Specfuscator: Evaluating Branch Removal as a Spectre Mitigation

Martin Schwarzl (@marv0x90), Claudio Canella, Daniel Gruss, Michael Schwarz

1st of March, 2021

Graz University of Technology

Motivation

- Explore a previously unexplored mitigation space against Spectre attacks - branch removal

Motivation

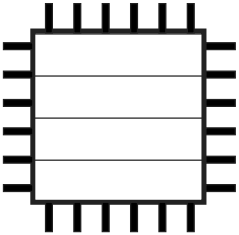
- Explore a previously unexplored mitigation space against Spectre attacks - branch removal
- Present Specfuscator, a solution based on a linearized control-flow

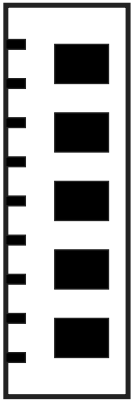
Motivation

- Explore a previously unexplored mitigation space against Spectre attacks - branch removal
- Present Specfuscator, a solution based on a linearized control-flow
- Evaluate Specfuscator on different set of use cases

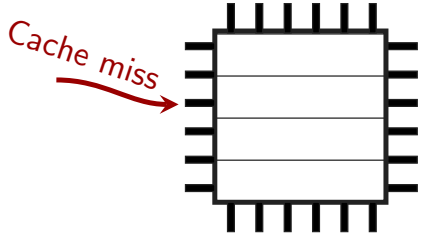
CPU Cache

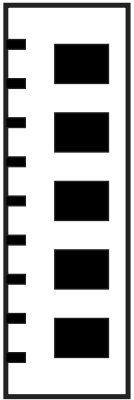
```
maccess(i);

maccess(i);
```



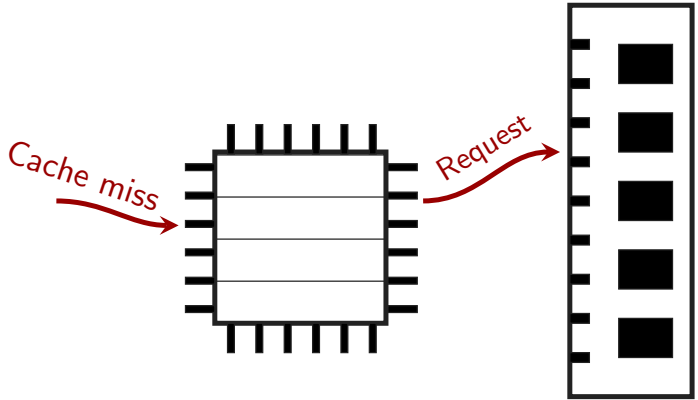


```
maccess(i);

maccess(i);
```

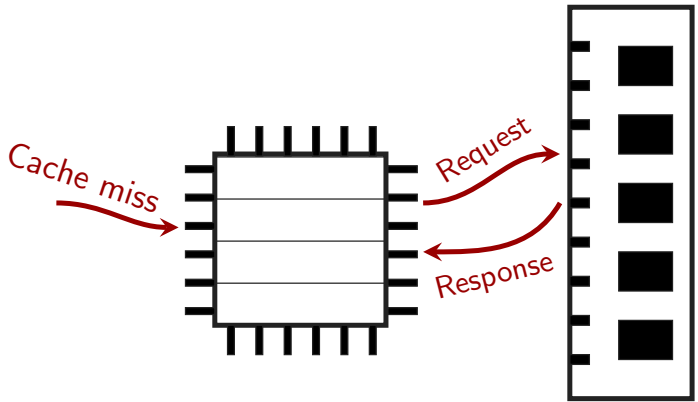




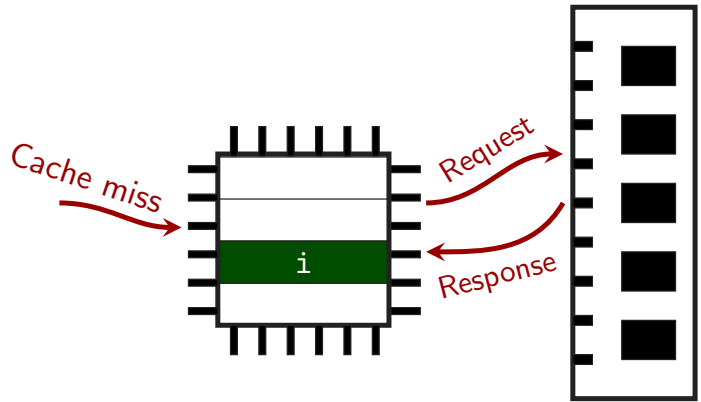
```
maccess(i);

maccess(i);
```

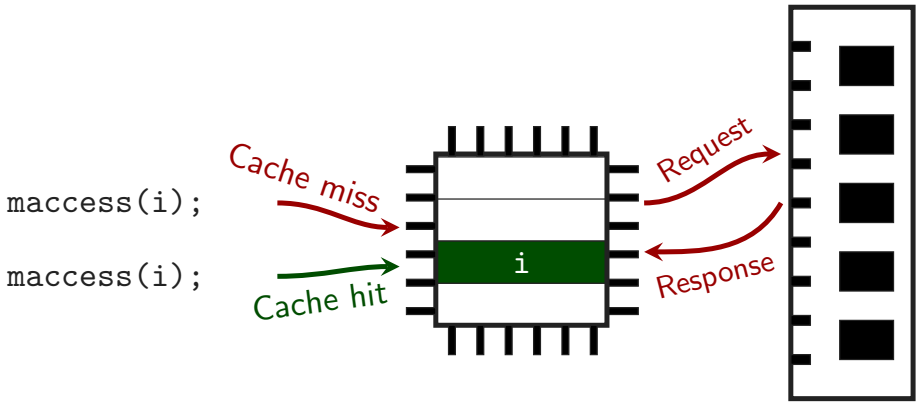


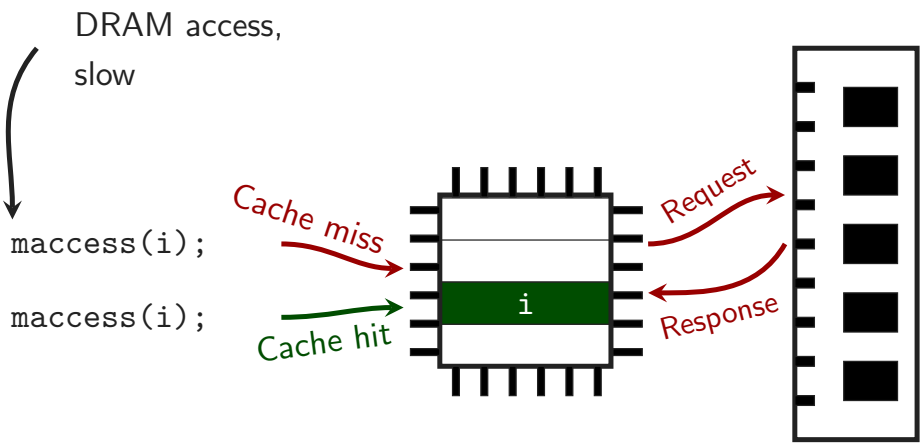
```
maccess(i);

maccess(i);
```

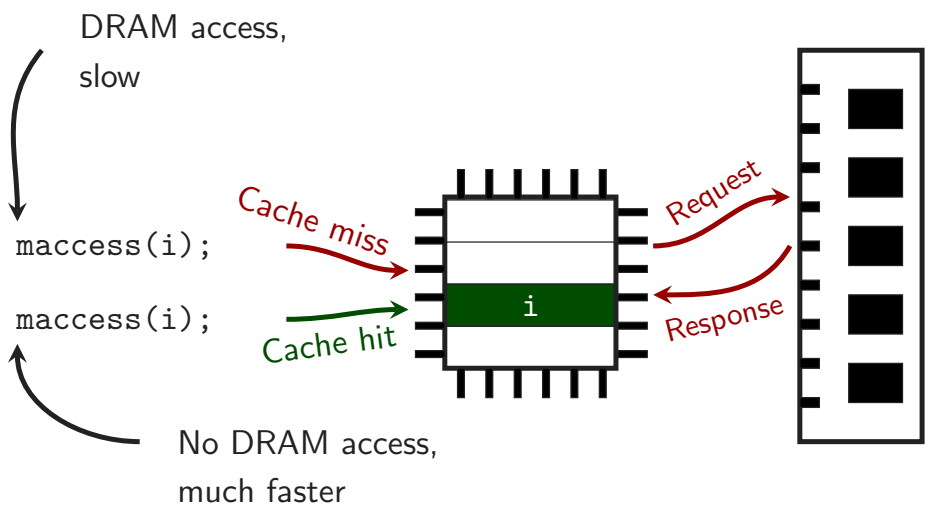


```
maccess(i);

maccess(i);
```




maccess(i);
maccess(i);
Cache miss
Cache hit
i
Request
Response

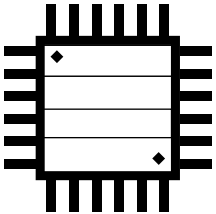

DRAM access,
slow
maccess(i);
maccess(i);
Cache miss
Cache hit
i
Request
Response


