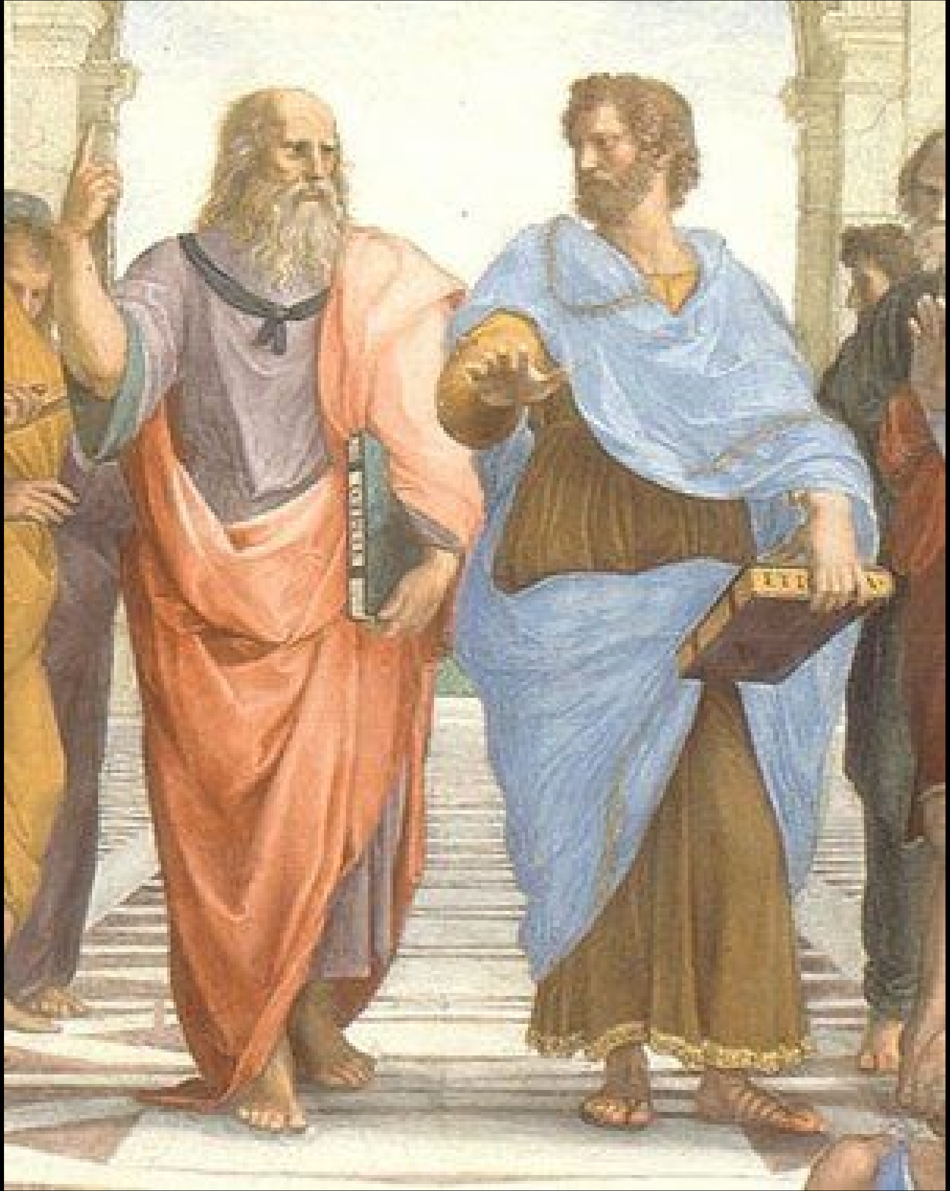
Beyond the Surface: Validation Challenges and Opportunities for Confidential Computing

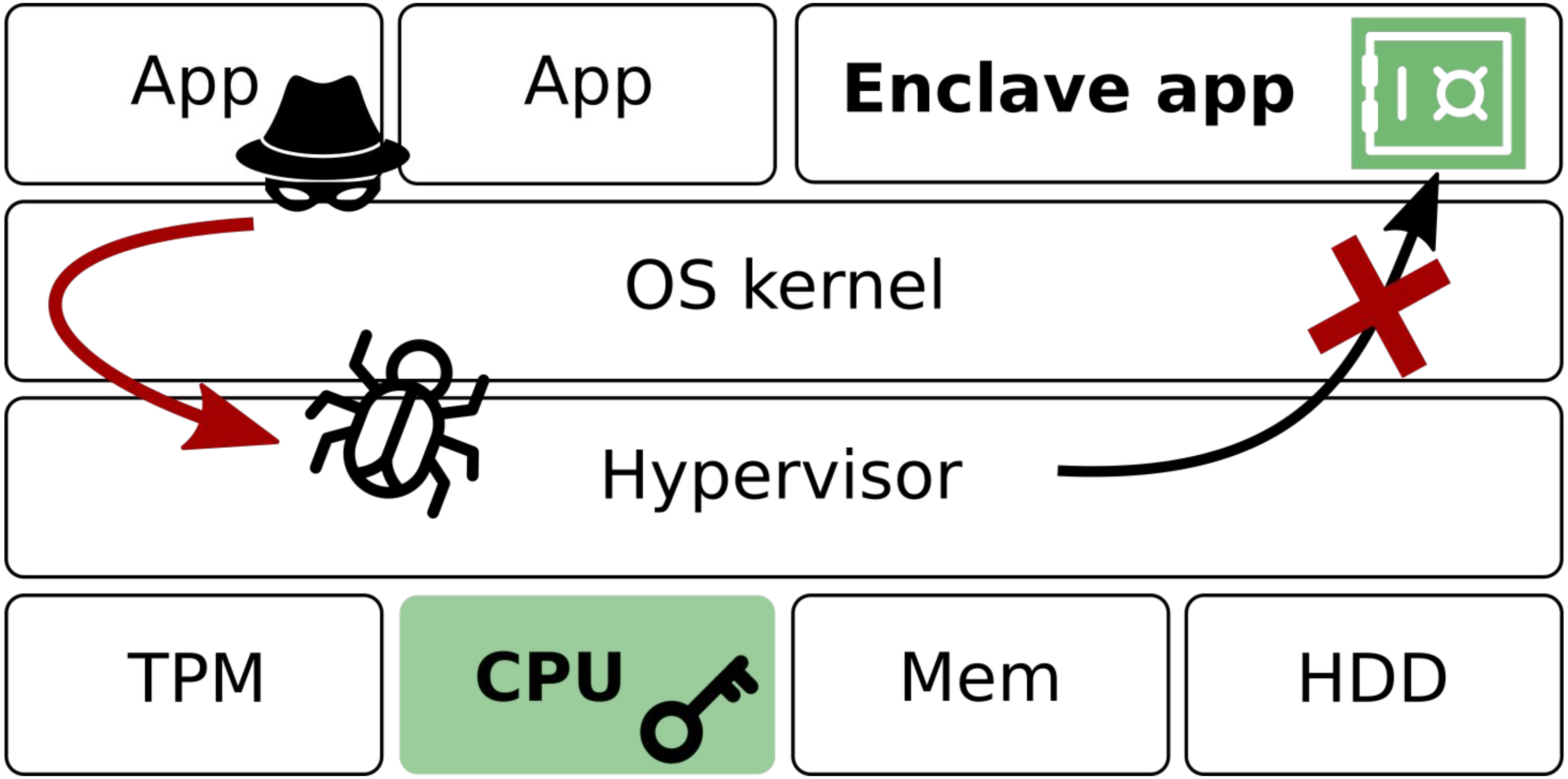
Jo Van Bulck

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

September 9, 2024, PAVeTrust @ FM'24

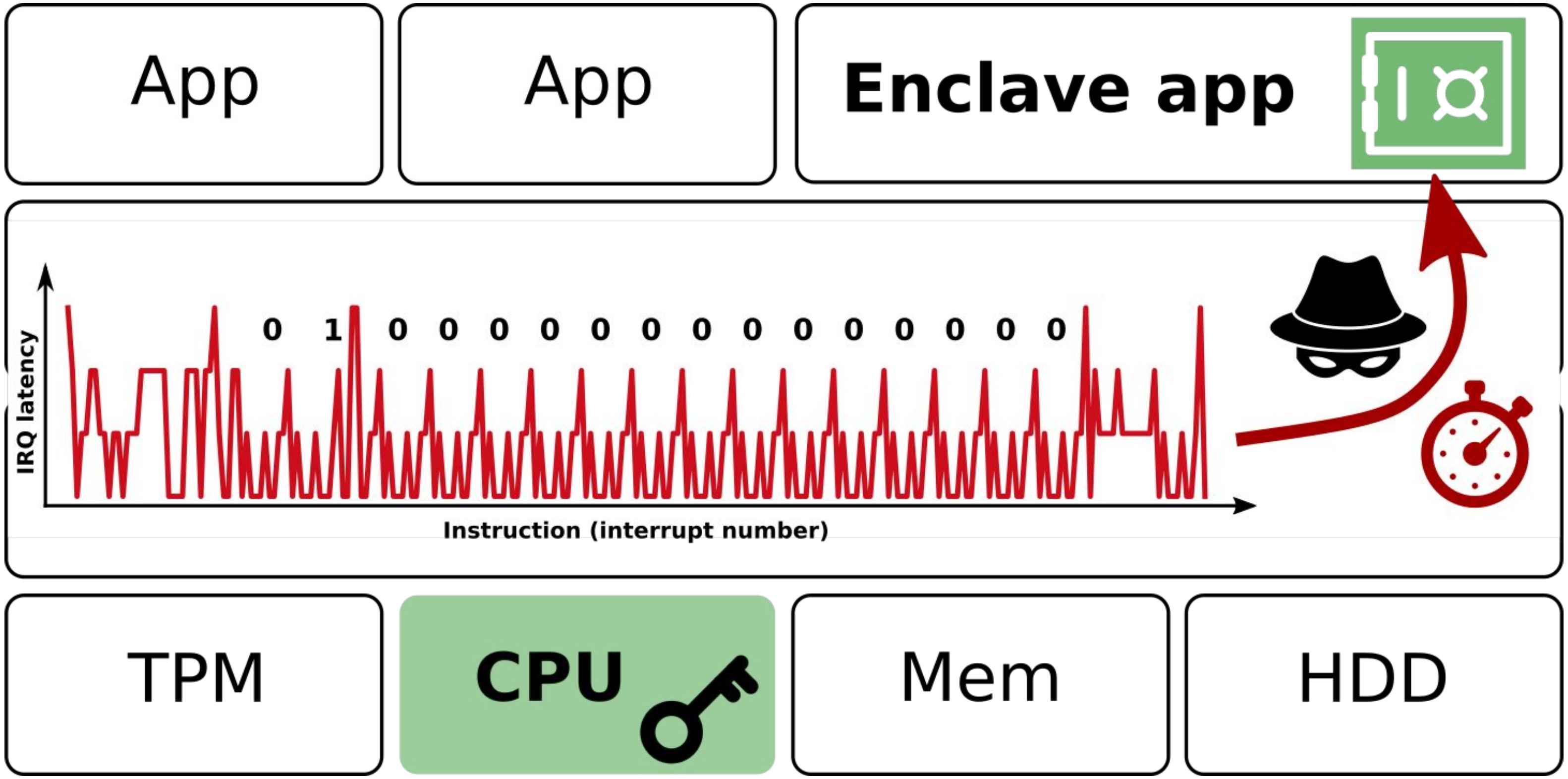


Enclaved Execution: Reducing Attack Surface



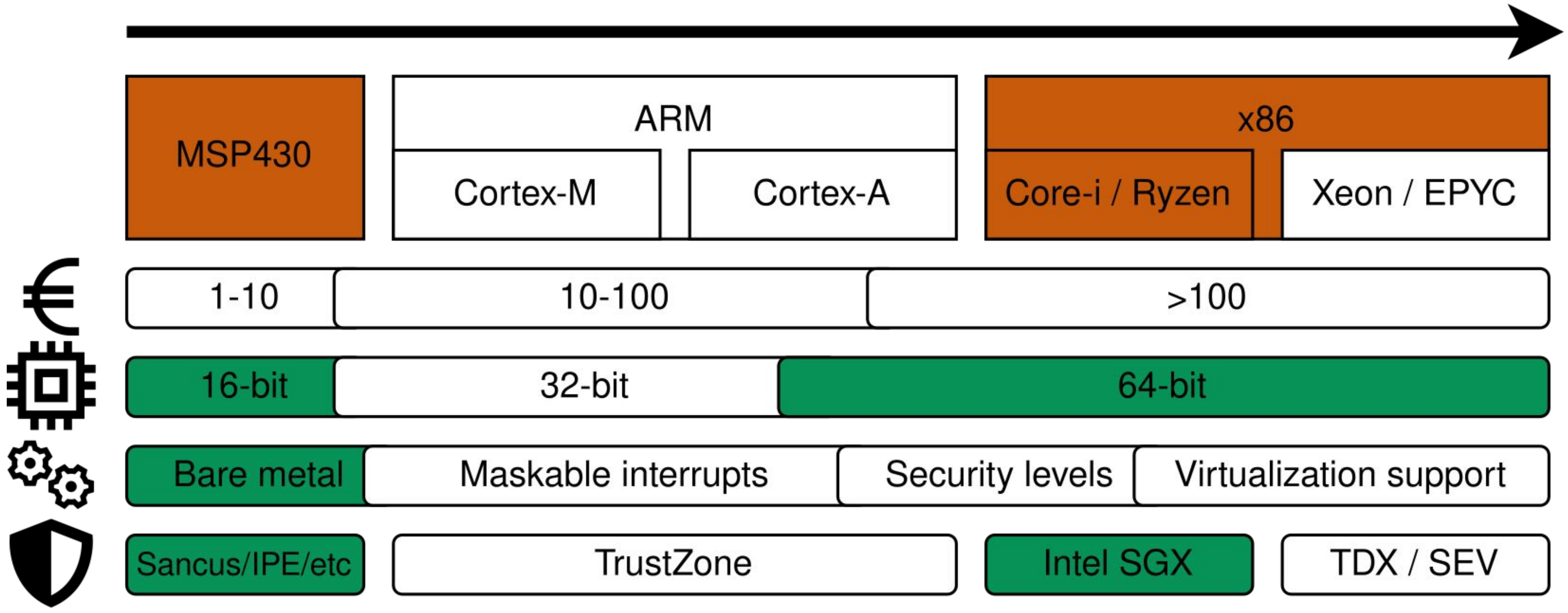
“Platonic” ideal: Hardware-level isolation and attestation

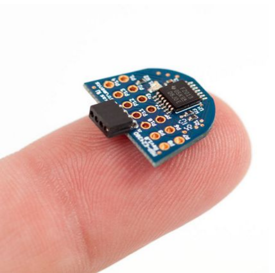
Enclaved Execution: Privileged Side-Channel Attacks



