# Security Isolation for System

Zeyu Mi

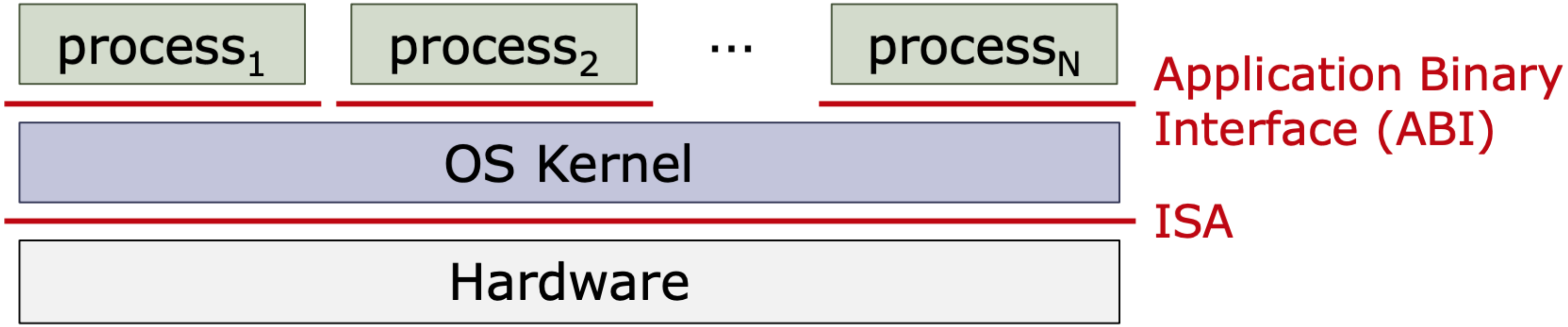
IPADS
INSTITUTE OF PARALLEL
AND DISTRIBUTED SYSTEMS

# What is system isolation?

- **An entity may encounter failures or even be malicious**
- **System isolation**
  - Contains effects of failures
  - Performance fairness
- **Unit of isolation**
  - Process
  - Virtual Machine
  - Or smaller entities

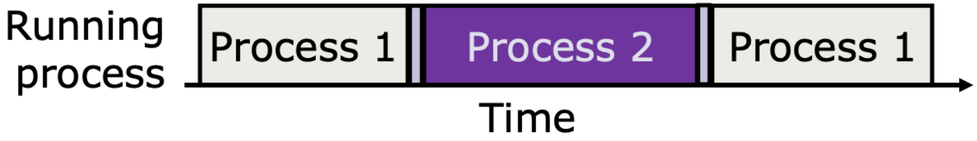
# Operating system (OS) and its Goals

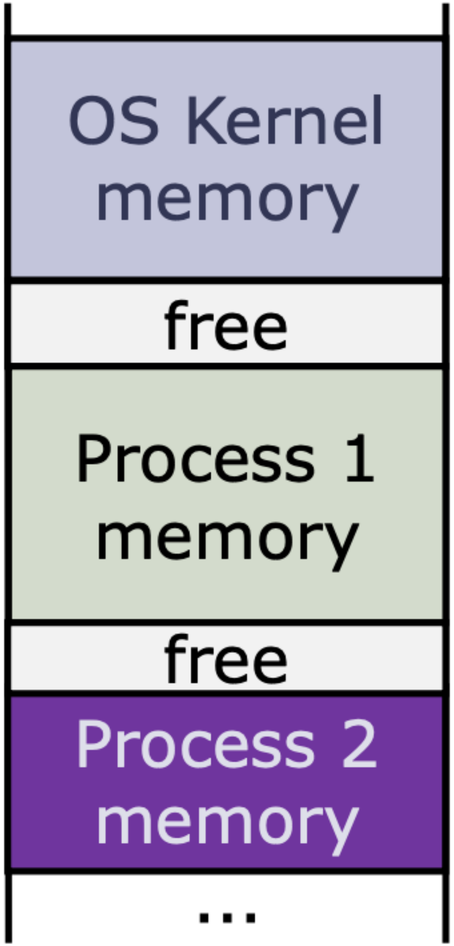
- **Resource Management**
  - OS controls how processes share hardware (CPU, memory, storage, network, etc.)
- **Abstraction**
  - Hide underline details
  - Provide usable interfaces
- **Protection and Privacy**
  - Process cannot access other process data



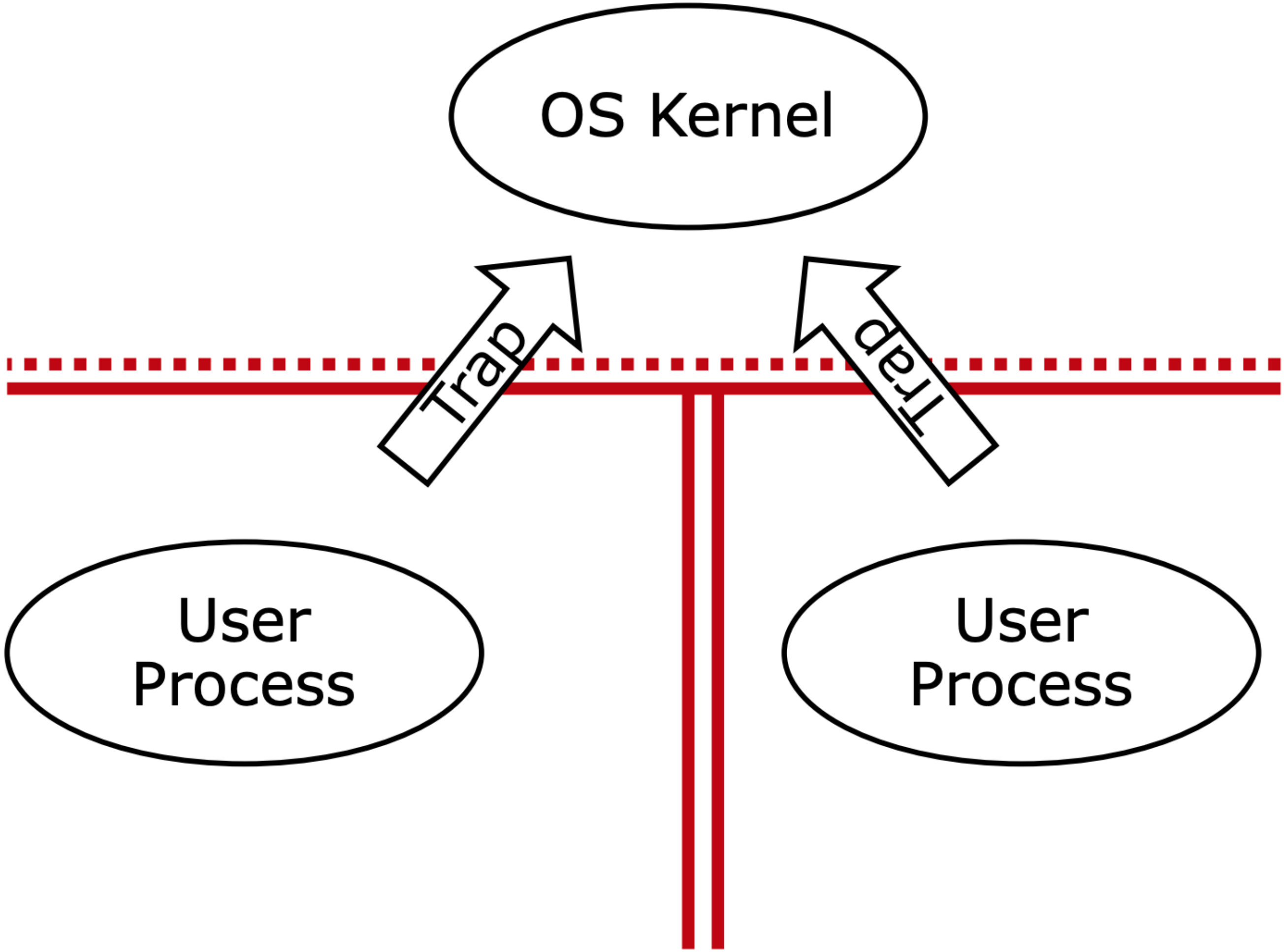
# Process Abstraction

- **Each process has a private address space**
  - Provided by OS
  - Cannot access other process spaces
- **The OS kernel schedules processes into cores**
  - Each process has a scheduling time slice
- **A process can invoke OS services via system calls**



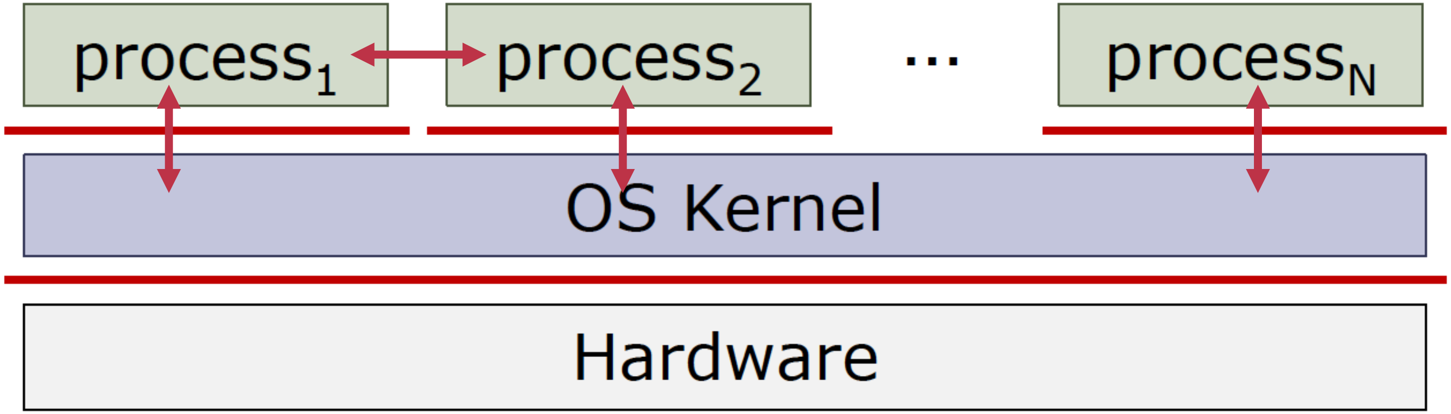


# Protection for a Single OS



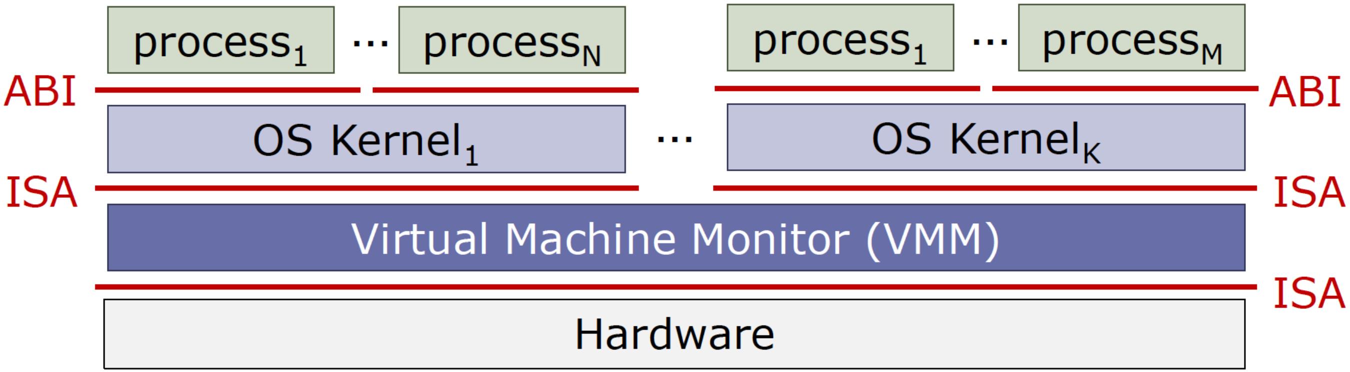
# System Isolation in OS

- **Isolation between processes (Horizontal Isolation)**
  - Enforced by OS kernel
  - Useful techniques: page tables, context switch, file abstraction…
- **Isolation between processes and the kernel (Vertical Isolation)**
  - CPU privilege and carefully designed interface



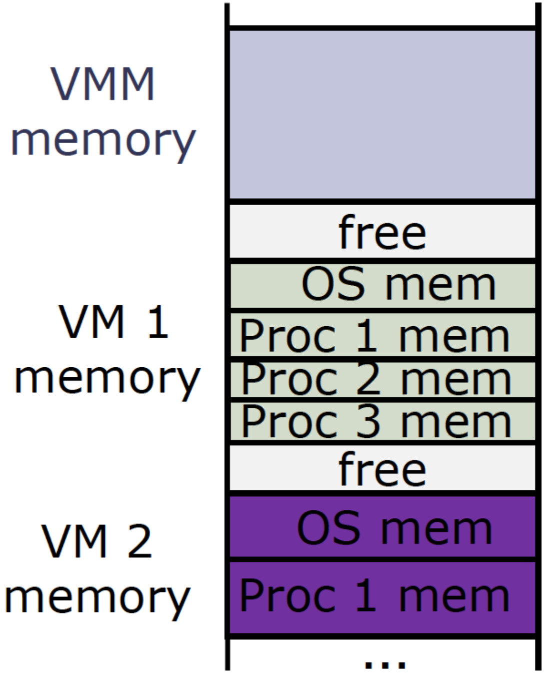
# Virtual Machine Monitor

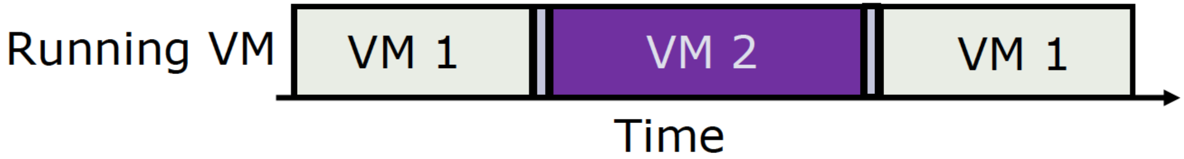
- **A VMM (aka Hypervisor) provides a system virtual machine to each OS**
- **VMM can run directly on hardware (type-1) or on another OS (type 2)**
- **Hardware virtualization**
  - VT-x: root and non-root mode
  - EPT
  - IOMMU and SR-IOV



