Interrupt Latency Timing Attacks Against Enclave Programs

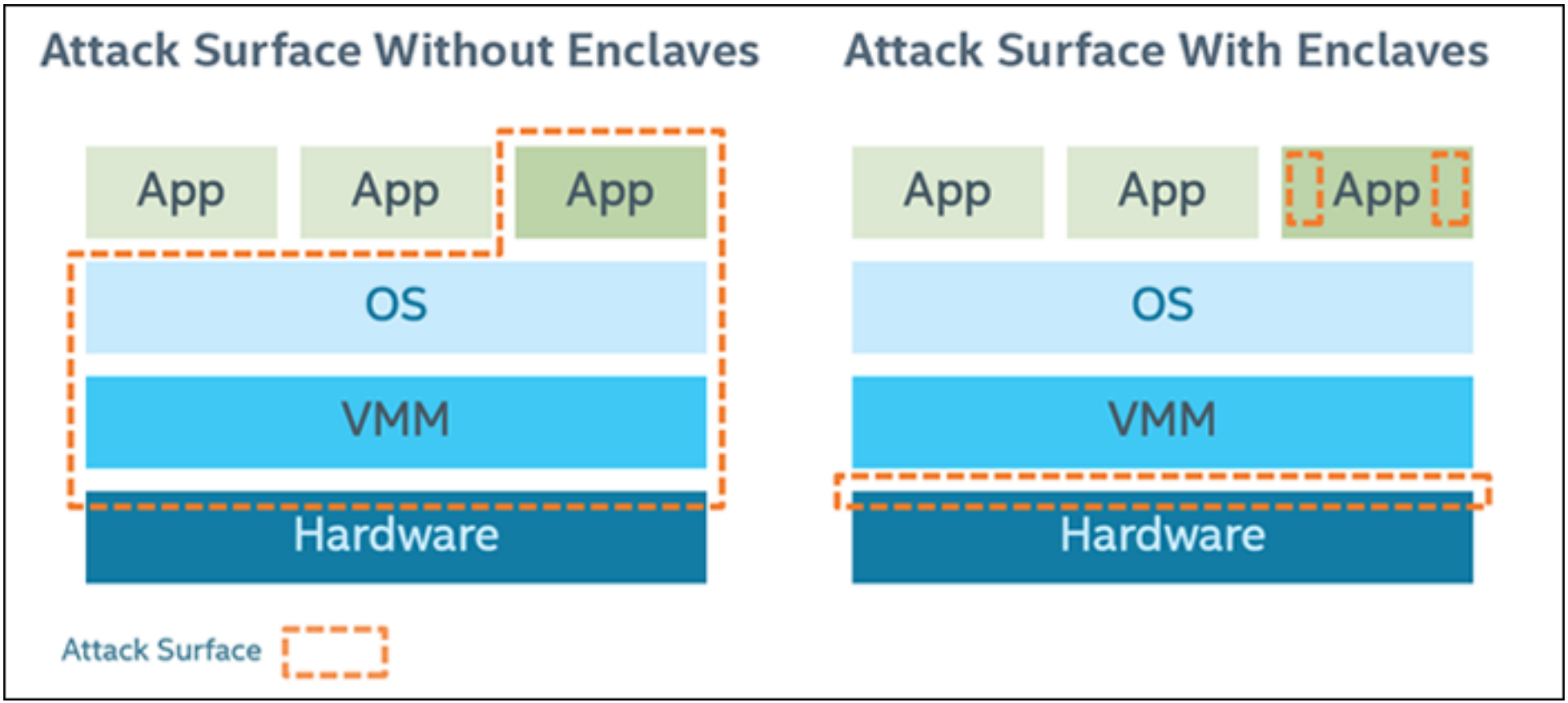
Jo Van Bulck Raoul Strackx Frank Piessens

imec-DistriNet, KU Leuven, Celestijnenlaan 200A, B-3001 Belgium

DRADS, April 28, 2017

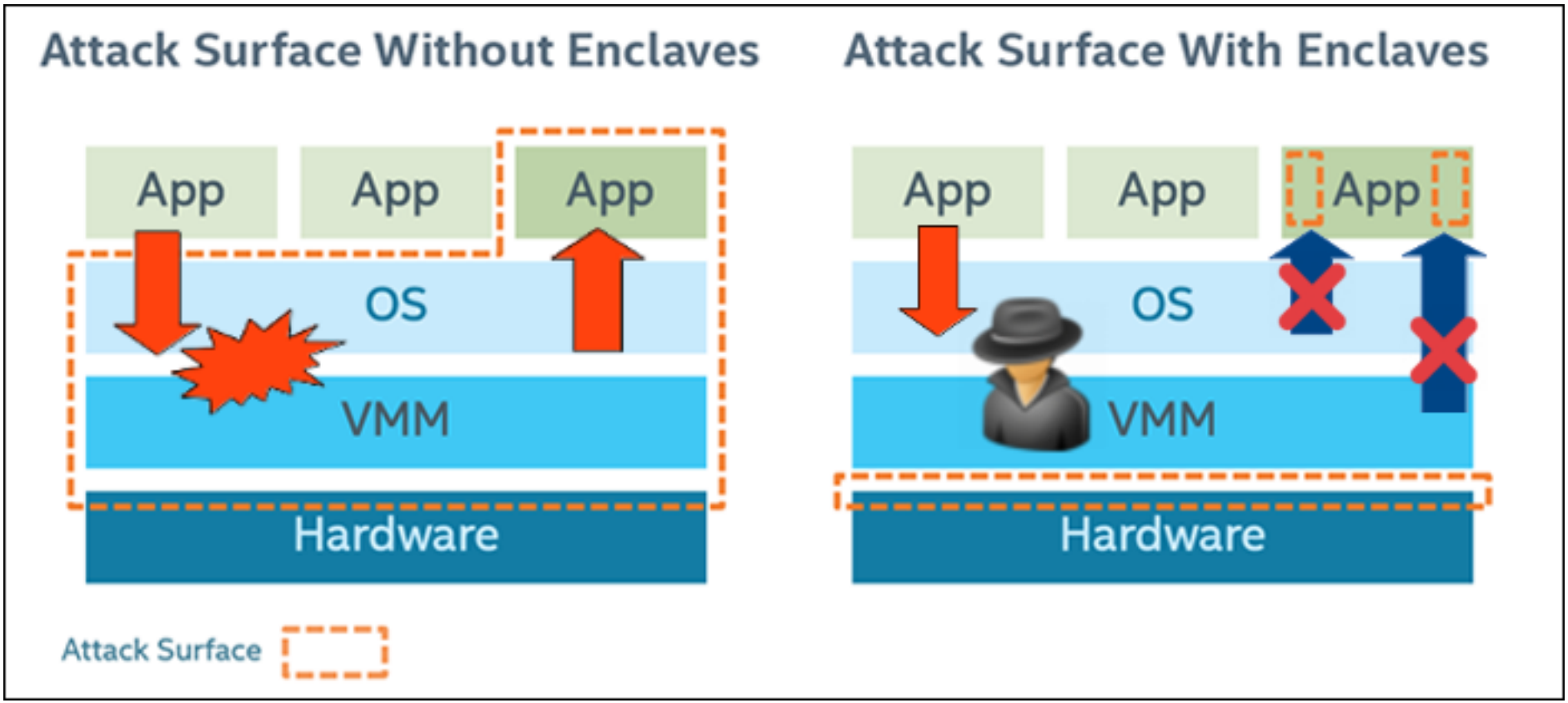
Road Map

Motivation: Application Attack Surface



https://software.intel.com/en-us/articles/intel-software-guard-extensions-tutorial-part-1-foundation

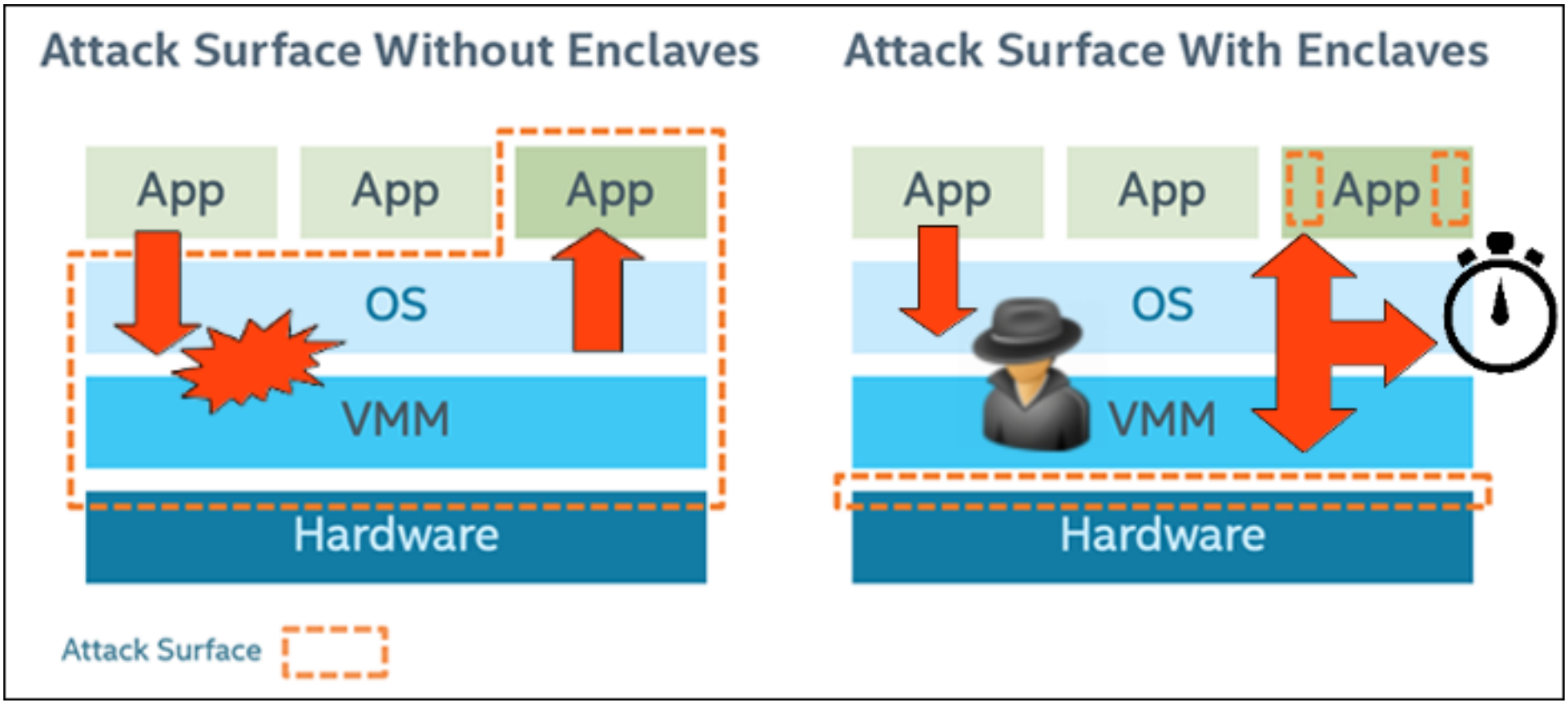
Motivation: Application Attack Surface



https://software.intel.com/en-us/articles/intel-software-guard-extensions-tutorial-part-1-foundation

Layered architecture $\leftrightarrow$ **hardware-only TCB**

Motivation: Application Attack Surface



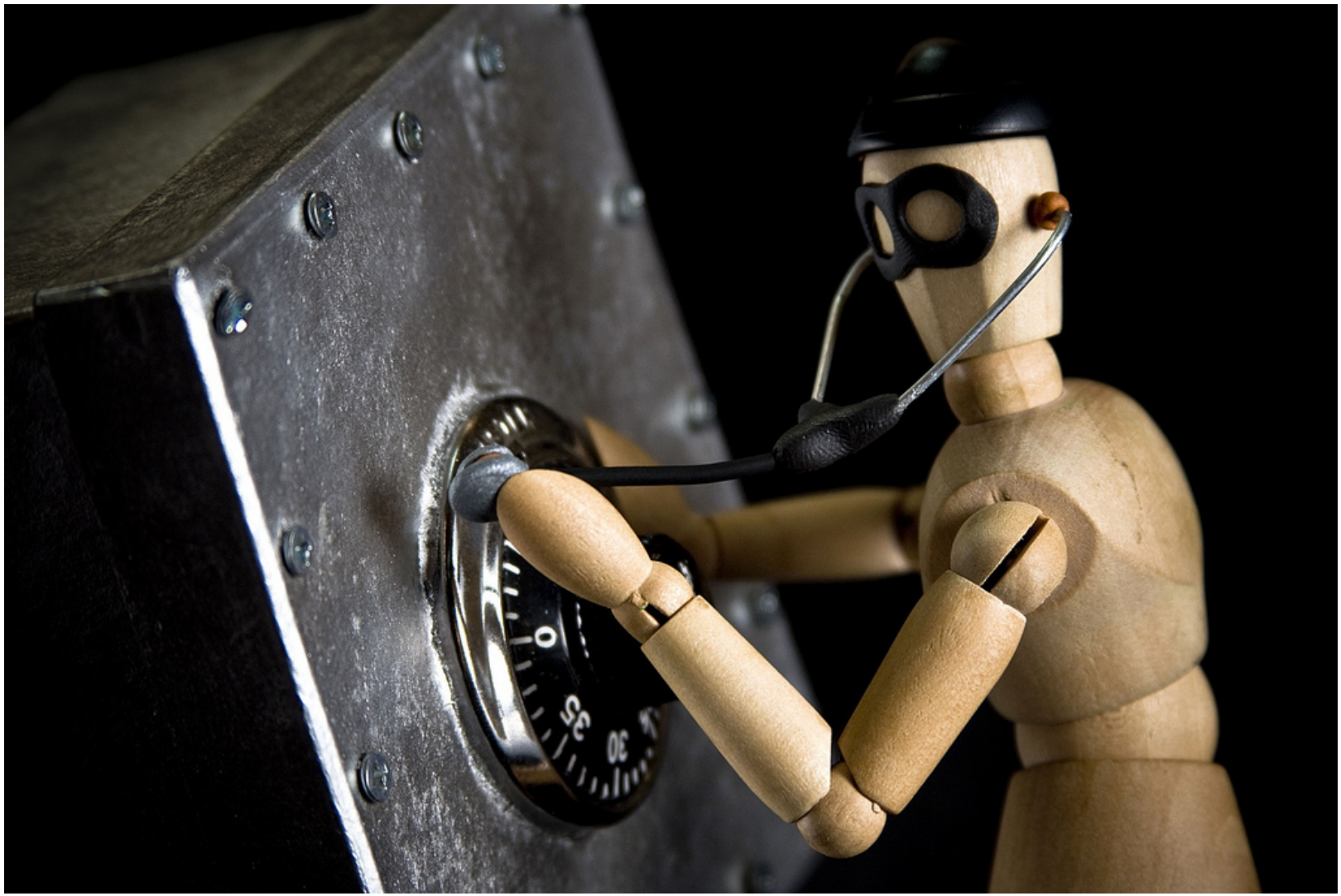
https://software.intel.com/en-us/articles/intel-software-guard-extensions-tutorial-part-1-foundation

