

Revizor: A Platform for Side Channel Testing

By Oleksii Oleksenko

Senior Researcher @ Azure Research, Microsoft



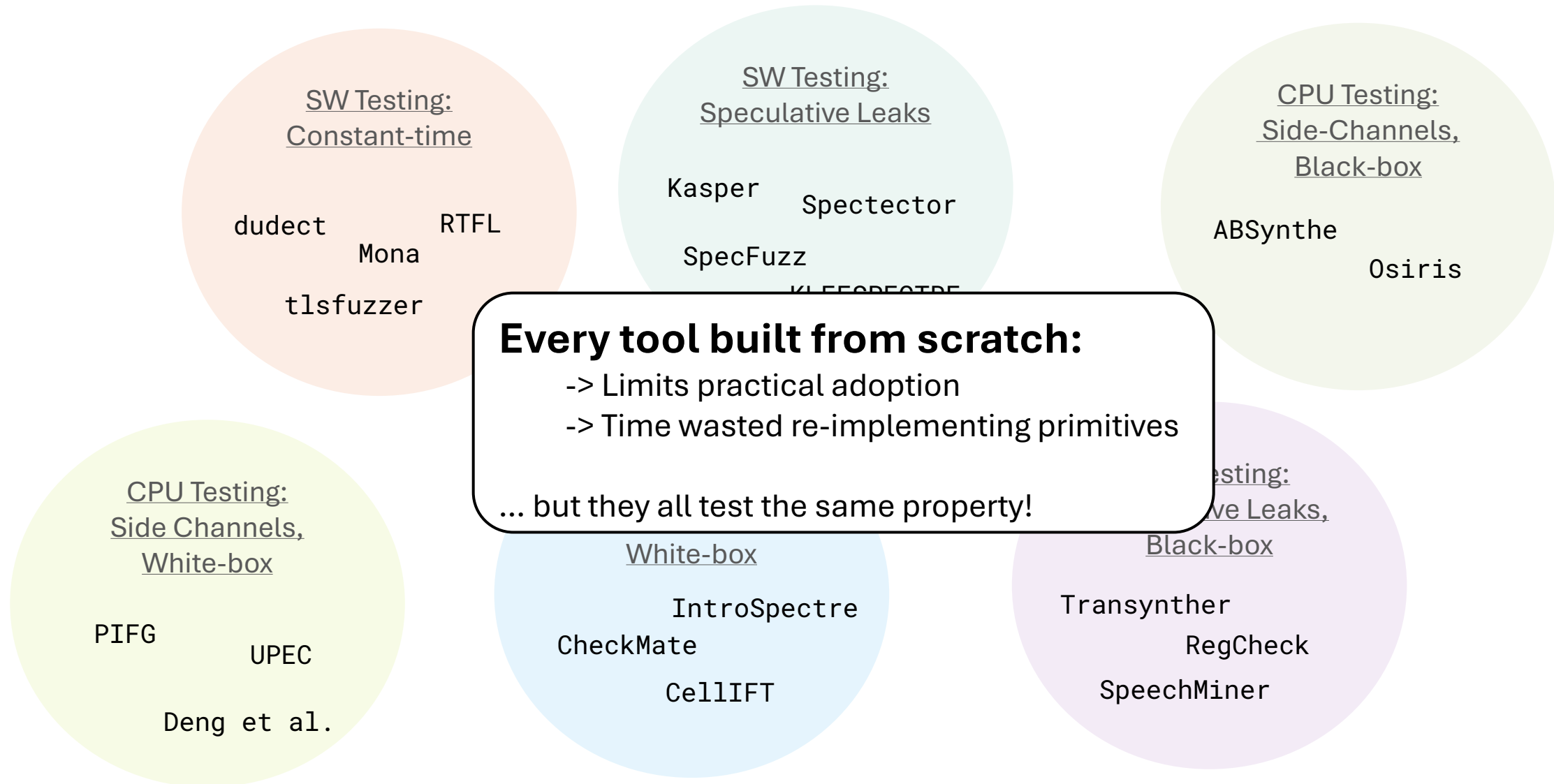
This Talk

- We are making a unified platform for side-channel testing
- In this talk, I will...
 - ...explain the key ideas behind the platform
 - ...brag about the cool things that we already built
 - ...try to convince you to build even more cool things using this platform

High-level Concepts



Why do we need a unified platform?



Tested Property: Information leakage

Information leakage occurs when side-channel traces (such as execution time, cache evictions) depend on the secret data being processed by a program.



Given:

- program P
- pair of inputs I_1 and I_2

$\text{Trace}(P, I)$ is the side-channel trace observed by the attacker when executing P with I

Information leakage occurs when:

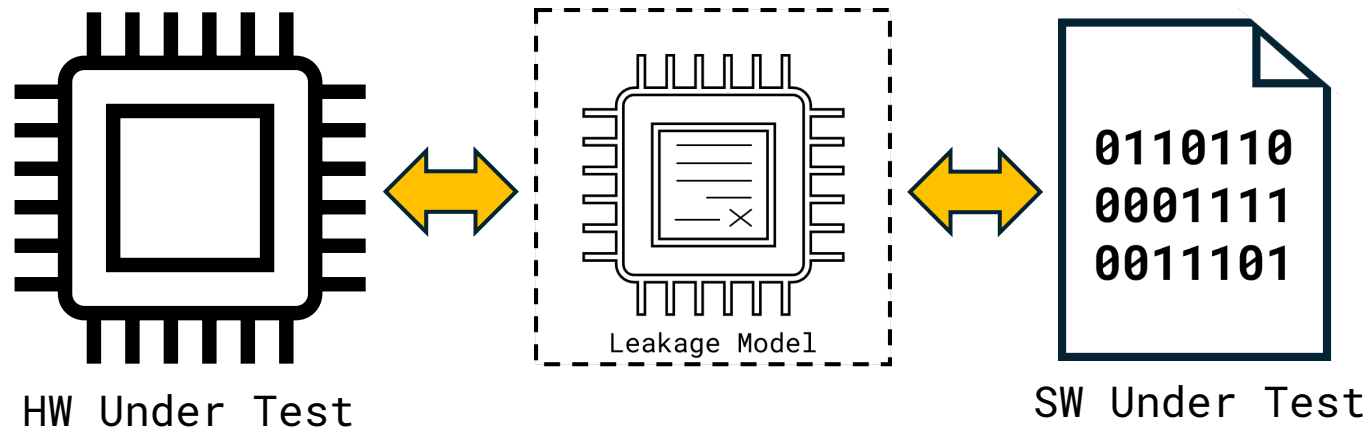
$I_1[\text{public}] = I_2[\text{public}]$ and $I_1[\text{secret}] \neq I_2[\text{secret}]$
and $\text{Trace}(P, I_1) \neq \text{Trace}(P, I_2)$

Universal, but
too broad!

Two-stage testing

(aka HW-SW Contract)

- Idea: Build a simplified model of observable HW effects (leakage model)
- The testing is split into two stages:



Speculative Leakage Model

(aka HW-SW Contract)



- Purpose: simplify and determinize SC measurements
- Takes $P+I$, executes them, and returns *contract trace*
- Records leaked data + emulates observable speculation

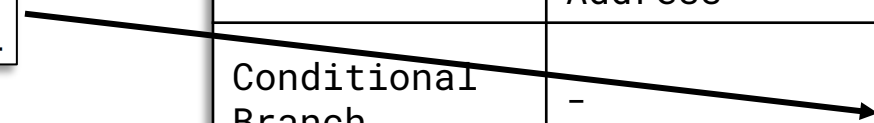
Plain Text Description

A cache timing side channel involves an agent detecting whether a piece of data is present in a specific level of the processor's caches, where its presence may be used to infer some other piece of information. One method to detect whether the data in question is present is to use timers to measure

The *bounds check bypass* method takes advantage of speculative execution after conditional branch instructions. An attacker discovers or causes the creation of 'confused deputy' code which allows the attacker to cause speculative operations to reveal information not normally accessible to the attacker.

Contract for cache-based leakage on a CPU with branch prediction

Instructions	Observation	Speculation
Load	Expose Address	-
Store	Expose Address	-
Conditional Branch	-	Speculatively take wrong target
Other	-	-



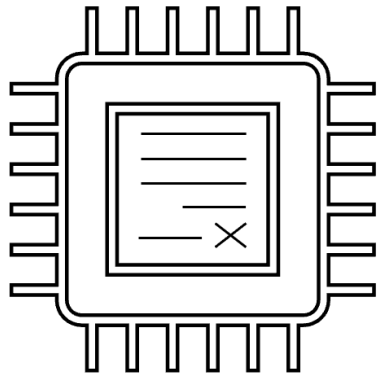
Example: Tracing on a Leakage Model

Program:

```
1. X = *a1
2. if a2 < 50:
3.     X = *a2
```

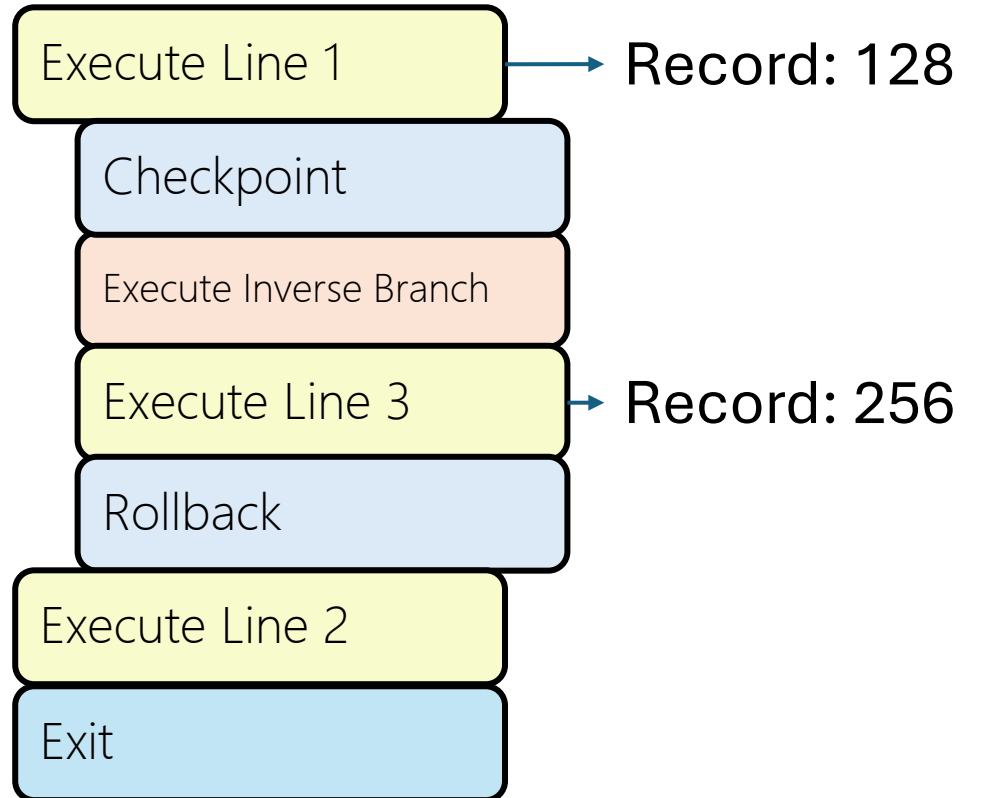
Input:

a1=128; a2=256



Leakage Model

Instructions	Observation	Speculation
Load	Expose Address	-
Store	Expose Address	-
Conditional Branch	-	Speculatively take wrong target
Other	-	-



Contract Trace:
[128, 256]

Leakage Definition, Refined

HW Testing

Does hardware expose any information not exposed by the model?

Given a program P and a pair of inputs (I_1, I_2) , the hardware under test violates the model if:

$$\begin{aligned} &\text{HardwareTrace}(P, I_1) \neq \text{HardwareTrace}(P, I_2) \\ &\quad \text{and} \\ &\text{ContractTrace}(P, I_1) = \text{ContractTrace}(P, I_2) \end{aligned}$$

Software Testing

Does a given program leak secrets on the model?

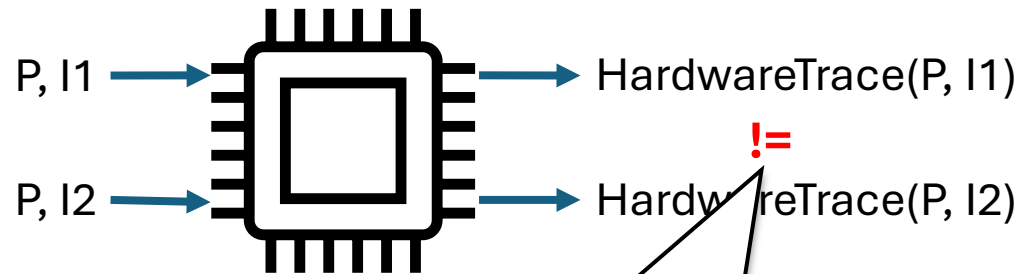
Given a program under test P and pair of inputs (I_1, I_2) , the program has an information leak if:

$$\begin{aligned} &I_1[\text{public}] = I_2[\text{public}] \quad \text{and} \quad I_1[\text{secret}] \neq I_2[\text{secret}] \\ &\quad \text{and} \\ &\text{ContractTrace}(P, I_1) \neq \text{ContractTrace}(P, I_2) \end{aligned}$$

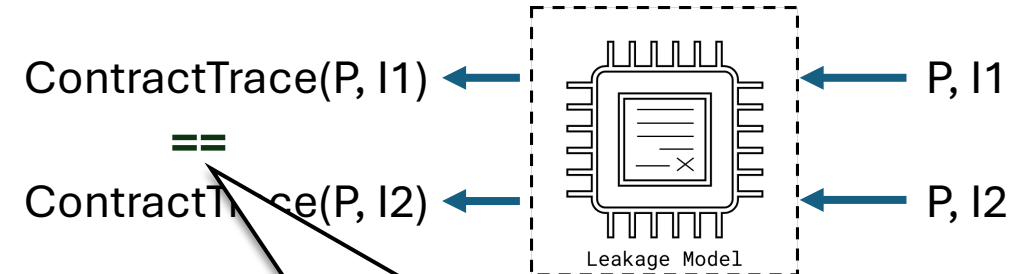
Intuition

$\text{HardwareTrace}(P, I1) \neq \text{HardwareTrace}(P, I2)$
and
 $\text{ContractTrace}(P, I1) == \text{ContractTrace}(P, I2)$

Key Insight: This property avoids defining what is a secret, and instead relies on “expected leaked information”

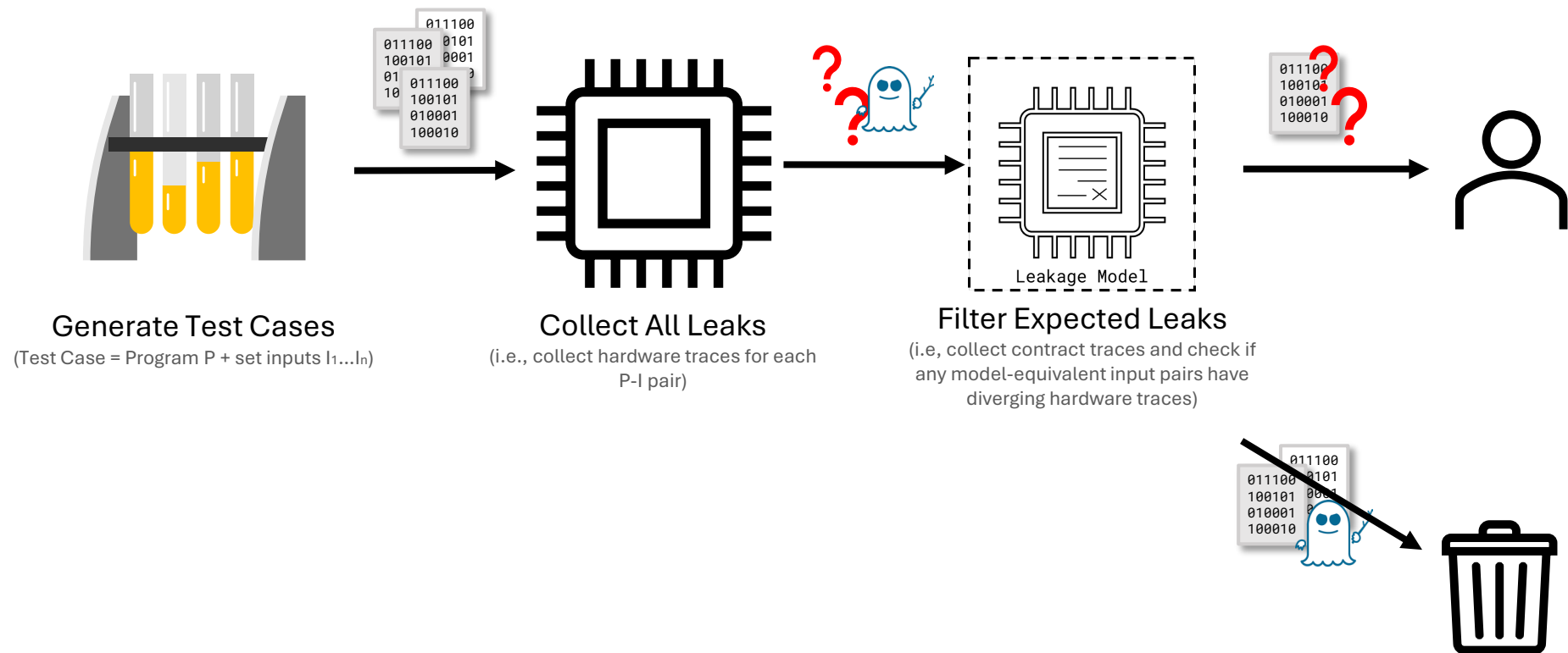


Inequality means that **some** of the differences between $I1$ and $I2$ are **exposed**

