

# The Secure Processor Paradox: When Security Metadata Worsens Microarchitecture Security

Fan Yao

[fan.yao@ucf.edu](mailto:fan.yao@ucf.edu)

<https://casrl.ece.ucf.edu>

University of Central Florida



HASP 2024 Keynote  
Nov. 2<sup>nd</sup> 2024



# Outline

---

- The Battlefield in Microarchitecture Security
- When Secure Processors Break uArch Security: the Metadata Perspective
- Architecting Side-channel Resistant Secure Metadata Mechanisms
- Takeaways and Conclusions

# Hardware as the New Battlefield for Security

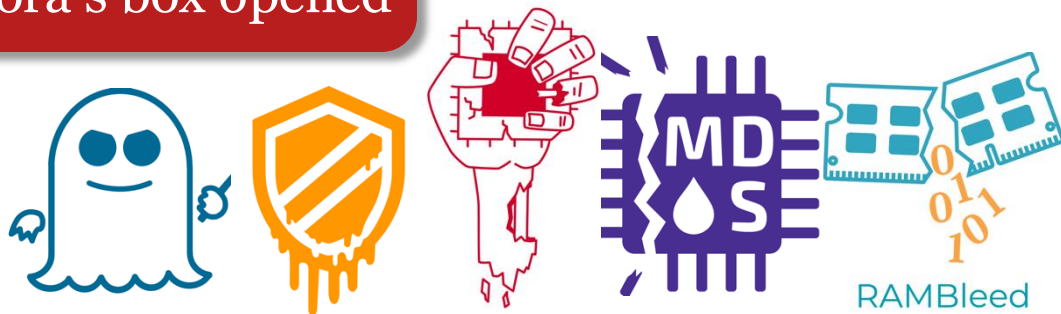
- **Performance** has been the overly-focused goal for HW design
- **Security** of processor/hardware has been mostly overlooked

COMPUTING

## At Least Three Billion Computer Chips Have the Spectre Security Hole

Companies are rushing out software fixes for Chipmageddon.

Pandora's box opened



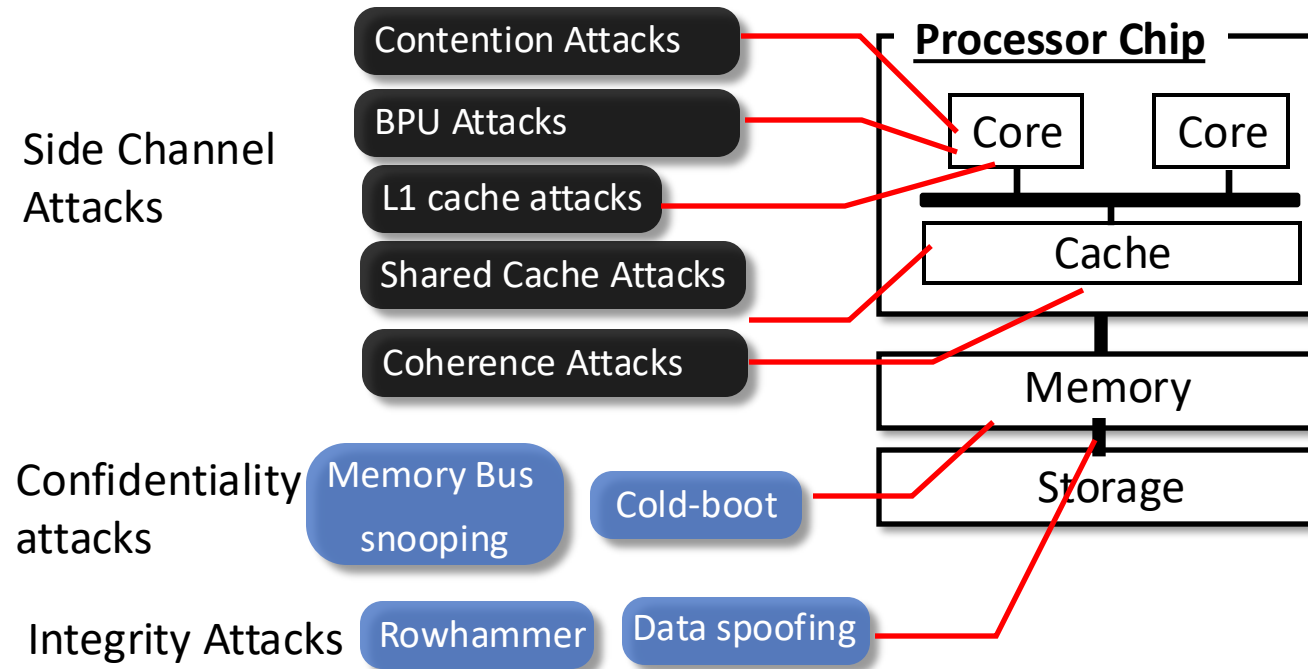
**“We’ve made tremendous gains in IT in the past 40 years, but if security is a war, we’re losing it”**



2018 Turing Award Lecture

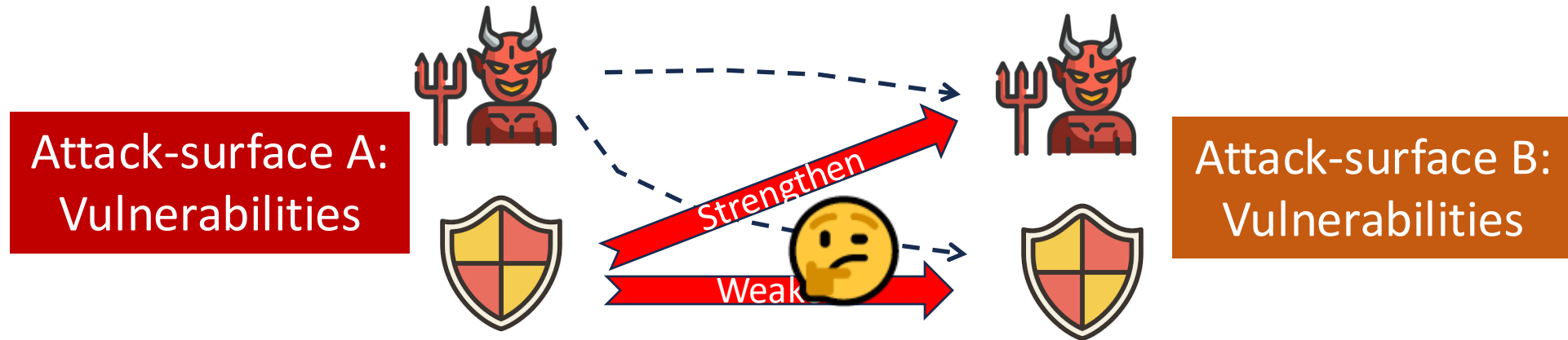
Source: iscaconf.org

# A Global Picture of Hardware Security



What is the overall security landscape with the co-existence of so many uarch/HW vulnerabilities?

# No Easy Answer to the Question!



💡 What we already know: uArch is not a standalone problem from [attack perspective](#)

Example 1-[**SpecHammer SP'22**]: Rowhammer enhances attack capability of Spectre

Example 2-[**PACMAN ISCA'22**]: Speculation breaks **HW-based pointer authentication**

Do secure mechanisms compose well in computing systems?

# uArch Security in the Era of Secure Processors

Security of data due to on-chip usage

## uArch Security

- ✓ Side and covert channels
- ✓ Timing-based leakage
- ✓ Attacks and defenses



Off-chip Data Security and HW-enforced access control

## Secure Processors

- ✓ CPU as root-of-trust
- ✓ Secure memory architectures
- ✓ Trusted execution env. (TEE)

uArch Security Architects:

Existing side channels work in secure processors!

(e.g., Cache attacks and port contention attacks)



Secure Processor Architects:

uArch attacks should be treated **individually** by architects/SE

Typical TEE threat models **exclude** side channels

# But...Really?

---

Leakage of data due to on-chip usage

## uArch Security

- ✓ Side and covert channels
- ✓ Timing-based leakage
- ✓ Attacks and defenses

Off-chip Data Security and HW-enforced access control

## Secure Processors

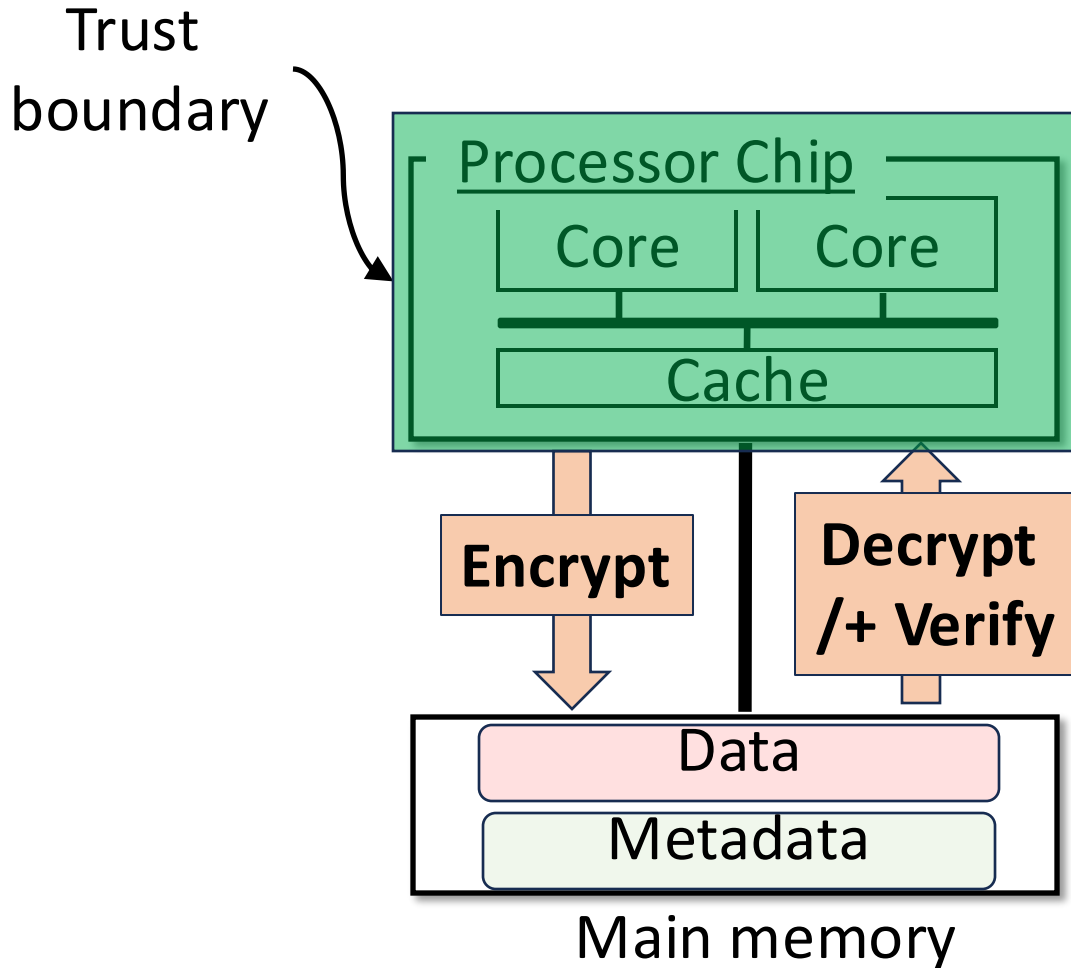
- ✓ CPU as root-of-trust
- ✓ Secure memory architectures
- ✓ Trusted execution env. (TEE)
- ✓ E.g., Intel SGX, TDX, AMD SEV



## WHAT IF:

The underlying secure processor designs break the assumption we made about microarchitecture security?