Untrusted OS $\rightarrow$ new class of powerful **side-channels**

Side-Channel Attack Principle

Source: `https://commons.wikimedia.org/wiki/File:WinonaSavingsBankVault.JPG`

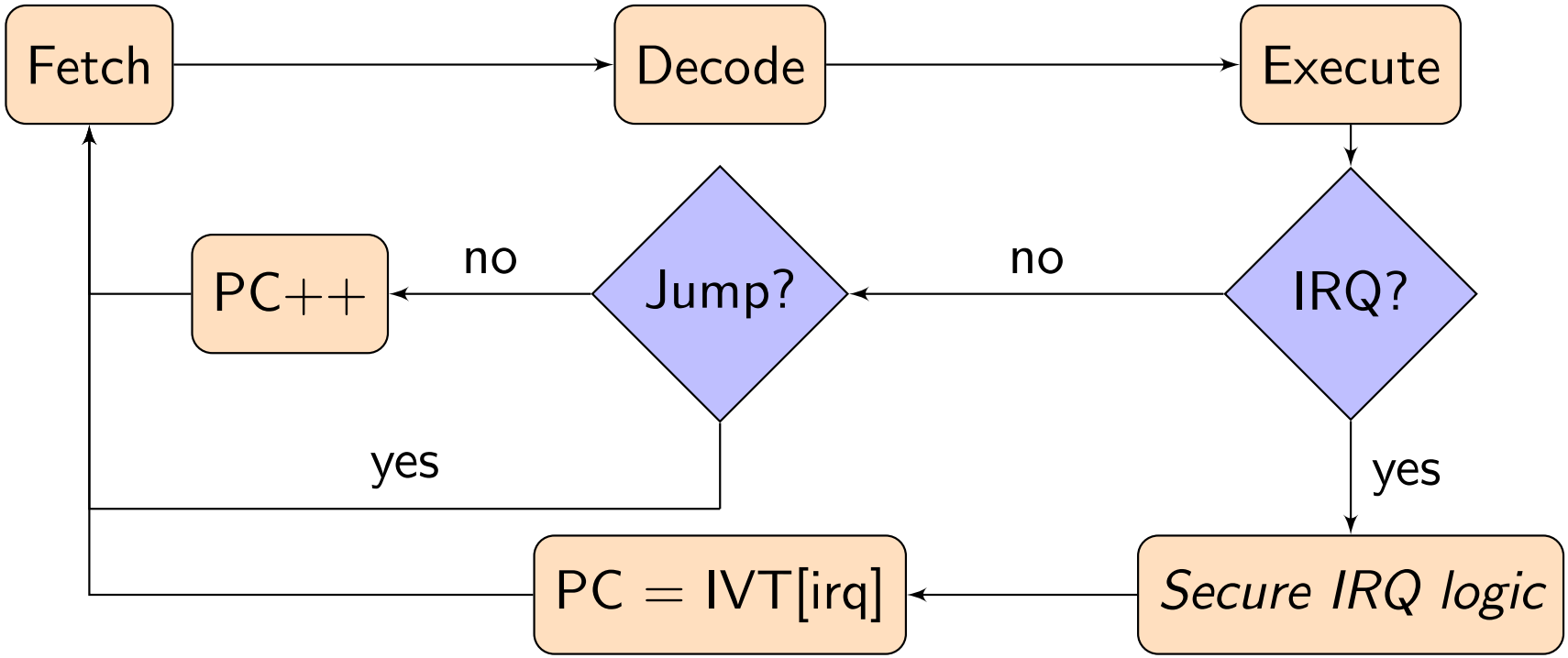
Side-Channel Attack Principle



Source: `https://flic.kr/p/69sHDa`

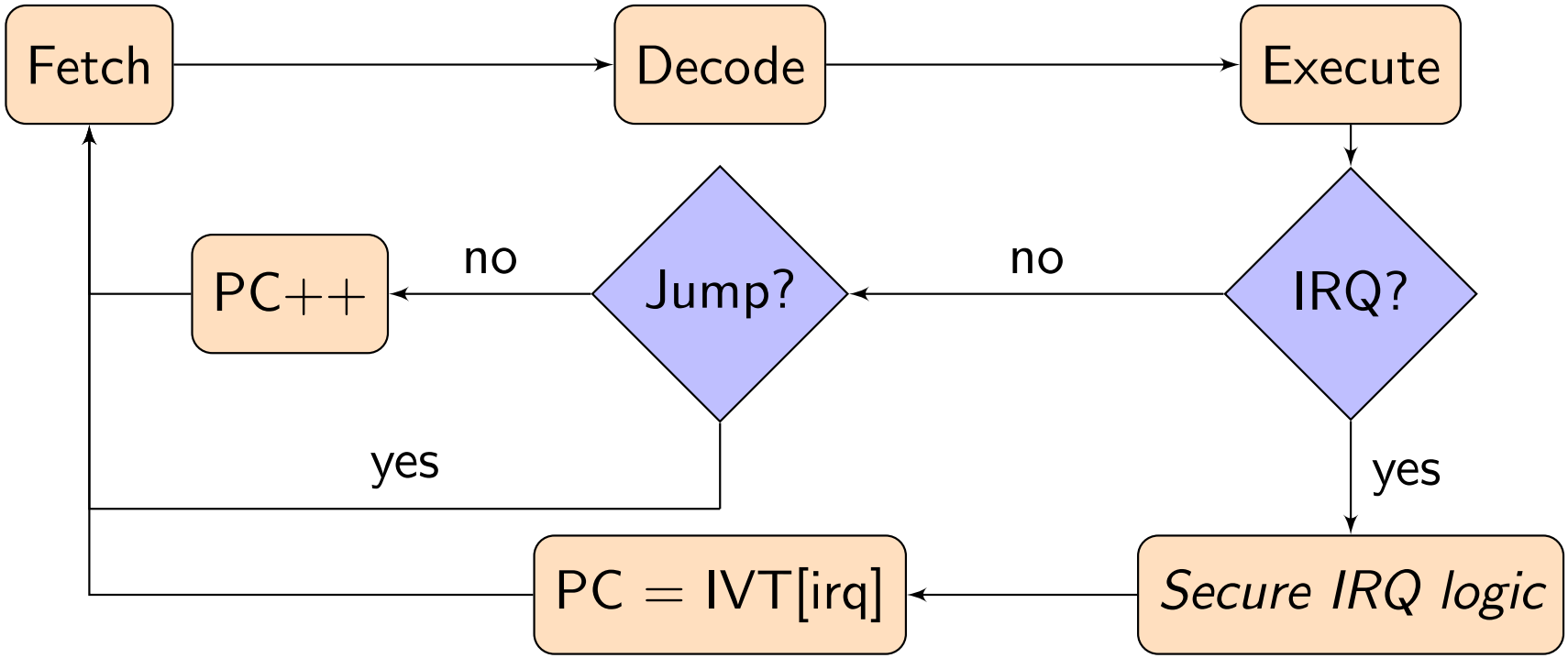
Road Map

Fetch-Decode-Execute CPU Operation



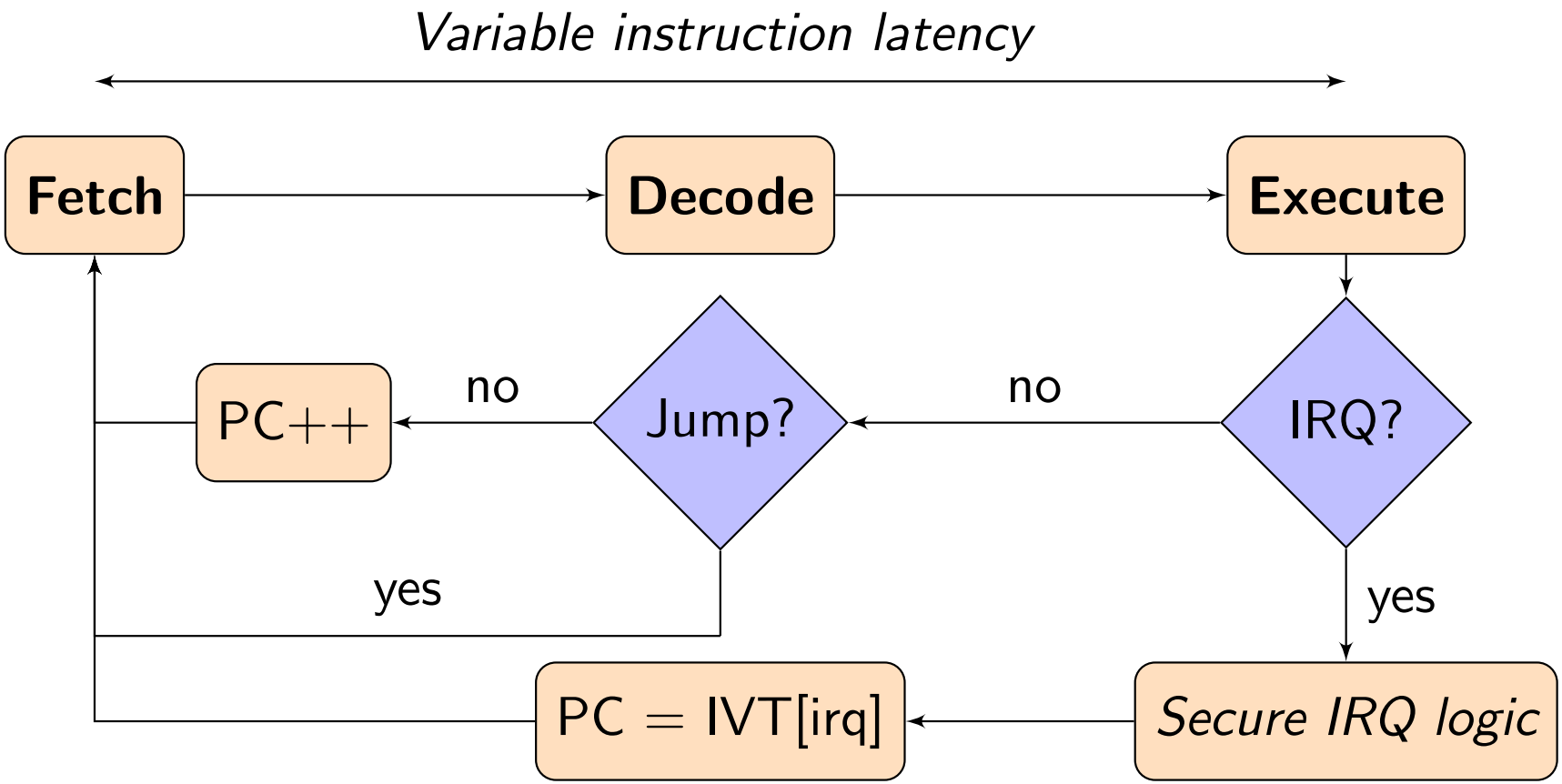
Fetch-Decode-Execute CPU Operation

Note: IRQ only served *after current instruction* has completed

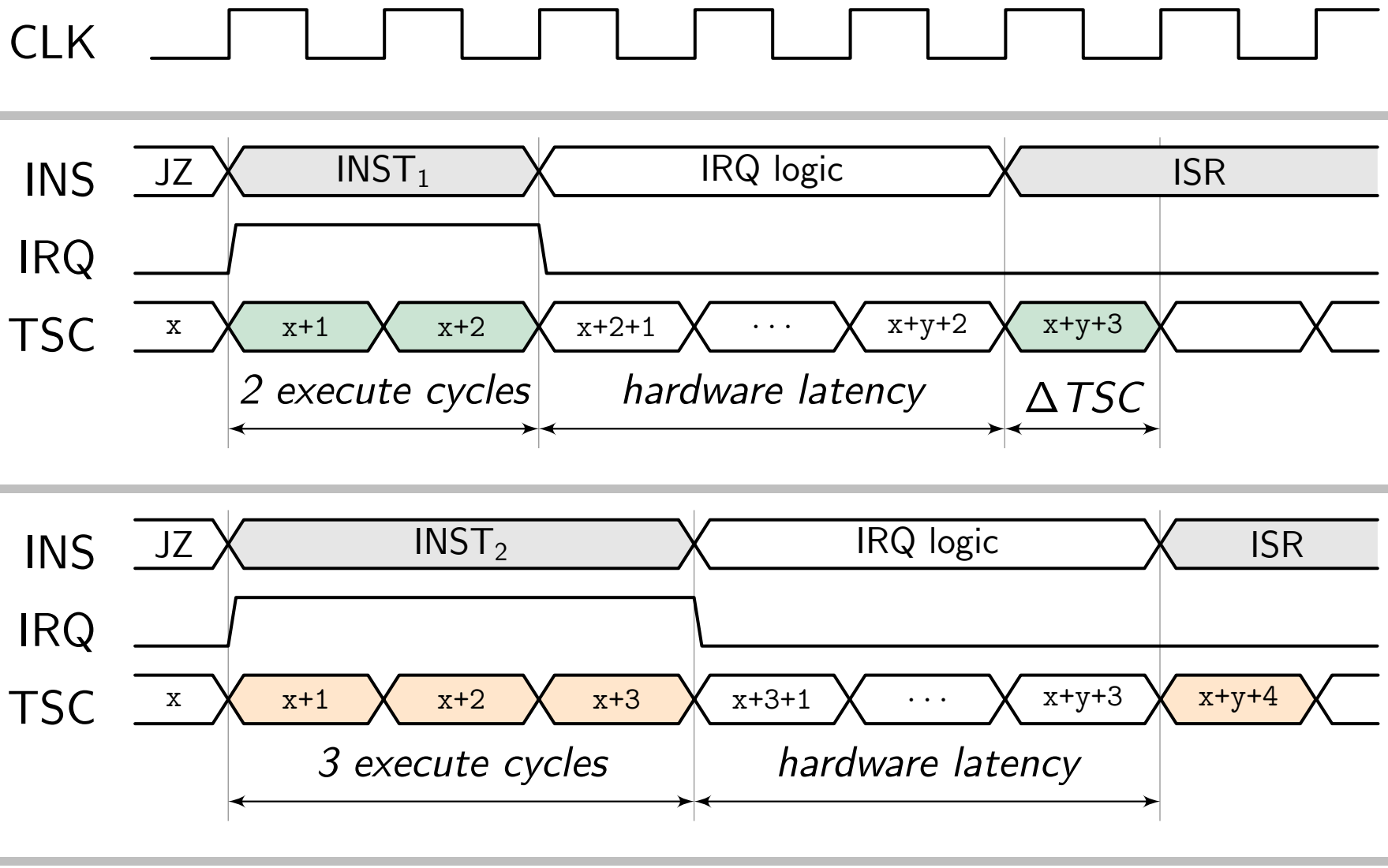


Wait a Cycle . . .

⇒ **IRQ latency leaks instruction execution time (!)**

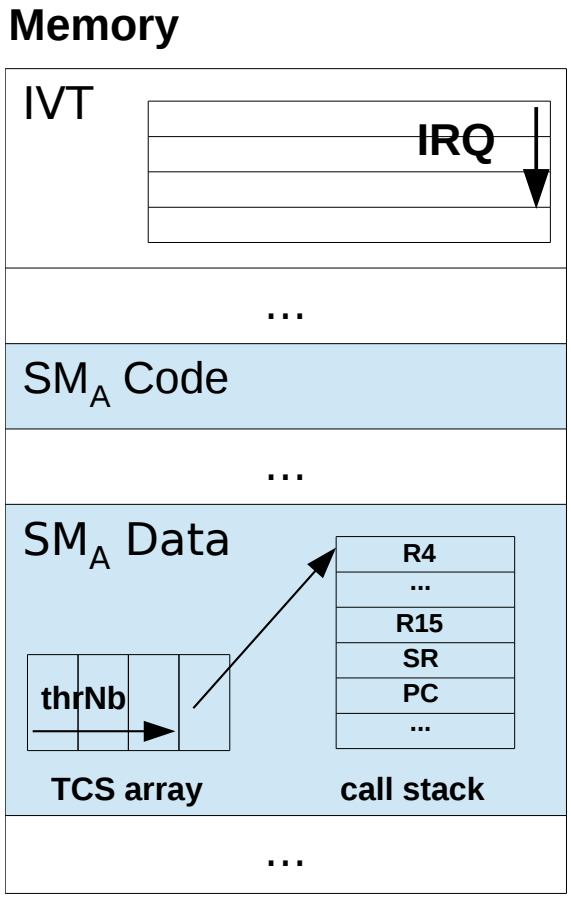


Interrupt Latency as a Side-Channel



Road Map

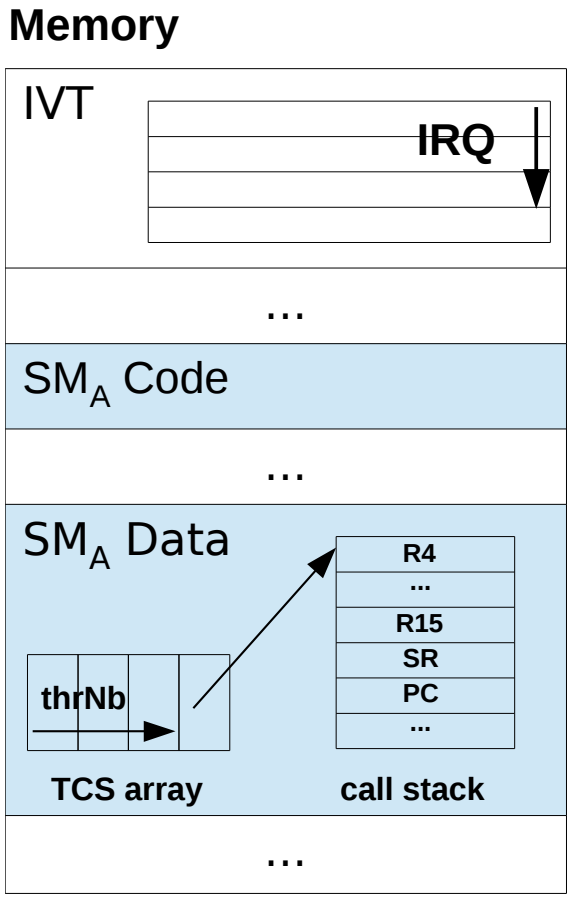
Sancus Protected Module Architecture



Low-cost embedded processor:

- **IoT** device: no pipeline/cache/MMU
- Extended **openMSP430** instruction set
- SM **isolation/authentication** primitives

Sancus Protected Module Architecture



Low-cost embedded processor:

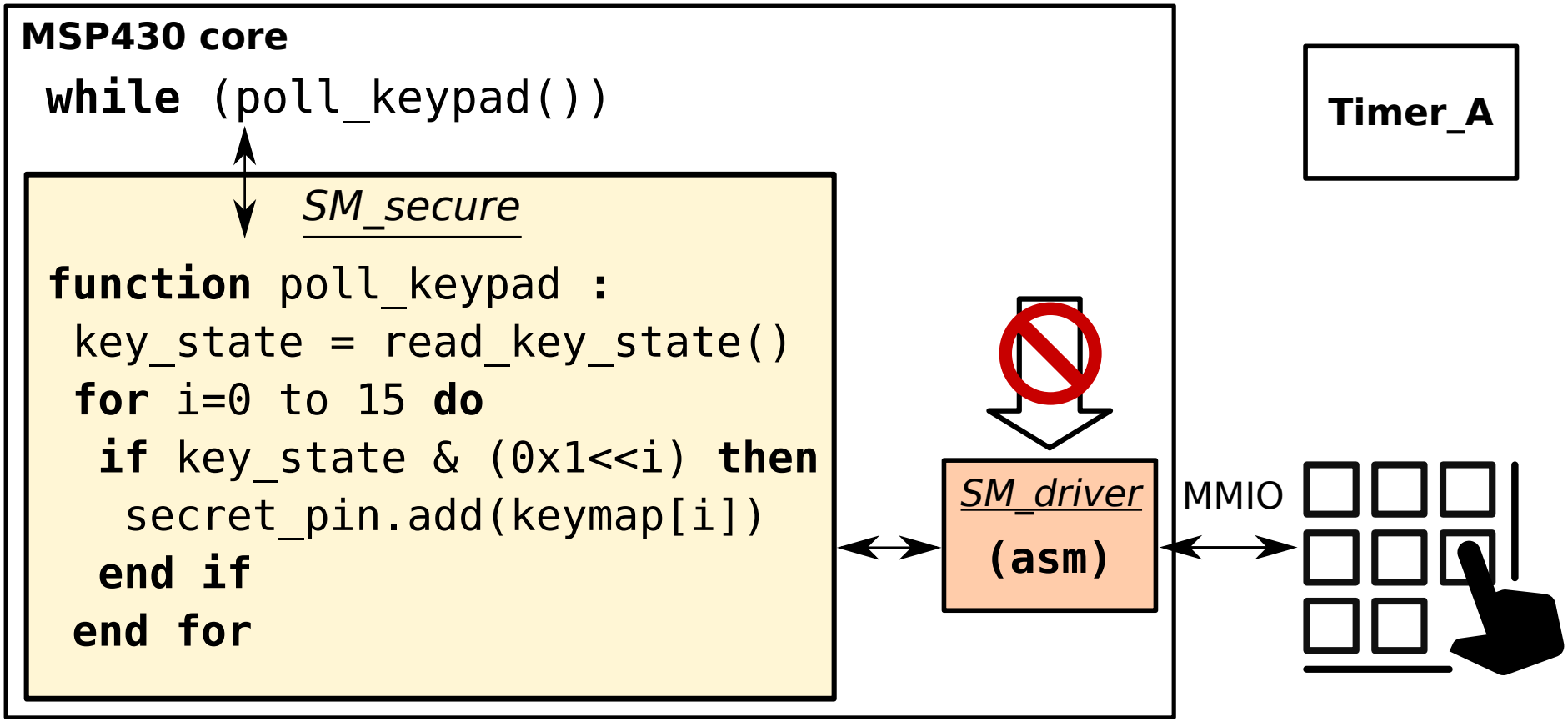
- **IoT** device: no pipeline/cache/MMU
- Extended **openMSP430** instruction set
- SM **isolation/authentication** primitives

Secure interrupts:

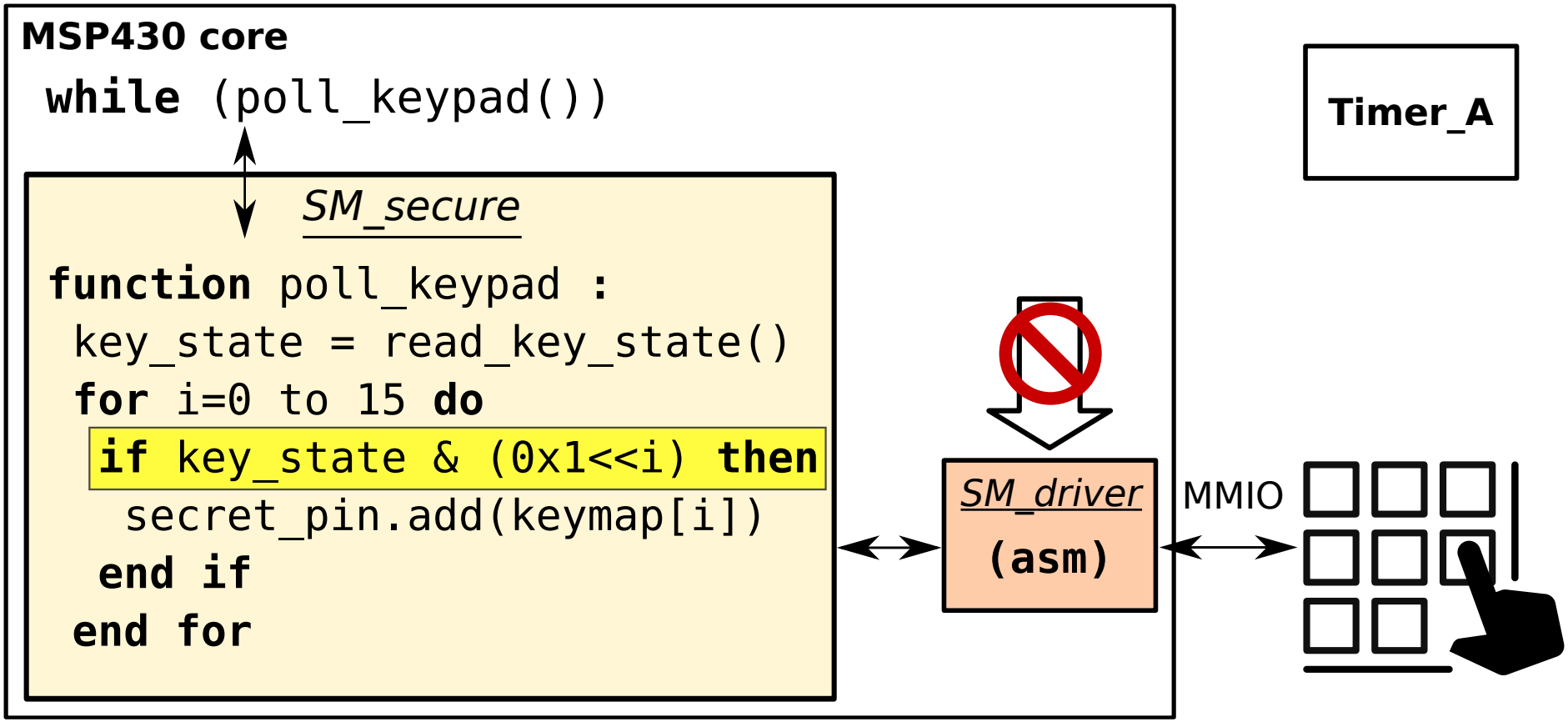
- HW-level **interrupt engine** saves and clears CPU registers
- **Multithreading** SW extensions

Noorman et al.: "Sancus 2.0: A low-cost security architecture for IoT devices", TOPS 2017 [NVBM+17].
Koeberl et al.: "Trustlite: A security architecture for tiny embedded devices", EuroSys 2014 [KSSV14].
De Clercq et al.: "Secure interrupts on low-end microcontrollers", ASAP 2014 [DCPSV14].
Van Bulck et al.: "Towards availability and real-time guarantees for protected module architectures", MASS 2016 [VBNMP16].

