



**AUGUST 7-8, 2024**  
BRIEFINGS

# Compromising Confidential Compute

One bug at a time



- Max
- Microsoft Offensive Research & Security Engineering – MORSE Team



# **black hat**<sup>®</sup> USA 2024

- **Security review of Intel TDX**
- Partnership between Microsoft and Intel
- 4-month teamwork





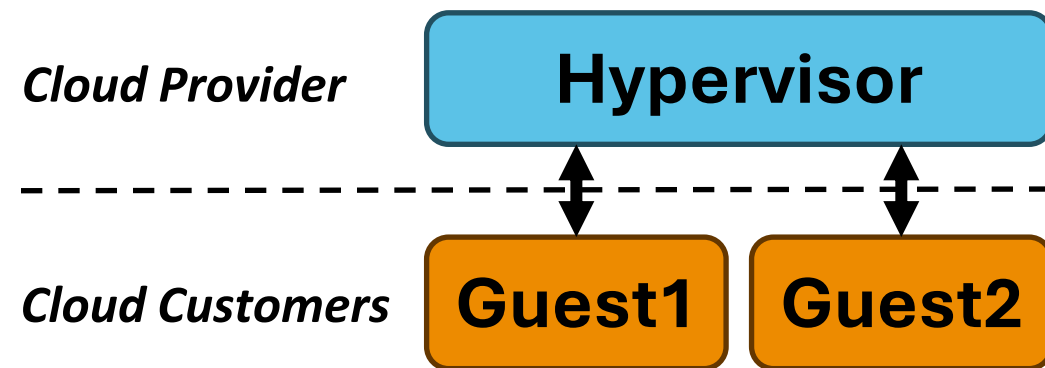


- 1. The TDX Module: technical overview**
- 2. Research approach and first findings**
- 3. Vulnerability 1**
- 4. Vulnerability 2**



# A change in virtualization architecture

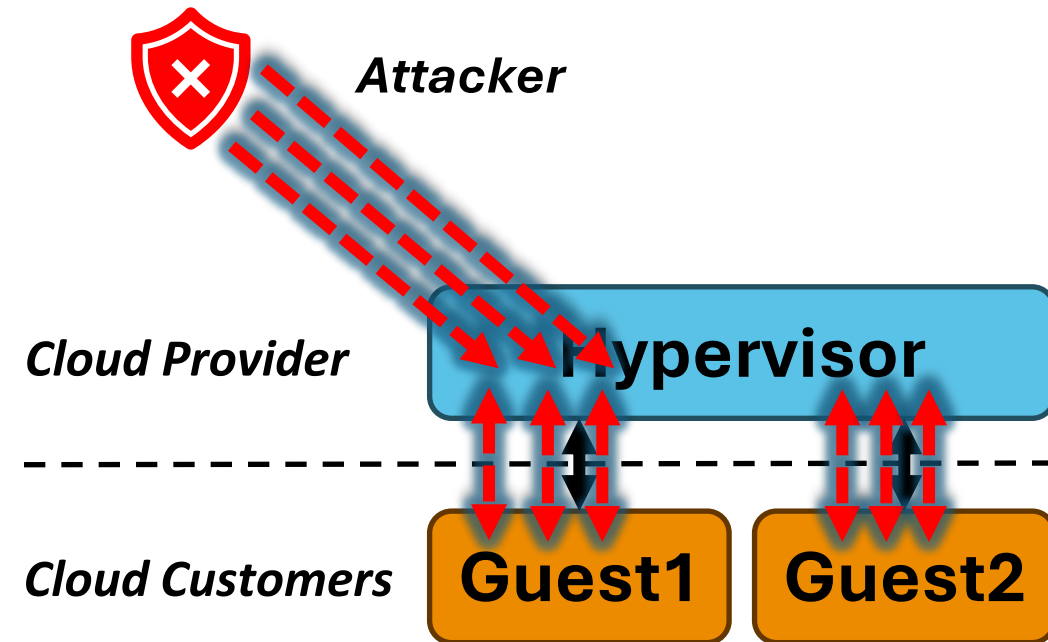
## Standard Architecture



- The guests' memory and registers are visible to the hypervisor

# A change in virtualization architecture

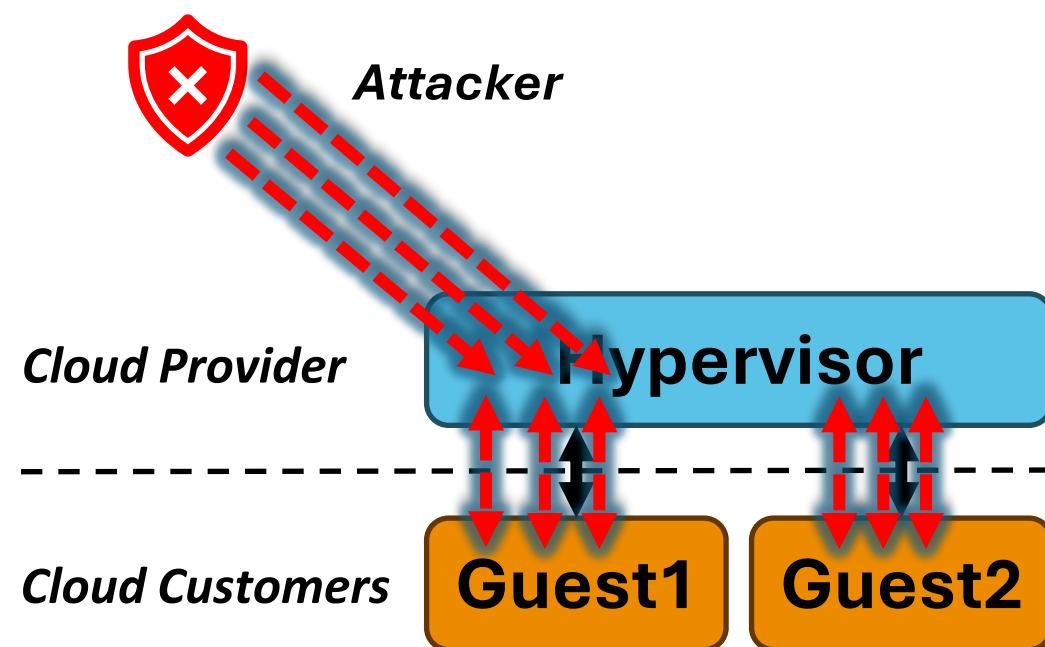
## Standard Architecture



- The guests' memory and registers are visible to the hypervisor

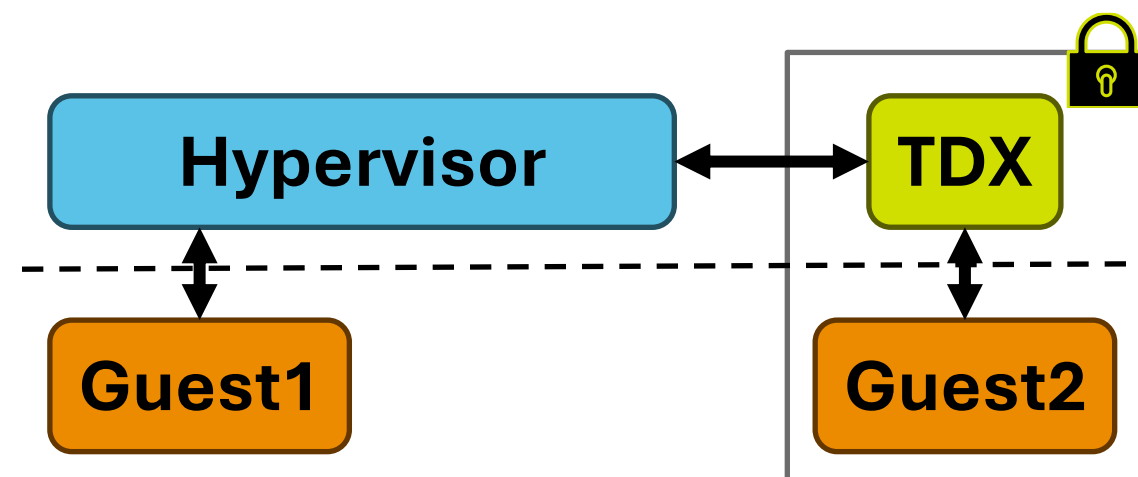
# A change in virtualization architecture

Standard Architecture



- The guests' memory and registers are visible to the hypervisor

TDX Architecture

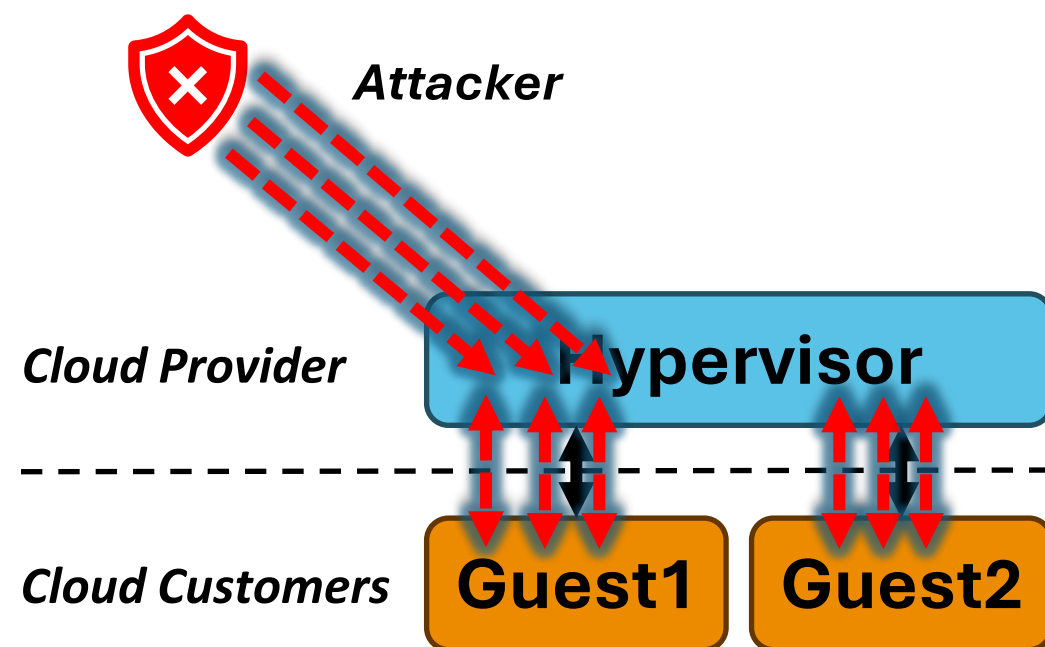


- TDX Module: firmware, *gatekeeper*
- Memory is encrypted, registers are hidden



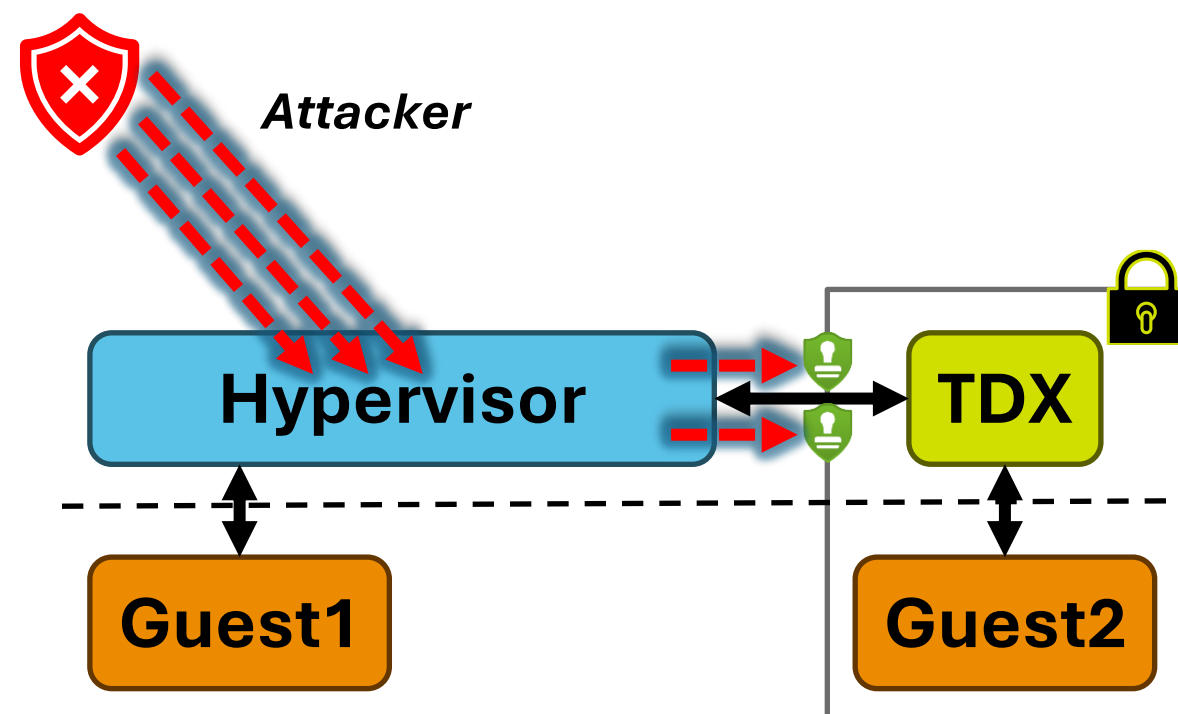
# A change in virtualization architecture

Standard Architecture



- The guests' memory and registers are visible to the hypervisor

TDX Architecture



- TDX Module: firmware, *gatekeeper*
- Memory is encrypted, registers are hidden



# The TDX Module

- Provides **confidentiality** and **integrity** guarantees to guests
- Available in future generation CPUs
- We're very interested in Confidential Computing in Azure
- **Our goal:** verify the security of the TDX module



**1. The TDX Module: technical overview**

2. Research approach and first findings

3. Vulnerability 1

4. Vulnerability 2



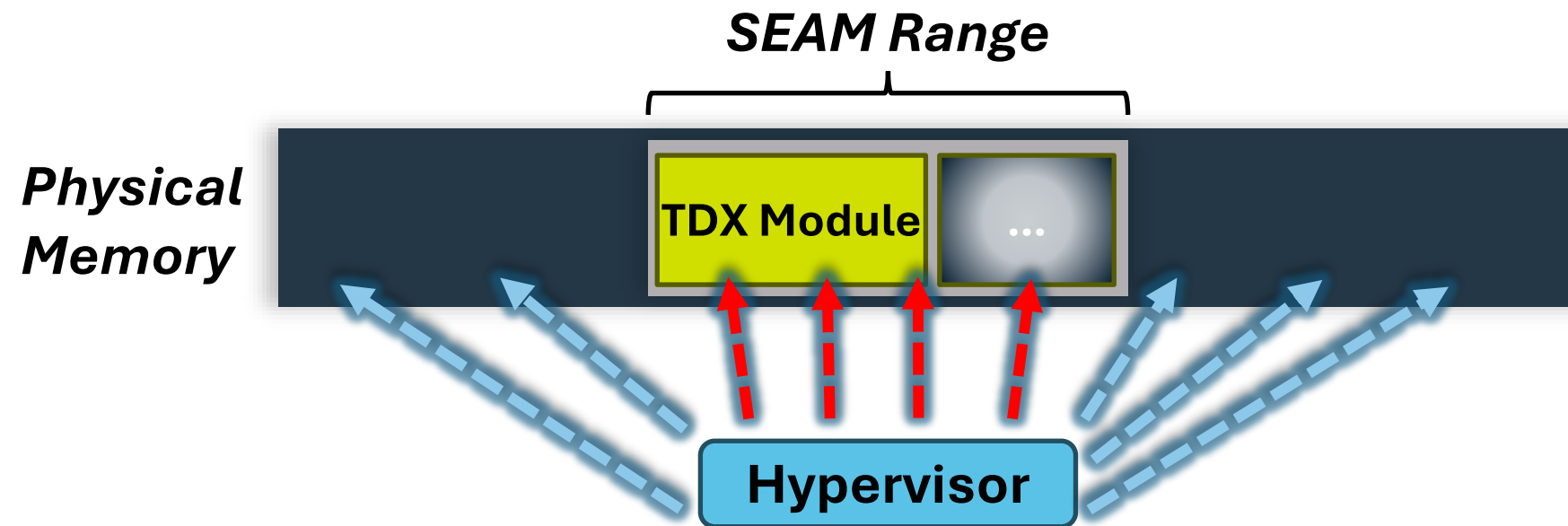
# Generalities


- Software that runs on the main CPU, **not** on a separate chip
- Open-Source, MIT license: <https://github.com/intel/tdx-module>
- Programmed in C, compiled by Clang, ELF binary
- Uses the standard x64 ISA, and runs in ring0 64bit paged mode



# Initialization time

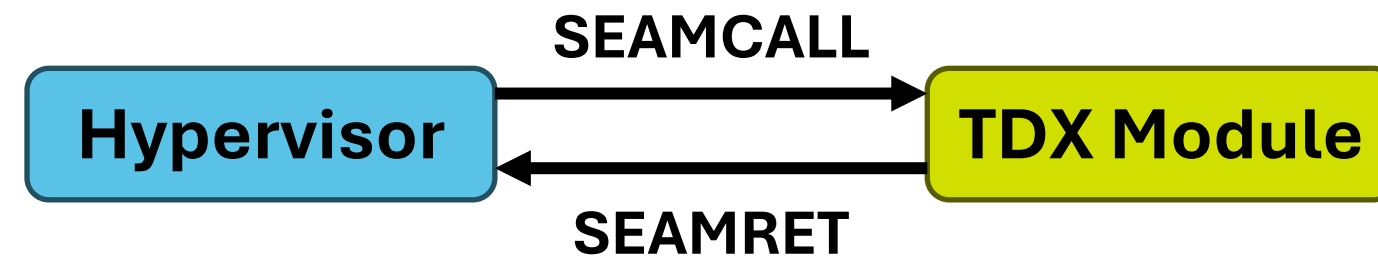
- The TDX Module is loaded in a protected range of physical memory called the **SEAM Range**



