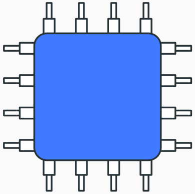
Turning Timing Differences into Data Leakage

Daniel Weber, Michael Schwarz

December 6, 2022

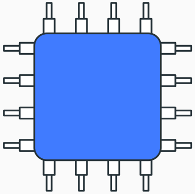
CISPA Helmholtz Center for Information Security

Agenda



CPU Caches

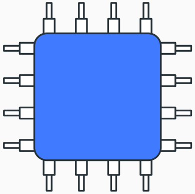
Agenda



CPU Caches

Stealthy Communication

Agenda



CPU Caches

Stealthy Communication

Leaking Inaccessible Data

Exercise Requirements

Hardware/Software Requirements:

- x86 CPU

Exercise Requirements

Hardware/Software Requirements:

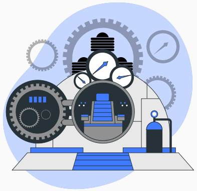
- x86 CPU
- Linux installation

Exercise Requirements

Hardware/Software Requirements:

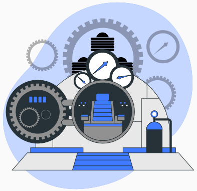
- x86 CPU
- Linux installation
- Installed tools: `python3`, `gcc`, `make`
- Installed Python package: `matplotlib`

CPU Optimizations



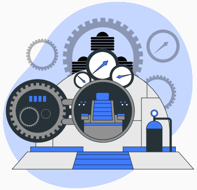
- Modern CPUs contain multiple **microarchitectural elements**

CPU Optimizations



- Modern CPUs contain multiple **microarchitectural elements**
- **Transparent** for the programmer

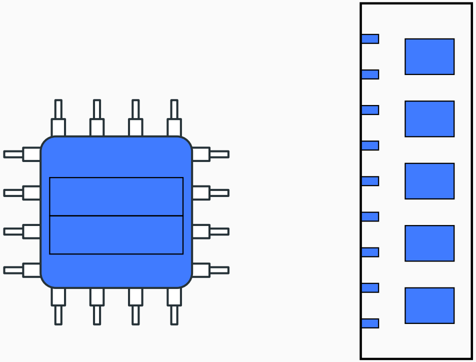
CPU Optimizations



- Modern CPUs contain multiple **microarchitectural elements**
- **Transparent** for the programmer
- **Optimize** for performance, power consumption, . . .

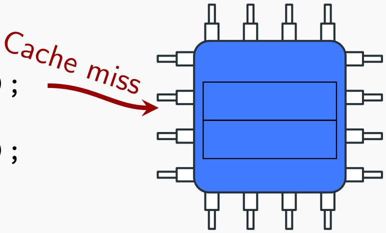
CPU Optimization: Cache

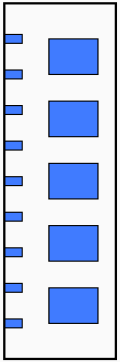
```
printf("%d", i);

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



CPU Optimization: Cache

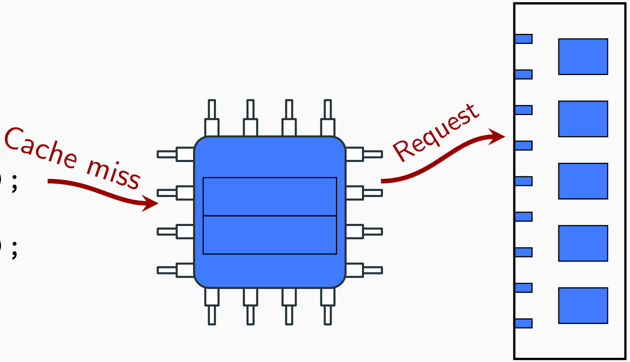
```
printf("%d", i);

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





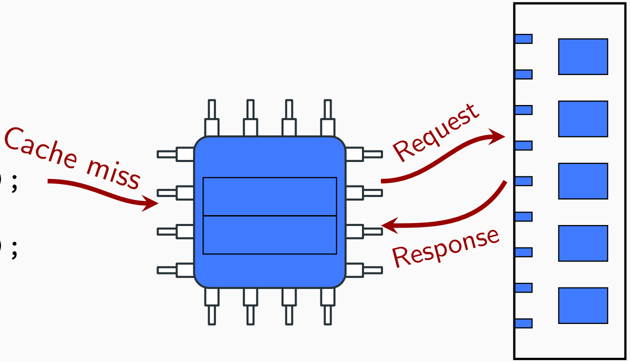
CPU Optimization: Cache

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

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



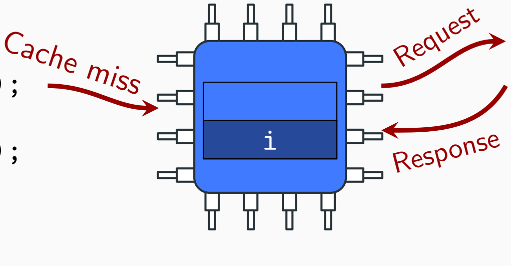
CPU Optimization: Cache

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

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



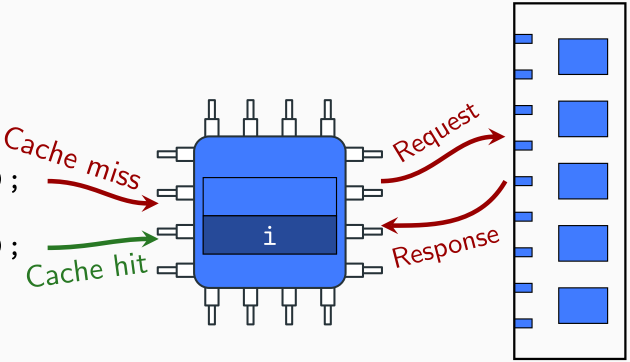
CPU Optimization: Cache

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

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

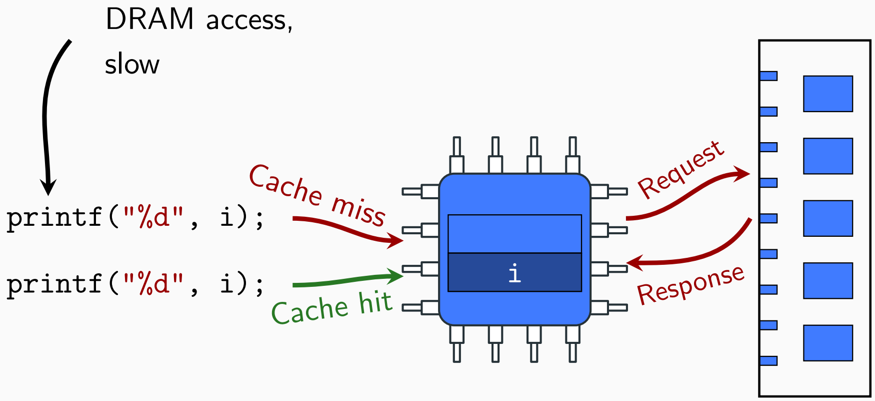


CPU Optimization: Cache

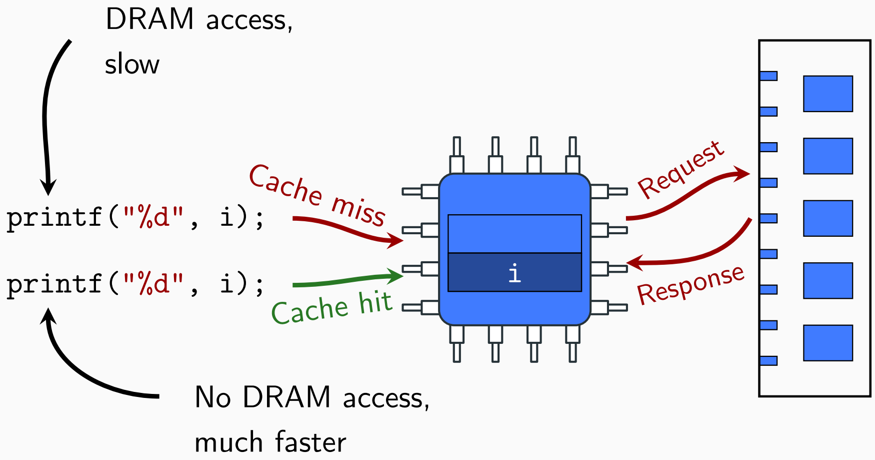
```
printf("%d", i);

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



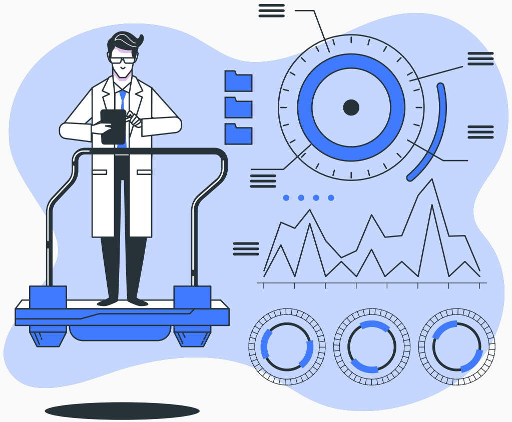
CPU Optimization: Cache



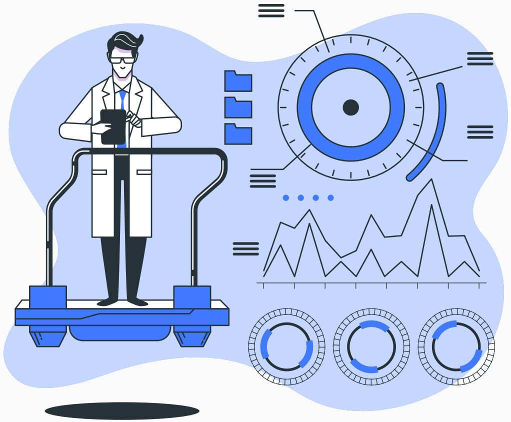
CPU Optimization: Cache



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

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



Does that really work?



Does that really work?

Can we **observe** these effects?

Experiment Idea

1) Time cache hits

Experiment Idea

1) Time cache hits
2) Time cache misses

Experiment Idea

1) Time cache hits
2) Time cache misses
3) Plot the timings

What Building Blocks do we need?

- **BB 1**: Bring memory **into** the cache

What Building Blocks do we need?

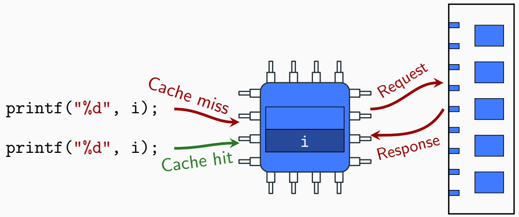
- **BB 1**: Bring memory **into** the cache
- **BB 2**: **Remove** memory from the cache

What Building Blocks do we need?

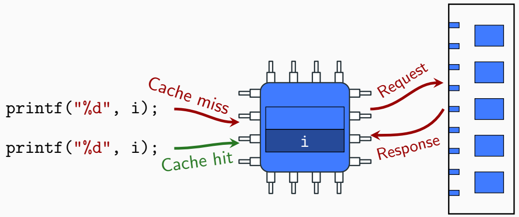
- **BB 1**: Bring memory **into** the cache
- **BB 2**: **Remove** memory from the cache
- **BB 3**: High-precision **time measurements**

How do we bring **memory into the cache**?

BB 1: Bring memory into the cache



BB 1: Bring memory into the cache



Easy! Just **access it**

How do we **remove memory** from the cache?

BB2: Remove Memory from the Cache

- Caches are limited in size

→ Access **many other addresses**

→ Original entry will be evicted

BB2: Remove Memory from the Cache

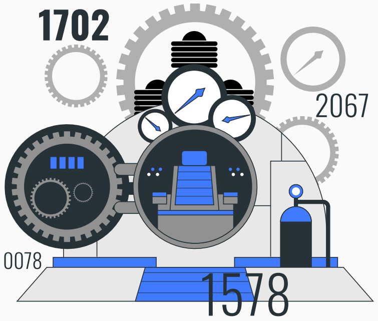
- Caches are limited in size

→ Access **many other addresses**

→ Original entry will be evicted

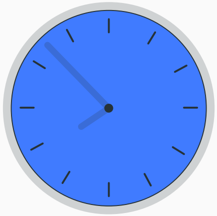
- Special **cache-maintenance instructions**

→ `CLFLUSH [rax]` and `CLFLUSHOPT [rax]`



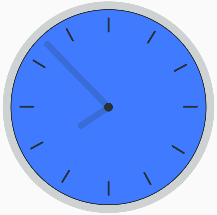
How do we get **high-precision time measurements**?

High-Precision Time Measurements



- x86 has two **instructions**: `rdtsc` and `rdtscp`

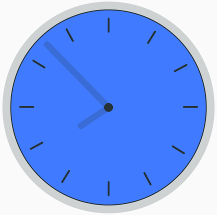
High-Precision Time Measurements



- x86 has two **instructions**: `rdtsc` and `rdtscp`
- Reads the processor's **time-stamp counter**

$\rightarrow$ CPU cycles since reset

High-Precision Time Measurements

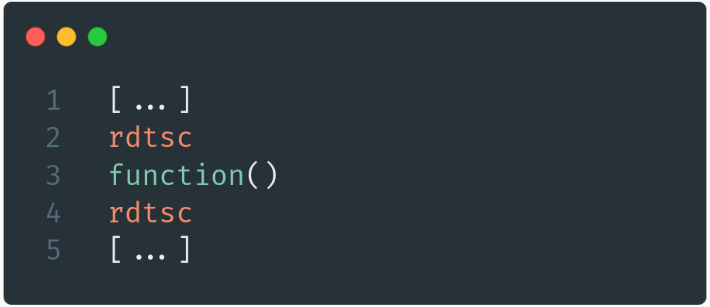


- x86 has two **instructions**: `rdtsc` and `rdtscp`
- Reads the processor's **time-stamp counter**

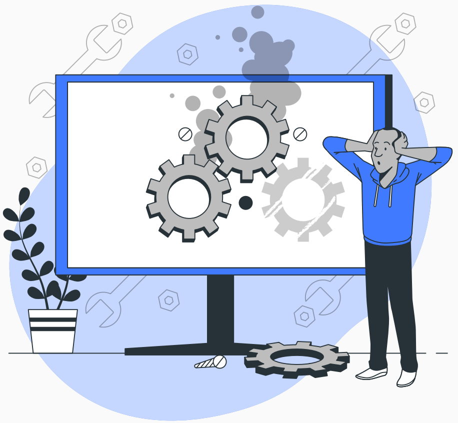
$\rightarrow$ CPU cycles since reset

- Highly accurate (**nanoseconds**), low overhead

High-Precision Time Measurements



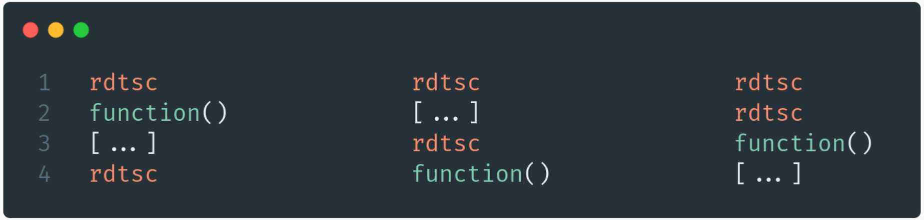
```
[ ... ]
rdtsc
function()
rdtsc
[ ... ]
```



What about out-of-order execution?

Timings can be Reordered

Out-of-order execution → different possibilities



