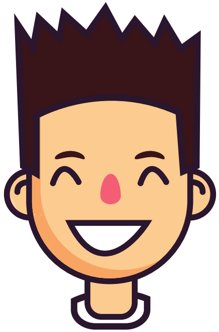
Exploiting the Microarchitecture: Transient Execution Attacks

Michael Schwarz (@misc0110)

April 11, 2019

Graz University of Technology

Who am I?



Michael Schwarz

PhD candidate @ Graz University of Technology

@misc0110

michael.schwarz@iaik.tugraz.at

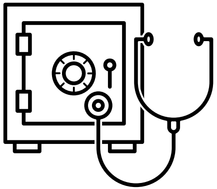
FOX BUSINESS WASHINGTON, D.C.
WASHINGTON, D.C.
WINTER STORM
NEWS ALERT
INTEL REVEALS DESIGN FLAW THAT COULD ALLOW HACKERS TO ACCESS DATA
FOX BUSINESS NETWORK
@FOXBUSINESS

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

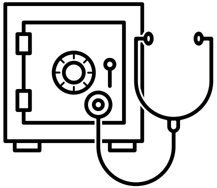
Side-Channel Attacks

- Bug-free software does not mean safe execution

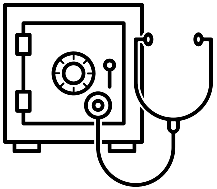


Side-Channel Attacks

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



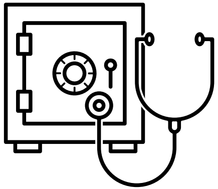
Side-Channel Attacks



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

Side-Channel Attacks

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



Power consumption

Execution time

CPU caches

●●●

Architecture and Microarchitecture

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

Architecture and Microarchitecture

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

Architecture and Microarchitecture

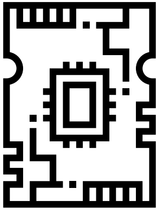
- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)
- Interface between hardware and software
- Microarchitecture is an ISA implementation

Architecture and Microarchitecture

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

Microarchitectural Components

- Modern CPUs contain multiple microarchitectural elements



Microarchitectural Components

- Modern CPUs contain multiple microarchitectural elements

Caches and buffers

Predictors

...

Microarchitectural Components

- Modern CPUs contain multiple microarchitectural elements

Caches and buffers

Predictors

•••

- Transparent for the programmer

Microarchitectural Components

- Modern CPUs contain multiple microarchitectural elements

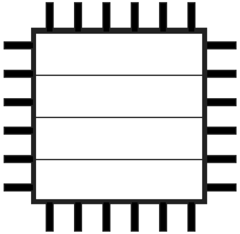
Caches and buffers

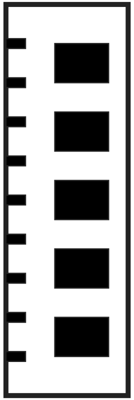
Predictors

- Transparent for the programmer
- Timing optimizations $\rightarrow$ side-channel leakage

CPU Cache

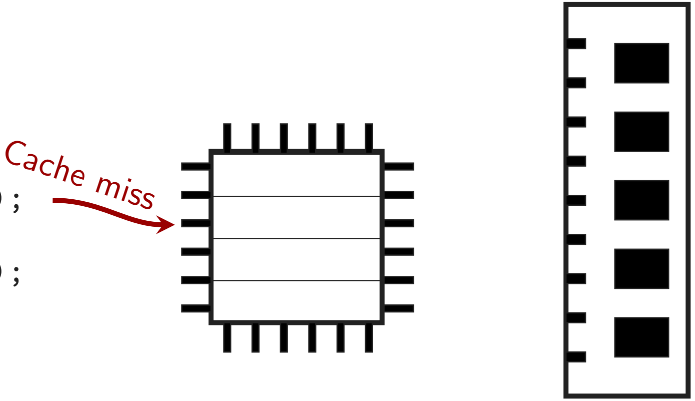
```
printf("%d", i);

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





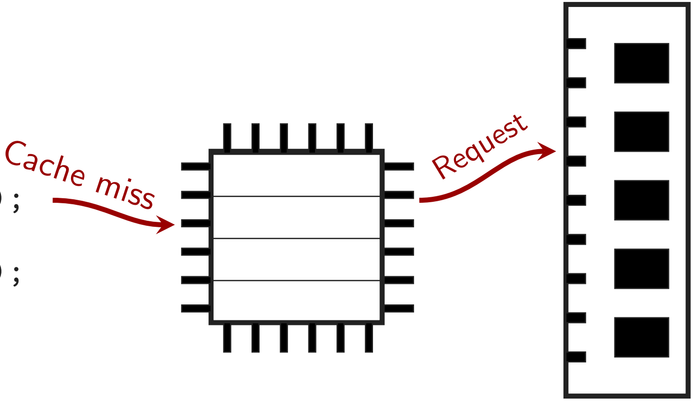
CPU Cache

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

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



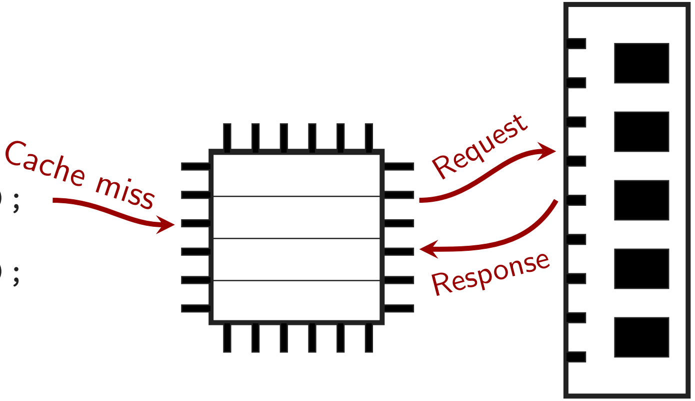
CPU Cache

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

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



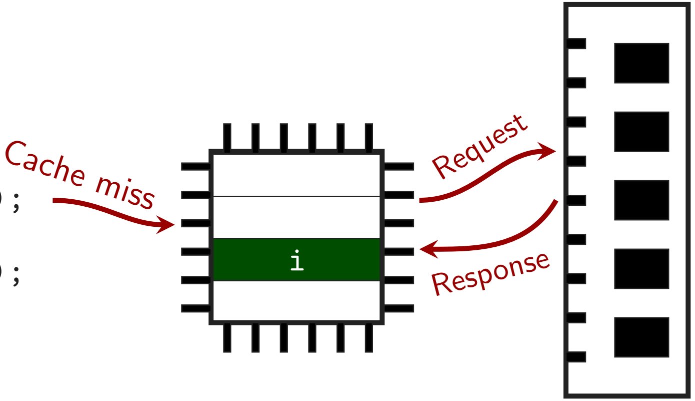
CPU Cache

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

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

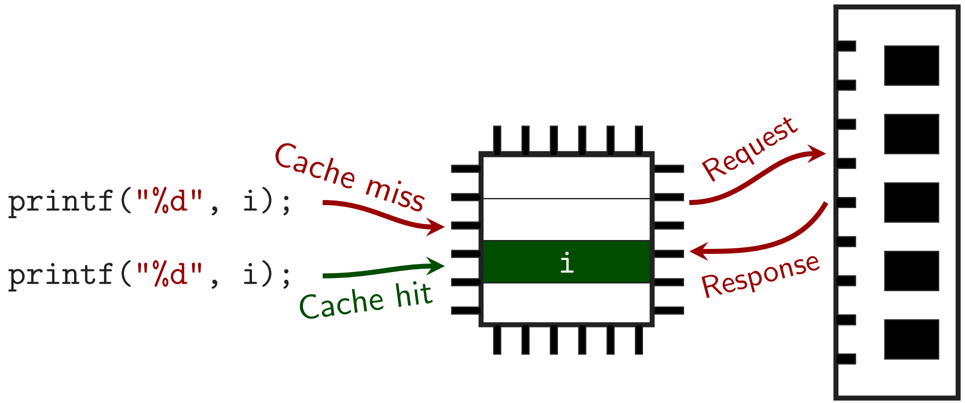


CPU Cache

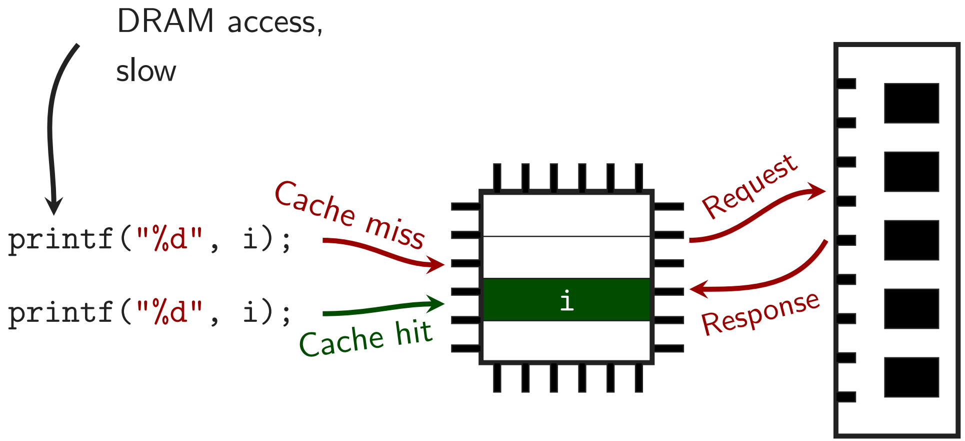
```
printf("%d", i);

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



CPU Cache

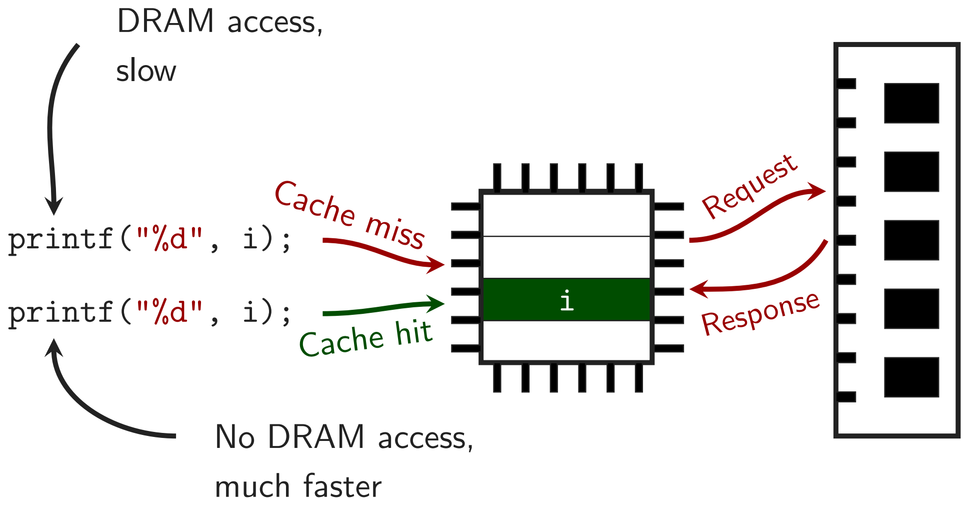


CPU Cache



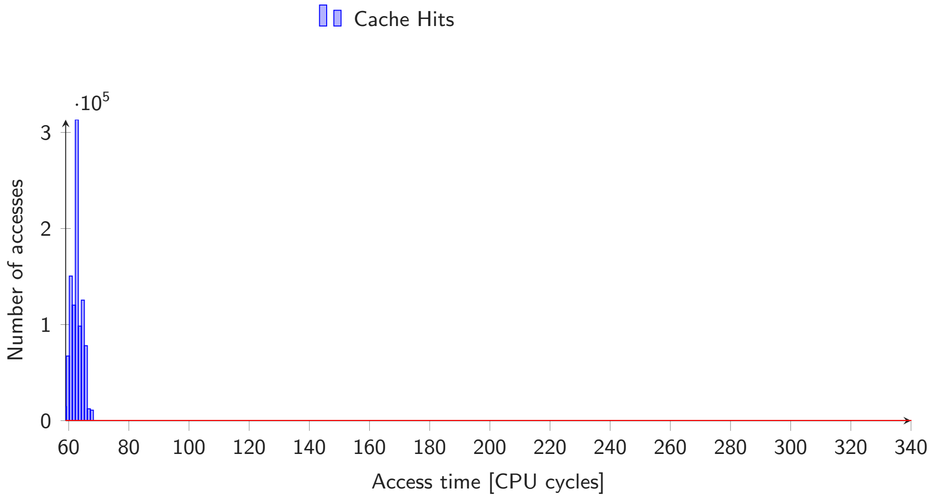
```
printf("%d", i);
printf("%d", i);
```

CPU Cache

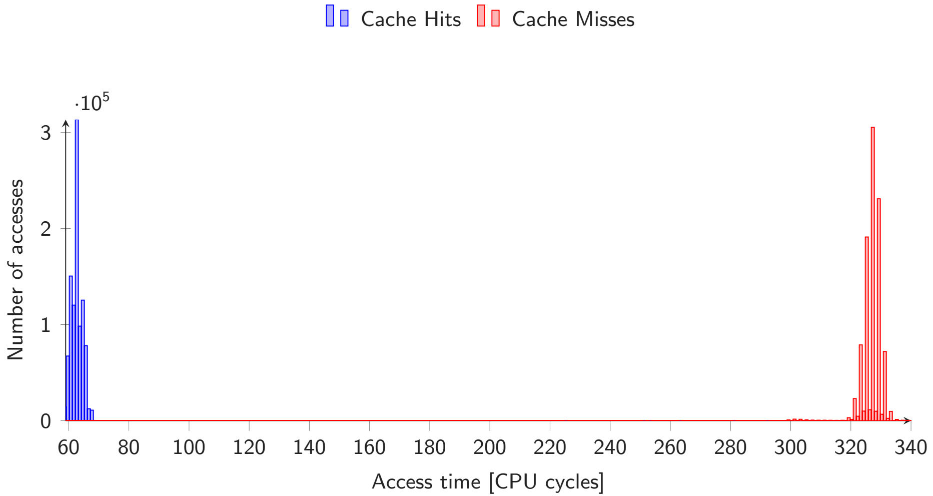


```
printf("%d", i);
printf("%d", i);
```