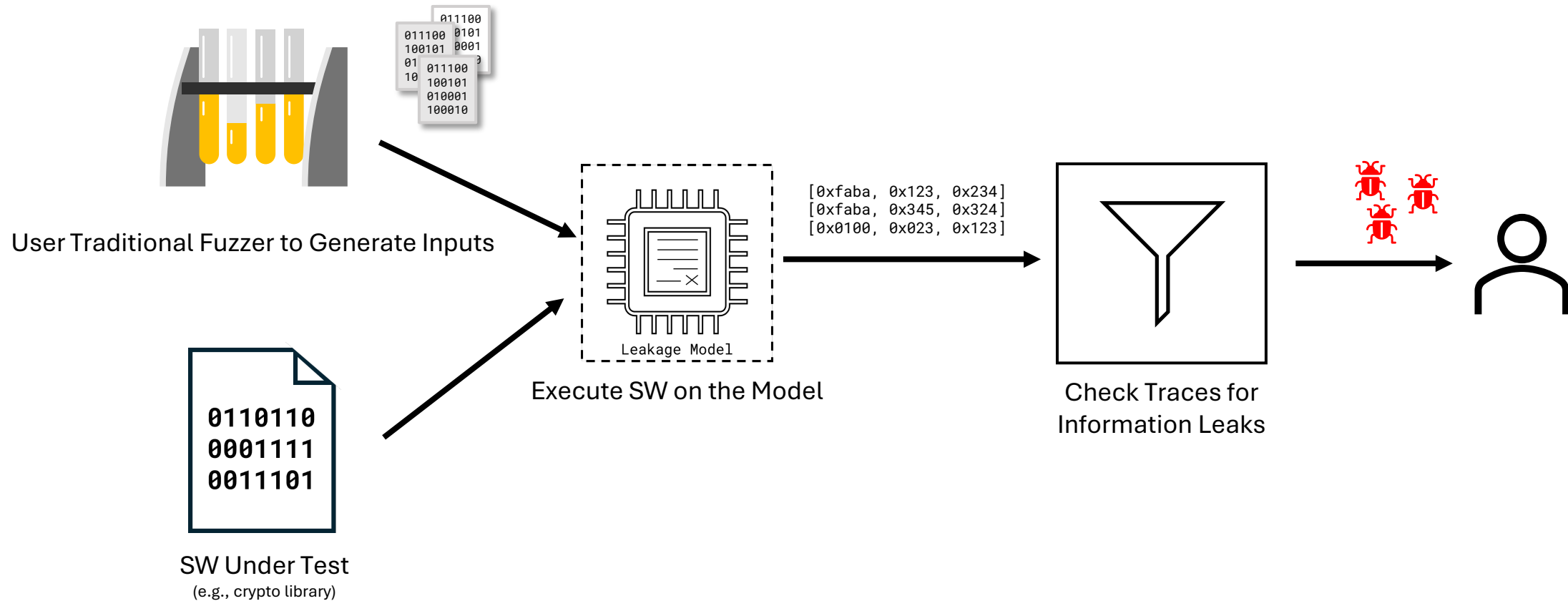


Equality means that **none** of the differences between $I1$ and $I2$ are **exposed**

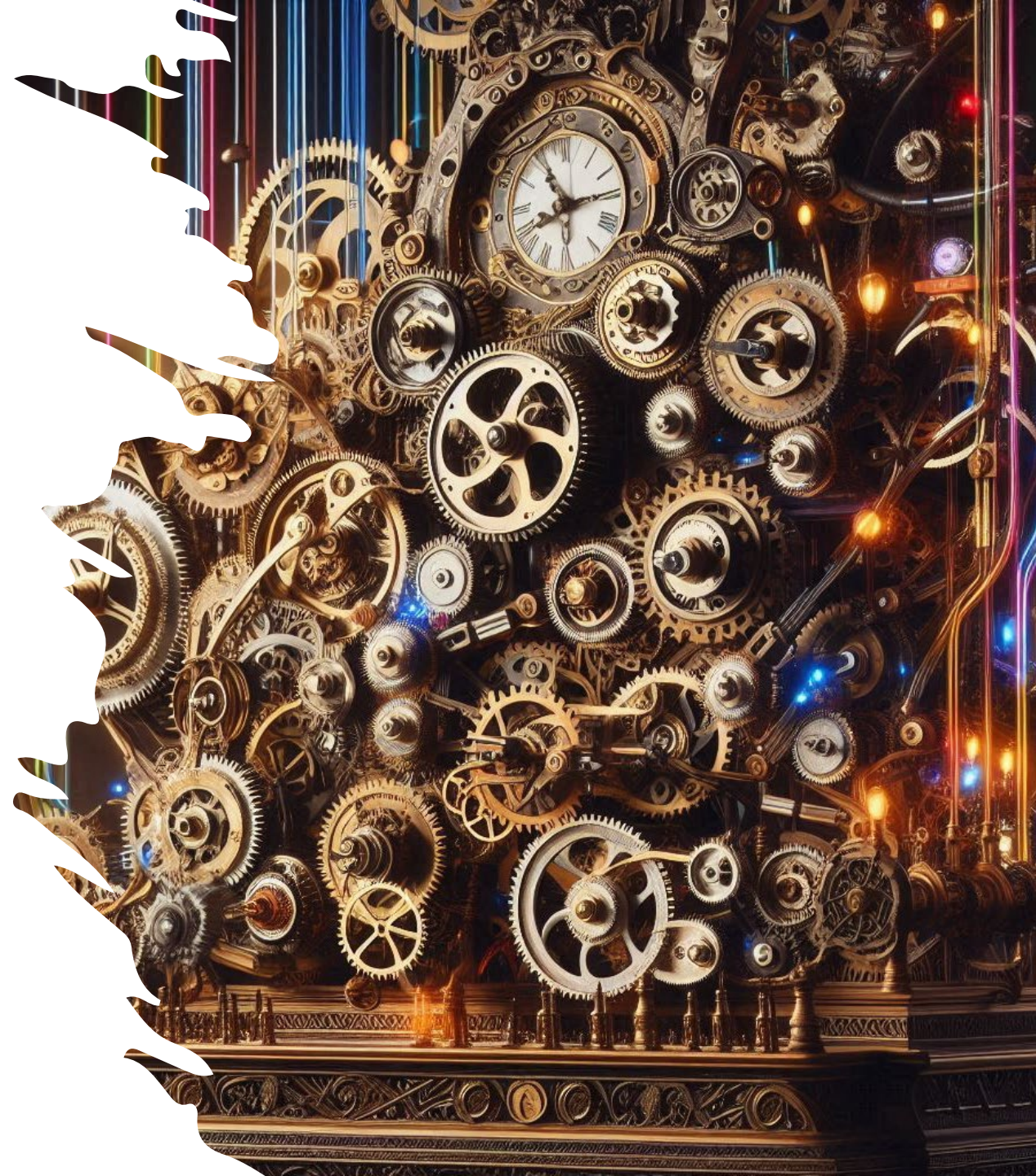
Testing Architecture: Hardware Side



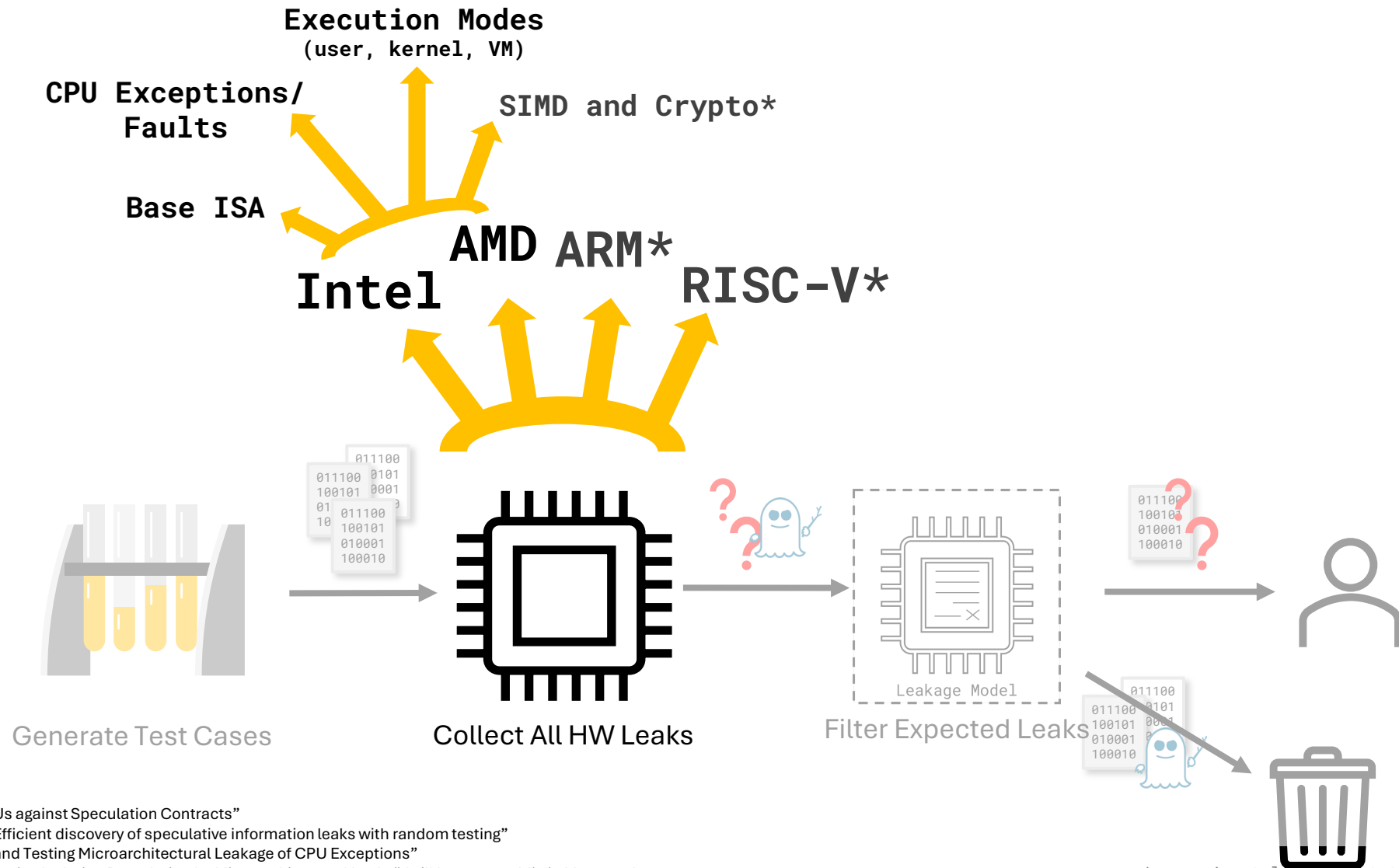
Testing Architecture: Software Side



Current Applications



Application 1: Black-box CPU testing



* experimental implementations and on-going work

Oleksenko et al. "Revizor: Testing Black-Box CPUs against Speculation Contracts"
Oleksenko et al. "Hide and Seek with Spectres: Efficient discovery of speculative information leaks with random testing"
Hofmann et al. "Speculation at Fault: Modeling and Testing Microarchitectural Leakage of CPU Exceptions"
Oleksenko et al. "Enter, Exit, Page Fault, Leak: Testing Isolation Boundaries for Microarchitectural Leaks" (will become public in May 2025)

Prog

Proof of concept

- Basic CPU features: arithmetic operations, loads/stores, branches
- Only Intel CPUs
- Reproduced Spectre V1, V4
- Detected a new variant of V1

Detecting new
variants already
with the first
implementation

	x86-64 Subset	Information Leakage Detected?															
		Intel								AMD							
		Core6				Core8 Xeon				Epyc							
	cond: Conditional branches	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	strn: String operations	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	dmul: Division and multiplication	✓	✓	✓	✓	✓	✓	✓	✓	×	×	×	×	×	×	×	×
	flag: Operations on flags	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
	lock: Atomics w/ LOCK prefix	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
	atom: Atomics w/o LOCK prefix	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
	dxftr: Data transfer (load/store)	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
	setc: Conditional byte set	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
	nop : NOP instructions	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
	logi: Logical operations	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
	version	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
		Int1	Int2	AMD1	AMD2	Int1	Int2	AMD1	AMD2	Int1	Int2	AMD1	AMD2	Int1	Int2	AMD1	AMD2
Fault	invalid read-only SMAP	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
PF	non-canonical	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
GP	MPX	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
#BR	A-bit D-bit	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
uCode assist	div by zero	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
#DE	div overflow	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
#UD	undef. opcode	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
#DB + #BP	invalid mode	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Progs

Spectre V1

novel: String Speculation

novel: Division Operand Prediction

MDS

Effectively, tiny specialized OS and novel store-to-load forwarding

novel trigger for MDS, Foreshadow, LVI-Null

LVI-Null

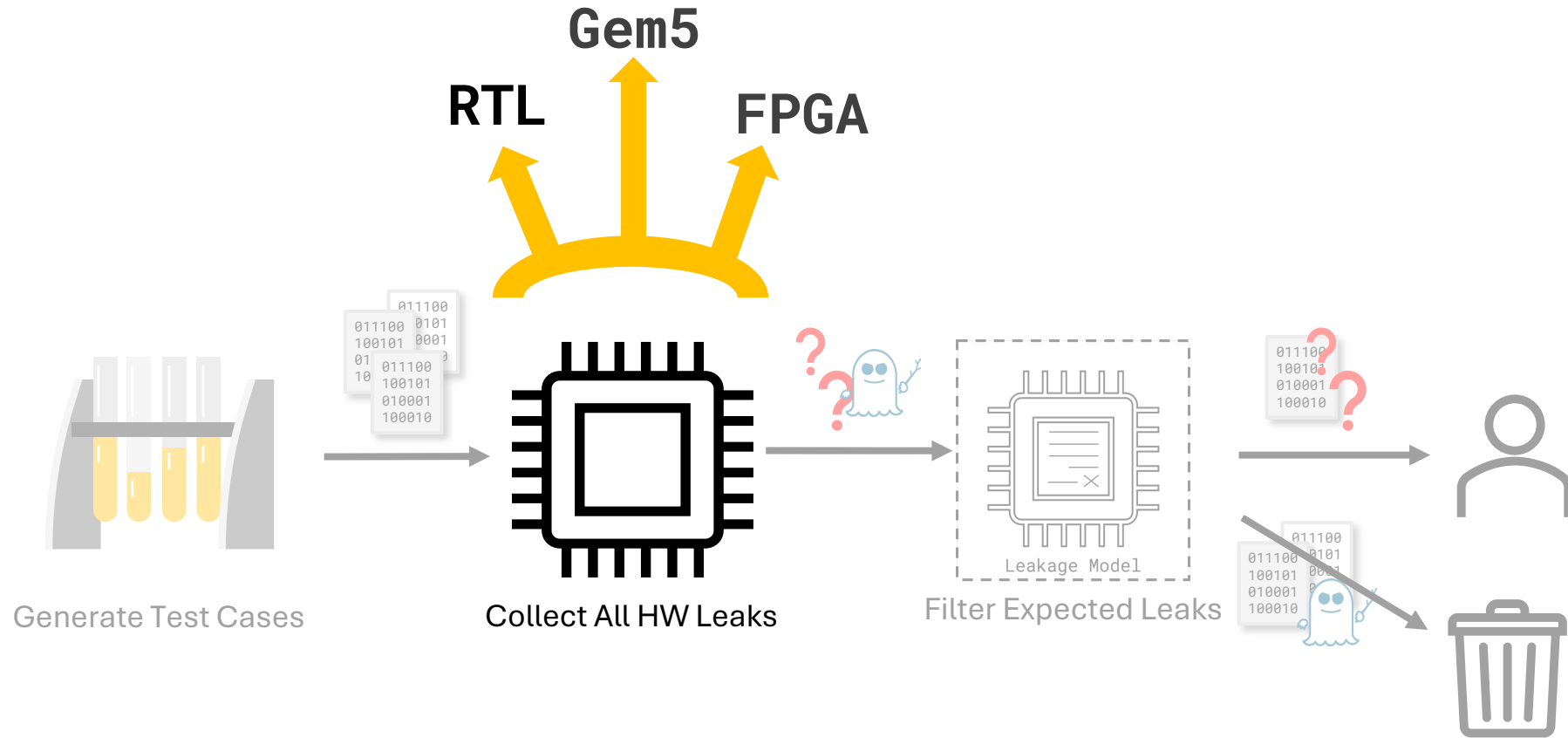
Foreshadow

Meltdown

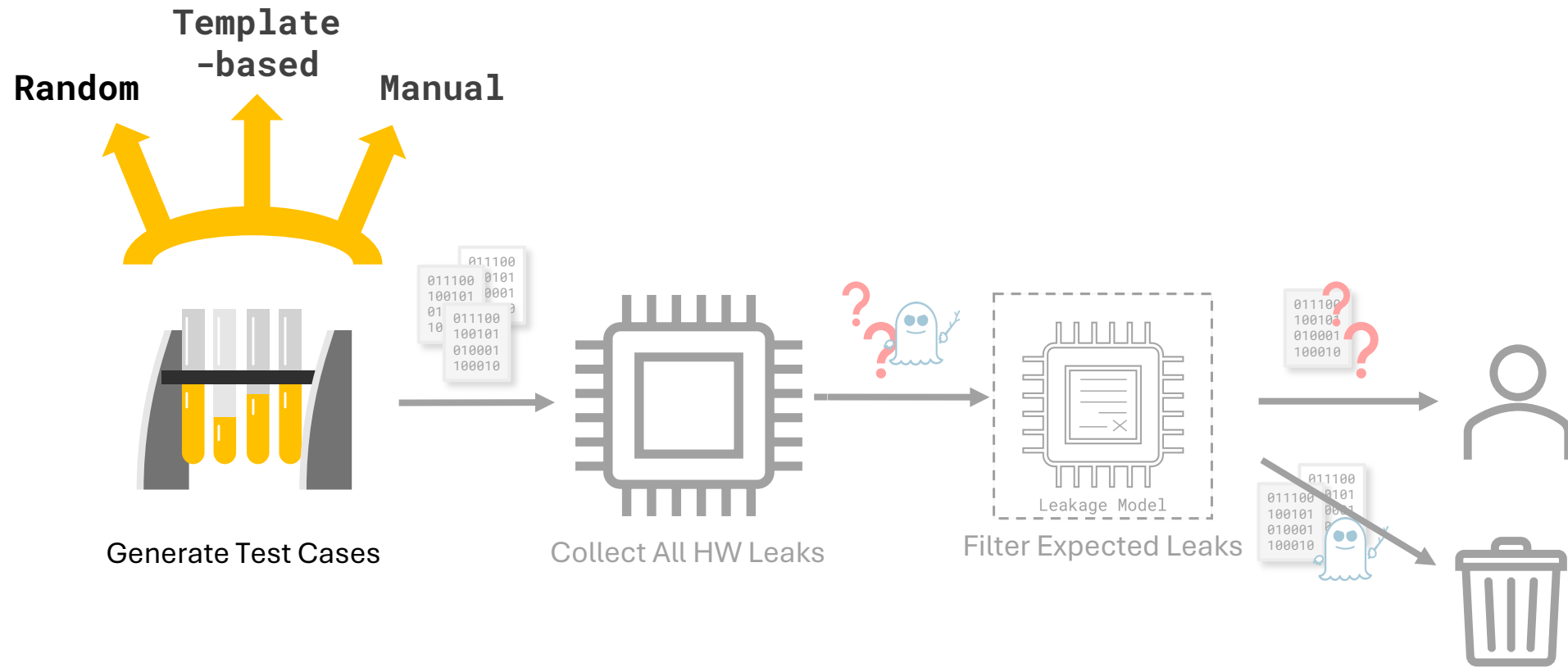
Non-canonical access speculation

novel leak (DSS)

