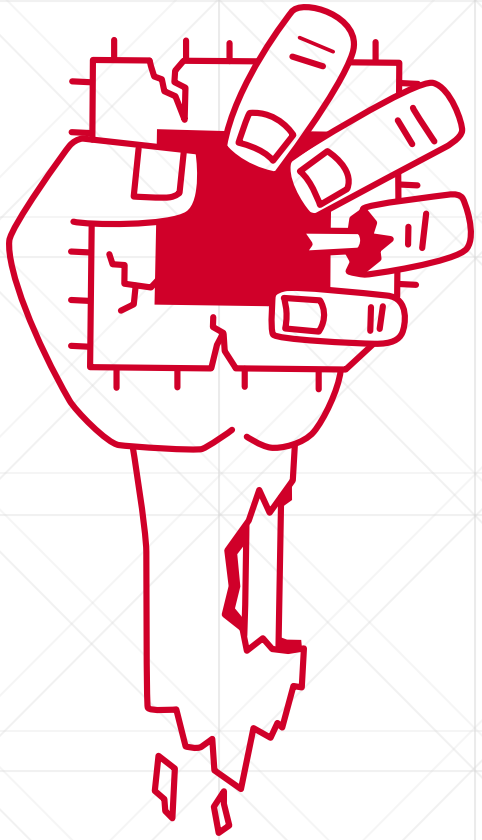
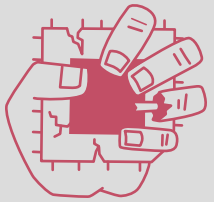


ZombieLoad

Cross-Privilege-Boundary Data Sampling

Michael Schwarz, Moritz Lipp, **Daniel Moghimi**, Jo Van Bulck, Julian Stecklina,
Thomas Prescher, Daniel Gruss

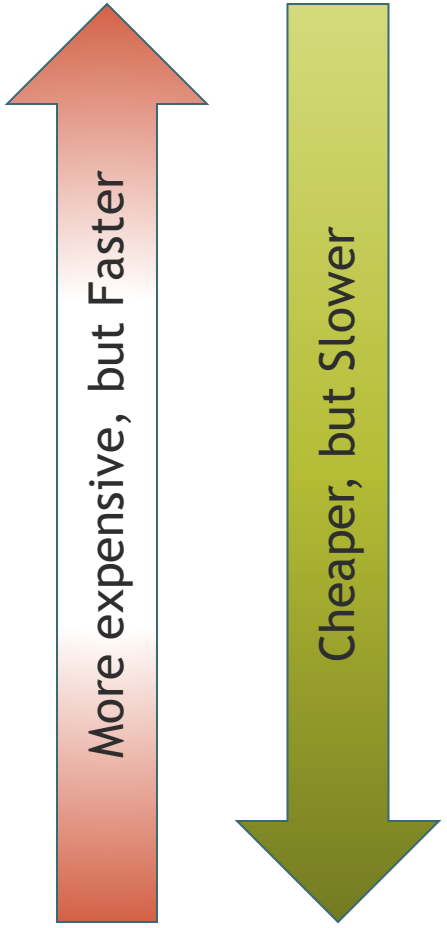
whoami

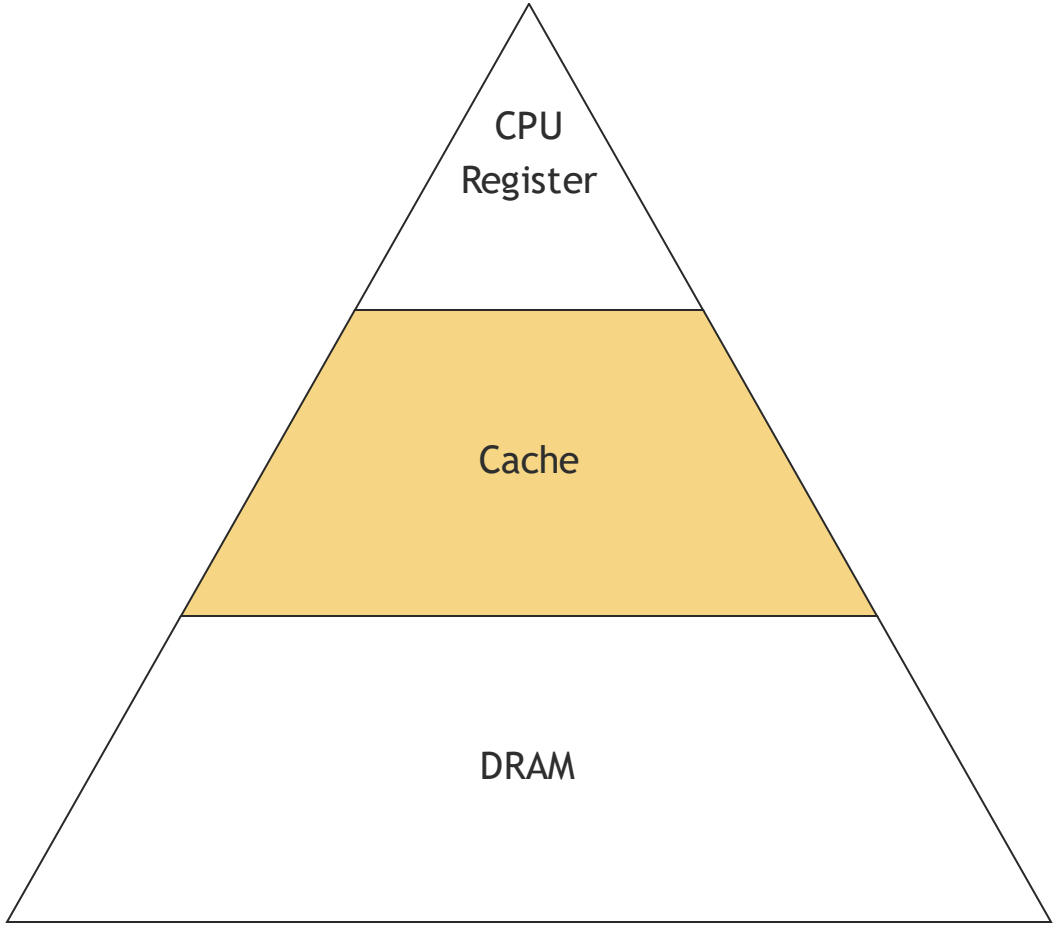


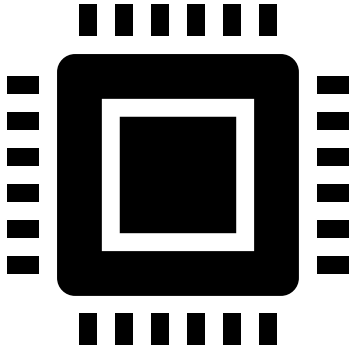
- Daniel Moghimi (@danielmgmi)
- Computer Security *since 2010*
  - Reverse Engineering
  - Binary Analysis
  - Application Security
- PhD Student *since 2017*
  - Microarchitectural Security
  - Side Channels
  - Breaking Cryptographic Implementations


$5,000

Background: Cache Attacks – Cache Memory



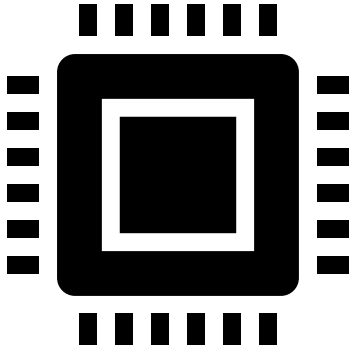




Background: Cache Attacks – Cache Miss

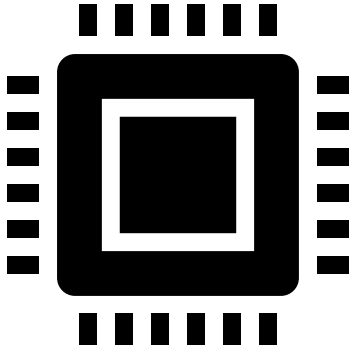
10100110

Cache



Background: Cache Attacks

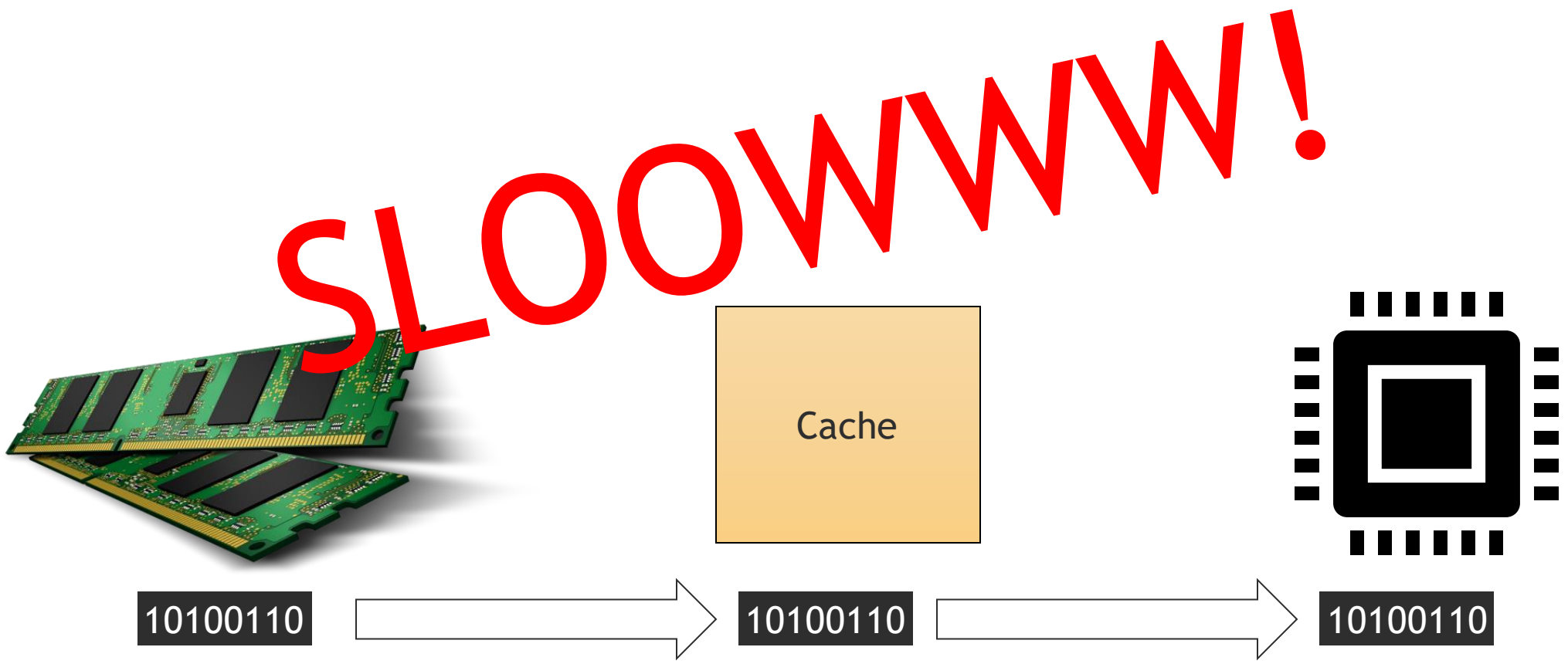
Cache



10100110

10100110

Background: Cache Attacks

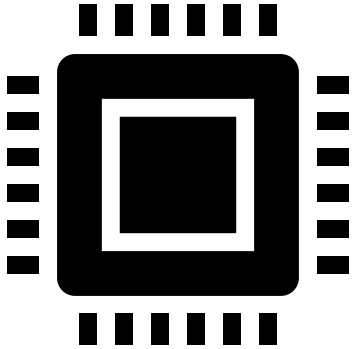


Background: Cache Attacks – Cache Hit

10100110

Cache

10100110

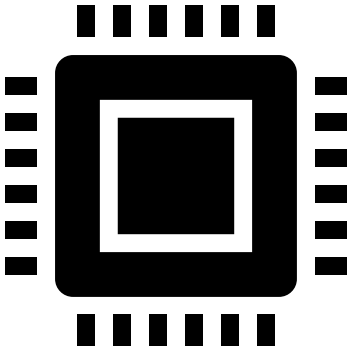


Background: Cache Attacks – Cache Hit

Yeah! Fast!!

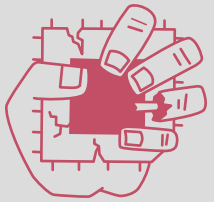
10100110

10100110



10100110

Background: Cache Attacks – Flush & Reload (Yarom et al.)



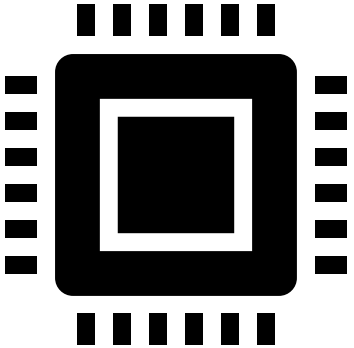
Attacker

Victim

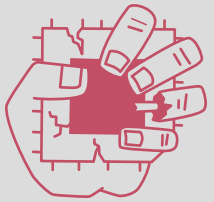
10100110

Cache

10100110



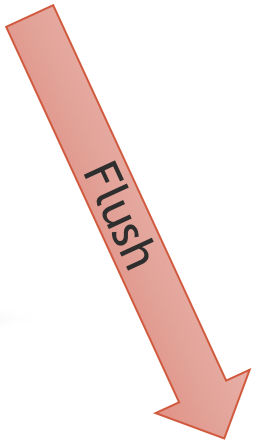
Background: Cache Attacks – Flush & Reload (Yarom et al.)

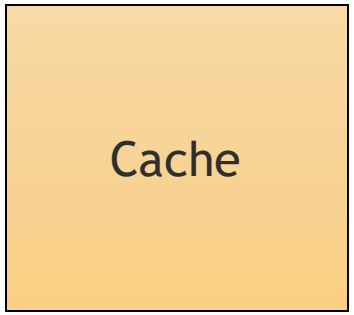


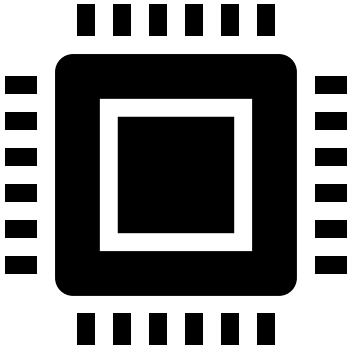
Attacker

Victim

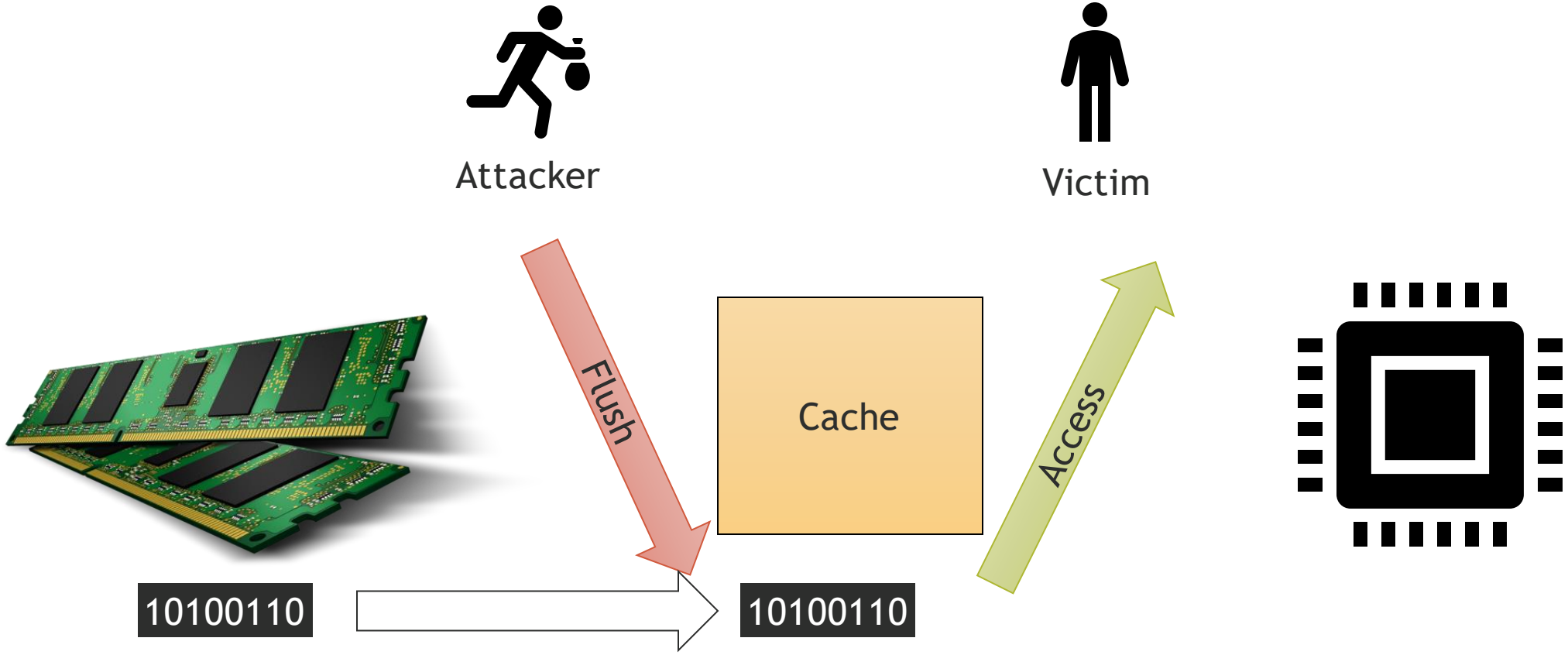
10100110





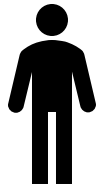


Background: Cache Attacks – Flush & Reload (Yarom et al.)



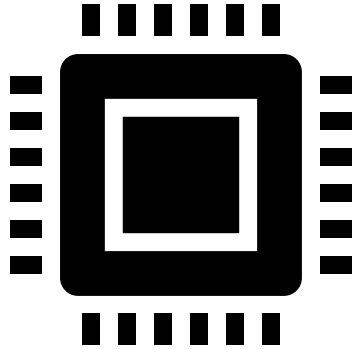
Background: Cache Attacks – Flush & Reload (Yarom et al.)