DRAM access,
slow
maccess(i);
maccess(i);
Cache miss
Cache hit
i
Request
Response
No DRAM access,
much faster

Shared Memory

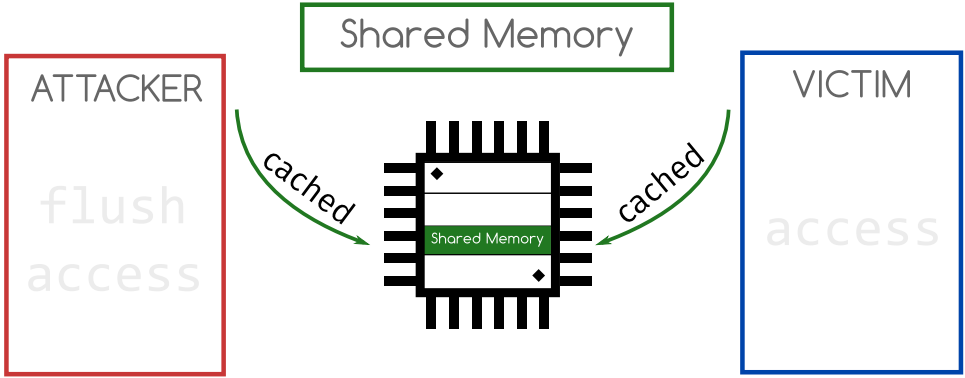
ATTACKER

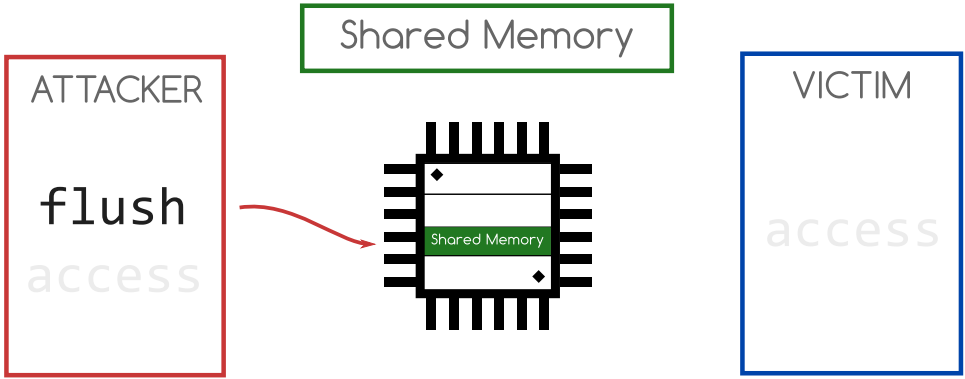
flush
access

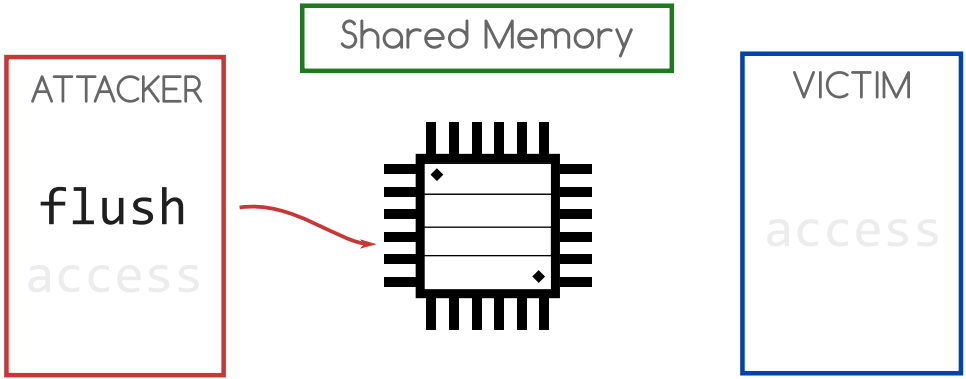


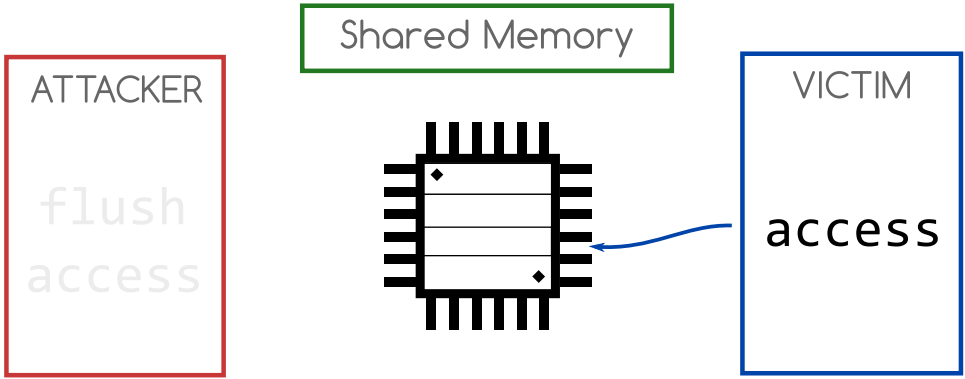
VICTIM

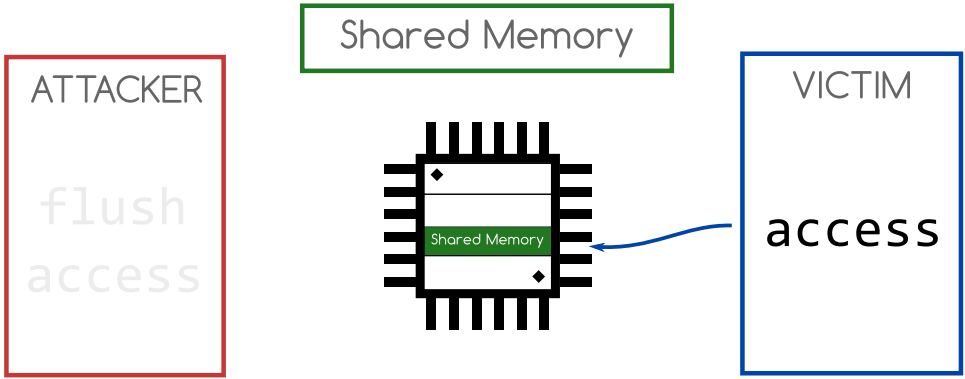
access

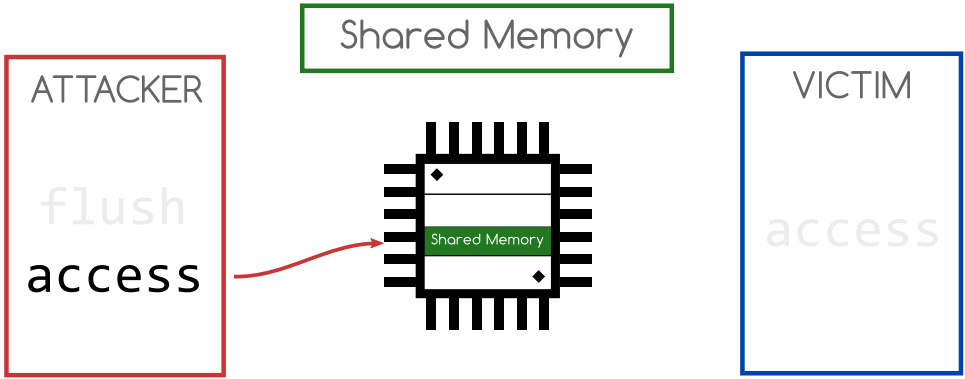

Shared Memory
ATTACKER
flush
access
cached
Shared Memory
cached
VICTIM
access

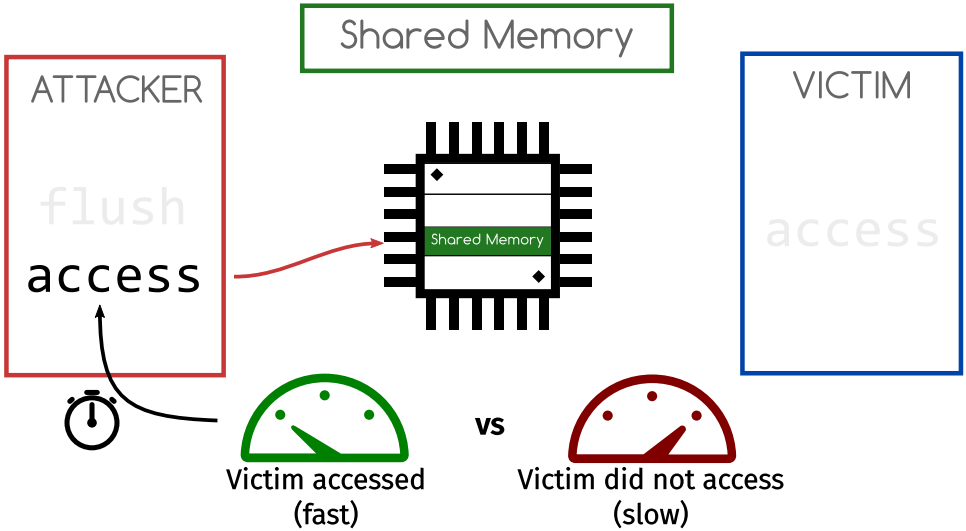

Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access


Shared Memory
ATTACKER
flush
access
VICTIM
access


Shared Memory
ATTACKER
flush
access
VICTIM
access


Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access


Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access


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

- CPU tries to predict the future, ...