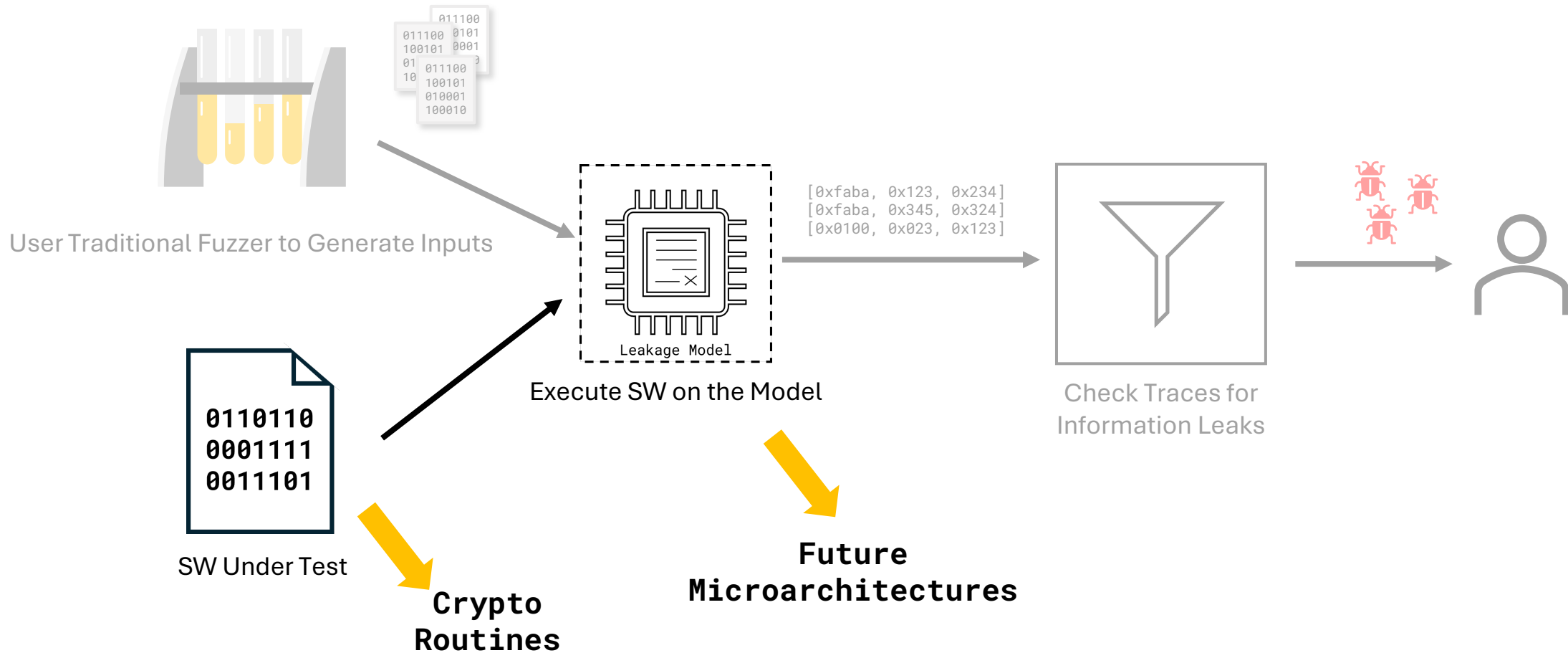
Application 2: White-box CPU testing



Application 3: Aid in Reverse Engineering



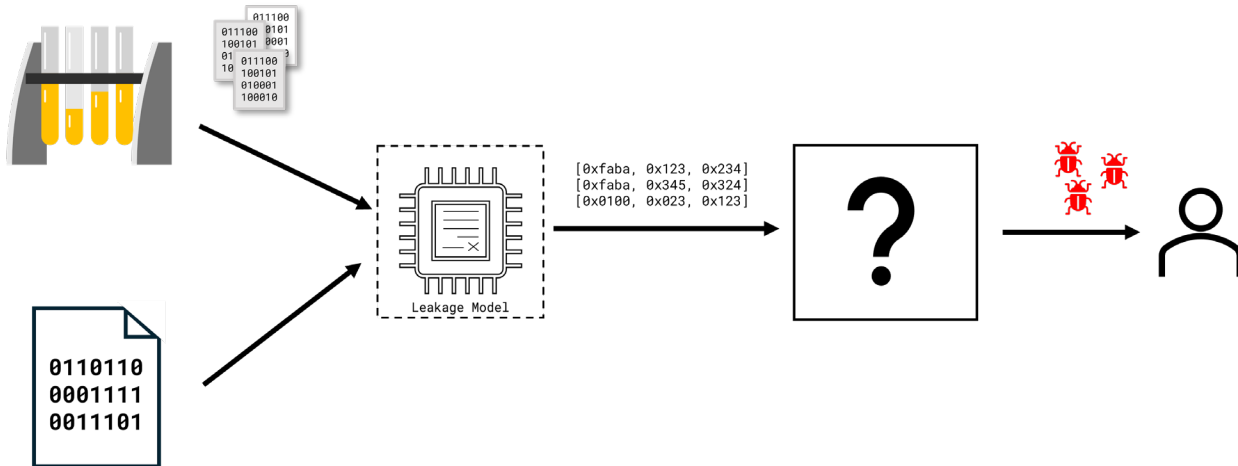
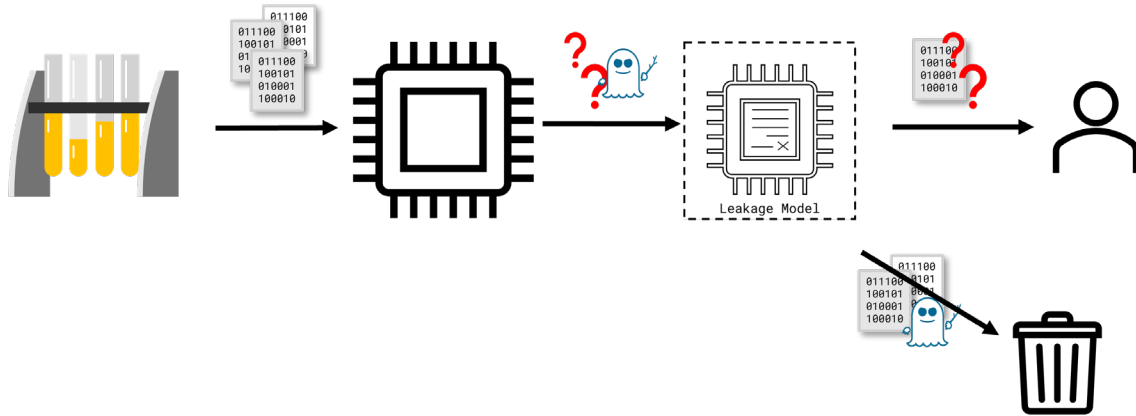
Application 4: Software Testing



Collaboration Opportunities?

- Want to try using Revizor? - Let us know!
 - It's an open-source project: <https://github.com/microsoft/sca-fuzzer>
 - Having trouble? We can arrange a meeting to walk you through
- Want to add new module/expand an existing one? – We are happy to help!
 - PRs a very welcome too
- Any ideas for interesting/unusual applications of the platform?

Revizor



<https://github.com/microsoft/sca-fuzzer>

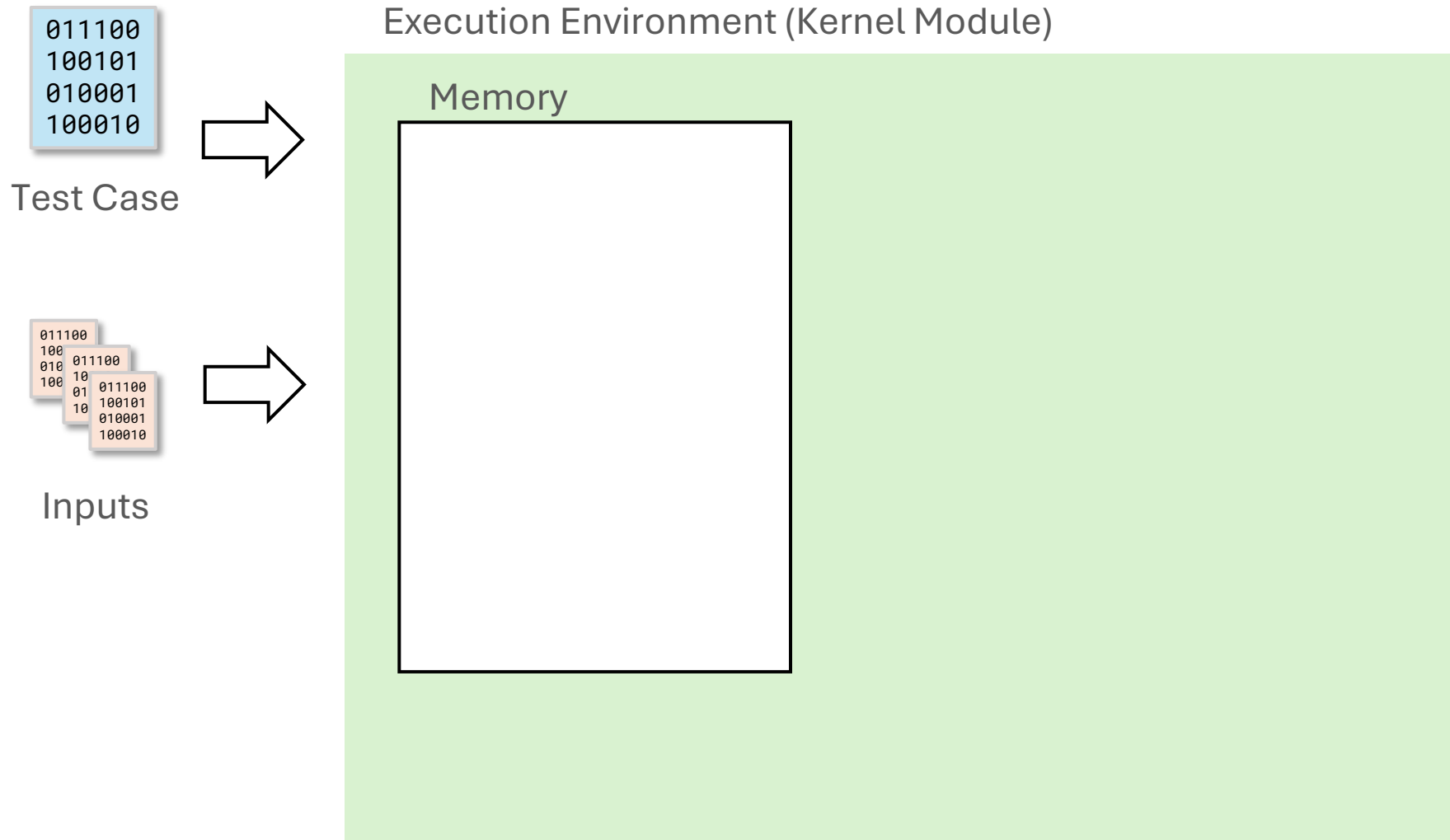


<https://microsoft.github.io/sca-fuzzer/>

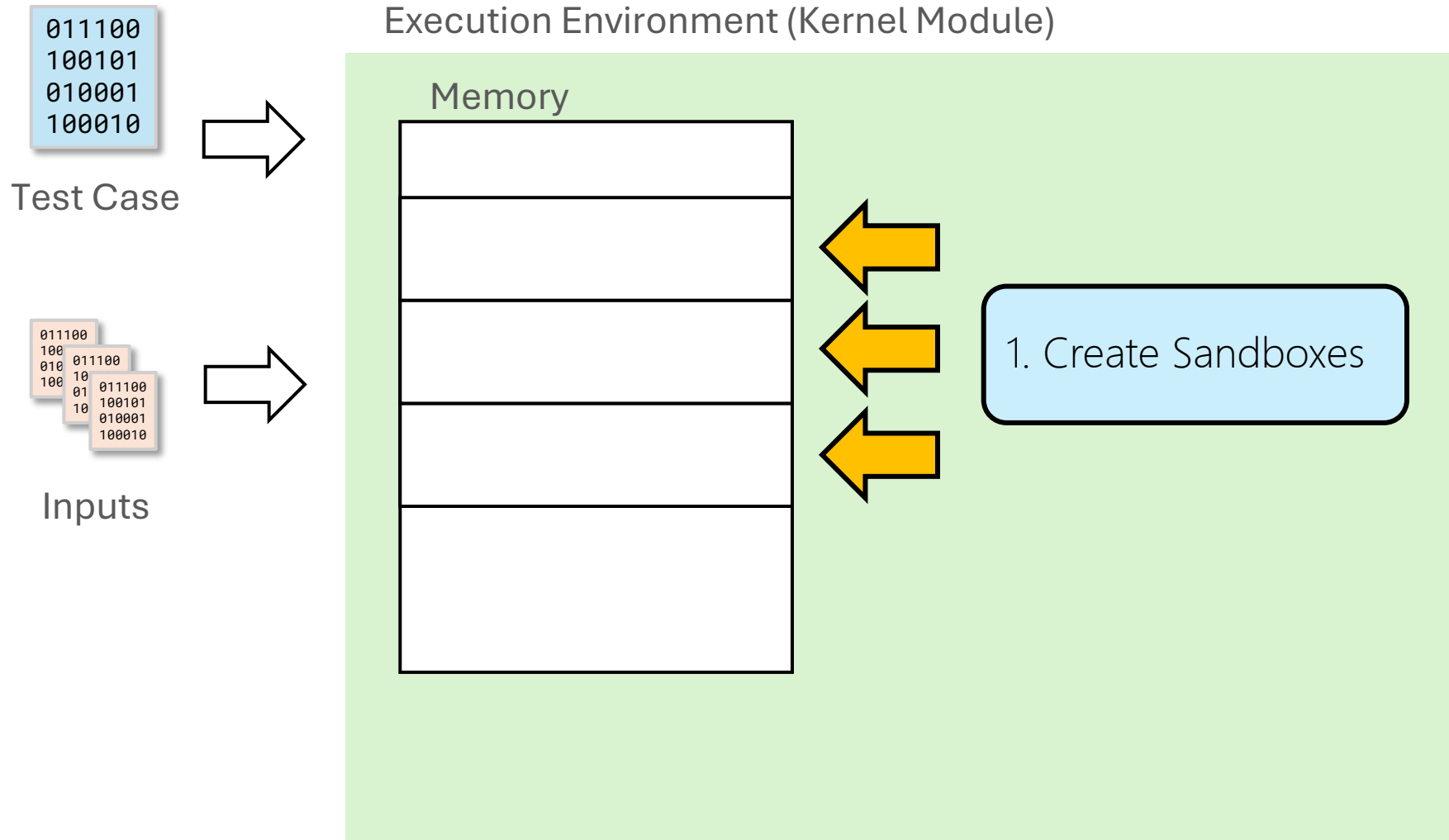
Want an extended demo?
Or a quick tutorial?
Just ask 😊