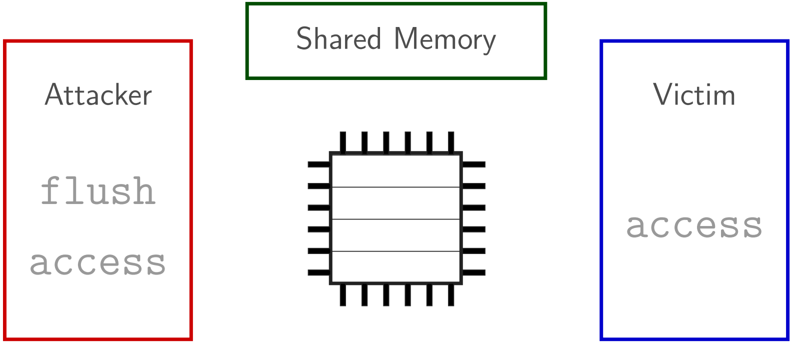
Caching speeds up Memory Accesses



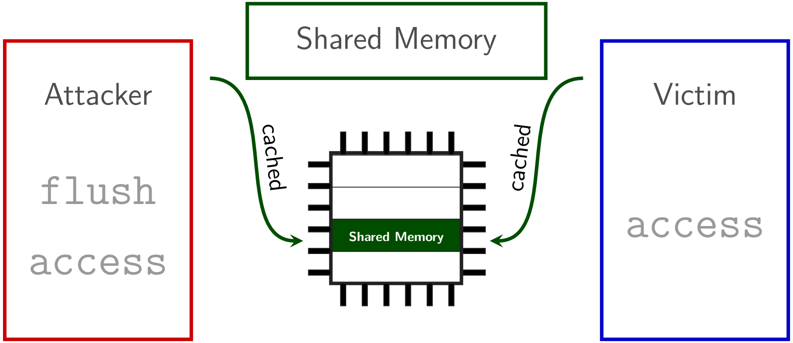
Caching speeds up Memory Accesses



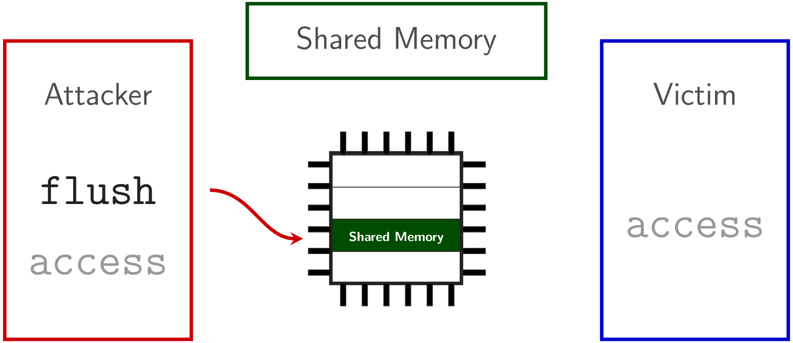
Flush+Reload



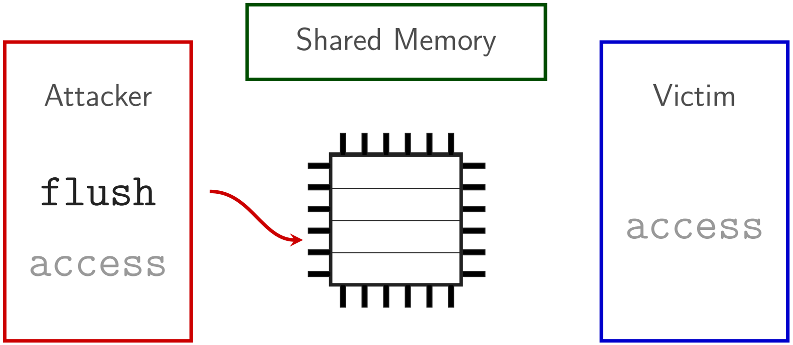
Flush+Reload



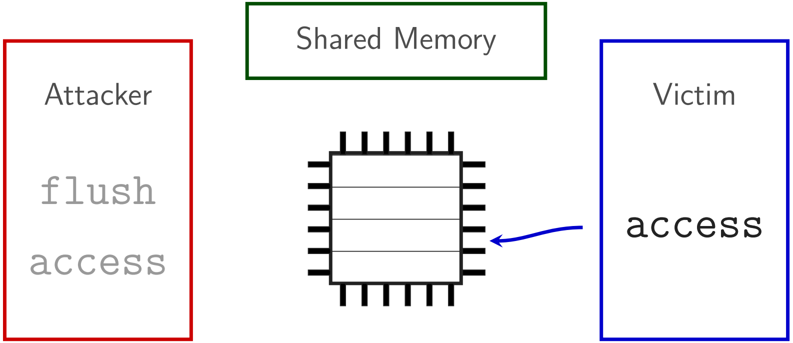
Flush+Reload



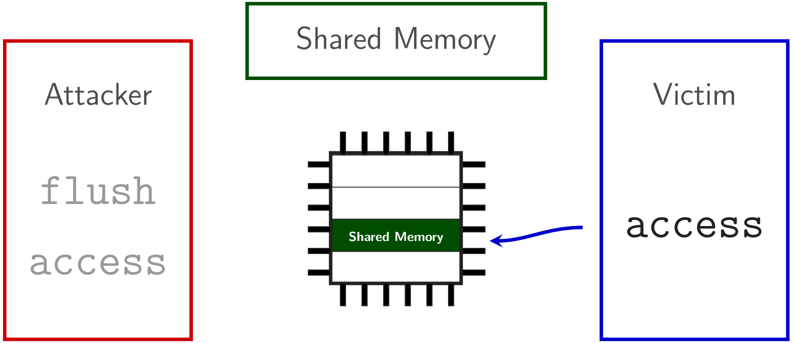
Flush+Reload



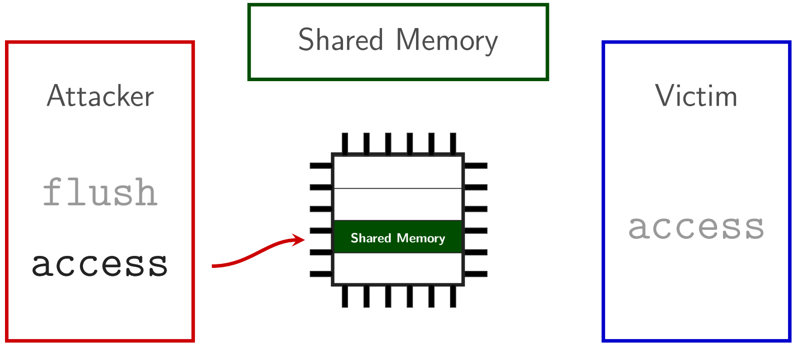
Flush+Reload



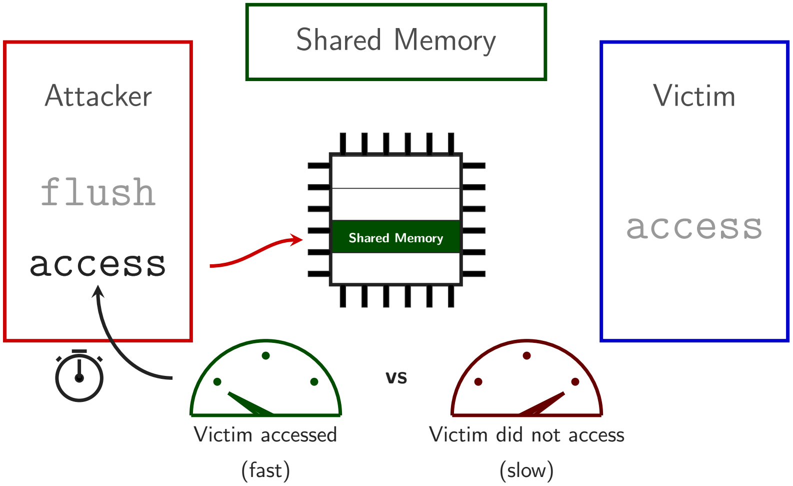
Flush+Reload



Flush+Reload



Flush+Reload



Toy example

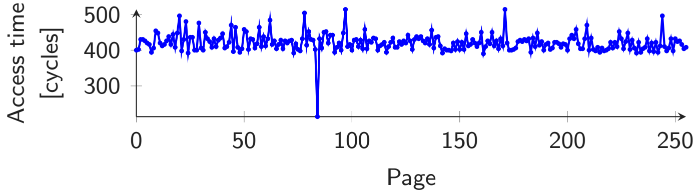
```
char array[256 * 4096]; // 256 pages of memory
```

Toy example

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

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

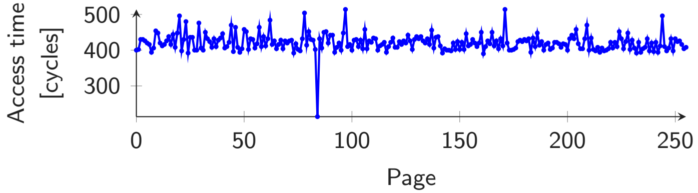
Building the Code

- Flush+Reload over all pages of the array



Building the Code

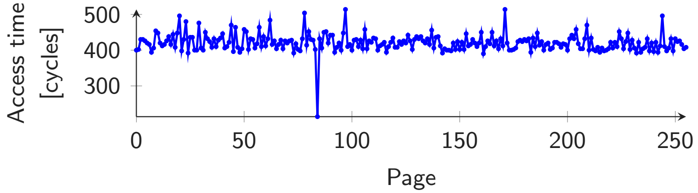
- Flush+Reload over all pages of the array



- “Unreachable” code line was actually executed

Building the Code

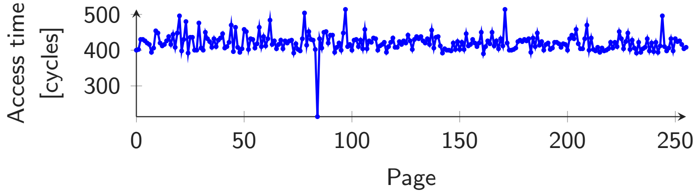
- Flush+Reload over all pages of the array



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

Building the Code

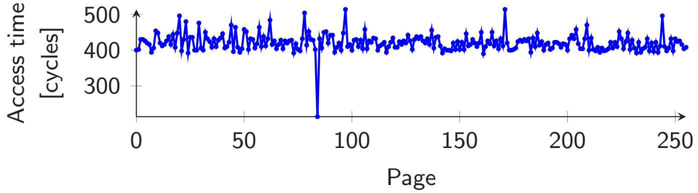
- Flush+Reload over all pages of the array



- "Unreachable" code line was actually executed
- Exception was only thrown afterwards
- Out-of-order instructions leave microarchitectural traces

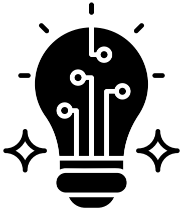
Building the Code

- Flush+Reload over all pages of the array



- "Unreachable" code line was actually executed
- Exception was only thrown afterwards
- Out-of-order instructions leave microarchitectural traces
- Give such instructions a name: transient instructions

