Microarchitectural Data Leakage

via Automated Attack Synthesis

Daniel Moghimi, Worcester Polytechnic University

Jun 23, 2020

Virtual Talk for Intel Product Security Incident Team (IPSIRT)

About Me

- Daniel Moghimi (@danielmgmi)
- Security Researcher
- PhD Student @ WPI
  - Microarchitectural Security
  - Side Channels
  - Breaking Crypto Implementations
  - Trusted Execution Environment (Intel SGX)

- Contributed to:
  - ZombieLoad, Fallout, LVI,
  - MemJam, Spoiler, CacheZoom, CopyCat
  - Jackhammer, TPM-Fail

Thanks…

- Berk Sunar @ WPI
- Moritz Lipp @ tugraz
- Michael Schwartz @ tugraz

Disclaimers

- Our findings and reasonings are based on:
  - RE
  - Patents
  - Analysis

- You may know more than me how Intel CPU works!!!

Today’s Agenda

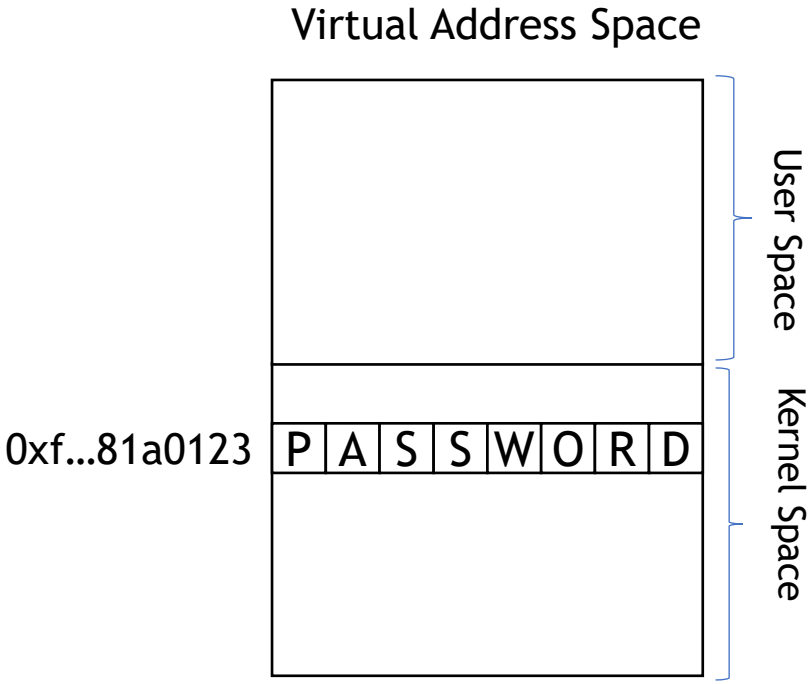
- Motivation: Meltdown-style Attacks
- Background: CPU Memory Subsystem
- Transynther, Automated Attack Synthesis
- MDS Root Cause Analysis and new subvariants
- Medusa attack and RSA key recovery

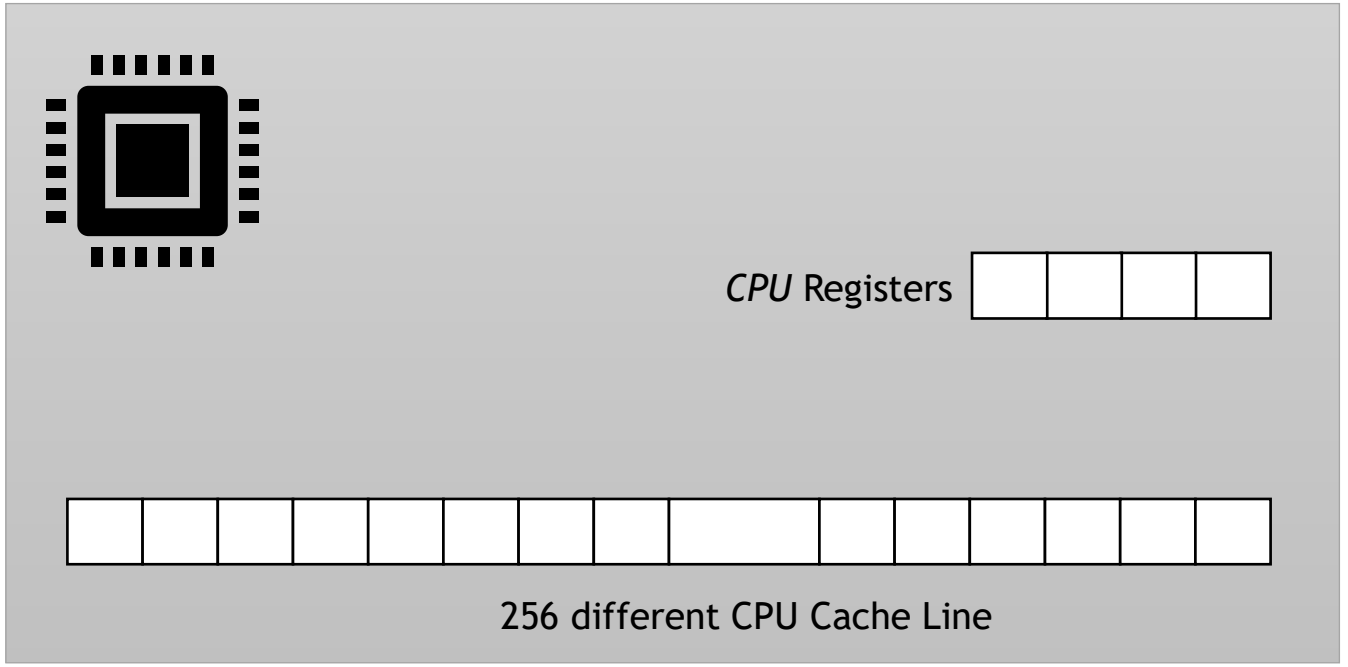
2018: Meltdown Attack?

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

2018: Meltdown Attack?

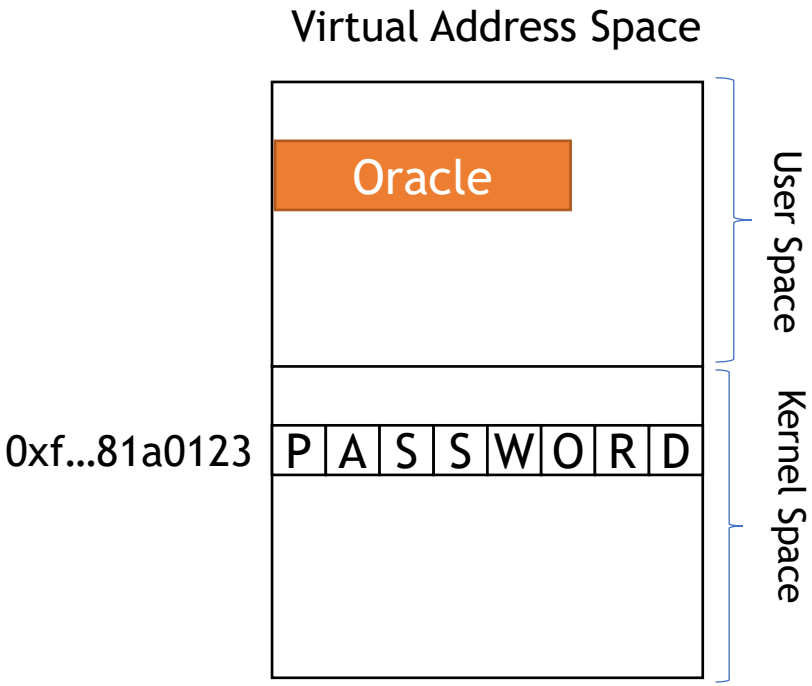
```
char secret = *(char *) 0xffffffff81a0123;
```

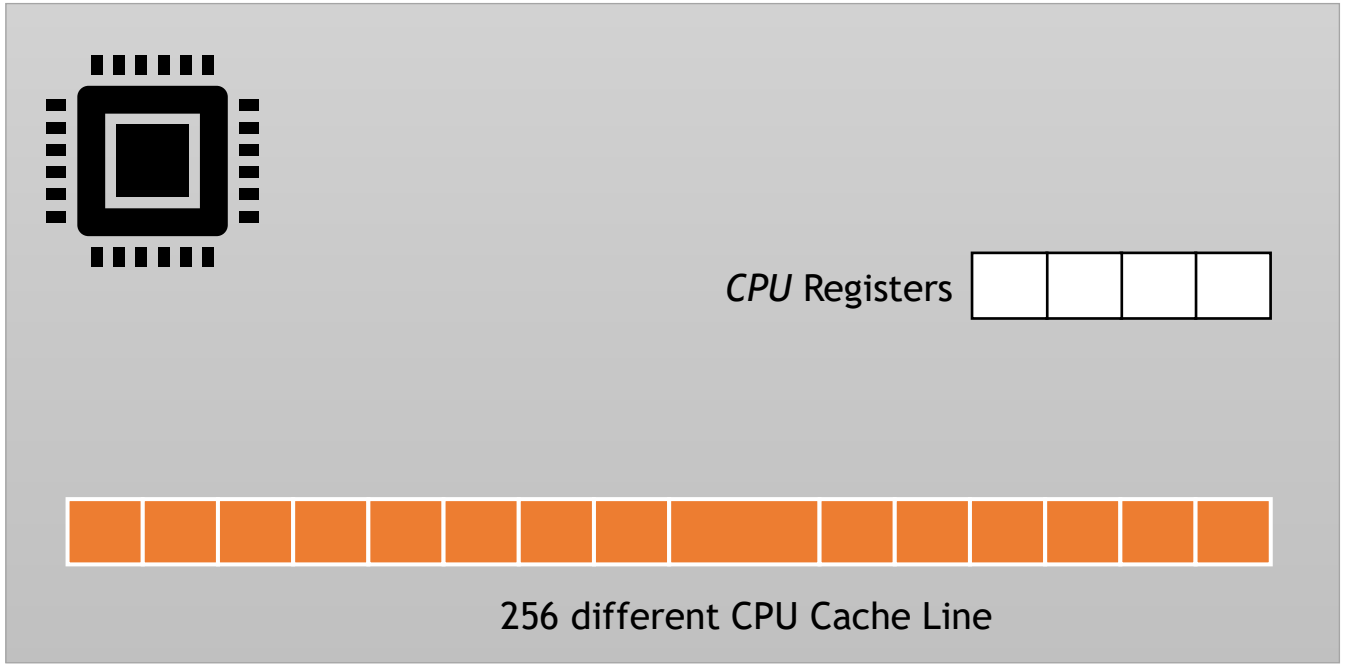




2018: Meltdown Attack?

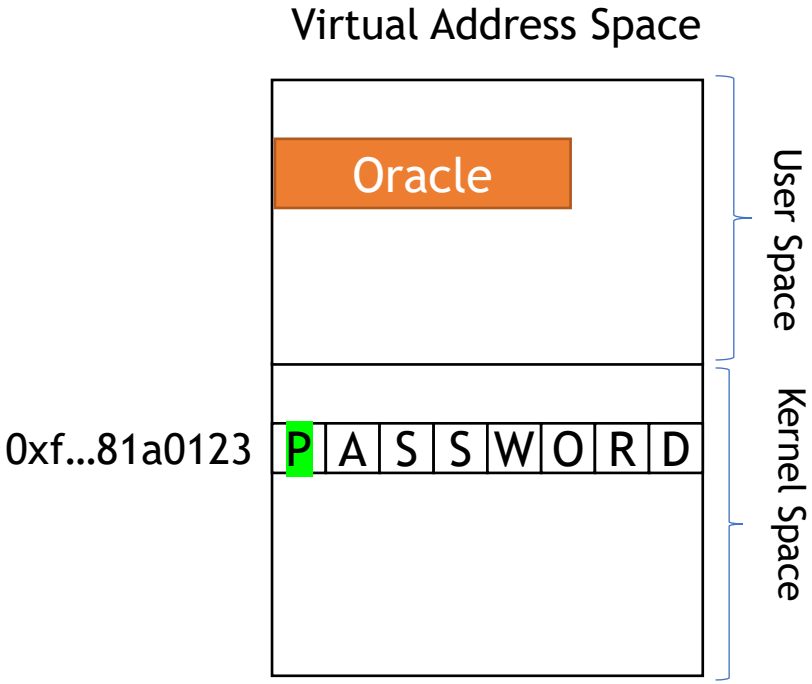
```
char secret = *(char *) 0xffffffff81a0123;
```

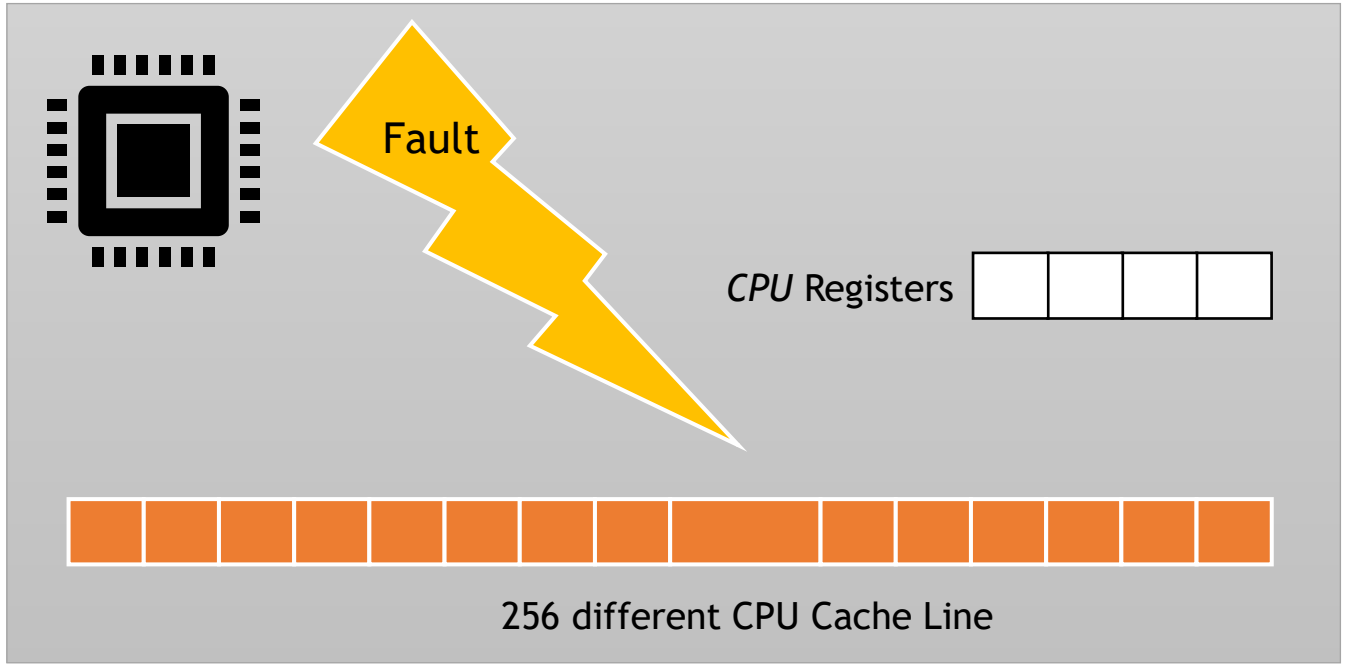




2018: Meltdown Attack? (Step 1)

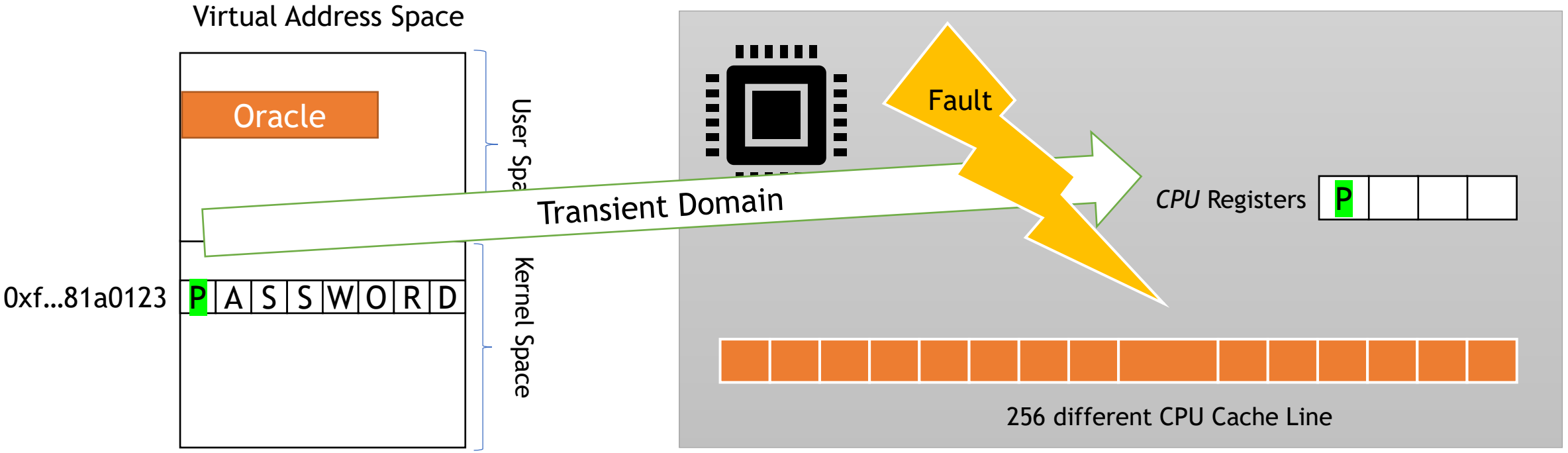
```
char secret = *(char *) 0xffffffff81a0123;
```





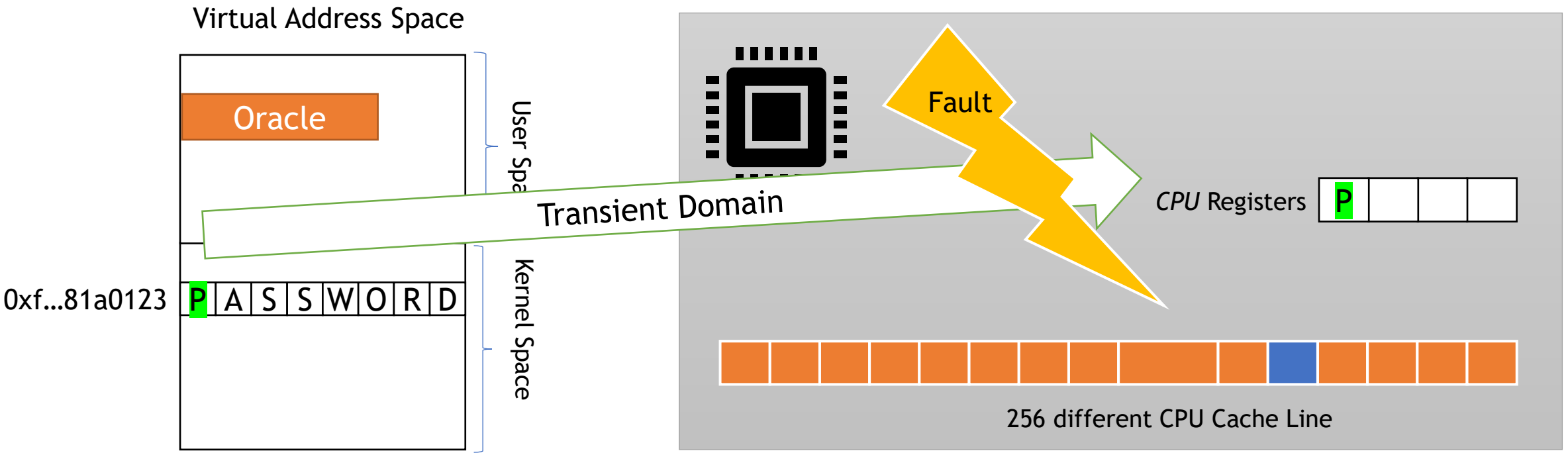
2018: Meltdown Attack? (Step 1)

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



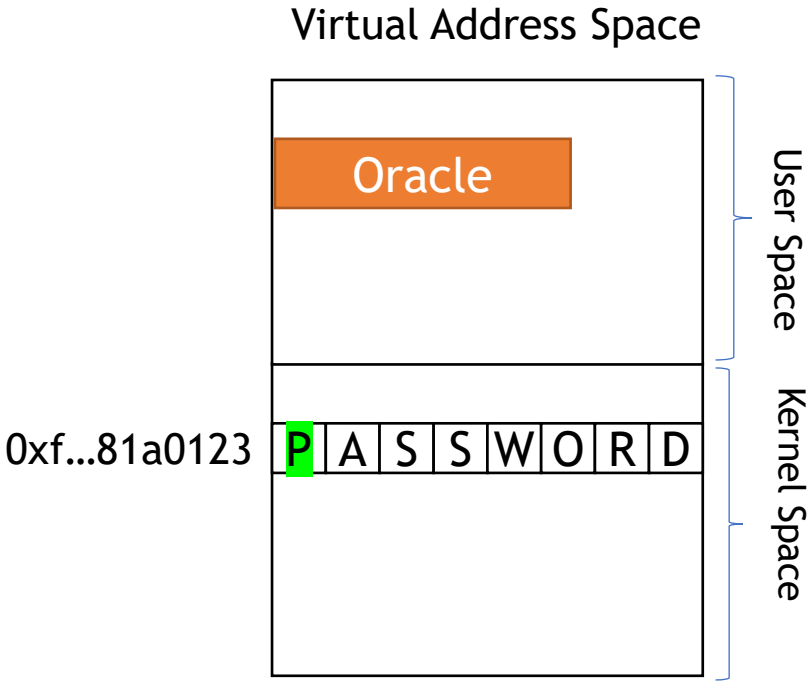
2018: Meltdown Attack? (Step 2)

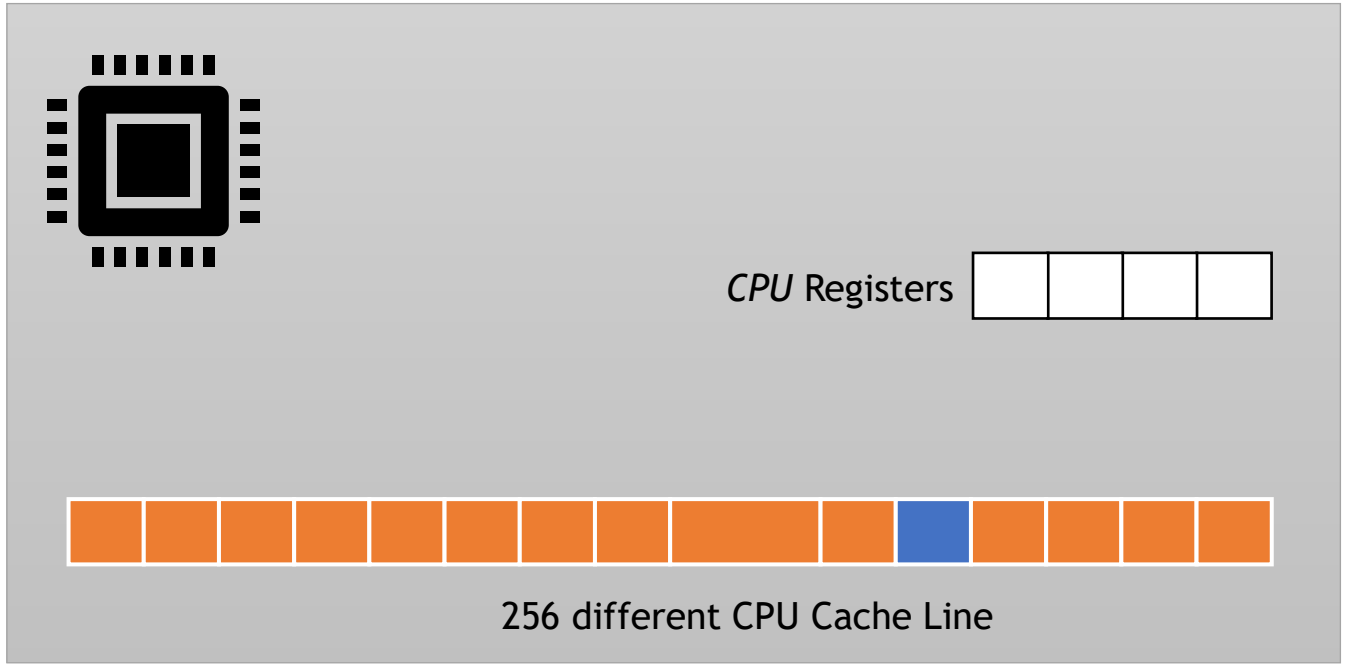
```
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```



2018: Meltdown Attack? (Step 2)

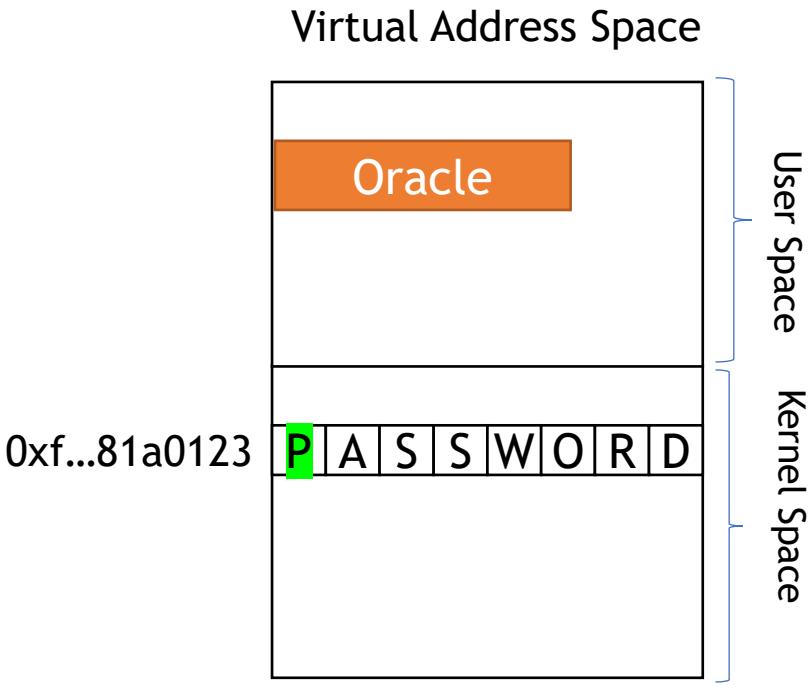
```
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```

