

Exploiting Microarchitectural Flaws

in the Heart of the Memory Subsystem

Daniel Moghimi, Worcester Polytechnic University



Feb 20, 2020

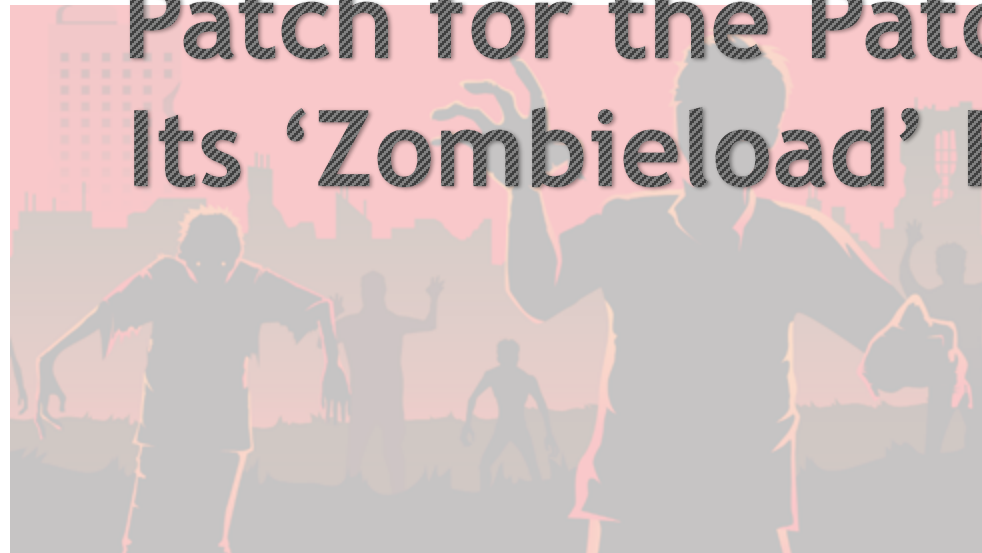
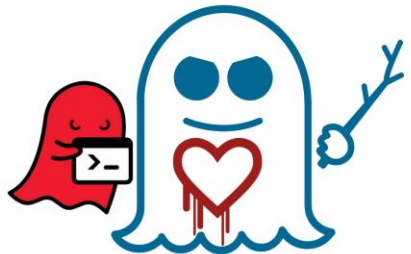
Columbia University

Spoiler!!

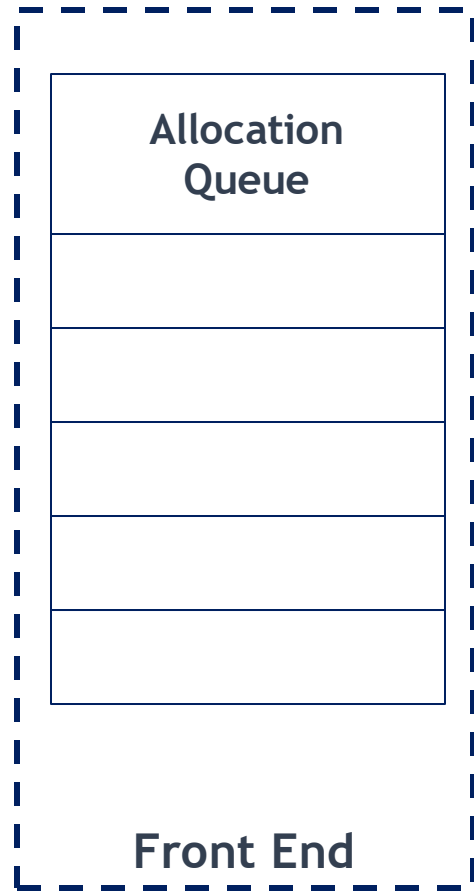
“New Intel CPU Vulnerability Bodes Well For AMD”



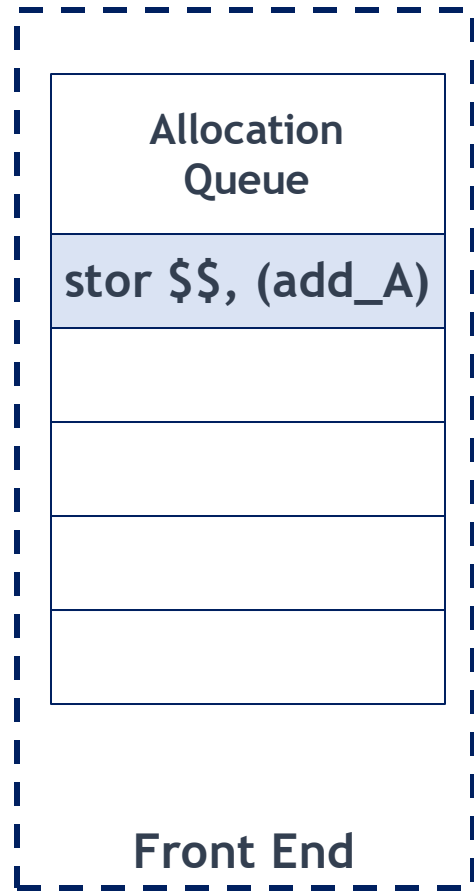
“Intel Is Patching the Patch for the Patch for Its ‘Zombieload’ Flaw”



