The Dark Side of Privilege

Understanding and Mitigating Software-Based Side Channels on Trusted Execution Environments

Jo Van Bulck

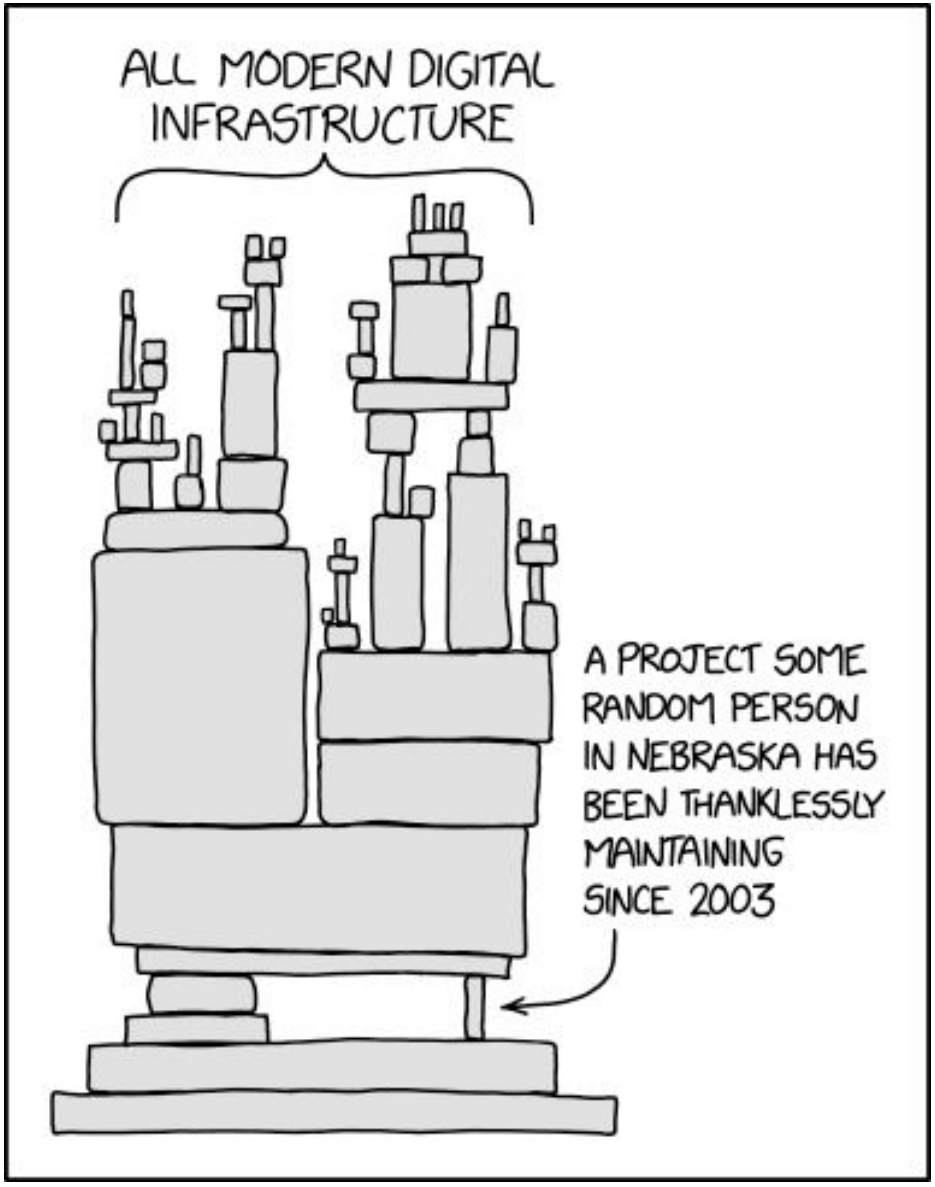
DistriNet, KU Leuven, Belgium jo.vanbulck@cs.kuleuven.be vanbulck.net

MIC-SEC Winter School, Dec 2, 2025

DistriNet

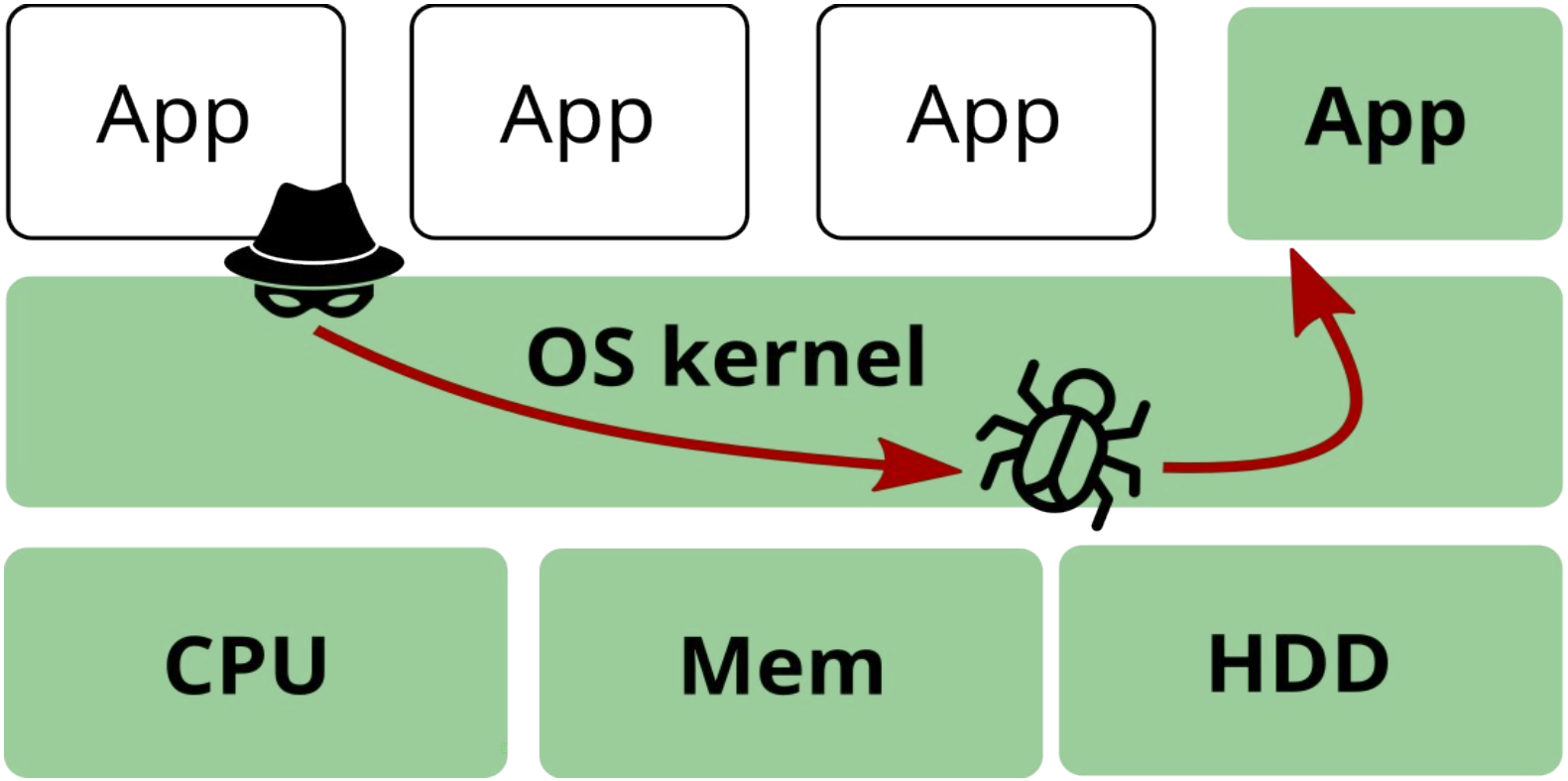
KU LEUVEN

Trust?



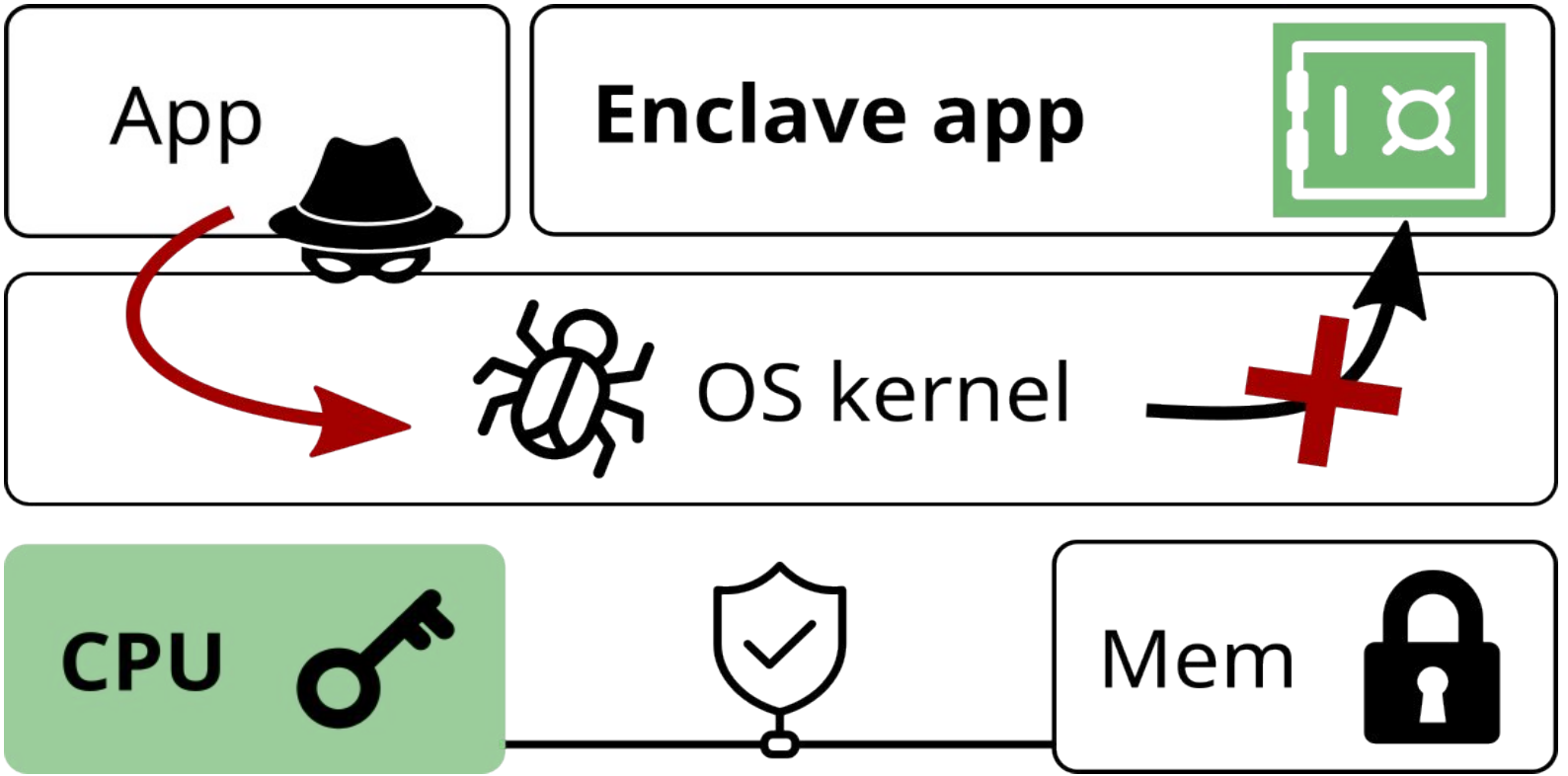
https://xkcd.com/2347/

Confidential Computing: Reducing Attack Surface



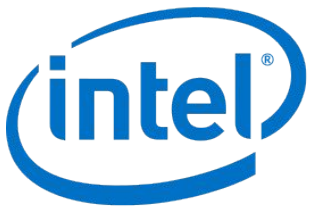
Traditional layered designs: Large trusted computing base

Confidential Computing: Reducing Attack Surface



Trusted execution: Hardware-level isolation and attestation

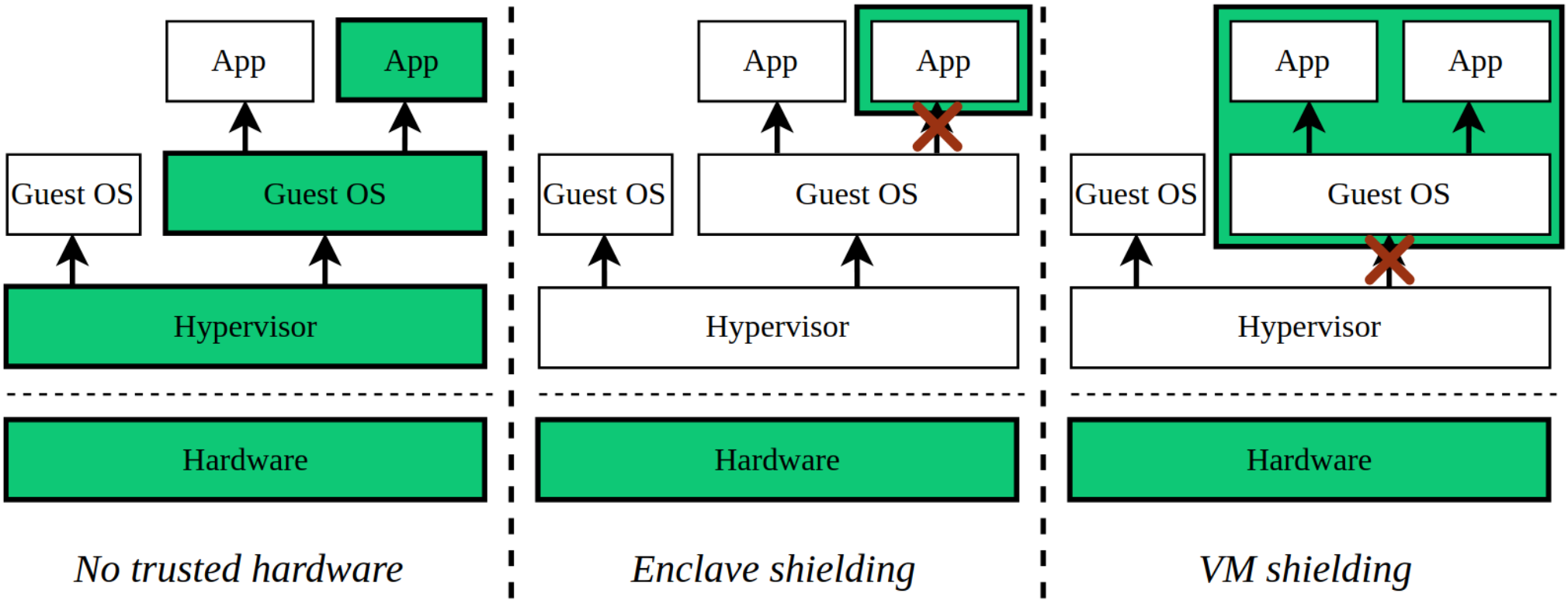
The Rise of Trusted Execution Environments (TEEs)



- 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...

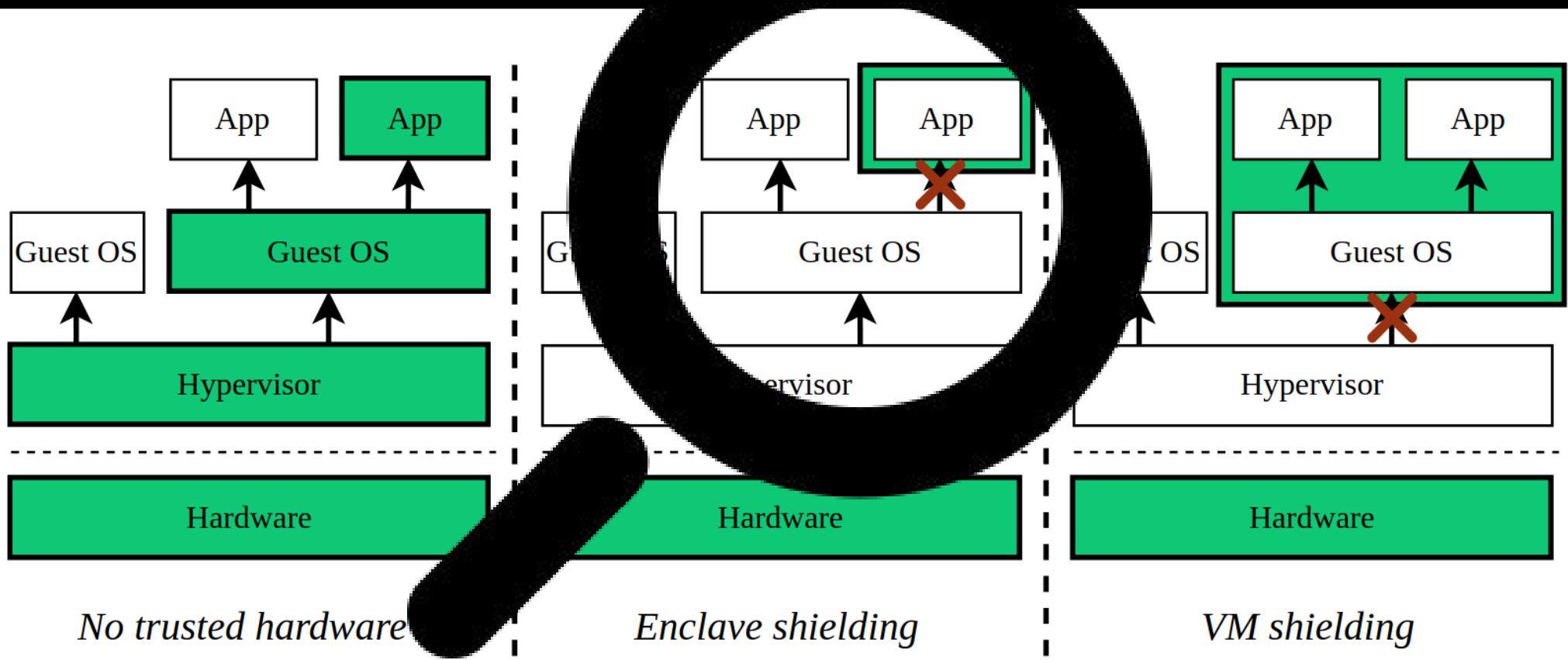
Confidential Computing Isolation Paradigms



Intel SGX, Arm TrustZone-M

AMD SEV, Intel TDX, Arm CCA (Arm TrustZone-A)

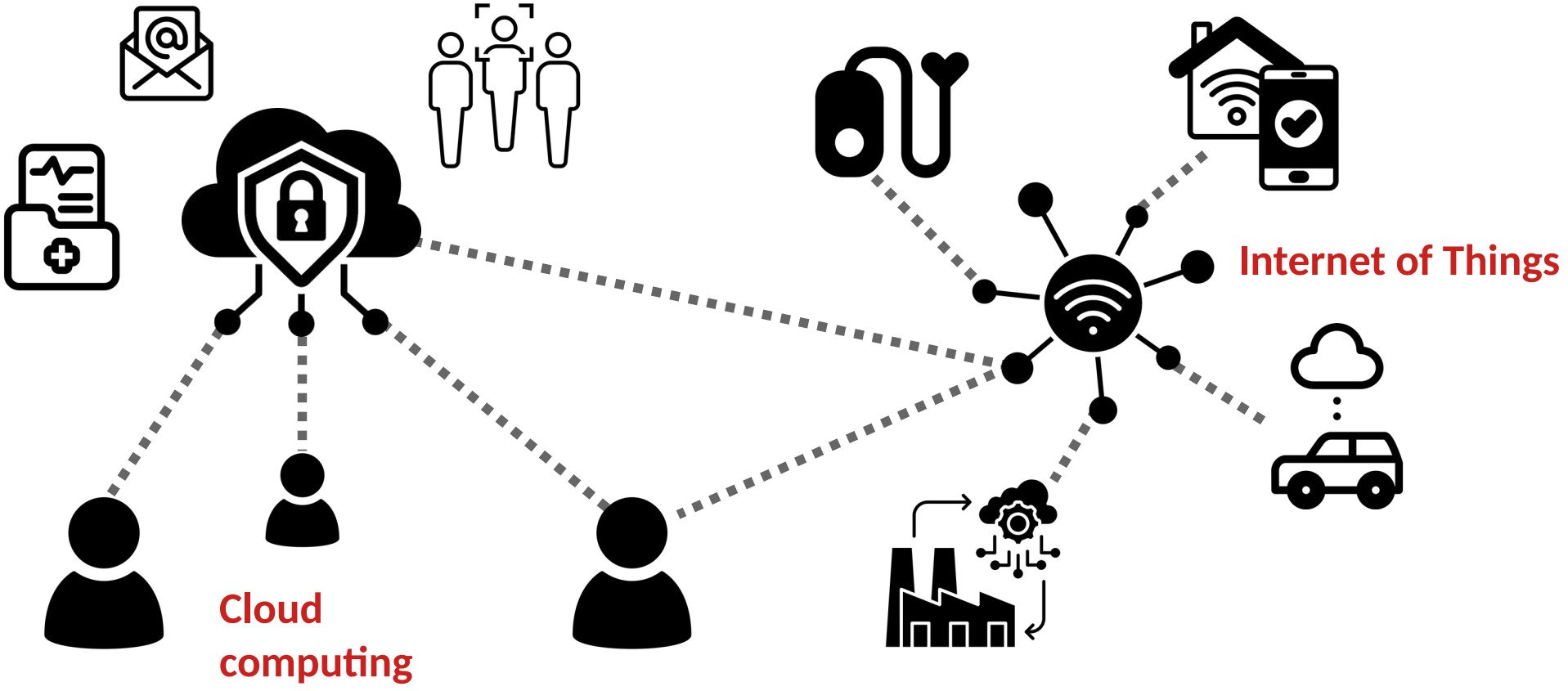
Confidential Computing Isolation Paradigms *(Focus for Today)*



Intel SGX, Arm TrustZone-M

AMD SEV, Intel TDX, Arm CCA (Arm TrustZone-A)

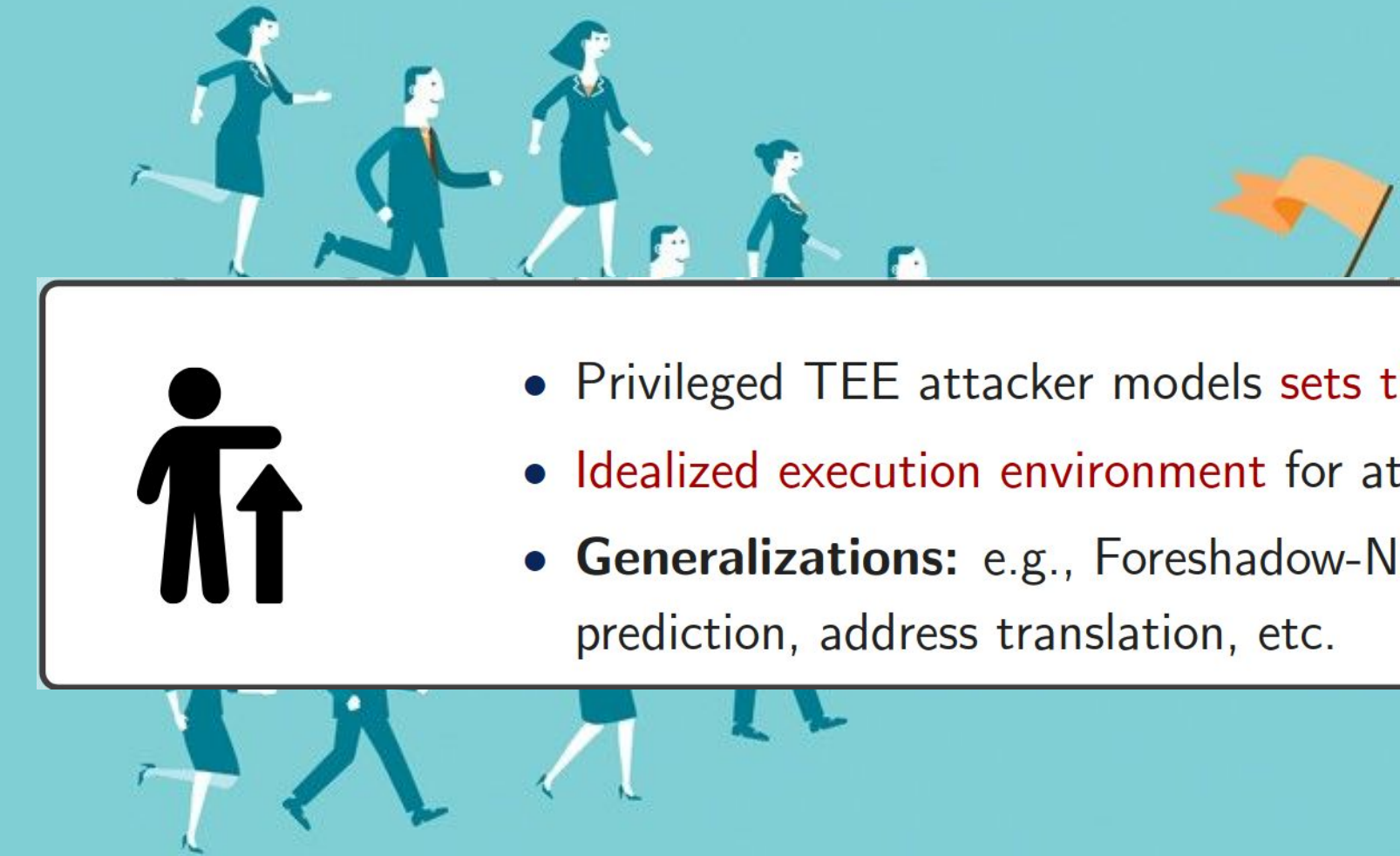
“Confidential Computing Today, Just Computing Tomorrow” *



* *Mark Russinovich, CTO Microsoft Azure*

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?

intel

Overview | About Intel | News & Events | Financial Info | Stock Info | Filings & Reports | Board & Governance | ESG

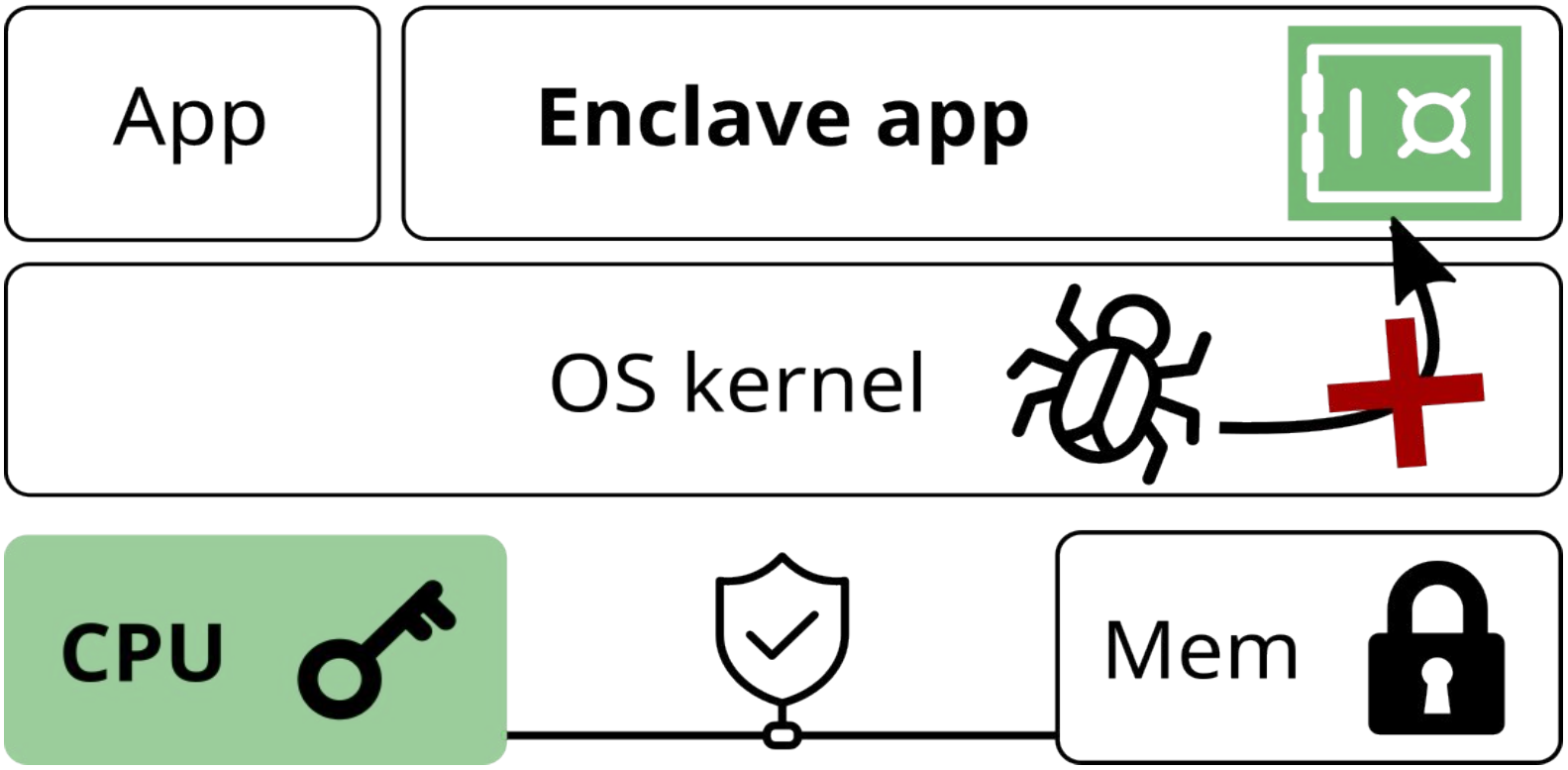
Overview | Press Releases | IR Calendar | Annual Stockholders' Meeting | Investor Meeting | Email Alerts | Presentations

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.

