

# Ahoi Attacks: Breaking Confidential VM with Malicious Interrupts

Shweta Shinde, ETH Zurich



# Secure & Trustworthy Systems Group

Est. 2020

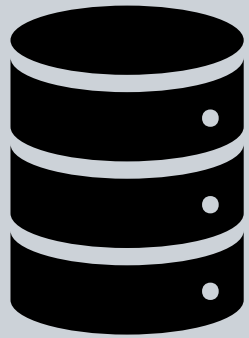
Confidential computing &  
system security

Foundational: how to  
protect phones, servers,  
and accelerators

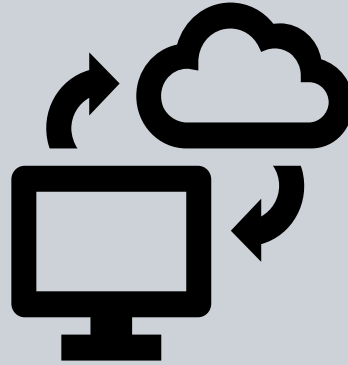
Practical:  
building large systems



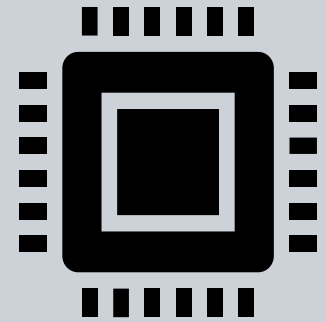
# Attack Surfaces for Confidential Data



At rest



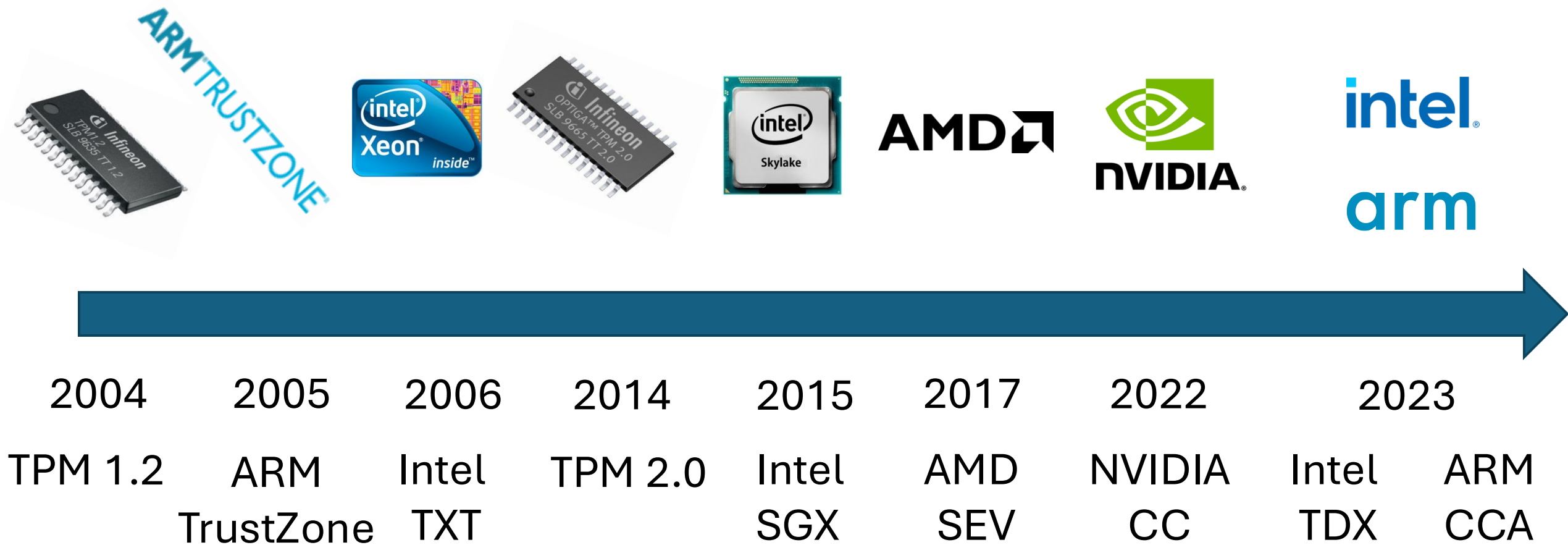
In Transit



In Use



*An incomplete history of the  
Evolution of Trusted Computing*



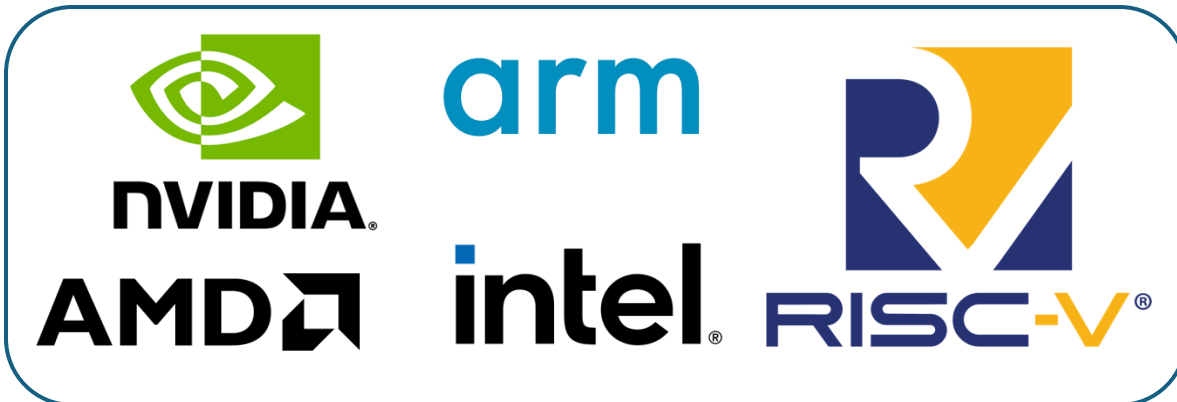
# Trusted Execution Environments: Versatile Principle Applied to Real Systems



