Spectre and Meltdown on x86 and ARM

Michael Schwarz, Moritz Lipp, Stefan Mangard

15.02.2018

www.iaik.tugraz.at

Vulnerability Assessment

- Meltdown and Spectre are two CPU vulnerabilities

Vulnerability Assessment

- Meltdown and Spectre are two CPU vulnerabilities
- Discovered in 2017 by 4 independent teams

Vulnerability Assessment

- Meltdown and Spectre are two CPU vulnerabilities
- Discovered in 2017 by 4 independent teams
- Due to an embargo, released at the beginning of 2018

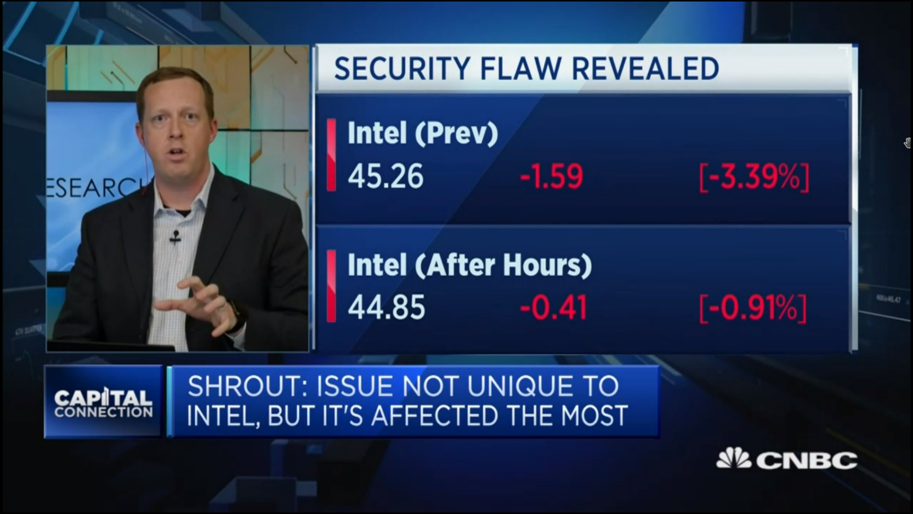
Vulnerability Assessment

- Meltdown and Spectre are two CPU vulnerabilities
- Discovered in 2017 by 4 independent teams
- Due to an embargo, released at the beginning of 2018
- News coverage followed by a lot of panic

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

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

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



SECURITY FLAW REVEALED

Intel (Prev)		
45.26	-1.59	[-3.39%]
Intel (After Hours)		
44.85	-0.41	[-0.91%]

CAPITAL CONNECTION

SHROUT: ISSUE NOT UNIQUE TO INTEL, BUT IT'S AFFECTED THE MOST

Vulnerability Assesment

A lot of confusion fueled the panic

- Which CPUs/vendors are affected?

Vulnerability Assesment

A lot of confusion fueled the panic

- Which CPUs/vendors are affected?
- Are smartphones/IoT devices affected?

Vulnerability Assesment

A lot of confusion fueled the panic

- Which CPUs/vendors are affected?
- Are smartphones/IoT devices affected?
- Can the vulnerabilities be exploited remotely?

Vulnerability Assesment

A lot of confusion fueled the panic

- Which CPUs/vendors are affected?
- Are smartphones/IoT devices affected?
- Can the vulnerabilities be exploited remotely?
- What data is at risk?

Vulnerability Assesment

A lot of confusion fueled the panic

- Which CPUs/vendors are affected?
- Are smartphones/IoT devices affected?
- Can the vulnerabilities be exploited remotely?
- What data is at risk?
- How hard is it to exploit the vulnerabilities?

Vulnerability Assesment

A lot of confusion fueled the panic

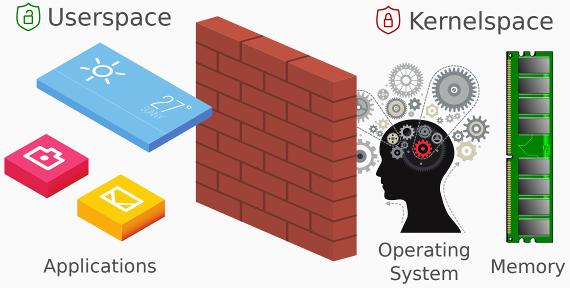
- Which CPUs/vendors are affected?
- Are smartphones/IoT devices affected?
- Can the vulnerabilities be exploited remotely?
- What data is at risk?
- How hard is it to exploit the vulnerabilities?
- Is it already exploited?

Let’s try to clarify these questions

MELTDOWN

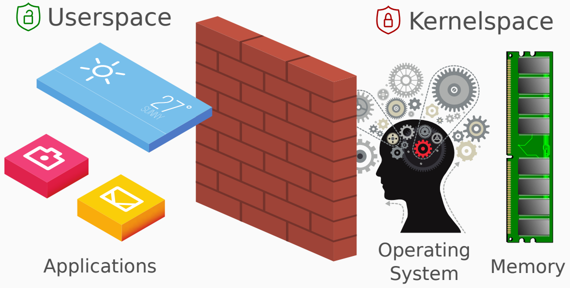
Hardware Isolation

- Kernel is isolated from user space



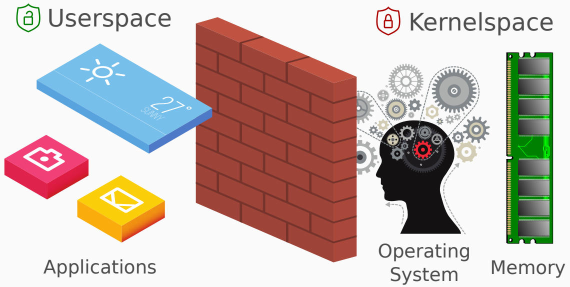
Hardware Isolation

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



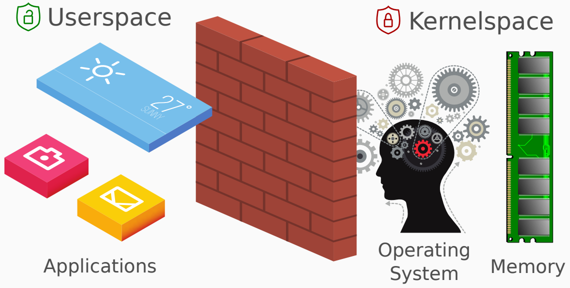
Hardware Isolation

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



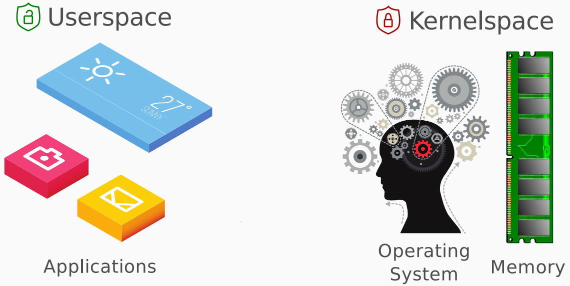
Hardware Isolation

- 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



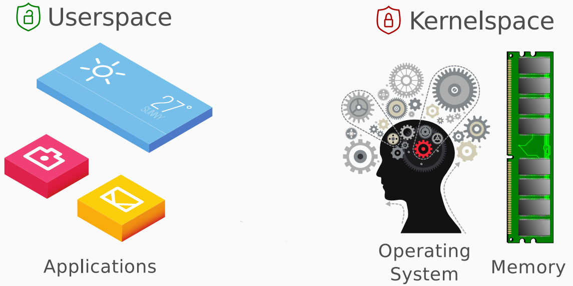
Meltdown Briefing

- Breaks isolation between applications and kernel



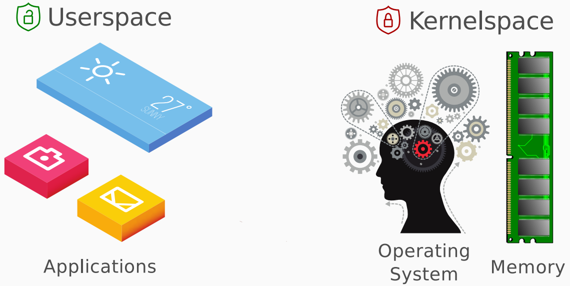
Meltdown Briefing

- Breaks isolation between applications and kernel
- User applications can access kernel addresses



Meltdown Briefing

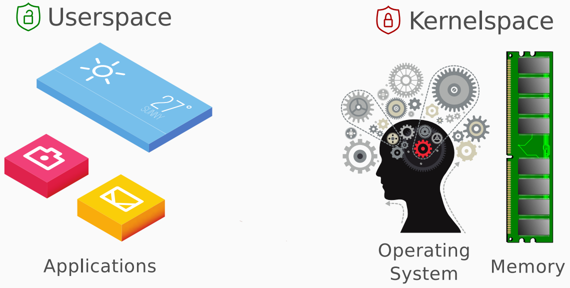
- Breaks isolation between applications and kernel
- User applications can access kernel addresses
- Entire physical memory is mapped in the kernel



Meltdown Briefing

- Breaks isolation between applications and kernel
- User applications can access kernel addresses
- Entire physical memory is mapped in the kernel

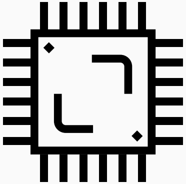
→ Meltdown can read whole DRAM



Meltdown Requirements

- Only on Intel CPUs and some unreleased ARMs (Cortex A75)

Meltdown Requirements



- Only on Intel CPUs and some unreleased ARMs (Cortex A75)
- AMD and other ARMs seem to be unaffected

Meltdown Requirements

- Only on Intel CPUs and some unreleased ARMs (Cortex A75)
- AMD and other ARMs seem to be unaffected
- Common cause: permission check done in parallel to load instruction

Meltdown Requirements

- Only on Intel CPUs and some unreleased ARMs (Cortex A75)
- AMD and other ARMs seem to be unaffected
- Common cause: permission check done in parallel to load instruction
- Race condition between permission check and dependent operation(s)

Meltdown Variant Requirements

- Meltdown variant: read privileged registers

Meltdown Variant Requirements

- Meltdown variant: read privileged registers
- Limited to some registers, no memory content

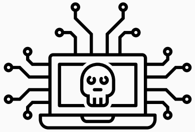
Meltdown Variant Requirements

- Meltdown variant: read privileged registers
- Limited to some registers, no memory content
- Reported by ARM

Meltdown Variant Requirements

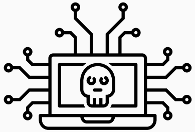
- Meltdown variant: read privileged registers
- Limited to some registers, no memory content
- Reported by ARM
- Affects some ARMs (Cortex A15, A57, and A72)

Meltdown Exploitability



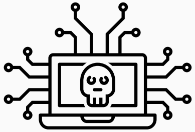
- Meltdown requires code execution on the device (e.g. Apps)

Meltdown Exploitability



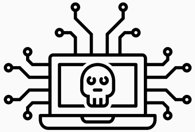
- Meltdown requires code execution on the device (e.g. Apps)
- Untrusted code can read entire memory of device

Meltdown Exploitability



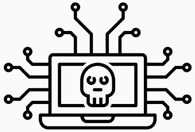
- Meltdown requires code execution on the device (e.g. Apps)
- Untrusted code can read entire memory of device
- Cannot be triggered remotely

Meltdown Exploitability



- Meltdown requires code execution on the device (e.g. Apps)
- Untrusted code can read entire memory of device
- Cannot be triggered remotely
- Proof-of-concept code available online

Meltdown Exploitability

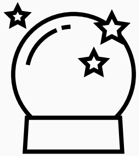


- Meltdown requires code execution on the device (e.g. Apps)
- Untrusted code can read entire memory of device
- Cannot be triggered remotely
- Proof-of-concept code available online
- No info about environment required → easy to reproduce

SPECTRE

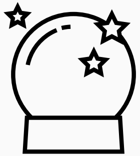
Spectre Briefing

- Mistrains branch prediction



Spectre Briefing

- Mistrains branch prediction
- CPU speculatively executes code which should not be executed



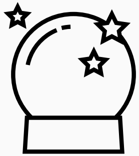
Spectre Briefing

- Mistrains branch prediction
- CPU speculatively executes code which should not be executed
- Can also mistrain indirect calls

Spectre Briefing

- Mistrains branch prediction
- CPU speculatively executes code which should not be executed
- Can also mistrain indirect calls

$\rightarrow$ Spectre "convinces" program to execute code



Spectre Requirements

- On Intel and AMD CPUs

Spectre Requirements

- On Intel and AMD CPUs
- Some ARMs (Cortex R and Cortex A) are also affected

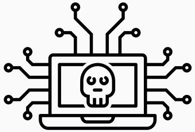
Spectre Requirements

- On Intel and AMD CPUs
- Some ARMs (Cortex R and Cortex A) are also affected
- Common cause: speculative execution of branches

Spectre Requirements

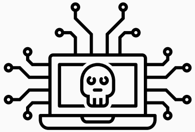
- On Intel and AMD CPUs
- Some ARMs (Cortex R and Cortex A) are also affected
- Common cause: speculative execution of branches
- Speculative execution leaves microarchitectural traces which leak secret

Spectre Exploitability



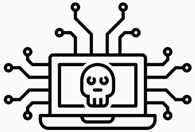
- Spectre requires code execution on the device (e.g. Apps)

Spectre Exploitability



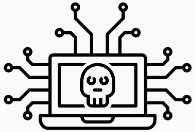
- Spectre requires code execution on the device (e.g. Apps)
- Untrusted code can convince trusted code to reveal secrets