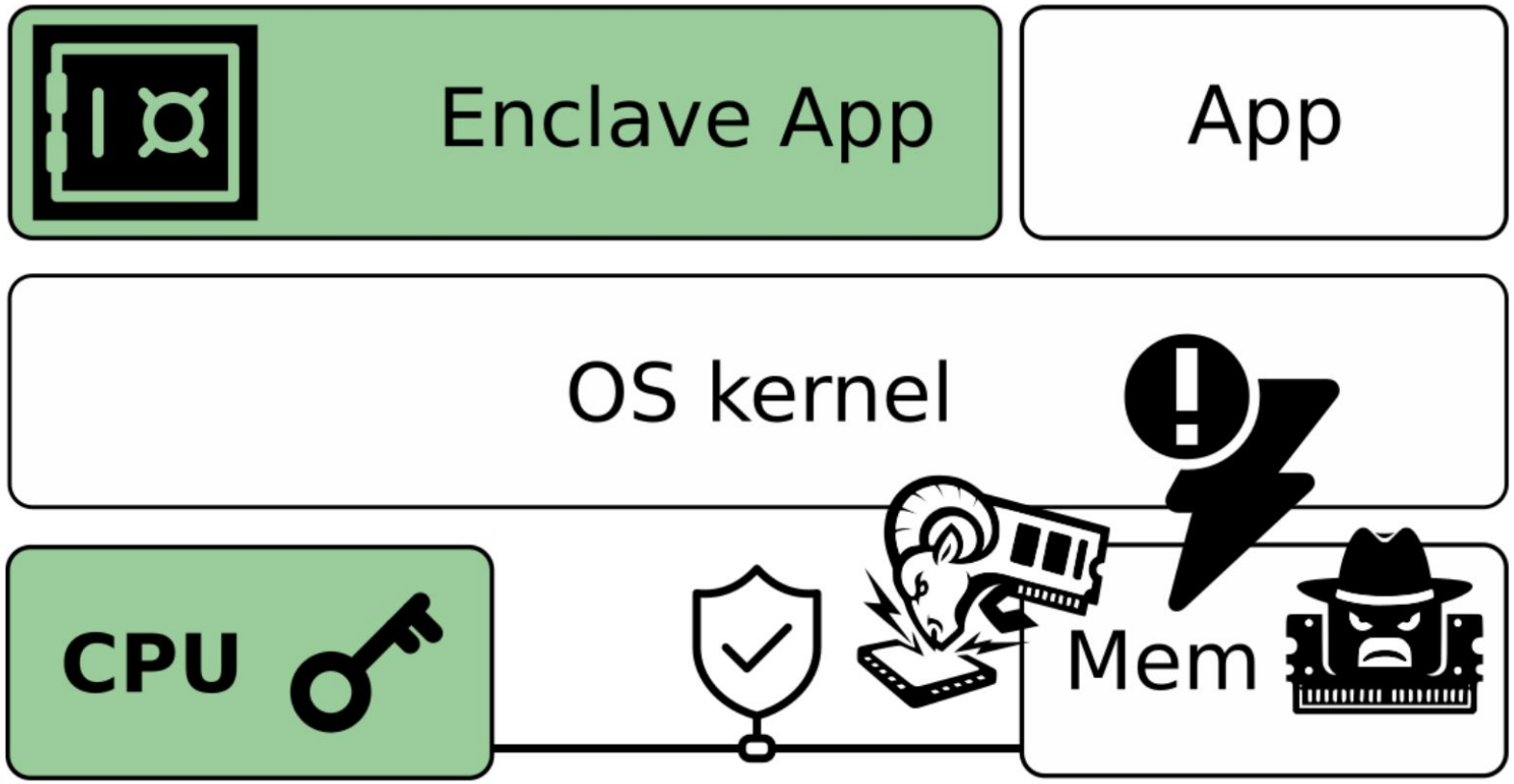
https://www.intc.com/news-events/press-releases/detail/1423/intel-xeon-scalable-platform-built-for-most-sensitive

Recap: Reducing Attack Surface with Enclaves

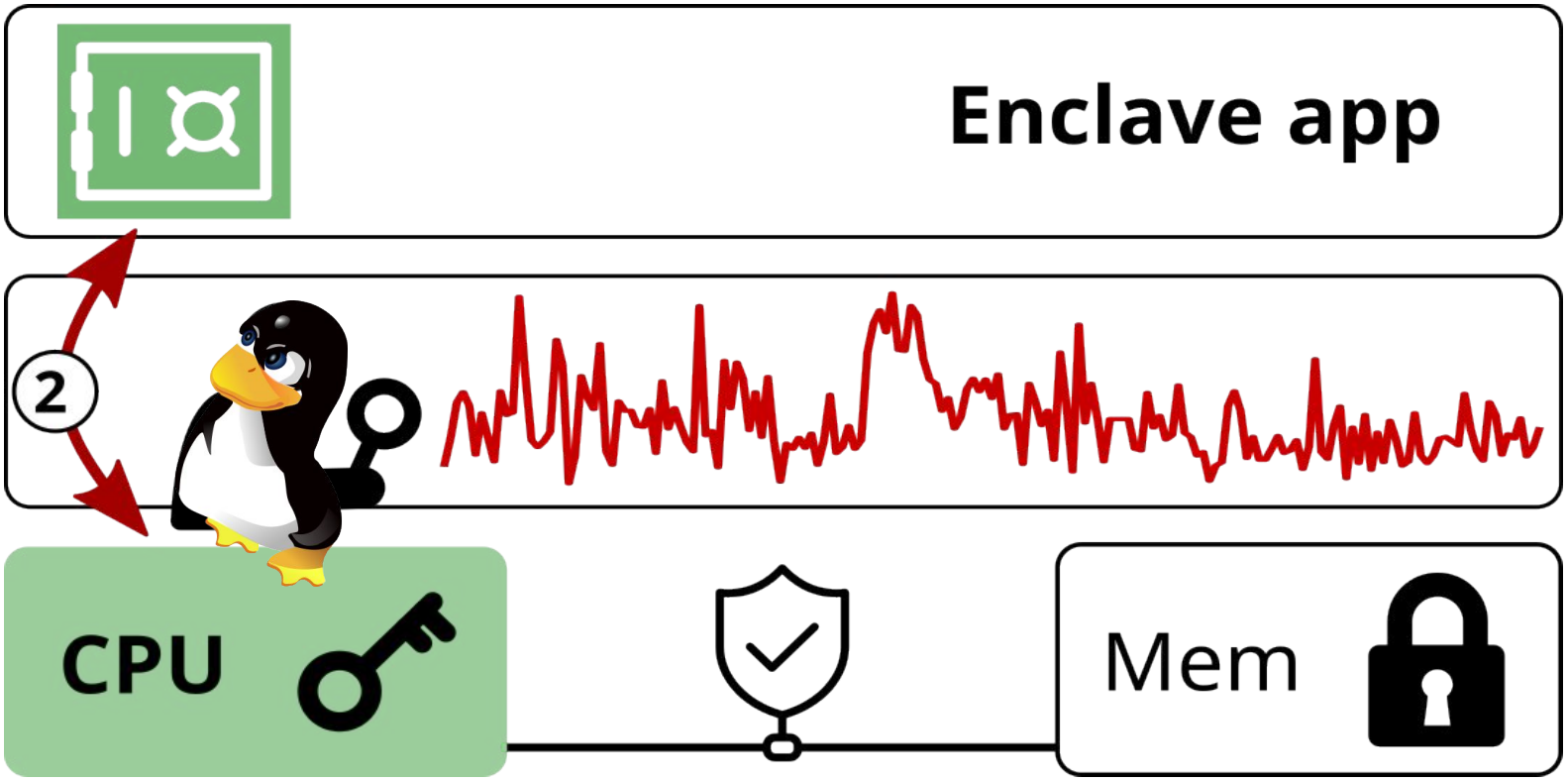


Intel SGX promise: Hardware-level **isolation and attestation**

Not Today: Physical-Access Attacks *(See Ingrid & Jesse, Wed)*



Today: Privileged Software-Based Side-Channel Attacks



Game changer: Untrusted OS → New class of powerful side channels!

Intel® SGX and Side-Channels

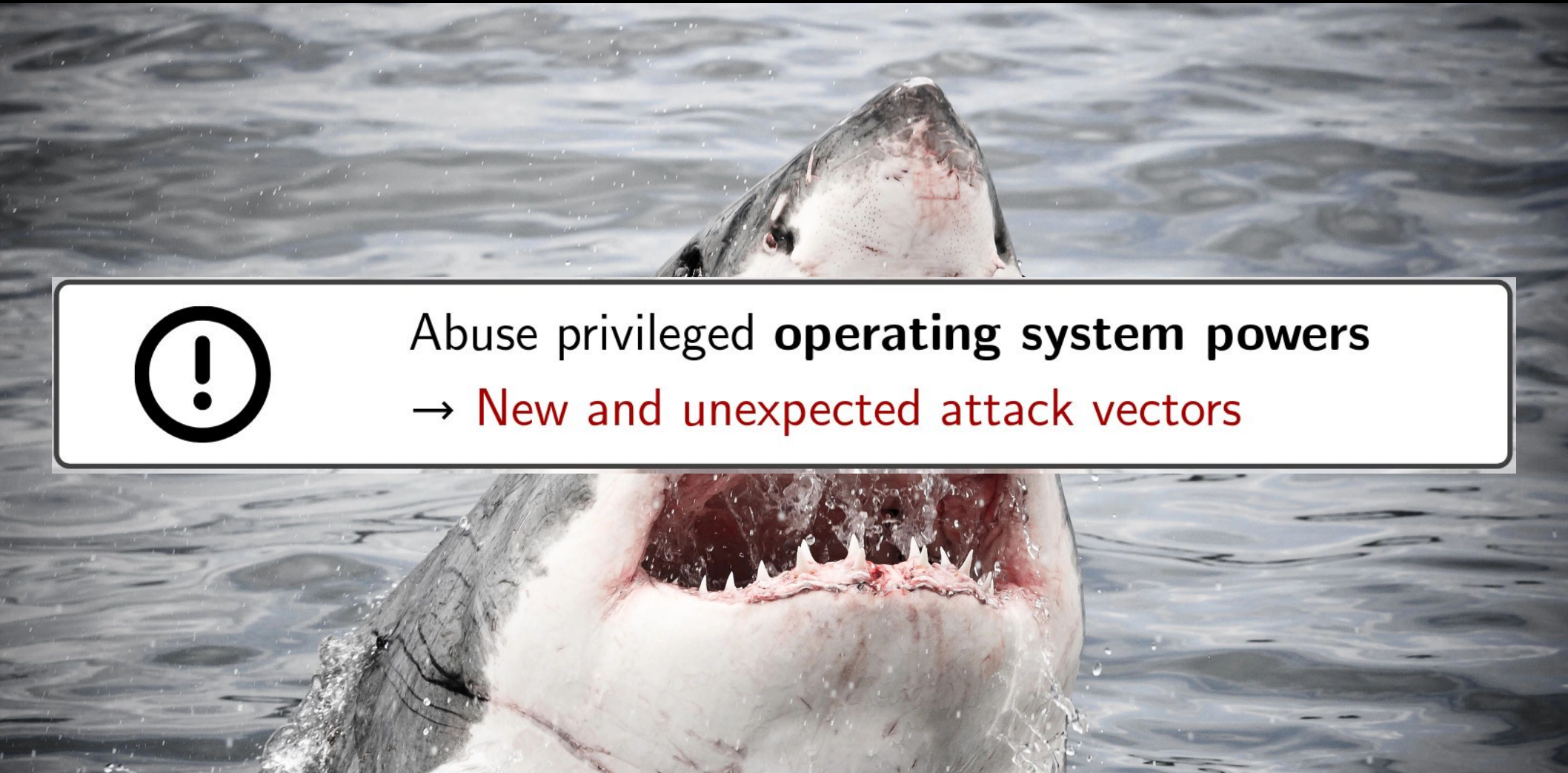
By Simon Paul Johnson, Published: 03/17/2017, Last Updated: 02/27/2018

Since launching Intel® Software Guard Extensions (Intel® SGX) on 6th Generation Intel® Core™ processors in 2015, there have been a number of academic articles looking at various usage models and the security of Intel SGX. Some of these papers focus on a class of attack known as a side-channel attack, where the attacker relies on the use of a shared resource to discover information about processing occurring in some other privileged domain that it does not have direct access to.

In general, these research papers do not demonstrate anything new or unexpected about the Intel SGX architecture. Preventing side channel attacks is a matter for the enclave developer. Intel makes this clear In the security objectives for Intel SGX, which we published as part of our workshop tutorial at the International Symposium on Computer Architecture in 2015, the slides for which can be found here [slides 109-121], and in the Intel® SGX SDK Developer's Manual.

Game Changer: Privileged “Bottom-Up” Adversary Model

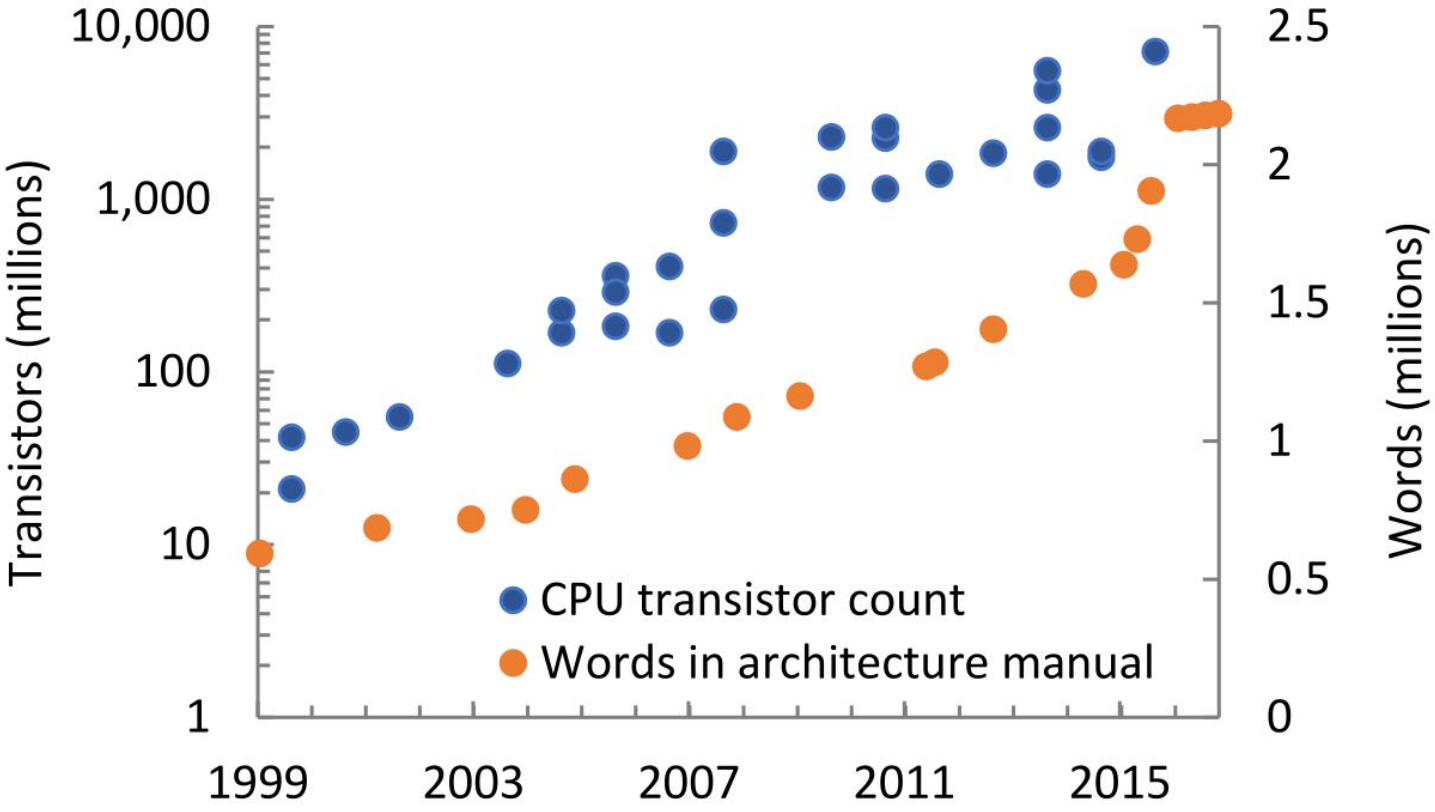
Game Changer: Privileged “Bottom-Up” Adversary Model



Abuse privileged **operating system powers**

→ New and unexpected attack vectors

Privileged x86 Interface and Complexity Growth…



Baumann, "Hardware is the New Software", HotOS'17.

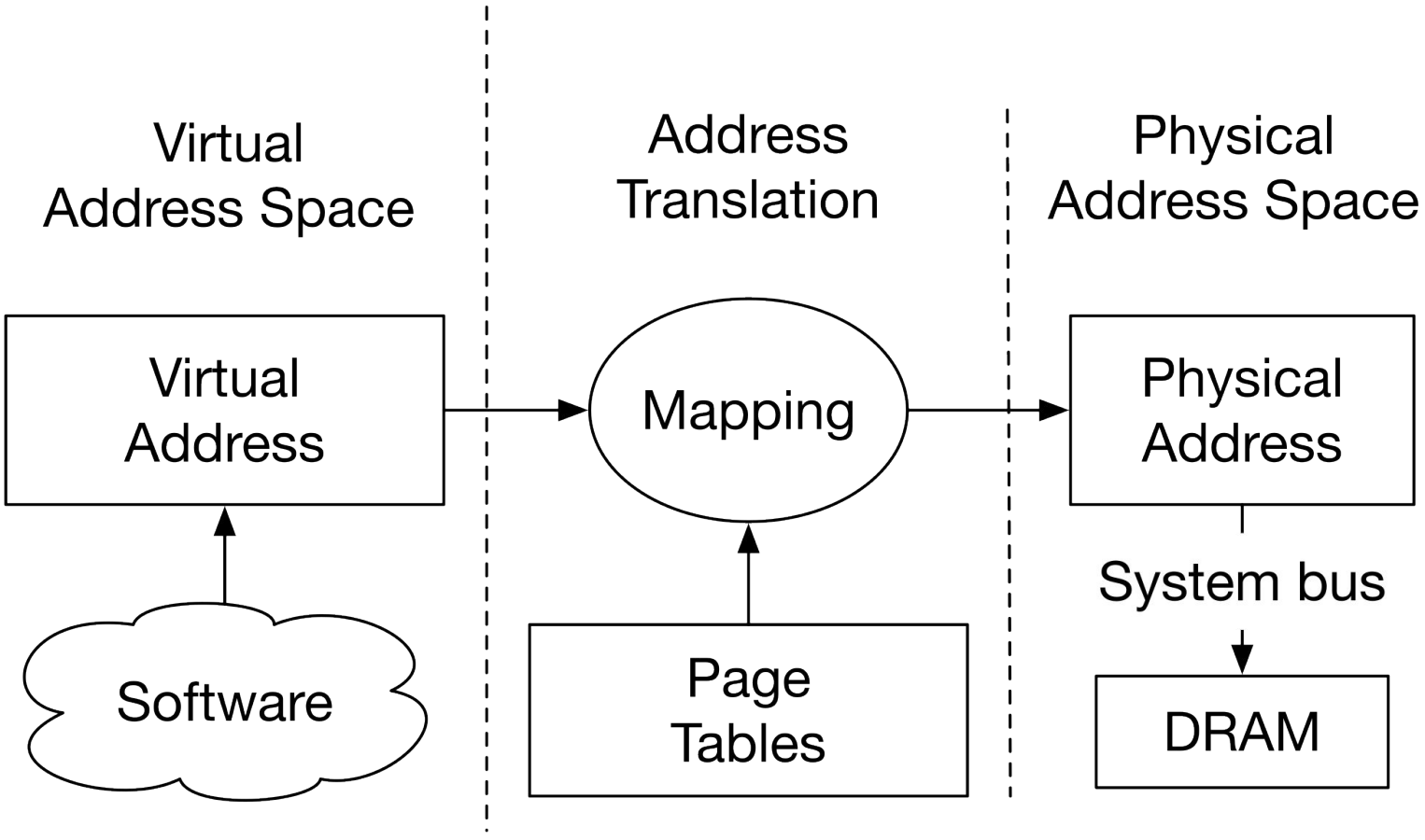
Systematizing the SGX Attack Landscape

	Attack \ Properties	x86 Interface						Constraints			
		PTE	GDT	IRQ	MSR	CR0	PMC	SMT	REP	SAE	Granular
μ-arch contention	Cache priming [181, 92, 29, 225, 79]	◐	○	◐	◐	○	◐	◐	●	○	64 B
	Branch prediction [156, 59, 105]	◐	○	◐	◐	◐	◐	◐	●	○	Inst
	DRAM row buffer conflicts [263]	○	○	○	◐	◐	○	○	●	○	1-8 KiB
	False dependencies [180]	○	○	○	◐	○	○	●	●	○	4 B
	Interrupt latency [**256**, 95, 208]	◐	○	●	◐	○	○	○	●	○	Inst
	Port contention [7]	○	○	○	◐	○	○	●	●	○	μ-op
Control channel	Page faults [277]	●	○	○	○	○	○	○	○	○	4 KiB
	Page table A/D [**258**, 263]	●	○	◐	○	◐	○	◐	○	○	4 KiB
	Page table flushing [**258**]	●	○	◐	○	○	○	◐	○	○	32 KiB
	Interrupt counting [**182**]	●	○	●	○	○	○	○	○	○	Inst
	IA32 segmentation faults [**91**]	◐	●	◐	○	○	○	○	○	○	1 B-4 KiB
	Alignment faults [**254**]	○	○	●	○	●	○	○	○	○	1 B
Transient	Foreshadow L1D extraction [**249**]	●	○	◐	○	○	○	◐	○	○	–
	Data sampling [**223**, 216, 211]	●	○	◐	○	○	○	◐	◐	○	–
	Spectre [39, 148]	◐	○	◐	○	○	○	◐	◐	◐	–
	Load value injection [**251**]	●	○	◐	○	○	○	◐	◐	◐	–
Interface	Memory safety [**254**, 154, 24, 266]	◐	○	◐	○	○	○	○	◐	◐	–
	Undervolting [**188**, 137, 210]	◐	○	◐	●	○	○	◐	●	◐	–
	Off-chip memory address bus [152]	●	○	◐	○	◐	○	○	○	○	64 B

Possibly every privileged CPU feature can be abused!

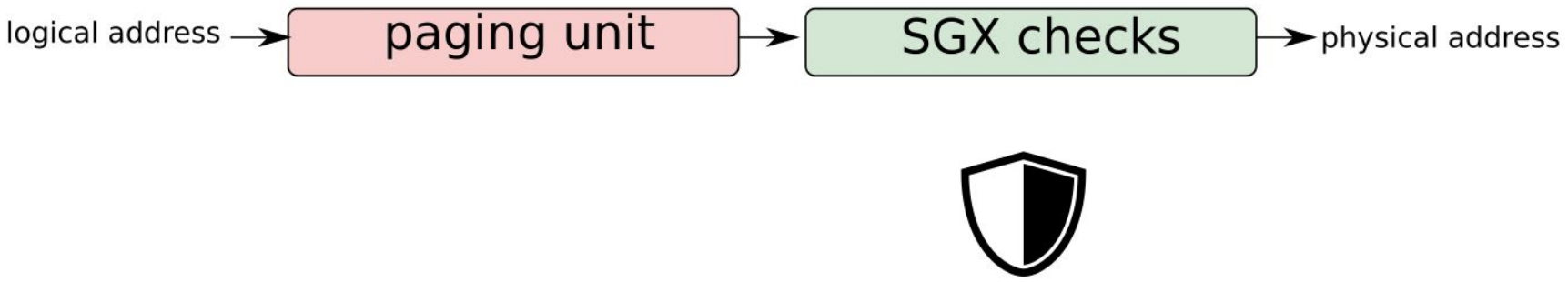
Idea #1: Deterministic Spatial Resolution with Page Tables

Background: The Virtual Memory Abstraction



Costan et al. “Intel SGX explained”, IACR 2016.

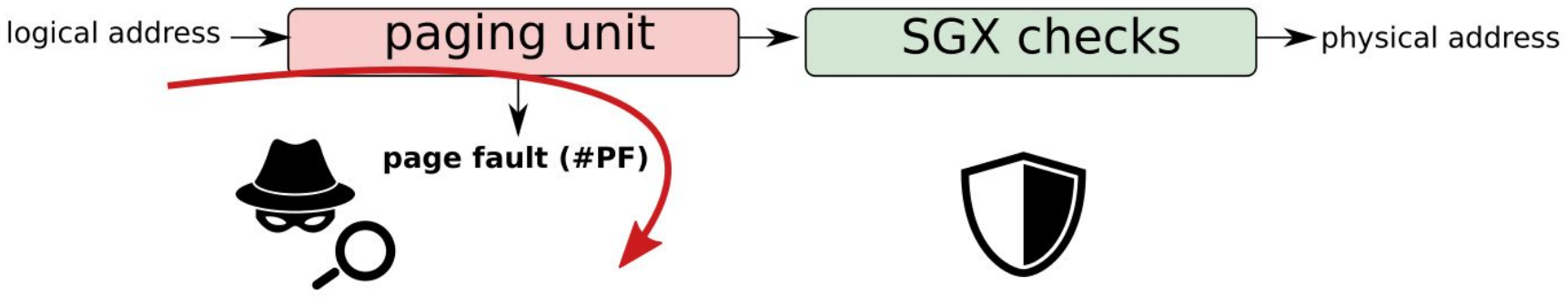
Idea: Page Faults as a Side Channel



SGX machinery protects against direct address remapping attacks

Xu et al. “Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems”, IEEE S&P 2015.

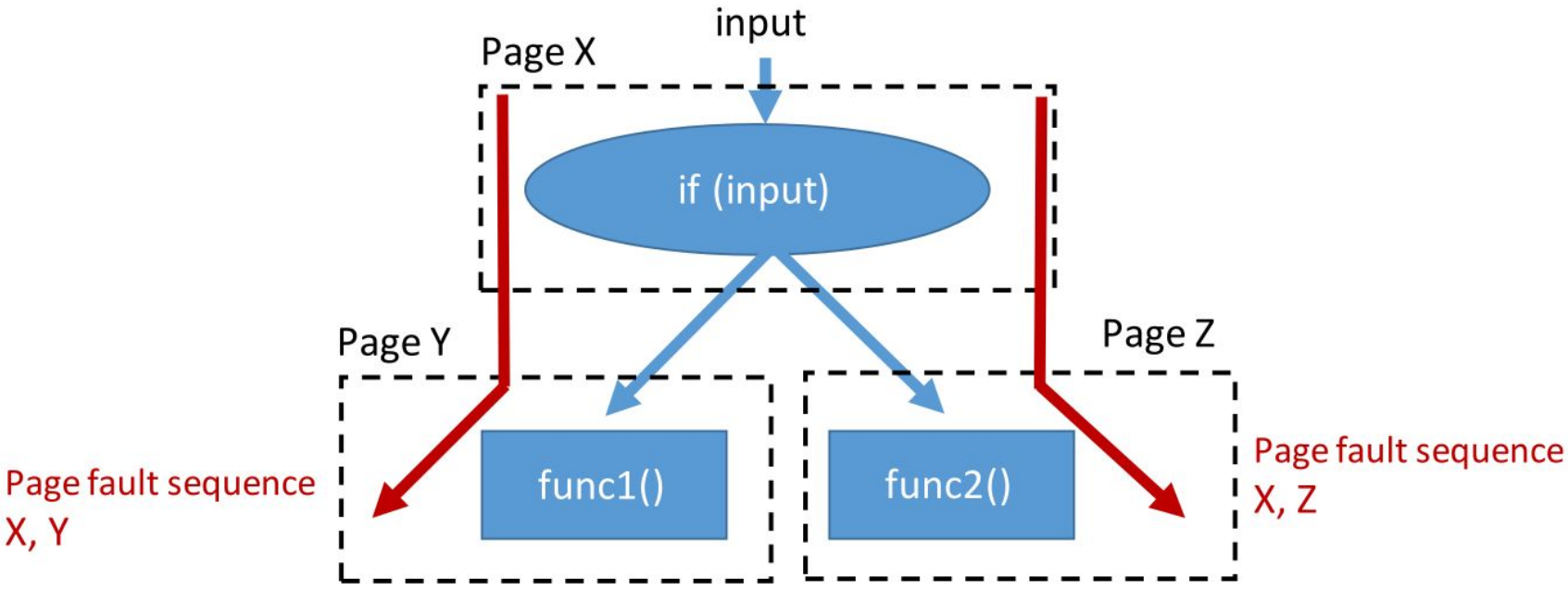
Idea: Page Faults as a Side Channel



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

Xu et al. “Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems”, IEEE S&P 2015.

