

  
**black hat**<sup>®</sup>

USA 2021

AUGUST 4-5, 2021

BRIEFINGS

# Alcatraz:

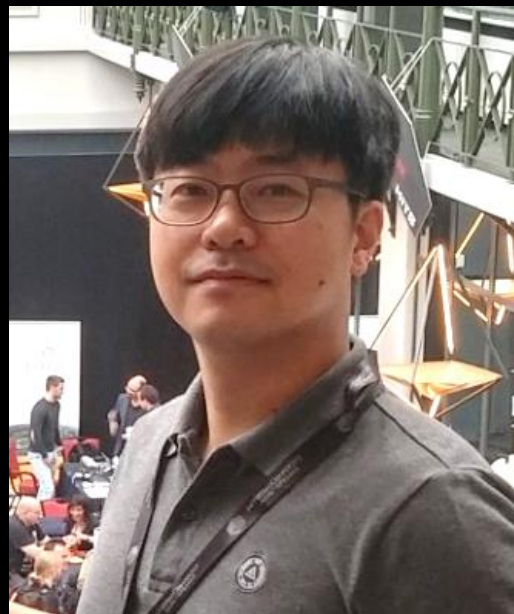


## A Practical Hypervisor Sandbox to Prevent Escapes from the KVM/QEMU and KVM-Based MicroVMs

Seunghun Han  
hanseunghun@nsr.re.kr

Sungjung Kim, Gaksoo Lim, Byungjoon Kim  
(sungjung || daygax || bjkim)@nsr.re.kr

# Who Am I?



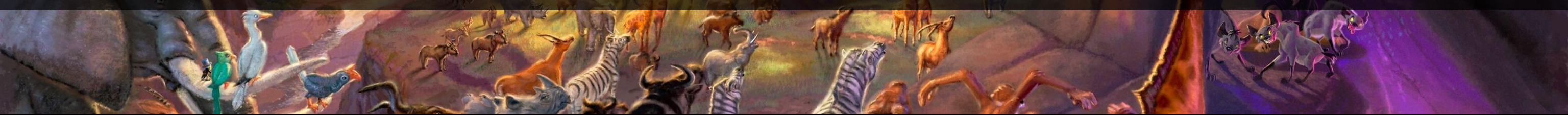
- **Senior security researcher** at the Affiliated Institute of ETRI
- **Review board member** of **Black Hat Asia** and **KimchiCon**
- **Speaker** at **USENIX Security**, **Black Hat Asia/Europe**, **HITBSecConf**, **BlueHat Shanghai**, **TyphoonCon**, **KimmchiCon**, **BECS**, etc.
- **Author** of “64-bit multi-core OS principles and structure, Vol.1&2”
- **Debian Linux maintainer** and **Linux kernel contributor**
- a.k.a kkamagui,  **@kkamagui1**

# Goal of This Talk

- **I present escape paths of KVM/QEMU and KVM-based microVMs**
  - KVM-based hypervisors are widely used from VMs to containers
  - KVM has higher privilege than the kernel and can be used for escapes
- **I introduce a new tool, Alcatraz, to prevent escapes**
  - It downgrades KVM's privilege (Ring -1) to the guest hypervisor (Ring 0)
  - It makes a sandbox for the KVM and monitors system calls to prevent escapes
    - Process creation, kernel code modification, and privilege escalation



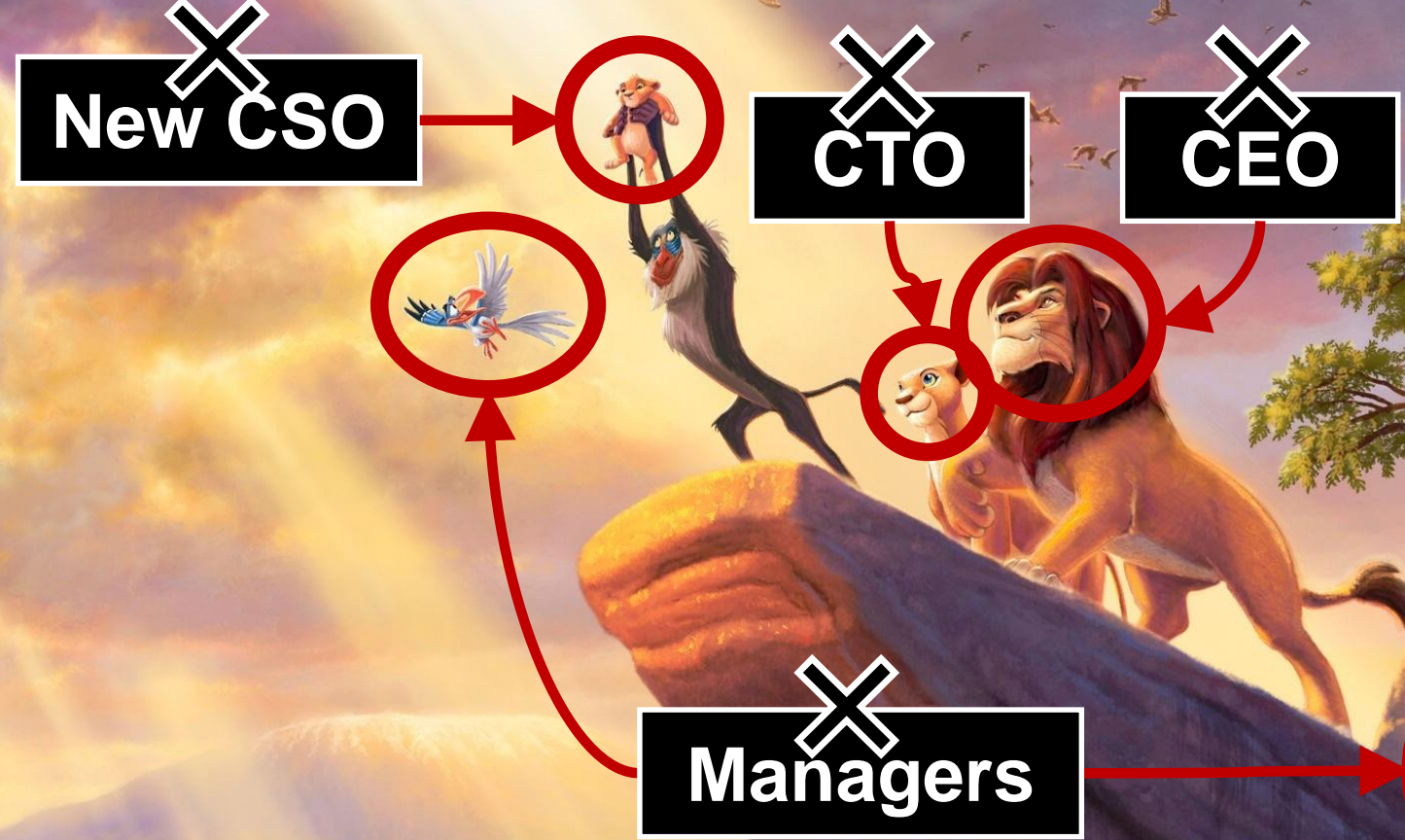
Once upon a time, there was ...



~~SMALL YOUNG~~

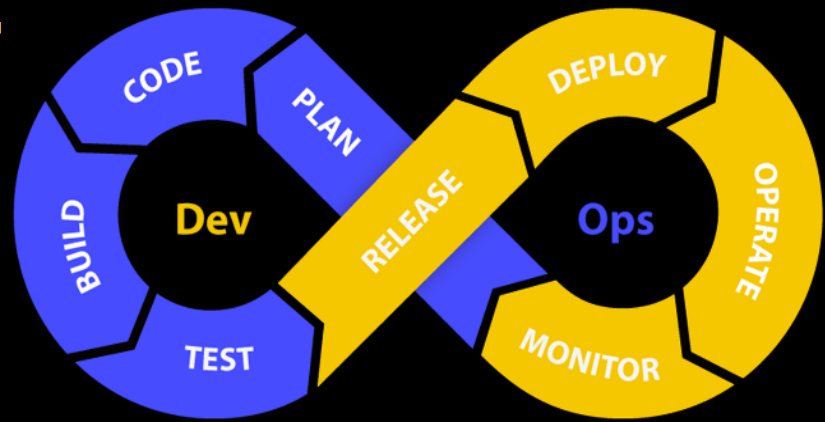


**RESEARCHER!**

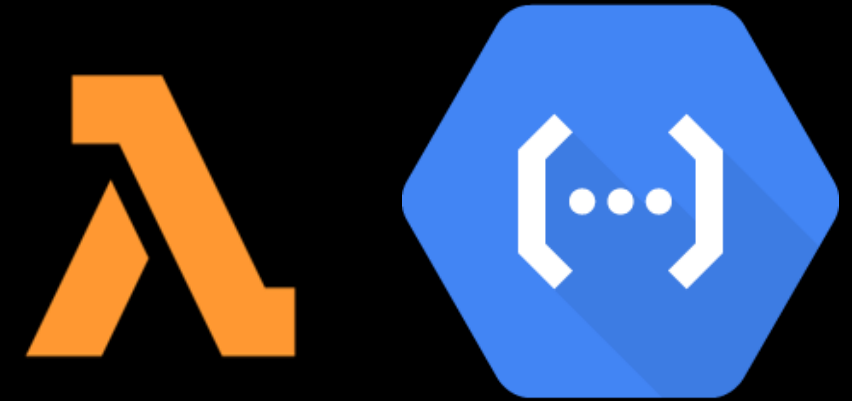


**There was JUST a security researcher**  
(not a CEO or CTO, not even a new CSO or manager)

# A New Era of Cloud Platforms has come

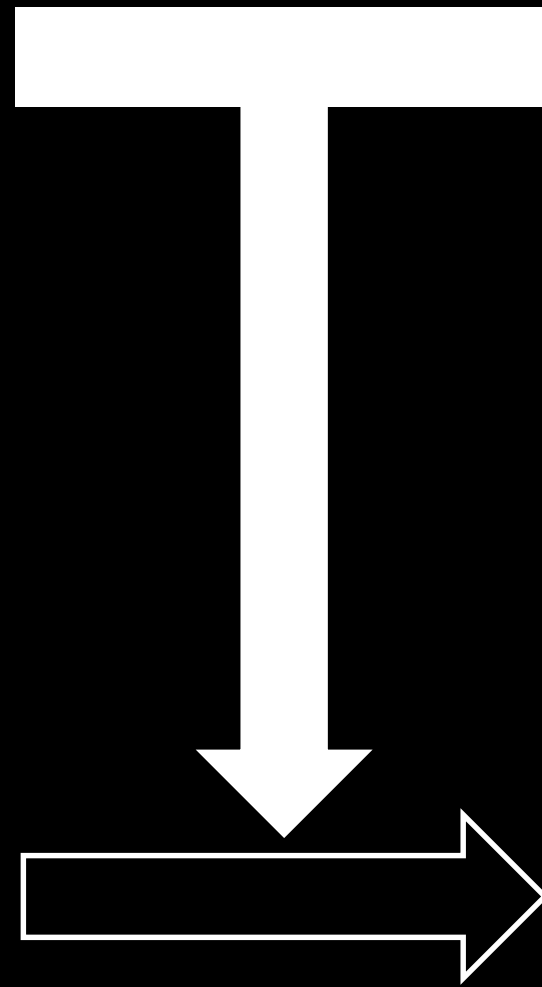


**DevOps**



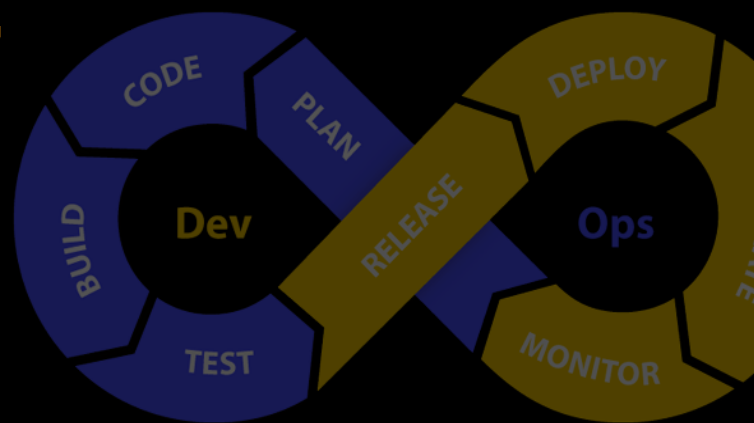
**Serverless**

**VM-based  
Technology**



**Container-based  
Technology**

# A New Era of Cloud Platforms has come

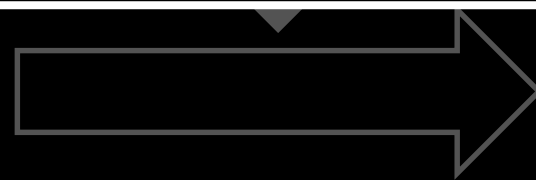


DevOps



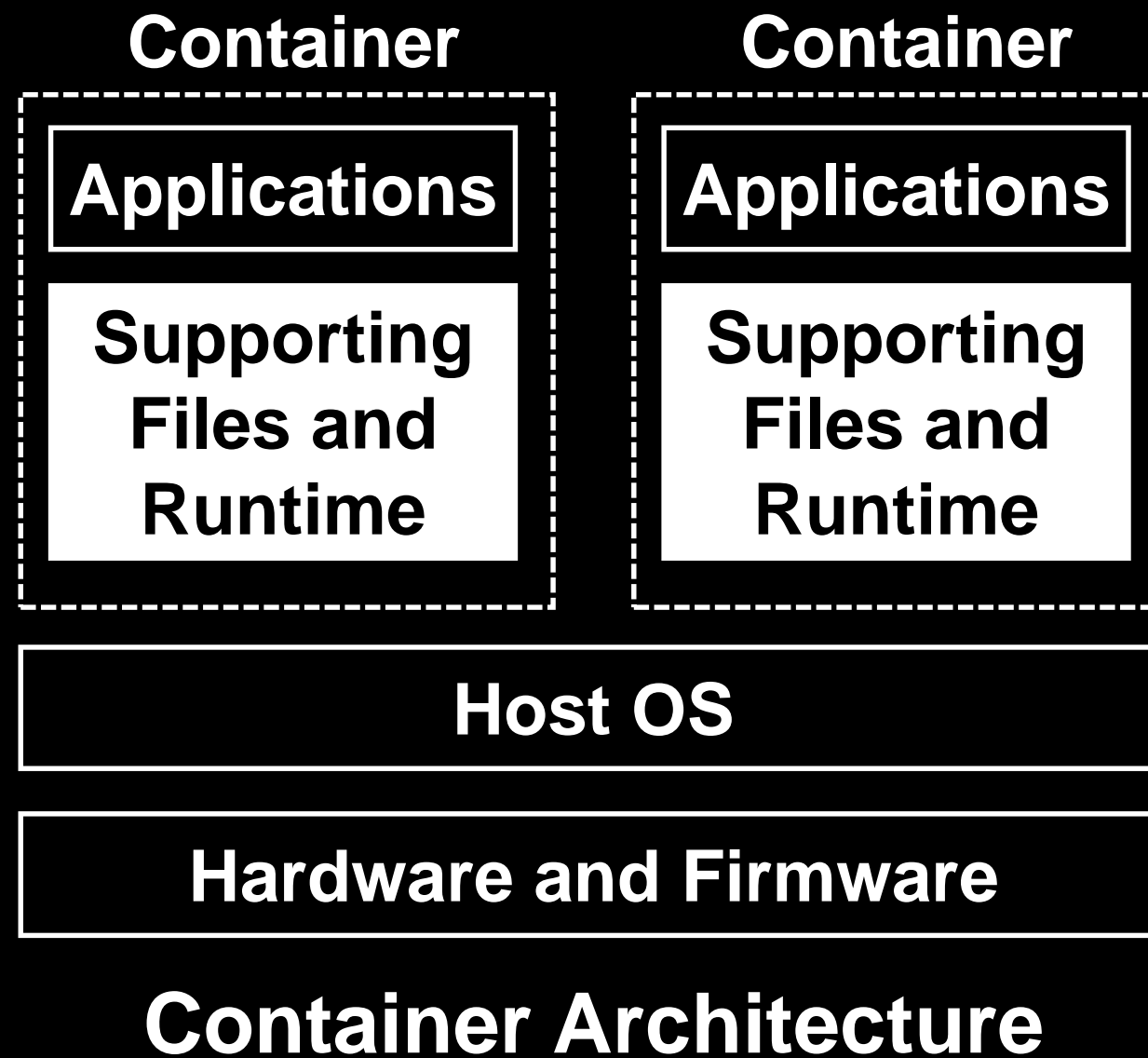
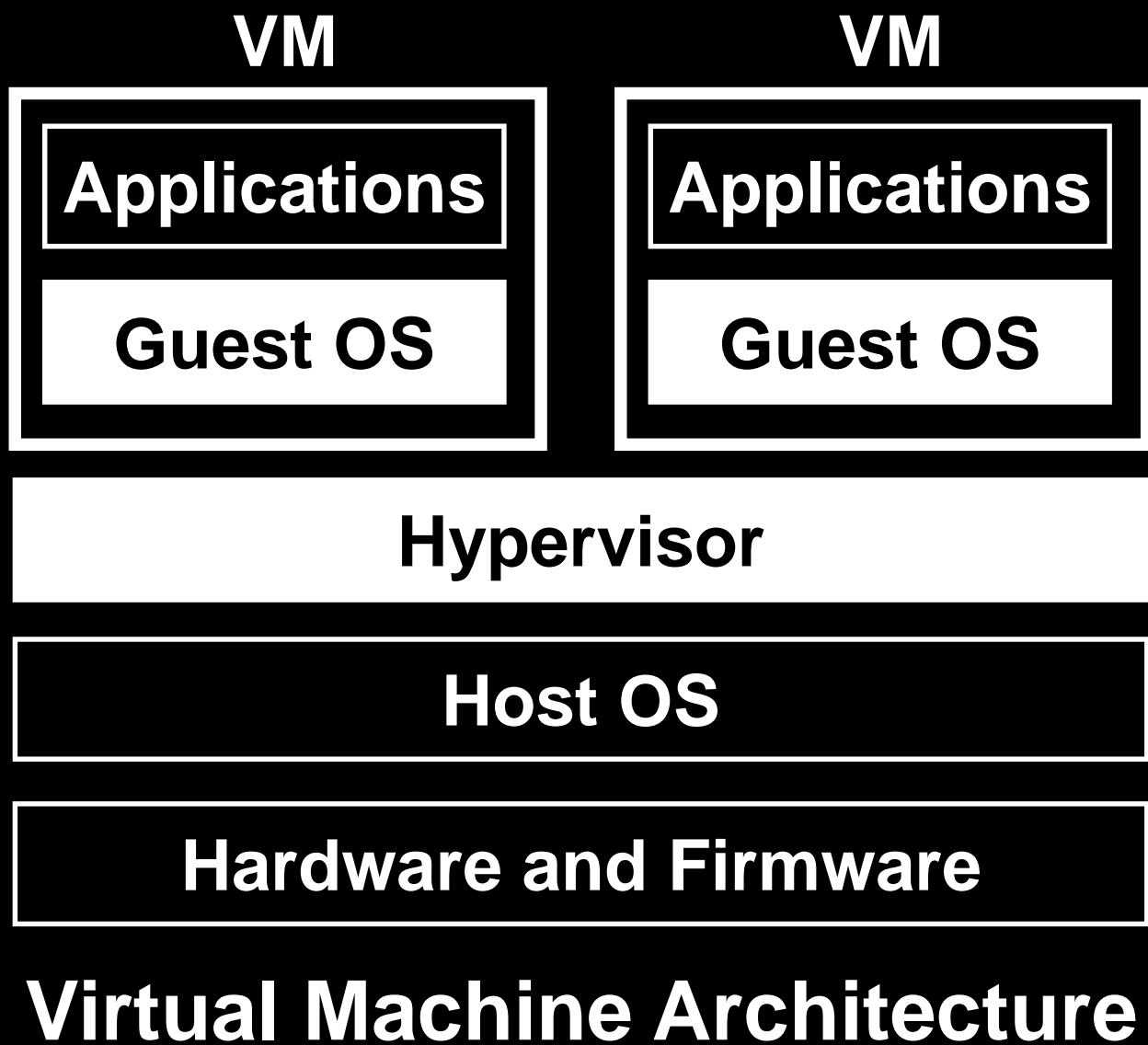
Serverless

VM-based  
Technology



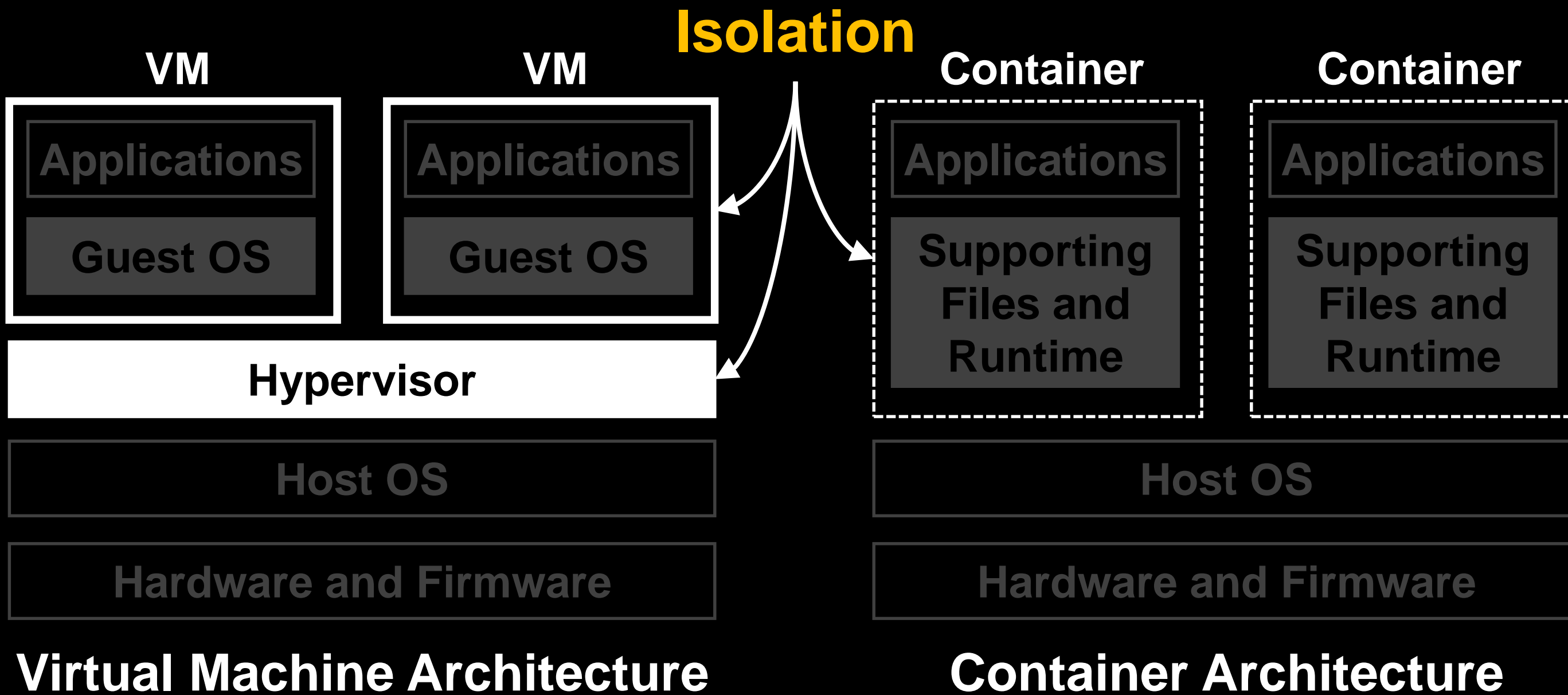
Container-based  
Technology

# Virtual Machine (VM) vs Container (1)

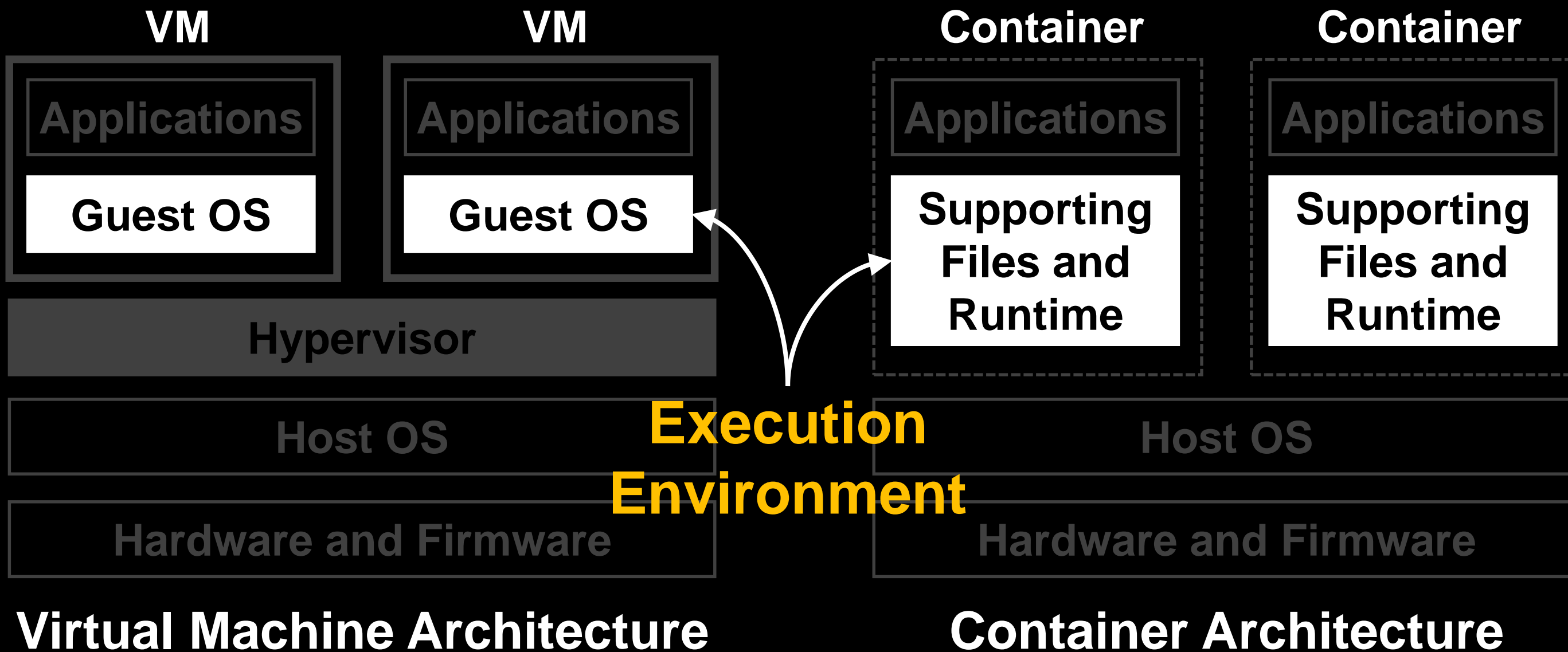




# Virtual Machine (VM) vs Container (1)



# Virtual Machine (VM) vs Container (1)



# Virtual Machine (VM) vs Container (2)

## - Virtual machine architecture

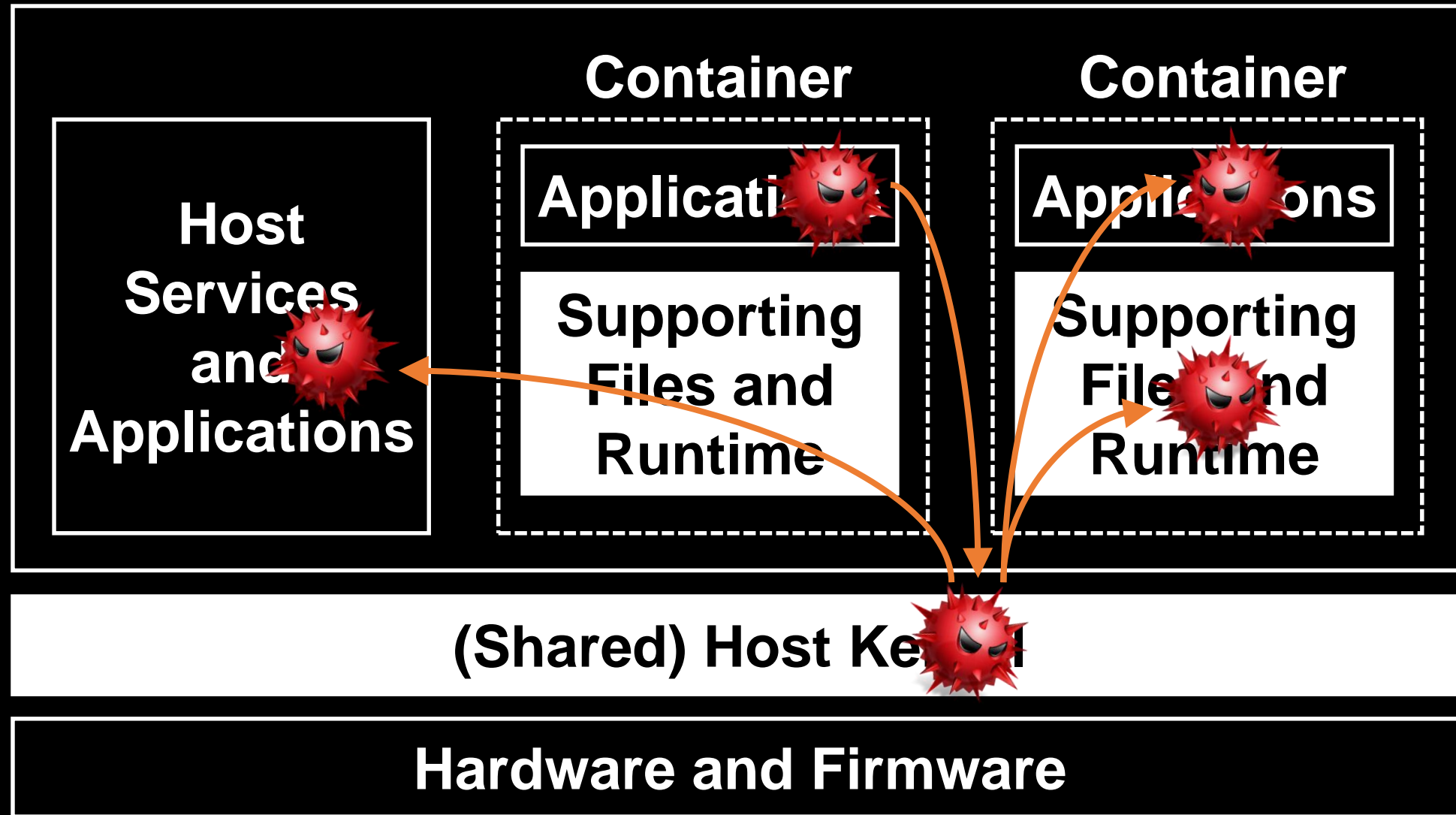
- uses **Virtual Machine Monitor (VMM)**, a.k.a. **hypervisor**
- provides strong isolation with virtualized hardware
  - A guest OS is needed, and performance overhead is high
- has each execution environment, such as kernel and user space