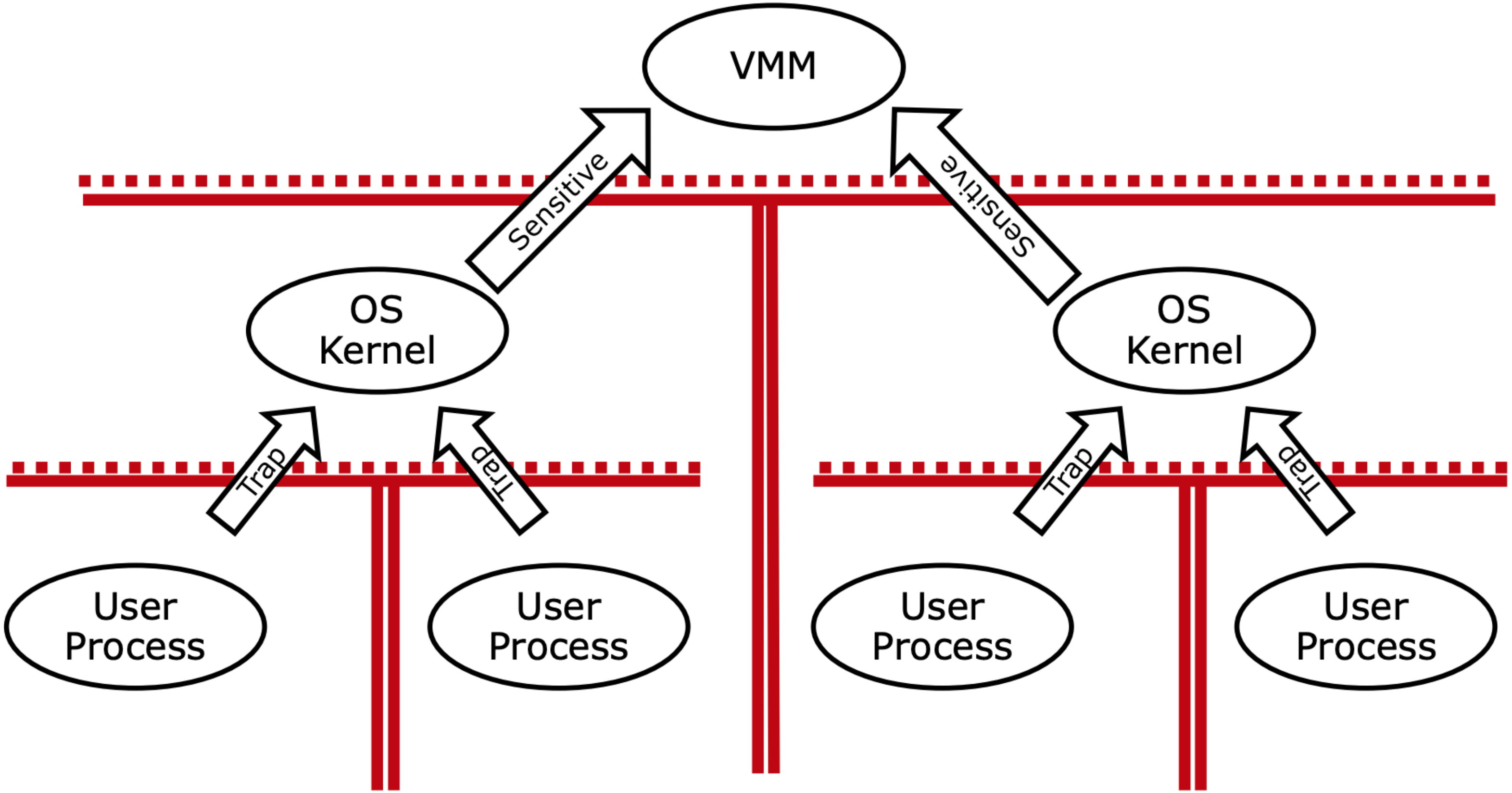
# Virtual Machine (VM) Abstraction

- **Each VM has a private address space**
  - Provided by the hypervisor
  - Cannot access other VM spaces
- **The hypervisor schedules processes into cores**
  - Each process has a scheduling time slice
- **A VM can invoke hypervisor services via hypercalls**



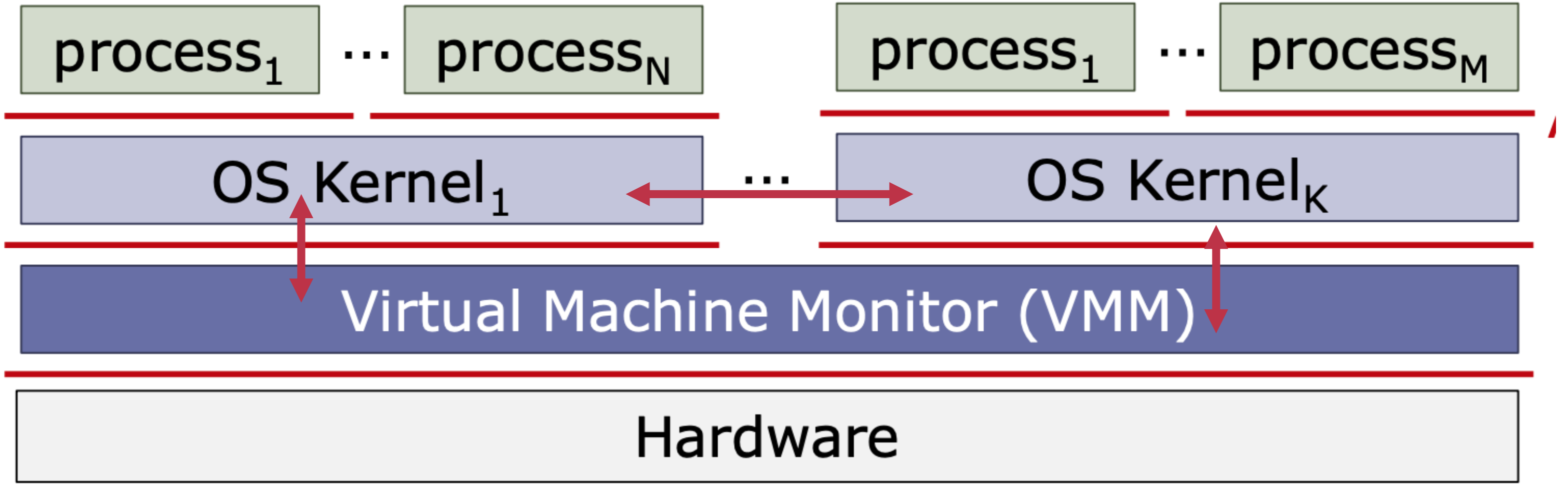


# Protection for Multiple OSes



# System Isolation in VMM

- **Isolation between OS kernels (Horizontal Isolation)**
  - Enforced by the VMM
  - Useful techniques: EPTs
- **Isolation between OS kernels and VMM (Vertical Isolation)**
  - Hardware virtualization and carefully designed interface



# Techniques for System Isolation

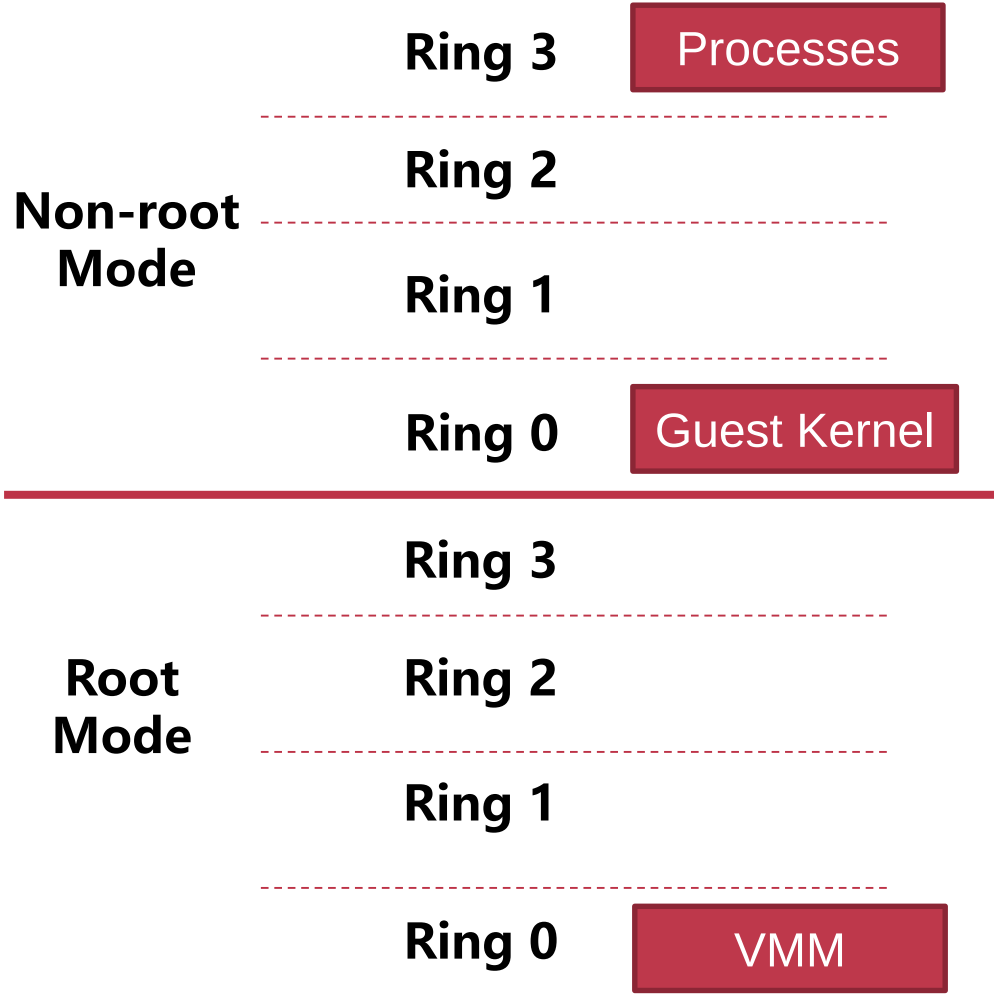
- CPU privileges
- Segmentation
- Page table
- Extended page table
- Memory domain
- Secure hardware modules

# Techniques for System Isolation

- CPU privileges
- Segmentation
- Page table
- Extended page table
- Memory domain
- Secure hardware modules

# CPU privileges

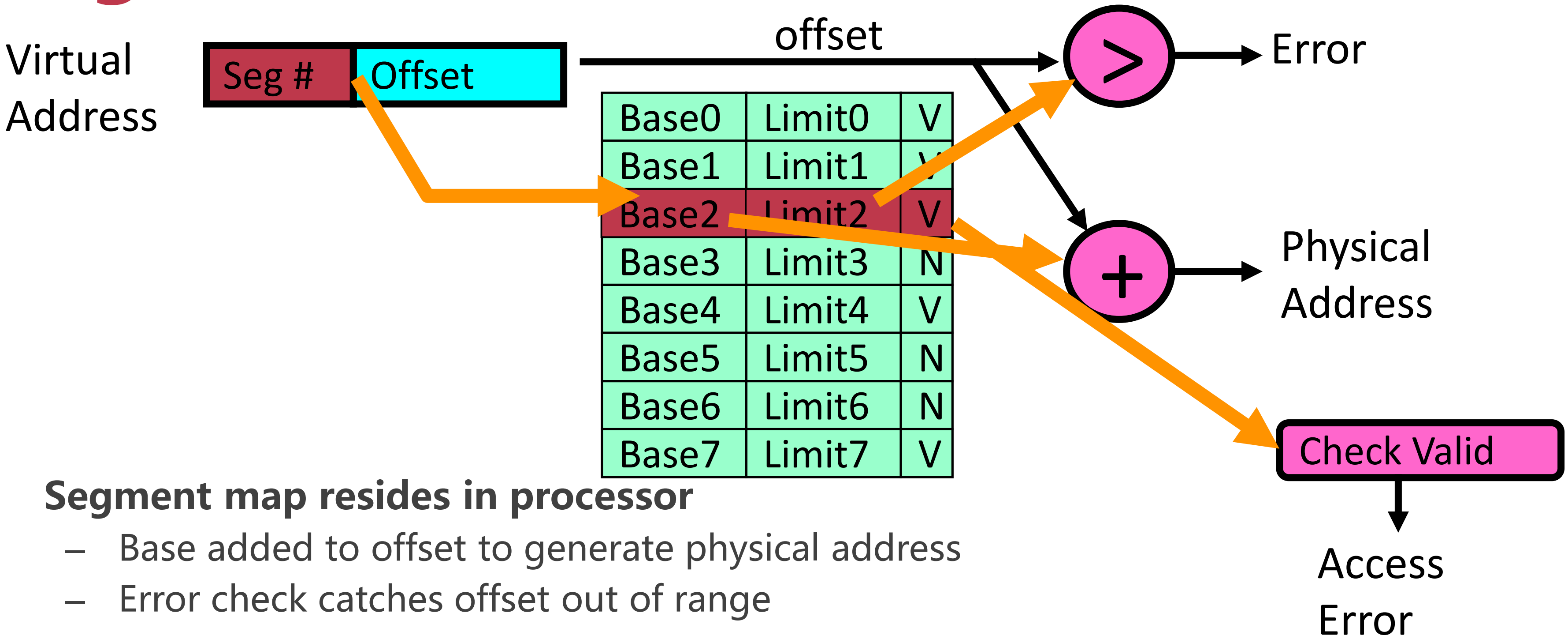
- **Privileged Mode**
  - Non-root Ring 0
  - Root Ring 0
- **Operations in privileged mode**
  - Configure address spaces
  - Access devices
  - Read/write special registers



