Microarchitectural Side-Channel Attacks

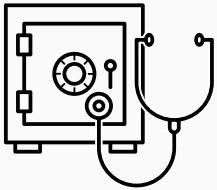
Michael Schwarz

September 18, 2019

www.iaik.tugraz.at

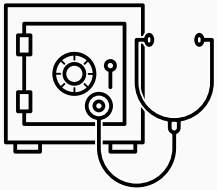
Side-channel Attacks

- Safe software infrastructure does not mean safe execution

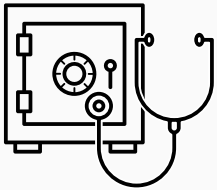


Side-channel Attacks

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

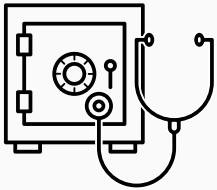


Side-channel Attacks



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

Side-channel Attacks

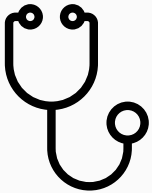


- Safe software infrastructure does not mean safe execution
- Information leaks because of the underlying hardware
- Exploit unintentional information leakage by side-effects

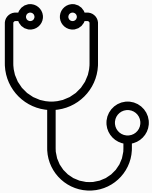
Power consumption

Execution time

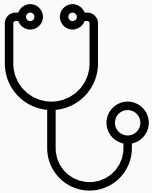
CPU caches



- 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

Example: PIN Comparison

- 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;
}
```

Example: PIN Comparison

- Measuring the execution times for different PINs

PIN	Time

Example: PIN Comparison

- Measuring the execution times for different PINs

PIN	Time
0000	

Example: PIN Comparison

- Measuring the execution times for different PINs

PIN	Time
0000	
1000	

Example: PIN Comparison

- Measuring the execution times for different PINs

PIN	Time
0000	
1000	
2000	

Example: PIN Comparison

- Measuring the execution times for different PINs

PIN	Time
0000	
1000	
2000	
3000	

Example: PIN Comparison

- Measuring the execution times for different PINs

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

Example: PIN Comparison

- Measuring the execution times for different PINs

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

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

Example: PIN Comparison

- 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

Example: PIN Comparison

- Measuring the execution times for different PINs

PIN	Time

Example: PIN Comparison

- Measuring the execution times for different PINs

PIN	Time
1000	

Example: PIN Comparison

- Measuring the execution times for different PINs

PIN	Time
1000	
1100	

Example: PIN Comparison

- Measuring the execution times for different PINs

PIN	Time
1000	
1100	
1200	

Example: PIN Comparison

- Measuring the execution times for different PINs

PIN	Time
1000	
1100	
1200	
1300	

Example: PIN Comparison

- Measuring the execution times for different PINs

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

Example: PIN Comparison

- Measuring the execution times for different PINs

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

- Repeat for every digit

Example: PIN Comparison

- Measuring the execution times for different PINs

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

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

Example: PIN Comparison

- Maximum 10 measurements per digit

Example: PIN Comparison

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

Example: PIN Comparison

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

Example: PIN Comparison

- 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

Example: PIN Comparison

- Many functions can be implemented with constant runtime

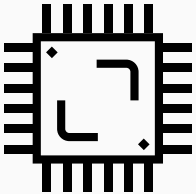
```
int check_pin(char* input) {
    const char* correct = "1234";
    int same = 0;
    for(int i = 0; i < 4; i++) {
        same |= correct[i] - input[i];
    }
    return (same == 0);
}
```

Example: PIN Comparison

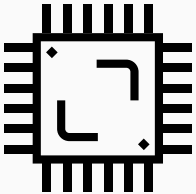
- Many functions can be implemented with constant runtime

```
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);
}
```

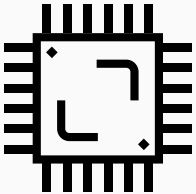
- Sometimes, there is still a side channel in the hardware



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

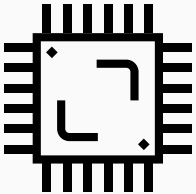


- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)
- Serves as the interface between hardware and software



- 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

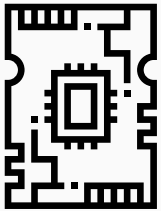
Architecture and Microarchitecture



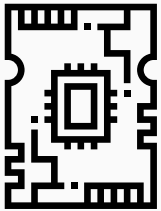
- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)
- Serves as the interface between hardware and software
- Microarchitecture is an actual implementation of the ISA

●●●

- Modern CPUs contain multiple microarchitectural elements



- Modern CPUs contain multiple microarchitectural elements

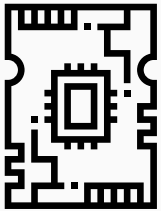


Caches and buffer

Predictor

Microarchitectural Elements

- Modern CPUs contain multiple microarchitectural elements

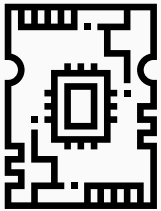


Caches and buffer

Predictor

- Transparent for the programmer

Microarchitectural Elements



- Modern CPUs contain multiple microarchitectural elements

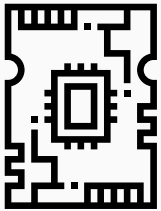
Caches and buffer

Predictor

- Transparent for the programmer
- Optimize program execution

Microarchitectural Elements

- Modern CPUs contain multiple microarchitectural elements

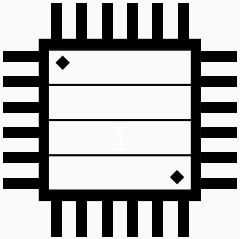


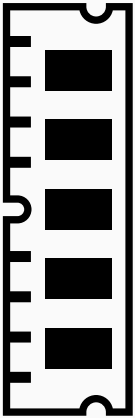
Caches and buffer

Predictor

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

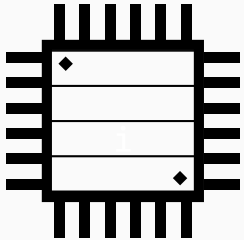
```
printf("%d", i);
printf("%d", i);
```

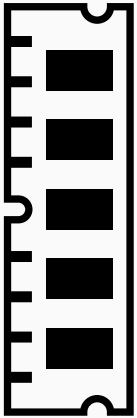


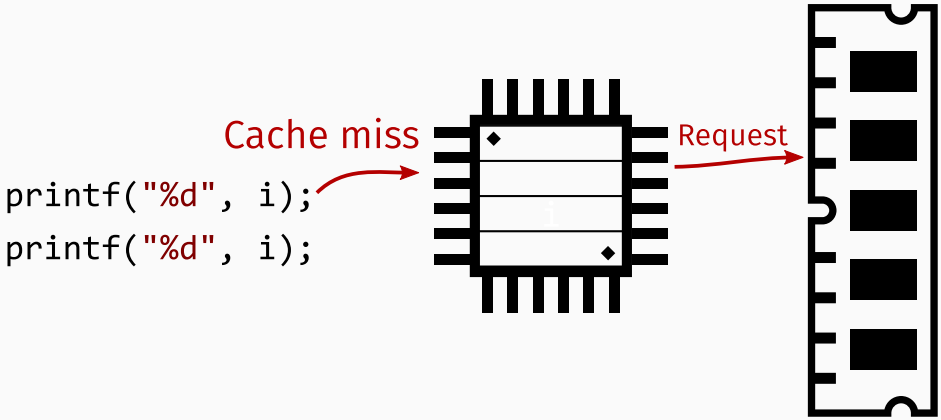


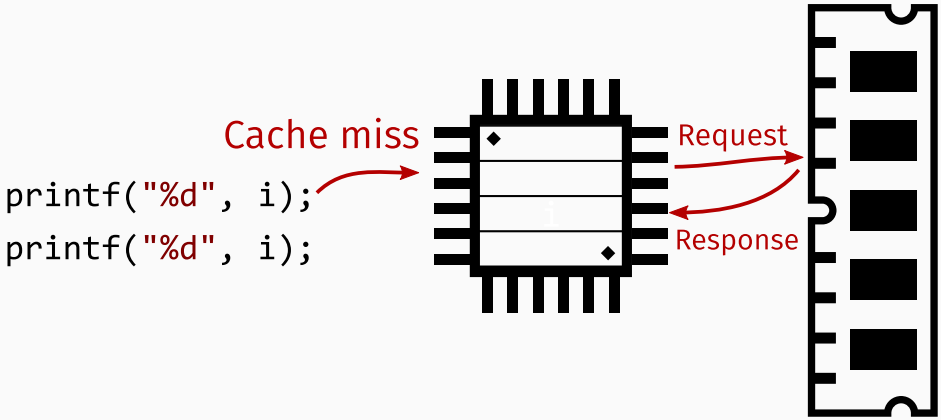
Cache miss

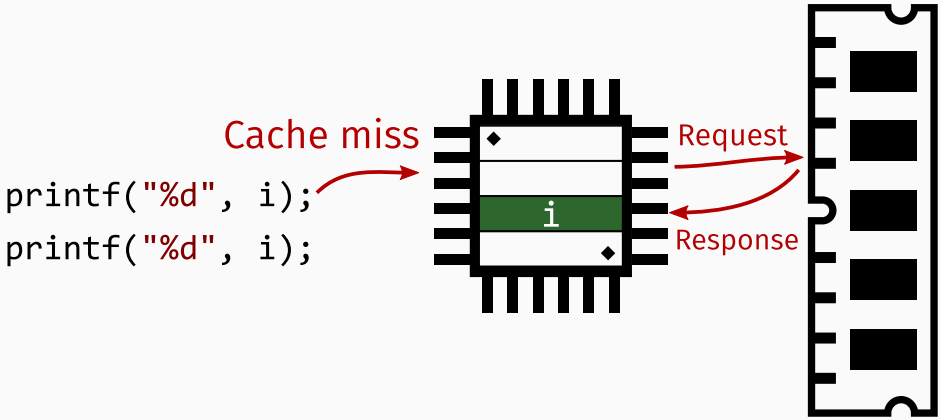
```
printf("%d", i);
printf("%d", i);
```

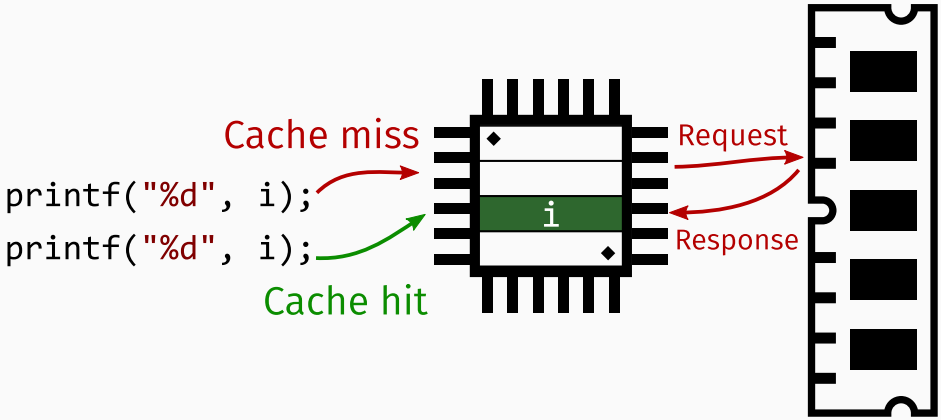


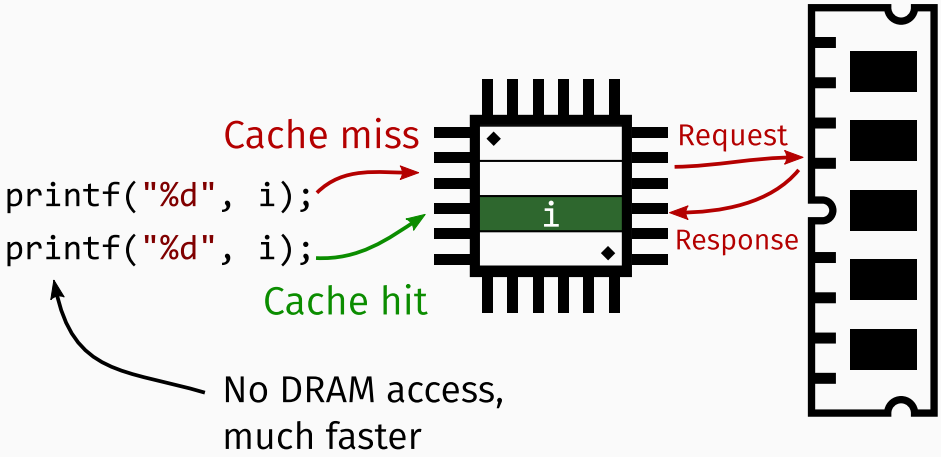


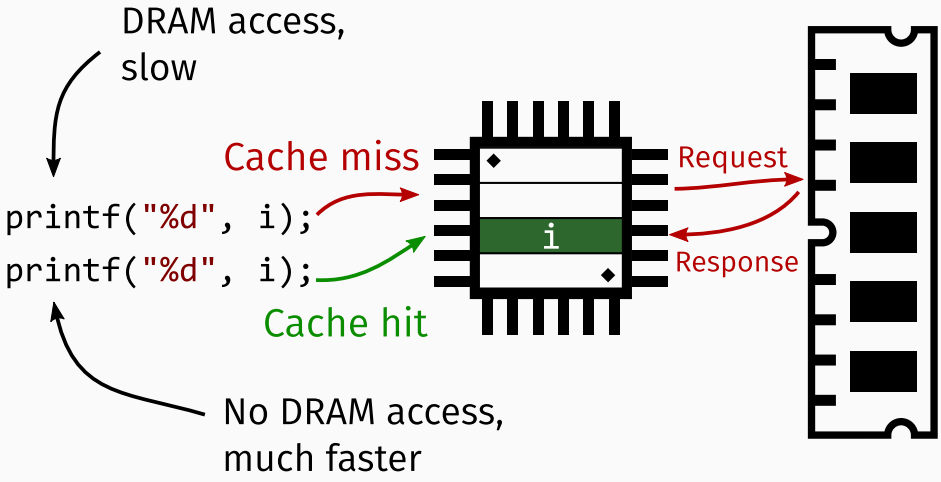

Cache miss
Request
printf("%d", i);
printf("%d", i);


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

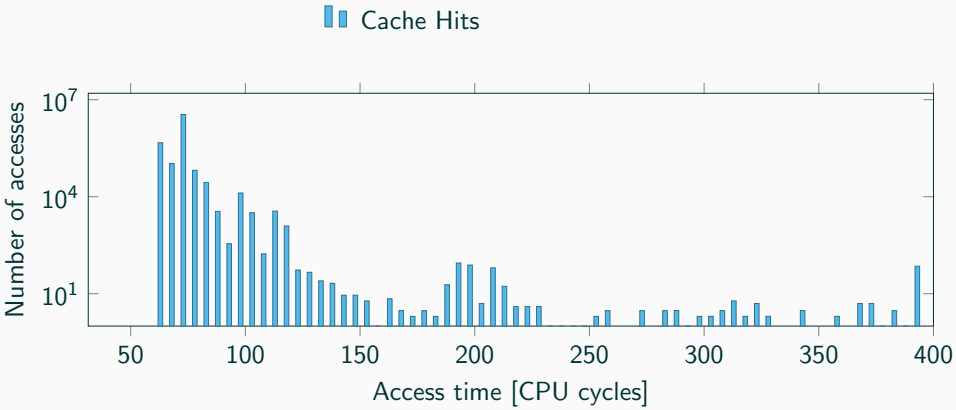

Cache miss
printf("%d", i);
printf("%d", i);
i
Request
Response


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

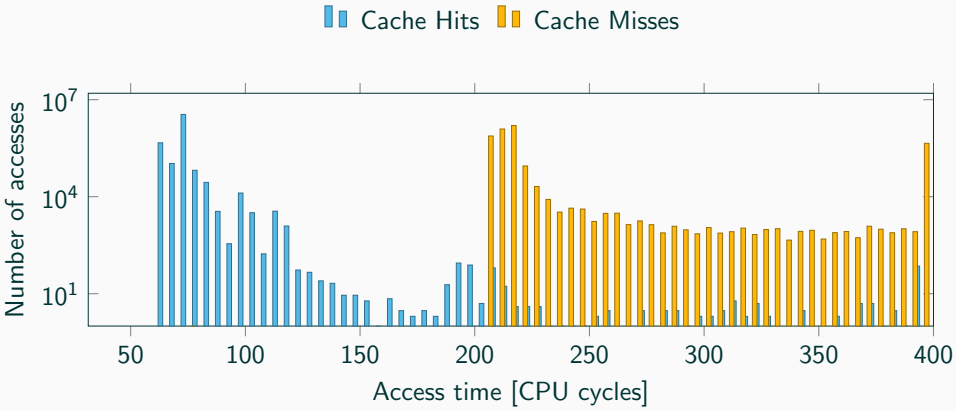

Cache miss
printf("%d", i);
printf("%d", i);
Cache hit
i
Request
Response
No DRAM access,
much faster


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

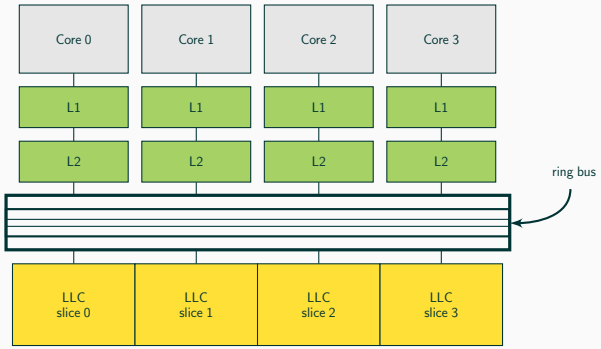
Memory Access Latency



Memory Access Latency

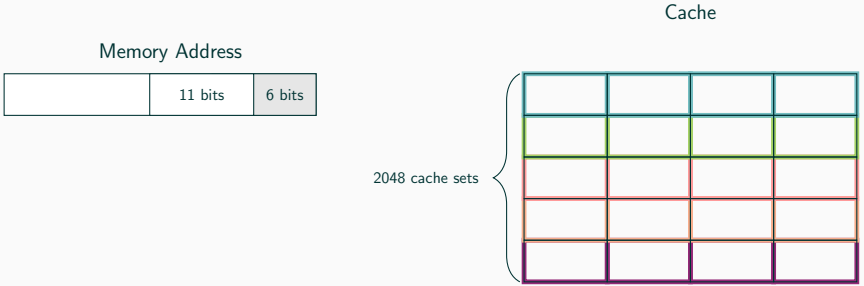


Cache hierarchy



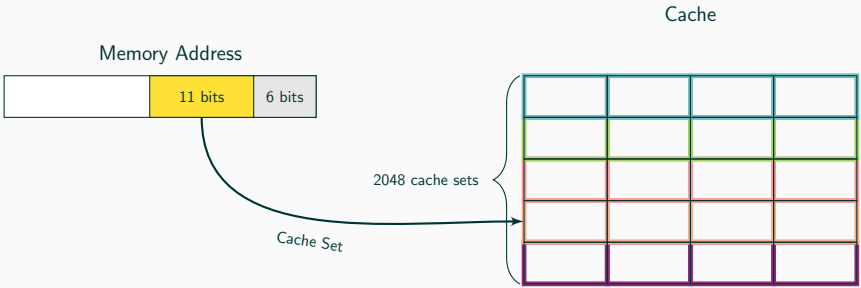
- L1 and L2 are private
- Last-level cache is
  - divided into slices
  - shared across cores
  - inclusive

Set-associative Last-level Cache



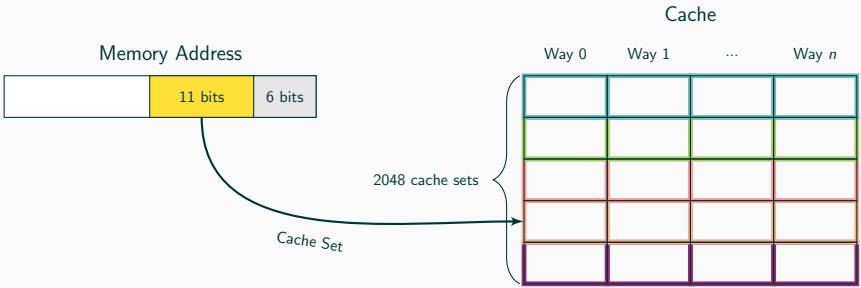
- Location in cache depends on the physical address of data

Set-associative Last-level Cache



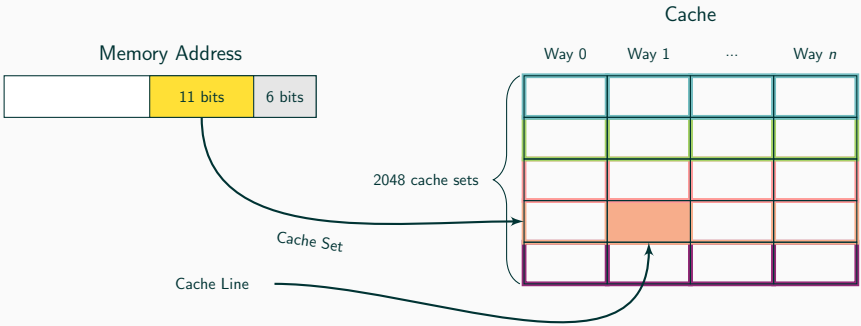
- Location in cache depends on the physical address of data
- Bits 6 to 16 determine the cache set

Set-associative Last-level Cache



- 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

Set-associative Last-level Cache

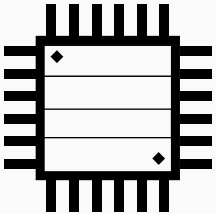


- Location in cache depends on the physical address of data
- Bits 6 to 16 determine the cache set
- A cache set has multiple ways to store the data
- A way inside a cache set is a cache line, determined by the cache replacement policy

Shared Memory

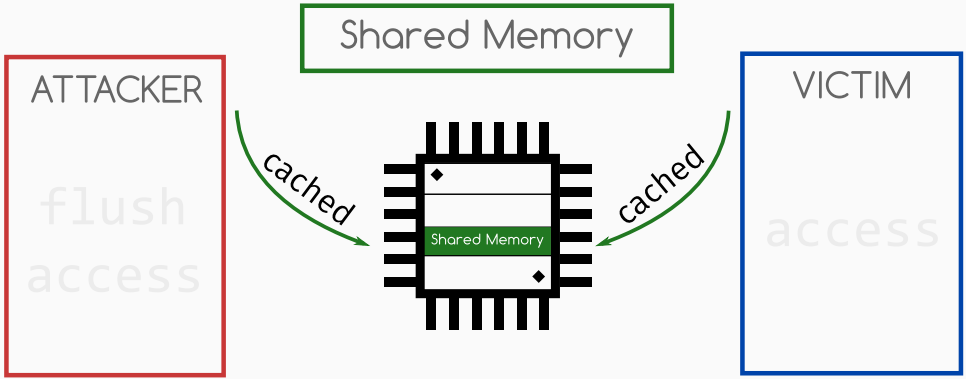
ATTACKER

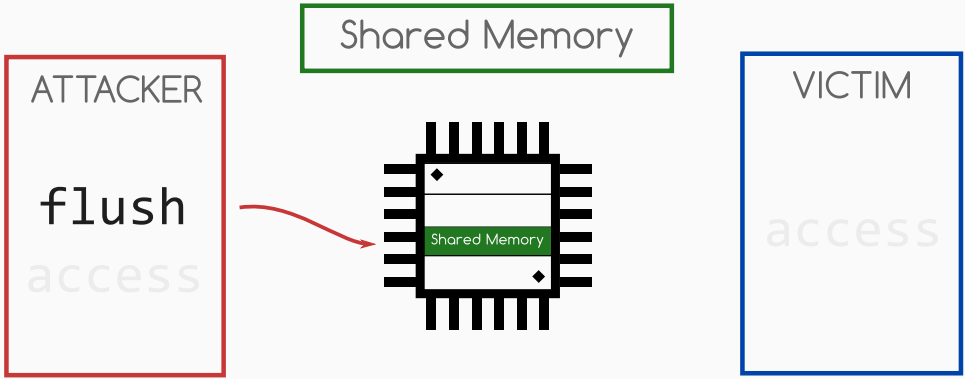
flush
access

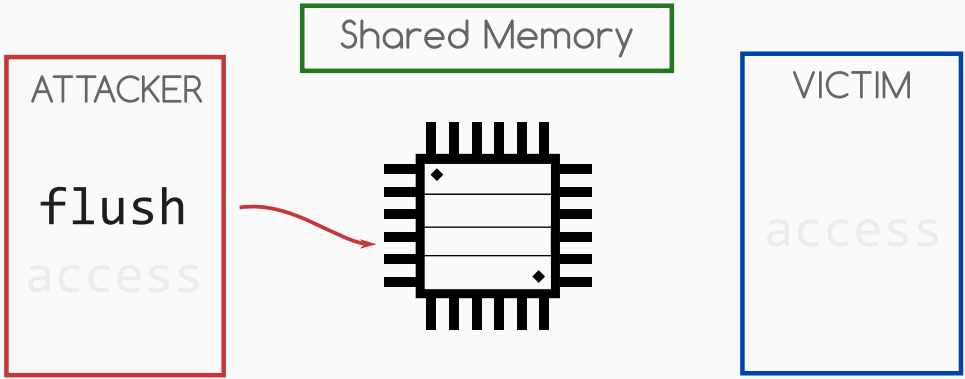


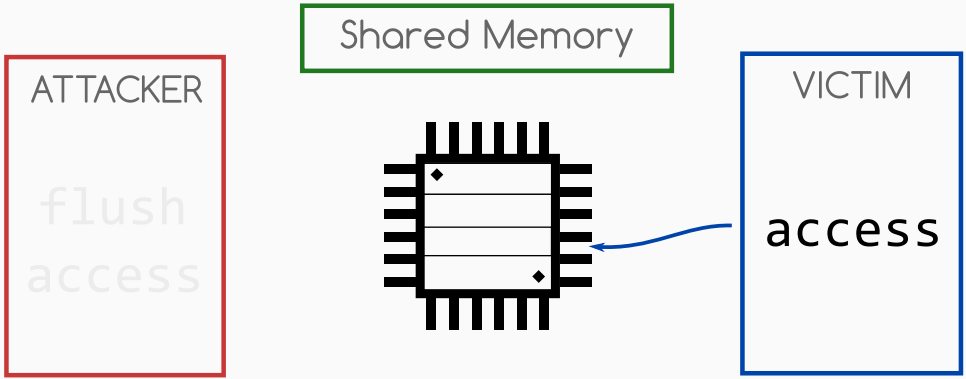
VICTIM

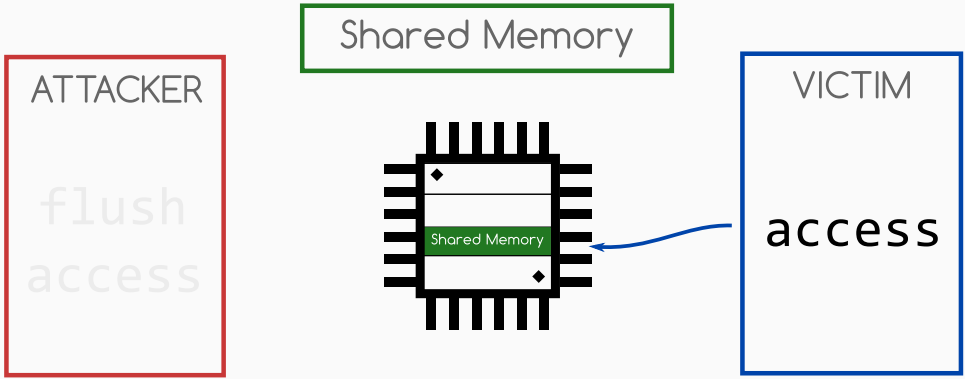
access

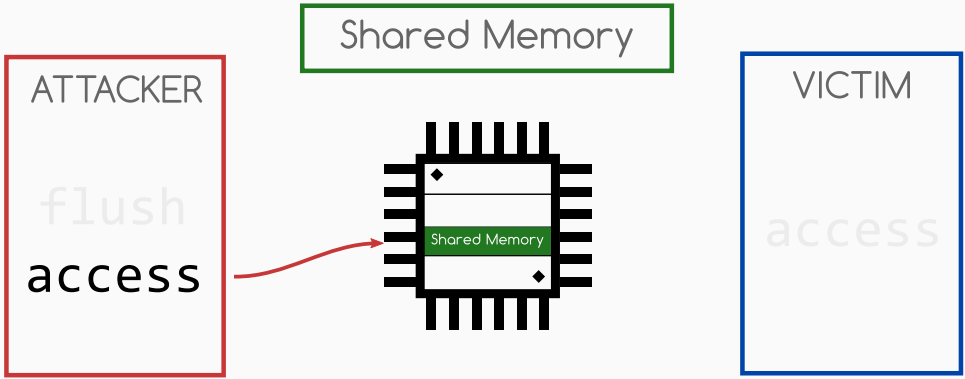

Shared Memory
ATTACKER
flush
access
cached
Shared Memory
cached
VICTIM
access

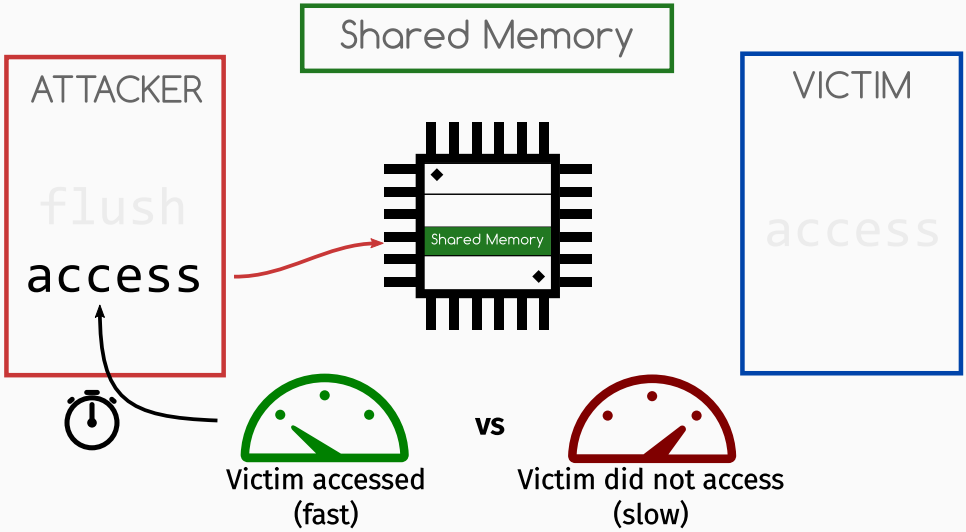

Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access

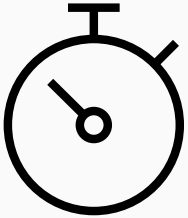

Shared Memory
ATTACKER
flush
access
VICTIM
access


Shared Memory
ATTACKER
flush
access
VICTIM
access

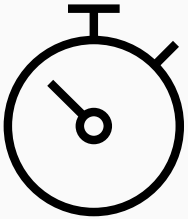

Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access


Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access


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

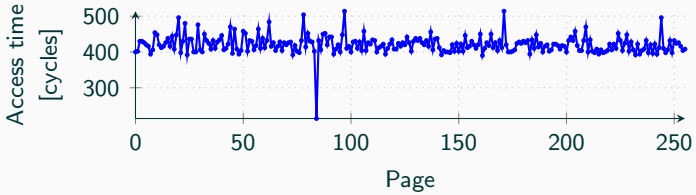


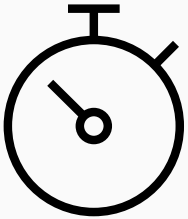
```
struct shared_data[256];
[...]
return shared_data[84];
[...]
```



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

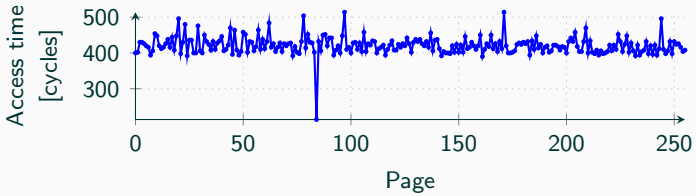
- Flush+Reload over memory locations





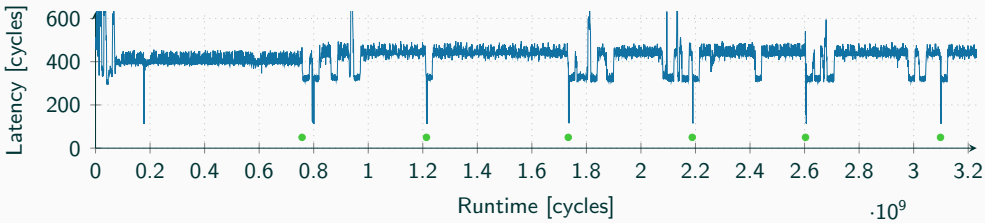
```
struct shared_data[256];
[...]
return shared_data[84];
[...]
```

- Flush+Reload over memory locations



- Accessed index results in faster access time

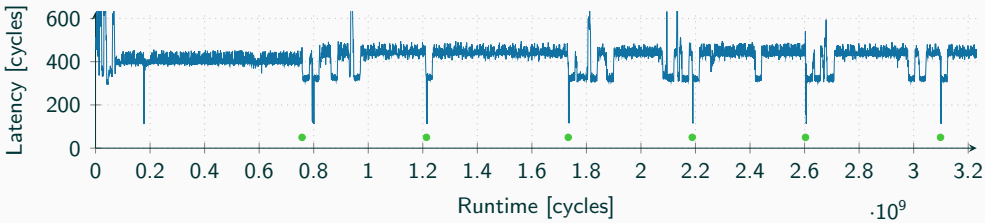
Flush+Reload on Keystrokes



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

KeyDrown: Eliminating Software-Based Keystroke Timing Side-Channel Attacks.
Michael Schwarz, Moritz Lipp, Daniel Gruss, Samuel Weiser, Clémentine Maurice, Raphael Spreitzer, Stefan Mangard. NDSS'18

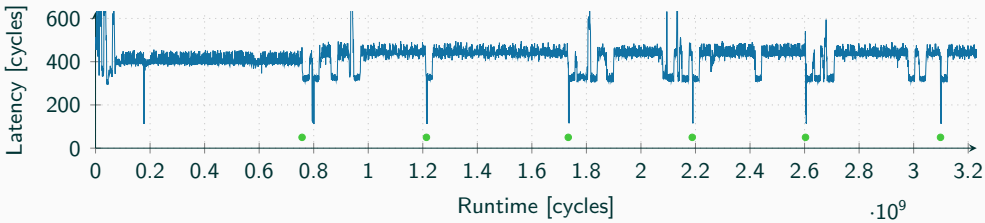
Flush+Reload on Keystrokes



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

KeyDrown: Eliminating Software-Based Keystroke Timing Side-Channel Attacks.
Michael Schwarz, Moritz Lipp, Daniel Gruss, Samuel Weiser, Clémentine Maurice, Raphael Spreitzer, Stefan Mangard. NDSS'18

Flush+Reload on Keystrokes

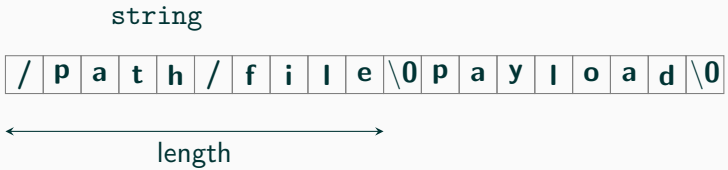


- Key presses trigger code execution in shared library (e.g., `libgdk`)
- Flush+Reload does not reveal actual key, only time difference between keys
- $\rightarrow$ Recover text with machine learning

KeyDrown: Eliminating Software-Based Keystroke Timing Side-Channel Attacks.
Michael Schwarz, Moritz Lipp, Daniel Gruss, Samuel Weiser, Clémentine Maurice, Raphael Spreitzer, Stefan Mangard. NDSS'18

A Double Fetch

A Double Fetch

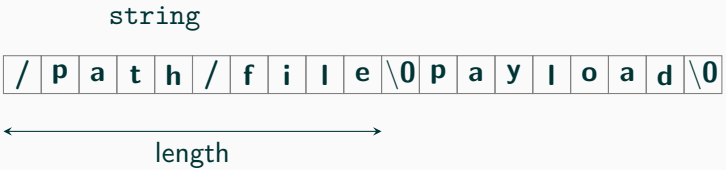


Thread 1

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

Thread 2

A Double Fetch

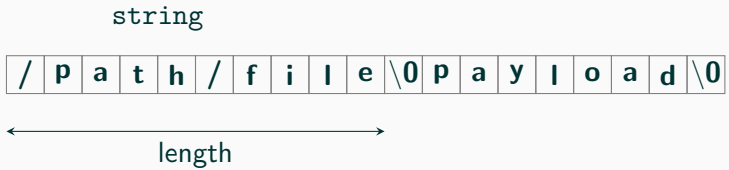


Thread 1

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

// <switch to kernel>
```