## - Container architecture

- uses **namespace and cgroup features** of **Linux kernel**
- provides weak isolation with the kernel
  - Supporting files are needed, and performance overhead is low
- shares the kernel and has its own user space only

# The container is good, but the problem is ...

## Host User Space

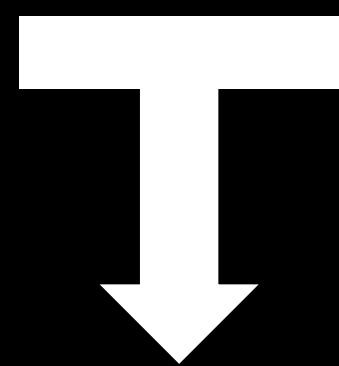


## Container Architecture

**Containers** are indispensable!

But, **strong isolation** is needed  
to **prevent** the **ESCAPES**

# Container



# Hypervisor



**kata**  
containers

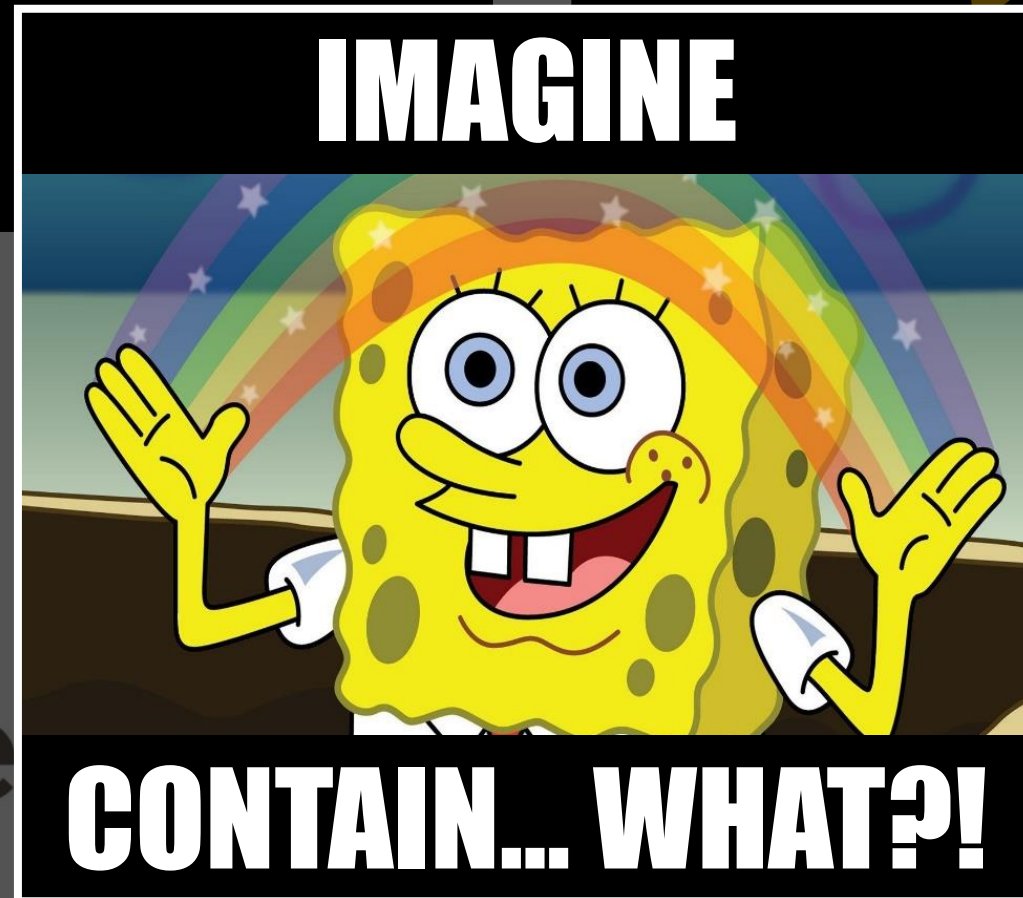


**Firecracker**



**gVisor**

# Container Hypervisor



Fire

visor

**To be or not to be,  
that is the question**



**- William Shakespeare**

**So, the security researcher realized ...**



To be **a container** or not to be **a container**,  
that is **NOT** the question



- Anonymous Researcher

The **REAL** question is ...

**How can we prevent escapes?**



# ALCATRAZ

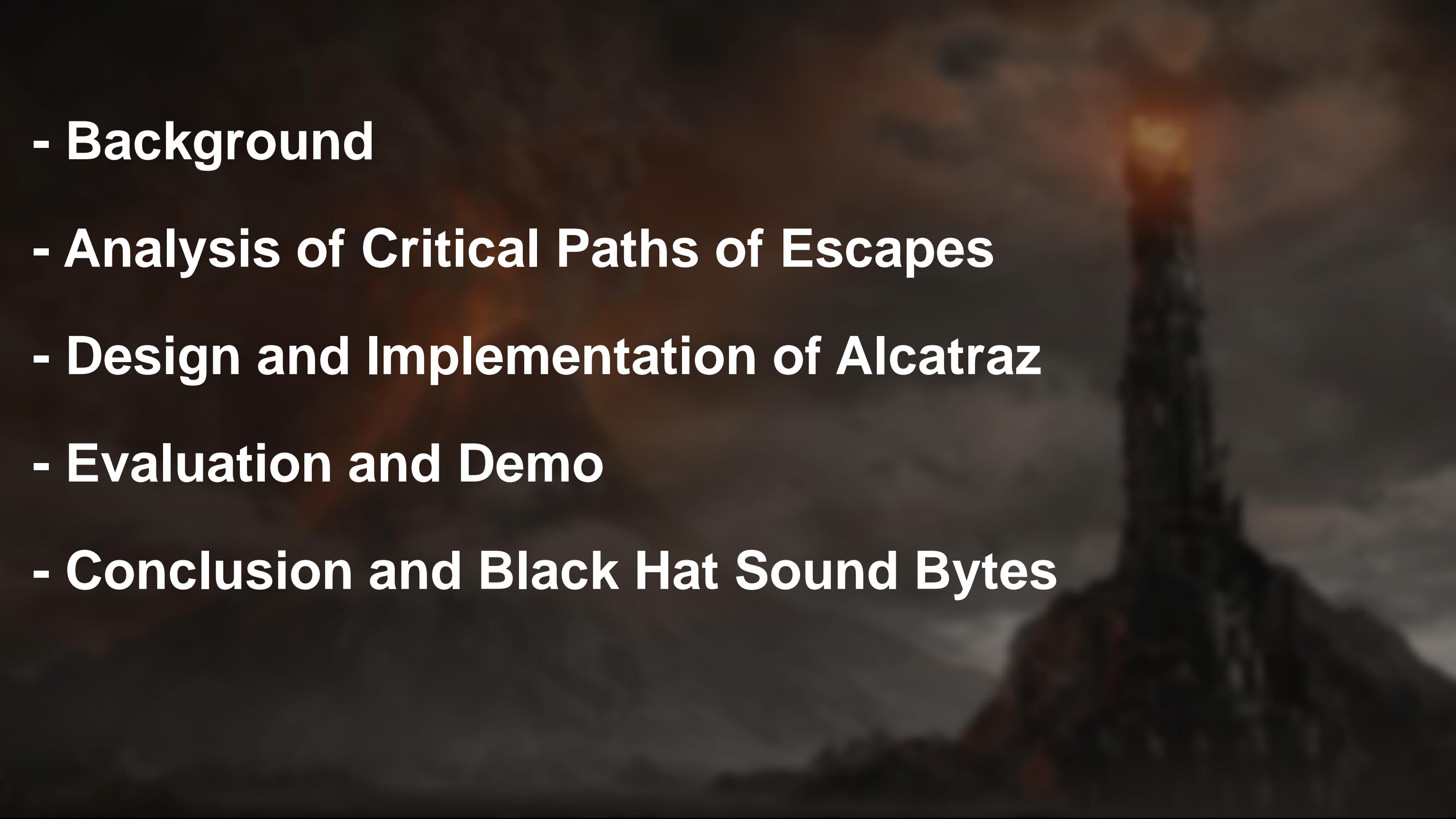
It's me!

FOR



THE SOLUTION!



- 
- **Background**
  - **Analysis of Critical Paths of Escapes**
  - **Design and Implementation of Alcatraz**
  - **Evaluation and Demo**
  - **Conclusion and Black Hat Sound Bytes**

- **Background**



- Analysis of Critical Paths of Escapes

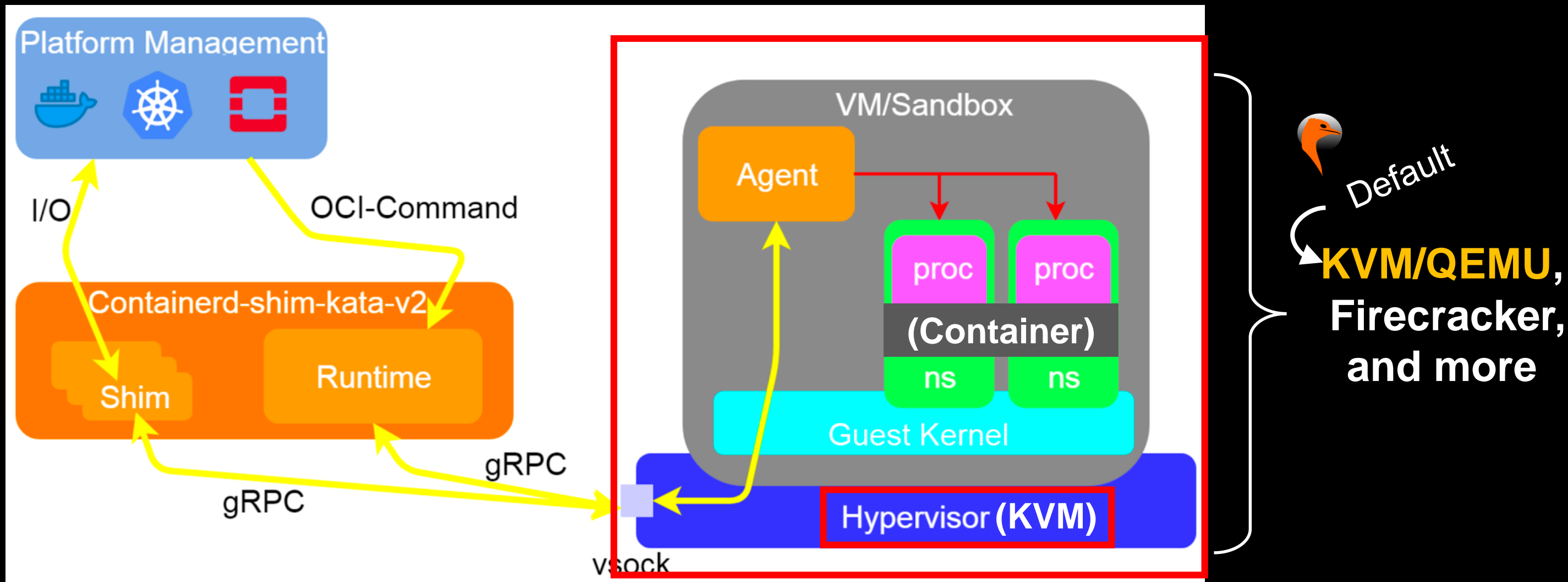
- Design and Implementation of Alcatraz

- Evaluation and Demo

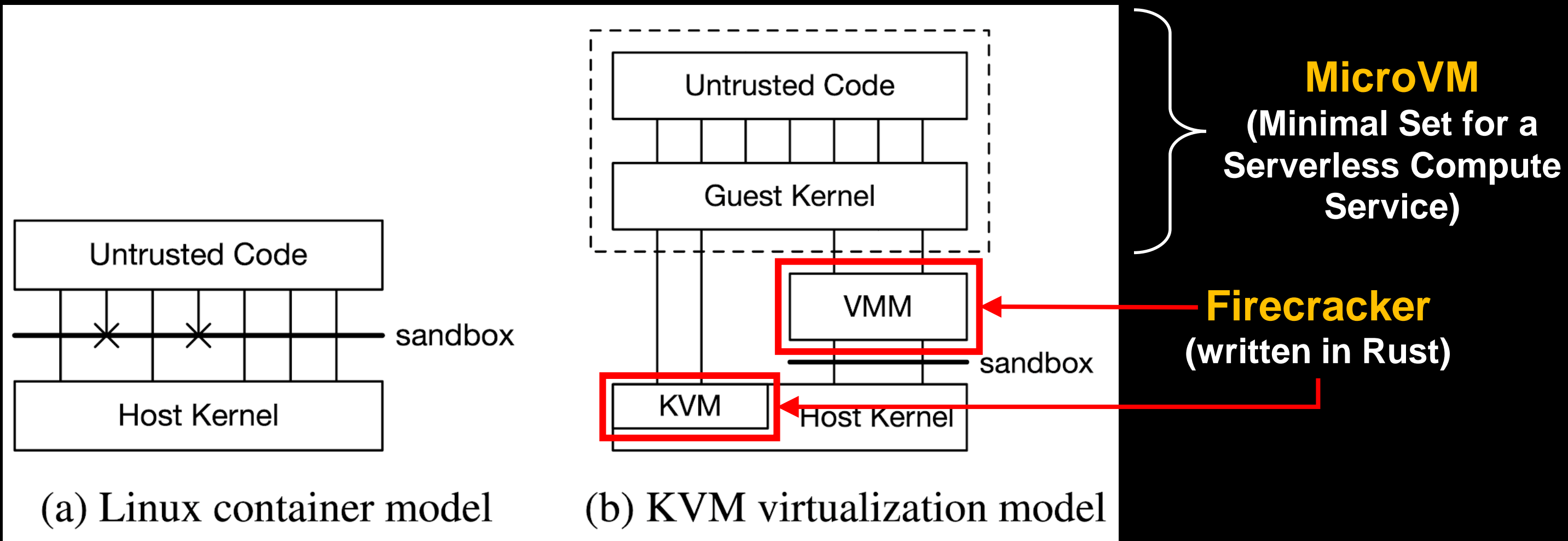
- Conclusion and Black Hat Sound Bytes

# Kata Container

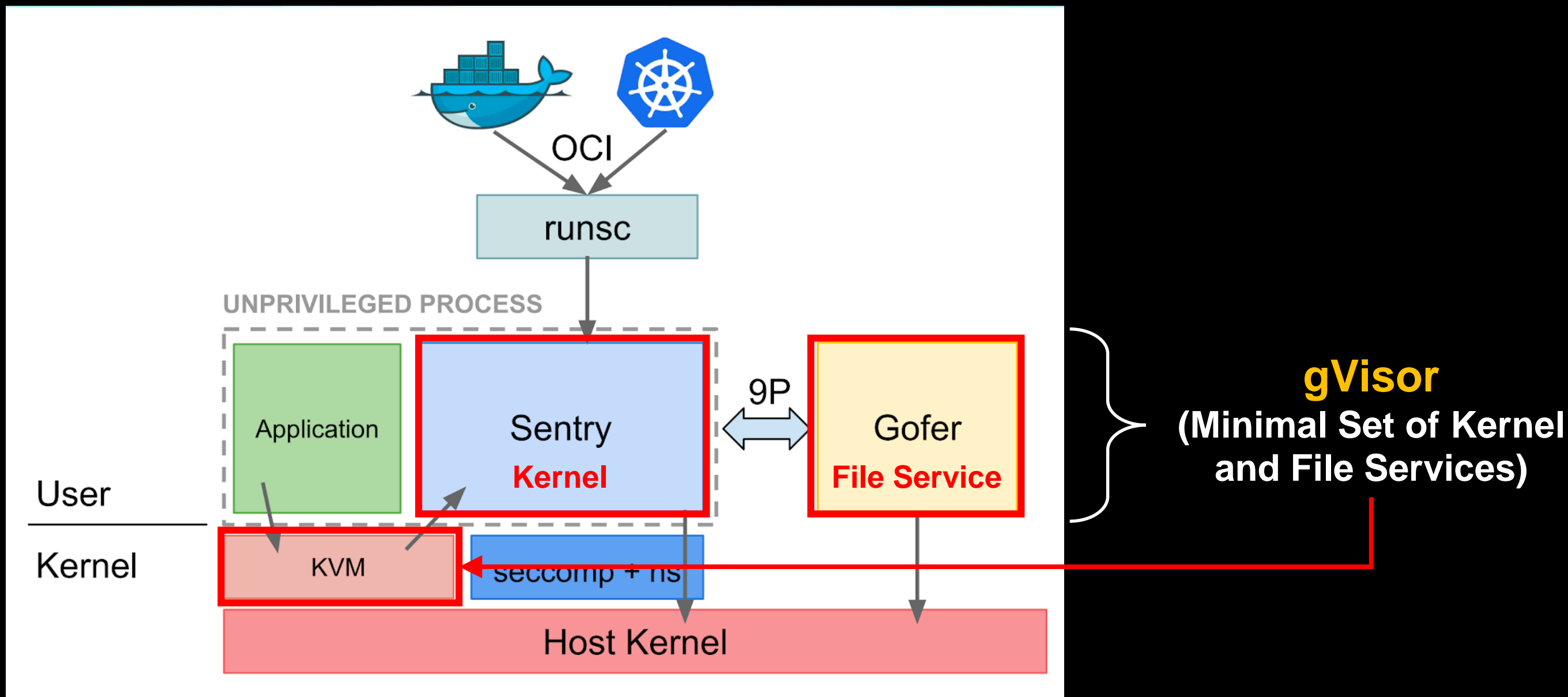
## Kata Container



**Kata Architecture - Kata Containers: An Emerging Architecture for Enabling MEC Services in Fast and Secure Way**

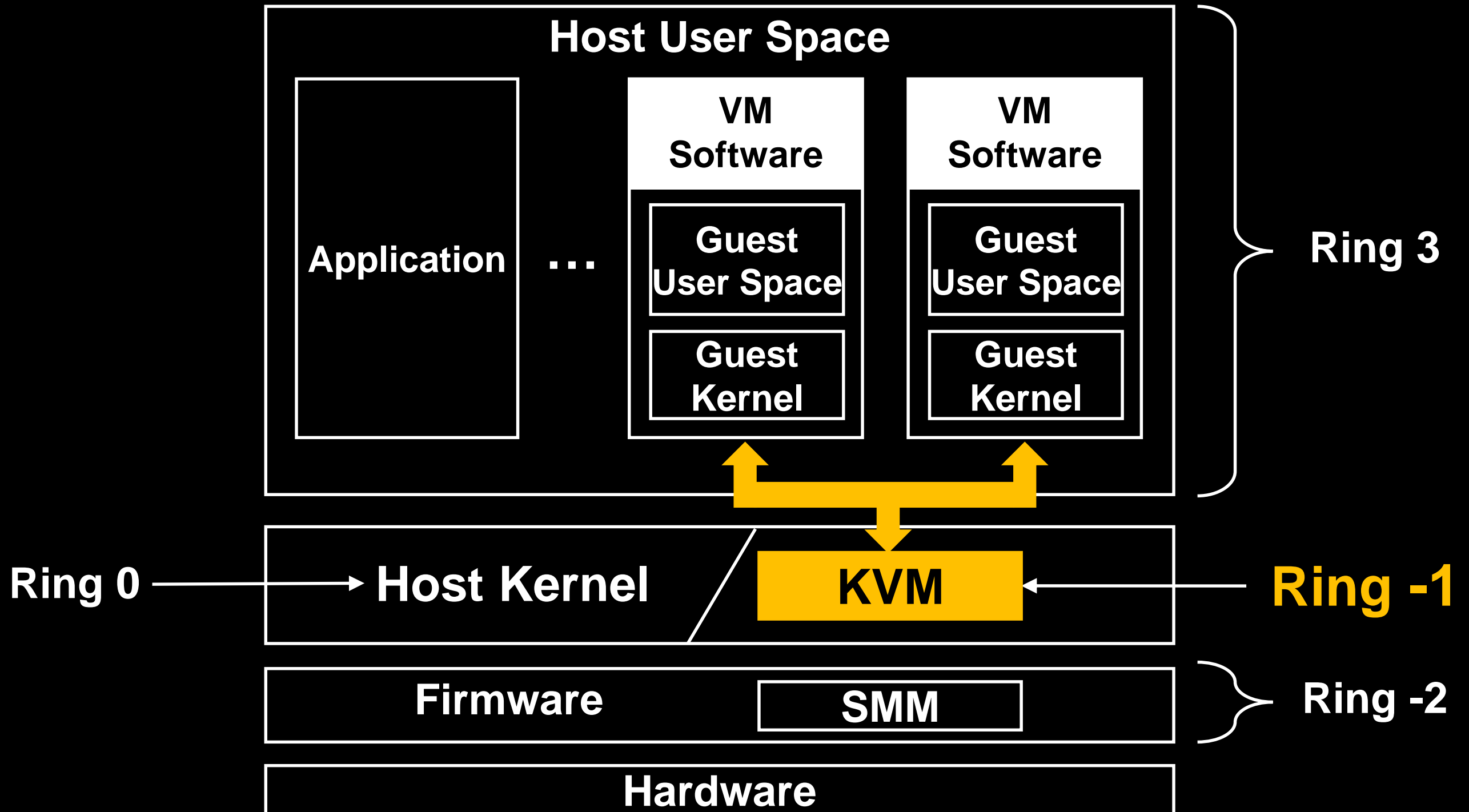


## The Security Model of Linux Containers - Firecracker: Lightweight Virtualization for Serverless Applications



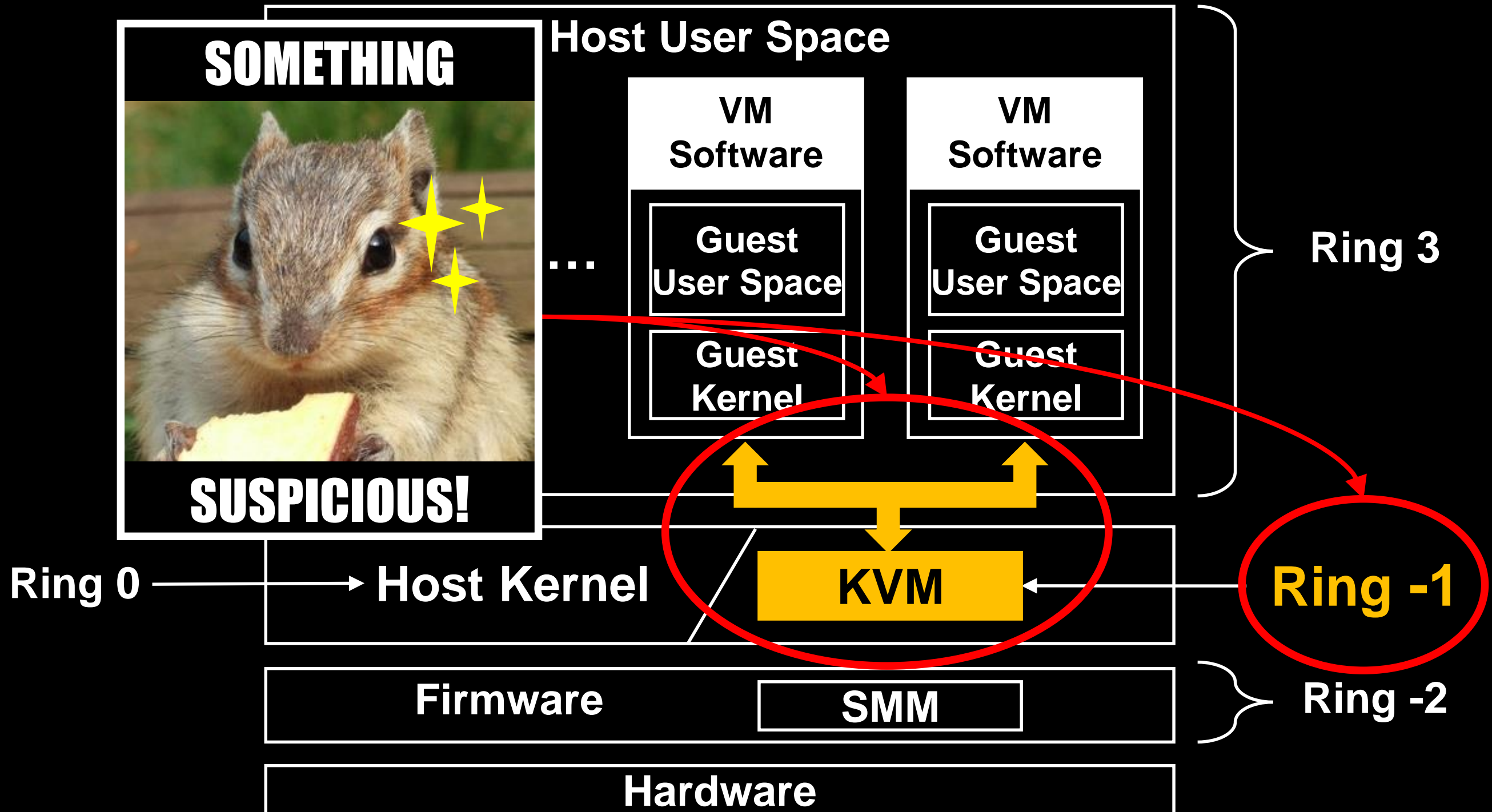
An OCI runtime powered by gVisor-Container Isolation at Scale (... and introducing gVisor)

# Kernel-based Virtual Machine (KVM)





# Kernel-based Virtual Machine (KVM)



A hypervisor attaches to VMs **tightly**

It has **Ring -1** privilege!