# Techniques for System Isolation

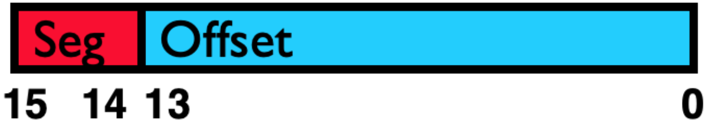
- CPU privileges
- **Segmentation**
- Page table
- Extended page table
- Memory domain
- Secure hardware modules

# Segmentation



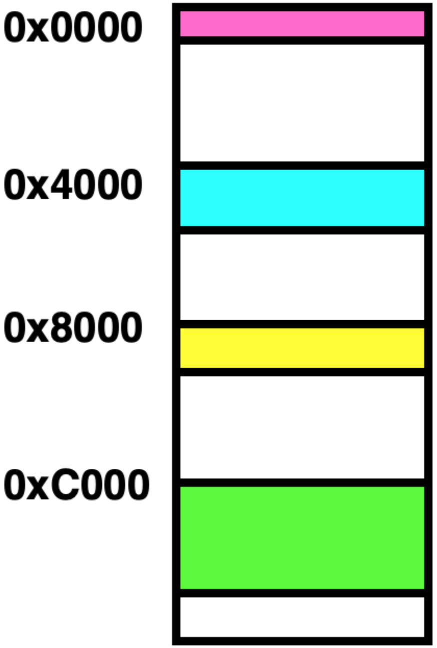
| | | |
|---|---|---|
| Base0 | Limit0 | V |
| Base1 | Limit1 | V |
| Base2 | Limit2 | V |
| Base3 | Limit3 | N |
| Base4 | Limit4 | V |
| Base5 | Limit5 | N |
| Base6 | Limit6 | N |
| Base7 | Limit7 | V |

- **Segment map resides in processor**
  - Base added to offset to generate physical address
  - Error check catches offset out of range
- **As many chunks of physical memory as entries**
  - Segment addressed by portion of virtual address
- **What is "V/N" (valid / not valid)?**
  - Can mark segments as invalid; requires check as well

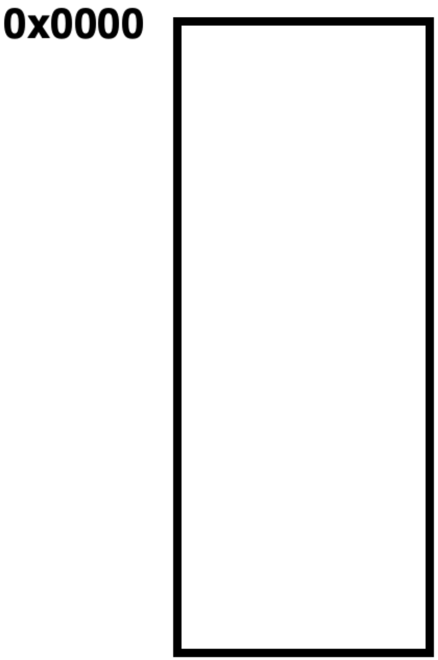


Virtual Address Format

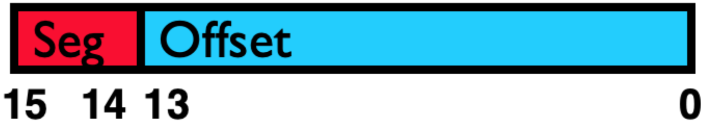
| Seg ID # | Base | Limit |
|---|---|---|
| 0 (code) | 0x4000 | 0x0800 |
| 1 (data) | 0x4800 | 0x1400 |
| 2 (shared) | 0xF000 | 0x1000 |
| 3 (stack) | 0x0000 | 0x3000 |



Virtual
Address Space

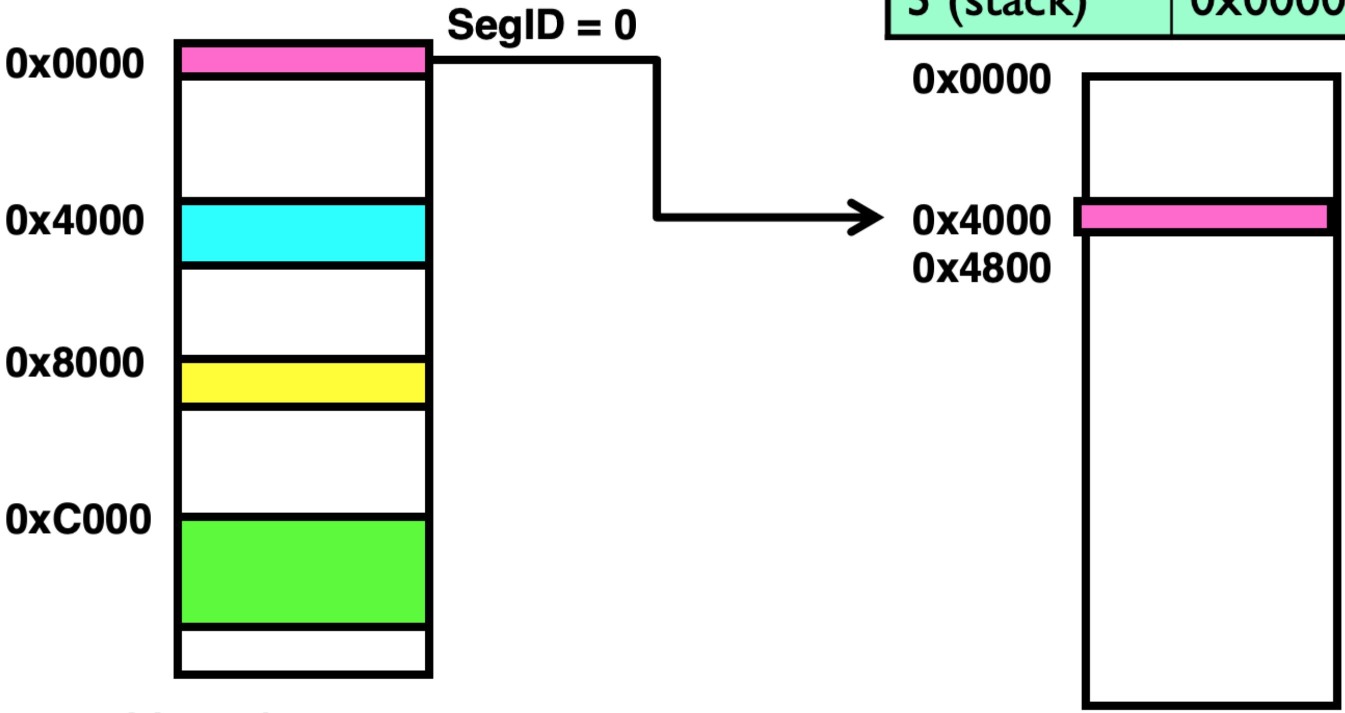


Physical
Address Space



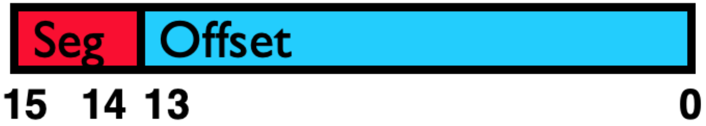
Virtual Address Format

| Seg ID # | Base | Limit |
|---|---|---|
| 0 (code) | 0x4000 | 0x0800 |
| 1 (data) | 0x4800 | 0x1400 |
| 2 (shared) | 0xF000 | 0x1000 |
| 3 (stack) | 0x0000 | 0x3000 |



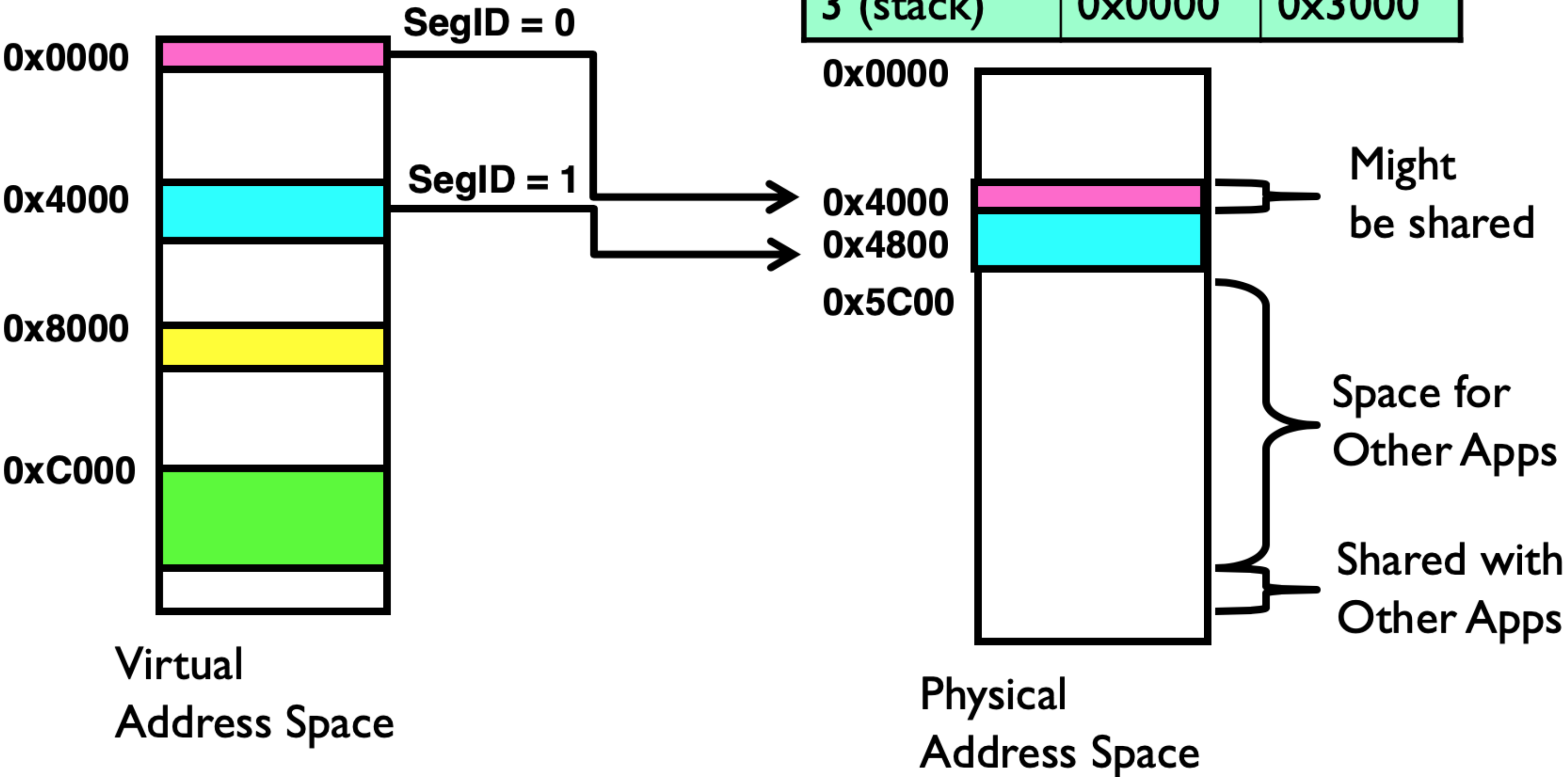
Virtual
Address Space

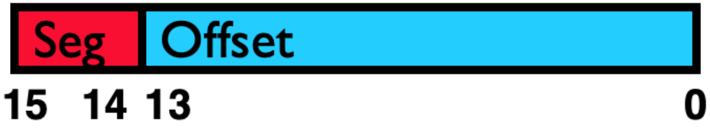
Physical
Address Space



Virtual Address Format

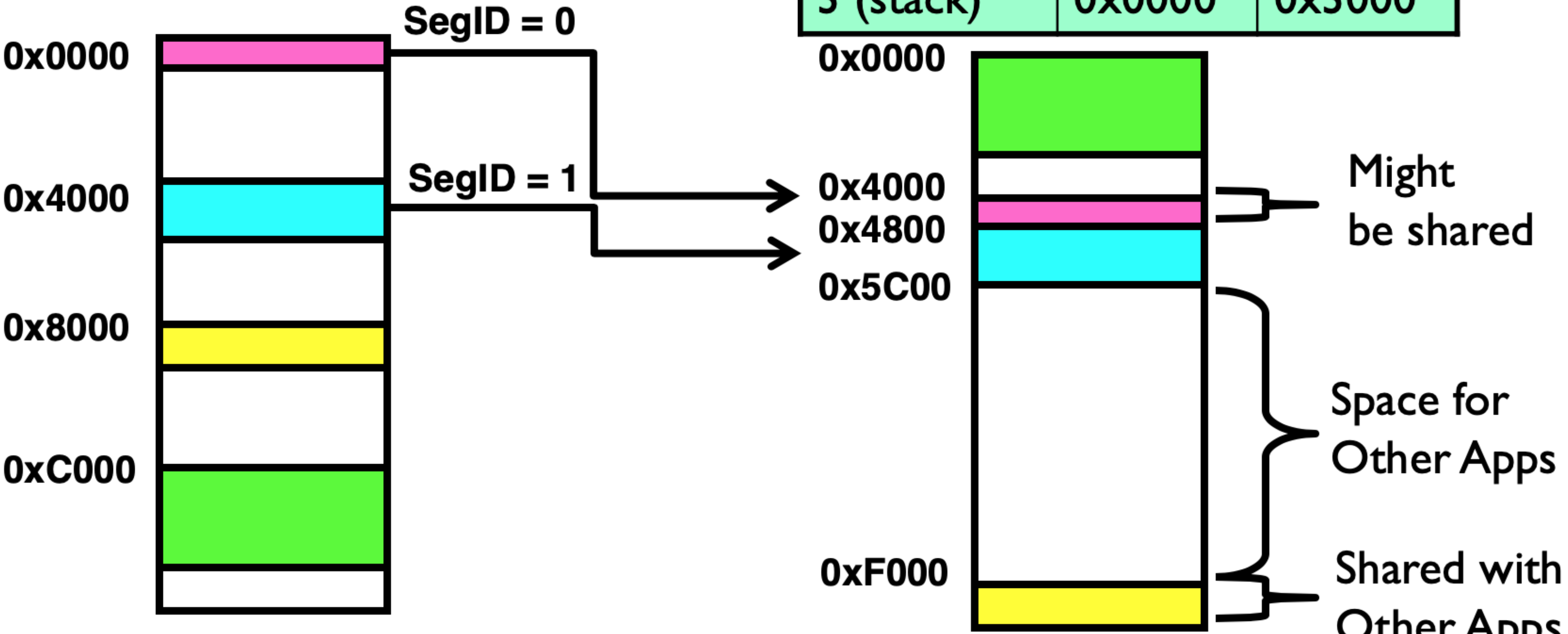
| Seg ID # | Base | Limit |
|---|---|---|
| 0 (code) | 0x4000 | 0x0800 |
| 1 (data) | 0x4800 | 0x1400 |
| 2 (shared) | 0xF000 | 0x1000 |
| 3 (stack) | 0x0000 | 0x3000 |