Reality #1: Microarchitectural side channels

Confidential Computing Spectrum

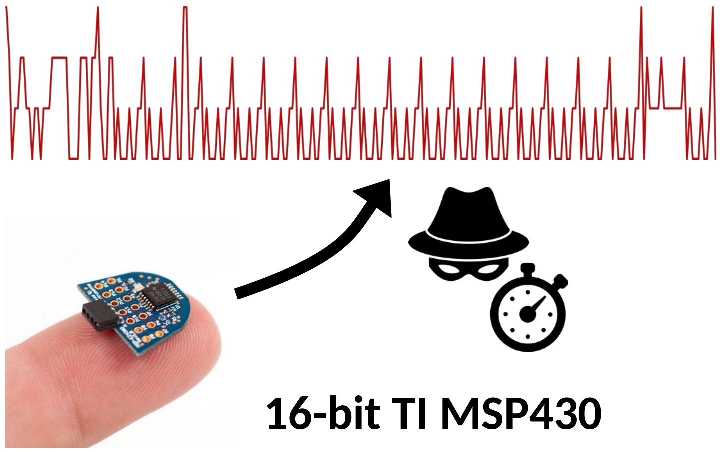




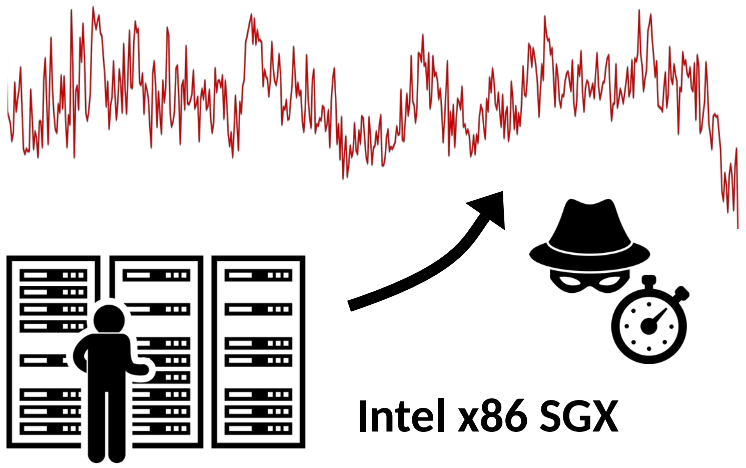
Reality #2: Heterogeneous CPU spectrum

Case Study: Hardware-Software Co-Design for Secure IRQs

Interrupts == Universal attack primitive



16-bit TI MSP430



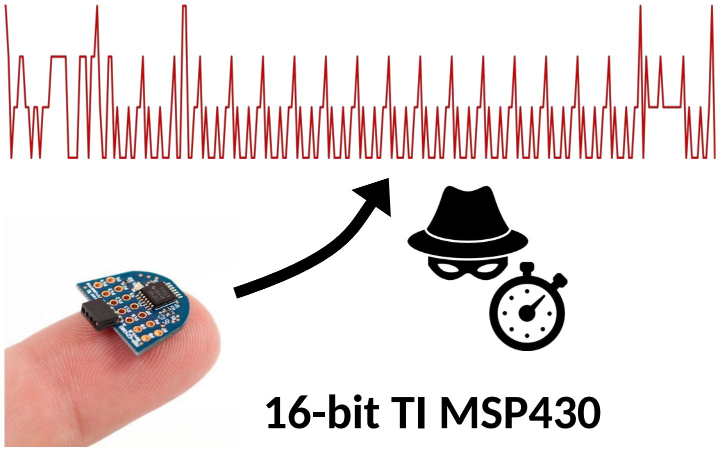
Intel x86 SGX

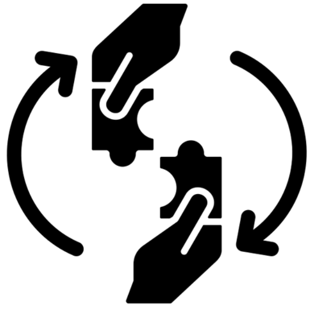
- Small CPU, open source

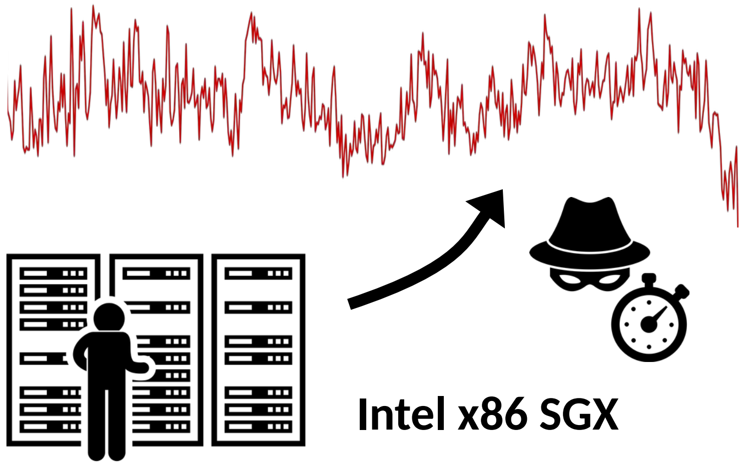
- Large CPU, proprietary

Case Study: Hardware-Software Co-Design for Secure IRQs

Interrupts == Universal attack primitive
→ *explore synergy low-end <> high-end TEEs*





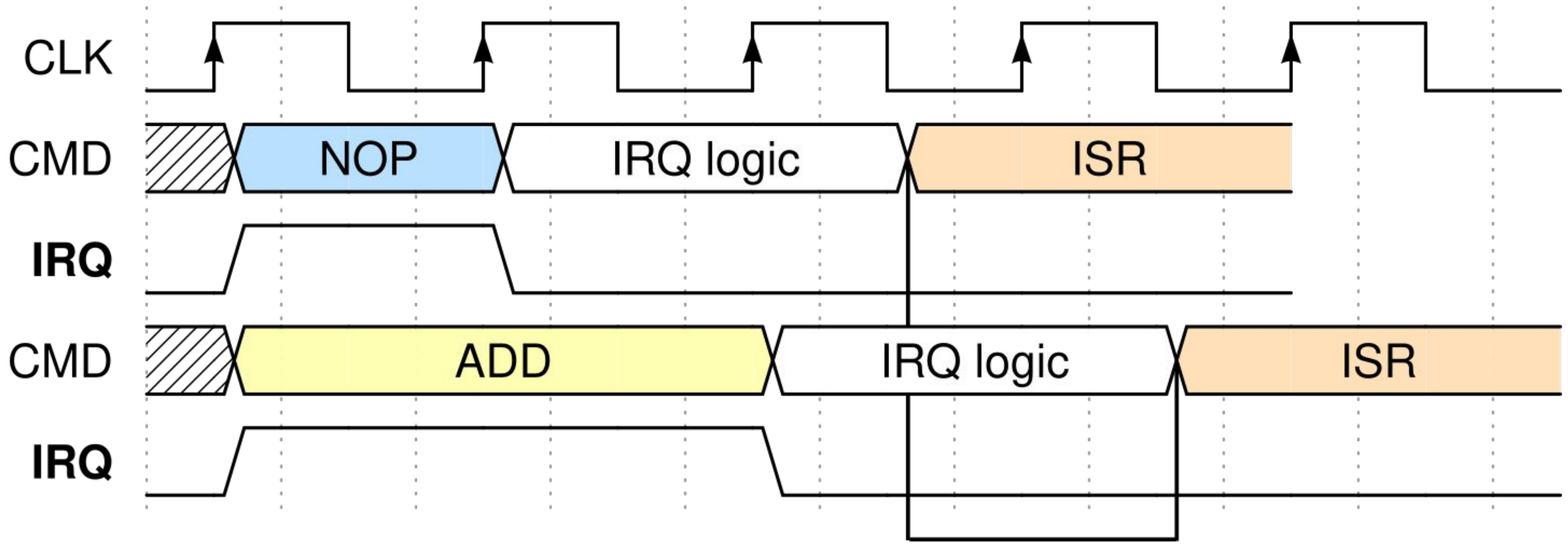


- Small CPU, open source
- *Formal operational semantics*

- Large CPU, proprietary
- *Pragmatic mitigation primitives*

Reality #1: IRQ Side channels?

Wait a Cycle: Interrupt Latency as a Side Channel



```
if (secret){ ADD @R5+, R6;} // 2 cycles
else       { NOP; NOP;    } // 2*1 cycle
```

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

TIMING LEAKS
EVERYWHERE
imgflip.com

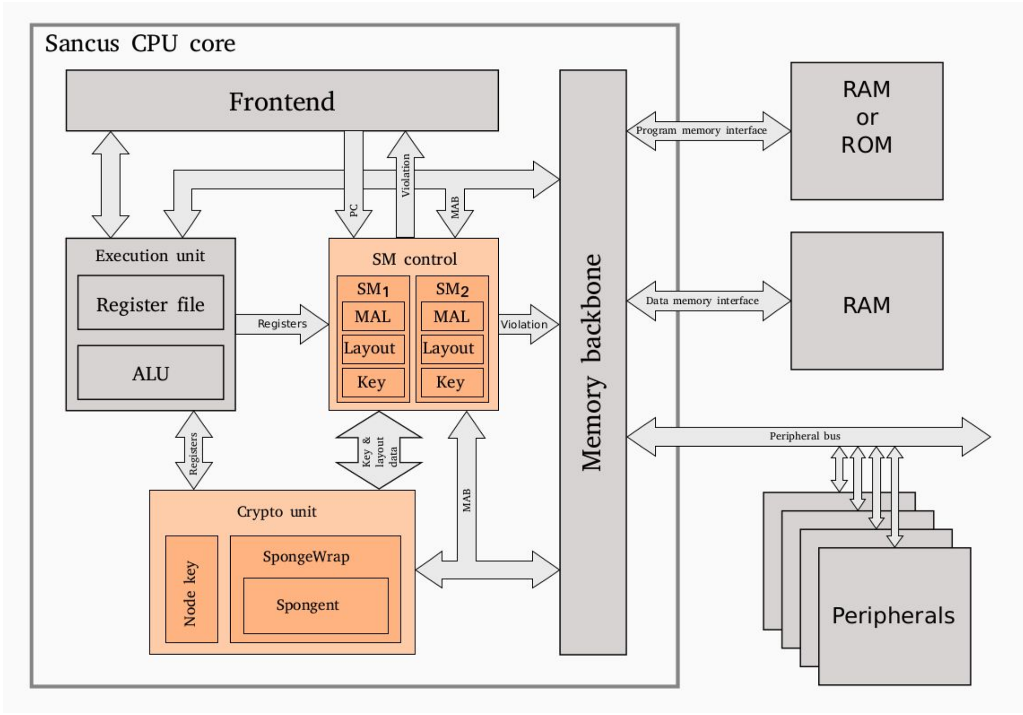
Sancus: Open-Source Trusted Computing for the IoT

Embedded enclaved execution:

- Isolation & attestation
- Save + clear CPU state on interrupt

Small CPU (openMSP430):

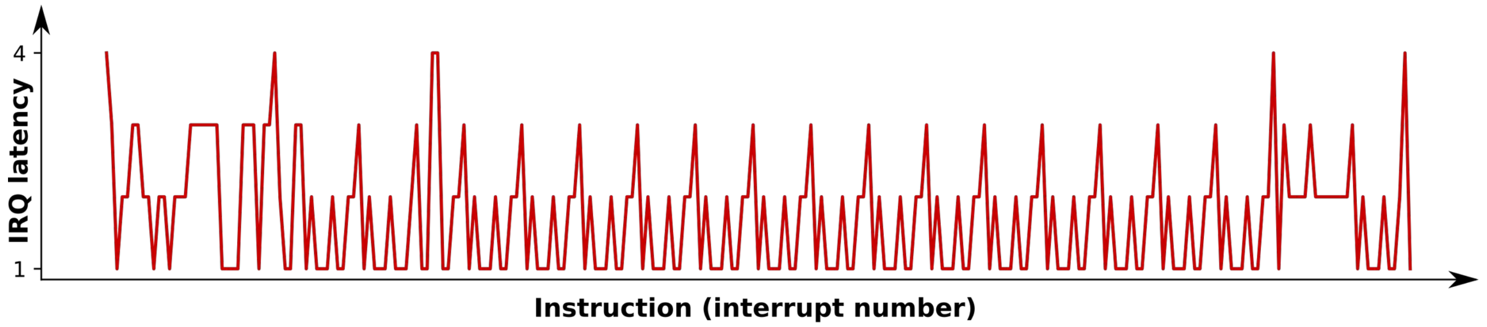
- Area: ≤ 2 kLUTs
- Deterministic execution: *no pipeline/cache/MMU/...*
- Research vehicle for rapid prototyping of attacks & mitigations

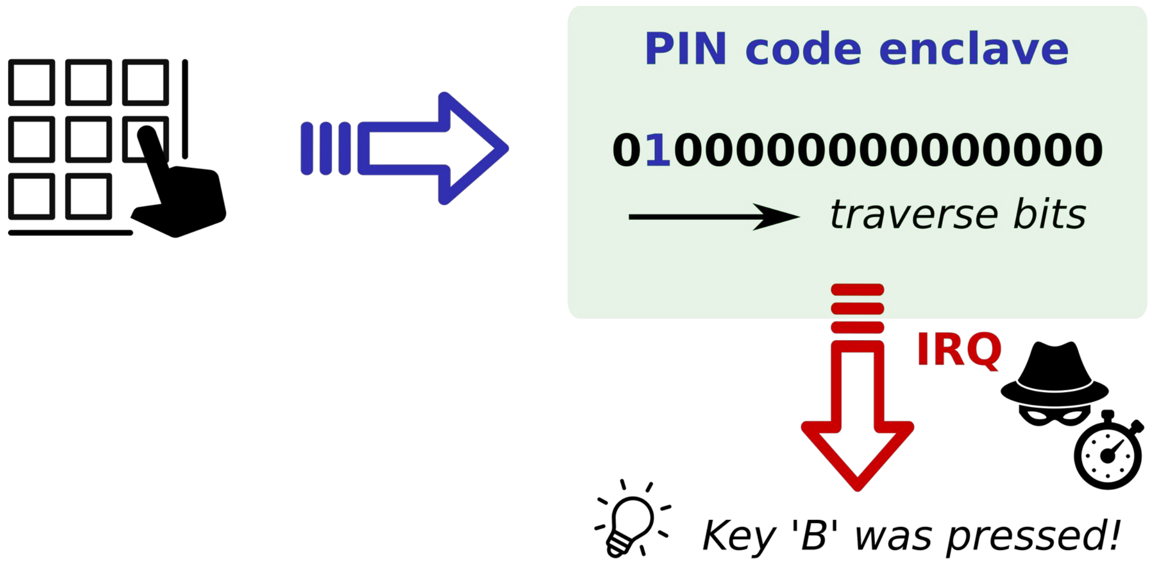


https://github.com/sancus-tee

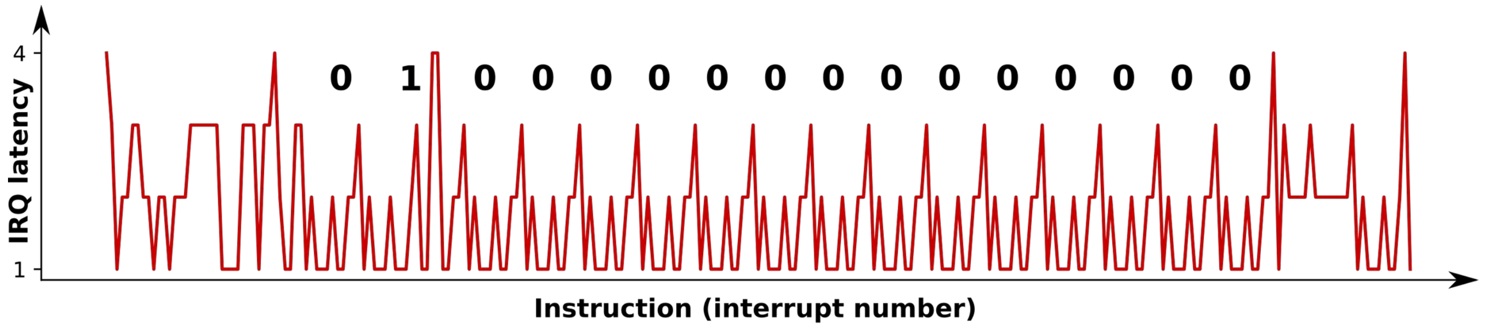
Noorman et al. Sancus 2.0: A Low-Cost Security Architecture for IoT devices. TOPS, 2017