Spectre Exploitability



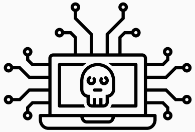
- Spectre requires code execution on the device (e.g. Apps)
- Untrusted code can convince trusted code to reveal secrets
- Can be triggered remotely (e.g. in the browser)

Spectre Exploitability



- Spectre requires code execution on the device (e.g. Apps)
- Untrusted code can convince trusted code to reveal secrets
- Can be triggered remotely (e.g. in the browser)
- Proof-of-concept code available online

Spectre Exploitability

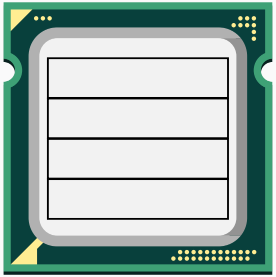


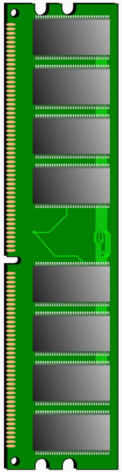
- Spectre requires code execution on the device (e.g. Apps)
- Untrusted code can convince trusted code to reveal secrets
- Can be triggered remotely (e.g. in the browser)
- Proof-of-concept code available online
- Info about environment required $\rightarrow$ hard to reproduce

Background

CPU Cache

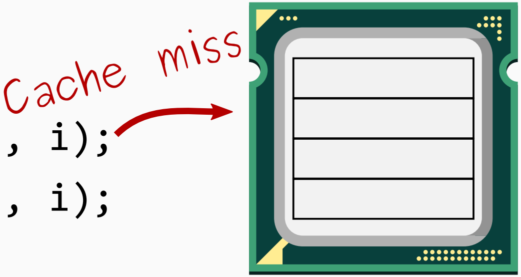
```
printf("%d", i);
printf("%d", i);
```

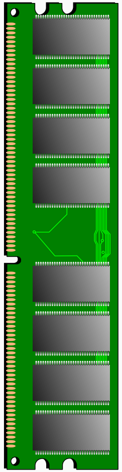




CPU Cache

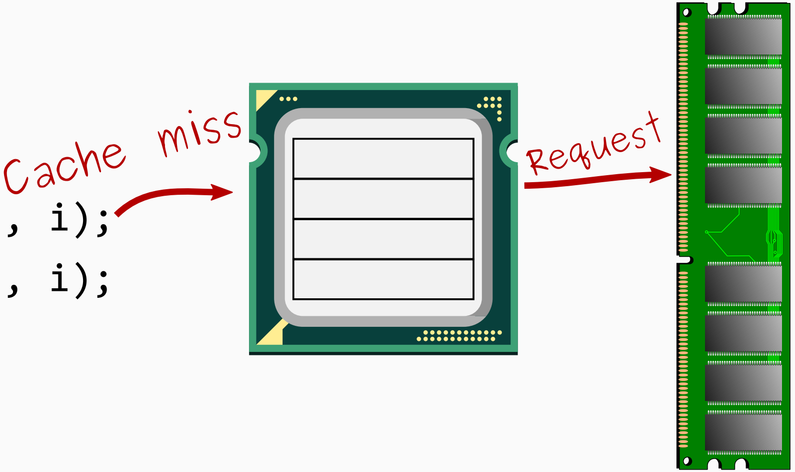
```
printf("%d", i);
printf("%d", i);
```





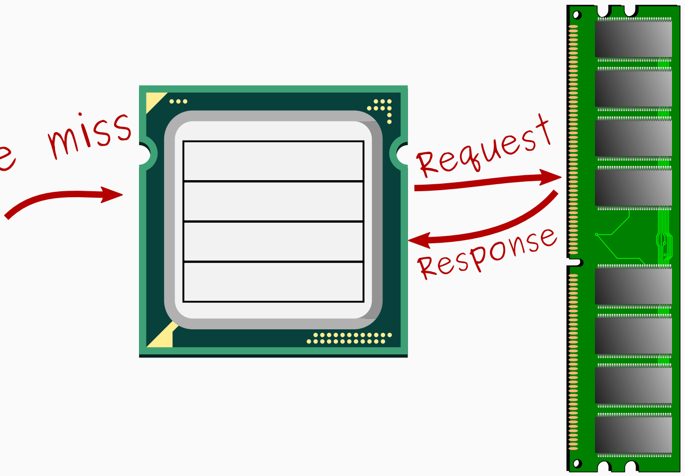
CPU Cache

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



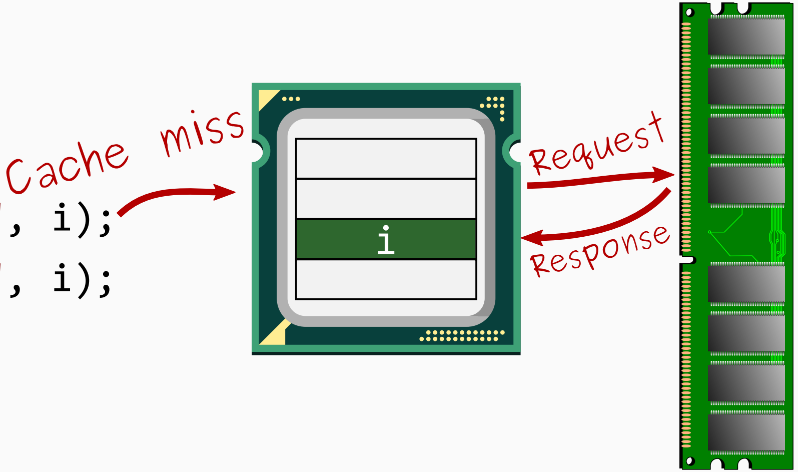
CPU Cache

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



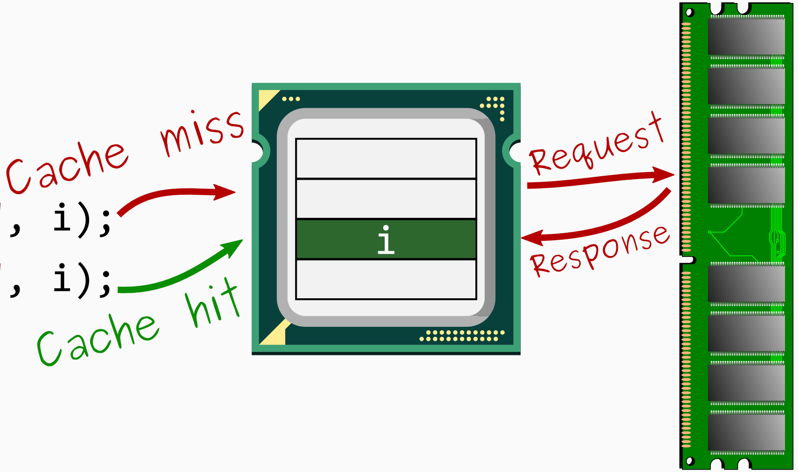
CPU Cache

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



CPU Cache

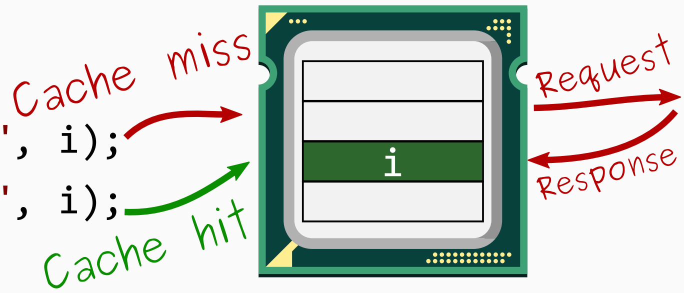
```
printf("%d", i);
printf("%d", i);
```