Building the Code



- Add another layer of indirection to test

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

Building the Code

- Add another layer of indirection to test

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


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

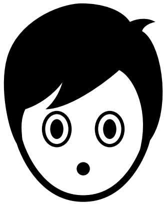
Building the Code

- Add another layer of indirection to test

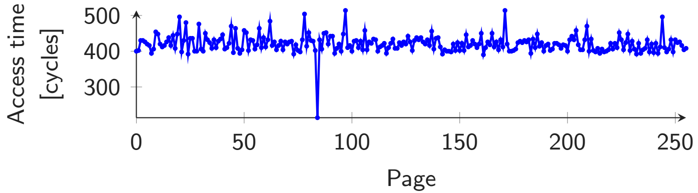
```
char array[256 * 4096]; // 256 pages of memory


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

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

Building the Code

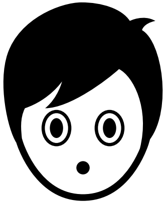


- Flush+Reload over all pages of the array

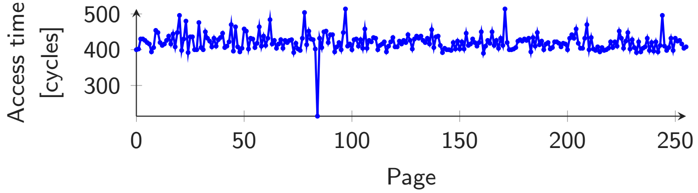


- Index of cache hit reveals data

Building the Code



- Flush+Reload over all pages of the array



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

Meltdown

MELTDOWN

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

Meltdown

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

Meltdown

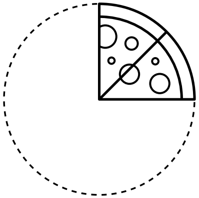
- CPU uses data in out-of-order execution before permission check
- Meltdown can read any kernel address
- Physical memory is usually mapped in kernel

Meltdown

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

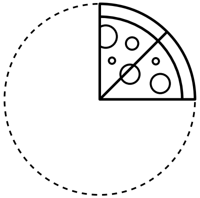
→ Read arbitrary memory

Uncached and uncachable memory



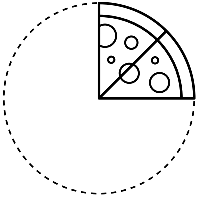
- Assumed Meltdown can one only read data from the L1

Uncached and uncachable memory



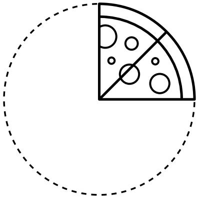
- Assumed Meltdown can one only read data from the L1
- Leakage from L3 or memory is possible, just slower

Uncached and uncachable memory



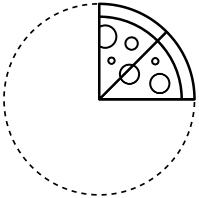
- Assumed Meltdown can one only read data from the L1
- Leakage from L3 or memory is possible, just slower
- Even leakage of UC (uncachable) memory regions...

Uncached and uncachable memory



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

Uncached and uncachable memory



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

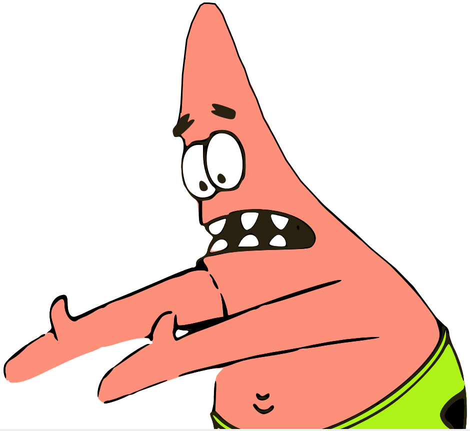
  → ...leaks from line fill buffer

Take the kernel addresses...

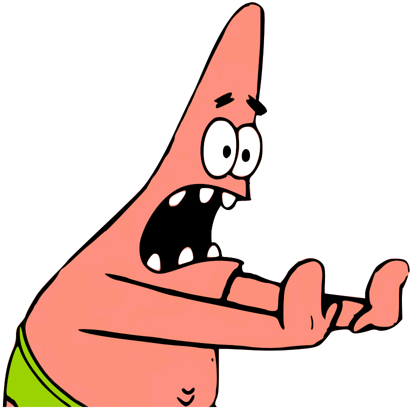
- Kernel addresses in user space are a problem

Take the kernel addresses...

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

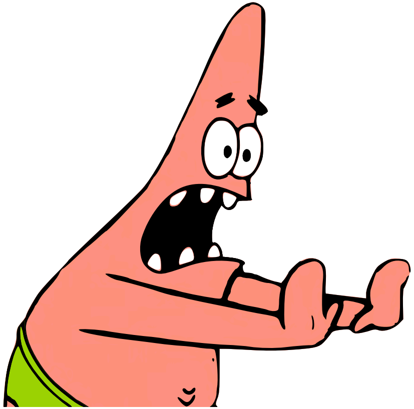


...and remove them



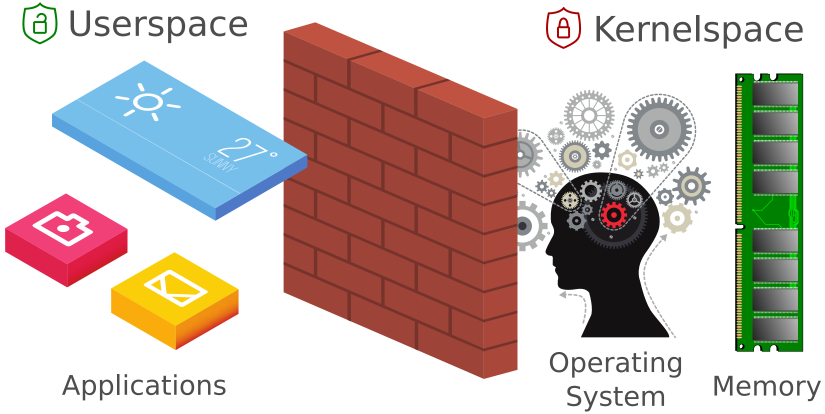
- ...and remove them if not needed?

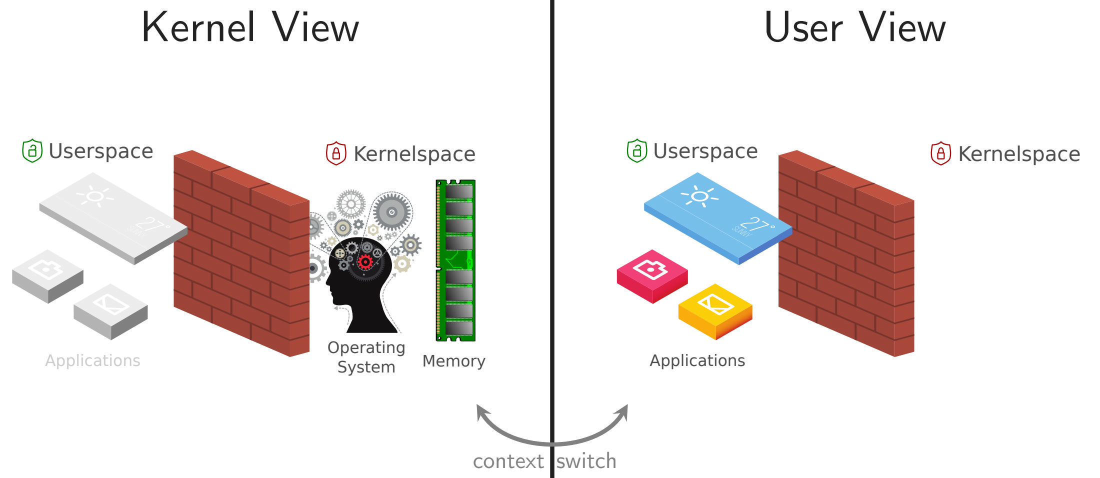
...and remove them



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

Meltdown Mitigation: KAISER




Kernel View
Userspace
Kernelspace
Applications
Operating System
Memory
User View
Userspace
Kernelspace
Applications
context switch

Mitigations

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

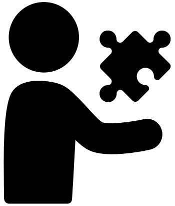
Mitigations

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

Mitigations

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

Problem Solved?

