

SCORE-to-Leak Forwarding

There and Back Again

Claudio Canella (@cc0x1f), Lukas Giner (@redrabbyte), Michael Schwarz (@misc0110)

October 2, 2020

Graz University of Technology

CISPA Helmholtz Center for Information Security



Claudio Canella

PhD student @ Graz University of Technology

 @cc0x1f

 claudio.canella@iaik.tugraz.at

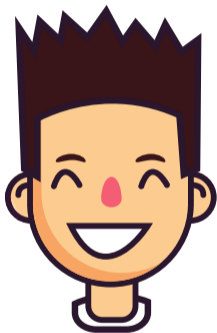


Lukas Giner

PhD student @ Graz University of Technology

🐦 @redrabbyte

✉️ lukas.giner@iaik.tugraz.at



Michael Schwarz

Faculty @ CISA Helmholtz Center for Information Security

🐦 @misc0110

✉ michael.schwarz@cispa.saarland



- How do loads handle **previous stores**?



- How do loads handle **previous stores**?
- How is Meltdown **mitigated in hardware**?



- How do loads handle **previous stores**?
- How is Meltdown **mitigated in hardware**?
- Can we abuse these mechanisms for **new attacks**?



- How do loads handle **previous stores**?
- How is Meltdown **mitigated in hardware**?
- Can we abuse these mechanisms for **new attacks**?
- Can we **mitigate** such attacks **efficiently**?



- Mental model of CPU is simple



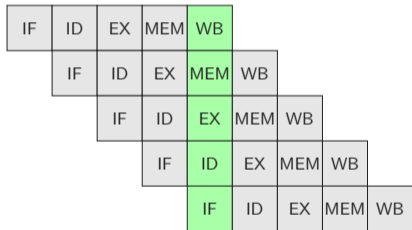
- Mental model of CPU is simple
- Instructions are executed **in program order**



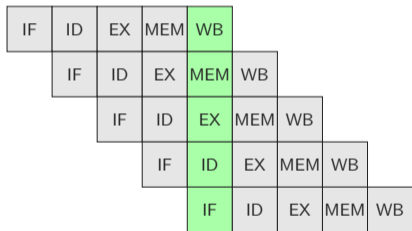
- Mental model of CPU is simple
- Instructions are executed **in program order**
- Pipeline **stalls** when stages are not ready



- Mental model of CPU is simple
- Instructions are executed **in program order**
- Pipeline **stalls** when stages are not ready
- If data is **not cached**, we need to wait

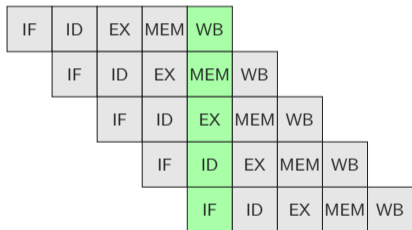


- Instructions are...
 - **fetched** (IF) from the L1 Instruction Cache



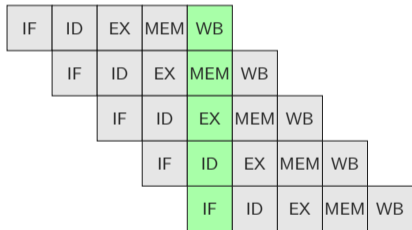
- Instructions are...

- **fetch**ed (IF) from the L1 Instruction Cache
- **dec**oded (ID)

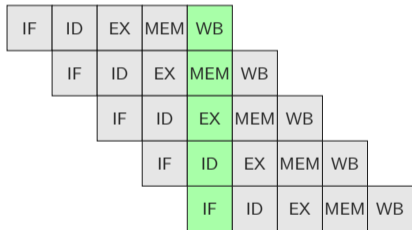


- Instructions are...

- **fetched** (IF) from the L1 Instruction Cache
- **decoded** (ID)
- **executed** (EX) by execution units



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



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



- No dependency between instructions? Why wait!



- No dependency between instructions? Why wait!
- Execute **out of order**



- No dependency between instructions? Why wait!
- Execute **out of order**, retire **in order**



- No dependency between instructions? Why wait!
- Execute **out of order**, retire **in order**
- What happens when an **instruction faults**?



- No dependency between instructions? Why wait!
- Execute **out of order**, retire **in order**
- What happens when an **instruction faults**?
- Undo out-of-order effects → instructions were **transient**



- Transient **instructions** are **undone**, what good are they?



- Transient **instructions** are **undone**, what good are they?
- Some changes are **persistent**, but invisible **architecturally**



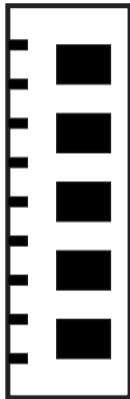
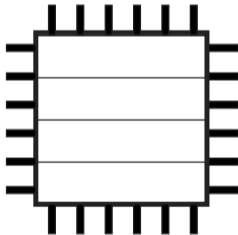
- Transient **instructions** are **undone**, what good are they?
 - Some changes are **persistent**, but invisible **architecturally**
- **encode data** in **microarchitectural state**



- Transient **instructions** are **undone**, what good are they?
 - Some changes are **persistent**, but invisible **architecturally**
- **encode data** in **microarchitectural state**
- The **cache** is a simple solution

```
mov var, rax
```

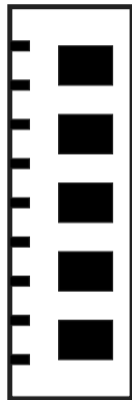
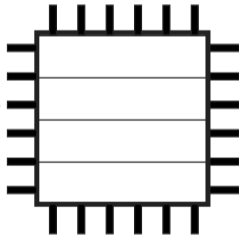
```
mov var, rbx
```



```
mov var, rax
```

```
mov var, rbx
```

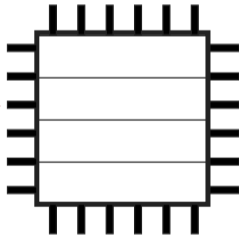
Cache miss



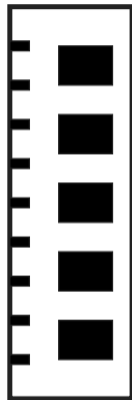
```
mov var, rax
```

```
mov var, rbx
```

Cache miss



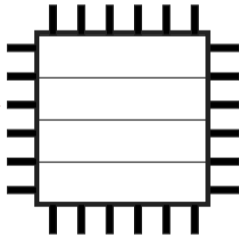
Request



```
mov var, rax
```

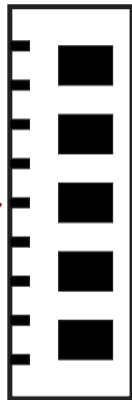
```
mov var, rbx
```

Cache miss

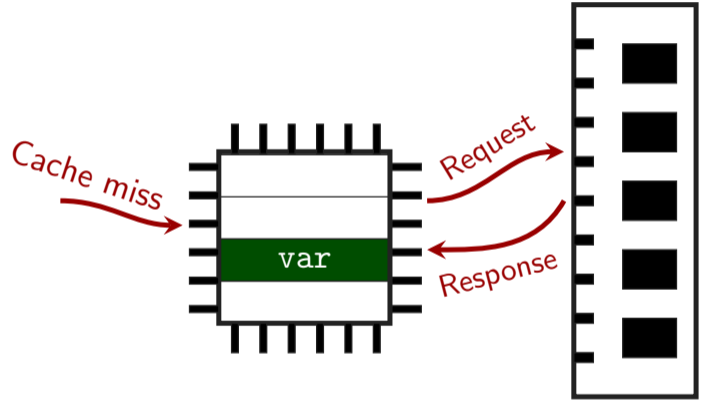


Request

Response



```
mov var, rax  
mov var, rbx
```

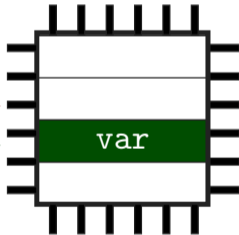


```
mov var, rax
```

```
mov var, rbx
```