- The hypervisor cannot access the SEAM Range 

# Run time

- The TDX Module executes only when **explicitly invoked**
- Two new CPU instructions: **SEAMCALL** and **SEAMRET**
- The TDX Module is invoked via **SEAMCALL**, and returns via **SEAMRET**



# Privileges

- “TDX Mode”, has access to the SEAM Range

**TDX Mode**

-----

**Host Mode**

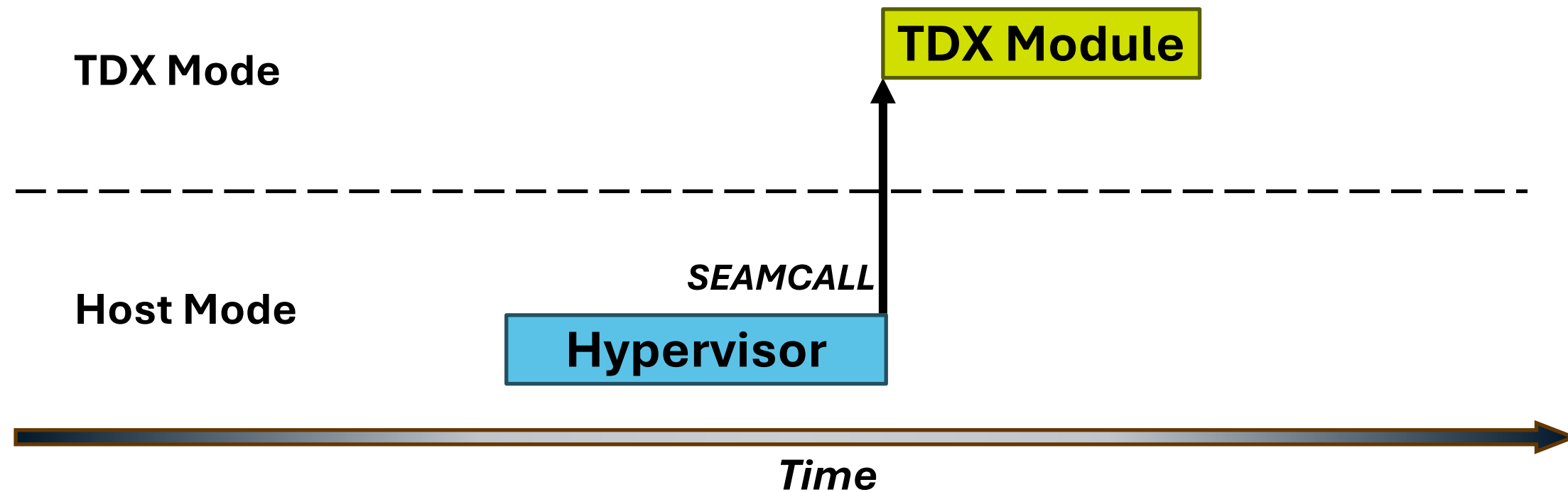
**Hypervisor**

*Time*



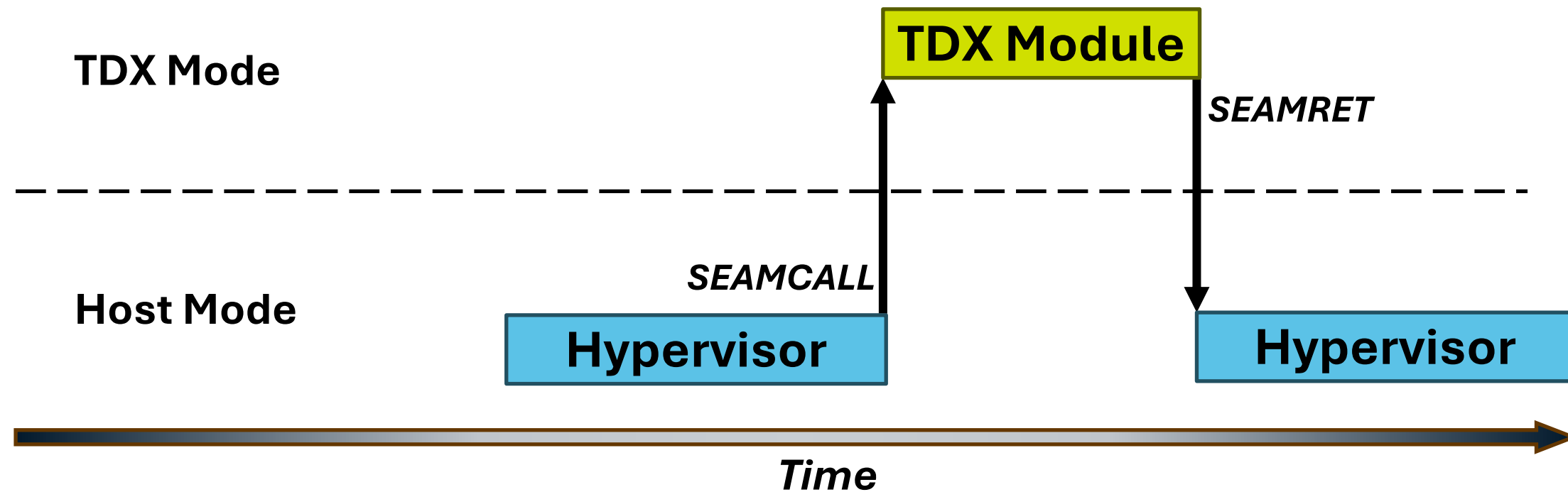
# Privileges

- “TDX Mode”, has access to the SEAM Range

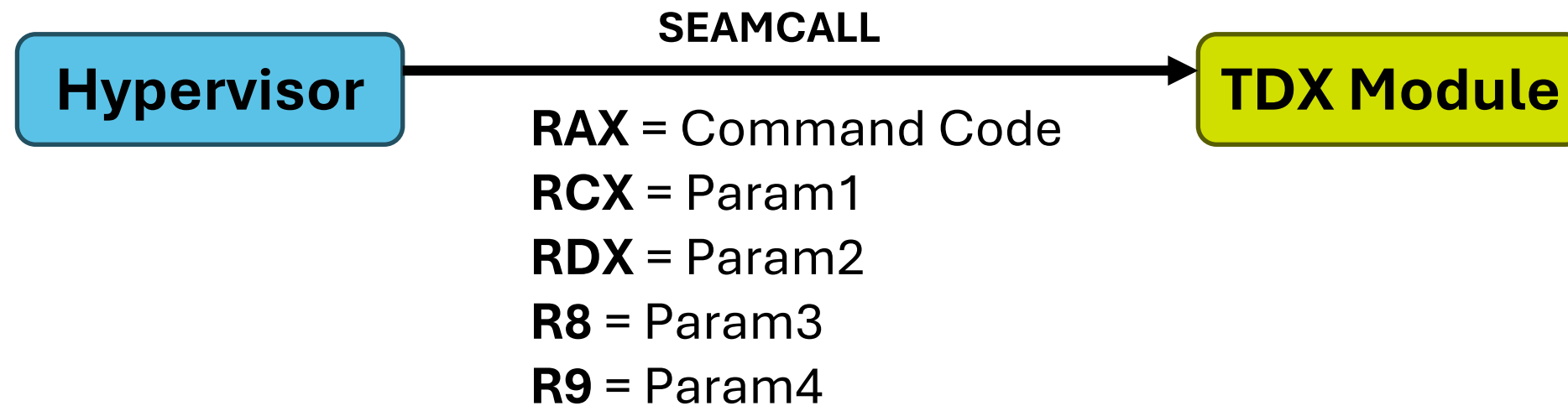


# Privileges

- “TDX Mode”, has access to the SEAM Range



# SEAMCALL commands



- The **SEAMCALL** interface implements commands, with parameters passed in registers
- Similar to **SYSCALL** / **SYSRET** to implement syscalls on traditional kernels
- Around ~80 commands
- Mostly guest management: “Create a guest”, ..., “Run a guest”



# Command: “run the guest”

*Guest Mode*

-----

*TDX Mode*

-----

*Host mode*

**Hypervisor**

*Time*

# Command: “run the guest”

*Guest Mode*

*TDX Mode*

**TDX Module**

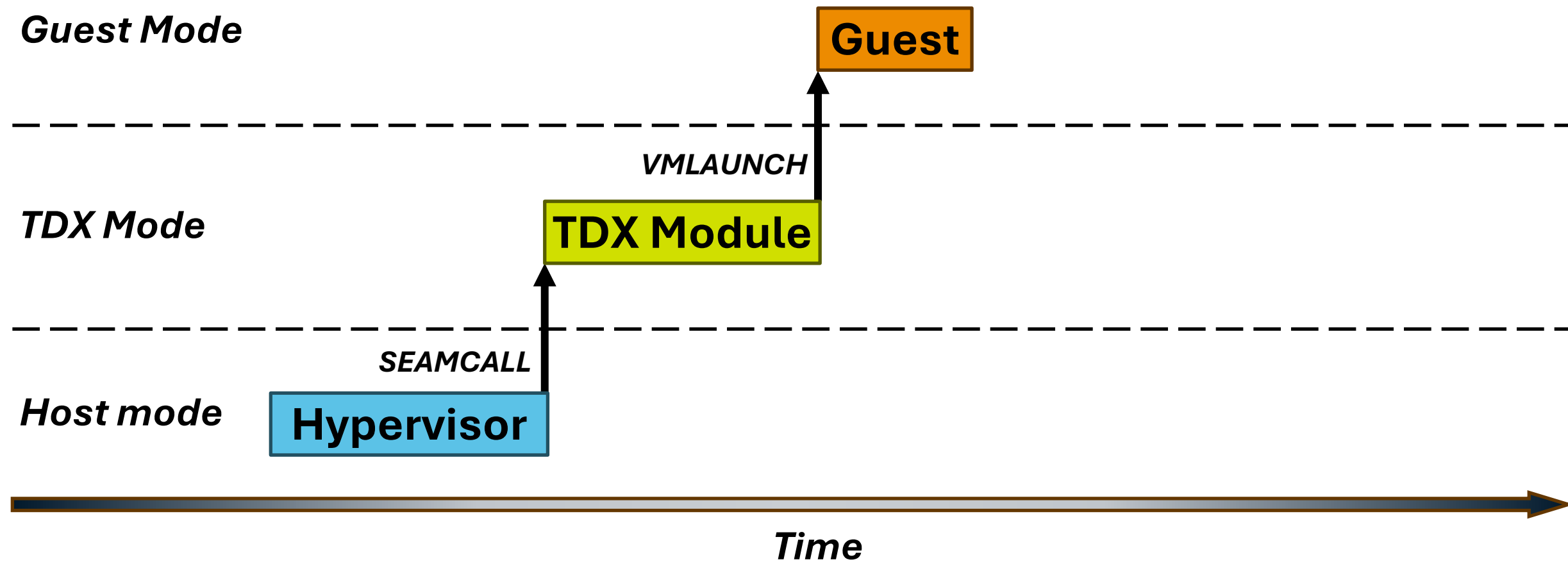
*SEAMCALL*

*Host mode*

**Hypervisor**

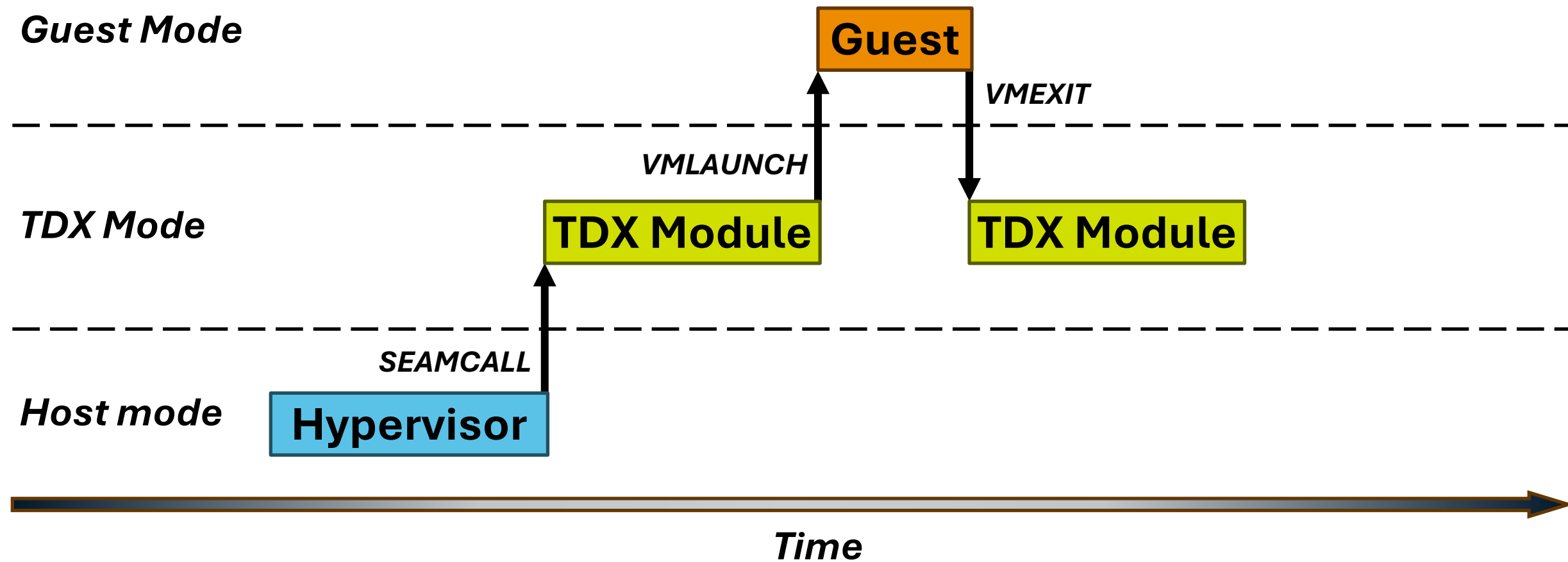
*Time*

# Command: “run the guest”

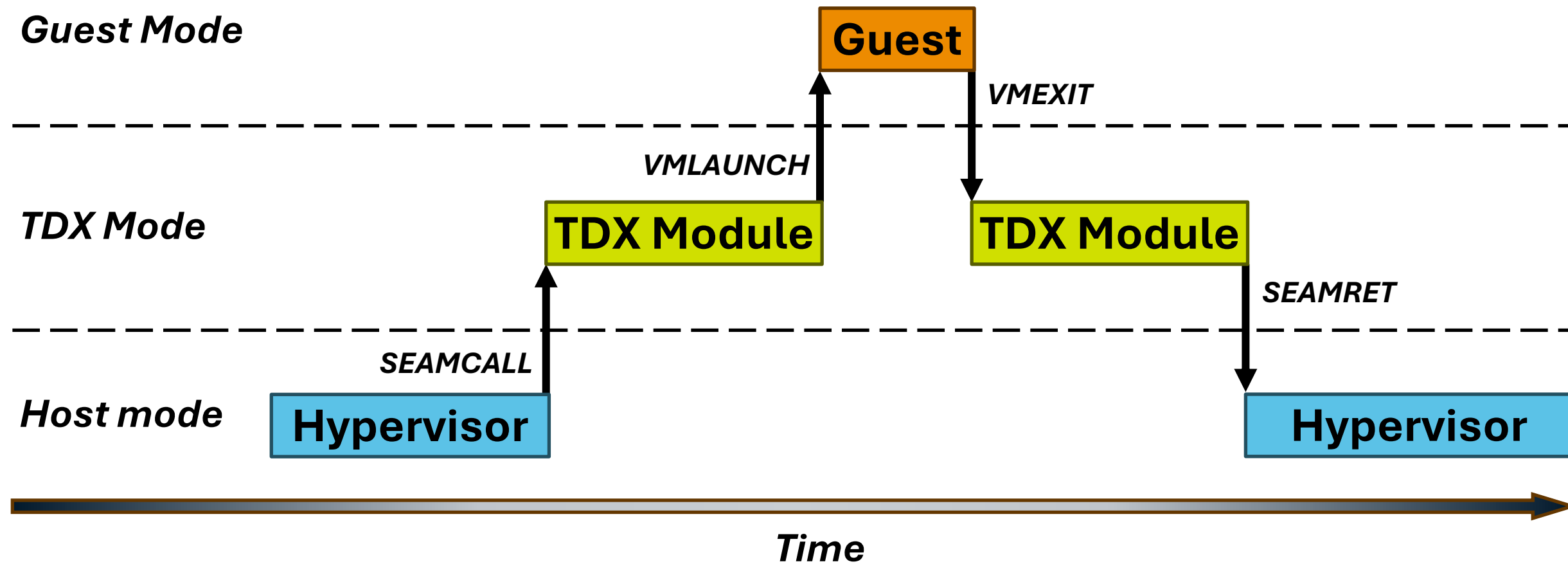




# Command: “run the guest”



# Command: “run the guest”





1. The TDX Module: technical overview

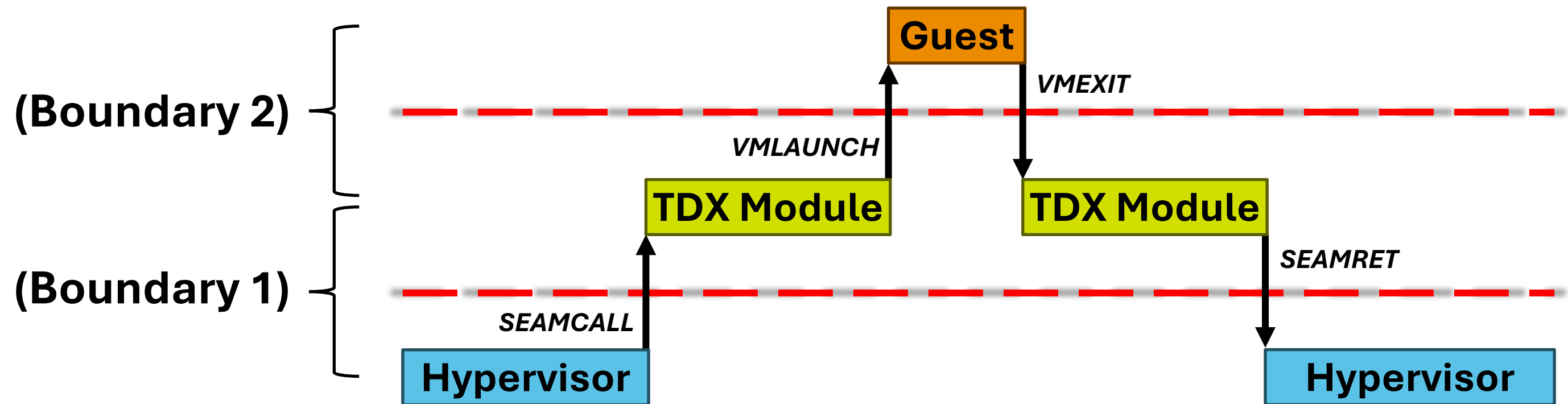
**2. Research approach and first findings**