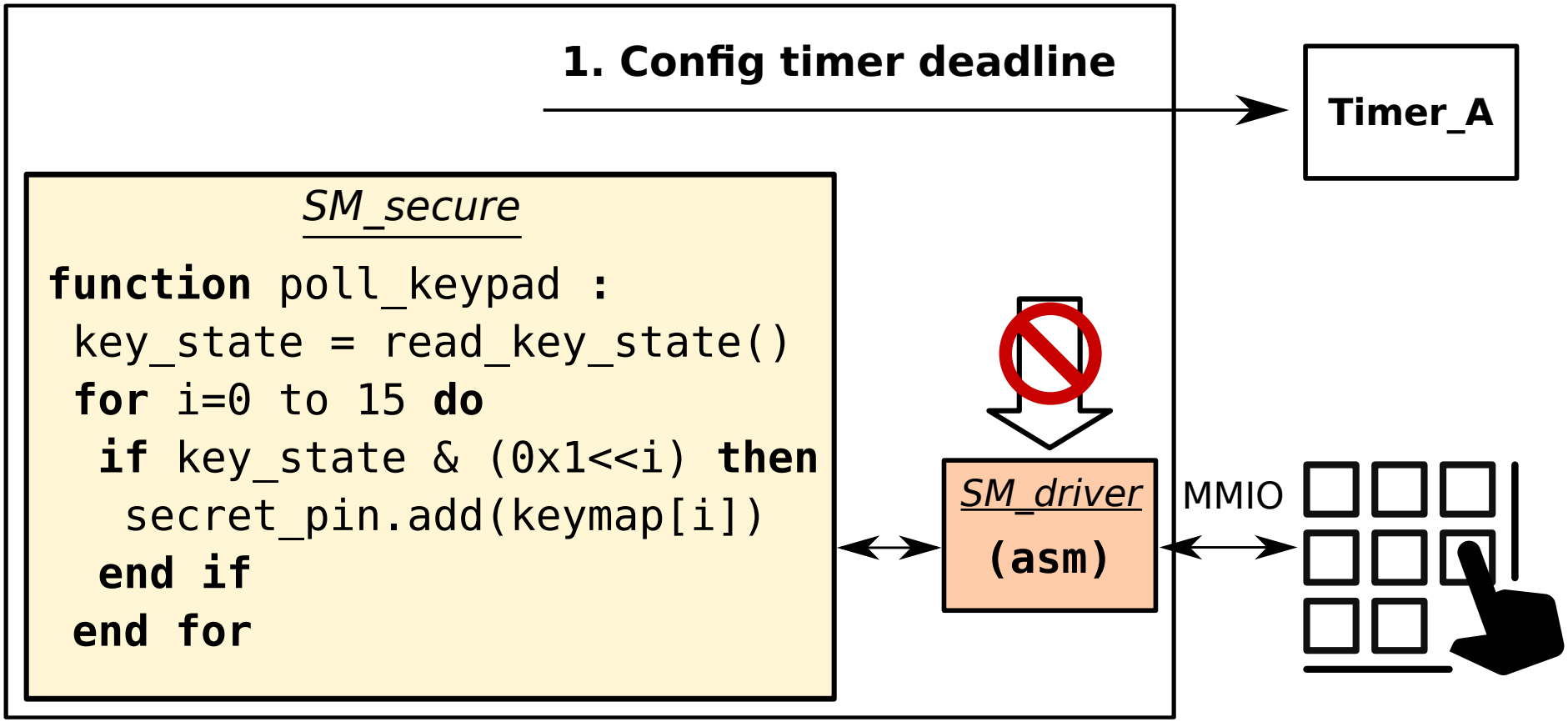
Secure Keypad Application Scenario



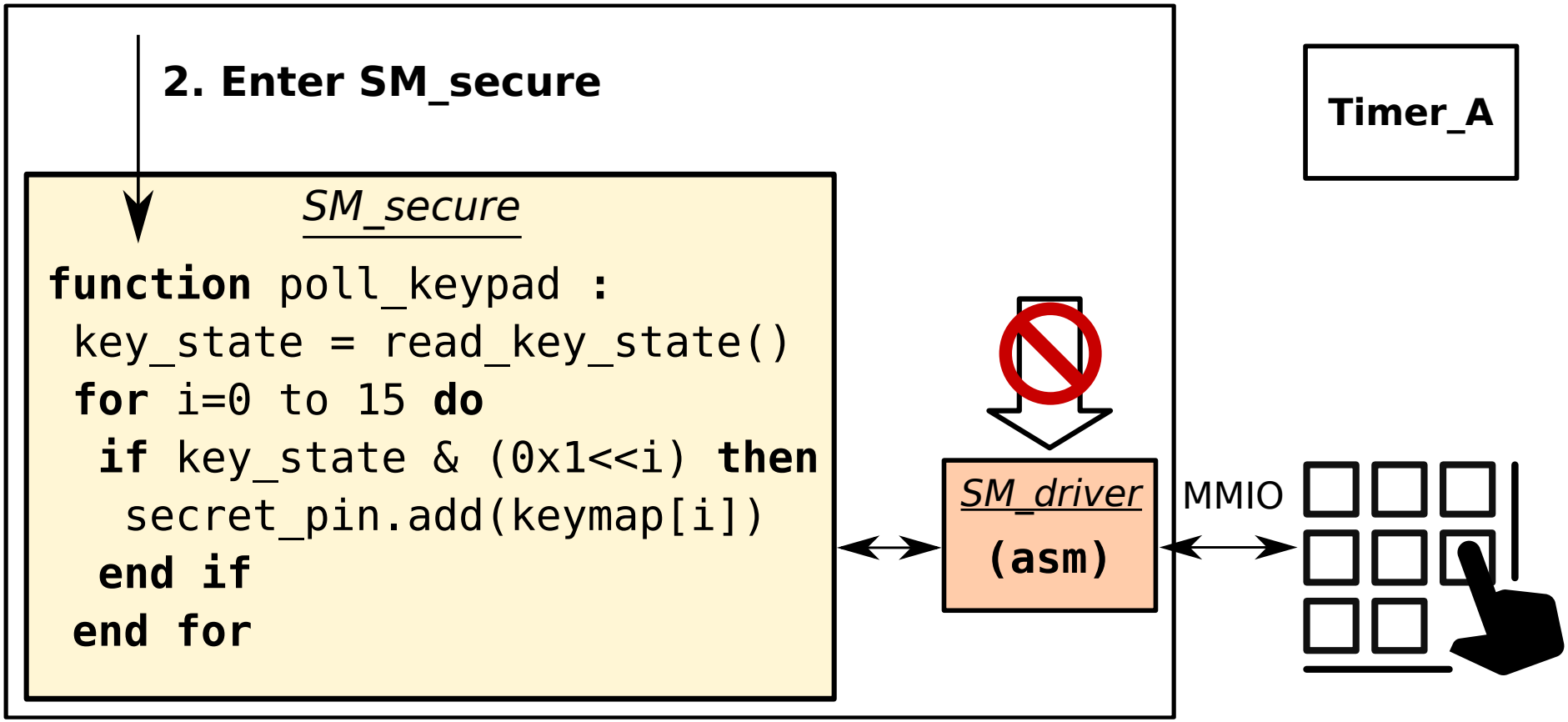
Secure Keypad Application Scenario



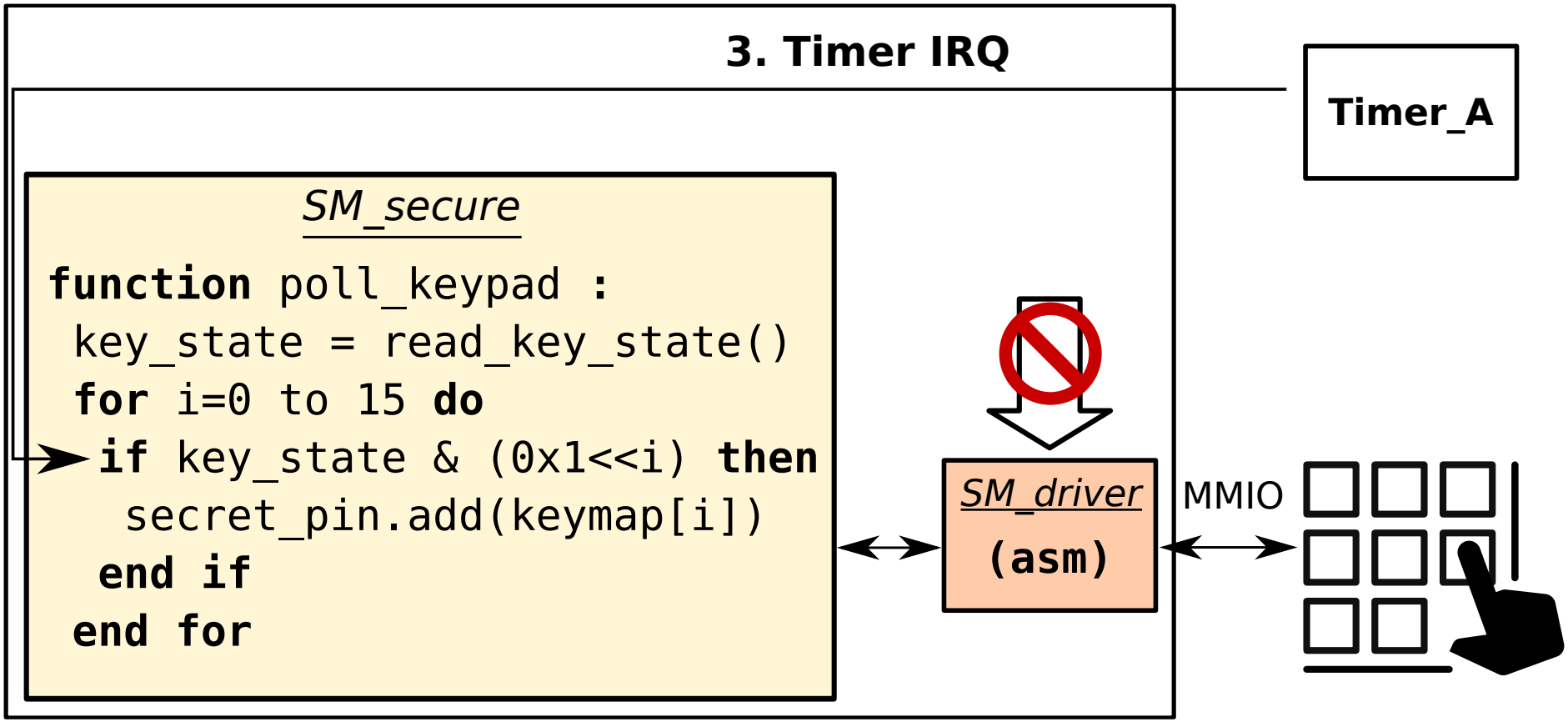
Secure Keypad Application Scenario



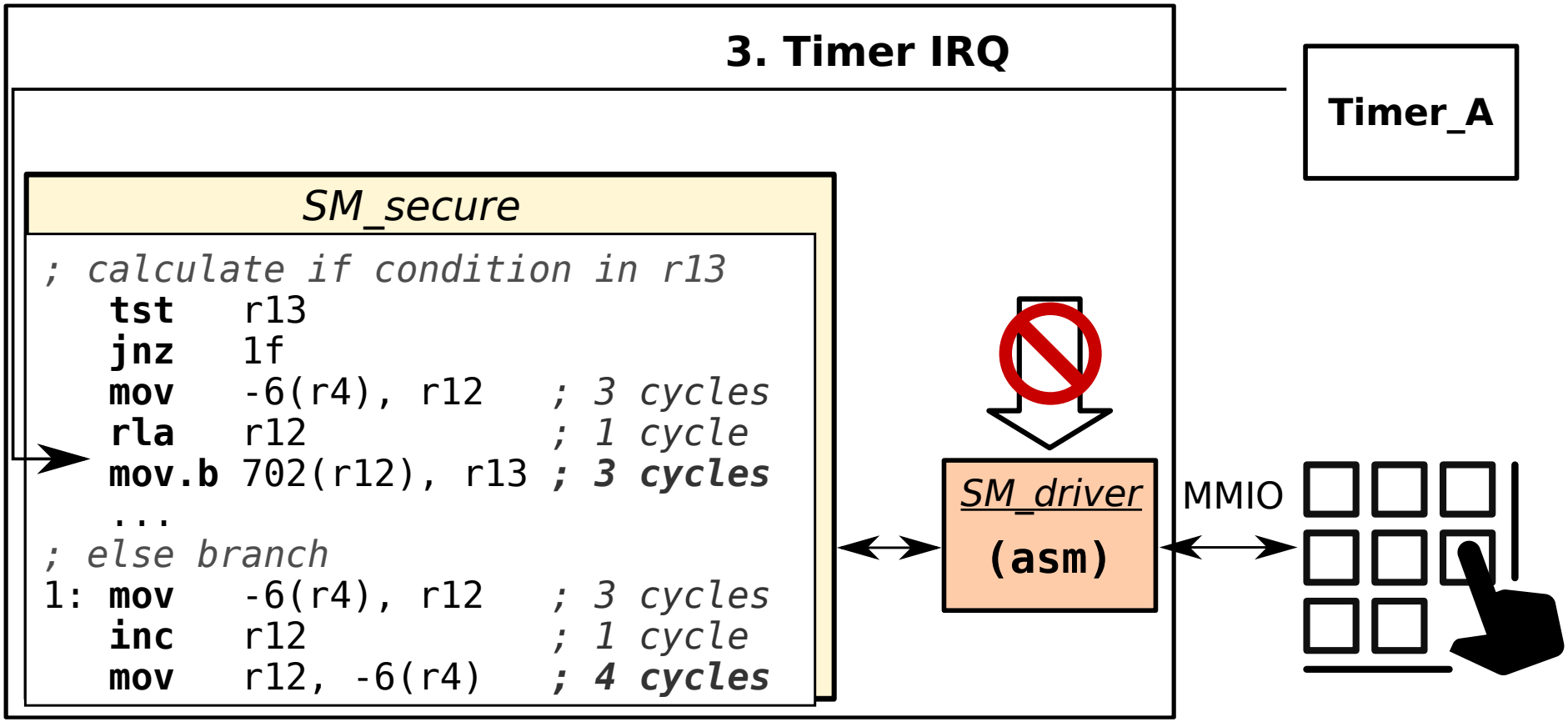
Secure Keypad Application Scenario



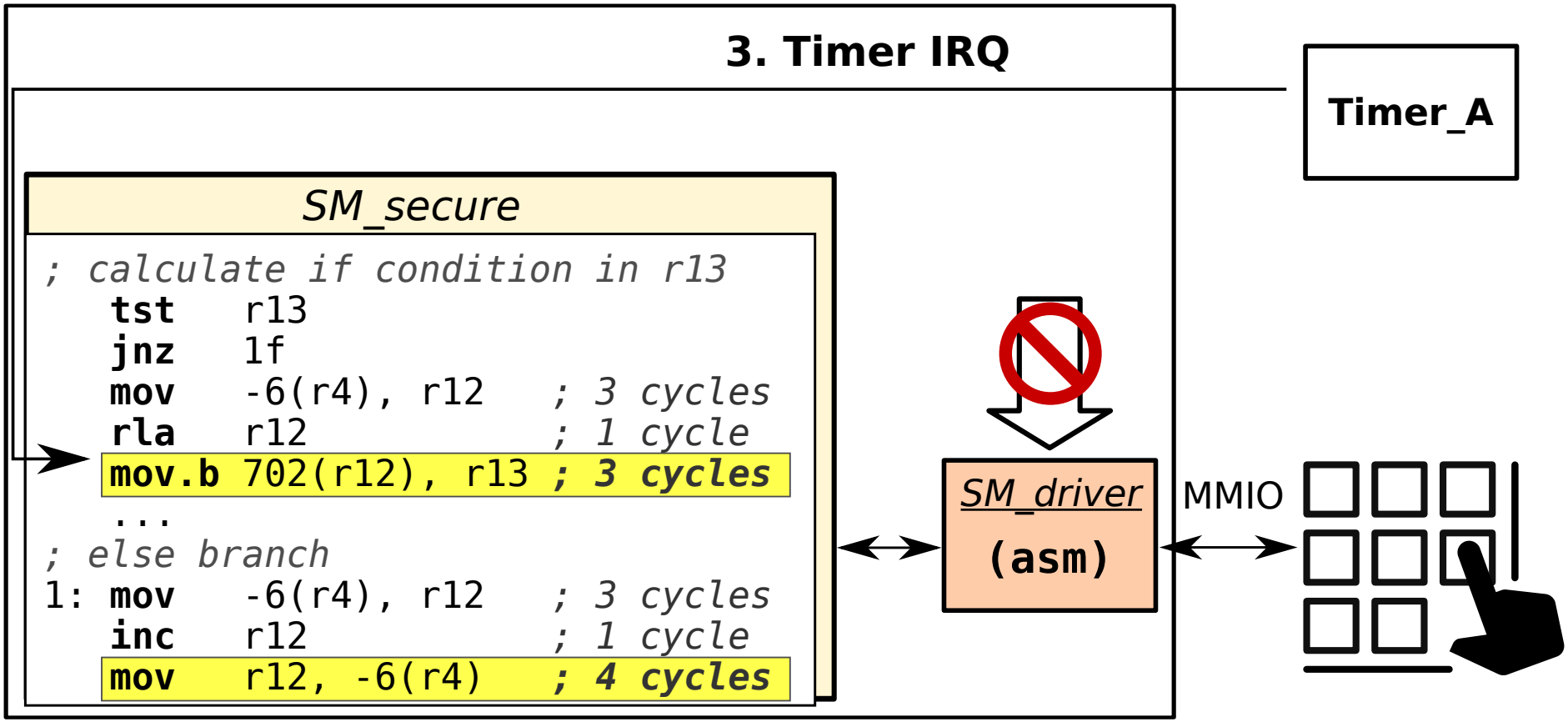
Secure Keypad Application Scenario



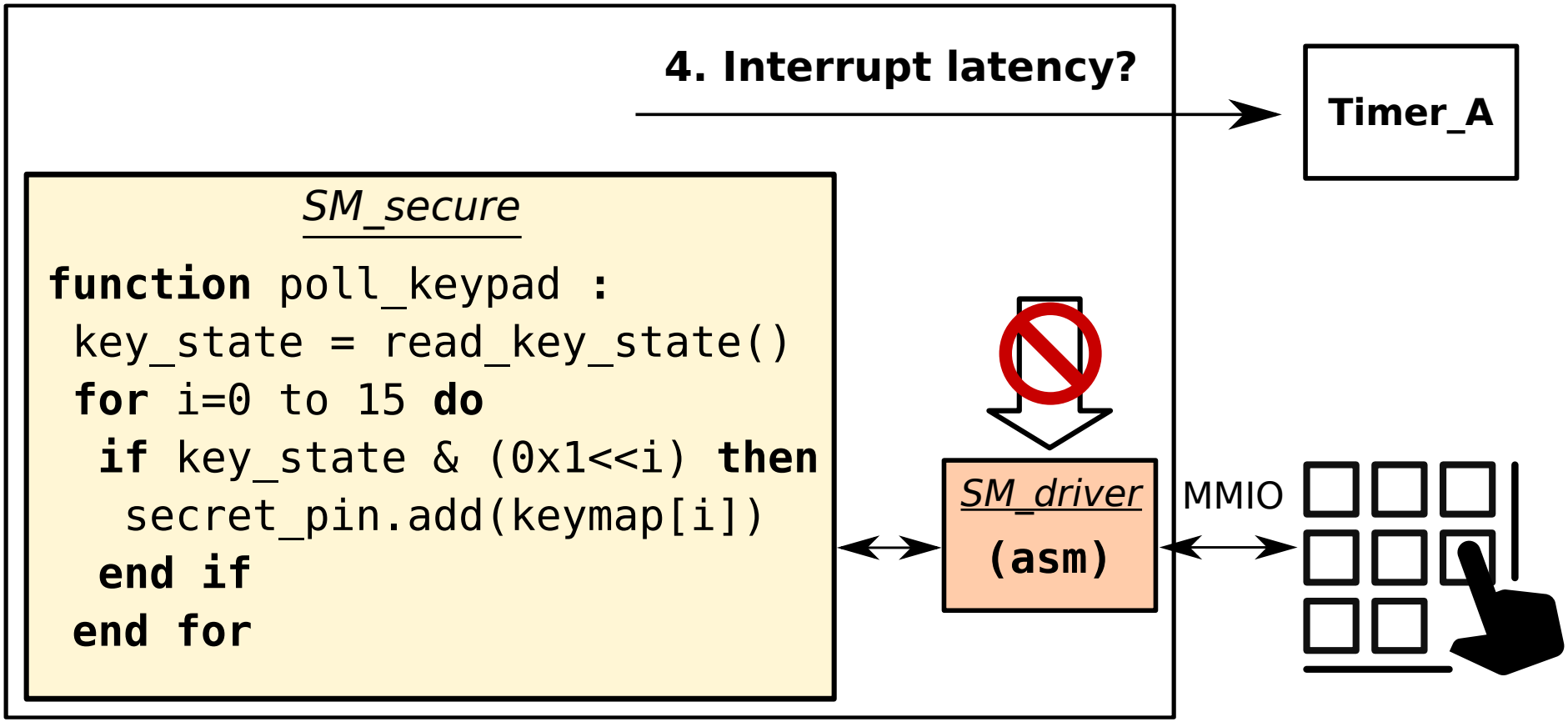
Secure Keypad Application Scenario



Secure Keypad Application Scenario

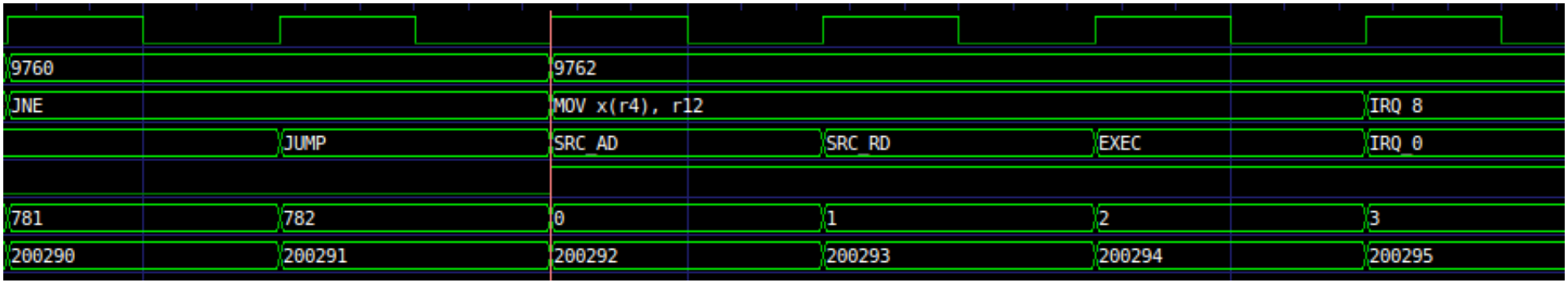


Secure Keypad Application Scenario



Practically Configuring the Timer Interrupt

⇒ *Near-exact copy "spy" SM to leak intermediary timings*

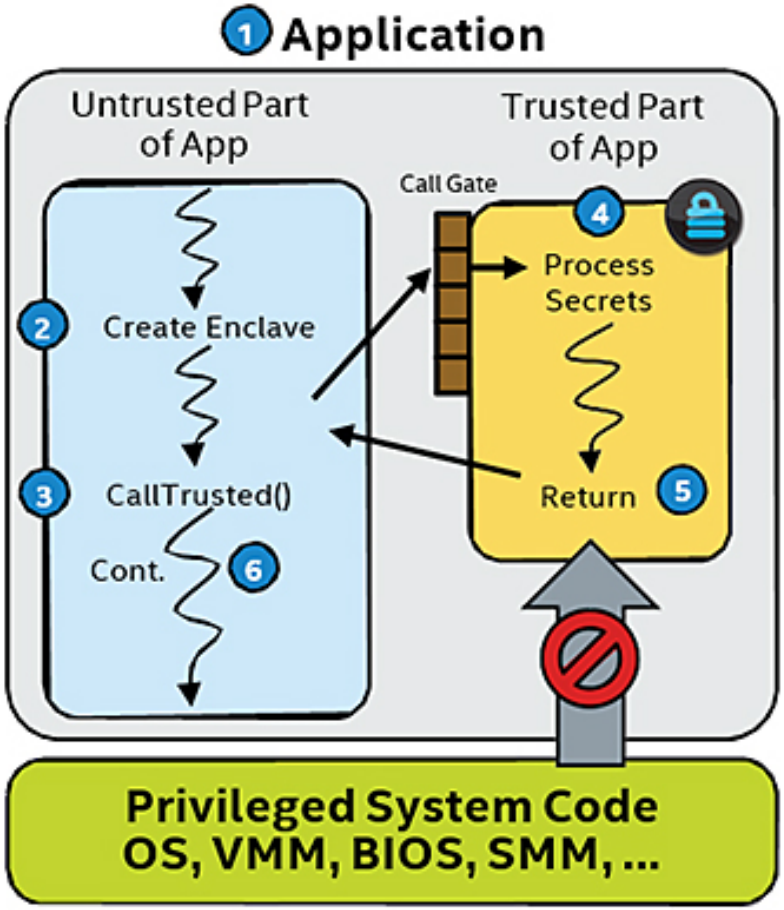



```
[main] spy SM execution time report:
    ------------------------------------
    738 | first key comparison
    233 | reti if to subsequent comparison
    208 | reti else to subsequent comparison

[main] enter secure PIN...
    [isr] key '3' was pressed!
    [isr] key '5' was pressed!
```