Backup

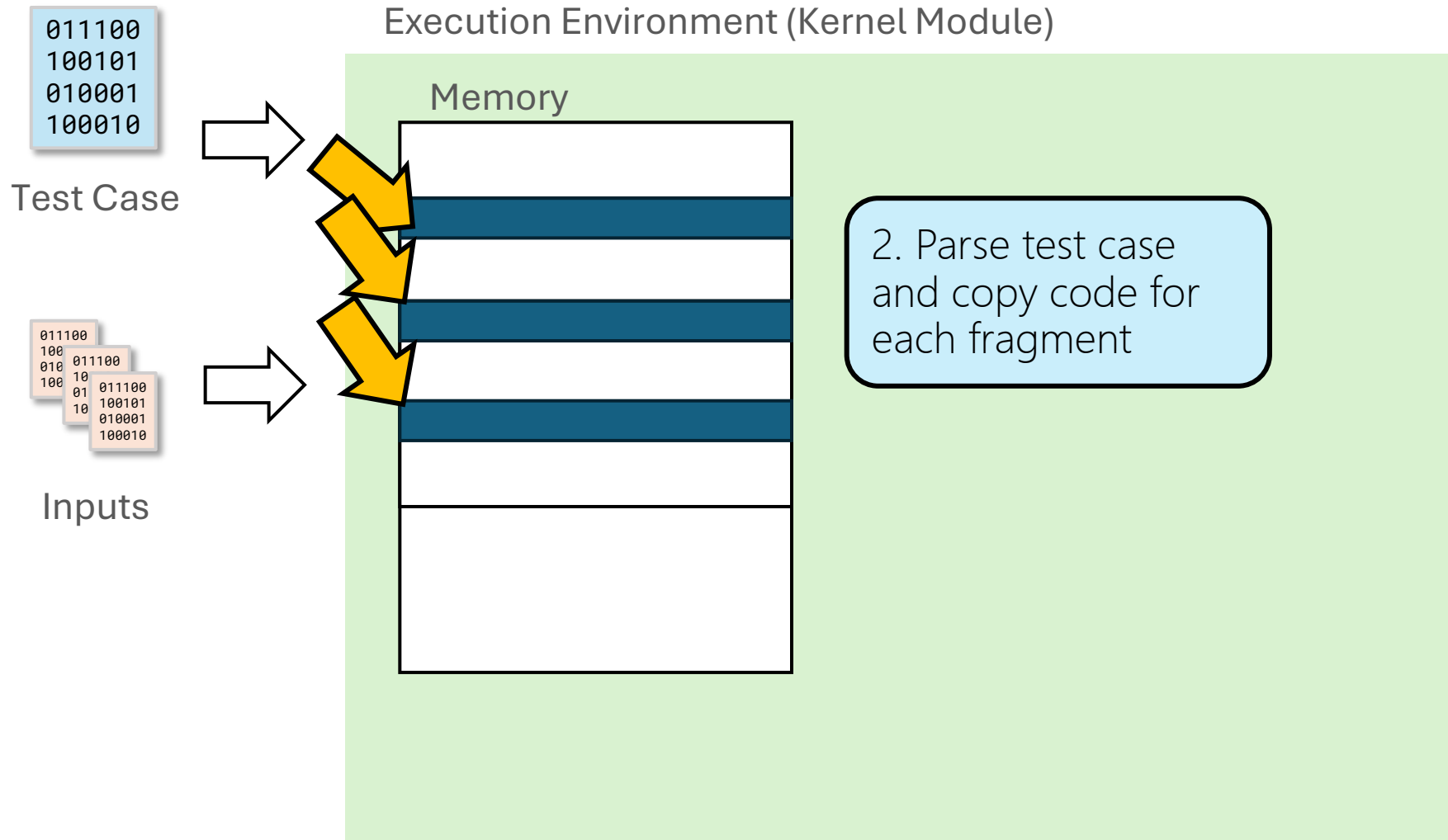
Measurement Process



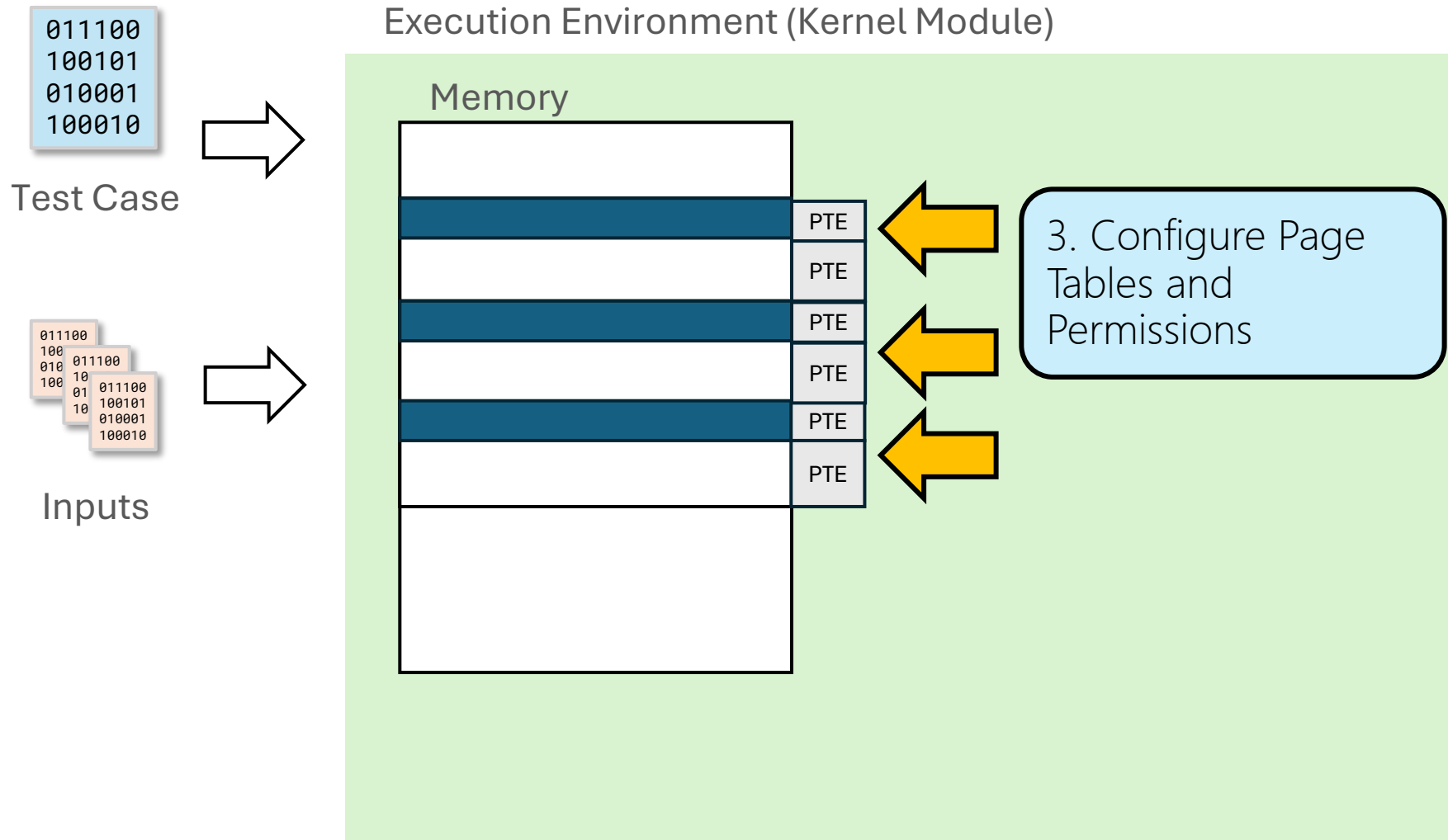
Measurement Process



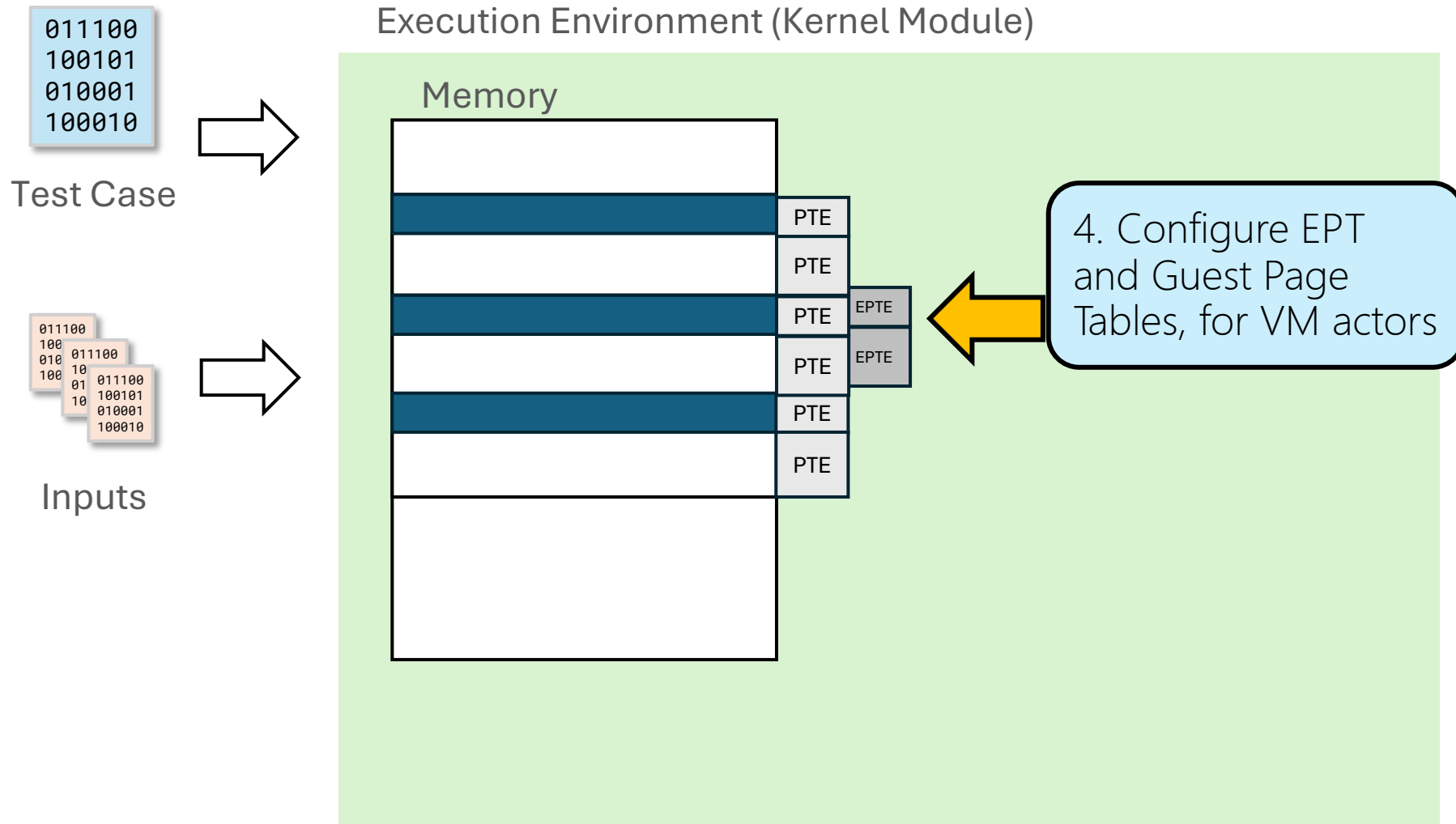
Measurement Process



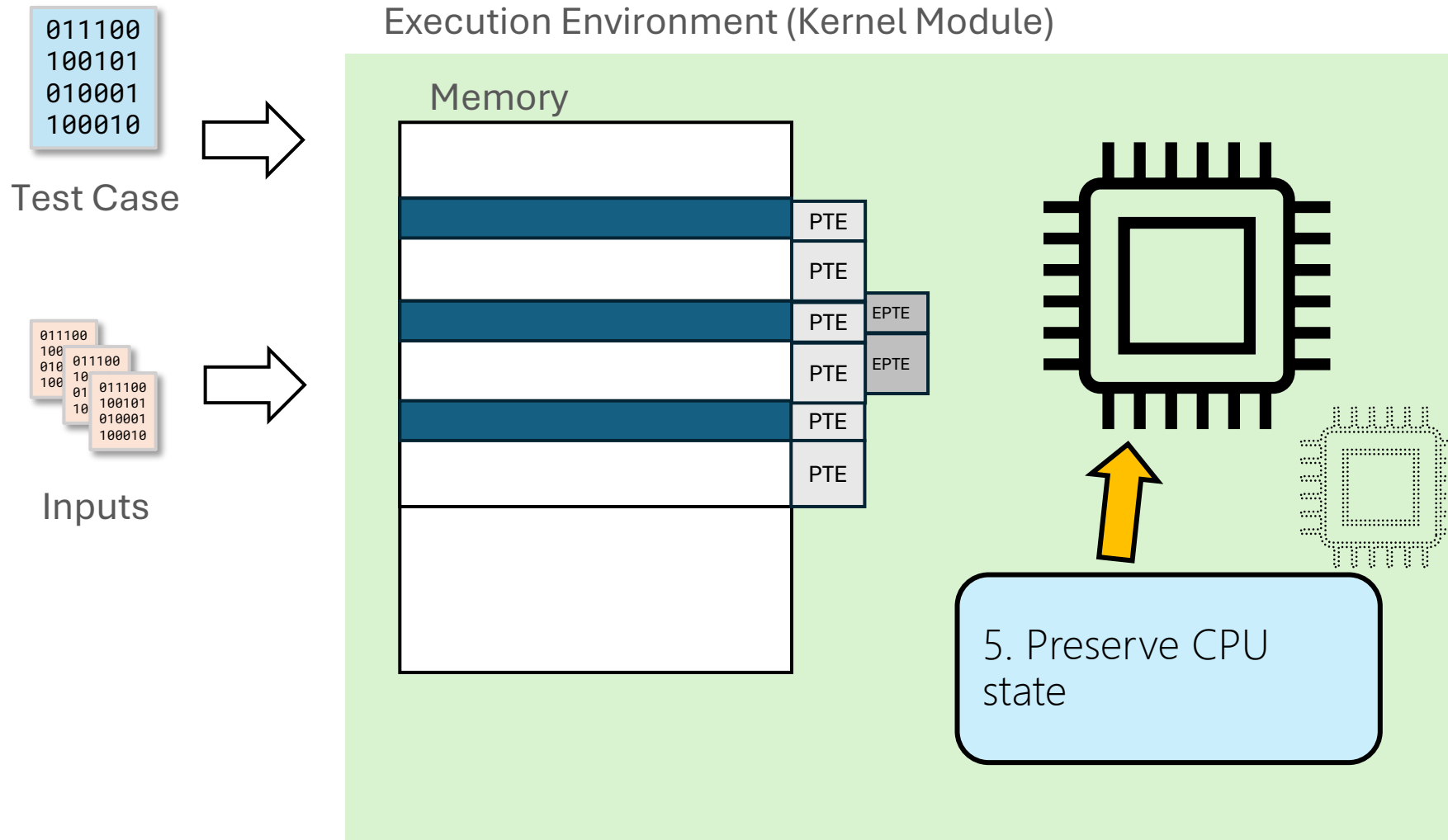
Measurement Process



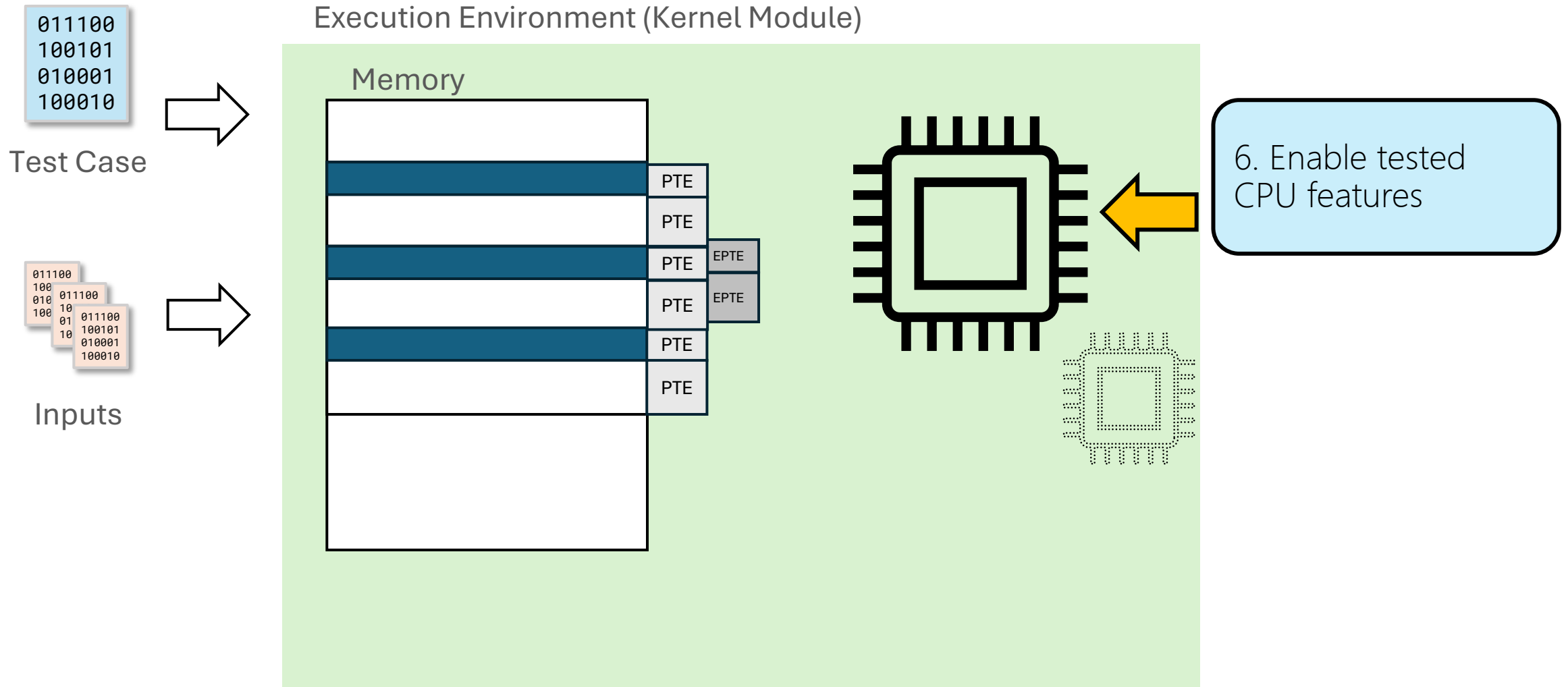
Measurement Process



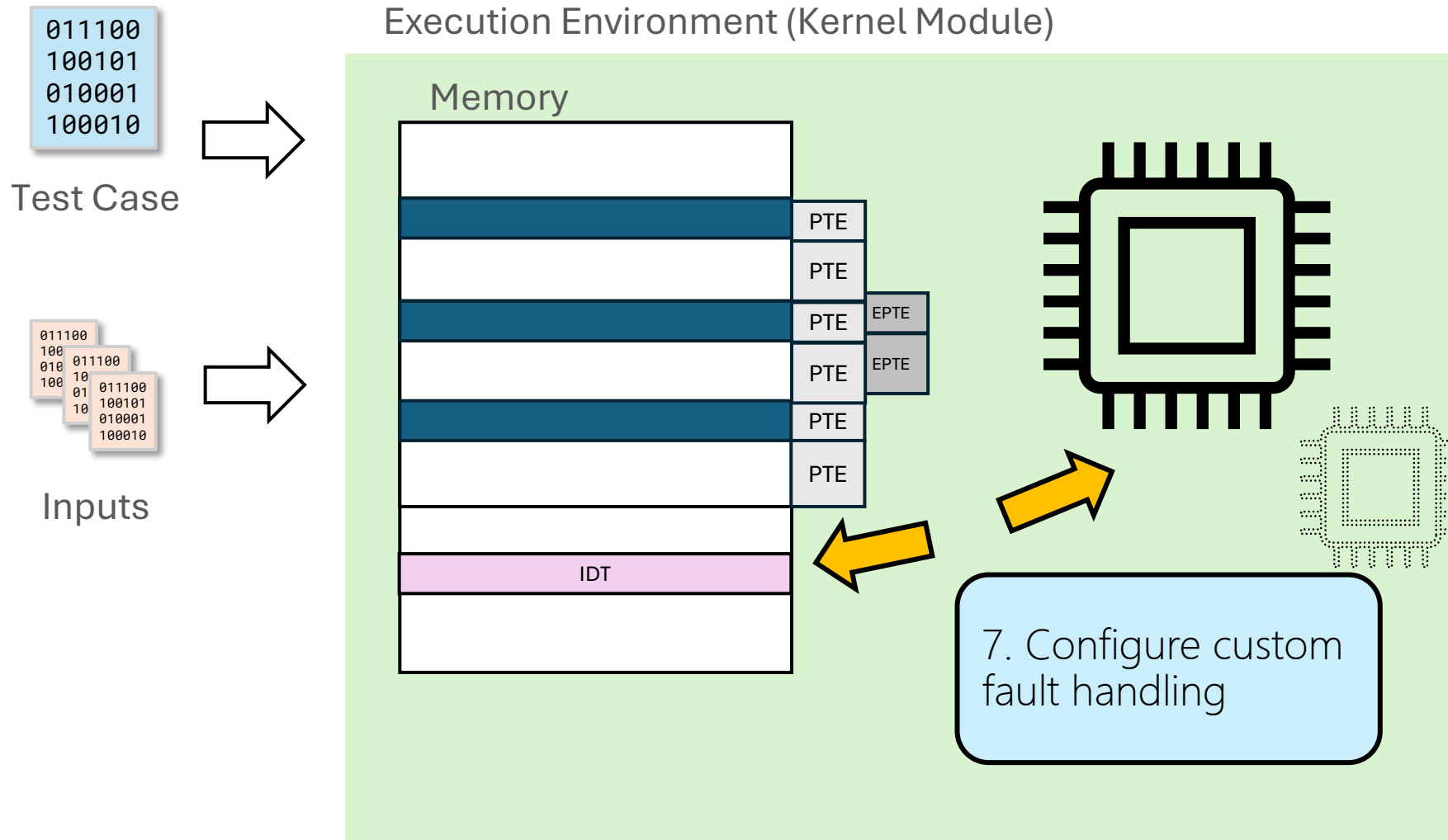
Measurement Process