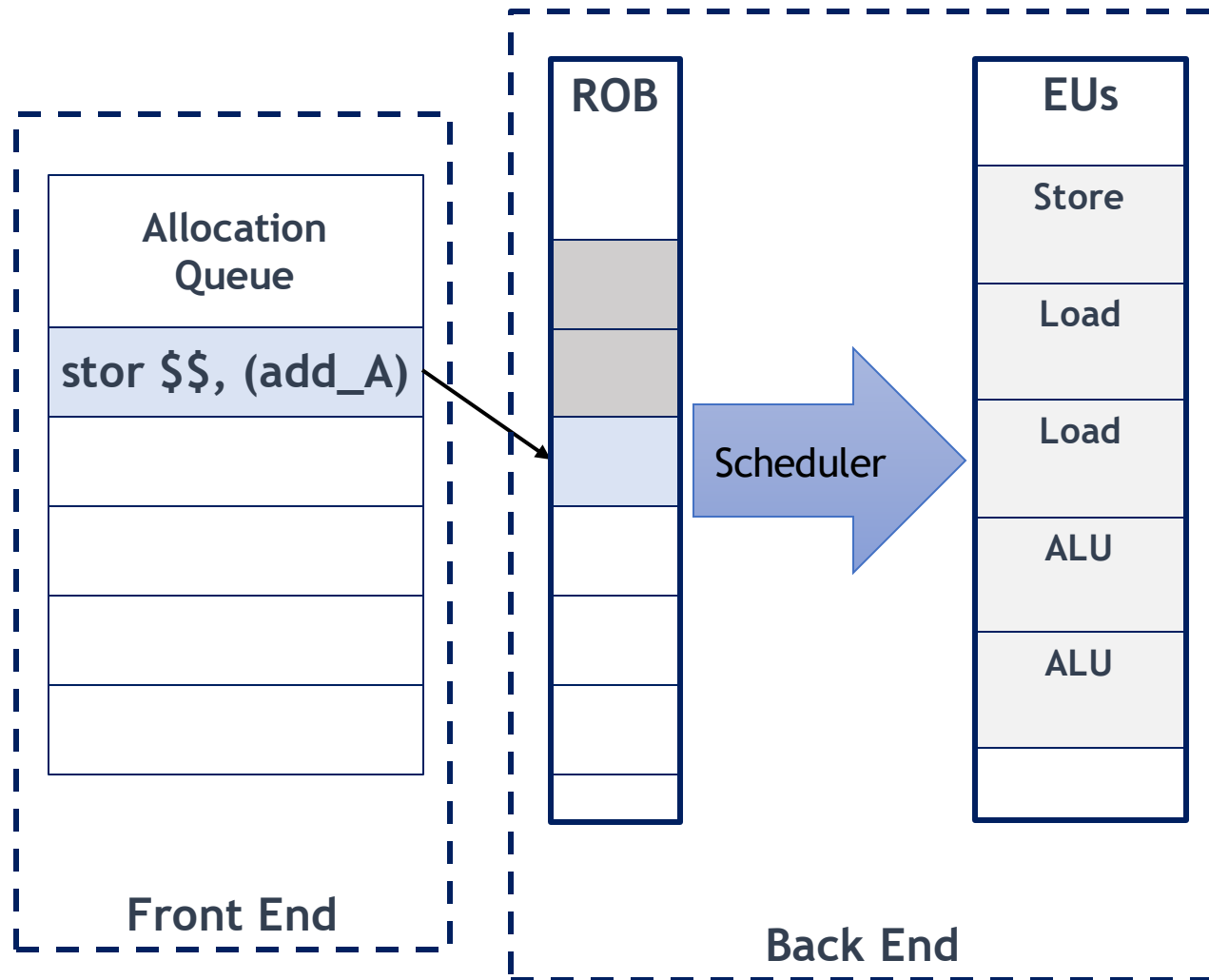
CPU Memory Subsystem



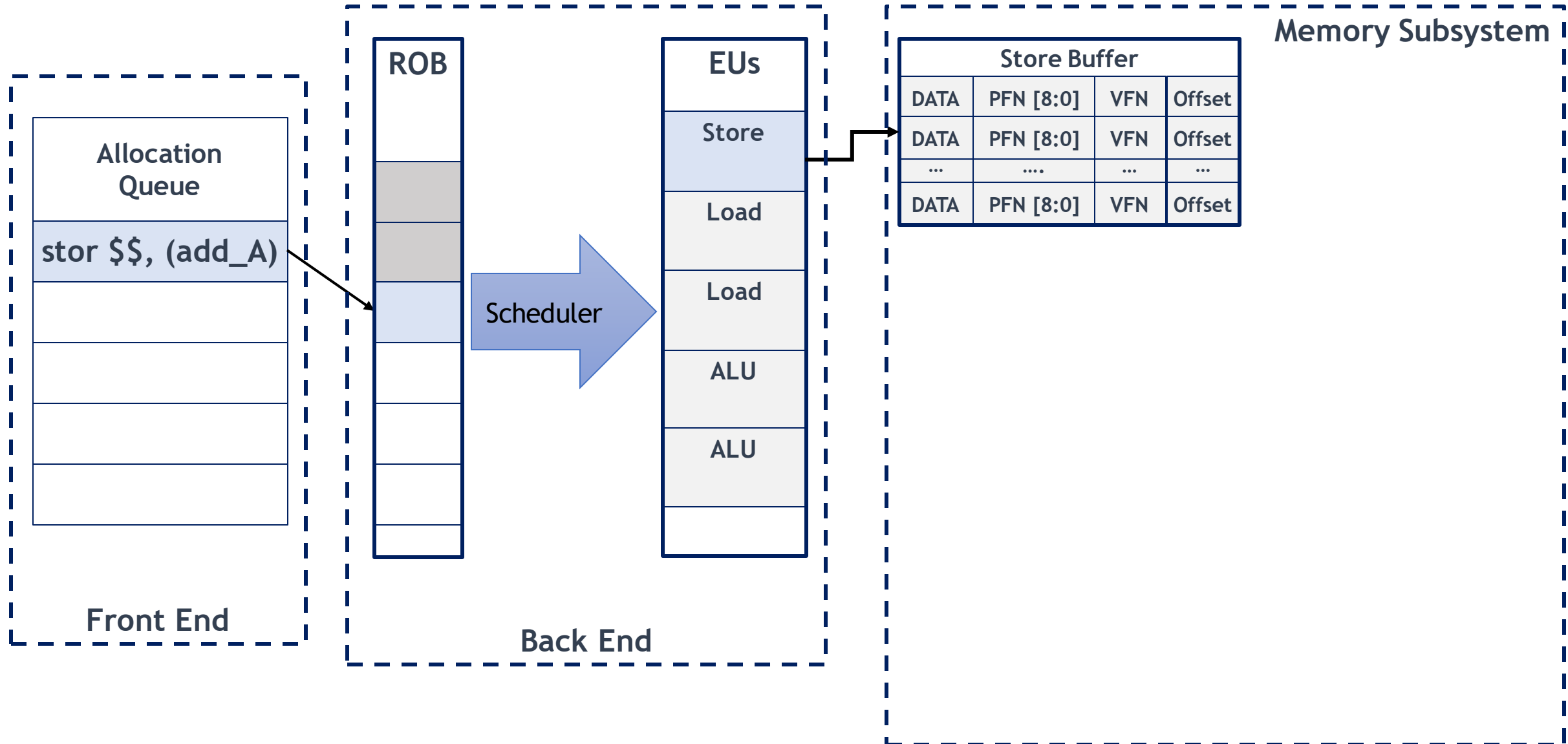
CPU Memory Subsystem



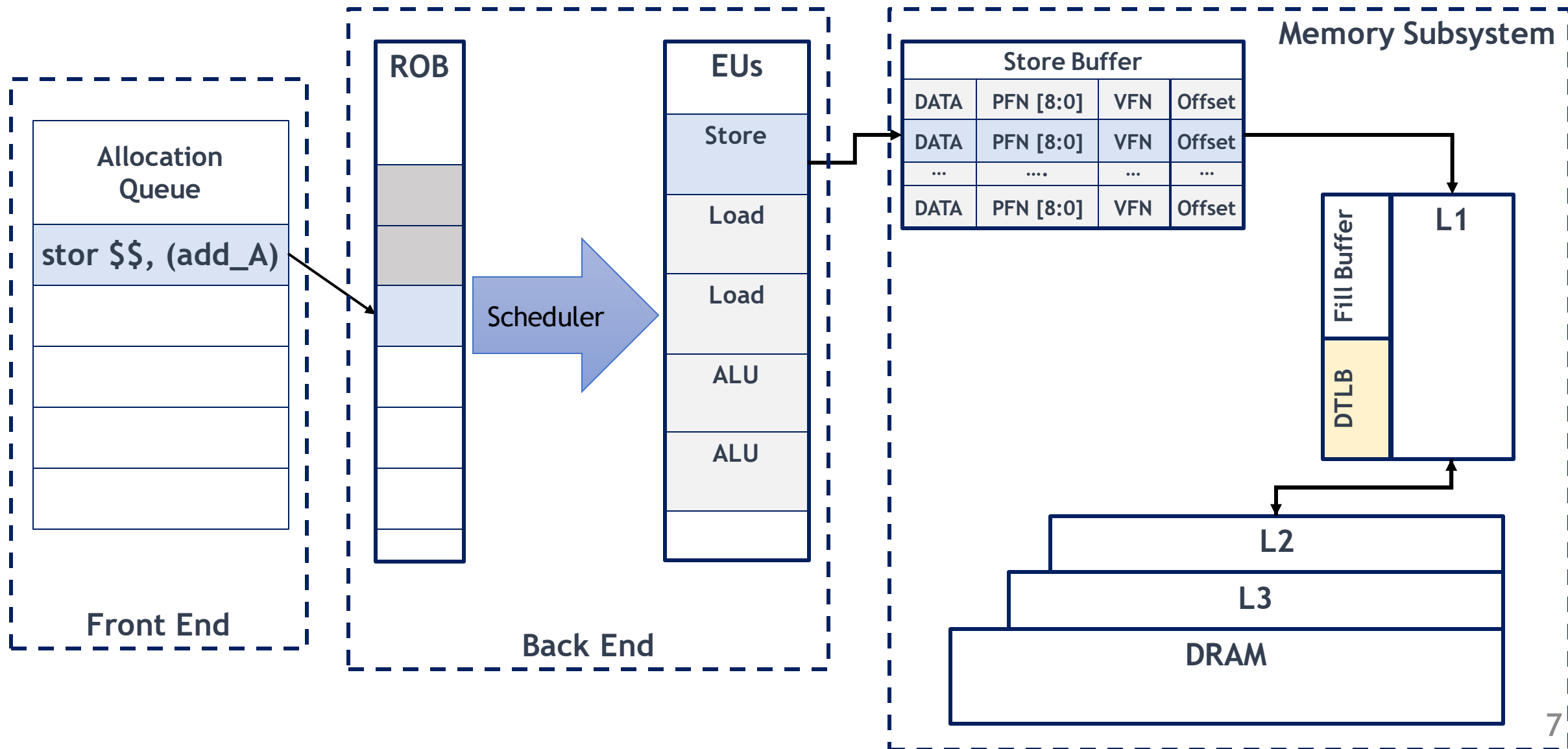
CPU Memory Subsystem



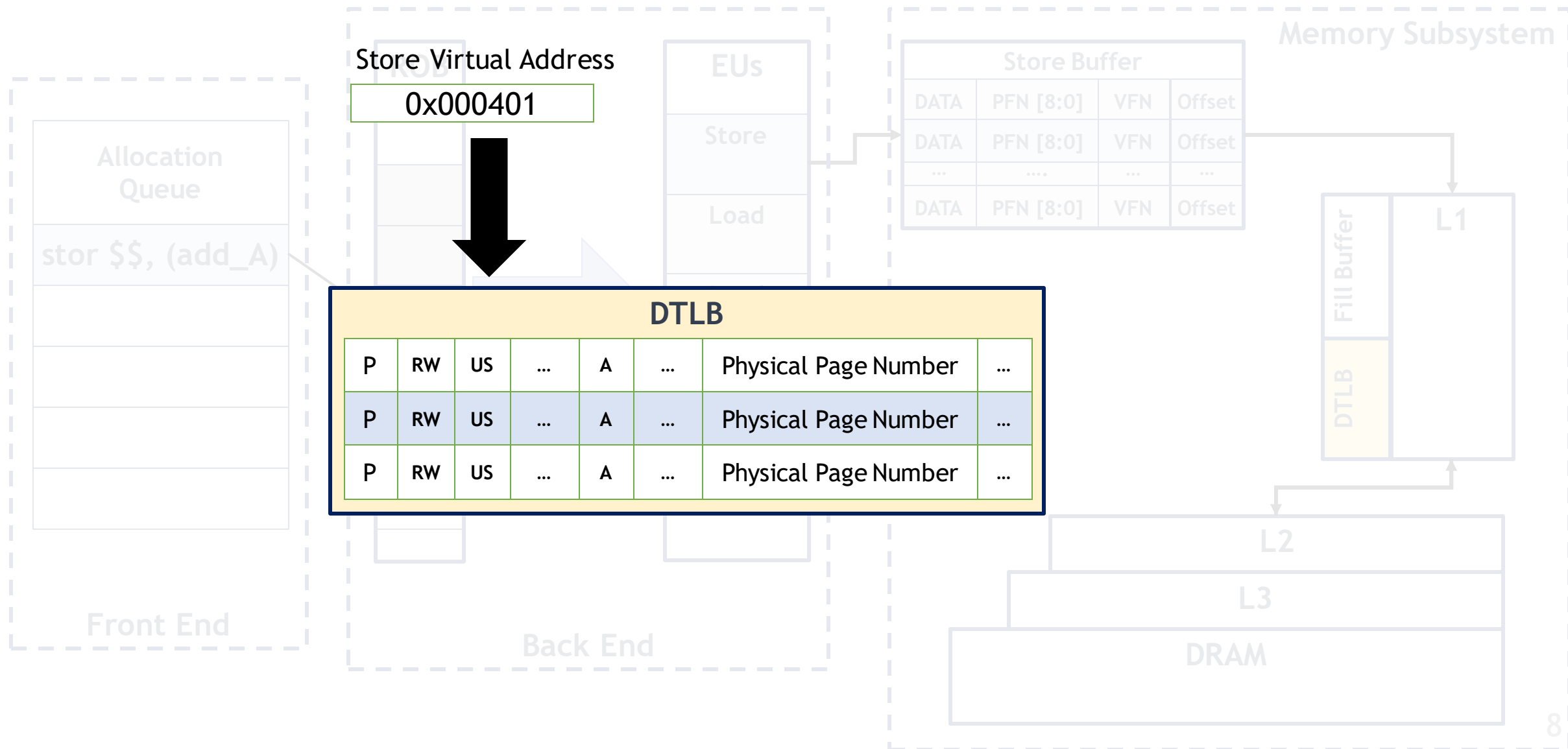
CPU Memory Subsystem



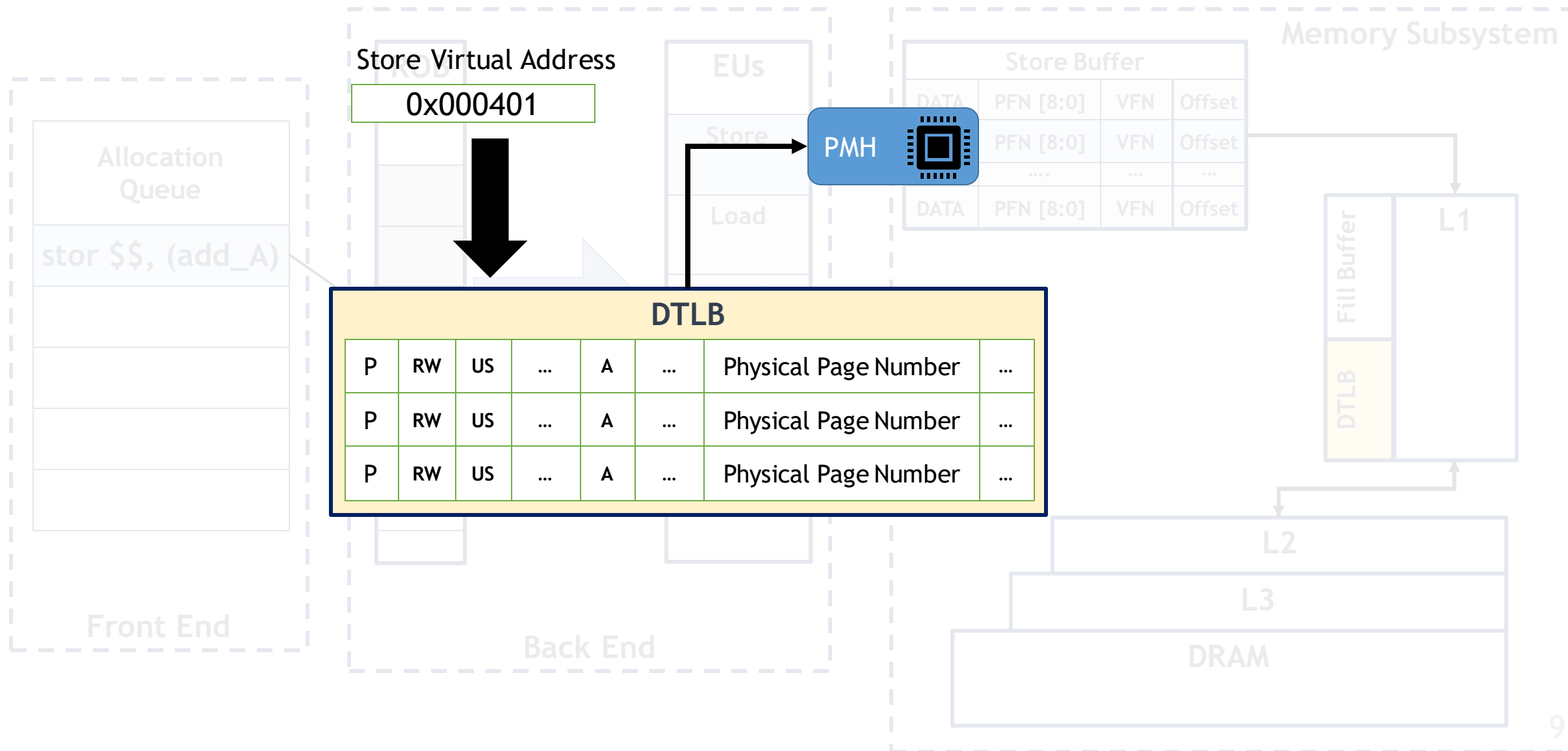
CPU Memory Subsystem



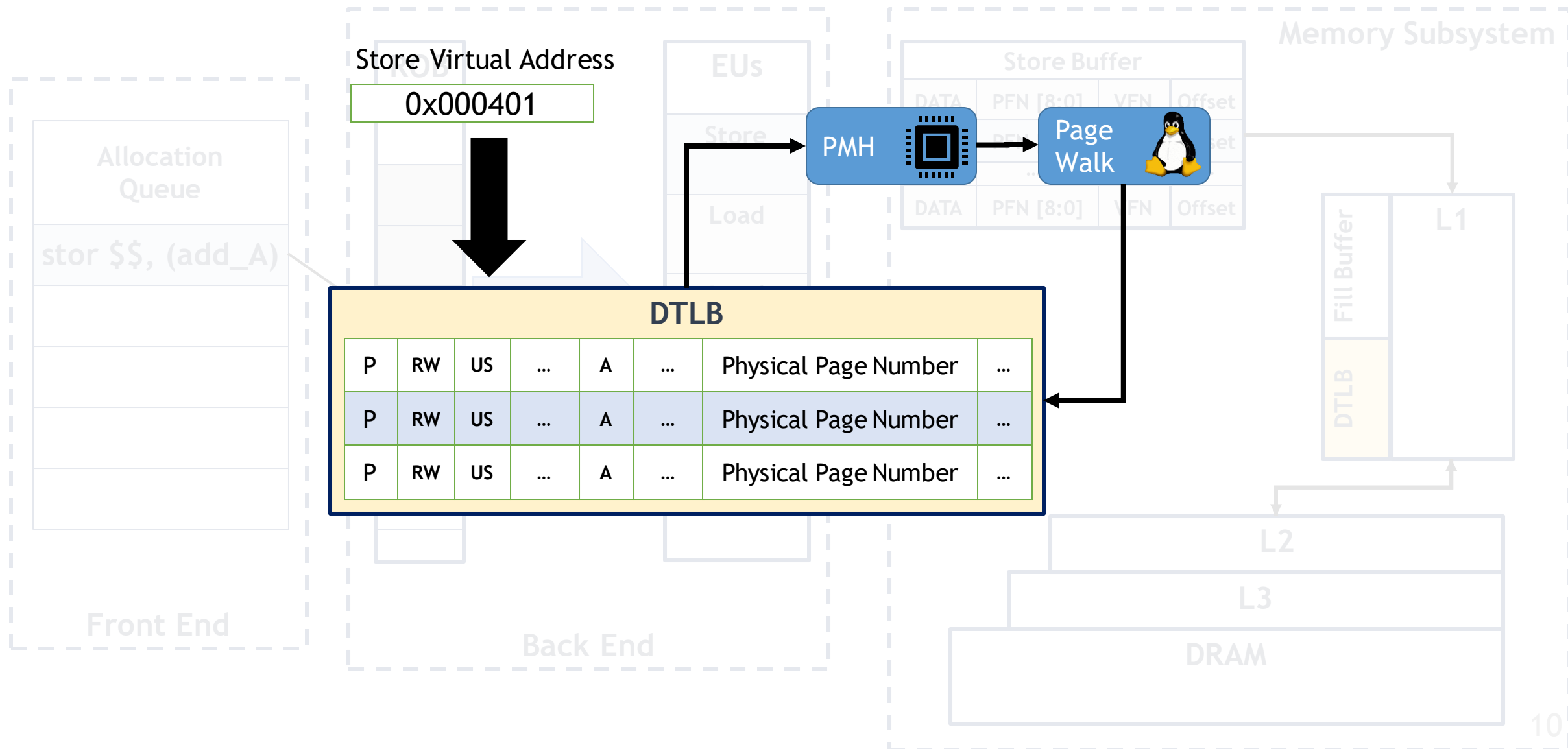
CPU Memory Subsystem



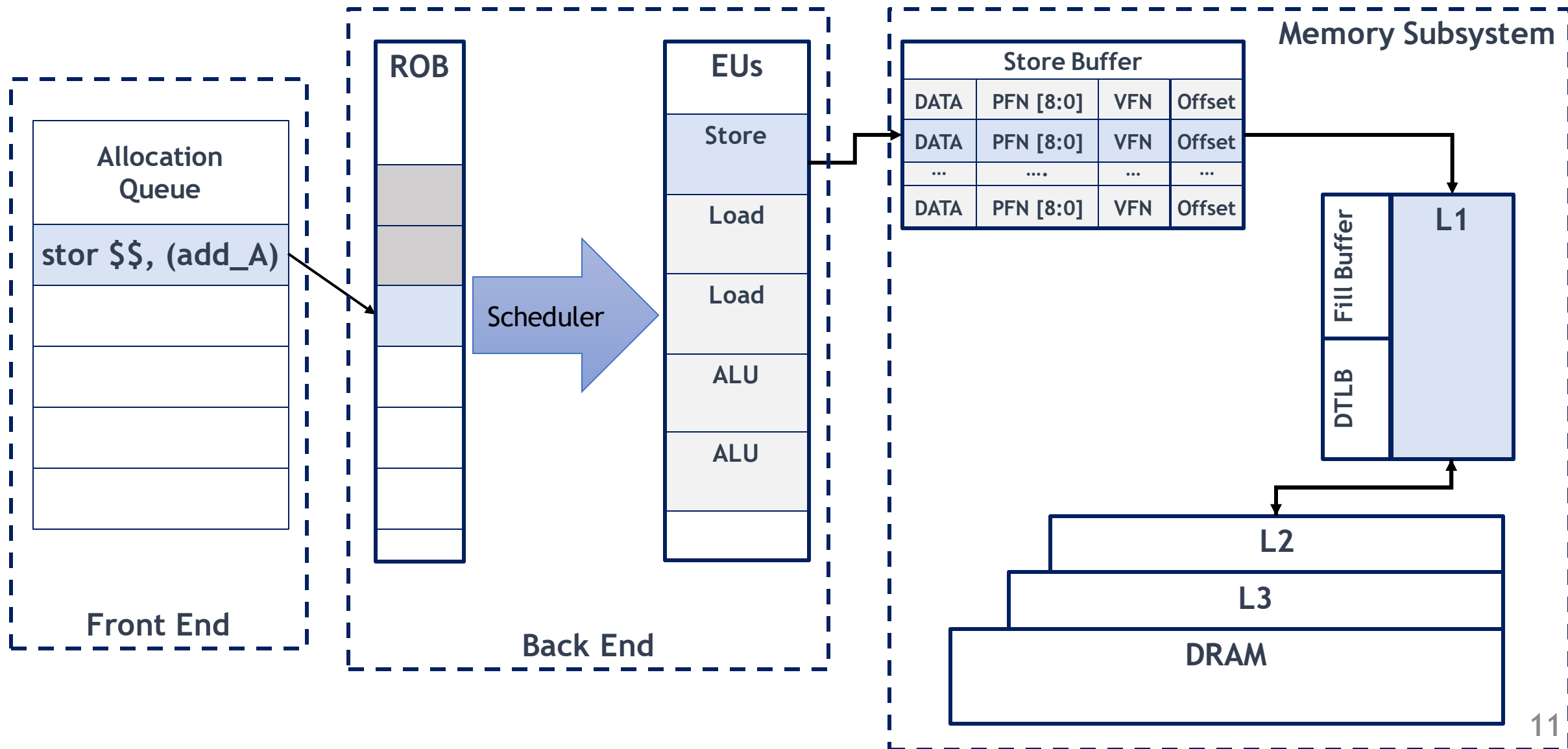
CPU Memory Subsystem



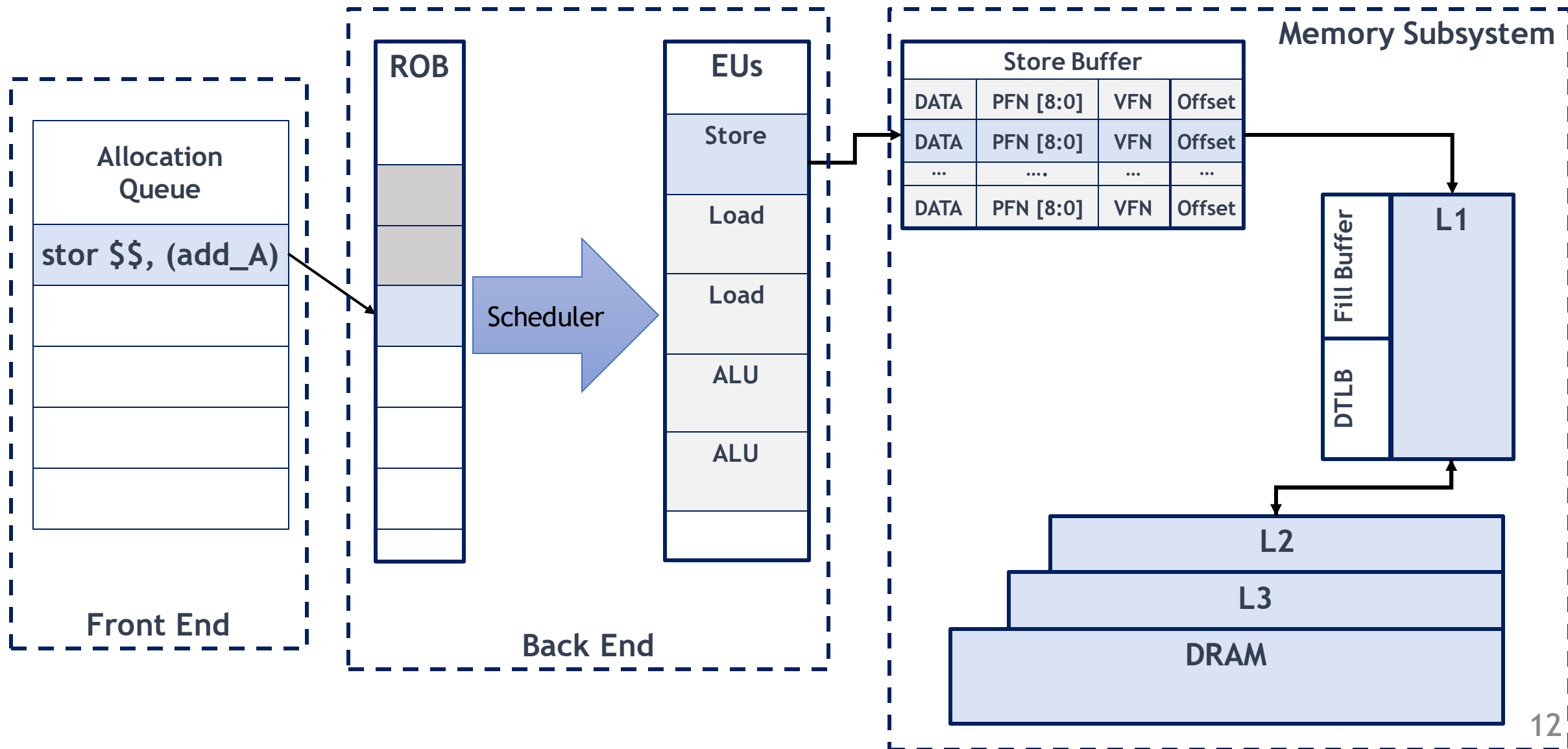
CPU Memory Subsystem



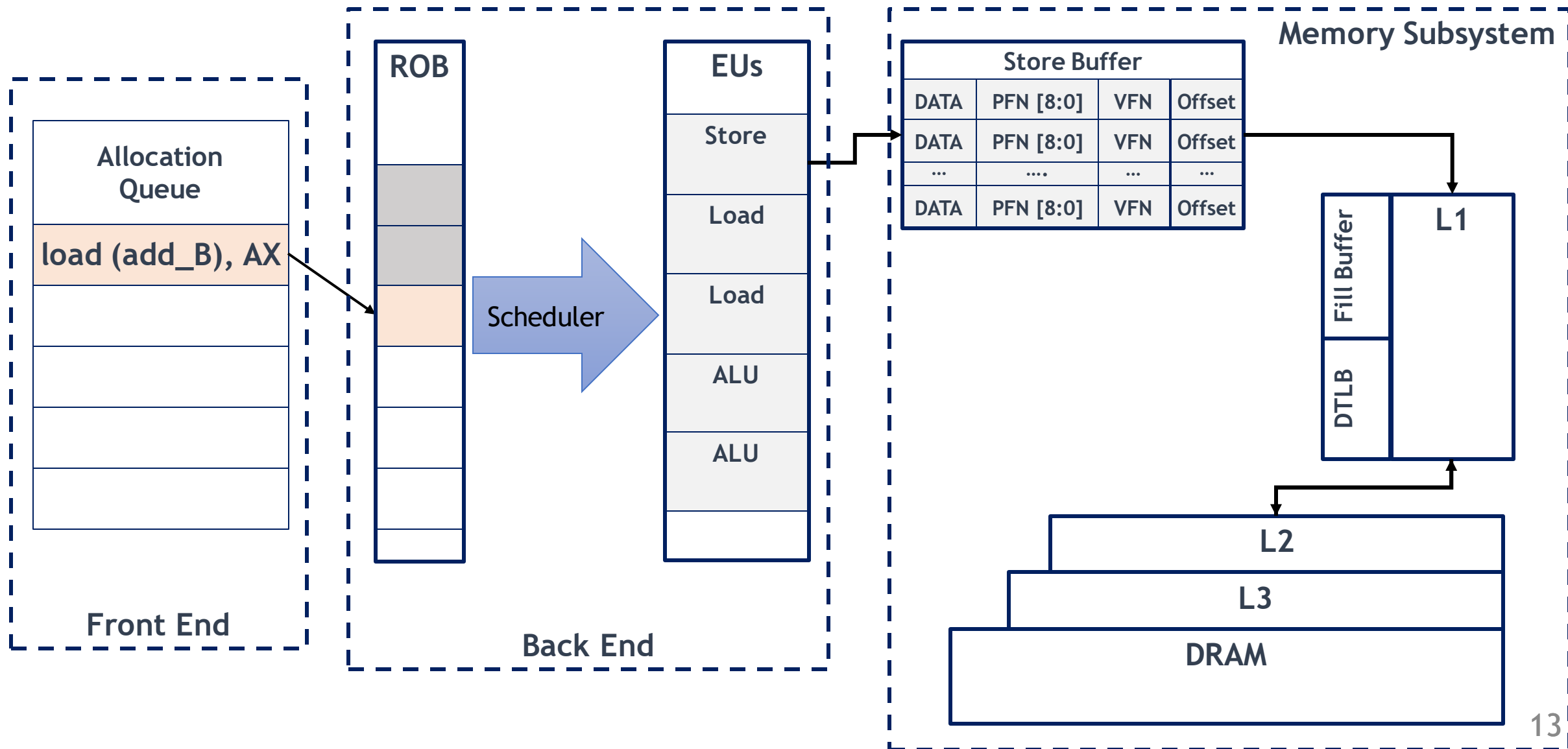
CPU Memory Subsystem



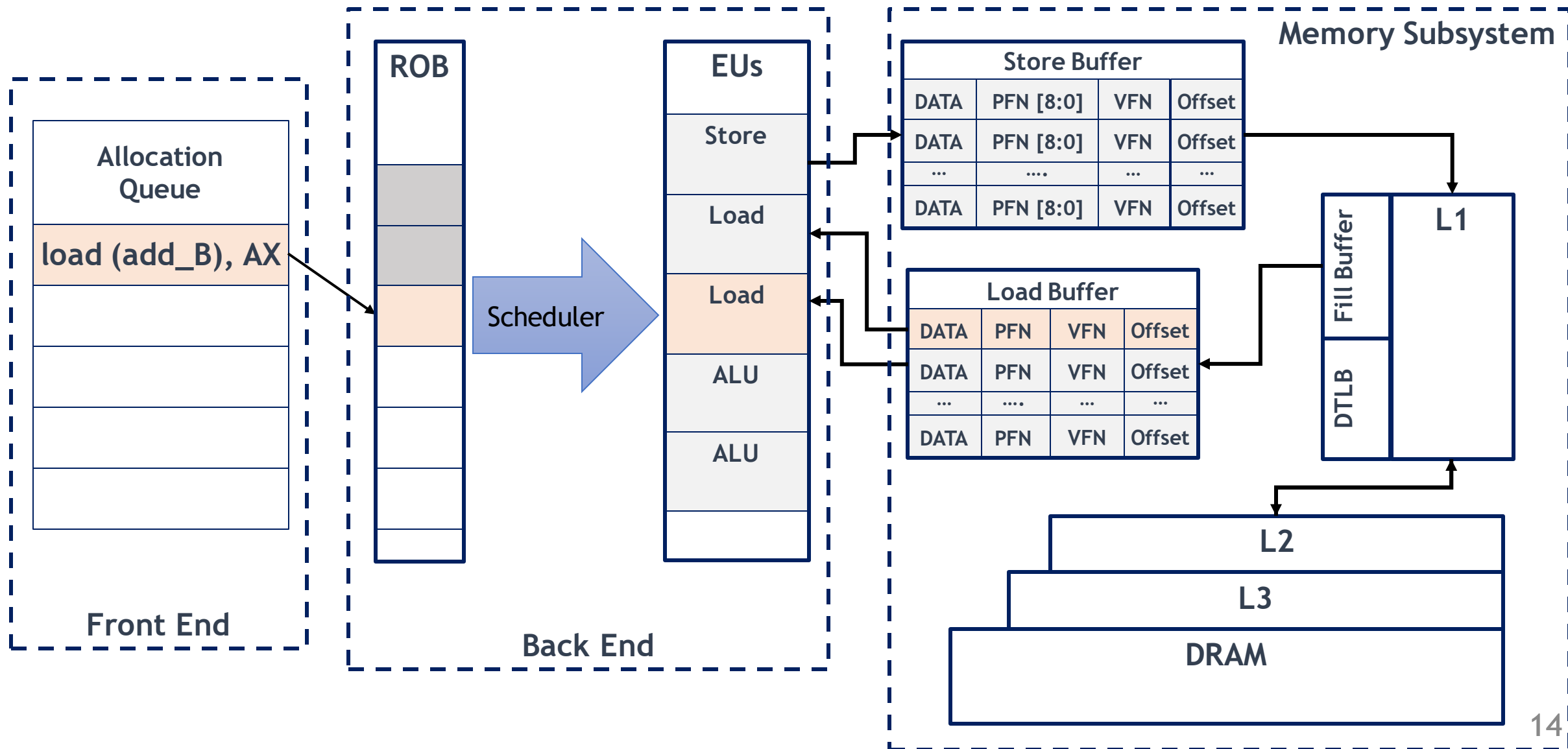
CPU Memory Subsystem



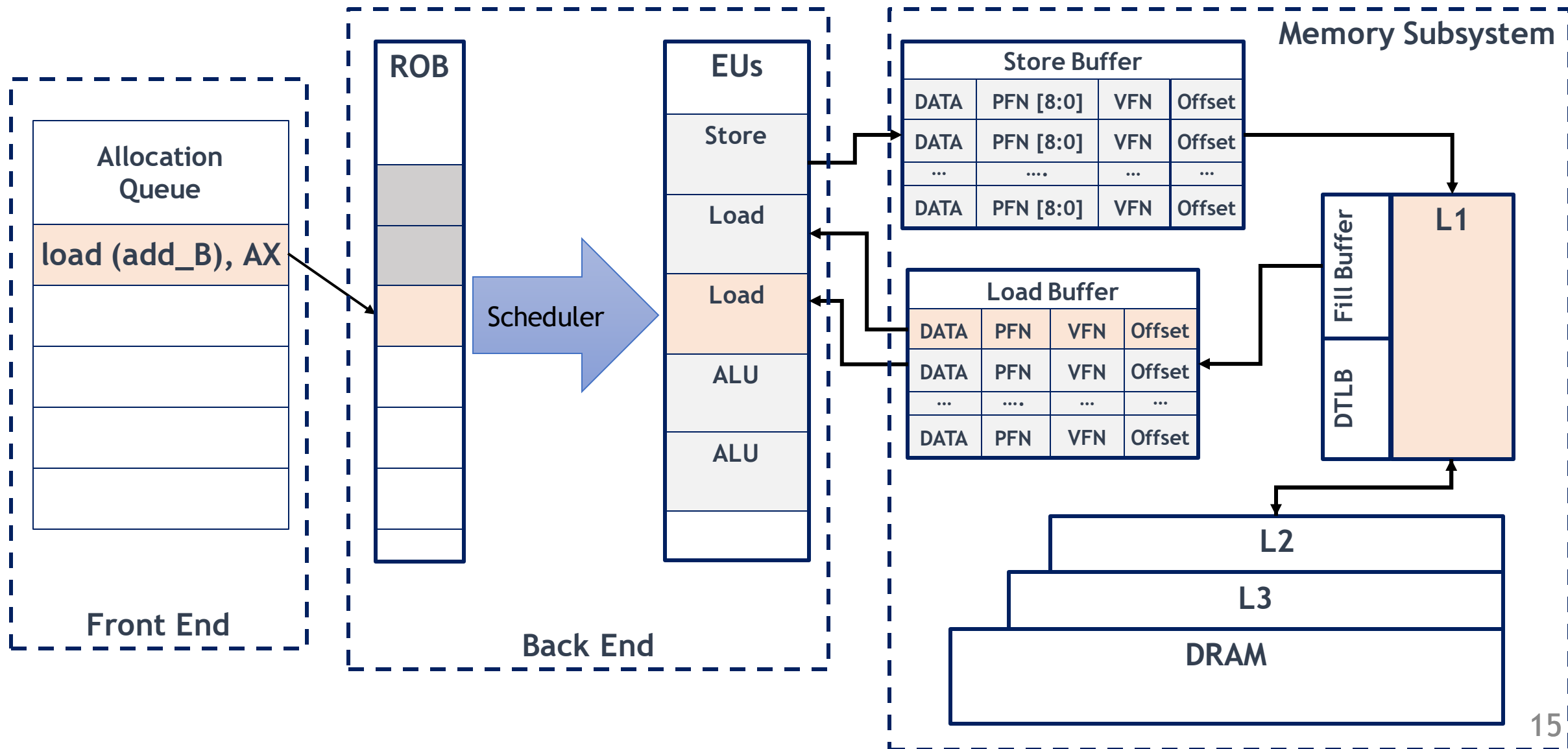
CPU Memory Subsystem



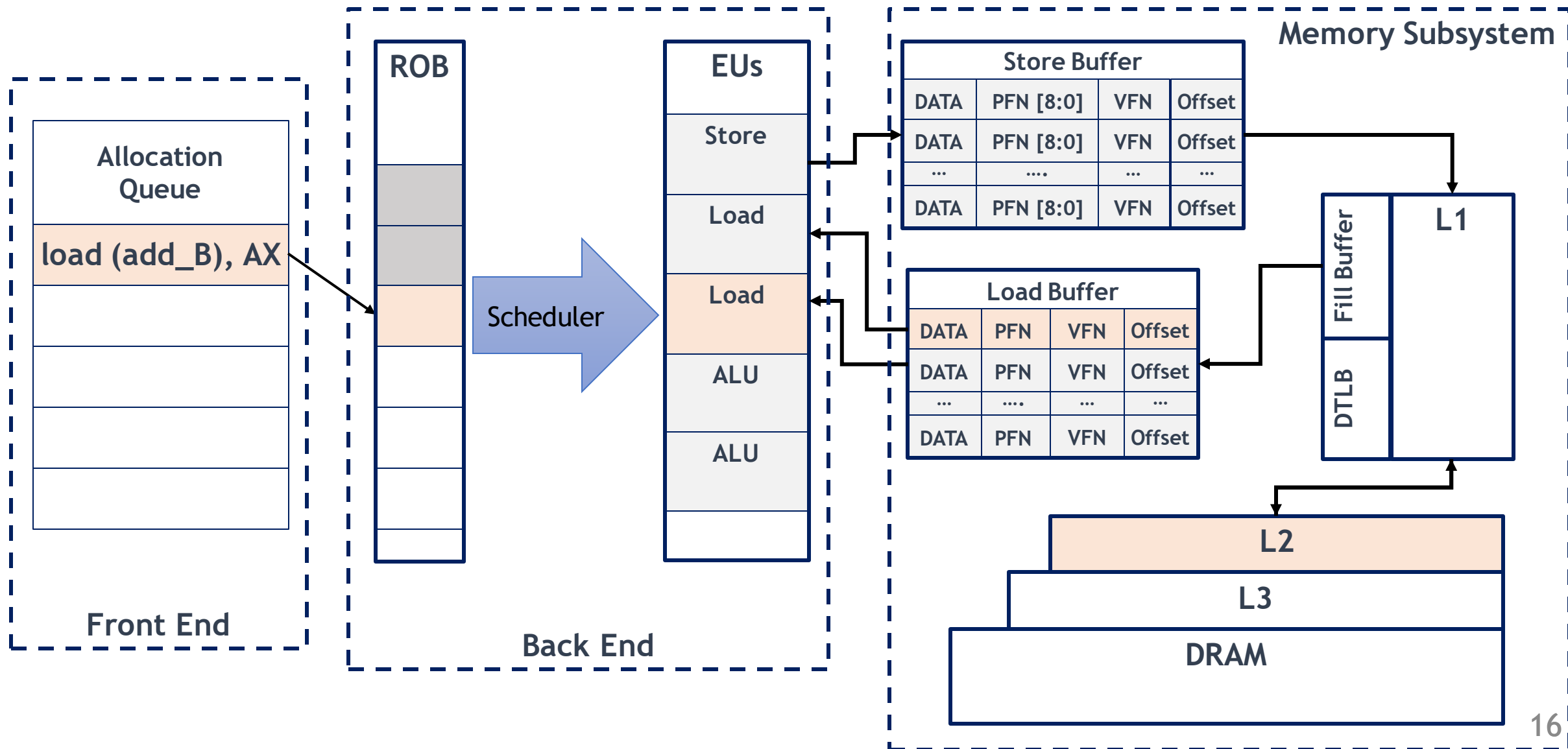
CPU Memory Subsystem



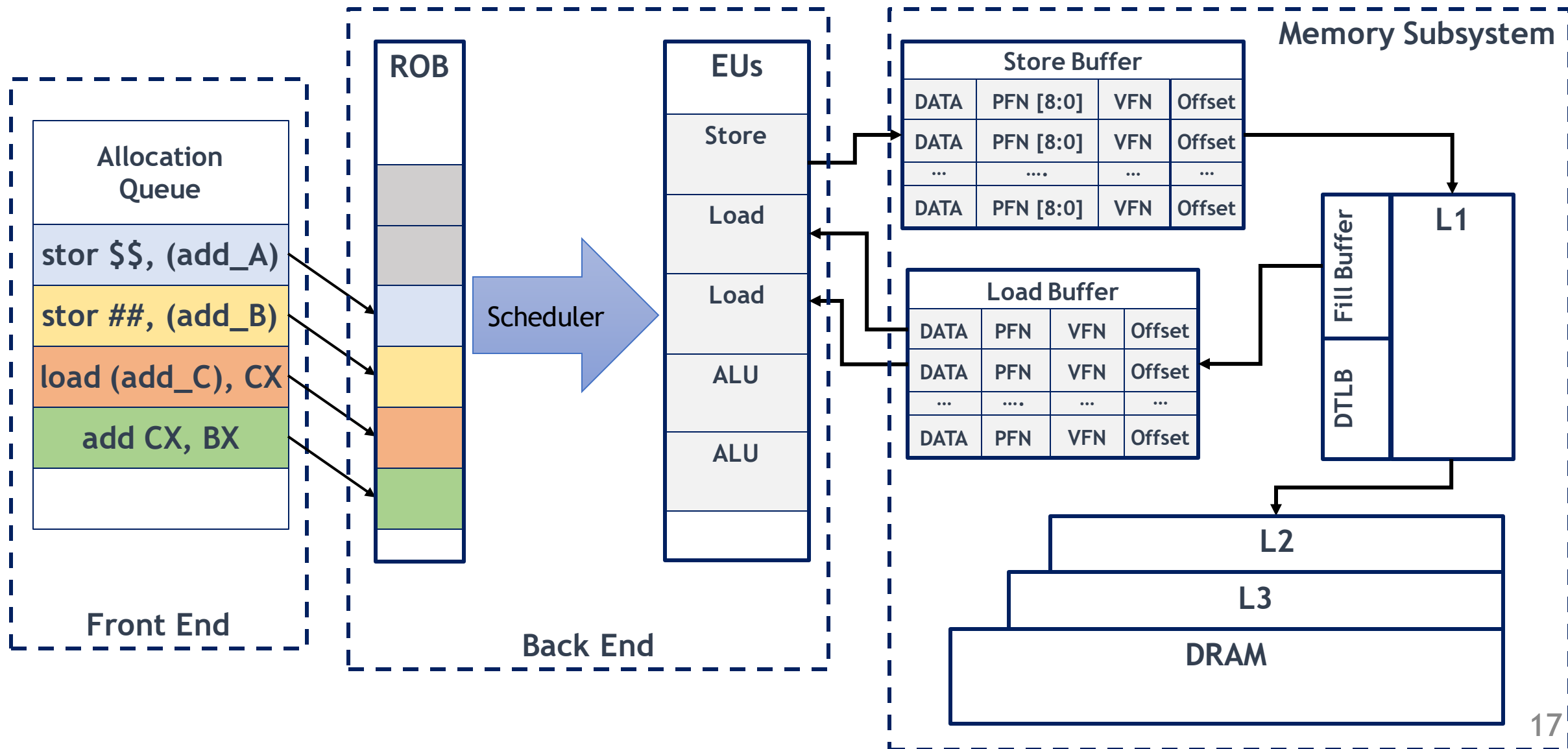
CPU Memory Subsystem



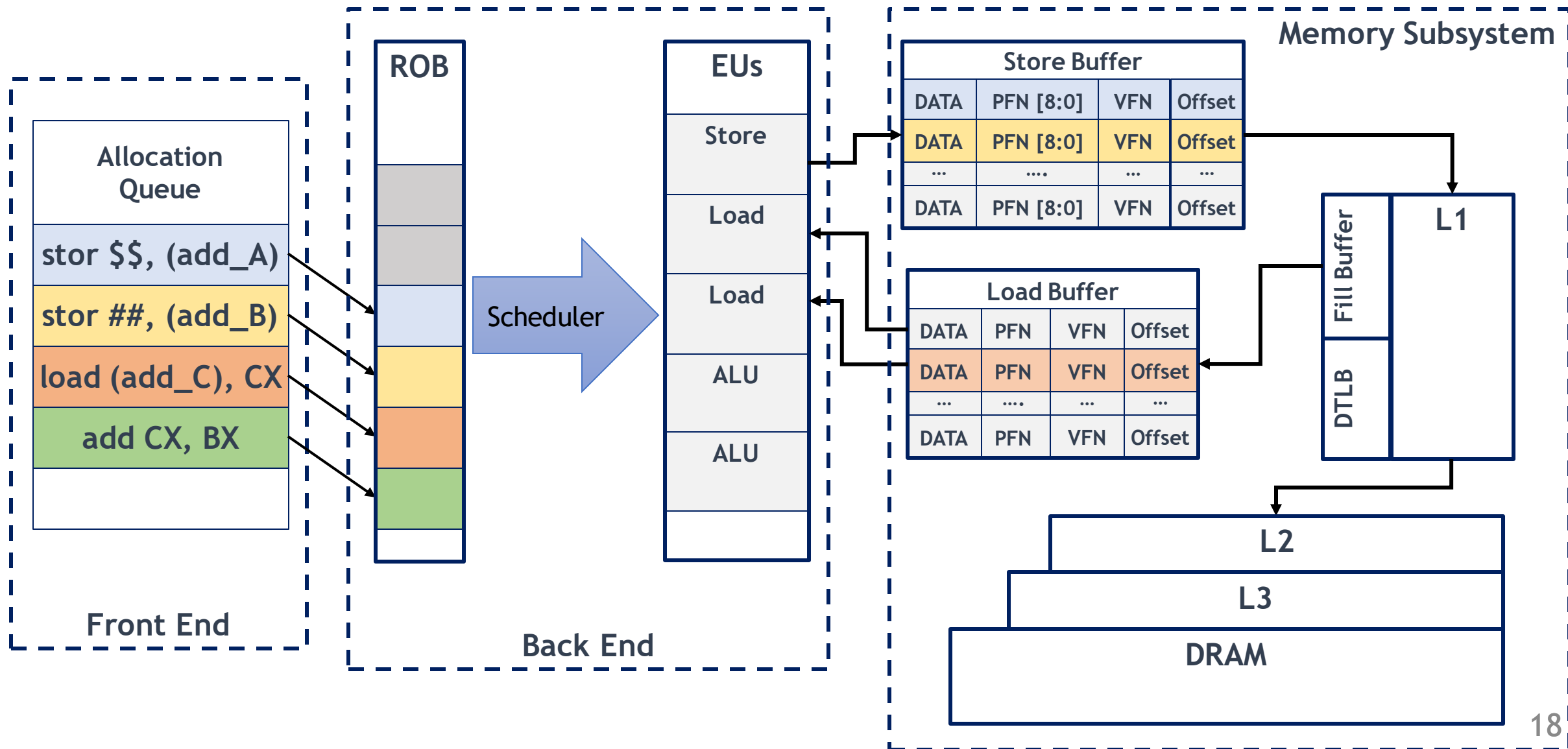
CPU Memory Subsystem



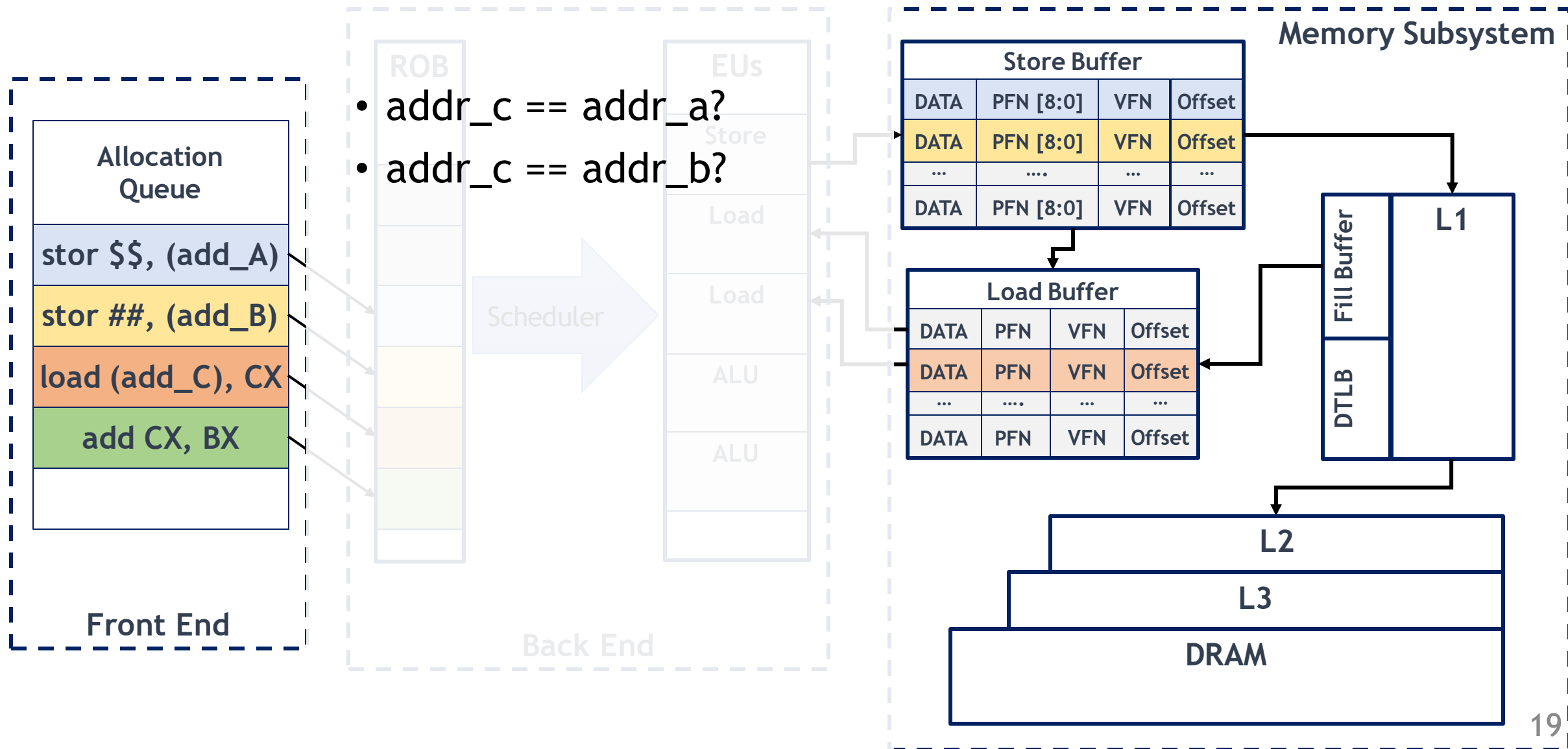
CPU Memory Subsystem



CPU Memory Subsystem - Store Forwarding



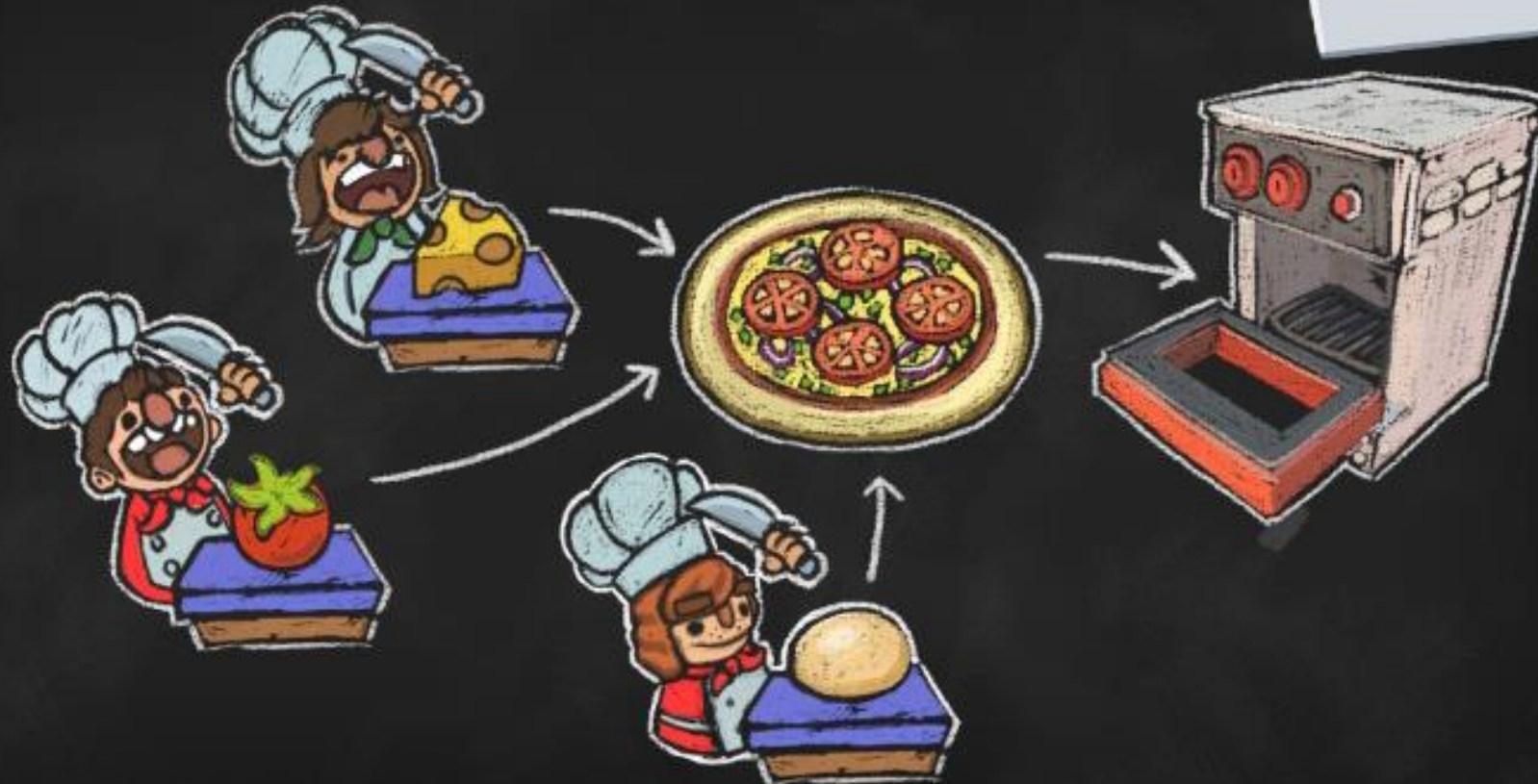
CPU Memory Subsystem - Store Forwarding





PIZZA

**NEW
RECIPE**



**PREPARE DOUGH WITH KNIFE AND CHOP CHEESE, TOMATO AND POSSIBLY
SOME PEPPERONI OR CHICKEN!**



Professor



Alex



Tom

1. Pepperoni
2. Chicken
3. Pepperoni



Professor

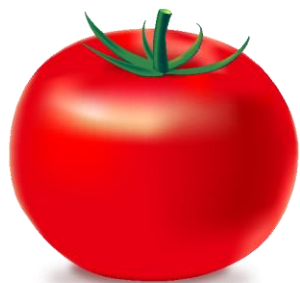


Alex

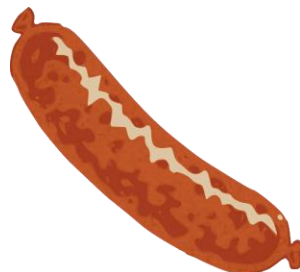


Tom

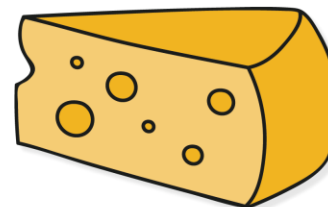
1. Pepperoni
2. Chicken
3. Pepperoni



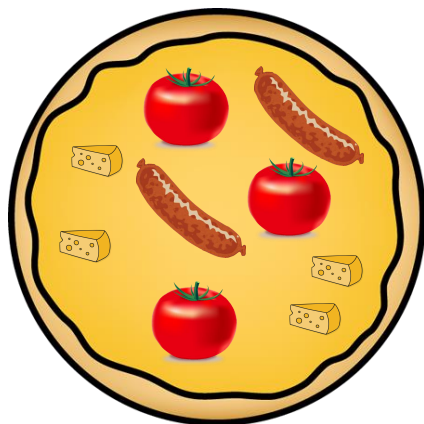
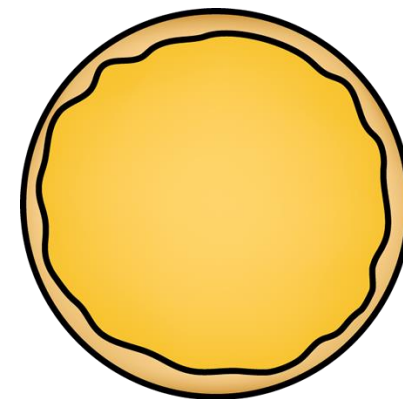
Cut
Tomato



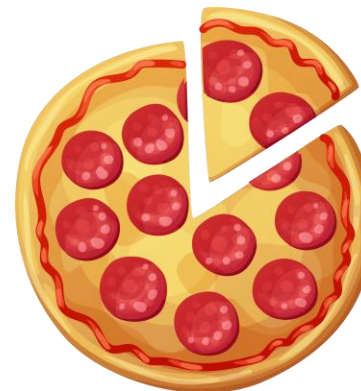
Cut and
Precook



Grind
Chees



Cook



Deliver



Professor

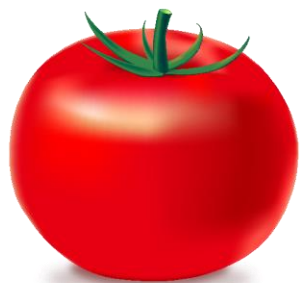


Alex

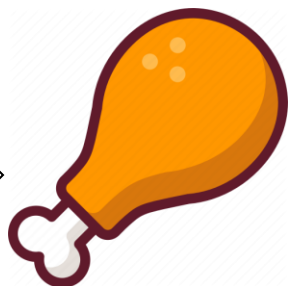


Tom

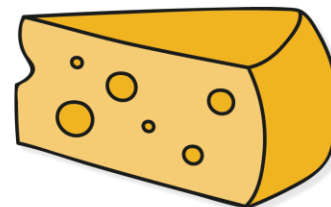
1. Pepperoni
2. Chicken
3. Pepperoni



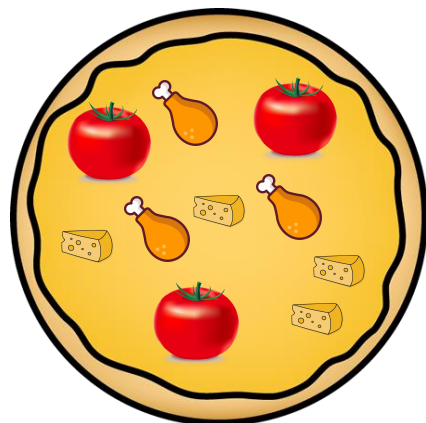
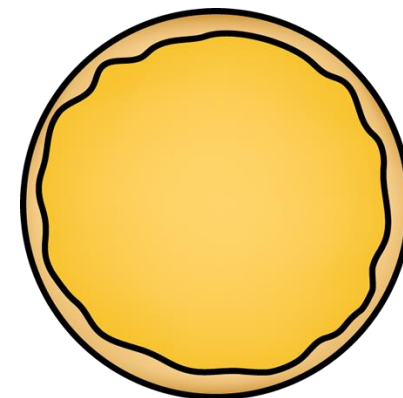
Cut
Tomato



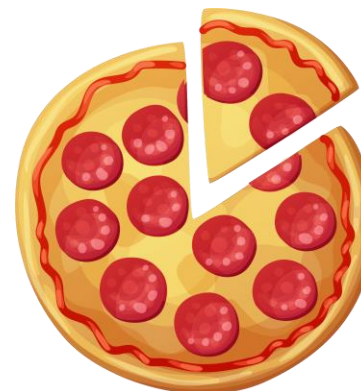
Cut and
Precook



Grind
Chees



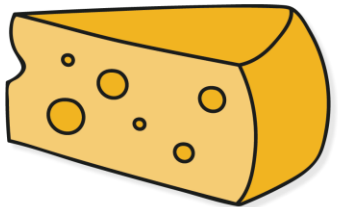
Cook



Deliver



1. Pepperoni
2. Chicken
3. Pepperoni





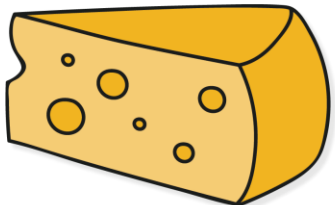
1. Pepperoni
2. Chicken
3. Pepperoni



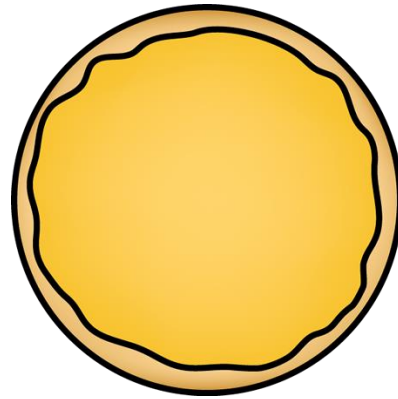
Cut



Cut



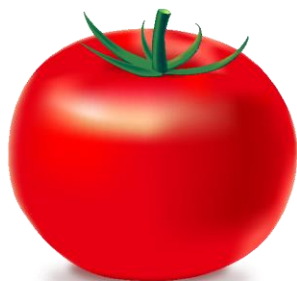
Grind



Precook
and mix



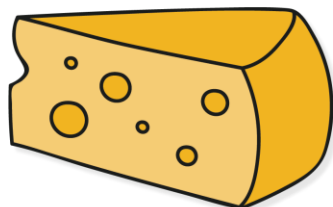
1. Pepperoni
2. Chicken
3. Pepperoni



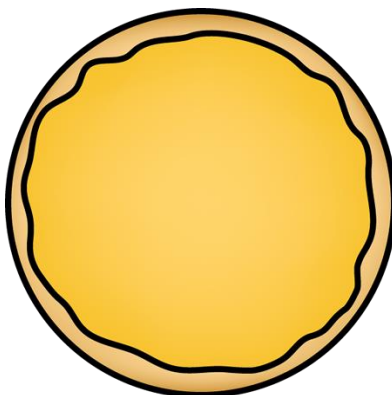
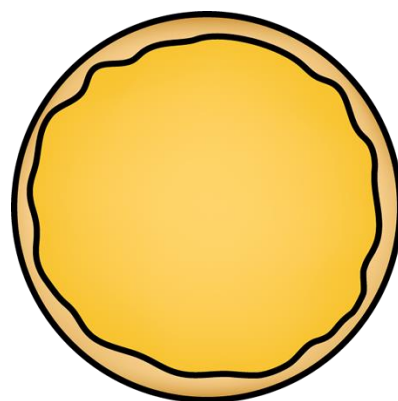
Cut



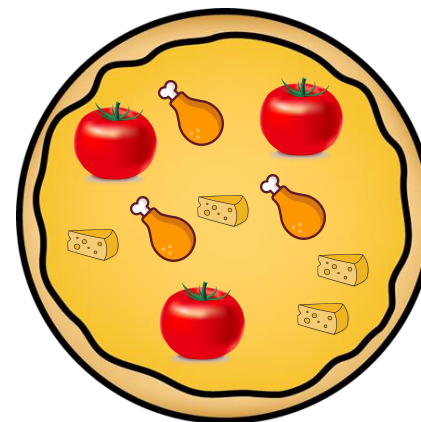
Cut



Grind



Precook
and mix





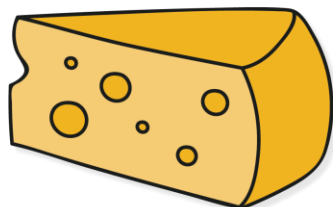
1. Pepperoni
2. Chicken
3. Pepperoni



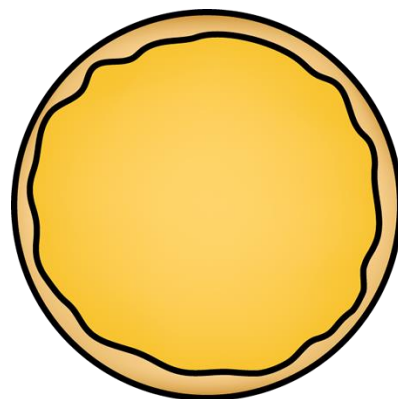
Cut



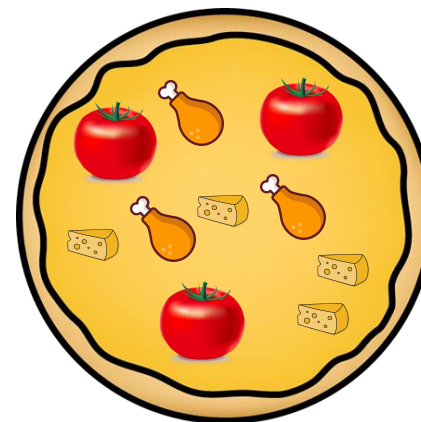
Cut



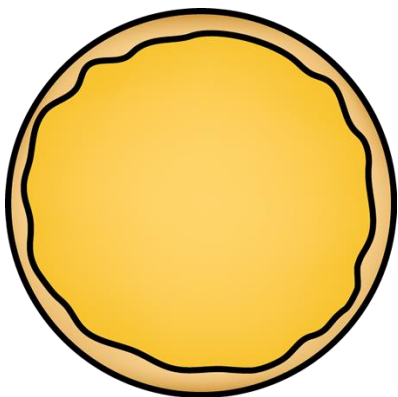
Grind



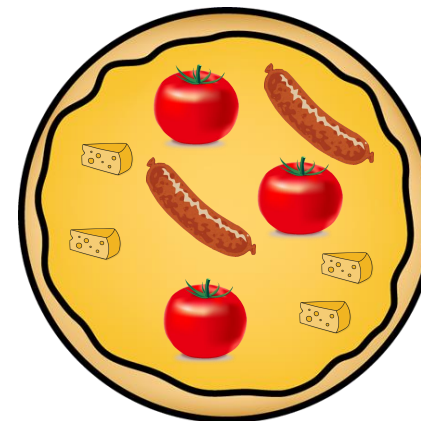
Precook
and mix



Cook and
deliver



Precook
and mix



Cook and
deliver

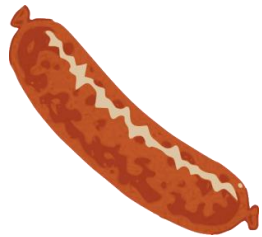


1. ~~Pepperoni~~

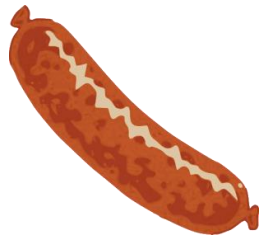
2. ~~Chicken~~

3. Pepperoni

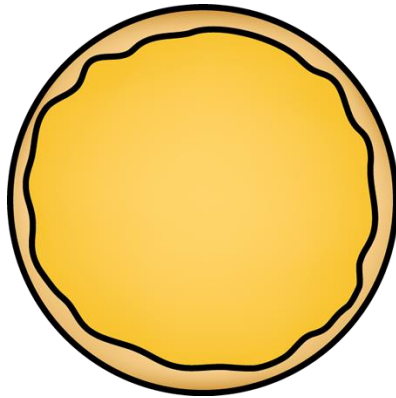
4. ???



Speculative Cut

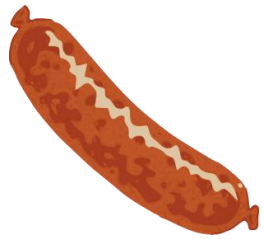


Speculative Cut

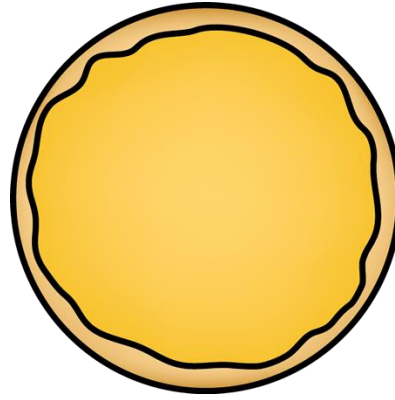




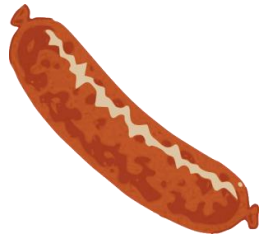
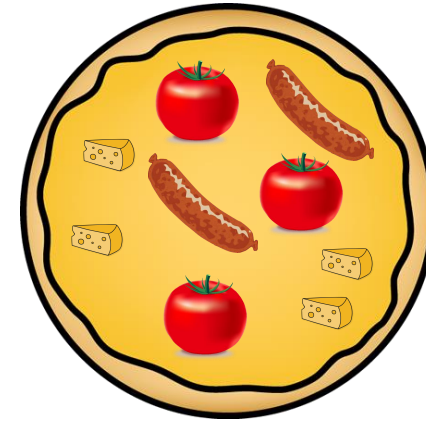
1. ~~Pepperoni~~
2. ~~Chicken~~
3. Pepperoni
4. Chicken



Speculative Cut



Precook
and mix

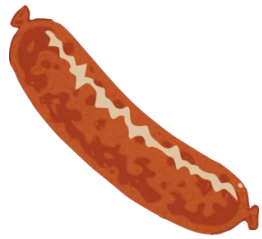


Speculative Cut

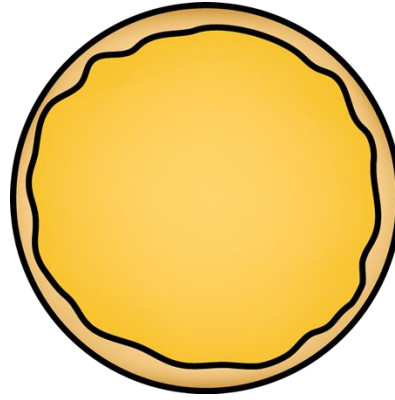




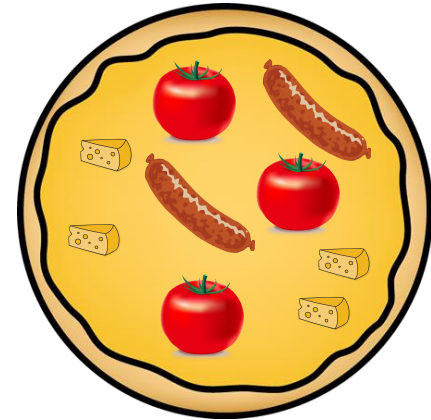
1. ~~Pepperoni~~
2. ~~Chicken~~
3. Pepperoni
4. Chicken



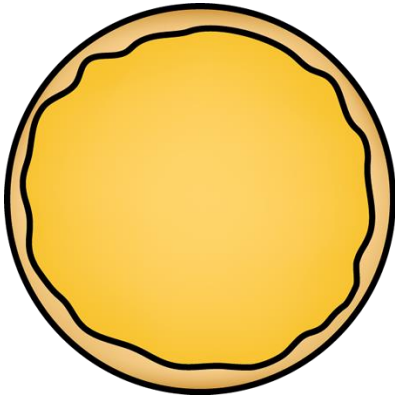
Speculative Cut



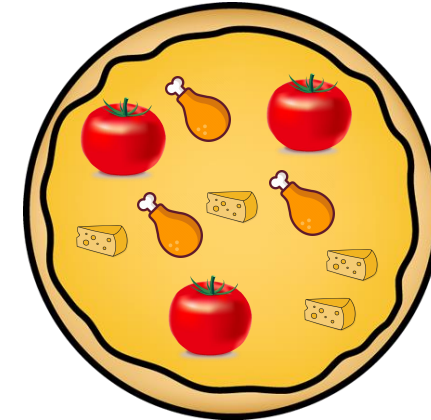
Precook
and mix

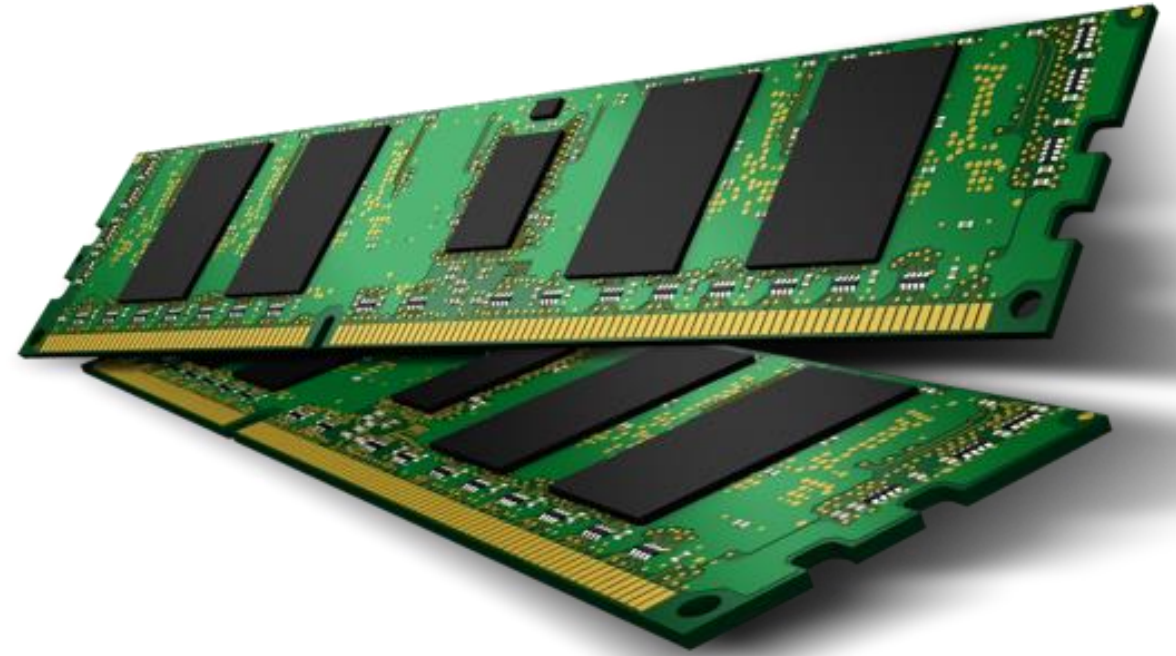


Speculative Cut



Precook
and mix





MemJam

MemJam Attack

- Memory loads/stores are executed out of order and speculatively.