# Secure Processor Architectures in a Nutshell



- Decades of research on secure processors

## Caches and Hash Trees for Efficient Memory Integrity Verification\*

Blaise Gassend, G. Edward Suh, Dwaine Clarke, Marten van Dijk<sup>†</sup> and Srinivas Devadas  
Massachusetts Institute of Technology  
Laboratory for Computer Science  
Cambridge, MA 02139, USA

## Improving Cost, Performance, and Security of Memory Encryption and Authentication \*

Chenyu Yan<sup>†</sup>, Brian Rogers<sup>‡</sup>, Daniel Engleder<sup>†</sup>, Yan Solihin<sup>‡</sup>, Milos Prvulovic<sup>†</sup>  
<sup>†</sup>College of Computing Georgia Institute of Technology  
<sup>‡</sup>Dept. of Electrical and Computer Engineering North Carolina State University

## Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly \*

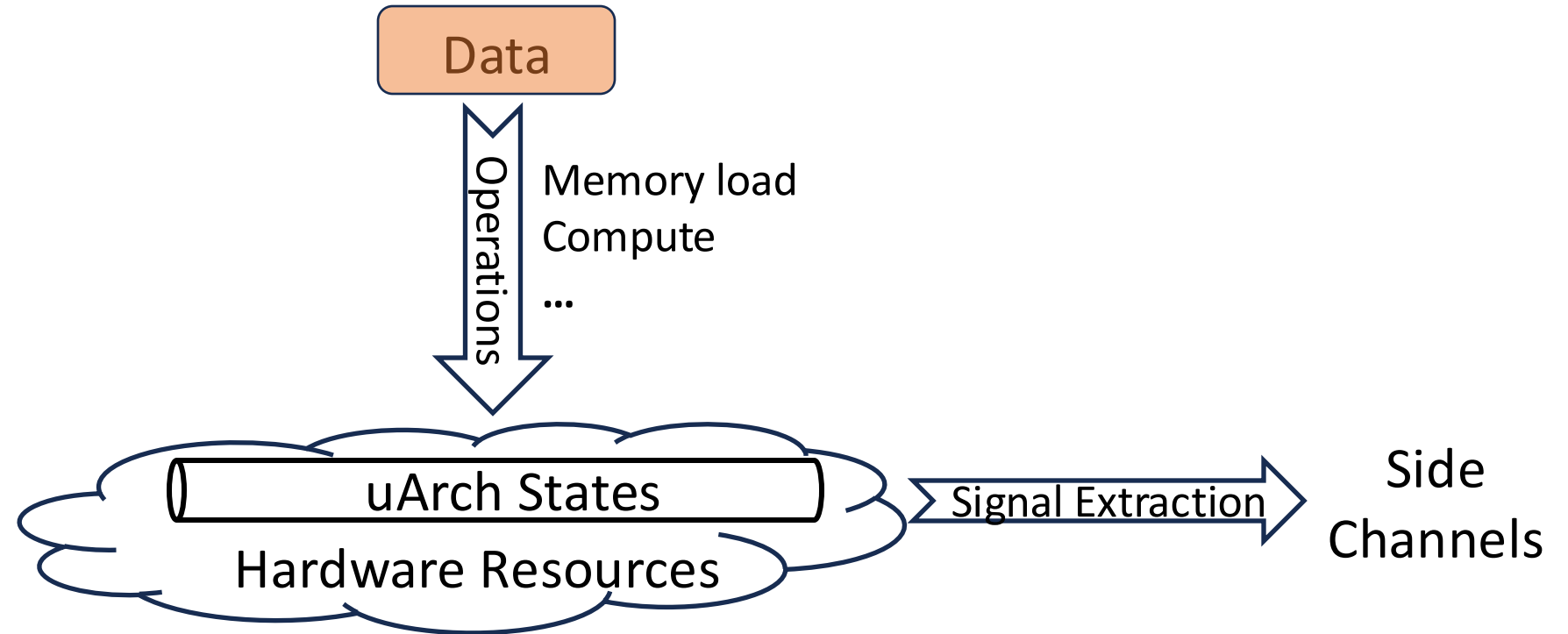
Brian Rogers, Siddhartha Chhabra, Yan Solihin  
Dept. of Electrical and Computer Engineering  
North Carolina State University  
{bmrogers, schhabr, solihin}@ncsu.edu  
Milos Prvulovic  
College of Computing  
Georgia Institute of Technology  
milos@cc.gatech.edu

- Foundations of today's TEE-enabled processors
- Industry solutions: Intel SGX/TDX, ADM SEV etc.

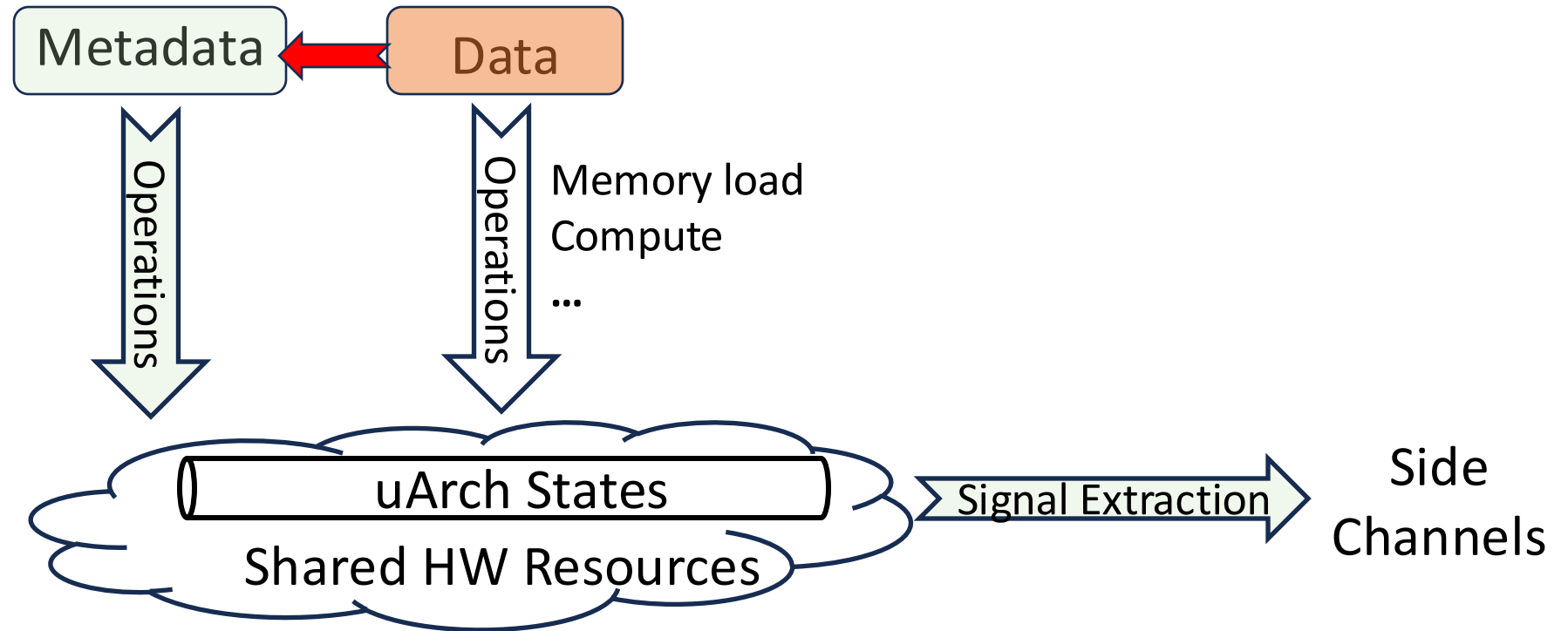


# uArch Attacks: The Classical Data-centric View

---



# The New View from Metadata



**MetaLeak:** Uncovering Side Channels in Secure Processor Architectures Exploiting Metadata, Md Hafizul Islam Chowdhuryy, Hao Zheng and Fan Yao, ISCA'2024

Microarchitecture security investigation in the design space of secure processors w.r.t. metadata mechanisms

