

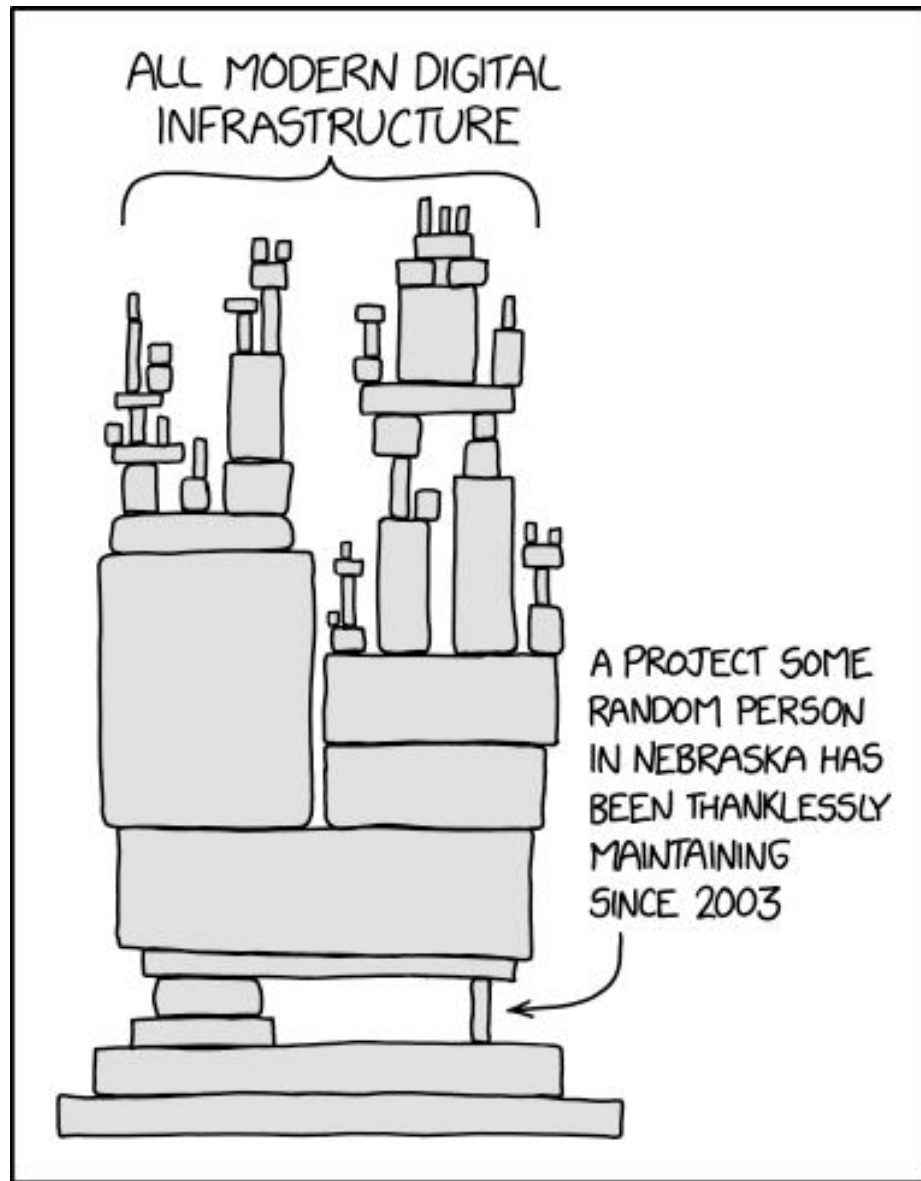
Cracks in the Shield: Understanding and Mitigating Side-Channel Threats to Confidential Computing

Jo Van Bulck

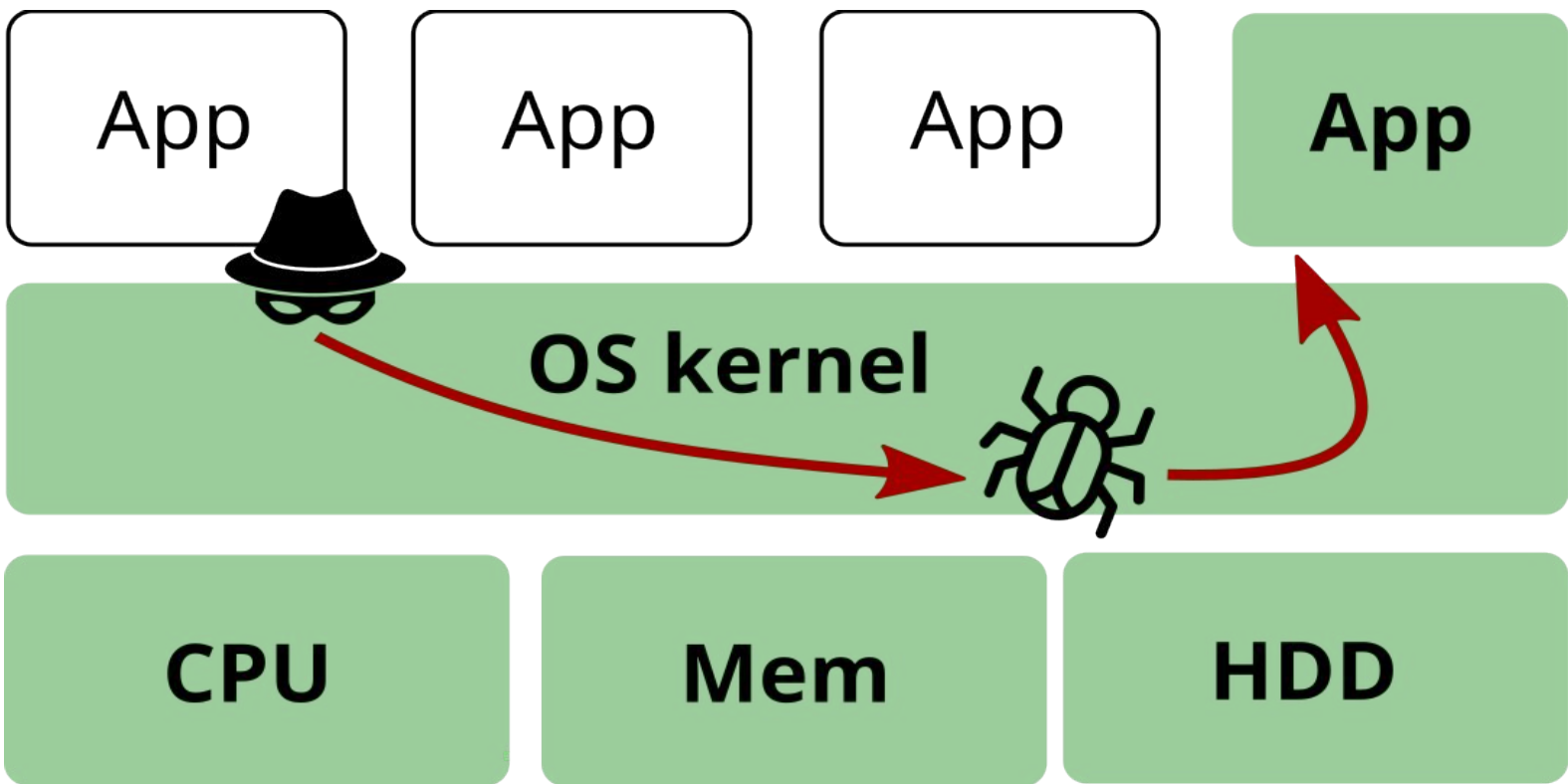
🏠 DistriNet, KU Leuven, Belgium ✉️ jo.vanbulck@cs.kuleuven.be 🌐 vanbulck.net

Graz Security Week, Sept 4, 2025

Trust?

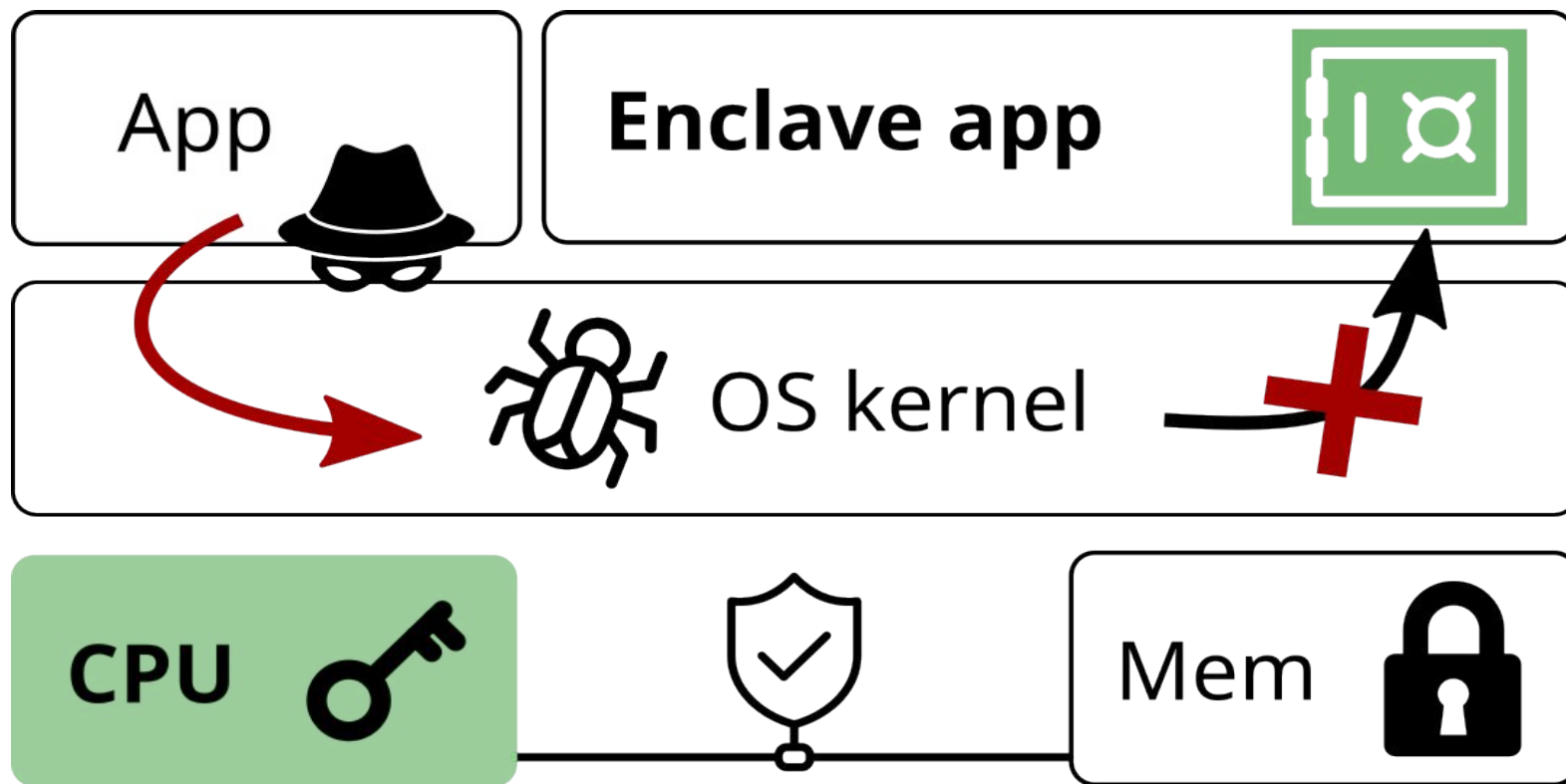


Confidential Computing: Reducing Attack Surface



Traditional layered designs: Large **trusted computing base**

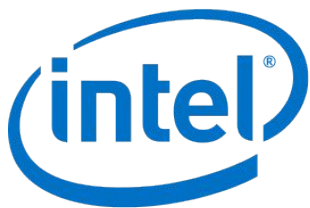
Confidential Computing: Reducing Attack Surface



Trusted execution: Hardware-level **isolation and attestation**

The Rise of Trusted Execution Environments (TEEs)

arm



AMD

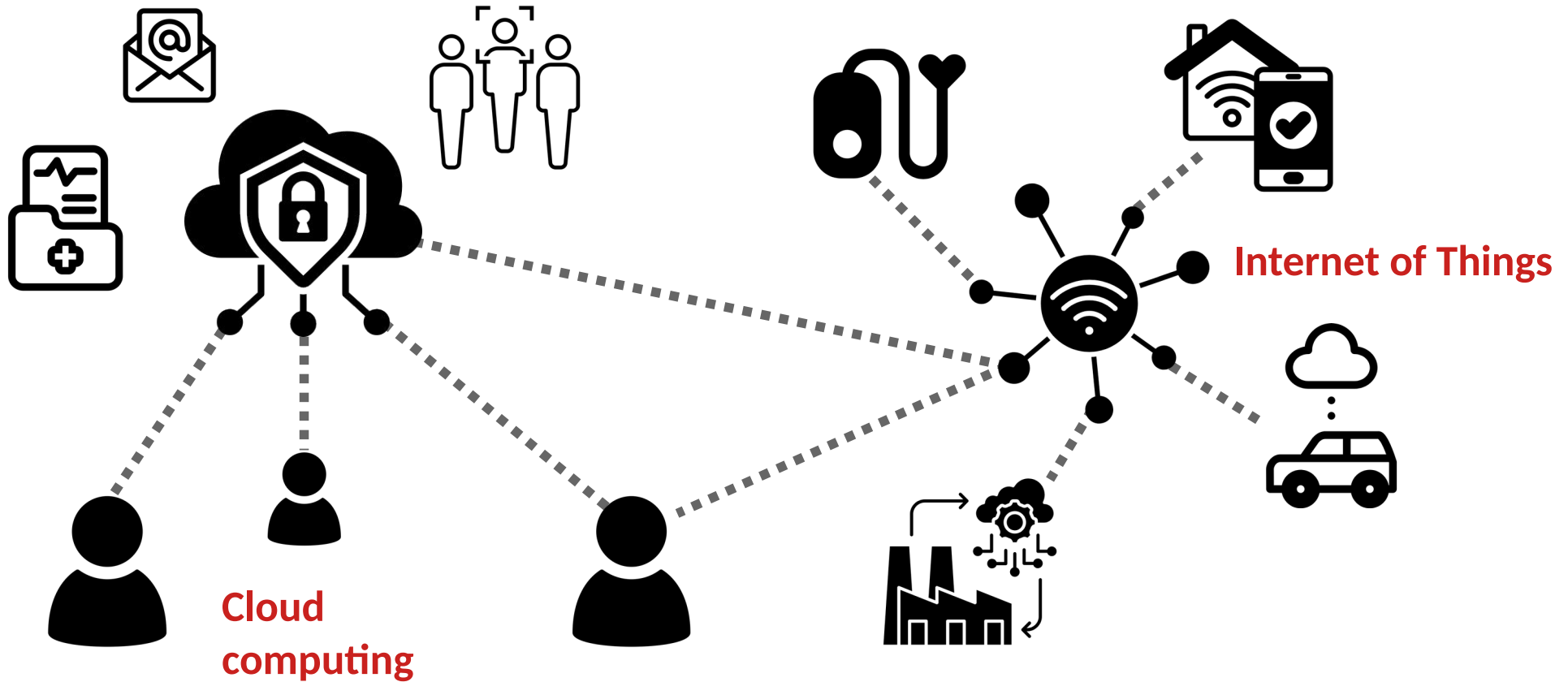


- 2004: ARM TrustZone
- 2015: **Intel Software Guard Extensions (SGX)**
- 2016: AMD Secure Encrypted Virtualization (SEV)
- 2018: IBM Protected Execution Facility (PEF)
- 2020: AMD SEV with Secure Nested Paging (SEV-SNP)
- 2022: Intel Trust Domain Extensions (TDX)
- 2023: ARM Confidential Compute Architecture (CCA)
- 2024: NVIDIA Confidential Computing



TEEs are here to stay...

“Confidential Computing Today, Just Computing Tomorrow” *



TEE Attack Research Leads the Way . . .



TEE Attack Research Leads the Way . . .



- Privileged TEE attacker models **sets the bar!**
- **Idealized execution environment** for attack research
- **Generalizations:** e.g., Foreshadow-NG, branch prediction, address translation, etc.



Motivation: Why Research TEE/SGX Security?



[Overview](#) [About Intel](#) [News & Events](#) [Financial Info](#) [Stock Info](#) [Filings & Reports](#) [Board & Governance](#) [ESG](#)

[Overview](#)

[Press Releases](#)

[IR Calendar](#)

[Annual Stockholders' Meeting](#)

[Investor Meeting](#)

[Email Alerts](#)

[Presentations](#)

Data Protection across the Compute Stack

Technologies such as disk- and network-traffic encryption protect data in storage and during transmission, but data can be vulnerable to interception and tampering while in use in memory. “Confidential computing” is a rapidly emerging usage category that protects data while it is in use in a Trusted Execution Environment (TEE). Intel SGX is the most researched, updated and battle-tested TEE for data center confidential computing, with the smallest attack surface within the system. It enables application isolation in private memory regions, called enclaves, to help protect up to 1 terabyte of code and data while in use.