3. Vulnerability 1

4. Vulnerability 2

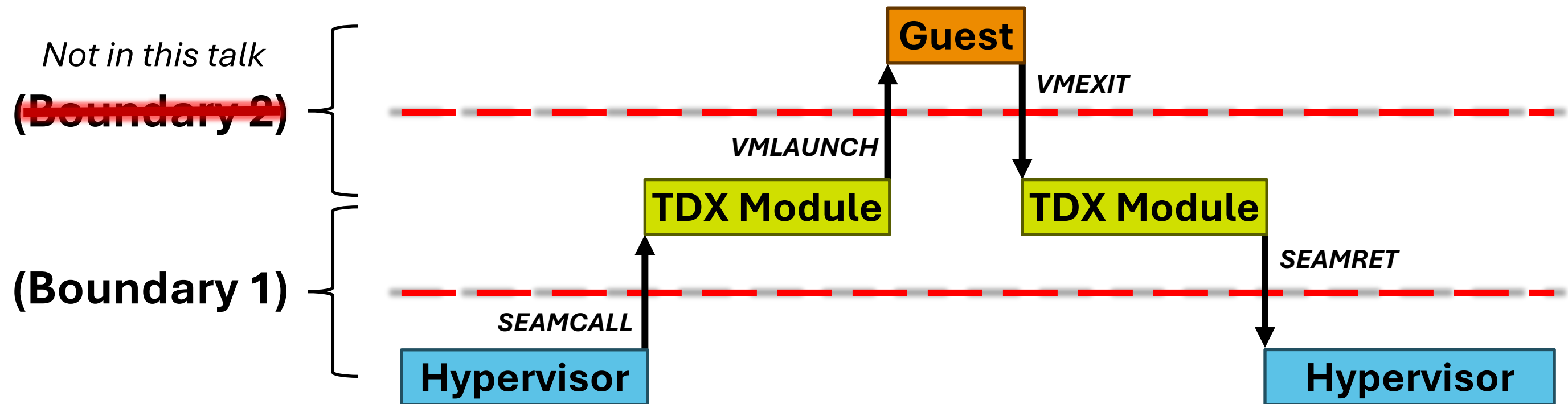


# Where to look for vulns?





# Where to look for vulns?



-  **Attack scenario:** the hypervisor is compromised, and tries to steal customer data

# Execution environment?

- Need a physical machine with a new-generation Intel CPU that supports TDX
- Execute **SEAMCALL** to talk to the TDX Module
- But we can't attach a debugger, can't inspect register states, can't inspect memory...
- 💡 Life is going to be hard if we go down that road

# Introducing Cornelius

- Started as an intellectual exercise to learn more about TDX
- Turned into a full emulator able to run the TDX Module in a VM
- Does not require TDX hardware
- Full introspection capabilities: can inspect register states and memory
- Bonus features: support for **ASAN**, **UBSAN**, **SANCOV**
- 🗨 Life is easy now



# Cornelius demo

```
Windows PowerShell
PS C:\br\Cornelius\Binaries> .\Test.exe .\pseamlr_1.5.01.02.so.consts .\pseamlr_1.5.01.02.so .\libtdx_1.5.01-pc.so
[+] Creating the Cornelius VM
[+] Executing PSEAMLR.INSTALL on VCPU0
[+] Executing PSEAMLR.INSTALL on VCPU1
[+] Executing PSEAMLR.INSTALL on VCPU2
[+] Executing PSEAMLR.INSTALL on VCPU3
[+] Executing TDH.SYS.INIT
[+] Executing TDH.SYS.LP.INIT on VCPU0
[+] Executing TDH.SYS.LP.INIT on VCPU1
[+] Executing TDH.SYS.LP.INIT on VCPU2
[+] Executing TDH.SYS.LP.INIT on VCPU3
[+] Executing TDH.SYS.CONFIG
[+] Executing TDH.SYS.KEY.CONFIG
[+] Executing TDH.SYS.TDMR.INIT
[+] Executing TDH.MNG.CREATE
[+] Executing TDH.MNG.KEY.CONFIG
[+] Executing TDH.MNG.ADDCX
[+] Executing TDH.MNG.INIT
[+] Executing TDH.VP.CREATE on TDVCPU0
[+] Executing TDH.VP.ADDCX on TDVCPU0
[+] Executing TDH.VP.INIT on TDVCPU0
[+] Executing TDH.VP.CREATE on TDVCPU1
[+] Executing TDH.VP.ADDCX on TDVCPU1
[+] Executing TDH.VP.INIT on TDVCPU1
[+] Executing TDH.MEM.SEPT.ADD with entry level 3
[+] Executing TDH.MEM.SEPT.ADD with entry level 2
[+] Executing TDH.MEM.SEPT.ADD with entry level 1
[+] Executing TDH.MR.FINALIZE
[+] Executing TDH.MEM.PAGE.AUG
[+] Executing TDH.VP.ENTER
[+] Taking snapshot
[+] TD executing TDG.MEM.PAGE.ACCEPT
[+] Doing TdVmexit in TD guest
[+] TD executing PSEAMLR.INFO
[+] Restoring snapshot
[+] Doing TdVmexit in TD guest
[+] P-SEAMLR coverage count: 224
[+] TDX module coverage count for VCPU0: 4806
[+] TDX module coverage count for VCPU1: 61
[+] TDX module coverage count for VCPU2: 61
[+] TDX module coverage count for VCPU3: 61
[+] Finished successfully
PS C:\br\Cornelius\Binaries>
```



# Cornelius demo

```
[+] TDX module coverage count for VCPU0: 4806  
[+] TDX module coverage count for VCPU1: 61  
[+] TDX module coverage count for VCPU2: 61  
[+] TDX module coverage count for VCPU3: 61  
[+] Finished successfully
```

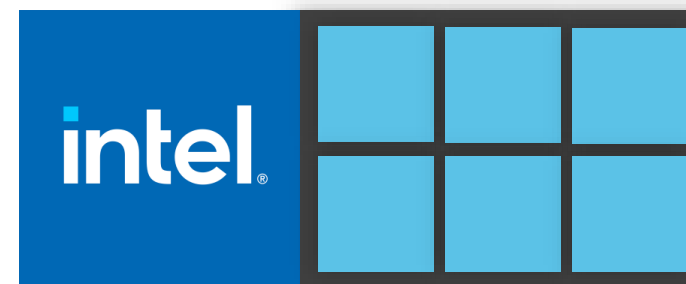
- Zooming in
- Sancov: we executed most commands on CPU0, so CPU0 has the highest coverage

# Initial assessment: no easy vulns

- Started looking for standard vulnerabilities, didn't find any...
- Good programming guidelines: extensive testing, static analysis
- Good mitigations: **CET, IBT, ASLR, W<sup>X</sup>**, etc
- Overall good quality, limited opportunities for traditional memory corruptions
- 💡 **Will have to think harder**

# Looking at context-switching

- During **SEAMCALL** and **SEAMRET**, the CPU performs a context-switch
- **SEAMCALL:**

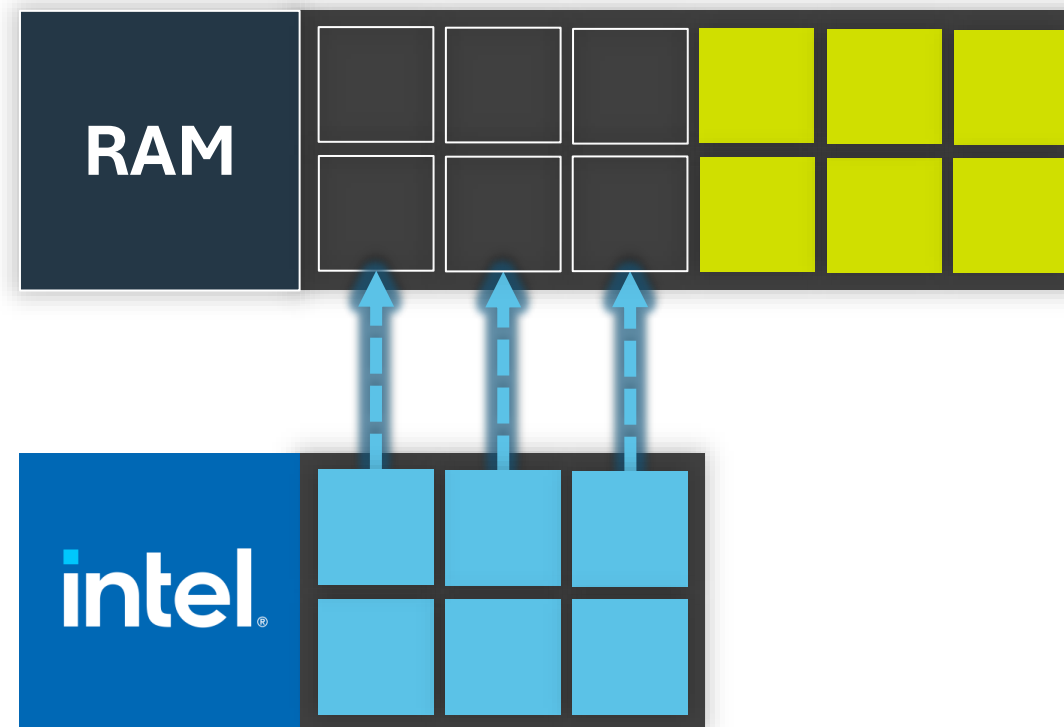


 Hypervisor value

 TDX Module value

# Looking at context-switching

- During **SEAMCALL** and **SEAMRET**, the CPU performs a context-switch
- **SEAMCALL:**



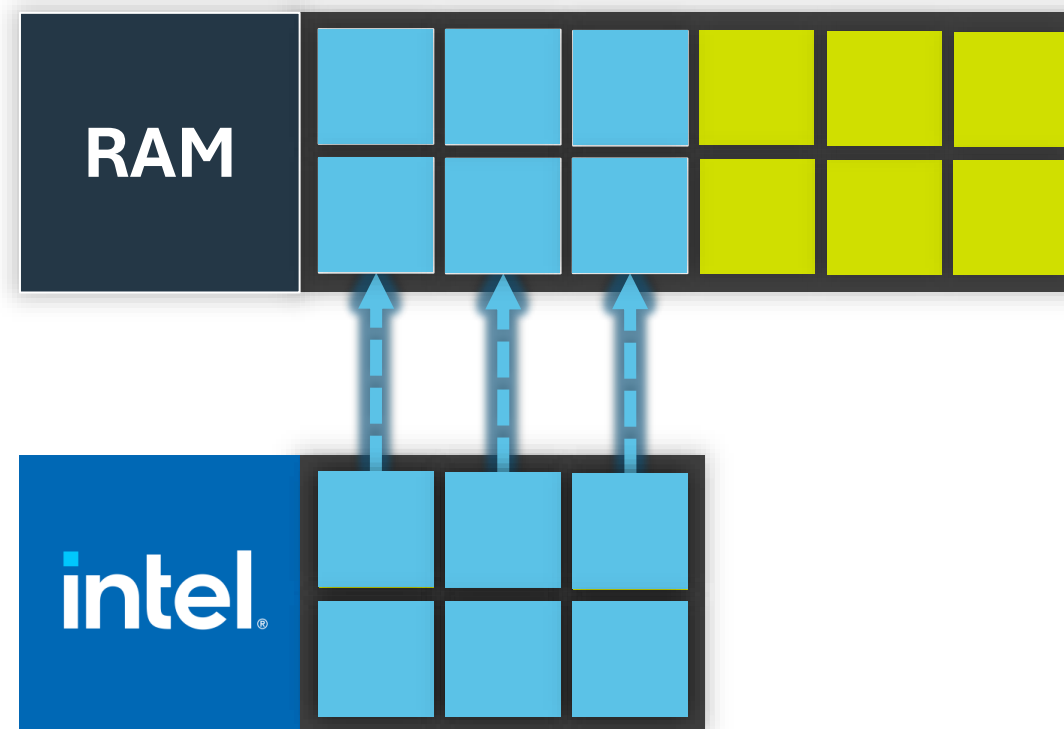
 Hypervisor value

 TDX Module value



# Looking at context-switching

- During **SEAMCALL** and **SEAMRET**, the CPU performs a context-switch
- **SEAMCALL:**

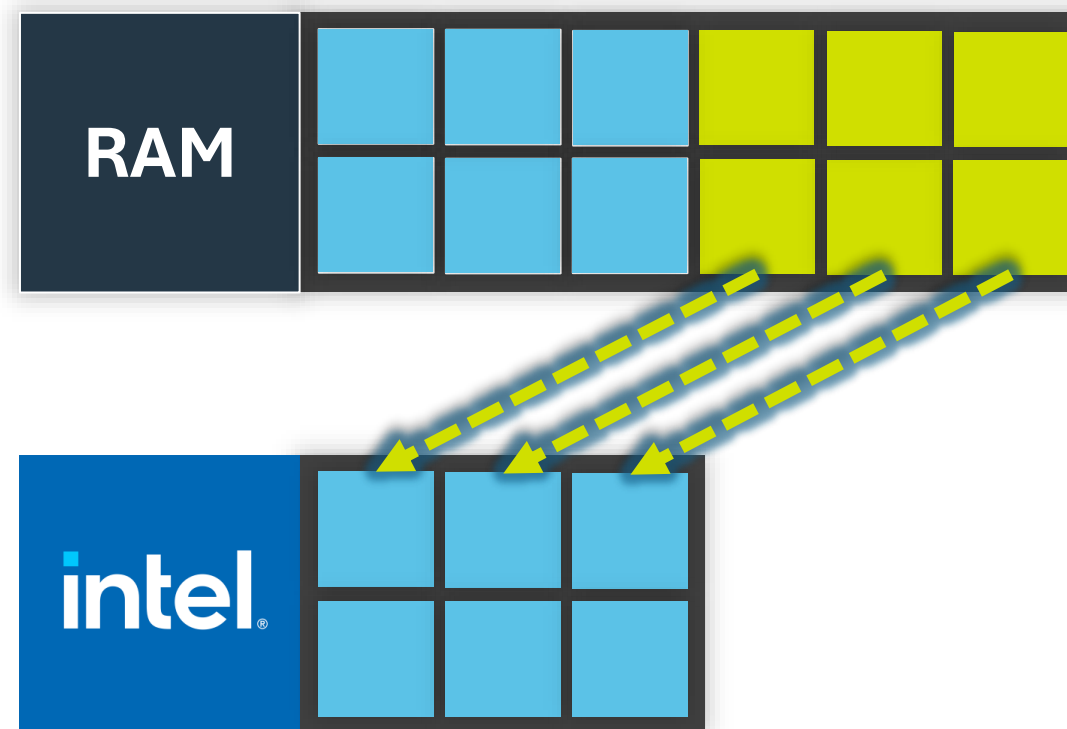


 Hypervisor value

 TDX Module value

# Looking at context-switching

- During **SEAMCALL** and **SEAMRET**, the CPU performs a context-switch
- **SEAMCALL:**

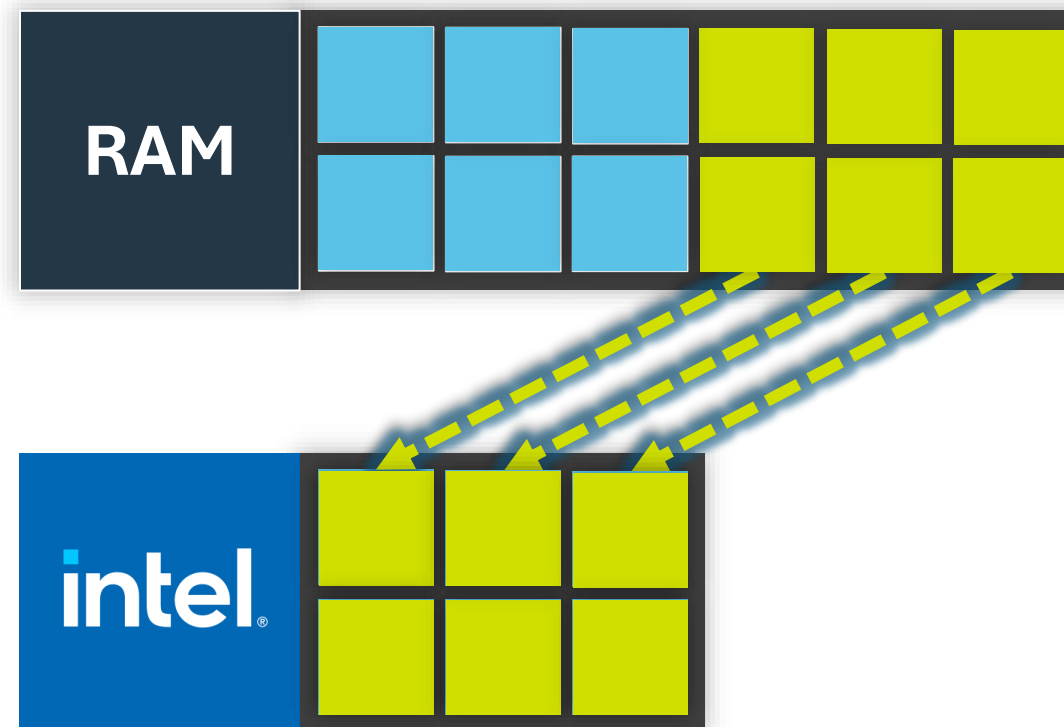


 Hypervisor value

 TDX Module value

# Looking at context-switching

- During **SEAMCALL** and **SEAMRET**, the CPU performs a context-switch
- **SEAMCALL:**