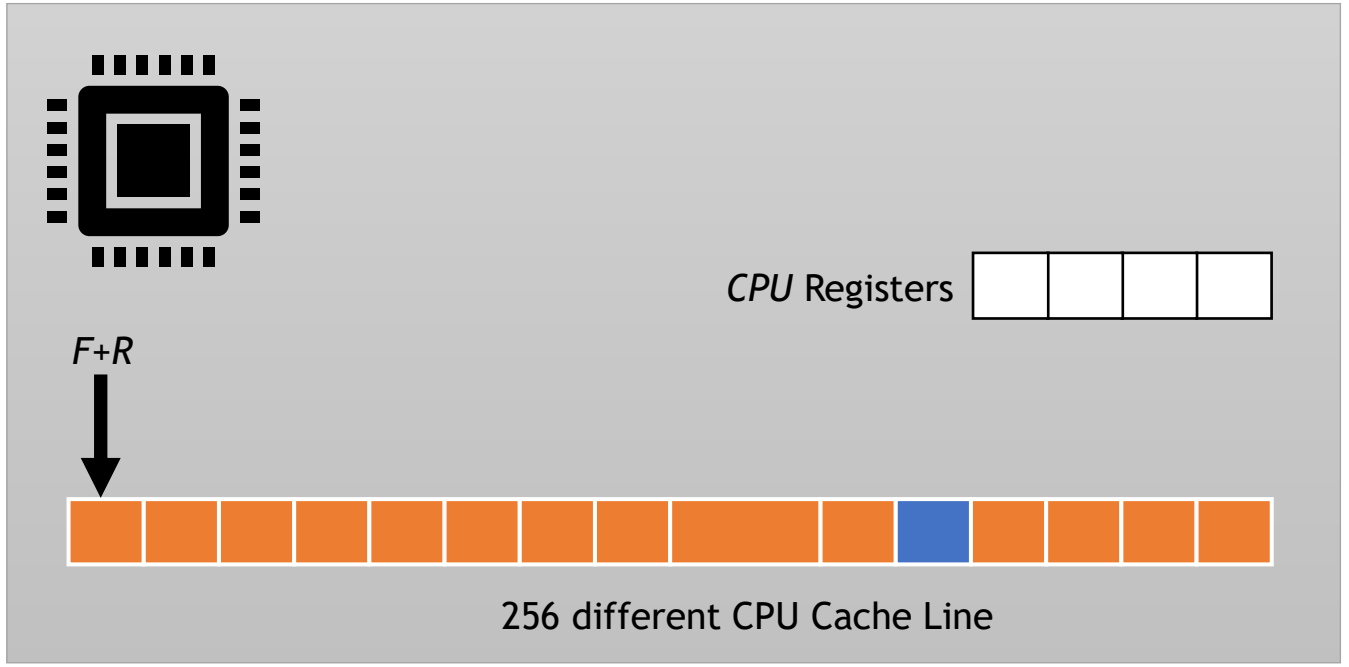




2018: Meltdown Attack? (Step 3)

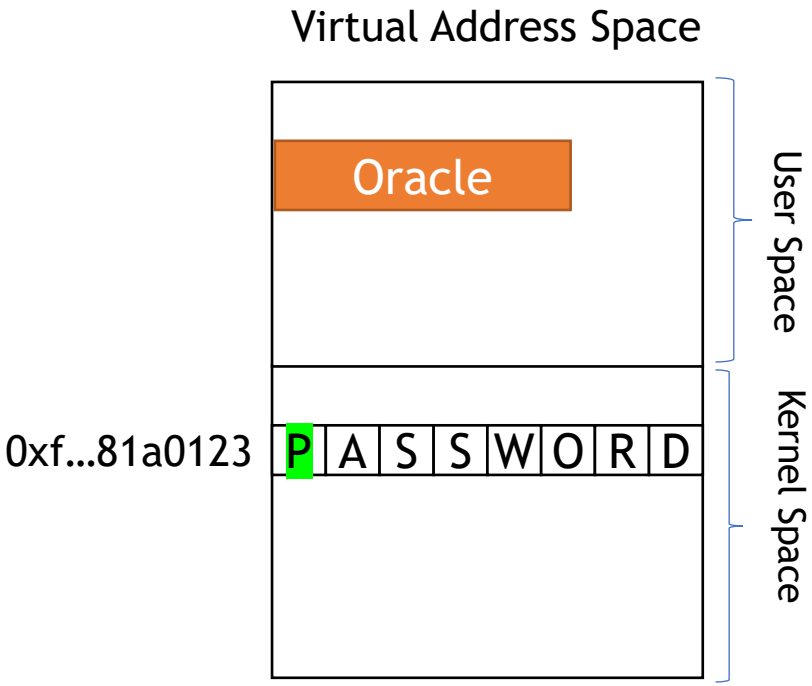
```
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```

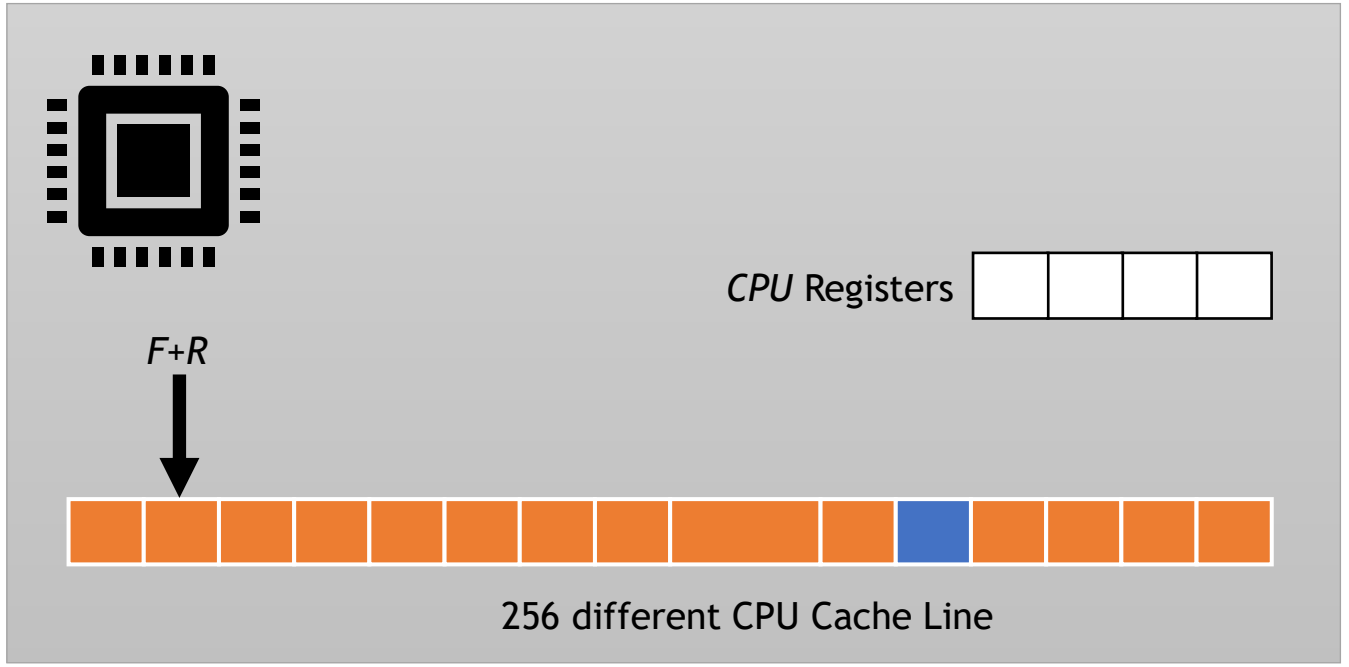




2018: Meltdown Attack? (Step 3)

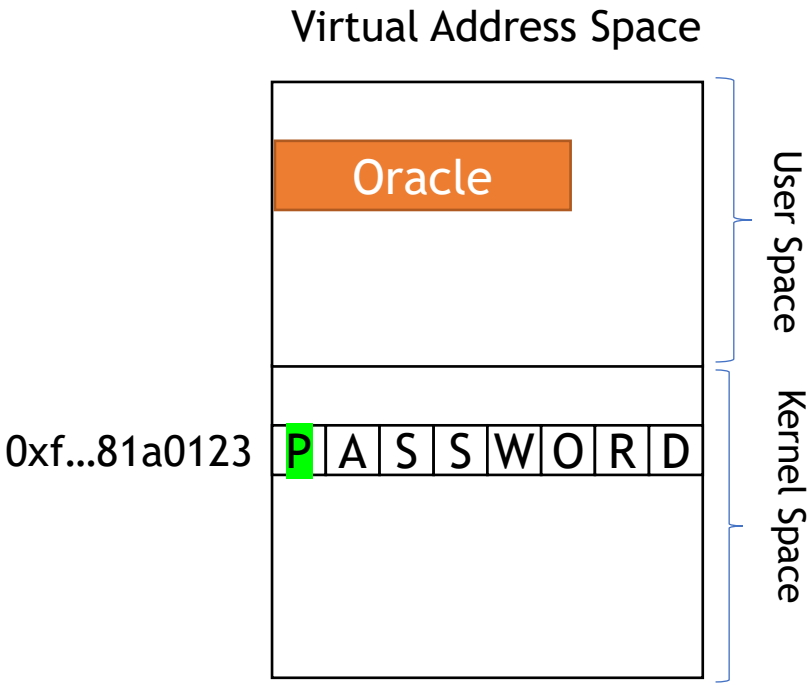
```
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```

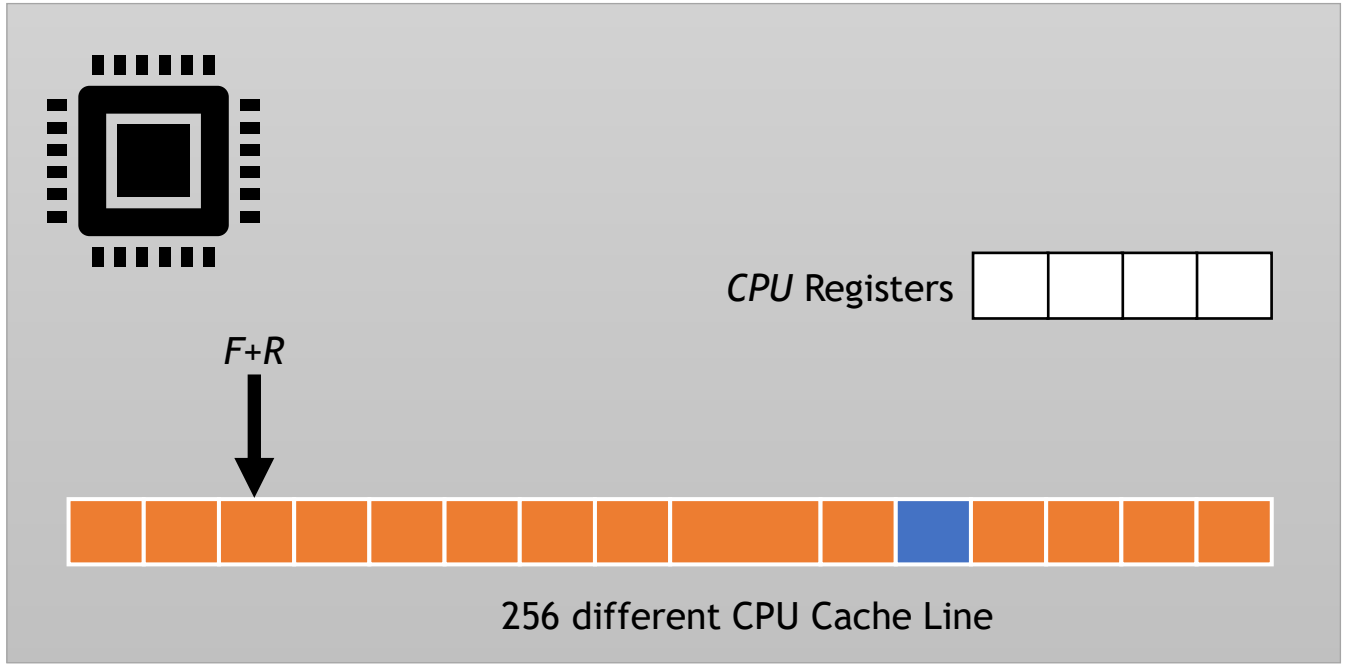




2018: Meltdown Attack? (Step 3)

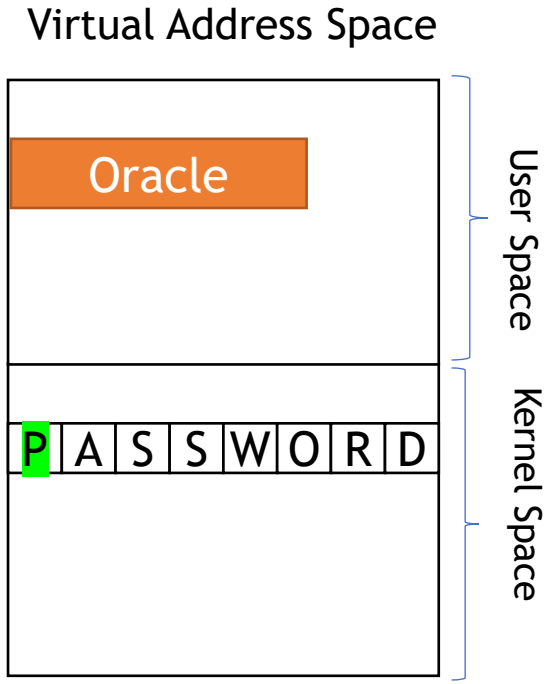
```
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```

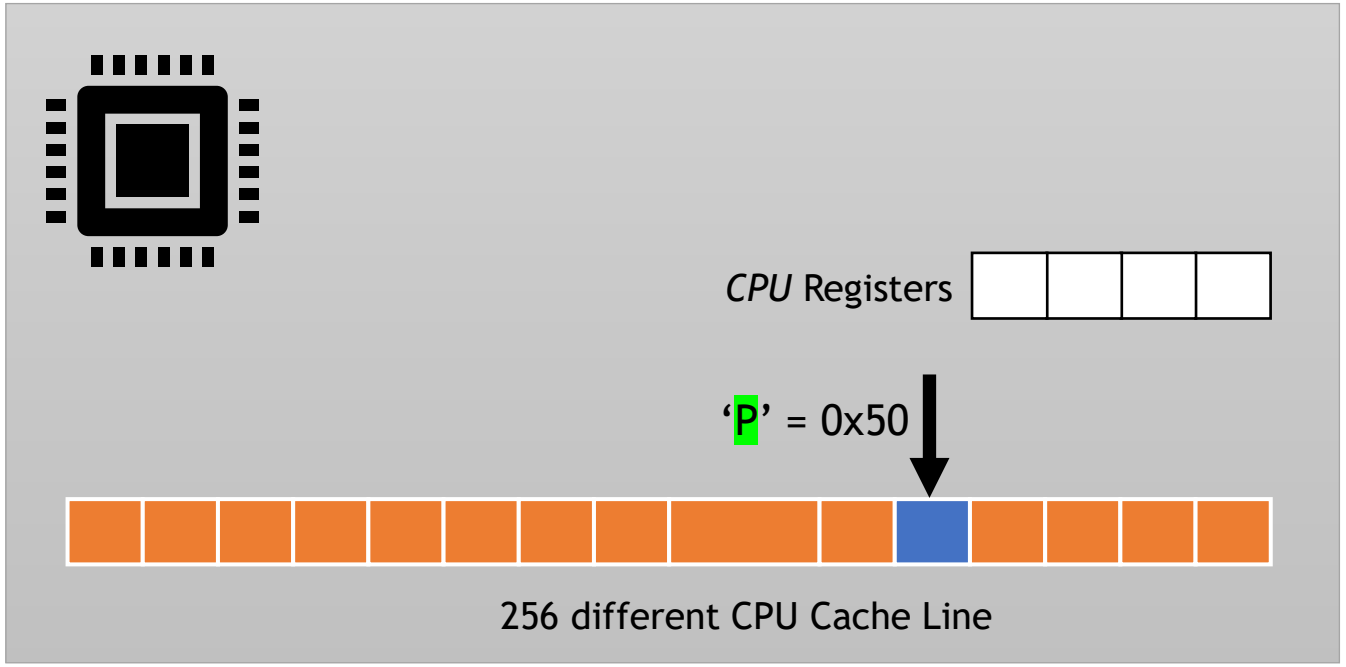




2018: Meltdown Attack? (Step 3)

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





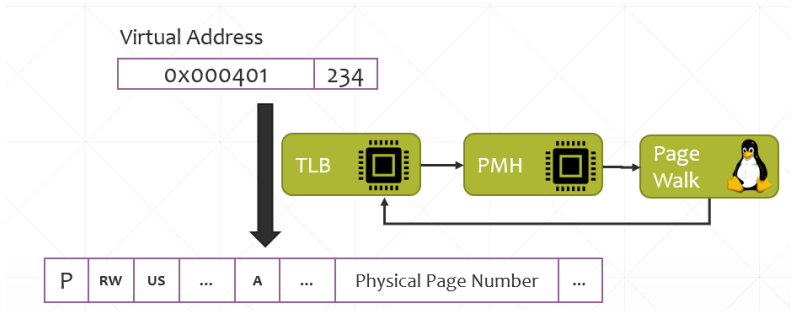
Microarchitecture Data Sampling (MDS)

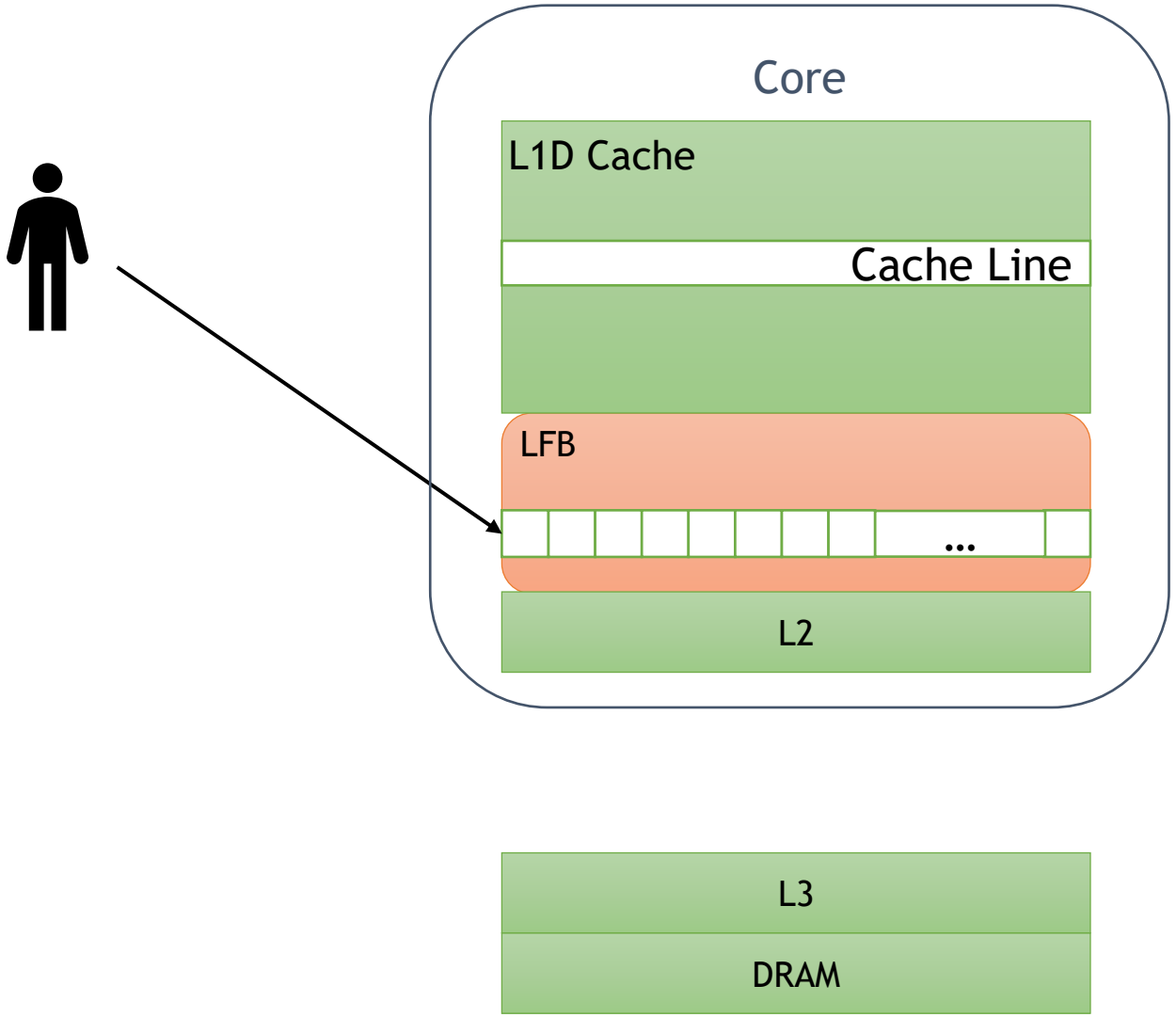
- Meltdown is fixed but you can still leak on the fix hardware.
- Which part of the CPU leak the data?!

- **Why does it leak?**

ZombieLoad Attack

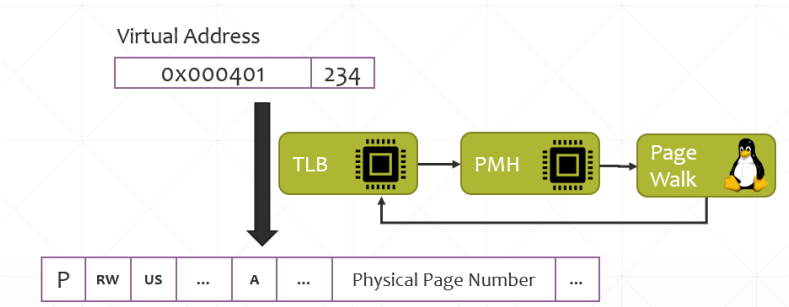
```
mov 0x401234, %rsi

