Microarchitectural Attacks: Protecting Cloud Accelerators

By

Ahmad "Daniel" Moghimi

PhD Candidate

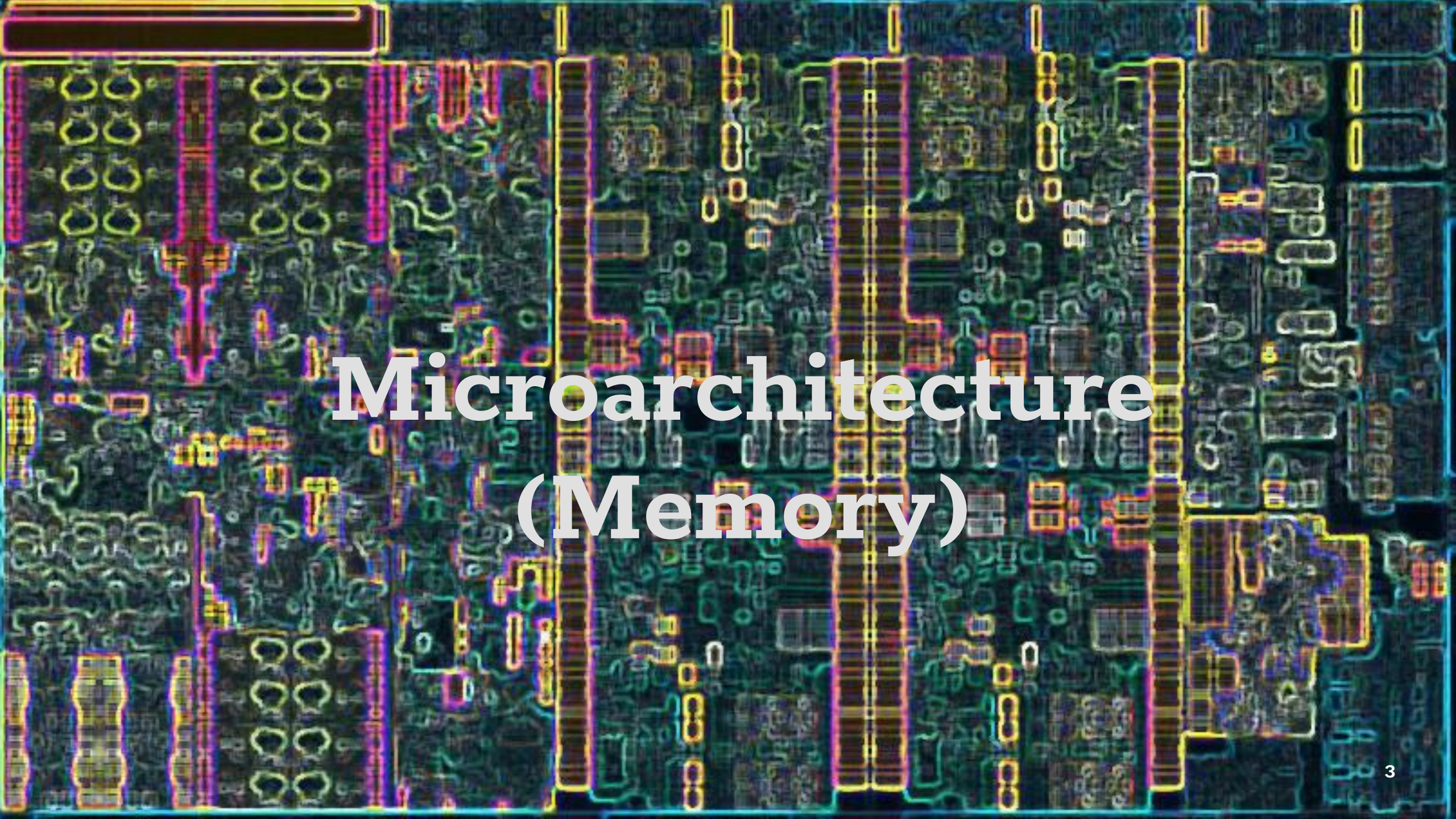
Worcester Polytechnic Institute (WPI)

@danielmgmi

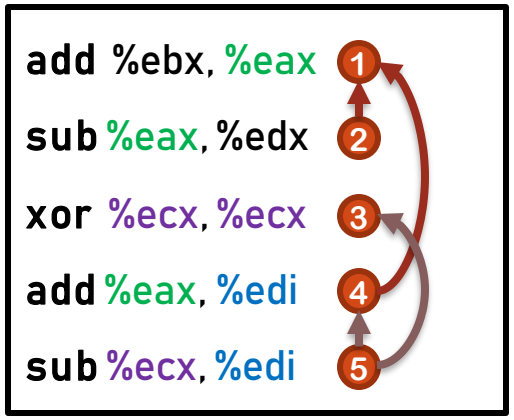
OUTLINE

- Summary of Recent Contributions:
  - Microarchiture→ MemJam
  - Intel SGX → CacheZoom
  - Intel EPID → CacheQuote
  - Speculation → Spoiler
  - Mitigation → MicroWalk

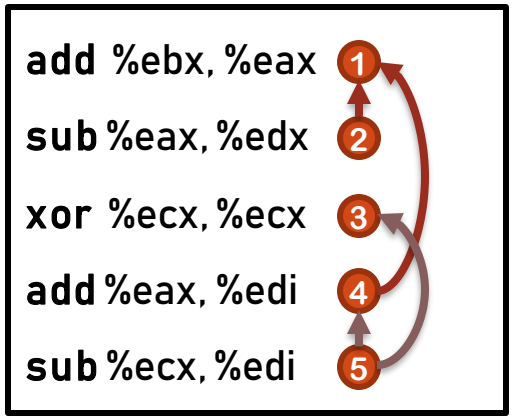
- Shared FPGA-CPU Hardware Security
  - Proposal
  - Lab Equipment/Setup
  - Ongoing Work

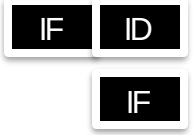

Microarchitecture
(Memory)

µArch Attacks: Data Dependency



µArch Attacks: Pipelined Memory Exec





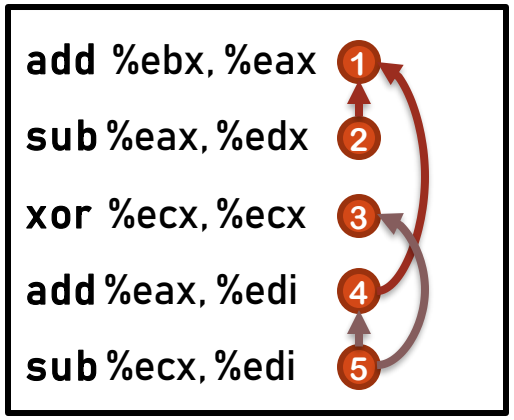
IF Instruction Fetch

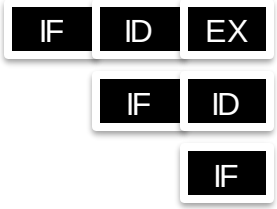
ID Instruction Decode

EX Execute

WB Write Back

µArch Attacks: Pipelined Memory Exec





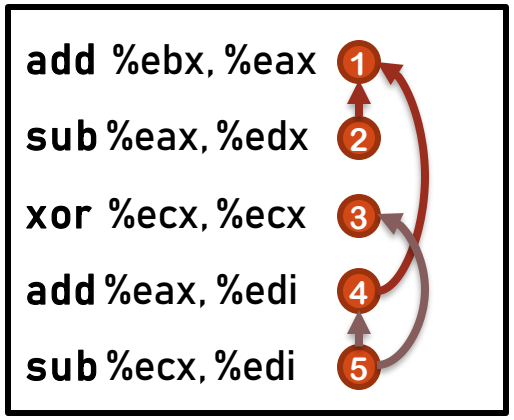
IF Instruction Fetch

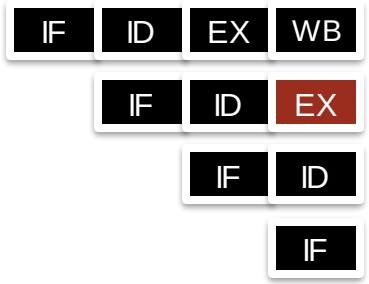
ID Instruction Decode

EX Execute

WB Write Back

µArch Attacks: Pipelined Memory Exec





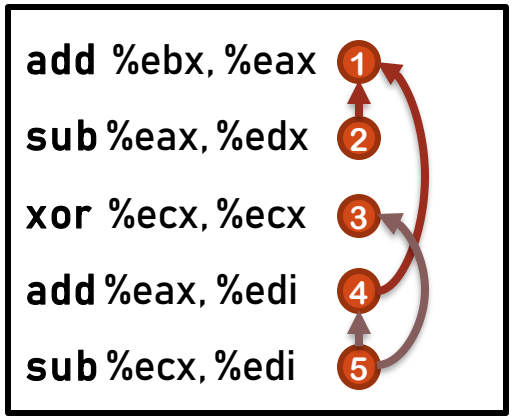
IF Instruction Fetch

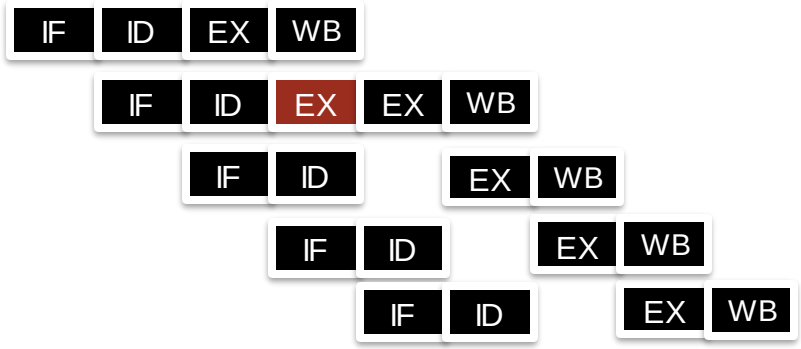
ID Instruction Decode

EX Execute

WB Write Back

µArch Attacks: Pipelined Memory Exec

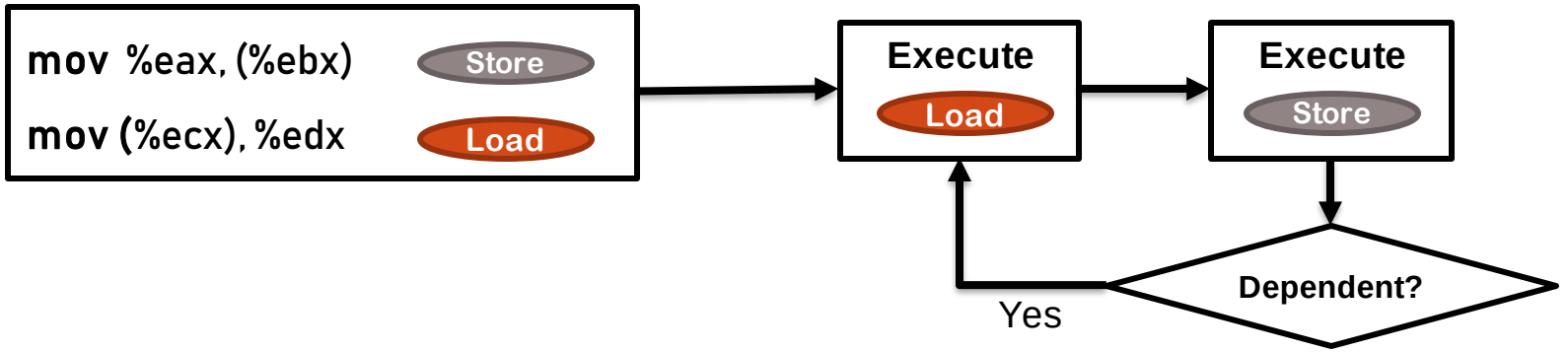




IF Instruction Fetch
ID Instruction Decode
EX Execute
WB Write Back

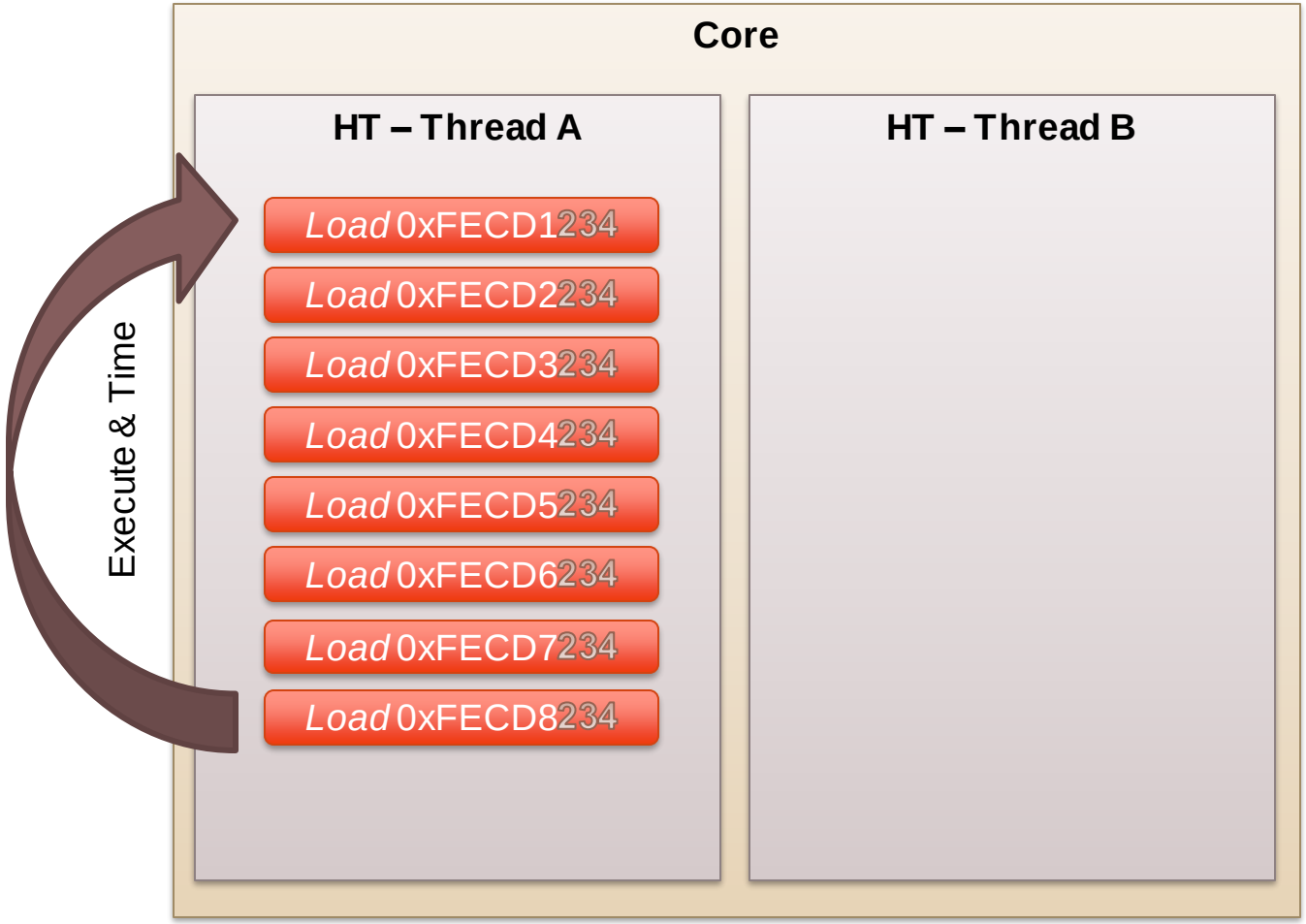
µArch Attacks: 4K Aliasing False Dependency

- Memory loads/stores are executed out of order and speculatively
- The dependency is verified after the execution!