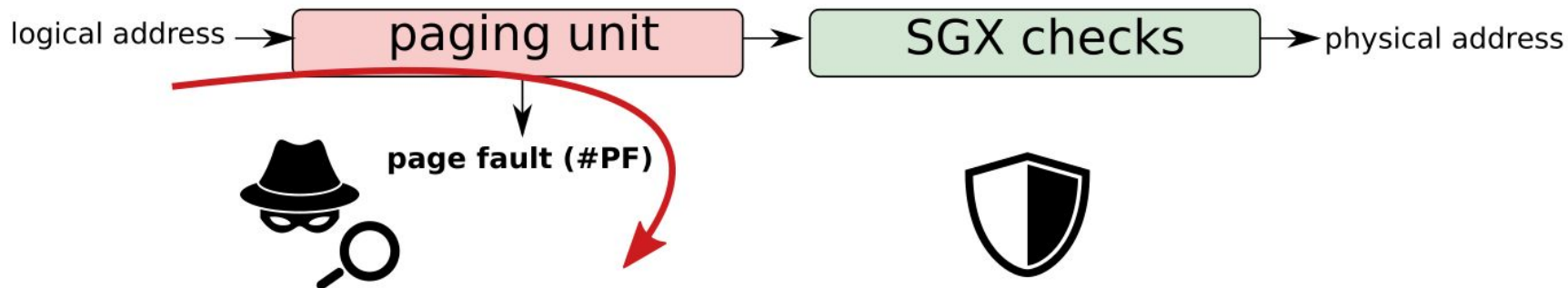


Idea: Page Faults as a Side Channel



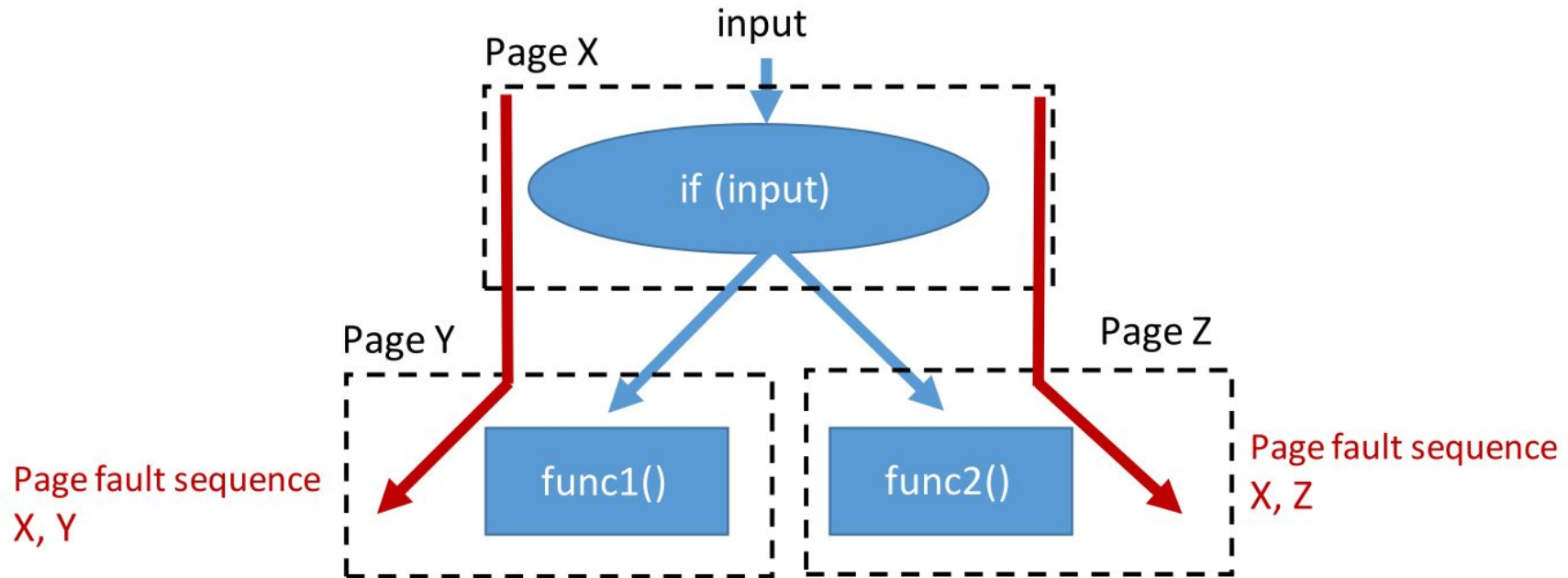
SGX machinery protects against direct address remapping attacks

Idea: Page Faults as a Side Channel



... but untrusted address translation may **fault(!)**

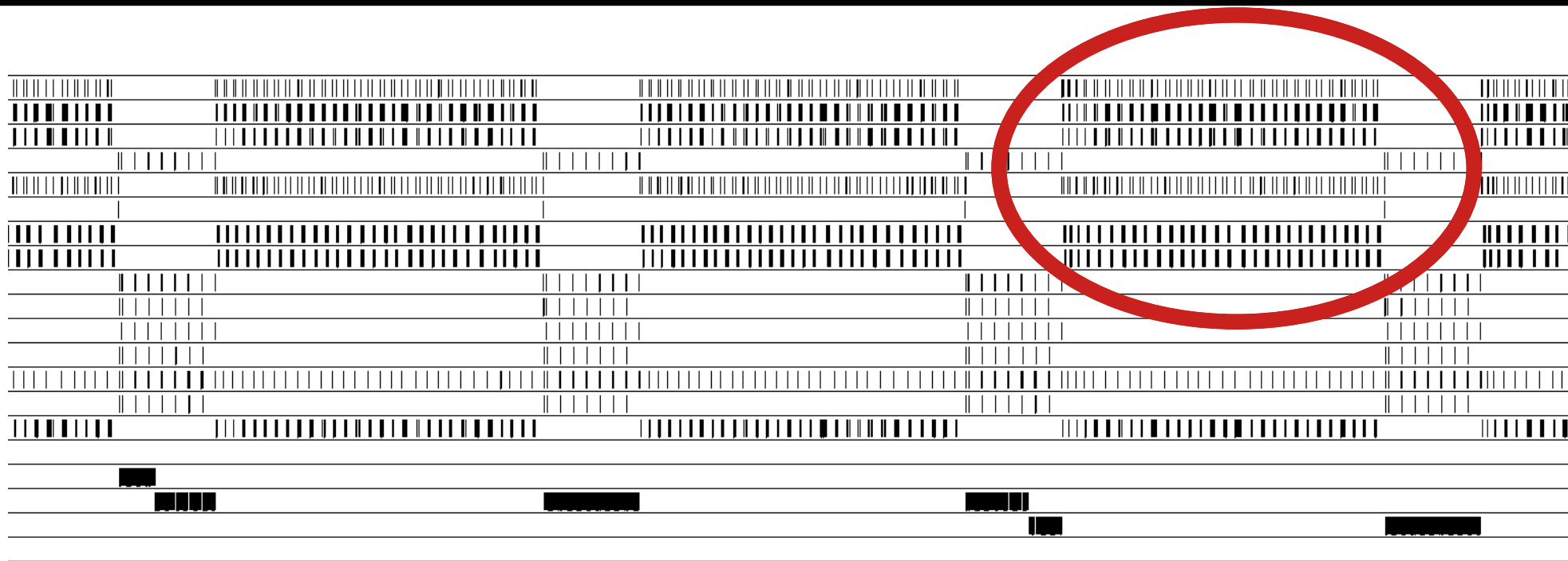
Intel SGX: Page Faults as a Side Channel



□ Xu et al.: “Controlled-channel attacks: Deterministic side channels for untrusted operating systems”, Oakland 2015.

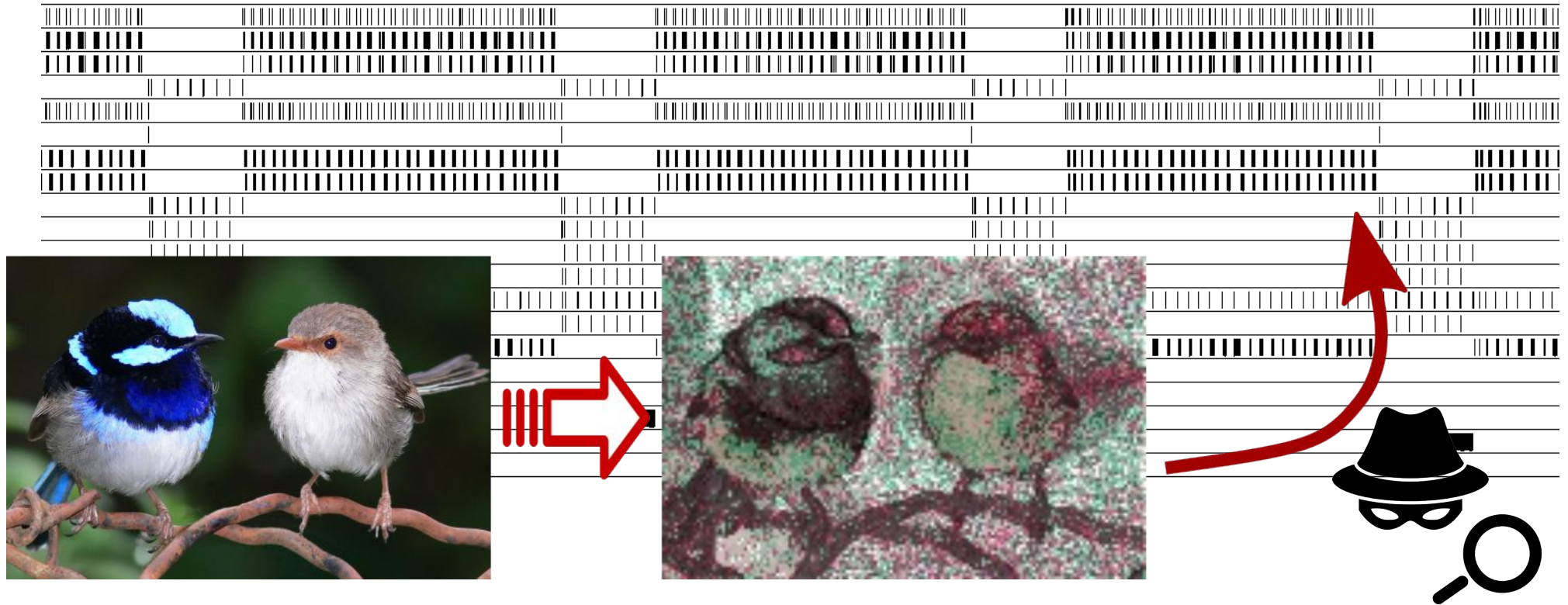
⇒ Page fault traces leak **private control data/flow**

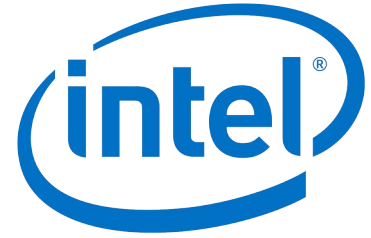
Spatial Resolution: Page-Granular Memory Access Traces



Detailed trace of (coarse-grained) **code and data accesses over time...**

Spatial Resolution: Page-Granular Memory Access Traces





Protection from Side-Channel Attacks

Intel® SGX does not provide explicit protection from side-channel attacks. It is the enclave developer's responsibility to address side-channel attack concerns.

In general, enclave operations that require an OCall, such as thread synchronization, I/O, etc., are exposed to the untrusted domain. If using an OCall would allow an attacker to gain insight into enclave secrets, then there would be a security concern. This scenario would be classified as a side-channel attack, and it would be up to the ISV to design the enclave in a way that prevents the leaking of side-channel information.

An attacker with access to the platform can see what pages are being executed or accessed. This side-channel vulnerability can be mitigated by aligning specific code and data blocks to exist entirely within a single page.

More important, the application enclave should use an appropriate crypto implementation that is side channel attack resistant inside the enclave if side-channel attacks are a concern.

Temporal Resolution Limitations for the Page-Fault Oracle

```
1  size_t  strlen (char *str)
2  {
3      char *s;
4
5      for (s = str; *s; ++s);
6      return (s - str);
7  }
```

```
1      mov    %rdi,%rax
2  1:  cmpb    $0x0,(%rax)
3      je     2f
4      inc    %rax
5      jmp    1b
6  2:  sub     %rdi,%rax
7      retq
```

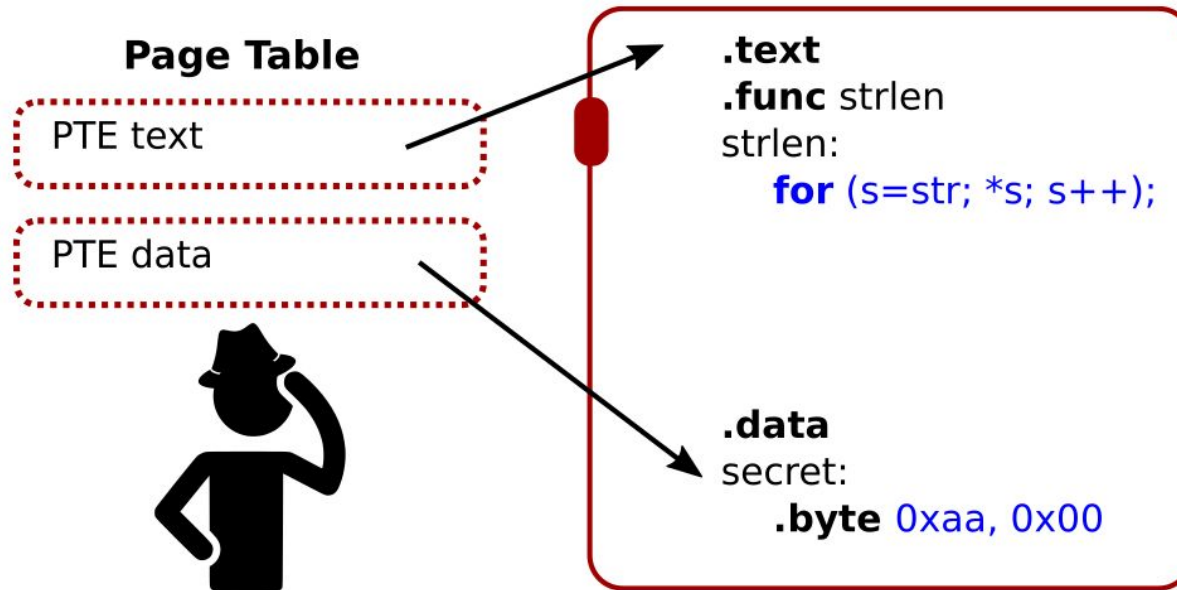
⇒ tight loop: 4 instructions, single memory operand, single code + data page

Counting strlen loop iterations?



Note: Page-fault attacks cannot make progress for 1 code + data page

Temporal Resolution Limitations for the Page-Fault Oracle



Counting strlen loop iterations?

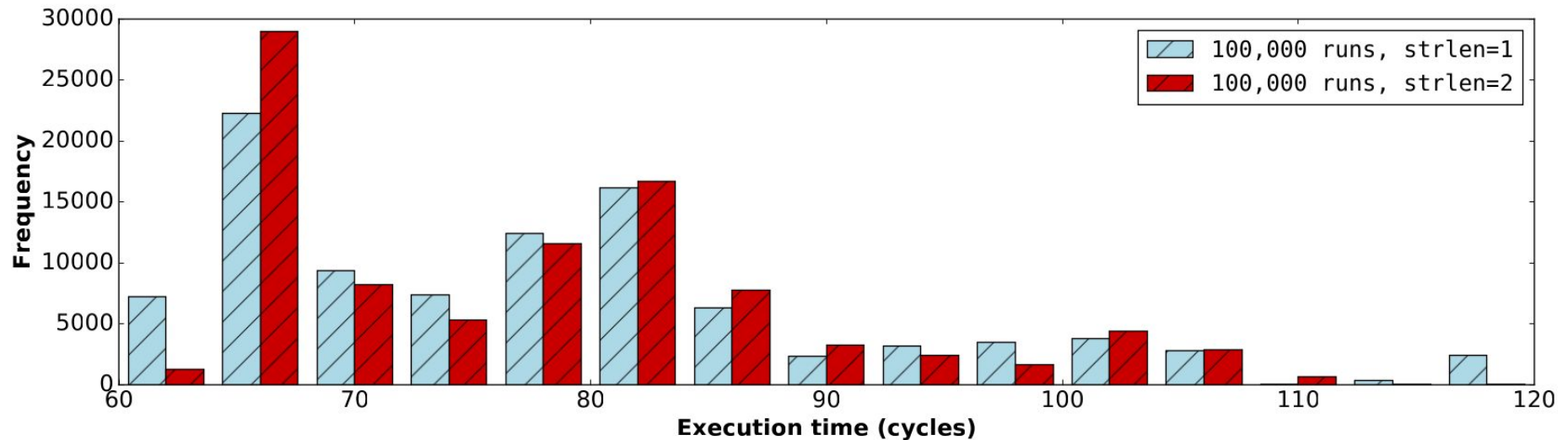


Progress requires **both pages present** (non-faulting) ↔ page fault oracle

Building the Side-Channel Oracle with Execution Timing?



Too noisy: modern x86 processors are lightning fast. . .



