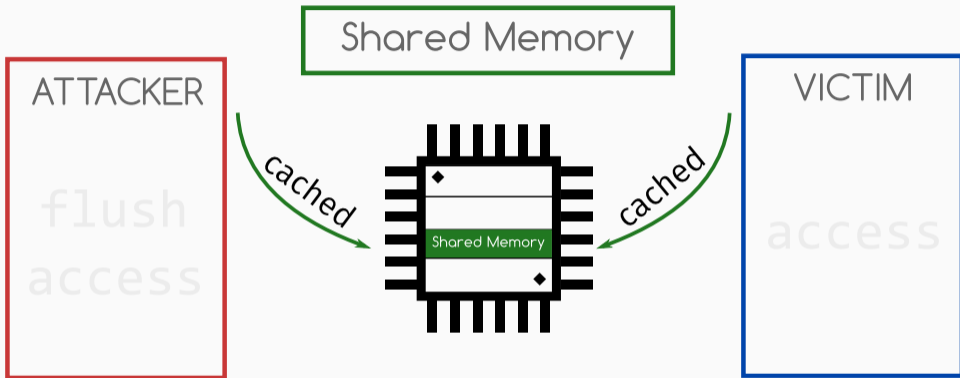
- Location in cache depends on the physical address of data
- Bits 6 to 16 determine the **cache set**
- A cache set has multiple **ways** to store the data
- A way inside a cache set is a **cache line**, determined by the **cache replacement policy**

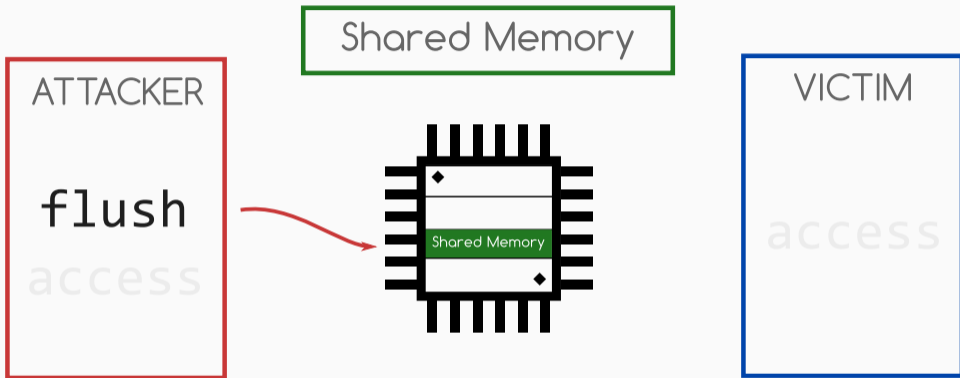


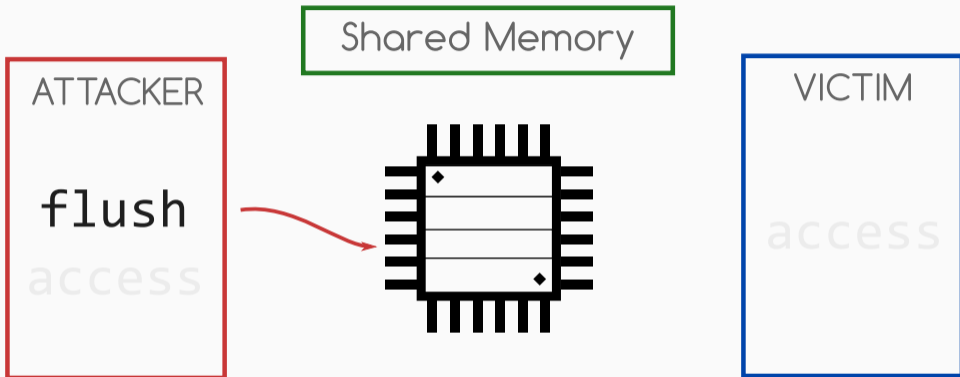


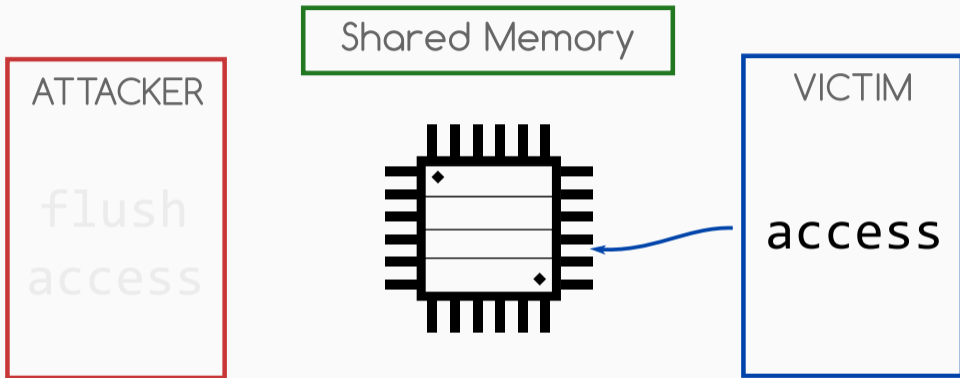
Shared Memory

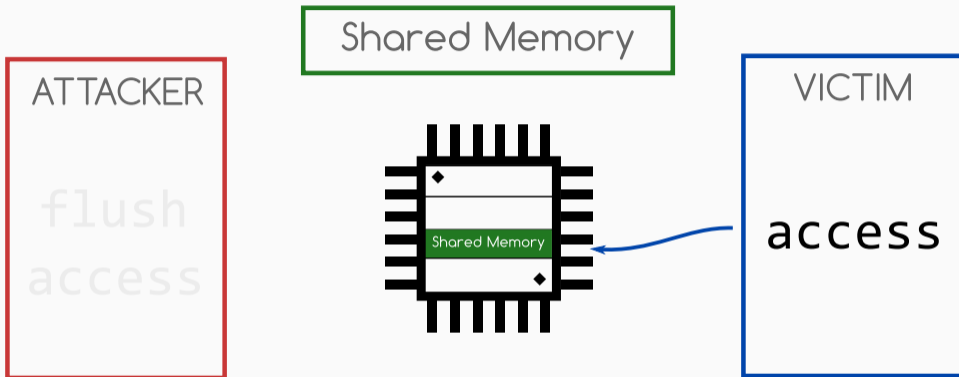


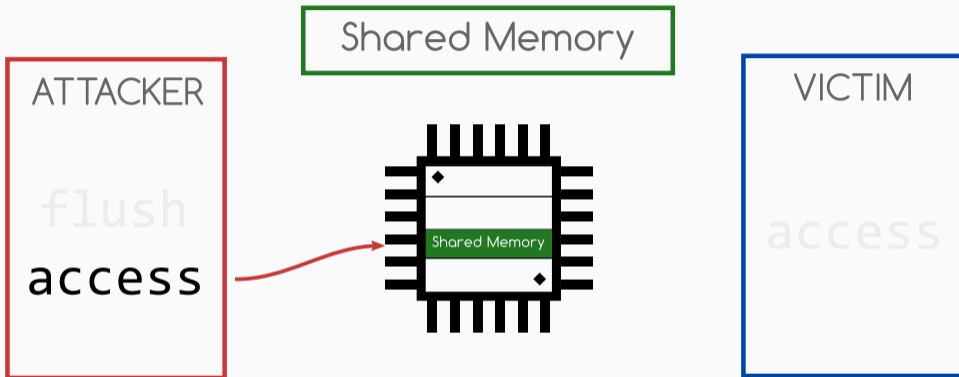


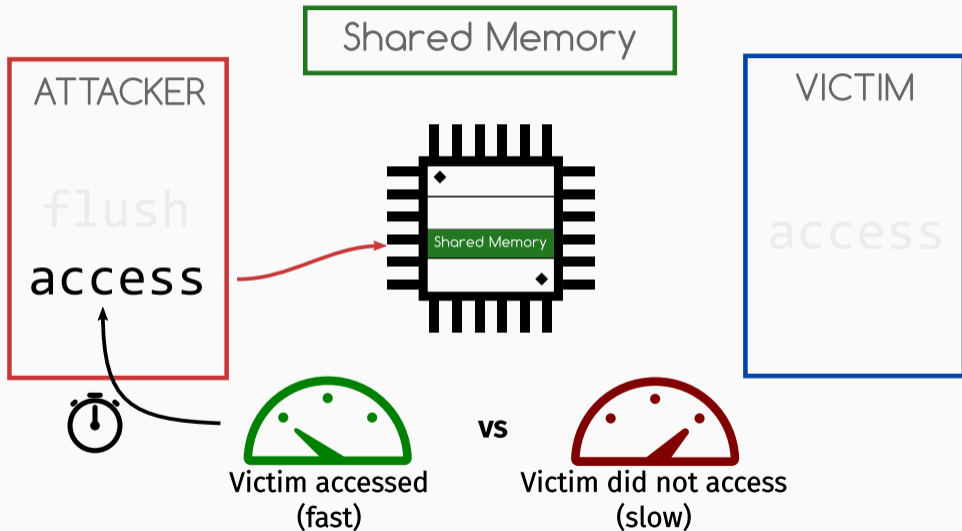




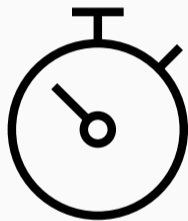












```
struct shared_data [256];  
[...]  
return shared_data [84];  
[...]
```



```
struct shared_data [256];  
[...]  
return shared_data [84];  
[...]
```

- Flush+Reload over memory locations



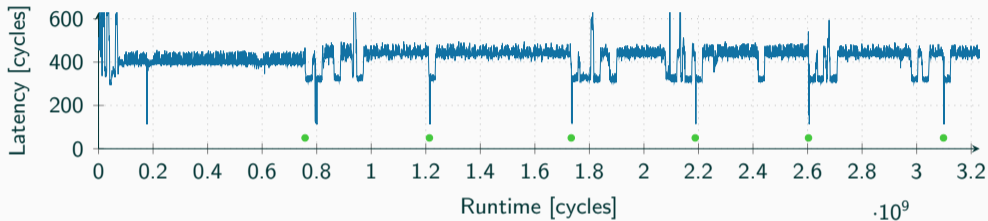


```
struct shared_data [256];  
[...]  
return shared_data [84];  
[...]
```

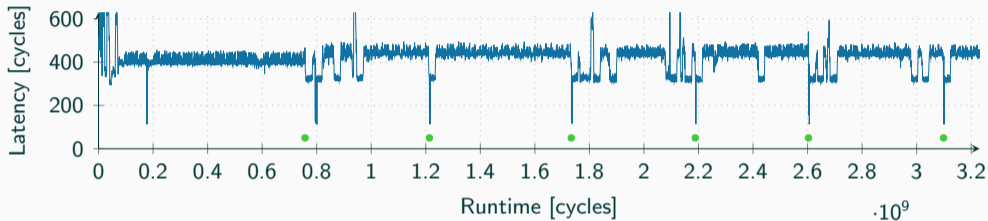
- Flush+Reload over memory locations



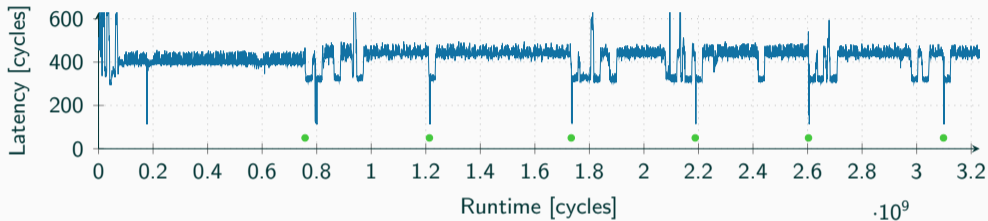
- Accessed index results in **faster access time**



- Key presses trigger code execution in shared library (e.g., `libgdk`)



- Key presses trigger code execution in shared library (e.g., `libgdk`)
- Flush+Reload does not reveal actual key, only **time difference** between keys



- Key presses trigger code execution in shared library (e.g., `libgdk`)
- Flush+Reload does not reveal actual key, only **time difference** between keys
- → Recover text with machine learning

string



string

/	p	a	t	h	/	f	i	l	e	\0	p	a	y	l	o	a	d	\0
---	---	---	---	---	---	---	---	---	---	----	---	---	---	---	---	---	---	----

←—————→  
length

Thread 1

```
strcpy(string, "/path/file\0payload");  
open(string, O_CREAT);
```

Thread 2



string

/	p	a	t	h	/	f	i	l	e	\0	p	a	y	l	o	a	d	\0
---	---	---	---	---	---	---	---	---	---	----	---	---	---	---	---	---	---	----

←————→  
length

Thread 1

```
strcpy(string, "/path/file\0payload");  
open(string, O_CREAT);  
  
// <switch to kernel>
```

Thread 2

string

/	p	a	t	h	/	f	i	l	e	\0	p	a	y	l	o	a	d	\0
---	---	---	---	---	---	---	---	---	---	----	---	---	---	---	---	---	---	----

←—————→  
length

Thread 1

```
strcpy(string, "/path/file\0payload");  
open(string, O_CREAT);  
  
// <switch to kernel>  
  
int len = strlen(string);  
char* local = malloc(len + 1);
```

Thread 2

string

/	p	a	t	h	/	f	i	l	e	X	p	a	y	l	o	a	d	\0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----

←—————→

length

Thread 1

Thread 2

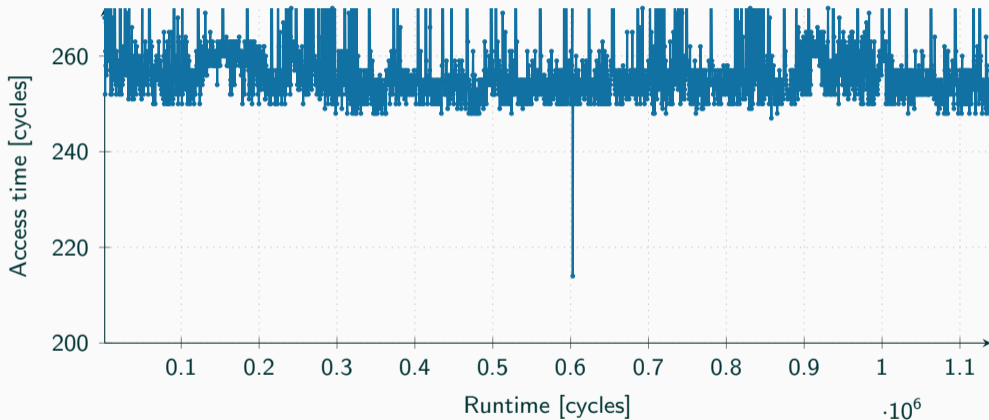
```
strcpy(string, "/path/file\0payload");  
open(string, O_CREAT);
```

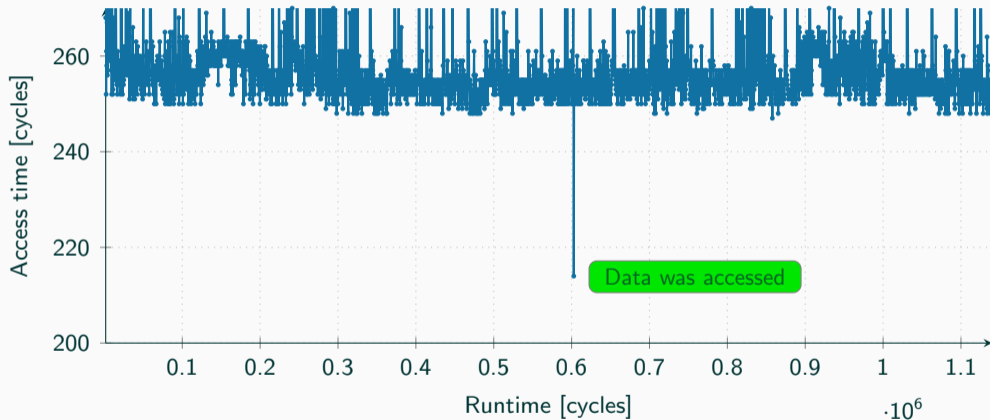
```
// <switch to kernel>
```

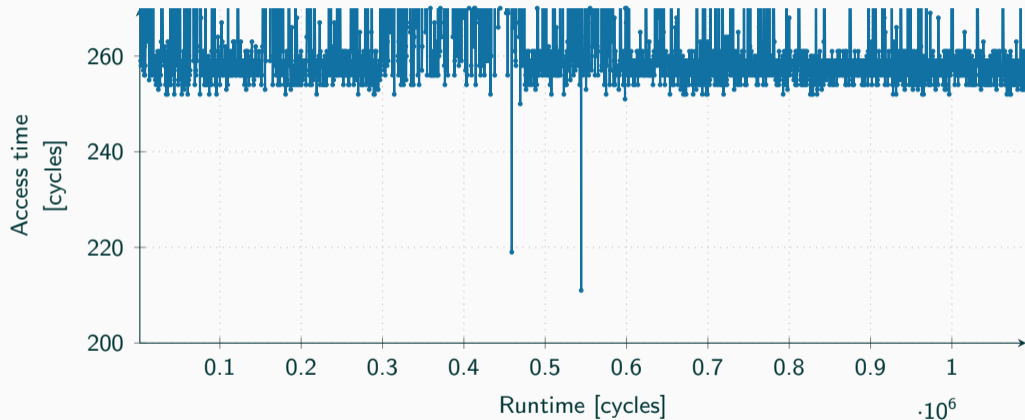
```
int len = strlen(string);  
char* local = malloc(len + 1);
```

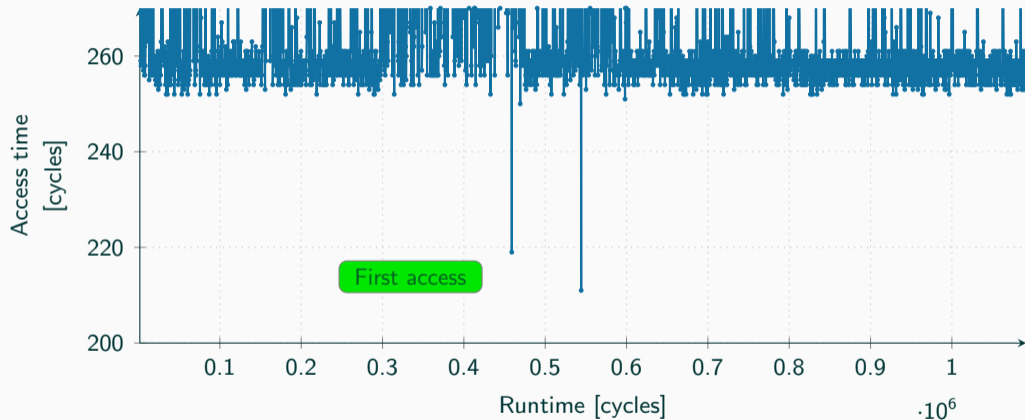
schedule → string[10] = 'X';