- CPU tries to predict the future, ...
  - ... based on what happened in the past

- CPU tries to predict the future, ...
  - ... based on what happened in the past
- Speculative execution of instructions

- CPU tries to predict the future, ...
  - ... based on what happened in the past
- Speculative execution of instructions
- Correct prediction, ...

- CPU tries to predict the future, ...
  - ... based on what happened in the past
- Speculative execution of instructions
- Correct prediction, ...
  - ... very fast

- CPU tries to predict the future, ...
  - ... based on what happened in the past
- Speculative execution of instructions
- Correct prediction, ...
  - ... very fast
  - otherwise: Discard results

- Speculative execution is exploitable (Spectre attacks [Koc+19])

- Speculative execution is exploitable (Spectre attacks [Koc+19])
- Allow it to leak data

- Speculative execution is exploitable (Spectre attacks [Koc+19])
- Allow it to leak data
- Require a certain code snippet (gadget)

- Speculative execution is exploitable (Spectre attacks [Koc+19])
- Allow it to leak data
- Require a certain code snippet (gadget)
- Execution based on predictions from different mechanisms(PHT,BTB,RSB,STL) [Koc+19; Can+19]

- Speculative execution is exploitable (Spectre attacks [Koc+19])
- Allow it to leak data
- Require a certain code snippet (gadget)
- Execution based on predictions from different mechanisms(PHT,BTB,RSB,STL) [Koc+19; Can+19]
- Via different side channels (AVX, Port contention)

```
if (index < data_size)
{
  y = lut[data[index]*4096]
}
```

```
index = 0
```

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

then

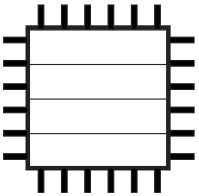
```
lut[data[index]]
```

else

```
{}
```

Lookup Table

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



Memory

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

```
index = 0
```

Lookup Table

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

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

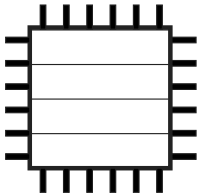
then

```
lut[data[index]]
```

else

Speculate

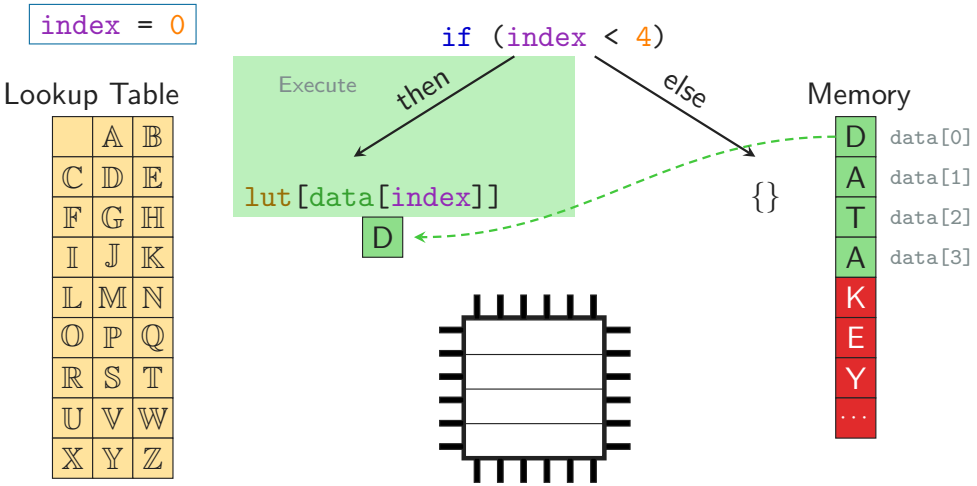
{}

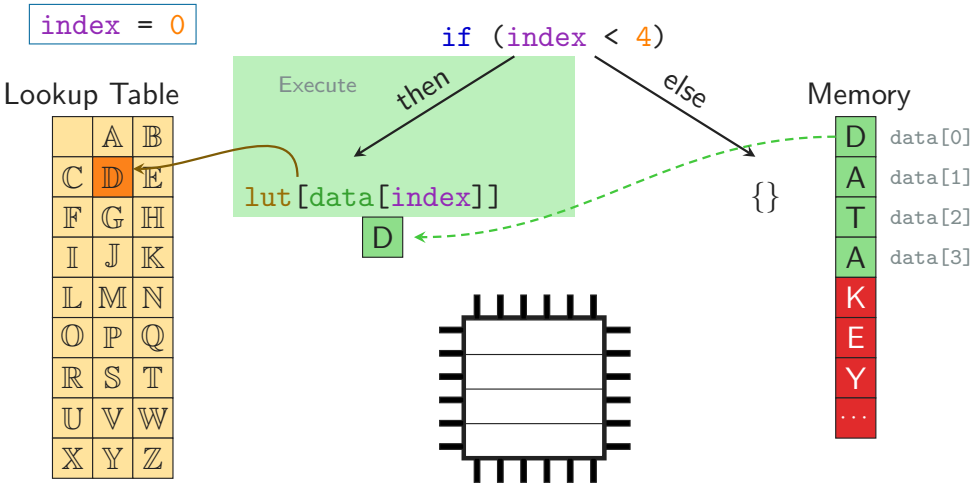


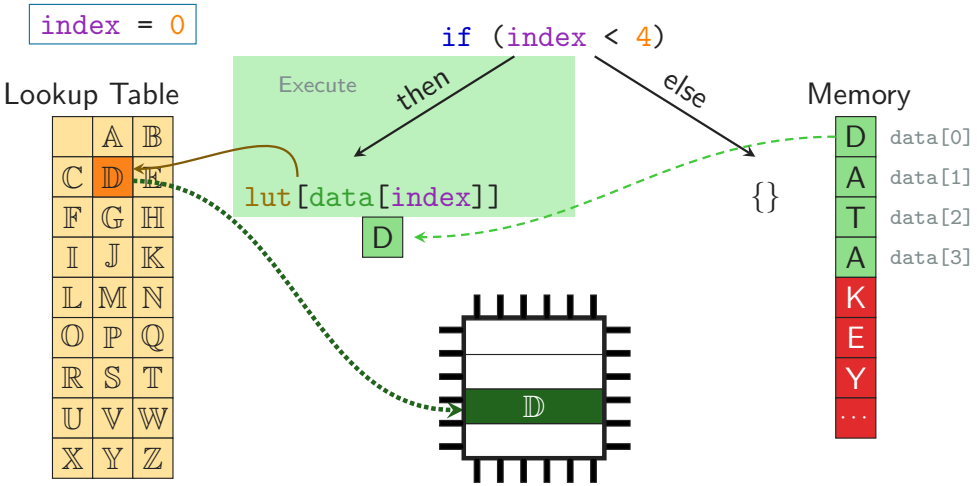
Memory

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

Spectre-PHT (aka Spectre Variant 1)




index = 0
if (index < 4)
then
else
Execute
lut[data[index]]
D
{}
Lookup Table
A B
C D E
F G H
I J K
L M N
O P Q
R S T
U V W
X Y Z
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]