Thread 2

A Double Fetch

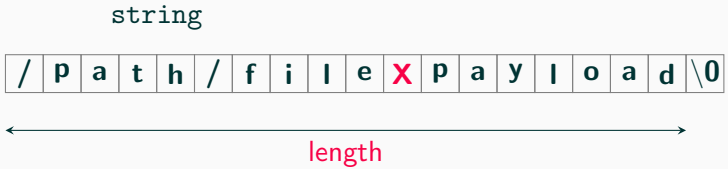


Thread 1

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

// <switch to kernel>

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

Thread 2

A Double Fetch



Thread 1

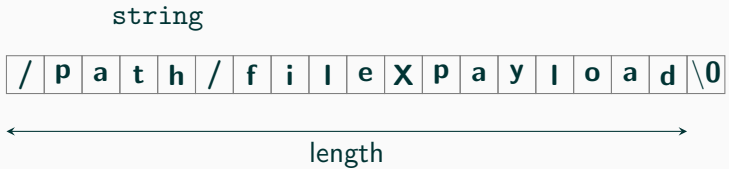
```
strcpy(string, "/path/file\0payload");
open(string, O_CREAT);

// <switch to kernel>

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

schedule

Thread 2

```
string[10] = 'X';
```

A Double Fetch



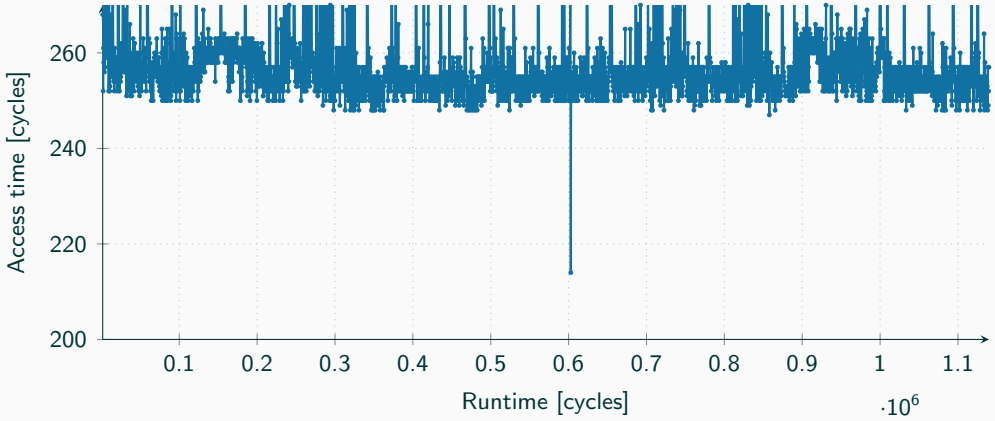
Thread 1

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

// <switch to kernel>

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

strcpy(local, string);
// <memory corruption>
```

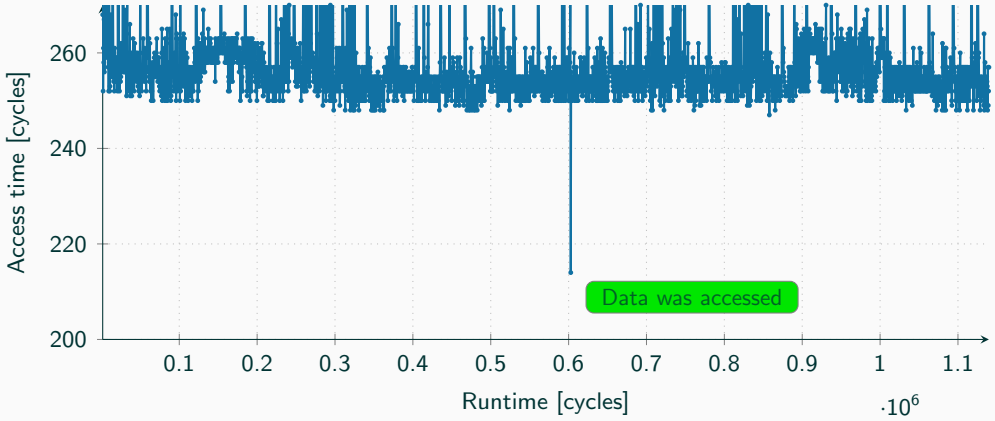
Thread 2

```
string[10] = 'X';
```

schedule

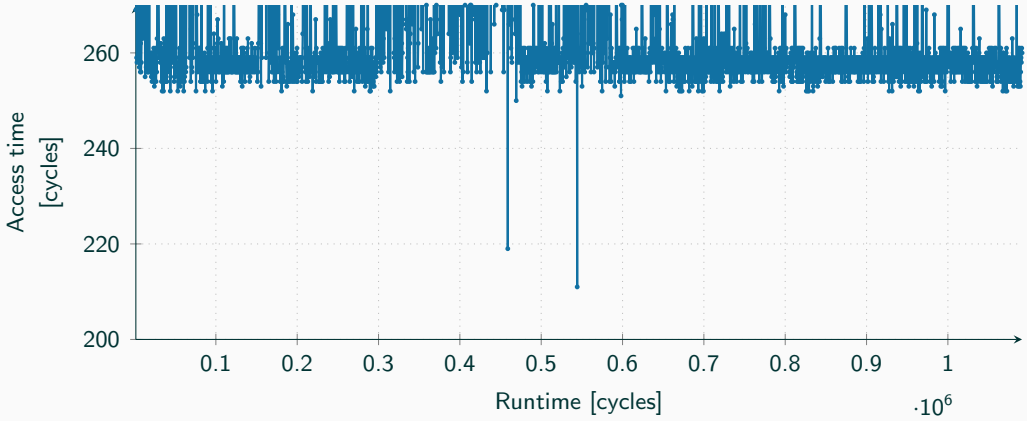


Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features.
Michael Schwarz, Daniel Gruss, Moritz Lipp, Clémentine Maurice, Thomas Schuster, Anders Fogh, Stefan Mangard. AsiaCCS'18



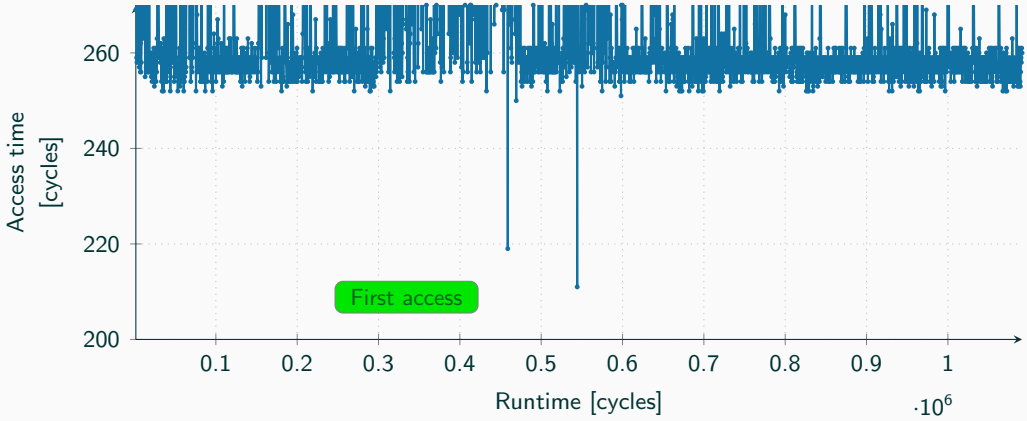
Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features.
Michael Schwarz, Daniel Gruss, Moritz Lipp, Clémentine Maurice, Thomas Schuster, Anders Fogh, Stefan Mangard. AsiaCCS'18

Double-fetch Detection



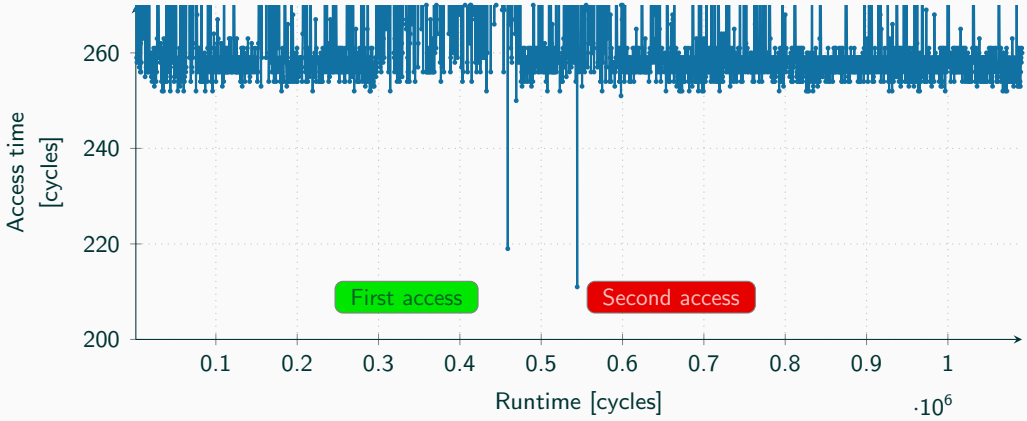
Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features.
Michael Schwarz, Daniel Gruss, Moritz Lipp, Clémentine Maurice, Thomas Schuster, Anders Fogh, Stefan Mangard. AsiaCCS'18

Double-fetch Detection



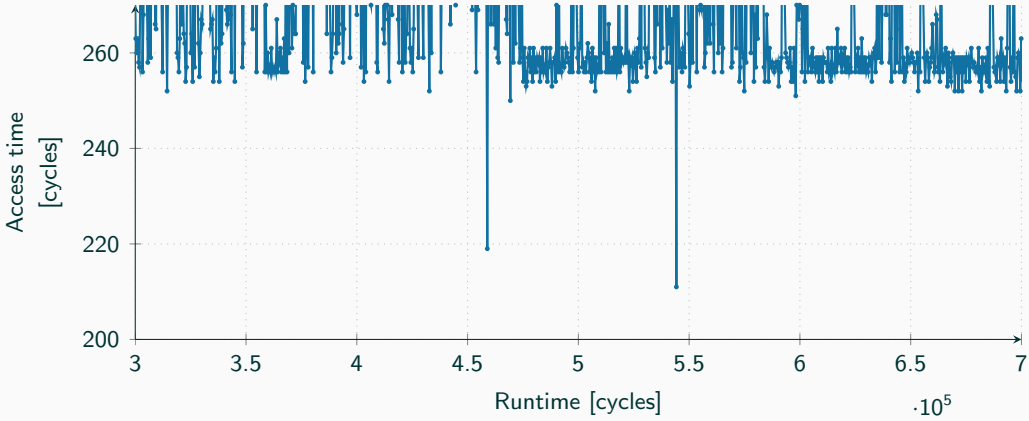
Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features.
Michael Schwarz, Daniel Gruss, Moritz Lipp, Clémentine Maurice, Thomas Schuster, Anders Fogh, Stefan Mangard. AsiaCCS'18