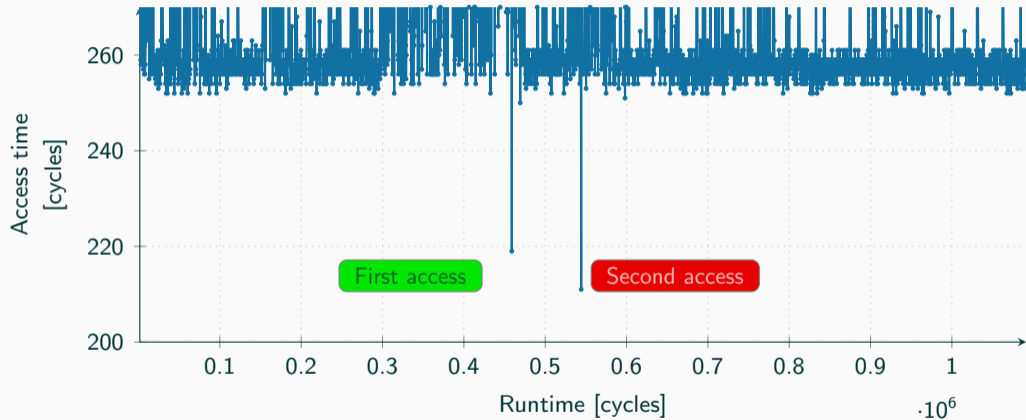


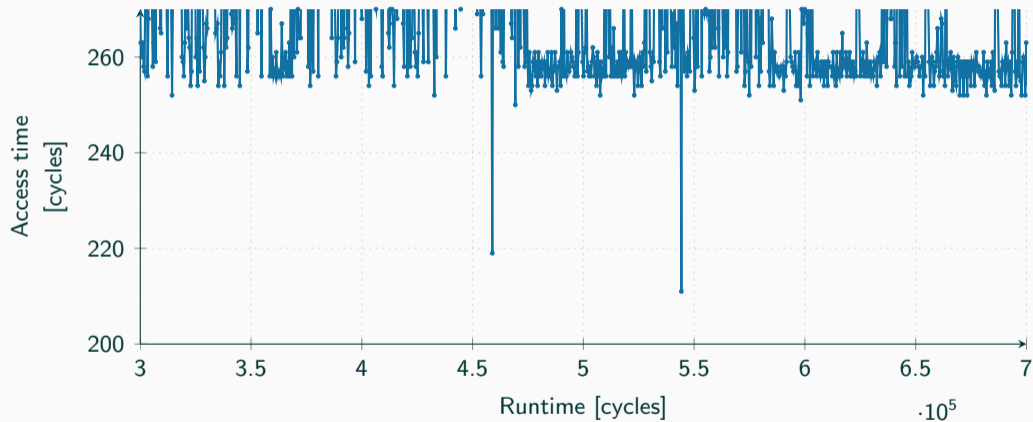


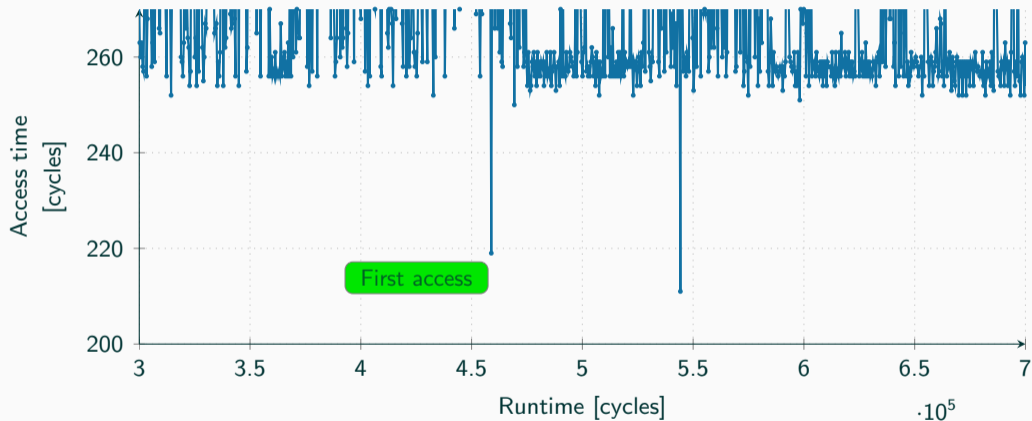


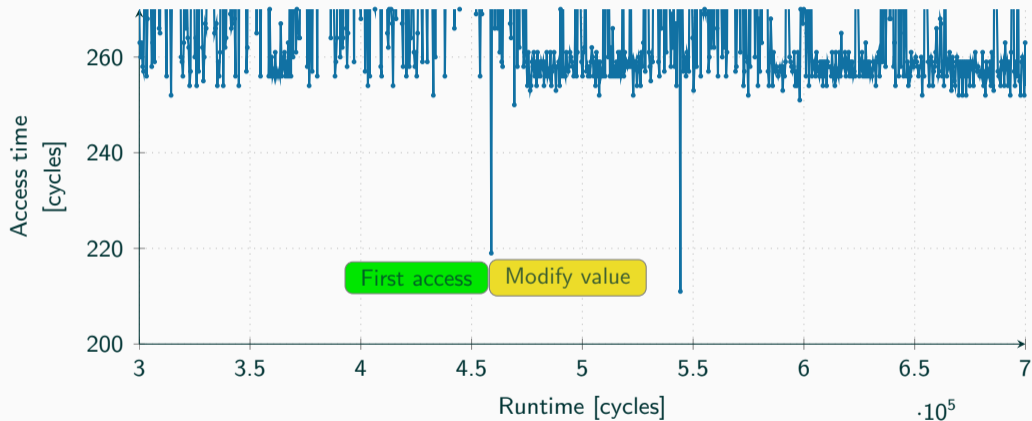


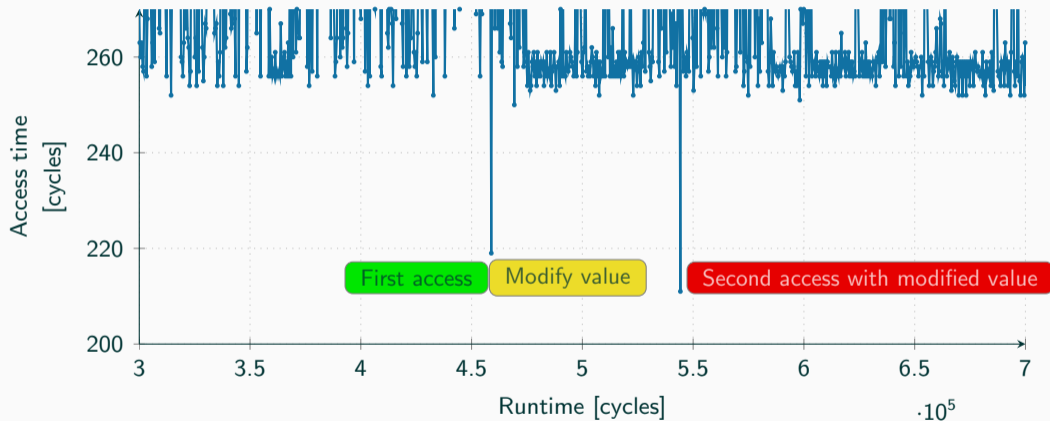


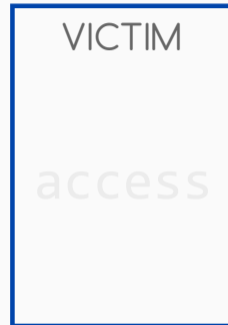
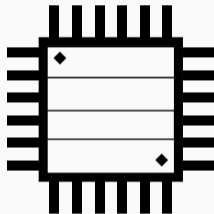


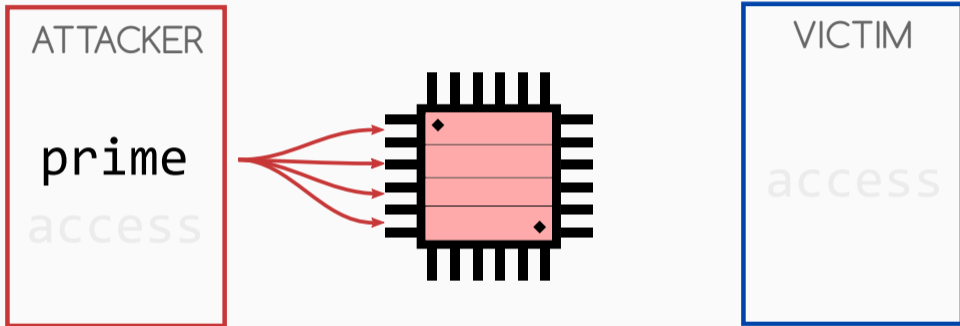


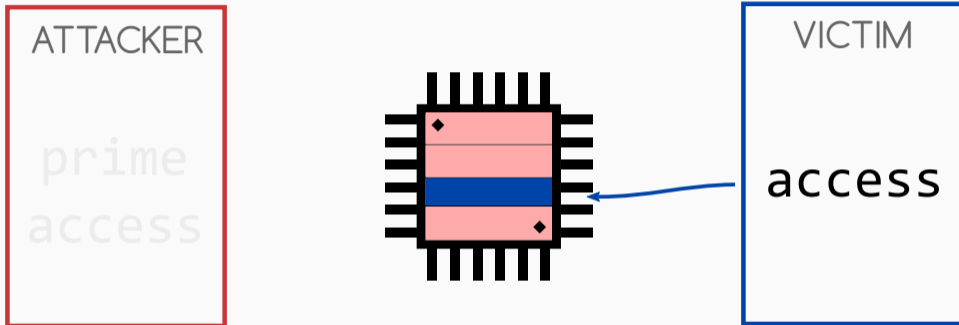




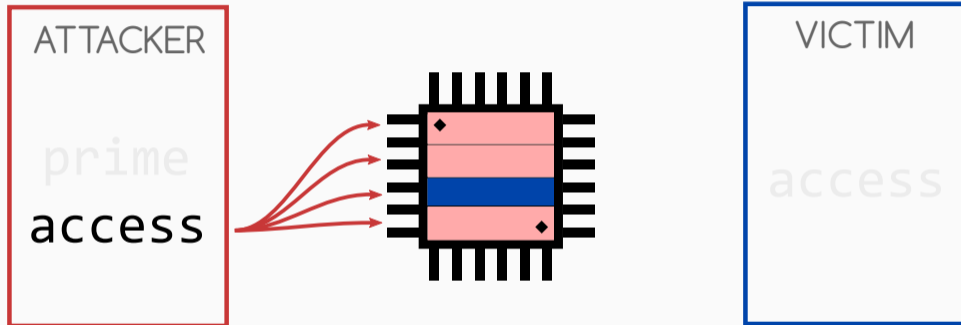


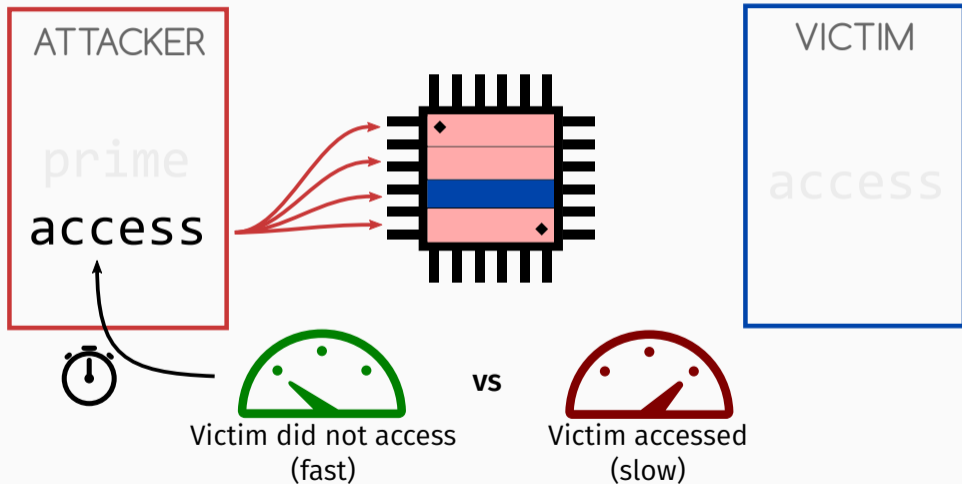




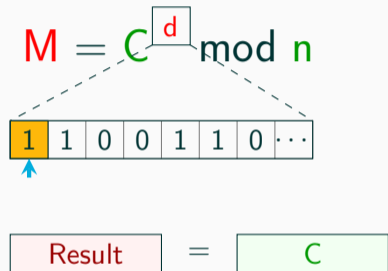








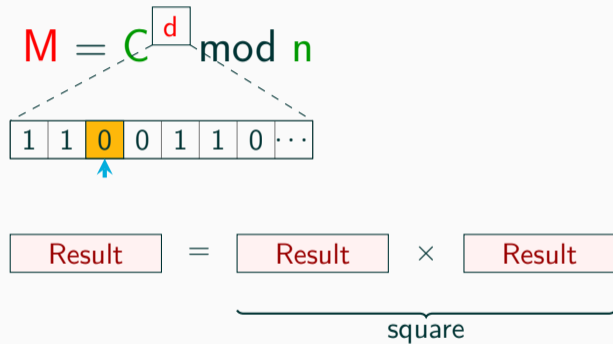
$$M = C^d \bmod n$$

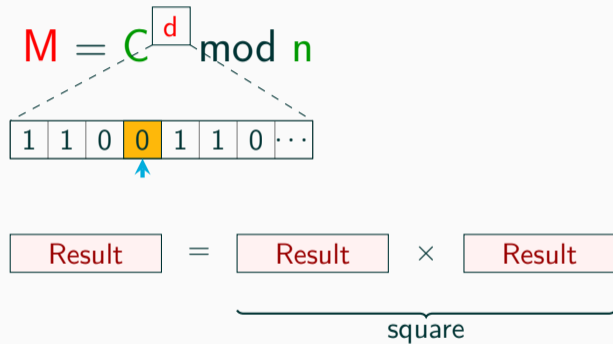


$$M = C^d \pmod n$$

1 1 0 0 1 1 0 ...

$$\text{Result} = \underbrace{\text{Result} \times \text{Result}}_{\text{square}} \times \underbrace{C}_{\text{multiply}}$$





$$M = C^d \pmod n$$

1 1 0 0 1 1 0 ...

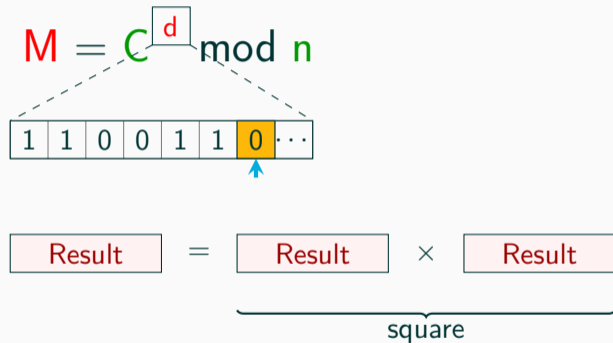
$$\text{Result} = \underbrace{\text{Result} \times \text{Result}}_{\text{square}} \times \underbrace{C}_{\text{multiply}}$$



$$M = C^d \pmod n$$

1 1 0 0 1 1 0 ...

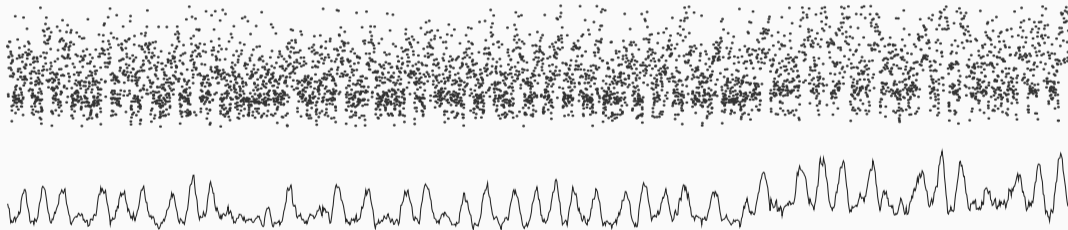
$$\text{Result} = \underbrace{\text{Result} \times \text{Result}}_{\text{square}} \times \underbrace{C}_{\text{multiply}}$$



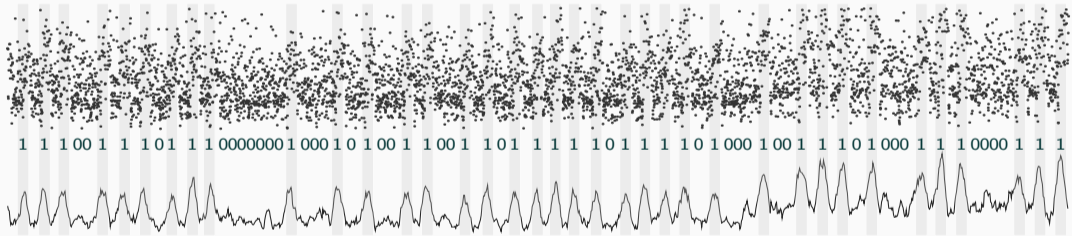
Raw Prime+Probe trace...



...processed with a simple moving average...



...allows to clearly see the bits of the exponent



Intel claiming it is  
out of scope



Side-channel Researcher





What is a **covert channel**?

- Two programs would like to communicate



What is a **covert channel**?

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





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...



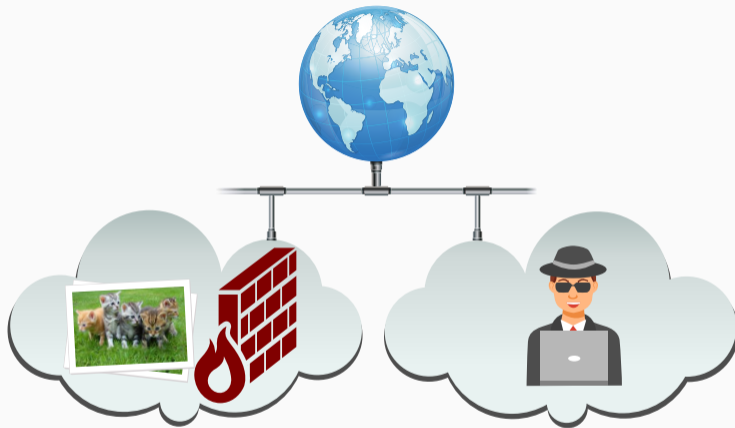
What is a **covert channel**?