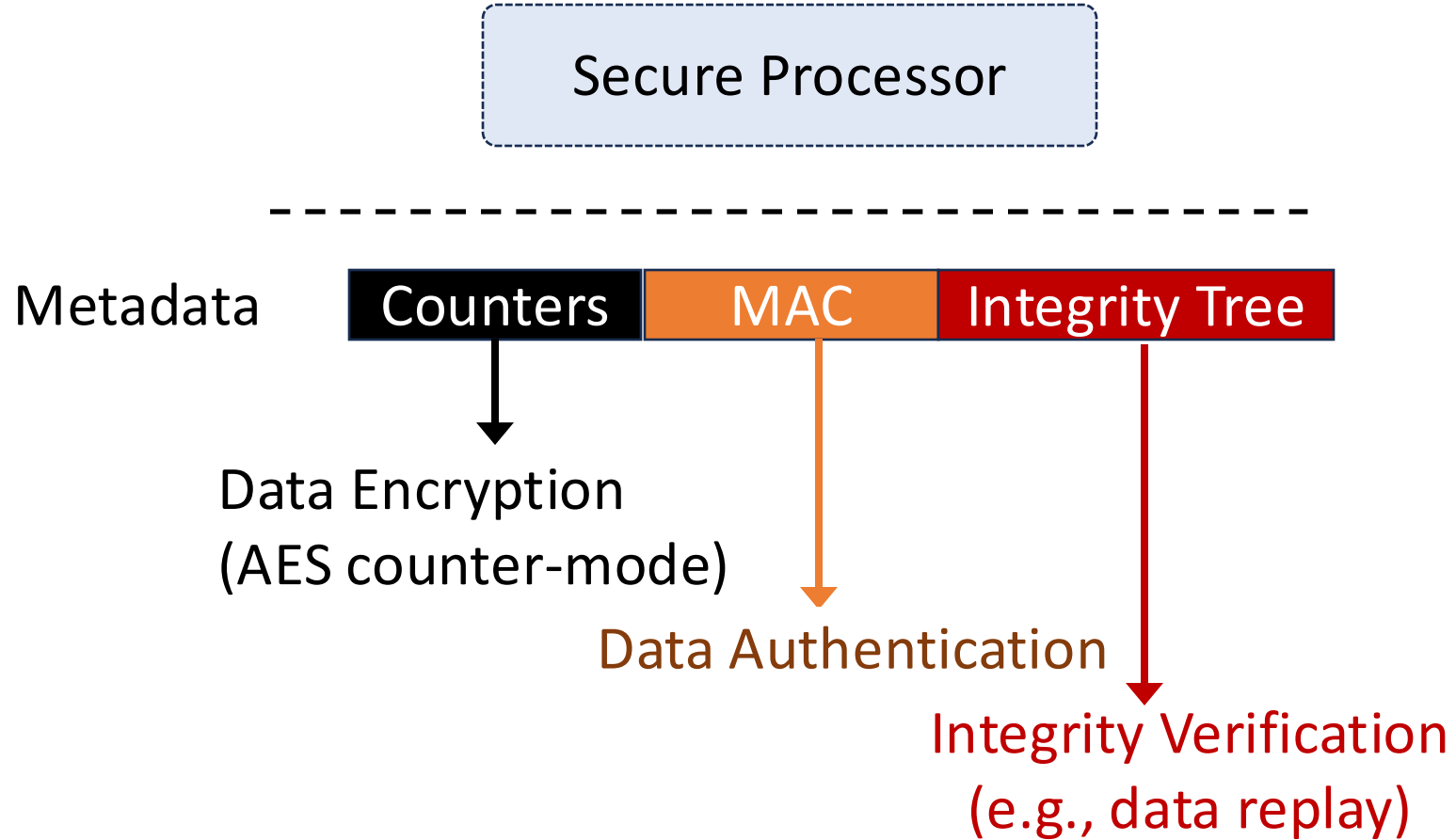
# Outline

---

- The Battlefield in Architecture Security
- **When Secure Processors Break uArch Security: the Metadata Perspective**
- Architecting Side-channel Resistant Secure Metadata Mechanisms
- Takeaways and Conclusions

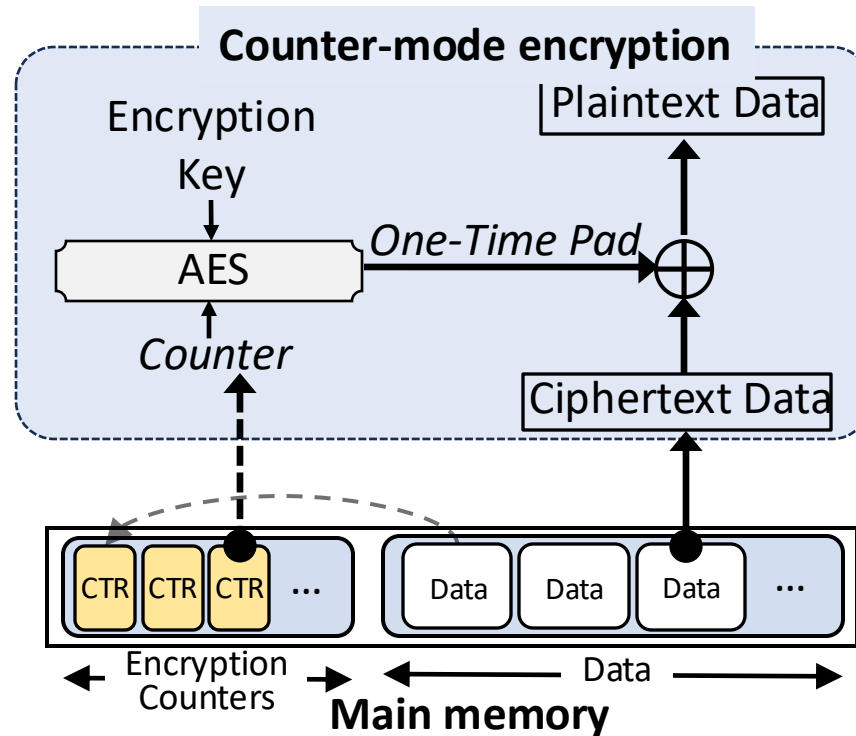
# Metadata in Secure Processors

---



# Investigating Encryption Counter Mechanisms

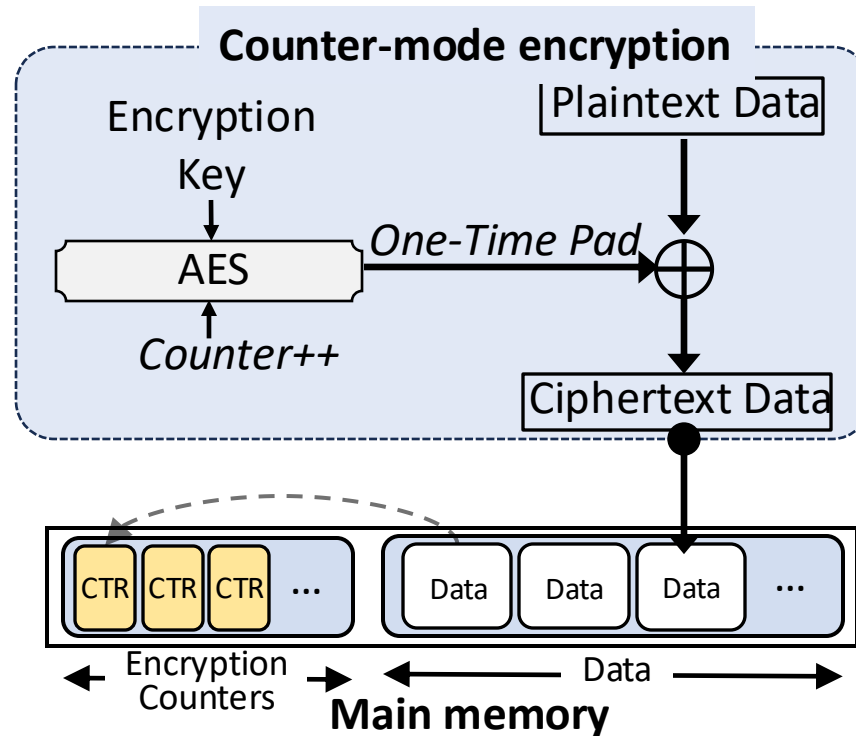
On data read



**Read:** load counter -> generate OTP -> XOR (C)

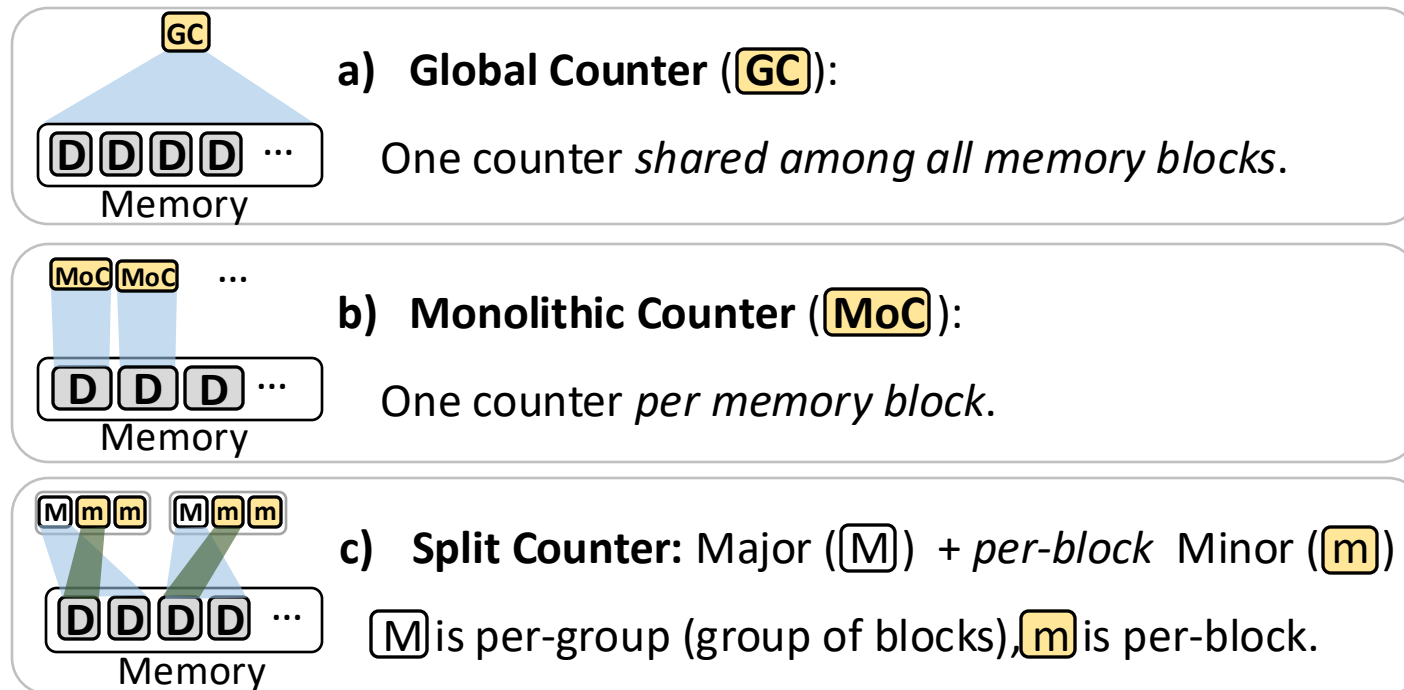
# Investigating Encryption Counter Mechanisms

On data write



Store: Inc. counter -> generate OTP -> XOR (P)

# Various Counter Mode Schemes



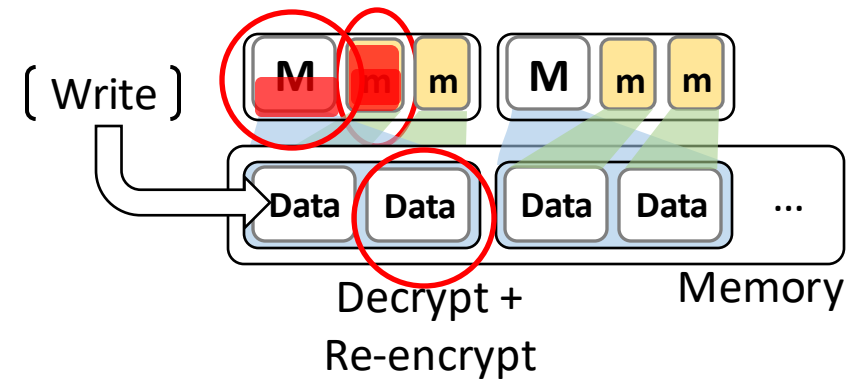
Counters are subject to **overflow** -> counter wrap around -> **re-encryption of data blocks** in the counter-sharing group