Double-fetch Detection



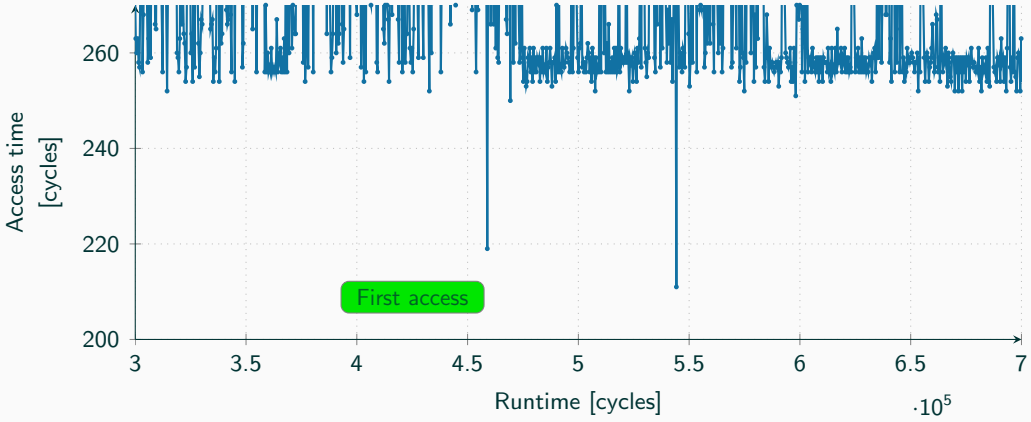
Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features.
Michael Schwarz, Daniel Gruss, Moritz Lipp, Clémentine Maurice, Thomas Schuster, Anders Fogh, Stefan Mangard. AsiaCCS'18

Cache-based Trigger



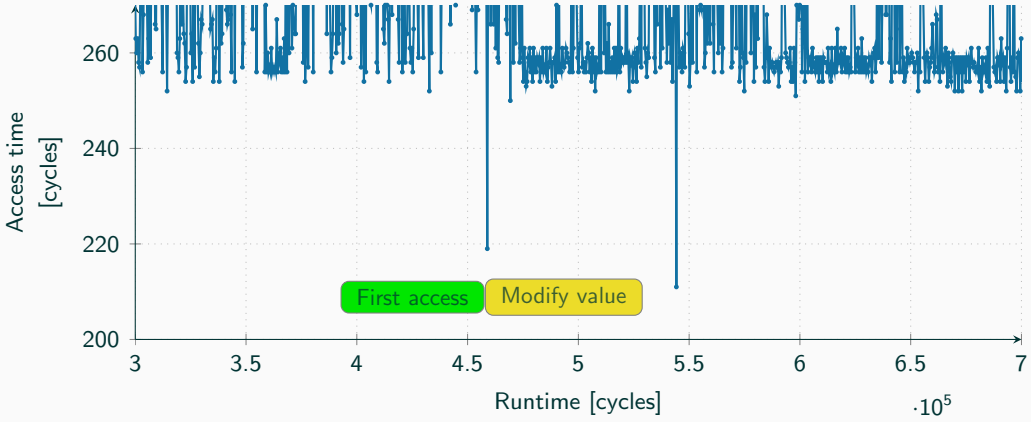
Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features.
Michael Schwarz, Daniel Gruss, Moritz Lipp, Clémentine Maurice, Thomas Schuster, Anders Fogh, Stefan Mangard. AsiaCCS'18

Cache-based Trigger



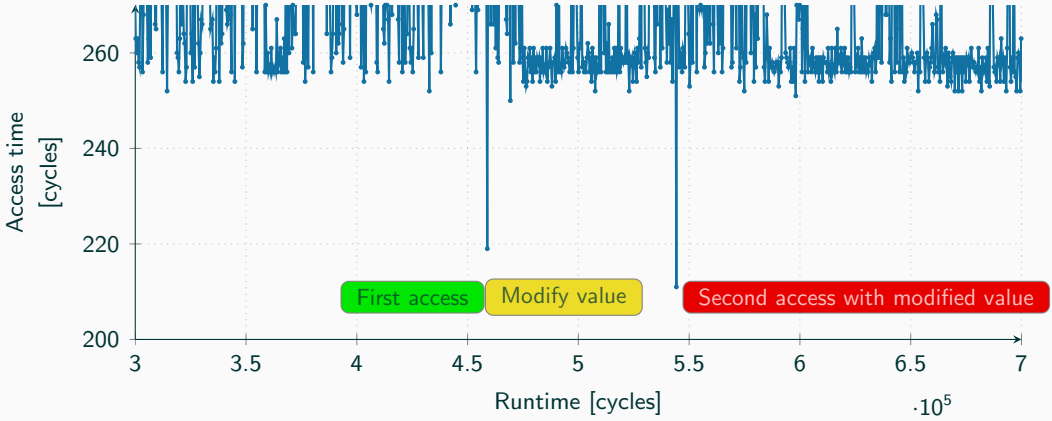
Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features.
Michael Schwarz, Daniel Gruss, Moritz Lipp, Clémentine Maurice, Thomas Schuster, Anders Fogh, Stefan Mangard. AsiaCCS'18

Cache-based Trigger



Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features.
Michael Schwarz, Daniel Gruss, Moritz Lipp, Clémentine Maurice, Thomas Schuster, Anders Fogh, Stefan Mangard. AsiaCCS'18

Cache-based Trigger

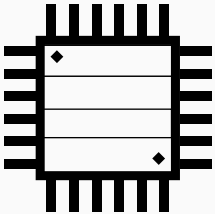


Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features.
Michael Schwarz, Daniel Gruss, Moritz Lipp, Clémentine Maurice, Thomas Schuster, Anders Fogh, Stefan Mangard. AsiaCCS'18

ATTACKER

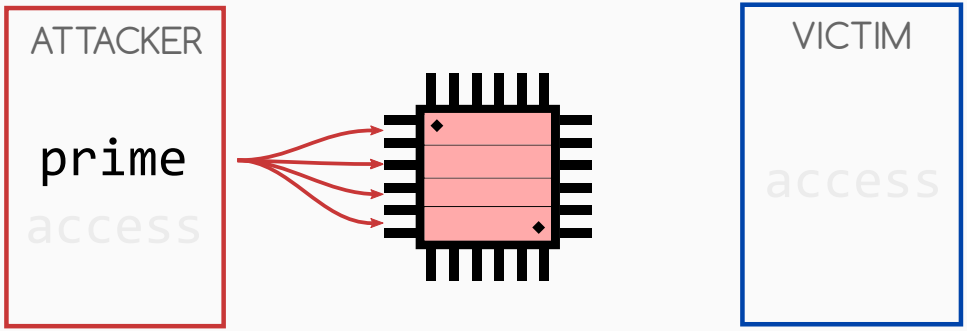
prime

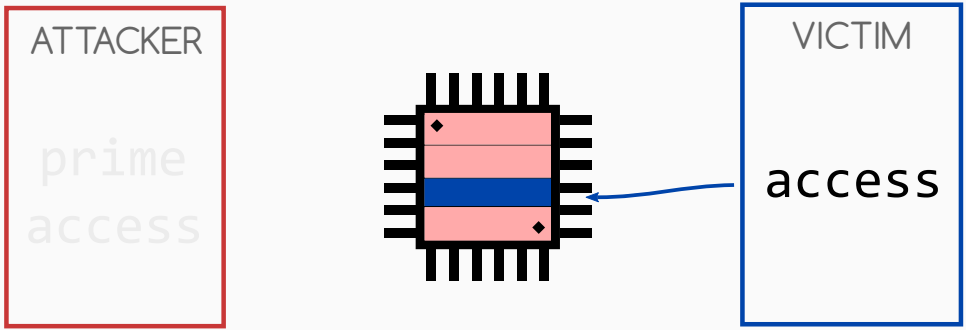
access

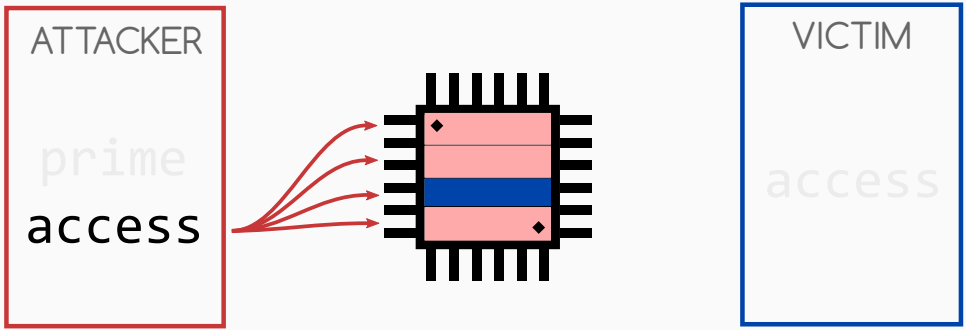


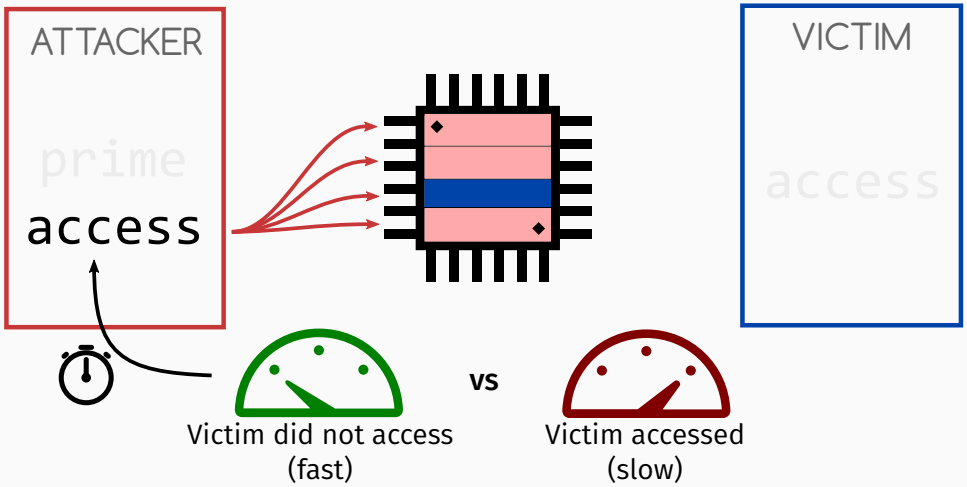
VICTIM

access

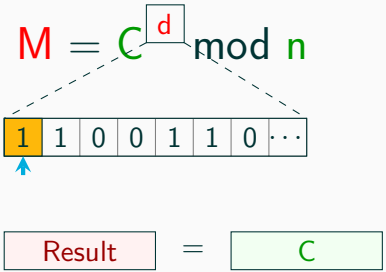

ATTACKER
prime
access
VICTIM
access

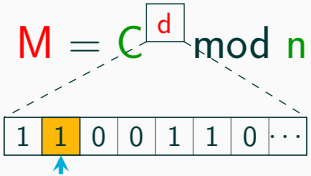

ATTACKER
prime
access
VICTIM
access

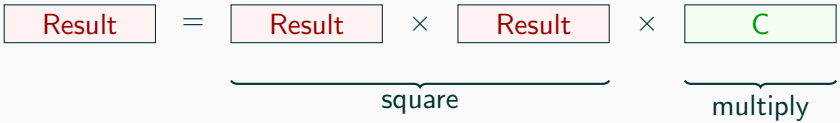

ATTACKER
prime
access
VICTIM
access

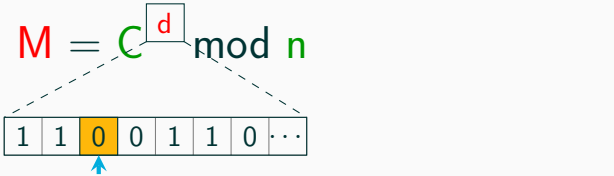

ATTACKER
prime
access
VICTIM
access
vs
Victim did not access
(fast)
Victim accessed
(slow)

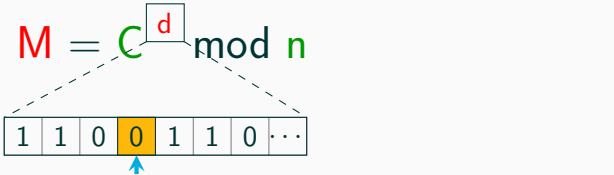
$$M = C^{d} \bmod n$$

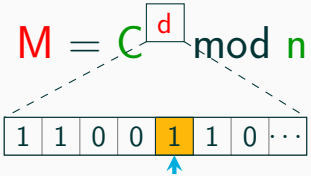

M = C d mod n
1 1 0 0 1 1 0 ···
Result = C

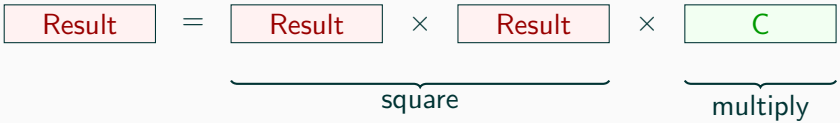

M = C d mod n
1 1 0 0 1 1 0 ···

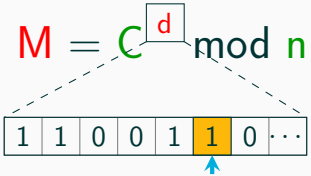

Result = Result × Result × C
square
multiply

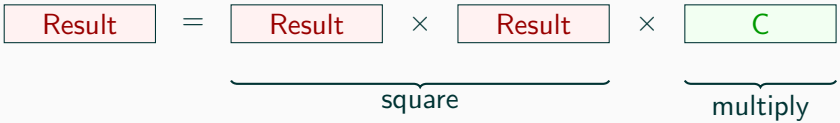

M = C^d mod n
1 1 0 0 1 1 0 ···
Result = Result × Result
square

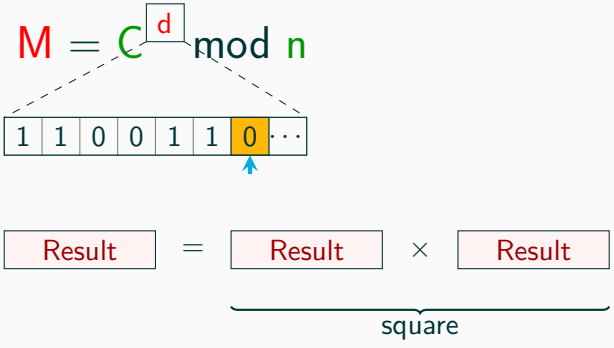

M = C^d mod n
1 1 0 0 1 1 0 ···
Result = Result × Result
square


M = C d mod n
1 1 0 0 1 1 0 ···


Result = Result × Result × C
square
multiply


M = C d mod n
1 1 0 0 1 1 0 ···

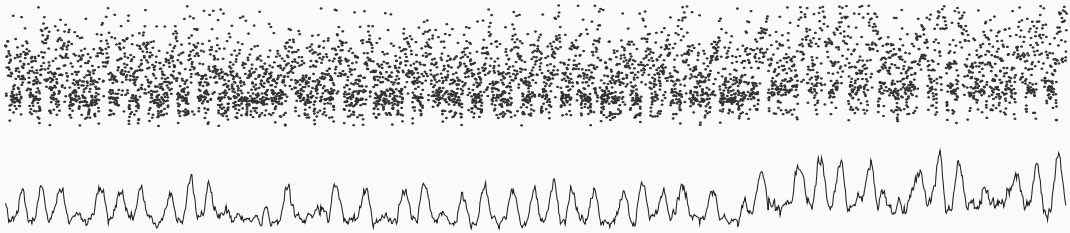

Result = Result × Result × C
square
multiply


M = C^d mod n
1 1 0 0 1 1 0 ···
Result = Result × Result
square

Raw Prime+Probe trace...

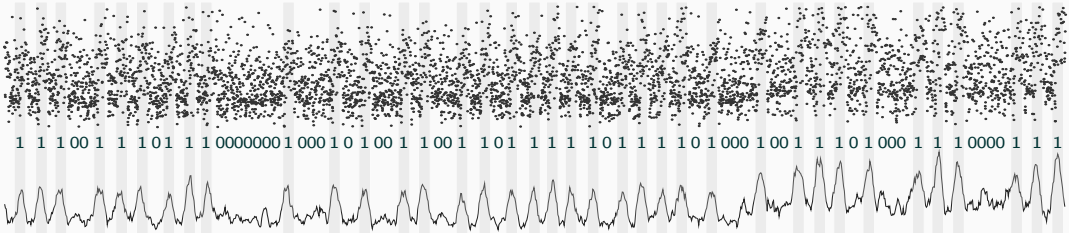
Malware Guard Extension: Using SGX to Conceal Cache Attacks.
Michael Schwarz, Samuel Weiser, Daniel Gruss, Clmentine Maurice, Stefan Mangard. DIMVA'17

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



Malware Guard Extension: Using SGX to Conceal Cache Attacks.
Michael Schwarz, Samuel Weiser, Daniel Gruss, Clmentine Maurice, Stefan Mangard. DIMVA'17

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



Malware Guard Extension: Using SGX to Conceal Cache Attacks.
Michael Schwarz, Samuel Weiser, Daniel Gruss, Clmentine Maurice, Stefan Mangard. DIMVA'17

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...

Covert channel

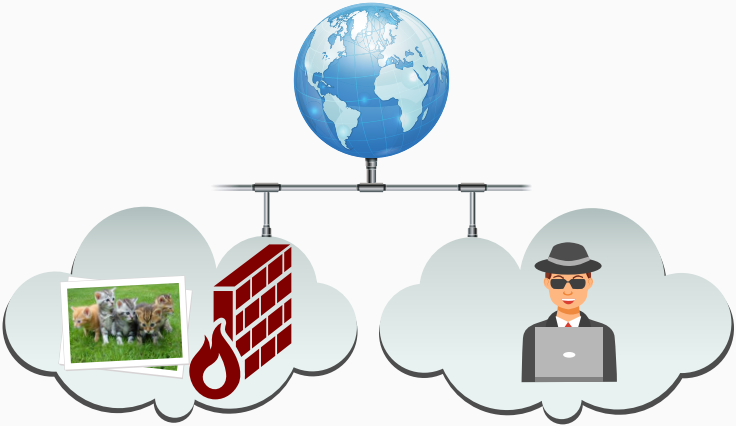
What is a covert channel?

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

What is a covert channel?

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

Covert channel



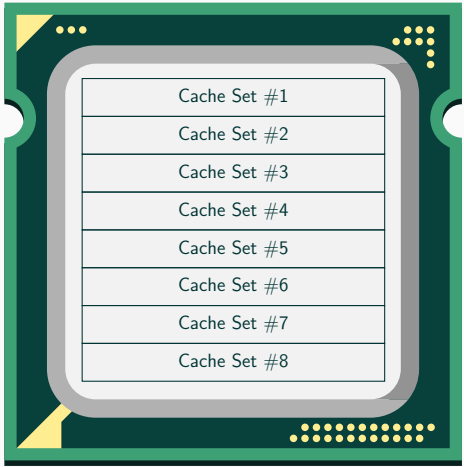
Covert channel

Sending Data

Sender

Last-level cache

Receiver

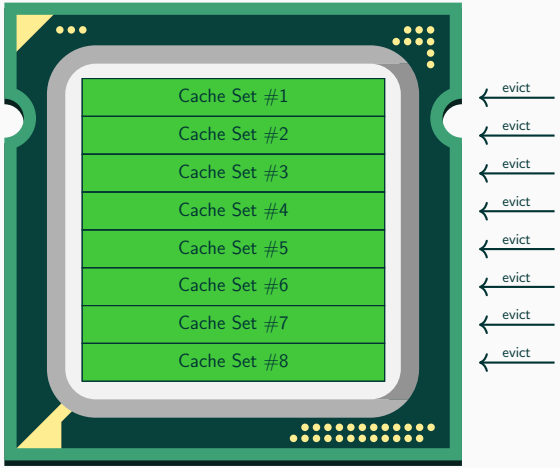


Sending Data

Sender

Last-level cache

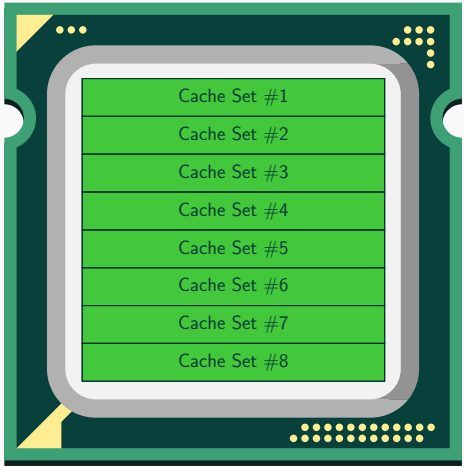
Receiver



Sending Data

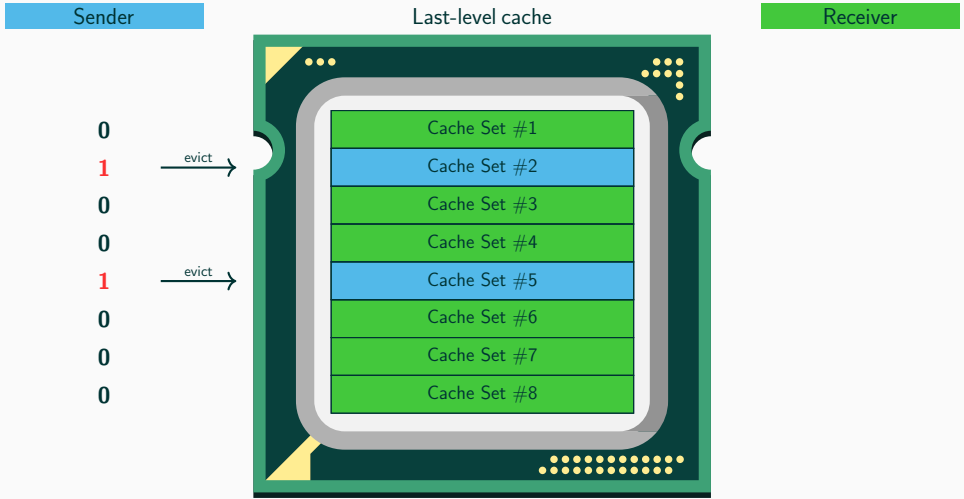
Sender

0
1
0
0
1
0
0
0

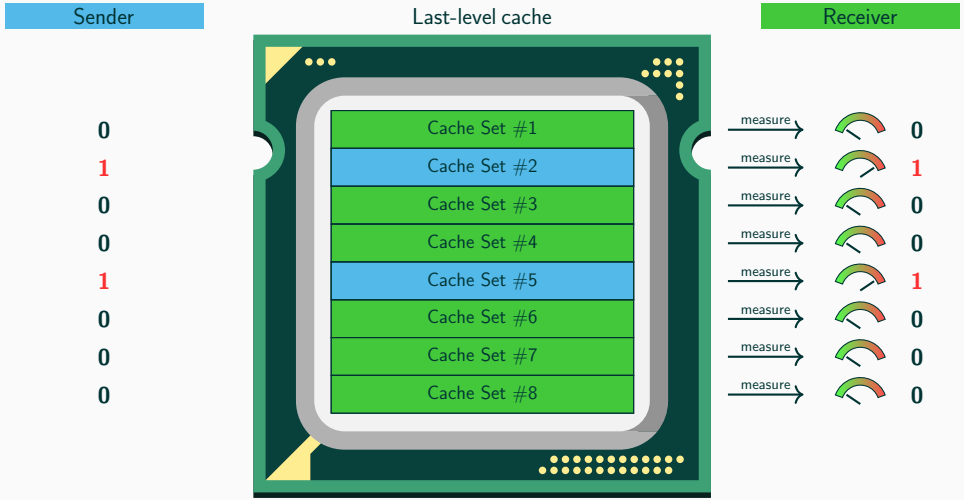


Receiver

Sending Data



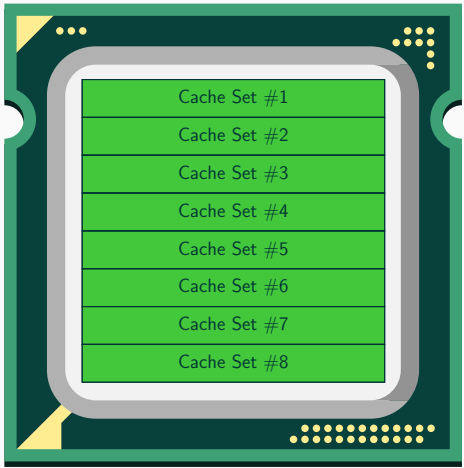
Sending Data



Sending Data

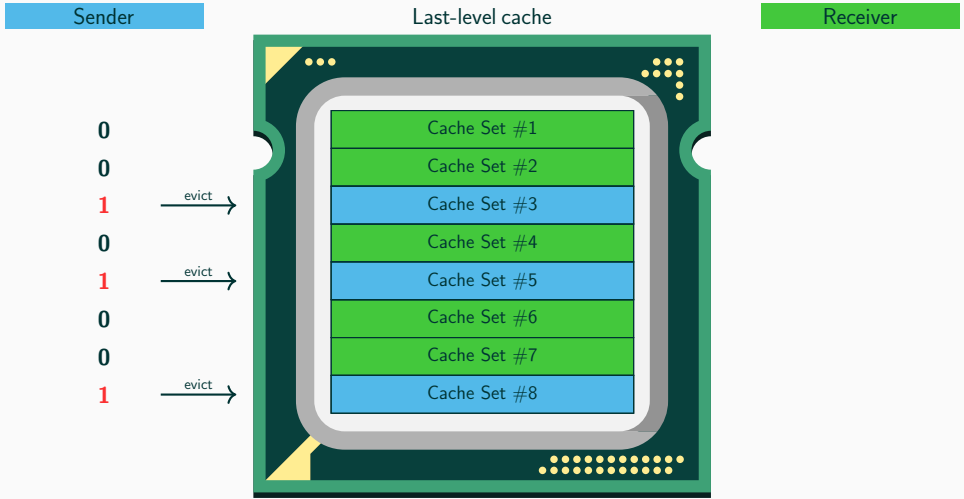
Sender

0
0
1
0
1
0
0
1

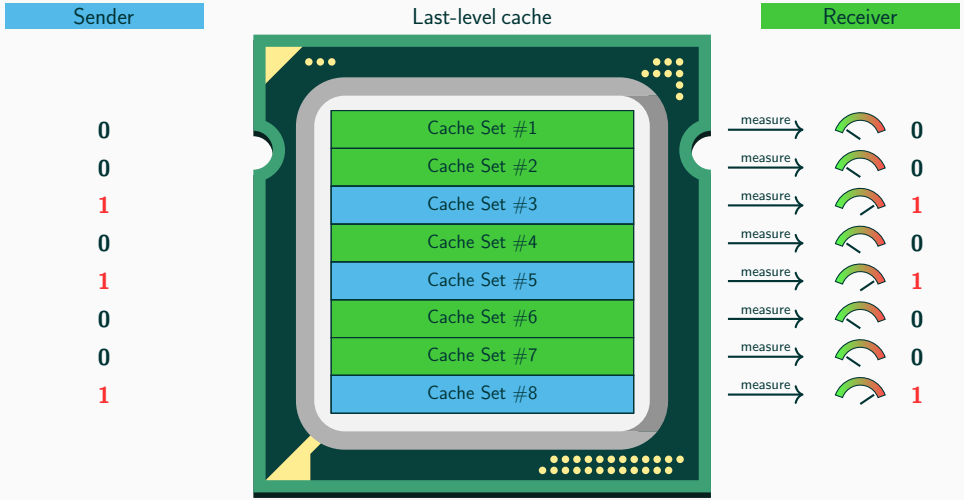


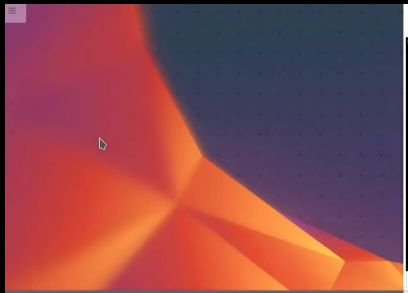
Receiver

Sending Data



Sending Data





local — Konsole

File Edit View Bookmarks Settings Help

michael@michael-hp ~ %

local

sender (ec2) — Konsole

File Edit View Bookmarks Settings Help

ubuntu@ip-172-31-51-32 ~ %

sender (ec2)