The attacker can **escape** from the VM  
with a **KVM vulnerability**  
(and subvert the system)

# Fortunately,

I was not the only one who was worried about **the excessive privilege** of a hypervisor!



Previous Works

# HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity \*

Ahmed M. Azab Peng Ning Zhi Wang Xuxian Jiang  
Department of Computer Science, North Carolina State University  
{amazab, pning, zhi\_wang, xuxian\_jiang}@ncsu.edu

Xiaolan Zhang  
IBM T.J. Watson Research Center  
cxzhang@us.ibm.com

Nathan C. Skalsky  
IBM Systems & Technology Group  
nskalsky@us.ibm.com

## ABSTRACT

This paper presents *HyperSentry*, a novel framework to enable integrity measurement of a running hypervisor (or any other highest privileged software layer on a system). Unlike existing solutions for protecting privileged software, HyperSentry does not introduce a higher privileged software layer below the integrity measurement target, which could start another race with malicious attackers in obtaining the highest privilege in the system. Instead, HyperSentry introduces a software component that is properly isolated from the hypervisor to enable *stealthy* and *in-context* measurement of the runtime integrity of the hypervisor. While stealthiness is necessary to ensure that a compromised hypervisor does not have a chance to hide the attack traces upon detecting an up-coming measurement, in-context measurement is

we implement a prototype of the framework along with an integrity measurement agent for the Xen hypervisor. Our experimental evaluation shows that HyperSentry is a low-overhead practical solution for real world systems.

## Categories and Subject Descriptors

D.4.6 [Operating Systems]: Security and Protection—*invasive software*

## General Terms

Security

## Keywords

Virtualization, Hypervisor Integrity, Integrity Measurement

## 1. INTRODUCTION



# CHIPSET BASED APPROACH TO DETECT VIRTUALIZATION MALWARE a.k.a. DeepWatch

Yuriy Bulygin

Joint work with David Samyde

Security Center of Excellence / PSIRT @ Intel Corporation

# Preventing and Detecting Xen Hypervisor Subversions

Joanna Rutkowska & Rafał Wojtczuk  
Invisible Things Lab

Black Hat USA 2008, August 7th, Las Vegas, NV

## Protecting Cloud Virtual Machines from Commodity Hypervisor and Host Operating System Exploits

Shih-Wei Li John S. Koh Jason Nieh  
Department of Computer Science  
Columbia University  
{shihwei, koh, nieh}@cs.columbia.edu

## Abstract

Hypervisors are widely deployed by cloud computing providers to support virtual machines, but their growing complexity poses a security risk as large codebases contain many vulnerabilities. We have created HypSec, a new hypervisor design for retrofitting an existing commodity hypervisor using microkernel principles to reduce its trusted computing base while protecting the confidentiality and integrity of virtual machines. HypSec partitions the hypervisor into an untrusted host that performs most complex hypervisor functionality without access to virtual machine data, and a trusted core that provides access control to virtual machine data and performs basic CPU and memory virtualization. Hardware virtualization support is used to isolate and protect the trusted core and execute it

maintenance effort. For example, KVM [44] is integrated with Linux and Hyper-V [56] is integrated with Windows. The result is a huge potential attack surface with access to VM data in CPU registers, memory, I/O data, and boot images. The surge in outsourcing of computational resources to the cloud and away from privately-owned data centers further exacerbates this security risk of relying on the trustworthiness of complex and potentially vulnerable hypervisor and host OS infrastructure. Attackers that successfully exploit hypervisor vulnerabilities can gain unfettered access to VM data, and compromise the privacy and integrity of all VMs—an undesirable outcome for both cloud providers and users.

Recent trends in application design and hardware virtualization support provide an opportunity to revisit hypervisor design requirements to address this crucial security problem.

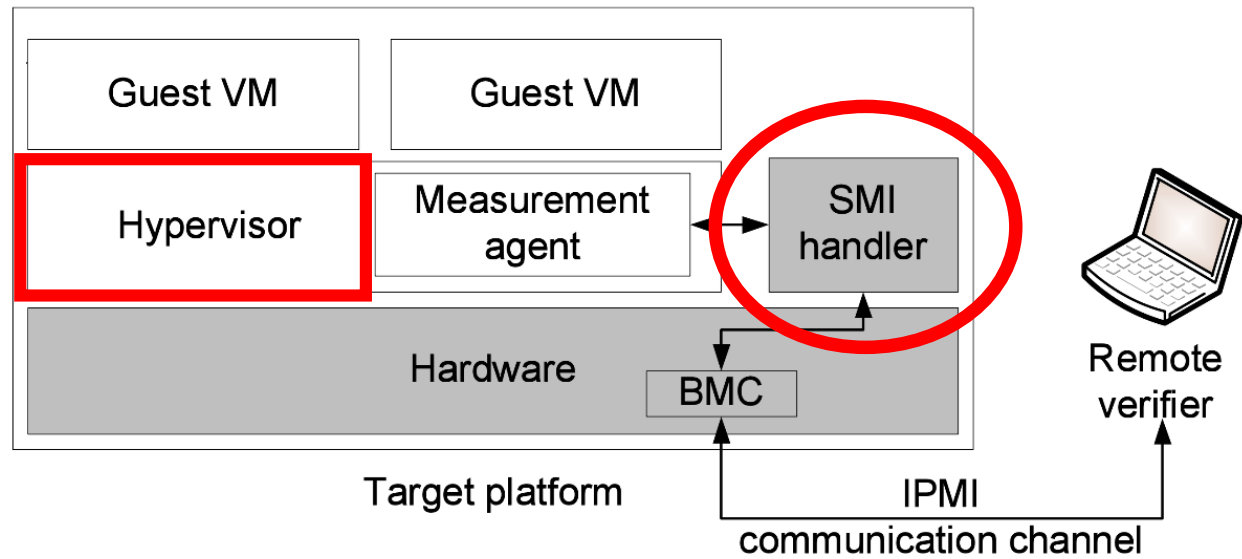


Figure 1: HyperSentry's architecture (Trusted components are shown in gray.)

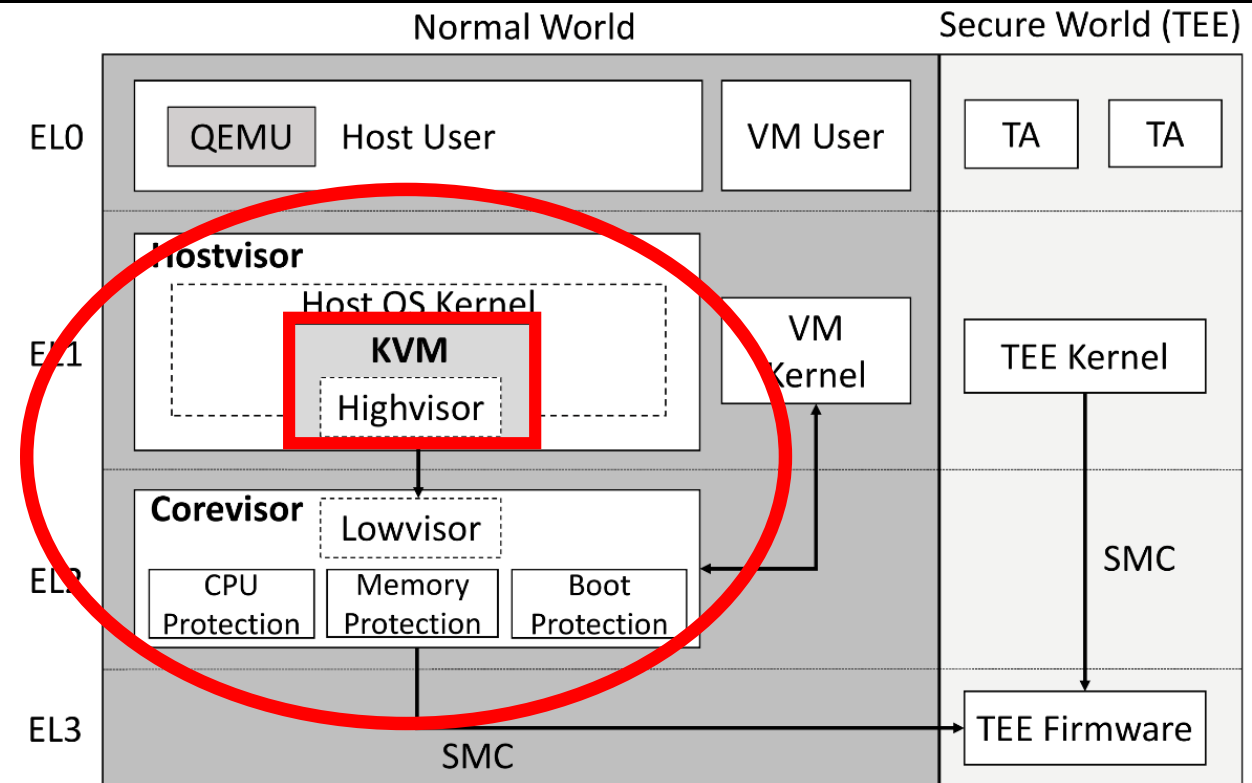
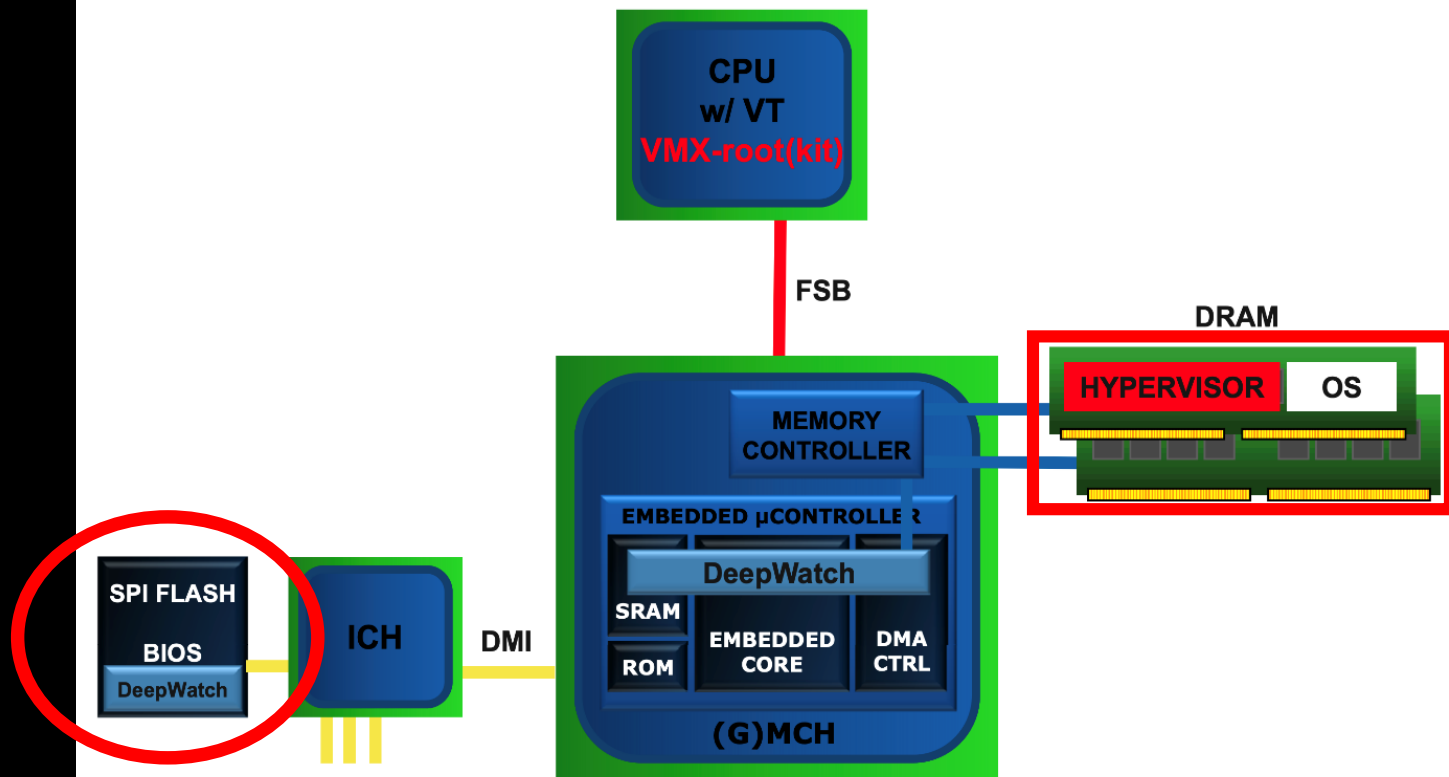
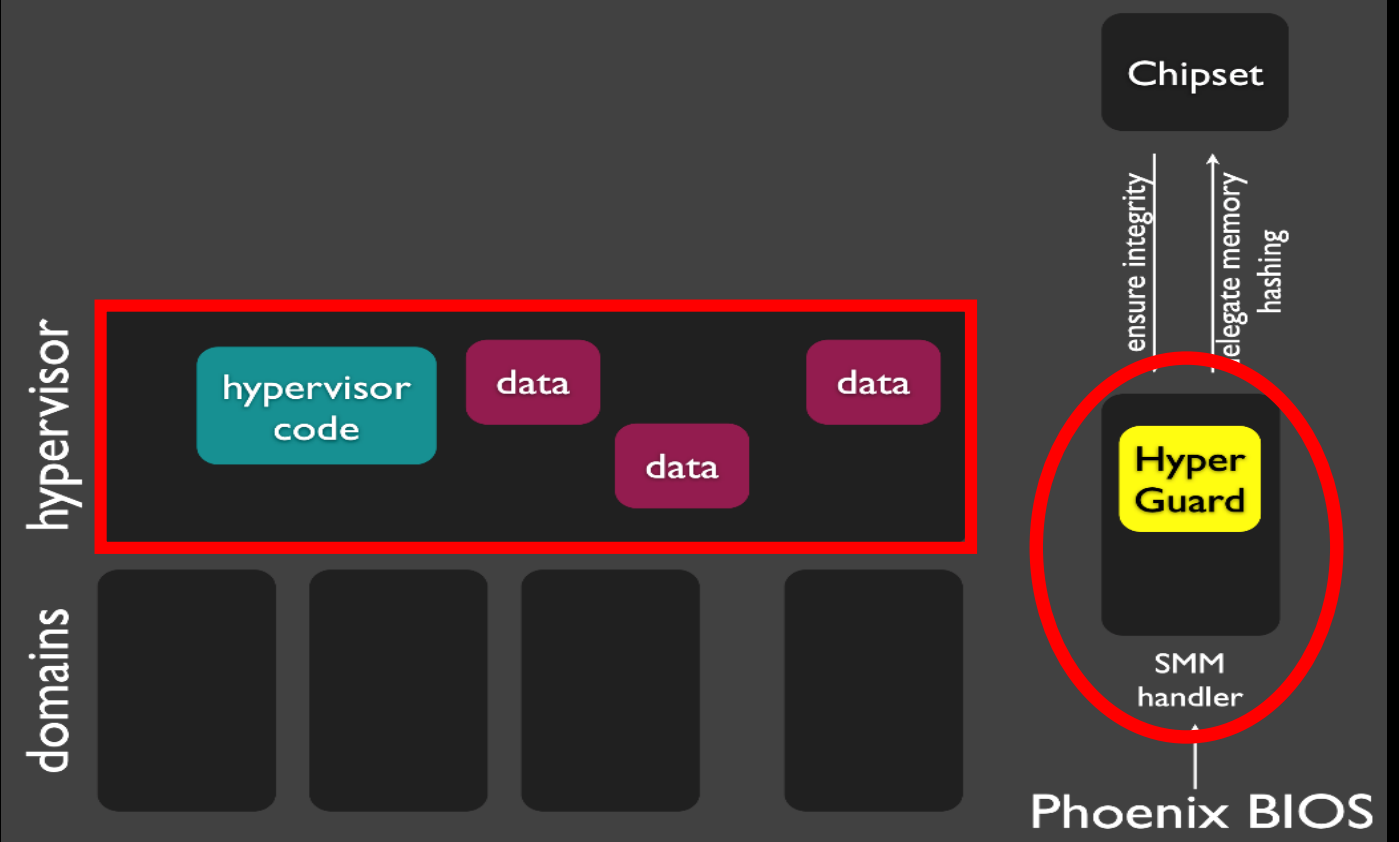


Figure 3: HypSec on KVM/ARM

# Firmware Modification

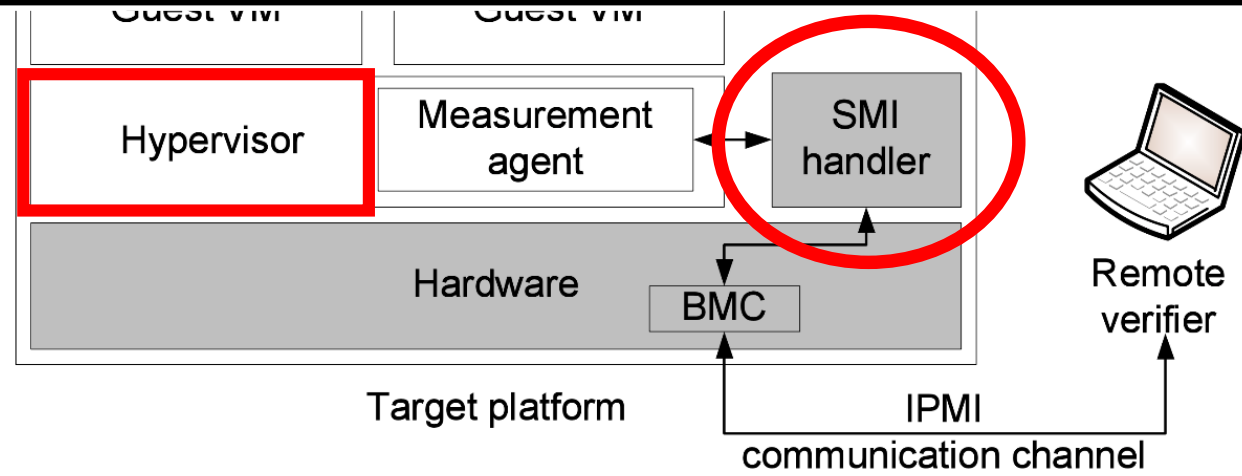
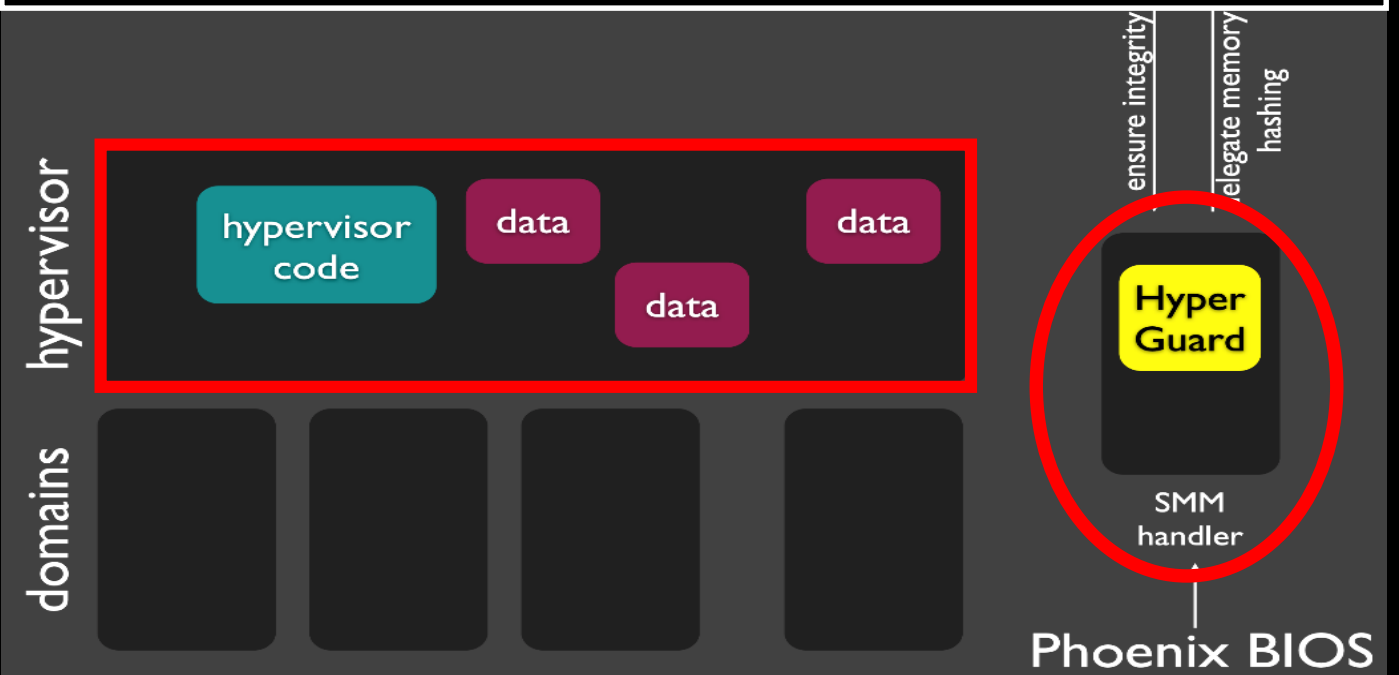
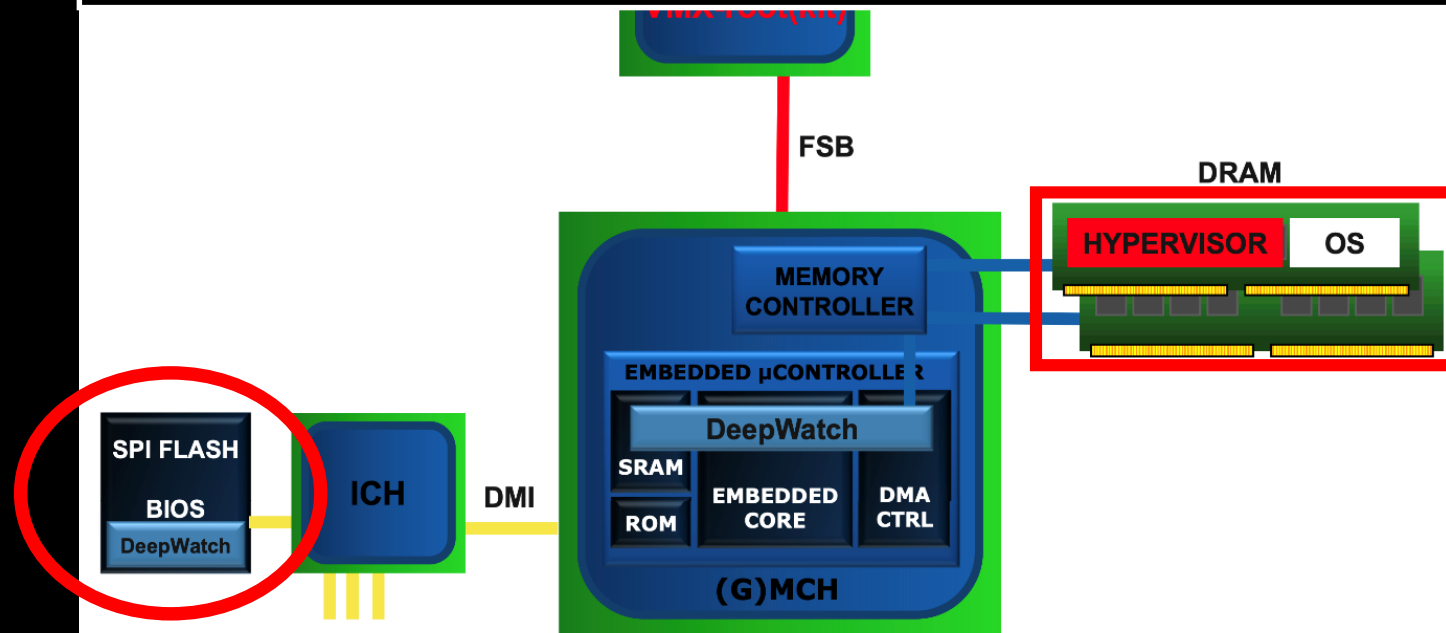


Figure 1: HyperSentry's architecture (Trusted components are shown in gray.)