# Timing Vulnerabilities in Memory Encryption

## Abstract Counter-mode Encryption Mechanism

```
Input:  $P_t$ : current block to encrypt
Function Encrypt( $P_t$ ):
   $ctr_{old} = ctr$ 
  Increment ( $ctr$ ) // Increment the counter
  if  $ctr_{old} = ctr^{max}$  then // Overflow detected
    // Re-encrypt memory blocks in group
    for  $P_i$  in  $\{G - P_t\}$  do
      Decrypt( $P_i$ ) with old counter
      Encrypt( $P_i$ ) with new counter
    Encrypt( $P_t$ ) using  $ctr$ 
  else
    Encrypt( $P_t$ ) using  $ctr$ 
```

- **Split counter:** Major ( $\boxed{M}$ ) + *per-block* Minor ( $\boxed{m}$ )



## Vulnerability Class-1

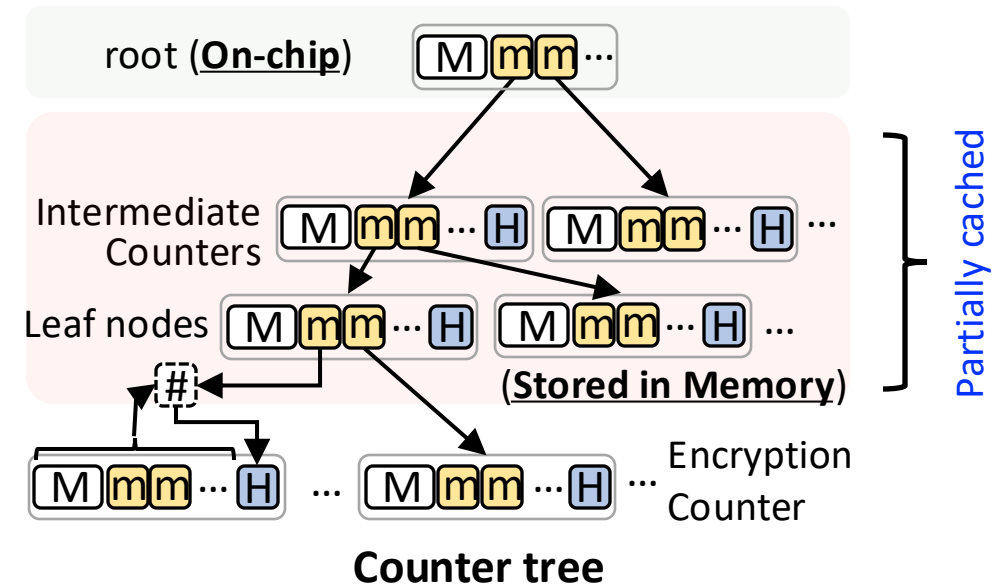
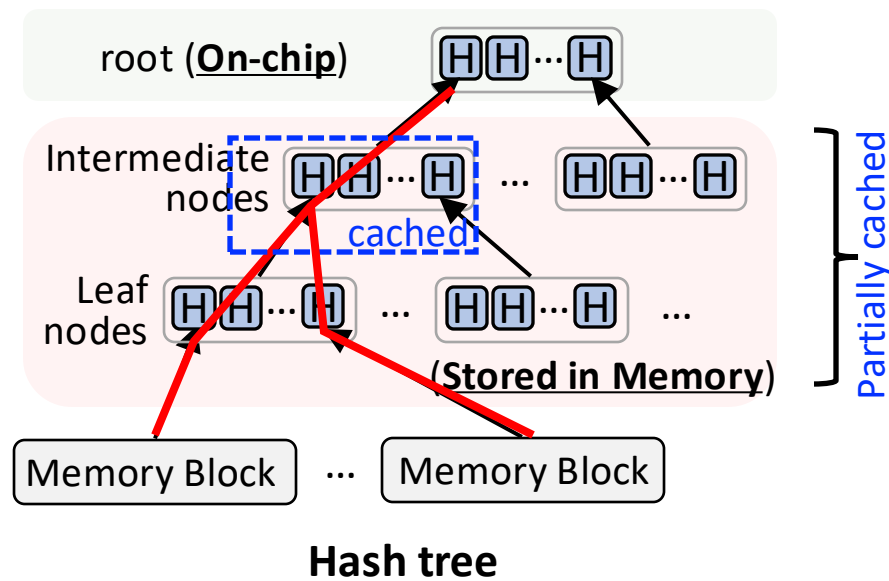
Encryption counters create **metadata state dependent** uArch paths for writes

1. **Slower:** Program data write leading to **counter overflow**
2. **Faster:** For regular write cases (not triggering counter overflow)



# Investigating Integrity Verification Schemes

- **Memory integrity protection:** Typically performed using **integrity tree**
  - Root of tree kept **on chip**
  - **Hash-based tree:** Each node in tree is a *hash* of its child nodes
  - **Counter-based tree:** Each node contains *write counters* for its child nodes



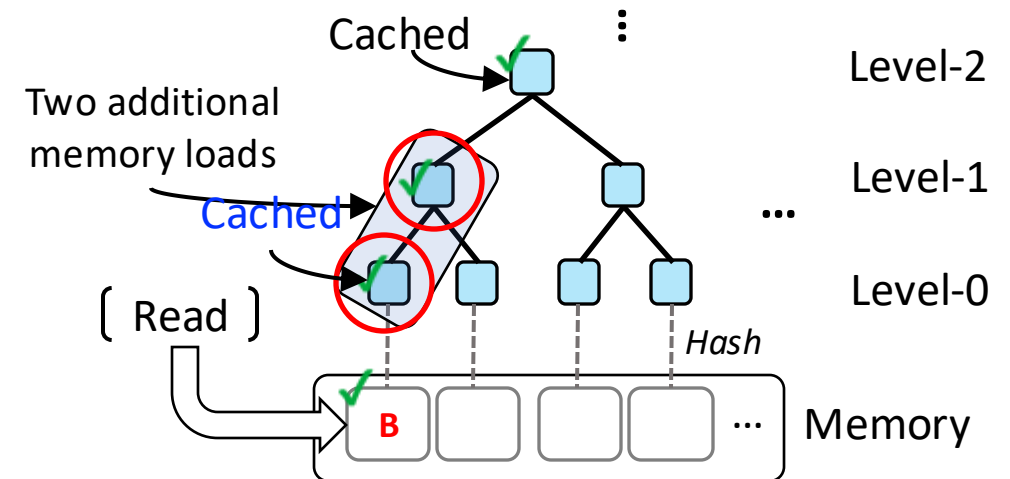
# Timing Vulnerabilities in Integrity Verification

**Observation: Integrity tree traversal typically proceeds to the first cached node**

Abstract Integrity Verification Mechanism

```
Input:  $\mathcal{B}$ : // Memory block to verify
Leaf node:  $N_A^i$  // The  $i^{th}$ -level ancestor tree node for an attached memory block  $\mathcal{B}$ 
Function Verify( $\mathcal{B}$ ):
  // Assume block  $N_A^L$  is cached
  for  $i$  from 1 to  $L$  do
    Load Block( $N_A^i$ )
    Verify (Block( $N_A^{i-1}$ )) with  $N_A^i$ 
  Verify( $\mathcal{B}$ ) with  $N_A^0$ 
```

- Integrity verification for data reads:

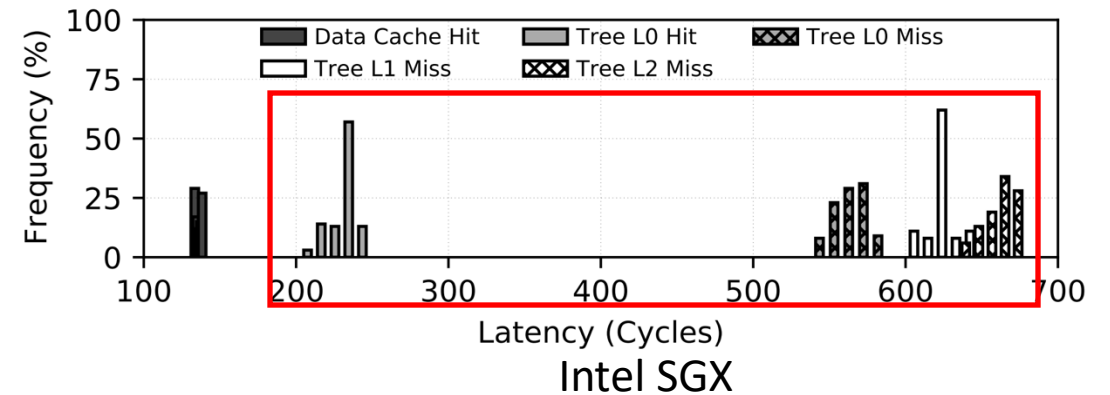
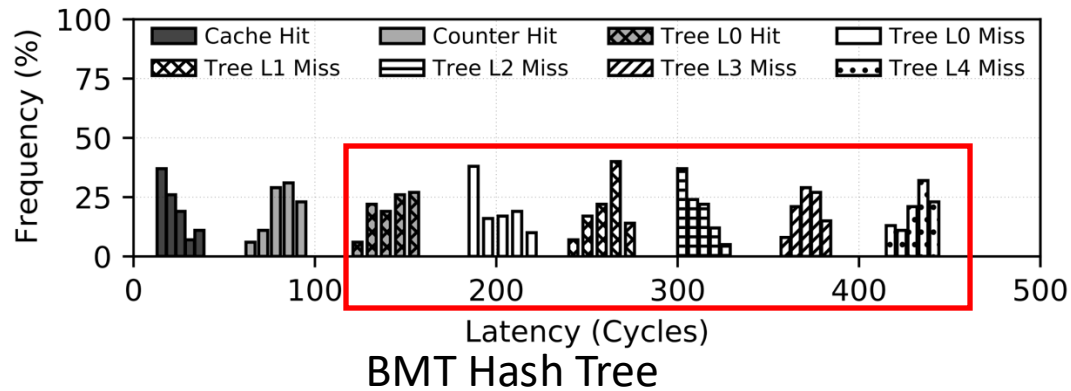