```
rdtsc            rdtsc            rdtsc
function()       [ ... ]          rdtsc
[ ... ]          rdtsc            function()
rdtsc            function()       [ ... ]
```

Prevent Reordering

- **Pseudo-serializing** instruction `rdtscp` (recent CPUs)

Prevent Reordering

- **Pseudo-serializing** instruction `rdtscp` (recent CPUs)
- **Serializing** instructions like `cpuid`

Prevent Reordering

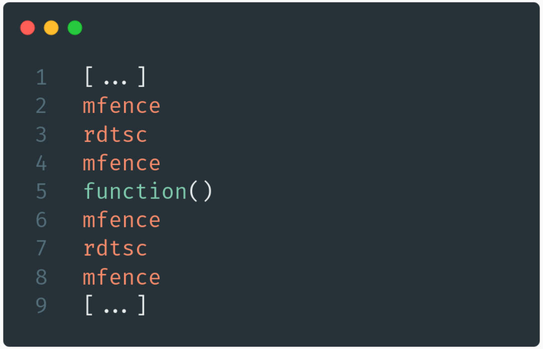
- **Pseudo-serializing** instruction `rdtscp` (recent CPUs)
- **Serializing** instructions like `cpuid`
- **Fences** like `mfence`

Prevent Reordering

- **Pseudo-serializing** instruction `rdtscp` (recent CPUs)
- **Serializing** instructions like `cpuid`
- **Fences** like `mfence`

Intel, *How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures White Paper*, December 2010.

BB3: High-Precision Time Measurements (Accurate)