Sensitive Apps



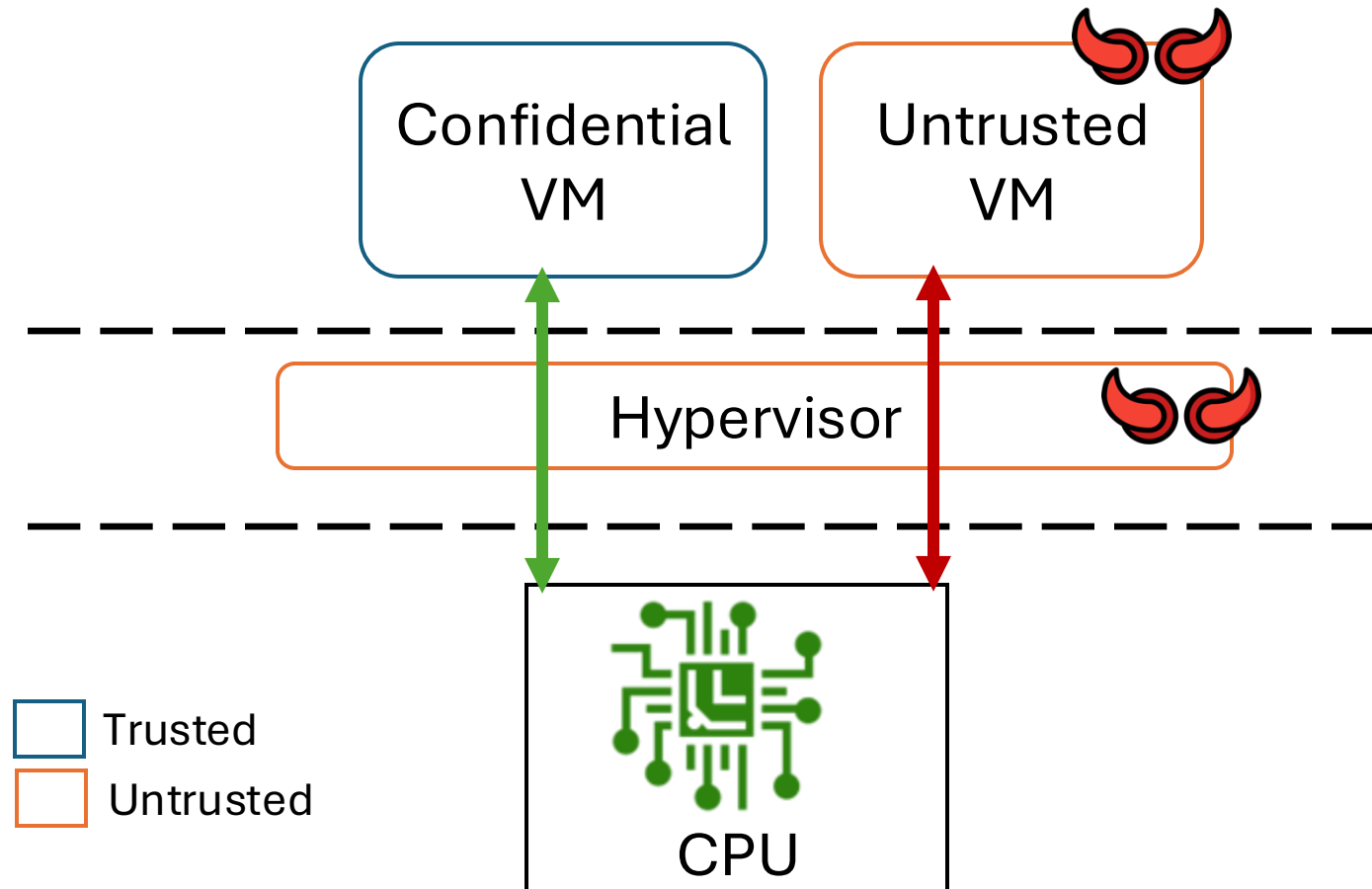
Cloud Providers,  
Operating Systems



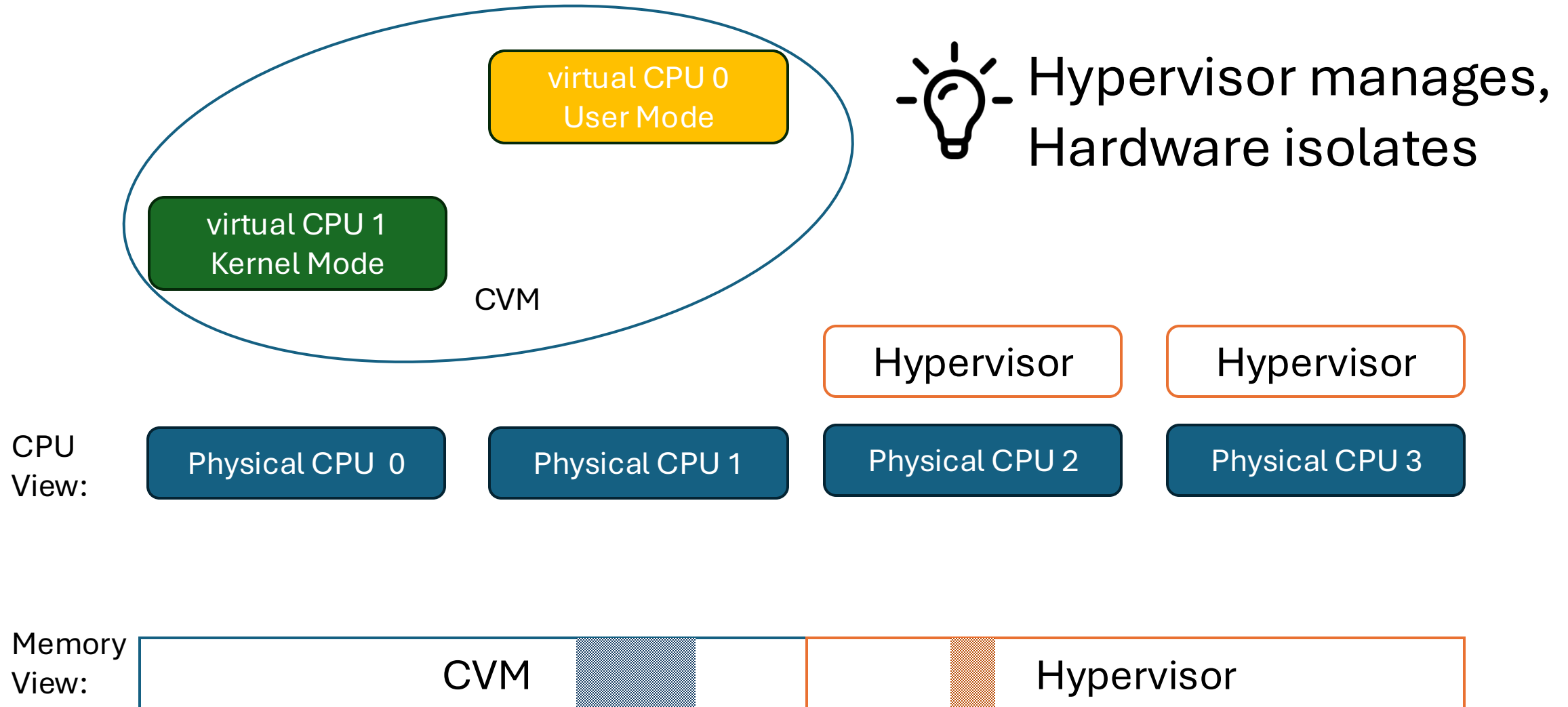
Servers, Mobiles,  
Sensor, GPUs,  
FPGAs, NICs

# Confidential Virtual Machines

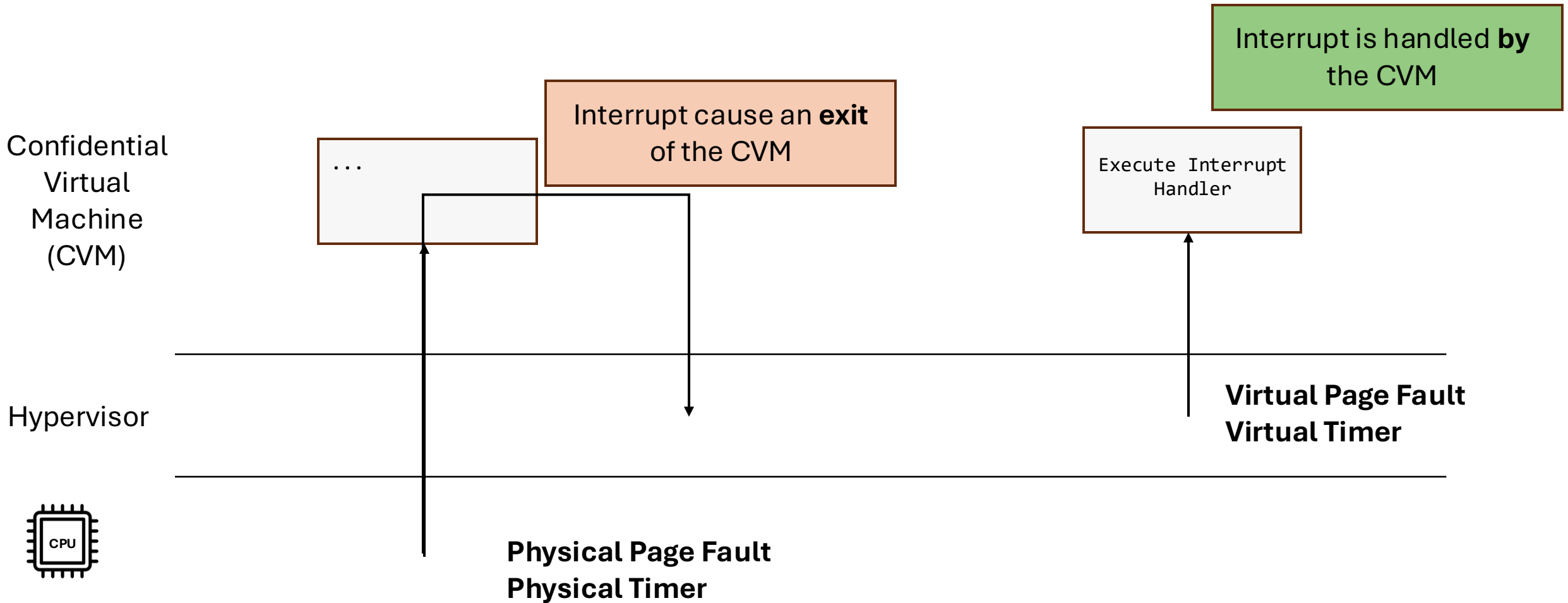
**New Generation CPUs**  
(e.g., AMD SEV-SNP, Intel TDX, ARM CCA)