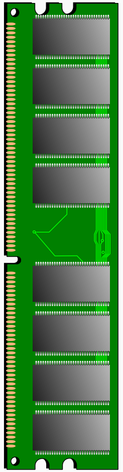


CPU Cache

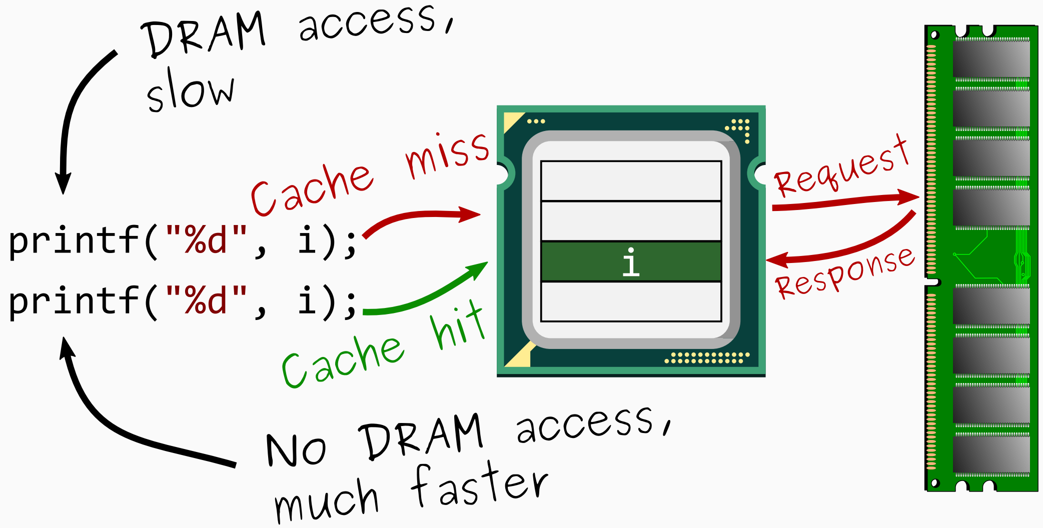
DRAM access, slow

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





CPU Cache



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

Flush+Reload

Shared Memory

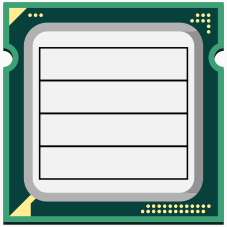
ATTACKER

flush

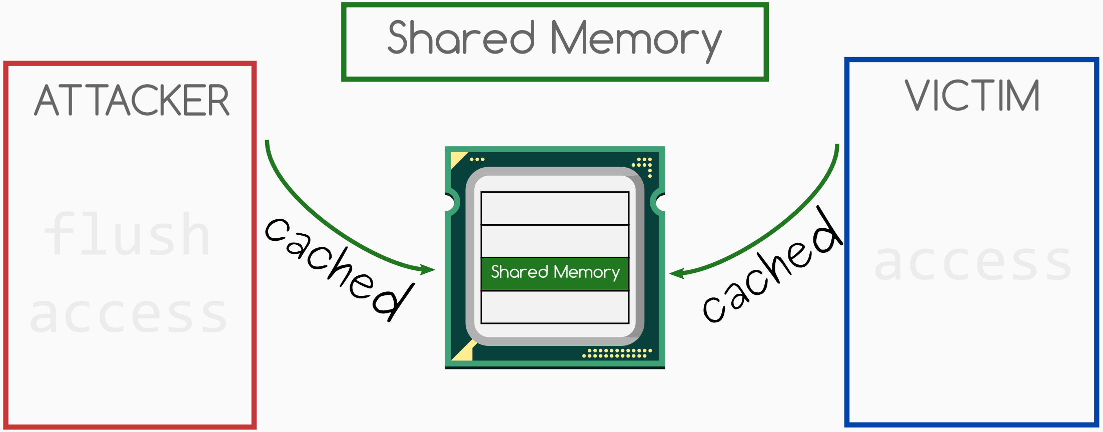
access

VICTIM

access



Flush+Reload



Flush+Reload

Shared Memory

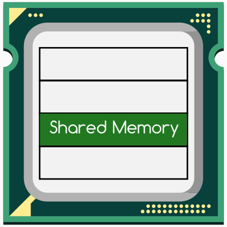
ATTACKER

flush

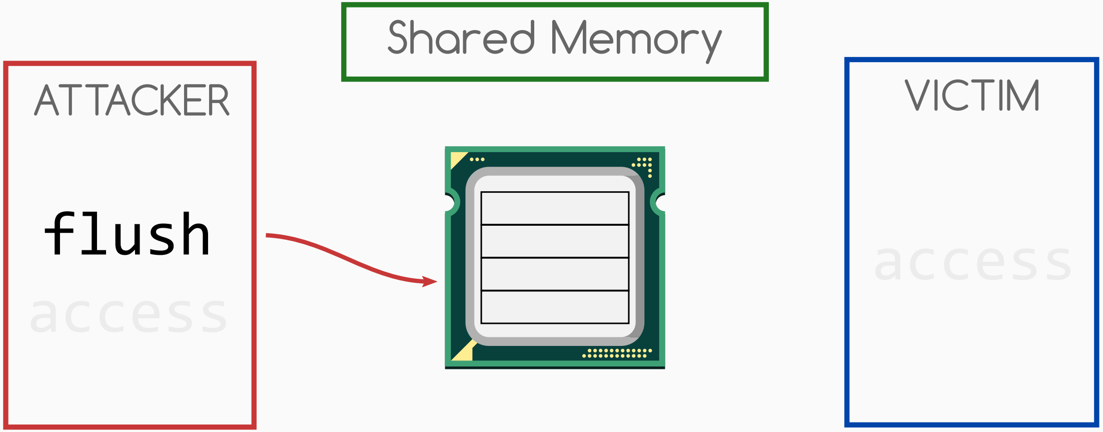
access

VICTIM

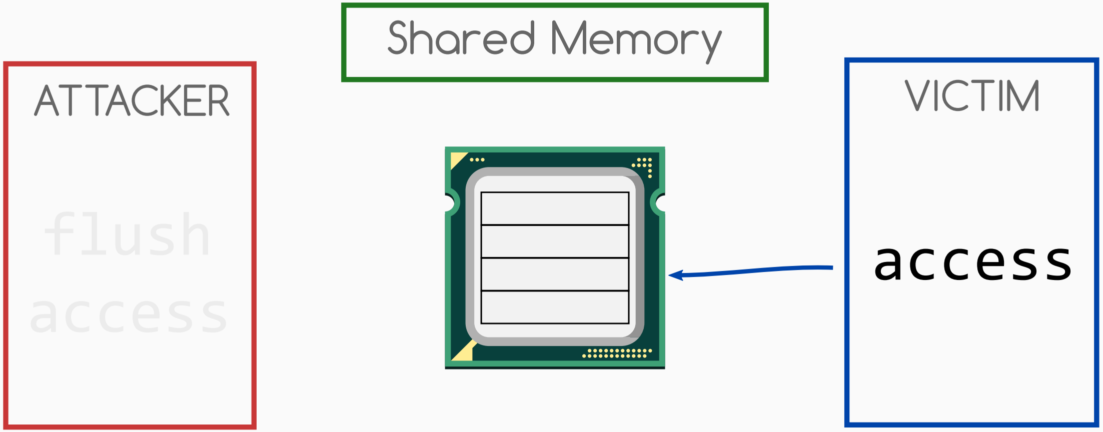
access



Flush+Reload



Flush+Reload



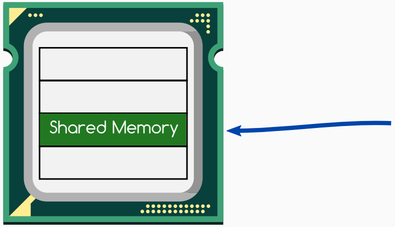
Flush+Reload

Shared Memory

ATTACKER

flush

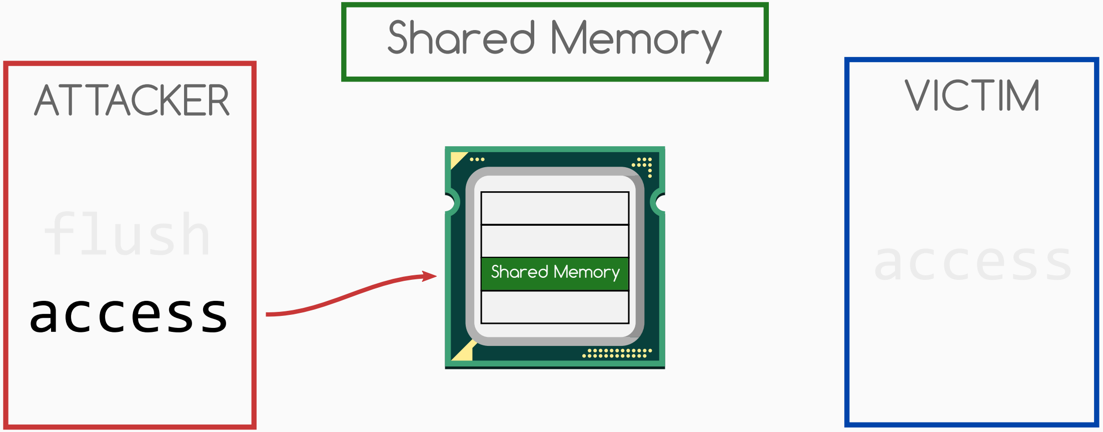
access



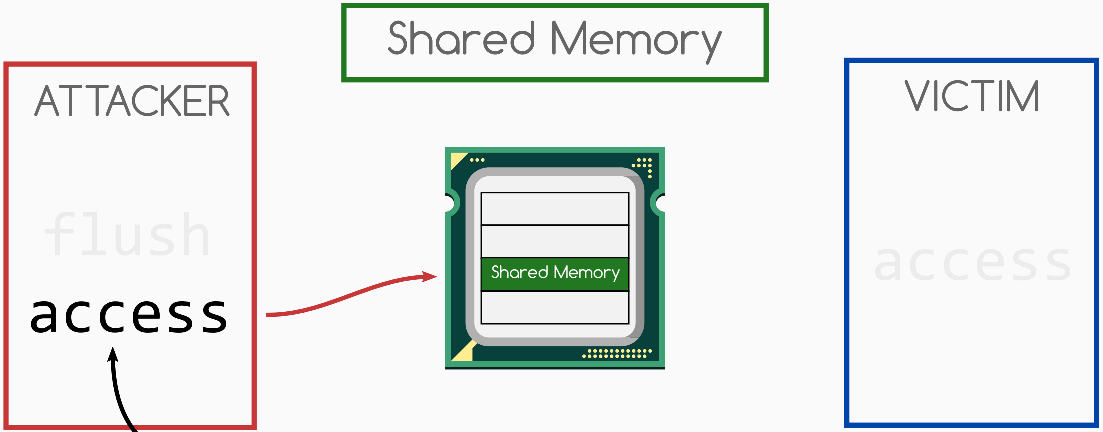
VICTIM

access

Flush+Reload



Flush+Reload



fast if victim accessed data,
slow otherwise

Out-of-order Execution

5. Cook everything until vegetables are soft

6. Add spices to taste [illegible] for 10 minutes

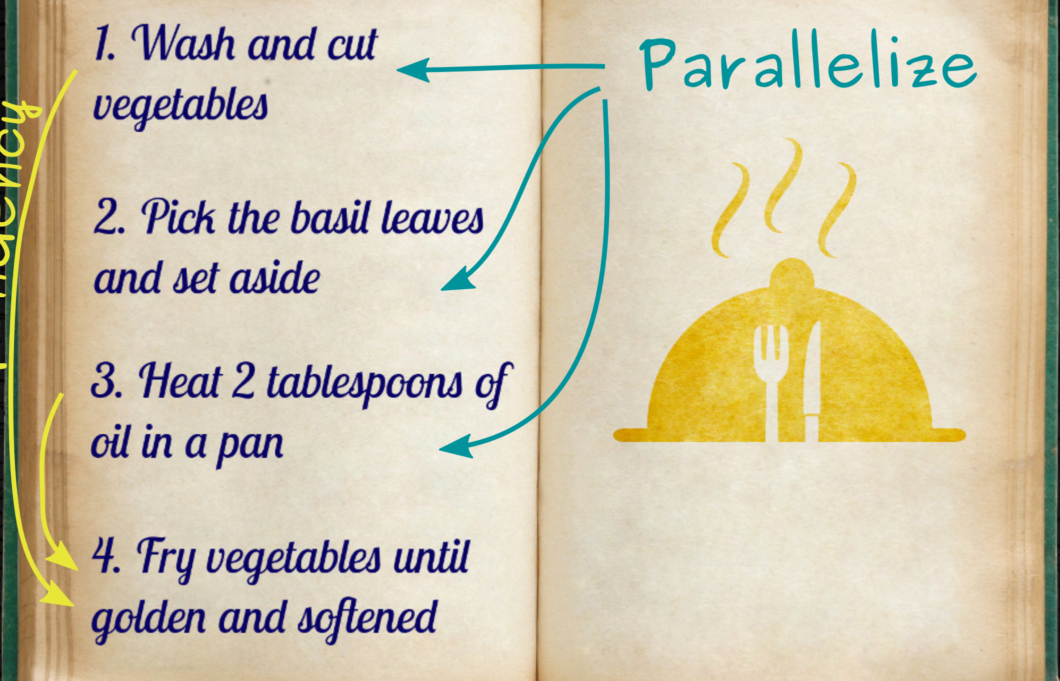
7. Serve with cooked and peeled potatoes

Wait for an hour

Wait for an hour

LATENCY

1. Wash and cut vegetables
2. Pick the basil leaves and set aside
3. Heat 2 tablespoons of oil in a pan
4. Fry vegetables until golden and softened

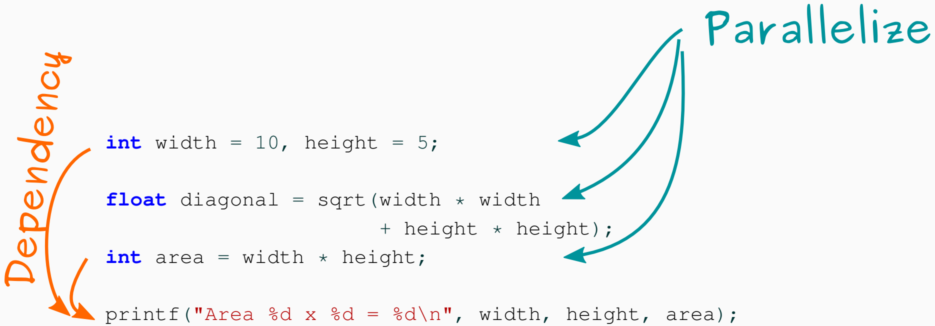


1. Wash and cut vegetables
2. Pick the basil leaves and set aside
3. Heat 2 tablespoons of oil in a pan
4. Fry vegetables until golden and softened

Out-of-order Execution

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

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

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

Out-of-order Execution



```
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);
```

We are ready for the gory details of Meltdown

Building Meltdown

```
char data = *(char*)0xffffffff81a000e0;
printf("%c\n", data);
```

Building Meltdown

```c
char data = *(char*)0xffffffff81a000e0;
printf("%c\n", data);
```

```
segfault at ffffffff81a000e0 ip 0000000000400535
            sp 00007ffce4a80610 error 5 in reader
```

Building Meltdown

```c
char data = *(char*)0xffffffff81a000e0;
printf("%c\n", data);
```

```
segfault at ffffffff81a000e0 ip 0000000000400535
          sp 00007ffce4a80610 error 5 in reader
```

- Kernel addresses are not accessible

Building Meltdown

```
char data = *(char*)0xffffffff81a000e0;
printf("%c\n", data);
```

```
segfault at ffffffff81a000e0 ip 0000000000400535
            sp 00007ffce4a80610 error 5 in reader
```

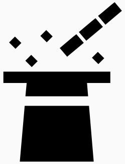
- Kernel addresses are not accessible
- Are privilege checks also done when executing instructions out of order?

Building Meltdown

- Adapted code

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

Building Meltdown



- Adapted code

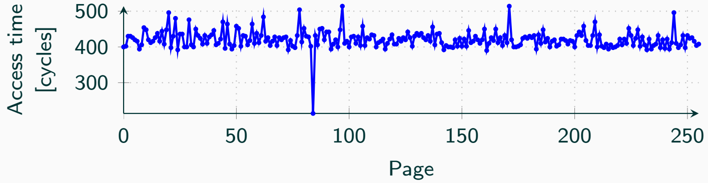
```
*(volatile char*)0;
array[84 * 4096] = 0; // unreachable
```

- Static code analyzer is not happy

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

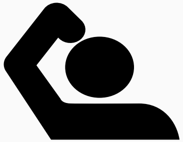
Building Meltdown

- Flush+Reload over all pages of the array

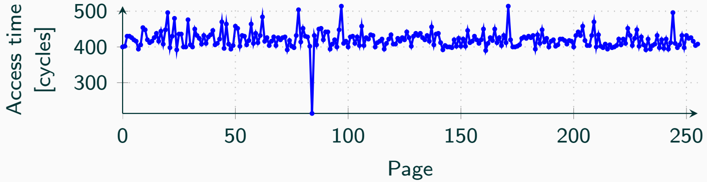


- “Unreachable” code line was actually executed

Building Meltdown



- Flush+Reload over all pages of the array



- “Unreachable” code line was actually executed
- Exception was only thrown afterwards

Building Meltdown

- Out-of-order instructions leave microarchitectural traces

Building Meltdown

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

Building Meltdown

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

Building Meltdown

- 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

Building Meltdown

- Combine the two things

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

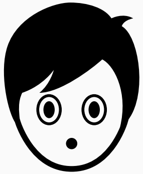
Building Meltdown

- Combine the two things

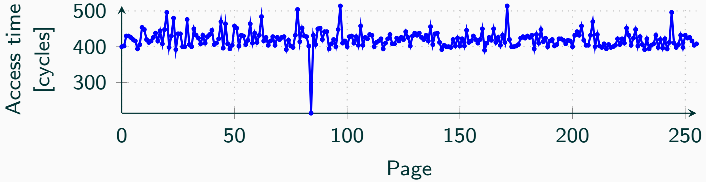
```
char data = *(char*)0xffffffff81a000e0;
array[data * 4096] = 0;
```

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

Building Meltdown

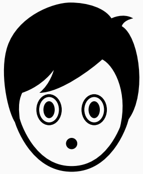


- Flush+Reload over all pages of the array

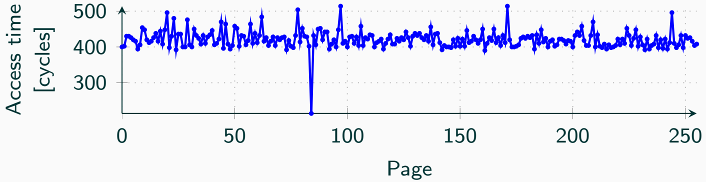


- Index of cache hit reveals data

Building Meltdown

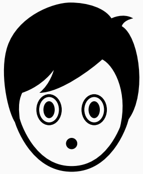


- Flush+Reload over all pages of the array



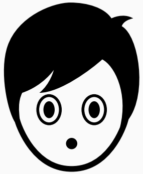
- Index of cache hit reveals data
- Permission check is in some cases not fast enough