Attacker



Victim

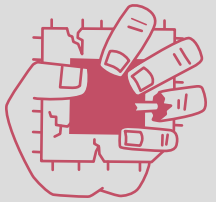
Cache



10100110

10100110

Background: Cache Attacks – Flush & Reload (Yarom et al.)

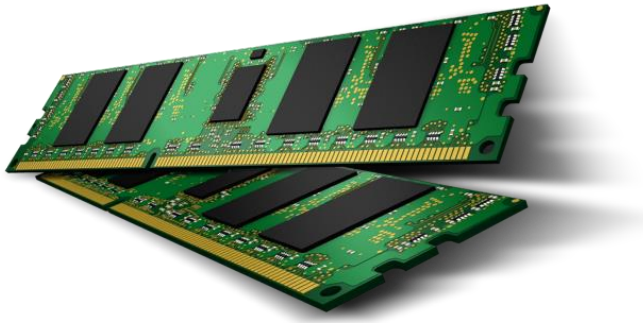


delta > threshold = cache hit

delta < threshold = cache miss

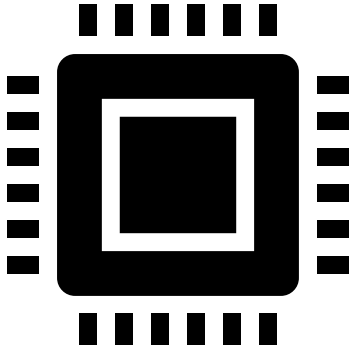
Attacker

Victim



Access

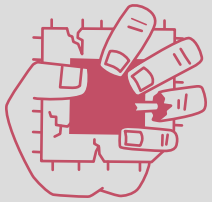
Cache



10100110

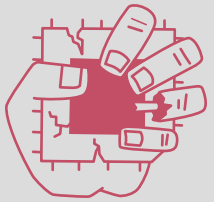
10100110

2018: Meltdown Attack?



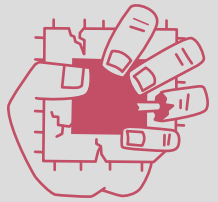
```
char secret = *(char *) 0xffffffff81a0123;
```

2018: Meltdown Attack?

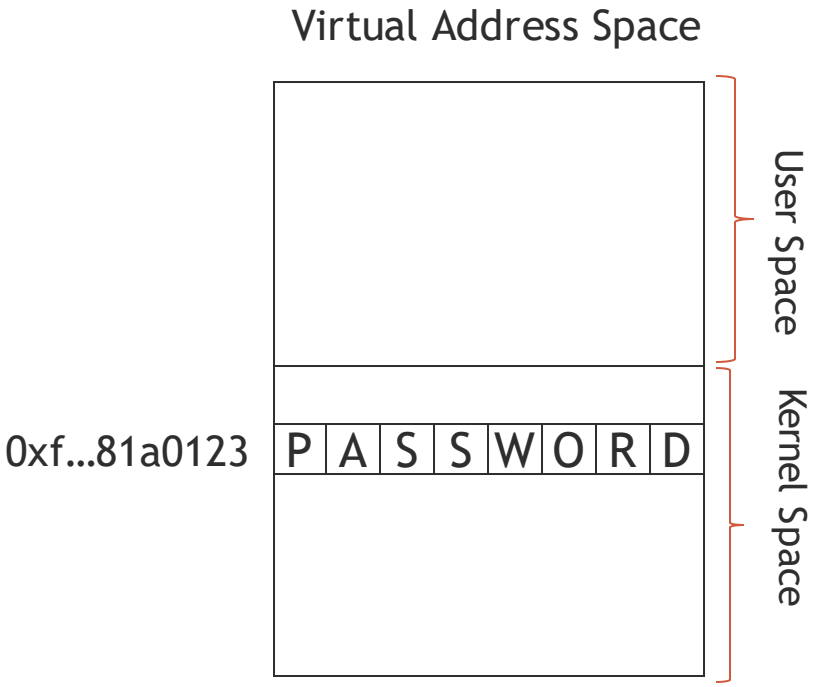


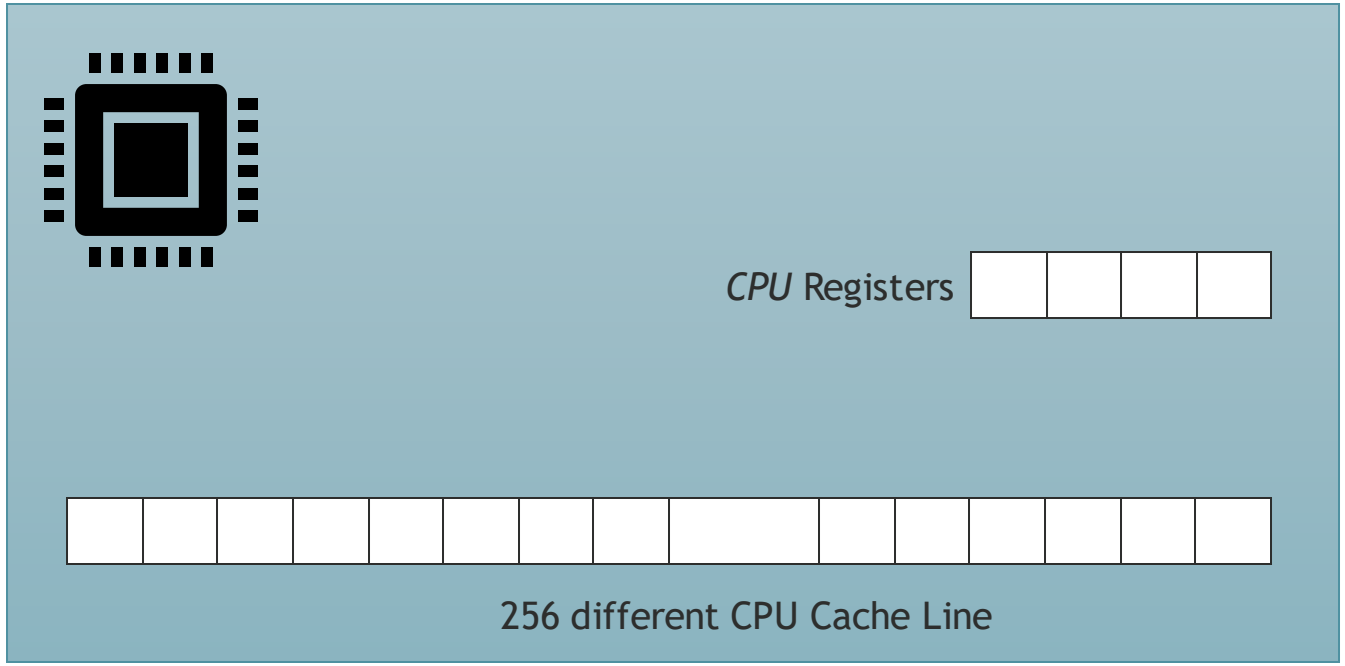
```
char secret = *(char *) 0xffffffff81a0123;
printf("%c\n", secret);
```

2018: Meltdown Attack?

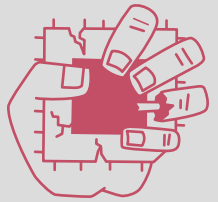


```
char secret = *(char *) 0xffffffff81a0123;
```

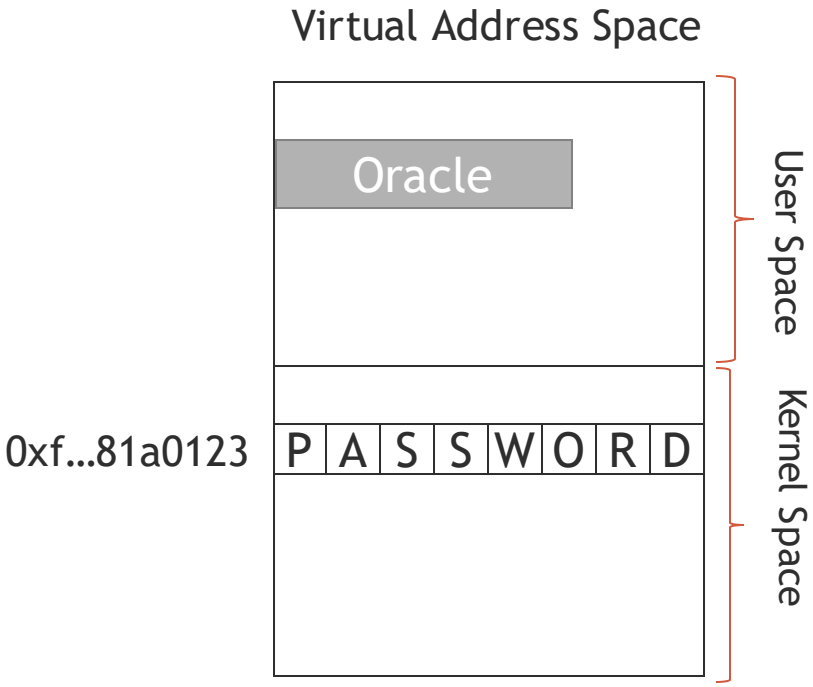


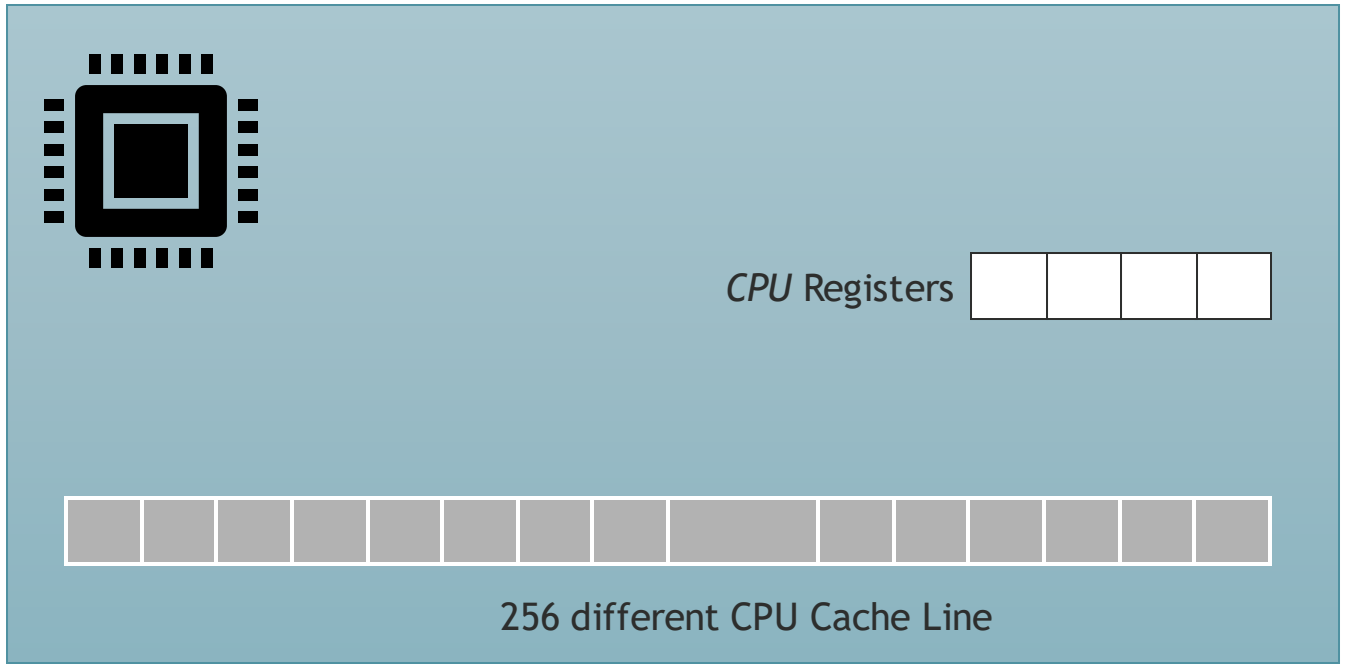


2018: Meltdown Attack?

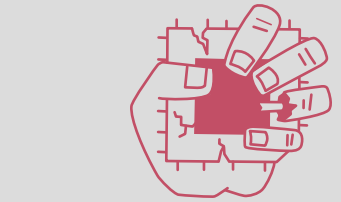


```
char secret = *(char *) 0xffffffff81a0123;
```

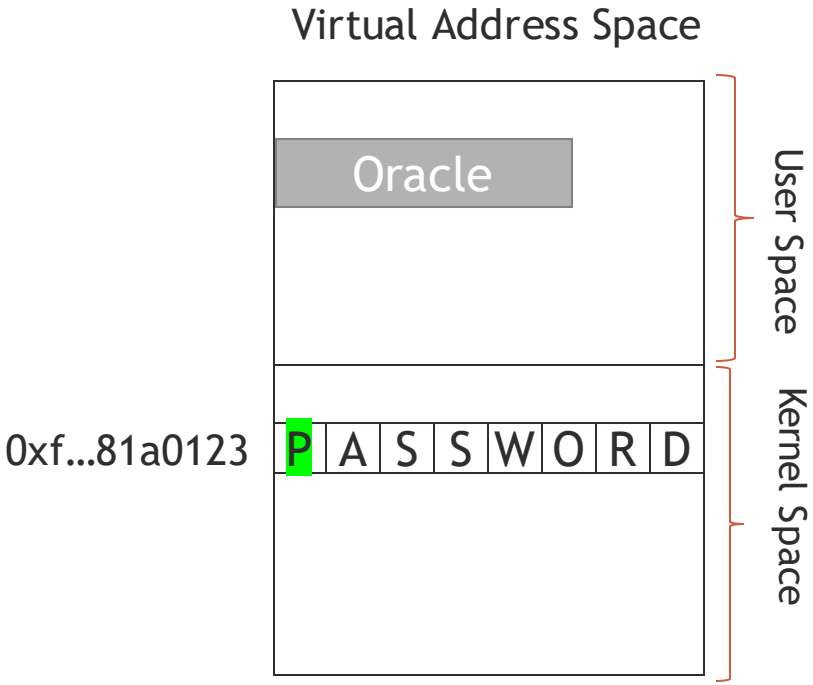


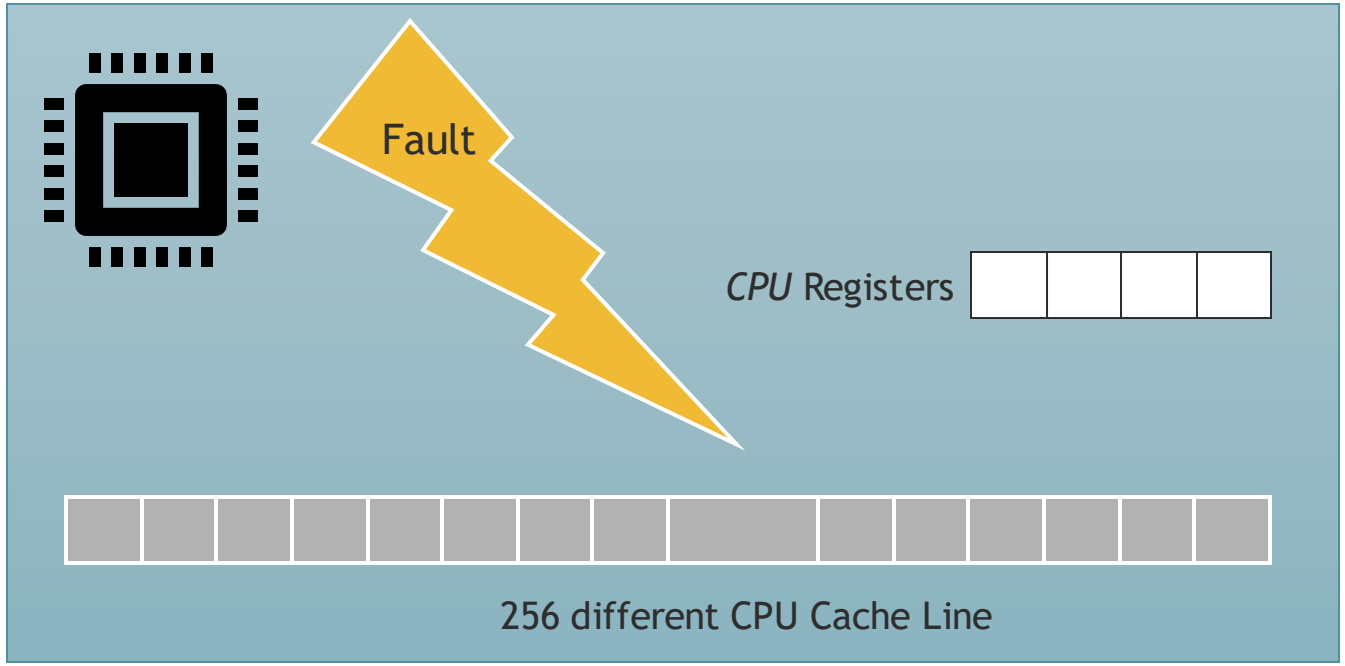


2018: Meltdown Attack?