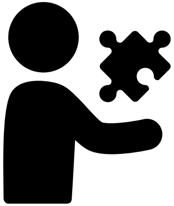


- Meltdown fully mitigated in software

Problem Solved?

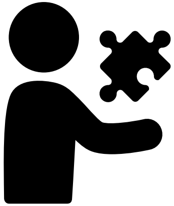
- Meltdown fully mitigated in software
- Problem seemed to be solved

Problem Solved?



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

Problem Solved?



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

There are no bugs,
just happy little accidents

Generalization

- Meltdown is a whole category of vulnerabilities

Generalization

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

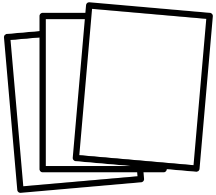
Generalization

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

Paging

- CPU uses virtual address spaces to isolate processes

Paging

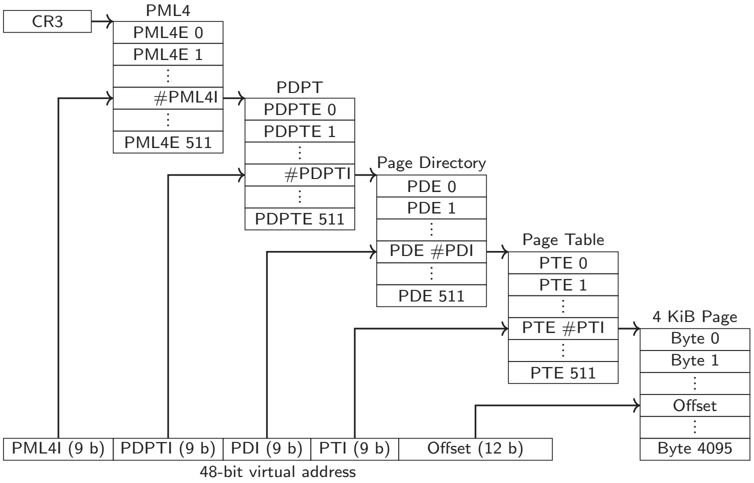


- CPU uses virtual address spaces to isolate processes
- Physical memory is organized in page frames

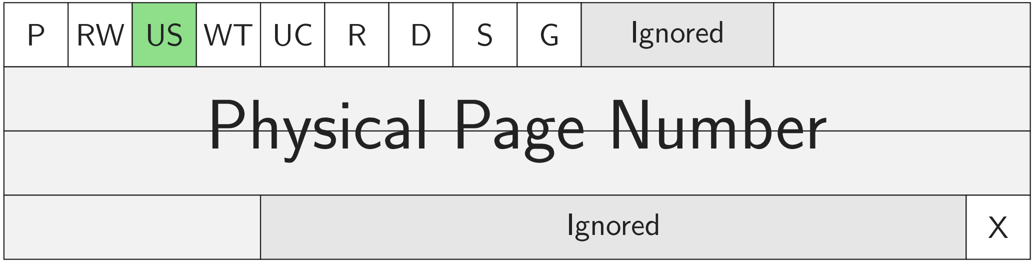
Paging

- 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

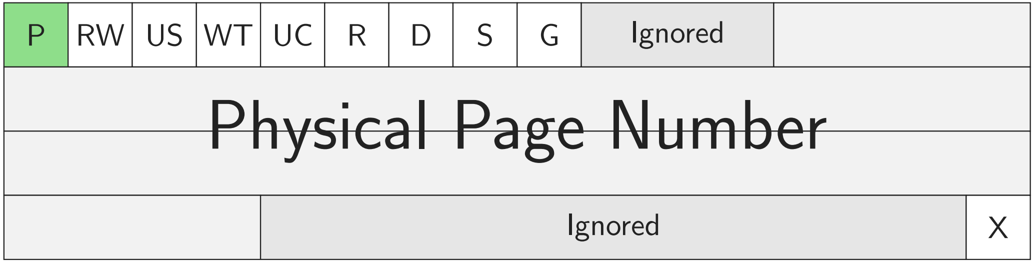


Page Table Entry

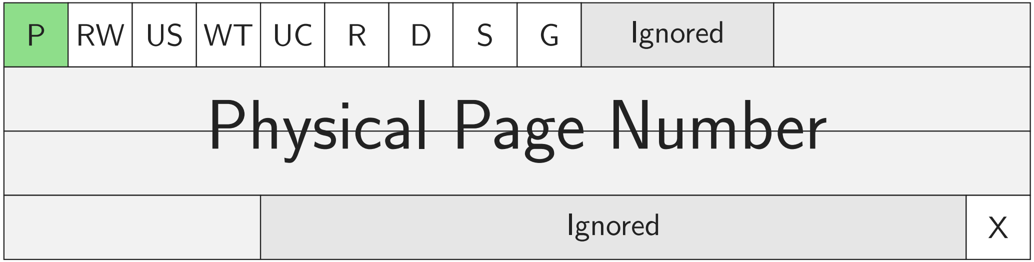


- User/Supervisor bit defines in which privilege level the page can be accessed

Page Table Entry



Page Table Entry



- Present bit is the next obvious bit

Foreshadow-NG

- An even worse bug $\rightarrow$ Foreshadow-NG/L1TF

Foreshadow-NG

- An even worse bug $\rightarrow$ Foreshadow-NG/L1TF
- Exploitable from VMs

Foreshadow-NG

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

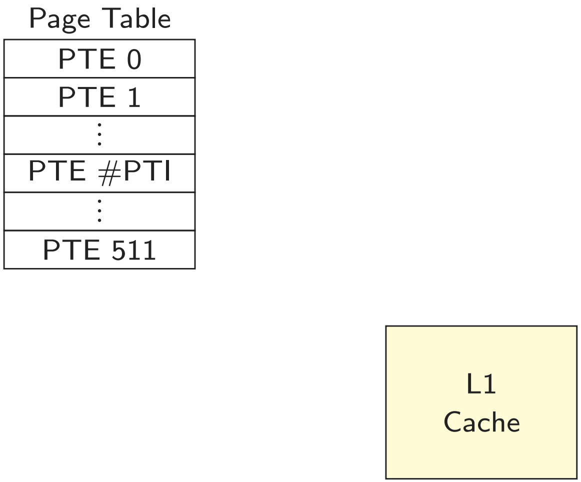
Foreshadow-NG

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

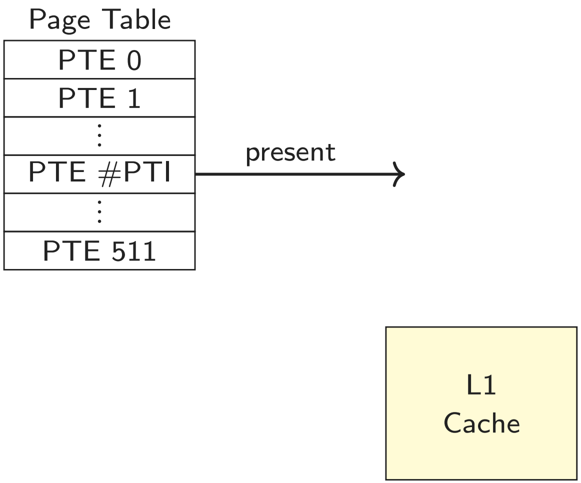
Foreshadow-NG

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

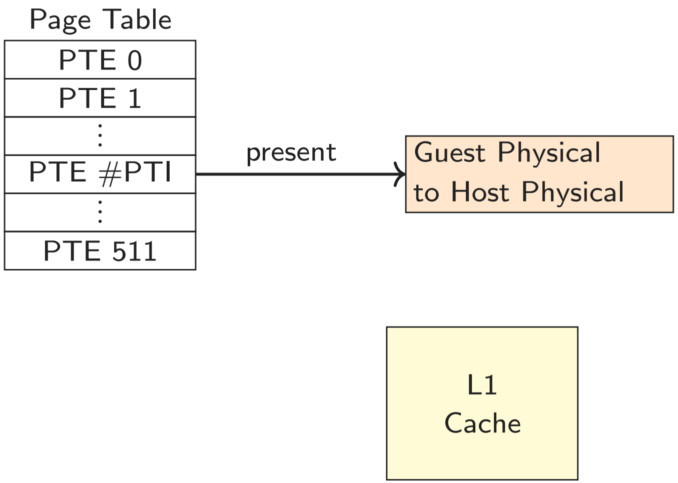
Foreshadow-NG



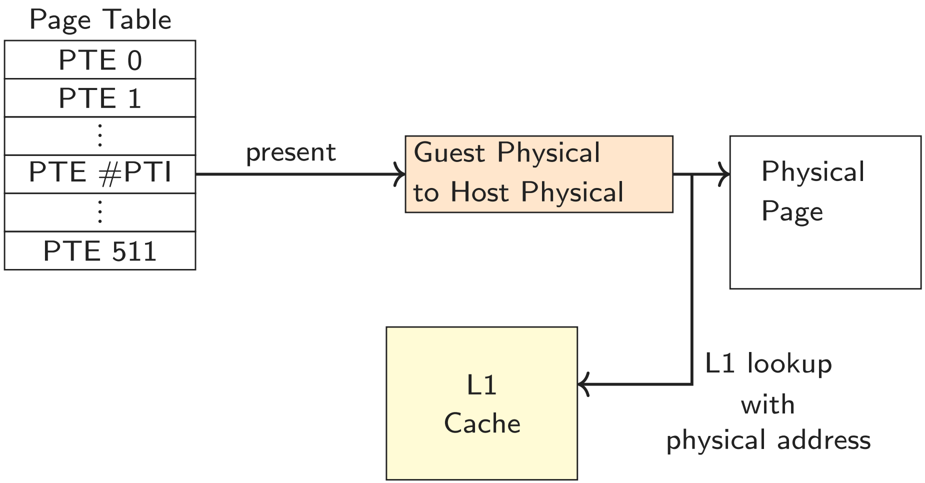
Foreshadow-NG



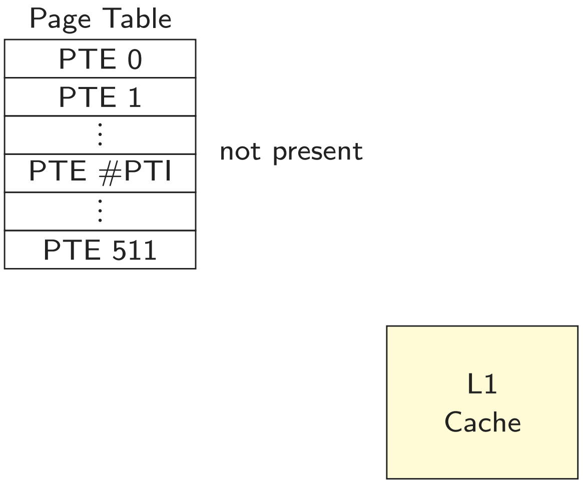
Foreshadow-NG



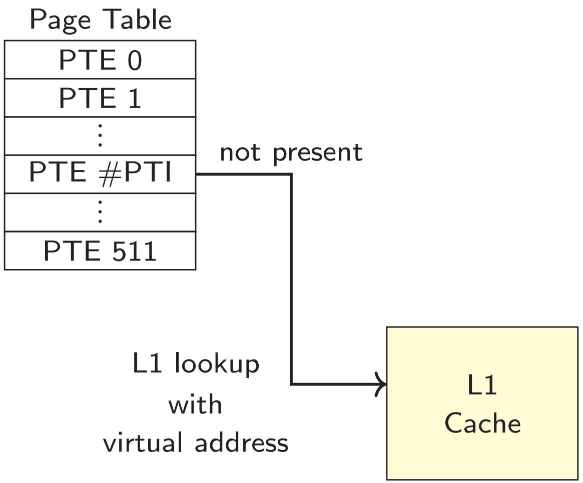
Foreshadow-NG



Foreshadow-NG



Foreshadow-NG



Demo

Foreshadow-NG

Foreshadow-NG Fix

- KAISER/KPTI/KVA does not help

Foreshadow-NG Fix

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

Foreshadow-NG Fix

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

Foreshadow-NG Fix

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

Foreshadow-NG Fix

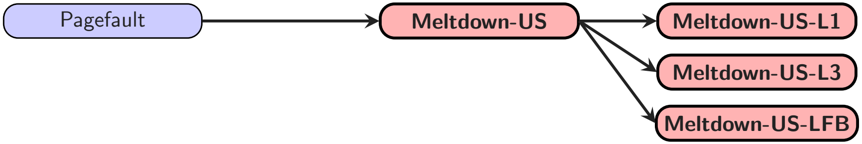
- KAISER/KPTI/KVA does not help
- Only software workarounds
  - → Flush L1 on VM entry
  - → Disable HyperThreading
- Workarounds might not be complete

Meltdown Variants

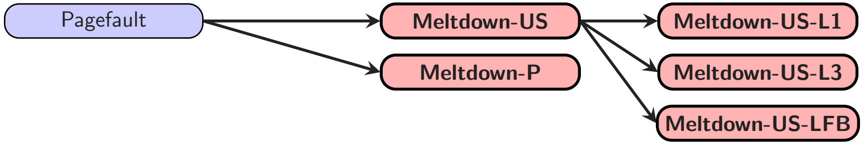
Meltdown Variants

Pagefault
Meltdown-US

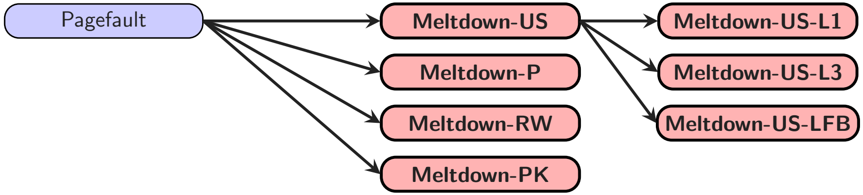
Meltdown Variants



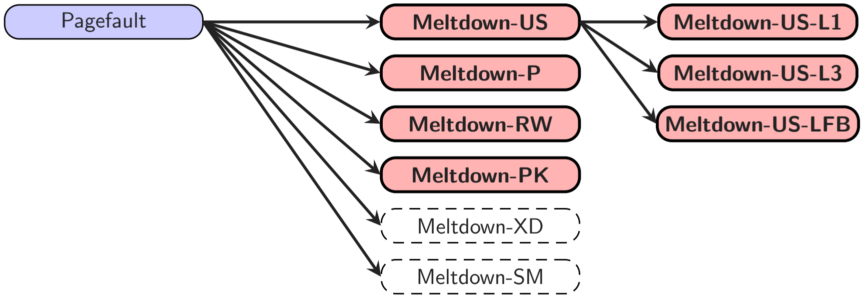
Meltdown Variants



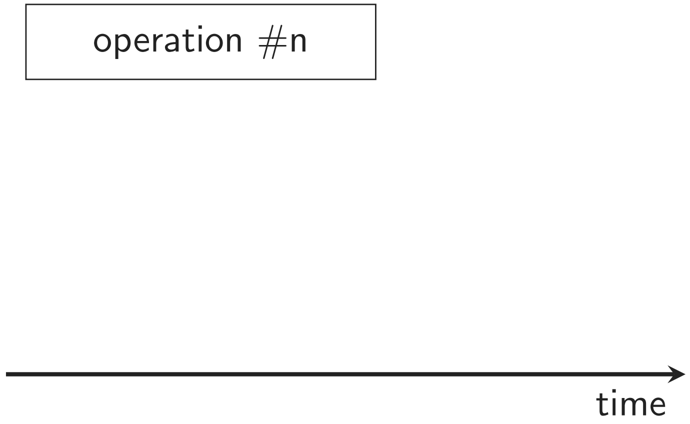
Meltdown Variants


$$\begin{tikzcd}
\text{Pagefault} \arrow[r] \arrow[dr] \arrow[ddr] \arrow[dddr] & \text{Meltdown-US} \arrow[r] \arrow[dr] \arrow[ddr] & \text{Meltdown-US-L1} \\
& \text{Meltdown-P} & \text{Meltdown-US-L3} \\
& \text{Meltdown-RW} & \text{Meltdown-US-LFB} \\
& \text{Meltdown-PK} &
\end{tikzcd}$$