Virtual Address Format

| Seg ID # | Base | Limit |
|---|---|---|
| 0 (code) | 0x4000 | 0x0800 |
| 1 (data) | 0x4800 | 0x1400 |
| 2 (shared) | 0xF000 | 0x1000 |
| 3 (stack) | 0x0000 | 0x3000 |



Virtual
Address Space

Physical
Address Space

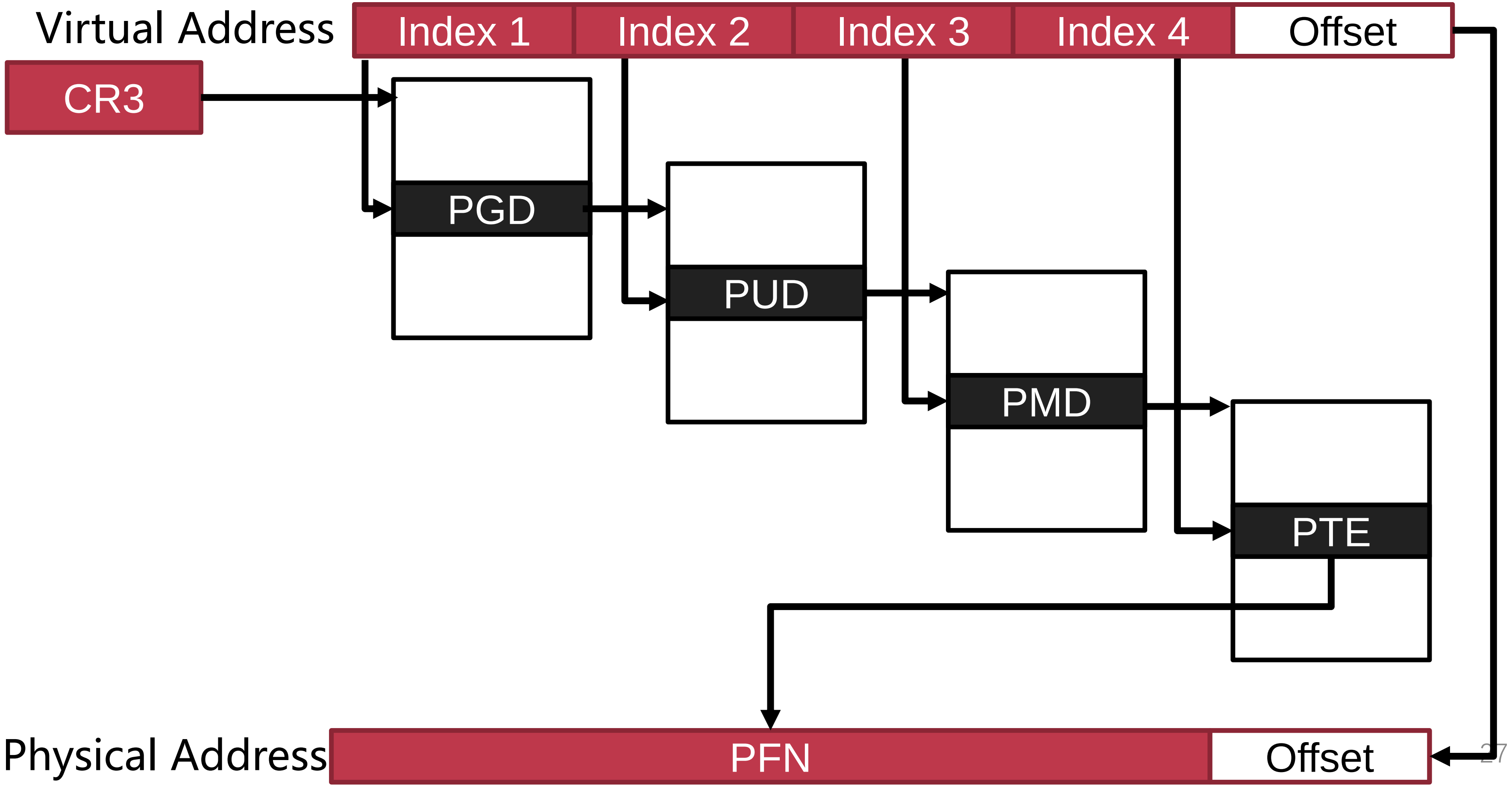
# Problems with Segmentation

- **Not supported in x86-64 mode**
- **Must fit variable-sized chunks into physical memory**
- **Fragmentation: wasted space**
  - External: free gaps between allocated chunks
  - Internal: do not need all memory within allocated chunks

# Techniques for System Isolation

- CPU privileges
- Segments
- **Page table**
- Extended page table
- Memory domain
- Secure hardware modules

# Hierarchical Page Table

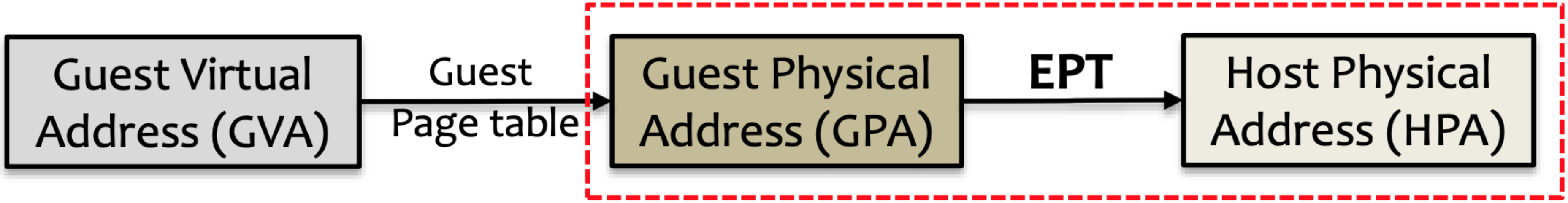


# Techniques for System Isolation

- CPU privileges
- Segments
- Page table
- **Extended page table**
- Memory domain
- Secure hardware modules

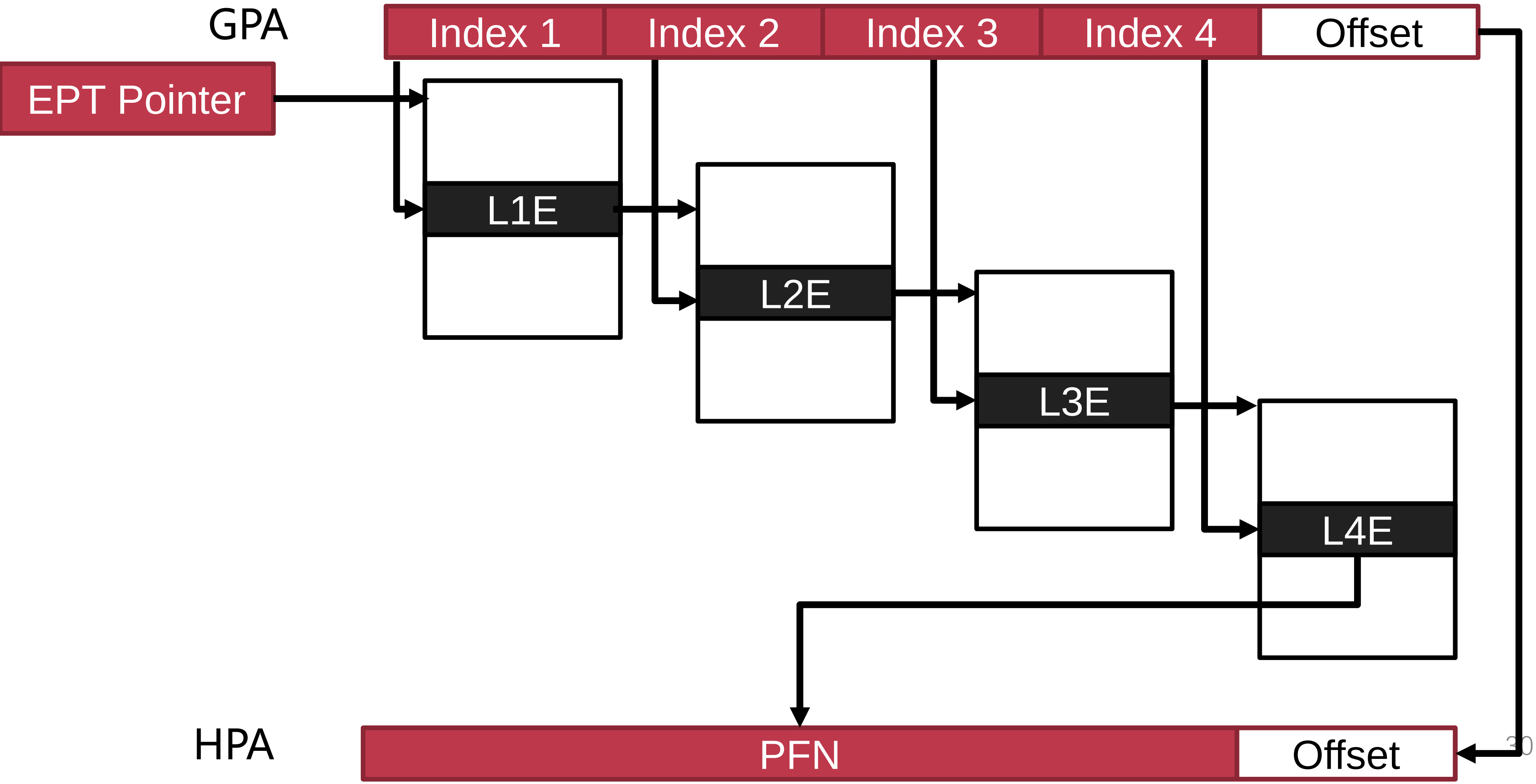
# EPT

- **Translate guest physical addr to host physical addr**
  - The two-level translation are all done by hardware

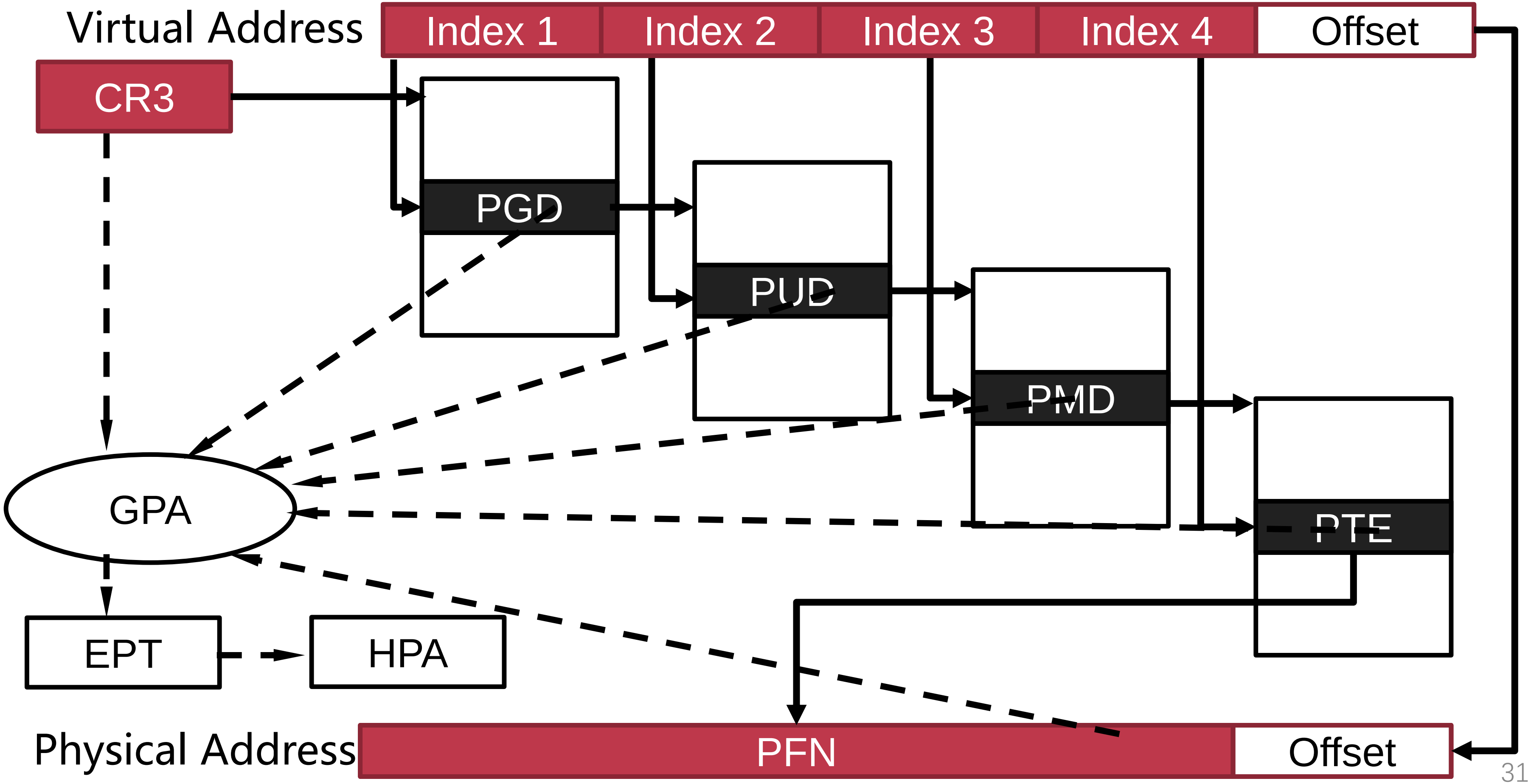

Guest Virtual Address (GVA)
Guest Page table
Guest Physical Address (GPA)
EPT
Host Physical Address (HPA)