receiver (ec2) — Konsole

File Edit View Bookmarks Settings Help

ubuntu@ip-172-31-58-32 ~ %

receiver (ec2)

sender (ec2) — Konsole <2>

File Edit View Bookmarks Settings Help

-= slurm 0.4.3 on ip-172-31-51-32 =-

```
                x
                x
               xx
              xxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                xx
               xx
               x
               x
```

Active Interface: eth0 Interface Speed: unknown

Current RX Speed: 0.05 KB/s	Current TX Speed: 0.33 KB/s
Graph Top RX Speed: 0.98 KB/s	Graph Top TX Speed: 2.64 KB/s
Overall Top RX Speed: 0.98 KB/s	Overall Top TX Speed: 2.64 KB/s
Received Packets: 84	Transmitted Packets: 82
MBytes Received: 0.005 MB	MBytes Transmitted: 0.019 MB
Errors on Receiving: 0	Errors on Transmission: 0

sender (ec2)

receiver (ec2) — Konsole <2>

File Edit View Bookmarks Settings Help

-= slurm 0.4.3 on ip-172-31-58-32 =-

```
                x
                x
                x
                x
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
                xx
                x
                x
                x
```

Active Interface: eth0 Interface Speed: unknown

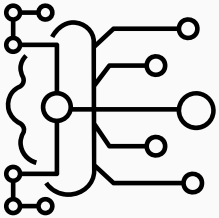
Current RX Speed: 0.05 KB/s	Current TX Speed: 0.29 KB/s
Graph Top RX Speed: 0.36 KB/s	Graph Top TX Speed: 0.84 KB/s
Overall Top RX Speed: 0.36 KB/s	Overall Top TX Speed: 0.84 KB/s
Received Packets: 38	Transmitted Packets: 36
MBytes Received: 0.002 MB	MBytes Transmitted: 0.010 MB
Errors on Receiving: 0	Errors on Transmission: 0

receiver (ec2)

HELLO FROM THE OTHER SIDE (DEMO): VIDEO STREAMING OVER CACHE COVERT CHANNEL

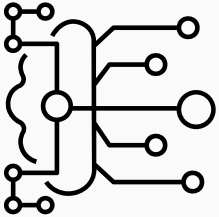
Line 17, Column 1 INSERT Soft Tabs: 4 (8) UTF-8 Normal

Other Microarchitectural Elements



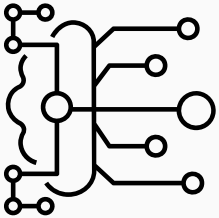
- Multiple other elements with timing differences

Other Microarchitectural Elements

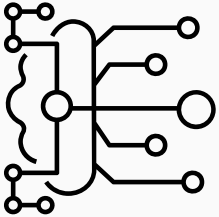


- Multiple other elements with timing differences
  - TLB

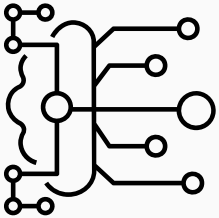
Other Microarchitectural Elements



- Multiple other elements with timing differences
  - TLB
  - DRAM

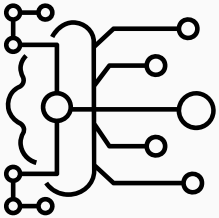


- Multiple other elements with timing differences
  - TLB
  - DRAM
  - Memory Bus



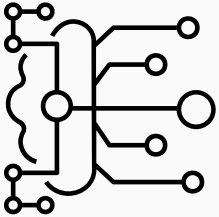
- Multiple other elements with timing differences
  - TLB
  - DRAM
  - Memory Bus
  - Execution Units

Other Microarchitectural Elements



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

Other Microarchitectural Elements



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

Meta Data

- So far, only memory accesses

Meta Data

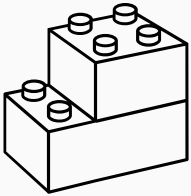
- So far, only memory accesses
- Meta data, no actual data

Meta Data

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

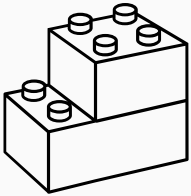
Meta Data

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



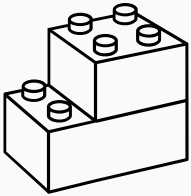
- Side channels can be part of an attack

Building Block



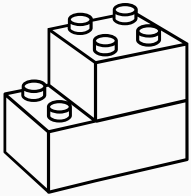
- Side channels can be part of an attack
- Also for conventional memory corruption attacks

Building Block



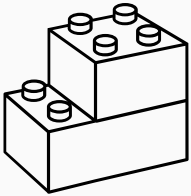
- Side channels can be part of an attack
- Also for conventional memory corruption attacks
- Side channels as building blocks

Building Block



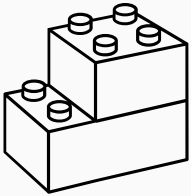
- Side channels can be part of an attack
- Also for conventional memory corruption attacks
- Side channels as building blocks
  - Required information (e.g., break ASLR)

Building Block



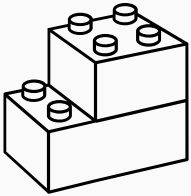
- Side channels can be part of an attack
- Also for conventional memory corruption attacks
- Side channels as building blocks
  - Required information (e.g., break ASLR)
  - Additional information (e.g., length of password)

Building Block



- 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

Building Block



- 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

- Meltdown is a CPU vulnerabilities

Meltdown: Reading Kernel Memory from User Space.
Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg. USENIX Security'18

Meltdown

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

Meltdown: Reading Kernel Memory from User Space.
Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg. USENIX Security'18

Meltdown

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

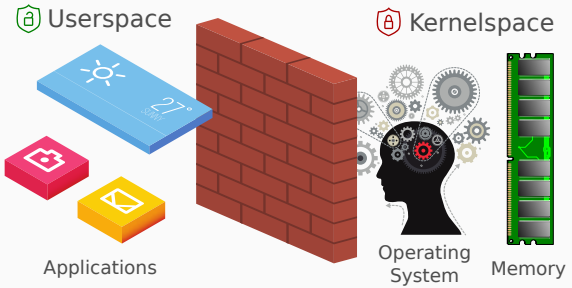
Meltdown: Reading Kernel Memory from User Space.
Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg. USENIX Security'18

Meltdown

- 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

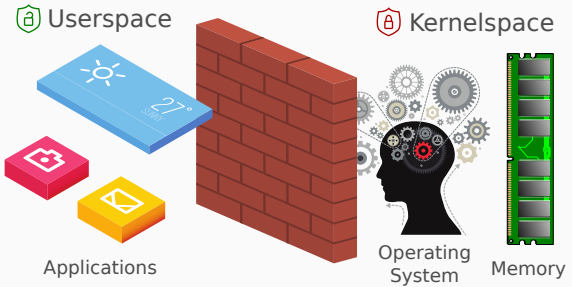
Meltdown: Reading Kernel Memory from User Space.
Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg. USENIX Security'18

- Kernel is isolated from user space

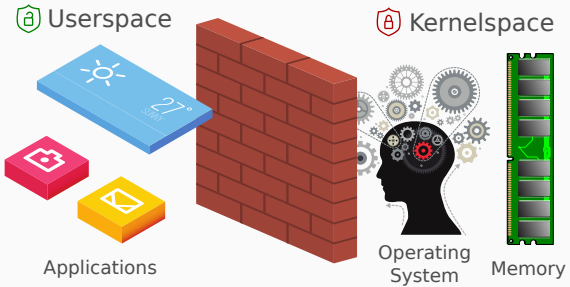


Hardware Isolation

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

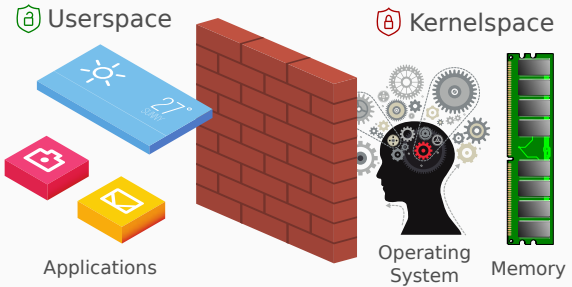


- Kernel is isolated from user space
- This isolation is a combination of hardware and software
- User applications cannot access anything from the kernel

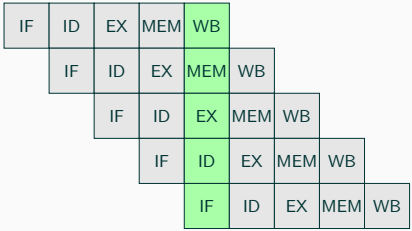


Hardware Isolation

- 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

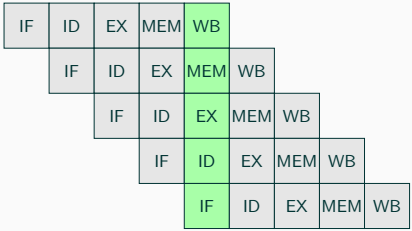


In-Order Execution



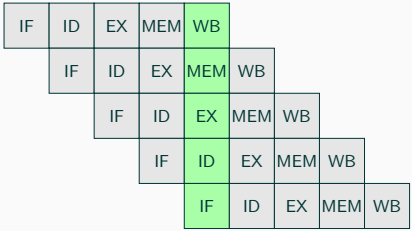
- Instructions are...
  - fetched (`IF`) from the L1 Instruction Cache

In-Order Execution



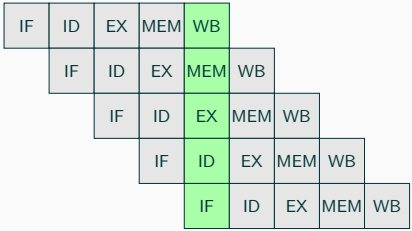
- Instructions are...
  - fetched (`IF`) from the L1 Instruction Cache
  - decoded (`ID`)

In-Order Execution



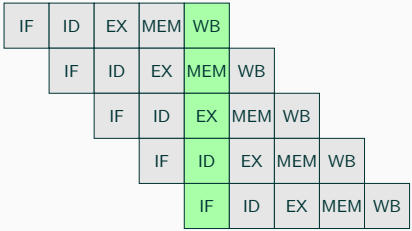
- Instructions are...
  - fetched (`IF`) from the L1 Instruction Cache
  - decoded (`ID`)
  - executed (`EX`) by execution units

In-Order Execution



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

In-Order Execution



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

In-Order Execution

- Instructions are executed in-order

In-Order Execution

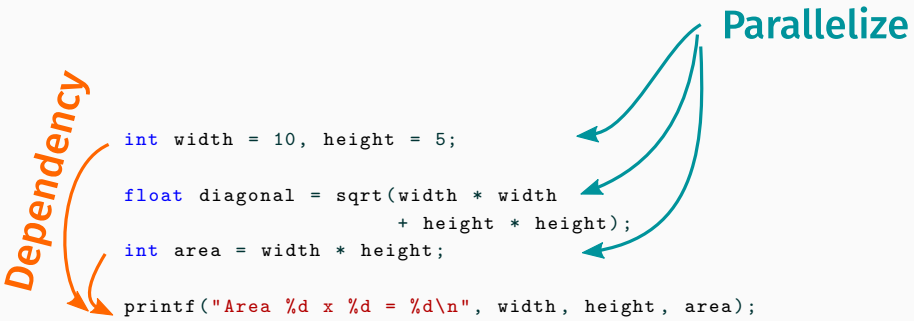
- Instructions are executed in-order
- Pipeline stalls when stages are not ready

In-Order Execution

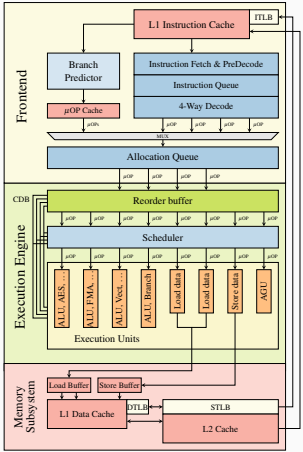
- Instructions are executed in-order
- Pipeline stalls when stages are not ready
- If data is not cached, we need to wait

Out-of-order Execution

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

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

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


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);

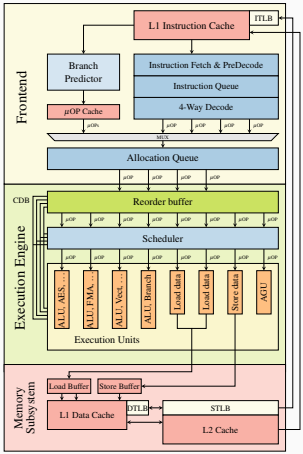
Out-of-Order Execution



Instructions are

- fetched and decoded in the front-end

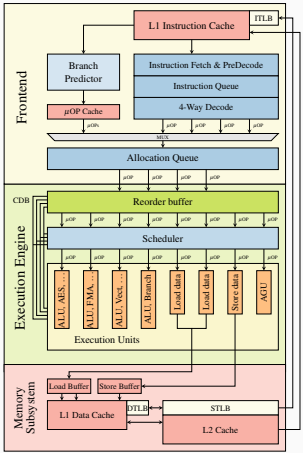
Out-of-Order Execution



Instructions are

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

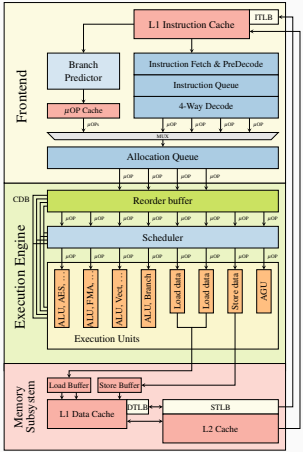
Out-of-Order Execution



Instructions are

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

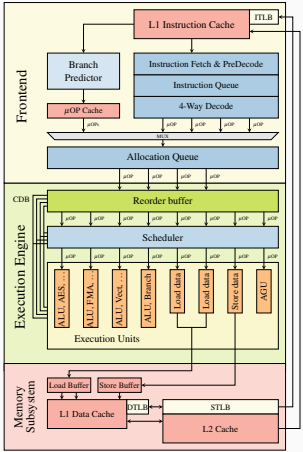
Out-of-Order Execution



Instructions

- are executed out-of-order

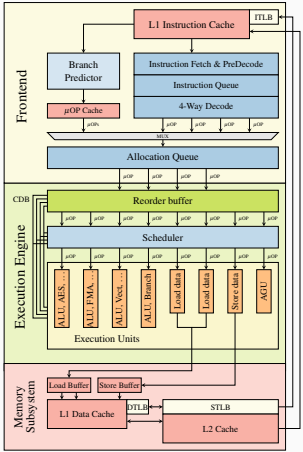
Out-of-Order Execution



Instructions

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

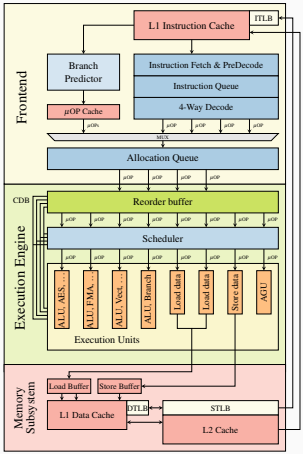
Out-of-Order Execution



Instructions

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

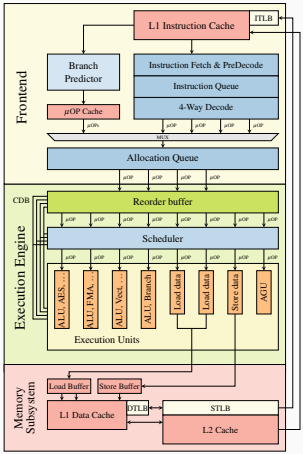
Out-of-Order Execution



Instructions

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

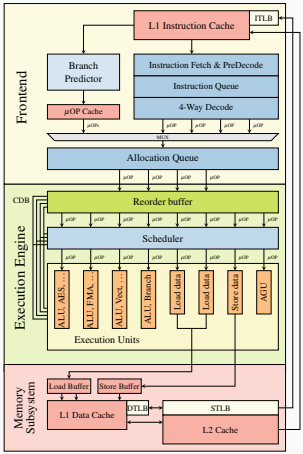
Out-of-Order Execution



Instructions

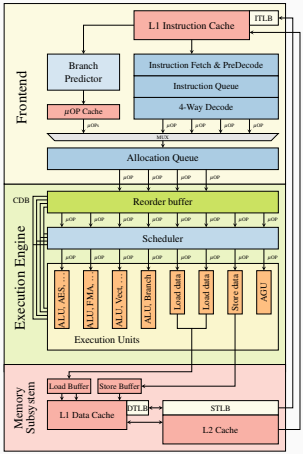
- are executed out-of-order
- wait until their dependencies are ready
  - Later instructions might execute prior earlier instructions
- retire in-order
  - State becomes architecturally visible

Out-of-Order Execution



Instructions

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



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 . . .

Building the Code

- 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

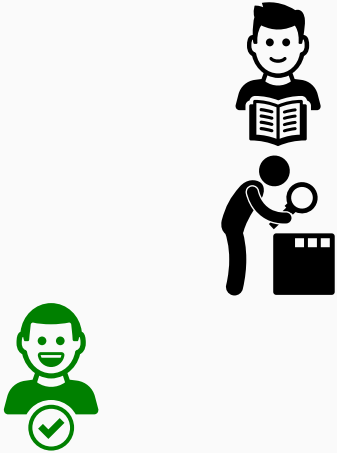
- Out-of-order instructions leave microarchitectural traces
  - We can see them for example in the cache
- Give such instructions a name: transient instructions

- 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

Loading an address

Loading an address

Loading an address



Loading an address

Loading an address

Loading an address

- Add another layer of indirection to test

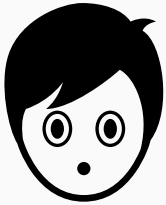
```
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```

Building the Code

- Add another layer of indirection to test

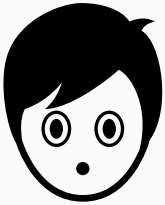
```
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```

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



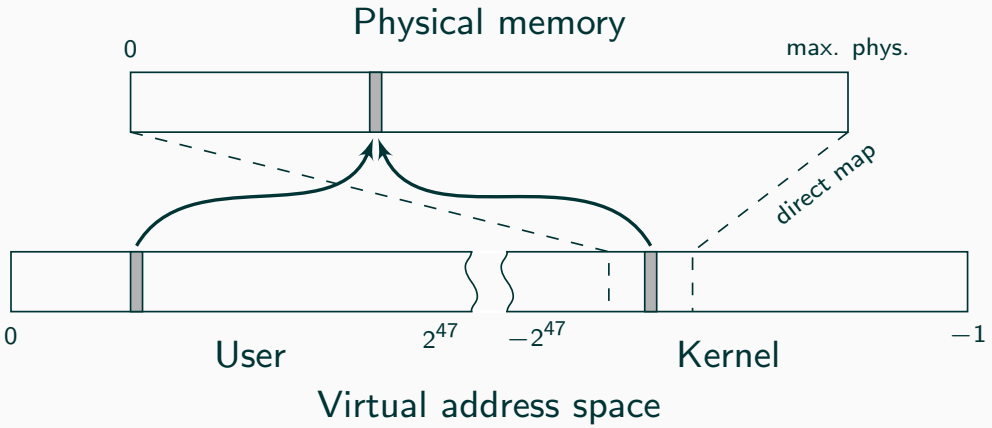
- Flush+Reload over all pages of the array

- 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


Physical memory
0
max. phys.
direct map
0
User
2^{47}
-2^{47}
Kernel
-1
Virtual address space

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

Meltdown: Reading Kernel Memory from User Space.
Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg. USENIX Security'18

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

Meltdown: Reading Kernel Memory from User Space.
Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg. USENIX Security'18

- 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

Meltdown: Reading Kernel Memory from User Space.
Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg. USENIX Security'18

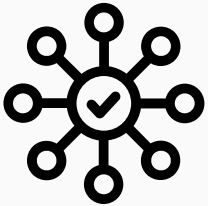
- Using out-of-order execution, we can read data at any address
- Index of cache hit reveals data
- Permission check is in some cases not fast enough
- Entire physical memory is typically accessible through kernel space

Meltdown: Reading Kernel Memory from User Space.
Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg. USENIX Security'18


I SHIT YOU NOT
THERE WAS KERNEL MEMORY ALL OVER THE TERMINAL

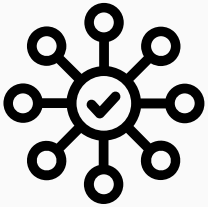

There are no bugs,
just happy little accidents

Generalization

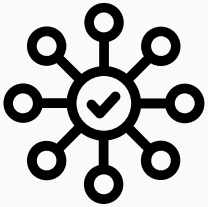


- Meltdown is a whole category of vulnerabilities

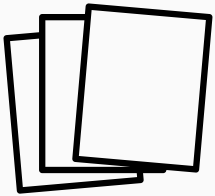
Generalization



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

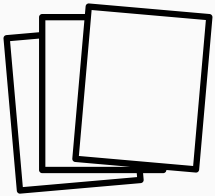


- Meltdown is a whole category of vulnerabilities
- Not only the user-accessible check
- Looking closer at the check...