# Confidential VMs: CPU & Memory Abstraction

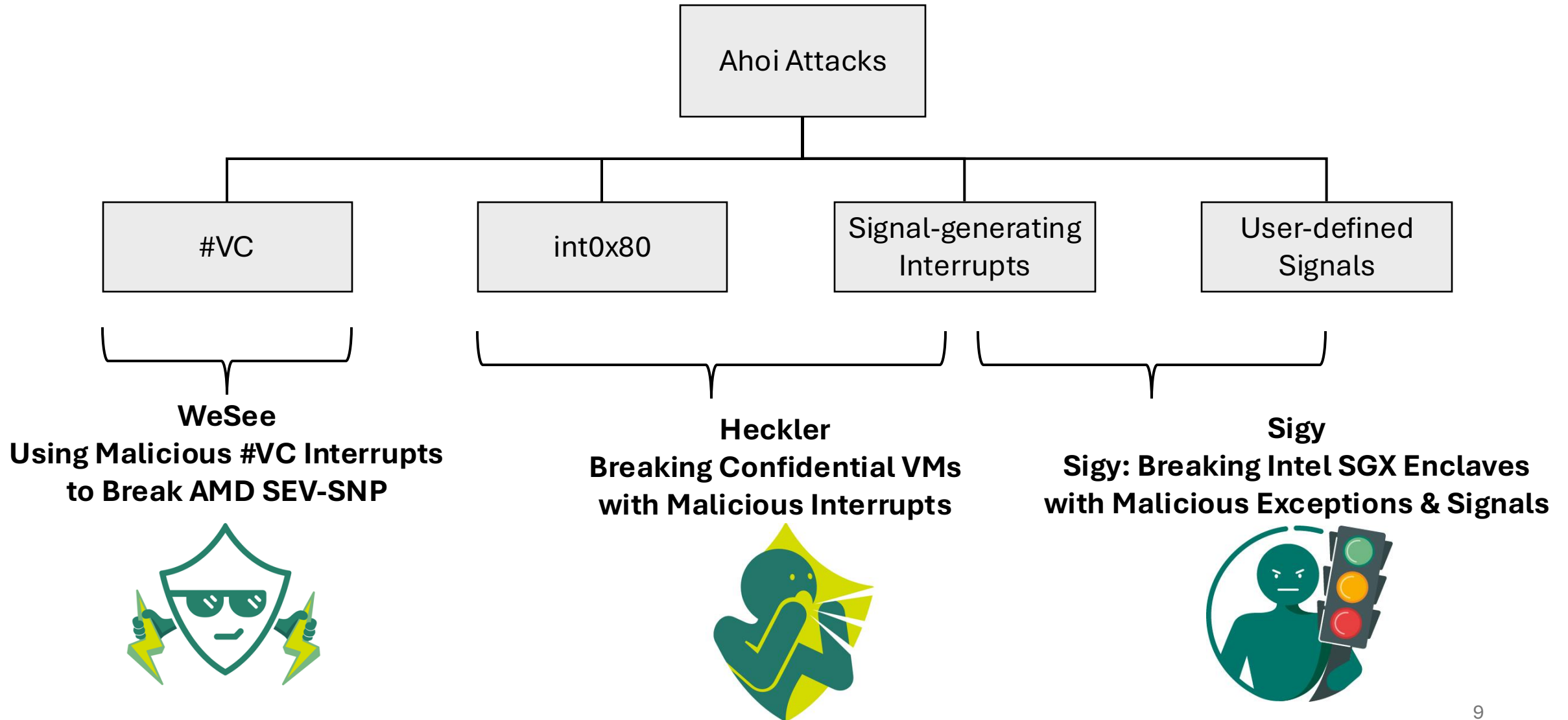


# Destination of Interrupts

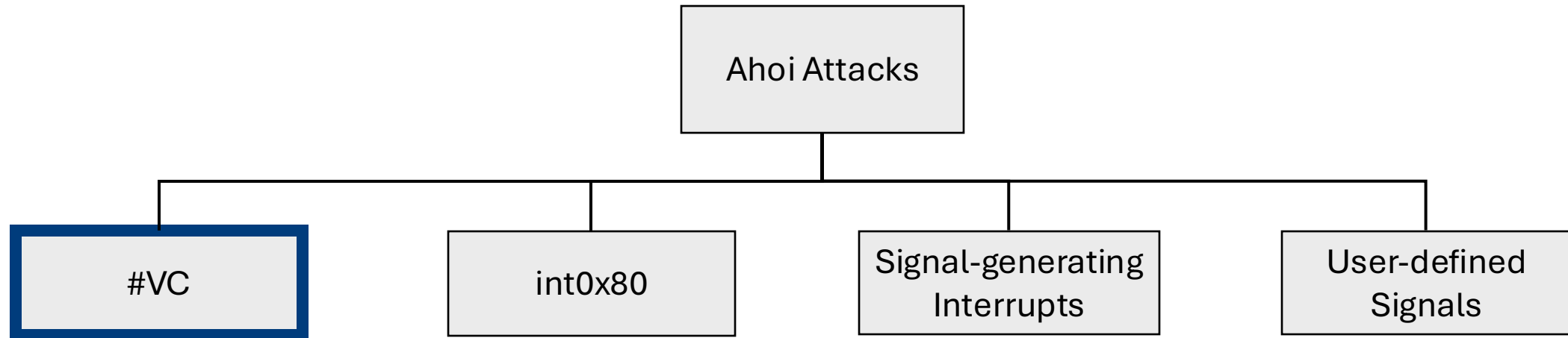




# Ahoi Attacks: Malicious Notifications

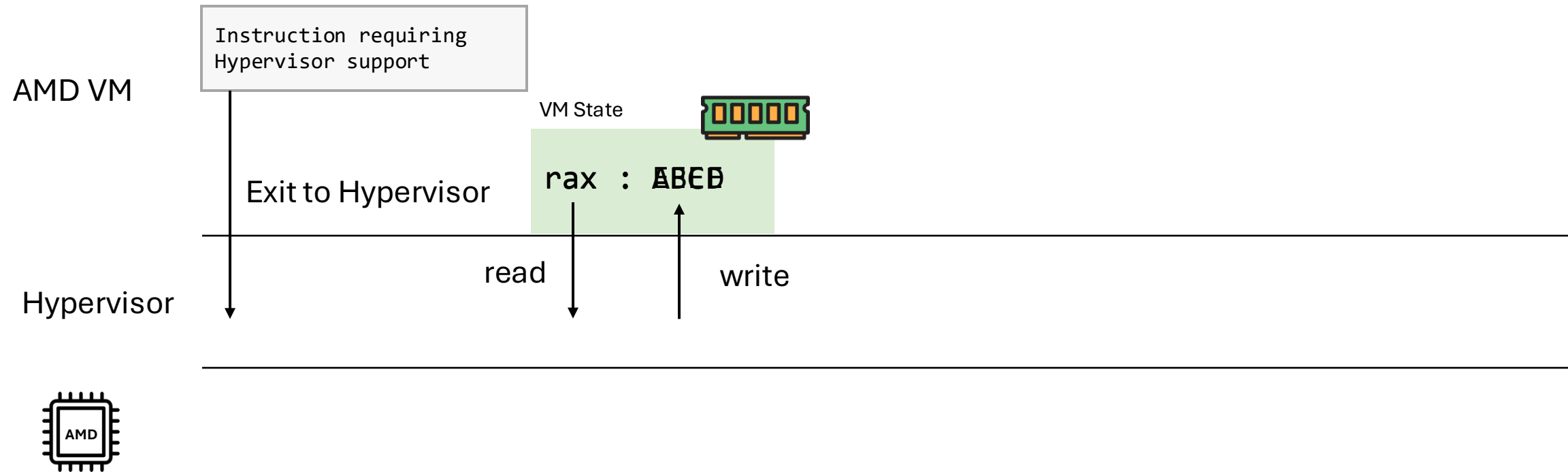


# Ahoi Attacks: Malicious Notifications

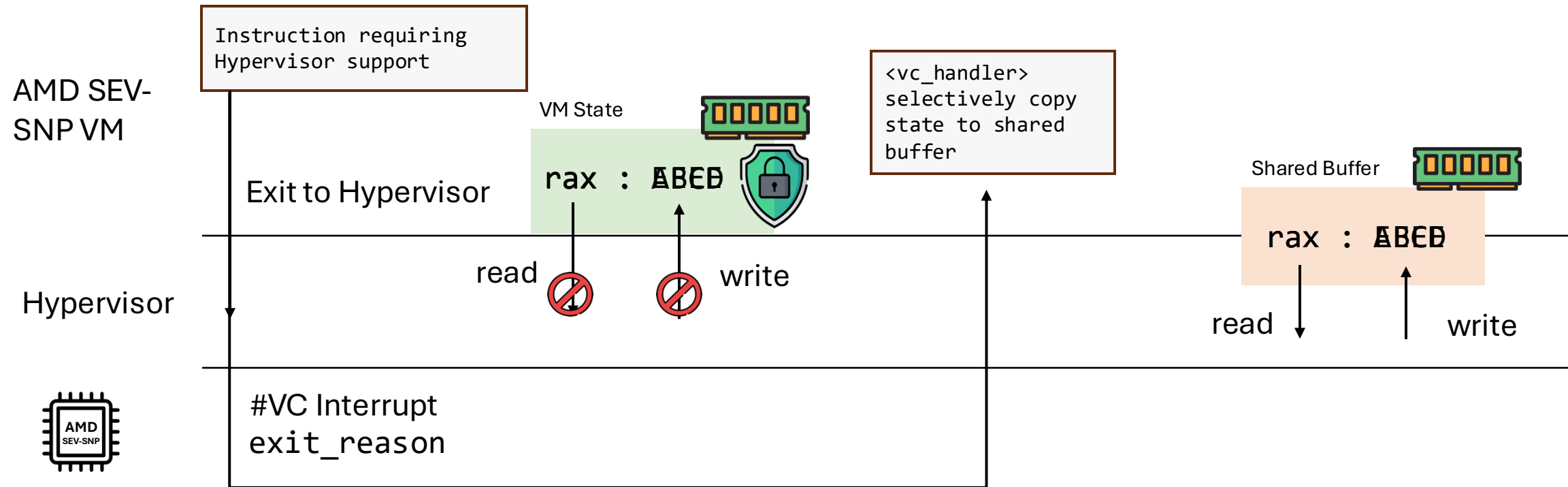


**WeSee**  
**Using Malicious #VC Interrupts**  
**to Break AMD SEV-SNP**  
**(IEEE S&P 2024)**

# AMD Virtualization: Guest – Trusted Hypervisor Communication



# AMD SEV-SNP: Guest – Untrusted Hypervisor Communication



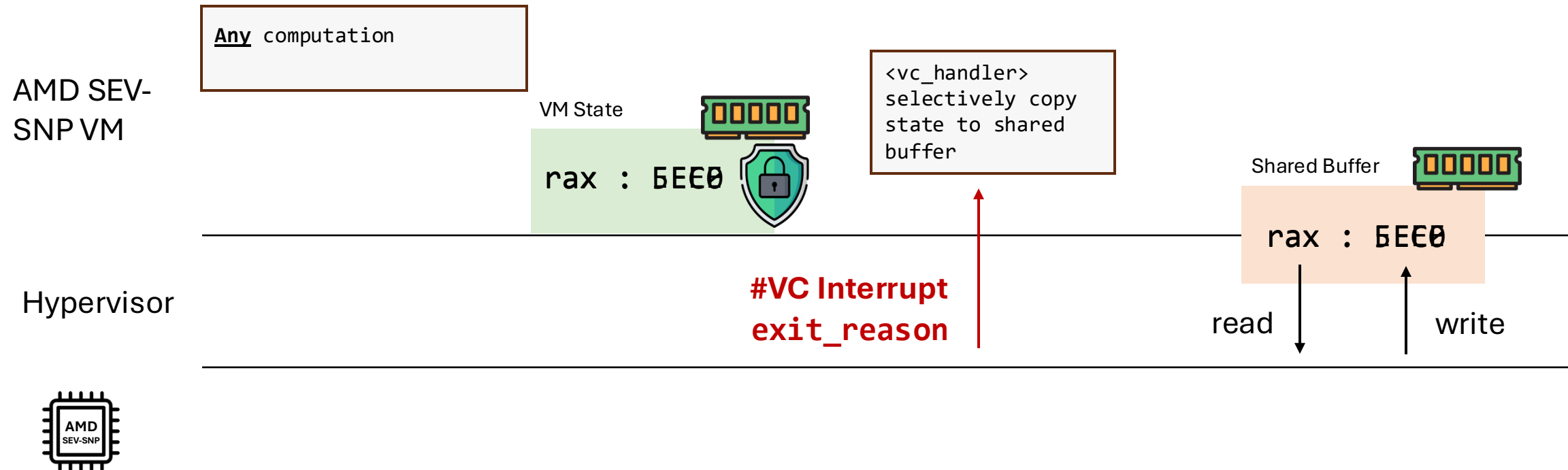
# AMD Virtualization: Guest – Untrusted Hypervisor Communication



# WeSee Attack

Hypervisor can fake a #VC interrupt ***at any time***, the VM will start executing the handler ☹️