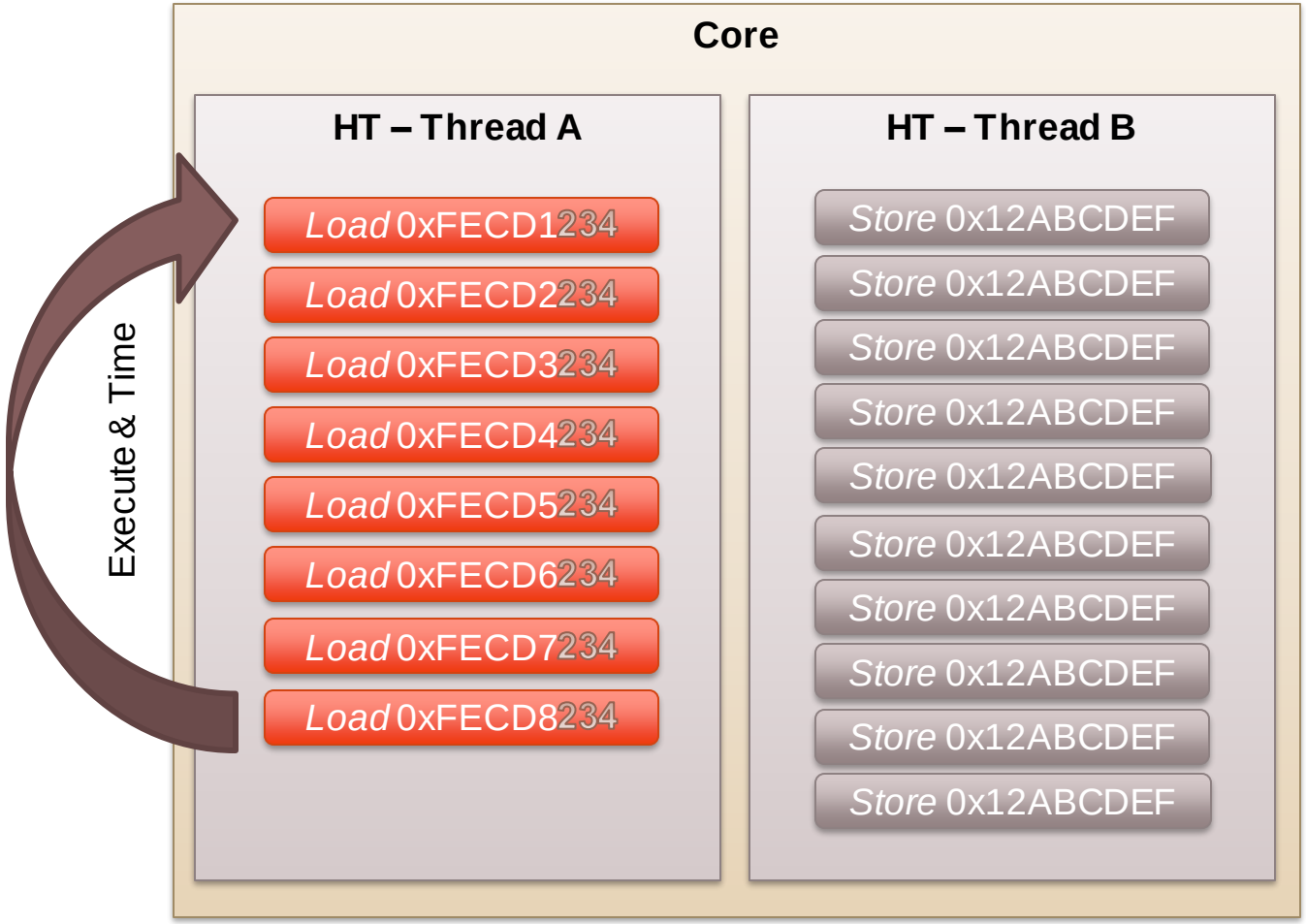


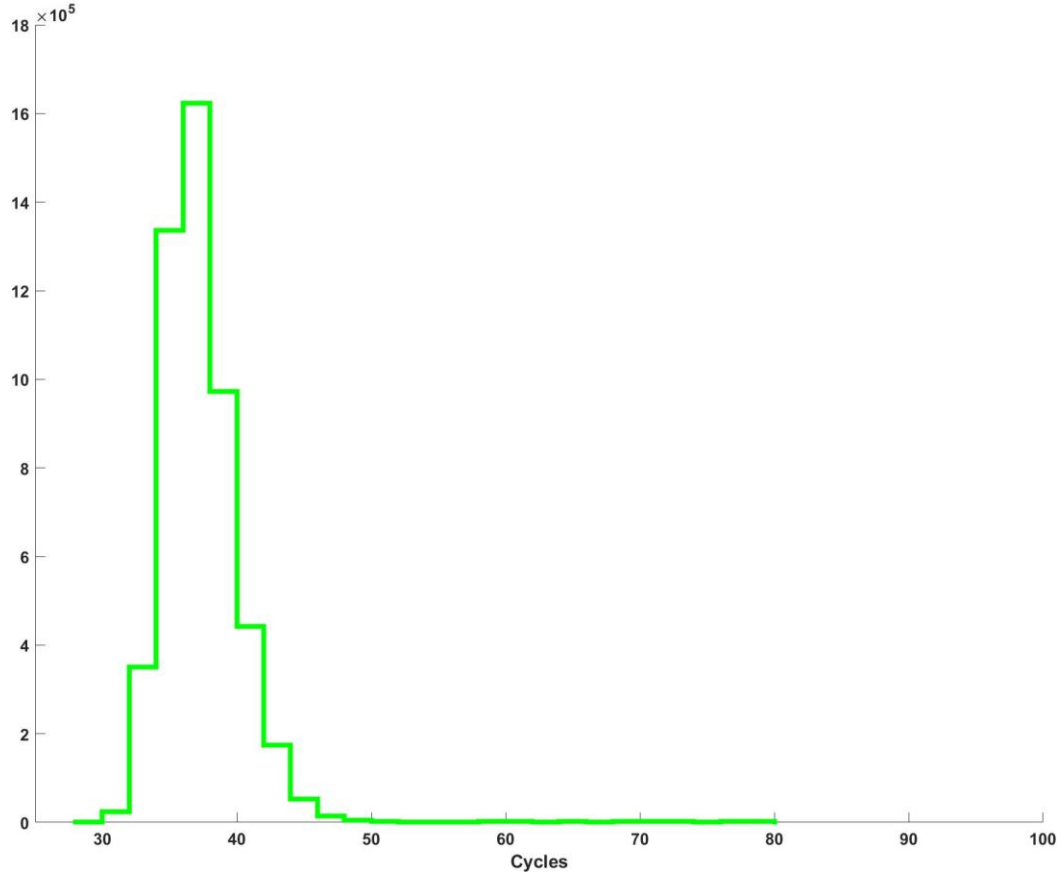
- 4K Aliasing: Addresses that are 4K apart are assumed dependent
- Re-execute the **load** and corresponding instructions due to false dependency
- Virtual-to-physical address translation → Memory disambiguation

µArch Attacks — Hyperthreading 4K Aliasing

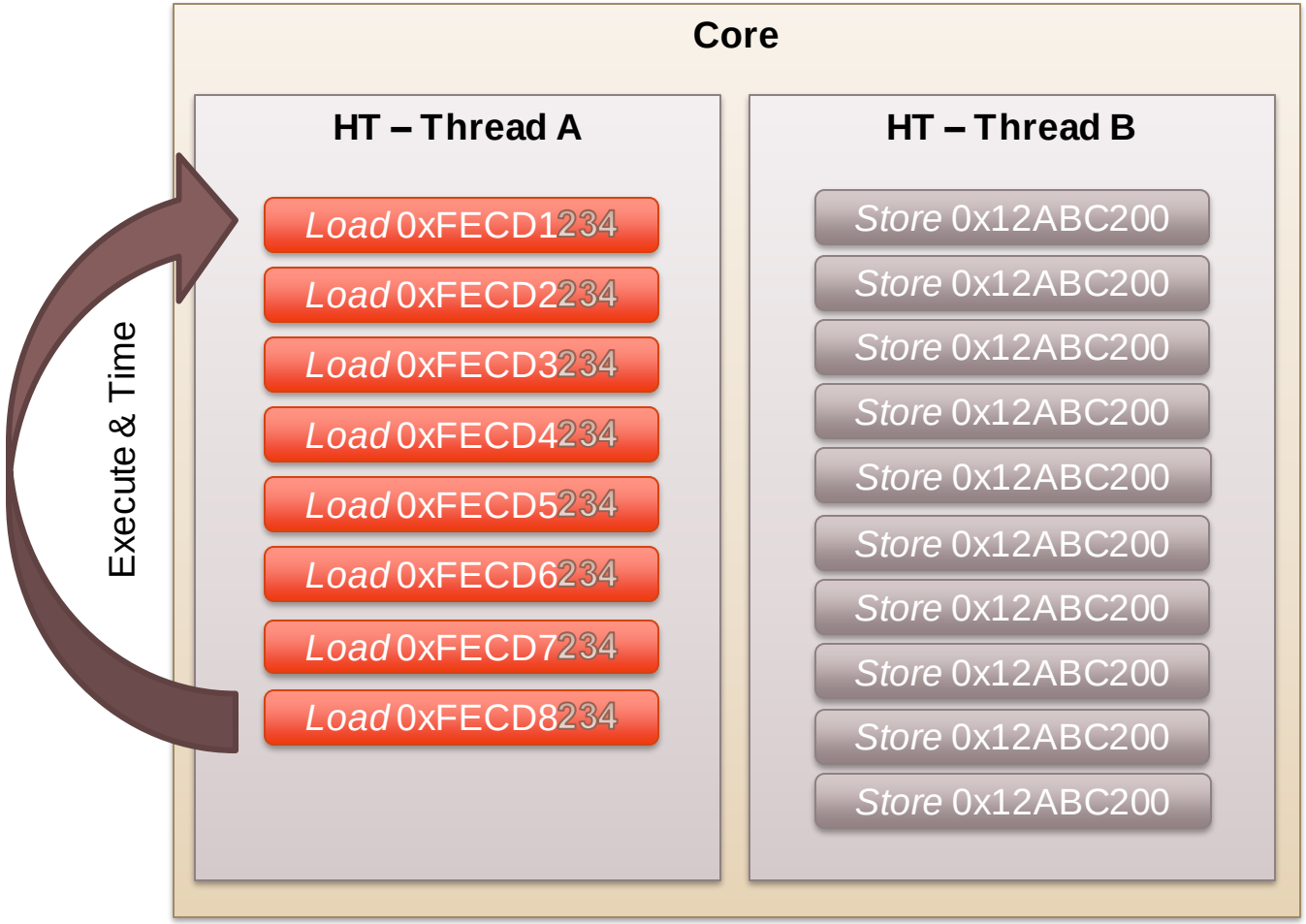


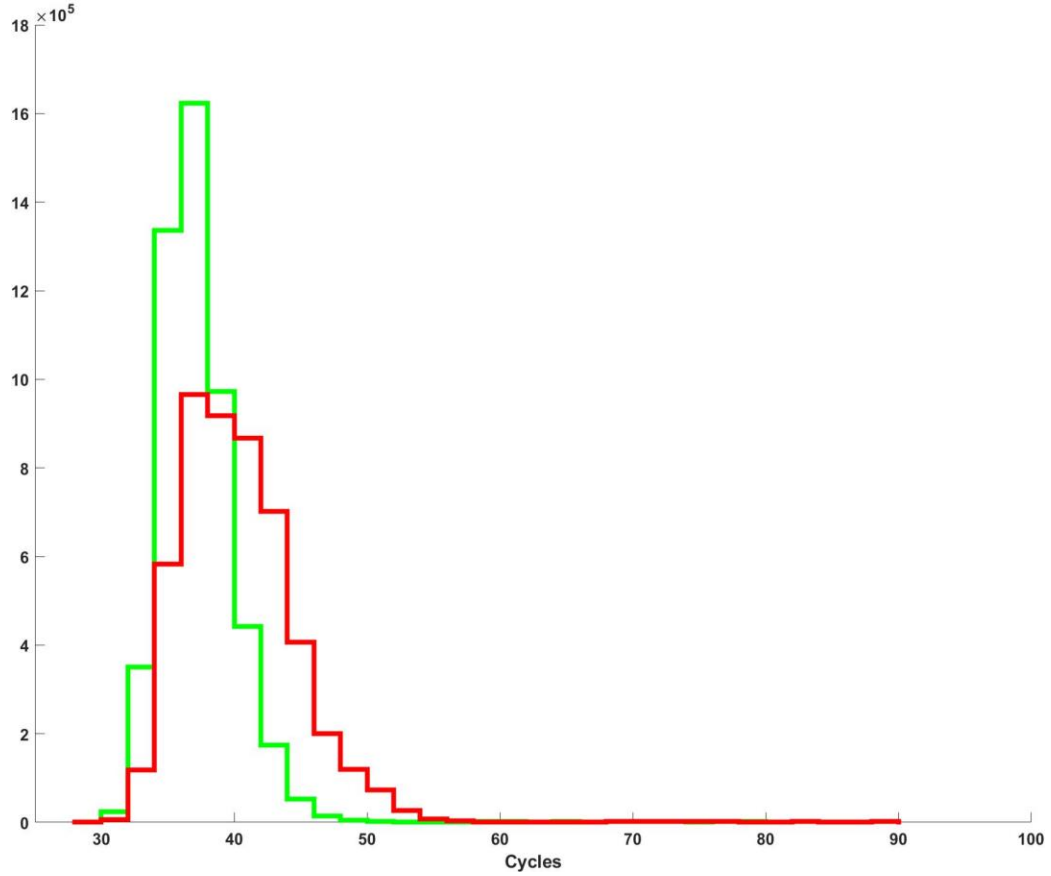
µArch Attacks — Hyperthreading 4K Aliasing