Challenge: Side-Channel Sampling Rate



**Slow
shutter speed**

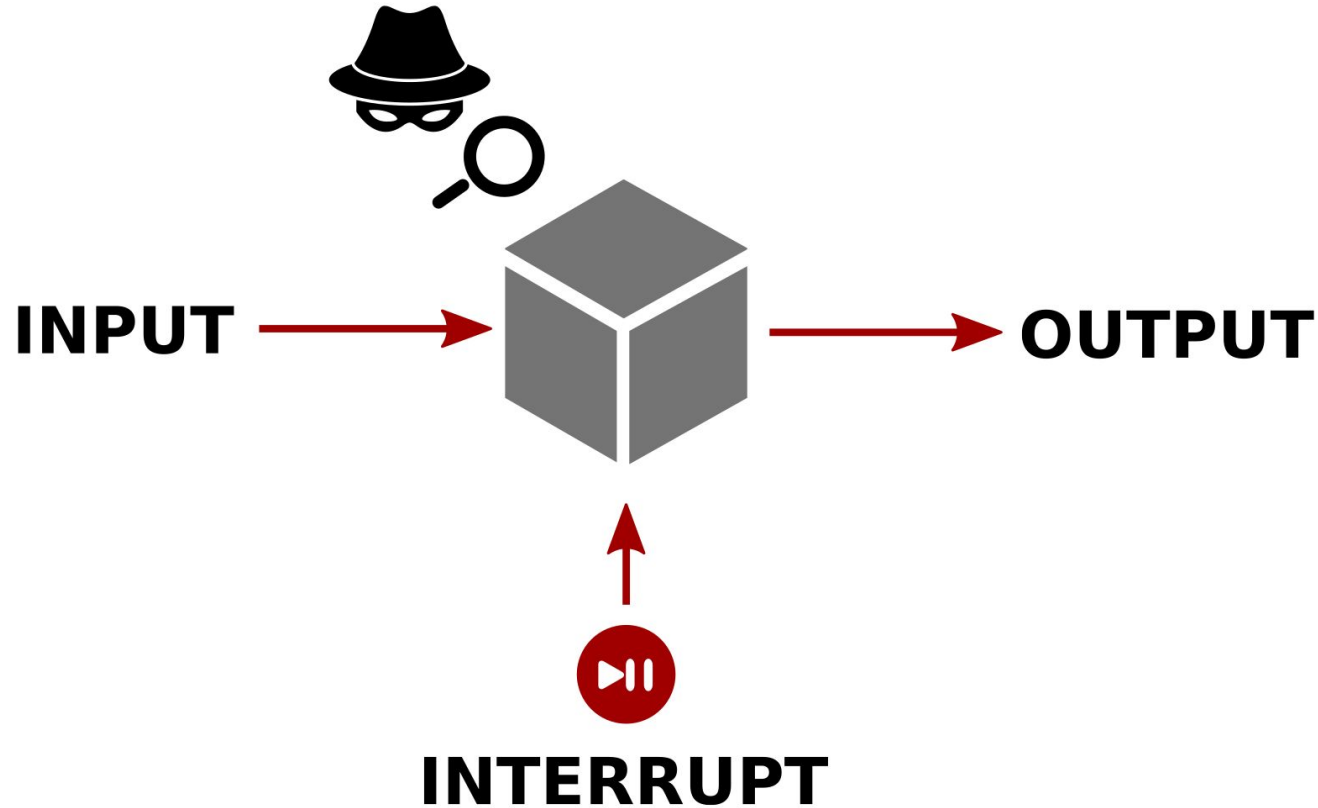


**Medium
shutter speed**

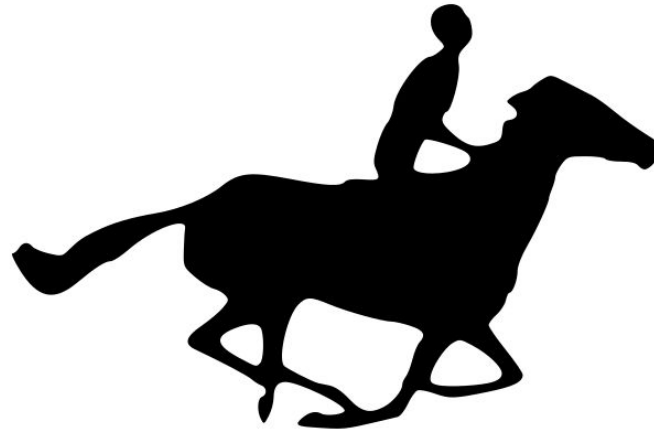


**Fast
shutter speed**

SGX-Step: Executing Enclaves one Instruction at a Time



SGX-Step: Executing Enclaves one Instruction at a Time



SGX-Step



**ACSAC 2023
Cybersecurity Artifacts
Impact Award**

 <https://github.com/jovanbulck/sgx-step>

 Watch

22

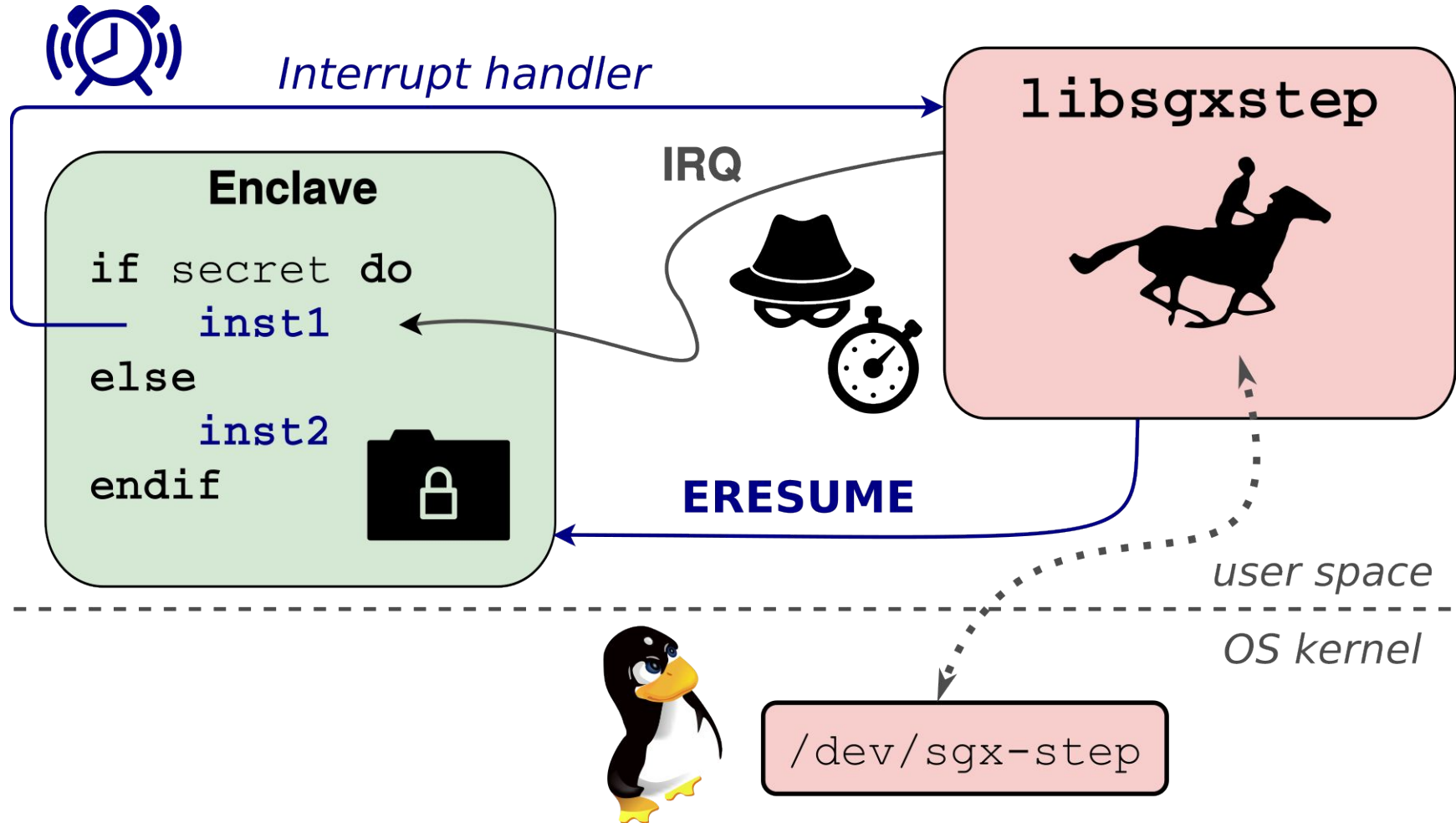
 Star

245

 Fork

52



























SGX-Step: Executing Enclaves one Instruction at a Time




























SGX-Step Demo: Single-Stepping Password Comparison

```
jo@breuer:~/sgx-step-demo$ sudo ./app █
```


SGX-Step: Enabling a New Line of High-Resolution Attacks

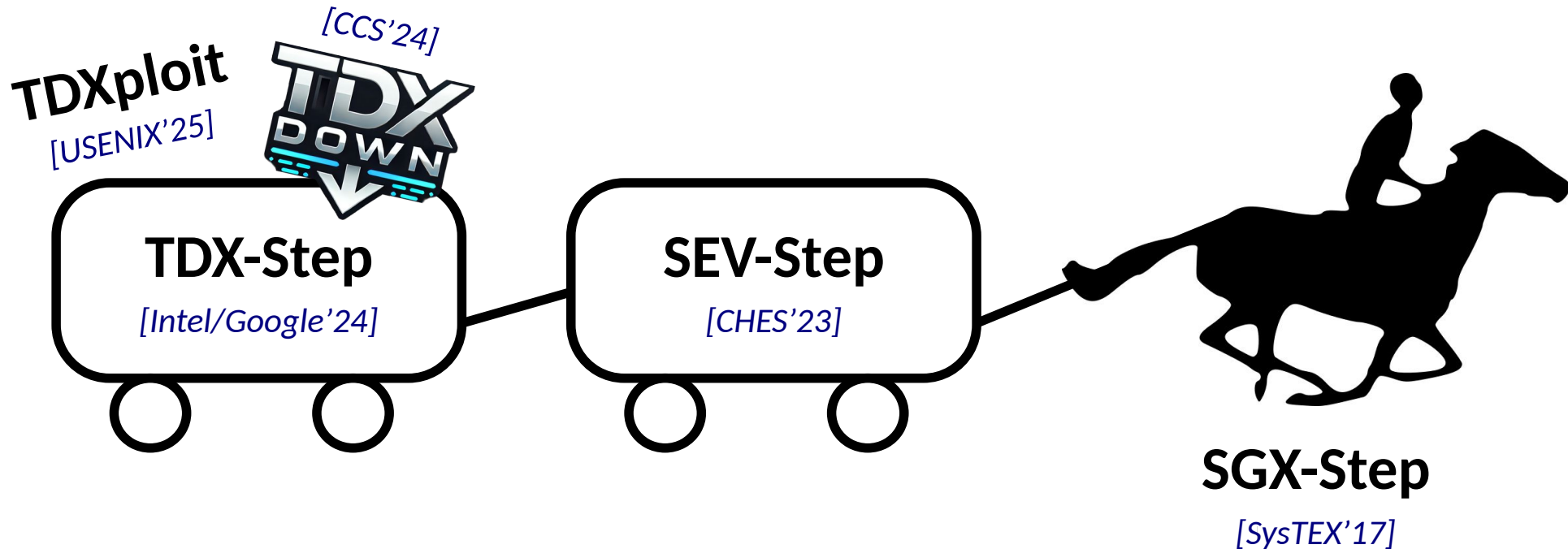
| Yr | Venue | Paper | Step | Use Case | Drv |
|-----|---------|------------------|------------|------------------------|---|
| '15 | S&P | Ctrl channel | ~ Page | Probe (page fault) | ✓  |
| '16 | ESORICS | AsyncShock | ~ Page | Exploit (mem safety) | –  |
| '17 | CHES | CacheZoom | ✗ >1 | Probe (L1 cache) | ✓  |
| '17 | ATC | Hahnel et al. | ✗ 0 - >1 | Probe (L1 cache) | ✓  |
| '17 | USENIX | BranchShadow | ✗ 5 - 50 | Probe (BPU) | ✗  |
| '17 | USENIX | Stealthy PTE | ~ Page | Probe (page table) | ✓  |
| '17 | USENIX | DarkROP | ~ Page | Exploit (mem safety) | ✓  |
| '17 | SysTEX | SGX-Step | ✓ 0 - 1 | Framework | ✓  |
| '18 | ESSoS | Off-limits | ✓ 0 - 1 | Probe (segmentation) | ✓  |
| '18 | AsiaCCS | Single-trace RSA | ~ Page | Probe (page fault) | ✓  |
| '18 | USENIX | Foreshadow | ✓ 0 - 1 | Probe (transient exec) | ✓  |
| '18 | EuroS&P | SgxPectre | ~ Page | Exploit (transient) | ✓  |
| '18 | CHES | CacheQuote | ✗ >1 | Probe (L1 cache) | ✓  |
| '18 | ICCD | SGXlinger | ✗ >1 | Probe (IRQ latency) | ✗  |
| '18 | CCS | Nemesis | ✓ 1 | Probe (IRQ latency) | ✓  |
| '19 | USENIX | Spoiler | ✓ 1 | Probe (IRQ latency) | ✓  |
| '19 | CCS | ZombieLoad | ✓ 0 - 1 | Probe (transient exec) | ✓  |
| '19 | CCS | Fallout | – | Probe (transient exec) | ✓  |
| '19 | CCS | Tale of 2 worlds | ✓ 1 | Exploit (mem safety) | ✓  |
| '19 | ISCA | MicroScope | ~ 0 - Page | Framework | ✗  |
| '20 | CHES | Bluethunder | ✓ 1 | Probe (BPU) | ✓  |
| '20 | USENIX | Big troubles | ~ Page | Probe (page fault) | ✓  |
| '20 | S&P | Plundervolt | – | Exploit (undervolt) | ✓  |
| '20 | CHES | Viral primitive | ✓ 1 | Probe (IRQ count) | ✓  |
| '20 | USENIX | CopyCat | ✓ 1 | Probe (IRQ count) | ✓  |
| '20 | S&P | LVI | ✓ 1 | Exploit (transient) | ✓  |

| Yr | Venue | Paper | Step | Use Case | Drv |
|-----|---------|-------------------|----------|------------------------|---|
| '20 | CHES | A to Z | ~ Page | Probe (page fault) | ✓  |
| '20 | CCS | Déjà Vu NSS | ~ Page | Probe (page fault) | ✓  |
| '20 | MICRO | PTHammer | – | Probe (page walk) | ✓  |
| '21 | USENIX | Frontal | ✓ 1 | Probe (IRQ latency) | ✓  |
| '21 | S&P | CrossTalk | ✓ 1 | Probe (transient exec) | ✓  |
| '21 | CHES | Online template | ✓ 1 | Probe (IRQ count) | ✓  |
| '21 | NDSS | SpeechMiner | – | Framework | ✓  |
| '21 | S&P | Platypus | ✓ 0 - 1 | Probe (voltage) | ✓  |
| '21 | DIMVA | Aion | ✓ 1 | Probe (cache) | ✓  |
| '21 | CCS | SmashEx | ✓ 1 | Exploit (mem safety) | ✓  |
| '21 | CCS | Util::Lookup | ✓ 1 | Probe (L3 cache) | ✓  |
| '22 | USENIX | Rapid prototyping | ✓ 1 | Framework | ✓  |
| '22 | CT-RSA | Kalyana expansion | ✓ 1 | Probe (L3 cache) | ✓  |
| '22 | SEED | Enclyzer | – | Framework | ✓  |
| '22 | NordSec | Self-monitoring | ~ Page | Defense (detect) | ✓  |
| '22 | AutoSec | Robotic vehicles | ✓ 1 - >1 | Exploit (timestamp) | ✓  |
| '22 | ACSAC | MoLE | ✓ 1 | Defense (randomize) | ✓  |
| '22 | USENIX | AEPIC | ✓ 1 | Probe (I/O device) | ✓  |
| '22 | arXiv | Confidential code | ✓ 1 | Probe (IRQ latency) | ✓  |
| '23 | ComSec | FaultMorse | ~ Page | Probe (page fault) | ✓  |
| '23 | CHES | HQC timing | ✓ 1 | Probe (L3 cache) | ✓  |
| '23 | ISCA | Belong to us | ✓ 1 | Probe (BPU) | ✓  |
| '23 | USENIX | BunnyHop | ✓ 1 | Probe (BPU) | ✓  |
| '23 | USENIX | DownFall | ✓ 0 - 1 | Probe (transient exec) | ✓  |
| '23 | USENIX | AEX-Notify | ✓ 1 | Defense (prefetch) | ✓  |