Building Meltdown



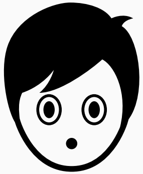
- Using out-of-order execution, we can read data at any address

Building Meltdown



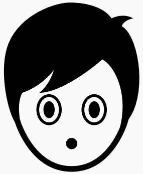
- Using out-of-order execution, we can read data at any address
- Privilege checks are sometimes too slow

Building Meltdown

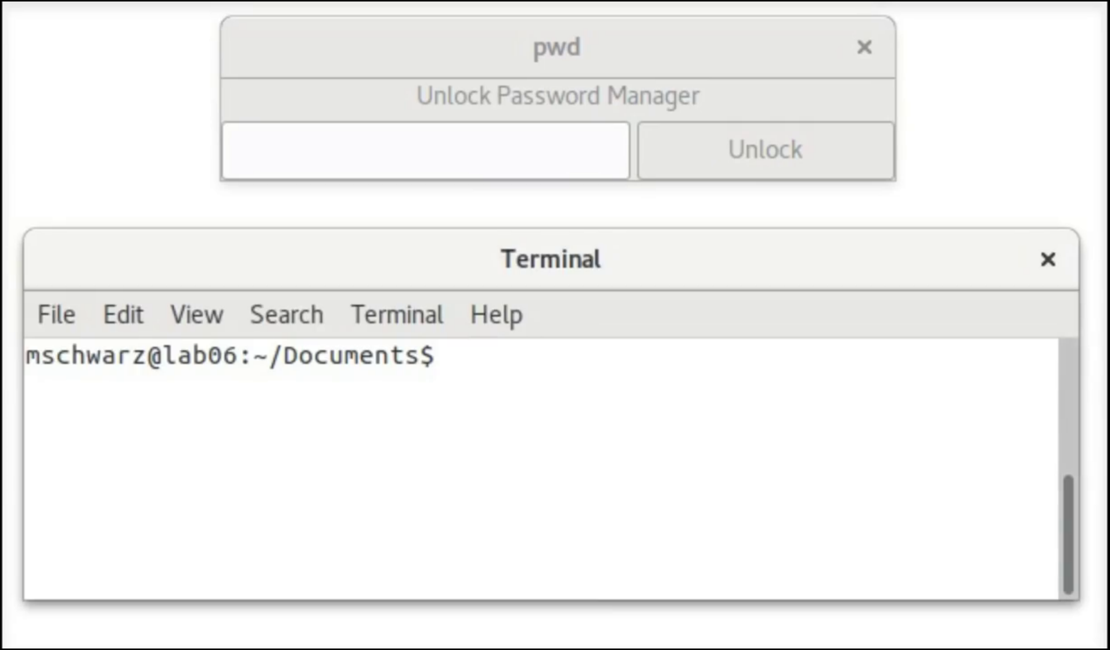


- Using out-of-order execution, we can read data at any address
- Privilege checks are sometimes too slow
- Allows to leak kernel memory

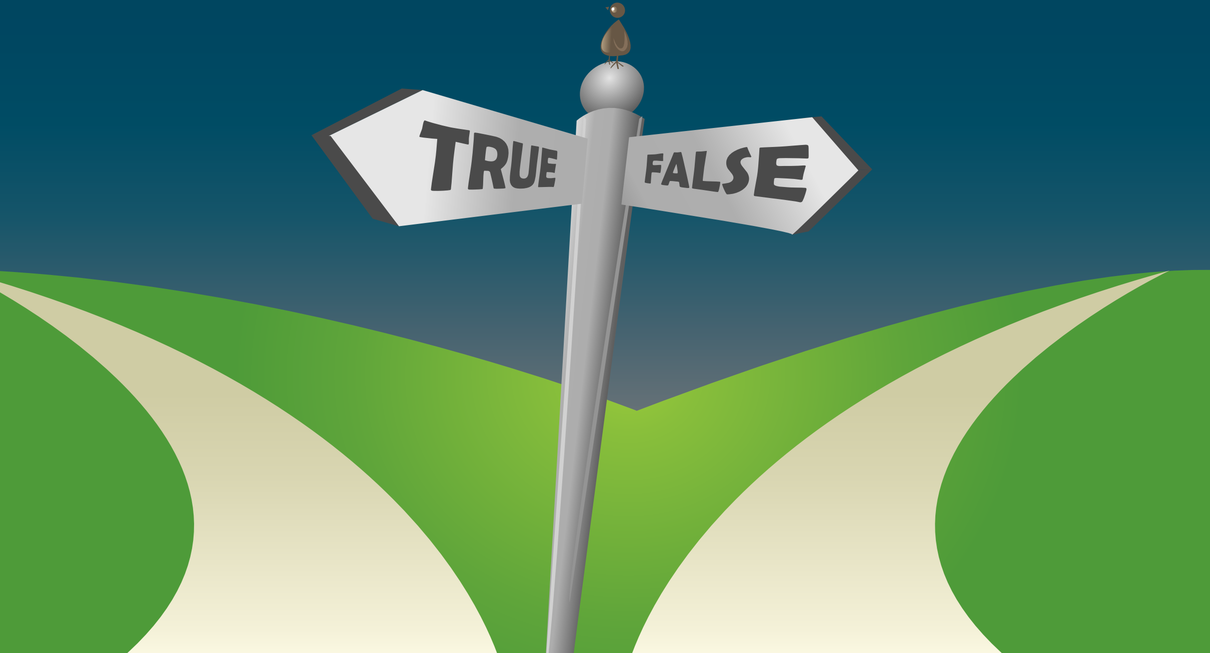
Building Meltdown



- Using out-of-order execution, we can read data at any address
- Privilege checks are sometimes too slow
- Allows to leak kernel memory
- Entire physical memory is typically also accessible in kernel address space



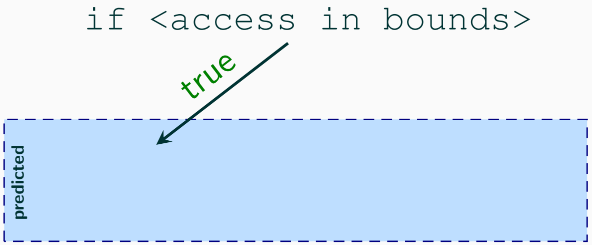
```
mschwarz@lab06:~/Documents$
```


TRUE
FALSE

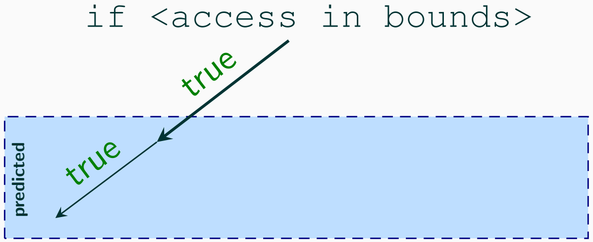
Spectre

```
if <access in bounds>
```

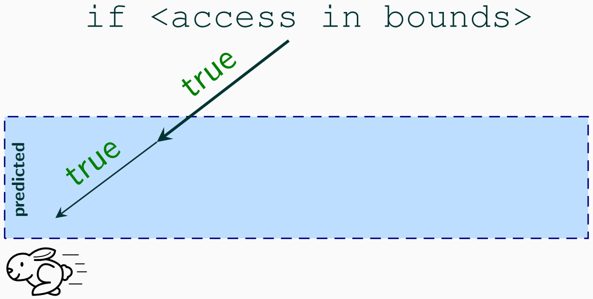
Spectre



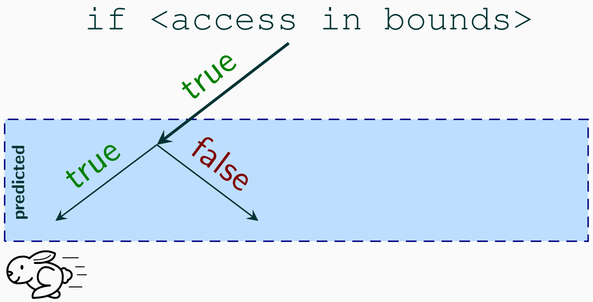
Spectre



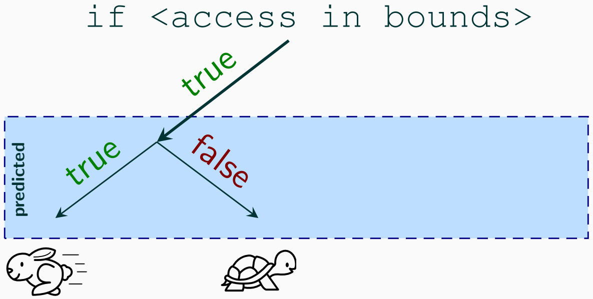
Spectre



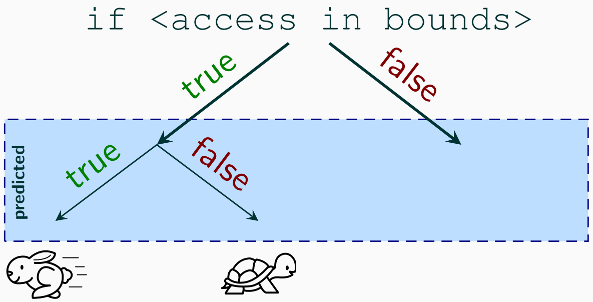
Spectre



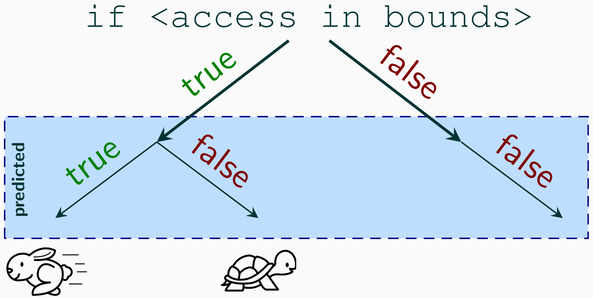
Spectre



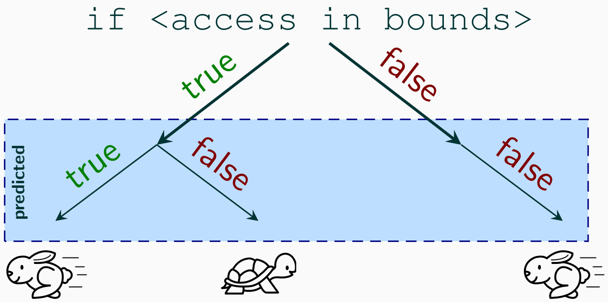
Spectre



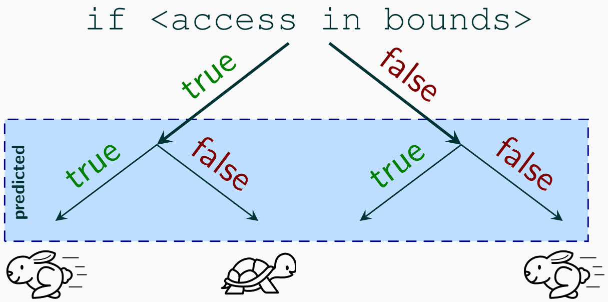
Spectre



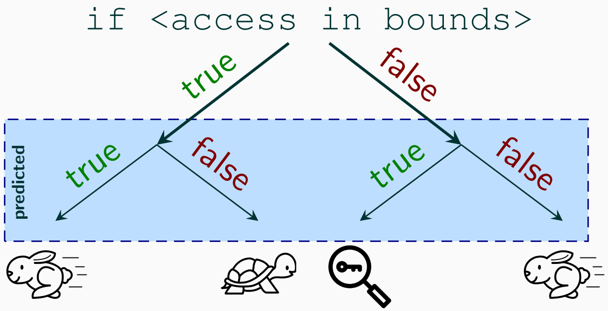
Spectre



Spectre



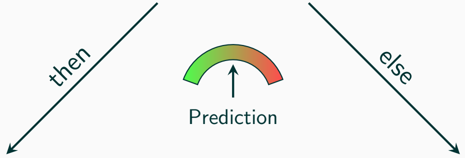
Spectre



We are ready for the gory details of Spectre

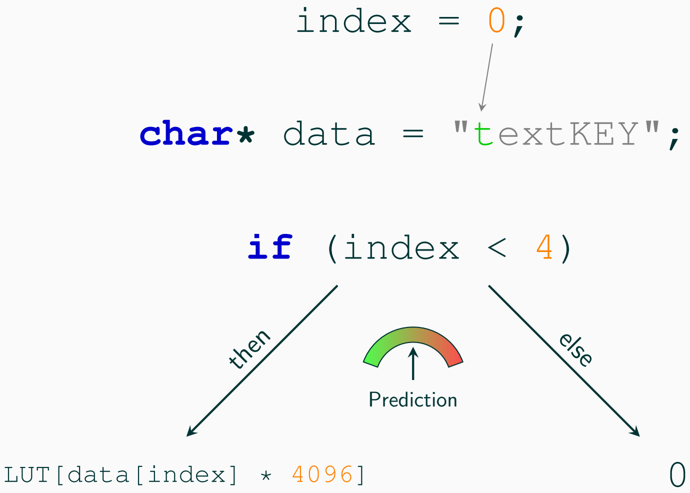
Spectre (variant 1)

```
index = 0;

char* data = "textKEY";

if (index < 4)
```