Intel SGX: Page Faults as a Side Channel

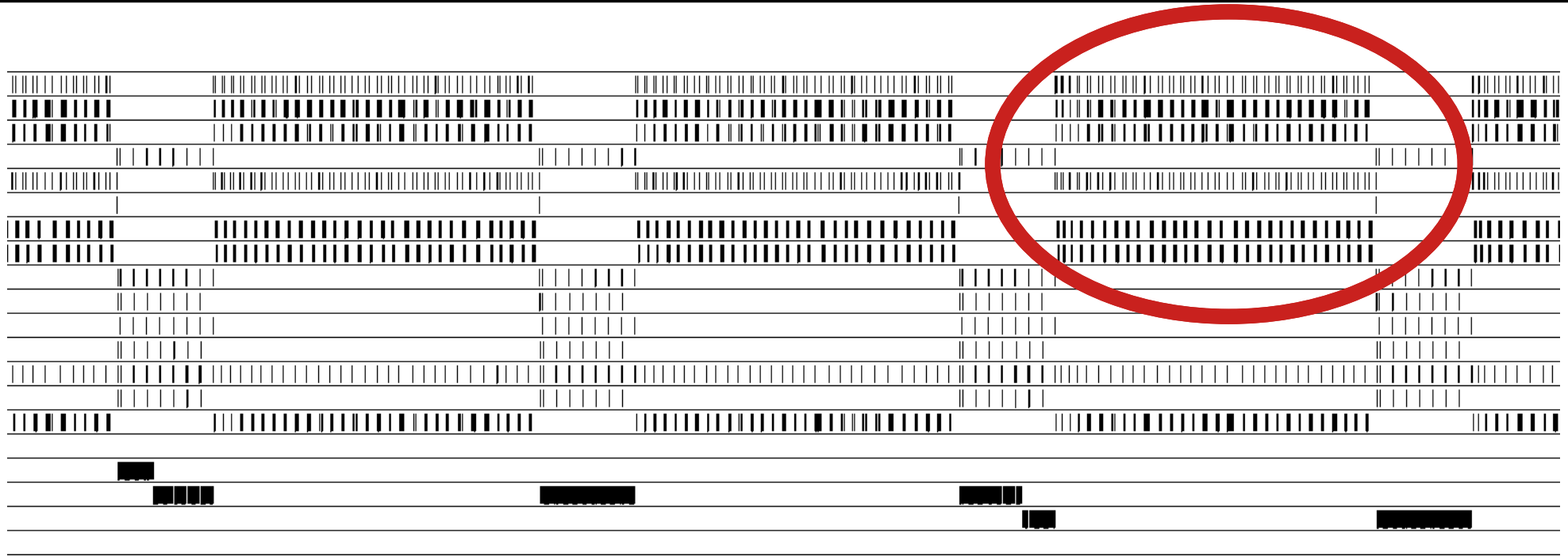


Xu et al.: "Controlled-channel attacks: Deterministic side channels for untrusted operating systems", Oakland 2015.

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

Xu et al. "Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems", IEEE S&P 2015.

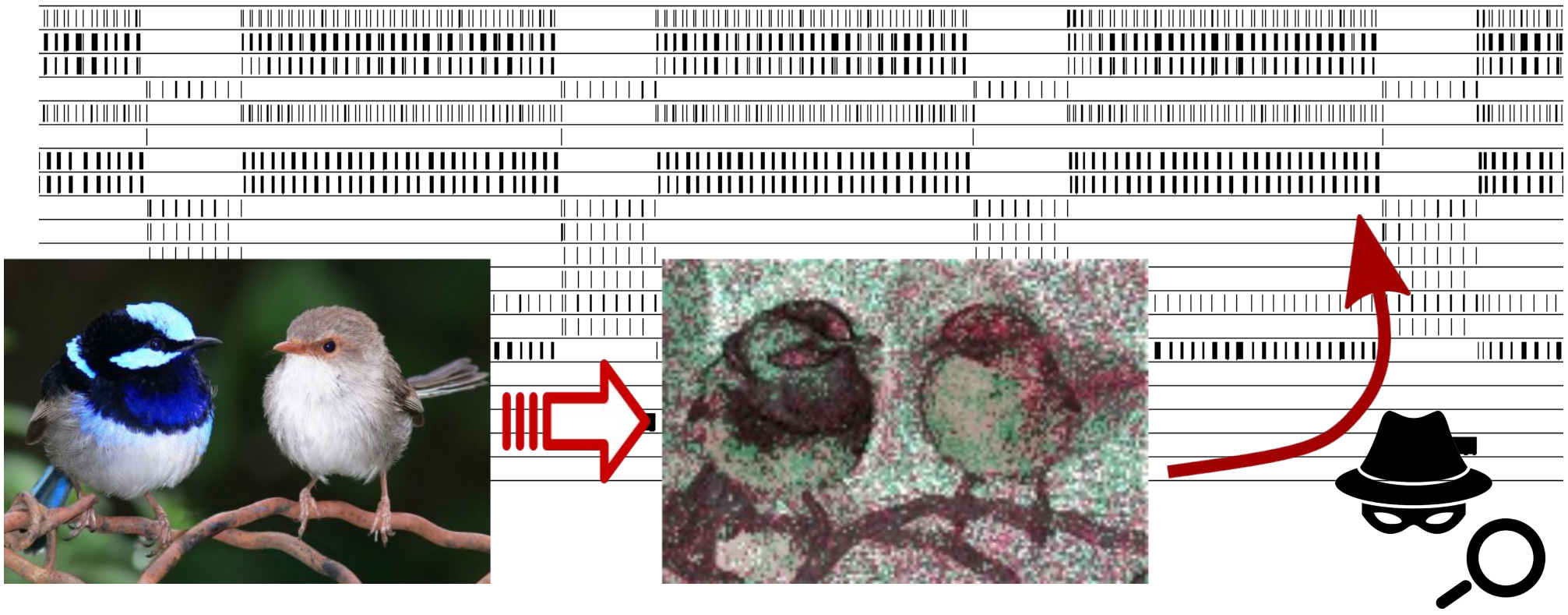
Spatial Resolution: Page-Granular Memory Access Traces



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

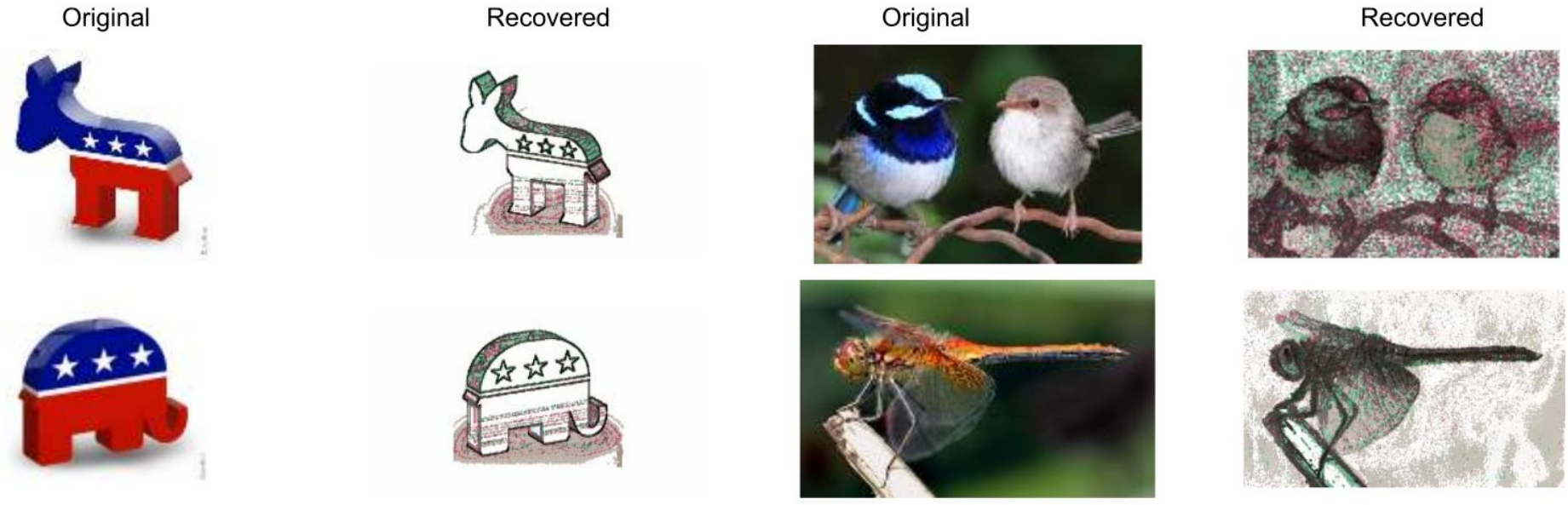
Vanoverloop et. al “TLBlur: Compiler-Assisted Automated Hardening against Controlled Channels on Off-the-Shelf Intel SGX Platforms”, USENIX’25.

Spatial Resolution: Page-Granular Memory Access Traces



Vanoverloop et. al “TLBlur: Compiler-Assisted Automated Hardening against Controlled Channels on Off-the-Shelf Intel SGX Platforms”, USENIX’25.

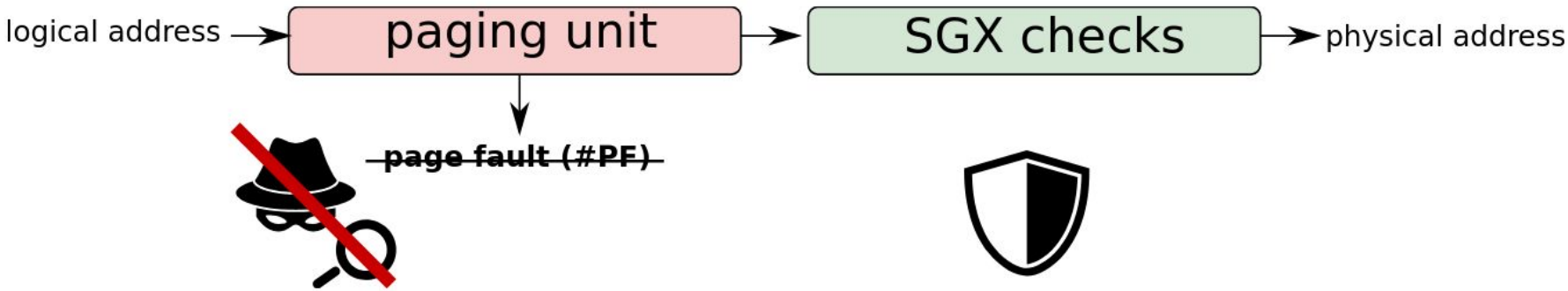
Spatial Resolution: Page-Granular Memory Access Traces



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

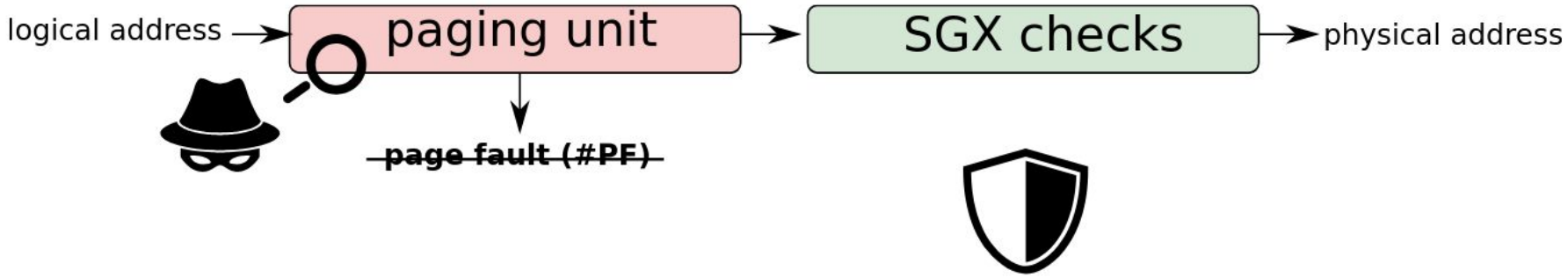
. . . but **many faults** and a coarse-grained **4 KiB granularity**

Naive Solutions: Hiding Enclave Page Faults



- Shih et al. "T-SGX: Eradicating controlled-channel attacks against enclave programs", NDSS 2017.
- Shinde et al. "Preventing page faults from telling your secrets", AsiaCCS 2016.

Naive Solutions: Hiding Enclave Page Faults



. . . But stealthy attacker can learn page visits without triggering faults!

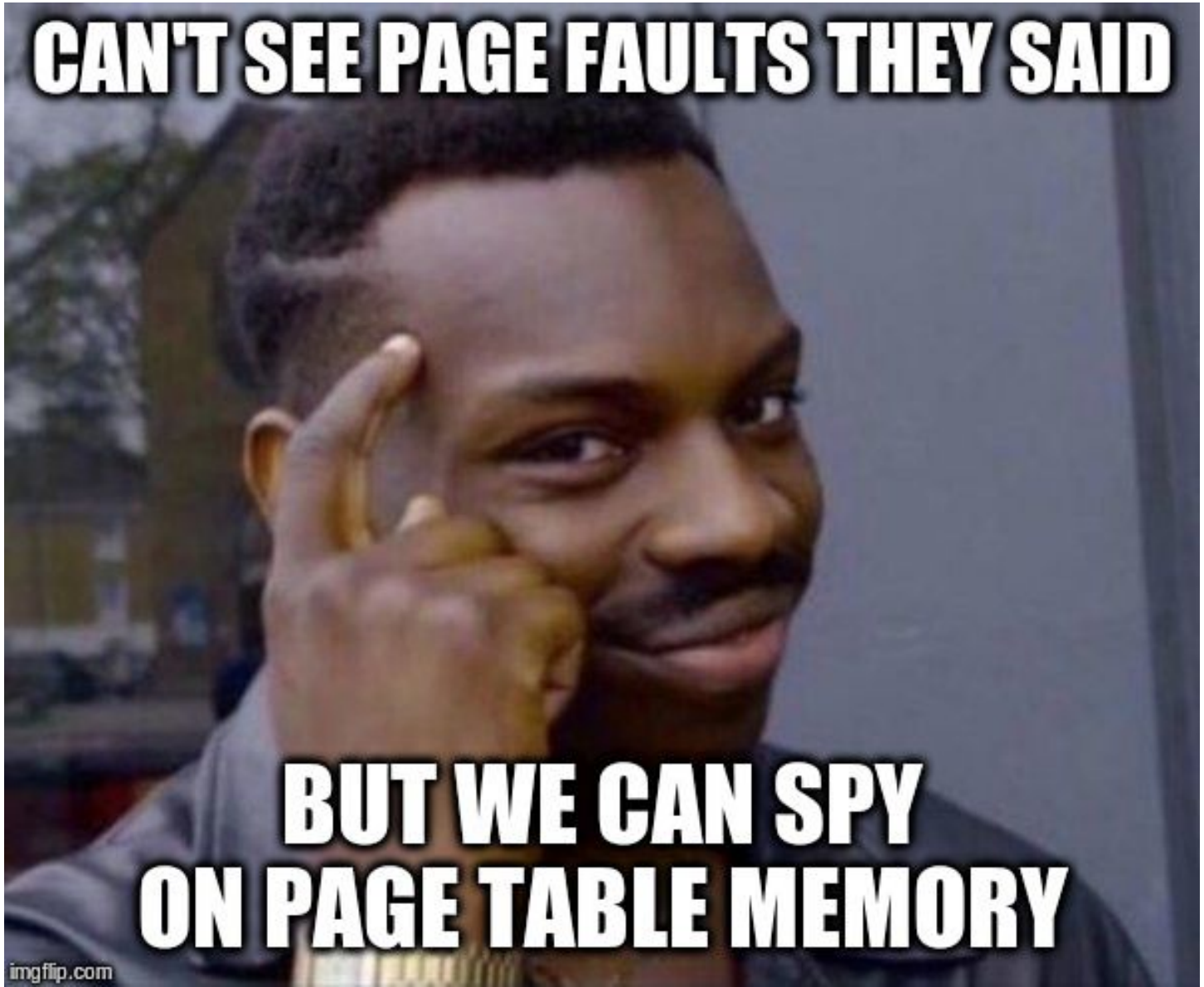
Documented Side-Effects of Address Translation

4.8 ACCESSED AND DIRTY FLAGS

For any paging-structure entry that is used during linear-address translation, bit 5 is the **accessed** flag.[2] For paging-structure entries that map a page (as opposed to referencing another paging structure), bit 6 is the **dirty** flag. These flags are provided for use by memory-management software to manage the transfer of pages and paging structures into and out of physical memory.

Whenever the processor uses a paging-structure entry as part of linear-address translation, it sets the accessed flag in that entry (if it is not already set).

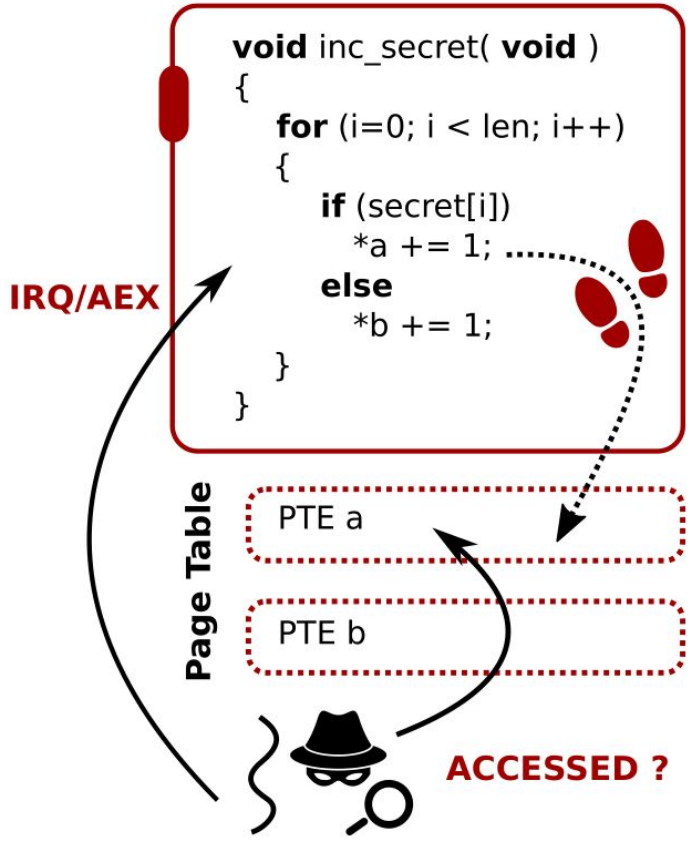
Whenever there is a write to a linear address, the processor sets the dirty flag (if it is not already set) in the paging-structure entry that identifies the final physical address for the linear address (either a PTE or a paging-structure entry in which the PS flag is 1).


CAN'T SEE PAGE FAULTS THEY SAID
BUT WE CAN SPY
ON PAGE TABLE MEMORY
imgflip.com

Telling your Secrets without Page Faults

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!



Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017.

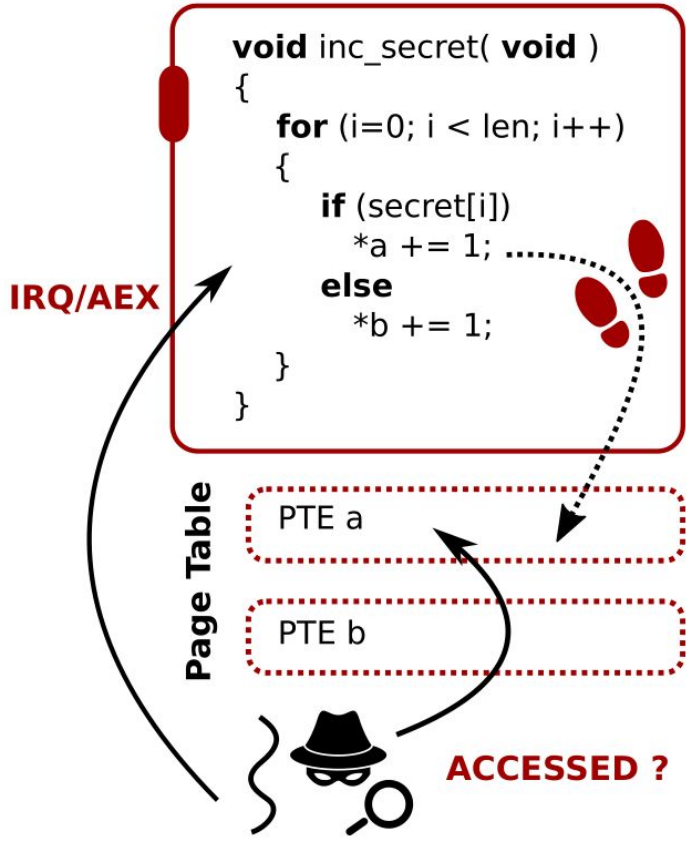
Telling your Secrets without Page Faults

1. Attack vector: PTE **status flags**:
   - A(ccessed) bit
   - D(irty) bit

   ⇝ Also updated in enclave mode!

2. Attack vector: Unprotected **page table memory**:
   - Cached as regular data
   - Accessed during address translation

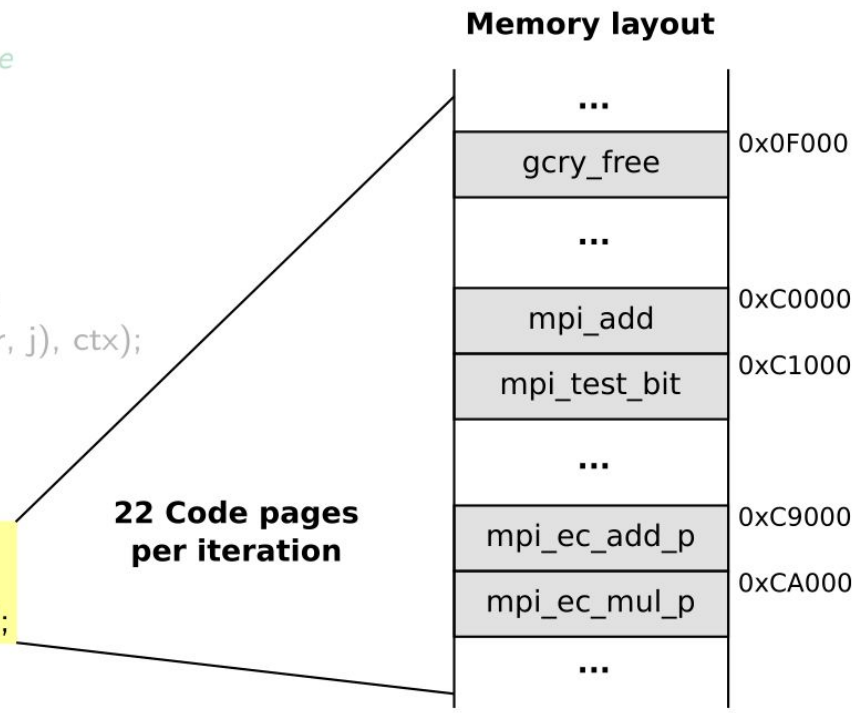
   ⇝ Flush+Reload cache timing attack!



Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017.

Attacking Libgcrypt EdDSA (Simplified)

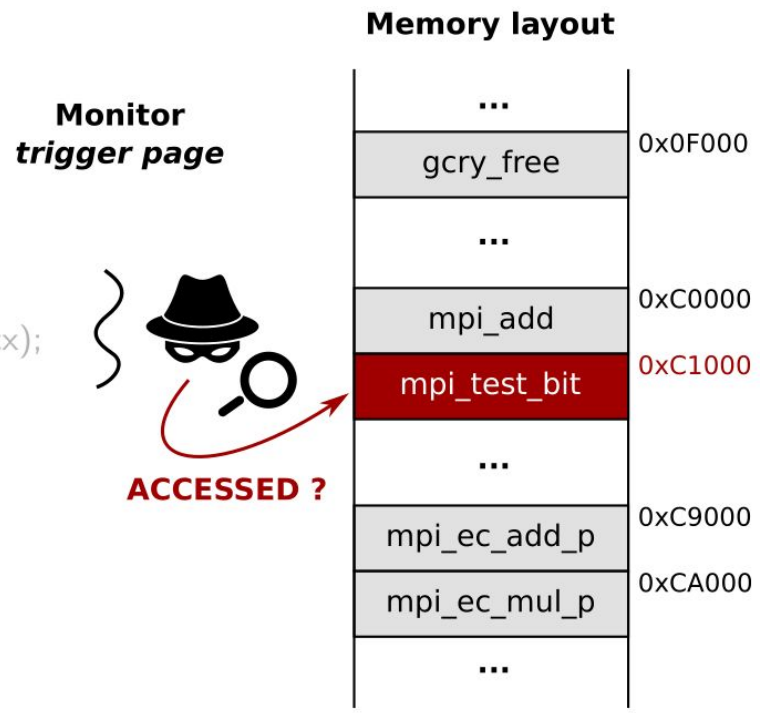
```
1  if (mpi_is_secure (scalar)) {
2      /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4      point_init (&tmppnt);
5
6      for (j=nbits-1; j >= 0; j--) {
7          _gcry_mpi_ec_dup_point (result, result, ctx);
8          _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
9          point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
10     }
11     point_free (&tmppnt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         _gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             _gcry_mpi_ec_add_points (result, result, point, ctx);
17     }
18 }
```



Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017.

Attacking Libgcrypt EdDSA (Simplified)

```
1  if (mpi_is_secure (scalar)) {
2      /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4      point_init (&tmppnt);
5
6      for (j=nbits-1; j >= 0; j--) {
7          _gcry_mpi_ec_dup_point (result, result, ctx);
8          _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
9          point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
10     }
11     point_free (&tmppnt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         _gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             _gcry_mpi_ec_add_points (result, result, point, ctx);
17     }
18 }
```



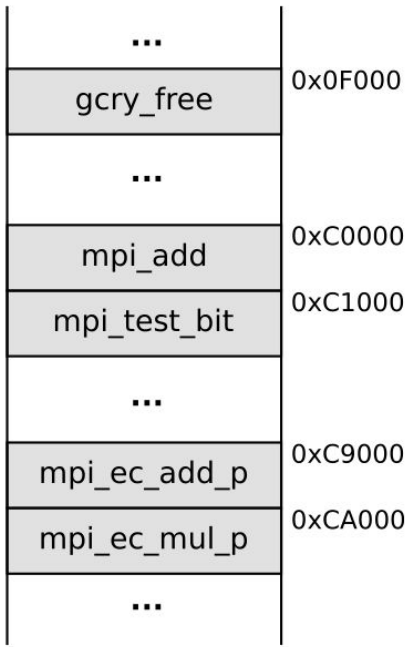
Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017.

Attacking Libgcrypt EdDSA (Simplified)

```
1  if (mpi_is_secure (scalar)) {
2      /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4      point_init (&tmppnt);
5
6      for (j=nbits-1; j >= 0; j--) {
7          _gcry_mpi_ec_dup_point (result, result, ctx);
8          _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
9          point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
10     }
11     point_free (&tmppnt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         _gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             _gcry_mpi_ec_add_points (result, result, point, ctx);
17     }
18 }
```

INTERRUPT

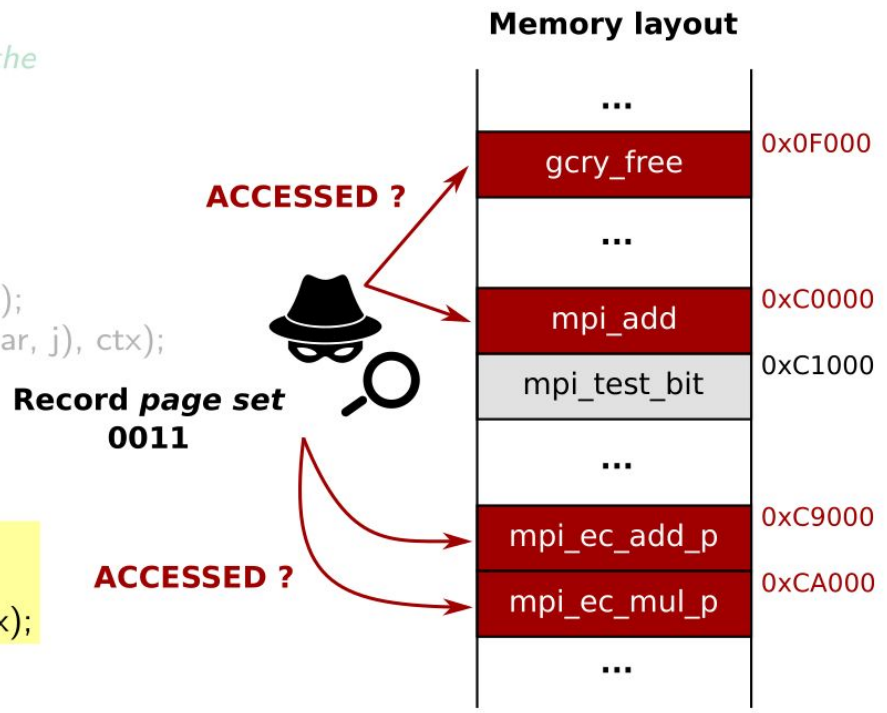
Memory layout



Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017.

Attacking Libgcrypt EdDSA (Simplified)

```
1  if (mpi_is_secure (scalar)) {
2      /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4      point_init (&tmppnt);
5
6      for (j=nbits-1; j >= 0; j--) {
7          _gcry_mpi_ec_dup_point (result, result, ctx);
8          _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
9          point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
10     }
11     point_free (&tmppnt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         _gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             _gcry_mpi_ec_add_points (result, result, point, ctx);
17     }
18 }
```



Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017.