# Firmware Modification



# Firmware Modification



# Hypervisor Retrofit

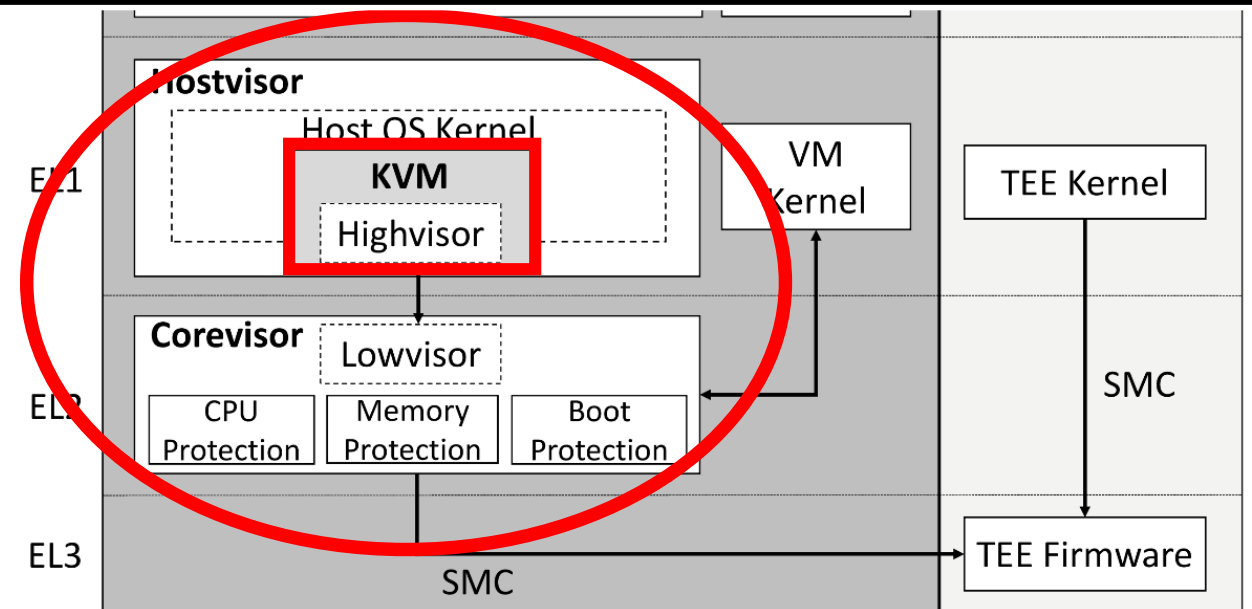


Figure 3: HypSec on KVM/ARM

# Firmware Modification

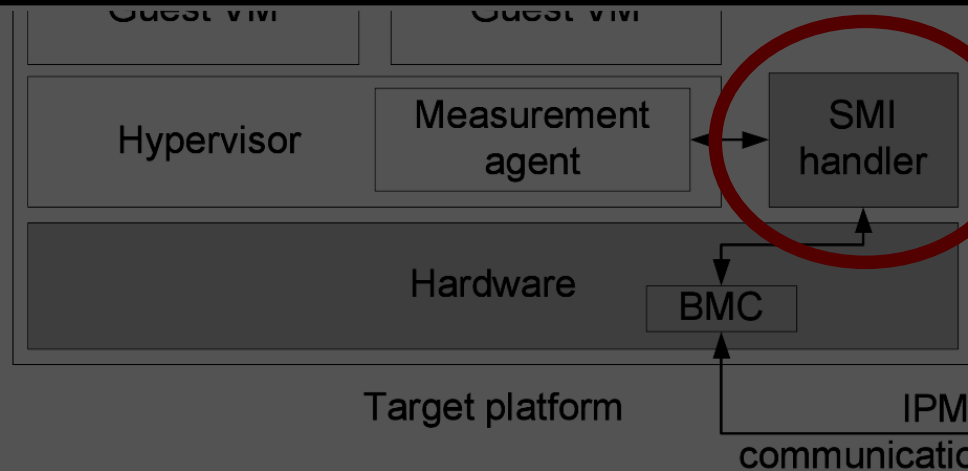


Figure 1: HyperSentry's architecture (components are shown in gray.)

# Firmware Modification



# Firmware Modification

# Hypervisor Retrofit

I need a more **PRACTICAL** solution!

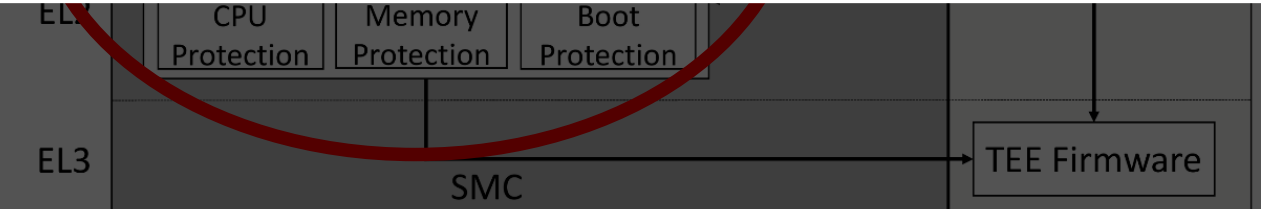
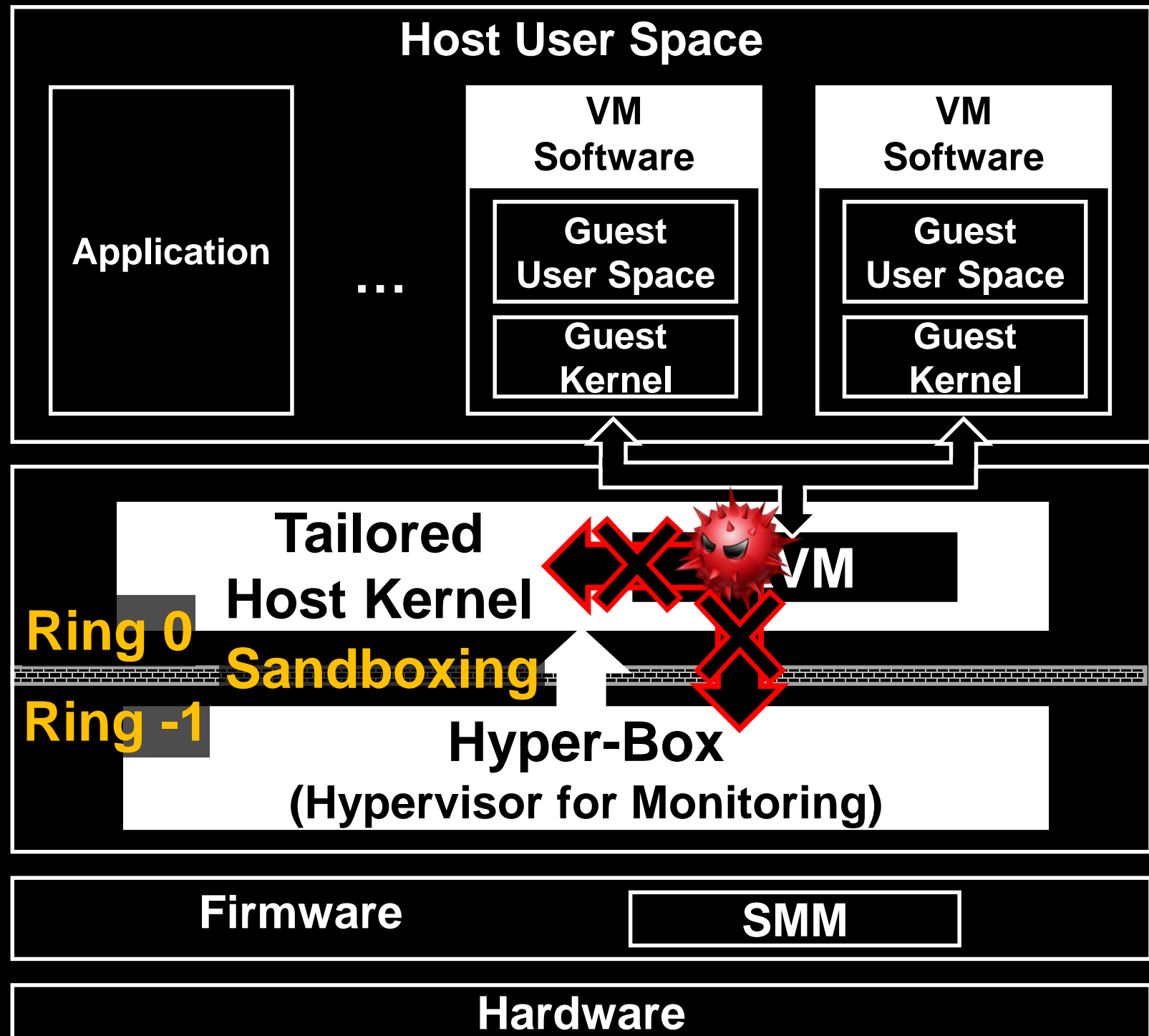
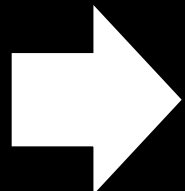
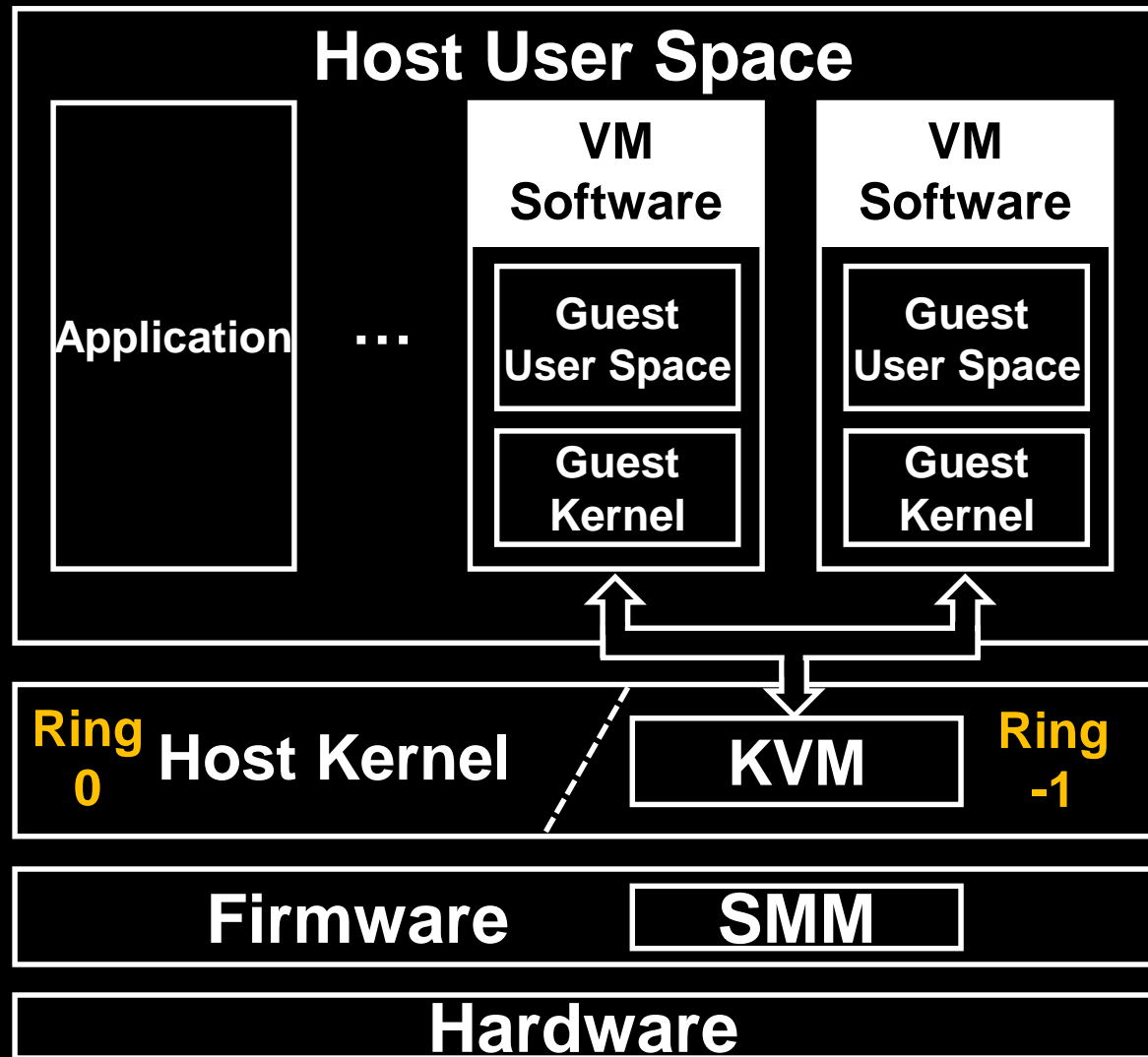


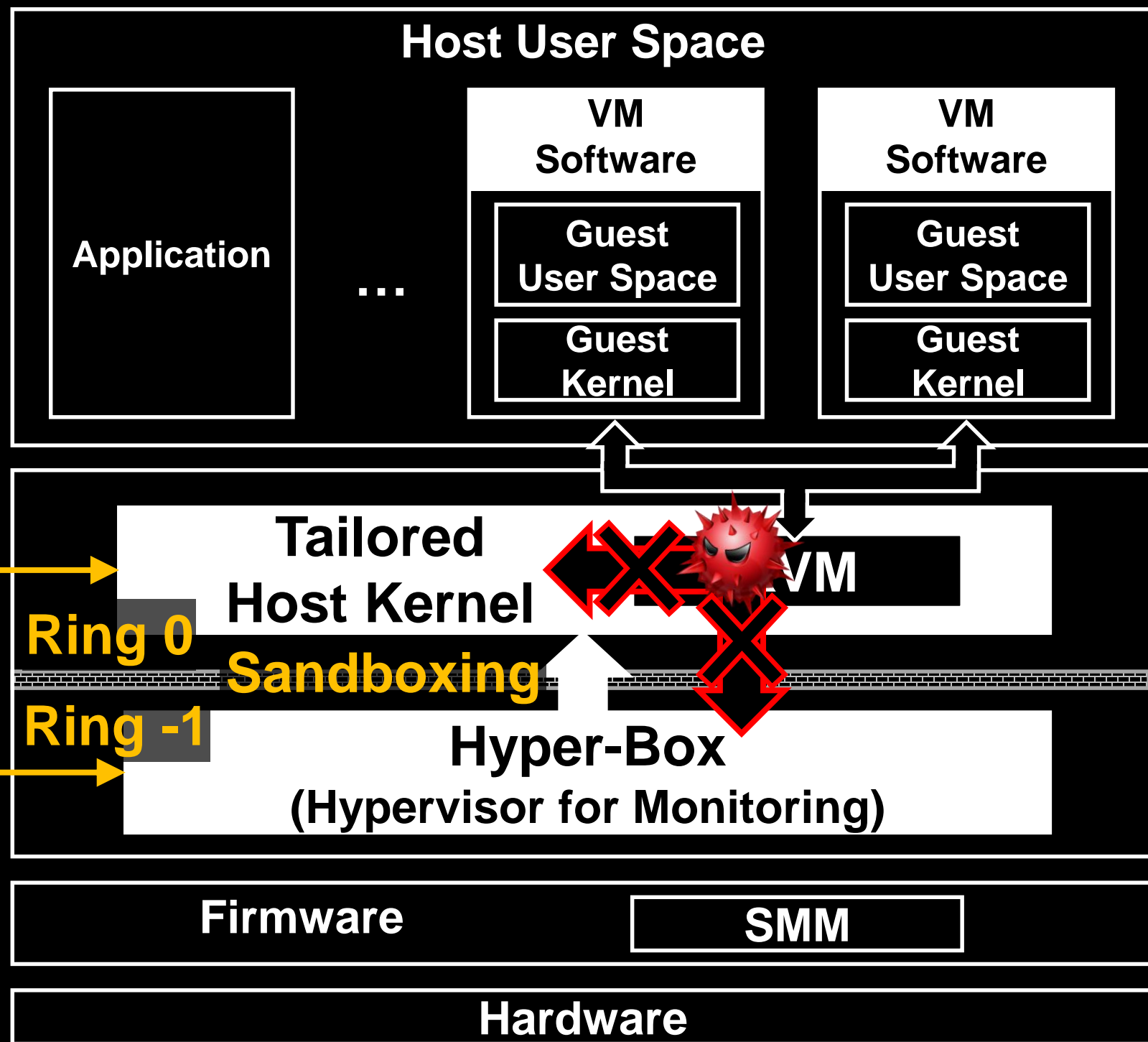
Figure 3: HypSec on KVM/ARM

# The Solution: Sandboxing Hypervisor





# The Solution: Sandboxing Hypervisor



- Background

- **Analysis of Critical Paths of Escapes**



- Design and Implementation of Alcatraz

- Evaluation and Demo

- Conclusion and Black Hat Sound Bytes

**CVE-2021-22543: Improper page handling of KVM, privilege escalation**  
CVE-2021-20181: TOCTOU in 9pfs of QEMU, privilege escalation  
CVE-2021- 3546: OOB in VIRTIO in QEMU, code execution  
CVE-2021- 3409: OOB in SDC of QEMU, code execution  
CVE-2020-35506: UAF in SCSI of QEMU, code execution  
CVE-2020-17380: OOB in SDC of QEMU, code execution  
CVE-2020-15863: OOB in NET of QEMU, code execution  
CVE-2020-14364: OOB in USB of QEMU, code execution  
CVE-2020- 1711: OOB in iSCSI of QEMU, code execution  
CVE-2019-18389: OOB in VIRTIO-GPU in QEMU, code execution  
**CVE-2019-14835: OOB in KVM, privilege escalation**  
**CVE-2019-14821: OOB in MMIO of KVM, privilege escalation**  
**CVE-2019- 7221: UAF in HR timer of KVM, code execution**  
**CVE-2018-16882: UAF in KVM, privilege escalation**  
CVE-2018-16847: OOB in NVME of QEMU, code execution  
**CVE-2018-12904: Improper VMX handling of KVM, privilege escalation**  
**CVE-2018-10901: Improper GDT handling of KVM, privilege escalation**  
**CVE-2018- 1087: Improper stack switching of KVM, privilege escalation**  
**CVE-2017-12188: Improper page handling of KVM, code execution**  
CVE-2017- 7980: OOB in display of QEMU, code execution  
CVE-2017- 5931: Integer overflow in VIRTIO of QEMU, code execution  
CVE-2017- 5667: OOB in SDC of QEMU, code execution  
CVE-2017- 2630: Stack overflow in NBD of QEMU, code execution  
CVE-2017- 2620: OOB in display of QEMU, code execution  
CVE-2017- 2615: OOB in display of QEMU, code execution  
**CVE-2016-10150: UAF in KVM, local privilege escalation**  
**CVE-2016- 9777: Improper IOAPIC handing in KVM, privilege escalation**  
CVE-2016- 9603: OOB in display of QEMU, code execution  
CVE-2016- 7161: OOB in NET of QEMU, code execution  
CVE-2016- 6351: OOB in SCSI of QEMU, code execution  
CVE-2016- 5338: OOB in SCSI of QEMU, code execution  
CVE-2016- 5126: OOB in iSCSI of QEMU, code execution  
**CVD-2016- 4440: Improper APICv handling of KVM, code execution**  
CVE-2016- 4439: OOB in SCSI of QEMU, code execution  
CVE-2016- 4002: OOB in NET of QEMU, code execution  
CVE-2016- 3710: OOB in VESA VGA of QEMU, code execution  
CVE-2016- 1714: OOB in NVRAM of QEMU, code execution  
CVE-2016- 1586: OOB in IDE of QEMU, code execution  
CVE-2016- 0749: OOB in SPICE of QEMU, code execution  
CVE-2015- 7512: OOB in NET of QEMU, code execution  
CVE-2015- 7504: OOB in NET of QEMU, code execution  
CVE-2015- 5279: OOB in NET of QEMU, code execution  
CVE-2015- 5278: OOB in NET of QEMU, code execution  
CVE-2015- 5260: OOB in SPICE of QEMU, code execution  
CVE-2015- 5225: OOB in VNC of QEMU, code execution  
CVE-2015- 5154: OOB in IDE of QEMU, code execution  
CVE-2015- 3456: OOB in FDC of QEMU (VENOM), code execution  
CVE-2015- 3214: OOB in PIT of QEMU, code execution  
CVE-2015- 3209: OOB in NET of QEMU, code execution  
CVE-2014- 8106: OOB in display of QEMU, code execution  
CVE-2014- 7840: OOB in migration function of QEMU, code execution  
CVE-2014- 3461: OOB in USB of QEMU, code execution  
CVE-2014- 0182: OOB in VIRTIO of QEMU, code execution  
CVE-2014- 0150: Integer overflow in VIRTIO-NET of QEMU, code execution  
CVE-2014- 0144: OOB in CLOOP of QEMU, code execution  
**CVE-2014- 0049: OOB in KVM, code execution**  
CVE-2013- 6399: OOB in VIRTIO of QEMU, code execution  
CVE-2013- 4544: OOB in NET of QEMU, code execution  
**CVE-2013- 4587: OOB in KVM, local privilege escalation**  
CVE-2013- 4542: OOB in SCSI of QEMU, code execution  
CVE-2013- 4541: OOB in USB of QEMU, code execution  
CVE-2013- 4540: OOB in GPIO of QEMU, code execution  
CVE-2013- 4539: OOB in Audio of QEMU, code execution  
CVE-2013- 4538: OOB in Display of QEMU, code execution  
CVE-2013- 4537: OOB in SDC of QEMU, code execution  
CVE-2013- 4536: OOB in PIC of QEMU, code execution  
CVE-2013- 4532: OOB in NET of QEMU, code execution  
CVE-2013- 4529: OOB in PCI of QEMU, code execution  
CVE-2013- 4527: OOB in HPET of QEMU, code execution  
CVE-2013- 4526: OOB in AHCI of QEMU, code execution  
CVE-2013- 4151: OOB in VIRTIO of QEMU, code execution  
CVE-2013- 4150: OOB in VIRTIO-NET of QEMU, code execution  
CVE-2013- 4149: OOB in VIRTIO-NET of QEMU, code execution  
CVE-2013- 4148: OOB in VIRTIO-NET of QEMU, code execution  
**CVE-2013- 1943: Improper page handling of KVM, local privilege escalation**  
**CVE-2013- 0311: Improper descriptor handling of KVM, privilege escalation**

CVE-2021-22543: Improper page handling of KVM, privilege escalation  
CVE-2021-20181: TOCTOU in 9pfs of QEMU, privilege escalation  
CVE-2021- 3546: OOB in VIRTIO in QEMU, code execution  
CVE-2021- 3409: OOB in SDC of QEMU, code execution  
CVE-2020-35506: UAF in SCSI of QEMU, code execution  
CVE-2020-17380: OOB in SDC of QEMU, code execution  
CVE-2020-15863: OOB in NET of QEMU, code execution  
CVE-2020-14364: OOB in USB of QEMU, code execution  
CVE-2020- 1711: OOB in iSCSI of QEMU, code execution  
CVE-2019-18389: OOB in VIRTIO-GPU in QEMU, code execution  
CVE-2019-14835: OOB in KVM, privilege escalation  
CVE-2019-14821: OOB in MMIO of KVM, privilege escalation  
CVE-2019- 7221: UAF in HR timer of KVM, code execution  
CVE-2018-16882: UAF in KVM, privilege escalation  
CVE-2018-16847: OOB in NVME of QEMU, code execution  
CVE-2018-12904: Improper VMX handling of KVM, privilege escalation  
CVE-2018-10901: Improper GDT handling of KVM, privilege escalation  
CVE-2018- 1087: Improper stack switching of KVM, privilege escalation  
CVE-2017-12188: Improper page handling of KVM, code execution  
CVE-2017- 7980: OOB in display of QEMU, code execution  
CVE-2017- 5931: Integer overflow in VIRTIO of QEMU, code execution  
CVE-2017- 5667: OOB in SDC of QEMU, code execution  
CVE-2017- 2630: Stack overflow in NBD of QEMU, code execution  
CVE-2017- 2620: OOB in display of QEMU, code execution  
CVE-2017- 2615: OOB in display of QEMU, code execution  
CVE-2016-10150: UAF in KVM, local privilege escalation  
CVE-2016- 9777: Improper IOAPIC handling in KVM, privilege escalation  
CVE-2016- 9603: OOB in display of QEMU, code execution  
CVE-2016- 7161: OOB in NET of QEMU, code execution  
CVE-2016- 6351: OOB in SCSI of QEMU, code execution  
CVE-2016- 5338: OOB in SCSI of QEMU, code execution  
CVE-2016- 5126: OOB in iSCSI of QEMU, code execution  
CVD-2016- 4440: Improper APICv handling of KVM, code execution  
CVE-2016- 4439: OOB in SCSI of QEMU, code execution  
CVE-2016- 4002: OOB in NET of QEMU, code execution  
CVE-2016- 3710: OOB in VESA VGA of QEMU, code execution  
CVE-2016- 1714: OOB in NVRAM of QEMU, code execution  
CVE-2016- 1586: OOB in IDE of QEMU, code execution  
CVE-2016- 0749: OOB in SPICE of QEMU, code execution  
CVE-2015- 7512: OOB in NET of QEMU, code execution  
CVE-2015- 7504: OOB in NET of QEMU, code execution  
CVE-2015- 5279: OOB in NET of QEMU, code execution  
CVE-2015- 5278: OOB in NET of QEMU, code execution  
CVE-2015- 5260: OOB in SPICE of QEMU, code execution  
OOB in VNC of QEMU, code execution  
OOB in IDE of QEMU, code execution  
OOB in FDC of QEMU (VENOM), code execution  
OOB in PIT of QEMU, code execution  
OOB in NET of QEMU, code execution  
OOB in display of QEMU, code execution  
OOB in migration function of QEMU, code execution  
OOB in USB of QEMU, code execution  
OOB in VIRTIO of QEMU, code execution  
Integer overflow in VIRTIO-NET of QEMU, code execution  
OOB in CLOOP of QEMU, code execution  
OOB in KVM, code execution  
OOB in VIRTIO of QEMU, code execution  
OOB in NET of QEMU, code execution  
OOB in KVM, local privilege escalation  
OOB in SCSI of QEMU, code execution  
OOB in USB of QEMU, code execution  
OOB in GPIO of QEMU, code execution  
OOB in Audio of QEMU, code execution  
OOB in Display of QEMU, code execution  
OOB in SDC of QEMU, code execution  
CVE-2013- 4536: OOB in PIC of QEMU, code execution  
CVE-2013- 4532: OOB in NET of QEMU, code execution  
CVE-2013- 4529: OOB in PCI of QEMU, code execution  
CVE-2013- 4527: OOB in HPET of QEMU, code execution  
CVE-2013- 4526: OOB in AHCI of QEMU, code execution  
CVE-2013- 4151: OOB in VIRTIO of QEMU, code execution  
CVE-2013- 4150: OOB in VIRTIO-NET of QEMU, code execution  
CVE-2013- 4149: OOB in VIRTIO-NET of QEMU, code execution  
CVE-2013- 4148: OOB in VIRTIO-NET of QEMU, code execution  
CVE-2013- 1943: Improper page handling of KVM, local privilege escalation  
CVE-2013- 0311: Improper descriptor handling of KVM, privilege escalation