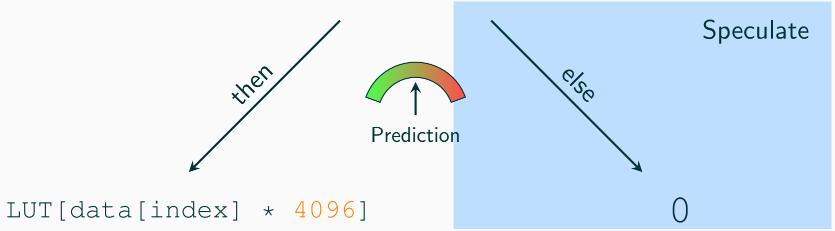
```
LUT[data[index] * 4096]                    0
```

Spectre (variant 1)

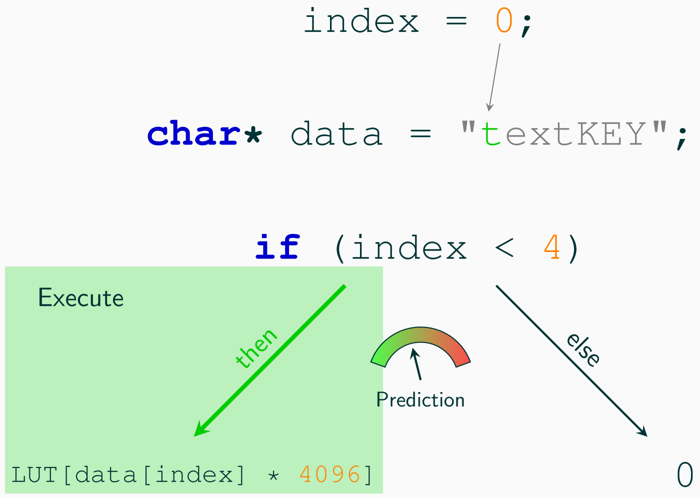


Spectre (variant 1)

```
index = 0;

char* data = "textKEY";

if (index < 4)
```



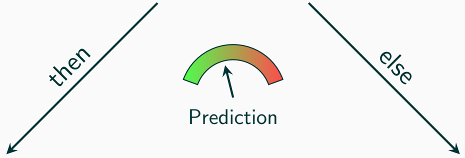
Spectre (variant 1)



```
index = 0;

char* data = "textKEY";

if (index < 4)

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

Spectre (variant 1)

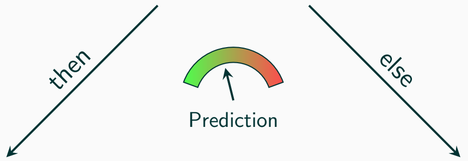
```
index = 1;

char* data = "textKEY";

if (index < 4)
```



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

Spectre (variant 1)

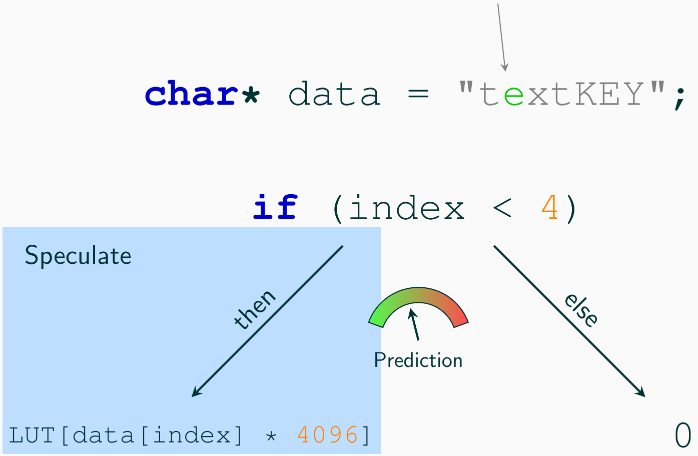
```
index = 1;

char* data = "textKEY";

if (index < 4)
```



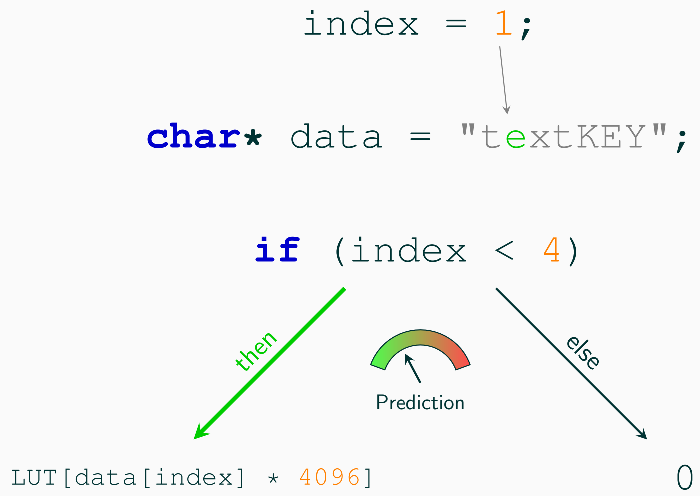
```
LUT[data[index] * 4096]                0
```

Spectre (variant 1)

```
index = 1;

char* data = "textKEY";

if (index < 4)
```



Spectre (variant 1)



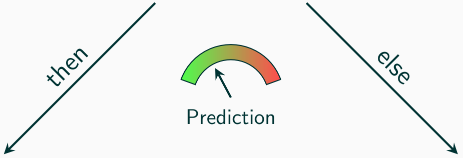
```
index = 1;

char* data = "textKEY";

if (index < 4)
```

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

0

Spectre (variant 1)

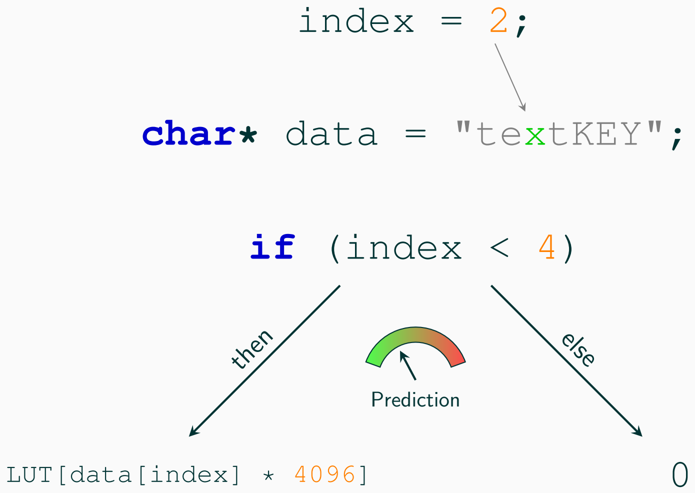
```
index = 2;

char* data = "textKEY";

if (index < 4)
```



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

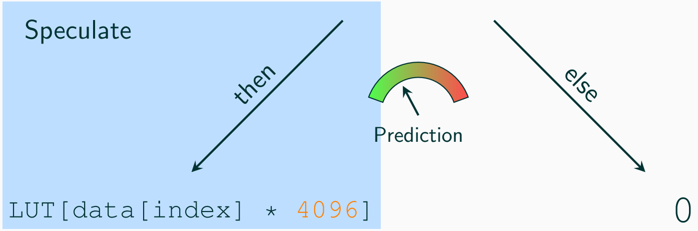
0

Spectre (variant 1)

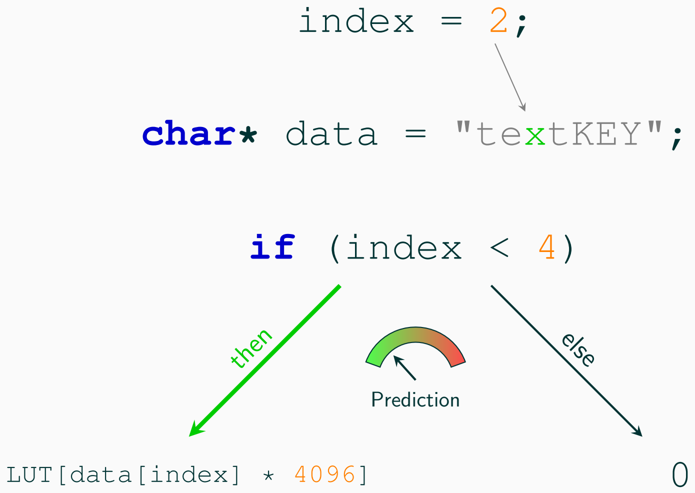


Spectre (variant 1)

```
index = 2;

char* data = "textKEY";

if (index < 4)
```



Spectre (variant 1)



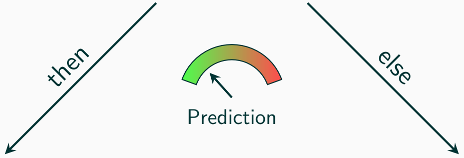
```
index = 2;

char* data = "textKEY";

if (index < 4)
```

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

```
0
```

Spectre (variant 1)

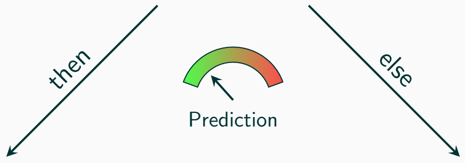
```
index = 3;

char* data = "textKEY";

if (index < 4)
```



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

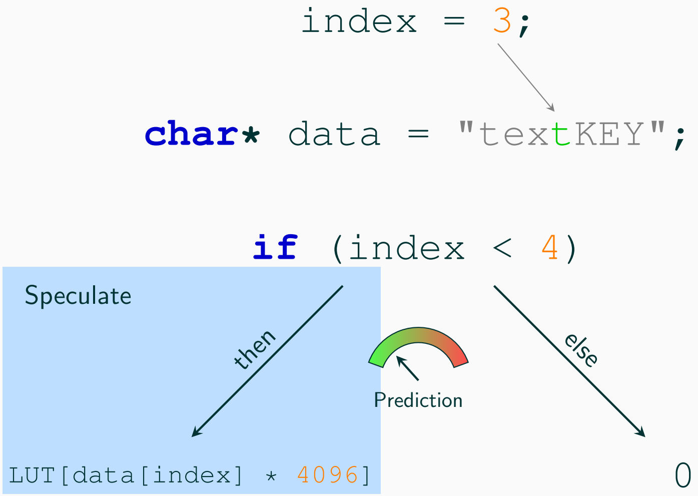
Spectre (variant 1)

```
index = 3;

char* data = "textKEY";

if (index < 4)
```

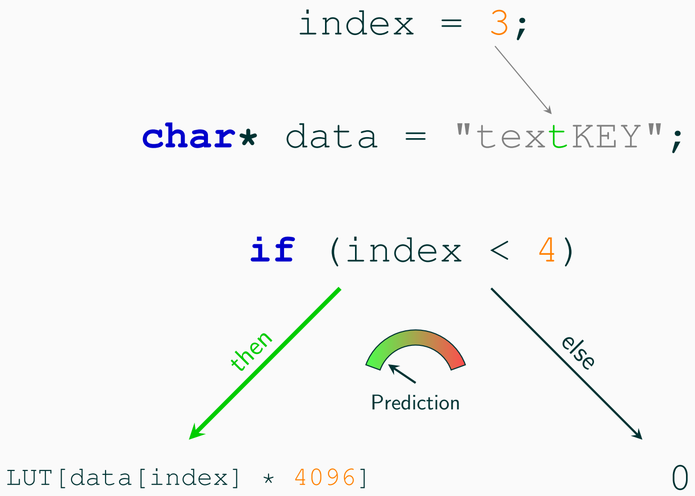


```
LUT[data[index] * 4096]                0
```

Spectre (variant 1)



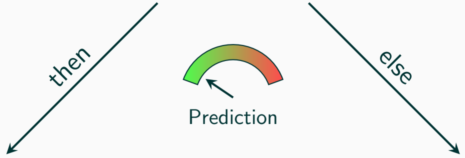
Spectre (variant 1)



```
index = 3;

char* data = "textKEY";

if (index < 4)

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

Spectre (variant 1)

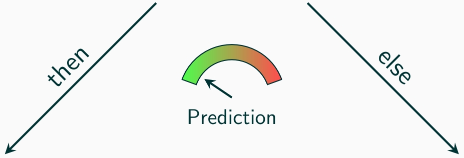
```
index = 4;

char* data = "textKEY";

if (index < 4)
```



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

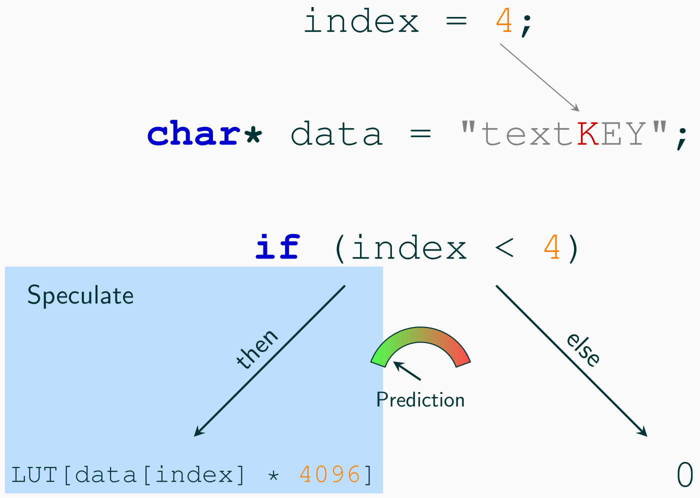
Spectre (variant 1)

```
index = 4;

char* data = "textKEY";

if (index < 4)
```