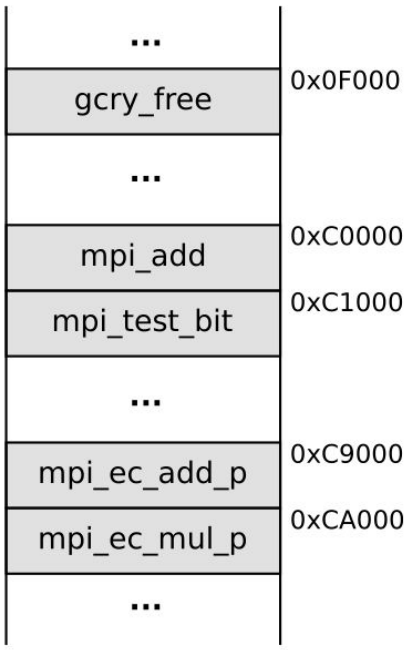
Attacking Libgcrypt EdDSA (Simplified)

```
1  if (mpi_is_secure (scalar)) {
2      /* If SCALAR is in secure memory we assume that it is the
3         secret key we use constant time operation. */
4      point_init (&tmppnt);
5
6      for (j=nbits-1; j >= 0; j--) {
7          _gcry_mpi_ec_dup_point (result, result, ctx);
8          _gcry_mpi_ec_add_points (&tmppnt, result, point, ctx);
9          point_swap_cond (result, &tmppnt, mpi_test_bit (scalar, j), ctx);
10     }
11     point_free (&tmppnt);
12 } else {
13     for (j=nbits-1; j >= 0; j--) {
14         _gcry_mpi_ec_dup_point (result, result, ctx);
15         if (mpi_test_bit (scalar, j))
16             _gcry_mpi_ec_add_points (result, result, point, ctx);
17     }
18 }
```

RESUME

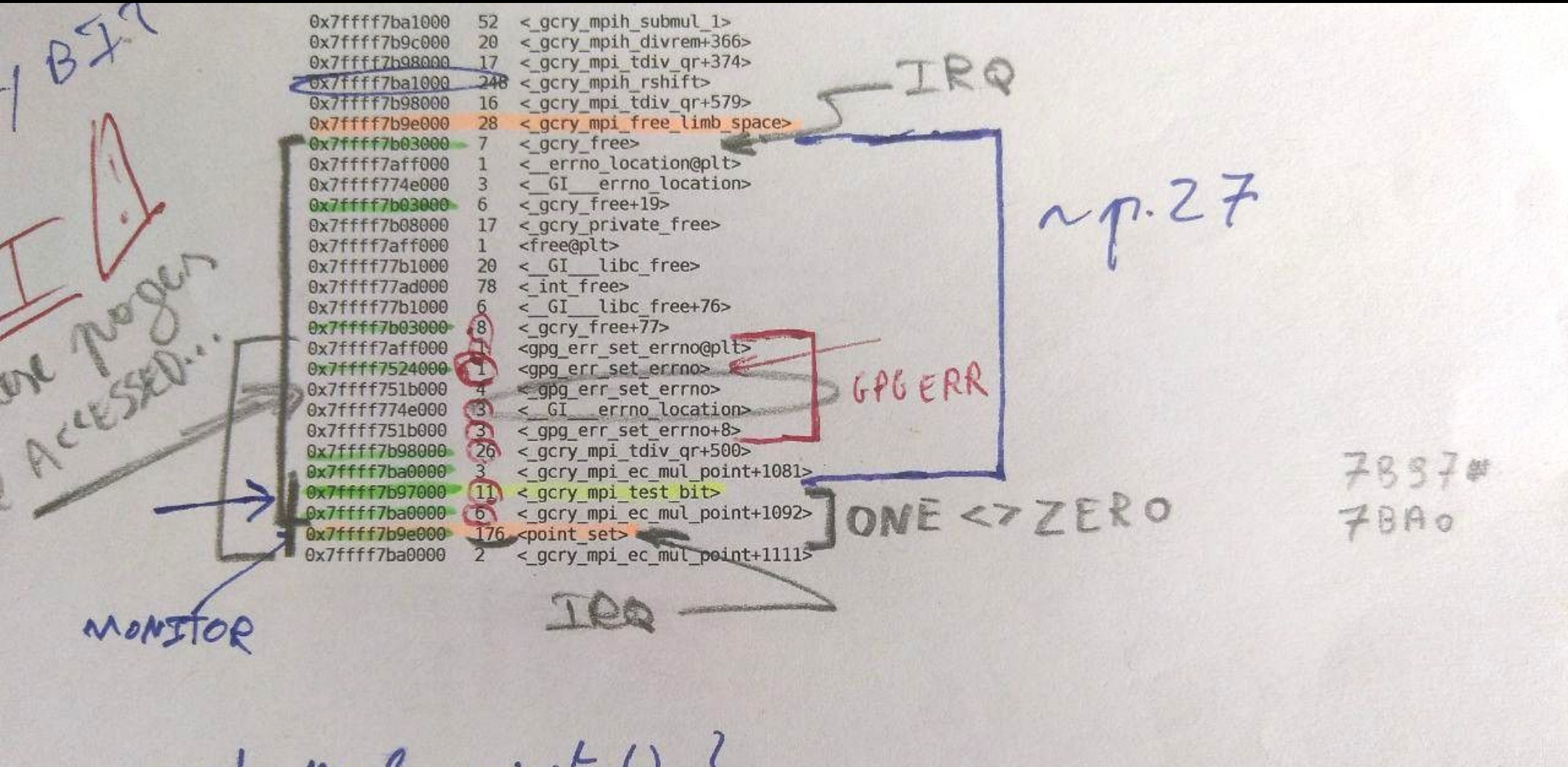
Full 512-bit key recovery, single run

Memory layout



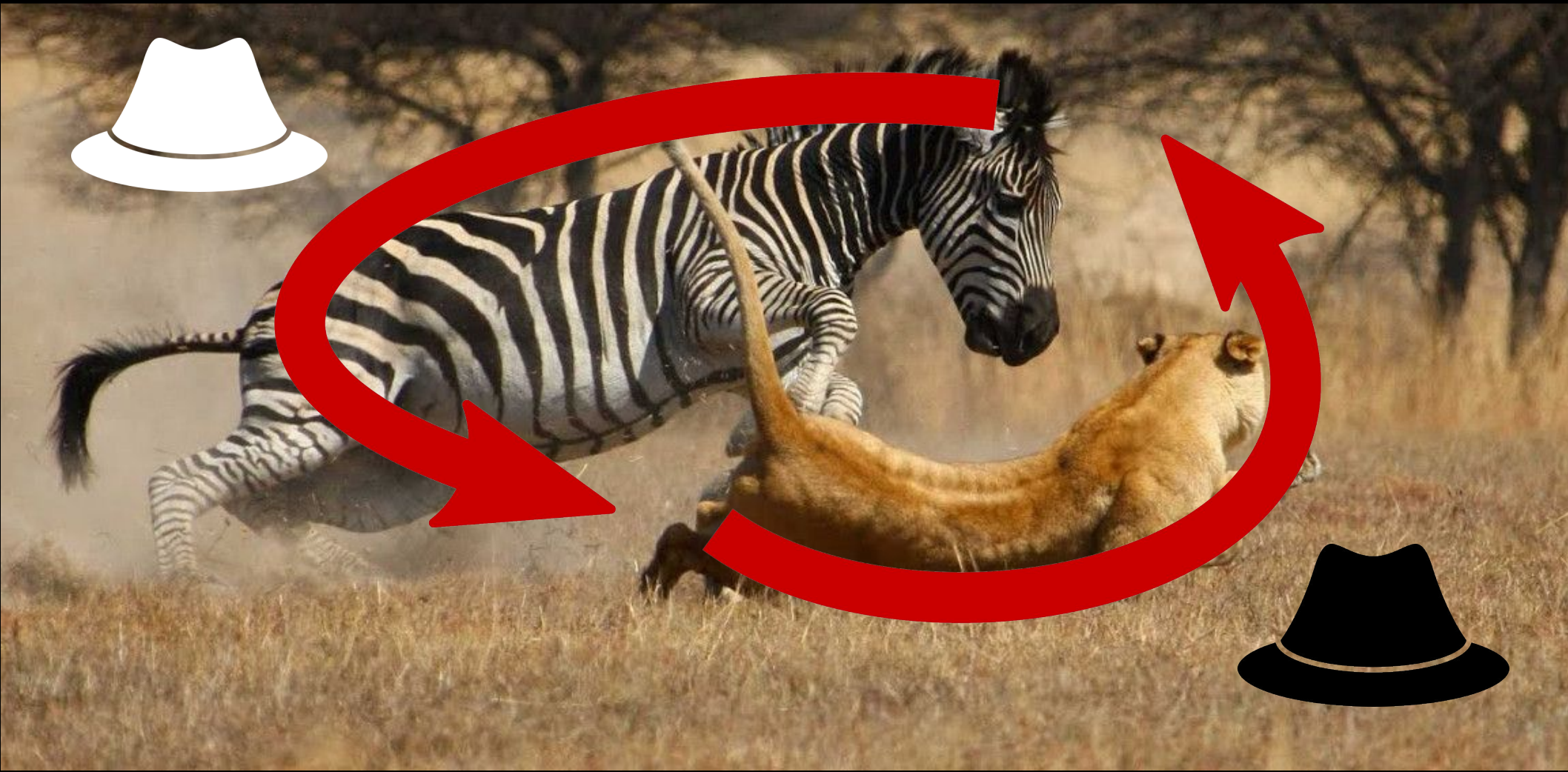
Van Bulck et al. "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution", USENIX 2017.

Side-channel Analysis: From Metadata Patterns to Secrets

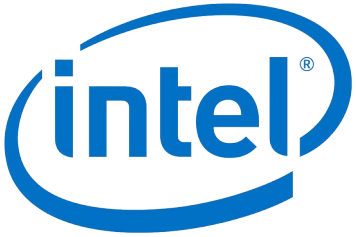


Scientific Understanding Driven by Attacker-Defender Race...

Scientific Understanding Driven by Attacker-Defender Race...



Idea #2: Improved Temporal Resolution with Timer Interrupts



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

```c
size_t strlen (char *str)
{
  char *s;

  for (s = str; *s; ++s);
  return (s - str);
}
```

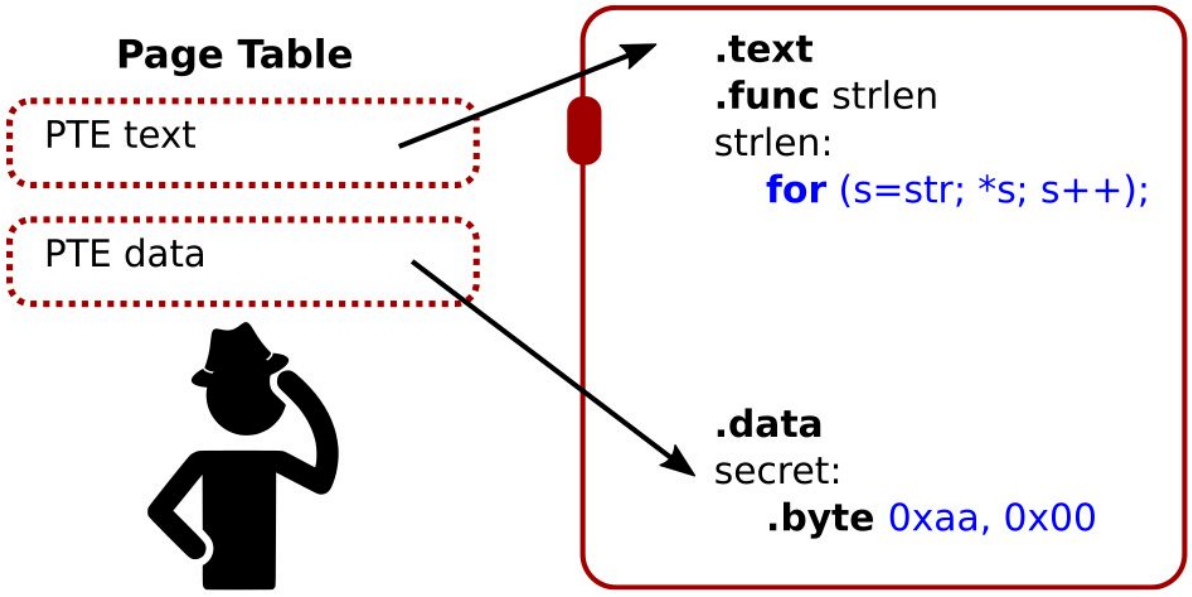
```
    mov  %rdi,%rax
1:  cmpb $0x0,(%rax)
    je   2f
    inc  %rax
    jmp  1b
2:  sub  %rdi,%rax
    retq
```

$\Rightarrow$ 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

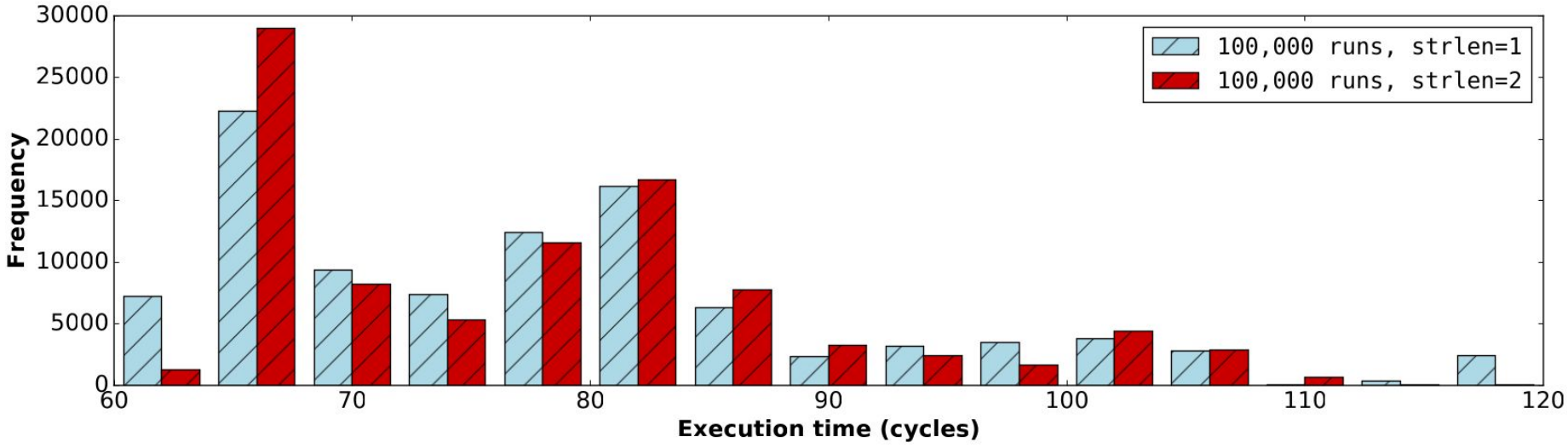


Counting `strlen` loop iterations?

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. . .



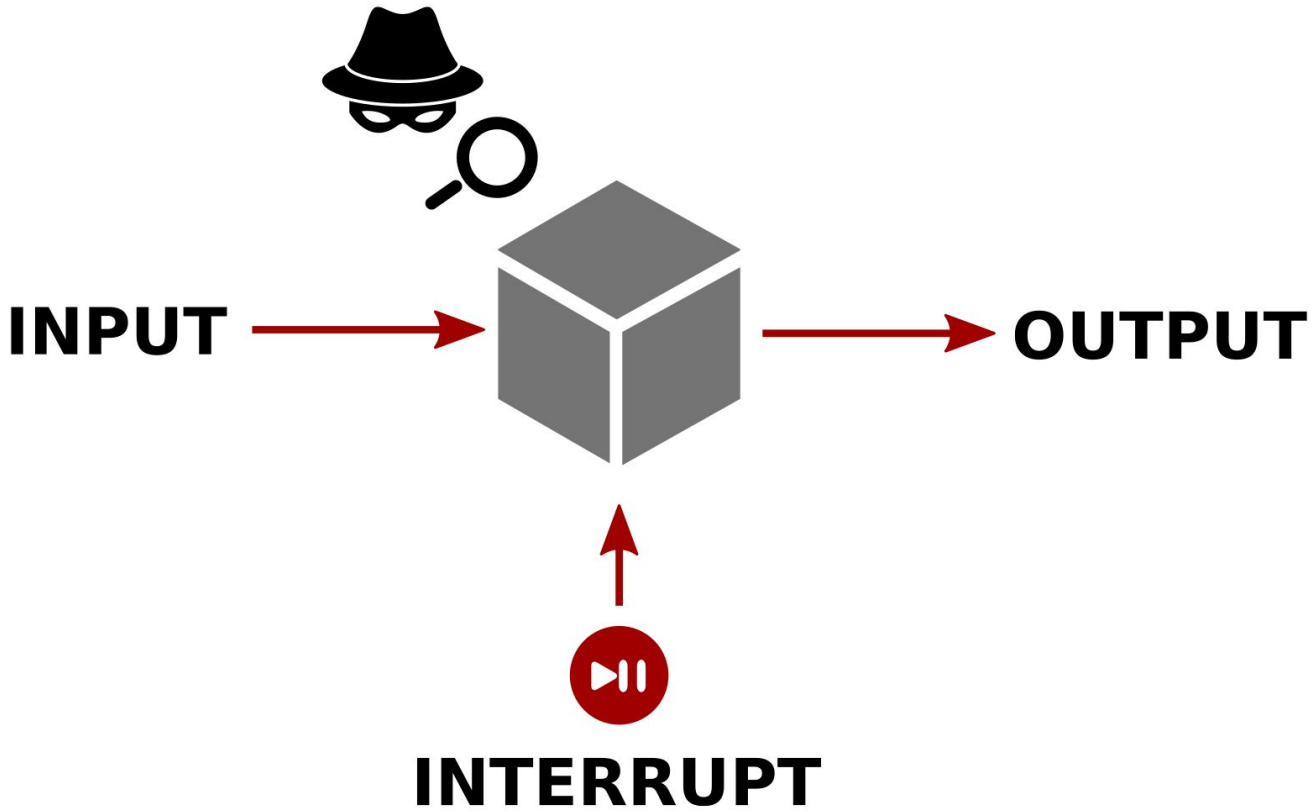
Challenge: Side-Channel Sampling Rate

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

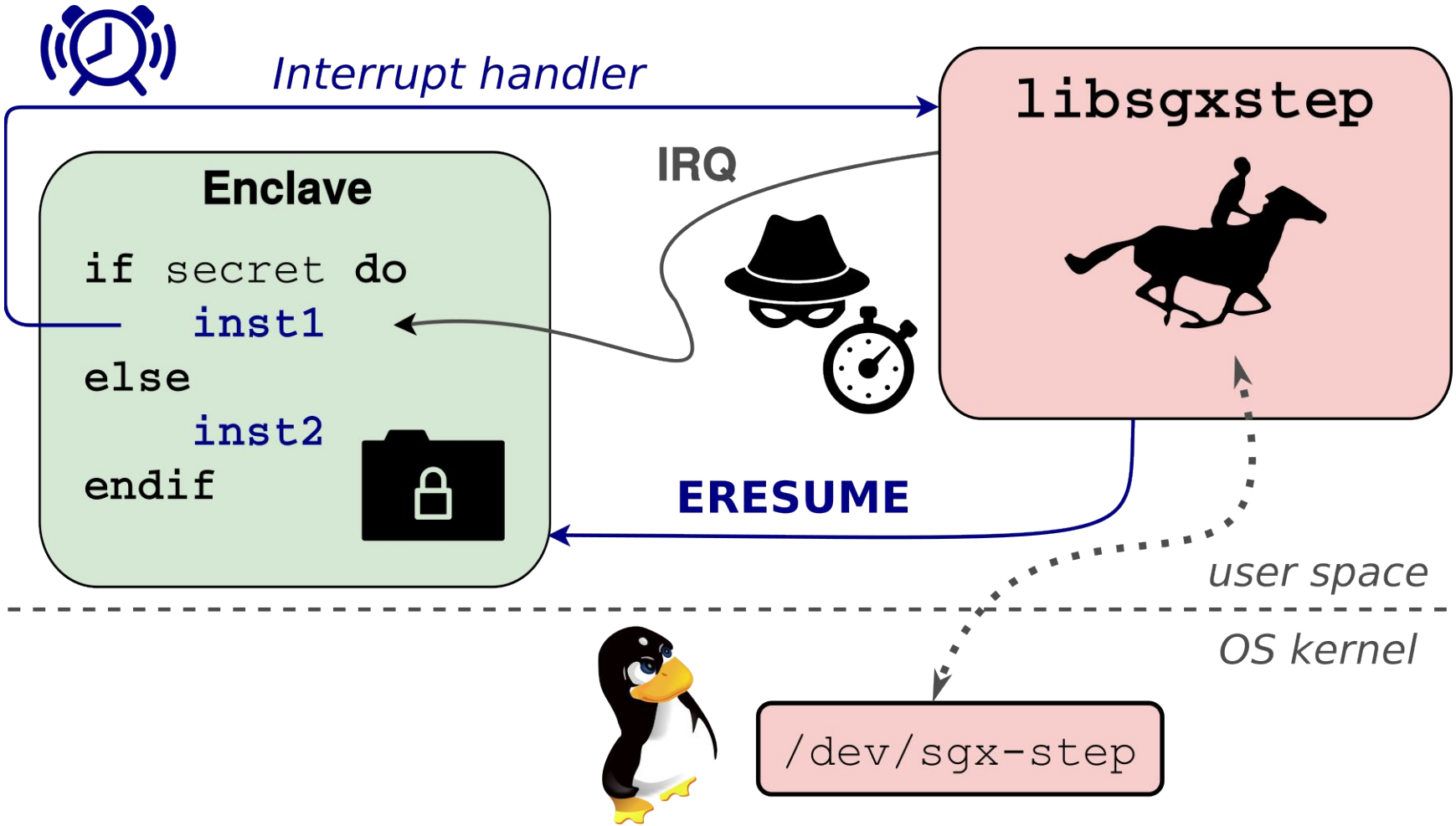
ACSAC 2023
Cybersecurity Artifacts
Impact Award

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

Watch	22	Star	245	Fork	52

Van Bulck et al., “SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control”, SysTEX 2017.

SGX-Step: Executing Enclaves one Instruction at a Time



SGX-Step Demo: Single-Stepping Password Comparison

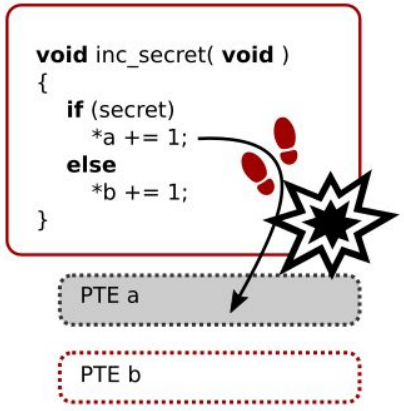
```
jo@breuer:~/sgx-step-demo$ sudo ./app
```

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	Kalyna 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)	✓

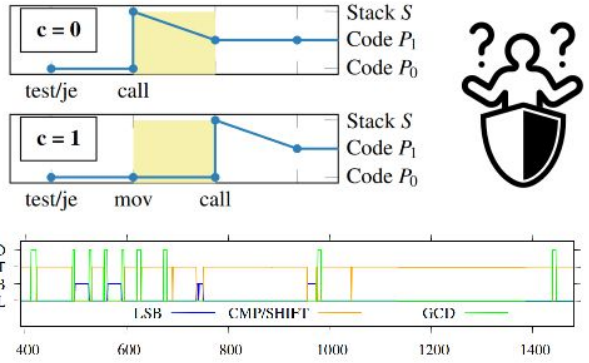
SGX-Step: A Versatile Open-Source Attack Framework



Interrupt latency

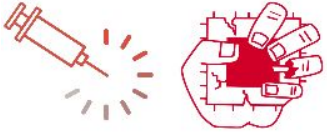
Page-table manipulation

SGX-Step



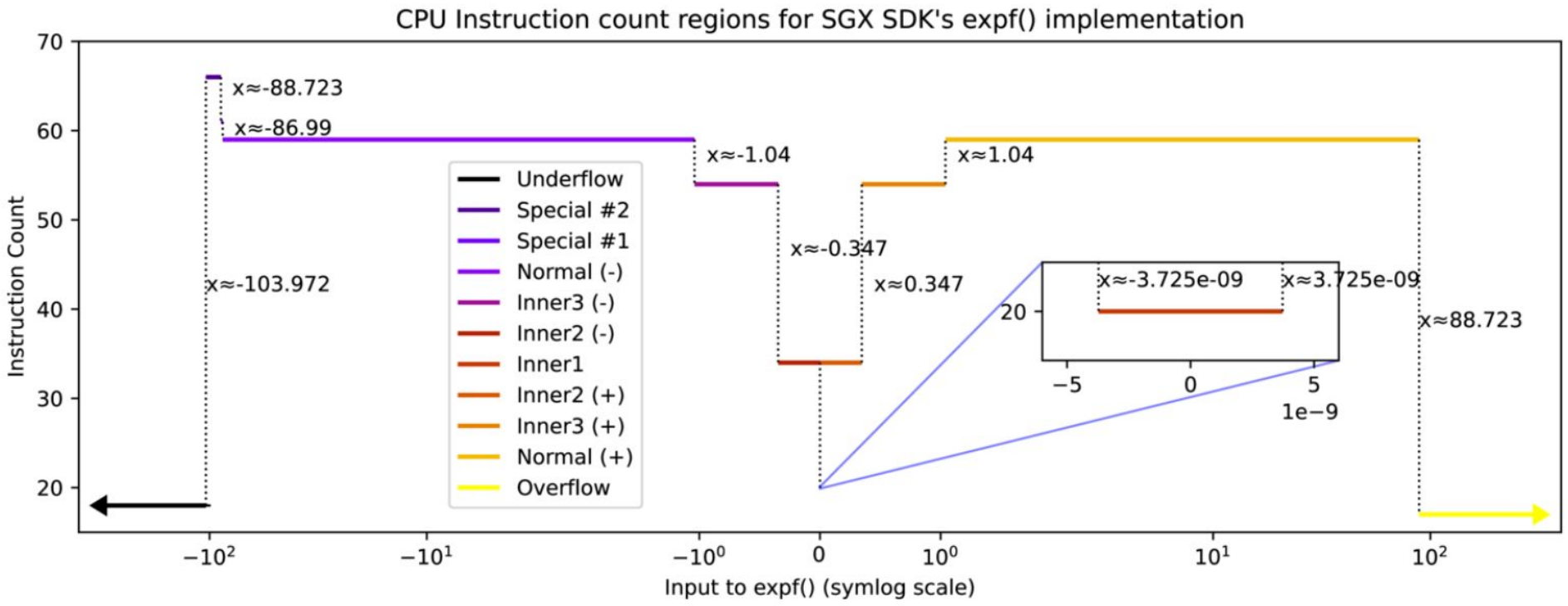
Interrupt counting

High-resolution probing



Zero-step replaying

Example: Deterministic Instruction Counting Attacks

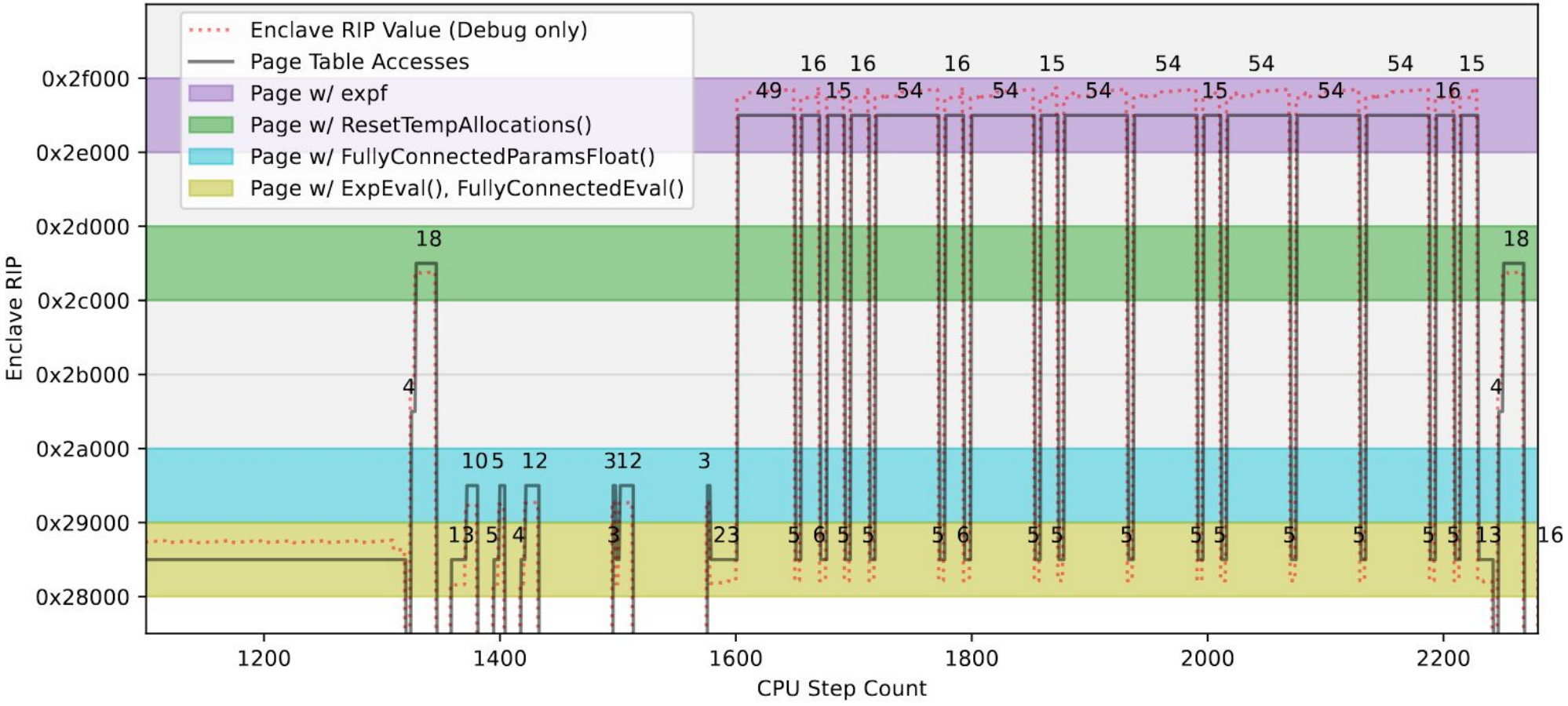


We can *count instructions!*

Inputs to `expf()` from 10^{-2} to 10^{2} fall into into **11 different count classes** (8 unique).

Spielman et al., "Activation Functions Considered Harmful: Recovering Neural Network Weights through Controlled Channels", RAID 2025.