Road Map

Intel SGX Helicopter View



https://software.intel.com/en-us/sgx/details

- Protected enclave in application's **virtual address space**
- **x86** CPU: ∃ pipeline, cache, out-of-order execution, ...
- Secure **interrupt** hardware mechanism: AEX/ERESUME

Reducing Noise

BIOS Maximize **execution time predictability:** disable Hyper-Threading, C-States, TurboBoost, SpeedStep

Kernel Reserve **dedicated CPU core** for victim enclave: isolcpus + /proc/smp_affinity + CONFIG_NO_HZ_FULL

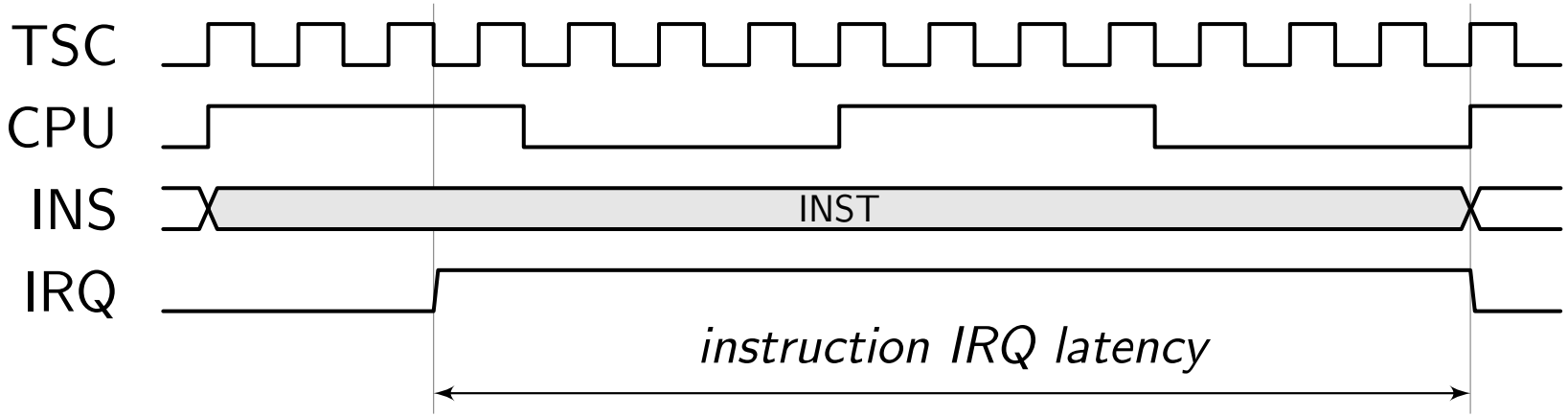
Hardware "sub-cycle accurate" APIC **TSC-deadline** timer

Reducing Noise

BIOS Maximize **execution time predictability:** disable Hyper-Threading, C-States, TurboBoost, SpeedStep

Kernel Reserve **dedicated CPU core** for victim enclave: isolcpus + /proc/smp_affinity + CONFIG_NO_HZ_FULL

Hardware "sub-cycle accurate" APIC **TSC-deadline** timer



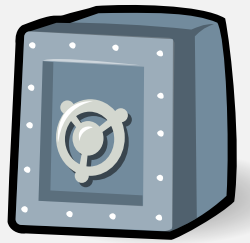
Interrupting and Resuming Enclaves

Goal: single-step through SGX enclave: interrupt each instruction sequentially and record corresponding *IRQ latency trace*

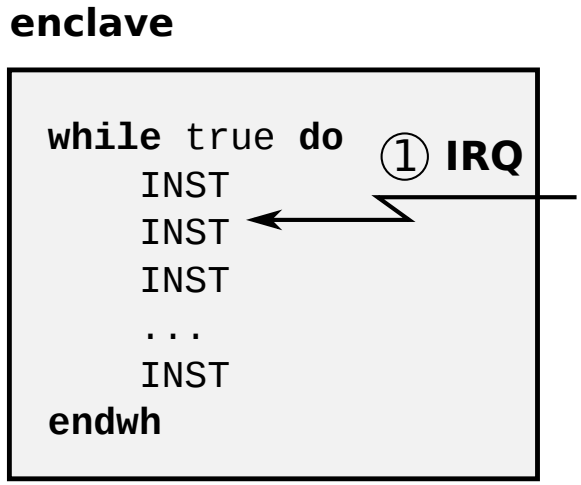
Interrupting and Resuming Enclaves

enclave

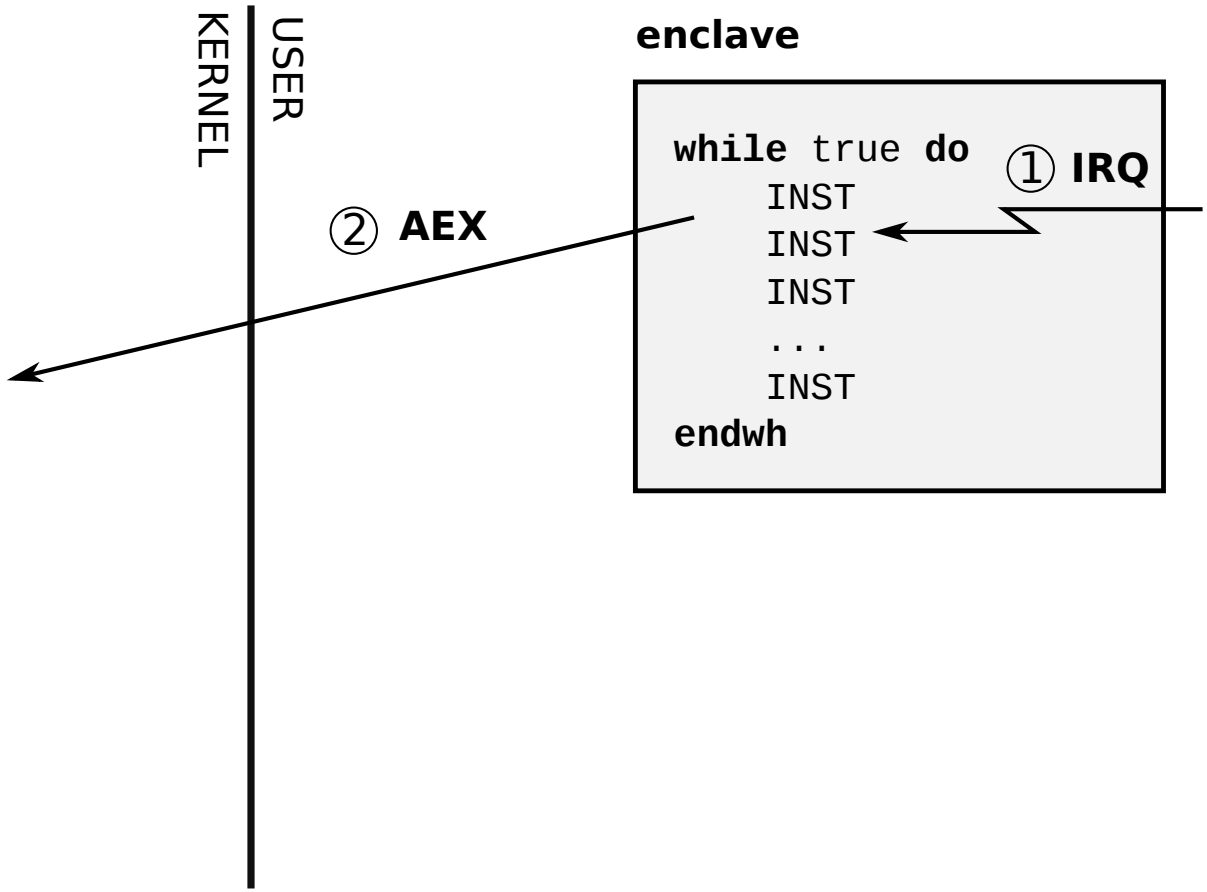
```
while true do
    INST
    INST
    INST
    ...
    INST
endwh
```