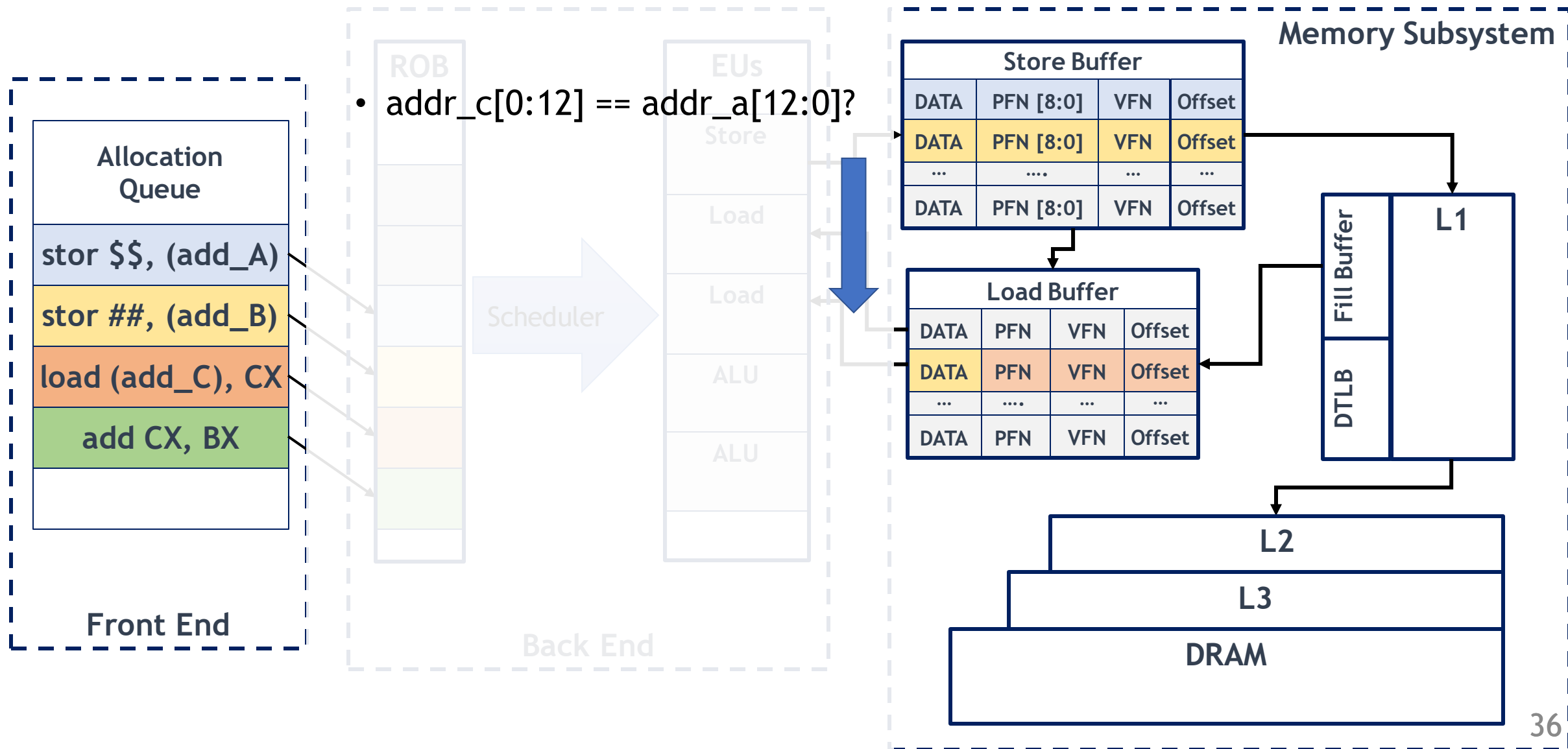
MemJam Attack

- Memory loads/stores are executed out of order and speculatively.
- Address translation can be expensive.

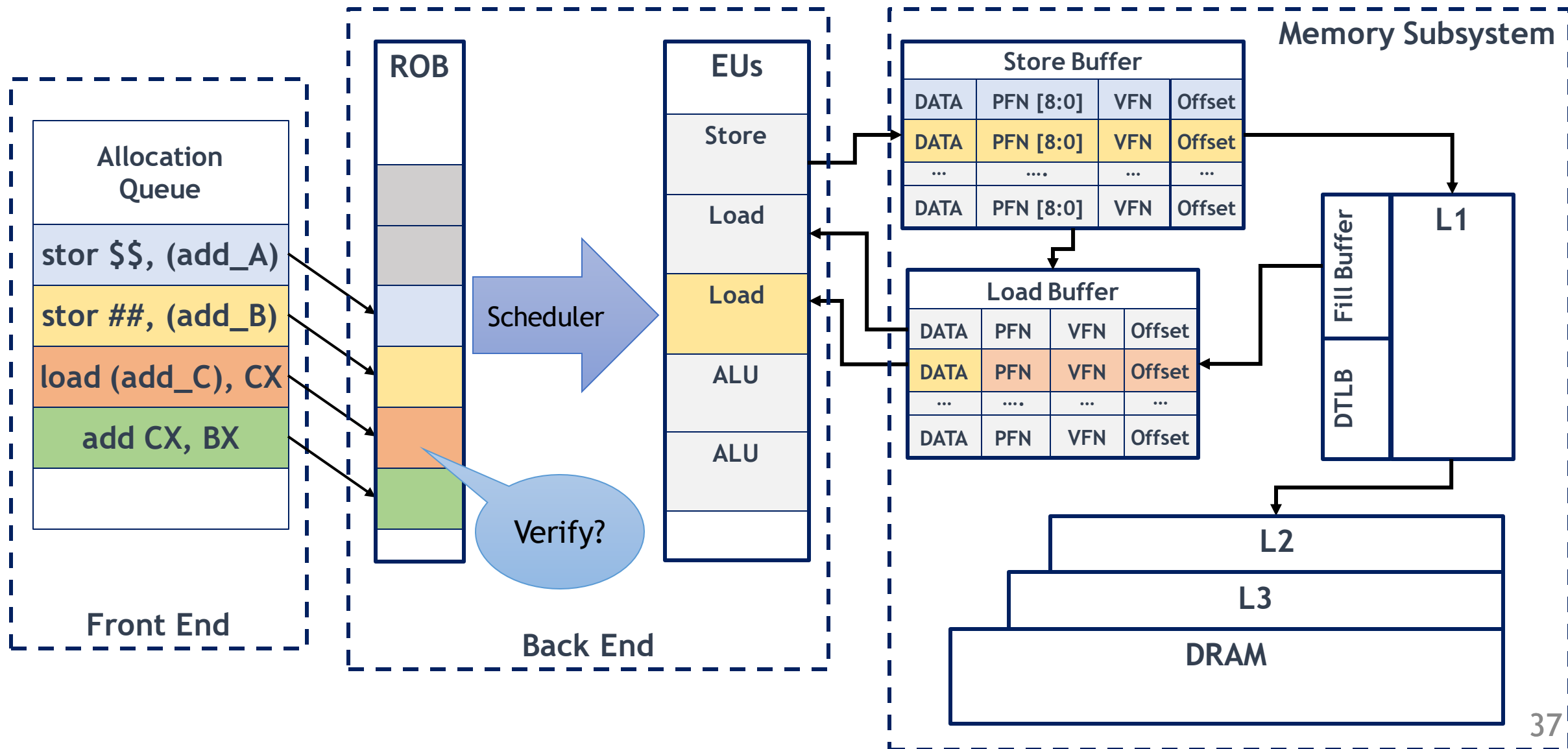
MemJam Attack

- Memory loads/stores are executed out of order and speculatively.
- Address translation can be expensive.
- 4K Aliasing: Addresses that are 4K apart are assumed dependent.

CPU Memory Subsystem - Store Forwarding



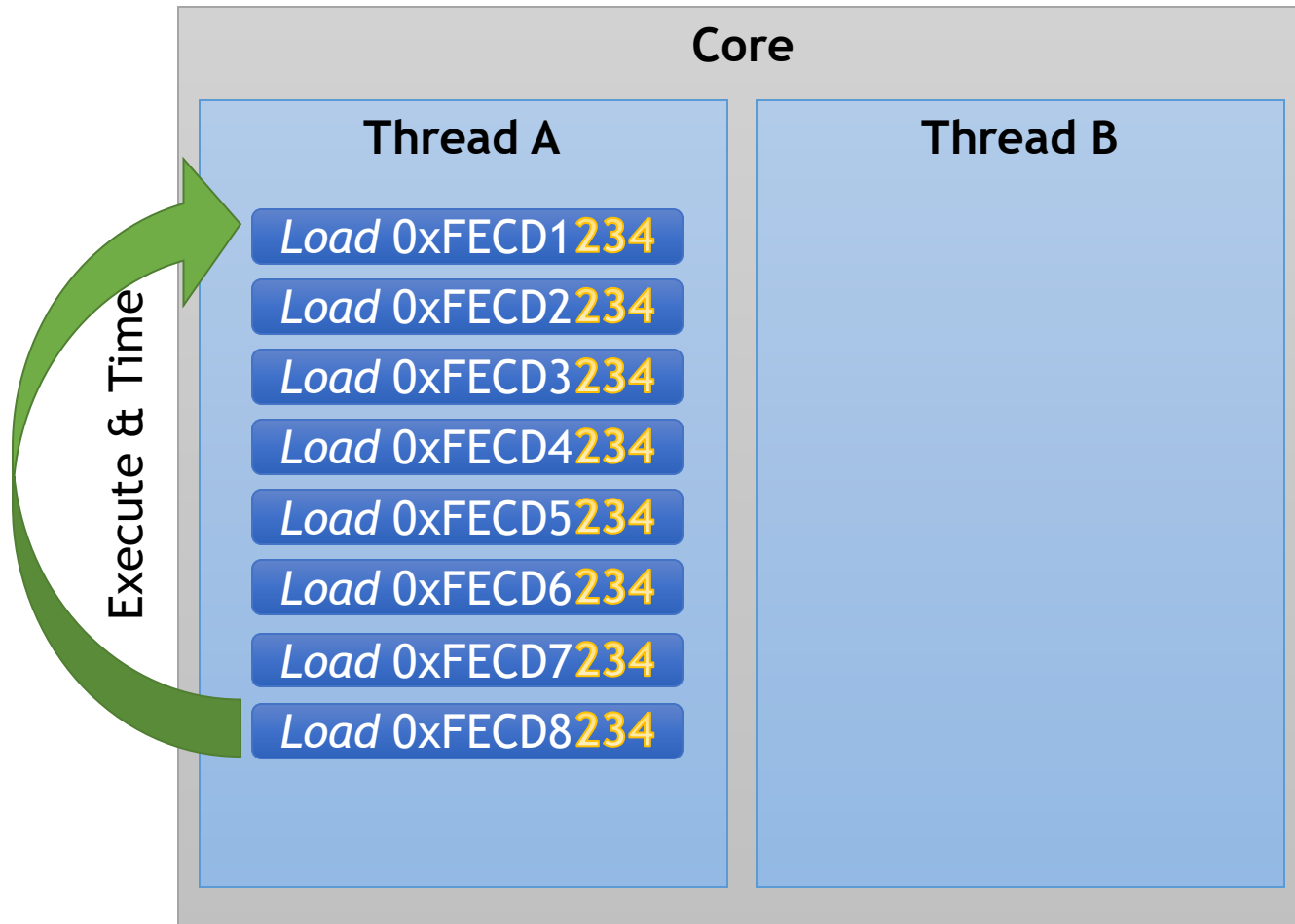
CPU Memory Subsystem - Store Forwarding



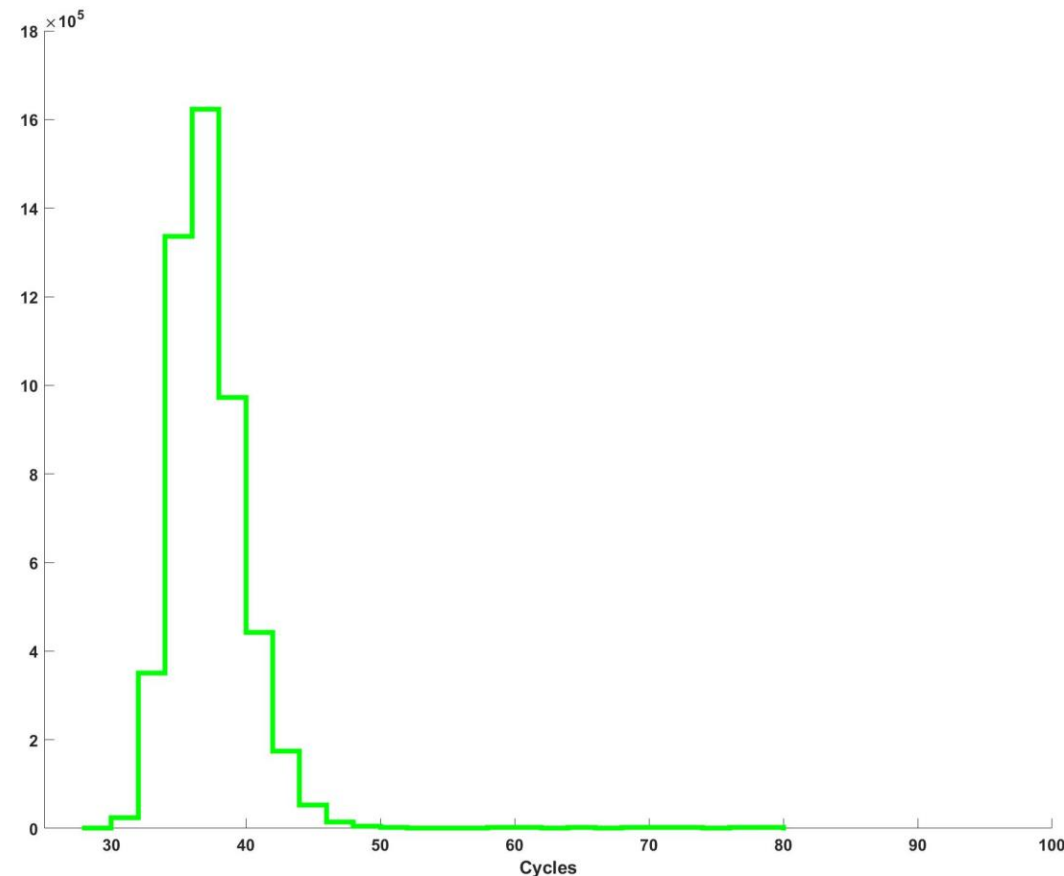
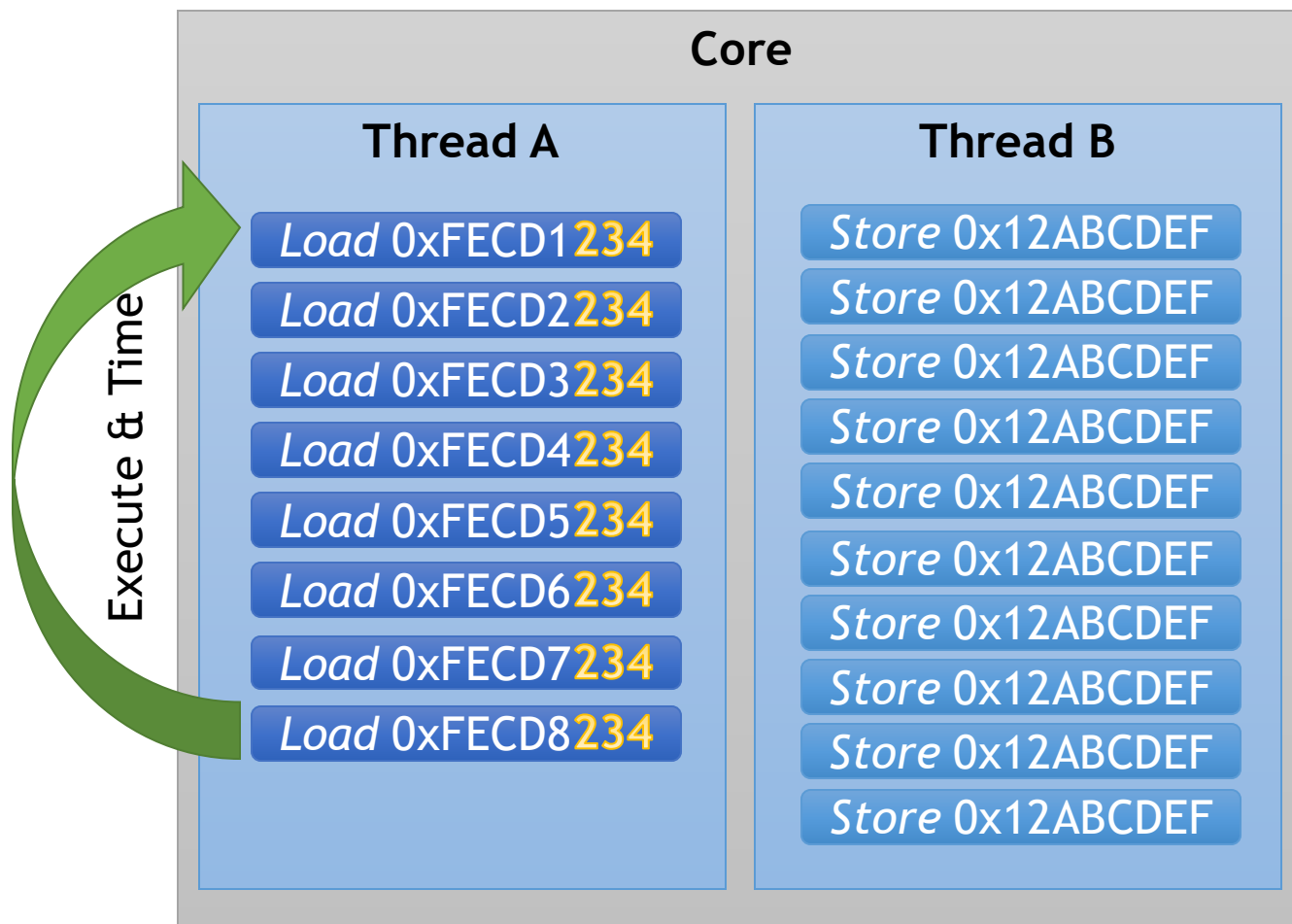
MemJam Attack

- Memory loads/stores are executed out of order and speculatively.
- Address translation can be expensive.
- 4K Aliasing: Addresses that are 4K apart are assumed dependent.
- The dependency is verified after the execution!
- Re-execute the load block due to false dependency.

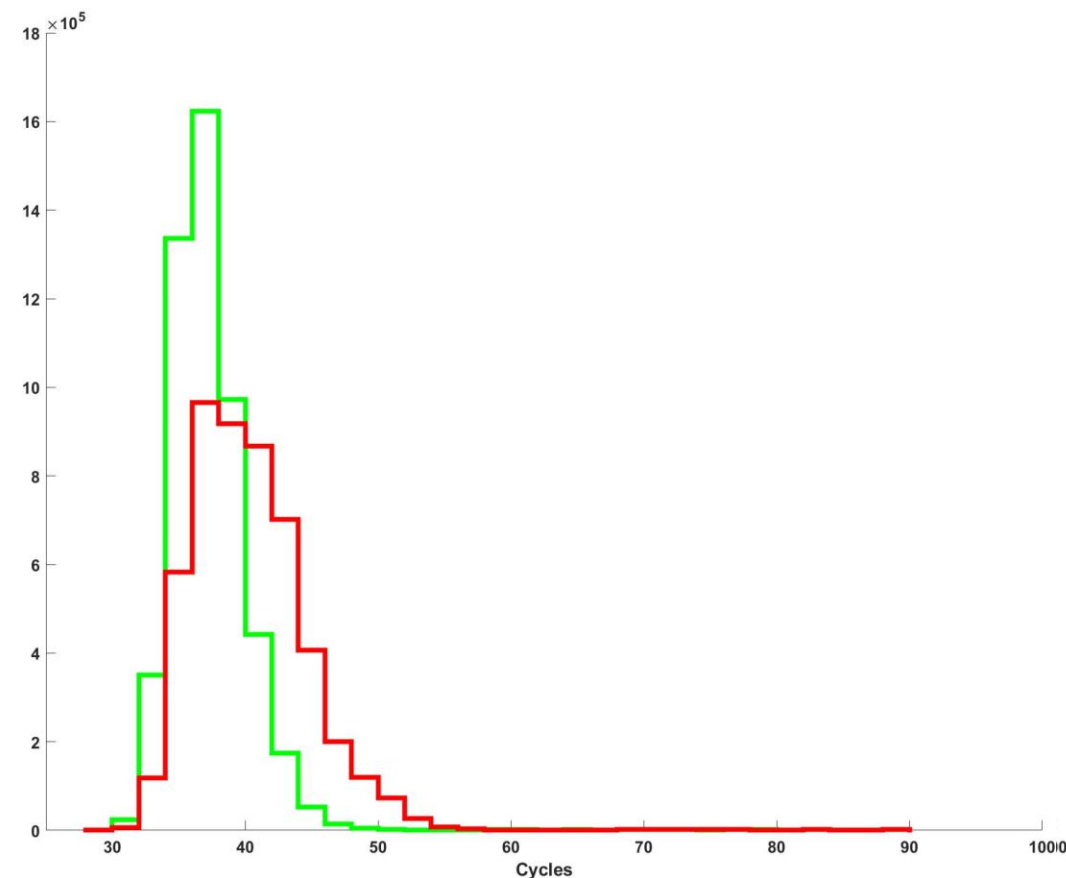
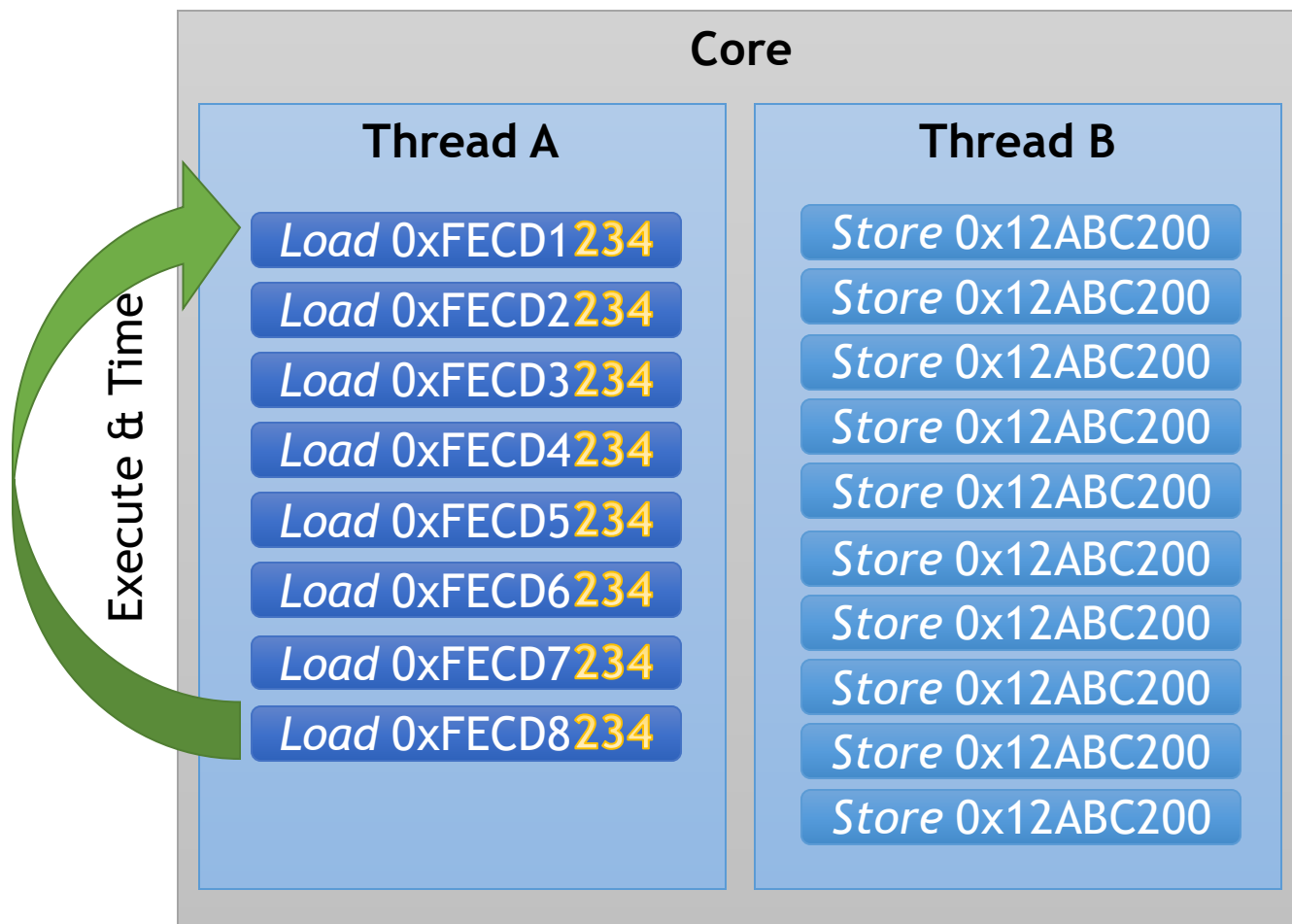
MemJam - 4K Aliasing across Sibling Threads



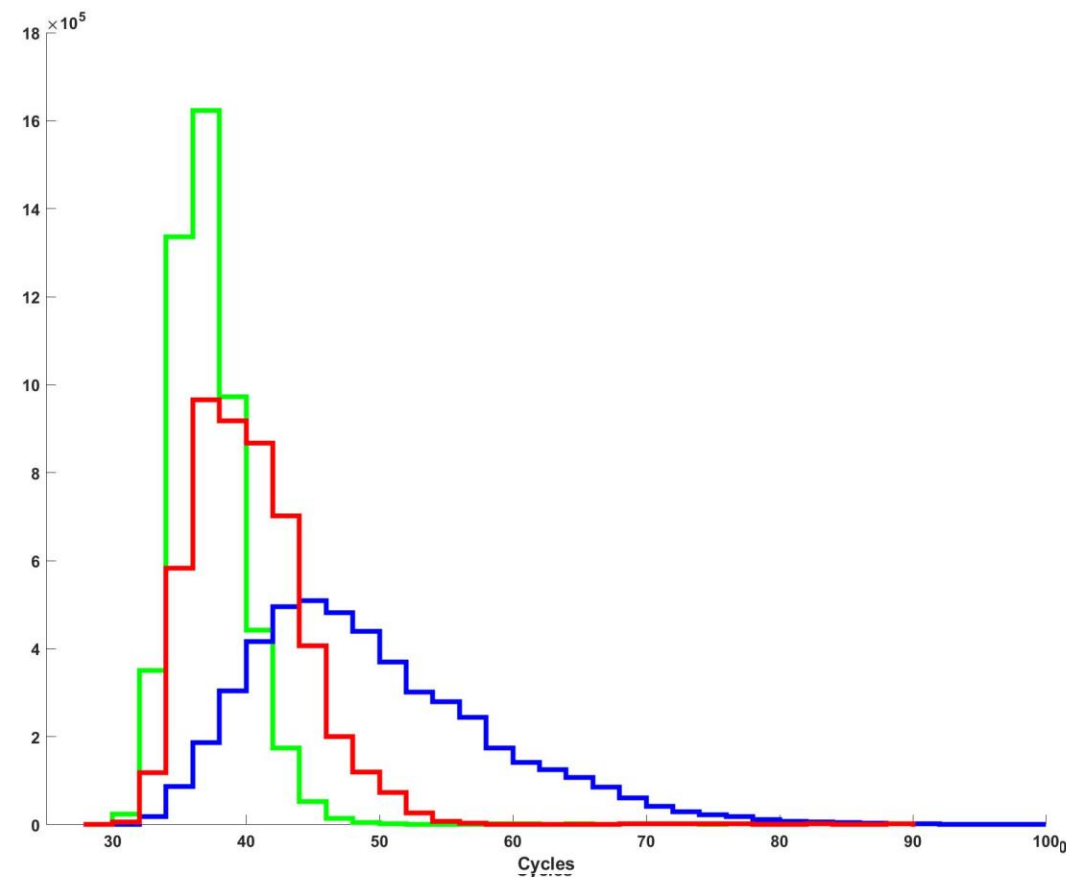
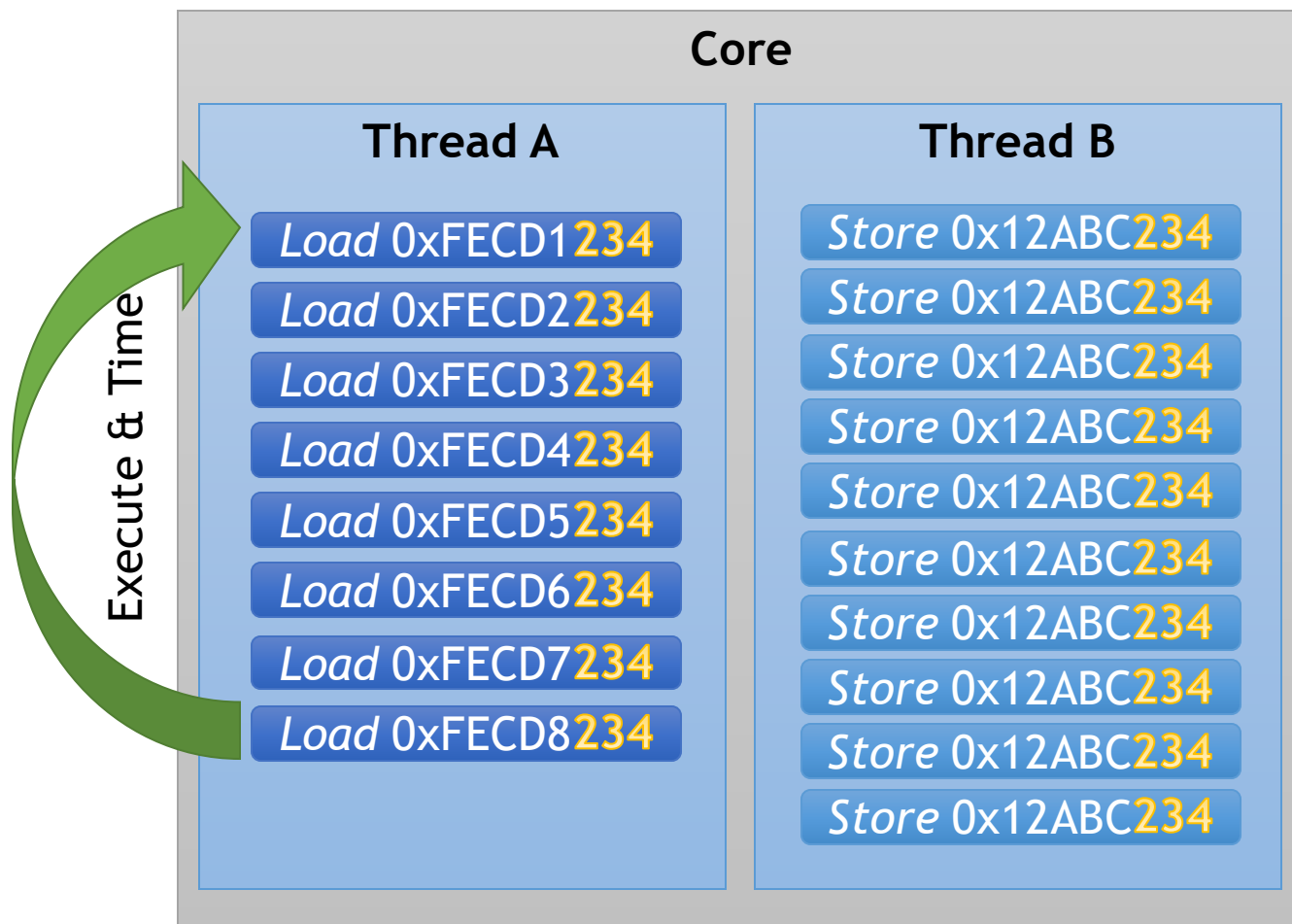
MemJam - 4K Aliasing across Sibling Threads



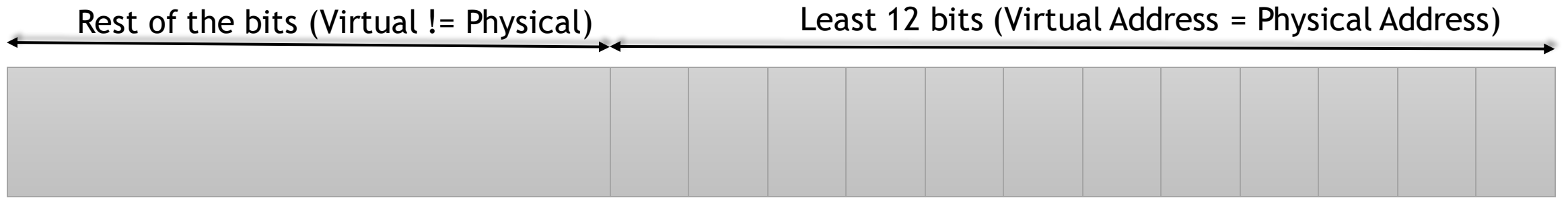
MemJam - 4K Aliasing across Sibling Threads



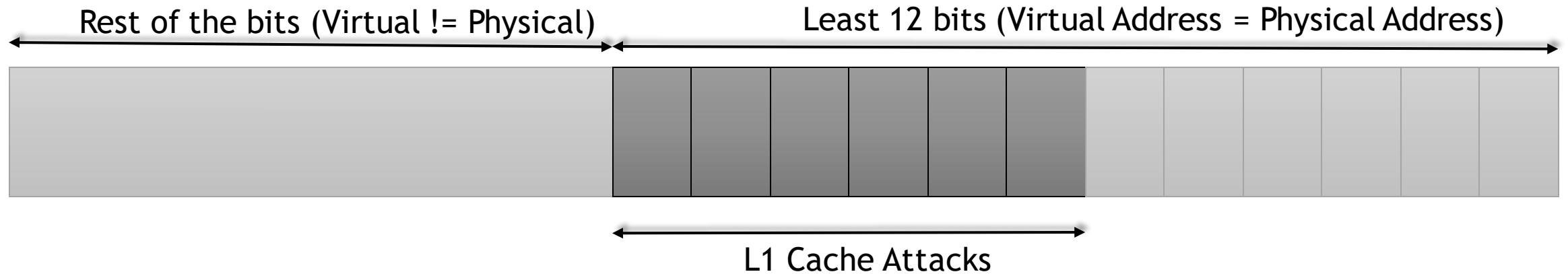
MemJam - 4K Aliasing across Sibling Threads



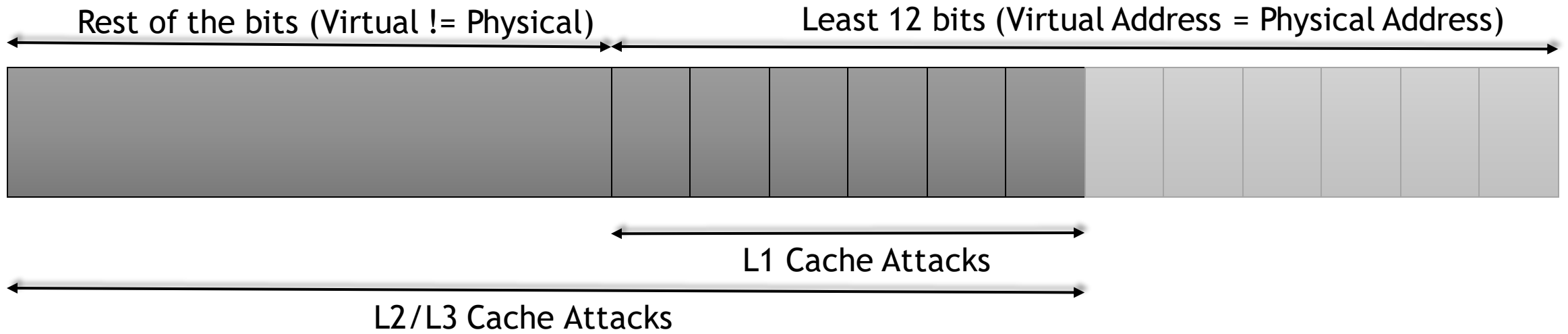
MemJam - Intra Cache Line Resolution



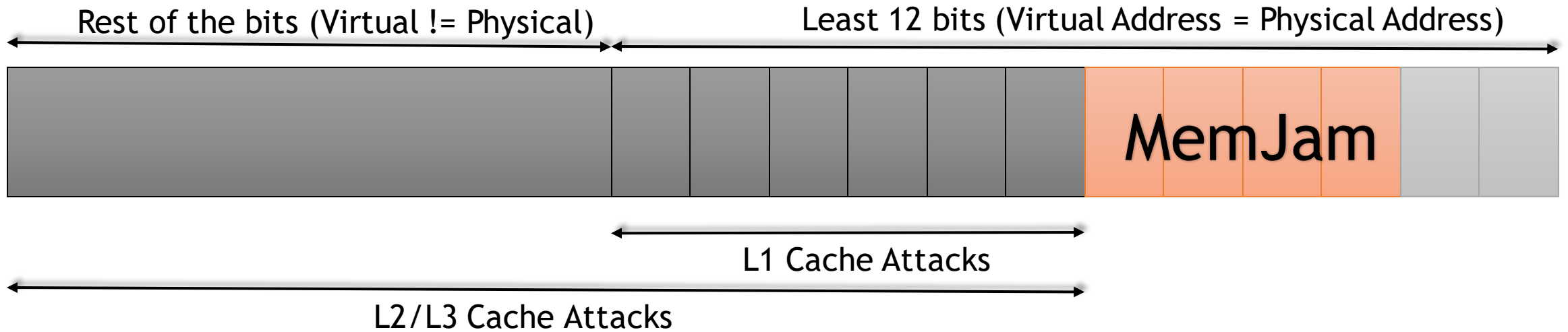
MemJam - Intra Cache Line Resolution



MemJam - Intra Cache Line Resolution



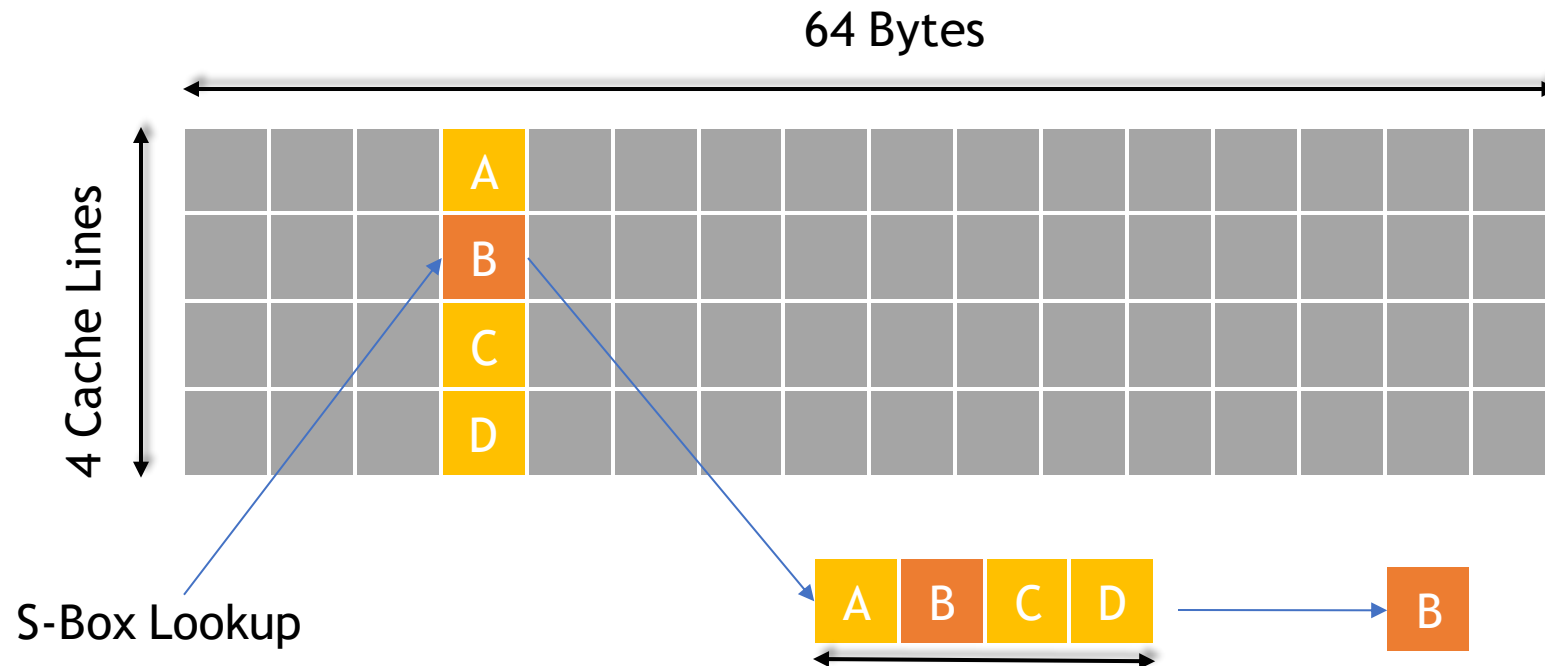
MemJam - Intra Cache Line Resolution



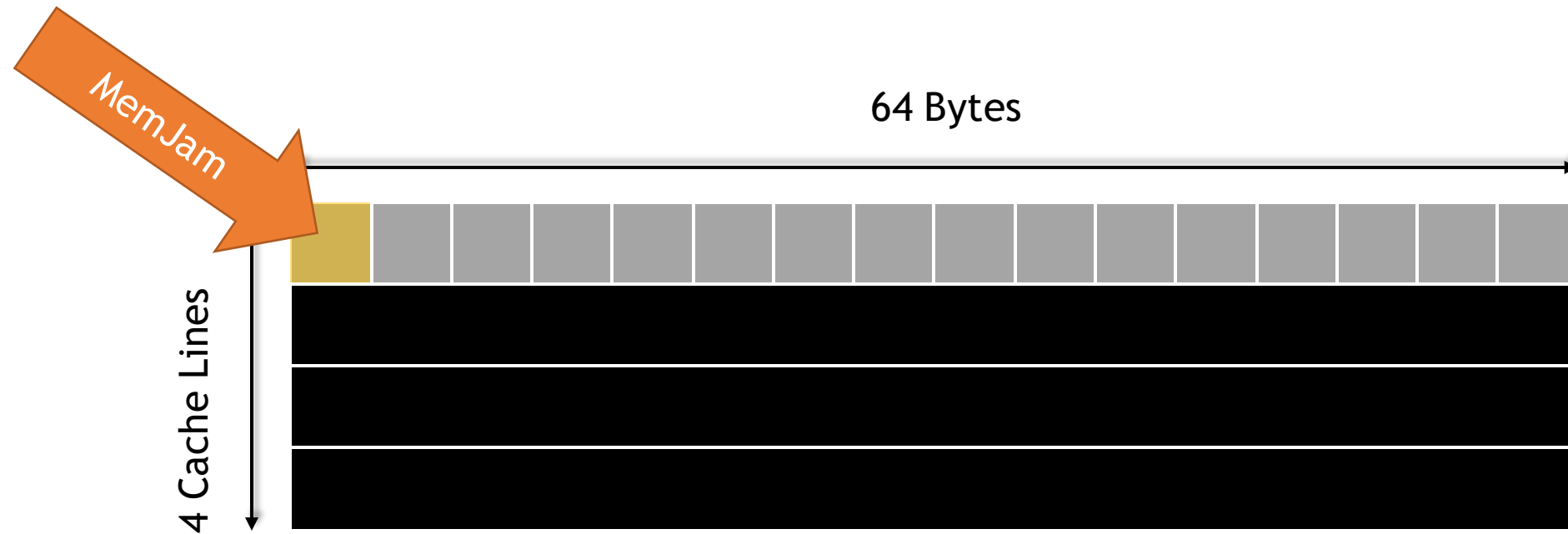
- Conflicted intra-cache line Leakage (4-byte granularity)
- Higher time → Memory accesses with the same bit 3 - 12
- 4 bits of intra-cache level leakage