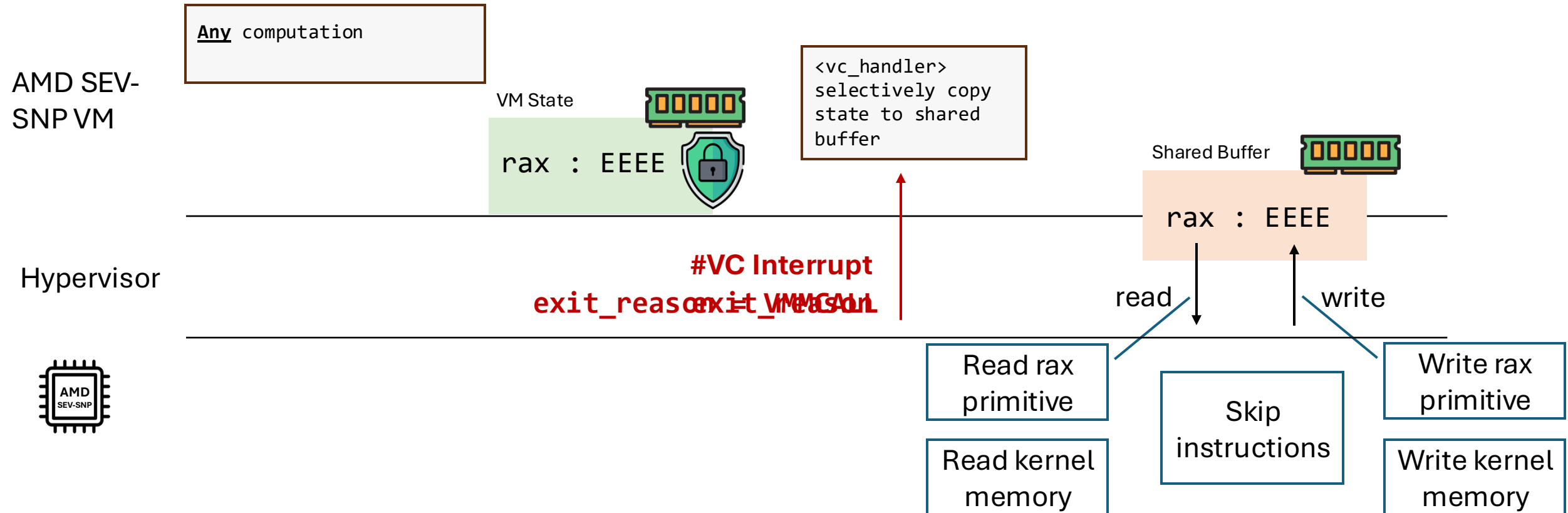
Hypervisor can control the exit\_reason ☹️ ☹️





# WeSee Attack Primitives

Exit Reason	Description	Reg. → Hyp.	Hyp. → Reg.	Sample Inst.
VMMCALL	Call to Hypervisor	rax	rax	
MMIO WRITE	Memory Mapped I/O write	#reg	-	mov rbx,[rax]
MMIO READ	Memory Mapped I/O read	-	#reg	mov [rax],rbx



# WeSee Attack : Write kernel memory

## 1. control a pointer

## 2. dereference the pointer and write to kernel

1. #VC MMIO read
2. write rax
3. write rax
4. skip instructions

Total #VC interrupts = 34

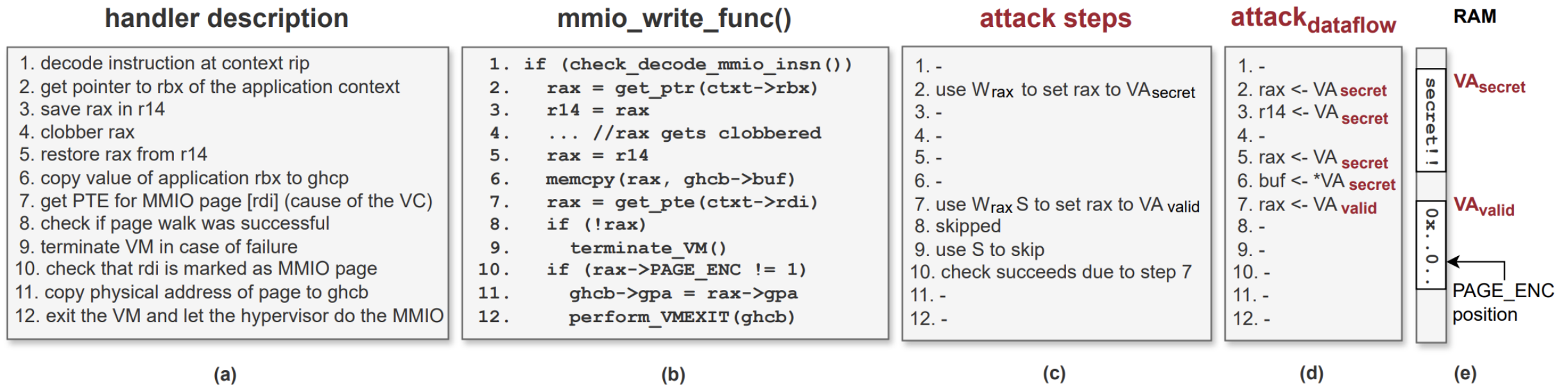


Figure 5. Read memory primitive. (a) Handler description for #VC caused by instruction `mov [rdi], rbx`. (b) Pseudo-code of MMIO write handling. (c) Attack steps. (d) Attack dataflow. (e) Memory used for attack.