```
[ … ]
mfence
rdtsc
mfence
function()
mfence
rdtsc
mfence
[ … ]
```

We got **all building blocks!**

Let’s get our hands dirty!

Exercise 1: Observing CPU Caches

The Task:

Build a histogram for cache hits and misses.

Exercise 1: Observing CPU Caches

The Task:

Build a histogram for cache hits and misses.

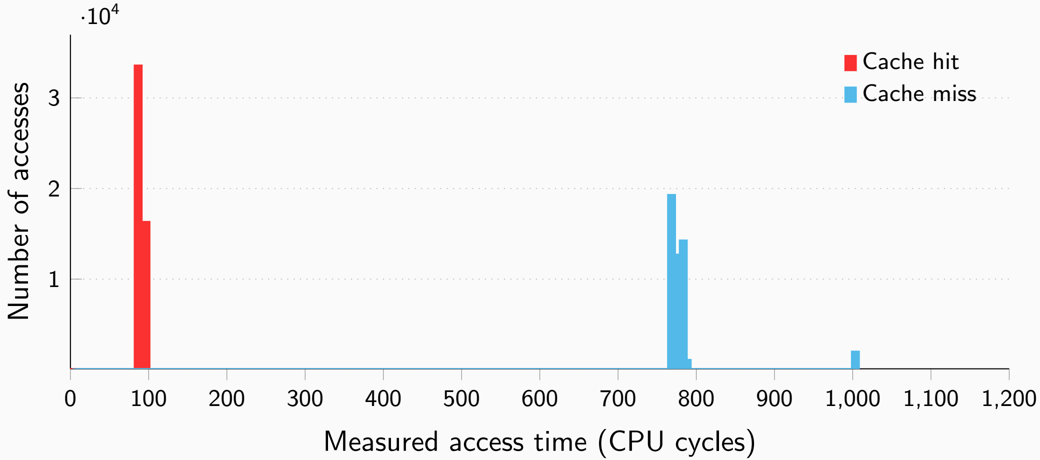
Hints for better results:

- Connect your laptop to power
- Close unrelated programs

Exercise 1:
Observing CPU Caches

(`https:///challenge.attacking.systems/cpu-caches.tar.gz`)

Mission Accomplished: Observing CPU Caches



Can we do something with that?

Sender

...

D (0x44)

E (0x45)

F (0x46)

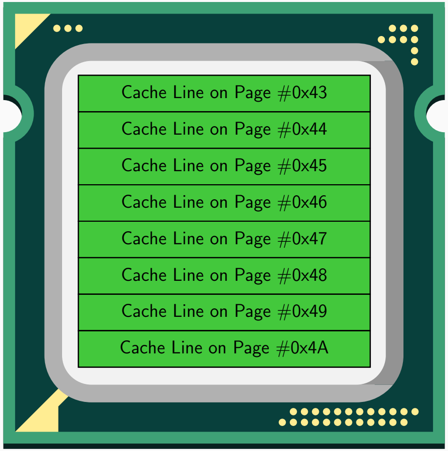
G (0x47)

H (0x48)

I (0x49)

...

Last-level cache



Receiver

Sender

...

D (0x44)

E (0x45)

F (0x46)

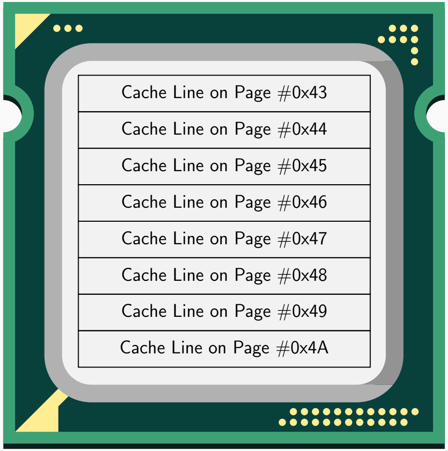
G (0x47)

H (0x48)

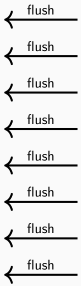
I (0x49)

...

Last-level cache



Receiver



Sender

. . .

D (0x44)

E (0x45)

F (0x46)

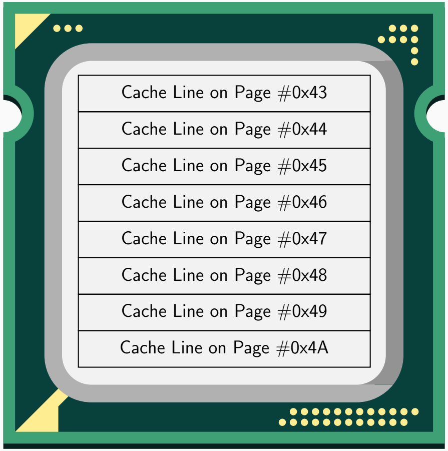
G (0x47)

H (0x48)

I (0x49)

. . .

Last-level cache



Receiver

Sender

Last-level cache

Receiver

...

D (0x44)

E (0x45)

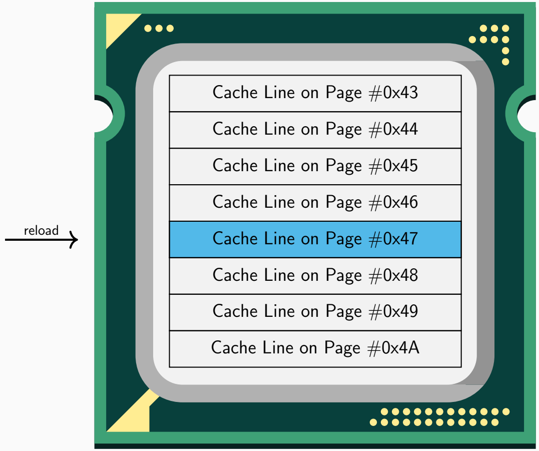
F (0x46)

G (0x47)

H (0x48)

I (0x49)

...



Sender

Last-level cache

Receiver

. . .

D (0x44)

E (0x45)

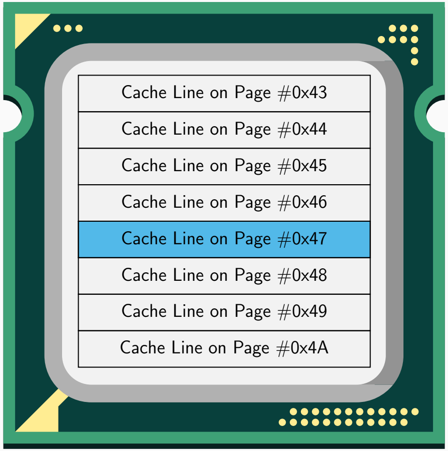
F (0x46)

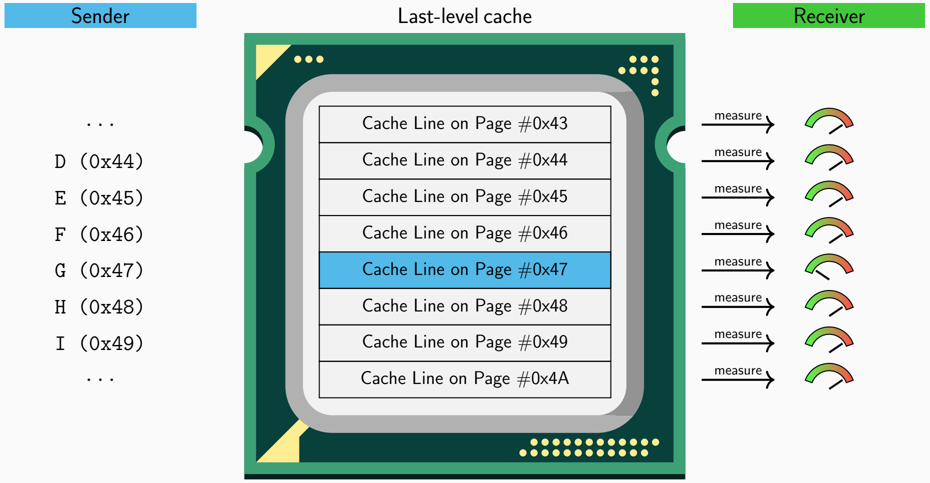
G (0x47)

H (0x48)

I (0x49)

. . .




Sender
...
D (0x44)
E (0x45)
F (0x46)
G (0x47)
H (0x48)
I (0x49)
...
Last-level cache
Cache Line on Page #0x43
Cache Line on Page #0x44
Cache Line on Page #0x45
Cache Line on Page #0x46
Cache Line on Page #0x47
Cache Line on Page #0x48
Cache Line on Page #0x49
Cache Line on Page #0x4A
Receiver
measure
measure
measure
measure
measure
measure
measure
measure

Sender

...

D (0x44)

E (0x45)

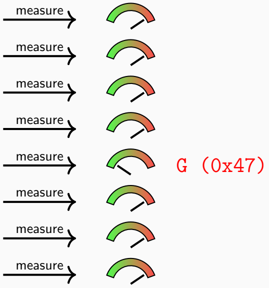
F (0x46)

G (0x47)

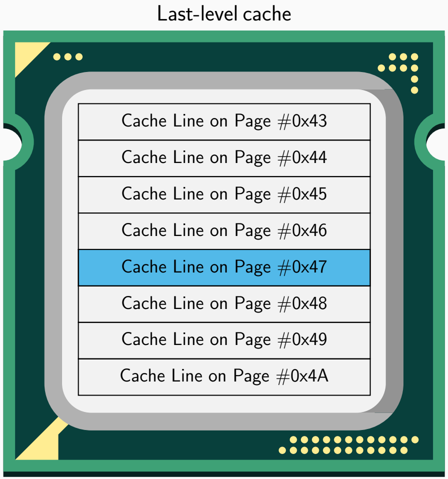
H (0x48)

I (0x49)

...



Receiver



Sender

...

D (0x44)

E (0x45)

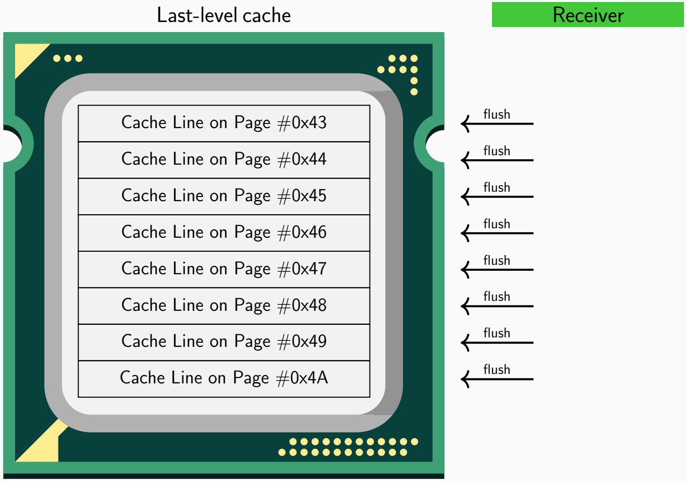
F (0x46)

G (0x47)

H (0x48)

I (0x49)

...



Sender

...

D (0x44)

E (0x45)

F (0x46)

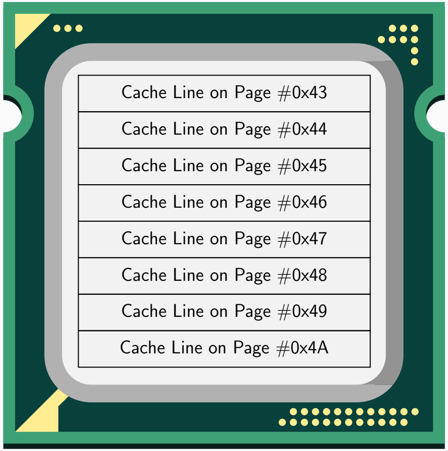
G (0x47)

H (0x48)

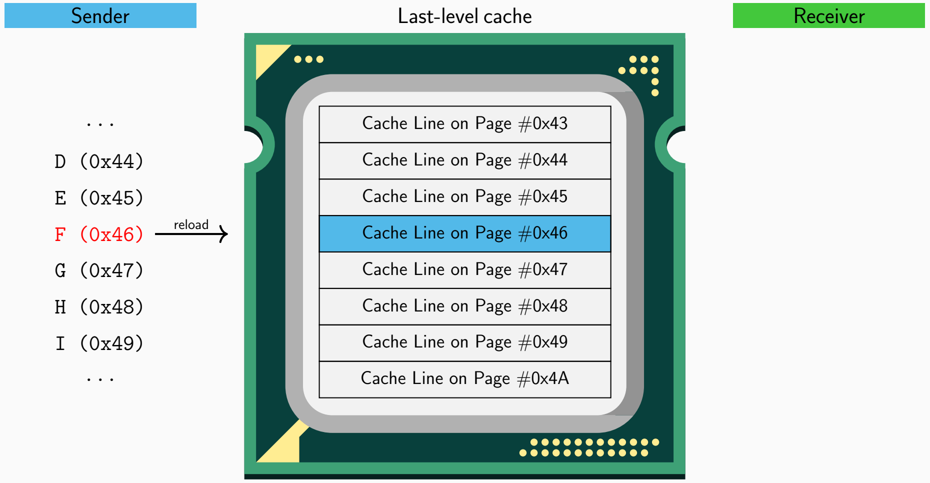
I (0x49)

...

Last-level cache



Receiver


Sender
Last-level cache
Receiver
...
D (0x44)
E (0x45)
F (0x46)
G (0x47)
H (0x48)
I (0x49)
...
reload
Cache Line on Page #0x43
Cache Line on Page #0x44
Cache Line on Page #0x45
Cache Line on Page #0x46
Cache Line on Page #0x47
Cache Line on Page #0x48
Cache Line on Page #0x49
Cache Line on Page #0x4A

Sender

Last-level cache

Receiver

. . .

D (0x44)

E (0x45)

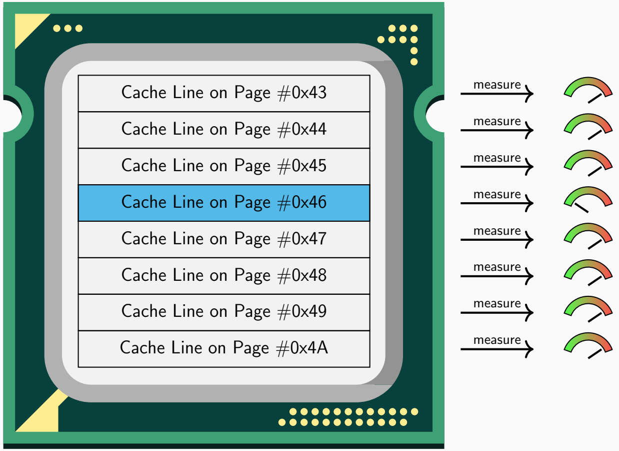
F (0x46)

G (0x47)

H (0x48)

I (0x49)

. . .



F (0x46)

CPU Optimization: Hardware Prefetcher

- CPUs optimize **recognizable access** patterns

CPU Optimization: Hardware Prefetcher

- CPUs optimize **recognizable access** patterns
- Pattern detected → **prefetch next** addresses

CPU Optimization: Hardware Prefetcher

- CPUs optimize **recognizable access** patterns
- Pattern detected → **prefetch next** addresses

```
for (size_t i = 0; i < 10; i++) {
    // CPU will prefetch arr[i+1] -> Cache hits
    sum += arr[i];
}
```

CPU Optimization: Hardware Prefetcher

- CPUs optimize **recognizable access** patterns
- Pattern detected → **prefetch next** addresses

```
for (size_t i = 0; i < 10; i++) {
    // CPU will prefetch arr[i+1] -> Cache hits
    sum += arr[i];
}
```

→ **Permutate** your accesses!

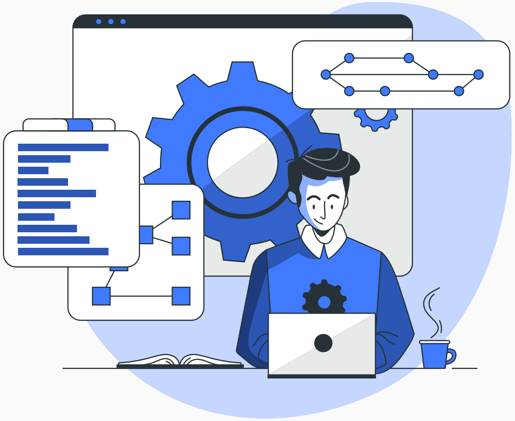
CPU Optimization: Hardware Prefetcher

- CPUs optimize **recognizable access** patterns
- Pattern detected → **prefetch next** addresses