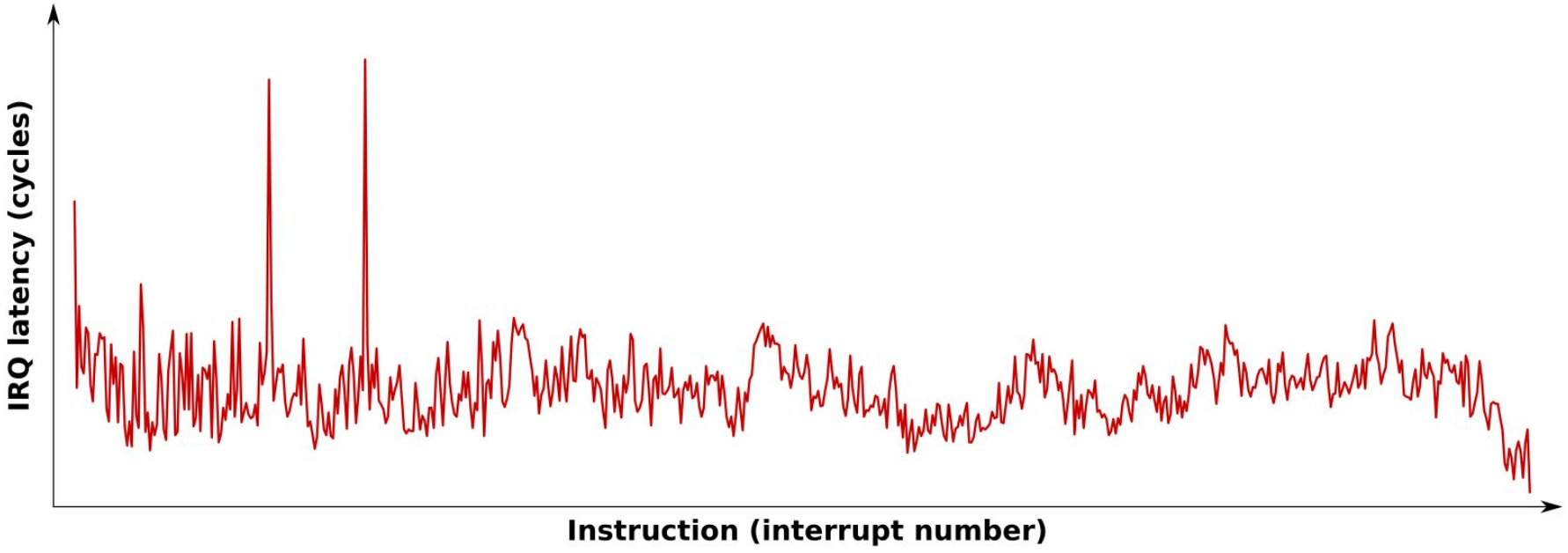
Example: Augmented Instruction-Level Page-Access Traces



Spielman et al., “Activation Functions Considered Harmful: Recovering Neural Network Weights through Controlled Channels”, RAID 2025.

Nemesis: Extracting IRQ Latency Traces with SGX-Step

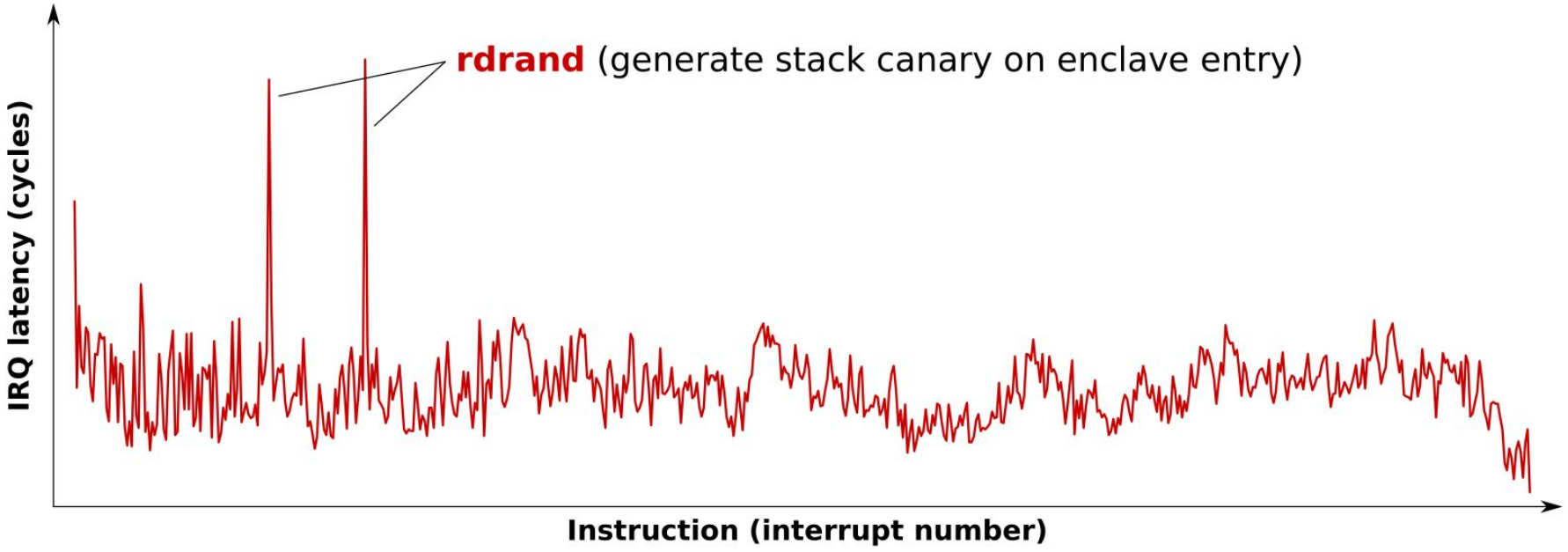
Enclave x-ray: IRQ latency leaks instruction-level μ-arch timing!



Van Bulck et al. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic", CCS 2018..

Nemesis: Extracting IRQ Latency Traces with SGX-Step

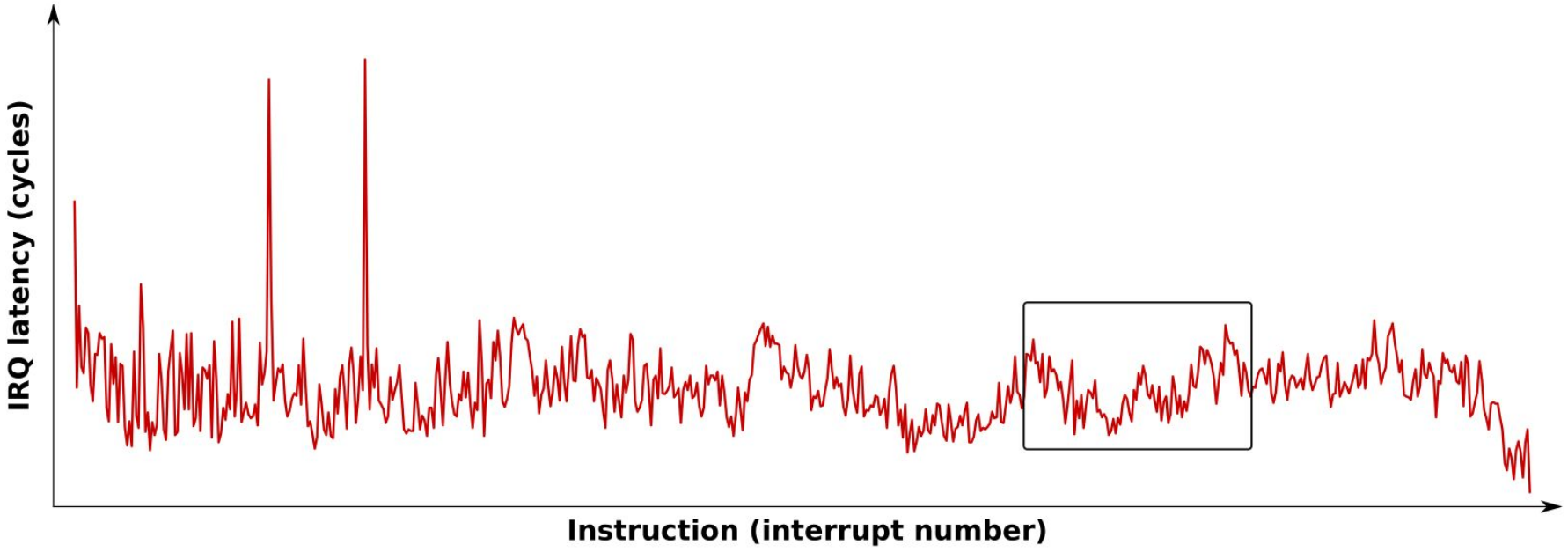
Enclave x-ray: Spotting high-latency instructions



Van Bulck et al. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic", CCS 2018..

Nemesis: Extracting IRQ Latency Traces with SGX-Step

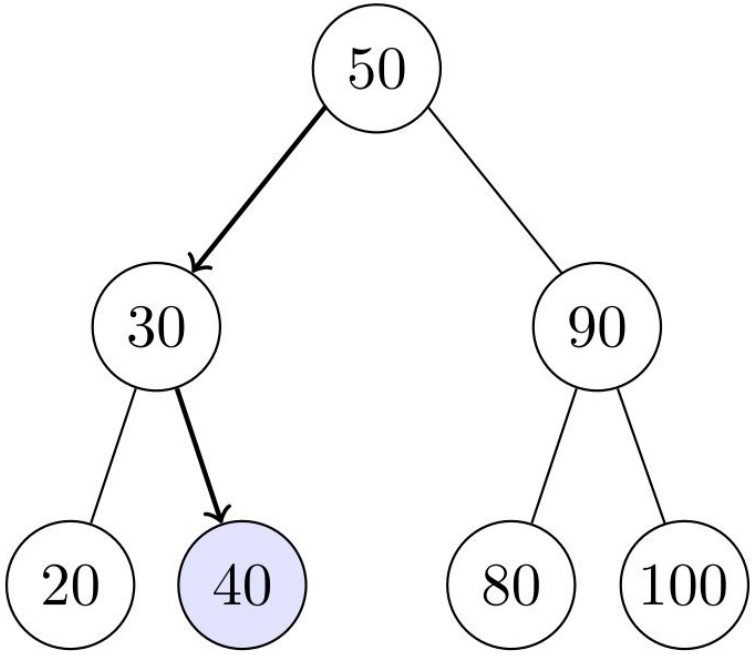
Enclave x-ray: Zooming in on `bsearch` function



Van Bulck et al. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic", CCS 2018..

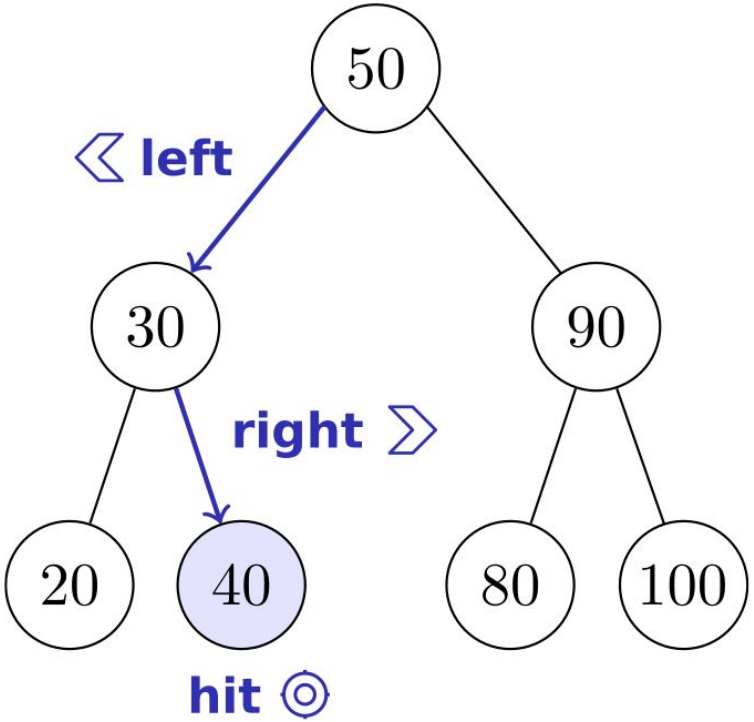
De-Anonymizing Enclave Lookups with Interrupt Latency

Binary search: Find 40 in {20, 30, 40, 50, 80, 90, 100}



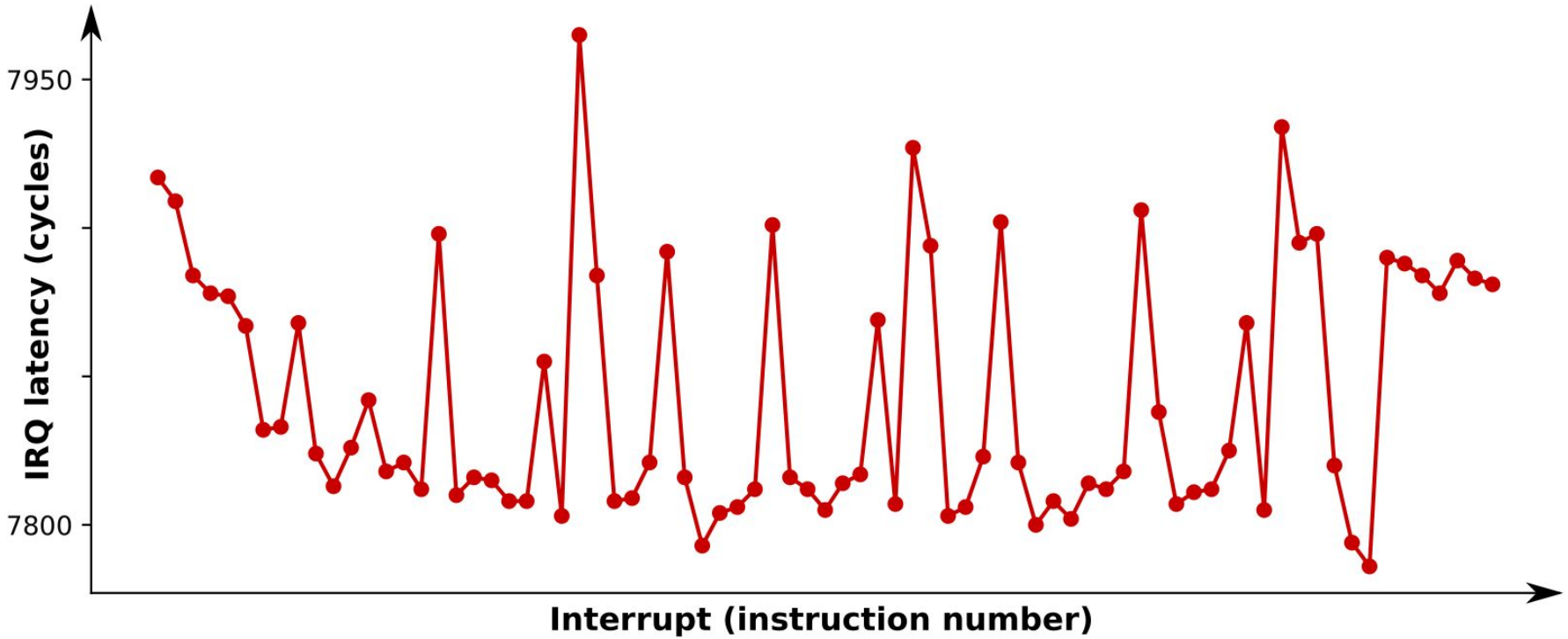
De-Anonymizing Enclave Lookups with Interrupt Latency

Adversary: Infer secret lookup in known array



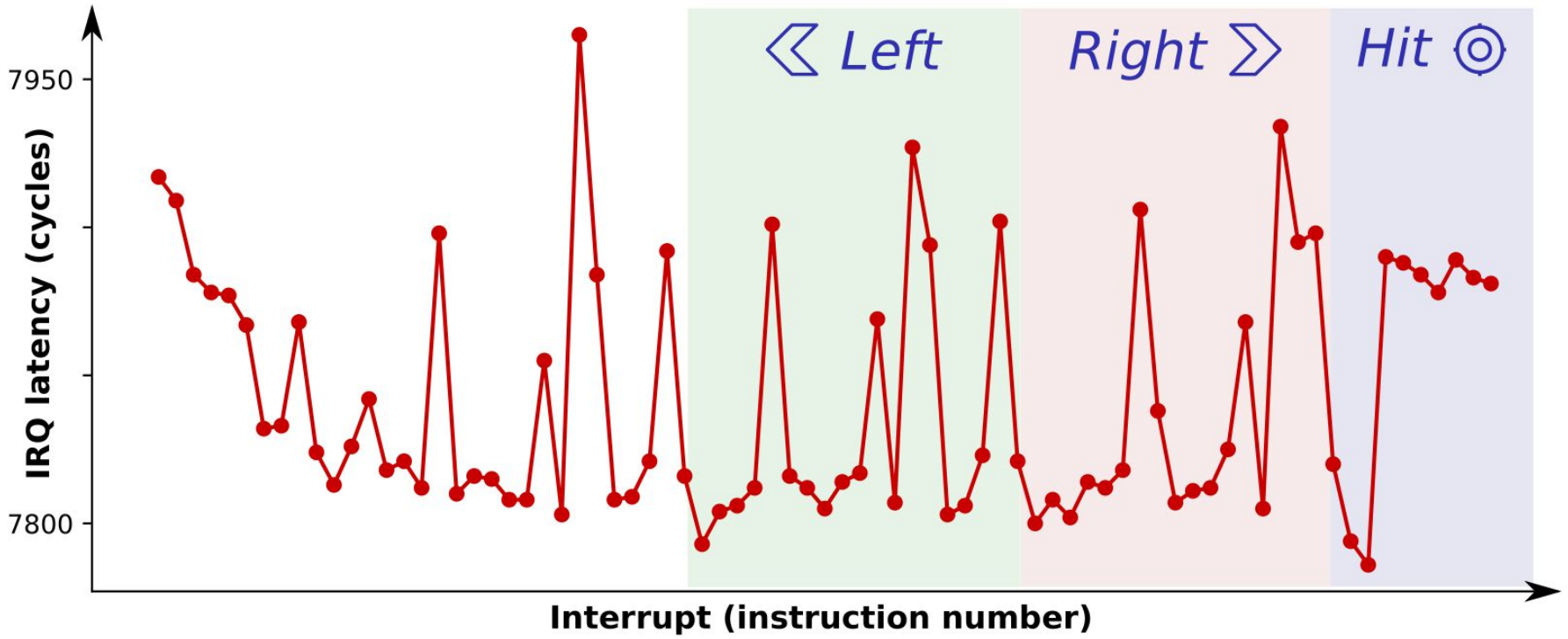
De-Anonymizing Enclave Lookups with Interrupt Latency

Goal: Infer lookup → reconstruct `bsearch` control flow



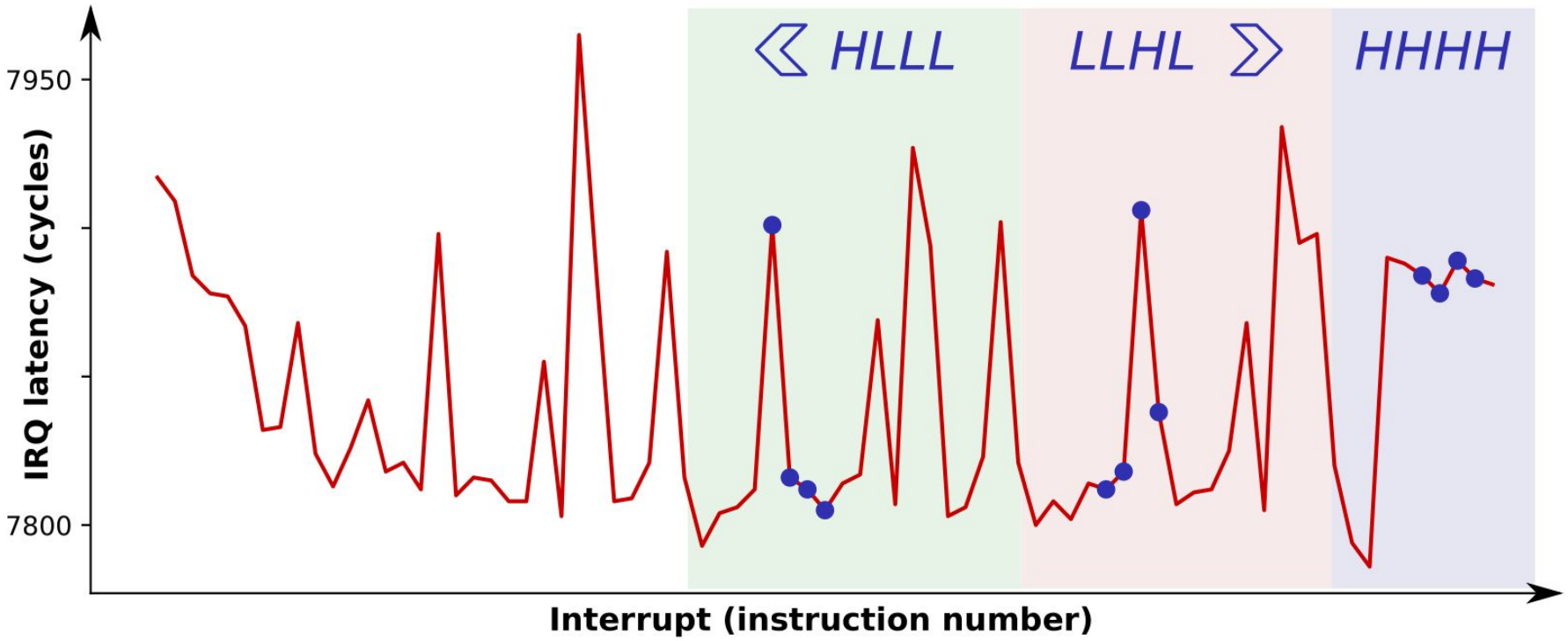
De-Anonymizing Enclave Lookups with Interrupt Latency

Goal: Infer lookup → reconstruct `bsearch` control flow

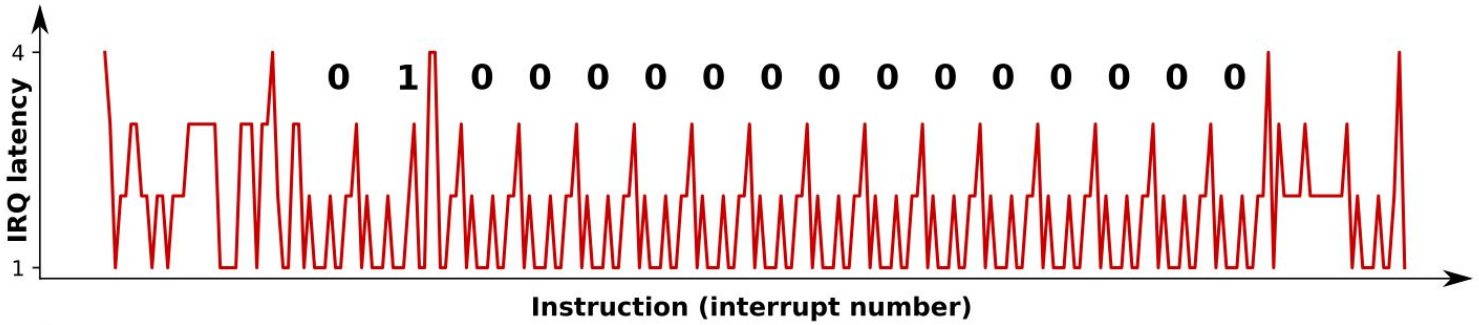


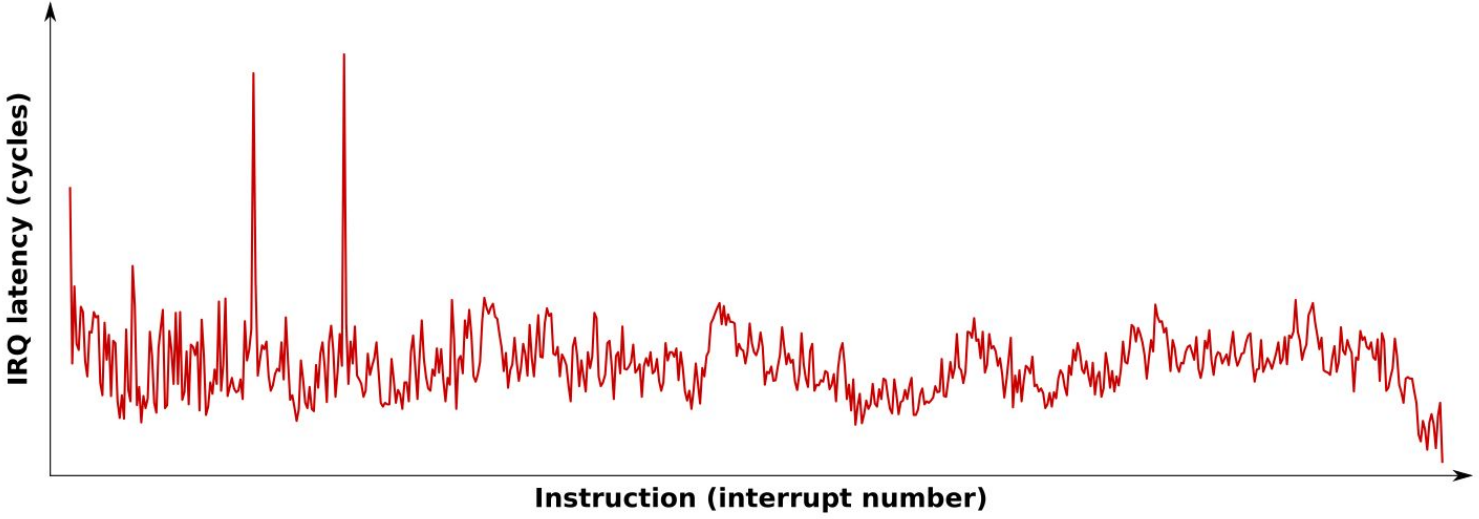
De-Anonymizing Enclave Lookups with Interrupt Latency

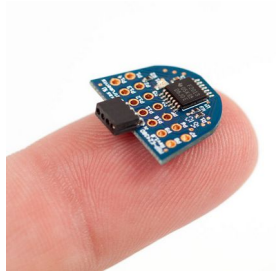
⇒ Sample **instruction latencies** in secret-dependent path



Interrupt-Latency Attacks Beyond Intel SGX





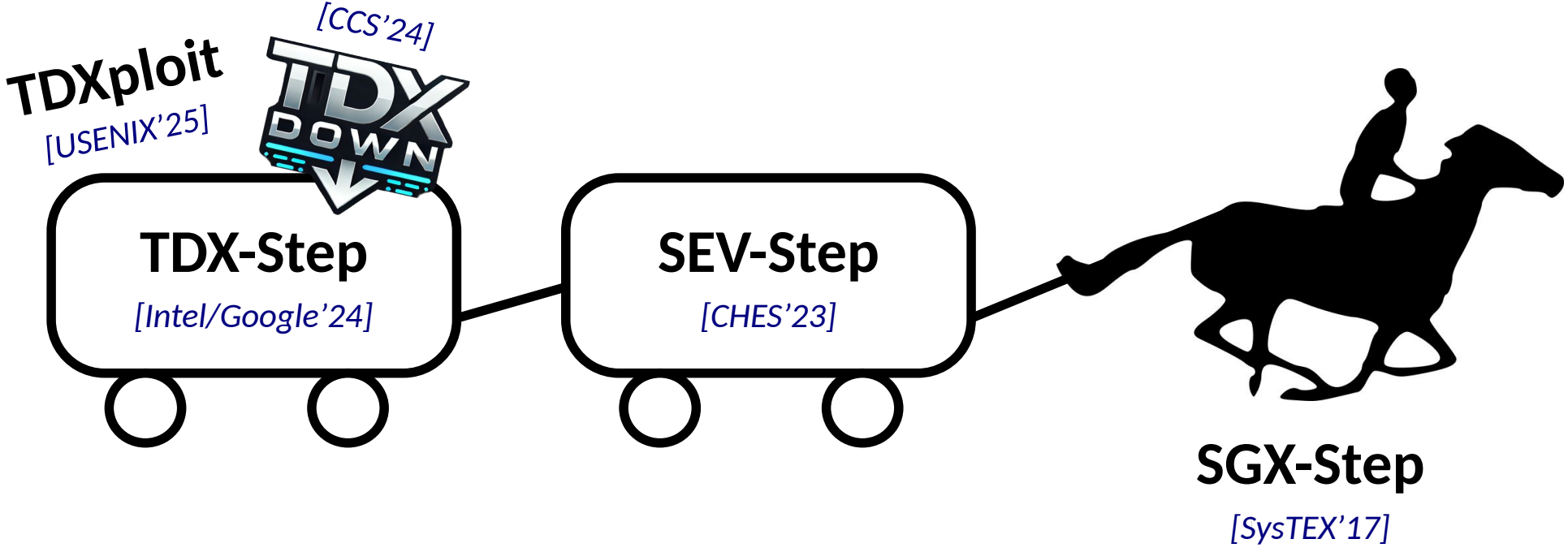


Texas Instruments

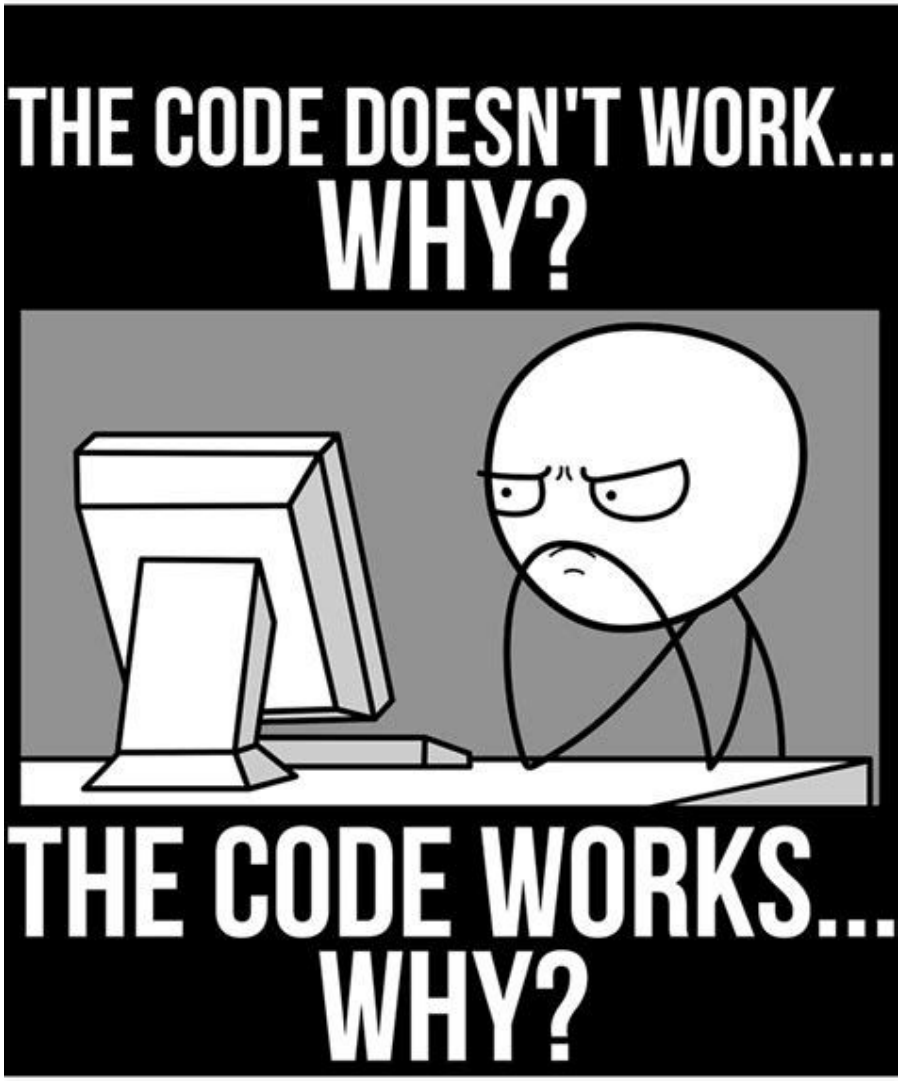
intel

Van Bulck et al. "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic", CCS 2018