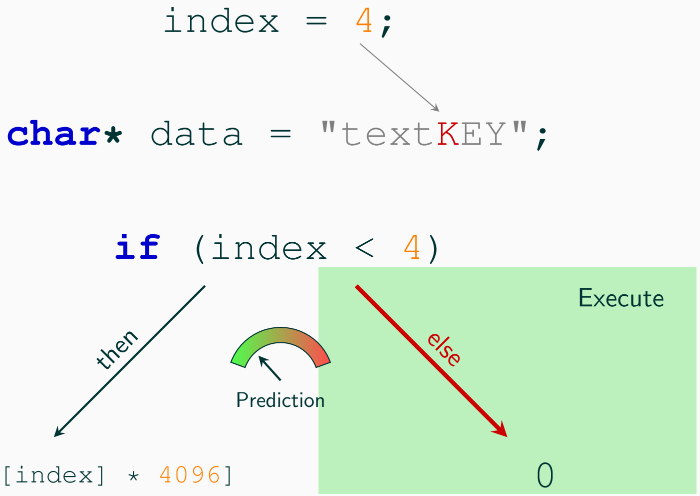
```
LUT[data[index] * 4096]                 0
```

Spectre (variant 1)



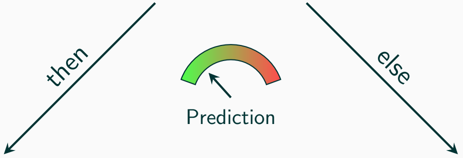
Spectre (variant 1)

```
index = 4;

char* data = "textKEY";

if (index < 4)
```



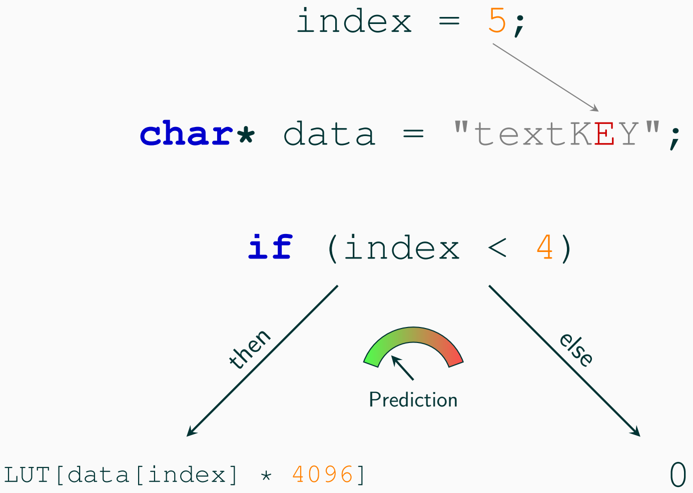
Spectre (variant 1)

```
index = 5;

char* data = "textKEY";

if (index < 4)
```

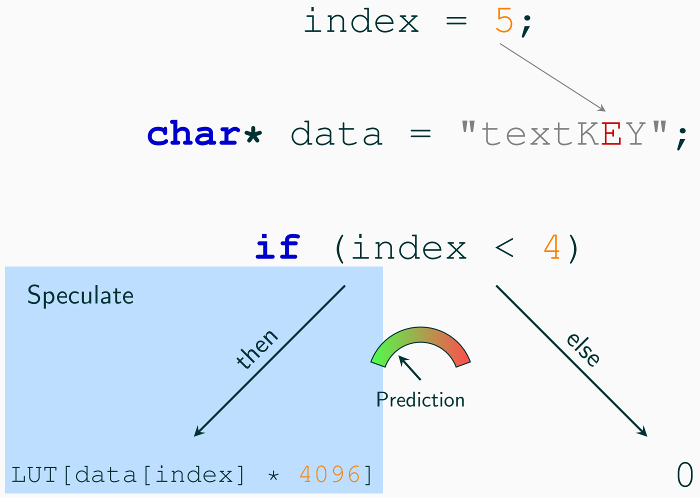


```
LUT[data[index] * 4096]                0
```

Spectre (variant 1)



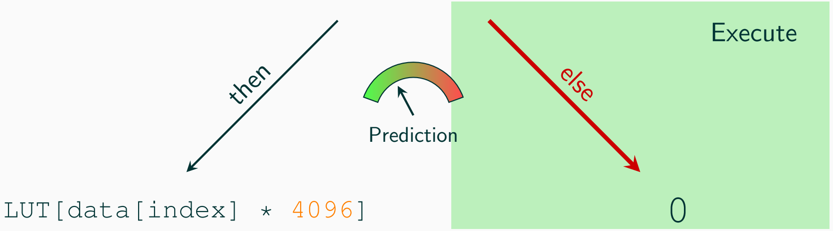
Spectre (variant 1)



```
index = 5;

char* data = "textKEY";

if (index < 4)
```

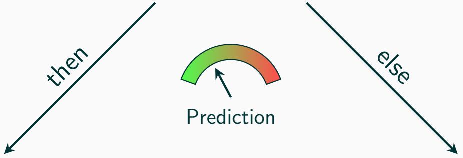
Spectre (variant 1)

```
index = 5;

char* data = "textKEY";

if (index < 4)
```



Spectre (variant 1)

```
index = 6;

char* data = "textKEY";

if (index < 4)
```

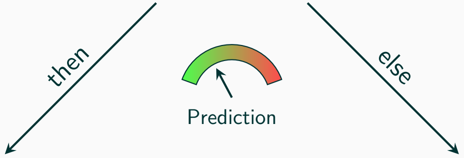


```
LUT[data[index] * 4096]
```

0

Spectre (variant 1)

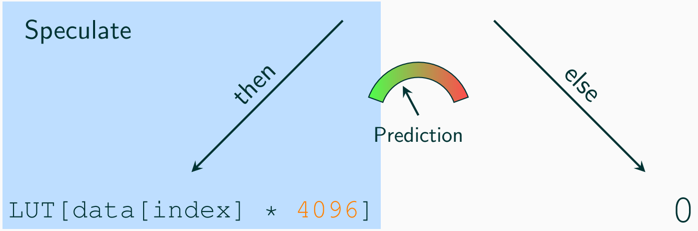
```
index = 6;

char* data = "textKEY";

if (index < 4)
```



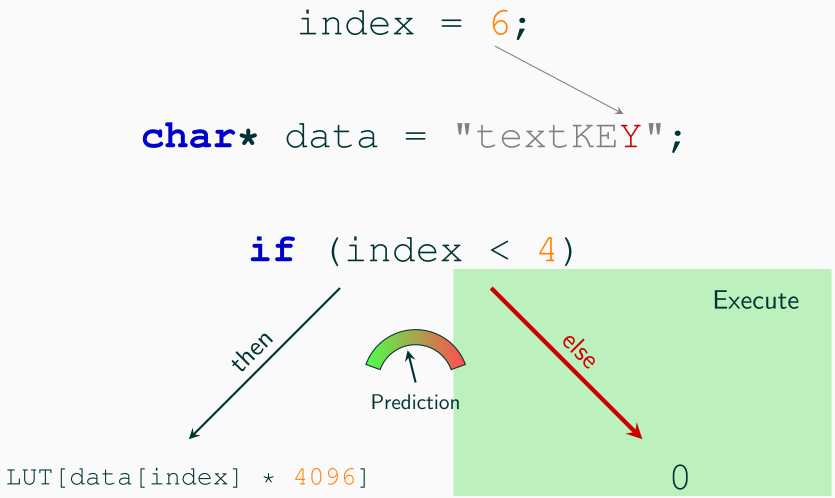
```
LUT[data[index] * 4096]                    0
```

Spectre (variant 1)

```
index = 6;

char* data = "textKEY";

if (index < 4)
```



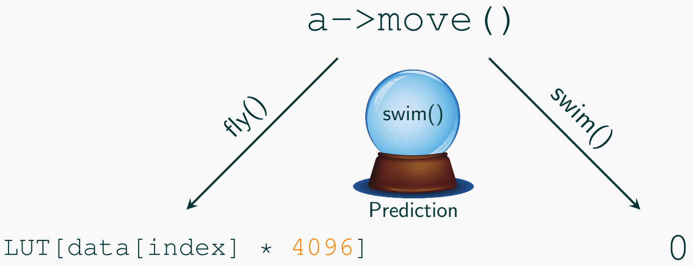
Spectre (variant 1)



```
index = 6;

char* data = "textKEY";

if (index < 4)

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

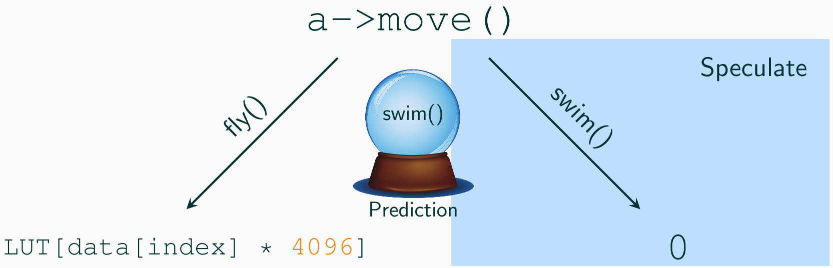
Spectre (variant 2)

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



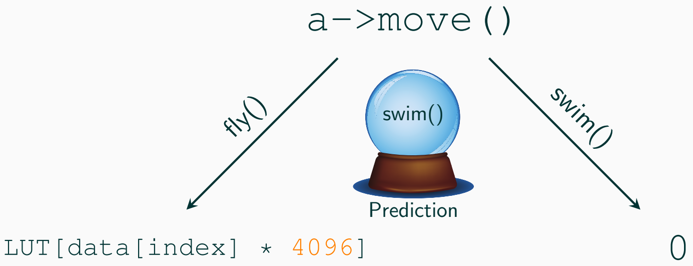
Spectre (variant 2)

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



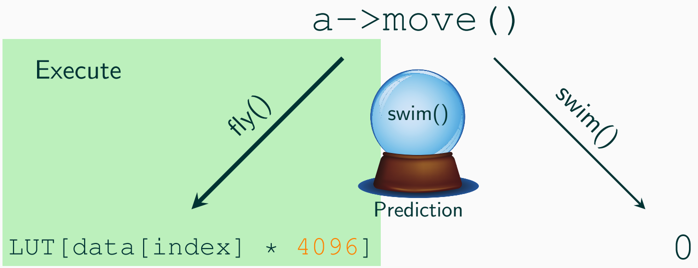
Spectre (variant 2)

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



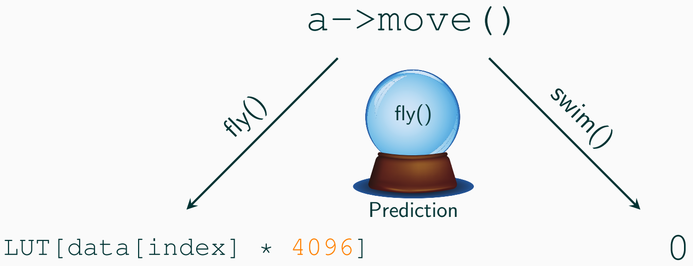
Spectre (variant 2)

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



Spectre (variant 2)

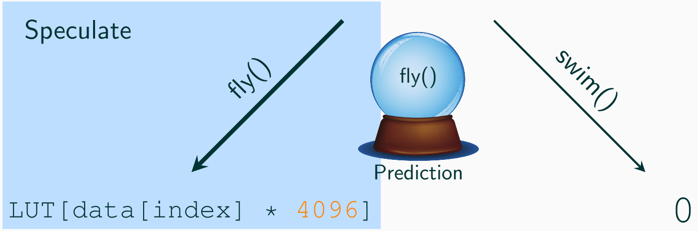
```
Animal* a = bird;
```



Spectre (variant 2)

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

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

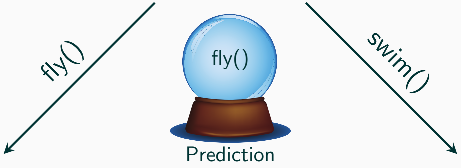


```
LUT[data[index] * 4096]
```

Spectre (variant 2)

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

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



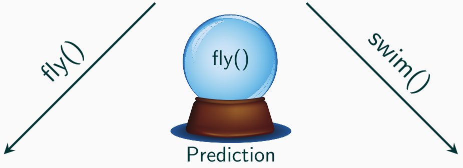
```
LUT[data[index] * 4096]
```

0

Spectre (variant 2)

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

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

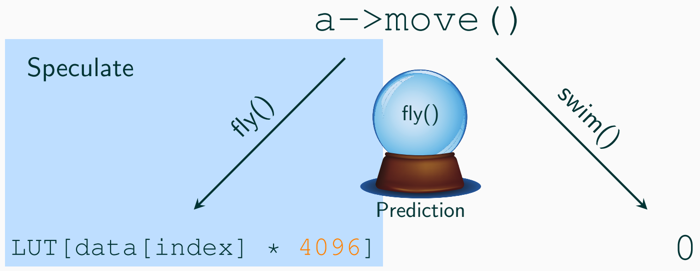


```
LUT[data[index] * 4096]
```

0

Spectre (variant 2)

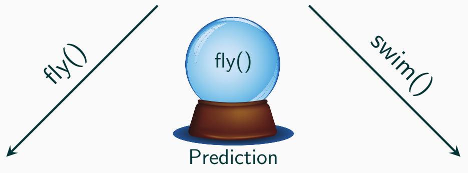
```
Animal* a = fish;
```



Spectre (variant 2)

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

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

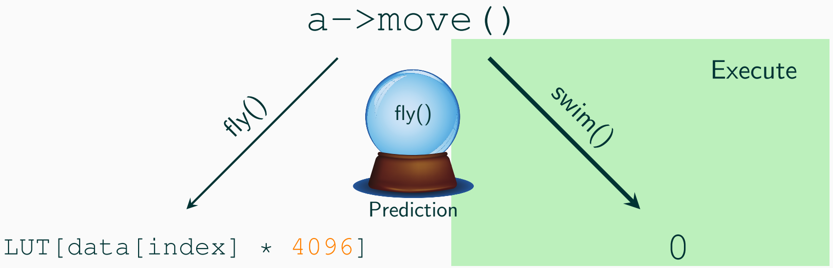