```
for (size_t i = 0; i < 10; i++) {
    // CPU will prefetch arr[i+1] -> Cache hits
    sum += arr[i];
}
```

→ **Permutate** your accesses!

→ Shift indices by **4096B**



Let's try this out!

Exercise 2: Covert Communication

The Task:

Build a covert communication channel using the CPU cache.

Exercise 2: Covert Communication

The Task:

Build a covert communication channel using the CPU cache.

Hints for better results:

- Connect your laptop to power
- Close unrelated programs

Exercise 2: Covert Communication

The Task:

Build a covert communication channel using the CPU cache.

Hints for better results:

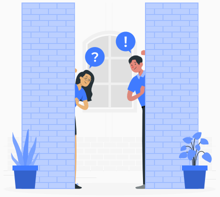
- Connect your laptop to power
- Close unrelated programs
- Use `permutate_index` function to prevent prefetch effects

Exercise 2:

Stealthy Communication

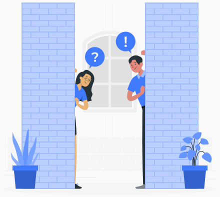
(`https:///challenge.attacking.systems/covert.tar.gz`)

Mission Accomplished: Covert Communication via the CPU Cache



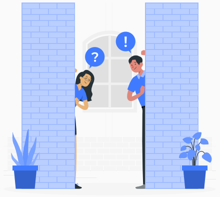
- Encoding data in the CPU cache

Mission Accomplished: Covert Communication via the CPU Cache



- Encoding data in the CPU cache
- Decoding from another process

Mission Accomplished: Covert Communication via the CPU Cache

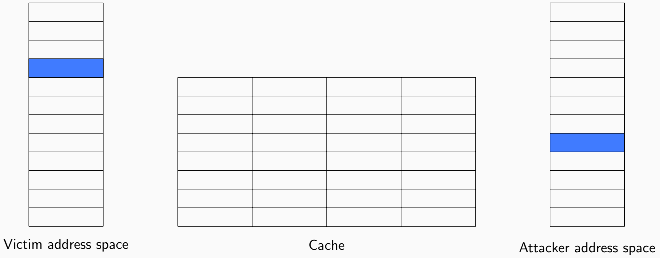


- Encoding data in the CPU cache
- Decoding from another process

→ **Stealthy Communication** between 2 processes

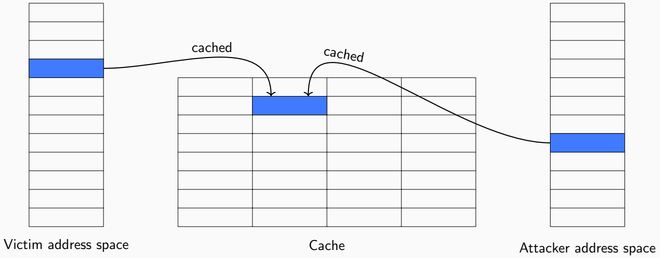
What happens if the sending party is a benign process?

Flush+Reload



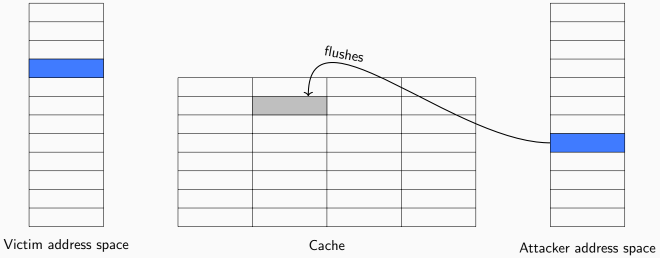
Step 1: Attacker maps shared library (shared memory, in cache)

Flush+Reload



Step 1: Attacker maps shared library (shared memory, in cache)

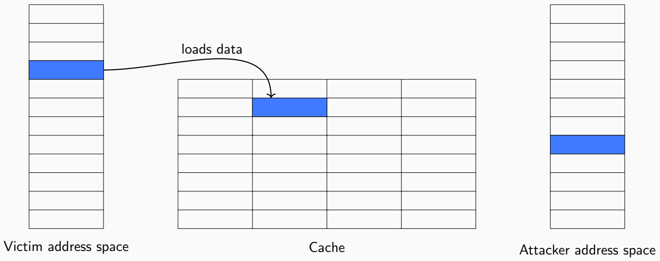
Flush+Reload



Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

Flush+Reload

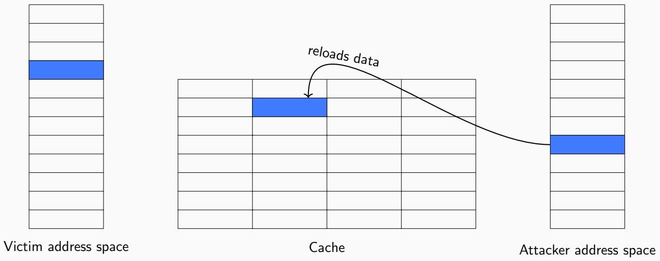


Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

Step 3: Victim loads the data

Flush+Reload



Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

Step 3: Victim loads the data

Step 4: Attacker measures the access time to reload the data

Cache Attacks

- Build **covert communication channels**

Cache Attacks

- Build **covert communication channels**
- **Monitor** function calls of other applications

Cache Attacks

- Build **covert communication channels**
- **Monitor** function calls of other applications
- Leak **cryptographic keys**

Cache Attacks

- Build **covert communication channels**
- **Monitor** function calls of other applications
- Leak **cryptographic keys**
- Leak information from **co-located virtual machines**
- ...

Can we leak something else than meta data?

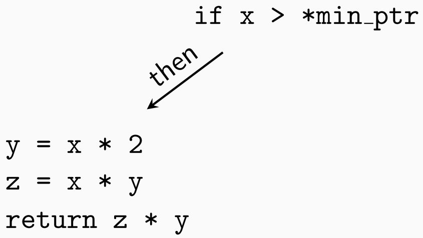
Can we leak something else than meta data?

Perhaps **real data?**

CPU Optimization: Transient Execution

```
if x > *min_ptr
```

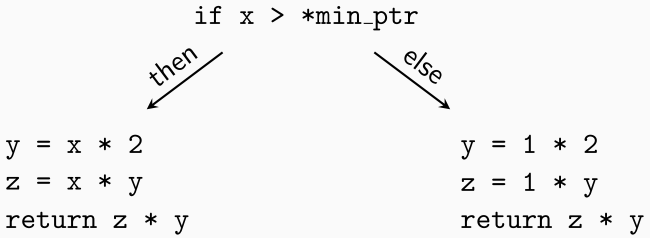
CPU Optimization: Transient Execution



```
if x > *min_ptr
```

```
y = x * 2
z = x * y
return z * y
```

CPU Optimization: Transient Execution

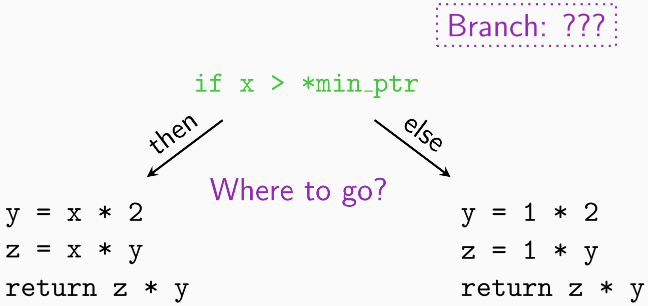


```
if x > *min_ptr
```

```
y = x * 2
z = x * y
return z * y
```

```
y = 1 * 2
z = 1 * y
return z * y
```

CPU Optimization: Transient Execution

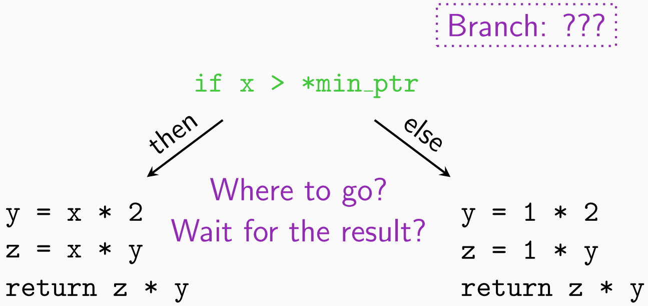


```
if x > *min_ptr
```

```
y = x * 2
z = x * y
return z * y
```

```
y = 1 * 2
z = 1 * y
return z * y
```

CPU Optimization: Transient Execution

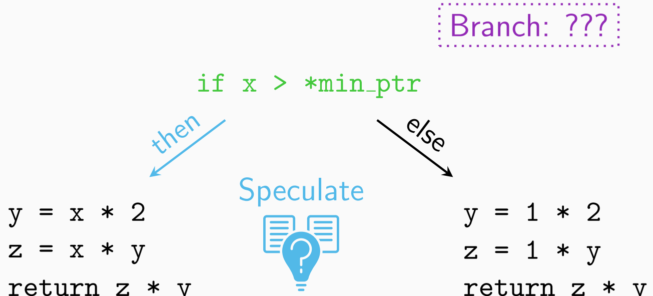


```
if x > *min_ptr
```

```
y = x * 2
z = x * y
return z * y
```

```
y = 1 * 2
z = 1 * y
return z * y
```

CPU Optimization: Transient Execution

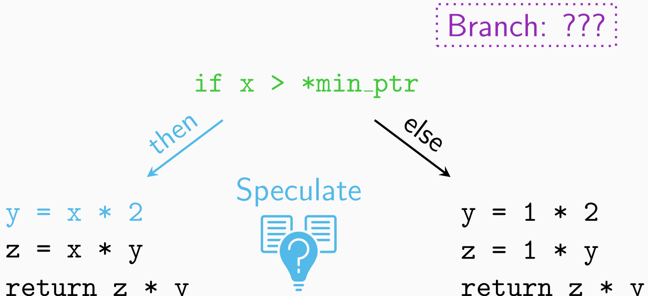


```
if x > *min_ptr
```

```
y = x * 2
z = x * y
return z * y
```

```
y = 1 * 2
z = 1 * y
return z * y
```

CPU Optimization: Transient Execution

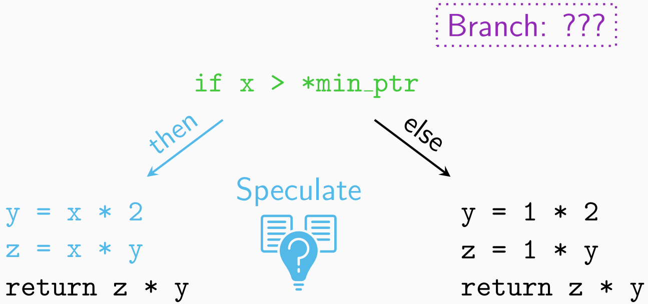


```
if x > *min_ptr
```

```
y = x * 2
z = x * y
return z * y
```

```
y = 1 * 2
z = 1 * y
return z * y
```

CPU Optimization: Transient Execution

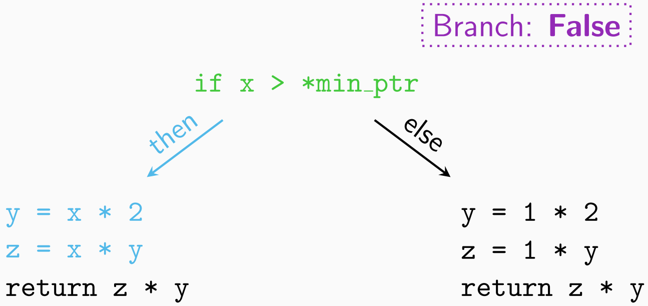


```
if x > *min_ptr
```

```
y = x * 2
z = x * y
return z * y
```

```
y = 1 * 2
z = 1 * y
return z * y
```

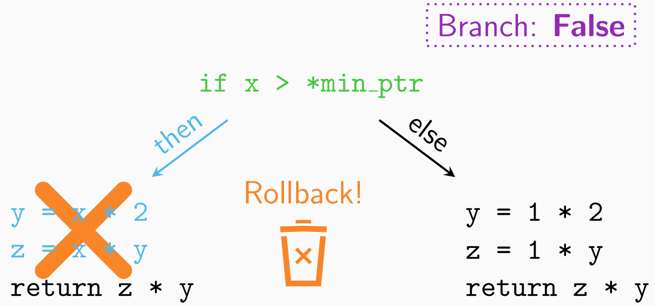
CPU Optimization: Transient Execution



```
y = x * 2
z = x * y
return z * y
```

```
y = 1 * 2
z = 1 * y
return z * y
```

CPU Optimization: Transient Execution



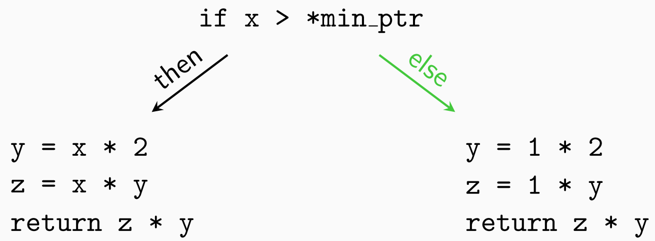
```
if x > *min_ptr
```