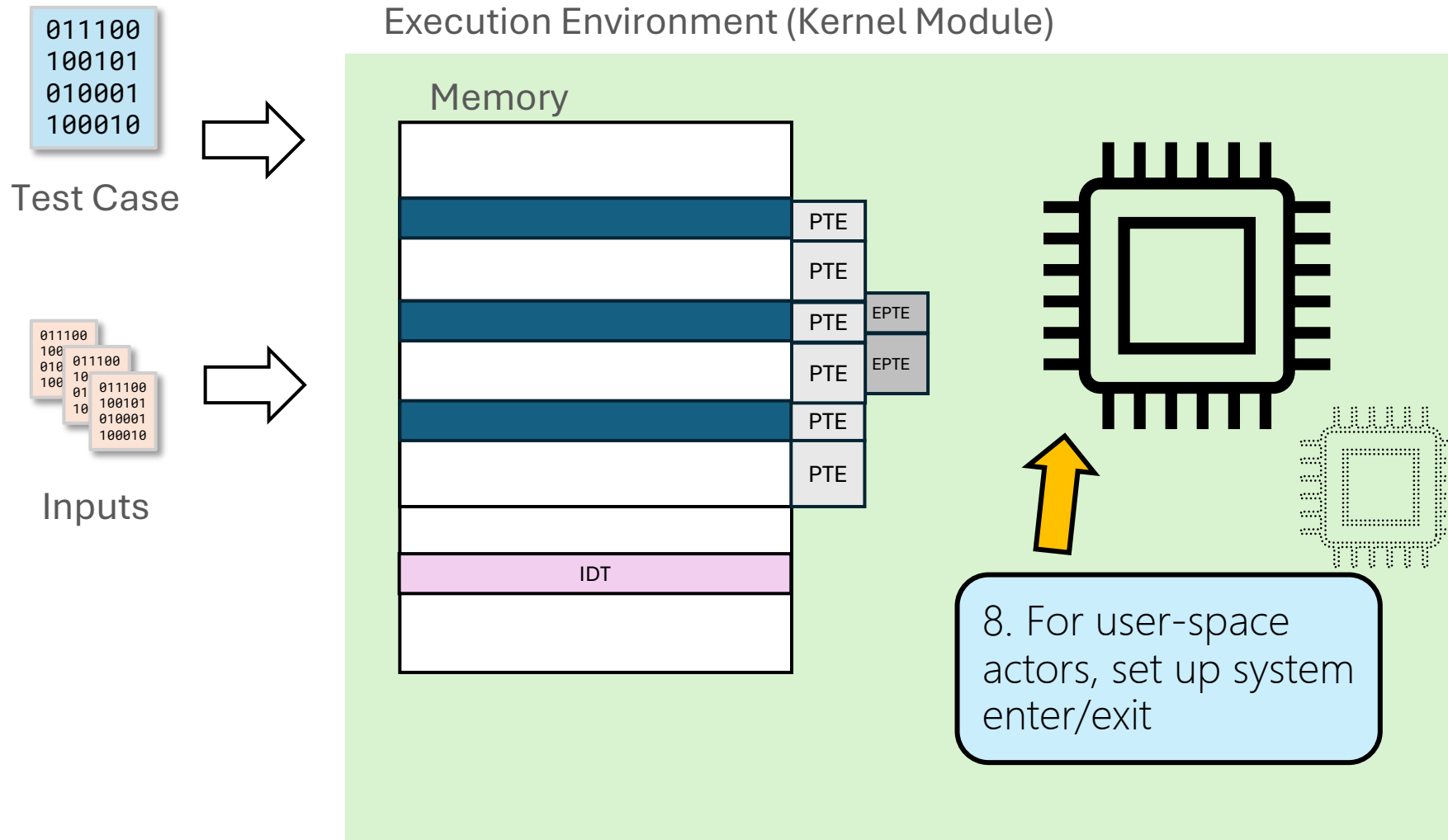
Measurement Process



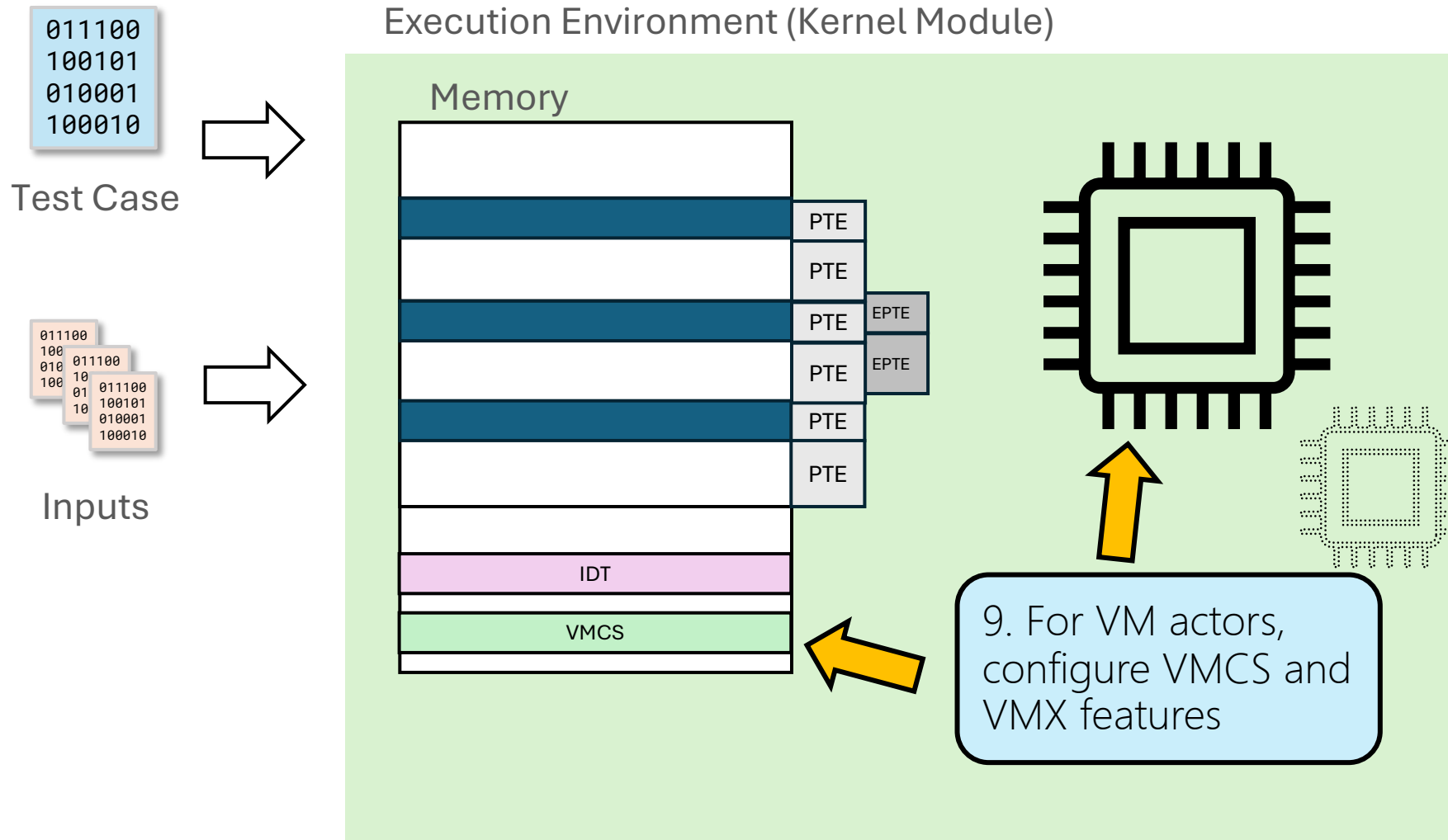
Measurement Process



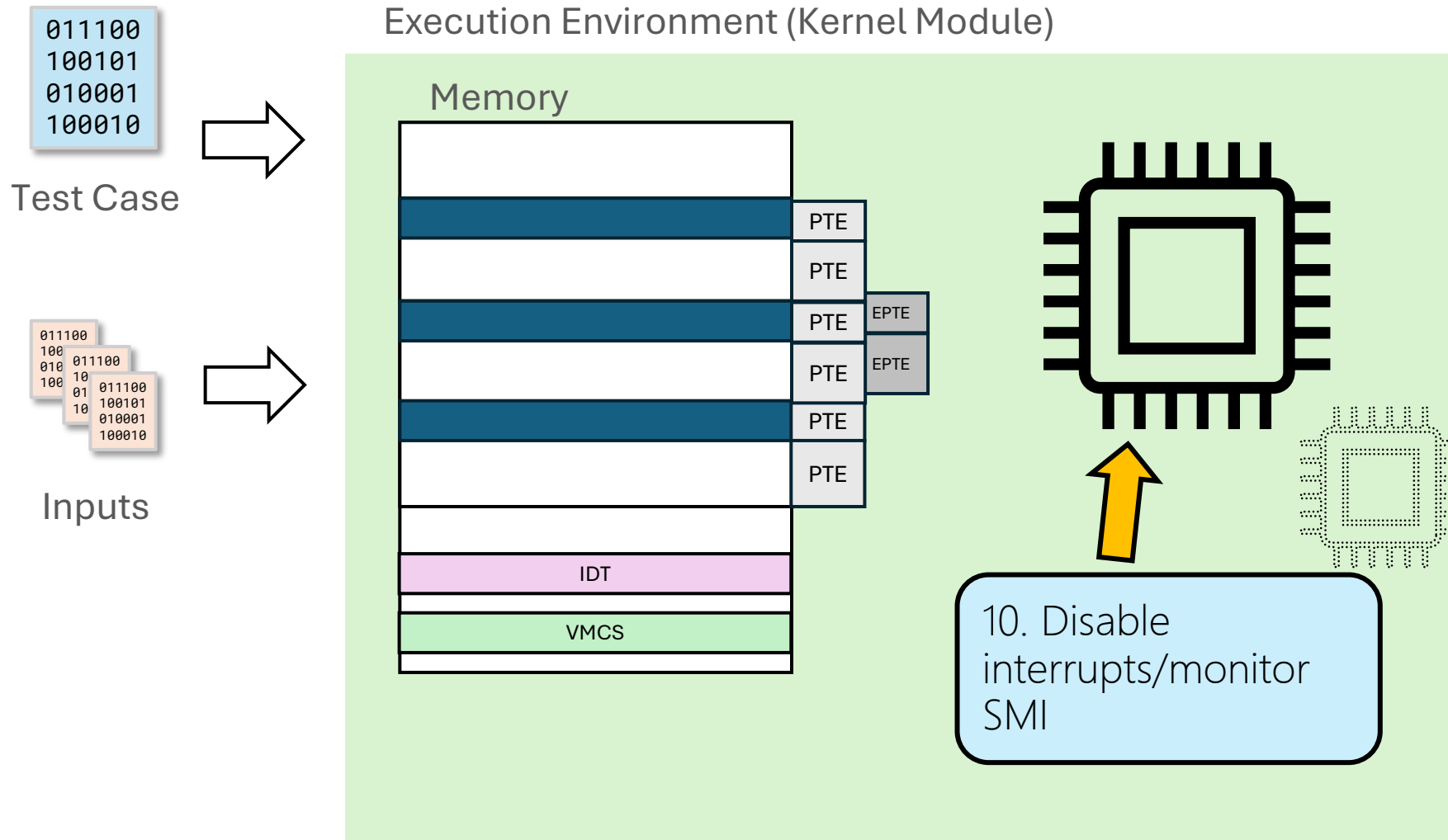
Measurement Process



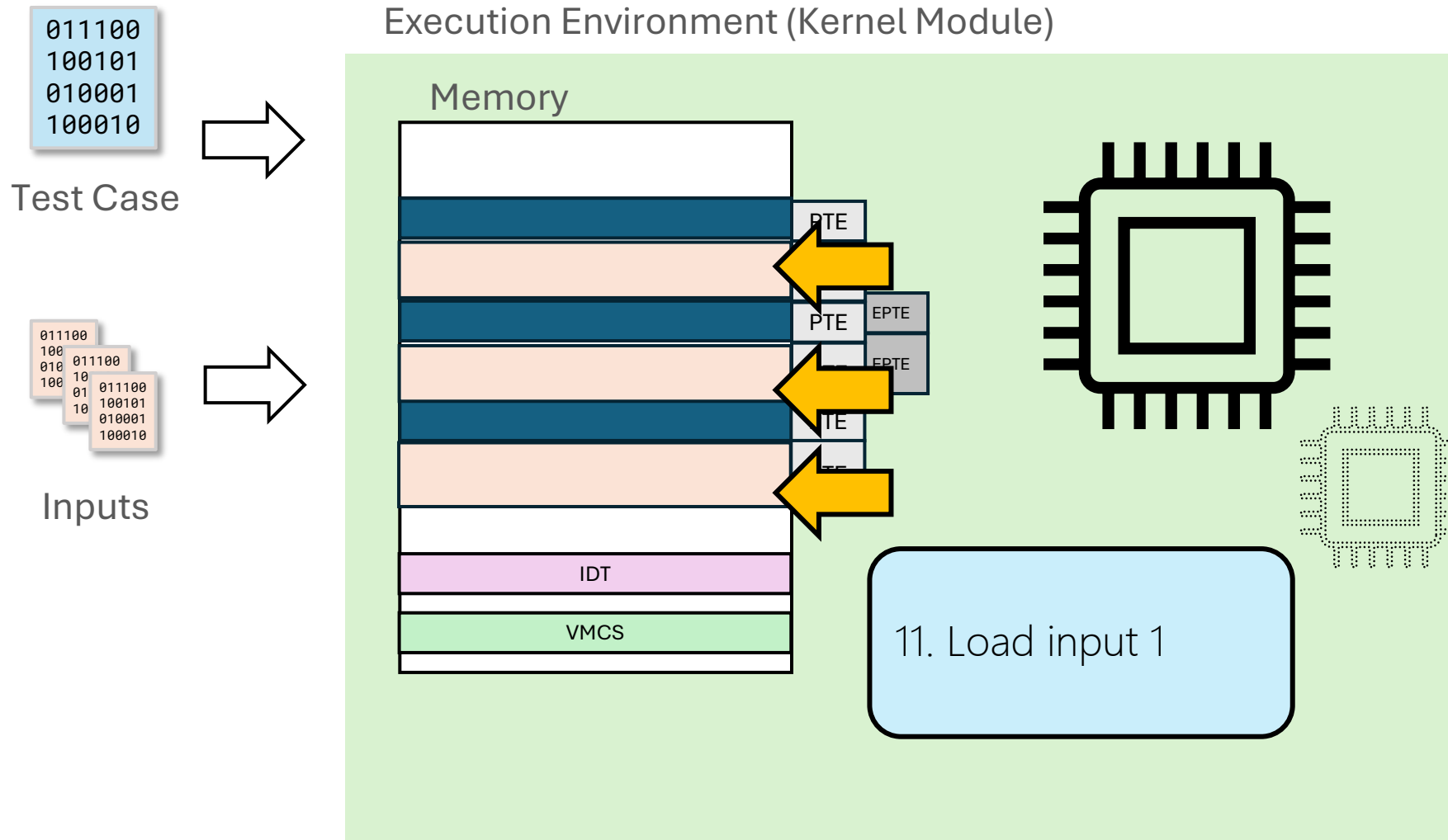
Measurement Process



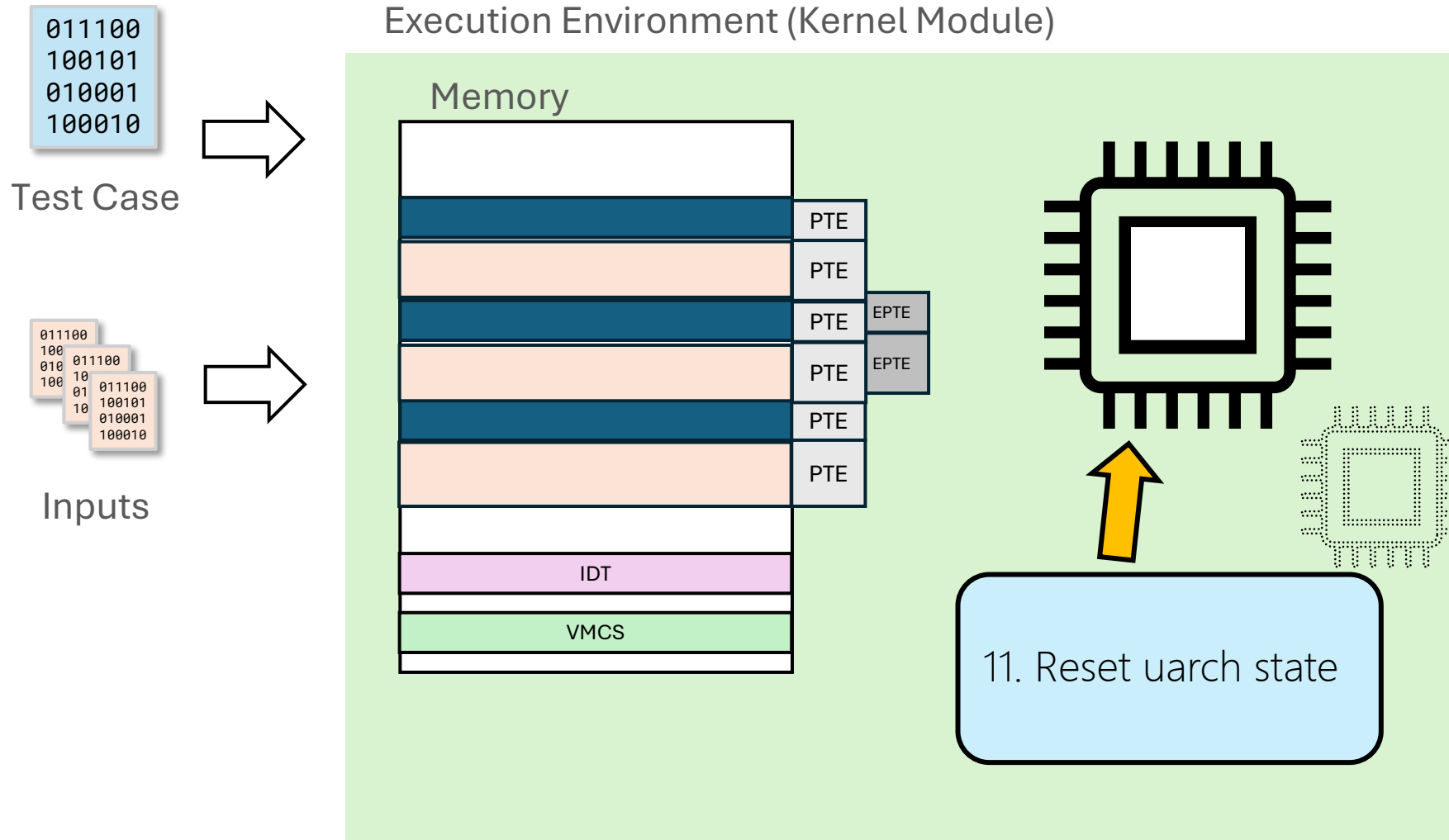
Measurement Process



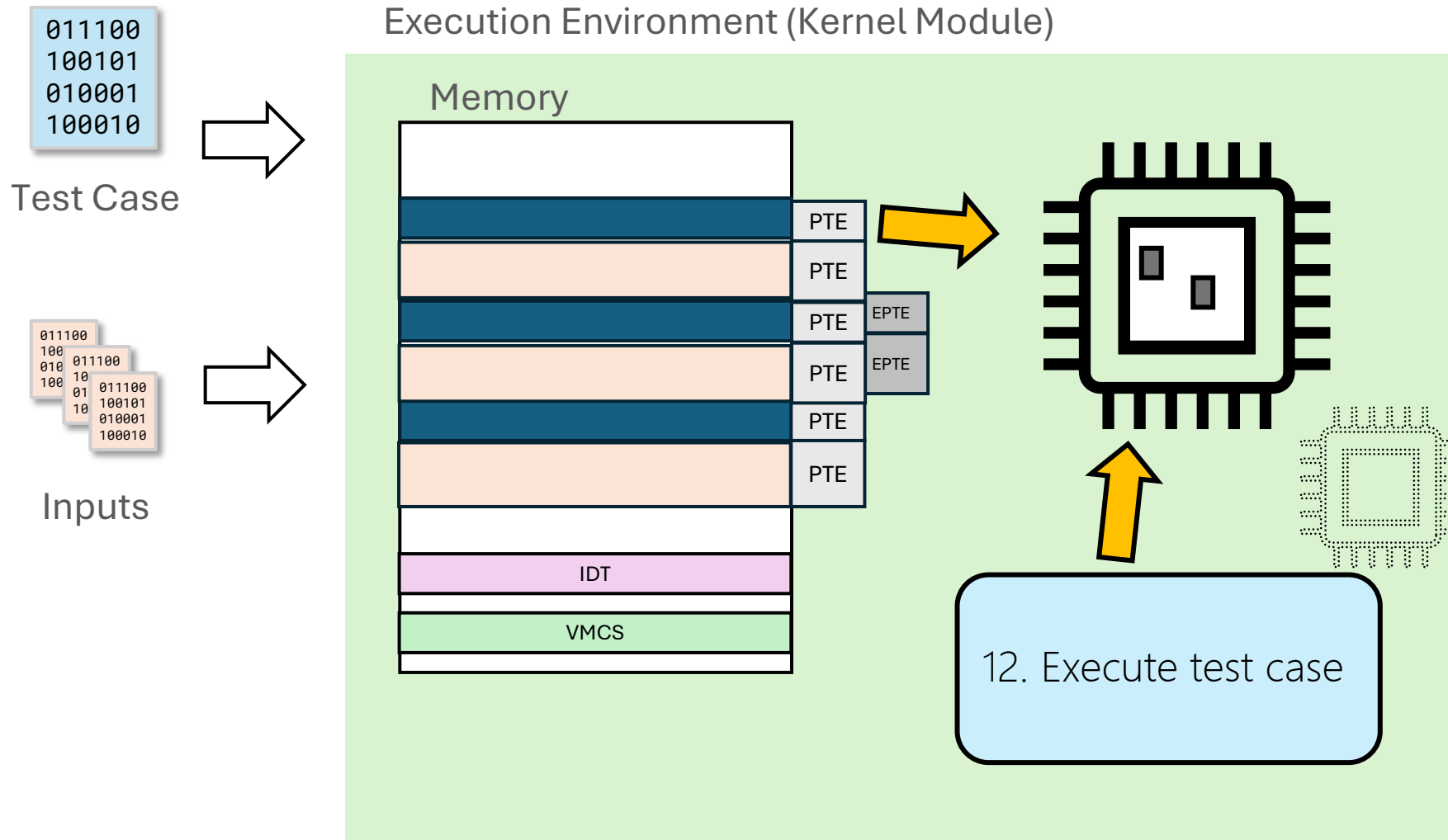
Measurement Process