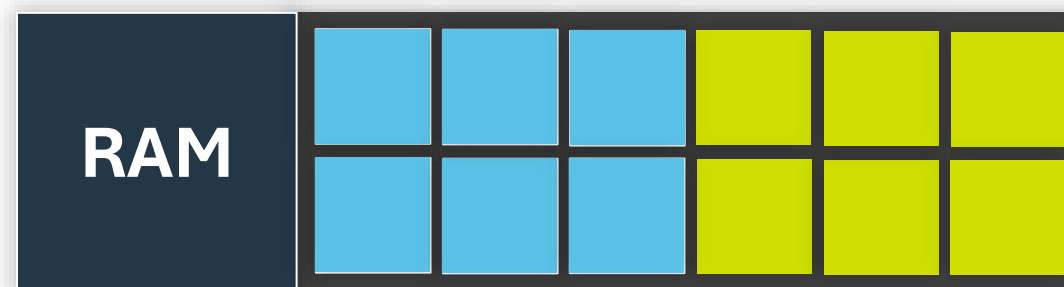
Hypervisor value



TDX Module value

# Looking at context-switching

- During **SEAMCALL** and **SEAMRET**, the CPU performs a context-switch
- **SEAMCALL:**



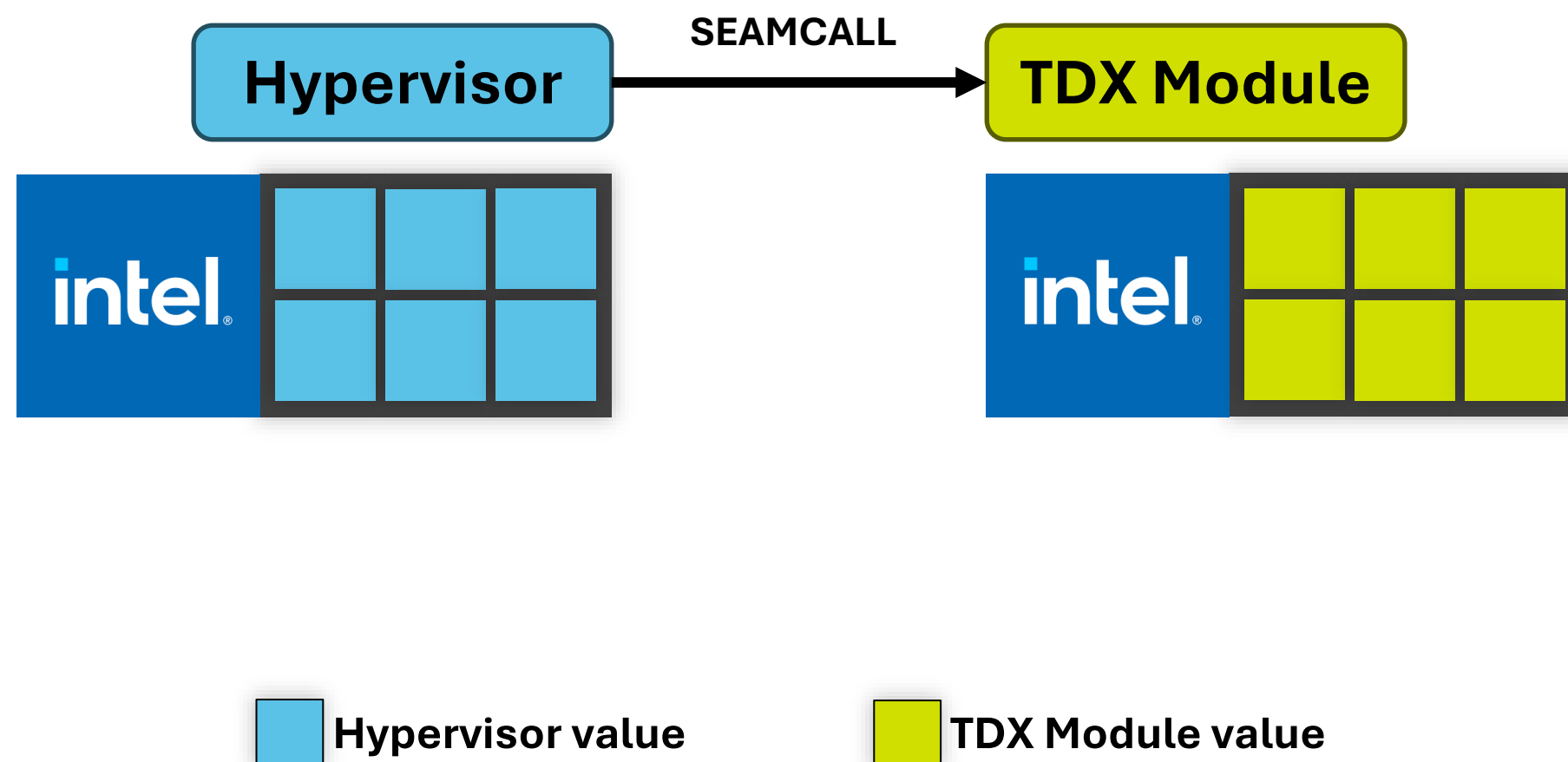
 Hypervisor value

 TDX Module value



# Looking at context-switching

- During **SEAMCALL** and **SEAMRET**, the CPU performs a context-switch
- **SEAMCALL**:

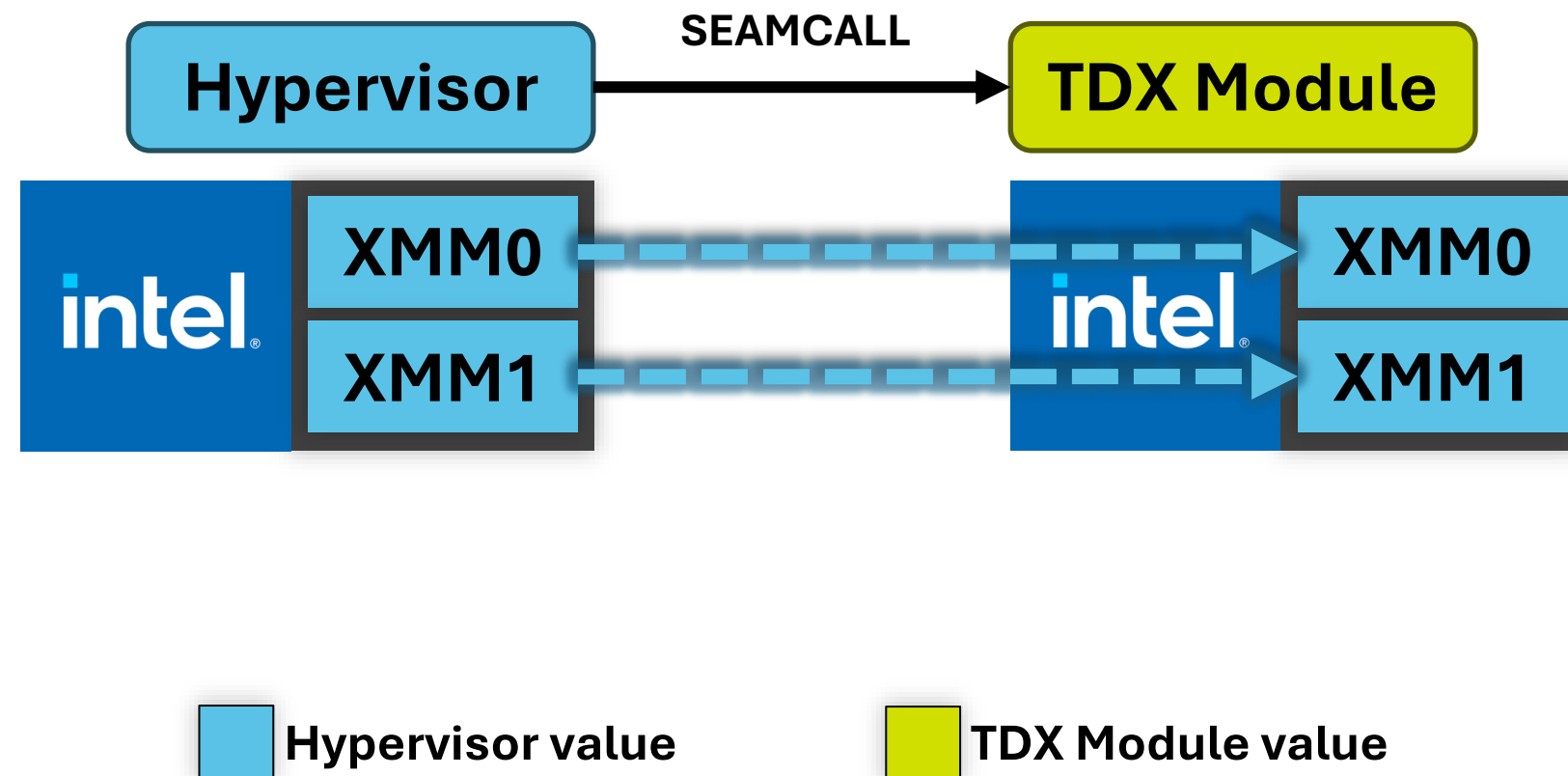


# Looking at context-switching

- During **SEAMCALL** and **SEAMRET**, the CPU performs a context-switch
- **But!** Not all registers are context-switched by the CPU
- The TDX Module has to context-switch some registers itself manually...
- 💬 ... Does it do so correctly?

# Context-switching: a quick test

- The **XMM** registers are switched **neither** by the CPU **nor** by the TDX Module
- The TDX Module doesn't use **XMM** registers so it doesn't bother, which is fine



# Context-switching: a quick test

- 💬 We can disable XMM registers in Cornelius, right?
- ... *Right?*





# Context-switching: a quick test

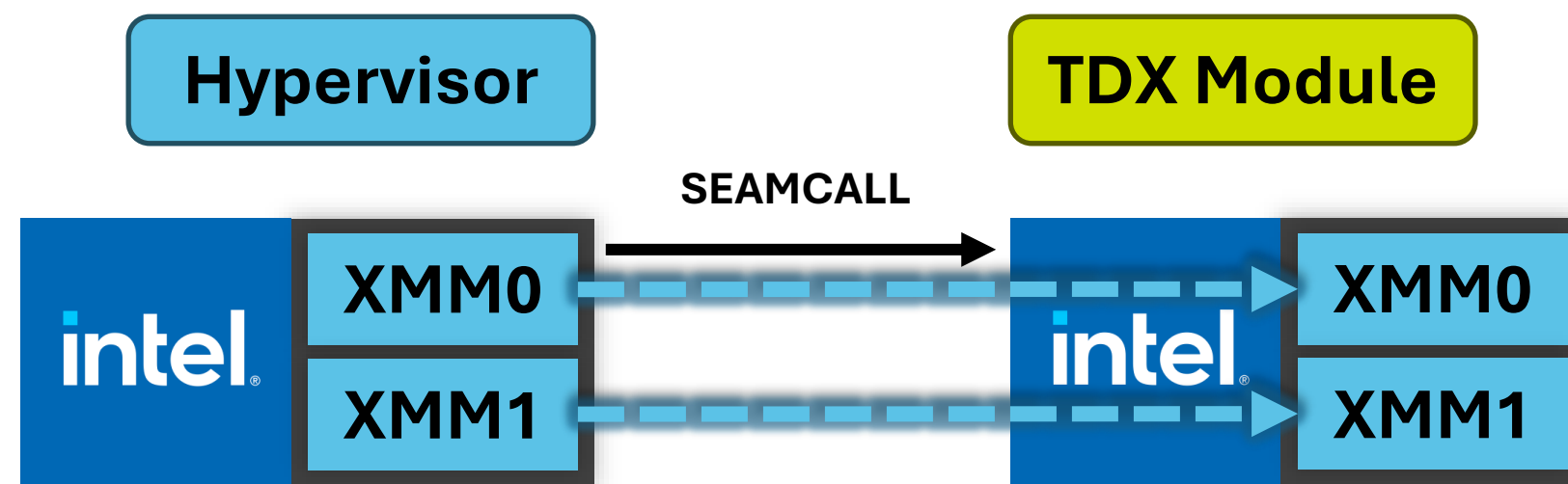
Windows PowerShell

```
PS C:\br\Cornelius\Binaries> .\Test.exe .\pseamldr_1.5.01.02.so.consts .\pseamldr_1.5.01.02.so .\libtdx_1.5.01-pc.so
[+] Creating the Cornelius VM
[+] Executing PSEAMLDR.INSTALL on VCPU0
[!][VCPU0] Unrecognized instruction on #UD at RIP = 0xffff800000009b6a
[!][VCPU0] Emulation error, RIP=ffff800000009b6a, callstack:
[!][VCPU0] > 0xffff800000013c63
PS C:\br\Cornelius\Binaries>
```

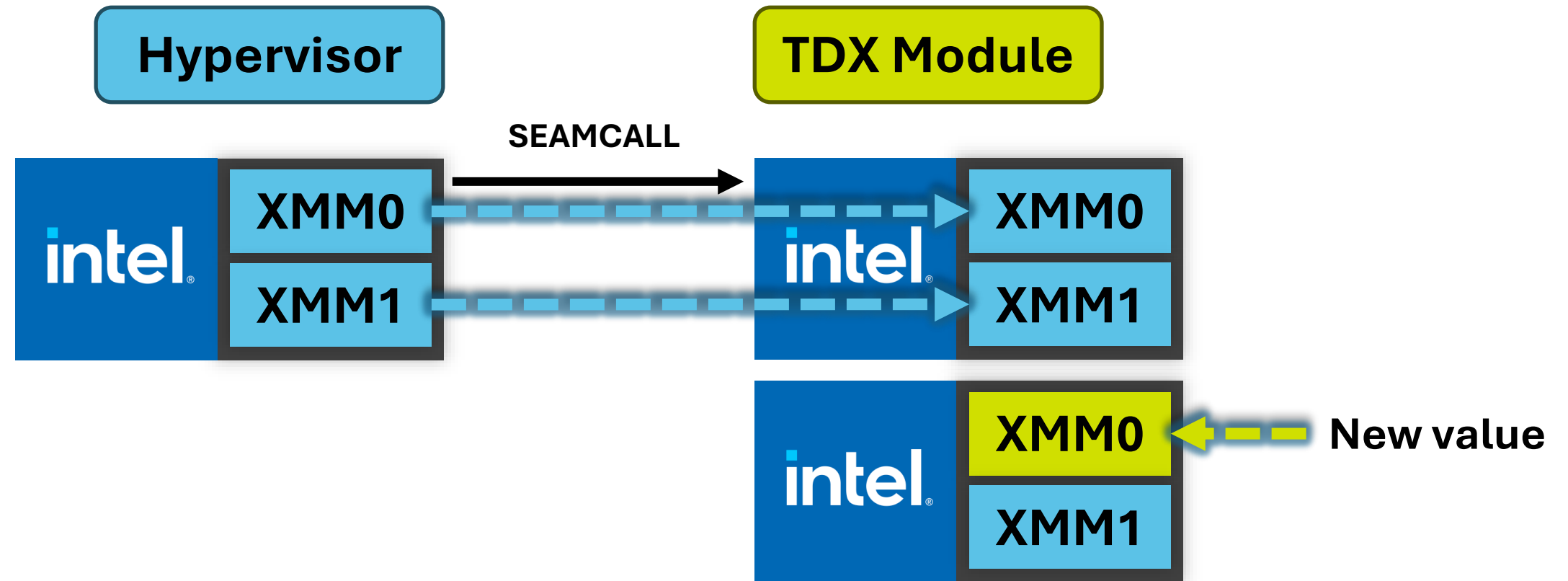
# A first bug

- Disabling **XMM** registers in Cornelius causes the TDX Module to crash
- Because the TDX Module does in fact use XMM registers
- But forgot to context-switch them