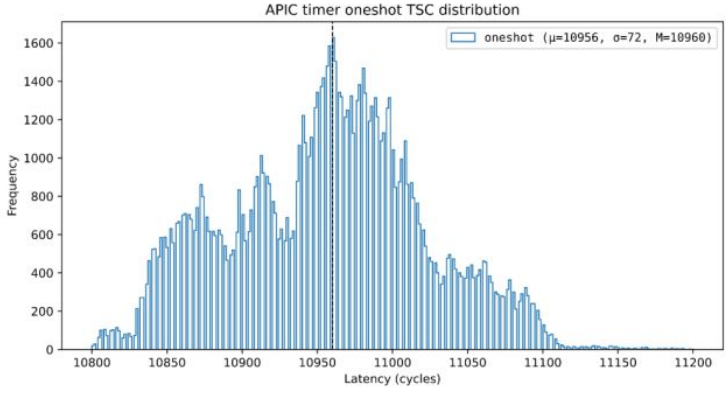
Single-Stepping Beyond Intel SGX

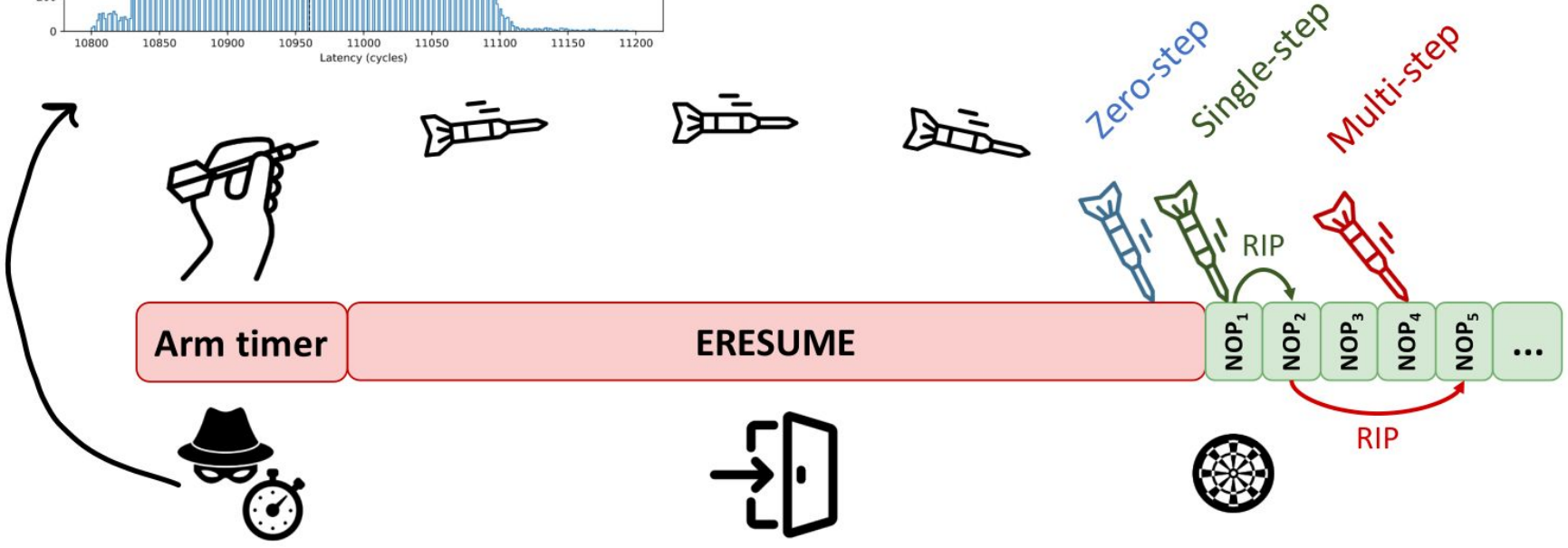


Based on slide from Luka Wilke.


THE CODE DOESN'T WORK...
WHY?
THE CODE WORKS...
WHY?

Root-Causing SGX-Step: Aiming the Timer Interrupt

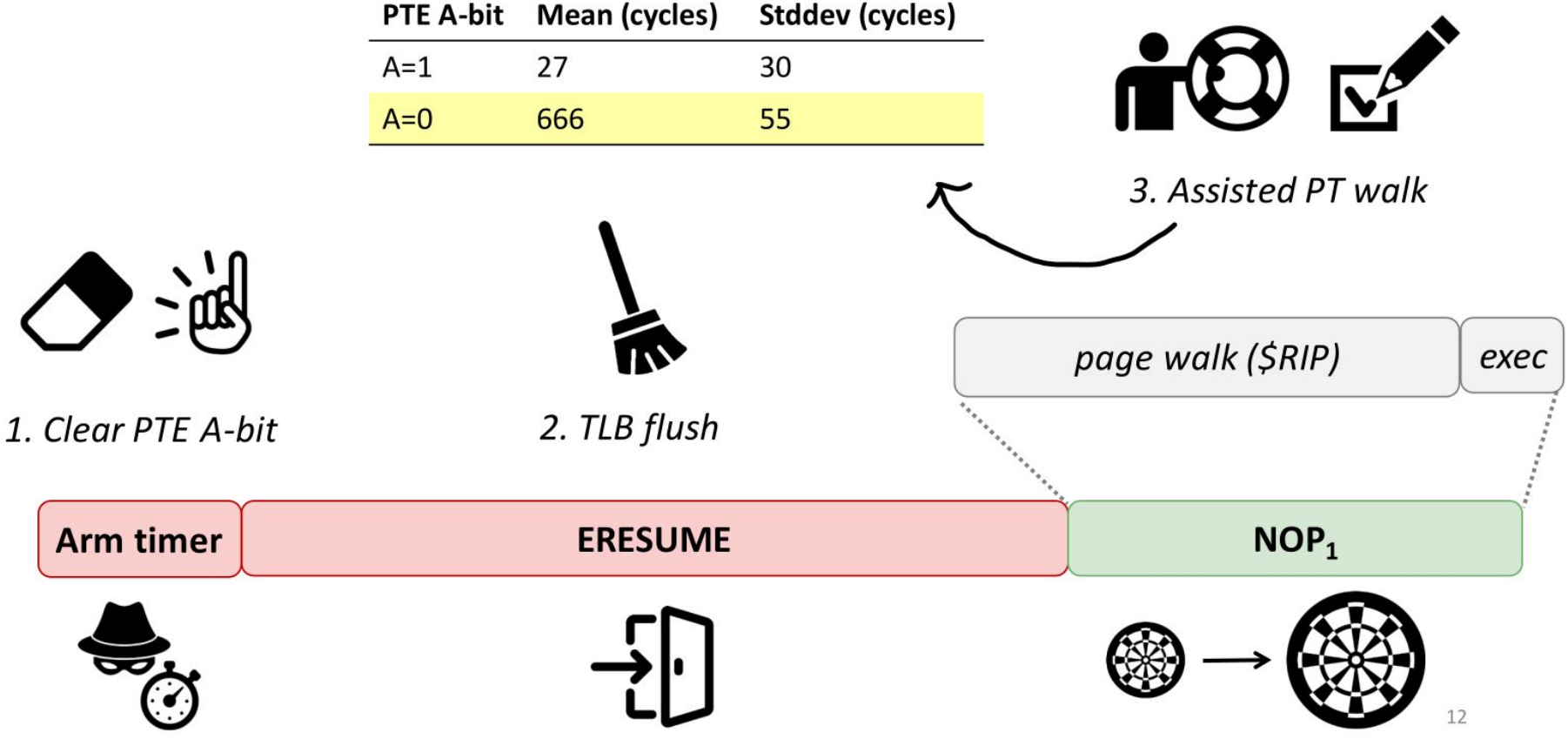




Constable et al. “AEX-Notify: Thwarting Precise Single-Stepping Attacks through Interrupt Awareness for Intel SGX Enclaves”, USENIX Security 2023.

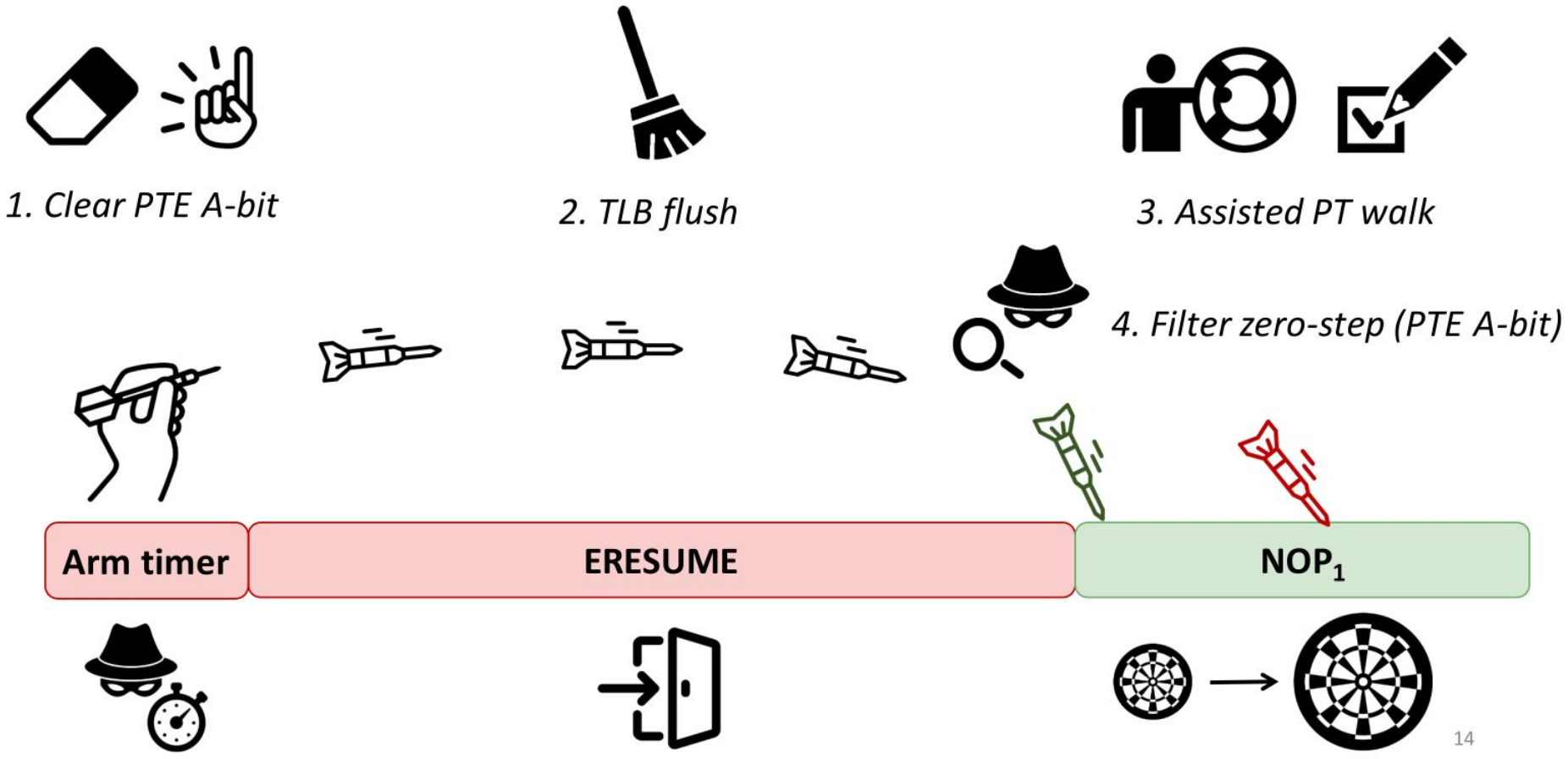
Root-Causing SGX-Step: Microcode Assists to the Rescue!

PTE A-bit	Mean (cycles)	Stddev (cycles)
A=1	27	30
A=0	666	55



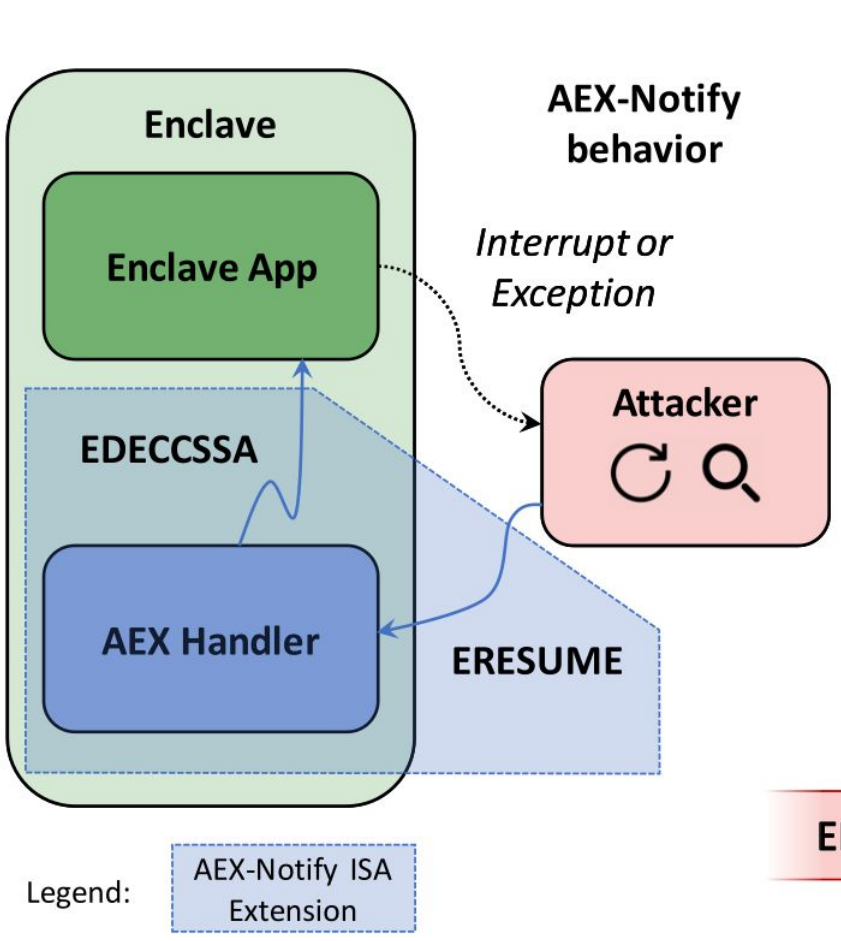
Constable et al. “AEX-Notify: Thwarting Precise Single-Stepping Attacks through Interrupt Awareness for Intel SGX Enclaves”, USENIX Security 2023.

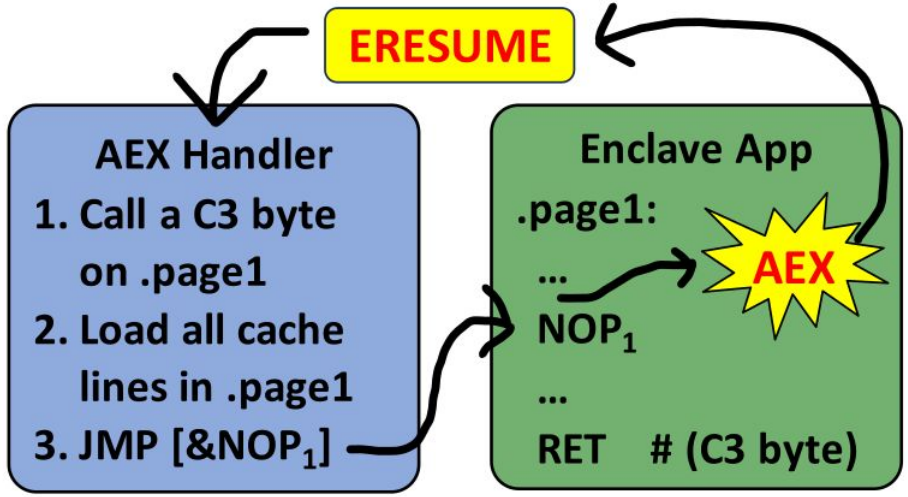
Root-Causing SGX-Step: Microcode Assists to the Rescue!

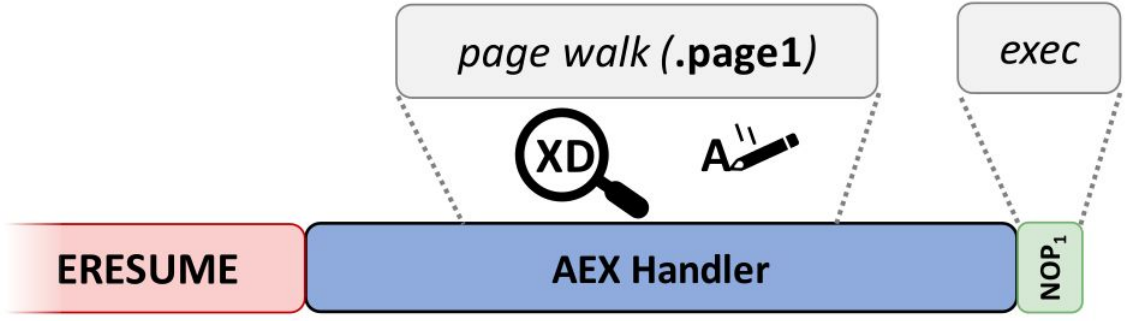


Constable et al. “AEX-Notify: Thwarting Precise Single-Stepping Attacks through Interrupt Awareness for Intel SGX Enclaves”, USENIX Security 2023.

AEX-Notify: Hardware-Software Co-Design Solution



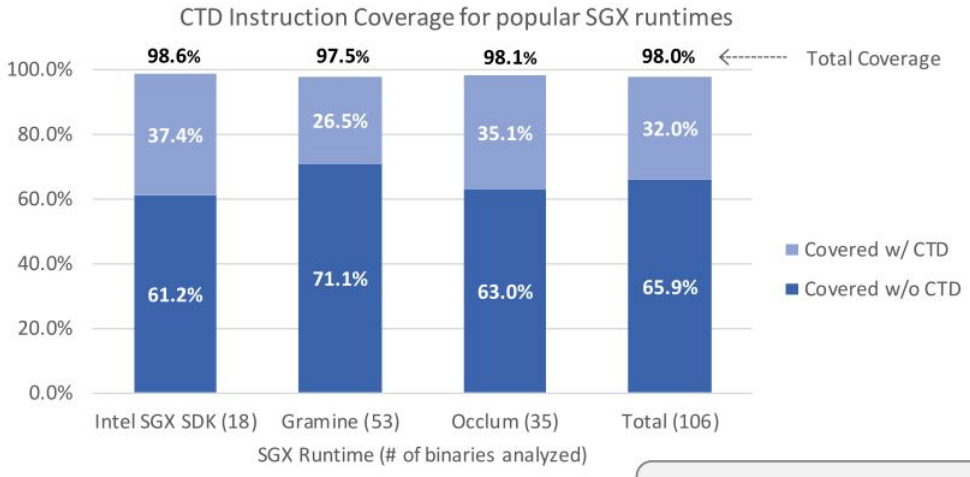


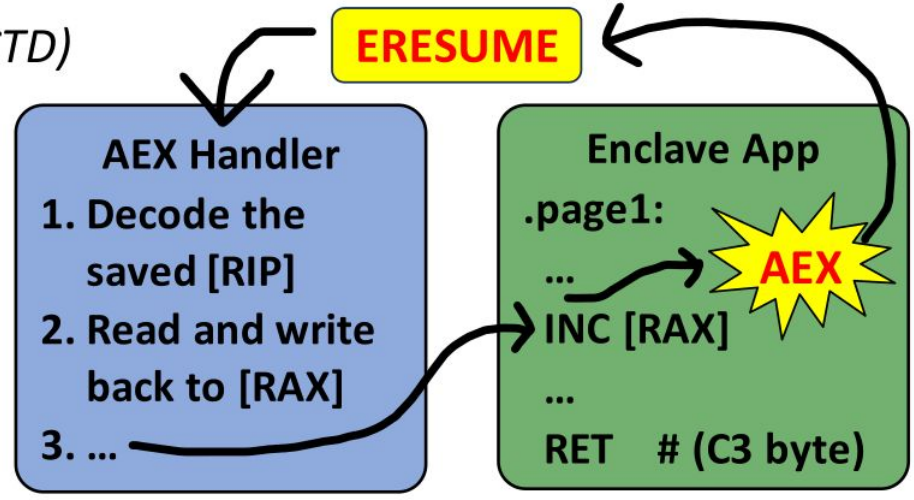


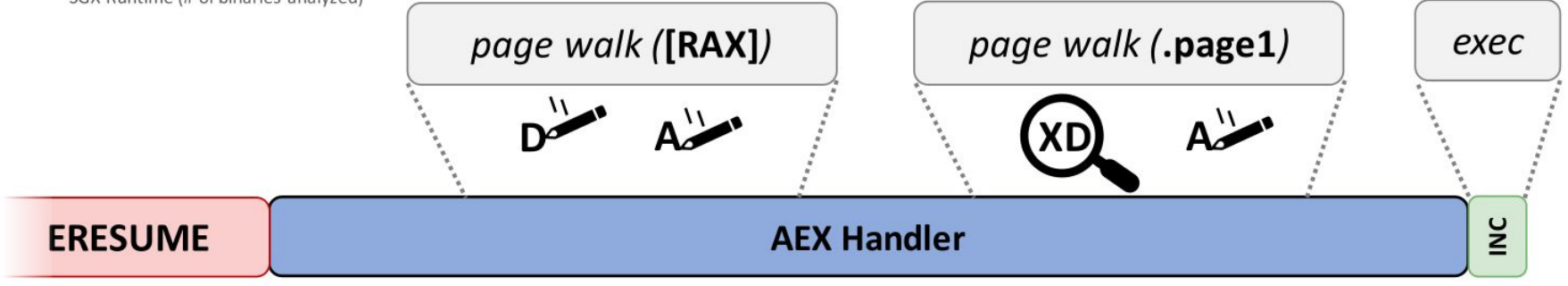
Constable et al. “AEX-Notify: Thwarting Precise Single-Stepping Attacks through Interrupt Awareness for Intel SGX Enclaves”, USENIX Security 2023.

AEX-Notify: Hardware-Software Co-Design Solution

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







Constable et al. “AEX-Notify: Thwarting Precise Single-Stepping Attacks through Interrupt Awareness for Intel SGX Enclaves”, USENIX Security 2023.

intel

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

phoronix

ARTICLES & REVIEWS | NEWS ARCHIVE | FORUMS | PREMIUM | CONTACT | CATEGORIES

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 Filter

sdk/trts/linux/trts_mitigation.S

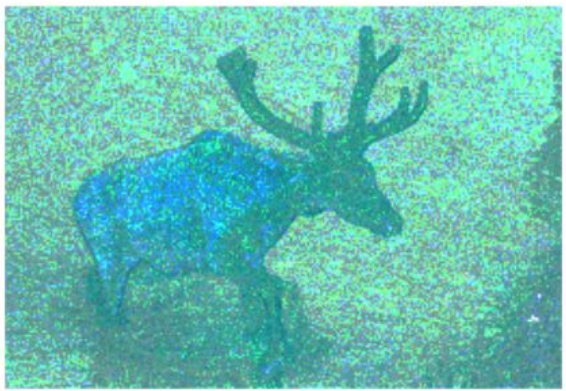
```
48   * Description:
49   *     The file provides mitigations for SGX-Step
50   */
71   *  Function:
constant_time_apply_sgxstep_mitigation_and_continue_execution
72   *      Mitigate SGX-Step and return to the point at which the
most recent
73   *      interrupt/exception occurred.
```

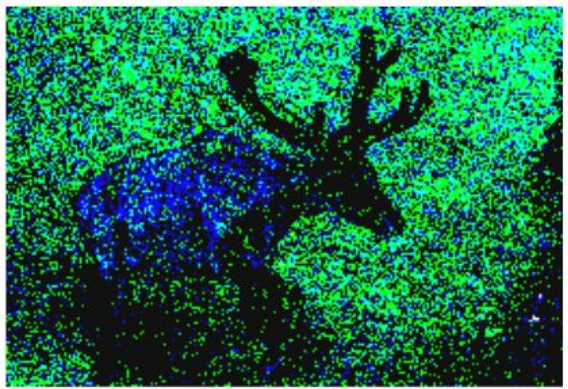
SGX-Step led to **changes in major OSs and enclave SDKs**

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

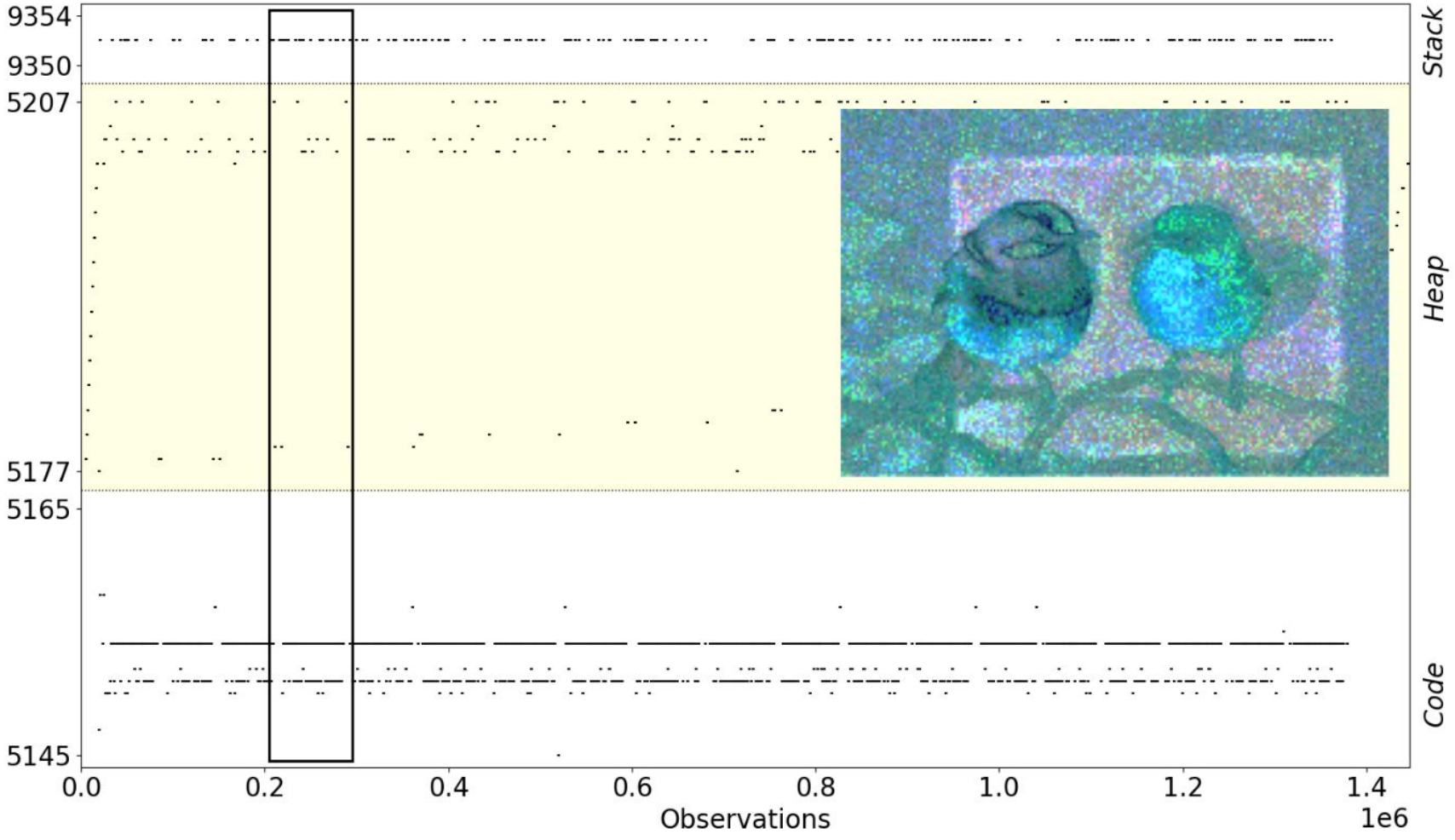




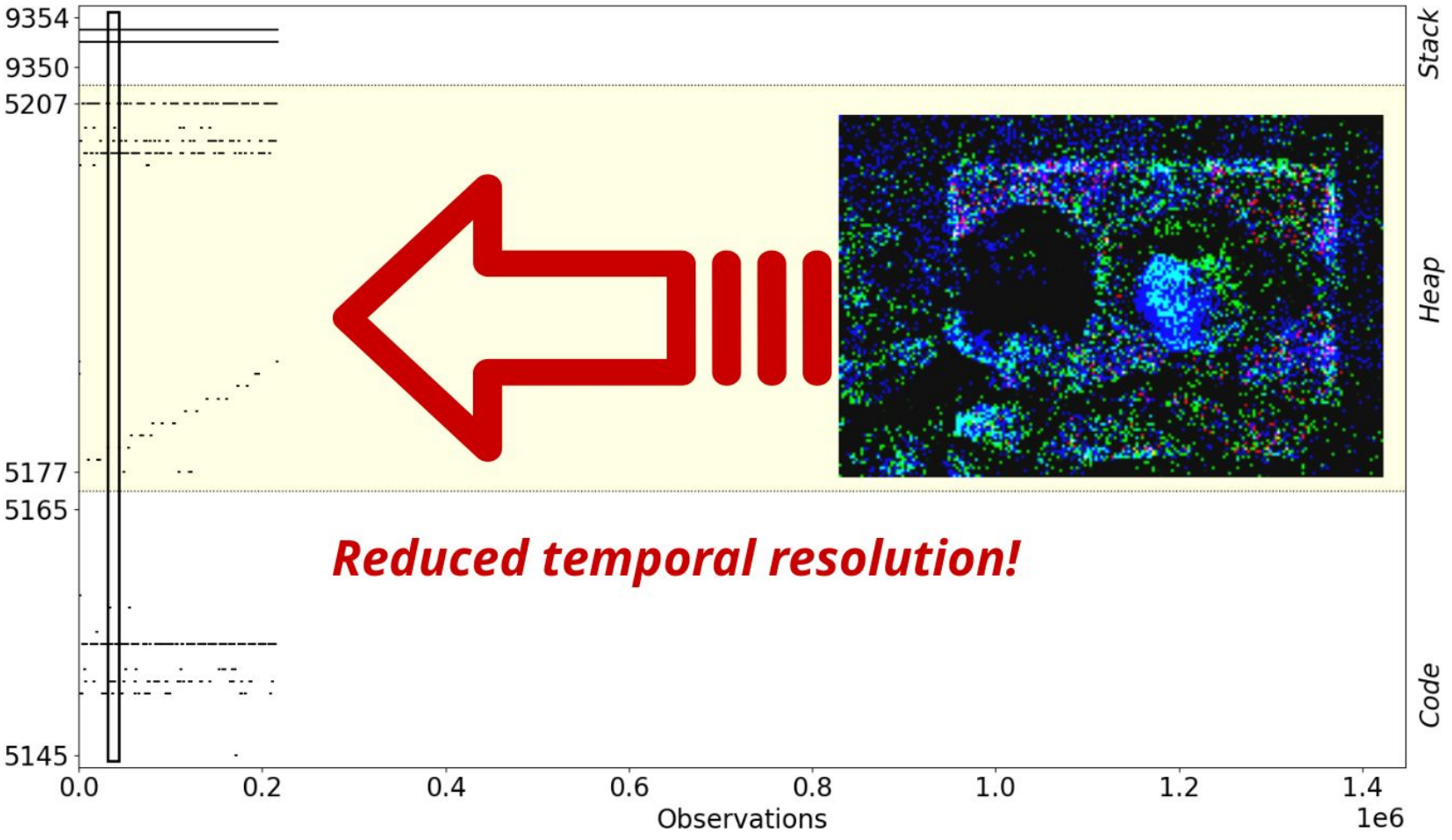
Original (left), Xu et al. (middle), our attack with AEX-Notify single-stepping defense (right)

Vanoverloop et. al “TLBlur: Compiler-Assisted Automated Hardening against Controlled Channels on Off-the-Shelf Intel SGX Platforms”, USENIX’25.