## Vulnerability Class-2

Integrity verification path varies according to **tree node caching state**  
Integrity tree traversal can lead to data reads with **highly-variant latencies**

# Latency Characterization: Secure Processor Reads

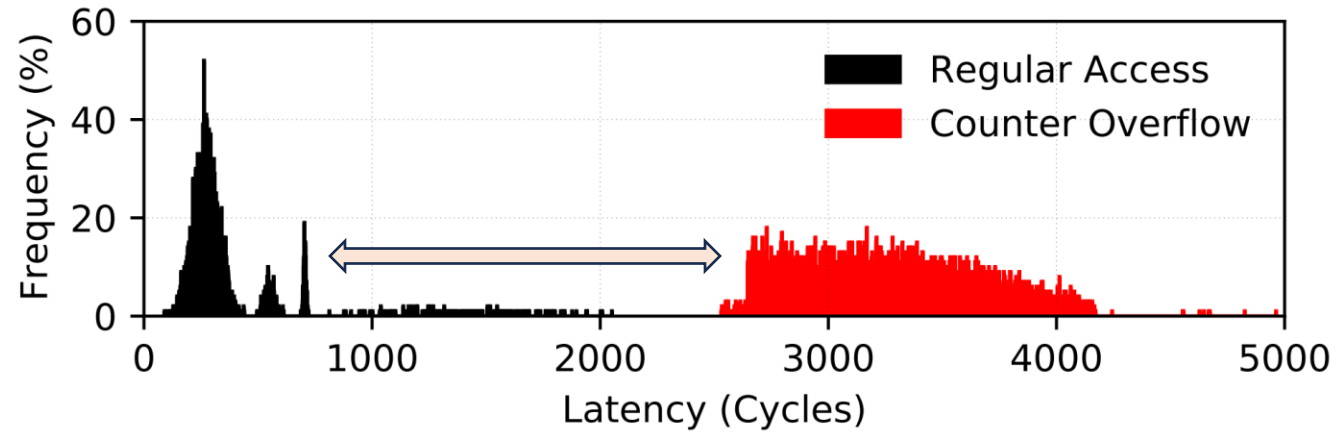
## Latency distribution due to integrity tree traversal



Highly distinguishable **multi-level latencies** w.r.t. integrity metadata accesses

# Latency Characterization: Secure Processor **Writes**

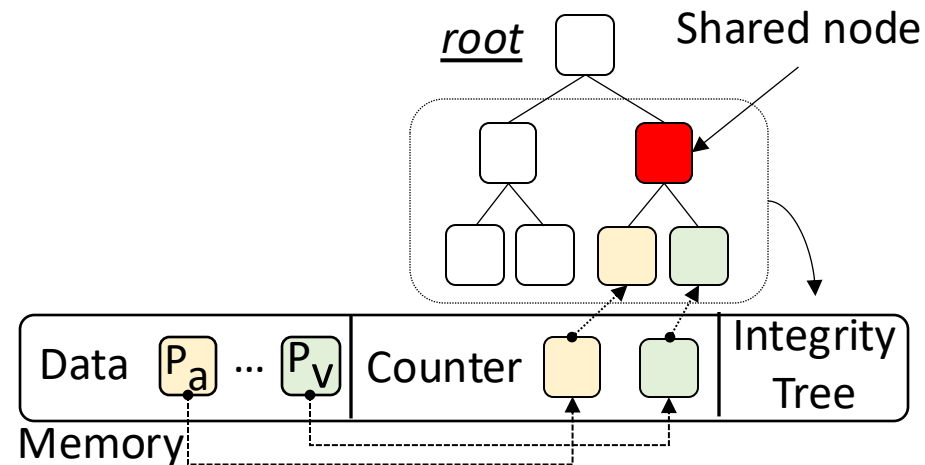
---



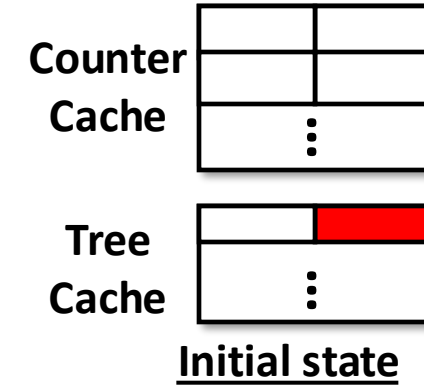
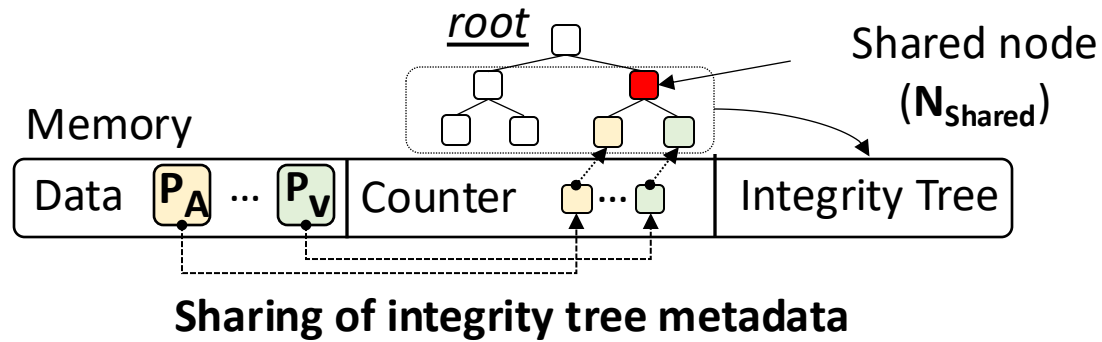
Memory access latency impacted by **counter overflow**

# Side Channels Exploiting Integrity Tree Metadata

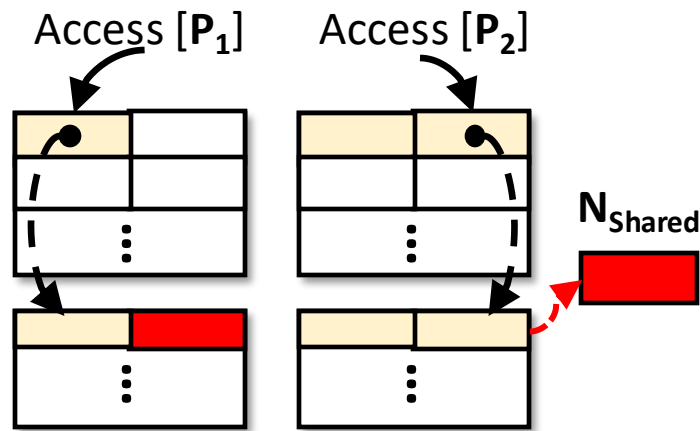
- Integrity tree is global
  - Integrity tree creates **shared tree blocks across security domains** (e.g., enclave instances).
  - Enabling **shared-memory** side channel even **without explicit data sharing**.



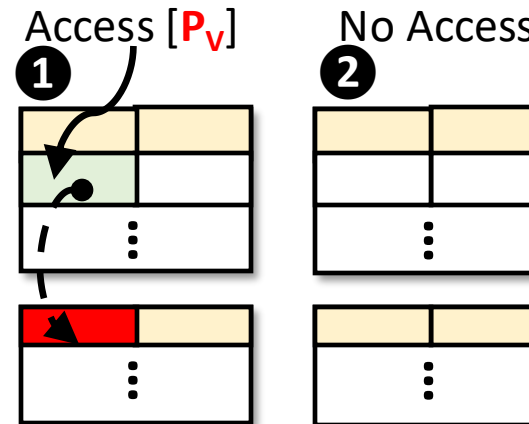
# High-level Exploitation Mechanism



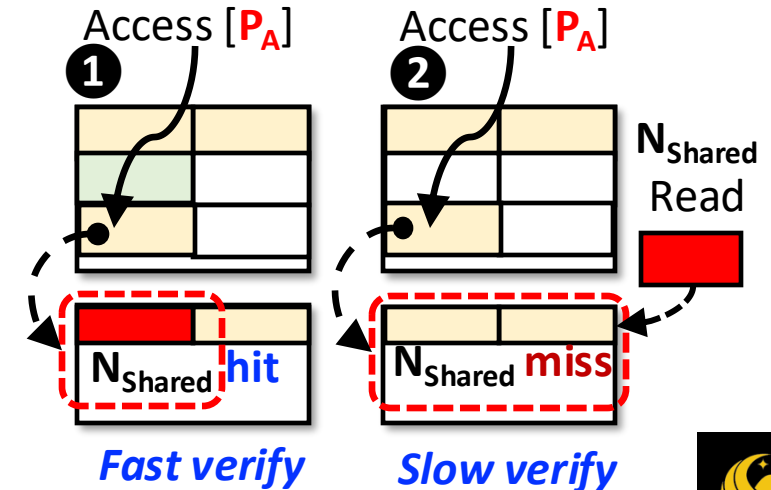
## Step 1: Evict shared tree block



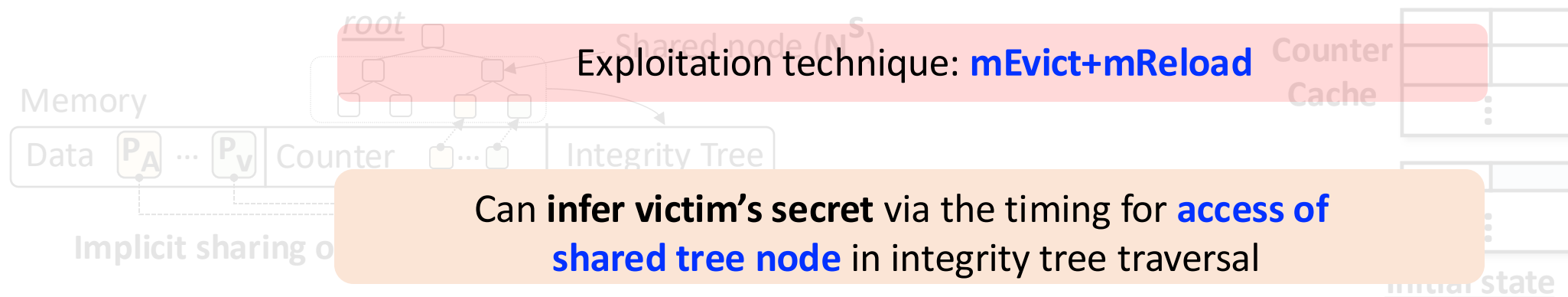
## Step 2: Victim execution



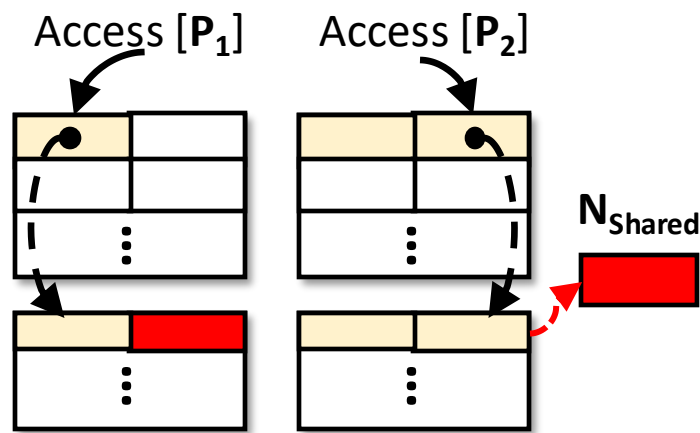
## Step 3: Timed load of a data block in $P_A$