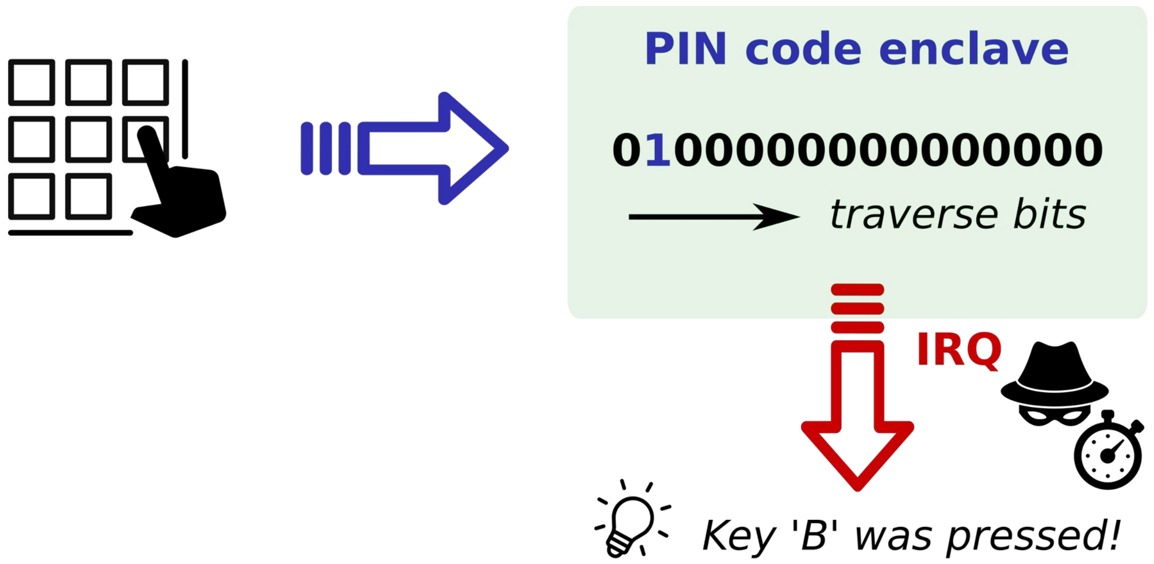
Example: Extracting Keystrokes with Interrupt Latency





Example: Extracting Keystrokes with Interrupt Latency

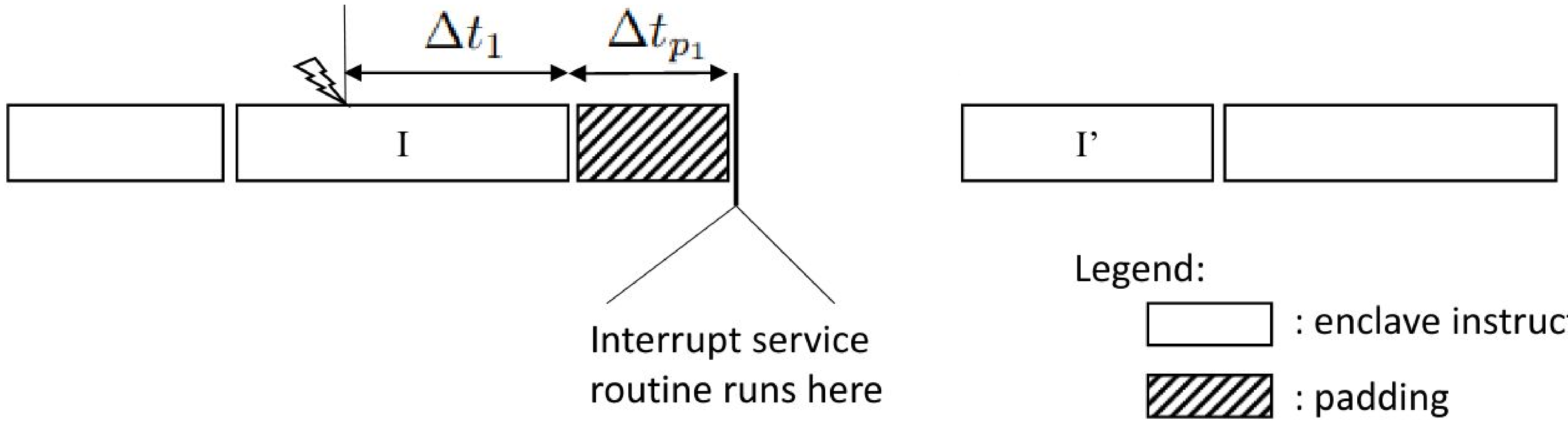




Mitigation Strategy #1: Hardware-Level Padding

Two-stage padding:

1) IRQ latency



Legend:

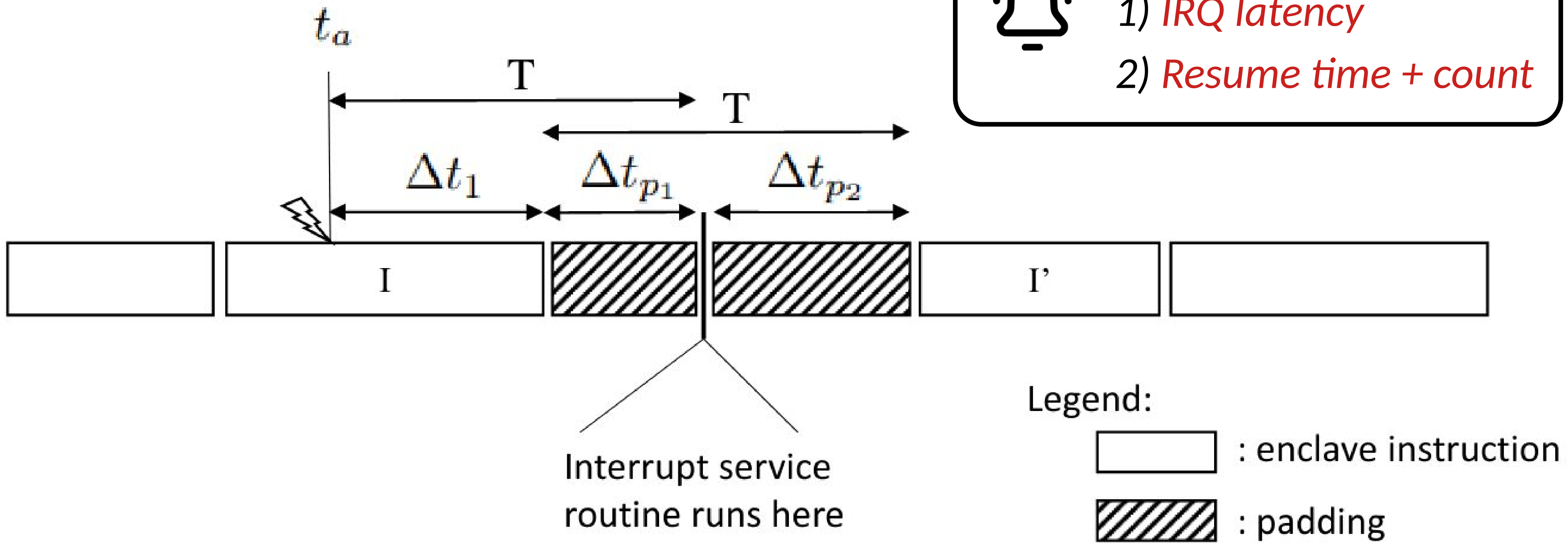
☐	: enclave instruction
▨	: padding

Mitigation Strategy #1: Hardware-Level Padding

Two-stage padding:

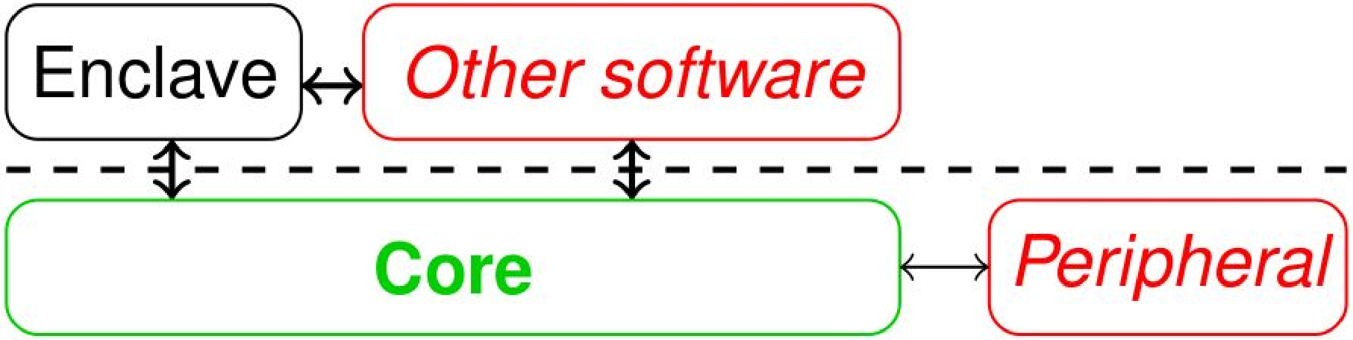
1) IRQ latency

2) Resume time + count



Deductive Validation: Proving Contextual Equivalence

- **Operational semantics:** Sancus$_{H/L}$ w/ and w/o interrupts (~35 pages)
- **Pen-and-paper proof:** *Any attack in Sancus$_L$ can also be expressed in Sancus$_H$ w/o interrupts…*



Busi et al. "Provably Secure Isolation for Interruptible Enclaved Execution on Small Microprocessors", CSF 2020.

Deductive Validation: Proving Contextual Equivalence

(CPU-NOT)

$$\frac{\begin{array}{c}\mathcal{B} \neq \langle \bot, \bot, t_{pad}\rangle \quad i, \mathcal{R}, pc_{old}, \mathcal{B} \vdash_{mac} \texttt{OK} \quad \mathcal{R}' = \mathcal{R}[\mathsf{pc} \mapsto \mathcal{R}[\mathsf{pc}] + 2][\mathtt{r} \mapsto \neg\mathcal{R}[\mathtt{r}]] \\ \mathcal{D} \vdash \delta, t, t_a \curvearrowright_D^{cycles(i)} \delta', t', t'_a \quad \mathcal{D} \vdash \langle \delta', t', t'_a, \mathcal{M}, \mathcal{R}', \mathcal{R}[\mathsf{pc}], \mathcal{B}\rangle \hookrightarrow_{\mathsf{I}} \langle \delta'', t'', t''_a, \mathcal{M}', \mathcal{R}'', \mathcal{R}[\mathsf{pc}], \mathcal{B}'\rangle\end{array}}{\mathcal{D} \vdash \langle \delta, t, t_a, \mathcal{M}, \mathcal{R}, pc_{old}, \mathcal{B}\rangle \to \langle \delta'', t'', t''_a, \mathcal{M}', \mathcal{R}'', \mathcal{R}[\mathsf{pc}], \mathcal{B}'\rangle} \; i = decode(\mathcal{M}, \mathcal{R}[\mathsf{pc}]) = \texttt{NOT r}$$

(CPU-AND)

$$\frac{\begin{array}{c}\mathcal{B} \neq \langle \bot, \bot, t_{pad}\rangle \quad i, \mathcal{R}, pc_{old}, \mathcal{B} \vdash_{mac} \texttt{OK} \quad \mathcal{R}' = \mathcal{R}[\mathsf{pc} \mapsto \mathcal{R}[\mathsf{pc}] + 2][\mathtt{r}_2 \mapsto \mathcal{R}[\mathtt{r}_1]\&\mathcal{R}[\mathtt{r}_2]] \\ \mathcal{R}'' = \mathcal{R}'[\mathtt{sr.N} \mapsto \mathcal{R}'[\mathtt{r}_2]\&\mathtt{0x8000}, \mathtt{sr.Z} \mapsto (\mathcal{R}'[\mathtt{r}_2] == 0), \mathtt{sr.C} \mapsto (\mathcal{R}'[\mathtt{r}_2] \neq 0), \mathtt{sr.V} \mapsto 0] \\ \mathcal{D} \vdash \delta, t, t_a \curvearrowright_D^{cycles(i)} \delta', t', t'_a \quad \mathcal{D} \vdash \langle \delta', t', t'_a, \mathcal{M}, \mathcal{R}'', \mathcal{R}[\mathsf{pc}], \mathcal{B}\rangle \hookrightarrow_{\mathsf{I}} \langle \delta'', t'', t''_a, \mathcal{M}', \mathcal{R}''', \mathcal{R}[\mathsf{pc}], \mathcal{B}'\rangle\end{array}}{\mathcal{D} \vdash \langle \delta, t, t_a, \mathcal{M}, \mathcal{R}, pc_{old}, \mathcal{B}\rangle \to \langle \delta'', t'', t''_a, \mathcal{M}', \mathcal{R}''', \mathcal{R}[\mathsf{pc}], \mathcal{B}'\rangle} \; i = decode(\mathcal{M}, \mathcal{R}[\mathsf{pc}]) = \texttt{AND}\ \mathtt{r}_1\ \mathtt{r}_2$$

(CPU-CMP)

$$\frac{\begin{array}{c}\mathcal{B} \neq \langle \bot, \bot, t_{pad}\rangle \quad i, \mathcal{R}, pc_{old}, \mathcal{B} \vdash_{mac} \texttt{OK} \quad \mathcal{R}' = \mathcal{R}[\mathsf{pc} \mapsto \mathcal{R}[\mathsf{pc}] + 2][\mathtt{r}_2 \mapsto \mathcal{R}[\mathtt{r}_1] - \mathcal{R}[\mathtt{r}_2]] \\ \mathcal{R}'' = \mathcal{R}'[\mathtt{sr.N} \mapsto (\mathcal{R}'[\mathtt{r}_2] < 0), \mathtt{sr.Z} \mapsto (\mathcal{R}'[\mathtt{r}_2] == 0), \mathtt{sr.C} \mapsto (\mathcal{R}'[\mathtt{r}_2] \neq 0), \mathtt{sr.V} \mapsto overflow(\mathcal{R}[\mathtt{r}_1] - \mathcal{R}[\mathtt{r}_2])] \\ \mathcal{D} \vdash \delta, t, t_a \curvearrowright_D^{cycles(i)} \delta', t', t'_a \quad \mathcal{D} \vdash \langle \delta', t', t'_a, \mathcal{M}, \mathcal{R}'', \mathcal{R}[\mathsf{pc}], \mathcal{B}\rangle \hookrightarrow_{\mathsf{I}} \langle \delta'', t'', t''_a, \mathcal{M}', \mathcal{R}''', \mathcal{R}[\mathsf{pc}], \mathcal{B}'\rangle\end{array}}{\mathcal{D} \vdash \langle \delta, t, t_a, \mathcal{M}, \mathcal{R}, pc_{old}, \mathcal{B}\rangle \to \langle \delta'', t'', t''_a, \mathcal{M}', \mathcal{R}''', \mathcal{R}[\mathsf{pc}], \mathcal{B}'\rangle} \; i = decode(\mathcal{M}, \mathcal{R}[\mathsf{pc}]) = \texttt{CMP}\ \mathtt{r}_1\ \mathtt{r}_2$$