```
y = x * 2
z = x * y
return z * y
```

```
y = 1 * 2
z = 1 * y
return z * y
```

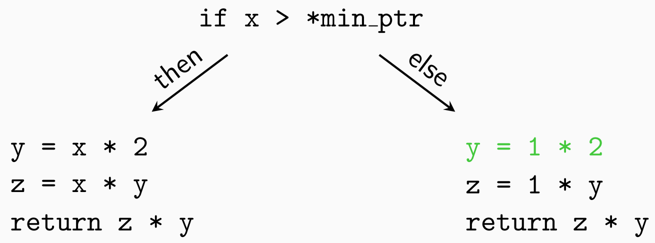
CPU Optimization: Transient Execution

Branch: **False**



CPU Optimization: Transient Execution

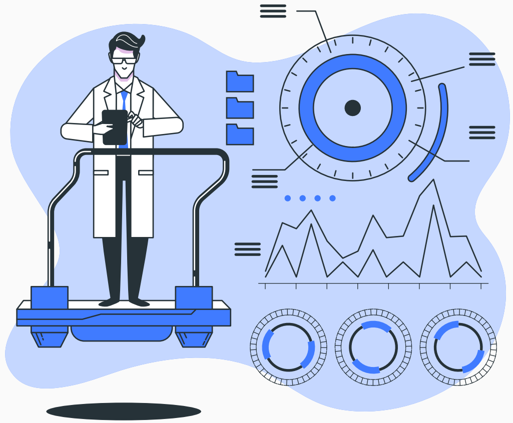
Branch: **False**



```
if x > *min_ptr
```

```
y = x * 2
z = x * y
return z * y
```

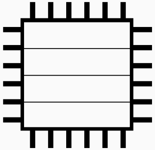
```
y = 1 * 2
z = 1 * y
return z * y
```



Maybe transient execution leaves **traces in the microarchitecture**?

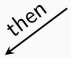
Transient Data Loads

```
x = 42
x_ptr = &x
flush(x_ptr)
if x > *min_ptr
```

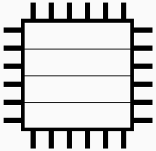


Transient Data Loads

```
x = 42
x_ptr = &x
flush(x_ptr)
if x > *min_ptr
```



```
y = *x_ptr
z = x * y
return z * y
```



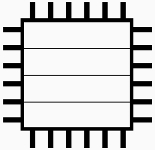
Transient Data Loads

```
x = 42
x_ptr = &x
flush(x_ptr)
if x > *min_ptr
```

$$\begin{tikzcd} \text{if} \arrow[d, "\text{then}"'] \arrow[dr, "\text{else}"] & \\ \text{then-branch} & \text{else-branch} \end{tikzcd}$$

```
y = *x_ptr
z = x * y
return z * y
```

```
y = 1 * 2
z = 1 * y
return z * y
```



Transient Data Loads

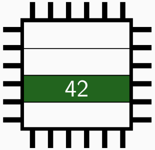
```
x = 42
x_ptr = &x
flush(x_ptr)
if x > *min_ptr
```

then

```
y = *x_ptr
z = x * y
return z * y
```

else

```
y = 1 * 2
z = 1 * y
return z * y
```



Transient Data Loads

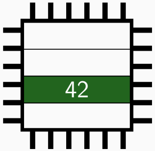
```
x = 42
x_ptr = &x
flush(x_ptr)
if x > *min_ptr
```

then

```
y = *x_ptr
z = x * y
return z * y
```

else

```
y = 1 * 2
z = 1 * y
return z * y
```



Transient Data Loads

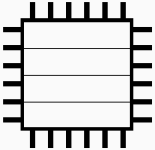
```
x = 42
x_ptr = &x
flush(x_ptr)
if x > *min_ptr
```

then

```
y = *x_ptr
z = x * y
return z * y
```

else

```
y = 1 * 2
z = 1 * y
return z * y
```



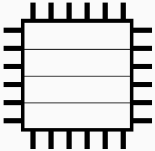
Transient Data Loads

```
x = 42
x_ptr = &x
flush(x_ptr)
if x > *min_ptr
```

Branch: ???

```
y = *x_ptr
z = x * y
return z * y
```

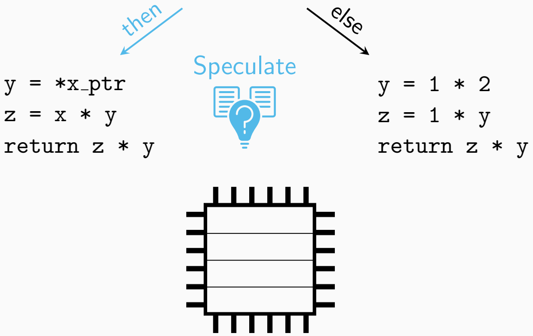
```
y = 1 * 2
z = 1 * y
return z * y
```



Transient Data Loads

```
x = 42
x_ptr = &x
flush(x_ptr)
if x > *min_ptr
```

Branch: ???



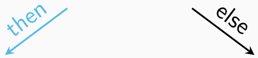
```
y = *x_ptr
z = x * y
return z * y
```

```
y = 1 * 2
z = 1 * y
return z * y
```

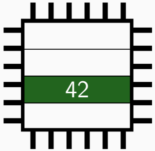
Transient Data Loads

```
x = 42
x_ptr = &x
flush(x_ptr)
if x > *min_ptr
```

Branch: ???



```
y = *x_ptr          y = 1 * 2
z = x * y           z = 1 * y
return z * y        return z * y
```



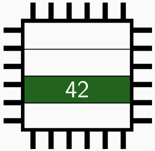
Transient Data Loads

```
x = 42
x_ptr = &x
flush(x_ptr)
if x > *min_ptr
```

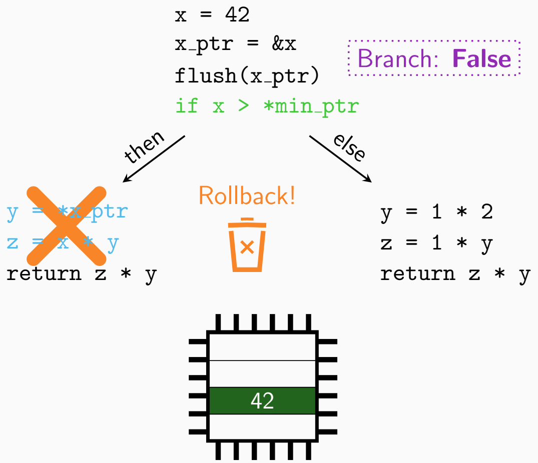
Branch: **False**

```
y = *x_ptr
z = x * y
return z * y
```

```
y = 1 * 2
z = 1 * y
return z * y
```



Transient Data Loads



```
x = 42
x_ptr = &x
flush(x_ptr)
if x > *min_ptr
```

~~y = *x_ptr~~
~~z = x * y~~

```
return z * y
```

```
y = 1 * 2
z = 1 * y
return z * y
```

Mic-Sec 22: Turning Timing Differences into Data Leakage

Transient Data Loads

```
x = 42
x_ptr = &x
flush(x_ptr)
if x > *min_ptr
```

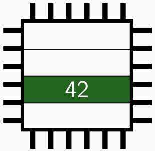
Branch: **False**

then

```
y = *x_ptr
z = x * y
return z * y
```

else

```
y = 1 * 2
z = 1 * y
return z * y
```



Transient Data Loads

```
x = 42
x_ptr = &x
flush(x_ptr)
if x > *min_ptr
```

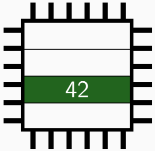
Branch: **False**

then

```
y = *x_ptr
z = x * y
return z * y
```

else

```
y = 1 * 2
z = 1 * y
return z * y
```



Transient Data Loads

```
x = 42
x_ptr = &x
flush(x_ptr)
if x > *min_ptr
```

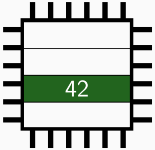
Branch: **False**

then

```
y = *x_ptr
z = x * y
return z * y
```

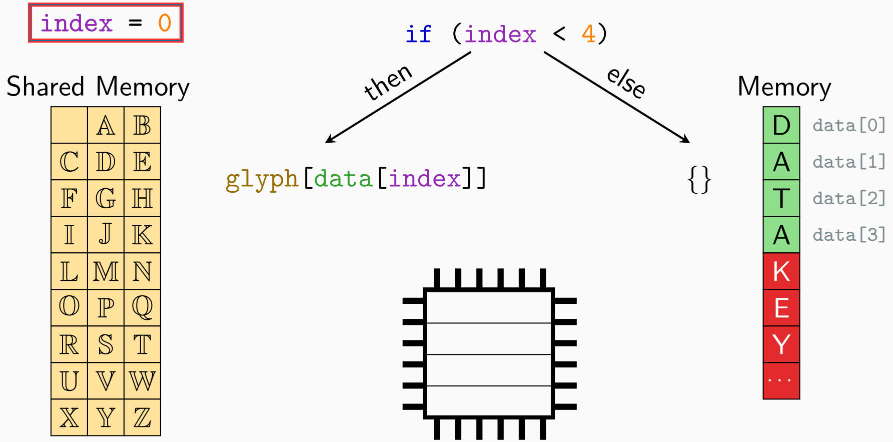
else