# WeSee Case Studies



## Leak KernelTLS keys

*Read memory 8 times*  
**288 #VCs**

Session key stored in kernel memory



## Disable iptables Firewall

Replace call to **filter function**  
to simply return **true**  
*Read memory 4 times*  
*Write memory 5 times*  
**238 #VCs**

iptables rules to filter network traffic in the kernel

```
1 endbr64  
2 mov $1, rax  
3 ret
```

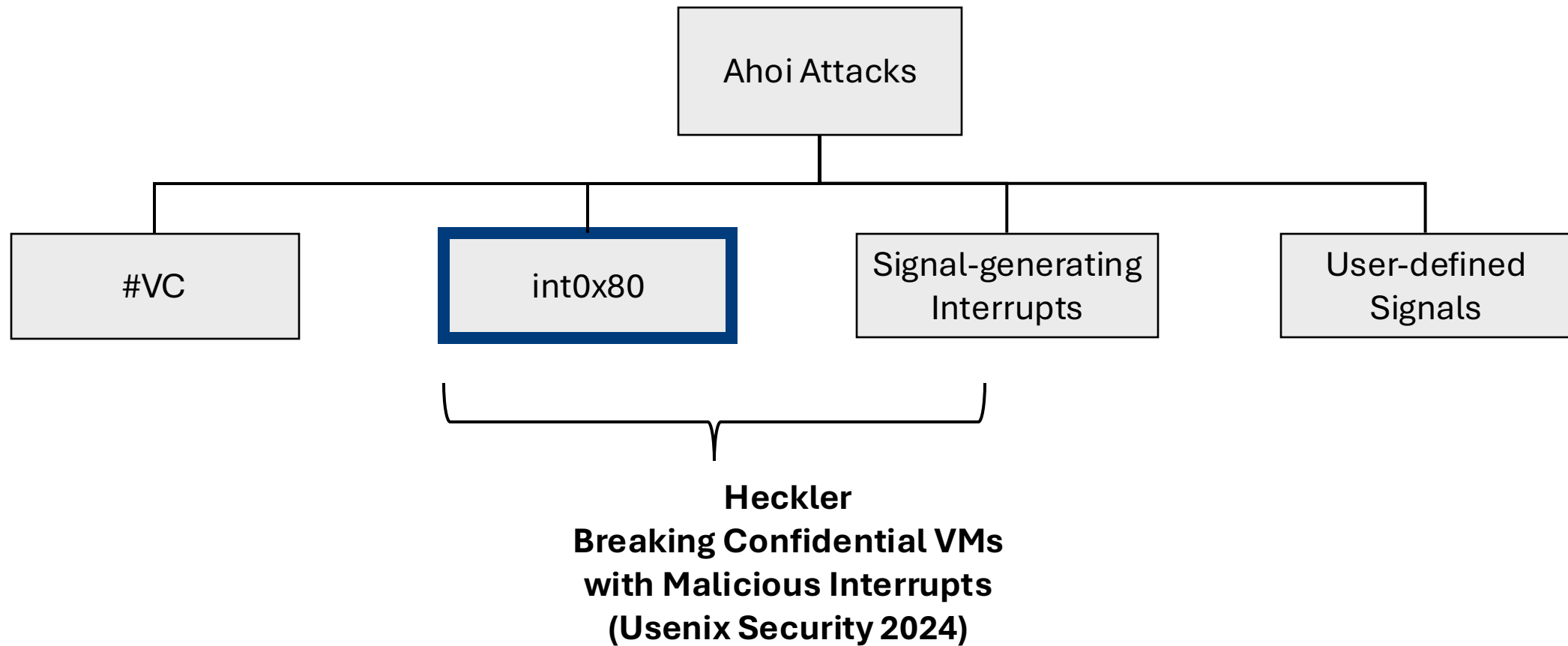


## Root Shell

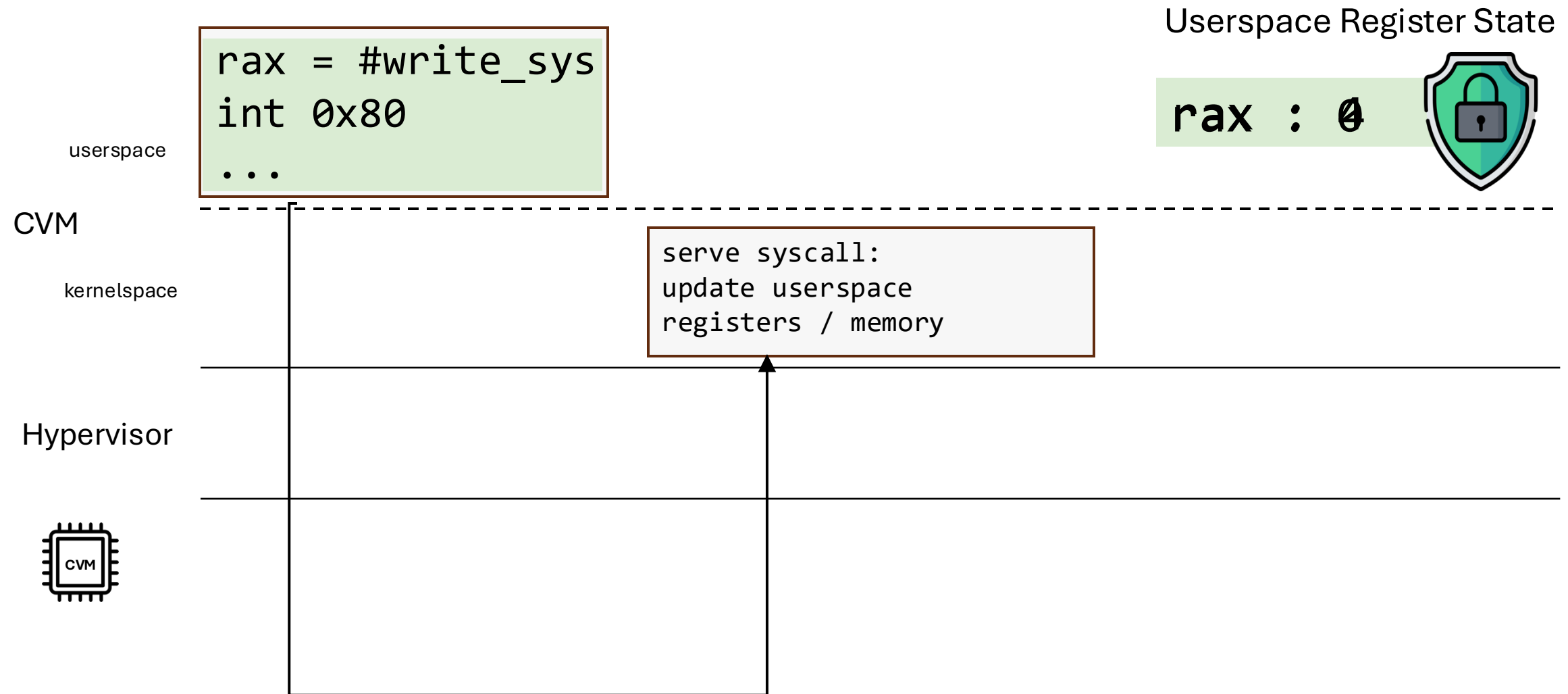
*Read memory 4 times*  
*Write memory 5 times*  
**2891 #VCs**

Code pages in the kernel

# Ahoi Attacks: Malicious Notifications



## int 0x80: Legacy Systemcall Flow



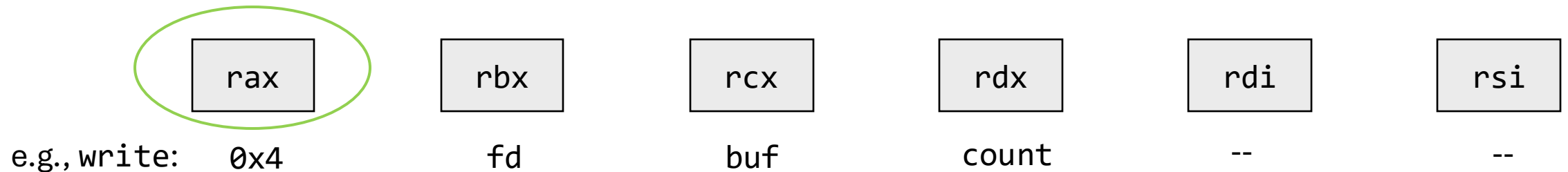
## int 0x80 Primitive

Kernel  
space

```
handle_int80:  
    /* read rax for #n */  
    call syscall_#n()  
    /* store result in rax */
```

Interesting values for #n?

- [0,384] translates to valid syscall
- [385,2<sup>64</sup>-1] will return an error  
→ rax changes to error value