```
LUT[data[index] * 4096]
```

0

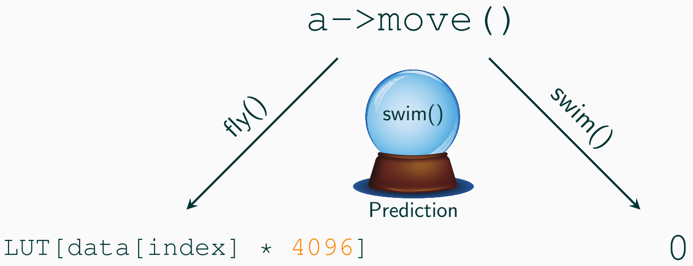
Spectre (variant 2)

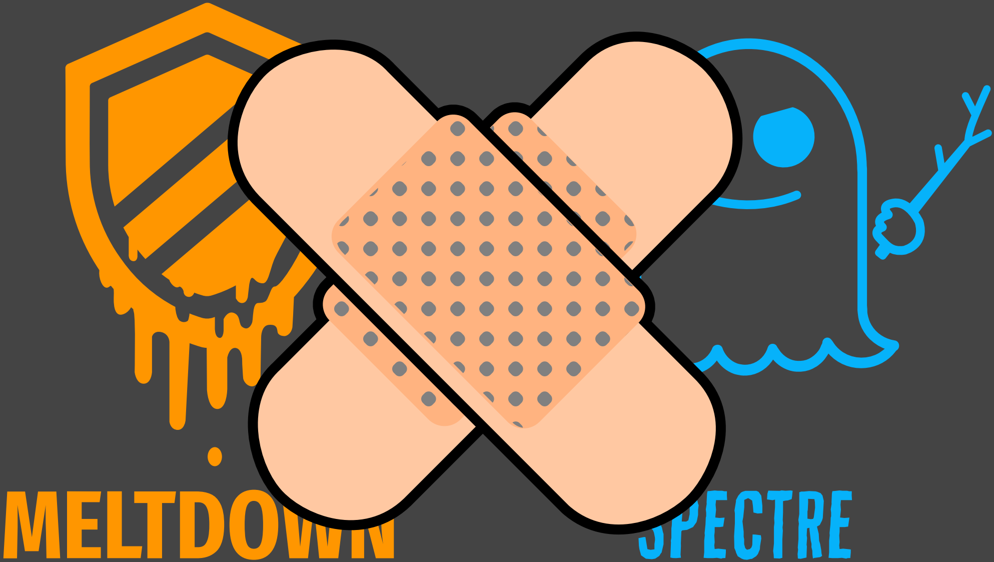
```
Animal* a = fish;
```



Spectre (variant 2)

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




MELTDOWN
SPECTRE

Idea

- Idea: unmap the kernel in user space

Idea

- Idea: unmap the kernel in user space
- Kernel addresses are then no longer present

Idea

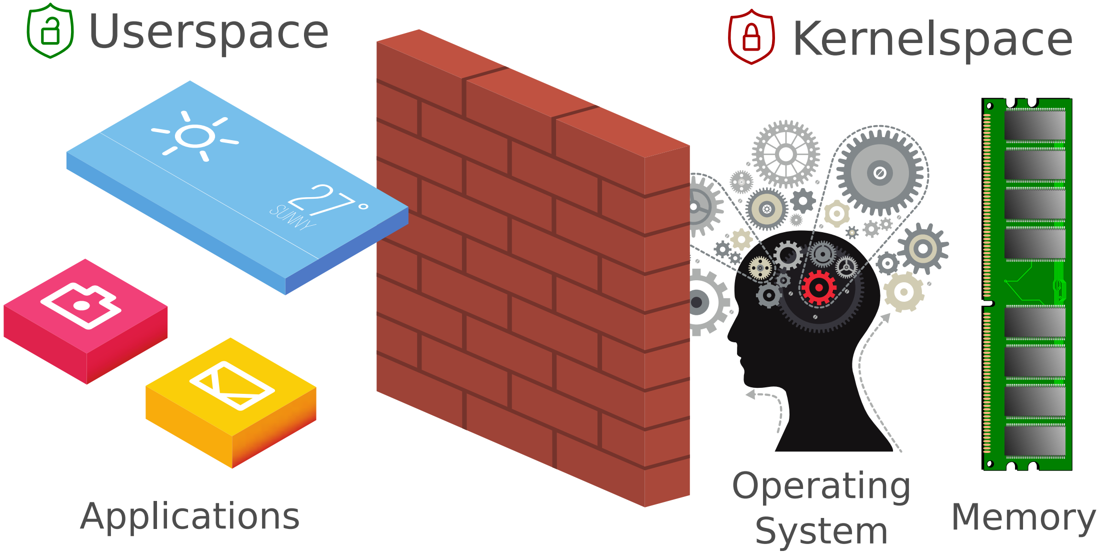
- Idea: unmap the kernel in user space
- Kernel addresses are then no longer present
- Memory which is not mapped cannot be accessed at all

Kernel **A**ddress **I**solation to have **S**ide channels **E**fficiently **R**emoved

KAISER /ˈkʌɪzə/

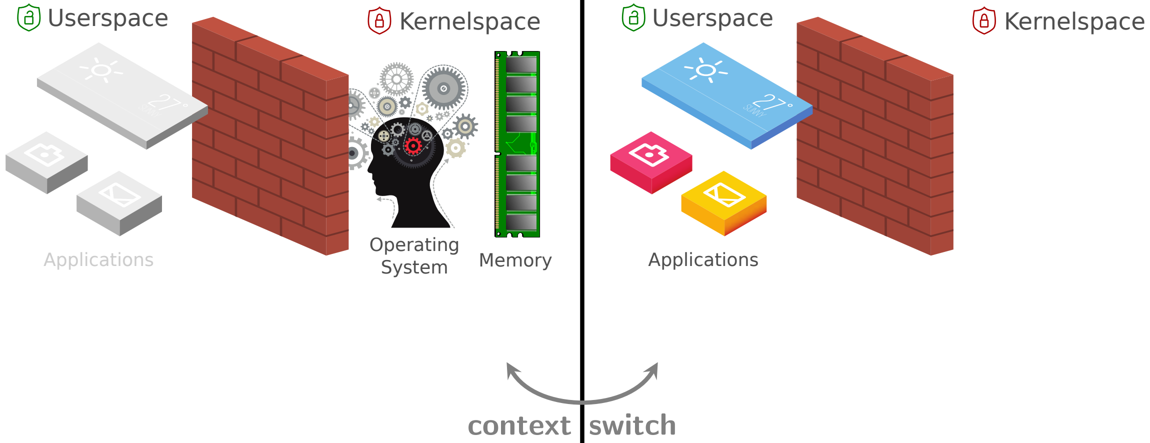
1. [german] Emperor, ruler of an empire
2. largest penguin, emperor penguin

Kernel **A**ddress **I**solation to have **S**ide channels **E**fficiently **R**emoved

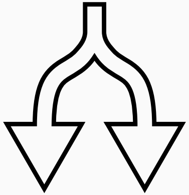

Userspace
Kernelspace
27°
SUNNY
Applications
Operating System
Memory

Kernel View

User View

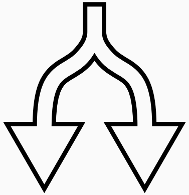


Kernel Address Space Isolation



- We published KAISER in July 2017

Kernel Address Space Isolation



- We published KAISER in July 2017
- Intel and others improved and merged it into Linux as KPTI (Kernel Page Table Isolation)

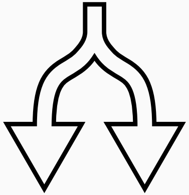
Kernel Address Space Isolation

- We published KAISER in July 2017
- Intel and others improved and merged it into Linux as KPTI (Kernel Page Table Isolation)
- Microsoft implemented similar concept in Windows 10

Kernel Address Space Isolation

- We published KAISER in July 2017
- Intel and others improved and merged it into Linux as KPTI (Kernel Page Table Isolation)
- Microsoft implemented similar concept in Windows 10
- Apple implemented it in macOS 10.13.2 and called it "Double Map"

Kernel Address Space Isolation



- We published KAISER in July 2017
- Intel and others improved and merged it into Linux as KPTI (Kernel Page Table Isolation)
- Microsoft implemented similar concept in Windows 10
- Apple implemented it in macOS 10.13.2 and called it “Double Map”
- All share the same idea: switching address spaces on context switch

Performance

- Depends on how often you need to switch between kernel and user space

Performance

- Depends on how often you need to switch between kernel and user space
- Can be slow, 40% or more on old hardware

Performance

- Depends on how often you need to switch between kernel and user space
- Can be slow, 40% or more on old hardware
- But modern CPUs have additional features

Performance

- Depends on how often you need to switch between kernel and user space
- Can be slow, 40% or more on old hardware
- But modern CPUs have additional features
- ⇒ Performance overhead on average below 2%

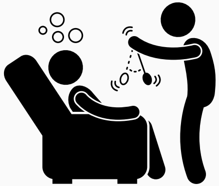
Meltdown and Spectre

MELTDOWN

SPECTRE

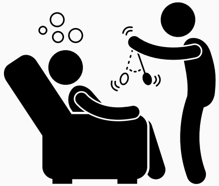
~~Meltdown and~~ Spectre

Spectre



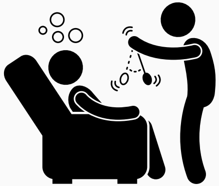
- Does not directly access kernel

Spectre



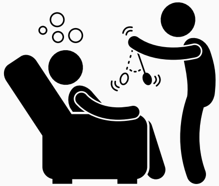
- Does not directly access kernel
- “Convinces” other programs to reveal their secrets

Spectre



- Does not directly access kernel
- “Convinces” other programs to reveal their secrets
- Much harder to fix, KAISER does not help

Spectre



- Does not directly access kernel
- "Convinces" other programs to reveal their secrets
- Much harder to fix, KAISER does not help
- Ongoing effort to patch via microcode update and compiler extensions

Spectre

- Trivial approach: disable speculative execution

Spectre

- Trivial approach: disable speculative execution
- No wrong speculation if there is no speculation

Spectre

- Trivial approach: disable speculative execution
- No wrong speculation if there is no speculation
- Problem: massive performance hit!

Spectre

- Trivial approach: disable speculative execution
- No wrong speculation if there is no speculation
- Problem: massive performance hit!
- Also: How to disable it?

Spectre

- Trivial approach: disable speculative execution
- No wrong speculation if there is no speculation
- Problem: massive performance hit!
- Also: How to disable it?
- Speculative execution is deeply integrated into CPU

Spectre Variant 1 Mitigations

Spectre Variant 1 Mitigations

- Workaround: insert instructions stopping speculation

Spectre Variant 1 Mitigations

- Workaround: insert instructions stopping speculation

→ insert after every bounds check

Spectre Variant 1 Mitigations

- Workaround: insert instructions stopping speculation

→ insert after every bounds check

- x86: `LFENCE`, ARM: `CSDB`

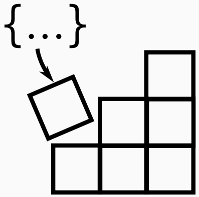
Spectre Variant 1 Mitigations

- Workaround: insert instructions stopping speculation

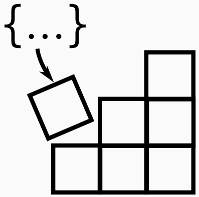
→ insert after every bounds check

- x86: `LFENCE`, ARM: `CSDB`
- Available on all Intel CPUs, retrofitted to existing ARMv7 and ARMv8

Spectre Variant 1 Mitigations

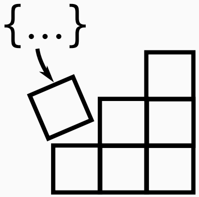


Spectre Variant 1 Mitigations



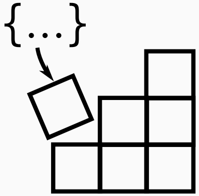
- Speculation barrier requires compiler supported

Spectre Variant 1 Mitigations



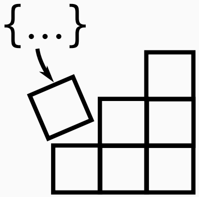
- Speculation barrier requires compiler supported
- Already implemented in GCC, LLVM, and MSVC

Spectre Variant 1 Mitigations



- Speculation barrier requires compiler supported
- Already implemented in GCC, LLVM, and MSVC
- Can be automated (MSVC) $\rightarrow$ not really reliable

Spectre Variant 1 Mitigations



- Speculation barrier requires compiler supported
- Already implemented in GCC, LLVM, and MSVC
- Can be automated (MSVC) → not really reliable
- Explicit use by programmer: `__builtin_load_no_speculate`

Spectre Variant 1 Mitigations

