Microarchitectural Cryptanalysis and Trusted Environments

Daniel Moghimi
Worcester Polytechnic University

Committee Members:
- Prof. Donald R. Brown (Department Head)
- Prof. Thomas Eisenbarth
- Prof. Simha Sethumadhavan
- Prof. Berk Sunar

September 4, 2020
PhD Area Exam

Motivation: Modern-day Computing and Multitenancy

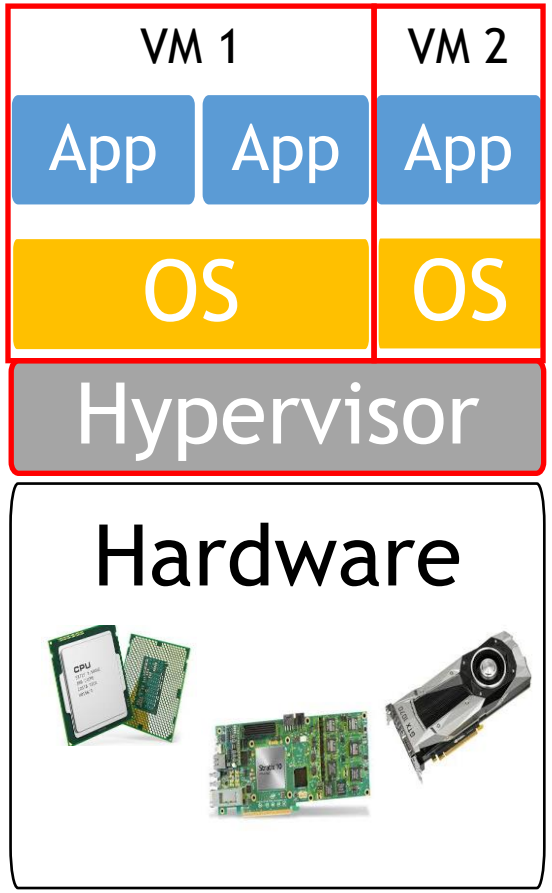
Single user, single task

30 years of evolution

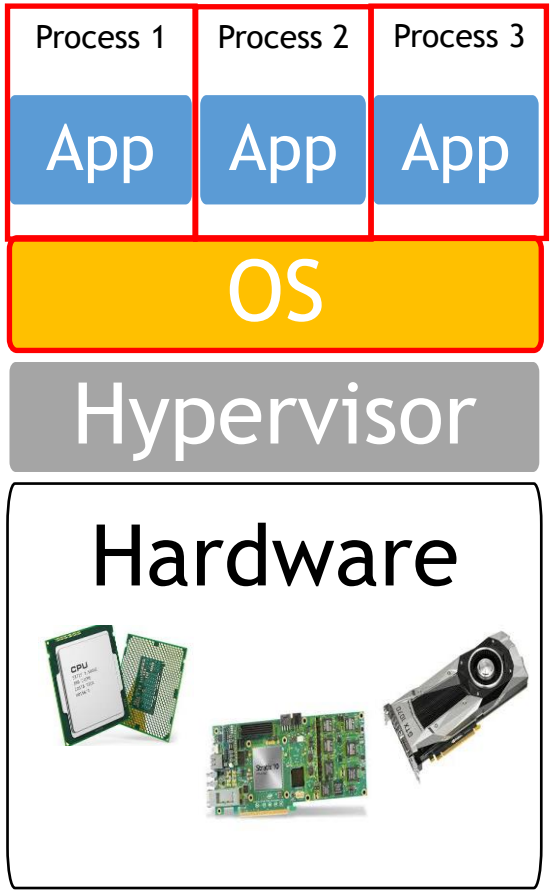
Multiuser, multitask, several security domains

Multiuser, multitask, several security domains

Motivation: Secure Isolation



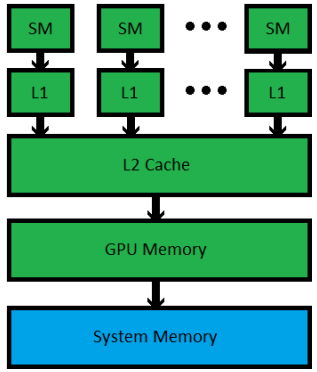
Virtual Machines



Process-Level Isolation

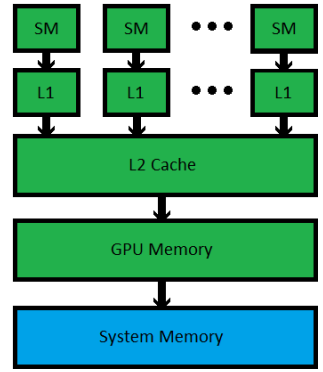
Motivation: Side-channel Attacks

- Many Different Channels:
  - Power analysis
  - EM analysis
  - *Timing analysis*
  - *CPU side channels*
    - *Software-based*
  - …



Motivation: Side-channel Attacks

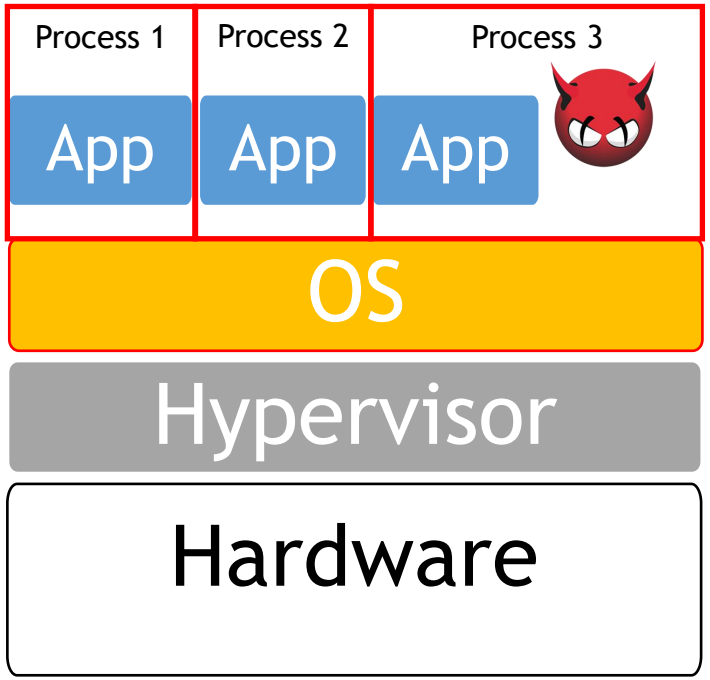
- Many Different Channels:
  - Power analysis
  - EM analysis
  - ***Timing analysis***
  - ***CPU side channels***
    - ***Software-based***
  - …



My work!!!

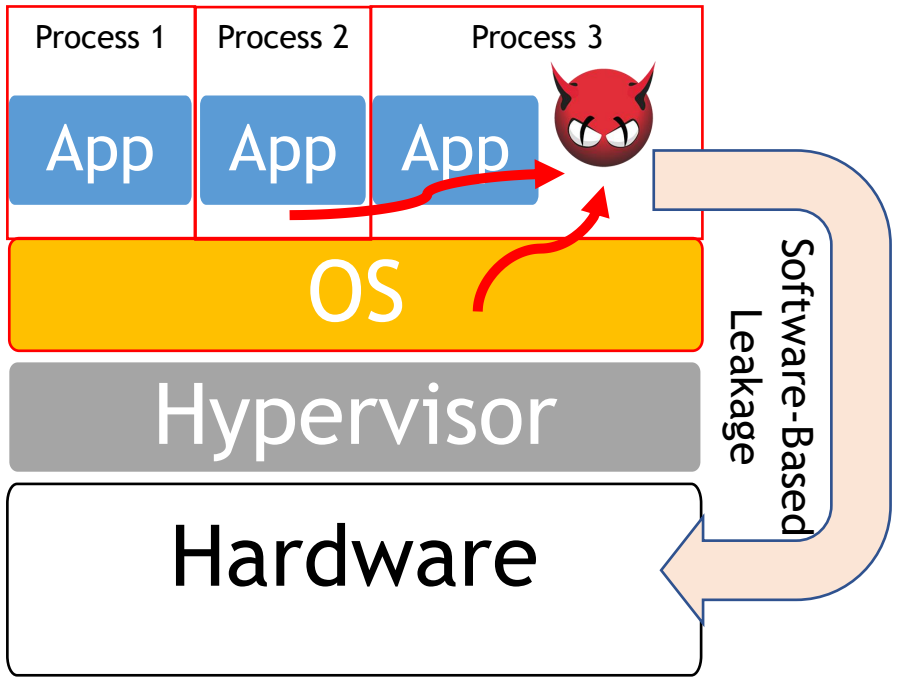
User-level Threat to Secure Isolation (T1)

- Software-based microarchitectural side channels
- A user-level adversary leaks the data or secret of other users.
  - It applies to:
    - process-level isolation
    - VM-level isolation



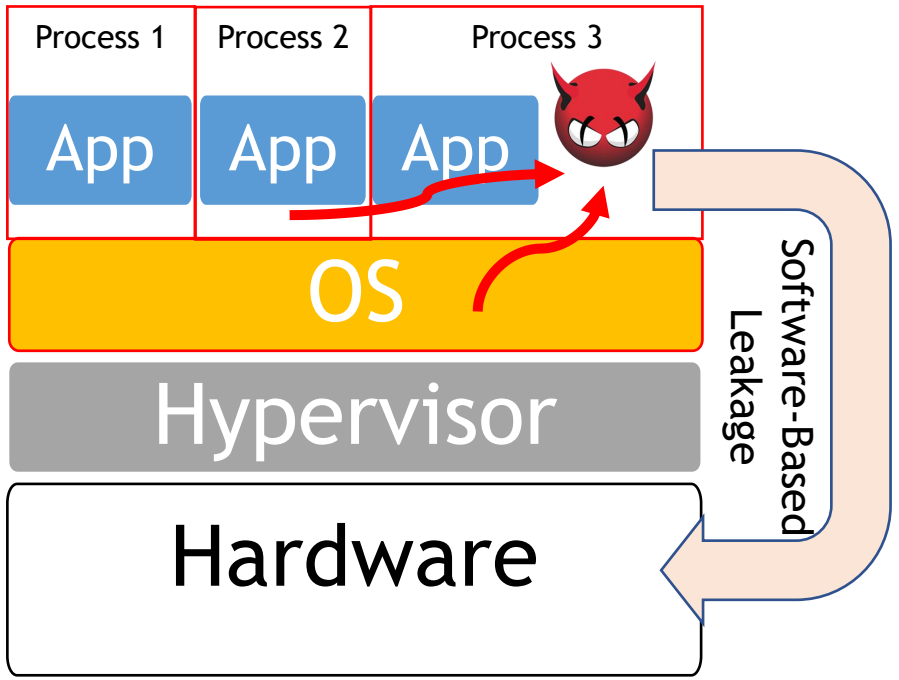
User-level Threat to Secure Isolation (T1)

- Software-based microarchitectural side channels
- A user-level adversary leaks the data or secret of other users.
  - It applies to:
    - process-level isolation
    - VM-level isolation



User-level Threat to Secure Isolation (T1)

- Software-based microarchitectural side channels
- A user-level adversary leaks the data or secret of other users.
  - It applies to:
    - process-level isolation
    - VM-level isolation



- **Security evaluation of the CPU microarchitecture**
- **Analysis of software-based side channels microarchitectural attacks**

Motivation: Is this the only threat model?!

- We can **not** trust:
  - cloud providers.
  - software developers.
  - OEMs and computer manufacturers.

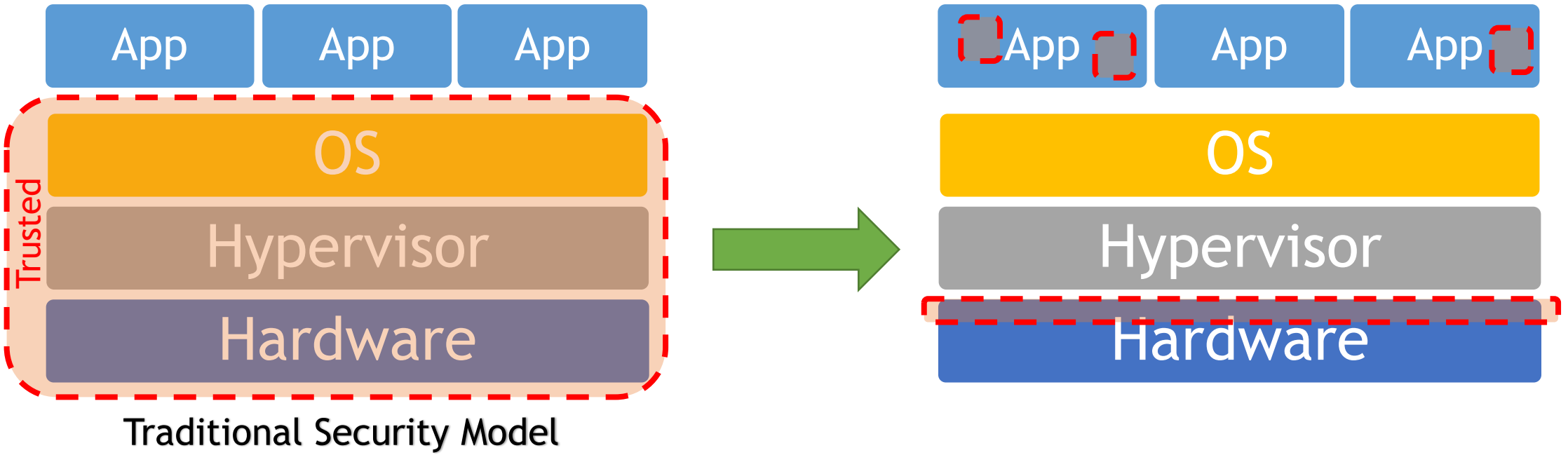
- Trusted Computing
  - Others can compute on the data without giving them the data.

- Example Applications:
  - Privacy-Preserving machine learning
  - Digital right management (DRM)
  - Anonymous blockchain transactions

Trusted Execution Environments

Trusted Execution Environment (TEE) - Intel SGX

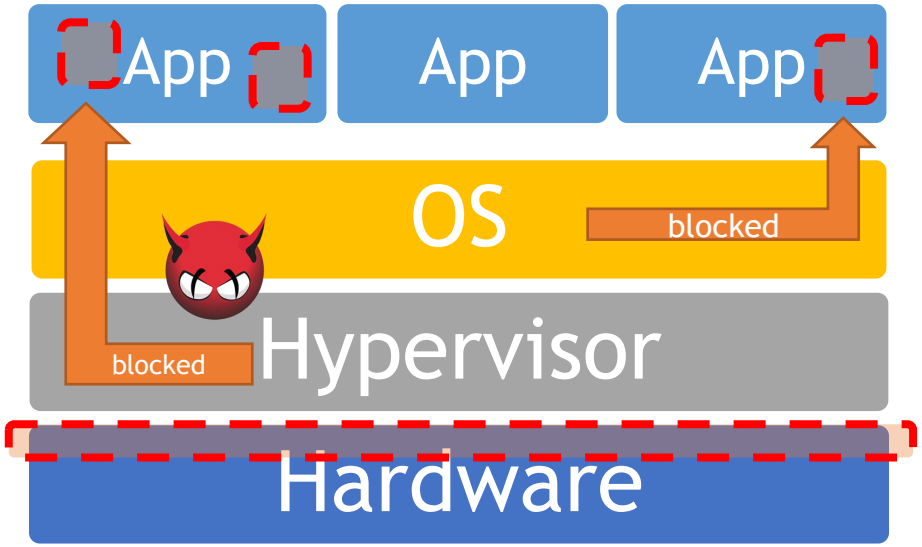
- Intel Software Guard eXtensions (SGX)



Traditional Security Model

System-level Threat to Trusted Execution Environments (T2)

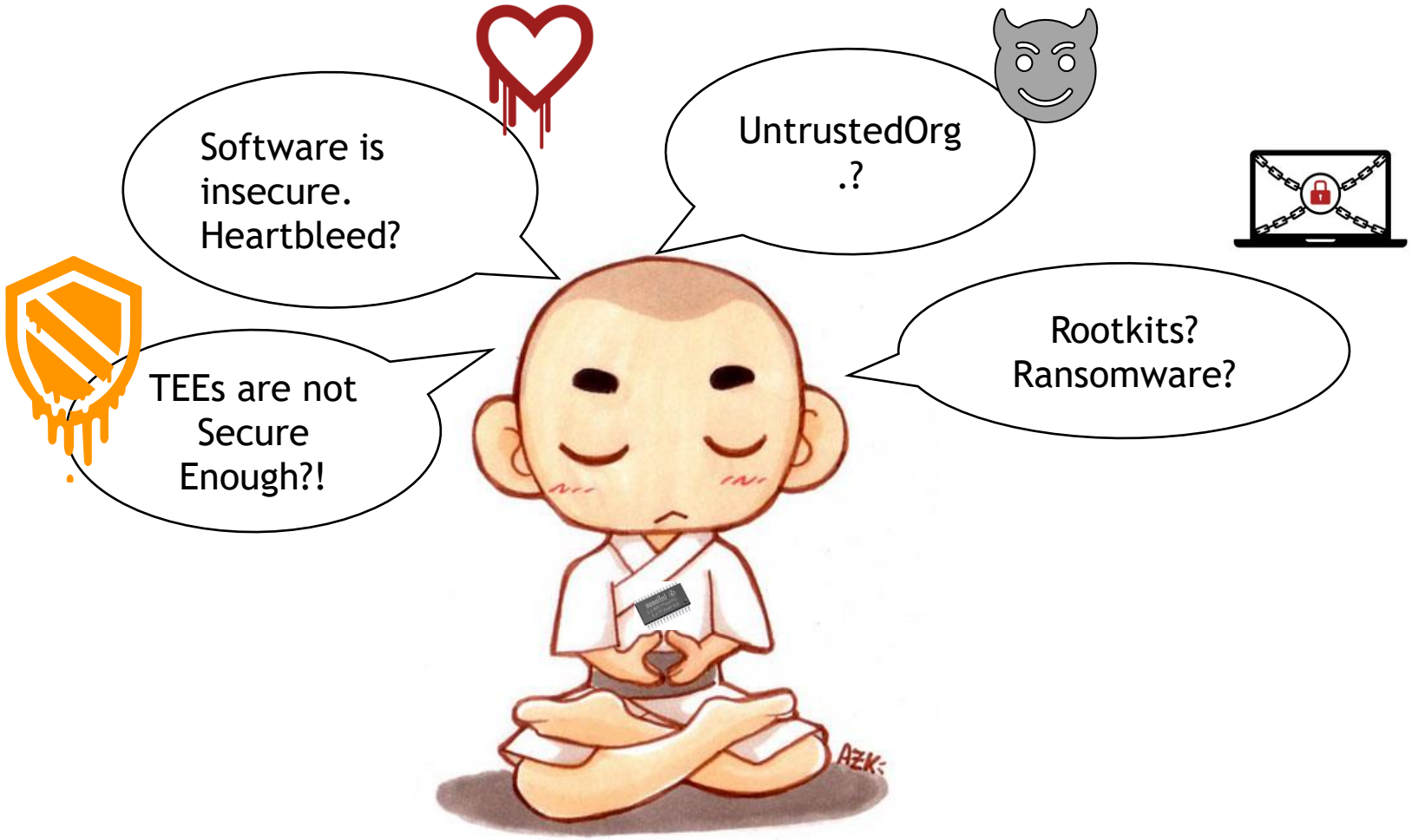
- Intel Software Guard eXtensions (SGX)
- **Enclave:** A hardware protected user-level software module
  - Mapped by the operating system
  - Loaded by the user program
  - Authenticated and encrypted by CPU
- It **must** protect secrets against system-level adversary



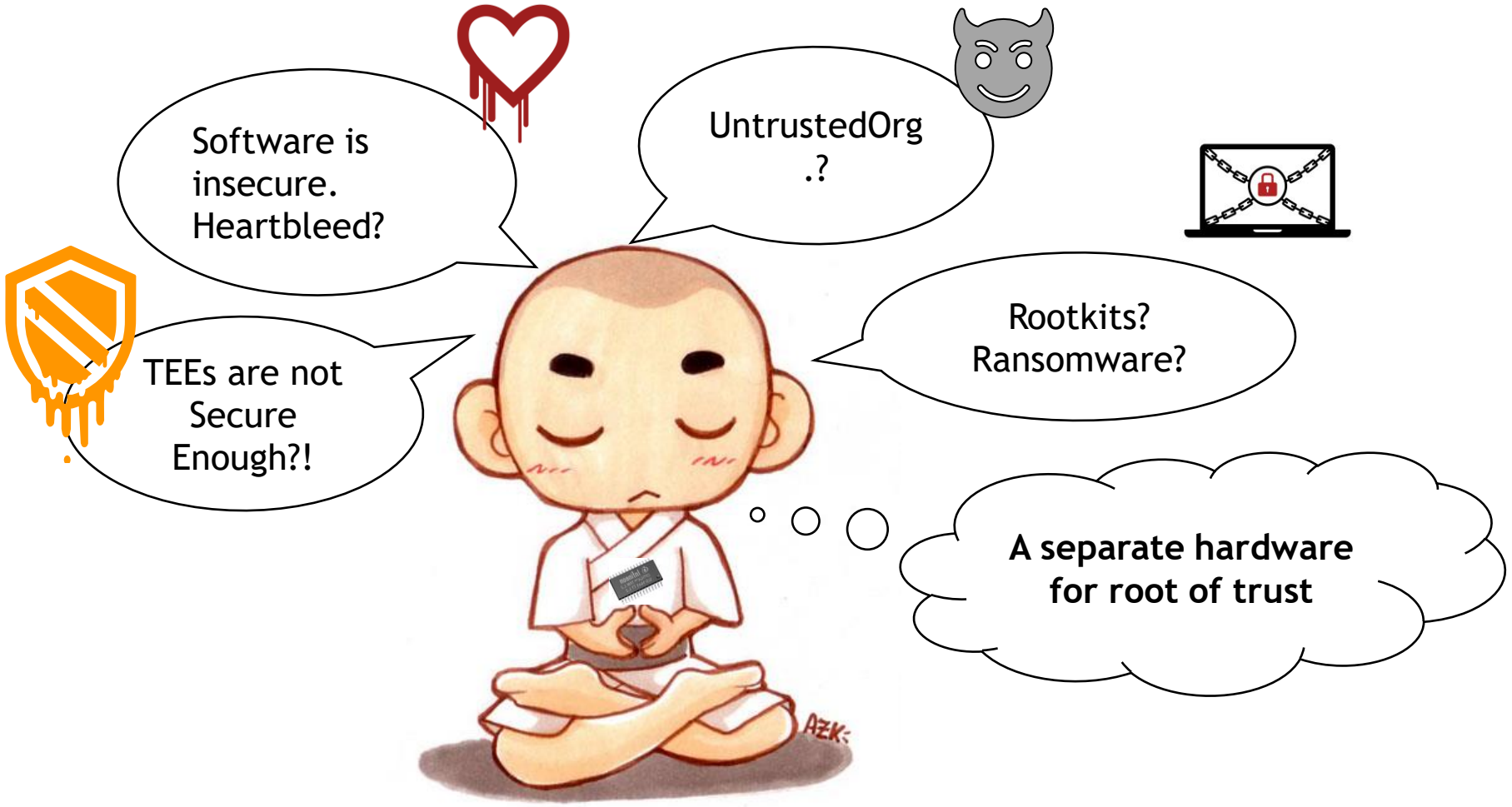
New Attacker Model:

Attacker gets full control over the OS

Beyond TEEs - Trusted Platform Module

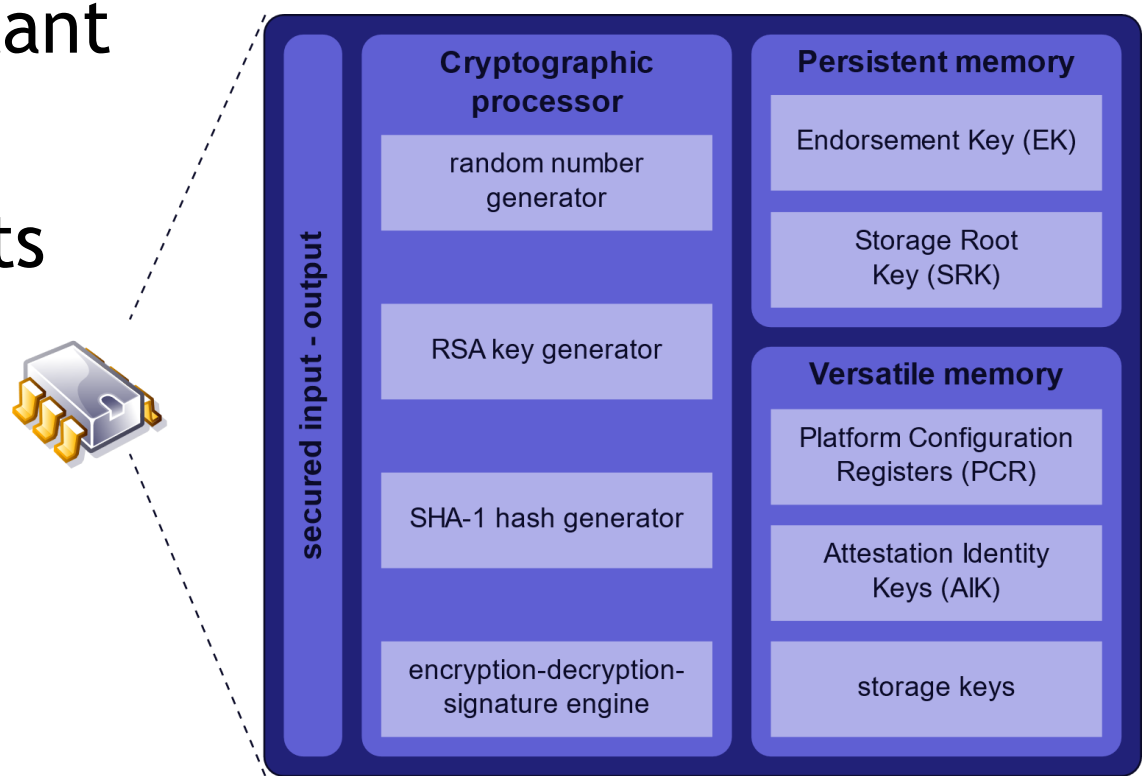


Beyond TEEs - Trusted Platform Module



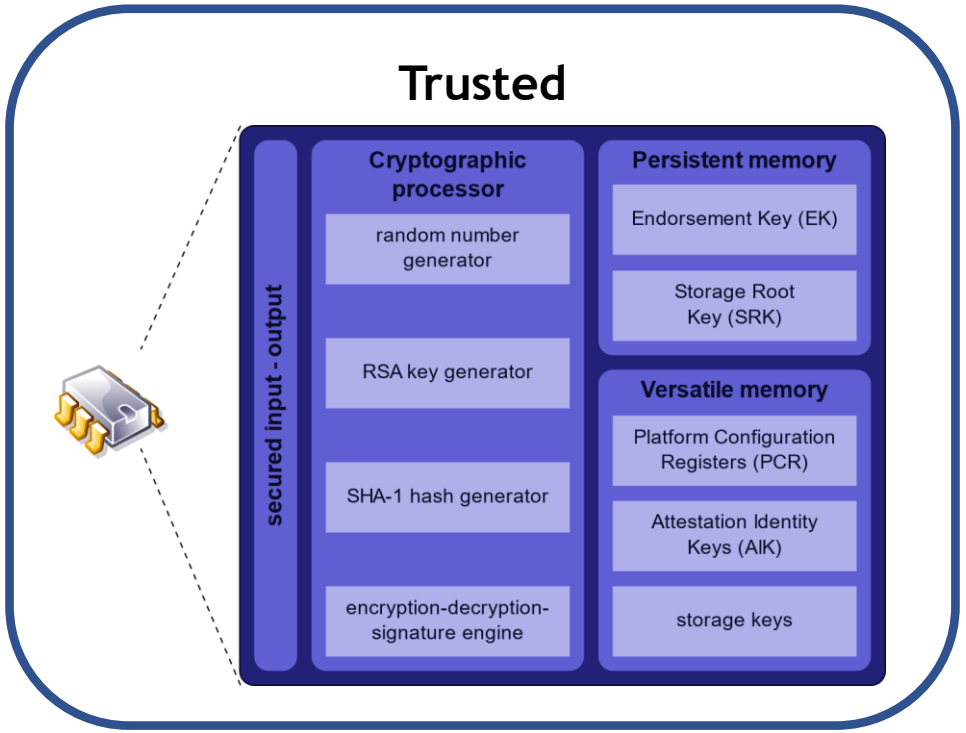
Trusted Platform Module (TPM)

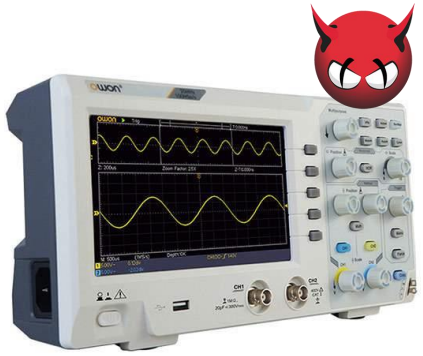
- Security chip for computers?
- Tamper and Side-Channel Resistant
- Cryptographic Co-processor
- Standardized by **TCG**, it supports
  - hash functions
  - encryption
  - **digital signatures**
  - …



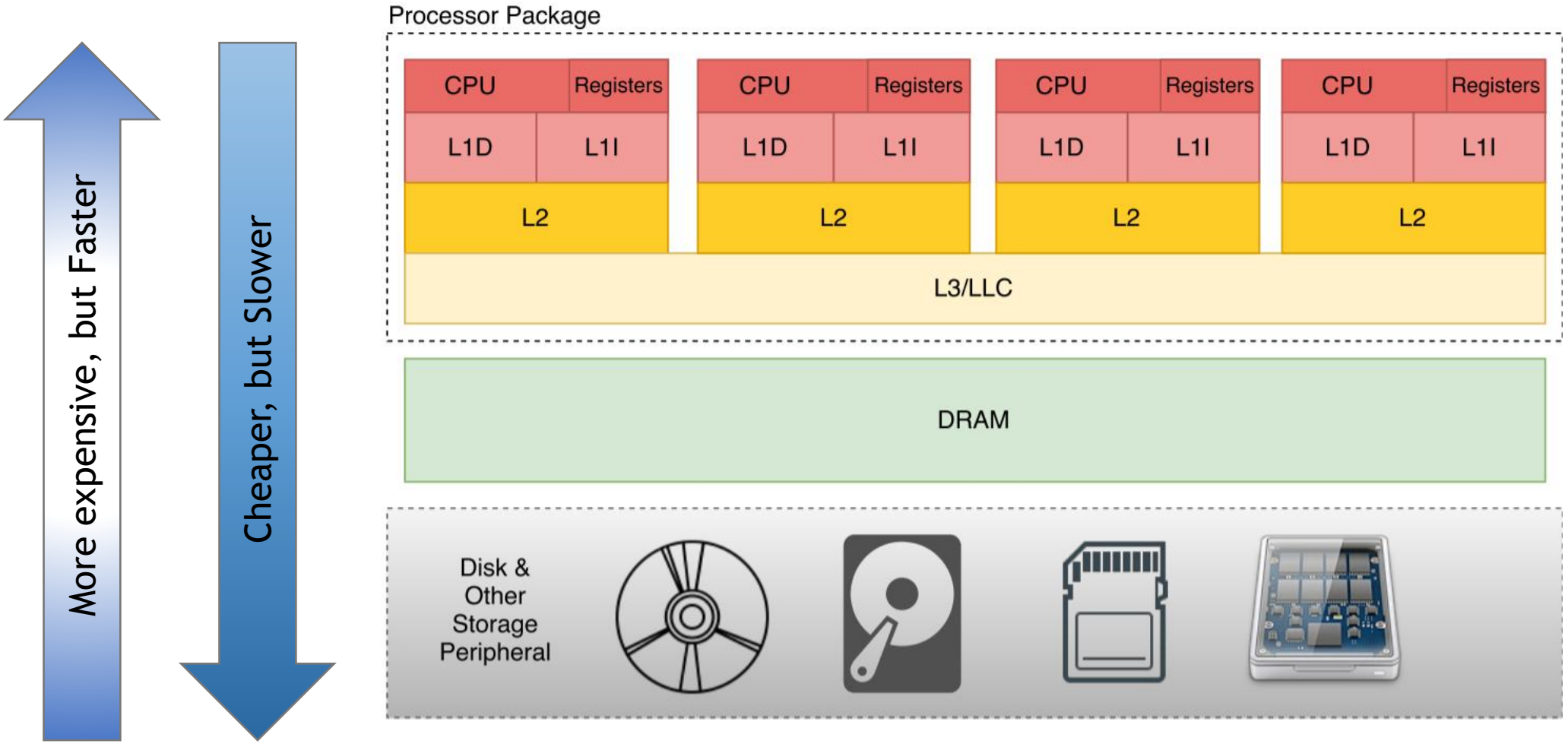
Remote and ~~Physical~~ Threats to TPM (T3)

- Our work focuses on Timing Attack





Background: Memory Hierarchy

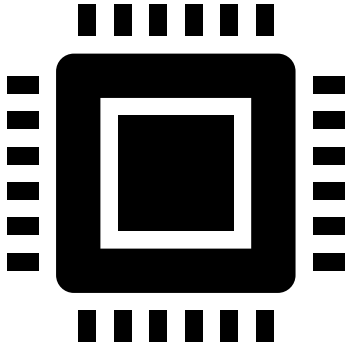


Background: Memory Hierarchy - Cache Miss

```
int x = buffer[0];
int y = buffer[0];
```

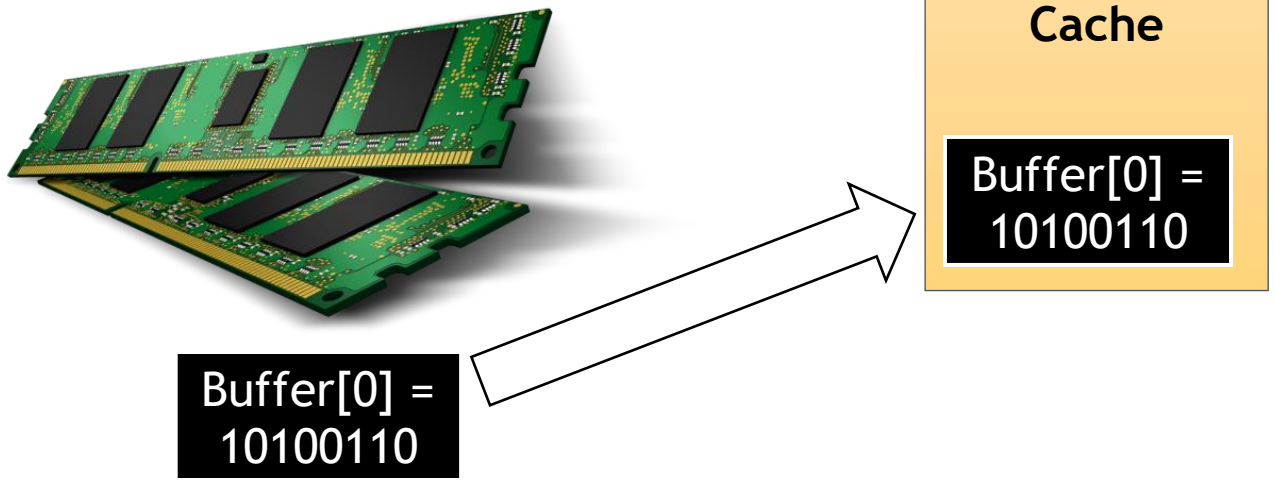
Buffer[0] = 10100110

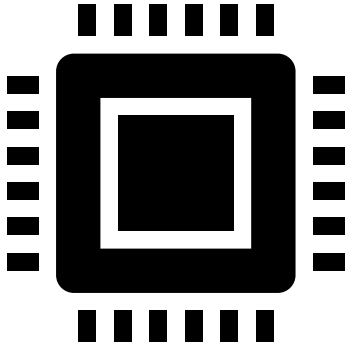
Cache



Background: Memory Hierarchy - Cache Miss

```
int x = buffer[0];
int y = buffer[0];
```

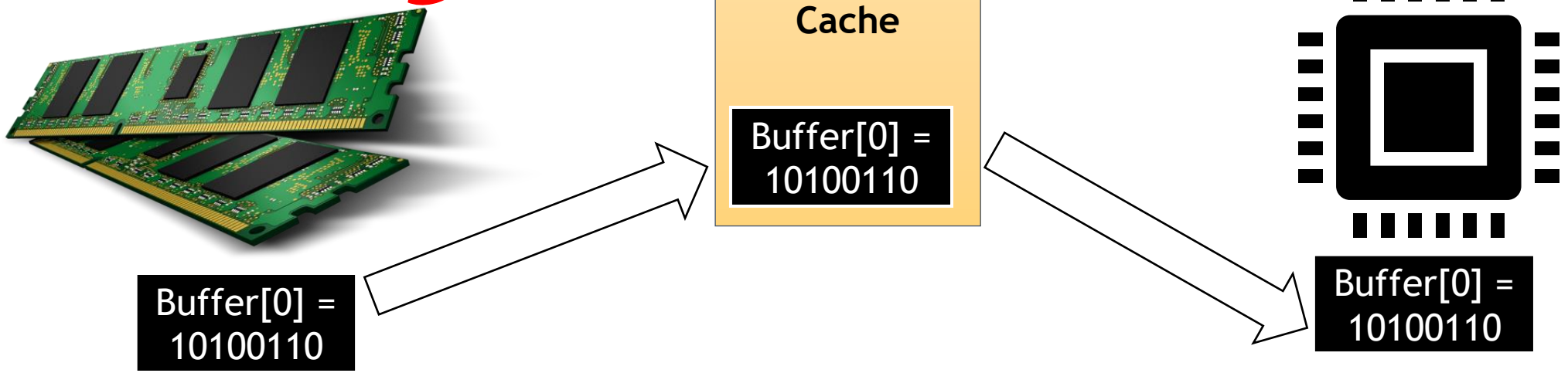




Background: Memory Hierarchy - Cache Miss

```
int x = buffer[0];
int y = buffer[0];
```

SLOOWWW!



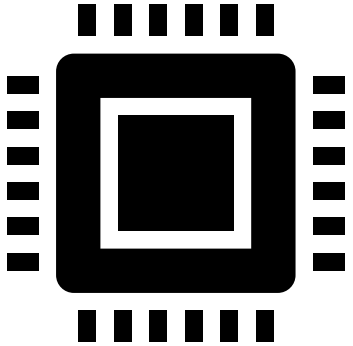
Background: Memory Hierarchy - Cache Hit

```
int x = buffer[0];
int y = buffer[0];
```

Buffer[0] = 10100110

Cache

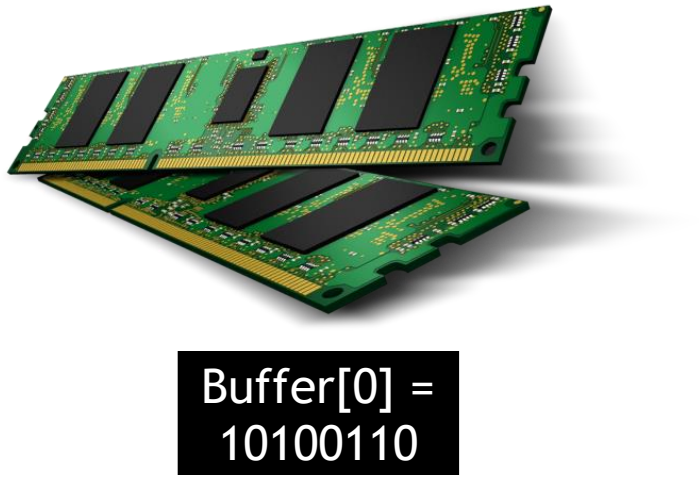
Buffer[0] = 10100110

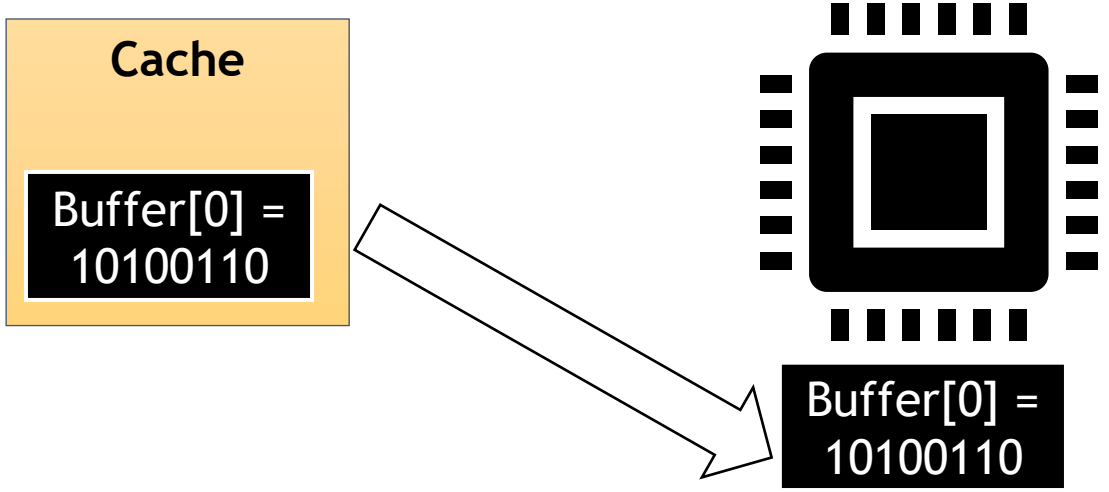


Background: Memory Hierarchy - Cache Hit

```
int x = buffer[0];
int y = buffer[0];
```

Fast!!





Background: Cache Attack - Flush & Reload

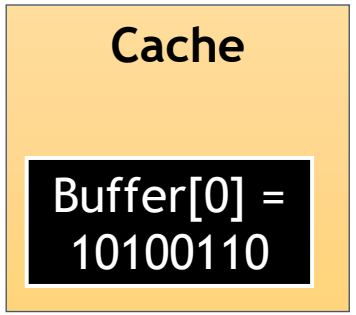
Attacker

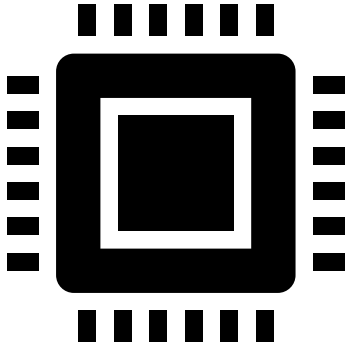
```
clflush(buffer[0]);
waste_some_cycle();
t1 = time();
int y = buffer[0];
t2 = time();
```

Victim

```
int x = buffer[0];
```

Buffer[0] = 10100110





Background: Cache Attack - Flush & Reload

Attacker

```
clflush(buffer[0]);
waste_some_cycle();
t1 = time();
int y = buffer[0];
t2 = time();
```

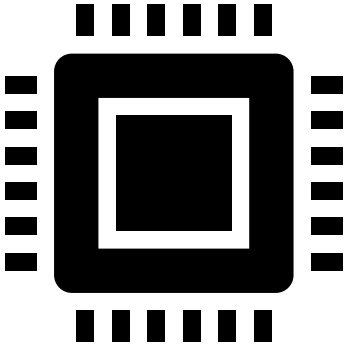
Victim

```
int x = buffer[0];
```

Buffer[0] = 10100110

Cache

Buffer[0] = 10100110



Background: Cache Attack - Flush & Reload

Attacker

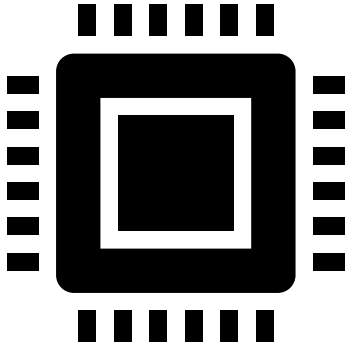
```
clflush(buffer[0]);
waste_some_cycle();
t1 = time();
int y = buffer[0];
t2 = time();
```

Victim

```
int x = buffer[0];
```

Buffer[0] = 10100110

Cache



Background: Cache Attack - Flush & Reload

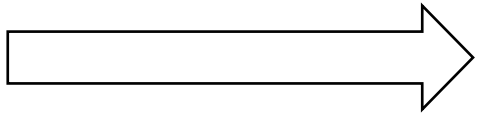
Attacker

```
clflush(buffer[0]);
waste_some_cycle();
t1 = time();
int y = buffer[0];
t2 = time();
```

Victim

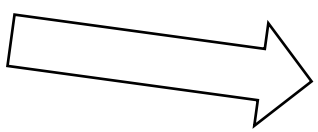
```
int x = buffer[0];
```

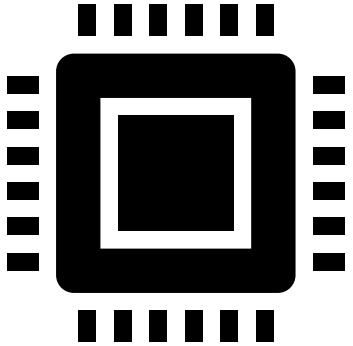
Buffer[0] = 10100110



Cache

Buffer[0] = 10100110





Buffer[0] = 10100110

Background: Cache Attack - Flush & Reload

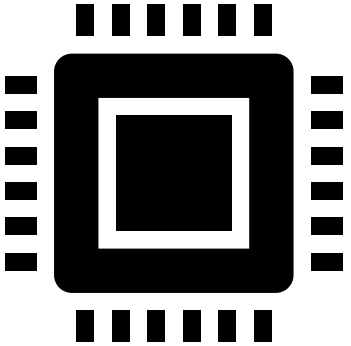
Attacker

```
clflush(buffer[0]);
waste_some_cycle();
t1 = time();
int y = buffer[0];
t2 = time();
```

Victim

```
int x = buffer[0];
```

Buffer[0] = 10100110



Cache

Buffer[0] = 10100110

Background: Cache Attack - Flush & Reload

Attacker

```
clflush(buffer[0]);
waste_some_cycle();
t1 = time();
int y = buffer[0];
t2 = time();
```

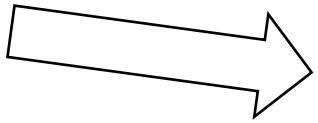
Victim

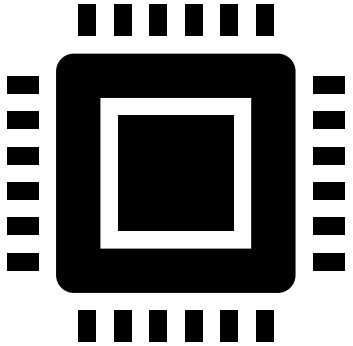
```
int x = buffer[0];
```

Buffer[0] = 10100110

Cache

Buffer[0] = 10100110





Buffer[0] = 10100110

Background: Cache Attack - Flush & Reload

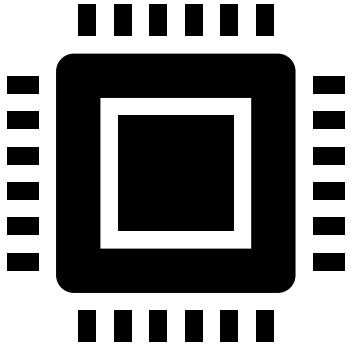
Attacker

```
clflush(buffer[0]);
waste_some_cycle();
t1 = time();
int y = buffer[0];
t2 = time();
```

Victim

```
int x = buffer[0];
```

Buffer[0] = 10100110



Cache

Buffer[0] = 10100110

Background: Cache Attack - Flush & Reload

Attacker

```
clflush(buffer[0]);
waste_some_cycle();
t1 = time();
int y = buffer[0];
t2 = time();
```

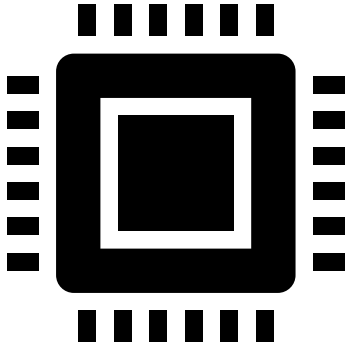
t2 - t1> threshold = cache miss

t2 - t1 < threshold = cache hit

Buffer[0] = 10100110

Cache

Buffer[0] = 10100110



Background: Cache Attacks

- There are many different type of cache attacks:
  - Flush+Reload (Flush+Flush)
  - Prime+Probe
  - Evict+Reload

- Cache attacks leak memory access patterns of collocated victims with 64-byte granularity.
- Secret-dependent memory accesses leak some information about the secret. Examples:
  - AES: S-Box lookups
  - RSA: Table lookups in fixed-window Montgomery exponentiation

Background: Transient Execution Attacks

- Date leakage as oppose to access pattern leakage

- Spectre
  - Due to the CPU's branch Predictor.

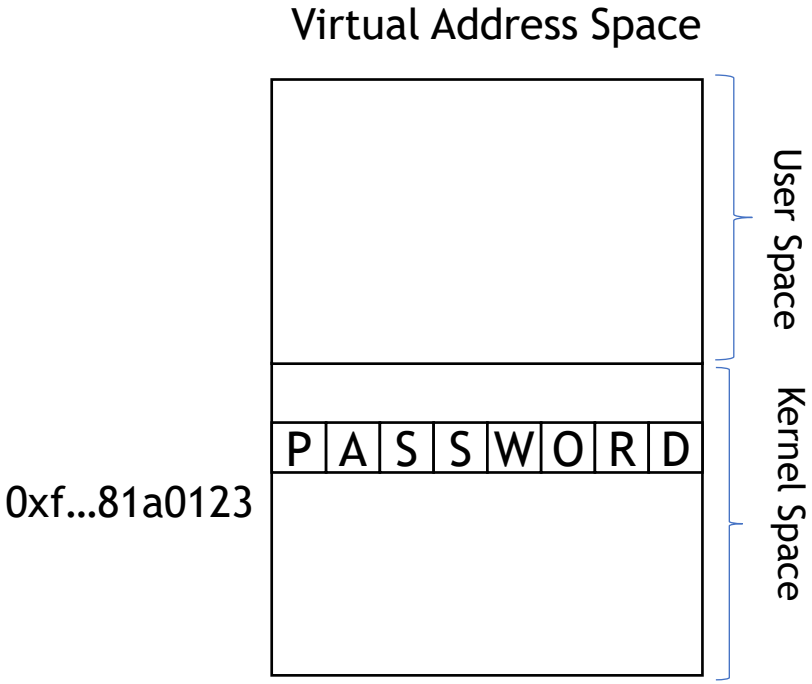
- Meltdown
  - Due the speculative behavior of the CPU's memory subsystem
  - Data leakage wo/ any assumption about the victim software

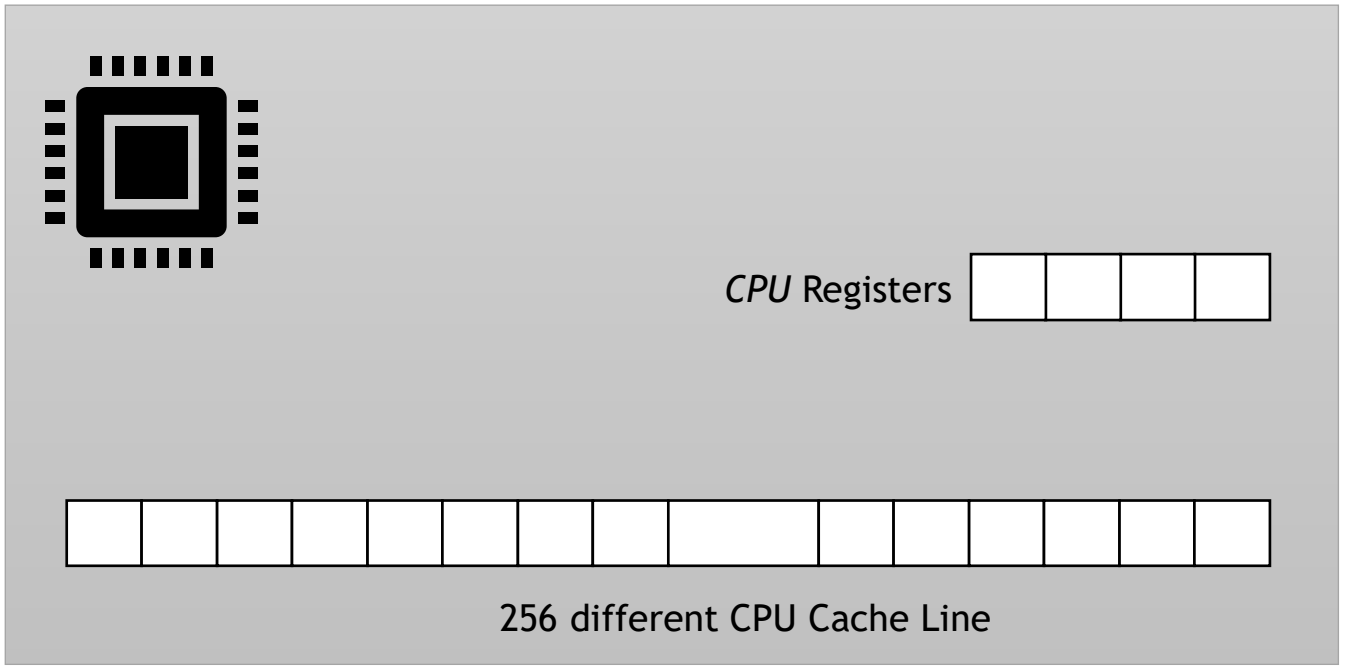
Background: Revisiting Meltdown-style Attacks

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

Background: Revisiting Meltdown-style Attacks

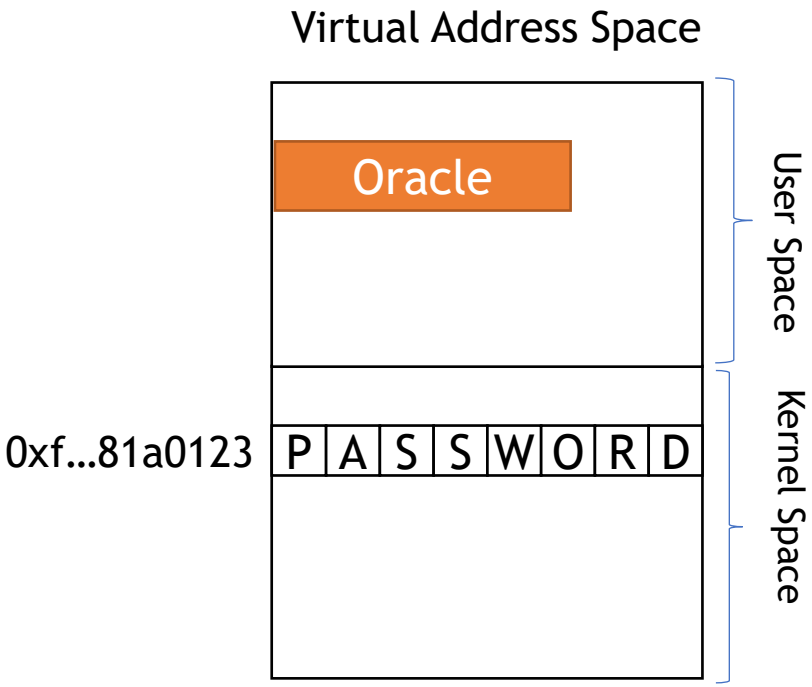
```
char secret = *(char *) 0xffffffff81a0123;
```

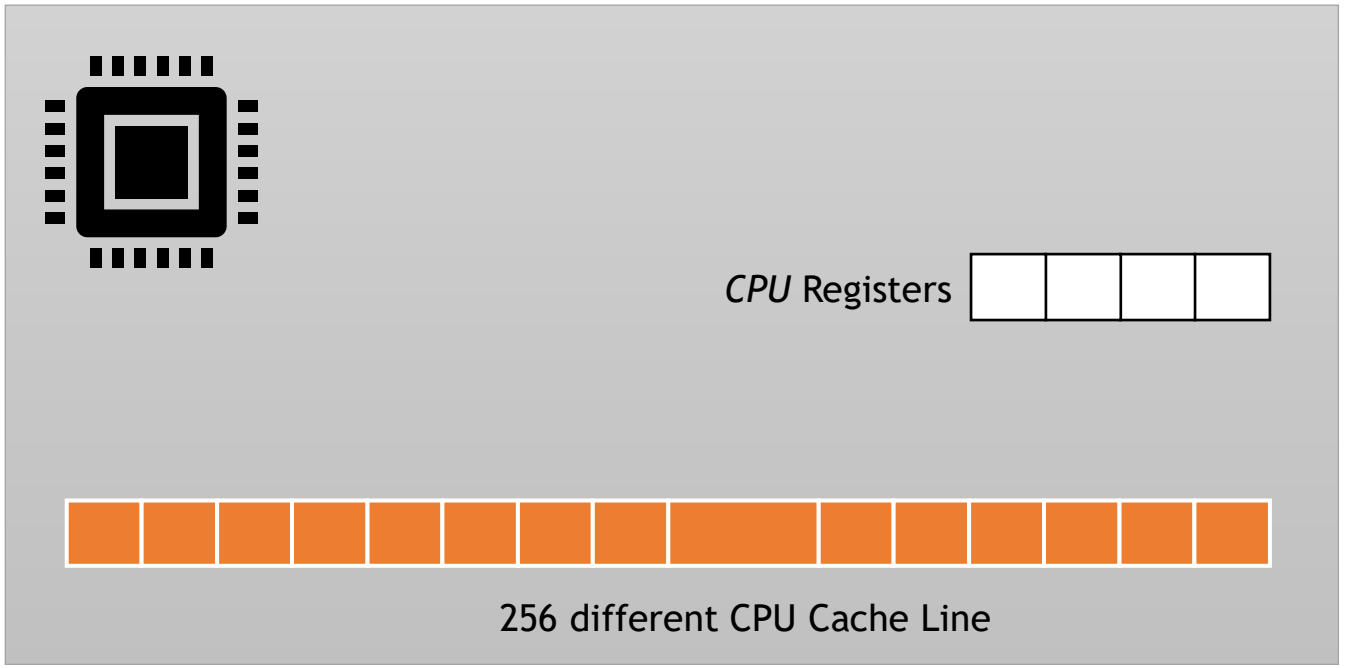




Background: Revisiting Meltdown-style Attacks

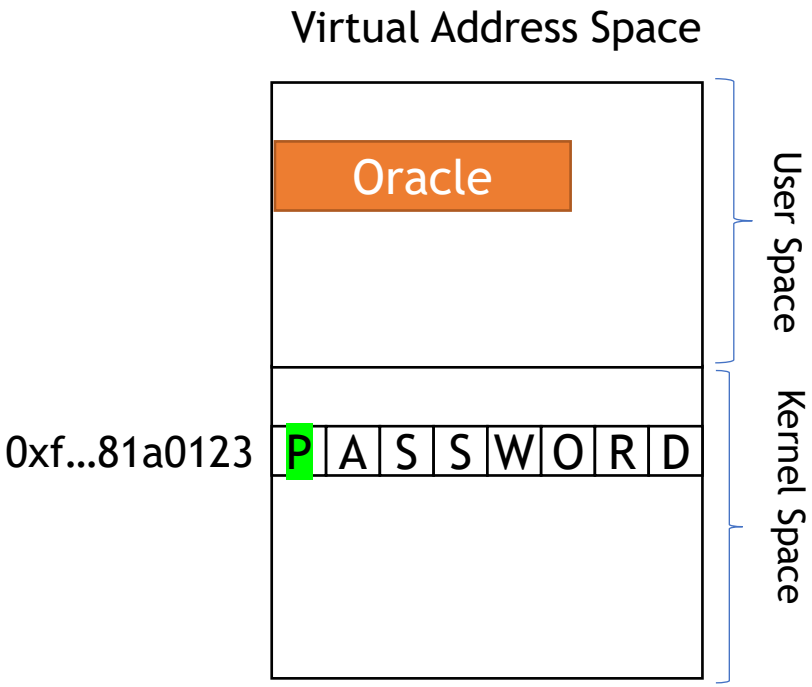
```
char secret = *(char *) 0xffffffff81a0123;
```