- 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



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

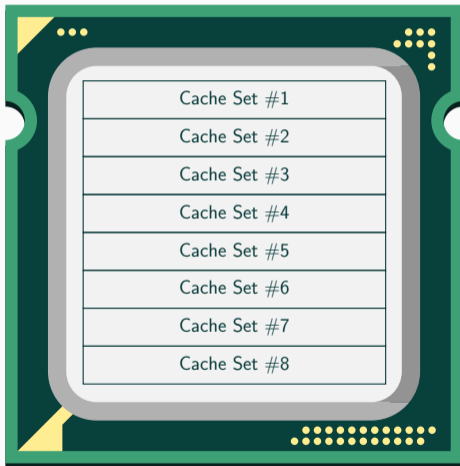




Sender

Last-level cache

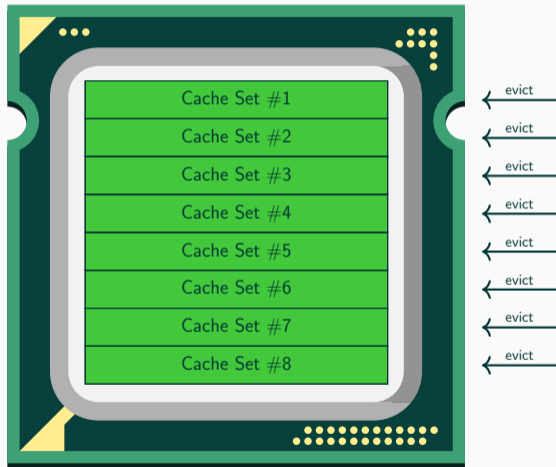
Receiver



Sender

Last-level cache

Receiver

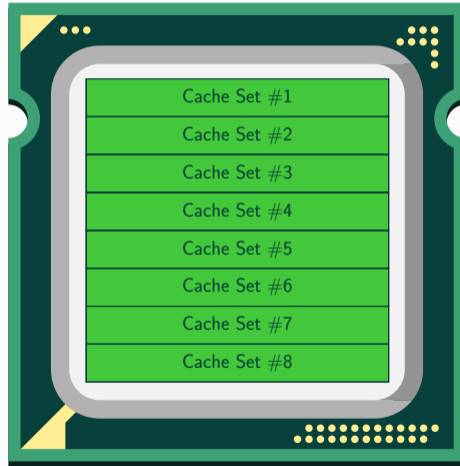


Sender

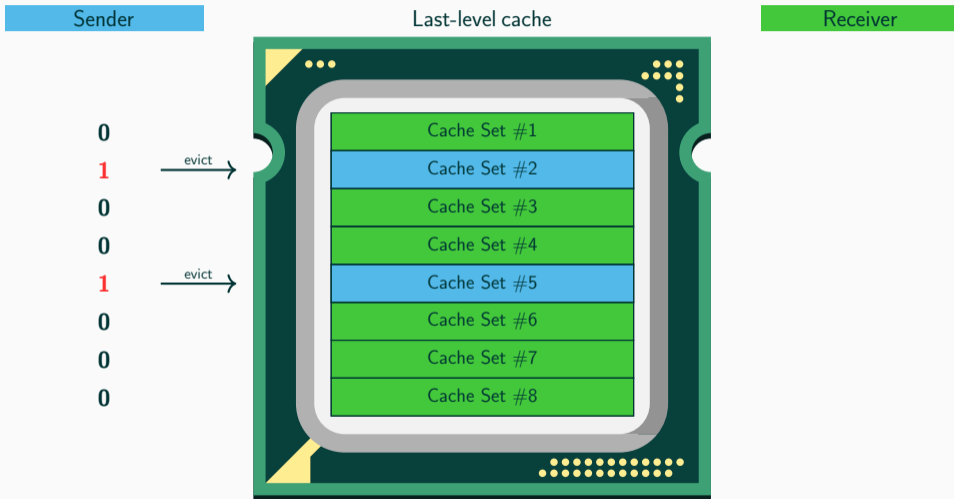
Last-level cache

Receiver

0  
1  
0  
0  
1  
0  
0  
0







Sender

Last-level cache

Receiver

0

1

0

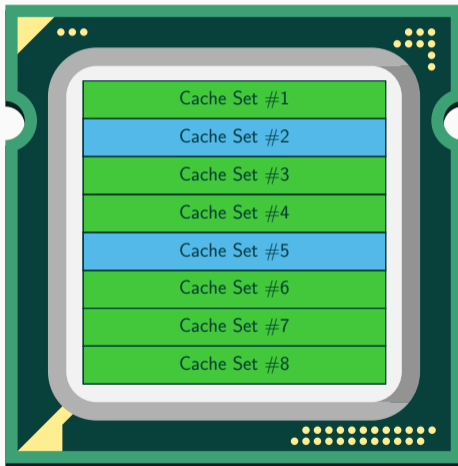
0

1

0

0

0

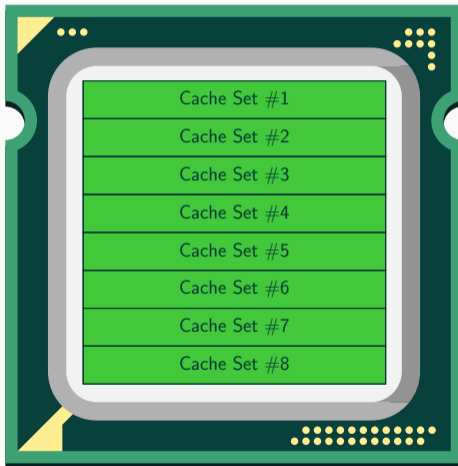


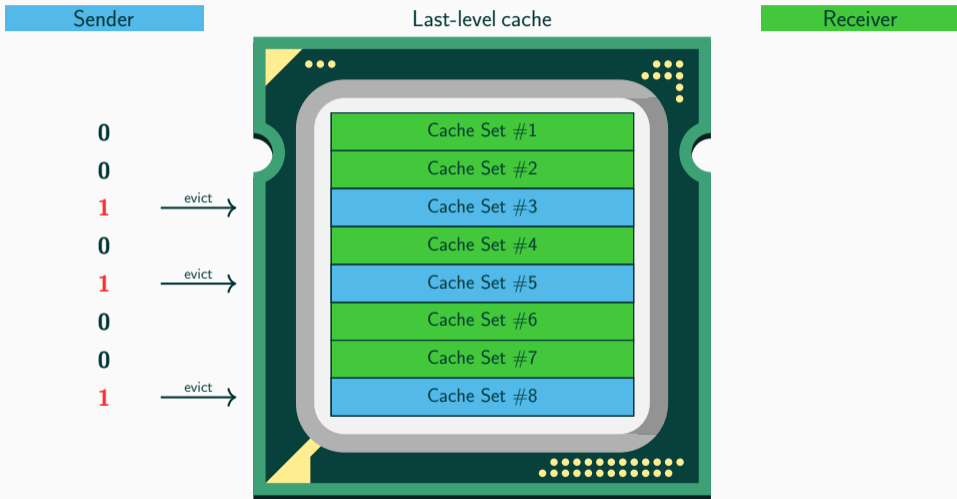
Sender

Last-level cache

Receiver

0  
0  
1  
0  
1  
0  
0  
1



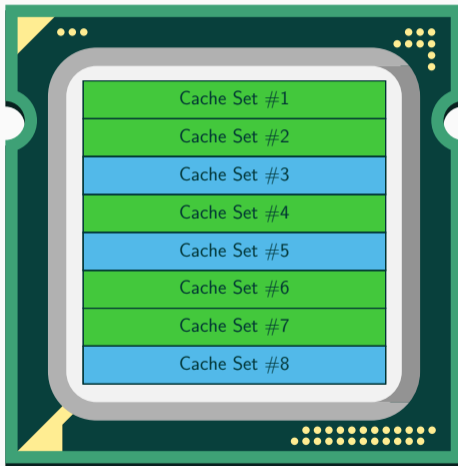


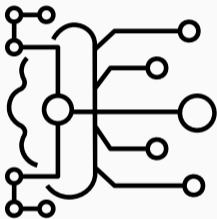
Sender

Last-level cache

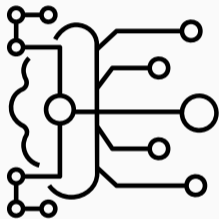
Receiver

0  
0  
1  
0  
1  
0  
0  
1

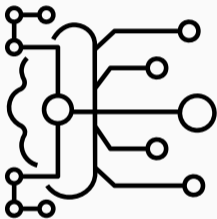




- Multiple other elements with timing differences

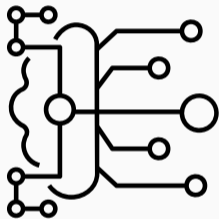


- Multiple other elements with timing differences
  - TLB

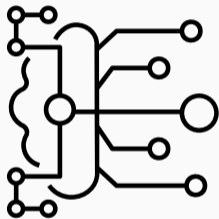


- Multiple other elements with timing differences
  - TLB
  - DRAM

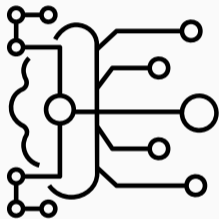




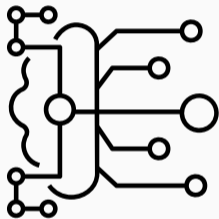
- Multiple other elements with timing differences
  - TLB
  - DRAM
  - Memory Bus



- Multiple other elements with timing differences
  - TLB
  - DRAM
  - Memory Bus
  - Execution Units



- Multiple other elements with timing differences
  - TLB
  - DRAM
  - Memory Bus
  - Execution Units
  - ...



- Multiple other elements with timing differences
  - TLB
  - DRAM
  - Memory Bus
  - Execution Units
  - ...
- Many side-channel attacks exploiting them



- So far, only memory accesses



- So far, only memory accesses
- **Meta data**, no actual data

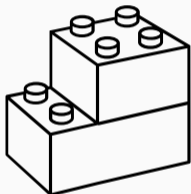


- So far, only memory accesses
- **Meta data**, no actual data
- Sufficient to **deduce** data...

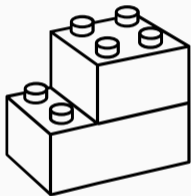


- So far, only memory accesses
- **Meta data**, no actual data
- Sufficient to **deduce** data...
- ...if memory accesses are **secret dependent**

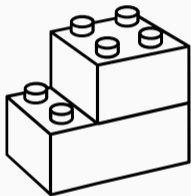




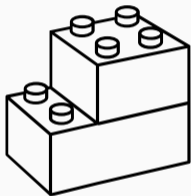
- Side channels can be part of an attack



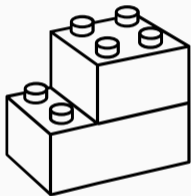
- Side channels can be part of an attack
- Also for **conventional** memory corruption **attacks**



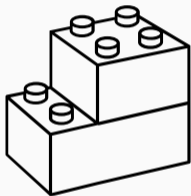
- Side channels can be part of an attack
- Also for **conventional** memory corruption **attacks**
- Side channels as **building blocks**



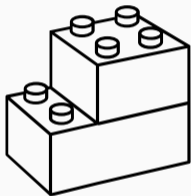
- Side channels can be part of an attack
- Also for **conventional** memory corruption **attacks**
- Side channels as **building blocks**
  - Required information (e.g., break ASLR)



- Side channels can be part of an attack
- Also for **conventional** memory corruption **attacks**
- Side channels as **building blocks**
  - Required information (e.g., break ASLR)
  - Additional information (e.g., length of password)



- Side channels can be part of an attack
- Also for **conventional** memory corruption **attacks**
- Side channels as **building blocks**
  - Required information (e.g., break ASLR)
  - Additional information (e.g., length of password)
  - Covertly transmit information



- Side channels can be part of an attack
- Also for **conventional** memory corruption **attacks**
- Side channels as **building blocks**
  - Required information (e.g., break ASLR)
  - Additional information (e.g., length of password)
  - Covertly transmit information
  - **Transient-execution attacks**



- Meltdown is a CPU vulnerabilities





- Meltdown is a CPU vulnerabilities
- Discovered in 2017 by multiple independent teams

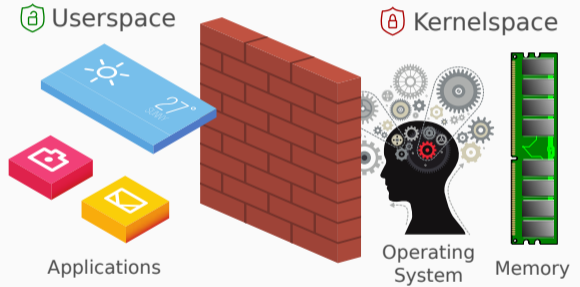


- Meltdown is a CPU vulnerabilities
- Discovered in 2017 by multiple independent teams
- Allows breaking the process isolation

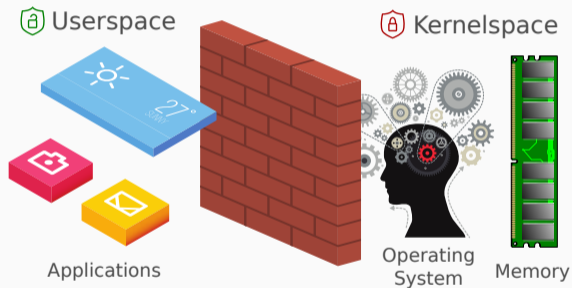


- Meltdown is a CPU vulnerabilities
- Discovered in 2017 by multiple independent teams
- Allows breaking the process isolation
- Side-channel attack is a core building block

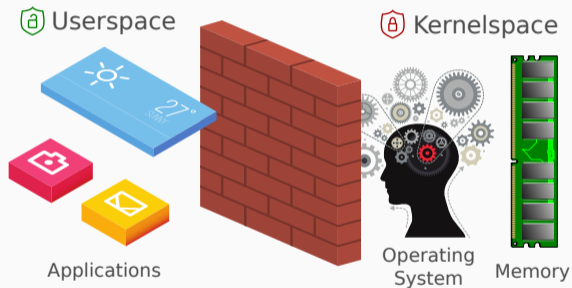
- Kernel is isolated from user space



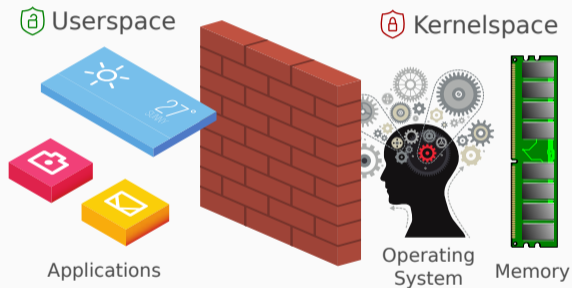
- Kernel is isolated from user space
- This **isolation** is a combination of hardware and software

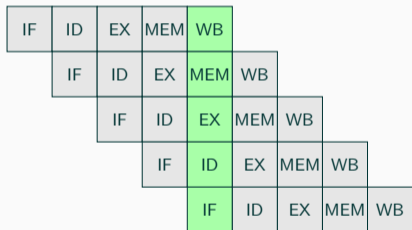


- Kernel is isolated from user space
- This **isolation** is a combination of hardware and software
- User applications cannot access anything from the kernel



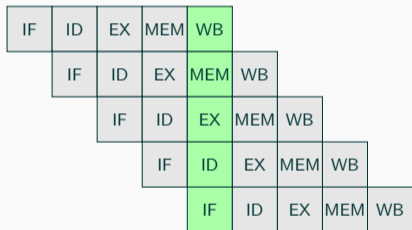
- Kernel is isolated from user space
- This **isolation** is a combination of hardware and software
- User applications cannot access anything from the kernel
- There is only a well-defined interface → **syscalls**





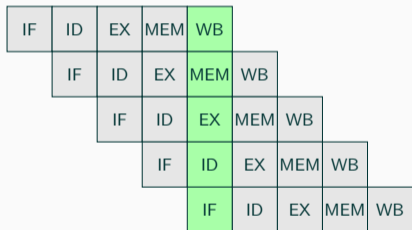
- Instructions are...
  - fetched (IF) from the L1 Instruction Cache





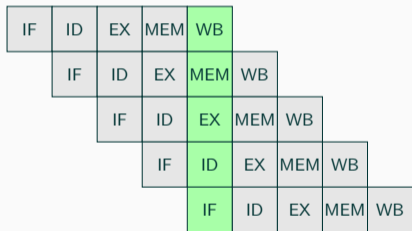
- Instructions are...

- fetched (IF) from the L1 Instruction Cache
- decoded (ID)

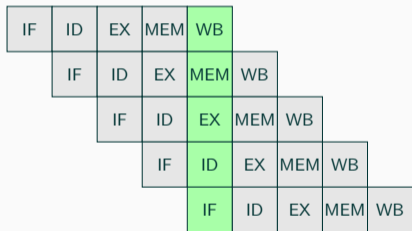


- Instructions are...