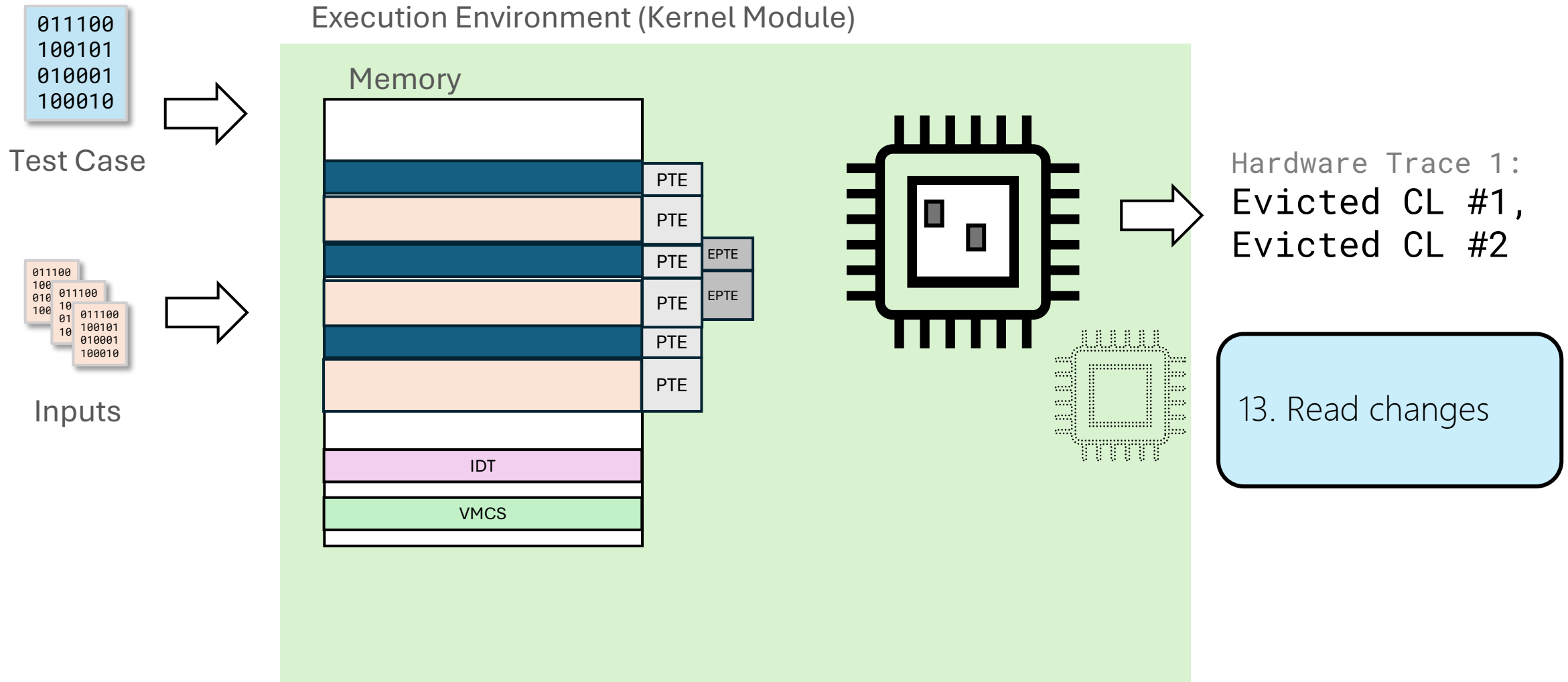
Measurement Process



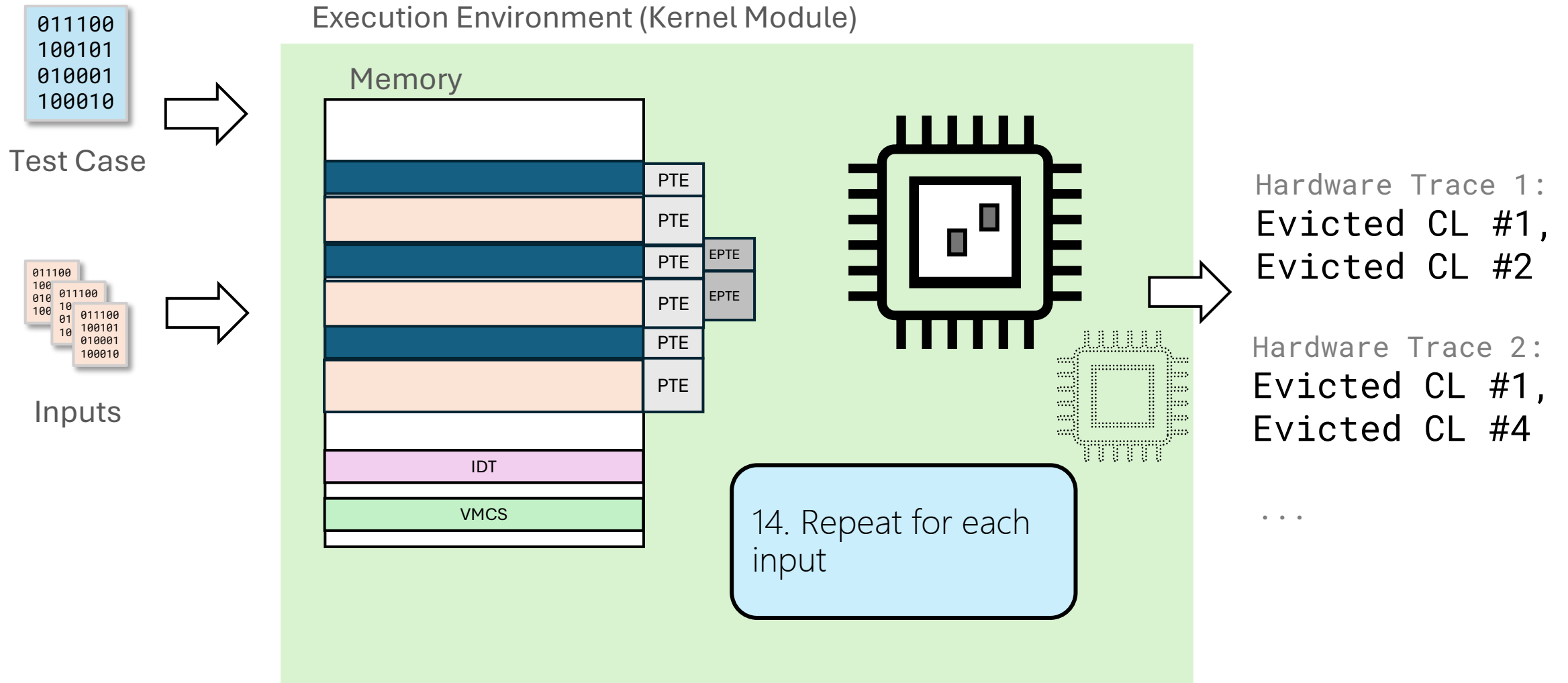
Measurement Process



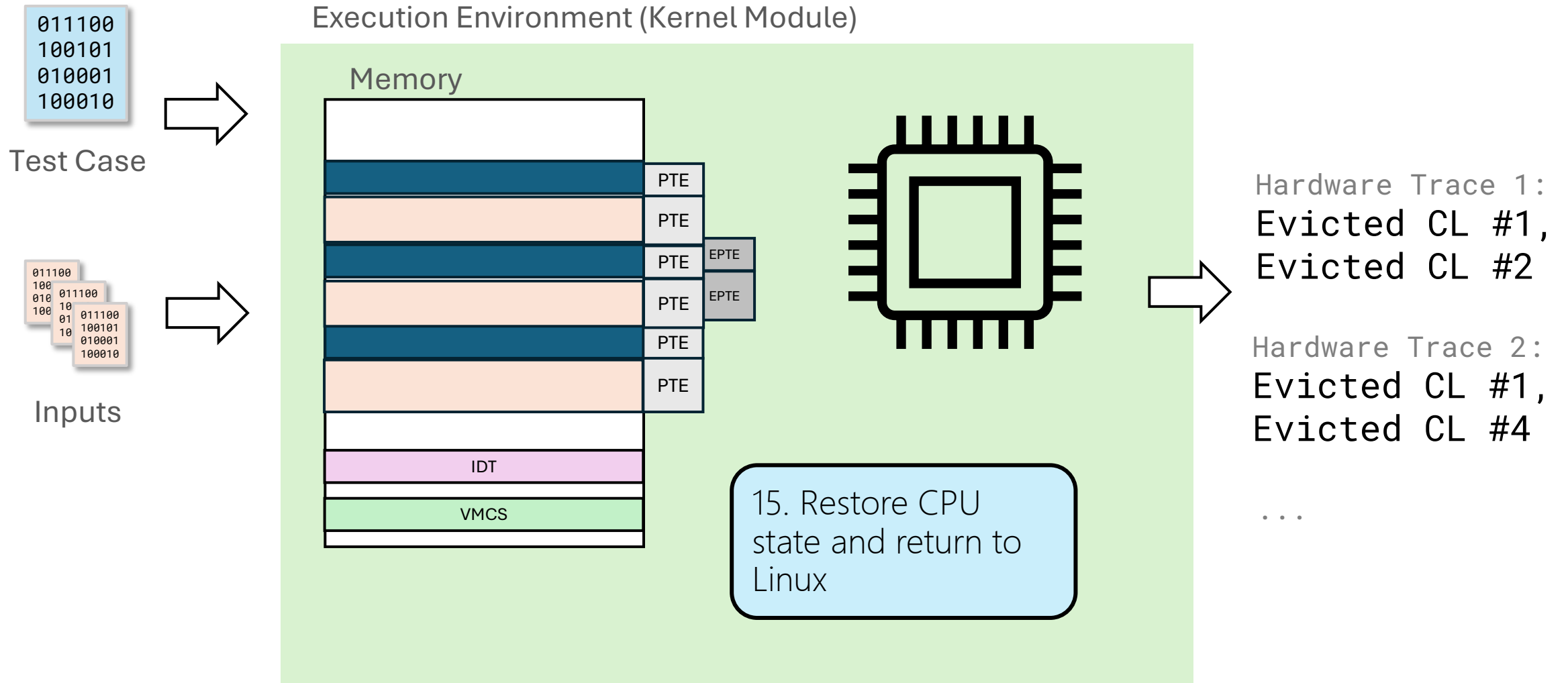
Measurement Process

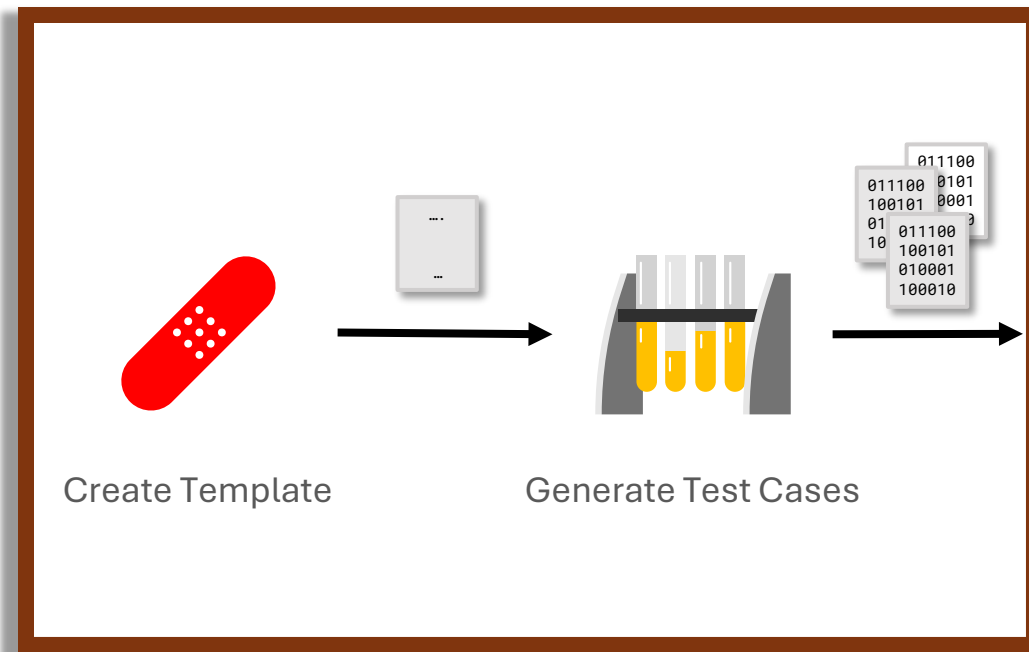


Measurement Process

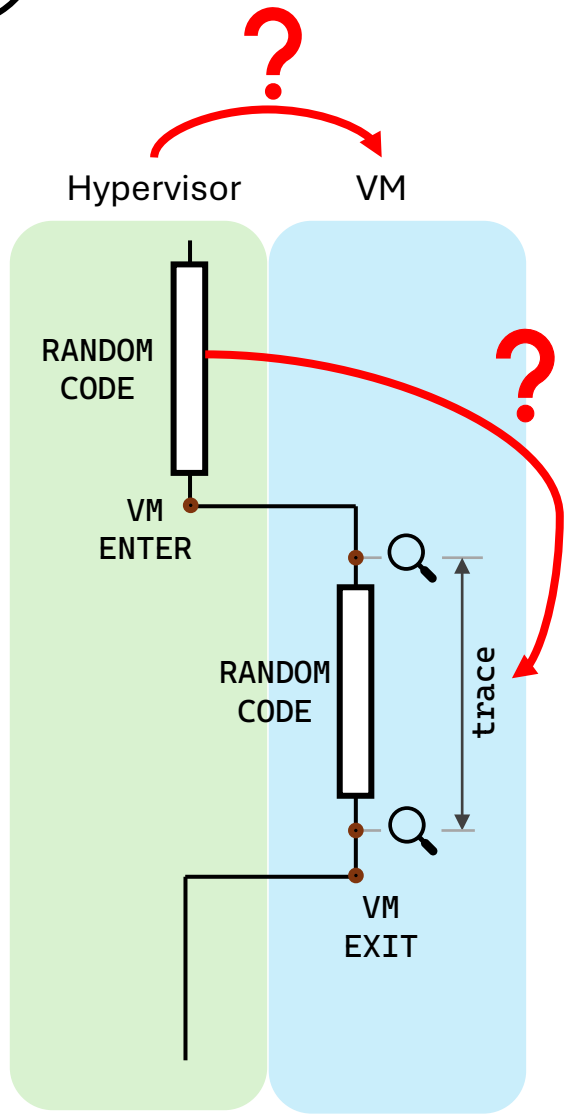
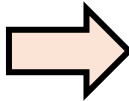
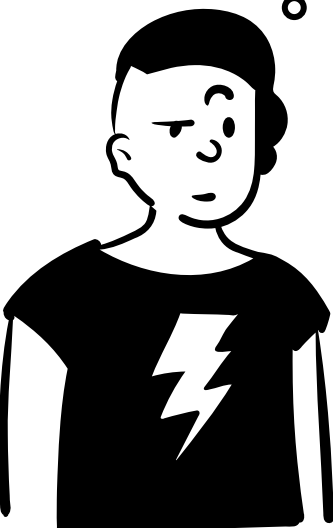