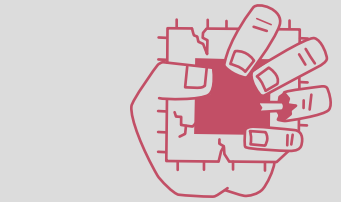


```
char secret = *(char *) 0xffffffff81a0123;
```

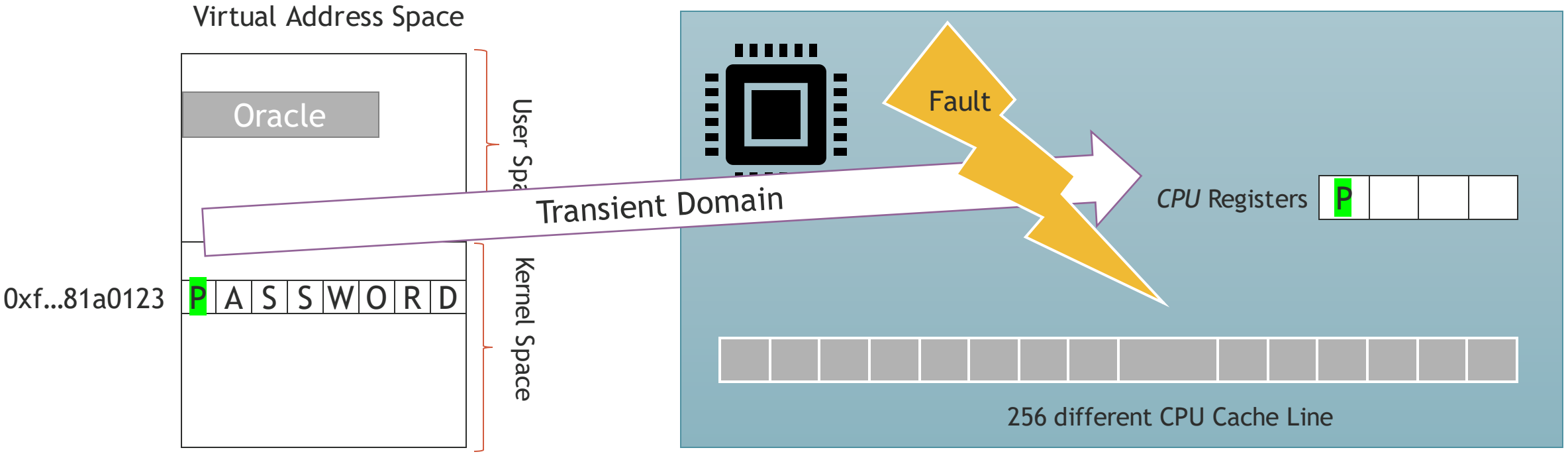




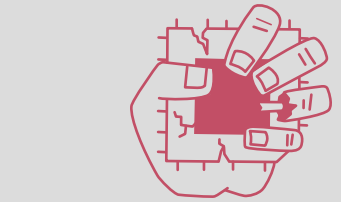
2018: Meltdown Attack?



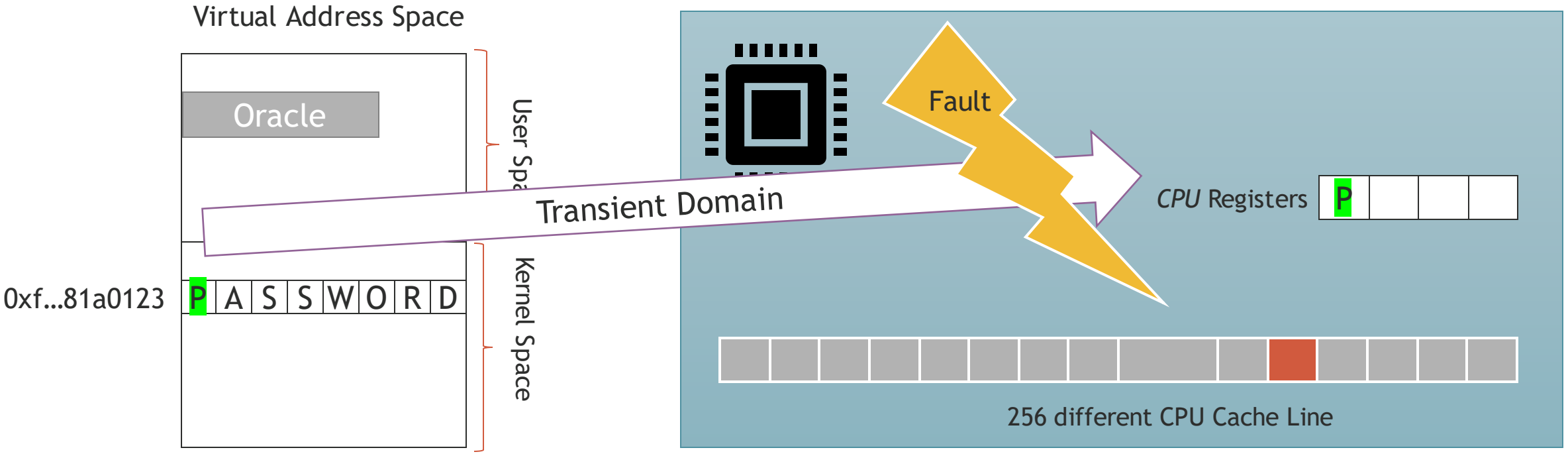
```
char secret = *(char *) 0xffffffff81a0123;
```



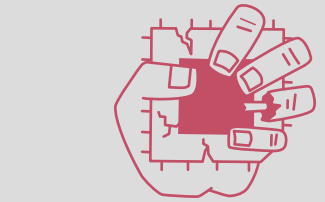
2018: Meltdown Attack?



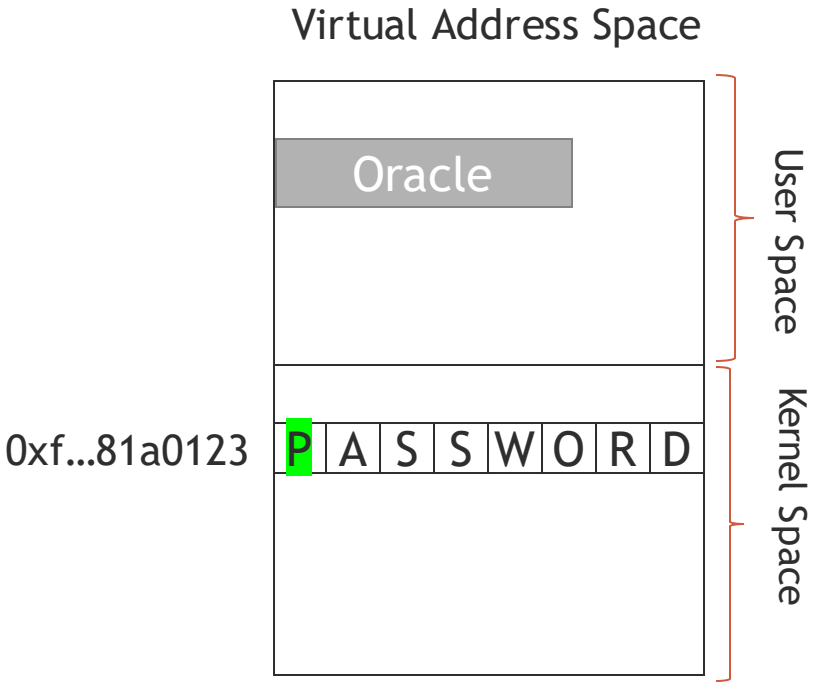
```
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```

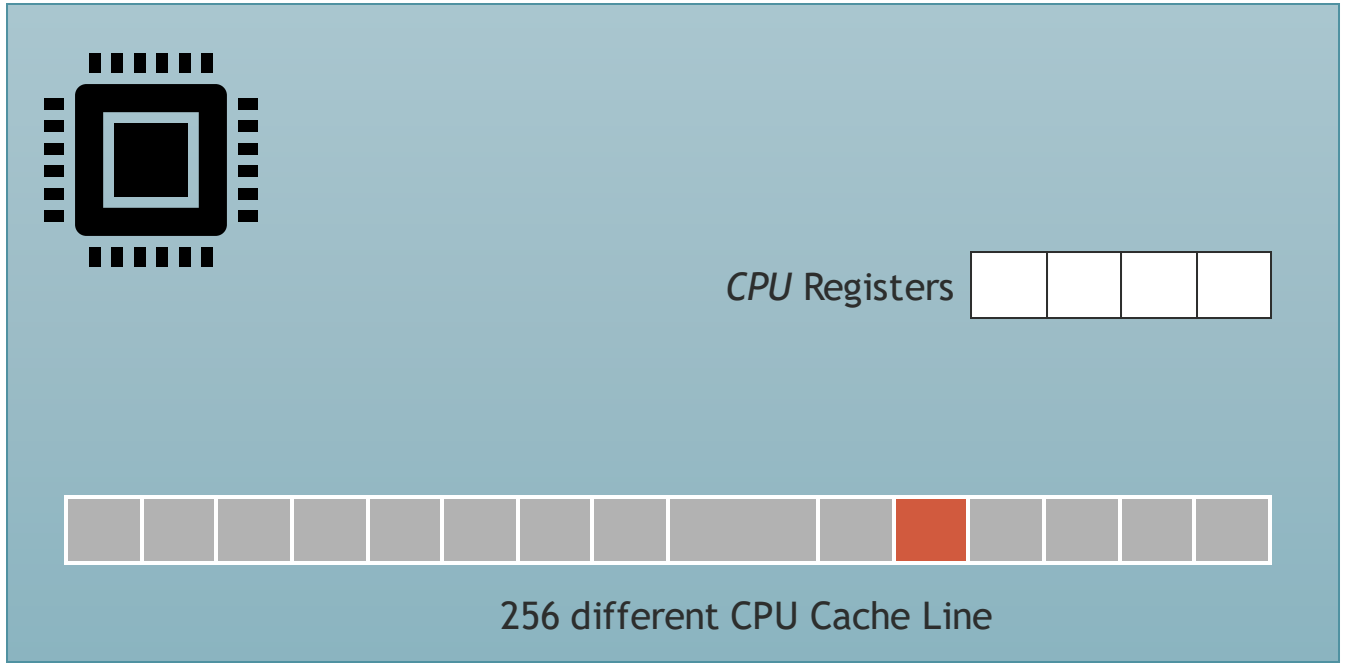


2018: Meltdown Attack?

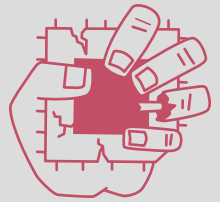


```
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```

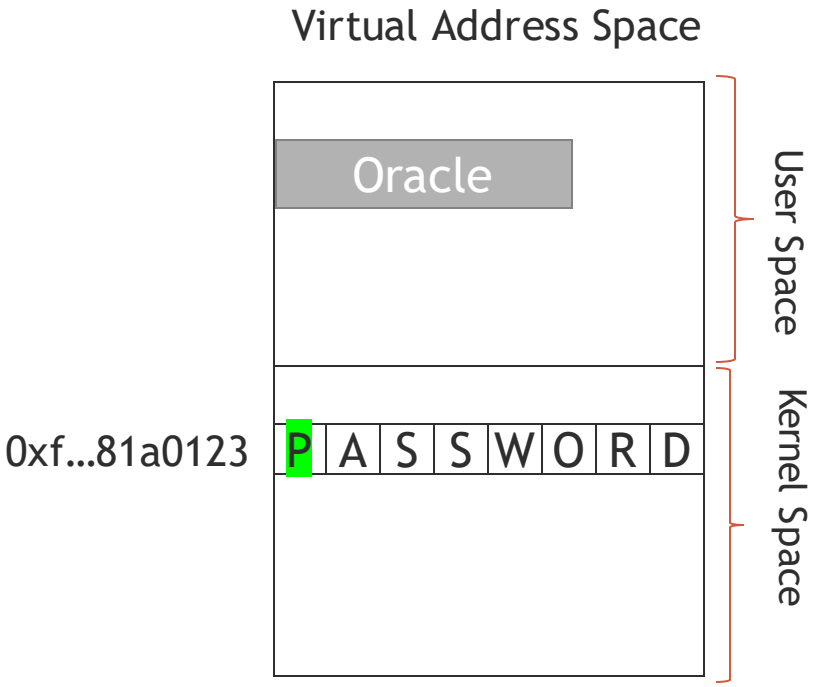


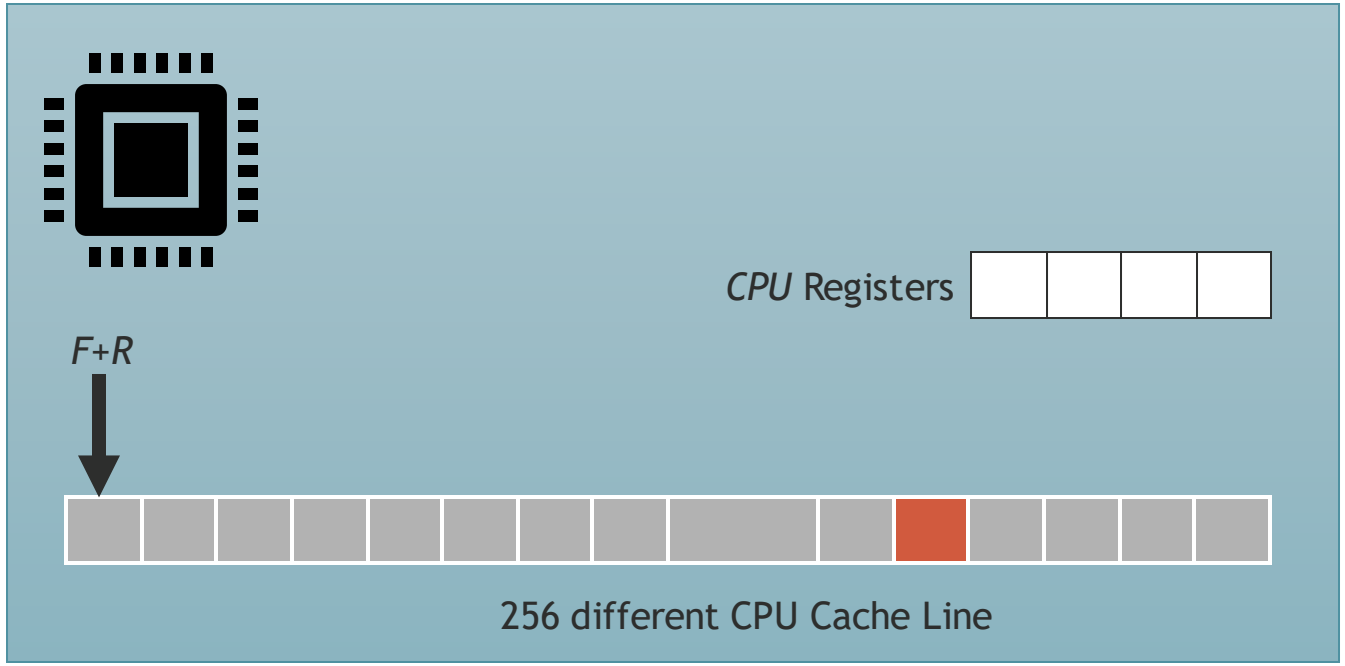


2018: Meltdown Attack?

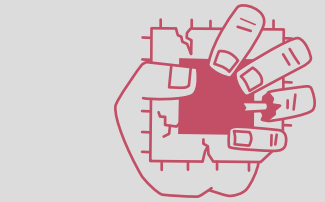


```
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```

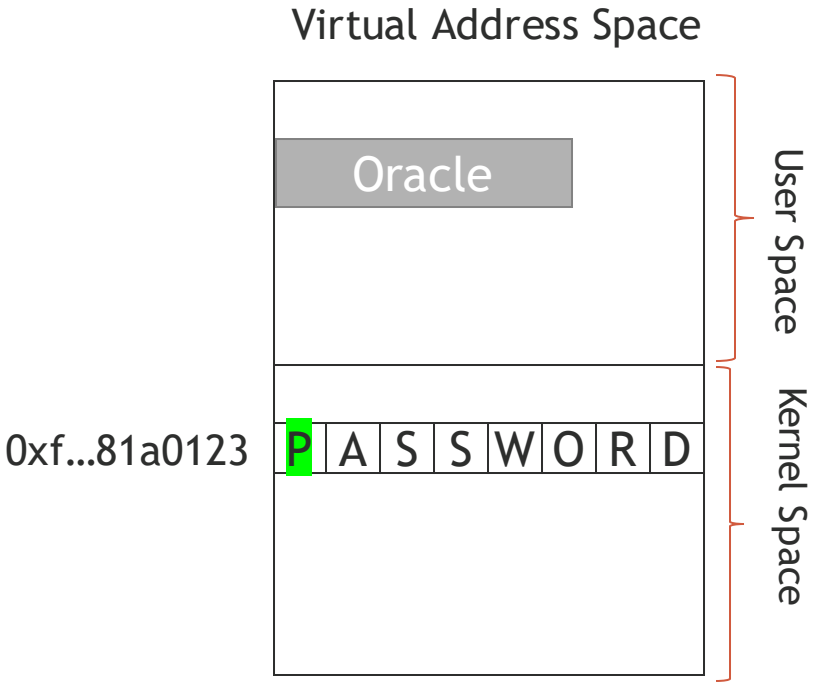


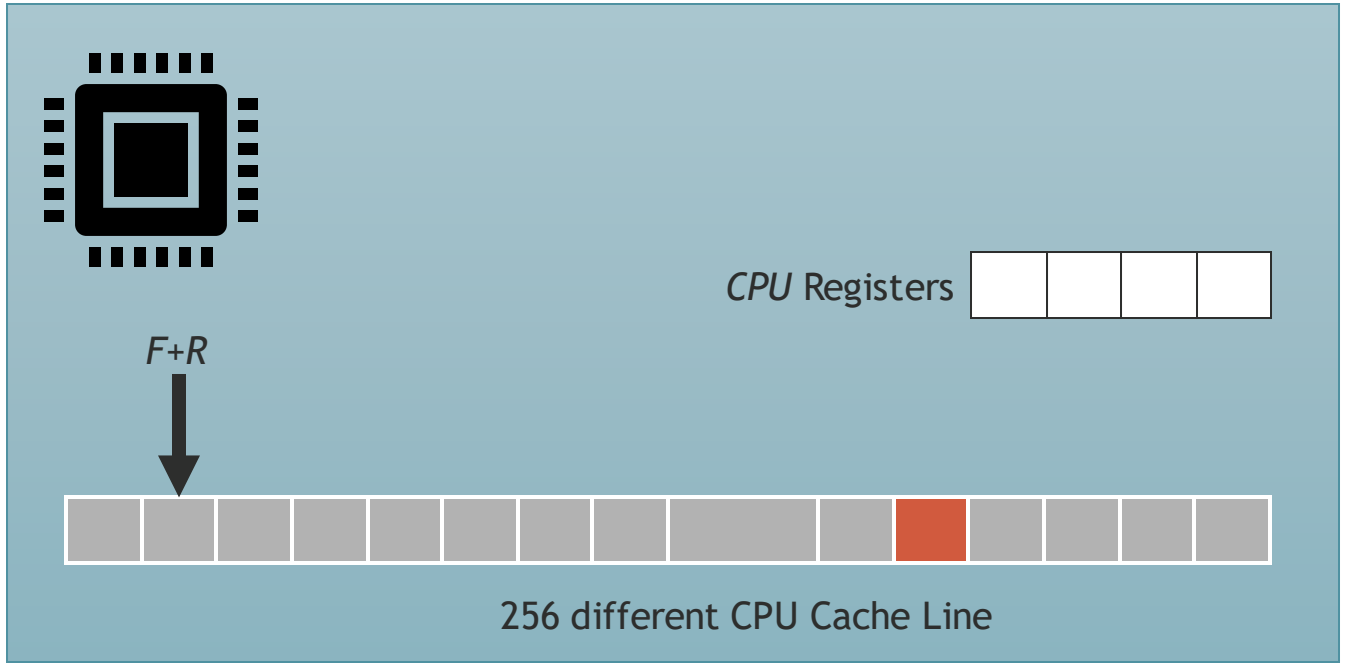


2018: Meltdown Attack?

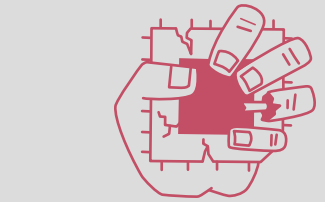