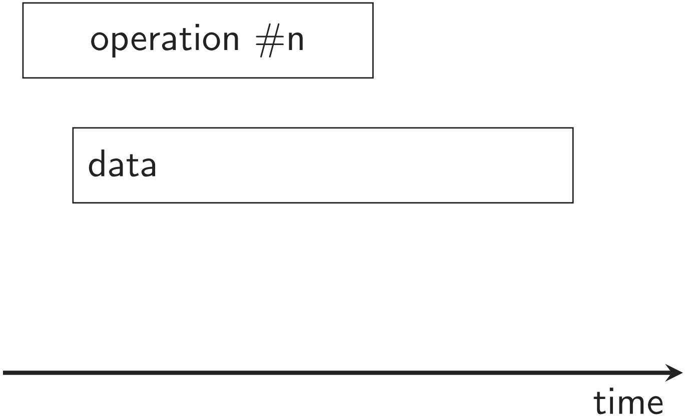
Meltdown Variants



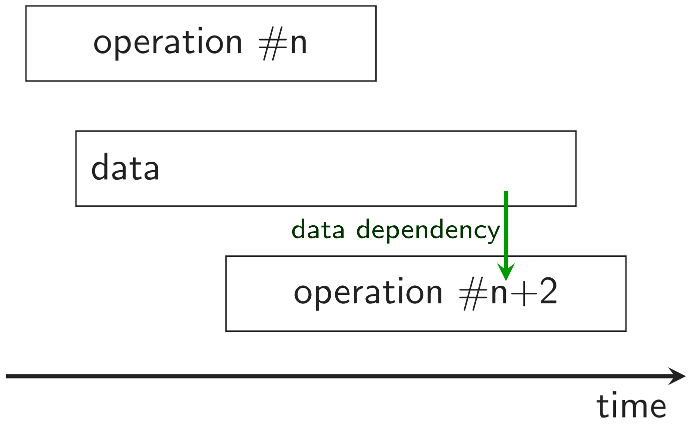
Meltdown Root Cause



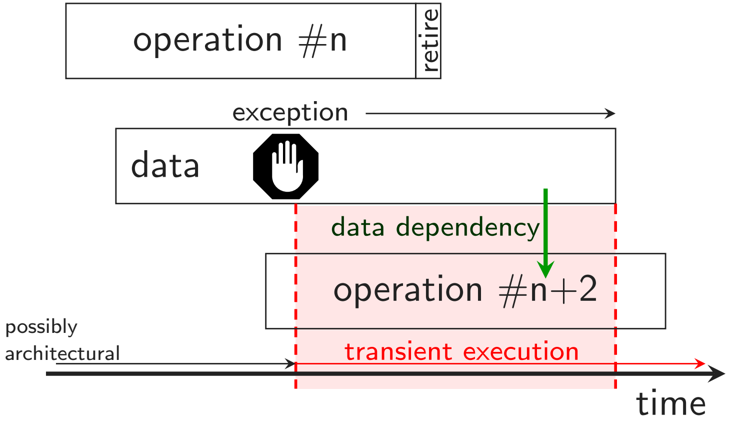
Meltdown Root Cause



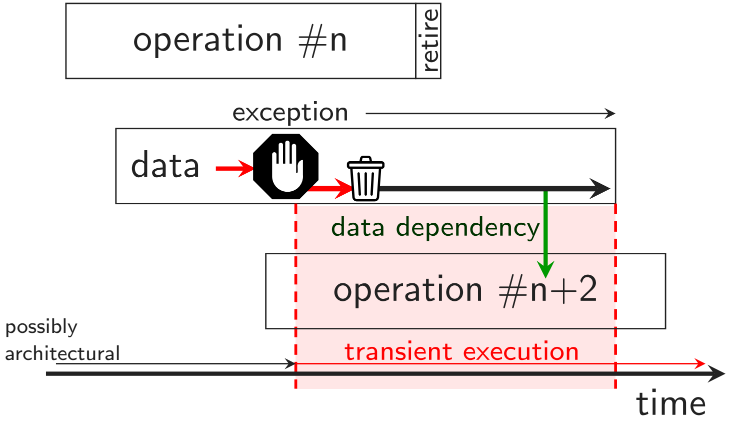
Meltdown Root Cause



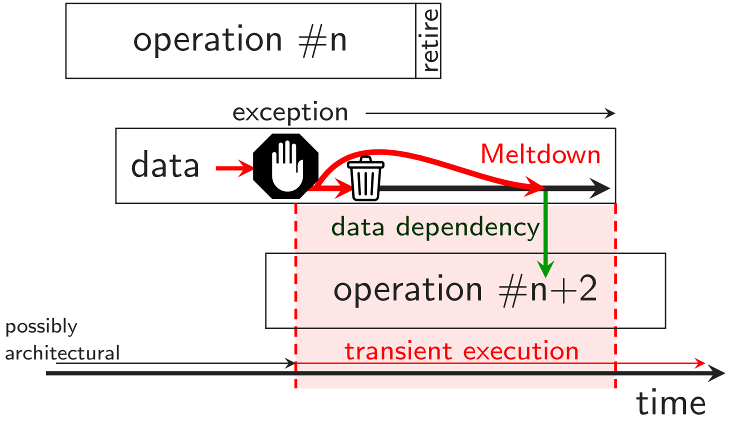
Meltdown Root Cause



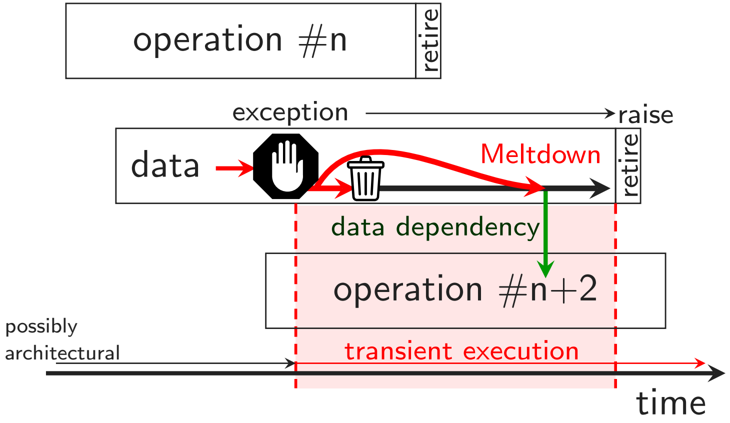
Meltdown Root Cause



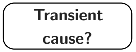
Meltdown Root Cause



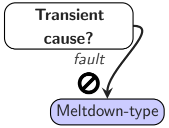
Meltdown Root Cause



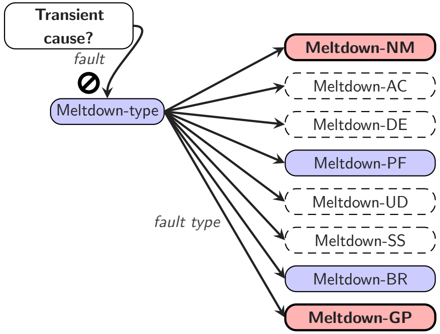
Meltdown Tree



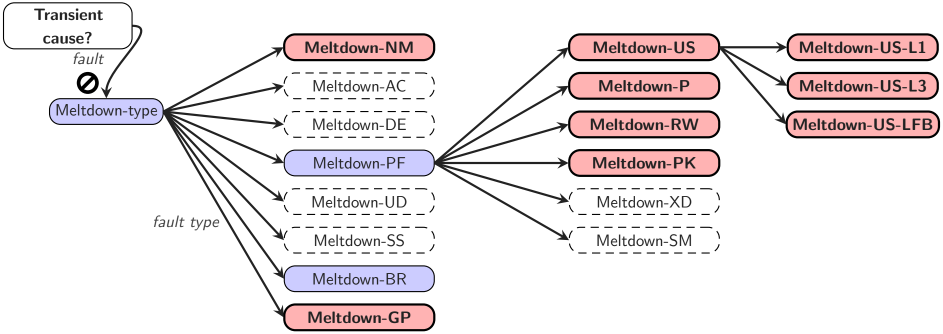
Meltdown Tree



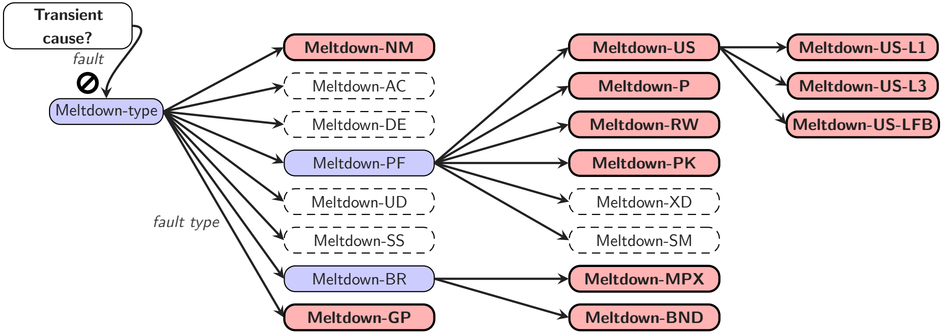
Meltdown Tree



Meltdown Tree



Meltdown Tree



Meltdown Outlook

- Meltdown is not a fully solved issue

Meltdown Outlook

- 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

Transient Execution Attacks

- Meltdown not the only transient execution attacks

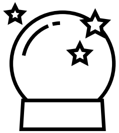
Transient Execution Attacks

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

Transient Execution Attacks

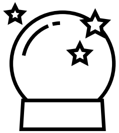
- Meltdown not the only transient execution attacks
- Spectre is a second class of transient execution attacks
- Instead of faults, exploit control (or data) flow predictions

Speculative Execution



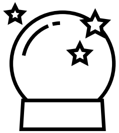
- CPU tries to predict the future (branch predictor), . . .

Speculative Execution



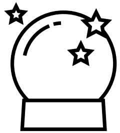
- CPU tries to predict the future (branch predictor), . . .
    - . . . based on events learned in the past

Speculative Execution



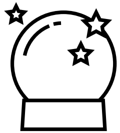
- CPU tries to predict the future (branch predictor), . . .
  - . . . based on events learned in the past
- Speculative execution of instructions

Speculative Execution



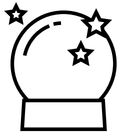
- CPU tries to predict the future (branch predictor), . . .
  - . . . based on events learned in the past
- Speculative execution of instructions
- If the prediction was correct, . . .

Speculative Execution



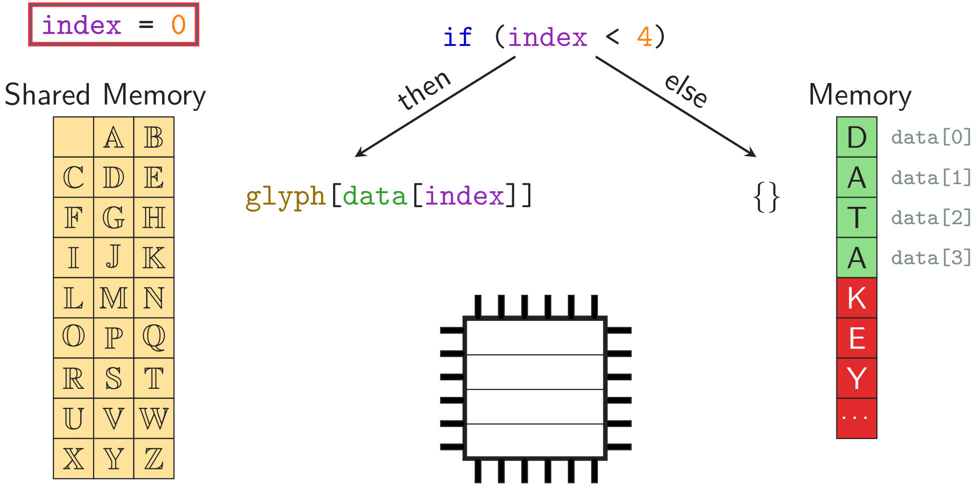
- CPU tries to predict the future (branch predictor), . . .
  - . . . based on events learned in the past
- Speculative execution of instructions
- If the prediction was correct, . . .
  - . . . very fast

Speculative Execution

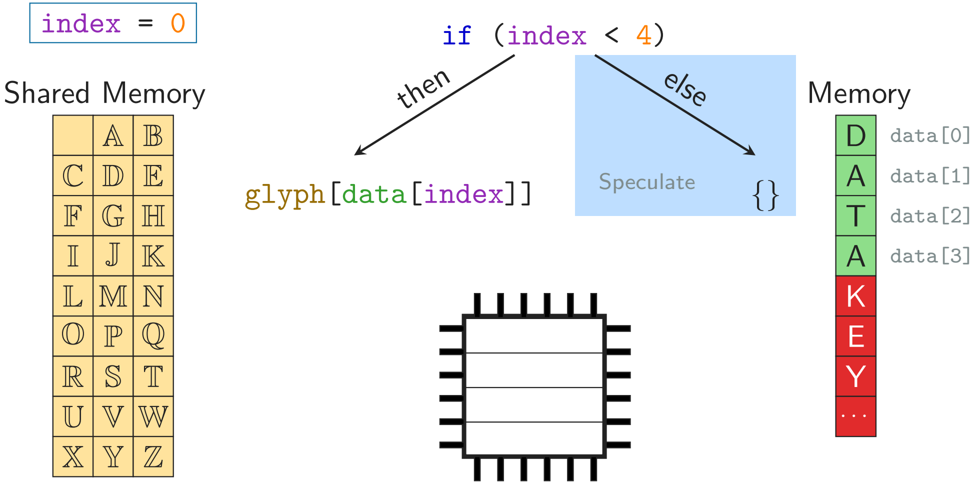


- CPU tries to predict the future (branch predictor), . . .
  - . . . based on events learned in the past
- Speculative execution of instructions
- If the prediction was correct, . . .
  - . . . very fast
  - otherwise: Discard results

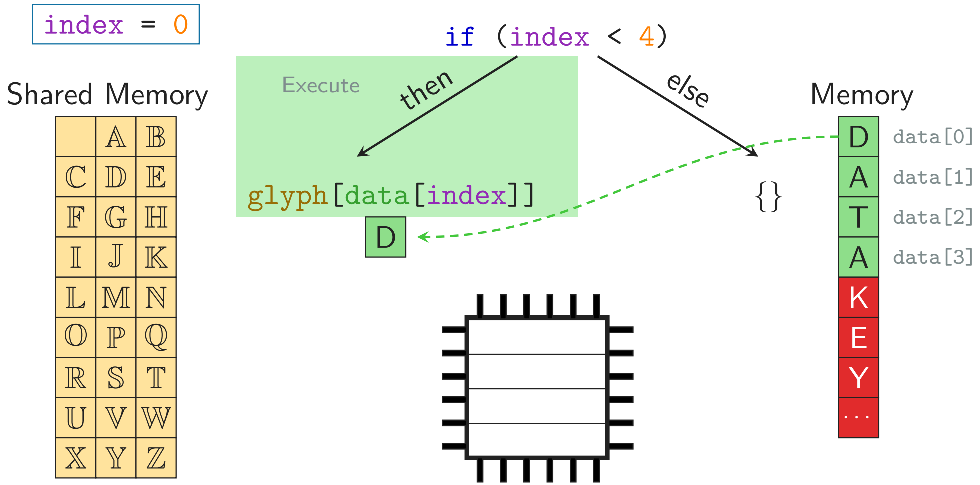
Spectre-PHT (aka Spectre Variant 1)



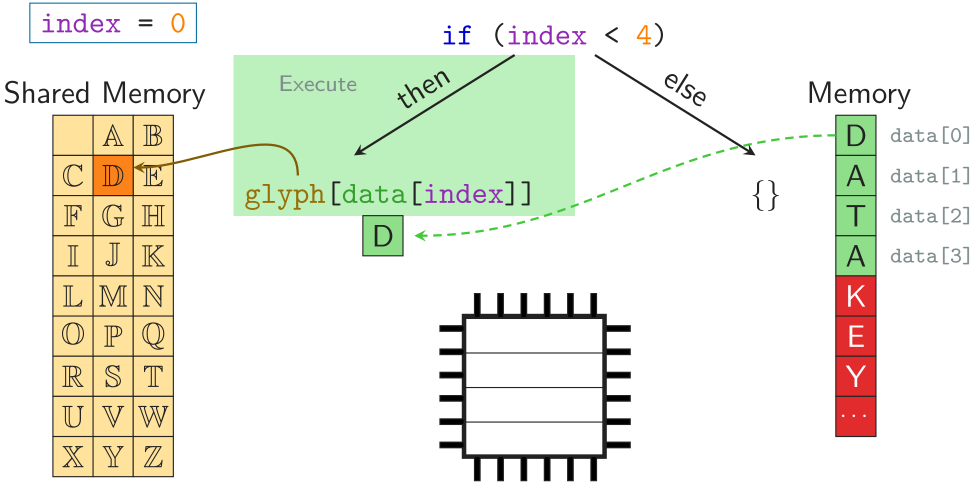
Spectre-PHT (aka Spectre Variant 1)



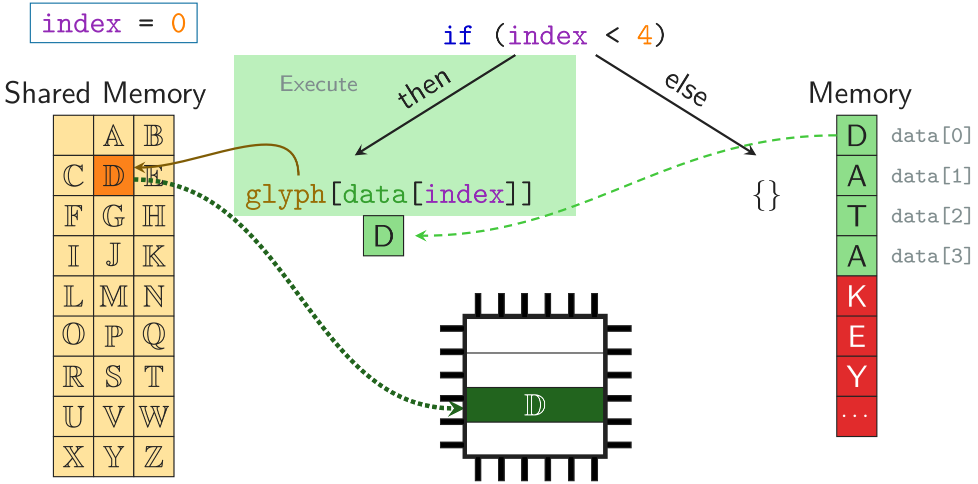
Spectre-PHT (aka Spectre Variant 1)



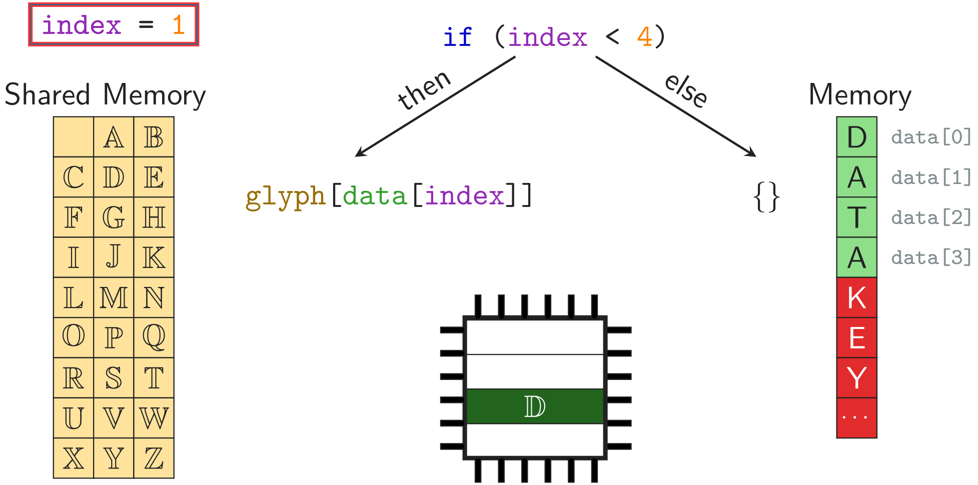
Spectre-PHT (aka Spectre Variant 1)



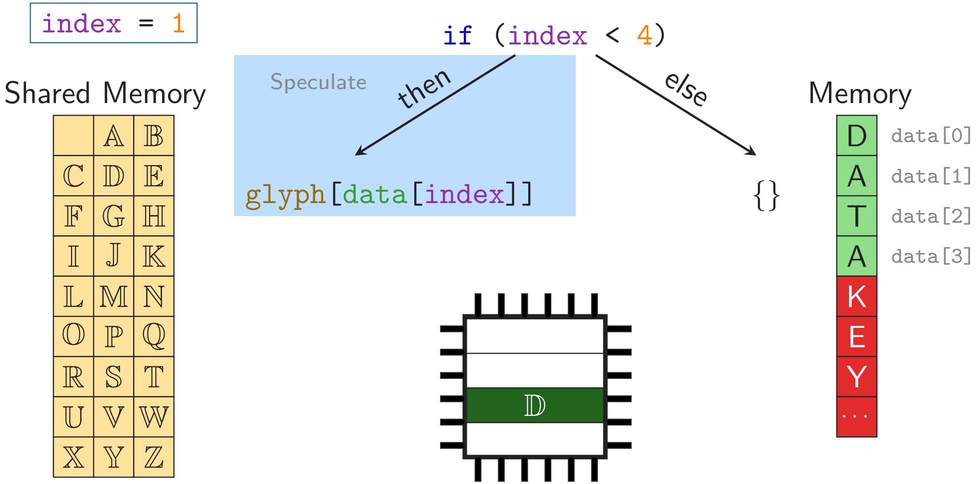
Spectre-PHT (aka Spectre Variant 1)



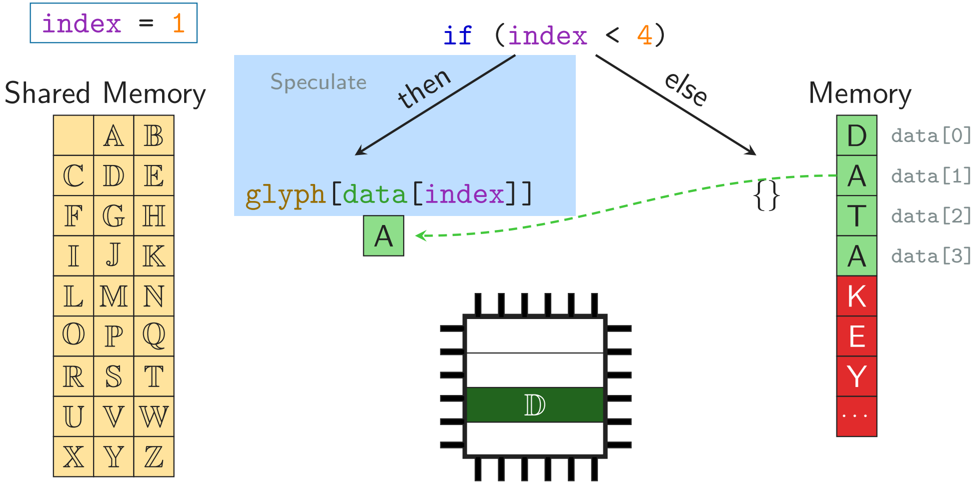