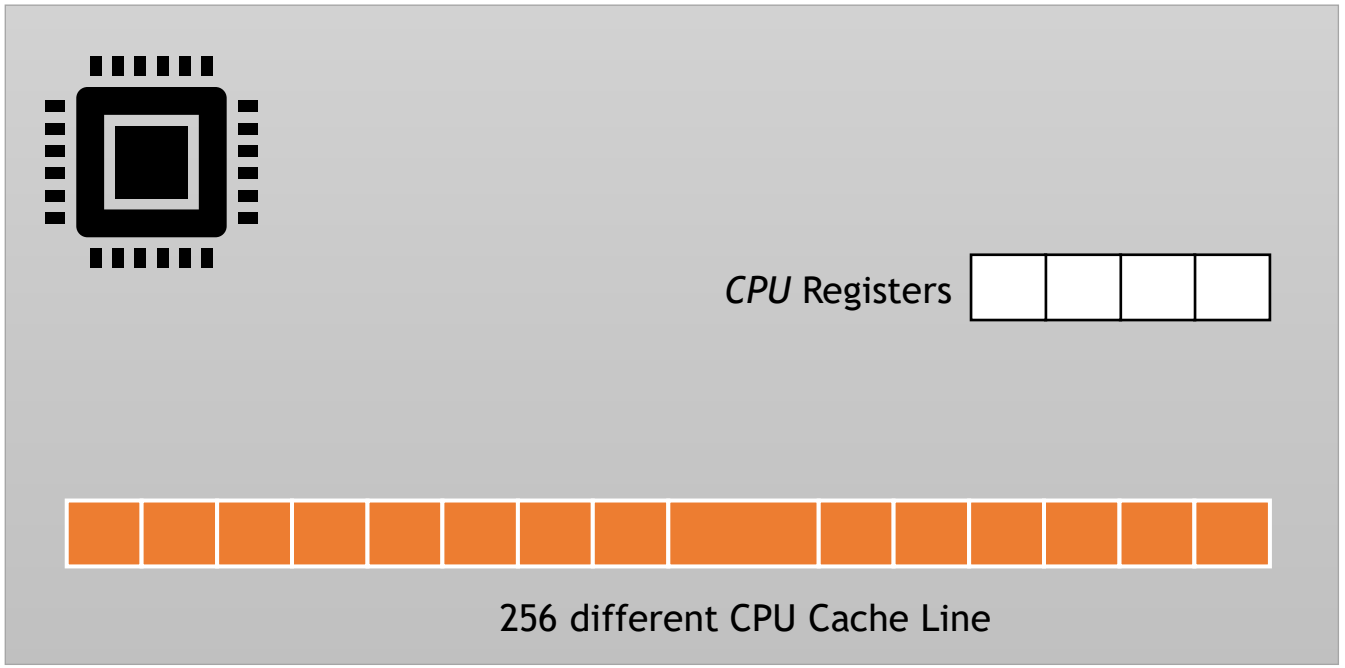




Background: Revisiting Meltdown-style Attacks (Step 1)

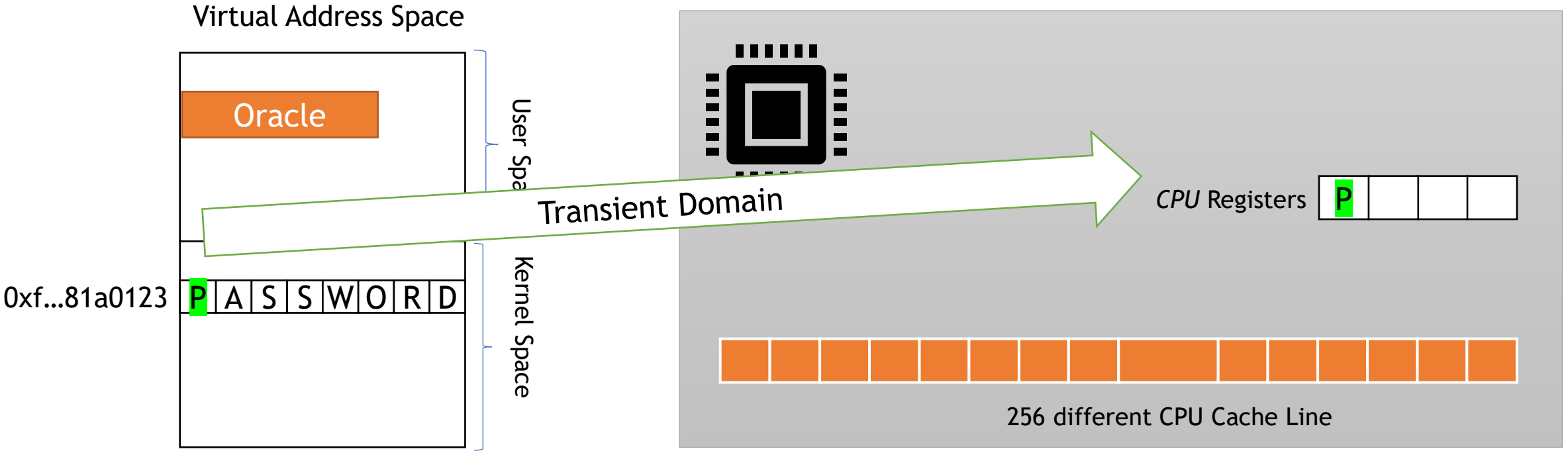
```
char secret = *(char *) 0xffffffff81a0123;
```





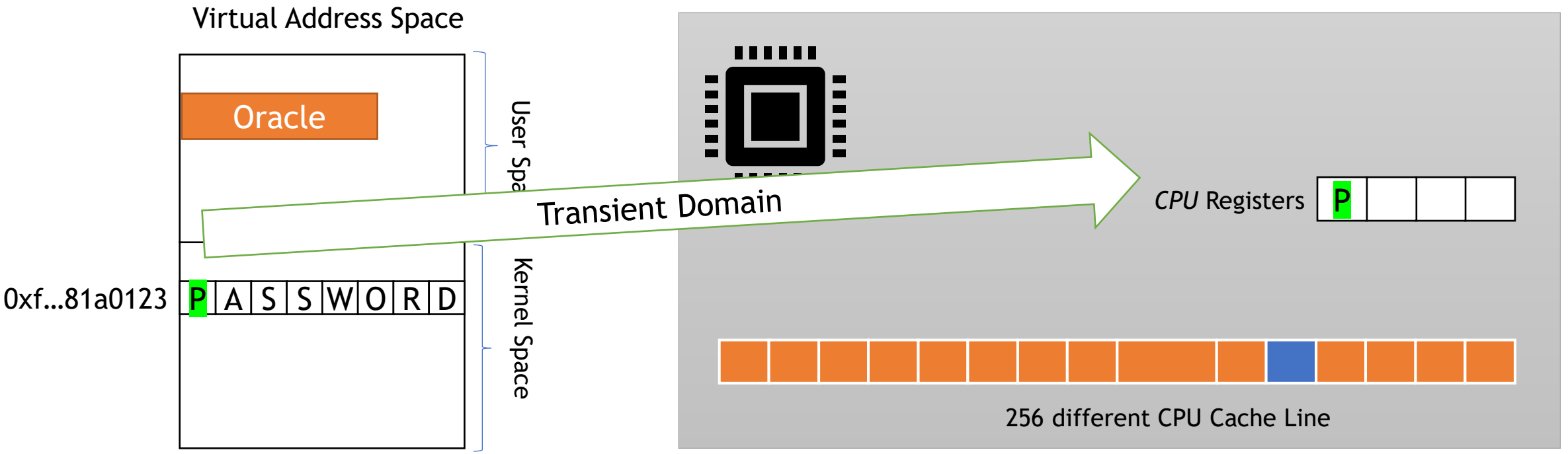
Background: Revisiting Meltdown-style Attacks (Step 1)

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



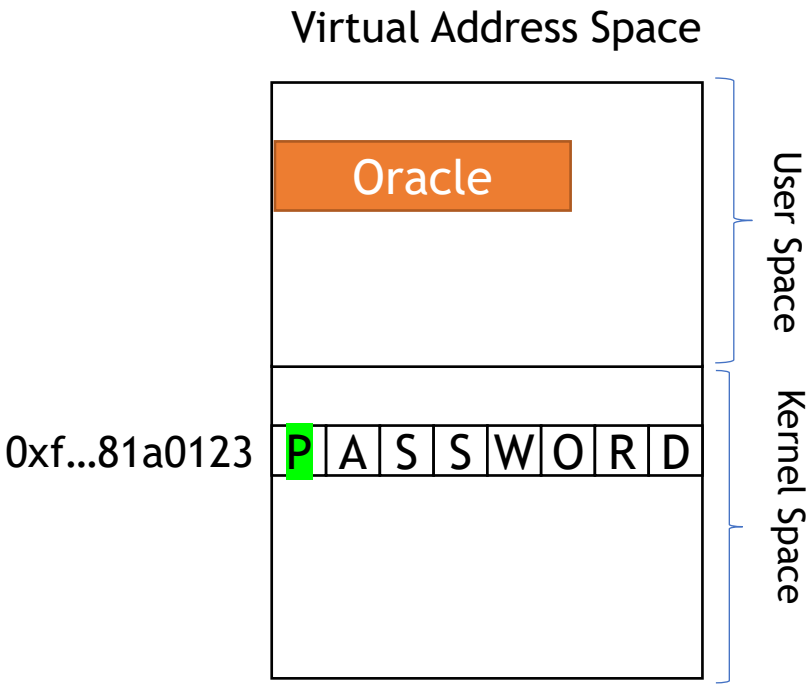
Background: Revisiting Meltdown-style Attacks (Step 2)

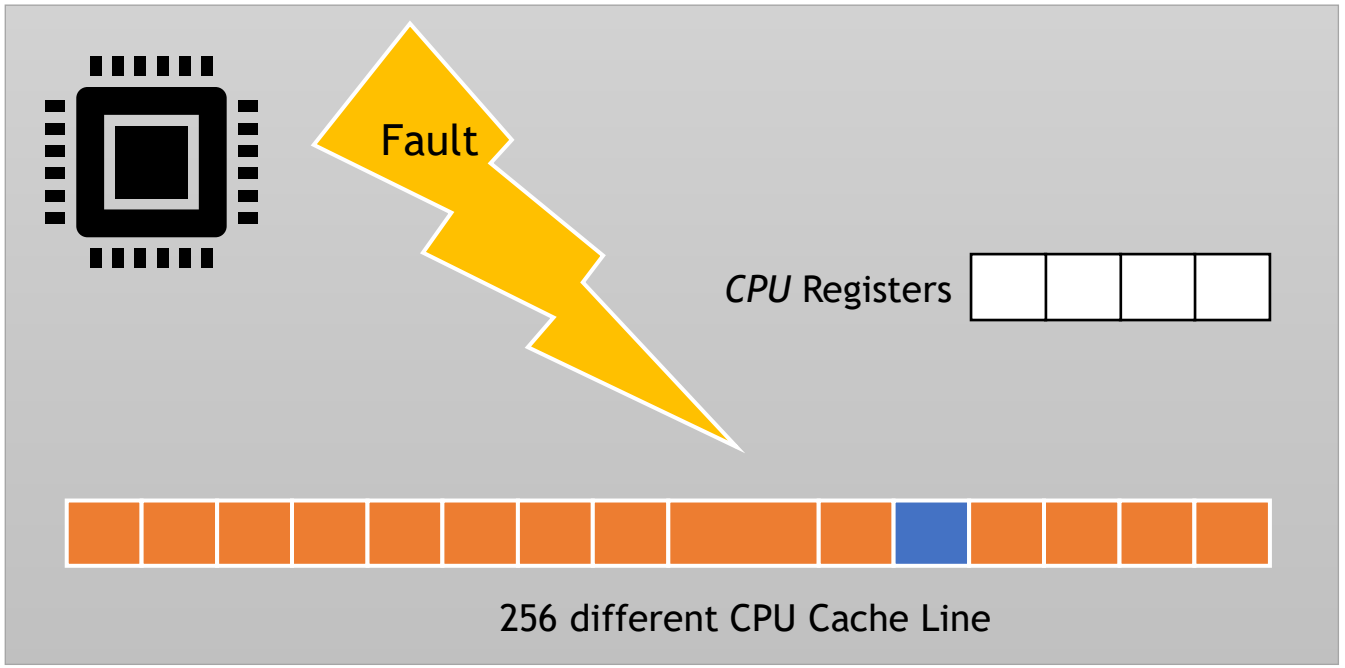
```
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```



Background: Revisiting Meltdown-style Attacks (Step 2)

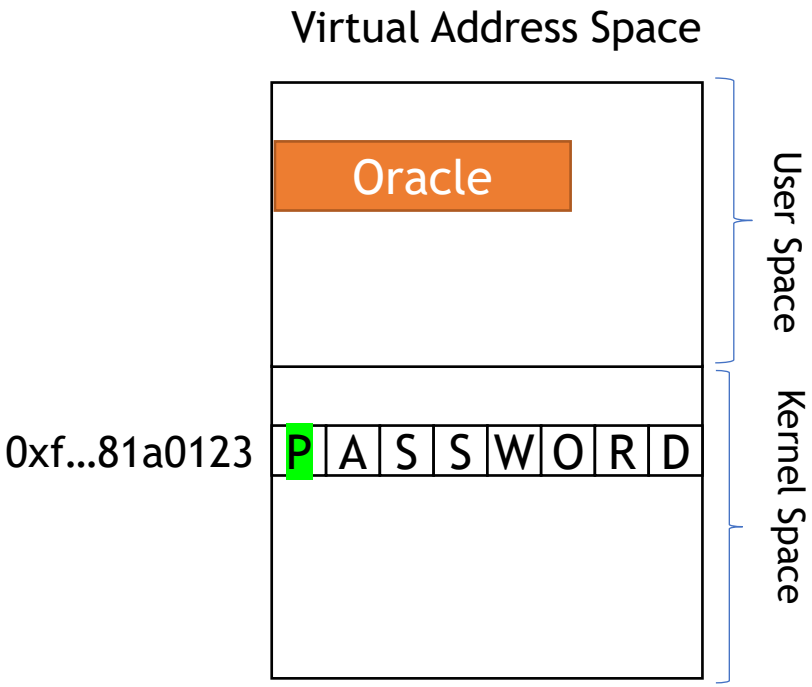
```
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```

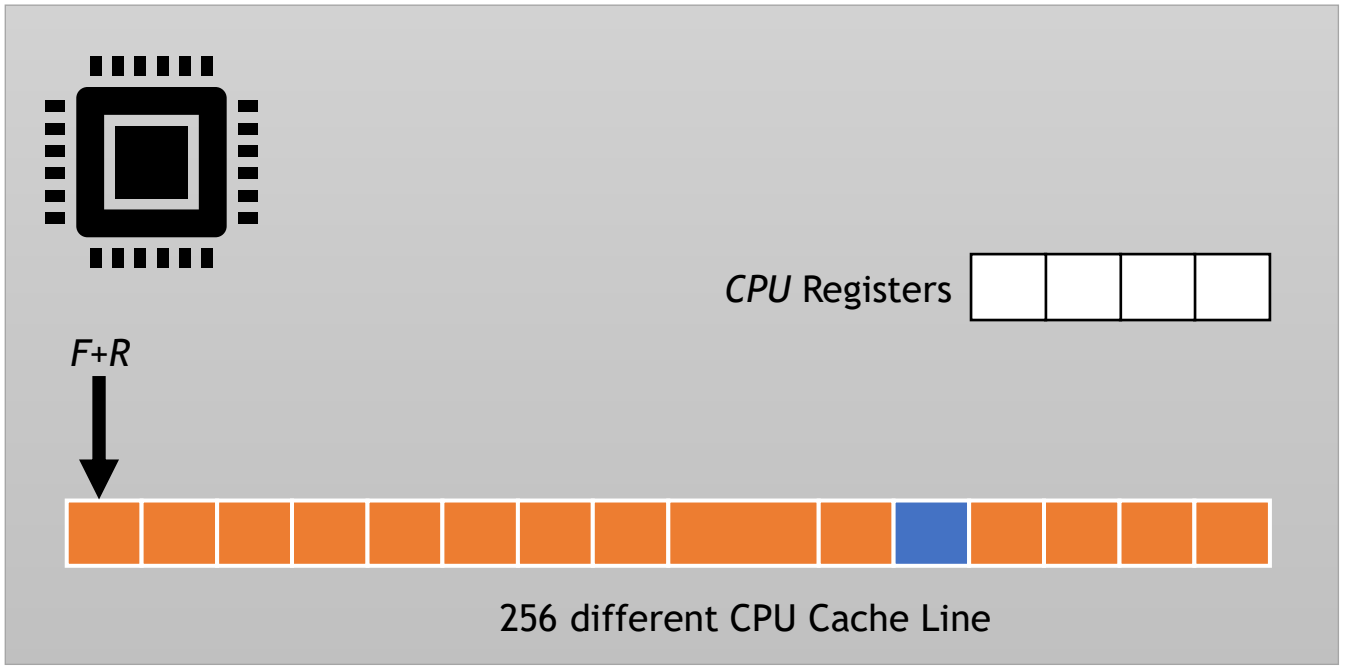




Background: Revisiting Meltdown-style Attacks (Step 3)

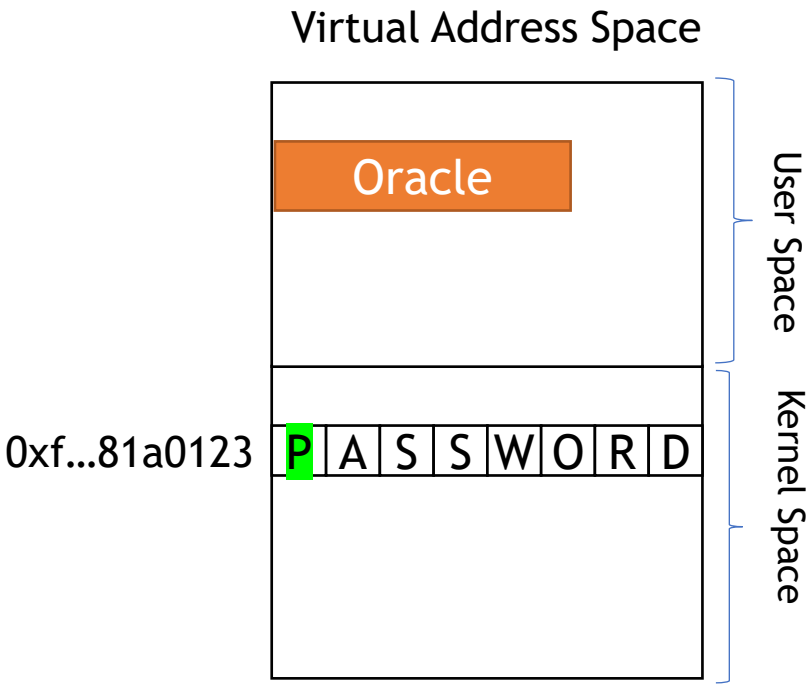
```
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```

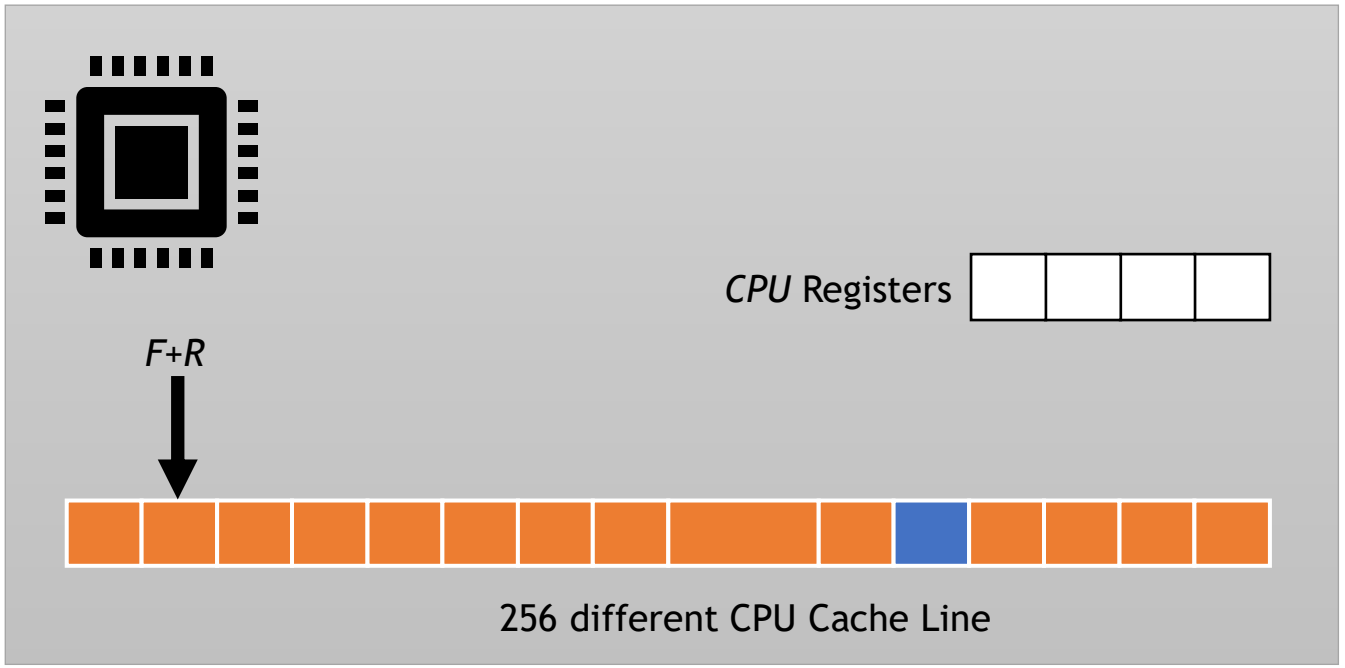




Background: Revisiting Meltdown-style Attacks (Step 3)

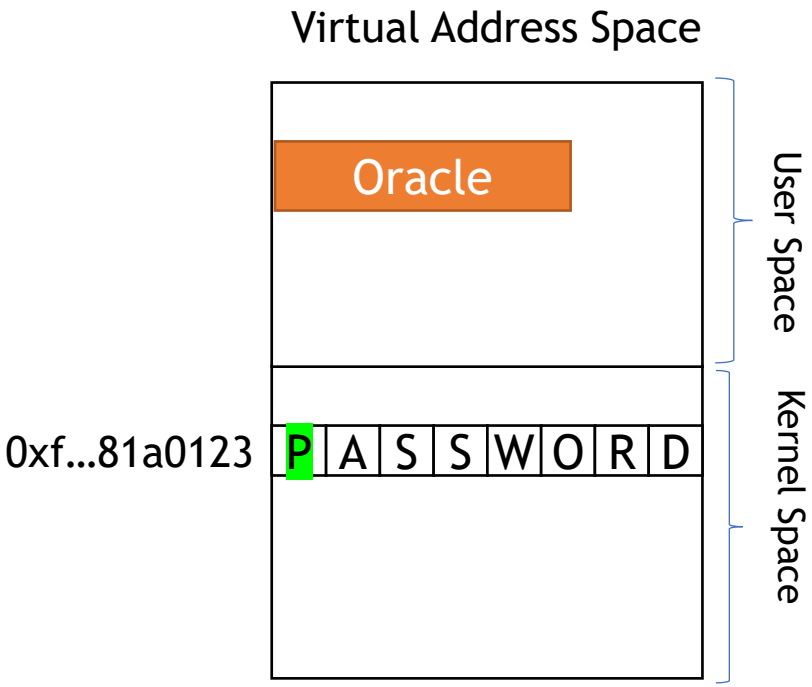
```
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```

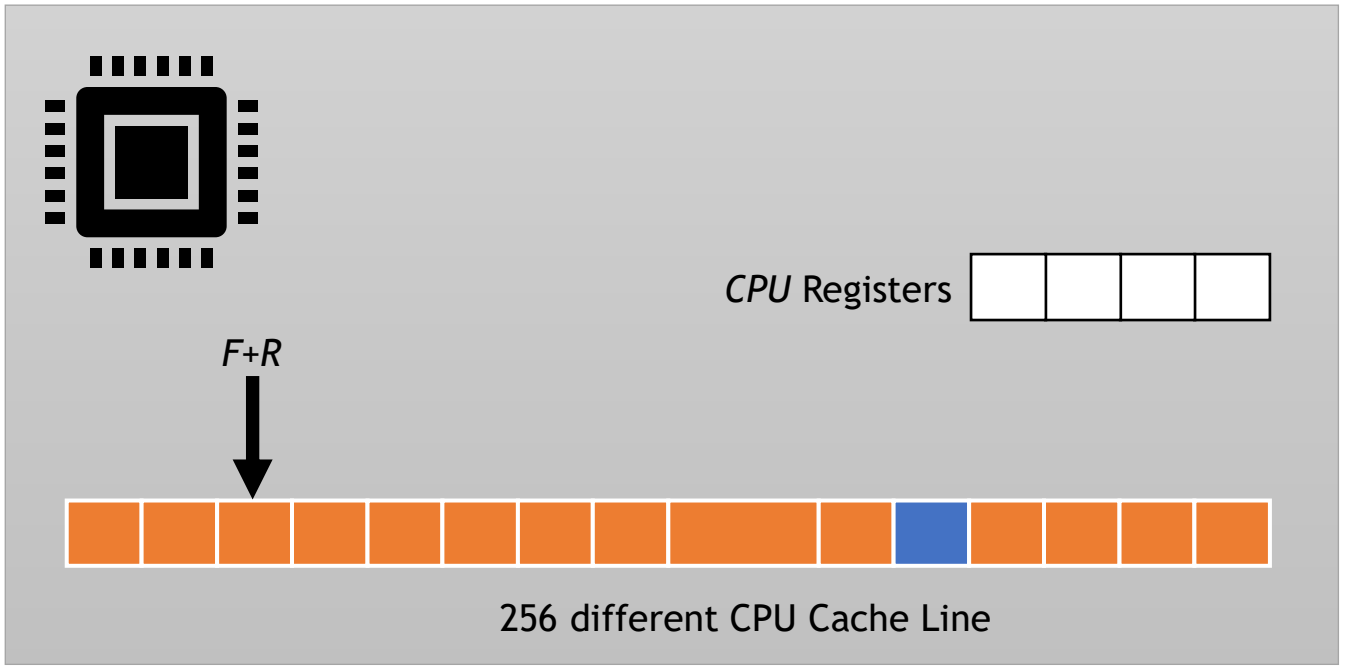




Background: Revisiting Meltdown-style Attacks (Step 3)