(CPU-ADD)

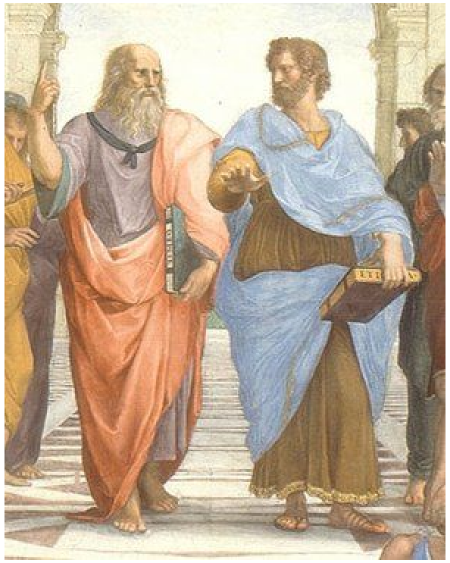
$$\frac{\begin{array}{c}\mathcal{B} \neq \langle \bot, \bot, t_{pad}\rangle \quad i, \mathcal{R}, pc_{old}, \mathcal{B} \vdash_{mac} \texttt{OK} \quad \mathcal{R}' = \mathcal{R}[\mathsf{pc} \mapsto \mathcal{R}[\mathsf{pc}] + 2][\mathtt{r}_2 \mapsto \mathcal{R}[\mathtt{r}_1] + \mathcal{R}[\mathtt{r}_2]] \\ \mathcal{R}'' = \mathcal{R}'[\mathtt{sr.N} \mapsto (\mathcal{R}'[\mathtt{r}_2] < 0), \mathtt{sr.Z} \mapsto (\mathcal{R}'[\mathtt{r}_2] == 0), \mathtt{sr.C} \mapsto (\mathcal{R}'[\mathtt{r}_2] \neq 0), \mathtt{sr.V} \mapsto overflow(\mathcal{R}[\mathtt{r}_1] + \mathcal{R}[\mathtt{r}_2])] \\ \mathcal{D} \vdash \delta, t, t_a \curvearrowright_D^{cycles(i)} \delta', t', t'_a \quad \mathcal{D} \vdash \langle \delta', t', t'_a, \mathcal{M}, \mathcal{R}'', \mathcal{R}[\mathsf{pc}], \mathcal{B}\rangle \hookrightarrow_{\mathsf{I}} \langle \delta'', t'', t''_a, \mathcal{M}', \mathcal{R}''', \mathcal{R}[\mathsf{pc}], \mathcal{B}'\rangle\end{array}}{\mathcal{D} \vdash \langle \delta, t, t_a, \mathcal{M}, \mathcal{R}, pc_{old}, \mathcal{B}\rangle \to \langle \delta'', t'', t''_a, \mathcal{M}', \mathcal{R}''', \mathcal{R}[\mathsf{pc}], \mathcal{B}'\rangle} \; i = decode(\mathcal{M}, \mathcal{R}[\mathsf{pc}]) = \texttt{ADD}\ \mathtt{r}_1\ \mathtt{r}_2$$

(CPU-SUB)

$$\frac{\begin{array}{c}\mathcal{B} \neq \langle \bot, \bot, t_{pad}\rangle \quad i, \mathcal{R}, pc_{old}, \mathcal{B} \vdash_{mac} \texttt{OK} \quad \mathcal{R}' = \mathcal{R}[\mathsf{pc} \mapsto \mathcal{R}[\mathsf{pc}] + 2][\mathtt{r}_2 \mapsto \mathcal{R}[\mathtt{r}_1] - \mathcal{R}[\mathtt{r}_2]] \\ \mathcal{R}'' = \mathcal{R}'[\mathtt{sr.N} \mapsto (\mathcal{R}'[\mathtt{r}_2] < 0), \mathtt{sr.Z} \mapsto (\mathcal{R}'[\mathtt{r}_2] == 0), \mathtt{sr.C} \mapsto (\mathcal{R}'[\mathtt{r}_2] \neq 0), \mathtt{sr.V} \mapsto overflow(\mathcal{R}[\mathtt{r}_1] - \mathcal{R}[\mathtt{r}_2])] \\ \mathcal{D} \vdash \delta, t, t_a \curvearrowright_D^{cycles(i)} \delta', t', t'_a \quad \mathcal{D} \vdash \langle \delta', t', t'_a, \mathcal{M}, \mathcal{R}'', \mathcal{R}[\mathsf{pc}], \mathcal{B}\rangle \hookrightarrow_{\mathsf{I}} \langle \delta'', t'', t''_a, \mathcal{M}', \mathcal{R}''', \mathcal{R}[\mathsf{pc}], \mathcal{B}'\rangle\end{array}}{\mathcal{D} \vdash \langle \delta, t, t_a, \mathcal{M}, \mathcal{R}, pc_{old}, \mathcal{B}\rangle \to \langle \delta'', t'', t''_a, \mathcal{M}', \mathcal{R}''', \mathcal{R}[\mathsf{pc}], \mathcal{B}'\rangle} \; i = decode(\mathcal{M}, \mathcal{R}[\mathsf{pc}]) = \texttt{SUB}\ \mathtt{r}_1\ \mathtt{r}_2$$

Mind the Gap: Studying Model Mismatches

- Inductively study 2 “provably secure” systems (Sancus + VRASED)
- >16 model mismatches/incompleteness



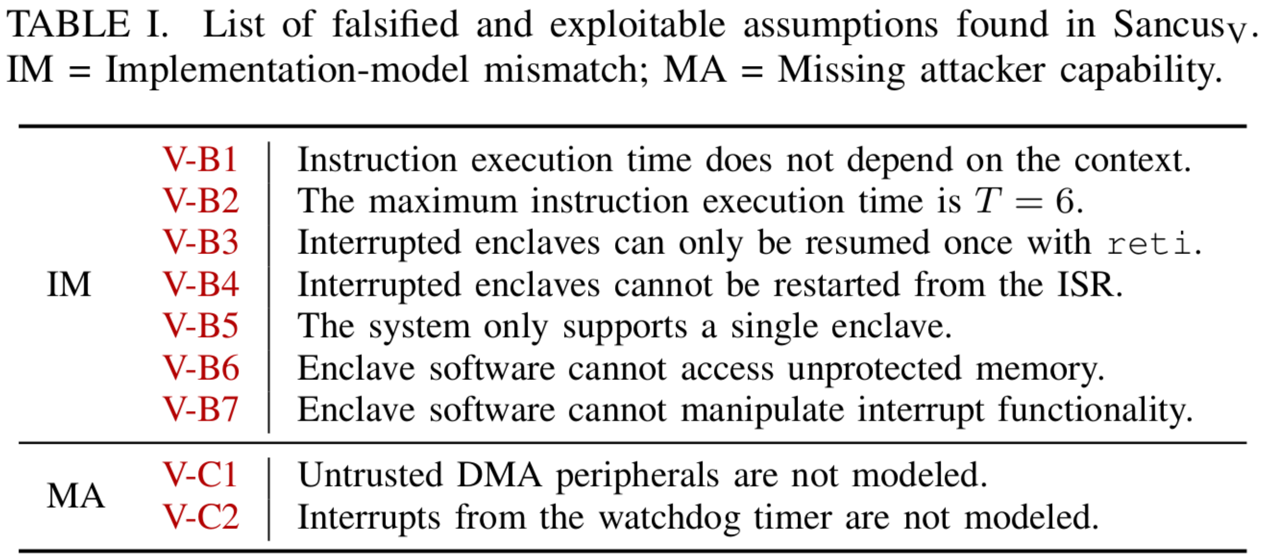
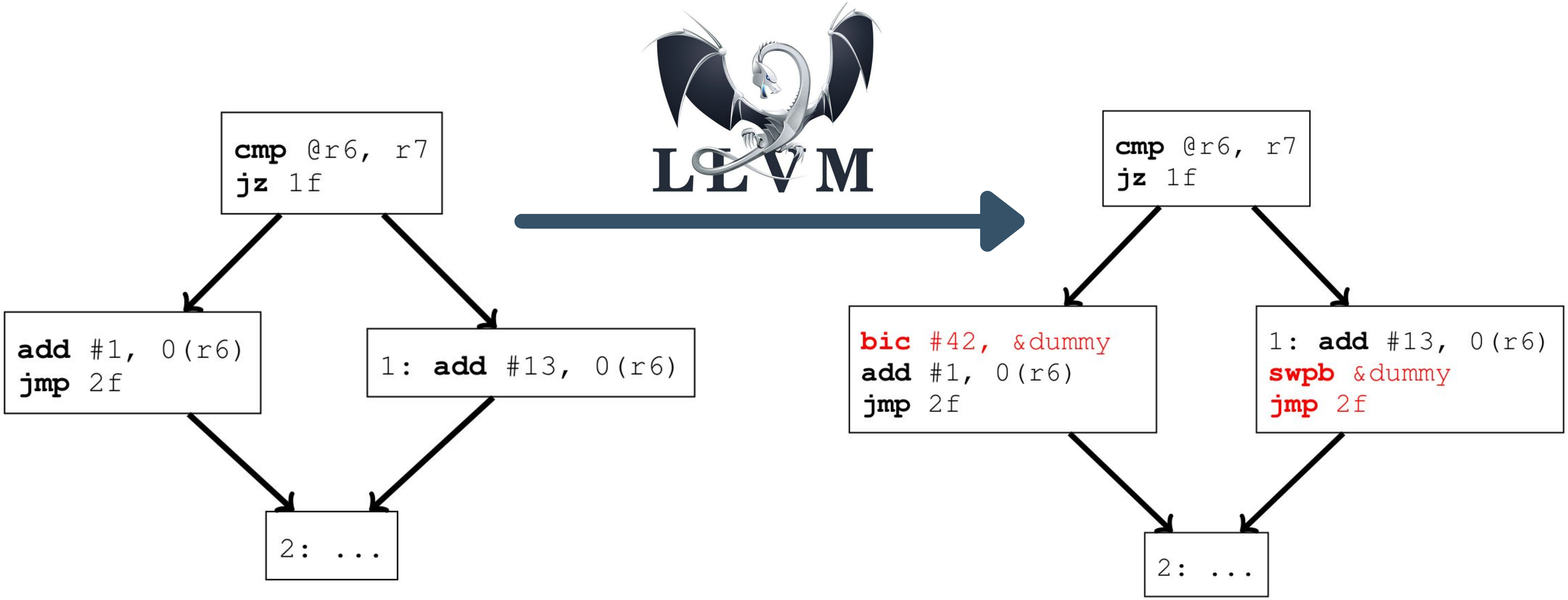


TABLE I. List of falsified and exploitable assumptions found in Sancus_V. IM = Implementation-model mismatch; MA = Missing attacker capability.

IM	V-B1	Instruction execution time does not depend on the context.
	V-B2	The maximum instruction execution time is $T = 6$.
	V-B3	Interrupted enclaves can only be resumed once with `reti`.
	V-B4	Interrupted enclaves cannot be restarted from the ISR.
	V-B5	The system only supports a single enclave.
	V-B6	Enclave software cannot access unprotected memory.
	V-B7	Enclave software cannot manipulate interrupt functionality.
MA	V-C1	Untrusted DMA peripherals are not modeled.
	V-C2	Interrupts from the watchdog timer are not modeled.

Bognar et al. “Mind the Gap: Studying the Insecurity of Provably Secure Embedded Trusted Execution Architectures”, S&P’22.
Busi et al. “Bridging the Gap: Automated Analysis of Sancus”, CSF’24.

Mitigation Strategy #2: Compile-Time Branch Balancing



```
cmp @r6, r7
jz 1f
```

```
add #1, 0(r6)
jmp 2f
```

```
1: add #13, 0(r6)
```

```
2: ...
```

```
cmp @r6, r7
jz 1f
```

```
bic #42, &dummy
add #1, 0(r6)
jmp 2f
```

```
1: add #13, 0(r6)
swpb &dummy
jmp 2f
```

```
2: ...
```

Winderix et al. “Compiler-Assisted Hardening of Embedded Software Against Interrupt Latency Side-Channel Attacks”, EuroS&P 2021.
Bognar et al. “MicroProfiler: Principled Side-Channel Mitigation through Microarchitectural Profiling”, EuroS&P 2023.

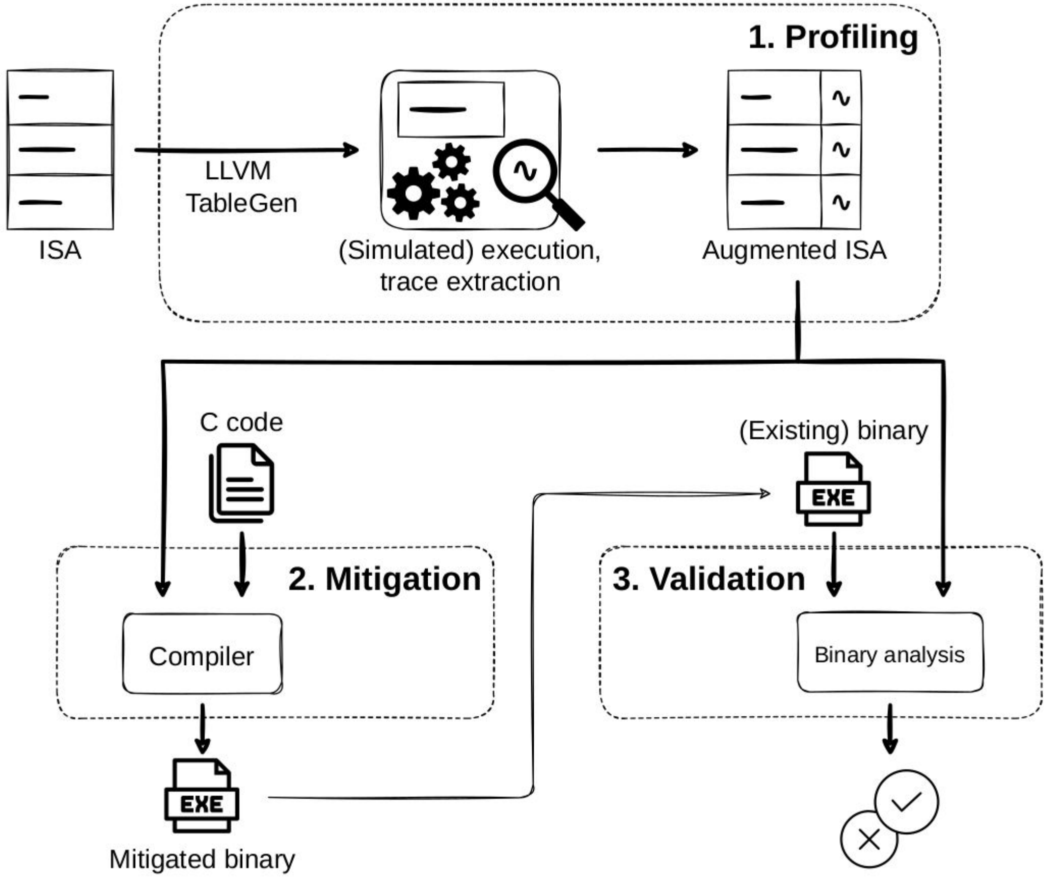
Inductive Validation: Microarchitectural Profiling

"Principled" ISA augmentation:

1) Exhaustively generate *all instructions*
2) Extract leakage model
3) Feed to compiler + binary validator

~ Hardware-software contract

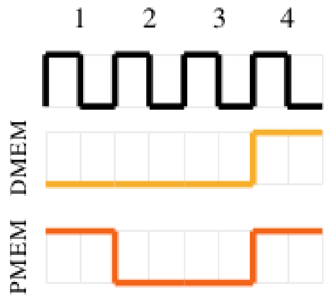
→ *incl. μ-arch timing*

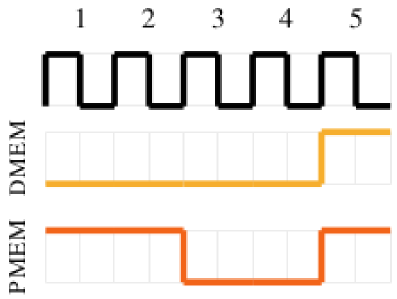


Winderix et al. "Compiler-Assisted Hardening of Embedded Software Against Interrupt Latency Side-Channel Attacks", EuroS&P 2021.
Bognar et al. "MicroProfiler: Principled Side-Channel Mitigation through Microarchitectural Profiling", EuroS&P 2023.