- **EPT is manipulated and maintained by hypervisor**
  - Hypervisor controls how guest accesses physical address
  - Any EPT violation triggers a VM Exit to hypervisor

# Hierarchical Extended Page Table

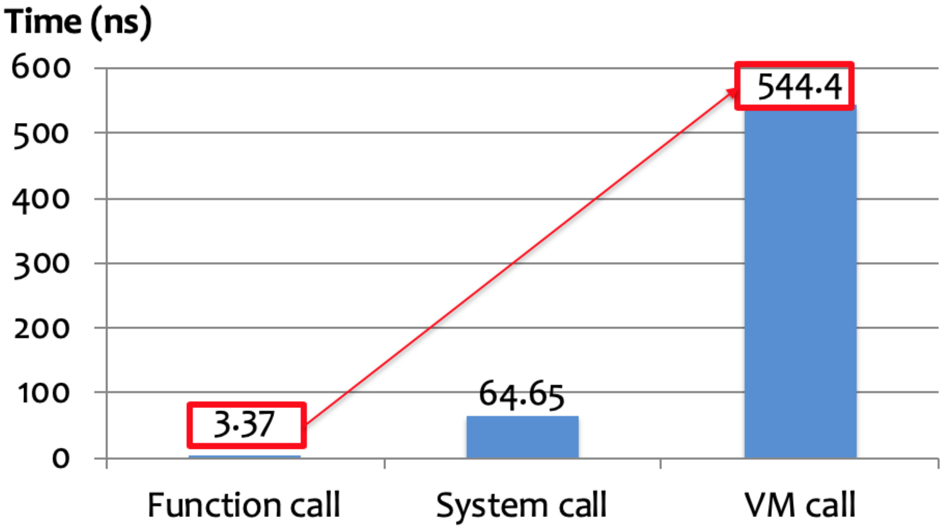


# Any GPA is translated to HPA



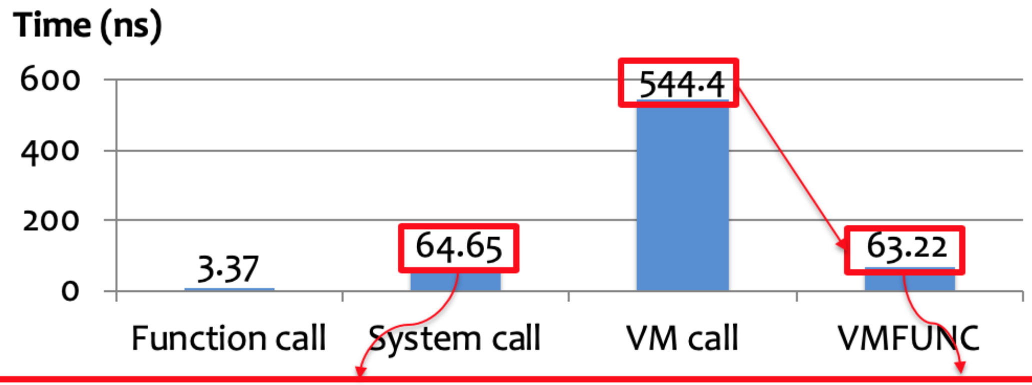
# VMFUNC

- **Context switch introduces large overhead**
  - Every EPT switch is intervened by hypervisor
  - A VM Exit takes much more time than function call



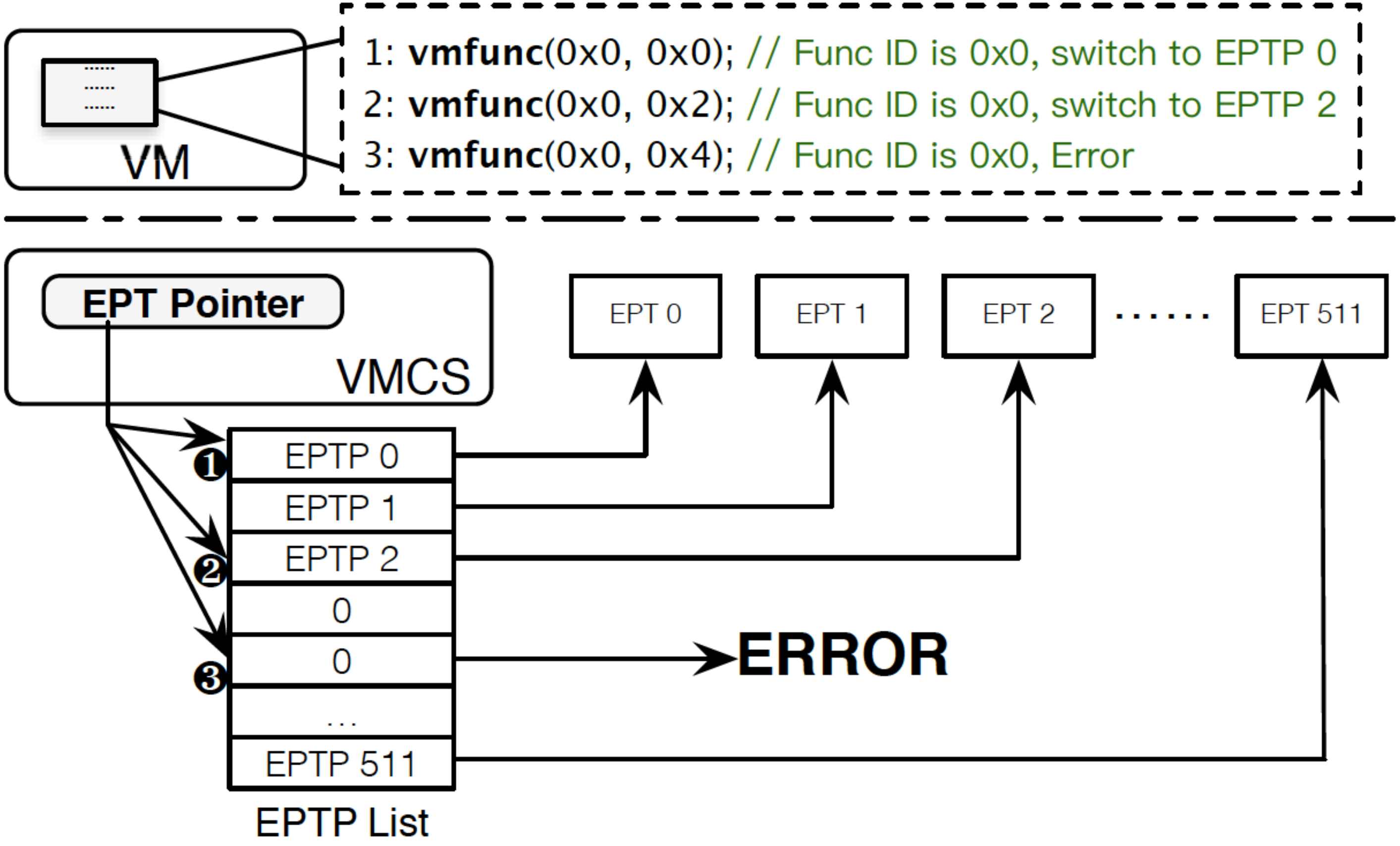
# VMFUNC

- **VM Functions: Intel virtualization extension**
  - Non-root guest VMs can directly invoke some functions without VM Exit
- **VM Function 0: EPTP Switching**
  - Software in guest VM can directly load a new EPT pointer



**VMFUNC can provide the hypervisor-level function at the cost of system calls**

# VMFUNC



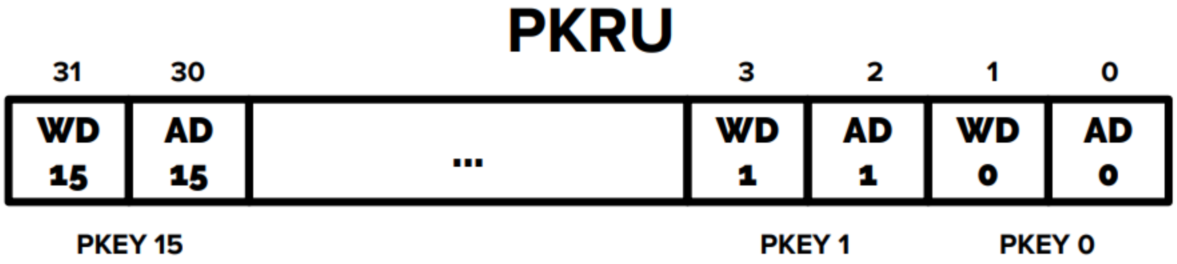
```
1: vmfunc(0x0, 0x0); // Func ID is 0x0, switch to EPTP 0
2: vmfunc(0x0, 0x2); // Func ID is 0x0, switch to EPTP 2
3: vmfunc(0x0, 0x4); // Func ID is 0x0, Error
```

# Techniques for System Isolation

- CPU privileges
- Segments
- Page table
- Extended page table
- **Memory domain**
- Secure hardware modules

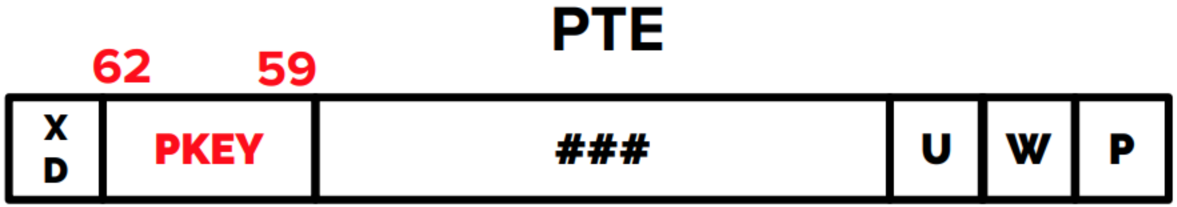
# Intel Memory Protection Keys (MPK)

- 32-bit PKRU register (Access/Write Disable)
- WRPKRU/RDPKRU



PKRU

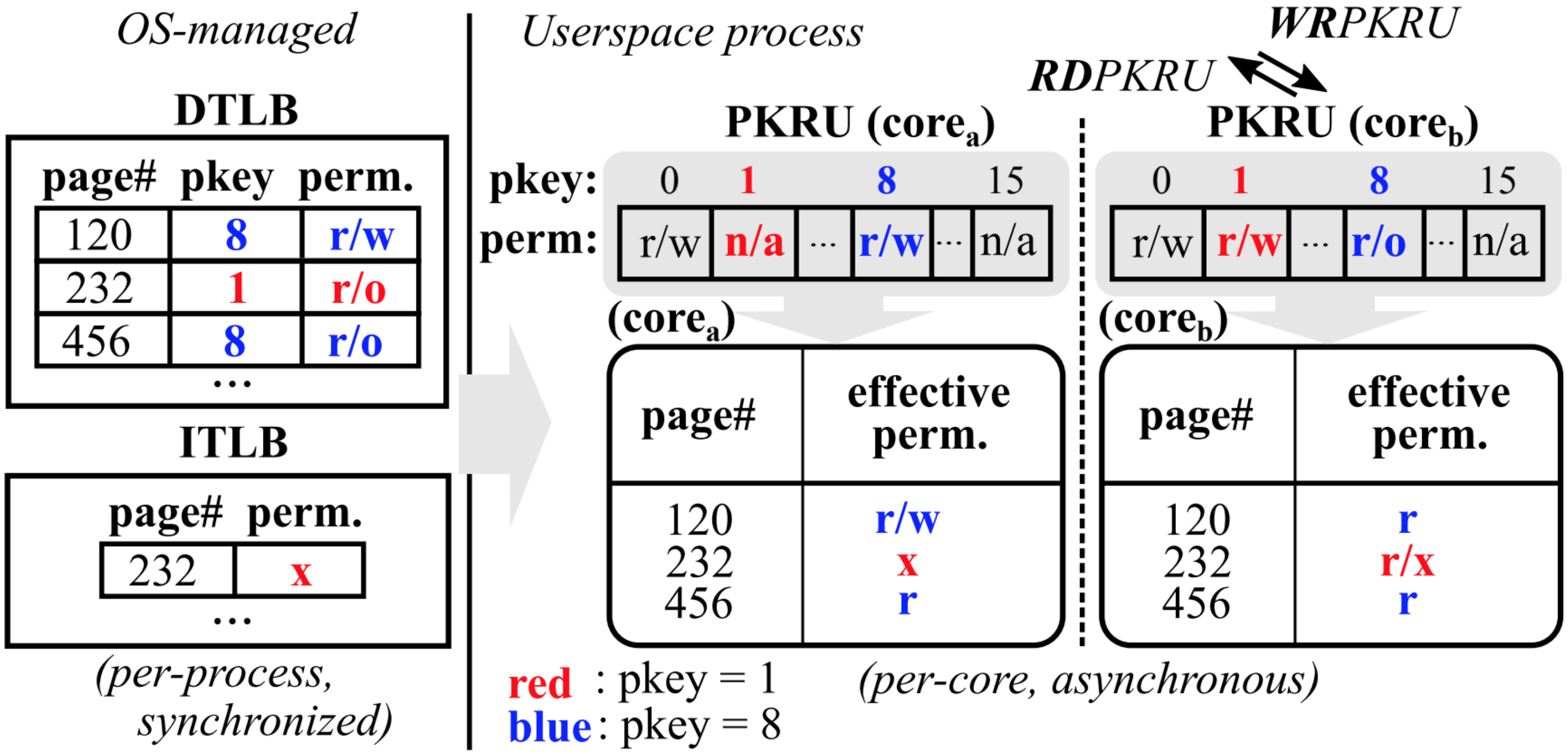
| 31 | 30 | | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|
| WD 15 | AD 15 | ... | WD 1 | AD 1 | WD 0 | AD 0 |
| PKEY 15 | | | PKEY 1 | | PKEY 0 | |