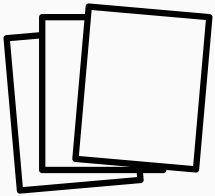


- CPU uses virtual address spaces to isolate processes

Paging

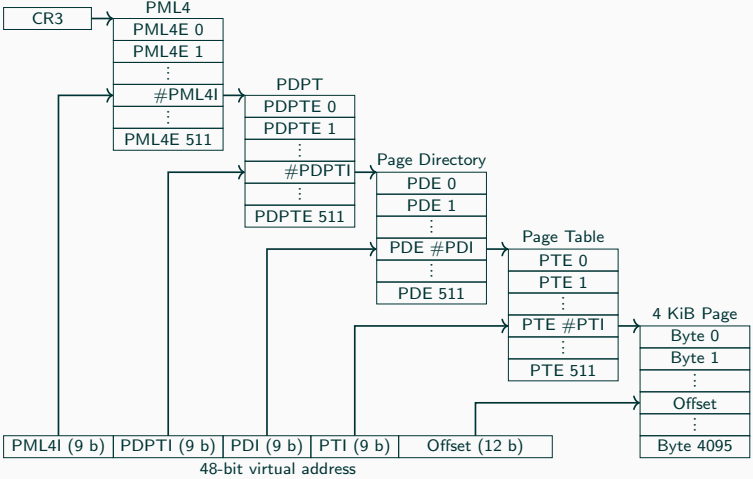


- CPU uses virtual address spaces to isolate processes
- Physical memory is organized in page frames



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

Address Translation on x86-64



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


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

- Present bit is the next obvious bit

- An even worse bug → Foreshadow-NG/L1TF

Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution.
Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. USENIX Security'18

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

Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution.
Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. USENIX Security'18

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

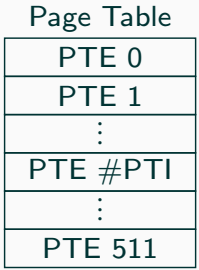
Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution.
Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. USENIX Security'18

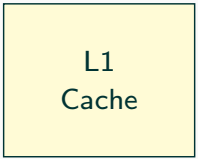
- An even worse bug → Foreshadow-NG/L1TF
- Exploitable from VMs
- Allows leaking data from the L1 cache
- Same mechanism as Meltdown

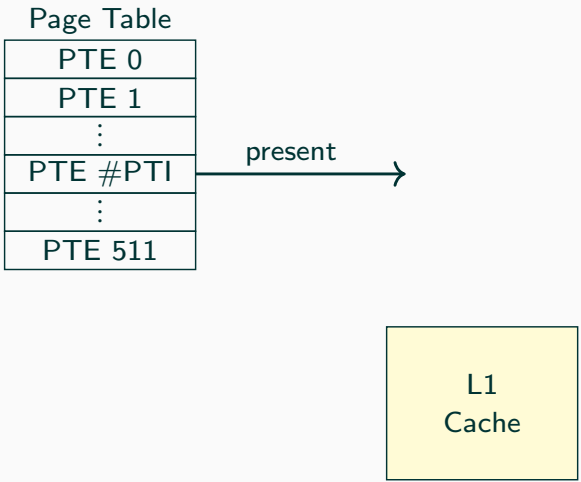
Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution.
Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. USENIX Security'18

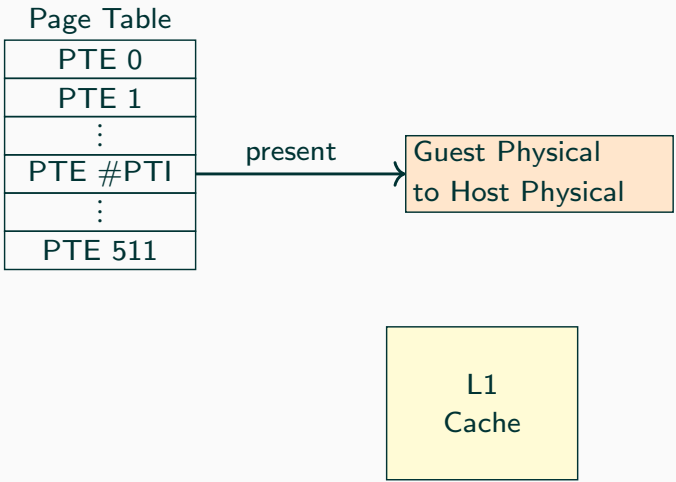
- An even worse bug → Foreshadow-NG/L1TF
- Exploitable from VMs
- Allows leaking data from the L1 cache
- Same mechanism as Meltdown
- Just a different bit in the PTE

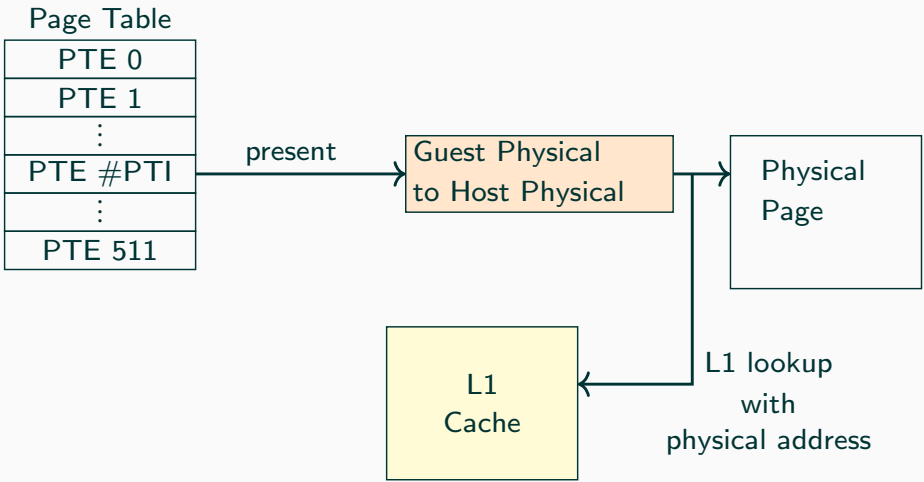
Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution.
Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. USENIX Security'18

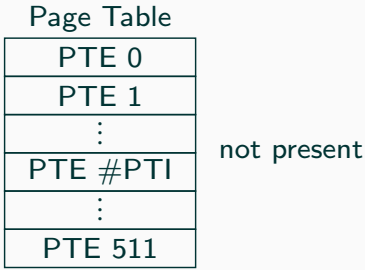

Page Table
PTE 0
PTE 1
PTE #PTI
PTE 511

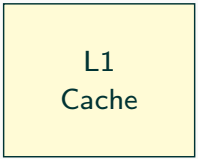

L1
Cache

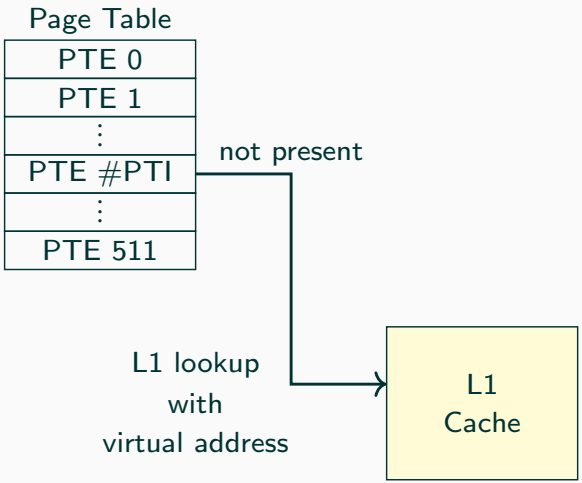

Page Table
PTE 0
PTE 1
PTE #PTI
PTE 511
present
L1
Cache


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


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


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


L1
Cache


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

- KAISER/KPTI/KVA does not help

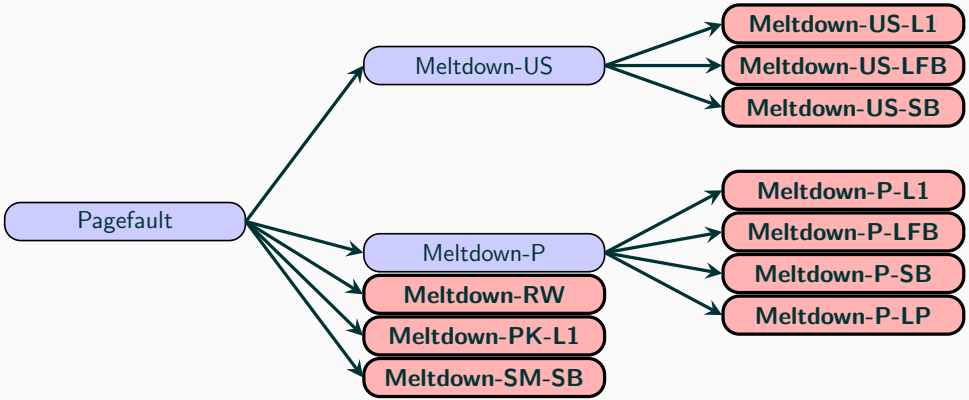
- KAISER/KPTI/KVA does not help
- Only software workarounds

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

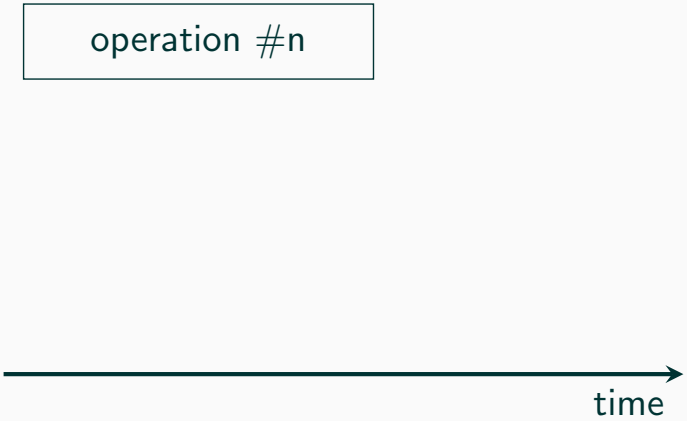
- KAISER/KPTI/KVA does not help
- Only software workarounds
  - → Flush L1 on VM entry
  - → Disable HyperThreading

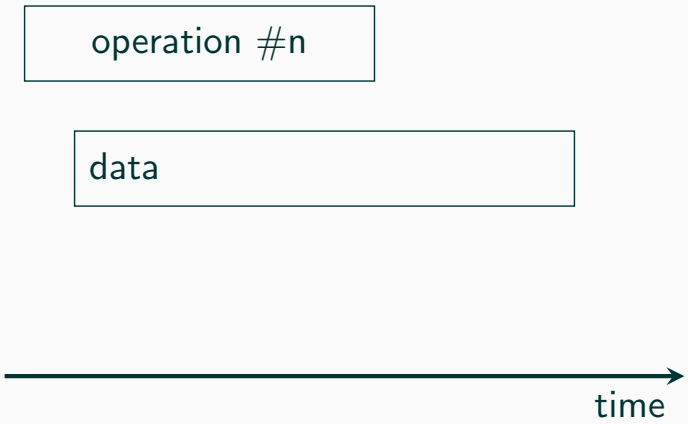
- KAISER/KPTI/KVA does not help
- Only software workarounds
  - → Flush L1 on VM entry
  - → Disable HyperThreading
- Workarounds might not be complete

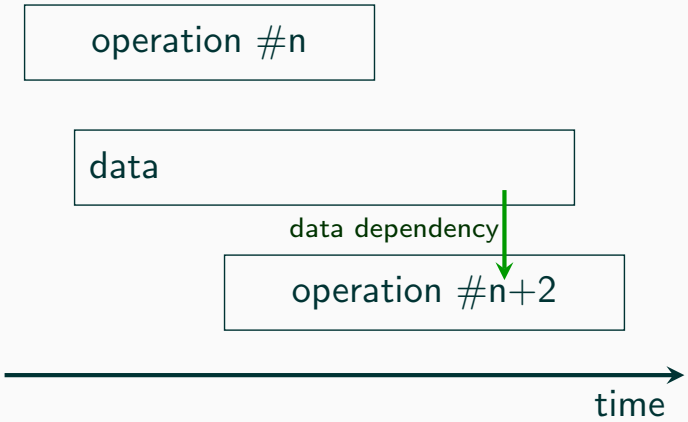
Meltdown Variants

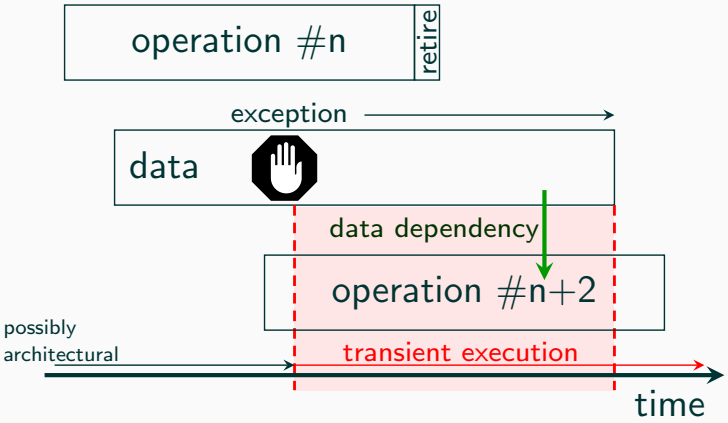


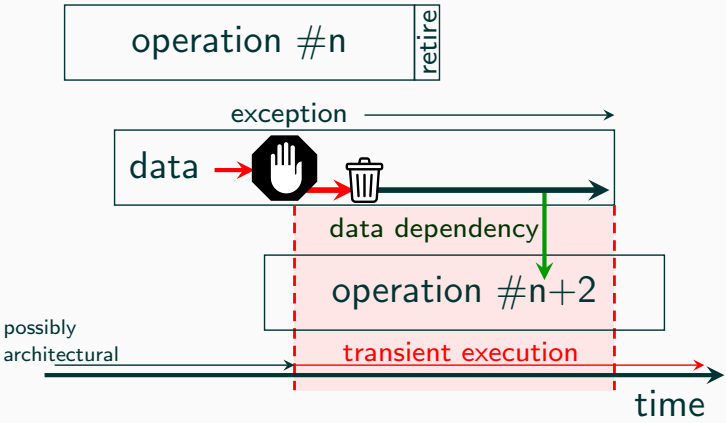
A Systematic Evaluation of Transient Execution Attacks and Defenses.
Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, Daniel Gruss.
USENIX Security'19

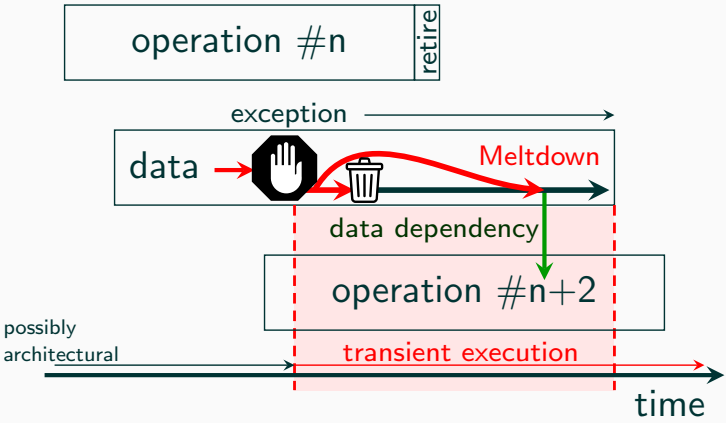

operation #n
time

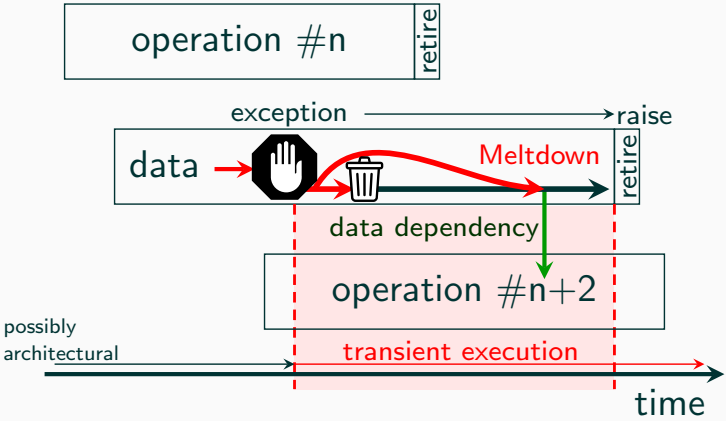

operation #n
data
time


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


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

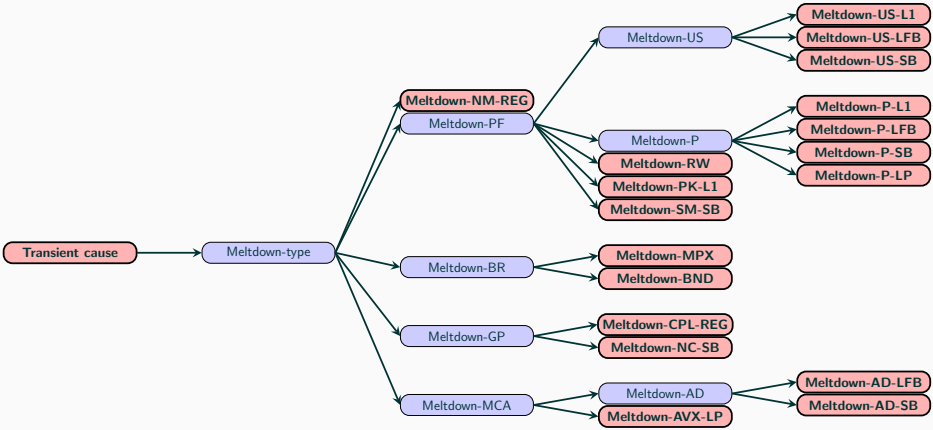

operation #n
retire
exception
data
data dependency
operation #n+2
possibly
architectural
transient execution
time


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


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

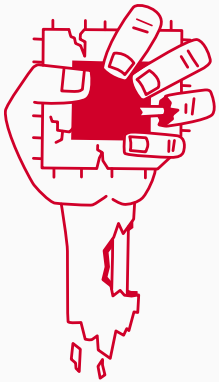

YOU GET A FAULT
AND YOU GET A FAULT.
EVERYONE GETS A FAULT
imgflip.com

Meltdown Tree



A Systematic Evaluation of Transient Execution Attacks and Defenses.
Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, Daniel Gruss.
USENIX Security'19

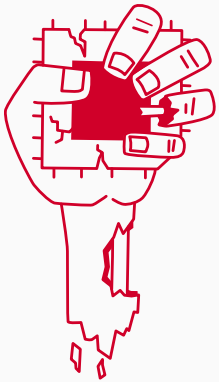
Latest Meltdown Variant: ZombieLoad



- Leaks from the fill buffer

ZombieLoad: Cross-Privilege-Boundary Data Sampling.
Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, Daniel Gruss. CCS'19

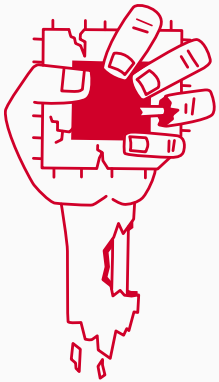
Latest Meltdown Variant: ZombieLoad



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

ZombieLoad: Cross-Privilege-Boundary Data Sampling.
Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, Daniel Gruss. CCS'19

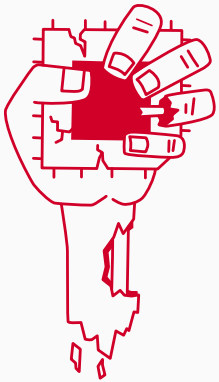
Latest Meltdown Variant: ZombieLoad



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

ZombieLoad: Cross-Privilege-Boundary Data Sampling.
Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, Daniel Gruss. CCS'19

Latest Meltdown Variant: ZombieLoad



- 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

ZombieLoad: Cross-Privilege-Boundary Data Sampling.
Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, Daniel Gruss. CCS'19


zombieload : zsh — Konsole <2>
File Edit View Bookmarks Settings Help
michael@hp /tmp/zombieload %
zombieload : zsh

Meltdown Outlook

- Meltdown is not a fully solved issue

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

Meltdown Outlook

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

Meltdown Outlook

- 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), ...

Spectre Attacks: Exploiting Speculative Execution.
Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom. S&P'19

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

Spectre Attacks: Exploiting Speculative Execution.
Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom. S&P'19

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

Spectre Attacks: Exploiting Speculative Execution.
Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom. S&P'19

Speculative Execution

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

Spectre Attacks: Exploiting Speculative Execution.
Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom. S&P'19

Speculative Execution

- 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

Spectre Attacks: Exploiting Speculative Execution.
Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom. S&P'19

Speculative Execution

- 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

Spectre Attacks: Exploiting Speculative Execution.
Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom. S&P'19

```
index = 0
```

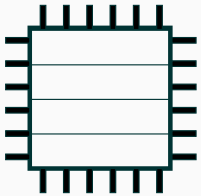
Shared Memory

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

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

then → `glyph[data[index]]`

else → `{}`



Memory

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

Spectre-PHT (aka Spectre Variant 1)

```
index = 0
```

Shared Memory

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

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

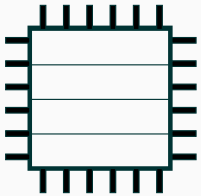
then

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

else

Speculate

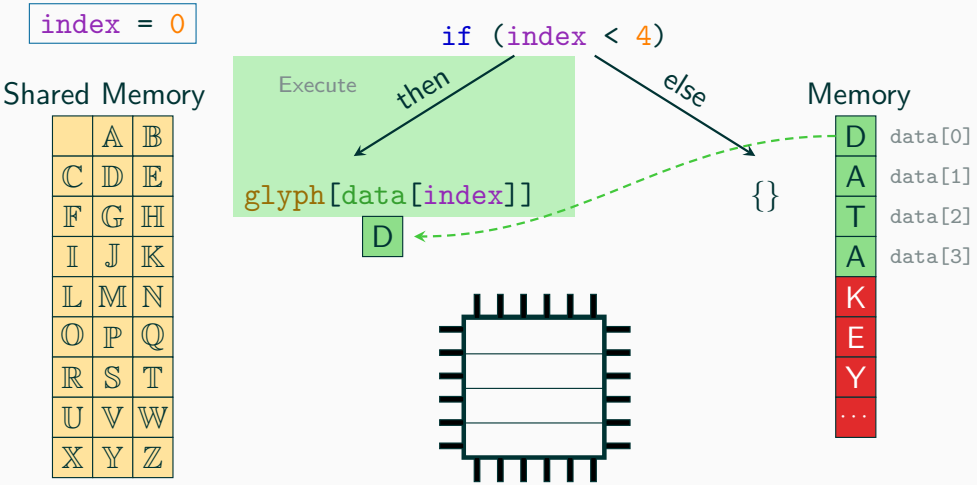
{}

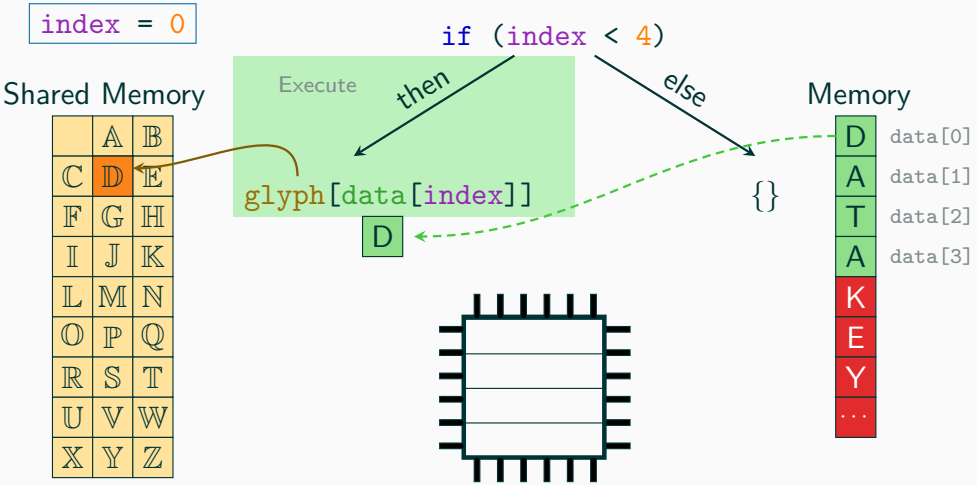


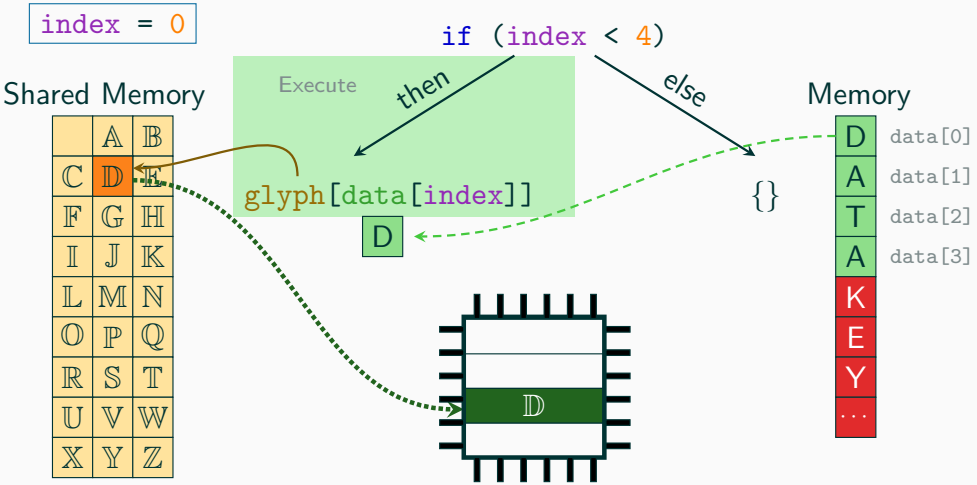
Memory

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

Spectre-PHT (aka Spectre Variant 1)