Inductive Validation: Systematic ISA Augmentation

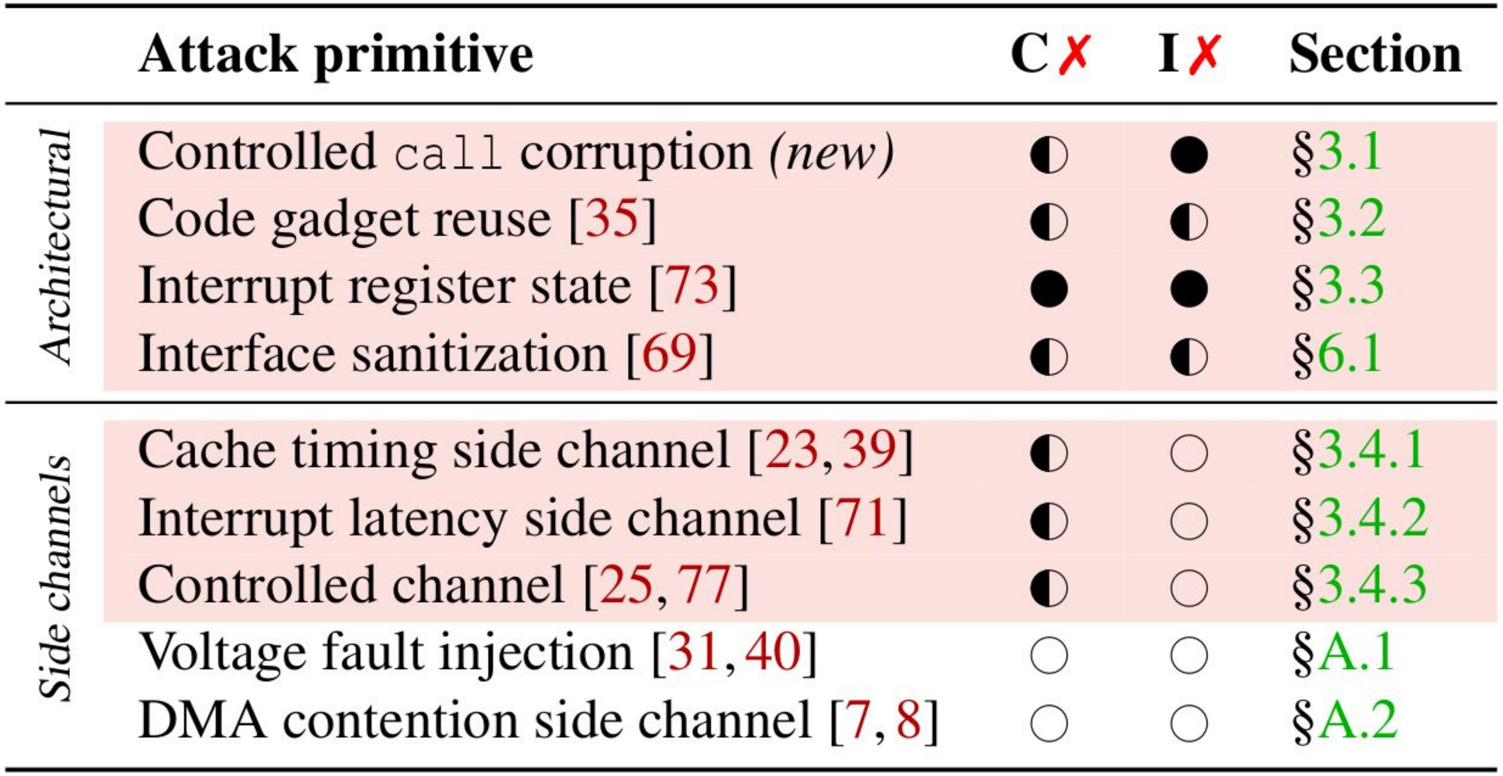
	Dummy and leakage trace	Members

Class #9	
	`mov #1, &dummy`
		`mov #0x1, 42(r6)`
		`mov #0x1, &dmem`
		`mov r10, 42(r6)`
		`mov r10, &dmem`
		`mov.b #0x1, 42(r6)`
		`mov.b #0x1, &dmem`
		`mov.b r10, 42(r6)`
		`mov.b r10, &dmem`
		`push #const`

Class #10	
	`mov #0x42, &dummy`
		`mov #const, 42(r6)`
		`mov #const, &dmem`
		`mov.b #const, 42(r6)`
		`mov.b #const, &dmem`

Bognar et al. “MicroProfiler: Principled Side-Channel Mitigation through Microarchitectural Profiling”, EuroS&P 2023.

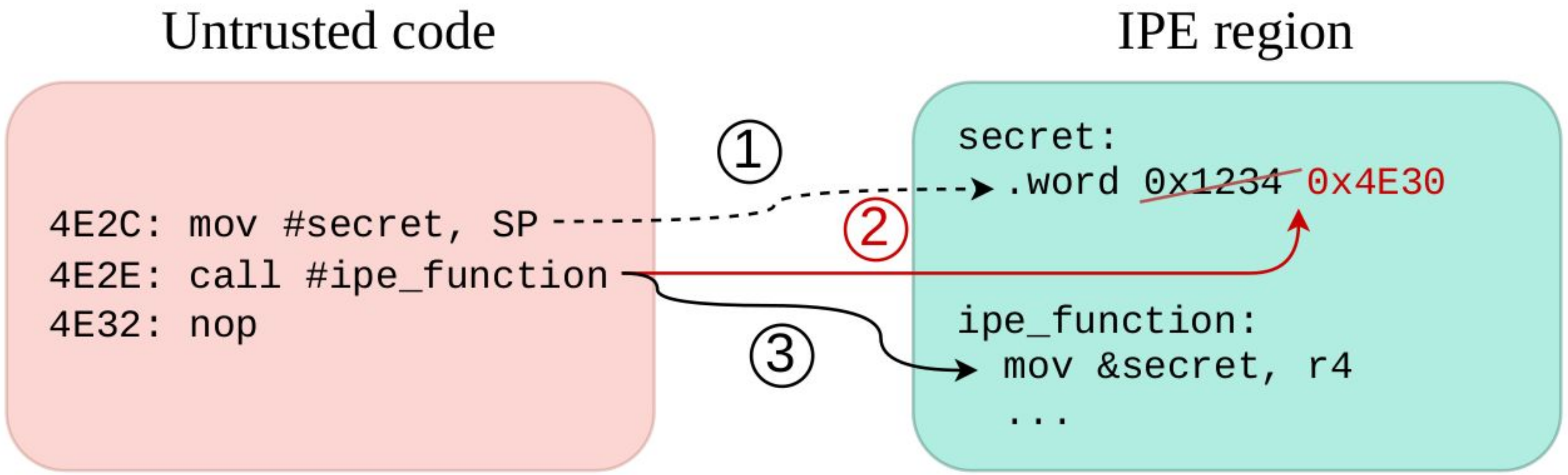
Excurse: Subverting and Securing TI IPE "Enclaves"



	Attack primitive	C ✗	I ✗	Section
Architectural	Controlled `call` corruption *(new)*	◐	●	§3.1
	Code gadget reuse [35]	◐	◐	§3.2
	Interrupt register state [73]	●	●	§3.3
	Interface sanitization [69]	◐	◐	§6.1
Side channels	Cache timing side channel [23, 39]	◐	○	§3.4.1
	Interrupt latency side channel [71]	◐	○	§3.4.2
	Controlled channel [25, 77]	◐	○	§3.4.3
	Voltage fault injection [31, 40]	○	○	§A.1
	DMA contention side channel [7, 8]	○	○	§A.2

Bognar et al. "Intellectual Property Exposure: Subverting and Securing Intellectual Property Encapsulation in Texas Instruments Microcontrollers", USENIX Sec'24.

Excurse: Subverting and Securing TI IPE “Enclaves”



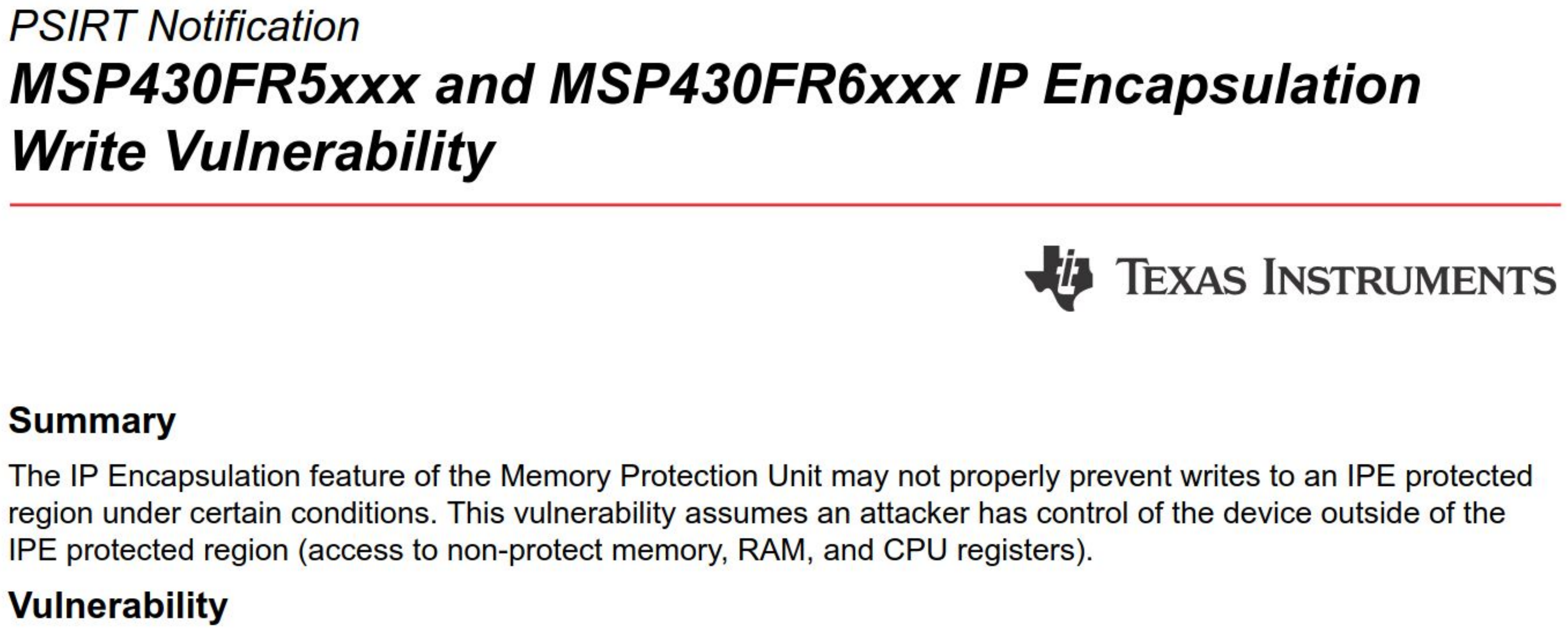
```
4E2C: mov #secret, SP
4E2E: call #ipe_function
4E32: nop
```

```
secret:
  .word ~~0x1234~~ 0x4E30

ipe_function:
  mov &secret, r4
  ...
```

Bognar et al. “Intellectual Property Exposure: Subverting and Securing Intellectual Property Encapsulation in Texas Instruments Microcontrollers”, USENIX Sec’24.

Excurse: Subverting and Securing TI IPE “Enclaves”



PSIRT Notification

MSP430FR5xxx and MSP430FR6xxx IP Encapsulation Write Vulnerability

TEXAS INSTRUMENTS

Summary

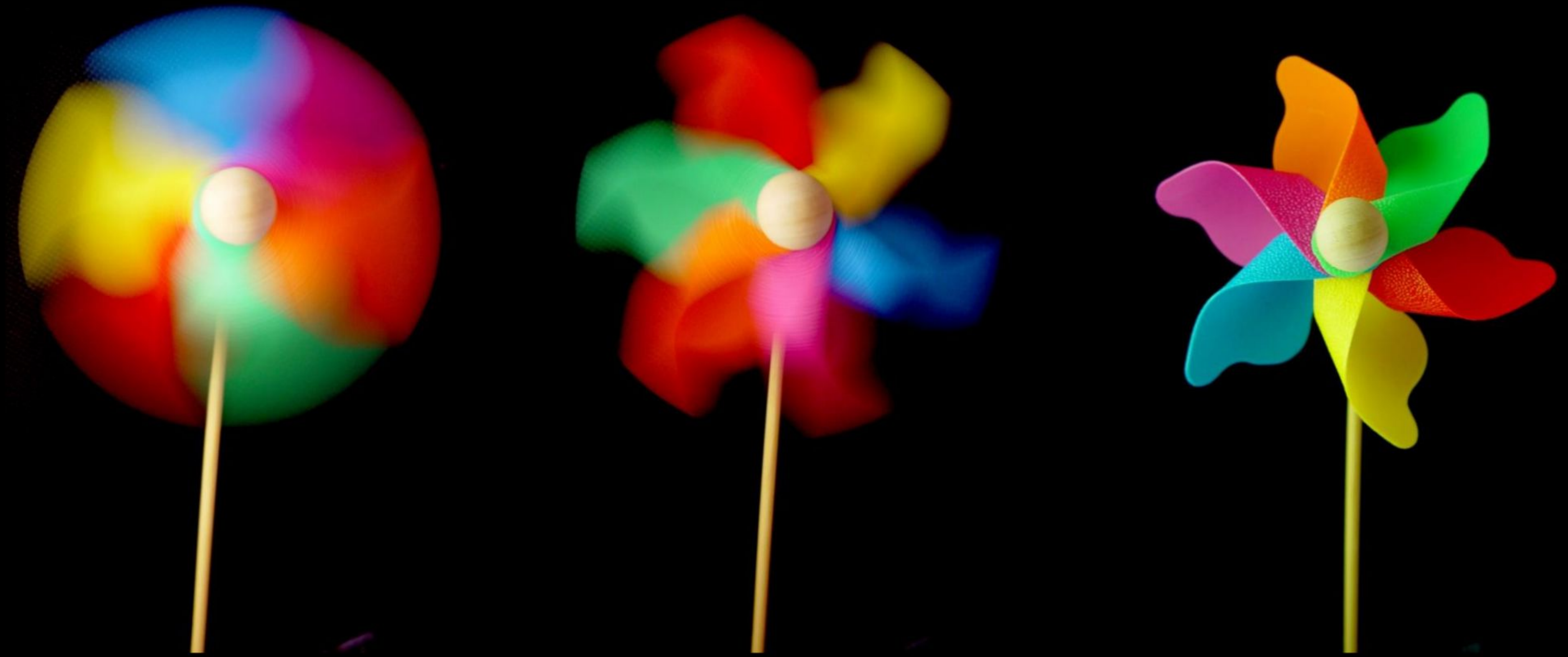
The IP Encapsulation feature of the Memory Protection Unit may not properly prevent writes to an IPE protected region under certain conditions. This vulnerability assumes an attacker has control of the device outside of the IPE protected region (access to non-protect memory, RAM, and CPU registers).