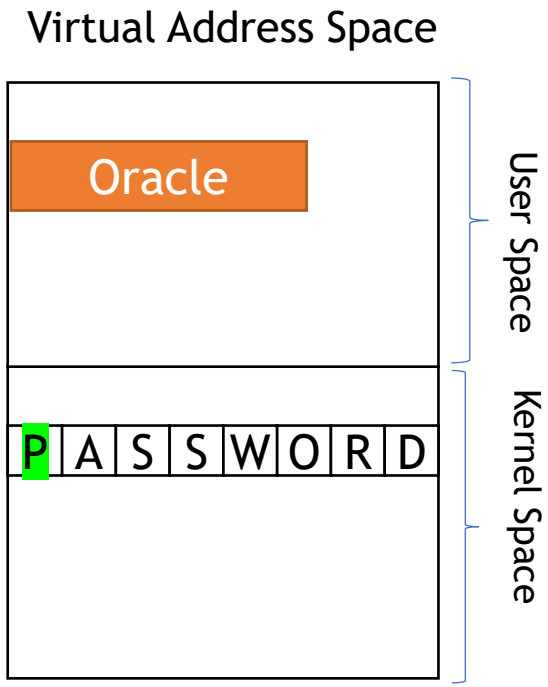
```
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```

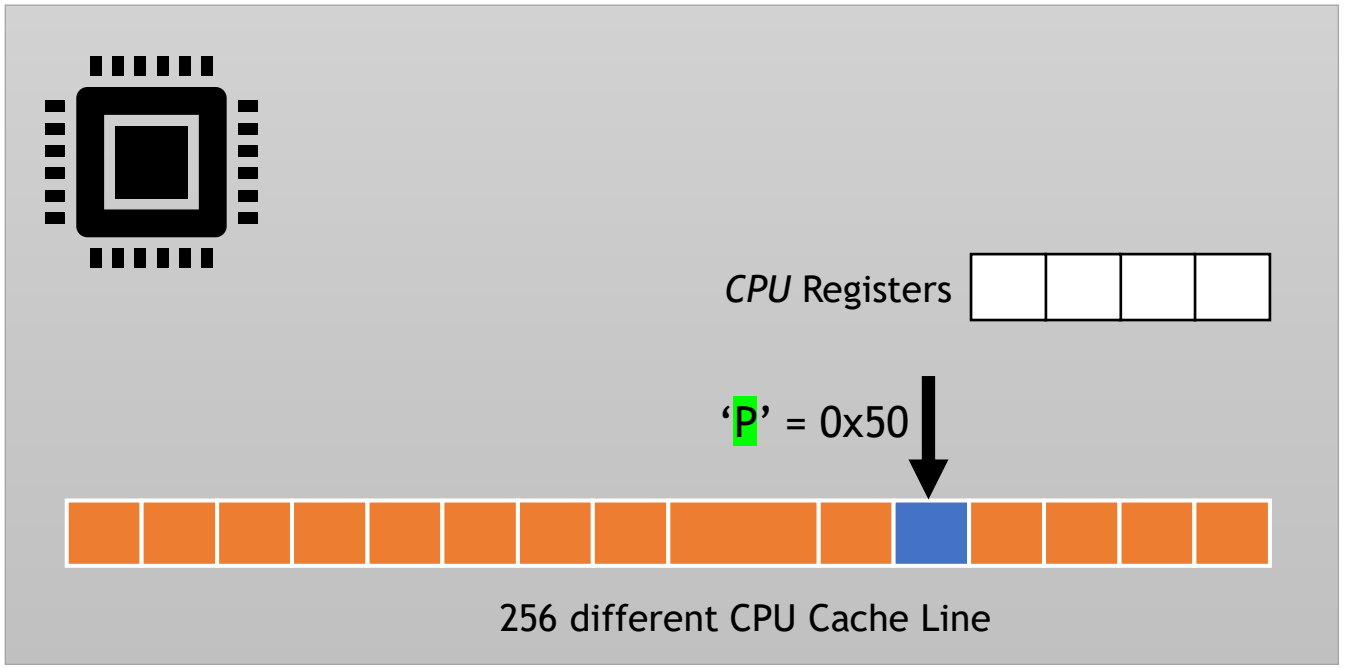




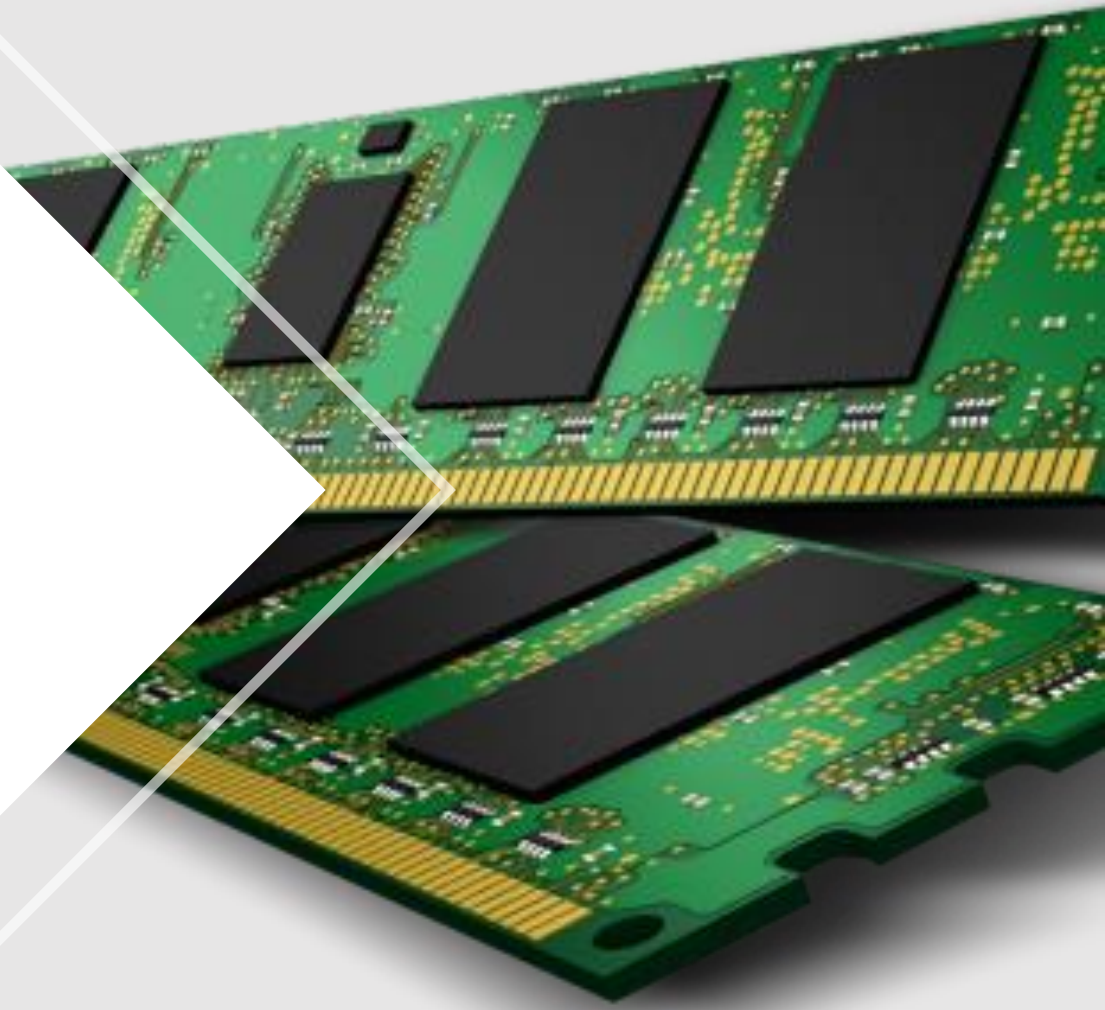
Background: Revisiting Meltdown-style Attacks (Step 3)

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



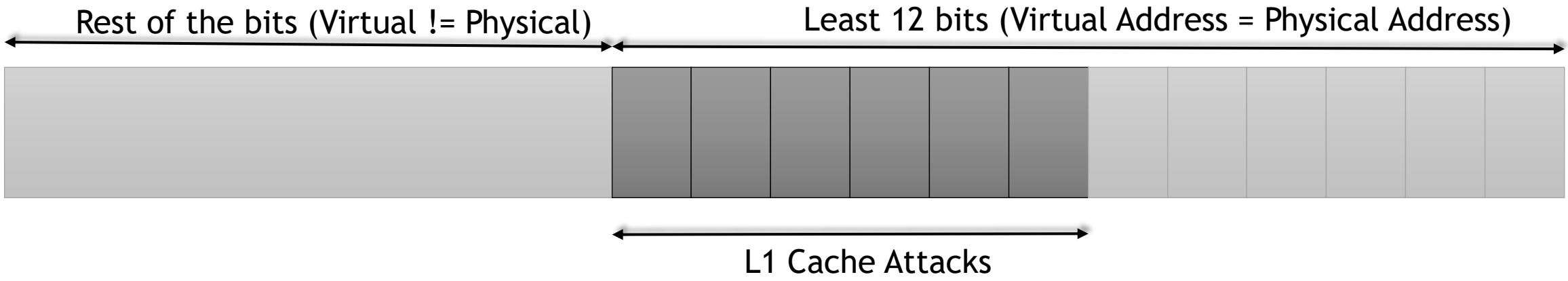


MemJam

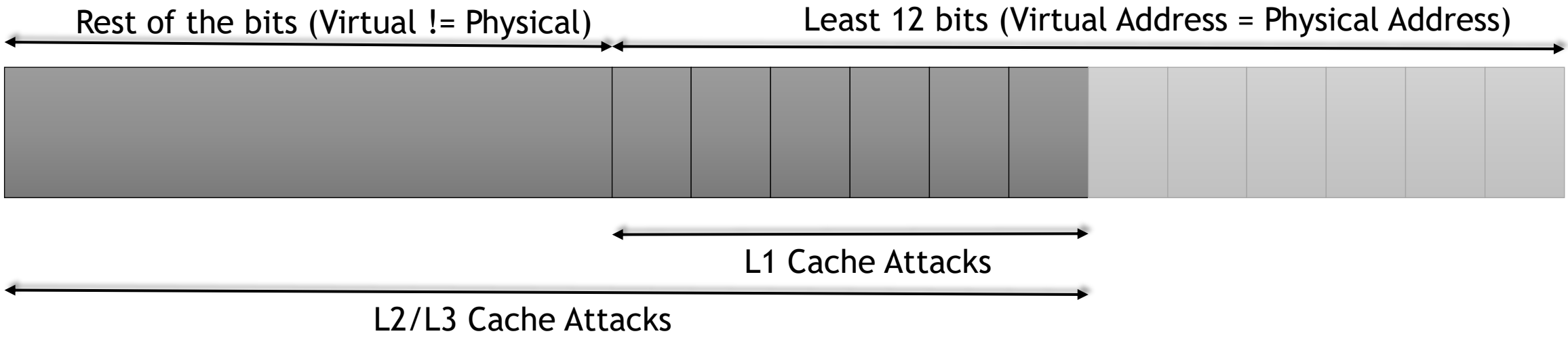


Cache Attacks - Cache Line Resolution

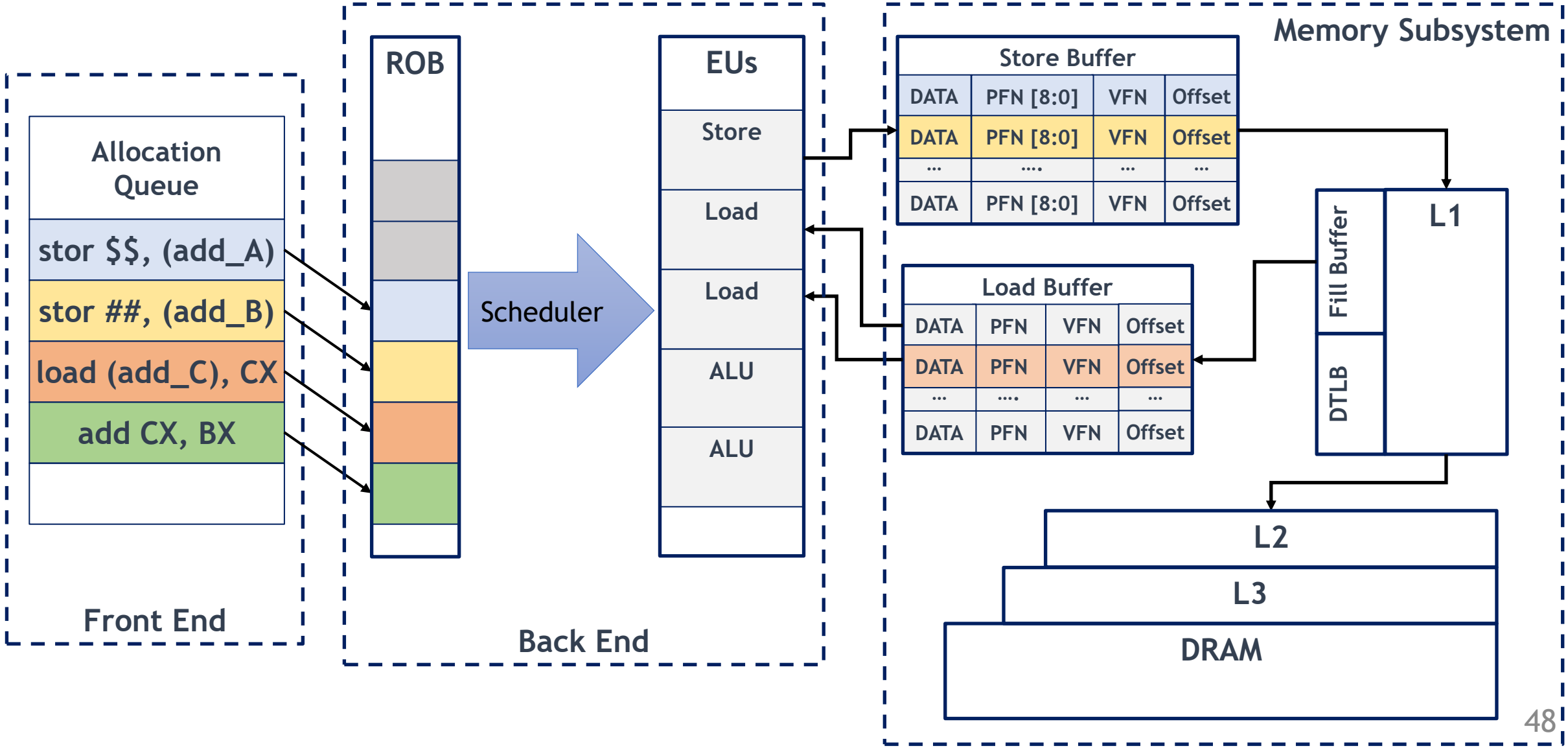
Cache Attacks - Cache Line Resolution



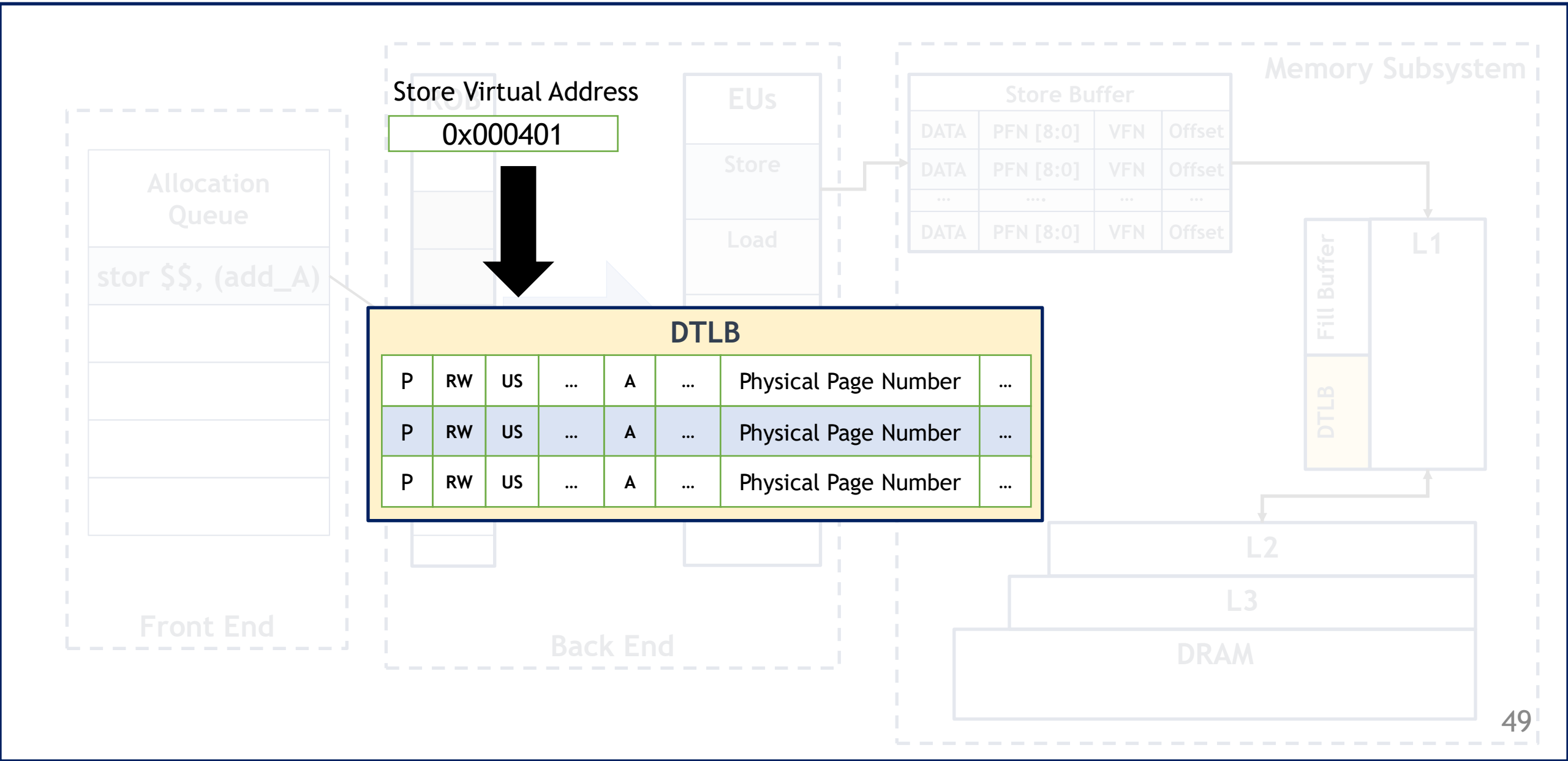
Cache Attacks - Cache Line Resolution



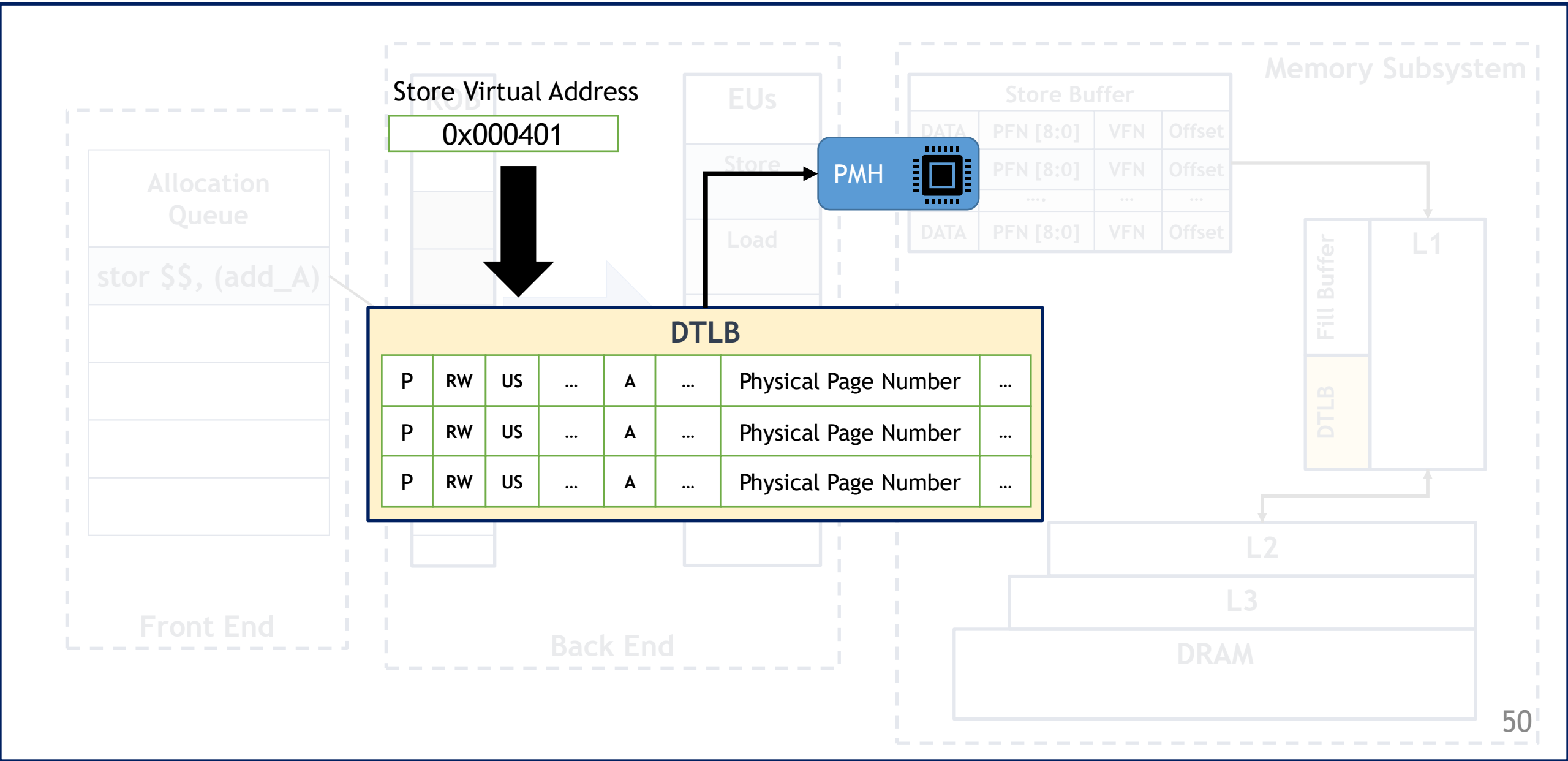
CPU Memory Subsystem



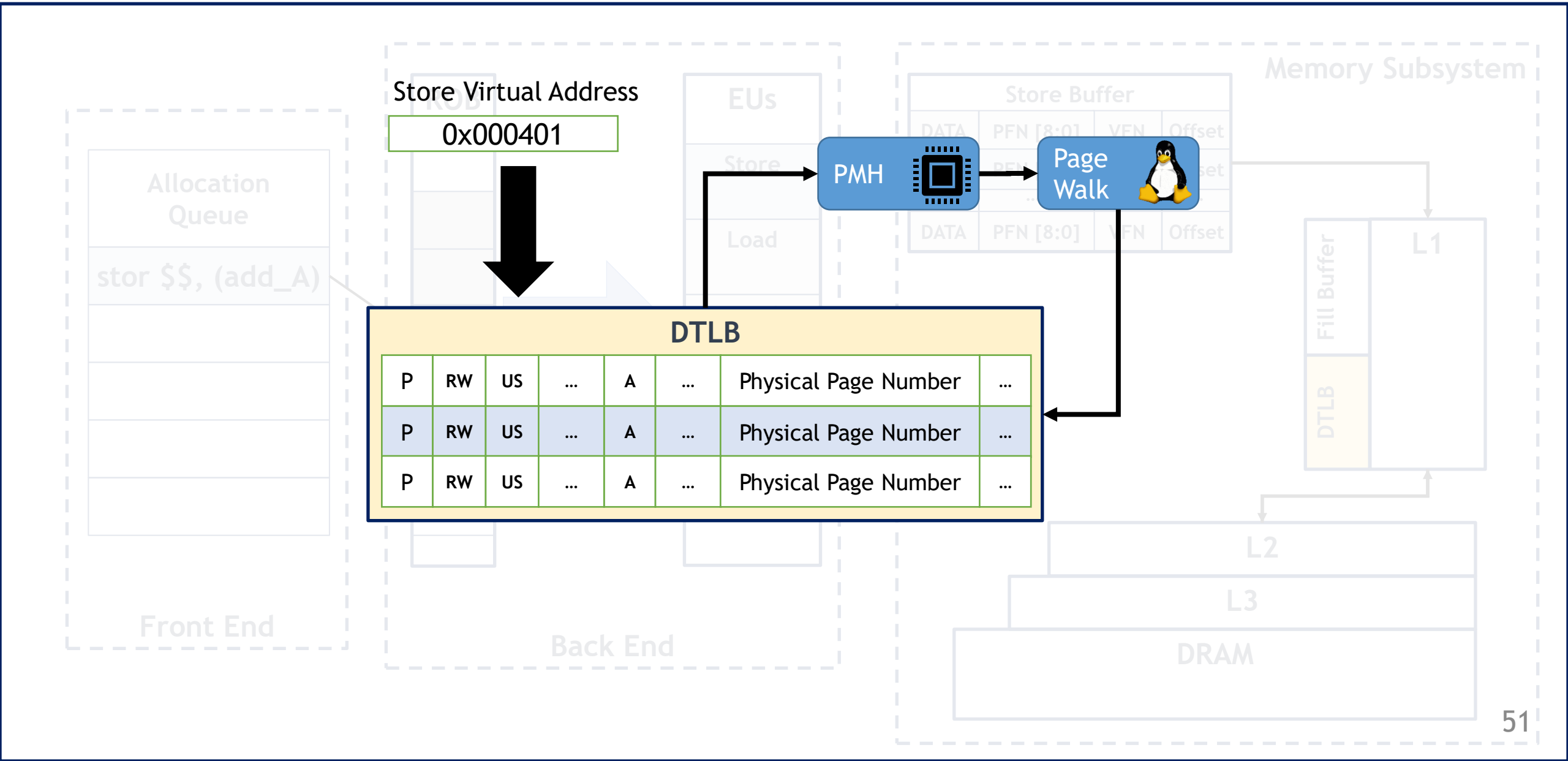
CPU Memory Subsystem – Address Translation



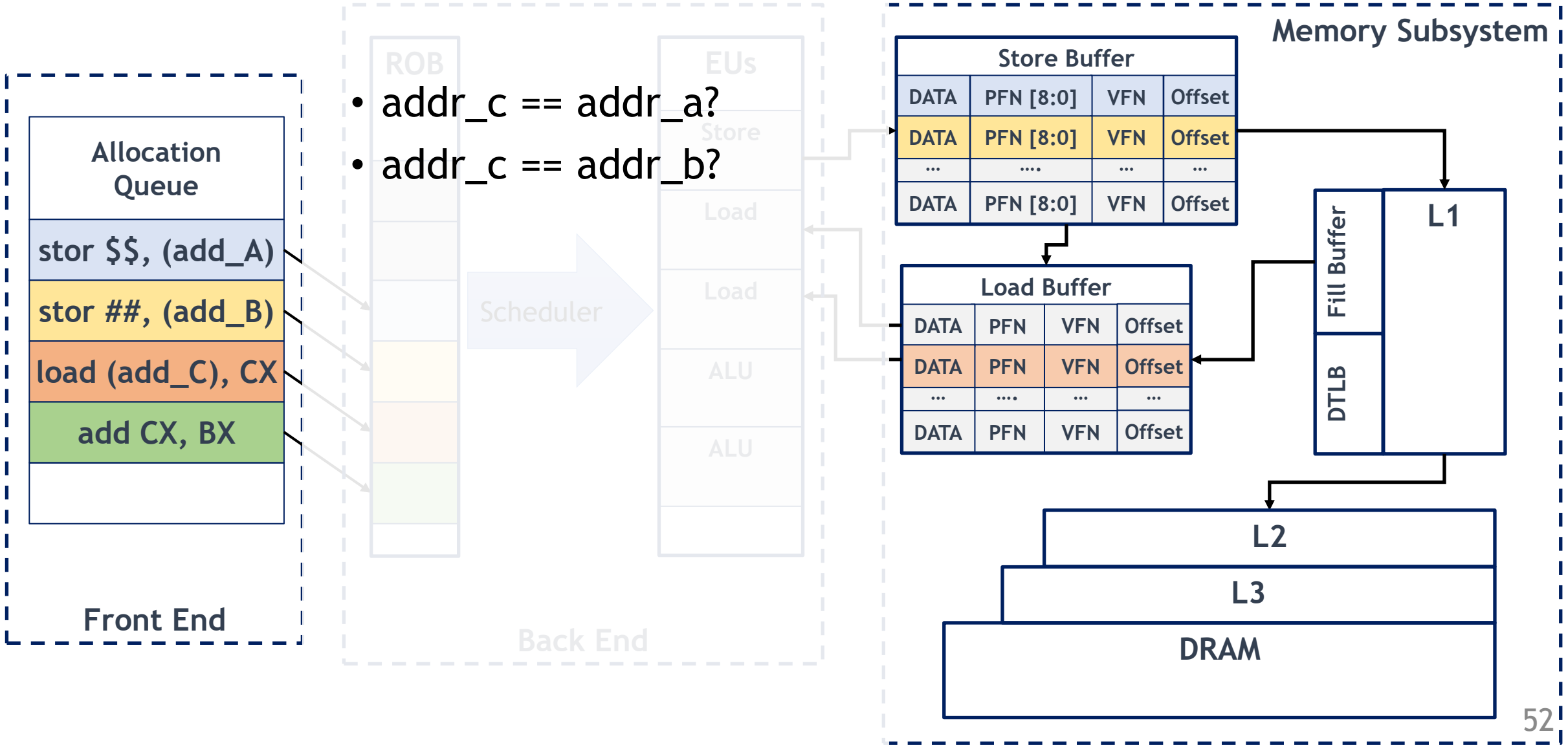
CPU Memory Subsystem - Address Translation



CPU Memory Subsystem - Address Translation



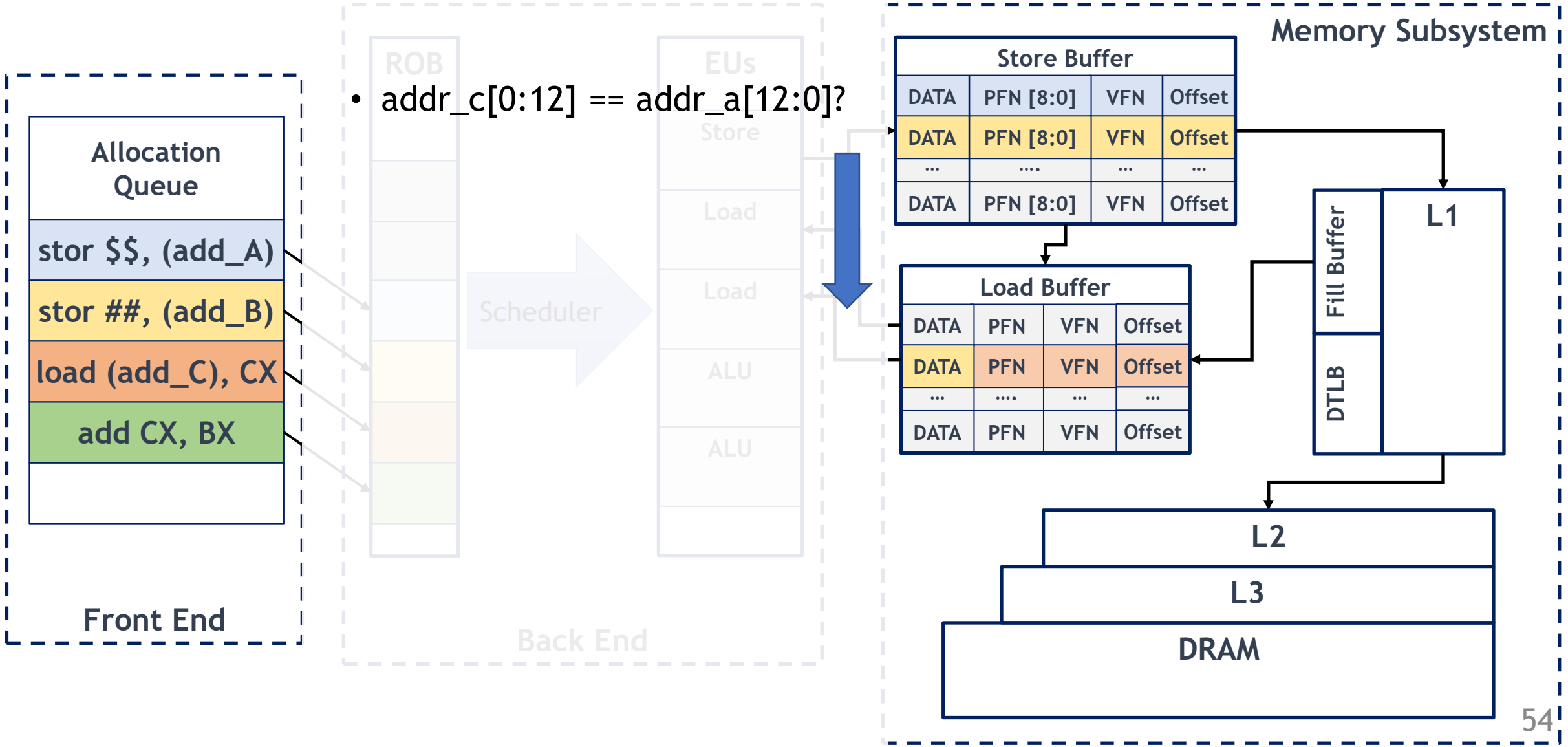
CPU Memory Subsystem - Store Forwarding



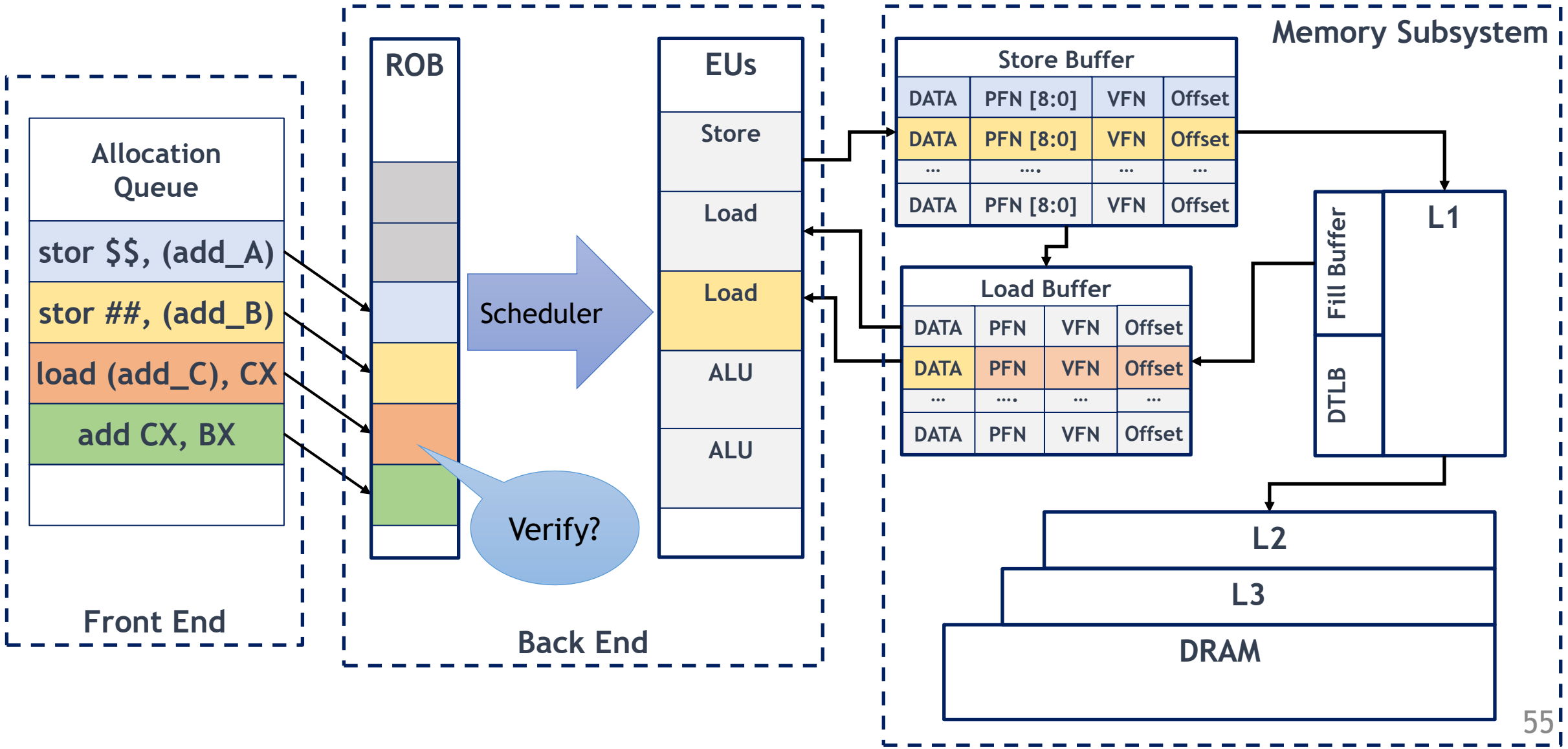
MemJam Attack

- 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.

CPU Memory Subsystem - Store Forwarding



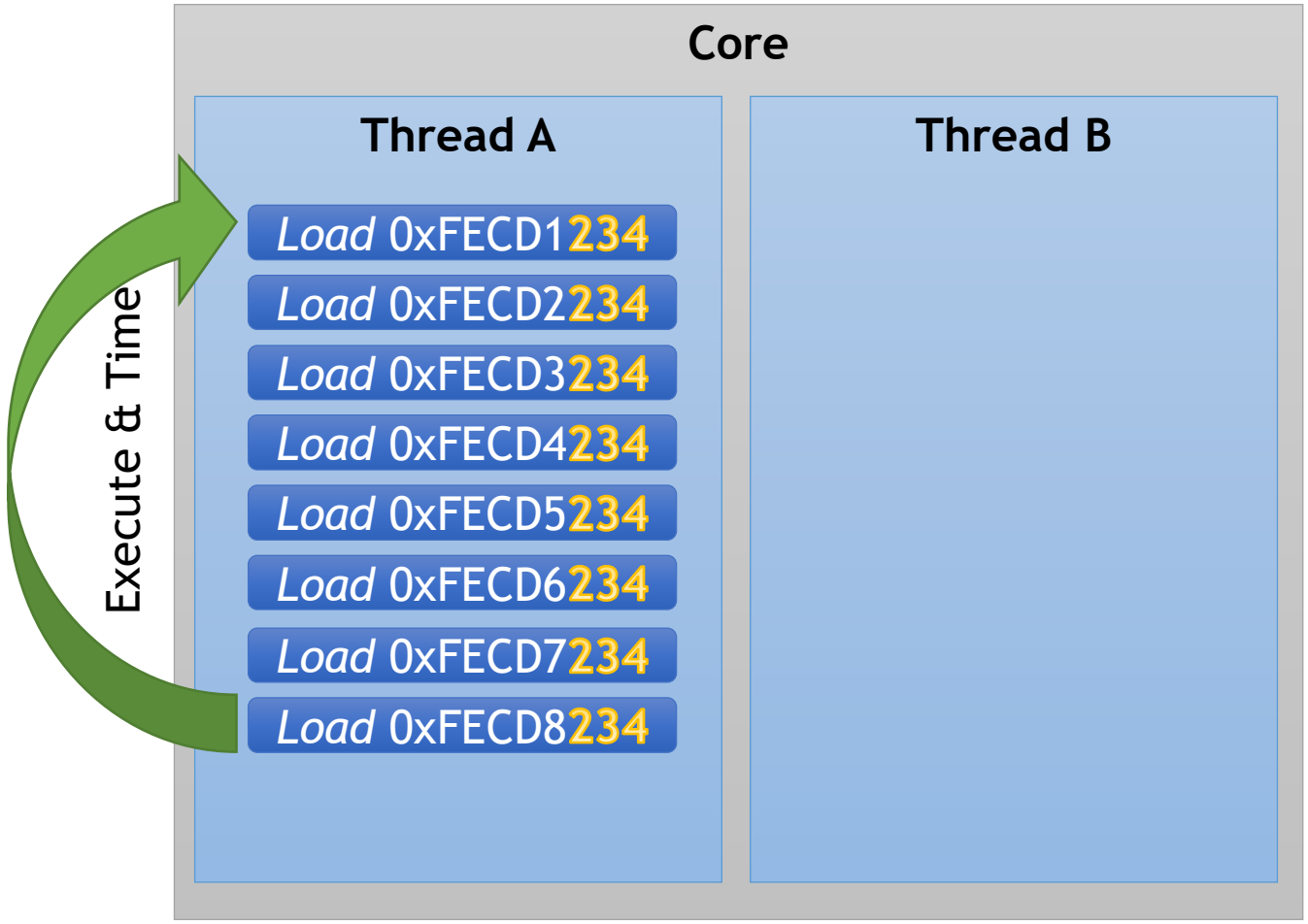
CPU Memory Subsystem – Store Forwarding



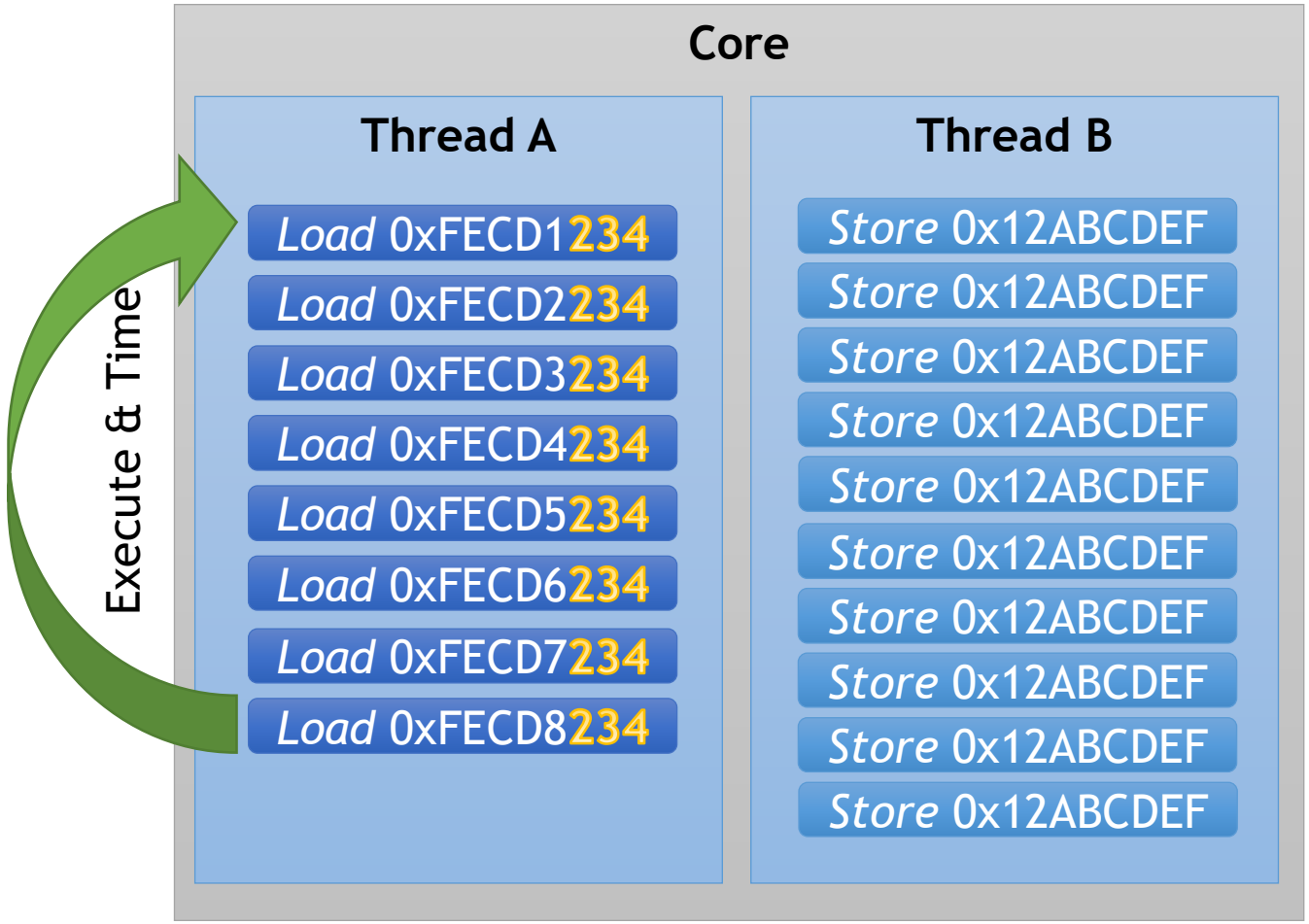
MemJam Attack

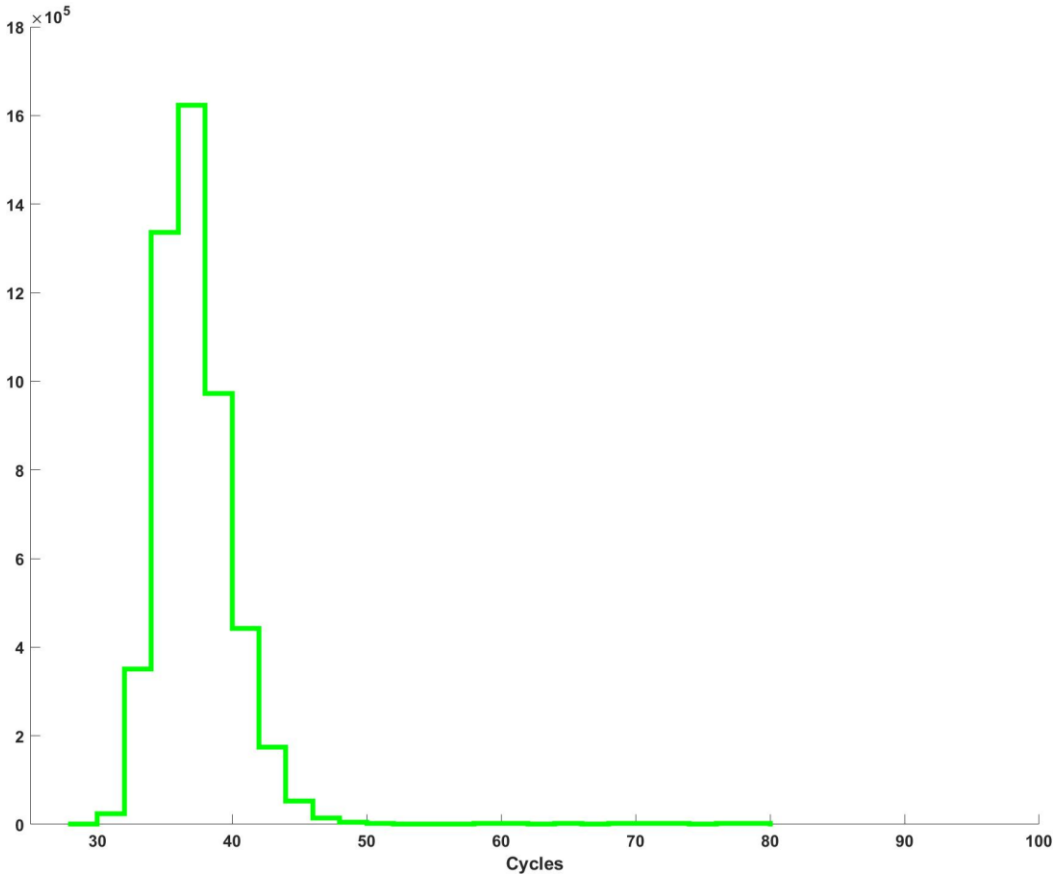
- Memory loads/stores are executed out of order and speculatively.
- Address translation can be expensive.
- 4K Aliasing: Addresses that are 4K apart are assumed dependent.
- The dependency is verified after the execution!
- Re-execution of the load block due to false dependency
  - It causes timing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

MemJam - 4K Aliasing across Sibling Threads

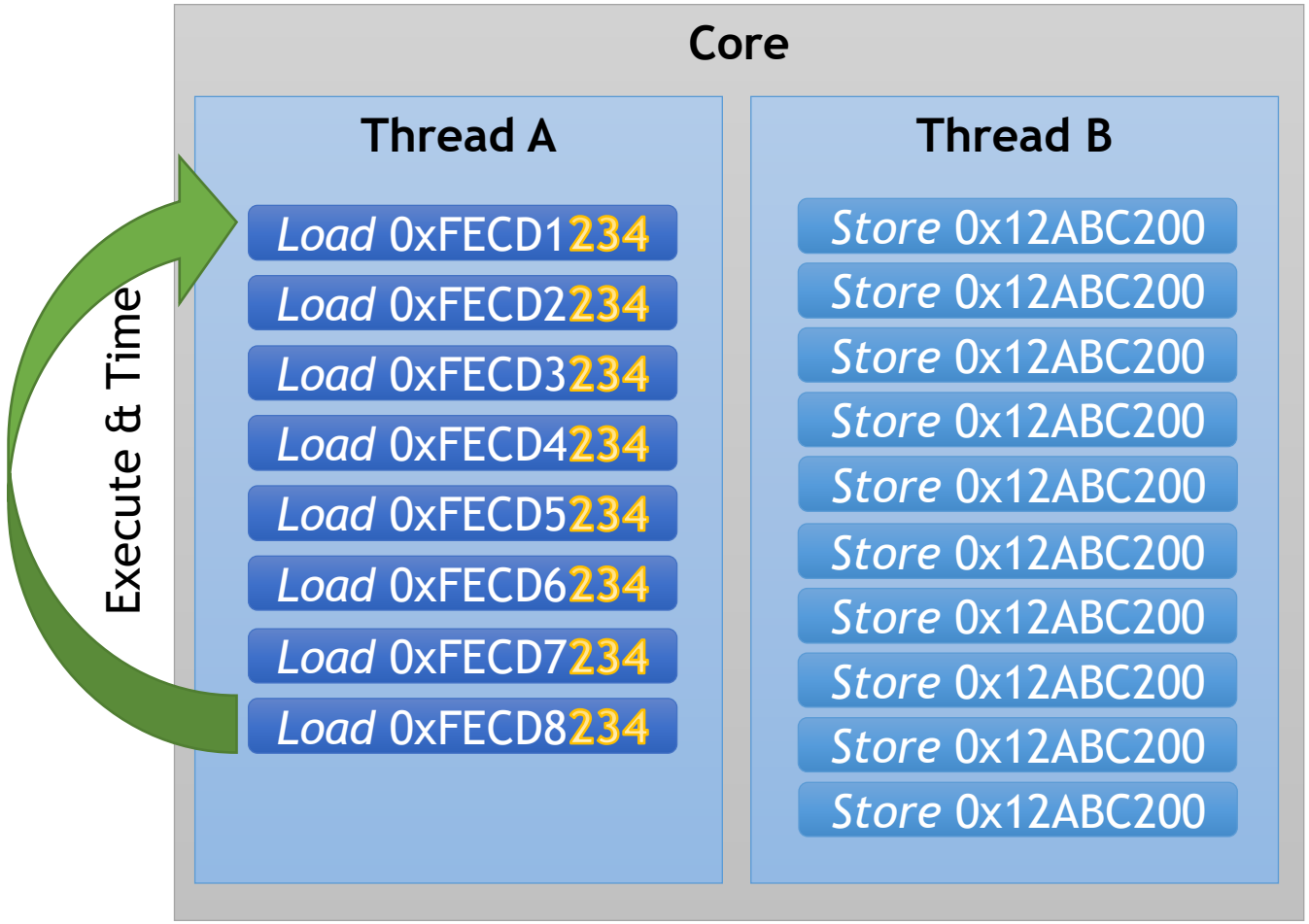


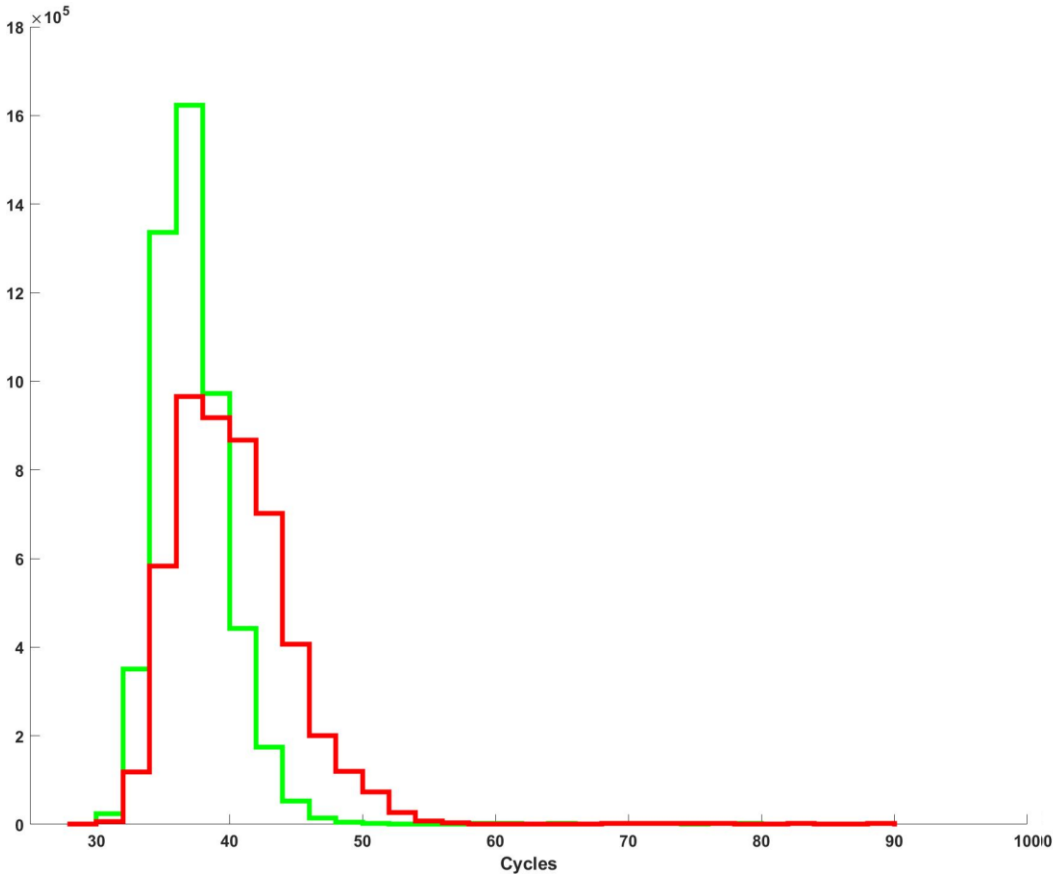
MemJam – 4K Aliasing across Sibling Threads



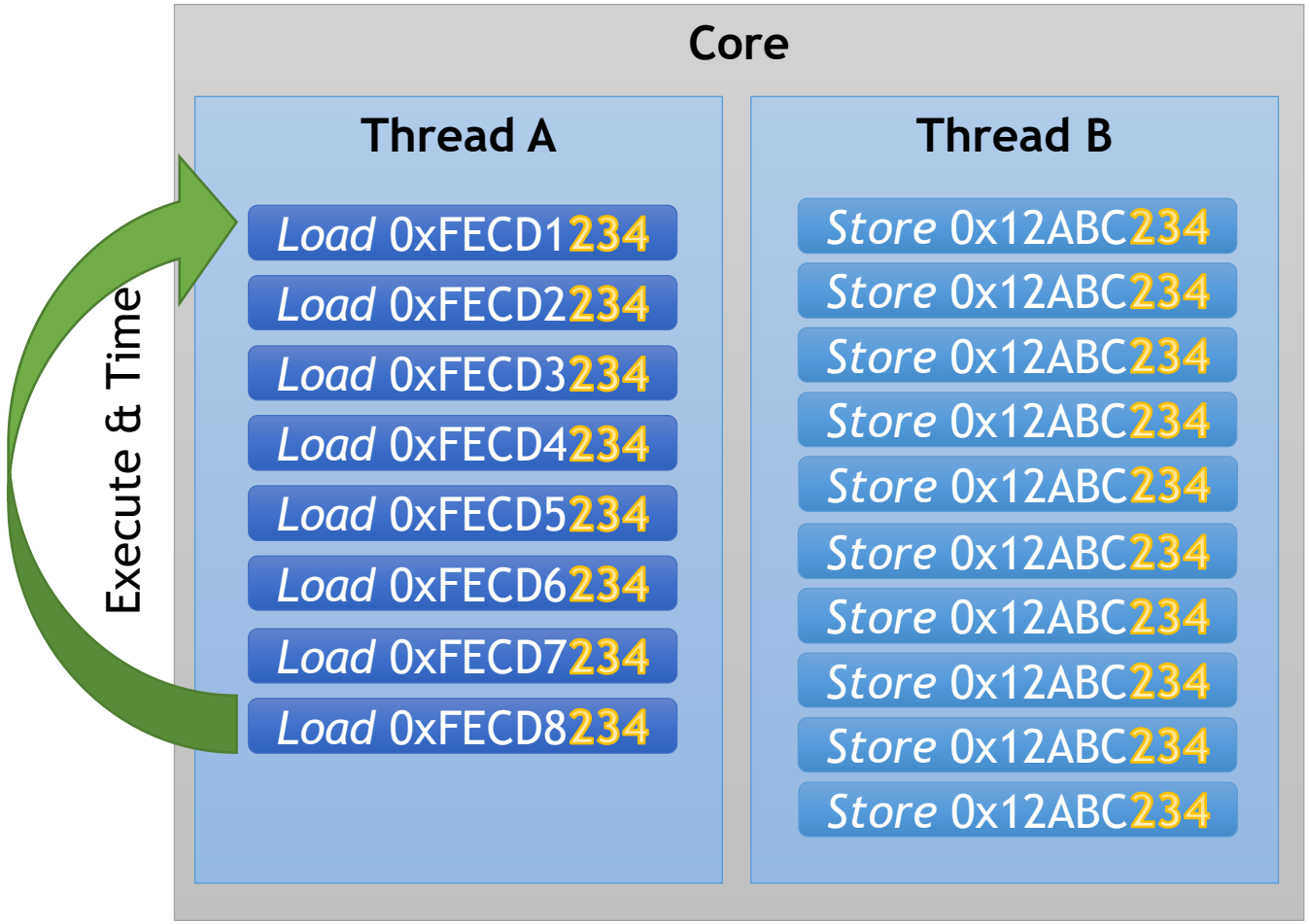


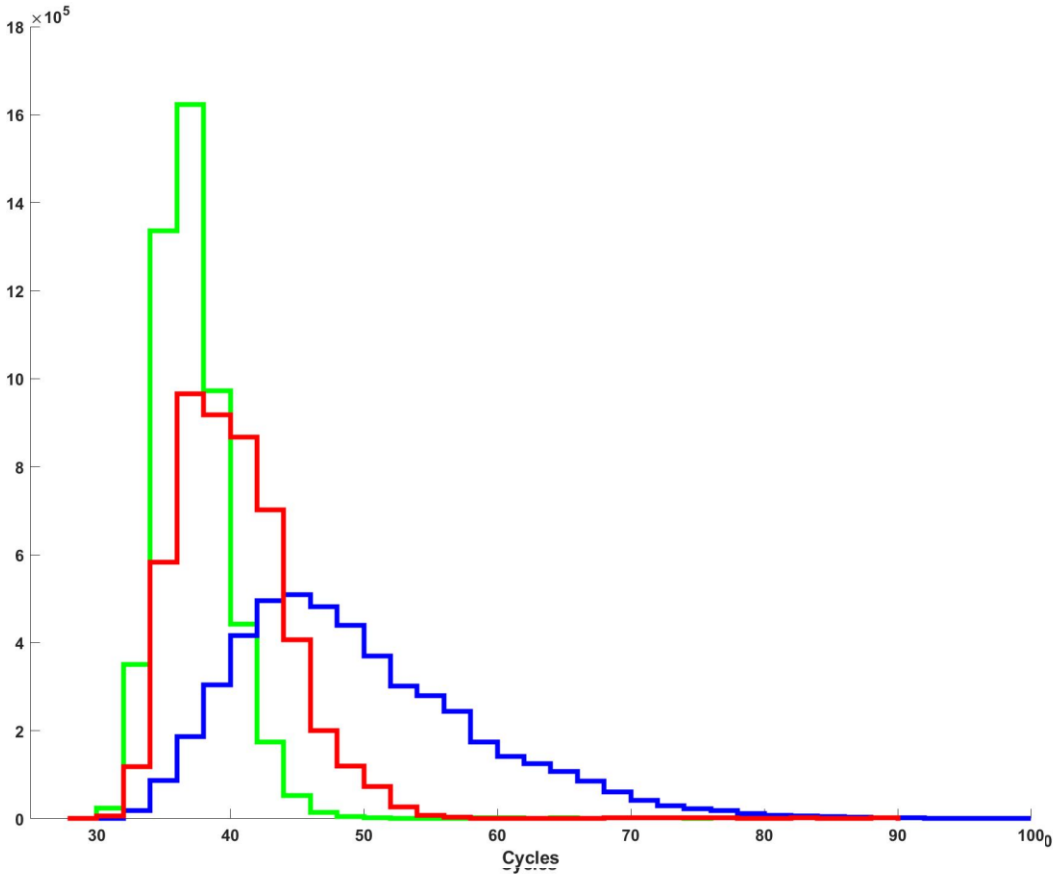
MemJam - 4K Aliasing across Sibling Threads



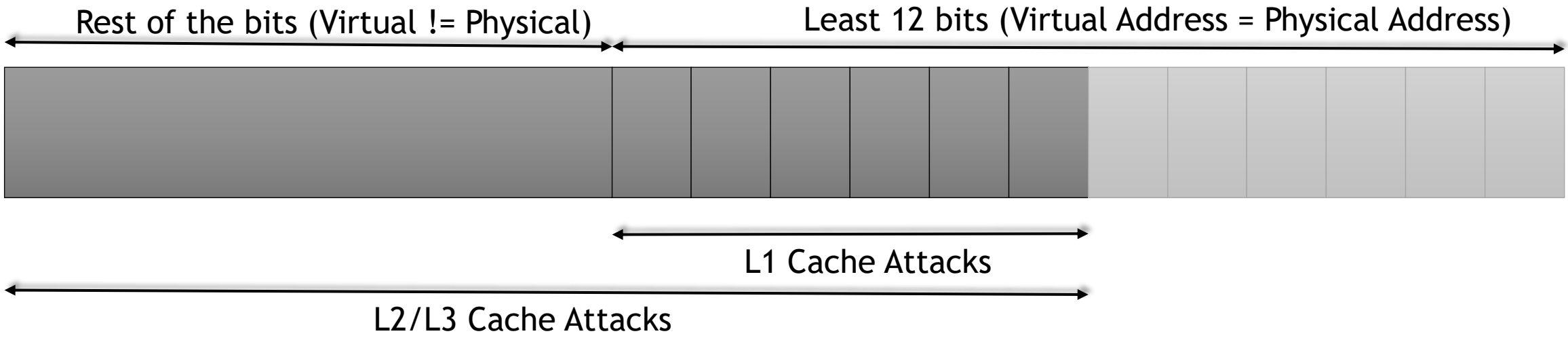


MemJam – 4K Aliasing across Sibling Threads

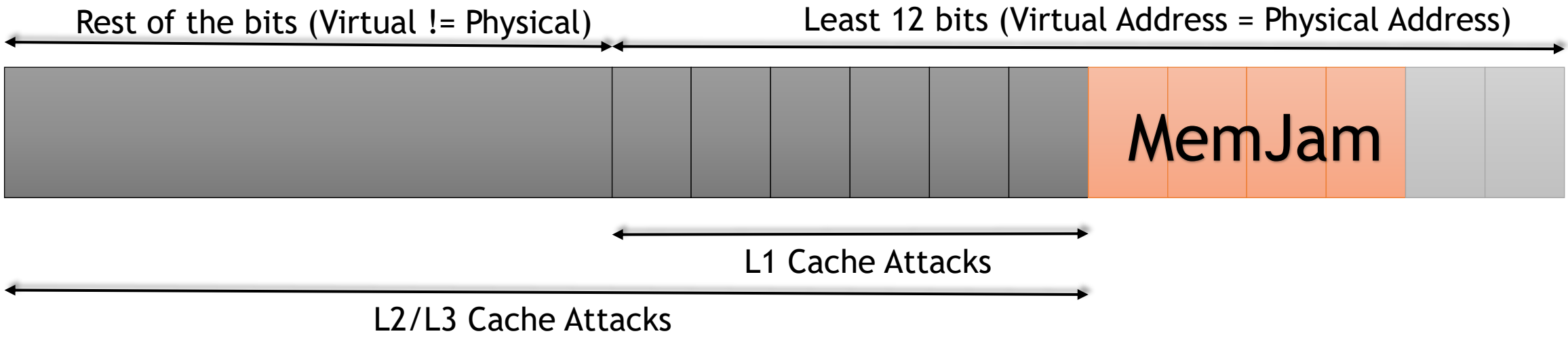




MemJam – Intra Cache Line Resolution



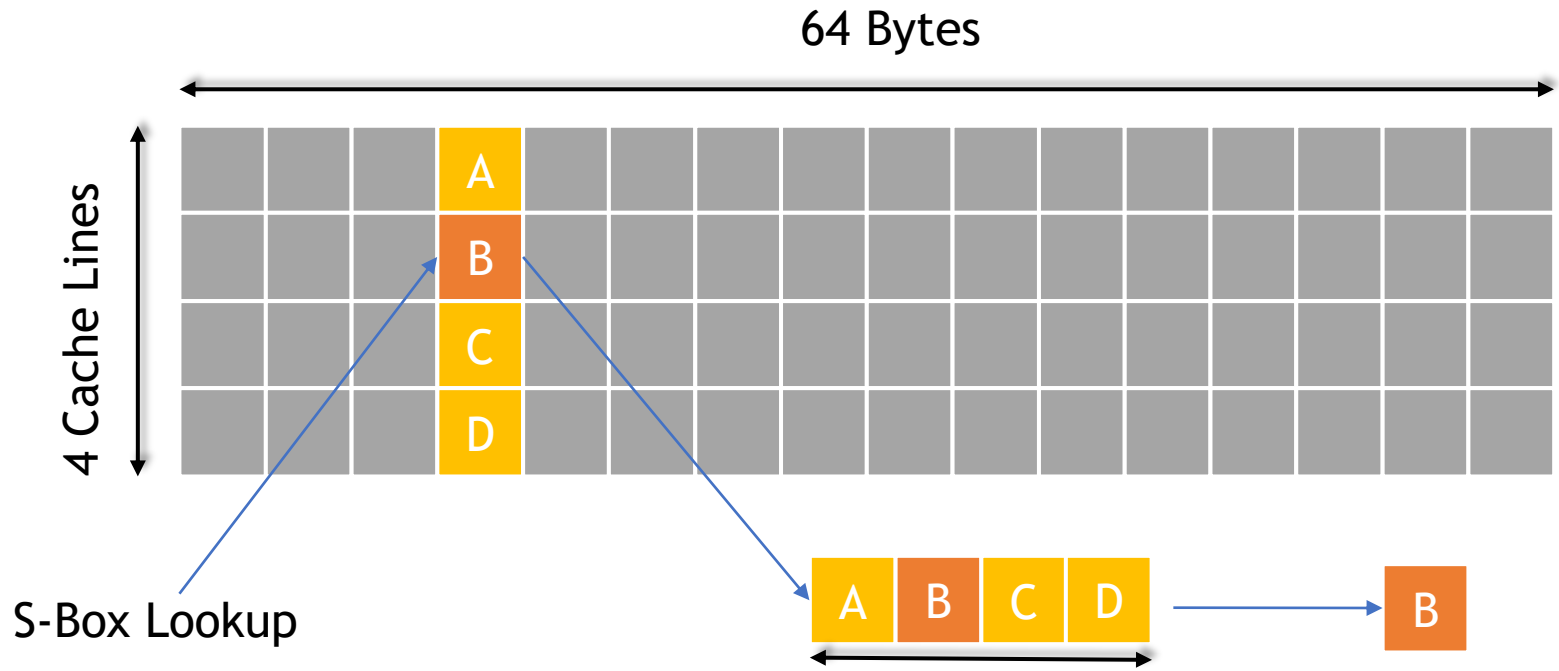
MemJam - Intra Cache Line Resolution



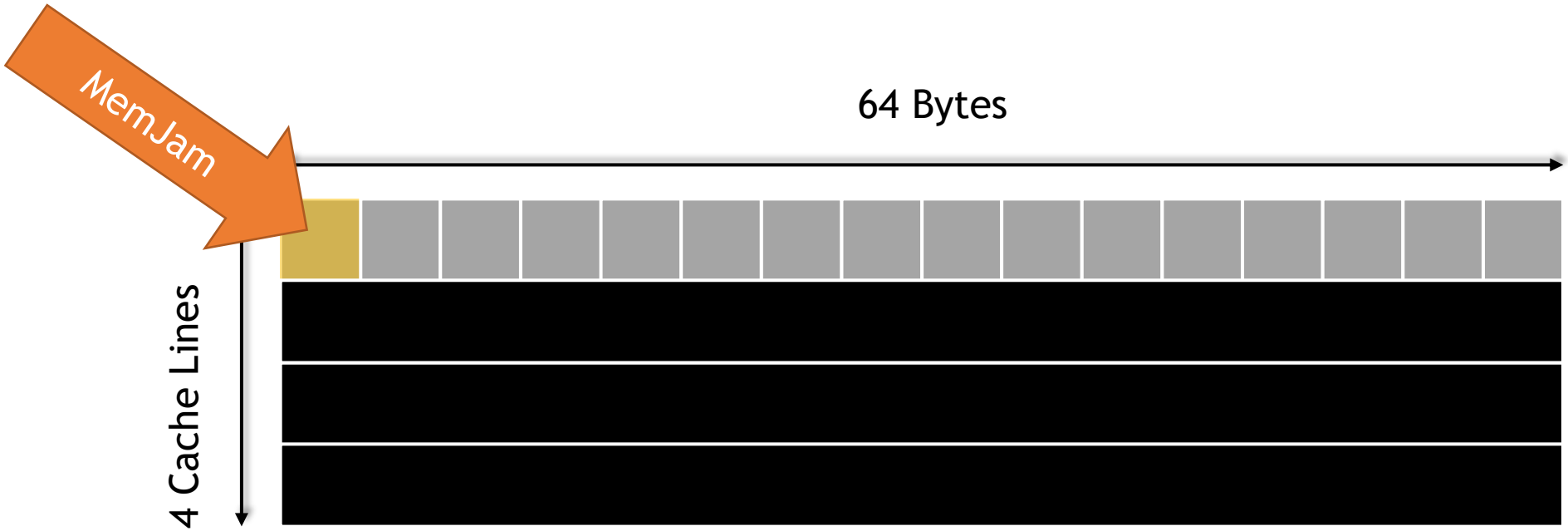
- Conflicted intra-cache line leakage (4-byte granularity)
- Higher time → Memory accesses with the same bit 3 - 12
- 4 bits of intra-cache level leakage

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

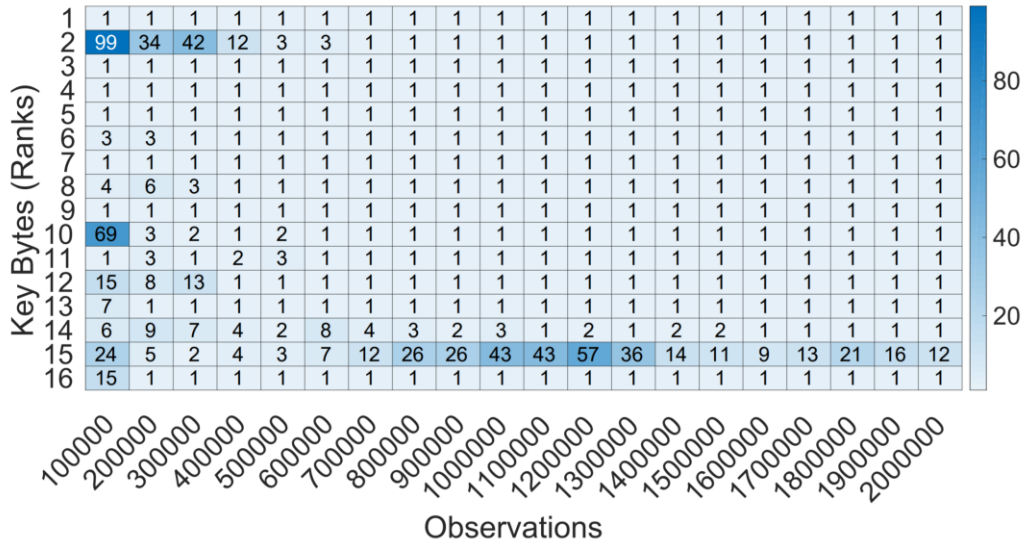


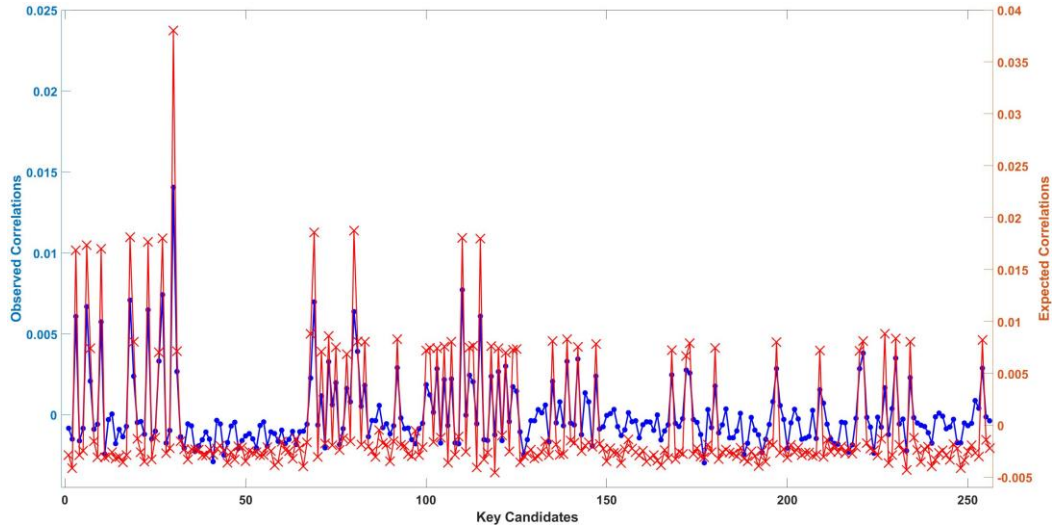
MemJam - Attacking So-Called Constant Time AES



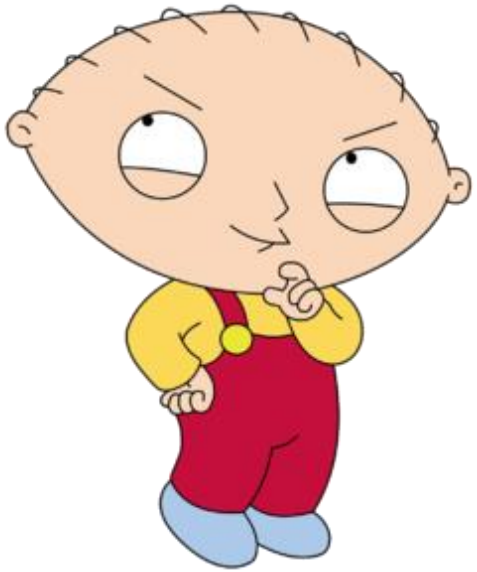
$$index = S^{-1}(c \oplus k) \longrightarrow index < 4$$

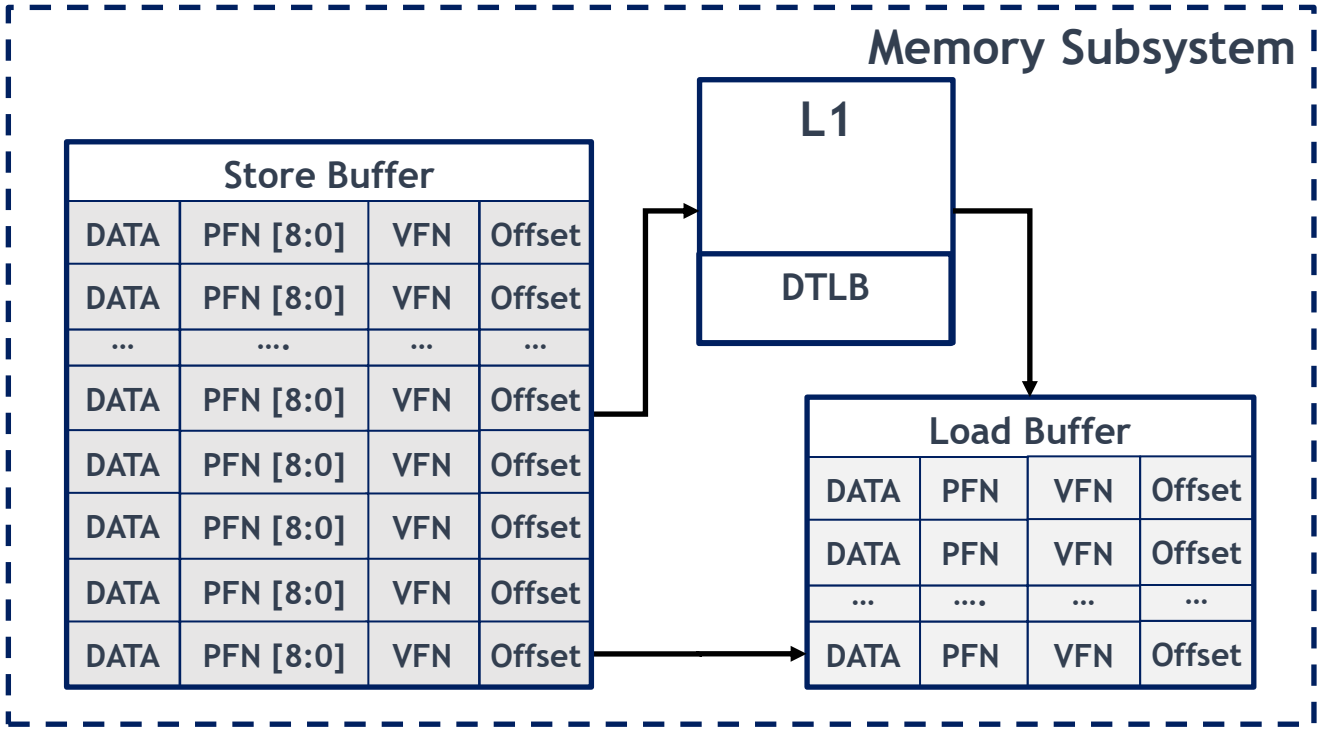
AES Key Recovery





Are there other Address Aliasing?





SPOILER

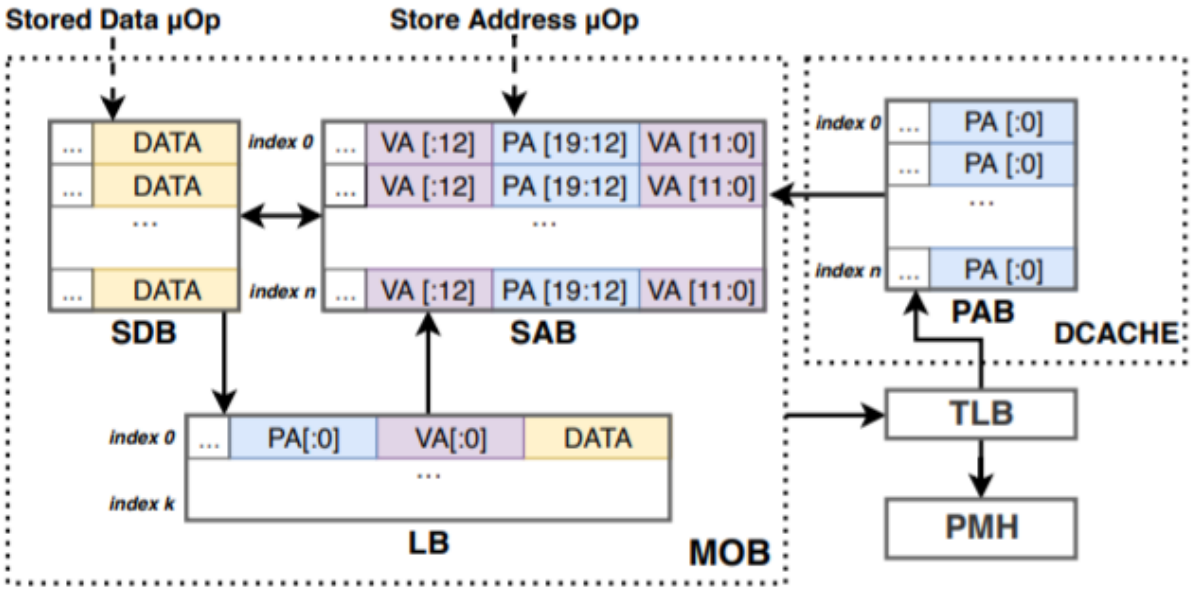
US 7,603,527 B2 *RESOLVING FALSE DEPENDENCIES OF SPECULATIVE LOAD INSTRUCTIONS*

"an operation X may determine whether the lower portion of the virtual address of a speculative load instruction matches the lower portion of virtual addresses of older store operations" **Loosnet Check**

….

"in an embodiment, the load instruction may have its input data forwarded from the store operation from which the load instruction depends at operation" **Store Forwarding**

"If there is a hit at operation X and a miss at operation Y, … the physical addresses of the load and the store may be compared at an operation Z" "In one embodiment, if there is a hit at operation X and ***the physical address of the load or the store operations is not valid, the physical address check at operation Z may be considered as a hit"*** *"In some embodiments,* ***the physical address check at operation Z may use a partial physical address****, e.g., base on data stored in the SAB. This makes the checking at operation Z conservative. Accordingly, in some embodiments,* ***a match may occur on a partial address and block…"***
Finenet Check

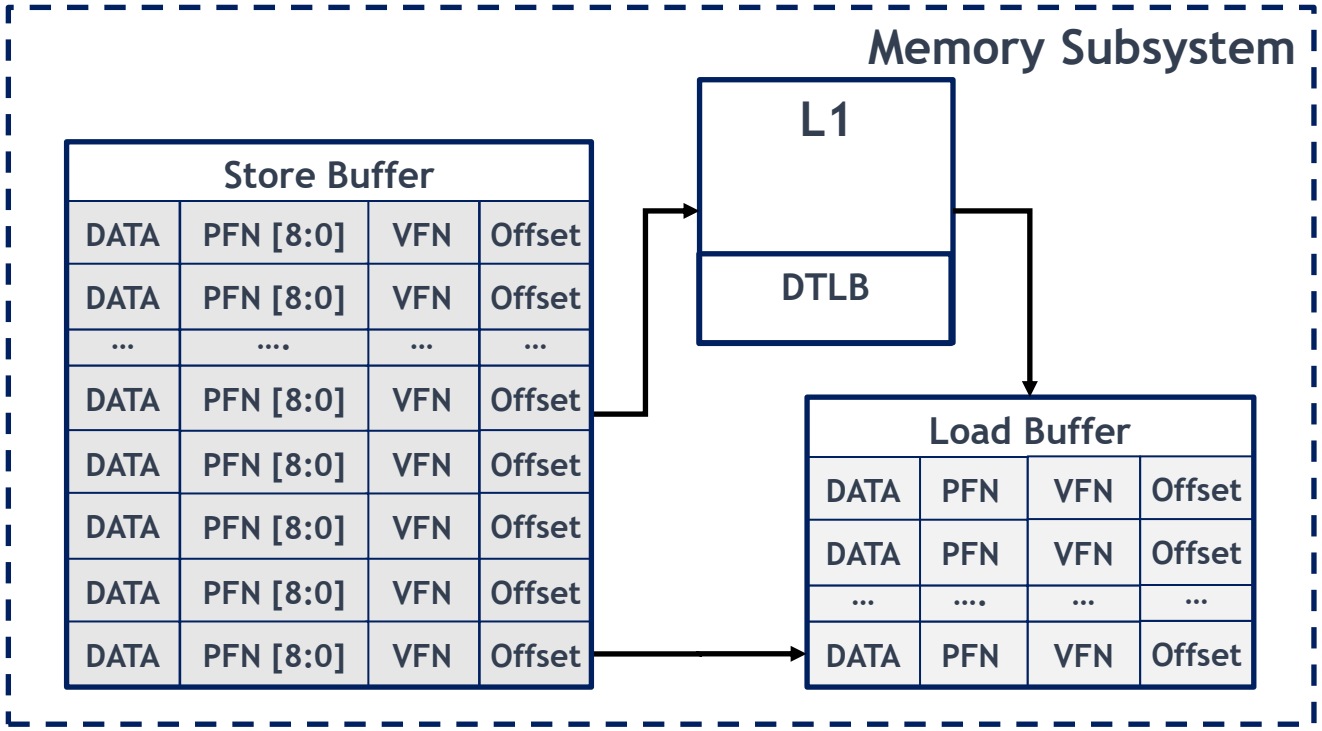


SPOILER Attack

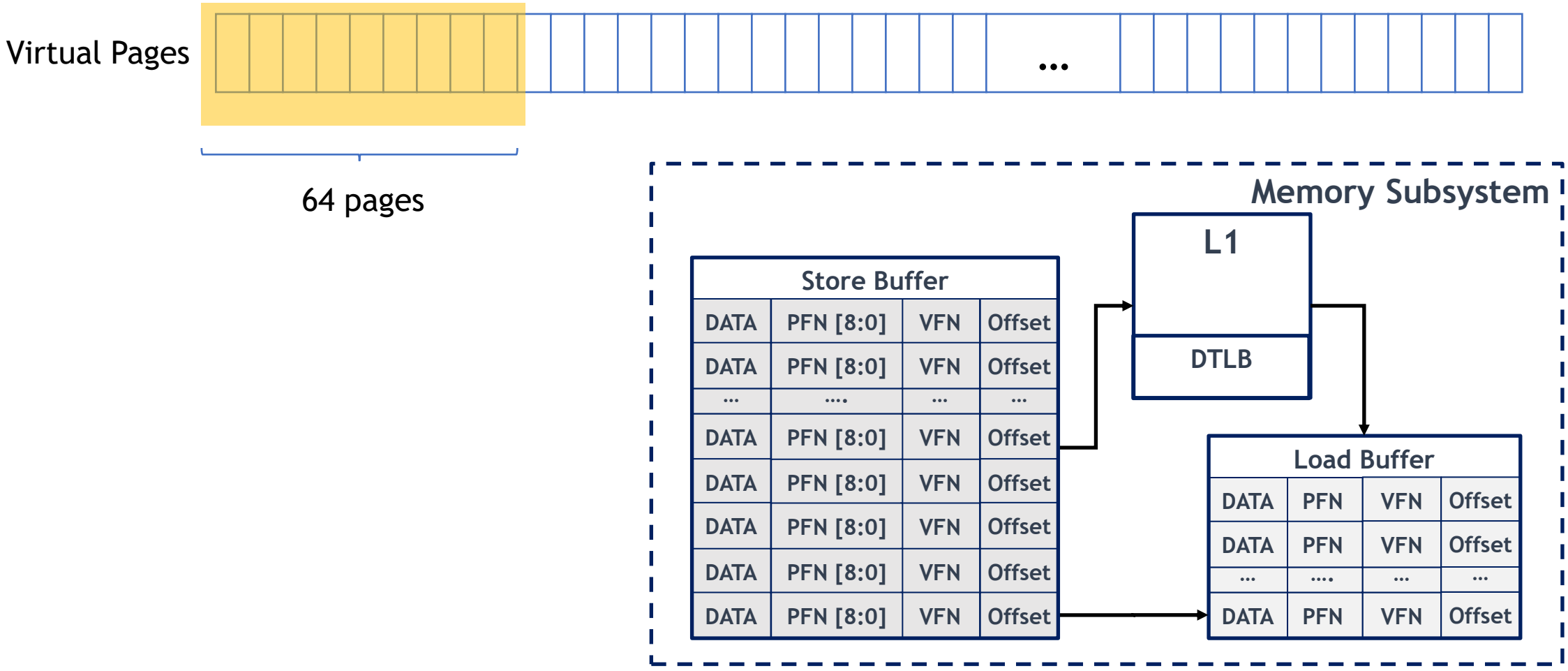
Dependency Resolution

Spoiler: Finding Undocumented Aliasing

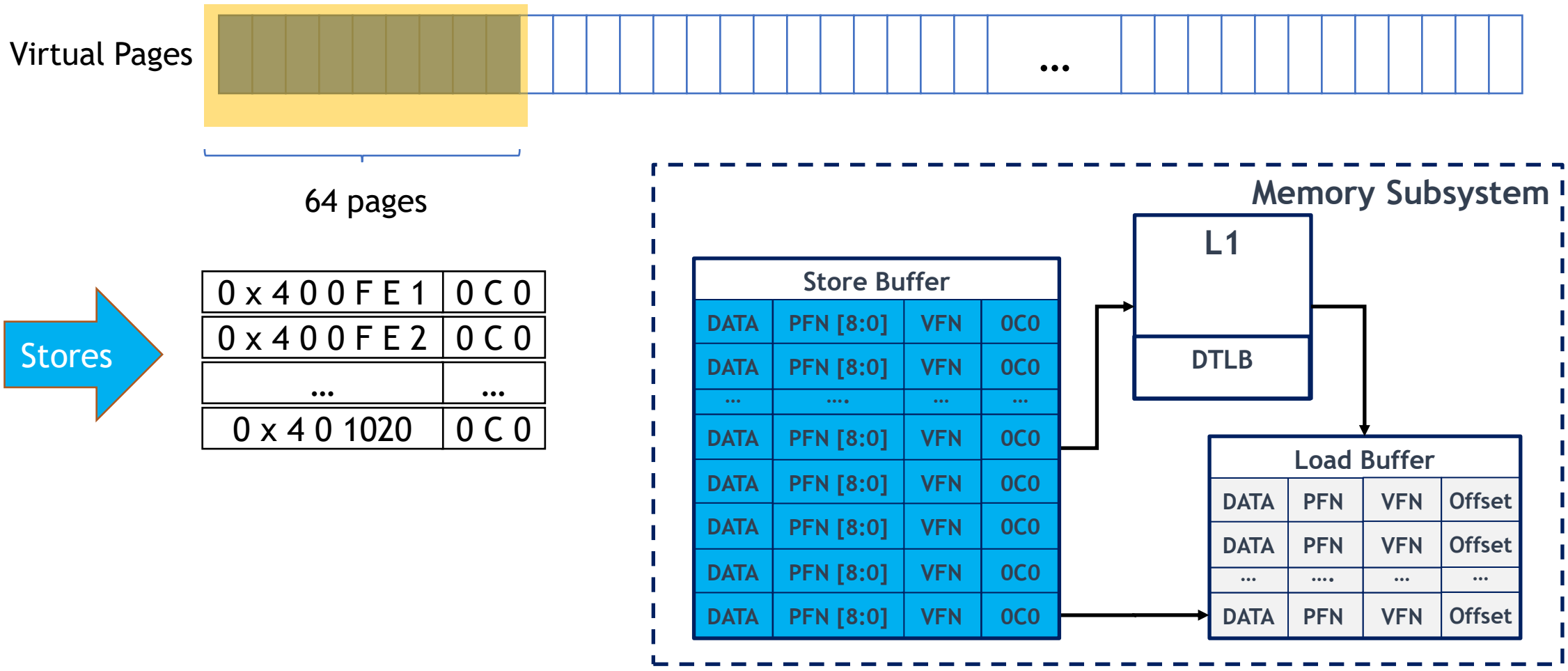
Virtual Pages



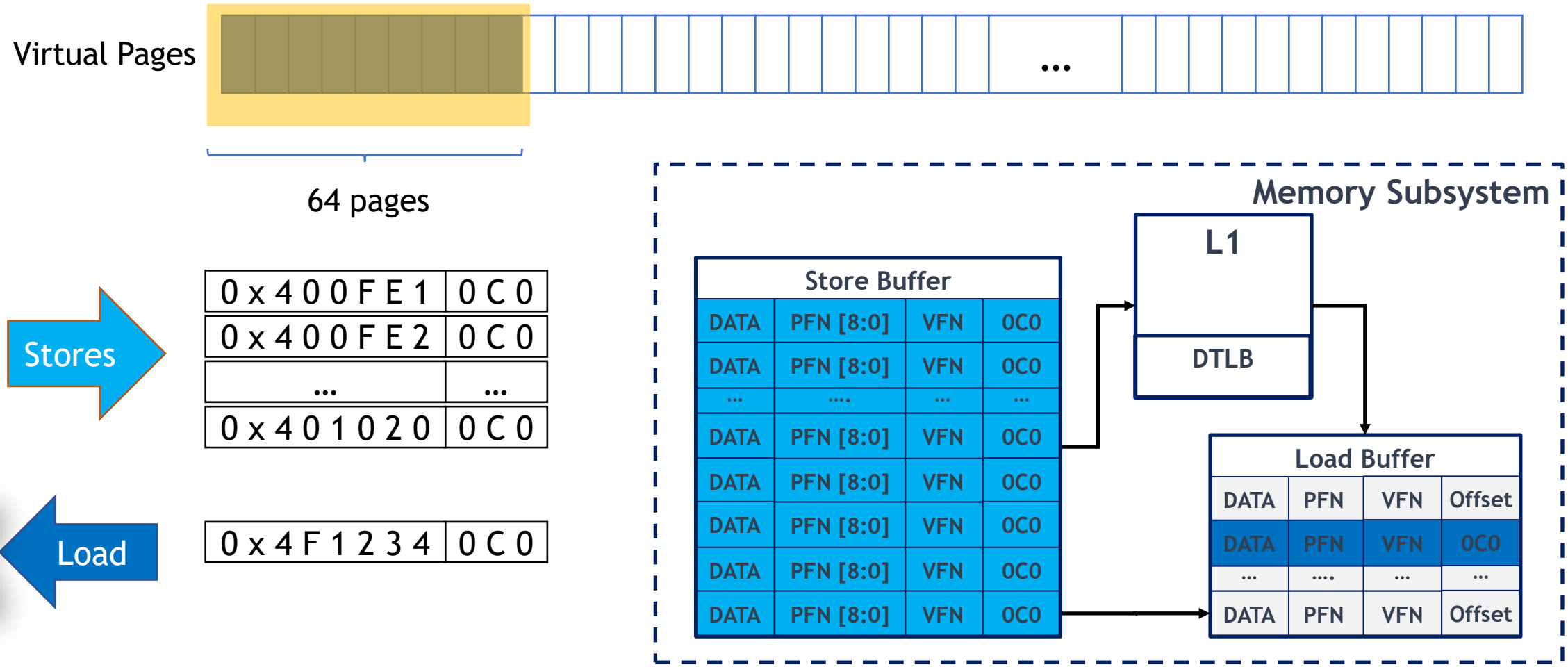
Spoiler: Finding Undocumented Aliasing



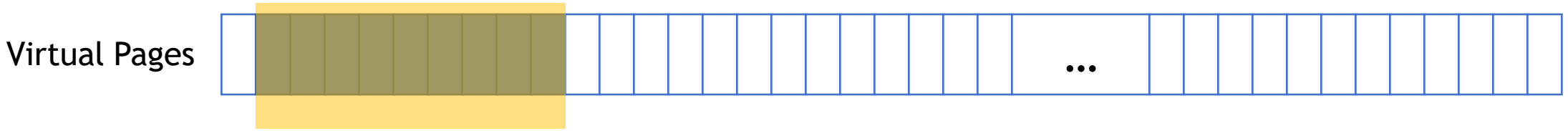
Spoiler: Finding Undocumented Aliasing

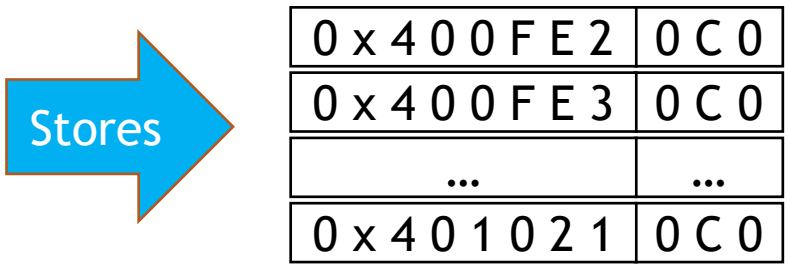


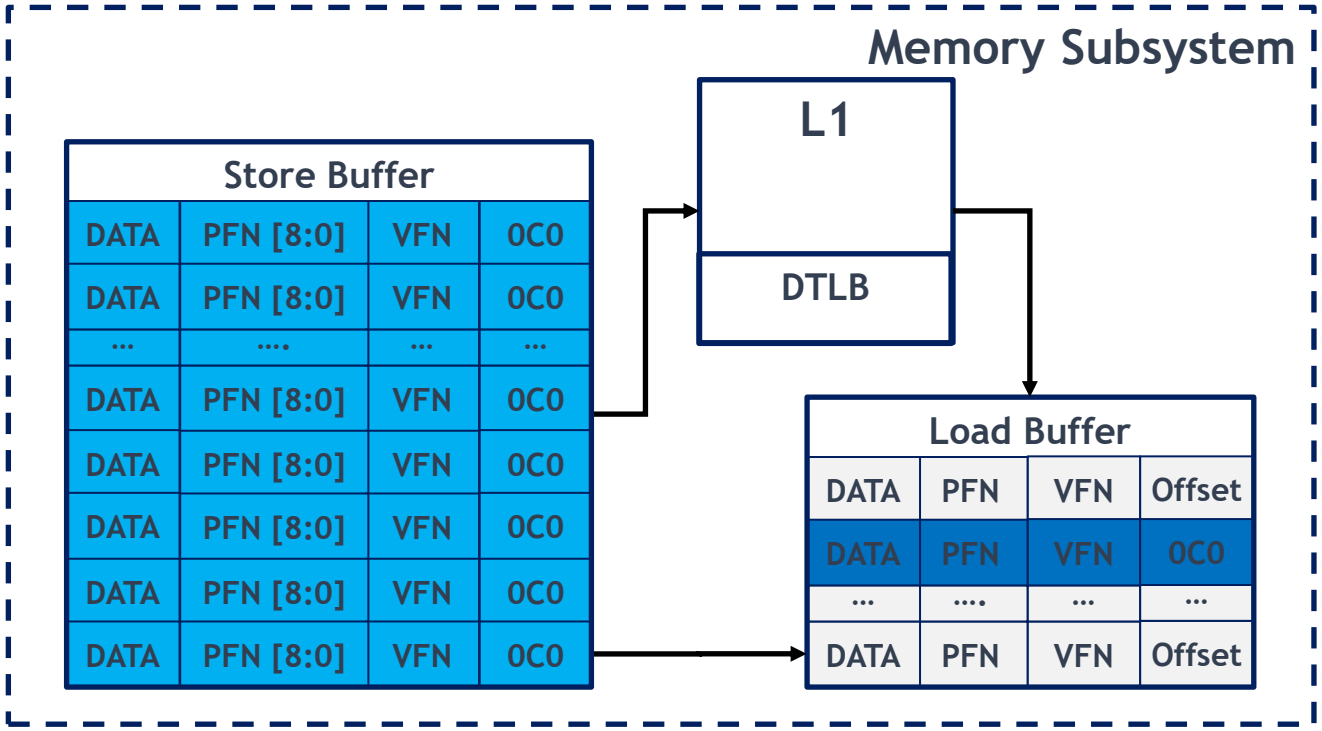
Spoiler: Finding Undocumented Aliasing



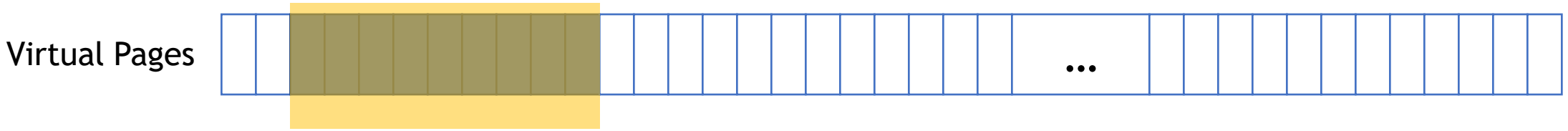
Spoiler: Finding Undocumented Aliasing

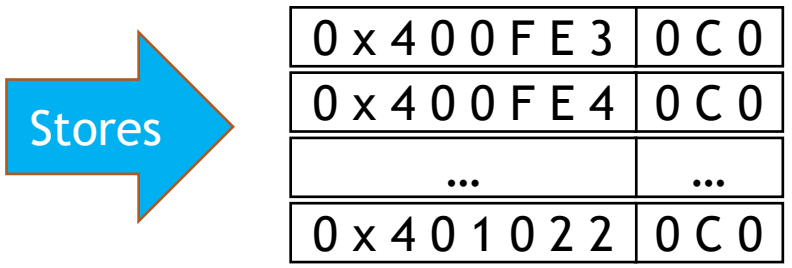


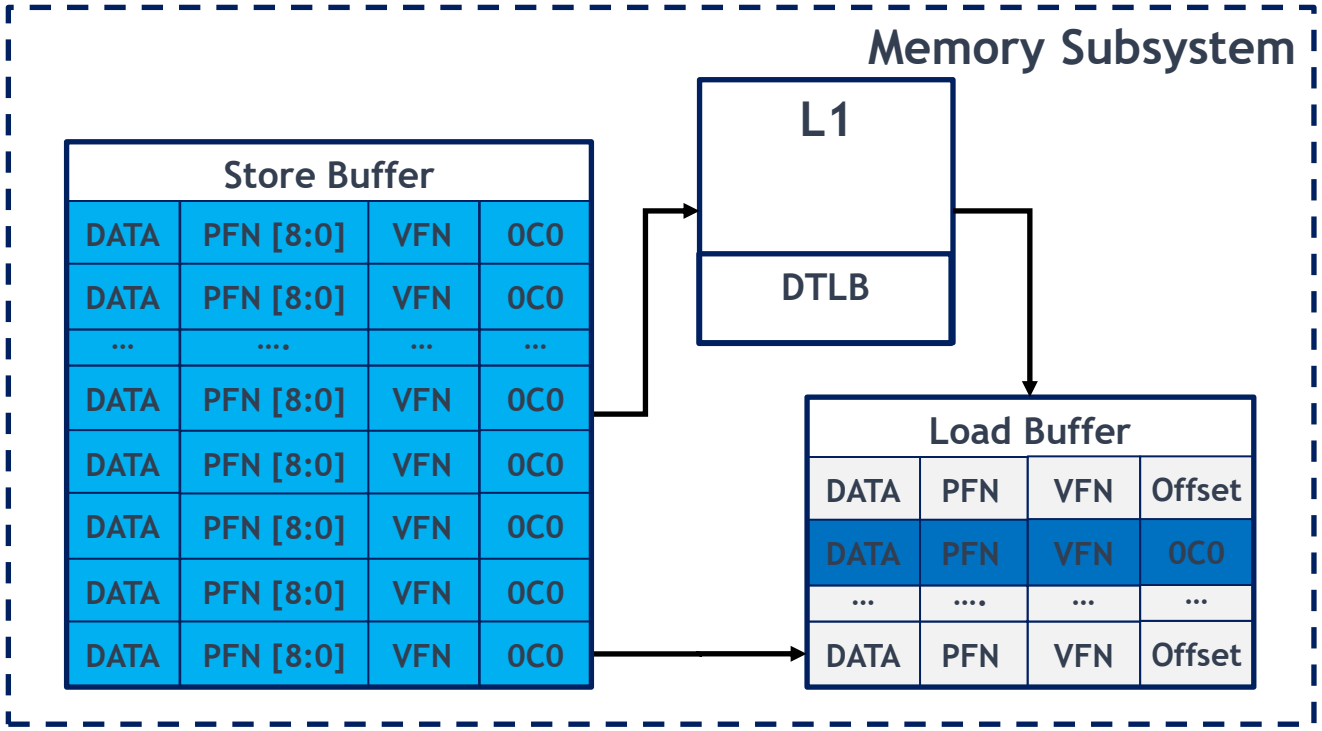




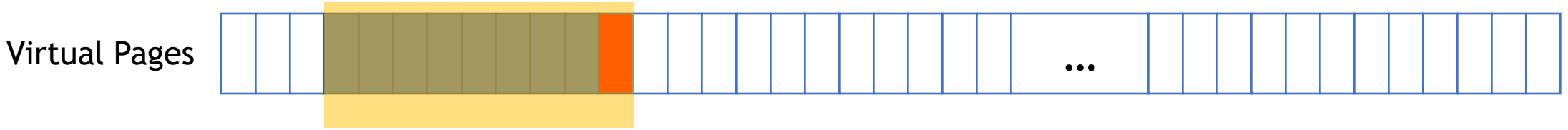
Spoiler: Finding Undocumented Aliasing

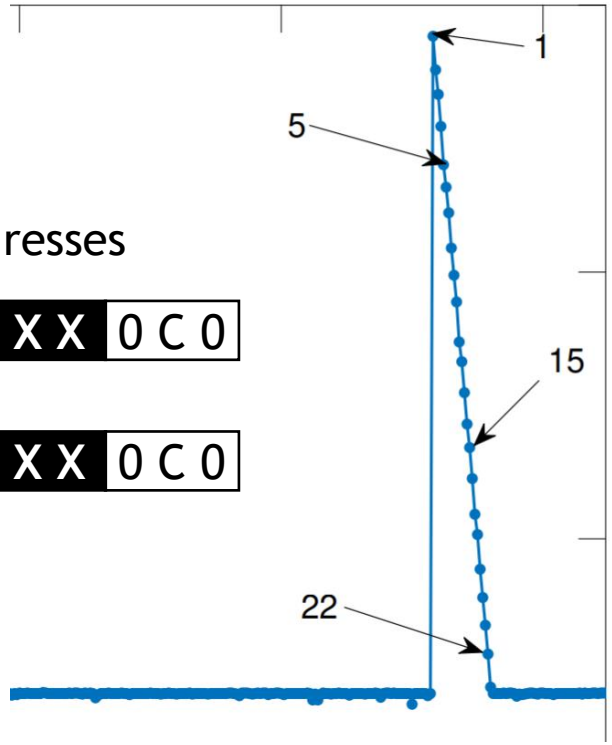






Spoiler: Finding Undocumented Aliasing

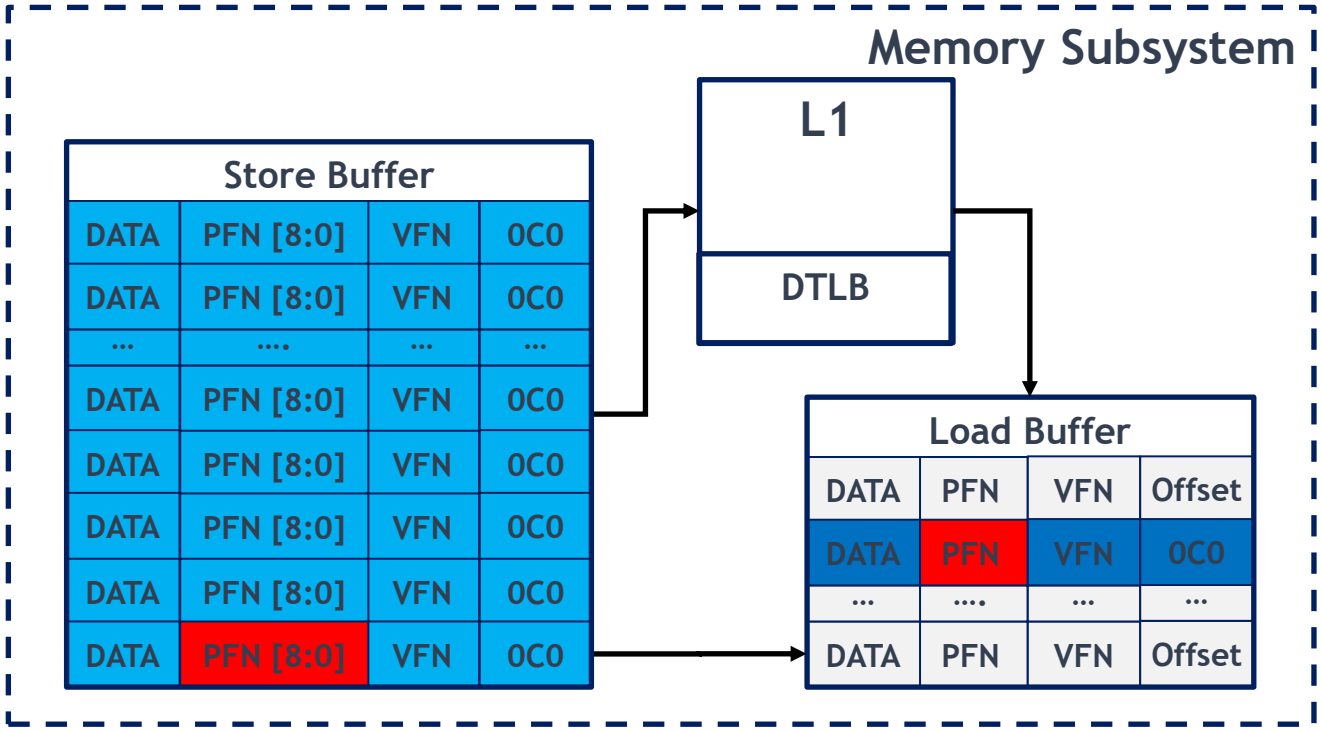




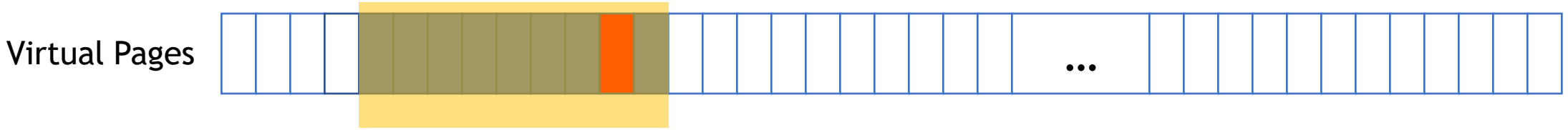
Physical Addresses

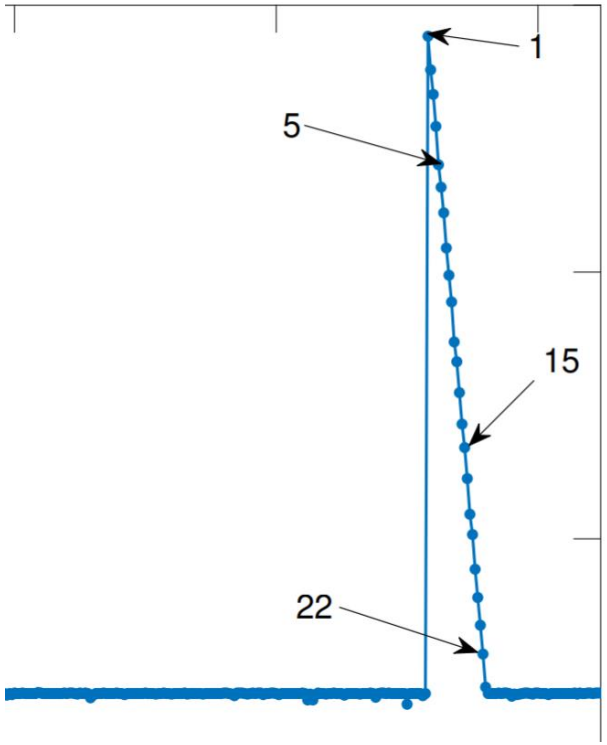
0 x 6 5 F 3 2 X X 0 C 0

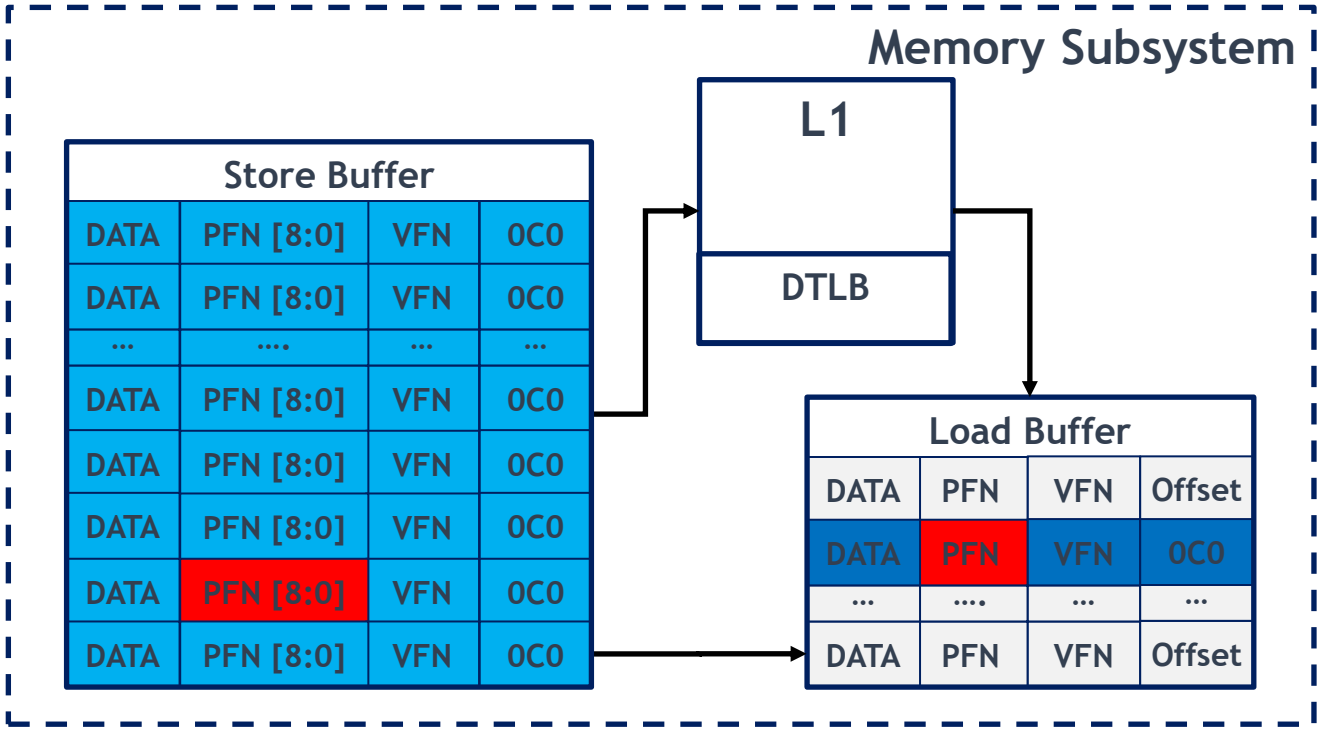
0 x 3 2 A C 2 X X 0 C 0



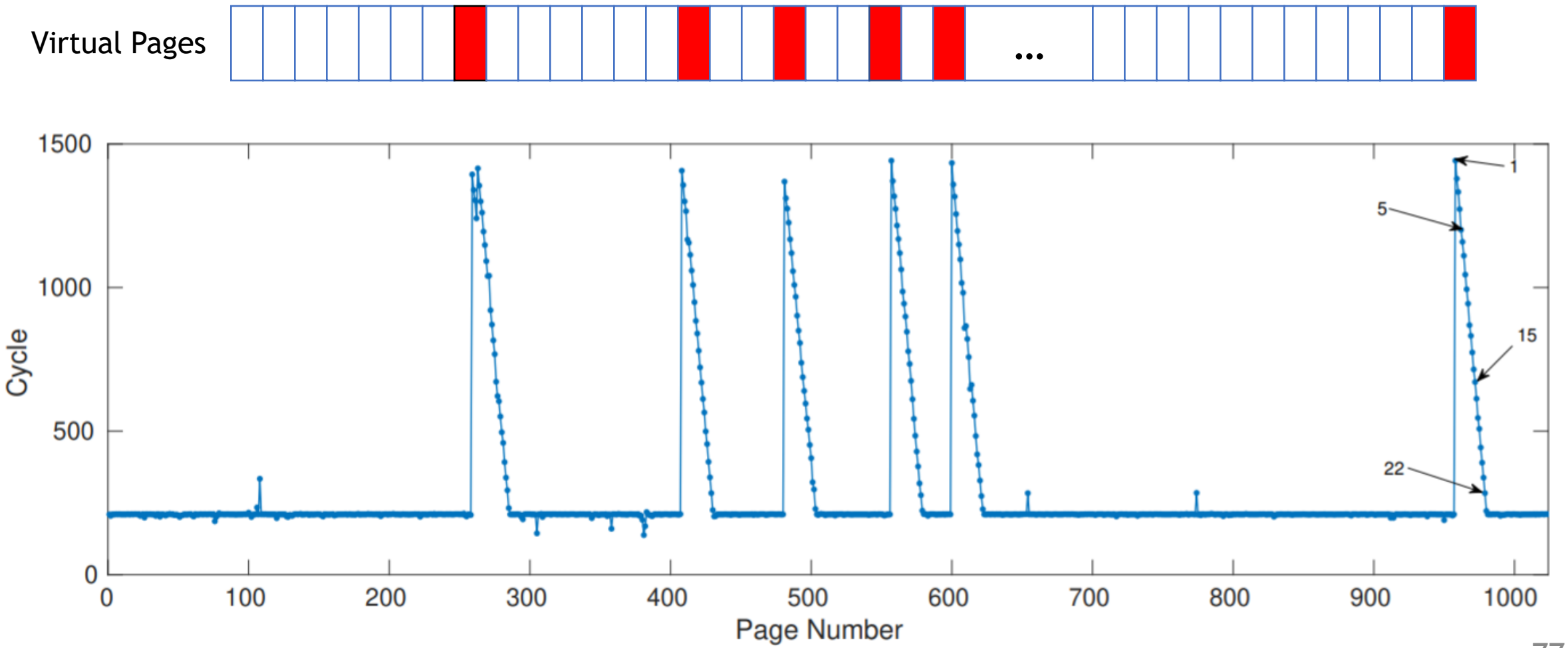
Spoiler: Finding Undocumented Aliasing



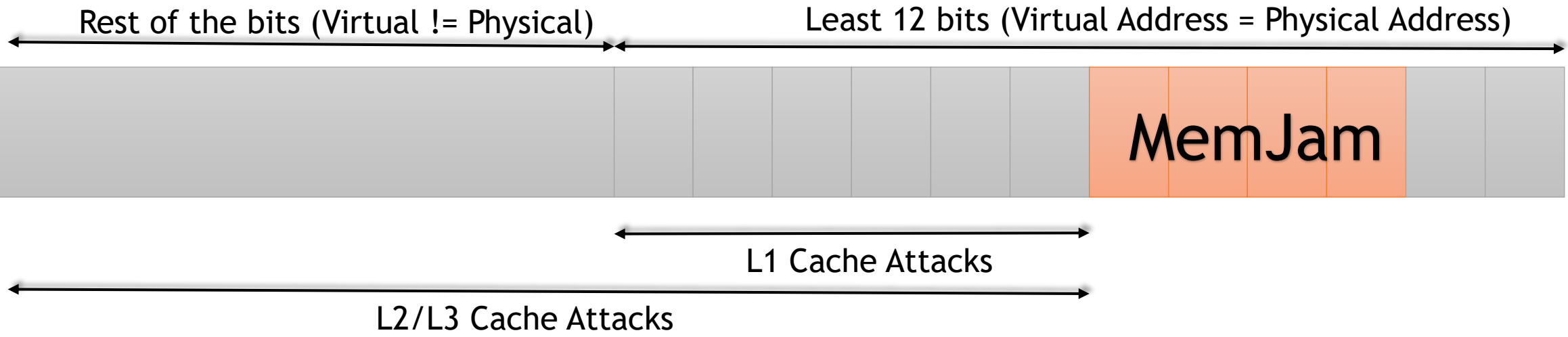




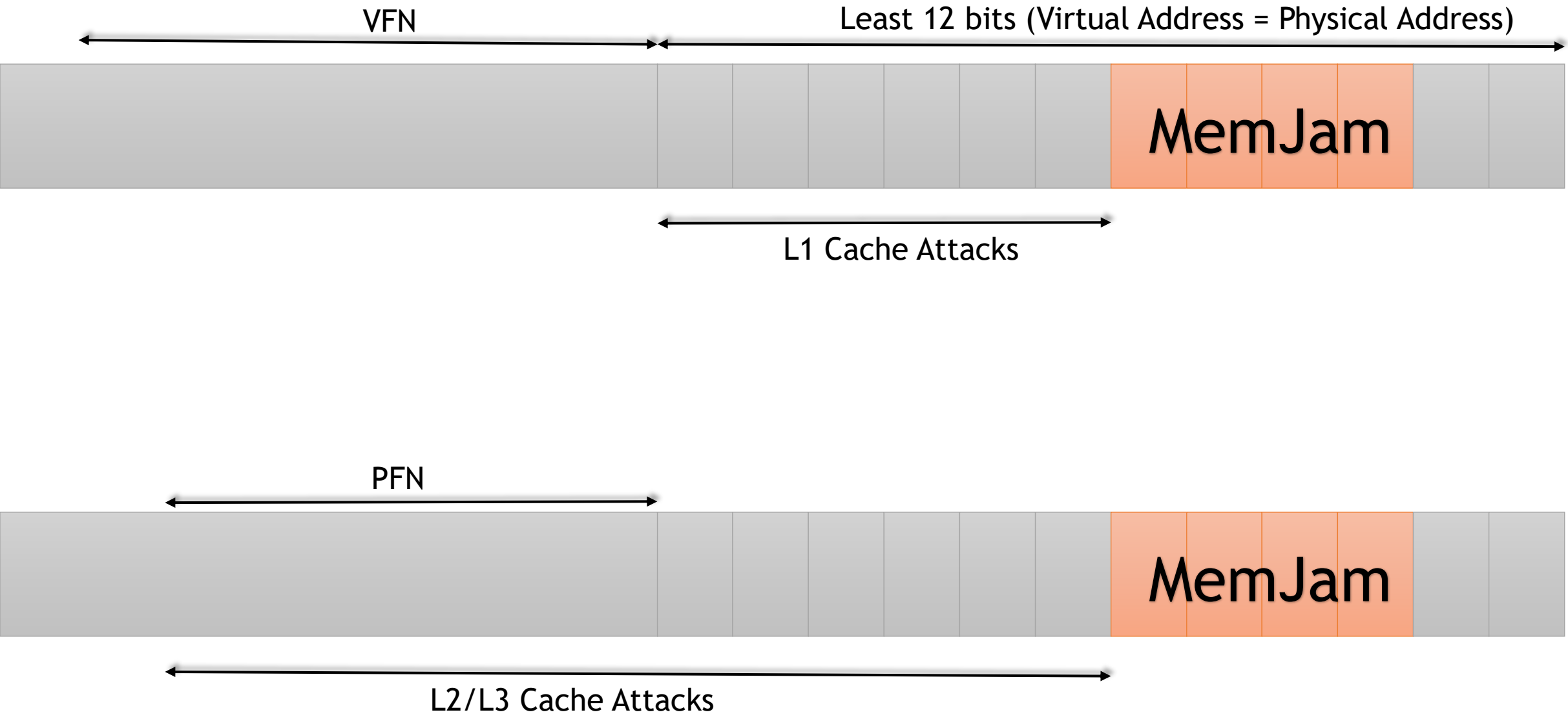
Spoiler: Finding Undocumented Aliasing



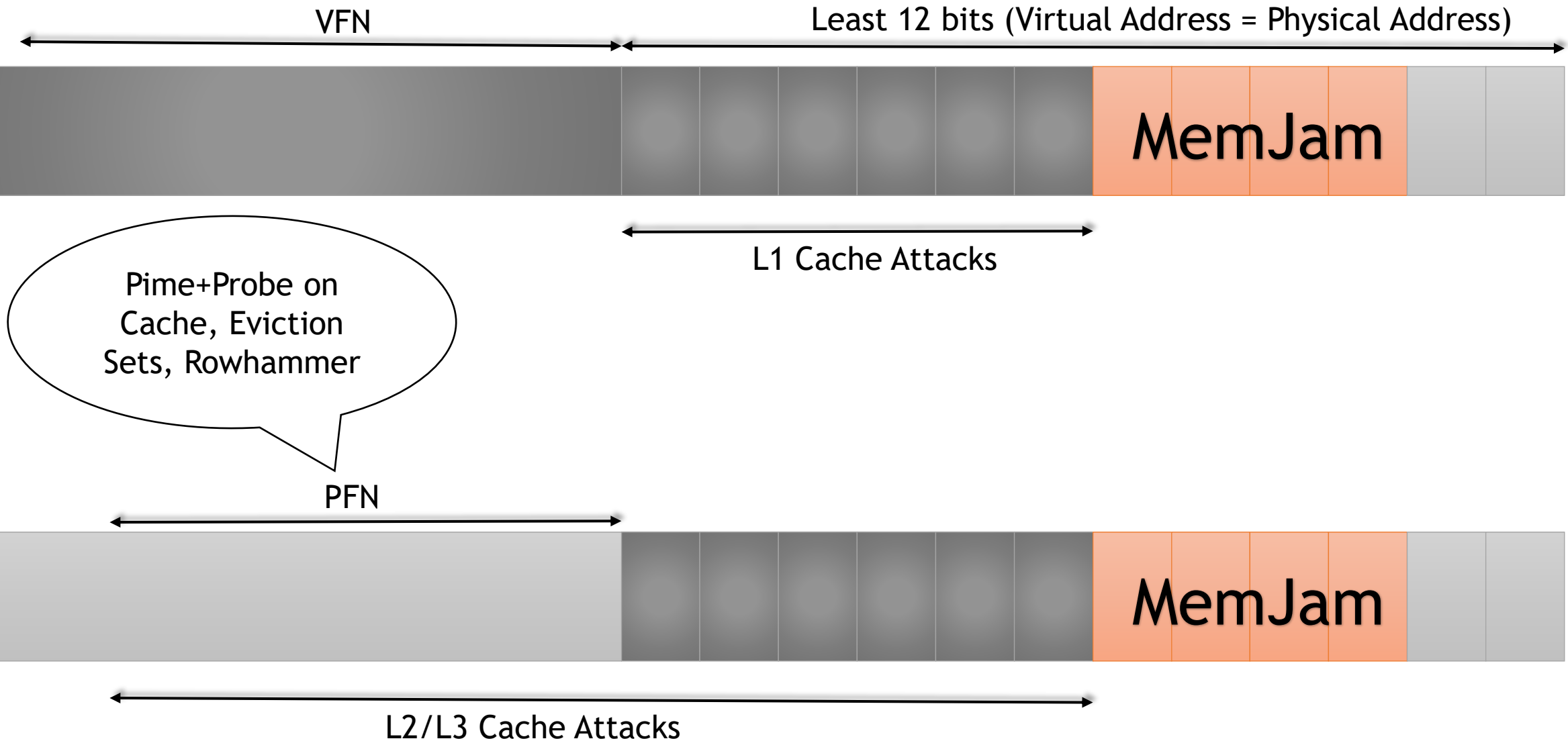
Spoiler: Learning on Physical Address Bits



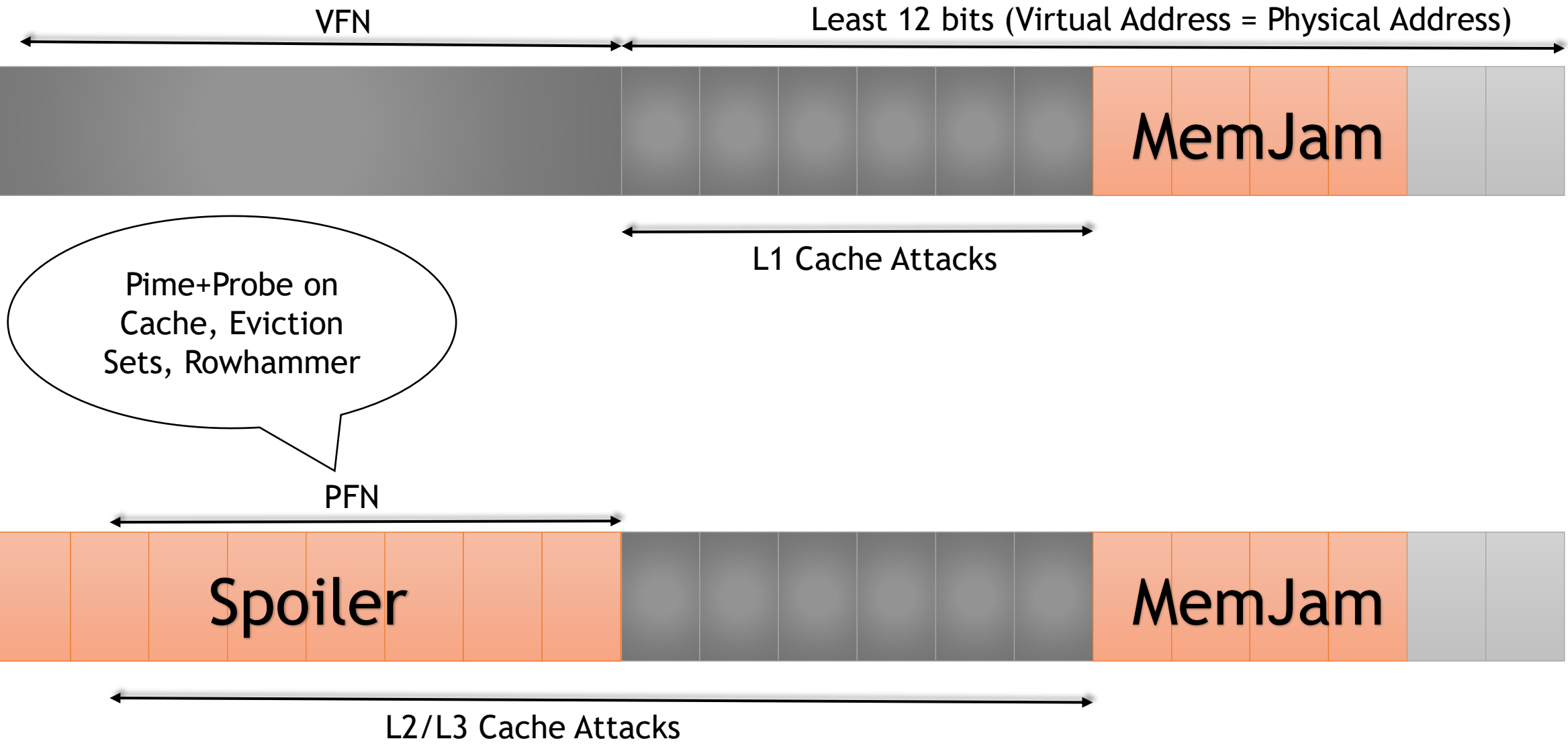
Spoiler: Learning on Physical Address Bits

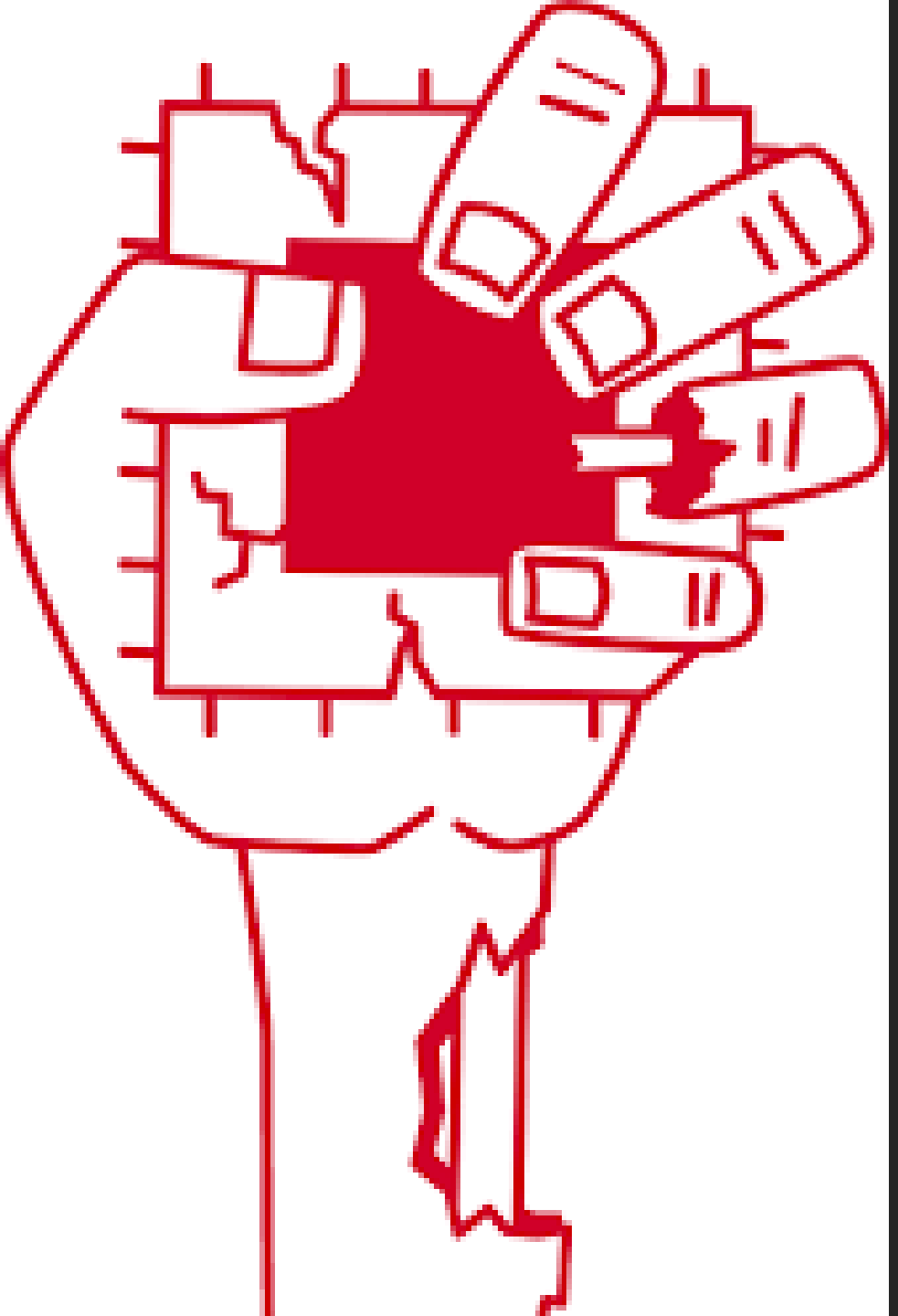


Spoiler: Learning on Physical Address Bits



Spoiler: Learning on Physical Address Bits





Microarchitectural Data Sampling

Microarchitecture Data Sampling (MDS)

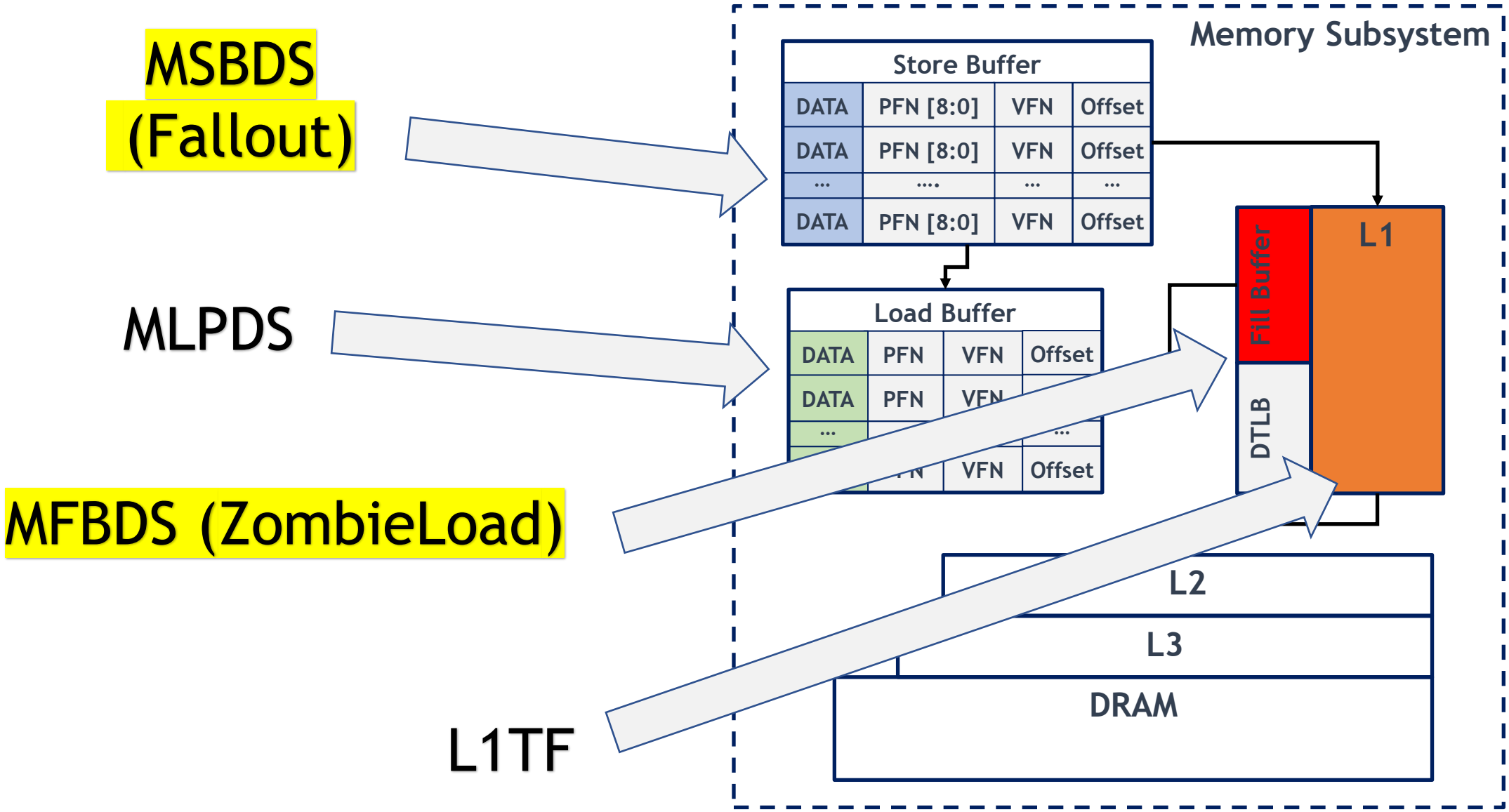
- Meltdown is fixed but we could steal leak data on the fixed CPU.