```
y = 1 * 2
z = 1 * y
return z * y
```



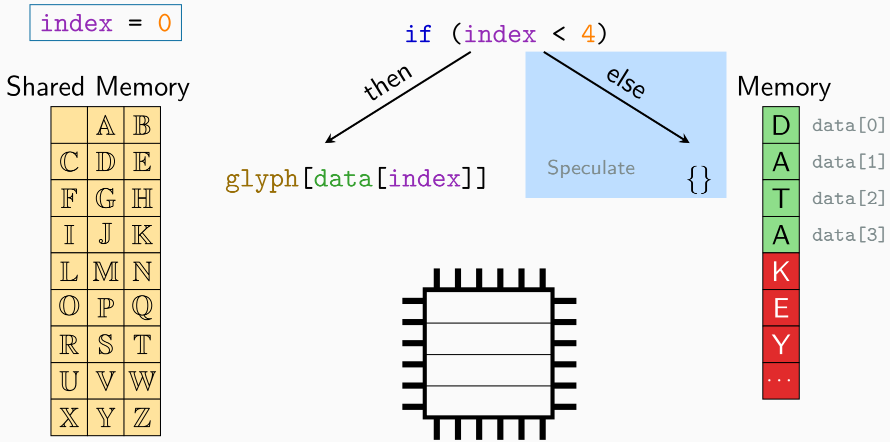
Transient execution **does leave traces** in the microarchitecture!

Spectre



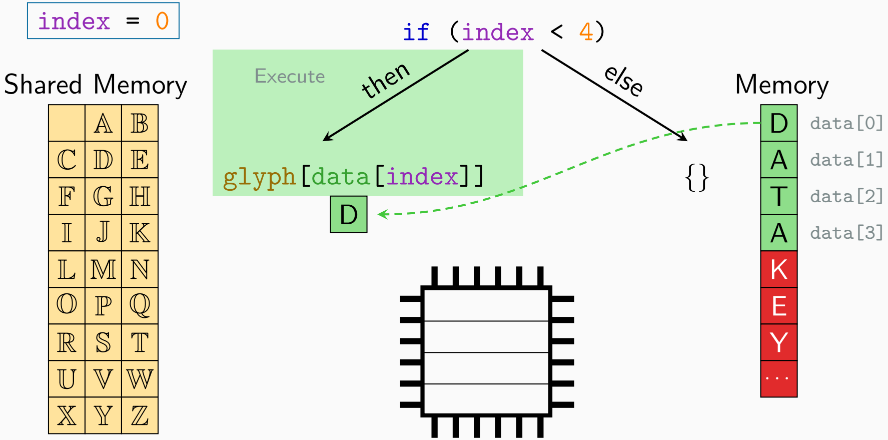
index = 0
Shared Memory
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
if (index < 4)
then
else
glyph[data[index]]
{}
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]

Spectre



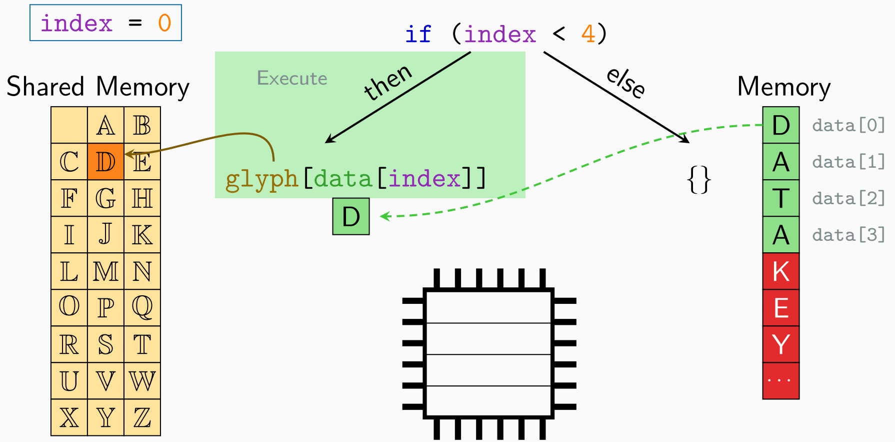
index = 0
Shared Memory
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
if (index < 4)
then
else
glyph[data[index]]
Speculate
{}
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]

Spectre



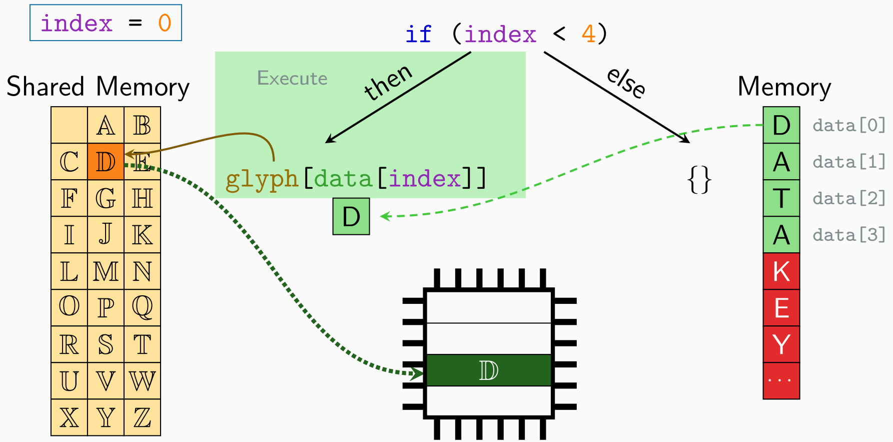
index = 0
Shared Memory
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
if (index < 4)
Execute
then
else
glyph[data[index]]
D
{}
Memory
D A T A K E Y ...
data[0]
data[1]
data[2]
data[3]

Spectre



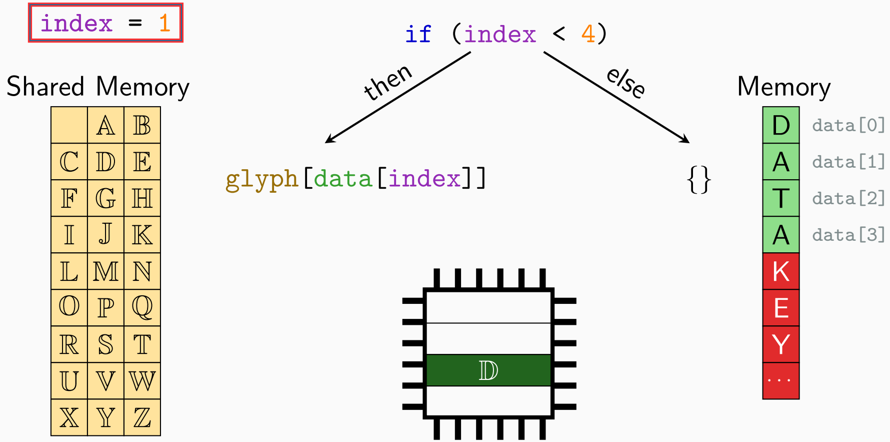
index = 0
if (index < 4)
then
else
Execute
glyph[data[index]]
D
{}
Shared Memory
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]

Spectre



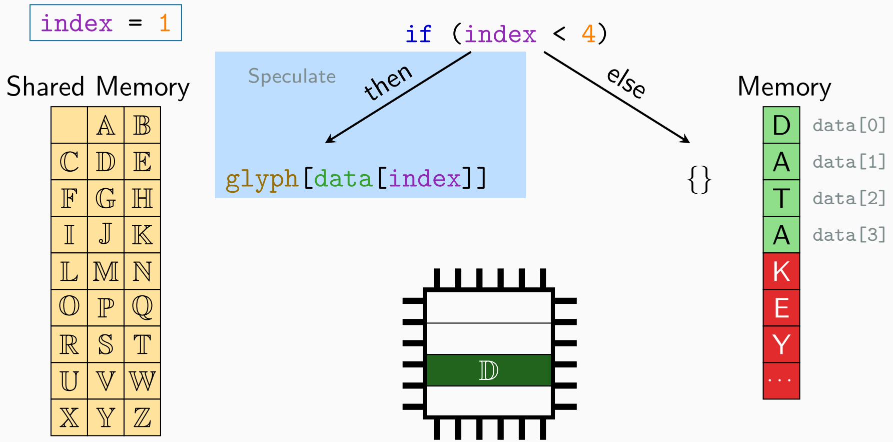
index = 0
if (index < 4)
then
else
Execute
glyph[data[index]]
D
{}
Shared Memory
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
D
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]

Spectre

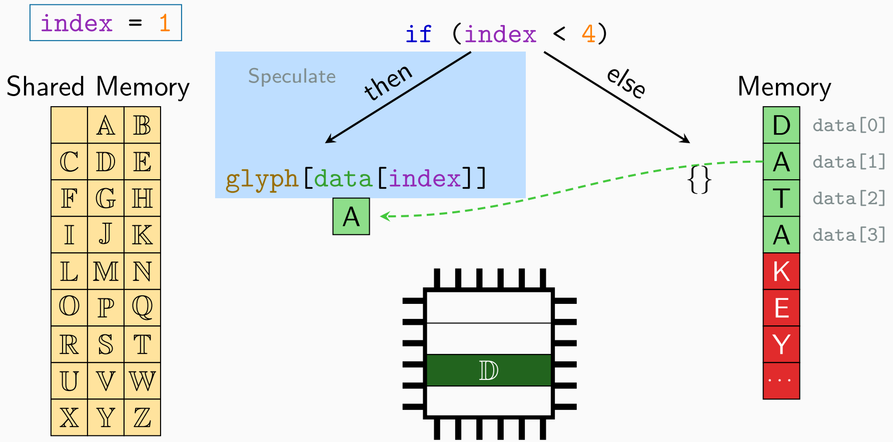


index = 1
Shared Memory
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
if (index < 4)
then
else
glyph[data[index]]
{}
D
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]

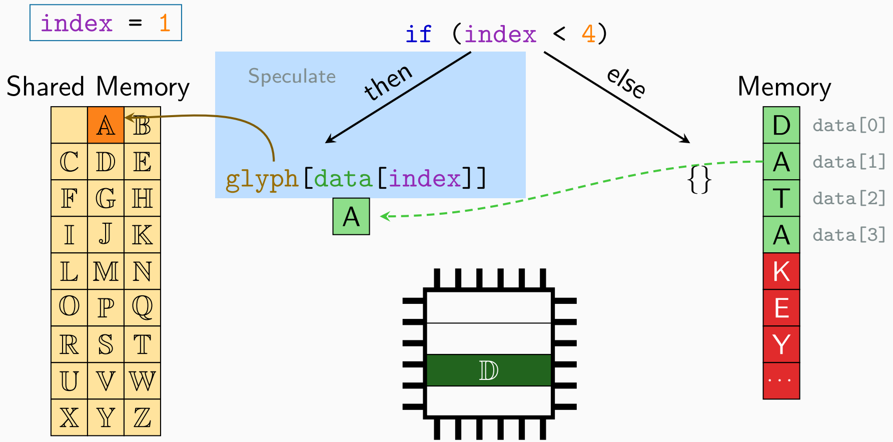
Spectre


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

Spectre

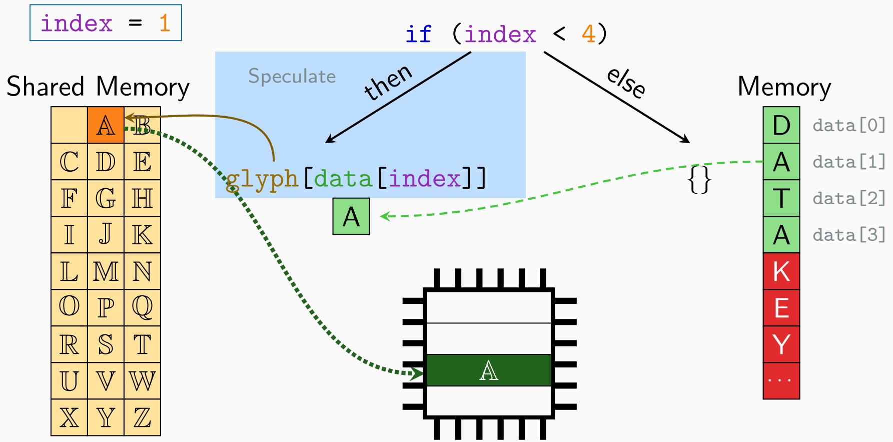

index = 1
Shared Memory
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
Speculate
if (index < 4)
then
else
glyph[data[index]]
A
{}
D
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]

Spectre



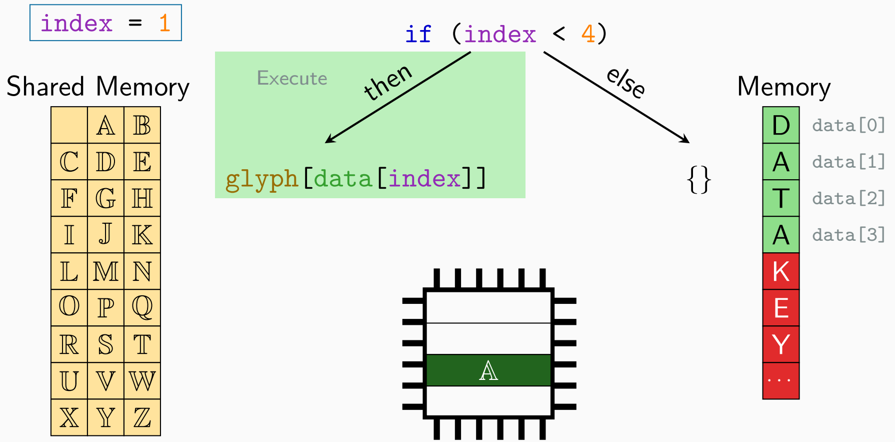
index = 1
if (index < 4)
Speculate
then
else
Shared Memory
A B
C D E
F G H
I J K
L M N
O P Q
R S T
U V W
X Y Z
glyph[data[index]]
A
{}
Memory
D data[0]
A data[1]
T data[2]
A data[3]
K
E
Y
...
D

Spectre

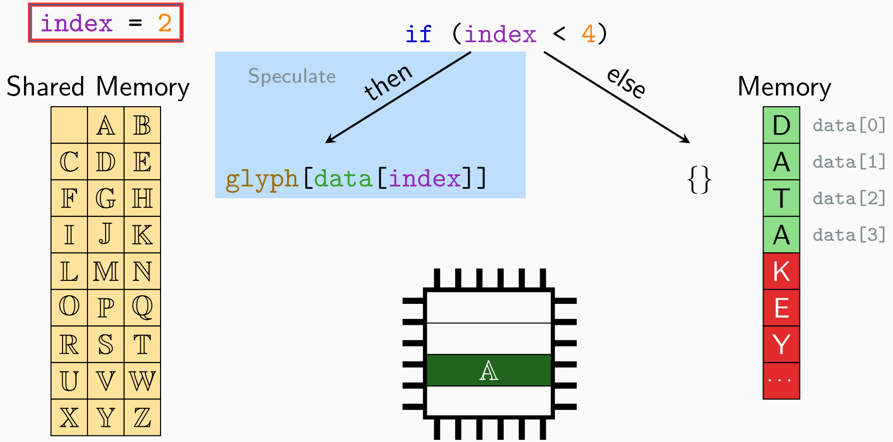


index = 1
if (index < 4)
Speculate
then
else
glyph[data[index]]
A
{}
Shared Memory
A B
C D E
F G H
I J K
L M N