# A first bug

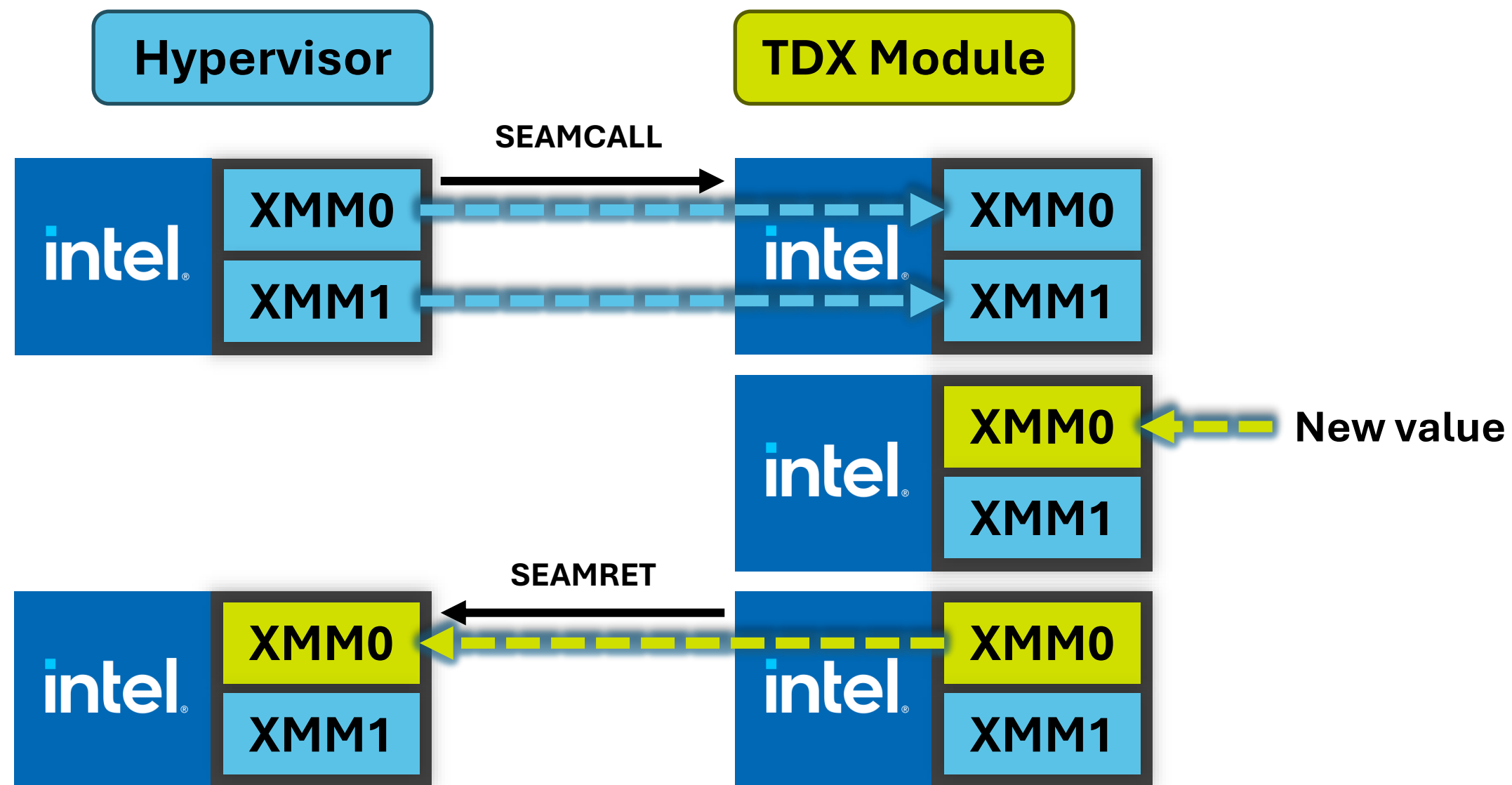


# A first bug





# A first bug



# Two impacts

1. Hypervisor registers get corrupted
2. Guest secrets can be leaked

# Two impacts

1. Hypervisor registers get corrupted
  2. ~~Guest secrets can be leaked~~ *Not the case*
- Intel fixed it as a functional bug

# Two impacts

1. Hypervisor registers get corrupted
2. ~~Guest secrets can be leaked~~ *Not the case*
  - Intel fixed it as a functional bug
  - 💡 Found in 20 seconds





1. The TDX Module: technical overview
2. Research approach and first findings
3. Vulnerability 1
4. Vulnerability 2



# ProcessorTrace: background

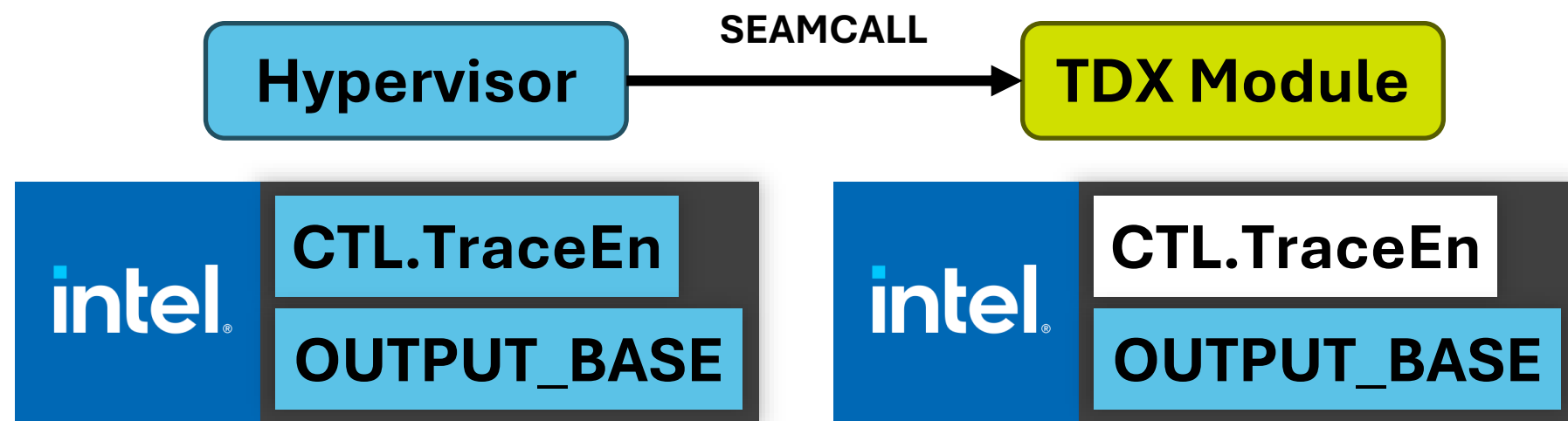
- ProcessorTrace (PT): tracing feature
- The CPU *records* the execution and creates a log in memory
- The TDX Module supports PT in guests

# ProcessorTrace: registers

- PT is controlled by several registers
- Two registers are important:
  1. **IA32\_RTIT\_CTL**: has a **TraceEn** bit that enables tracing
  2. **IA32\_RTIT\_OUTPUT\_BASE**: contains the physical address where the log is written

# ProcessorTrace context switches

- **SEAMCALL**: the CPU forces **CTL.TraceEn** to zero



Hypervisor value



TDX Module value

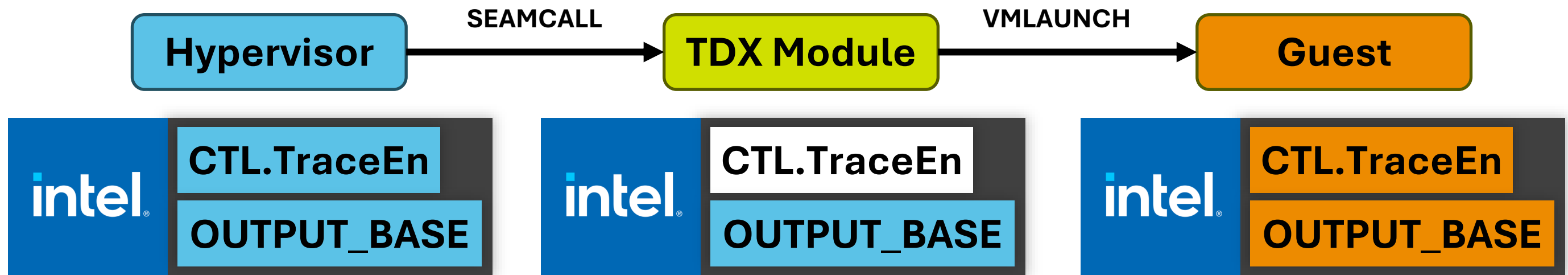


Guest value



# ProcessorTrace context switches

- **SEAMCALL**: the CPU forces **CTL.TraceEn** to zero
- **VMLAUNCH**: the TDX module does a c-switch in software to install the guest values



 Hypervisor value

 TDX Module value

 Guest value



# ProcessorTrace context switches

- Focus on the 2<sup>nd</sup> context switch
- Made in software



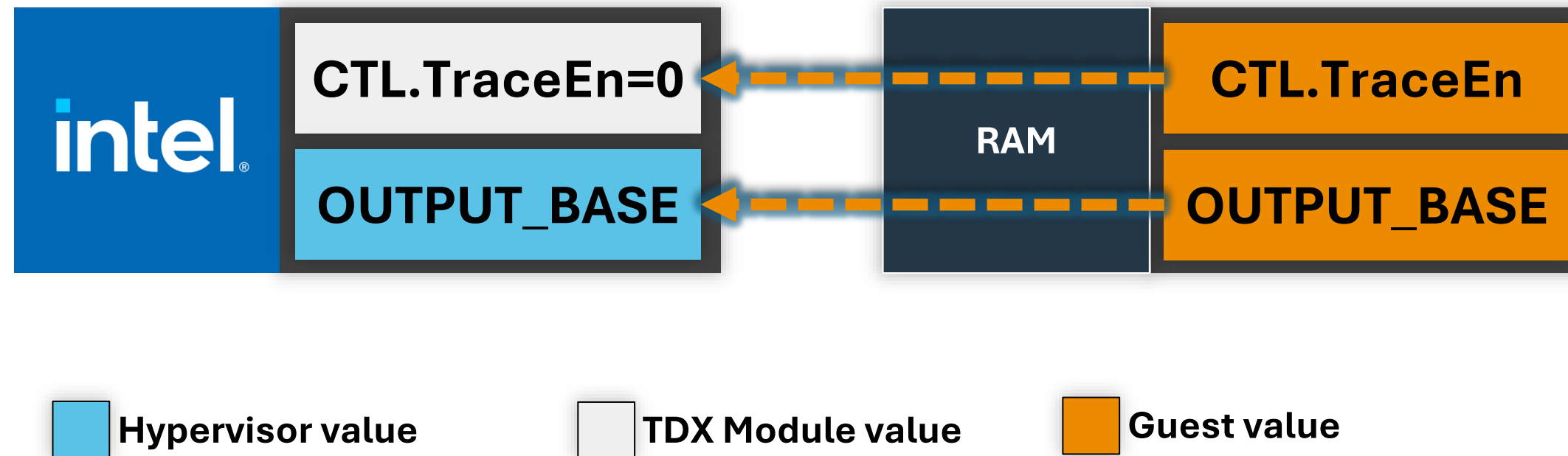
 Hypervisor value

 TDX Module value

 Guest value

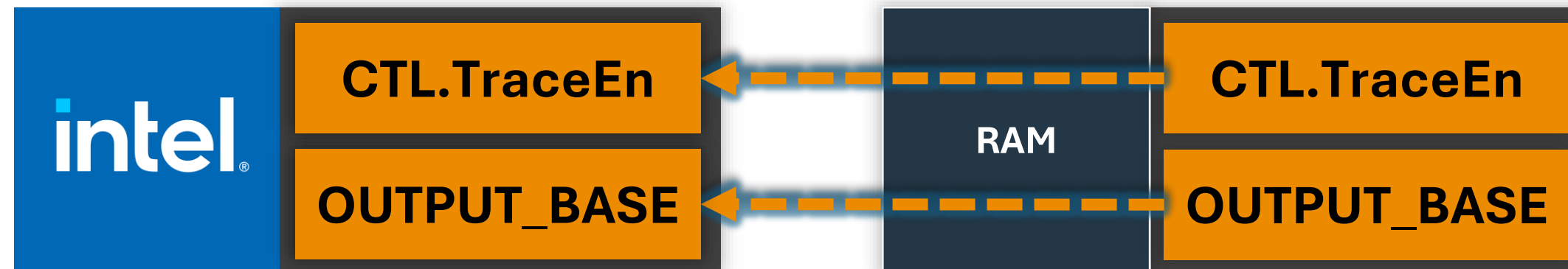
# ProcessorTrace context switches

- Focus on the 2<sup>nd</sup> context switch
- Made in software



# ProcessorTrace context switches

- Focus on the 2<sup>nd</sup> context switch
- Made in software



 Hypervisor value

 TDX Module value

 Guest value

# ProcessorTrace context switches

- Focus on the 2<sup>nd</sup> context switch
- Made in software

*Guest Mode*

-----

*TDX Mode*

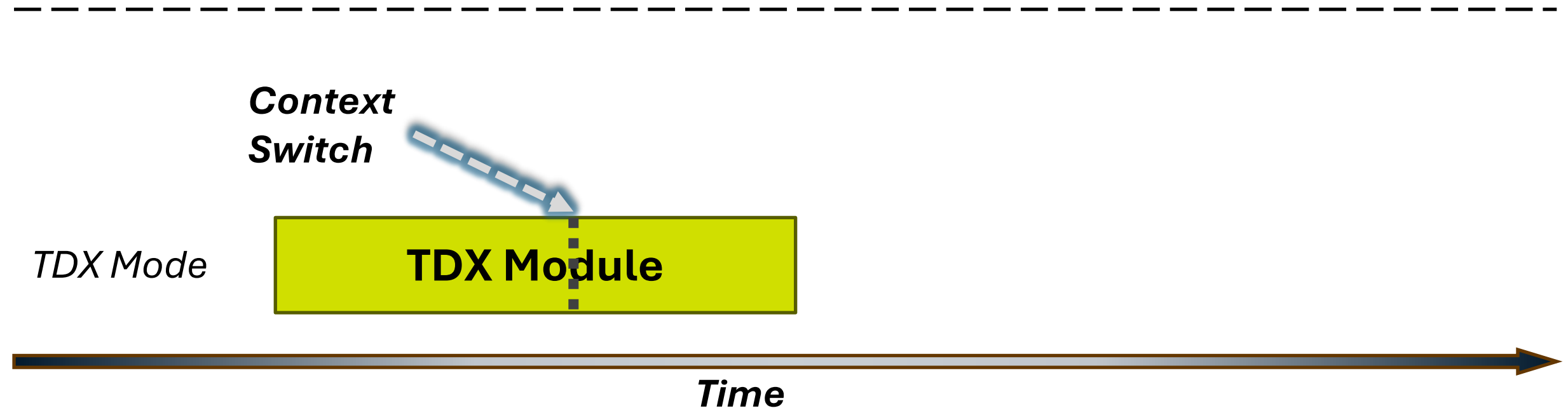
**TDX Module**

*Time*

# ProcessorTrace context switches

- Focus on the 2<sup>nd</sup> context switch
- Made in software

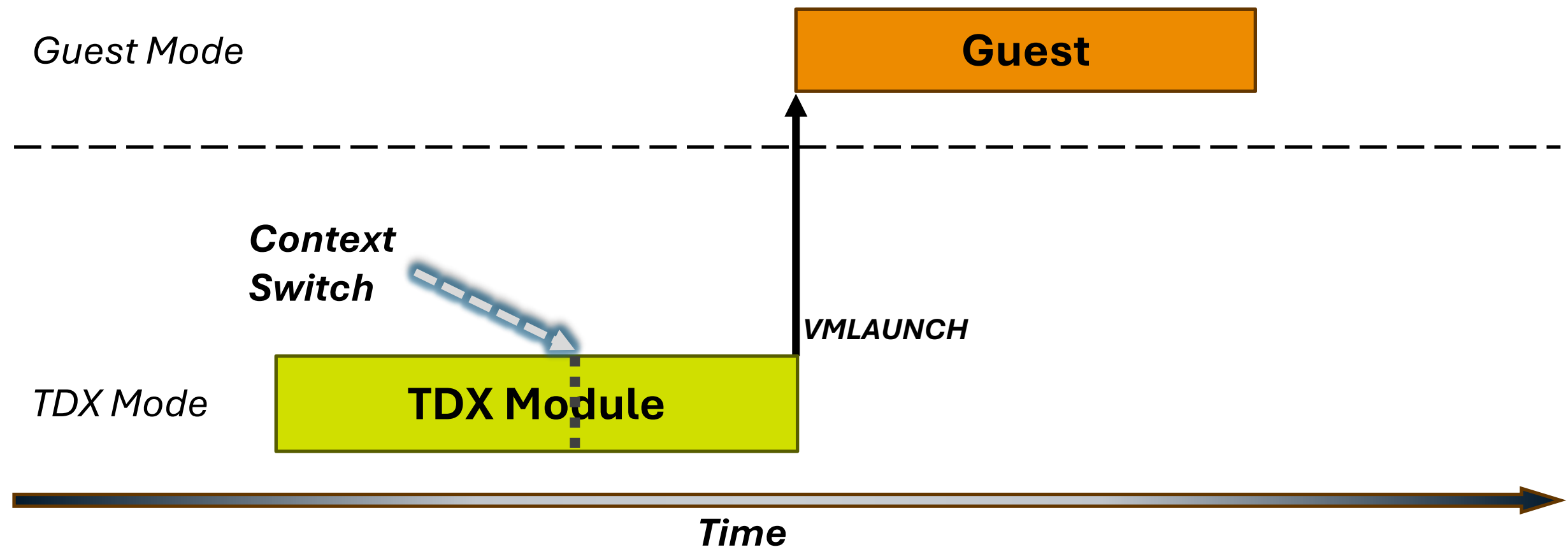
*Guest Mode*





# ProcessorTrace context switches

- Focus on the 2<sup>nd</sup> context switch
- Made in software



# ProcessorTrace context switches

- Focus on the 2<sup>nd</sup> context switch
- Made in software
- The TDX Module is still executing afterwards
- 🗨 Problem? No...