```
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```

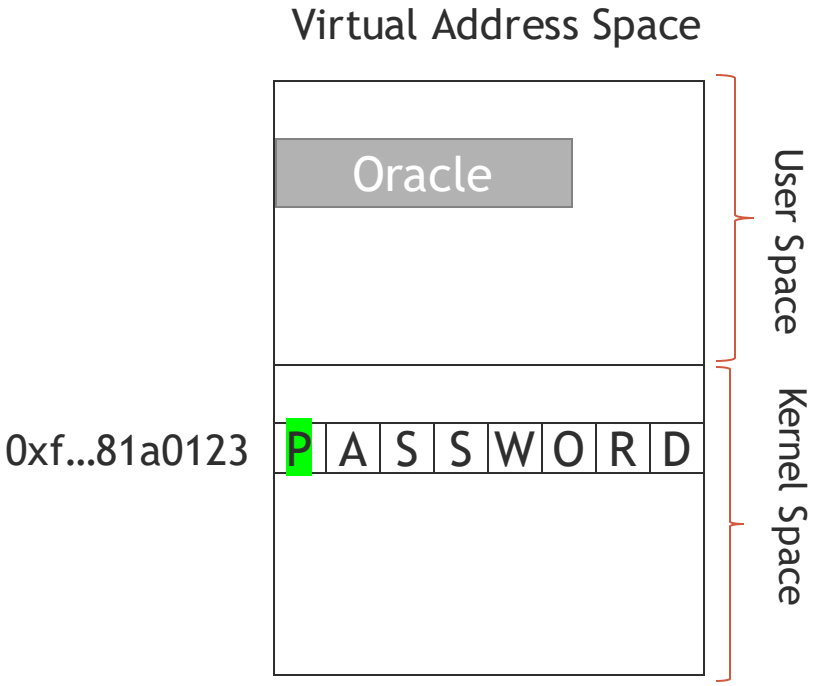


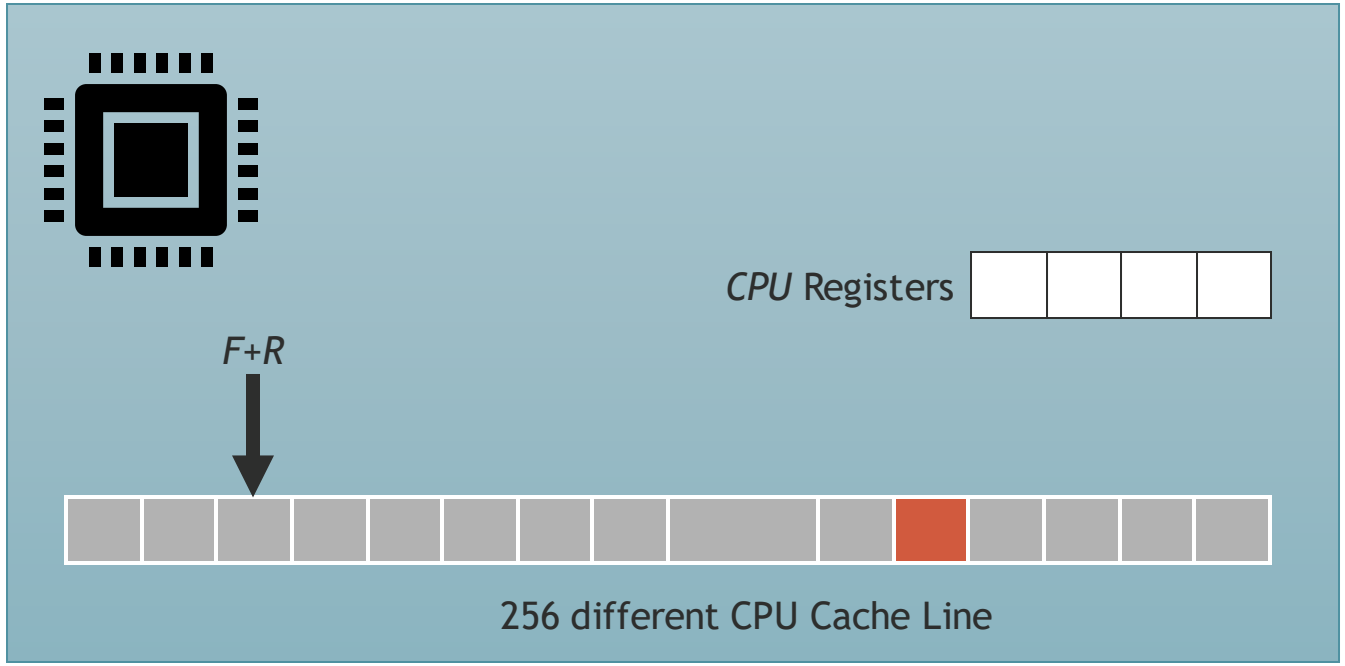


2018: Meltdown Attack?

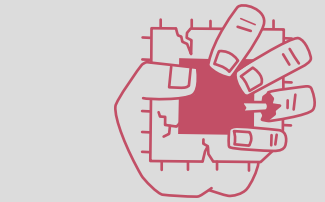


```
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```

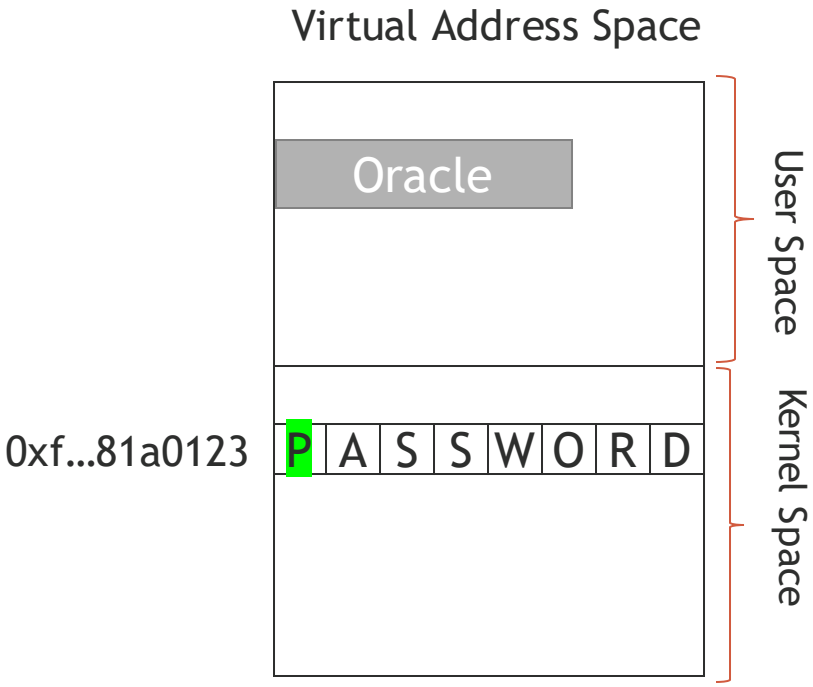


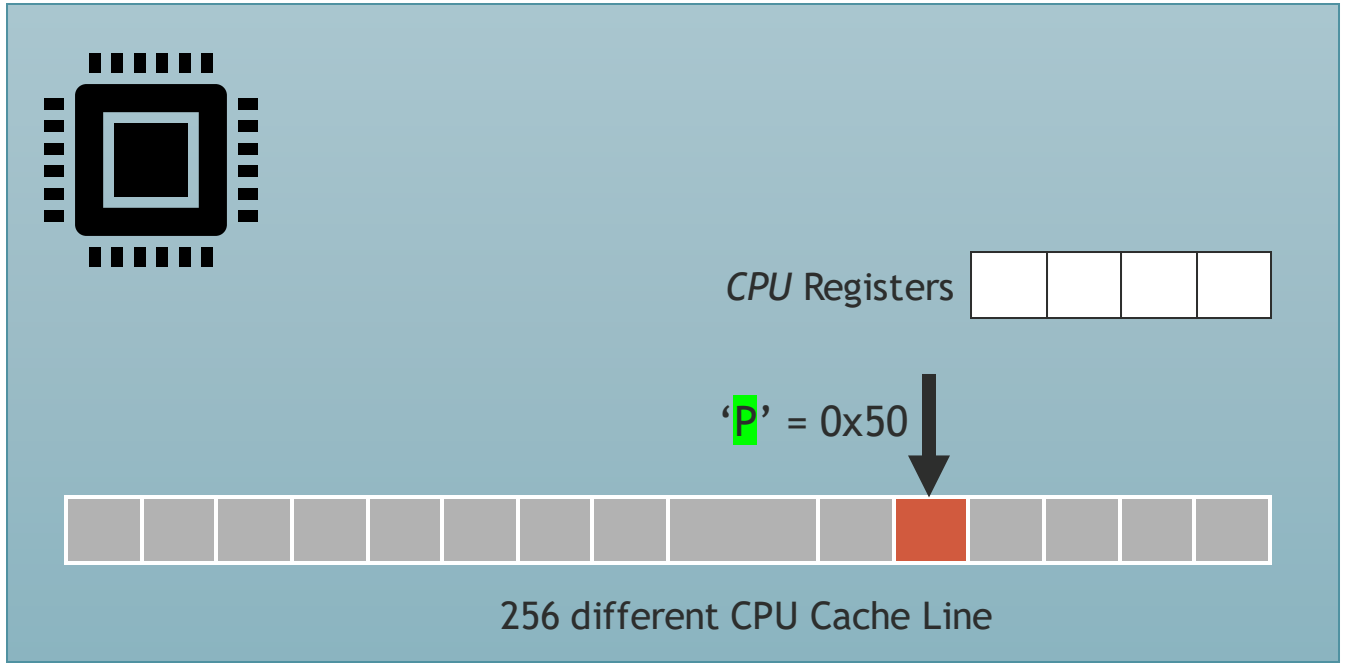


2018: Meltdown Attack?

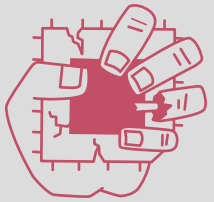


```
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```





Meltdown-style Attacks !!!

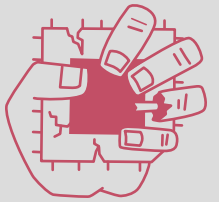


- Can we do Meltdown with other faults/microcode-assists?
- Which part of the CPU leak the data?!

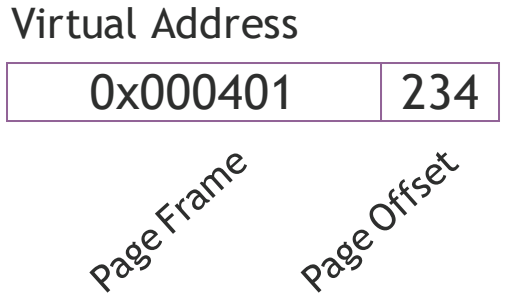
- Can we still leak somebody's data?
  - KPTI
  - Meltdown-resistant CPUs, .e.g. Coffee Lake


Yes!
Zombie Loads...

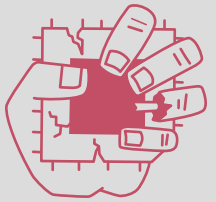
ZombieLoad – How does CPU Work these days?



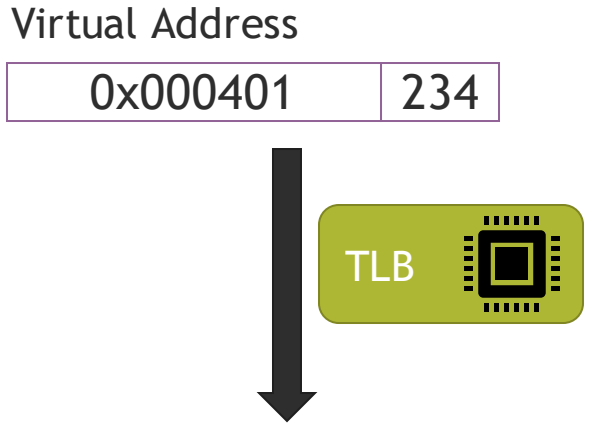
```
mov 0x401234, %rsi

mov (%rsi), %rax
```



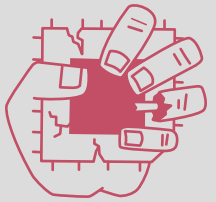
ZombieLoad – How does CPU Work these days?



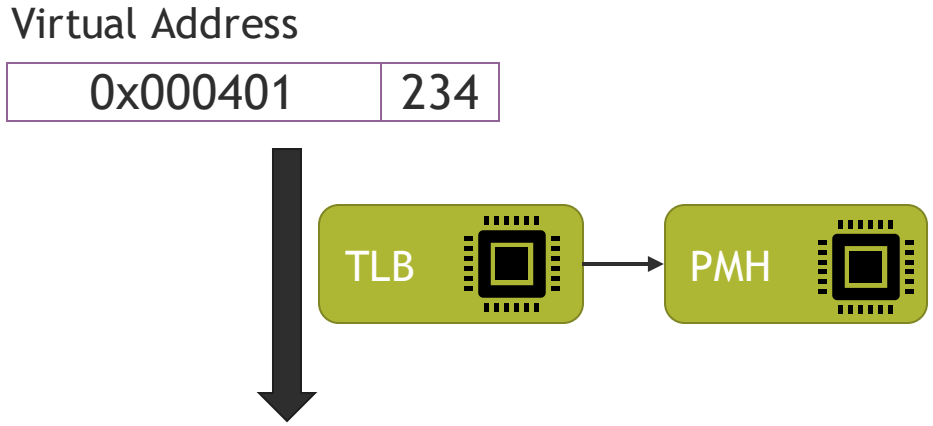
```
mov 0x401234, %rsi

mov (%rsi), %rax
```



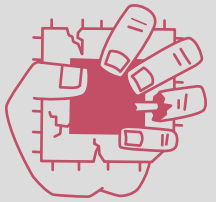
ZombieLoad – How does CPU Work these days?



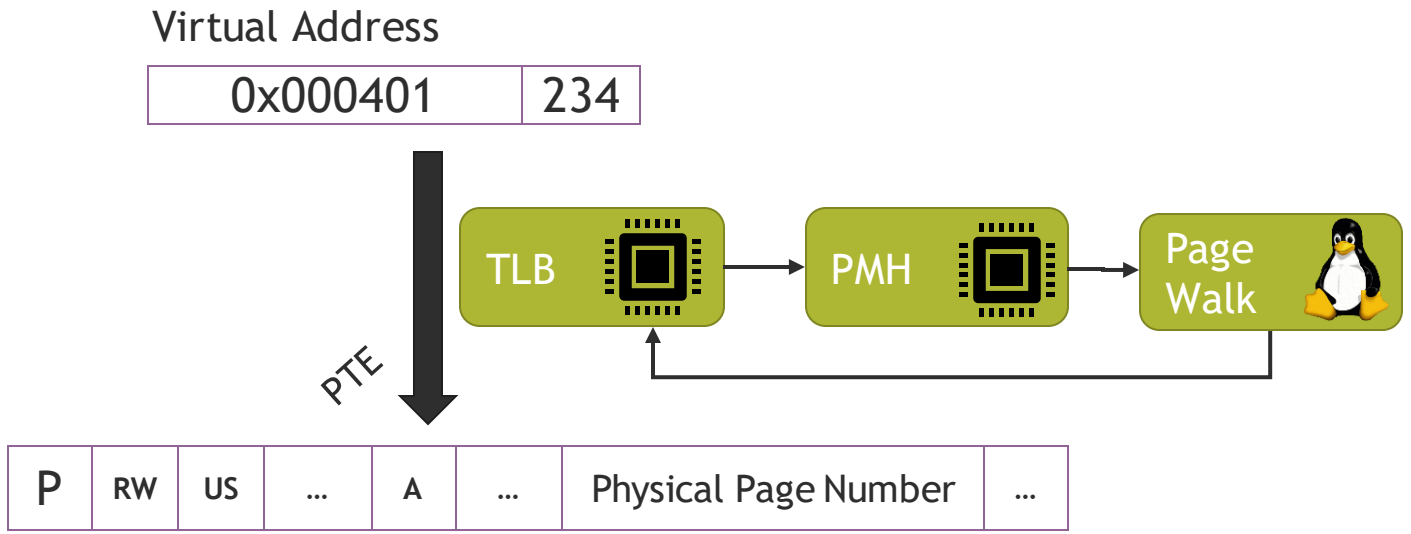
```
mov 0x401234, %rsi

mov (%rsi), %rax
```



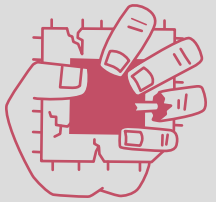
ZombieLoad – How does CPU Work these days?