O P Q
R S T
U V W
X Y Z
Memory
D data[0]
A data[1]
T data[2]
A data[3]
K
E
Y
...
A

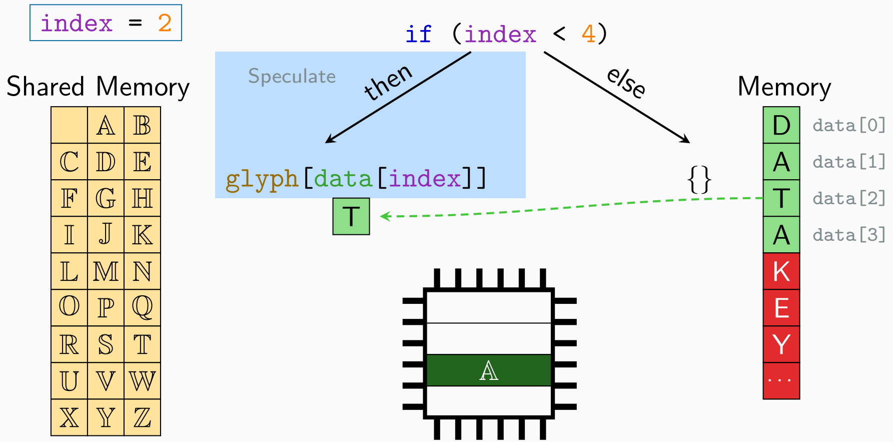
Spectre


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

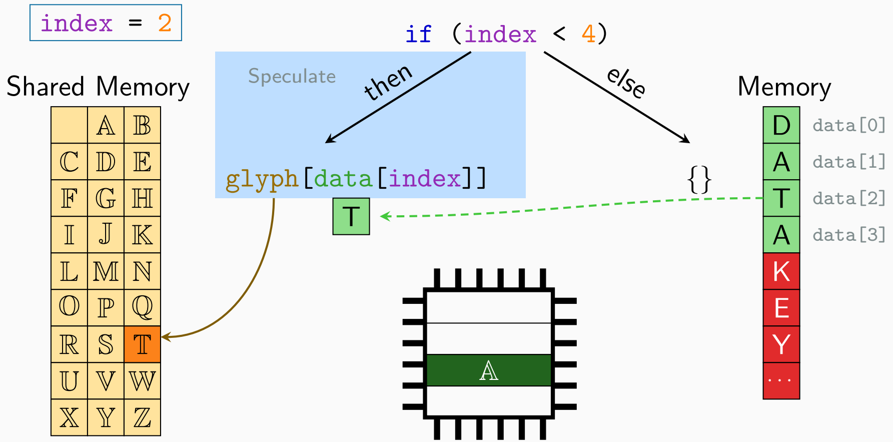
Spectre


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

Spectre

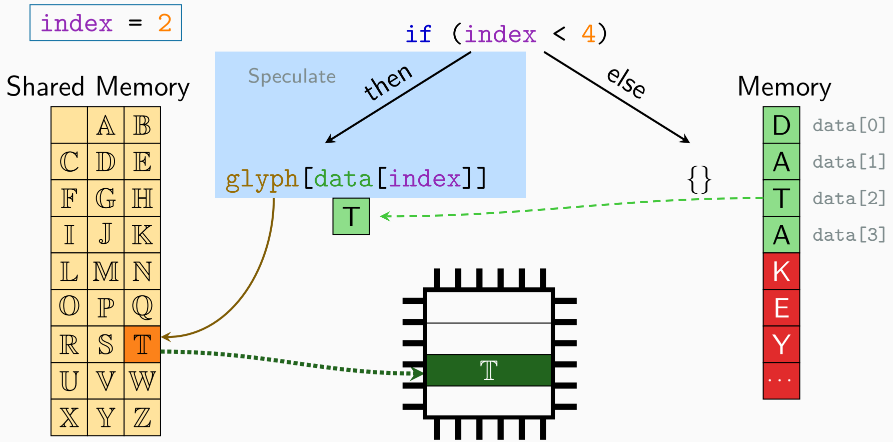

index = 2
if (index < 4)
Speculate
then
else
glyph[data[index]]
{}
T
Shared Memory
A B
C D E
F G H
I J K
L M N
O P Q
R S T
U V W
X Y Z
A
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]

Spectre



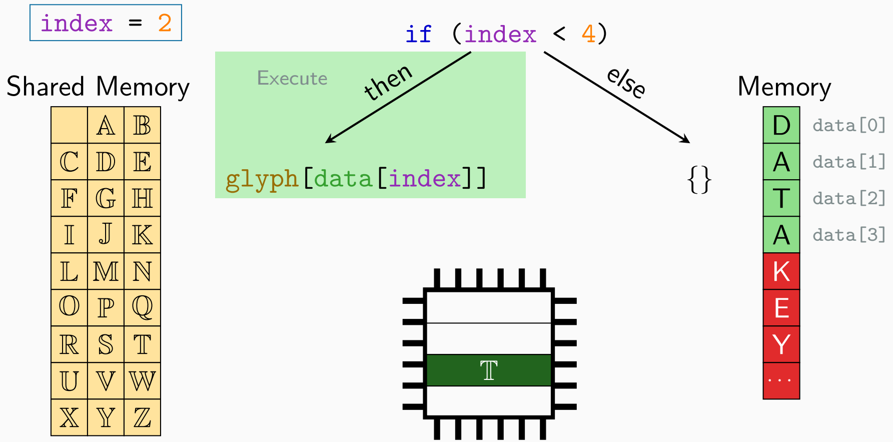
index = 2
if (index < 4)
Speculate
then
else
glyph[data[index]]
{}
T
Shared Memory
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
A
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]

Spectre

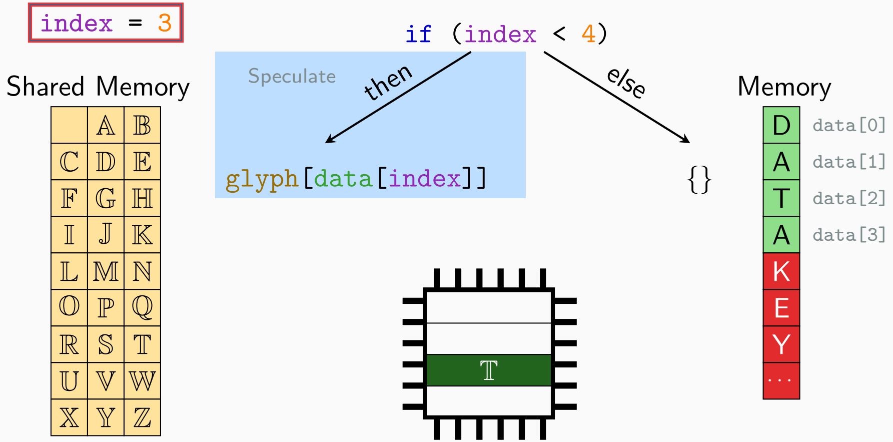


index = 2
if (index < 4)
Speculate
then
else
glyph[data[index]]
{}
T
Shared Memory
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
T
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]

Spectre

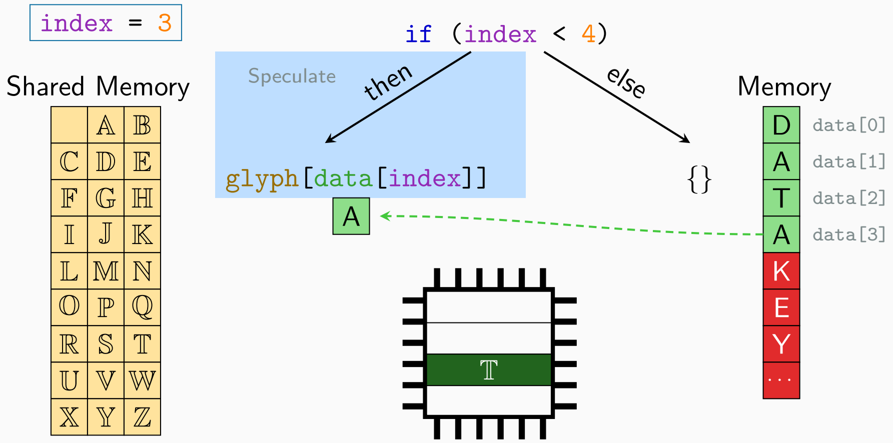

index = 2
Shared Memory
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
if (index < 4)
then
else
Execute
glyph[data[index]]
{}
T
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]

Spectre

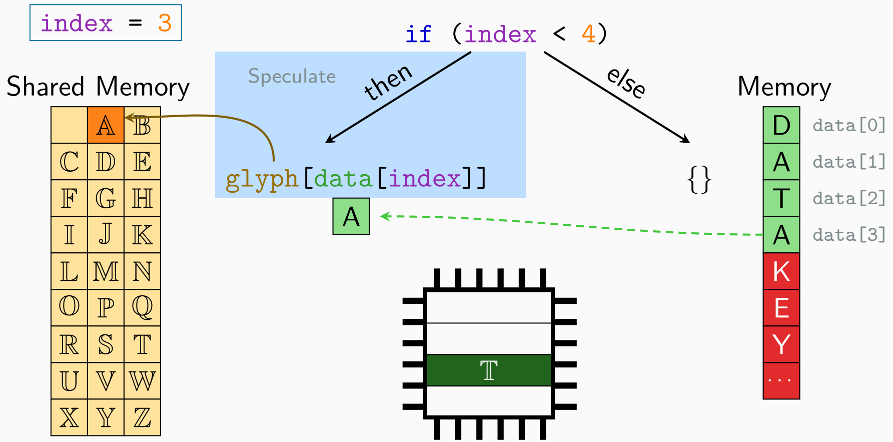


index = 3
Shared Memory
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
if (index < 4)
Speculate
then
else
glyph[data[index]]
{}
T
Memory
D
A
T
A
K
E
Y
...
data[0]
data[1]
data[2]
data[3]

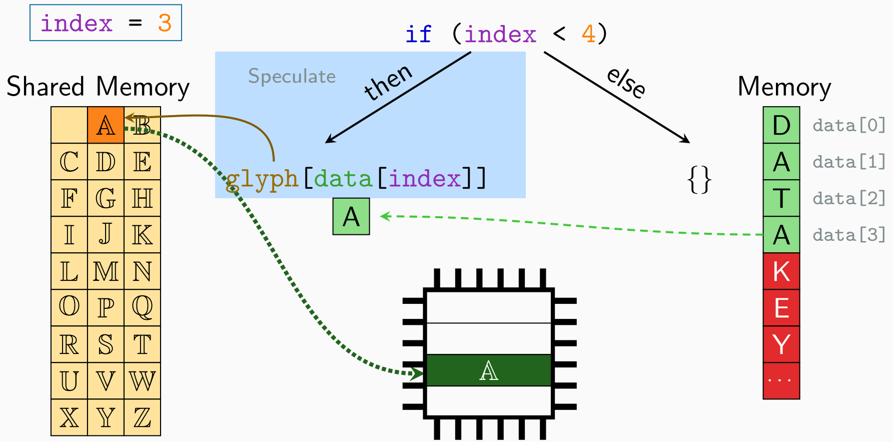
Spectre


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

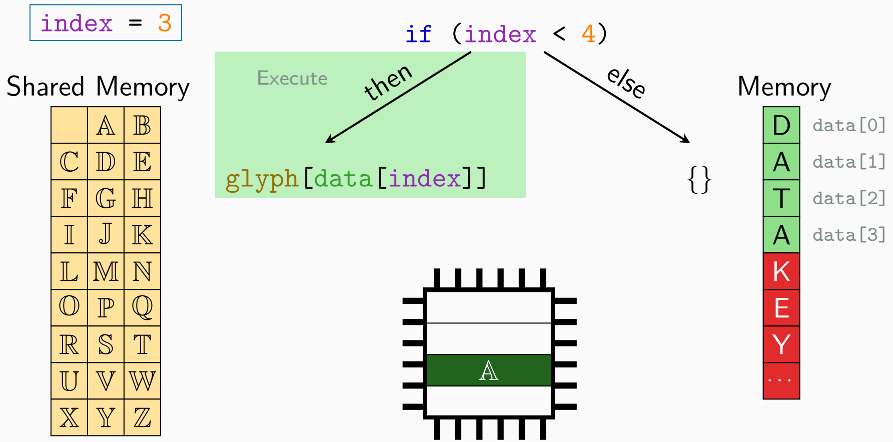
Spectre


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

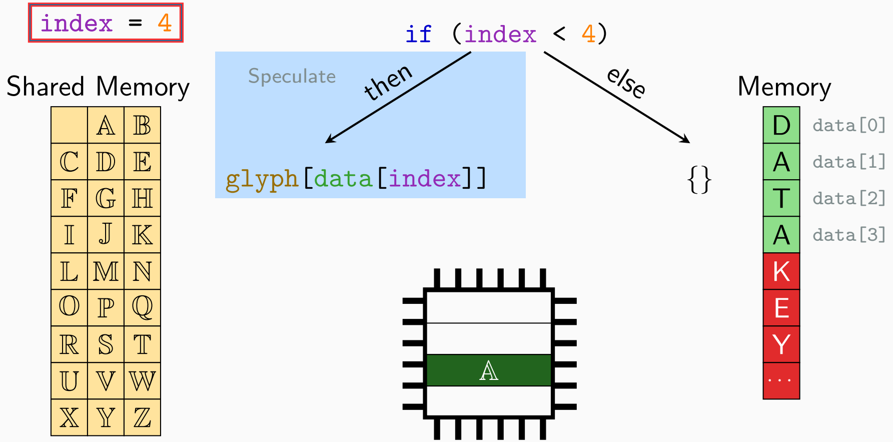
Spectre


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

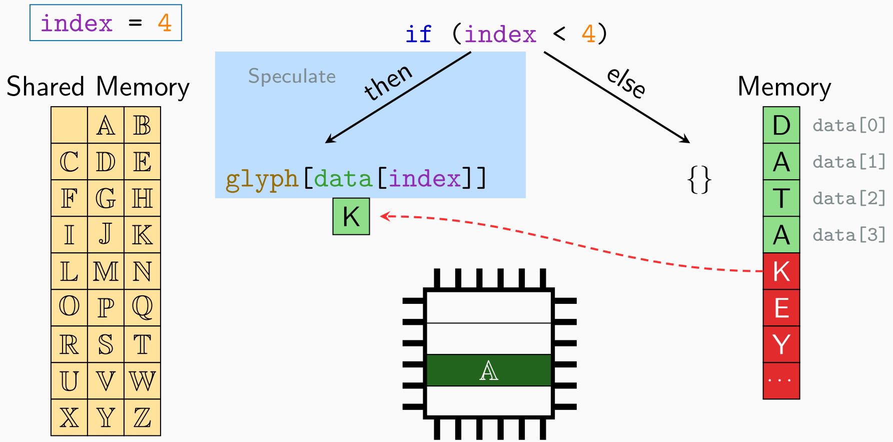
Spectre


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

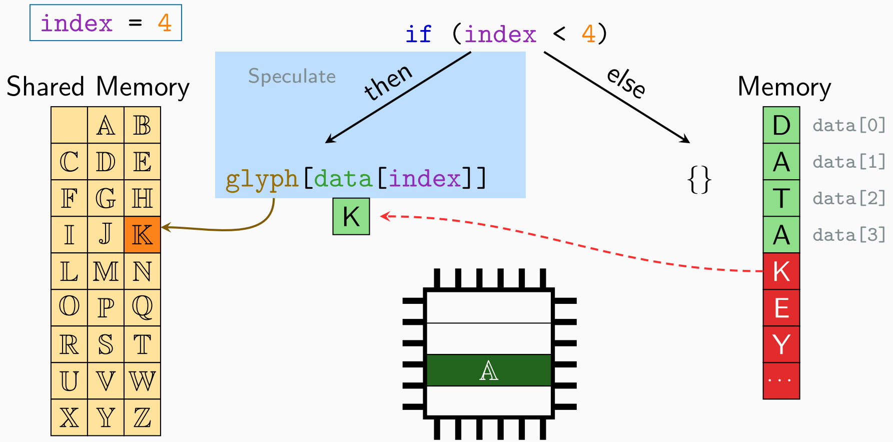
Spectre


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

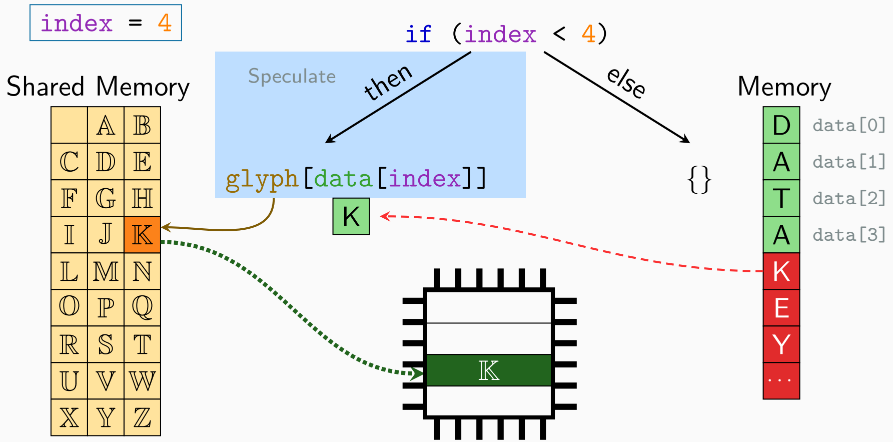
Spectre


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

Spectre

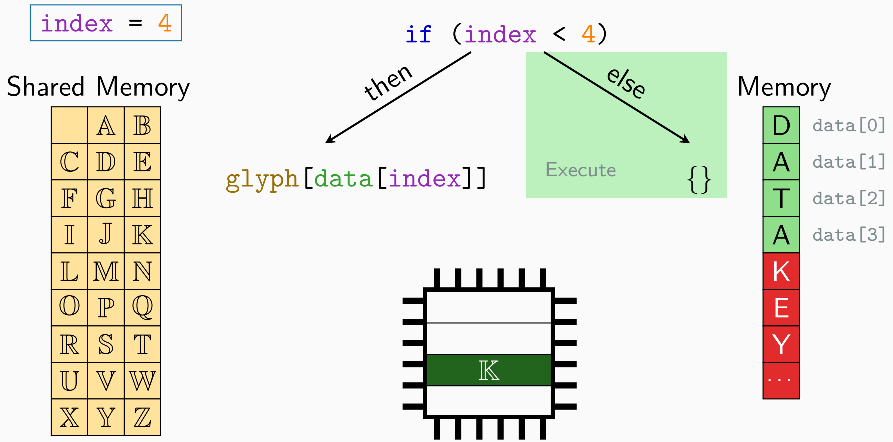

index = 4
if (index < 4)
Speculate
then
else
Shared Memory
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