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


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

Spectre-PHT (aka Spectre Variant 1)

```
index = 1
```

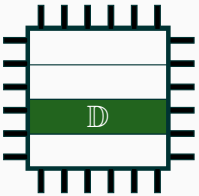
Shared Memory

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

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

then → `glyph[data[index]]`

else → `{}`



Memory

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

Spectre-PHT (aka Spectre Variant 1)

```
index = 1
```

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)
```

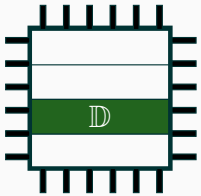
Speculate

then

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

else

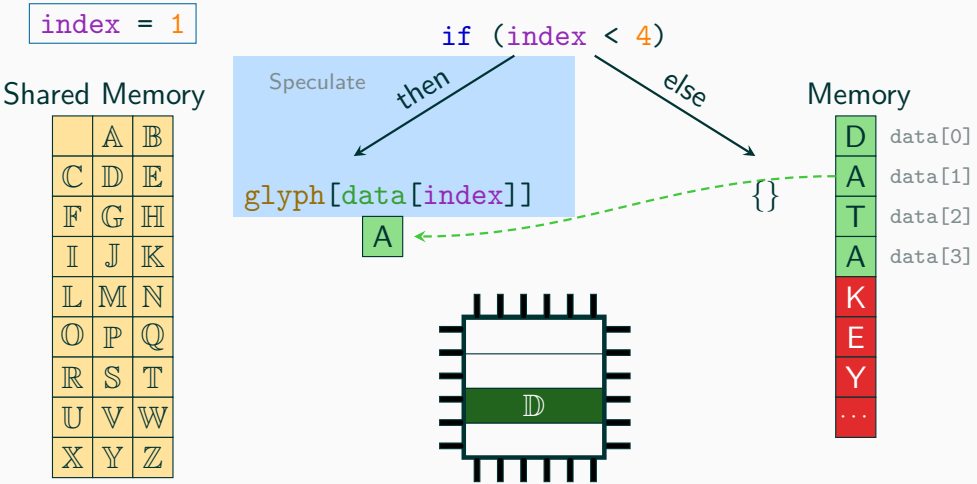
{}

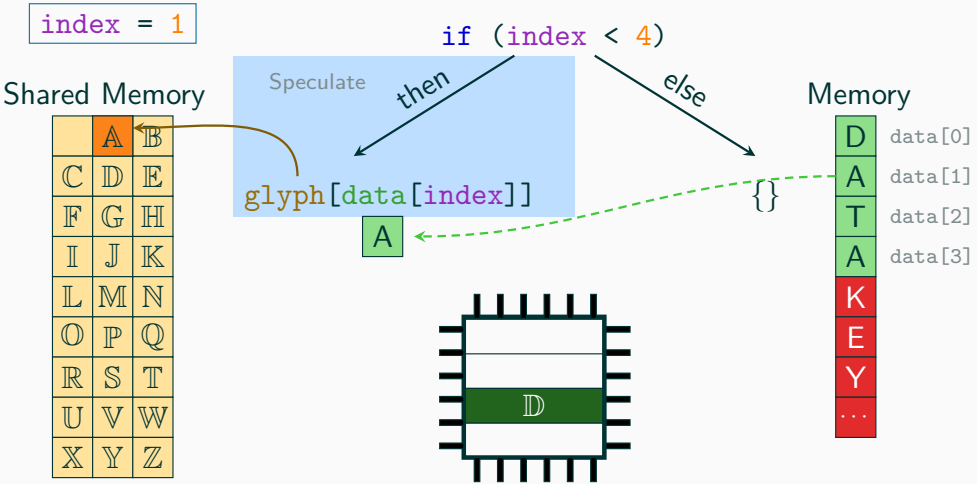


Memory

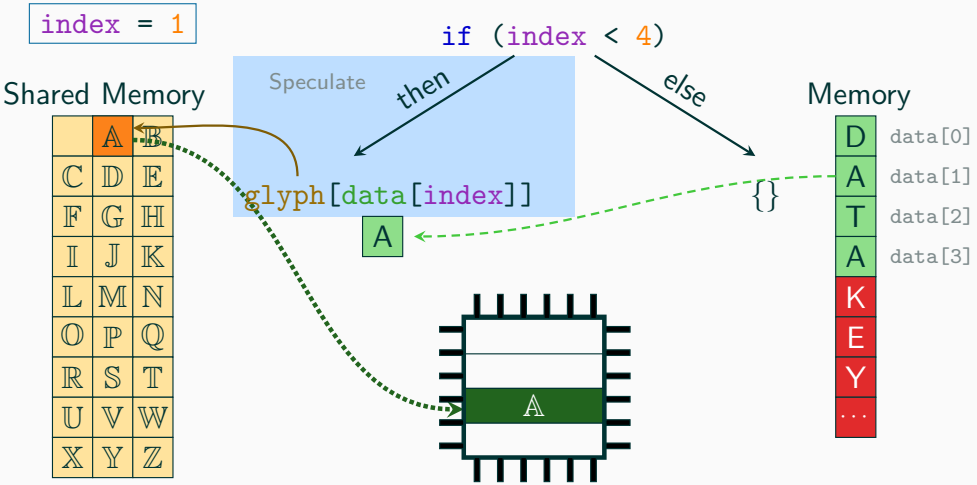
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

Spectre-PHT (aka Spectre Variant 1)




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

Spectre-PHT (aka Spectre Variant 1)



```
index = 1
```

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

Execute

then

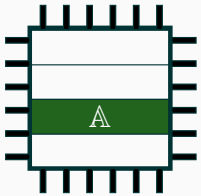
```
glyph[data[index]]
```

else

{}

Shared Memory

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



Memory

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

Spectre-PHT (aka Spectre Variant 1)

```
index = 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

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

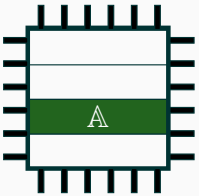
Speculate

then

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

else

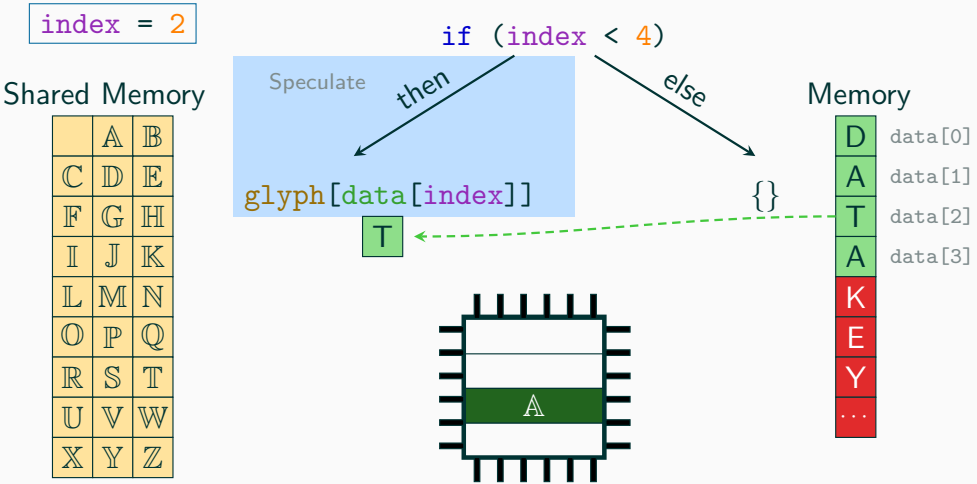
{}



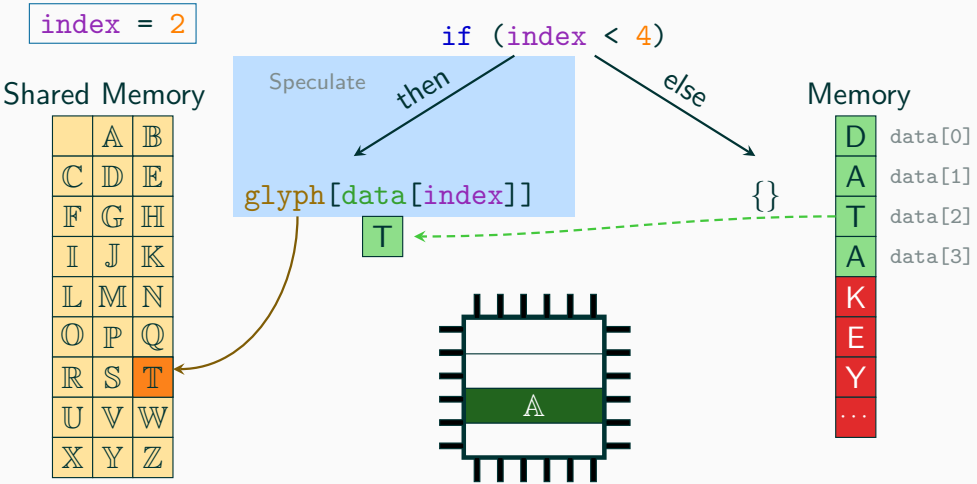
Memory

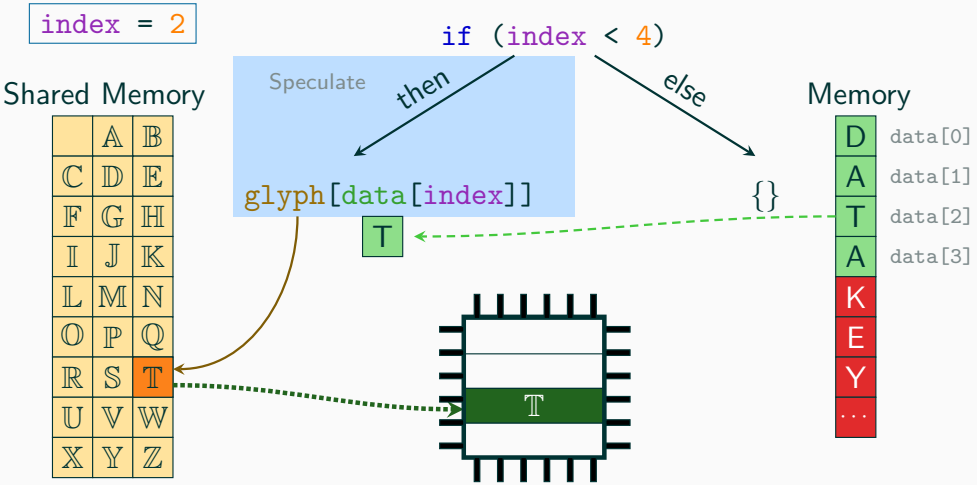
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

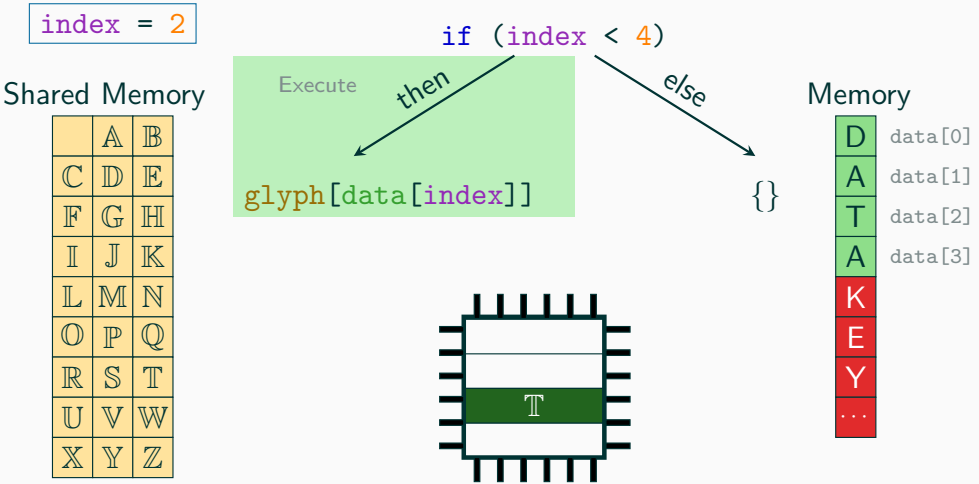
Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)




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


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

```
index = 3
```

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

Speculate

then

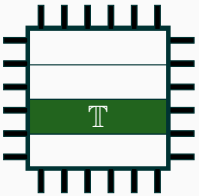
```
glyph[data[index]]
```

else

```
{}
```

Shared Memory

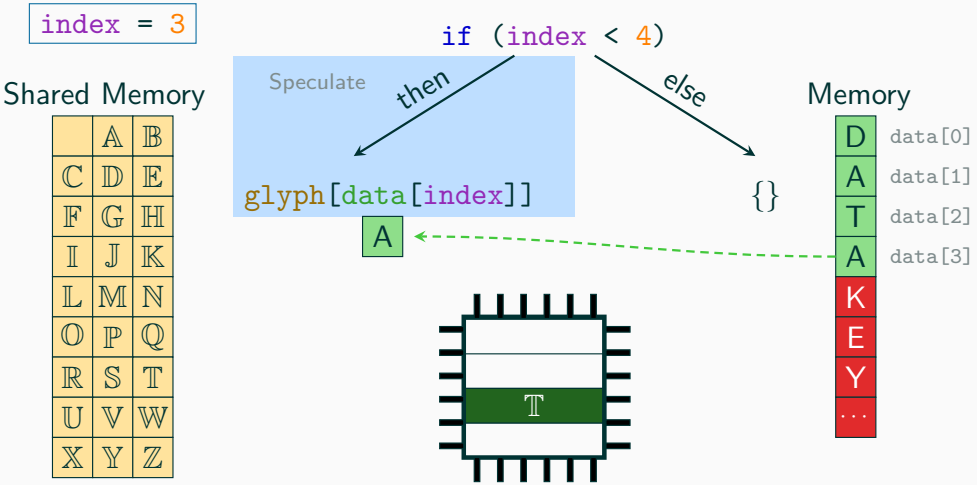
	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z



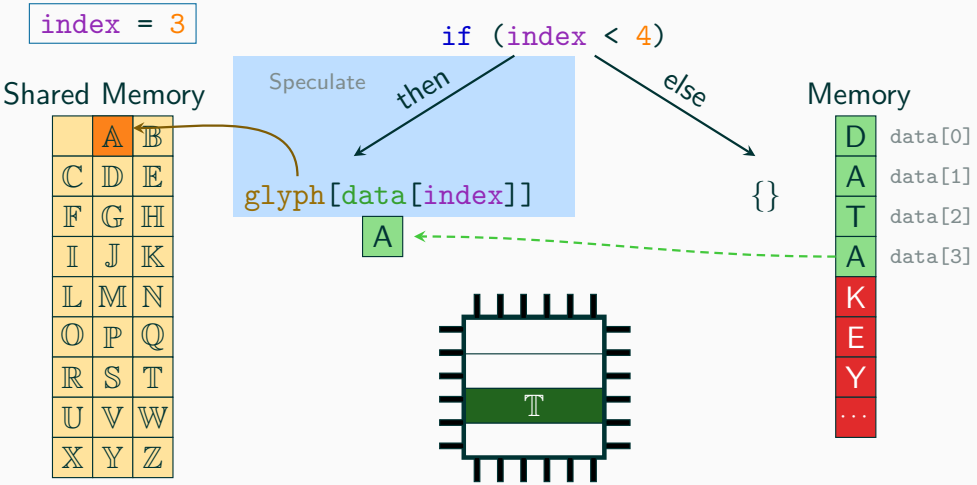
Memory

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

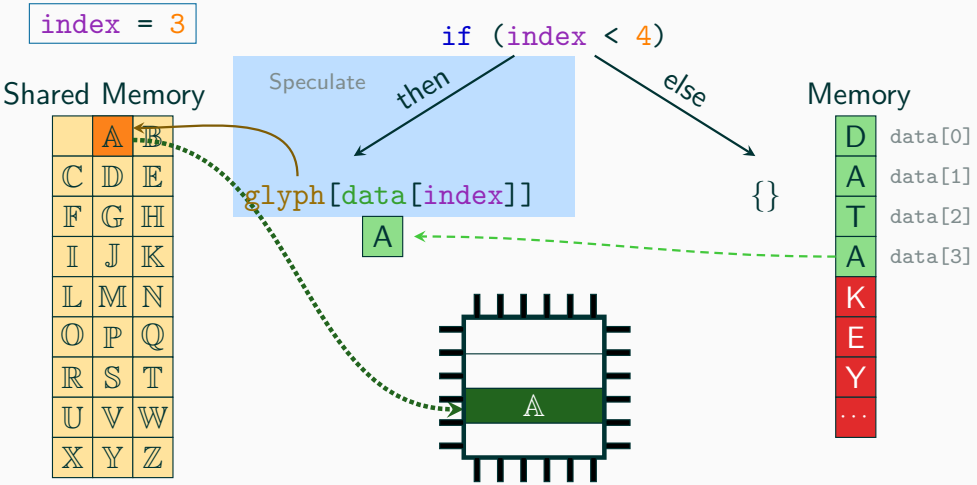
Spectre-PHT (aka Spectre Variant 1)

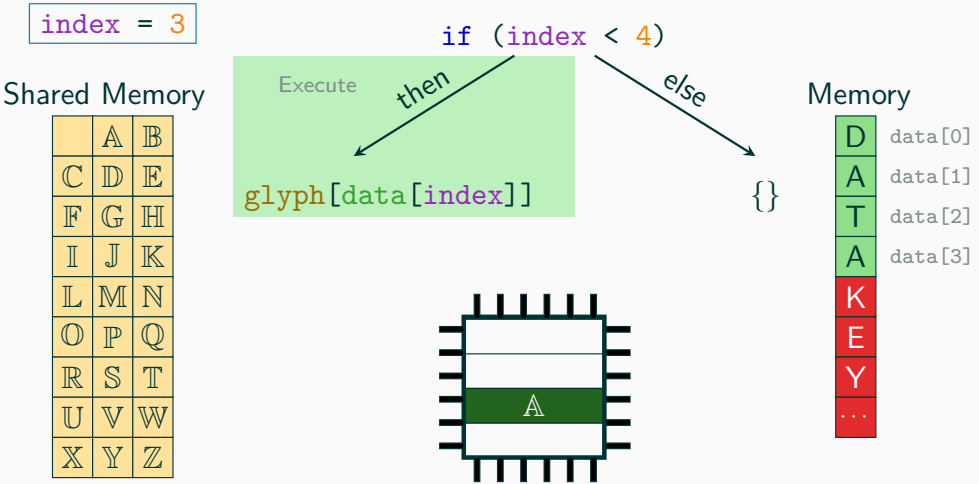


Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)




index = 3
if (index < 4)
Execute
then
else
glyph[data[index]]
{}
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
...
A

```
index = 4
```

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

Speculate

then

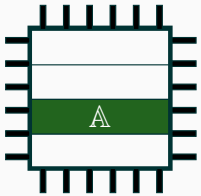
```
glyph[data[index]]
```

else

```
{}
```

Shared Memory

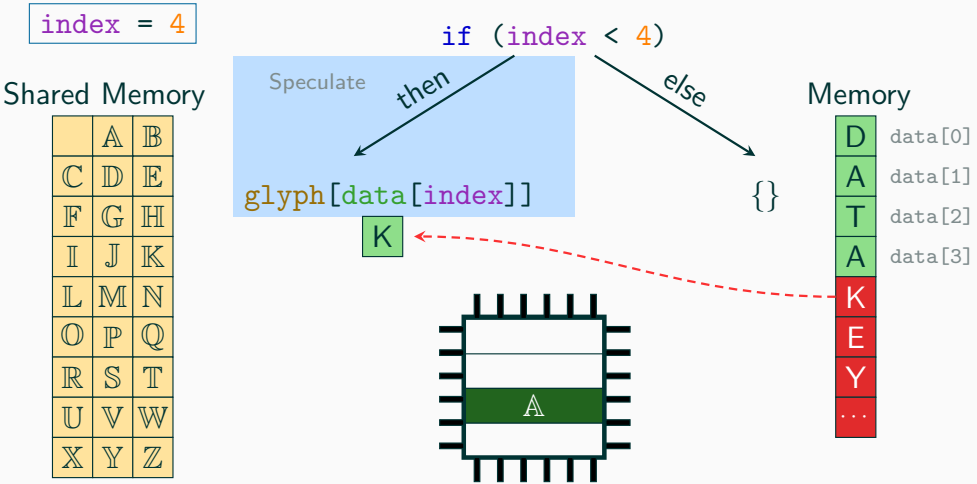
	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z



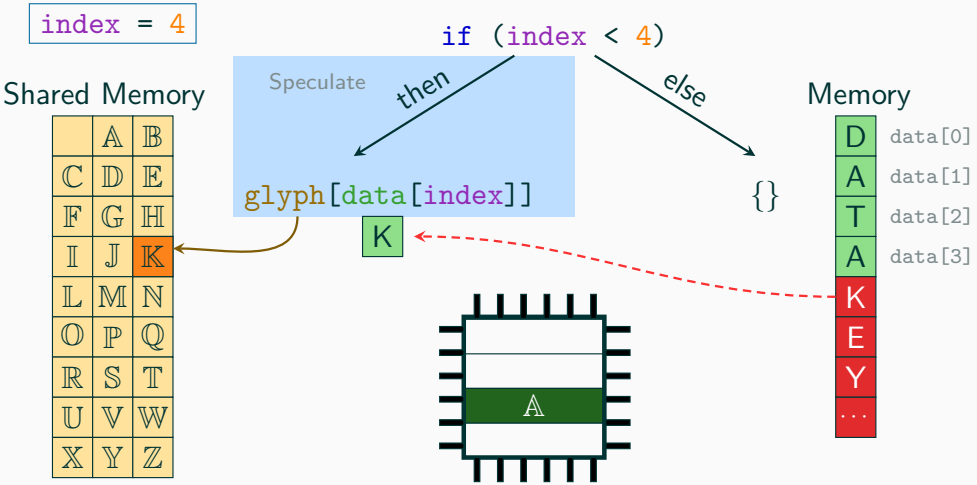
Memory

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

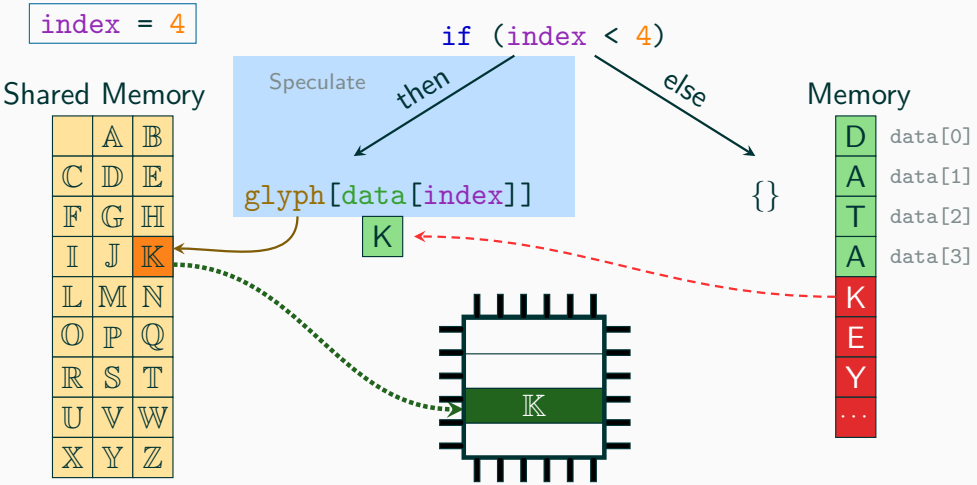
Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)

```
index = 4
```