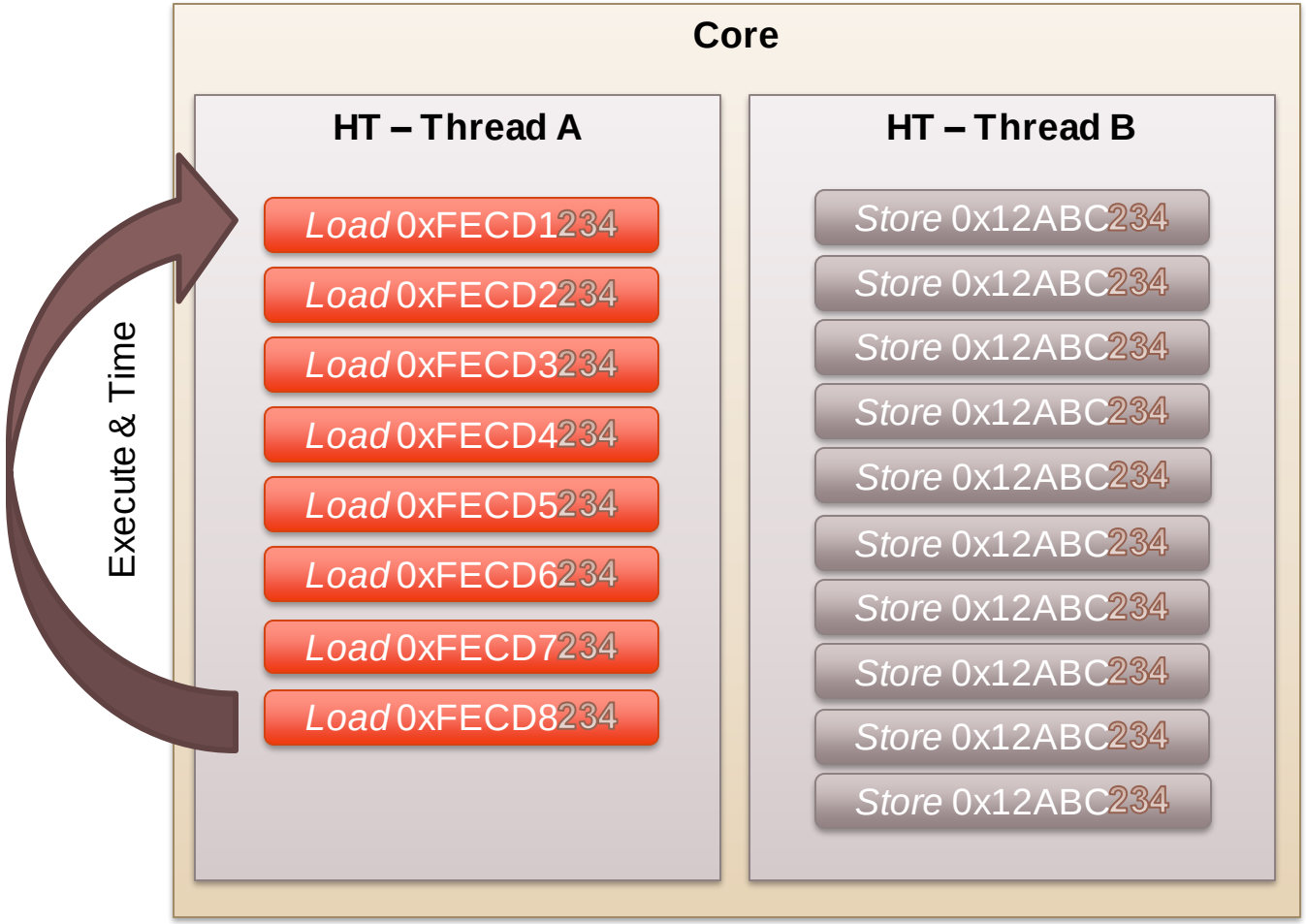


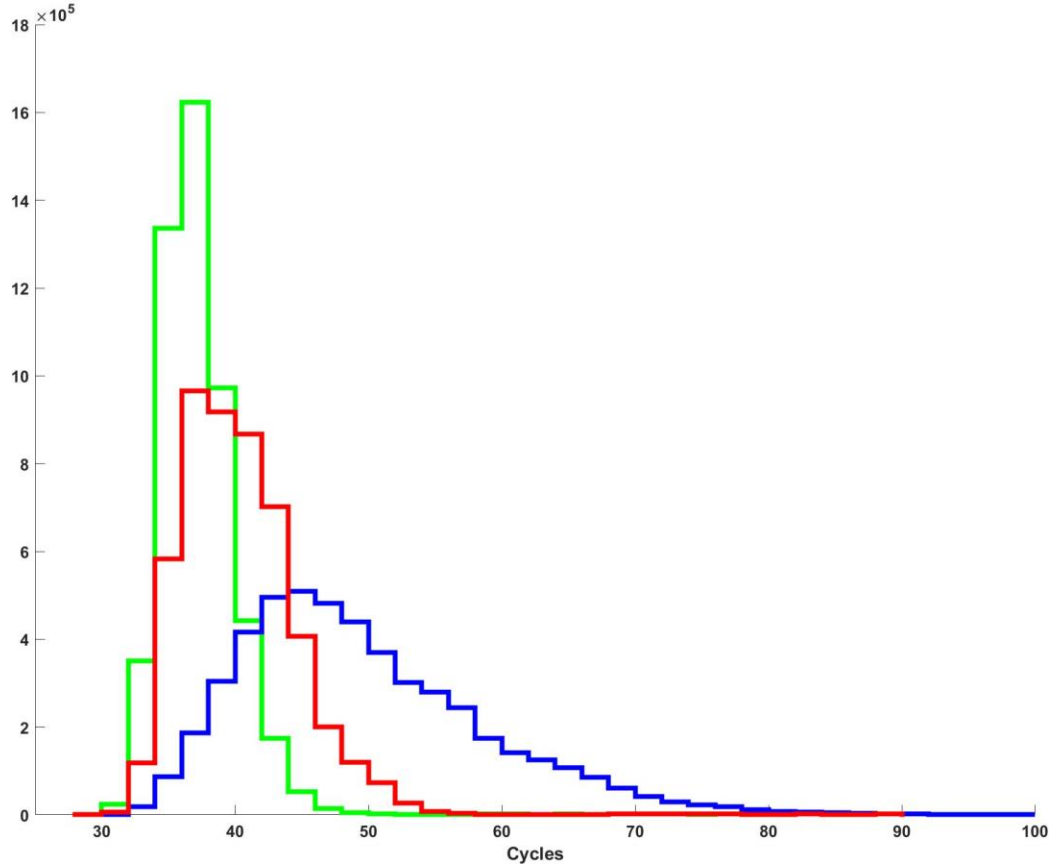
µArch Attacks — Hyperthreading 4K Aliasing