PTE

| | 62 … 59 | | | | |
|---|---|---|---|---|---|
| X D | PKEY | ### | U | W | P |

# Intel Memory Protection Keys (MPK)



*OS-managed*

**DTLB**

| page# | pkey | perm. |
|---|---|---|
| 120 | 8 | r/w |
| 232 | 1 | r/o |
| 456 | 8 | r/o |
| … | | |

**ITLB**

| page# | perm. |
|---|---|
| 232 | x |
| … | |

*(per-process, synchronized)*

*Userspace process*

***WR**PKRU*

***RD**PKRU*

**PKRU (core$_a$)**

| pkey: | 0 | 1 | | 8 | | 15 |
|---|---|---|---|---|---|---|
| perm: | r/w | n/a | ··· | r/w | ··· | n/a |

**PKRU (core$_b$)**

| pkey: | 0 | 1 | | 8 | | 15 |
|---|---|---|---|---|---|---|
| perm: | r/w | r/w | ··· | r/o | ··· | n/a |

**(core$_a$)**

| page# | effective perm. |
|---|---|
| 120<br>232<br>456 | r/w<br>x<br>r |

**(core$_b$)**

| page# | effective perm. |
|---|---|
| 120<br>232<br>456 | r<br>r/x<br>r |

**red** : pkey = 1
**blue** : pkey = 8

*(per-core, asynchronous)*

# Techniques for System Isolation

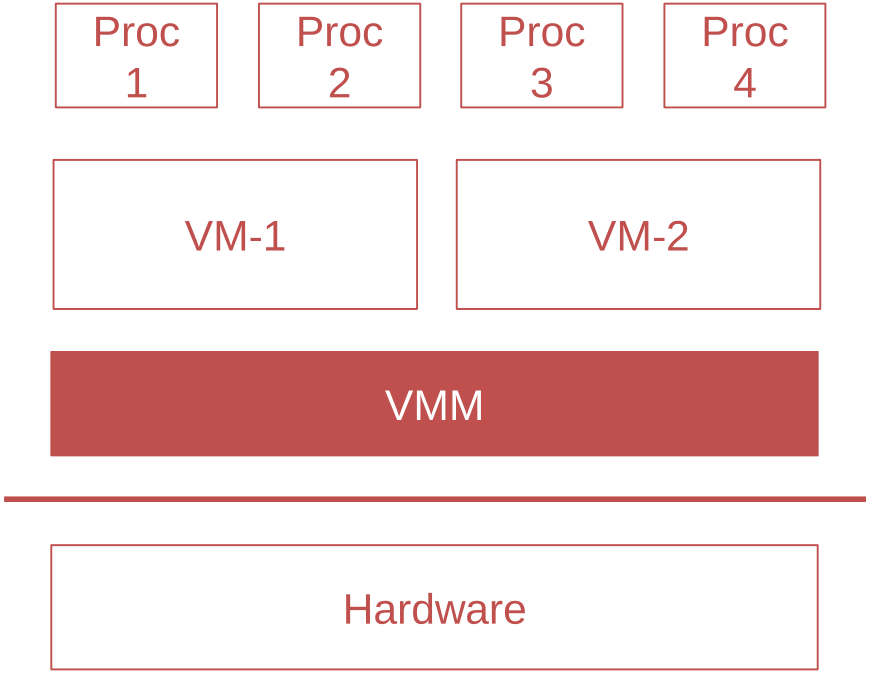
- CPU privileges
- Segments
- Page table
- Extended page table
- Memory domain
- **Secure hardware modules**

# Secure Hardware Modules

- **Many companies has released their secure hardware modules**
  - Intel SGX
  - AMD SME/SEV
- **Encrypted memory for processes or VMs**
  - Privileged software cannot access these memory
- **Will be introduced in subsequent courses**

# Principles for System Isolation

- Fine-grained isolation
- Reduced attack surface
- Small TCB and Reference monitor
- Defense in depth



# Principles for System Isolation

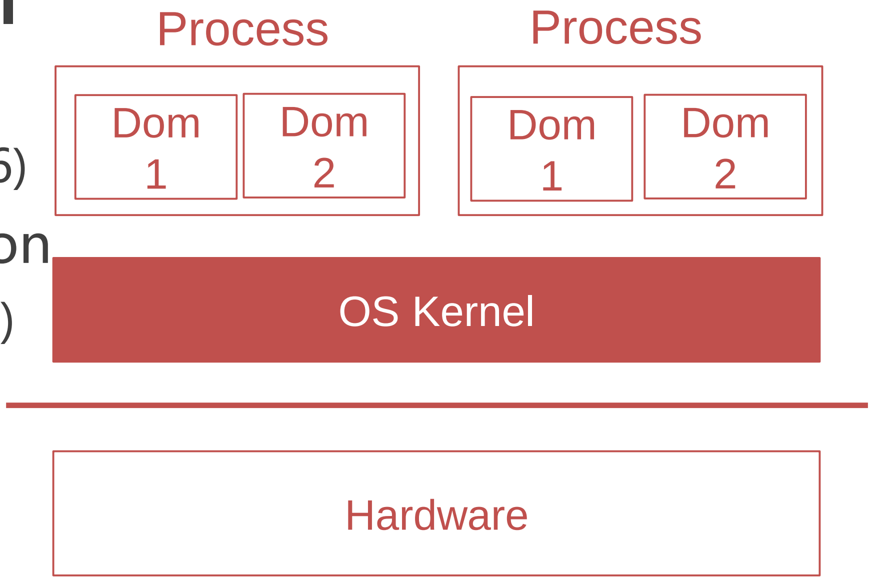
- **Fine-grained isolation**
- Reduced attack surface
- Small TCB and Reference monitor
- Defense in depth

# Fine-grained isolation

- **A process/kernel usually contains different modules**
  - Vulnerable modules and confidential ones
  - Untrusted libraries and trusted libraries
- **These modules should have different privileges to the sensitive data**
- **There is a need to provide fine-grained isolation**

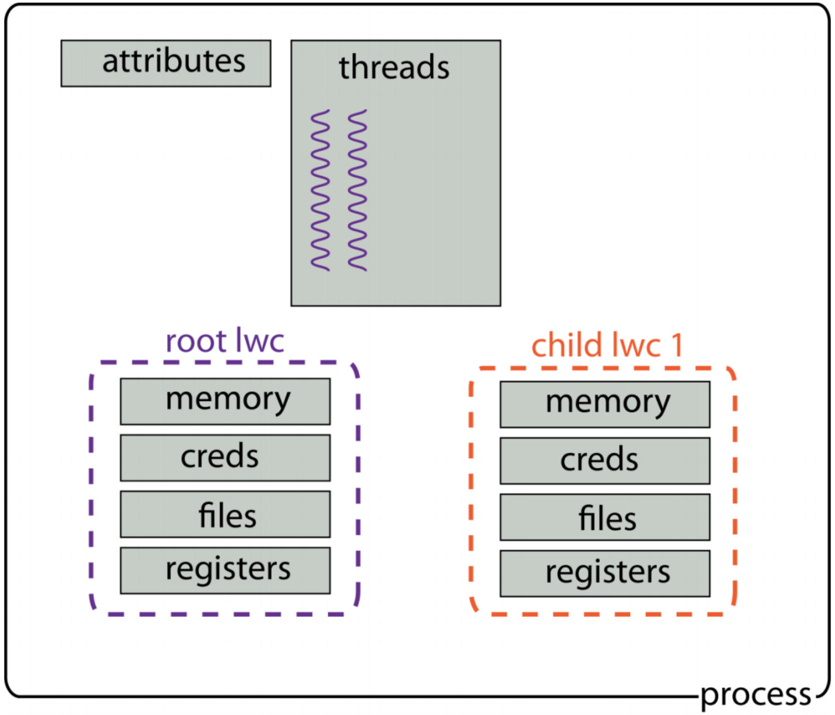
# Fine-grained isolation

- **Many techniques can be leveraged to achieve this goal**
  - Page table
    - Light-weight context (OSDI 2016)
  - CPU privilege and Segmentation
    - Lord of the x86 Rings (CCS 2018)
  - EPT and VMFUNC
    - SeCage (CCS 2015)
  - MPK
    - Hodor (ATC 2019)



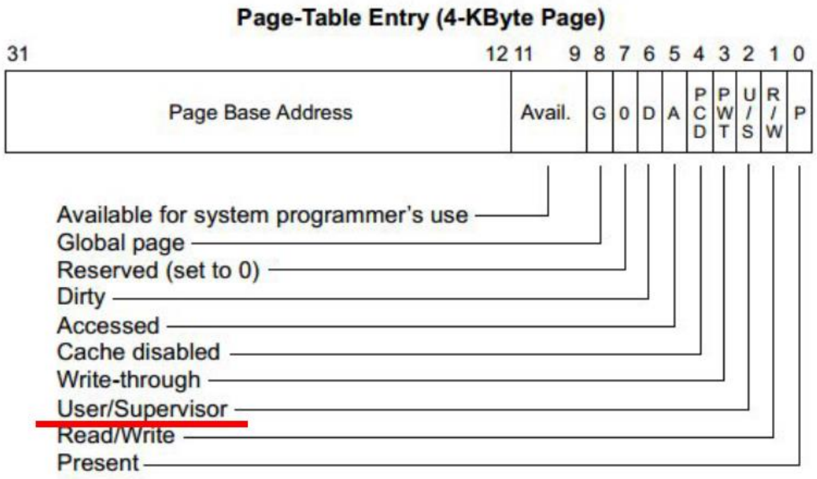
# Light-Weight Context (LWC)

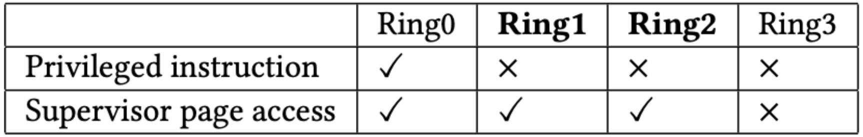
- **Multiple LWC for one process**
- **Each LWC has a private page table**
- **Switching LWC should get trapped into the kernel**
  - The kernel then changes related environment, including the page table



# Lord of the x86 Rings

- **Switching LWC is slow**
  - Cost 6050 cycles in a Skylake CPU
- **Why not using CPU rings?**
  - Ring 0-2 can access supervisor pages
  - Ring 3 cannot access these pages
  - Only Ring 0 can execute privilege instructions



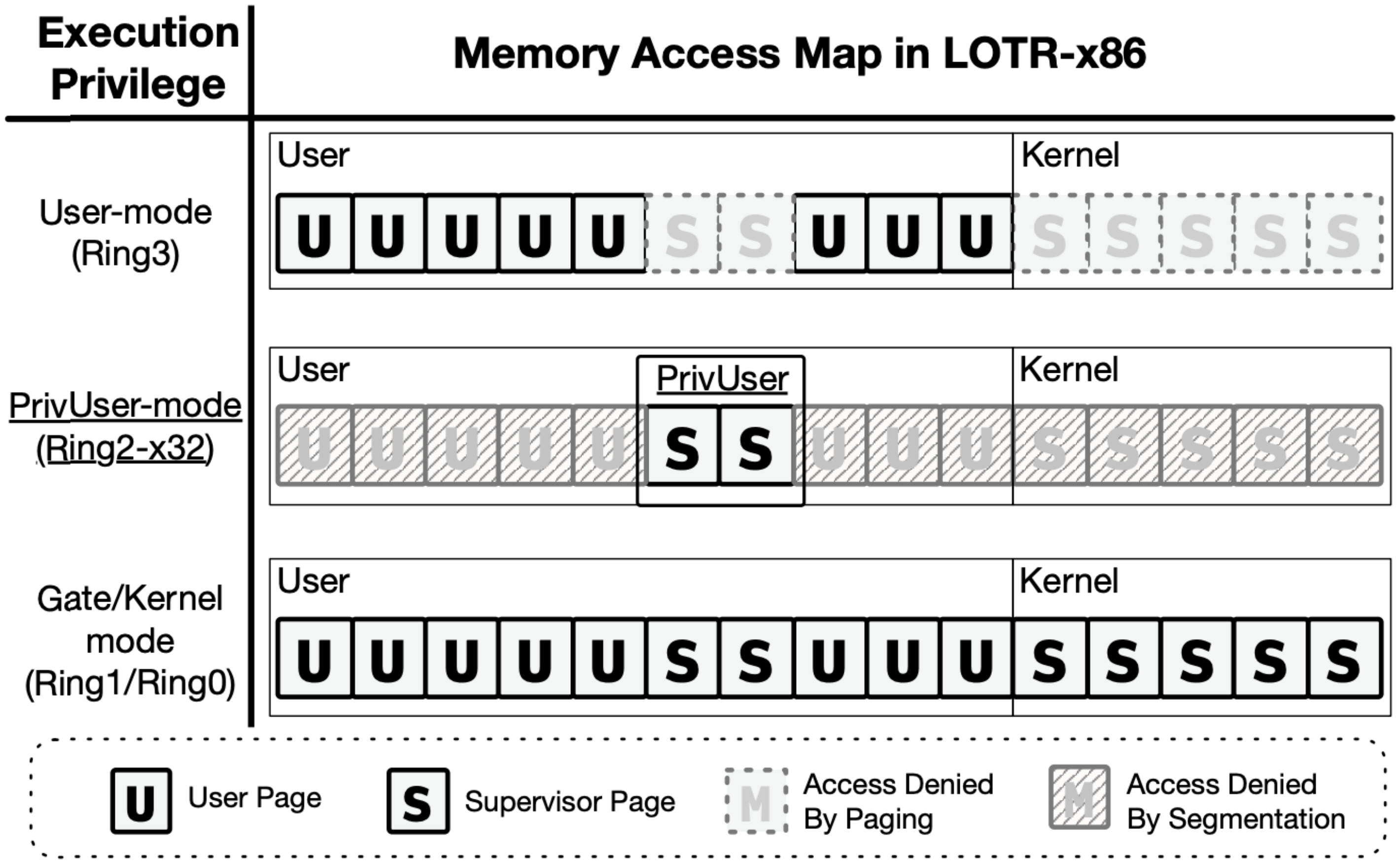


| | Ring0 | **Ring1** | **Ring2** | Ring3 |
|---|---|---|---|---|
| Privileged instruction | ✓ | × | × | × |
| Supervisor page access | ✓ | ✓ | ✓ | × |

# Lord of the x86 Rings

- **Put privileged user to ring 1 or ring 2**
- **Map privileged user data and code to supervisor mode**
- **Question**
  - What if the program in Ring 1 or Ring 2 is malicious?
  - It can access kernel data or code!