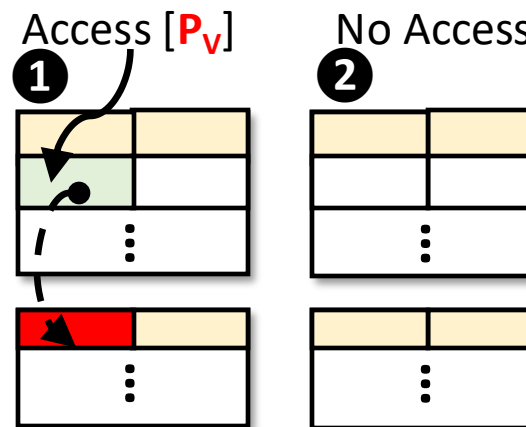
# High-level Exploitation Mechanism



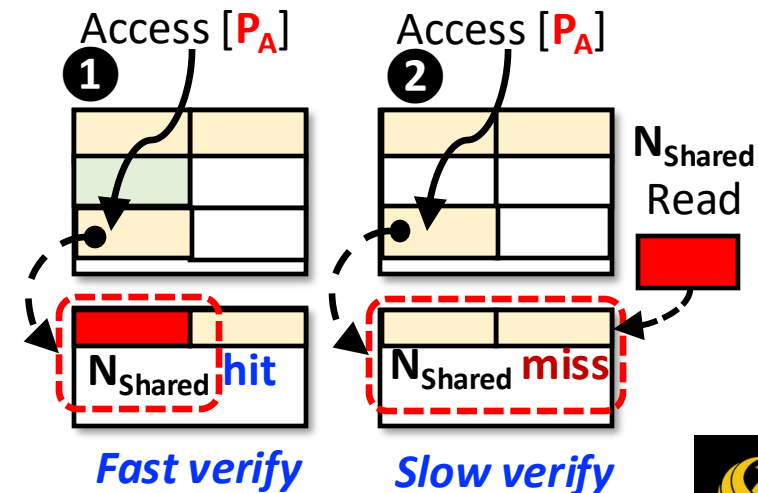
**Step 1:** Evict shared tree blocks



**Step 2:** Victim execution

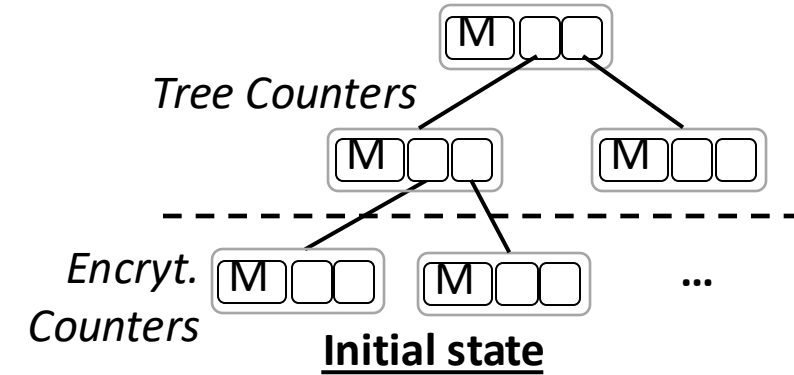


**Step 3:** Timed load of a data block in  $P_A$



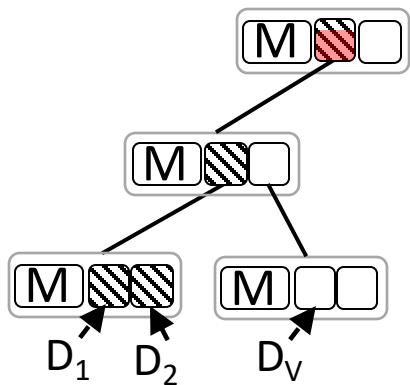
# Side Channels Exploiting Shared Counter

- **Recall:** Counter overflow leads to **high latency variations**
- Encryption counter blocks typically not shared across pages
- Counters in **counter-based integrity tree** are shared across domains



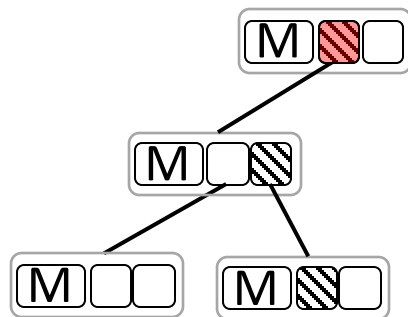
**Step 1:** Preset shared counter

Write  $[P_1, P_2]$

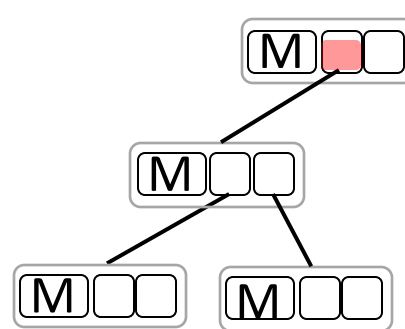


**Step 2:** Victim execution

① Write  $[P_v]$

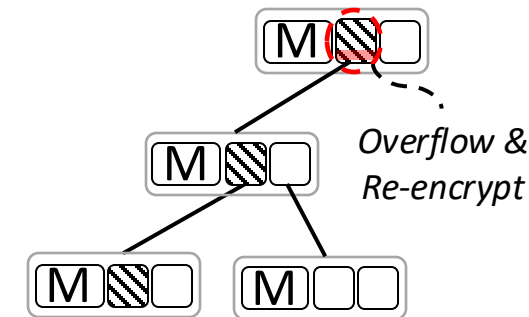


② No Write

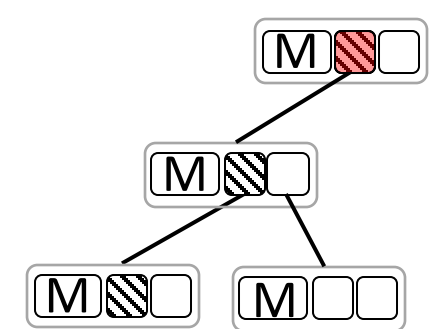


**Step 3:** Infer counter overflow

① Write  $[P_1] \rightarrow$  Read  $[P_A]$  ② Write  $[P_1] \rightarrow$  Read  $[P_A]$



**Slow**



**Fast**



# MetaLeak-C: High-level Exploitation Mechanism

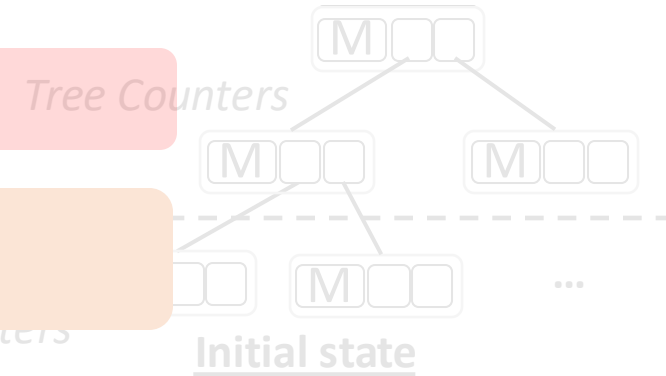
Recall: Counter overflow leads to high latency variations.

Exploitation technique: **mPreset+mOverflow**

- Encryption counters are typically per-page → not shared across processes.

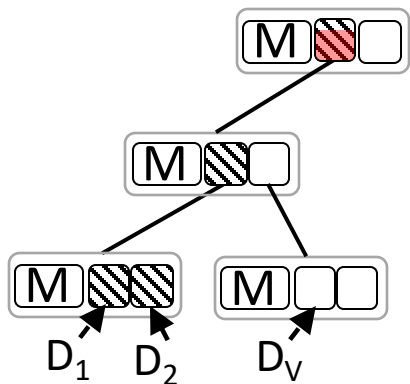
- Counters in Co

Allows **monitoring of victim's targeted write** via inferring **counter state**



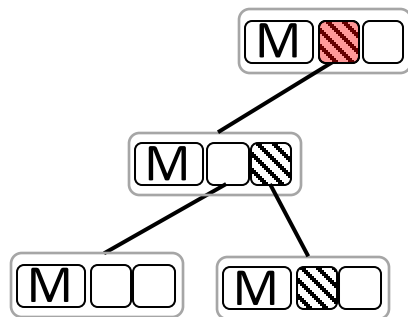
**Step 1:** Preset shared counter

Write [ $P_1$ ,  $P_2$ ]