glyph[data[index]]
K
{}
Memory
D A T A K E Y ...
data[0]
data[1]
data[2]
data[3]
A

Spectre



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

Spectre


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

Let's leak some data!

Exercise 3: When Predictions go Wrong

The Task:

Leak the secret password by exploiting the victim API.

Exercise 3: When Predictions go Wrong

The Task:

Leak the secret password by exploiting the victim API.

Hints for better results:

- Connect your laptop to power
- Close unrelated programs
- Use `permutate_index` function to prevent prefetch effects

Exercise 3: When Predictions go Wrong

The Task:

Leak the secret password by exploiting the victim API.

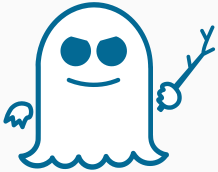
Hints for better results:

- Connect your laptop to power
- Close unrelated programs
- Use `permutate_index` function to prevent prefetch effects
- Be patient!

Exercise 3: When Predictions go Wrong

(`https:///challenge.attacking.systems/spectre.tar.gz`)

Mission Accomplished: Leaking actual data

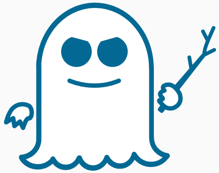


- Branch predictor mistrained

Mission Accomplished: Leaking actual data

- Branch predictor mistrained
- Data encoded in CPU cache

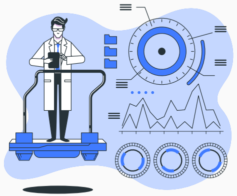
Mission Accomplished: Leaking actual data



- Branch predictor mistrained
- Data encoded in CPU cache

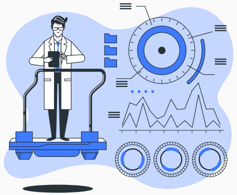
$\rightarrow$ **Inaccessible data leaked** through transient execution

Recap

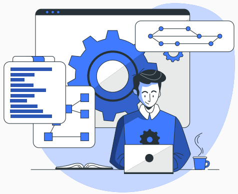


Observe Optimizations

Recap

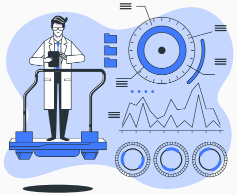


Observe Optimizations

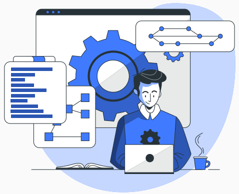


Leak Access Patterns
(Meta Data)

Recap



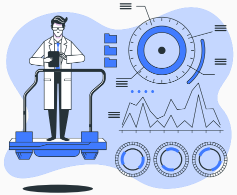
Observe Optimizations



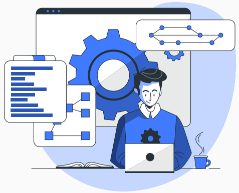
Leak Access Patterns (Meta Data)

Leaking Inaccessible Data

Recap



Observe Optimizations



Leak Access Patterns (Meta Data)

Leaking Inaccessible Data

CPU optimizations can lead to **severe data leakage!**

Turning Timing Differences into Data Leakage

Daniel Weber, Michael Schwarz

December 6, 2022

CISPA Helmholtz Center for Information Security

References

- Icons and Images from `storyset.com` and `thenounproject.com`
- Some Animations from Moritz Lipp