- **fetched**  (IF) from the L1 Instruction Cache
- **decoded**  (ID)
- **executed**  (EX) by execution units



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



- 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)



- Instructions are executed **in-order**



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



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







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

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

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

## Parallelize

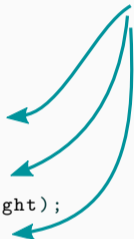
Dependency

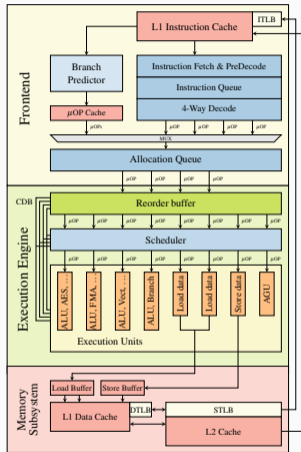
```
int width = 10, height = 5;

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

int area = width * height;

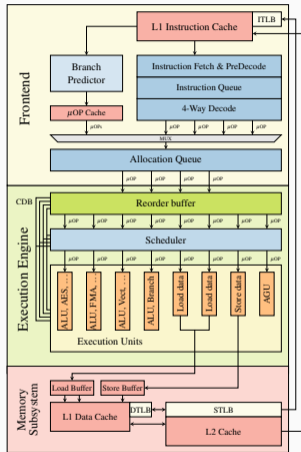
printf("Area %d x %d = %d\n", width, height, area);
```





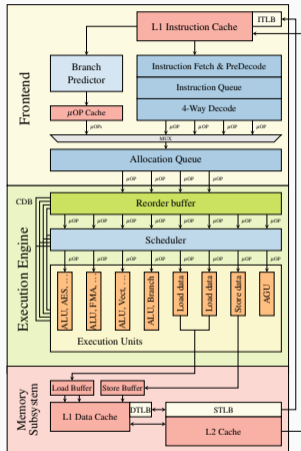
Instructions are

- fetched and decoded in the **front-end**



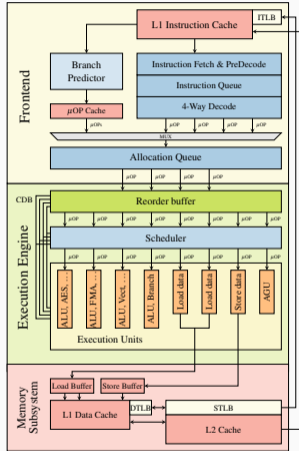
Instructions are

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



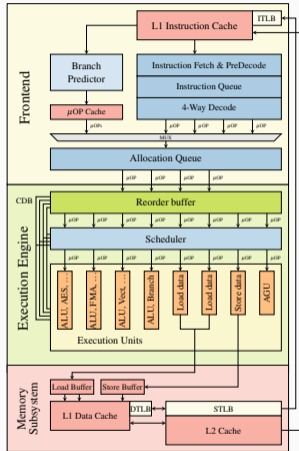
Instructions are

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



## Instructions

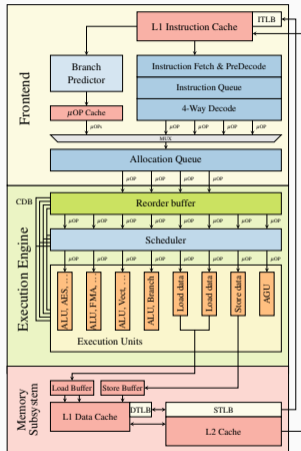
- are executed **out-of-order**



## Instructions

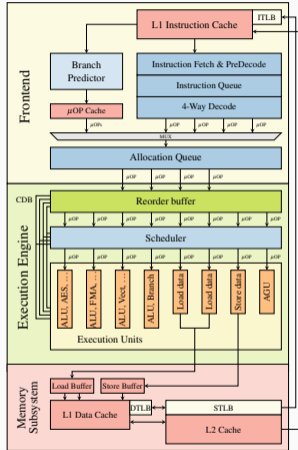
- are executed **out-of-order**
- wait until their **dependencies are ready**





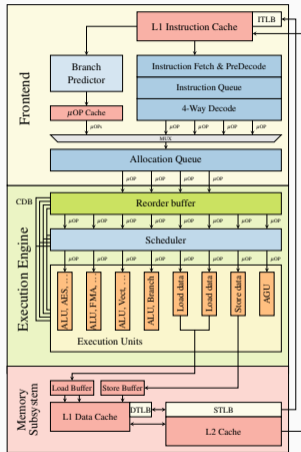
## Instructions

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



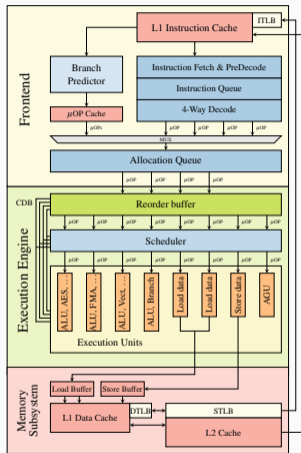
## Instructions

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



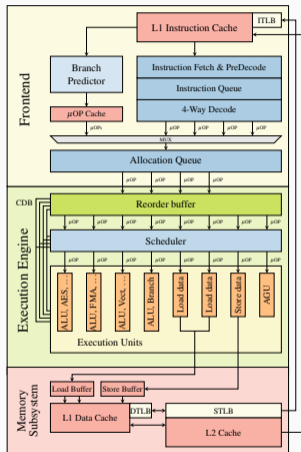
## Instructions

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



## 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



## 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 ...



- New code

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







- New code

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

- volatile because compiler was not happy

```
warning: statement with no effect [-Wunused-value]  
    *(char*)0;
```



- New code

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

- volatile because compiler was not happy

```
warning: statement with no effect [-Wunused-value]  
    *(char*)0;
```

- Static code analyzer is still not happy

```
warning: Dereference of null pointer  
    *(volatile char*)0;
```



- Flush+Reload over all pages of the array





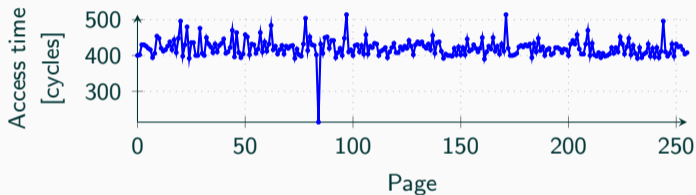
- Flush+Reload over all pages of the array



- “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**



- Out-of-order instructions **leave microarchitectural traces**



- Out-of-order instructions **leave microarchitectural traces**
  - We can see them for example in the cache



- Out-of-order instructions **leave microarchitectural traces**
  - We can see them for example in the cache
- Give such instructions a name: **transient instructions**





- Out-of-order instructions **leave microarchitectural traces**
  - We can see them for example in the cache
- Give such instructions a name: **transient instructions**
- We can indirectly observe the **execution of transient instructions**















- Add another **layer of indirection** to test

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





- Add another **layer of indirection** to test

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

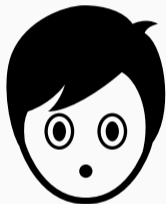
- Then check whether any part of array is **cached**



- Flush+Reload over all pages of the array



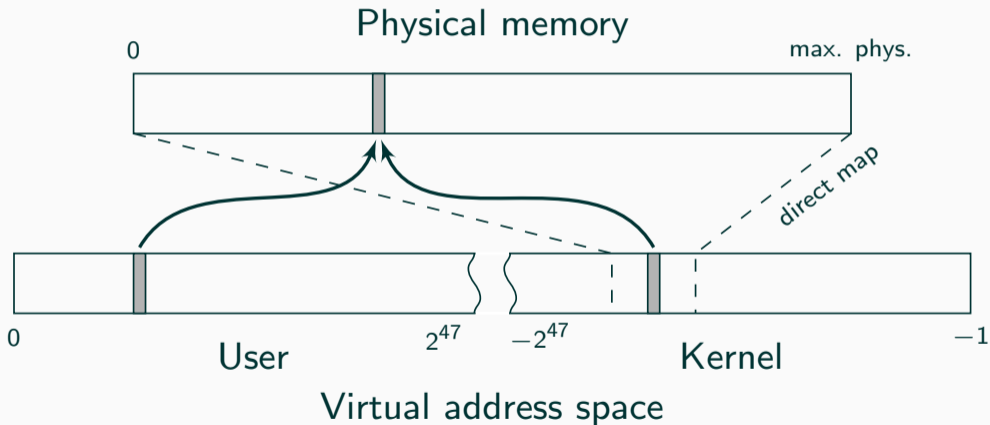
- Index of cache hit reveals data



- Flush+Reload over all pages of the array



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





MELTDOWN

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



MELTDOWN

- Using **out-of-order execution**, we can read **data at any address**
- **Index** of cache hit reveals **data**



MELTDOWN

- Using **out-of-order execution**, we can read **data at any address**
- **Index** of cache hit reveals **data**
- **Permission check** is in some cases **not fast enough**



- Using **out-of-order execution**, we can read **data at any address**
- **Index** of cache hit reveals **data**
- **Permission check** is in some cases **not fast enough**
- **Entire physical memory** is typically accessible through kernel space



**I SHIT YOU NOT**

**THERE WAS KERNEL MEMORY ALL  
OVER THE TERMINAL**





**There are no bugs,  
just happy little accidents**





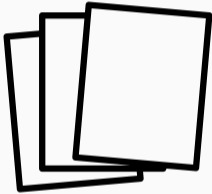
- Meltdown is a whole **category of vulnerabilities**



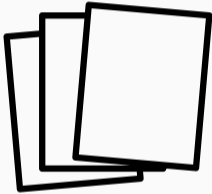
- Meltdown is a whole **category of vulnerabilities**
- Not only the user-accessible check



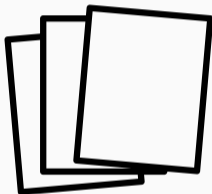
- Meltdown is a whole **category of vulnerabilities**
- Not only the user-accessible check
- Looking closer at the check...



- CPU uses **virtual address spaces** to isolate processes

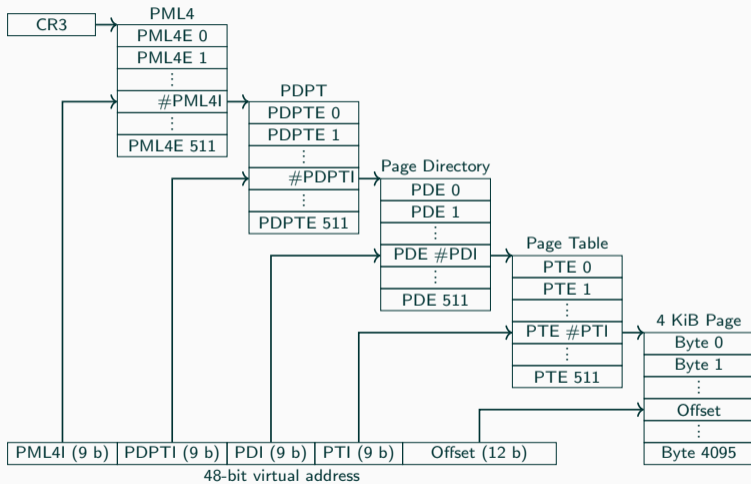


- CPU uses **virtual address spaces** to isolate processes
- Physical memory is organized in **page frames**



- 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**





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

- 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

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





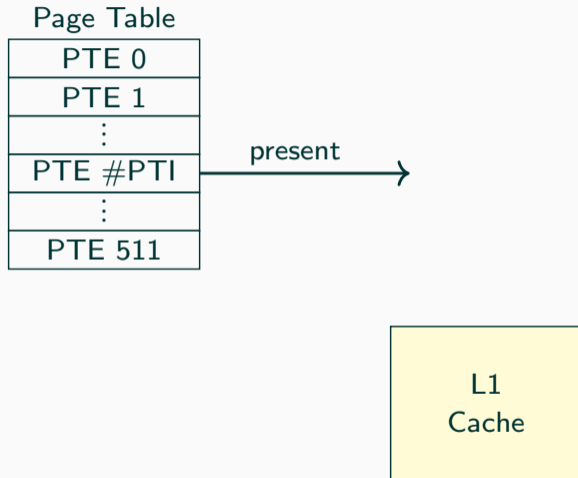
- An even **worse** bug → Foreshadow-NG/L1TF
- Exploitable from **VMs**
- Allows **leaking** data from the **L1** cache
- Same mechanism as Meltdown
- Just a **different bit** in the PTE

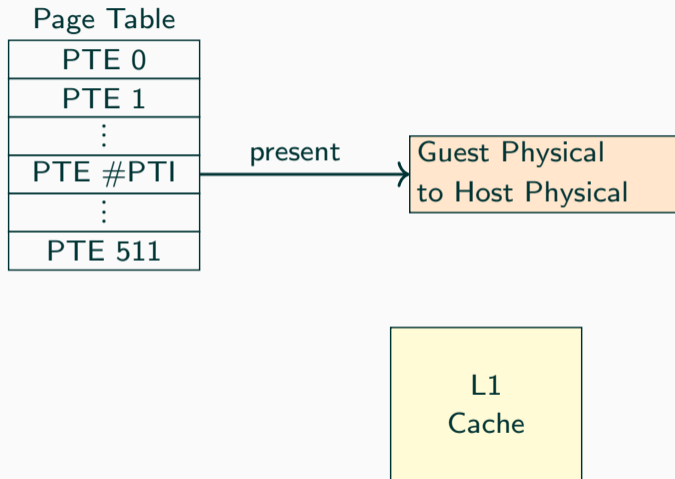
Page Table

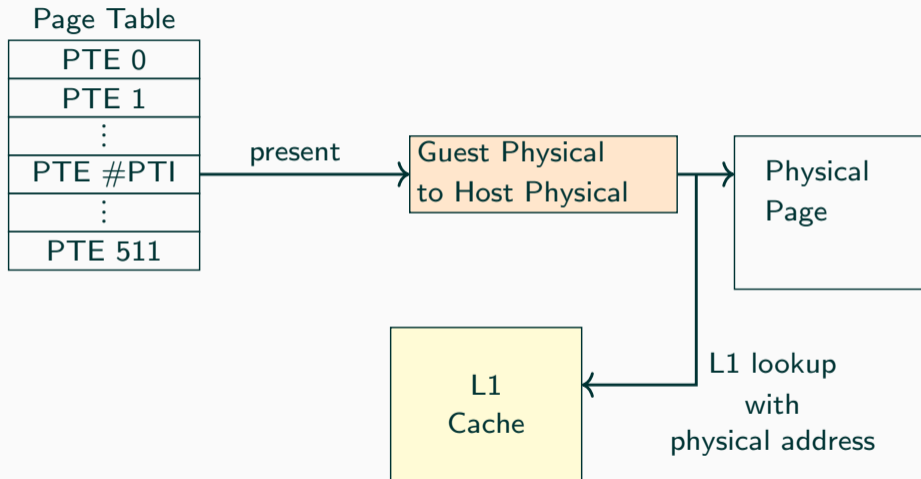
PTE 0
PTE 1
⋮
PTE #PTI
⋮
PTE 511



L1  
Cache







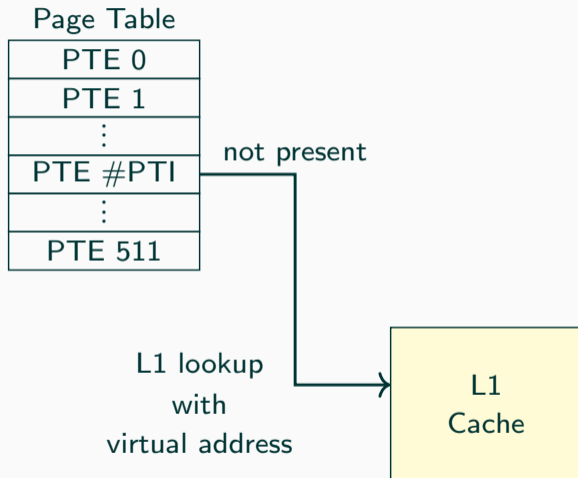
Page Table

PTE 0
PTE 1
⋮
PTE #PTI
⋮
PTE 511

not present



L1  
Cache





- KAISER/KPTI/KVA does not help





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



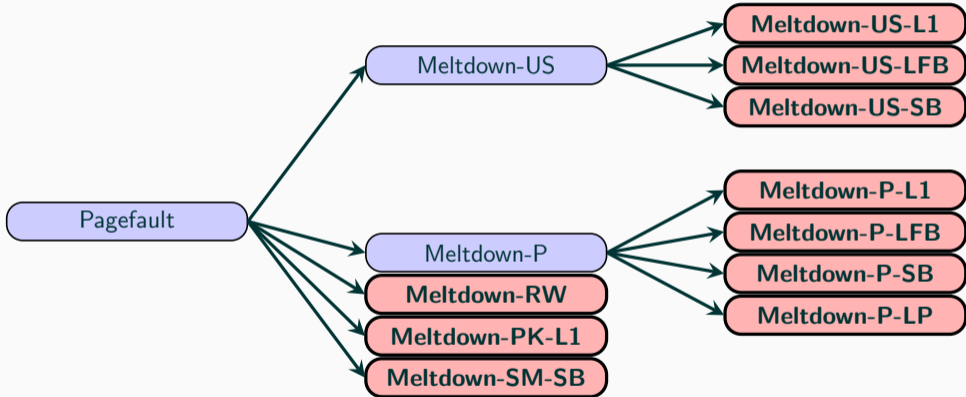
- KAISER/KPTI/KVA does not help
- Only **software workarounds**
  - **Flush L1** on VM entry



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

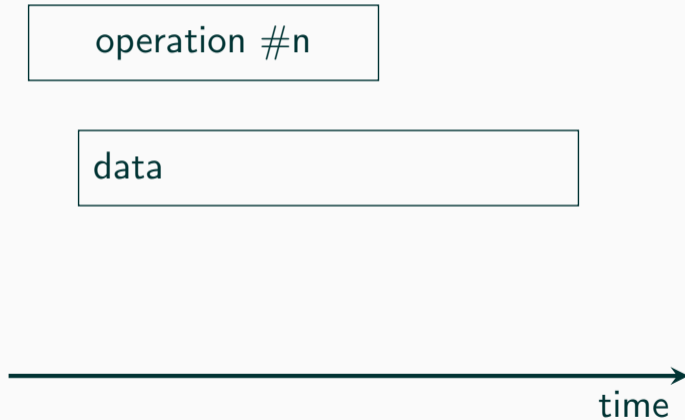


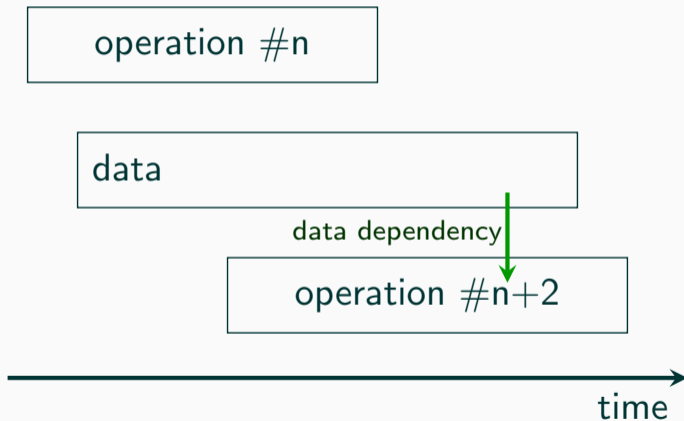
- KAISER/KPTI/KVA does not help
- Only **software workarounds**
  - **Flush L1** on VM entry
  - Disable **HyperThreading**
- Workarounds might not be complete