# Segmentation to the Rescue



**Execution Privilege**

**Memory Access Map in LOTR-x86**

User-mode (Ring3)

User | Kernel

U U U U U S S U U U S S S S S

PrivUser-mode (Ring2-x32)

User | PrivUser | Kernel

U U U U U S S U U U S S S S S

Gate/Kernel mode (Ring1/Ring0)

User | Kernel

U U U U U S S U U U S S S S S

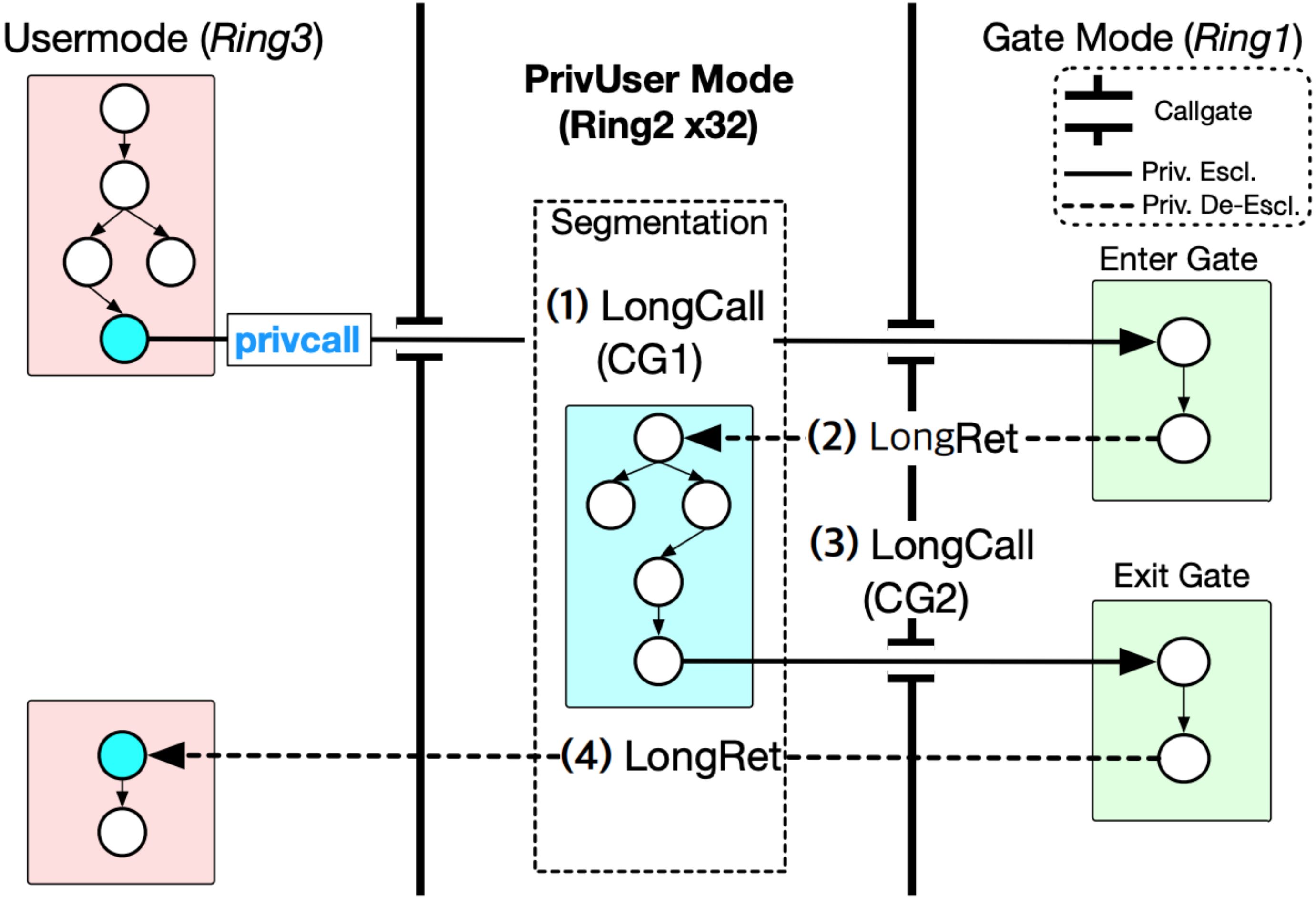
U User Page

S Supervisor Page

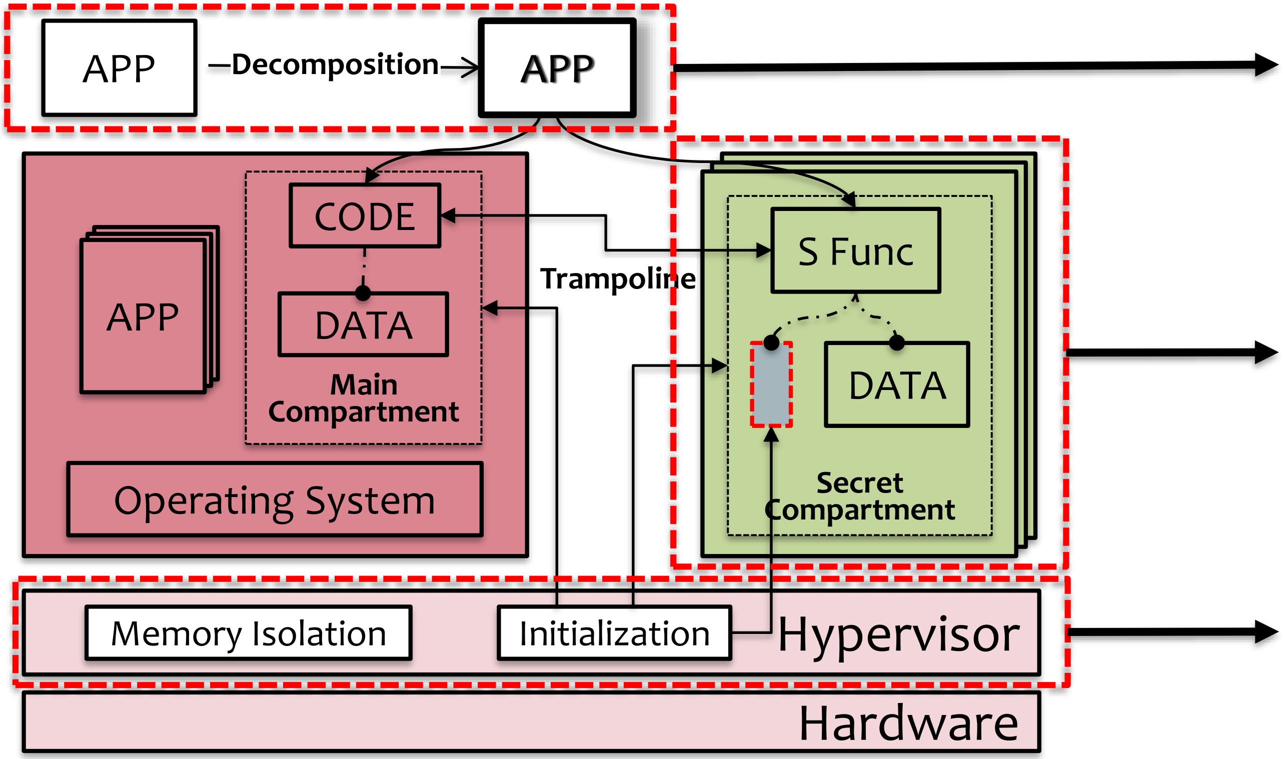
M Access Denied By Paging

M Access Denied By Segmentation

# Lord of the x86 Rings



# SeCage

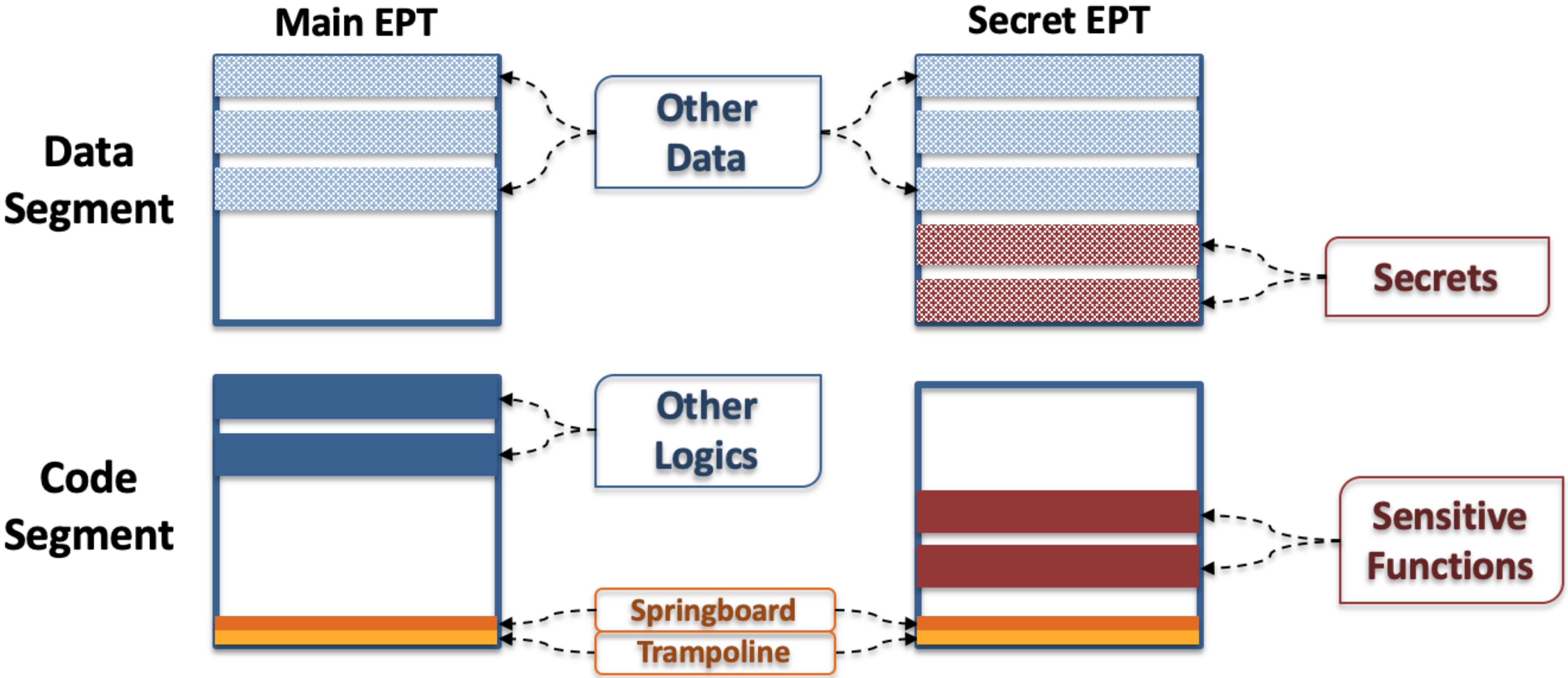


**Application decomposition**

**Life-cycle Management**

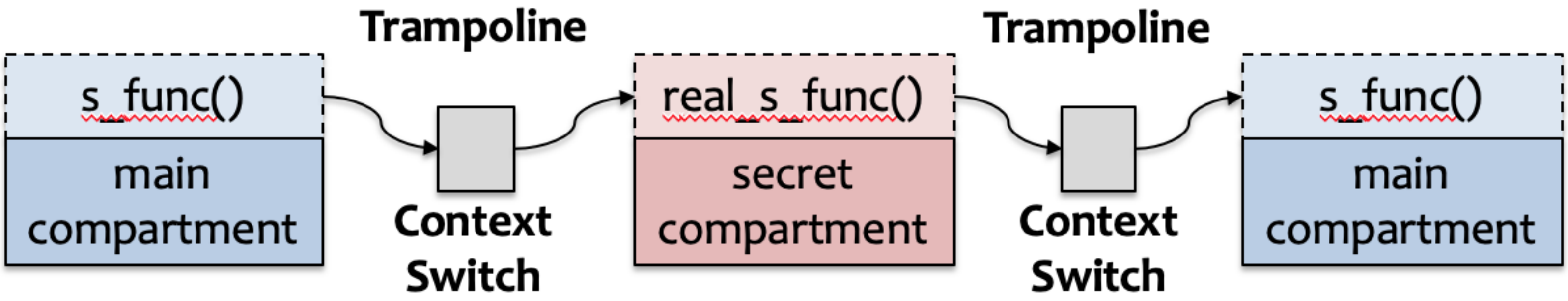
**Runtime Protection**

# SeCage: Different EPTs for two Parts

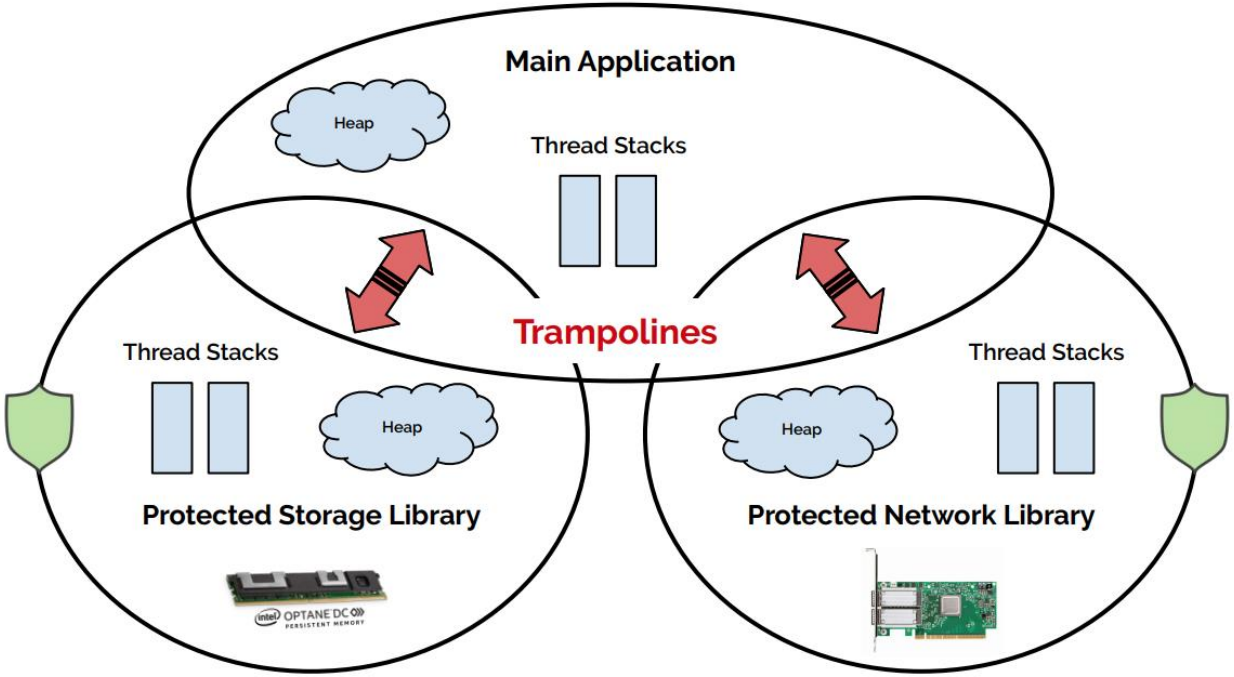


# SeCage: Using VMFUNC to Switch EPTs

- **Trampoline control flow**



# Hodor


Main Application
Heap
Thread Stacks
Trampolines
Thread Stacks
Heap
Protected Storage Library
intel OPTANE DC PERSISTENT MEMORY
Heap
Thread Stacks
Protected Network Library

# Hodor

- **Leverage MPK to provide different memory views**

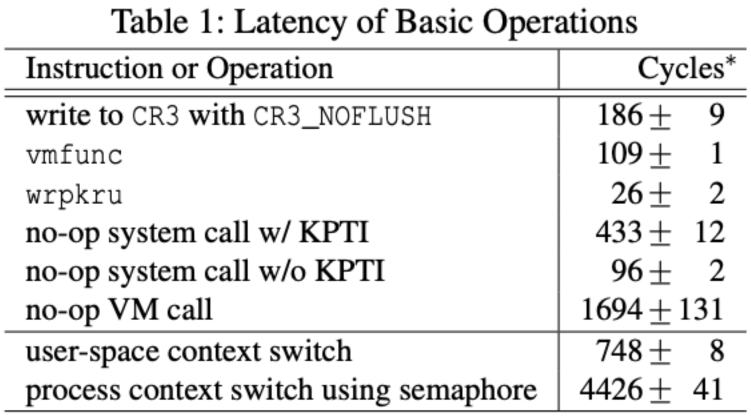


Table 1: Latency of Basic Operations

| Instruction or Operation | Cycles* |
|---|---|
| write to `CR3` with `CR3_NOFLUSH` | 186 ± 9 |
| `vmfunc` | 109 ± 1 |
| `wrpkru` | 26 ± 2 |
| no-op system call w/ KPTI | 433 ± 12 |
| no-op system call w/o KPTI | 96 ± 2 |