Vulnerability

Bognar et al. “Intellectual Property Exposure: Subverting and Securing Intellectual Property Encapsulation in Texas Instruments Microcontrollers”, USENIX Sec’24.

Reality #2: Larger CPUs?

Challenge: Side-Channel Sampling Rate

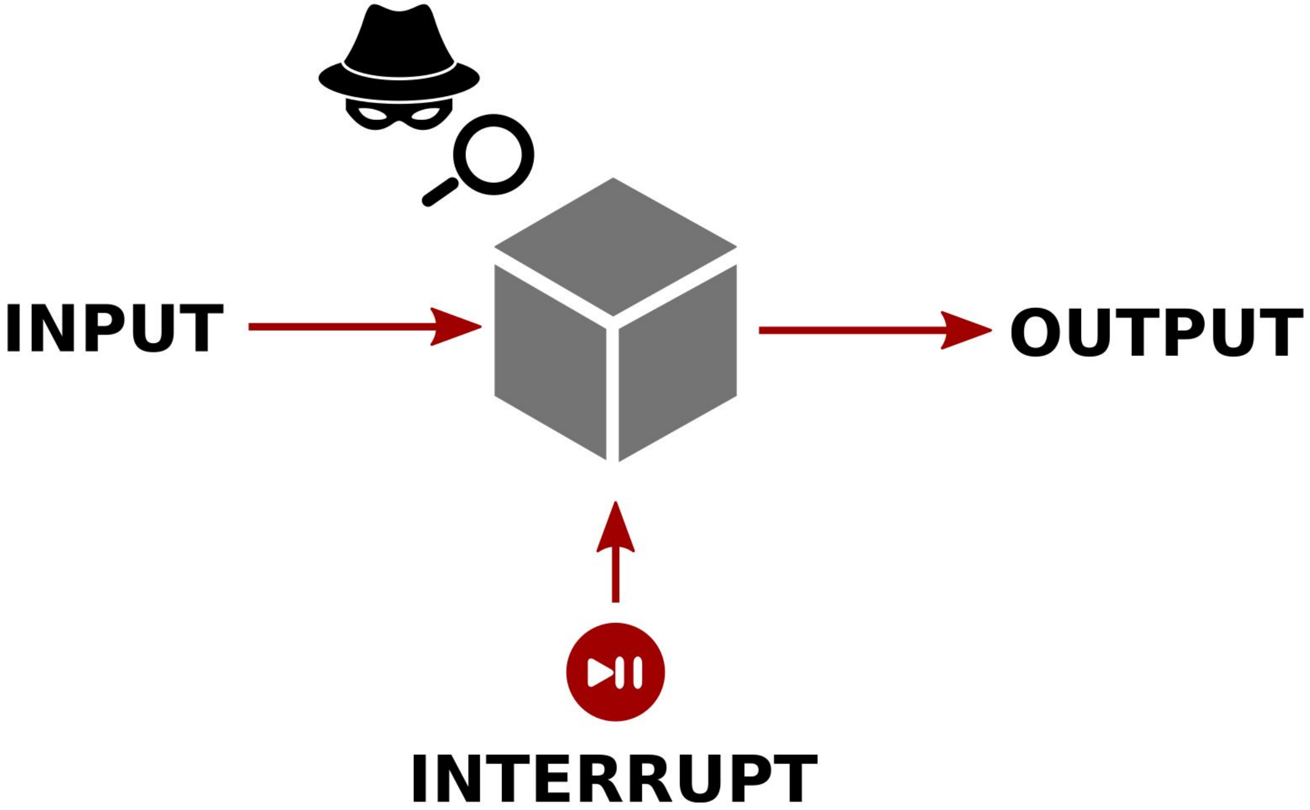


Slow shutter speed

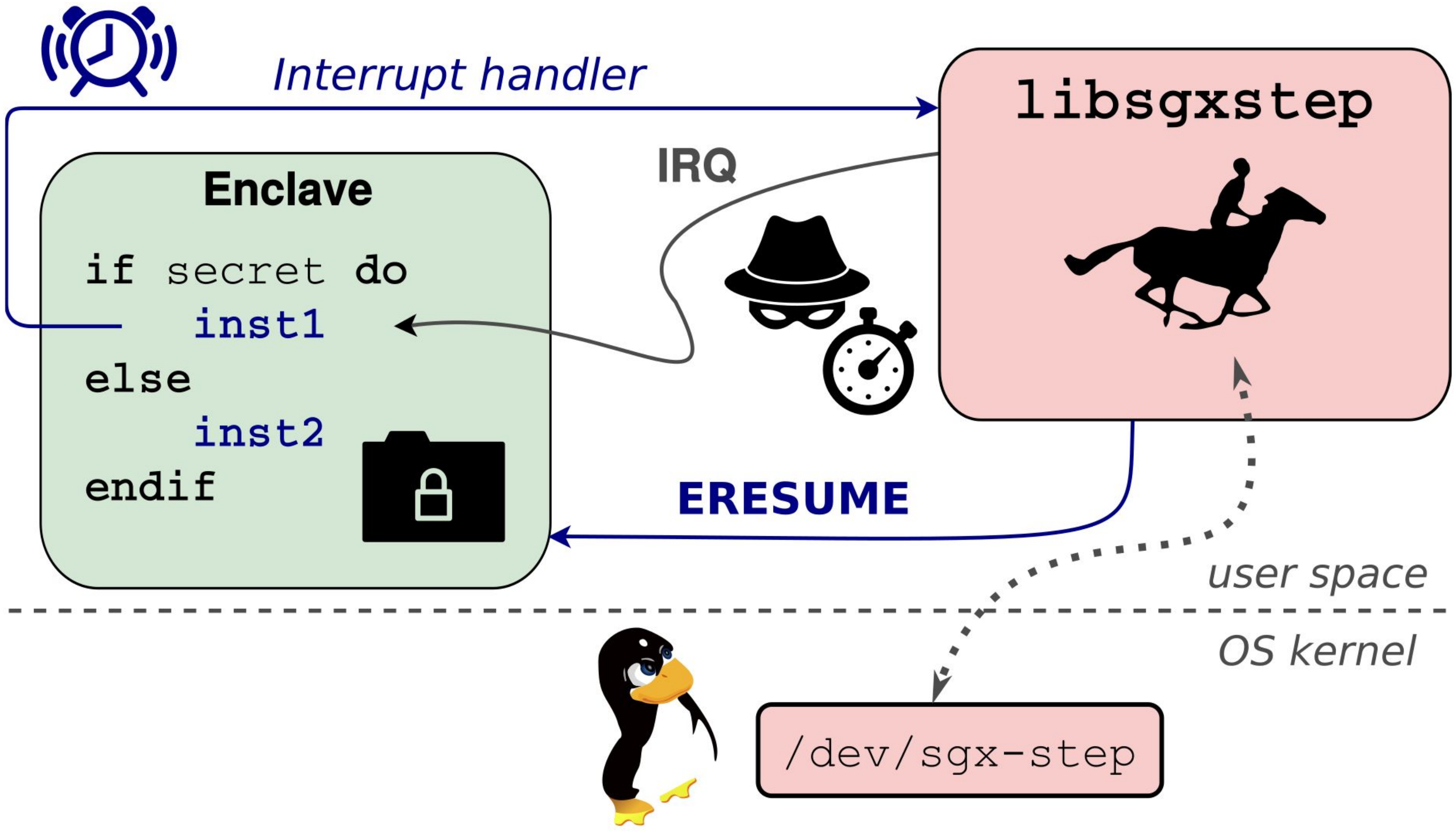
Medium shutter speed

Fast shutter speed

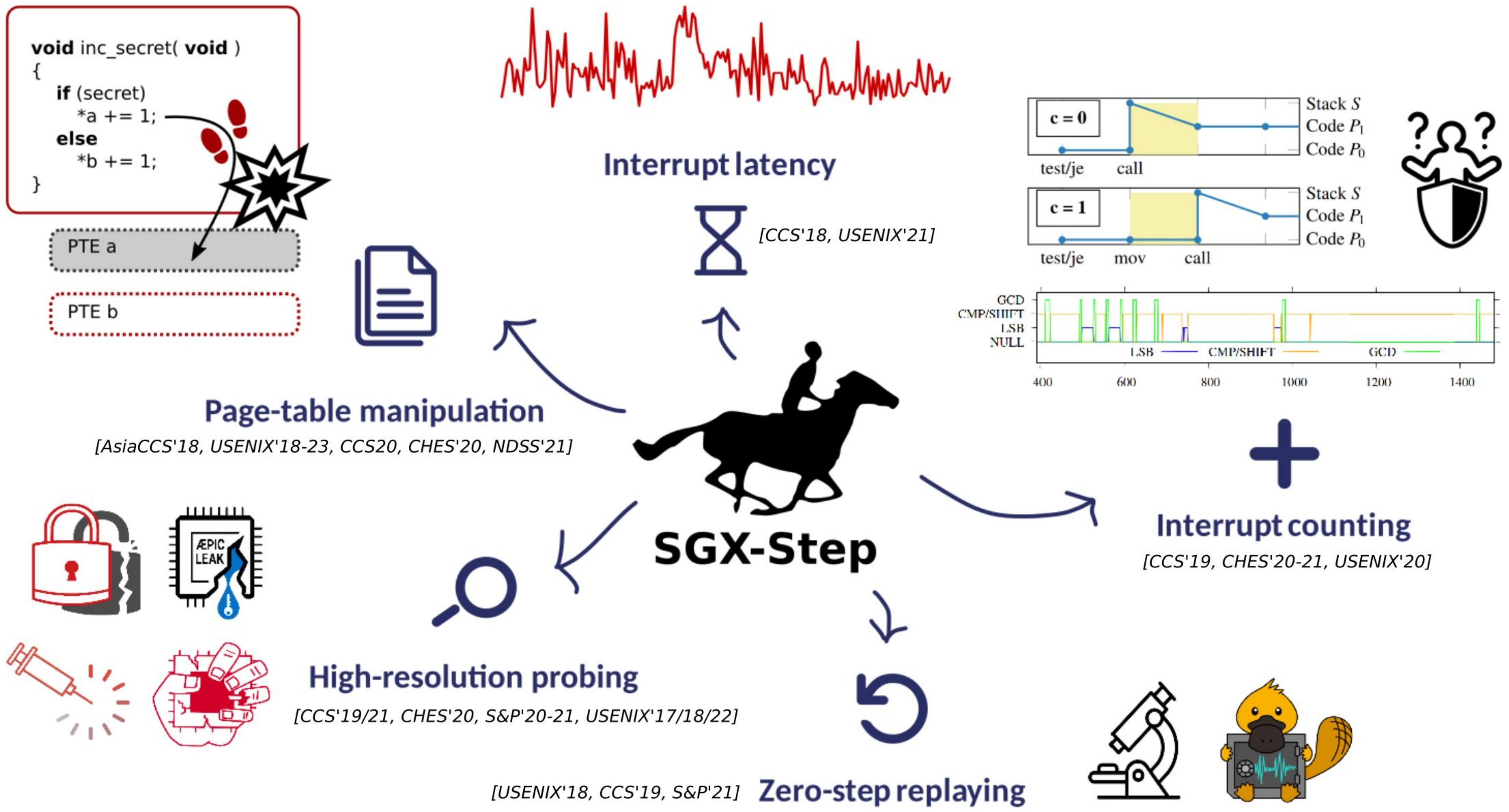
SGX-Step: Executing Enclaves one Instruction at a Time



SGX-Step: Executing Enclaves one Instruction at a Time



SGX-Step: A Versatile Open-Source Attack Toolkit



```
void inc_secret( void )
{
    if (secret)
       *a += 1;
    else
       *b += 1;
}
```

Interrupt latency

[CCS'18, USENIX'21]

Page-table manipulation

[AsiaCCS'18, USENIX'18-23, CCS20, CHES'20, NDSS'21]

Interrupt counting

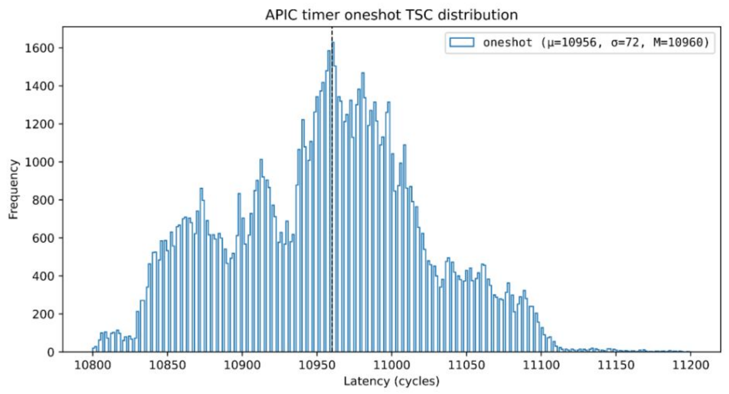
[CCS'19, CHES'20-21, USENIX'20]