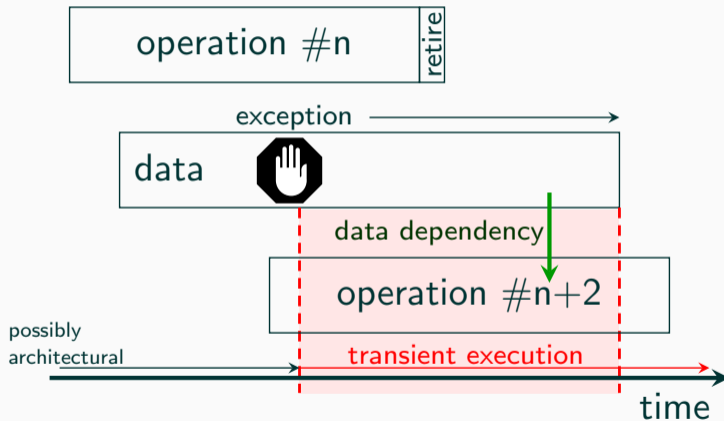
operation #n

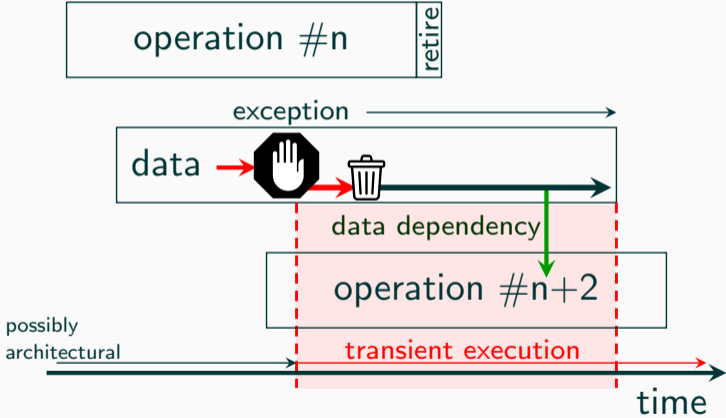


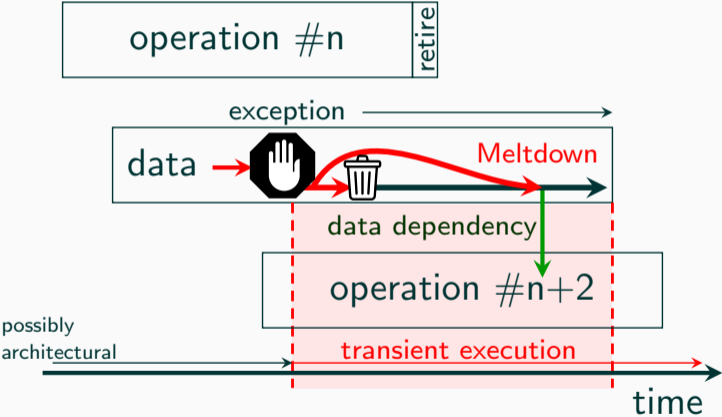


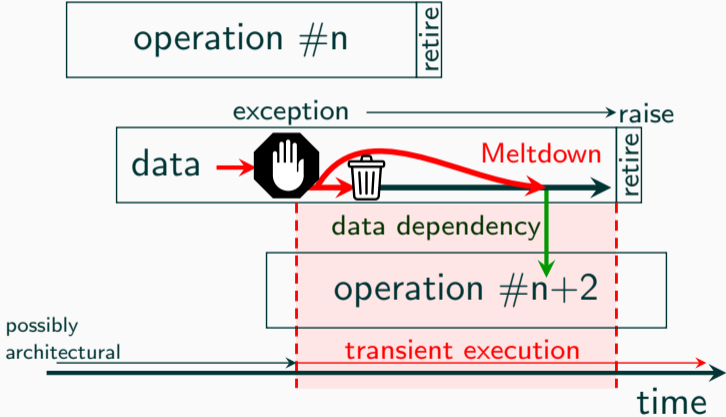








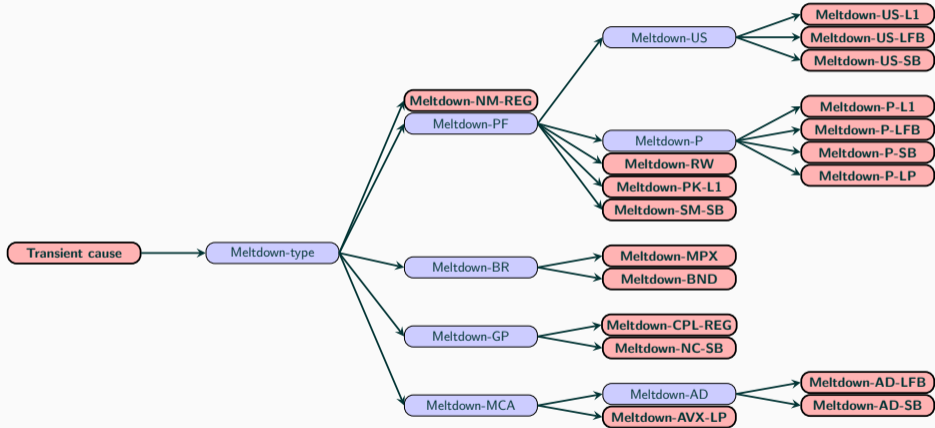


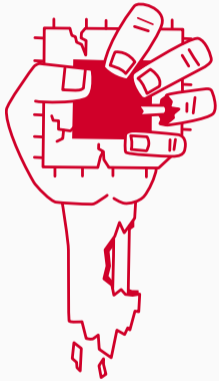


**YOU GET A FAULT**

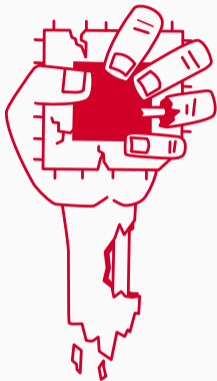


**AND YOU GET A FAULT.  
EVERYONE GETS A FAULT**



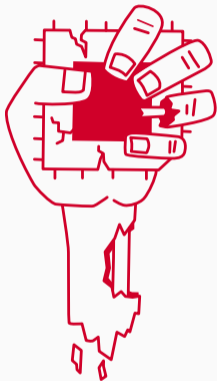


- Leaks from the **fill buffer**

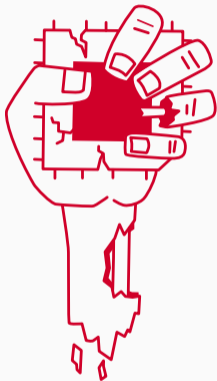


- Leaks from the **fill buffer**
- Crosses all privilege boundaries (Kernel, VM, SGX)





- Leaks from the **fill buffer**
- Crosses all privilege boundaries (Kernel, VM, SGX)
- Explored microcode assists as new type of faults



- Leaks from the **fill buffer**
- Crosses all privilege boundaries (Kernel, VM, SGX)
- Explored microcode assists as new type of faults
- Disadvantage: **minimal control** over leaked data



- 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
- **More** Meltdown-type **issues** to come



- Meltdown is **not** a fully **solved** issue
- The tree is extensible
- **More** Meltdown-type **issues** to come
- 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



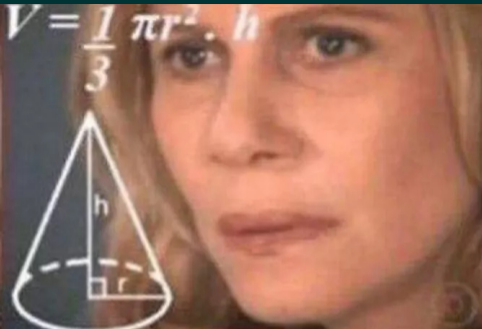
- CPU tries to predict the future (branch predictor), ...
  - ...based on events learned in the past
- **Speculative execution** of instructions
- If the prediction was correct, ...



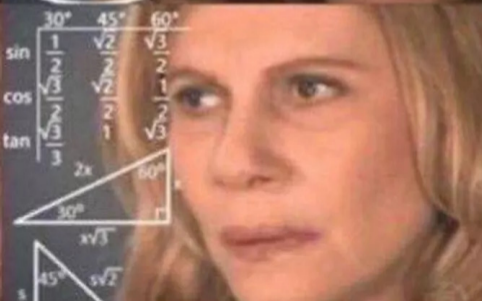
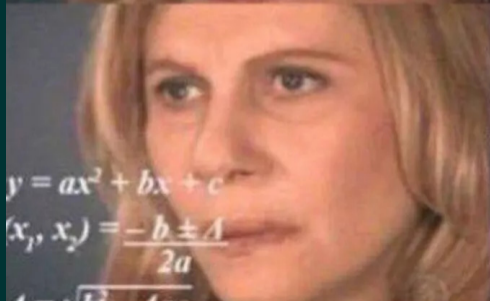
- CPU tries to predict the future (branch predictor), ...
  - ...based on events learned in the past
- **Speculative execution** of instructions
- If the prediction was correct, ...
  - ...very fast



- 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



$$V = \frac{1}{3} \pi r^2 \cdot h$$



$$y = ax^2 + bx + c$$
$$(x_1, x_2) = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

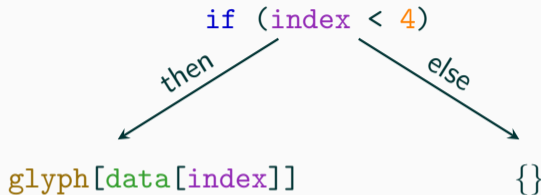
	30°	45°	60°
sin	$\frac{1}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{\sqrt{3}}{2}$
cos	$\frac{\sqrt{3}}{2}$	$\frac{\sqrt{2}}{2}$	$\frac{1}{2}$
tan	$\frac{1}{\sqrt{3}}$	1	$\sqrt{3}$



`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



Memory

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

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]]
```

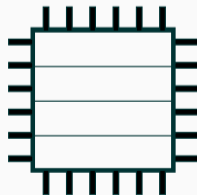
else

Speculate

```
{ }
```

Memory

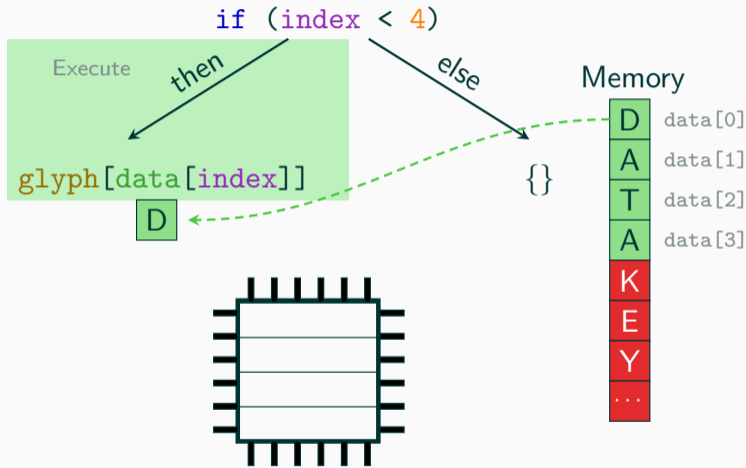
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

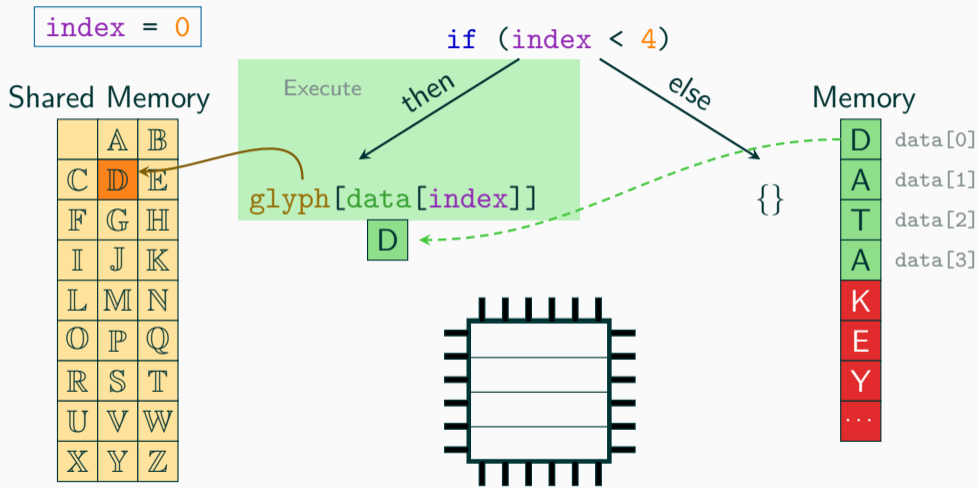


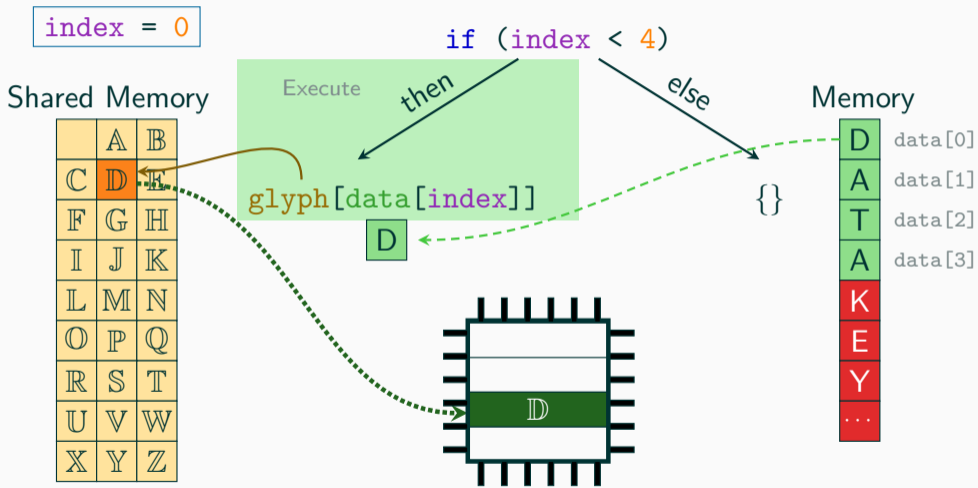
index = 0

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



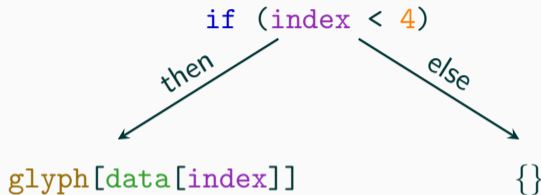




`index = 1`

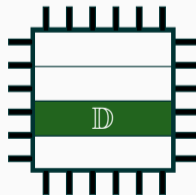
Shared Memory

	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



Memory

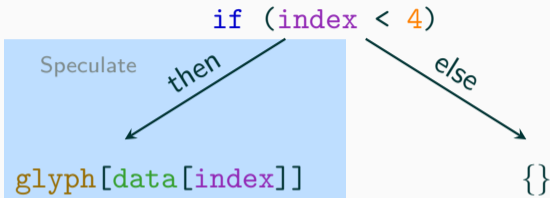
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	



index = 1

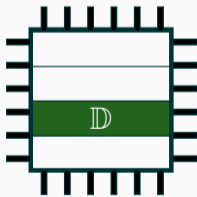
Shared Memory

	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



Memory

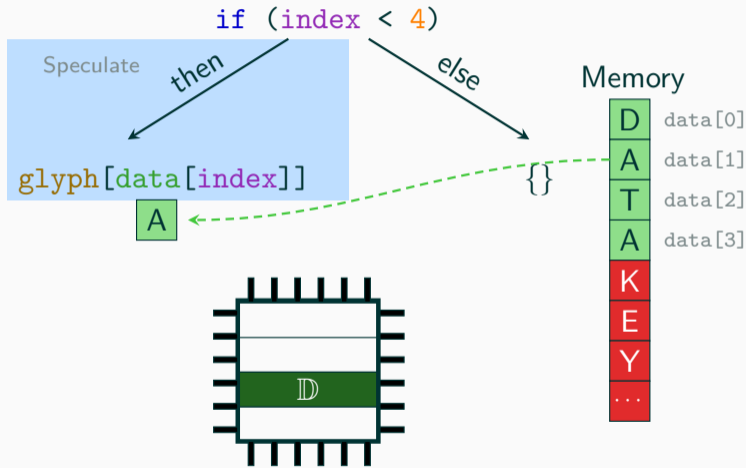
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	



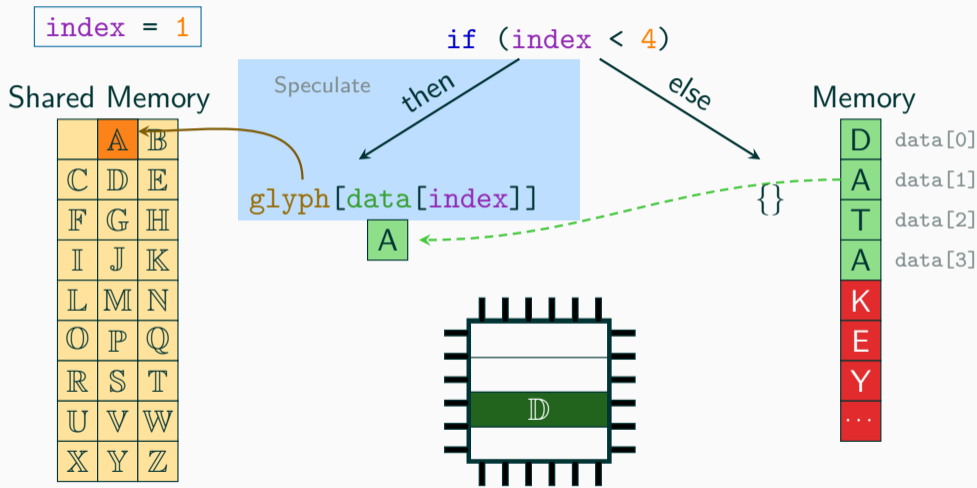
index = 1

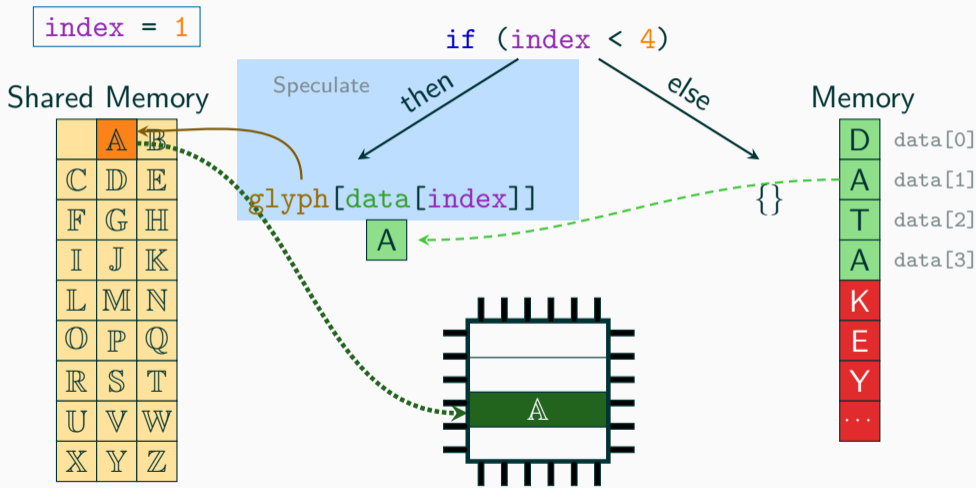
Shared Memory

	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









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)
```