**PLEASE TELL ME**

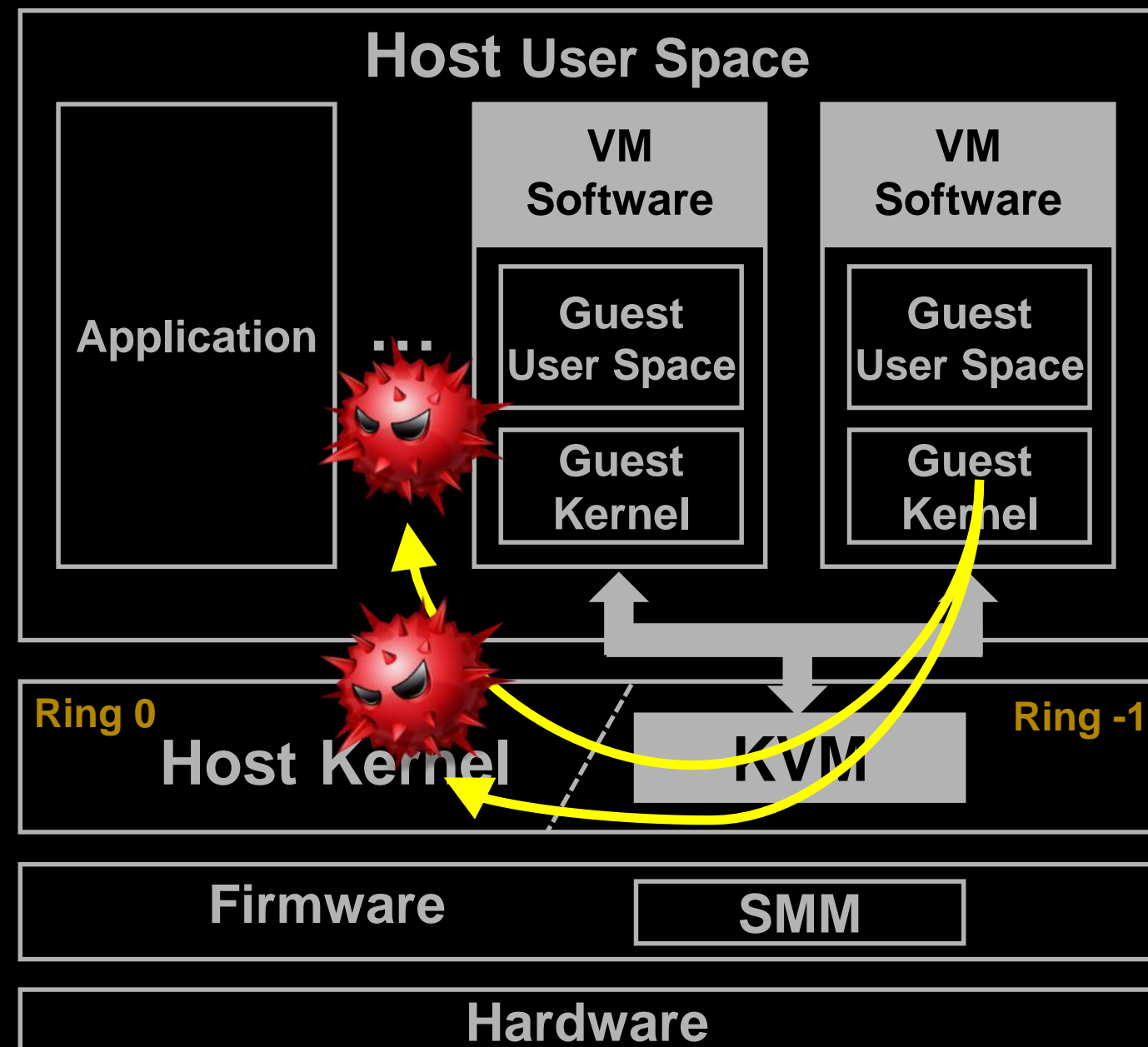


**WHERE AM I, HUH?**

# Extracting Escape Paths from CVEs (1)

## Code Execution with a **KVM** Vulnerability

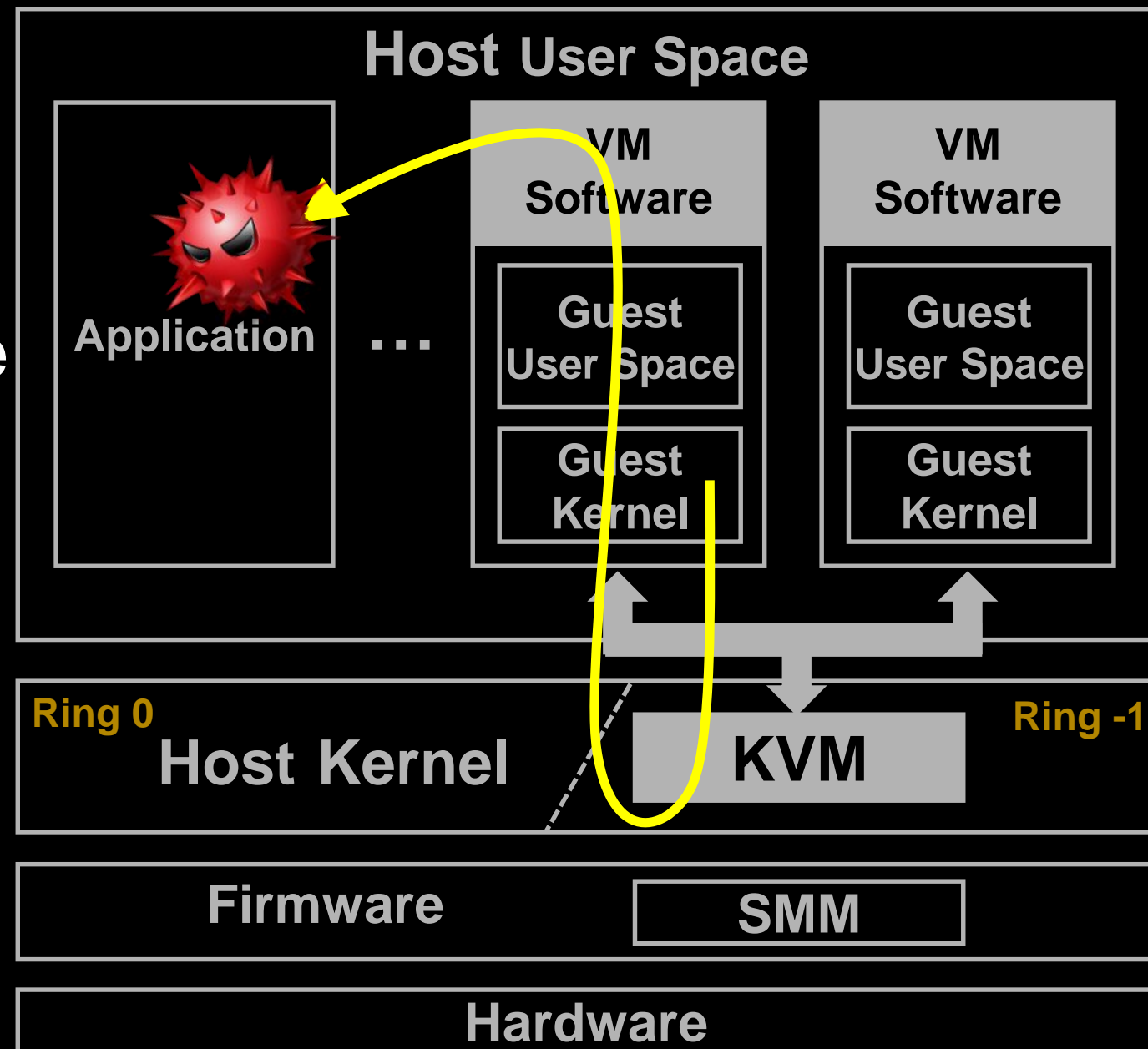
- Executes code outside of a VM
  - Small shellcode is used for creating a **ROOT** privilege process of a host
    - Process creation: Guest → Host
    - Privilege escalation: Host user → root
- It is also possible to modify kernel or KVM code
  - But, disabling security features are needed! 😞



# Extracting Escape Paths from CVEs (2)

## Code Execution with a VM Vulnerability

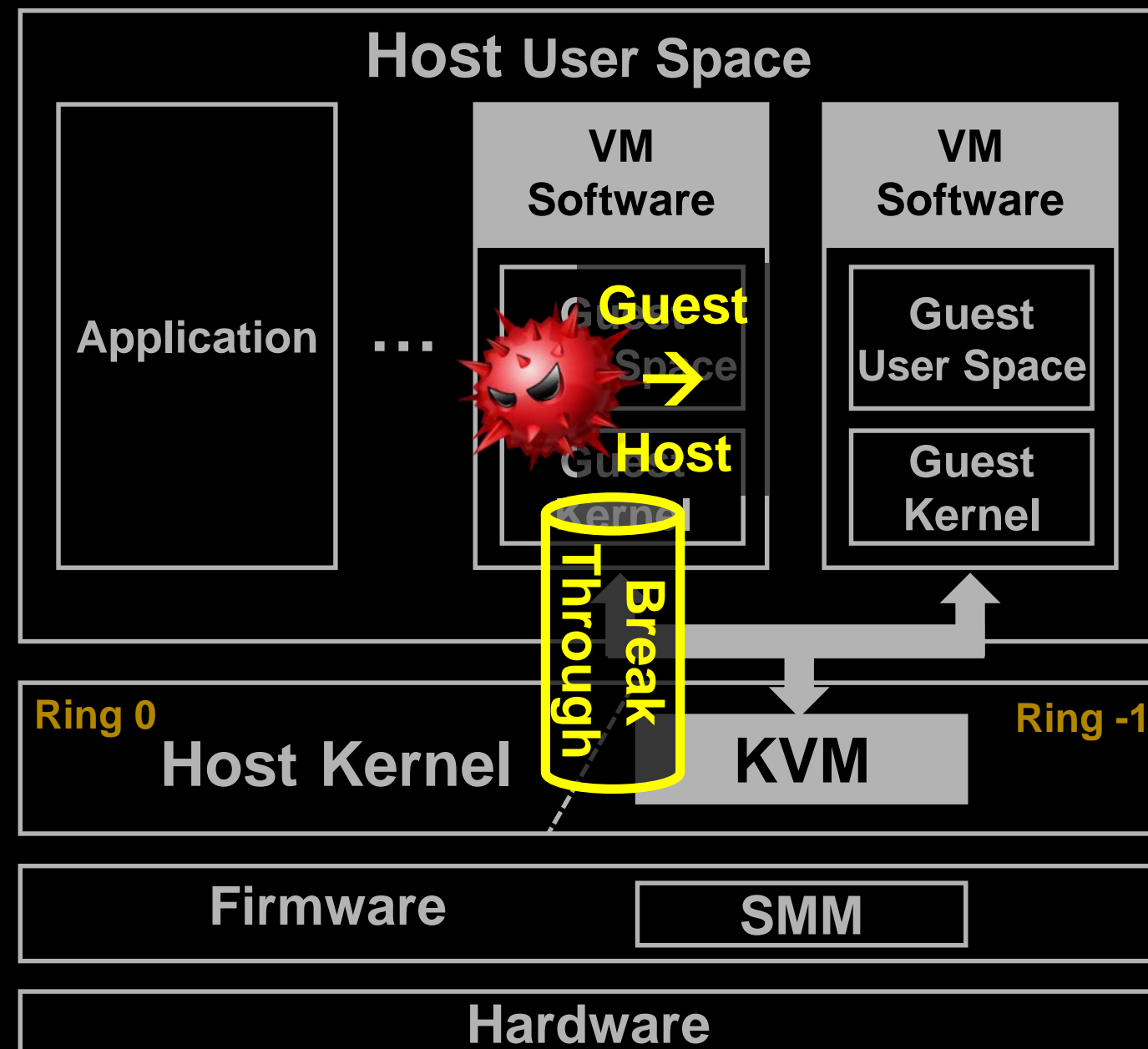
- Also executes code outside of a VM
  - A VM software has lots of code for services and hardware emulations
    - Full-featured VMs (QEMU) has a large code base
    - Others have smaller ones, but ... 😞
  - Small shellcode is used for creating a USER privilege process of a host
    - Process creation: Guest → Host
    - Privilege escalation is needed!



# Extracting Escape Paths from CVEs (3)

## Privilege Escalation with a **KVM** Vulnerability

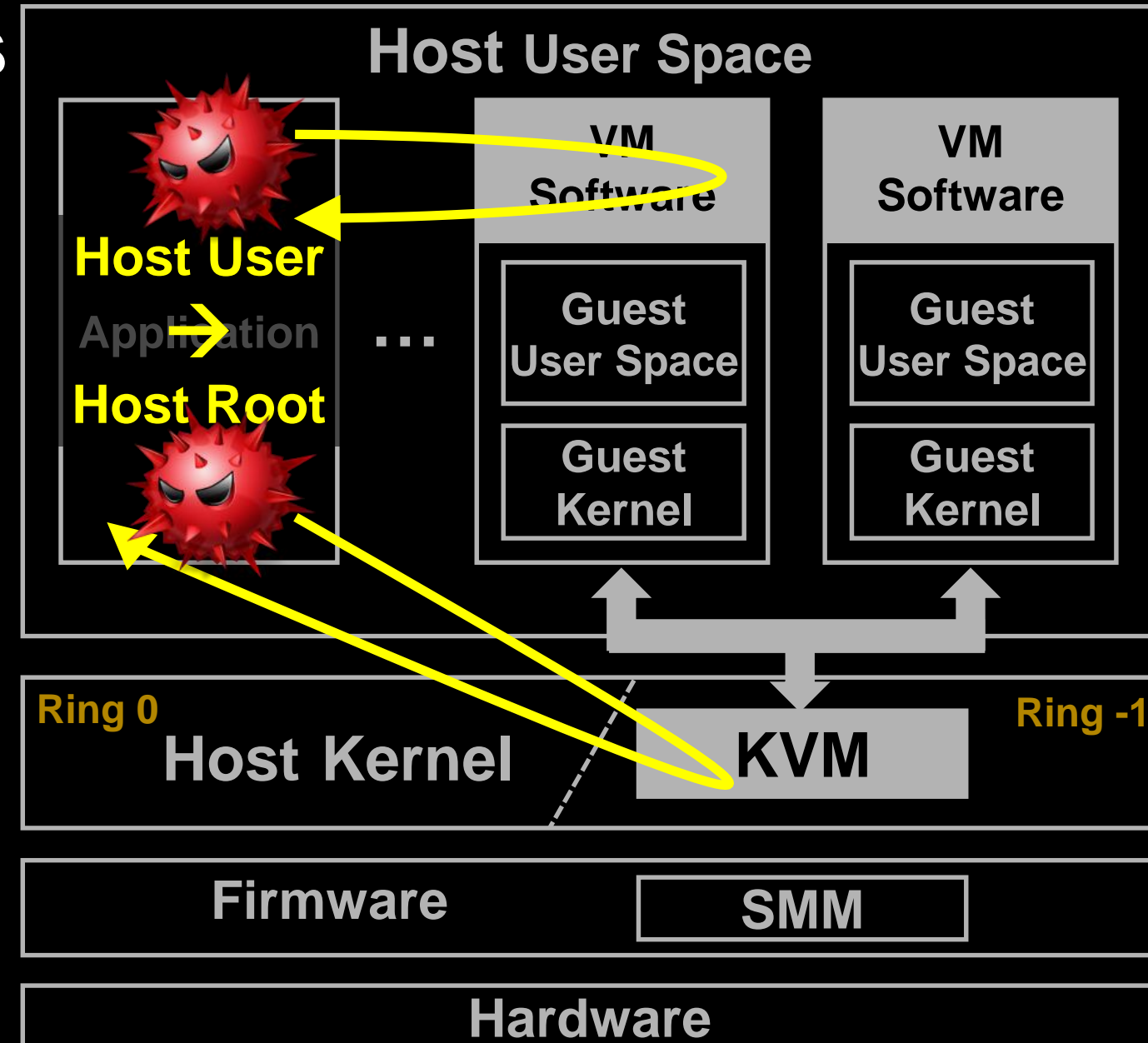
- Elevates the privilege of a VM
  - A VM is a process and runs in the guest privilege of a host
  - But, **improper handlings give HOST privilege to the VM**
    - Page tables, interrupts, registers, etc.
- It is possible to create a root privilege process of a host and modify kernel code



# Extracting Escape Paths from CVEs (4)

## Local Privilege Escalation with a KVM and VM Vuln.

- Elevates the privilege of a process
  - Some KVM vulnerabilities can be exploited through `/dev/kvm`
    - It can be used to elevate privilege (and create a root privilege process)
  - Some VM vulnerabilities can be used for privilege escalation if the VM or related daemon runs in root privilege
    - But, a VM usually runs in user privilege for security reasons 😞





## 3d Red Pill

# A Guest-to-Host Escape on QEMU/KVM Virtio Device

College of Cyber Security Jinan University  
Zhijian Shao, Jian Weng, Yue Zhang

OOB in VIRTIO-GPU of **QEMU**

## VENOM Vulnerability: Community Patching and Mitigation Update

May 13, 2015 Editorial Team Executive Viewpoint



OOB in FDC of **QEMU**

## Escape From The Docker-KVM-QEMU Machine

Shengping Wang, Xu Liu  
Qihoo 360 Marvel Team

OOB in NET of **QEMU**

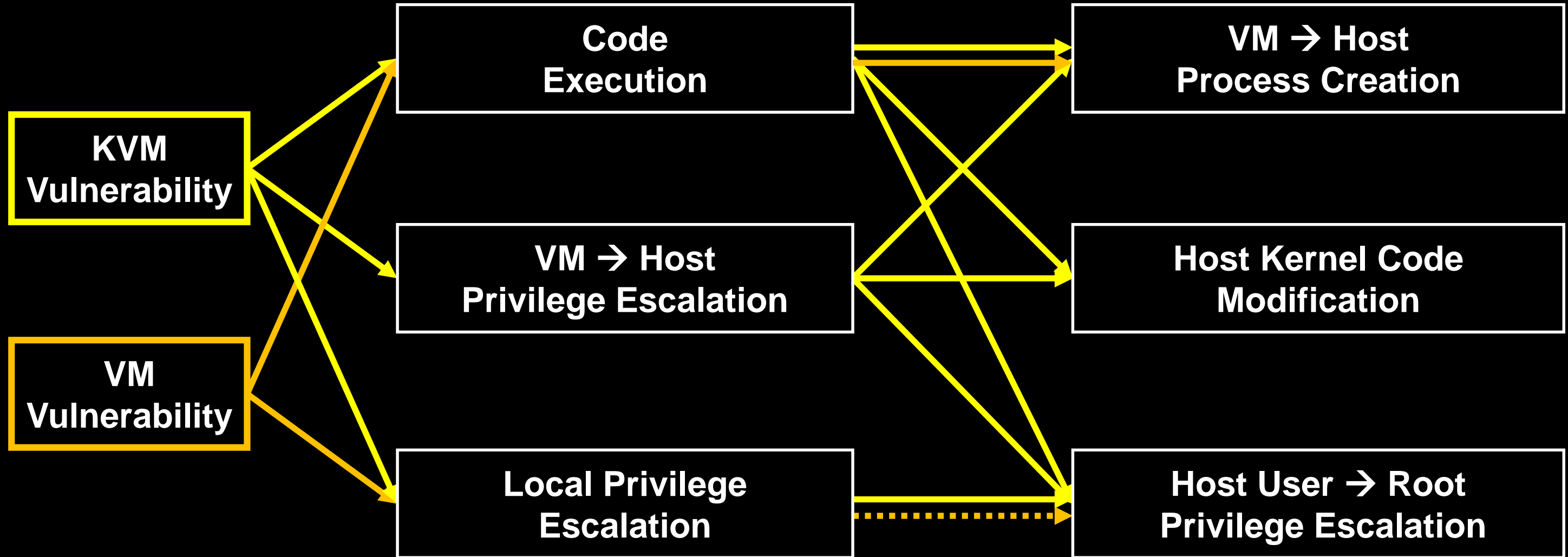
## Virtunoid: Breaking out of KVM

Nelson Elhage

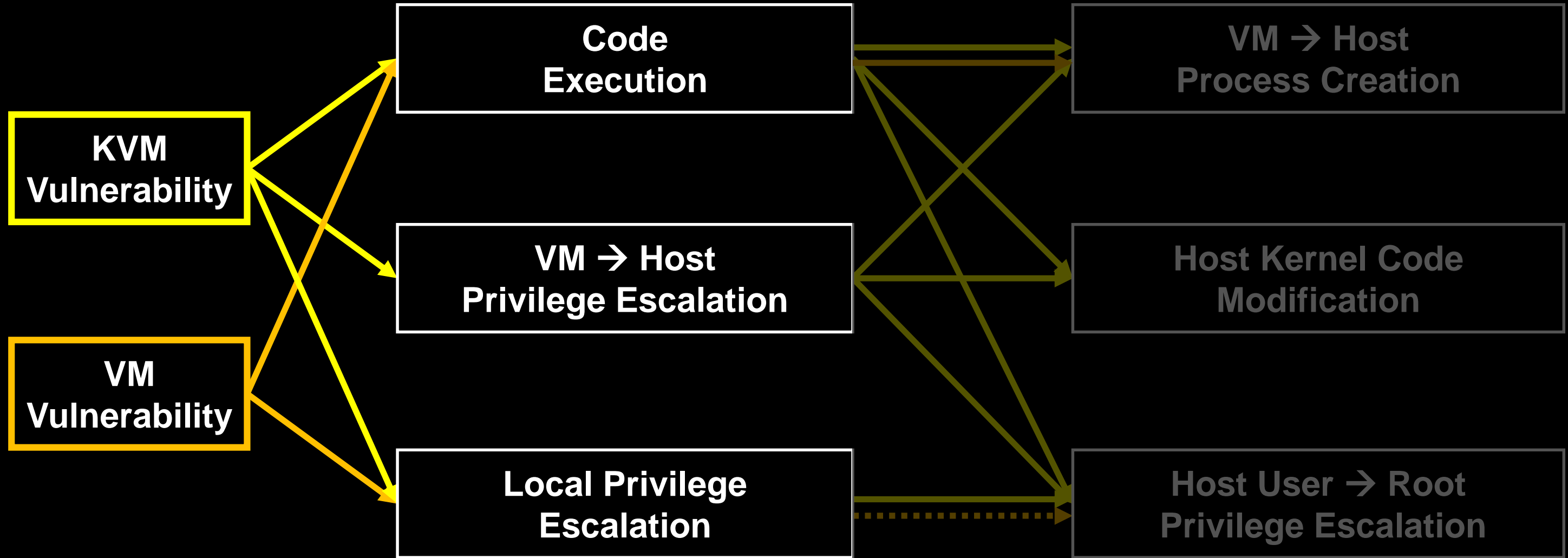
July 25, 2011

UAF in Timer of **QEMU**

# So, critical paths of escapes are ...



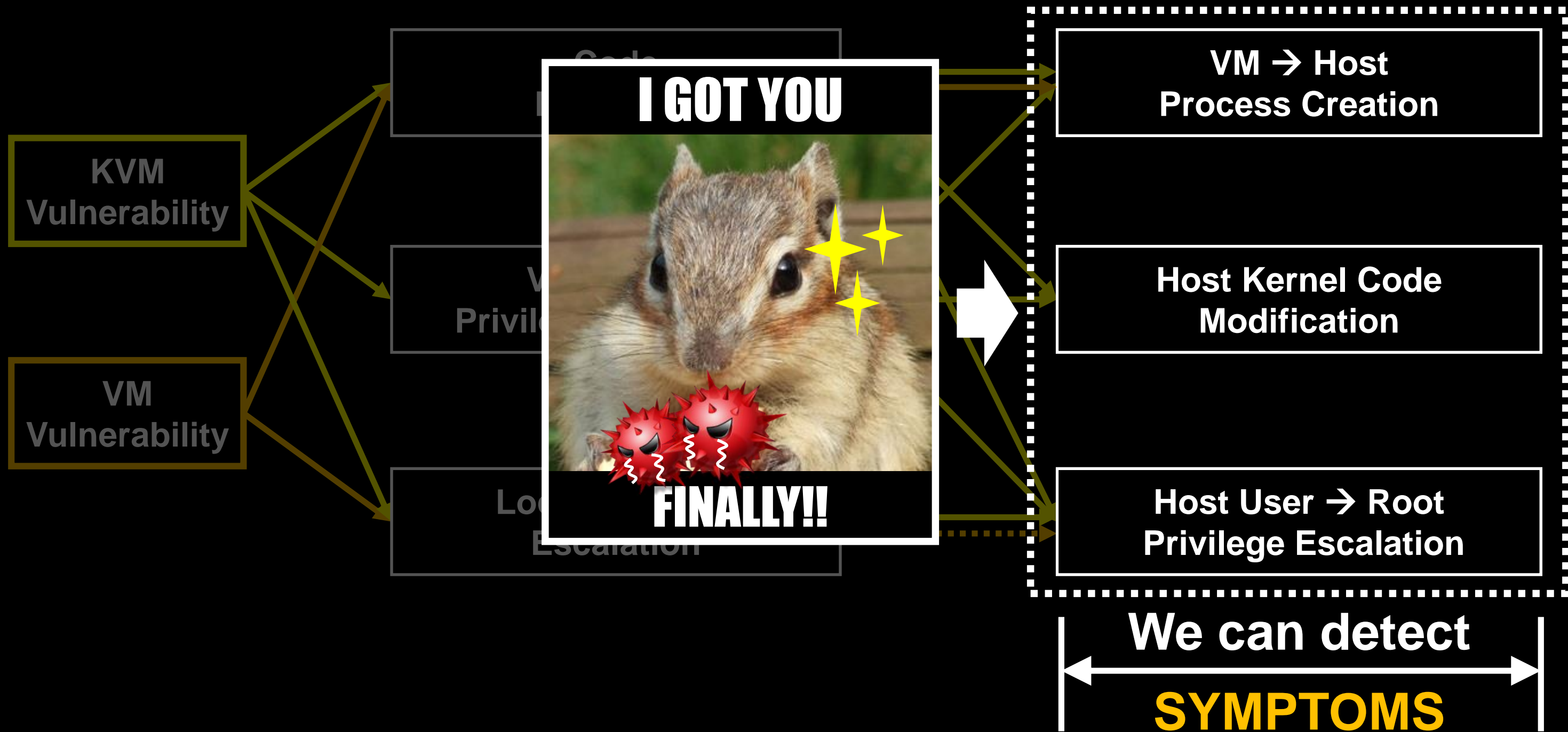
# So, critical paths of escapes are ...