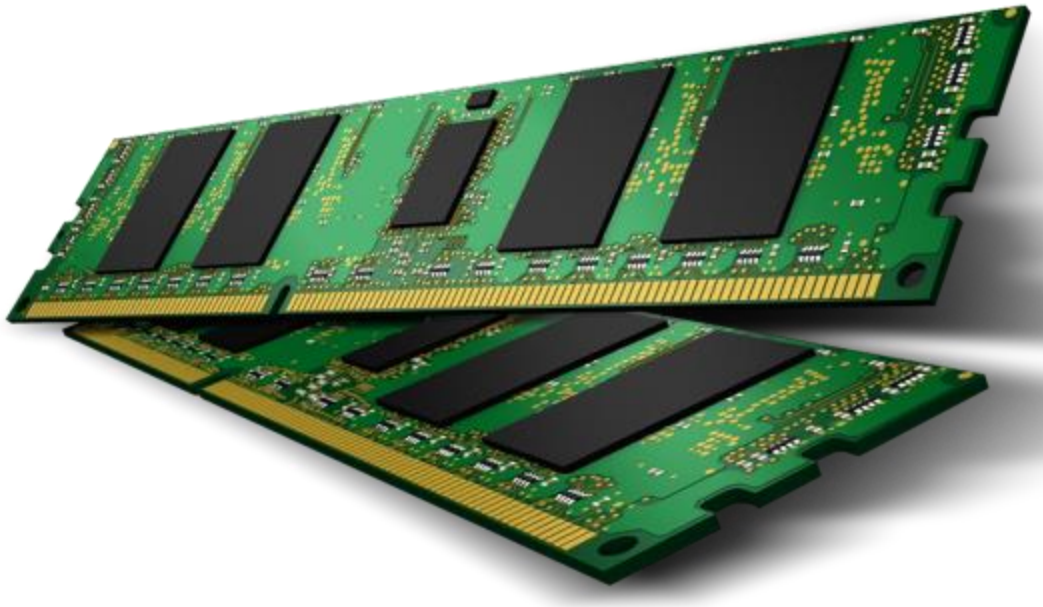


µArch Attacks — Hyperthreading 4K Aliasing

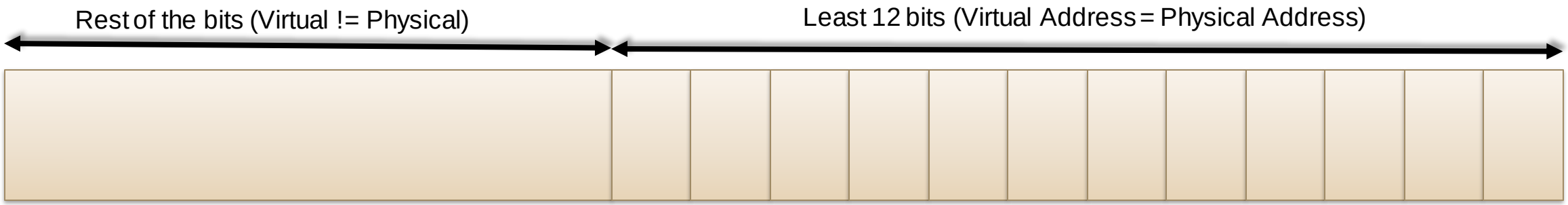




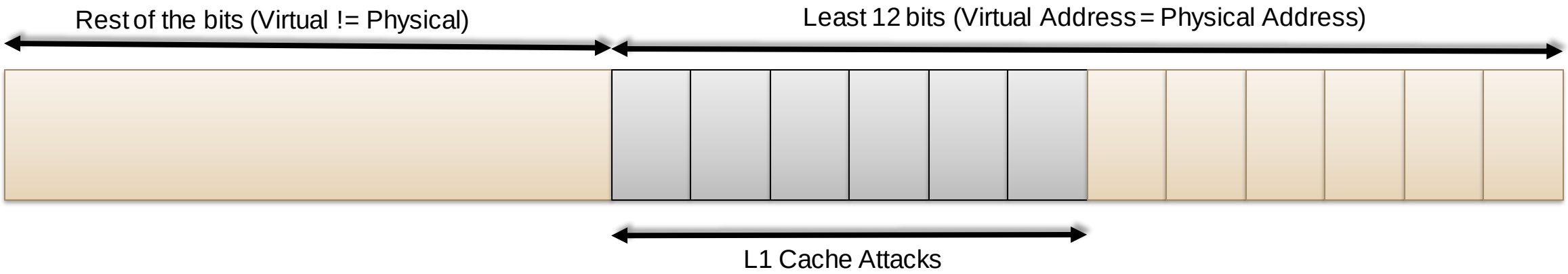
MemJam



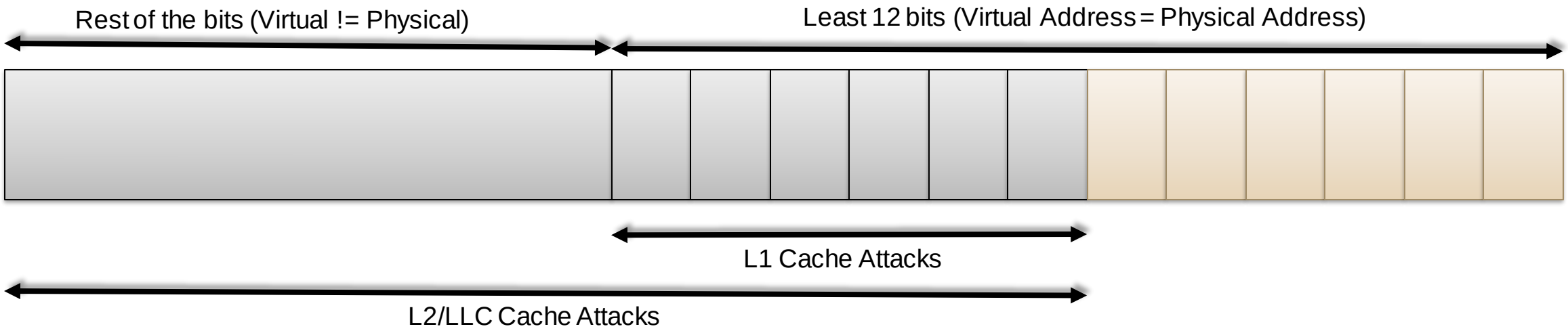
MemJam — Intra Cache Line Resolution



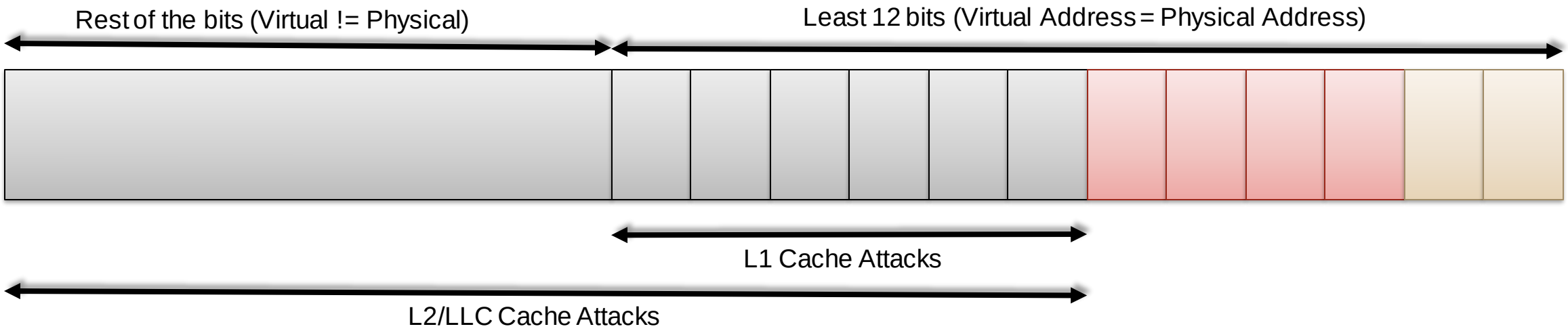
MemJam — Intra Cache Line Resolution



MemJam — Intra Cache Line Resolution

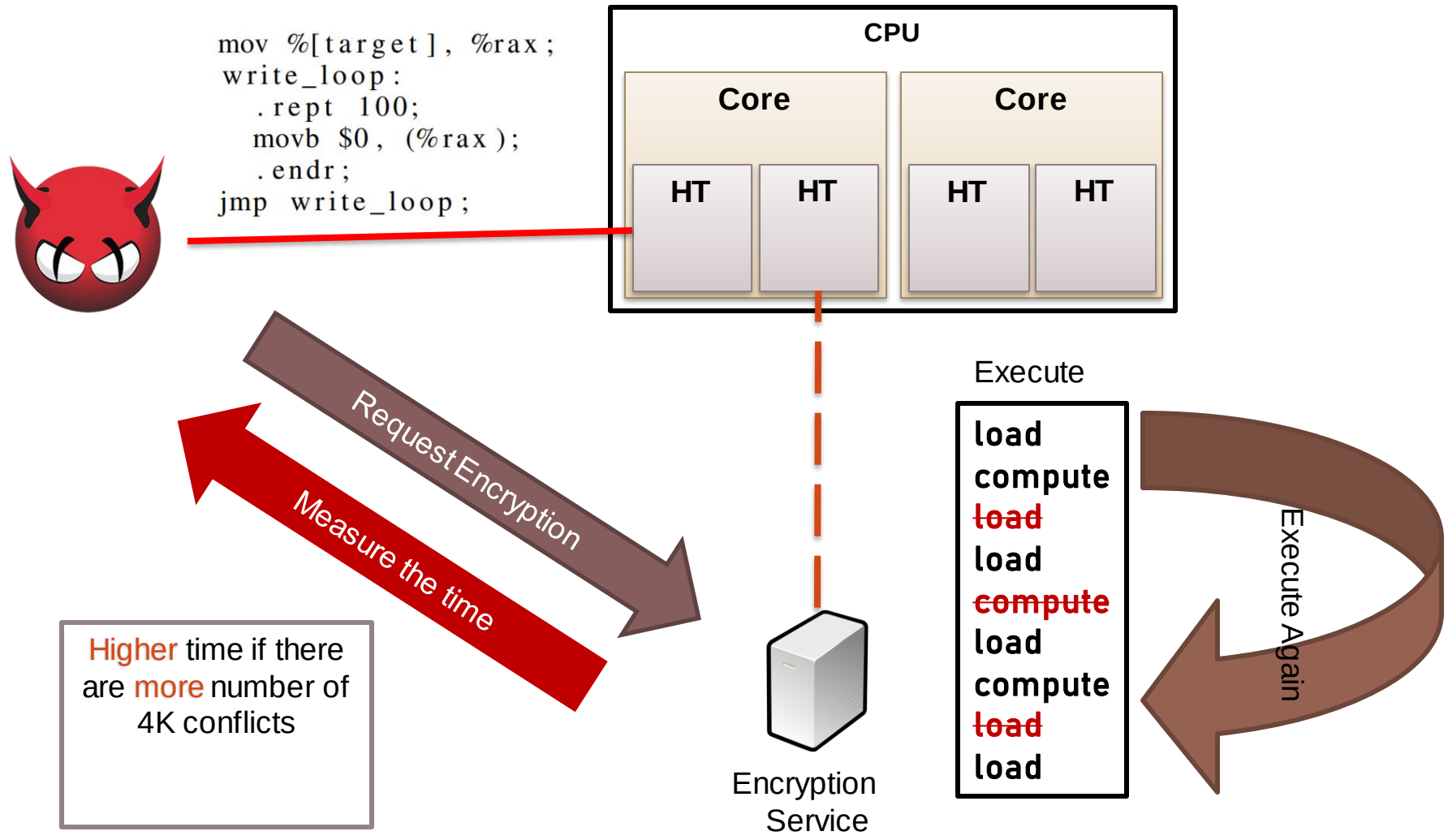


MemJam — Intra Cache Line Resolution



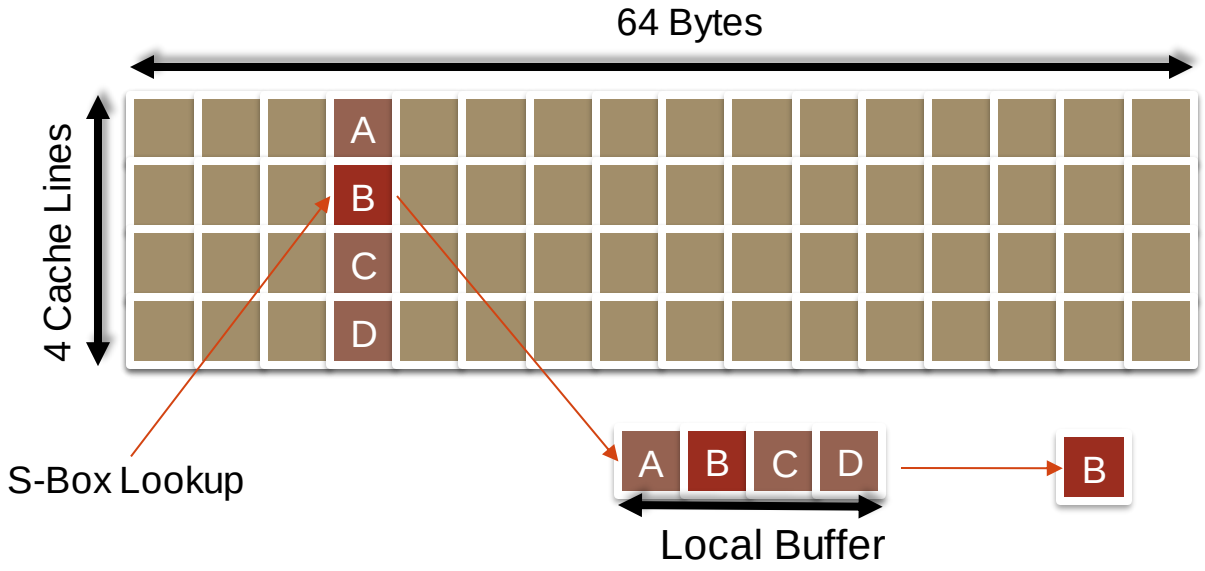
- Conflicted intra-cache line Leakage (4-byte granularity)
- Higher time **correlates** → Memory accesses with the same bit 3 to 12
- 4 bits of intra-cache level leakage

MemJam Attack

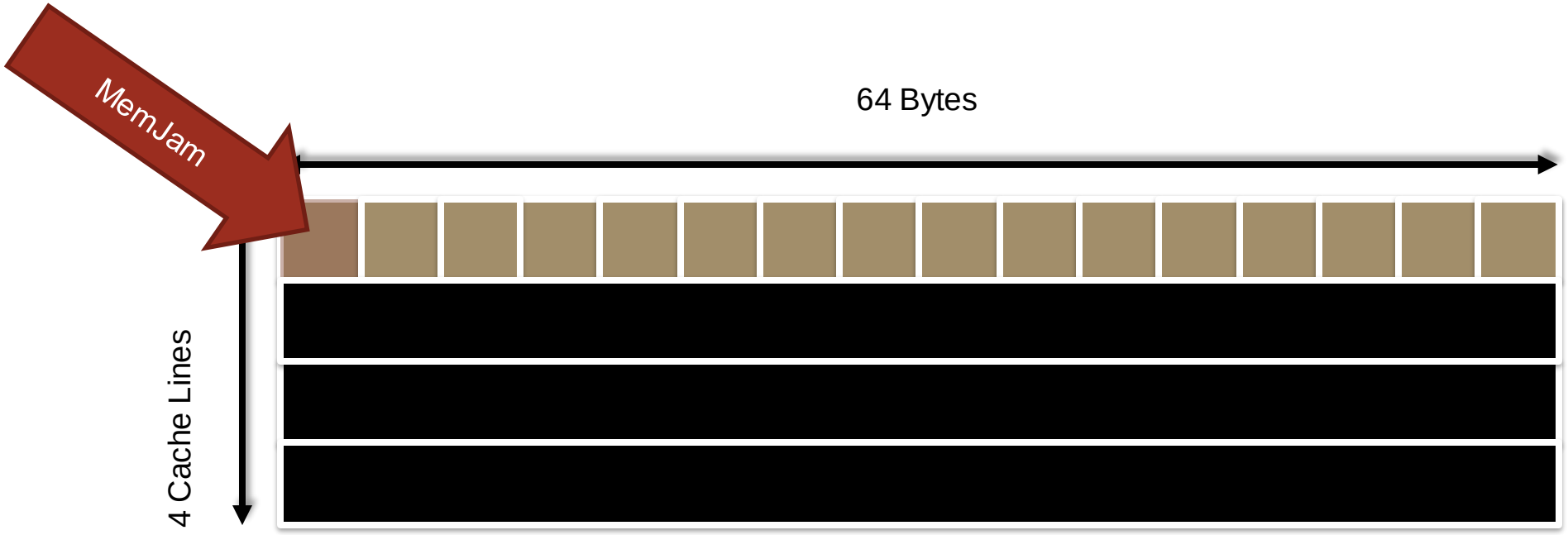


Constant time AES — Safe2Encrypt_RIJ128

- Scatter-gather implementation of AES
  - 256 S-Box — 4 Cache Line
  - Cache independent access pattern
- Implemented and distributed as part of Intel products
  - Intel SGX Linux Software Development Kit (SDK)
  - Intel IPP Cryptography Library