Shared Memory

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

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

then

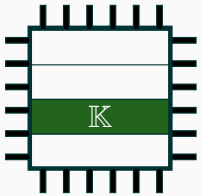
```
glyph[data[index]]
```

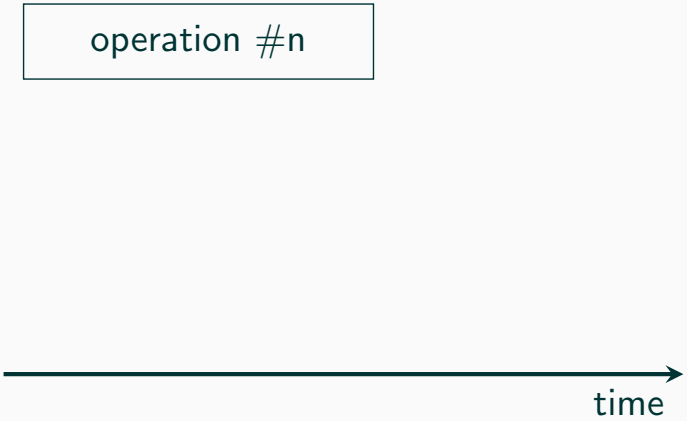
else

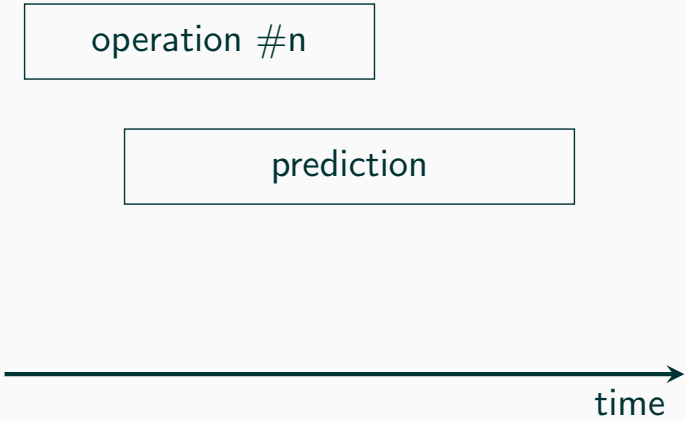
Execute {}

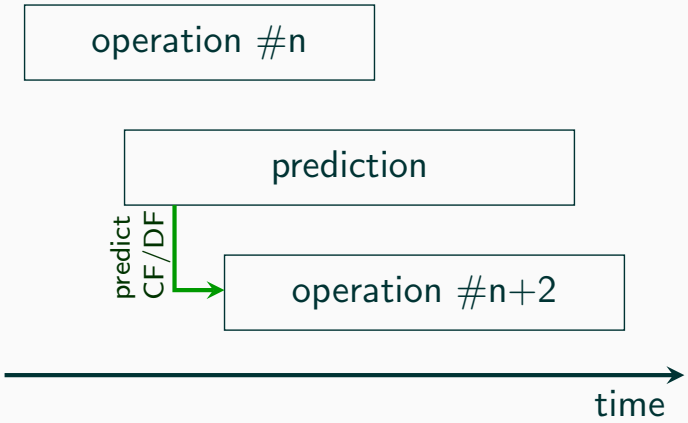
Memory

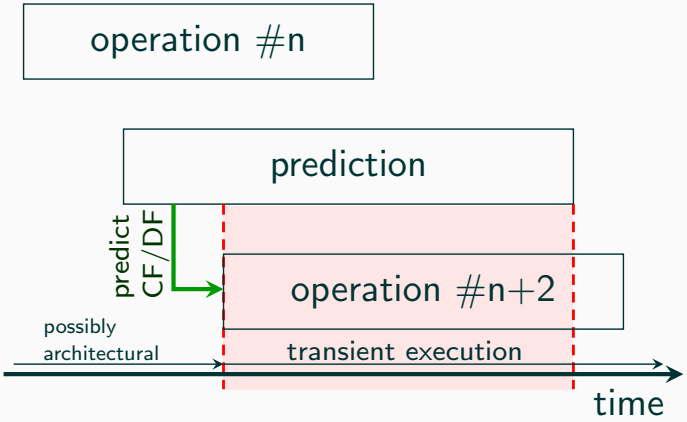
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

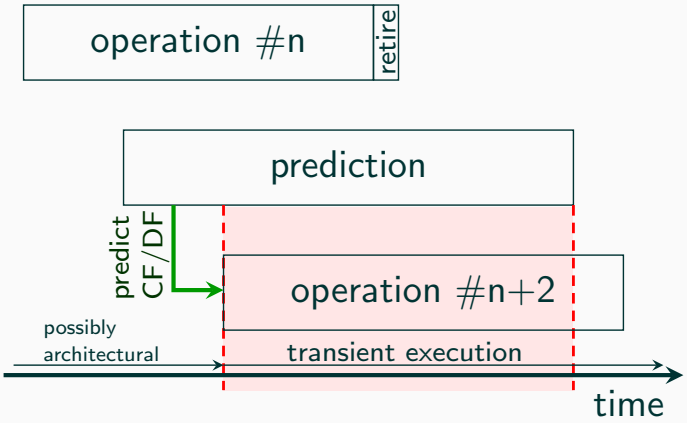


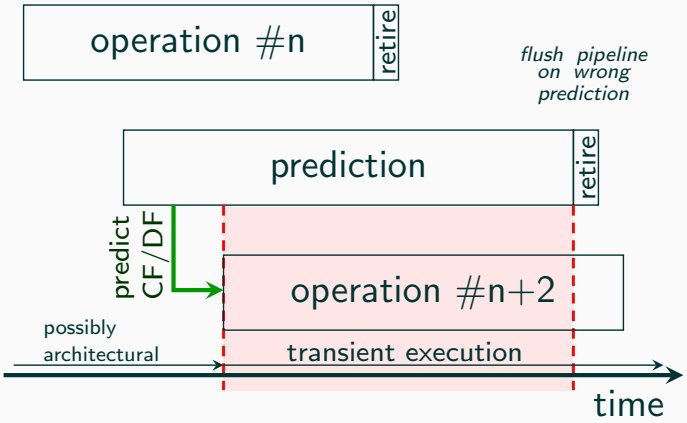

operation #n
time

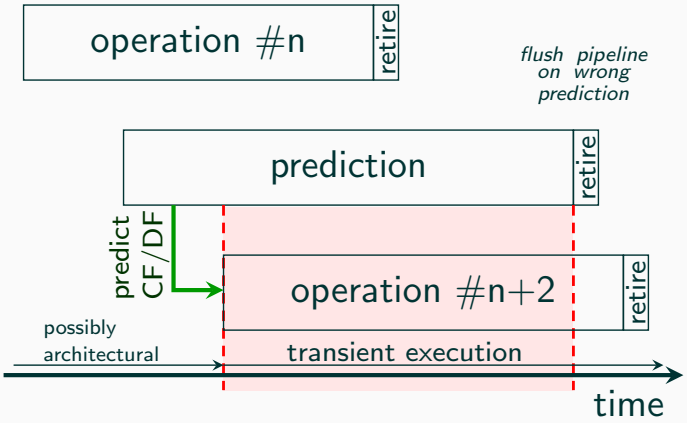

operation #n
prediction
time

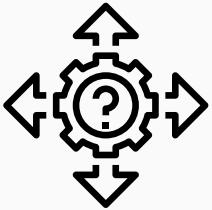

operation #n
prediction
predict
CF/DF
operation #n+2
time


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

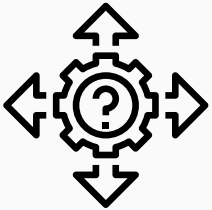

operation #n
retire
prediction
predict CF/DF
operation #n+2
possibly architectural
transient execution
time


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

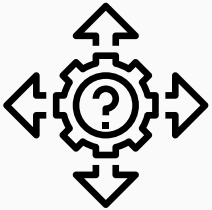

operation #n
retire
flush pipeline on wrong prediction
prediction
retire
predict CF/DF
operation #n+2
retire
possibly architectural
transient execution
time



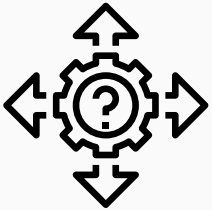
- Many predictors in modern CPUs



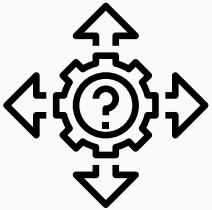
- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)



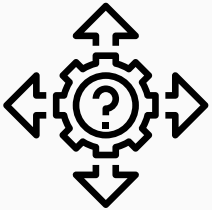
- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)
  - Call/Jump destination (BTB)



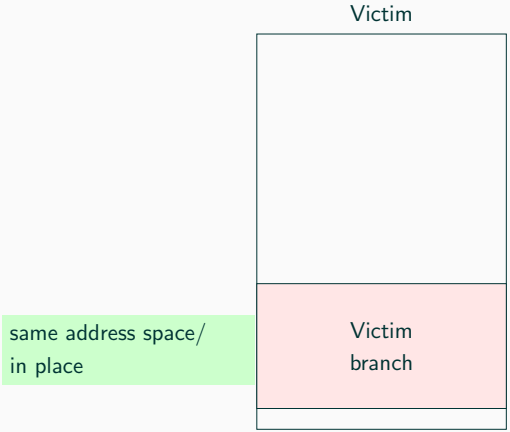
- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)
  - Call/Jump destination (BTB)
  - Function return destination (RSB)

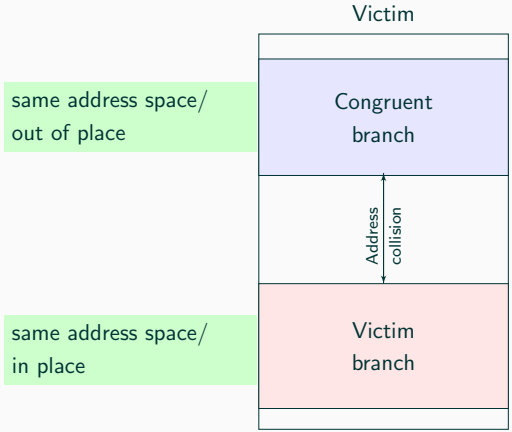


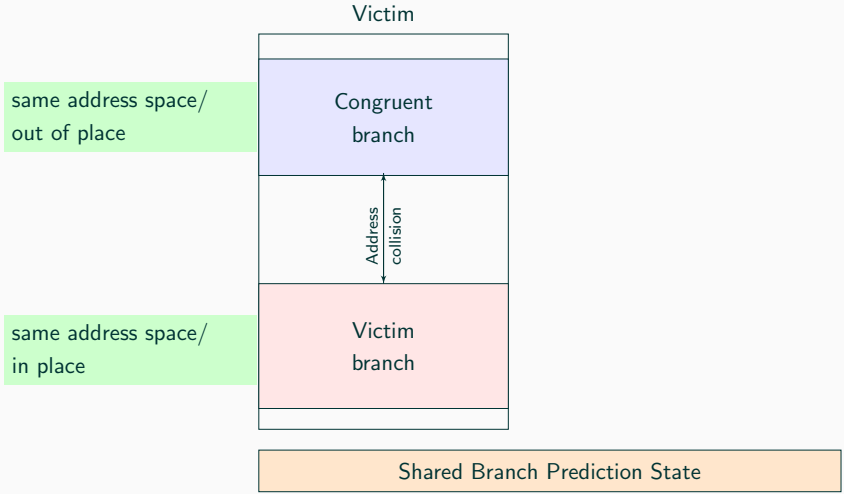
- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)
  - Call/Jump destination (BTB)
  - Function return destination (RSB)
  - Load matches previous store (STL)



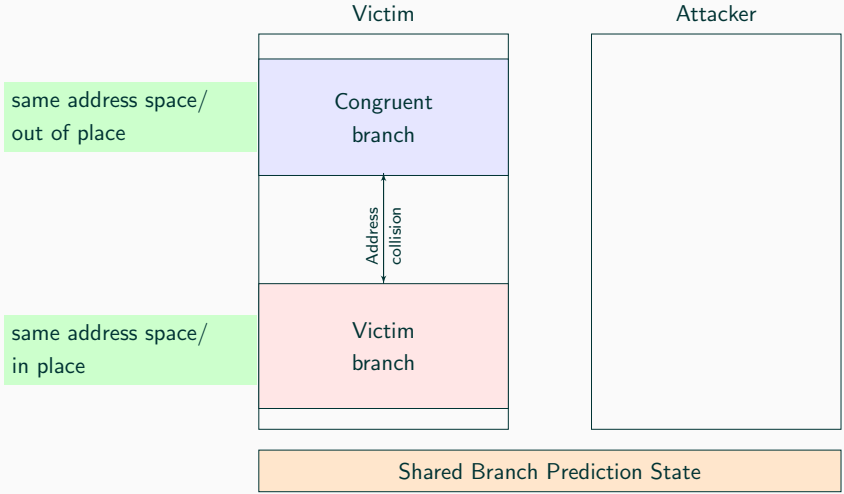
- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)
  - Call/Jump destination (BTB)
  - Function return destination (RSB)
  - Load matches previous store (STL)
- Most are even shared among processes

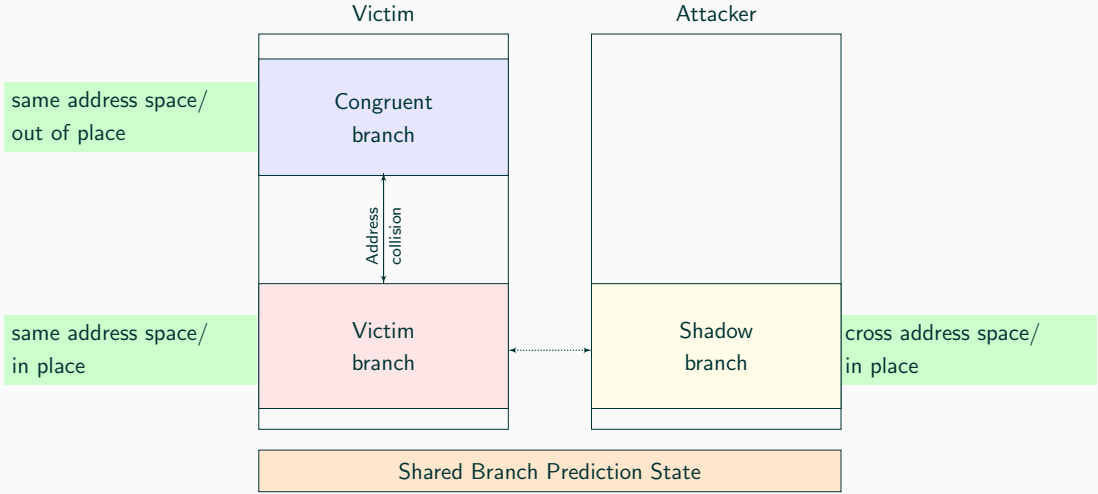

Victim
same address space/
in place
Victim
branch

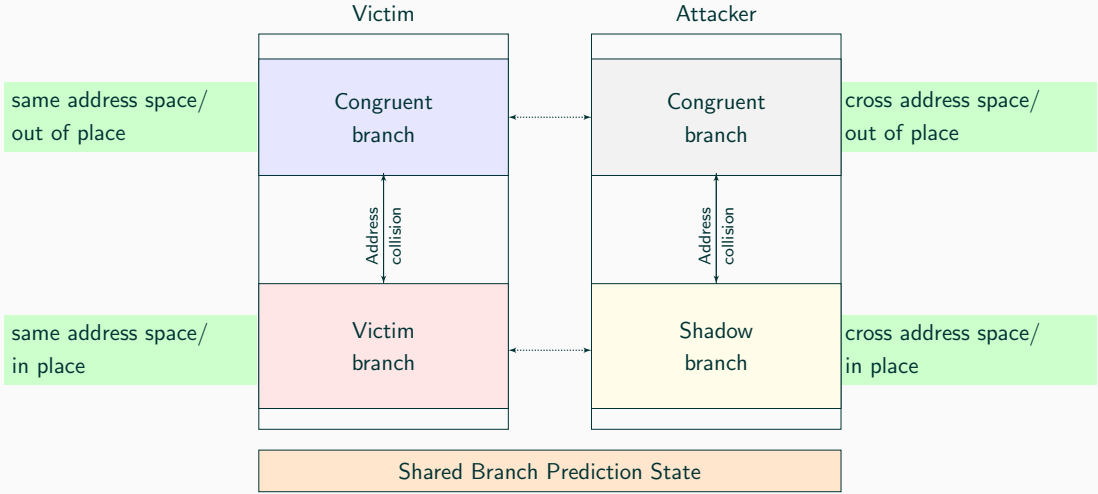

Victim
same address space/
out of place
Congruent
branch
Address
collision
same address space/
in place
Victim
branch


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

Spectre Mistraining

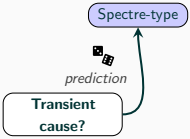



Victim
Attacker
same address space/
out of place
Congruent
branch
Address
collision
same address space/
in place
Victim
branch
Shadow
branch
cross address space/
in place
Shared Branch Prediction State

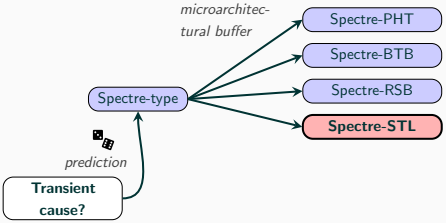

Victim
Attacker
same address space/
out of place
Congruent
branch
Congruent
branch
cross address space/
out of place
Address
collision
Address
collision
same address space/
in place
Victim
branch
Shadow
branch
cross address space/
in place
Shared Branch Prediction State

Transient cause?

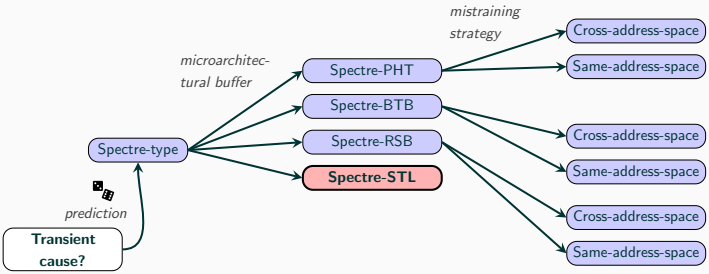
Spectre Variants



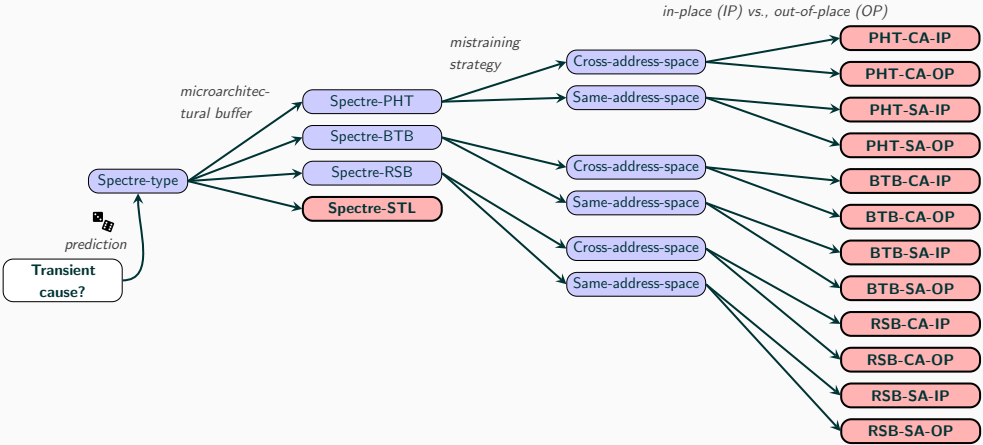
Spectre Variants

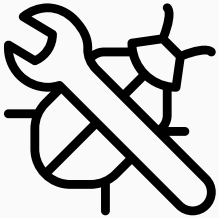


Spectre Variants

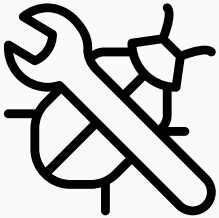


Spectre Variants

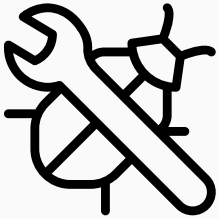




- Spectre is not a bug

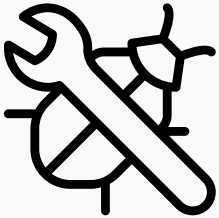


- Spectre is not a bug
- It is an useful optimization



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

→ Cannot simply fix it (as with Meltdown)

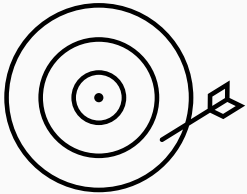


- Spectre is not a bug
- It is an useful optimization

→ Cannot simply fix it (as with Meltdown)

- Workarounds for critical code parts

Spectre defenses in 3 categories:



C1 Mitigating or reducing the accuracy of covert channels

C2 Mitigating or aborting speculation

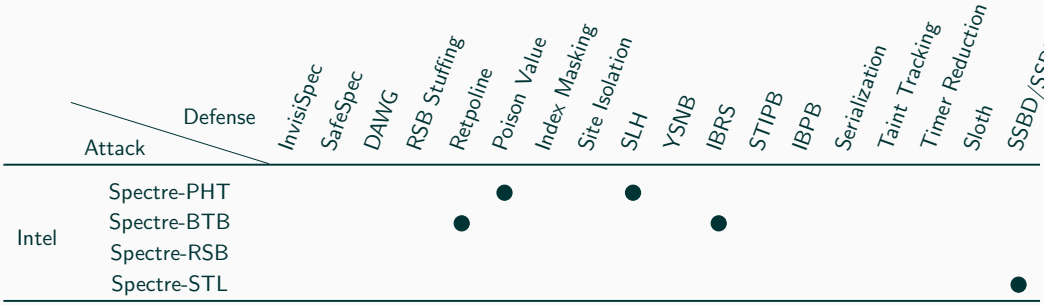
C3 Ensuring secret data cannot be reached

Spectre: Defense Analysis

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

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

Spectre: Defense Analysis



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

Spectre: Defense Analysis

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

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

Spectre: Defense Analysis

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

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

Spectre: Defense Analysis

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

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

Spectre: Defense Analysis

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

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

Spectre: Defense Analysis

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

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

Spectre: Defense Analysis

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

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

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

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

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

Spectre Misconceptions

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

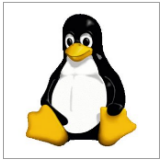
Spectre Misconceptions

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

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

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



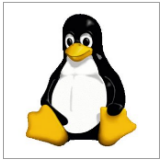
On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

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



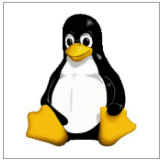
On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

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



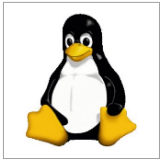
On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

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

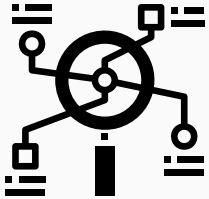
Written by Michael Larabel in Linux Kernel on 24 November 2018 at 09:00 AM EST. 6 Comments



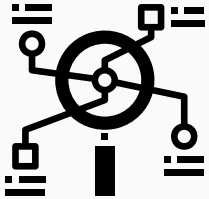
On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

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).