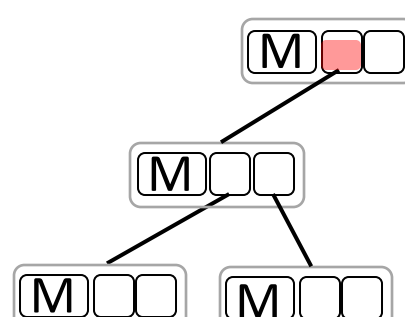


**Step 2:** Victim execution

1 Write [ $P_v$ ]

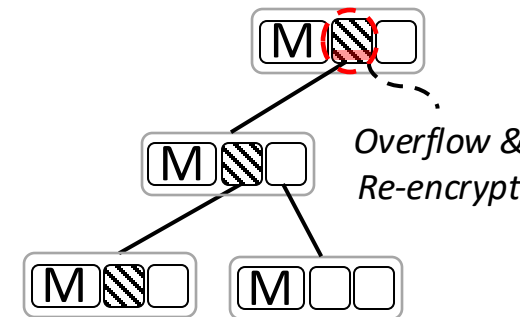


2 No Write

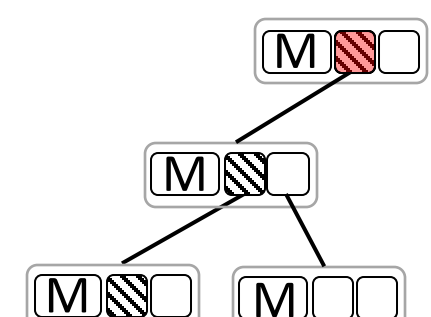


**Step 3:** Infer counter overflow

1 Write [ $P_1$ ] → Read [ $P_A$ ] 2 Write [ $P_1$ ] → Read [ $P_A$ ]



Slow

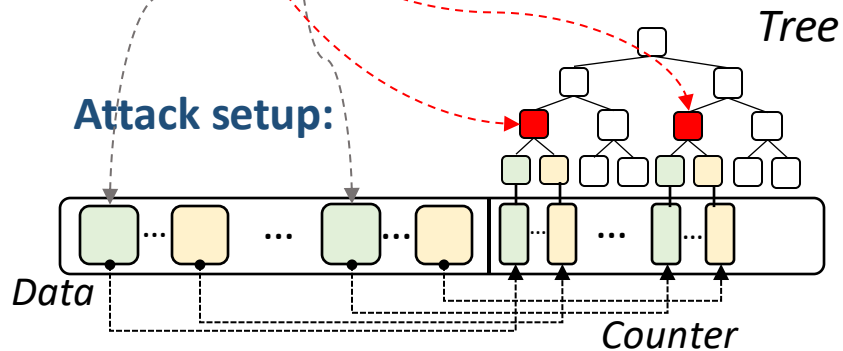





Fast

# Attacks against Real-world Programs: libjpeg

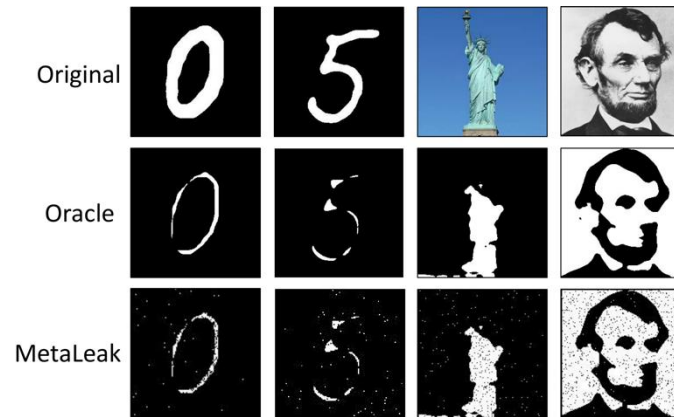
## Exploited gadget:

```
1  encode_one_block (...) {
2  ...
3  /* Encode the coefficients */
4  for (k = 1; k < DCTSIZE2; k++) {
5    if (block[jpeg_natural_order[k]] == 0) {
6      r++;
7    } else {
8      ...
9      /* Check for out-of-range coefficient */
10     if (nbits > MAX_COEF_BITS) { ... }
11   }
12 }
```



-  Victim's memory block
-  Attacker's memory block
-  Shared tree block

## Results:



Victim access detection accuracy: **94.3%**

Image reconstruction using Integrity tree side channels

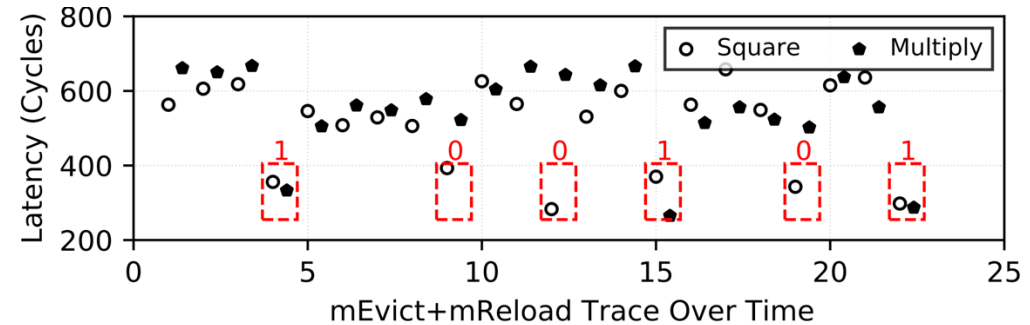
# Attacks against Real-world Programs: libgcrypt

## Exploited gadget:

```
1  gcry_mpi_powm (...) {
2  ...
3  /* Main loop */
4  for (;;) {
5  /* Square operation */
6  _gcry_mpih_sqr_n_basecase(...);
7  /* Check if exponent bit is 1 */
8  if ( (mpi_limb_signed_t)e < 0 ) {
9  /* Multiplication operation */
10 _gcry_mpih_mul_karatsuba_case(...);
11 }
12 }
```

Attack mounted in SGX processor

## Results:



mEvict+mReload latency traces for secret exponent bit '100101'

Accuracy of exponent bit stealing:  
91.2%

# How Metadata Mechanisms Break uArch Security

Existing uArch defenses cannot mitigate the metadata-based attacks

Assumptions made by typical microarchitectural defenses (data-centric)

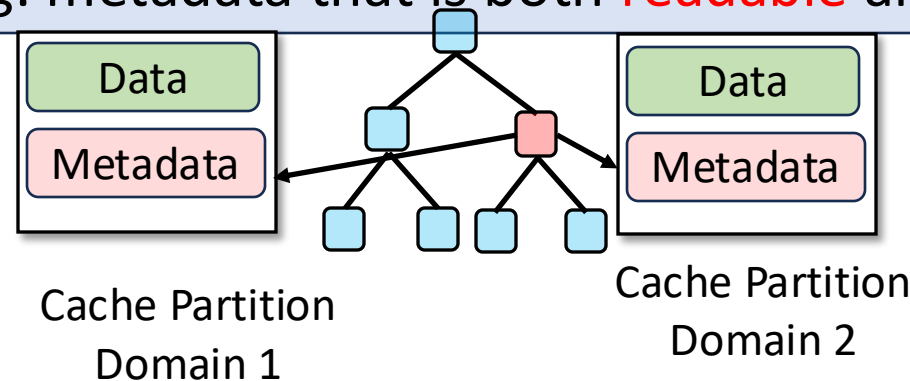
For read-only memory sharing exploits

--> **Disabling data/memory sharing** on untrusted domains

For interference-based exploit (no memory sharing)

--> **Isolation** of shared HW resource for **data access**

New dimension of sharing: metadata that is both **readable** and **writable** (indirectly)



Not compatible with coherence mechanisms

# Takeaways

---

- Should be cautious about metadata usage in secure processors
- Scope of metadata-based mechanisms can **be much broader**
  - Industry: many variants of TEEs:
    - ❖ Intel, AMD, ARM, Apple and Qualcomm
  - Academia: burgeoning of proposals of secure processor designs
    - ❖ More compact counters (easier overflow?)
    - ❖ TEE in GPUs and accelerators
- Have microarchitecture security mindset in secure processor designs

# Outline

---

- The Battlefield in Architecture Security
- When Secure Processors Break uArch Security: the Metadata Perspective
- **Architecting Side-channel Resistant Secure Metadata Mechanisms**
- Takeaways and Conclusions

# Observations from the Metadata Exploit

---

- Metadata sharing breaks the **assumption of sharing** in uArch Security
- Hard to address from the *classical* uArch defense perspective

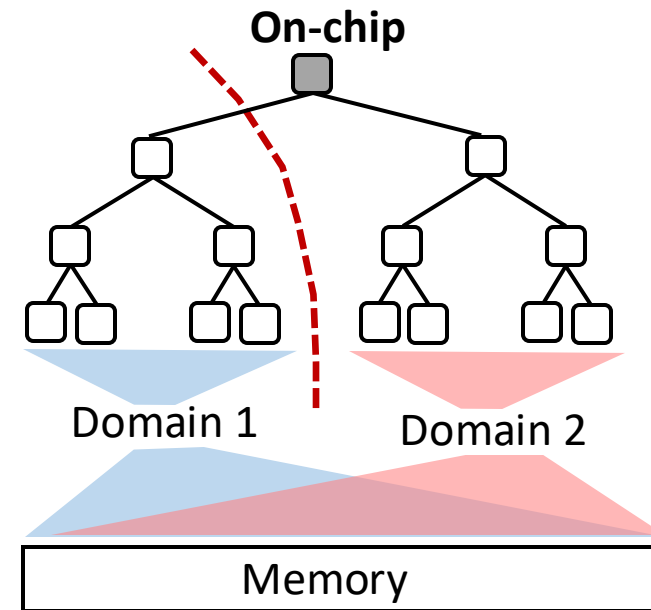
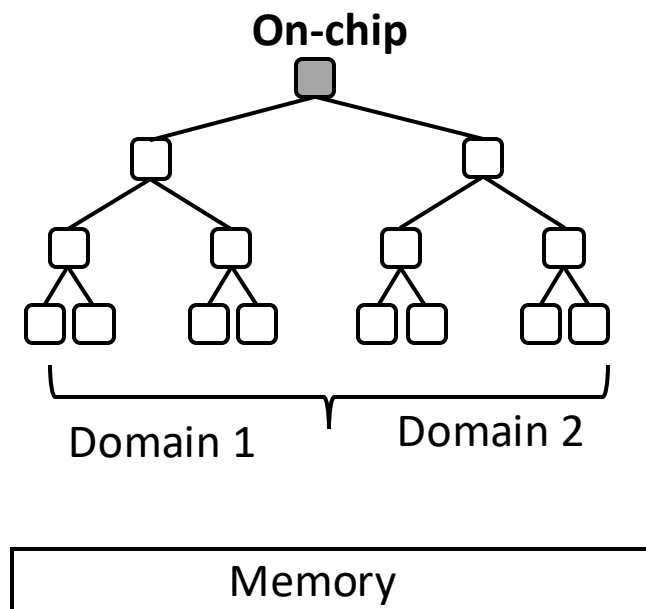
Need to rethink the secure processor designs for microarchitecture security!