```
char secret = *(char *) 0xffffffffffffa0123;
```

whatever

- Which part of the CPU leak the data?!

CPU Memory Subsystem – Leaky Buffers



Microarchitecture Data Sampling (MDS)

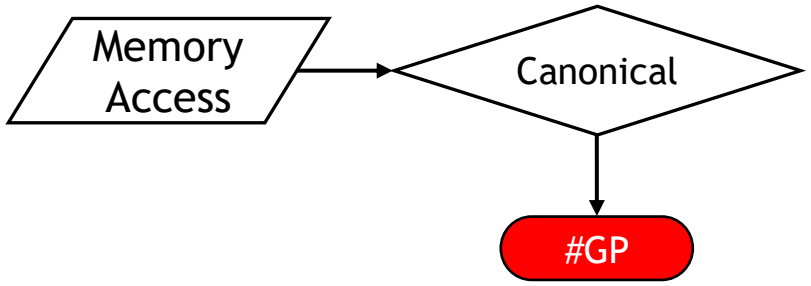
- Meltdown is fixed but we could steal leak data on the fixed CPU.

```
char secret = *(char *) 0xffffffffffffa0123;
```

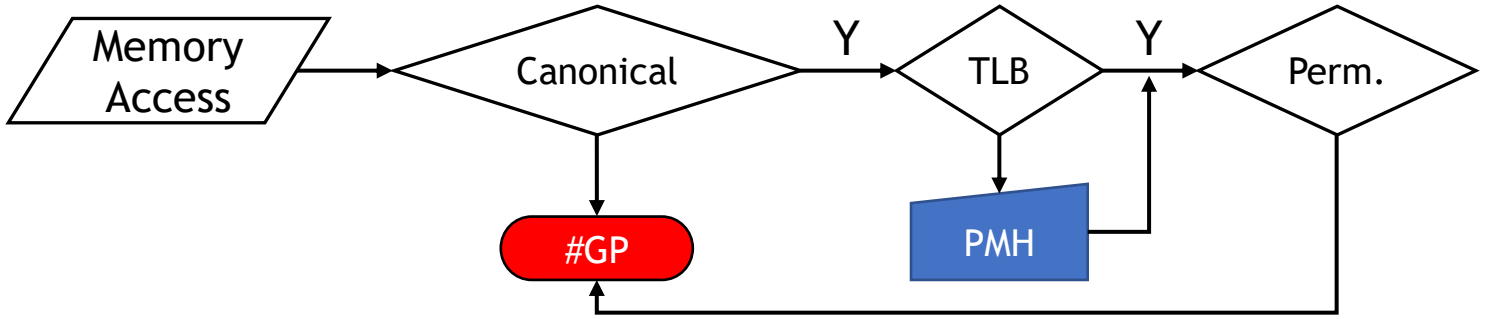
whatever

- Which part of the CPU leak the data?!
  - Store Buffer (Fallout)
  - Line Fill Buffer (ZombieLoad)

- What data is Leaked?
  - The data that is processed by a concurrent thread (hyperthreading)
  - The data from the previous process context



Virtual Address

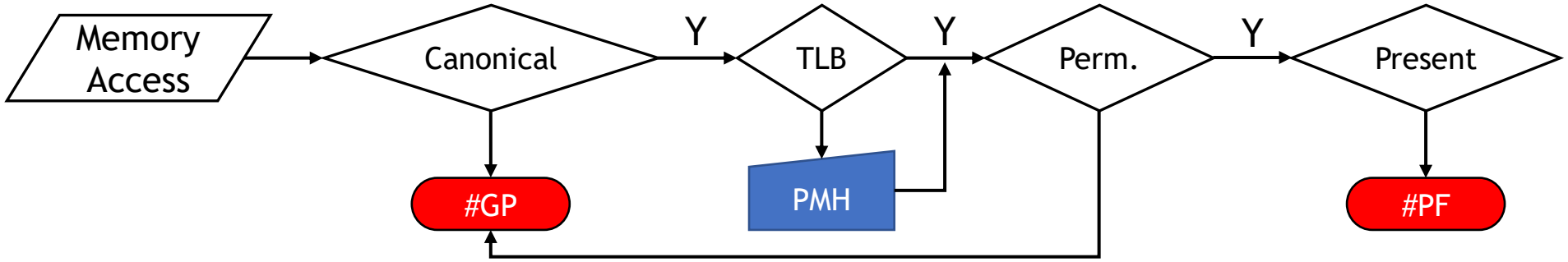


Virtual Address

VFN	Offset

PTE

P	RW	US	…	A	…	Physical Page Number	…

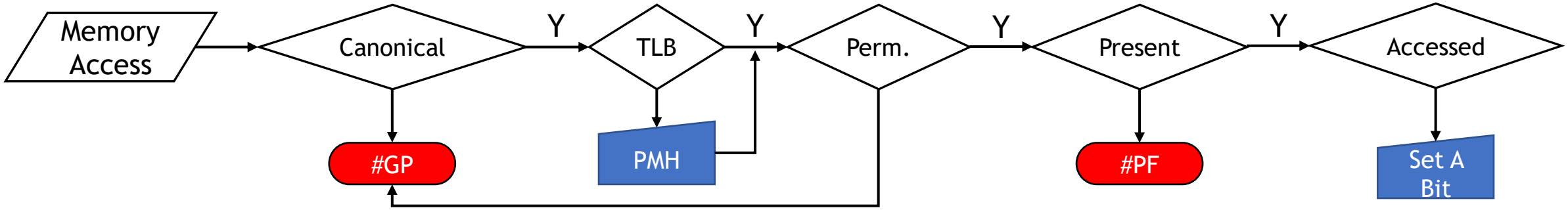


Virtual Address

VFN	Offset

PTE

P	RW	US	…	A	…	Physical Page Number	…

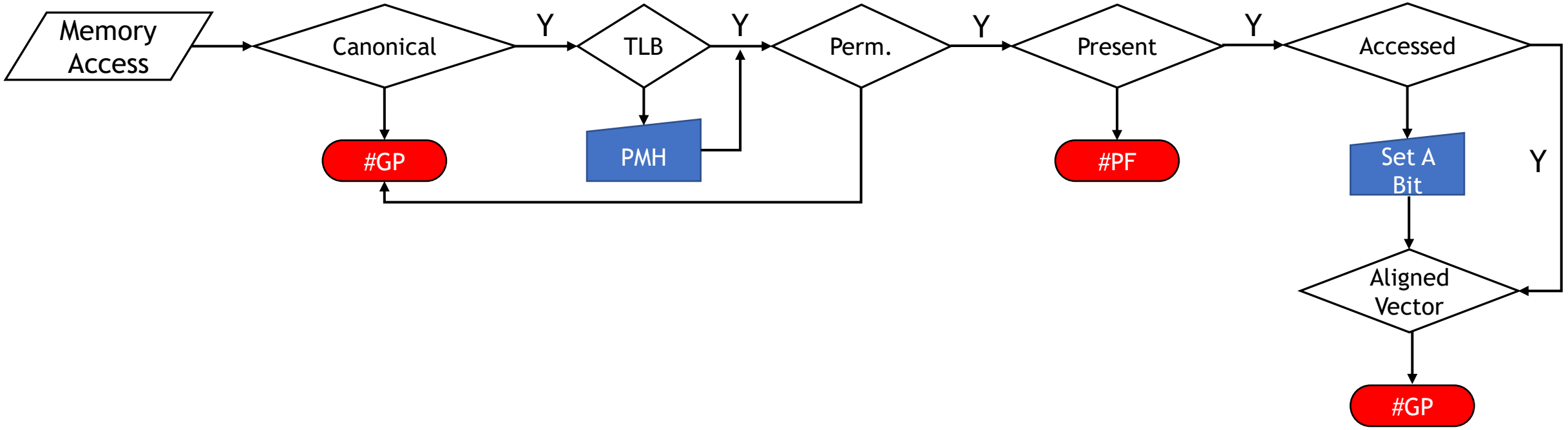


Virtual Address

VFN	Offset

PTE

P	RW	US	…	A	…	Physical Page Number	…

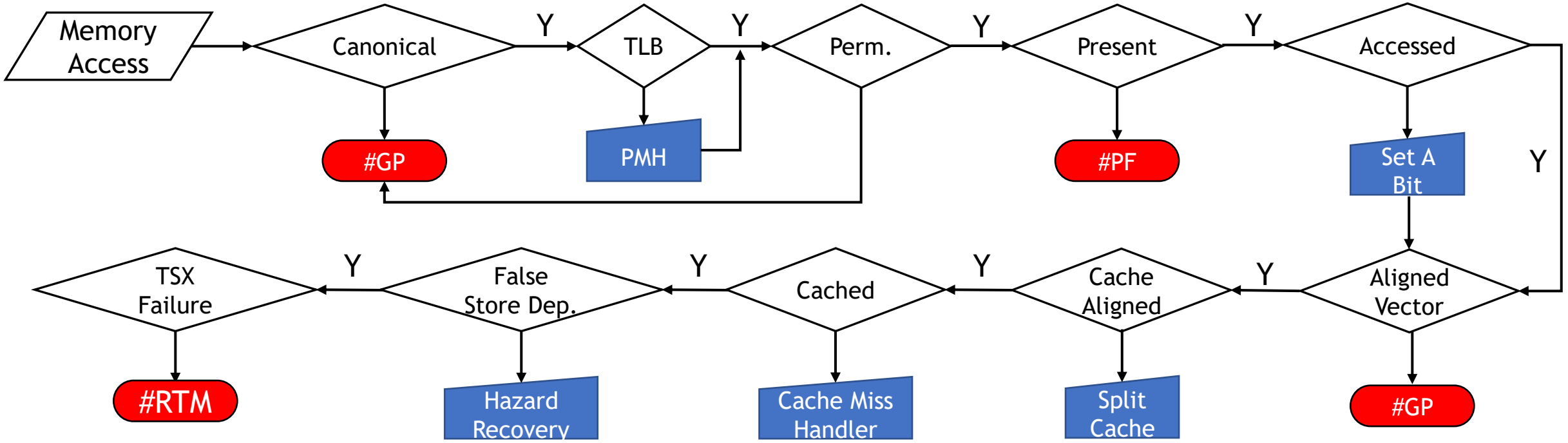


Virtual Address

VFN	Offset

PTE

P	RW	US	…	A	…	Physical Page Number	…



Virtual Address

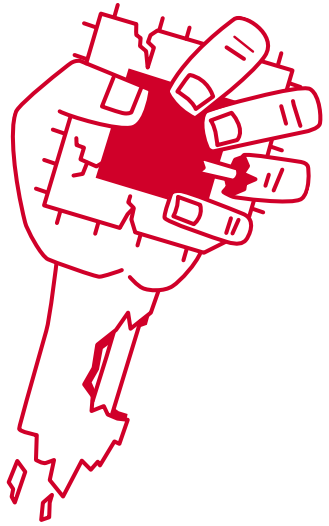
PTE

Challenges with MDS Testing?

- Reproducing attacks is not reliable. It may depend on:
  - State of the CPU's pipeline during execution
  - CPU configuration (generation, frequency, microcode patch, 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!!



Transynther

Transynther (Fuzzing-based Random MDS Testing)

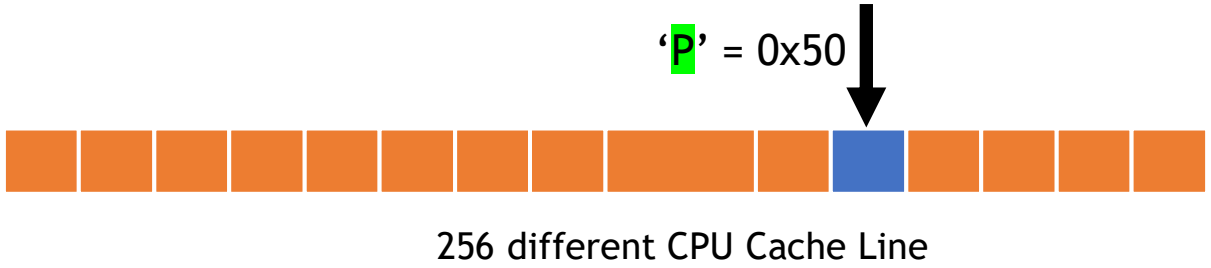
Step 1:

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

Step 2:

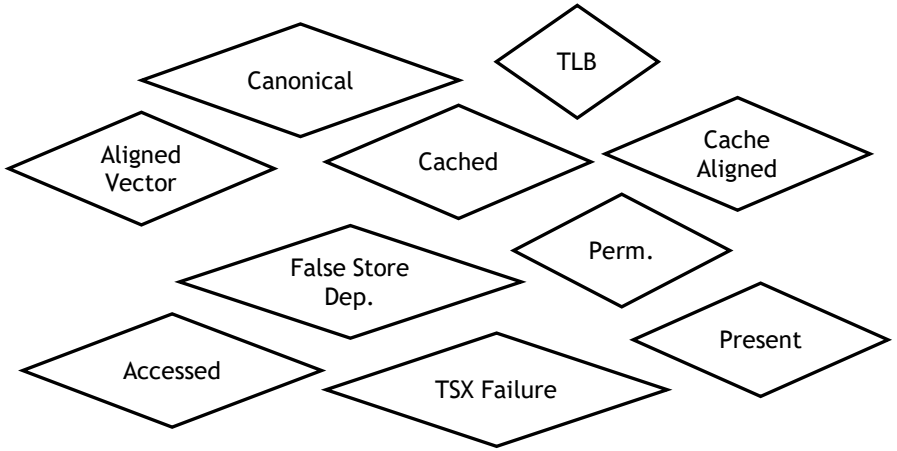
```
char x = oracle[secret * 4096];
```

Step 3:



Transynther (Fuzzing-based Random MDS Testing)

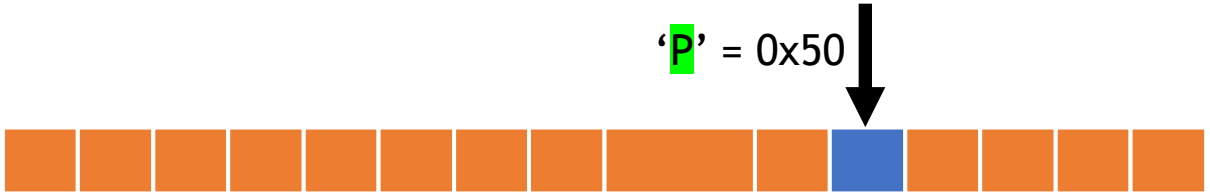
Step 1:



Step 2:

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

Step 3:

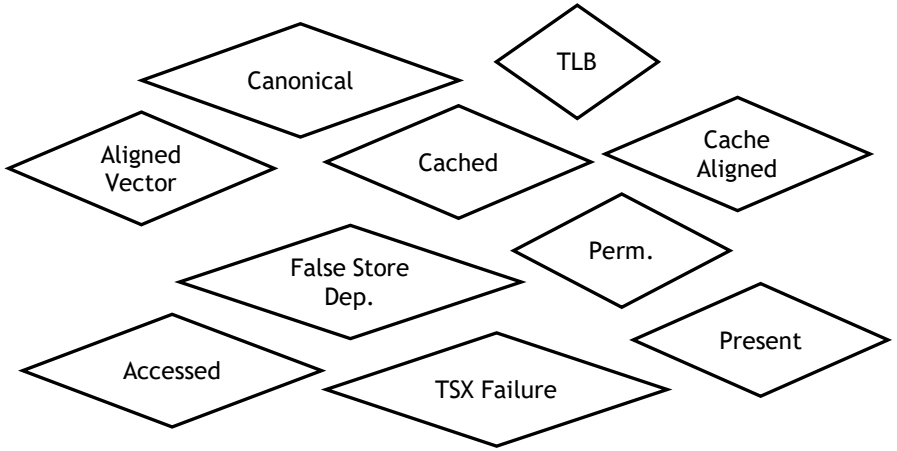


256 different CPU Cache Line

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

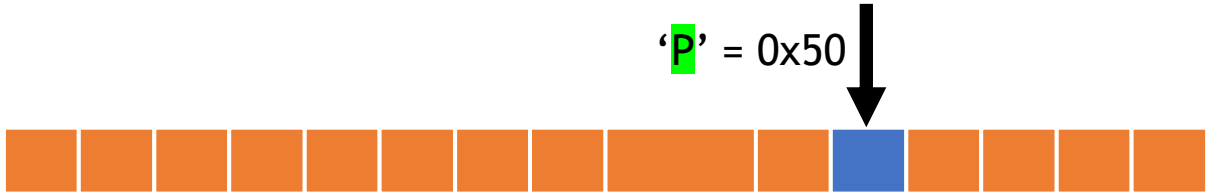


Step 1:

Step 2:

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

Step 3:



256 different CPU Cache Line

Transynther (Fuzzing-based Random MDS Testing)

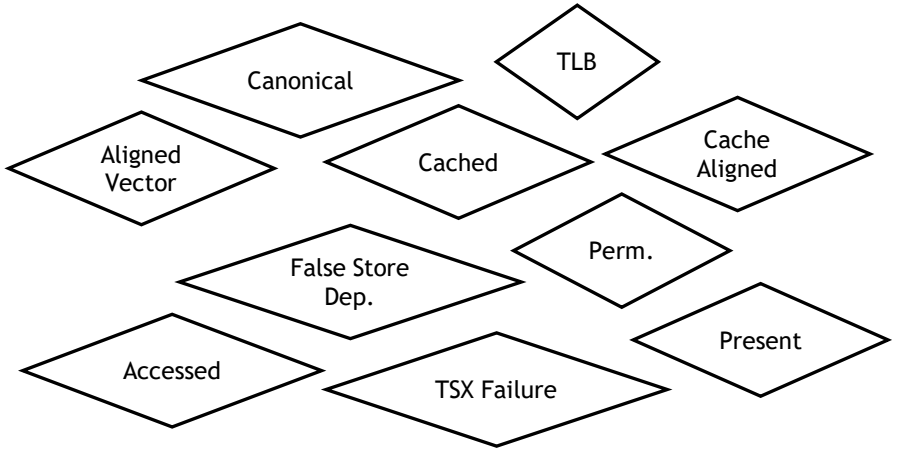
Step 0: Buffer Grooming

Stores Same Thread: 0x41424344

Loads Same Thread: 0x51525354

Stores Hyper Thread: 0x61626364

Loads Hyper thread Thread: 0x71727374



Random instructions

Step 1:

Step 2:

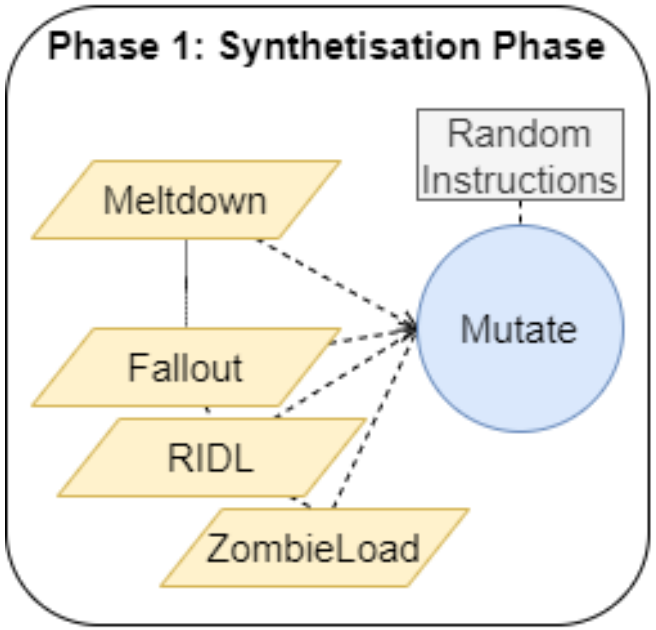
```
char x = oracle[secret * 4096];
```

Step 3:

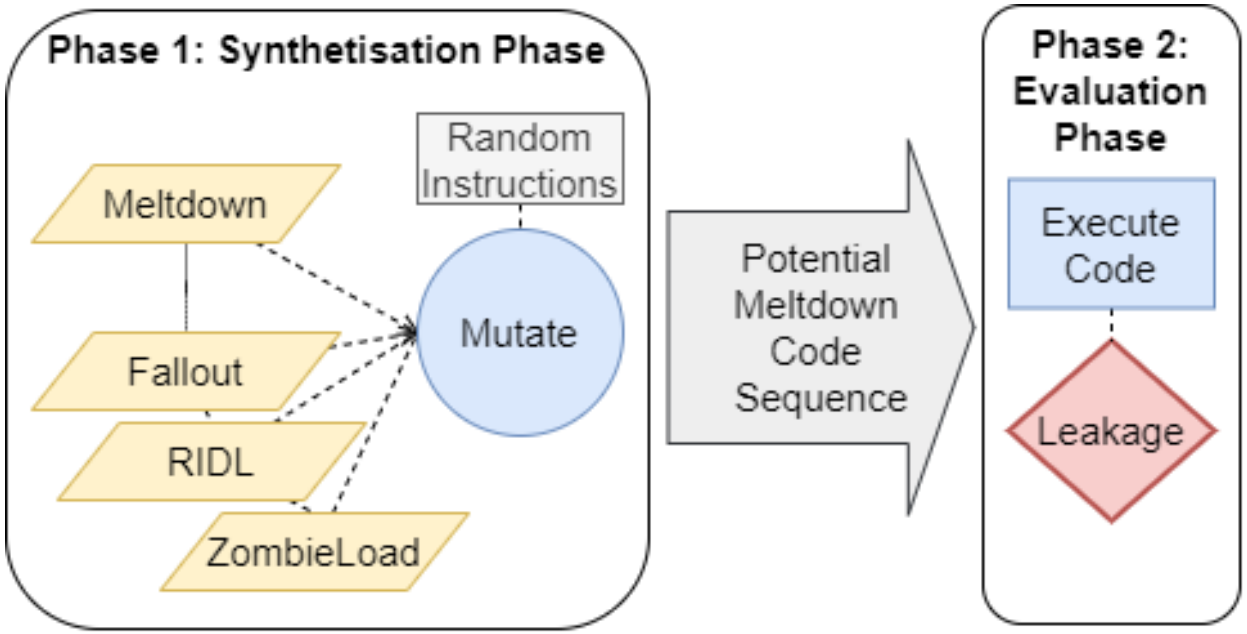
'P' = 0x50

256 different CPU Cache Line

Transynther (Fuzzing-based MDS Testing)



Transynther (Fuzzing-based MDS Testing)



Transynther (Fuzzing-based MDS Testing)

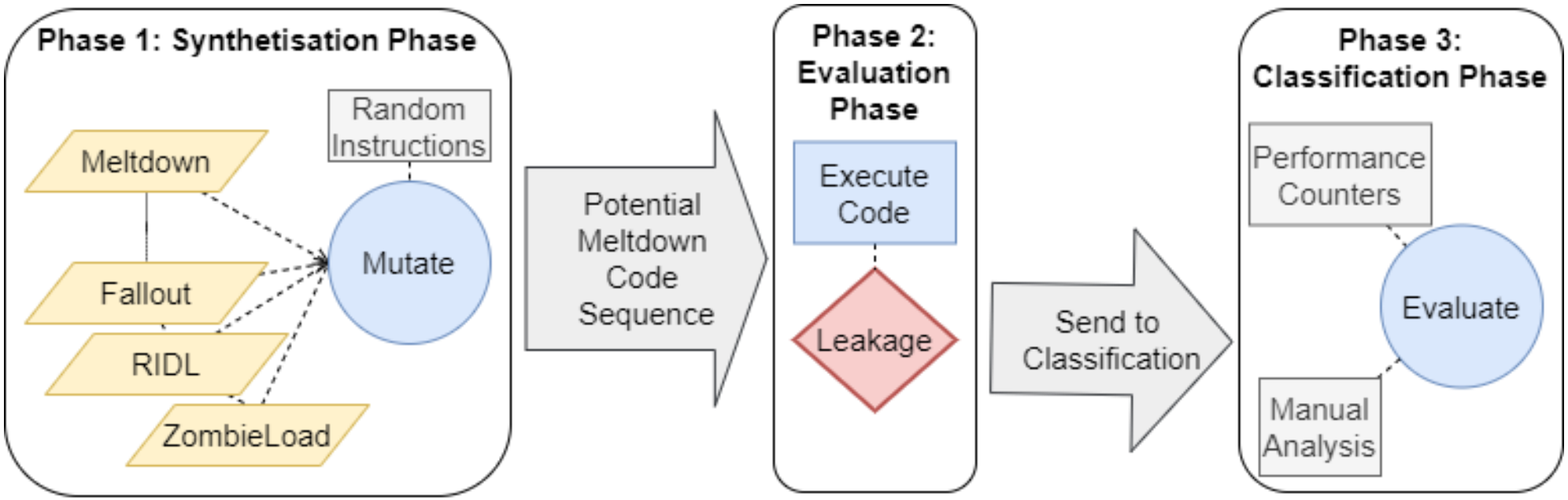


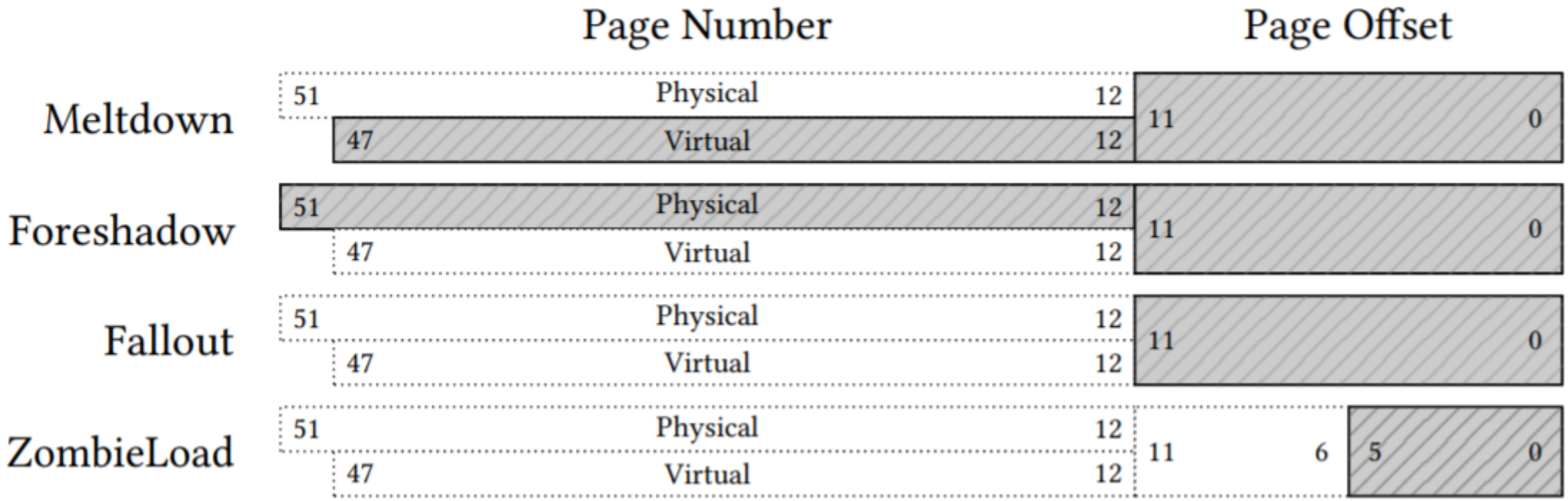
Table 2: Leakage variants discovered by Transynther.

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


MEDUSA

Meltdown-style Attacks



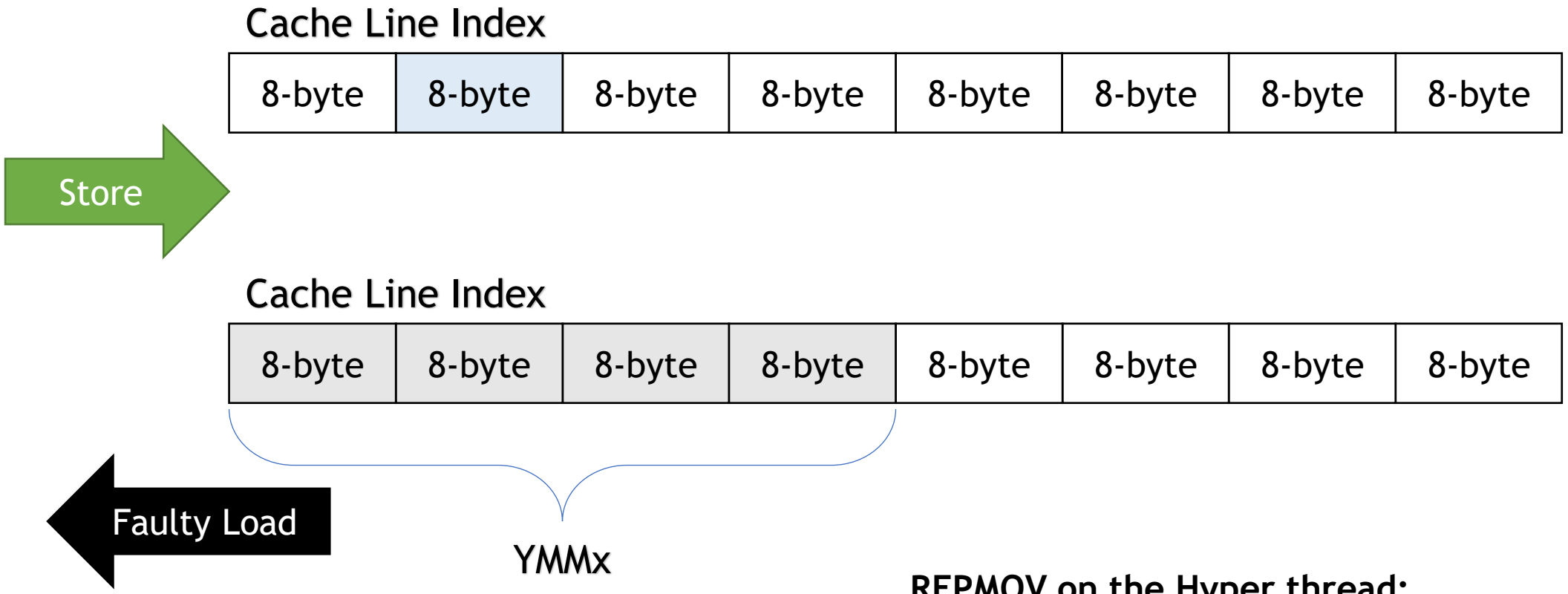
Medusa Attack

- Medusa partially solves the problem with ZombieLoad
  - Prefiltered data leakage
  - Less noise than ZombieLoad
- Medusa only leaks the write combining data.

Medusa Attack

- Medusa partially solves the problem with ZombieLoad
  - Prefiltered data leakage
  - Less noise than ZombieLoad
- Medusa only leaks the write combining data.
- Implicit WC, i.e., ‘rep mov’, ‘rep sto’, can be leaked.
  - Memory Copy Routines
  - File IO
- Served by a Write Combining Buffer (or just the Fill Buffer).
- Three variants
  - Based on different ways of massaging the microarchitecture

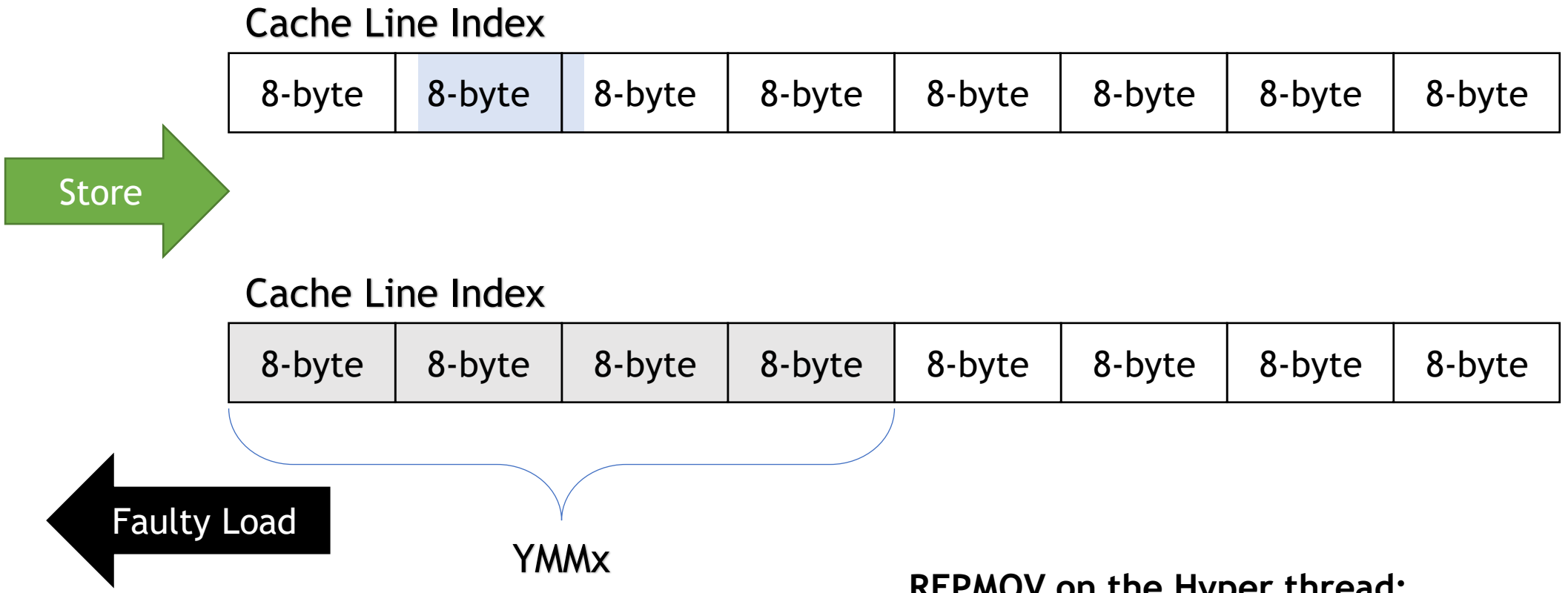
Medusa Attack – V2 Unaligned S2L Forwarding



REPMOV on the Hyper thread:

ABCDEFGH IJKLMNOP QRSTUVWX YZ…

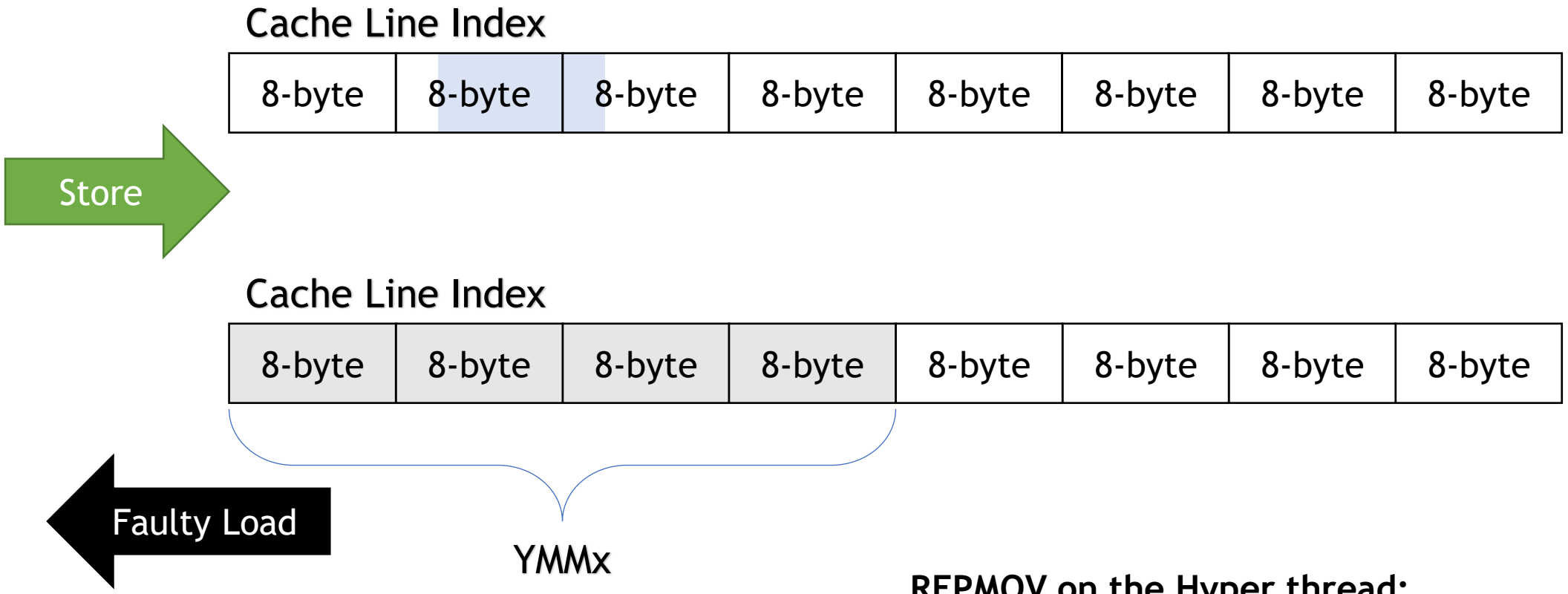
Medusa Attack – V2 Unaligned S2L Forwarding



REPMOV on the Hyper thread:

ABCDEFGH IJKLMNOP QRSTUVWX YZ…

Medusa Attack – V2 Unaligned S2L Forwarding



REPMOV on the Hyper thread:

ABCDEFGH IJKLMNOP QRSTUVWX YZ…

OpenSSL RSA Key Recovery

- OpenSSL Base64 Decoder uses inline Memcpy(-oS)
- Triggered 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-----
```

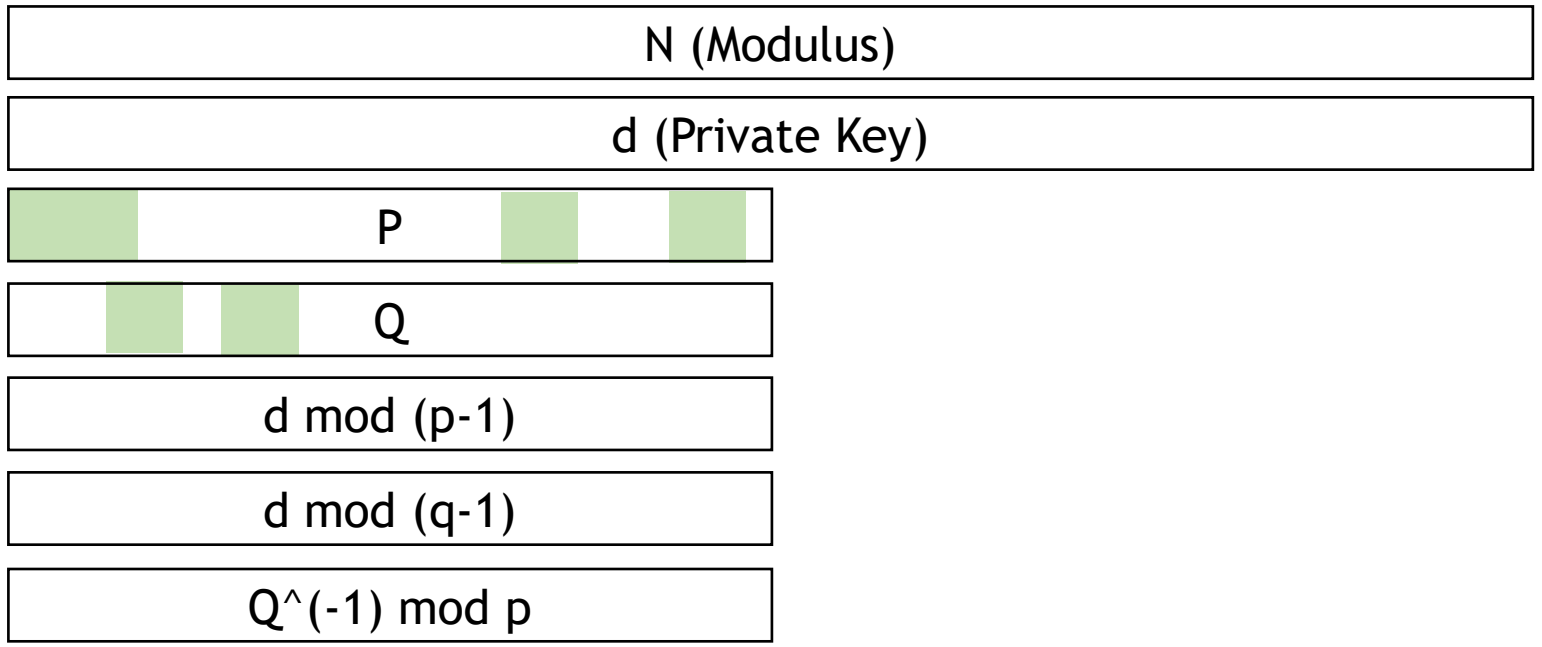
OpenSSL RSA Key Recovery

- OpenSSL Base64 Decoder uses inline Memcpy(-oS)
- Triggered 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-----
```

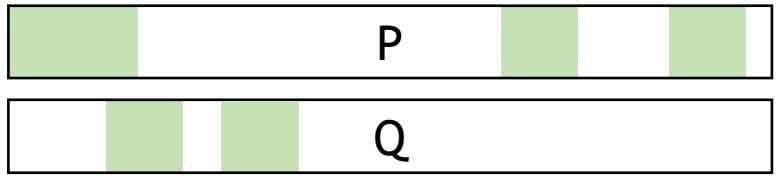
OpenSSL RSA Key Recovery

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



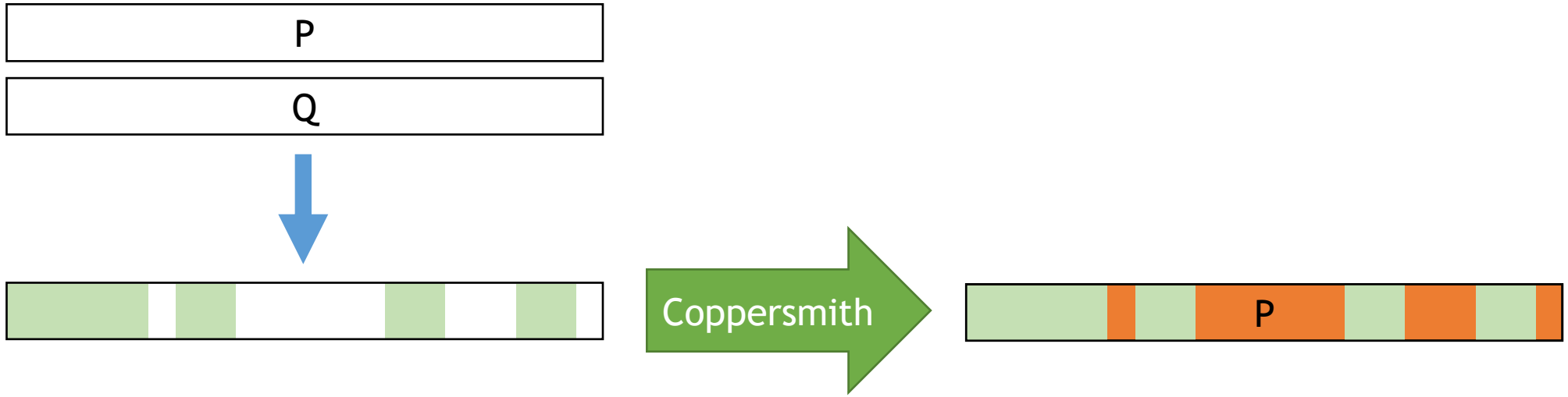
OpenSSL RSA Key Recovery - Coppersmith

- 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



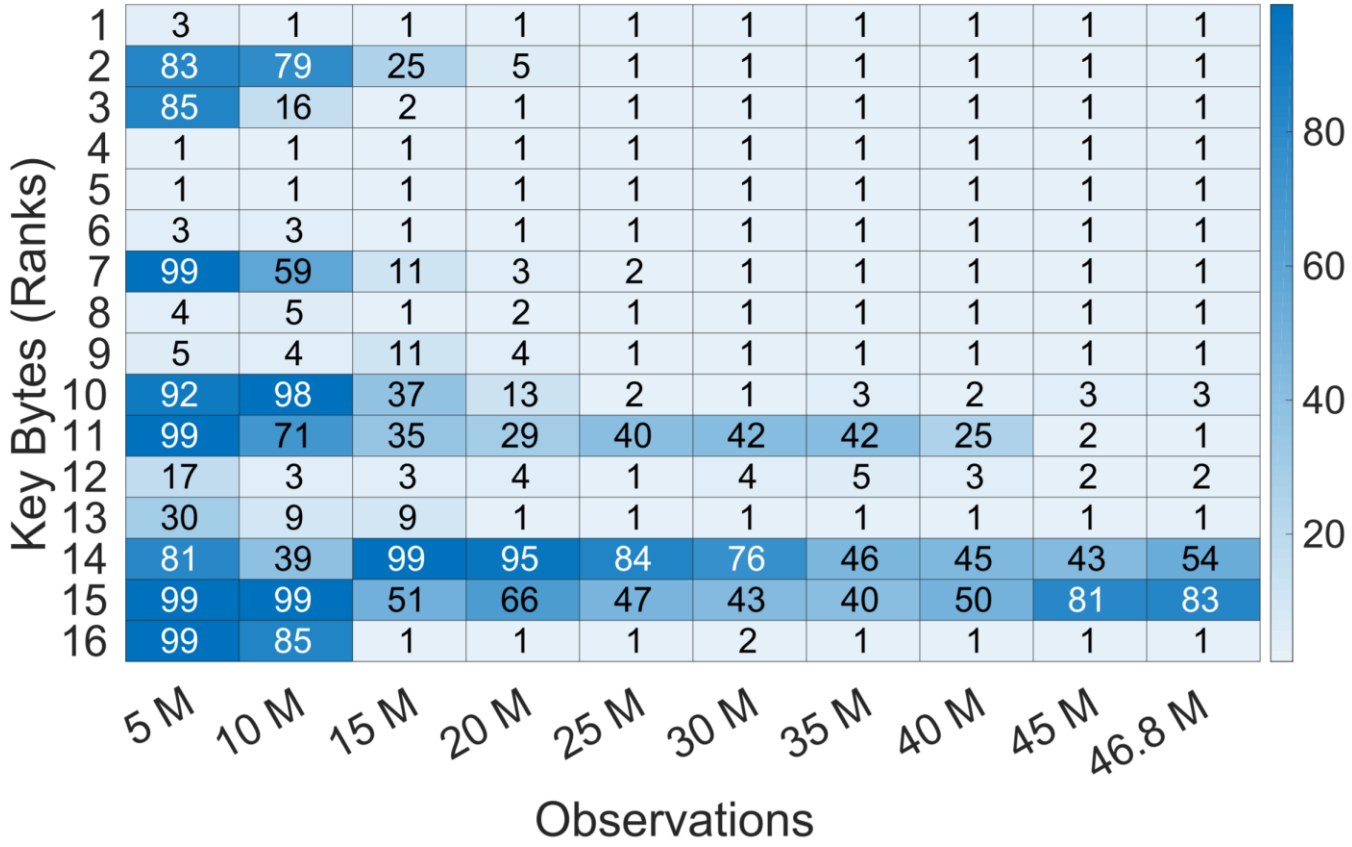
OpenSSL RSA Key Recovery - Coppersmith Attack

- 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.

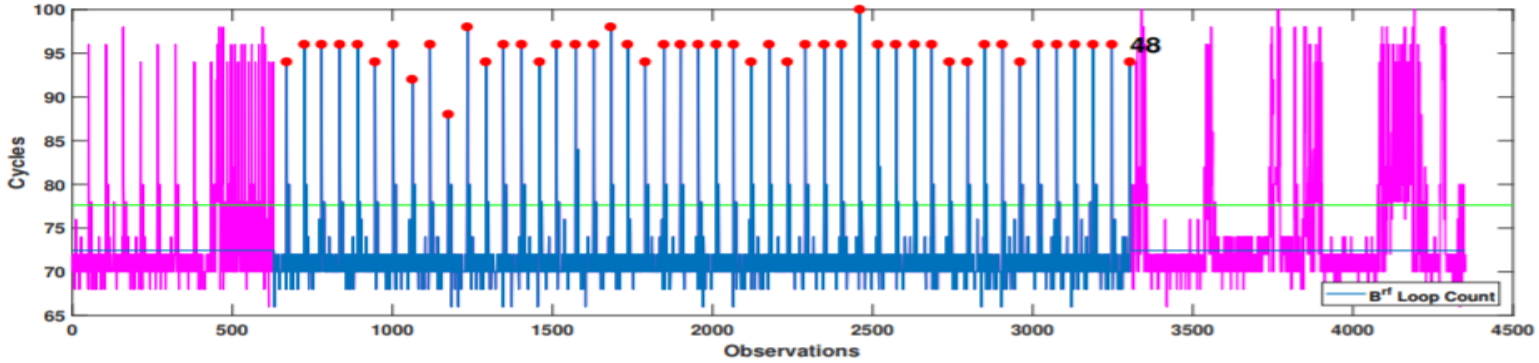


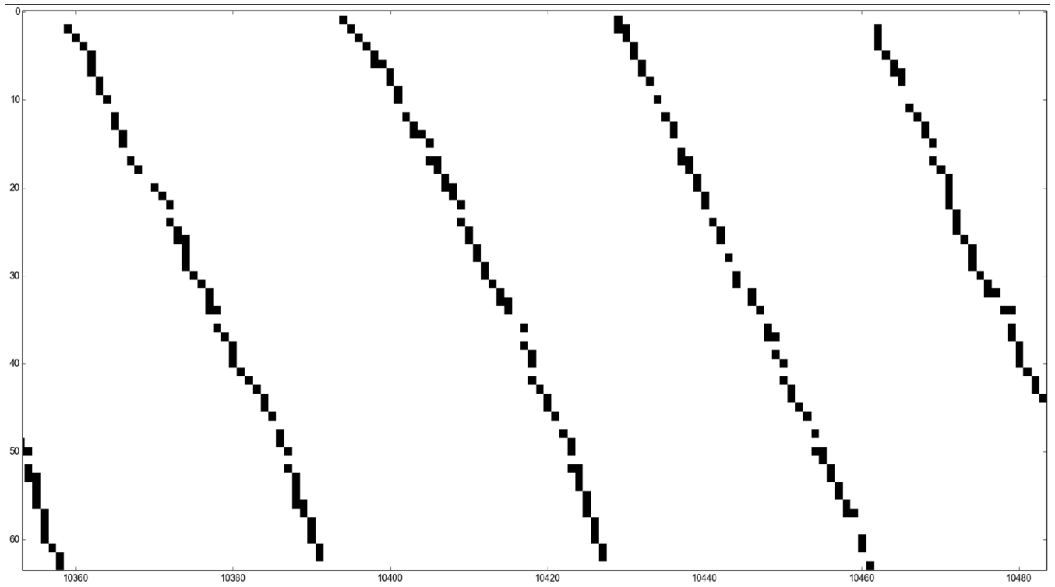
System-level Threat to Trusted Execution Environments (T2)

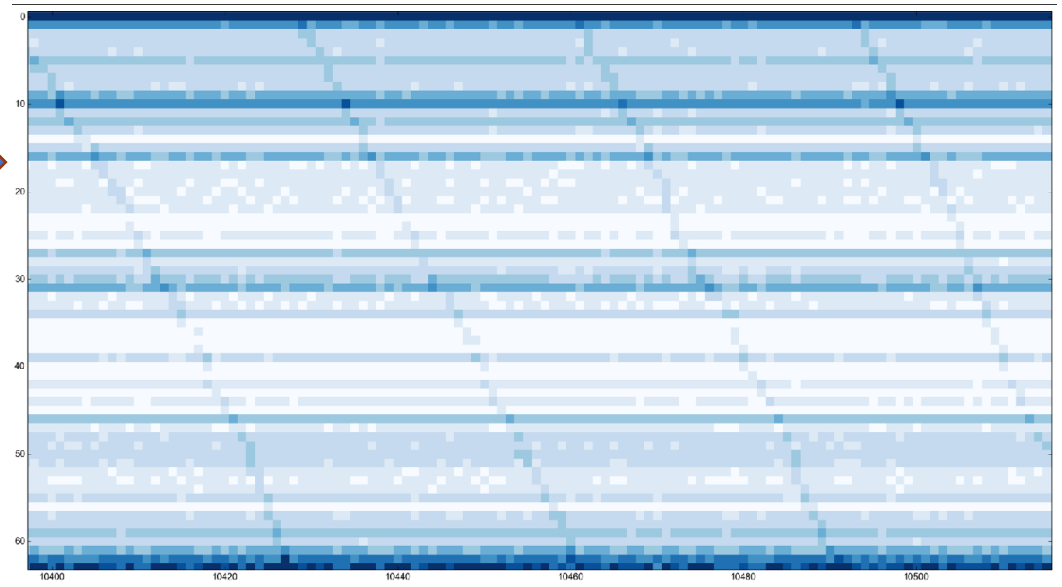
MemJam Attack on SGX



CacheZoom and CacheQuote

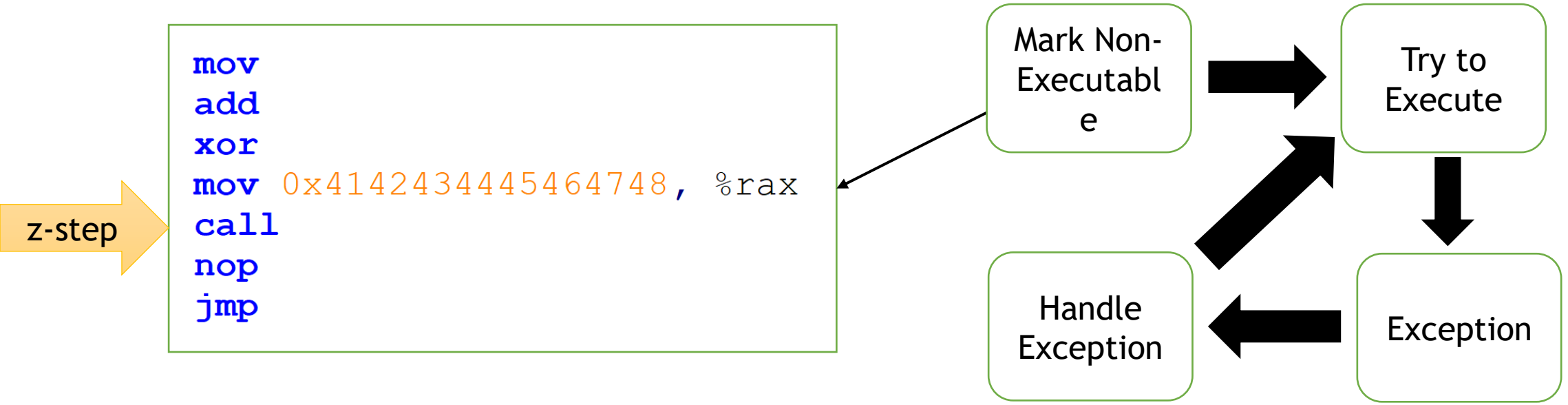






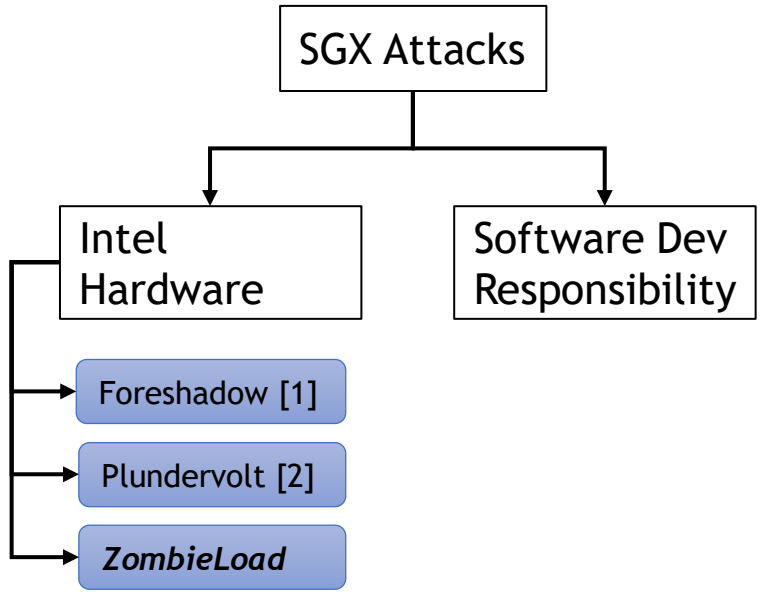
ZombieLoad - Recovering Intel SGX Sealing Key

- We can read register values of a trusted enclave by combining
  - ZombieLoad
  - NX Zero Stepping



Intel SGX Attack Taxonomy

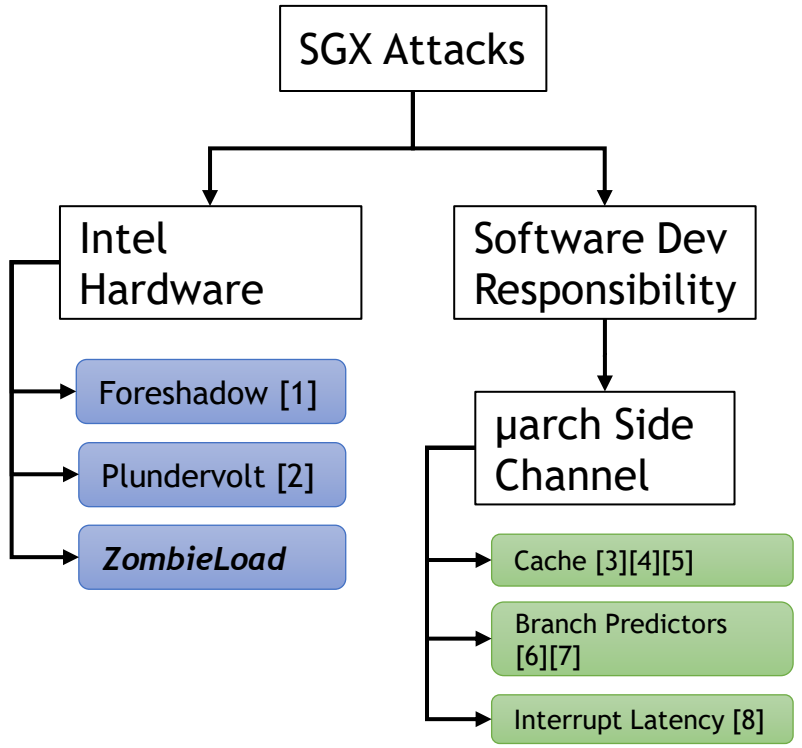
- Intel's Responsibility
  - Microcode Patches / Hardware mitigation
  - TCB Recovery
  - Hyperthreading is out
    - Remote Attestation Warning



[1] Van Bulck et al. "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution." USENIX Security 2018.
[2] Murdock et al. "Plundervolt: Software-based fault injection attacks against Intel SGX." IEEE S&P 2020.

Intel SGX Attack Taxonomy

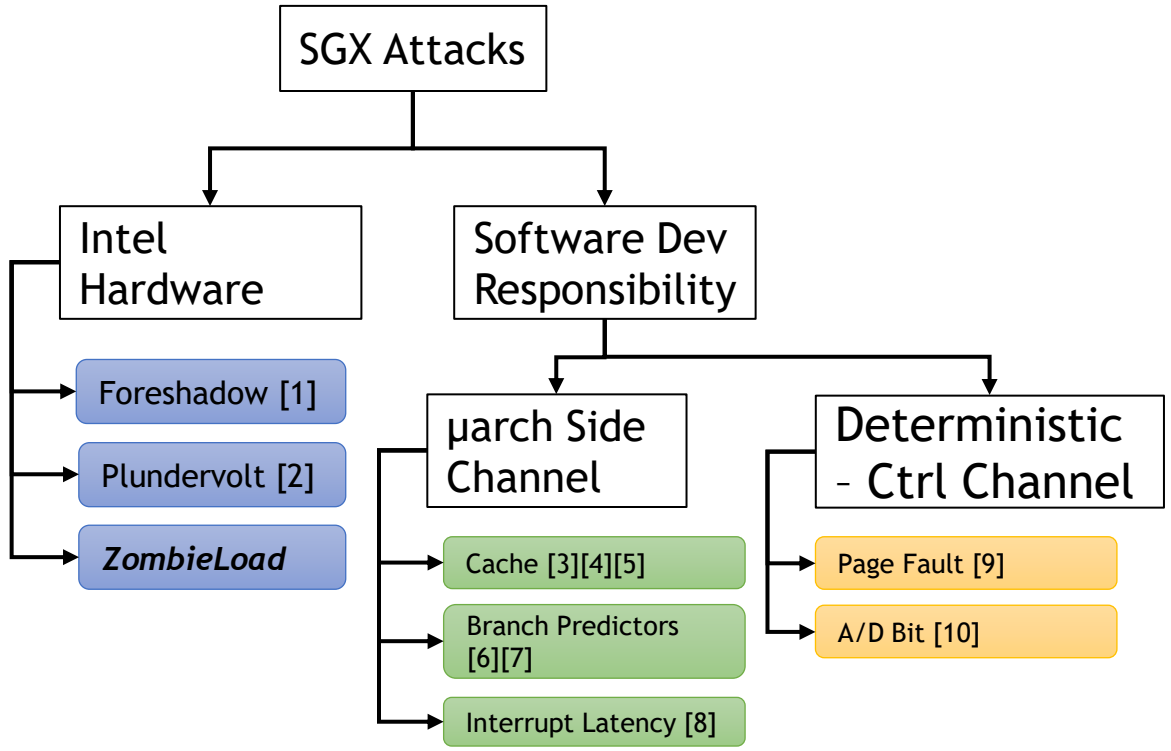
- Intel's Responsibility
  - Microcode Patches / Hardware mitigation
  - TCB Recovery
  - Hyperthreading is out
    - Remote Attestation Warning
- µarch Side Channel
  - Constant-time Coding
  - Flushing and Isolating buffers
  - Probabilistic



[1] Van Bulck et al. "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution." USENIX Security 2018.
[2] Murdock et al. "Plundervolt: Software-based fault injection attacks against Intel SGX." IEEE S&P 2020.
[3] Moghimi et al. "Cachezoom: How SGX amplifies the power of cache attacks." CHES 2017.
[4] Brasser et al. "Software grand exposure:{SGX} cache attacks are practical." USENIX WOOT 2017.
[5] Schwarz et al. "Malware guard extension: Using SGX to conceal cache attacks." DIMVA 2017.
[6] Evtyushkin, Dmitry, et al. "Branchscope: A new side-channel attack on directional branch predictor." ACM SIGPLAN 2018.
[7] Lee, Sangho, et al. "Inferring fine-grained control flow inside {SGX} enclaves with branch shadowing." USENIX Security 2017.
[8] Van Bulck et al. "Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic." ACM CCS 2018.

Intel SGX Attack Taxonomy

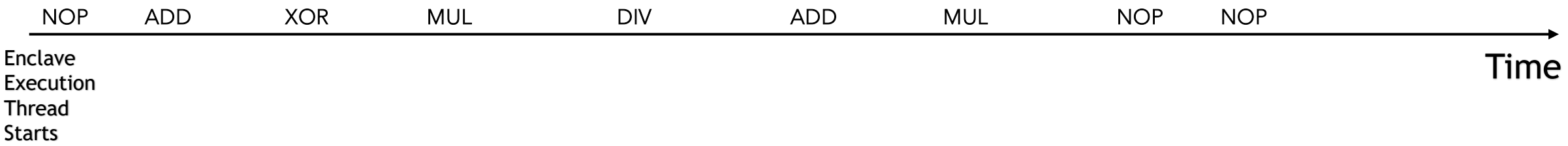
- Intel's Responsibility
  - Microcode Patches / Hardware mitigation
  - TCB Recovery
  - Hyperthreading is out
    - Remote Attestation Warning
- µarch Side Channel
  - Constant-time Coding
  - Flushing and Isolating buffers
  - Probabilistic
- Deterministic Attacks
  - Page Fault, A/D Bit, etc. (4kB Granularity)



[1] Van Bulck et al. "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution." USENIX Security 2018.
[2] Murdock et al. "Plundervolt: Software-based fault injection attacks against Intel SGX." IEEE S&P 2020.
[3] Moghimi et al. "Cachezoom: How SGX amplifies the power of cache attacks." CHES 2017.
[4] Brasser et al. "Software grand exposure:{SGX} cache attacks are practical." USENIX WOOT 2017.
[5] Schwarz et al. "Malware guard extension: Using SGX to conceal cache attacks." DIMVA 2017.
[6] Evtyushkin, Dmitry, et al. "Branchscope: A new side-channel attack on directional branch predictor." ACM SIGPLAN 2018.
[7] Lee, Sangho, et al. "Inferring fine-grained control flow inside {SGX} enclaves with branch shadowing." USENIX Security 2017.
[8] Van Bulck et al. "Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic." ACM CCS 2018.
[9] Xu et al. "Controlled-channel attacks: Deterministic side channels for untrusted operating systems." IEEE S&P 2015.
[10] Wang, Wenhao, et al. "Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX." ACM CCS 2017.

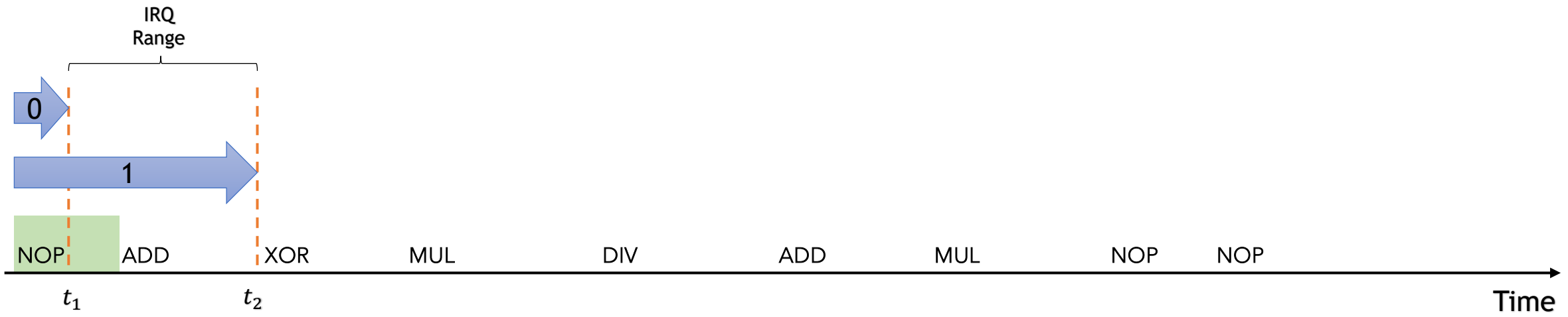
CopyCat Attack

- Malicious OS controls the interrupt handler



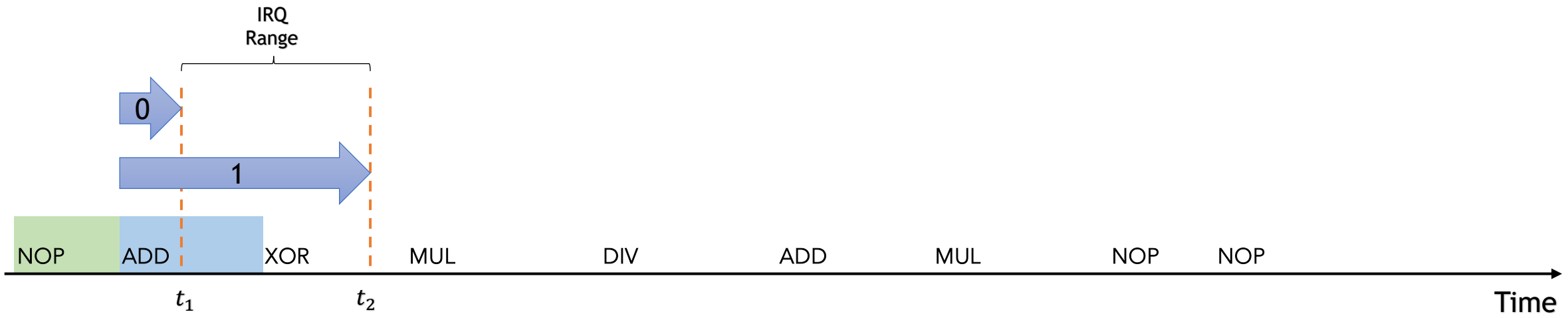
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions



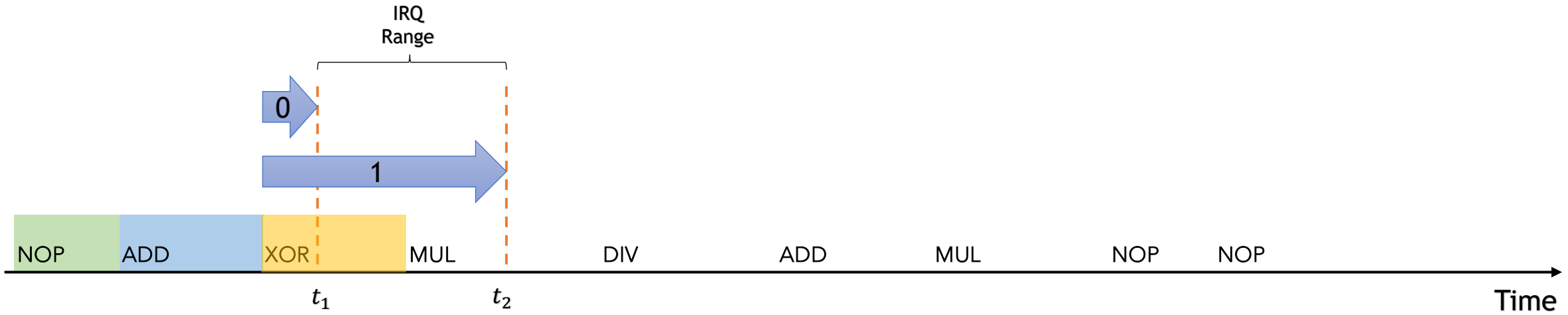
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions



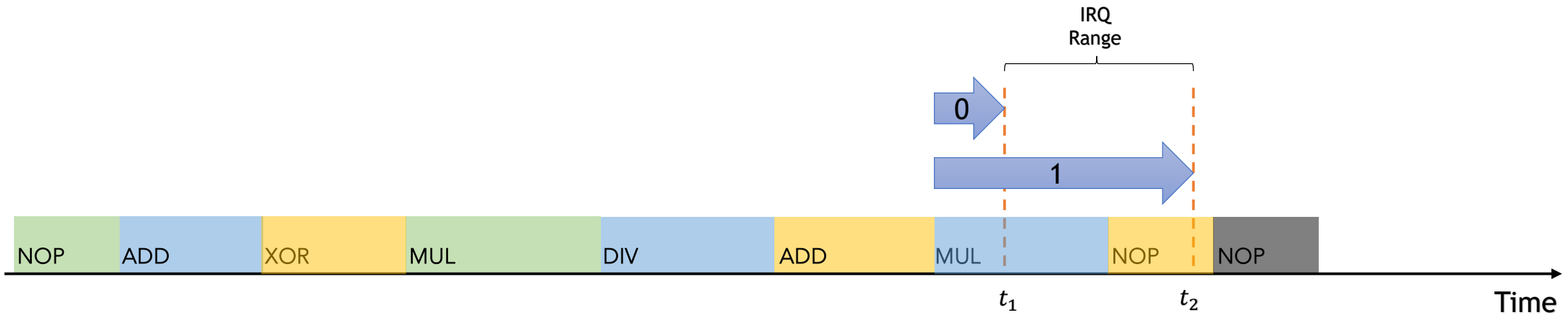
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions



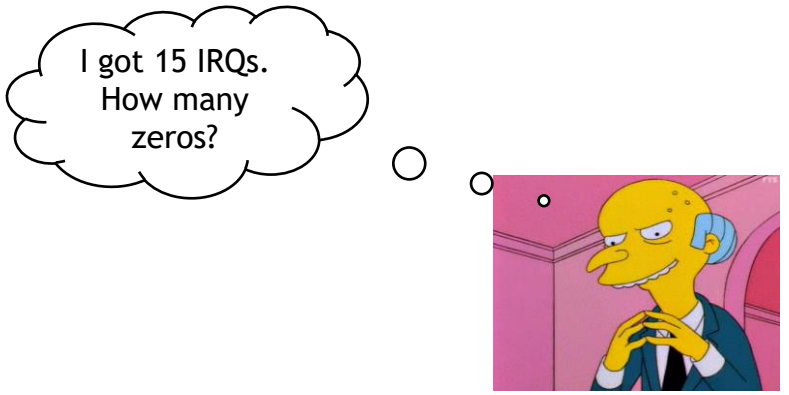
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions



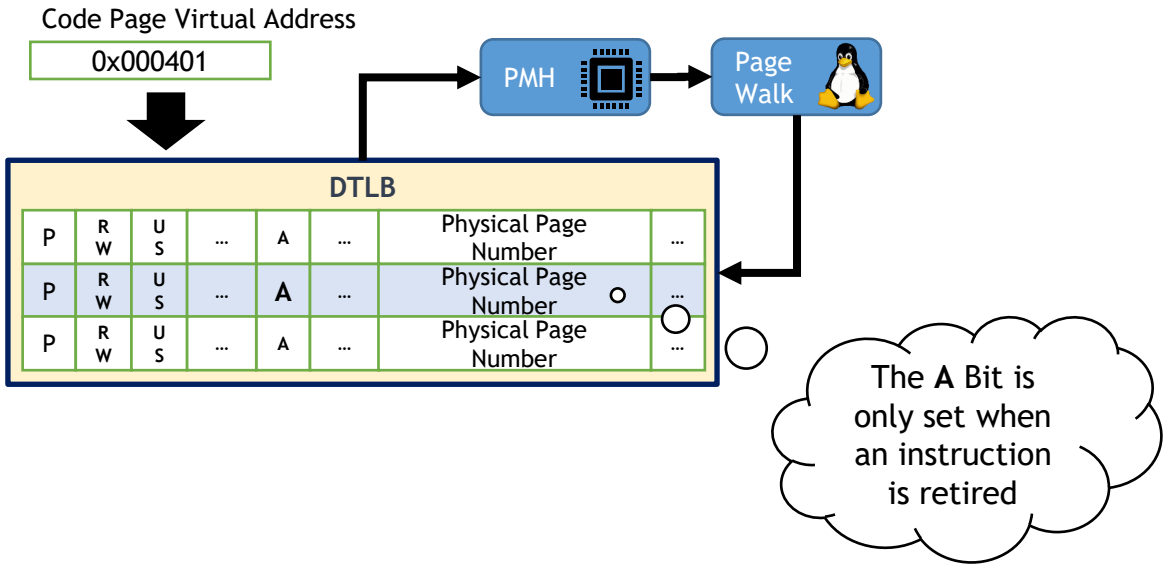
CopyCat Attack

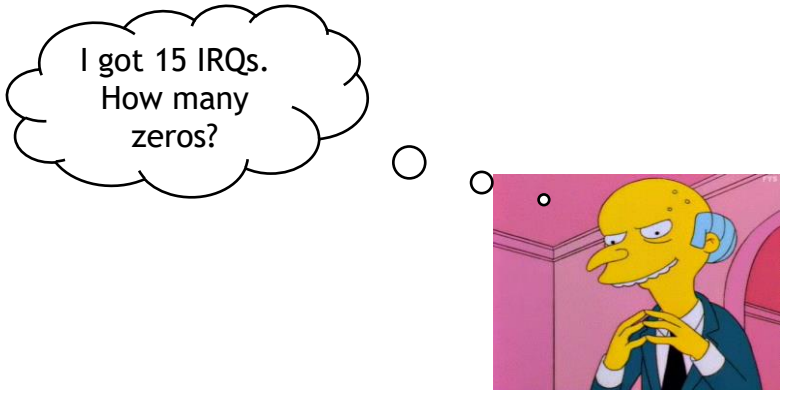
- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions



CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
- Filtering Zeros out: Clear the **A** bit before, Check the **A** bit after



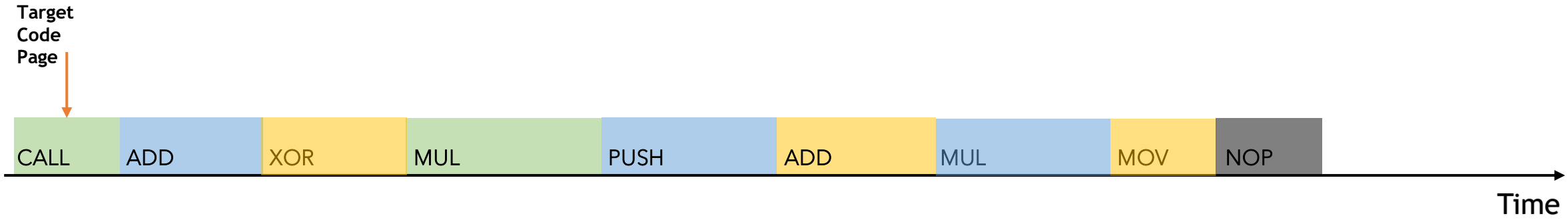


CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
- Filtering Zeros out: Clear the **A** bit before, Check the **A** bit after
- Deterministic Instruction Counting

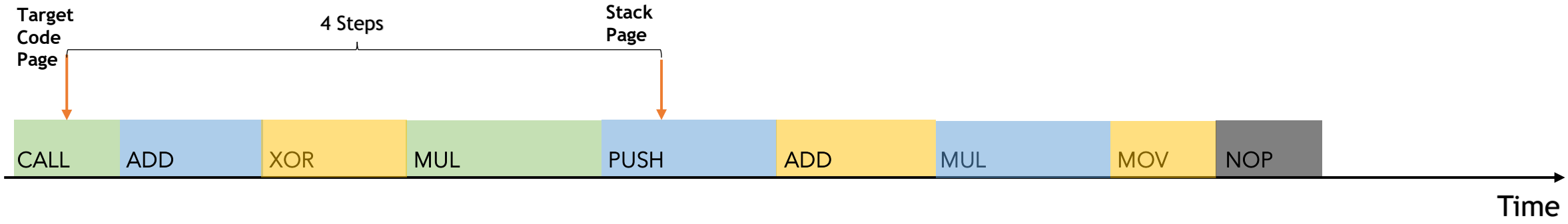
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
- Filtering Zeros out: Clear the **A** bit before, Check the **A** bit after
- Deterministic Instruction Counting
- Counting from start to end is not useful.
  - A Secondary oracle
  - Page table attack as a deterministic secondary oracle



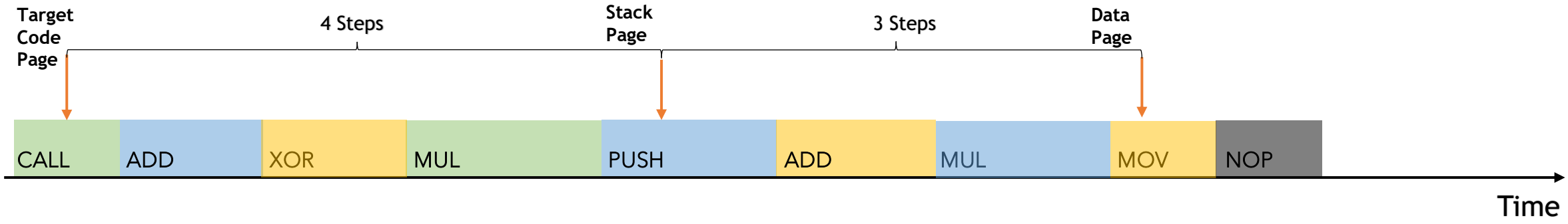
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
- Filtering Zeros out: Clear the **A** bit before, Check the **A** bit after
- Deterministic Instruction Counting
- Counting from start to end is not useful.
  - A Secondary oracle
  - Page table attack as a deterministic secondary oracle



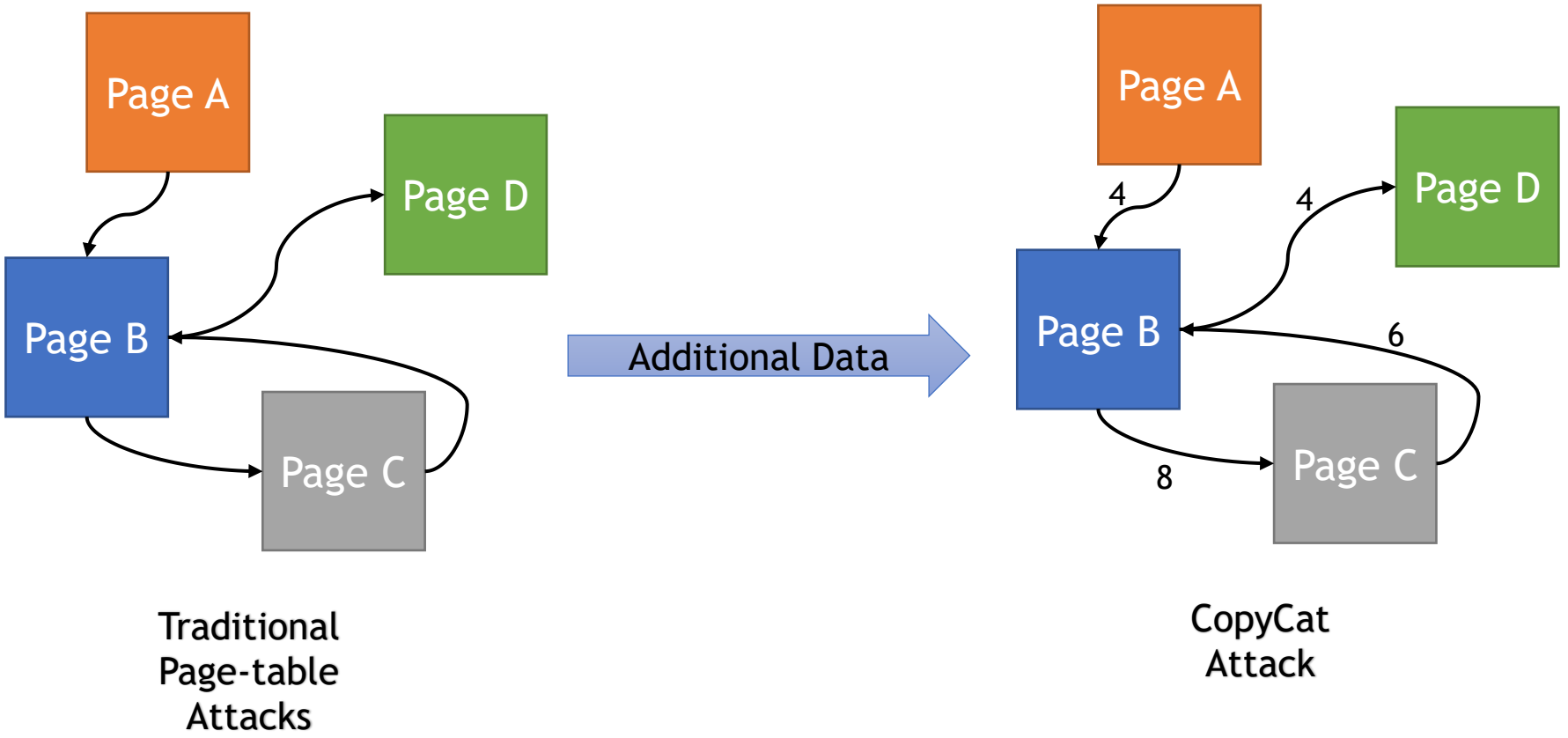
CopyCat Attack

- Malicious OS controls the interrupt handler
- A threshold to execute 1 or 0 instructions
- Filtering Zeros out: Clear the **A** bit before, Check the **A** bit after
- Deterministic Instruction Counting
- Counting from start to end is not useful.
  - A Secondary oracle
  - Page table attack as a deterministic secondary oracle



CopyCat Attack

- Previous controlled-channel attacks leak page access patterns.
- CopyCat additionally leaks number of executed instructions per each page.



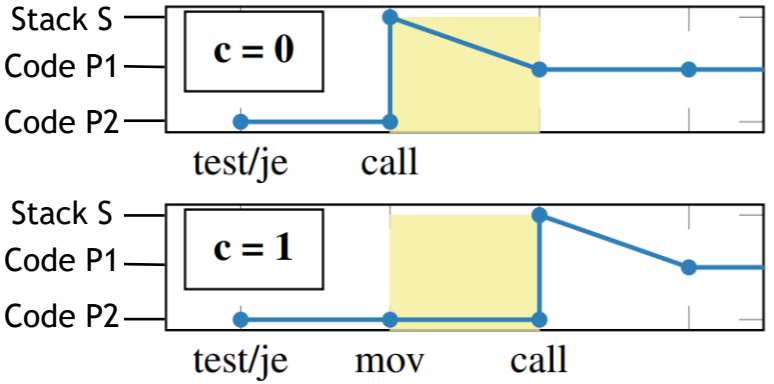
CopyCat - Leaking Branches