Execute

then

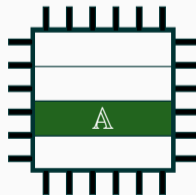
```
glyph[data[index]]
```

else

```
{
```

Memory

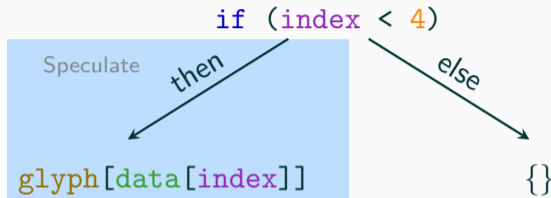
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	



`index = 2`

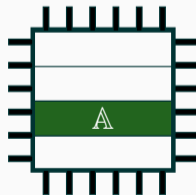
Shared Memory

	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

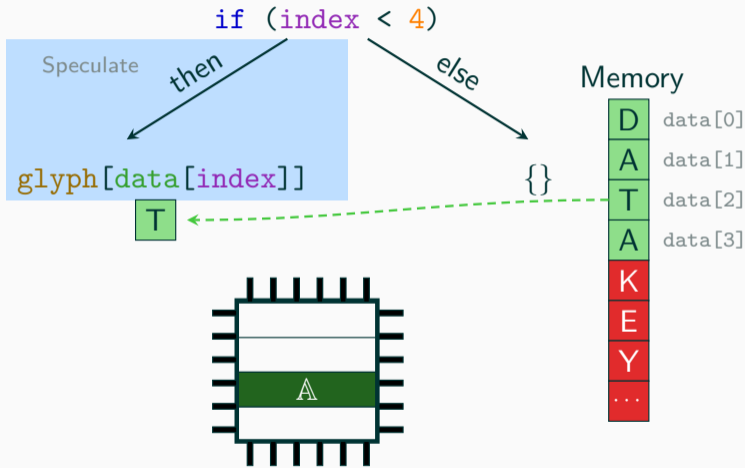
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

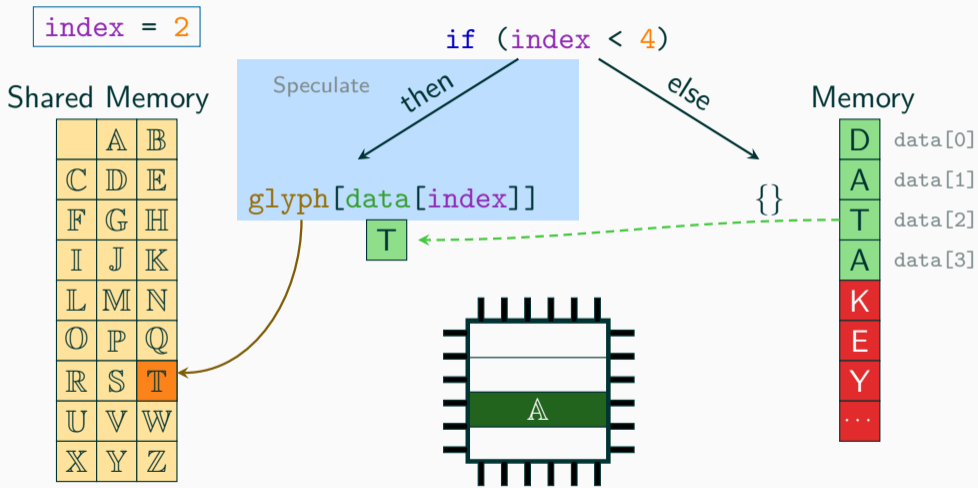


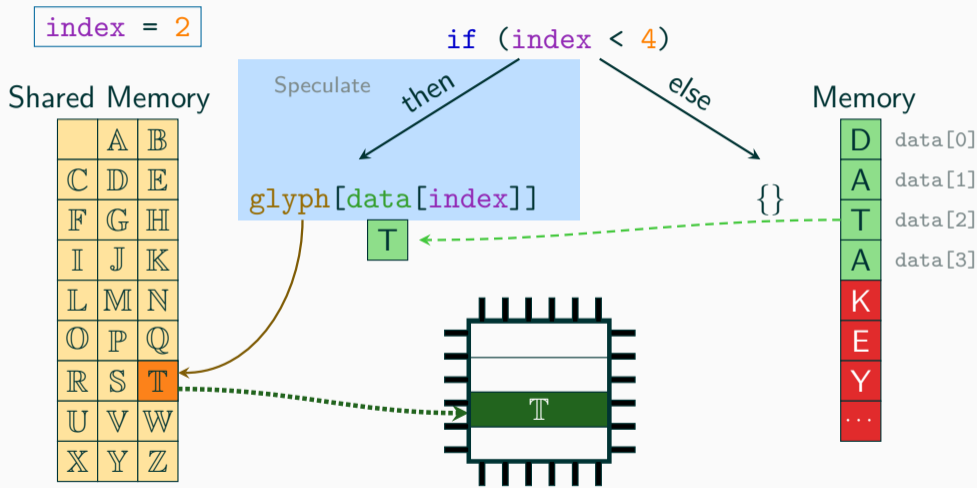
index = 2

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



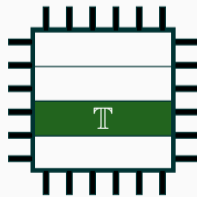
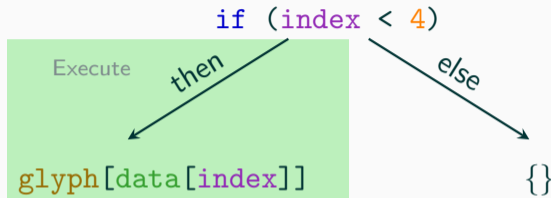




index = 2

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

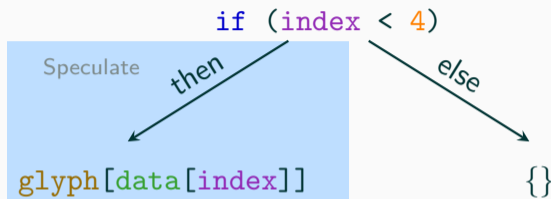
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	



`index = 3`

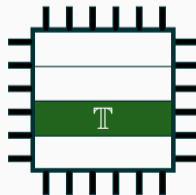
Shared Memory

	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

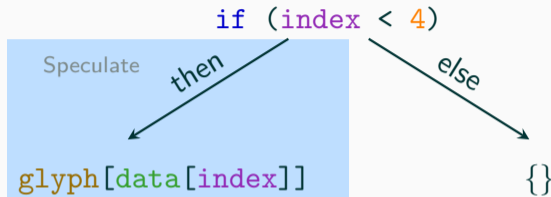
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	



index = 3

Shared Memory

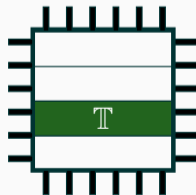
	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

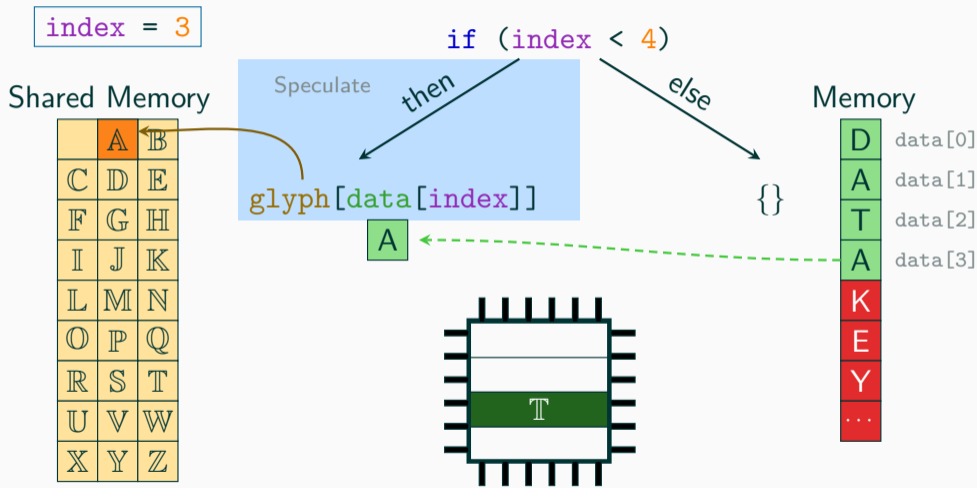


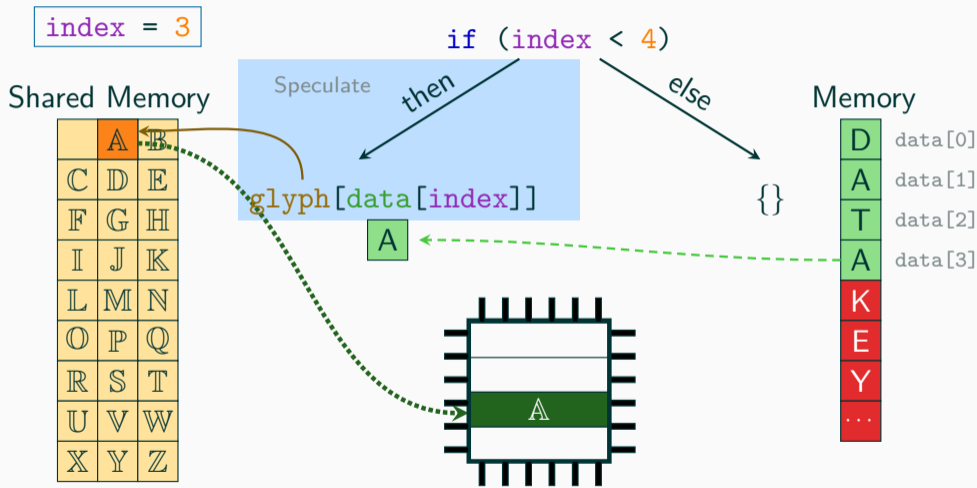
A

Memory

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



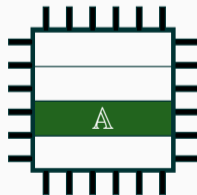
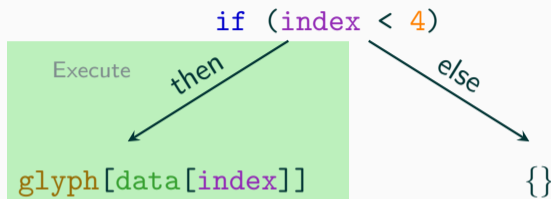




index = 3

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



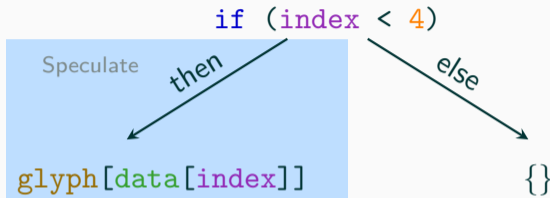
Memory

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

`index = 4`

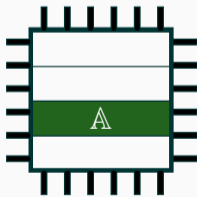
Shared Memory

	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

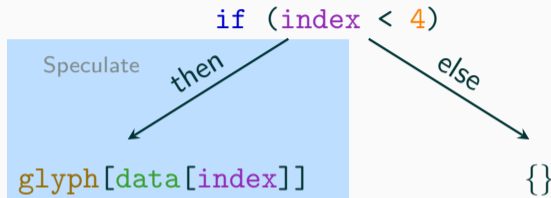
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	



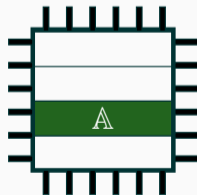
index = 4

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

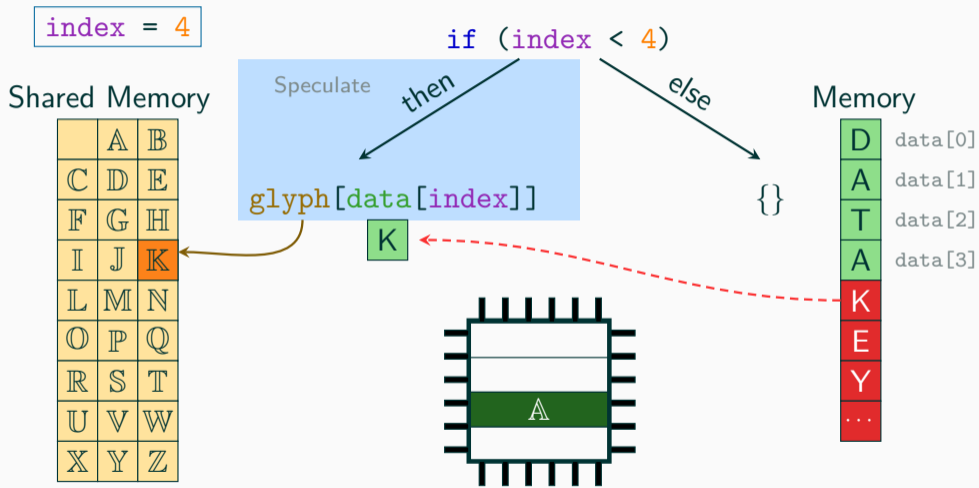


K

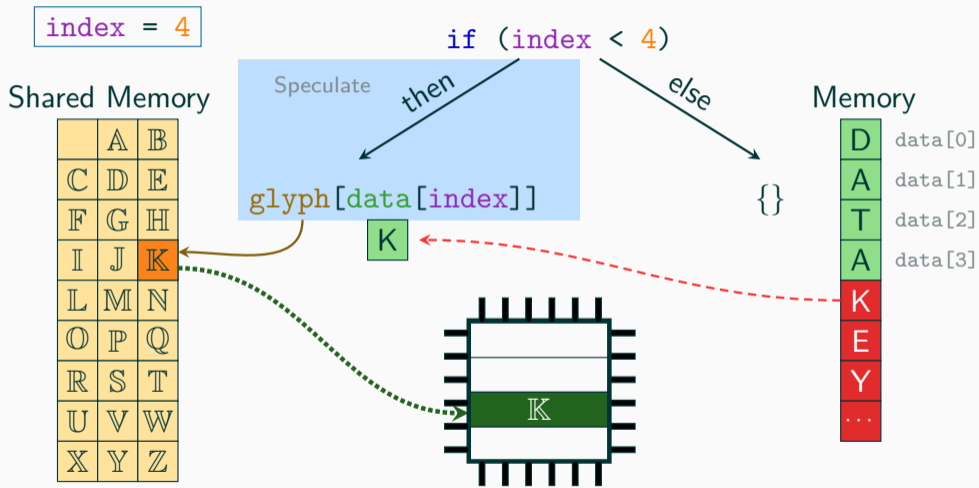


Memory

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



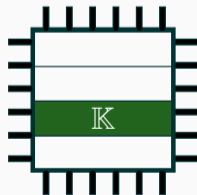
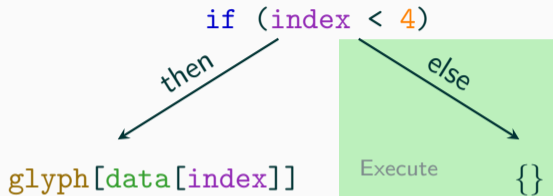




index = 4

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

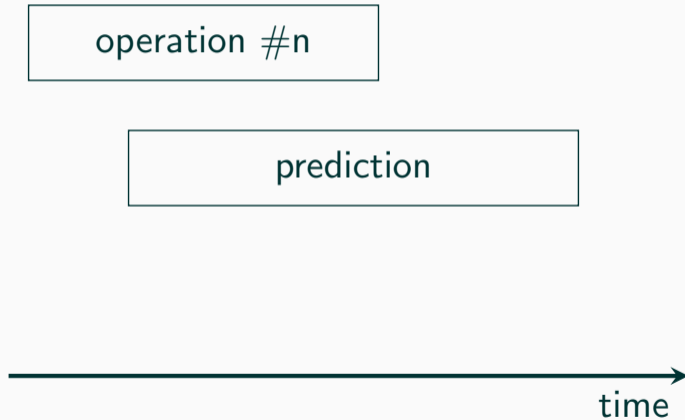


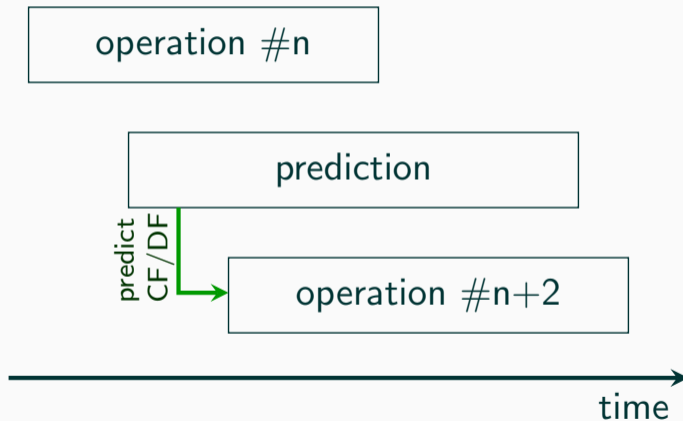
Memory

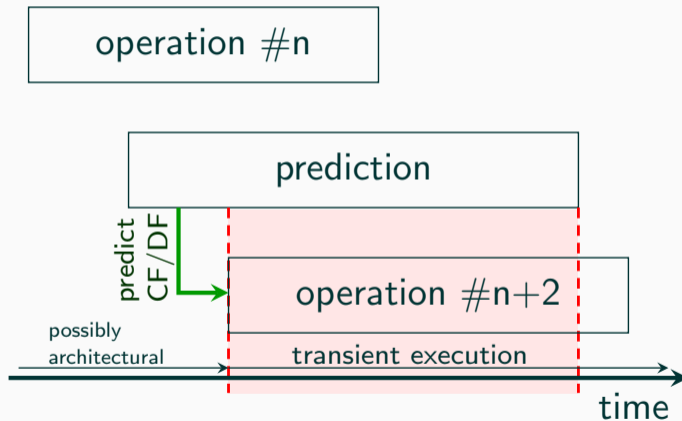
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