Spectre-PHT (aka Spectre Variant 1)



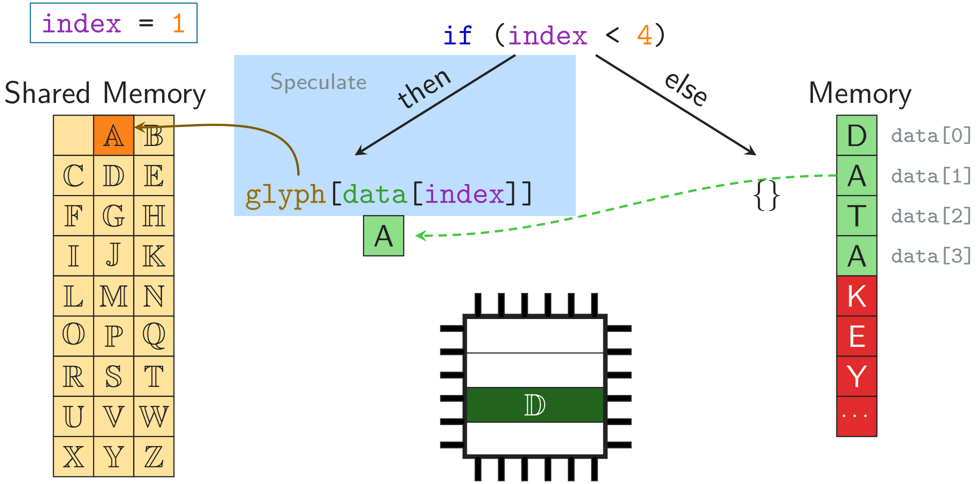
Spectre-PHT (aka Spectre Variant 1)



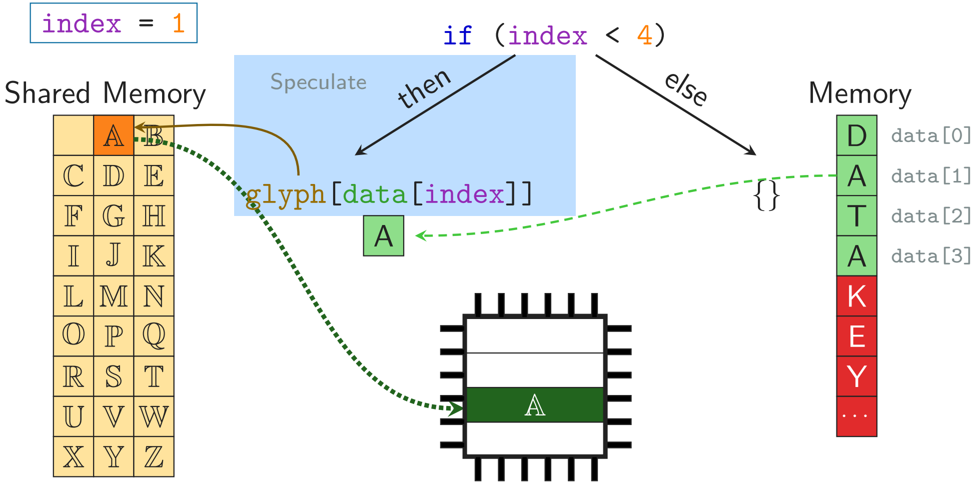
Spectre-PHT (aka Spectre Variant 1)



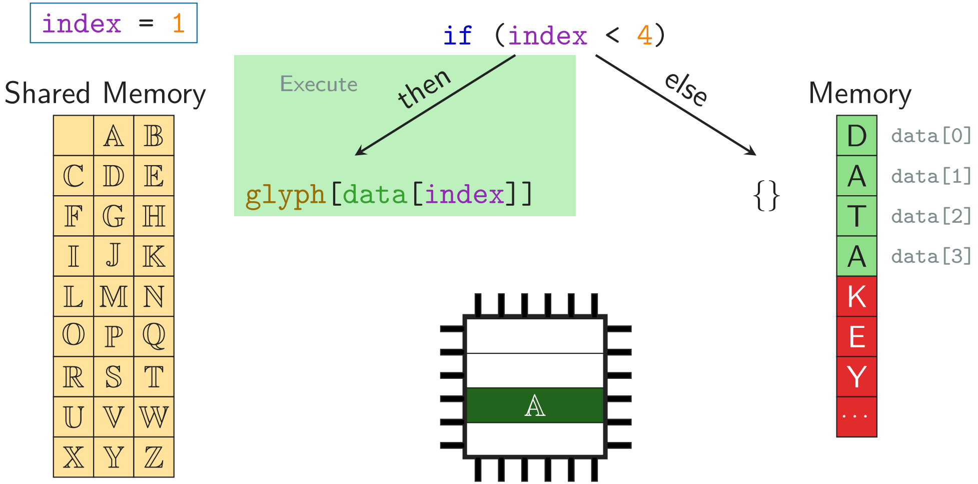
Spectre-PHT (aka Spectre Variant 1)



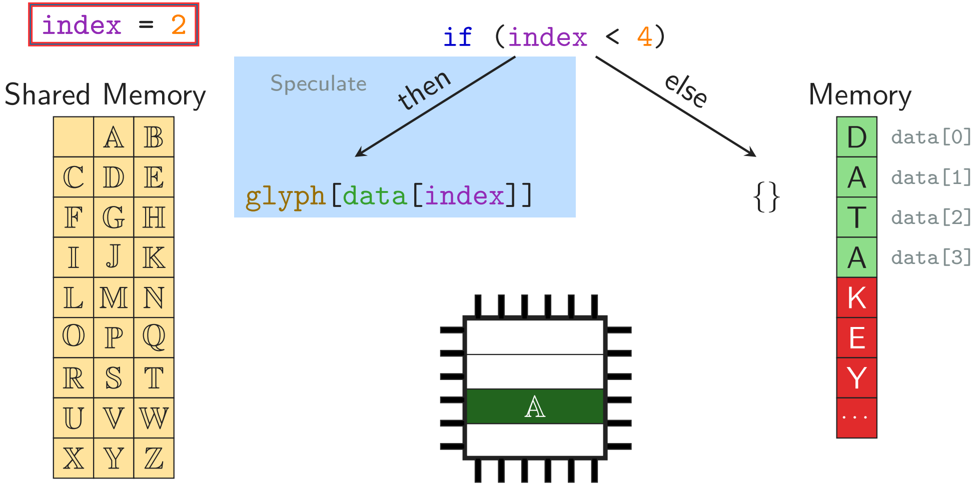
Spectre-PHT (aka Spectre Variant 1)



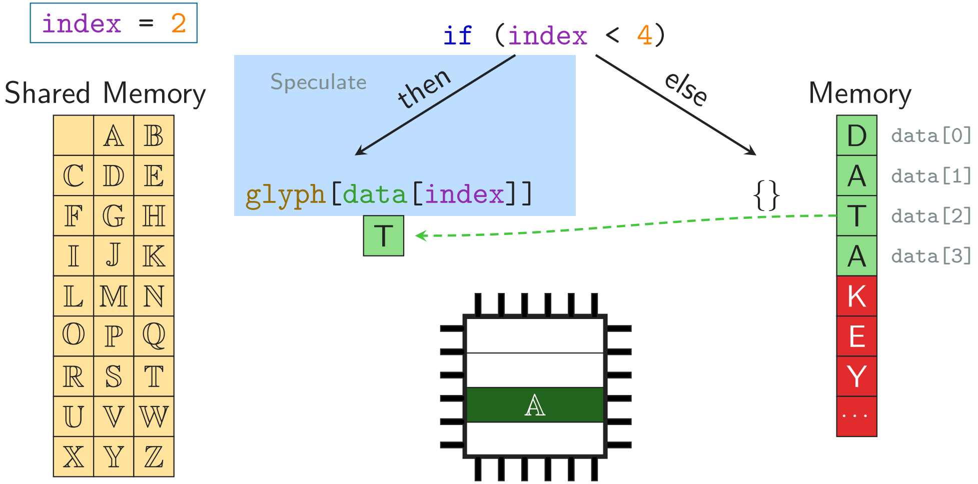
Spectre-PHT (aka Spectre Variant 1)



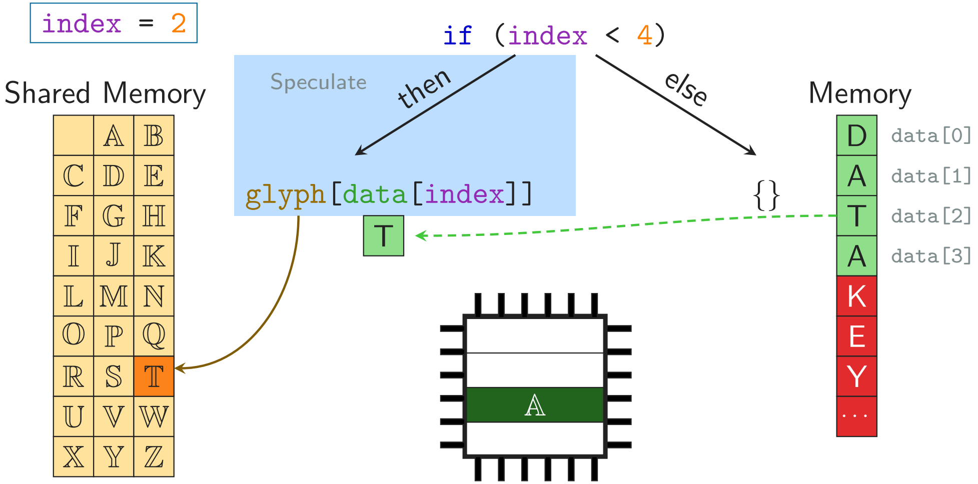
Spectre-PHT (aka Spectre Variant 1)



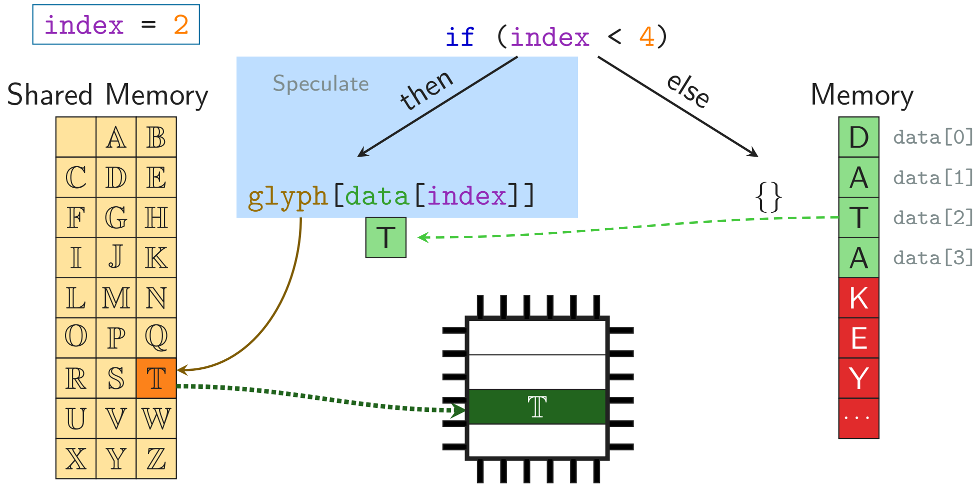
Spectre-PHT (aka Spectre Variant 1)



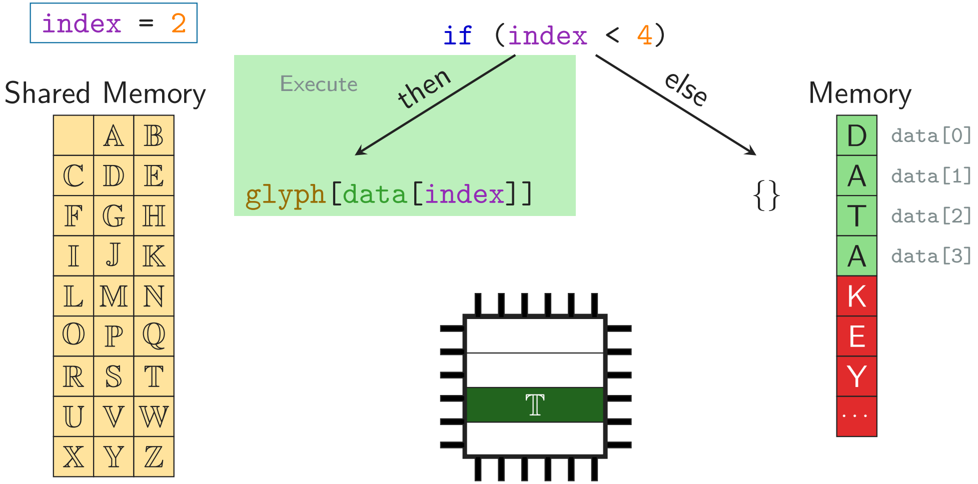
Spectre-PHT (aka Spectre Variant 1)



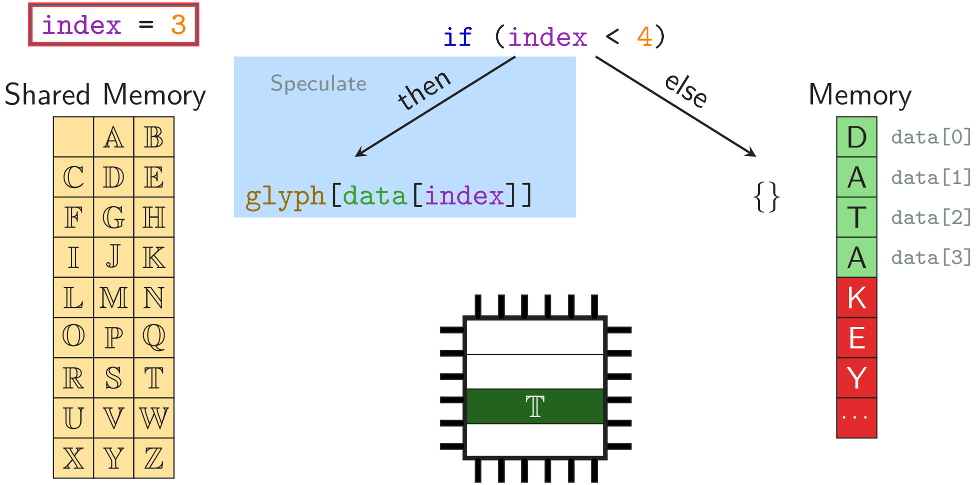
Spectre-PHT (aka Spectre Variant 1)



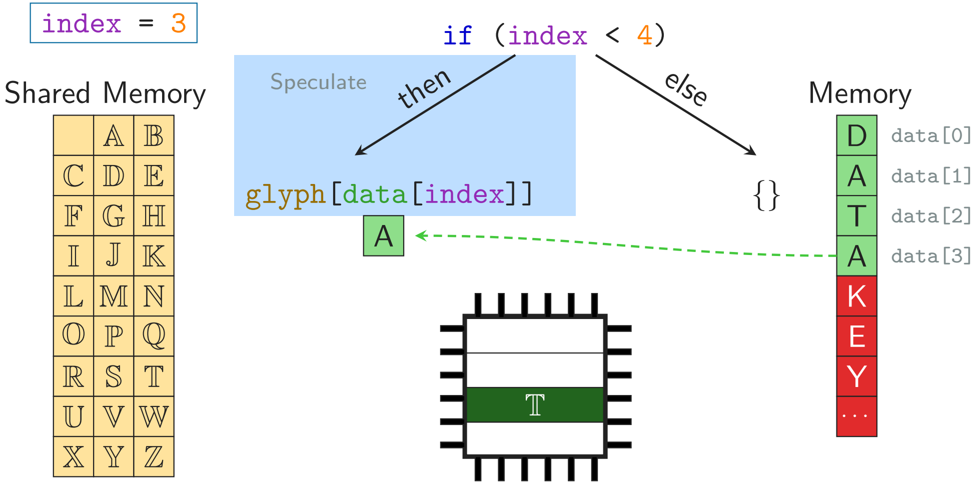
Spectre-PHT (aka Spectre Variant 1)



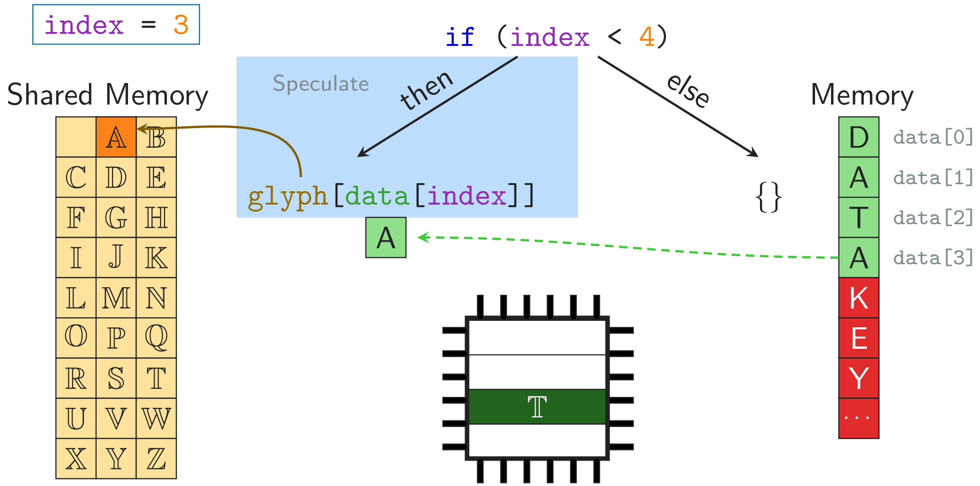
Spectre-PHT (aka Spectre Variant 1)



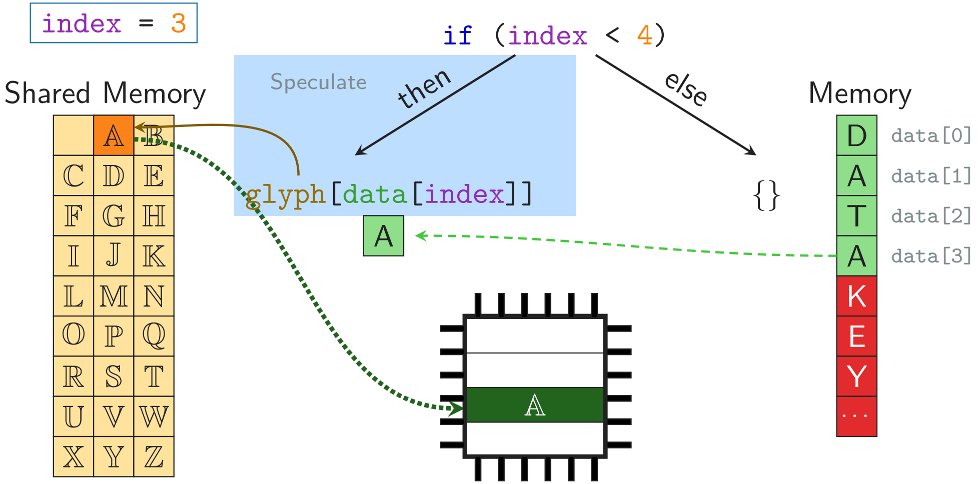
Spectre-PHT (aka Spectre Variant 1)



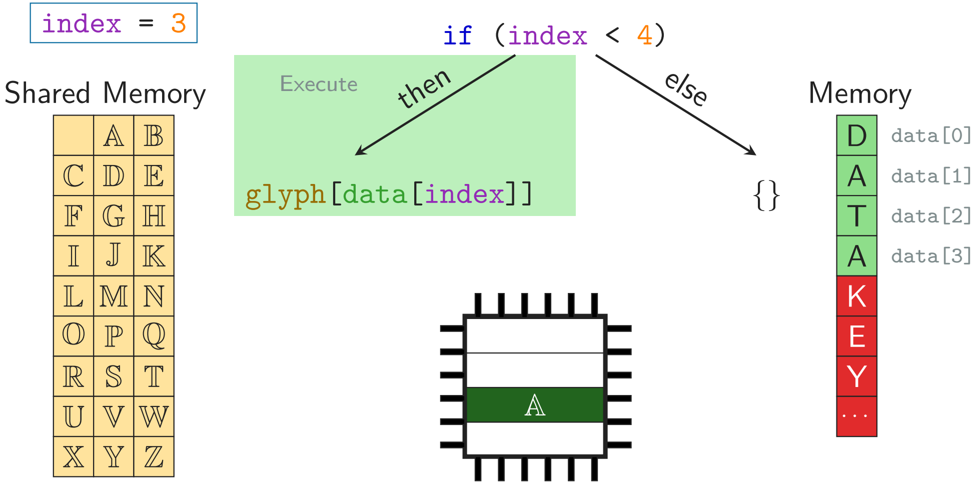
Spectre-PHT (aka Spectre Variant 1)



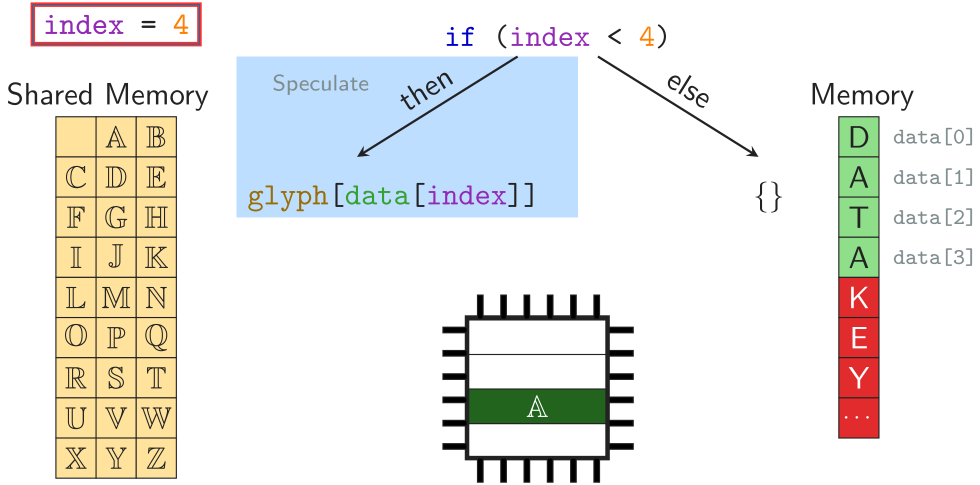
Spectre-PHT (aka Spectre Variant 1)



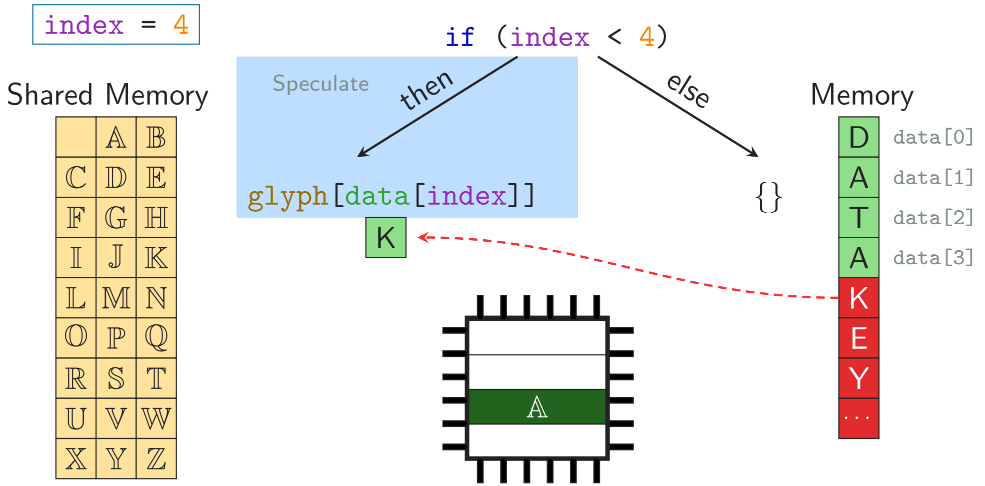
Spectre-PHT (aka Spectre Variant 1)



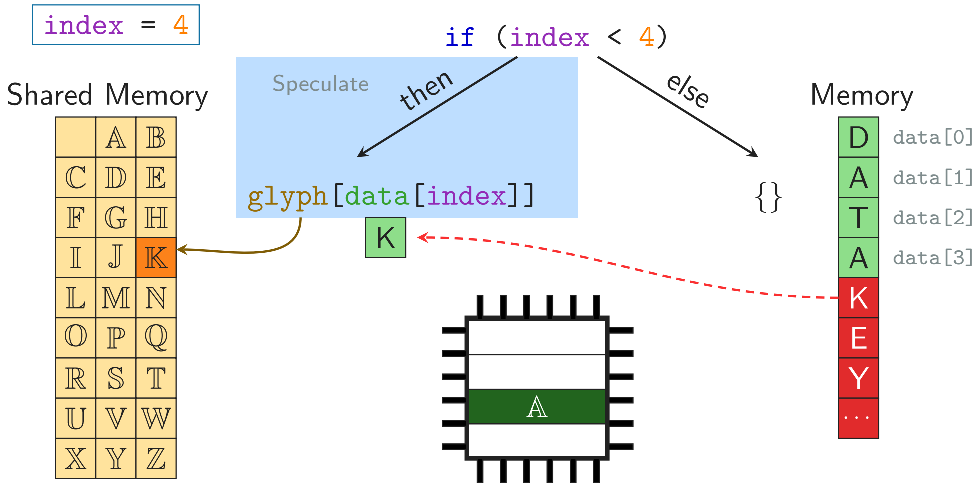
Spectre-PHT (aka Spectre Variant 1)



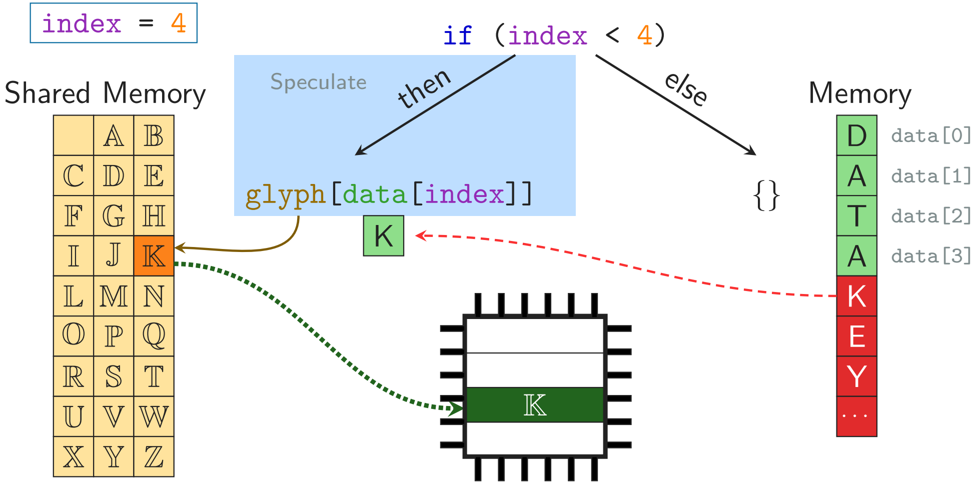
Spectre-PHT (aka Spectre Variant 1)



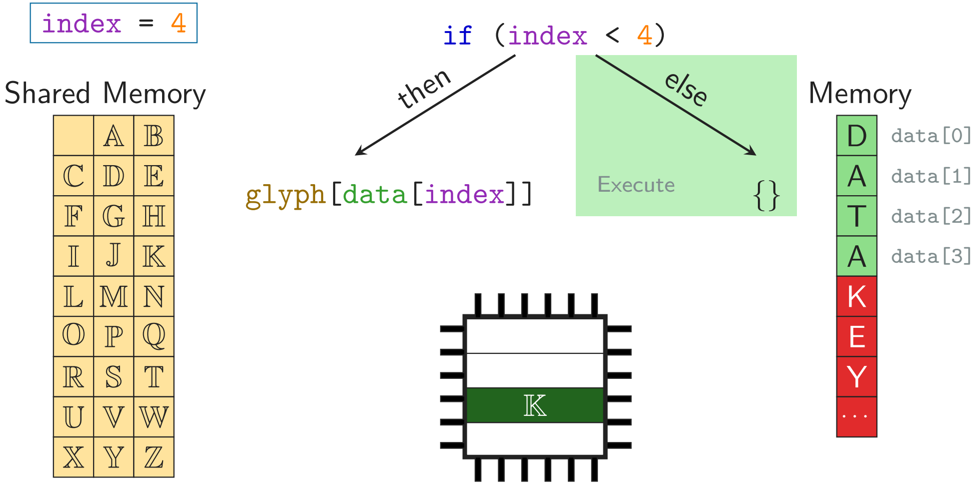
Spectre-PHT (aka Spectre Variant 1)



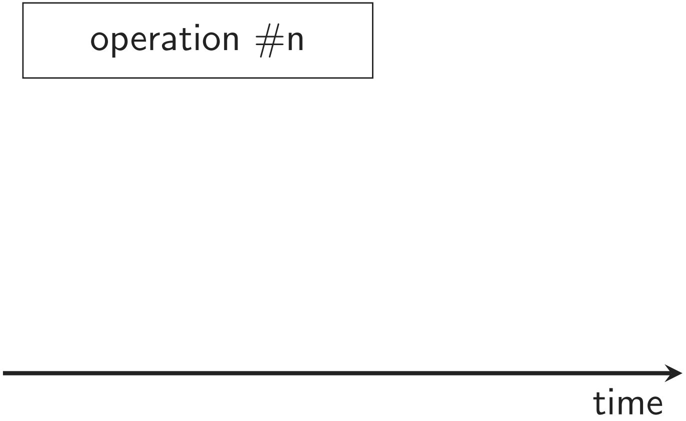
Spectre-PHT (aka Spectre Variant 1)



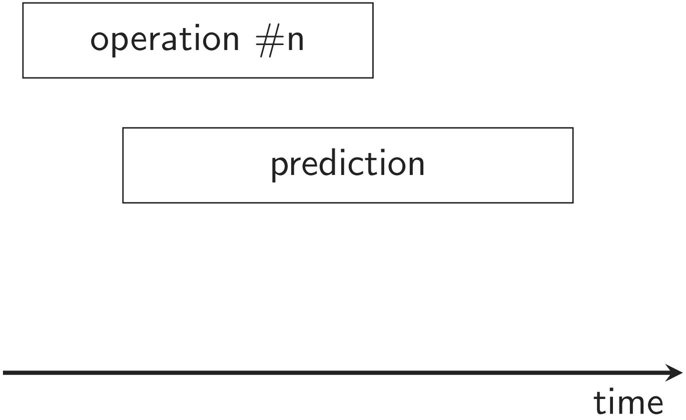
Spectre-PHT (aka Spectre Variant 1)



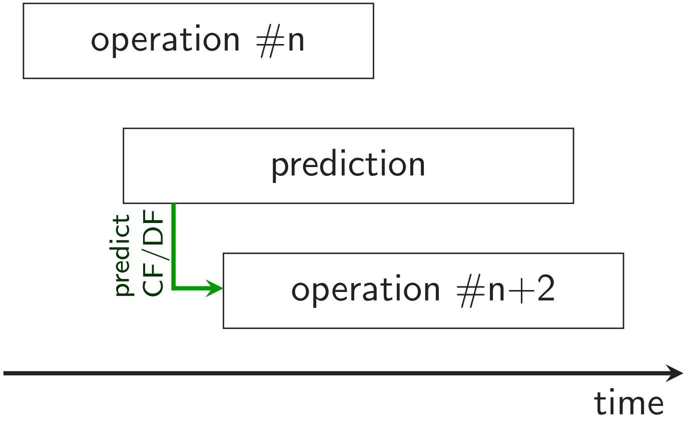
Spectre Root Cause



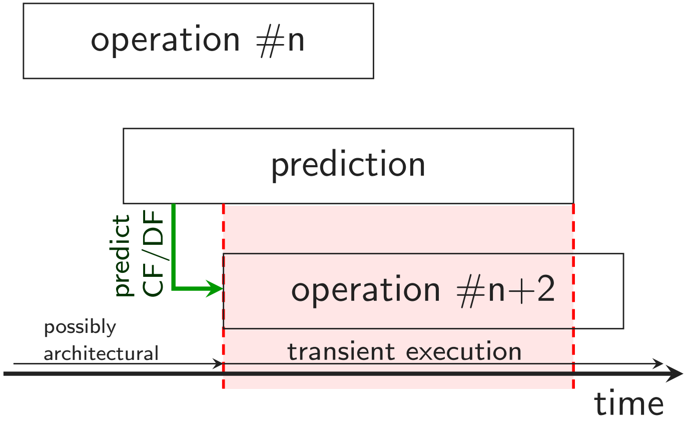
Spectre Root Cause



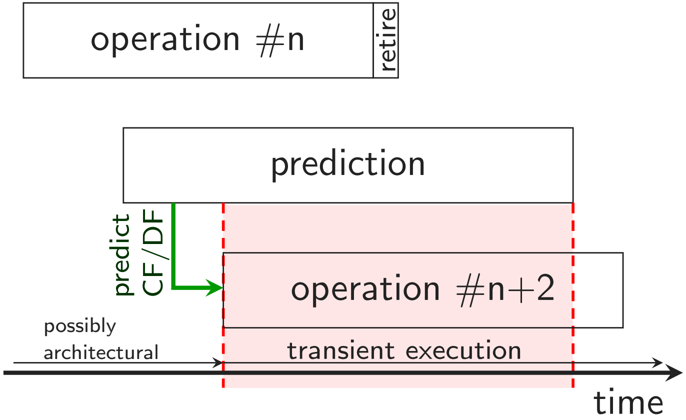
Spectre Root Cause



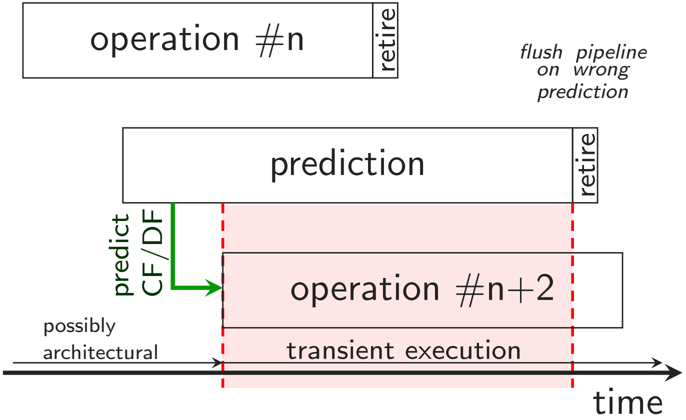
Spectre Root Cause



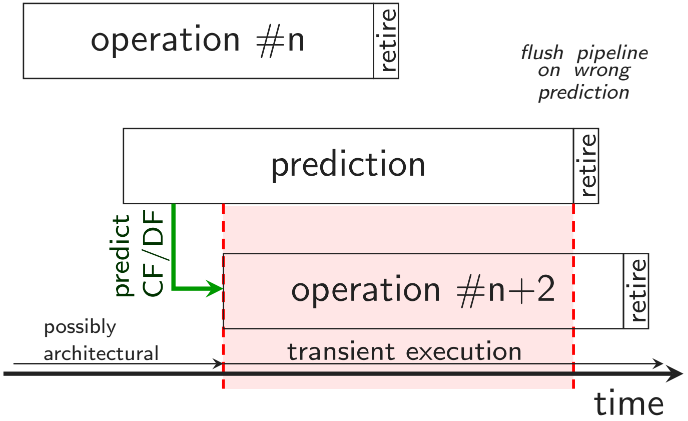
Spectre Root Cause



Spectre Root Cause



Spectre Root Cause



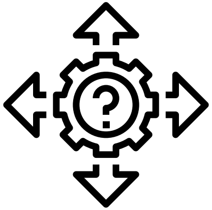
Spectre Root Cause

- Many predictors in modern CPUs

Spectre Root Cause

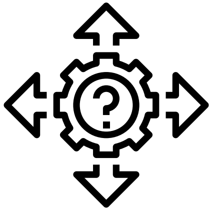
- Many predictors in modern CPUs
    - Branch taken/not taken (PHT)

Spectre Root Cause



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

Spectre Root Cause



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

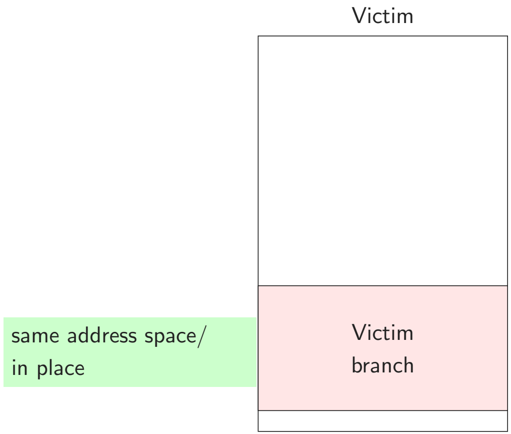
Spectre Root Cause

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

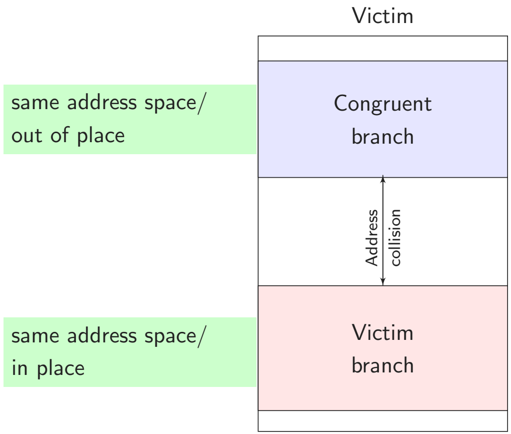
Spectre Root Cause

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

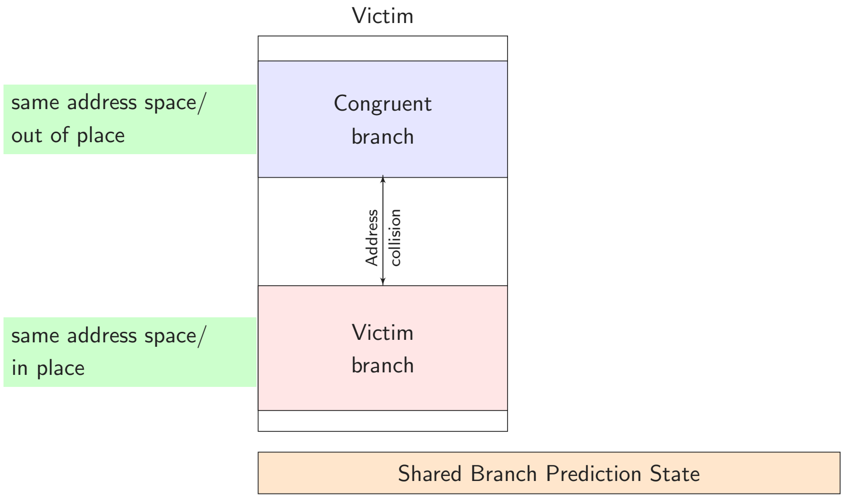
Spectre Mistraining



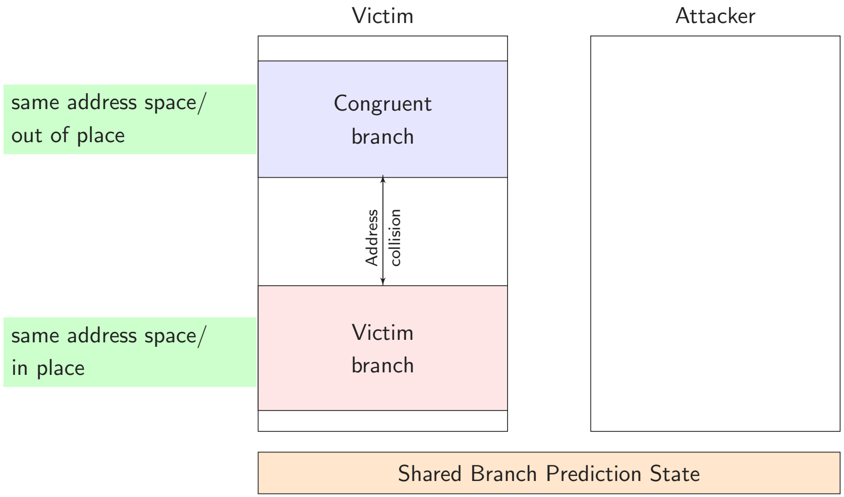
Spectre Mistraining



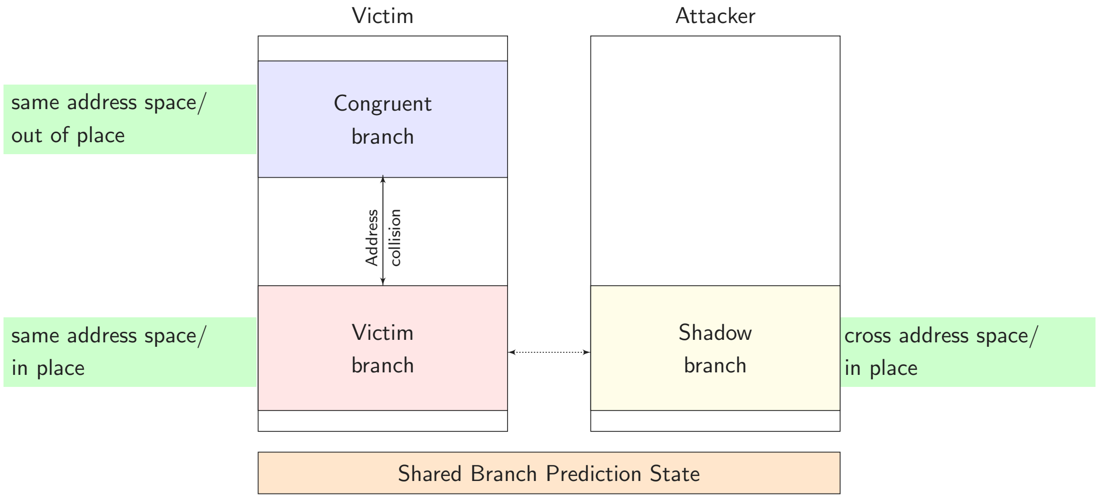
Spectre Mistraining



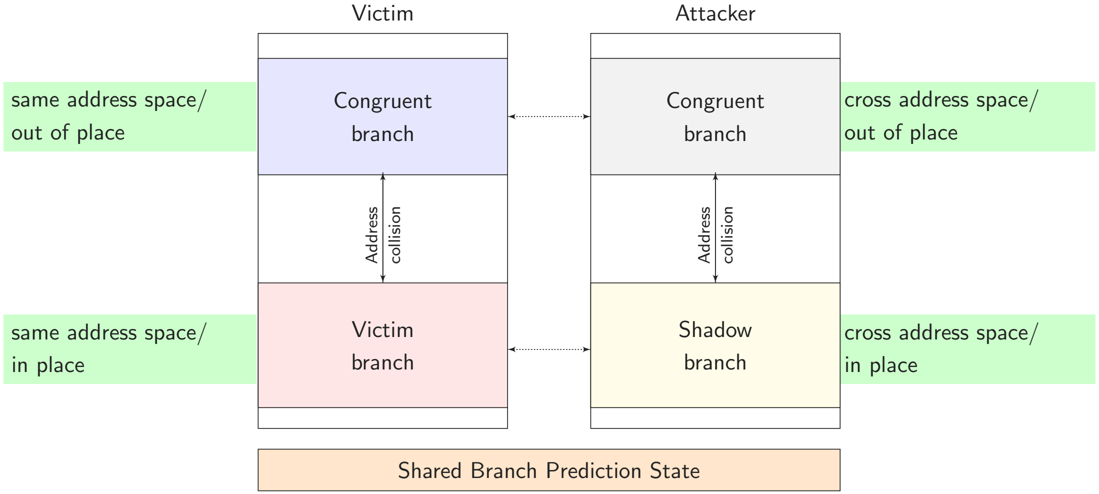
Spectre Mistraining


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

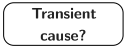
Spectre Mistraining



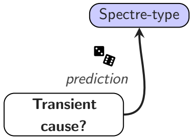
Spectre Mistraining