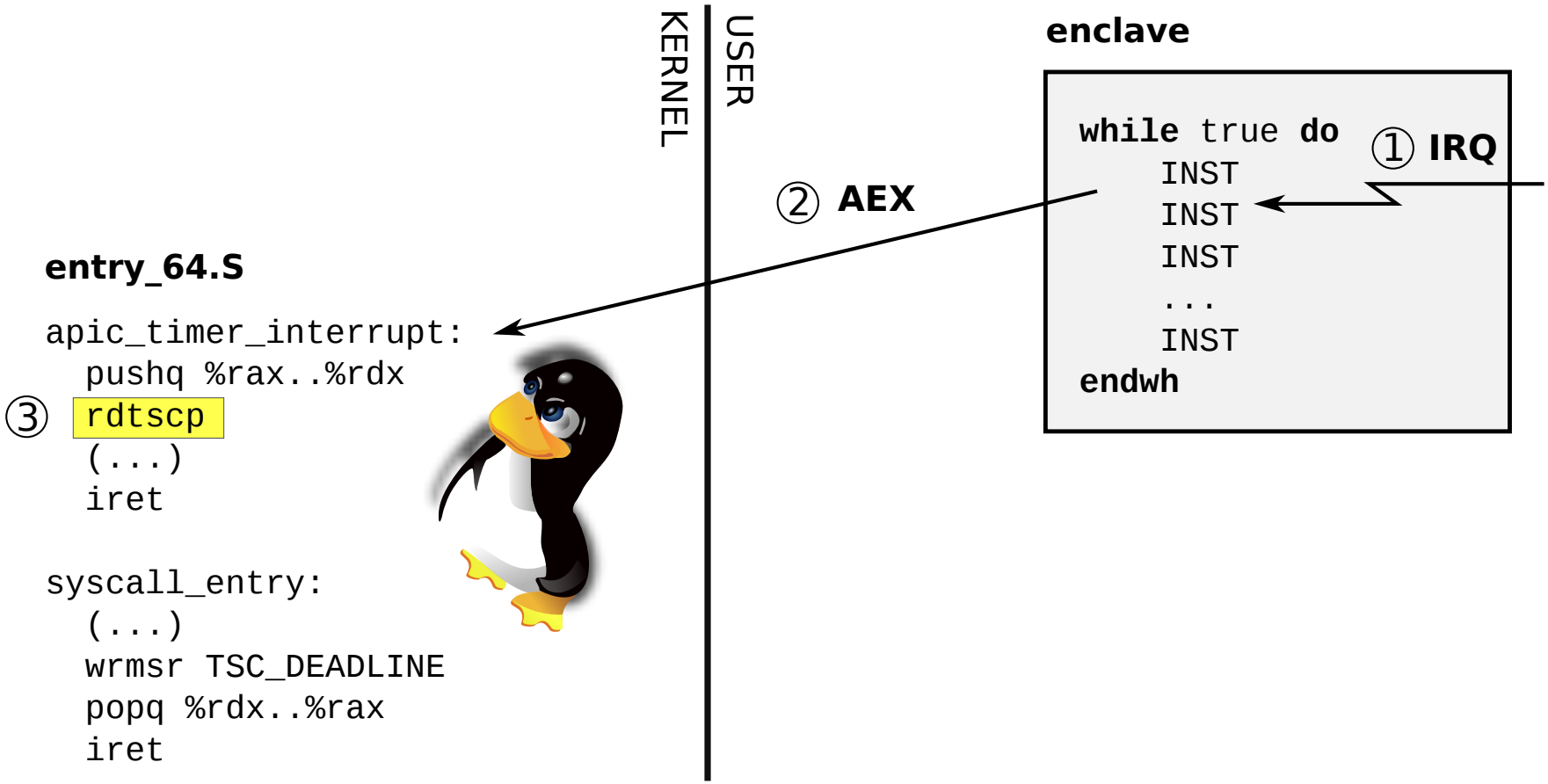
Interrupting and Resuming Enclaves



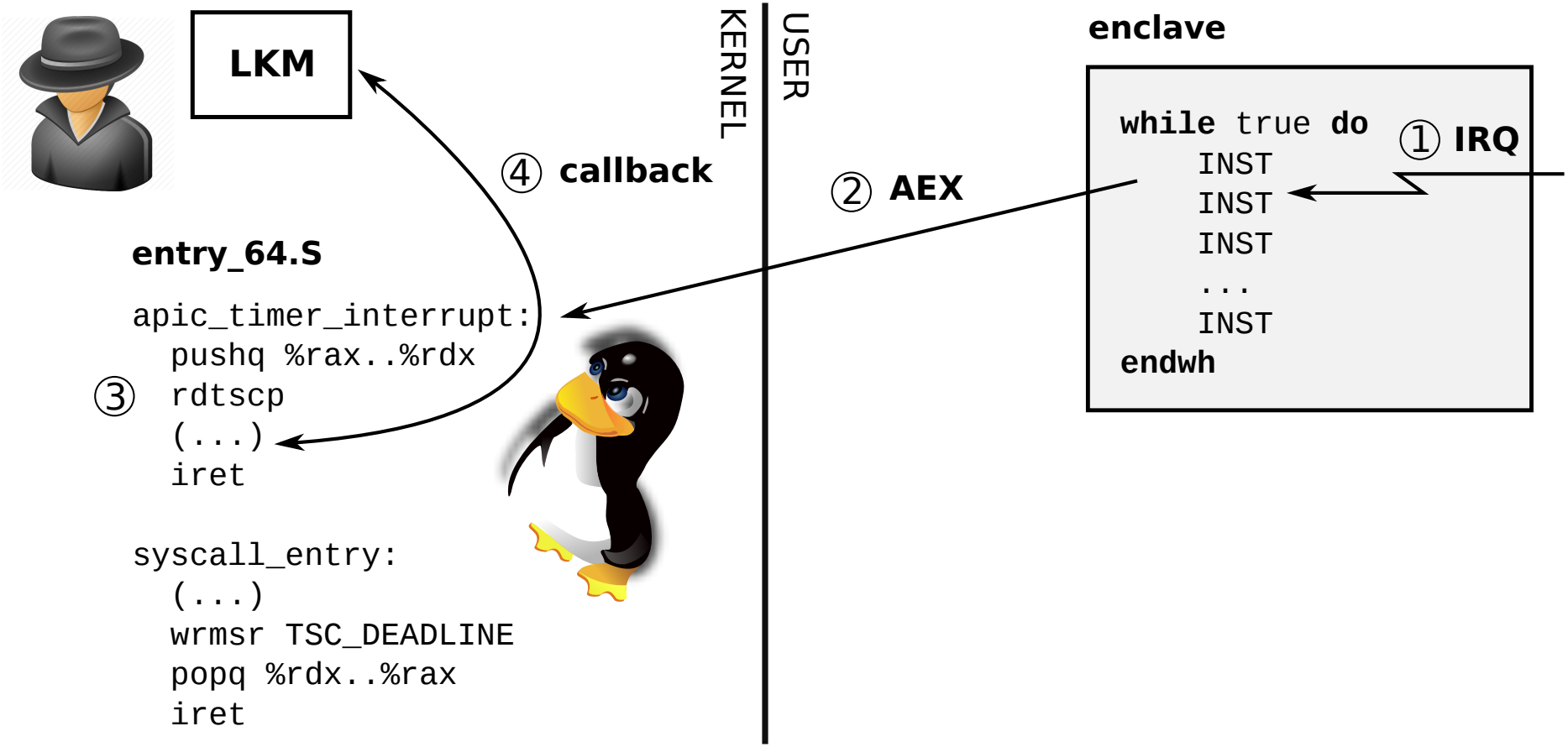
Interrupting and Resuming Enclaves



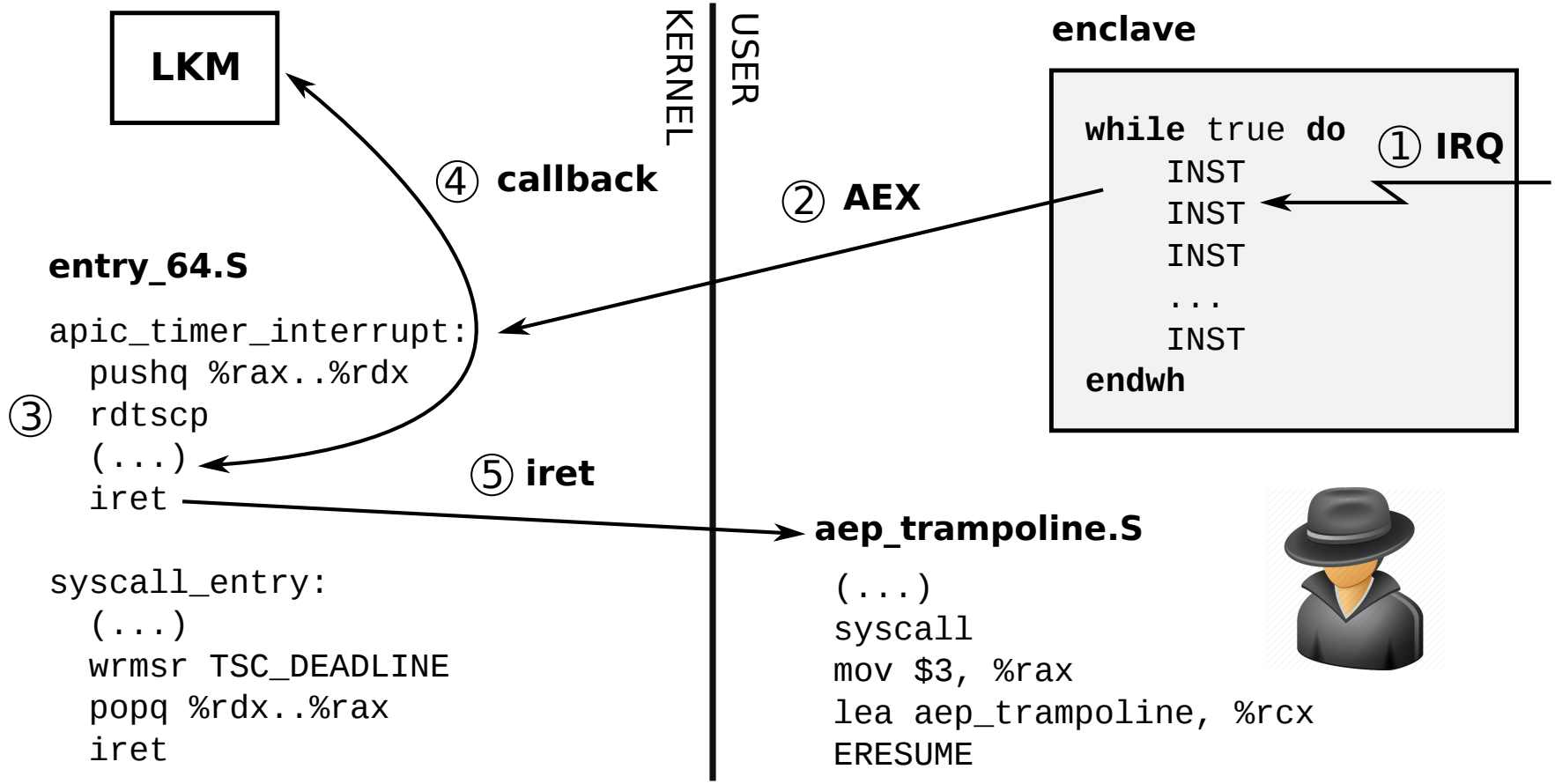
Interrupting and Resuming Enclaves



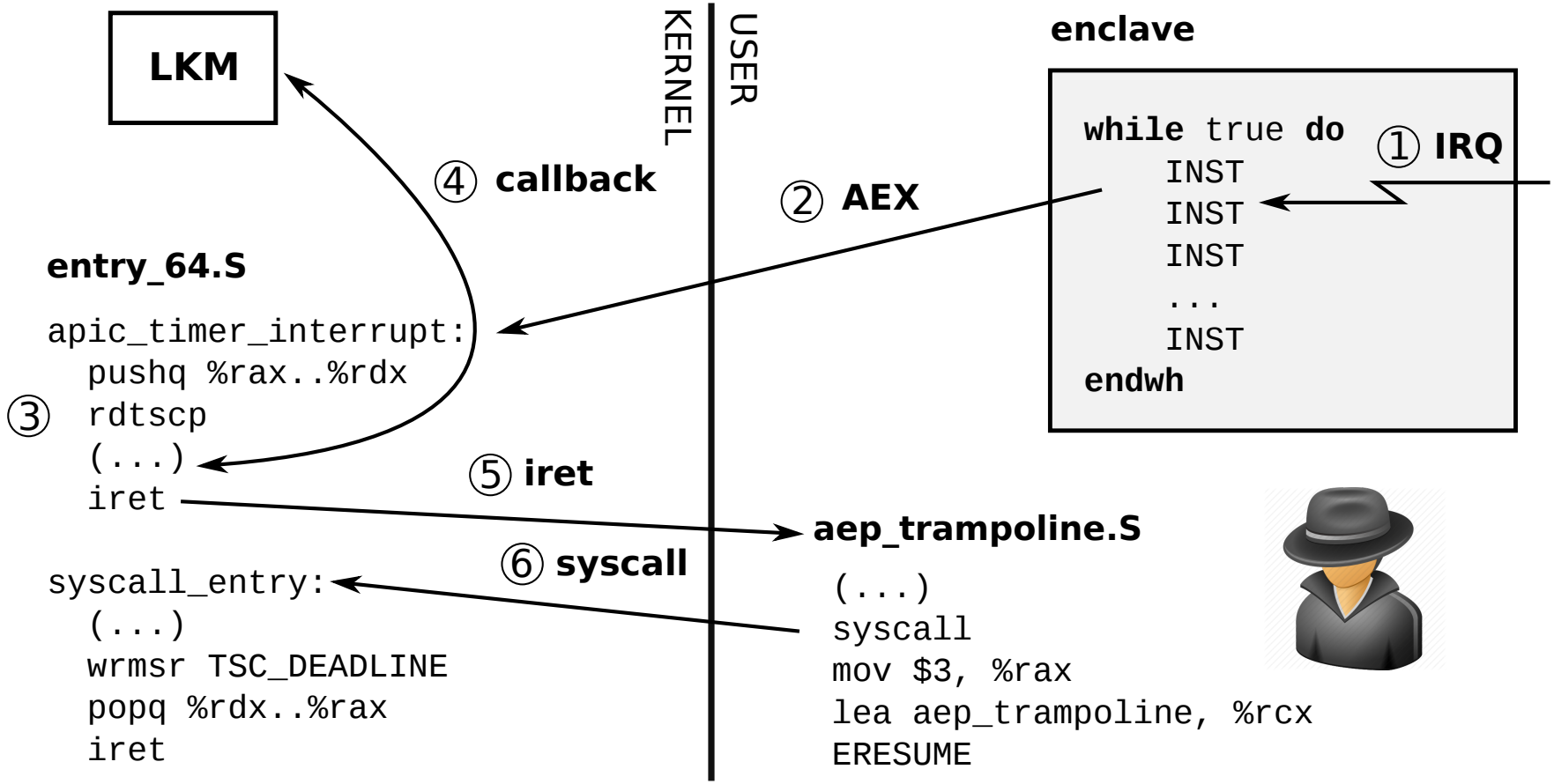
Interrupting and Resuming Enclaves



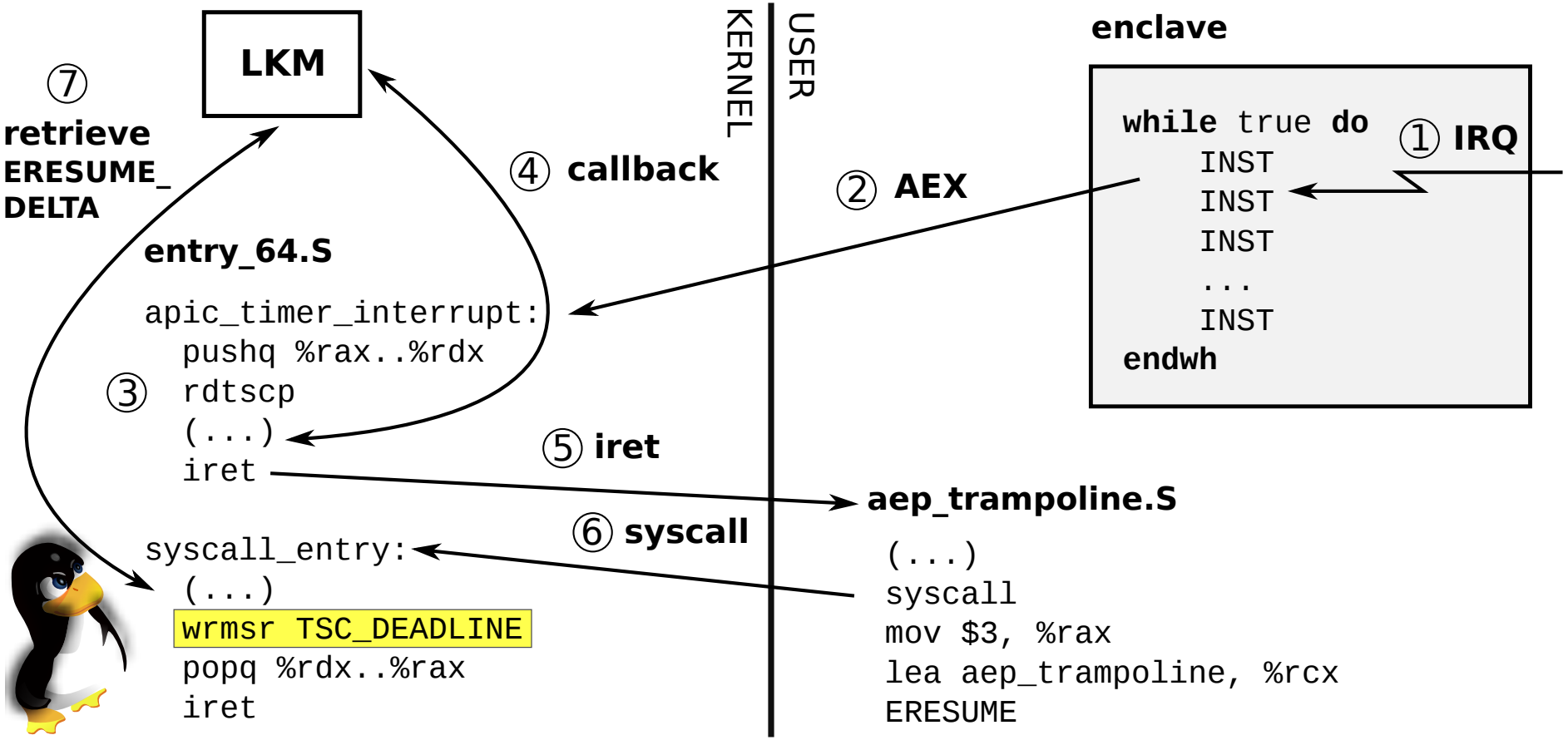
Interrupting and Resuming Enclaves



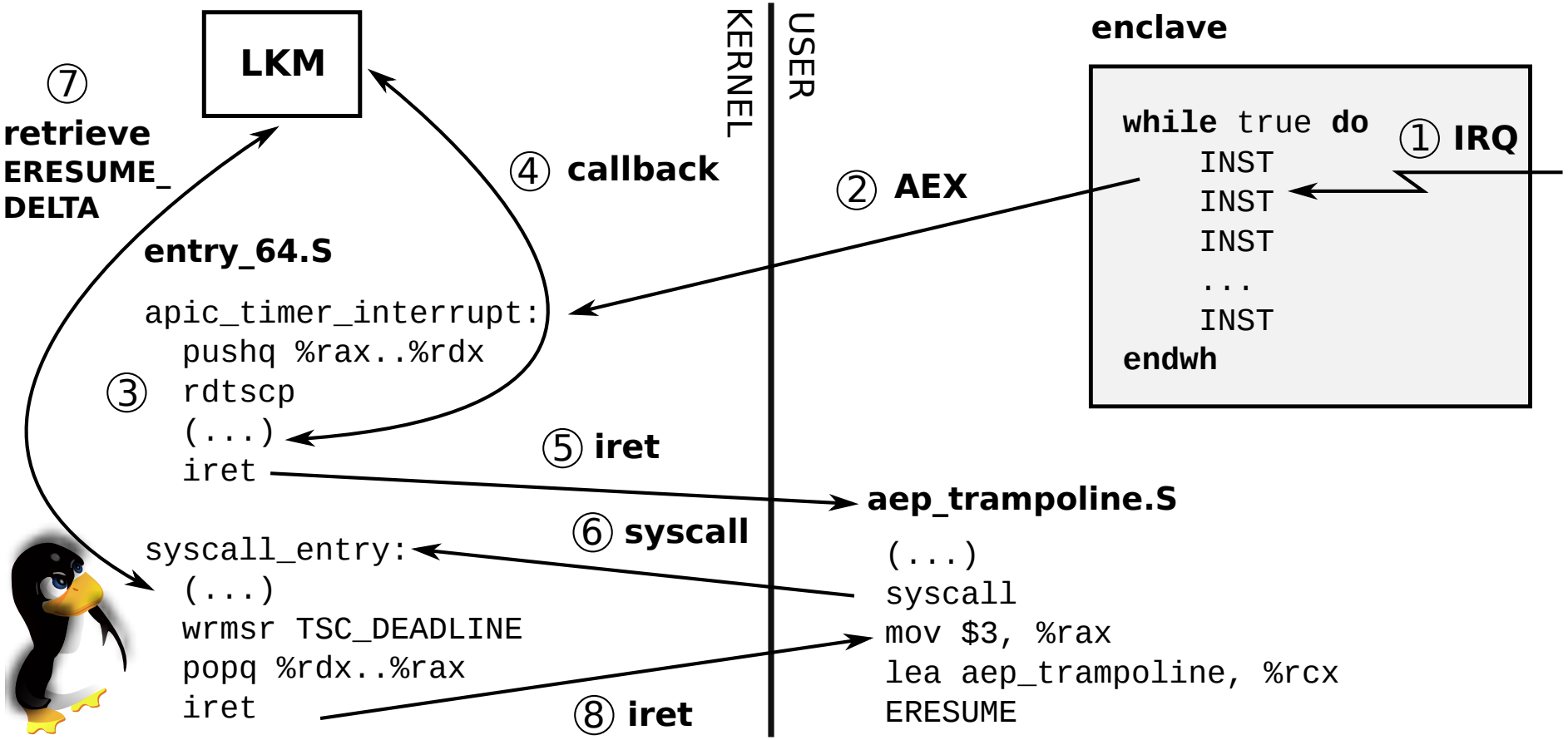
Interrupting and Resuming Enclaves



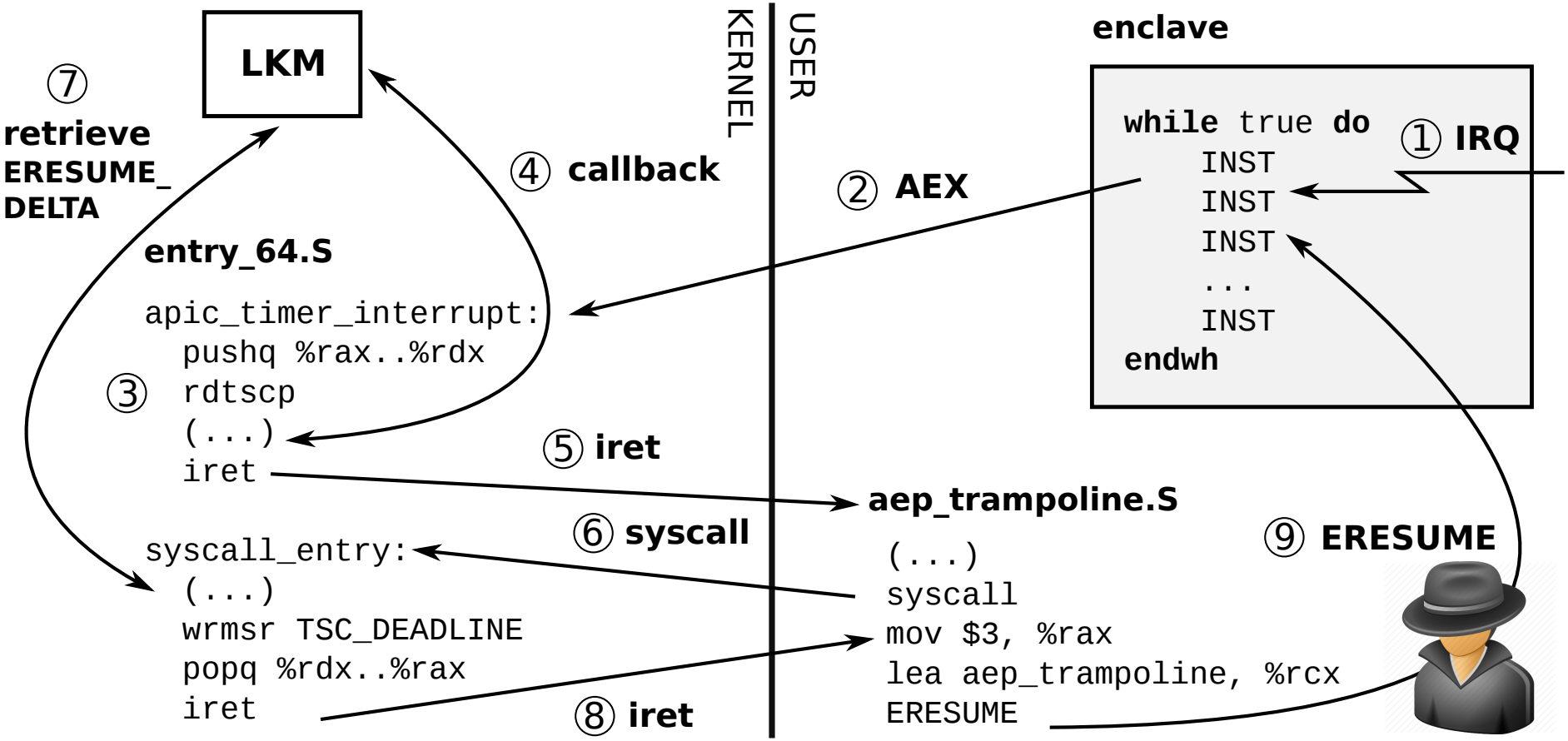
Interrupting and Resuming Enclaves



Interrupting and Resuming Enclaves

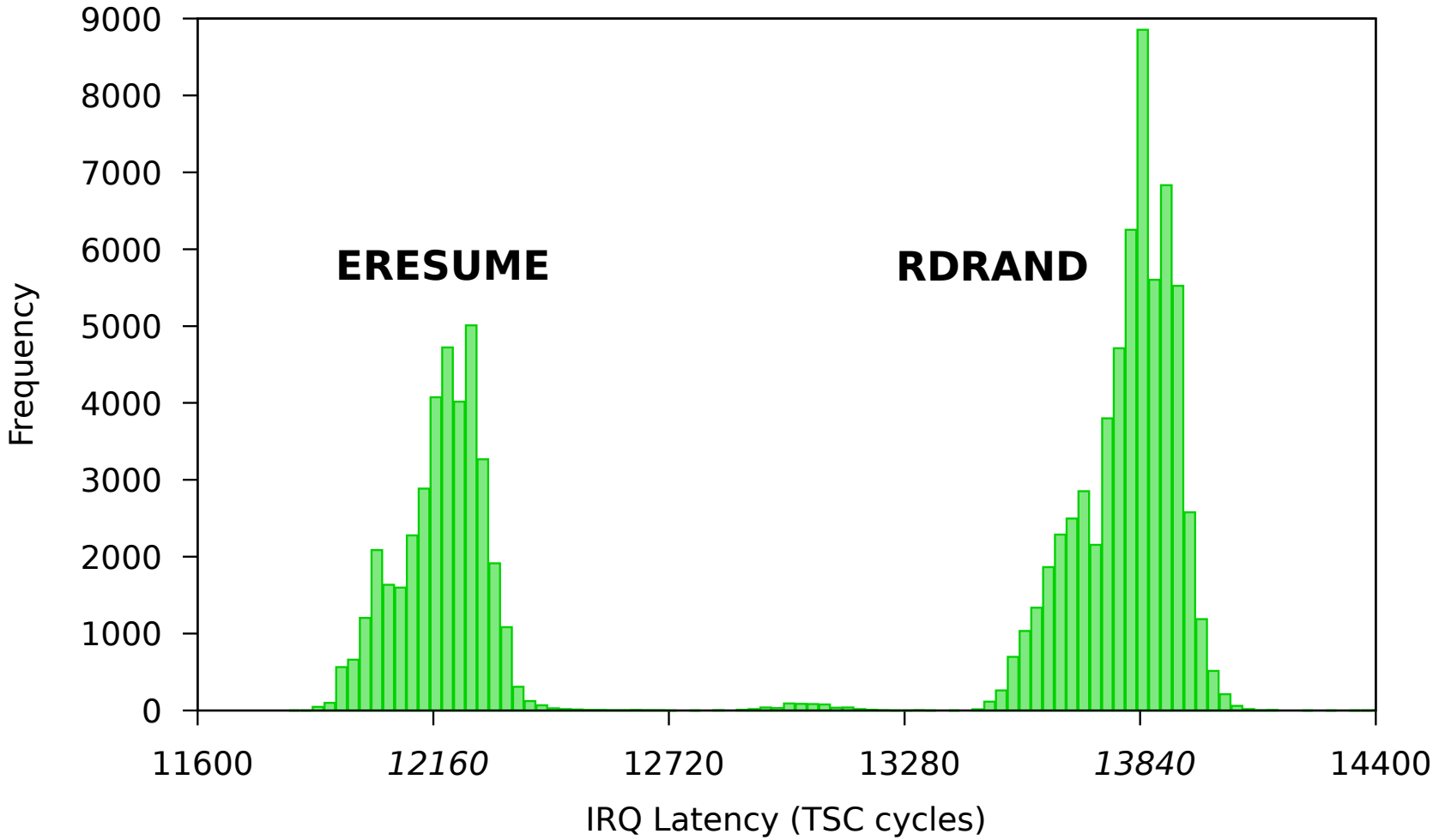


Interrupting and Resuming Enclaves



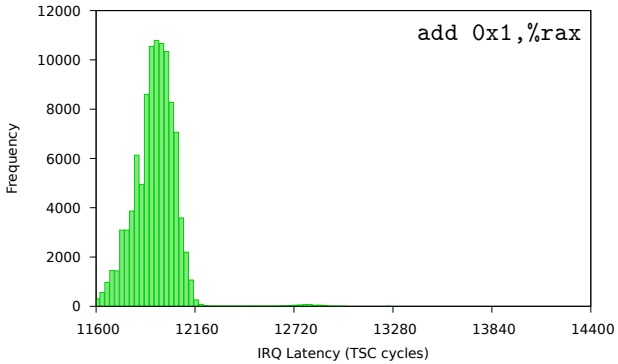