```
if(c == 0) {
  r = add(r, d);
}
else {
  r = add(r, s);
}
```

C Code

Compile

```
test %eax, %eax
je label
mov %edx, %esi
label:
call add
mov %eax, -0xc(%rbp)
```



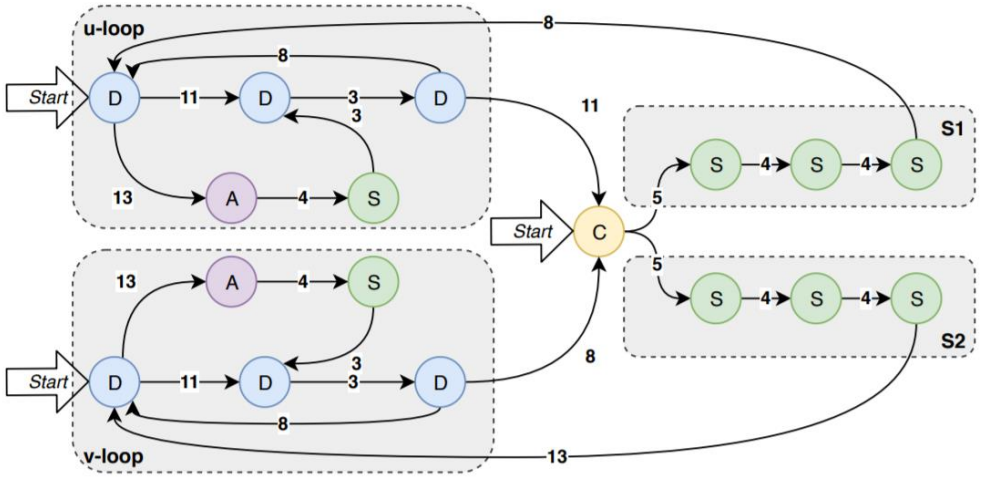
Binary Extended Euclidean Algorithm (BEEA)

- Previous attacks only leak some of the branches w/ some noise.

```
1:  procedure MODINV(u, modulus v)
2:      b_i ← 0 d_i ← 1, u_i ← u, v_i = v,
3:      while isEven(u_i) do
4:          u_i ← u_i/2
5:          if isOdd(b_i) then
6:              b_i ← b_i − u
7:          b_i ← b_i/2
8:      while isEven(v_i) do
9:          v_i ← v_i/2
10:         if isOdd(d_i) then
11:             d_i ← d_i − u
12:         d_i ← d_i/2
13:     if u_i > v_i then
14:         u_i ← u_i − v_i, b_i ← b_i − d_i
15:     else
16:         v_i ← v_i − u_i, d_i ← d_i − b_i
17:
        return d_i
```

Binary Extended Euclidean Algorithm (BEEA)

- Previous attacks only leak some of the branches w/ some noise.
- CopyCat synchronously leaks all the branches wo/ any noise.



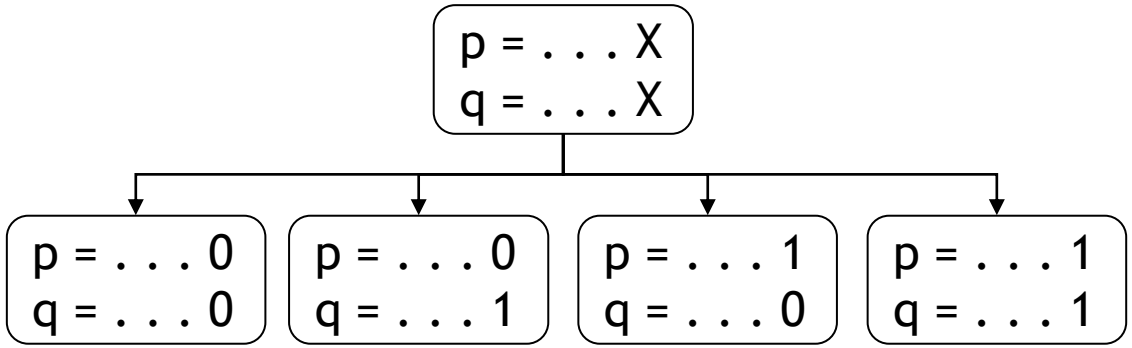