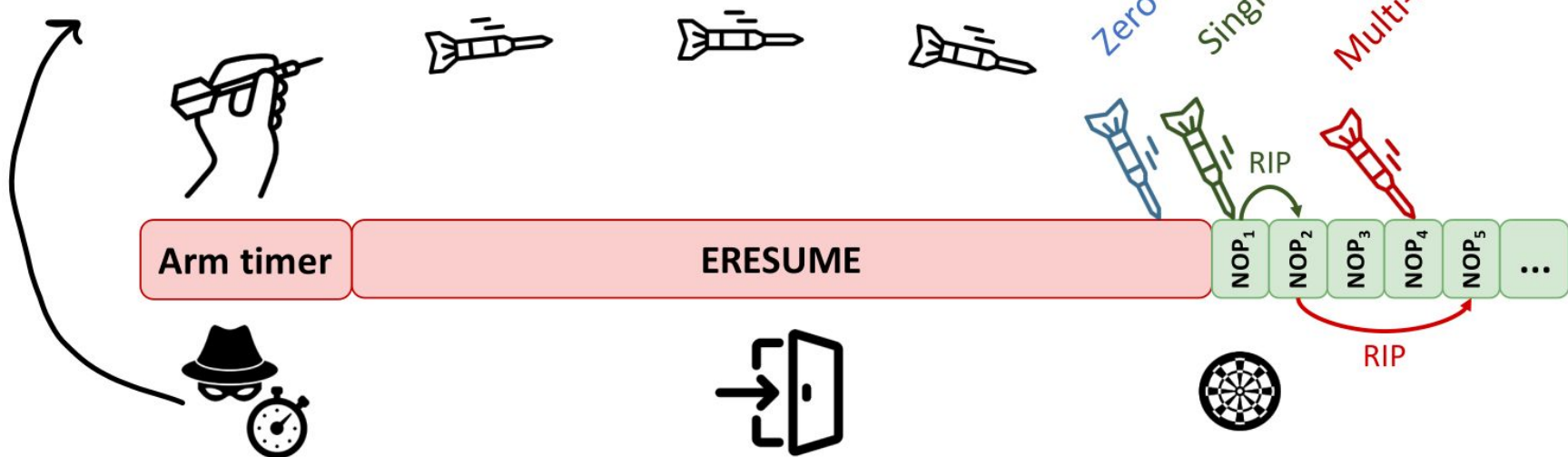
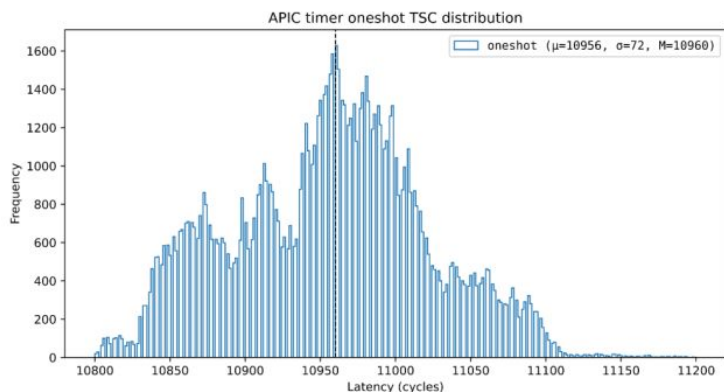
SGX-Step: A Versatile Open-Source Attack Framework



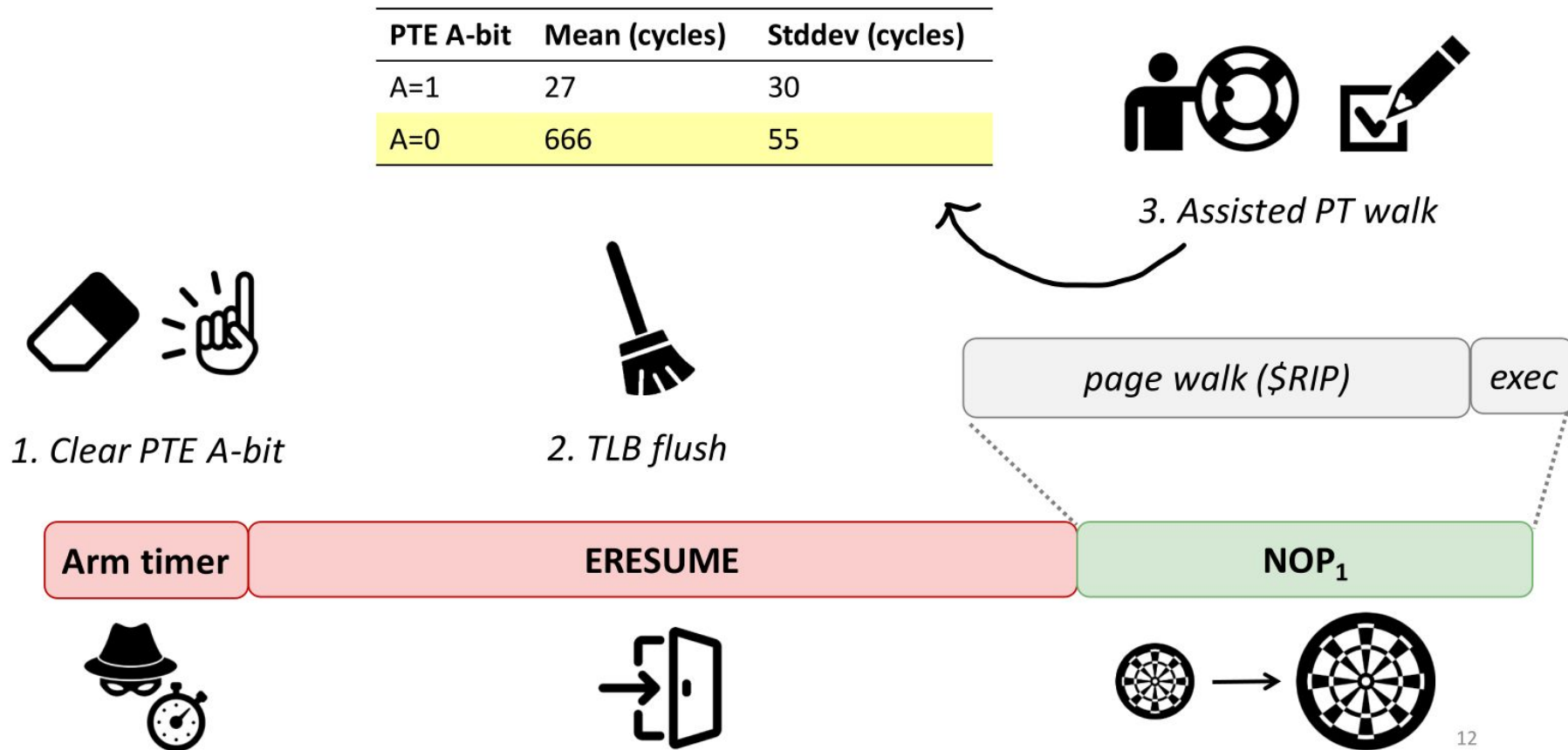
Single-Stepping Beyond Intel SGX



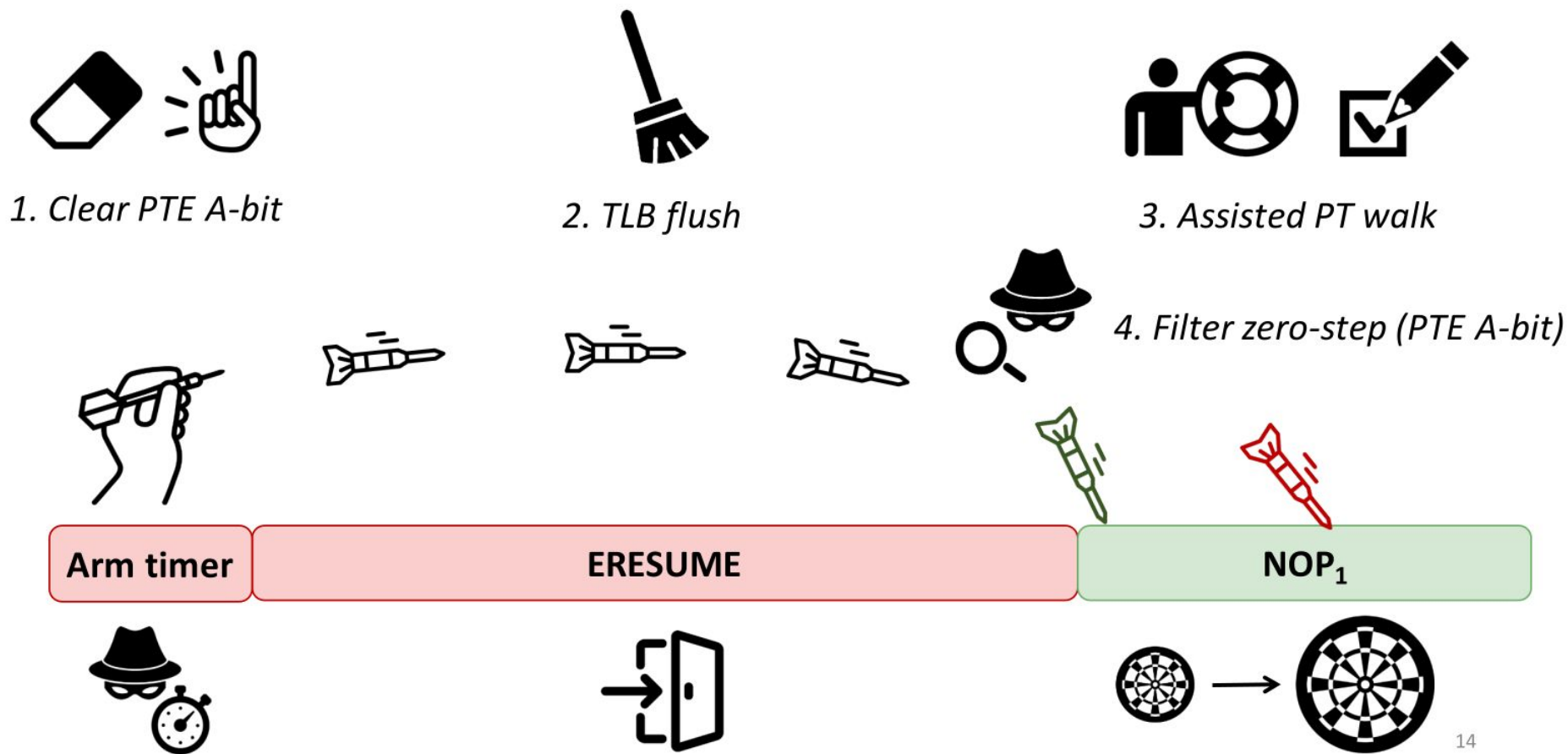
Root-causing SGX-Step: Aiming the timer interrupt



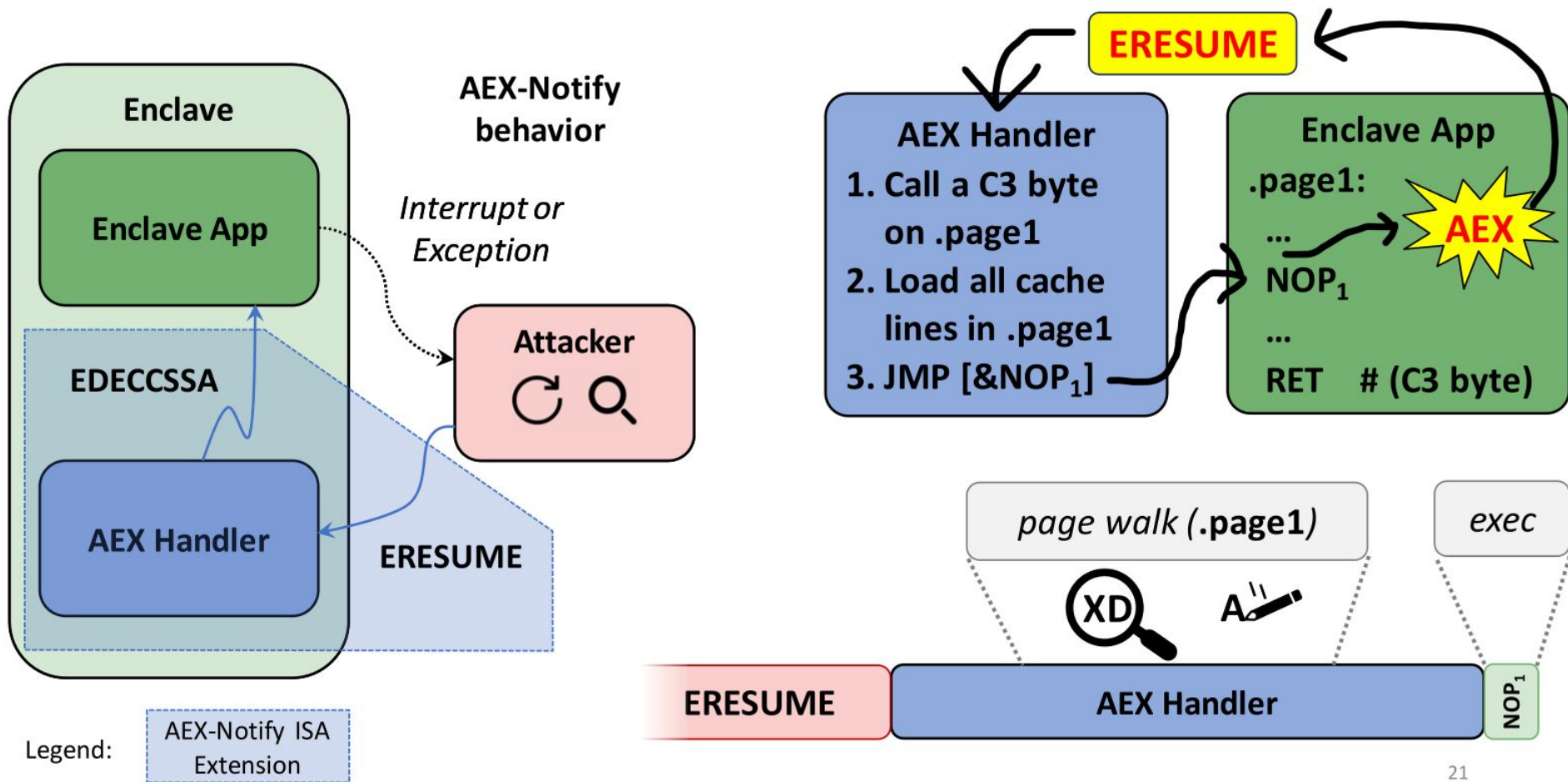
Root-causing SGX-Step: Microcode assists to the rescue!



Root-causing SGX-Step: Microcode assists to the rescue!

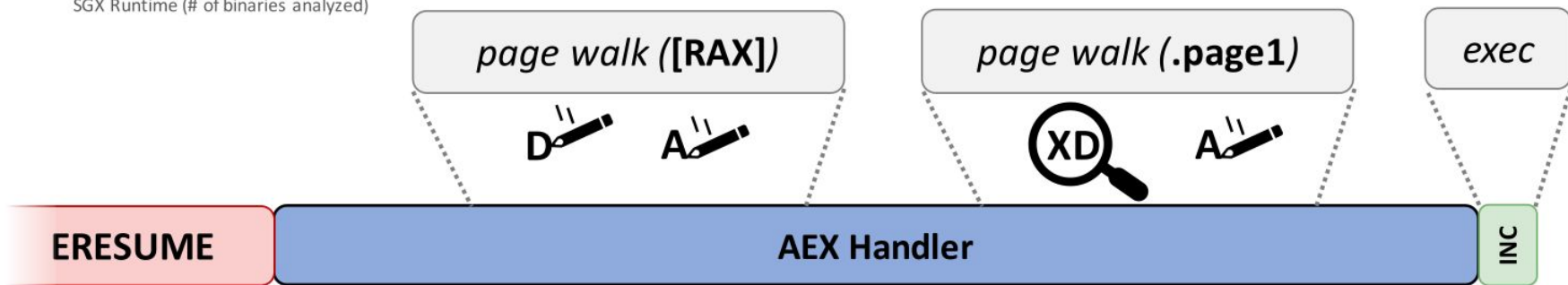
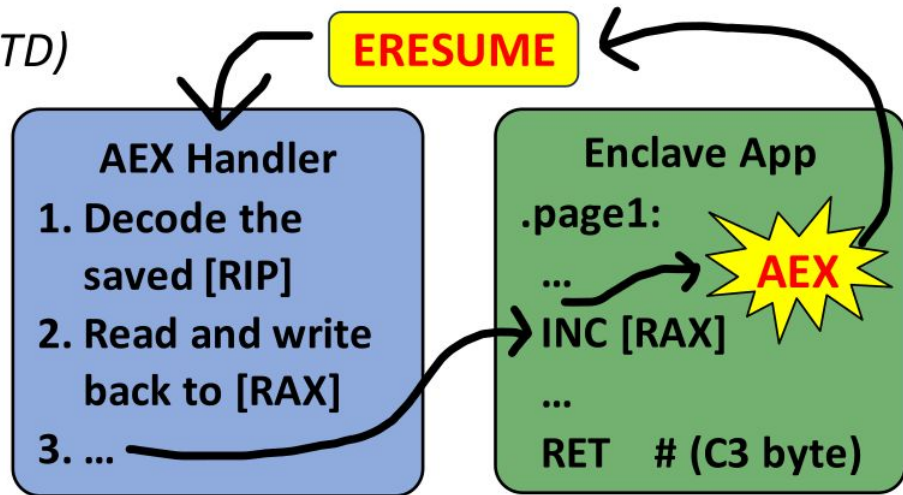
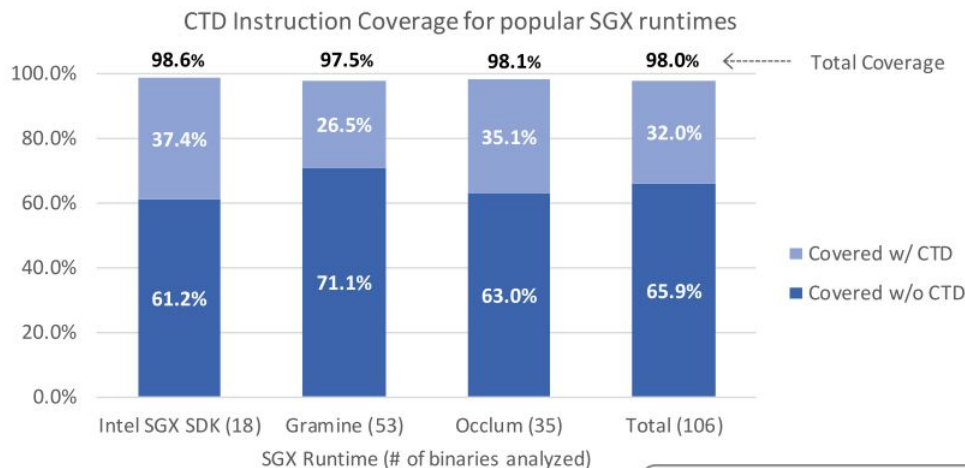


AEX-Notify: Hardware-Software Co-Design Solution



AEX-Notify: Hardware-Software Co-Design Solution

We implemented a fast, constant-time decoder (CTD)



CHAPTER 8

ASYNCHRONOUS ENCLAVE EXIT NOTIFY AND THE EDECCSSA USER LEAF FUNCTION

8.1 INTRODUCTION

Asynchronous Enclave Exit Notify (AEX-Notify) is an extension to Intel® SGX that allows Intel SGX enclaves to be notified after an asynchronous enclave exit (AEX) has occurred. EDECCSSA is a new Intel SGX user leaf function (ENCLU[EDECCSSA]) that can facilitate AEX notification handling, as well as software exception handling. This chapter provides information about changes to the Intel SGX architecture that support AEX-Notify and ENCLU[EDECCSSA].