Configuring the Timer Interrupt

$\Rightarrow$ *RDRAND execution time >> cycles to complete ERESUME*

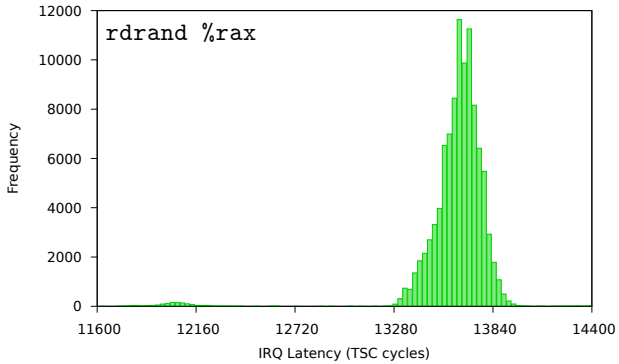


Microbenchmarks: x86 Latency Distributions

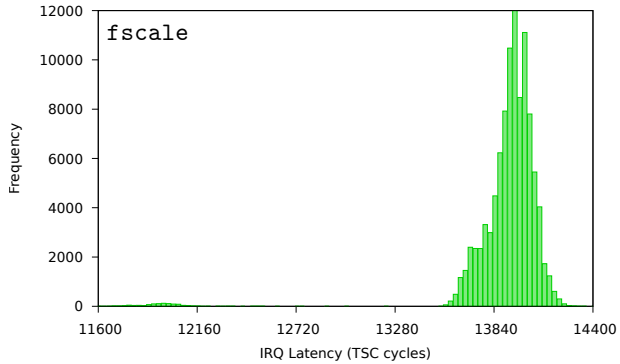
Note: IRQ latency leaks *interrupted instruction type*



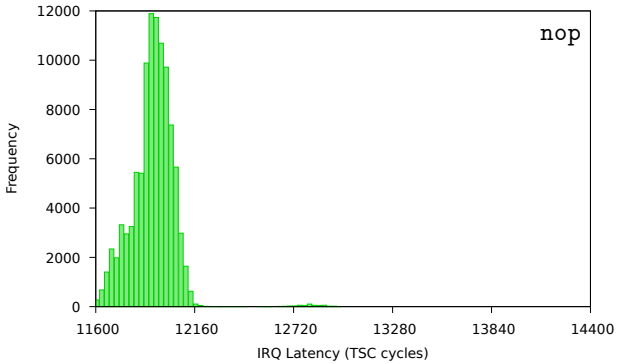
Register increment



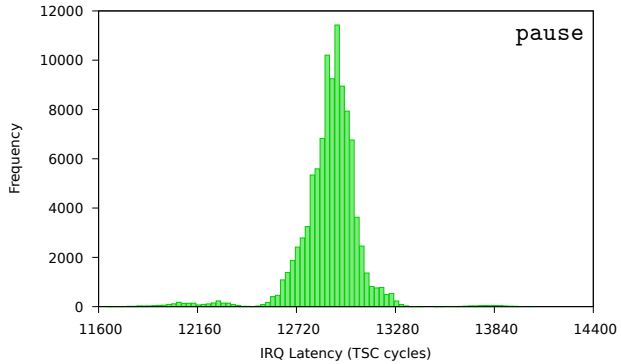
Hardware-level random number



Floating point operation



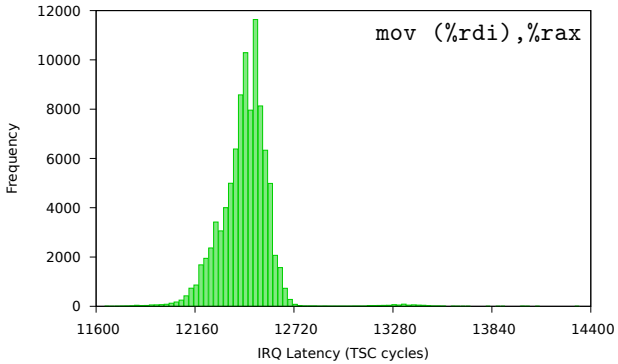
No-operation



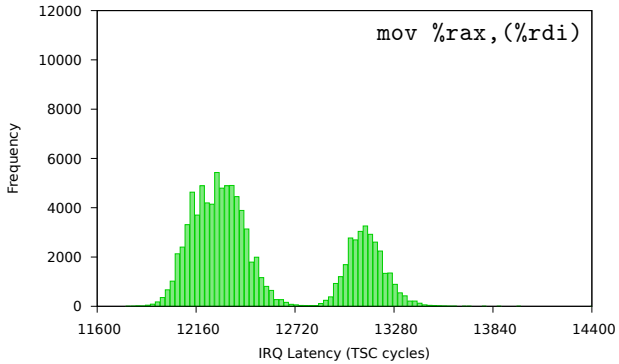
Spin loop hint (delaying nop)

Microbenchmarks: Data Caching Behavior

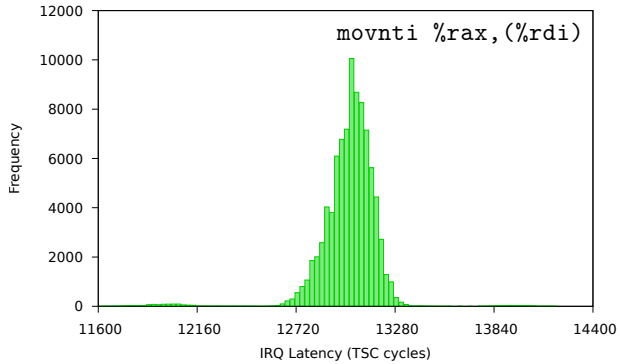
Note: IRQ latency leaks *micro-architectural state*