```
1:  procedure MODINV(u, modulus v)
2:      b_i ← 0 d_i ← 1, u_i ← u, v_i = v,
3:      while isEven(u_i) do
4:          u_i ← u_i/2
5:          if isOdd(b_i) then
6:              b_i ← b_i − u
7:          b_i ← b_i/2
8:      while isEven(v_i) do
9:          v_i ← v_i/2
10:         if isOdd(d_i) then
11:             d_i ← d_i − u
12:         d_i ← d_i/2
13:     if u_i > v_i then
14:         u_i ← u_i − v_i, b_i ← b_i − d_i
15:     else
16:         v_i ← v_i − u_i, d_i ← d_i − b_i
17:
        return d_i
```

CopyCat on WolfSSL - Cryptanalysis

- Single-trace attack during RSA key generation: $q_{inv} = q^{-1}\ mod\ p$
  - We know that $\mathrm{p}.\mathrm{q} = \mathrm{N}$, and N is public

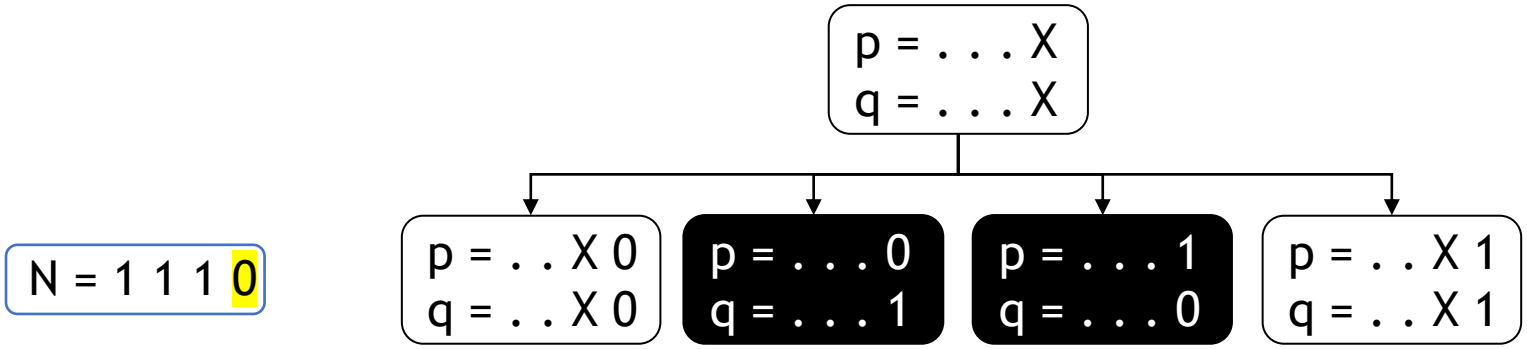
CopyCat on WolfSSL - Cryptanalysis

- Single-trace attack during RSA key generation: $q_{inv} = q^{-1} \, mod \, p$
  - We know that $p.q = N$, and N is public
  - Branch and prune algorithm with the help of the recovered trace



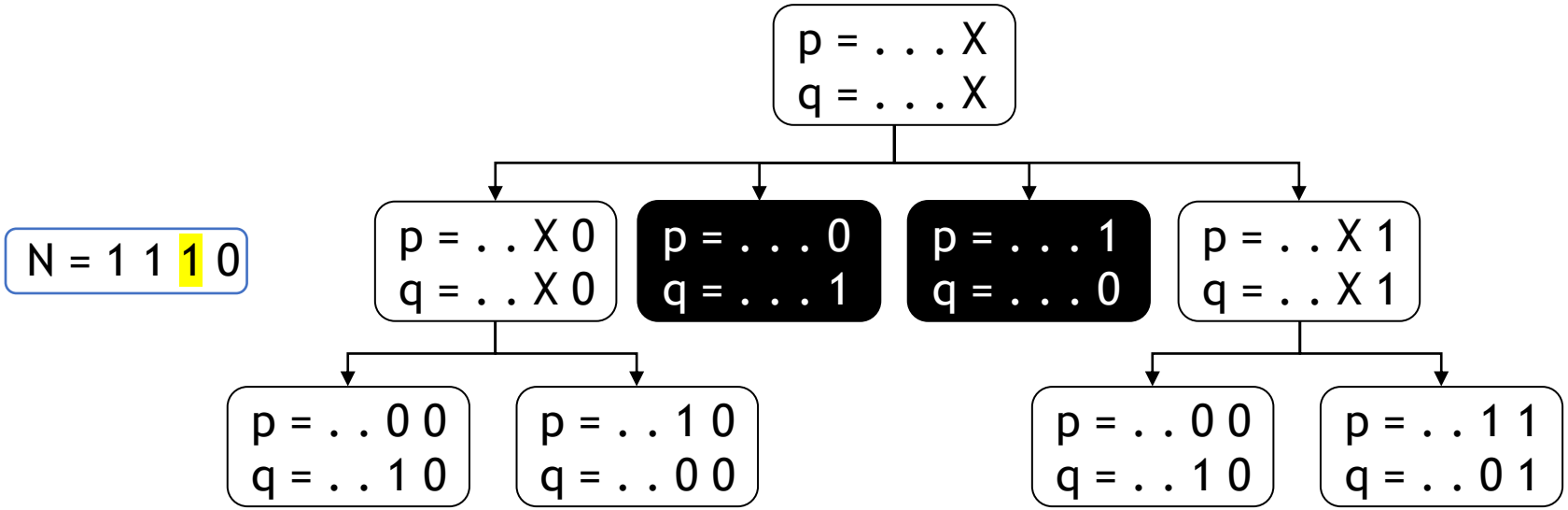
CopyCat on WolfSSL - Cryptanalysis

- Single-trace Attack during RSA Key Generation: $q_{inv} = q^{-1} \, mod \, p$
  - We know that $p.q = N$, and N is public
  - Branch and prune algorithm with the help of the recovered trace



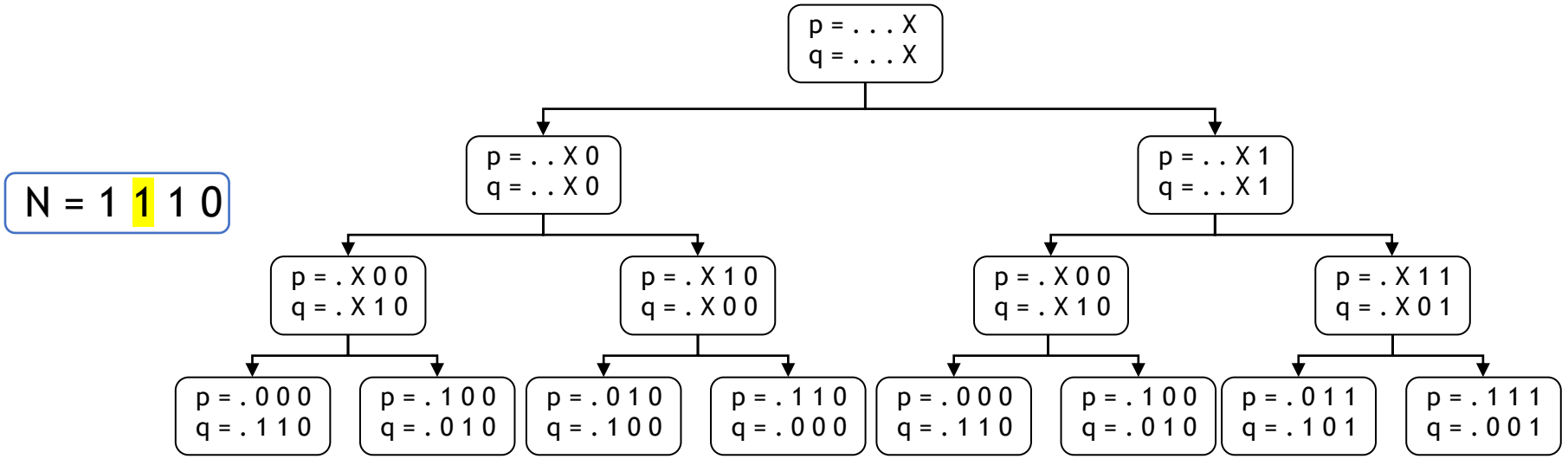
CopyCat on WolfSSL - Cryptanalysis

- Single-trace Attack during RSA Key Generation: $q_{inv} = q^{-1} \, mod \, p$
  - We know that $p.q = N$, and N is public
  - Branch and prune algorithm with the help of the recovered trace



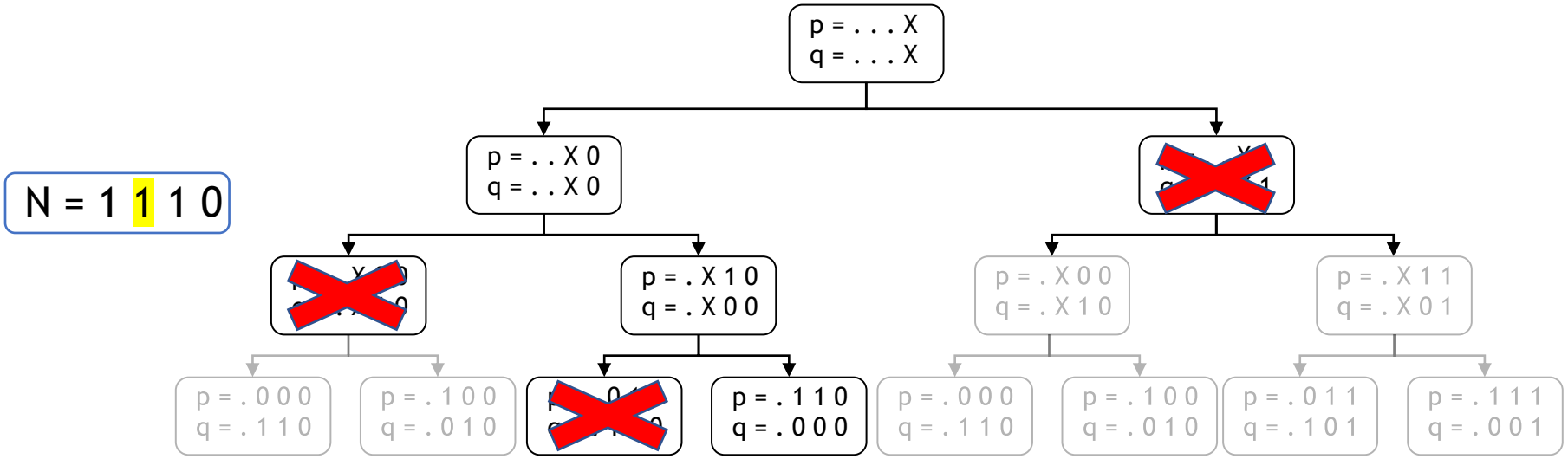
CopyCat on WolfSSL - Cryptanalysis

- Single-trace Attack during RSA Key Generation: $q_{inv} = q^{-1} \, mod \, p$
  - We know that $p.q = N$, and N is public
  - Branch and prune algorithm with the help of the recovered trace



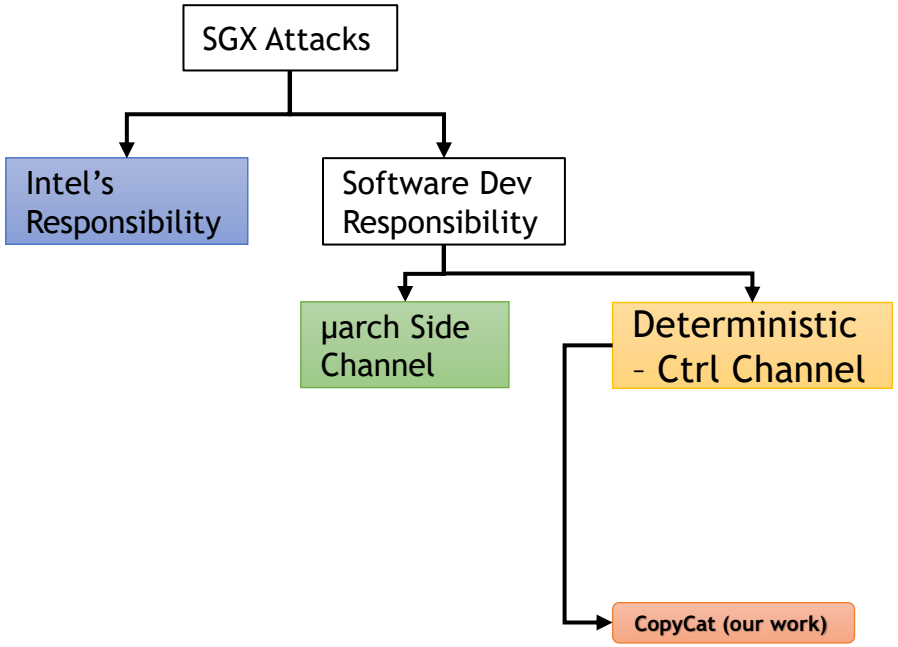
CopyCat on WolfSSL - Cryptanalysis

- Single-trace Attack during RSA Key Generation: $q_{inv} = q^{-1} \, mod \, p$
  - We know that $p.q = N$, and N is public
  - Branch and prune algorithm with the help of the recovered trace



Benefits of CopyCat compared to Previous Attacks

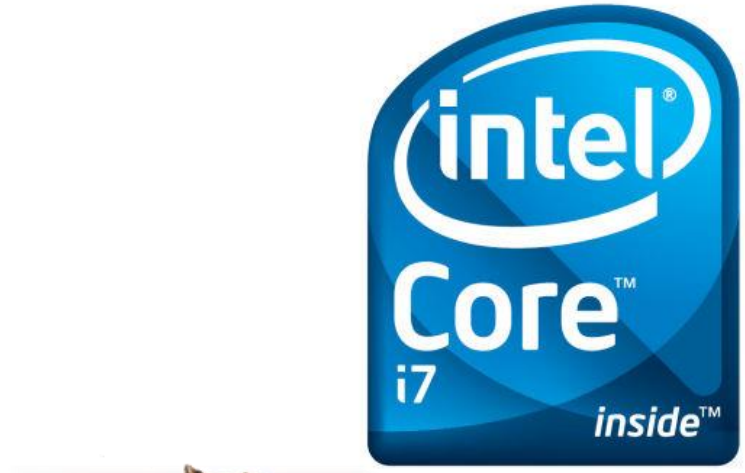
- Instruction level granularity
  - Imbalance number of instructions
  - Leak the outcome of branches
- Fully deterministic and reliable
  - Millions of instructions tested
- Easy to scale and replicate
  - No reverse engineering of branches and microarchitectural components
  - Tracking all the branches synchronously



Threats to Trusted Platform Model (T3)

High-resolution Timing Test

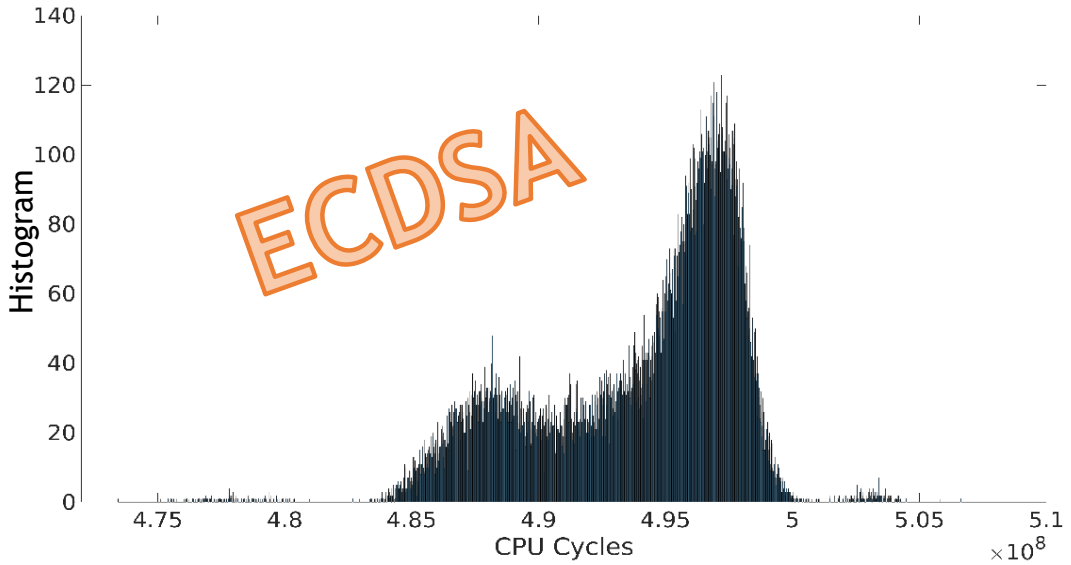
- TPM frequency ~= 32-120 MHz
- CPU Frequency is more than 2 GHz



100x faster!!

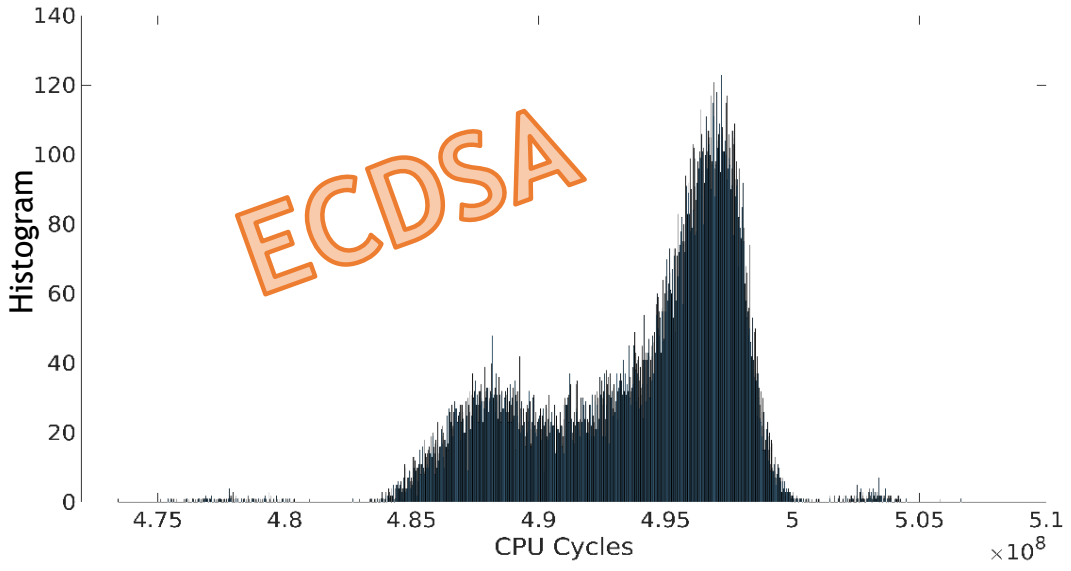
High-resolution Timing Test - Intel PTT (fTPM)

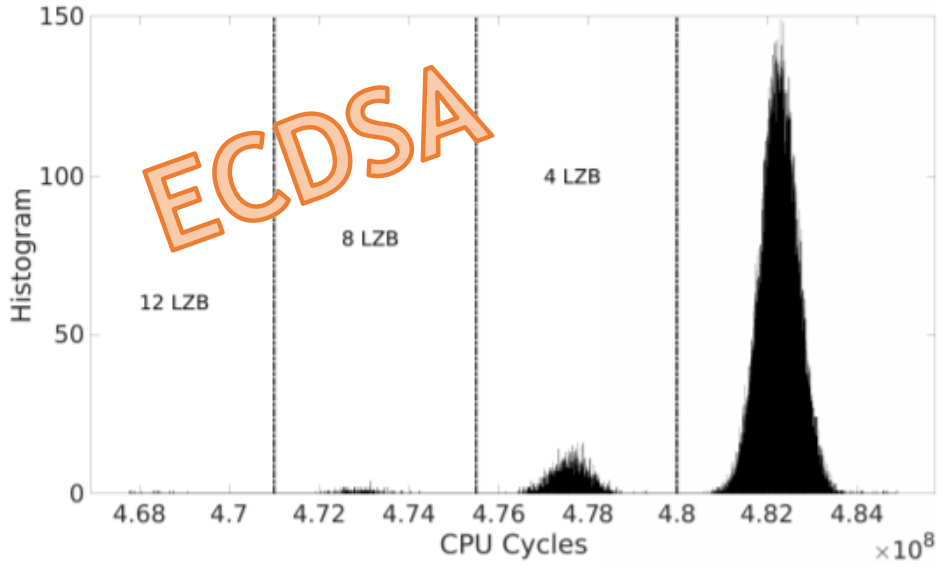
- Intel Platform Trust Technology (PTT)
  - Integrated firmware-TPM inside the CPU package



High-resolution Timing Test - Intel PTT (fTPM)

- Intel Platform Trust Technology (PTT)
  - Integrated firmware-TPM inside the CPU package
- Kernel Driver to increase the Resolution





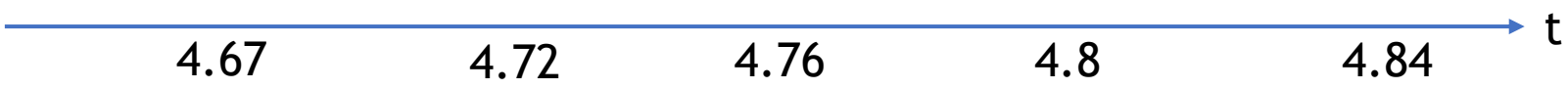
High-resolution Timing Test – ECDSA Nonce Leakage

- Intel fTPM: 4-bit Window Nonce Length Leakage
  - ECDSA
  - ECSChnorr
  - BN-256 (ECDAA)

$ECDSA\ Sign$:
$(x_1, y_1) = k_i \times G$
$r_i = x_1\ mod\ n$
$s_i = {k_i}^{-1}(z + r_i d)\ mod\ n$

Nonce

01010001001111111…111
00001001001111111…111
11010001001111111…111
00000000001111111…111
00000000000011111…111



High-resolution Timing Test – ECDSA Nonce Leakage

- Intel fTPM: 4-bit Window Nonce Length Leakage
  - ECDSA
  - ECSChnorr
  - BN-256 (ECDAA)

$$ECDSA\ Sign:$$
$$(x_1, y_1) = k_i \times G$$
$$r_i = x_1\ mod\ n$$
$$s_i = {k_i}^{-1}(z + r_i d)\ mod\ n$$

Nonce

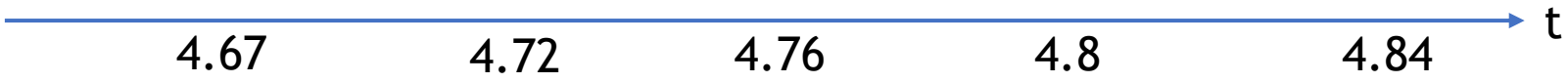
0101000100111111…111

0000100100111111…111

1101000100111111…111

0000000000111111…111

0000000000001111…111



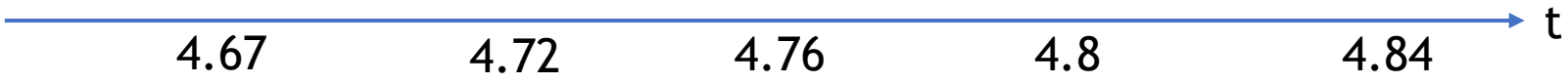
High-resolution Timing Test - ECDSA Nonce Leakage

- Intel fTPM: 4-bit Window Nonce Length Leakage
  - ECDSA
  - ECSChnorr
  - BN-256 (ECDAA)

$$ECDSA\ Sign:$$
$$(x_1, y_1) = k_i \times G$$
$$r_i = x_1\ mod\ n$$
$$s_i = {k_i}^{-1}(z + r_i d)\ mod\ n$$

Nonce

01010001001111111...111
00001001001111111...111
11010001001111111...111
00000000001111111...111
00000000000011111...111



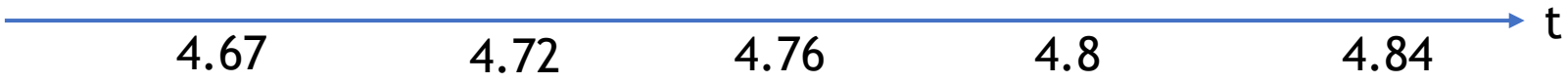
High-resolution Timing Test - ECDSA Nonce Leakage

- Intel fTPM: 4-bit Window Nonce Length Leakage
  - ECDSA
  - ECSChnorr
  - BN-256 (ECDAA)

$$ECDSA\ Sign:$$
$$(x_1, y_1) = k_i \times G$$
$$r_i = x_1\ mod\ n$$
$$s_i = {k_i}^{-1}(z + r_i d)\ mod\ n$$

Nonce

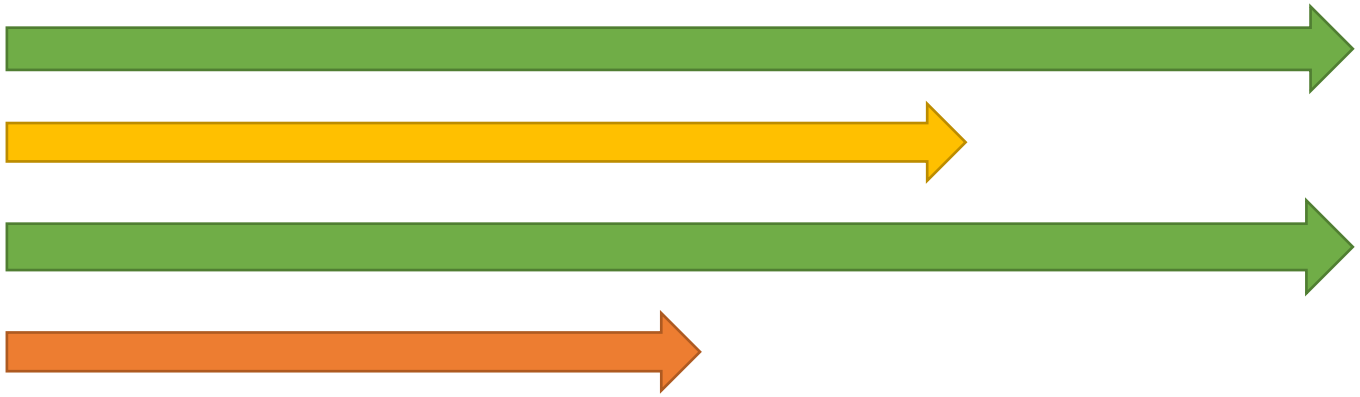
01010001001111111...111
00001001001111111...111
11010001001111111...111
00000000001111111...111
00000000000011111...111



High-resolution Timing Test - ECDSA Nonce Leakage

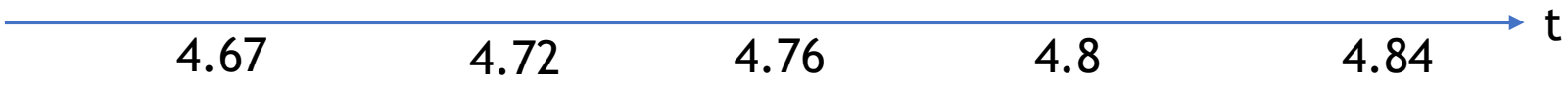
- Intel fTPM: 4-bit Window Nonce Length Leakage
  - ECDSA
  - ECSChnorr
  - BN-256 (ECDAA)

$$ECDSA\ Sign:$$
$$(x_1, y_1) = k_i \times G$$
$$r_i = x_1\ mod\ n$$
$$s_i = {k_i}^{-1}(z + r_i d)\ mod\ n$$



Nonce

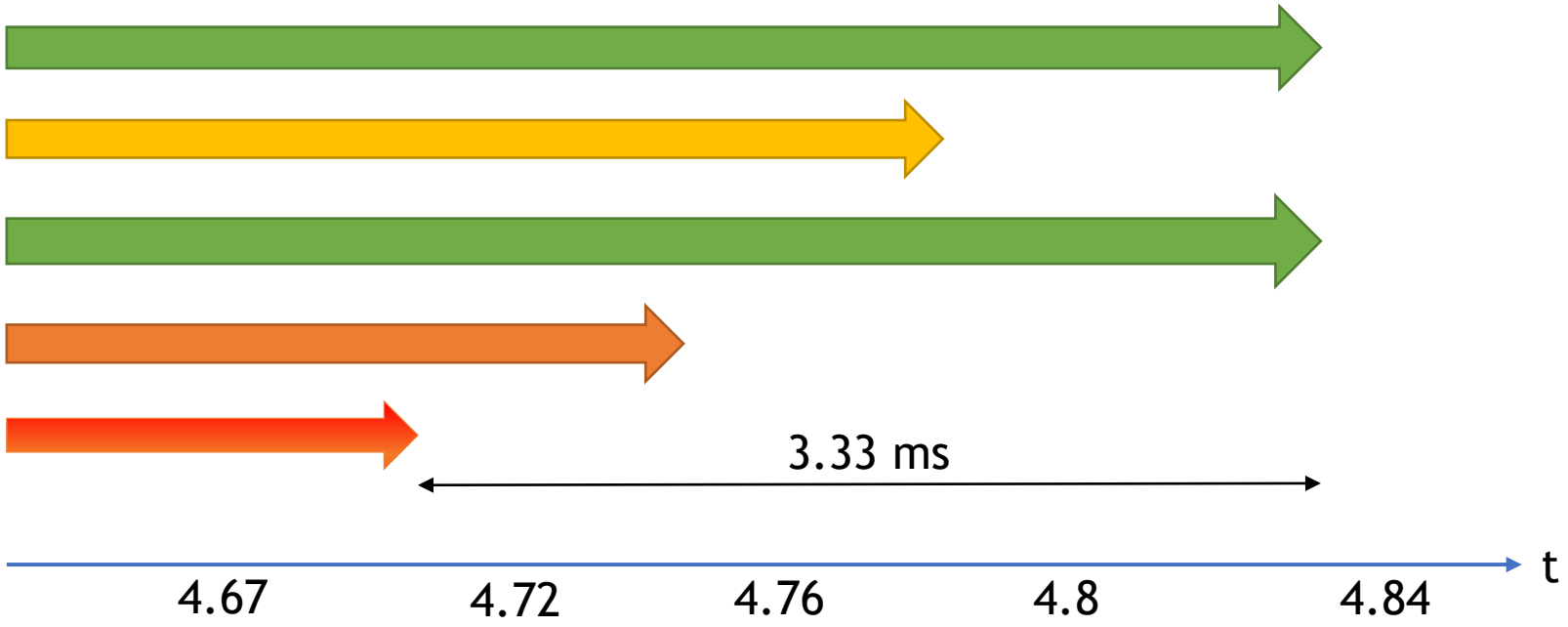
01010001001111111...111
00001001001111111...111
11010001001111111...111
00000000001111111...111
00000000000011111...111



High-resolution Timing Test - ECDSA Nonce Leakage

- Intel fTPM: 4-bit Window Nonce Length Leakage
  - ECDSA
  - ECSChnorr
  - BN-256 (ECDAA)

$$ECDSA\ Sign:$$
$$(x_1, y_1) = k_i \times G$$
$$r_i = x_1\ mod\ n$$
$$s_i = {k_i}^{-1}(z + r_i d)\ mod\ n$$



Nonce

01010001001111111...111
00001001001111111...111
11010001001111111...111
00000000001111111...111
00000000000001111...111

High-resolution Timing Test - Analysis Of Devices

- RSA and ECDSA timing test on 3 dedicated TPM and Intel fTPM
- Various non-constant behaviour for both RSA and ECDSA

Machine	CPU	Vendor	TPM	Firmware/Bios
NUC 8i7HNK	Core i7-8705G	Intel	PTT (fTPM)	NUC BIOS 0053
NUC 7i3BNK	Core i3-7100U	Intel	PTT (fTPM)	NUC BIOS 0076
Asus GL502VM	Core i7-6700HQ	Intel	PTT (fTPM)	Latest OEM
Asus K501UW	Core i7 6500U	Intel	PTT (fTPM)	Latest OEM
Dell XPS 8920	Core i7-7700	Intel	PTT (fTPM)	Dell BIOS 1.0.4
Dell Precision 5510	Core i5-6440HQ	Nuvoton	rls NPCT	NTC 1.3.2.8
Lenovo T580	Core i7-8650U	STMicro	ST33TPHF2ESPI	STMicro 73.04
NUC 7i7DNKE	Core i7-8650U	Infineon	SLB 9670	NUC BIOS 0062

TPM-Fail – Recovering Private ECDSA Key

- TPM is programmed with an unknown key.
- We already have a template for t_i.

- Attack Steps:

1. Collect list of signatures (r_i, s_i) and timing samples t_i.
2. Filter signatures based on t_i and keeps (r_i, s_i) with a known bias.
3. Lattice-based attack to recover private key d, from signatures with biased nonce k_i.

Lattice and Hidden Number Problem

- $s = k^{-1}(z + dr) \; mod \; n \rightarrow \; k_i^{-1} \; - s_i^{-1} r_i d - s_i^{-1} z \equiv 0 \; mod \; n$

Lattice and Hidden Number Problem

- $s = k^{-1}(z + dr) \, mod \, n \rightarrow \; k_i^{-1} \; - s_i^{-1} r_i d - s_i^{-1} z \equiv 0 \, mod \, n$
- $A_i = -s_i^{-1} r_i, B_i = -s_i^{-1} z \; \rightarrow k_i + A_i d + B_i = 0$

Lattice and Hidden Number Problem

- $s = k^{-1}(z + dr)\ mod\ n \rightarrow\ k_i^{-1}\ - s_i^{-1} r_i d - s_i^{-1} z \equiv 0\ mod\ n$
- $A_i = -s_i^{-1} r_i, B_i = -s_i^{-1} z\ \rightarrow k_i + A_i d + B_i = 0$
- Let X be the upper bound on $\mathrm{k_i}$ and $(d, k_{0,}\ k_1 \ldots, k_n)$ is unknown

Boneh and Venkatesan[1]

[1] Boneh D, Venkatesan R. Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. InAnnual International Cryptology Conference 1996 Aug 18 (pp. 129-142). Springer, Berlin, Heidelberg.

Lattice and Hidden Number Problem

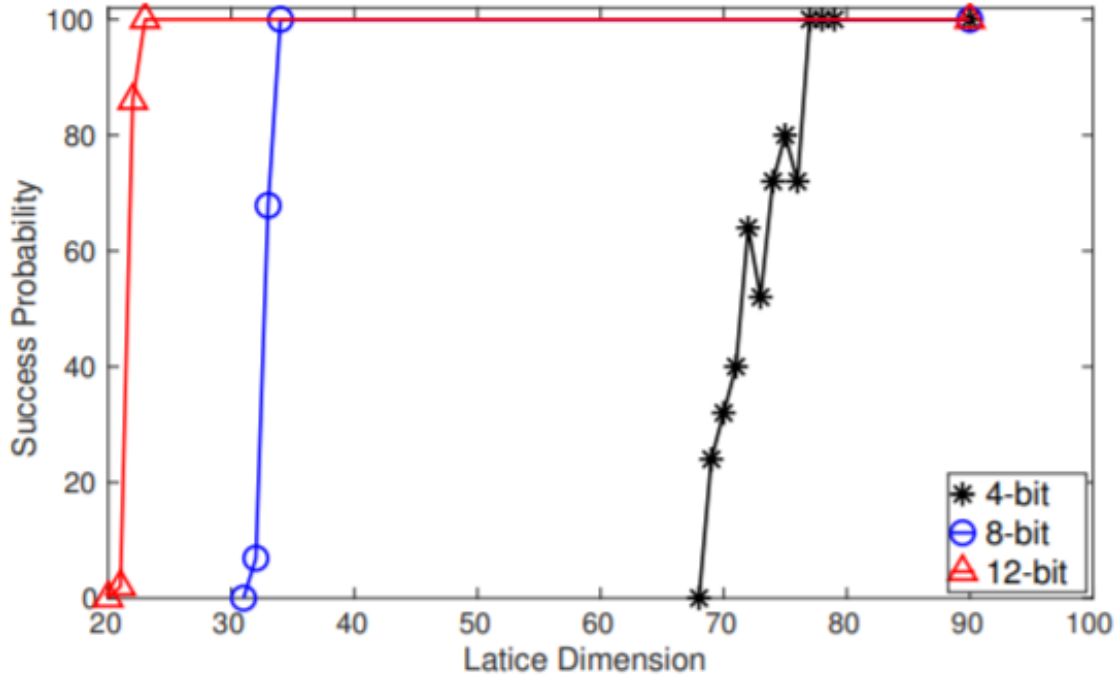
- $s = k^{-1}(z + dr)\ mod\ n \rightarrow\ k_i^{-1}\ - s_i^{-1} r_i d - s_i^{-1} z \equiv 0\ mod\ n$
- $A_i = -s_i^{-1} r_i, B_i = -s_i^{-1} z\ \rightarrow k_i + A_i d + B_i = 0$
- Let X be the upper bound on $\mathrm{k_i}$ and $(d, k_{0,}\ k_1 \ldots, k_n)$ is unknown
- Lattice Construction:

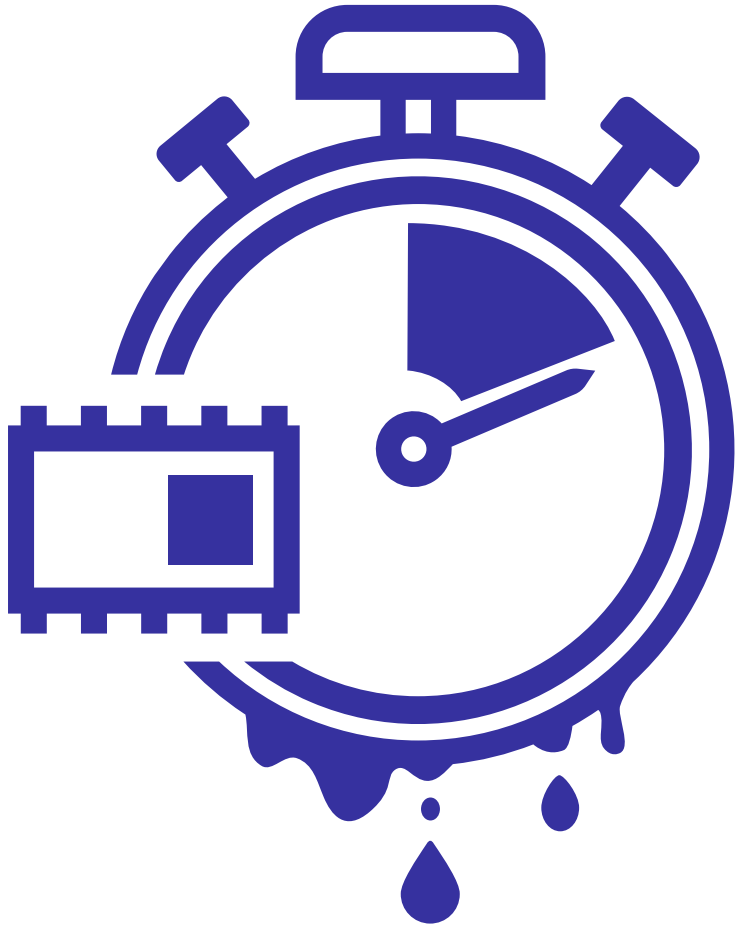
$$\begin{bmatrix} n & & & & & \\ & n & & & & \\ & & \ddots & & & \\ & & & n & & \\ A_1 & A_2 & \ldots & A_t & \frac{X}{n} & \\ B_1 & B_2 & \ldots & B_t & & X \end{bmatrix}$$

TPM-Fail – Key Recovery Results

- Intel fTPM
  - ECDSA, ECSchnorr and BN-256 (ECDAA)
  - Three different threat model System, User, Network
- STMicroelectronics TPM
  - CC EAL4+ Certified

Threat Model	TPM	Scheme	#Sign.	Time
Local System	ST TPM	ECDSA	39,980	80 mins
Local System	fTPM	ECDSA	1,248	4 mins
Local System	fTPM	ECSchnorr	1,040	3 mins
Local User	fTPM	ECDSA	15,042	18 mins

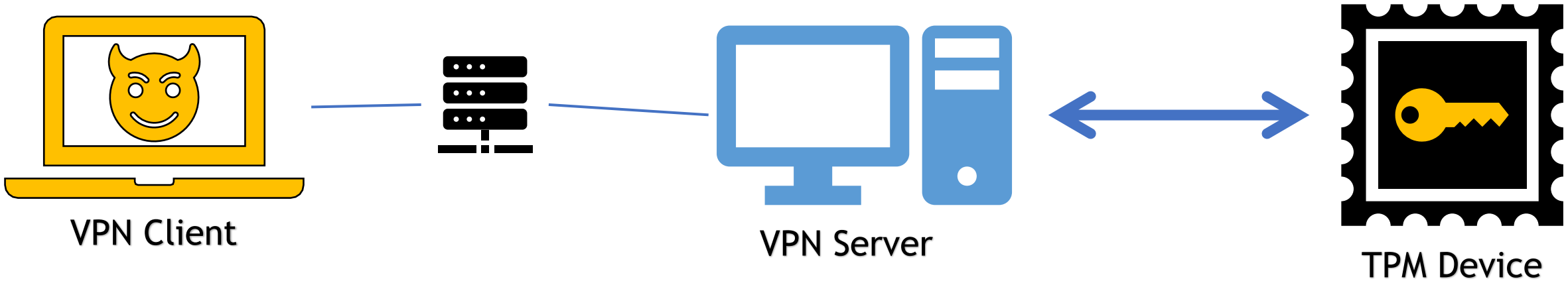




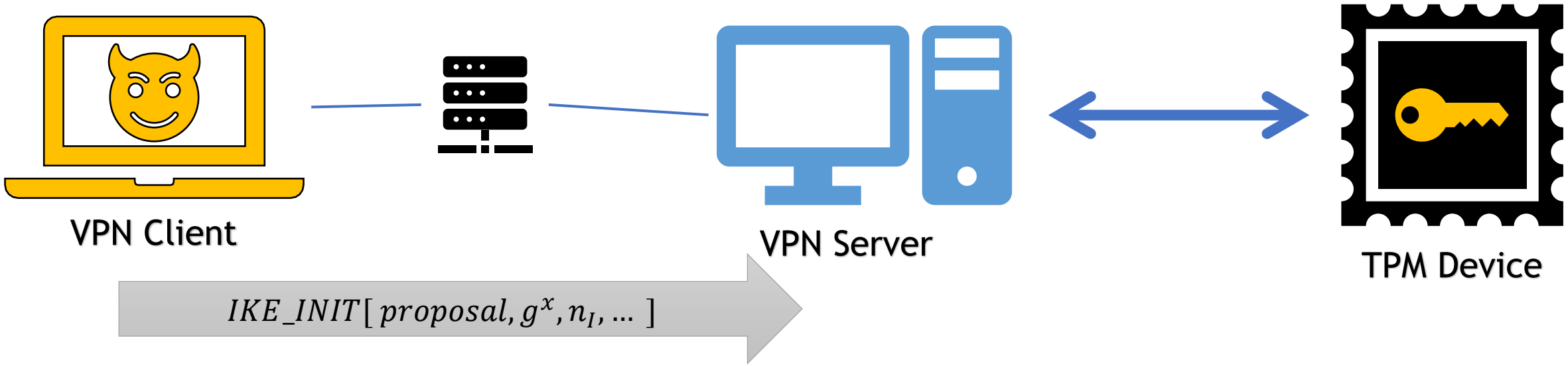
Timing difference for each window	**1.11 ms**
ping 192.168.1.x	average rtt **0.713 ms**
ping 1.1.1.1 (Cloudflare DNS)	average rtt **19.312 ms**

Bonus: Timing Attack over A Network (T0)

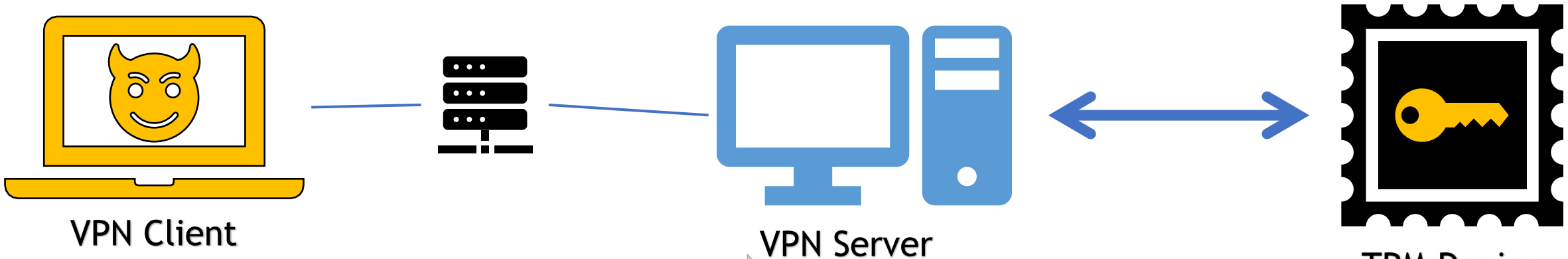
TPM-Fail Case Study: StrongSwan VPN



TPM-Fail Case Study: StrongSwan VPN



TPM-Fail Case Study: StrongSwan VPN

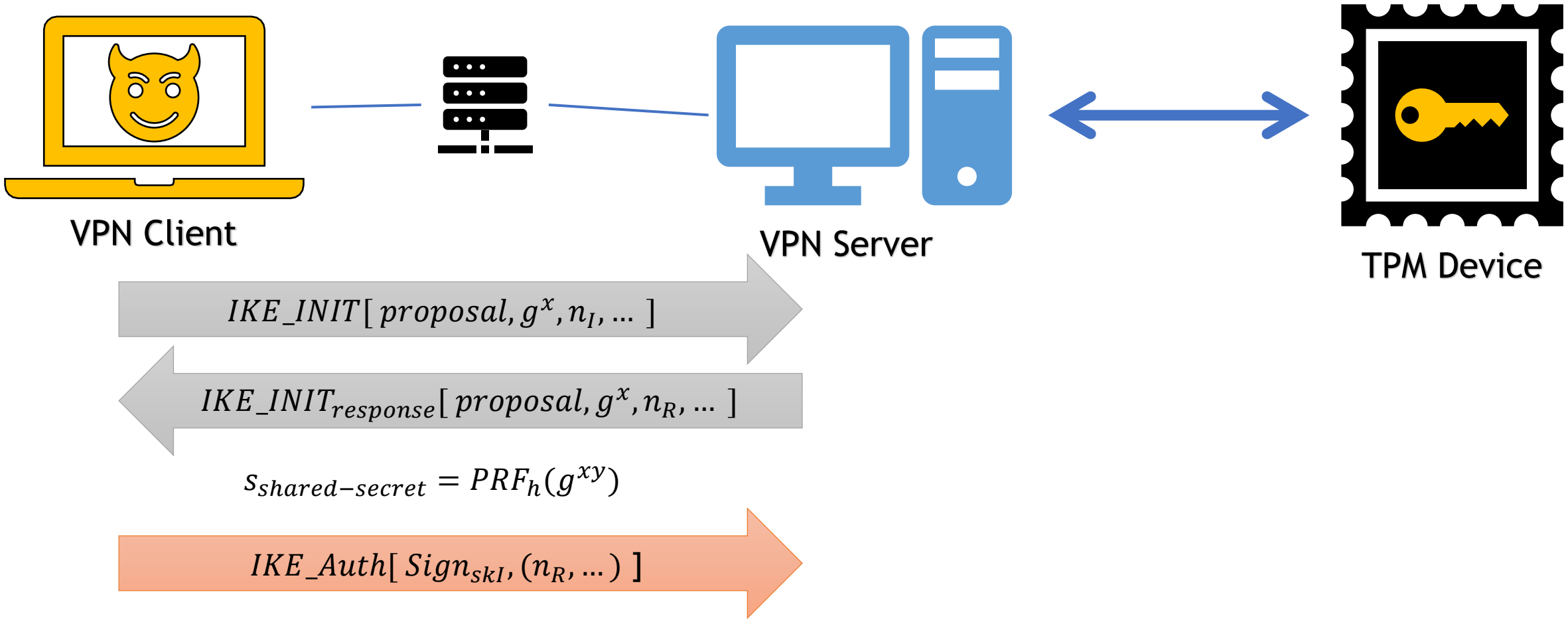


$IKE\_INIT[\, proposal, g^x, n_I, \dots \,]$

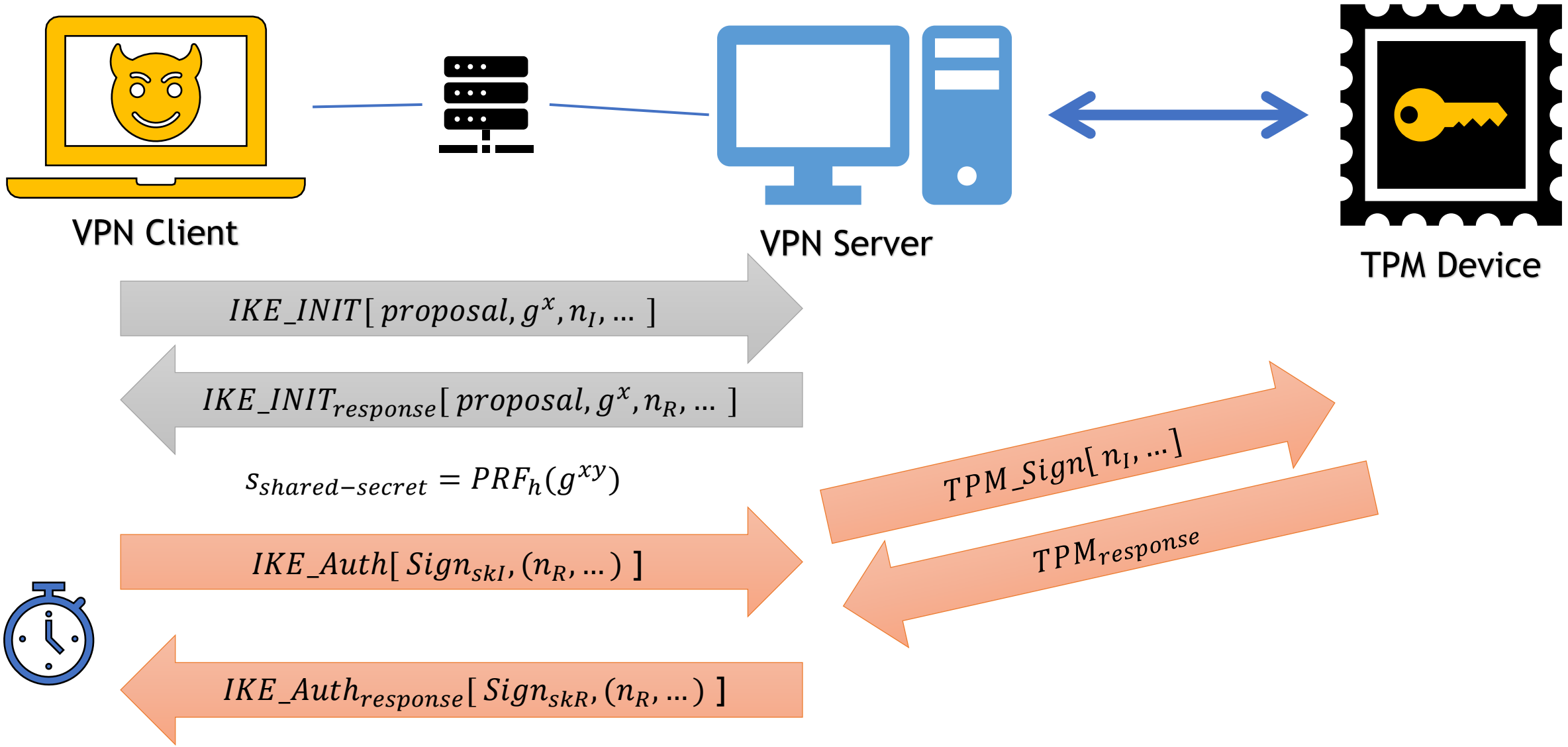
$IKE\_INIT_{response}[\, proposal, g^x, n_R, \dots \,]$

$s_{shared-secret} = PRF_h(g^{xy})$

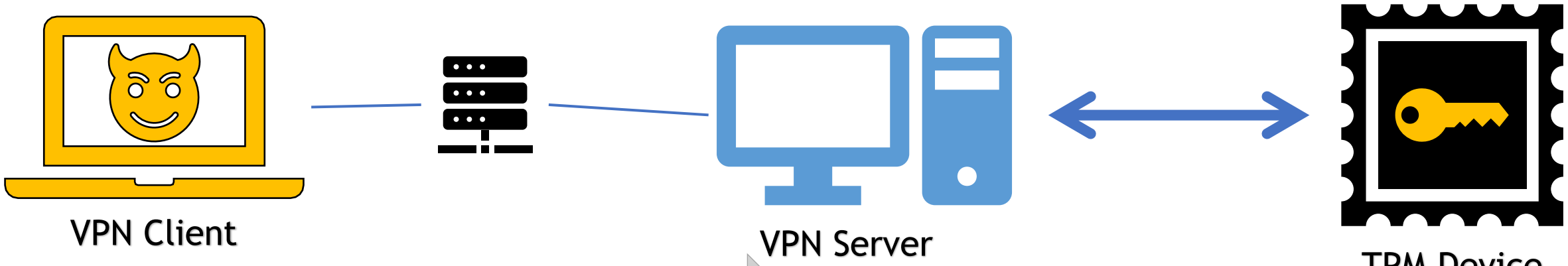
TPM-Fail Case Study: StrongSwan VPN



TPM-Fail Case Study: StrongSwan VPN



TPM-Fail Case Study: StrongSwan VPN



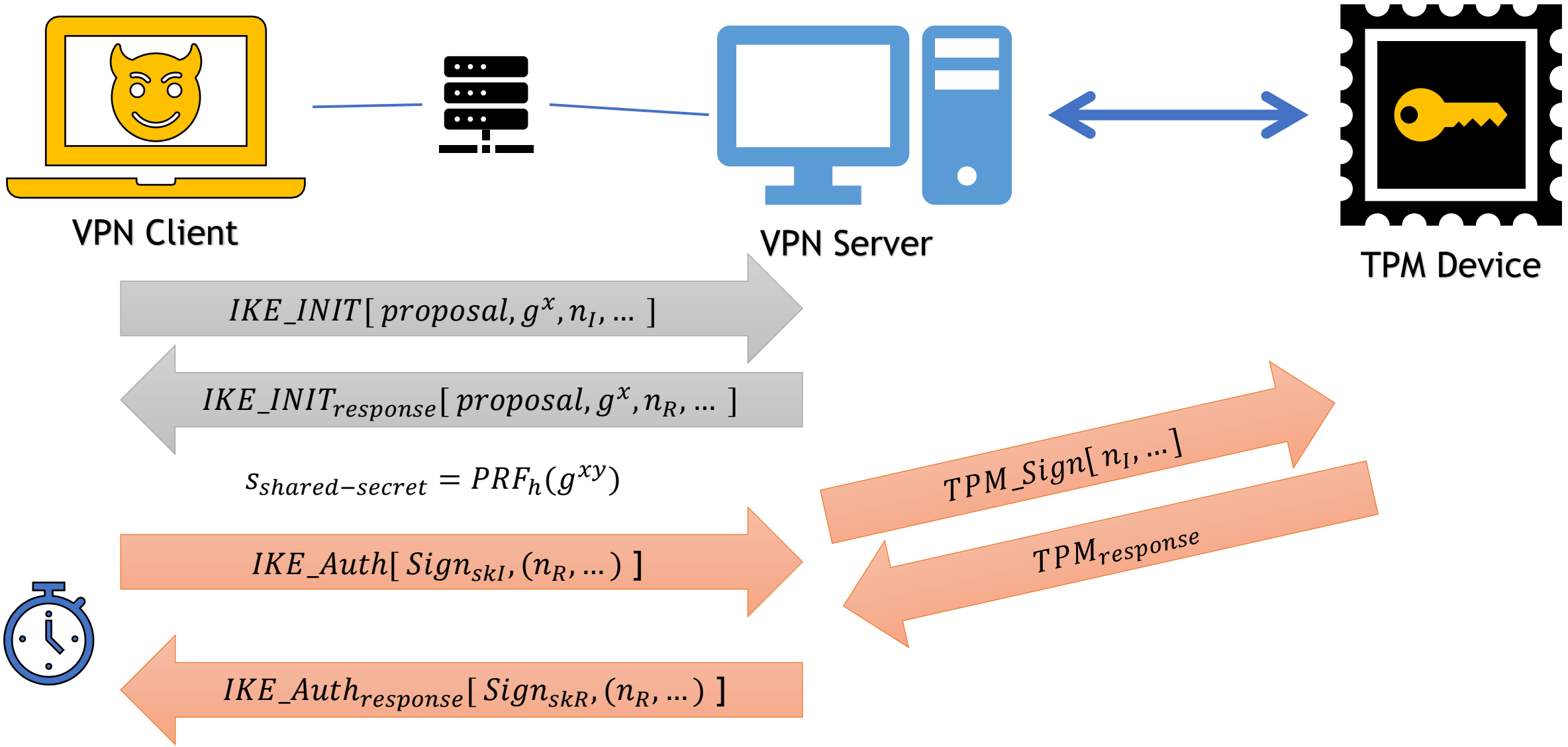
$IKE\_INIT[\ proposal, g^x, n_I, \dots\ ]$

$IKE\_INIT_{response}[\ proposal, g^x, n_R, \dots\ ]$

$s_{shared-secret} = PRF_h(g^{xy})$

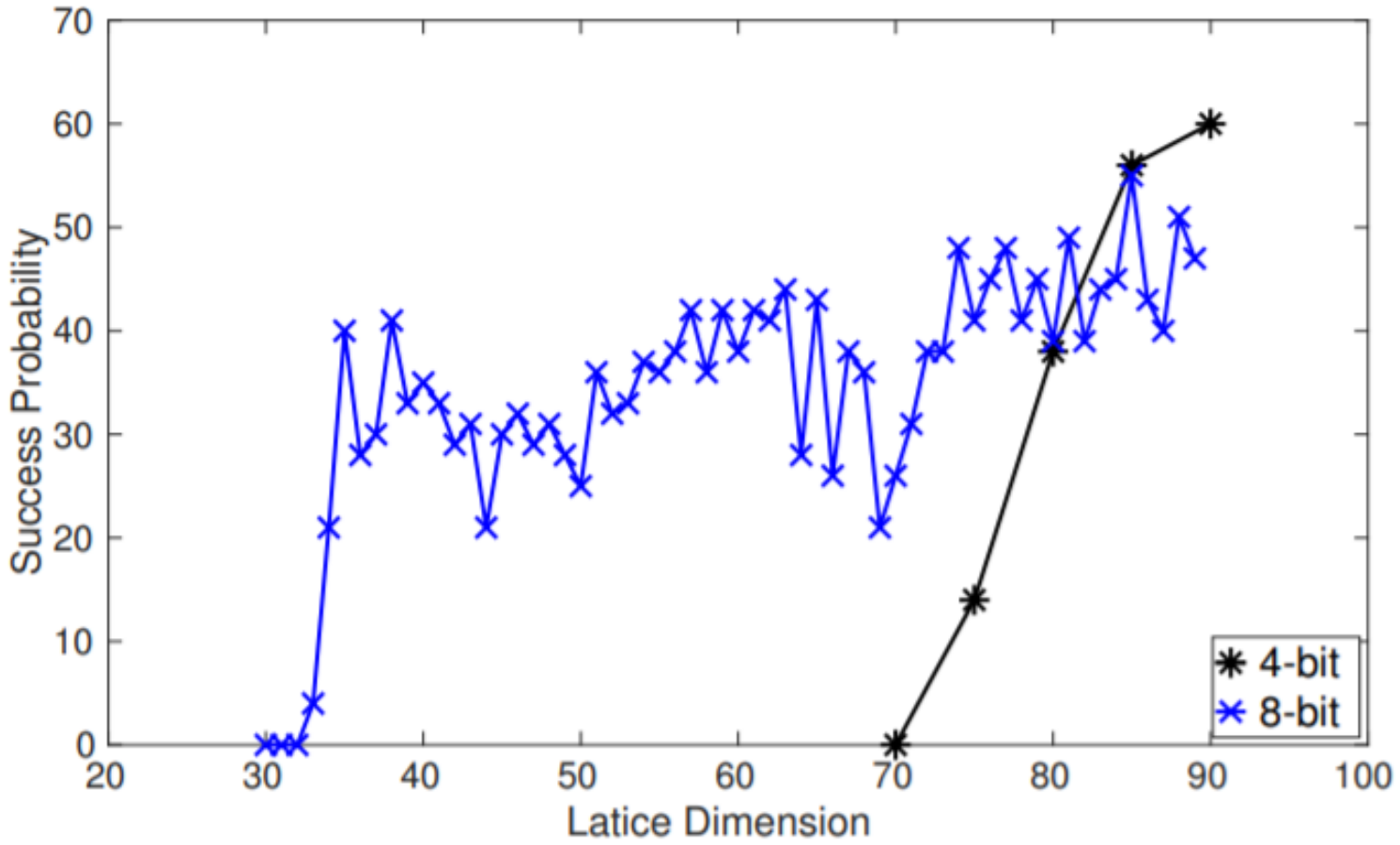
$IKE\_Auth[\ Sign_{skI}, (n_R, \dots)\ ]$

TPM-Fail Case Study: StrongSwan VPN



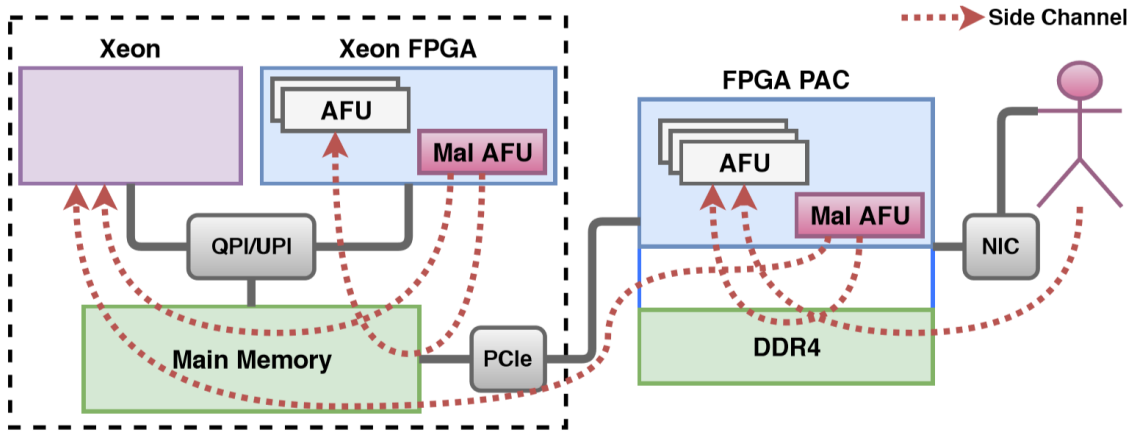
TPM-Fail Case Study: StrongSwan VPN

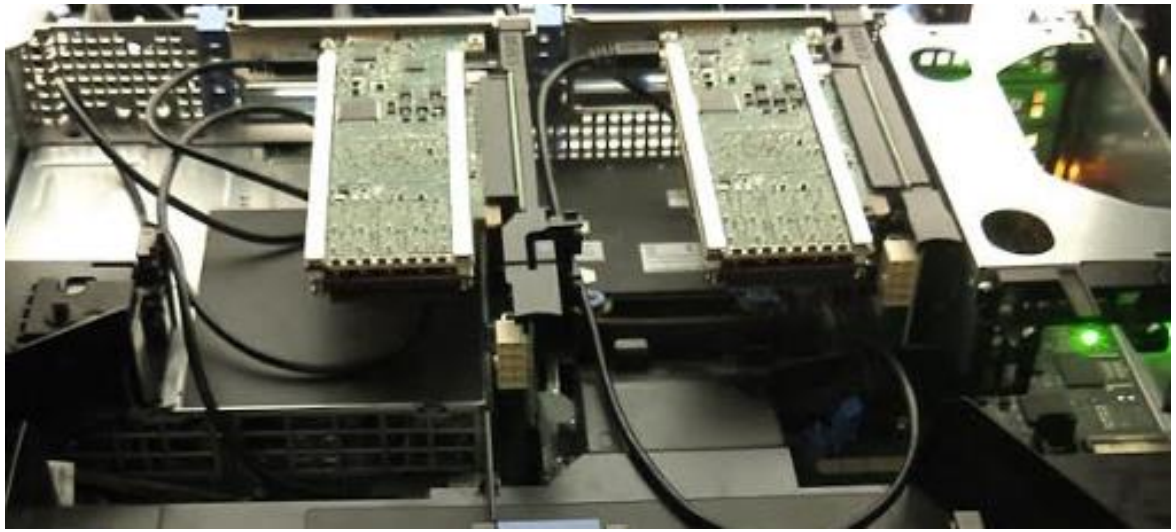
- Stealing private keys remotely after 44,000 handshake ~= 5 hours



Additional Contributions - JackHammer

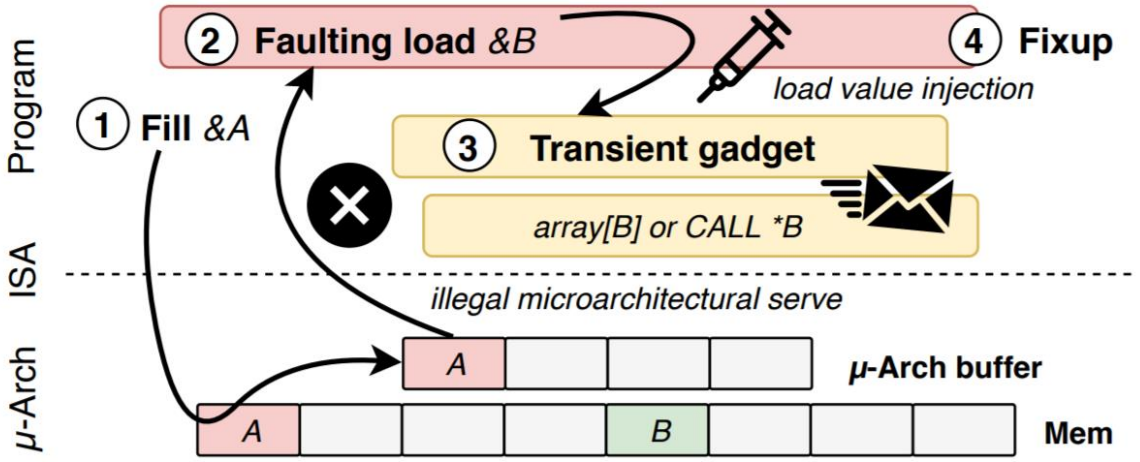
- Heterogeneous FPGA-CPU Platform
- Side-channel analysis of FPGA Cache
- Faster Rowhammer Attack

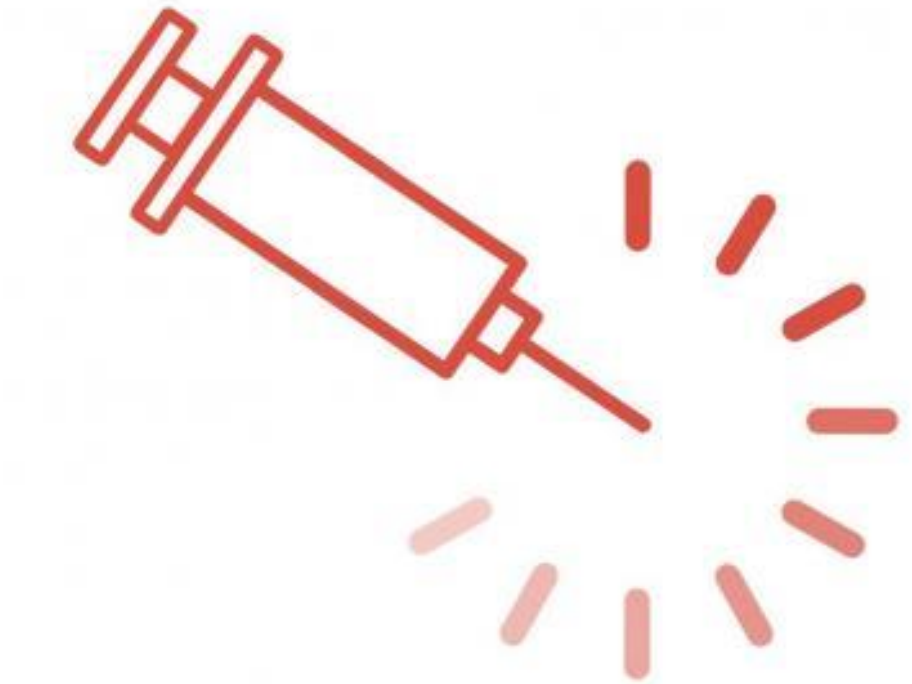




Additional Contributions - LVI

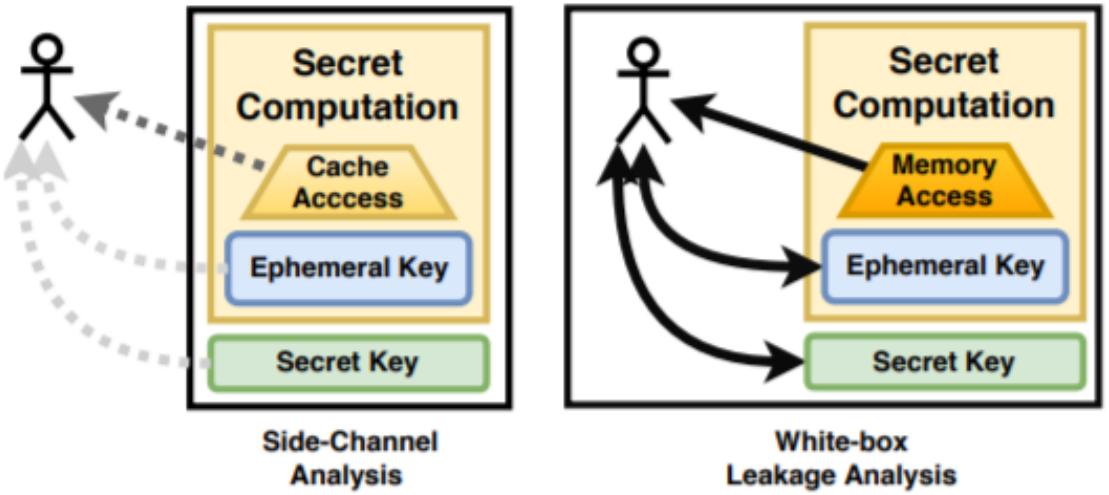
- Load Value Injection
- Inverse-meltdown attack
- Transient control and data flow hijacking

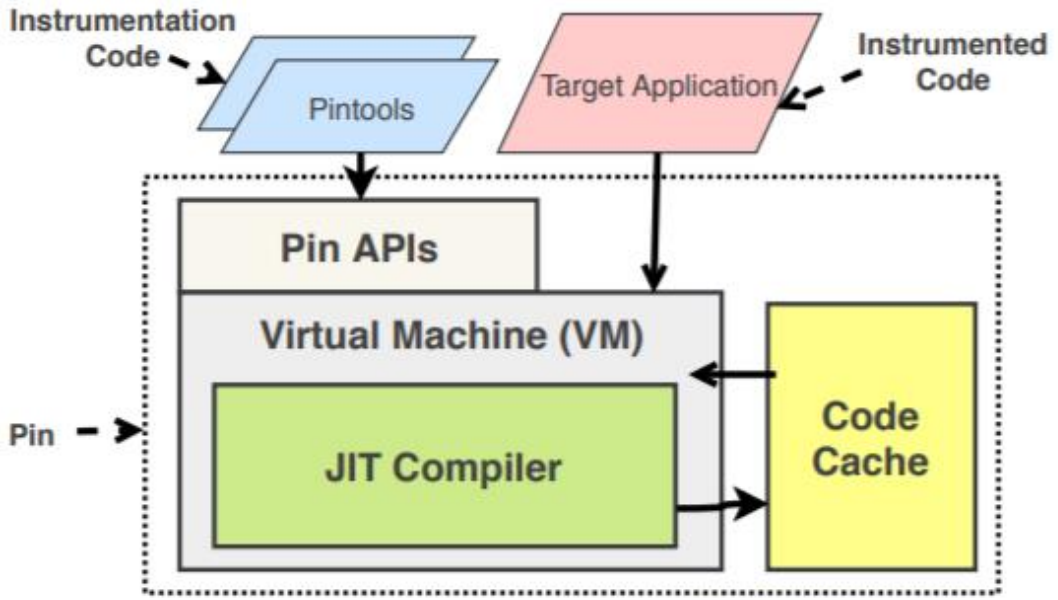




Additional Contributions - MicroWalk

- A Frameork for finding software-based side channel leakages in binaries
- Based on dynamic binary Instrumentation and a white-box analysis model





Summary of Contributed Publications

1) **D Moghimi**, B Sunar, T Eisenbarth, N Heninger. *"TPM-Fail: TPM meets Timing and Lattice Attacks"* USENIX Security 2020.

2) **D Moghimi**, M Lipp, B Sunar, M Schwarz. "Medusa: Microarchitectural Data Leakage via Automated Attack Synthesis" USENIX Security 2020.

3) **D Moghimi**, J Van Bulck, N Heninger, F Piessens, B Sunar. "CopyCat: Controlled Instruction-Level Attacks on Enclaves" USENIX Security 2020.

4) Z Weissman, T Tiemann, **D Moghimi**, E Custodio, T Eisenbarth, B Sunar. "JackHammer: Efficient Rowhammer on Heterogeneous FPGA-CPU Platforms" TCHES 2020.

5) J Van Bulck, **D Moghimi**, M Schwarz, M Lipp, M Minkin, D Genkin, Y Yarom, B Sunar, D Gruss, F Piessens."LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection" IEEE S&P 2020.

6) C Canella, D Genkin, L Giner, D Gruss, M Lipp, M Minkin, **D Moghimi**, F Piessens, M Schwarz, B Sunar, J Van Bulck. "Fallout: Leaking Data on Meltdown-resistant CPUs" CCS 2019.

7) M Schwarz, M Lipp, **D Moghimi**, J Van Bulck, J Stecklina, T Prescher, D Gruss. "ZombieLoad: Cross-Privilege-Boundary Data Sampling" CCS 2019.

8) S Islam, A Moghimi, I Bruhns, M Krebbel, B Gulmezoglu, T Eisenbarth, B Sunar. "SPOILER: Speculative Load Hazards Boost Rowhammer and Cache Attacks" USENIX Security 2019.

9) **A Moghimi**, J Wichelmann, T Eisenbarth, B Sunar. "MemJam: A False Dependency Attack against Constant-Time Crypto Implementations" (Extended Version) IJPP 2019.

10) J Wichelmann, **A Moghimi**, T Eisenbarth, B Sunar. "MicroWalk: *A Framework for Finding Side Channels in Binaries"* ACSAC 2018.