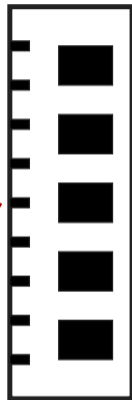
Cache miss

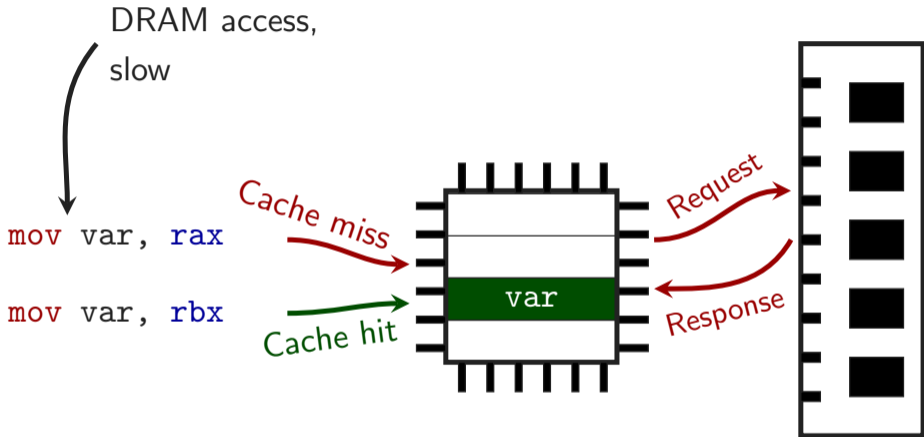
Cache hit

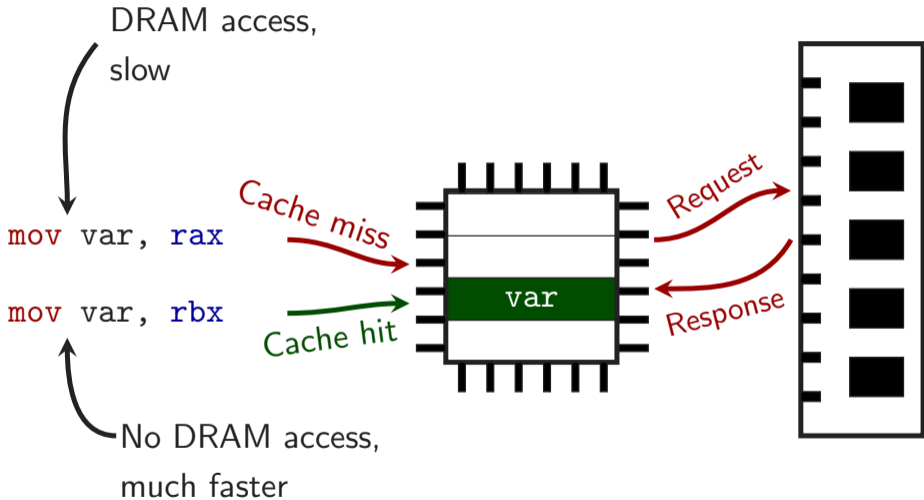


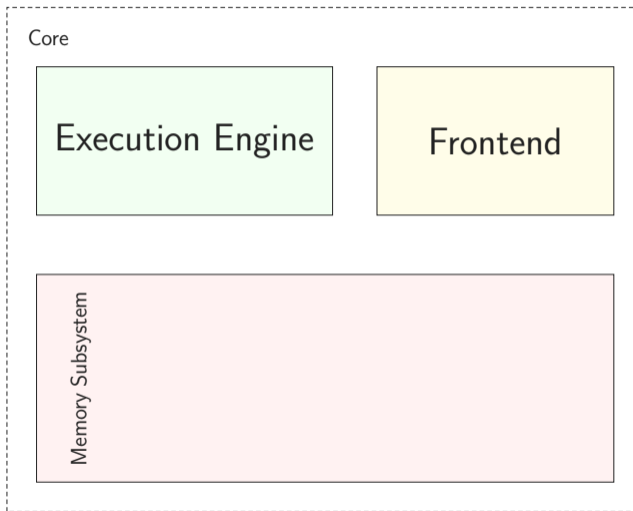
Request

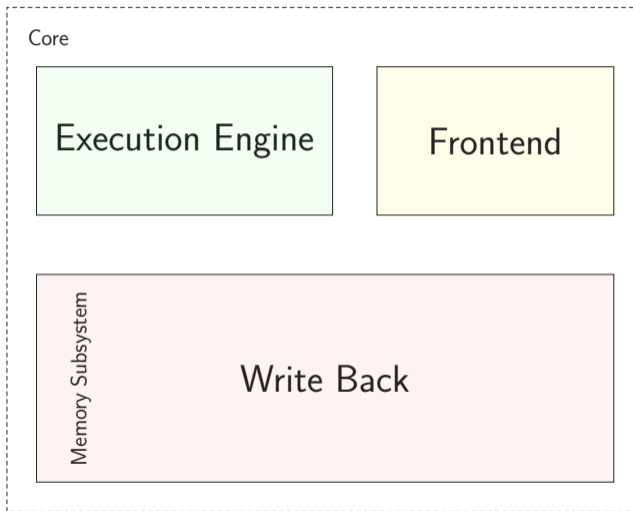
Response

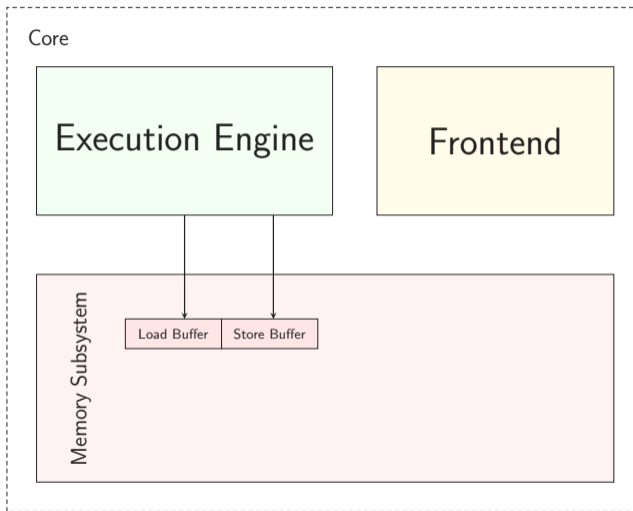


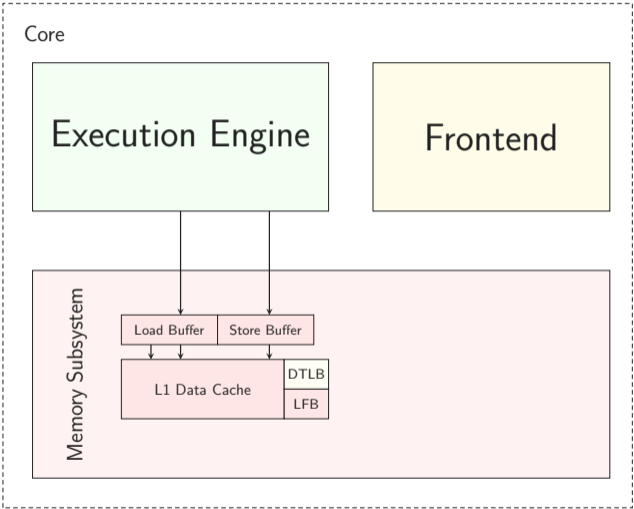


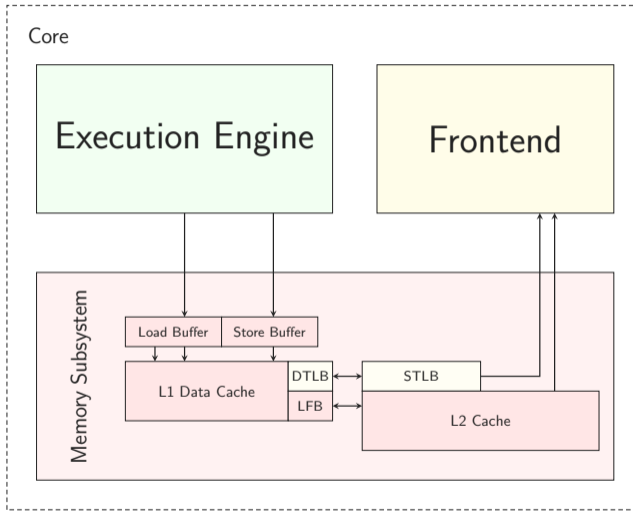


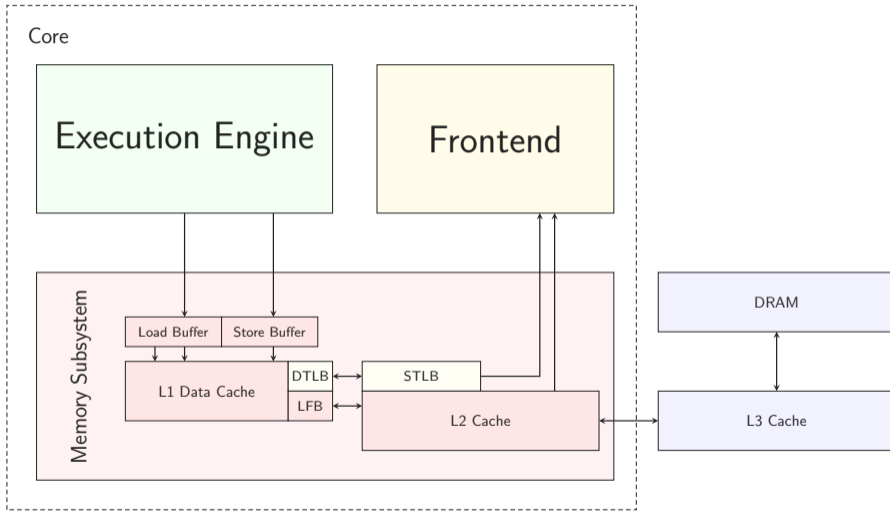






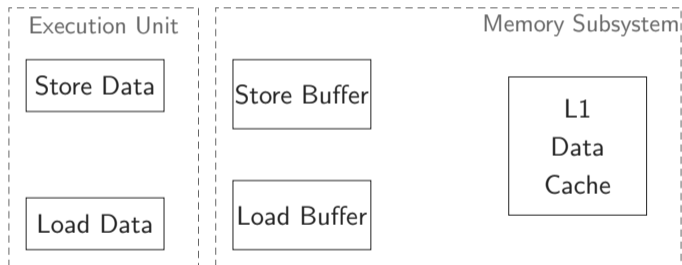






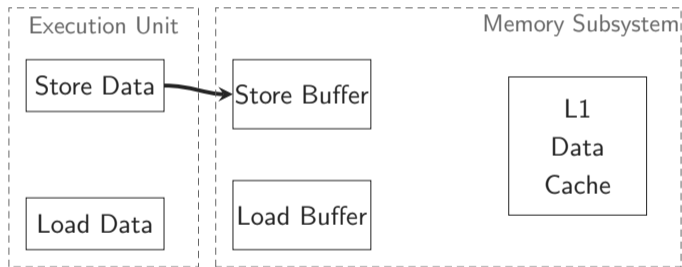
`mov data → [address]`

`mov [address] → reg`



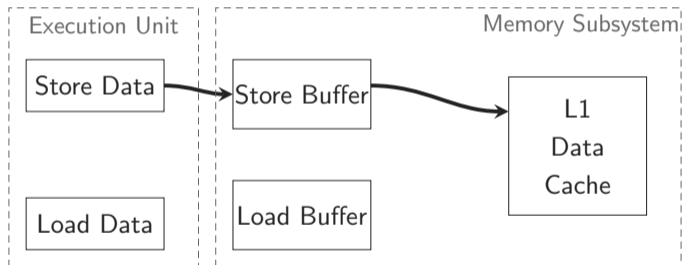
`mov data → [address]`

`mov [address] → reg`



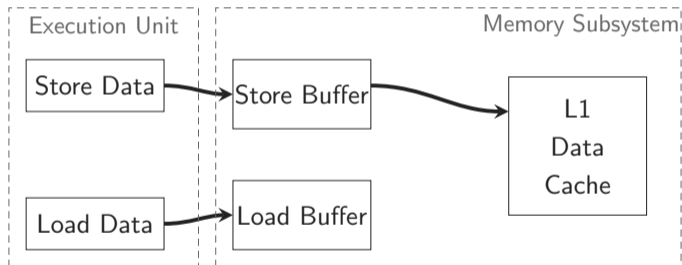
`mov data → [address]`

`mov [address] → reg`

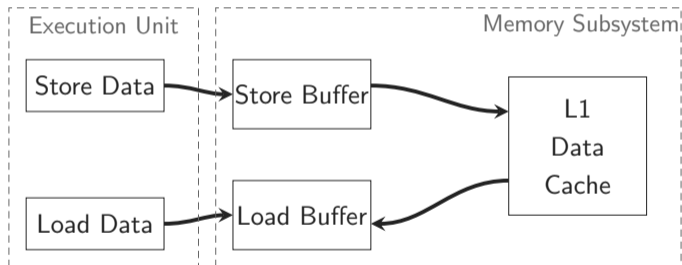


`mov data → [address]`

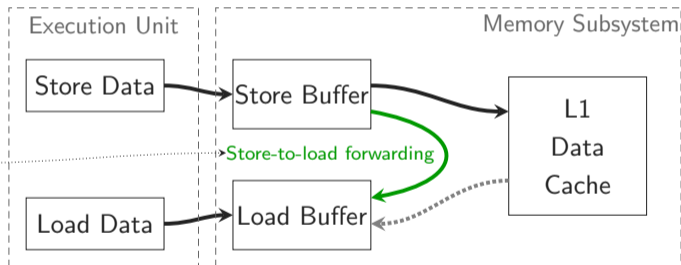
`mov [address] → reg`



```
mov data → [ address ]  
      ↙      ↘  
      Compare  
mov [ address ] → reg
```



`mov data → [address]`
Match
`mov [address] → reg`



True Positive

Addresses



Forwarding



True Negative

Addresses



Forwarding



True Positive

Addresses



Forwarding



True Negative

Addresses



Store-to-Leak

Forwarding



True Positive

Addresses



Forwarding



False Positive



True Negative

Addresses



Store-to-Leak

False Negative



Forwarding



True Positive

Addresses



Forwarding



Fallout

False Positive



Store-to-Leak

True Negative

Addresses



Forwarding



False Negative



True Positive

Addresses



Forwarding



False Positive



Fallout



Store-to-Leak

True Negative

Addresses



Forwarding

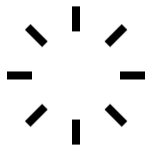


False Negative

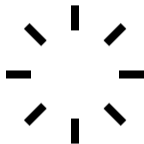


Spectre-STL

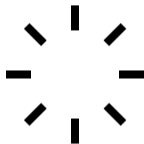




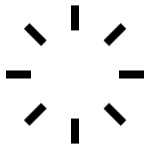
- Optimization during **transient execution**



- Optimization during **transient execution**
- Failures **not** visible **architecturally**



- Optimization during **transient execution**
- Failures **not** visible **architecturally**
- Store-to-Leak relies on **correct matching**



- Optimization during **transient execution**
- Failures **not** visible **architecturally**
- Store-to-Leak relies on **correct matching**
- Only stores to **valid addresses** are forwarded

Intel Architecture Optimization Reference Manual

[The store execution phase] Fills the store buffers with **linear and physical address** and data.

Intel Architecture Optimization Reference Manual

[The store execution phase] Fills the store buffers with **linear and physical address** and data. Once store **address** and data are **known**, the store data **can be forwarded** to the following load operations that need it. [...]



- Virtual address that is **physically backed** → valid



- Virtual address that is **physically backed** → valid
- Permission checks are **deferred**




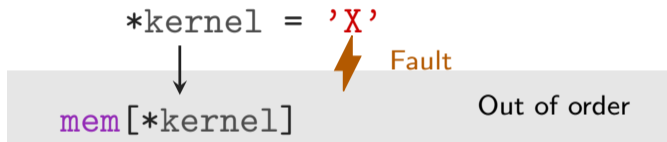
- Virtual address that is **physically backed** → valid
 - Permission checks are **deferred**
- Exploited in Meltdown attacks



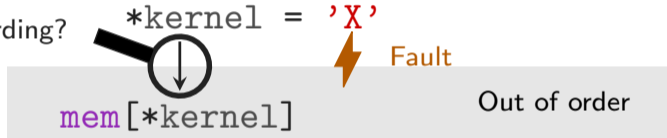
- Virtual address that is **physically backed** → valid
 - Permission checks are **deferred**
- Exploited in Meltdown attacks
- Store-to-load forwarding on **inaccessible** addresses

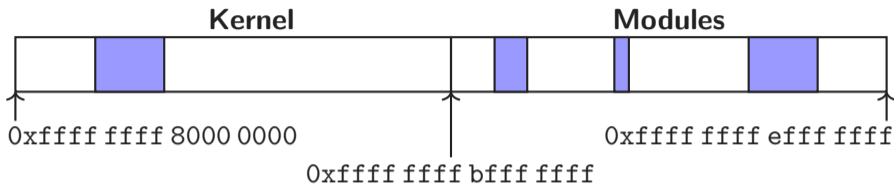
```
*kernel = 'X'
```

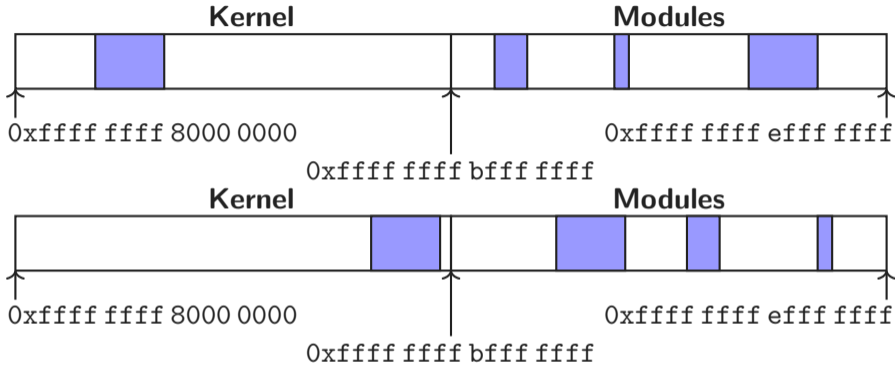
*kernel = 'X'
 Fault

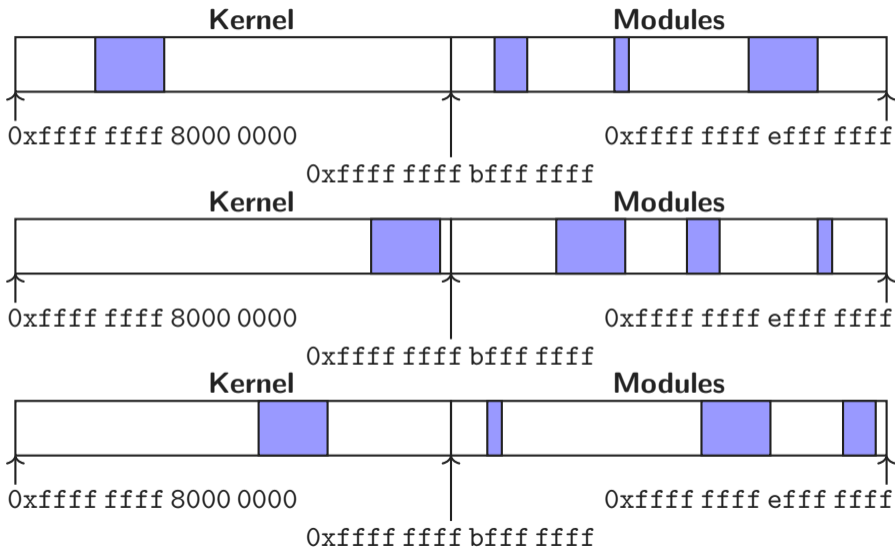


Store-to-load forwarding?



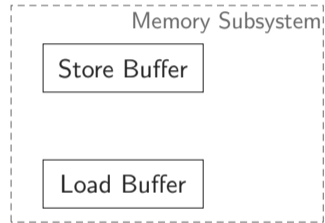
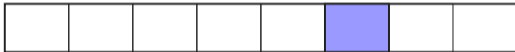




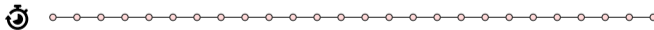


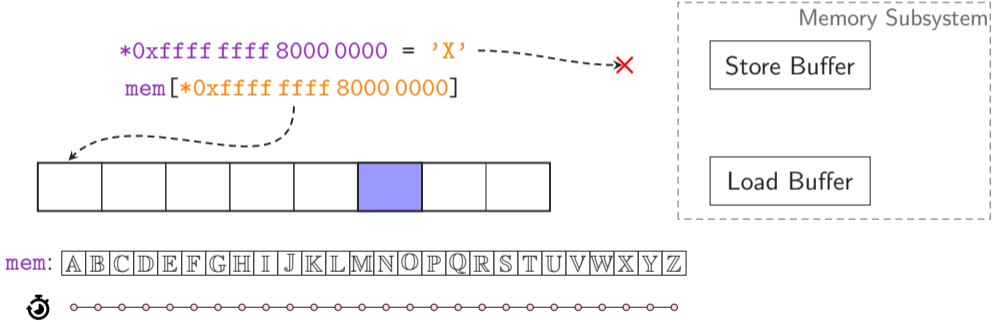
```
*0xffff ffff 8000 0000 = 'X'
```

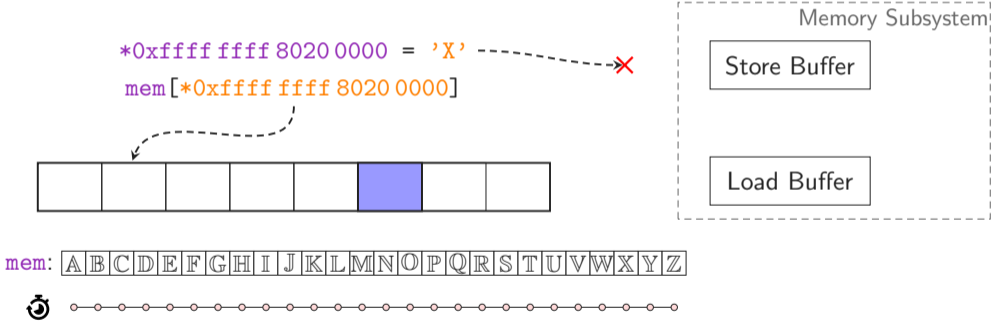
```
mem[*0xffff ffff 8000 0000]
```



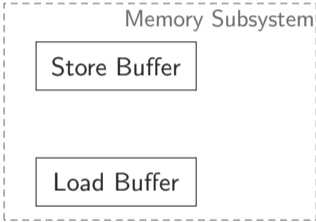
```
mem: ABCDEFGHIJKLMNOPQRSTUVWXYZ
```





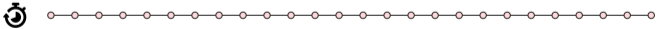


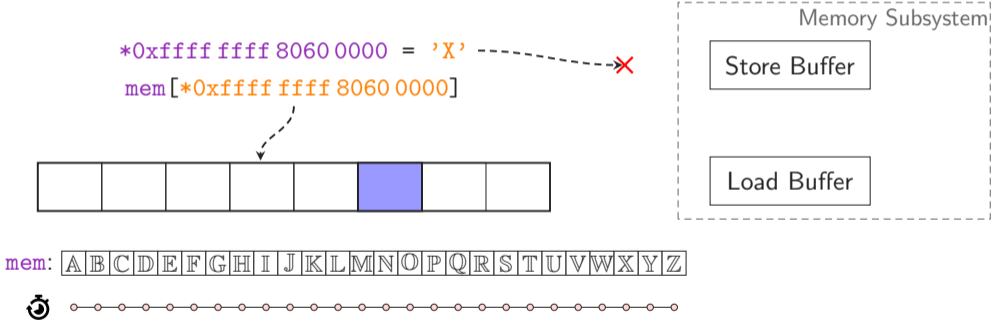
```
*0xffff ffff 8040 0000 = 'X'  
mem[*0xffff ffff 8040 0000]
```



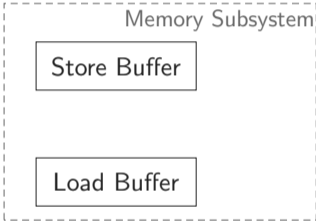
mem:

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
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