mov (%rsi), %rax
```

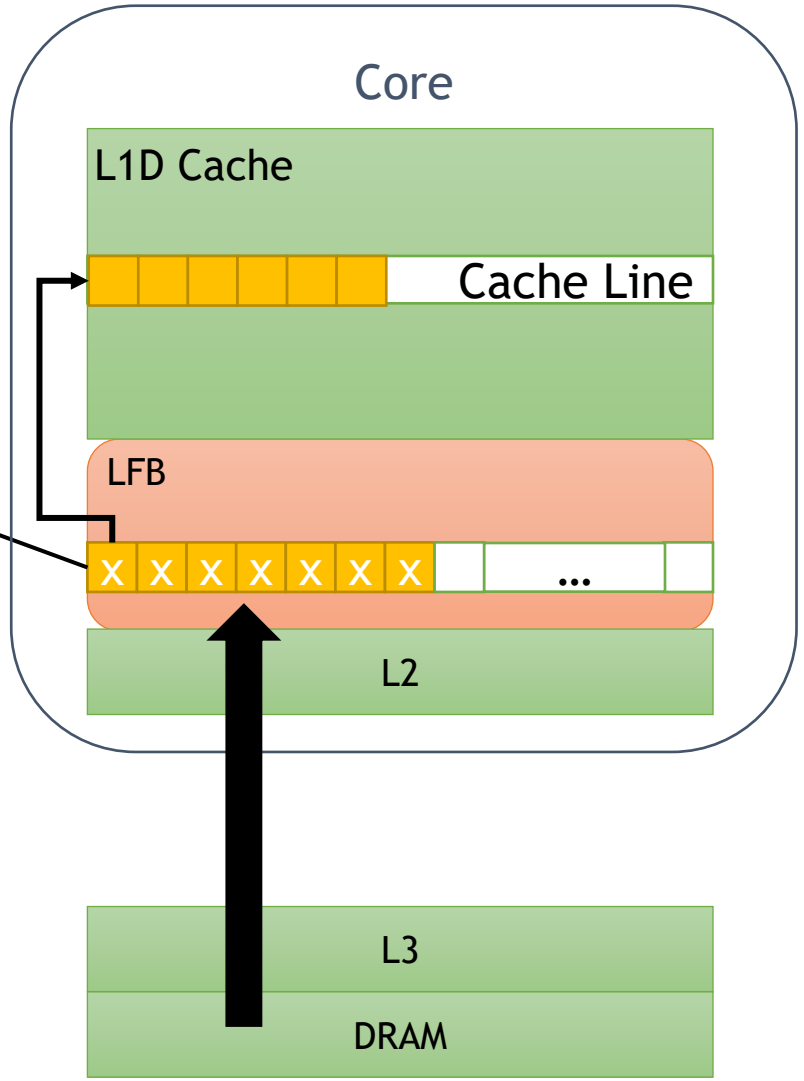




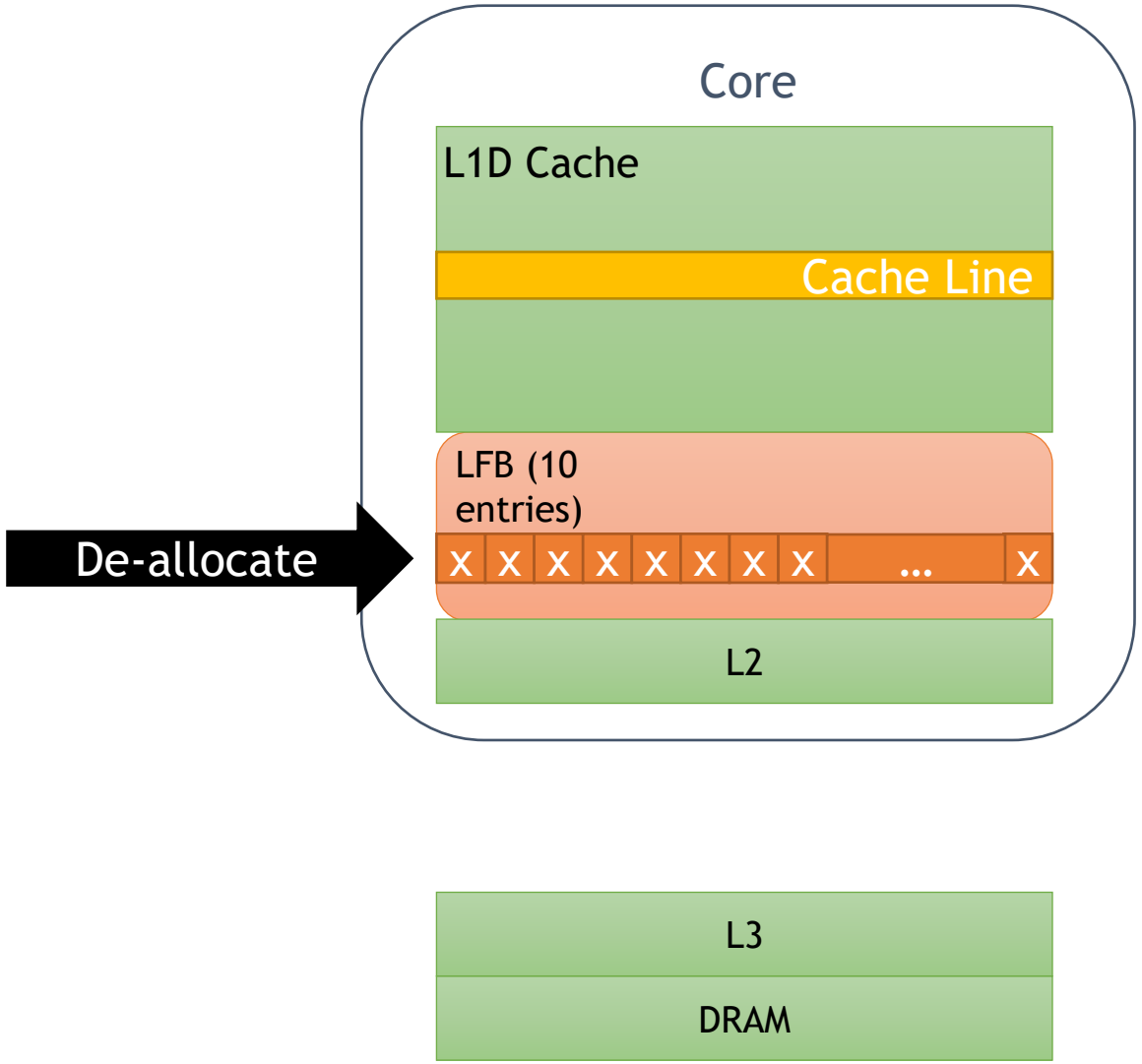
ZombieLoad Attack

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

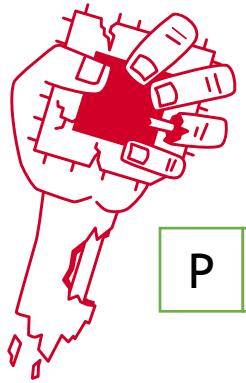




ZombieLoad Attack



ZombieLoad Attack



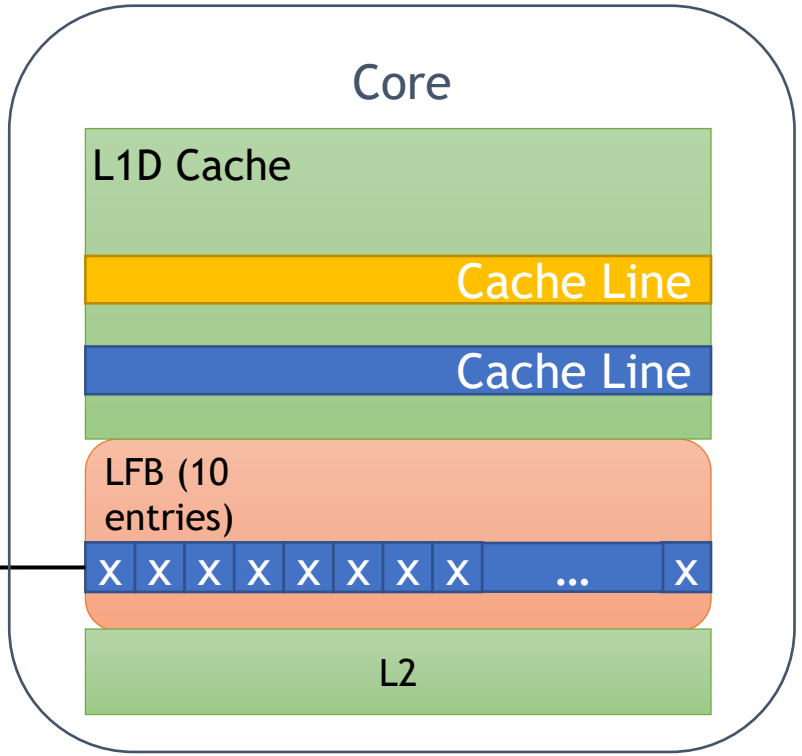
Variant 2: #RTM (AKA TAA)

P	RW	US	…	A	…	Physical Page Number	…

Variant 1: #GP

Variant 3: MC

x x x x

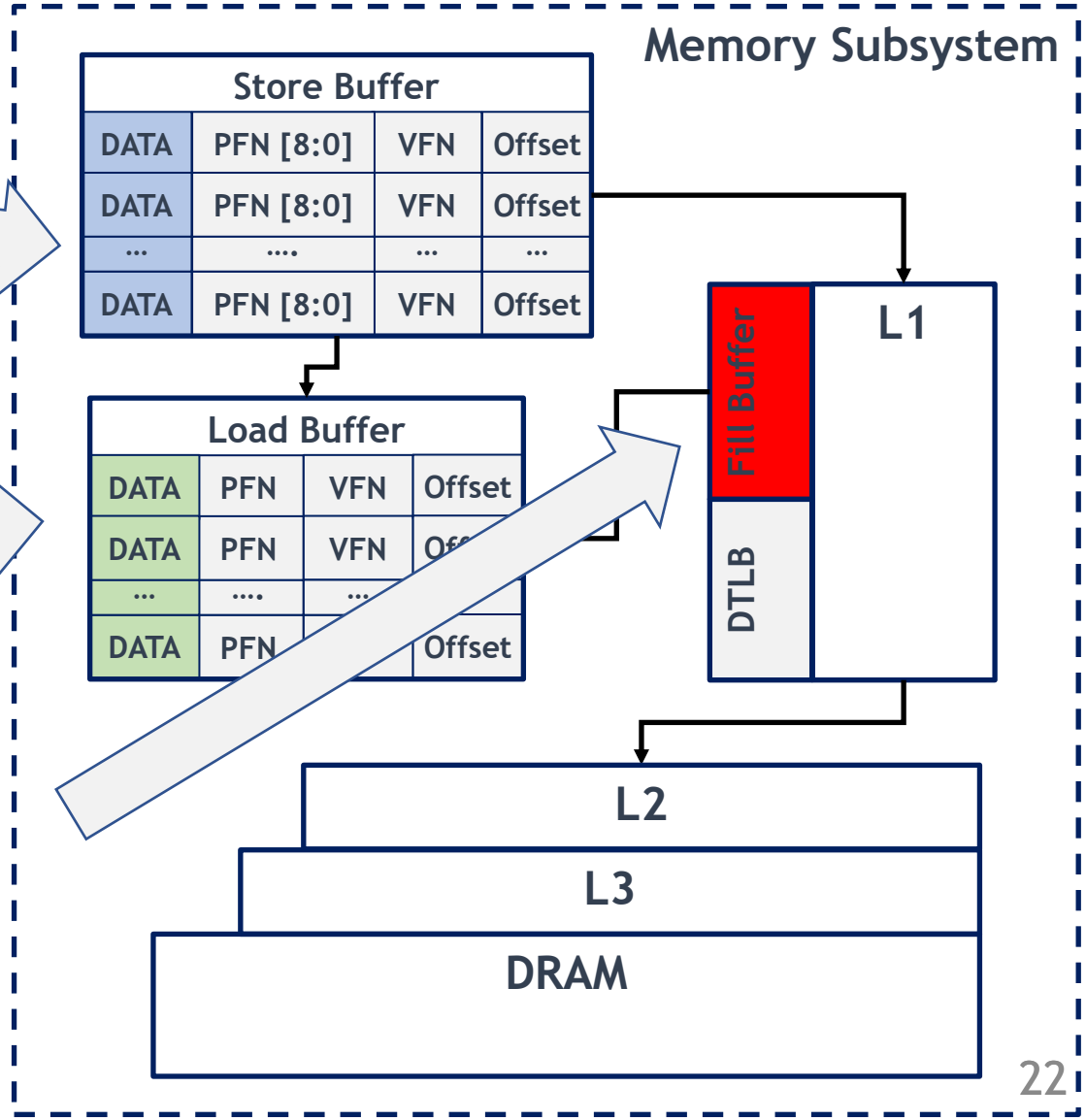


CPU Memory Subsystem – Leaky Buffers

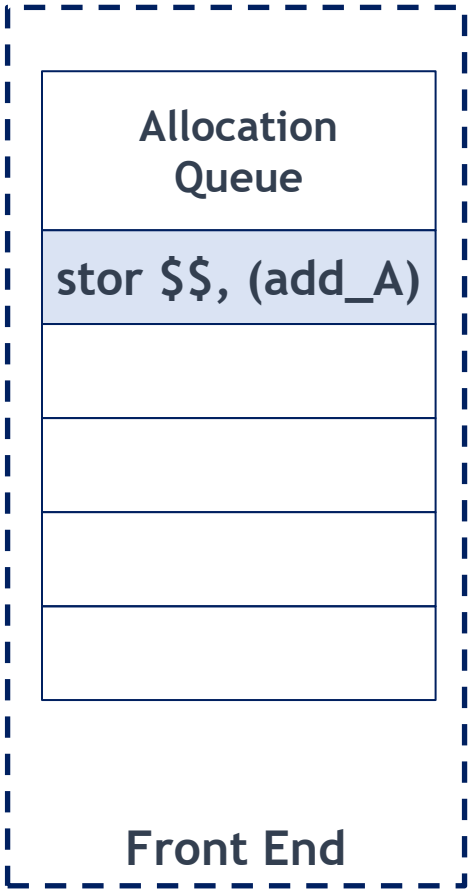
MSBDS

MLPDS

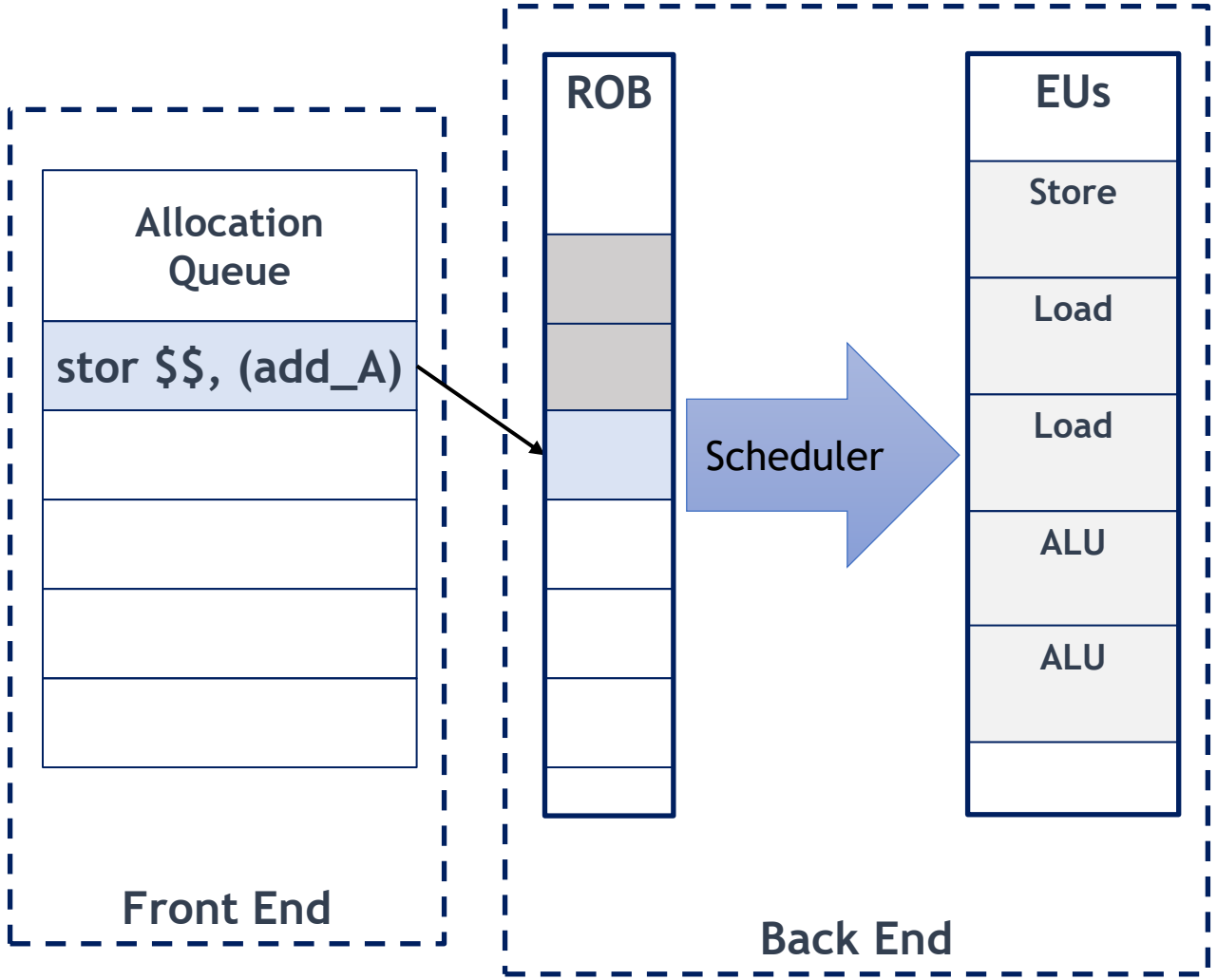
MFBDS



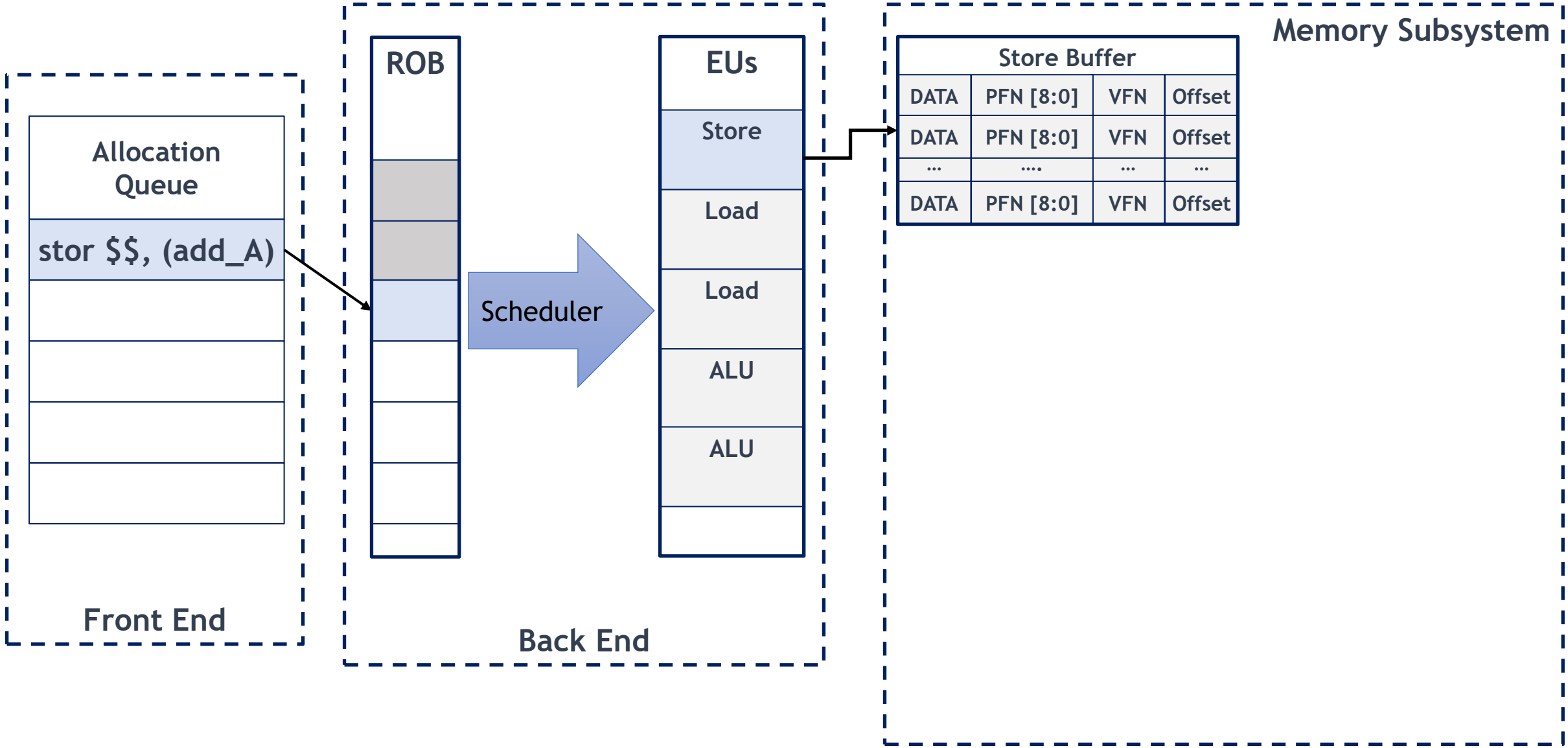
CPU Memory Subsystem



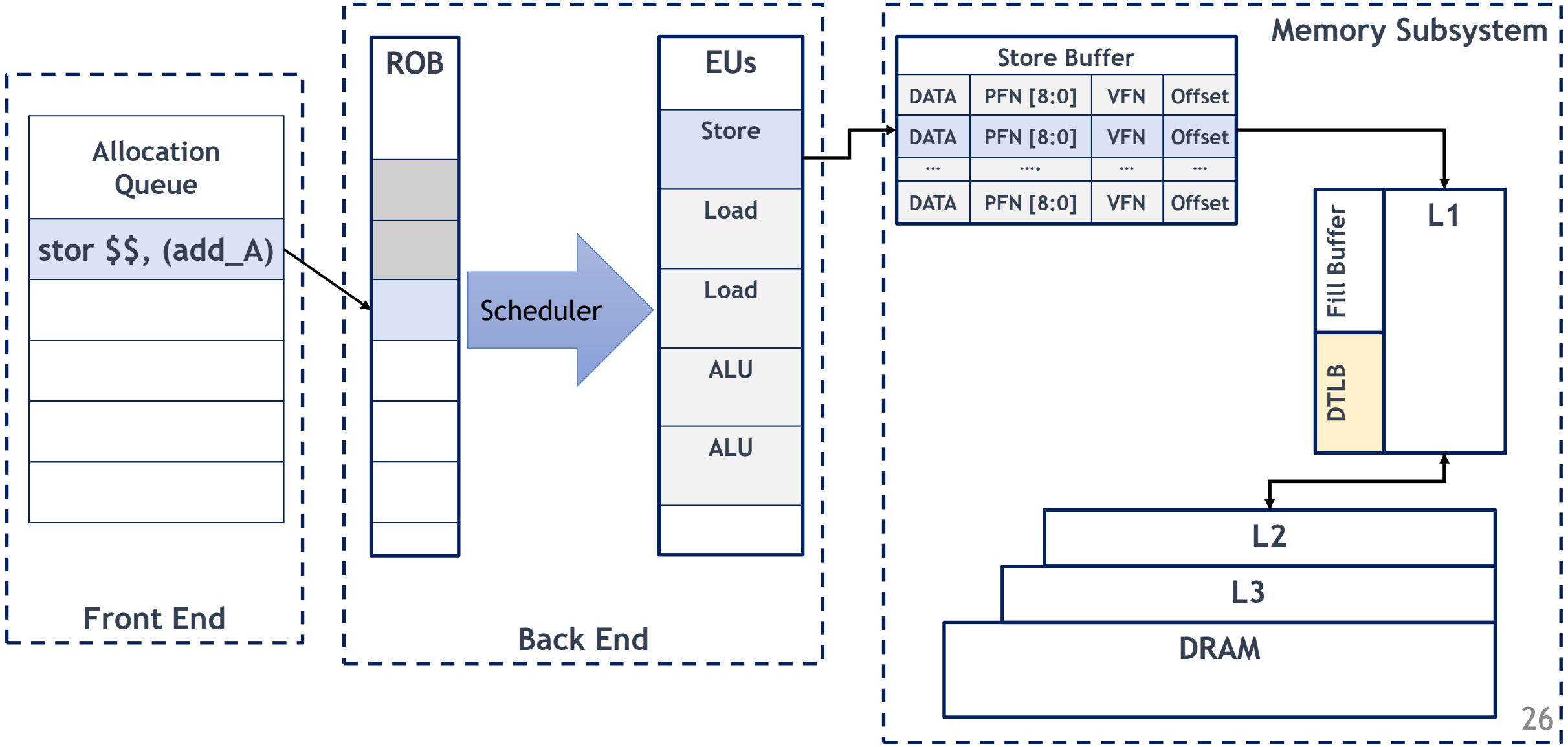
CPU Memory Subsystem



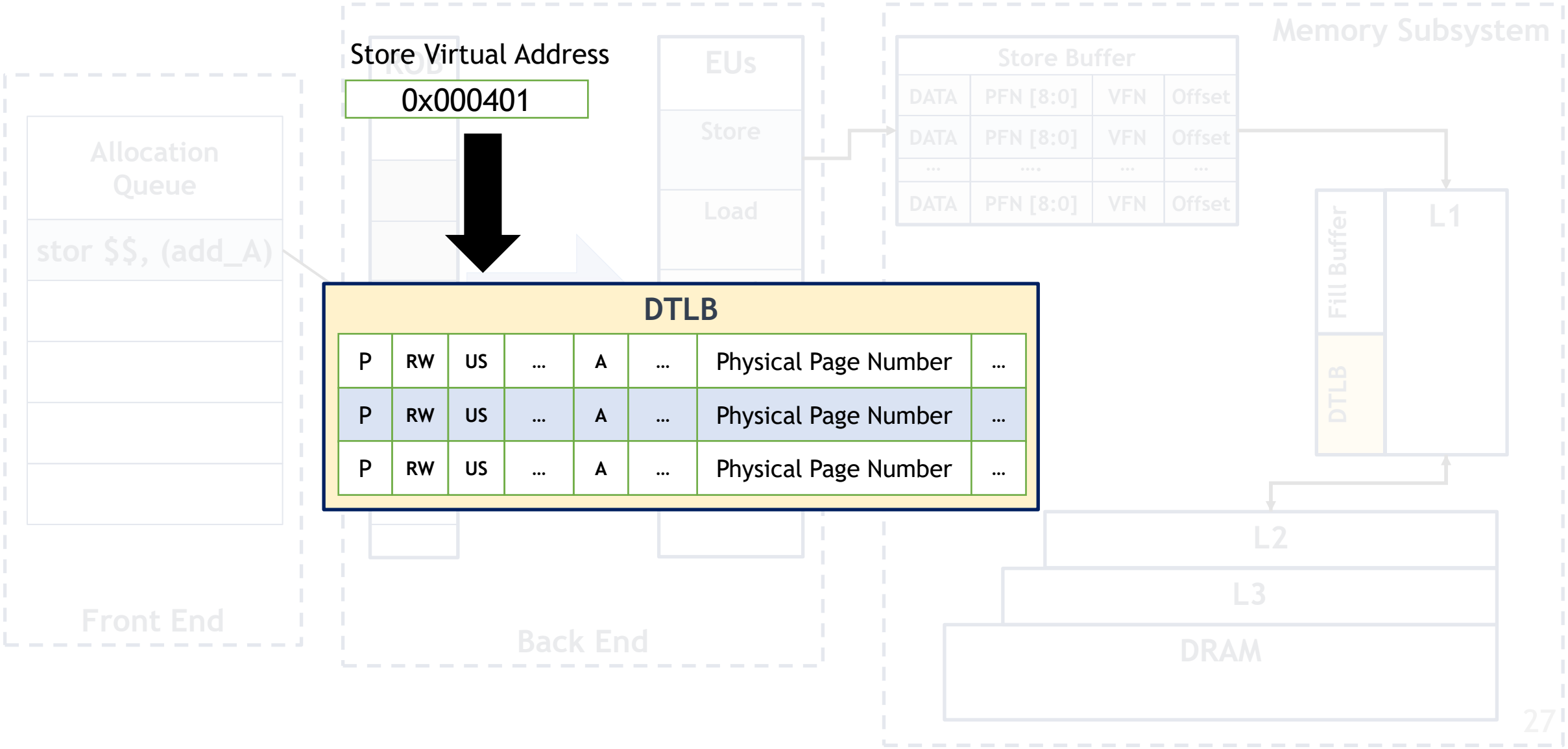
CPU Memory Subsystem



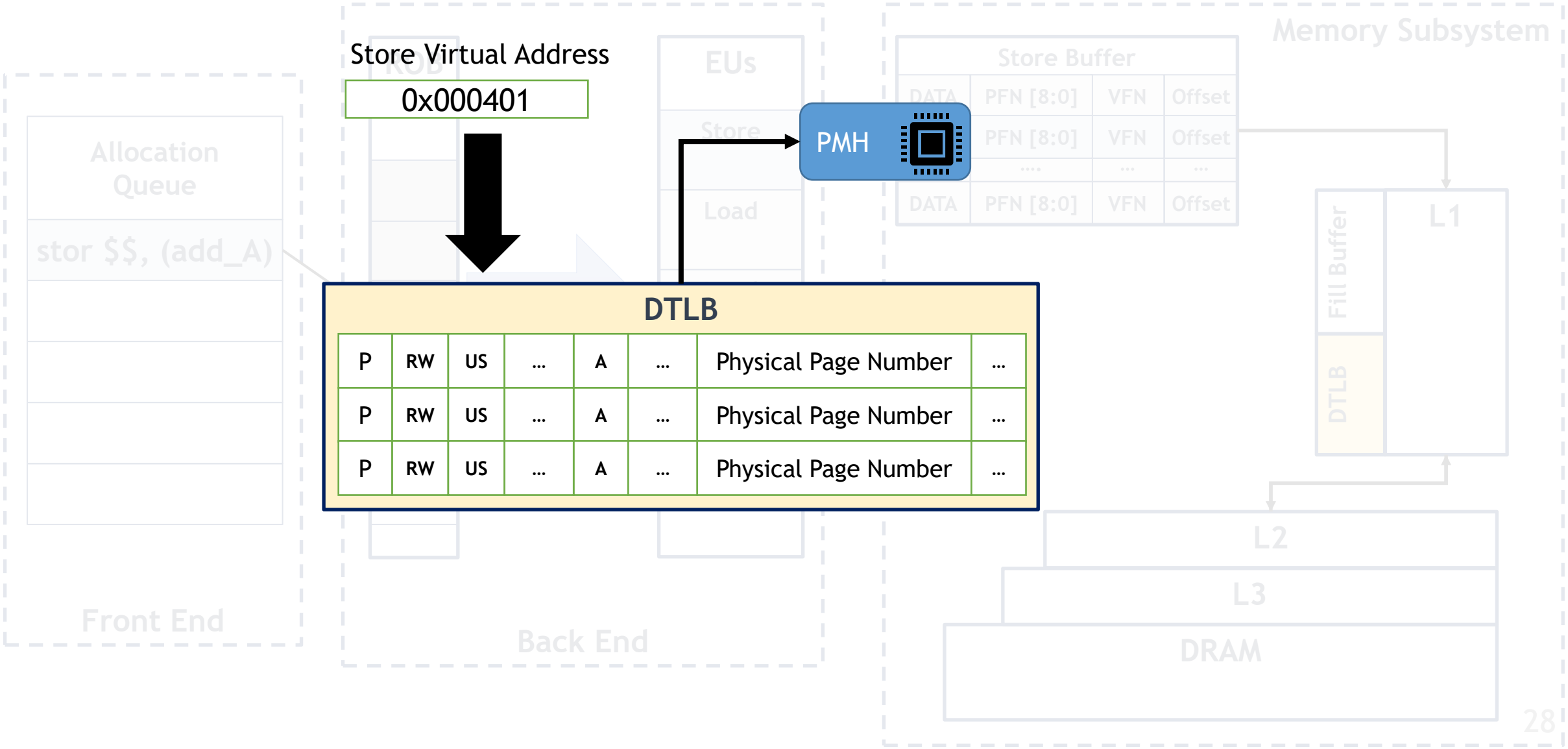
CPU Memory Subsystem



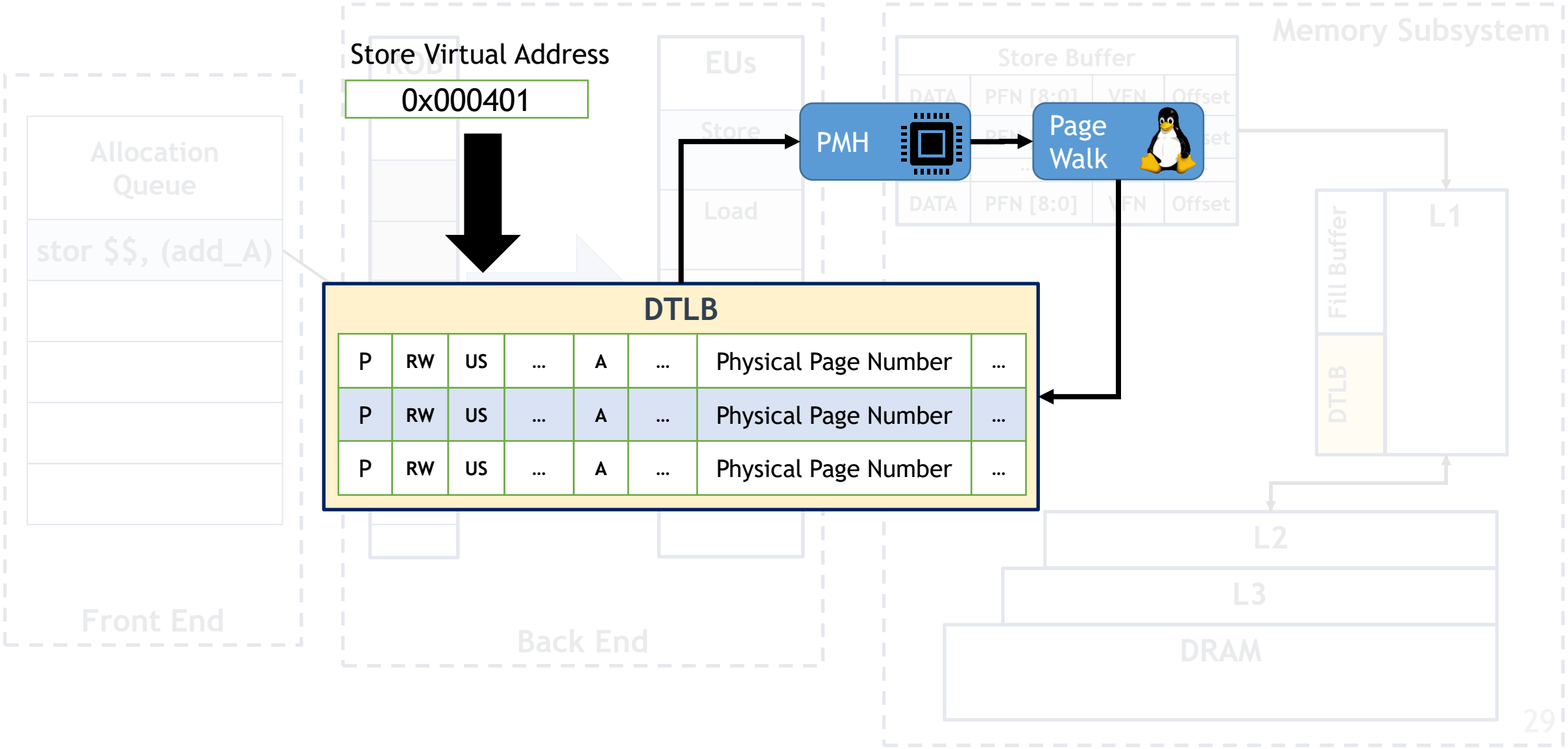
CPU Memory Subsystem



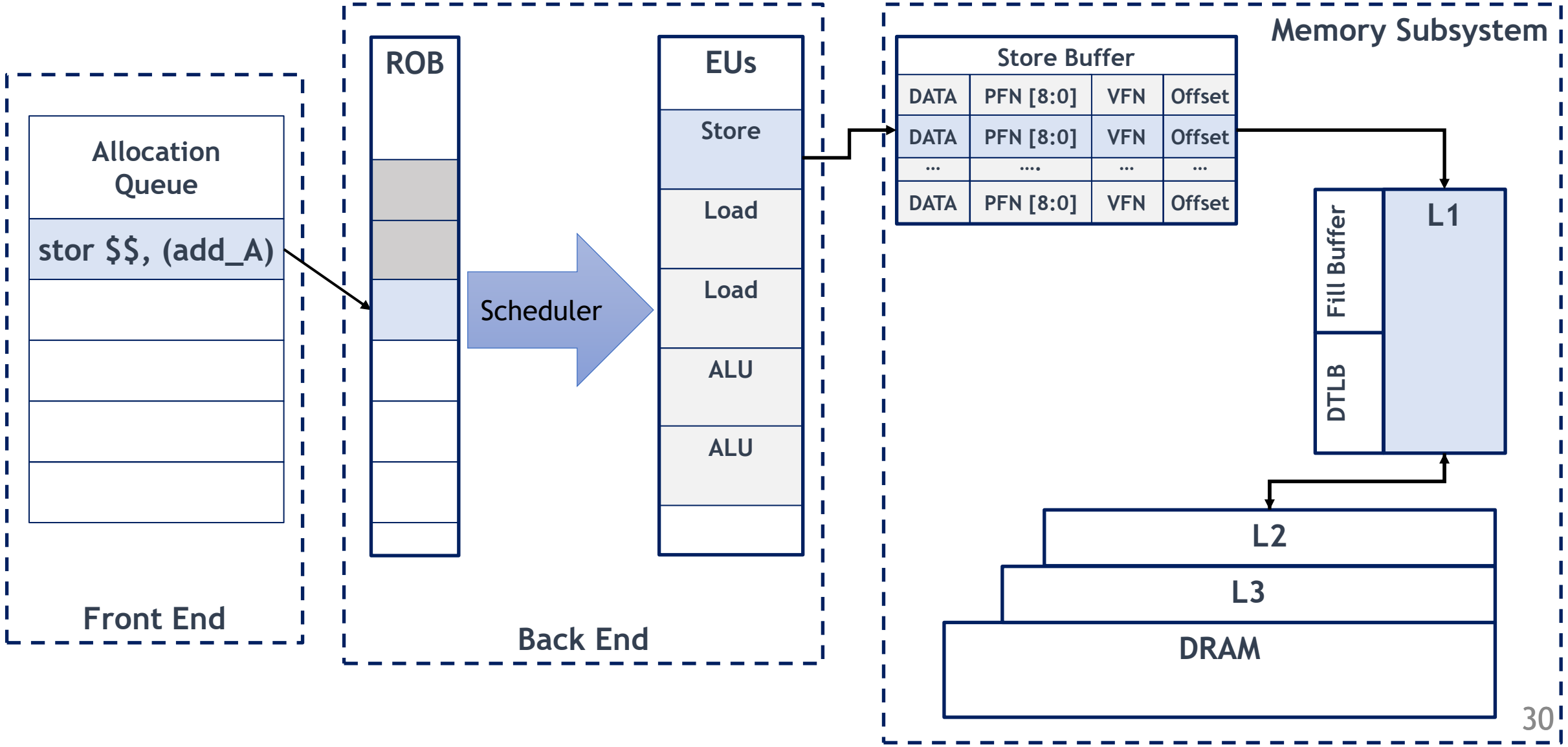
CPU Memory Subsystem



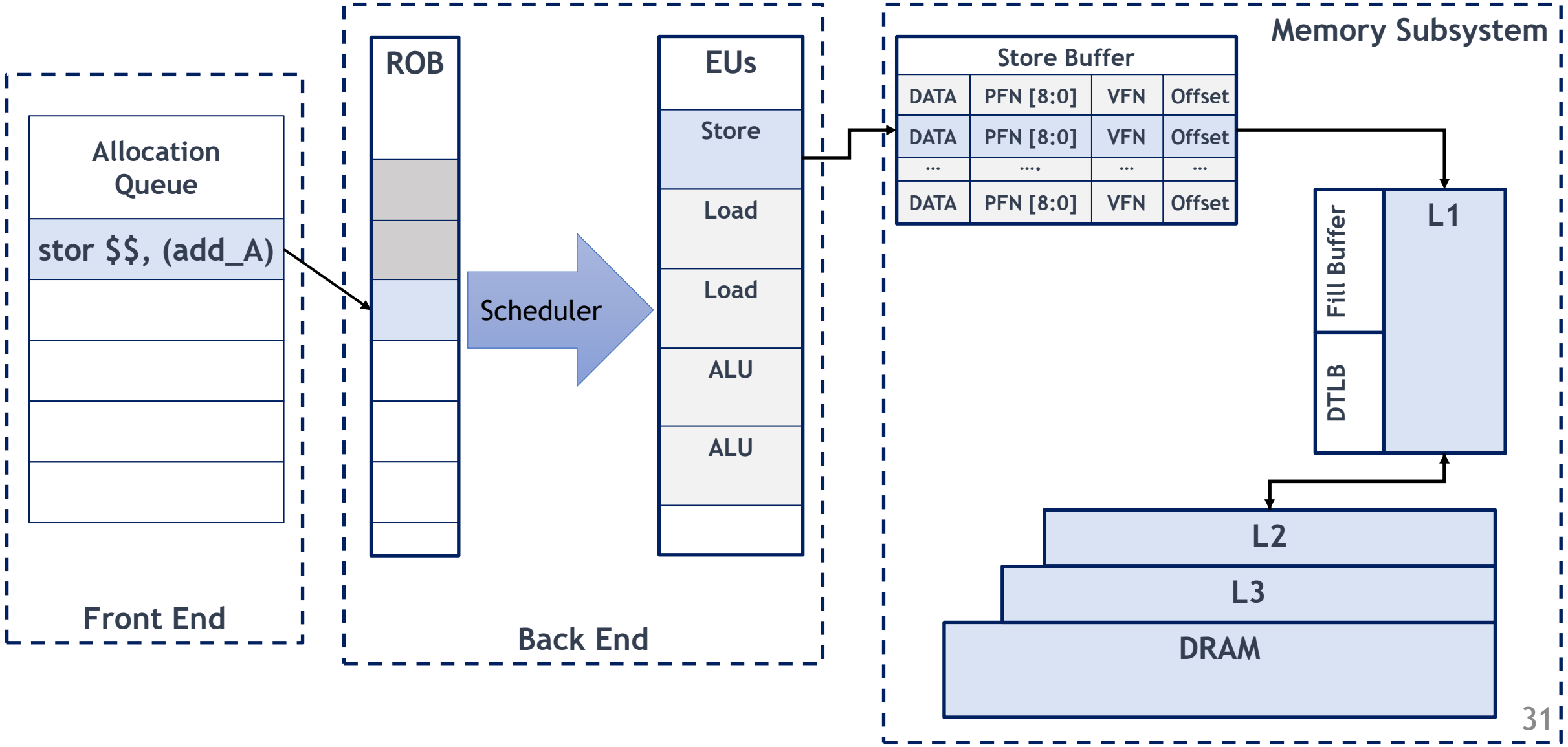
CPU Memory Subsystem



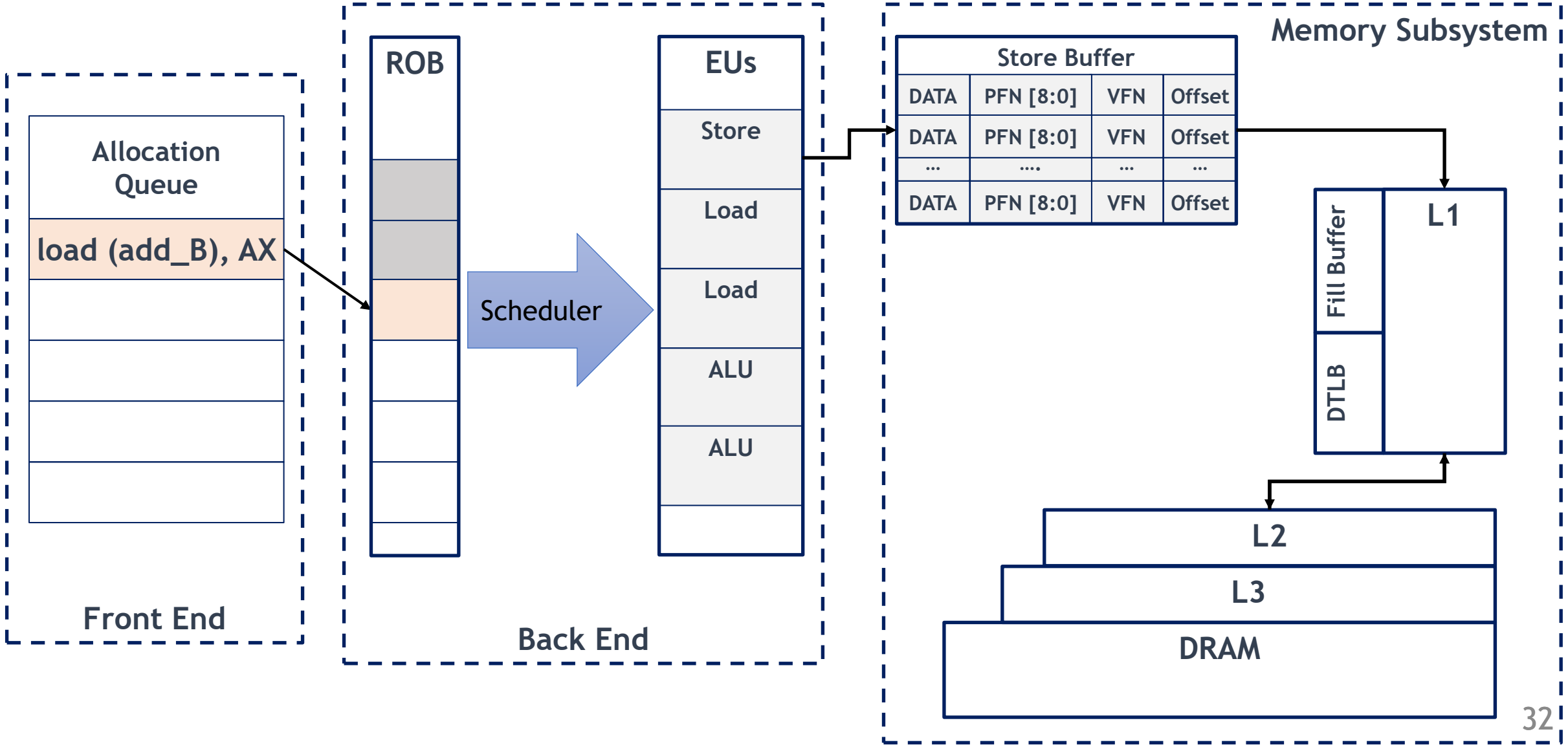
CPU Memory Subsystem



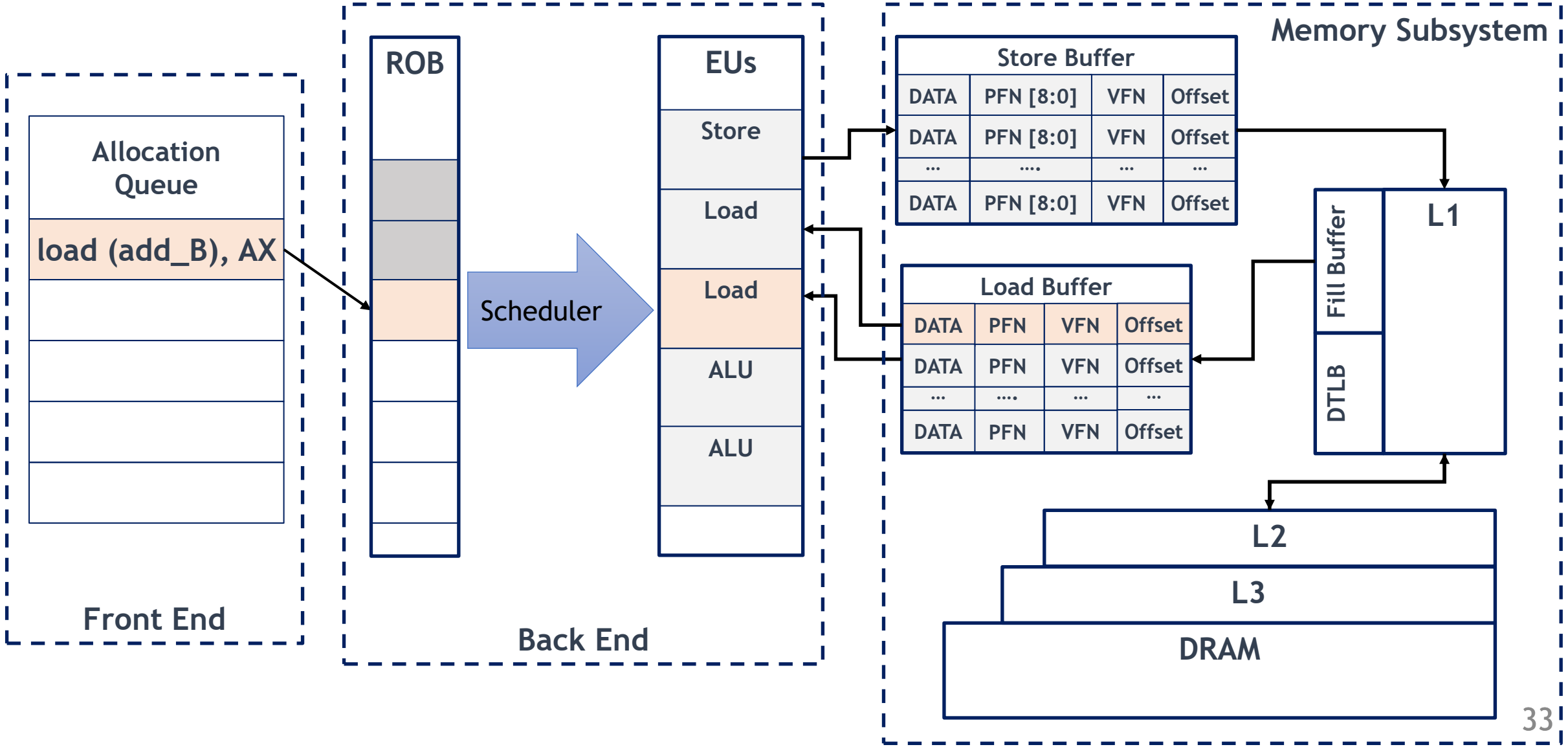
CPU Memory Subsystem



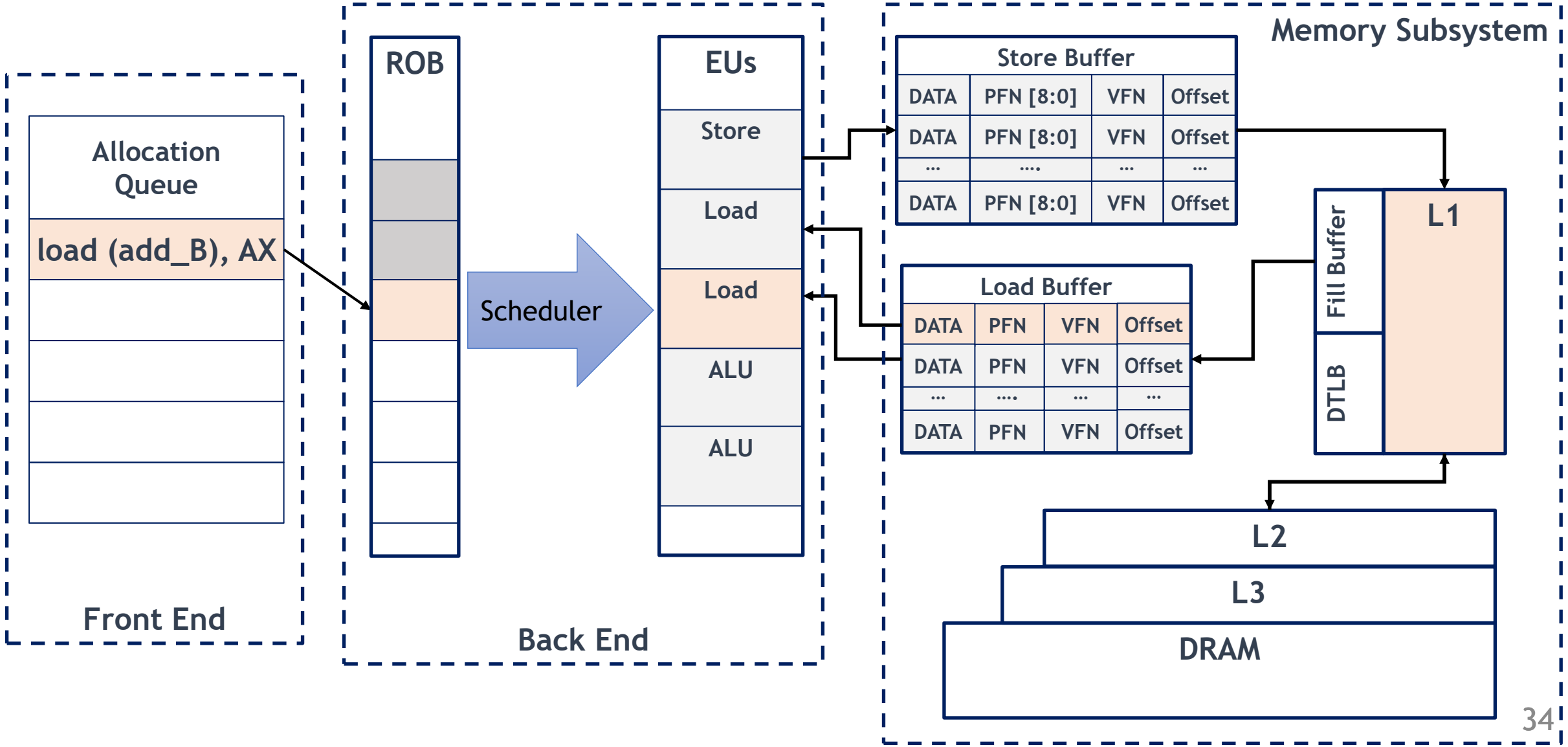
CPU Memory Subsystem



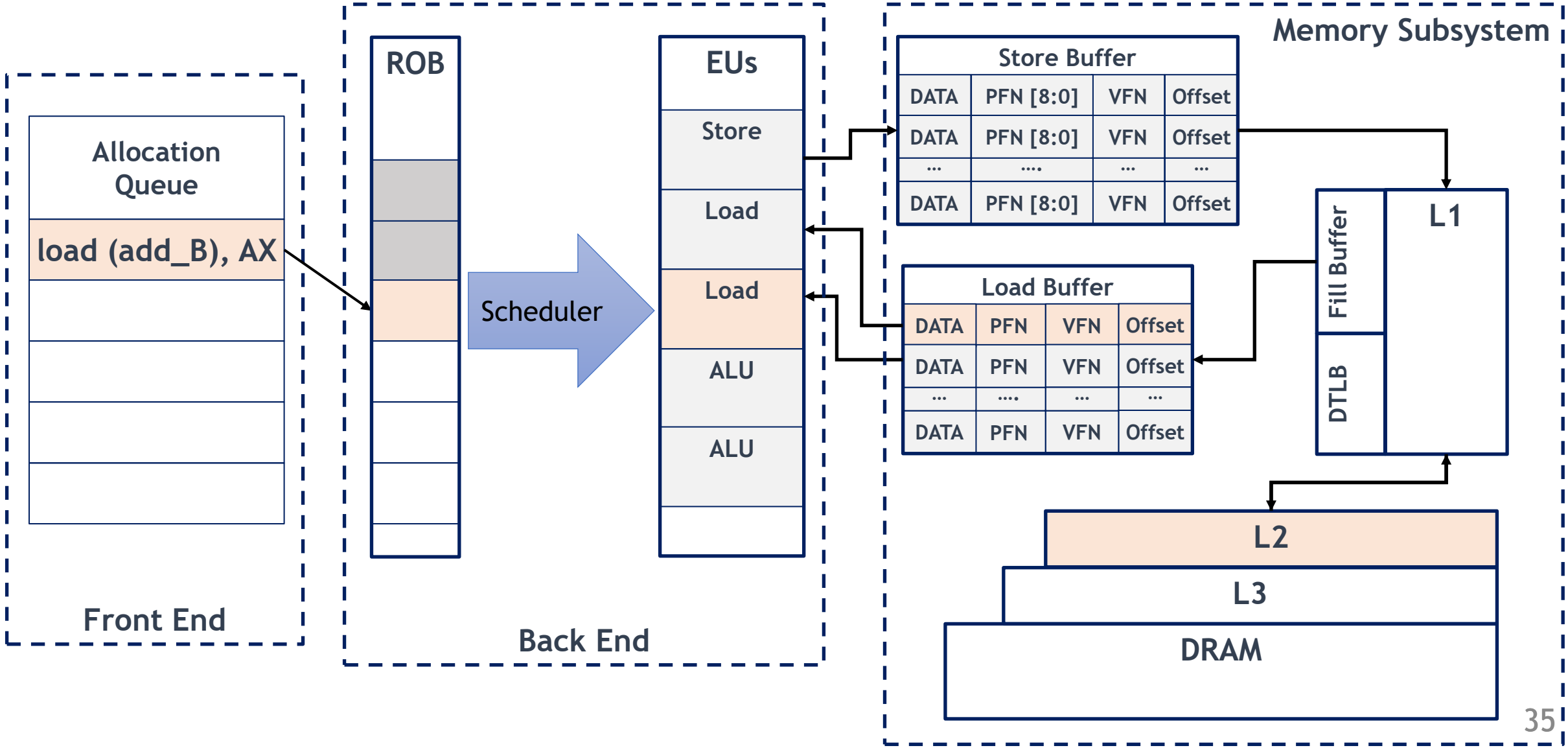
CPU Memory Subsystem



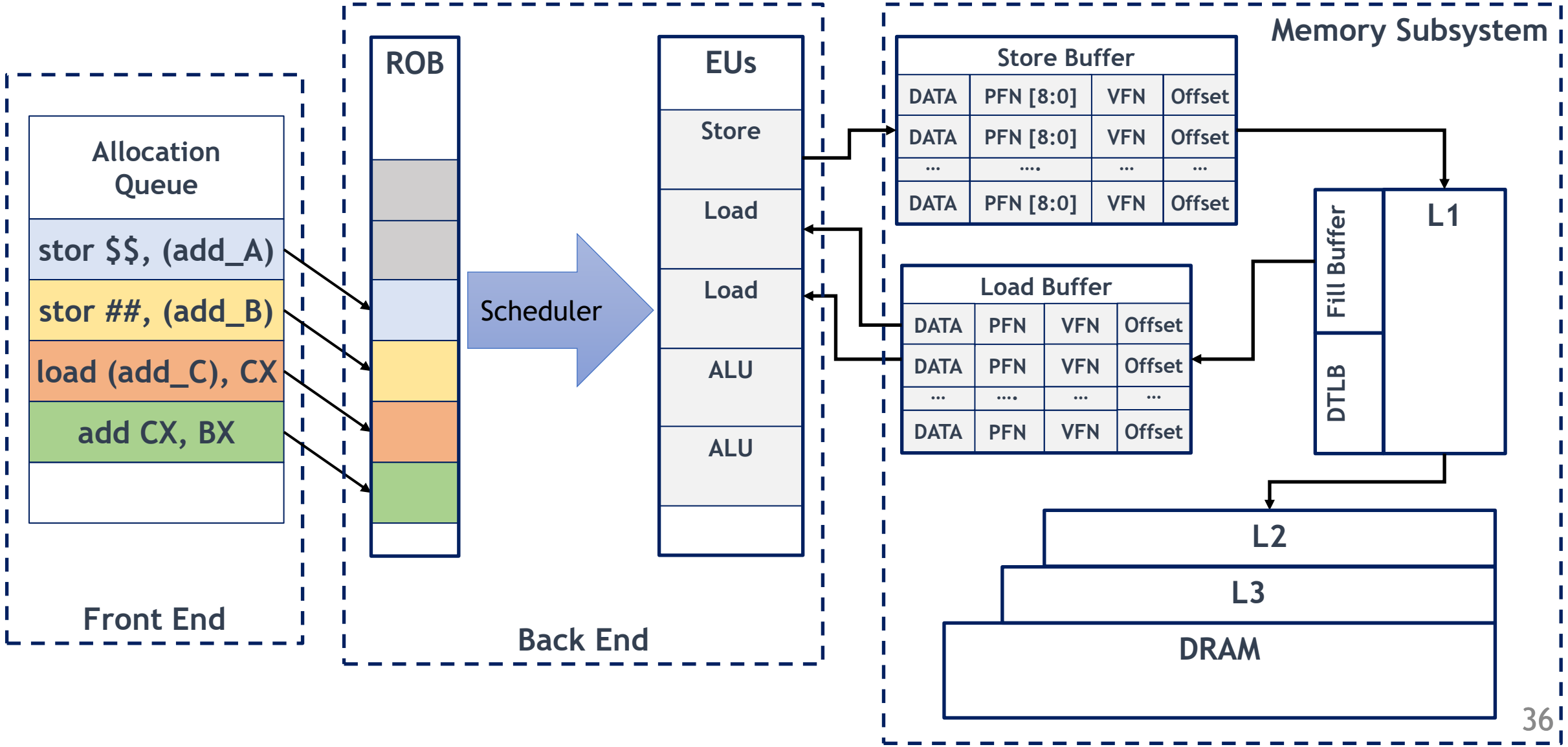
CPU Memory Subsystem



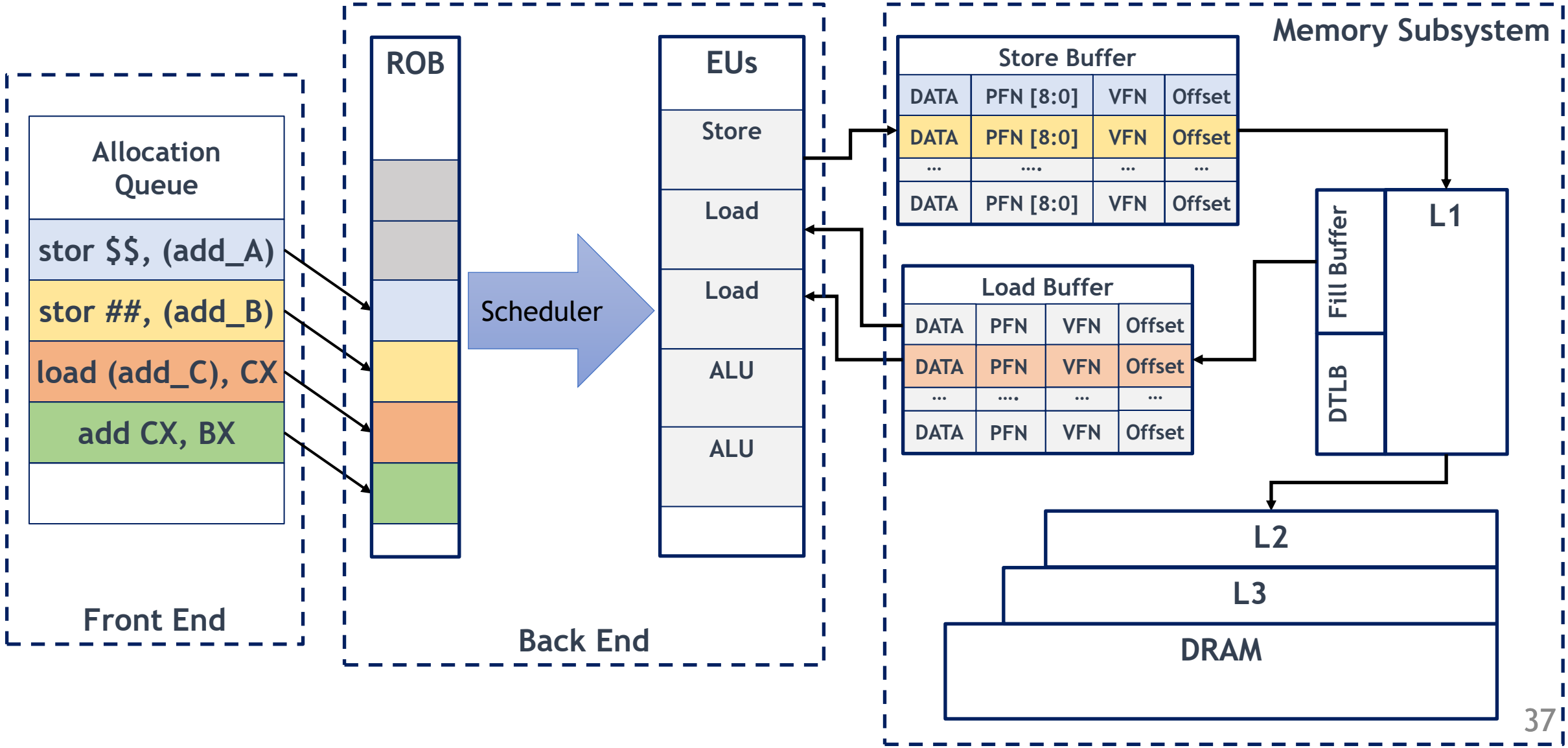
CPU Memory Subsystem



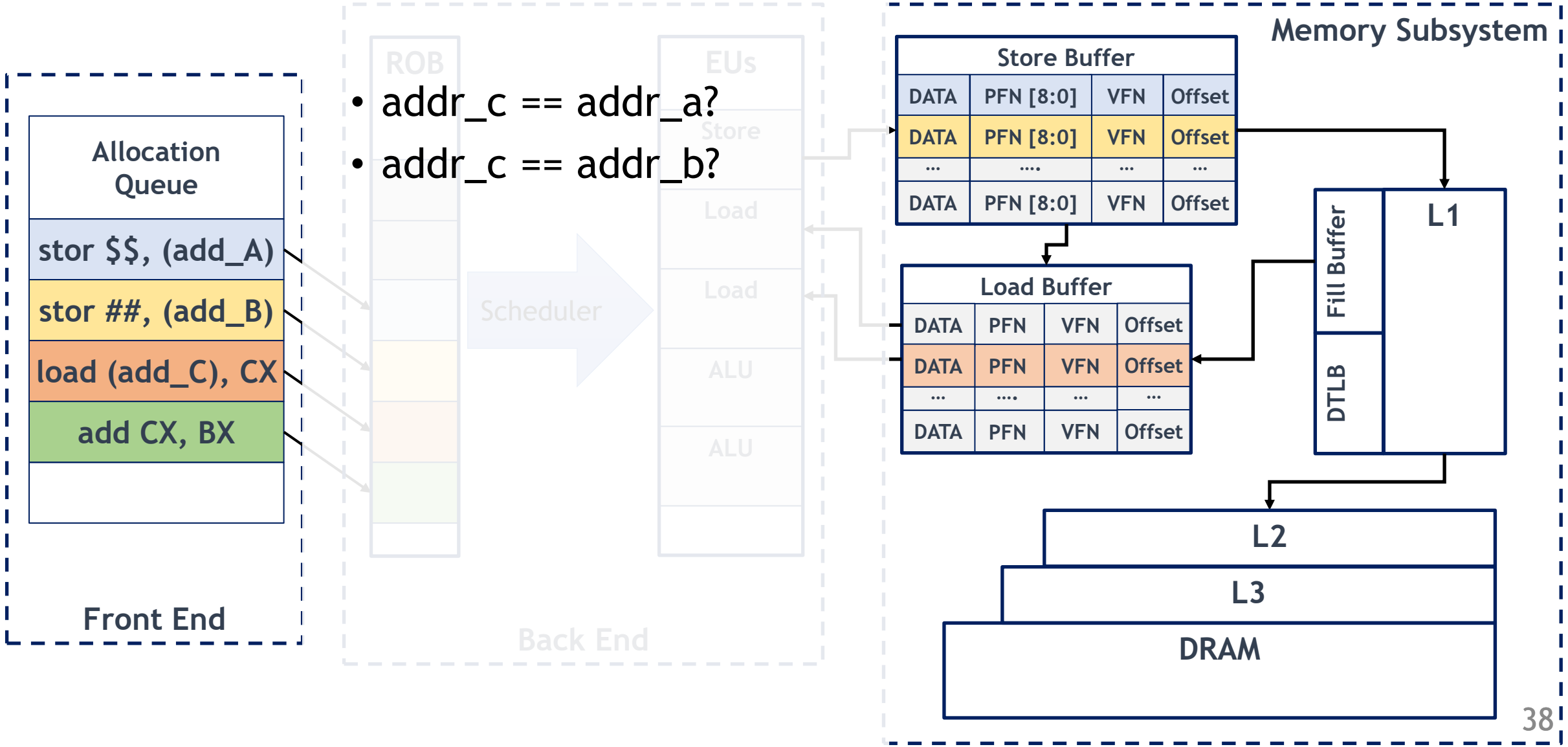
CPU Memory Subsystem

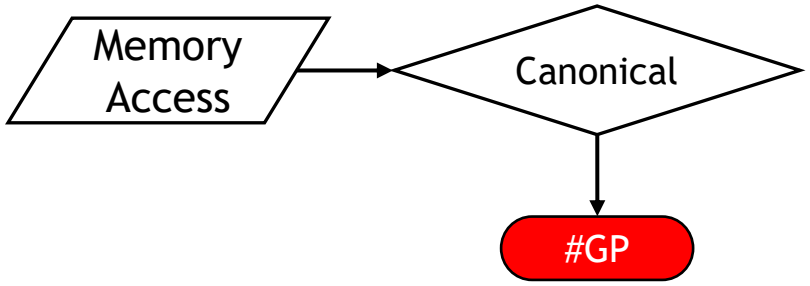


CPU Memory Subsystem - Store Forwarding

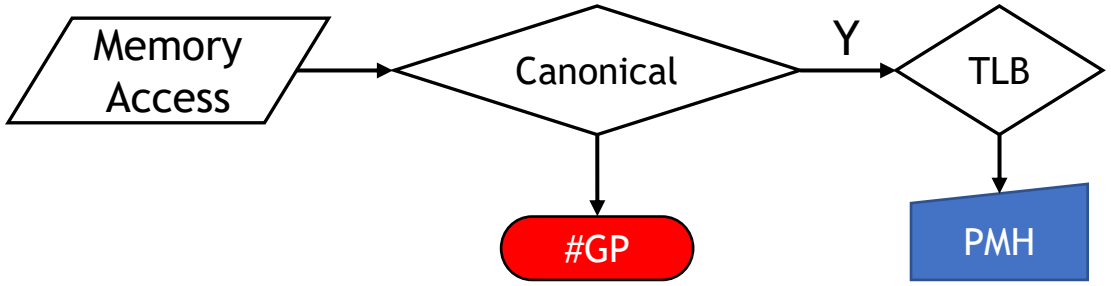


CPU Memory Subsystem - Store Forwarding





Virtual Address

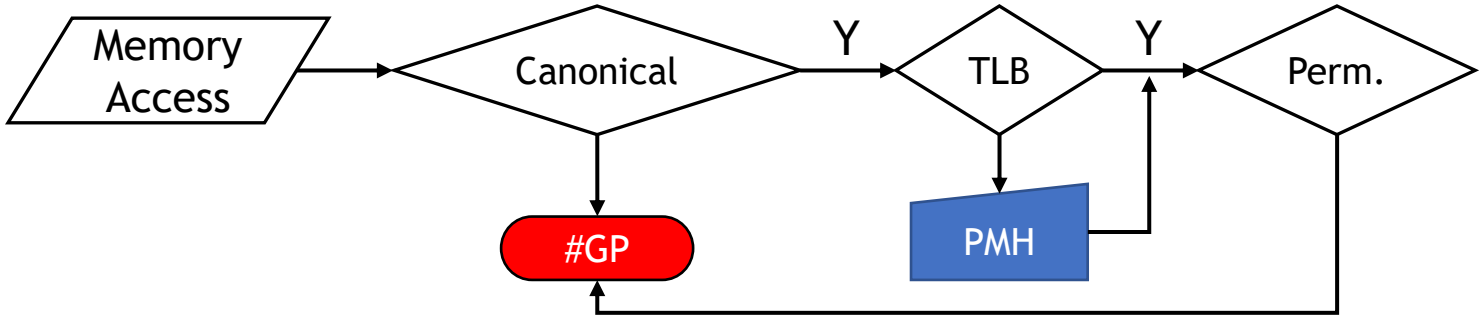


Virtual Address

VFN	Offset

PTE

P	RW	US	…	A	…	Physical Page Number	…

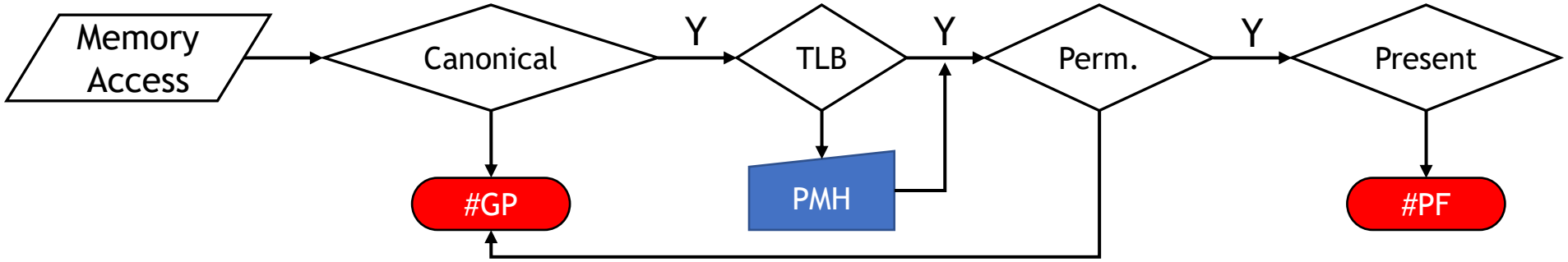


Virtual Address

VFN	Offset