```
// Unprotected

int array[N];

int get_value(unsigned int n) {
  int tmp;

  if (n < N) {
    tmp = array[n]
  } else {
    tmp = FAIL;
  }

  return tmp;
}
```

Spectre Variant 1 Mitigations

```
// Unprotected

int array[N];

int get_value(unsigned int n) {
  int tmp;

  if (n < N) {
    tmp = array[n]
  } else {
    tmp = FAIL;
  }

  return tmp;
}
```

```
// Protected

int array[N];

int get_value(unsigned int n) {

  int *lower = array;
  int *ptr = array + n;
  int *upper = array + N;

  return
    __builtin_load_no_speculate
    (ptr, lower, upper, FAIL);

}
```

Spectre Variant 1 Mitigations

Spectre Variant 1 Mitigations

- Speculation barrier works if affected code constructs are known

Spectre Variant 1 Mitigations

- Speculation barrier works if affected code constructs are known
- Programmer has to fully understand vulnerability

Spectre Variant 1 Mitigations

- Speculation barrier works if affected code constructs are known
- Programmer has to fully understand vulnerability
- Automatic detection is not reliable

Spectre Variant 1 Mitigations

- Speculation barrier works if affected code constructs are known
- Programmer has to fully understand vulnerability
- Automatic detection is not reliable
- Non-negligible performance overhead of barriers

Spectre Variant 2 Mitigations (Microcode/MSRs)

Intel released microcode updates

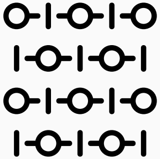
- Indirect Branch Restricted Speculation (IBRS):

Spectre Variant 2 Mitigations (Microcode/MSRs)

Intel released microcode updates

- Indirect Branch Restricted Speculation (IBRS):
    - Do not speculate based on anything before entering IBRS mode

Spectre Variant 2 Mitigations (Microcode/MSRs)

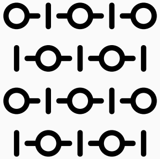


Intel released microcode updates

- Indirect Branch Restricted Speculation (IBRS):
  - Do not speculate based on anything before entering IBRS mode

  $\rightarrow$ lesser privileged code cannot influence predictions

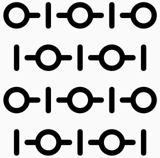
Spectre Variant 2 Mitigations (Microcode/MSRs)



Intel released microcode updates

- Indirect Branch Restricted Speculation (IBRS):
  - Do not speculate based on anything before entering IBRS mode
  - → lesser privileged code cannot influence predictions
- Indirect Branch Predictor Barrier (IBPB):

Spectre Variant 2 Mitigations (Microcode/MSRs)



Intel released microcode updates

- Indirect Branch Restricted Speculation (IBRS):
  - Do not speculate based on anything before entering IBRS mode

  $\rightarrow$ lesser privileged code cannot influence predictions
- Indirect Branch Predictor Barrier (IBPB):
  - Flush branch-target buffer

Spectre Variant 2 Mitigations (Microcode/MSRs)

Intel released microcode updates

- Indirect Branch Restricted Speculation (IBRS):
  - Do not speculate based on anything before entering IBRS mode

  $\rightarrow$ lesser privileged code cannot influence predictions
- Indirect Branch Predictor Barrier (IBPB):
  - Flush branch-target buffer
- Single Thread Indirect Branch Predictors (STIBP):

Spectre Variant 2 Mitigations (Microcode/MSRs)

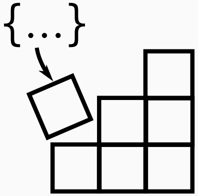
Intel released microcode updates

- Indirect Branch Restricted Speculation (IBRS):
    - Do not speculate based on anything before entering IBRS mode

    → lesser privileged code cannot influence predictions
- Indirect Branch Predictor Barrier (IBPB):
    - Flush branch-target buffer
- Single Thread Indirect Branch Predictors (STIBP):
    - Isolates branch prediction state between two hyperthreads

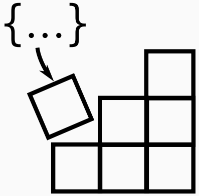
Spectre Variant 2 Mitigations (Software)

Retpoline (compiler extension)



Spectre Variant 2 Mitigations (Software)

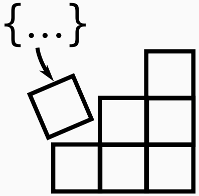
Retpoline (compiler extension)



```
  push <call_target>
  call 1f
2:                     ; speculation will continue here
  lfence               ; speculation barrier
  jmp 2b               ; endless loop
1:
  lea 8(%rsp), %rsp    ; restore stack pointer
  ret                  ; the actual call to <call_target>
```

→ always predict to enter an endless loop

Spectre Variant 2 Mitigations (Software)



Retpoline (compiler extension)

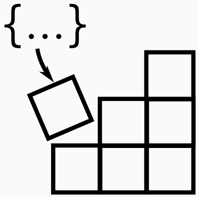
```
  push <call_target>
  call 1f
2:                       ; speculation will continue here
  lfence                 ; speculation barrier
  jmp 2b                 ; endless loop
1:
  lea 8(%rsp), %rsp      ; restore stack pointer
  ret                    ; the actual call to <call_target>
```

→ always predict to enter an endless loop

- instead of the correct (or wrong) target function

Spectre Variant 2 Mitigations (Software)

Retpoline (compiler extension)



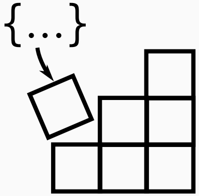
```
  push <call_target>
  call 1f
2:                        ; speculation will continue here
  lfence                  ; speculation barrier
  jmp 2b                  ; endless loop
1:
  lea 8(%rsp), %rsp       ; restore stack pointer
  ret                     ; the actual call to <call_target>
```

→ always predict to enter an endless loop

- instead of the correct (or wrong) target function → performance?

Spectre Variant 2 Mitigations (Software)

Retpoline (compiler extension)



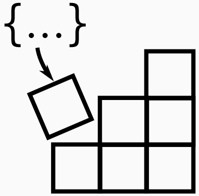
```
  push <call_target>
  call 1f
2:                        ; speculation will continue here
  lfence                  ; speculation barrier
  jmp 2b                  ; endless loop
1:
  lea 8(%rsp), %rsp ; restore stack pointer
  ret                     ; the actual call to <call_target>
```

→ always predict to enter an endless loop

- instead of the correct (or wrong) target function → performance?
- On Broadwell or newer:

Spectre Variant 2 Mitigations (Software)

Retpoline (compiler extension)



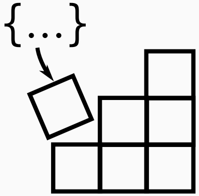
```
    push <call_target>
    call 1f
2:                          ; speculation will continue here
    lfence                  ; speculation barrier
    jmp 2b                  ; endless loop
1:
    lea 8(%rsp), %rsp       ; restore stack pointer
    ret                     ; the actual call to <call_target>
```

→ always predict to enter an endless loop

- instead of the correct (or wrong) target function → performance?
- On Broadwell or newer:
    - **ret** may fall-back to the BTB for prediction

Spectre Variant 2 Mitigations (Software)

Retpoline (compiler extension)



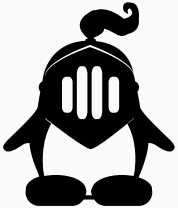
```
    push <call_target>
    call 1f
2:                        ; speculation will continue here
    lfence                ; speculation barrier
    jmp 2b                ; endless loop
1:
    lea 8(%rsp), %rsp     ; restore stack pointer
    ret                   ; the actual call to <call_target>
```

→ always predict to enter an endless loop

- instead of the correct (or wrong) target function → performance?
- On Broadwell or newer:
  - **ret** may fall-back to the BTB for prediction

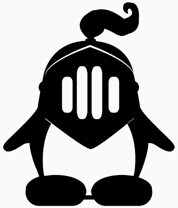
  → microcode patches to prevent that

Spectre Variant 2 Mitigations (Software)



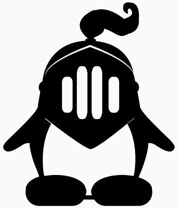
- ARM provides hardened Linux kernel

Spectre Variant 2 Mitigations (Software)



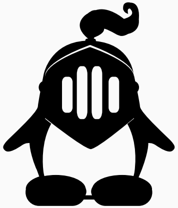
- ARM provides hardened Linux kernel
- Clears branch-predictor state on context switch

Spectre Variant 2 Mitigations (Software)



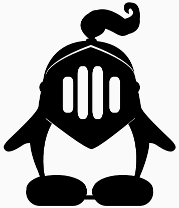
- ARM provides hardened Linux kernel
- Clears branch-predictor state on context switch
- Either via instruction (`BPIALL`)...

Spectre Variant 2 Mitigations (Software)



- ARM provides hardened Linux kernel
- Clears branch-predictor state on context switch
- Either via instruction (`BPIALL`)...
- ...or workaround (disable/enable MMU)

Spectre Variant 2 Mitigations (Software)



- ARM provides hardened Linux kernel
- Clears branch-predictor state on context switch
- Either via instruction (`BPIALL`)...
- ...or workaround (disable/enable MMU)
- Non-negligible performance overhead ($\approx$ 200-300 ns)

What does not work

- Prevent access to high-resolution timer

What does not work

- Prevent access to high-resolution timer

→ Own timer using timing thread

What does not work

- Prevent access to high-resolution timer

$\rightarrow$ Own timer using timing thread

- Flush instruction only privileged

What does not work

- Prevent access to high-resolution timer

→ Own timer using timing thread

- Flush instruction only privileged

→ Cache eviction through memory accesses

What does not work

- Prevent access to high-resolution timer

→ Own timer using timing thread

- Flush instruction only privileged

→ Cache eviction through memory accesses

- Just move secrets into secure world

What does not work

- Prevent access to high-resolution timer

→ Own timer using timing thread

- Flush instruction only privileged

→ Cache eviction through memory accesses

- Just move secrets into secure world

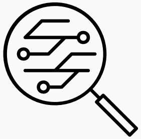
→ Spectre works on secure enclaves

What to do now?

Don’t panic

- Is the used hardware even affected?

Don’t panic



- Is the used hardware even affected?
- Can untrusted users run code on affected hardware?

Don’t panic

- Is the used hardware even affected?
- Can untrusted users run code on affected hardware?
- Is a software attack even in the threat model?

Don’t panic

- Is the used hardware even affected?
- Can untrusted users run code on affected hardware?
- Is a software attack even in the threat model?
- Is confidentiality required on the hardware?

Learn from it

We have ignored software side-channels for many many years:

Learn from it

We have ignored software side-channels for many many years:

- attacks on crypto

Learn from it

We have ignored software side-channels for many many years:

- attacks on crypto $\rightarrow$ "software should be fixed"

Learn from it

We have ignored software side-channels for many many years:

- attacks on crypto $\rightarrow$ "software should be fixed"
- attacks on ASLR

Learn from it

We have ignored software side-channels for many many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”

Learn from it

We have ignored software side-channels for many many years:

- attacks on crypto → “software should be fixed”
- attacks on ASLR → “ASLR is broken anyway”
- attacks on SGX and TrustZone

Learn from it

We have ignored software side-channels for many many years:

- attacks on crypto $\rightarrow$ “software should be fixed”
- attacks on ASLR $\rightarrow$ “ASLR is broken anyway”
- attacks on SGX and TrustZone $\rightarrow$ “not part of the threat model”

Learn from it

We have ignored software side-channels for many many years:

- attacks on crypto → "software should be fixed"
- attacks on ASLR → "ASLR is broken anyway"
- attacks on SGX and TrustZone → "not part of the threat model"

→ for years we solely optimized for performance

When you read the manuals...

After learning about a side channel you realize:

When you read the manuals...

After learning about a side channel you realize:

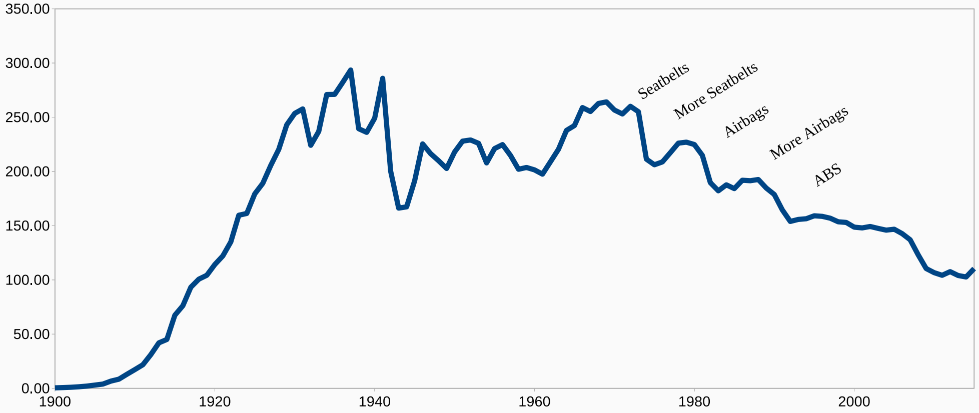
- the side channels were documented in the Intel manual

When you read the manuals...

After learning about a side channel you realize:

- the side channels were documented in the Intel manual
- only now we understand the implications

What do we learn from it?



Motor Vehicle Deaths in U.S. by Year

A unique chance

A unique chance to

- rethink processor design
- grow up, like other fields (car industry, construction industry)
- find good trade-offs between security and performance

Conclusion

- Underestimated microarchitectural attacks for a long time
  - Basic techniques were there for years
- Industry and customers must embrace security mechanisms
  - Run through the same development (for security) as the automobile industry (for safety)
  - It should not be “performance first”, but “security first”

Any Questions?

Spectre and Meltdown on x86 and ARM

Michael Schwarz, Moritz Lipp, Stefan Mangard

15.02.2018

www.iaik.tugraz.at