**No one knows**

← until the vulnerability is revealed →

# So, critical paths of escapes are ...



- Background
- Analysis of Critical Paths of Escapes
- **Design and Implementation of Alcatraz**
- Evaluation and Demo
- Conclusion and Black Hat Sound Bytes



# What is Alcatraz?

- Alcatraz is a practical hypervisor sandbox to prevent escapes

- It does not need SMM and retrofits of the hypervisor



- It downgrades KVM's privilege and makes a sandbox with Intel VT

- Alcatraz consists of **Hyper-box** and **tailored Linux kernel**

- Hyper-box is a pico hypervisor made from scratch

- It has core mechanisms for preventing escapes

- Tailored Linux kernel has only one system call interface for x86\_64

- It narrows the attack surface and monitoring points



# Hyper-Box (1)

- **Makes a sandbox with the memory and register protection techniques**
  - It leverages the Extended Page Table (EPT) of Intel VT
    - Guest Physical Address (PA) are translated to host PA with EPT
    - **EPT can prevent code and RO data modifications!**
  - It protects security-related bits of control registers (CRs)
    - Write Protect (**WP**) of CR0, Supervisor Mode Execution Protection (**SMEP**) of CR4
- **Monitors system calls and prevents unauthorized behaviors**
  - It sets **H/W breakpoints** to the system call entry point, fork, exit, and cred functions
    - `wake_up_new_task()`, `cgroup_release()`, `entry_SYSCALL_64()`, and `commit_creds()`
  - It detects and prevents unauthorized behaviors
    - **code modification, process creation, and privilege escalation**

# Hyper-Box (2)

- **Emulates VMX instructions for nested virtualization**
  - It downgrades KVM's privilege (Ring -1) to a guest hypervisor (Ring 0)
  - So, It has to emulate VMX instructions of KVM because Ring 0 cannot execute them
    - However, performance overhead is high
- **Uses VT features to reduce the performance overhead**
  - Virtual Machine Control Structure (VMCS) shadowing
    - It allows a guest hypervisor to execute VMREAD and VMWRITE directly
    - **So, emulations or interventions are reduced!**
  - Virtual Process ID (VPID)
    - It gives unique ID to vCPU and Translation Lookaside Buffer (TLB)
    - **It prevents TLB flushes when VMExit occurs!**



# Hyper-Box (2)

- Emulates VMX instructions
  - It downgrades KVM's privilege
  - So, It has to emulate VMX instructions
  - However, performance
- Uses VT features to reduce performance overhead
  - Virtual Machine Control Structure (VMCS)
  - It allows a guest hypervisor to read and write VMCS fields
  - So, emulations or interventions
  - Virtual Process ID (VPID)
    - It gives unique ID to vCPU and Translation Lookaside Buffer (TLB)
    - It prevents TLB flushes when VMExit occurs!

AGAIN, WHERE AM I?



GIVE ME PICTURES!

Virtualization

most hypervisor (Ring 0)

because Ring 0 cannot execute them

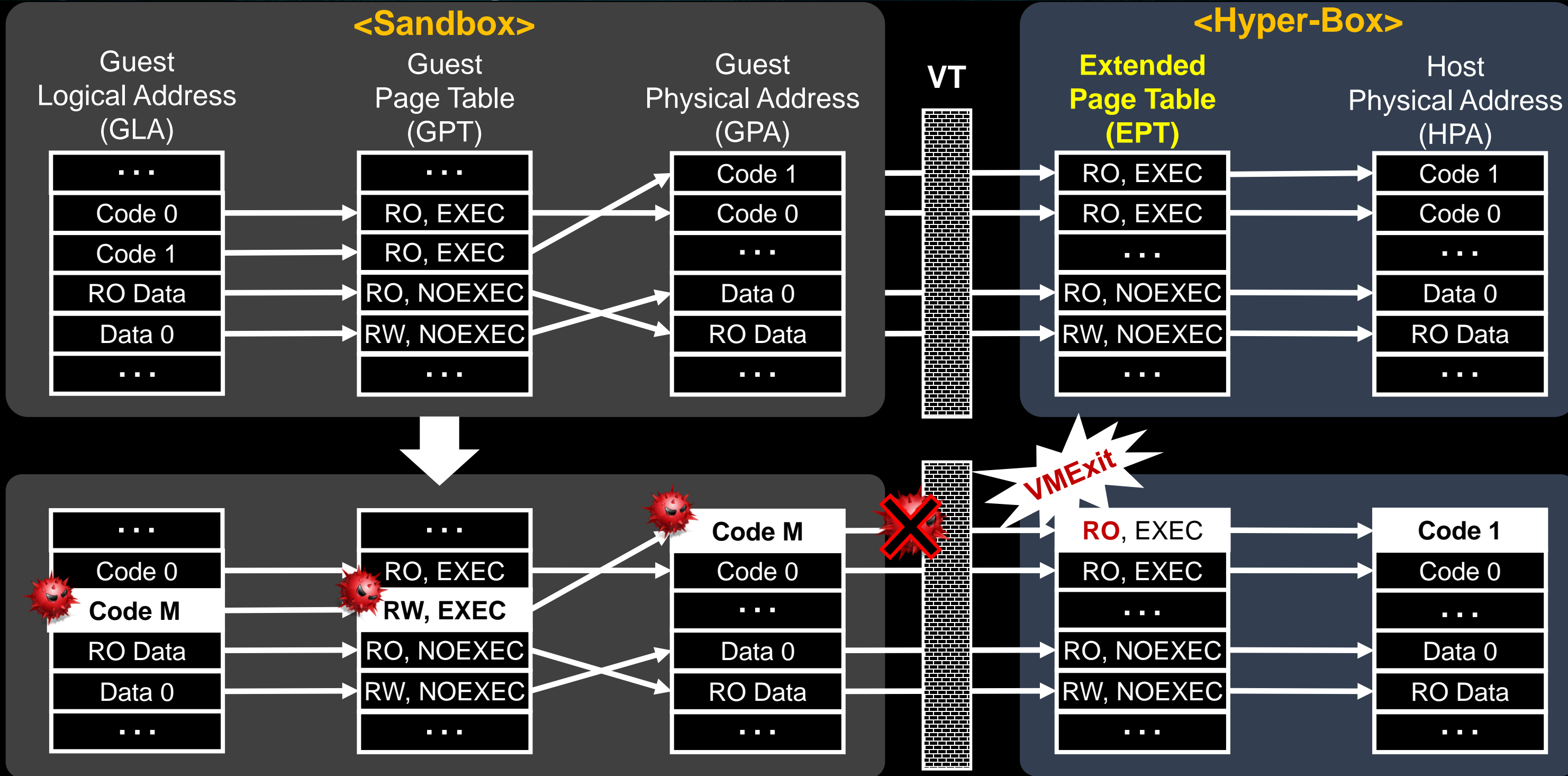
performance overhead

allowing

VMREAD and VMWRITE directly

# Hyper-Box Design

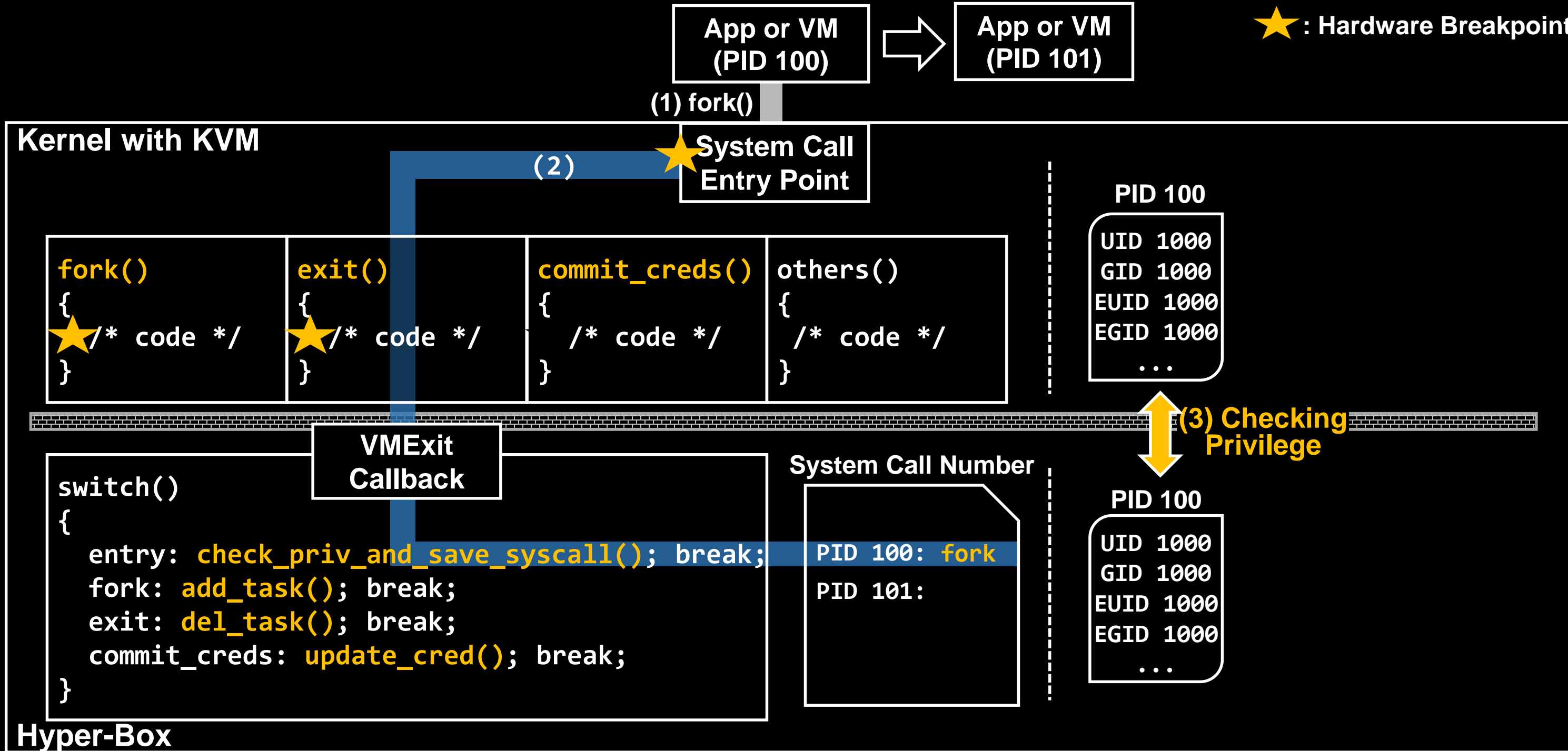
## Preventing Unauthorized Code Modification



# Hyper-Box Design

## Preventing Unauthorized Process Creation (1)

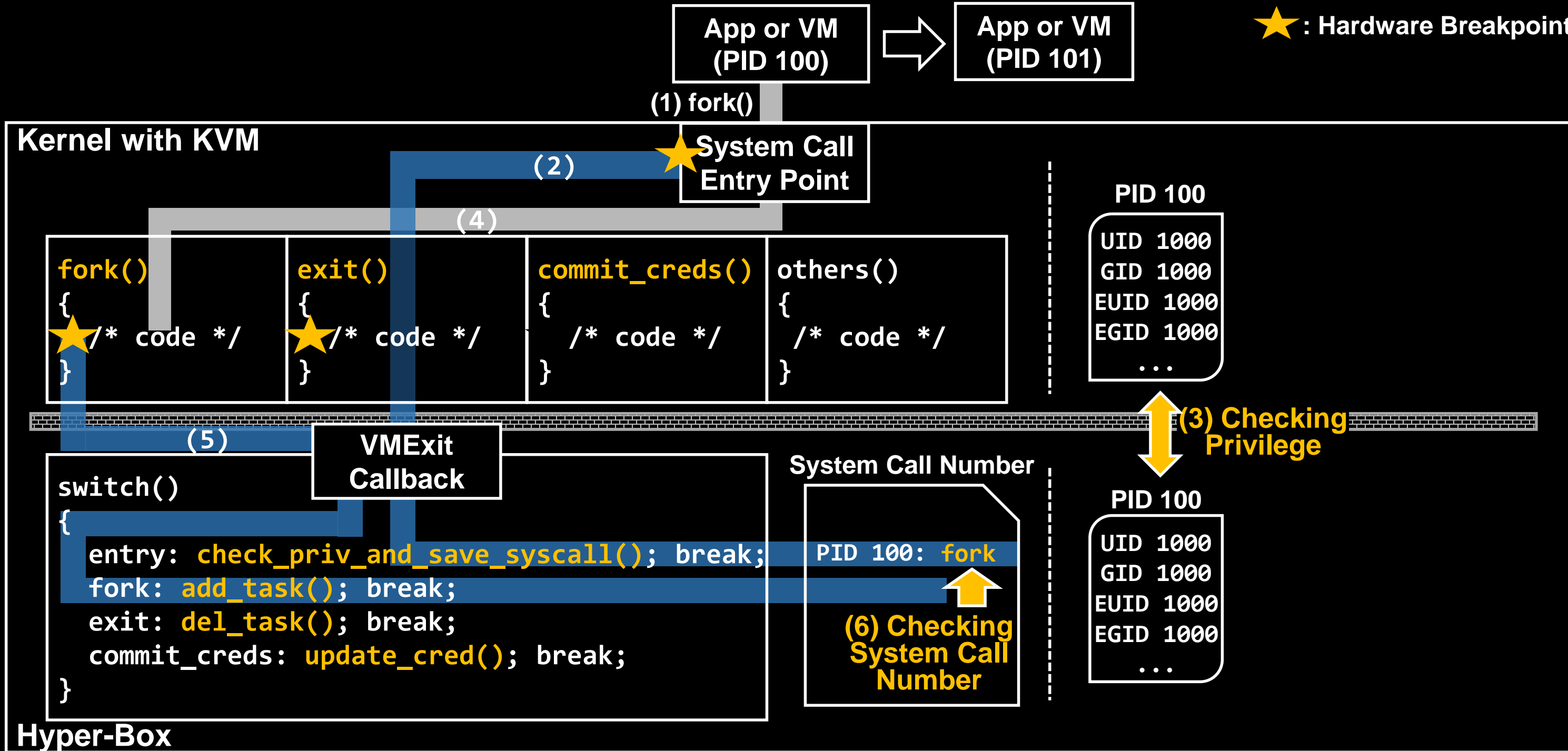
★ : Hardware Breakpoint



# Hyper-Box Design

## Preventing Unauthorized Process Creation (1)

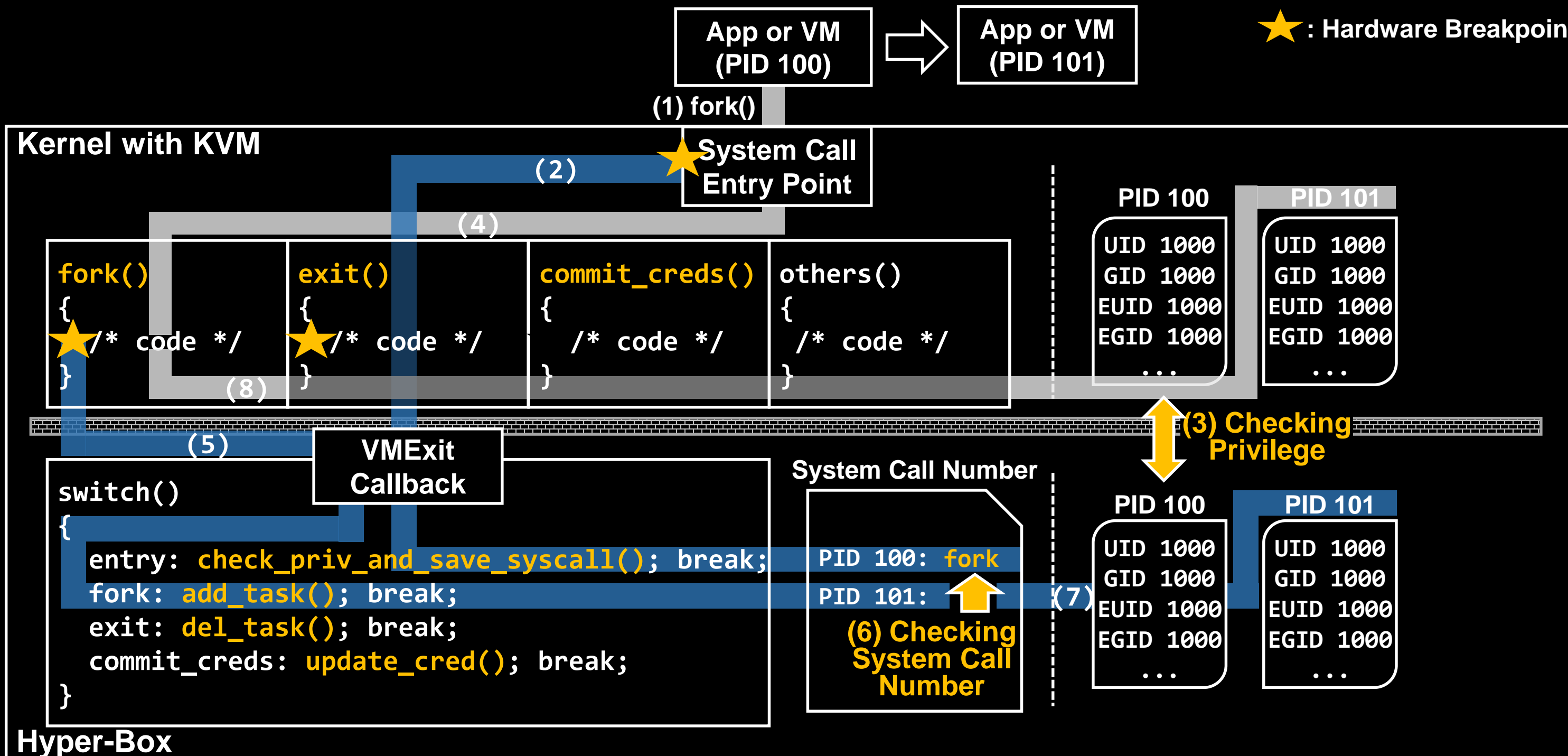
★ : Hardware Breakpoint



# Hyper-Box Design

## Preventing Unauthorized Process Creation (1)

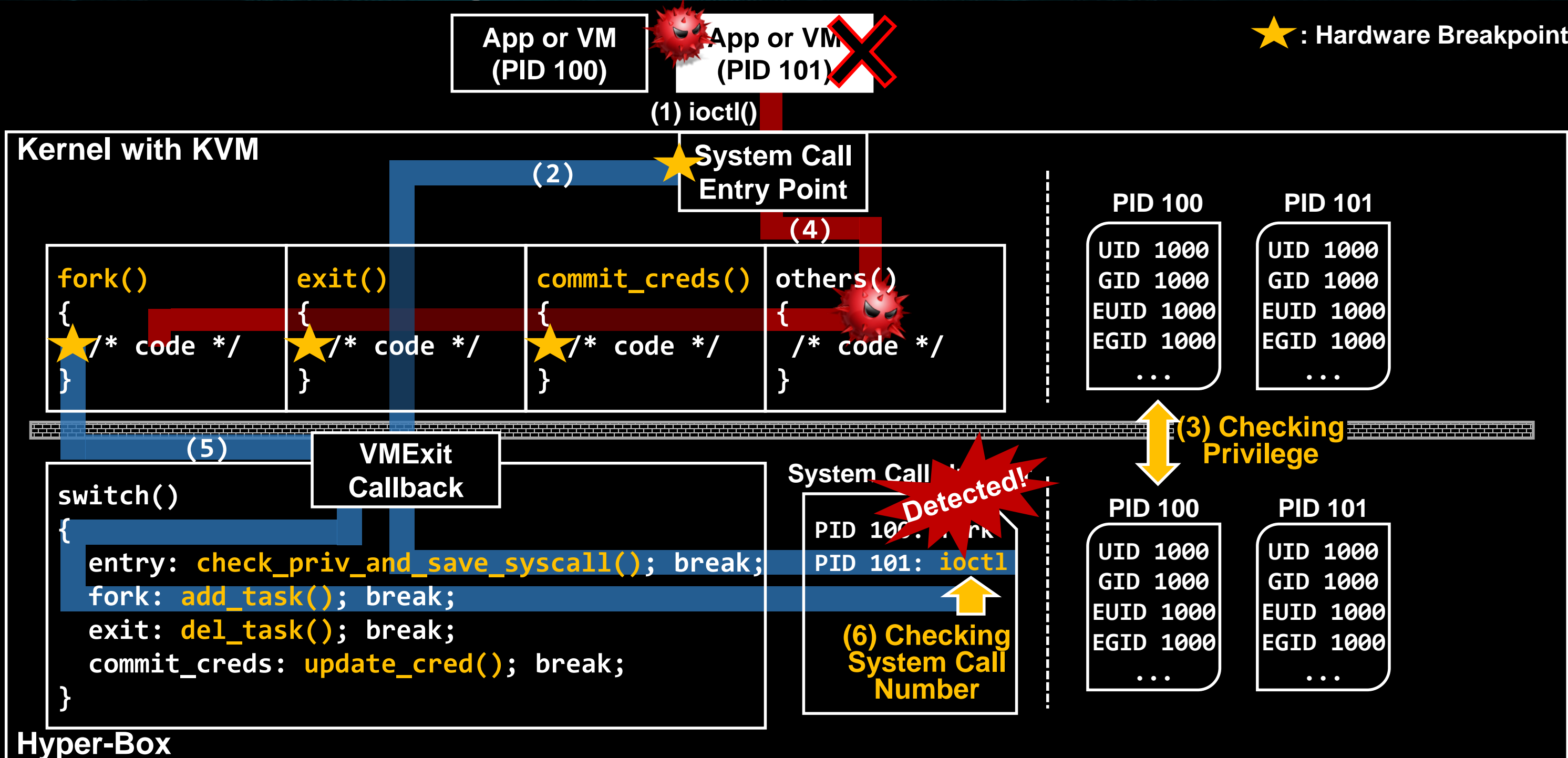
★ : Hardware Breakpoint



# Hyper-Box Design

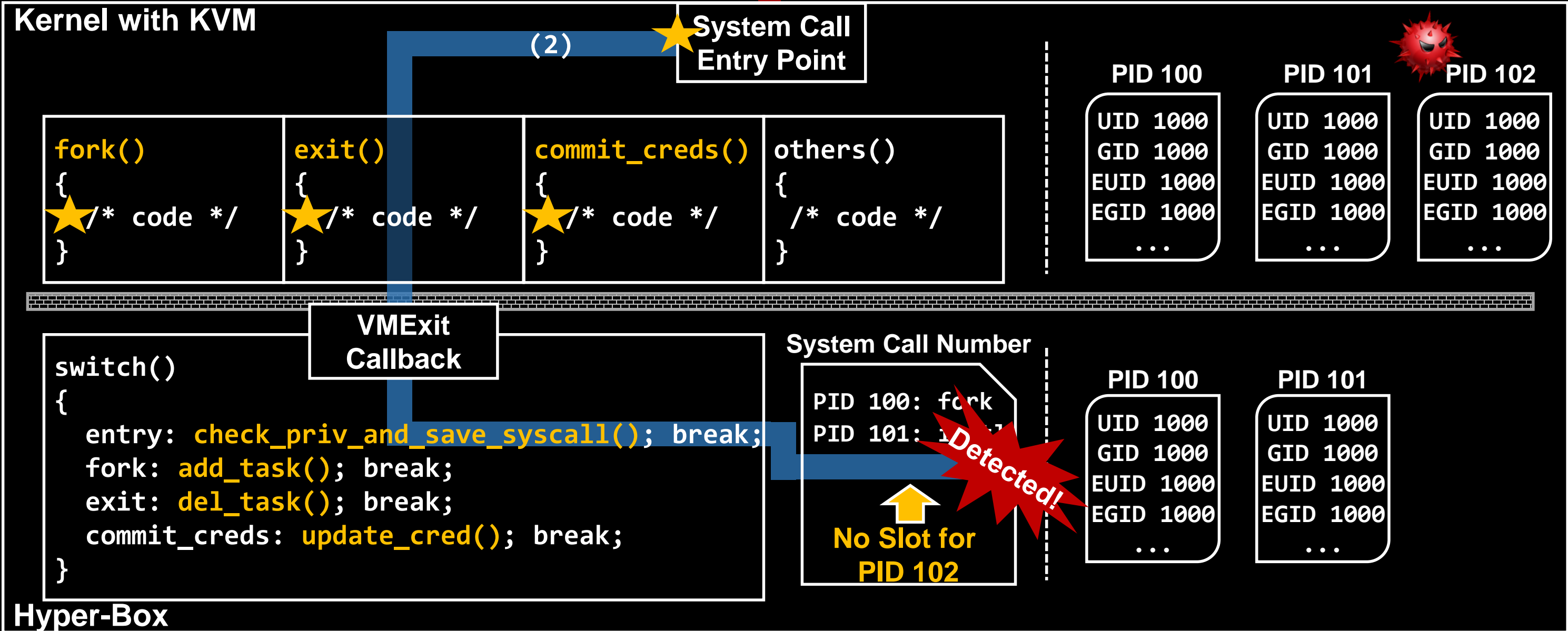
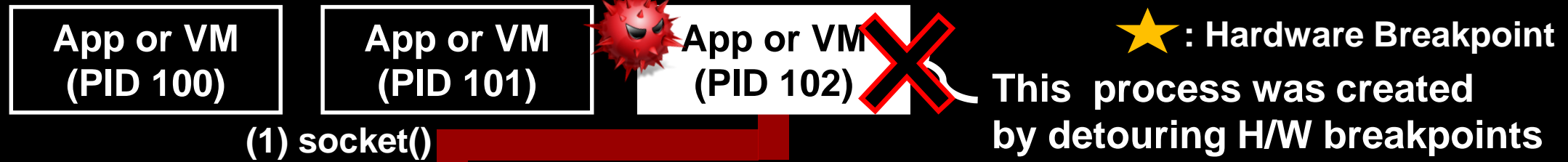
## Preventing Unauthorized Process Creation (2)

★ : Hardware Breakpoint



# Hyper-Box Design

## Preventing Unauthorized Process Creation (3)



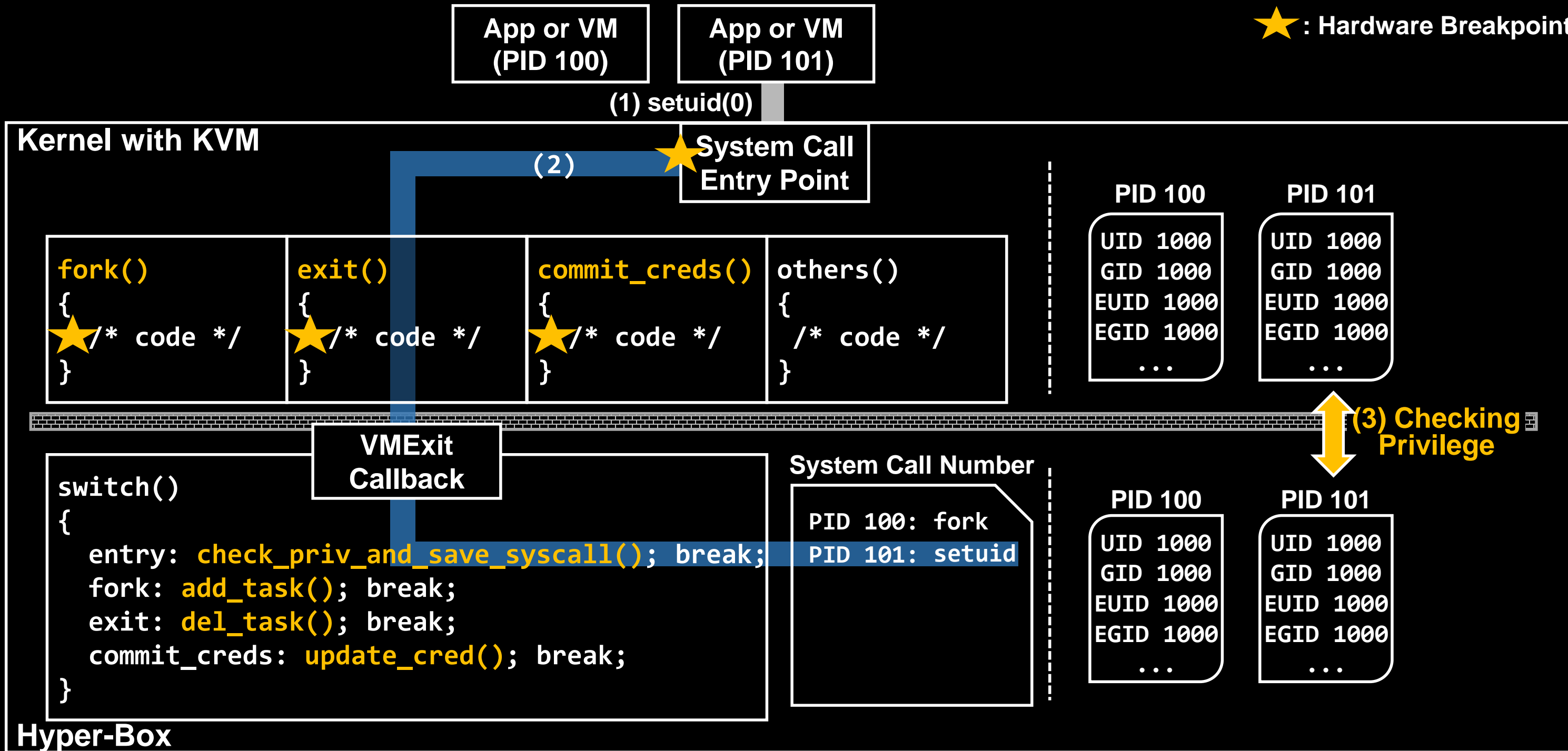
- **Some processes are not allowed to create a child process**
  - QEMU process cannot create it for security reasons
  - Hyper-box has a disallowed list that contains process names
    - **It checks the list for every system call and blocks the request**
- **Some kernel drivers load other modules with a kernel thread**
  - They call `call_usermodehelper()`, and the function creates a kernel thread
  - The kernel thread executes `modprobe (/usr/bin/kmod)` with kernel API
    - It means the system call number is unknown, but `modprobe` is created!
  - Hyper-box has a workaround feature that allows the kernel thread to execute a process
    - But, the feature is only for logging and not recommended
    - **Load all necessary modules before Alcatraz or include modules to the kernel!**