The following list summarizes the a details are provided in Section 8.3)

- SECS.ATTRIBUTES.AEXNOTIFY:
- TCS.FLAGS.AEXNOTIFY: This e
- SSA.GPRSGX.AEXNOTIFY: Enclave-writable byte that allows enclave software to dynamically enable/disable AEX notifications.

An AEX notification is delivered by ENCLU[ERESUME] when the following conditions are met:



*SGX-Step led to **new x86 processor instructions!***

→ shipped in millions of devices ≥ 4th Gen Xeon CPU

Intel AEX Notify Support Prepped For Linux To Help Enhance SGX Enclave Security

Written by [Michael Larabel](#) in [Intel](#) on 6 November 2022 at 06:01 AM EST. [5 Comments](#)



Future Intel CPUs and some existing processors via a microcode update will support a new feature called the Asynchronous EXit (AEX) notification mechanism to help with Software Guard Extensions (SGX) enclave security. Patches for the Linux kernel are pending for implementing this Intel AEX Notify support with capable processors.

Intel's Asynchronous EXit (AEX) notification mechanism lets SGX enclaves run a handler after an AEX event. Those handlers can be used for things like mitigating SGX-Step as an attack framework for precise enclave execution control.



Code 1 in intel/linux-sgx X



Filter

intel sdk/trts/linux/trts_mitigation.S

```
48 * Description:
49 *   The file provides mitigations for SGX-Step
50 */
71 * Function:
   constant_time_apply_sgxstep_mitigation_and_continue_execution
72 *   Mitigate SGX-Step and return to the point at which the
   most recent
73 *   interrupt/exception occurred.
```

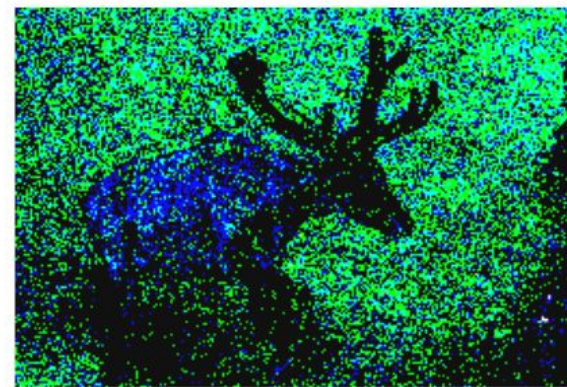
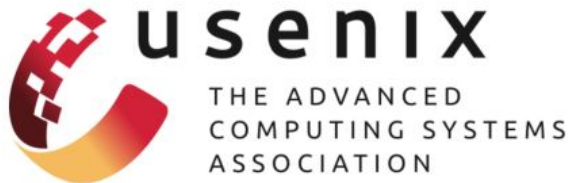


SGX-Step led to **changes in major OSs and enclave SDKs**

There's a Catch...

Finally note that our proposed mitigation does not protect against interrupting enclaves and observing application code and data page accesses at a coarse-grained 4 KiB spatial resolution. In contrast to the fine-grained, instruction-granular interrupt-driven attacks we consider in this work, such controlled-channel attacks have received ample attention [18, 47, 56, 59] from the research community.

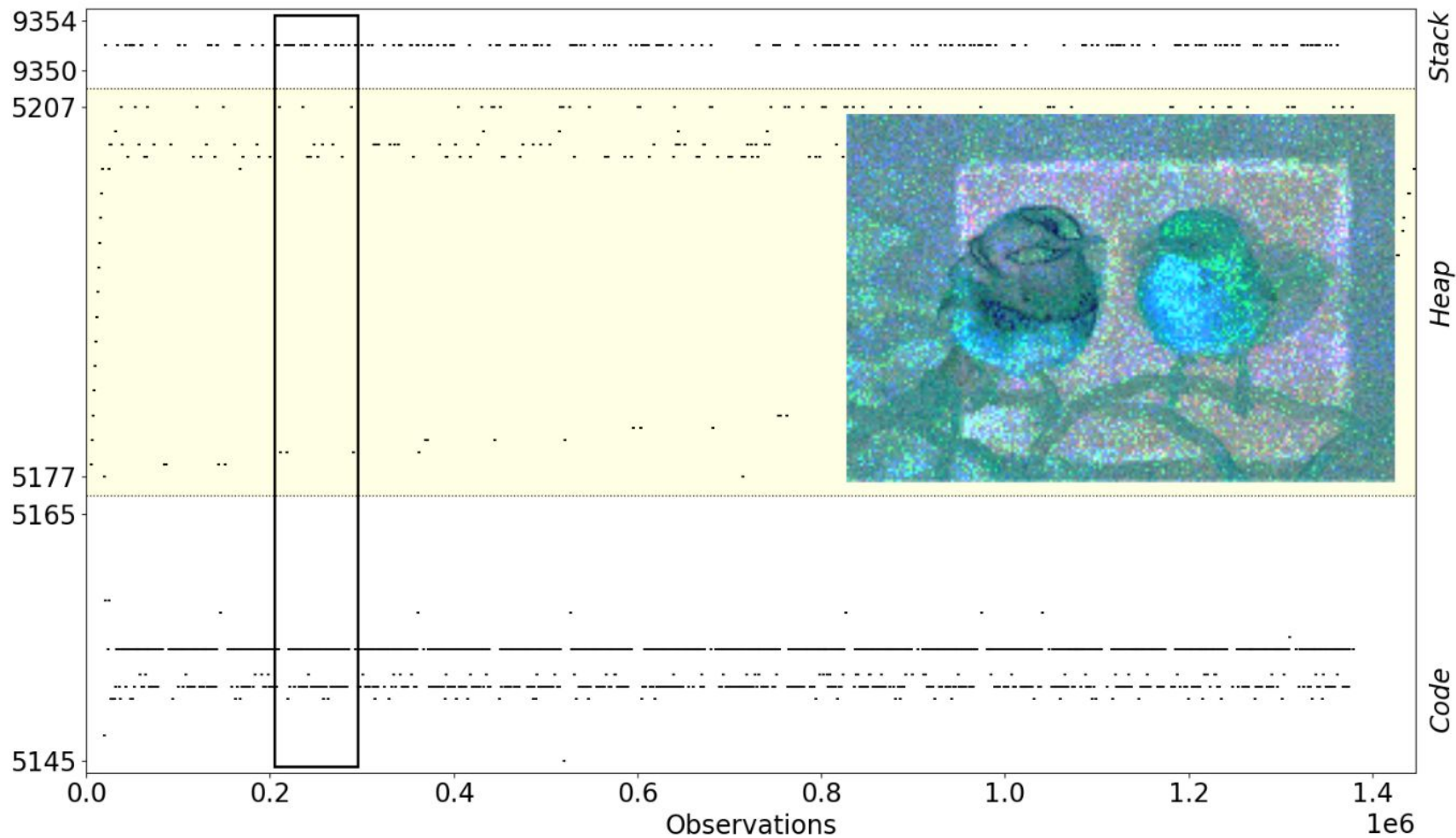
Why Mitigating Single-Stepping is Not Enough



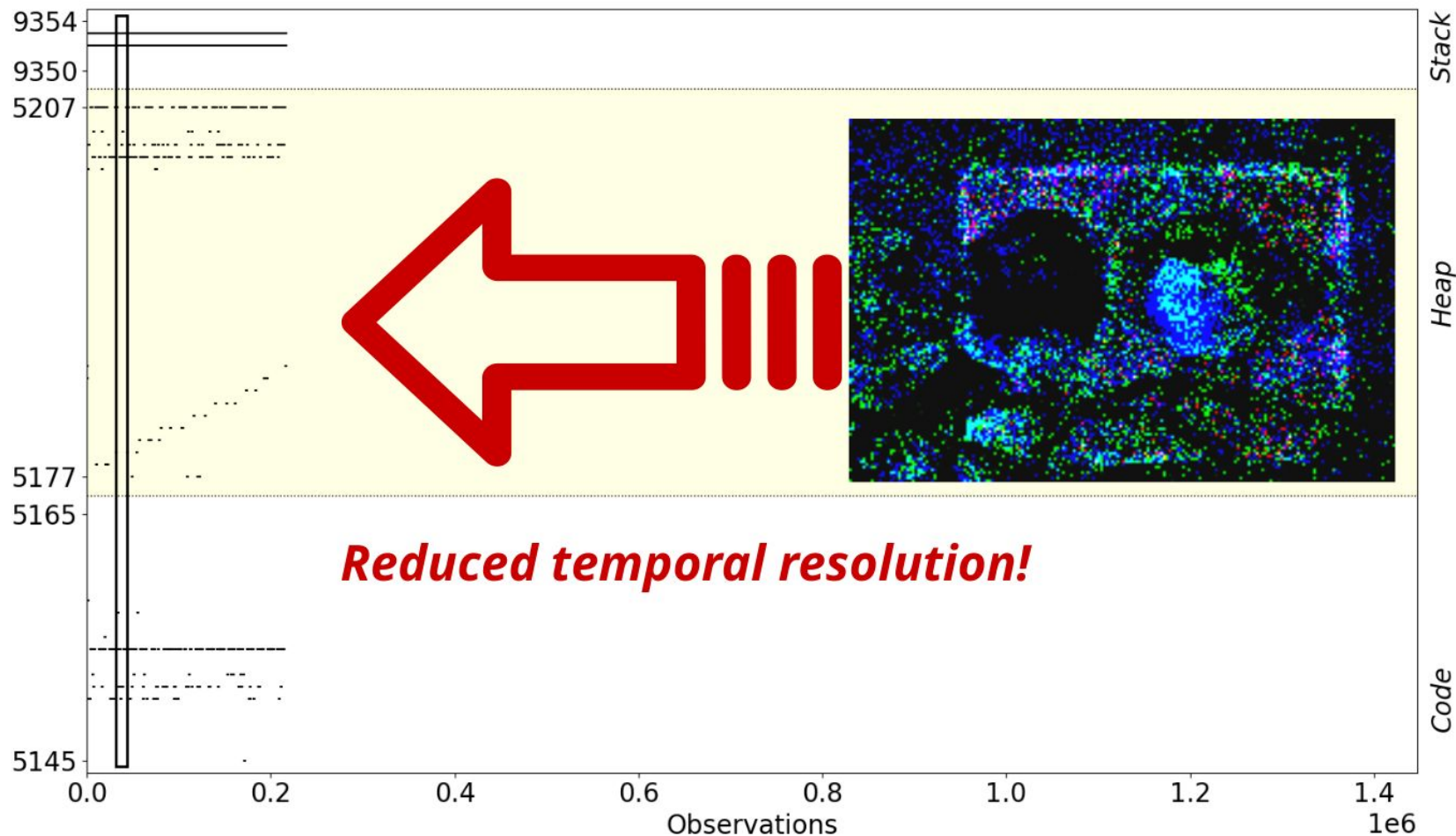
Original (left), Xu et al. (middle), our attack with AEX-Notify single-stepping defense (right)



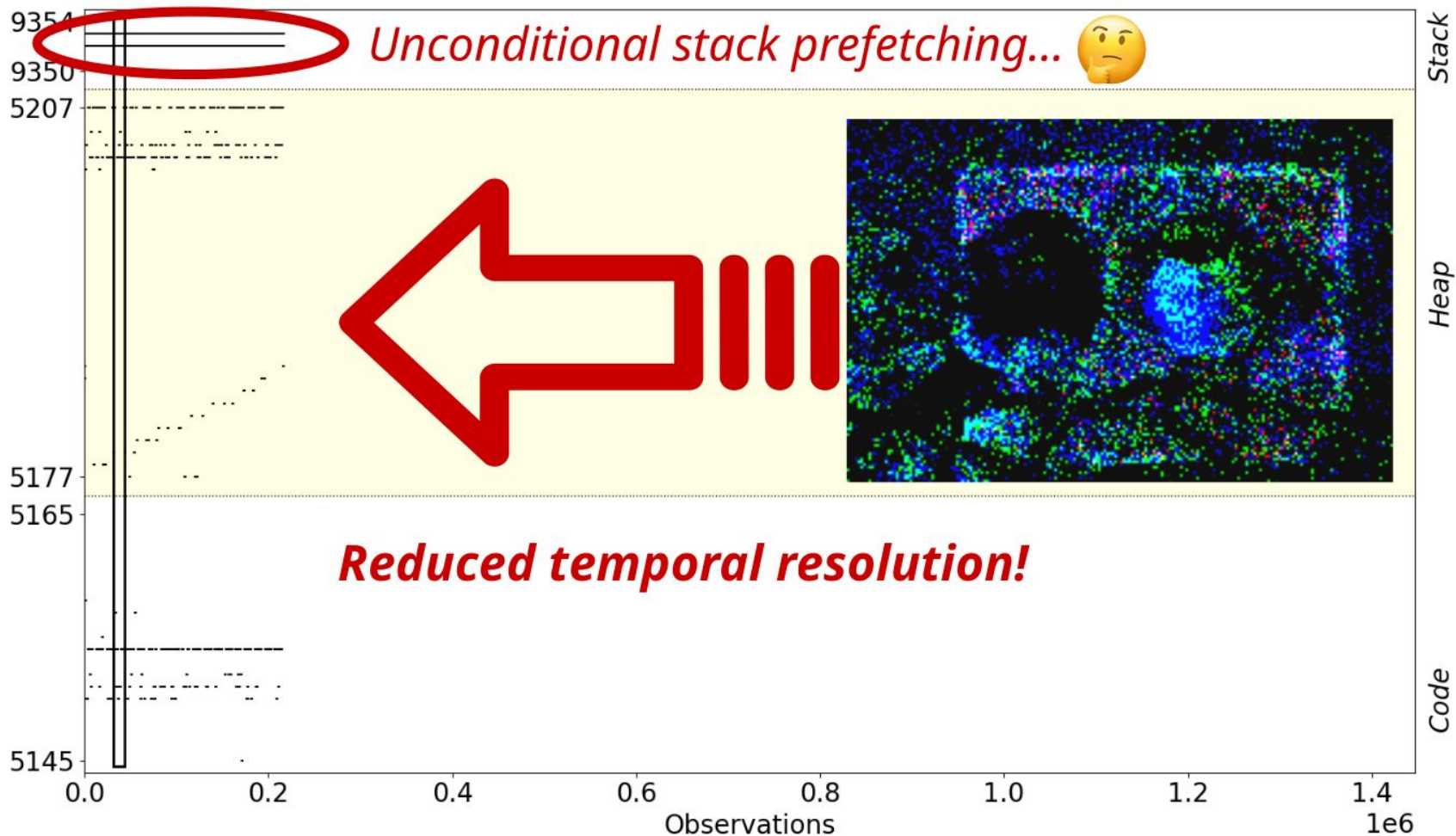
Libjpeg: AEX-Notify's Temporal Reduction in Practice



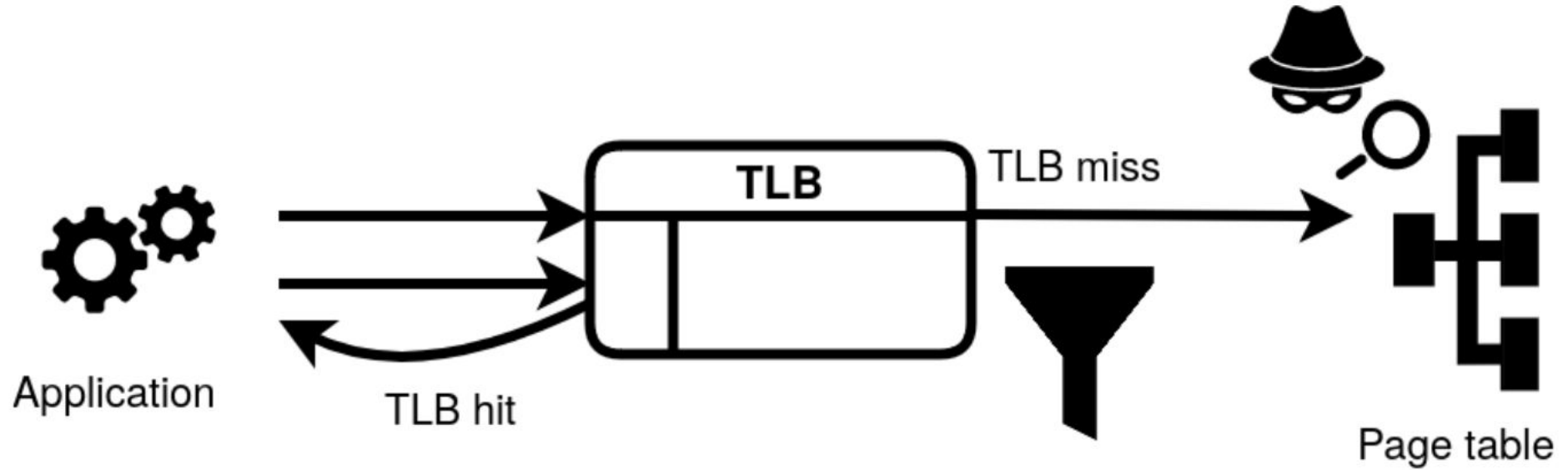
Libjpeg: AEX-Notify's Temporal Reduction in Practice