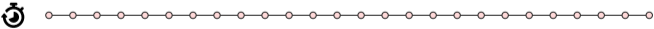


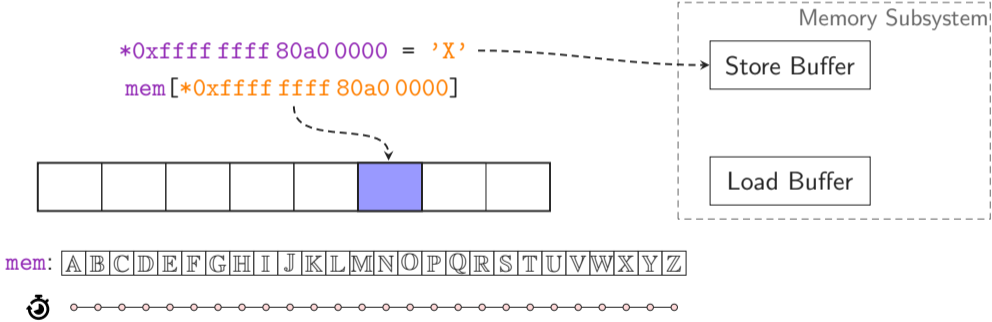
```
*0xffff ffff 8080 0000 = 'X'  
mem[*0xffff ffff 8080 0000]
```

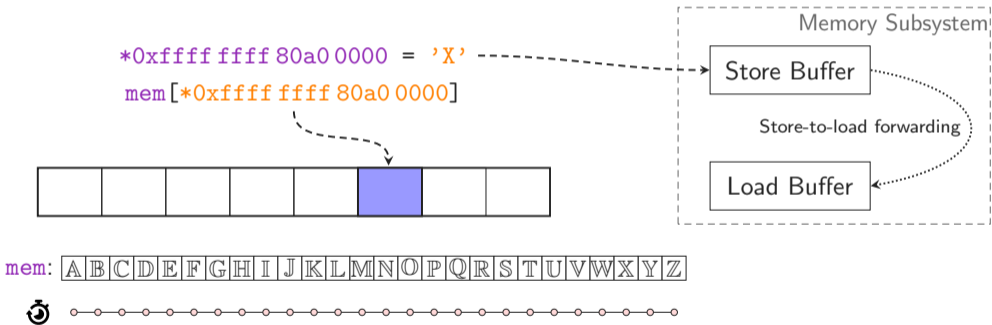


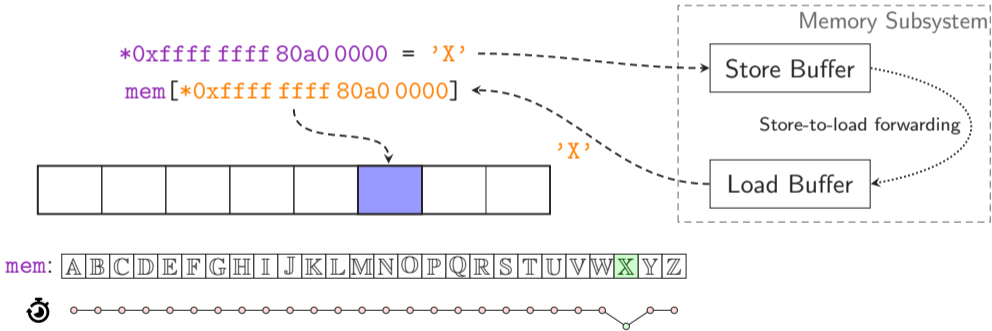
mem:

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
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



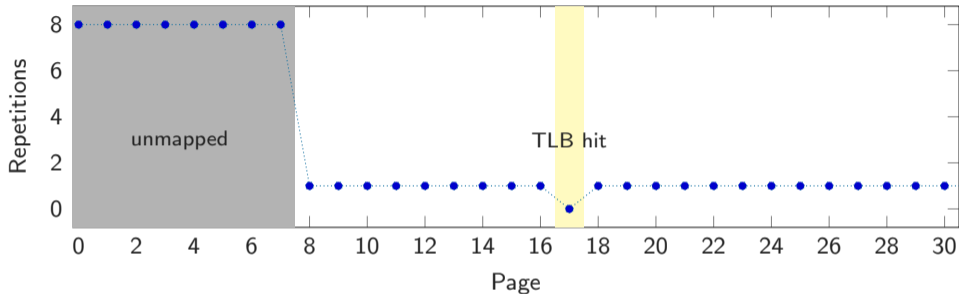




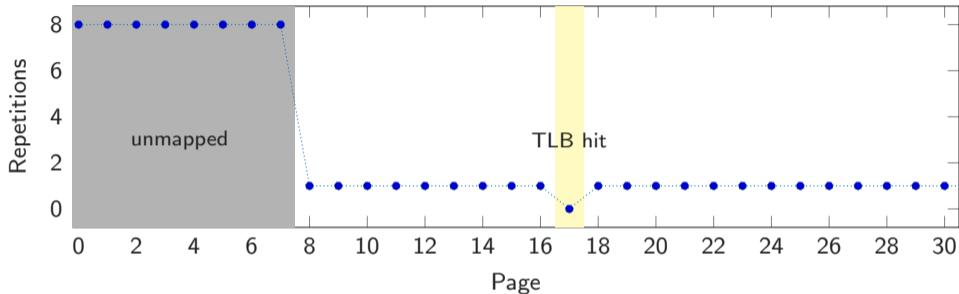


What if an address
is **valid**
but **unknown**?

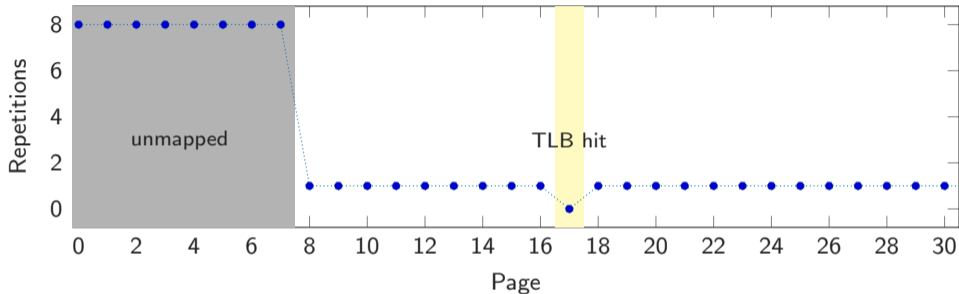




- Forward on **first** try → in **TLB**

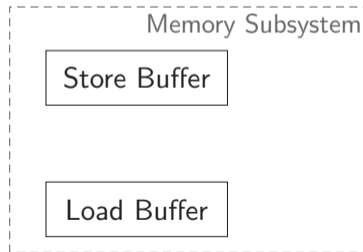
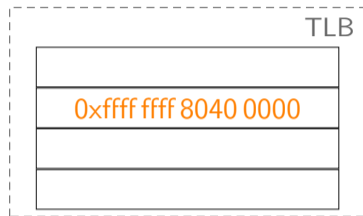
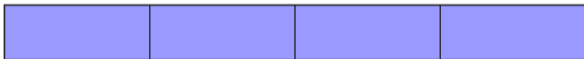


- Forward on **first** try → in **TLB**
- On **second** try → **valid**, not in TLB

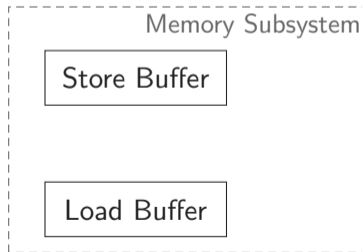
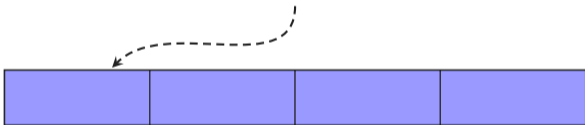


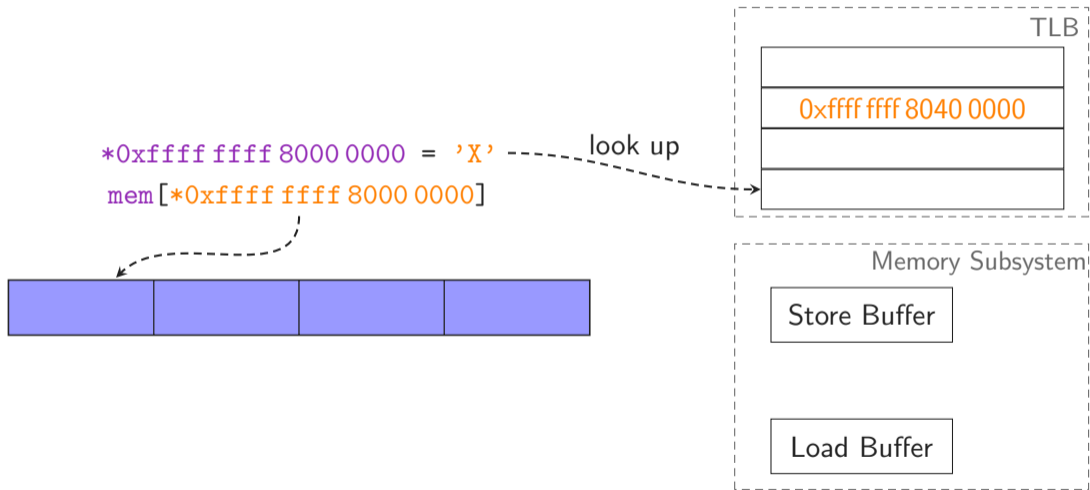
- Forward on **first** try → in **TLB**
- On **second** try → **valid**, not in TLB
- **Not** forwarded → **unmapped**

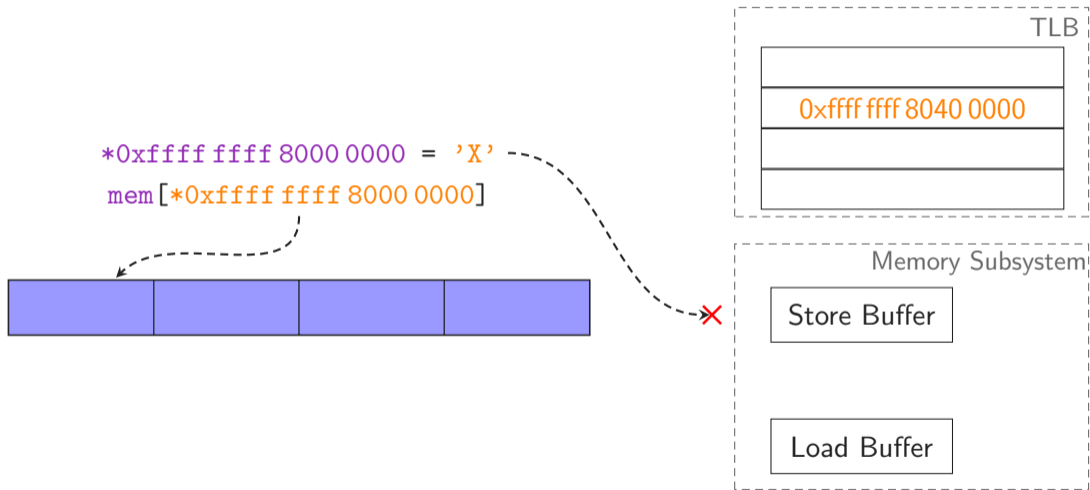
```
*0xffff ffff 8000 0000 = 'X'  
mem[*0xffff ffff 8000 0000]
```



```
*0xffff ffff 8000 0000 = 'X'  
mem[*0xffff ffff 8000 0000]
```

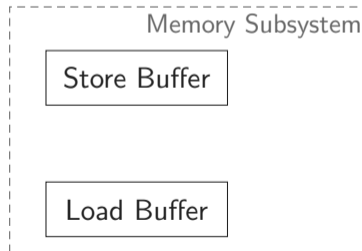
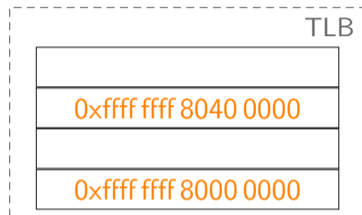
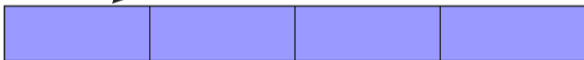




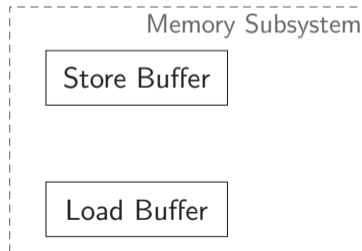
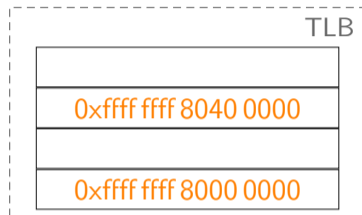
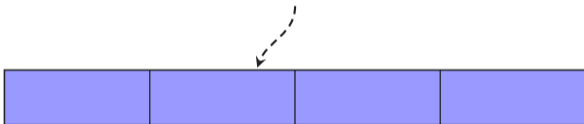


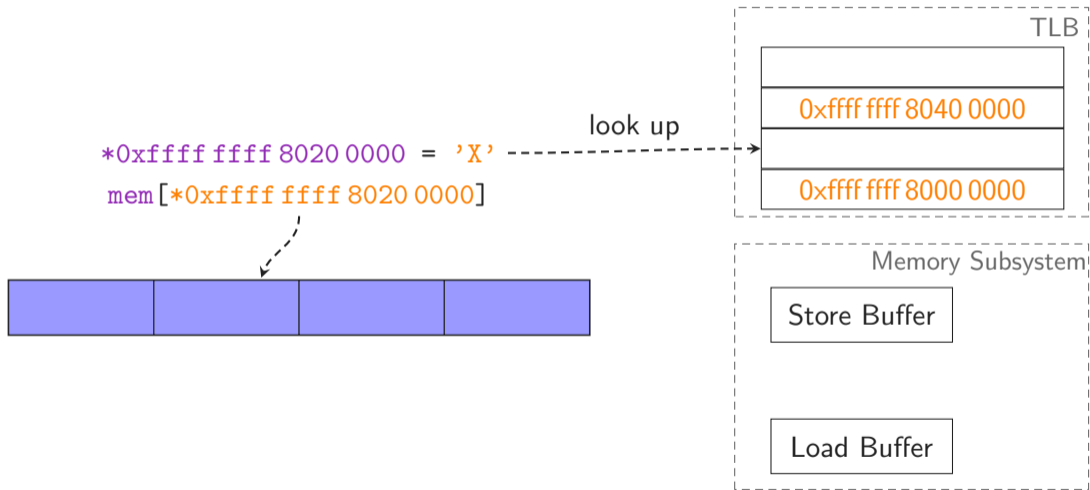
`*0xffff ffff 8000 0000 = 'X'`

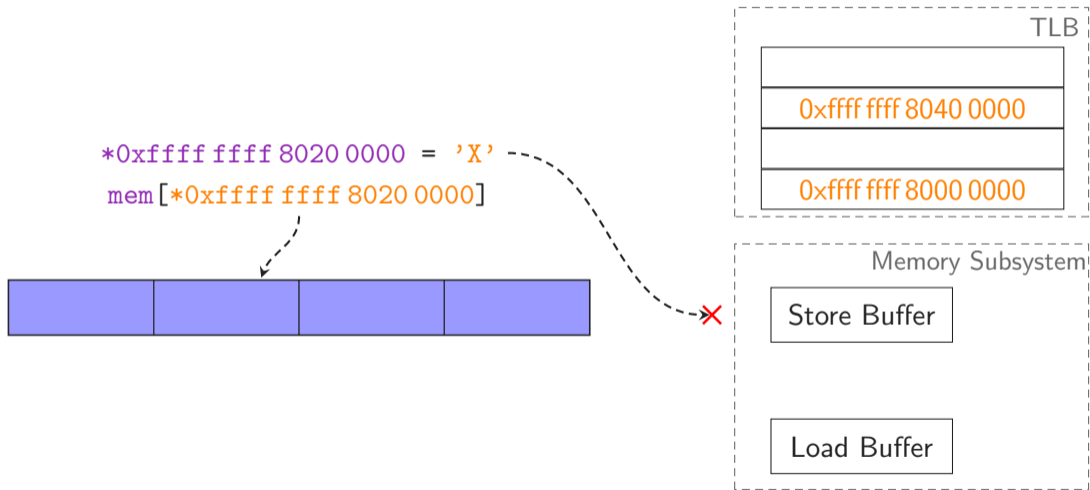
`mem[*0xffff ffff 8000 0000]`



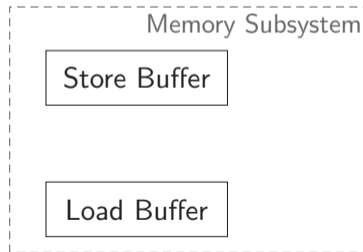
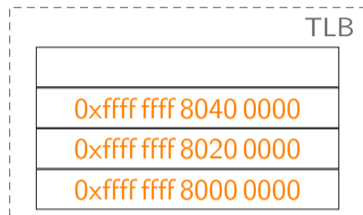
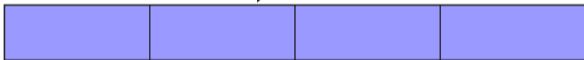
```
*0xffff ffff 8020 0000 = 'X'  
mem[*0xffff ffff 8020 0000]
```