# Hyper-Box Design

## Preventing Unauthorized Privilege Escalation (1)

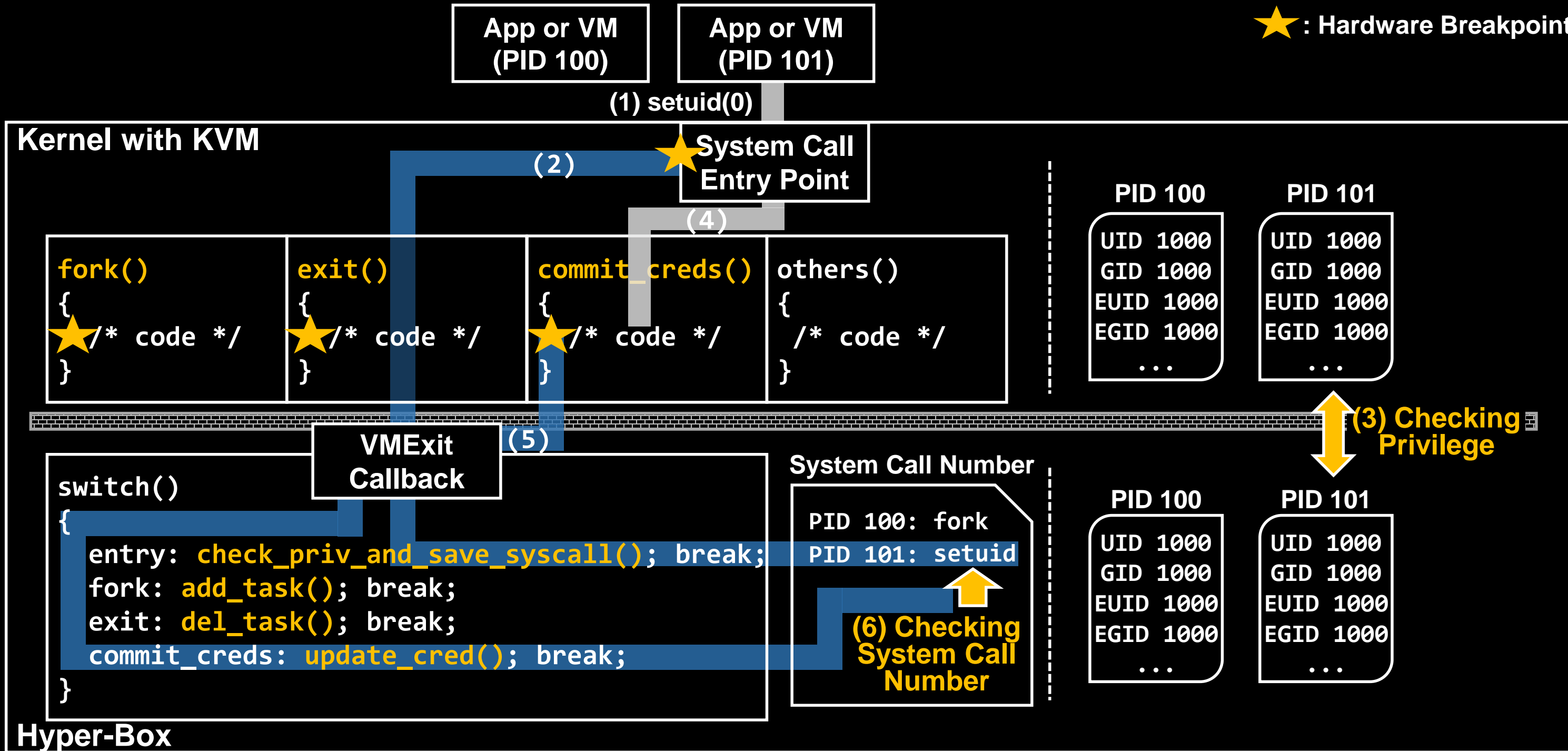
★ : Hardware Breakpoint



# Hyper-Box Design

## Preventing Unauthorized Privilege Escalation (1)

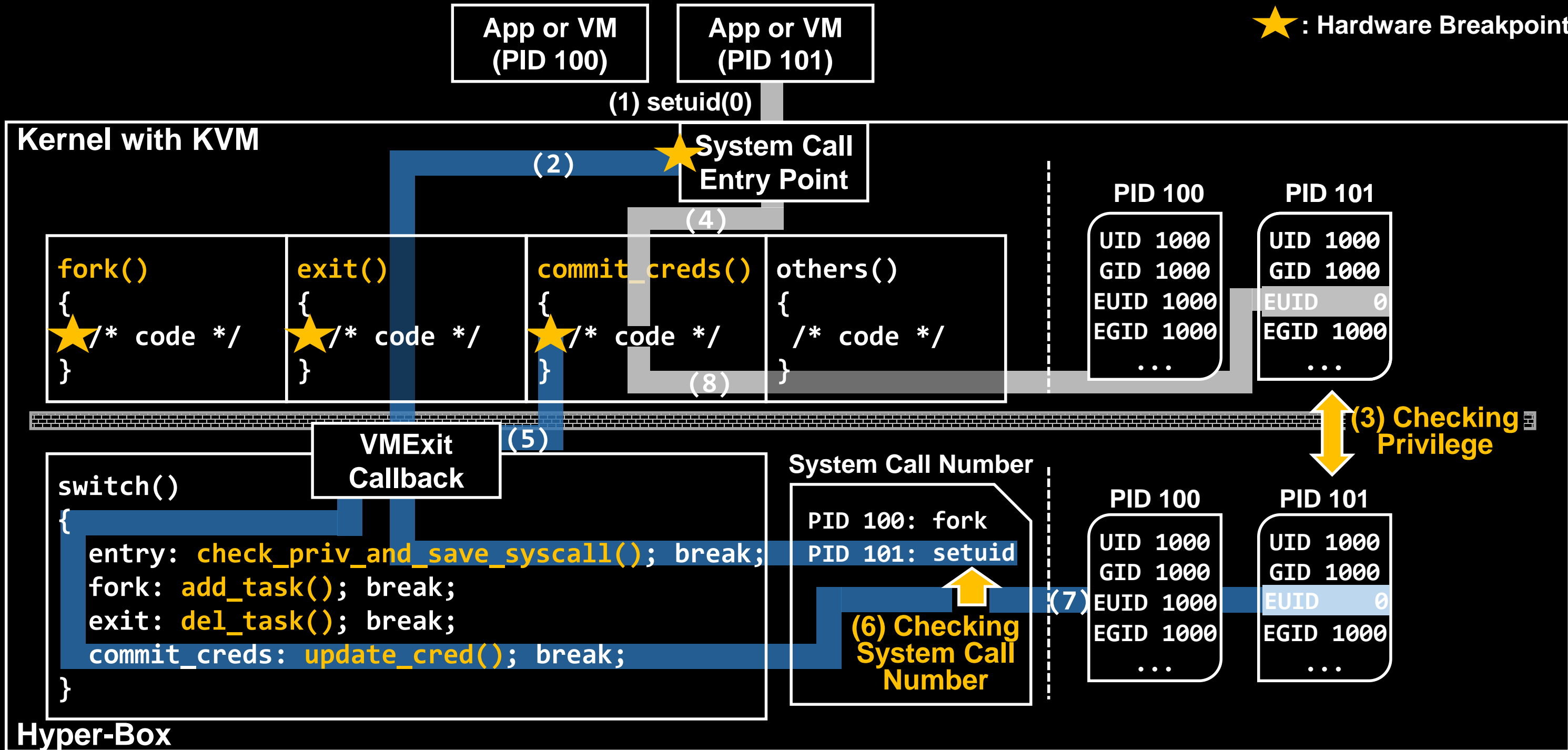
★ : Hardware Breakpoint



# Hyper-Box Design

## Preventing Unauthorized Privilege Escalation (1)

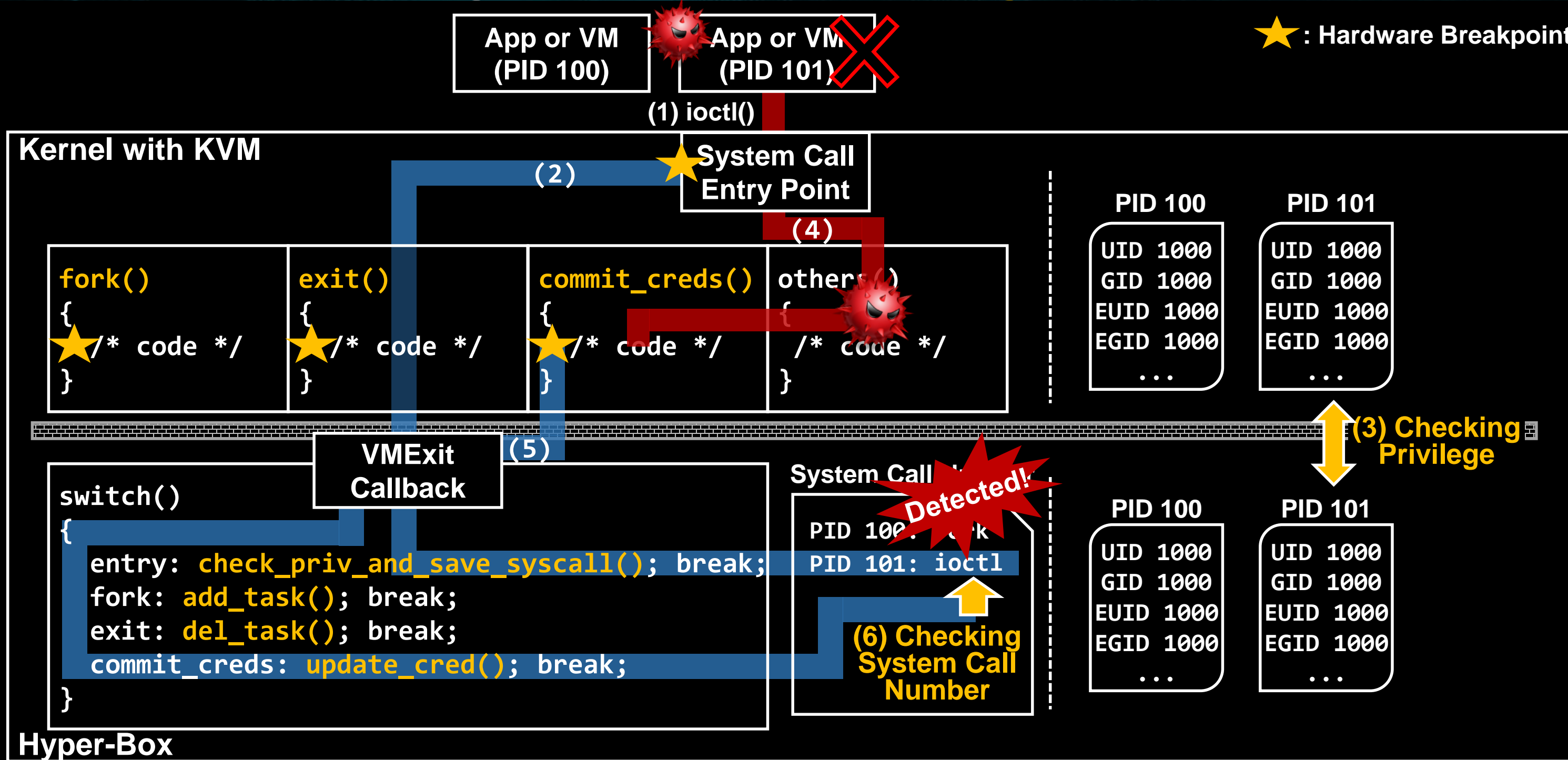
★ : Hardware Breakpoint



# Hyper-Box Design

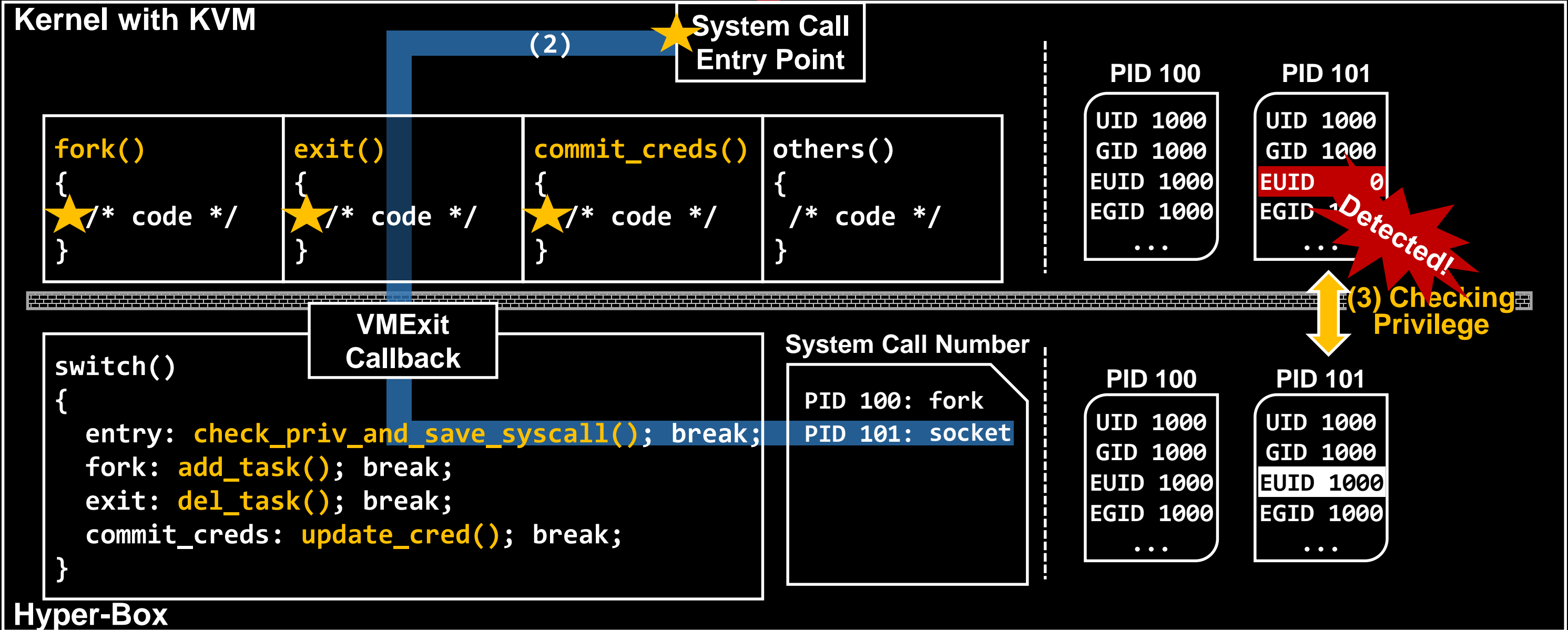
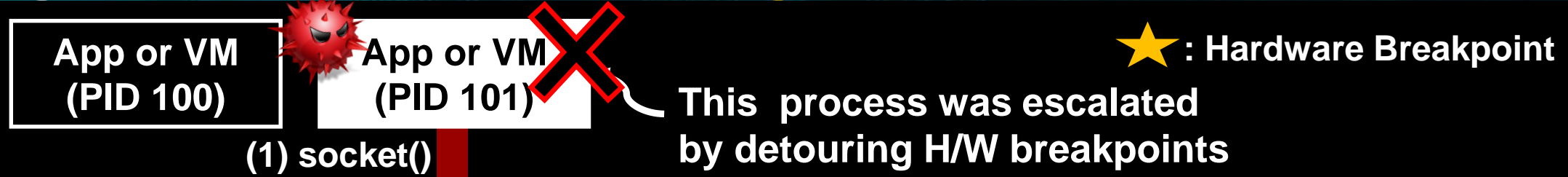
## Preventing Unauthorized Privilege Escalation (2)

★ : Hardware Breakpoint



# Hyper-Box Design

## Preventing Unauthorized Privilege Escalation (3)



# Hyper-Box Design

## Emulating VMX instructions – Only KVM

Kernel with KVM

VM 1

VM 2

Ring 0 → Ring -1 (VMXON of KVM)

(1) VMXON, VMXOFF

(3) VMExit

**VMCS 1**  
VM\_HOST\_RIP: 0xAAAABBBB  
VM\_HOST\_RSP: 0x11111111

**VMCS 2**  
VM\_HOST\_RIP: 0xAAAABBBB  
VM\_HOST\_RSP: 0x22222222

**KVM (Guest Hypervisor)**  
0xAAAABBBB: VMExitHandler  
{/\* Processing Events \*/}

(2) VMPTRLD, VMPTRST, VMCLEAR, VMREAD, VMWRITE, VMLAUNCH, VMRESUME, INVEPT, INVVPID, INVPCID, etc.

**VMCS 0 (Sandbox)**

VM\_HOST\_RIP : 0xEEEEEEFFF  
VM\_HOST\_RSP : 0x0000FFFF  
VM\_GUEST\_RIP: 0x00000000  
VM\_GUEST\_RSP: 0x00000000

**VMCS Data List**

VMCS 1-RIP: 0xAAAABBBB  
VMCS 1-RSP: 0x11111111  
VMCS 2-RIP: 0xAAAABBBB  
VMCS 2-RSP: 0x22222222

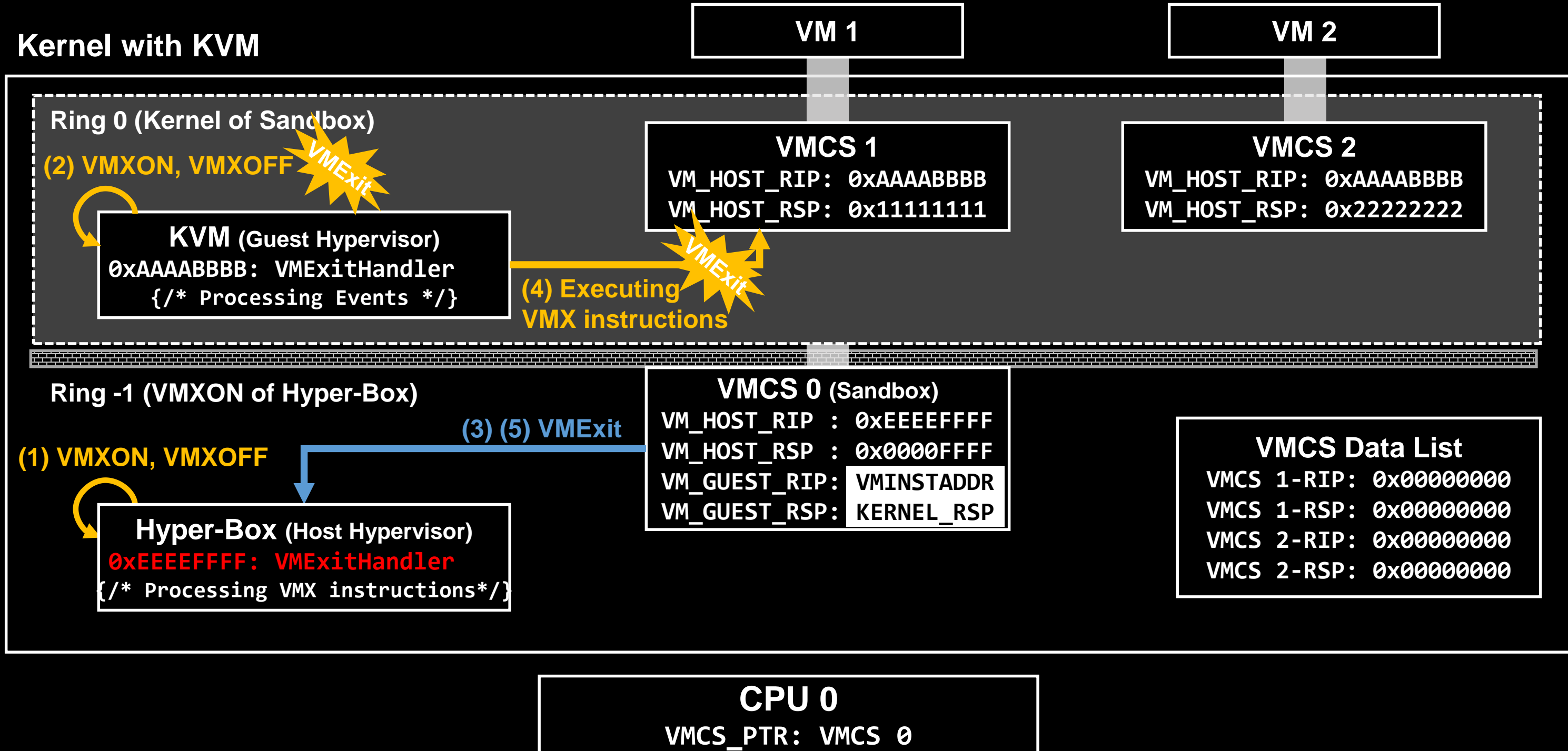
**Hyper-Box (Host Hypervisor)**  
0xEEEEEEFFF: VMExitHandler  
{/\* Processing VMX instructions\*/}

**CPU 0**

VMCS\_PTR: VMCS 1 or 2

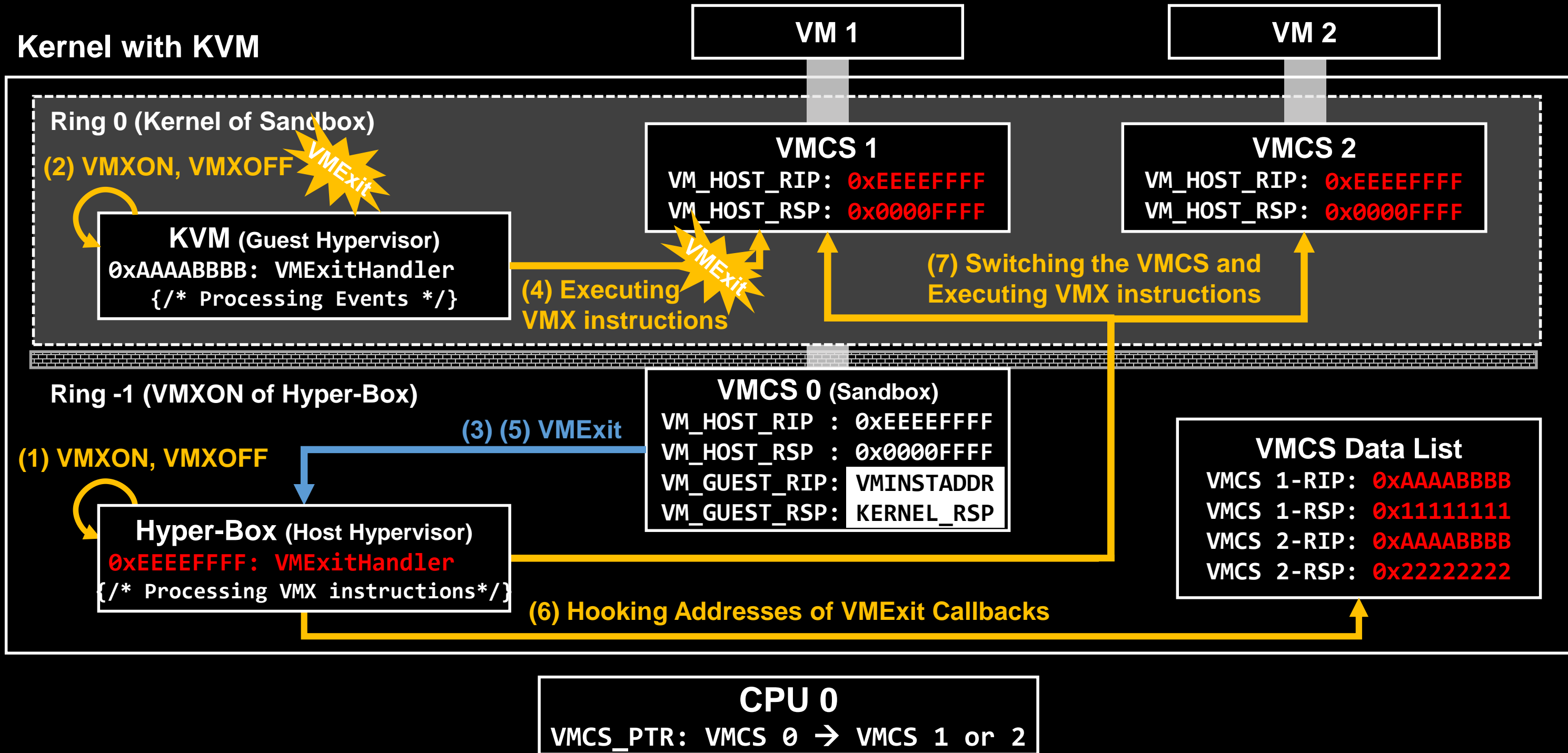
# Hyper-Box Design

## Emulating VMX instructions – Executing Instructions (1)



# Hyper-Box Design

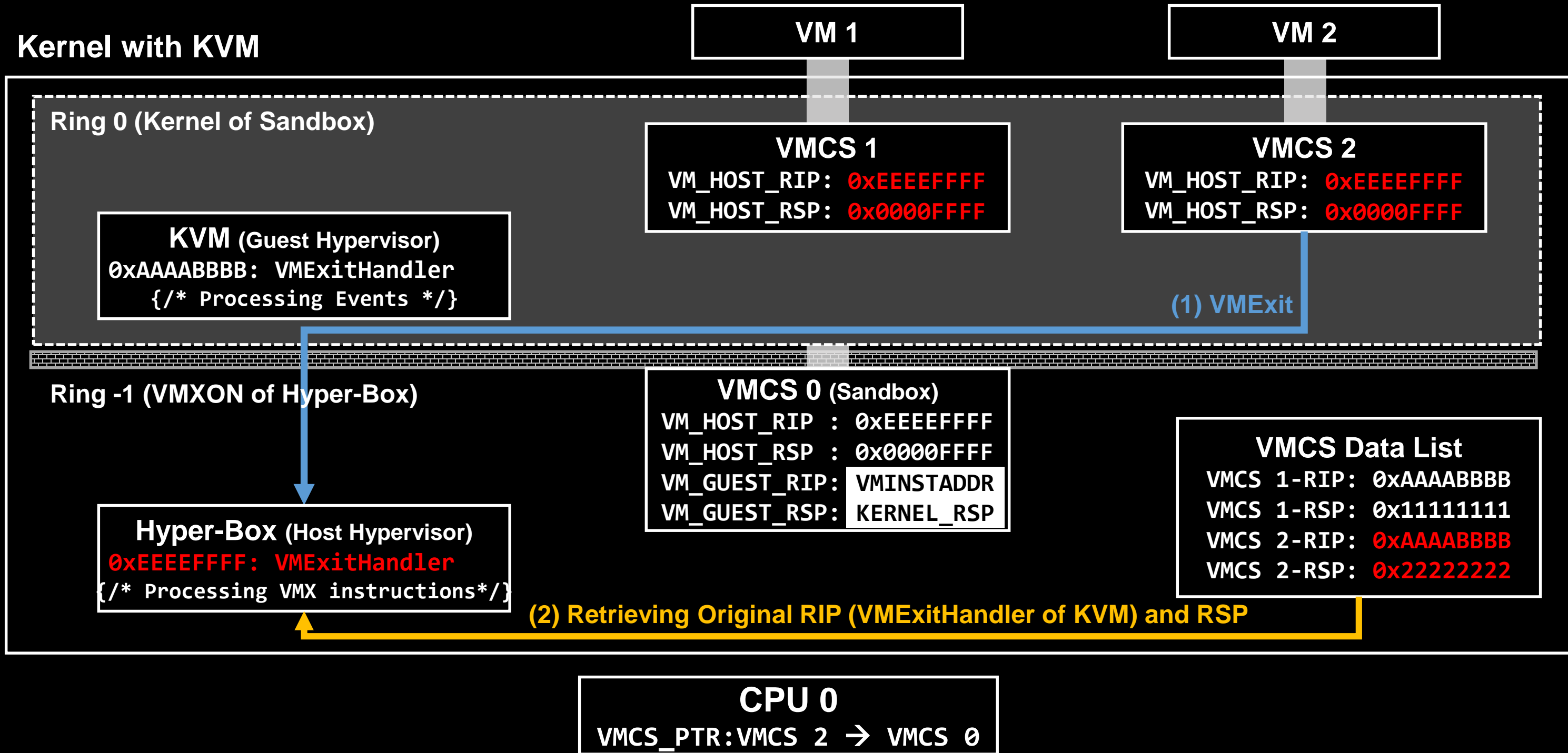
## Emulating VMX instructions – Executing Instructions (2)