Measurement Process

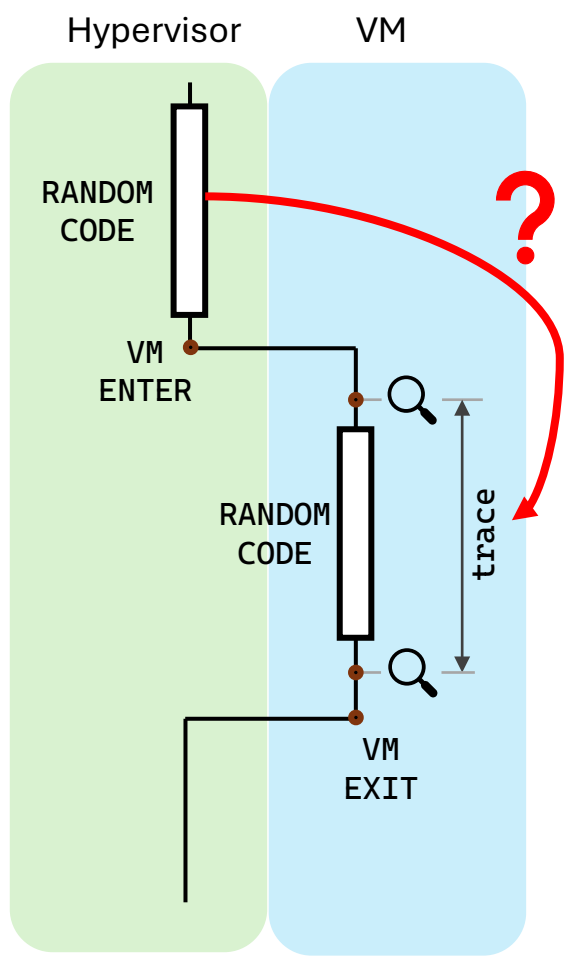
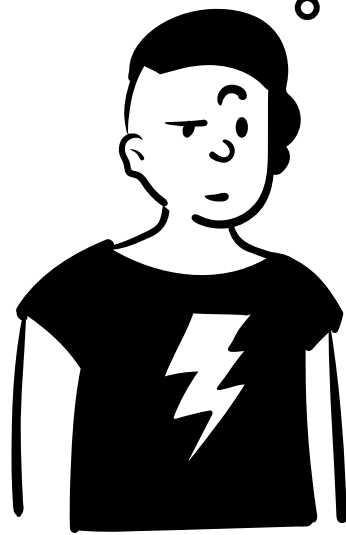


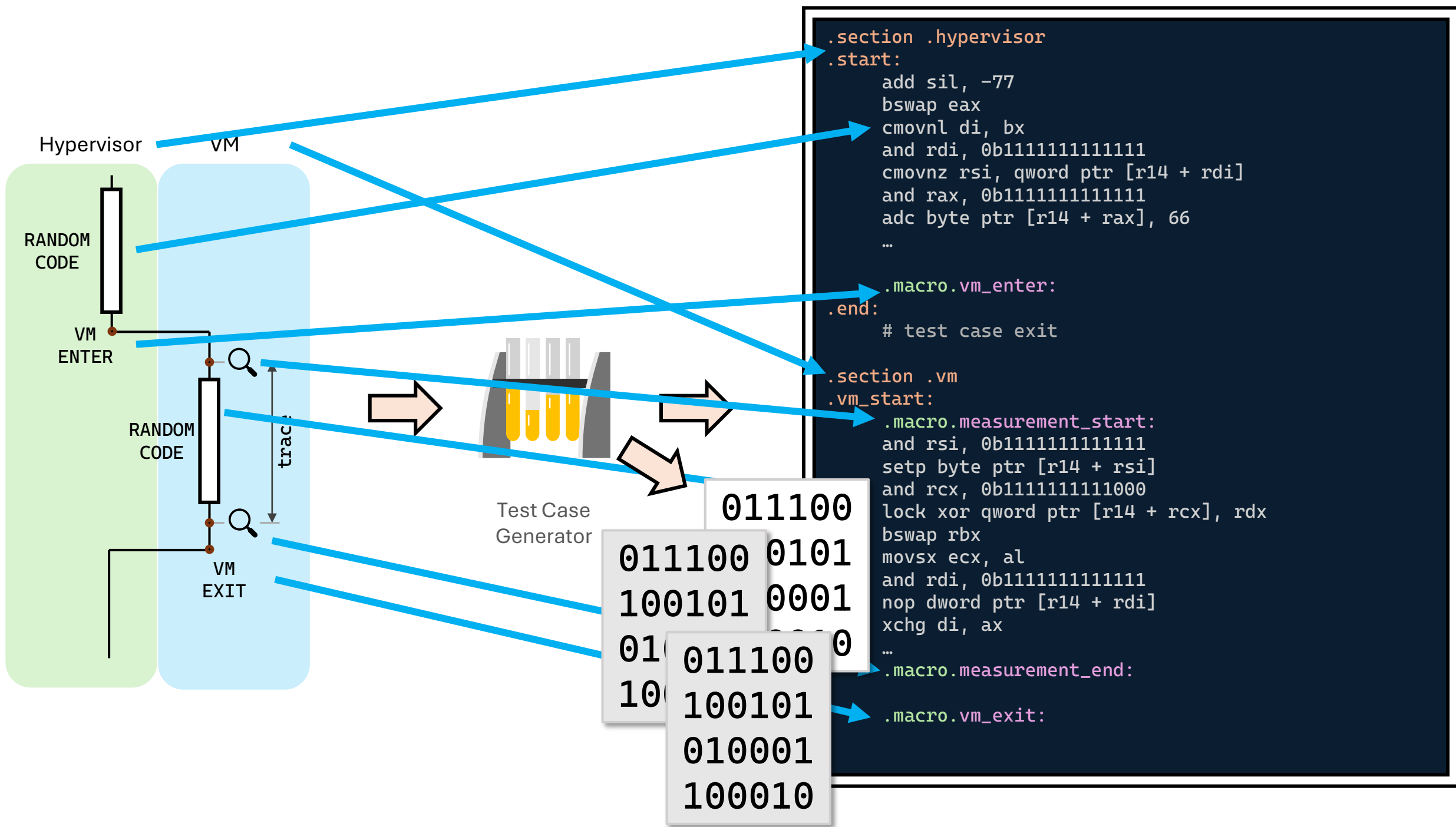


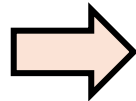
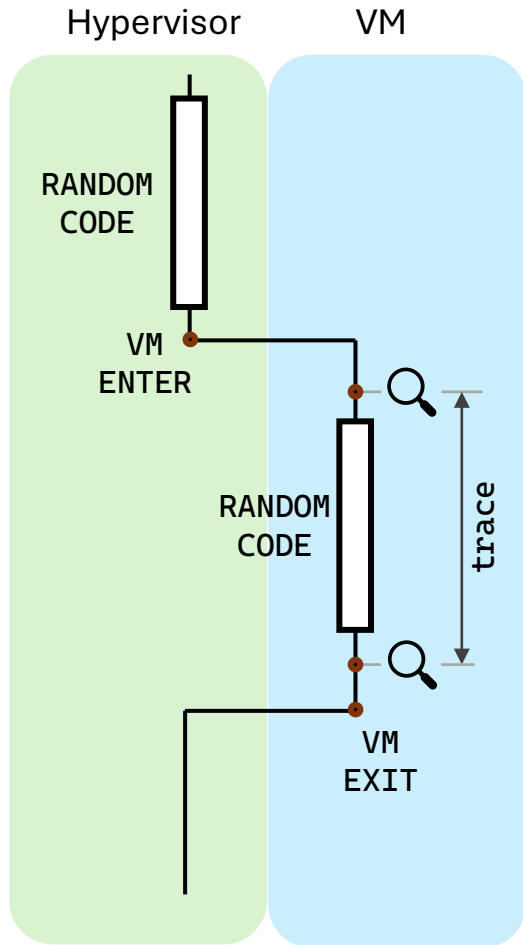
Can a VM leak information from Hypervisor?



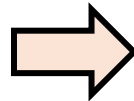
Can a VM leak information from Hypervisor?







Test Case
Generator



011100
011100 0101
100101 0001
010000 0000
100001
011100
100101
010001
100010

```
.section .hypervisor
.start:
    add sil, -77
    bswap eax
    cmovnl di, bx
    and rdi, 0b11111111111111
    cmovnz rsi, qword ptr [r14 + rdi]
    and rax, 0b11111111111111
    adc byte ptr [r14 + rax], 66
    ...

    .macro vm_enter:
.end:
    # test case exit

.section .vm
.vm_start:
    .macro measurement_start:
        and rsi, 0b11111111111111
        setp byte ptr [r14 + rsi]
        and rcx, 0b11111111111000
        lock xor qword ptr [r14 + rcx], rdx
        bswap rbx
        movsx ecx, al
        and rdi, 0b11111111111111
        nop dword ptr [r14 + rdi]
        xchg di, ax
        ...
    .macro measurement_end:

    .macro vm_exit:
```



```

.section .hypervisor
.start:
    add sil, -77
    bswap eax
    cmovnl di, bx
    and rdi, 0b11111111111111
    cmovnz rsi, qword ptr [r14 + rdi]
    and rax, 0b11111111111111
    adc byte ptr [r14 + rax], 66
    ...

    .macro vm_enter:
.end:
    # test case exit

.section .vm
.vm_start:
    .macro measurement_start:
    and rsi, 0b11111111111111
    setp byte ptr [r14 + rsi]
    and rcx, 0b1111111111000
    lock xor qword ptr [r14 + rcx], rdx
    bswap rbx
    movsx ecx, al
    and rdi, 0b11111111111111
    nop dword ptr [r14 + rdi]
    xchg di, ax
    ...
    .macro measurement_end:
    .macro vm_exit:

```

011100

011100 0101

100101 0001

010001

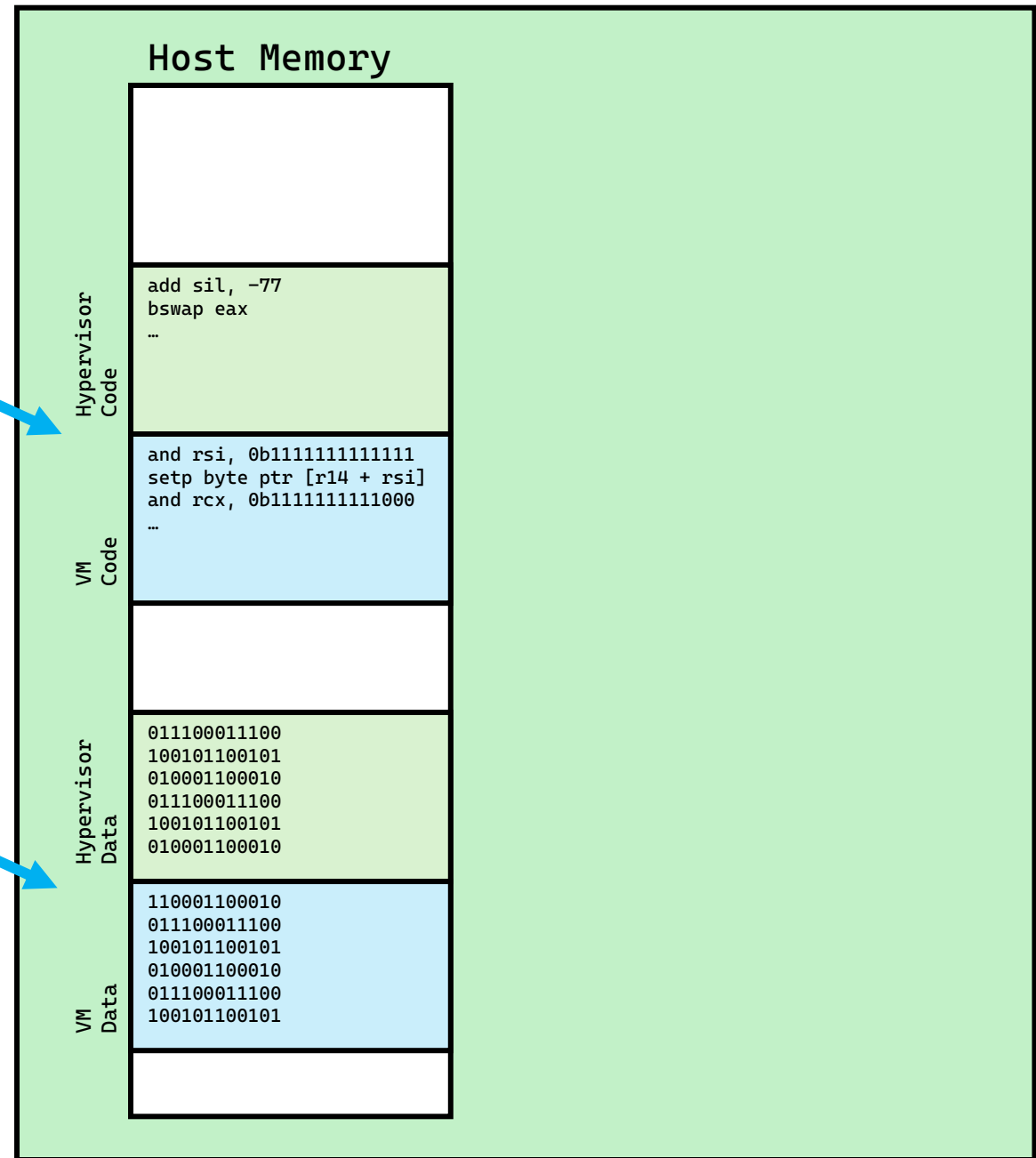
011100

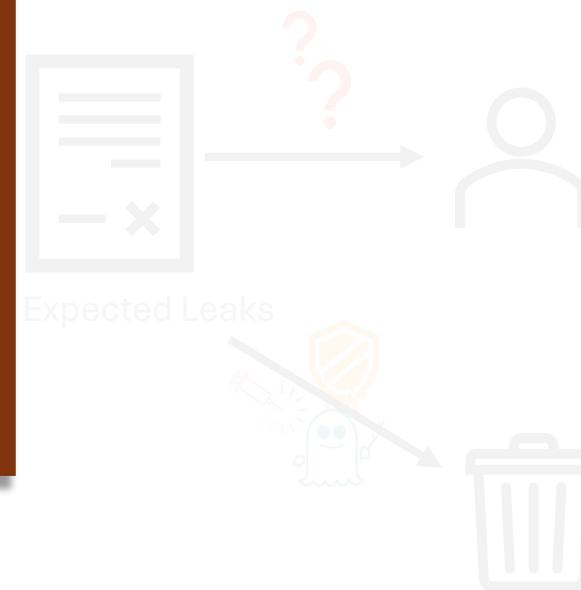
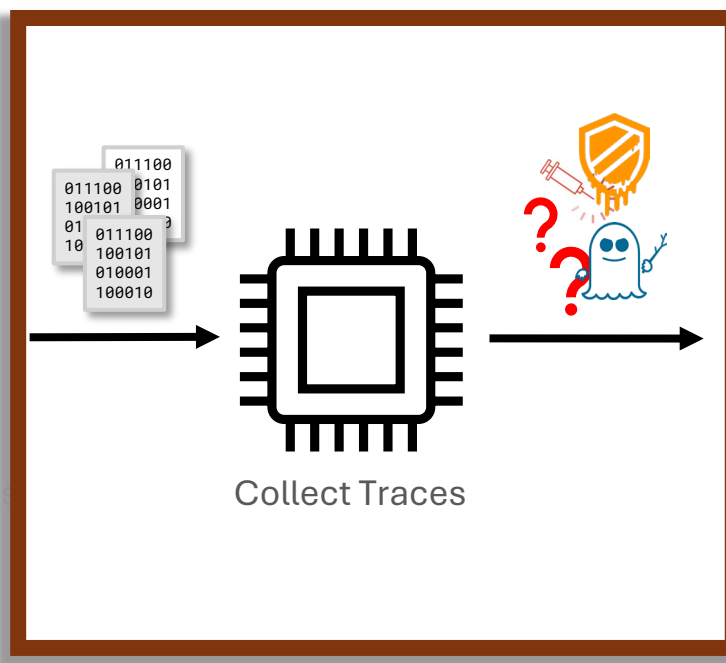
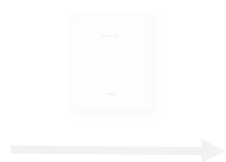
100101