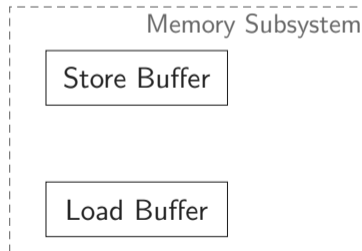
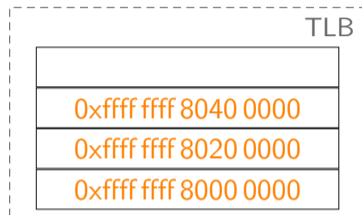
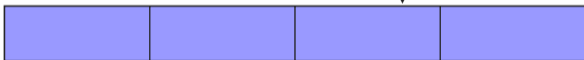


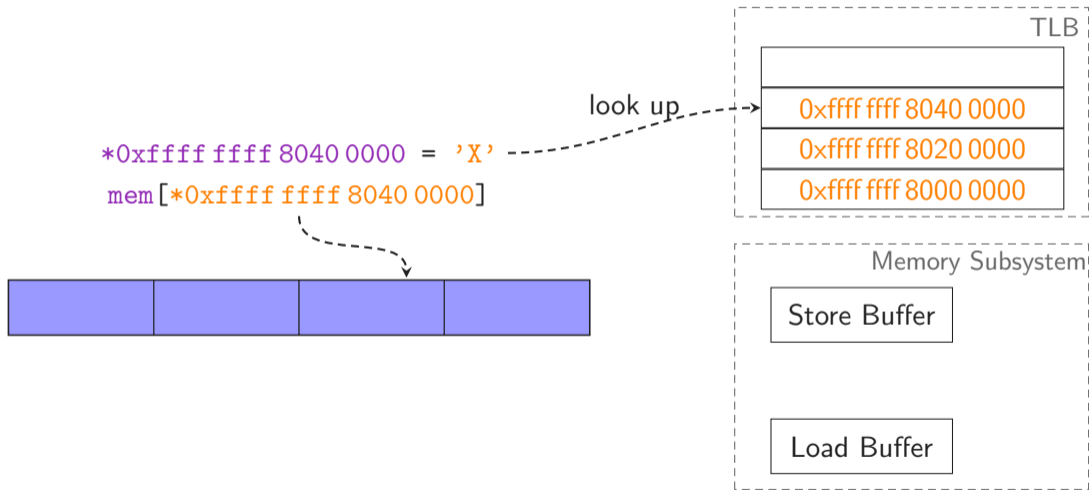


```
*0xffff ffff 8020 0000 = 'X'  
mem[*0xffff ffff 8020 0000]
```

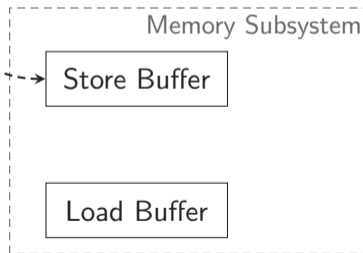
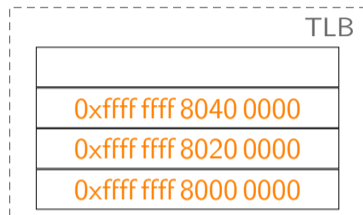
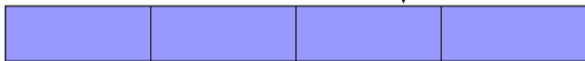


```
*0xffff ffff 8040 0000 = 'X'  
mem[*0xffff ffff 8040 0000]
```

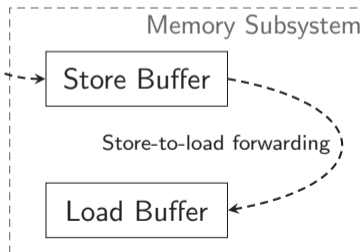
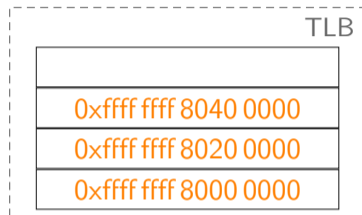
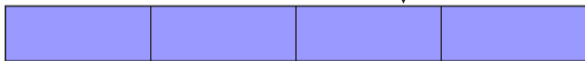


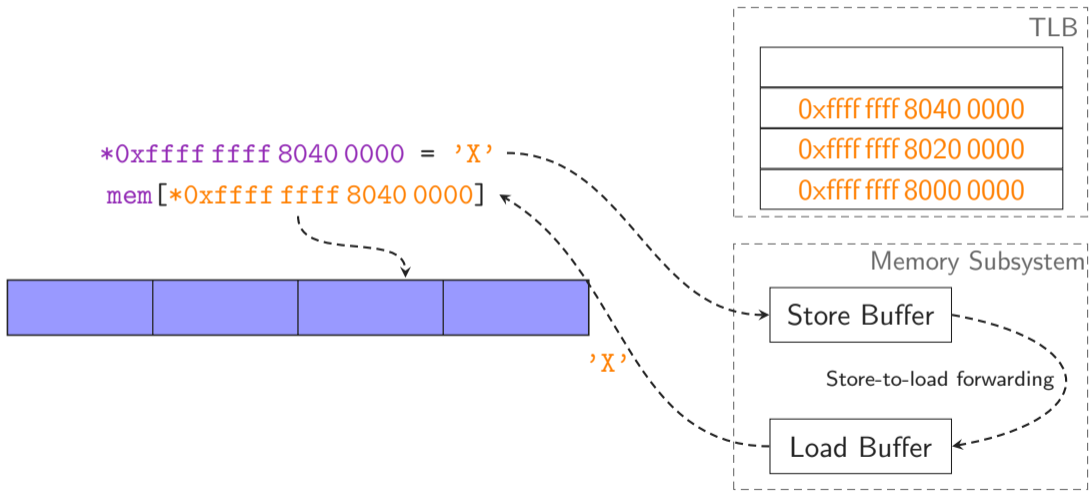


```
*0xffff ffff 8040 0000 = 'X'  
mem[*0xffff ffff 8040 0000]
```

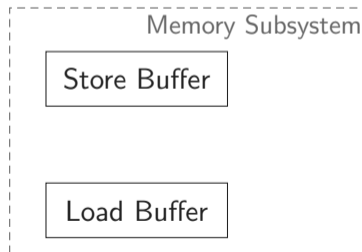
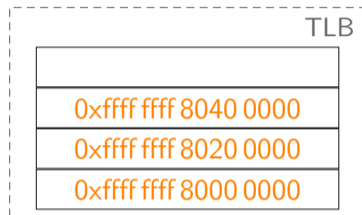
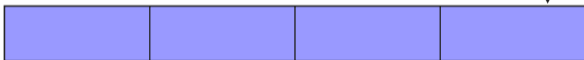


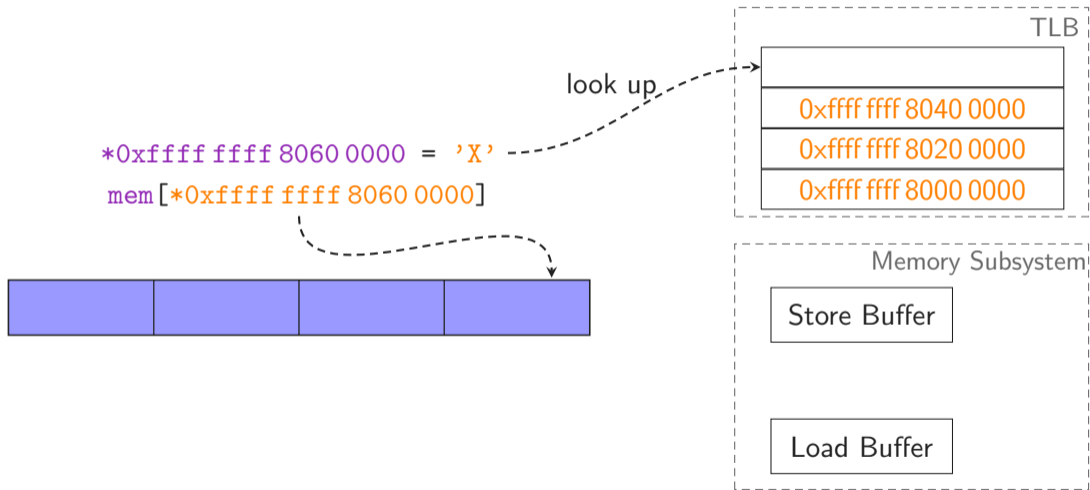
```
*0xffff ffff 8040 0000 = 'X'  
mem[*0xffff ffff 8040 0000]
```

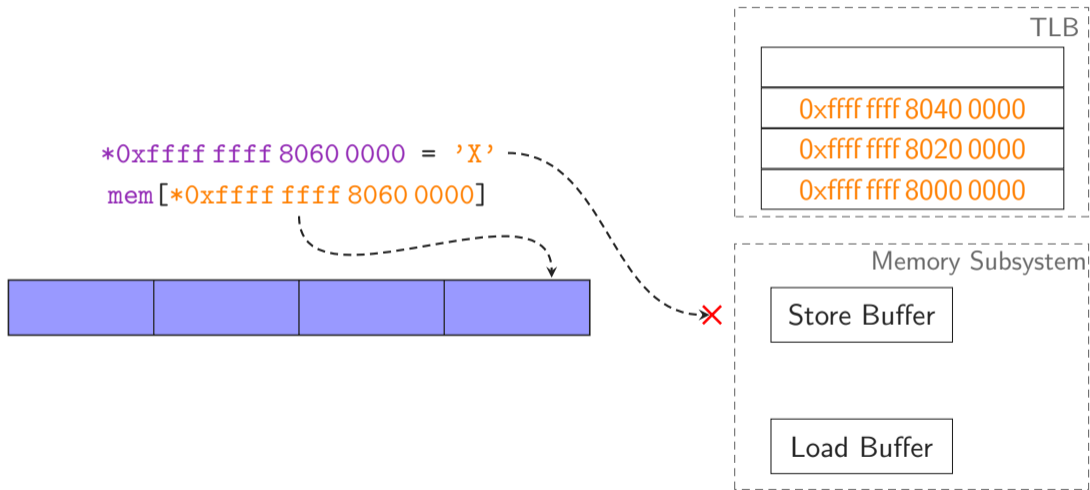




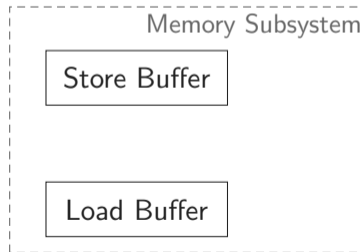
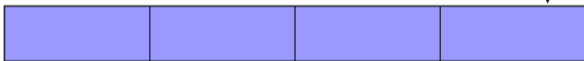
```
*0xffff ffff 8060 0000 = 'X'  
mem[*0xffff ffff 8060 0000]
```

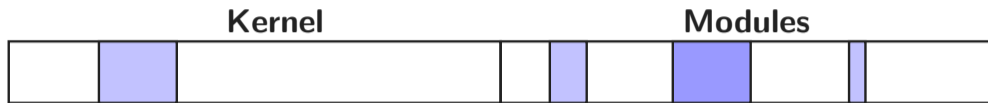


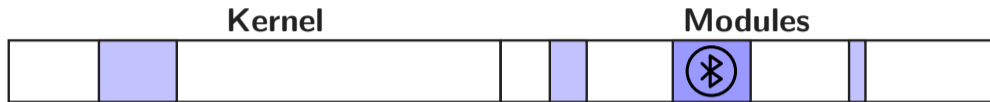


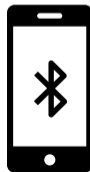
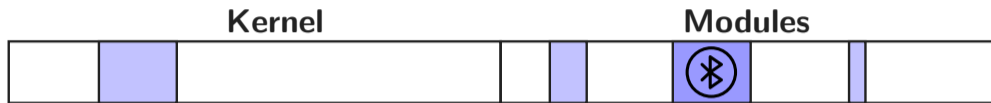


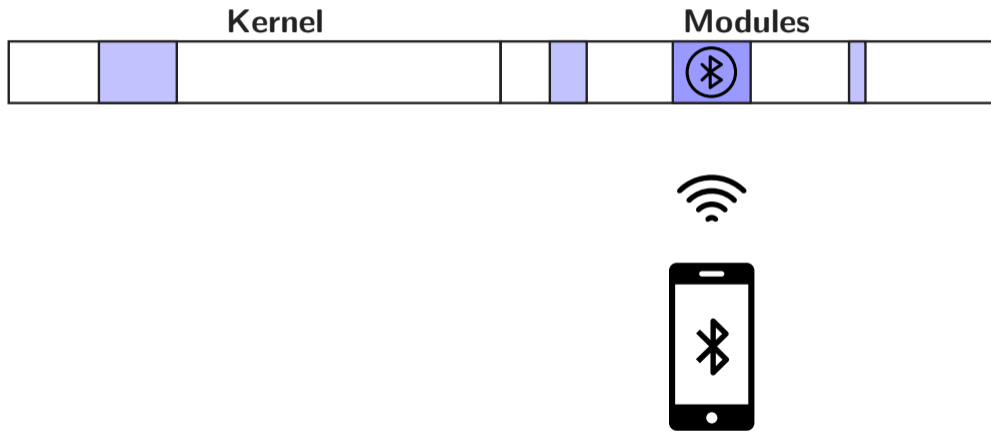
```
*0xffff ffff 8060 0000 = 'X'  
mem[*0xffff ffff 8060 0000]
```

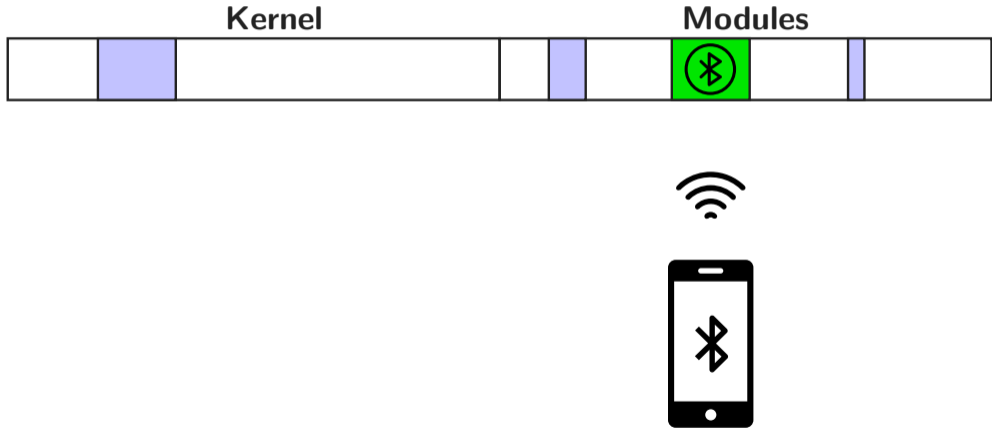


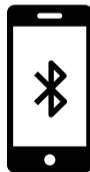
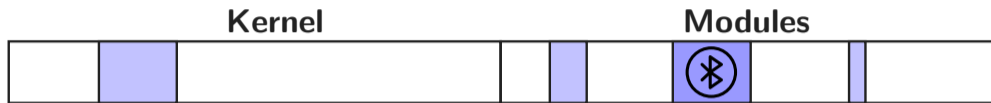












```
lginer@t460slug .._store_buffer/experiments/access_trace (git)-[master] % ./kmod_map_x86
```

Terminal - lginer@t460slug: ~/TU/2018_store_buffer/experiments/access_trace

File Edit View Terminal Tabs Help

```
lginer@t460slug ..]store_buffer/experiments/access_trace (git)-[master] %  
./kmod_map_x86 | python2 find_module.py bluetooth iwlwifi | ./jumper 1 4
```