11) F Dall, G De Micheli, T Eisenbarth, D Genkin, N Heninger, **A Moghimi**, Y Yarom. *"CacheQuote: Efficiently Recovering Long-term Secrets of SGX EPID via Cache Attacks"* TCHES 2018.

12) **A Moghimi**, T Eisenbarth, B Sunar. *"MemJam: A False Dependency Attack against Constant-Time Crypto Implementations in SGX"* CT-RSA 2018.

13) **A Moghimi**, G Irazoqui, T Eisenbarth. *"CacheZoom: How SGX Amplifies The Power of Cache Attacks"* CHES 2017.

Coordinated Disclosure

- Crpytographic Libraries:
  - Intel IPP (CVE-2018-12155, CVE-2018-3691)
  - WolfSSL (CVE-2019-1996{0-3})
  - OpenSSL and Libgcrypt (No CVE available).
- Trusted Platform Modules
  - Intel fTPM (CVE-2019-11090)
  - STMicrolectronics (CVE-2019-16863)
- Intel CPUs
  - Fallout (CVE-2018-12126)
  - SPOILER (CVE-2019-0162)
  - MemJam (No CVE)

Conclusion

- Improved understanding of the side-channel attack surface:
  - Software-based side-channel attacks are practical.
  - Future CPUs and cryptographic software are more secure.

Conclusion

- Improved understanding of the side-channel attack surface:
  - Software-based side-channel attacks are practical.
  - Future CPUs and cryptographic software are more secure.

- Proper threat modeling is crucial
  - These attacks apply across many different threat models (T0, T1, T2, T3).
  - Vulnerabilities occur because of porting a previous design to a different threat model, e.g. Intel SGX, Cryptographic Implementations

Conclusion

- Automated testing for CPU attacks (Transynther)
  - helps us to understand the root cause and impact of these issues better.
  - can be used to verify hardware mitigations.

Conclusion

- Automated testing for CPU attacks (Transynther)
  - helps us to understand the root cause and impact of these issues better.
  - can be used to verify hardware mitigations.

- Automated testing of software (MicroWalk)
  - helps us to identify vulnerable code at scale
  - reduces analysis effort for software security

Conclusion

- Automated testing for CPU attacks (Transynther)
  - helps us to understand the root cause and impact of these issues better.
  - can be used to verify hardware mitigations.

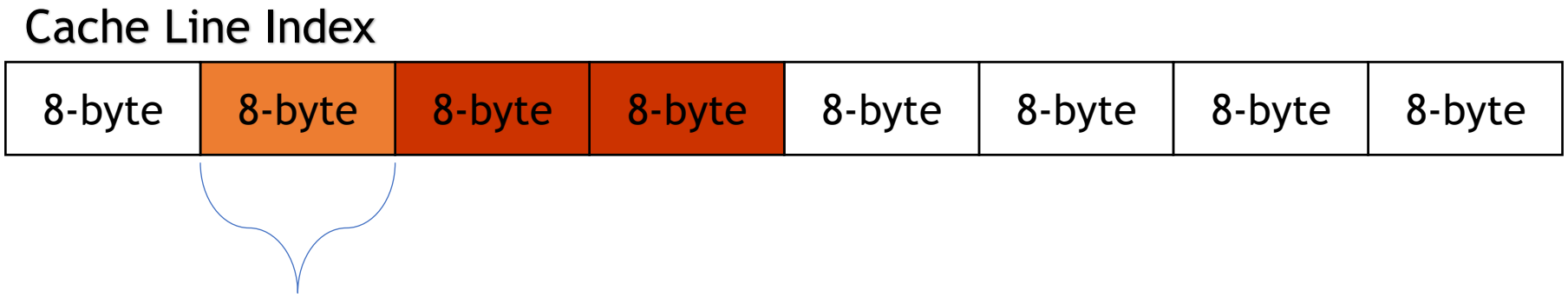
- Automated testing of software (MicroWalk)
  - helps us to identify vulnerable code at scale
  - reduces analysis effort for software security

- Hardware and software security are not separate problems.
  - covers cryptography, computer architecture and systems security.

THANKS

- Questions?

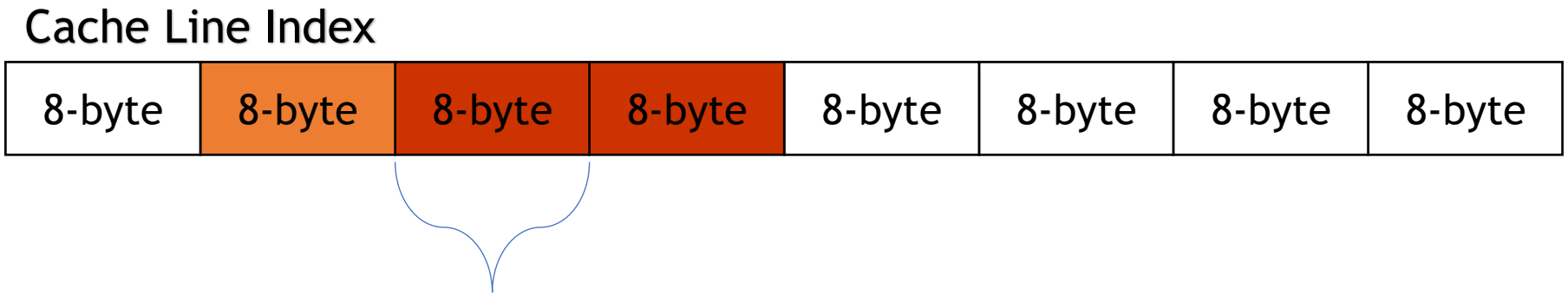
Medusa Attack - V1 Cache Indexing



Faulty Load

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

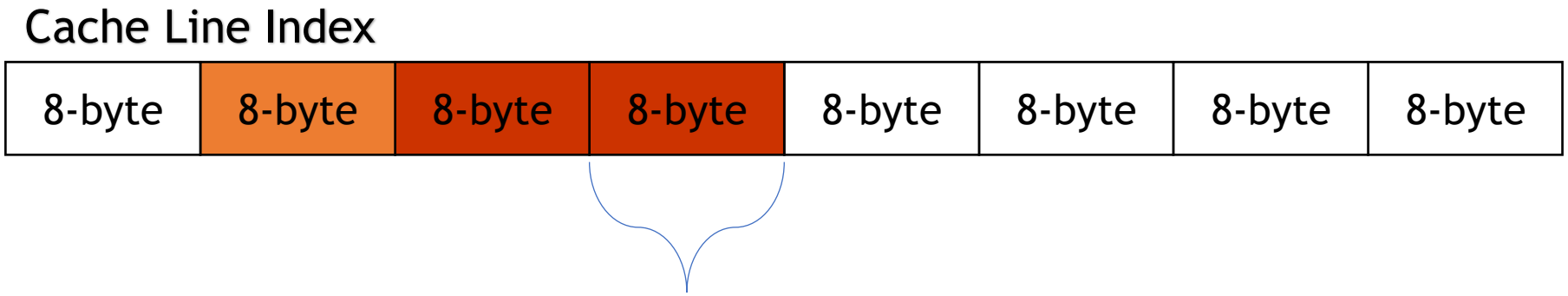
Medusa Attack - V1 Cache Indexing



Faulty Load

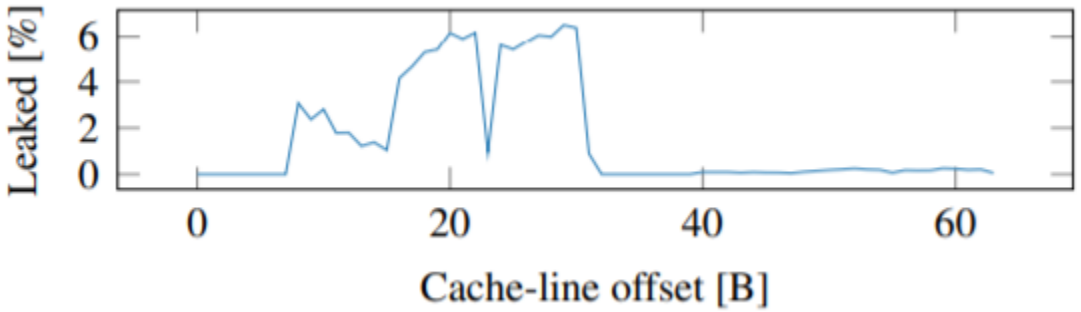
An invalid (Non-canon) address:
0x5550000000000008-20

Medusa Attack - V1 Cache Indexing

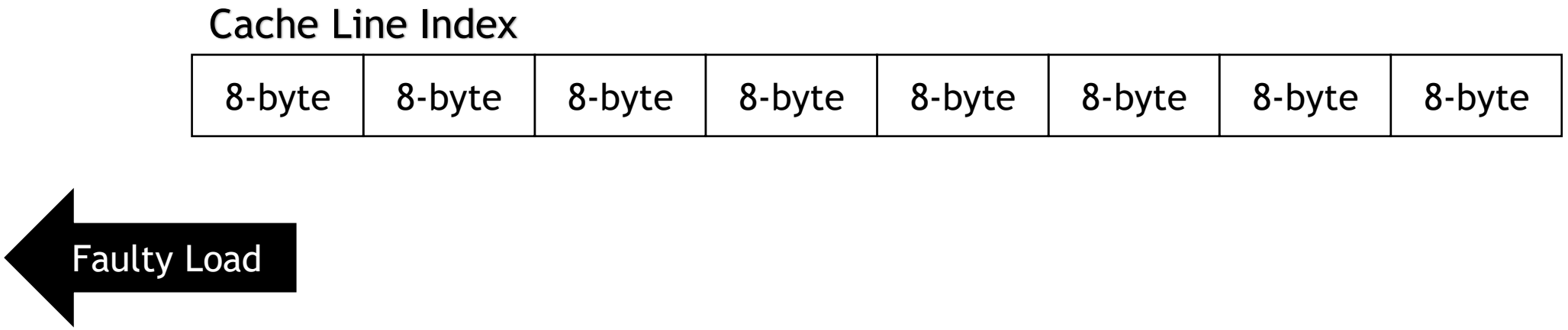


Faulty Load

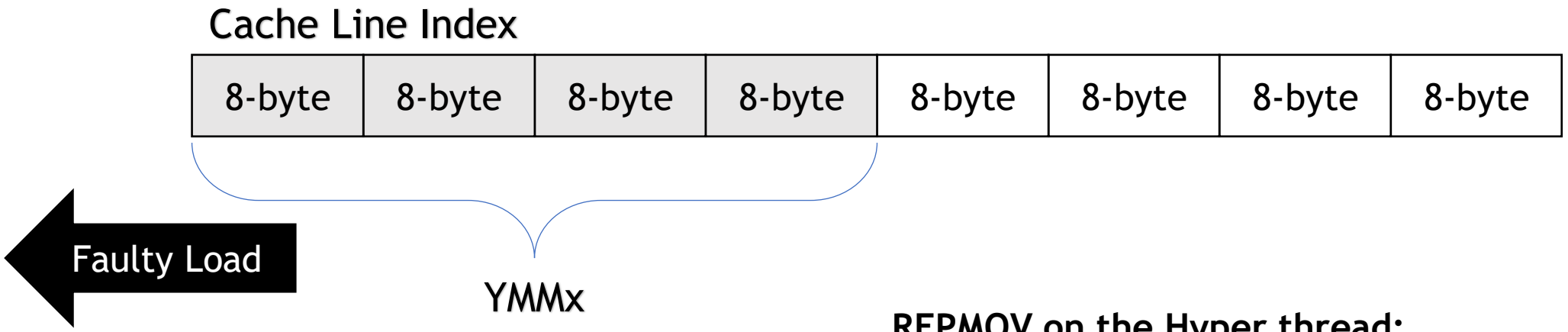
An invalid (Non-canon) address:
0x5550000000000008-20



Medusa Attack – V2 Unaligned S2L Forwarding



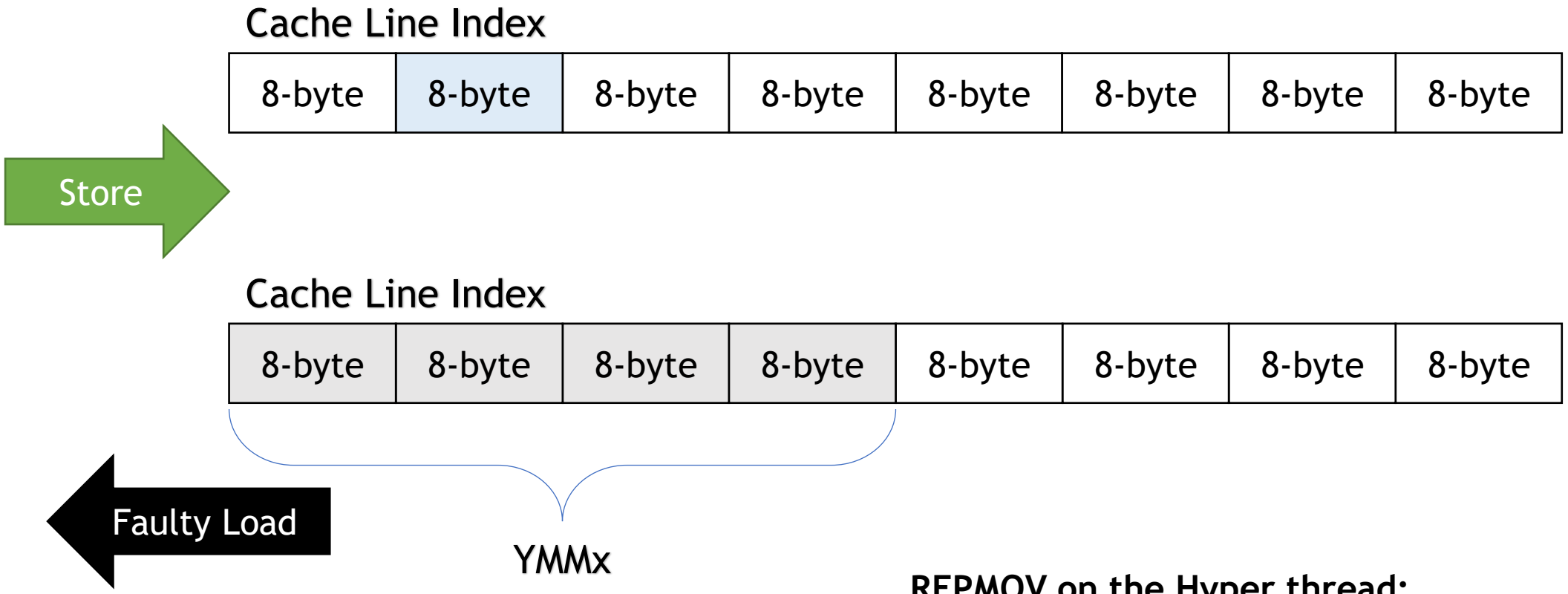
Medusa Attack – V2 Unaligned S2L Forwarding



REPMOV on the Hyper thread:

ABCDEFGH IJKLMNOP QRSTUVWX YZ…

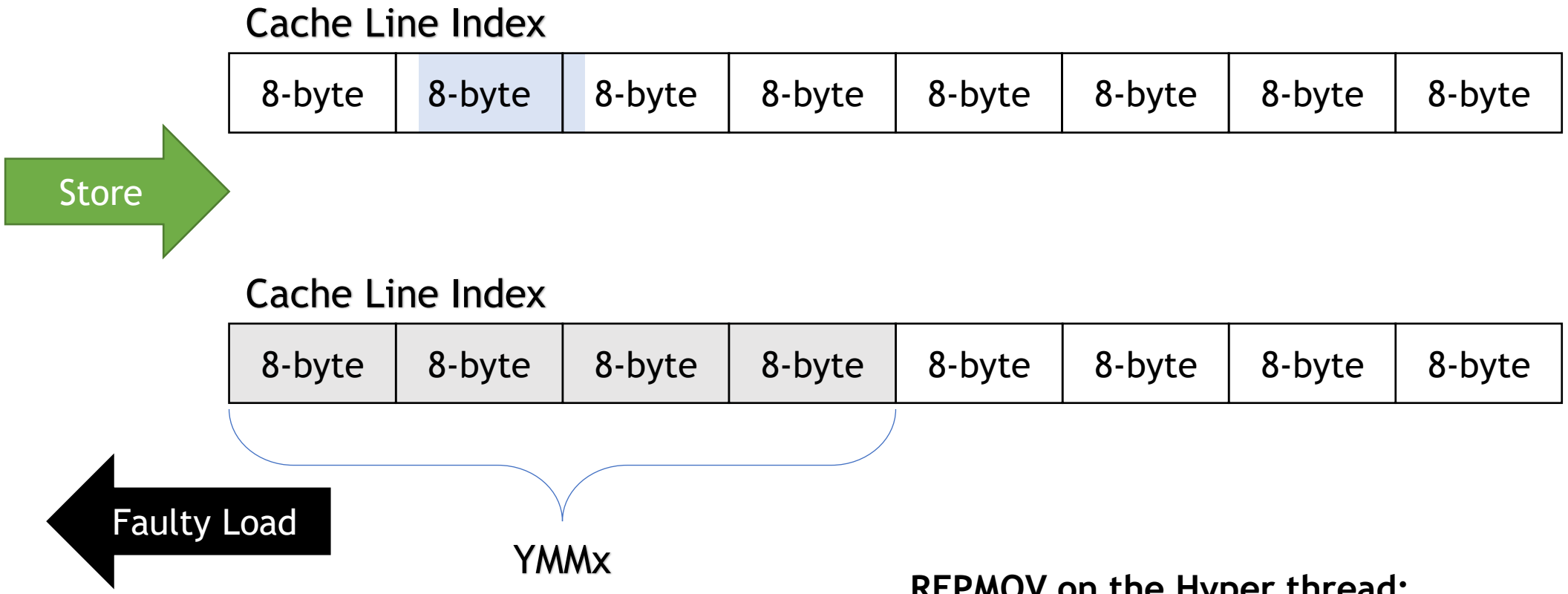
Medusa Attack – V2 Unaligned S2L Forwarding



REPMOV on the Hyper thread:

ABCDEFGH IJKLMNOP QRSTUVWX YZ…

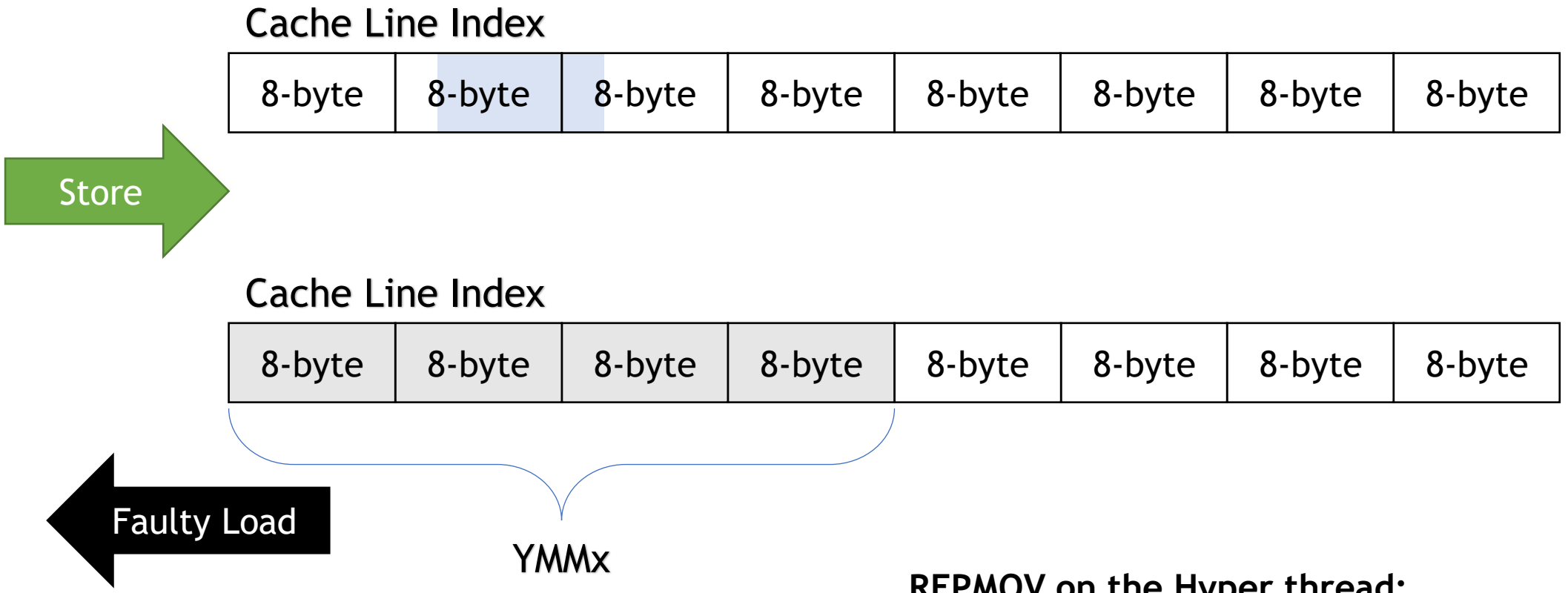
Medusa Attack – V2 Unaligned S2L Forwarding



REPMOV on the Hyper thread:

ABCDEFGH IJKLMNOP QRSTUVWX YZ…

Medusa Attack – V2 Unaligned S2L Forwarding

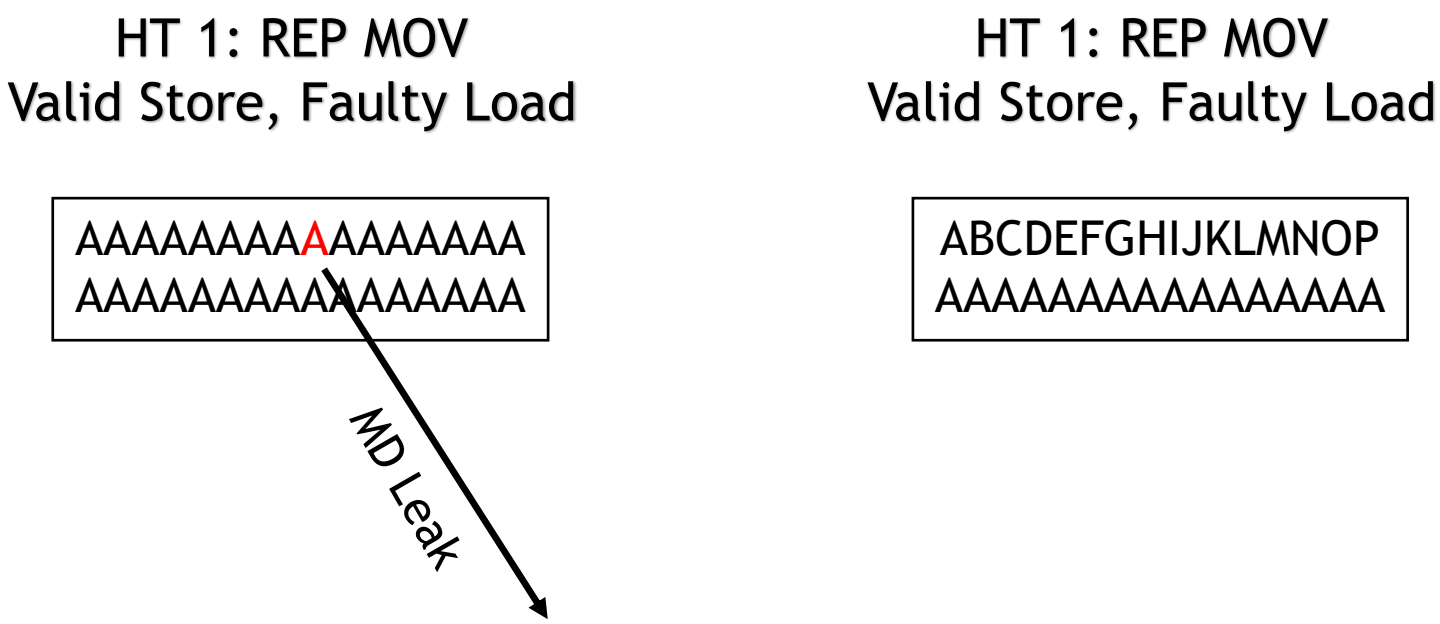


REPMOV on the Hyper thread:

ABCDEFGH IJKLMNOP QRSTUVWX YZ…

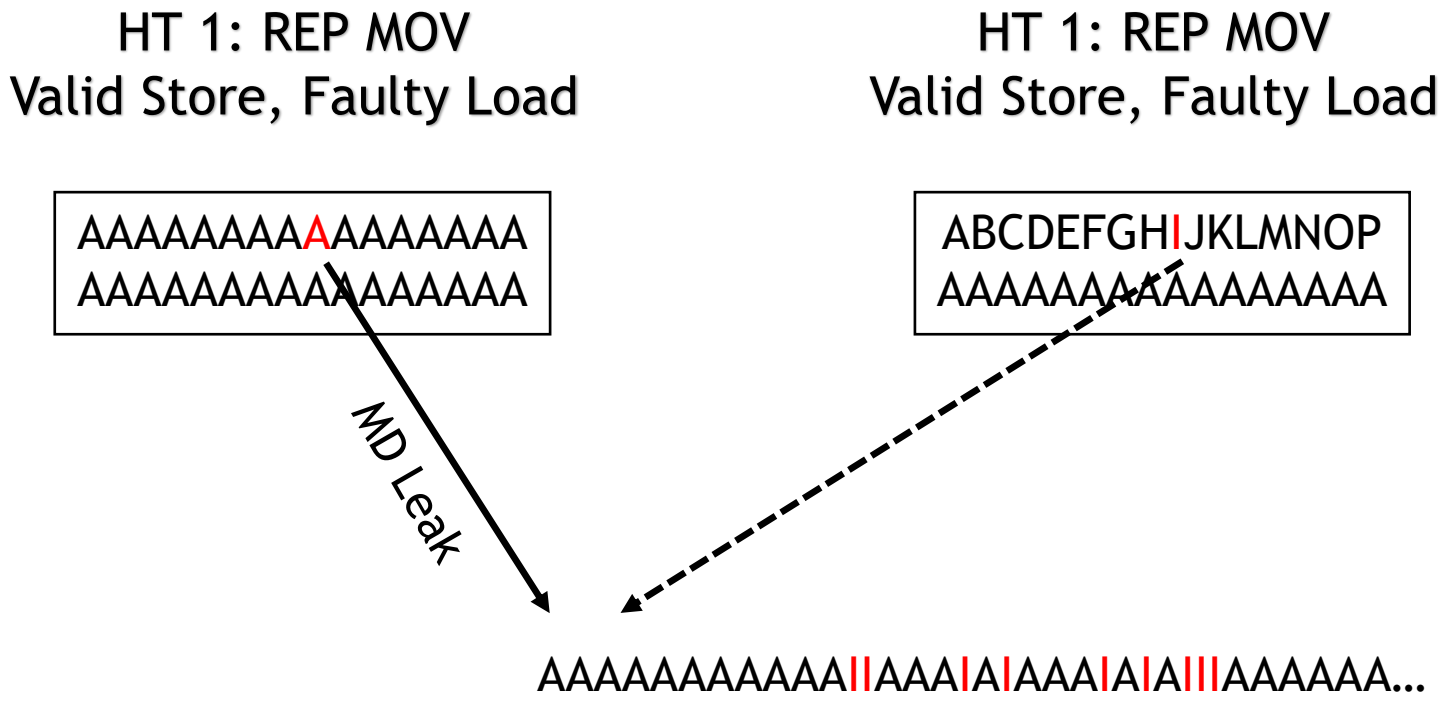
Medusa Attack – V3 Shadow *REP MOV*

- 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



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.

Trusted Computing Group - EAL 4+ Moderate

- https://trustedcomputinggroup.org/membership/certification/

TPM Security Evaluation

TCG members are required to demonstrate successful Common Criteria certification of their TPM product.

For the TPM 1.2 Family, the Common Criteria Security Assurance Level is at EAL4+ Moderate, in accordance to the PC Client TPM 1.2 Protection Profile by the TCG.

For the TPM 2.0 Family, the Common Criteria Security Assurance Level is at EAL4+ Moderate, in accordance to the PC Client TPM 2.0 Protection Profile by the TCG.

- https://trustedcomputinggroup.org/membership/certification/tpm-certified-products/

TPM Certified Products

TCG Certified Programs | TNC Certified Products List | Storage Certified Products List

Search:

Company Name	Product Name	Product Revision	Specification	Details	Security Evaluation	Cert. Status	Cert. Complete Date
STMicroelectronics	TPM ST33TPHF2X	1.256, 1.257, 2.256	Version 2.0 - Revision 1.38		Completed	Completed	2019.10.18
STMicroelectronics	TPM ST33GTPMA	3.256, 6.526	Version 2.0 - Revision 1.38		Completed	Completed	2019.10.18
Nuvoton Technologies Corporation (NTC)	TPM NPCT75x	7.4.0.0	Version 1.2 - Revision 116		Complete	Complete	2019.08.14
Nuvoton Technologies Corporation (NTC)	TPM NPCT75x	7.2.1.0	Version 2.0 - Revision 1.38		Complete	Complete	2019.01.18
Infineon Technologies	TPM SLI9670 TPM SLM9670	13.11	Version 2.0 - Revision 1.38		Complete	Complete	2018.12.18
Infineon Technologies	TPM SLB9670	7.85	Version 2.0 -		Complete	Complete	2018.10.29

- ST33TPHF2ESPI Data Brief: https://www.st.com/resource/en/data_brief/st33tphf2espi.pdf

- ST33TPHF2ESPI CC Evaluation: https://www.ssi.gouv.fr/uploads/2018/10/anssi-cible-cc-2018_41en.pdf

Intrinsic countermeasures for cryptographic algorithm against side channel attacks like timing attacks (TA), SPA and DPA.

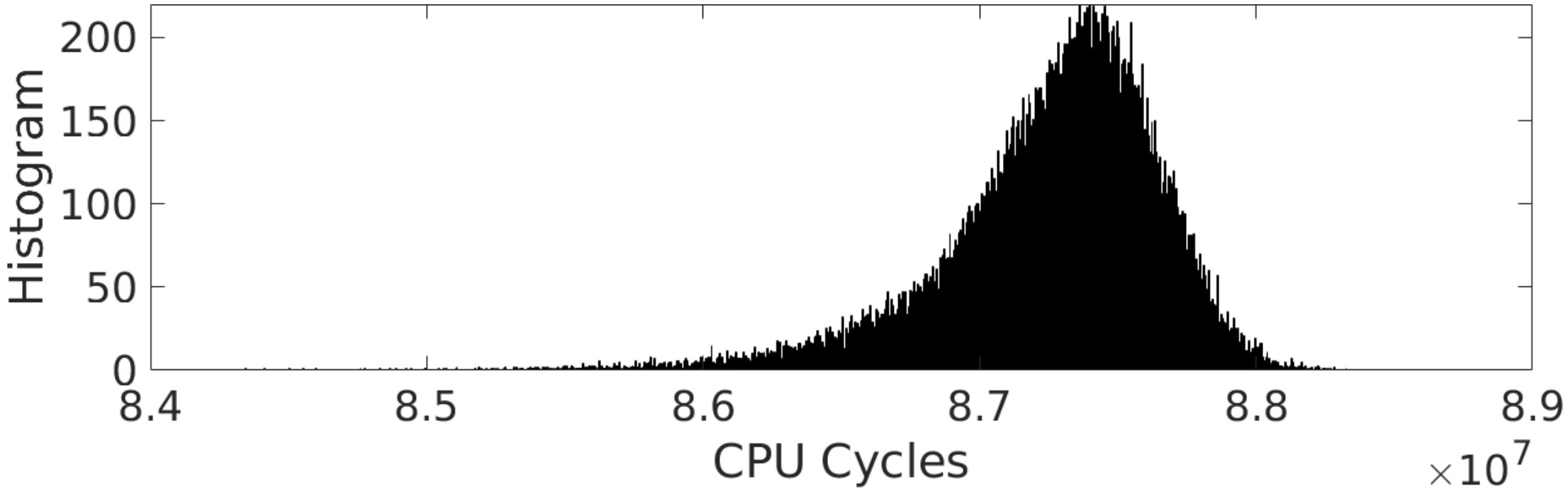
Detection of abnormal behavior of the following operational conditions:

- High voltage supply
- Glitches

Detection of abnormal TOE behavior:

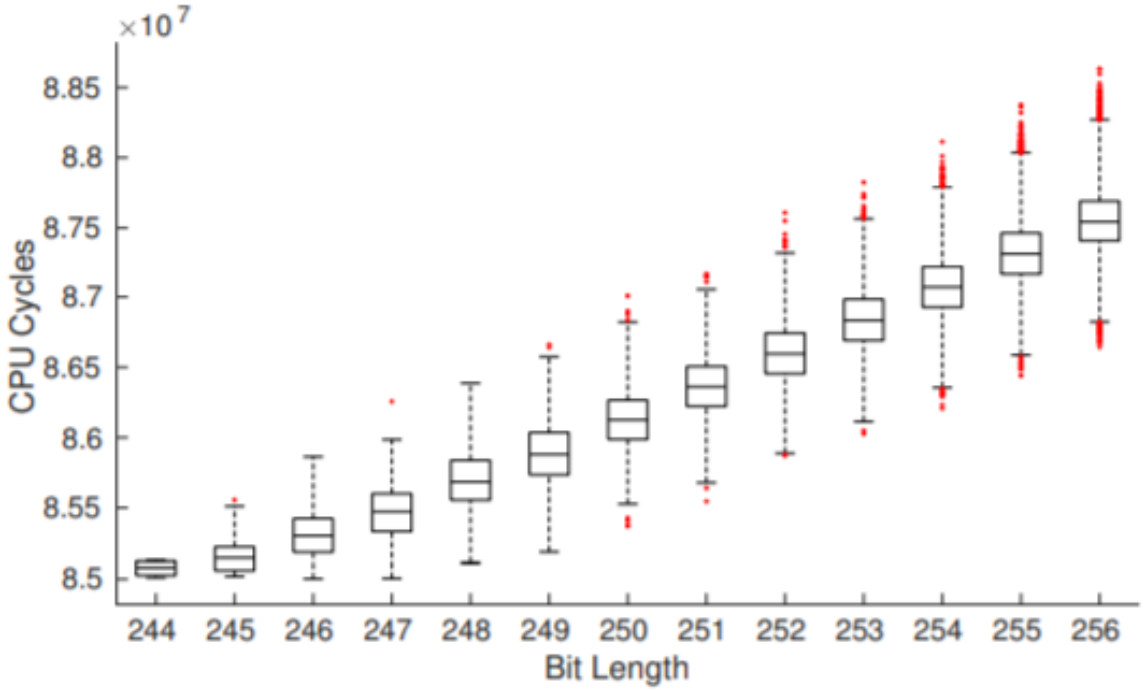
- MPU error
- TRNG failure

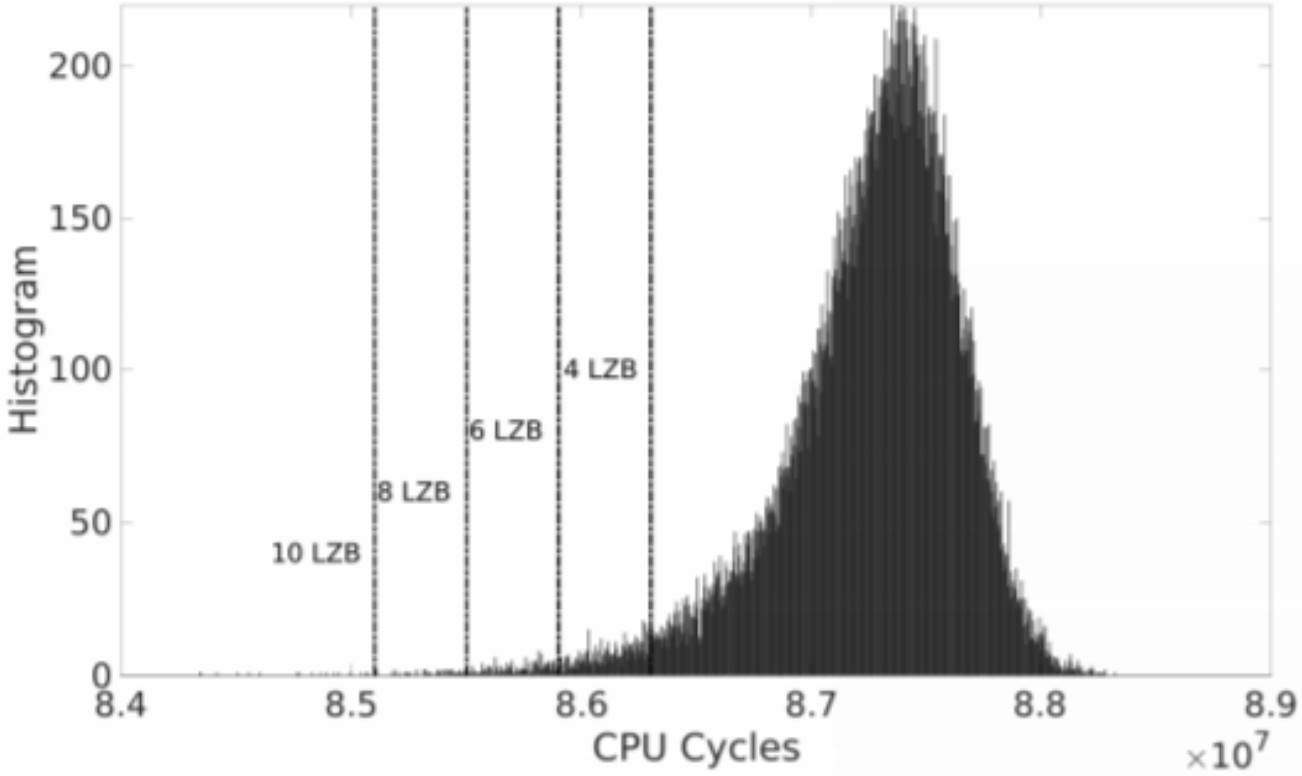
STMicroelectronics - ECDSA



High-resolution Timing Test – ECDSA Nonce Leakage (STM)

- STMicroelectronics' TPM: Bit-by-Bit Nonce Length Leakage





Responsible Disclosure (Ice Lake)

- MSBDS (Fallout) on Ice Lake
  - November 2019: Intel sent us an Ice Lake Machine
  - March 2019: Tested Transyther on the Ice Lake CPU
  - Mar 27, 2020: Reported MSBDS Leakage on Ice Lake
  - May 5, 2020: Intel Completed triage
    - MDS mitigations are not deployed properly
      - Chicken bits were not enabled for all mitigations.
      - OEMs shipped with old/wrong microcode.
    - Embargoed till July
  - July 13, 2020: MDS advisory and list of affected CPUs were updated.

MC Version	MC Date	Vulnerable	Leakage (bytes/s)		
			clflush	*lock inc*	Unmodified
0x32 (stock)	2019-07-05	✓	577.87	754.99	1.58
0x36	2019-07-18	✓	148.24	529.84	0.62
0x46	2019-09-05	✓	130.15	695.80	0.11
0x48	2019-09-12	✓	271.69	620.07	0.59
0x50	2019-10-27	✓	96.54	542.10	0.25
0x56	2019-11-05	✓	145.46	751.40	0.08
0x5a	2019-11-19	✓	532.40	645.32	0.70
0x66	2020-01-09	✗	0	0	0
0x70	2020-02-17	✗	0	0	0
0x82	2020-04-22	✗	0	0	0
0x86	2020-05-05	✗	0	0	0

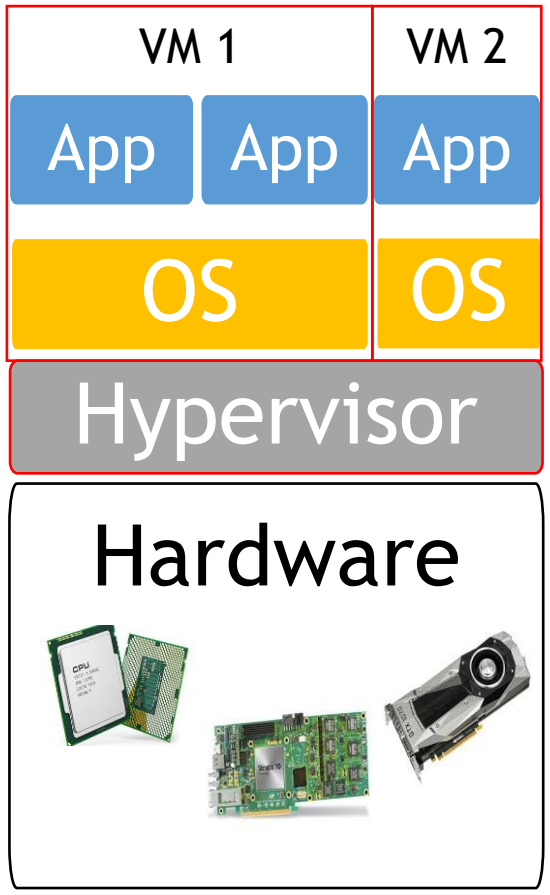
Processor	Stepping: All Unless Otherwise Noted	Microarchitectural Store Buffer Data Sampling CVE-2018-12126 / INTEL-SA-00233
		Intel Guidance: Microarchitectural Data Sampling Deep Dive
06_7EH	5	Hardware+MCU

Table 6: List of MDS-affected processors by Family/Model

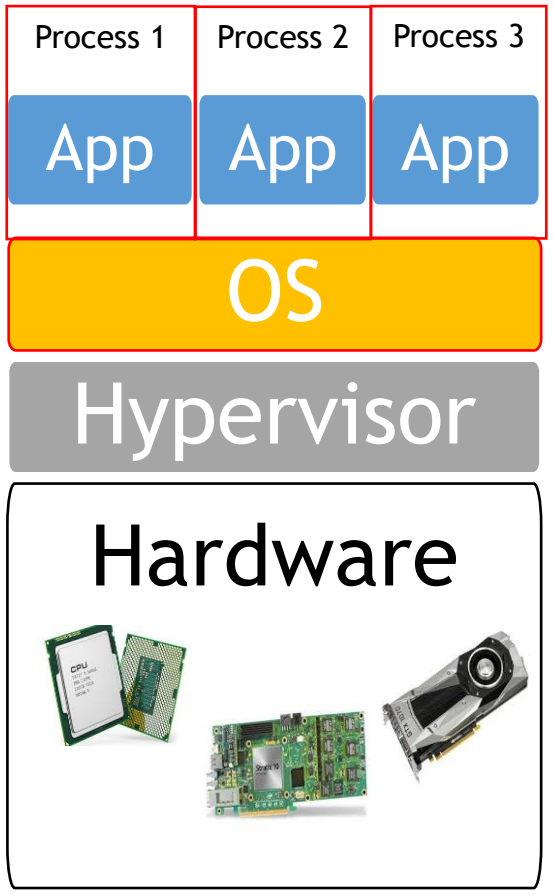
Family_Model	Step	Processor Families / Processor Number Series	MFBDS	MSBDS	MLPDS
06_7EH	5	10th Generation Intel® Core™ Processor Family based on Ice Lake (U, Y) microarchitecture	No	Yes	No

057	MDS_NO Bit in IA32_ARCH_CAPABILITIES MSR is Incorrectly Set
Problem	MDS_NO bit (bit 5) in IA32_ARCH_CAPABILITIES MSR (10Ah) is set, incorrectly indicating full activation of all MDS (microarchitectural data sampling) mitigations.
Implication	Due to this erratum, the IA32_ARCH_CAPABILITIES MDS_NO bit incorrectly reports the activation of all MDS mitigations actions.
Workaround	It is possible for the BIOS to contain a workaround for this erratum.
Status	For the steppings affected, refer to the Summary Table of Changes.

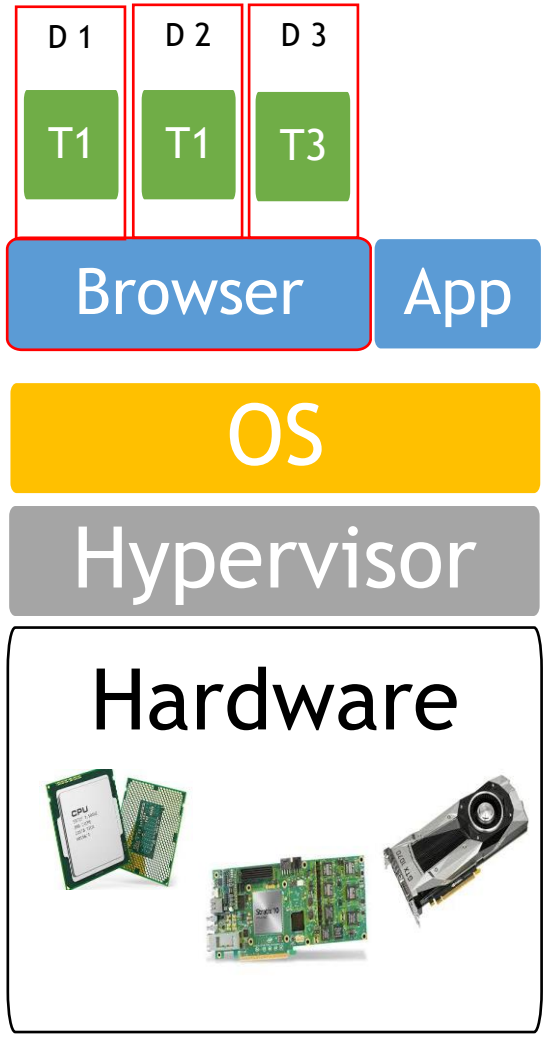
Motivation: Secure Isolation



Virtual Machines



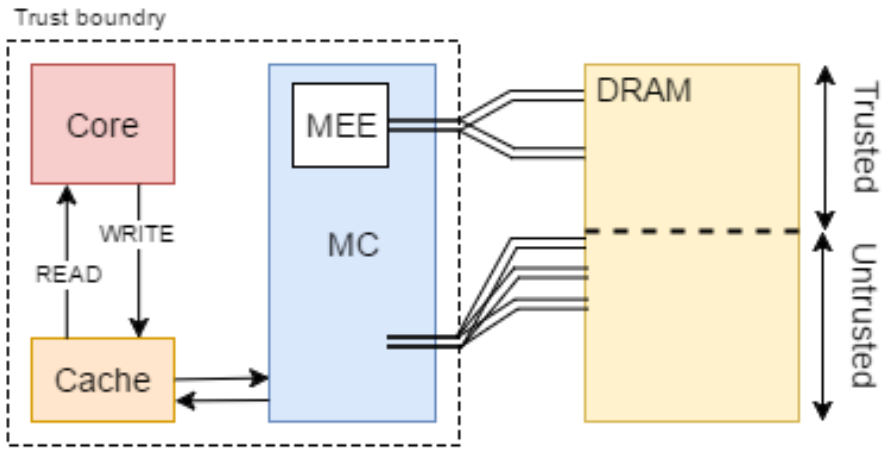
Process-Level Isolation

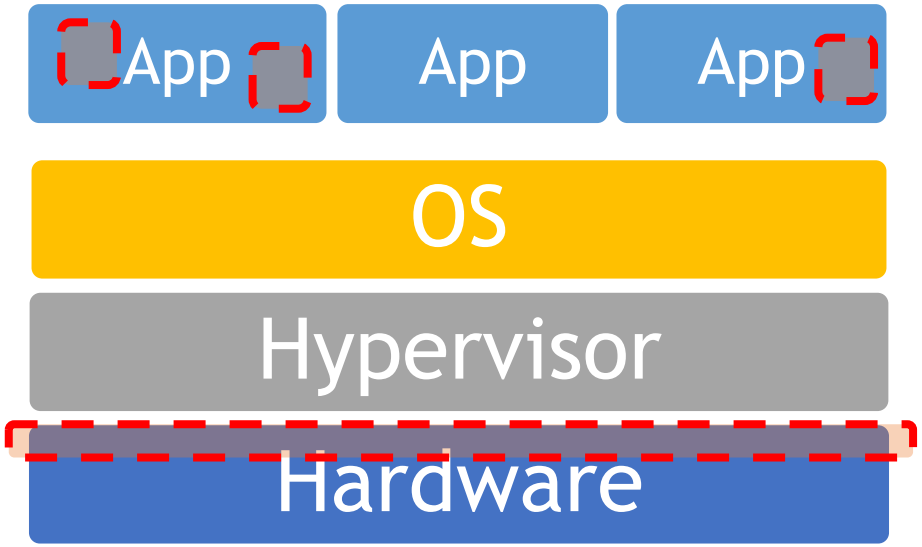


In-process Isolation

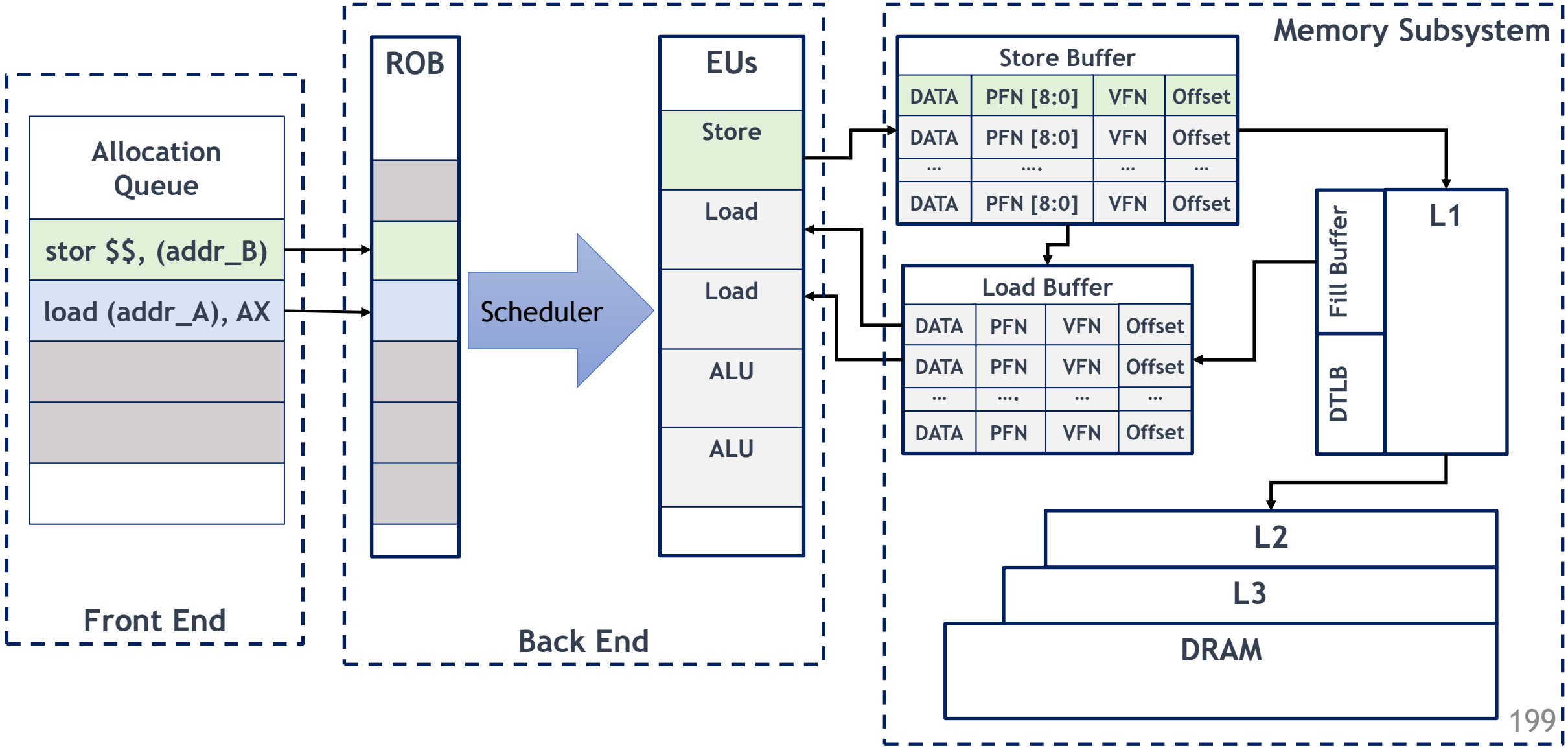
Trusted Execution Environment (TEE) - Intel SGX

- Intel Software Guard eXtensions (SGX)
- **Enclave:** A hardware protected user-level software module
  - Mapped by the operating system
  - Loaded by the user program
  - Authenticated and encrypted by CPU

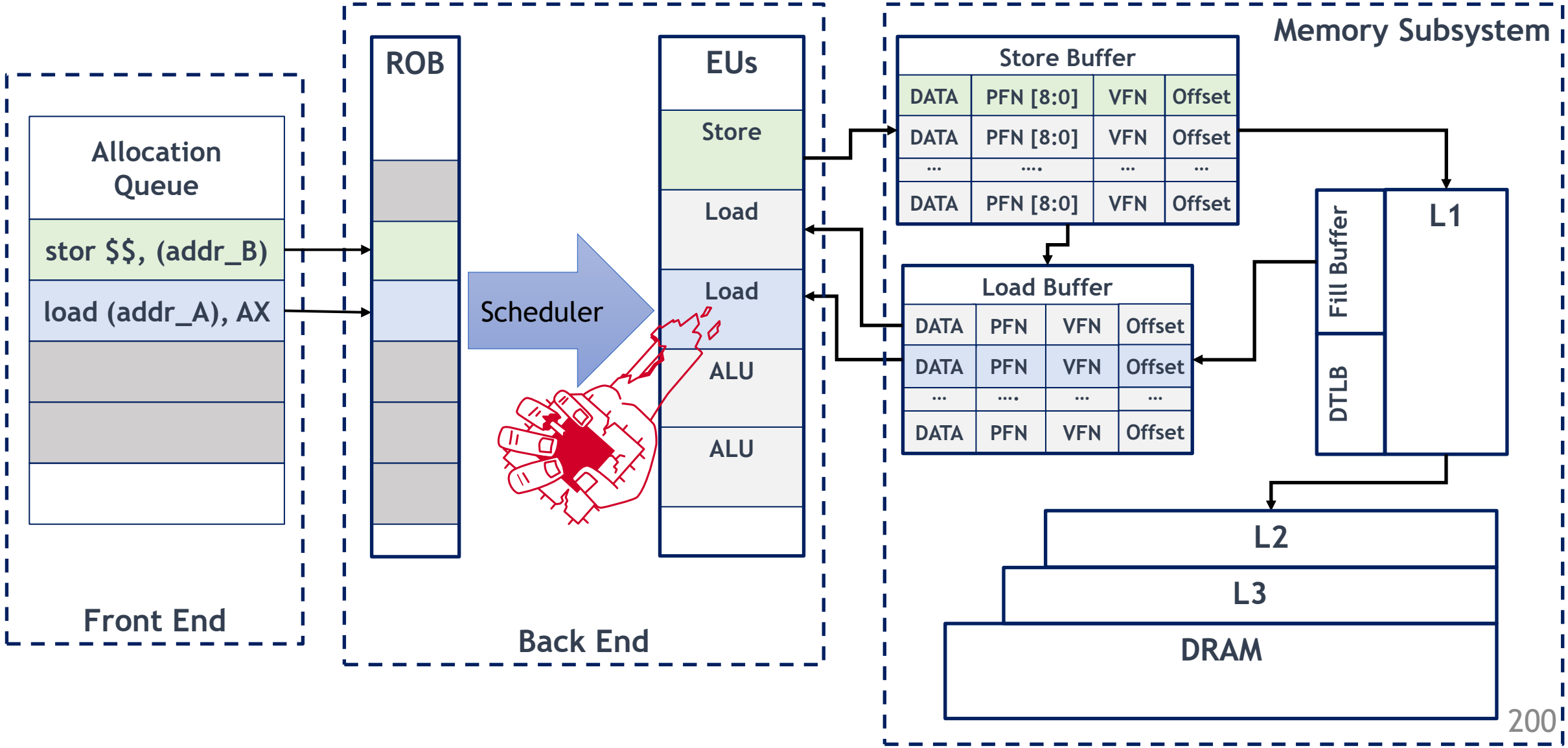




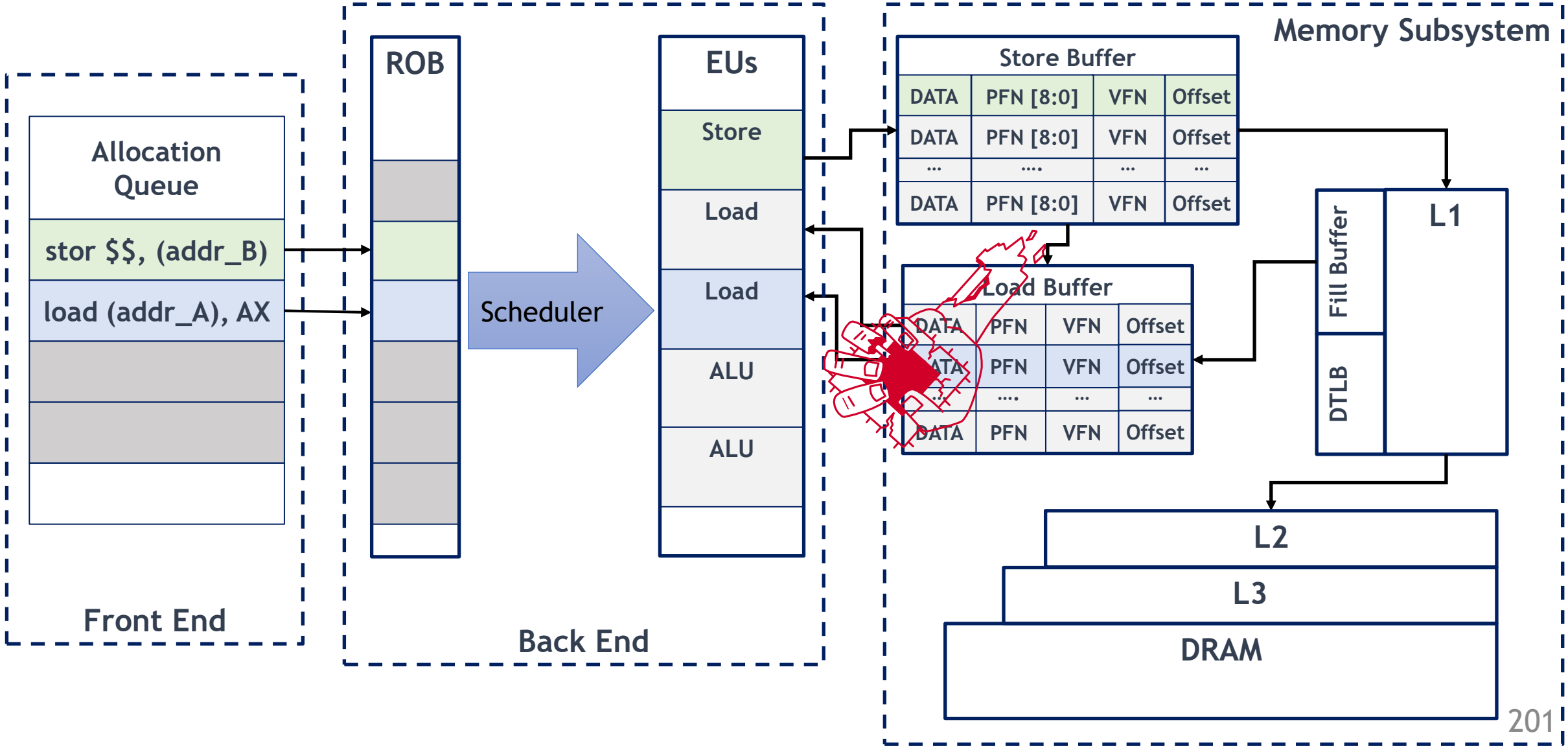
CPU Memory Subsystem - Hazard Recovery



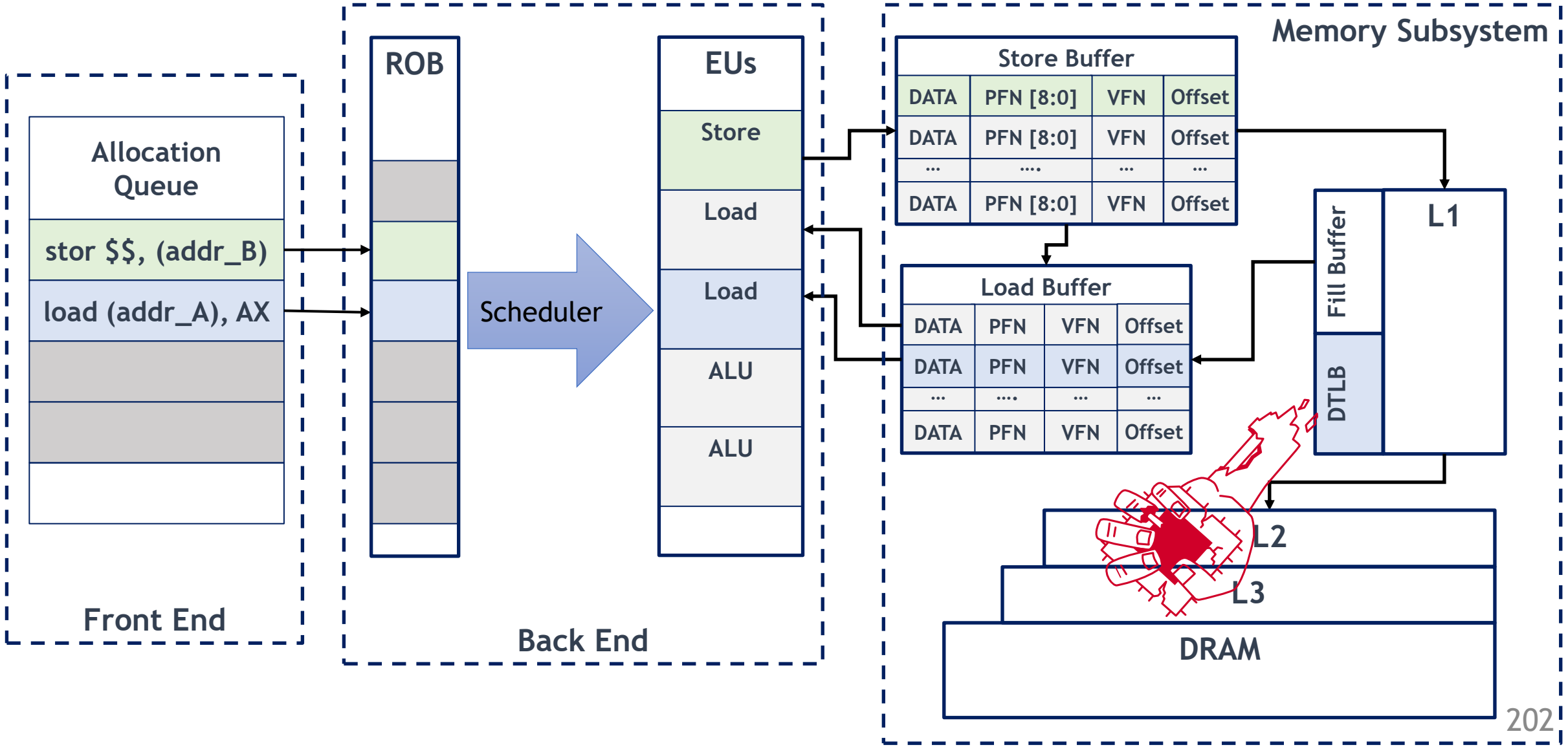
CPU Memory Subsystem - Hazard Recovery



CPU Memory Subsystem - Hazard Recovery



CPU Memory Subsystem - Hazard Recovery



How about other Crypto libraries?

- Libgcrypt uses a variant of BEEA
  - Single trace attack on DSA, Elgamal, ECDSA, RSA Key generation
- OpenSSL uses BEEA for computing GCD
  - Single trace attack on RSA Key generation when computing $\gcd(q-1, p-1)$

	Operation (Subroutine)	Implementation	Secret Branch	Exploitable	Computation → Vulnerable Callers	Single-Trace Attack
WolfSSL	Scalar Multiply (`wc_ecc_mulmod_ex`)	Montgomery Ladder w/ Branches	✔	✔	$(k \times G) \rightarrow$ `wc_ecc_sign_hash`	✘
	Greatest Common Divisor (`fp_gcd`)	Euclidean (Divisions)	✔	✘	N/A	N/A
	Modular Inverse (`fp_invmod`)	BEEA	✔	✔	$(k^{-1} \bmod n) \rightarrow$ `wc_DsaSign`	✔
					$(q^{-1} \bmod p) \rightarrow$ `wc_MakeRsaKey`	✔
					$(e^{-1} \bmod \Lambda(N)) \rightarrow$ `wc_MakeRsaKey`	✔
Libgcrypt	Greatest Common Divisor (`mpi_gcd`)	Euclidean (Divisions)	✔	✘	N/A	N/A
	Modular Inverse (`mpi_invm`)	Modified BEEA [43, Vol II, §4.5.2]	✔	✔	$(k^{-1} \bmod n) \rightarrow$ `{dsa,elgamal}.c::sign,_gcry_ecc_ecdsa_sign`	✔
					$(q^{-1} \bmod p) \rightarrow$ `generate_{std,fips,x931}`	✔
					$(e^{-1} \bmod \Lambda(N)) \rightarrow$ `generate_{std,fips,x931}`	✔
OpenSSL	Greatest Common Divisor (`BN_gcd`)	BEEA	✔	✔	$gcd(q-1, p-1) \rightarrow$ `RSA_X931_derive_ex`	✔
	Modular Inverse (`BN_mod_inverse_no_branch`)	BEEA w/ Branches	✘	N/A	N/A	N/A
IPP Crypto	Greatest Common Divisor (`ippsGcd_BN`)	Modified Lehmer's GCD	✔	?	$gcd(q-1, e) \rightarrow$ `cpIsCoPrime`	N/A
				?	$gcd(p-1, q-1) \rightarrow$ `isValidPriv1_rsa`	N/A
	Modular Inverse (`cpModInv_BNU`)	Euclidean (Divisions)	✔	✘	N/A	N/A

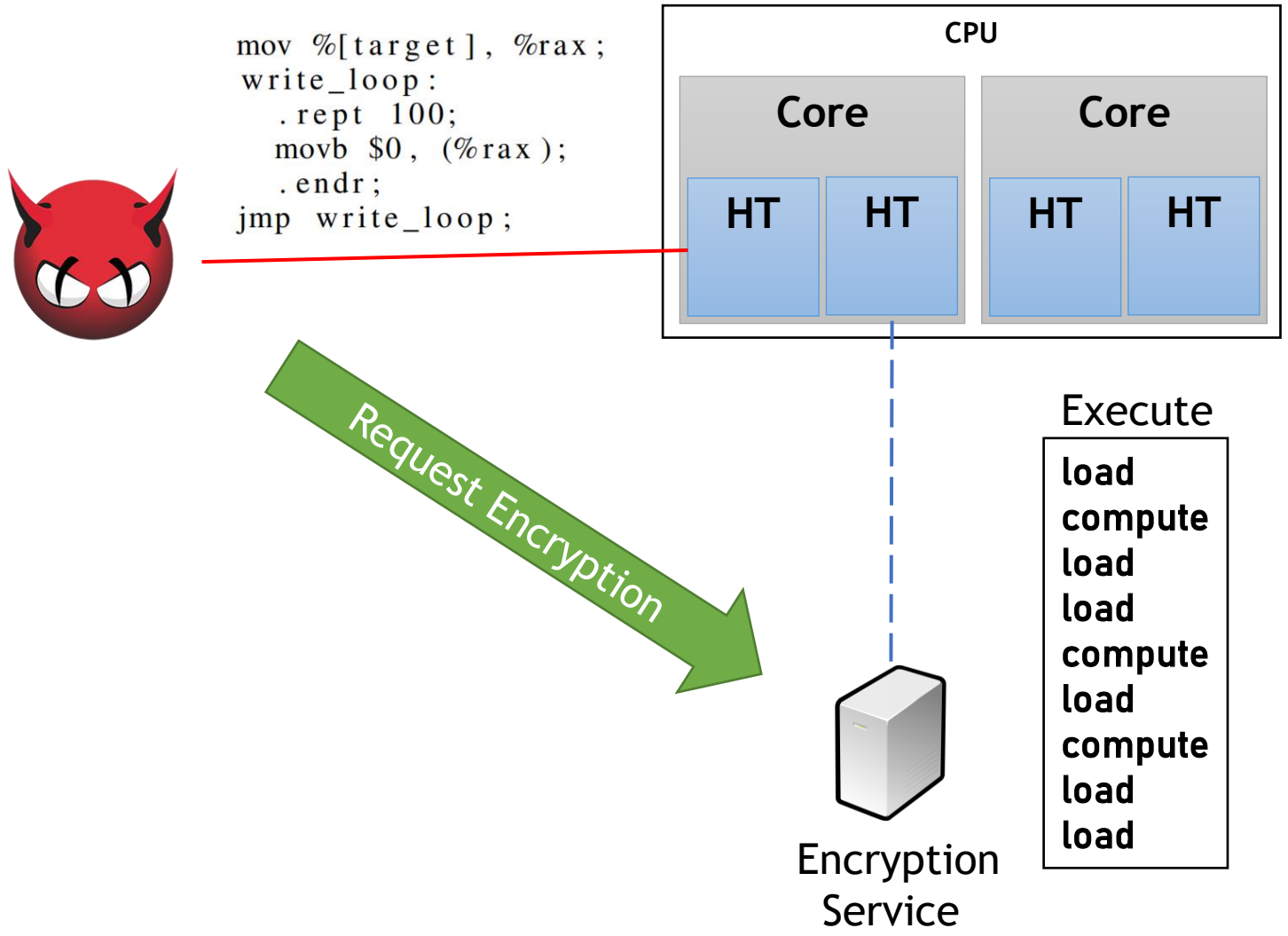
CopyCat on WolfSSL - Cryptanalysis

- Single-trace Attack during DSA signing: $k_{inv} = k^{-1} \, mod \, n$
  - Iterative over the entire recovered trace with n as input → k_{inv}
  - Plug k_{inv} in $s_1 = \ k_1^{-1}(h - r_1.x) mod \, n$ → get private key x

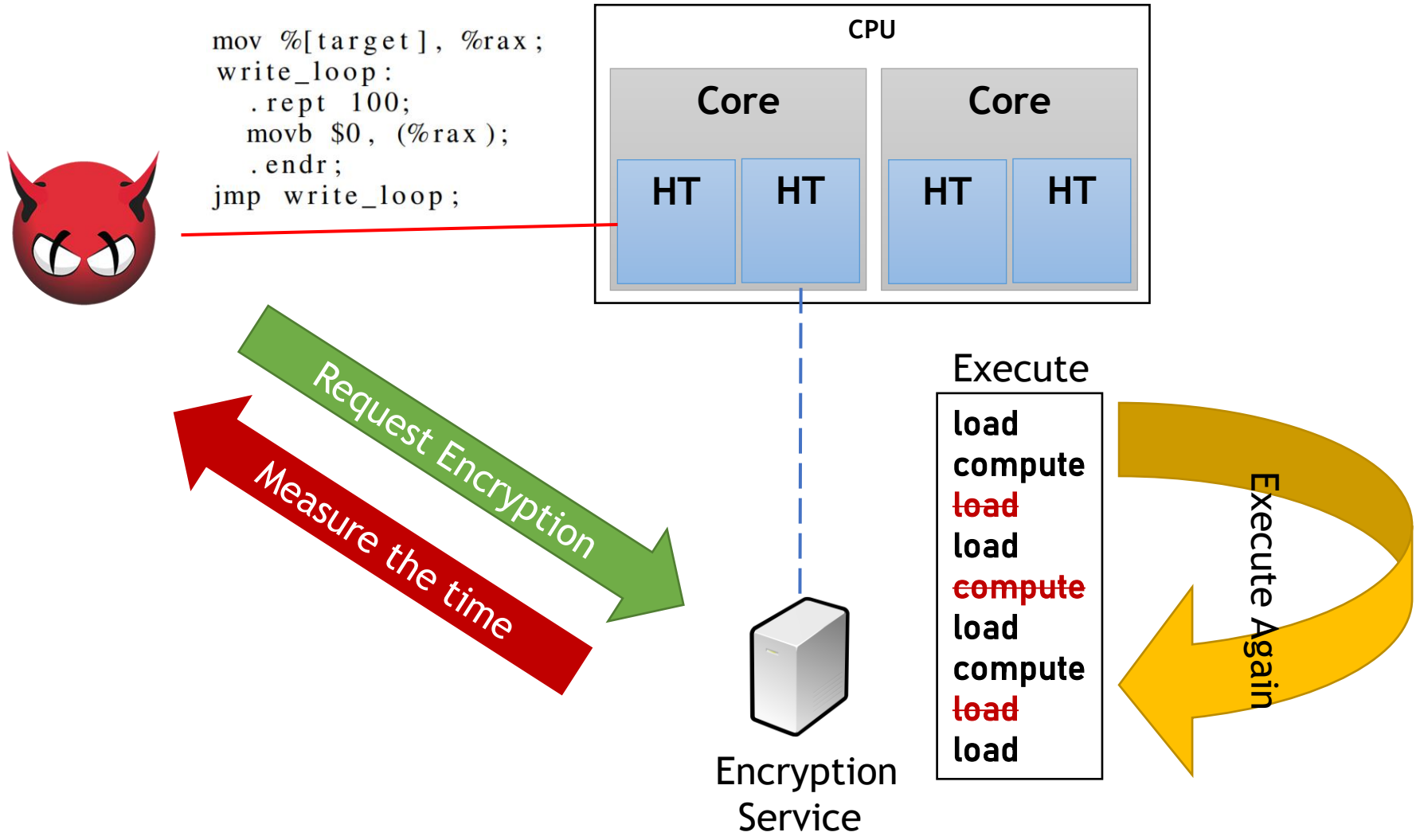
CopyCat on WolfSSL – Cryptanalysis Results

- Executed each attack 100 times.
- DSA $k^{-1}\ mod\ n$
  - Average 22,000 IRQs
  - 75 ms to iterate over an average of 6,320 steps
- RSA $q^{-1}\ mod\ p$
  - Average 106490 IRQs
  - 365 ms to iterate over an average of 39,400 steps
- RSA $e^{-1}\ mod\ \lambda(N)$
  - $e^{-1}\ mod\ \lambda(N)$
  - Average 230,050 IRQs
  - 800ms to iterate over an average of 81,090 steps
- Experimental traces always match the leakage model in all experiments → Successful single-trace key recovery

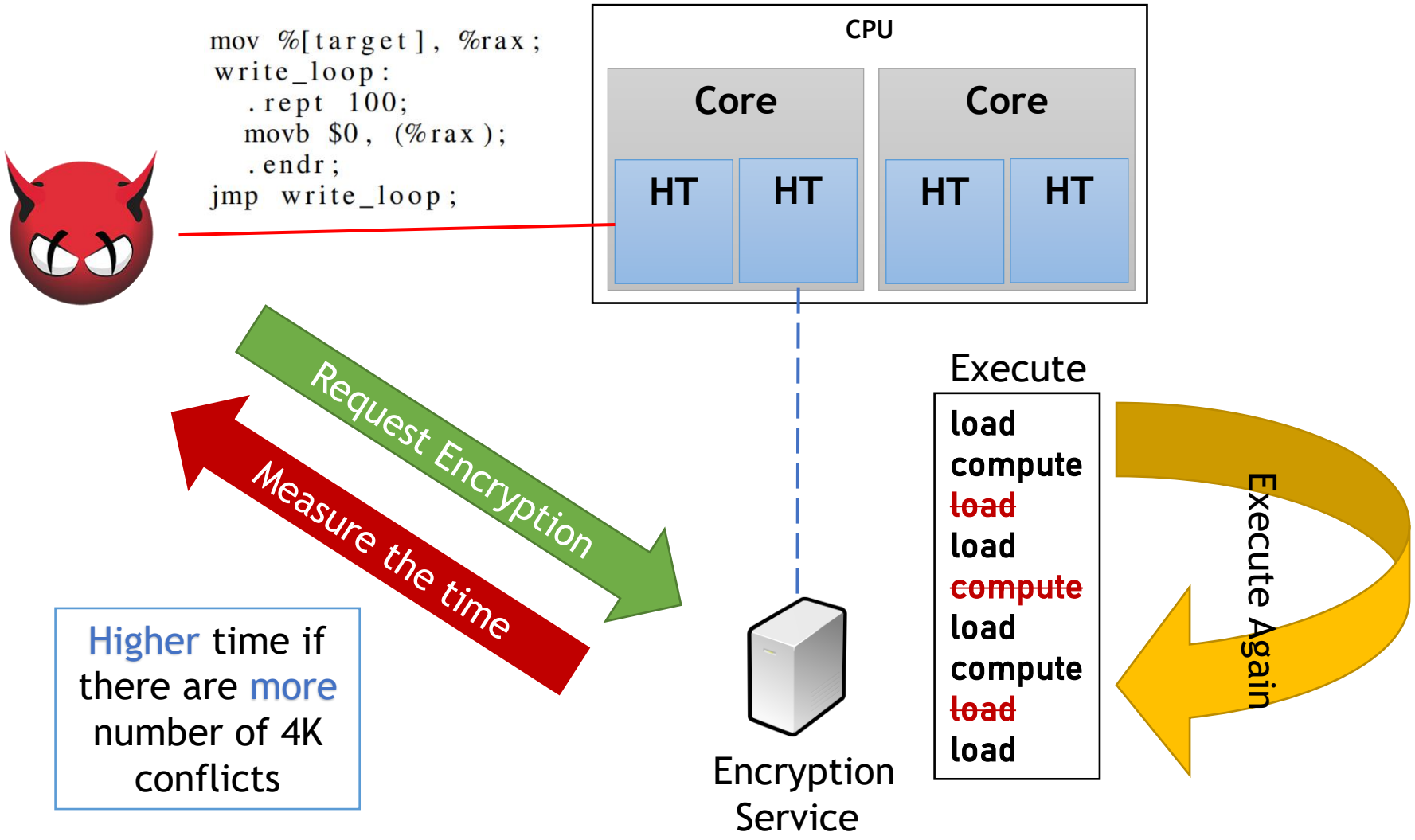
MemJam Attack Scenario



MemJam Attack Scenario

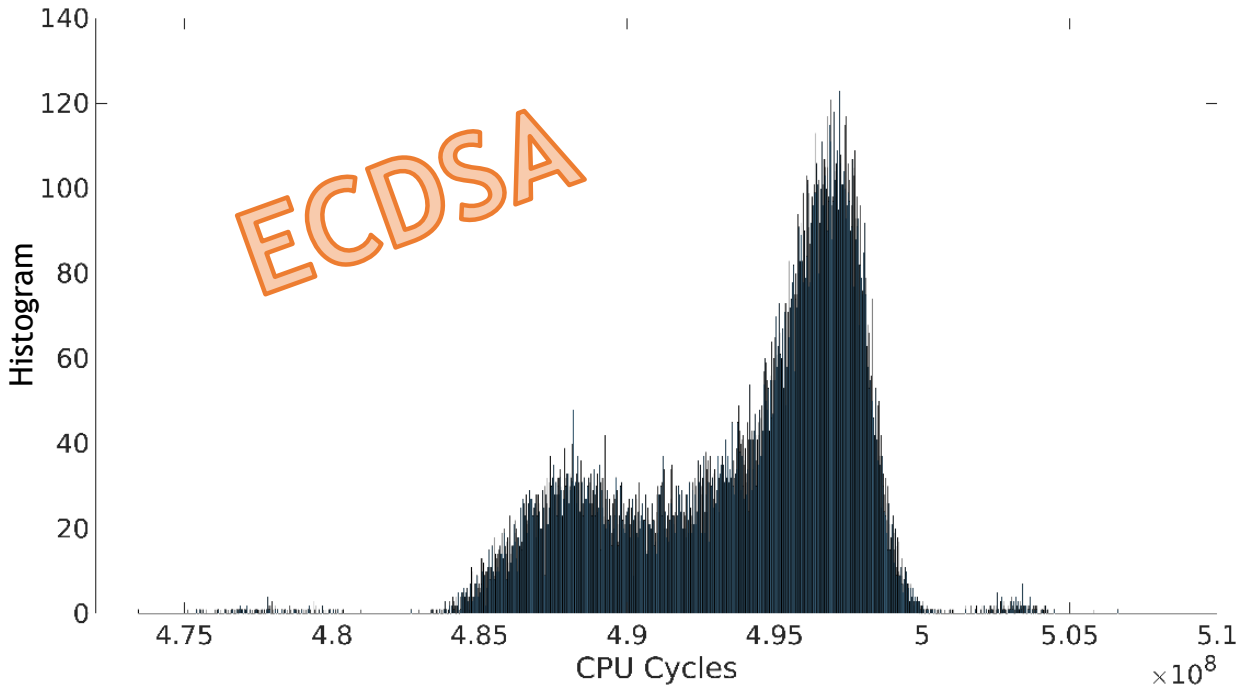


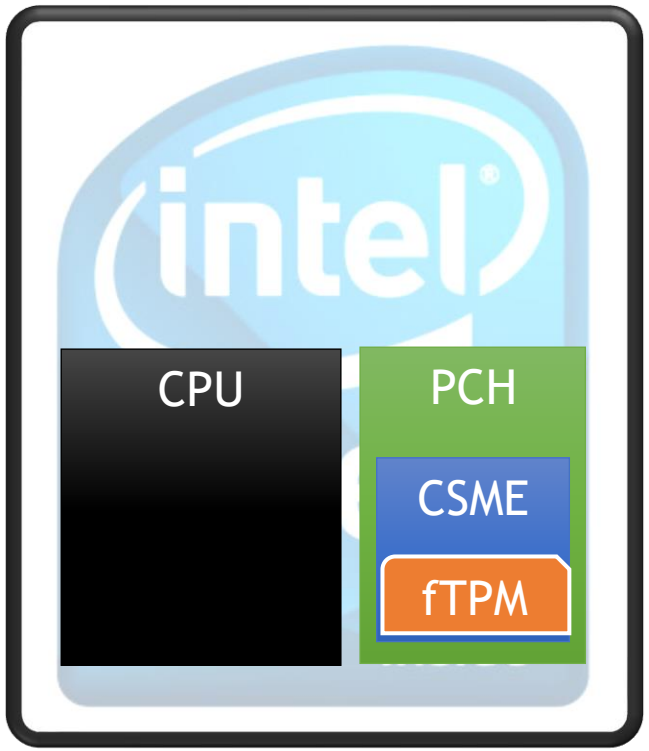
MemJam Attack Scenario



High-resolution Timing Test - Intel PTT (fTPM)

- Intel Platform Trust Technology (PTT)
  - Integrated firmware-TPM inside the CPU package
  - Runs on top of Converged Security and Management Engine (CSME)

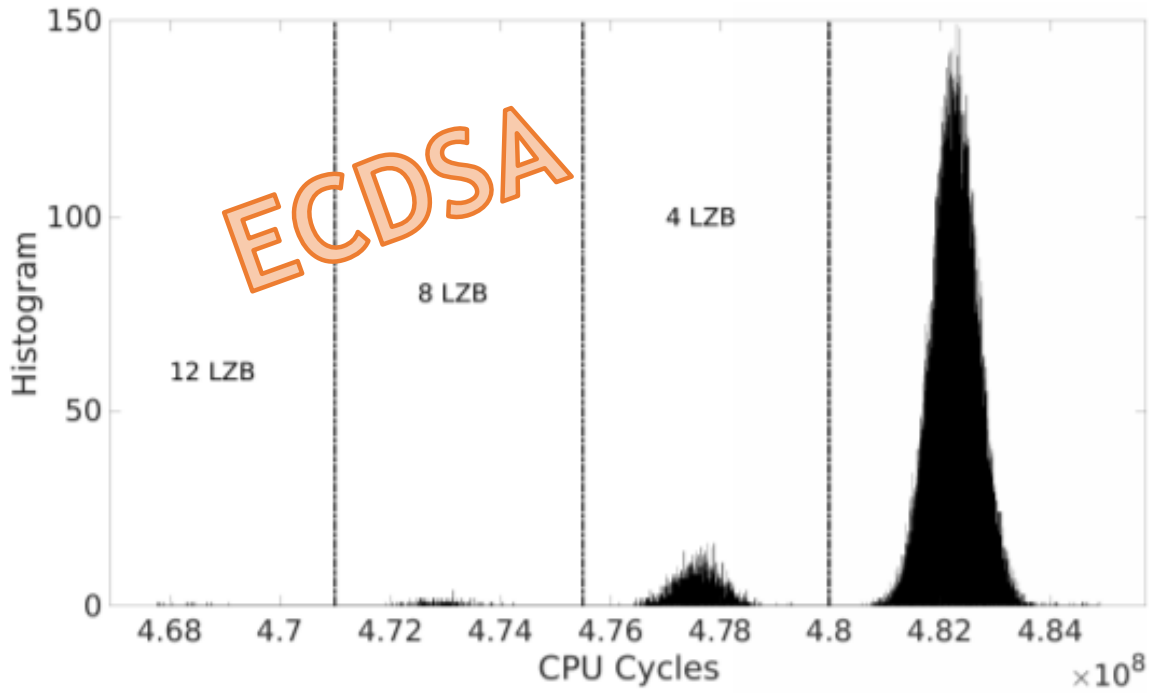


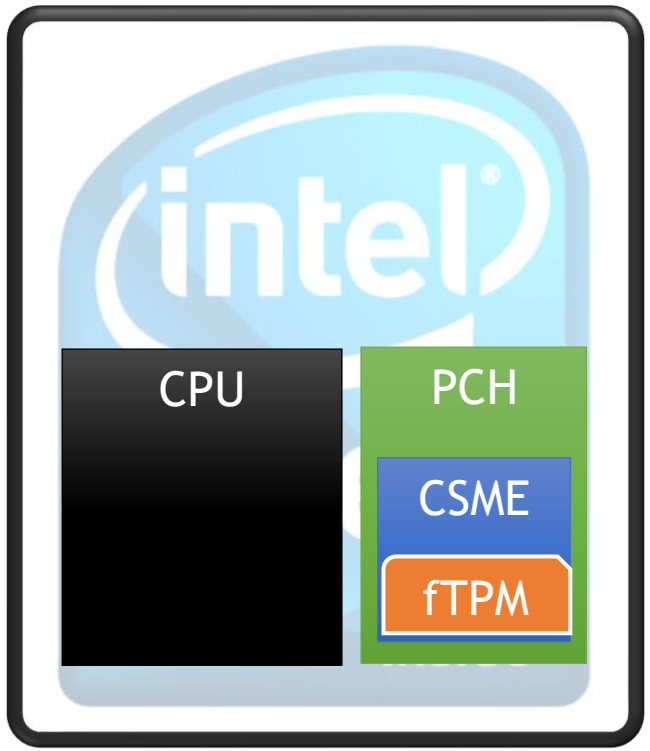


High-resolution Timing Test - Intel PTT (fTPM)

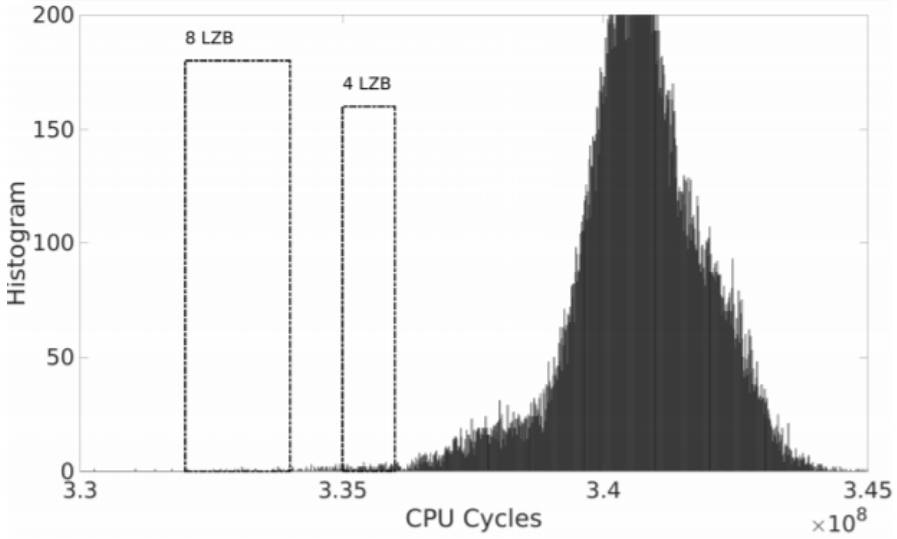
- Linux TPM Command Response Buffer (CRB) driver
- Kernel Driver to increase the Resolution