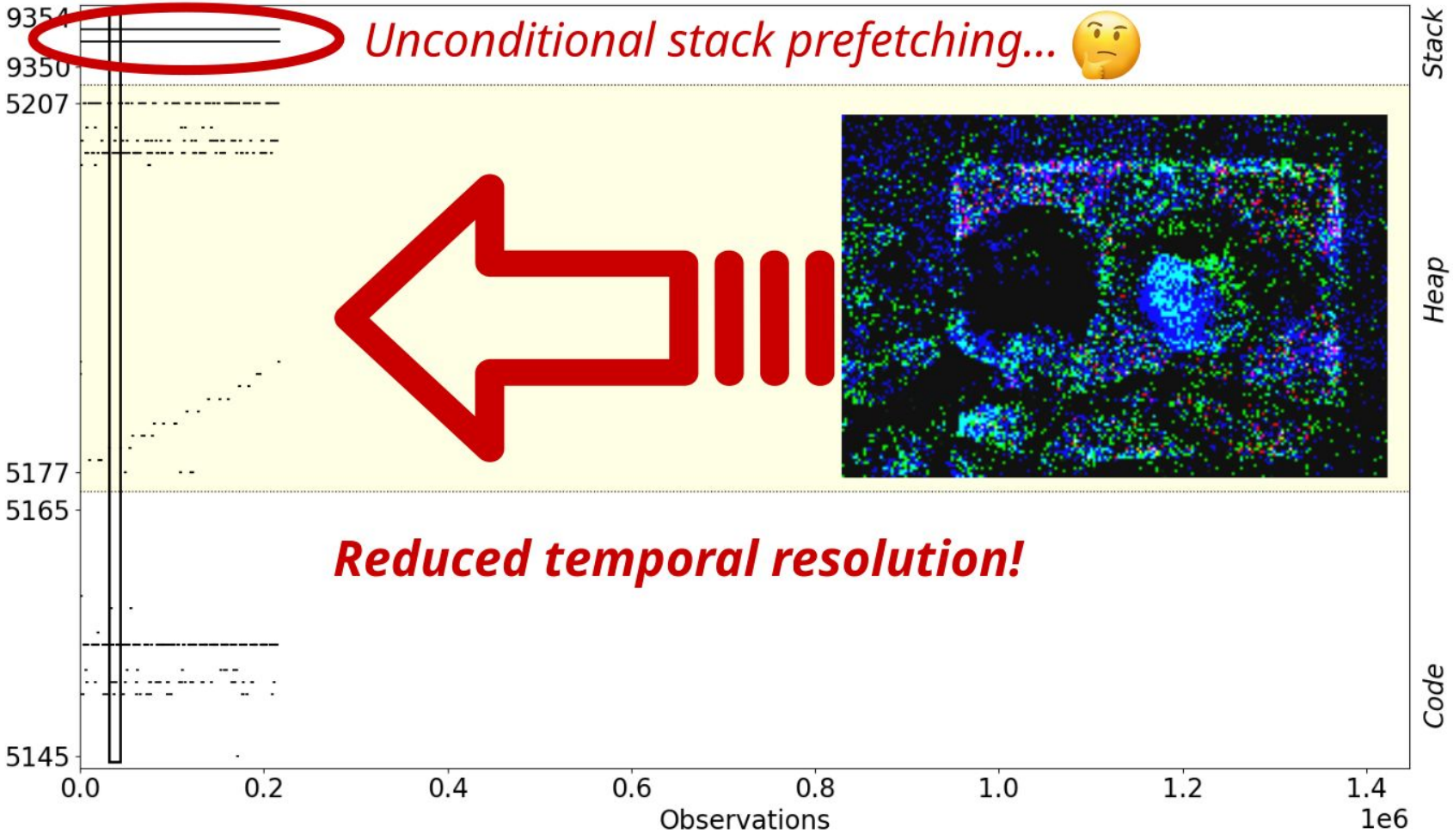
Libjpeg: AEX-Notify's Temporal Reduction in Practice



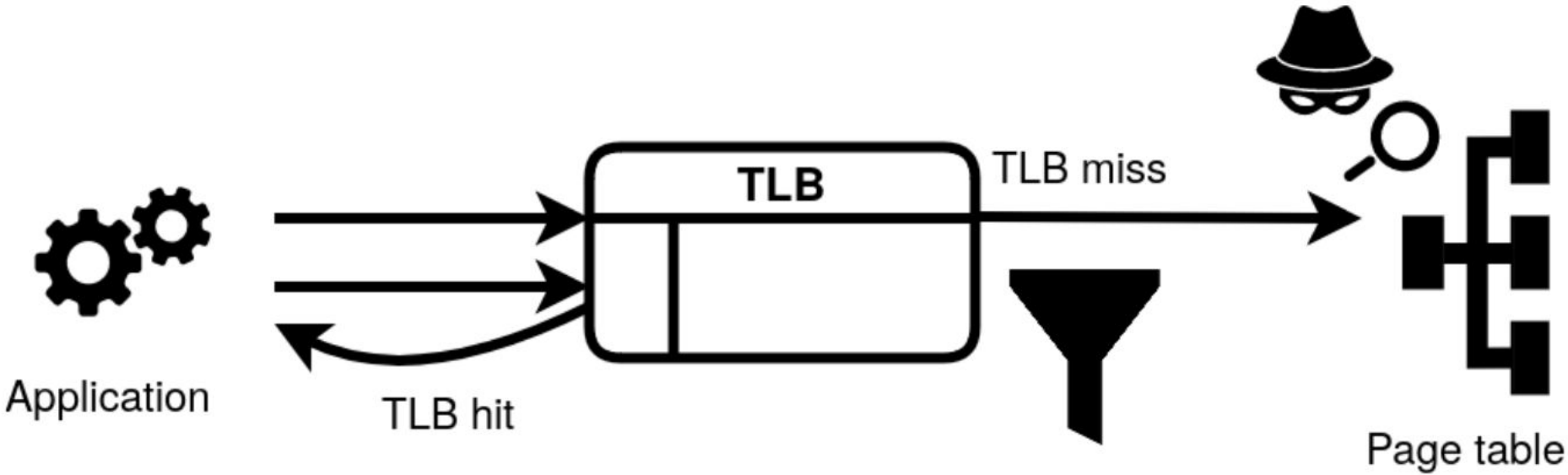
Libjpeg: AEX-Notify's Temporal Reduction in Practice



Libjpeg: AEX-Notify's Temporal Reduction in Practice

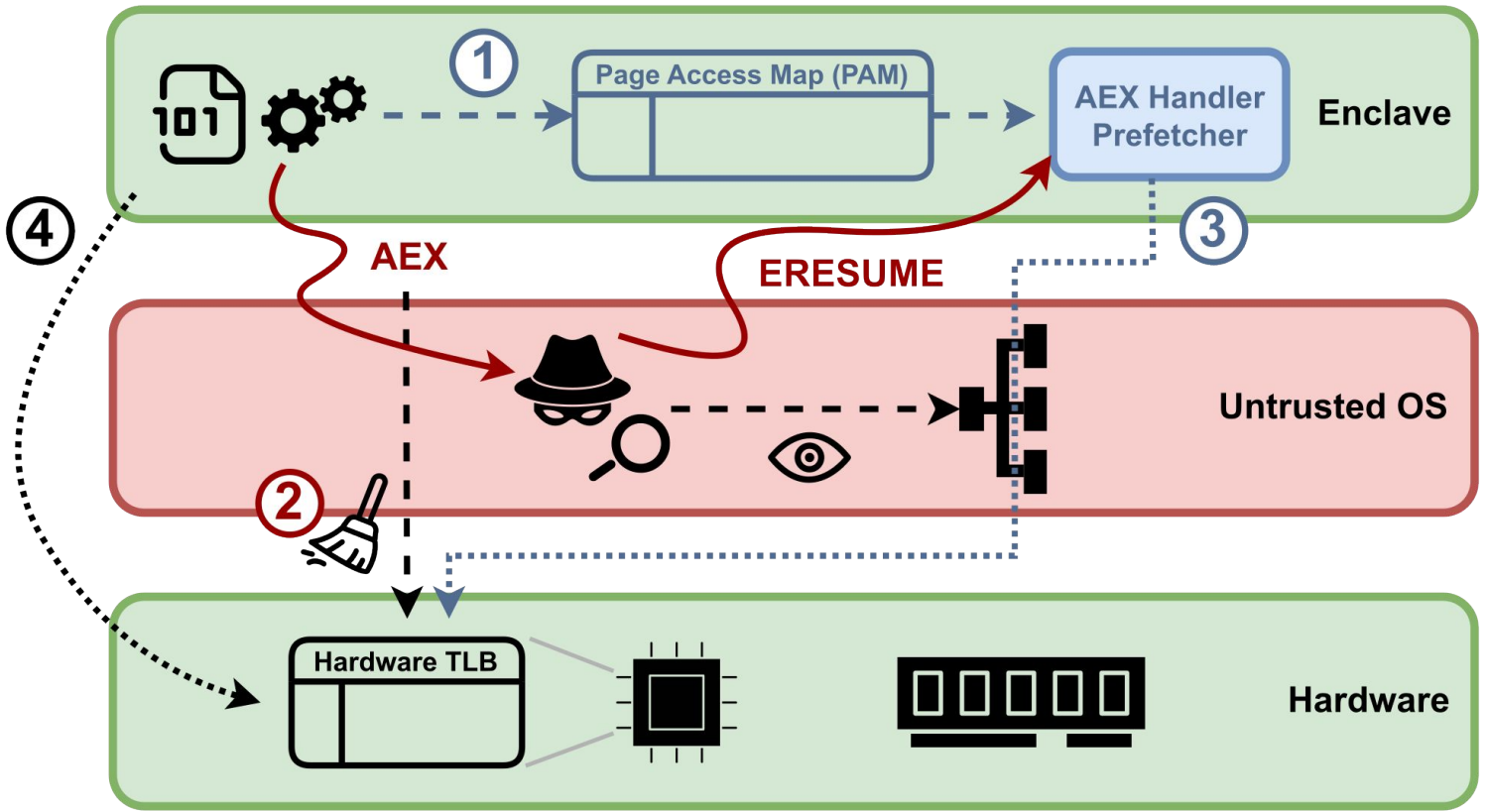


Idea: TLB as a “Filter” to Hide Page Accesses



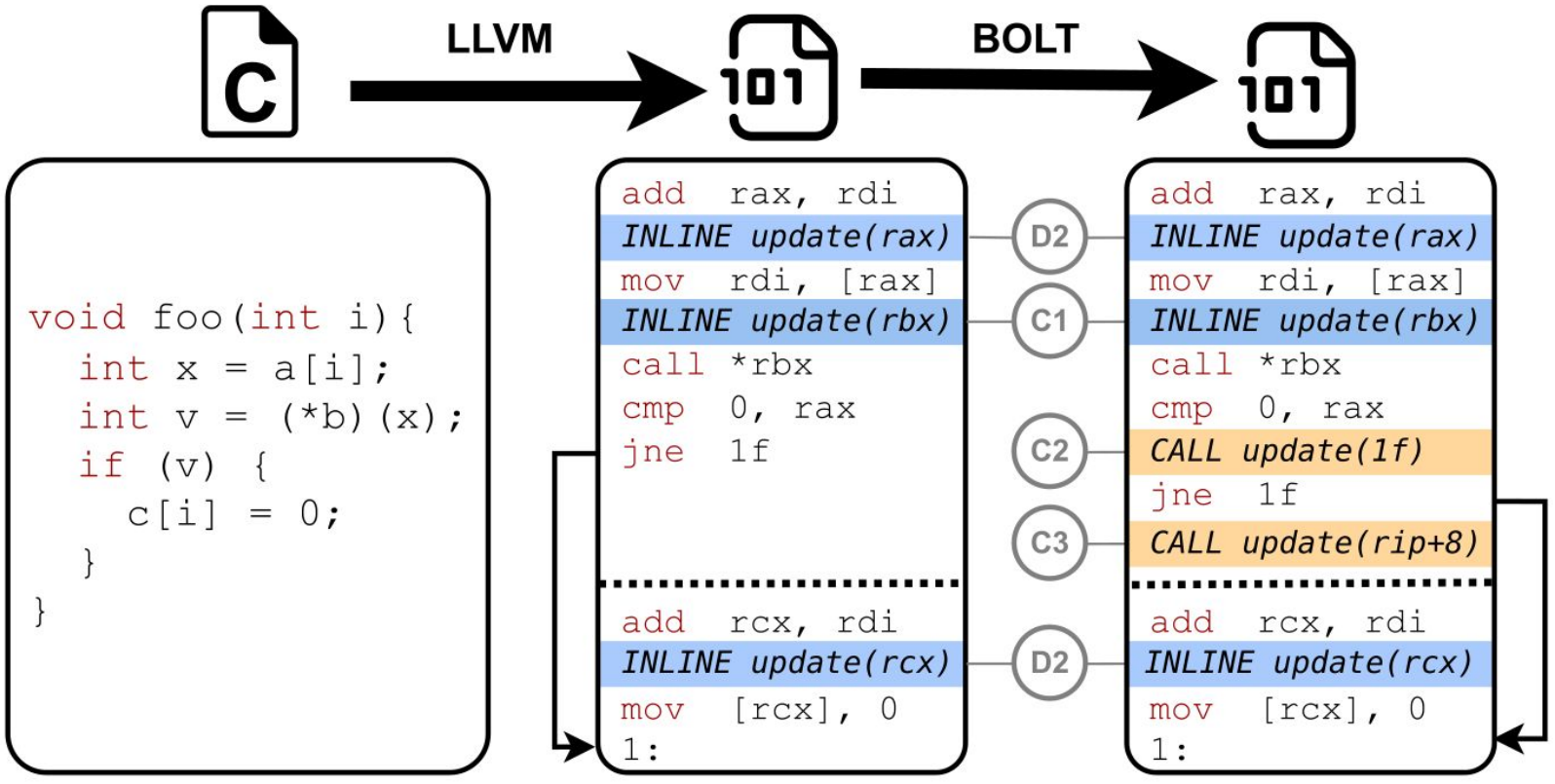
Vanoverloop et. al “TLBlur: Compiler-Assisted Automated Hardening against Controlled Channels on Off-the-Shelf Intel SGX Platforms”, USENIX’25.

TLBlur: Self-Monitoring and Restoring Enclave Page Accesses



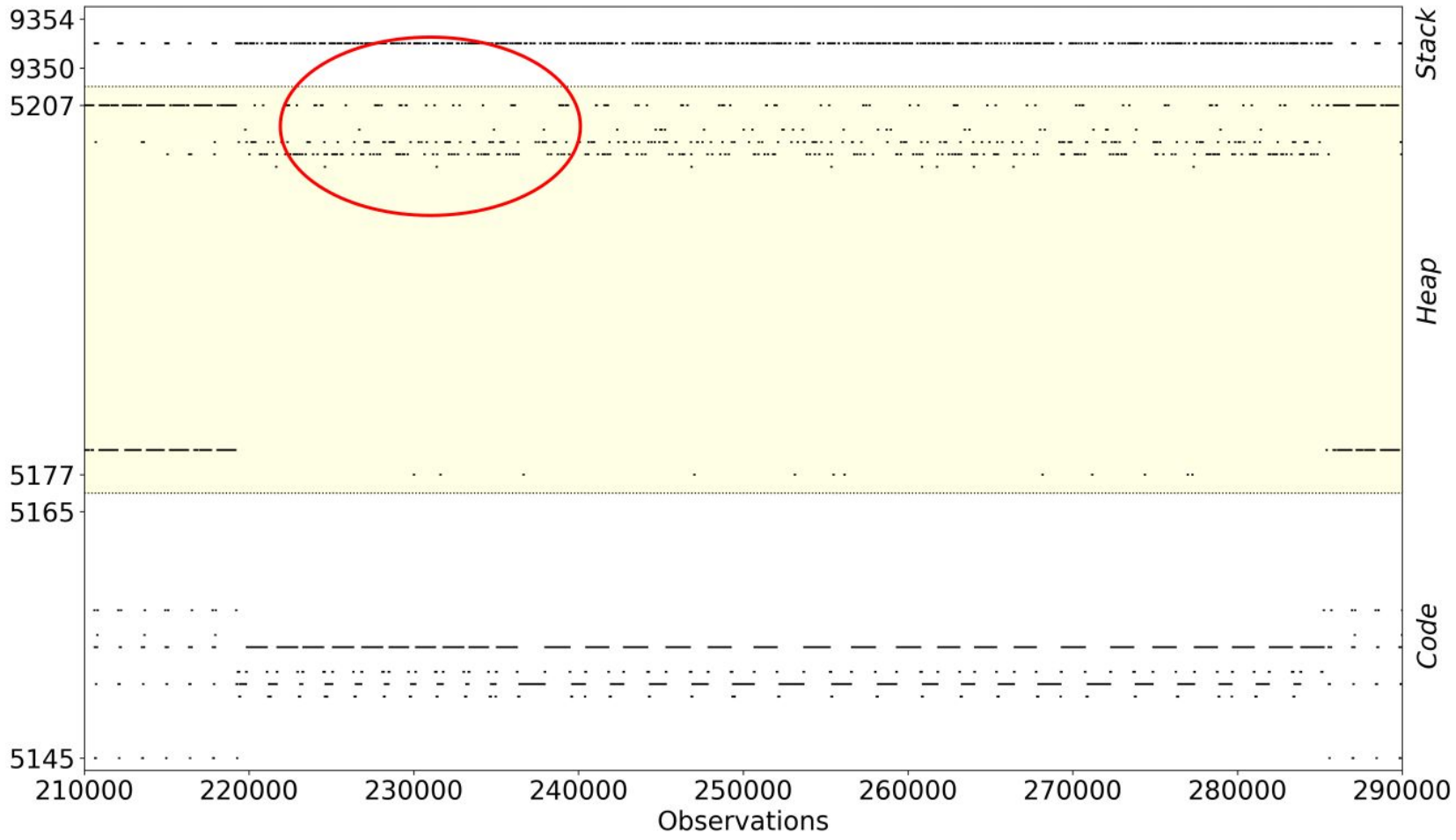
Vanoverloop et. al "TLBlur: Compiler-Assisted Automated Hardening against Controlled Channels on Off-the-Shelf Intel SGX Platforms", USENIX'25.

Instrumentation to Self-Monitor Page Accesses at Runtime

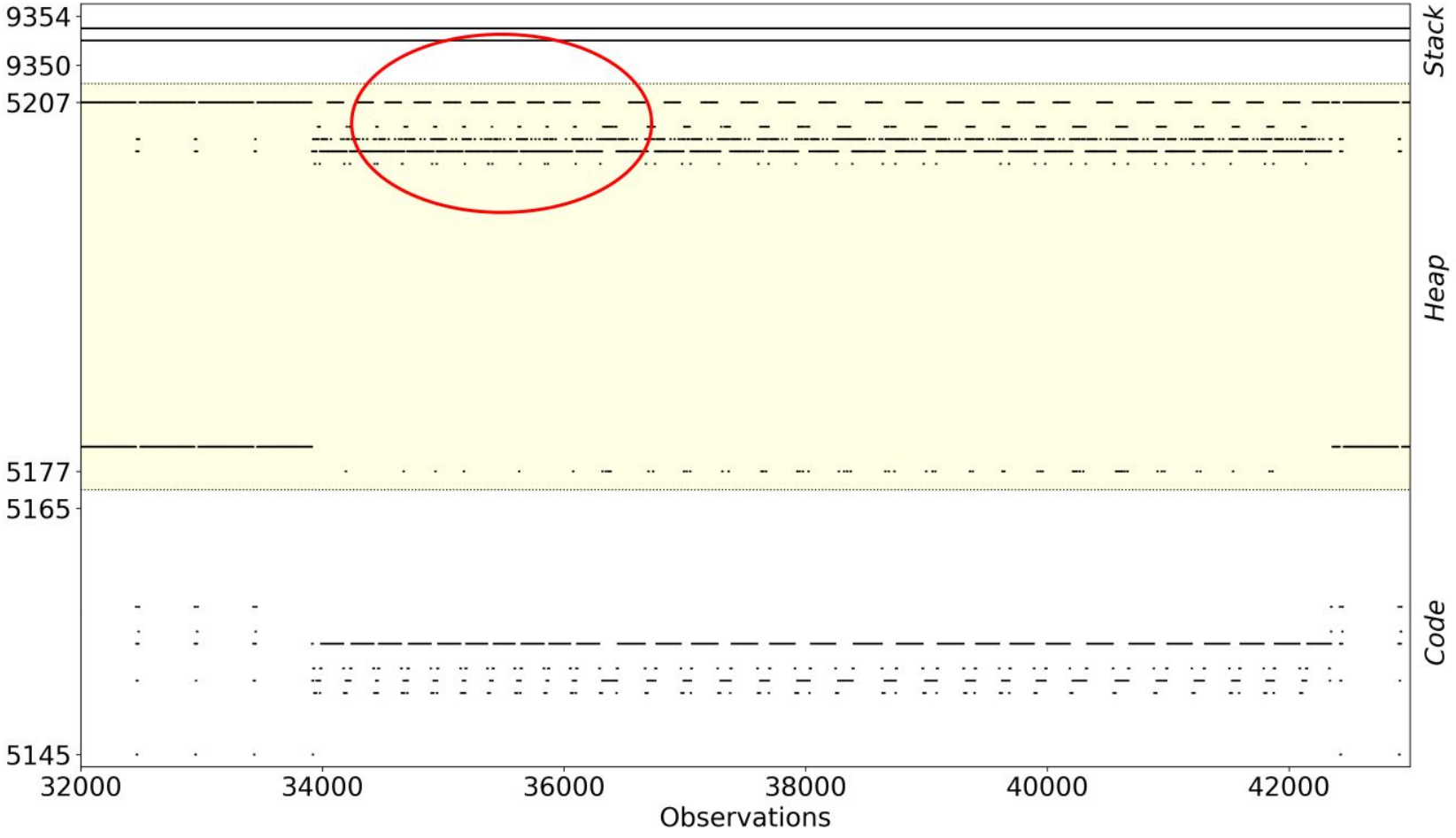


Vanoverloop et. al “TLBlur: Compiler-Assisted Automated Hardening against Controlled Channels on Off-the-Shelf Intel SGX Platforms”, USENIX’25.