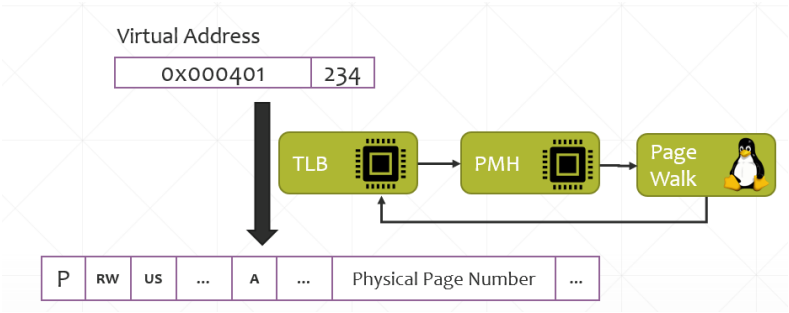
```
mov 0x401234, %rsi

mov (%rsi), %rax
```

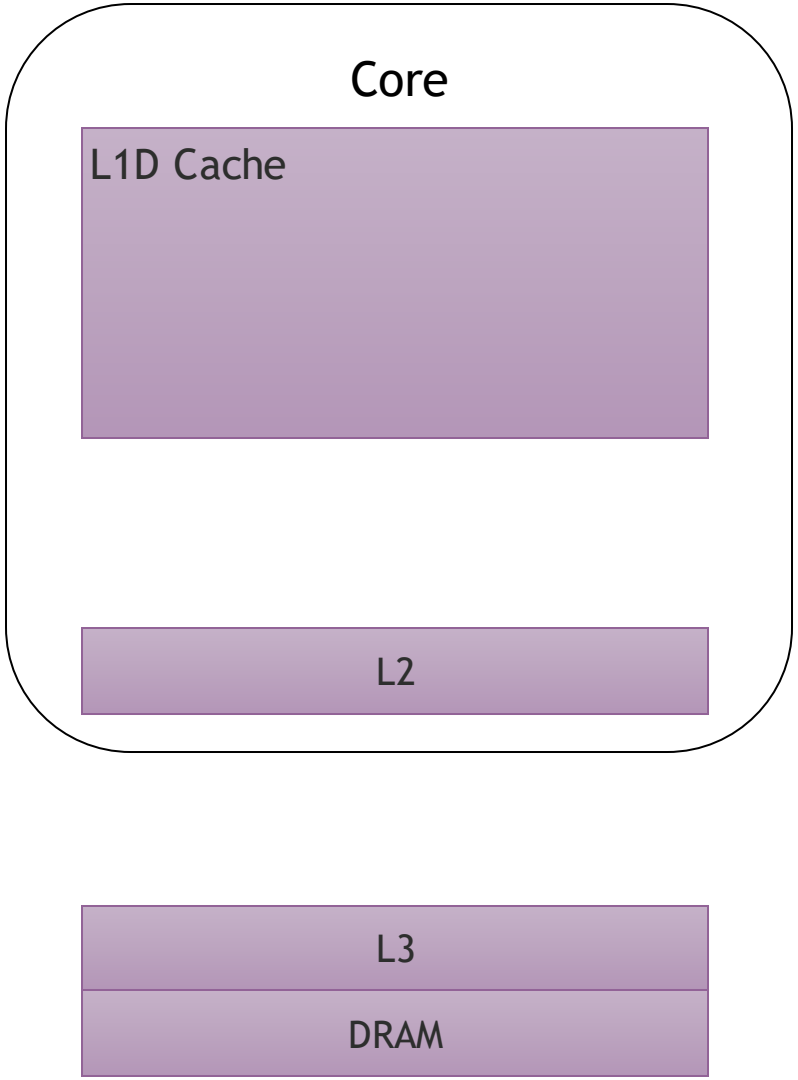


ZombieLoad – How does CPU Work these days?

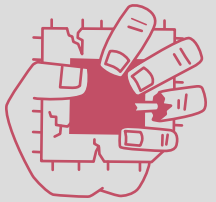


```
mov 0x401234, %rsi

mov (%rsi), %rax
```

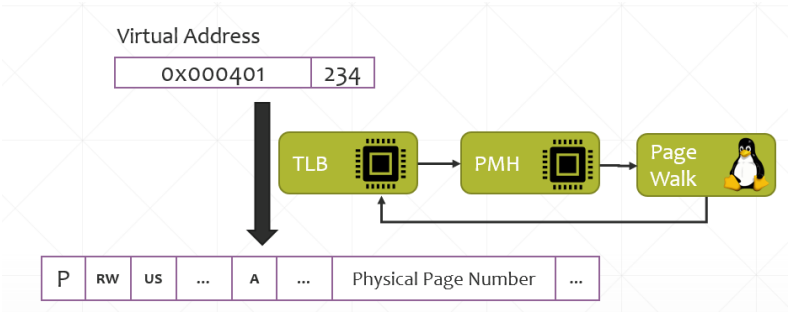


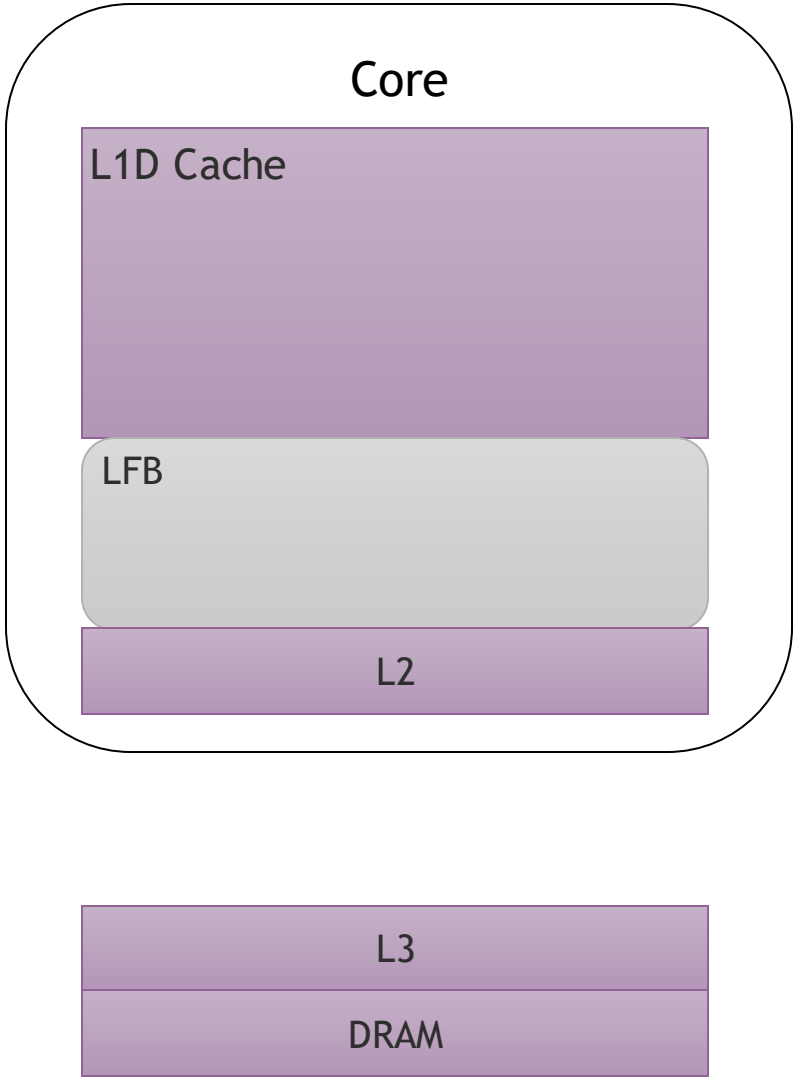


ZombieLoad – How does CPU Work these days?

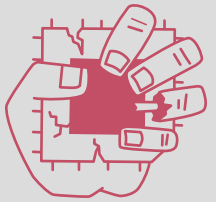


```
mov 0x401234, %rsi

mov (%rsi), %rax
```

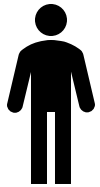


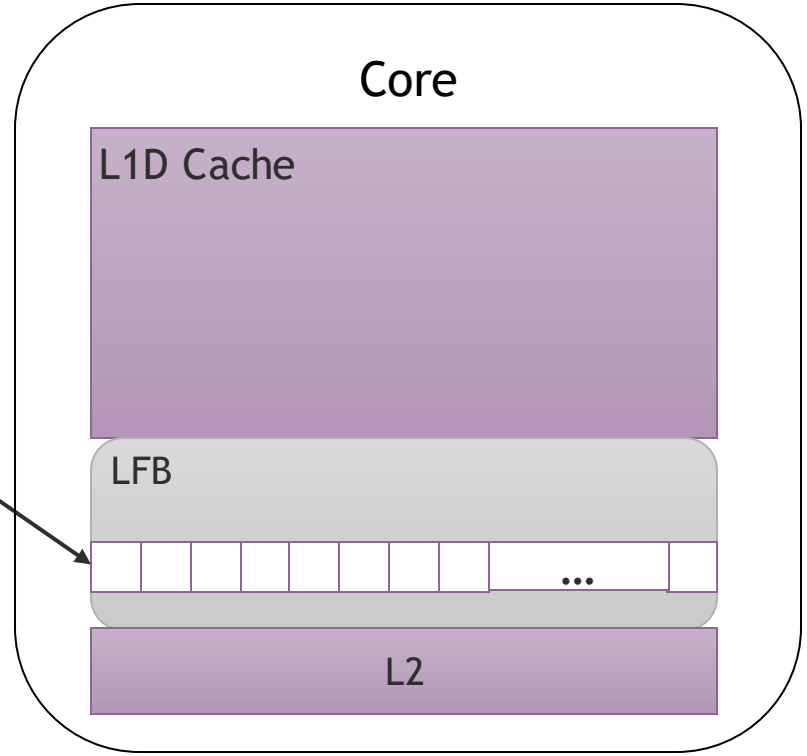


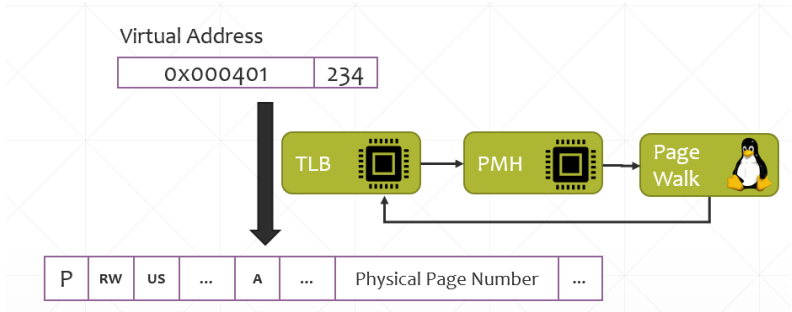
ZombieLoad – How does CPU Work these days?



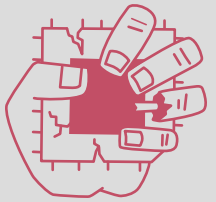
```
mov 0x401234, %rsi

mov (%rsi), %rax
```



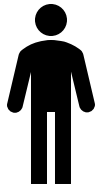


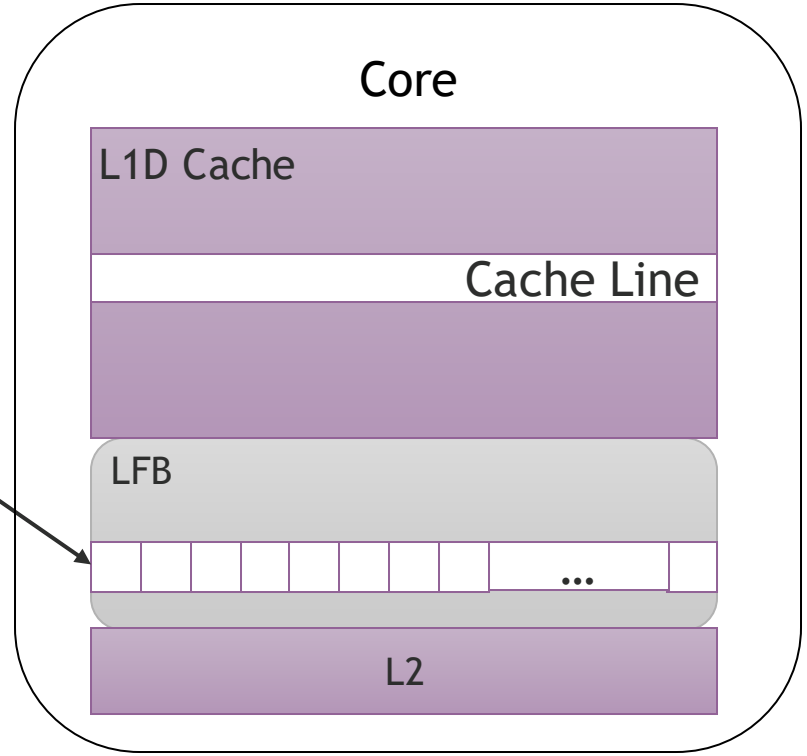


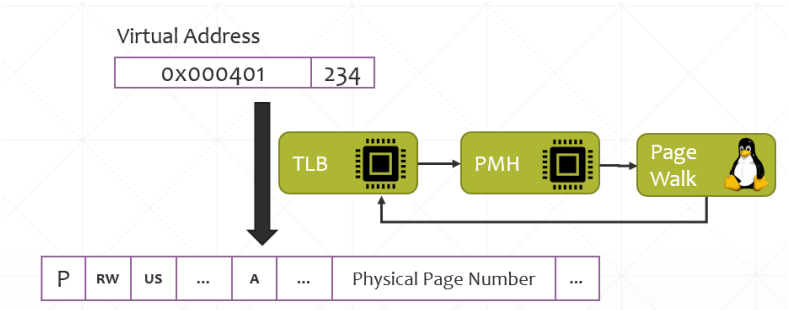
ZombieLoad – How does CPU Work these days?



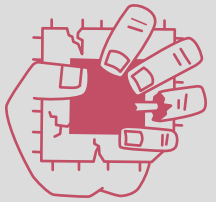
```
mov 0x401234, %rsi

mov (%rsi), %rax
```



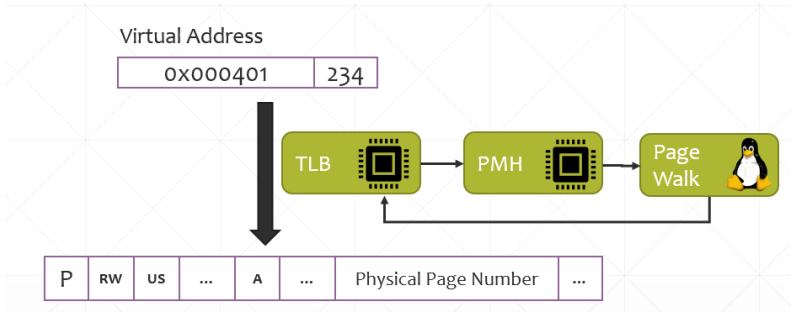


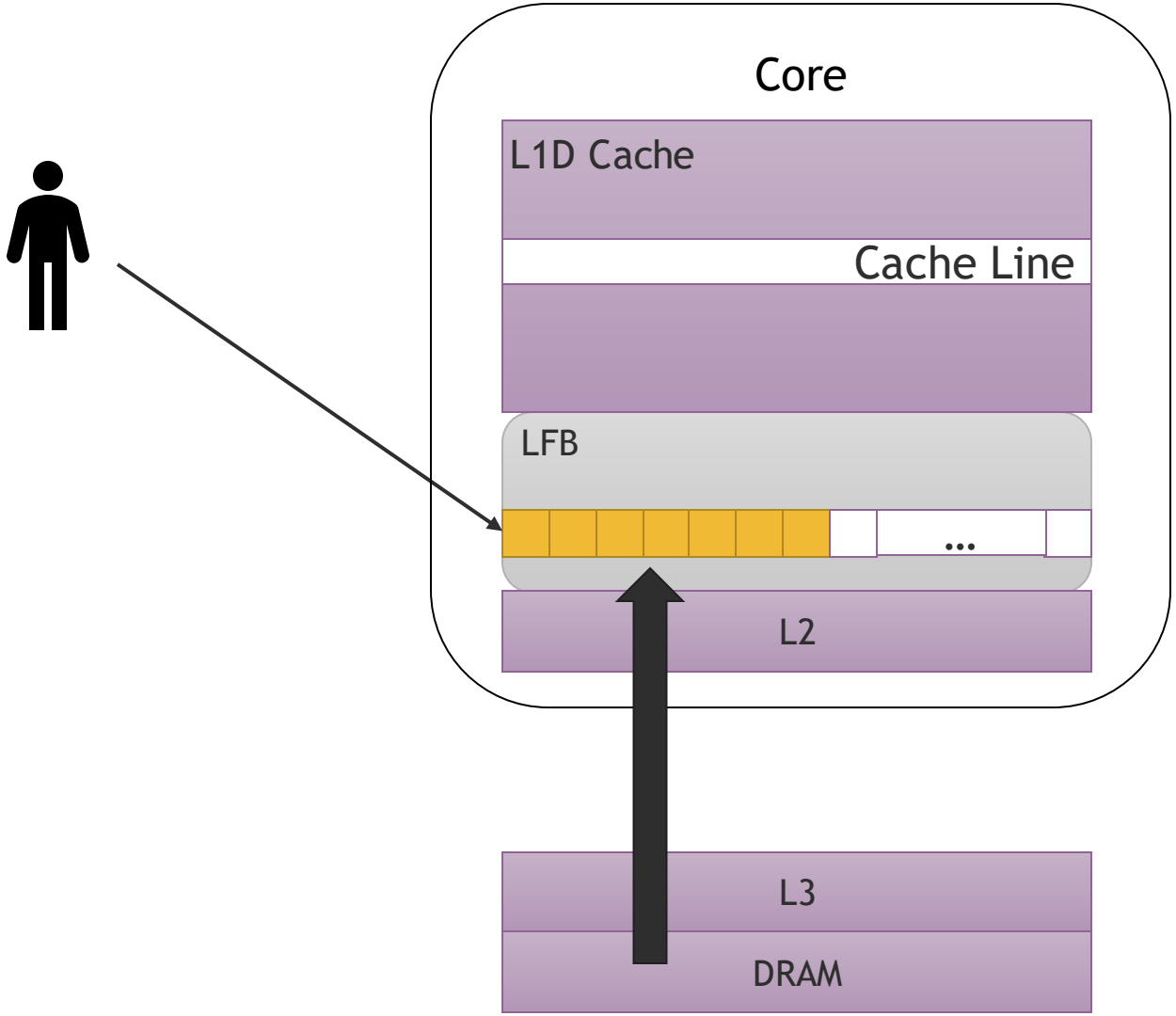


ZombieLoad – How does CPU Work these days?