Restart syscall

**$\text{rax} = 0 \rightarrow \text{rax} = -4$**



# Remote Authentication

## Victim

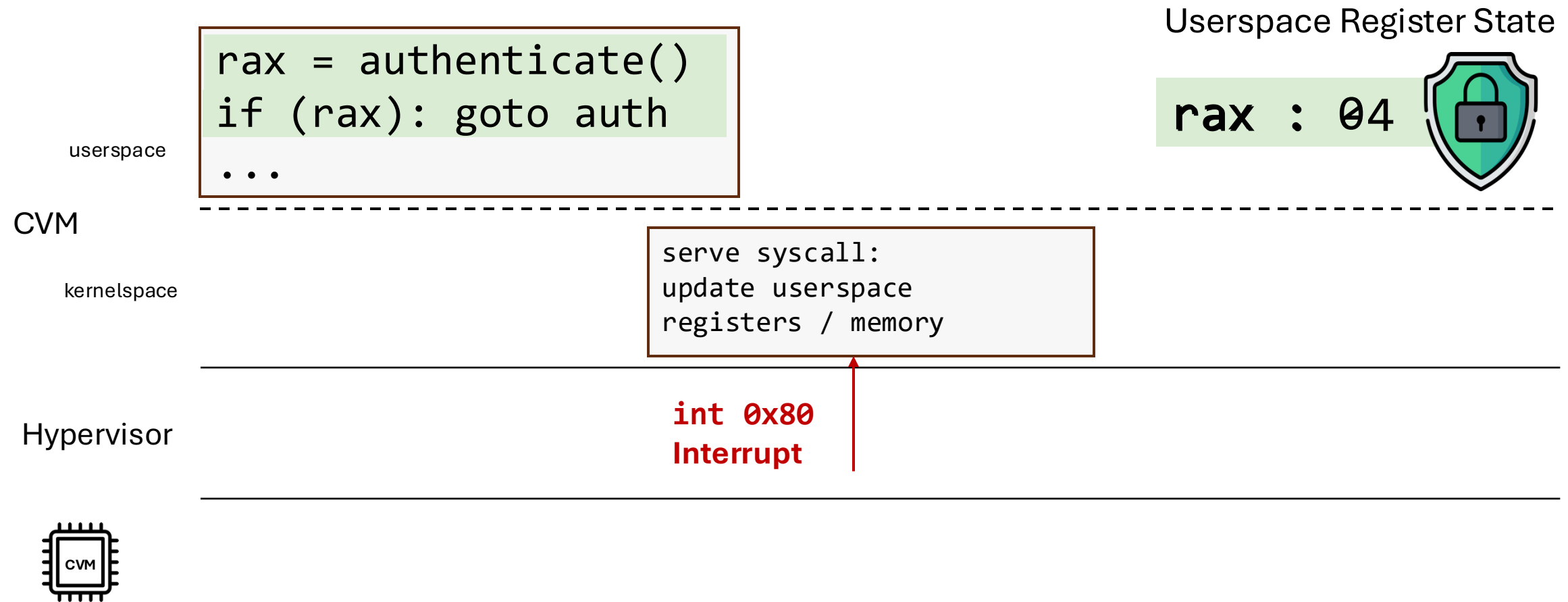
```
rax = authenticate()  
// Branch based on rax
```

OpenSSH

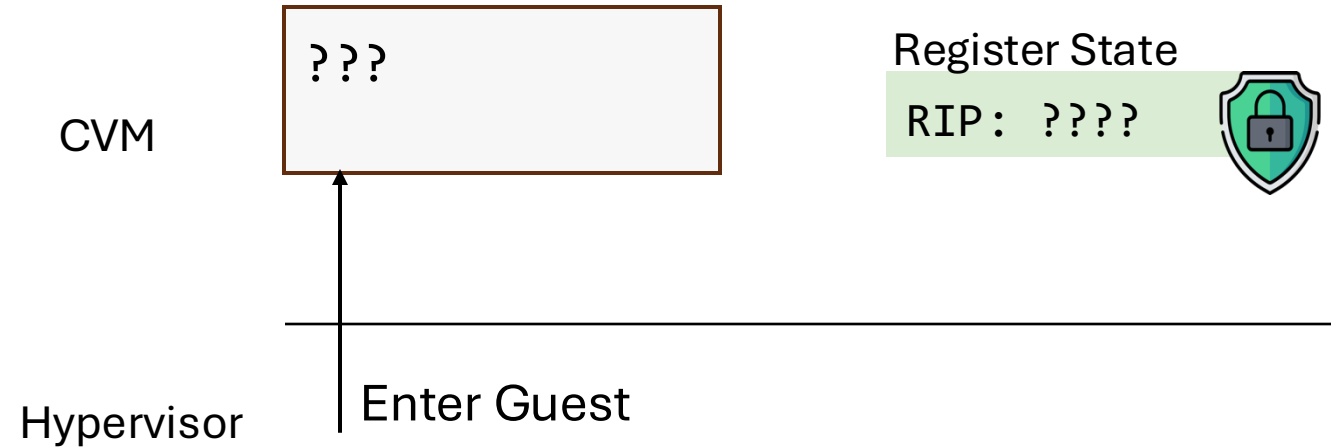
Interrupt

Hypervisor

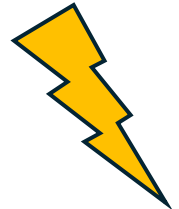
# Heckler Attack (Intel TDX and AMD SEV-SNP)



# Hypervisor View on CVM Execution State



```
rax = authenticate()  
if (rax): goto auth  
...
```



## Tracking the page-level execution state of CVMs

```
rax = authenticate()  
if (rax): goto auth  
...
```

Page A

```
call authenticate  
test eax,eax  
je auth  
...
```

Page B

```
do authentication  
...  
ret
```

Hypervisor  
observes page  
faults

Page A  
Page B  
Page A

**Inject interrupt before marking  
Page A as executable**

## Case Studies

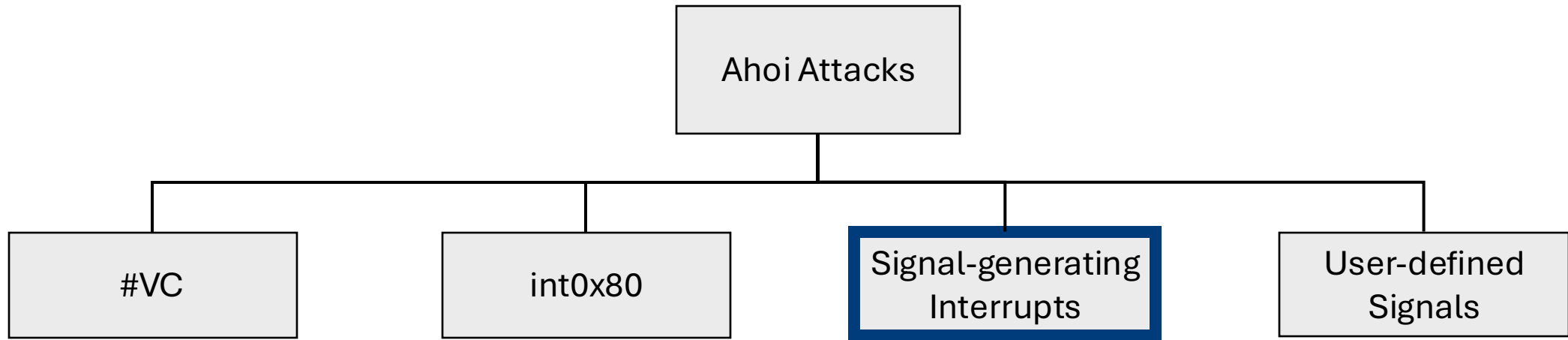


*Bypass OpenSSH's password authentication*



*Bypass Sudo authentication through PAM*

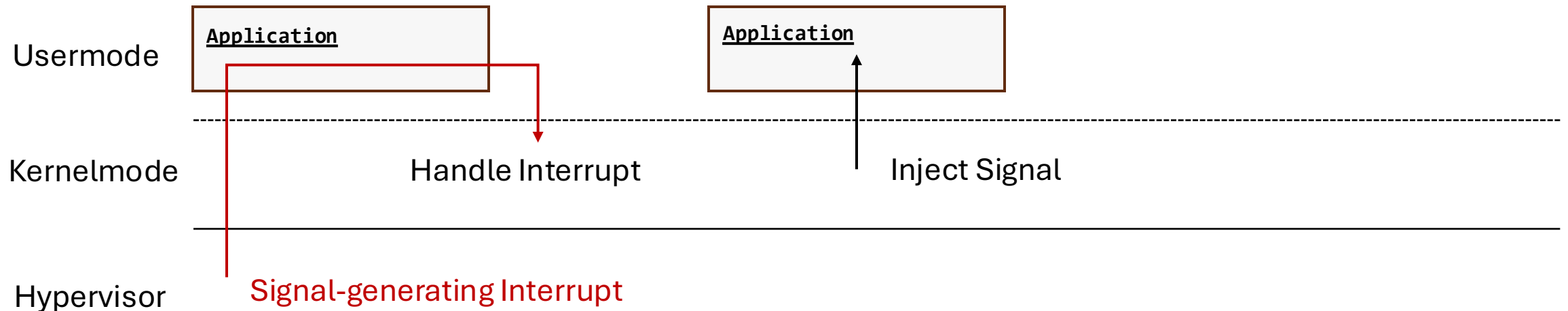
## Ahoi Attacks: Malicious Notifications





## Interrupt to Signal Conversion

- SIGFPE (0, 14, 16)
- SIGTRAP (1), SIGILL (6), SIGSEGV (4, 5, 10), and SIGBUS (11, 12, 17, 29)



## Signal Generating Interrupts (AMD SEV-SNP)

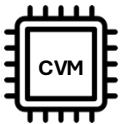
```
try {  
    v = Covariance(...);  
} catch(ArithmeticException ex) {  
    v = 0;  
}
```