15:16 D: 25 August



SmartThings

Autowergabe

1482Vlog

YouTube

Royalty Free

Webinjection

LastPass

Autofill with LastPass

Tap to show matching logins

BENACHR. SPERREN

ALLE LÖSCHEN



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

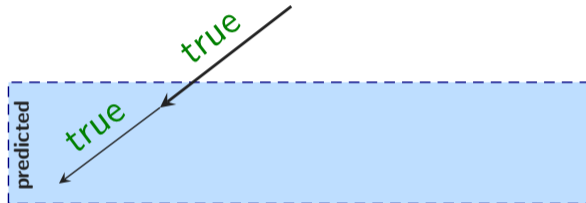


```
if <access in bounds>
```

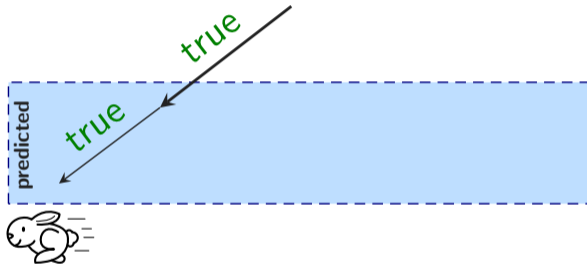
true

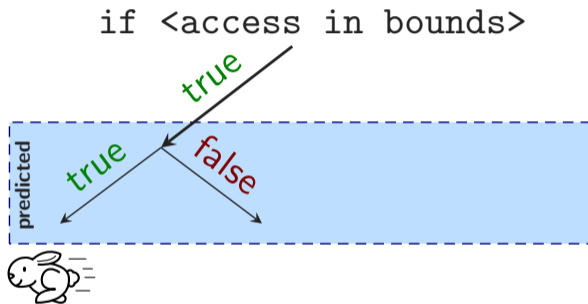


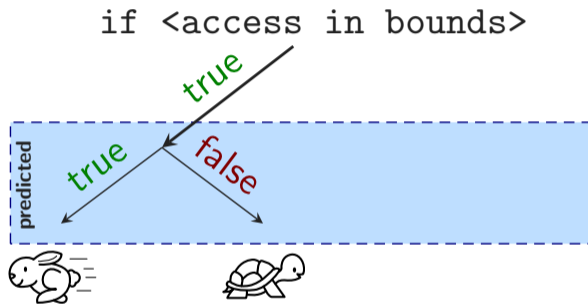
```
if <access in bounds>
```

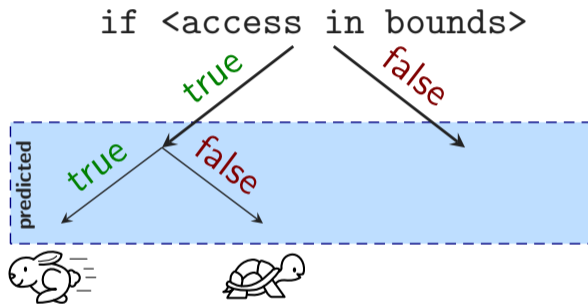


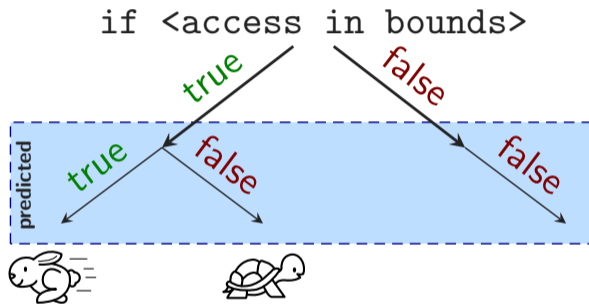
if <access in bounds>

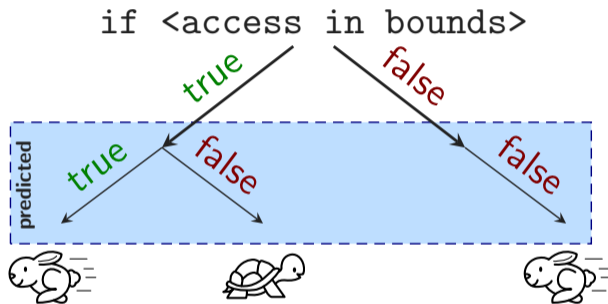


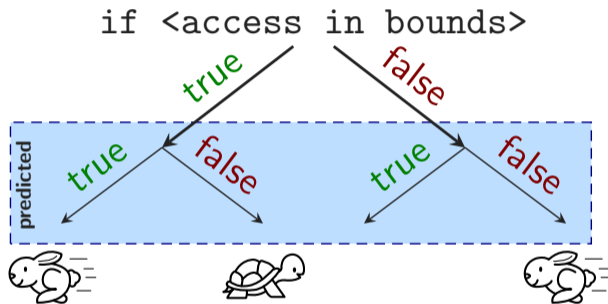


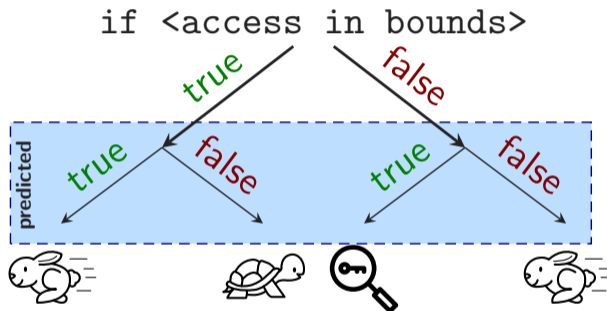










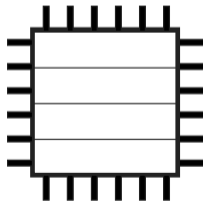


`index = 0`

Shared Memory

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

```
if (index < 4)
  then
    glyph[data[index]]
  else
    {}
```



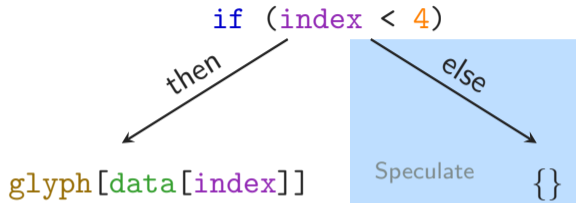
Memory

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

```
index = 0
```

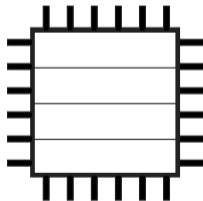
Shared Memory

	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



Memory

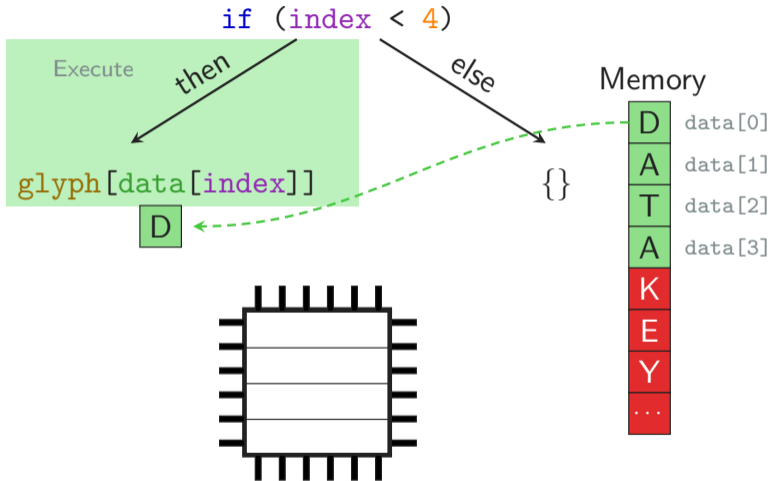
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

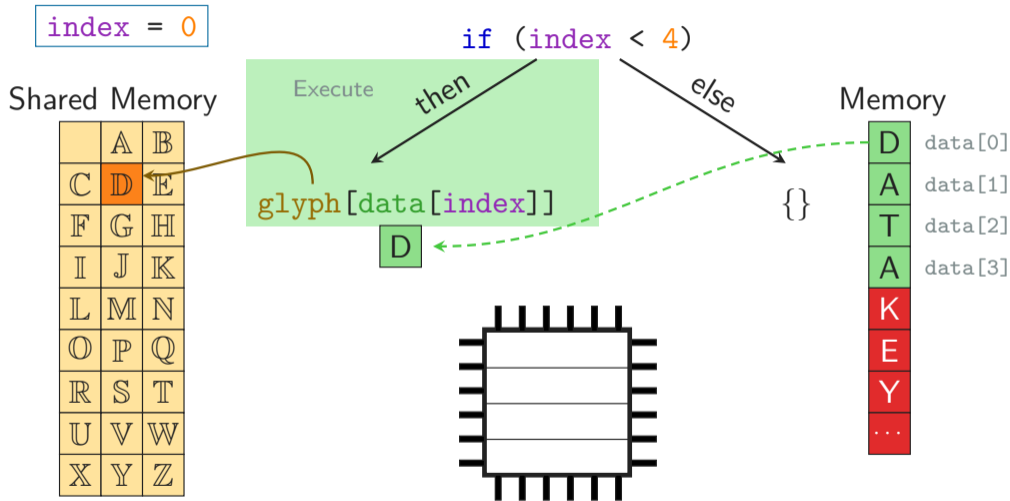


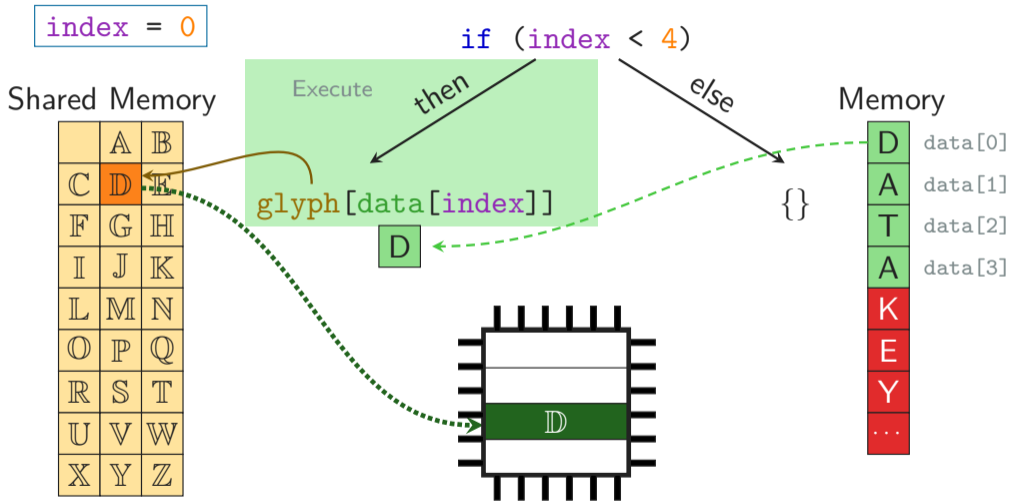

```
index = 0
```

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







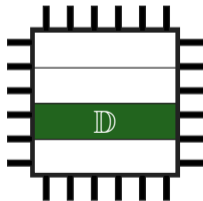
`index = 1`

Shared Memory

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

```

if (index < 4)
  then
    glyph[data[index]]
  else
    {}
  
```



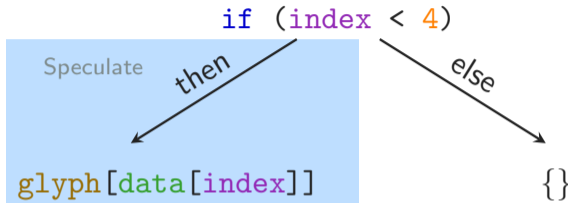
Memory

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

index = 1

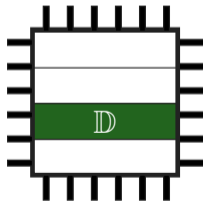
Shared Memory

	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



Memory

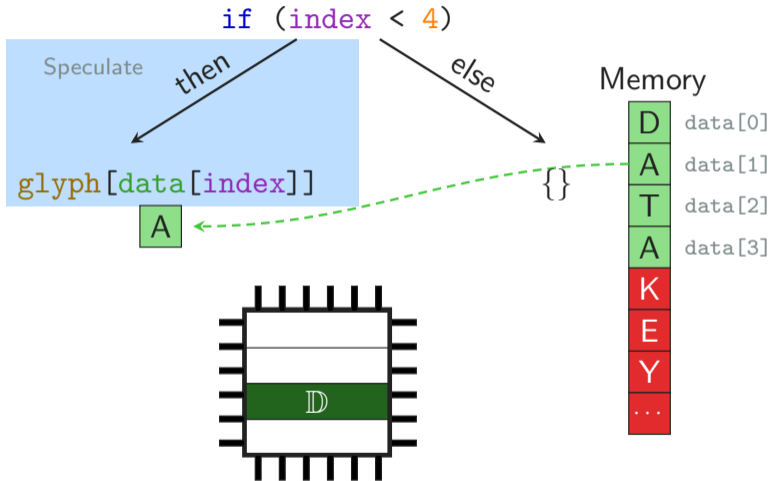
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

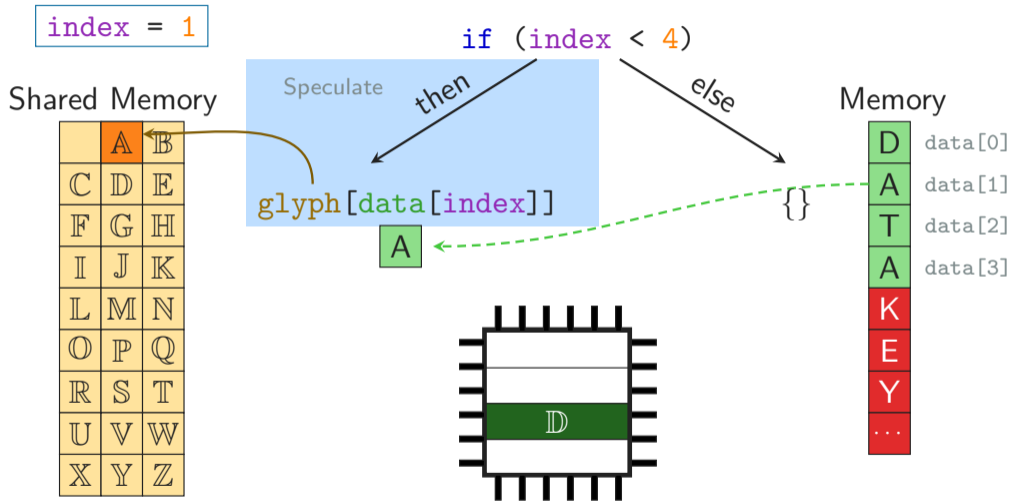


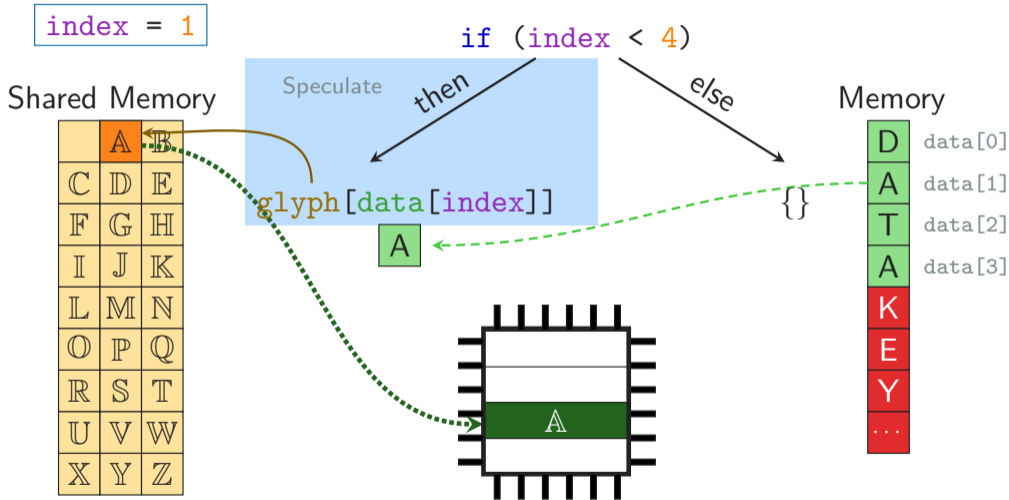
index = 1

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



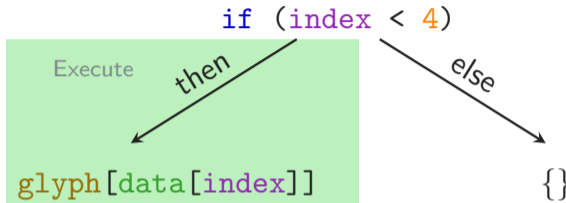





```
index = 1
```

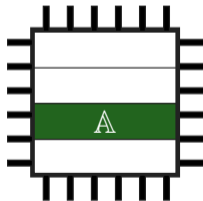
Shared Memory

	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



Memory

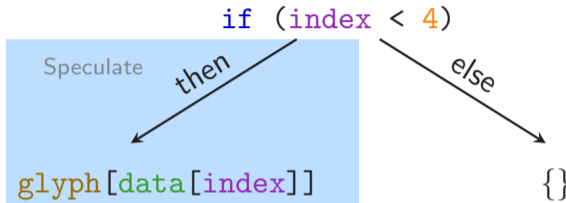
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	