IvLeague: Side Channel-resistant Secure Architectures Using Isolated Domains of Dynamic Integrity Trees Md Hafizul Islam Chowdhury and Fan Yao, MICRO 2024

Architectural support for leakage resistant integrity metadata mechanisms in secure processor

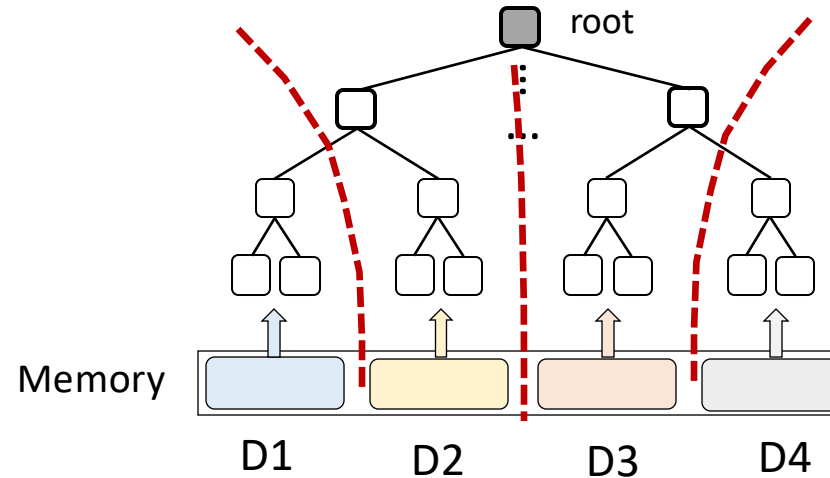
# Side-channel Resistant Integrity Metadata Mechanisms

- Main idea: **metadata-level isolation** for integrity verification (IV)
  - Ensure no tree node sharing in memory between domains



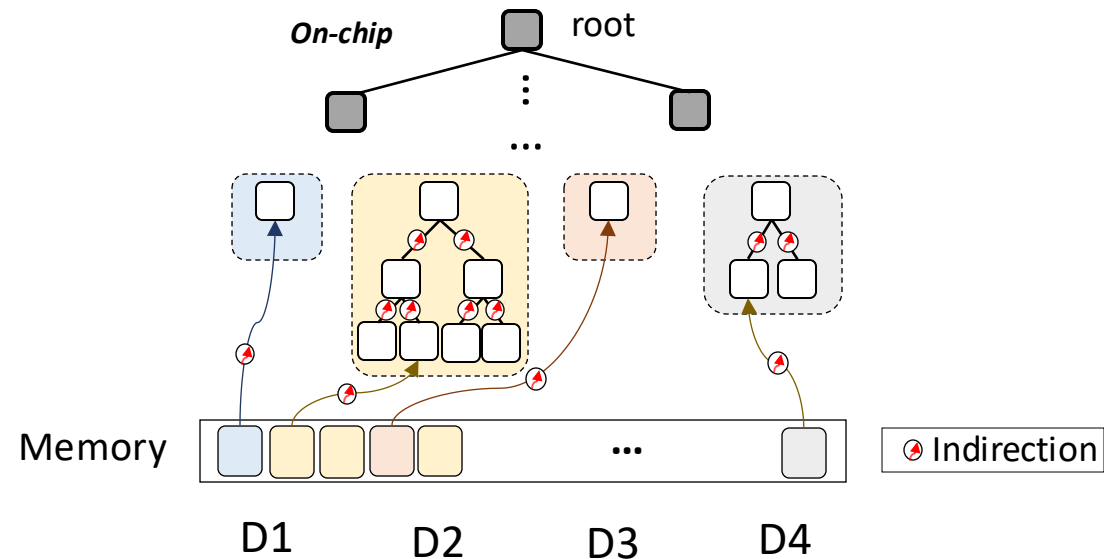


# Statically Partitioning the Integrity Tree?



- Static partitioning
  - Fixed number of supported domains, fixed coverage per domain
  - Low domain management overhead (similar to global tree)
- Does not scale well according to runtime domains (e.g., enclaves)
- Could not support application with larger dynamic memory footprint
- Rely on the OS (**untrusted**) to map pages from fixed region to domains

# Fully Dynamic Isolated Integrity Trees?

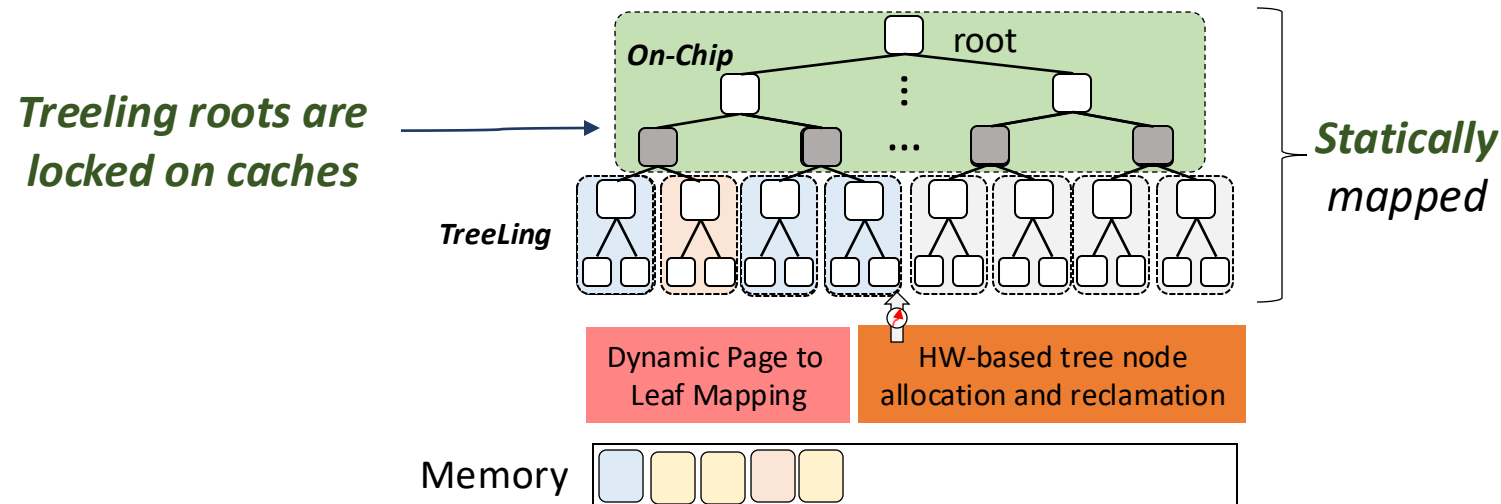


- Build and grow per-domain tree at runtime-> flexible memory coverage
- High runtime domain scalability
  - High metadata overhead for tree construction (i.e., indirection)
  - High tree traversal overhead -> long IV latency for reads

# IvLeague: Dynamic Domains of Isolated Static Trees



Split the integrity tree into many *small but fixed-sized* sub trees (TreeLing)



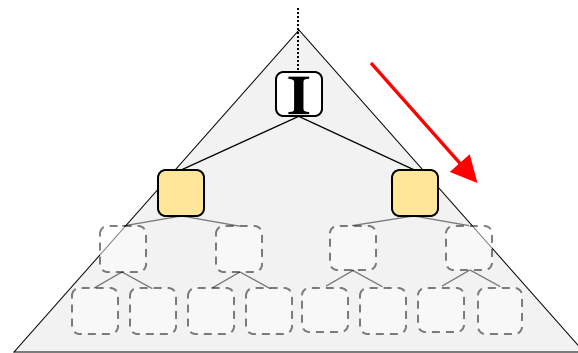
- Each sub-tree (TreeLing) is statically mapped, *no indirection needed for leaf-to-root traversal*
- TreeLings are allocated to domain on-demand, *resize integrity coverage during runtime*
- Support a large number of runtime domains (upto 4K)

# IvLeague: Performance Optimization Opportunities



IvLeague's **dynamic intra- and inter-TreeLing** management enables performance optimization not applicable in default secure processor designs

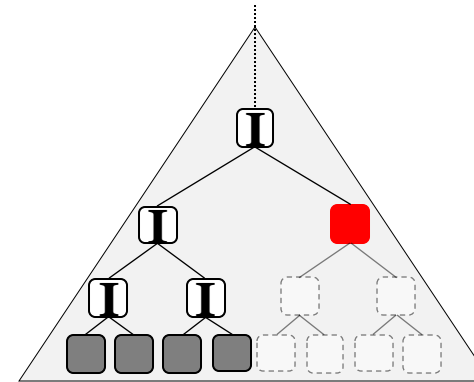
**I** Intermediate tree node      **■** Utilized tree node  
**□** Unutilized tree node



**Intermediate node as leaf**  
**Gradual intra-tree expansion**

Reduced IV path length

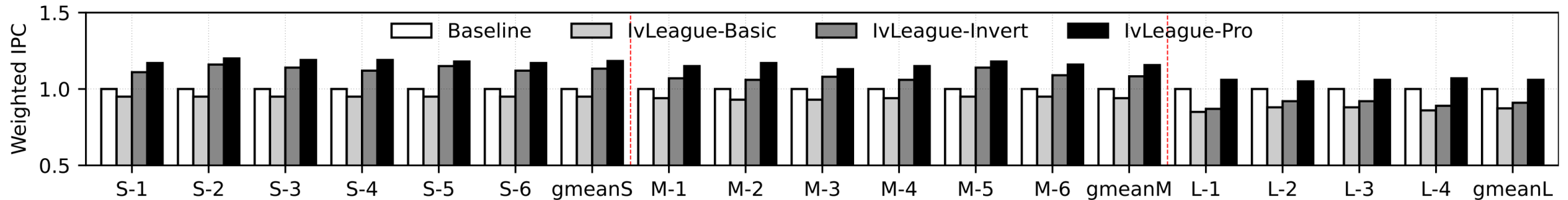
**■** Regular page      **■** Hot page



**Tracking and swapping**  
**mapping for hot pages**

Fast hotpage verification

# How does IvLeague Perform



Comparison of performance (i.e., Weighted IPC normalized to Baseline) under different schemes.

**Performance of IvLeague-Basic:**      ↓2.7%      ↓5.5%      ↓17.4%  
 Compared to *baseline*              Small              Medium              Large

**Performance of IvLeague-Invert/IvLeague-Pro:**      ↑8.2%/↑13.5%      ↑3.4%/↑9.3%      ↓13.2%/↑3.4%  
 Compared to *baseline*              Small              Medium              Large



Side channel-resistant integrity mechanisms can have **better performance** than the **baseline insecure scheme** with global integrity tree!

# Takeaways and Conclusions

---

- uArch attacks are becoming ubiquitous
  - “*The new buffer overflow*”
- uArch security *cannot* be considered as a **standalone problem!**
  - Look at uarch security from a broader perspective
- The need to understand composability of security mechanisms
  - Would a defense for one threat bring a bigger issue for another?
- Performance and security can co-exist if done well
- Lots of things to explore for cross-threat model uArch security research!

IvLeague: Side Channel-resistant Secure Architectures Using Isolated Domains of Dynamic Integrity Trees Md Hafizul Islam Chowdhury and Fan Yao, MICRO 2024

MICRO paper presentation on **Tuesday Session 8A**. Welcome to attend!

# Thanks! Questions?

Fan Yao, Email: [fan.yao@ucf.edu](mailto:fan.yao@ucf.edu)  
UCF CASR Lab (<https://casr.ece.ucf.edu>)