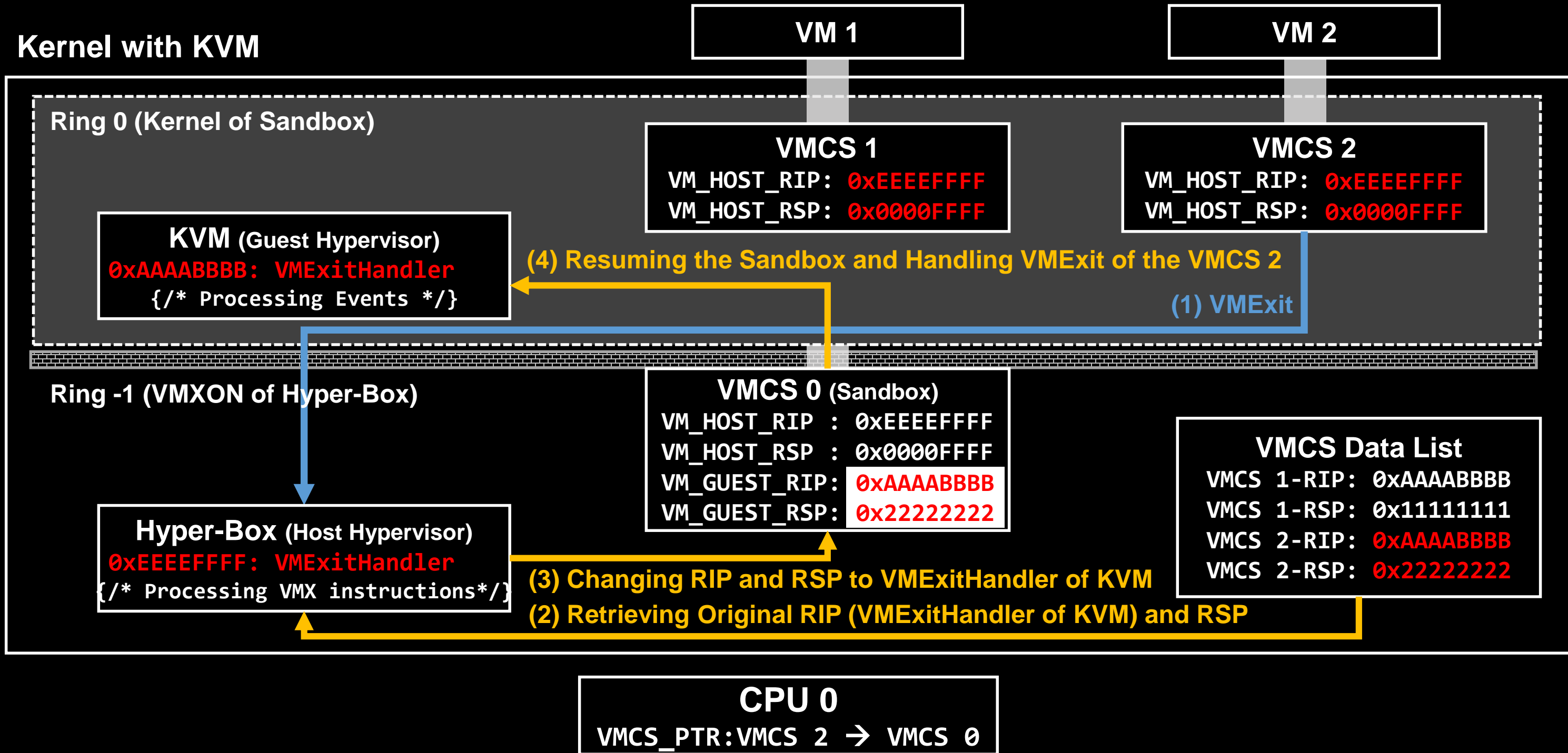
# Hyper-Box Design

## Emulating VMX instructions – Handling VMExits (1)



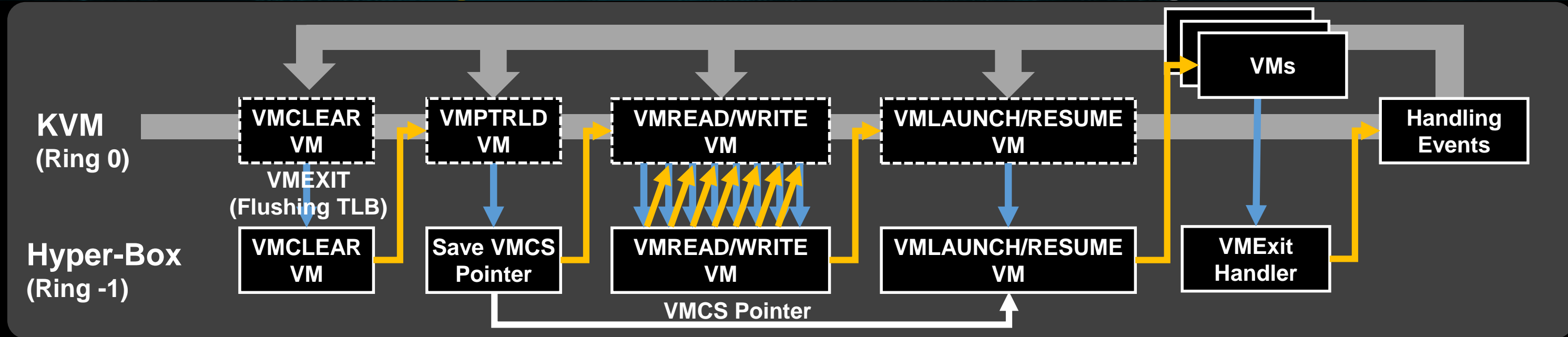
# Hyper-Box Design

## Emulating VMX instructions – Handling VMExits (2)



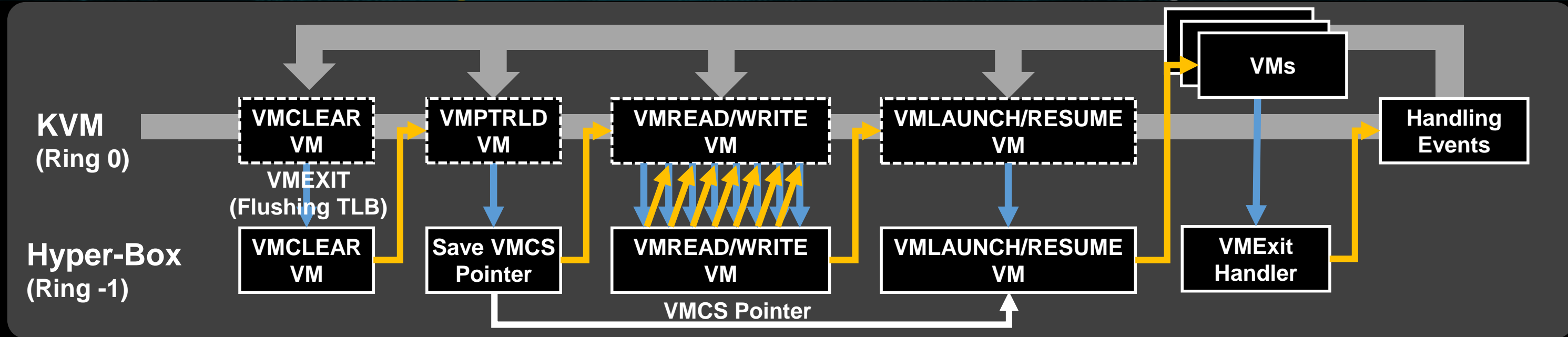
# Hyper-Box Design

## Emulating VMX instructions – Reducing Overhead (1)

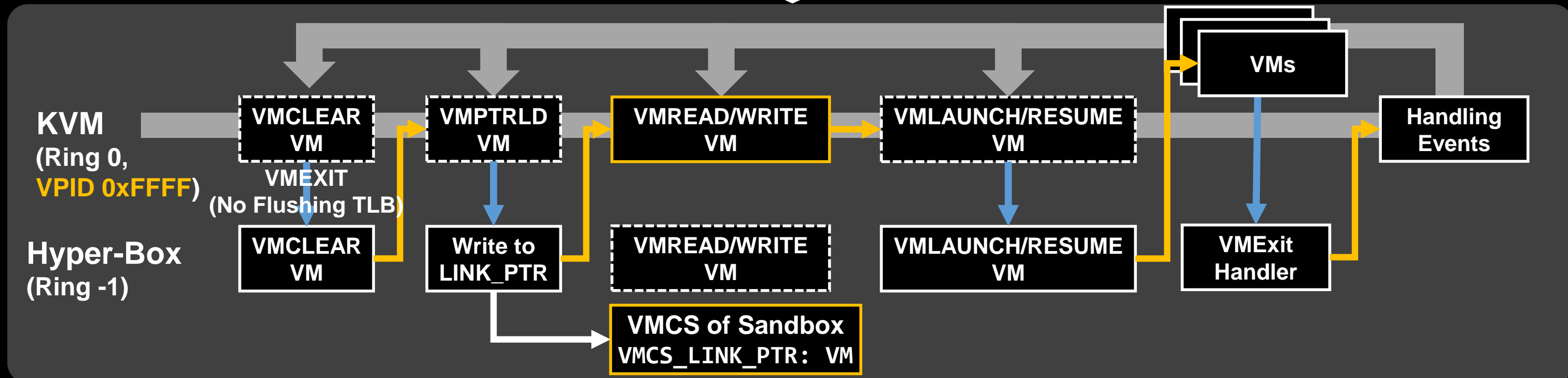


# Hyper-Box Design

## Emulating VMX instructions – Reducing Overhead (2)

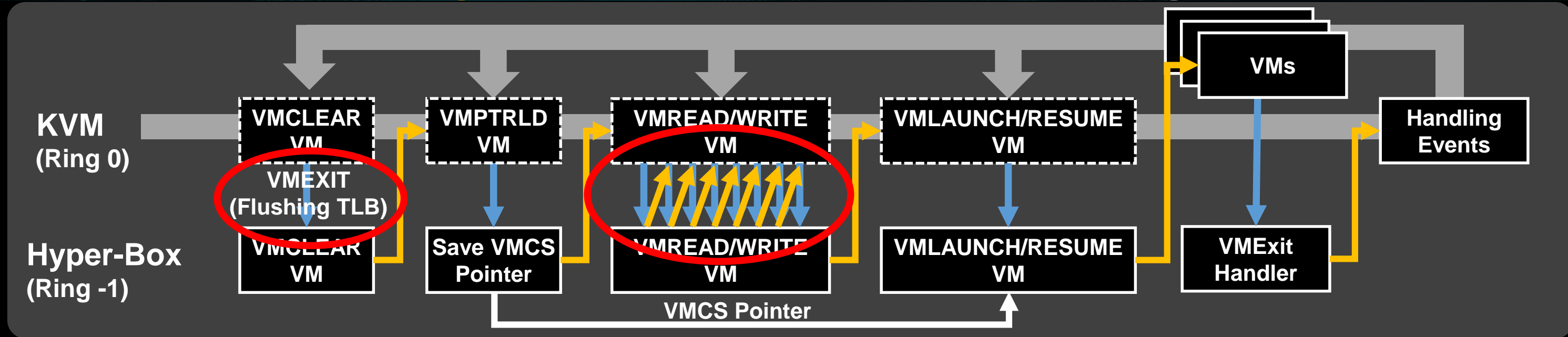


Enabling VMCS Shadowing and VPID features

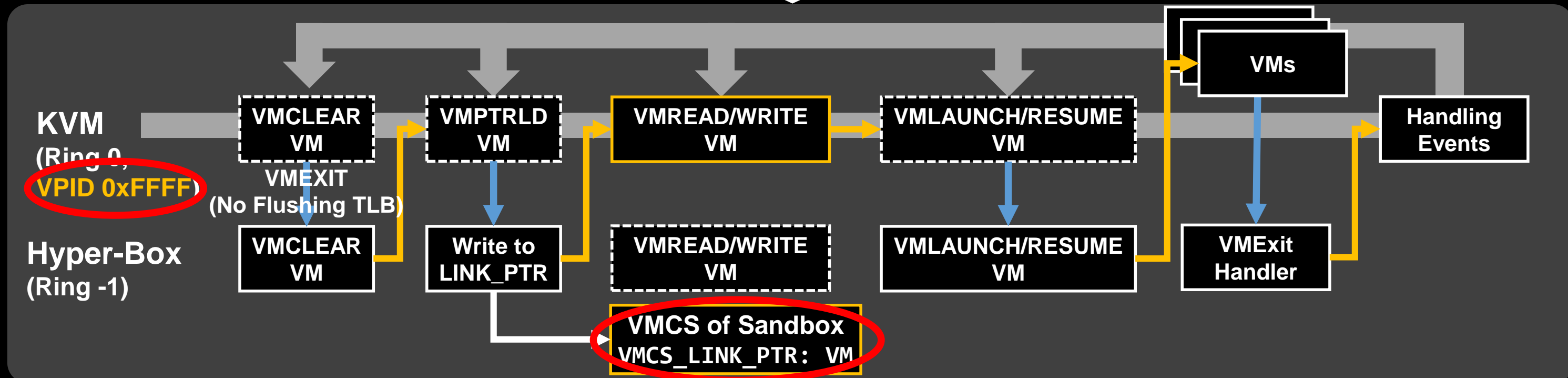


# Hyper-Box Design

## Emulating VMX instructions – Reducing Overhead (3)




Enabling VMCS Shadowing and VPID features



# Implementation of Alcatraz

- **Hyper-box is a loadable kernel module**
  - So, It can be loaded any time **BEFORE** you start VMs
    - If not, it shows some errors related to the VMXON fail
  - It was made with C and assembly code
- **Tailored Linux kernel is just recompiled version**
  - Some options of a kernel config file were **REMOVED**
    - CONFIG\_IA32\_EMULATION, CONFIG\_COMPAT, CONFIG\_COMPAT\_32, CONFIG\_X86\_X32 → System call interfaces
    - CONFIG\_JUMP\_LABEL → Runtime code modifications
  - No more system call interface for legacy!

- Background
- Analysis of Critical Paths of Escapes
- Design and Implementation of Alcatraz
- **Evaluation and Demo** 
- Conclusion and Black Hat Sound Bytes

# Benchmark Machine

## - Host Machine (Server)

- CPU: Intel Xeon Gold 6242R 2 CPU (40 cores, 80 thread)
- RAM: 512GB
- HDD: NVME 1TB
- OS: Ubuntu 20.04.01
- Kernel: 5.8.0-44-generic Linux kernel

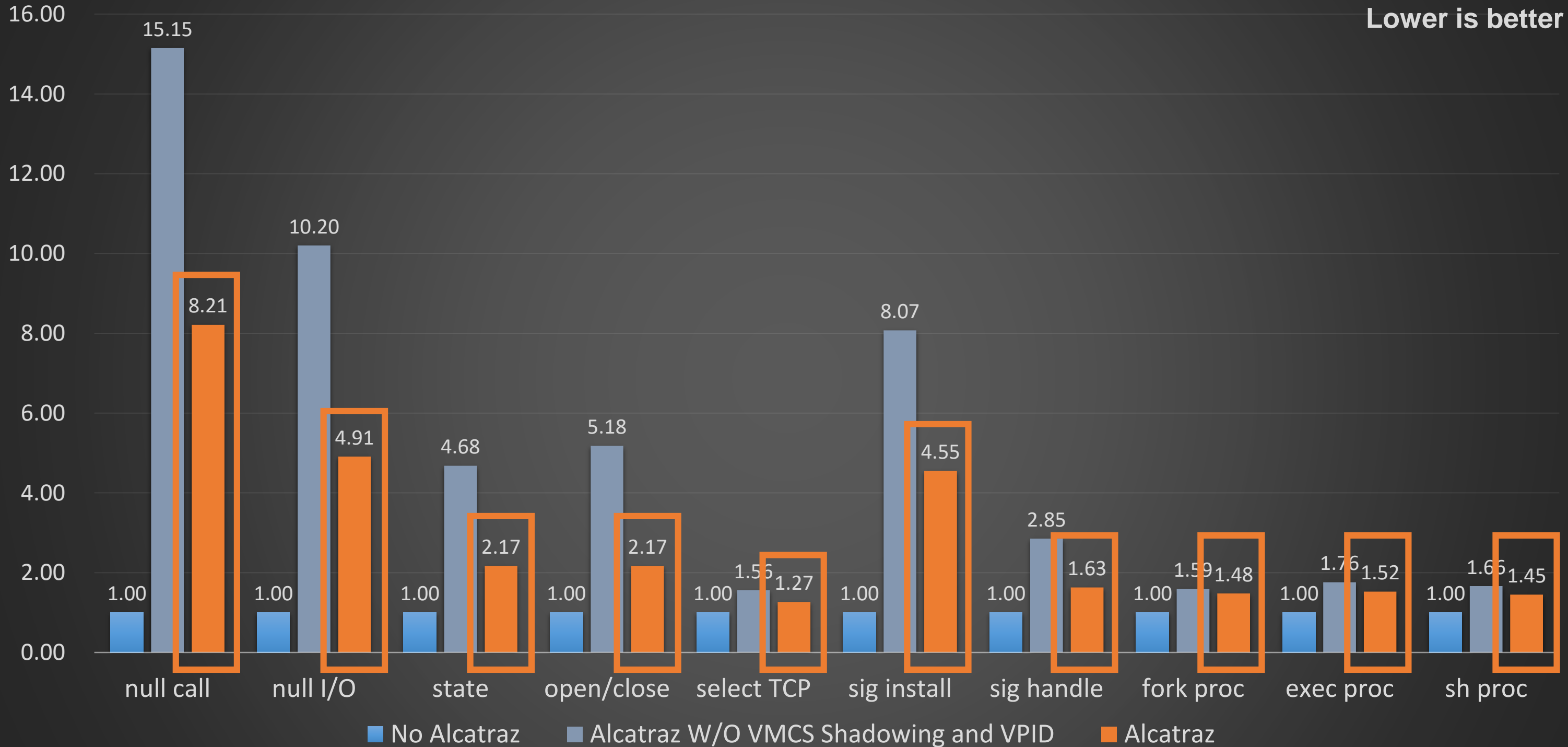
## - Guest Machine (QEMU)

- CPU: 4 cores
- RAM: 16GB
- HDD: 60GB
- OS: Ubuntu 20.04.01
- Kernel: 5.8.0-44-generic Linux kernel

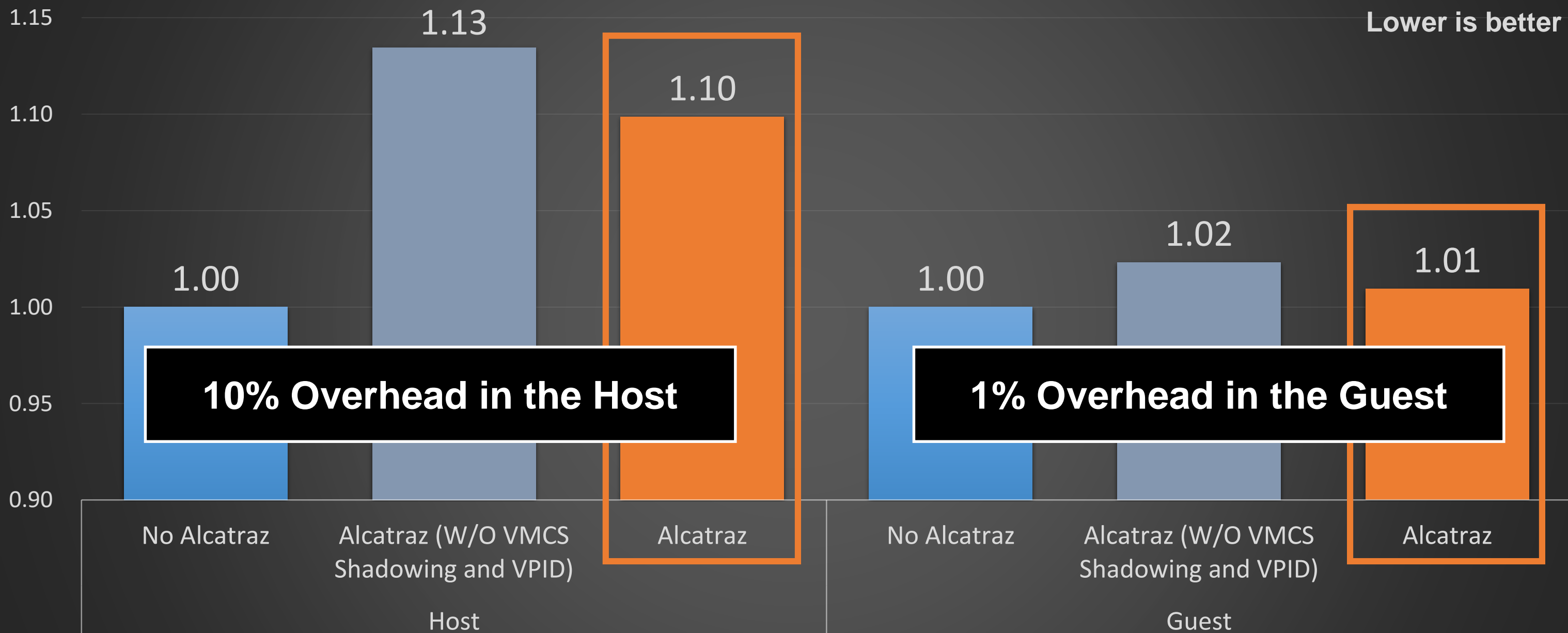


# Evaluation – Imbench

Lower is better



# Evaluation – Kernel Compilation



ALCATRAZ

# DEMO



Please check the project link: <https://github.com/kkamagui/alcatraz>

# Conclusion and Black Hat Sound Bytes

- **Containers, VMs, and microVMs suffer from escapes**
  - The hypervisor can provide strong isolation
  - But, it has excessive privilege and connects with VMs tightly
- **Taking a higher ground is not always the answer**
  - Researchers usually try to solve the problems with the higher privilege
  - The higher privilege we get, the more constraints we have!
- **Alcatraz is a practical hypervisor sandbox to prevent escapes**
  - It downgrades KVM's privilege and makes a sandbox with Intel VT
  - It has approximately 10% overhead in the host and 1% overhead in the guest (kernel compilation benchmark)

# Questions?

**SENIOR  
RESEARCHER!**



**BLACK HAT USA  
SPEAKER!**

Project : <https://github.com/kkamagui/alcatraz>  
Contact: hanseunghun@nsr.re.kr, @kkamagui1

# Reference

- Seunghun, H., Junghwan, K., Wook, S., HyungChun K., and Eungki, P. *Myth and truth about hypervisor-based kernel protector: The reason why you need shadow-box*. Black Hat Asia. 2017.
- Seunghun, H., Jun-Hyeok, P., Wook, S., Junghwan, K., and HyungChun K. *Shadow-box v2: The practical and omnipotent sandbox for ARM*. Black Hat Asia. 2018.
- Randazzo, A., and Ilenia T. *Kata containers: An emerging architecture for enabling mec services in fast and secure way*. IEEE IOTSMS. 2019.
- Agache, A., et al. *Firecracker: Lightweight virtualization for serverless applications*. NSDI. 2020.
- Young, E., et al. *The true cost of containing: A gVisor case study*. HotCloud. 2019.
- Azab, A., et al. *Hypersentry: enabling stealthy in-context measurement of hypervisor integrity*. ACM CCS. 2010.
- Rutkowska, J., and Rafał W. *Preventing and detecting Xen hypervisor subversions*. Black Hat USA. 2008
- Bulygin, Y., and David S. *Chipset based approach to detect virtualization malware*. Black Hat USA. 2008
- Li, S., John S., and Jason N. *Protecting cloud virtual machines from hypervisor and host operating system exploits*. USENIX Security. 2019.
- Zhijian S., Jian W., and Yue Z. *3d red pill: A guest-to-host escape on QEMU/KVM virtio Devices*. Black Hat Asia. 2020.
- Elhage, N. *Virtunoid: A kvm guest->host privilege escalation exploit*. Black Hat USA. 2011.
- Understanding the VENOM Vulnerability, <https://blog.trendmicro.com/understanding-the-venom-vulnerability/>