CVM

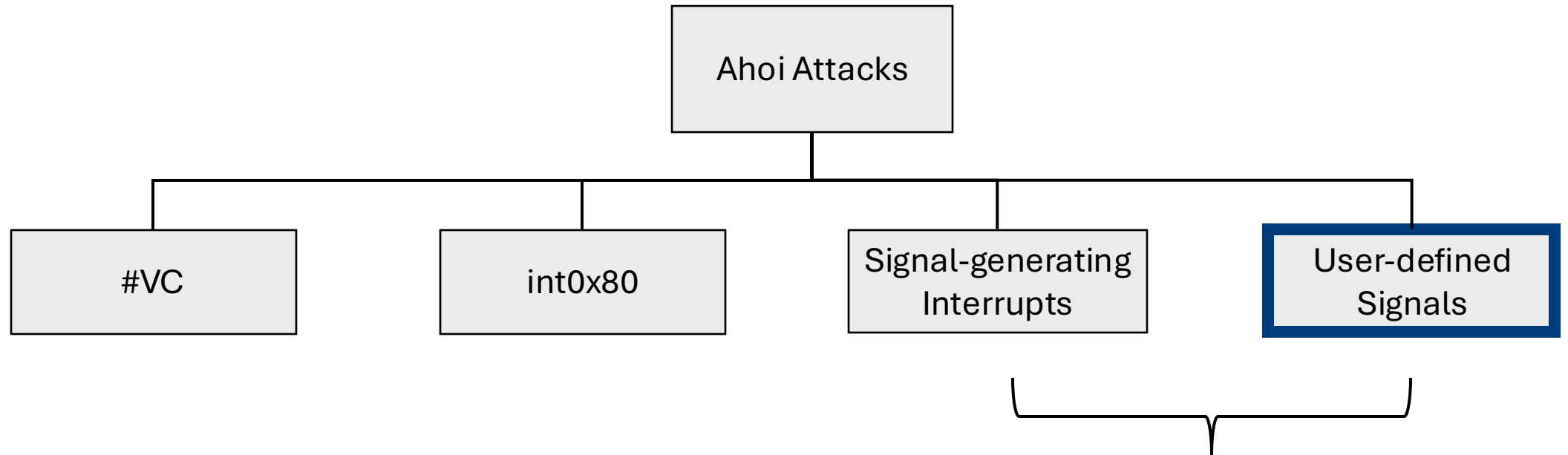
serve interrupt:  
Inject signal to  
userspace

Hypervisor

**int 0x10  
Interrupt**



# Ahoi Attacks: Malicious Notifications



**Sigy**

**Sigy: Breaking Intel SGX Enclaves  
with Malicious Exceptions & Signals  
(AsiaCCS 2025)**

## Signal Generating Interrupts (Intel SGX)

```
try {  
    v = Covariance(...);  
} catch(ArithmeticException ex) {  
    v = 0;  
}
```

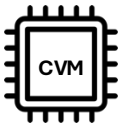
Any signal  
(sigusr1, sighup, sigfpe)

serve interrupt:  
Inject signal to  
userspace

Enclave

OS

Hypervisor



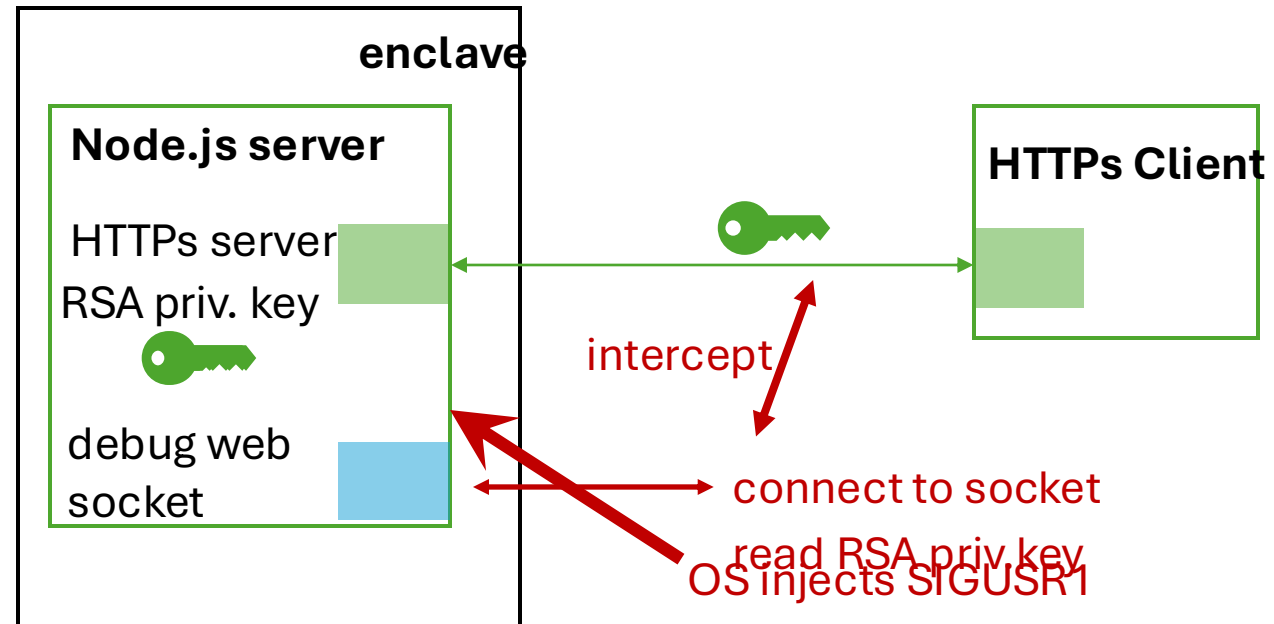
# Java

```
public boolean setData(List dataSet) {  
    Vec origMean = this.mean;  
    try {  
        //can overflow  
        Vec newMean = meanVector(dataSet);  
        Matrix covariance = covarianceMatrix(newMean,dataSet);  
        this.mean = newMean;  
        setCovariance(covariance);  
    } catch(ArithmeticException ex) {  
        this.mean = origMean;  
    }  
}
```

instruction  
pointer →

OS injects SIGFPE

# Node.js Server





## Sigy: Sending Malicious Signals to Intel SGX Enclaves

- Hardware exceptions converted to signals: SIGFPE
  - User-defined signals: SIGUSR1, SIGHUP
  - When the OS fakes notifications, enclave automatically executes handlers
- 7 runtimes and library OSes are vulnerable
  - OpenEnclave, Gramine, Scone, Asylo, Teaclave, Occlum, EnclaveOS
- Exploits
  - Data analytics (C and Java) to break integrity
  - Webservers (Nginx, Node.js) to leak secrets

## A short backstory: How did you ‘uncover’ these attacks?

We were looking at the SGX exception handling and broke a few runtimes



But... we didn't reason about directly injecting fake interrupts

A short backstory: How did you ‘uncover’ these attacks?

Next, we were building a new smartphone

## It's TEEtime: A New Architecture Bringing Sovereignty to Smartphones

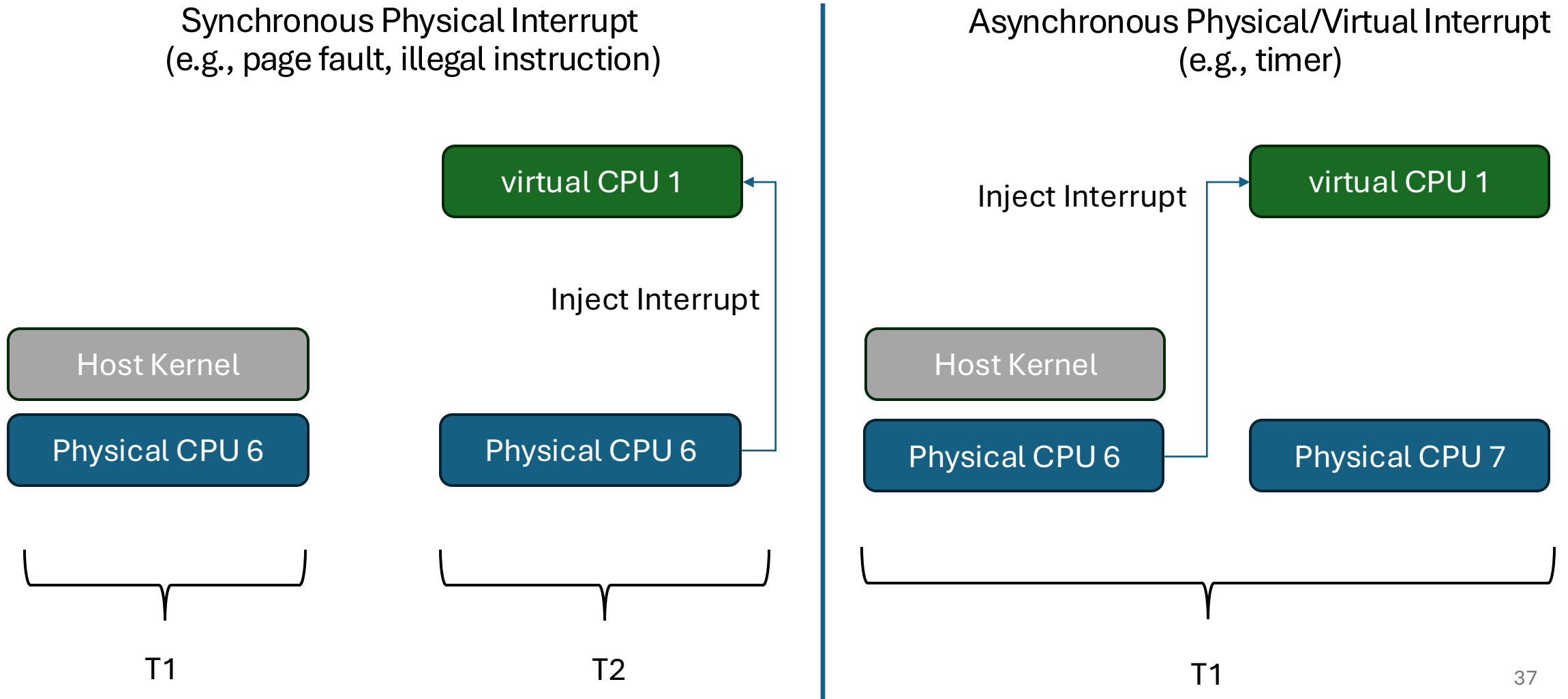
Friederike Groschupp Mark Kuhne Moritz Schneider Ivan Puddu Shweta Shinde Srdjan Capkun  
ETH Zurich  
firstname.lastname@inf.ethz.ch