MemJam - Attacking So-Called Constant Time AES

- Scatter-gather implementation of AES
 - Intel SGX Software Development Kit (SDK) and IPP Cryptography Library
 - 256 S-Box – 4 Cache Line
 - Cache independent access pattern

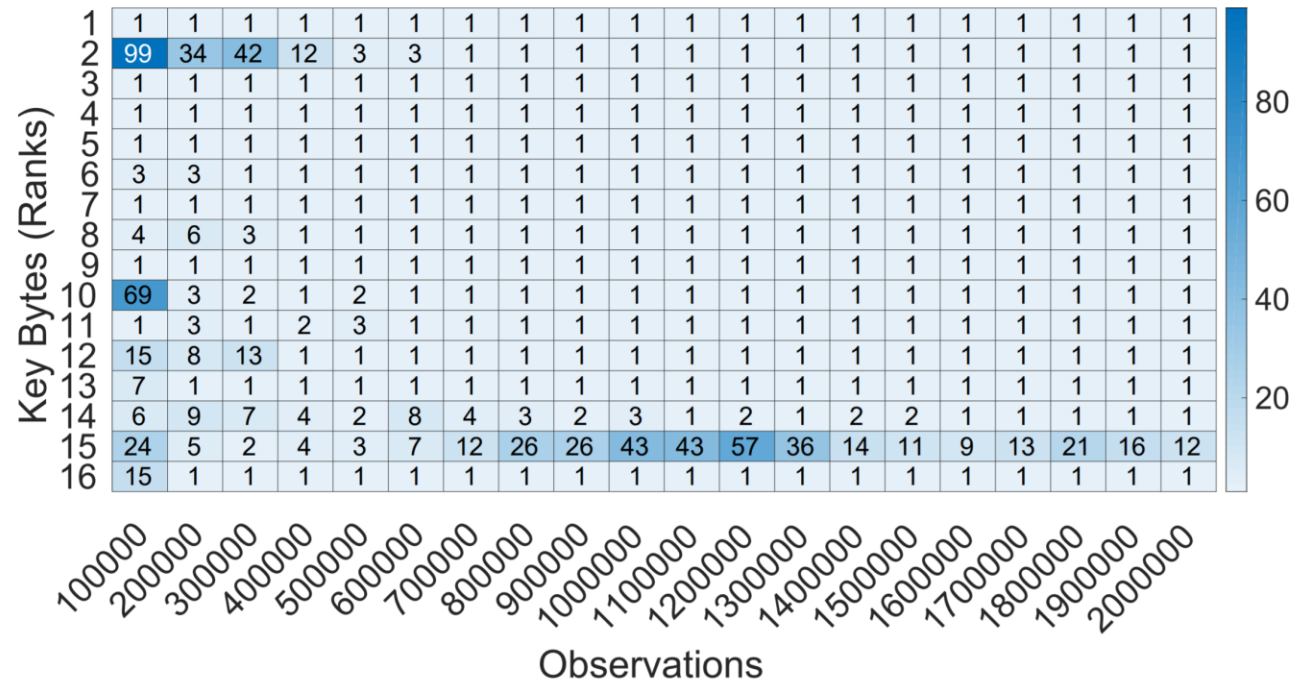


MemJam - Attacking So-Called Constant Time AES

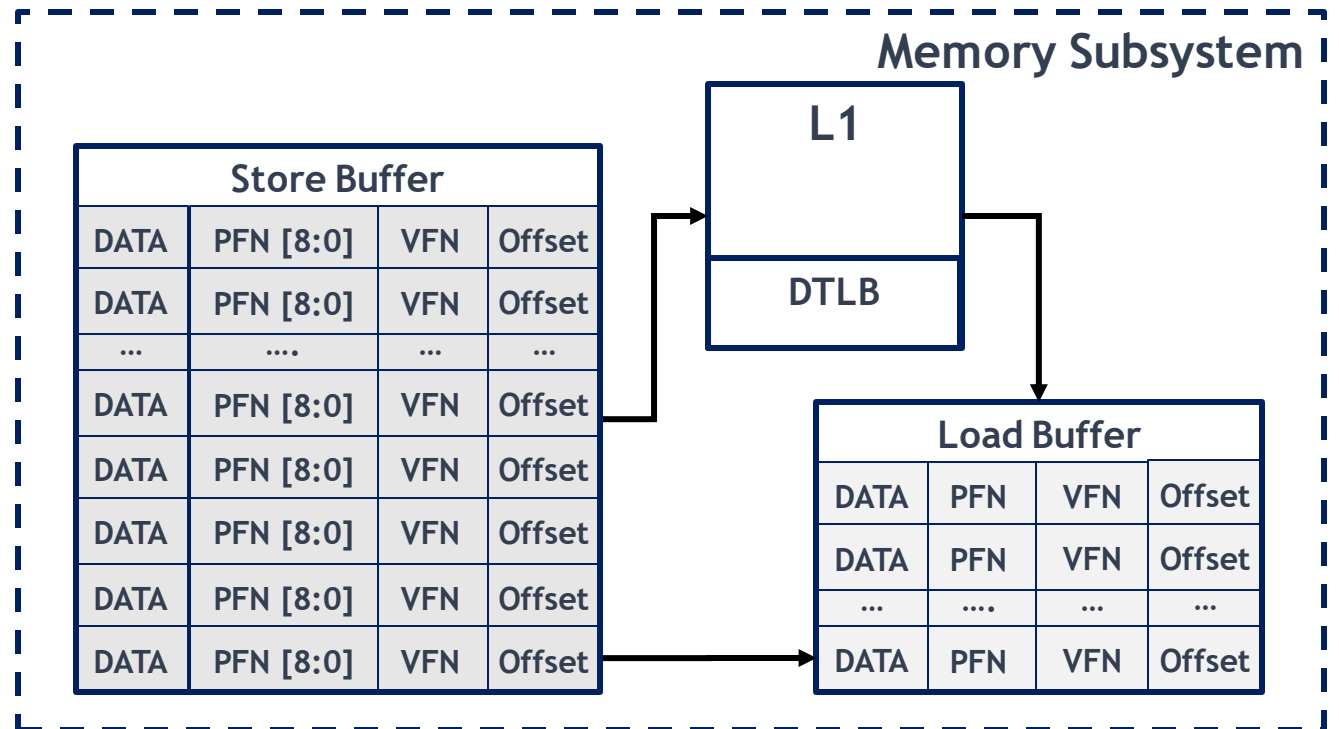
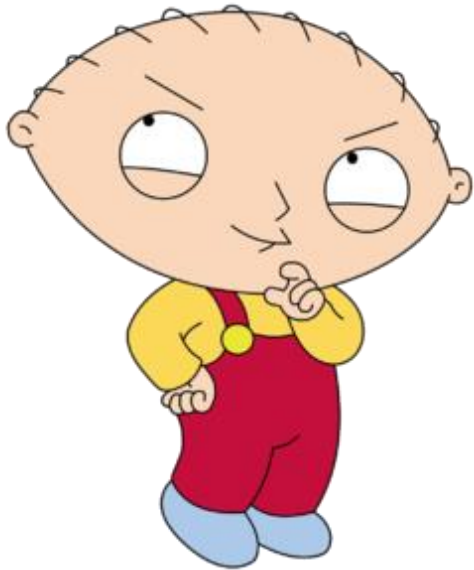


$$index = S^{-1}(c \oplus k) \longrightarrow index < 4.$$

MemJam - AES Key Recovery

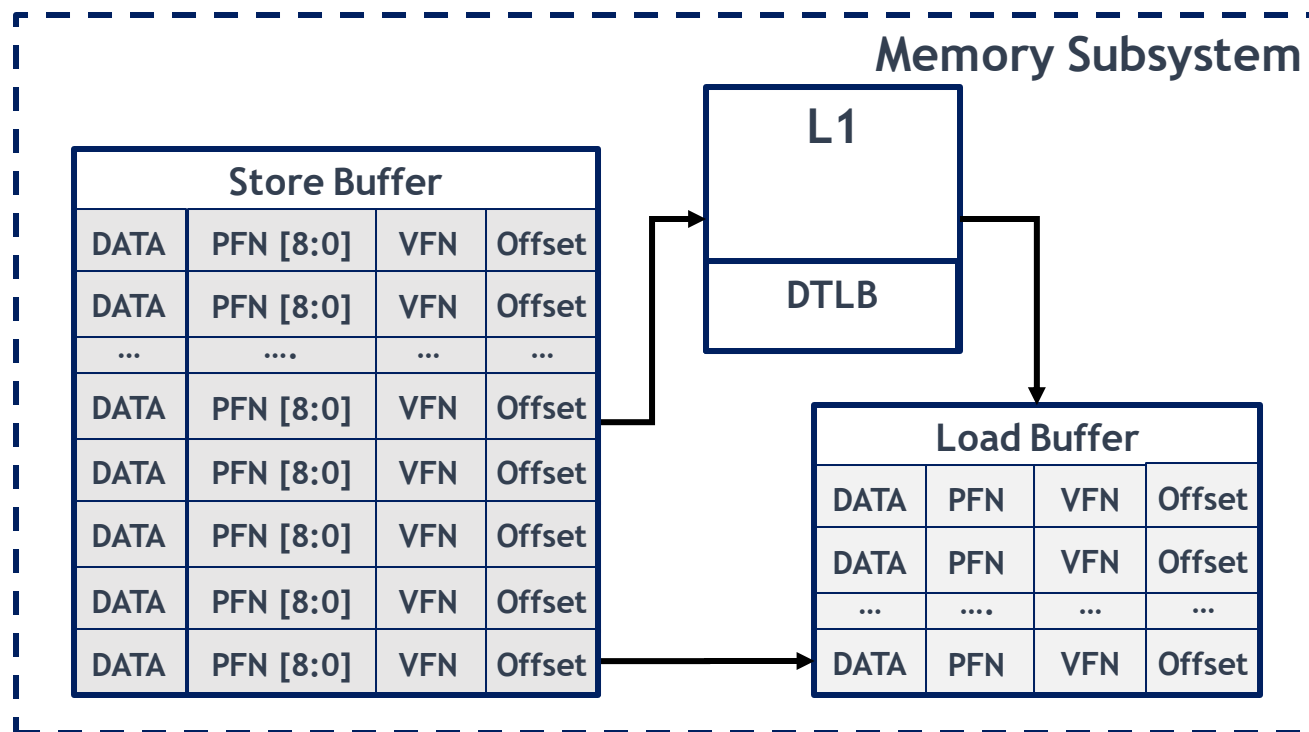


Are there other Address Aliasing?

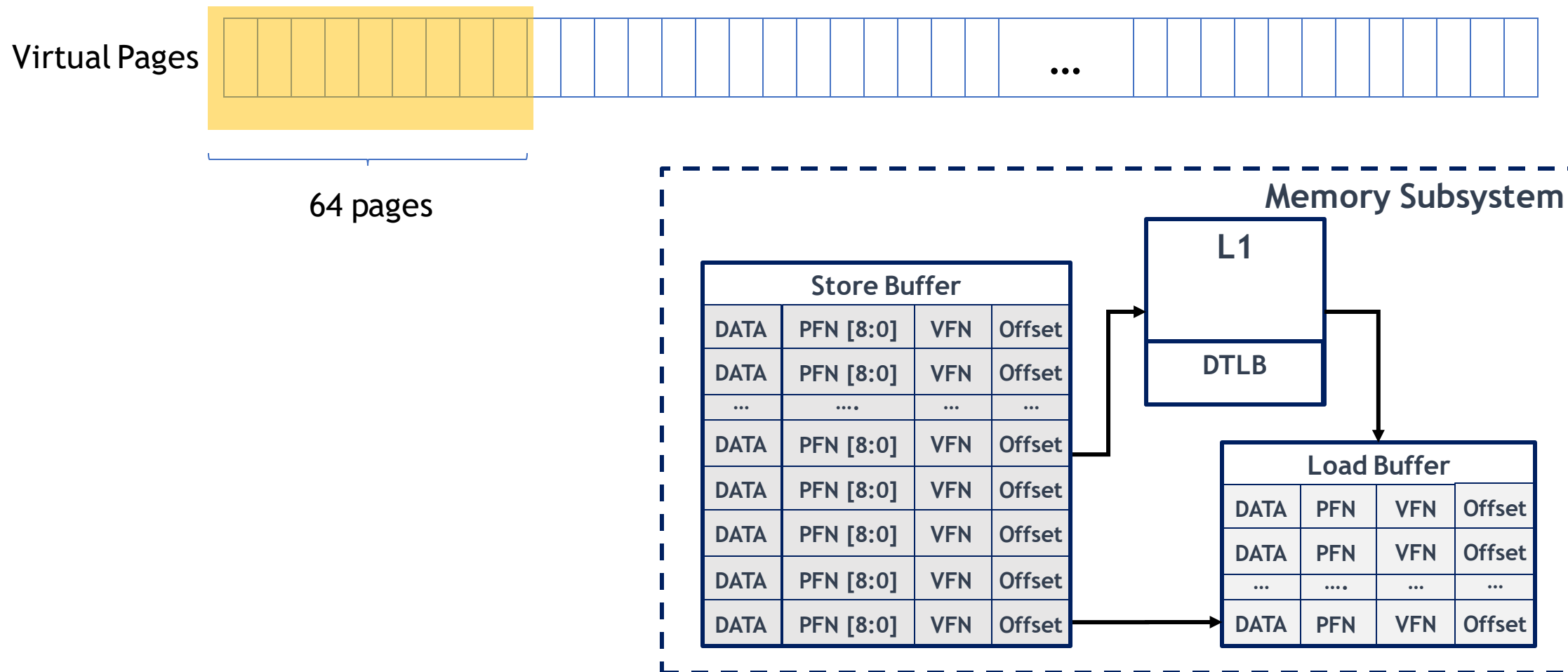


Spoiler: Finding Undocumented Aliasing

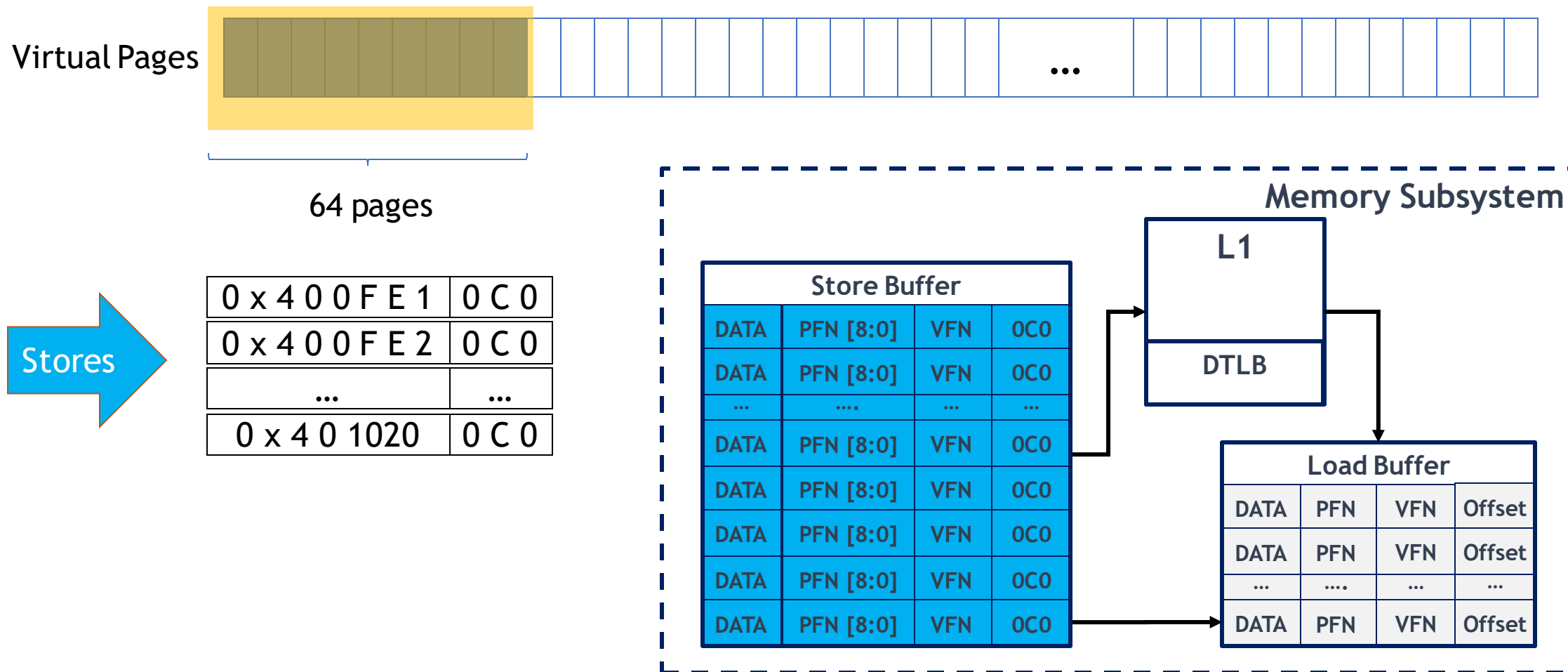
Virtual Pages



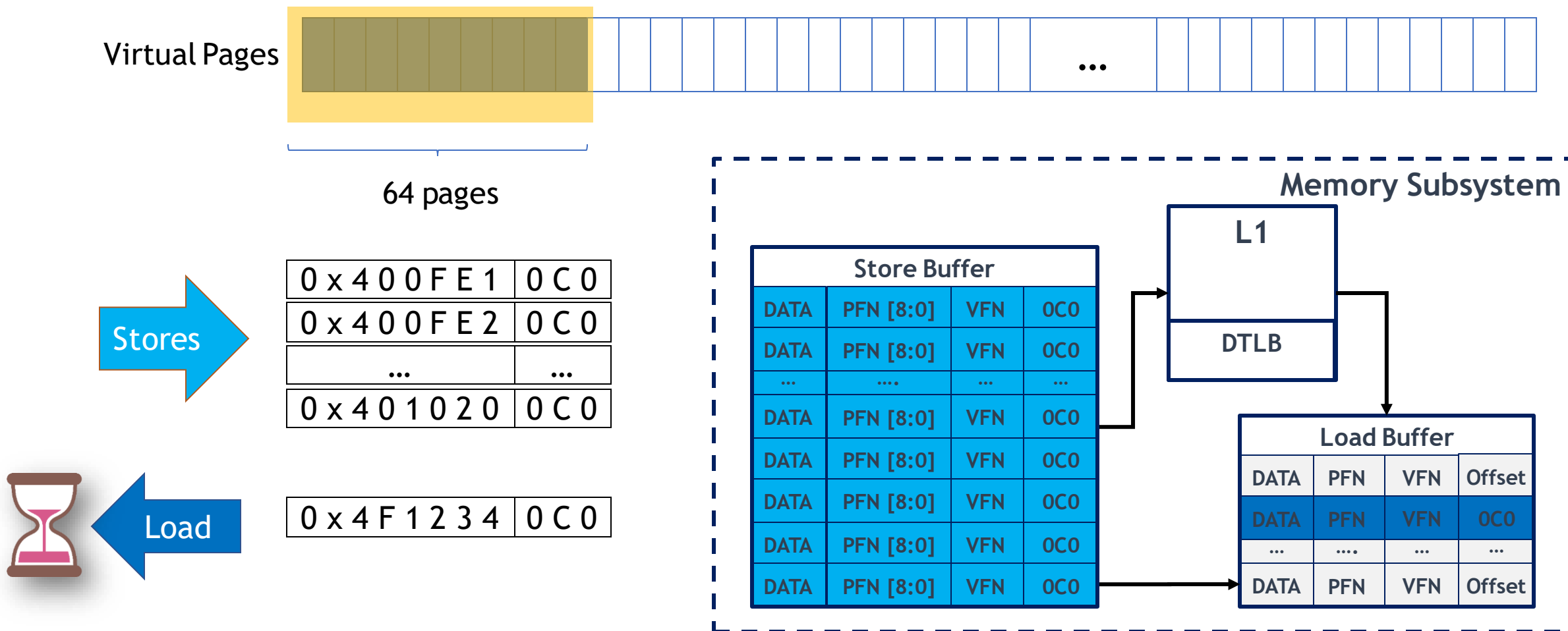
Spoiler: Finding Undocumented Aliasing



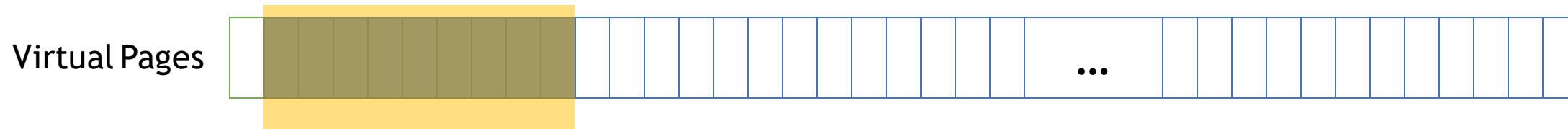
Spoiler: Finding Undocumented Aliasing



Spoiler: Finding Undocumented Aliasing



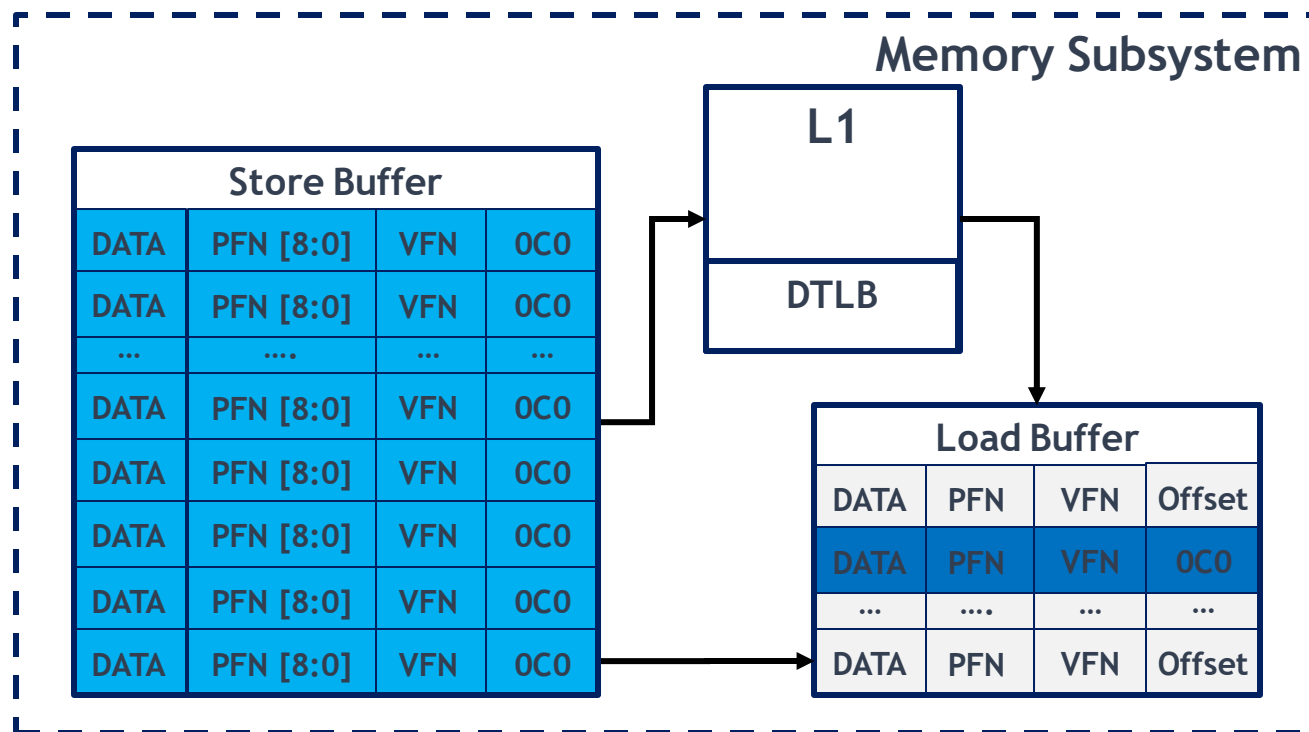
Spoiler: Finding Undocumented Aliasing



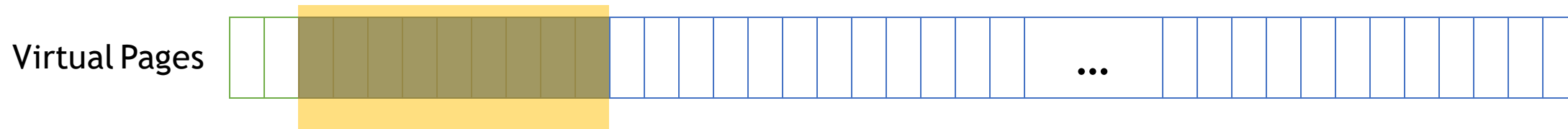
0 x 4 0 0 F E 2	0 C 0
0 x 4 0 0 F E 3	0 C 0
...	...
0 x 4 0 1 0 2 1	0 C 0



0 x 4 F 1 2 3 4	0 C 0
-----------------	-------



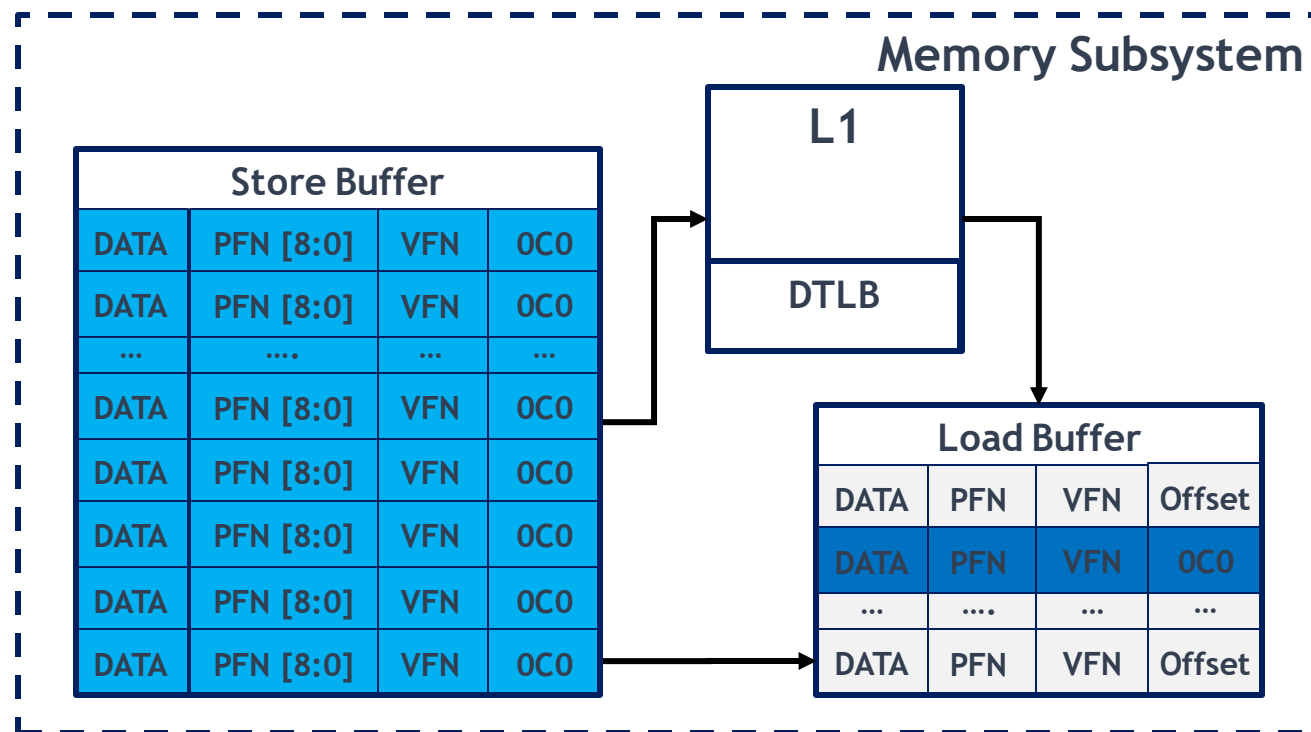
Spoiler: Finding Undocumented Aliasing



0 x 4 0 0 F E 3	0 C 0
0 x 4 0 0 F E 4	0 C 0
...	...
0 x 4 0 1 0 2 2	0 C 0

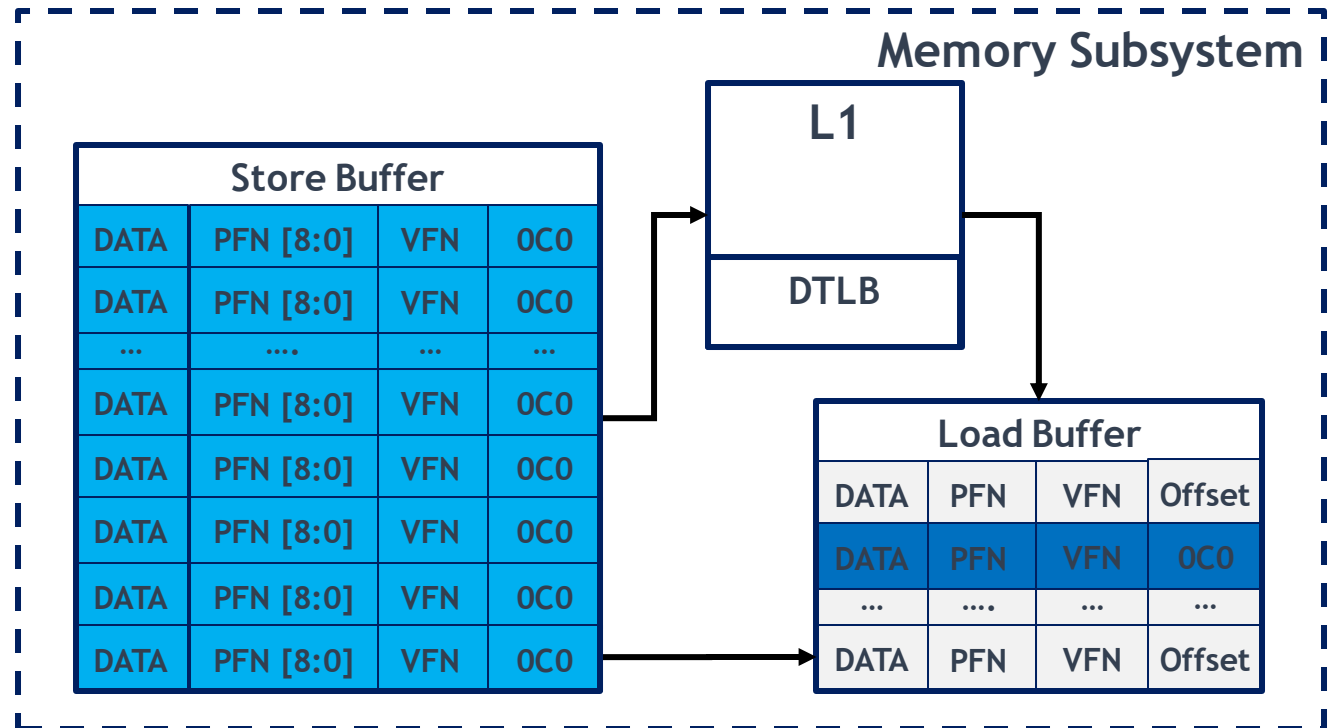
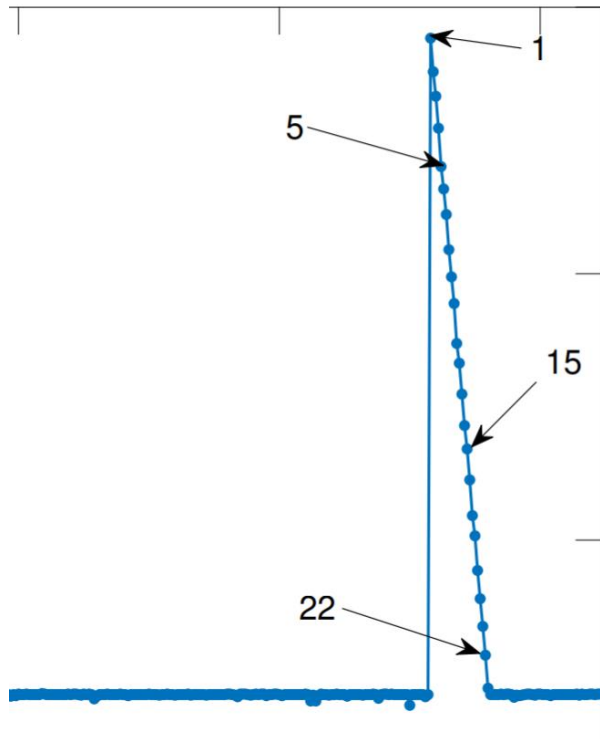
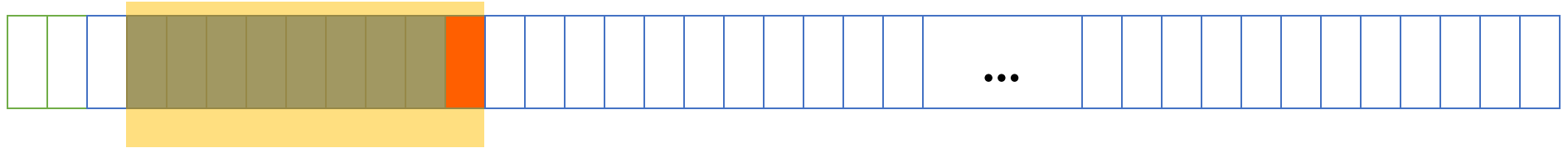


0 x 4 F 1 2 3 4	0 C 0
-----------------	-------

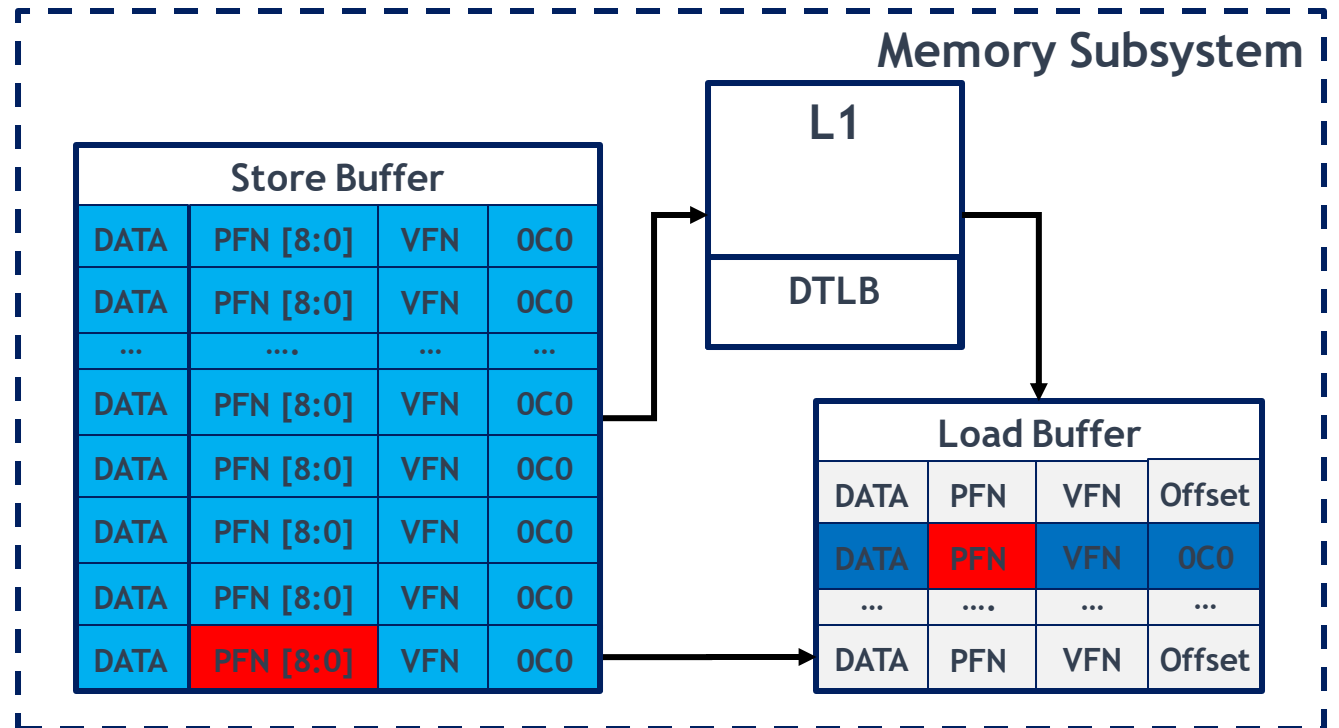
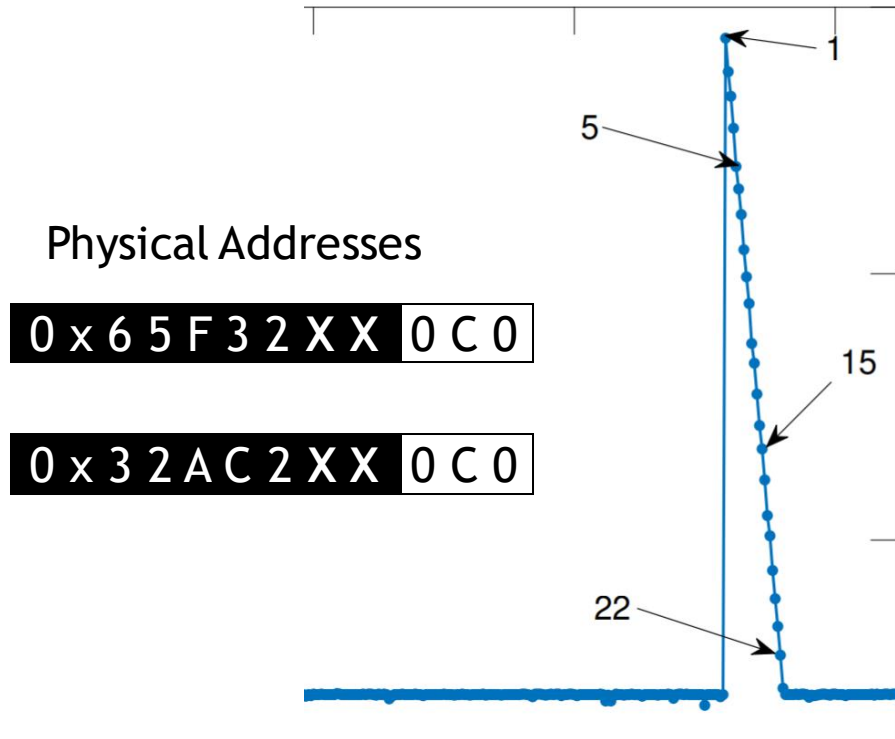
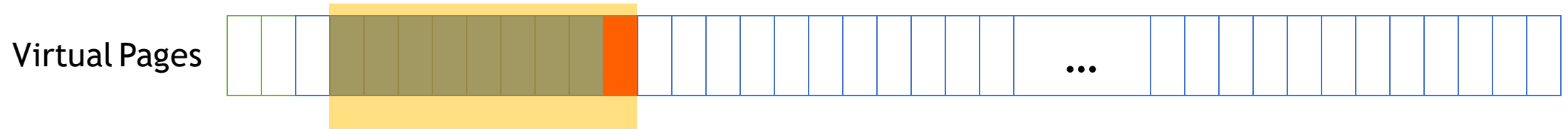


Spoiler: Finding Undocumented Aliasing

Virtual Pages

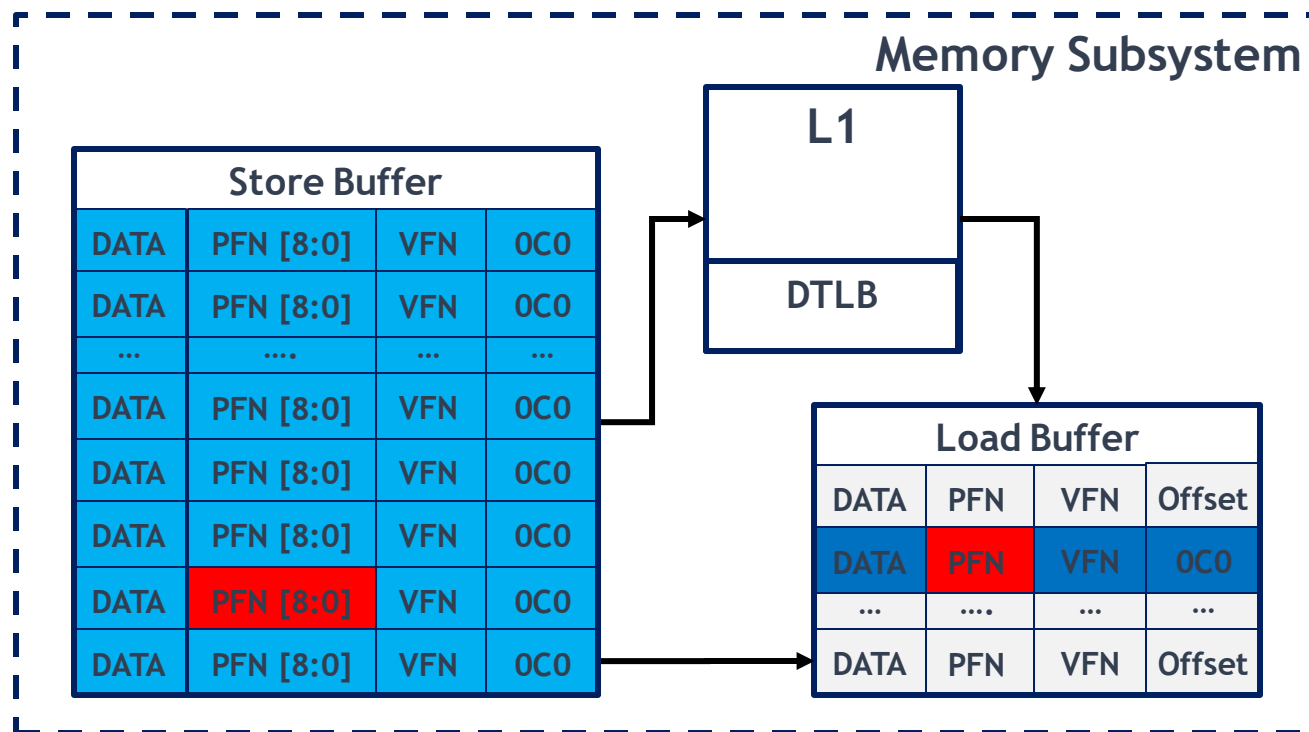
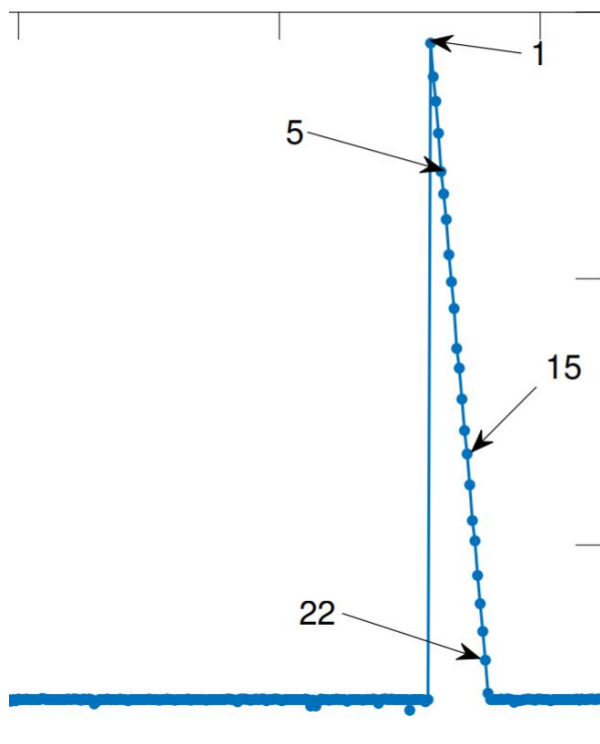
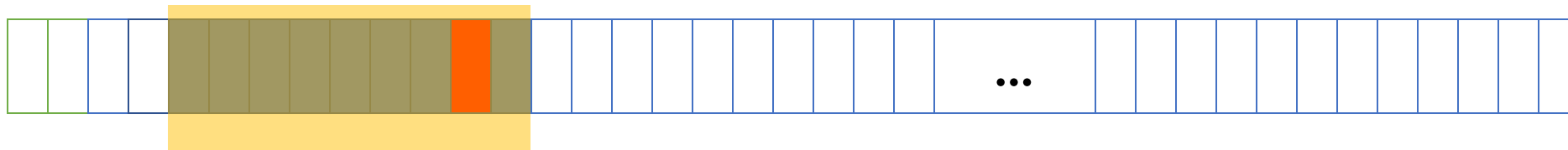