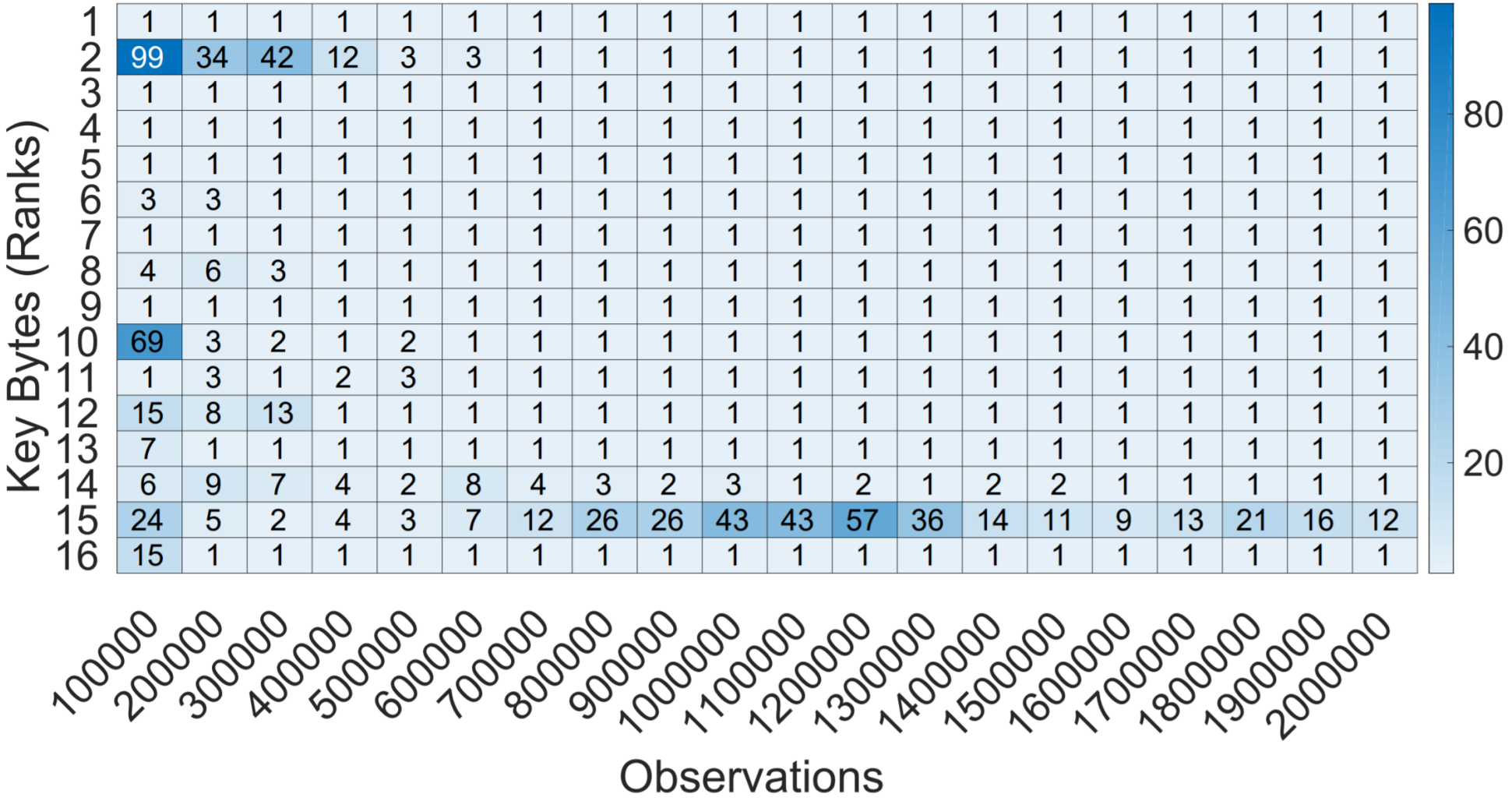
MemJam Attack on Safe2Encrypt_RIJ128



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

Local Buffer

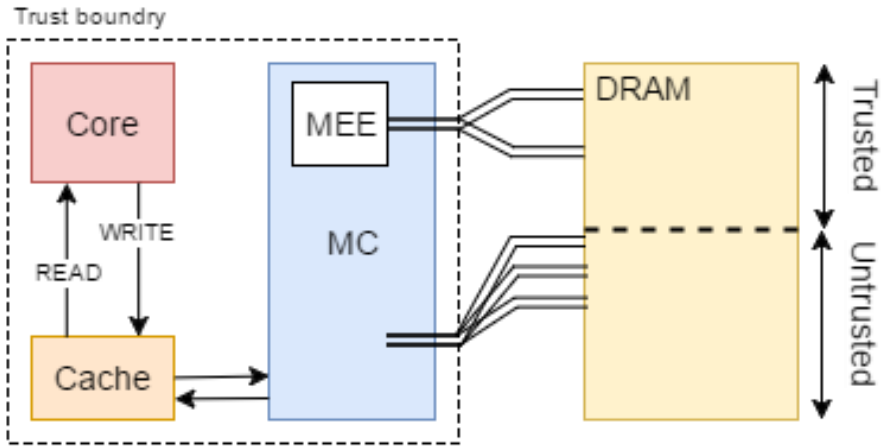
MemJam Attack on Safe2Encrypt_RIJ128

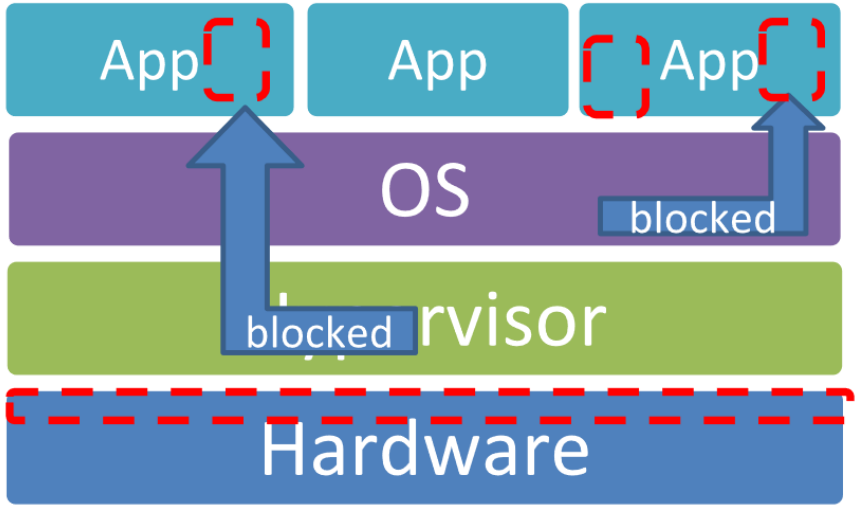


Intel SGX

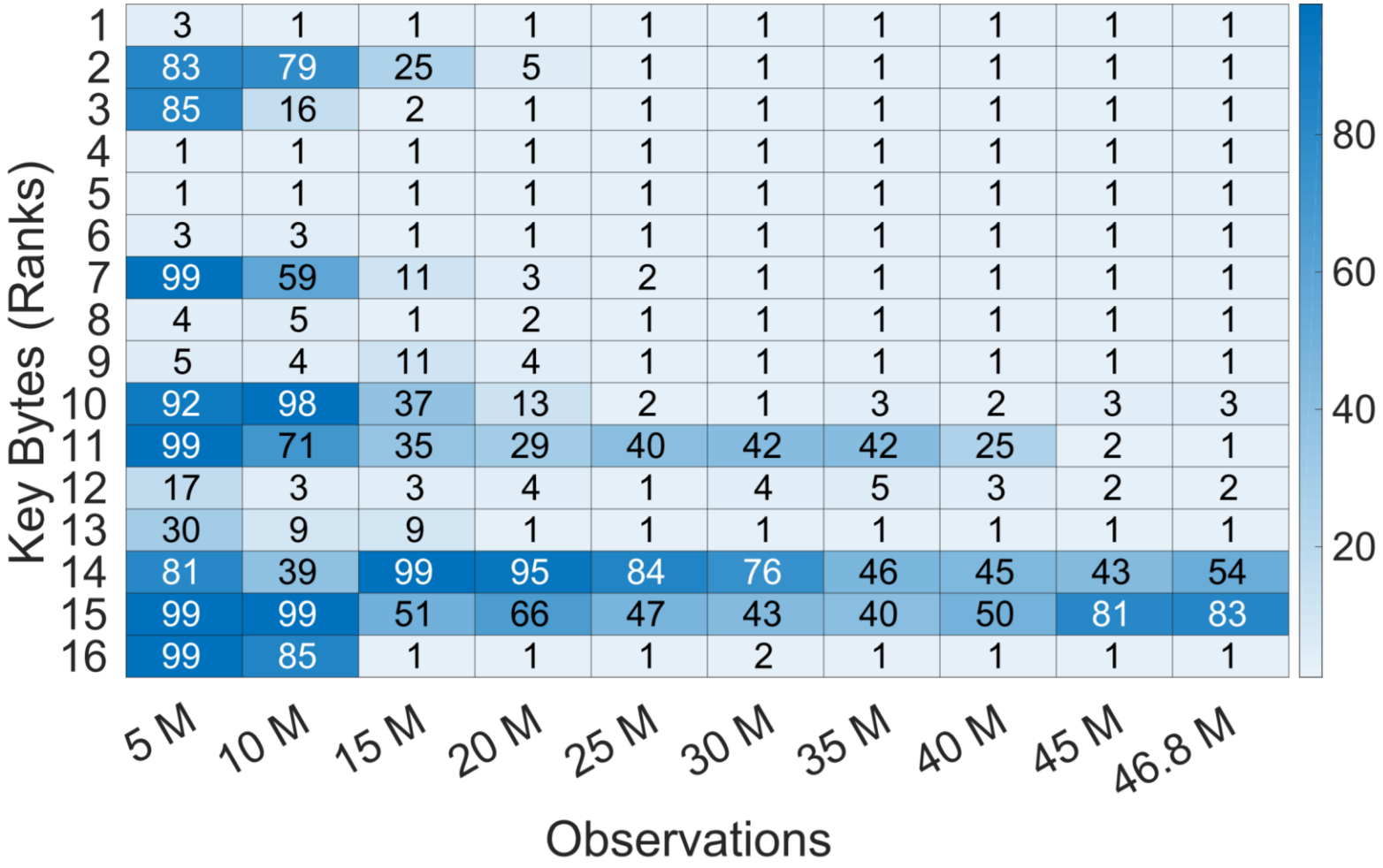
INTEL SOFTWARE GUARD EXTENSION (SGX)

- Trusted Execution Environment (TEE)
- Enclave: Hardware protected user-level software module
  - Loaded by the user program
  - Mapped by the Operating System
  - Authenticated and Encrypted by CPU
- Memory accesses are protected by the hardware



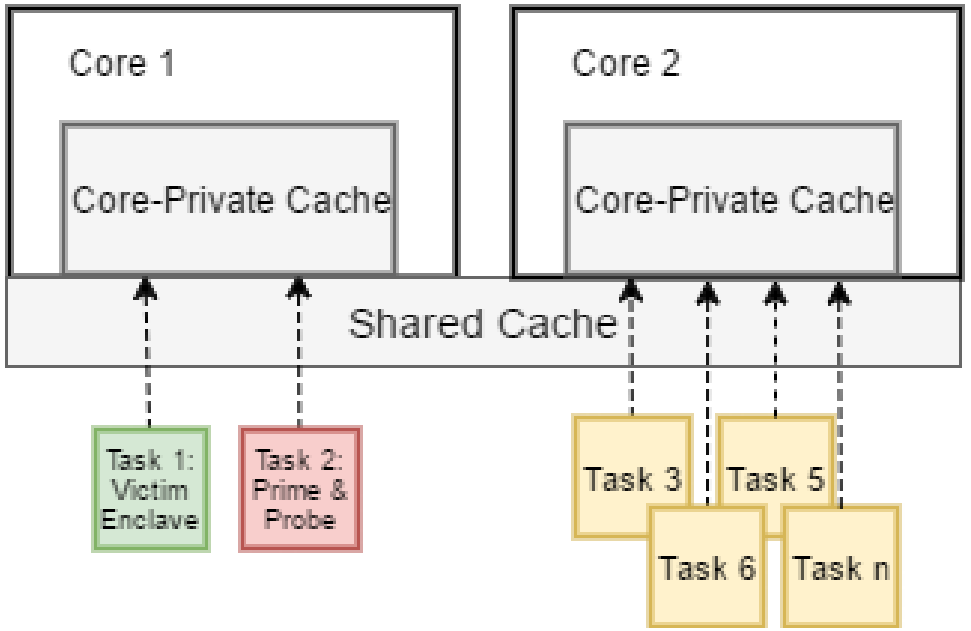


MemJam Attack on SGX

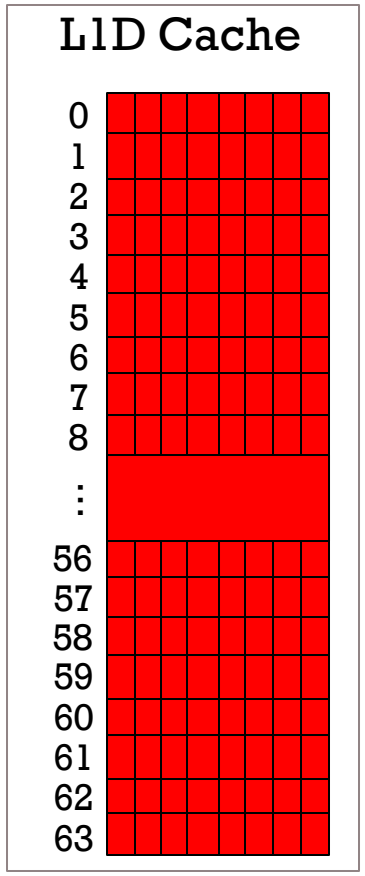


CacheZoom: Controlled Cache Attack ON SGX

1. Isolation of the target & victim cache
2. Stabilize the processor frequency
3. Perform the attack on small exec steps by interrupting the victim
4. Measure and filter the remaining noise



CacheZoom: Interrupted Cache Attack

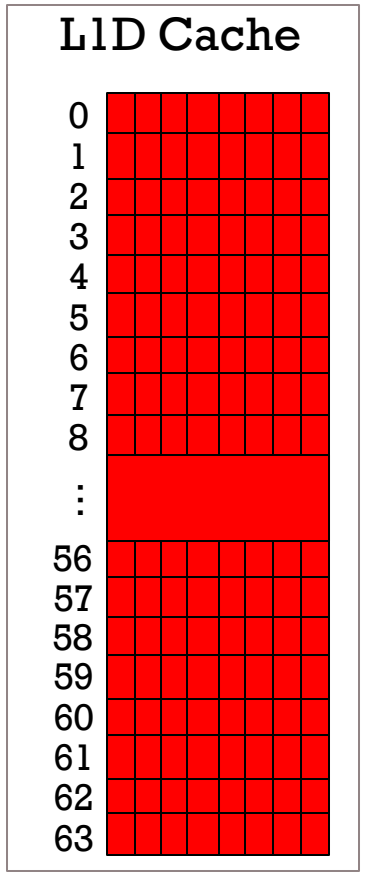


PC

```
for (;;) {
    t0 =
        Te0[(s0 >> 24)       ] ^
        Te1[(s1 >> 16) & 0xff] ^
        Te2[(s2 >>  8) & 0xff] ^
        Te3[(s3      ) & 0xff] ^
        rk[4];
    t1 =
        Te0[(s1 >> 24)       ] ^
        Te1[(s2 >> 16) & 0xff] ^
        Te2[(s3 >>  8) & 0xff] ^
        Te3[(s0      ) & 0xff] ^
        rk[5];
    t2 =
        Te0[(s2 >> 24)       ] ^
        Te1[(s3 >> 16) & 0xff] ^
        Te2[(s0 >>  8) & 0xff] ^
        Te3[(s1      ) & 0xff] ^
        rk[6];
    t3 =
        Te0[(s3 >> 24)       ] ^
        Te1[(s0 >> 16) & 0xff] ^
        Te2[(s1 >>  8) & 0xff] ^
        Te3[(s2      ) & 0xff] ^
        rk[7];
    rk += 8;
    x |= PreFetchTe();
    if (--r == 0) {
        break;
    }
    s0 =
        Te0[(t0 >> 24)       ] ^
        Te1[(t1 >> 16) & 0xff] ^
        Te2[(t2 >>  8) & 0xff] ^
        Te3[(t3      ) & 0xff] ^
        rk[0];
    s1 =
        Te0[(t1 >> 24)       ] ^
        Te1[(t2 >> 16) & 0xff] ^
        Te2[(t3 >>  8) & 0xff] ^
        Te3[(t0      ) & 0xff] ^
        rk[1];
    s2 =
        Te0[(t2 >> 24)       ] ^
        Te1[(t3 >> 16) & 0xff] ^
        Te2[(t0 >>  8) & 0xff] ^
        Te3[(t1      ) & 0xff] ^
        rk[2];
    s3 =
        Te0[(t3 >> 24)       ] ^
        Te1[(t0 >> 16) & 0xff] ^
        Te2[(t1 >>  8) & 0xff] ^
        Te3[(t2      ) & 0xff] ^
        rk[3];
}
```

Step 1: Attacker prime all the L1D sets

CacheZoom: Interrupted Cache Attack



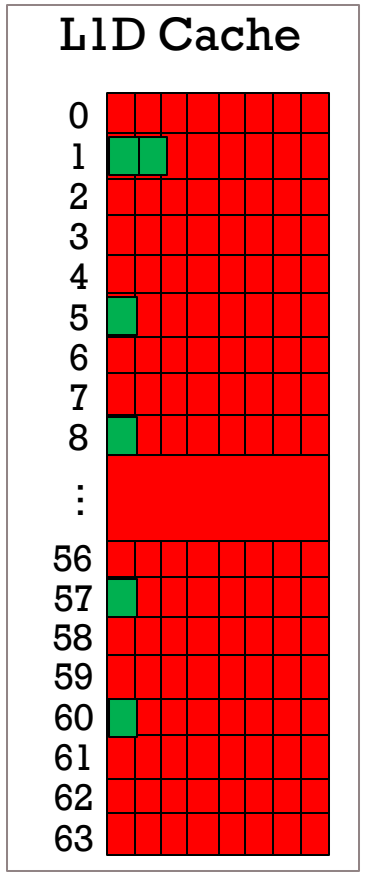
PC

```
for (;;) {
    t0 =
        Te0[(s0 >> 24)       ] ^
        Te1[(s1 >> 16) & 0xff] ^
        Te2[(s2 >>  8) & 0xff] ^
        Te3[(s3      ) & 0xff] ^
        rk[4];
    t1 =
        Te0[(s1 >> 24)       ] ^
        Te1[(s2 >> 16) & 0xff] ^
        Te2[(s3 >>  8) & 0xff] ^
        Te3[(s0      ) & 0xff] ^
        rk[5];
    t2 =
        Te0[(s2 >> 24)       ] ^
        Te1[(s3 >> 16) & 0xff] ^
        Te2[(s0 >>  8) & 0xff] ^
        Te3[(s1      ) & 0xff] ^
        rk[6];
    t3 =
        Te0[(s3 >> 24)       ] ^
        Te1[(s0 >> 16) & 0xff] ^
        Te2[(s1 >>  8) & 0xff] ^
        Te3[(s2      ) & 0xff] ^
        rk[7];
    rk += 8;
    x |= PreFetchTe();
    if (--r == 0) {
        break;
    }
    s0 =
        Te0[(t0 >> 24)       ] ^
        Te1[(t1 >> 16) & 0xff] ^
        Te2[(t2 >>  8) & 0xff] ^
        Te3[(t3      ) & 0xff] ^
        rk[0];
    s1 =
        Te0[(t1 >> 24)       ] ^
        Te1[(t2 >> 16) & 0xff] ^
        Te2[(t3 >>  8) & 0xff] ^
        Te3[(t0      ) & 0xff] ^
        rk[1];
    s2 =
        Te0[(t2 >> 24)       ] ^
        Te1[(t3 >> 16) & 0xff] ^
        Te2[(t0 >>  8) & 0xff] ^
        Te3[(t1      ) & 0xff] ^
        rk[2];
    s3 =
        Te0[(t3 >> 24)       ] ^
        Te1[(t0 >> 16) & 0xff] ^
        Te2[(t1 >>  8) & 0xff] ^
        Te3[(t2      ) & 0xff] ^
        rk[3];
}
```