```
t = rdtsc();
iowrite32(CRB_START_INVOKE, &g_priv->regs_t->ctrl_start);
while((ioread32(&g_priv->regs_t->ctrl_start) &
      CRB_START_INVOKE) == CRB_START_INVOKE);
tscrequest[requestcnt++] = rdtsc() - t;
```

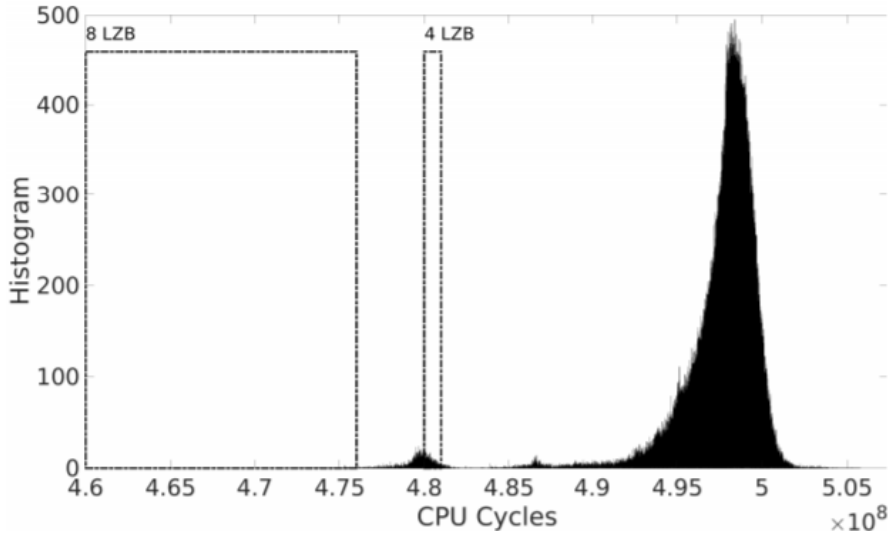


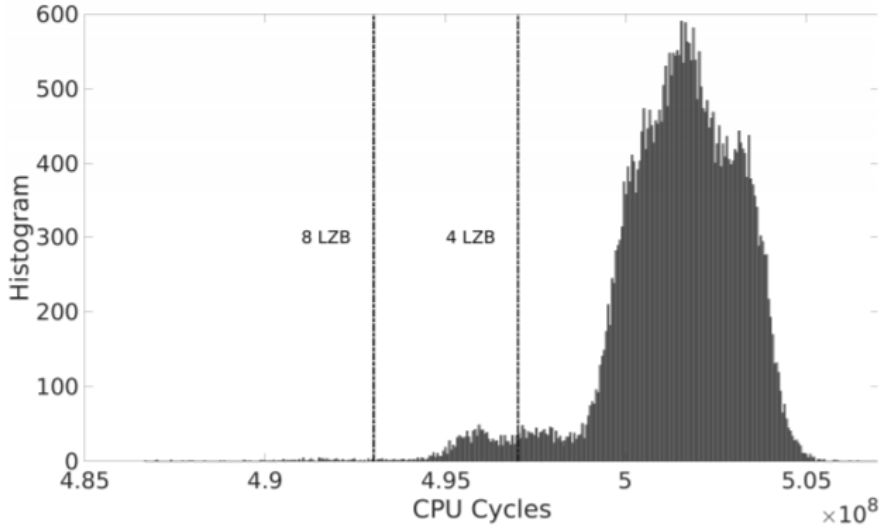


Remote StrongSwan VPN

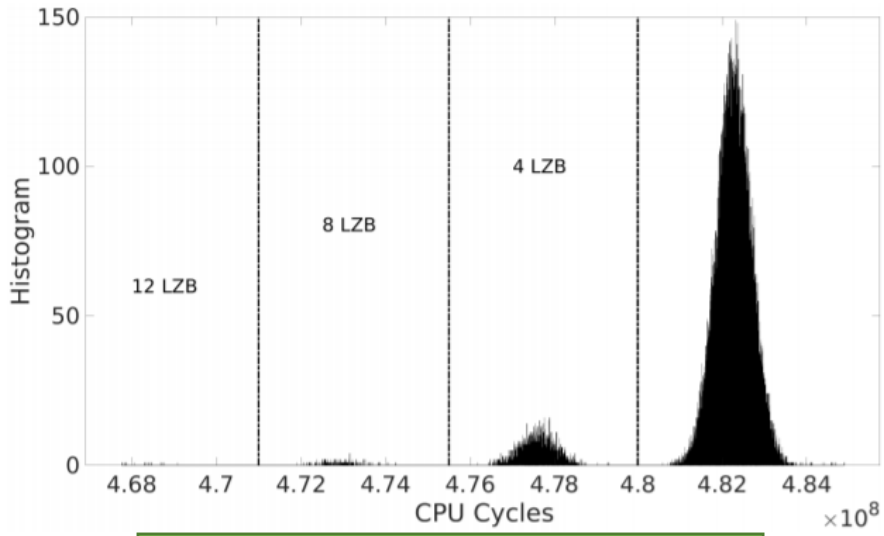


User Adversary





Remote Sample UDP App



System Adversary