Spoiler: Finding Undocumented Aliasings

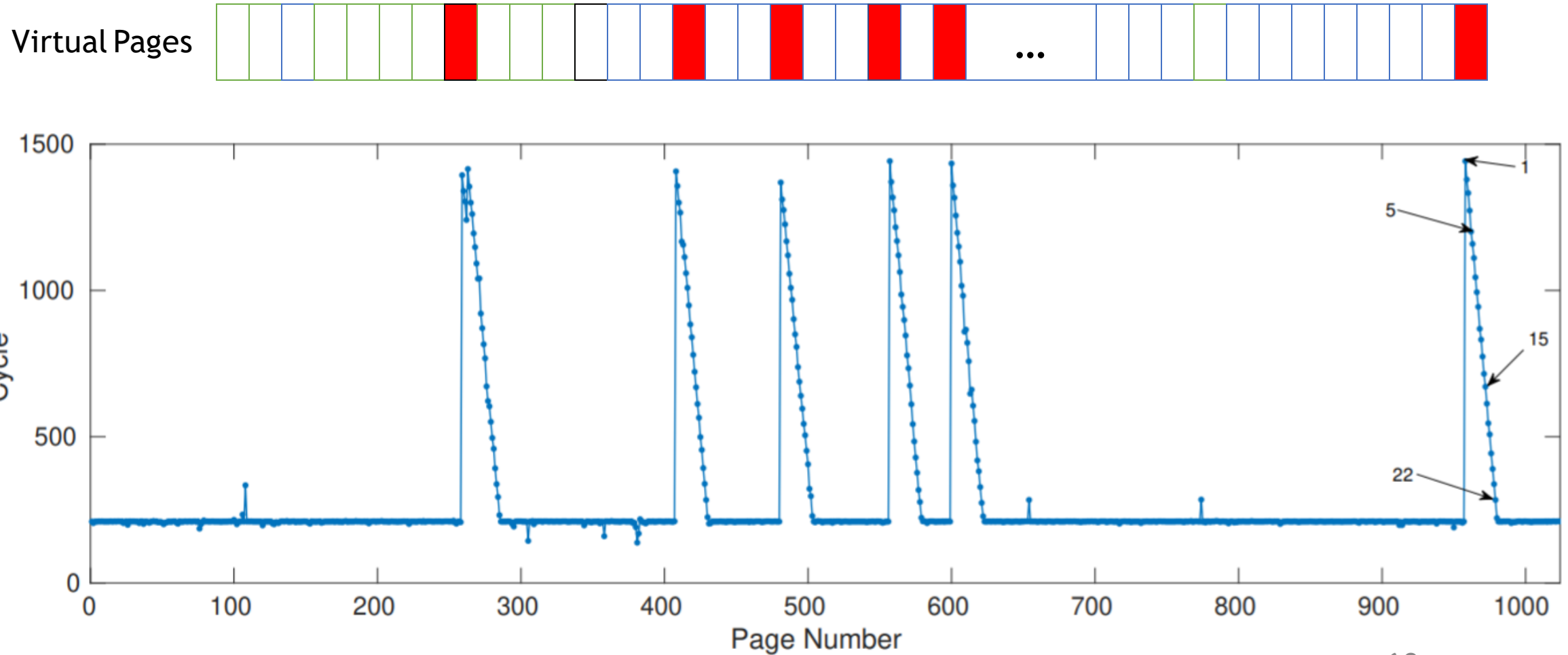


Spoiler: Finding Undocumented Aliasing

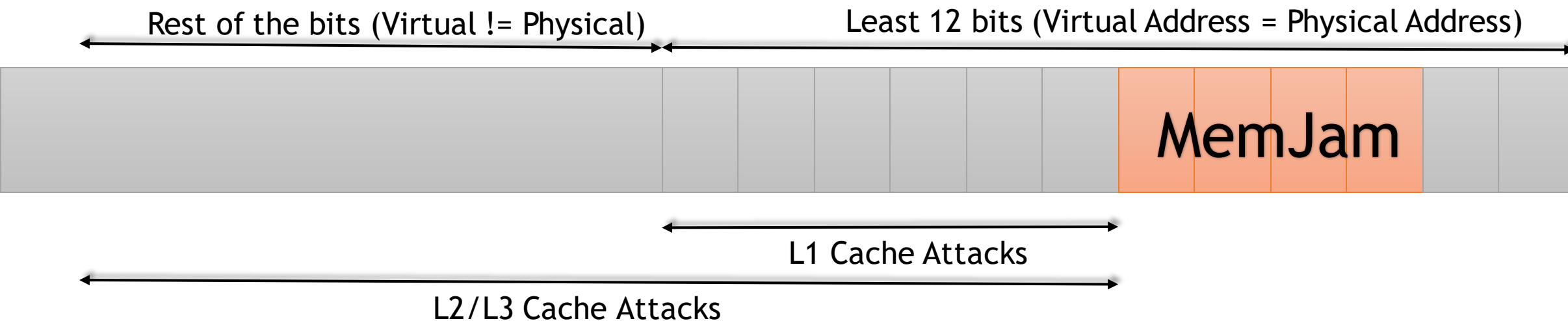
Virtual Pages



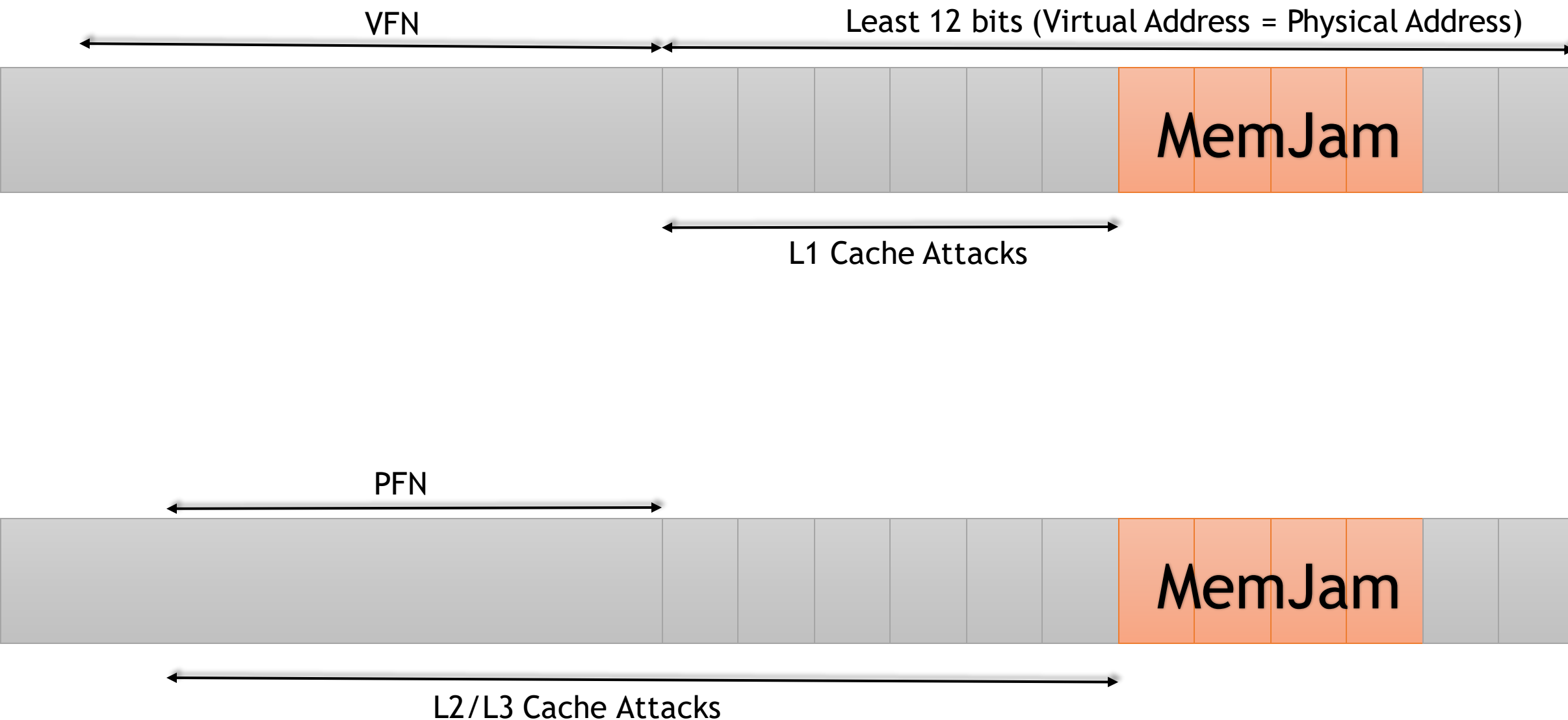
Spoiler: Finding Undocumented Aliasing



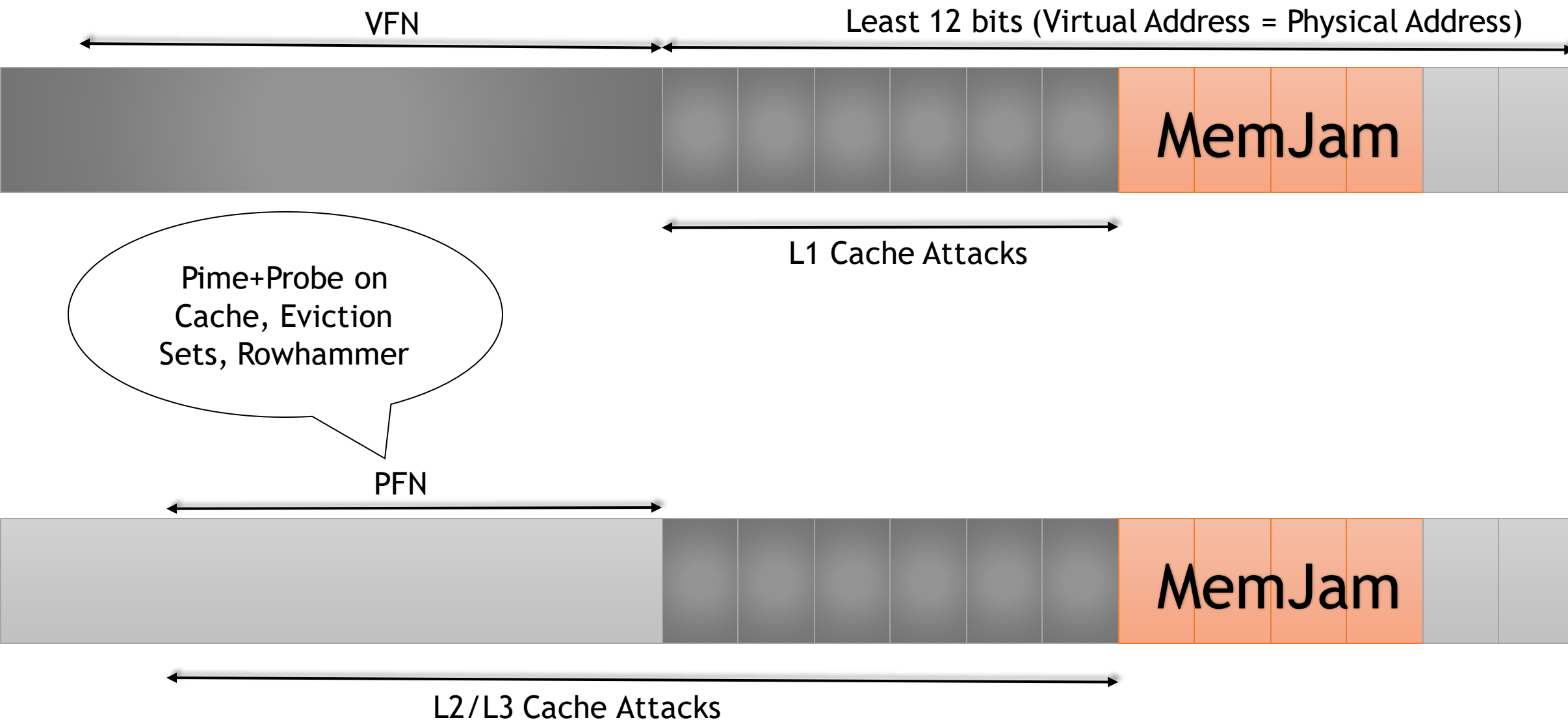
Spoiler: Learning on Physical Address Bits



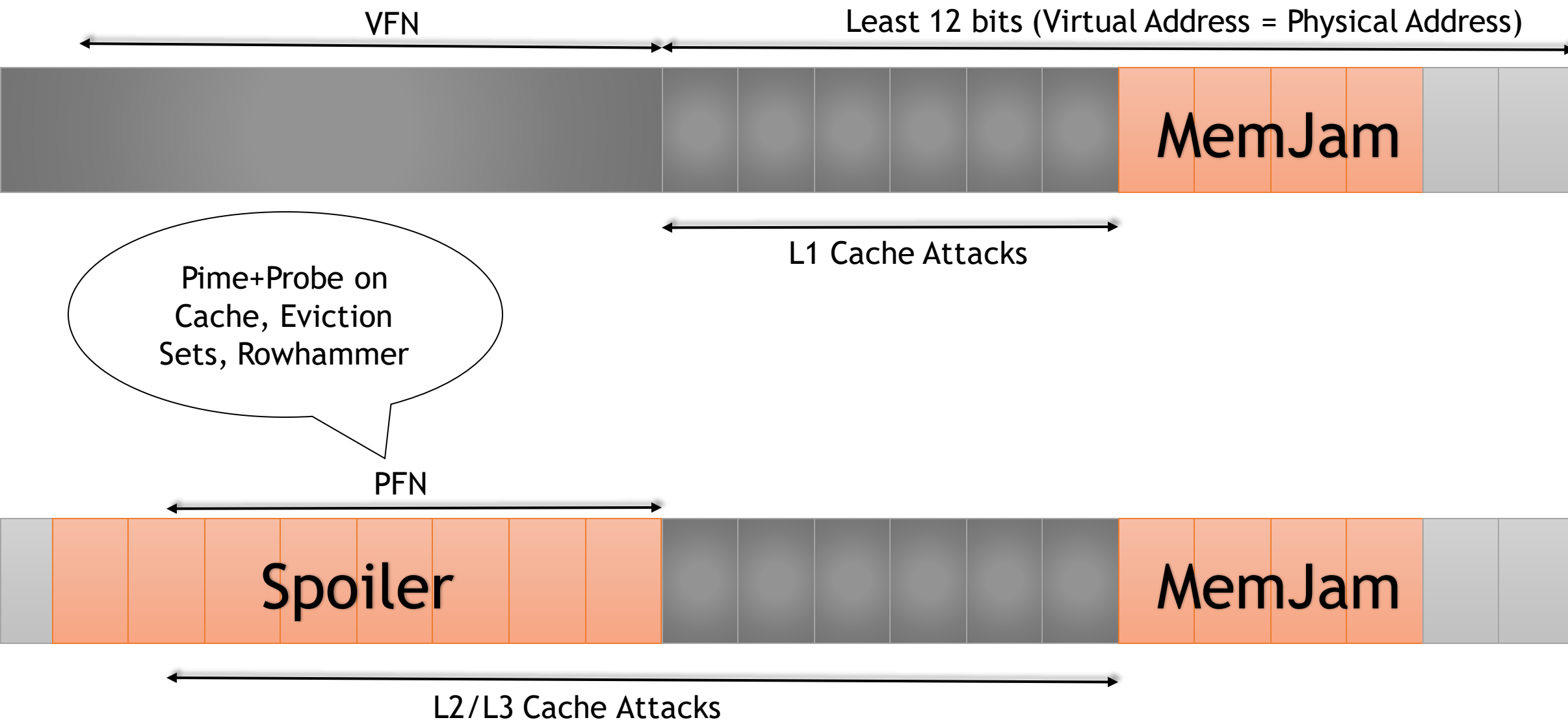
Spoiler: Learning on Physical Address Bits

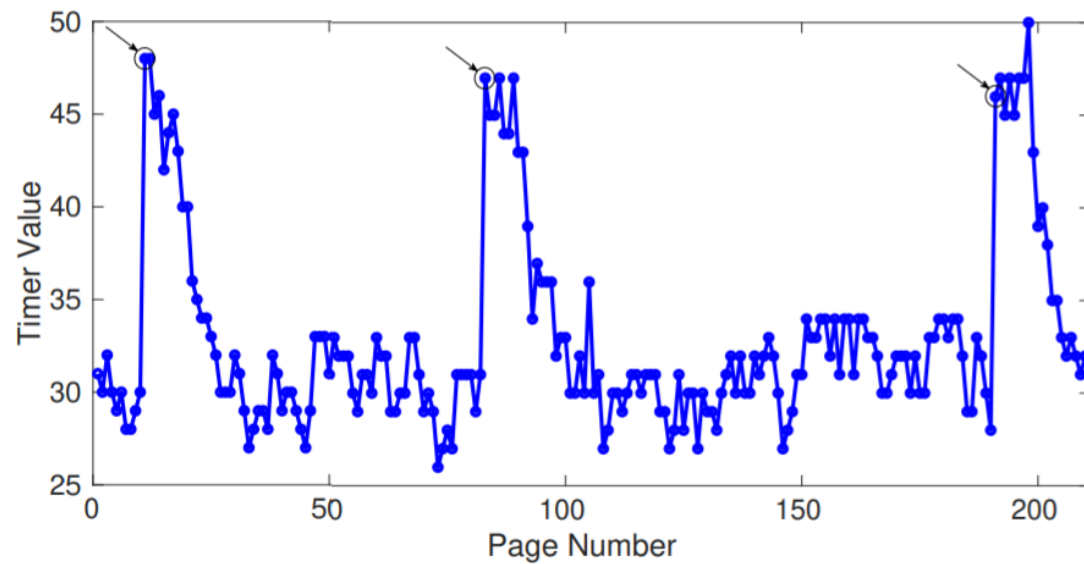


Spoiler: Learning on Physical Address Bits



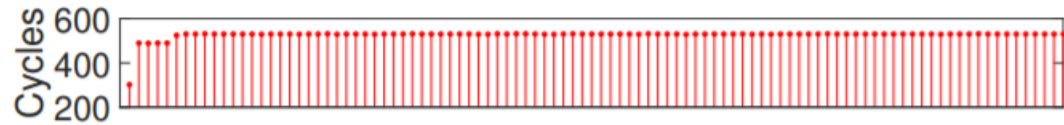
Spoiler: Learning on Physical Address Bits



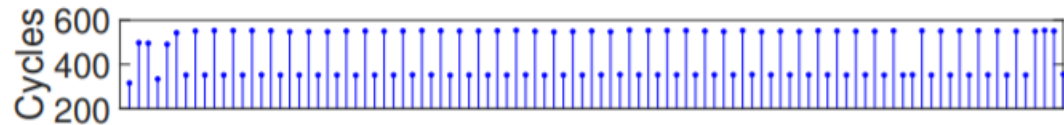


Algorithm	R	t_{total}	t_{AAS}	t_{ESS}	Success
Classic [42]	3	46s	-	100%	80%
Improved [14]	3	35s	-	100%	80%
AA (ours)	10	10s	54%	46%	67%
AA (ours)	20	12s	75%	25%	100%

Spoiler - JavaScript
Eviction Sets



(a) 19 bits used by memory controller, no unknown bits



(b) 21 bits used by memory controller, 1 unknown bit

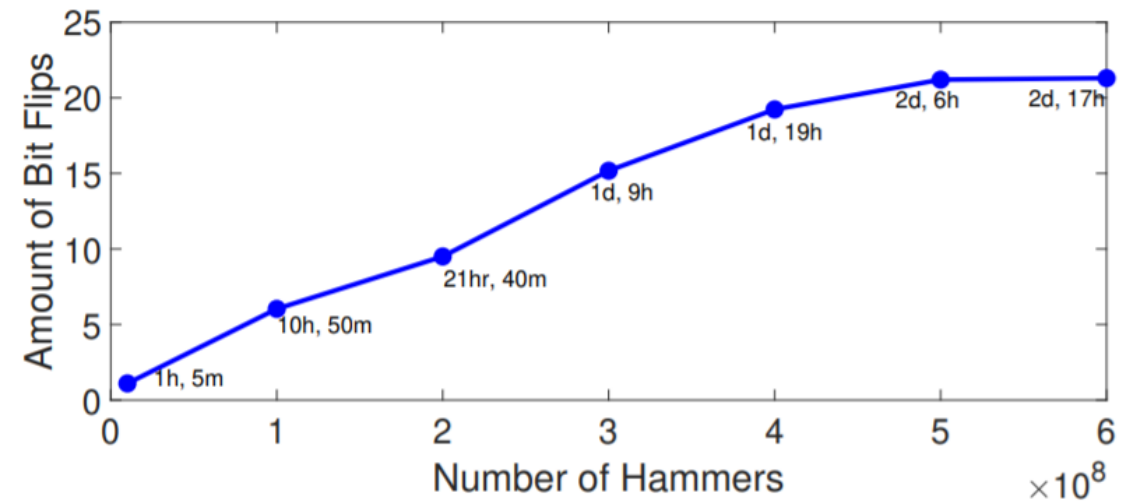
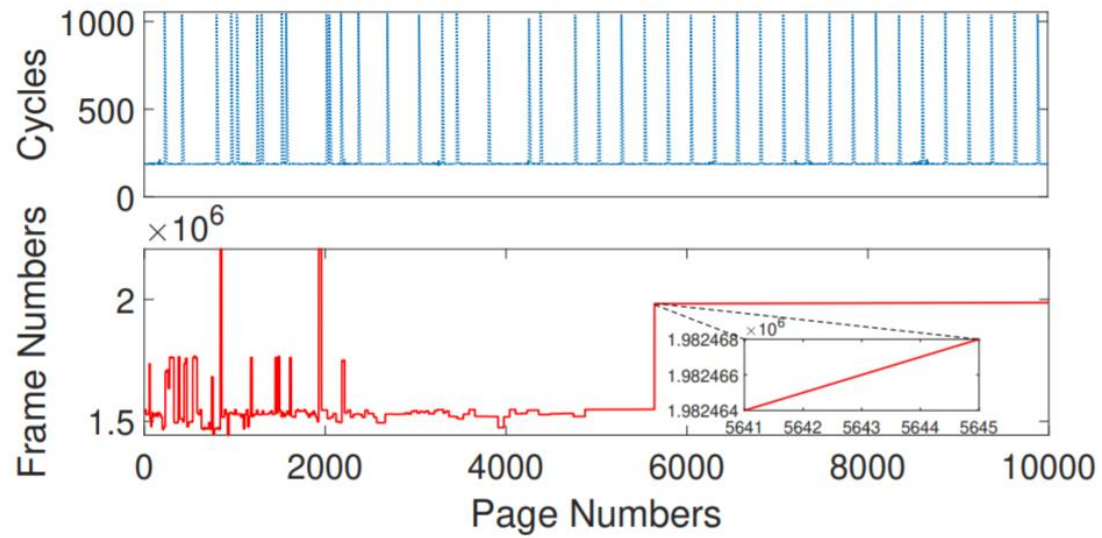


(c) 22 bits used by memory controller, 2 unknown bits

System Model	DRAM Configuration	# of Bits
Dell XPS-L702x (Sandy Bridge)	1 x (4GB 2Rx8)	21
	2 x (4GB 2Rx8)	22
Dell Inspiron-580 (Nehalem)	1 x (2GB 2Rx8) (b)	21
	2 x (2GB 2Rx8) (c)	22
	4 x (2GB 2Rx8) (d)	23
Dell Optiplex-7010 (Ivy Bridge)	1 x (2GB 1Rx8) (a)	19
	2 x (2GB 1Rx8)	20
	1 x (4GB 2Rx8) (e)	21
	2 x (4GB 2Rx8)	22

Spoiler - Rowhammer

- Row Buffer Conflict
- Single-sided Rowhammer



Spoiler - Rowhammer

- Detecting Contiguous Memory
- Double-sided Rowhammer

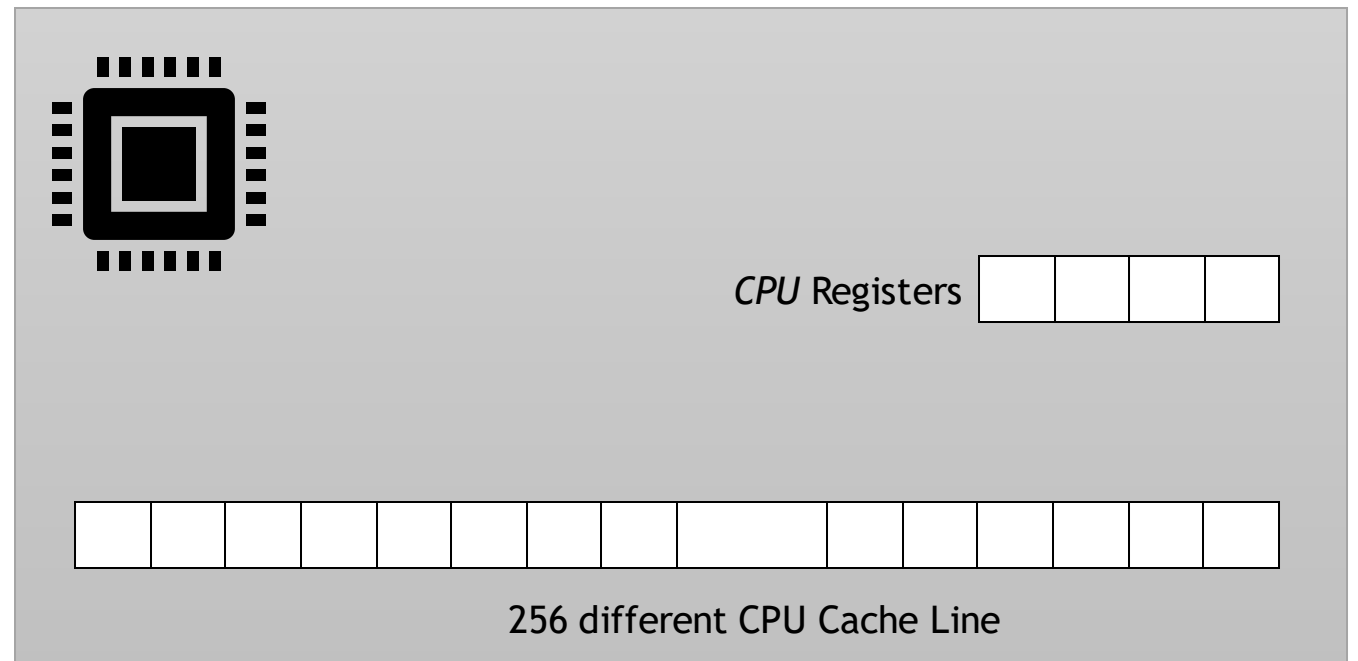
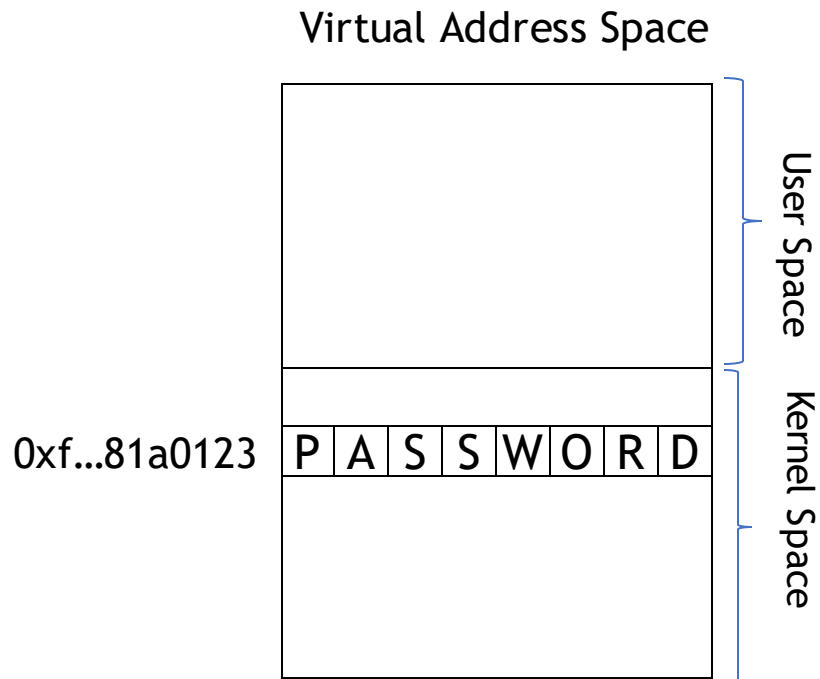
2018: Meltdown Attack?

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



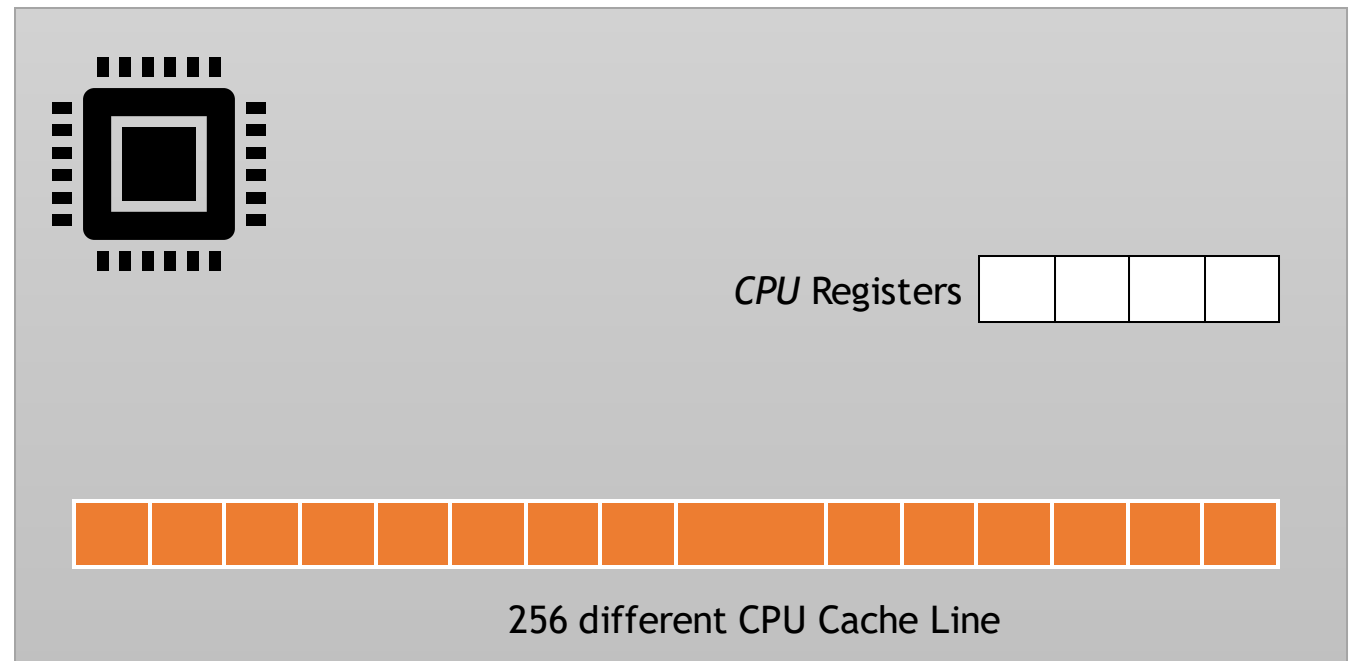
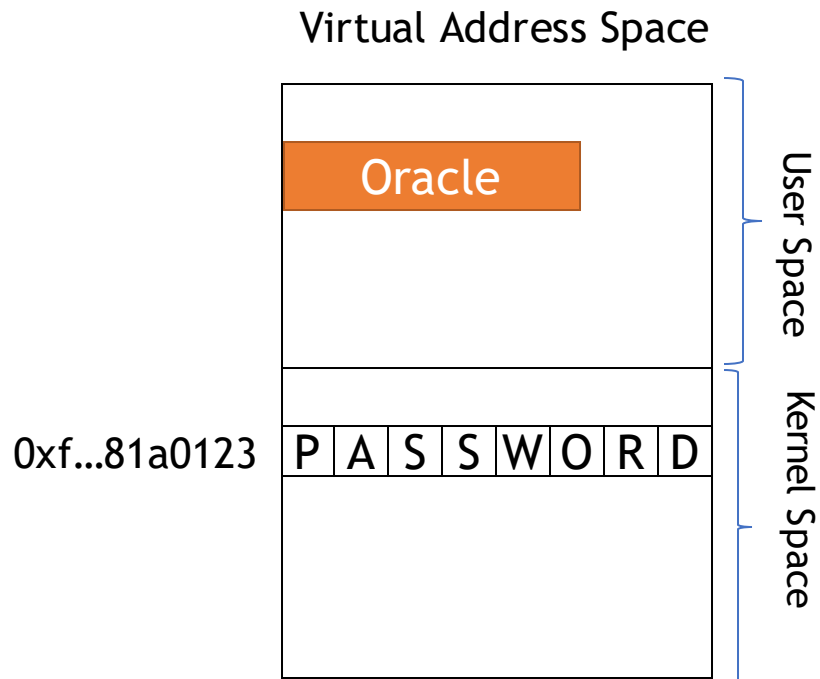
2018: Meltdown Attack?

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



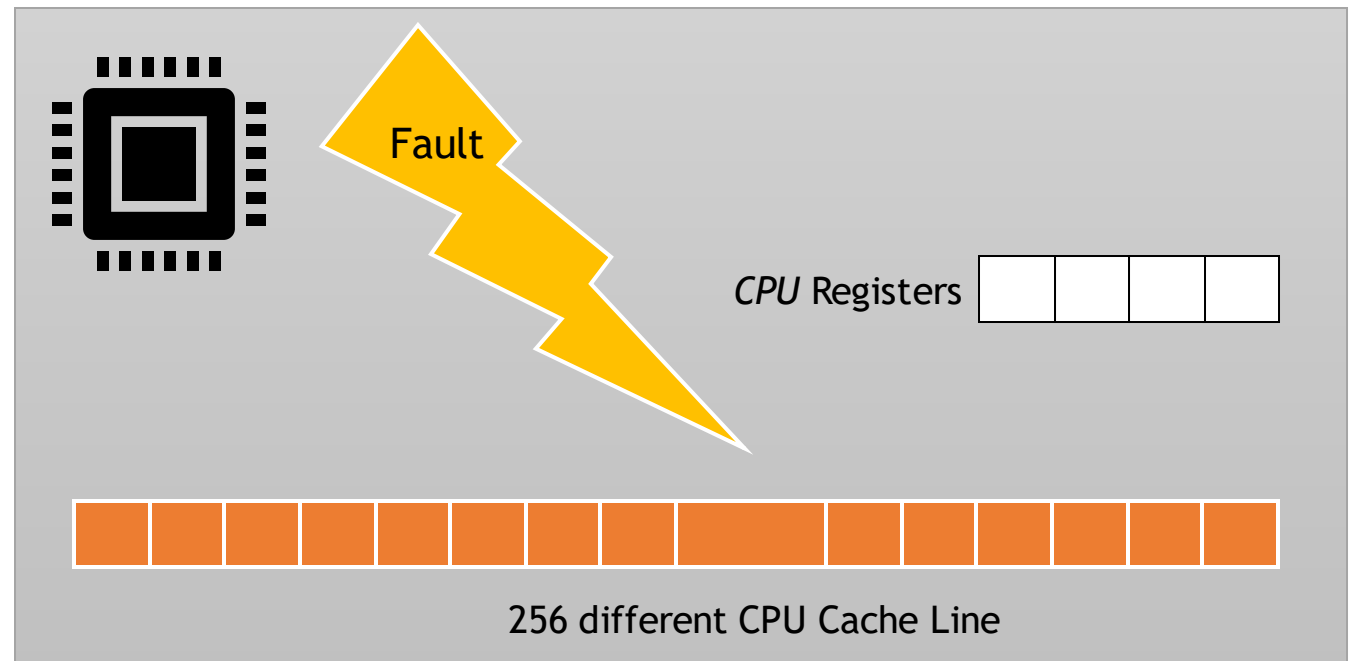
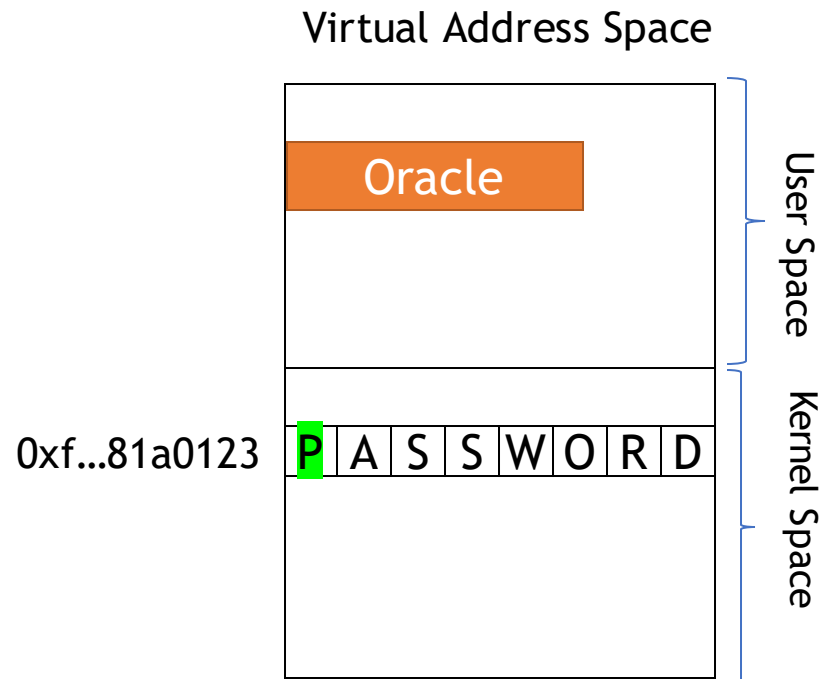
2018: Meltdown Attack?

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



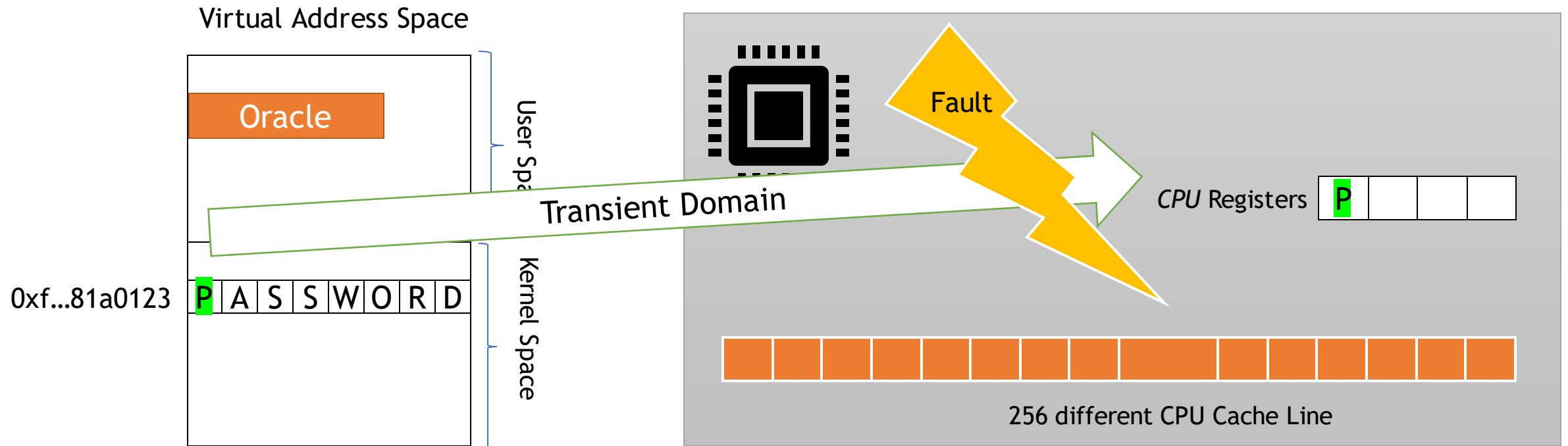
2018: Meltdown Attack?

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



2018: Meltdown Attack?