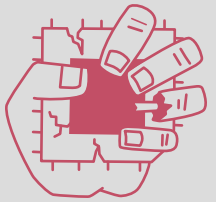


```
mov 0x401234, %rsi

mov (%rsi), %rax
```

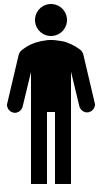


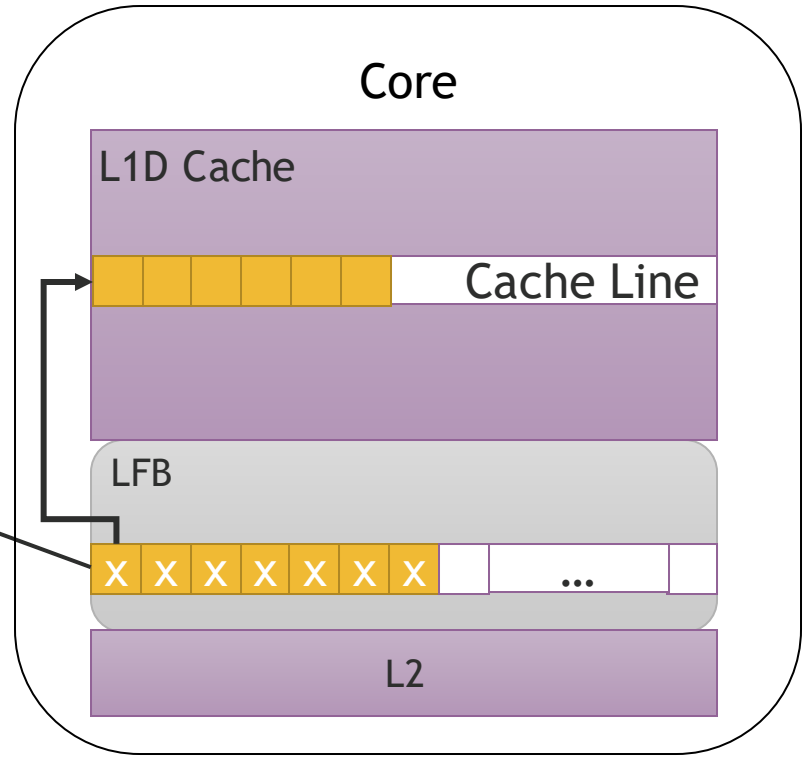


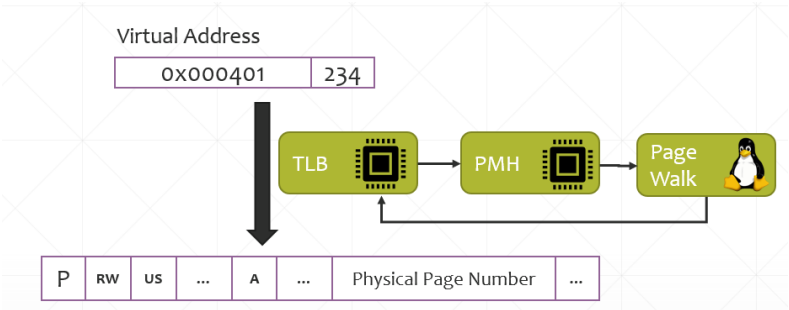
ZombieLoad – How does CPU Work these days?



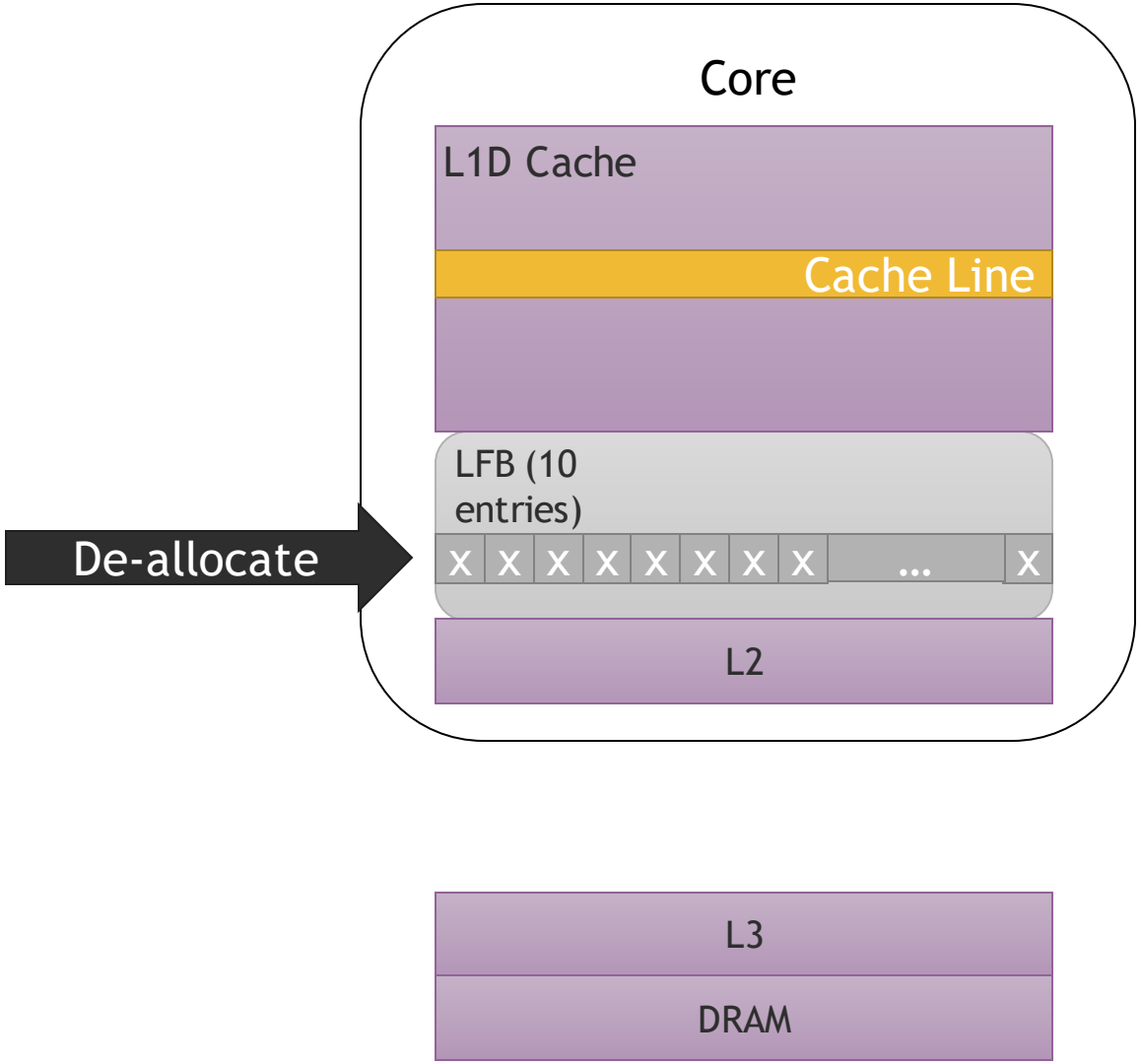
```
mov 0x401234, %rsi
mov (%rsi), %rax
```



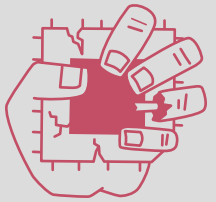




ZombieLoad Attack !?!

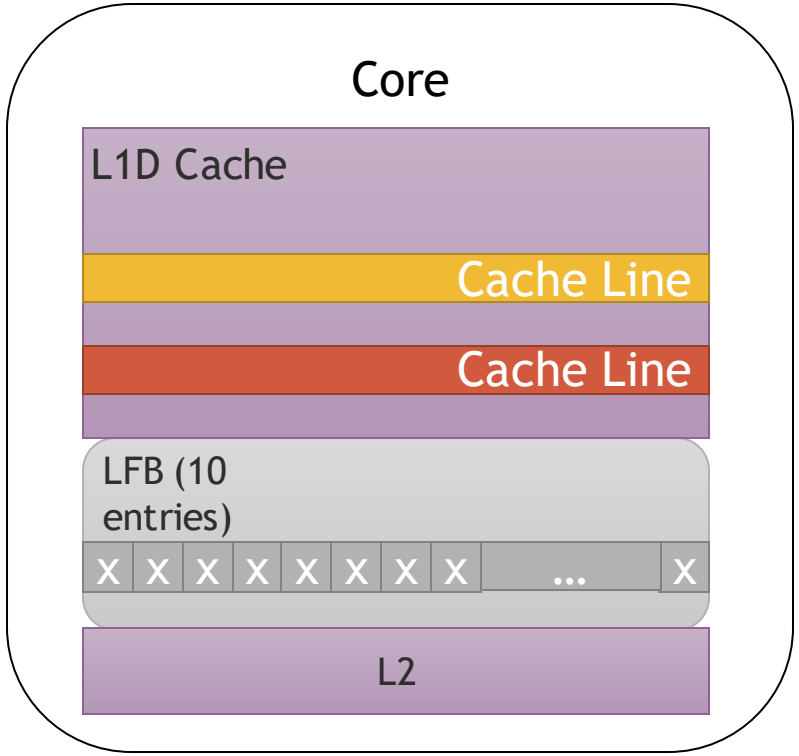


ZombieLoad Attack !?!

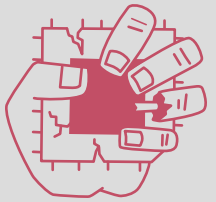


P	RW	US	…	A	…	Physical Page Number	…

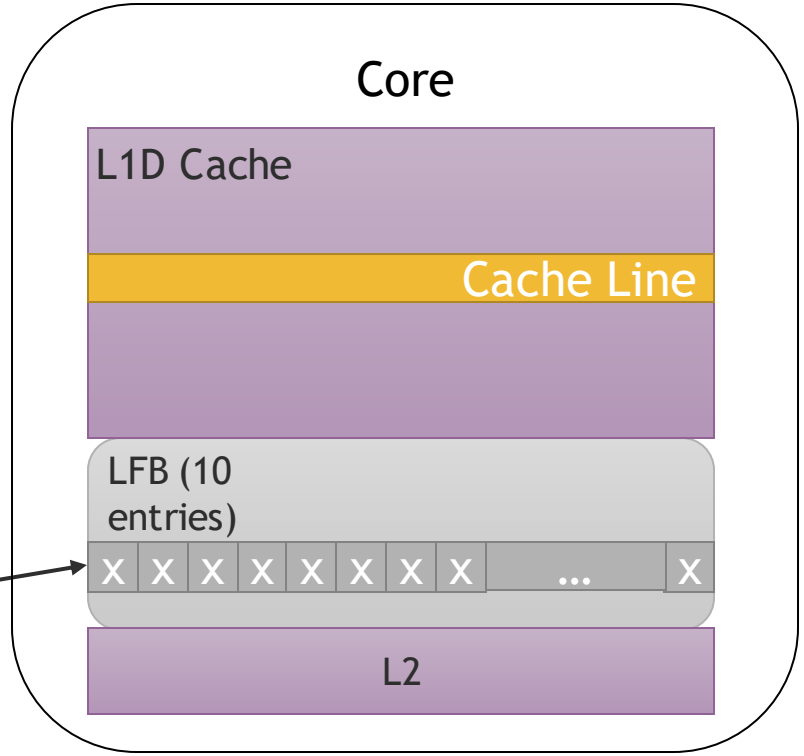
```
char secret = *(char *) 0xffffffff81a0123;
```



ZombieLoad Attack !?!

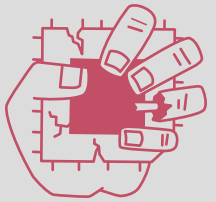


P	RW	US	…	A	…	Physical Page Number	…

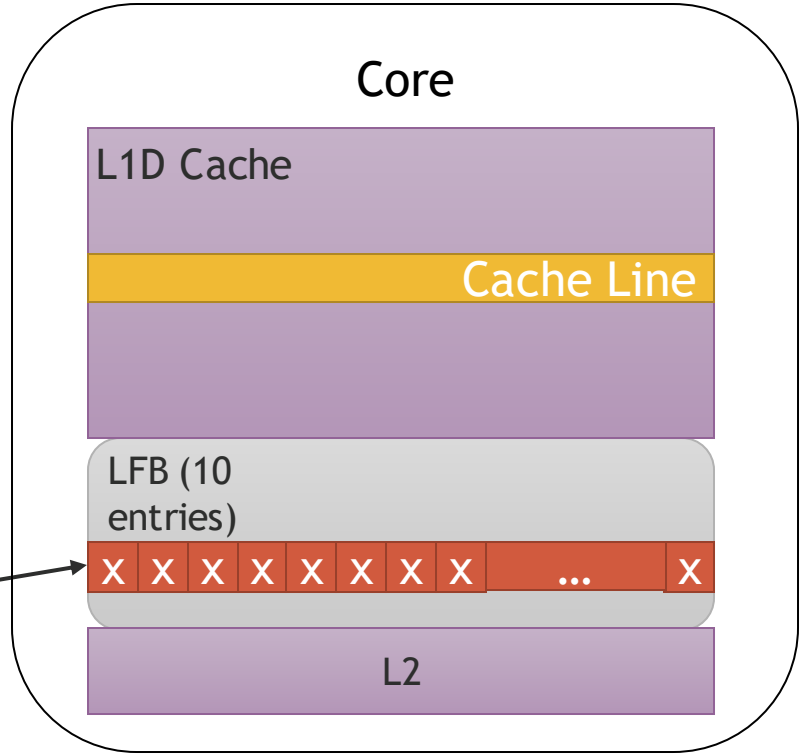



```
char secret = *(char *) 0xffffffff81a0123;
```

ZombieLoad Attack !?!

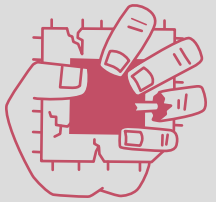


P	RW	US	…	A	…	Physical Page Number	…



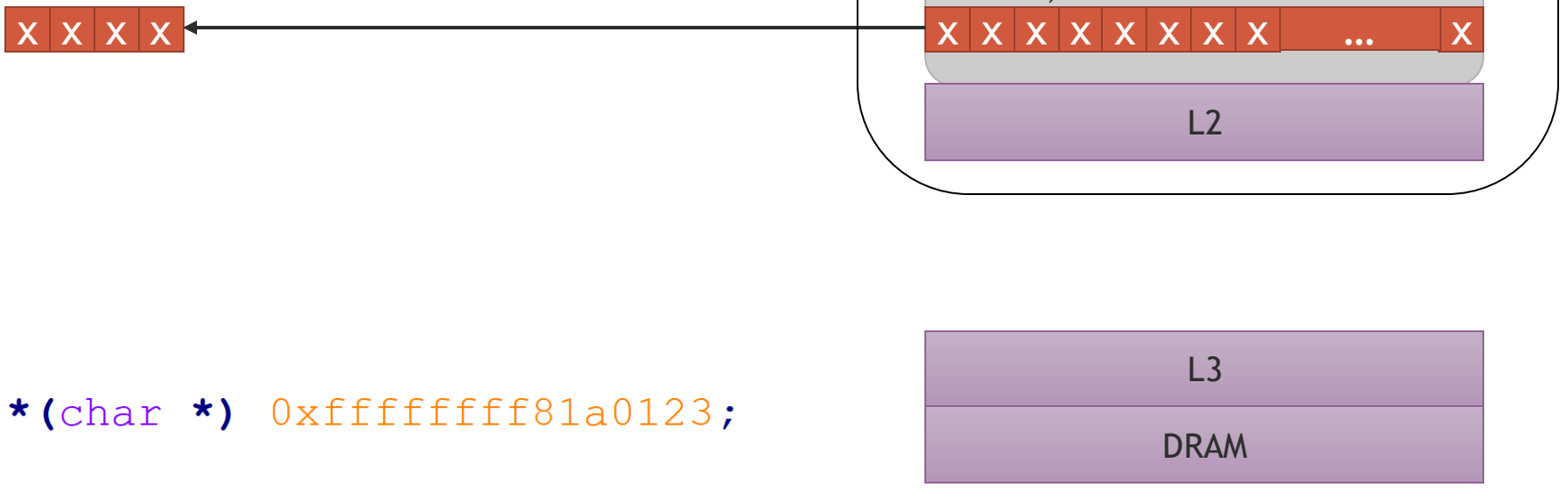

```
char secret = *(char *) 0xffffffff81a0123;
```

ZombieLoad Attack !?!