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

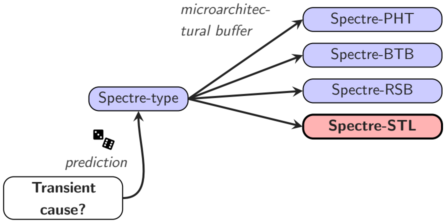
Spectre Variants



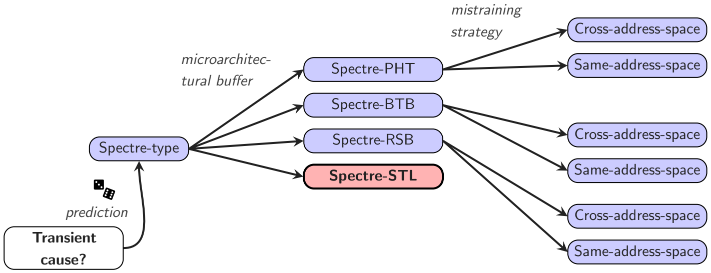
Spectre Variants



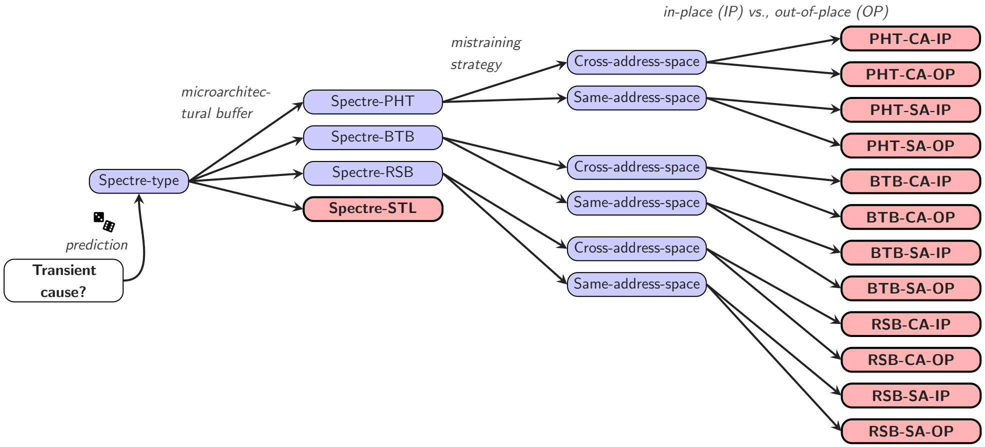
Spectre Variants



Spectre Variants



Spectre Variants



Spectre Fix

- Spectre is not a bug

Spectre Fix

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

Spectre Fix

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

→ Cannot simply fix it (as with Meltdown)

Spectre Fix

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

→ Cannot simply fix it (as with Meltdown)

- Workarounds for critical code parts

Spectre Defense Categorization

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

	Defense Attack	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		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

	Defense / Attack	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	□	□	□					◐							■	◐	■	●

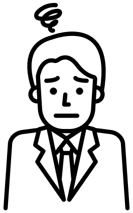
Attack is mitigated (●), partially mitigated (◐), not mitigated (○), theoretically mitigated (■), theoretically impeded (◧), not theoretically impeded (□), or out of scope (◇).

Spectre: Defense Analysis

	Defense / Attack	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	□	□	□	◇	◇	◇	◇	◐	◇	◇	◇	◇	◇	◇	■	◐	■	●

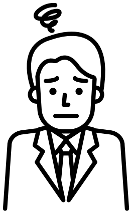
Attack is mitigated (●), partially mitigated (◐), not mitigated (○), theoretically mitigated (■), theoretically impeded (◧), not theoretically impeded (□), or out of scope (◇).

Spectre Misconceptions



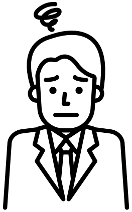
- Many countermeasures only consider the cache to get data...

Spectre Misconceptions



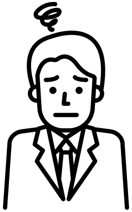
- Many countermeasures only consider the cache to get data...
- ...but there are other possibilities, e.g.,

Spectre Misconceptions



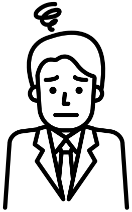
- Many countermeasures only consider the cache to get data...
- ...but there are other possibilities, e.g.,
    - Port contention (SMoTherSpectre)

Spectre Misconceptions



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

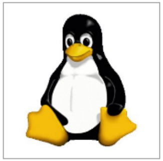
Spectre Misconceptions



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



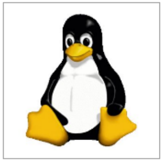
On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

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



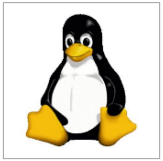
On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

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



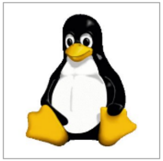
On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

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



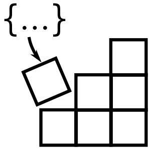
On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

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).

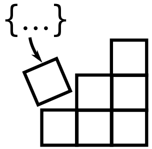
Spectre-BTB Mitigations (Software)

Retpoline (compiler extension)



Spectre-BTB Mitigations (Software)

Retpoline (compiler extension)

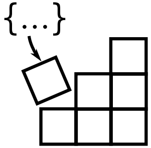