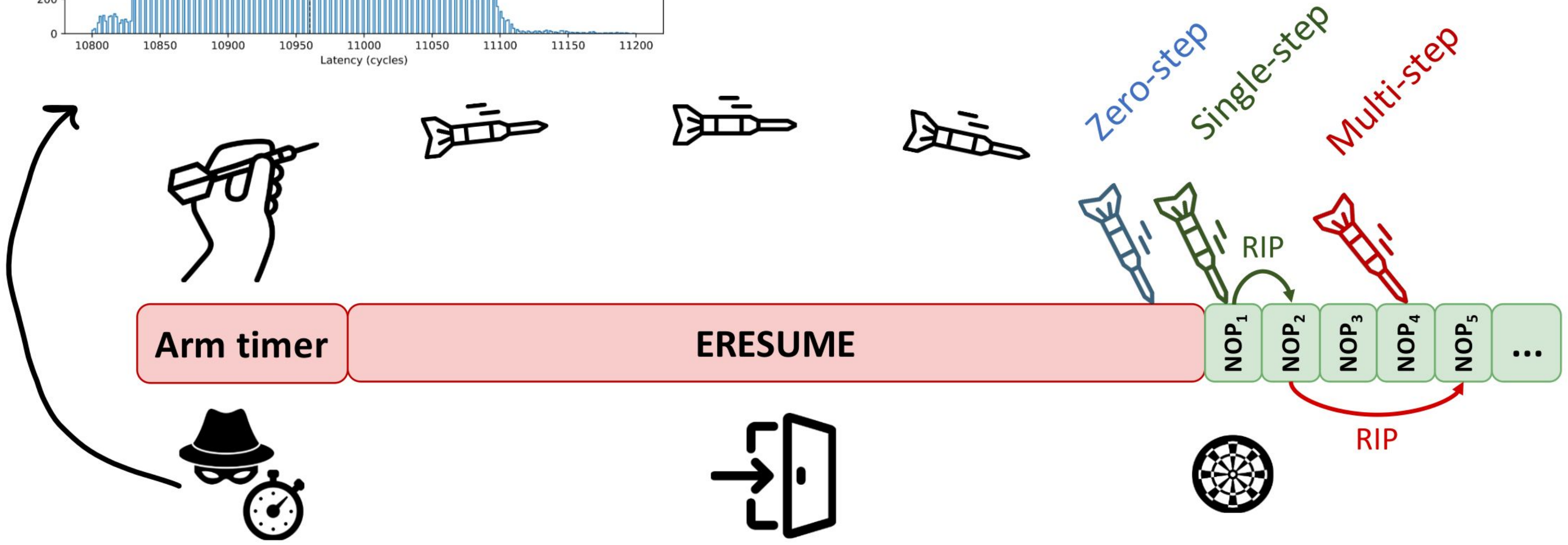
High-resolution probing

[CCS'19/21, CHES'20, S&P'20-21, USENIX'17/18/22]

[USENIX'18, CCS'19, S&P'21] **Zero-step replaying**

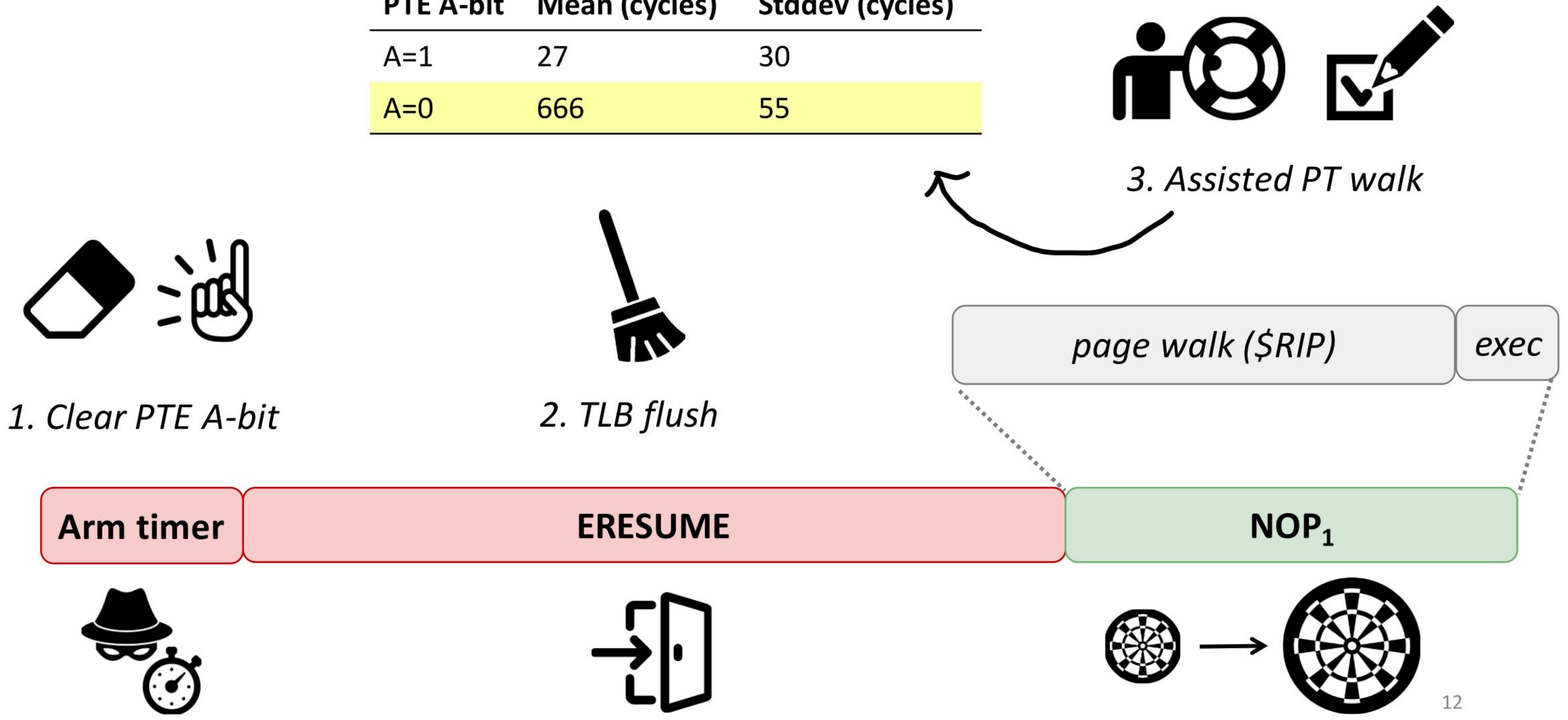
Root-Causing SGX-Step: Aiming the Timer Interrupt



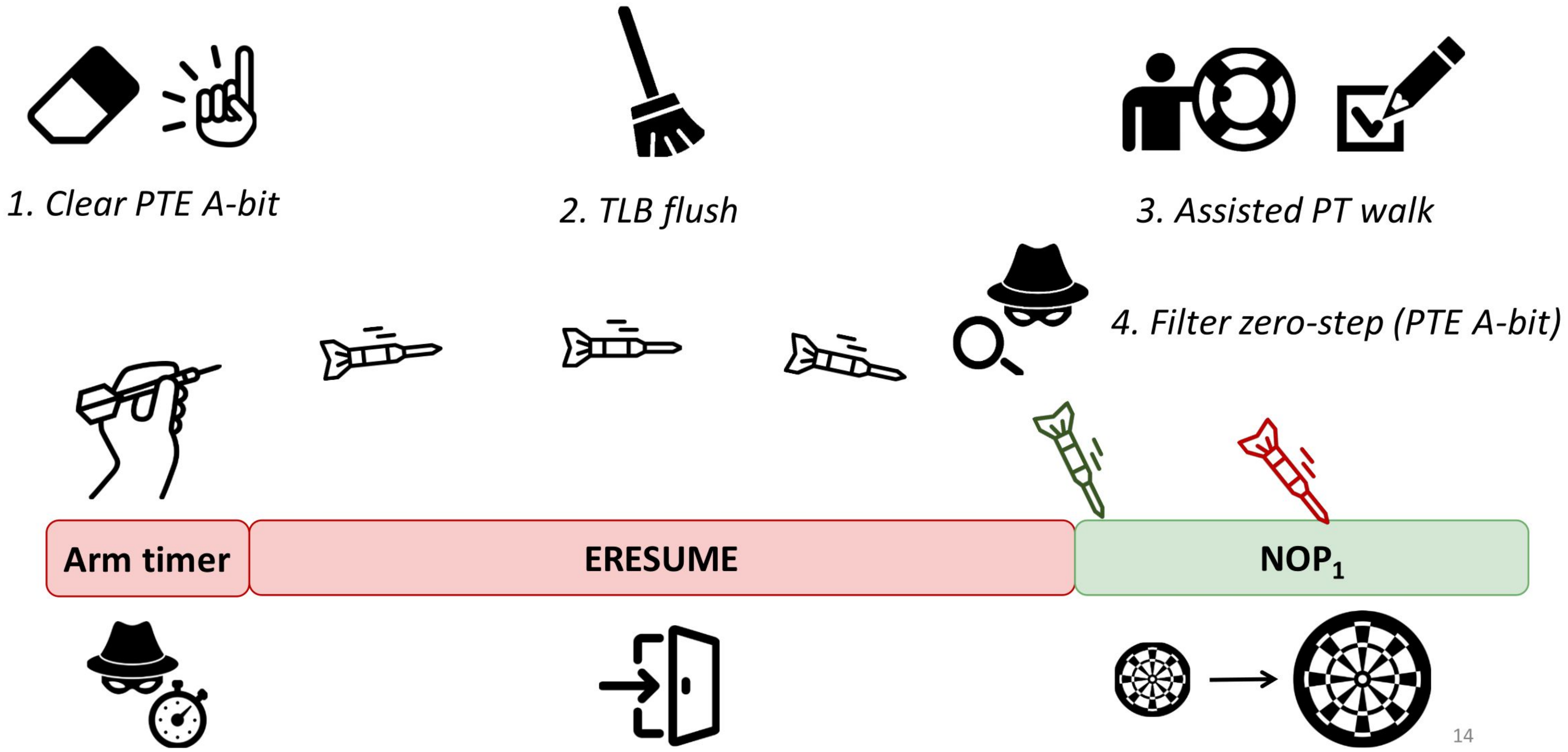


Root-Causing SGX-Step: CPU Microcode Assists

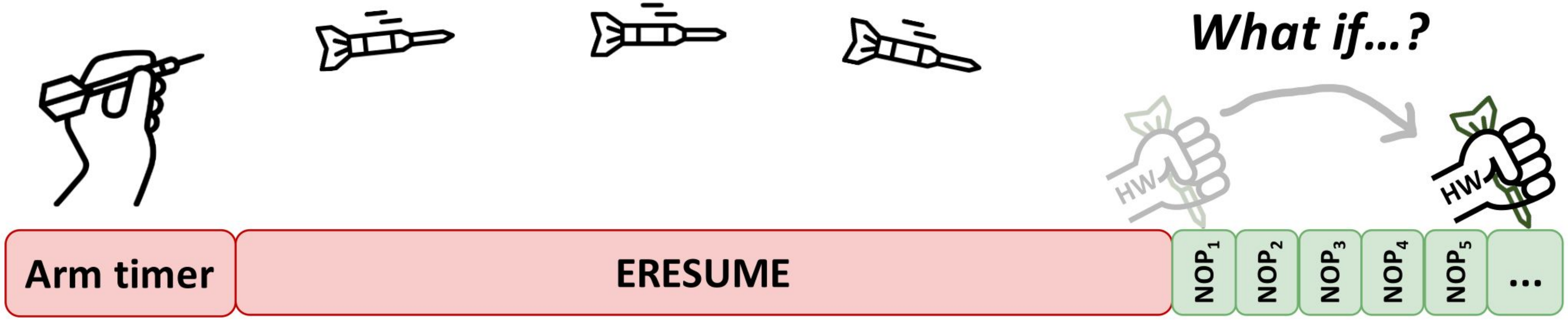
PTE A-bit	Mean (cycles)	Stddev (cycles)
A=1	27	30
A=0	666	55

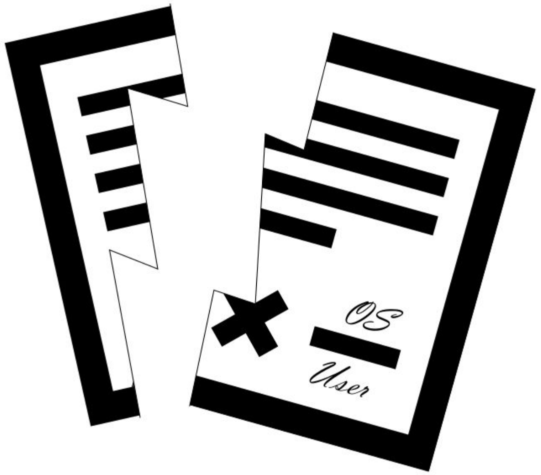


Root-Causing SGX-Step: CPU Microcode Assists



Ideas That Were Rejected (1)