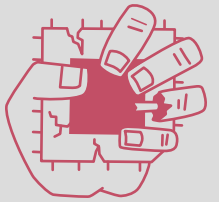


P	RW	US	…	A	…	Physical Page Number	…

```
char secret = *(char *) 0xffffffff81a0123;
```

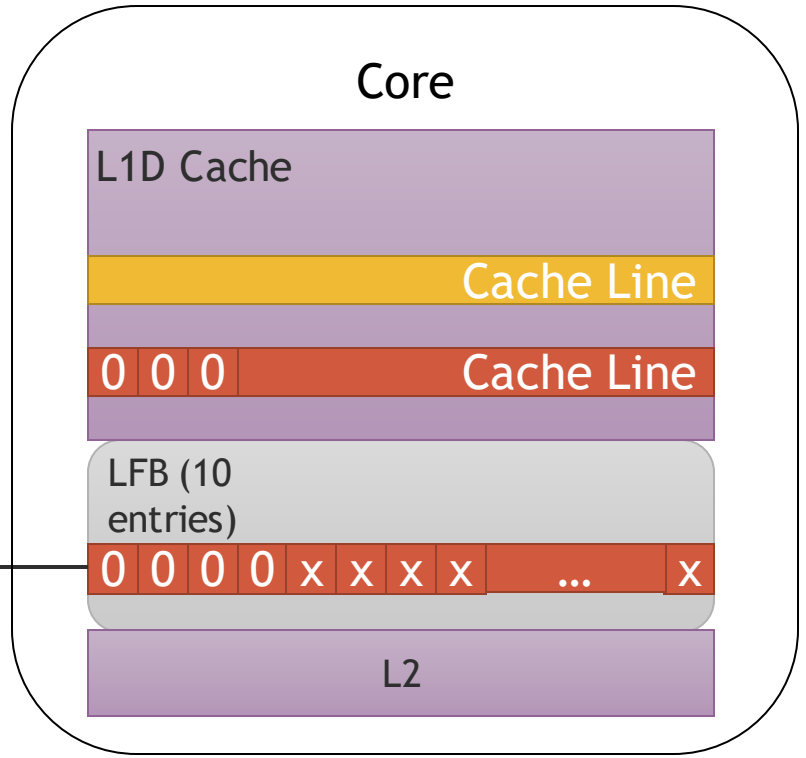


ZombieLoad Attack !?!



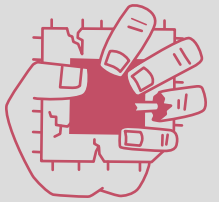
P	RW	US	…	A	…	Physical Page Number	…

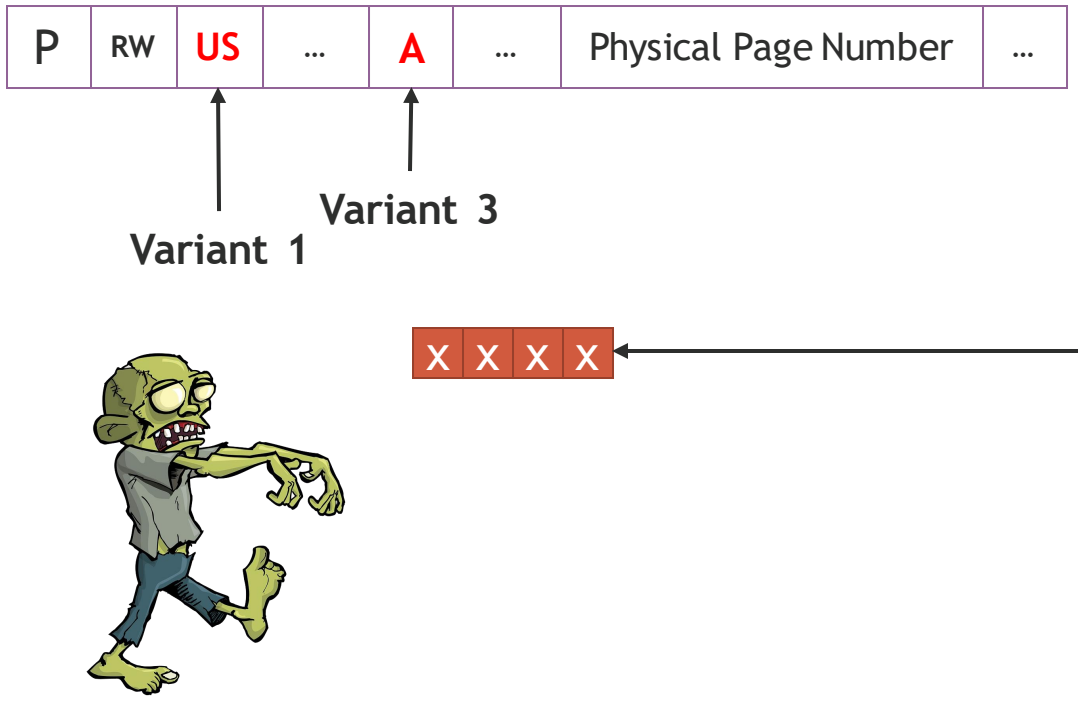
x x x x

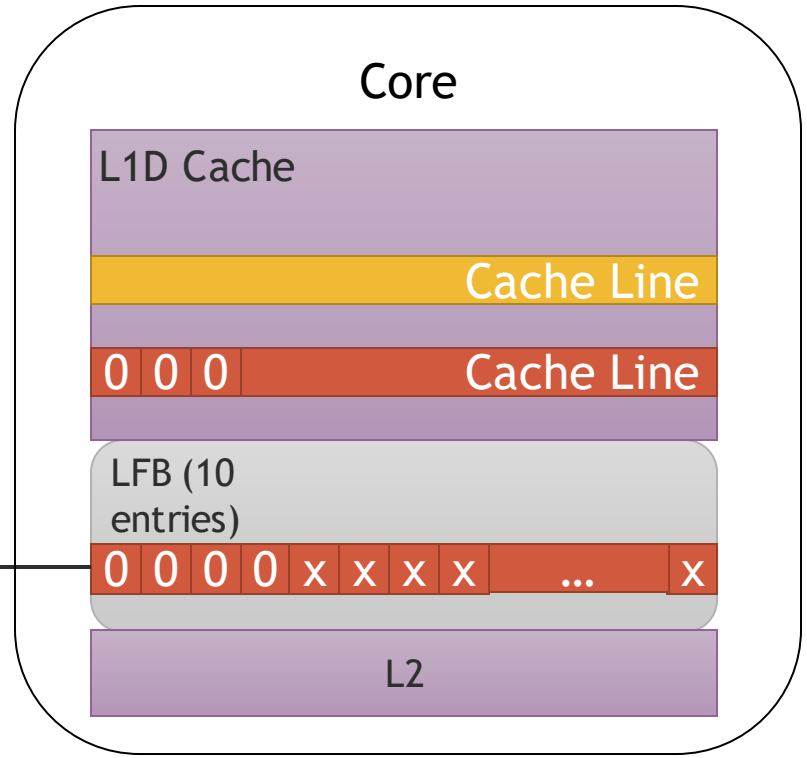



```
char secret = *(char *) 0xffffffff81a0123;
```

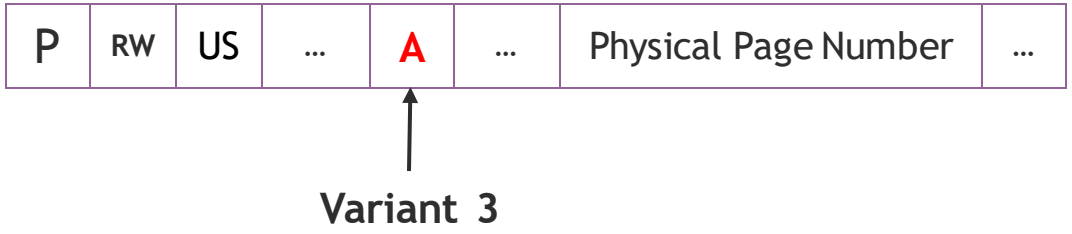
ZombieLoad Attack !?!







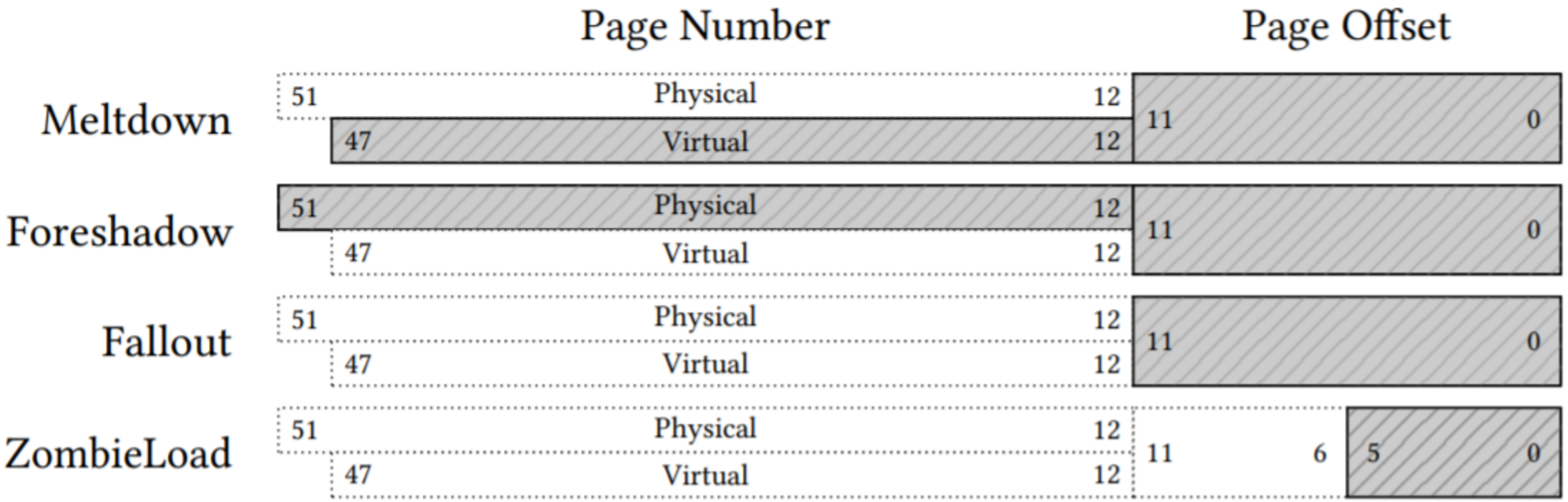
ZombieLoad – Microcode Assist on '*A*' Bit



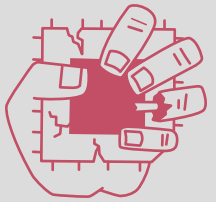
- **Access Bit**
  - **CPU** tells → **OS**: A page has been accessed by setting the 'A' Bit
  - **OS** tells → **CPU:** A page has not been accessed (just allocated) by clearing the bit

- 'A' Bit Microcode Assist
  - Microcode Assists: The CPU executes an internal event handler to service complex instructions/operations
  - The microcode assist flushes the pipeline.
  - Intel CPUs set 'A' bit using a microcode assist

ZombieLoad VS. other Meltdown-Style Attacks



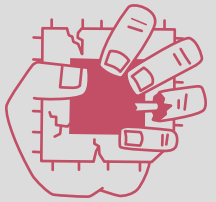
What can we do with this data leakage?



- **Architecturally**
  - Attack across Process Context Switches
  - Attack across Simultaneous Multithreading (SMT) AKA. Intel Hyperthreading

- **Scenarios:**
  - Cross-Process
  - Cross-VM
  - Intel SGX

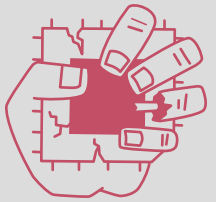
Data Sampling - Domino Attack



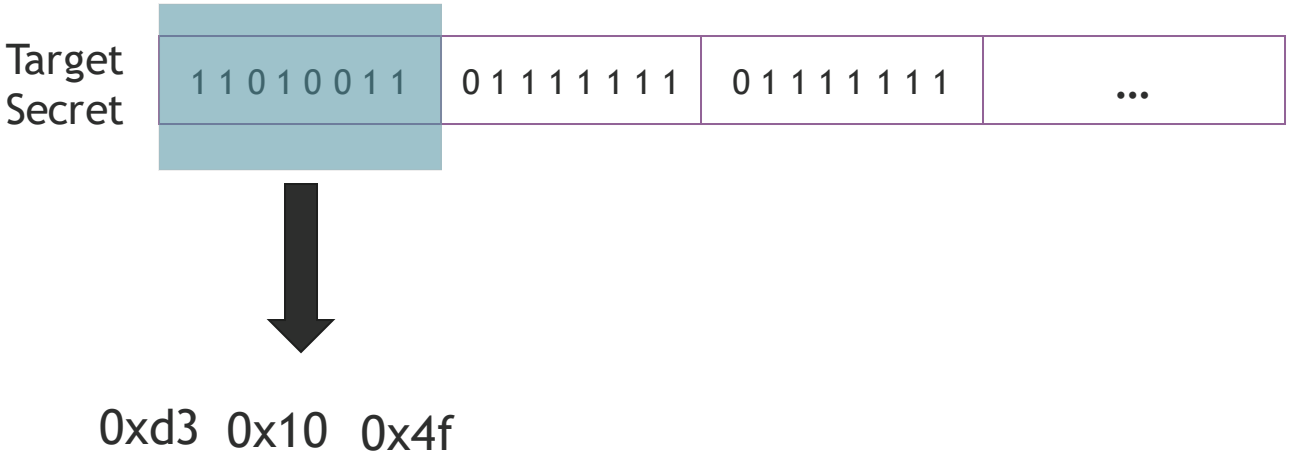
- We may leak bytes of data from other unimportant fill buffer entries
- Leak domino bytes to perform error correction

Target Secret				
	1 1 0 1 0 0 1 1	0 1 1 1 1 1 1 1	0 1 1 1 1 1 1 1	…

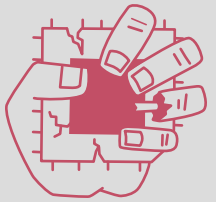
Data Sampling - Domino Attack



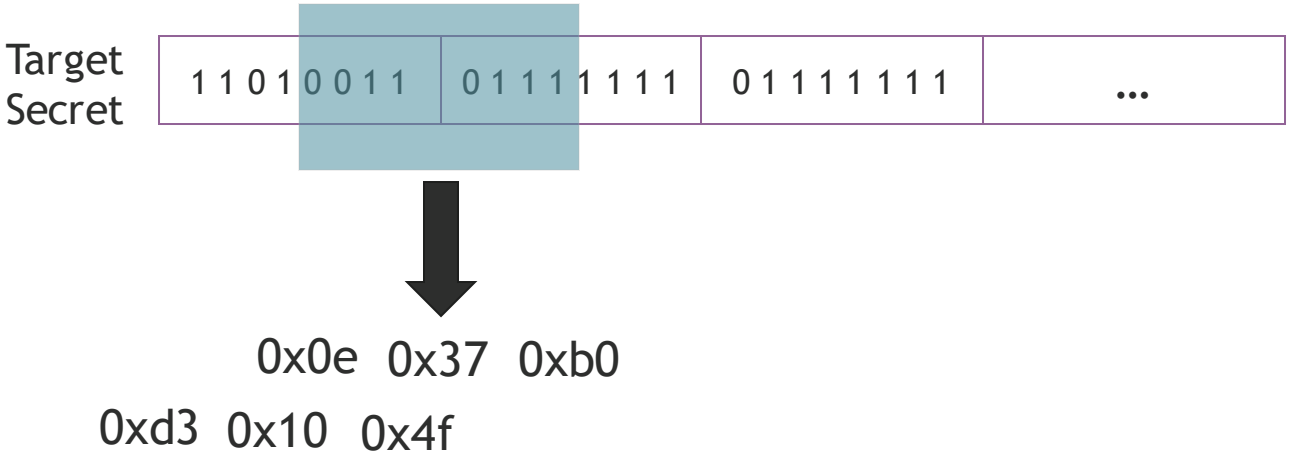
- We may leak bytes of data from other unimportant fill buffer entries
- Leak domino bytes to perform error correction



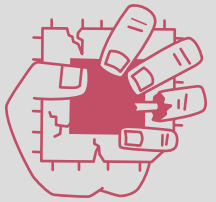
Data Sampling - Domino Attack



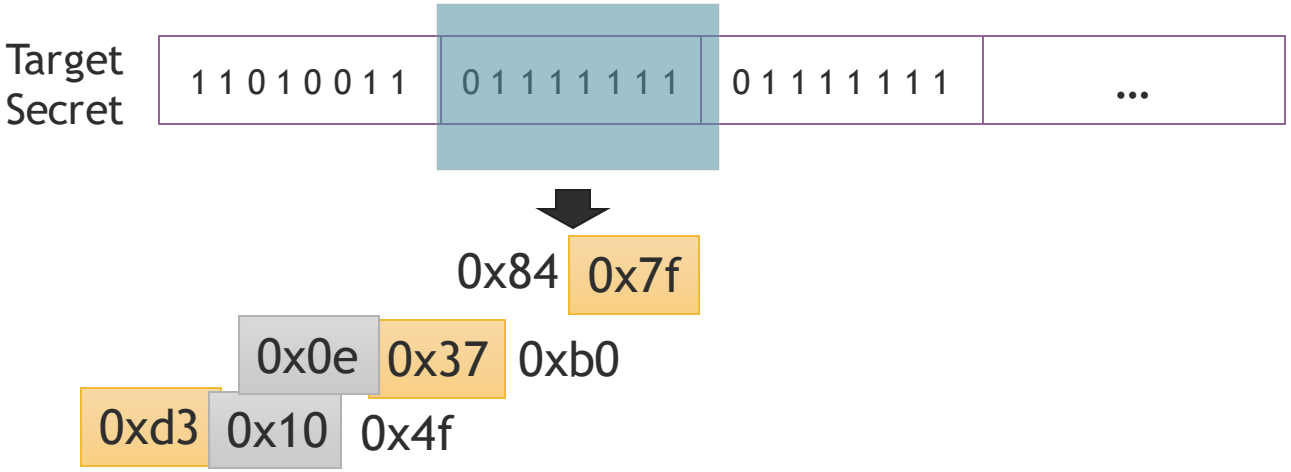
- We may leak bytes of data from other unimportant fill buffer entries
- Leak domino bytes to perform error correction