010001

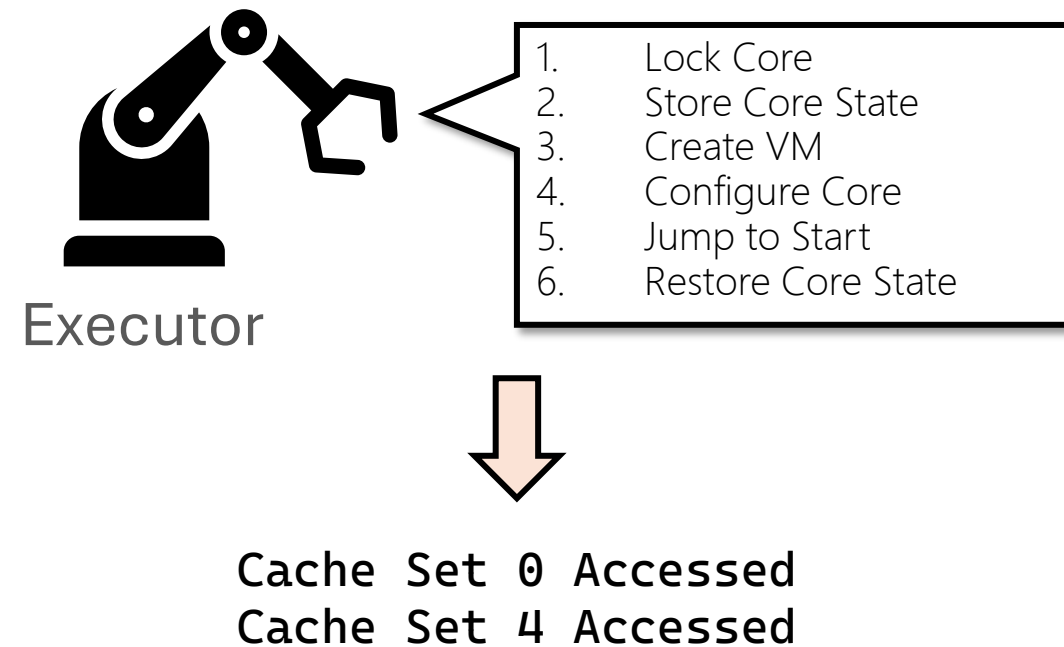
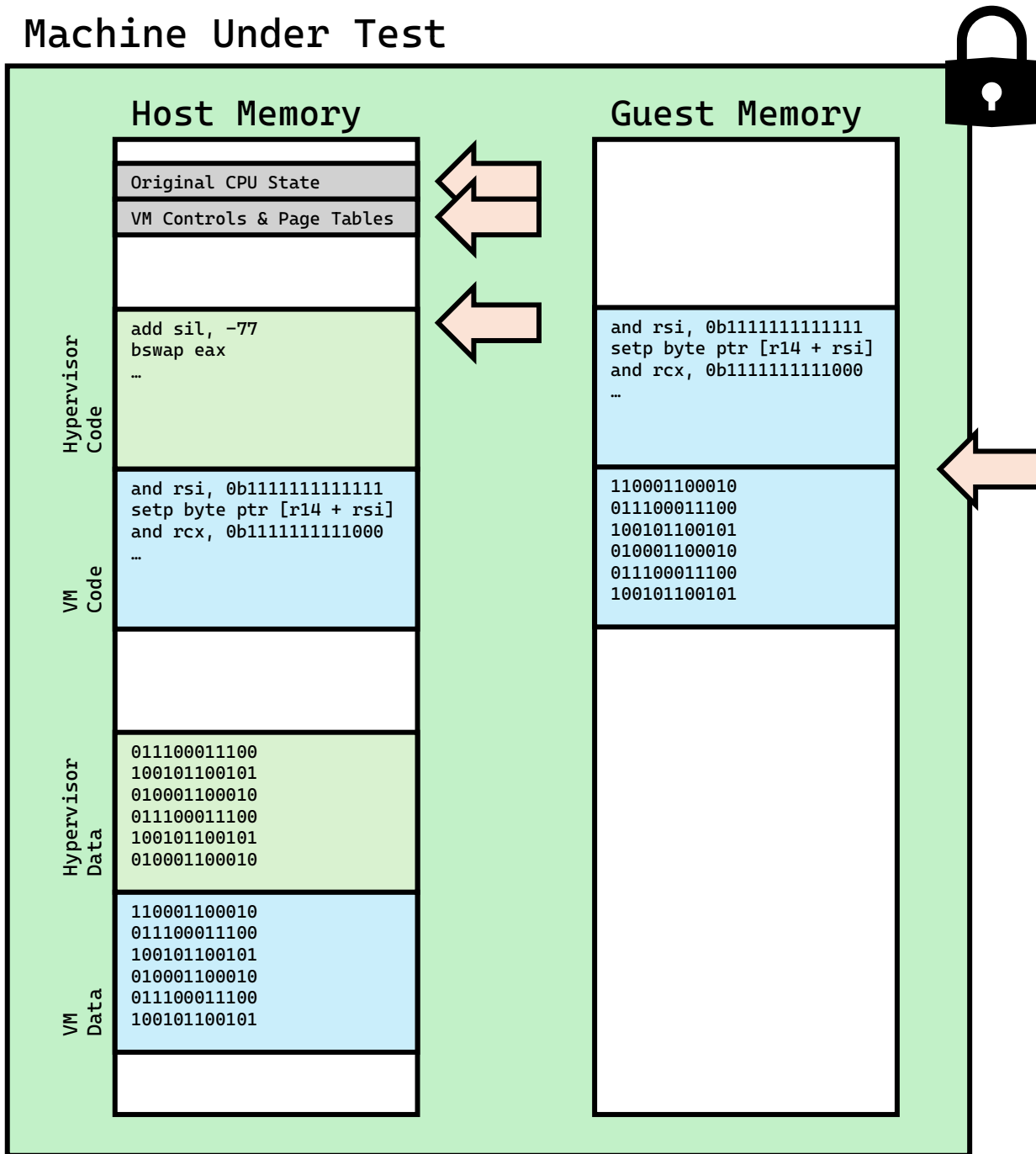
100010

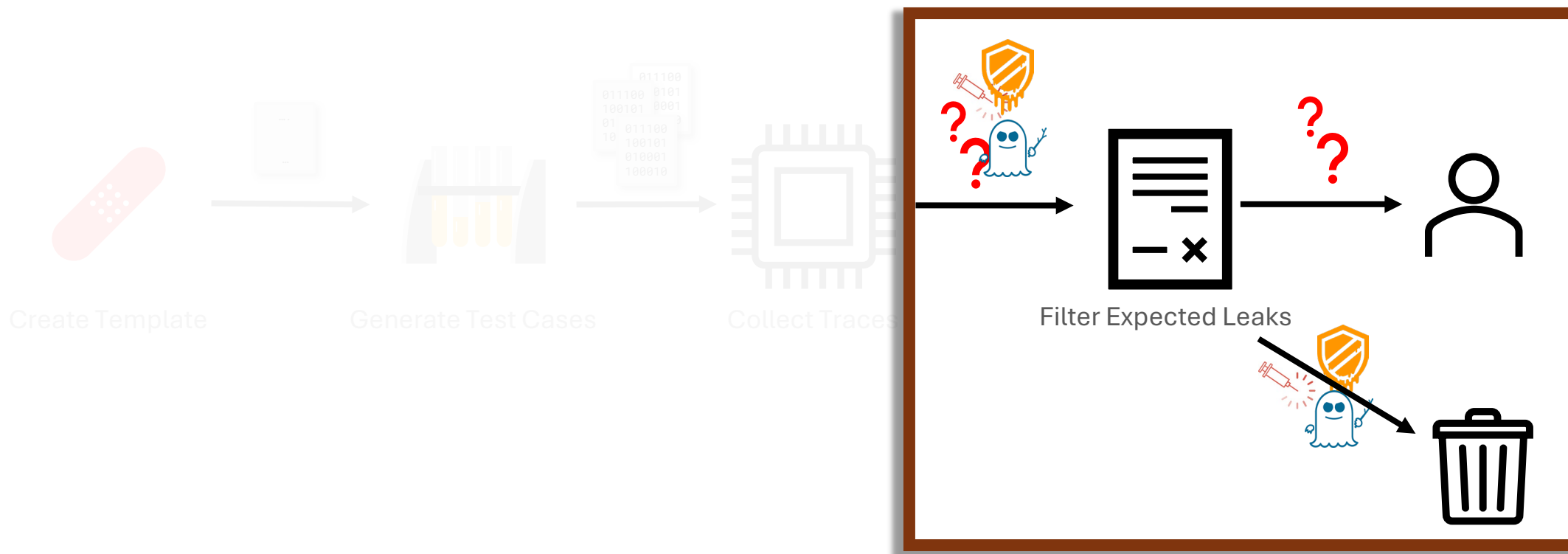
Machine Under Test





Machine Under Test





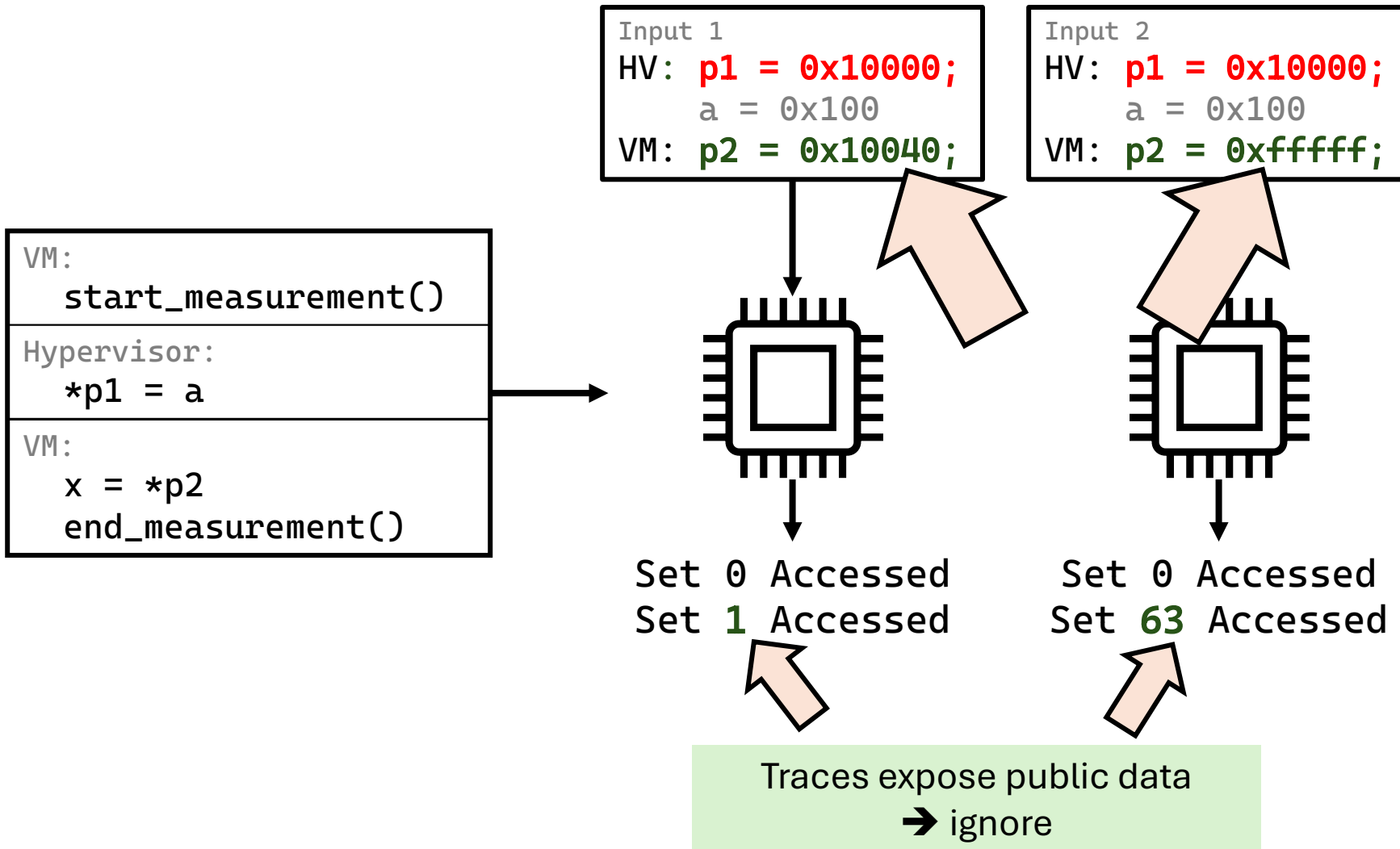
Non-interference

Attacker's data is public; victim's data is private

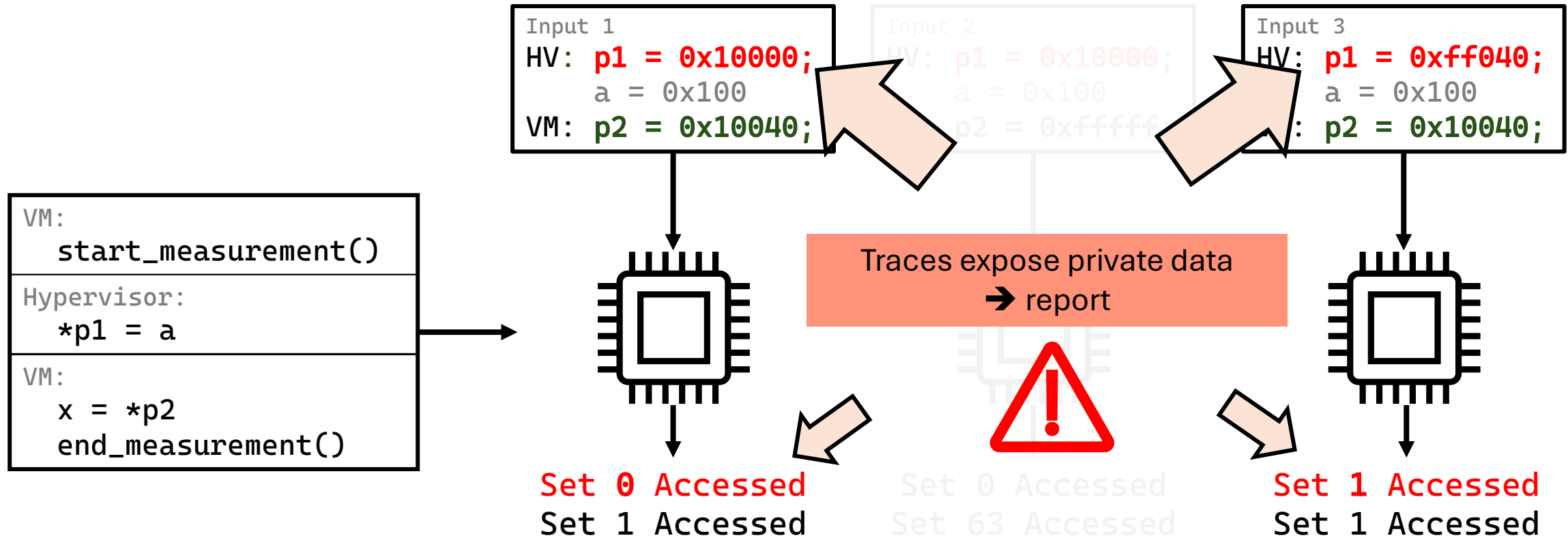
VM: start_measurement()
Hypervisor: *p1 = a
VM: x = *p2 end_measurement()

Non-interference

Attacker's data is public; victim's data is private



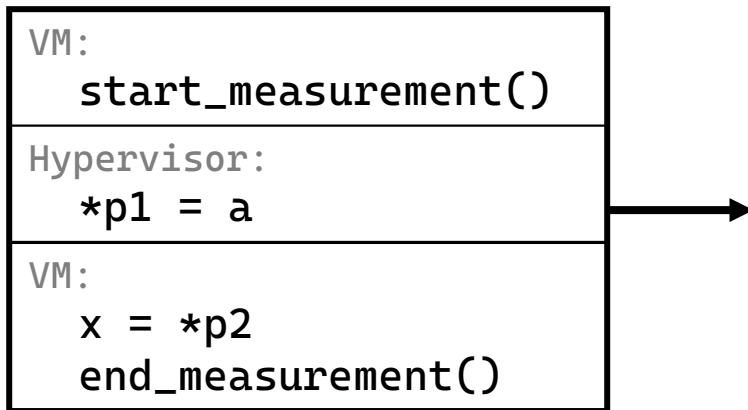
Non-interference



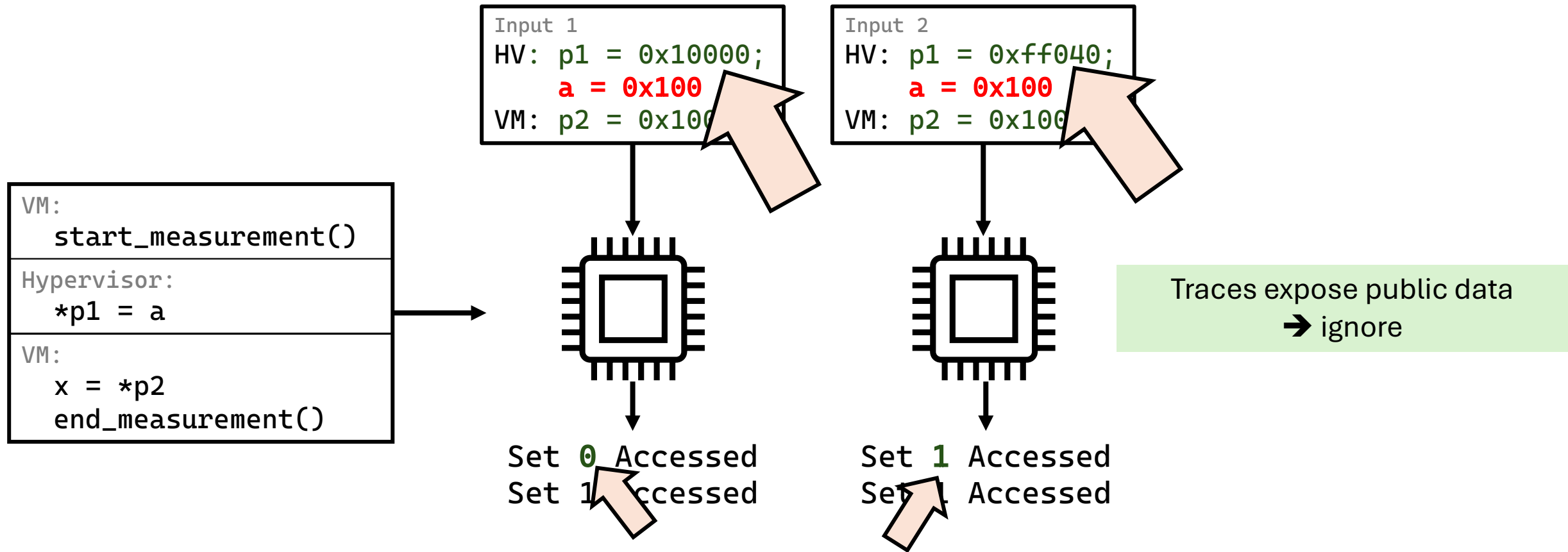
Contract-based Non-interference

Attacker's data is public + victim's addresses are public;

The rest of victim's data is private



Contract-based Non-interference



Contract-based Non-interference