We wanted secure device interrupts on Arm ... for device isolation  
which prompted us to look at Intel/AMD ...  
and rest is history

# Mitigations

## But First, Root Cause Analysis

Hypervisor can inject interrupts that should never be generated by it





CVE 2024-25742

author Borislav Petkov (AMD) <bp@alien8.de> 2024-01-05 11:14:07 +0100  
 committer Borislav Petkov (AMD) <bp@alien8.de> 2024-01-29 17:08:22 +0100  
 commit e3ef461af35a8c74f2f4ce6616491ddb355a208f (patch)  
 tree 105ccfe4c6c6774dfde29111f36ae23427f69227  
 parent 41bcc98fb7931d63d03f326a746ac4d429c1dd3 (diff)  
 download linux-e3ef461af35a8c74f2f4ce6616491ddb355a208f.tar.gz

**x86/sev: Harden #VC instruction emulation somewhat**

Compare the opcode bytes at rIP for each #VC exit reason to verify the instruction which raised the #VC exception is actually the right one.

Signed-off-by: Borislav Petkov (AMD) <bp@alien8.de>

Acked-by: Tom Lendacky <thomas.lendacky@amd.com>

Link: <https://lore.kernel.org/r/20240105101407.11694-1-bp@alien8.de>

# WeSee (#VC interrupt) Mitigations

HotFix Upstream for our case studies (v6.9)

- compares **last instruction** in application with the **#VC exit\_reason**
- This is not sufficient to stop all malicious #VC injections

MMIO read  
mov rax,[rdi]

AMD SEV-  
SNP VM

Hypervisor

<vc\_handler>  
selectively copy  
state to shared  
buffer



<patch>  
Instruction == VMCALL && exit\_reason == VMCALL:  
Return OK  
...  
Return ERROR  
**Unrecoverable Kernel Panic**

#VC Interrupt   
exit\_reason = VMCALL

# Heckler (Int0x80 + Signals) Mitigations



CVE 2024-25743

HotFix Upstream to disable int0x80 syscall interrupt on CVMs (v6.7)

- Subsequent patch enables int0x80 **only** for TDX again and perform validation of interrupt source
- On SEV-SNP the guest cannot distinguish between interrupt sources

Signal generating interrupt sources remain unfixed until today (only SEV-SNP)



CVE 2024-25744

```
author      Kirill A. Shutemov <kirill.shutemov@linux.intel.com> 2023-12-04 11:31:38 +0300
committer   Dave Hansen <dave.hansen@linux.intel.com>      2023-12-07 09:51:10 -0800
commit      b82a8dbd3d2f4563156f7150c6f2ecab6e960b30 (patch)
tree        4c87ffd52af476e9ed34ddac4ff3f8ec83c32a27
parent      33cc938e65a98f1d29d0a18403dbbee050dcad9a (diff)
download    linux-b82a8dbd3d2f4563156f7150c6f2ecab6e960b30.tar.gz
```

## x86/coco: Disable 32-bit emulation by default on TDX and SEV


The INT 0x80 instruction is used for 32-bit x86 Linux syscalls. The kernel expects to receive a software interrupt as a result of the INT 0x80 instruction. However, an external interrupt on the same vector triggers the same handler.

The kernel interprets an external interrupt on vector 0x80 as a 32-bit system call that came from userspace.

A VMM can inject external interrupts on any arbitrary vector at any time. This remains true even for TDX and SEV guests where the VMM is untrusted.

# Mitigations using Interrupt Modes on AMD SEV-SNP

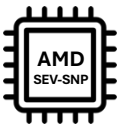
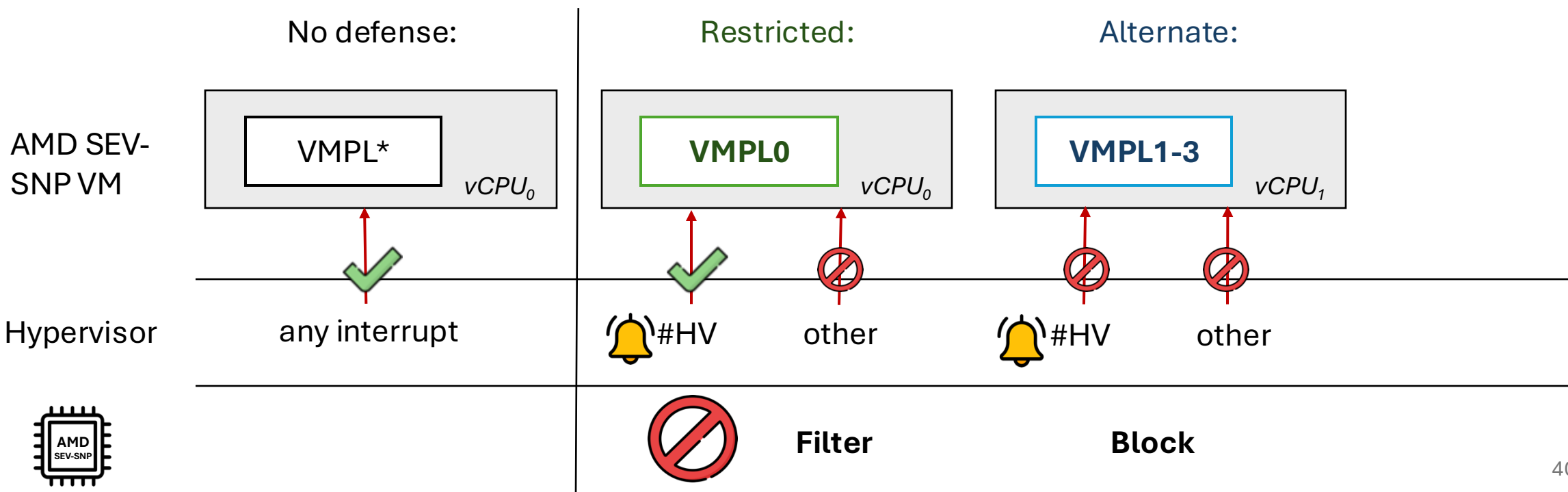
## AMD **Alternate** and **Restricted** Mode

 New #HV (28) Doorbell Interrupt

Filter interrupt injections

**VMPL 1-3**  
(Linux guest)

**VMPL 0**  
(Trusted Bridge)





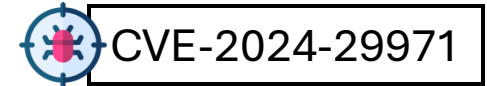
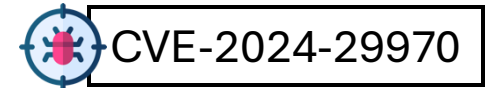
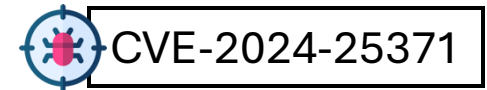
## Sigy Mitigations

Gramine has mitigated this issue with a patch

Other vendors are taking steps to fix it

In general:

- Re-assess handlers
- Filter signals per app
- But some apps need signals (e.g., sigfpe) that can be faked



# Summary

## Ahoi attacks: New class of attacks

- Malicious Interrupt Injection to trigger handlers
- WeSee (IEEE S&P 2024, Distinguished Paper)
- Heckler (Usenix Security 2024 )
- Sigy (AsiaCCS 2025)



<https://ahoi-attacks.github.io/>

## Mitigations

- (Temporary) patches to guest Linux
- New interrupt injection modes
- Interrupt Isolation

**DEVLORE: Extending Arm CCA to Integrated Devices**  
**A Journey Beyond Memory to Interrupt Isolation**

Andrin Bertschi\*    Supraja Sridhara\*    Friederike Groschupp    Mark Kuhne  
Benedict Schlüter    Clément Thorens    Nicolas Dutly    Srdjan Capkun    Shweta Shinde  
*ETH Zurich*

<https://arxiv.org/abs/2408.05835>



Thank you

<https://shwetashinde.org/>  
shweta.shinde@inf.ethz.ch