In-memory  
value is zero

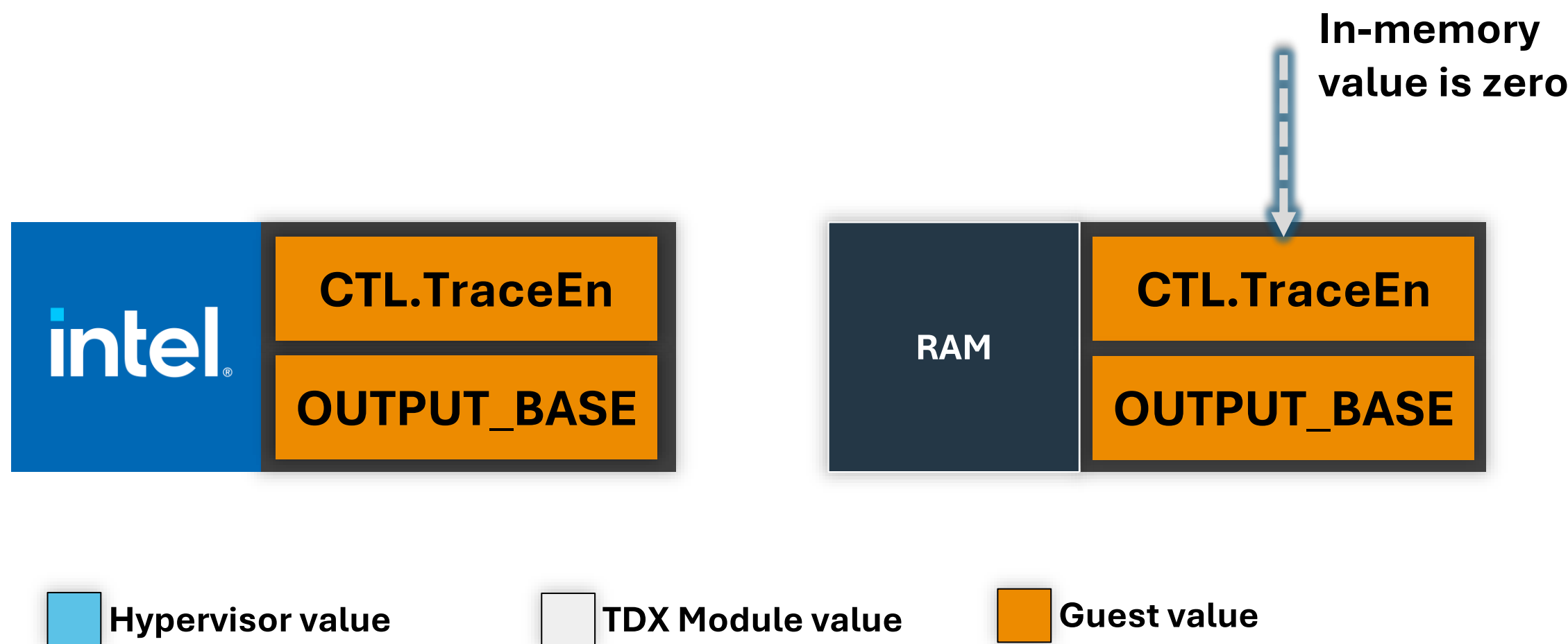
 Hypervisor value

 TDX Module value

 Guest value

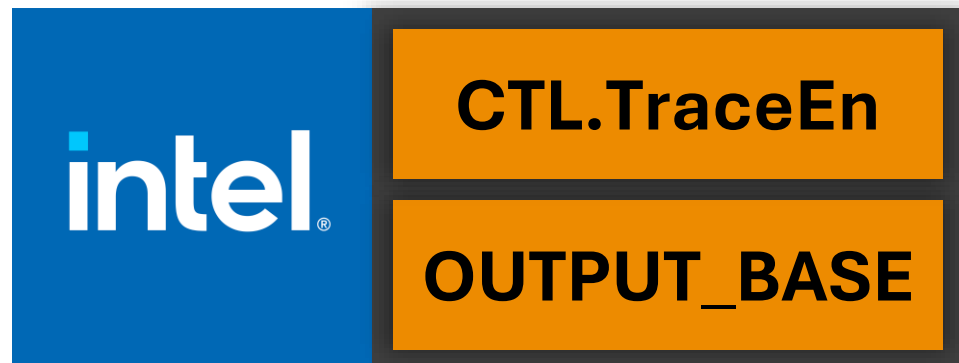
# Problem

- Except that... On debuggable guests, the in-memory state is accessible to the hypervisor!



# Problem

- Except that... On debuggable guests, the in-memory state is accessible to the hypervisor!
- The hypervisor can set **TraceEn**=1 in the in-memory state
- The hypervisor can therefore enable PT in the TDX Module



 Hypervisor value

 TDX Module value

 Guest value

# Vulnerability

- There is a window where PT is enabled in the TDX Module

*Guest Mode*

-----

*TDX Mode*

**TDX Module**

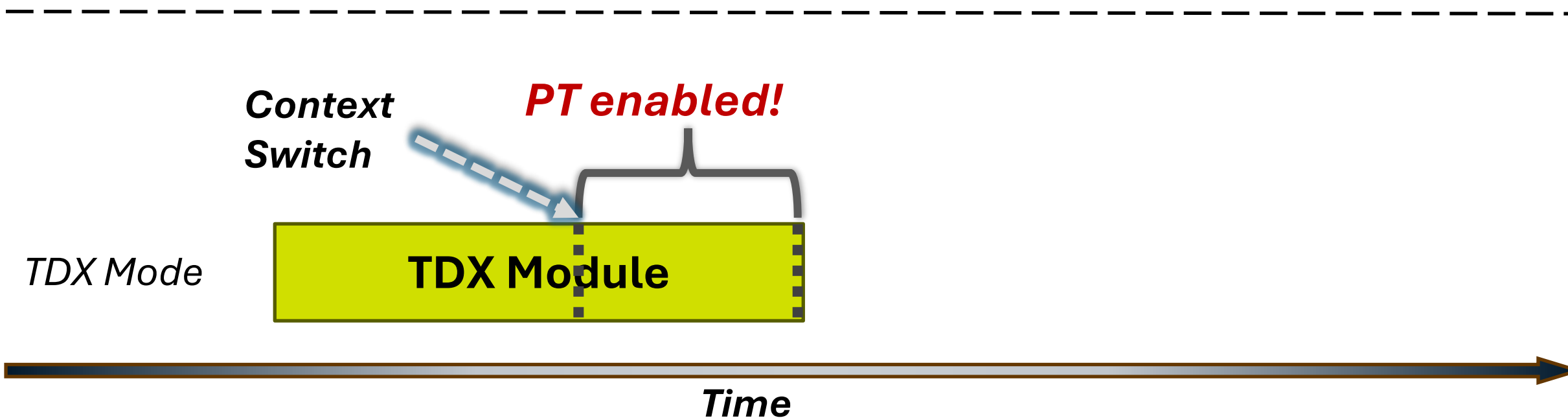
*Time*



# Vulnerability

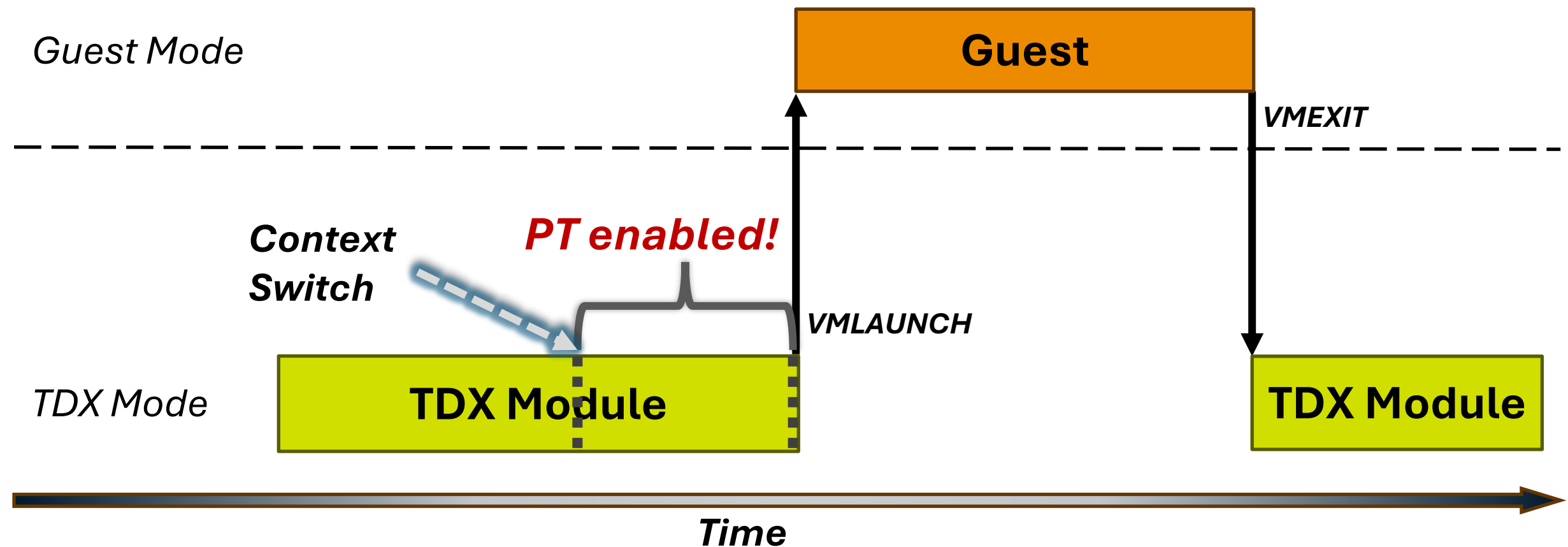
- There is a window where PT is enabled in the TDX Module

*Guest Mode*



# Vulnerability

- There is a window where PT is enabled in the TDX Module



# Assembling the pieces

- The hypervisor can control **OUTPUT\_BASE...**
- 💡 ... **Meaning: the hypervisor can decide where the log gets written to in memory**
- Via additional PT registers, the hypervisor can ~mostly control the contents of the log...
- 💡 ... **Meaning: the hypervisor can decide what data gets written in memory**
- While the TDX Module executes, the CPU is in TDX Mode...
- 💡 ... **Meaning: the SEAM Range is accessible**

# The primitive

- The hypervisor can set **OUTPUT\_BASE** to point to the SEAM Range, and have TDX memory be overwritten by the PT log, the contents of which are controlled by the hypervisor
- The hypervisor effectively gets a **write-what-where** primitive in TDX memory
- **Achieve complete privilege escalation**

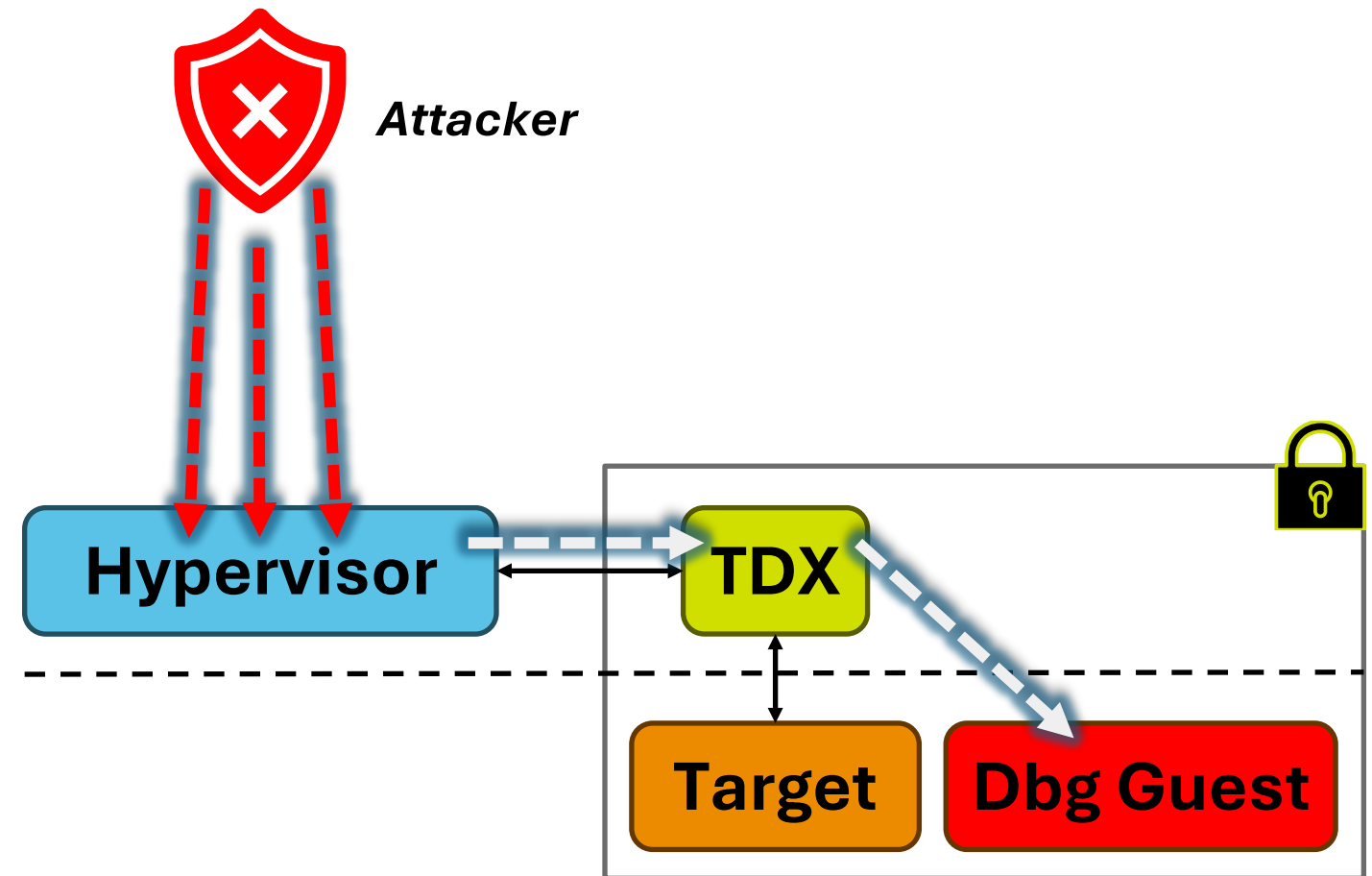
# The primitive

- The hypervisor can set **OUTPUT\_BASE** to point to the SEAM Range, and have TDX memory be overwritten by the PT log, the contents of which are controlled by the hypervisor
- The hypervisor effectively gets a **write-what-where** primitive in TDX memory
- **Achieve complete privilege escalation**
- What about ASLR in the TDX Module? ...
- ... ASLR is on the **virtual** memory, not the **physical** memory