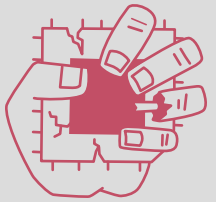
Data Sampling - Domino Attack



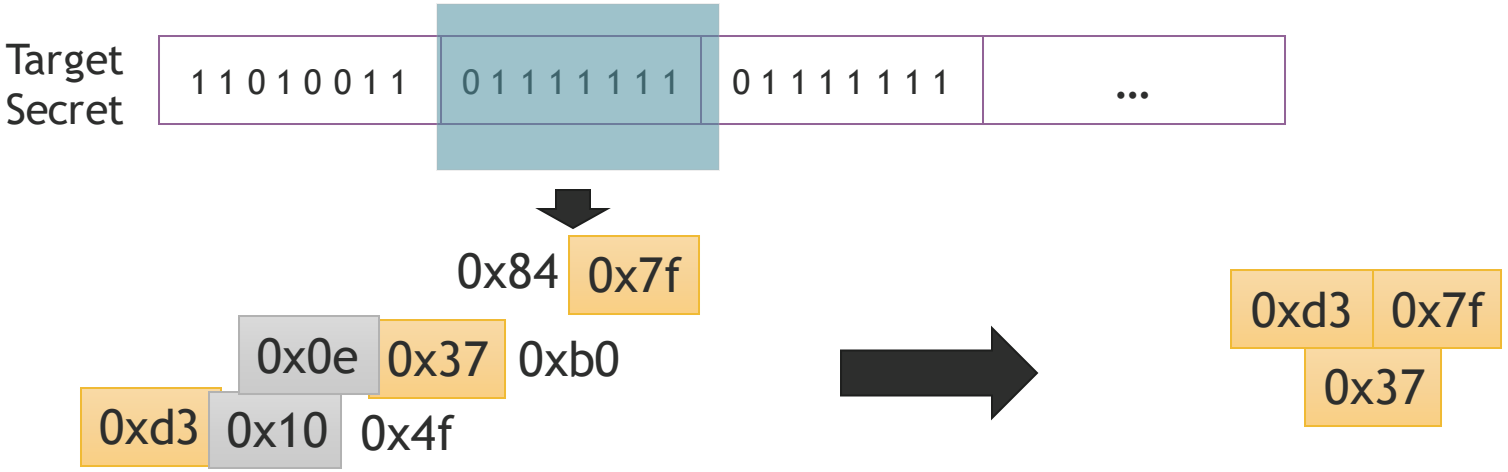
- We may leak bytes of data from other unimportant fill buffer entries
- Leak domino bytes to perform error correction

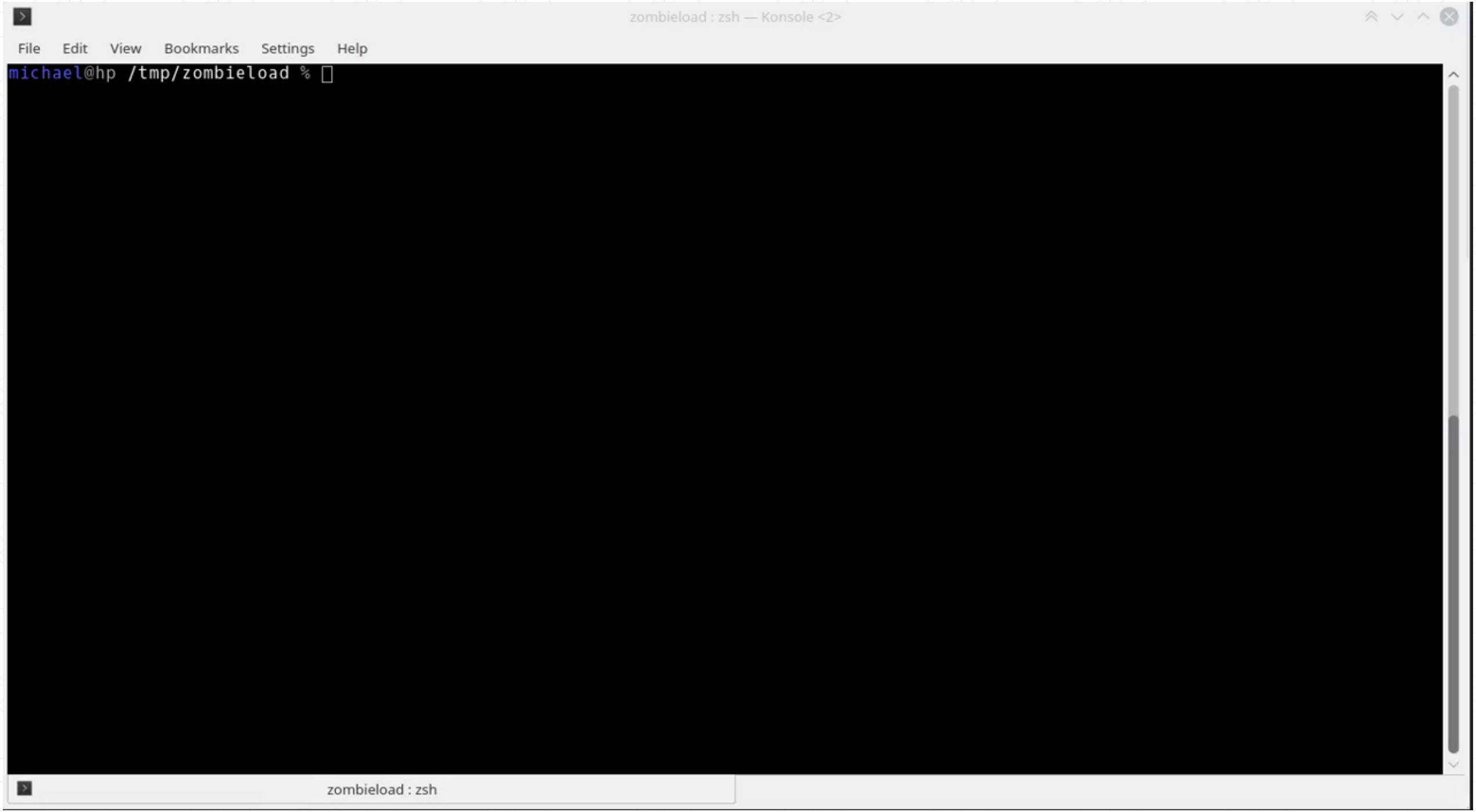


Data Sampling - Domino Attack

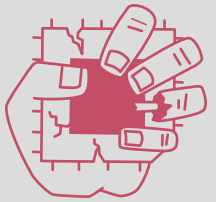


- We may leak bytes of data from other unimportant fill buffer entries
- Leak domino bytes to perform error correction



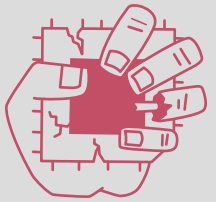

zombieload : zsh — Konsole <2>
File Edit View Bookmarks Settings Help
michael@hp /tmp/zombieload %
zombieload : zsh

Recovering Intel SGX Sealing Key



- Intel SGX allow developers to have hardware support for TEE
- Malicious OS is part of the threat model
- We can read register values of a trusted enclave with help of a malicious OS

Recovering Intel SGX Sealing Key

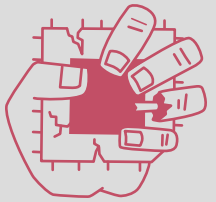


- Intel SGX allow developers to have hardware support for TEE
- Malicious OS is part of the threat model
- We can read register values of a trusted enclave with help of a malicious OS

sgx-step

```
mov
add
xor
mov 0x4142434445464748, %rax
call
nop
jmp
```

Recovering Intel SGX Sealing Key

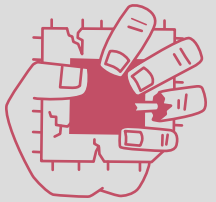


- Intel SGX allow developers to have hardware support for TEE
- Malicious OS is part of the threat model
- We can read register values of a trusted enclave with help of a malicious OS

sgx-step

```
mov
add
xor
mov 0x4142434445464748, %rax
call
nop
jmp
```

Recovering Intel SGX Sealing Key

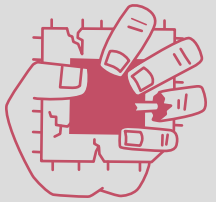


- Intel SGX allow developers to have hardware support for TEE
- Malicious OS is part of the threat model
- We can read register values of a trusted enclave with help of a malicious OS

sgx-step

```
mov
add
xor
mov 0x4142434445464748, %rax
call
nop
jmp
```

Recovering Intel SGX Sealing Key

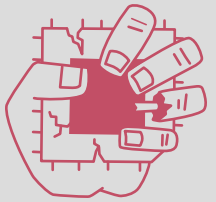


- Intel SGX allow developers to have hardware support for TEE
- Malicious OS is part of the threat model
- We can read register values of a trusted enclave with help of a malicious OS

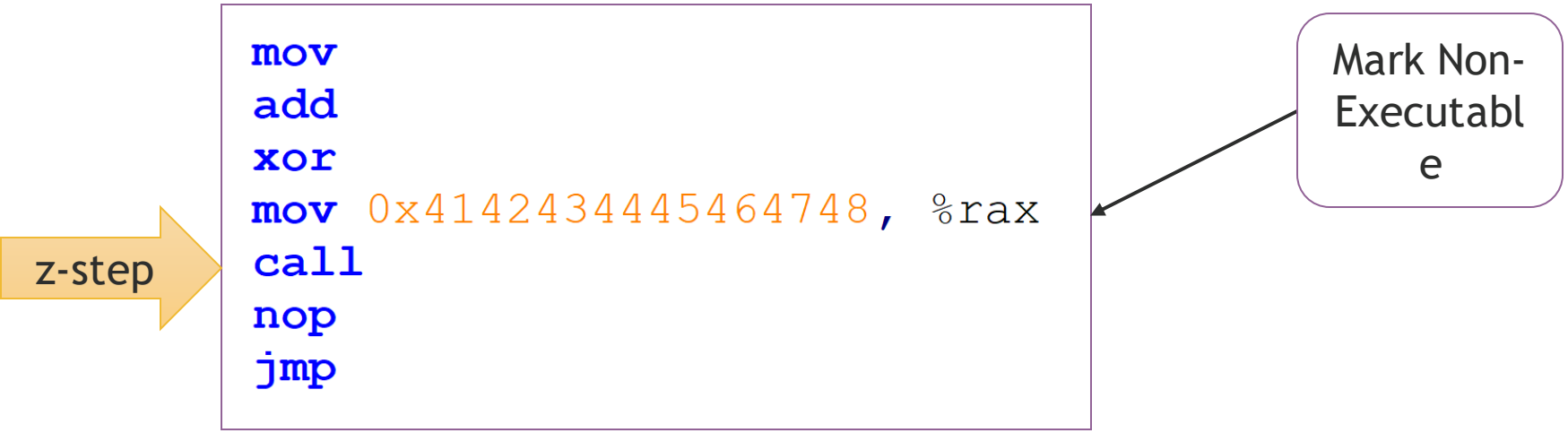
```
mov
add
xor
mov 0x4142434445464748, %rax
call
nop
jmp
```

sgx-step

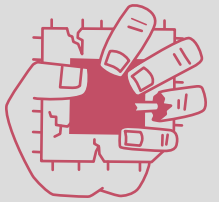
Recovering Intel SGX Sealing Key



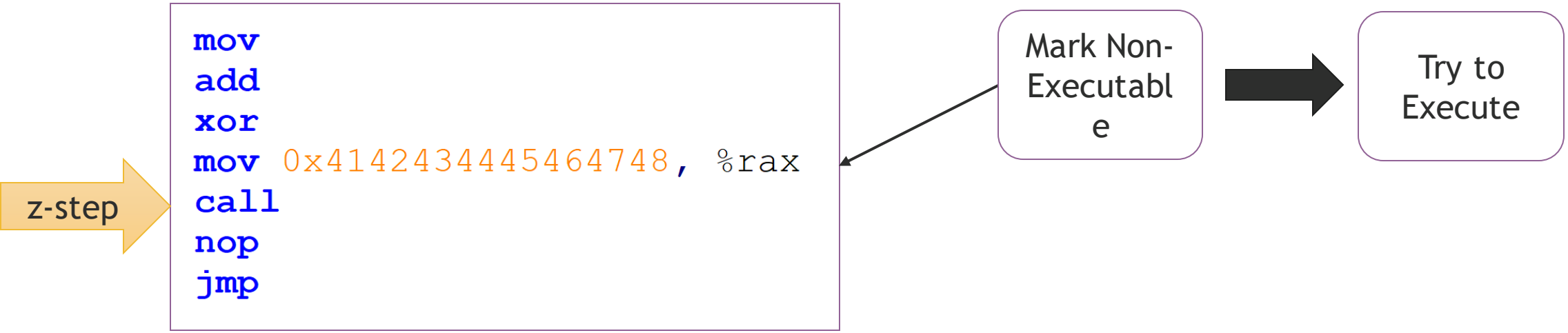
- Intel SGX allow developers to have hardware support for TEE
- Malicious OS is part of the threat model
- We can read register values of a trusted enclave with help of a malicious OS



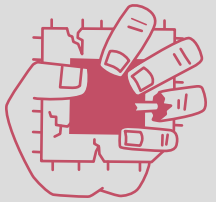
Recovering Intel SGX Sealing Key



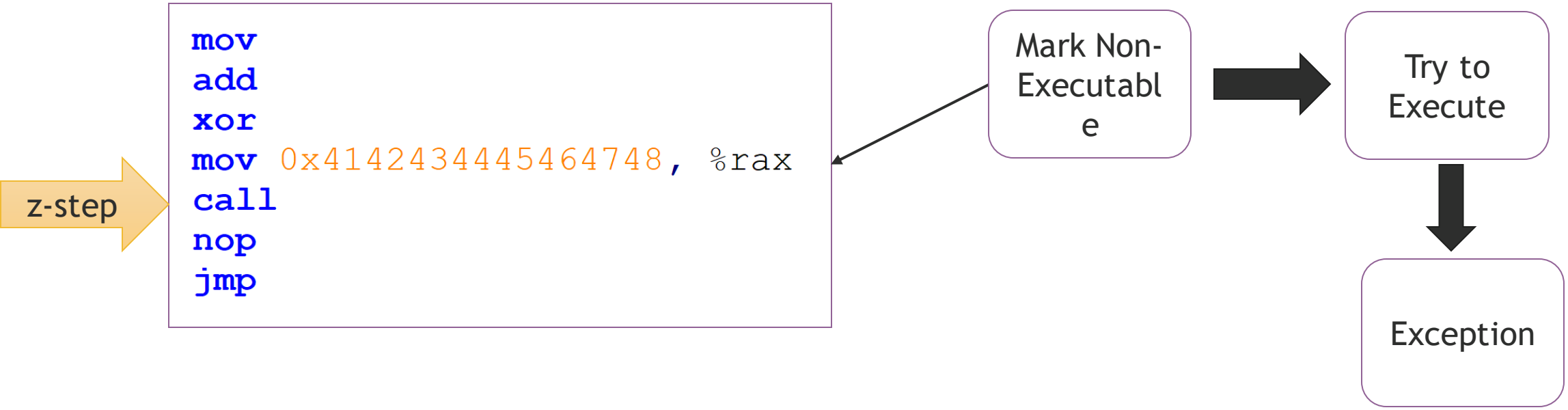
- Intel SGX allow developers to have hardware support for TEE
- Malicious OS is part of the threat model
- We can read register values of a trusted enclave with help of a malicious OS



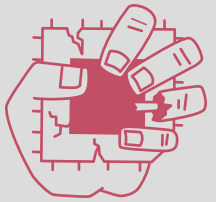
Recovering Intel SGX Sealing Key



- Intel SGX allow developers to have hardware support for TEE
- Malicious OS is part of the threat model
- We can read register values of a trusted enclave with help of a malicious OS



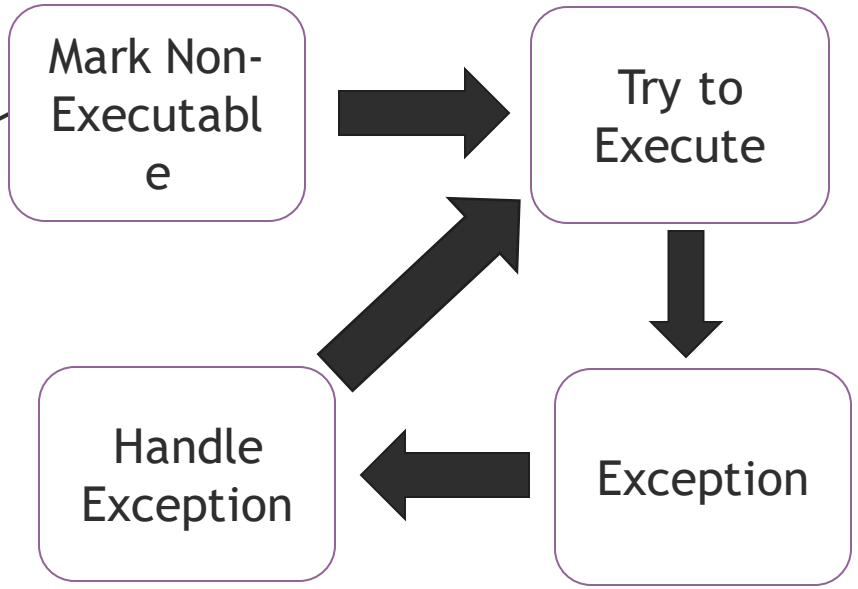
Recovering Intel SGX Sealing Key



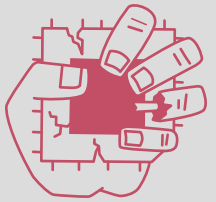
- Intel SGX allow developers to have hardware support for TEE
- Malicious OS is part of the threat model
- We can read register values of a trusted enclave with help of a malicious OS

z-step

```
mov
add
xor
mov 0x4142434445464748, %rax
call
nop
jmp
```

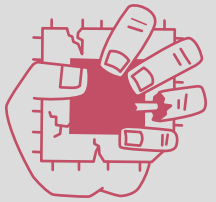


Recovering Intel SGX Sealing Key



- Intel SGX allow developers to have hardware support for TEE
- Malicious OS is part of the threat model
- We can read register values of a trusted enclave with help of a malicious OS
- Repeated Context Switch in the transient domain w/ the same register values

Is there any Mitigation?



- **Short-term**
  - Intel suggested an instruction sequence to fill all the buffers across context switch
  - Disable hyperthreading
  - Intel SGX: Remote attestation to Verify hyperthreading is Disabled

- **Long-term**
  - Microarchitectural hardware fixes (Buy new CPUs !! ☺)

https://zombieloadattack.com/