`index = 2`

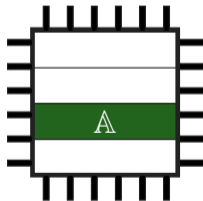
Shared Memory

	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



Memory

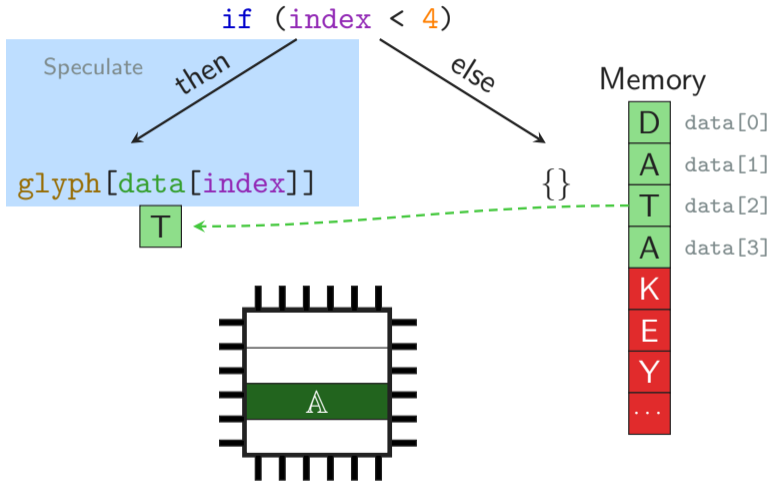
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

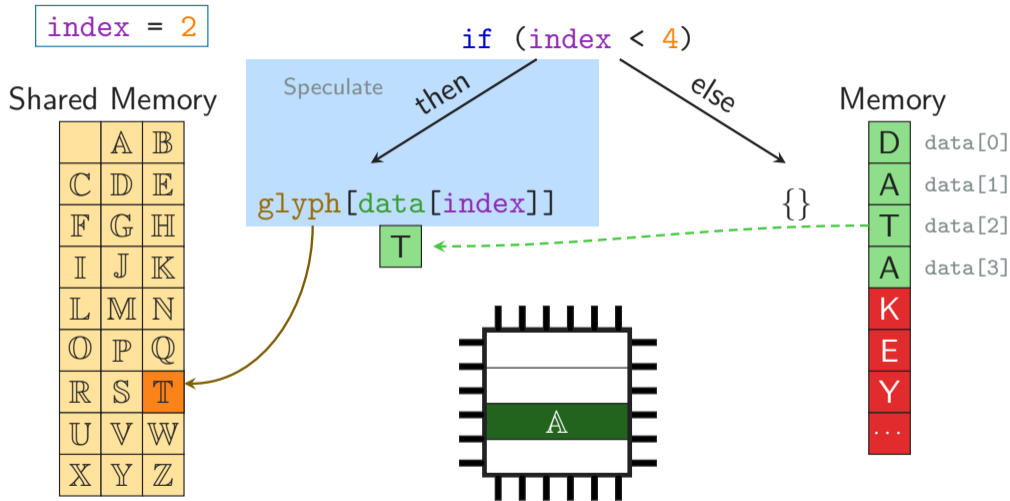


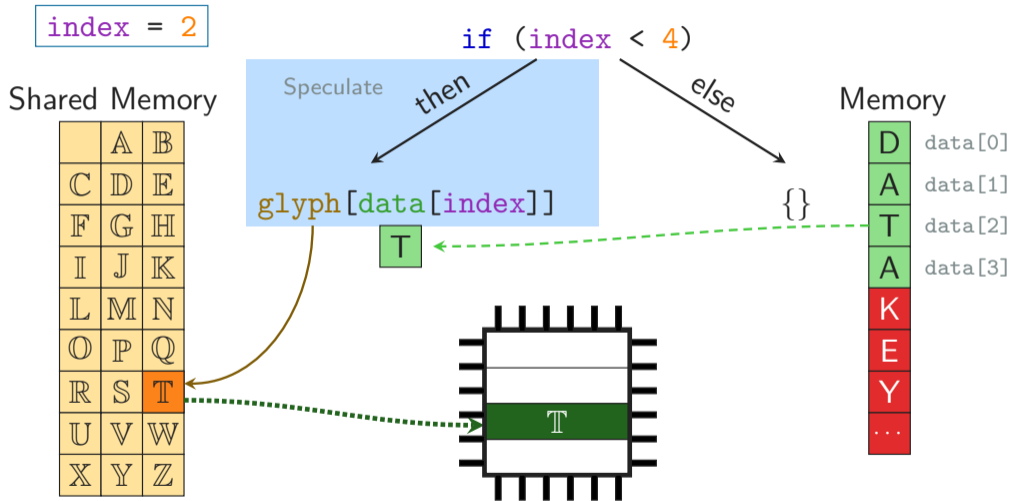
index = 2

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



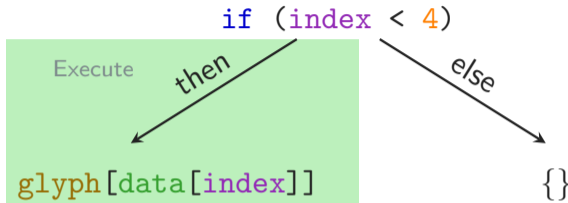




`index = 2`

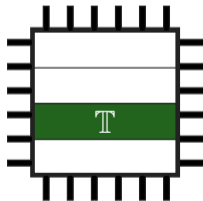
Shared Memory

	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



Memory

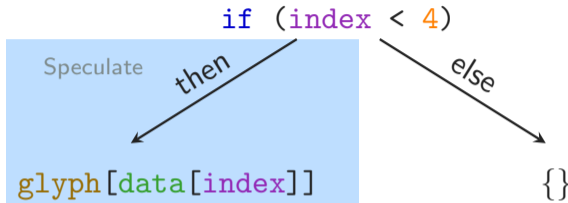
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	



`index = 3`

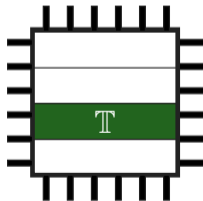
Shared Memory

	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



Memory

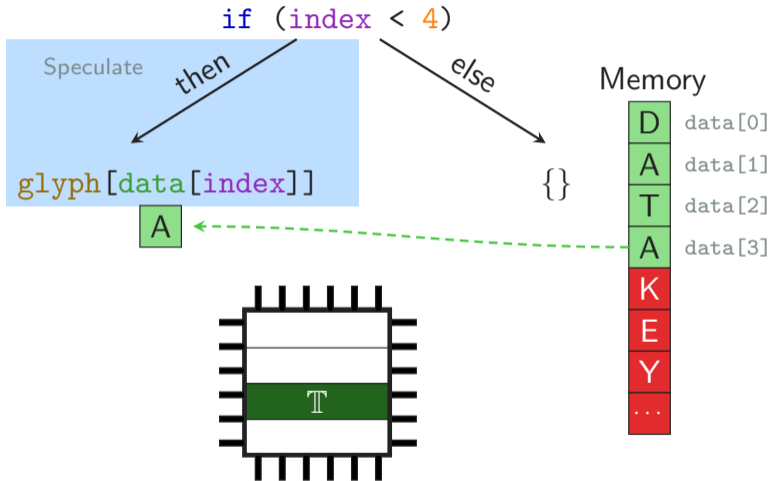
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

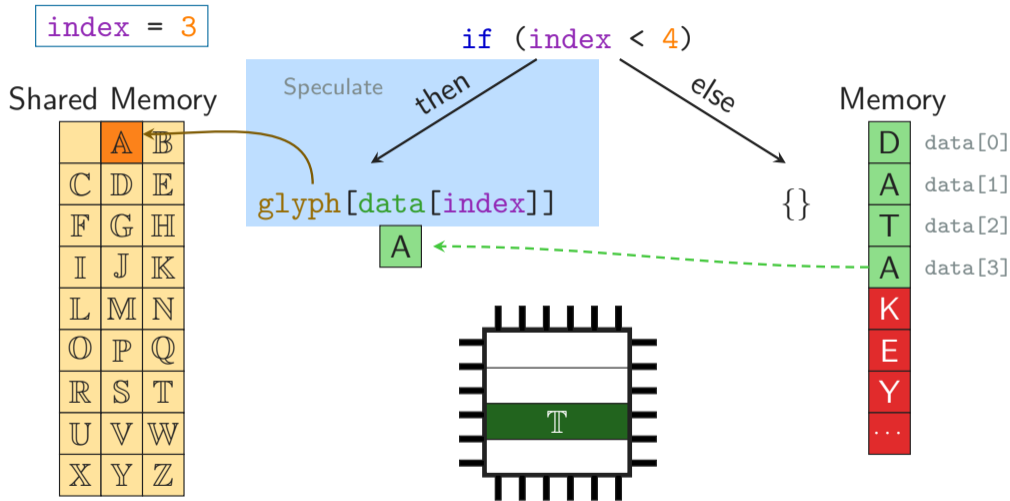


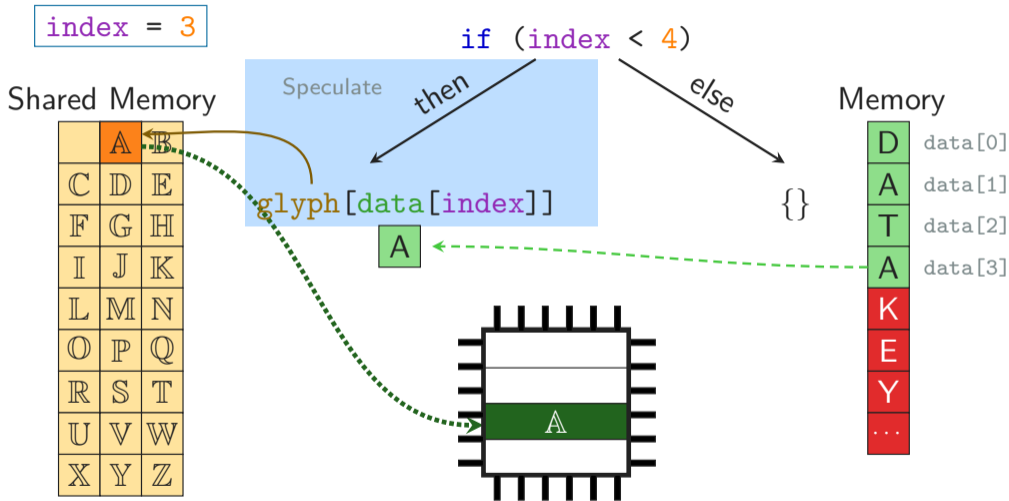
index = 3

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



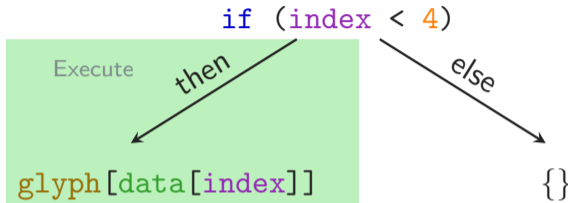




index = 3

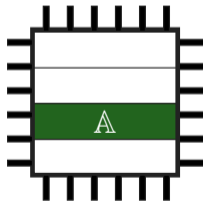
Shared Memory

	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



Memory

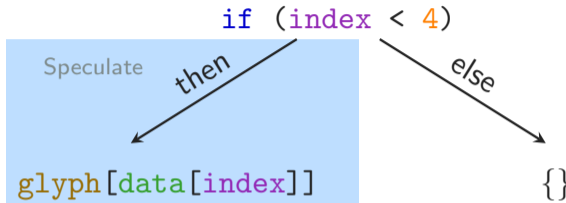
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	



`index = 4`

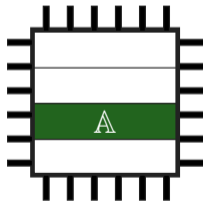
Shared Memory

	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



Memory

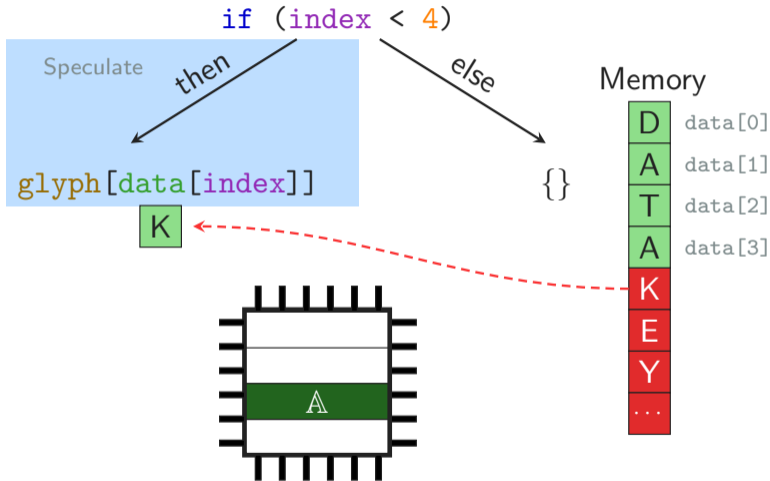
D	data[0]
A	data[1]
T	data[2]
A	data[3]
K	
E	
Y	
...	