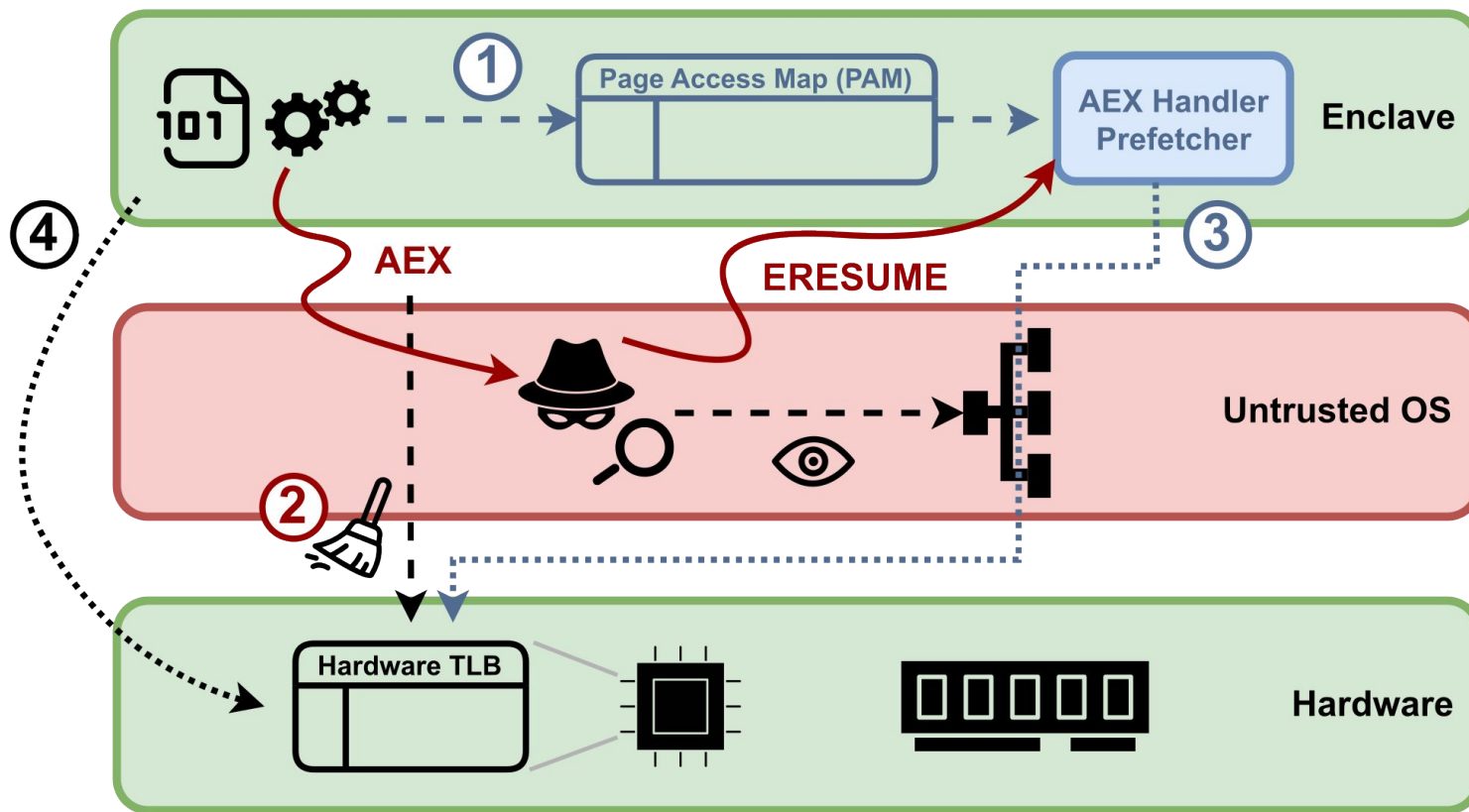
Libjpeg: AEX-Notify's Temporal Reduction in Practice



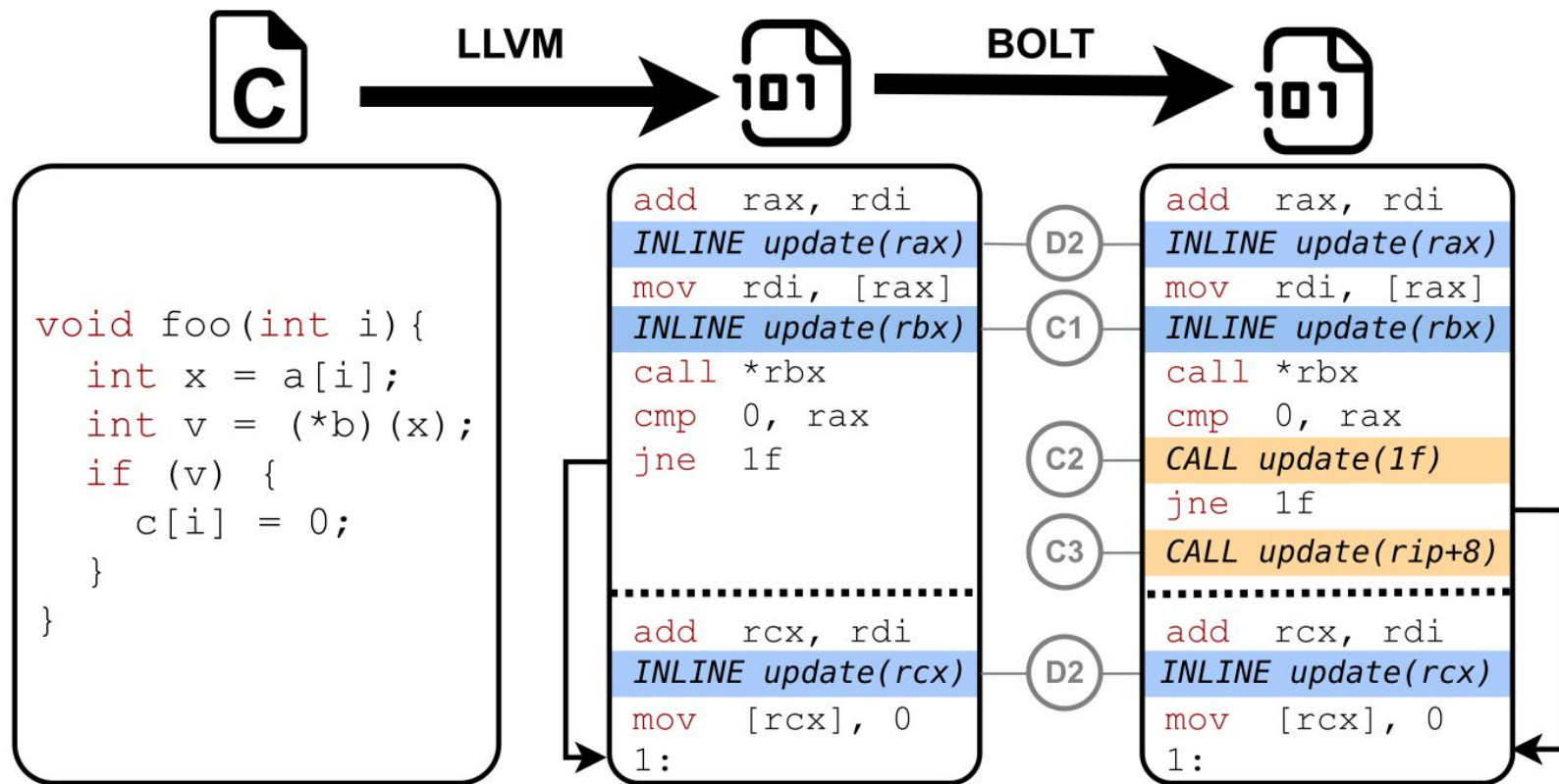
Idea: TLB as a “Filter” to Hide Page Accesses



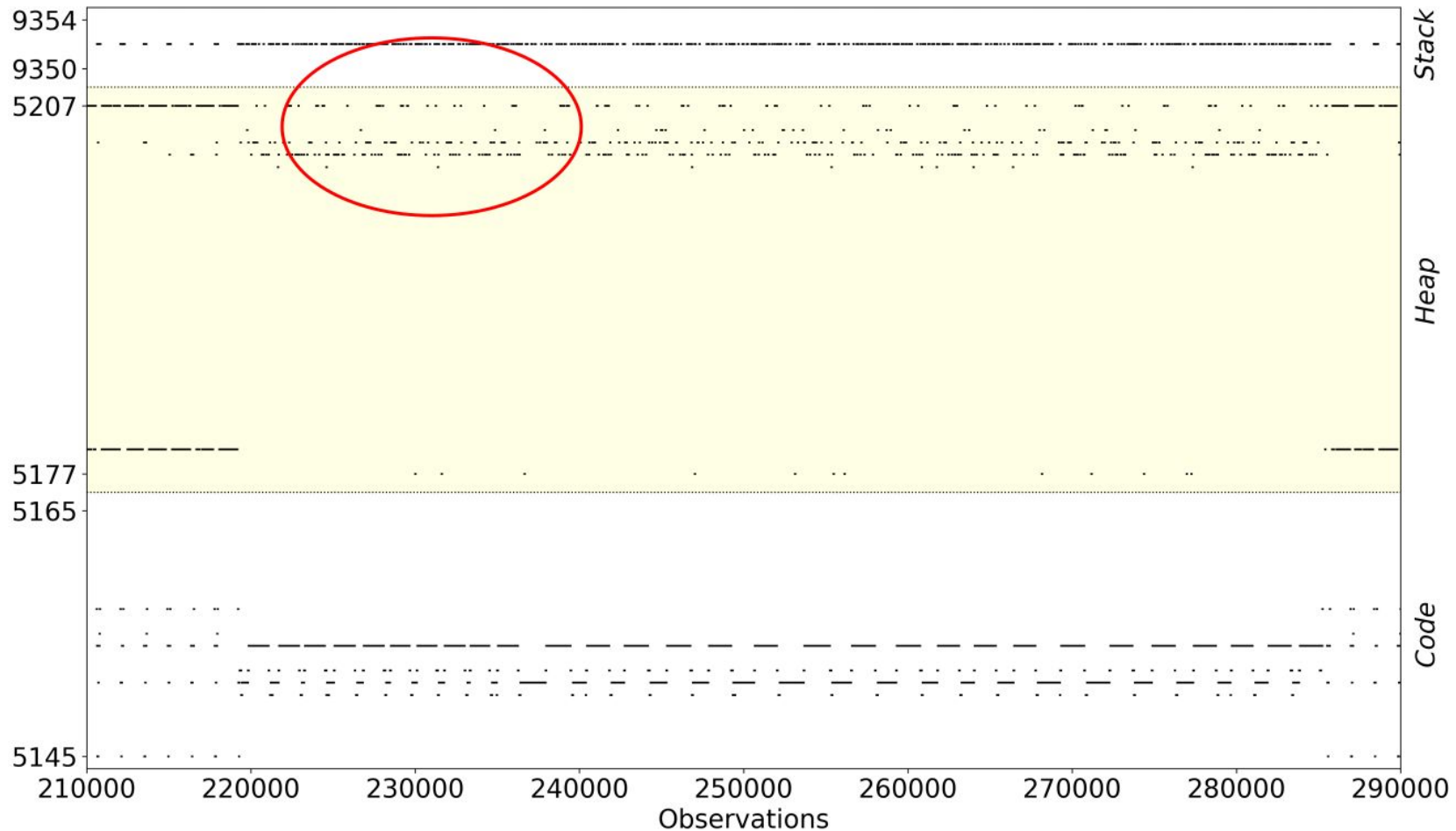
TLBlur: Self-Monitoring and Restoring Enclave Page Accesses



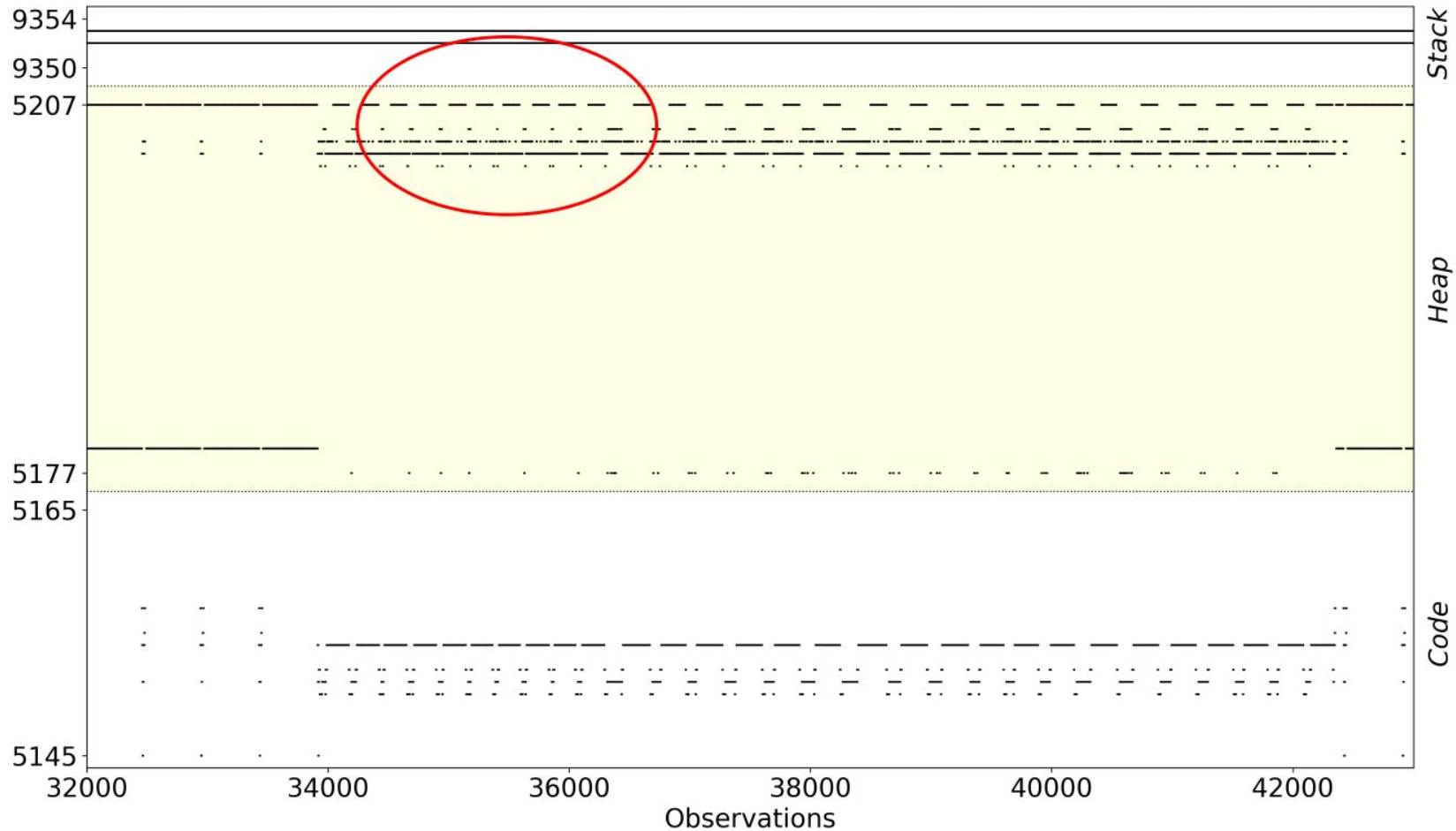
Instrumentation to Self-Monitor Page Accesses at Runtime