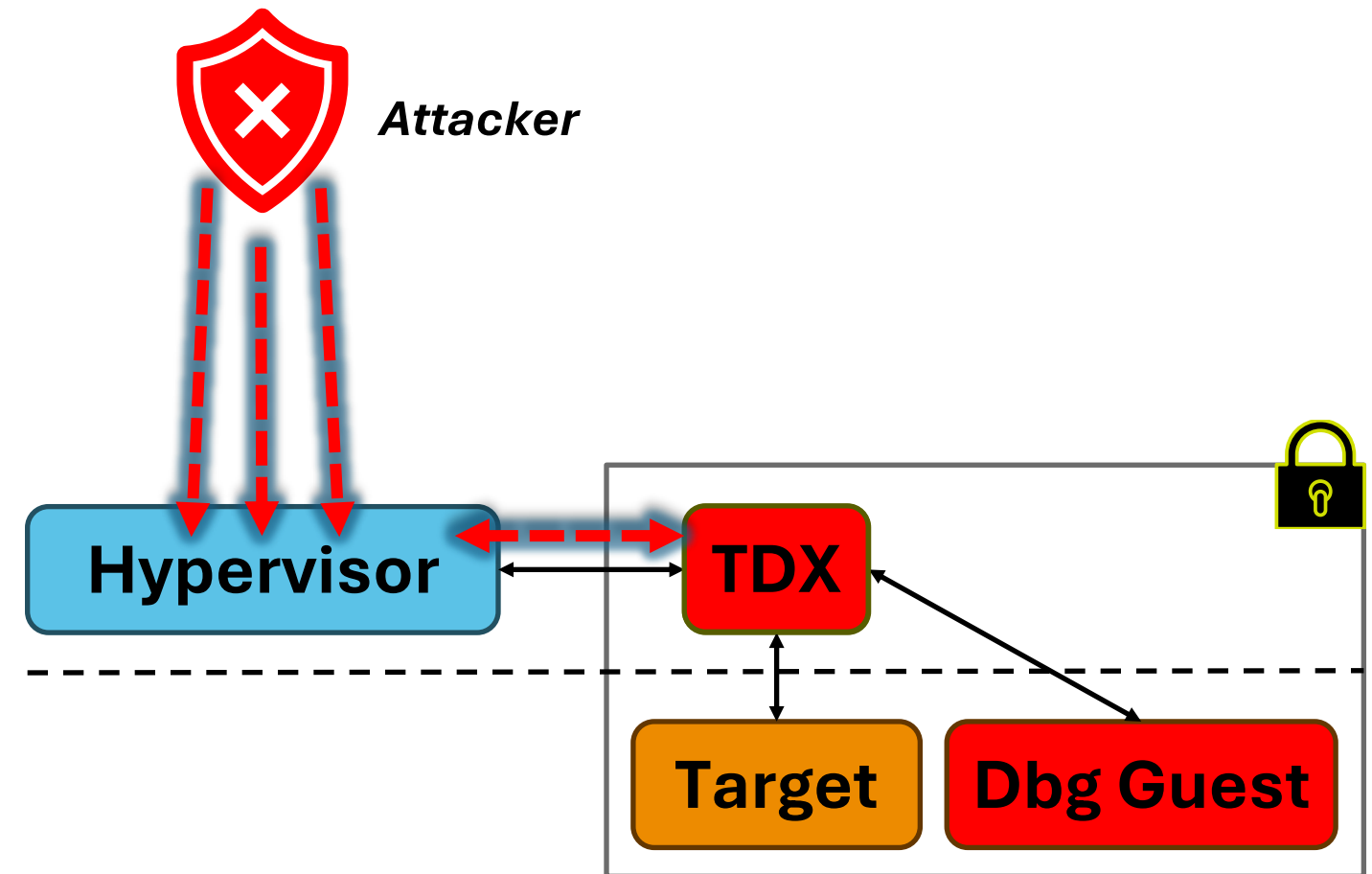
# Attack scenario

1. Create a debuggable guest



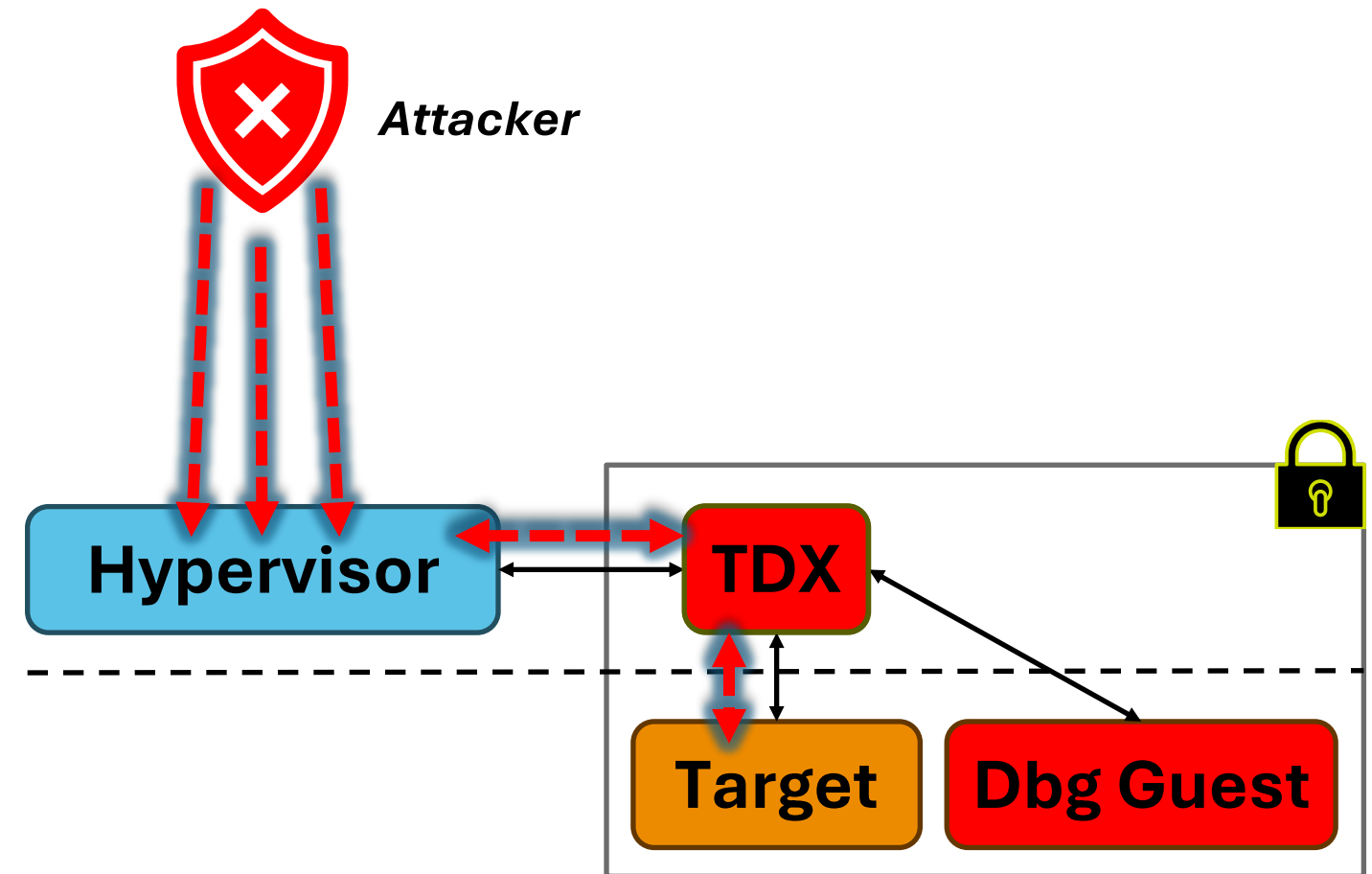
# Attack scenario

1. Create a debuggable guest
2. Escalate privileges into the TDX Module



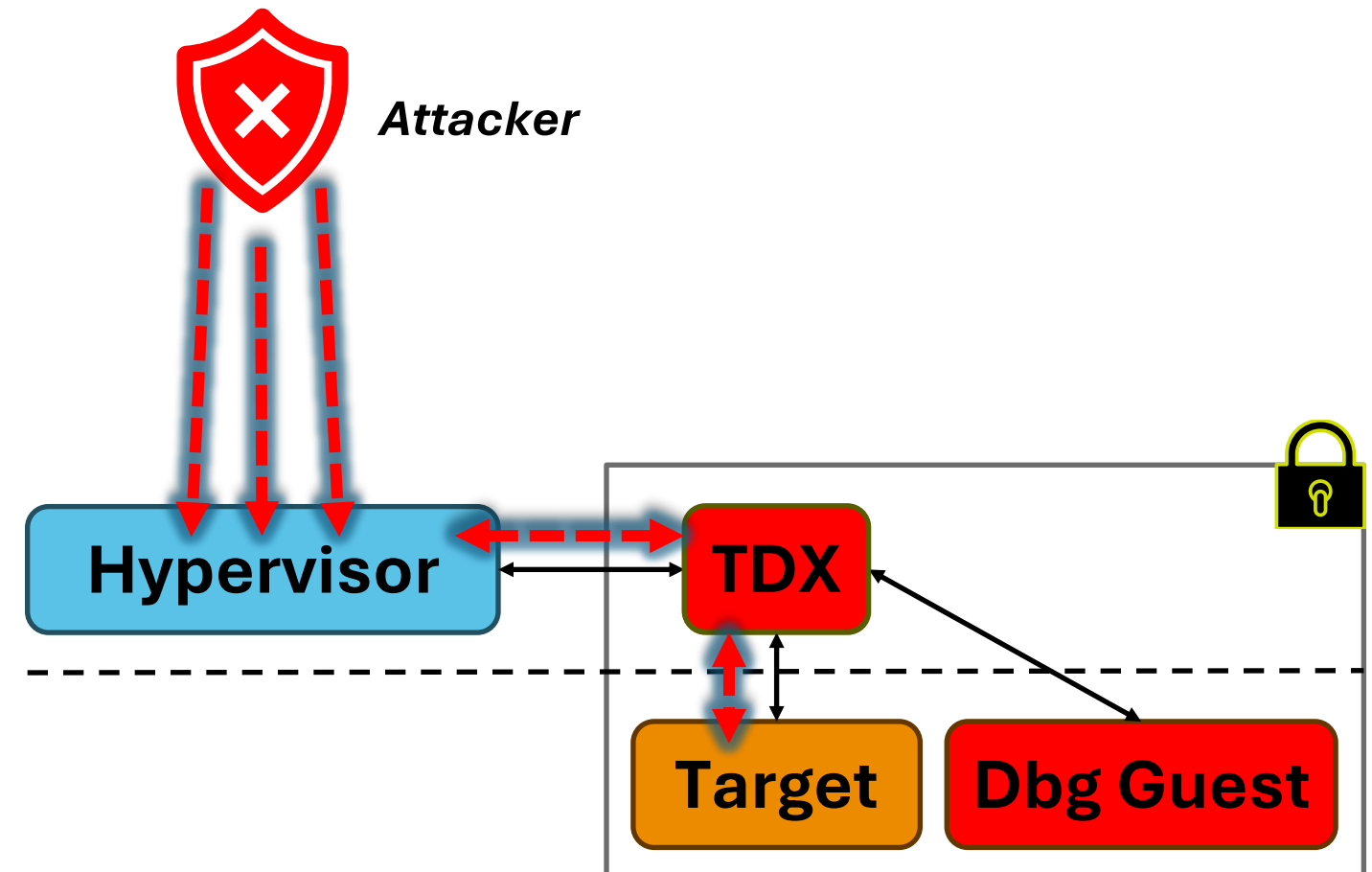
# Attack scenario

1. Create a debuggable guest
2. Escalate privileges into the TDX Module
3. Steal data from the target



# Attack scenario

1. Create a debuggable guest
  2. Escalate privileges into the TDX Module
  3. Steal data from the target
- Defeat the confidentiality guarantees
  - **CVE-2024-39283**
  - Affected all versions of the TDX Module
  - Fixed by Intel in version 1.5.01







1. The TDX Module: technical overview
2. Research approach and first findings
3. Vulnerability 1
4. Vulnerability 2





# What is SEAMCALL, actually?

- Looking at the very instruction pseudo-code, from the Intel specification

## Operation

IF not in VMX operation or inSMM or inSEAM or ((IA32\_EFER.LMA & CS.L) == 0)  
THEN #UD;  
ELSIF in VMX non-root operation  
THEN VMexit("basic reason" = SEAMCALL,  
"VM exit from VMX root operation" (bit 29) = 0);  
ELSIF CPL > 0 or IA32\_SEAMRR\_MASK.VALID == 0 or "events blocking by MOV-SS"  
THEN #GP(0);

(Source: [Intel® Trust Domain CPU Architectural Extensions](#))

# SEAMCALL unconditionality

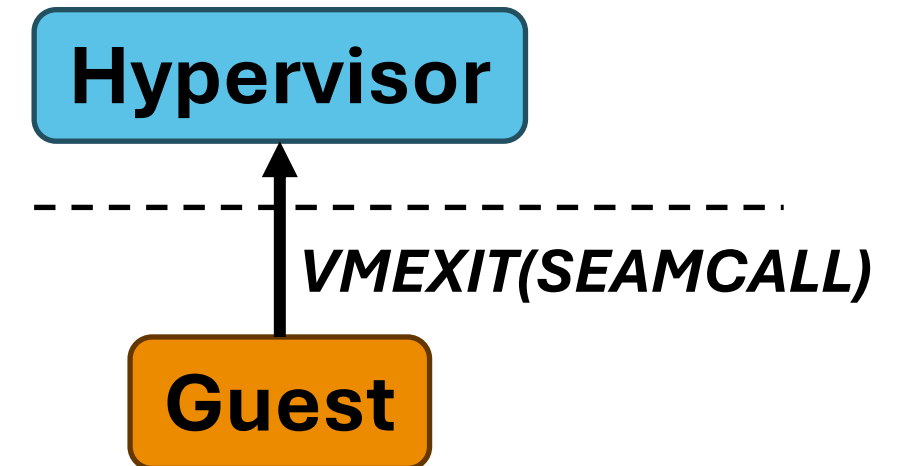
```
IF not in VMX operation or inSMM or inSEAM or ((IA32_EFER.LMA & CS.L) == 0)  
THEN #UD;
```

- **SEAMCALL** is unconditionally recognized
- No toggle to enable or disable it
- 💡 **Weird, normally there should be a toggle for new CPU features**


# VMEXIT(SEAMCALL) unconditionality

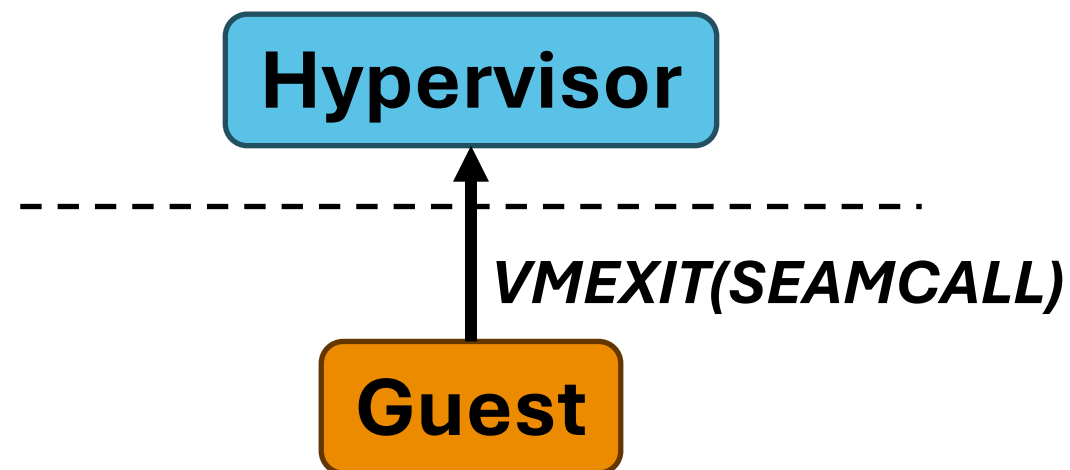
```
ELSIF in VMX non-root operation  
  THEN VMexit("basic reason" = SEAMCALL,  
    "VM exit from VMX root operation" (bit 29) = 0);
```

- **SEAMCALL** has a *VMEXIT Reason* associated to it
- The VMEXIT(SEAMCALL) unconditionally triggers if the guest executes **SEAMCALL**
- 💡 Weird again, normally there should be a toggle




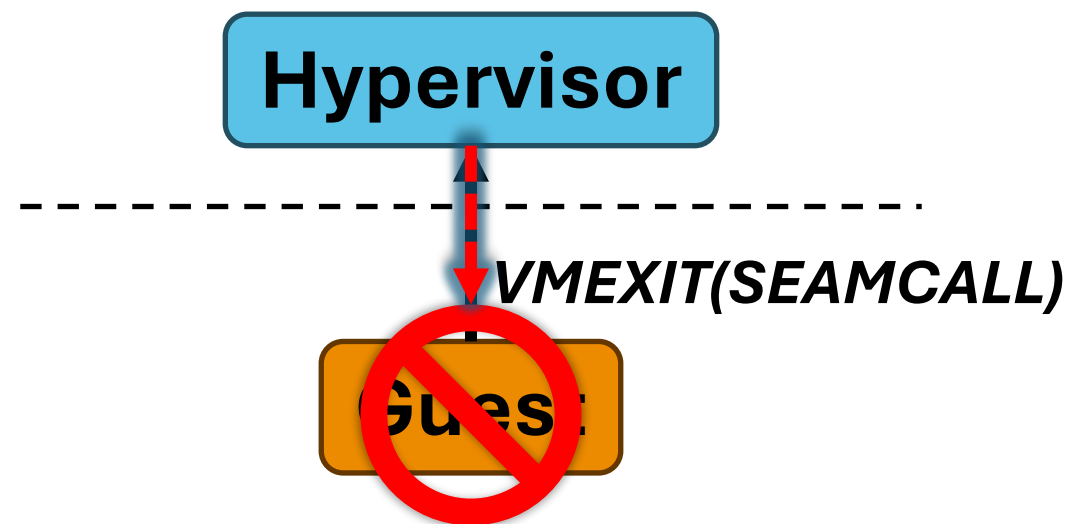
# Unconditionality

- TDX is a new feature, so current hypervisors do not know about it
-  What happens if a guest executes SEAMCALL but the hypervisor doesn't recognize VMEXIT(SEAMCALL)?