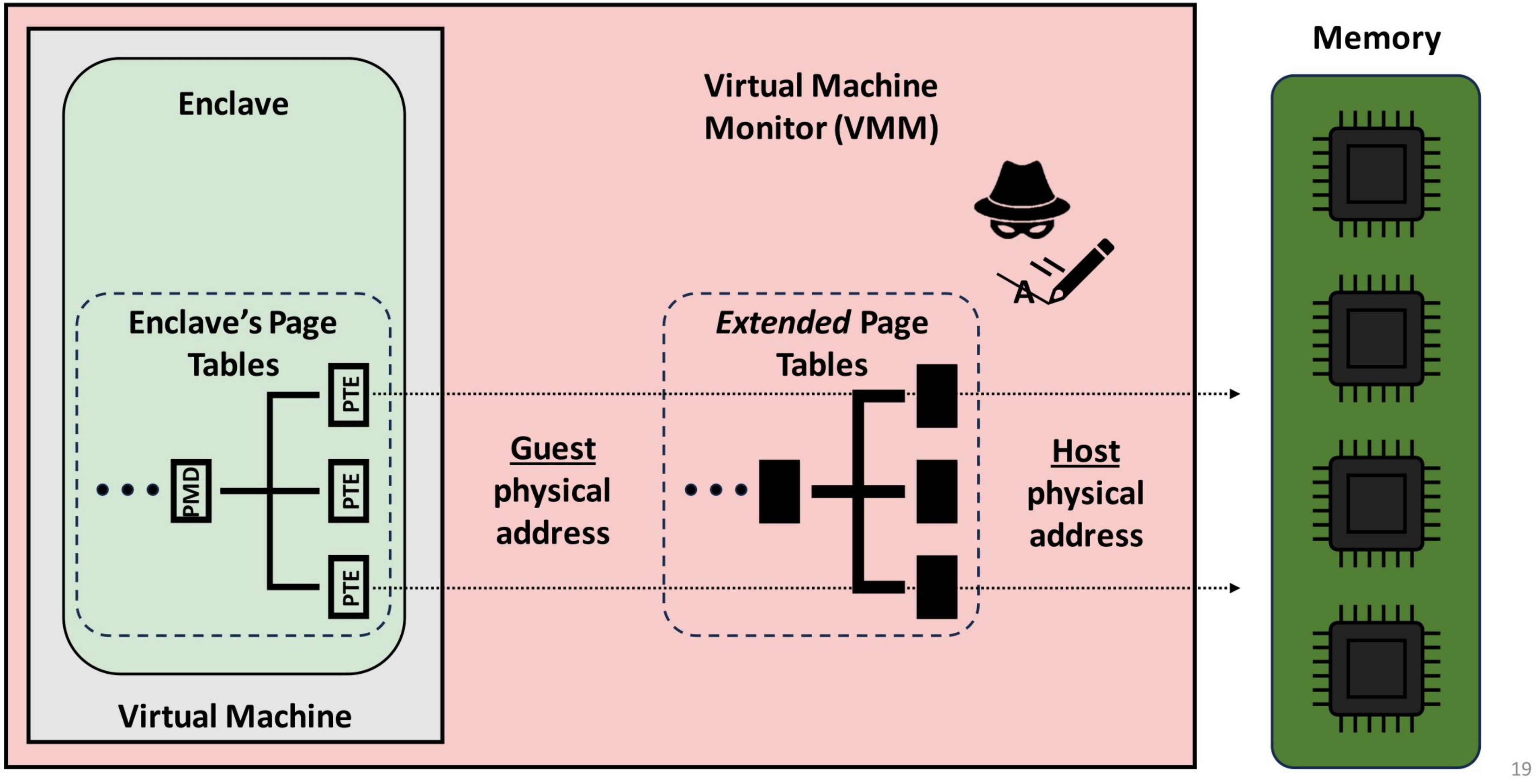


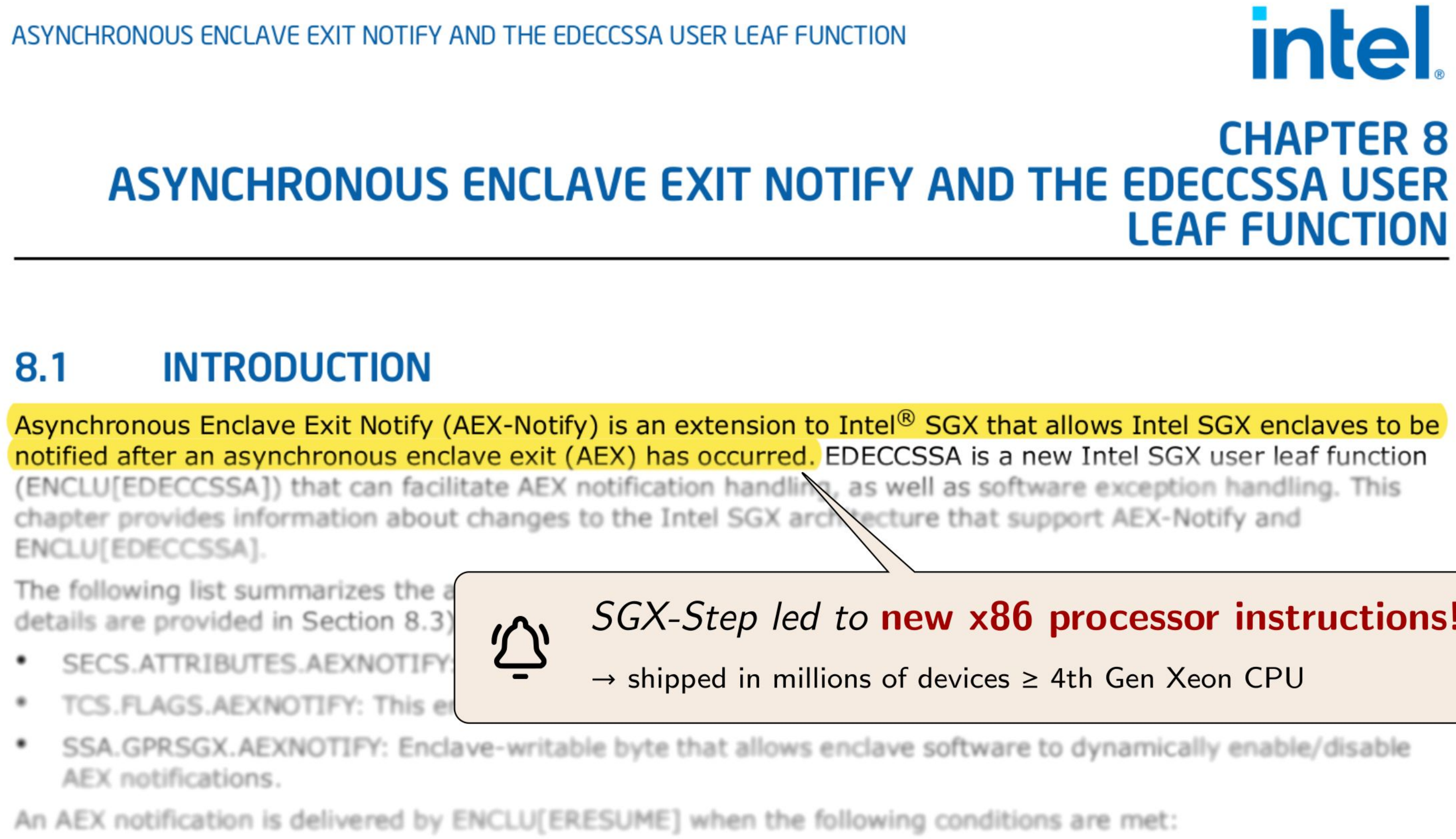


Highly complex

Constable et al. “AEX-Notify: Thwarting Precise Single-Stepping Attacks through Interrupt Awareness for Intel SGX Enclaves”, USENIX Sec’23.

Ideas That Were Rejected (2)





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)

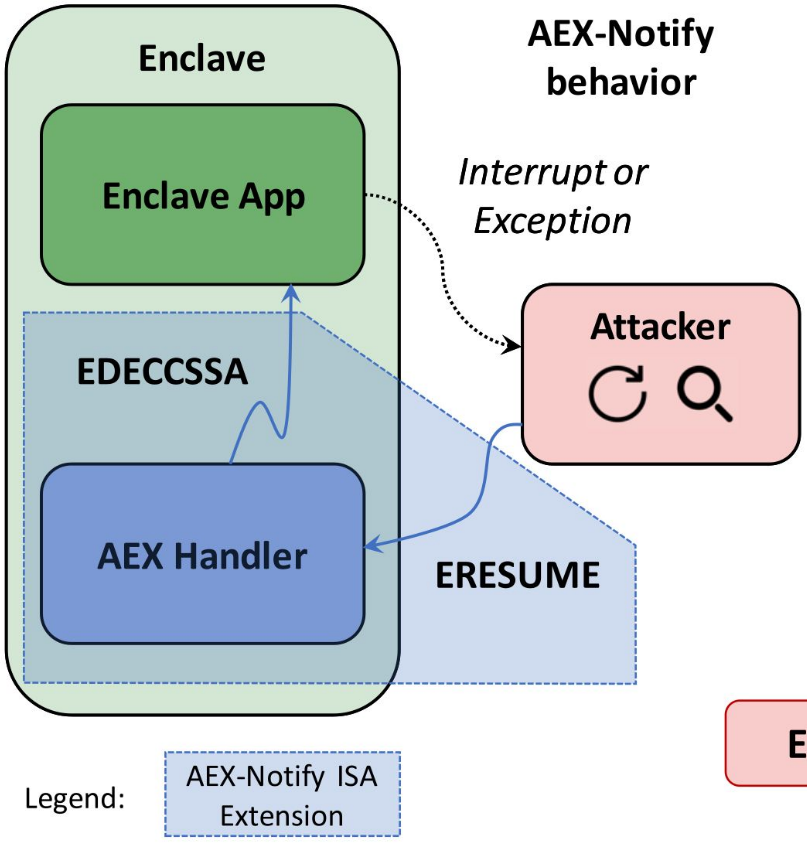
- SECS.ATTRIBUTES.AEXNOTIFY:
- TCS.FLAGS.AEXNOTIFY: This e
- SSA.GPRSGX.AEXNOTIFY: Enclave-writable byte that allows enclave software to dynamically enable/disable AEX notifications.

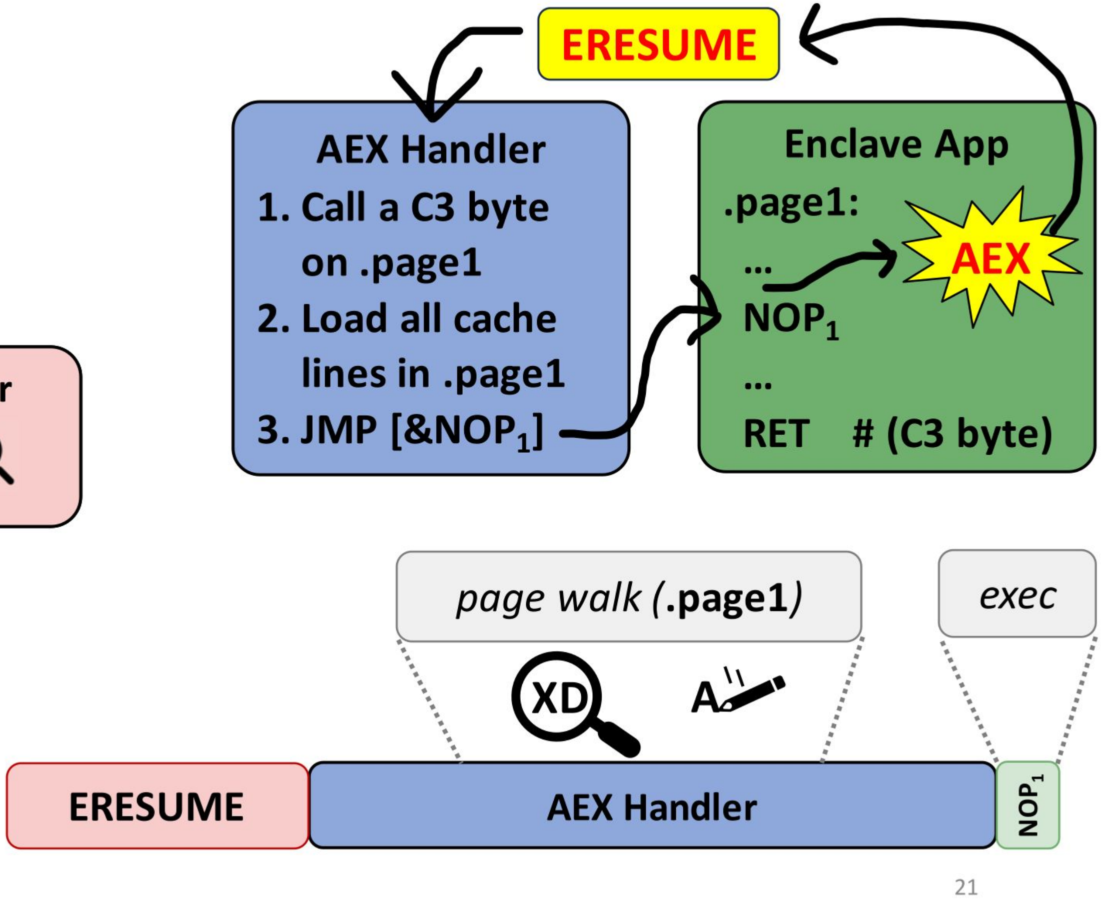
An AEX notification is delivered by ENCLU[ERESUME] when the following conditions are met:

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

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

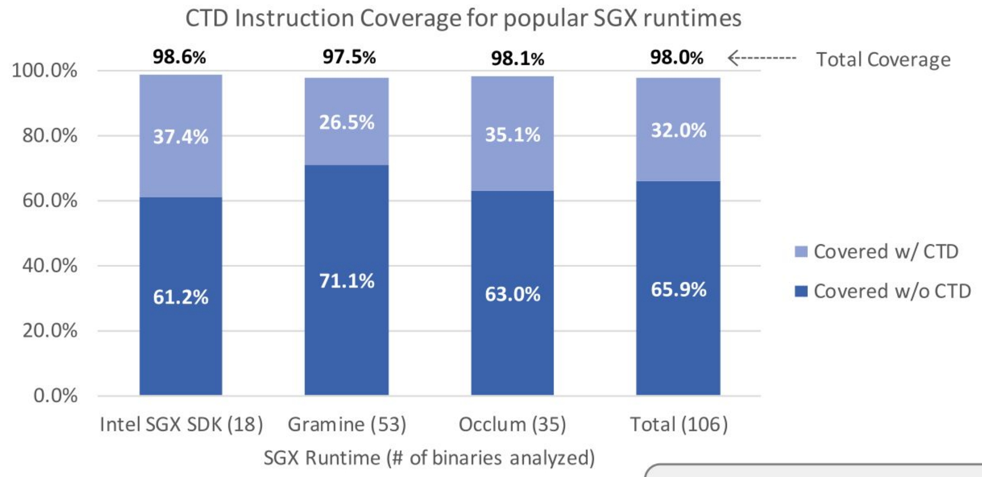
AEX-Notify: Idea Overview

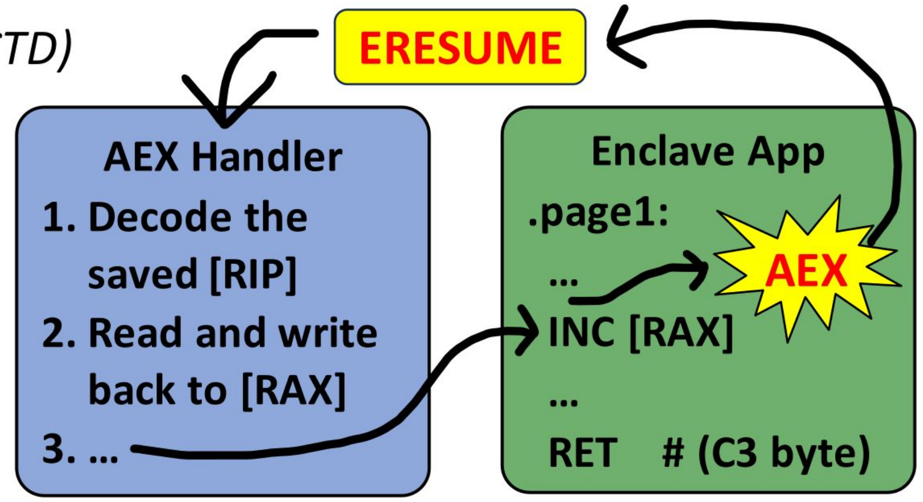


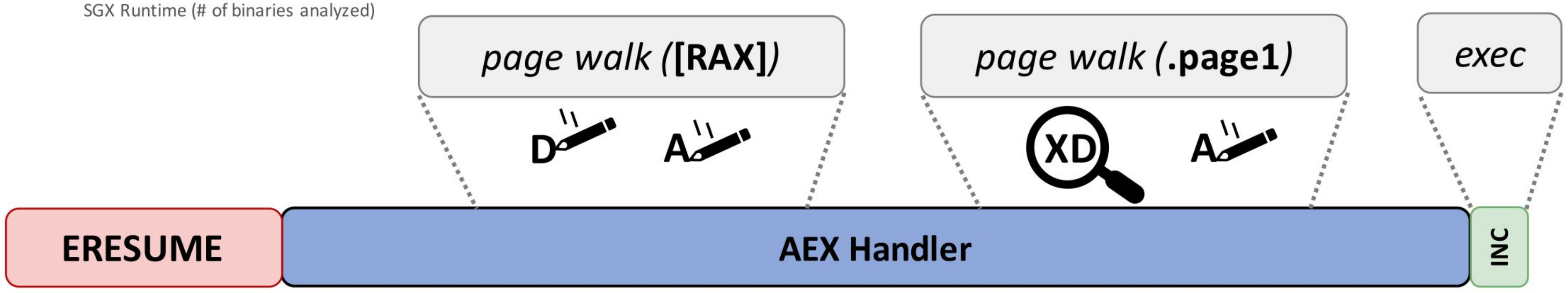


AEX-Notify: Software Implementation

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







Conclusions and Take-Away

Value of **deductive formal models**

… guided and refined by **inductive validation!**

Synergy attacks ↔ defenses; small research prototypes ↔ (high-end) real-world CPUs

Thank you! Questions?