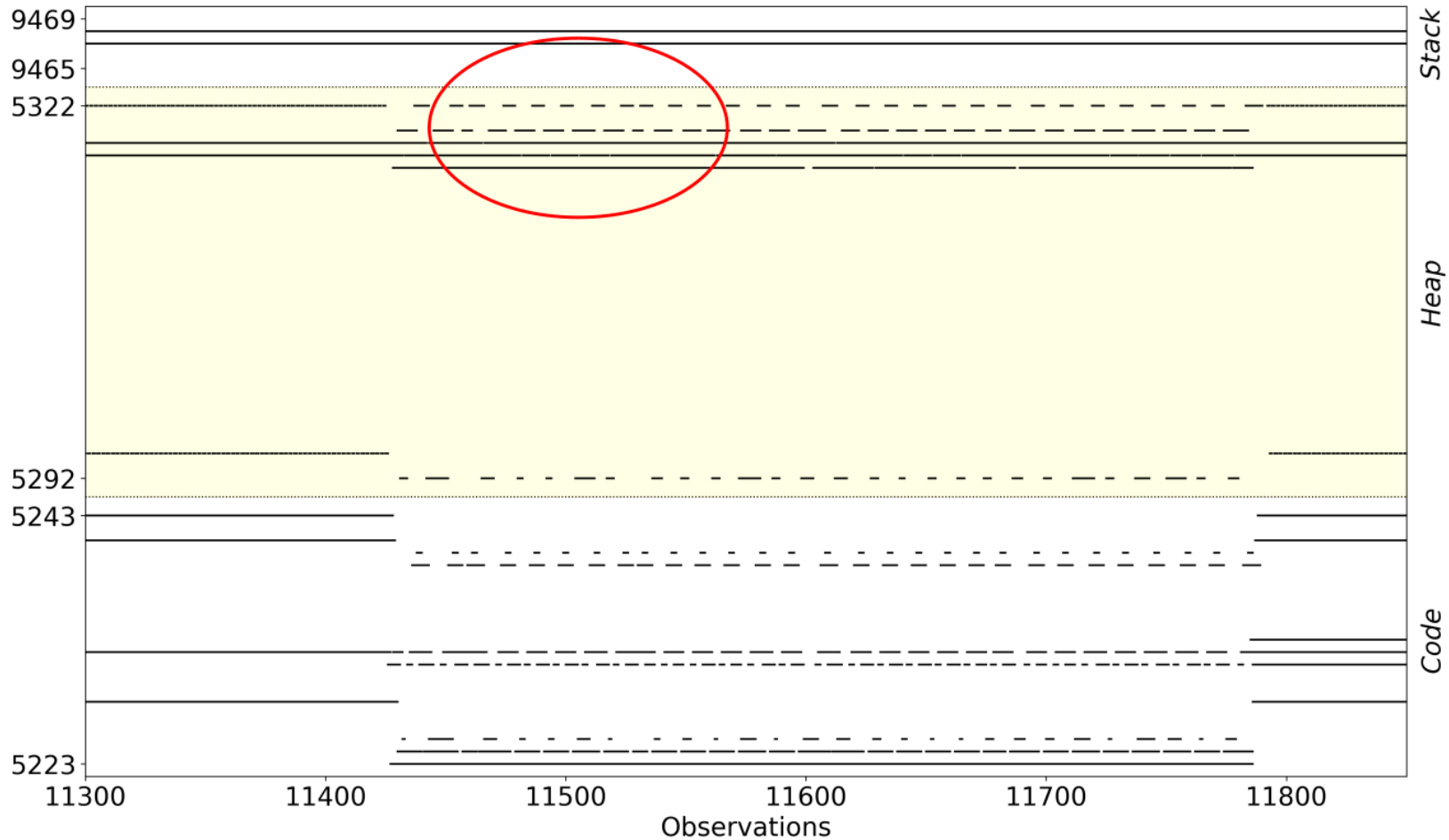
Leakage Reduction in Practice: Libjpeg Single-Stepping



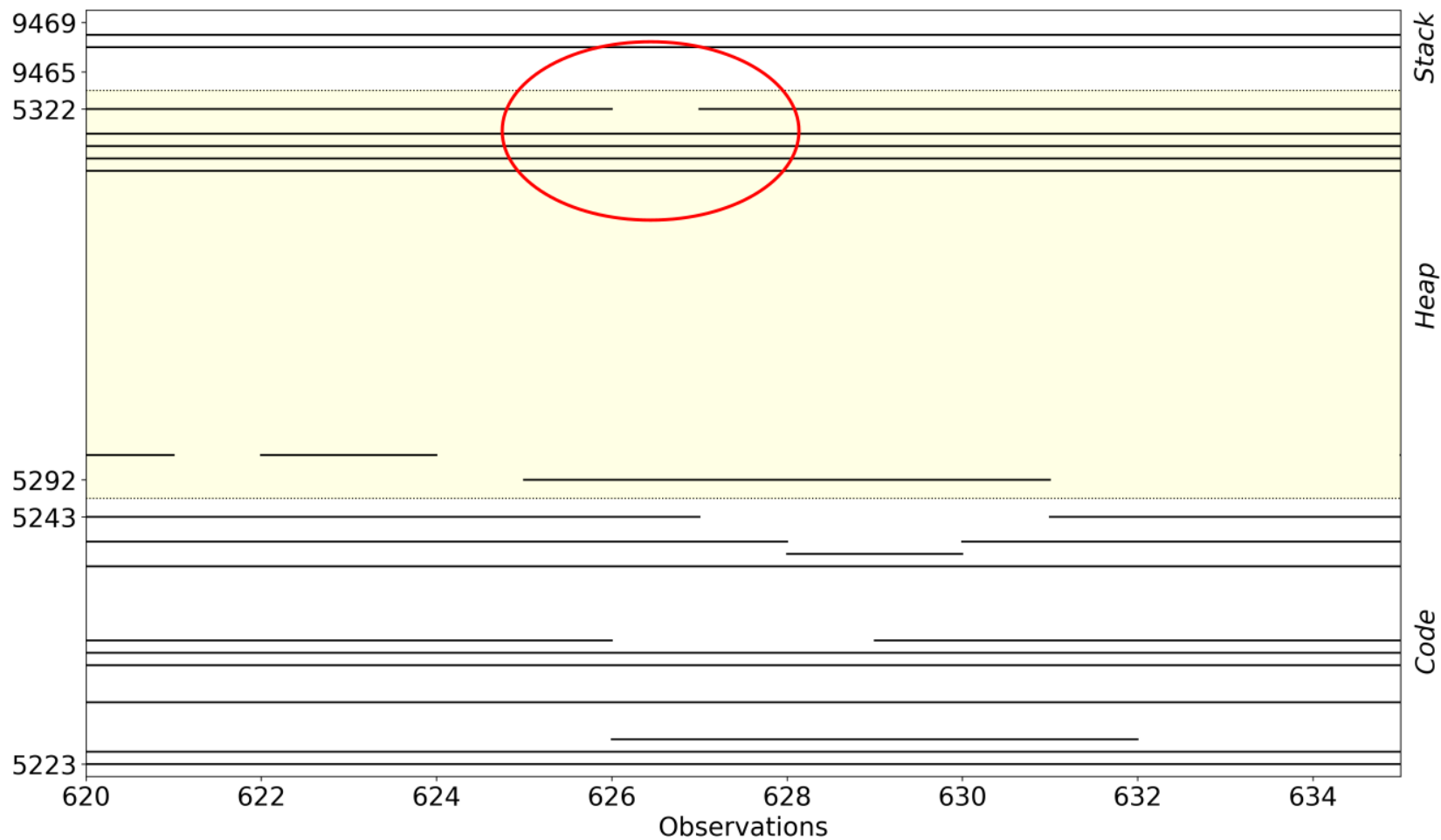
Leakage Reduction in Practice: Libjpeg Page Faults



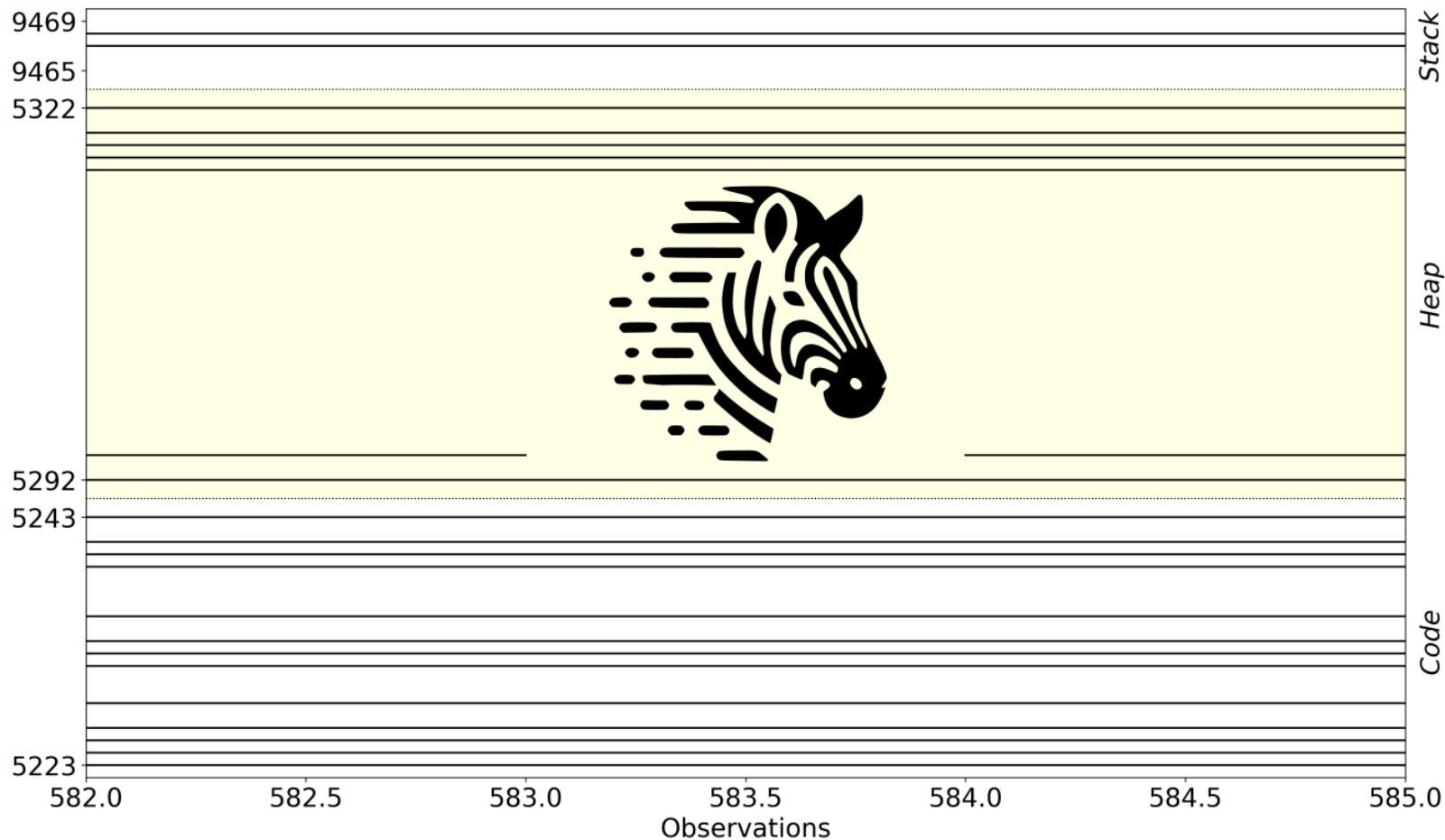
Leakage Reduction in Practice: Libjpeg TLBlur (N=10)



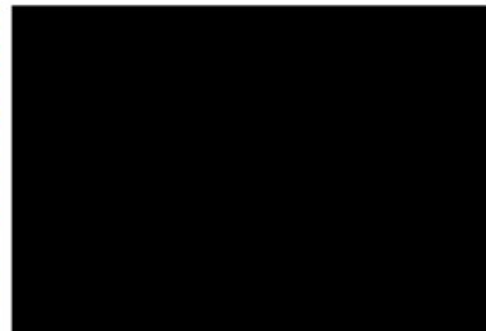
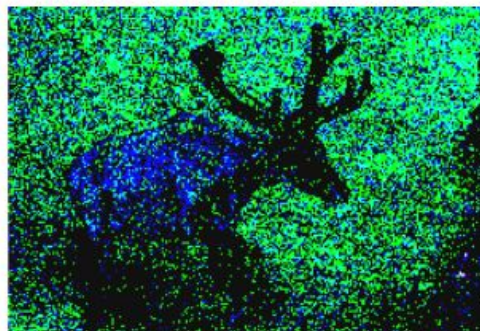
Leakage Reduction in Practice: Libjpeg TLBlur (N=20)



Leakage Reduction in Practice: Libjpeg TLBlur (N=30)



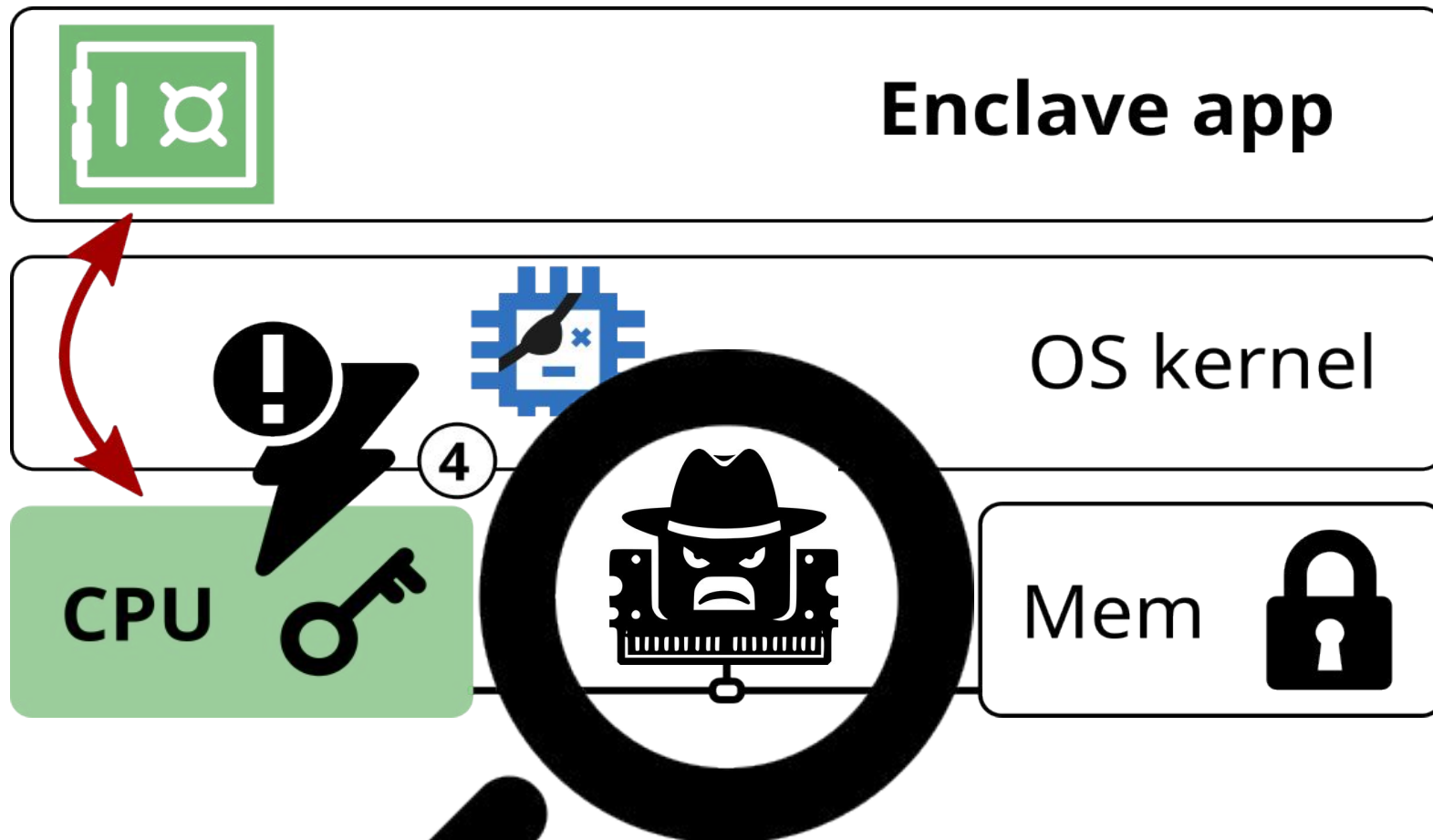
TLBlur: Compiler-Assisted Leakage Reduction in Practice



Automated “blurring” of page-access traces in space and time



Confidential-Computing: Off-Chip Attacks?



32.6. Encryption engines

In order to conceal the enclave data while it is out of the CPU package, the memory controller has an encryption engine to transparently encrypt and decrypt enclave memory.

In CPUs prior to Ice Lake, the Memory Encryption Engine (MEE) is used to encrypt pages leaving the CPU caches. MEE uses a n-ary Merkle tree with root in SRAM to maintain integrity of the encrypted data. This provides integrity and anti-replay protection but does not scale to large memory sizes because the time required to update the Merkle tree grows logarithmically in relation to the memory size.

CPUs starting from Icelake use Total Memory Encryption (TME) in the place of MEE. TME-based SGX implementations do not have an integrity Merkle tree, which means integrity and replay-attacks are not mitigated. B, it includes additional changes to prevent cipher text from being returned and SW memory aliases from being created.