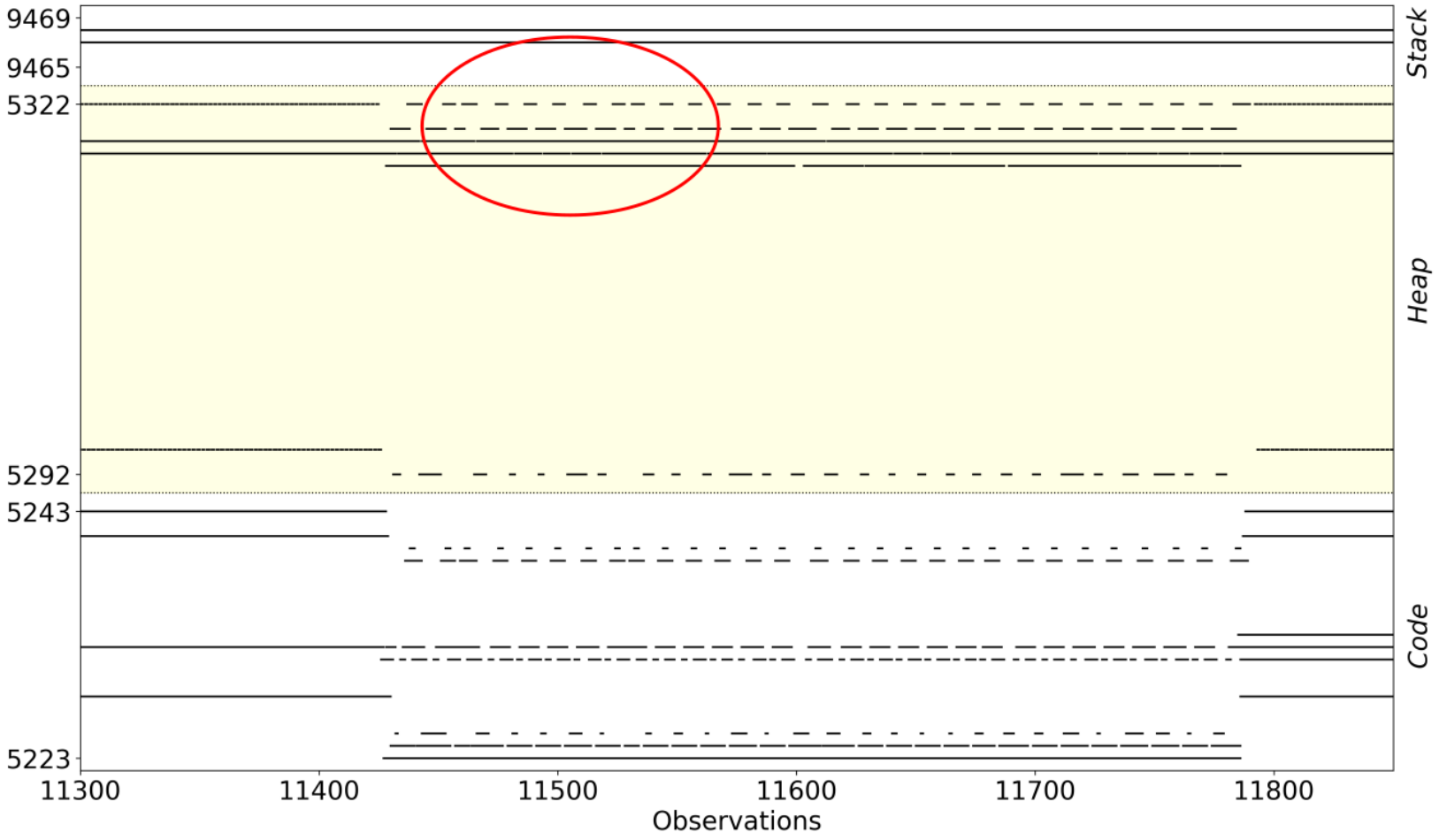
Leakage Reduction in Practice: Libjpeg Single-Stepping



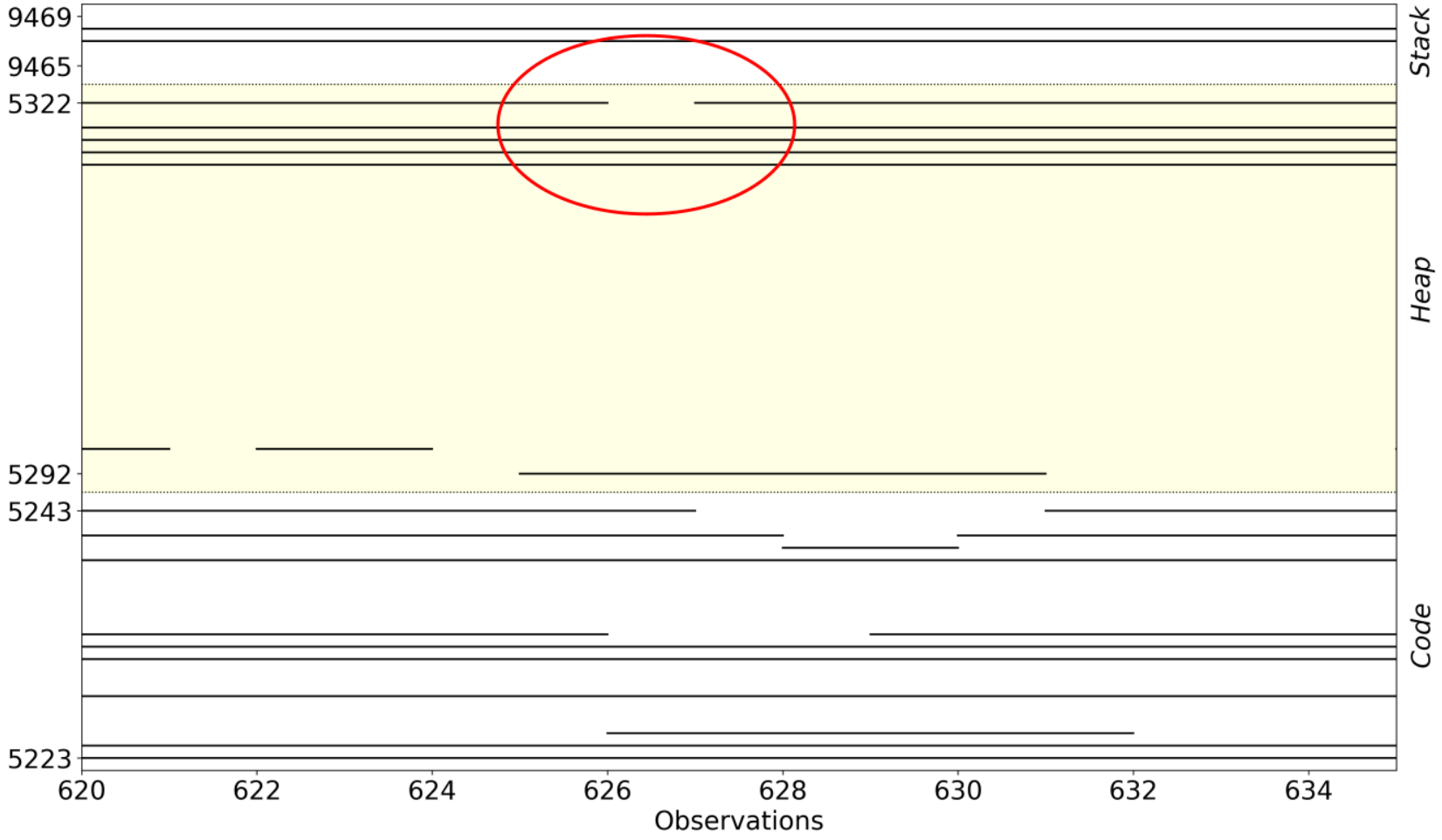
Leakage Reduction in Practice: Libjpeg Page Faults



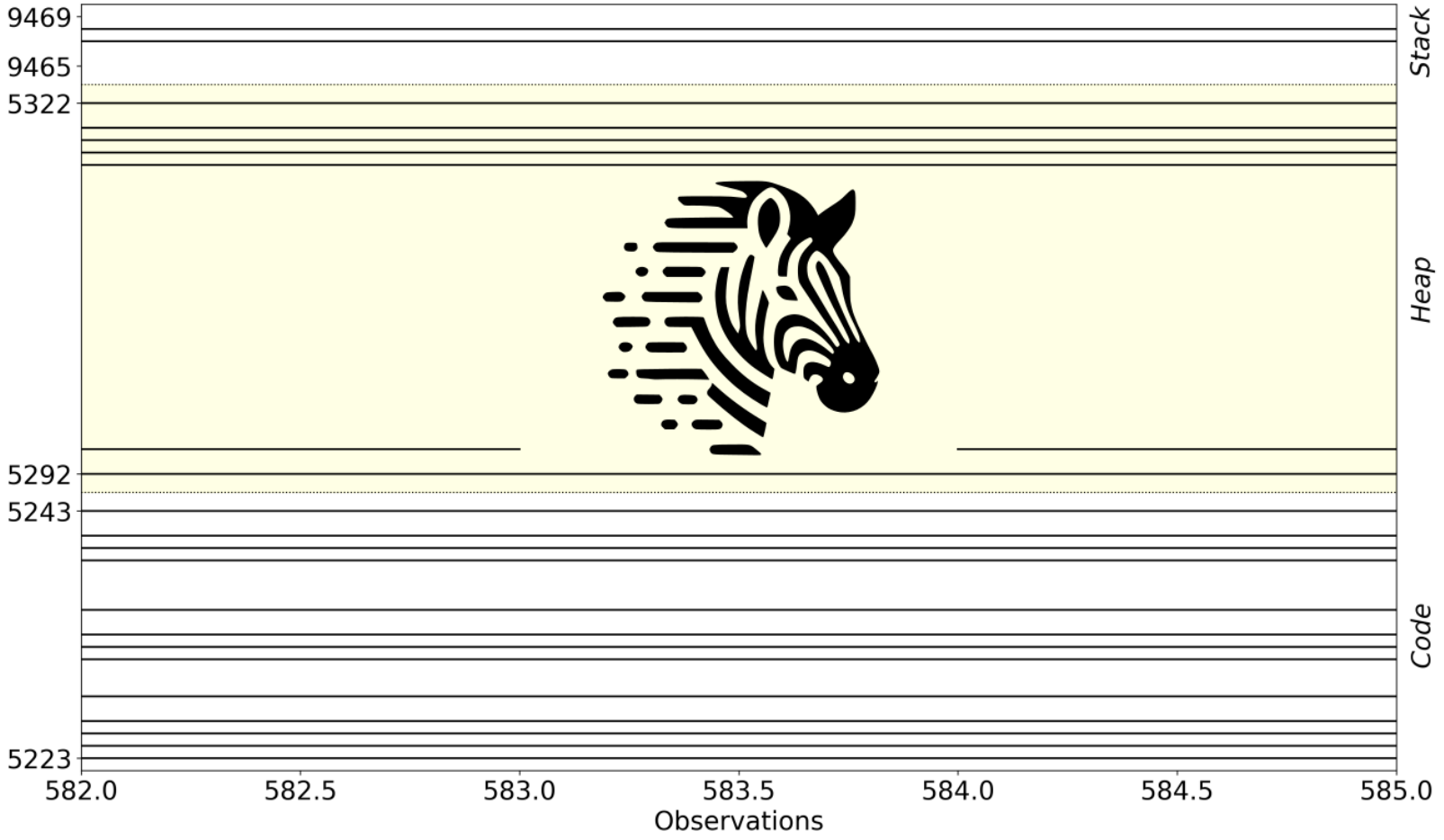
Leakage Reduction in Practice: Libjpeg TLBlur (N=10)



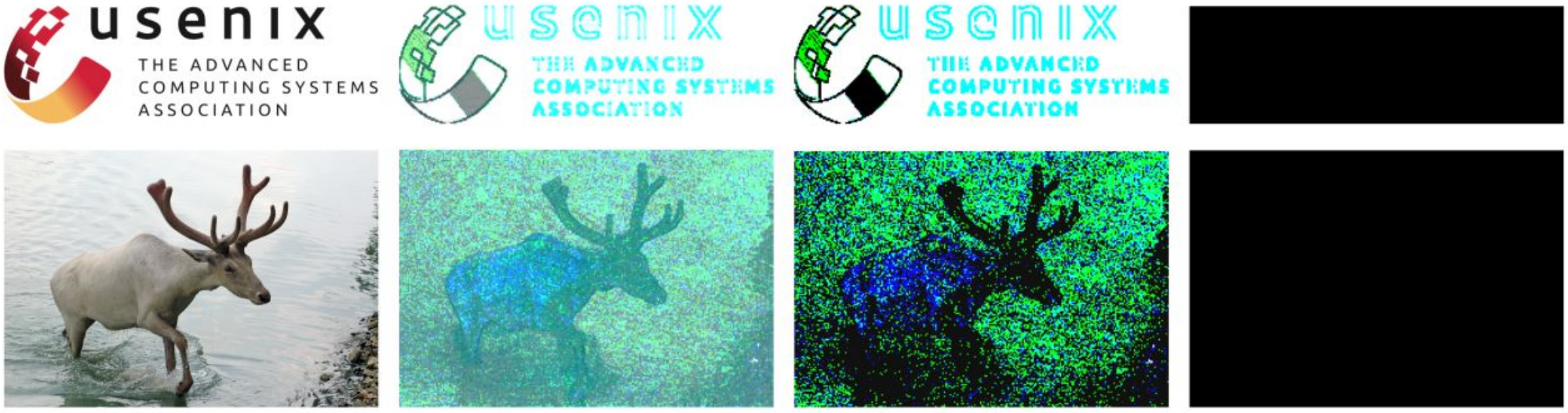
Leakage Reduction in Practice: Libjpeg TLBlur (N=20)



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



TLBlur: Compiler-Assisted Leakage Reduction in Practice



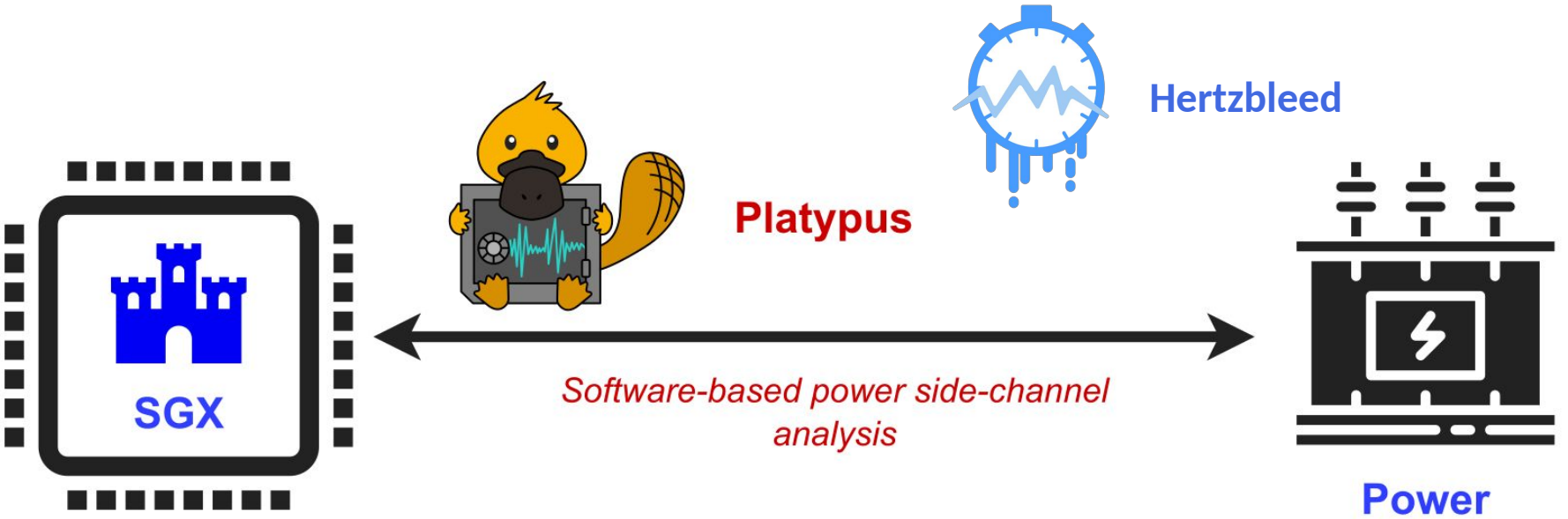
Automated “blurring” of page-access traces in space and time

Vanoverloop et. al “TLBlur: Compiler-Assisted Automated Hardening against Controlled Channels on Off-the-Shelf Intel SGX Platforms”, USENIX’25.

Idea #3: Model-Specific Registers and Performance-Monitoring Counters?

Software-Based DVFS Attacks

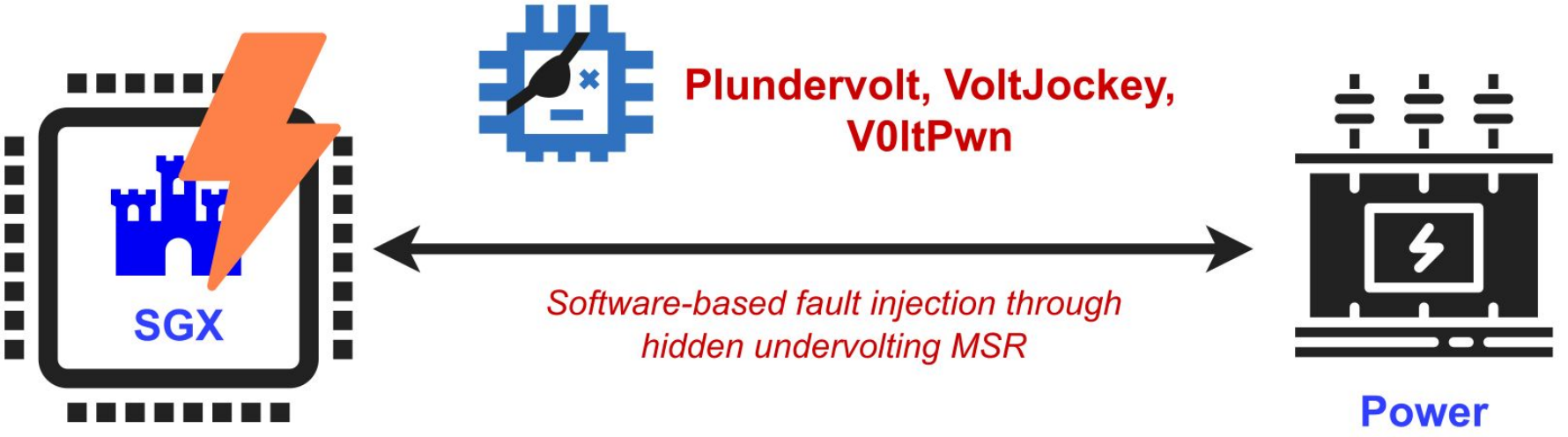
Software-Based DVFS Attacks



Lipp et al. "PLATYPUS: Software-based Power Side-Channel Attacks on x86", S&P 21.

Requirements: Software-only attacker → *Metadata* + *direct data* leakage(!)

Software-Based DVFS Attacks



Murdock et al. "Plundervolt: Software-Based Fault Injection Attacks against Intel SGX", S&P 20.

Requirements: Software-only attacker → Fault injection (*integrity* breach)

How a little bit of undervolting can cause a lot of problems

Intel® Extreme Tuning Utility

Warning

Altering clock frequency or voltage may:

- damage or reduce the useful life of the processor and other system componer
- may reduce system stability and performance.

Product warranties may not apply if the processor is operated beyond its specifications. Check with the manufacturers of system and components for additional details.

I agree | I agree, don't show again | Cancel

Intel Power Management

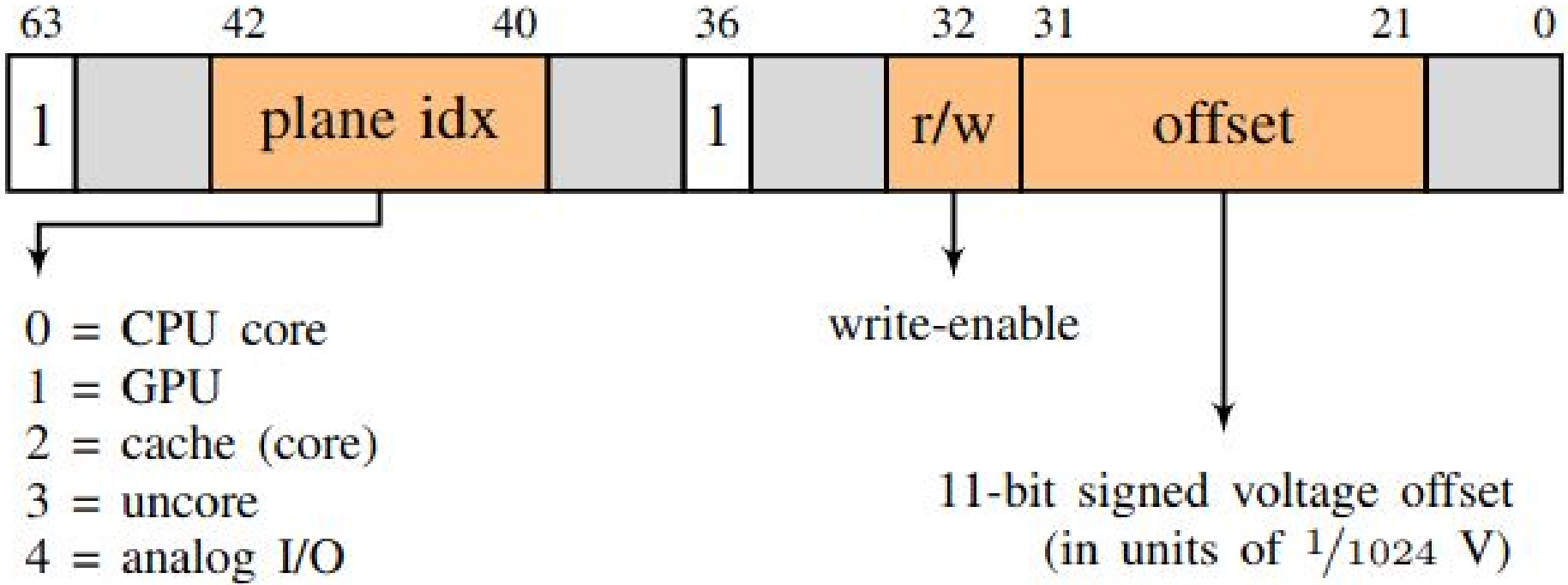


Fig. 1. Layout of the undocumented undervolting MSR with address `0x150`.

Plundervolt: Will it Fault?

```
uint64_t multiplier = 0x1122334455667788;
uint64_t var = 0xdeadbeef * multiplier;

while (var == 0xdeadbeef * multiplier)
{
    var = 0xdeadbeef;
    var *= multiplier;
}
var ^= 0xdeadbeef * multiplier;
```

Plundervolt: Will it Fault?

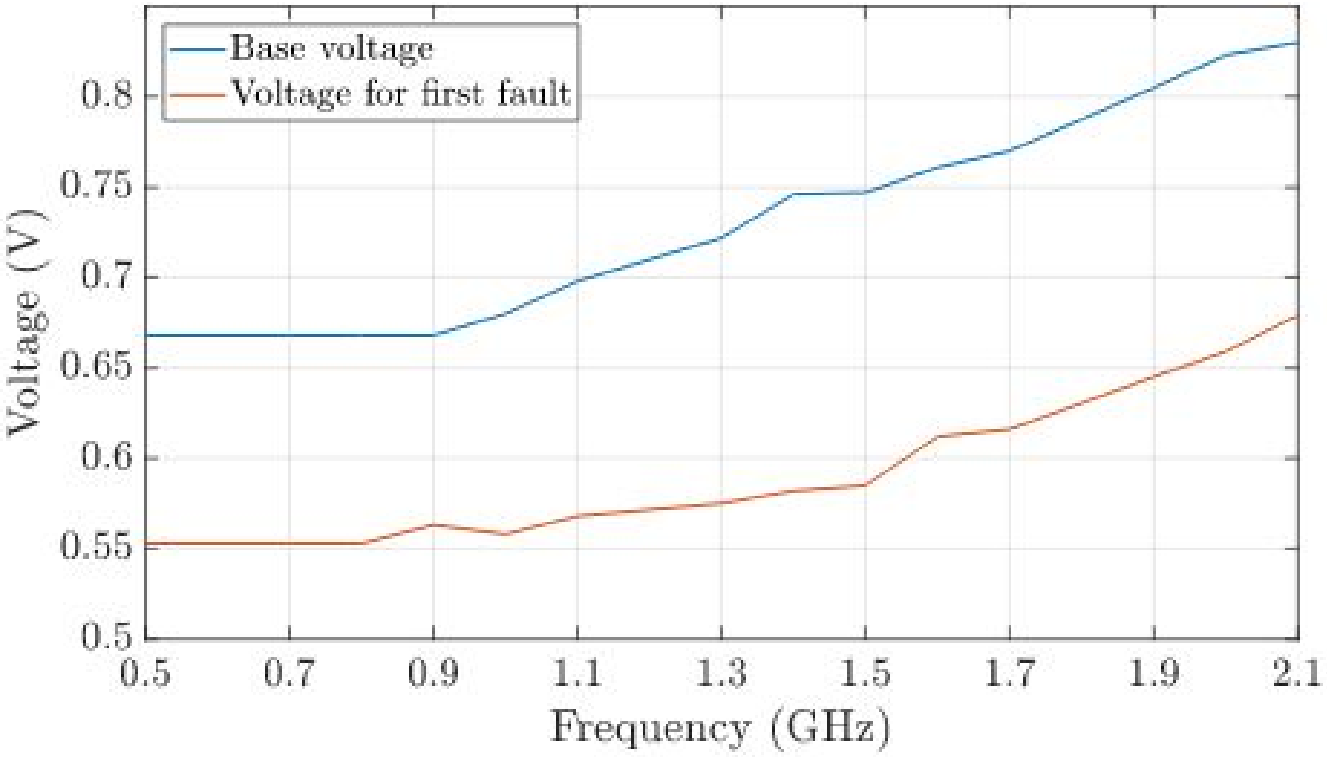


Fig. 3. Base voltage (blue) and voltage for first fault (orange) vs. CPU frequency for the i3-7100U-A

Plundervolt: Differential Fault Analysis of AES-NI in SGX

```
[Enclave] plaintext: 5ABB97CCFE5081A4598A90E1CEF1BC39
[Enclave] CT1: DE49E9284A625F72DB87B4A559E814C4 <- faulty
[Enclave] CT2: BDFADCE3333976AD53BB1D718DFC4D5A <- correct

input to round 10:
[Enclave]    1: CD58F457 A9F61565 2880132E 14C32401
[Enclave]    2: AEEBC19C D0AD3CBA A0BCBAFA C0D77D9F

input to round  9:
[Enclave]    1: 6F6356F9 26F8071F 9D90C6B2 E6884534
[Enclave]    2: 6F6356C7 26F8D01F 9DF7C6B2 A4884534

input to round  8:
[Enclave]    1: 1C274B5B 2DFD8544 1D8AEAC0 643E70A1
[Enclave]    2: 1C274B5B 2DFD8544 1D8AEAC0 6466 70A1
```

Plundervolt: Beyond Crypto—Inducing Memory-Safety Faults

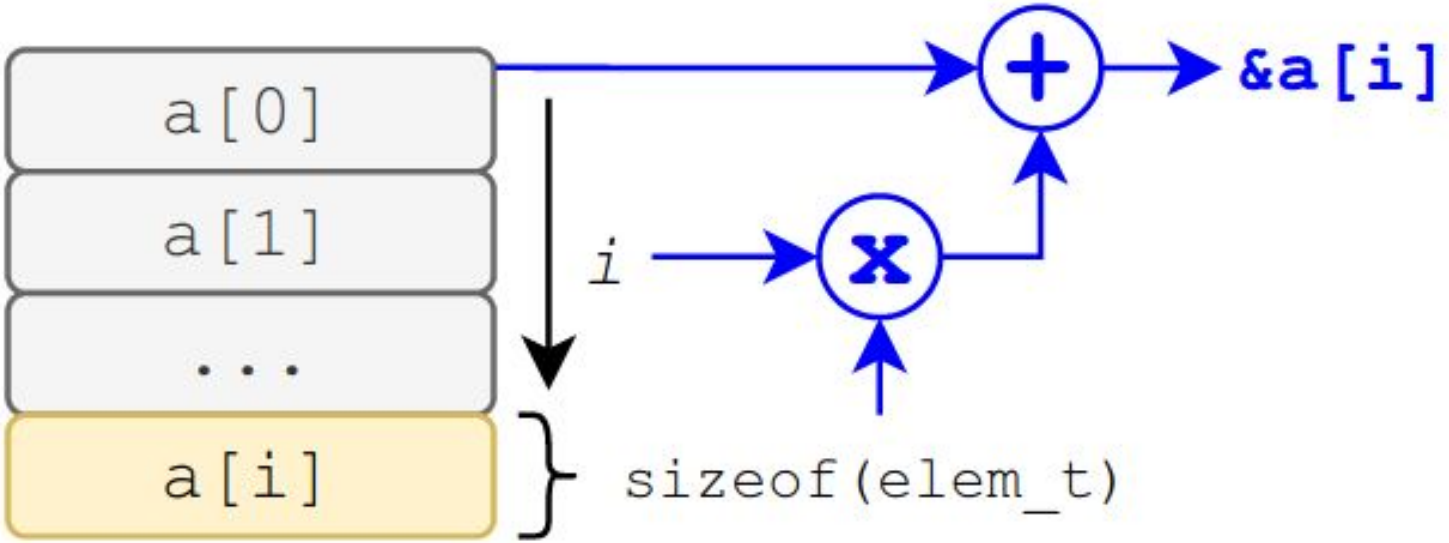
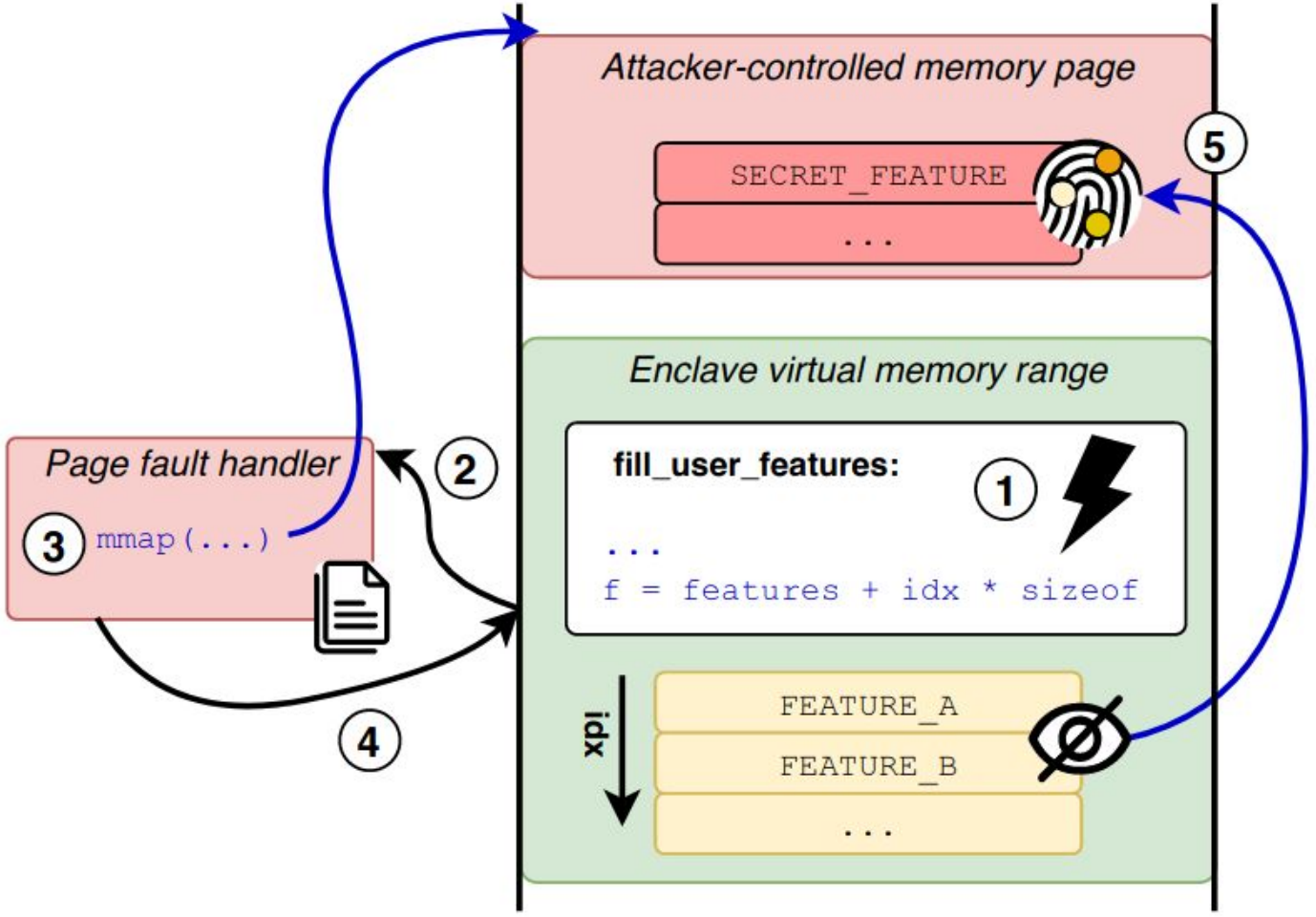


Figure 4. The address of element `a[i]` in an array is computed as `&a[0] + i * sizeof(elem_t)`.

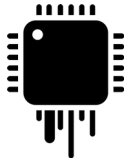
Plundervolt: Beyond Crypto—Inducing Memory-Safety Faults



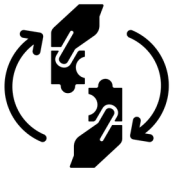
DALL·E 3

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!