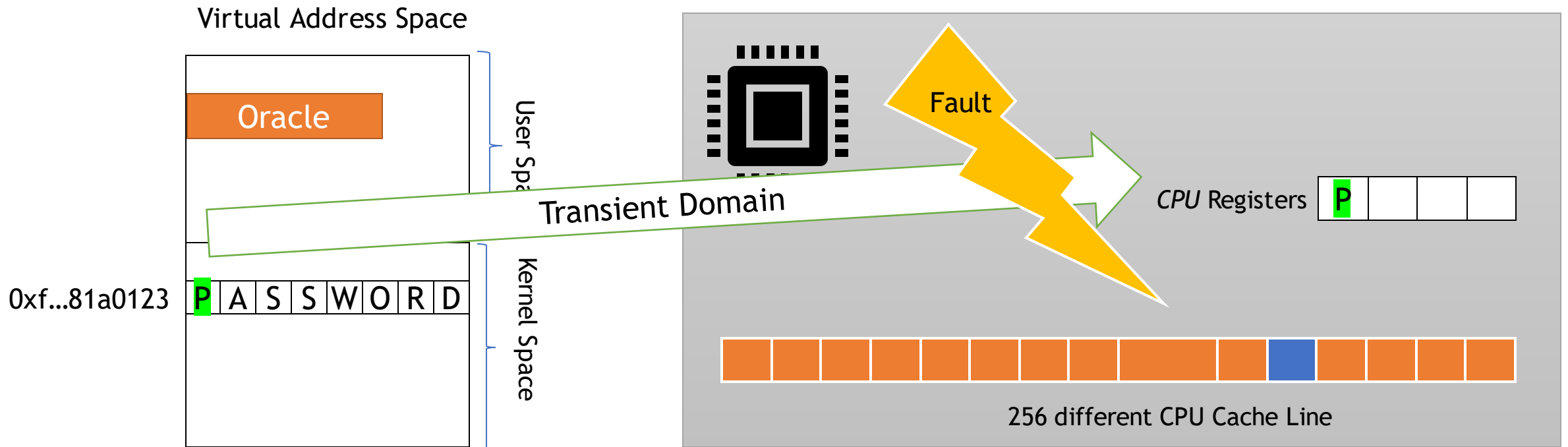
➡ `char secret = *(char *) 0xffffffff81a0123;`



2018: Meltdown Attack?

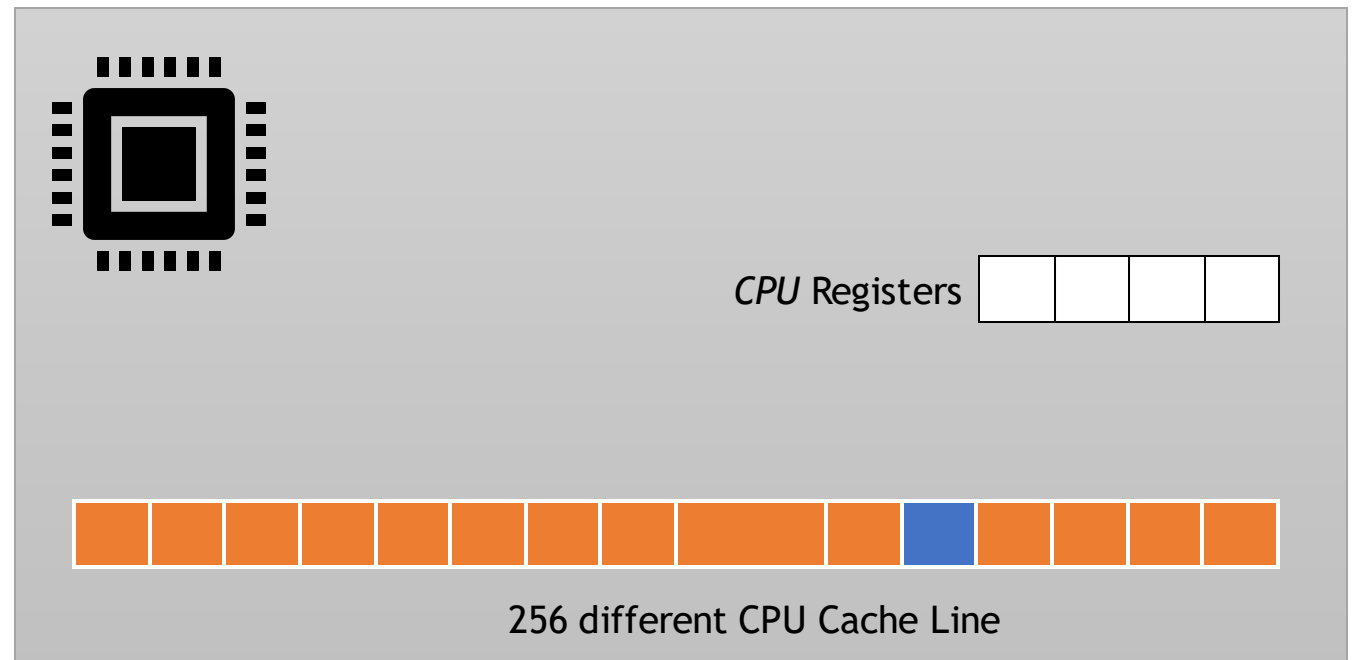
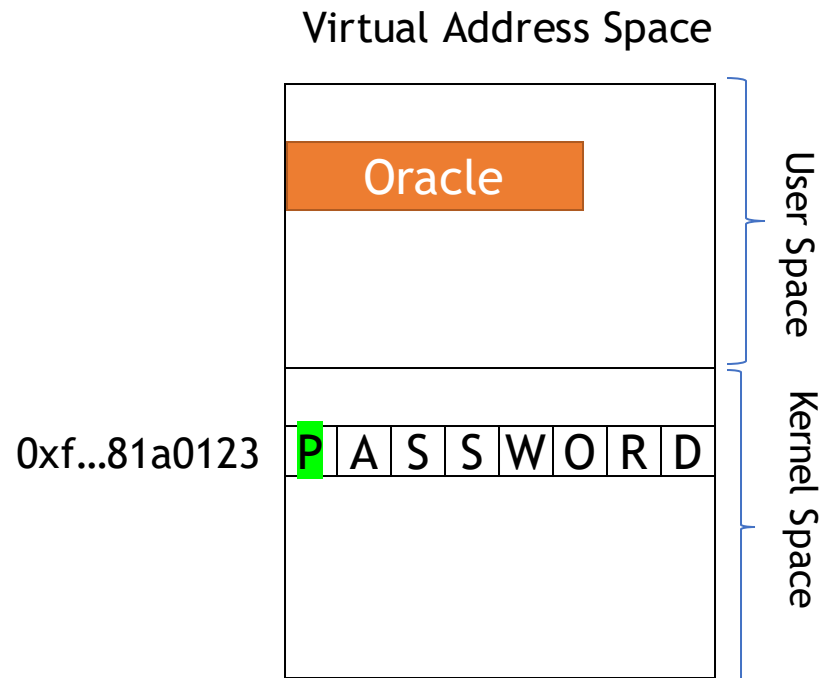
➔

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



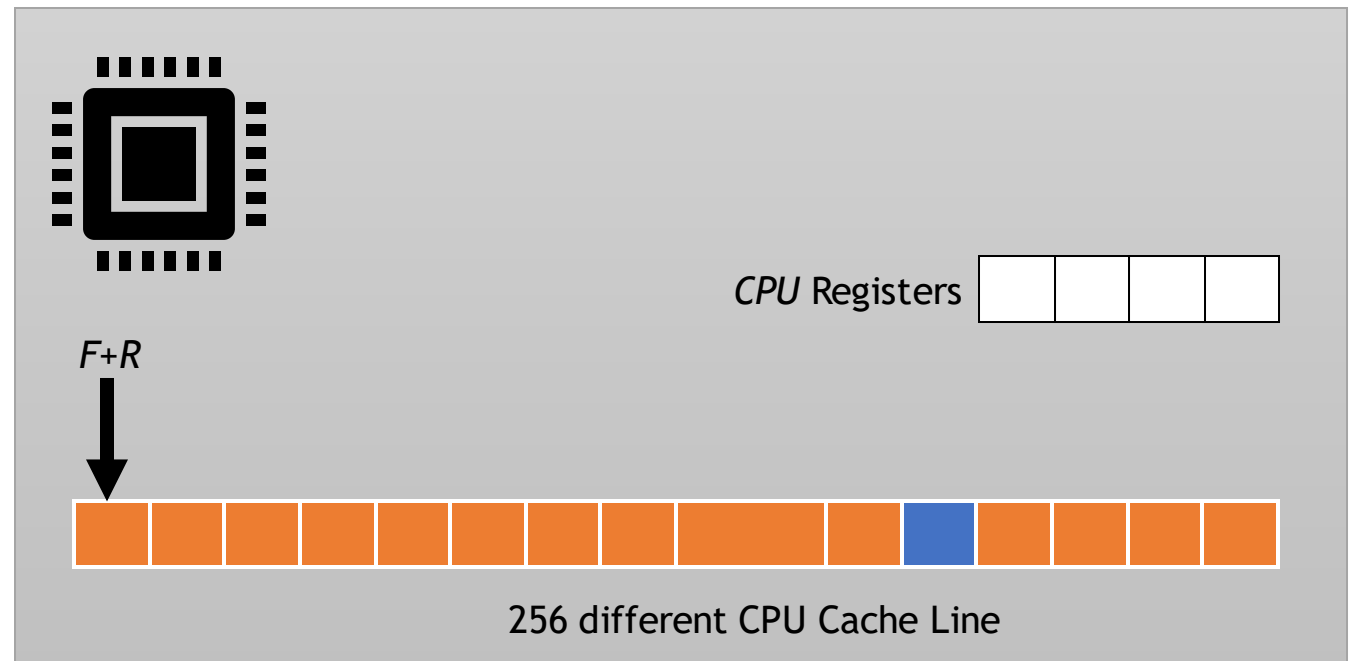
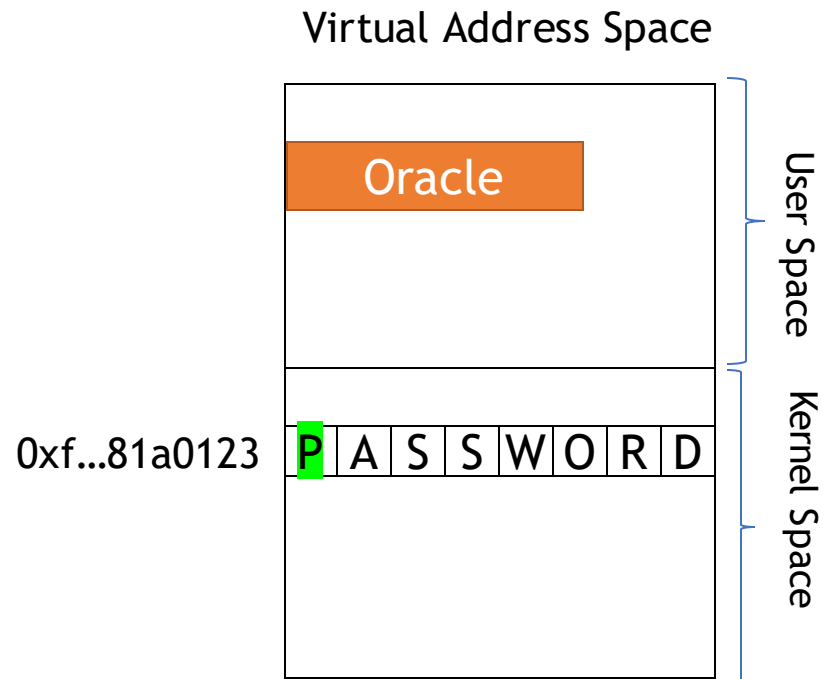
2018: Meltdown Attack?

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



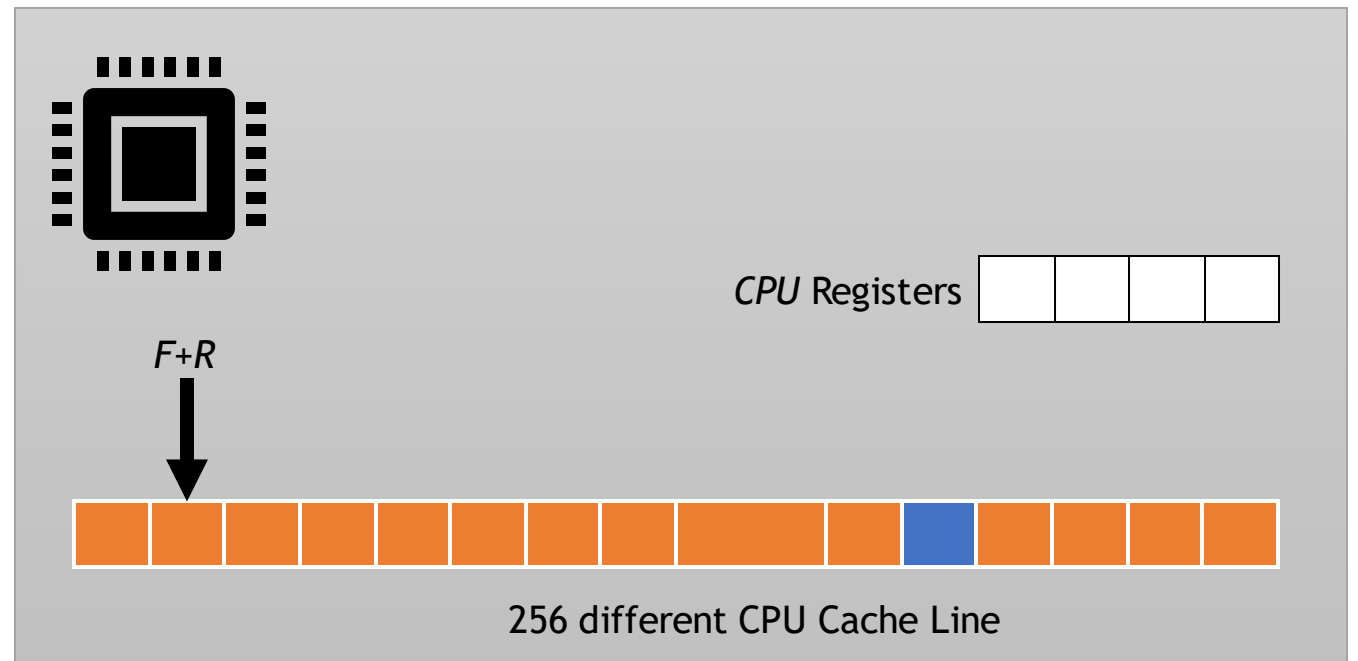
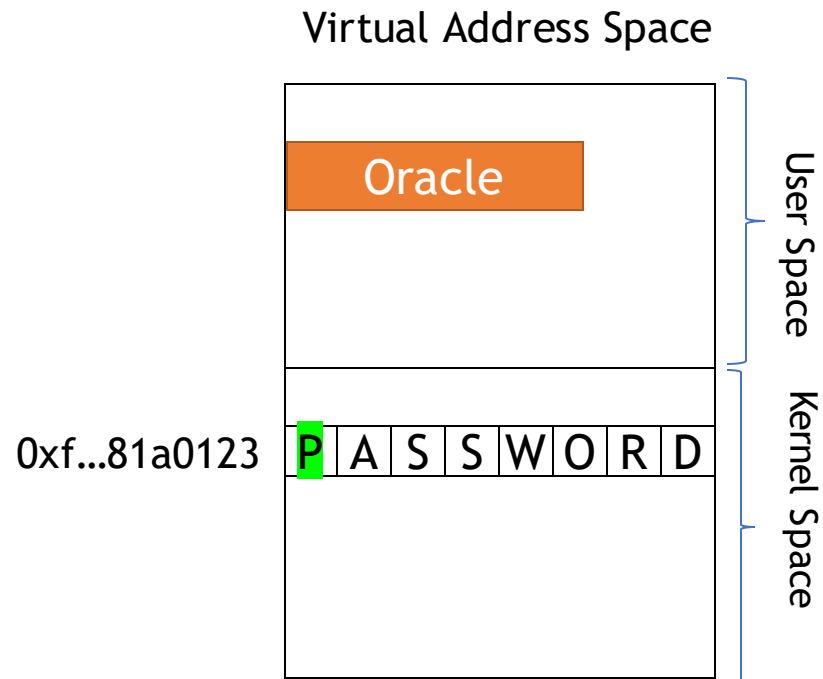
2018: Meltdown Attack?

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



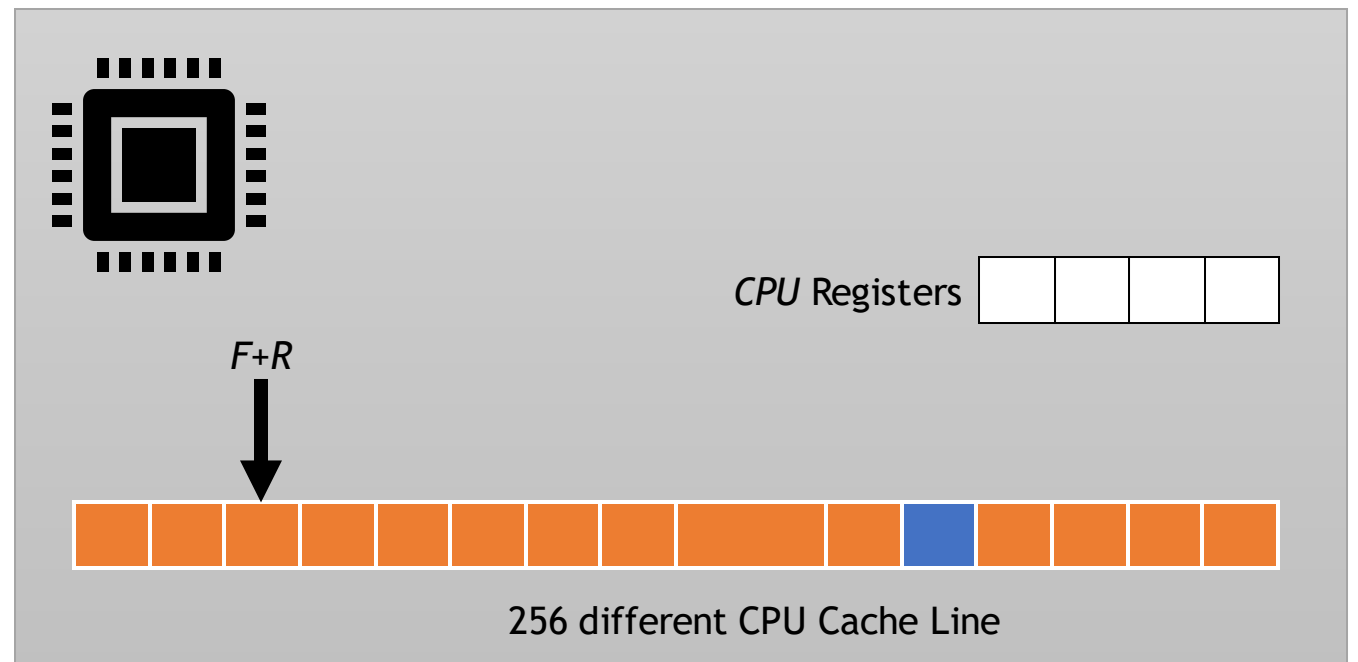
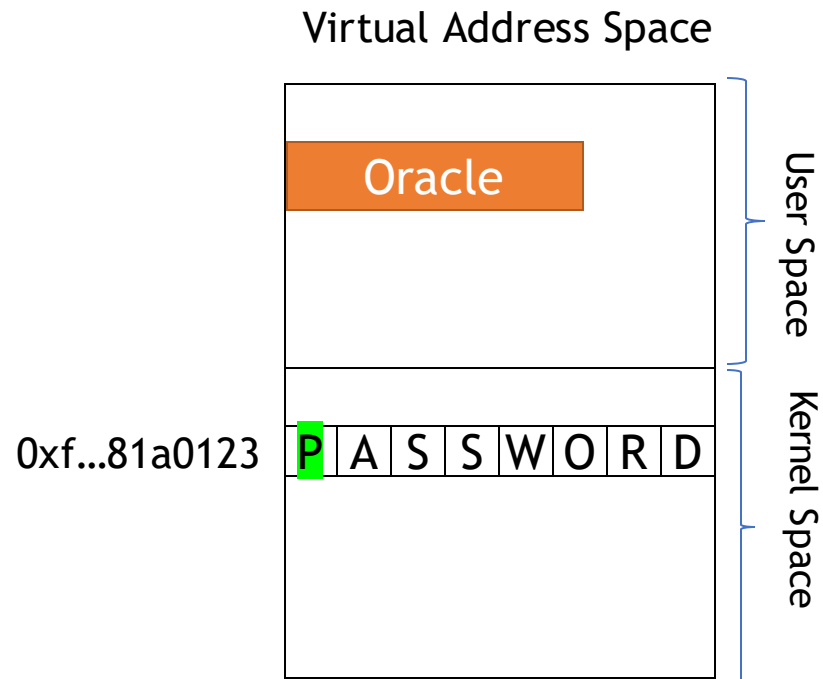
2018: Meltdown Attack?

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



2018: Meltdown Attack?

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

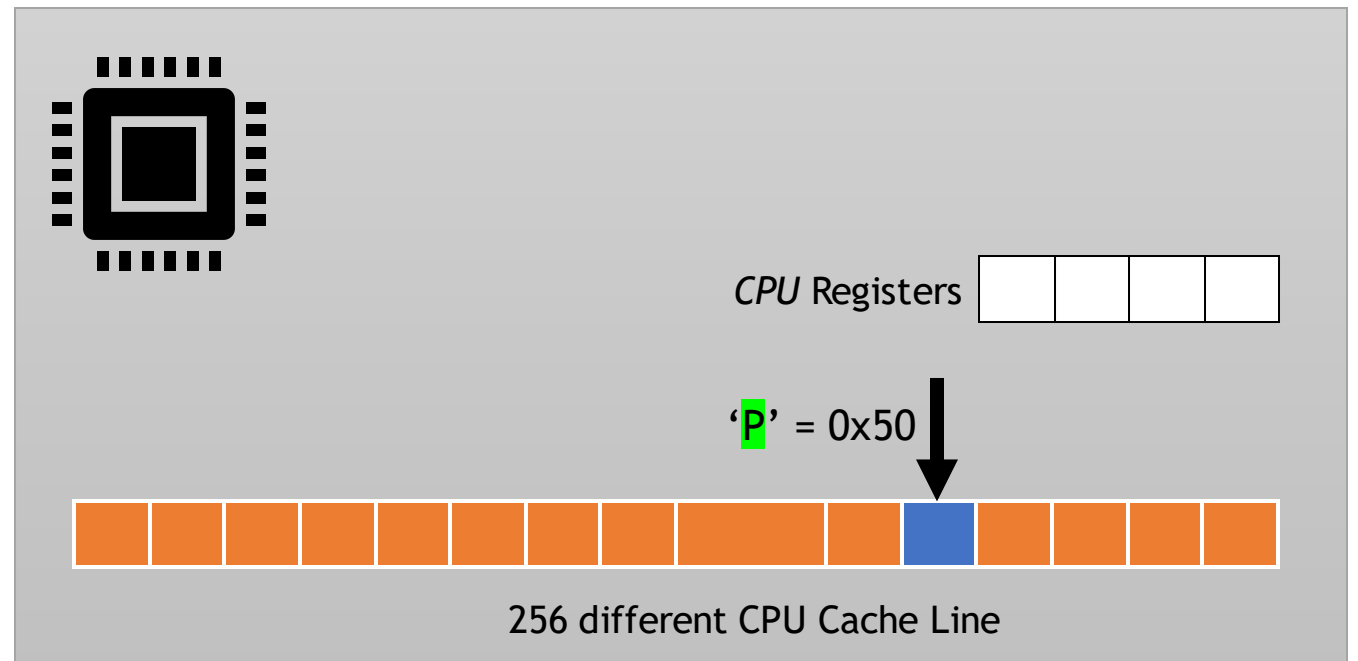
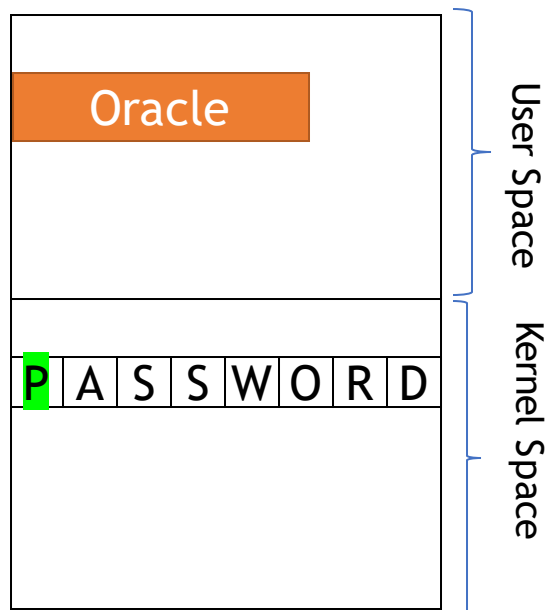


2018: Meltdown Attack?

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

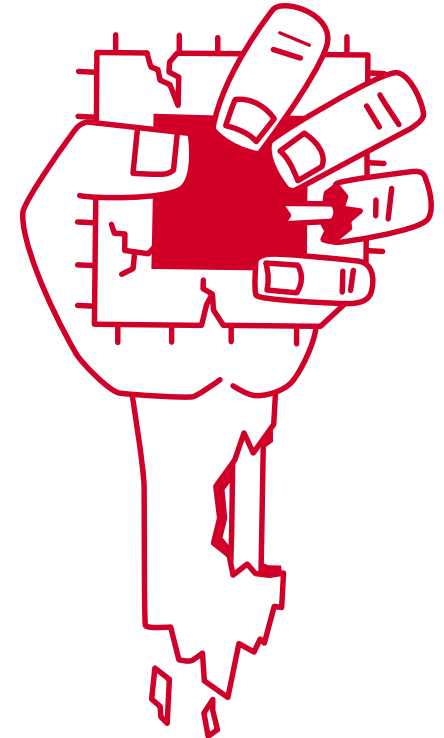


Virtual Address Space

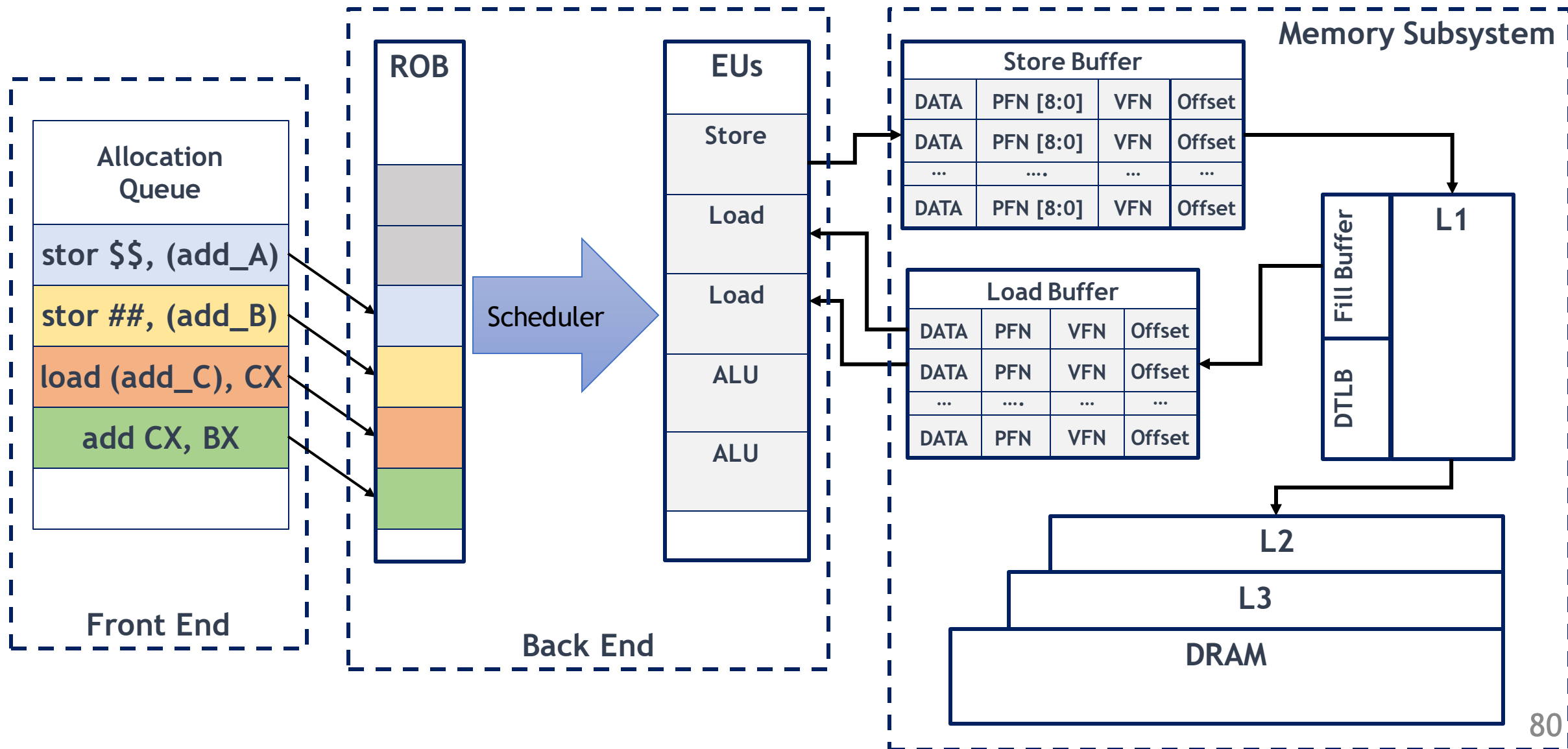


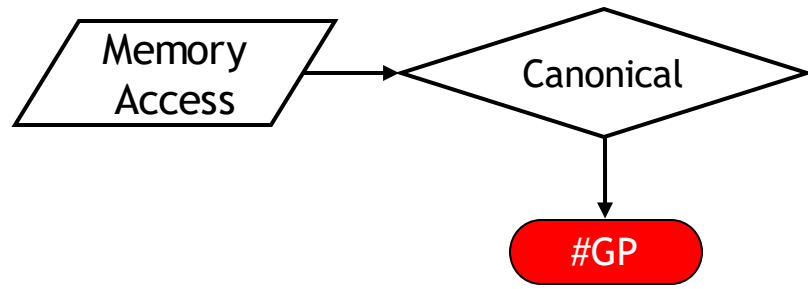
Microarchitecture Data Sampling (MDS)

- Meltdown is fixed but you can still leak on the fix hardware.
- Which part of the CPU leak the data?!
- **Why does it leak?**



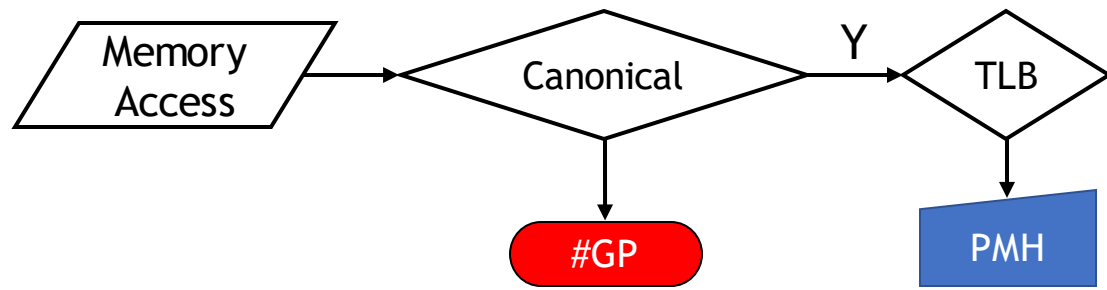
CPU Memory Subsystem - Challenges?





Virtual Address

VFN	Offset
-----	--------

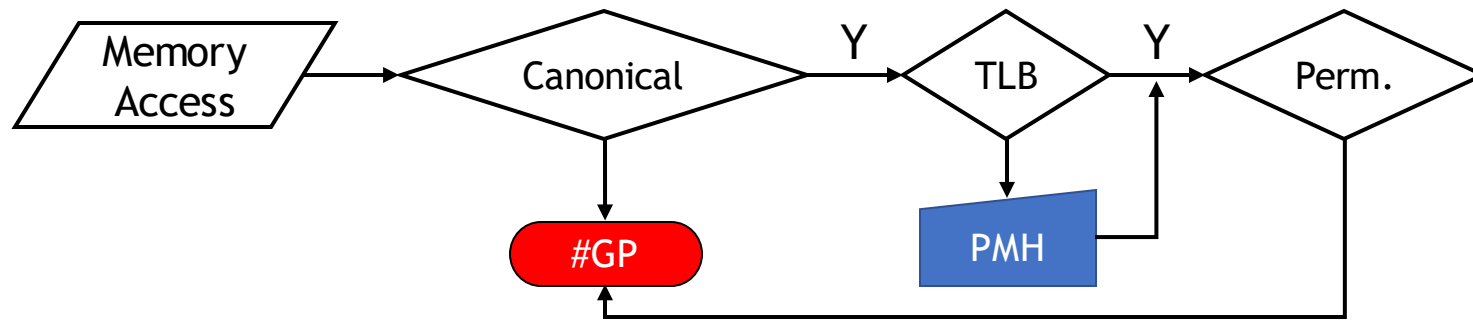


Virtual Address

VFN	Offset
-----	--------

PTE

P	RW	US	...	A	...	Physical Page Number	...
---	----	----	-----	---	-----	----------------------	-----

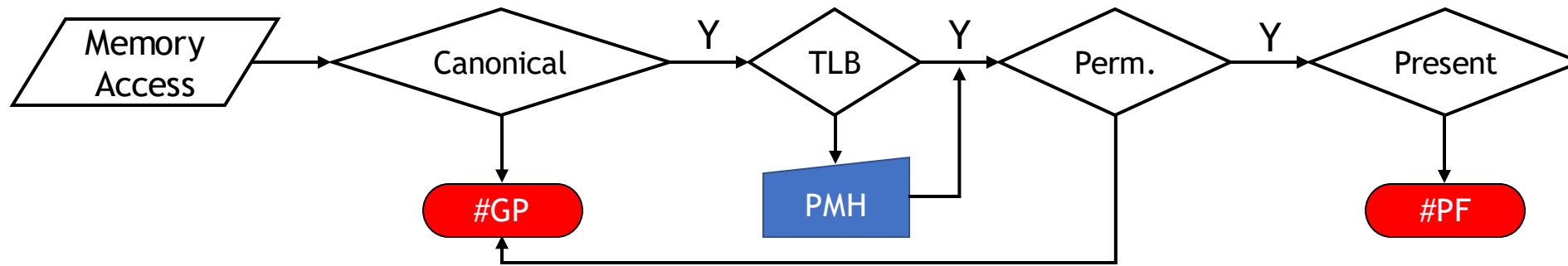


Virtual Address

VFN	Offset
-----	--------

PTE

P	RW	US	...	A	...	Physical Page Number	...
---	----	----	-----	---	-----	----------------------	-----

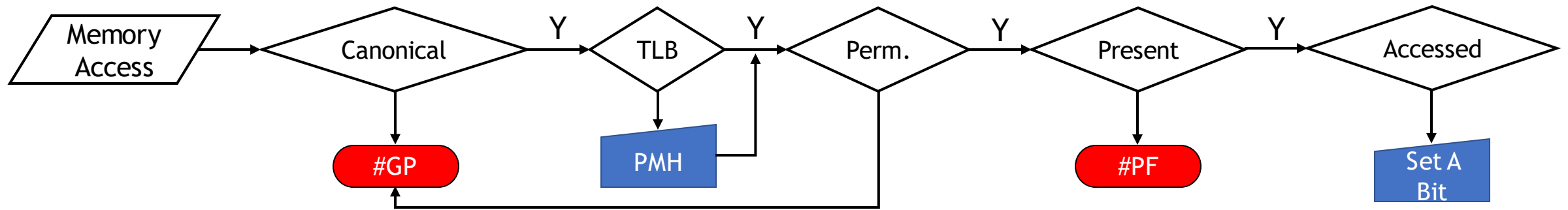


Virtual Address

VFN	Offset
-----	--------

PTE

P	RW	US	...	A	...	Physical Page Number	...
---	----	----	-----	---	-----	----------------------	-----

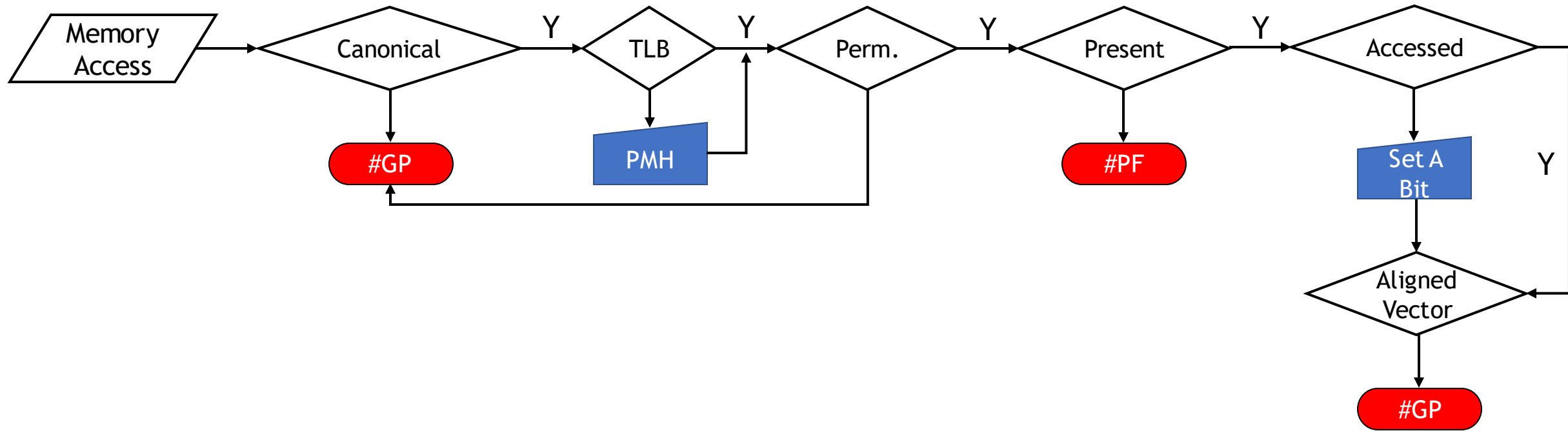


Virtual Address

VFN	Offset
-----	--------

PTE

P	RW	US	...	A	...	Physical Page Number	...
---	----	----	-----	---	-----	----------------------	-----

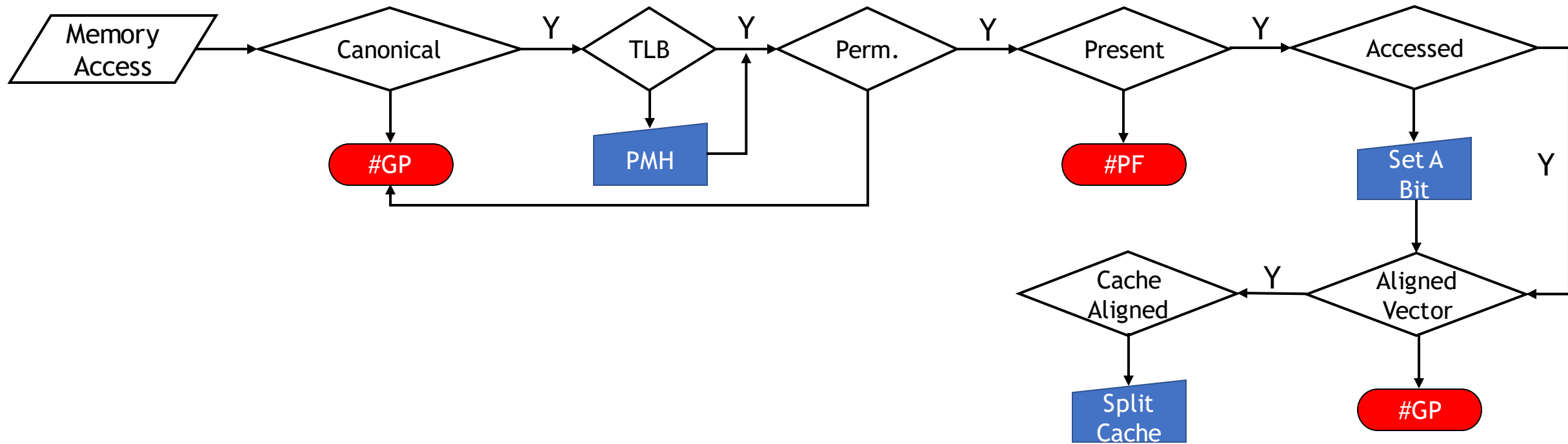


Virtual Address

VFN	Offset
-----	--------

PTE

P	RW	US	...	A	...	Physical Page Number	...
---	----	----	-----	---	-----	----------------------	-----

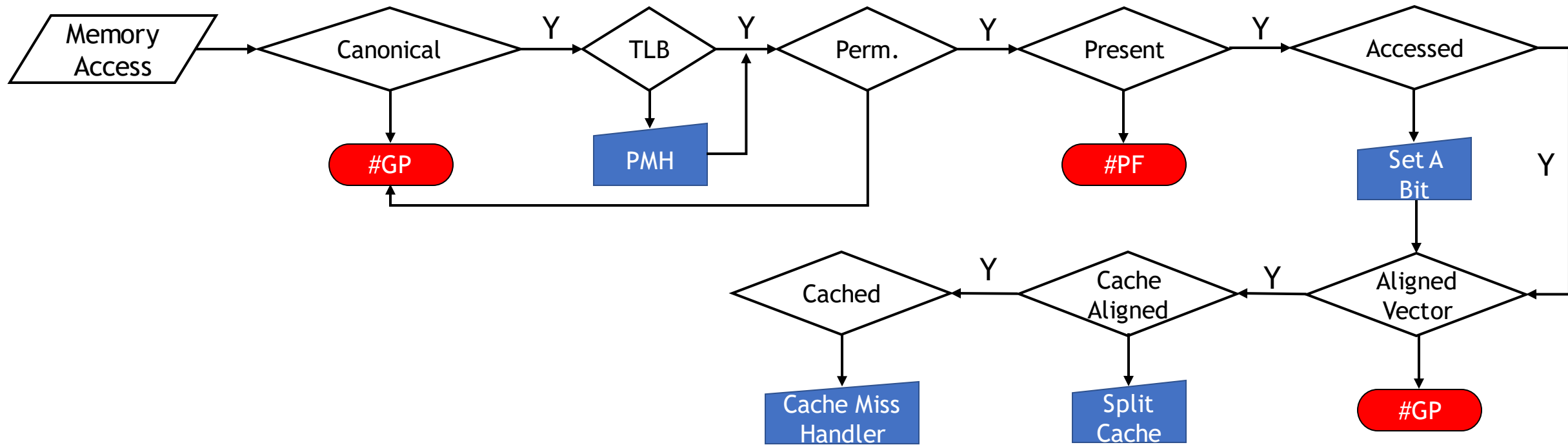


Virtual Address

VFN	Offset
-----	--------

PTE

P	RW	US	...	A	...	Physical Page Number	...
---	----	----	-----	---	-----	----------------------	-----