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

Spectre-PHT (aka Spectre Variant 1)

```
index = 1
```

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

then

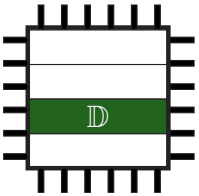
```
lut[data[index]]
```

else

```
{}
```

Lookup Table

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



Memory

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

```
index = 1
```

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

Speculate

then

else

```
lut[data[index]]
```

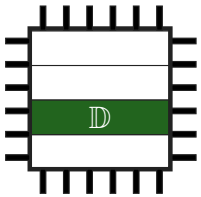
{}

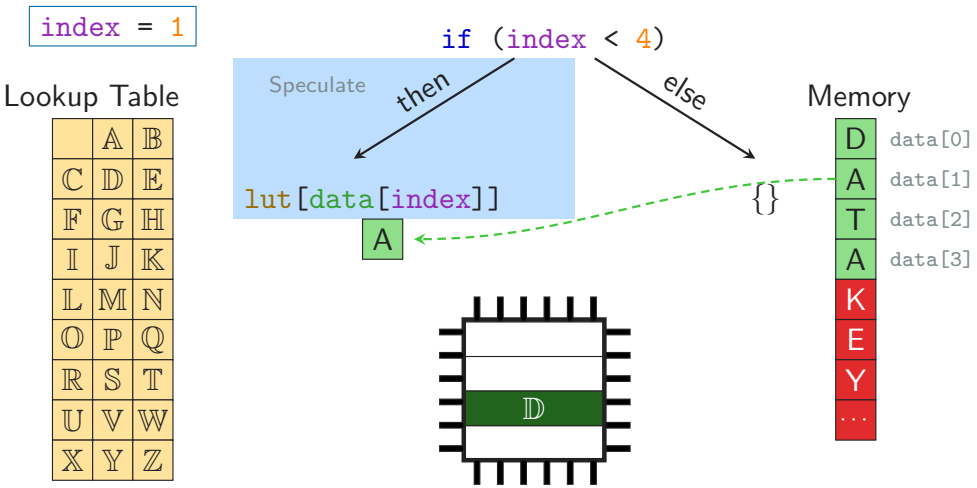
Lookup Table

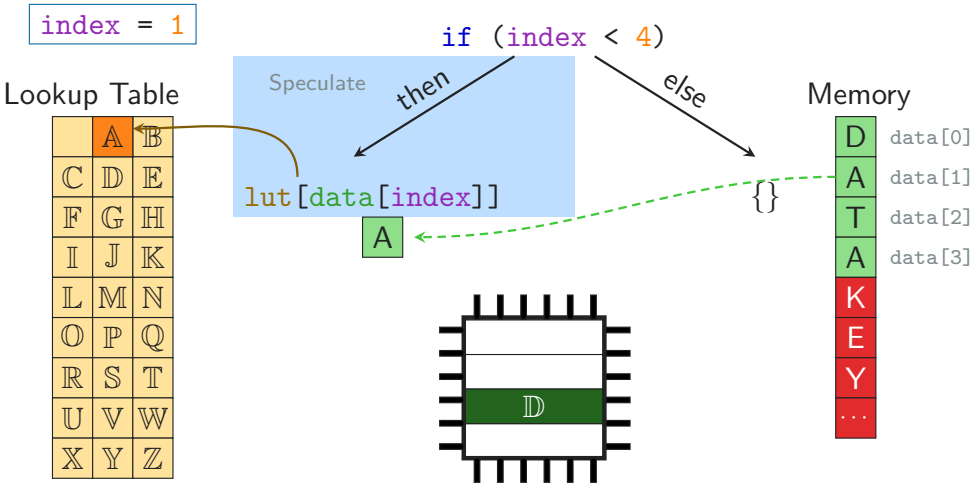
	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

Memory

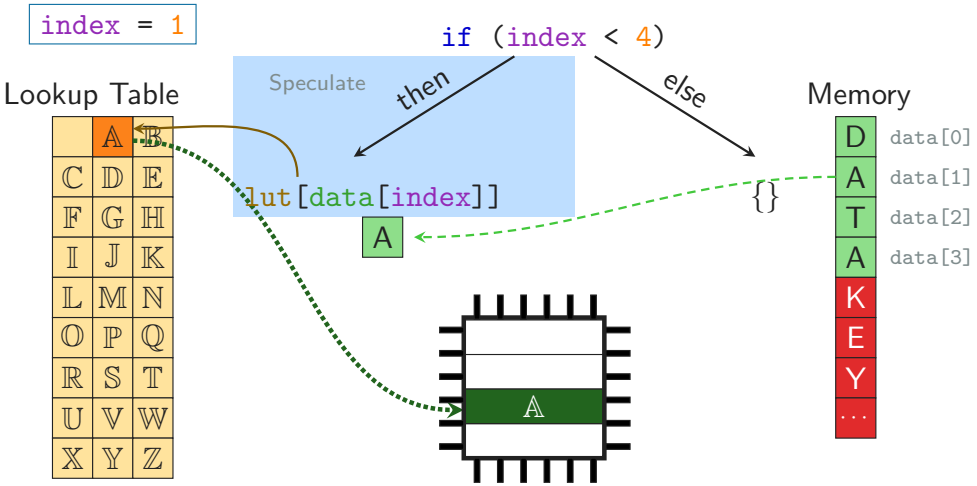
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	




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


index = 1
if (index < 4)
Speculate
then
else
Lookup Table
A B
C D E
F G H
I J K
L M N
O P Q
R S T
U V W
X Y Z
lut[data[index]]
A
{}
Memory
D data[0]
A data[1]
T data[2]
A data[3]
K
E
Y
...
D

Spectre-PHT (aka Spectre Variant 1)



```
index = 1
```

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

then

Execute

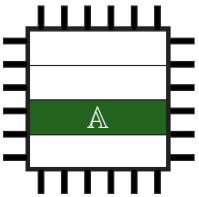
```
lut[data[index]]
```

else

{}

Lookup Table

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



Memory

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

```
index = 2
```

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

Speculate

then

else

```
lut[data[index]]
```

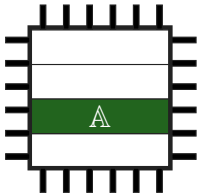
{}

Lookup Table

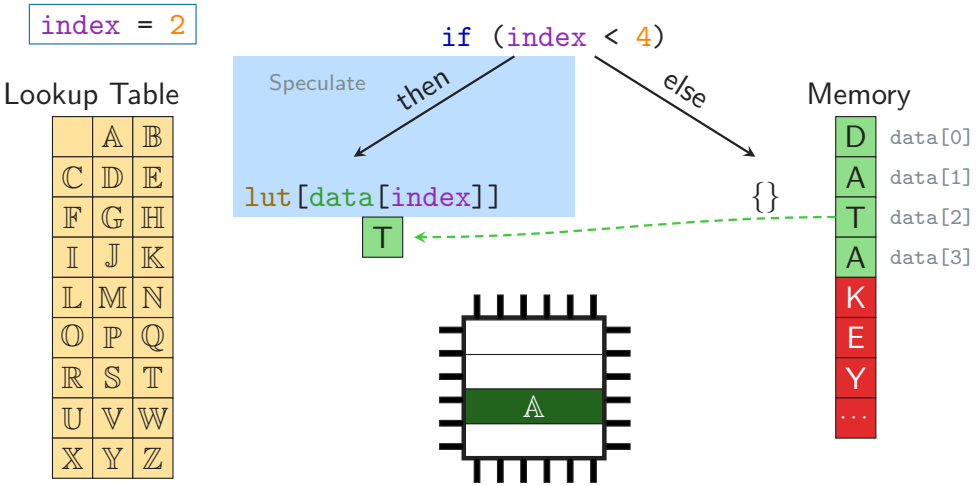
	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