Step 1: Attacker prime all the L1D sets

Step 2: Victim executes some codes

CacheZoom: Interrupted Cache Attack



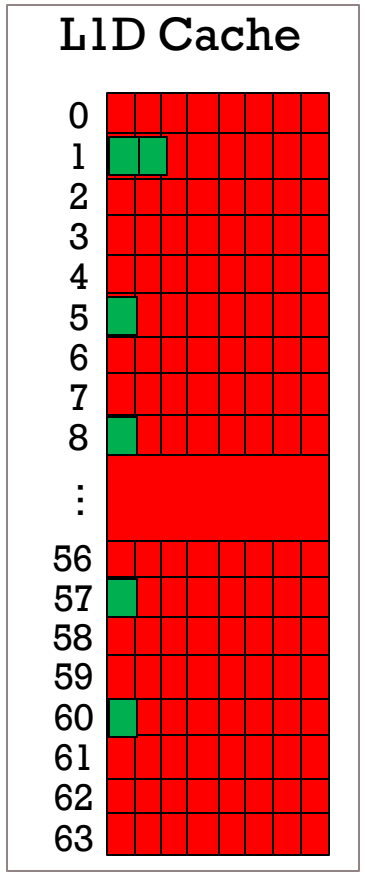
PC

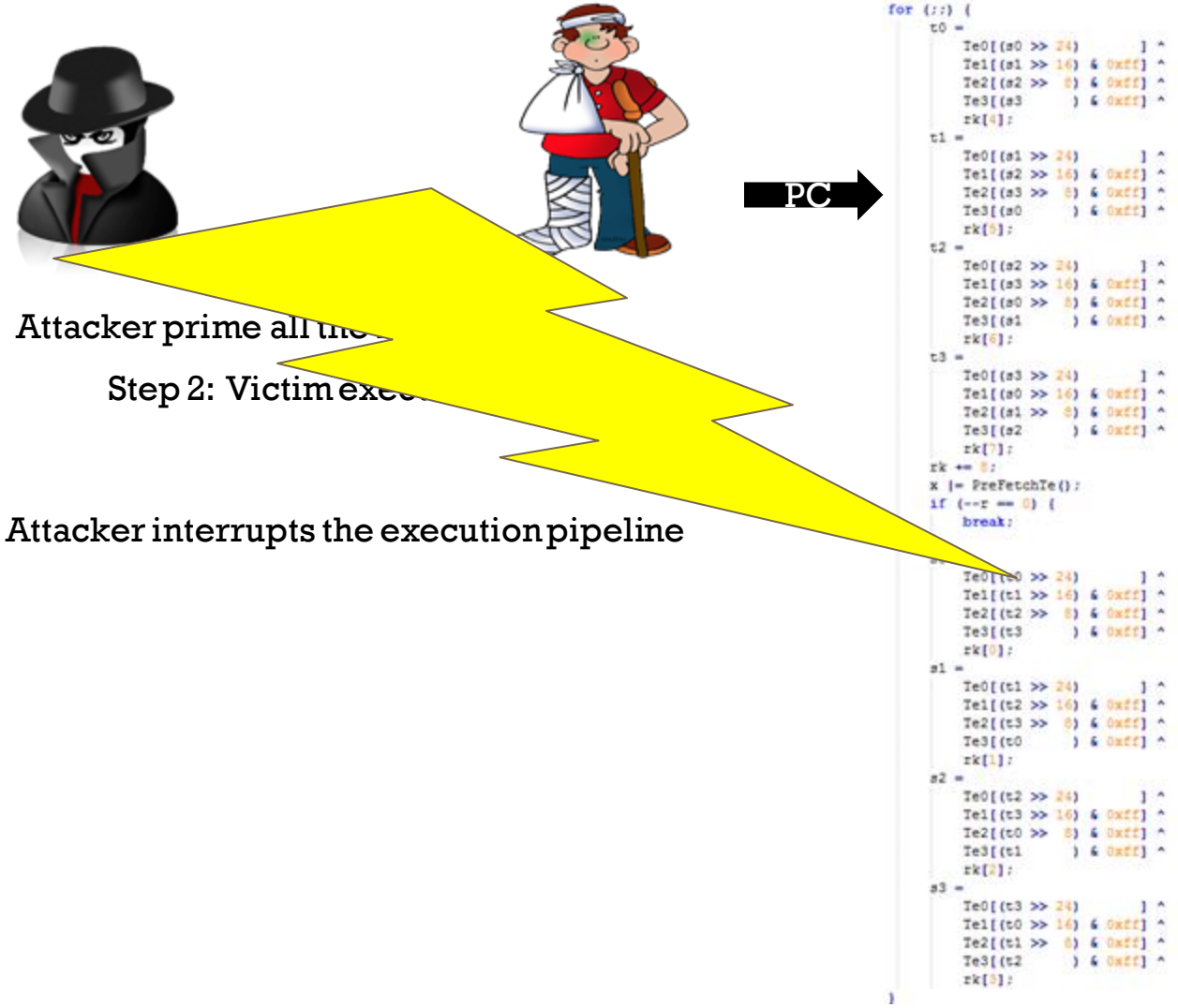
Step 1: Attacker prime all the L1D sets

Step 2: Victim executes some codes

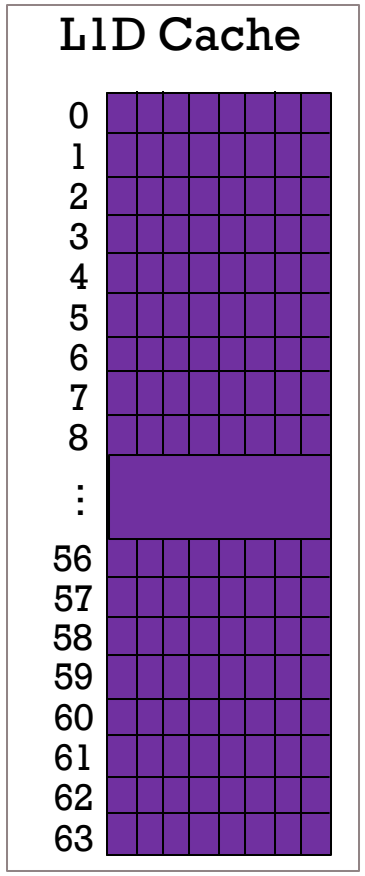
```
for (;;) {
    t0 =
        Te0[(s0 >> 24)       ] ^
        Te1[(s1 >> 16) & 0xff] ^
        Te2[(s2 >>  8) & 0xff] ^
        Te3[(s3      ) & 0xff] ^
        rk[4];
    t1 =
        Te0[(s1 >> 24)       ] ^
        Te1[(s2 >> 16) & 0xff] ^
        Te2[(s3 >>  8) & 0xff] ^
        Te3[(s0      ) & 0xff] ^
        rk[5];
    t2 =
        Te0[(s2 >> 24)       ] ^
        Te1[(s3 >> 16) & 0xff] ^
        Te2[(s0 >>  8) & 0xff] ^
        Te3[(s1      ) & 0xff] ^
        rk[6];
    t3 =
        Te0[(s3 >> 24)       ] ^
        Te1[(s0 >> 16) & 0xff] ^
        Te2[(s1 >>  8) & 0xff] ^
        Te3[(s2      ) & 0xff] ^
        rk[7];
    rk += 8;
    x |= PreFetchTe();
    if (--r == 0) {
        break;
    }
    s0 =
        Te0[(t0 >> 24)       ] ^
        Te1[(t1 >> 16) & 0xff] ^
        Te2[(t2 >>  8) & 0xff] ^
        Te3[(t3      ) & 0xff] ^
        rk[0];
    s1 =
        Te0[(t1 >> 24)       ] ^
        Te1[(t2 >> 16) & 0xff] ^
        Te2[(t3 >>  8) & 0xff] ^
        Te3[(t0      ) & 0xff] ^
        rk[1];
    s2 =
        Te0[(t2 >> 24)       ] ^
        Te1[(t3 >> 16) & 0xff] ^
        Te2[(t0 >>  8) & 0xff] ^
        Te3[(t1      ) & 0xff] ^
        rk[2];
    s3 =
        Te0[(t3 >> 24)       ] ^
        Te1[(t0 >> 16) & 0xff] ^
        Te2[(t1 >>  8) & 0xff] ^
        Te3[(t2      ) & 0xff] ^
        rk[3];
}
```

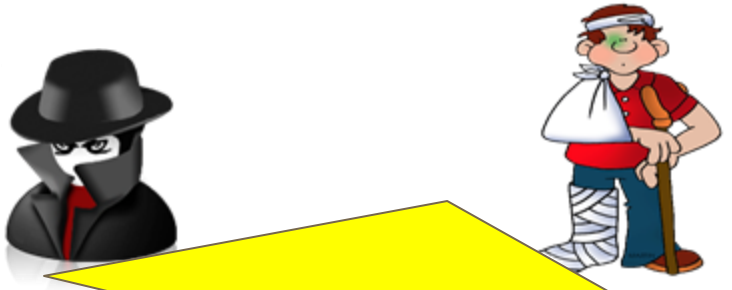
CacheZoom: Interrupted Cache Attack





CacheZoom: Interrupted Cache Attack





PC

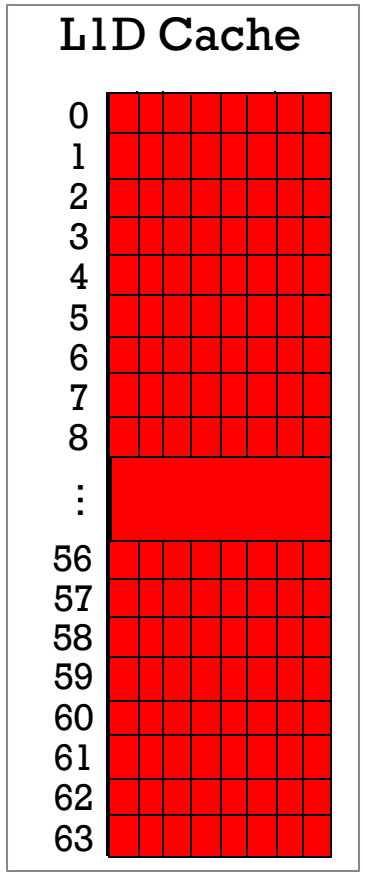
Step 1: Attacker prime all the

Step 2: Victim exec

Step 3: Attacker interrupts the execution pipeline

Step 4: Attacker probes the access times
→ Go to step 1

CacheZoom: Interrupted Cache Attack



Step 1: Attacker prime all the L1D sets

Step 2: Victim executes some codes

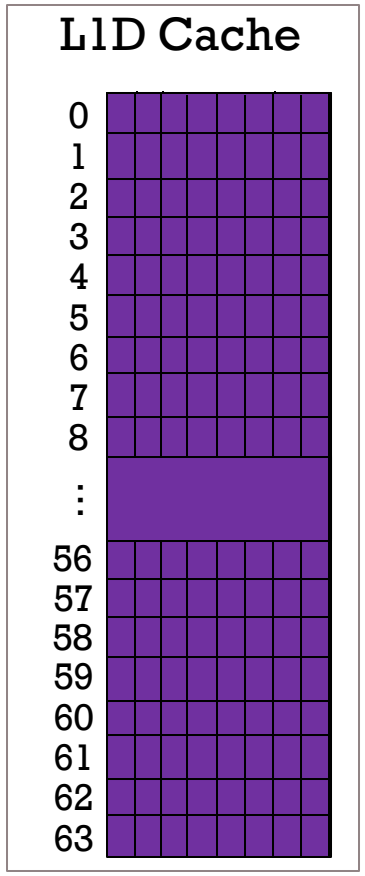
PC

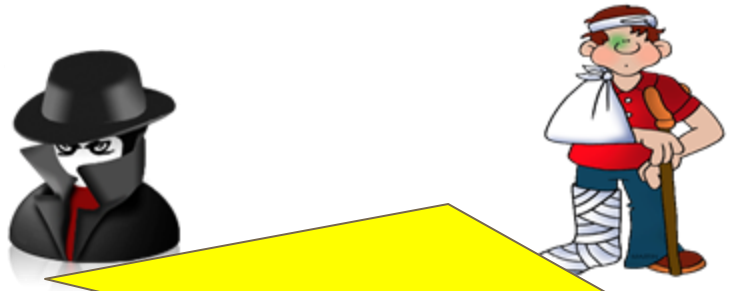
Step 3: Attacker interrupts the execution pipeline

Step 4: Attacker probes the access times
→ Go to step 1

```
for (;;) {
    t0 =
        Te0[(s0 >> 24)       ] ^
        Te1[(s1 >> 16) & 0xff] ^
        Te2[(s2 >>  8) & 0xff] ^
        Te3[(s3      ) & 0xff] ^
        rk[4];
    t1 =
        Te0[(s1 >> 24)       ] ^
        Te1[(s2 >> 16) & 0xff] ^
        Te2[(s3 >>  8) & 0xff] ^
        Te3[(s0      ) & 0xff] ^
        rk[5];
    t2 =
        Te0[(s2 >> 24)       ] ^
        Te1[(s3 >> 16) & 0xff] ^
        Te2[(s0 >>  8) & 0xff] ^
        Te3[(s1      ) & 0xff] ^
        rk[6];
    t3 =
        Te0[(s3 >> 24)       ] ^
        Te1[(s0 >> 16) & 0xff] ^
        Te2[(s1 >>  8) & 0xff] ^
        Te3[(s2      ) & 0xff] ^
        rk[7];
    rk += 8;
    x |= PreFetchTe();
    if (--r == 0) {
        break;
    }
    s0 =
        Te0[(t0 >> 24)       ] ^
        Te1[(t1 >> 16) & 0xff] ^
        Te2[(t2 >>  8) & 0xff] ^
        Te3[(t3      ) & 0xff] ^
        rk[0];
    s1 =
        Te0[(t1 >> 24)       ] ^
        Te1[(t2 >> 16) & 0xff] ^
        Te2[(t3 >>  8) & 0xff] ^
        Te3[(t0      ) & 0xff] ^
        rk[1];
    s2 =
        Te0[(t2 >> 24)       ] ^
        Te1[(t3 >> 16) & 0xff] ^
        Te2[(t0 >>  8) & 0xff] ^
        Te3[(t1      ) & 0xff] ^
        rk[2];
    s3 =
        Te0[(t3 >> 24)       ] ^
        Te1[(t0 >> 16) & 0xff] ^
        Te2[(t1 >>  8) & 0xff] ^
        Te3[(t2      ) & 0xff] ^
        rk[3];
}
```