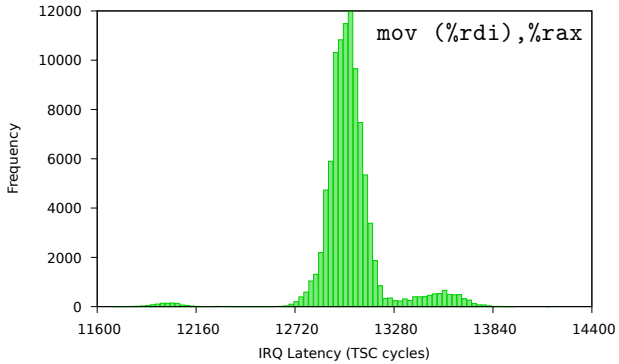
Enclave load



Enclave store



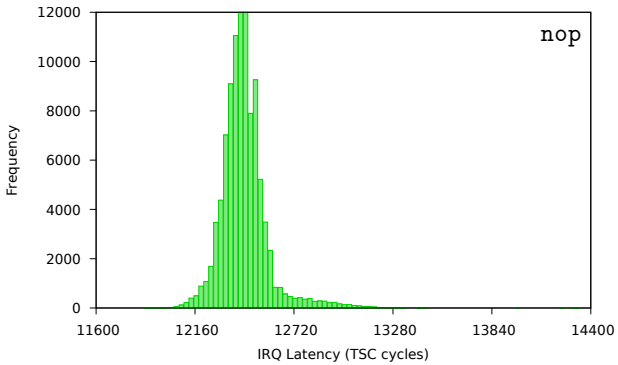
Enclave store non-temporal



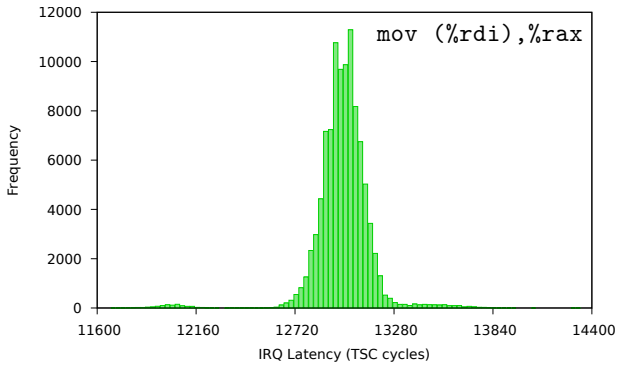
Unprotected flush + load

Microbenchmarks: Address Translation Latency

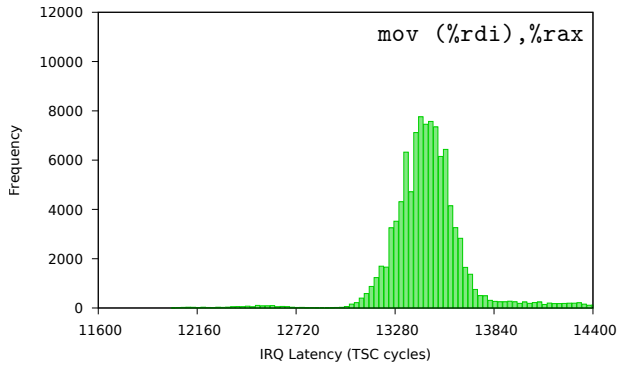
Note: IRQ latency leaks *page accesses*



Nop + flush code PTE



Load + flush data PTE

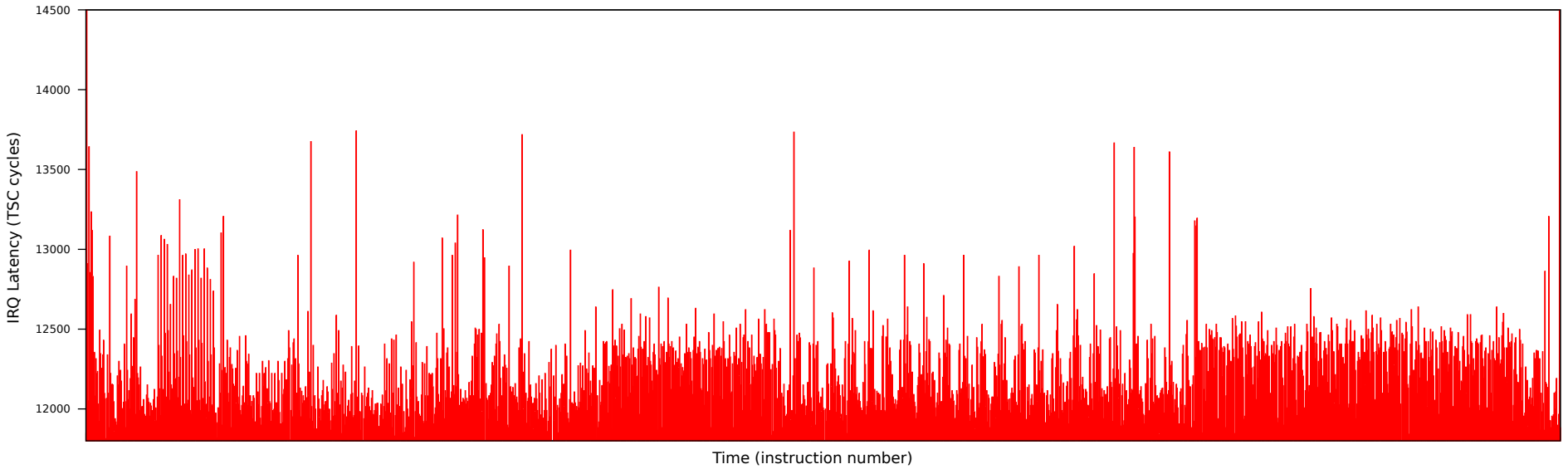


Load + flush code/data PTE

Macrobenchmark: Modular Exponentiation

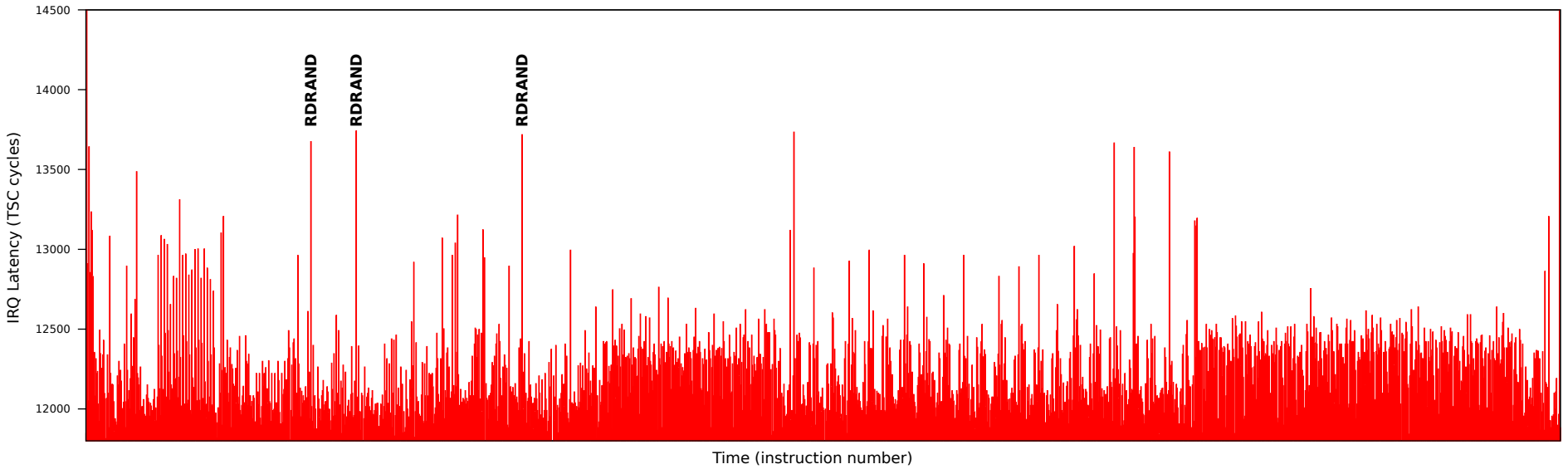
function SQUARE_AND_MULTIPLY(c,d,e,n)
    $r \leftarrow rand()$
    $c \leftarrow c * r^e \bmod n$
    $m \leftarrow 1$
    for most to least significant bit b in d **do**
        $m \leftarrow m^2 \bmod n$
        if b **then**
            $m \leftarrow m * c \bmod n$
        end if
    end for
    return $m * r^{-1} \bmod n$
end function

Extracted Interrupt Latency Trace



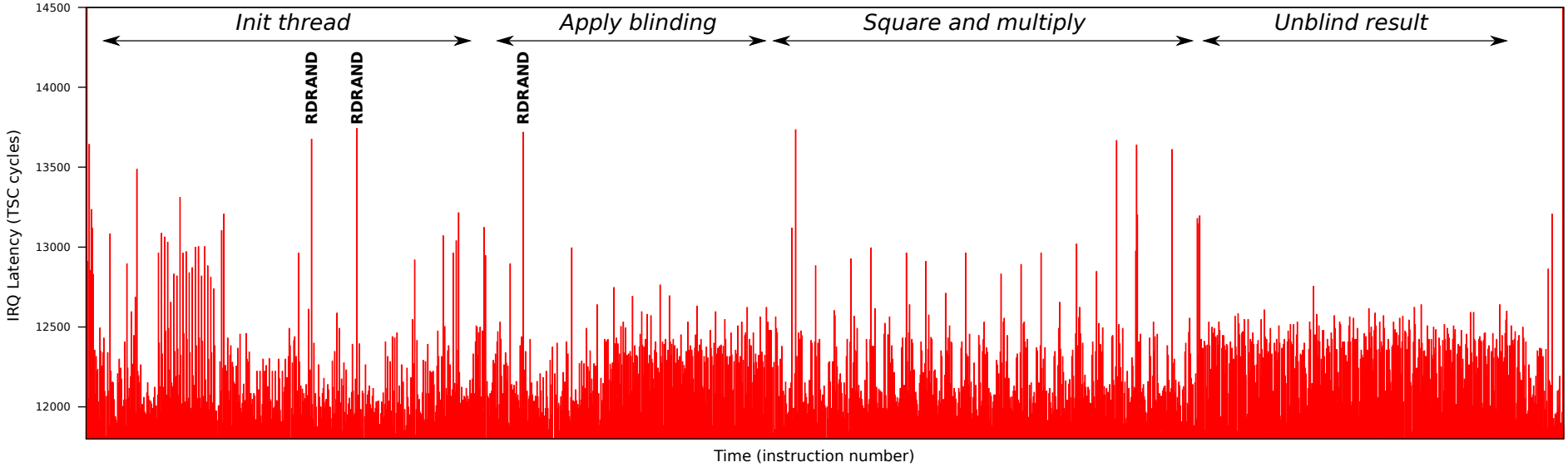
- Extracted from a single **dummy RSA decryption**

Extracted Interrupt Latency Trace



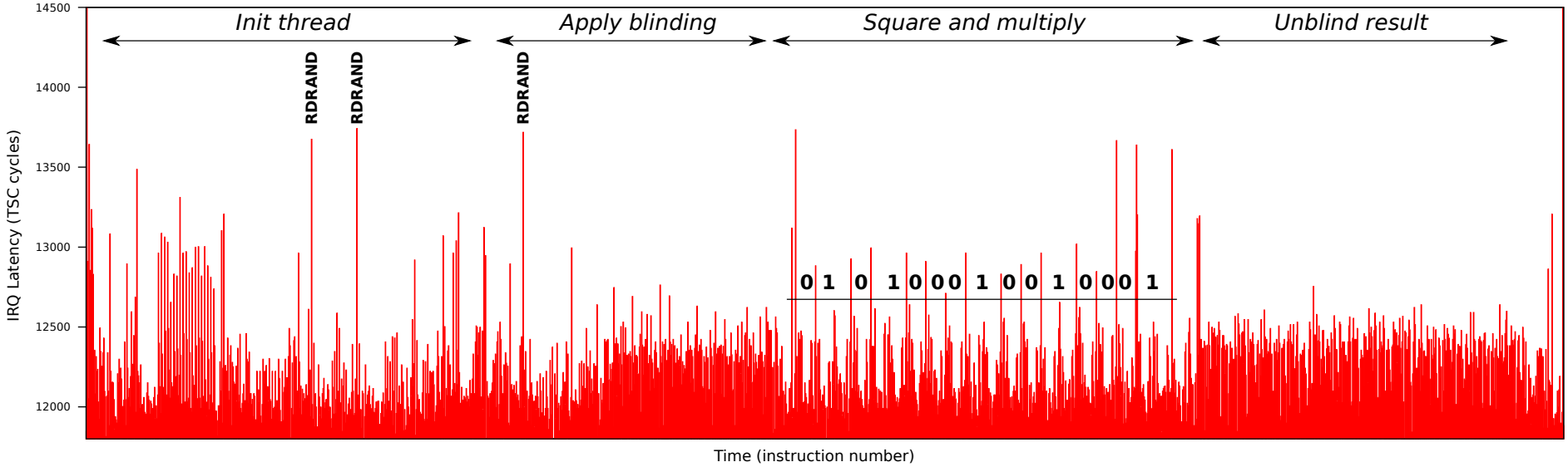
- Extracted from a single **dummy RSA decryption**
- **Distinct instructions** for stack canary + blinding: RDRAND

Extracted Interrupt Latency Trace



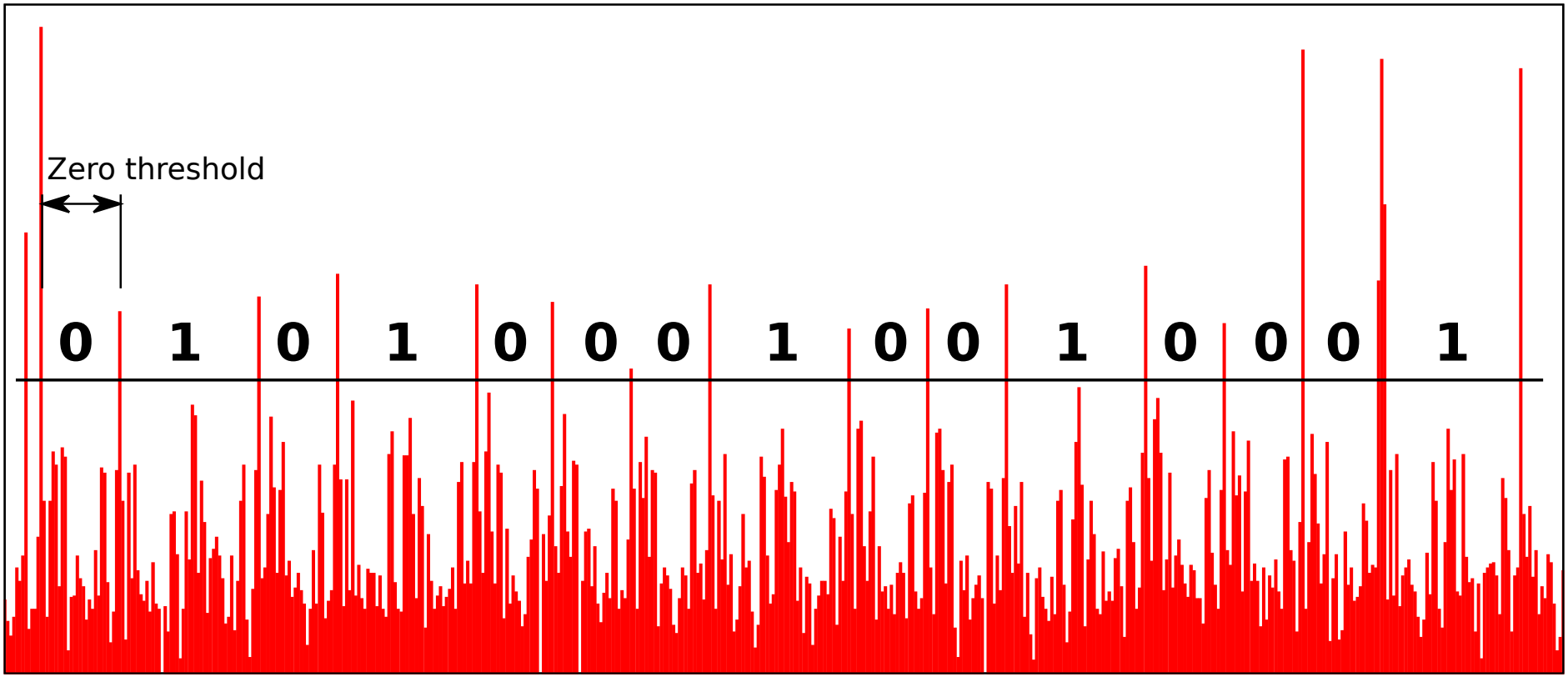
- Extracted from a single **dummy RSA decryption**
- **Distinct instructions** for stack canary + blinding: RDRAND
- Sharply defined **algorithm phases**

Extracted Interrupt Latency Trace



- Extracted from a single **dummy RSA decryption**
- **Distinct instructions** for stack canary + blinding: RDRAND
- Sharply defined **algorithm phases**
- Full 16-bit **key recovery**

Extracted Interrupt Latency Trace



Flush page table entry for *global variable accessed every loop iteration*

Road Map

Conclusion

$\Rightarrow$ (First) remote side-channel applicable to **embedded + high-end** trusted computing hardware

Conclusion

$\Rightarrow$ (First) remote side-channel applicable to **embedded + high-end** trusted computing hardware

IRQ latency trace reveals **micro-architectural** behavior:

- Lots of *noise/non-determinism* on modern CPUs
- Abuse subtle timing differences with *machine learning*?

Conclusion

$\Rightarrow$ (First) remote side-channel applicable to **embedded + high-end** trusted computing hardware

IRQ latency trace reveals **micro-architectural** behavior:

- Lots of *noise/non-determinism* on modern CPUs
- Abuse subtle timing differences with *machine learning*?

Defense techniques:

- Eliminate *if branches* $\leftrightarrow$ legacy applications
- Sancus *hardware patch* to force worst-case IRQ latency

References I

R. De Clercq, F. Piessens, D. Schellekens, and I. Verbauwhede.
Secure interrupts on low-end microcontrollers.
In *Application-specific Systems, Architectures and Processors (ASAP), 2014 IEEE 25th International Conference on*, pp. 147–152. IEEE, 2014.

P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan.
TrustLite: A security architecture for tiny embedded devices.
In *Proceedings of the Ninth European Conference on Computer Systems*, pp. 10:1–10:14. ACM, 2014.

J. Noorman, J. Van Bulck, J. T. Mühlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Götzfried, T. Müller, and F. Freiling.
Sancus 2.0: A low-cost security architecture for IoT devices.
ACM Transactions on Privacy and Security (TOPS), 2017.
Accepted for publication.

J. Van Bulck, J. Noorman, J. T. Mühlberg, and F. Piessens.
Towards availability and real-time guarantees for protected module architectures.
In *MODULARITY Companion Proceedings '16*, pp. 146–151, New York, 2016. ACM.

Y. Xu, W. Cui, and M. Peinado.
Controlled-channel attacks: Deterministic side channels for untrusted operating systems.
In *2015 IEEE Symposium on Security and Privacy*, pp. 640–656. IEEE, 2015.