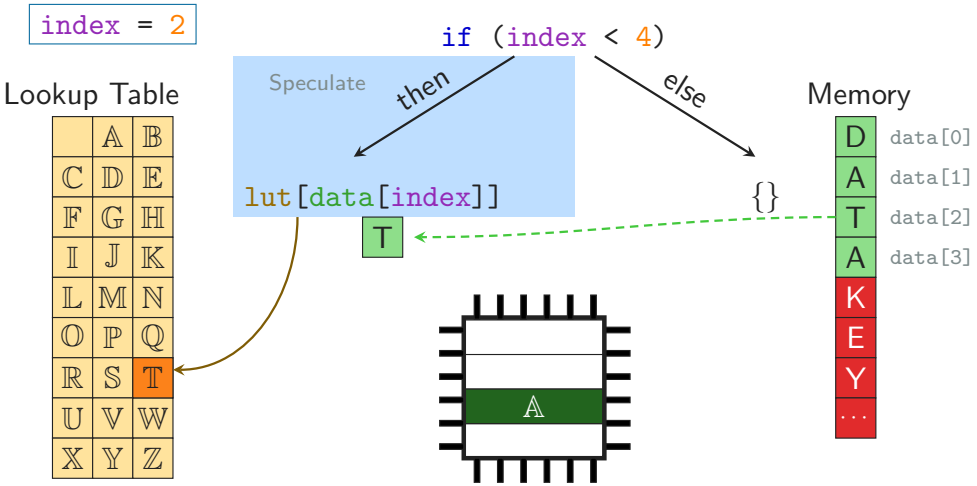
Memory

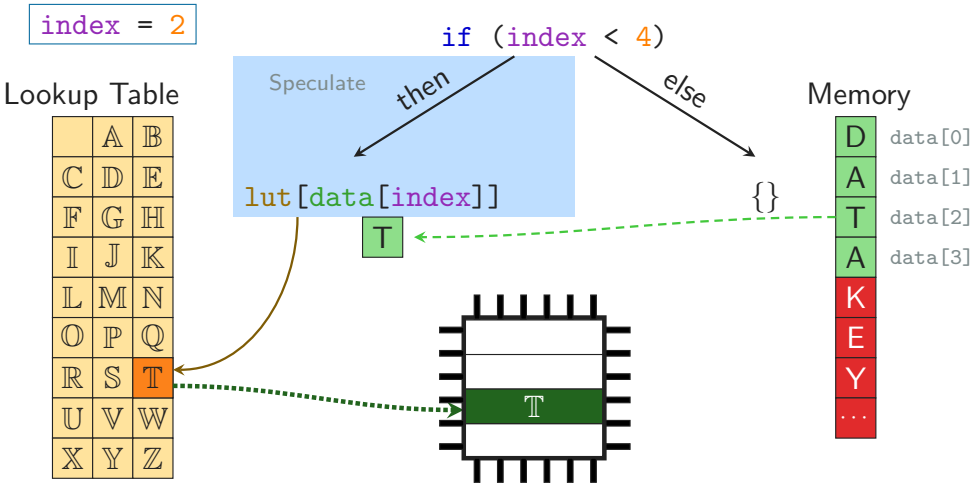
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	



Spectre-PHT (aka Spectre Variant 1)




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


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

Spectre-PHT (aka Spectre Variant 1)

```
index = 2
```

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

then

Execute

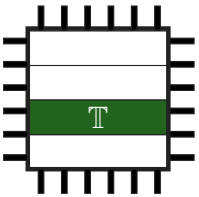
```
lut[data[index]]
```

else

{}

Lookup Table

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



Memory

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

```
index = 3
```

Lookup Table

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

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

Speculate

then

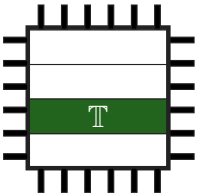
```
lut[data[index]]
```

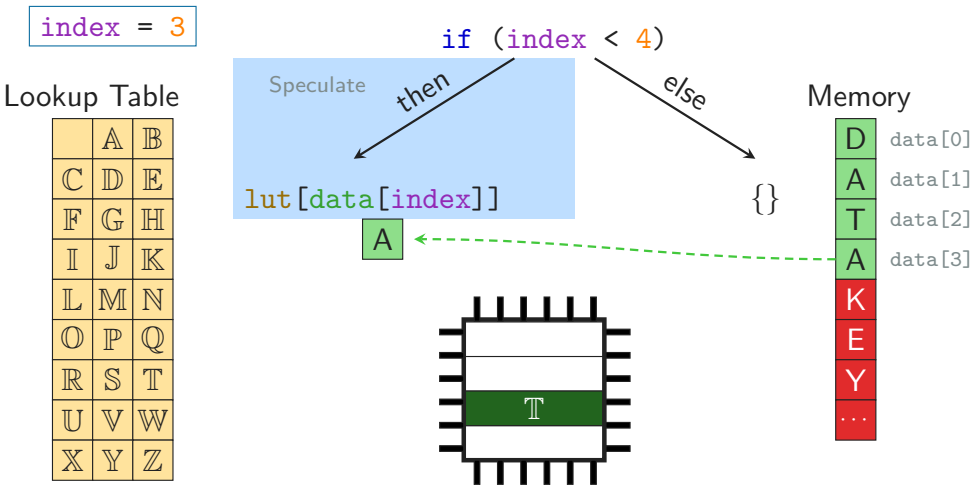
else

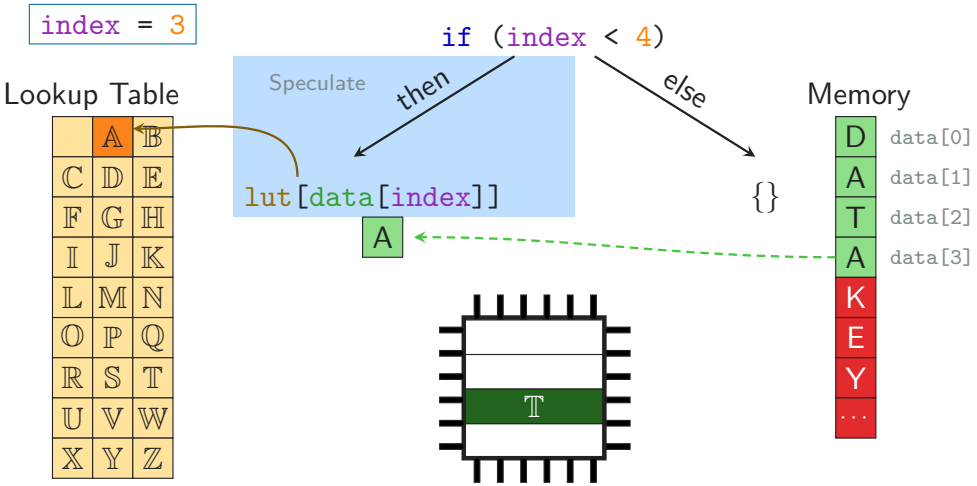
{}

Memory

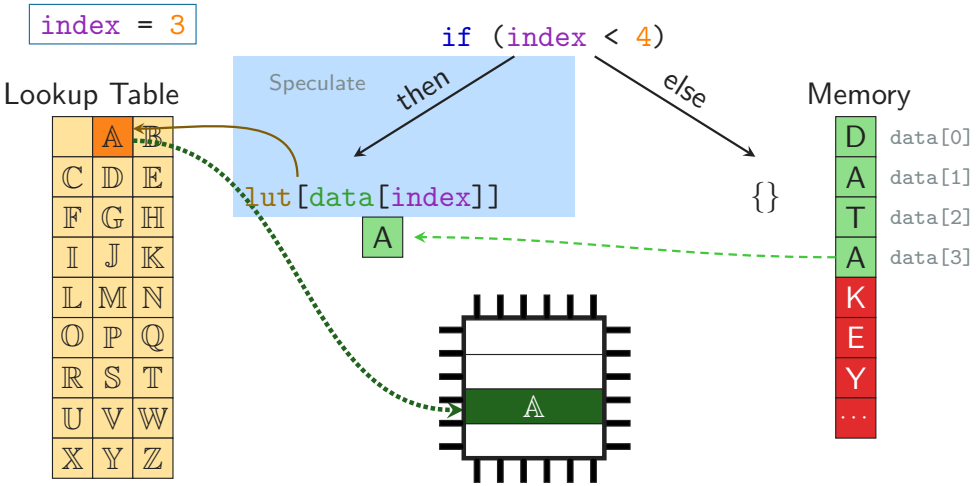
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	




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


index = 3
if (index < 4)
then
else
Speculate
Lookup Table
Memory
lut[data[index]]
A
{}
data[0]
data[1]
data[2]
data[3]
T

Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)

```
index = 3
```

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

then

Execute

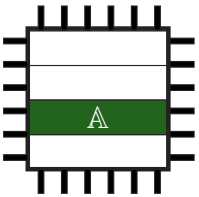
```
lut[data[index]]
```

else

{}

Lookup Table

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



Memory

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

```
index = 4
```

Lookup Table

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

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

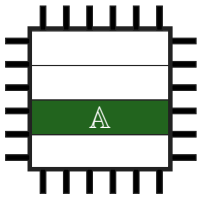
Speculate

then

```
lut[data[index]]
```

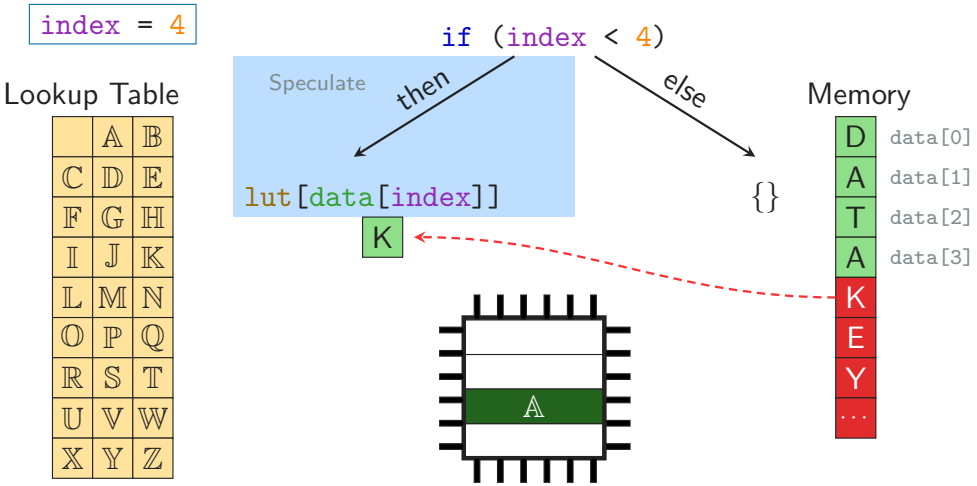
else

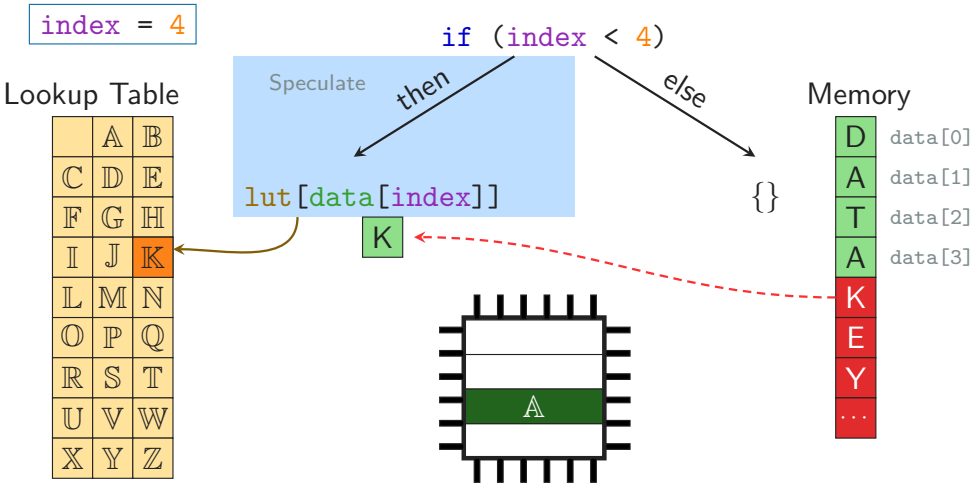
{}



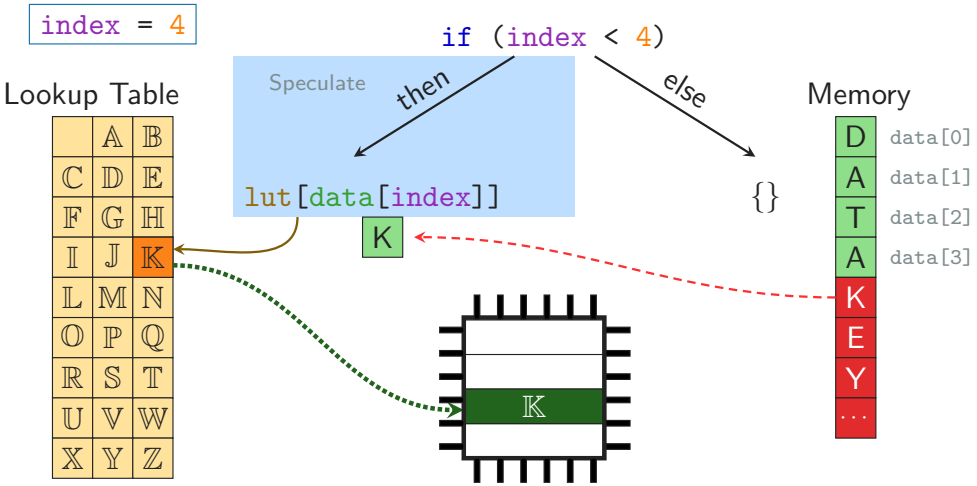
Memory

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


index = 4
Lookup Table
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
if (index < 4)
Speculate
then
else
lut[data[index]]
K
{}
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]
A


index = 4
if (index < 4)
then
else
Speculate
lut[data[index]]
K
{}
Lookup Table
Memory
data[0]
data[1]
data[2]
data[3]
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
D
A
T
A
K
E
Y
...
A

Spectre-PHT (aka Spectre Variant 1)



Spectre-PHT (aka Spectre Variant 1)

```
index = 4
```

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

then

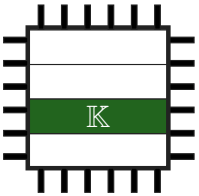
```
lut[data[index]]
```

else

Execute {}

Lookup Table

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



Memory

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

Application-Level

Spectre mitigations

Application-Level

System-Level

Spectre mitigations

Application-Level

System-Level

Hardware-Level

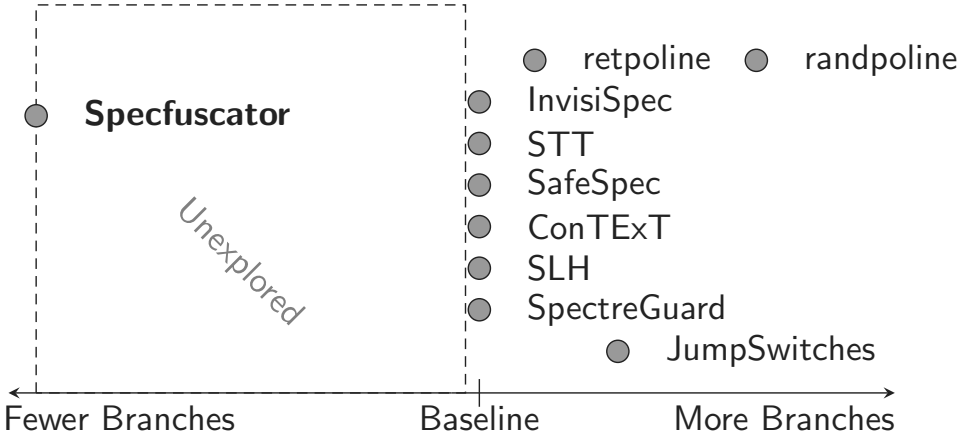
Spectre mitigations

Application-Level

System-Level

Hardware-Level

Placement of our work



- `mov` instruction is turing-complete [Ste13]

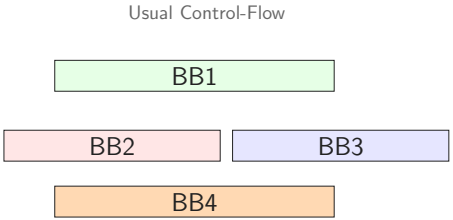
- `mov` instruction is turing-complete [Ste13]
- *M/o/Vfuscator* [Chr15] is a x86_32-bit mov-only compiler (LCC)

- `mov` instruction is turing-complete [Ste13]
- *M/o/Vfuscator* [Chr15] is a x86_32-bit mov-only compiler (LCC)
- Compiled program is mov-only and the control-flow linearized

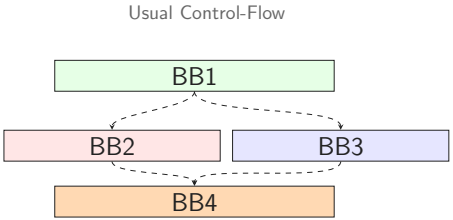
- `mov` instruction is turing-complete [Ste13]
- *M/o/Vfuscator* [Chr15] is a x86_32-bit mov-only compiler (LCC)
- Compiled program is mov-only and the control-flow linearized
- Arithmetics performed via lookup-tables

- `mov` instruction is turing-complete [Ste13]
- *M/o/Vfuscator* [Chr15] is a x86_32-bit mov-only compiler (LCC)
- Compiled program is mov-only and the control-flow linearized
- Arithmetics performed via lookup-tables
- Low-performance solution *i.e.*, Doom with one frame every 7 hours [Chr15]