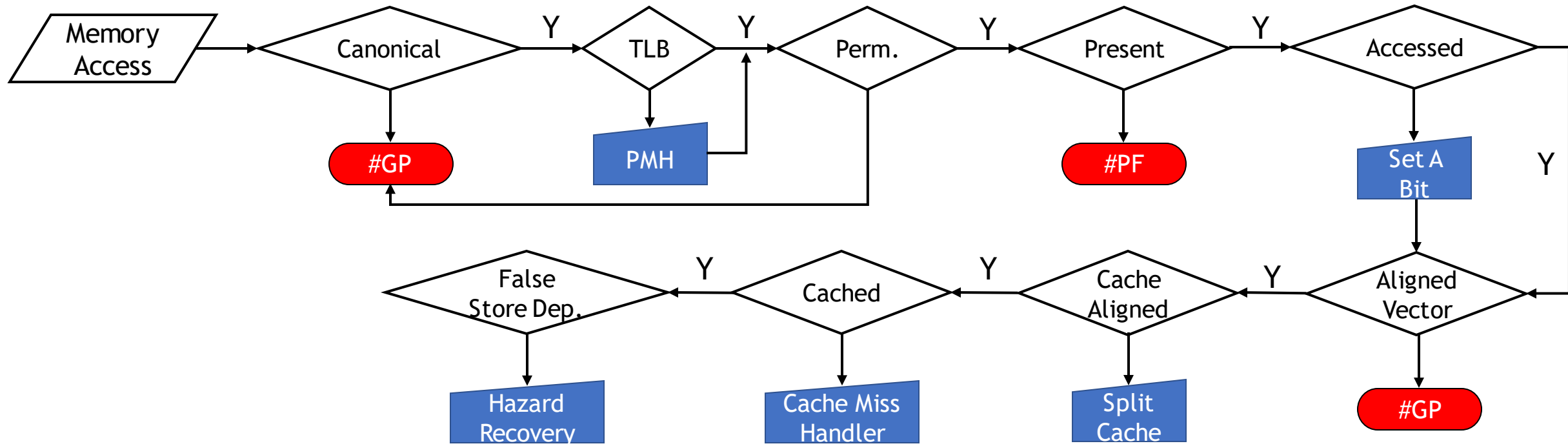


Virtual Address

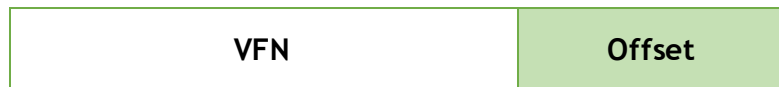
VFN	Offset
-----	--------

PTE

P	RW	US	...	A	...	Physical Page Number	...
---	----	----	-----	---	-----	----------------------	-----

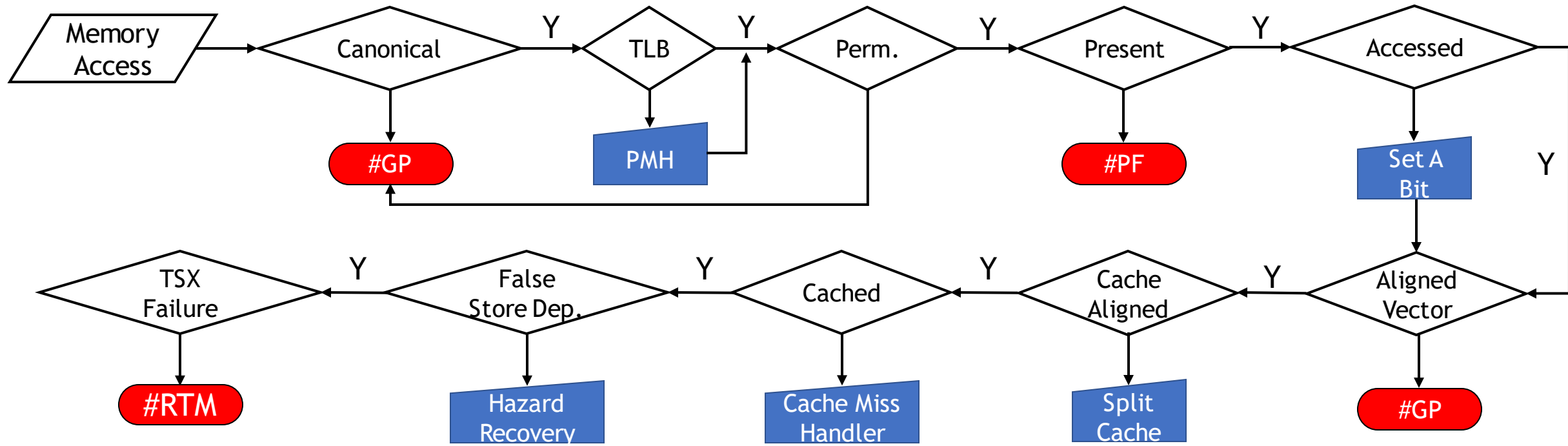


Virtual Address



PTE





Virtual Address



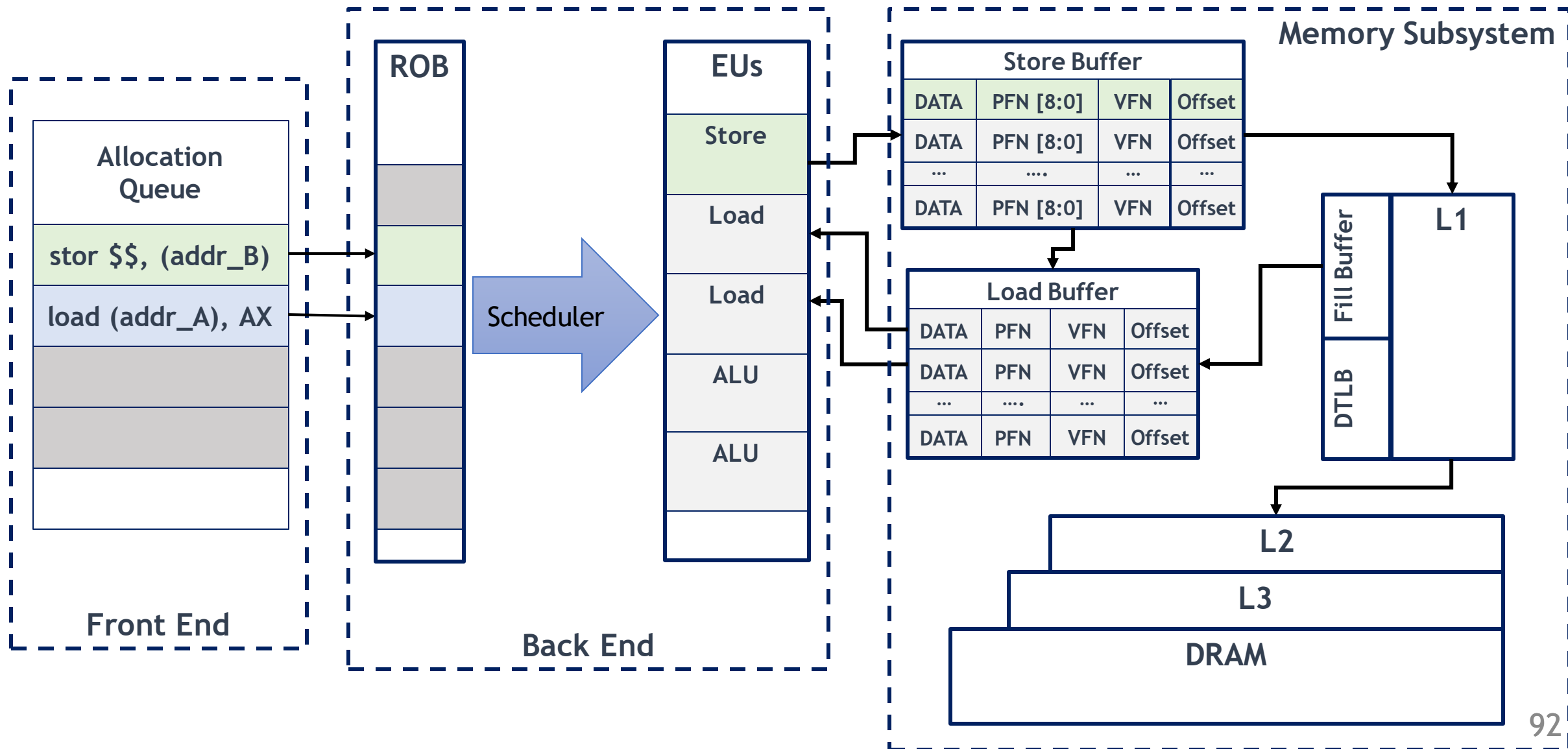
PTE



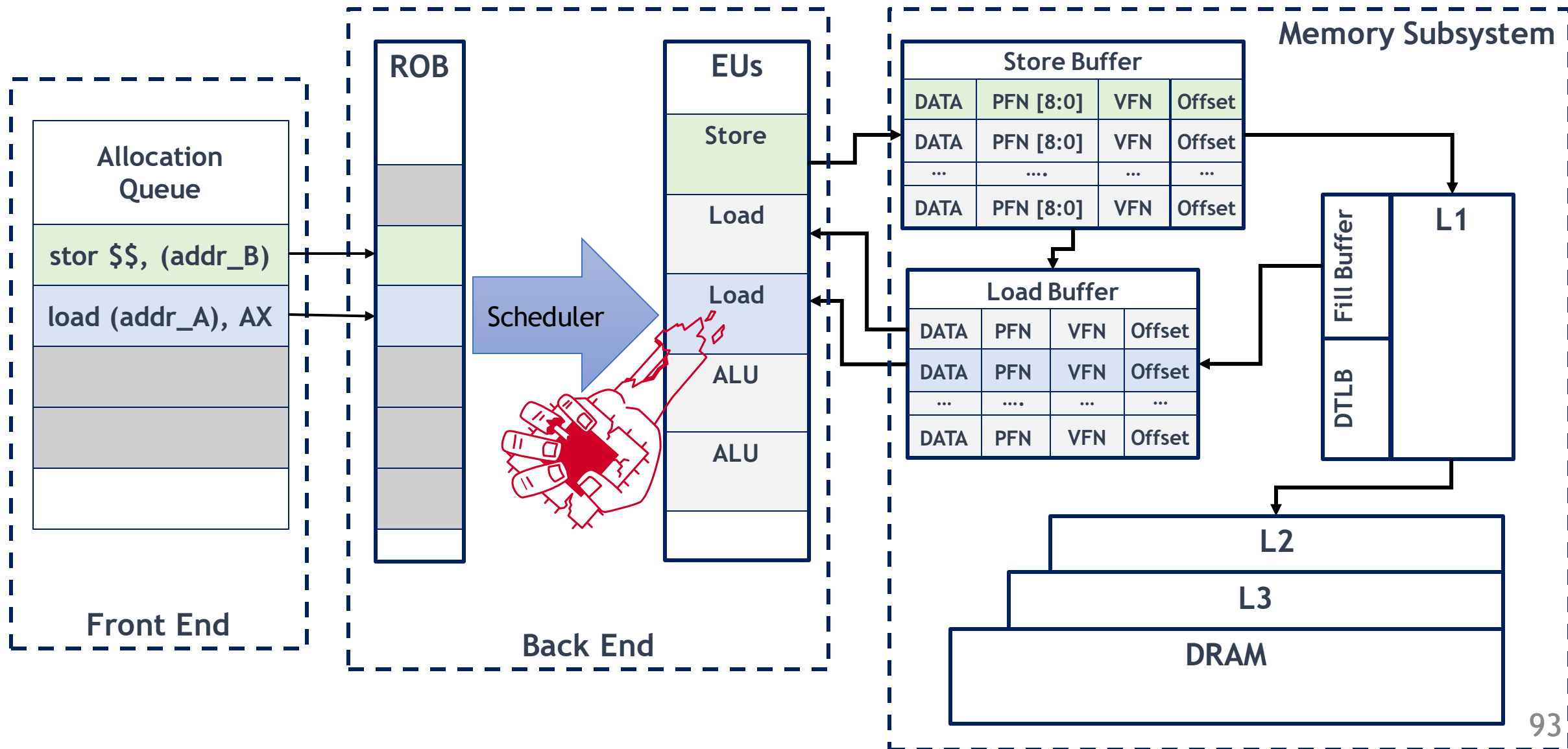
Fault VS. Assist Dilemma

- Microcode Assists: The CPU executes an internal event handler to service complex instructions/operations
- Fault (#GP, #PF, #RTM): An assist that run a software-based callback

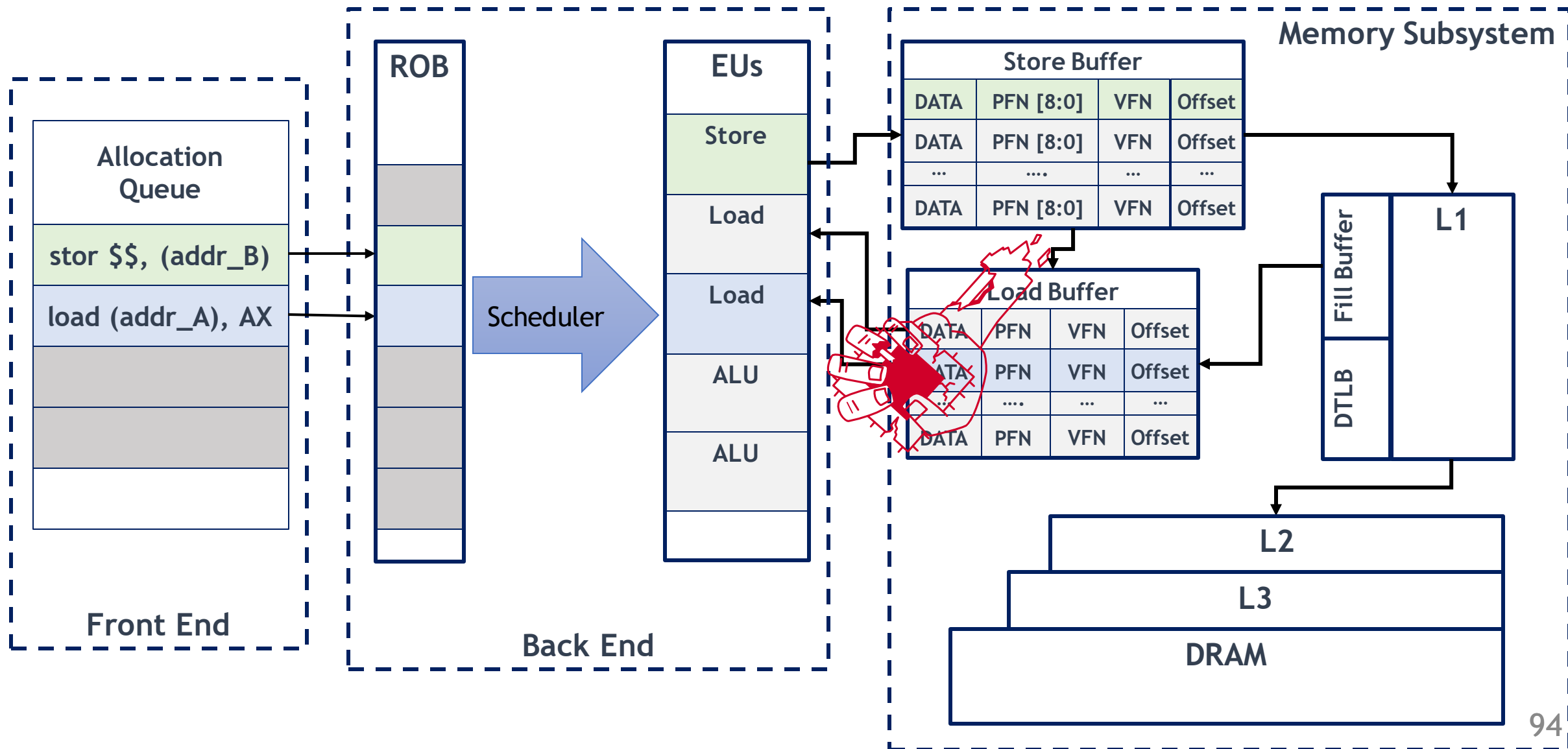
CPU Memory Subsystem - Hazard Recovery



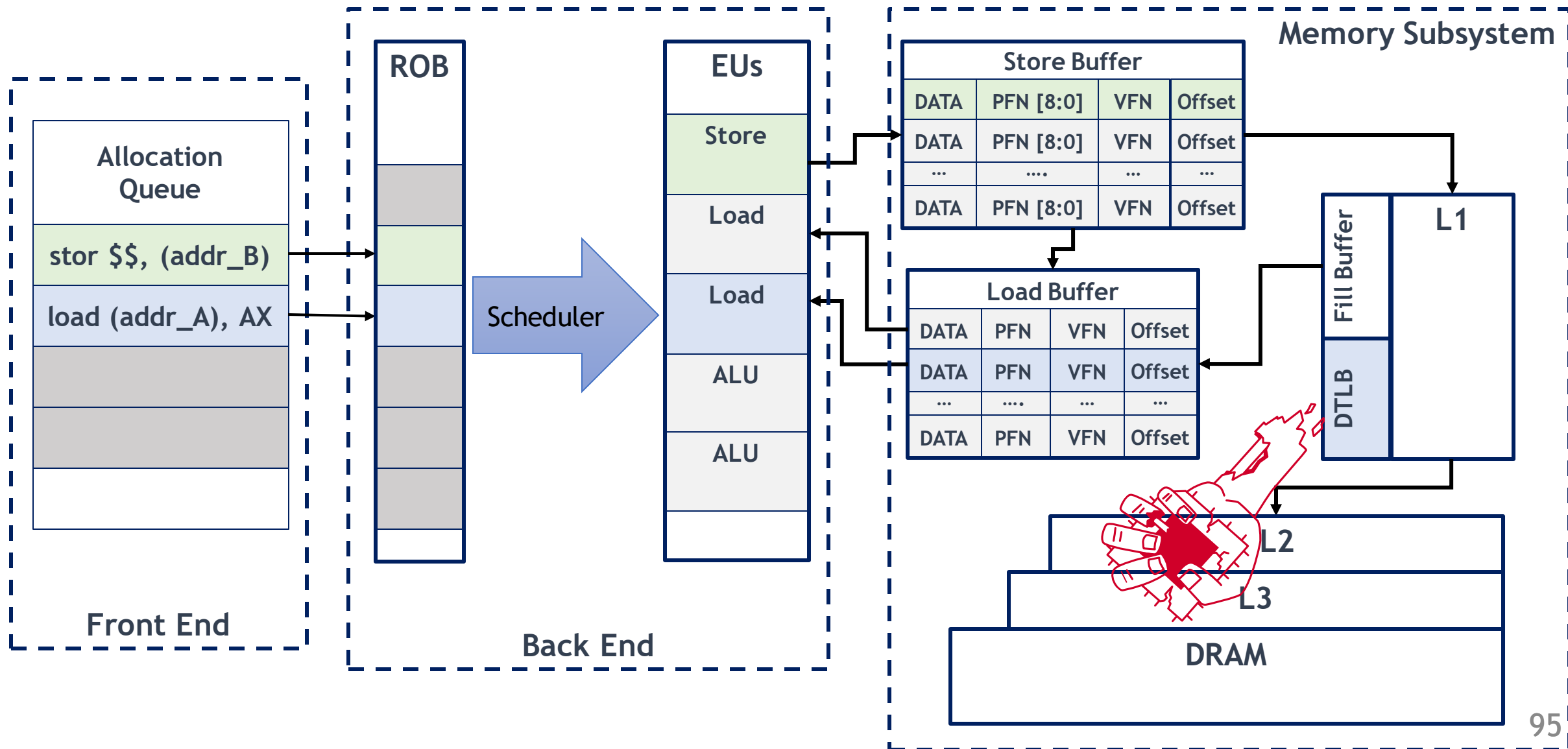
CPU Memory Subsystem - Hazard Recovery



CPU Memory Subsystem - Hazard Recovery



CPU Memory Subsystem - Hazard Recovery



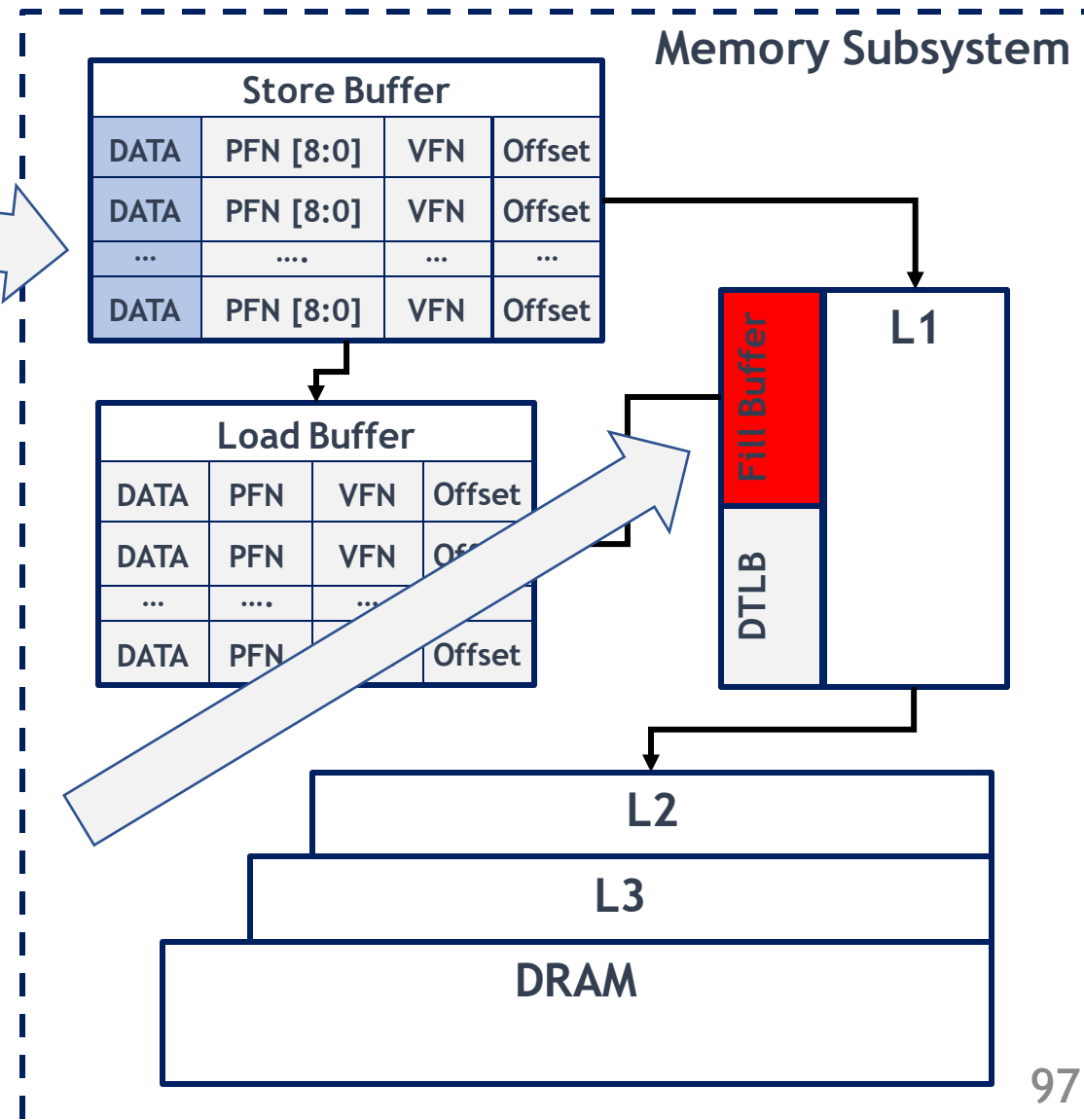
MDS Attacks (ZombieLoad, RIDL, Fallout, ...)

- The CPU must flush the pipeline before executing an assist.
- Upon an Exception/Fault/Assist on a Load, Intel CPUs:
 - Execute the load until the last stage.
 - Flush the pipeline at the retirement stage (Cheap Recovery Logic).
 - Continue the load with some data to reach the retirement stage.
- Which data?

CPU Memory Subsystem - Leaky Buffers

Fallout

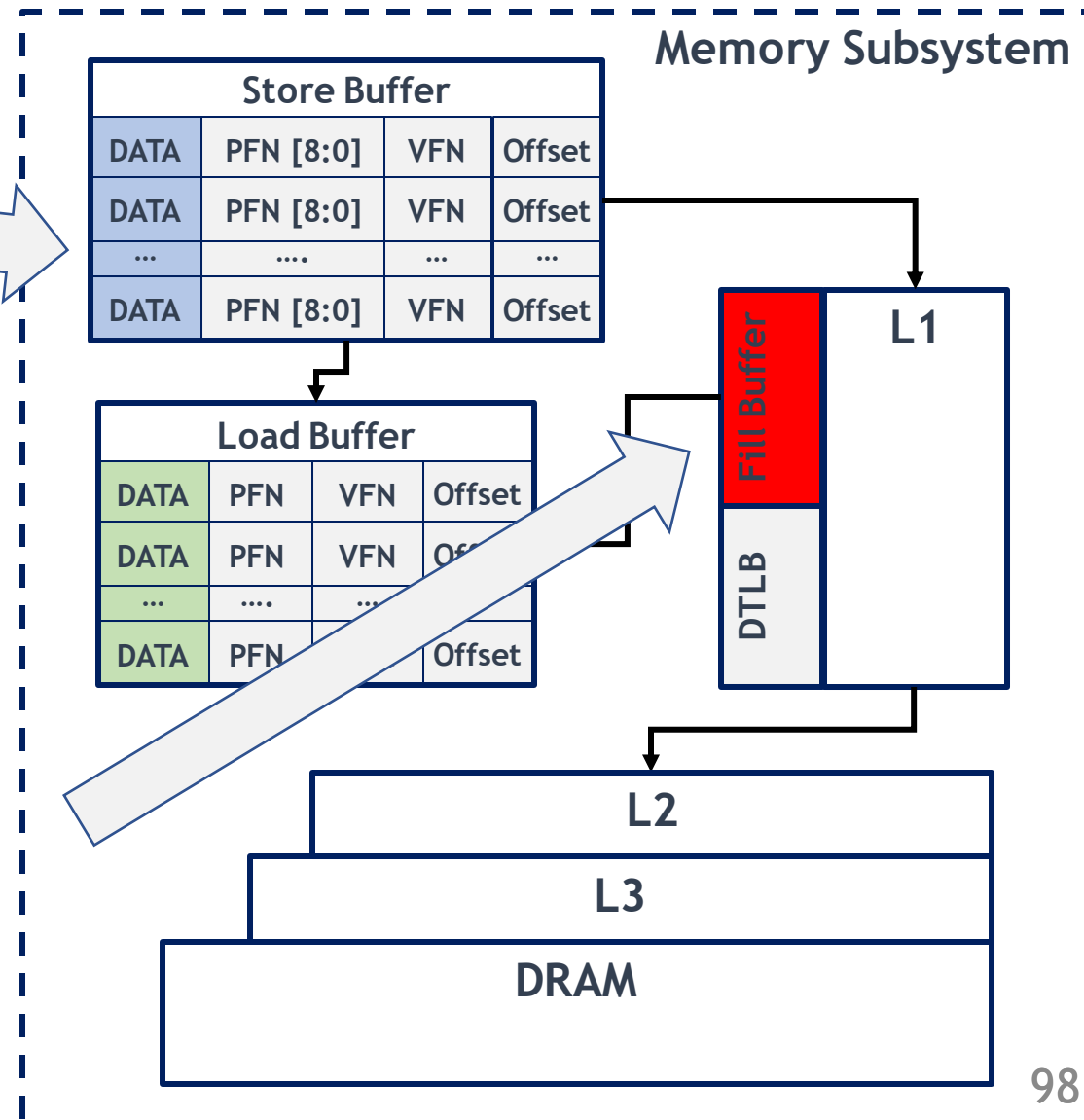
ZombieLoad



CPU Memory Subsystem - Leaky Buffers

Fallout

ZombieLoad



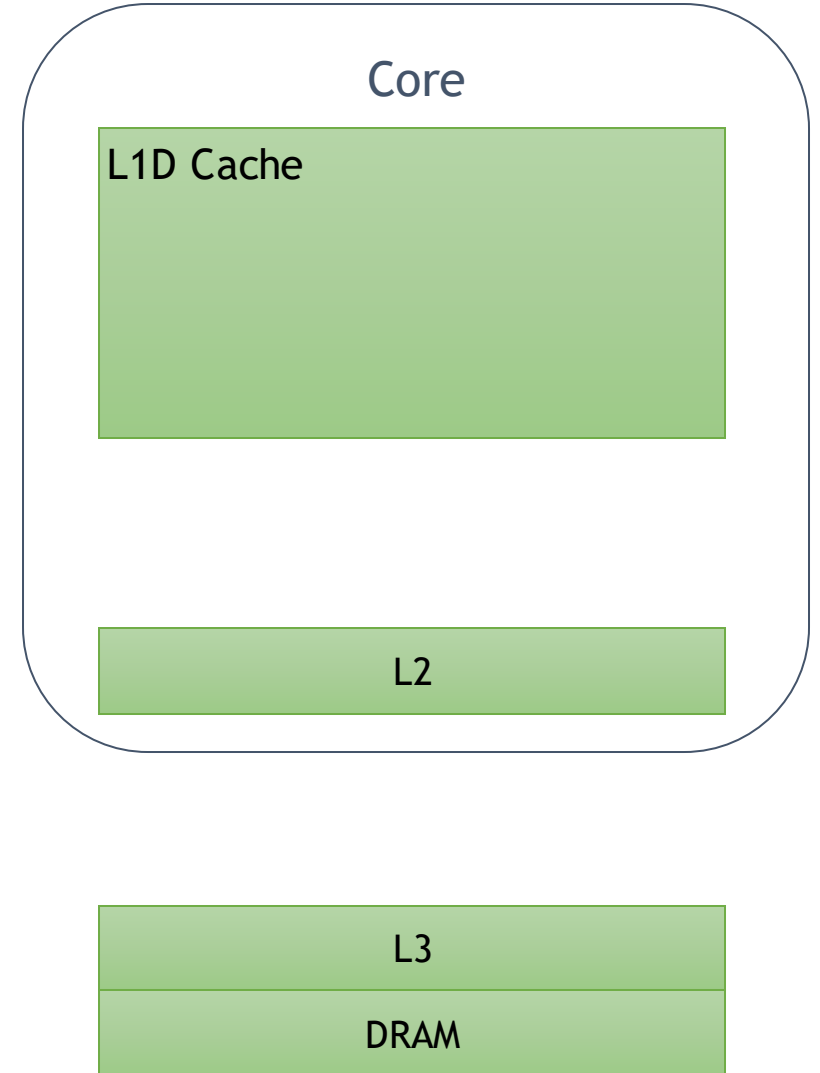
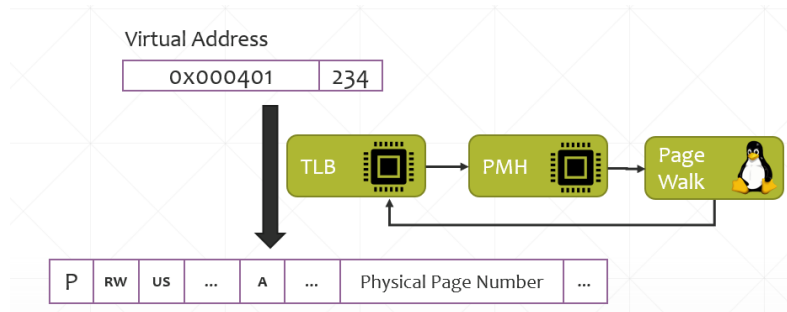
MDS Attacks (ZombieLoad, RIDL, Fallout, ...)

- The CPU must flush the pipeline before executing an assist.
- Upon an Exception/Fault/Assist on a Load, Intel CPUs:
 - Execute the load until the last stage.
 - Flush the pipeline at the retirement stage (Cheap Recovery Logic).
 - Continue the load with some data to reach the retirement stage.
- Which data? (Fill buffer, Store Buffer, Load Buffer)
- Which one will be leaked first?

ZombieLoad Attack

```
mov 0x401234, %rsi
```

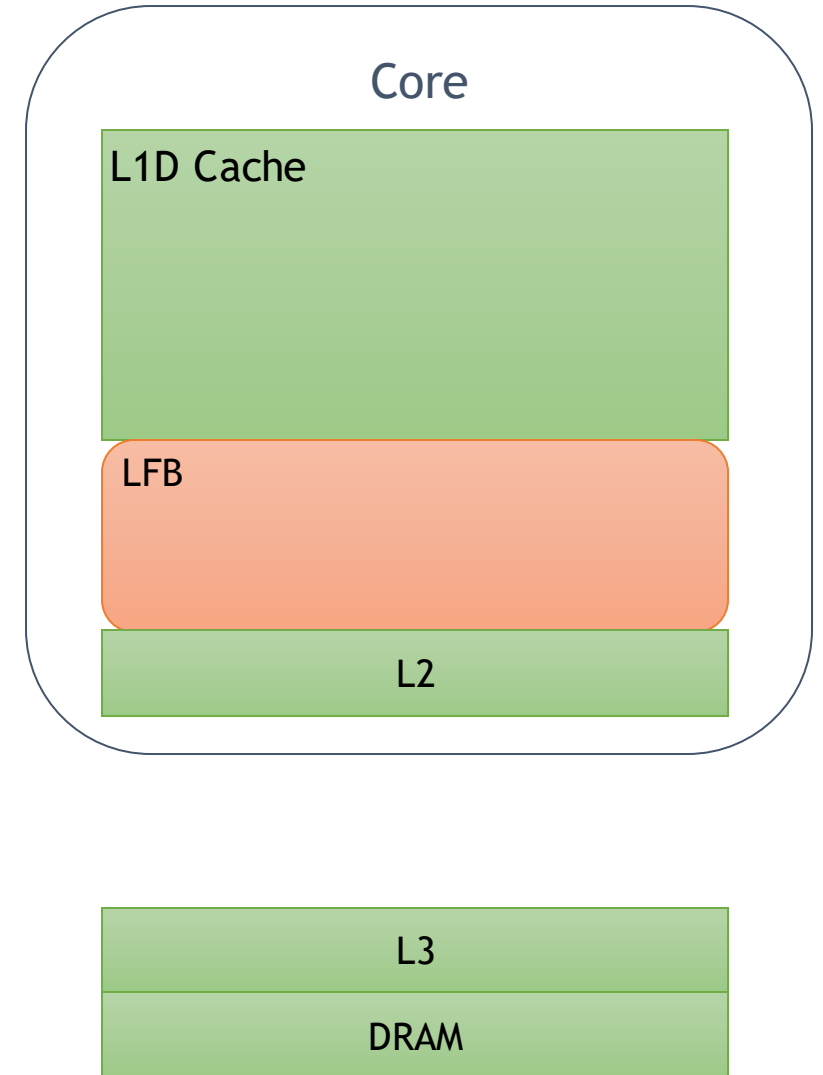
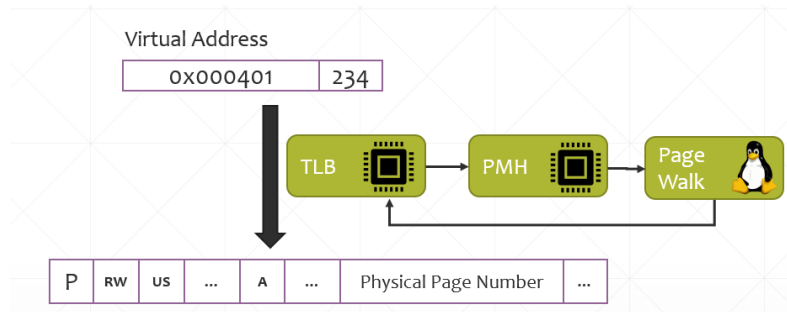
```
mov (%rsi), %rax
```



ZombieLoad Attack

```
mov 0x401234, %rsi
```

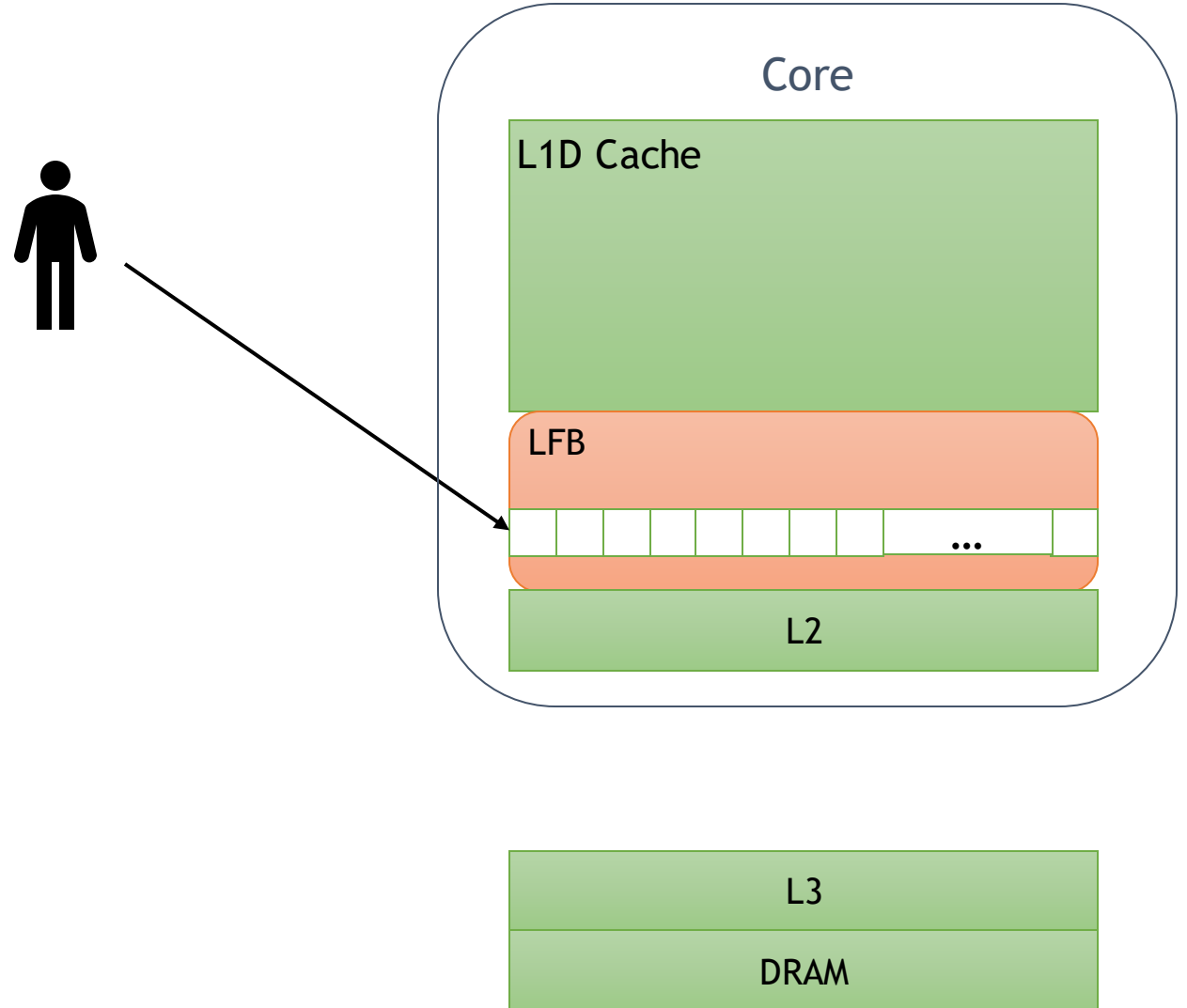
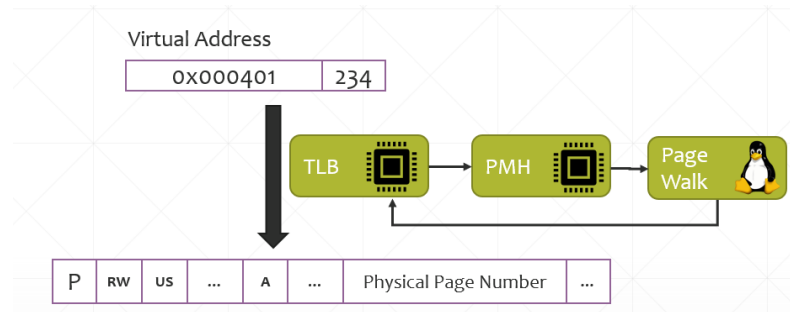
```
mov (%rsi), %rax
```



ZombieLoad Attack

```
mov 0x401234, %rsi
```

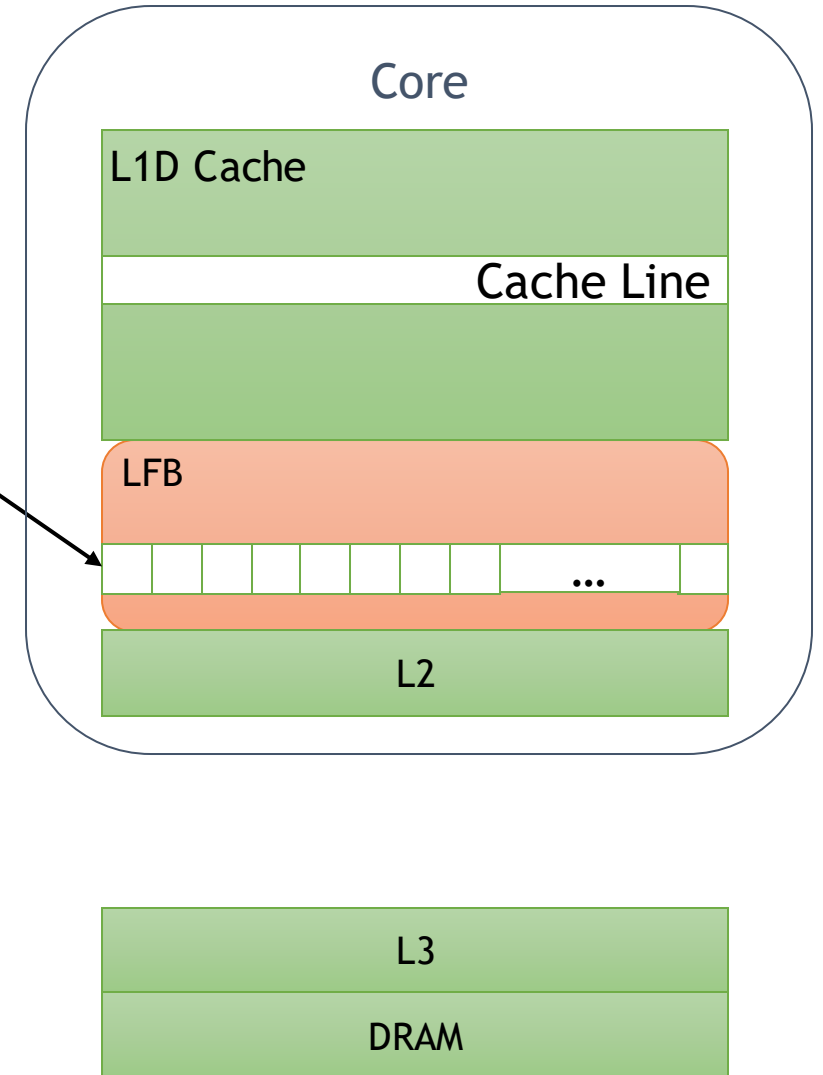
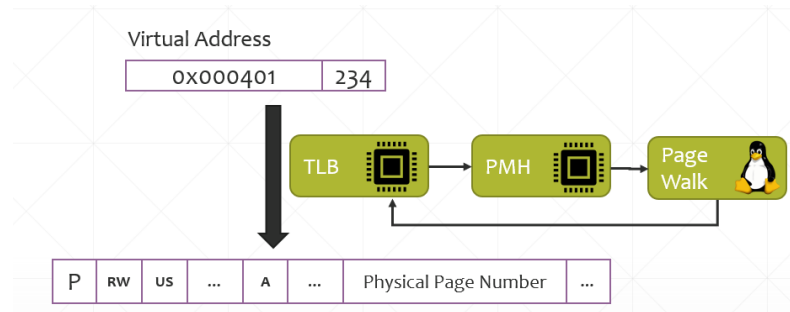
```
mov (%rsi), %rax
```