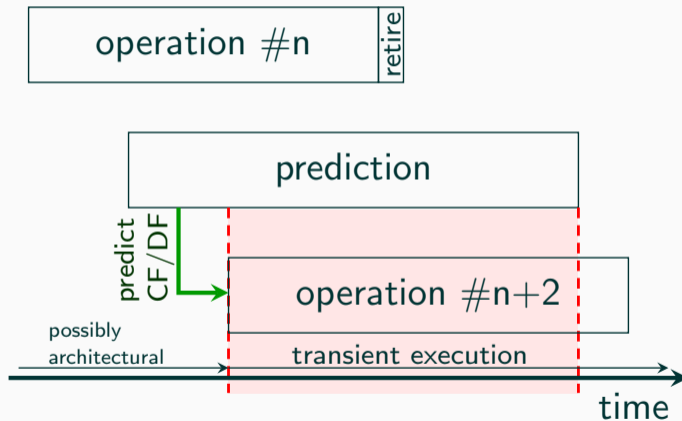
operation #n

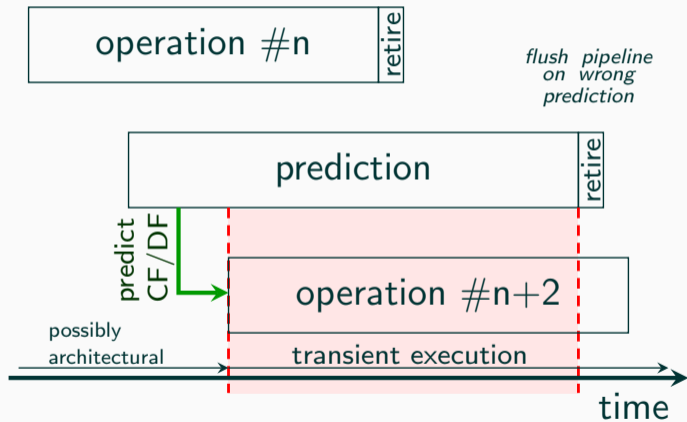




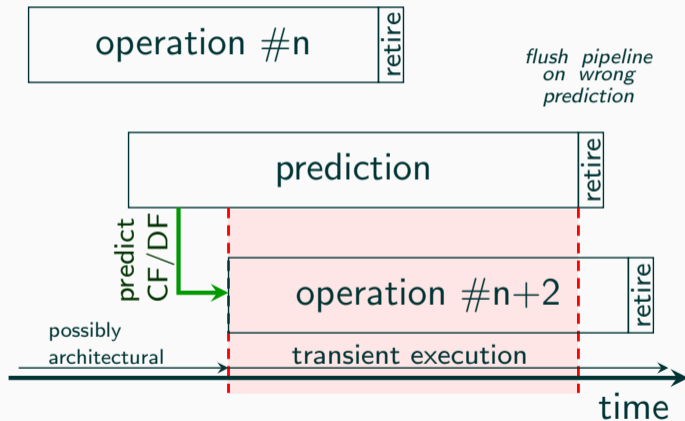


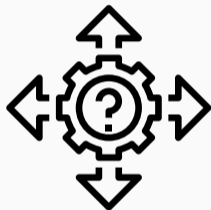












- Many predictors in modern CPUs



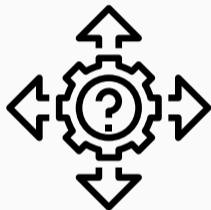
- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)



- 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)



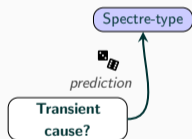
- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)
  - Call/Jump destination (BTB)
  - Function return destination (RSB)
  - Load matches previous store (STL)

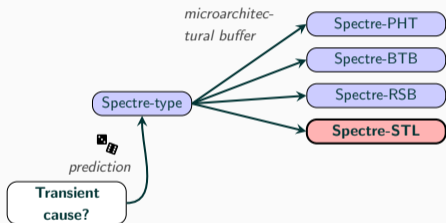


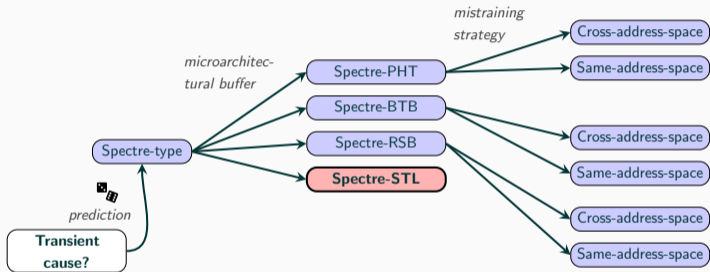
- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)
  - Call/Jump destination (BTB)
  - Function return destination (RSB)
  - Load matches previous store (STL)
- Most are even shared among processes

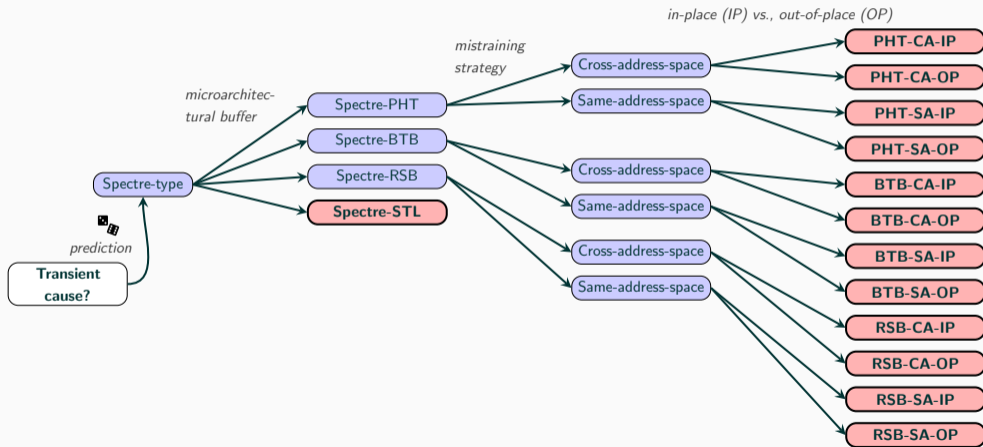
Transient  
cause?













- 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



# Linux 4.19.4 & 4.14.83 Released With STIBP Code Dropped

Written by [Michael Larabel](#) in [Linux Kernel](#) on 24 November 2018 at 09:00 AM EST. [6 Comments](#)



On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

Greg Kroah-Hartman issued this latest round of stable point releases as basic maintenance updates. While these point releases don't tend to be too notable and generally go unmentioned on Phoronix, this round is worth pointing out since 4.19.4 and 4.14.83 are the releases that end up [reverting the STIBP behavior](#) that applied Single Thread Indirect Branch Predictors to all processes on supported systems. That is what was introduced in Linux 4.20 and then back-ported to the 4.19/4.14 LTS branches, which in turn hurt the performance a lot. So for now the code is removed.

As covered yesterday, [there is improved STIBP code on the way](#) for Linux 4.20 that by default just apply STIBP to SECCOMP threads and processes requesting it via prctl() but otherwise is off by default (that behavior can also be changed via kernel parameters).

# Linux 4.19.4 & 4.14.83 Released With STIBP Code Dropped

Written by [Michael Larabel](#) in [Linux Kernel](#) on 24 November 2018 at 09:00 AM EST. [6 Comments](#)



On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

Greg Kroah-Hartman issued this latest round of stable point releases as basic maintenance updates. While these point releases don't tend to be too notable and generally go unmentioned on Phoronix, this round is worth pointing out since 4.19.4 and 4.14.83 are the releases that end up [reverting the STIBP behavior](#) that applied Single Thread Indirect Branch Predictors to all processes on supported systems. That is what was introduced in Linux 4.20 and then back-ported to the 4.19/4.14 LTS branches, which in turn hurt the performance a lot. So for now the code is removed.

As covered yesterday, [there is improved STIBP code on the way](#) for Linux 4.20 that by default just apply STIBP to SECCOMP threads and processes requesting it via `prctl()` but otherwise is off by default (that behavior can also be changed via kernel parameters).

# Linux 4.19.4 & 4.14.83 Released With STIBP Code Dropped

Written by Michael Larabel in Linux Kernel on 24 November 2018 at 09:00 AM EST. 6 Comments



On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

Greg Kroah-Hartman issued this latest round of stable point releases as basic maintenance updates. While these point releases don't tend to be too notable and generally go unmentioned on Phoronix, this round is worth pointing out since 4.19.4 and 4.14.83 are the releases that end up reverting the STIBP behavior that applied Single Thread Indirect Branch Predictors to all processes on supported systems. That is what was introduced in Linux 4.20 and then back-ported to the 4.19/4.14 LTS branches, which in turn hurt the performance a lot. So for now the code is removed.

As covered yesterday, there is improved STIBP code on the way for Linux 4.20 that by default just apply STIBP to SECCOMP threads and processes requesting it via prctl() but otherwise is off by default (that behavior can also be changed via kernel parameters).

# Linux 4.19.4 & 4.14.83 Released With STIBP Code Dropped

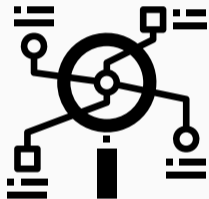
Written by Michael Larabel in Linux Kernel on 24 November 2018 at 09:00 AM EST. 6 Comments



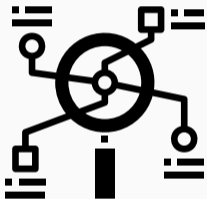
On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

Greg Kroah-Hartman issued this latest round of stable point releases as basic maintenance updates. While these point releases don't tend to be too notable and generally go unmentioned on Phoronix, this round is worth pointing out since 4.19.4 and 4.14.83 are the releases that end up reverting the STIBP behavior that applied Single Thread Indirect Branch Predictors to all processes on supported systems. That is what was introduced in Linux 4.20 and then back-ported to the 4.19/4.14 LTS branches, which in turn hurt the performance a lot. So for now the code is removed.

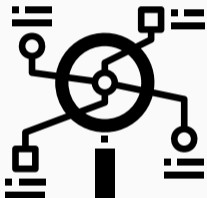
As covered yesterday, there is improved STIBP code on the way for Linux 4.20 that by default just apply STIBP to SECCOMP threads and processes requesting it via prctl() but otherwise is off by default (that behavior can also be changed via kernel parameters).



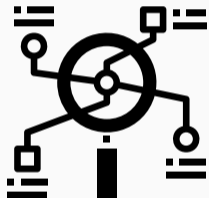
- Current mitigations are either **incomplete** or **cost performance**



- 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**

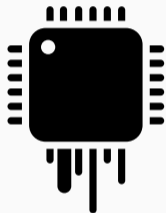


- Current mitigations are either **incomplete or cost performance**
- More **research** required
- Both on **attacks and defenses**
- Efficient defenses only possible when attacks are known

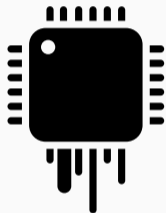




- Side channels so far
  - leak meta data



- Side channels so far
  - leak meta data
  - covertly transmit data

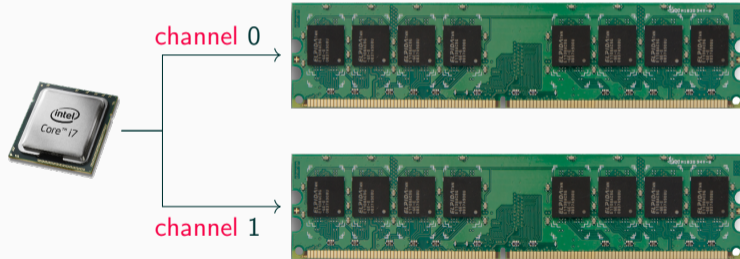


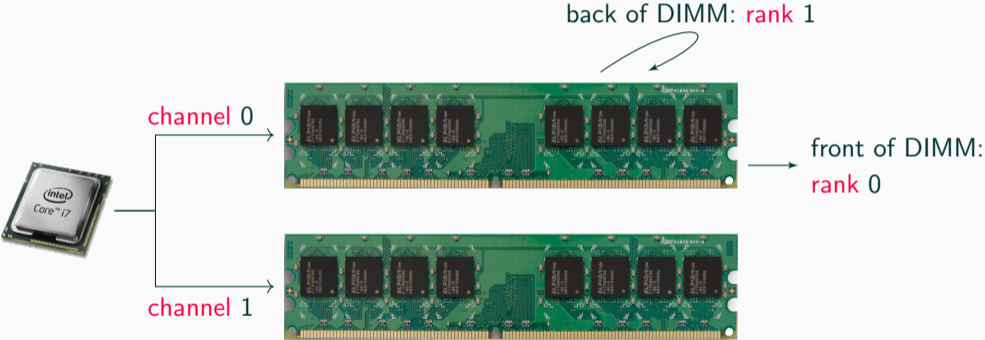
- Side channels so far
  - leak meta data
  - covertly transmit data
- As a building block
  - leak data

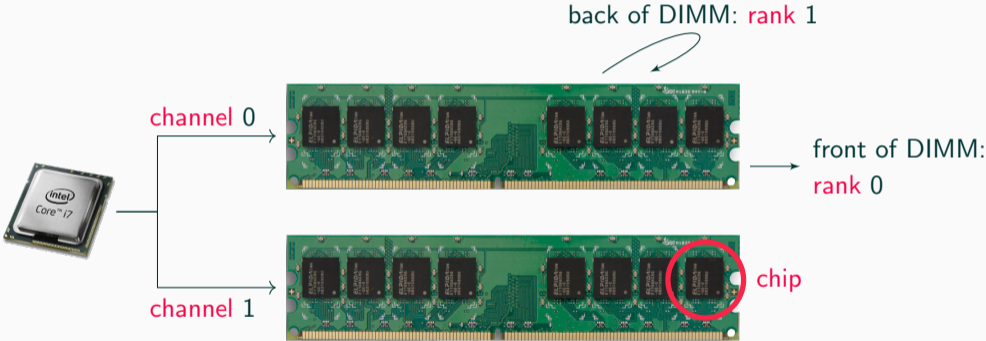


- Side channels so far
  - leak meta data
  - covertly transmit data
- As a building block
  - leak data
- What about modifying data?



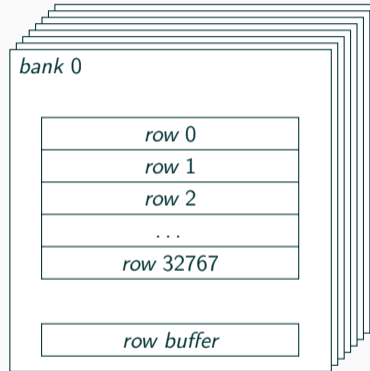




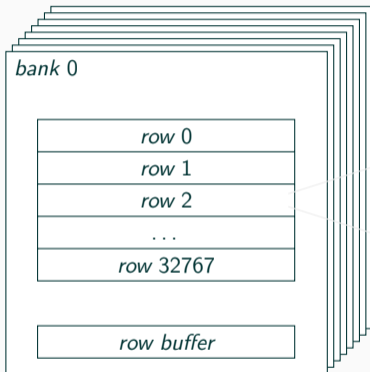




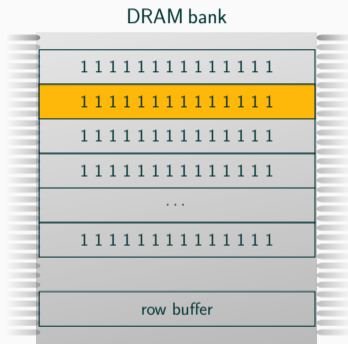
chip



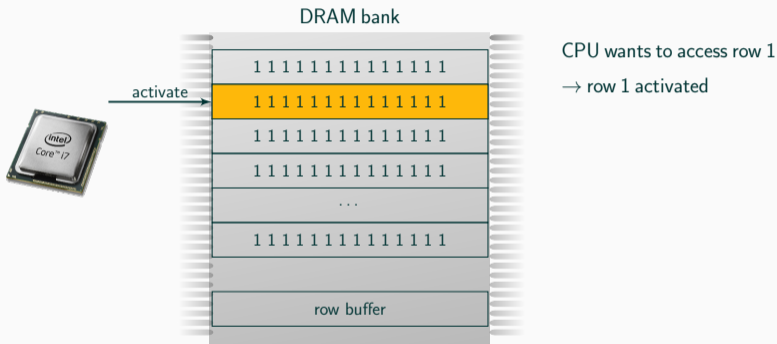
chip

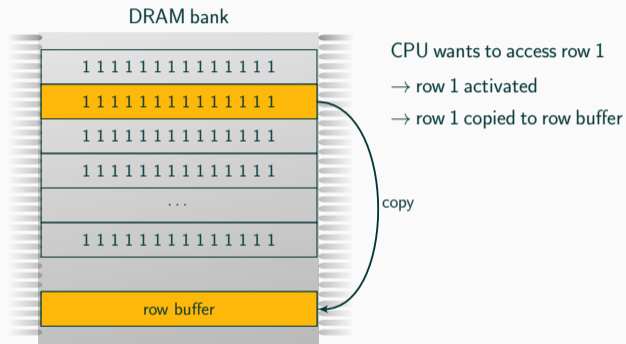


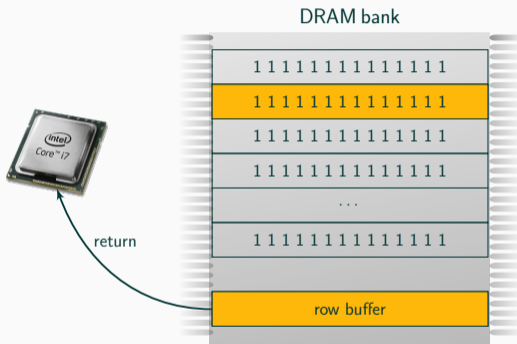
64k cells  
1 capacitor,  
1 transistor each



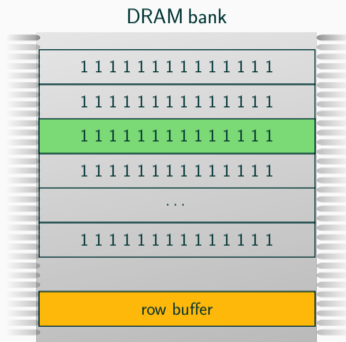
CPU wants to access row 1



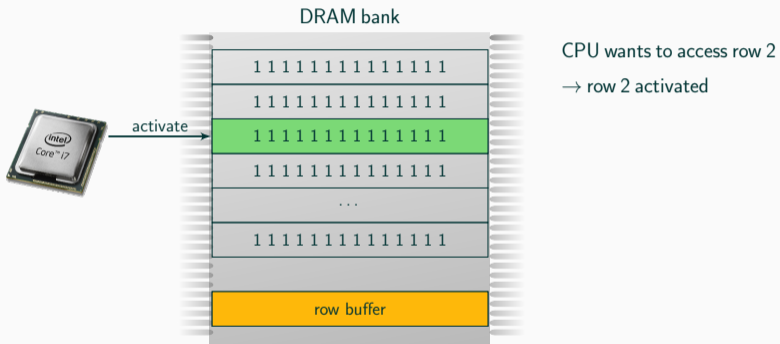




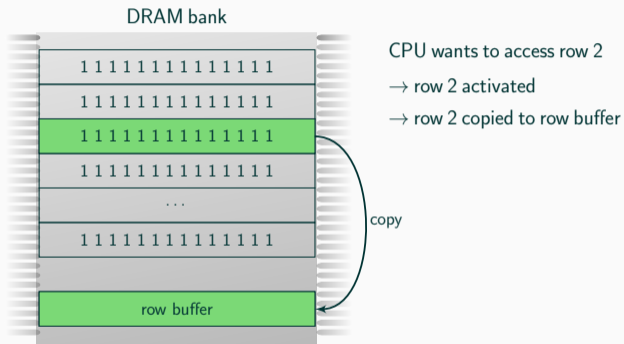
CPU wants to access row 1  
→ row 1 activated  
→ row 1 copied to row buffer