CacheZoom: Interrupted Cache Attack





Step 1: Attacker prime all the

Step 2: Victim exec

PC

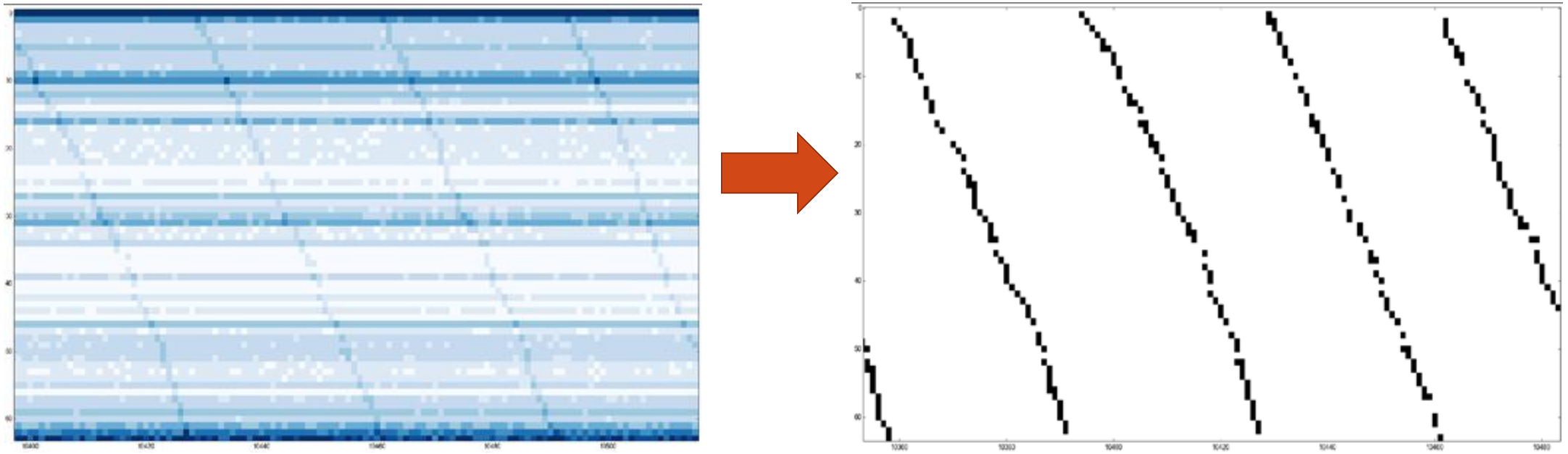
Step 3: Attacker interrupts the execution pipeline

Step 4: Attacker probes the access times
→ Go to step 1

```
for (;;) {
    t0 =
        Te0[(s0 >> 24)       ] ^
        Te1[(s1 >> 16) & 0xff] ^
        Te2[(s2 >>  8) & 0xff] ^
        Te3[(s3      ) & 0xff] ^
        rk[4];
    t1 =
        Te0[(s1 >> 24)       ] ^
        Te1[(s2 >> 16) & 0xff] ^
        Te2[(s3 >>  8) & 0xff] ^
        Te3[(s0      ) & 0xff] ^
        rk[5];
    t2 =
        Te0[(s2 >> 24)       ] ^
        Te1[(s3 >> 16) & 0xff] ^
        Te2[(s0 >>  8) & 0xff] ^
        Te3[(s1      ) & 0xff] ^
        rk[6];
    t3 =
        Te0[(s3 >> 24)       ] ^
        Te1[(s0 >> 16) & 0xff] ^
        Te2[(s1 >>  8) & 0xff] ^
        Te3[(s2      ) & 0xff] ^
        rk[7];
    rk += 8;
    x |= PreFetchTe();
    if (--r == 0) {
        break;
    }

        Te0[(t0 >> 24)       ] ^
        Te1[(t1 >> 16) & 0xff] ^
        Te2[(t2 >>  8) & 0xff] ^
        Te3[(t3      ) & 0xff] ^
        rk[0];
    s1 =
        Te0[(t1 >> 24)       ] ^
        Te1[(t2 >> 16) & 0xff] ^
        Te2[(t3 >>  8) & 0xff] ^
        Te3[(t0      ) & 0xff] ^
        rk[1];
    s2 =
        Te0[(t2 >> 24)       ] ^
        Te1[(t3 >> 16) & 0xff] ^
        Te2[(t0 >>  8) & 0xff] ^
        Te3[(t1      ) & 0xff] ^
        rk[2];
    s3 =
        Te0[(t3 >> 24)       ] ^
        Te1[(t0 >> 16) & 0xff] ^
        Te2[(t1 >>  8) & 0xff] ^
        Te3[(t2      ) & 0xff] ^
        rk[3];
}
```

CacheZoom: Interrupted Cache Attack



CacheQuote

CacheQuote Attack

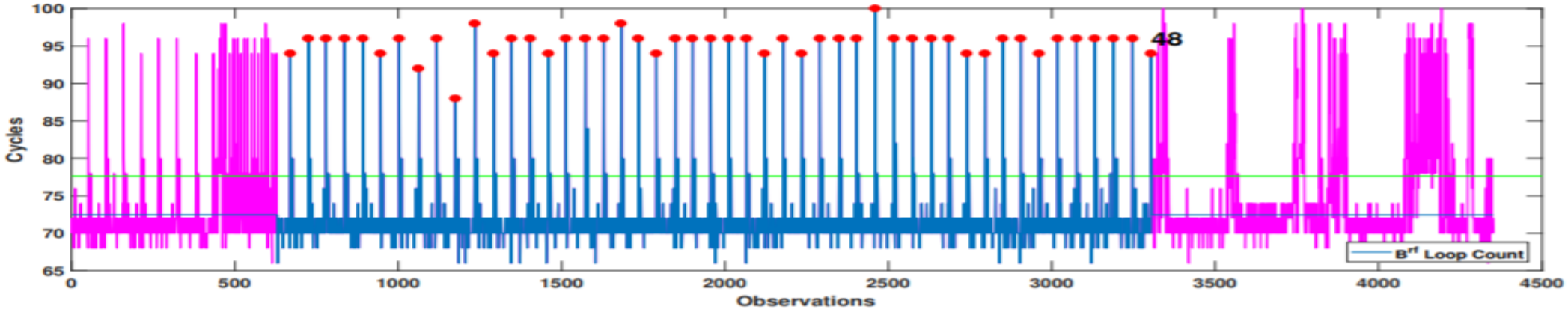
- Quoting Enclave:
  - EPID Signature scheme built-in enclave by Intel
  - Attest the integrity of user-provided enclave

- EPID Implementation (is)was not constant-time

CacheQuote Attack

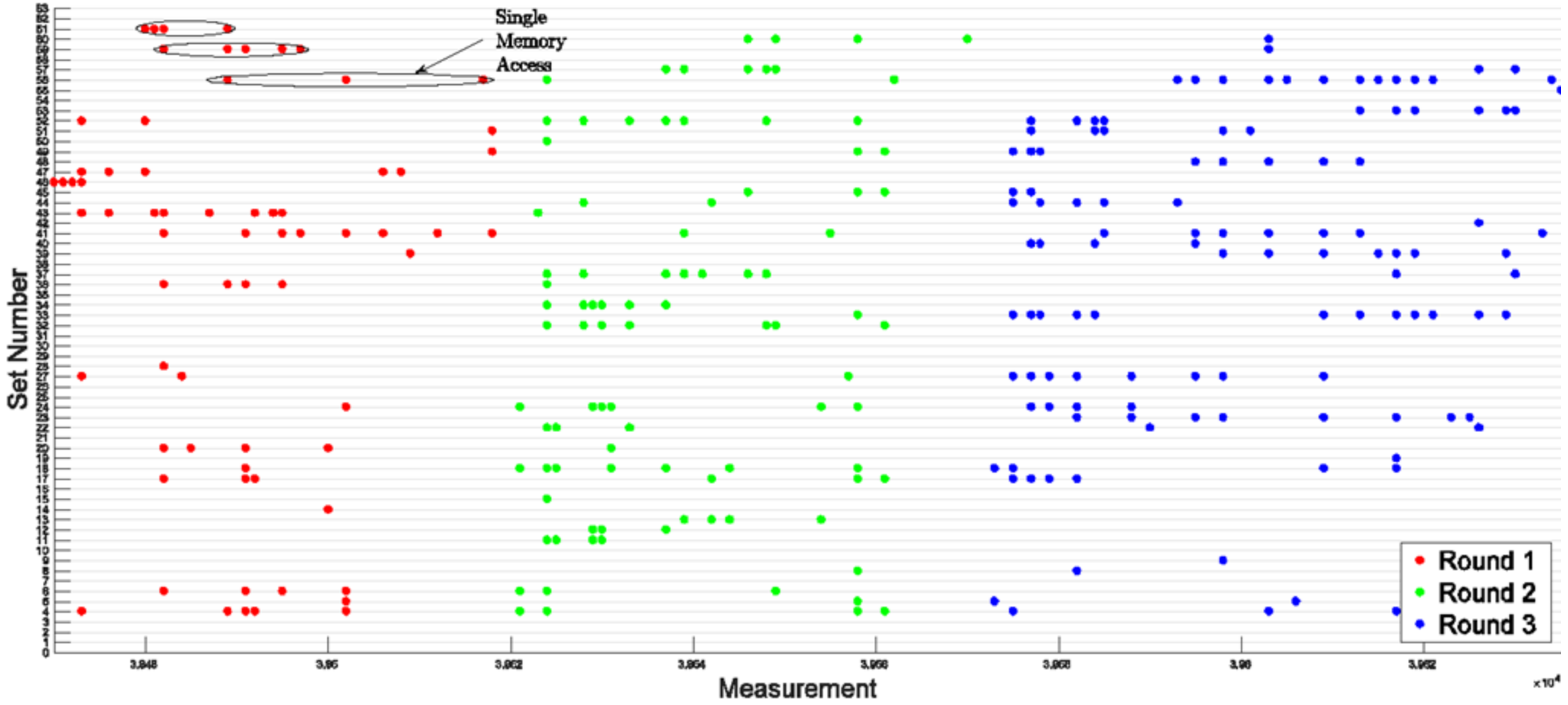
- Loop iteration leaks Leading Zero Bits
- CacheZoom to accurately measure
- Feed the short vectors to a lattice and

procedure MULPOINT(point P, window size the Booth recoding)
  $P_0 \leftarrow \mathcal{O}$
  for $i \leftarrow 1$ **to** 2^{w-1} **do**
    $P_i \leftarrow P \cdot P_{i-1}$
  end for
  $i \leftarrow \max\{j : s_j \neq 0\}$
  $r \leftarrow P_{|s_i|}$
  $i \leftarrow i - 1$
  while $i \geq 0$ **do**
    $r \leftarrow r^{2^w}$
    $t \leftarrow P_{|s_i|}$
    $r \leftarrow r \cdot \text{ctSelect}(t, t^{-1}, \text{isNegative}(s_i))$
    $i \leftarrow i - 1$
  end while
  return r
end procedure