ZombieLoad Attack

```
mov 0x401234, %rsi
```

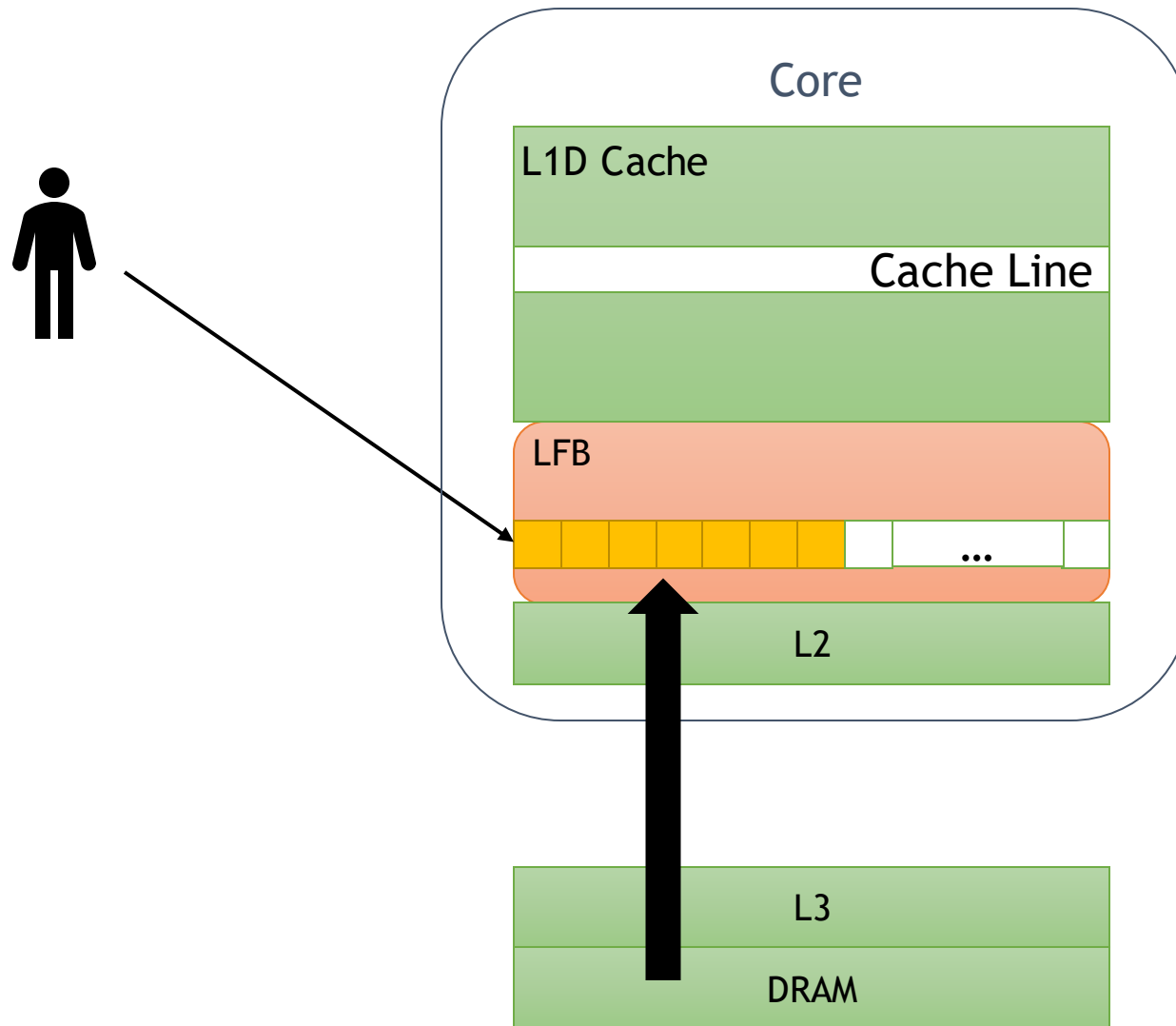
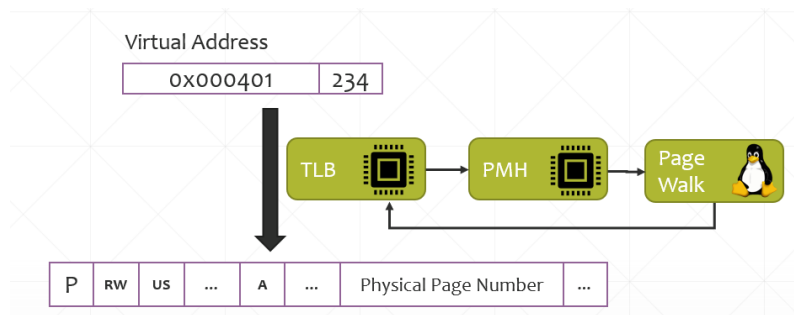
```
mov (%rsi), %rax
```



ZombieLoad Attack

```
mov 0x401234, %rsi
```

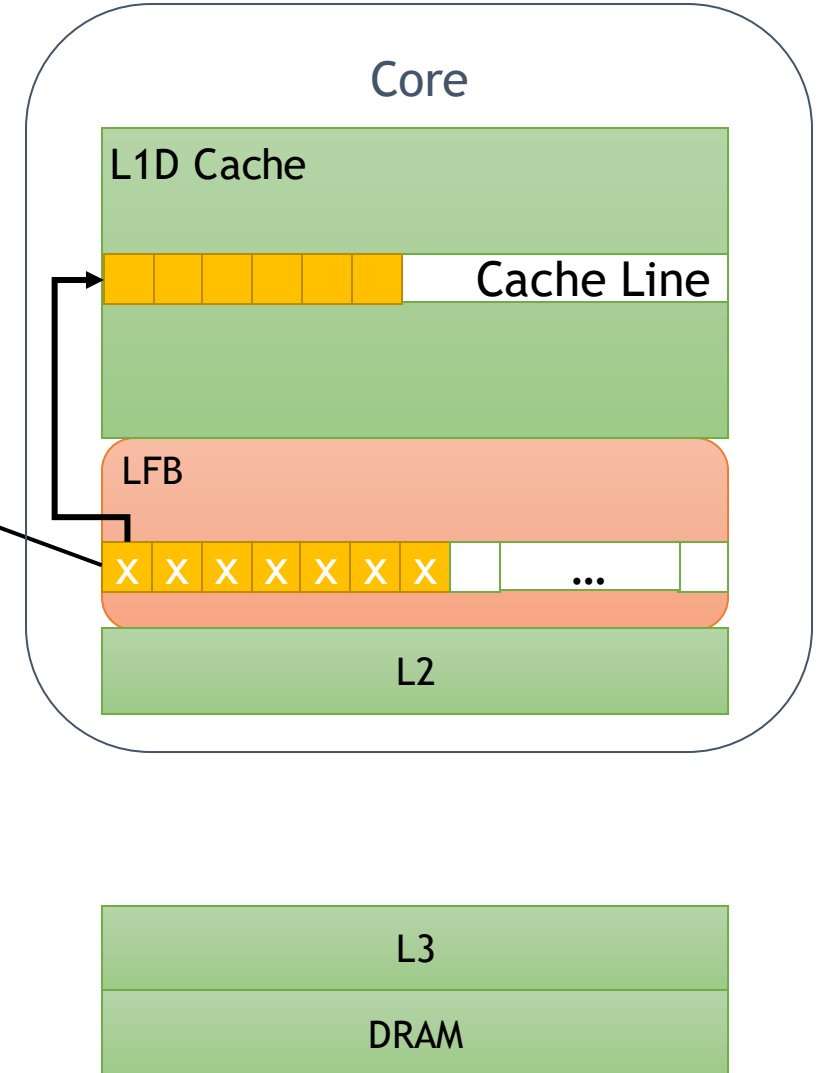
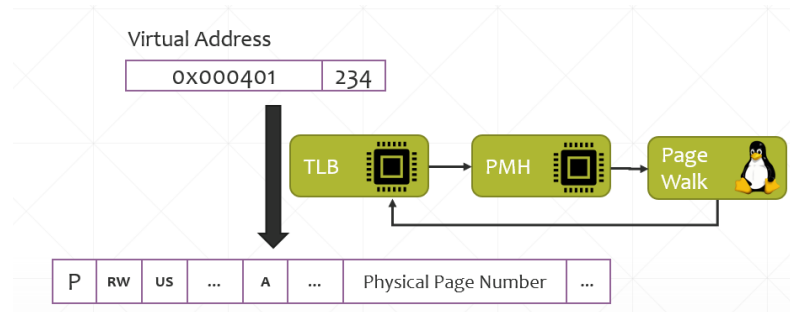
```
mov (%rsi), %rax
```



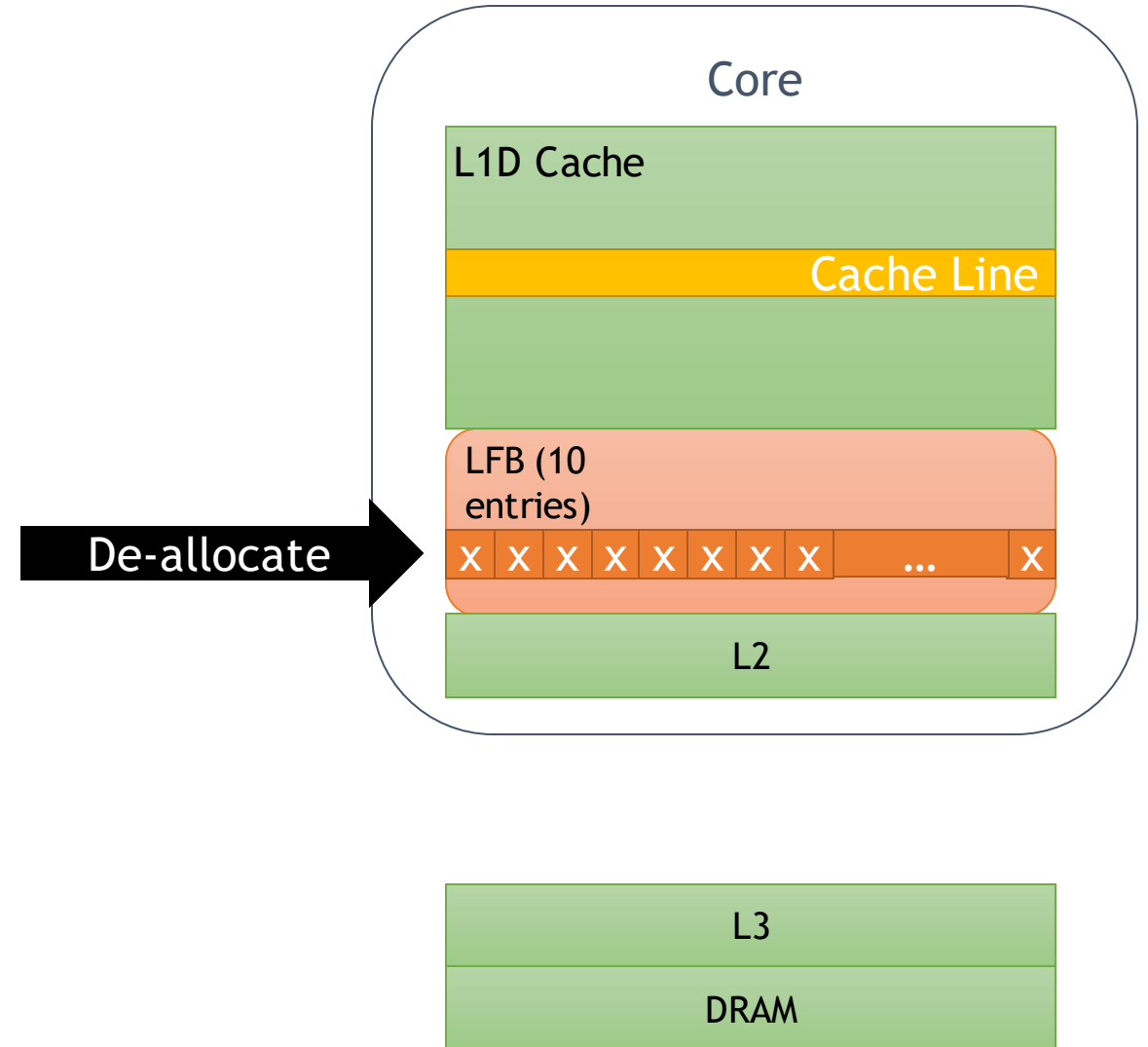
ZombieLoad Attack

```
mov 0x401234, %rsi
```

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



ZombieLoad Attack

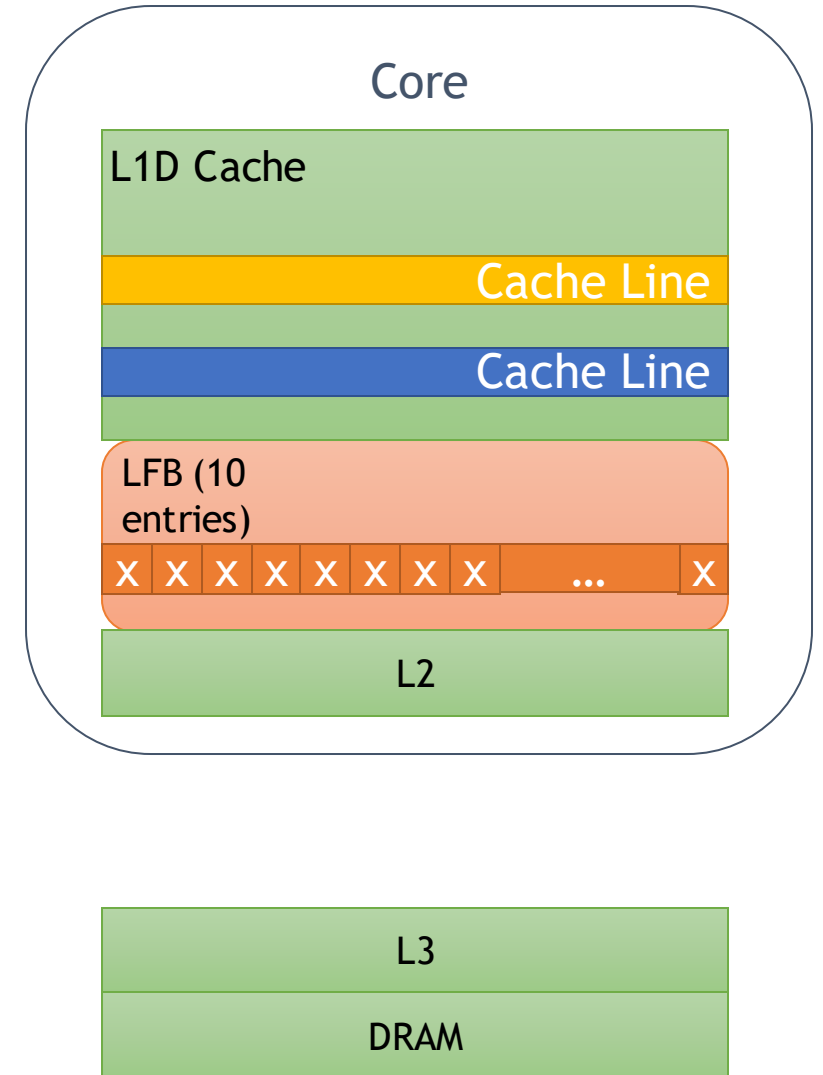


ZombieLoad Attack

P	RW	US	...	A	...	Physical Page Number	...
---	----	----	-----	---	-----	----------------------	-----



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

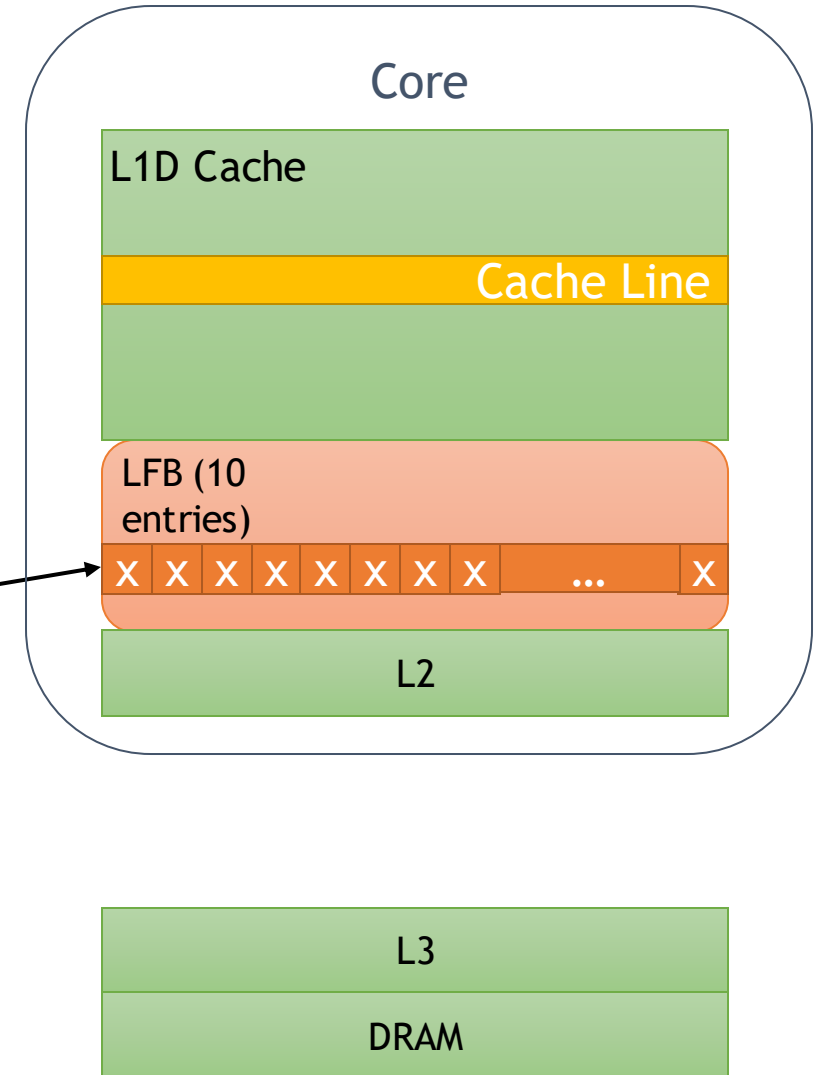


ZombieLoad Attack

P	RW	US	...	A	...	Physical Page Number	...
---	----	----	-----	---	-----	----------------------	-----

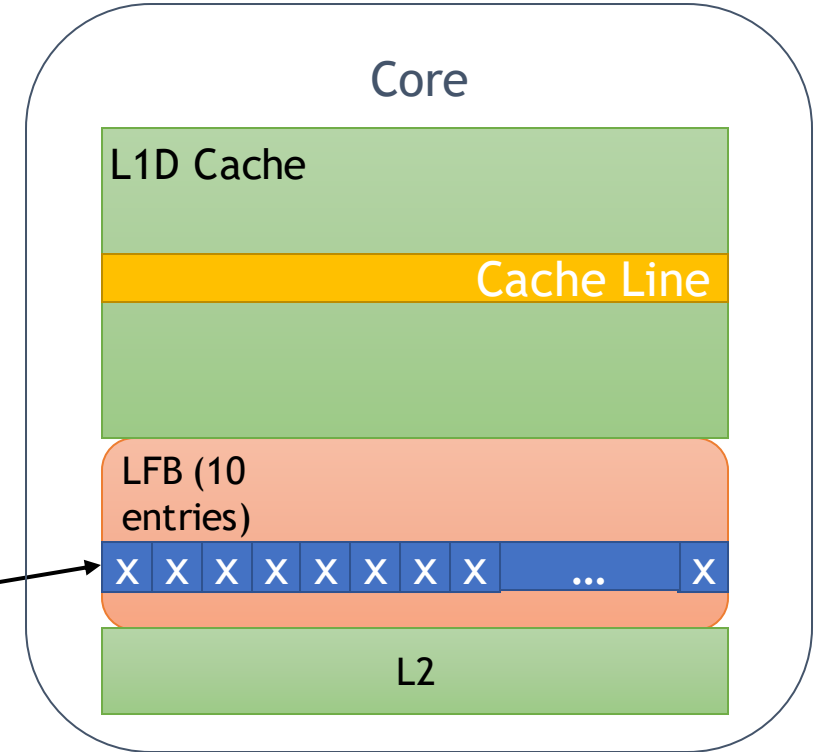


```
char secret = *(char *) 0xffffffff81a0123;  
printf("%c\n", secret);
```



ZombieLoad Attack

P	RW	US	...	A	...	Physical Page Number	...
---	----	----	-----	---	-----	----------------------	-----



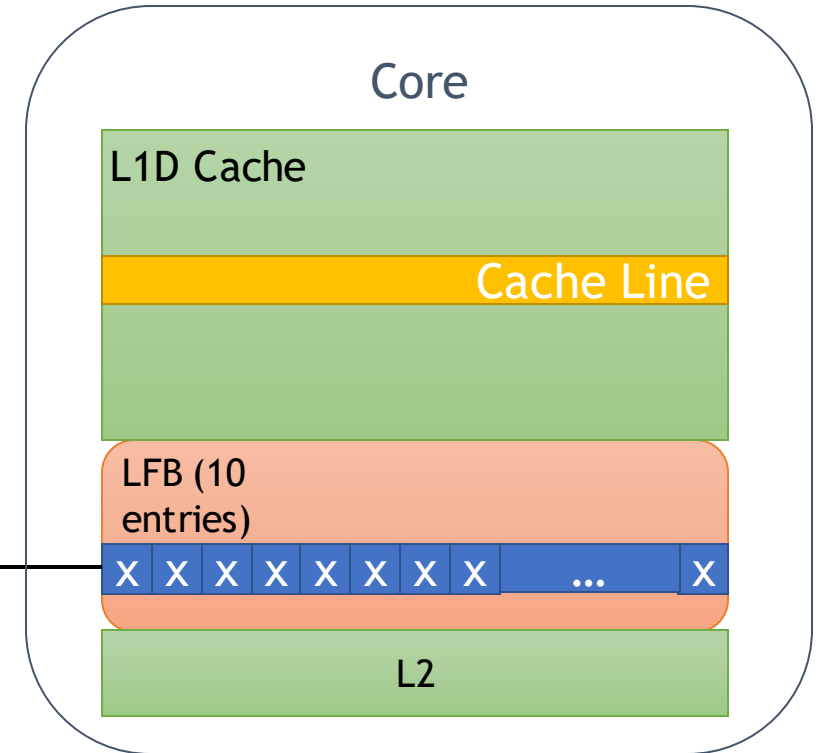
```
char secret = *(char *) 0xffffffff81a0123;  
printf("%c\n", secret);
```

ZombieLoad Attack

P	RW	US	...	A	...	Physical Page Number	...
---	----	----	-----	---	-----	----------------------	-----

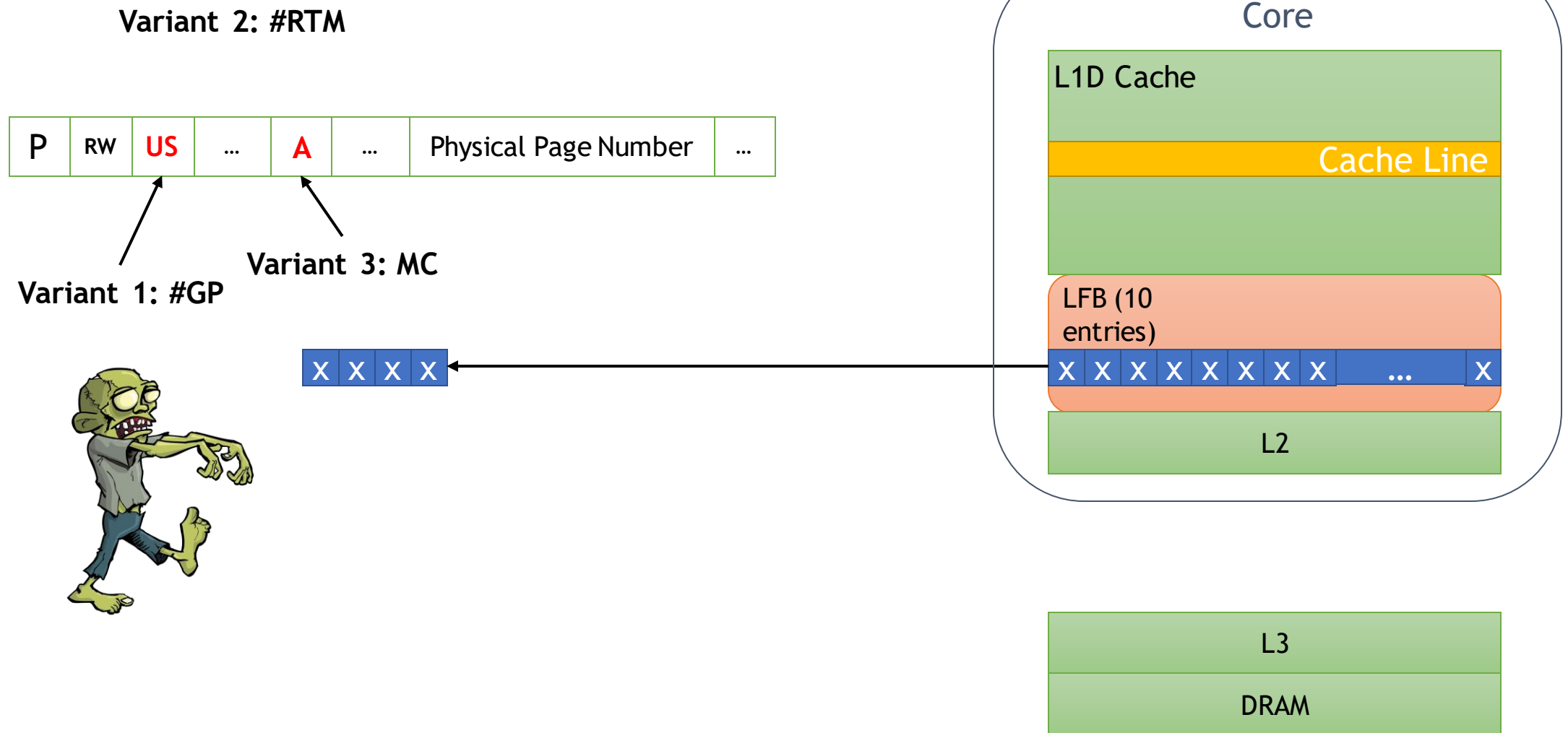


X X X X



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

ZombieLoad Attack

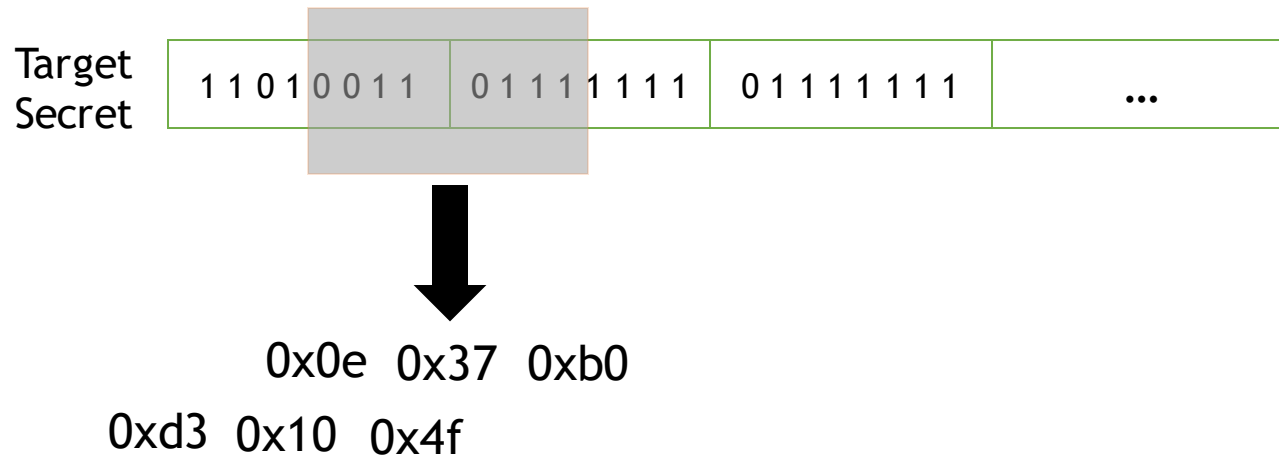


Meltdown-style Attacks

	Page Number			Page Offset		
Meltdown	51	Physical	12	11		0
	47	Virtual	12			
Foreshadow	51	Physical	12	11		0
	47	Virtual	12			
Fallout	51	Physical	12	11		0
	47	Virtual	12			
ZombieLoad	51	Physical	12	11	6	5
	47	Virtual	12			

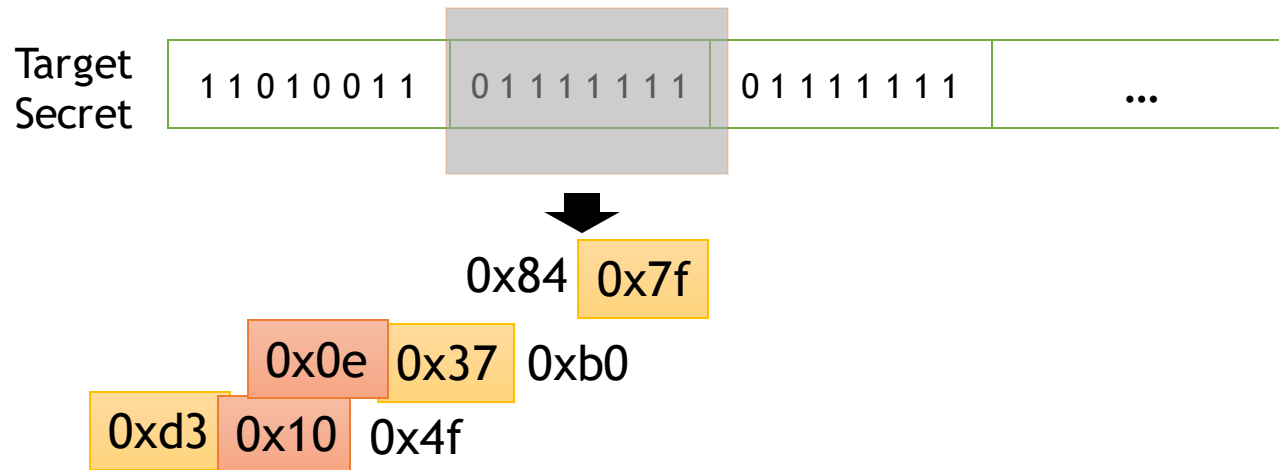
Data Sampling - Domino Attack

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



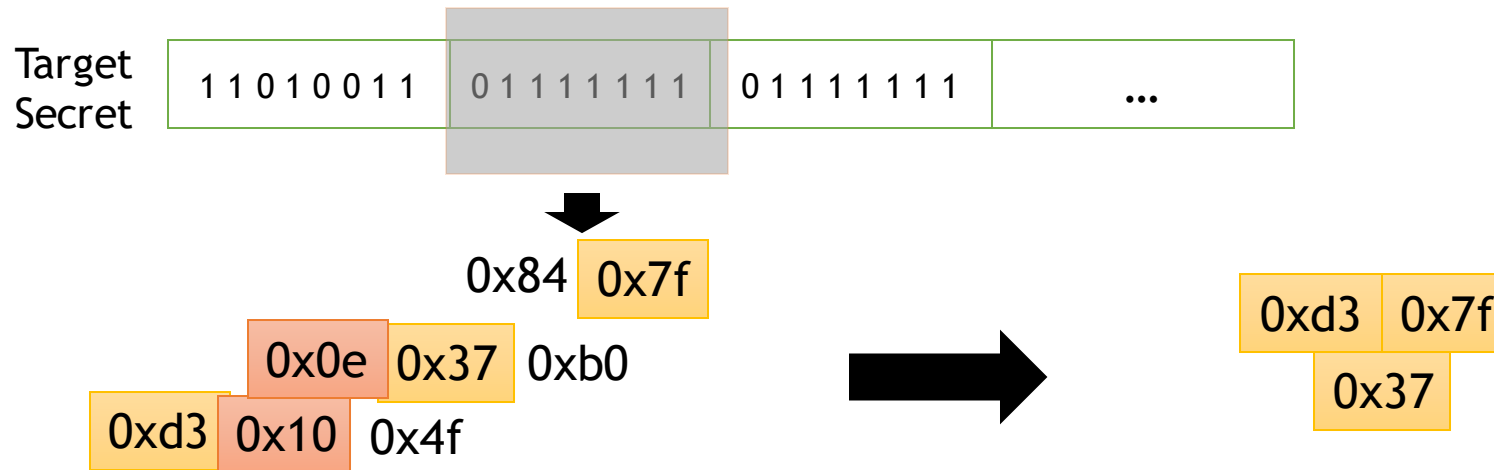
Data Sampling - Domino Attack

- 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



File Edit View Bookmarks Settings Help

michael@hp /tmp/zombieload %

ZombieLoad - 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
```

ZombieLoad - 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
```

ZombieLoad - 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
```

ZombieLoad - Recovering Intel SGX Sealing Key

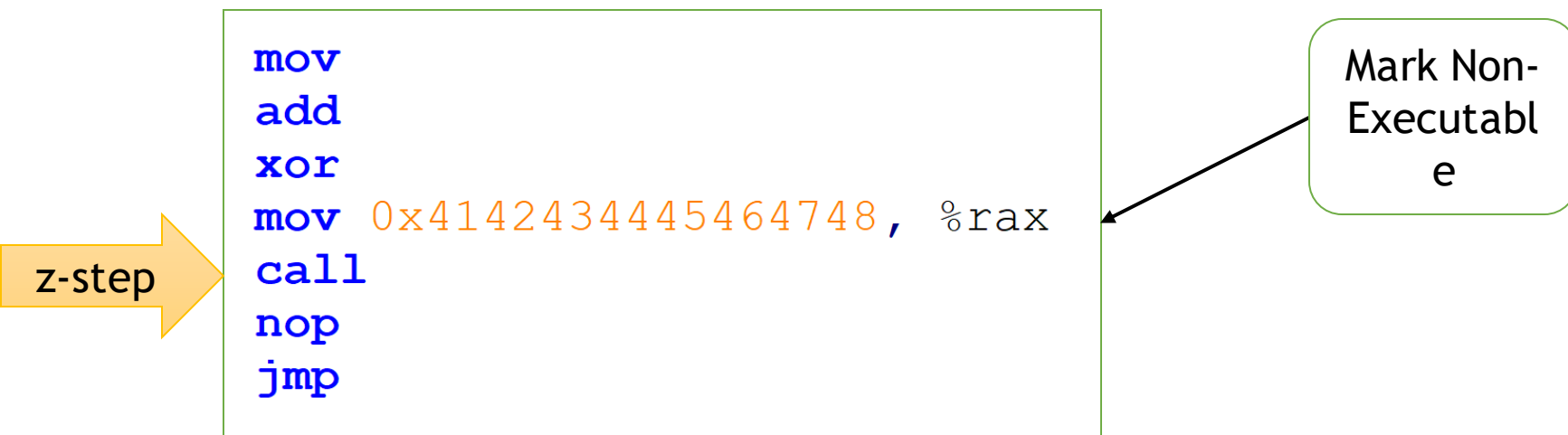
- Intel SGX allow developers to have hardware support for TEE
- Malicious OS is part of the threat model
- We can read register values of a trusted enclave with help of a malicious OS

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



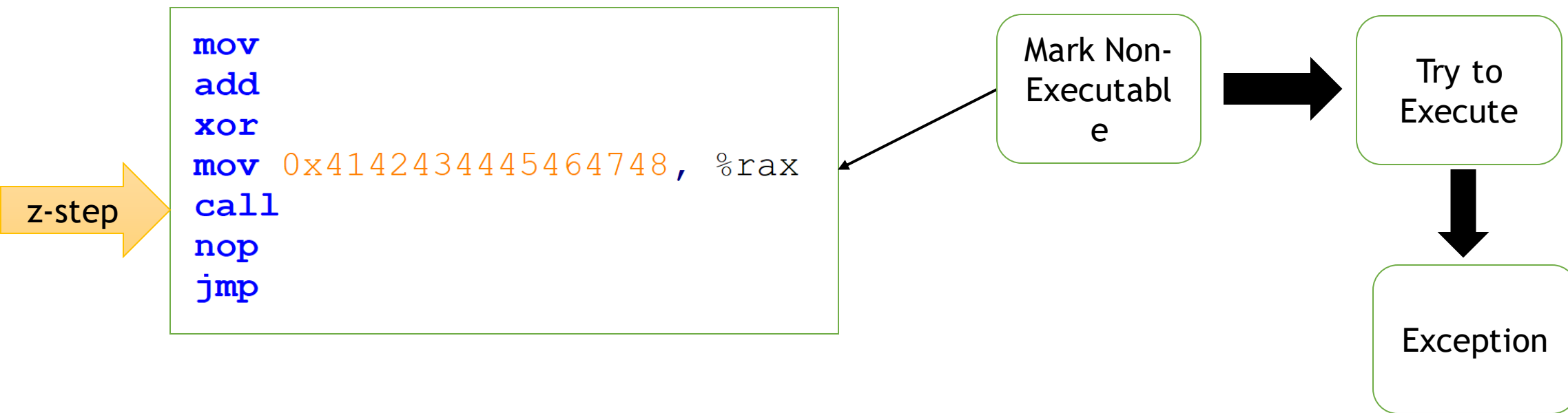
ZombieLoad - Recovering Intel SGX Sealing Key

- 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



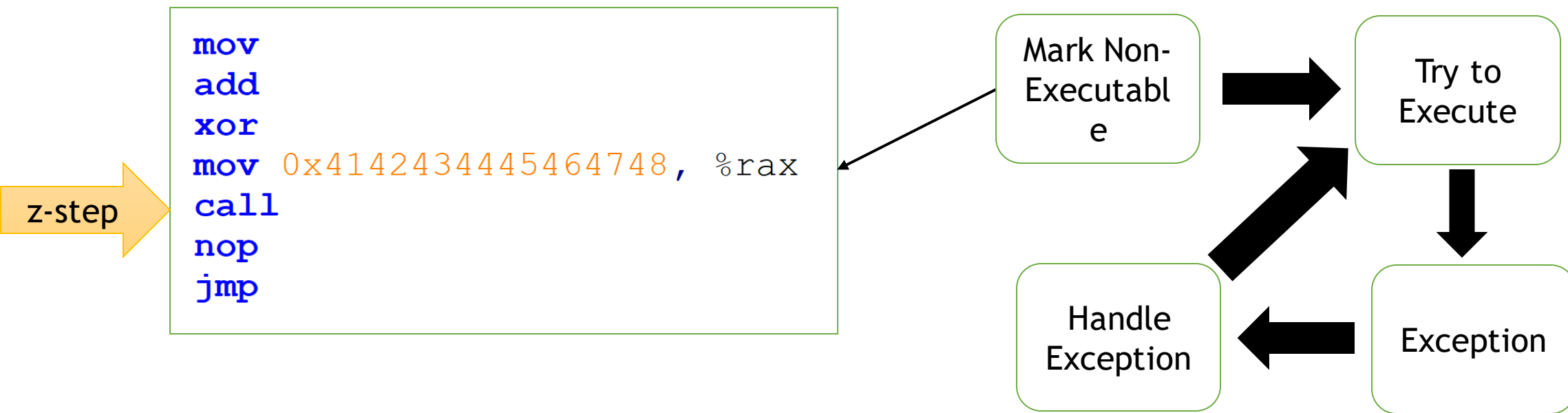
ZombieLoad - Recovering Intel SGX Sealing Key

- 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



ZombieLoad - 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



ZombieLoad - 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

Transynther and Medusa

- A Tool based on Fuzzing-techniques to Generate Data Leakage Code
 - Microarchitectural Grooming to Find new MDS Variants/Subvariants
 - Medusa, A New Variant that only Leaks Write Combining Stores
-
- Medusa
 - Write Combining fills up the entire Data Bus.
 - We leak only the Upper-half of the Data Bus to recover pre-filtered data.
 - Implicit WC, i.e., 'rep mov', 'rep stos', can be leaked.

Mitigation

- Spoiler and MemJam
 - **Hardware:** No plan to fix, No hardware mitigation!
 - **Software:** Constant-time implementation (Secret Obliviousness)
- MDS
 - **Hardware:**
 - Everything is vulnerable before IceLake CPU
 - Disable Hyperthreading to reduce the impact
 - **Software:** ~~Special Microcode Sequence~~

Questions?!



Daniel Moghimi

@danielmgmi

Publications

- MemJam: A False Dependency Attack against Constant-Time Crypto Implementations (IACR CT-RSA 2018, IJPP 2019)
- SPOILER: Speculative Load Hazards Boost Rowhammer and Cache Attacks (Usenix Security 2019).
- ZombieLoad: Cross-Privilege-Boundary Data Sampling. (ACM CCS 2019)
- Fallout: Leaking Data on Meltdown-resistant CPU. (ACM CCS 2019)
- Medusa (will appear at Usenix Security 2020)