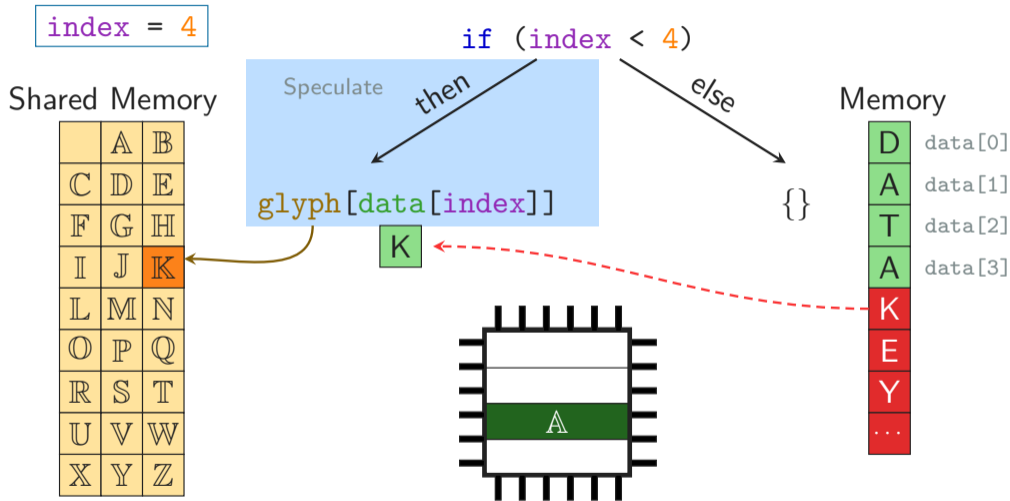


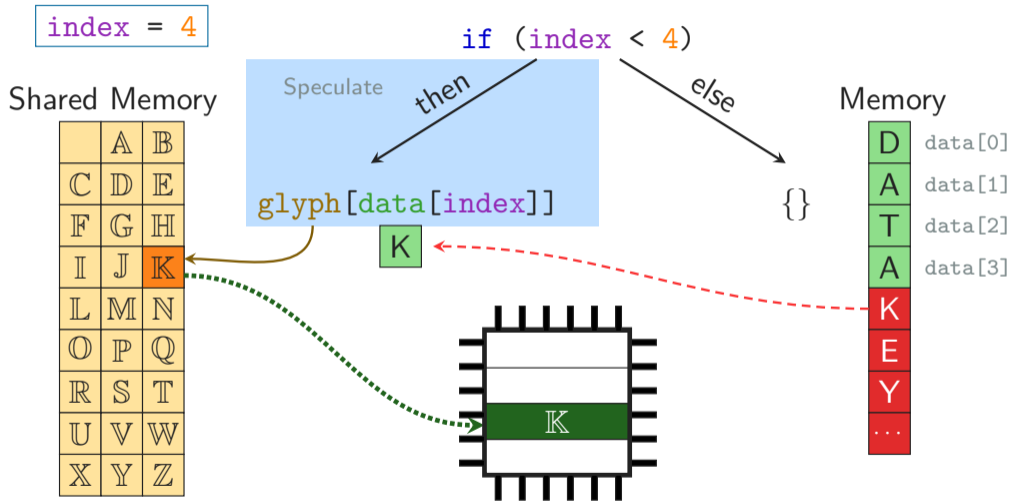
`index = 4`

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



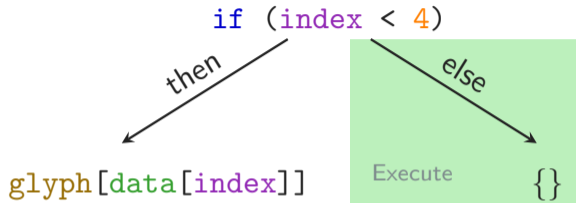




```
index = 4
```

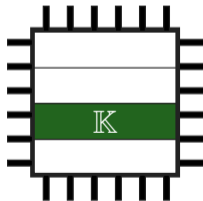
Shared Memory

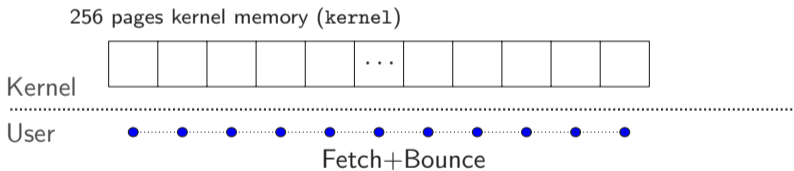
	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z



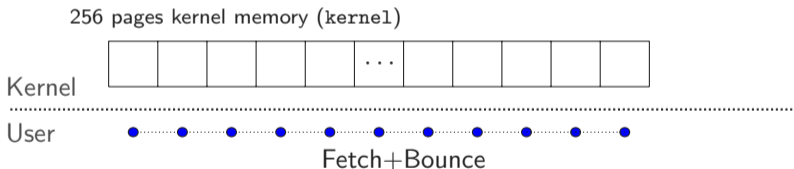
Memory

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



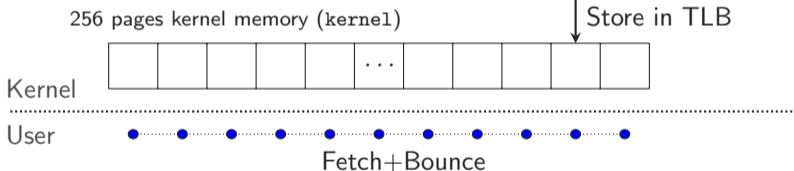


```
if (x < len(array))  
    y = kernel[array[x] * 4096]
```



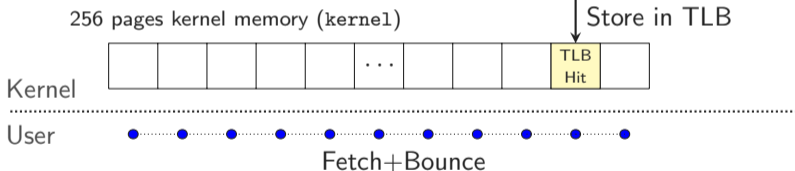
```
if (x < len(array))
```

```
    y = kernel[array[x] * 4096]
```



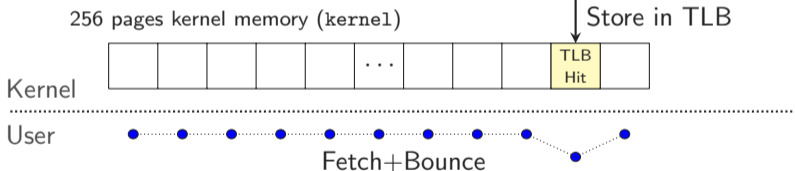
```
if (x < len(array))
```

```
    y = kernel[array[x] * 4096]
```



```
if (x < len(array))
```

```
    y = kernel[array[x] * 4096]
```





MELTDOWN

- Exploits deferred permission check in **out-of-order** execution



MELTDOWN

- Exploits deferred permission check in **out-of-order** execution
- Can read **any kernel address**



MELTDOWN

- Exploits deferred permission check in **out-of-order** execution
- Can read **any kernel address**
- Kernel maps all **physical memory**



MELTDOWN

- Exploits deferred permission check in **out-of-order** execution
 - Can read **any kernel address**
 - Kernel maps all **physical memory**
- Read arbitrary memory by **encoding in cache**



Assumptions

1. Stalling CPU might be **too costly**



Assumptions

1. Stalling CPU might be **too costly**
2. Stalling might require **redesigning** parts of CPU pipeline



Assumptions

1. Stalling CPU might be **too costly**
2. Stalling might require **redesigning** parts of CPU pipeline

Hypothesis

Load is executed, returned value is **zeroed out** on faults



Two tests:

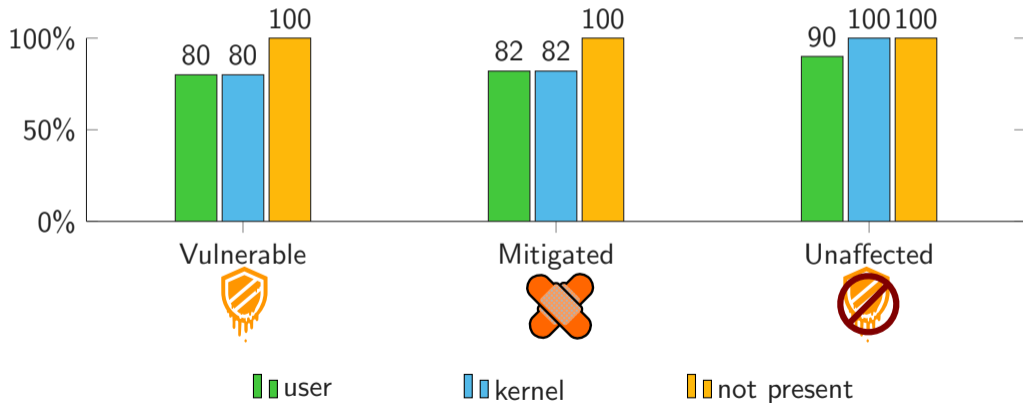
1. Perform **Meltdown attack**



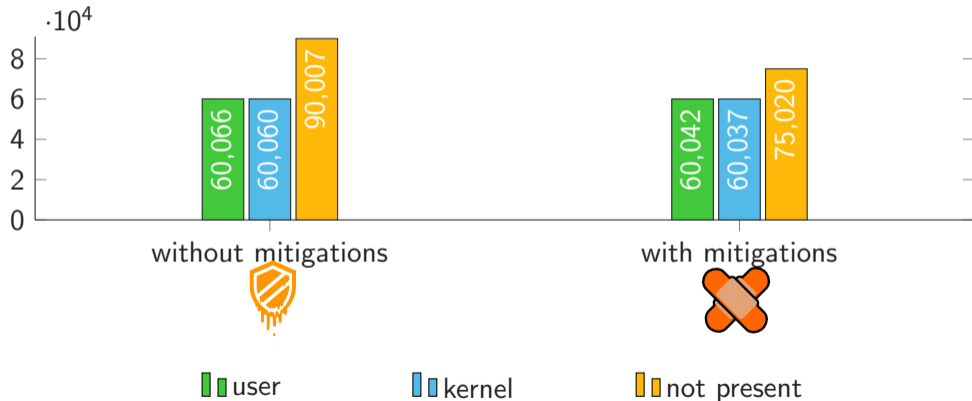
Two tests:

1. Perform **Meltdown attack**
2. Use **Performance Counters**

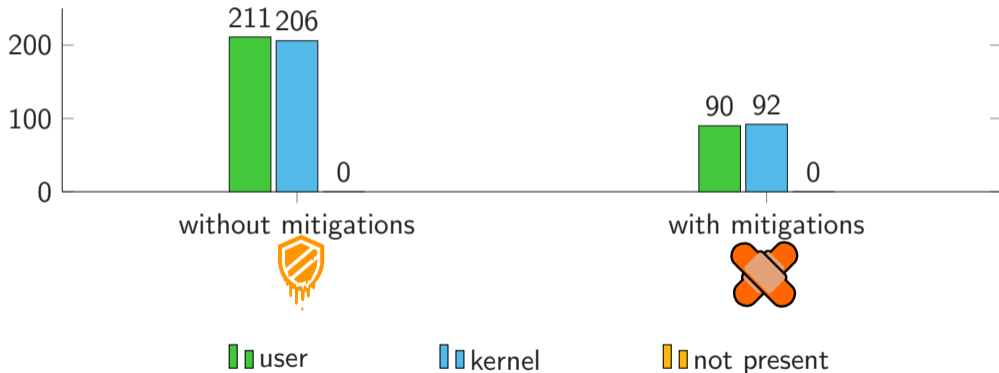
- Intel: `CYCLE_ACTIVITY.STALLS_MEM_ANY`
- AMD: Dispatch Stalls

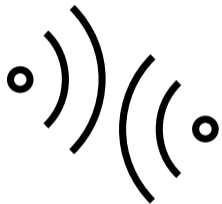


- Track number of **issued μ OPs** on the load ports
- `UOPS_DISPATCHED_PORT.PORT_2`, `UOPS_DISPATCHED_PORT.PORT_3`



- L1D_PEND_MISS.PENDING_CYCLES





- Can we use the hardware-based mitigations for an attack?

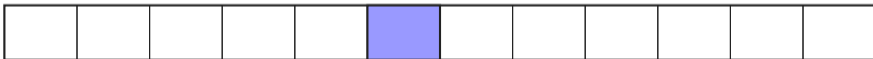


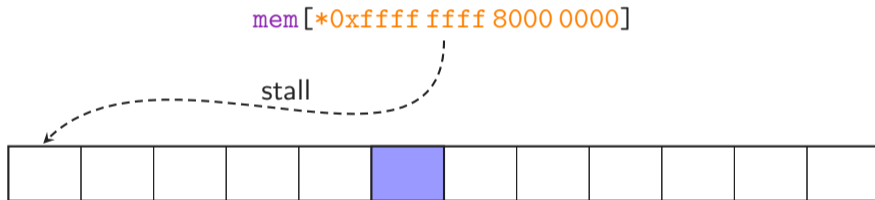
- Can we use the hardware-based mitigations for an attack?
- EchoLoad: fast and reliable **KASLR break**

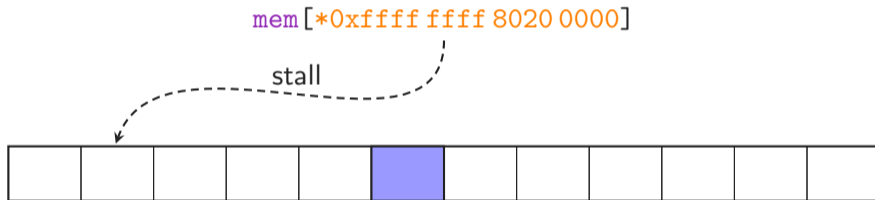


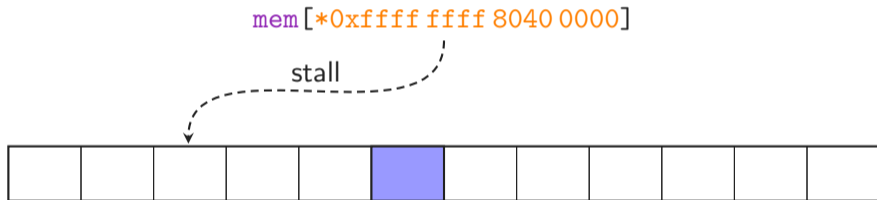
- Can we use the hardware-based mitigations for an attack?
- EchoLoad: fast and reliable **KASLR break**
- **Encodes** the returned value in the **cache**

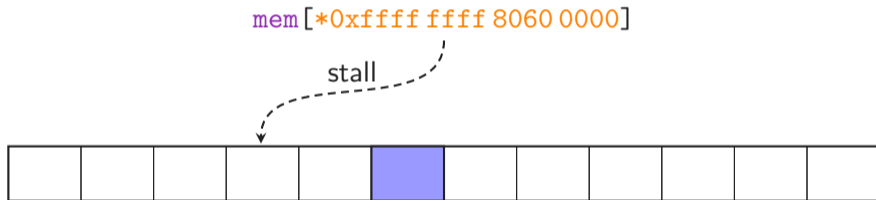
```
mem[*0xffff ffff 8000 0000]
```





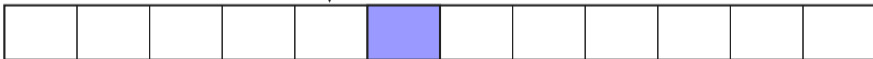




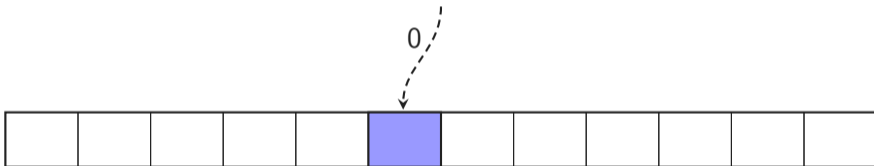


```
mem[*0xffff ffff 8080 0000]
```

stall



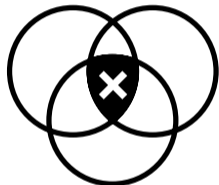
```
mem[*0xffff ffff 80a0 0000]
```



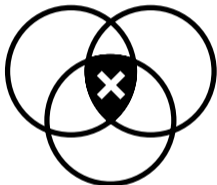
CPU		Speculation	TSX	Segfault
i7-6700K	Time (F-Score)	63 μ s (0.999)	48 μ s (1.000)	133 μ s (1.000)
i9-9900K	Time (F-Score)	33 μ s (1.000)	29 μ s (1.000)	86 μ s (1.000)
Xeon Silver 4208	Time (F-Score)	51 μ s (0.994)	40 μ s (1.000)	127 μ s (1.000)

CPU		Speculation	TSX	Segfault
i7-6700K	Time (F-Score)	63 μ s (0.999)	48 μ s (1.000)	133 μ s (1.000)
i9-9900K	Time (F-Score)	33 μ s (1.000)	29 μ s (1.000)	86 μ s (1.000)
Xeon Silver 4208	Time (F-Score)	51 μ s (0.994)	40 μ s (1.000)	127 μ s (1.000)

- Works in **SGX** and **JavaScript**

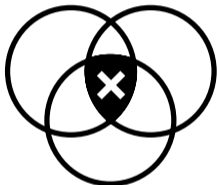


Timing difference



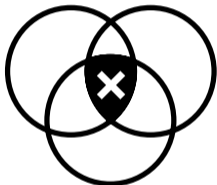
Timing difference

- between **mapped** and **unmapped pages**



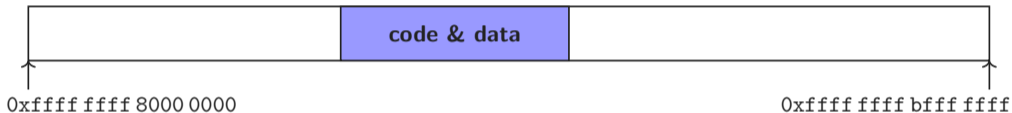
Timing difference

- between **mapped** and **unmapped pages**
- for **different page sizes**

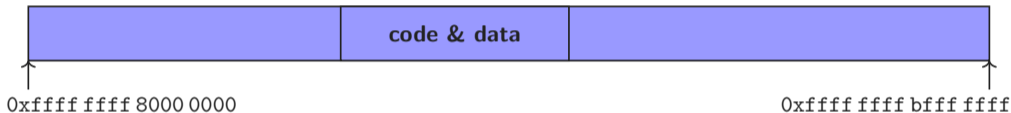


Timing difference

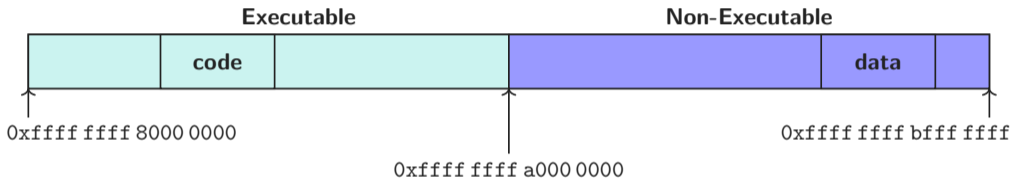
- between **mapped** and **unmapped pages**
- for **different page sizes**
- between **executable** and **non-executable pages**