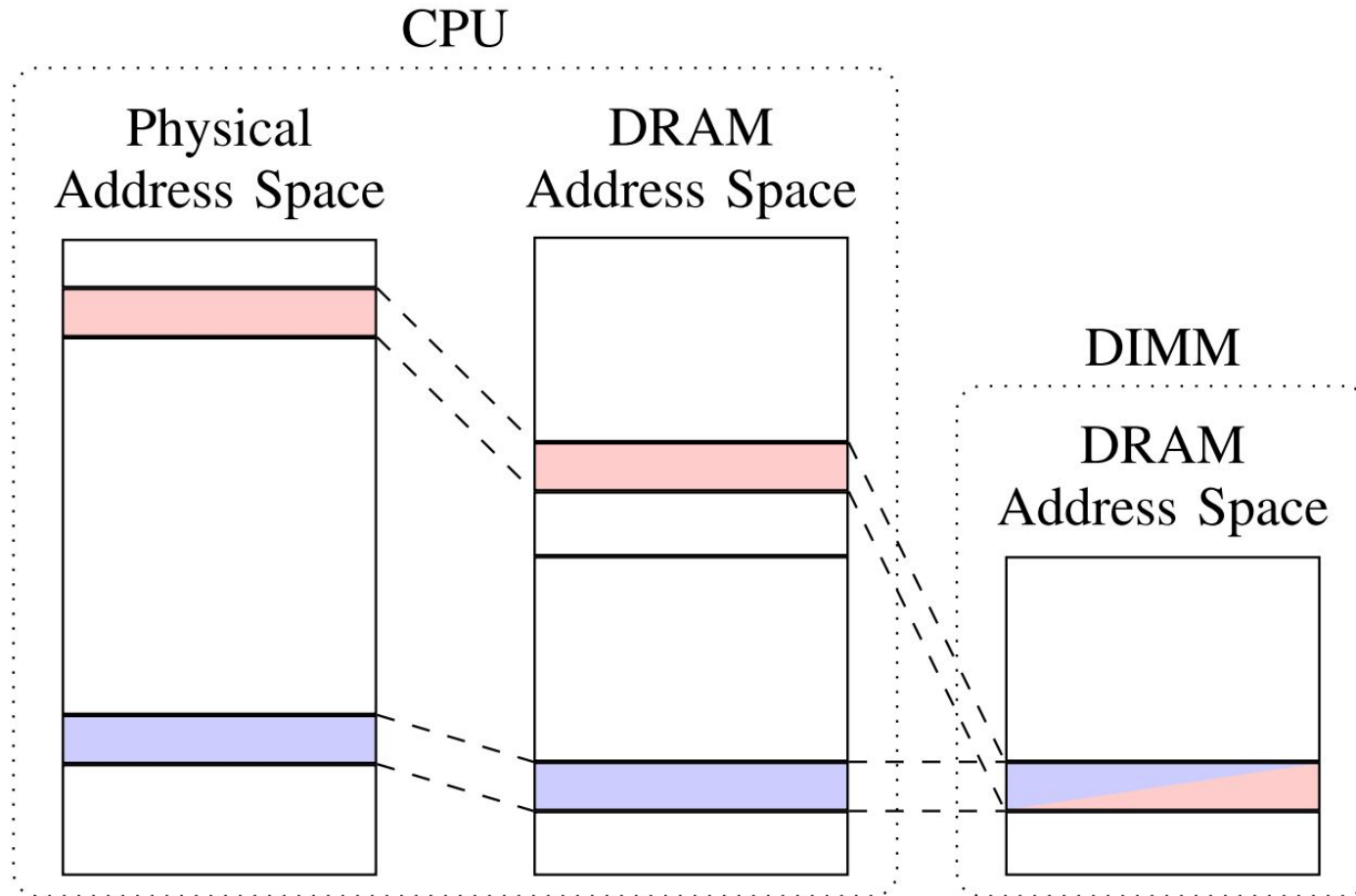
DMA to enclave memory is blocked by range registers on both MEE and TME systems (SDM section 41.10).

Background: TEE Trust in DRAM

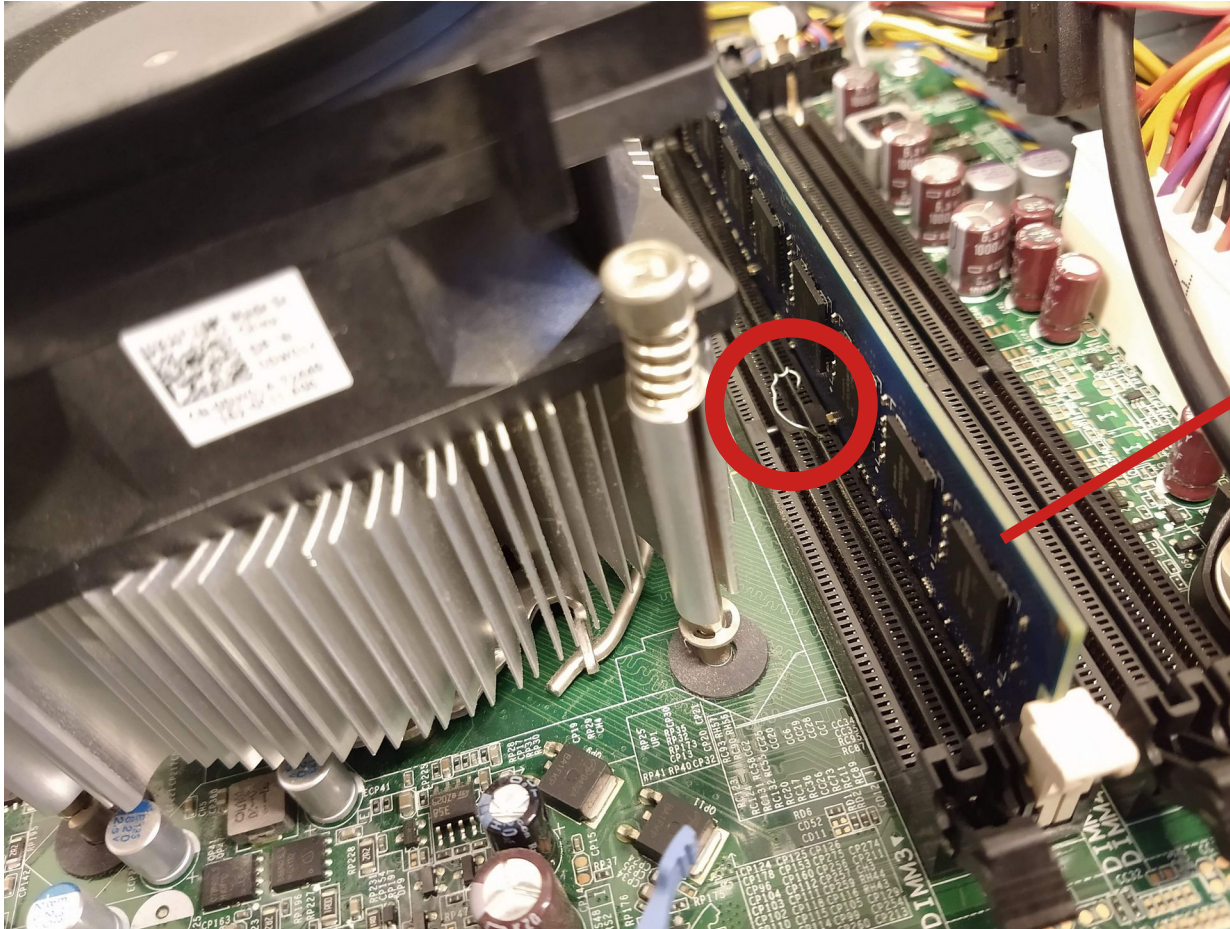
- Originally: Strong cryptographic protection
 - Limitations: Overhead, small size, ...
- Scalable solutions move away from strong cryptographic guarantees

| TEE | Encryption | Guarantees | | |
|--------------------|---------------|-----------------|-----------|-----------|
| | | Confidentiality | Integrity | Freshness |
| Classic Intel SGX | AES-CTR | ✓ | ✓ | ✓ |
| Scalable Intel SGX | AES-XTS | ✓ | ✗ | ✗ |
| Intel TDX | AES-XTS | ✓ | ✗ | ✗ |
| AMD SEV-SNP | AES-XEX | ✓ | ✗ | ✗ |
| Arm CCA | AES-XEX/QARMA | ✓ | ✗ | ✗ |

Idea: Memory Aliasing Attacks



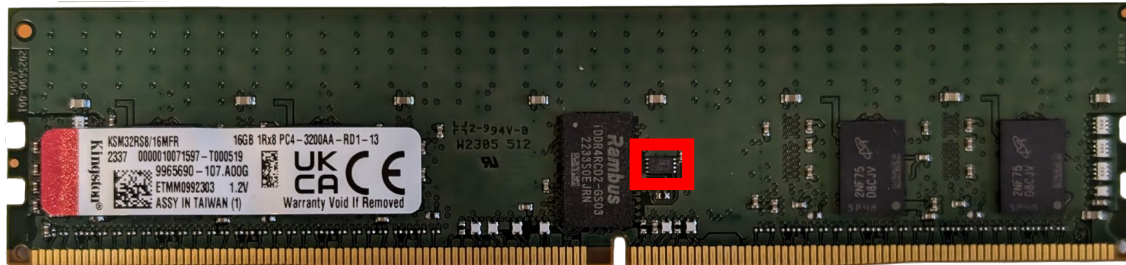
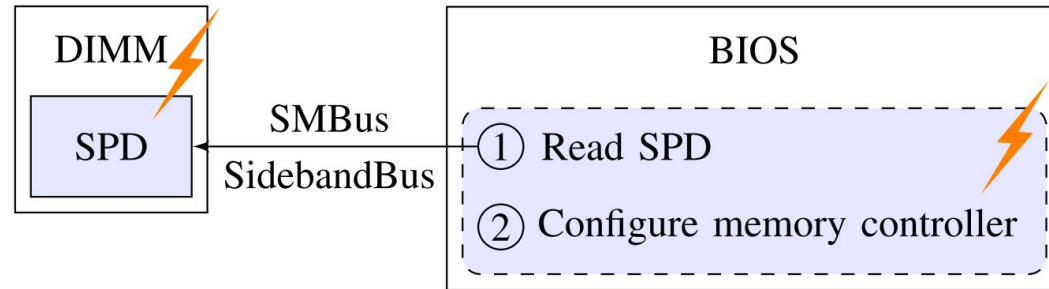
DDR3: The “Paperclip” Attack...



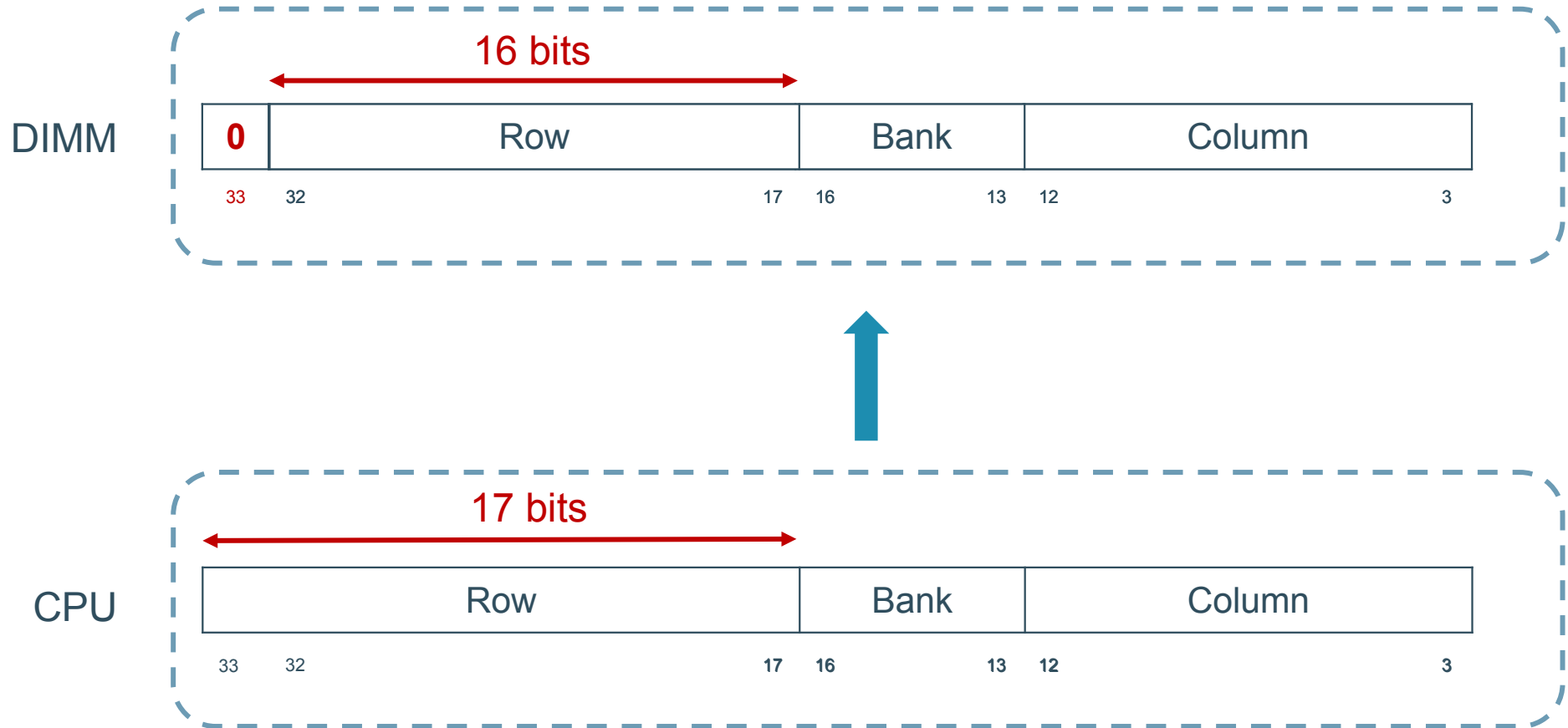
DDR3
<>
DDR4...

Background: Memory Initialization

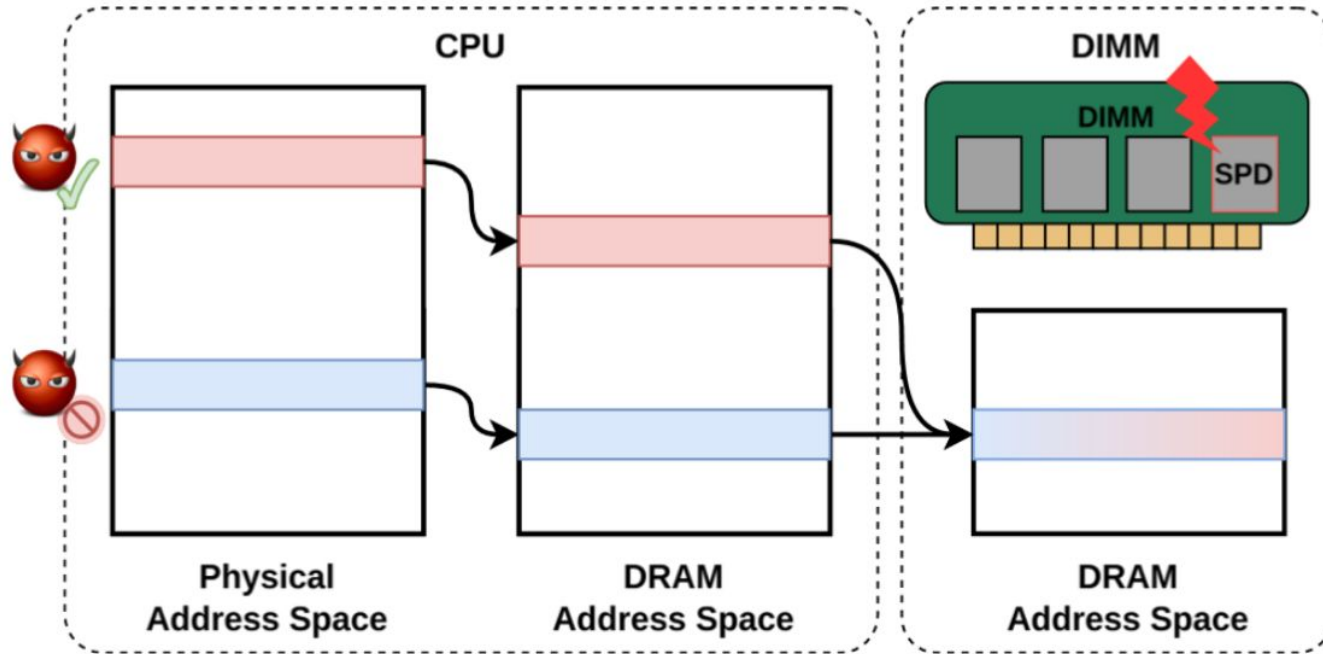
- BIOS programs memory controller based on DIMM configuration
- Incorrect configuration leads to inconsistent memory view



SPD-Based Memory Address Aliasing



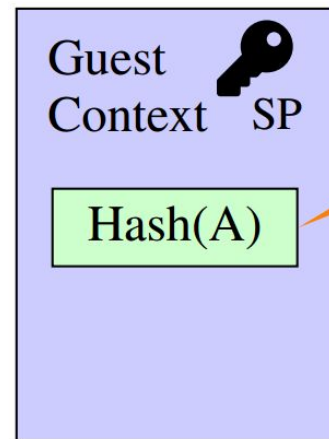
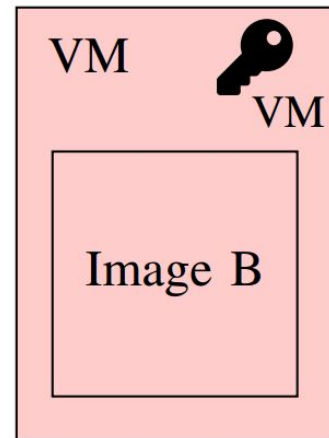
SPD-Based Memory Address Aliasing



- BIOS configures memory controller
- **Malicious SPD contents** introduces aliases

Breaking AMD SEV-SNP

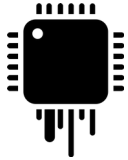
- Attestation
 - SP takes measurement of launched VM
 - Encrypted under SP key
 - No freshness, can replay launch digest



Conclusions and Take-Away



New era of **confidential computing** for the cloud and IoT



... but current architectures are **not perfect!**



Scientific understanding driven by **attacker-defender race**



Thank you!