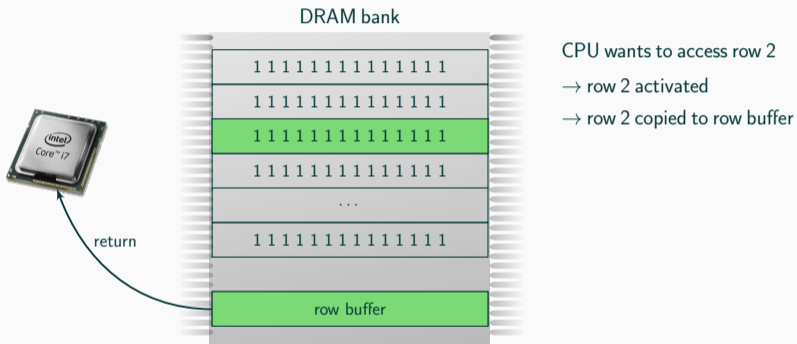


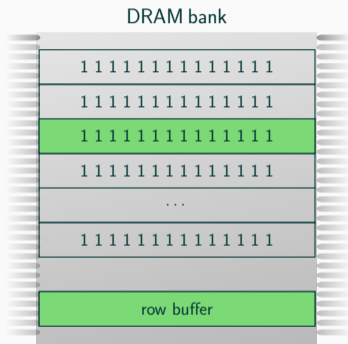
CPU wants to access row 2









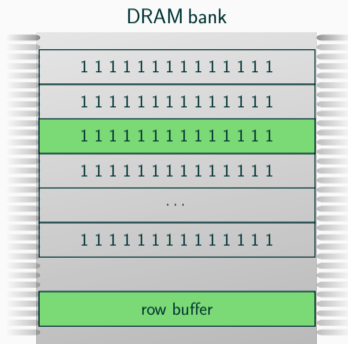


CPU wants to access row 2

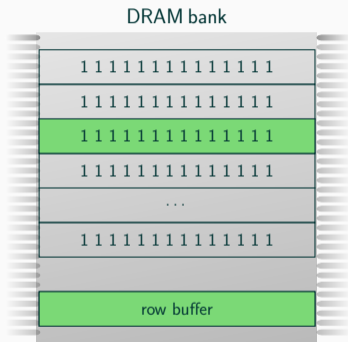
→ row 2 activated

→ row 2 copied to row buffer

→ **slow** (row conflict)

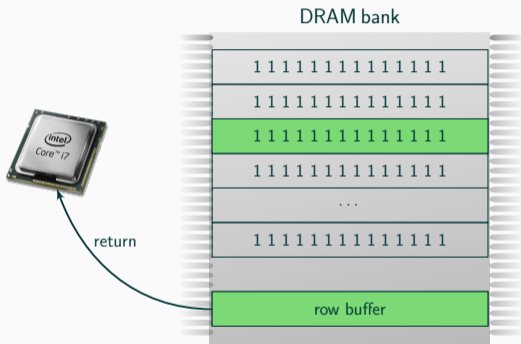


CPU wants to access row 2—again



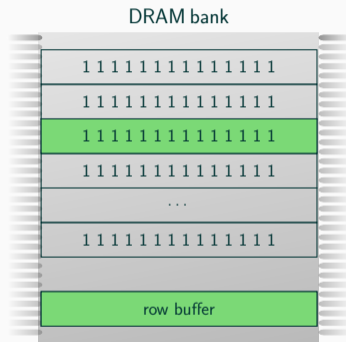
CPU wants to access row 2—again

→ row 2 already in row buffer



CPU wants to access row 2—again

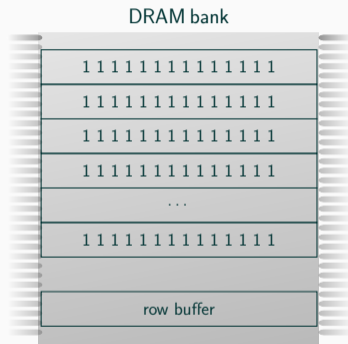
→ row 2 already in row buffer



CPU wants to access row 2—again

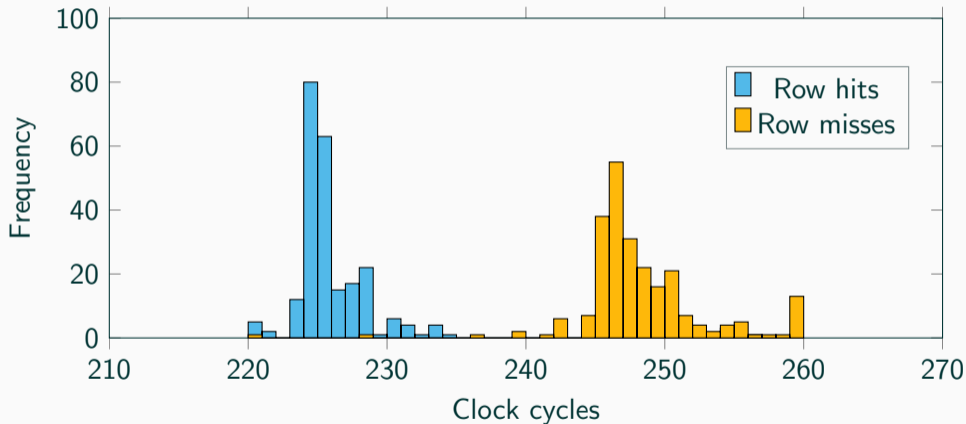
→ row 2 already in row buffer

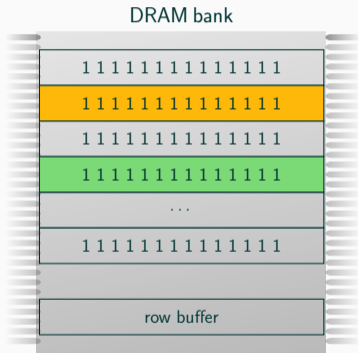
→ **fast** (row hit)

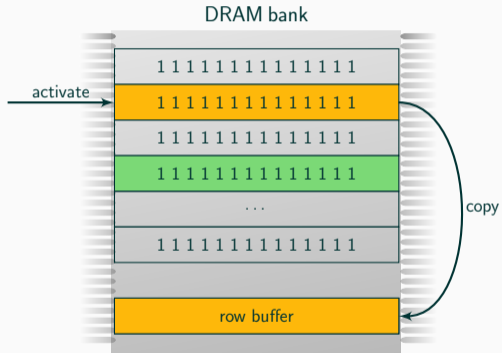


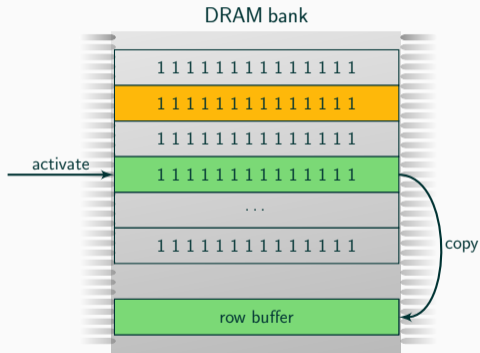
**row buffer = cache**



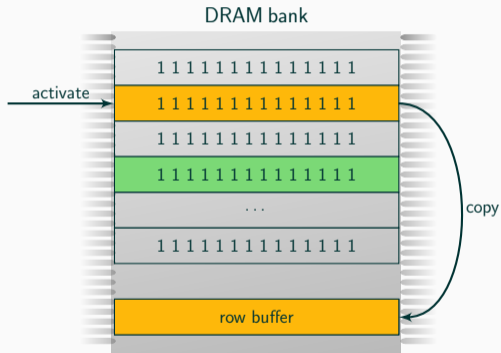




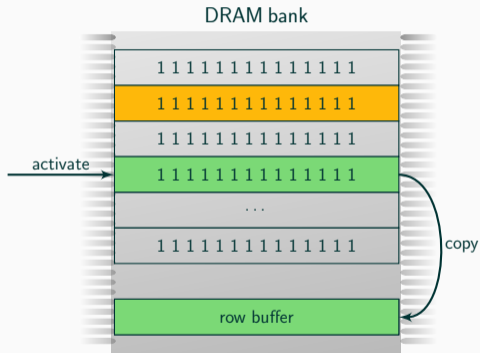




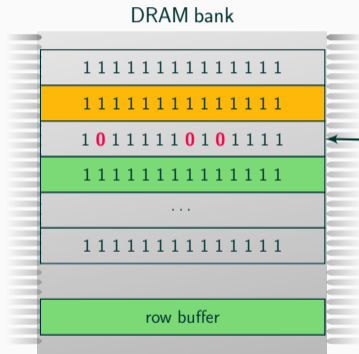
Cells leak faster upon proximate accesses → Rowhammer



Cells leak faster upon proximate accesses → Rowhammer



Cells leak faster upon proximate accesses → Rowhammer



bit flips in row 2!



Cells leak faster upon proximate accesses → Rowhammer



- 85% affected (estimation 2014)
- 52% affected (estimation 2015)





- 85% affected (estimation 2014)
- 52% affected (estimation 2015)



- First believed to be safe
- We showed bit flips in 2016
- 67% affected (estimation 2016)



- Single bit flips allow
  - modifying instructions



- Single bit flips allow
  - modifying instructions
  - breaking cryptography



- Single bit flips allow
  - modifying instructions
  - breaking cryptography
  - changing permissions



- Single bit flips allow
  - modifying instructions
  - breaking cryptography
  - changing permissions
  - crashing systems



- Single bit flips allow
  - modifying instructions
  - breaking cryptography
  - changing permissions
  - crashing systems
  - ...



- Single bit flips allow
  - modifying instructions
  - breaking cryptography
  - changing permissions
  - crashing systems
  - ...
- In software, no permissions required

- Program containing **conditional jump** after password check:  
`je 80486c1 <check_password+0x44>`





- Program containing **conditional jump** after password check:  
`je 80486c1 <check_password+0x44>`
- Machine code is  
`0x74 0x07`





- Program containing **conditional jump** after password check:  
`je 80486c1 <check_password+0x44>`
- Machine code is  
`0x74 0x07 = 0b01110100 0b00000111`

- Program containing **conditional jump** after password check:  
`je 80486c1 <check_password+0x44>`
- Machine code is  
`0x74 0x07 = 0b01110100 0b00000111`



JE

0 1 1 1 0 1 0 0



- Program containing **conditional jump** after password check:  
`je 80486c1 <check_password+0x44>`
- Machine code is  
`0x74 0x07 = 0b01110100 0b00000111`







- Program containing **conditional jump** after password check:  
`je 80486c1 <check_password+0x44>`
- Machine code is  
`0x74 0x07 = 0b01110100 0b00000111`



- Program containing **conditional jump** after password check:  
`je 80486c1 <check_password+0x44>`
- Machine code is  
`0x74 0x07 = 0b01110100 0b00000111`









- Program containing **conditional jump** after password check:  
je 80486c1 <check\_password+0x44>
- Machine code is  
0x74 0x07 = 0b01110100 0b00000111

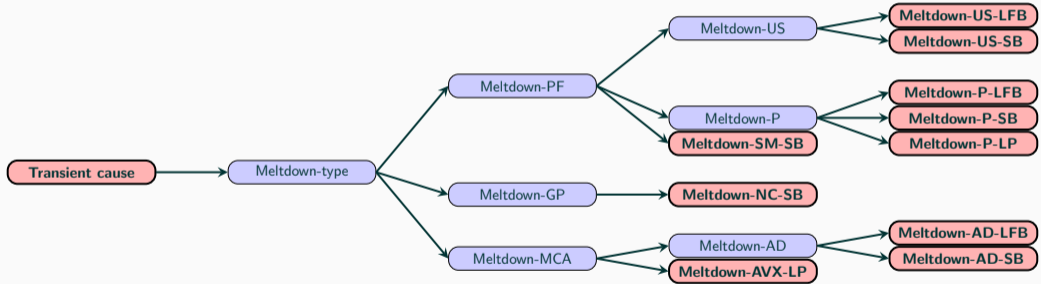


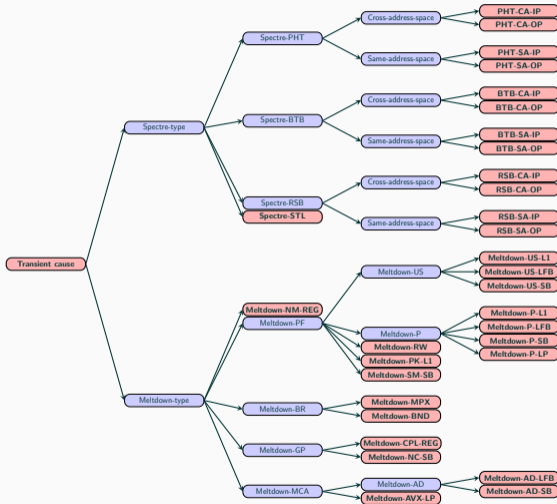




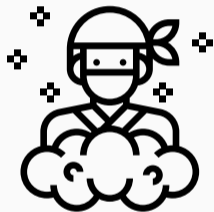


- More attacks **exploiting performance optimizations** in hardware
  - **New variants** are disclosed frequently

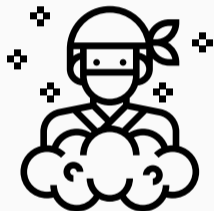




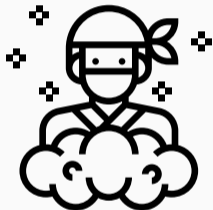




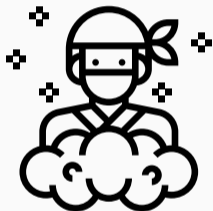
- Transient Execution Attacks are...



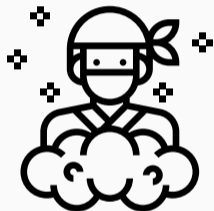
- **Transient Execution Attacks** are...
  - ...a **novel class** of attacks



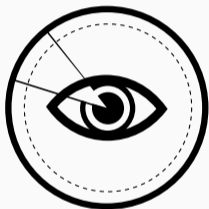
- **Transient Execution Attacks** are...
  - ...a **novel class** of attacks
  - ...extremely **powerful**



- **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



A **unique chance** to

- rethink processor design
- grow up, like other fields (car industry, construction industry)

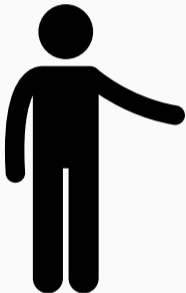


- **Optimizations** in hardware often lead to side channels



- **Optimizations** in hardware often lead to side channels
- Unknown and **novel** side channels are likely to **exist**

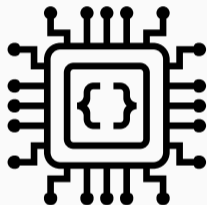




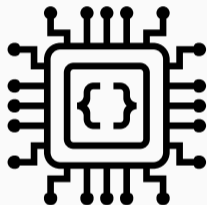
- Optimizations in hardware often lead to side channels
- Unknown and novel side channels are likely to exist
- Next to no permissions required for attacks



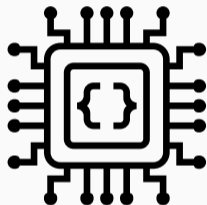
- Optimizations in hardware often lead to side channels
- Unknown and novel side channels are likely to exist
- Next to no permissions required for attacks
- Building countermeasures is extremely hard



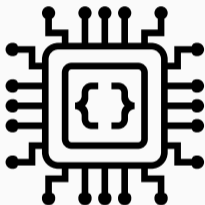
- Only an **overview** over some attacks



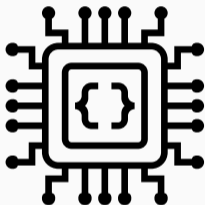
- Only an **overview** over some attacks
- Many more side-channel attacks



- Only an **overview** over some attacks
- Many more side-channel attacks
- Also some defenses, especially for crypto



- Only an **overview** over some attacks
- Many more side-channel attacks
- Also some defenses, especially for crypto
- Master course: **Embedded Security**



- Only an **overview** over some attacks
- Many more side-channel attacks
- Also some defenses, especially for crypto
- Master course: **Embedded Security**
- Talks from our group on YouTube: “InfoSec @ TU Graz”



IAIK  TU  
Graz  
Technical University of Applied Sciences

**Bachelor@IAIK 2020**  
Themen.Vorstellung.

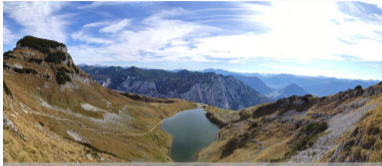
**+ Student Research  
Awards 2019**

Freitag 29. Nov., 12:00 - 13:00  
IAIK-Foyer  
Inffeldgasse 16a, Erdgeschoß.

[www.iaik.tugraz.at](http://www.iaik.tugraz.at)

- Looking for a **bachelor thesis**?





IAIK  TU  
Graz  
Technical University of Applied Sciences

**Bachelor@IAIK 2020**  
Themen.Vorstellung.

**+ Student Research  
Awards 2019**

Freitag 29. Nov., 12:00 - 13:00  
IAIK-Foyer  
Inffeldgasse 16a, Erdgeschoß.

[www.iaik.tugraz.at](http://www.iaik.tugraz.at)

- Looking for a **bachelor thesis**?
- Our institute presents many topics



IAIK  TU  
Graz  
Graz University of Technology

## Bachelor@IAIK 2020

Themen.Vorstellung.

### + Student Research Awards 2019

Freitag 29. Nov., 12:00 - 13:00  
IAIK-Foyer  
Inffeldgasse 16a, Erdgeschoß.

[www.iaik.tugraz.at](http://www.iaik.tugraz.at)

- Looking for a **bachelor thesis**?
- Our institute presents many topics
- **Awards** for students who contributed to **publications**

**Any Questions?**