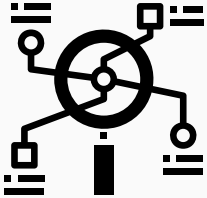


- Current mitigations are either incomplete or cost performance



- Current mitigations are either incomplete or cost performance

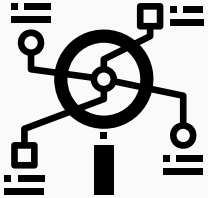
→ More research required



- Current mitigations are either incomplete or cost performance

→ More research required

- Both on attacks and defenses

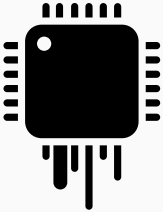


- Current mitigations are either incomplete or cost performance

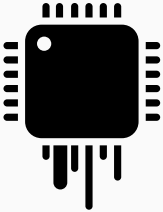
→ More research required

- Both on attacks and defenses

→ Efficient defenses only possible when attacks are known

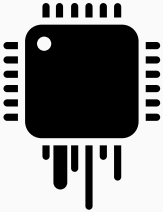


- Side channels so far
  - leak meta data

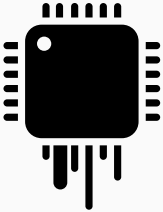


- Side channels so far
  - leak meta data
  - covertly transmit data

Leaking Data



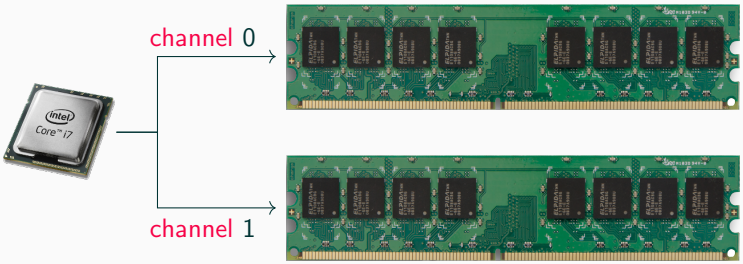
- Side channels so far
  - leak meta data
  - covertly transmit data
- As a building block
  - leak data



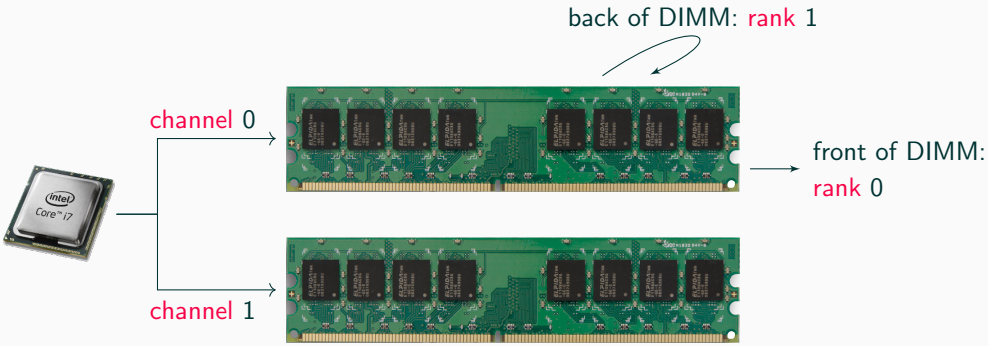
- Side channels so far
  - leak meta data
  - covertly transmit data
- As a building block
  - leak data
- What about modifying data?

DRAM organization

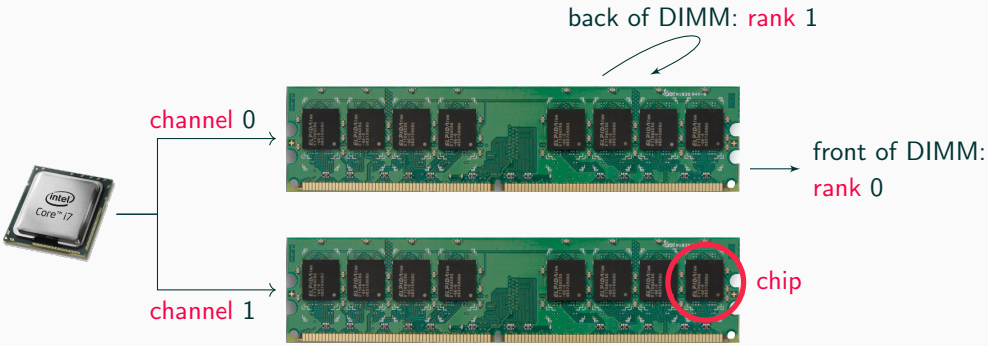
DRAM organization



DRAM organization

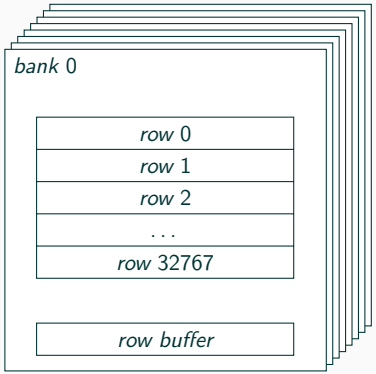


DRAM organization



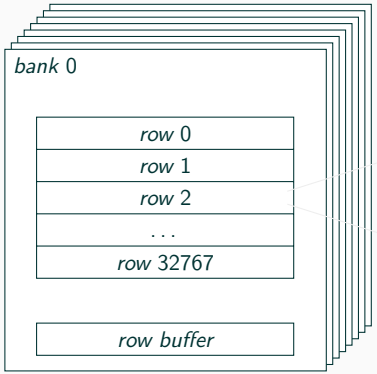
DRAM organization

chip



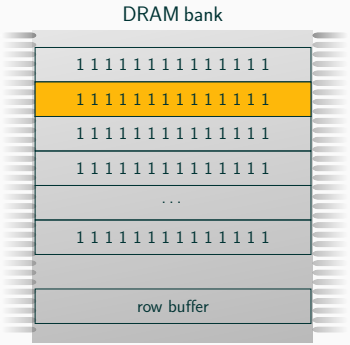
DRAM organization

chip



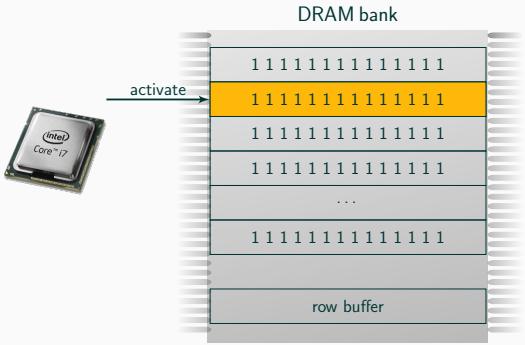
64k cells
1 capacitor,
1 transitor each

How reading from DRAM works



CPU wants to access row 1

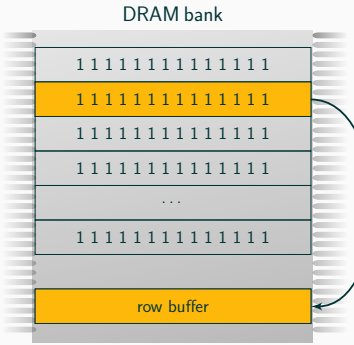
How reading from DRAM works



CPU wants to access row 1

→ row 1 activated

How reading from DRAM works

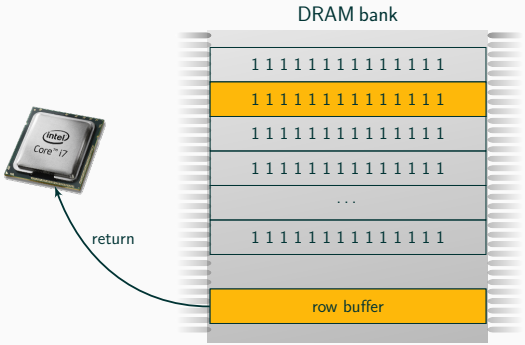


CPU wants to access row 1

→ row 1 activated

→ row 1 copied to row buffer

How reading from DRAM works

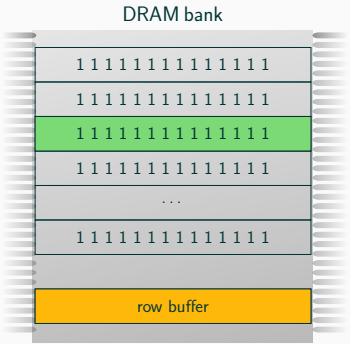


CPU wants to access row 1

→ row 1 activated

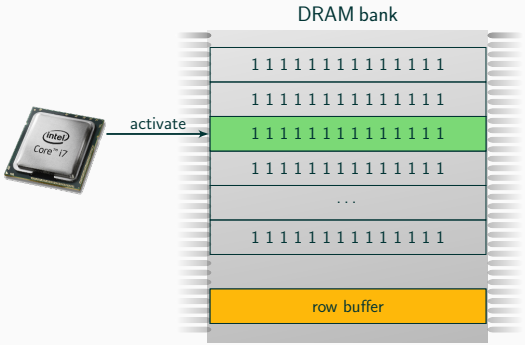
→ row 1 copied to row buffer

How reading from DRAM works



CPU wants to access row 2

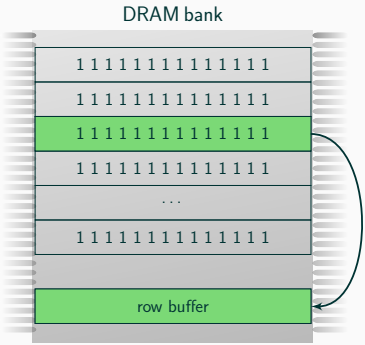
How reading from DRAM works



CPU wants to access row 2

$\rightarrow$ row 2 activated

How reading from DRAM works

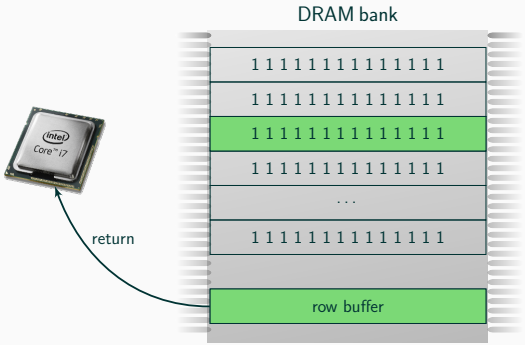


CPU wants to access row 2

→ row 2 activated

→ row 2 copied to row buffer

How reading from DRAM works

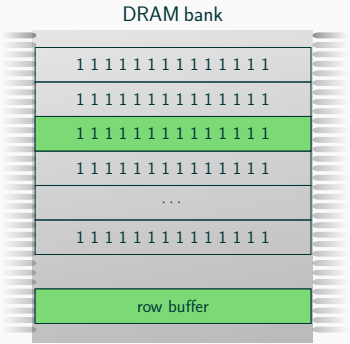


CPU wants to access row 2

→ row 2 activated

→ row 2 copied to row buffer

How reading from DRAM works



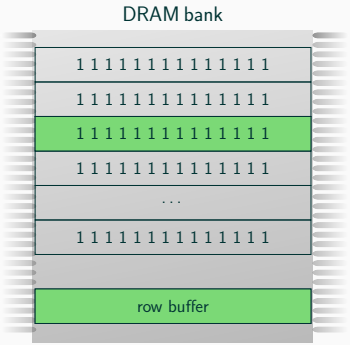
CPU wants to access row 2

→ row 2 activated

→ row 2 copied to row buffer

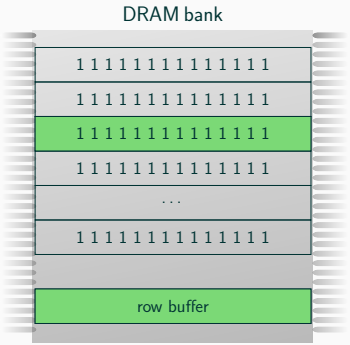
→ slow (row conflict)

How reading from DRAM works



CPU wants to access row 2—again

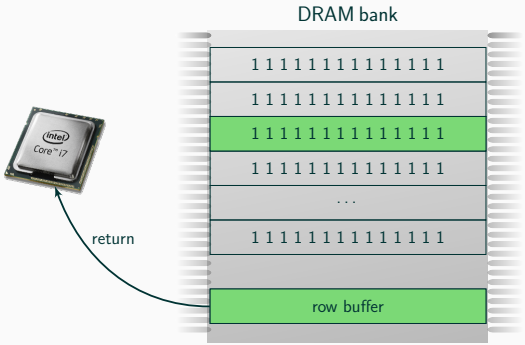
How reading from DRAM works



CPU wants to access row 2—again

→ row 2 already in row buffer

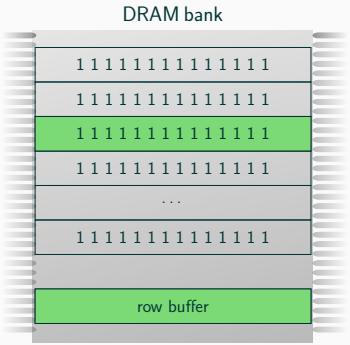
How reading from DRAM works



CPU wants to access row 2—again

→ row 2 already in row buffer

How reading from DRAM works

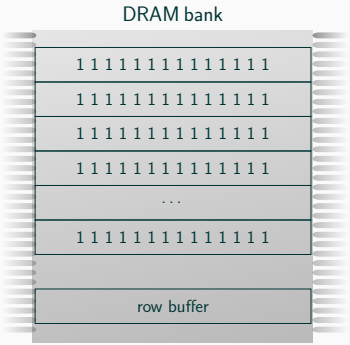


CPU wants to access row 2—again

→ row 2 already in row buffer

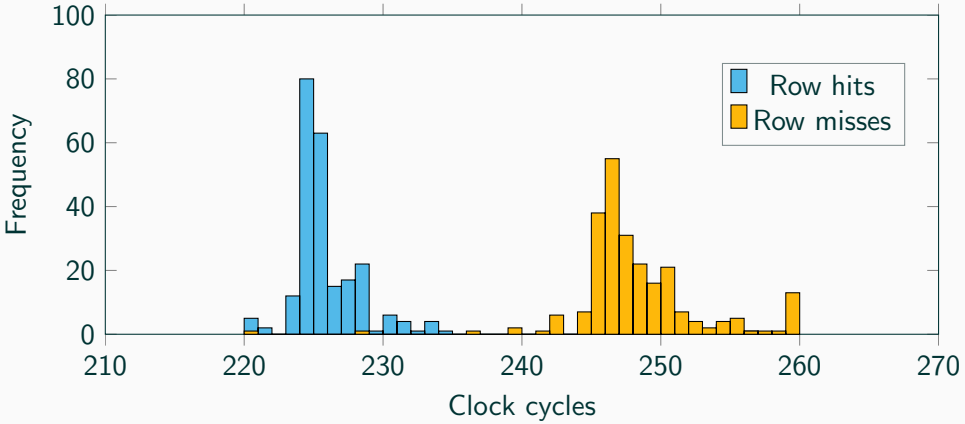
→ fast (row hit)

How reading from DRAM works



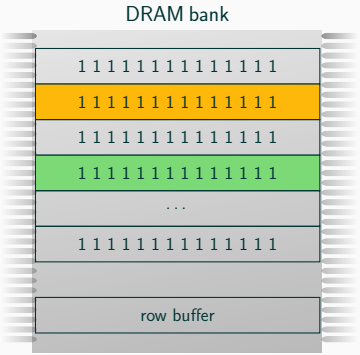
row buffer = cache

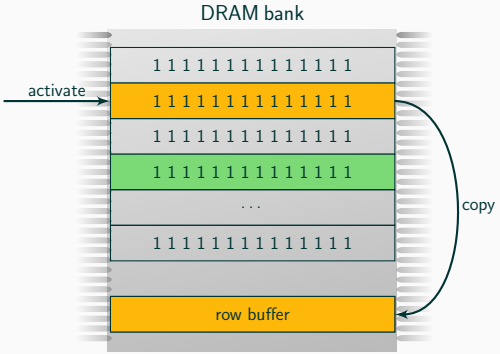
Timing difference

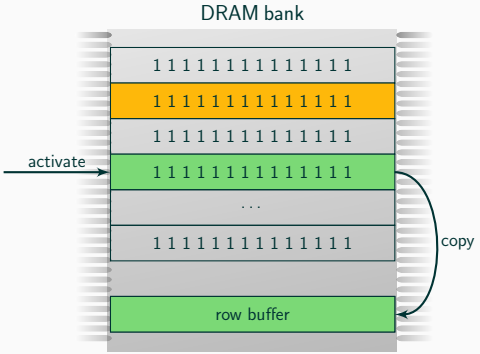


DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks.
Peter Pessl, Daniel Gruss, Clmentine Maurice, Michael Schwarz, Stefan Mangard. USENIX Security'16

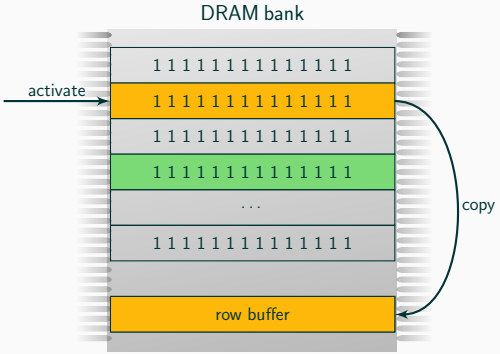
Rowhammer


DRAM bank
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
. . .
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
row buffer

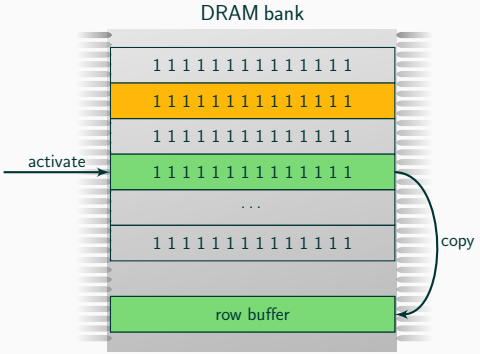

DRAM bank
activate
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
. . .
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
copy
row buffer



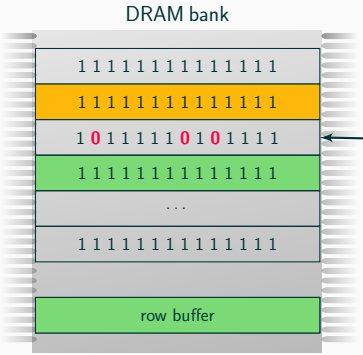
Cells leak faster upon proximate accesses → Rowhammer



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

How widespread is the issue?

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

How widespread is the issue?

- 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

Modifying Bits

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

Modifying Bits

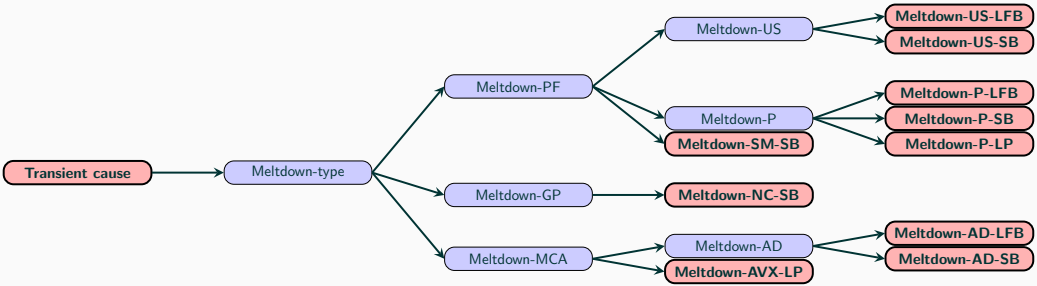
- Single bit flips allow
  - modifying instructions
  - breaking cryptography
  - changing permissions
  - crashing systems
  - ...

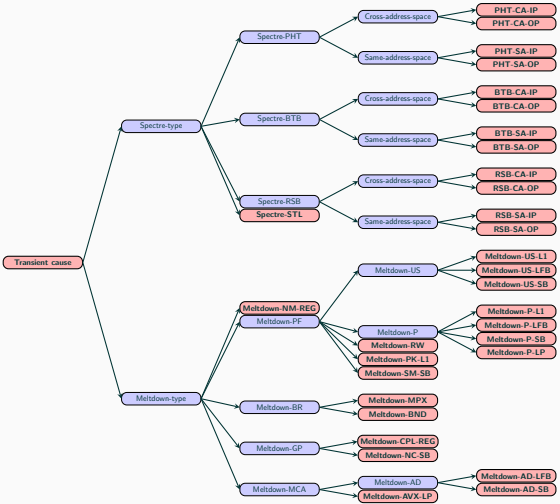
Modifying Bits

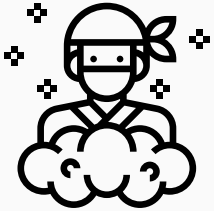
- Single bit flips allow
  - modifying instructions
  - breaking cryptography
  - changing permissions
  - crashing systems
  - ...
- In software, no permissions required

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

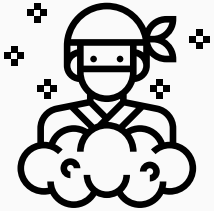
Microarchitectural Data Sampling (MDS)



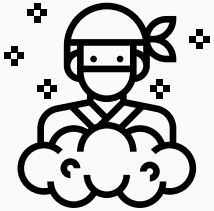

Transient cause
Spectre-type
Spectre-PHT
Cross-address-space
PHT-CA-IP
PHT-CA-OP
Same-address-space
PHT-SA-IP
PHT-SA-OP
Spectre-BTB
Cross-address-space
BTB-CA-IP
BTB-CA-OP
Same-address-space
BTB-SA-IP
BTB-SA-OP
Spectre-RSB
Spectre-STL
Cross-address-space
RSB-CA-IP
RSB-CA-OP
Same-address-space
RSB-SA-IP
RSB-SA-OP
Meltdown-type
Meltdown-US
Meltdown-US-L1
Meltdown-US-LFB
Meltdown-US-SB
Meltdown-NM-REG
Meltdown-PF
Meltdown-P
Meltdown-P-L1
Meltdown-P-LFB
Meltdown-P-SB
Meltdown-P-LP
Meltdown-RW
Meltdown-PK-L1
Meltdown-SM-SB
Meltdown-BR
Meltdown-MPX
Meltdown-BND
Meltdown-GP
Meltdown-CPL-REG
Meltdown-NC-SB
Meltdown-MCA
Meltdown-AD
Meltdown-AD-LFB
Meltdown-AD-SB
Meltdown-AVX-LP



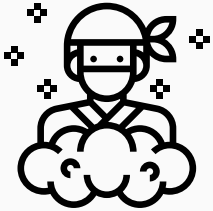
- Transient Execution Attacks are...



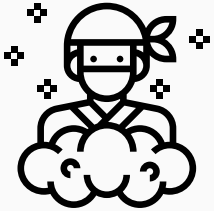
- Transient Execution Attacks are...
  - ...a novel class of attacks



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

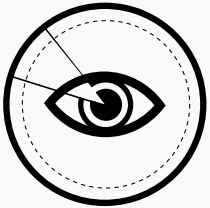


- Transient Execution Attacks are...
  - ...a novel class of attacks
  - ...extremely powerful
  - ...only at the beginning



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

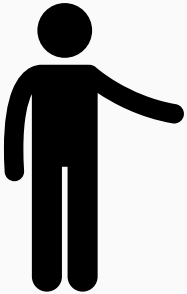
A unique chance



A unique chance to

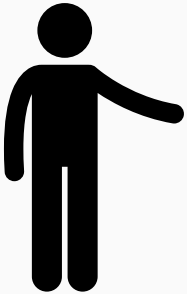
- rethink processor design
- grow up, like other fields (car industry, construction industry)

Conclusion



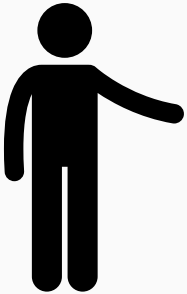
- Optimizations in hardware often lead to side channels

Conclusion



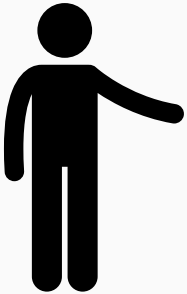
- Optimizations in hardware often lead to side channels
- Unknown and novel side channels are likely to exist

Conclusion



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

Conclusion



- 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


BRACE YOURSELVES
MORE BUGS ARE COMING
imgflip.com

Any Questions?