# Unconditionality: a problem?

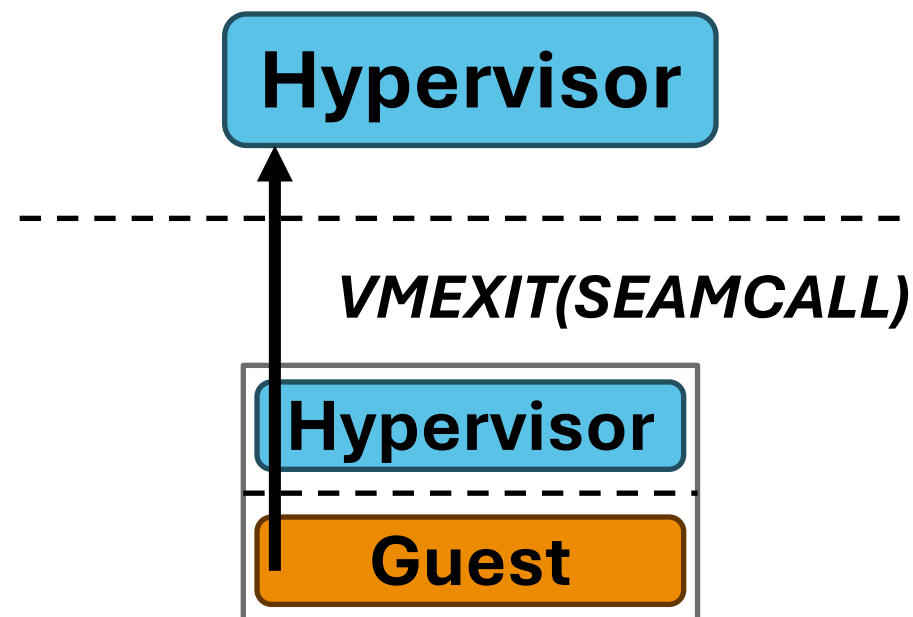
- TDX is a new feature, so current hypervisors do not know about it
-  What happens if a guest executes SEAMCALL but the hypervisor doesn't recognize VMEXIT(SEAMCALL)?
- The hypervisor **kills the guest**, because it doesn't know how to emulate the operation





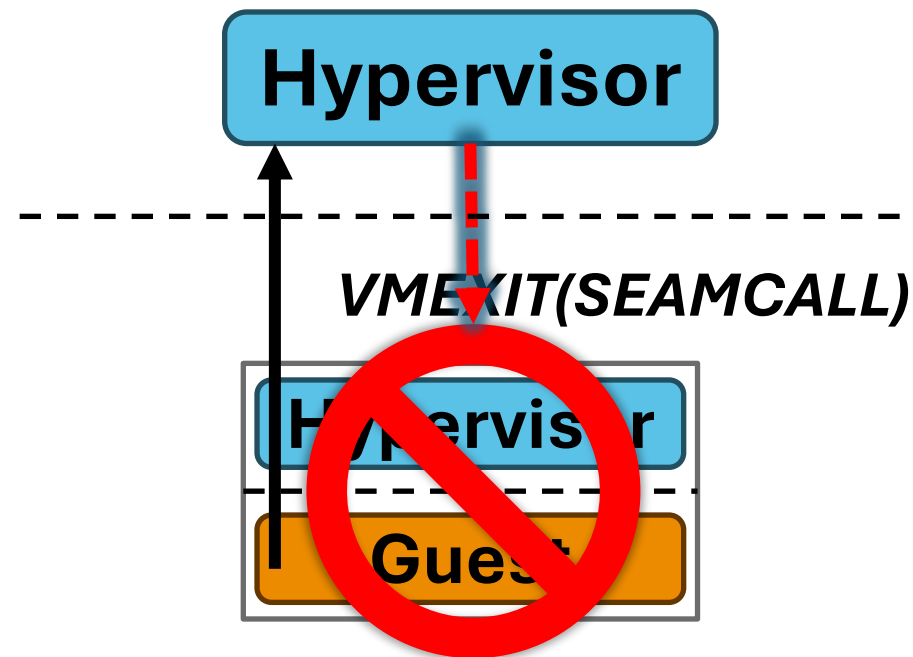
# Nested scenarios

- The guest is itself a hypervisor that runs a guest
- If the nested guest executes **SEAMCALL**, it's the outer hypervisor that handles it



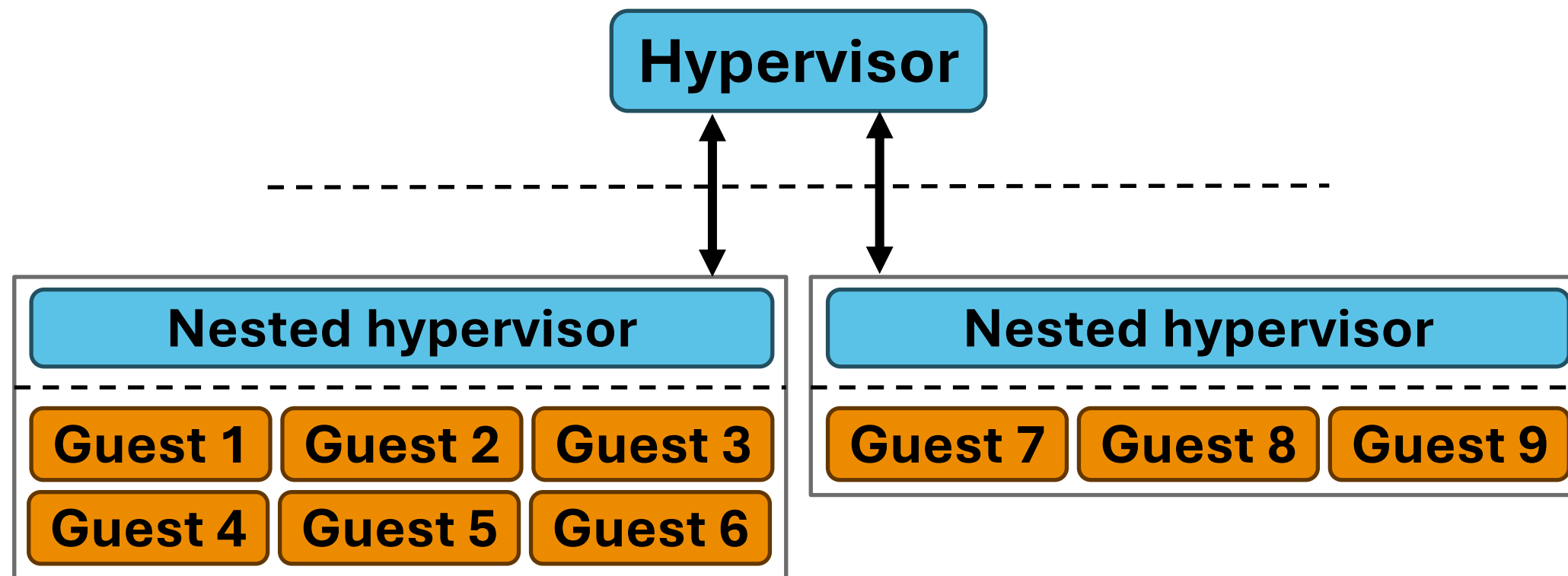
# Nested scenarios

- The guest is itself a hypervisor that runs a guest
- If the nested guest executes **SEAMCALL**, it's the outer hypervisor that handles it
- The outer hypervisor kills the whole guest: its hypervisor and its nested guests



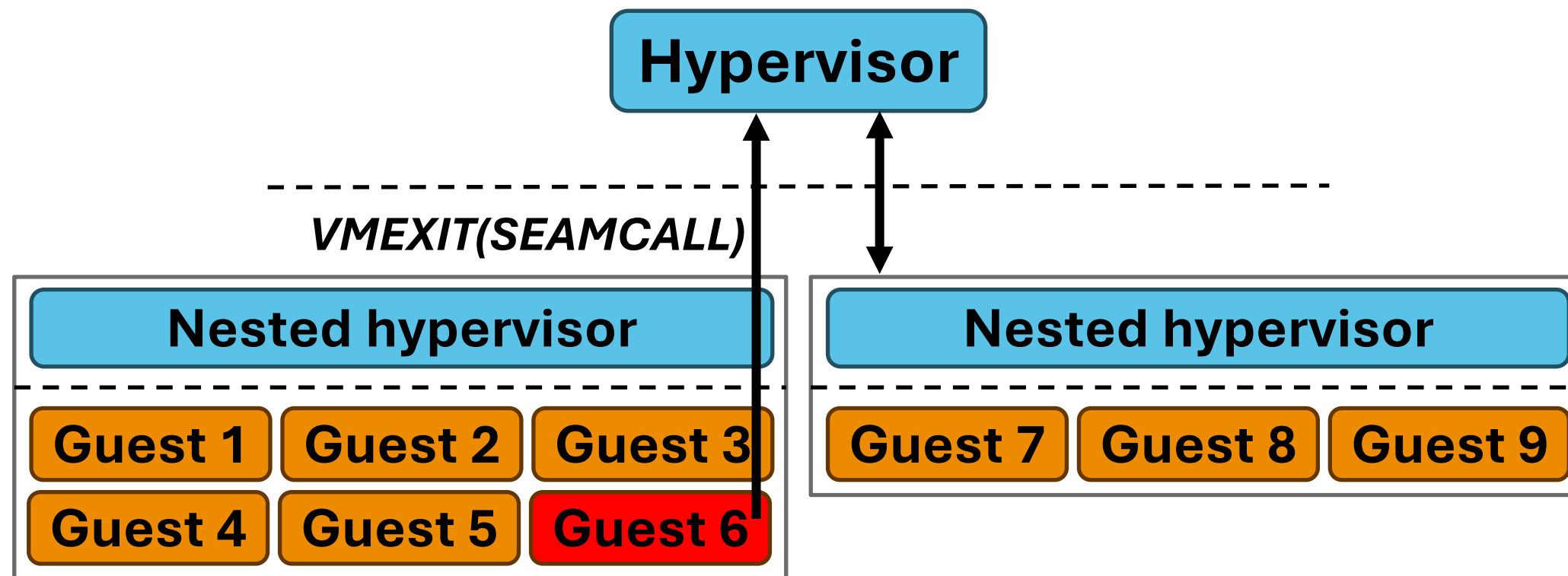
# Nested scenarios in Azure

- Containers run in nested VMs



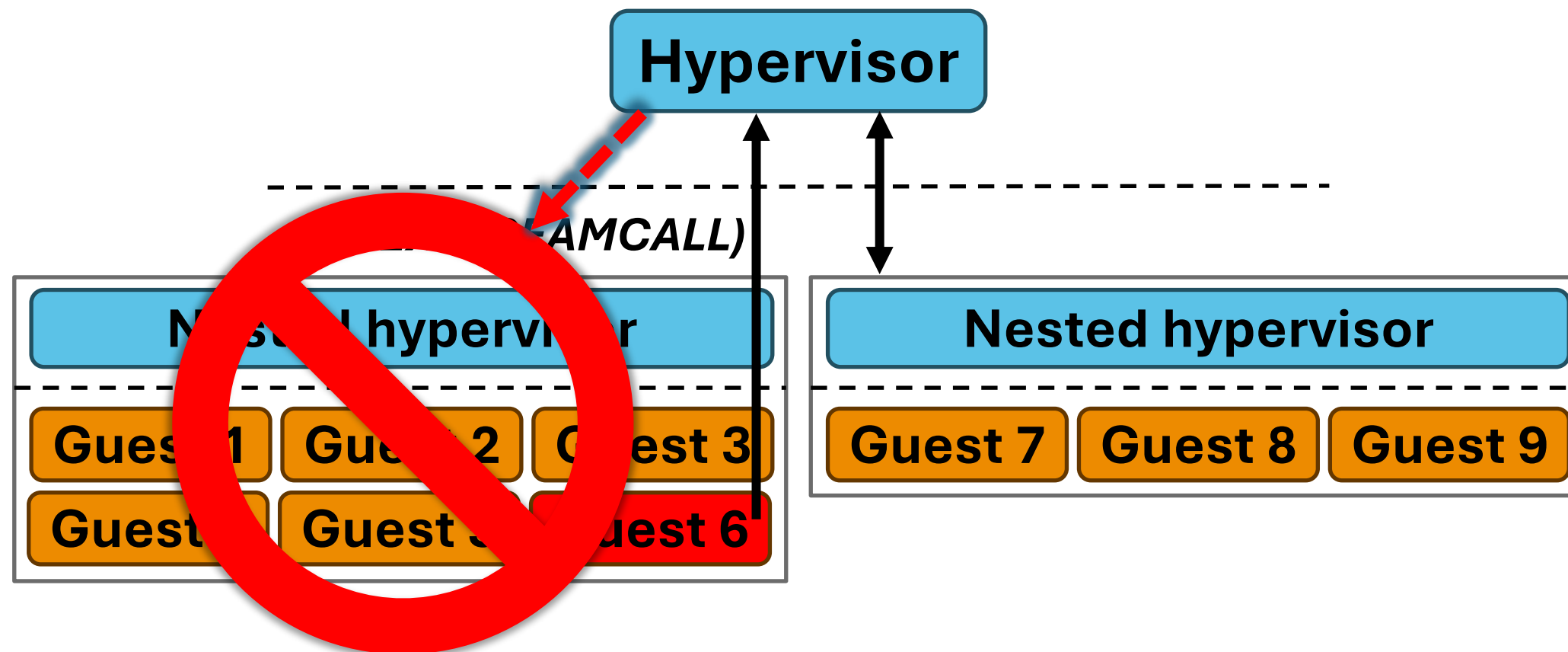
# Nested scenarios: vulnerability

- If a malicious guest executes **SEAMCALL**, all of nested system gets killed



# Nested scenarios: vulnerability

- If a malicious guest executes **SEAMCALL**, all of nested system gets killed
- Ability to DoS other customers by just executing **SEAMCALL**





# An additional bug

- What's more: there's a priority inversion between VMEXIT(SEAMCALL) and the CPL check

## Operation

IF not in VMX operation or inSMM or inSEAM or  $((\text{IA32\_EFER.LMA} \ \& \ \text{CS.L}) == 0)$   
THEN #UD;

ELSIF in VMX non-root operation

THEN VMexit("basic reason" = SEAMCALL,  
"VM exit from VMX root operation" (bit 29) = 0);

ELSIF  $\text{CPL} > 0$  or  $\text{IA32\_SEAMRR\_MASK.VALID} == 0$  or "events blocking by MOV-SS"  
THEN #GP(0);

- The malicious customer doesn't even have to be in kernelmode: they can directly execute SEAMCALL from usermode!

# Affected systems

- Remember: we're talking about the case where the hypervisor doesn't know about TDX
- Future setups where an **old** hypervisor runs on **new** hardware
- Not an unexpected setup in the cloud, legitimate for various reasons

# Affected systems

- Remember: we're talking about the case where the hypervisor doesn't know about TDX
- Future setups where an **old** hypervisor runs on **new** hardware
- Not an unexpected setup in the cloud, legitimate for various reasons
- **CVE-2024-22374**
- Intel fixed half of the vulnerability via a microcode update
- We patched all Hyper-V versions to recognize VMEXIT(SEAMCALL)



## Takeaways





# Whitepaper

## Technical Report of Joint Security Review by Microsoft and Intel on Intel® TDX1.5

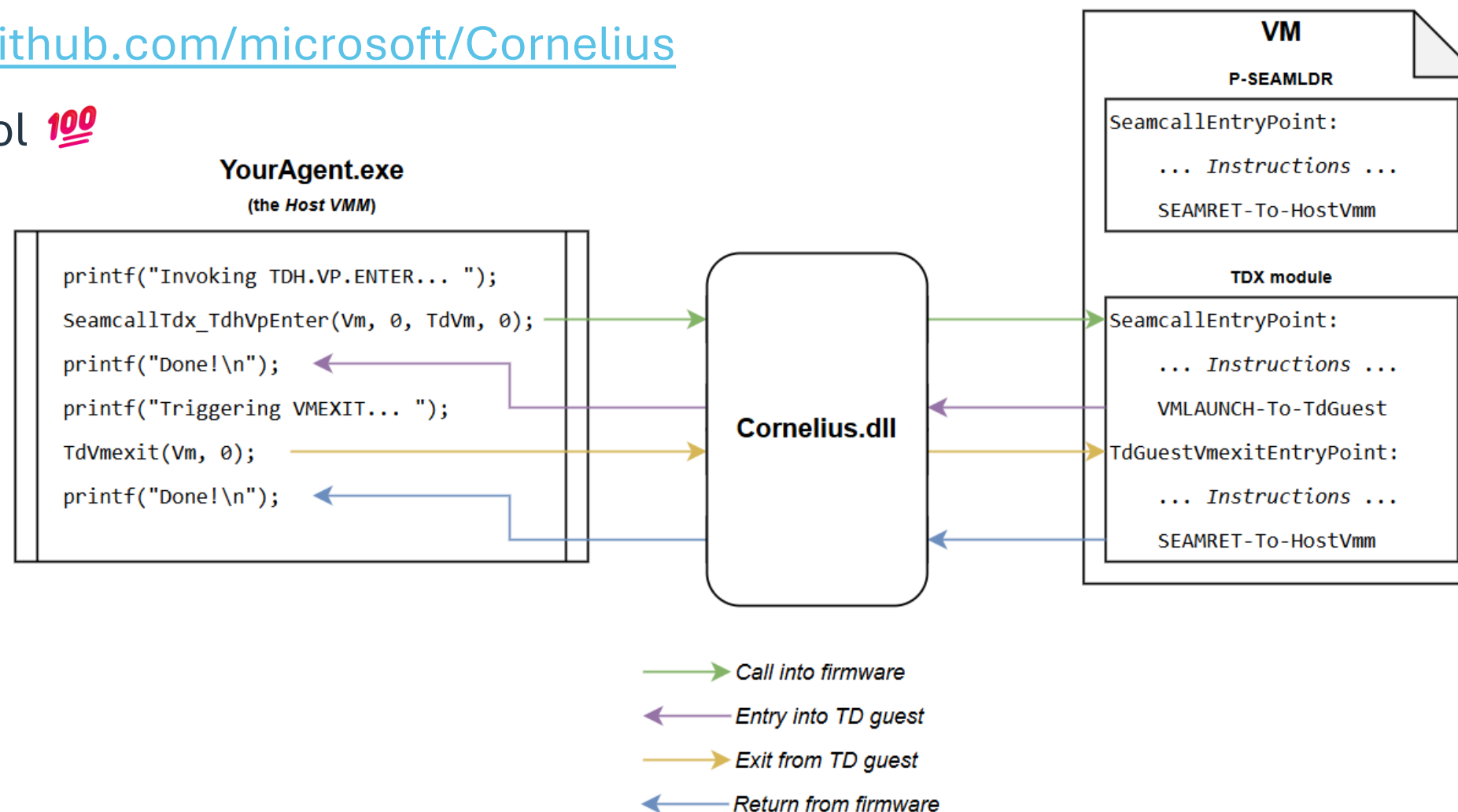
August 2024

- Whitepaper covering our research: [link](#)
- Partnership between Microsoft and Intel
- **21 findings, with 6 confirmed vulnerabilities**
- BlueHat IL 2024: [Compromising Confidential Compute and then fixing it](#) (YouTube)
- Intel Blog Post: [Intel and Microsoft joint security review of Intel TDX 1.5](#)



# Cornelius

- Cornelius is now open-source
- <https://github.com/microsoft/Cornelius>
- Great tool **100**



# TDX: a fun target to look at

- TDX Module source code: <https://github.com/intel/tdx-module>
- Written with security in mind, finding bugs is hard
- Good mitigations
- **Perfect intellectual exercise** 🧠

# Bug bounty

- Intel has a bug bounty program that covers the TDX Module
- Random idea: you can write a fuzzer based on Cornelius, find bugs, report them





# **black hat**<sup>®</sup> USA 2024

Thank you



Microsoft







# **black hat**<sup>®</sup> USA 2024

PS: we're recruiting!

[aka.ms/morsejobs](https://aka.ms/morsejobs)