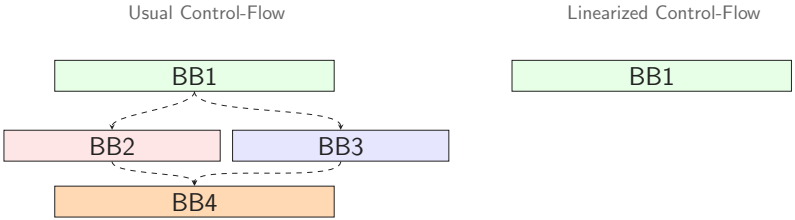
Linearized control-flow



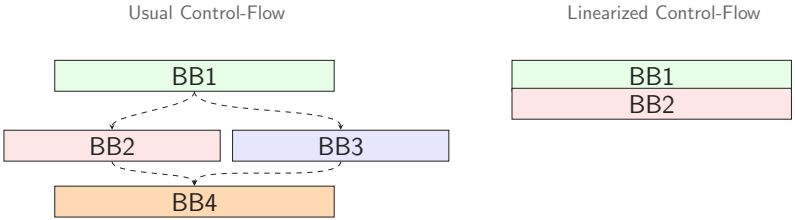
Linearized control-flow



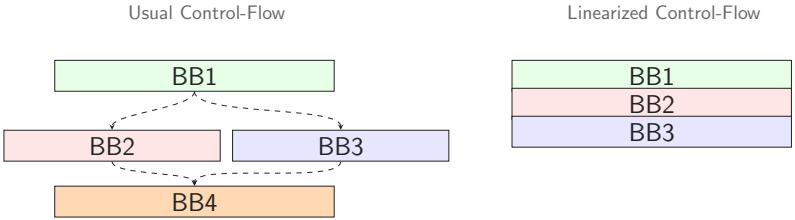
Linearized control-flow



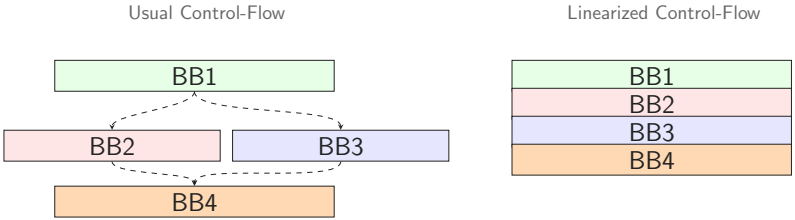
Linearized control-flow



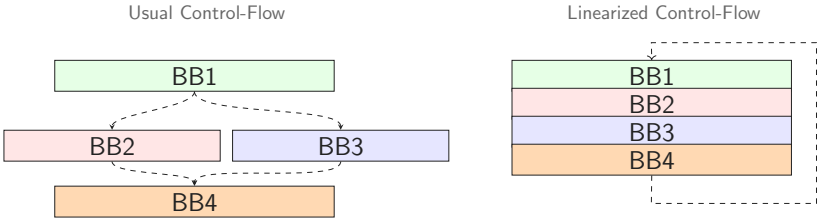
Linearized control-flow



Linearized control-flow



Linearized control-flow



- Internal calls:

- Internal calls:
  - `mov %eax,%cs` at the end of the program triggers SIGILL

- Internal calls:
  - `mov %eax,%cs` at the end of the program triggers SIGILL
  - target register is used and selected target is checked for each block

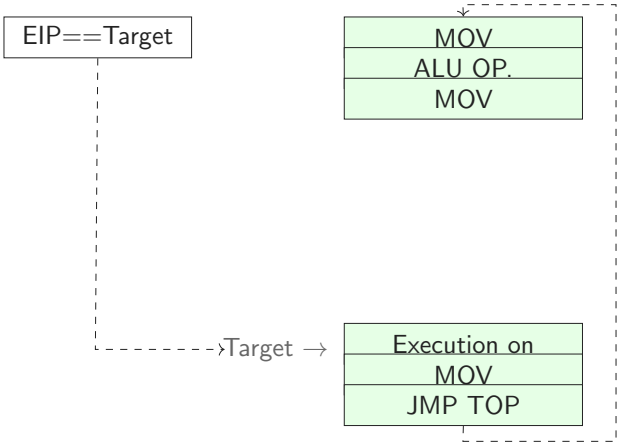
- Internal calls:
  - `mov %eax,%cs` at the end of the program triggers SIGILL
  - target register is used and selected target is checked for each block
  - SIGILL handler is installed and program starts to begin

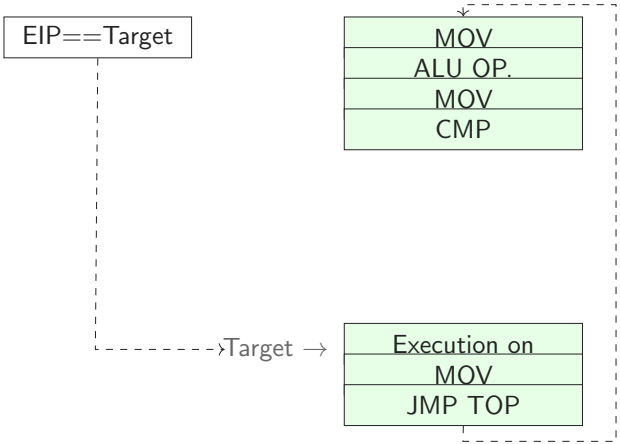
- Internal calls:
  - `mov %eax,%cs` at the end of the program triggers SIGILL
  - target register is used and selected target is checked for each block
  - SIGILL handler is installed and program starts to begin
- Library calls:

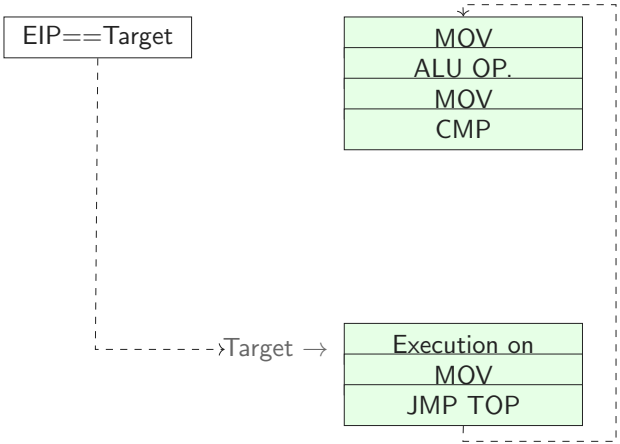
- Internal calls:
  - `mov %eax,%cs` at the end of the program triggers SIGILL
  - target register is used and selected target is checked for each block
  - SIGILL handler is installed and program starts to begin
- Library calls:
  - `mov (0),%eax` - Null Pointer Dereference

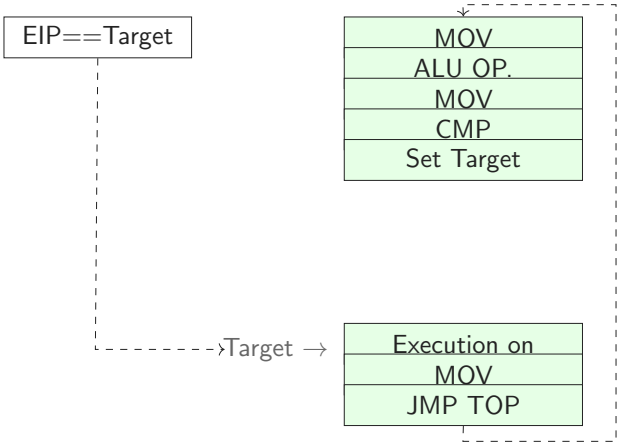
- Internal calls:
  - `mov %eax,%cs` at the end of the program triggers SIGILL
  - target register is used and selected target is checked for each block
  - SIGILL handler is installed and program starts to begin
- Library calls:
  - `mov (0),%eax` - Null Pointer Dereference
  - SIGSEGV handler is installed and dispatcher to library function is used

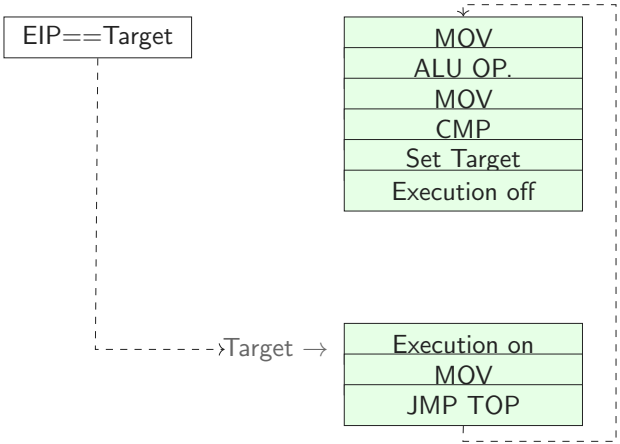
- Internal calls:
  - `mov %eax,%cs` at the end of the program triggers SIGILL
  - target register is used and selected target is checked for each block
  - SIGILL handler is installed and program starts to begin
- Library calls:
  - `mov (0),%eax` - Null Pointer Dereference
  - SIGSEGV handler is installed and dispatcher to library function is used
  - stack needs to be prepared accordingly to calling convention

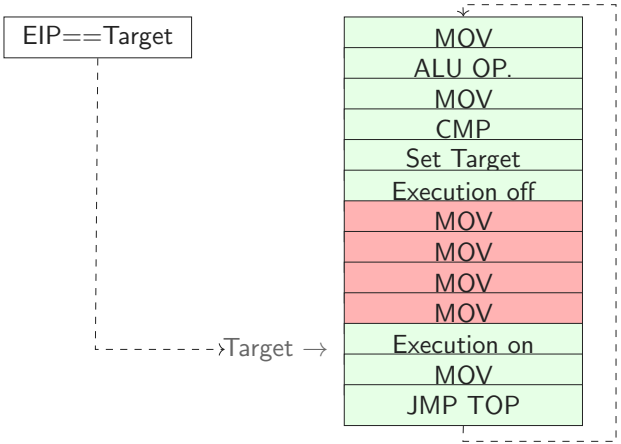

EIP==Target
MOV
ALU OP.
MOV
Target →
Execution on
MOV
JMP TOP


EIP==Target
MOV
ALU OP.
MOV
CMP
Target →
Execution on
MOV
JMP TOP


EIP==Target
MOV
ALU OP.
MOV
CMP
Target →
Execution on
MOV
JMP TOP


EIP==Target
MOV
ALU OP.
MOV
CMP
Set Target
Target →
Execution on
MOV
JMP TOP


EIP==Target
MOV
ALU OP.
MOV
CMP
Set Target
Execution off
Target →
Execution on
MOV
JMP TOP


EIP==Target
Target →
MOV
ALU OP.
MOV
CMP
Set Target
Execution off
MOV
MOV
MOV
MOV
Execution on
MOV
JMP TOP

- Based on *M/o/Vfuscator* but with optimizations:

- Based on *M/o/Vfuscator* but with optimizations:
- Instead of `SIGILL` at the end we use a direct jump

- Based on *M/o/Vfuscator* but with optimizations:
- Instead of `SIGILL` at the end we use a direct jump
- Arithmetics are performed using the actual x86 instructions *i.e.*, *addl*, *subl* etc.