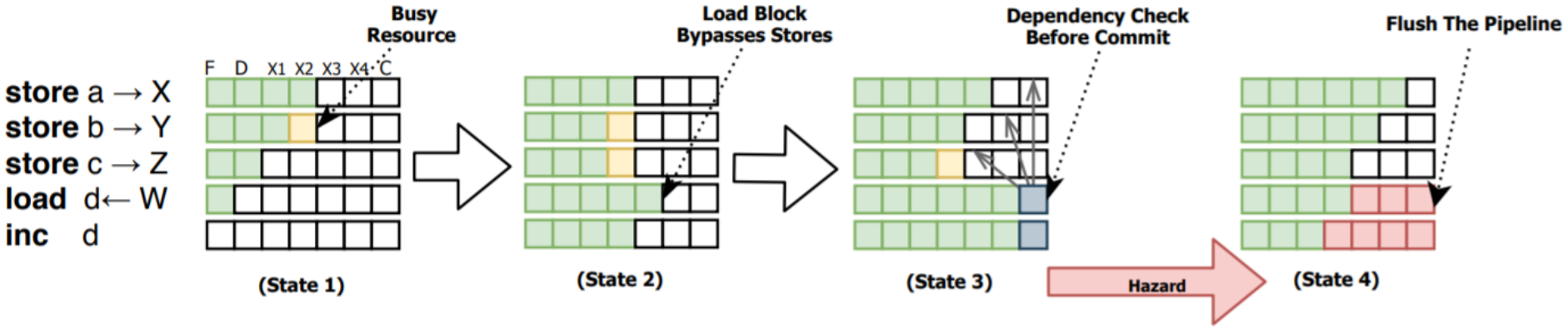


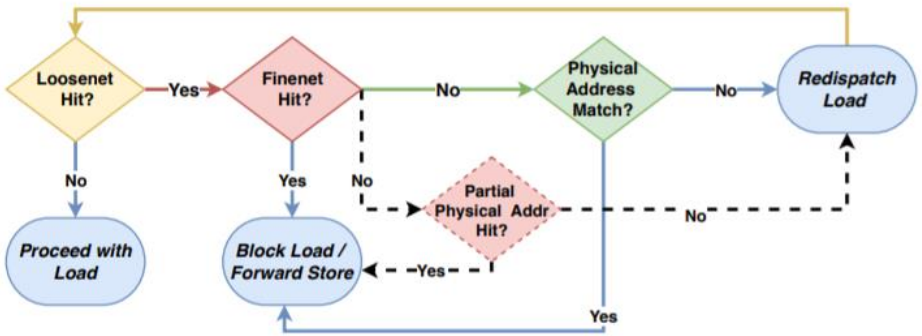
Memory Speculation

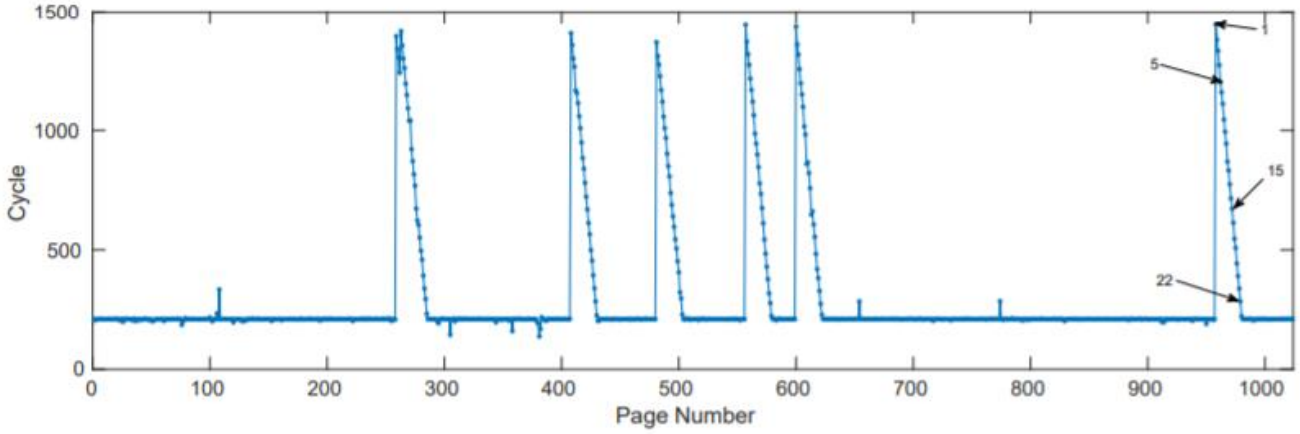
Speculative Memory Accesses



Spoiler on **Spoiler** Attack

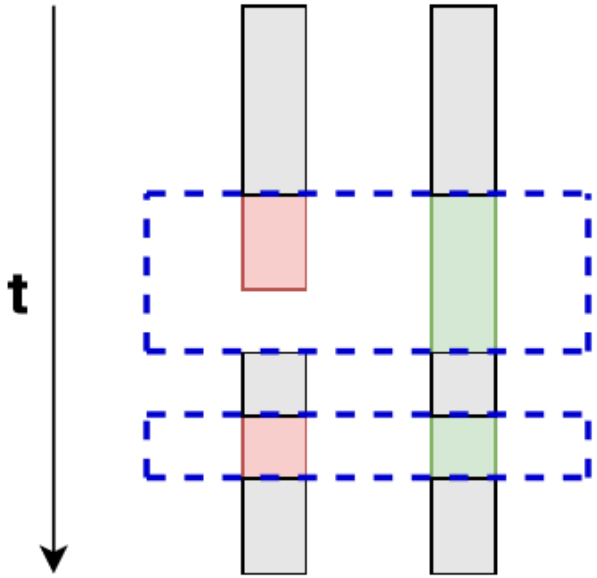


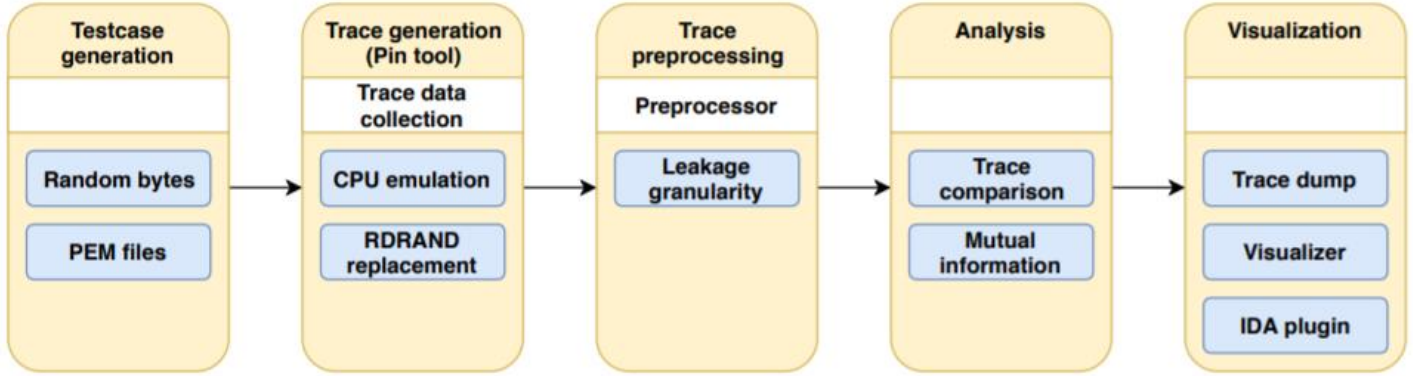




MicroWalk: Finding µArch Sources in Binaries

- Detecting Leakages based on Binary Instrumentation and Mutual Information Analysis





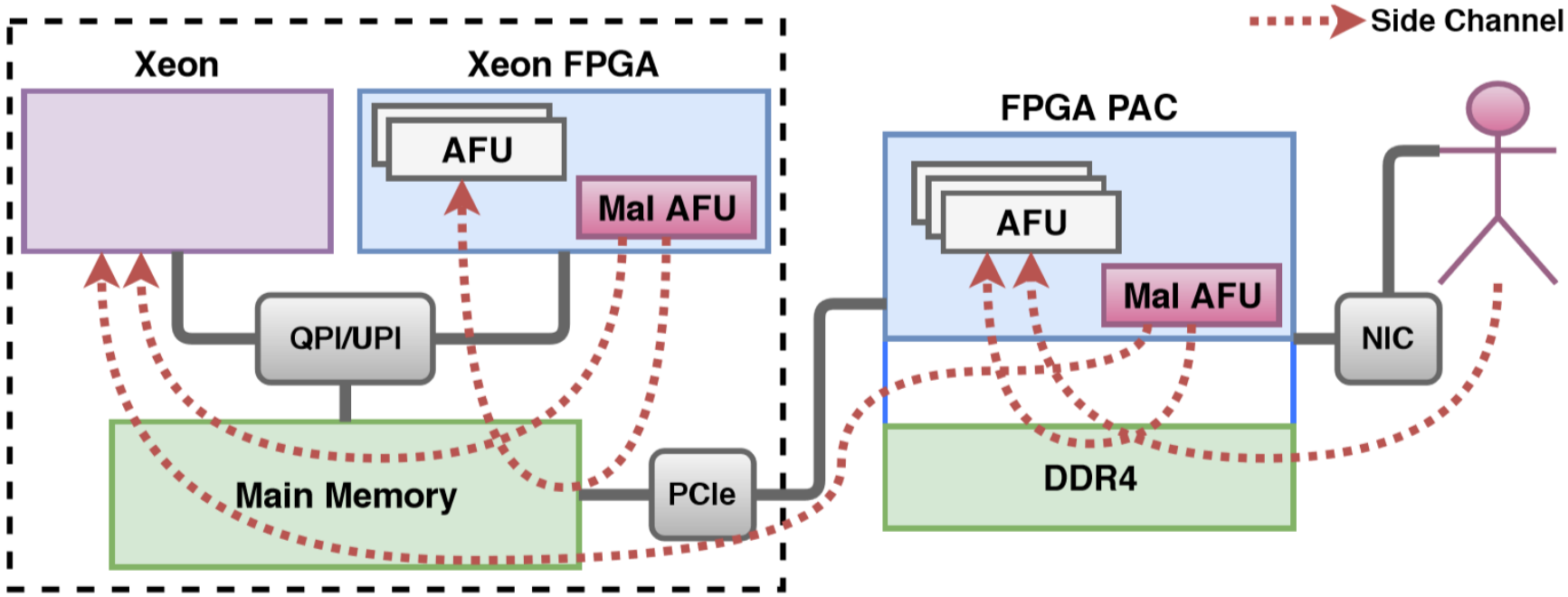
Accelerators in the Cloud

Side-channel Threats Shared FPGA-CPU Platforms

- FPGAs on the cloud can boost applications
  - Optimized Application-specific Hardware Configuration
  - e.g Real-time Artificial Intelligence

- New Attack Surface:
  - Accelerator Function Units (AFUs) placed on the FPGA can be used to interact with the CPU or other AFUs for malicious purpose.
    - AFU to AFU Attack
    - AFU to HPS Attack
    - AFU to CPU Attack
    - CPU to AFU Attack
    - Across VMS ?

Shared FPGA-CPU Platforms



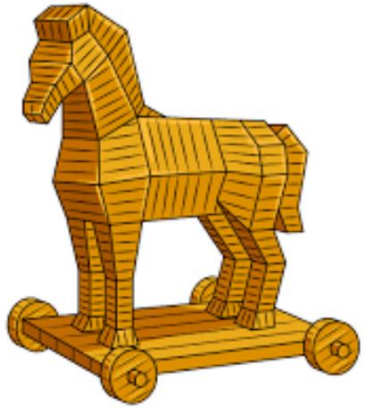
Attack Vectors

- Rowhammer

- Cache Attacks

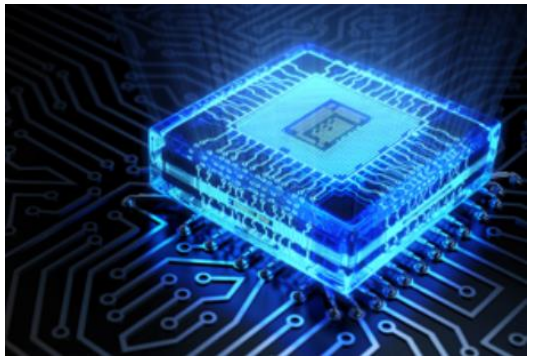
- DMA/IOMMU

- Trojan Bitstreams



- Cold Boot

- FPGA-centric Attacks

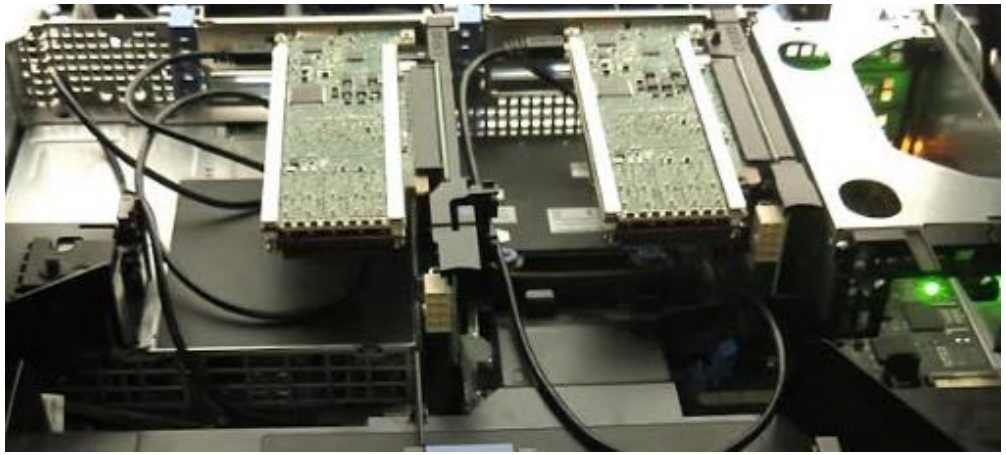


What is interesting about FPGA-CPU in the Cloud?

- Infancy, Attack/Defense Playground (Intel SGX in 2015)
- Customizable Hardware → More Devastating Attacks
  - E.g. Design your own timers, Direct access to memory interface, etc.
- Complex Threat Model

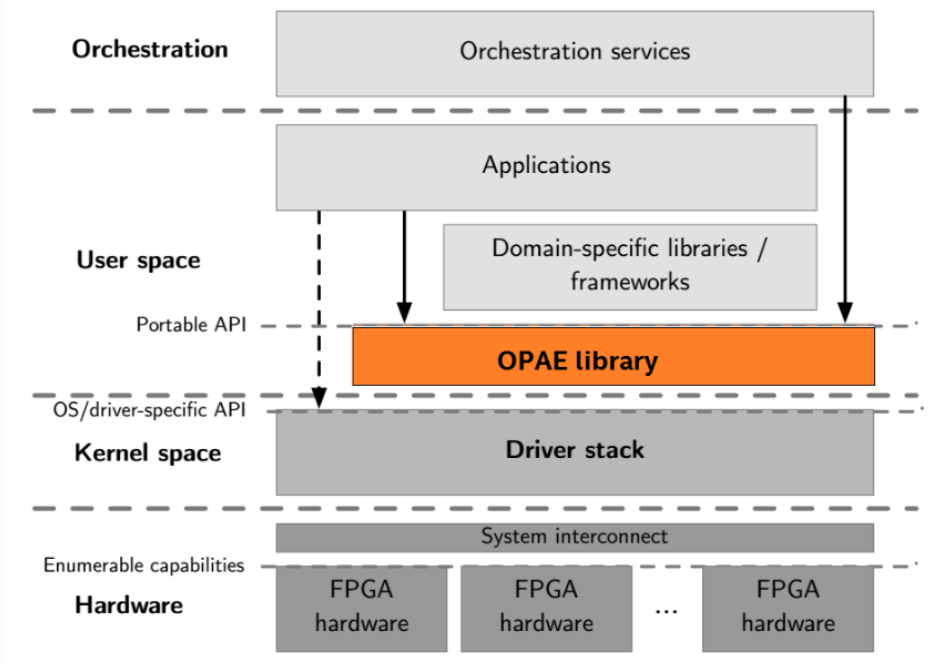
Lab/Collaboration Setup

- Weekly Meeting (2 Faculty + 3 Students = 5 people are actively involved.)
- Software
  - OPAE Stack
  - Intel Quartus (Synthesis)
  - KVM (Virtualization Scenario)
- Hardware
  - Remote Access to Intel Labs (Xeon)
  - Local Server including Intel PAC
  - Heavy Load Workstation (Synthesis)



Ongoing Work:
Threat Modeling and Security Analysis

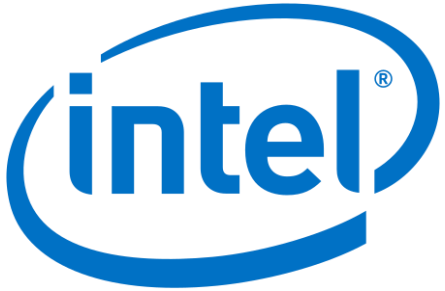
- Threat Modeling of the Technology based on Modern Use Cases
- Security Analysis of the Entire Stack Based on Available Resources



WPI + Lubeck Team

Acknowledgements

- Thanks to Carlos Rosaz, Matthias Schunter, Anand Rajan, Evan Custodio from Intel



THANKS

- Questions?

Ongoing Work:
Replicating µArch Attacks on FPGA-CPU Interface

- Memory Interface and the Cache Coherency Protocol
- Side-channel Analysis of Memory Operations