Current Linux design

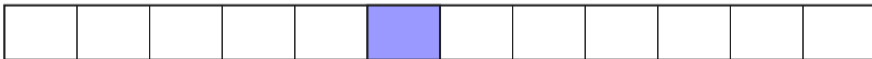


Step 1: Mitigating difference between **mapped and unmapped pages**

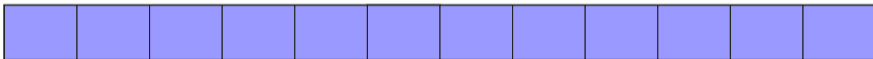


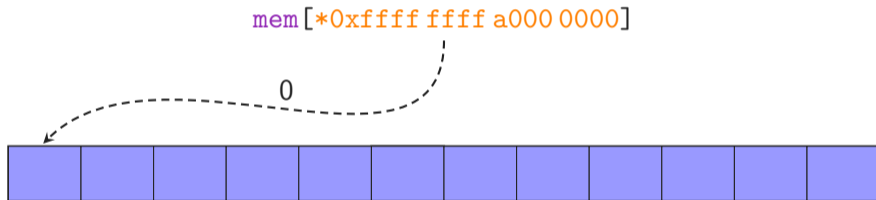
Step 2: Mitigating difference between **executable** and **non-executable** pages

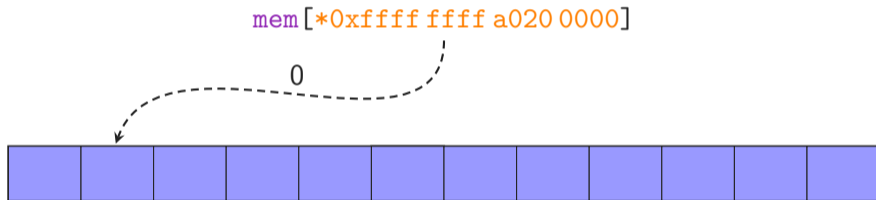
```
mem[*0xffff ffff a000 0000]
```

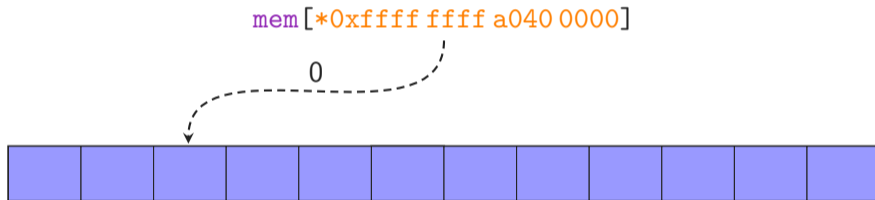


```
mem[*0xffff ffff a000 0000]
```



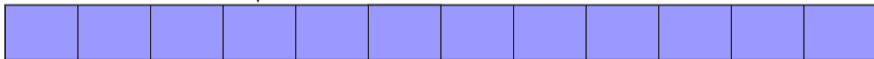




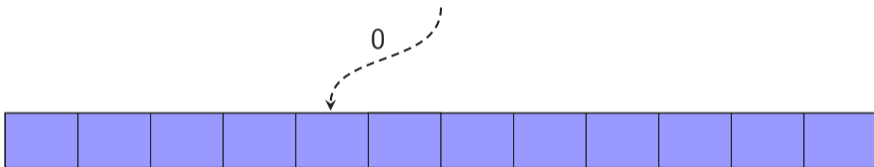


```
mem[*0xffff ffff a060 0000]
```

0

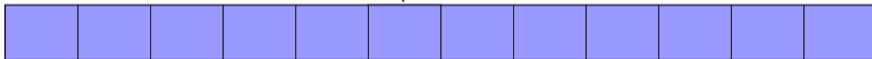
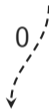


```
mem[*0xffff ffff a080 0000]
```

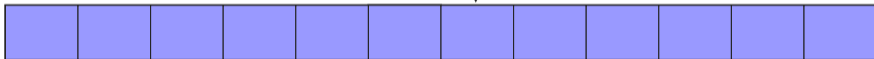
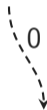


```
mem[*0xffff ffff a0a0 0000]
```

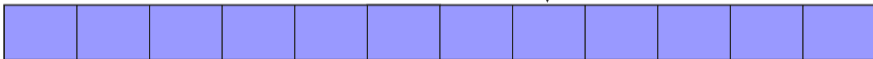
0



```
mem[*0xffff ffff a0c0 0000]
```

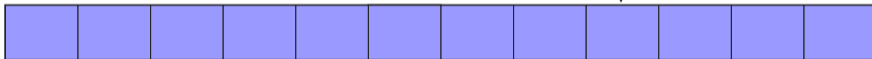


```
mem[*0xffff ffff a0e0 0000]
```



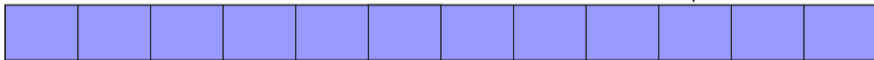
```
mem[*0xffff ffff a100 0000]
```

0



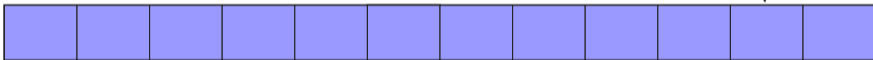
```
mem[*0xffff ffff a120 0000]
```

0

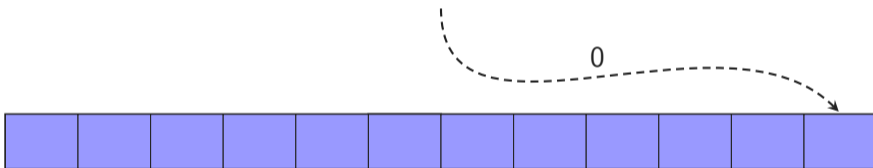


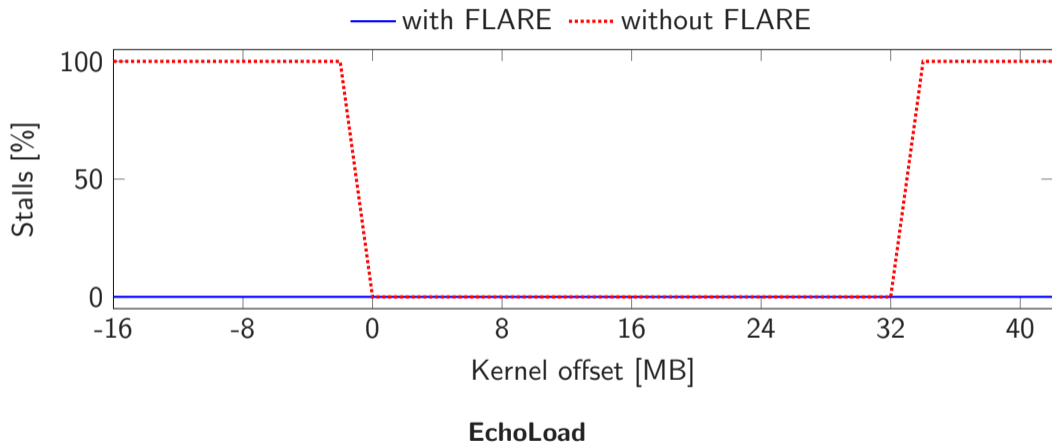

```
mem[*0xffff ffff a140 0000]
```

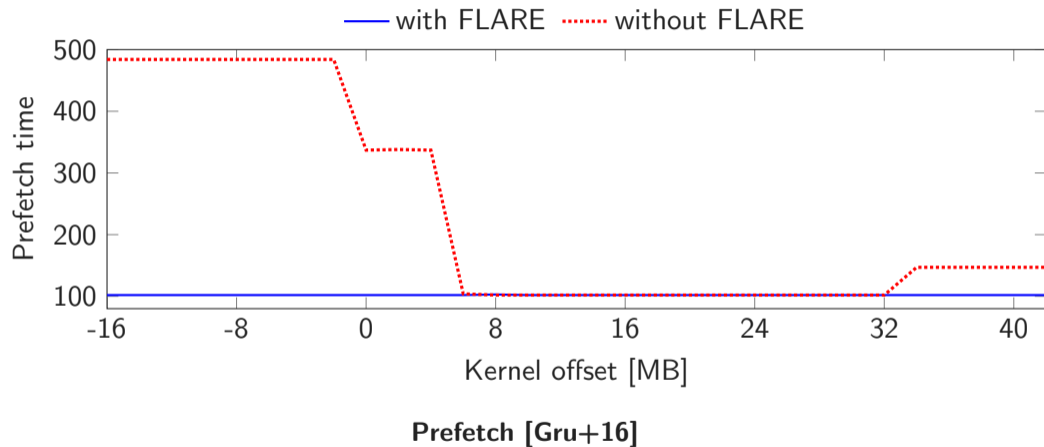
0

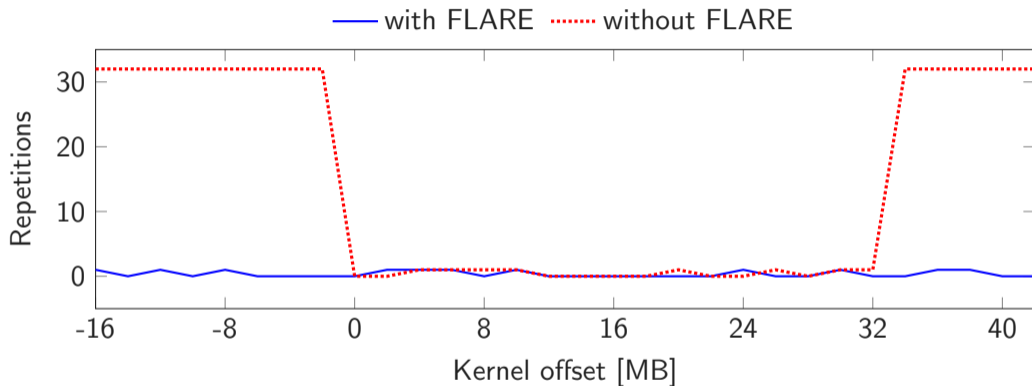


```
mem[*0xffff ffff a160 0000]
```

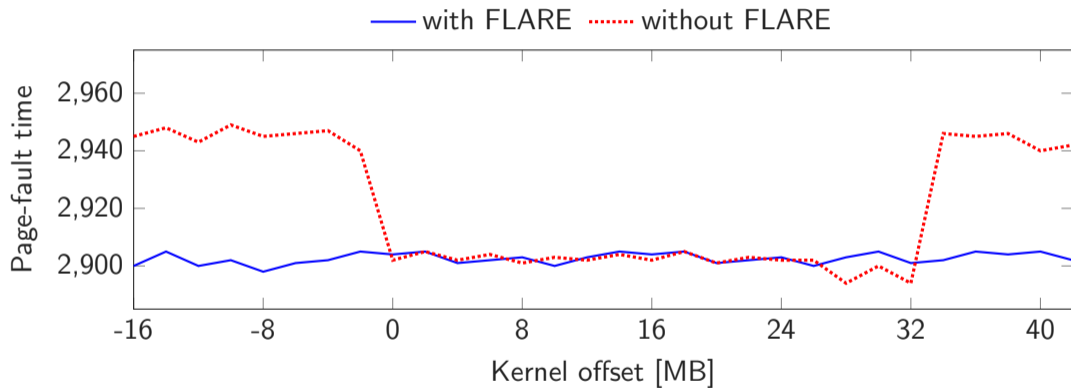




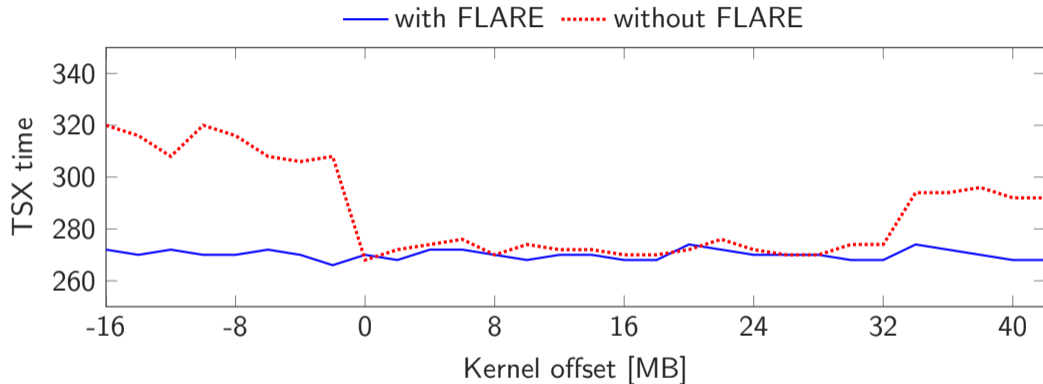




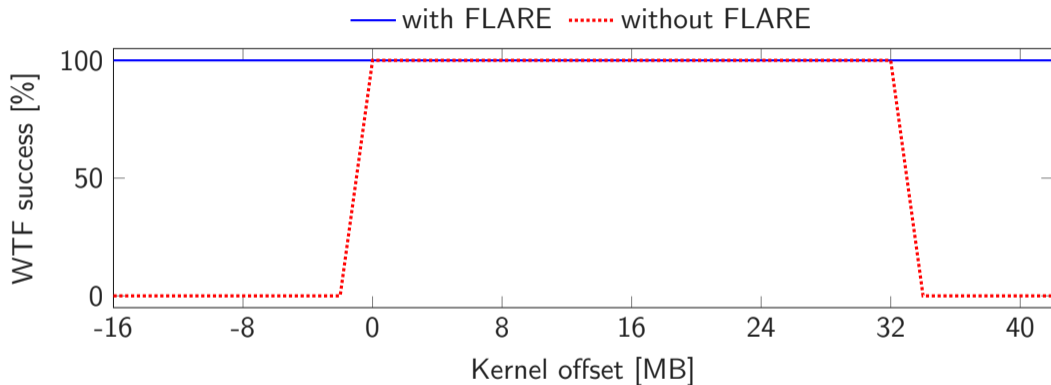
Data Bounce [Sch+19]



Double page fault [HWH13]



DrK [JLK16]



Fallout [Can+19]



You can find our **proof-of-concept** implementation of FLARE on:

- <https://github.com/IAIK/FLARE>



- **Optimizations** introduce security problems



- **Optimizations** introduce security problems
- **Mitigations** can overlook edge-cases



- **Optimizations** introduce security problems
- **Mitigations** can overlook edge-cases
- Once again requires **software workaround**

SCORE-to-Leak Forwarding

There and Back Again






Claudio Canella (@cc0x1f), Lukas Giner (@redrabbyte), Michael Schwarz (@misc0110)

October 2, 2020

Graz University of Technology

CISPA Helmholtz Center for Information Security

References

-  C. Canella, D. Genkin, L. Giner, D. Gruss, M. Lipp, M. Minkin, D. Moghimi, F. Piessens, M. Schwarz, B. Sunar, J. Van Bulck, and Y. Yarom. Fallout: Leaking Data on Meltdown-resistant CPUs. In: CCS. 2019.
-  D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard. Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR. In: CCS. 2016.
-  R. Hund, C. Willems, and T. Holz. Practical Timing Side Channel Attacks against Kernel Space ASLR. In: S&P. 2013.
-  Y. Jang, S. Lee, and T. Kim. Breaking Kernel Address Space Layout Randomization with Intel TSX. In: CCS. 2016.
-  M. Schwarz, C. Canella, L. Giner, and D. Gruss. Store-to-Leak Forwarding: Leaking Data on Meltdown-resistant CPUs. In: arXiv:1905.05725 (2019).