- Based on *M/o/Vfuscator* but with optimizations:
- Instead of `SIGILL` at the end we use a direct jump
- Arithmetics are performed using the actual x86 instructions *i.e.*, *addl*, *subl* etc.
- Strip the binary and remove all lookup tables to reduce the binary size

- Based on *M/o/Vfuscator* but with optimizations:
- Instead of `SIGILL` at the end we use a direct jump
- Arithmetics are performed using the actual x86 instructions *i.e.*, *addl*, *subl* etc.
- Strip the binary and remove all lookup tables to reduce the binary size
- Exploit x86 addressing modes to save additional loads

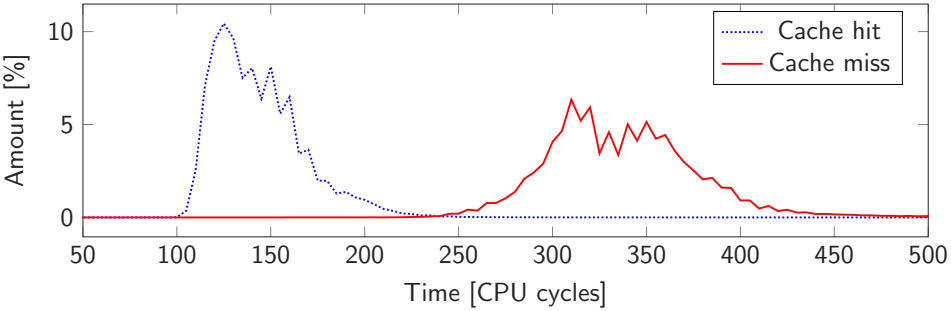
- Based on *M/o/Vfuscator* but with optimizations:
- Instead of `SIGILL` at the end we use a direct jump
- Arithmetics are performed using the actual x86 instructions *i.e.*, *addl*, *subl* etc.
- Strip the binary and remove all lookup tables to reduce the binary size
- Exploit x86 addressing modes to save additional loads
- `cmp` instruction is used and the emulated cflags are set

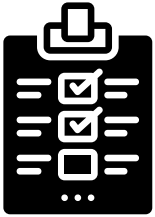
- Specfuscator guarantees code free of Spectre gadgets by design

- Specfuscator guarantees code free of Spectre gadgets by design
- We compiled each of the Spectre Pocs from Kocher

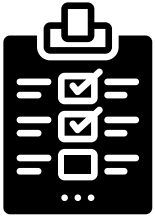
- Specfuscator guarantees code free of Spectre gadgets by design
- We compiled each of the Spectre Pocs from Kocher
- As expected, the PoCs did not leak any data

Flush+Reload in movfuscated code

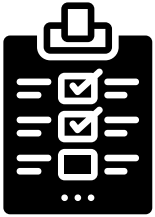




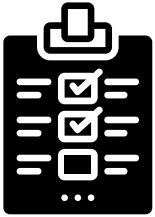
- Choose a different set of test open-source programs



- Choose a different set of test open-source programs
- Clang vs. Clang with lfences vs. LCC vs. *M/o/Vfuscator* vs. Specfuscator

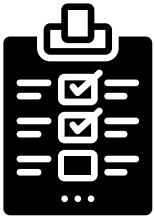


- Choose a different set of test open-source programs
- Clang vs. Clang with lfences vs. LCC vs. *M/o/Vfuscator* vs. Specfuscator
- We compare the `runtime`,`compile time` and `binary size` and calculate the overhead factors



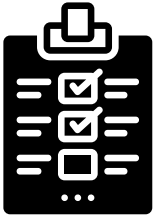
- Choose a different set of test open-source programs
- Clang vs. Clang with lfences vs. LCC vs. *M/o/Vfuscator* vs. Specfuscator
- We compare the `runtime`,`compile time` and `binary size` and calculate the overhead factors

Performance Evaluation: Runtime overhead factor



Test program	M/o/Vfuscator	Specfuscator	Clang (fences)	LCC	Clang (baseline)
	times	times	times	times	baseline
aes	424.17	221.53	1.31	1.17	1.13 ms
hello	1.10	1.11	1.00	1.04	0.89 ms
maze	310.03	88.98	1.10	1.13	0.97 ms
nqueens	319.84	234.46	1.99	4.99	1.89 ms
prime	980.27	161.59	1.93	0.96	1.65 ms
s2	46085.82	981.20	20.89	26.64	0.71 ms
sudoku	656.91	149.69	2.15	1.17	1.13 ms

Performance Evaluation: Compile Time



Test program	M/o/Vfuscator		Specfuscator		Clang (fences)		LCC		Clang (baseline)	
	time	size	time	size	time	size	time	size	time	size
aes	4.80	218.69	2.95	151.15	1.20	1.00	0.53	1.01	101.89 ms	33.21 kB
hello	2.23	388.28	1.86	279.18	1.07	1.01	0.71	0.89	38.36 ms	13.62 kB
maze	3.93	394.09	2.12	274.96	1.05	1.01	0.63	0.86	46.80 ms	13.82 kB
nqueens	2.39	386.75	2.05	278.22	1.17	1.01	0.75	0.88	40.19 ms	13.64 kB
prime	2.39	389.97	1.81	279.47	1.06	1.01	0.62	0.89	39.02 ms	13.64 kB
s2	2.87	395.22	1.89	279.72	1.00	1.01	0.78	0.89	39.34 ms	13.62 kB
sudoku	3.47	398.10	2.05	280.39	1.10	1.01	0.68	0.91	37.76 ms	14.00 kB

Discussion

- We explored a new path to mitigate Spectre by removing branches

- We explored a new path to mitigate Spectre by removing branches
- Specfuscator is the most radical solution

- We explored a new path to mitigate Spectre by removing branches
- Specfuscator is the most radical solution
- The performance overhead heavily depends on the compiled program (from factor 1.05 to 22k)

- We explored a new path to mitigate Spectre by removing branches
- Specfuscator is the most radical solution
- The performance overhead heavily depends on the compiled program (from factor 1.05 to 22k)
- There is space to research in this unexplored direction

- We explored a new path to mitigate Spectre by removing branches
- Specfuscator is the most radical solution
- The performance overhead heavily depends on the compiled program (from factor 1.05 to 22k)
- There is space to research in this unexplored direction

Conclusion

- Branch removal (Control-flow linearization) does mitigate Spectre attacks

Conclusion

- Branch removal (Control-flow linearization) does mitigate Spectre attacks
- Overhead varies depending on the program

Conclusion

- Branch removal (Control-flow linearization) does mitigate Spectre attacks
- Overhead varies depending on the program
- Branchless software might be considered as an alternative mitigation

Conclusion

- Branch removal (Control-flow linearization) does mitigate Spectre attacks
- Overhead varies depending on the program
- Branchless software might be considered as an alternative mitigation

Specfuscator: Evaluating Branch Removal as a Spectre Mitigation

Martin Schwarzl (@marv0x90), Thomas Schuster, Michael Schwarz, Daniel Gruss

1st of March, 2021

Graz University of Technology

References

C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss. A Systematic Evaluation of Transient Execution Attacks and Defenses. In: USENIX Security Symposium. Extended classification tree and PoCs at https://transient.fail/. 2019.

Christopher Domas. M/o/Vfuscator. 2015. URL: https://github.com/xoreaxeaxeax/movfuscator.

P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre Attacks: Exploiting Speculative Execution. In: S&P. 2019.

Stephen Dolan. mov is Turing-complete. 2013. URL: https://drwho.virtadpt.net/files/mov.pdf.

Acknowledgments

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No 681402). Funding was provided by generous gifts from Cloudflare, from Intel, and from ARM. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.