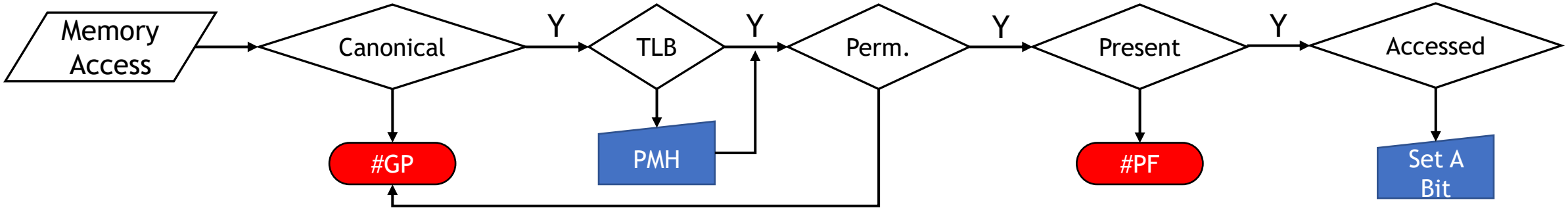
PTE

P	RW	US	…	A	…	Physical Page Number	…



Virtual Address

PTE

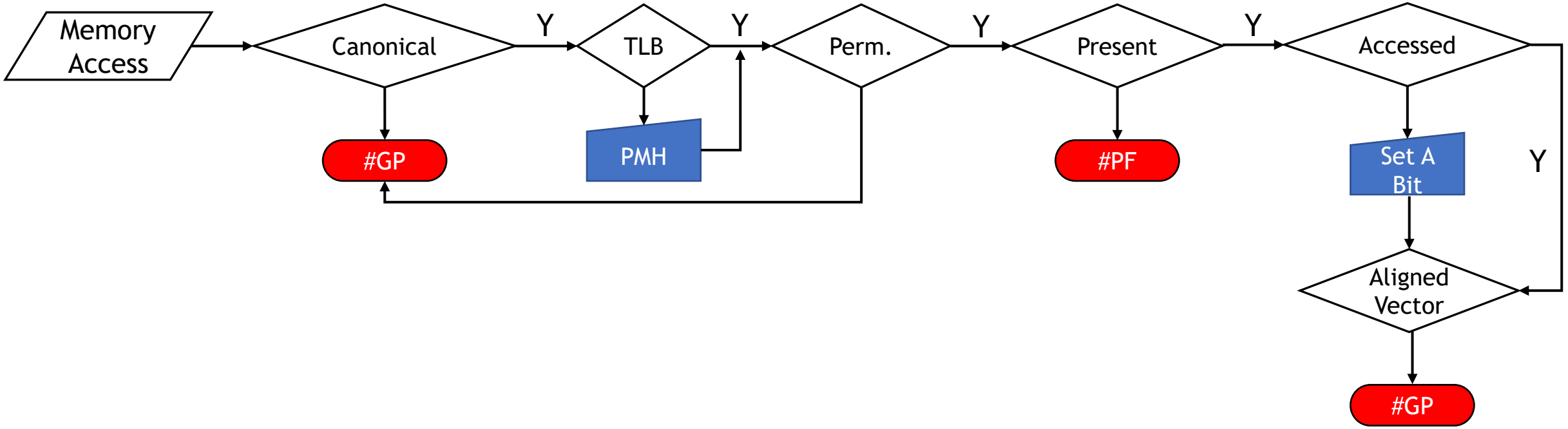


Virtual Address

VFN	Offset

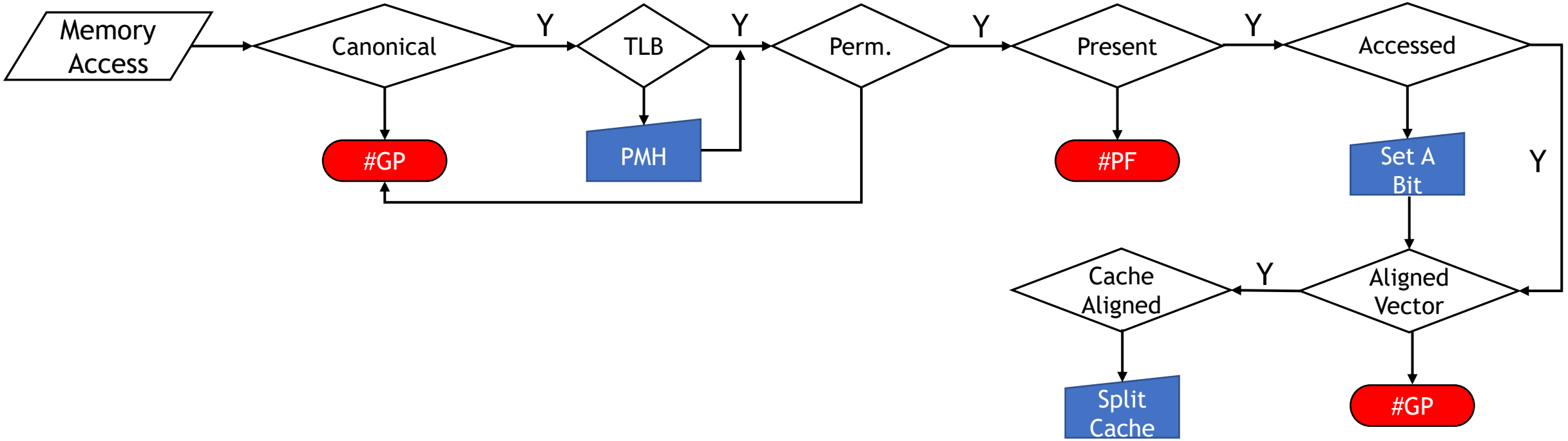
PTE

P	RW	US	…	A	…	Physical Page Number	…



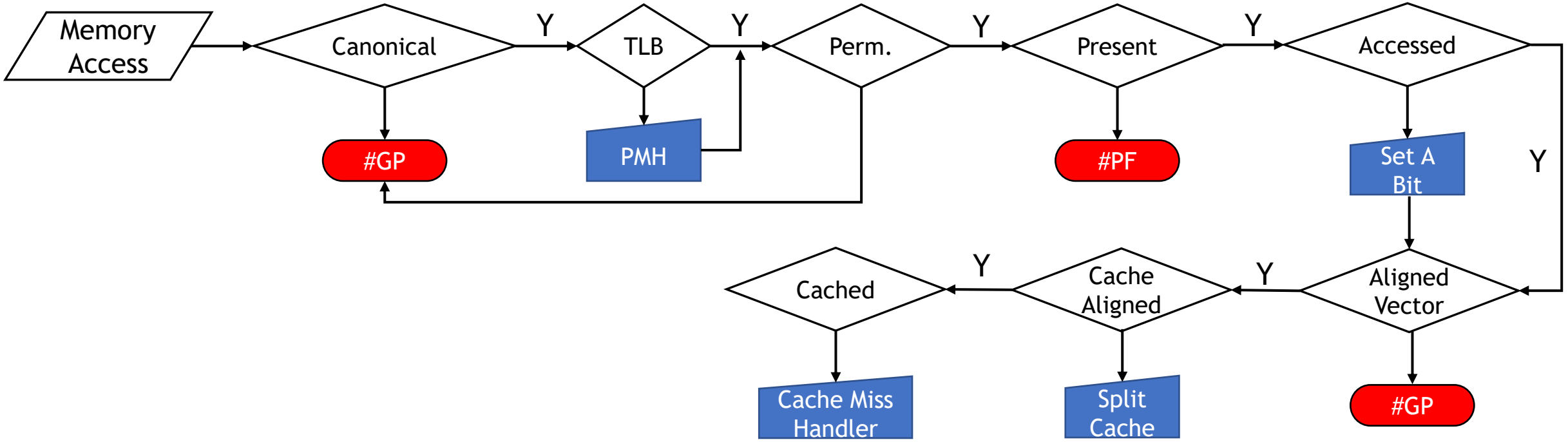
Virtual Address

PTE



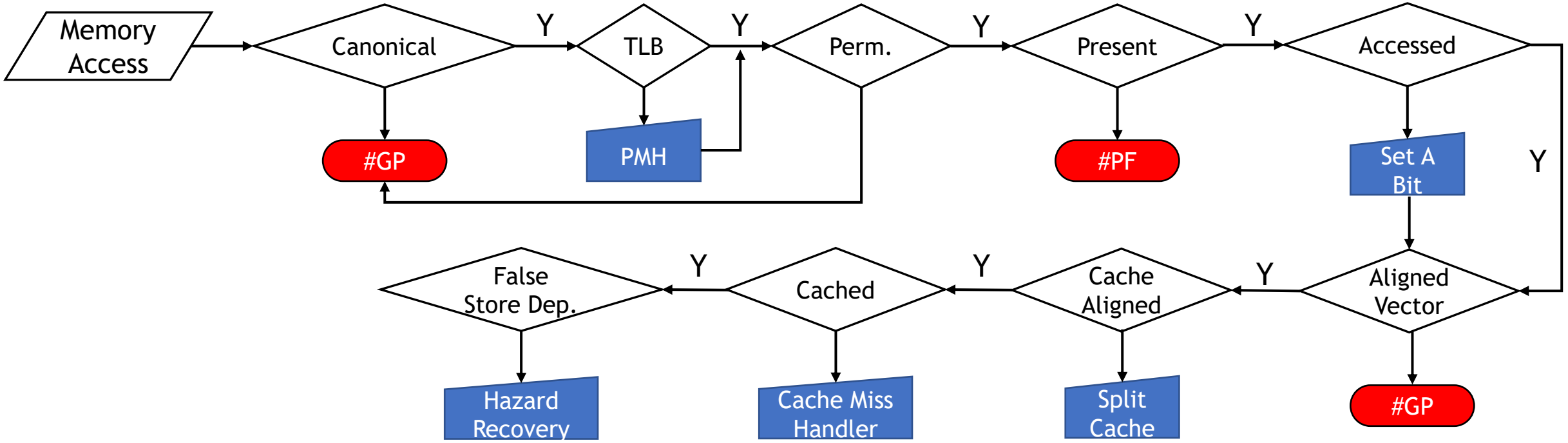
Virtual Address

PTE



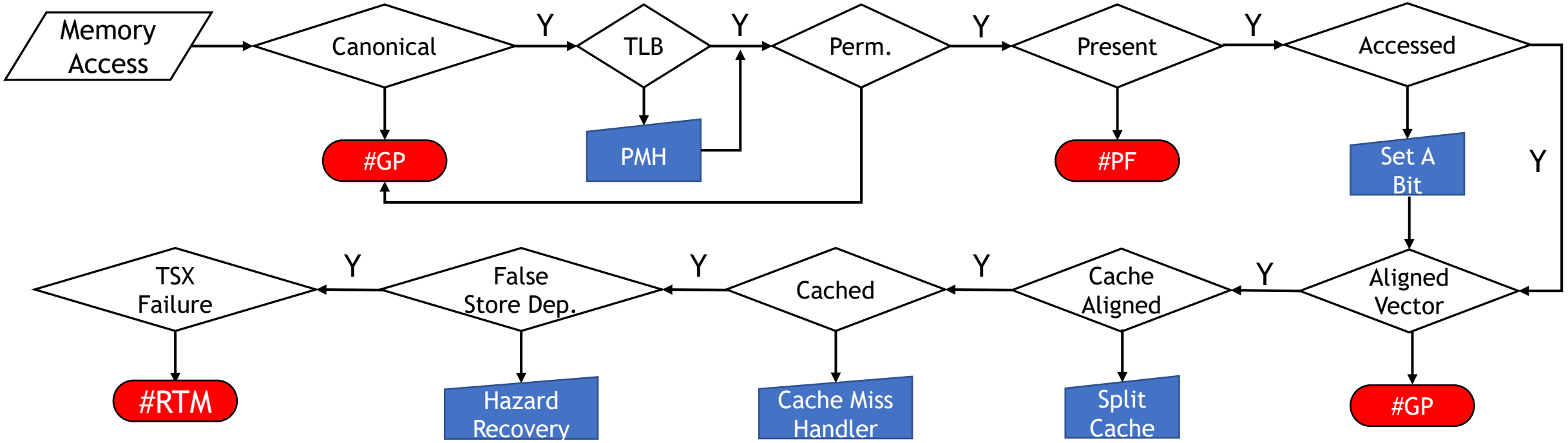
Virtual Address

PTE



Virtual Address

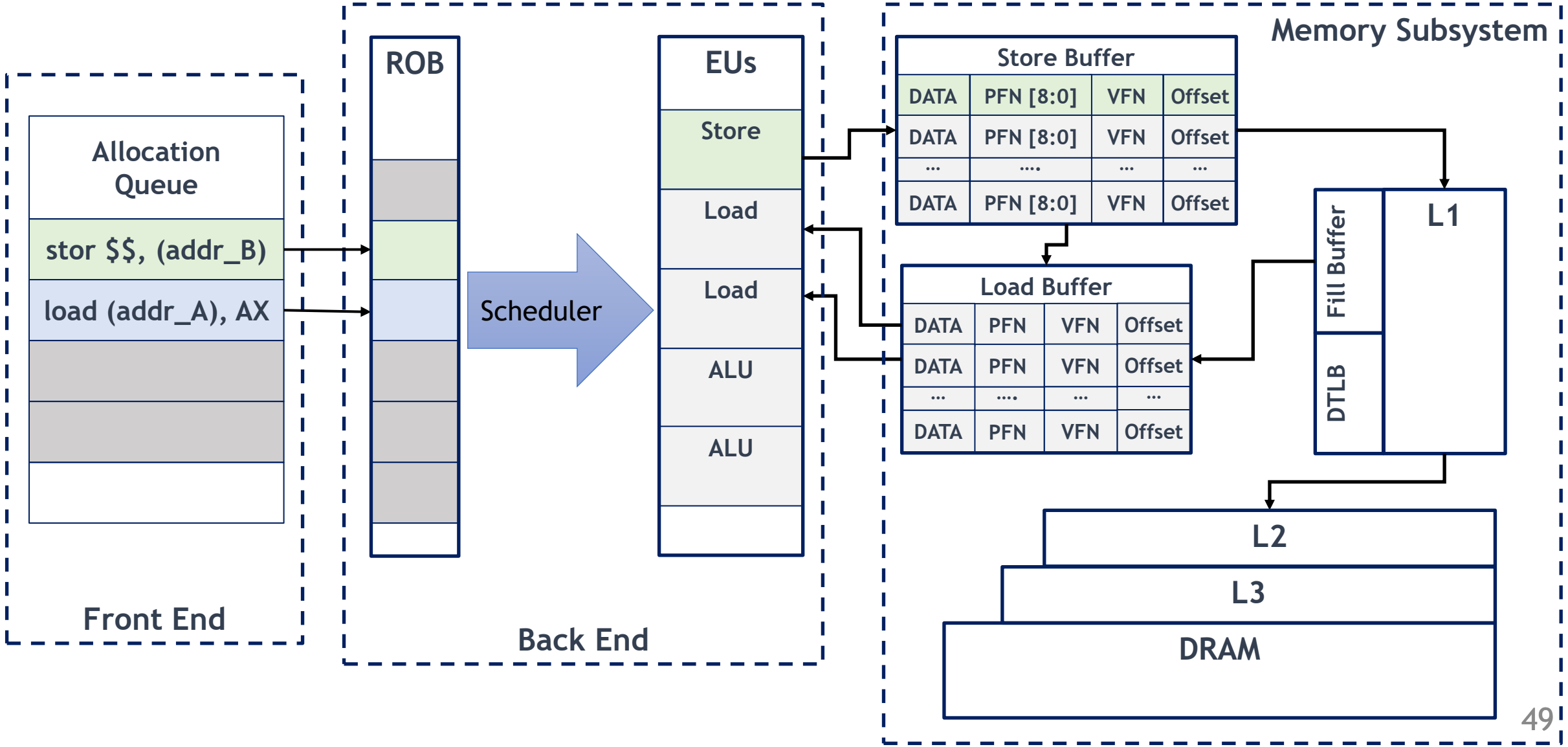
PTE



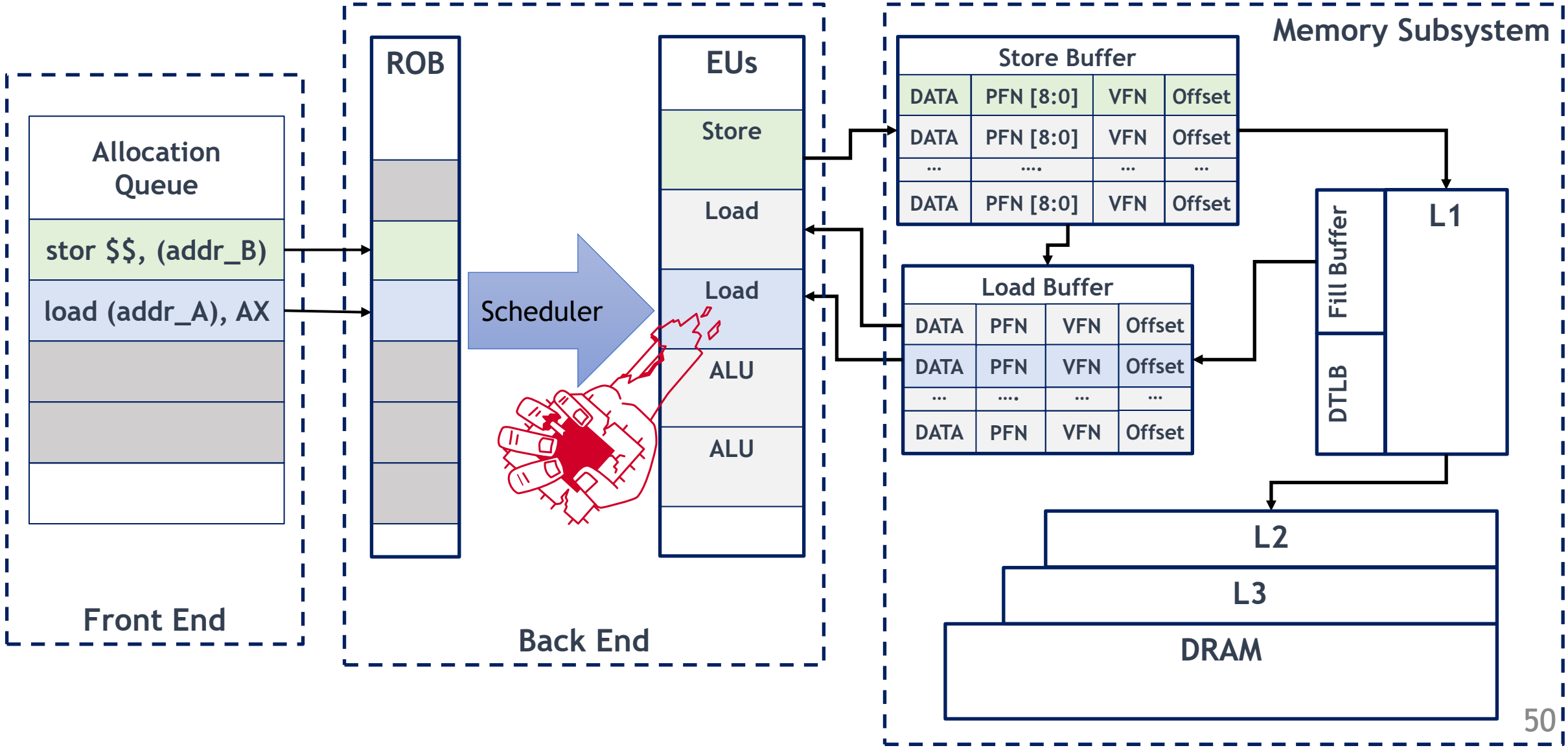
Virtual Address

PTE

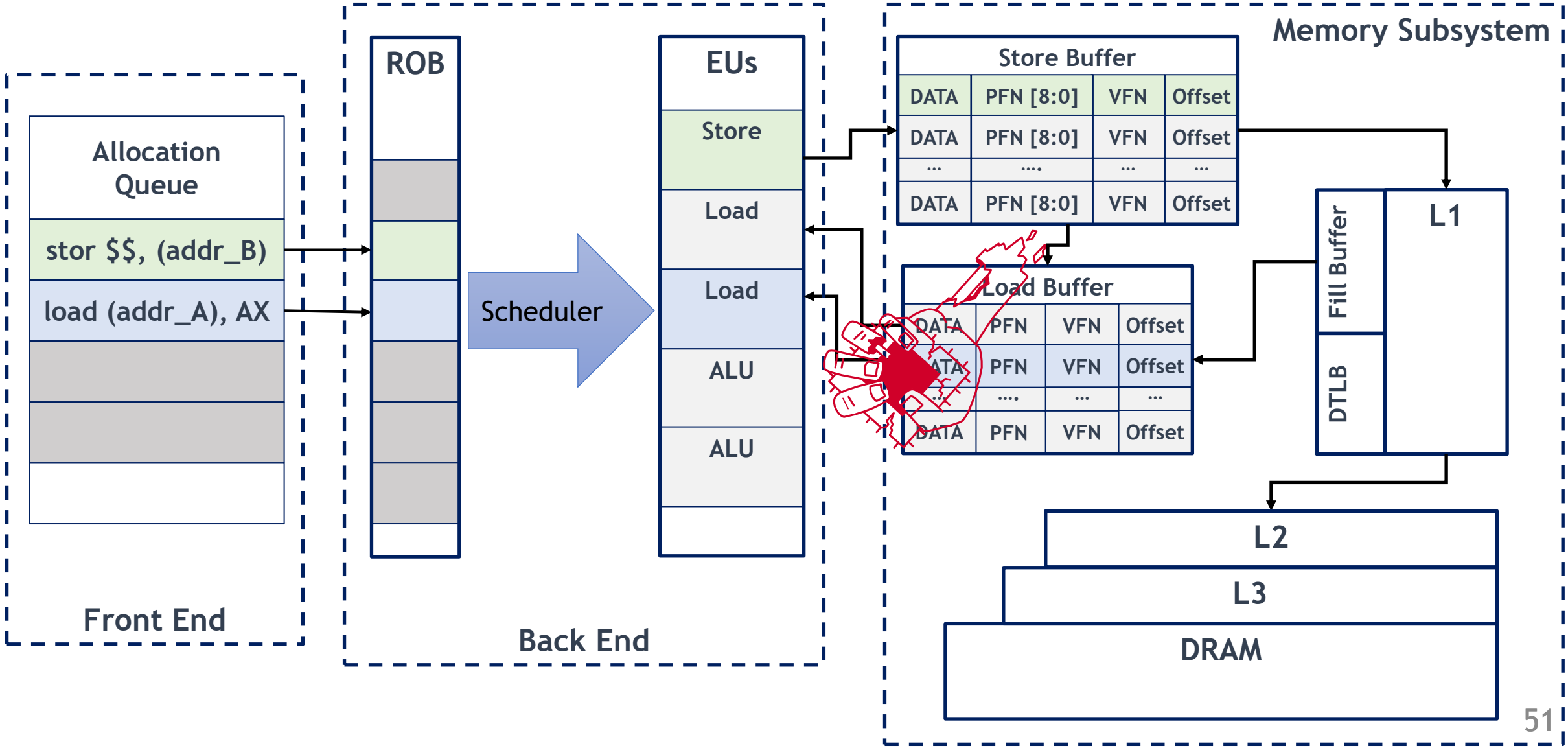
CPU Memory Subsystem - Hazard Recovery



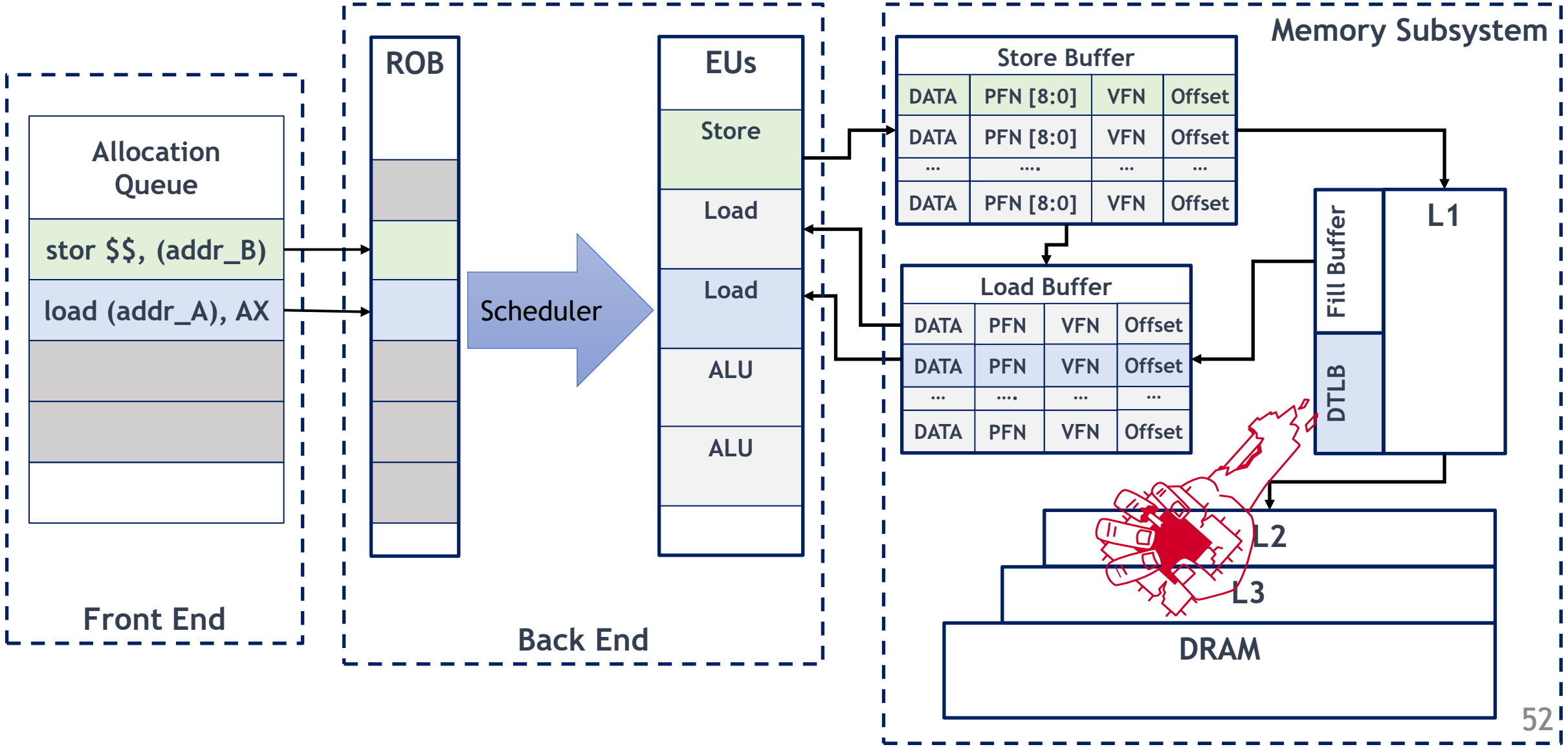
CPU Memory Subsystem - Hazard Recovery



CPU Memory Subsystem - Hazard Recovery



CPU Memory Subsystem - Hazard Recovery



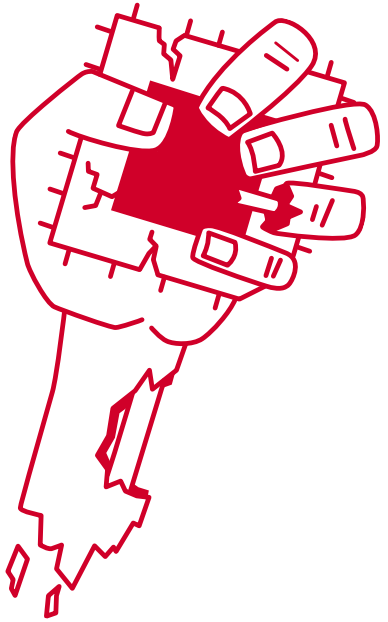
Challenges with MDS Testing?

- Reproducing attacks is not reliable. It may depend on:
  - massaging the pipeline with other instructions
  - CPU configuration (generation, frequency, microcode patch and etc)

- No public tool to find new variants or to verify hardware patches:
  - Too many things to test (Addressing mode, cache state, assists, and faults)
  - Previous POCs may not work after MC update, but what does it mean?

- Impossible to quantify the impact of leakage:
  - We should care about leakage rate and what data is leaked.
  - My POC is faster than your POC!!

Let’s see this problem in action?! (Demo)



Transynther

Transynther (Fuzzing-based Random MDS Testing)

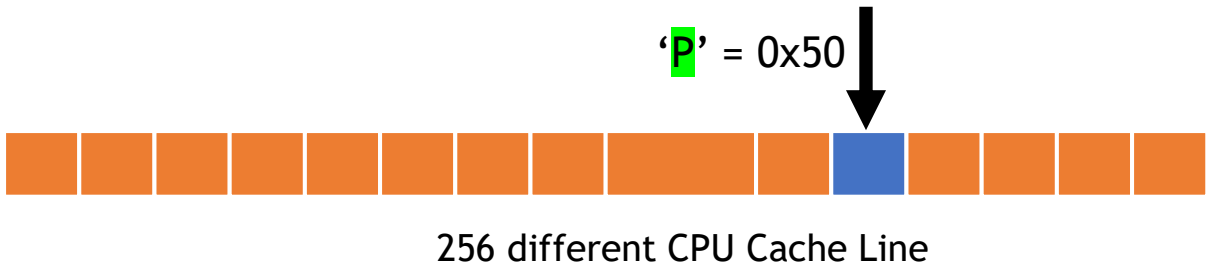
Step 1:

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

Step 2:

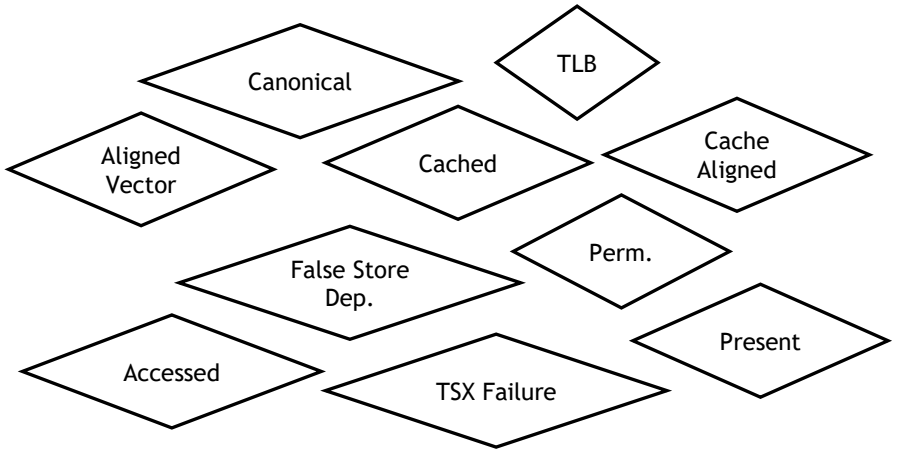
```
char x = oracle[secret * 4096];
```

Step 3:



Transynther (Fuzzing-based Random MDS Testing)

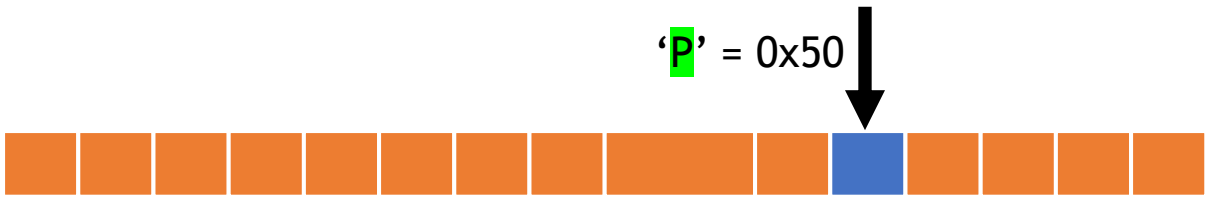
Step 1:



Step 2:

```
char x = oracle[secret * 4096];
```

Step 3:



256 different CPU Cache Line

Transynther (Fuzzing-based Random MDS Testing)

Step 0: Buffer Grooming

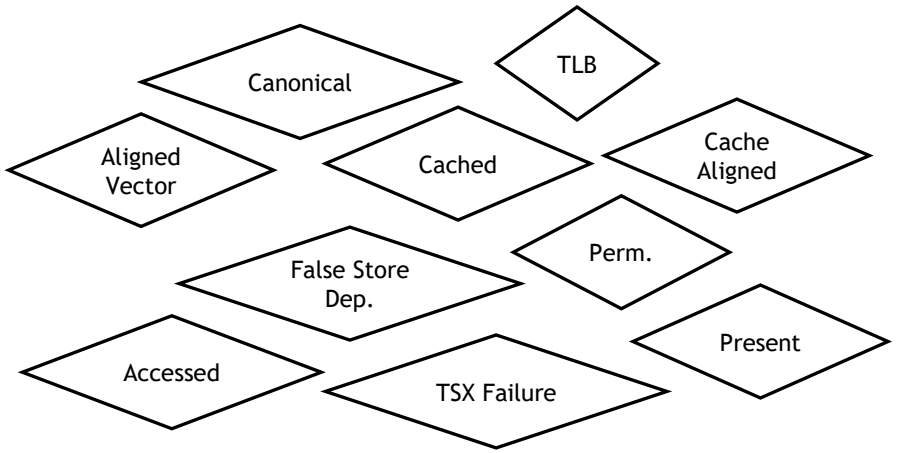
Stores Same Thread: 0x41424344

Loads Same Thread: 0x51525354

Stores Hyper Thread: 0x61626364

Loads Hyper thread Thread: 0x71727374

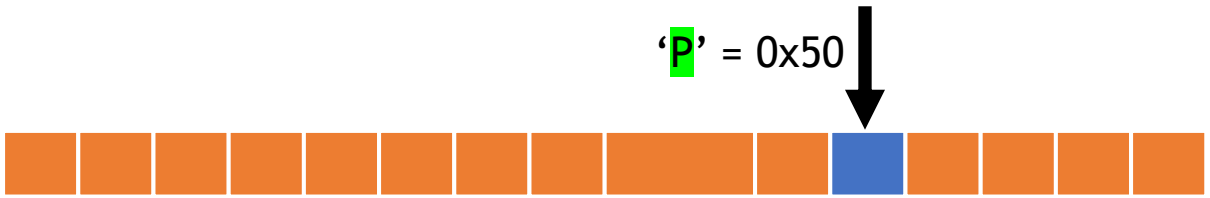
Step 1:



Step 2:

```
char x = oracle[secret * 4096];
```

Step 3:



256 different CPU Cache Line

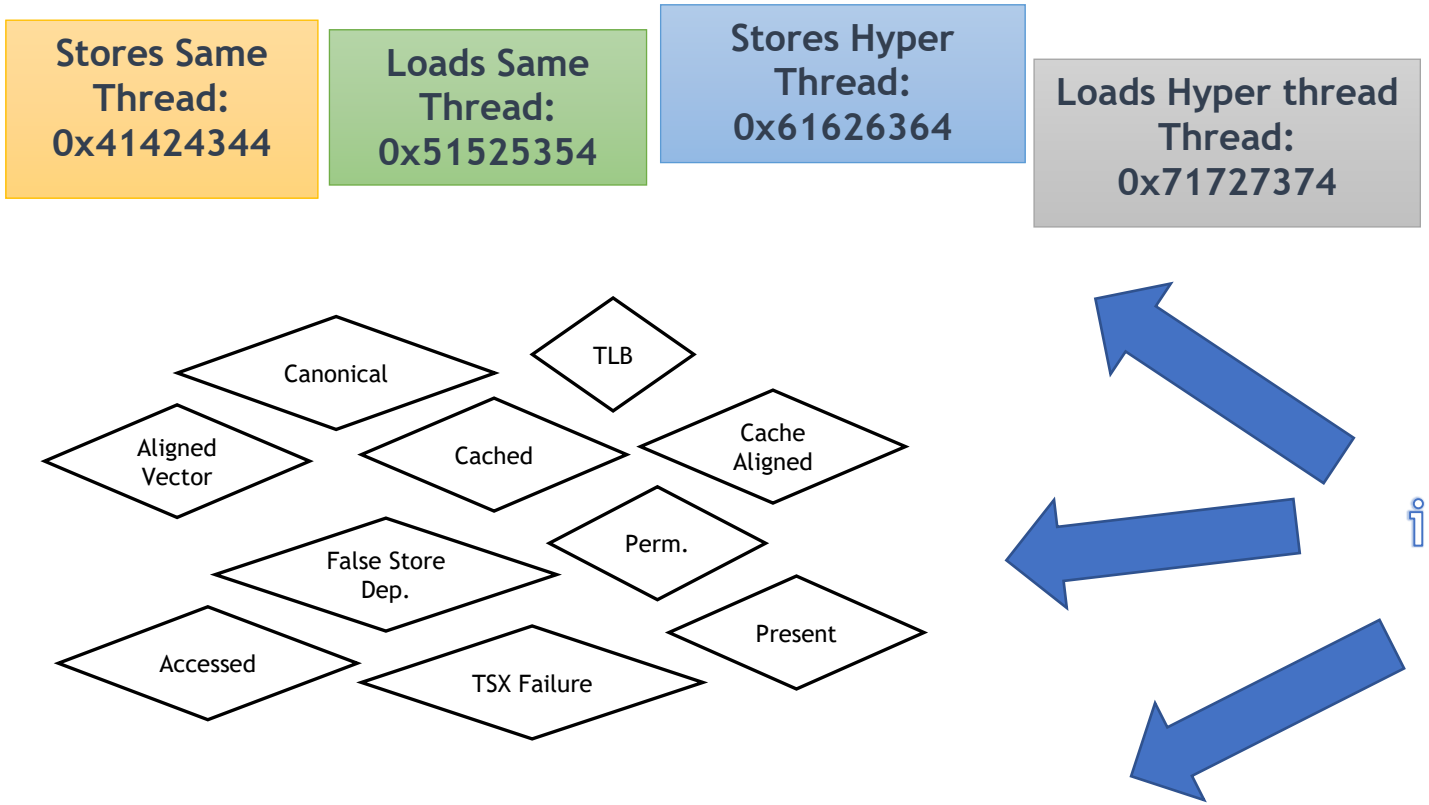
Transynther (Fuzzing-based Random MDS Testing)

Step 0: Buffer Grooming

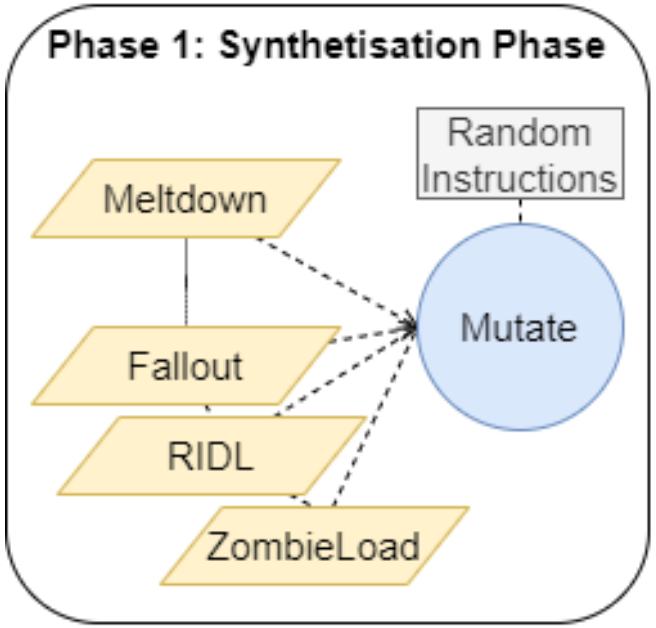
Step 1:

Step 2:

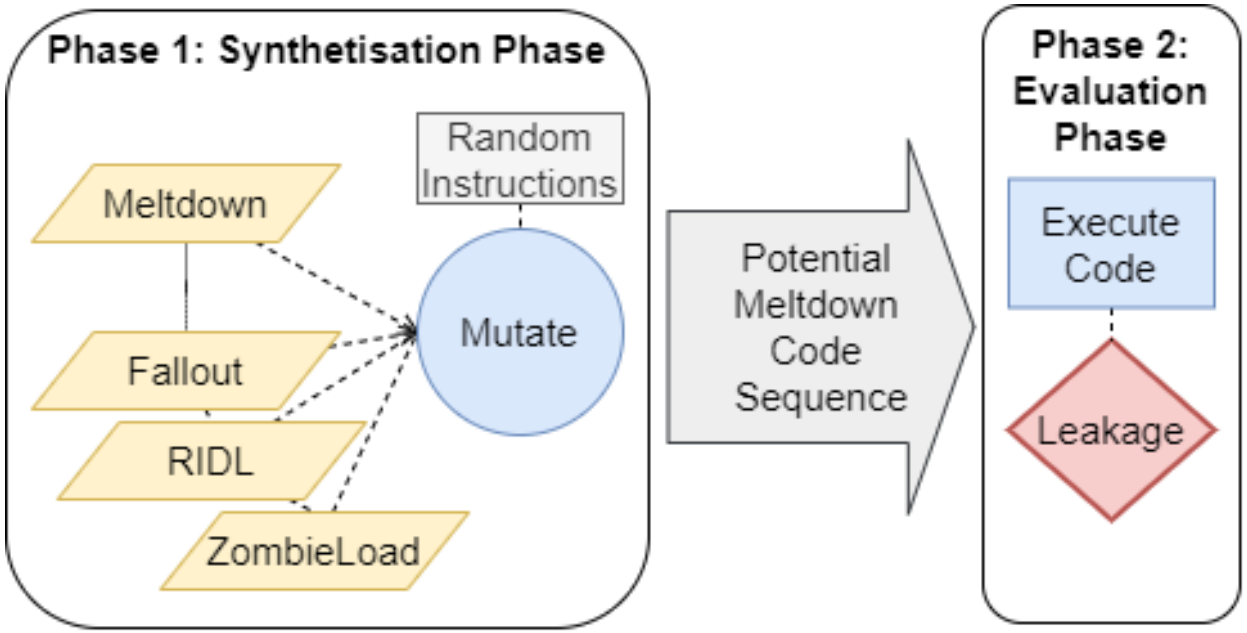
Step 3:



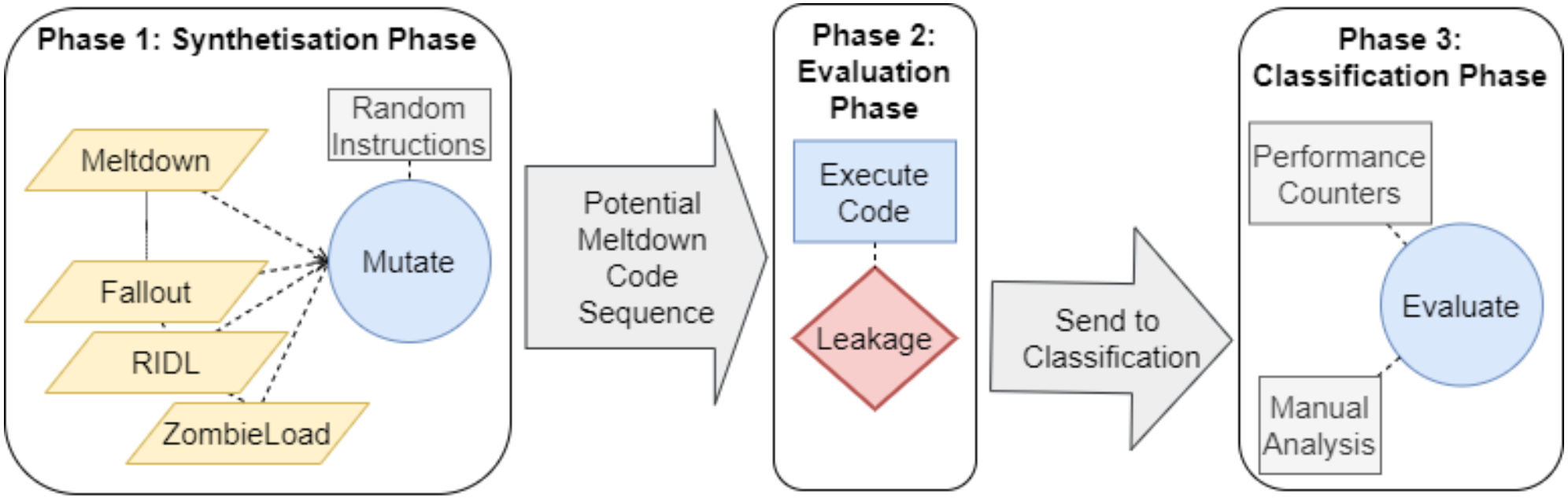
Transynther (Fuzzing-based MDS Testing)



Transynther (Fuzzing-based MDS Testing)



Transynther (Fuzzing-based MDS Testing)



Transynther Demo

Table 2: Leakage variants discovered by Transynther.

Case	Preparation	Store	Load	Name
①	(access ⊘, random instructions)	-	⤳ + 🔒 / ✋ / ⊘	MLPDS
②	(access ⊘, random instructions)	-	AVX ⤳ + 🔒 / ✋ / ⊘	MLPDS
③	(access ⊘, random instructions)	-	AVX + 🔒 / ✋ / <✖>	Medusa
④	(access ⊘, random instructions)	-	AVX ⇄ + 🔒 / ✋ / ⊘ / <✖> / ✓	Medusa
⑤	-	store (to load)	🔒 / ✋ / <✖> / ✓	S2L
⑥	(`rep mov` + store, store + fence + load)	store (to load)	🔒 / ✋ / <✖> / ✓	-
⑦	-	store (4K Aliasing) + 🔒 / ✋ / ⊘ / <✖> / ✓	🔒 / ✋	MSBDS
⑧	-	store (4K Aliasing, to load) + 🔒 / ✋ / ⊘ / <✖> / ✓	AVX ⇄ + 🔒 / ✋ / ⊘ / <✖> / ✓	MSBDS, S2L
⑨	(Sibling on/off)	store (random address) + ⊘	🔒 / <✖>	MSBDS
⑩	(Sibling on/off + clflush (store address))	store (Cache Offset of Load) + ⊘	🔒 / <✖>	MSBDS
⑪	(Sibling on/off + repmov (to Load))	store (to Load)	AVX ⇄ + 🔒 / ✋ / ⊘ / <✖> / ✓	Medusa, MLPDS
⑫	-	Store (Unaligned to Load)	🔒 / ✋ / <✖>	Medusa
⑬	(random instructions)	AVX Store (to Load)	<✖>	Medusa, MLPDS, MSBDS
⑭	-	random fill stores	<✖>	MSBDS

<✖> Non-canonical Address Fault ⊘ Non-present Page Fault 🔒 Supervisor Protection Fault ⇄ AVX Alignment Fault
✋ Access-bit Assist ⤳ Split-Cache Access Assist ✓ Access without fault or Assist

Demo some Interesting Leakage Pattern

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? (First come first serve)

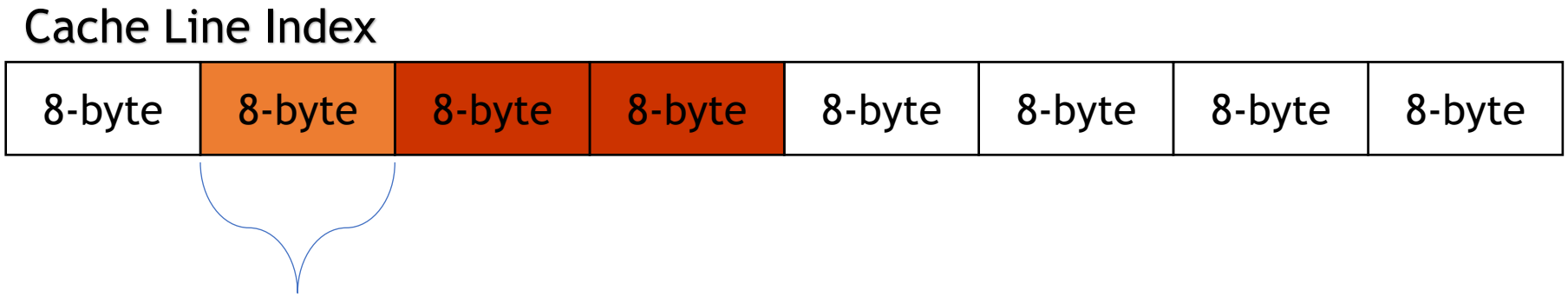
MEDUSA

Medusa Attack

- Write Combining fills up the entire Data Bus.
- Medusa only leaks the Upper-half of the Data Bus.
- Implicit WC, i.e., ‘rep mov’, ‘rep stos’, can be leaked.
- Served by a Write Combining Buffer (or just the the Fill Buffer)

- Advantages:
  - Prefiltered data
  - Less Noise
  - More targeted (maybe also a disadvantage)

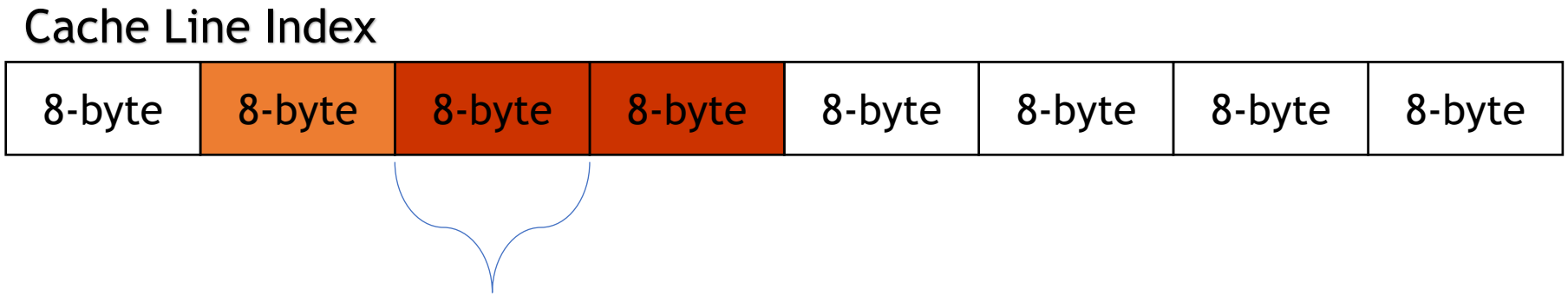
Medusa Attack – V1 Cache Indexing



Faulty Load

An invalid (Non-canon) address:
0x5550000000000008-20

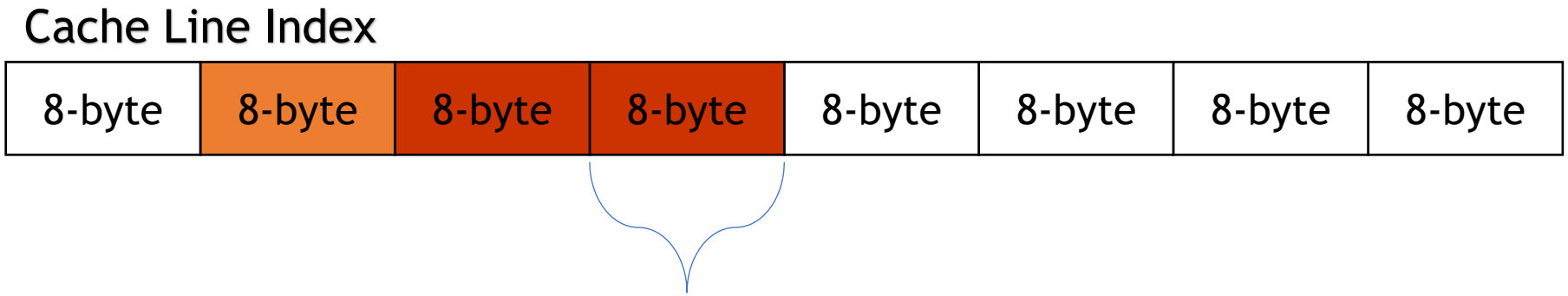
Medusa Attack – V1 Cache Indexing



Faulty Load

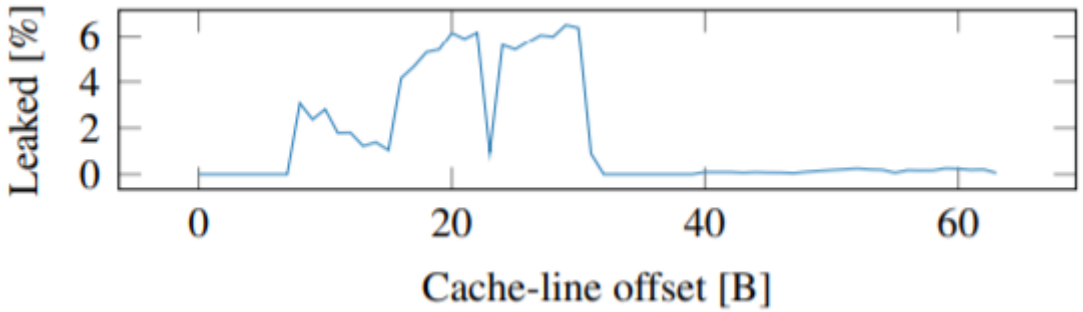
An invalid (Non-canon) address:
0x5550000000000008-20

Medusa Attack – V1 Cache Indexing

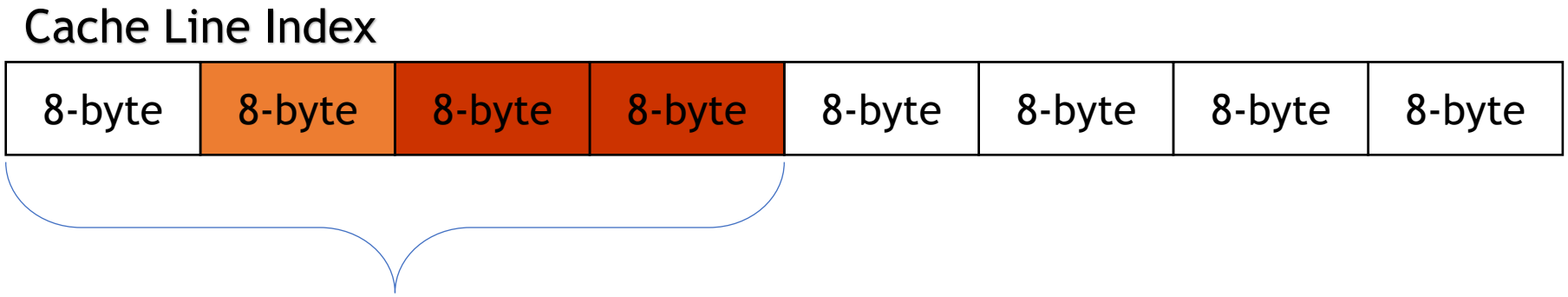


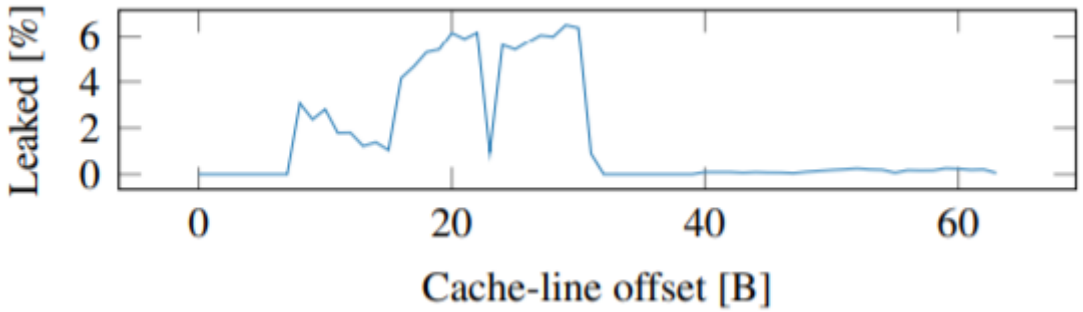
Faulty Load

An invalid (Non-canon) address:
0x5550000000000008-20

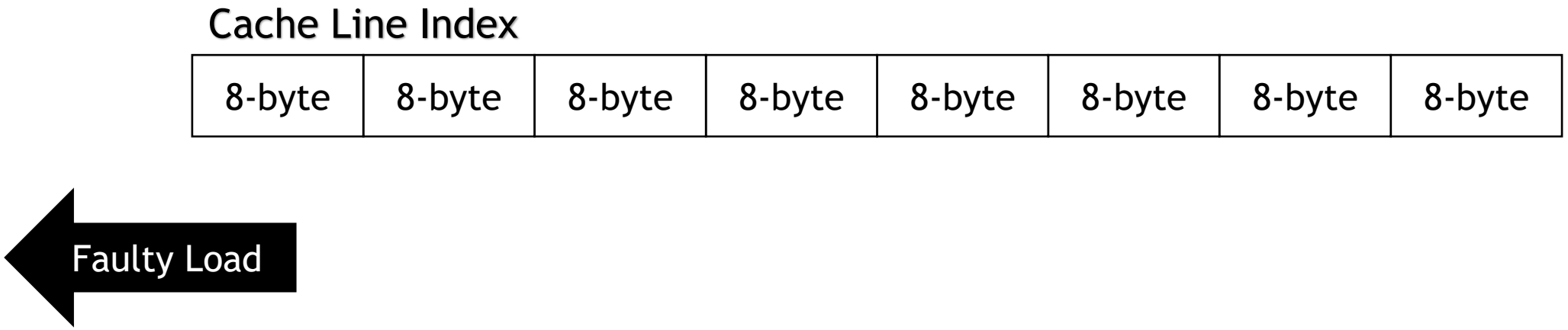


Medusa Attack – V1 Cache Indexing

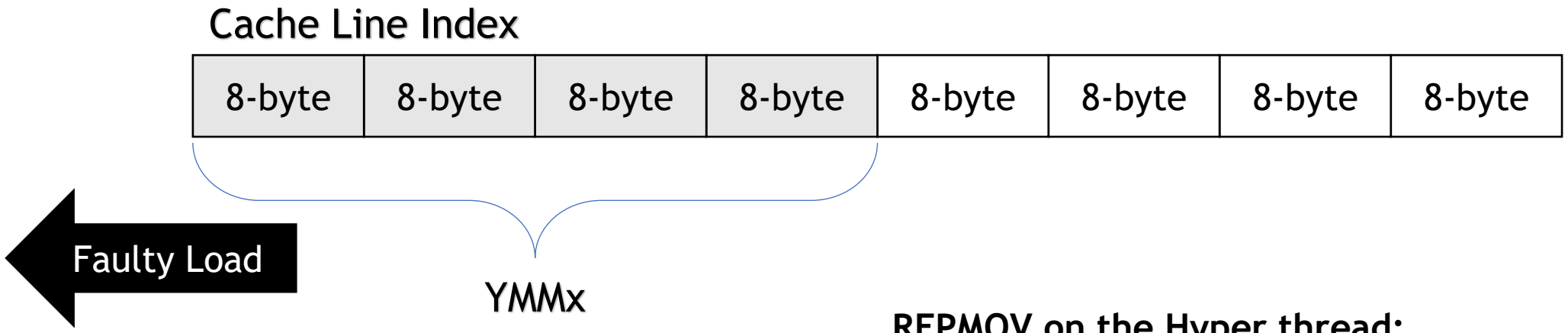




Medusa Attack – V2 Unaligned S2L Forwarding



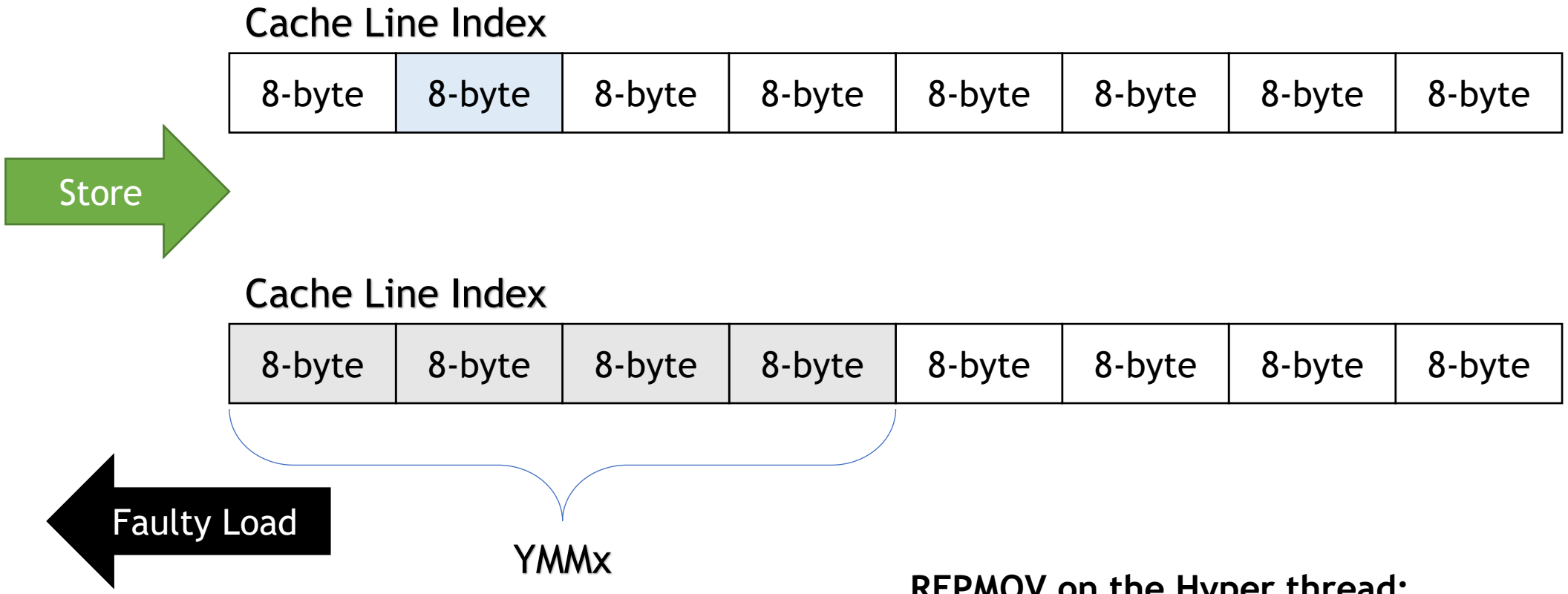
Medusa Attack – V2 Unaligned S2L Forwarding



REPMOV on the Hyper thread:

ABCDEFGH IJKLMNOP QRSTUVWX YZ…

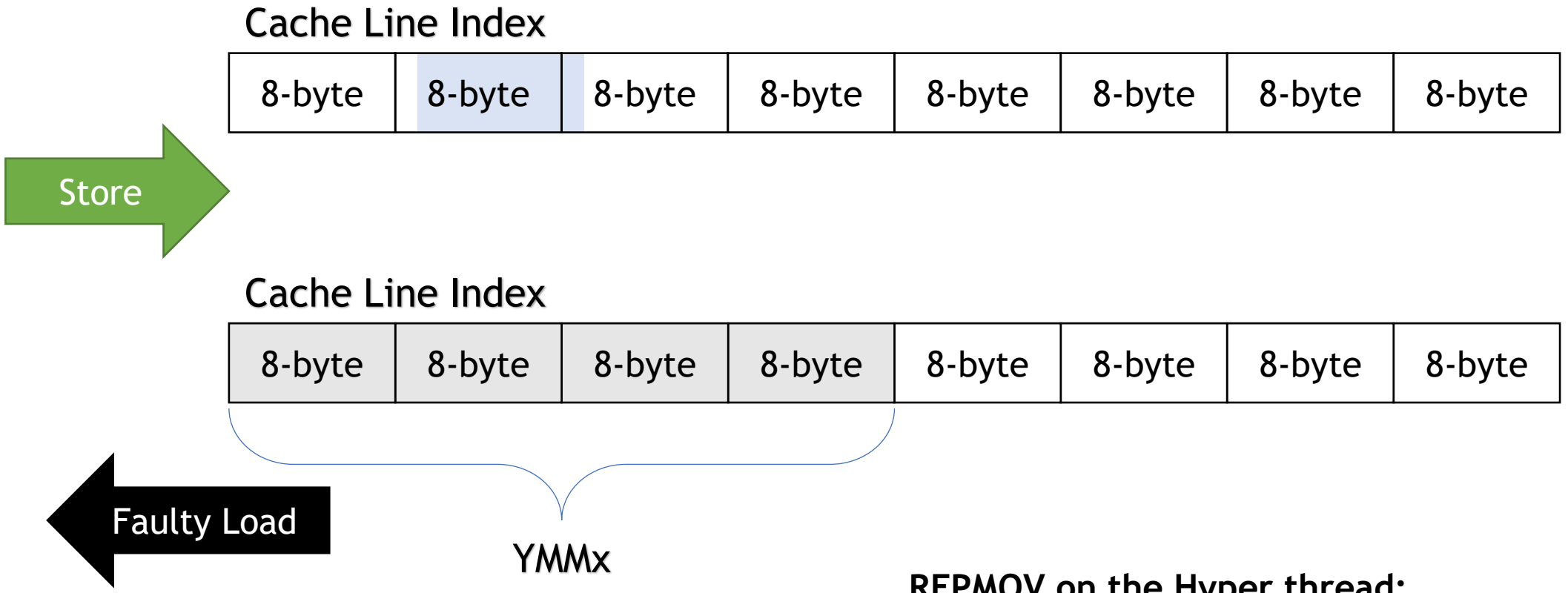
Medusa Attack – V2 Unaligned S2L Forwarding



REPMOV on the Hyper thread:

ABCDEFGH IJKLMNOP QRSTUVWX YZ…

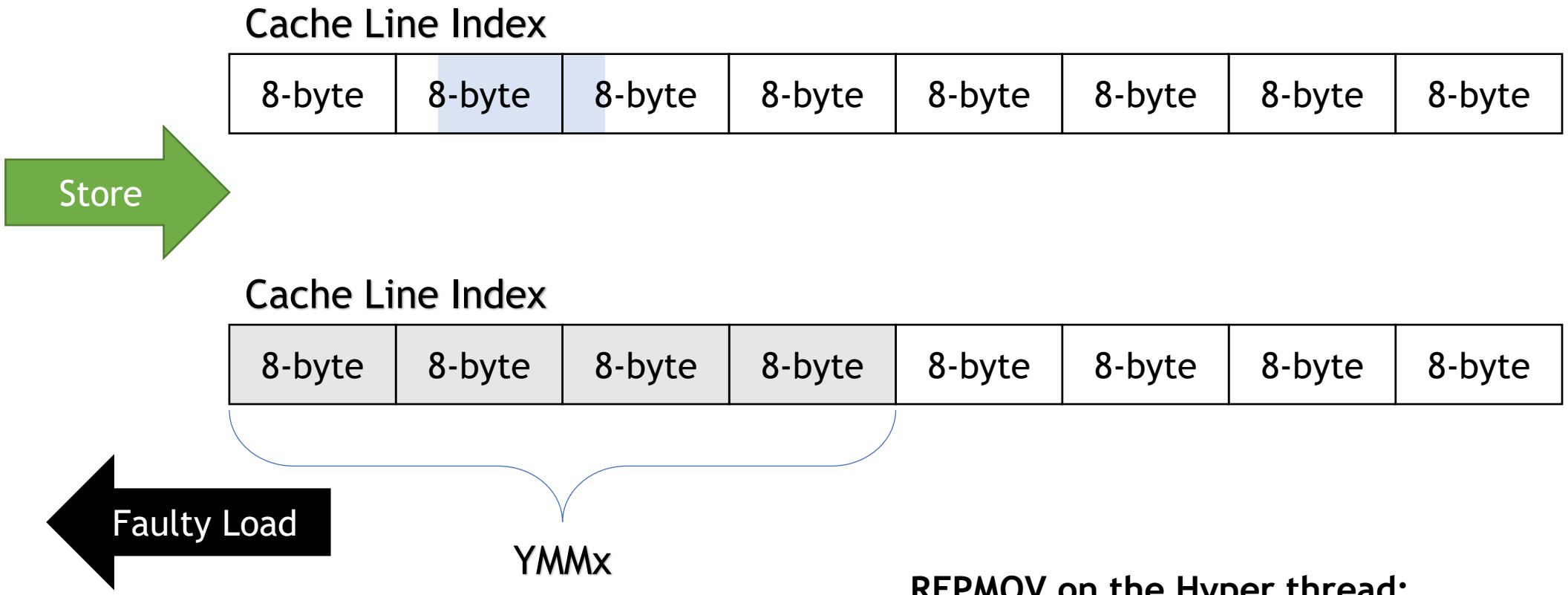
Medusa Attack – V2 Unaligned S2L Forwarding



REPMOV on the Hyper thread:

ABCDEFGH IJKLMNOP QRSTUVWX YZ…

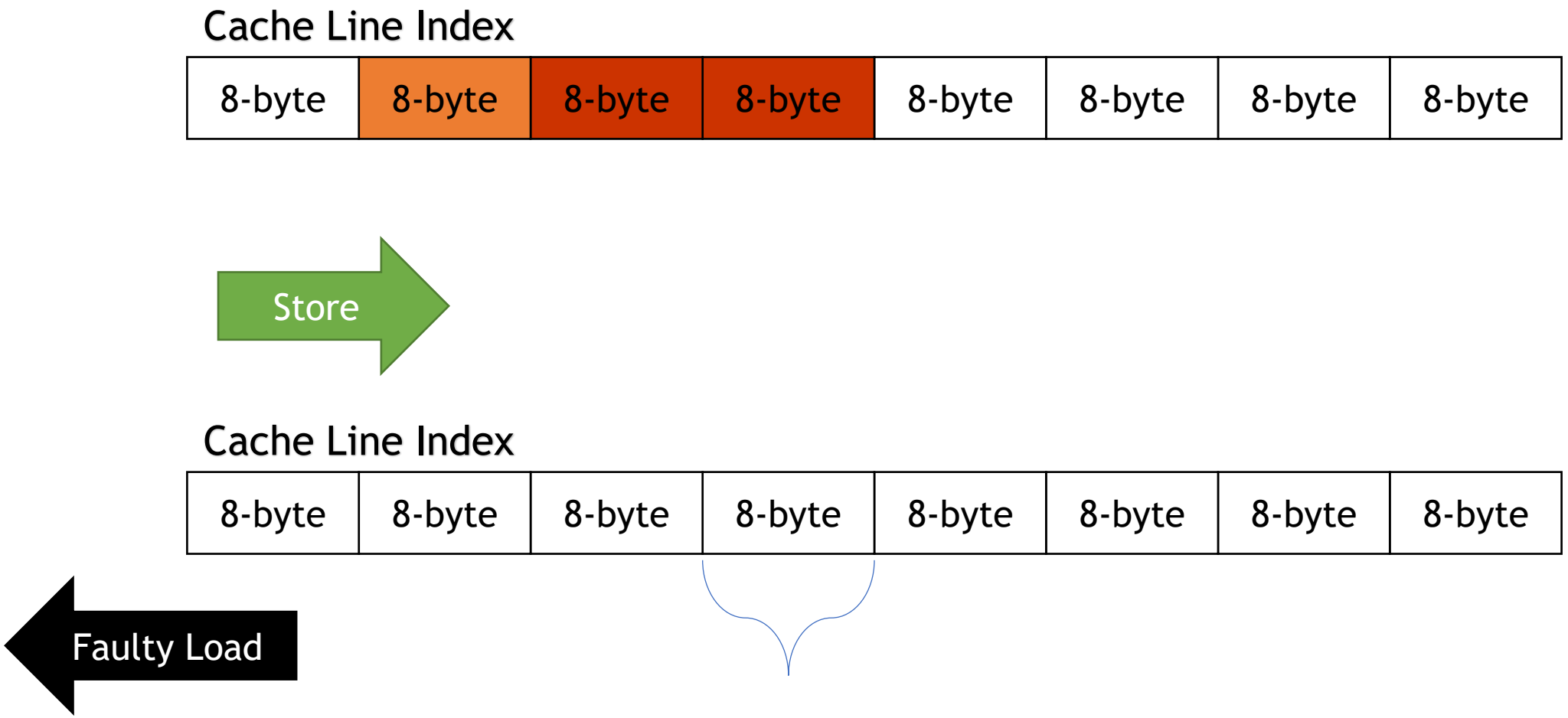
Medusa Attack – V2 Unaligned S2L Forwarding



REPMOV on the Hyper thread:

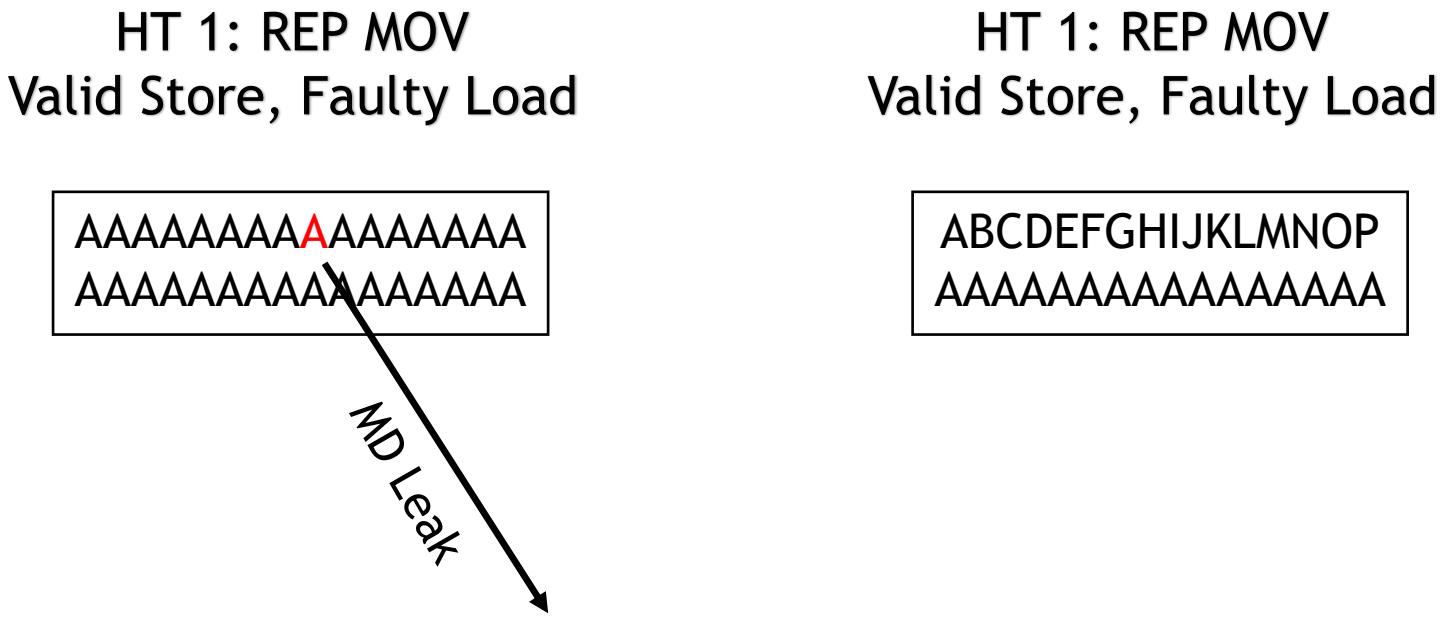
ABCDEFGH IJKLMNOP QRSTUVWX YZ…

Medusa Attack – V2 Unaligned S2L Forwarding



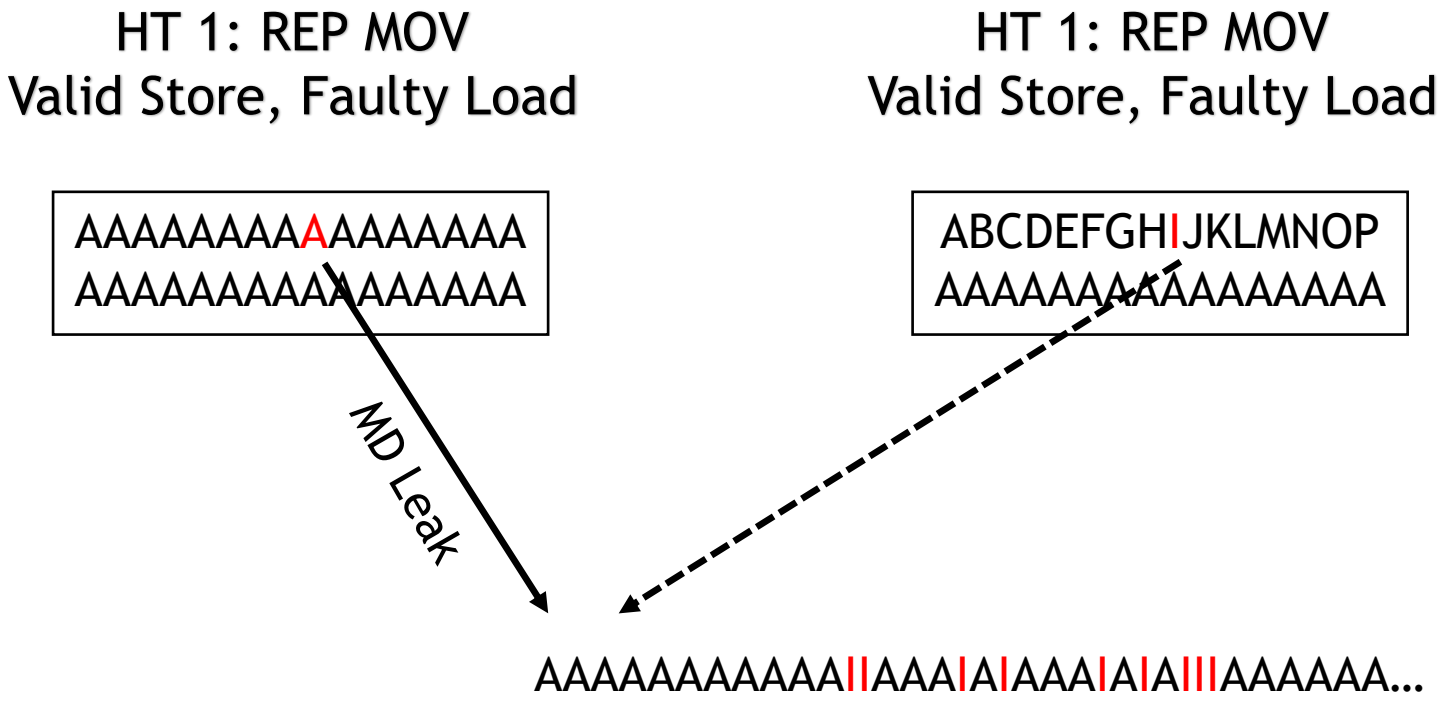
Medusa Attack – V3 Shadow *REP MOV*

- A *REP MOV* that fault on the load leaks:
  - the data from the legitimate store address
  - but also the data from the *REP MOV* running on the hyper thread



Medusa Attack – V3 Shadow *REP MOV*

- A *REP MOV* that fault on the load leaks:
  - the data from the legitimate store address
  - but also the data from the *REP MOV* running on the hyper thread



OpenSSL RSA Key Recovery

- OpenSSL Base64 Decoder uses inline Memcpy(-oS)
- Triggerred during the RSA Key Decoding from the PEM format:

```
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDmTvQjjtGtnIqMwmmaLW+YjbYTsNR8PGKXr78iYwrMV5Ye4VGy
BwS6qLD4s/EzCzGIDwkWCVx+gVHvh2wGW15Ddof0gVAtAMkR6gRABy4TkK+6YFSK
AyjmHvKCfFHvc9loeFGDyjmwFFkfdwzppXnH1Wwt0OlnyCU1GbQ1w7AHuwIDAQAB
AoGBAMyDri7pQ29NBIfMmGQuFtw8c0R3EamlIdQbX7qUguFEoe2YHqjdrKho5oZj
nDu8o+Zzm5jzBSzdf7oZ4qaeekv0fO+ZSz6CKYLbuzG2IXUB8nHJ7NuH3lacfivD
V4Cfg0yFnTK+MDG/xTVqywrCTsslkTCYC/XZOXU5Xt5z32FZAkEA/nLWQhMC4YPM
0LqMtgKzfgQdJ7vbr43WVVNpC/dN/ibUASI/3YwY0uUtqSjilIghIY7pRohrPJ6W
ntSJw0UAhQJBAOe2b9cfiOTFKXxyU4j315VkulFfTyL6GwXi/7mvpcDCixDLNRyk
uRigmdKjtIUrAX0pwjgXa6niqJ691jExez8CQQCcMZZAvTbZhHSn9LwHxqS0SIY1
K+ZxX5ogirFDPS5NQzyE7adSsntSioh6/LQKBX6BAR9FwtxBPACtwz5F9geZAkA8
a3z0SlvG04aC1cjkgUPsx6wxxbl79F2RhmSKRbvh7JiYk3RQ+L7vJgmWPGu5AcLM
oVPsjmbbkKfJZNTyVOW/AkABepEi++ZQQW0FXJWZ3nM+2CNcXYCtTgi4bGkvnZPp
/1pAy9rjeVJYhb8acTRnt+dU+uZ74CTtfuzUTZLOIuVe
-----END RSA PRIVATE KEY-----
```

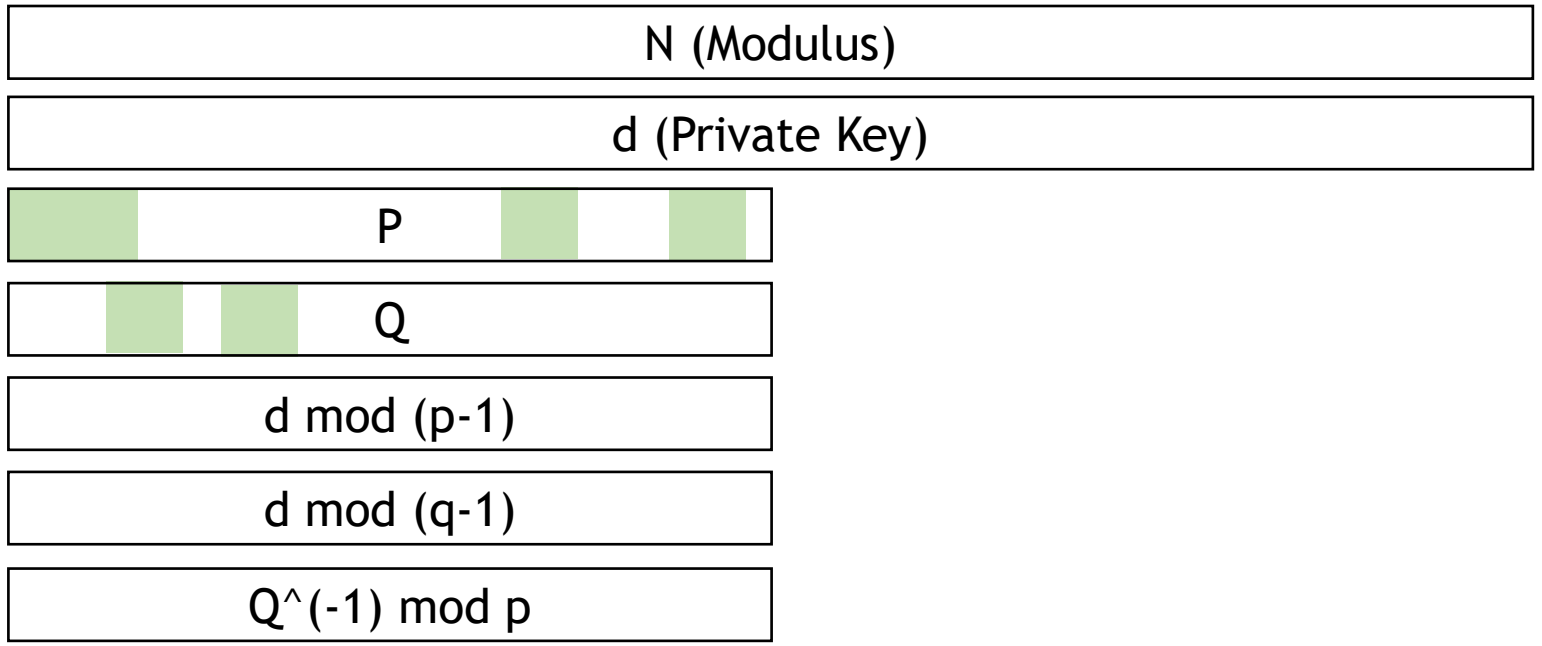
OpenSSL RSA Key Recovery

- OpenSSL Base64 Decoder uses inline Memcpy(-oS)
- Triggerred during the RSA Key Decoding from the PEM format:

```
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDmTvQjjtGtnIqMwmmaLW+YjbYTsNR8PGKXr78iYwrMV5Ye4VGy
BwS6qLD4s/EzCzGIDwkWCVx+gVHvh2wGW15Ddof0gVAtAMkR6gRABy4TkK+6YFSK
AyjmHvKCfFHvc9loeFGDyjmwFFkfdwzppXnH1Wwt0OlnyCU1GbQ1w7AHuwIDAQAB
AoGBAMyDri7pQ29NBIfMmGQuFtw8c0R3EamlIdQbX7qUguFEoe2YHqjdrKho5oZj
nDu8o+Zzm5jzBSzdf7oZ4qaeekv0fO+ZSz6CKYLbuzG2IXUB8nHJ7NuH3lacfivD
V4Cfg0yFnTK+MDG/xTVqywrCTsslkTCYC/XZOXU5Xt5z32FZAkEA/nLWQhMC4YPM
0LqMtgKzfgQdJ7vbr43WVVNpC/dN/ibUASI/3YwY0uUtqSjilIghIY7pRohrPJ6W
ntSJw0UAhQJBAOe2b9cfiOTFKXxyU4j315VkulFfTyL6GwXi/7mvpcDCixDLNRyk
uRigmdKjtIUrAX0pwjgXa6niqJ691jExez8CQQCcMZZAvTbZhHSn9LwHxqS0SIY1
K+ZxX5ogirFDPS5NQzyE7adSsntSioh6/LQKBX6BAR9FwtxBPACtwz5F9geZAkA8
a3z0SlvG04aC1cjkgUPsx6wxxbl79F2RhmSKRbvh7JiYk3RQ+L7vJgmWPGu5AcLM
oVPsjmbbkKfJZNTyVOW/AkABepEi++ZQQW0FXJWZ3nM+2CNcXYCtTgi4bGkvnZPp
/1pAy9rjeVJYhb8acTRnt+dU+uZ74CTtfuzUTZLOIuVe
-----END RSA PRIVATE KEY-----
```

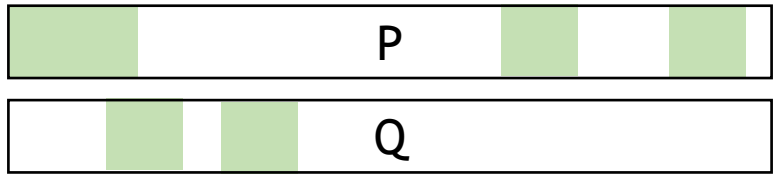
OpenSSL RSA Key Recovery

- OpenSSL Base64 Decoder uses inline Memcpy(-oS)
- Triggerred during the RSA Key Decoding from the PEM format:



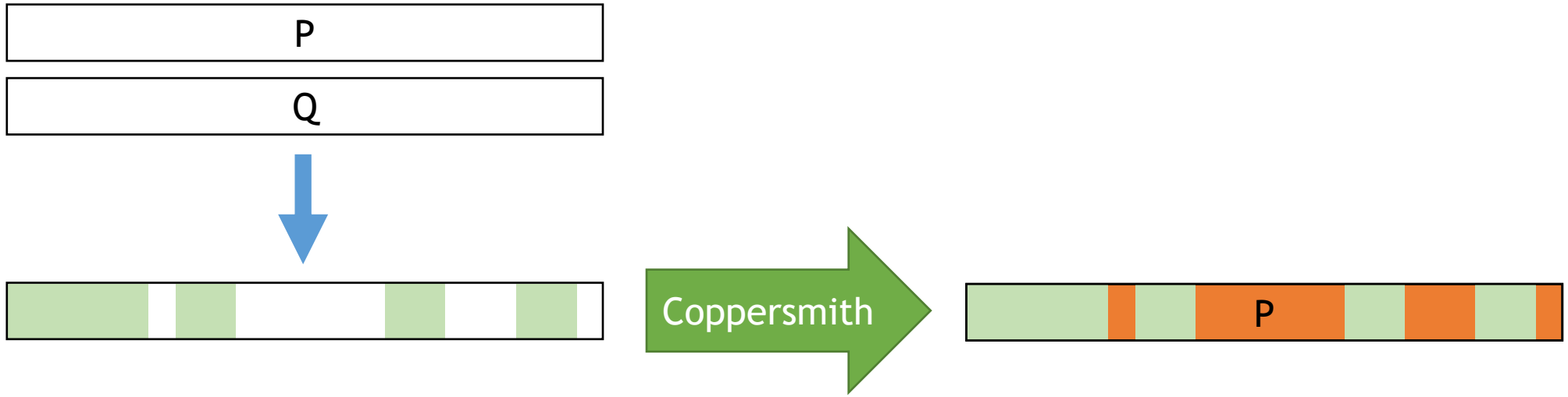
OpenSSL RSA Key Recovery - Coppersmith

- Knowledge of at least $^1/_3$ of P+Q
- Create a n dimensional hidden number problem where n is relative to the number of recovered chunks
- Feed it to the lattice-based algorithm to find the short vector



OpenSSL RSA Key Recovery – Coppersmith Attack

- Knowledge of at least $^1/_3$ of P+Q.
- Creating a n dimensional hidden number problem where n is relative to the number of recovered chunks.
- Feeding it to the lattice-based algorithm to find the short vector.



Conclusion

- Automated Testing for CPU Attacks,
  - helps us to understand the root cause of these issues better.
  - can be used to verify hardware mitigations (e.g., Fallout on ICL).
  - can help us to improve the leakage rate and understand the impact of attacks better.
- The impact of attacks depend also on the exploitation technique.
- Potentials and Future work:
  - Can we integrate such tools with feedback from hardware/simulator?

Questions?!

Daniel Moghimi
@danielmgmi

- Moghimi, D., Lipp, M., Sunar, B., & Schwarz, M. (2020). Medusa: Microarchitectural Data Leakage via Automated Attack Synthesis. In 29th USENIX Security Symposium (USENIX Security 20).

CPU Memory Subsystem