```
  push <call_target>
  call 1f
2:                      ; speculation continues here
  lfence                ; speculation barrier
  jmp 2b                ; endless loop
1:
  lea 8(%rsp), %rsp     ; restore stack pointer
  ret                   ; the call to <call_target>
```

→ Always predict to enter an endless loop

Spectre-BTB Mitigations (Software)

Retpoline (compiler extension)



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

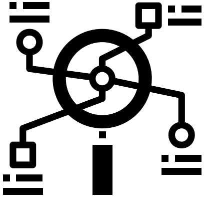
→ Always predict to enter an endless loop

- What if someone decides to fix the wrong prediction?

Spectre Mitigations

- Current mitigations are either incomplete or cost performance

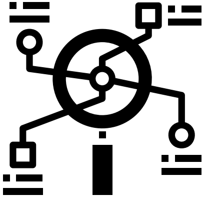
Spectre Mitigations



- Current mitigations are either incomplete or cost performance

→ More research required

Spectre Mitigations



- Current mitigations are either incomplete or cost performance

→ More research required

- Both on attacks and defenses

Spectre Mitigations

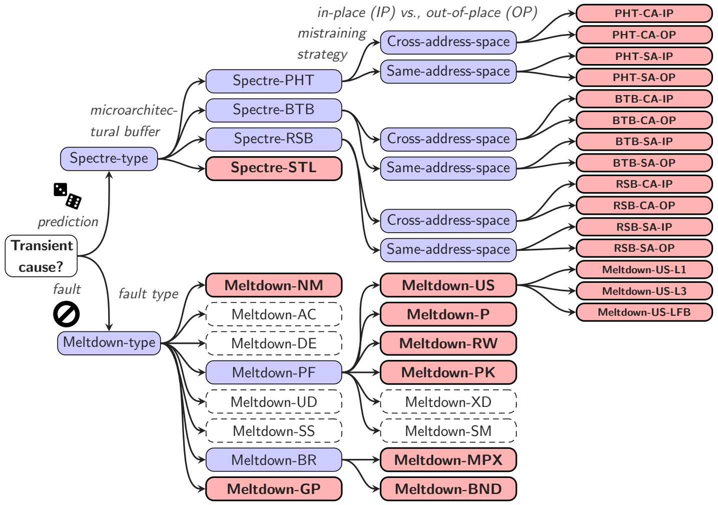
- Current mitigations are either incomplete or cost performance

→ More research required

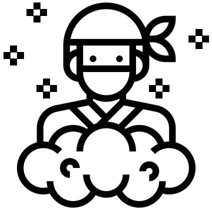
- Both on attacks and defenses

→ Efficient defenses only possible when attacks are known

Transient Execution Attacks

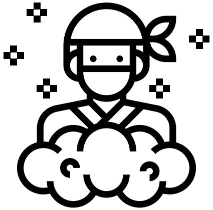


Transient Execution Attacks



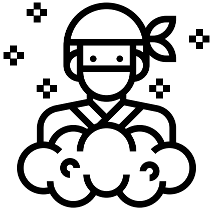
- Transient Execution Attacks are...

Transient Execution Attacks



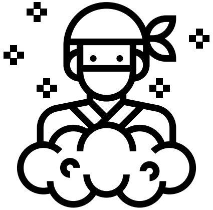
- Transient Execution Attacks are...
  - ...a novel class of attacks

Transient Execution Attacks



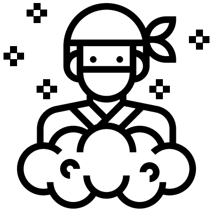
- Transient Execution Attacks are...
  - ...a novel class of attacks
  - ...extremely powerful

Transient Execution Attacks



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

Transient Execution Attacks



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

BRACE YOURSELVES
MORE BUGS ARE COMING
imgflip.com

Exploiting the Microarchitecture: Transient Execution Attacks

Michael Schwarz (@misc0110)

April 11, 2019

Graz University of Technology