| no-op VM call | 1694 ± 131 |
| user-space context switch | 748 ± 8 |
| process context switch using semaphore | 4426 ± 41 |

- **Have to replace illegal wrpkru instructions**

# Principles for System Isolation

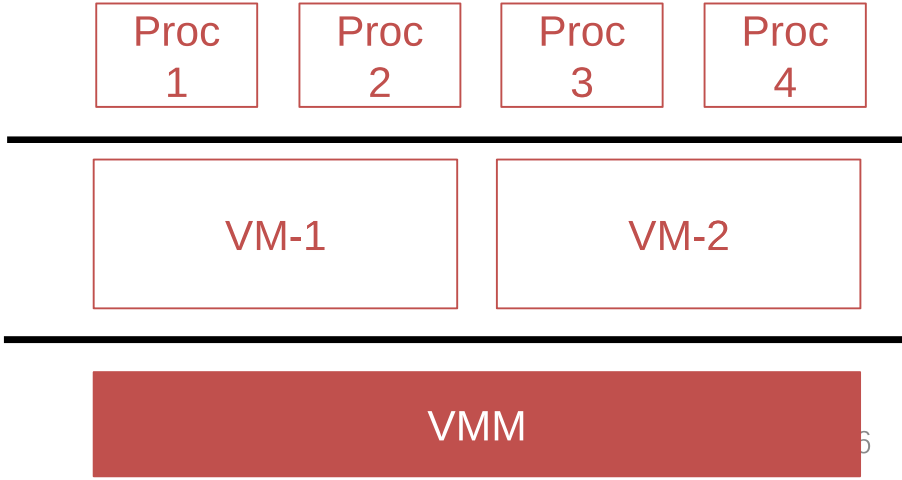
- Fine-grained isolation
- **Reduced attack surface**
- Small TCB and Reference monitor
- Defense in depth

# Reduced Attack Surface

- **Attack surface**
  - Points where an unauthorized user (the "attacker") can try to enter data to or extract data
  - The interface between two entities
    - API level
    - System call level
    - …
- **Keep attack surface as small as possible**
  - A fundamental isolation principle

# Attack Surface

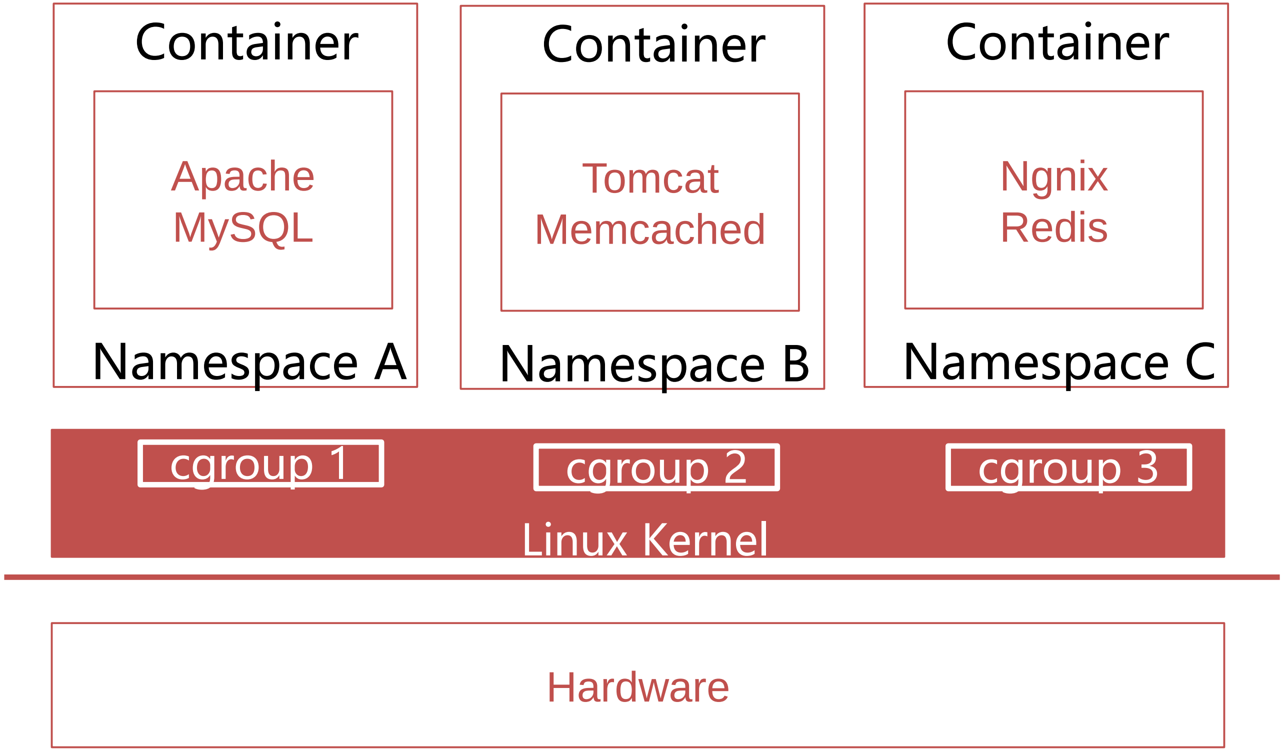
- **Interface number**
  - More than 350 system calls in Linux
  - Only 10 hypercalls in KVM
  - KVM has a much smaller attack surface if the call number matters
- **Invoked Code**
  - Small interface may cover a large number of code



# Container

- **Lightweight virtualization: container**
  - Process isolation
  - Namespace
  - Cgroups
- **All containers share the same Linux kernel**
- **Large attack surface**
  - System call

# Container



Container

Apache
MySQL

Namespace A

Container

Tomcat
Memcached

Namespace B

Container

Ngnix
Redis

Namespace C

cgroup 1

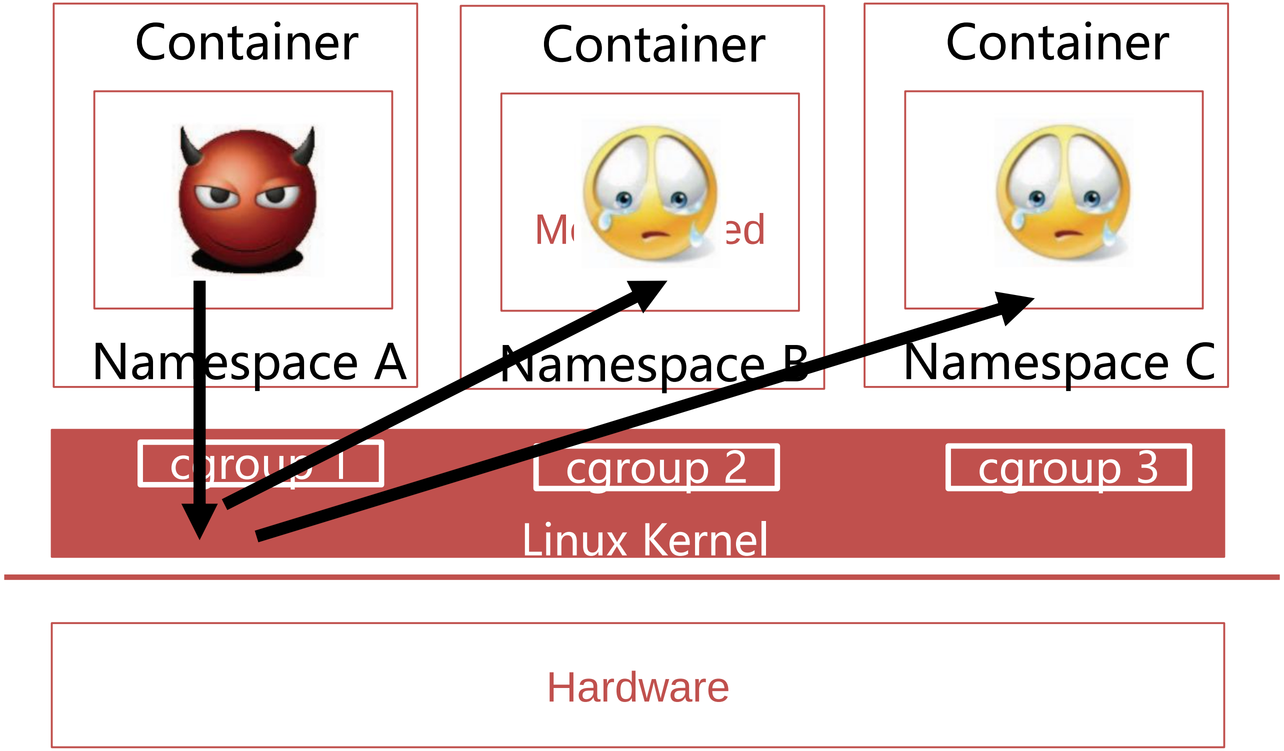
cgroup 2

cgroup 3

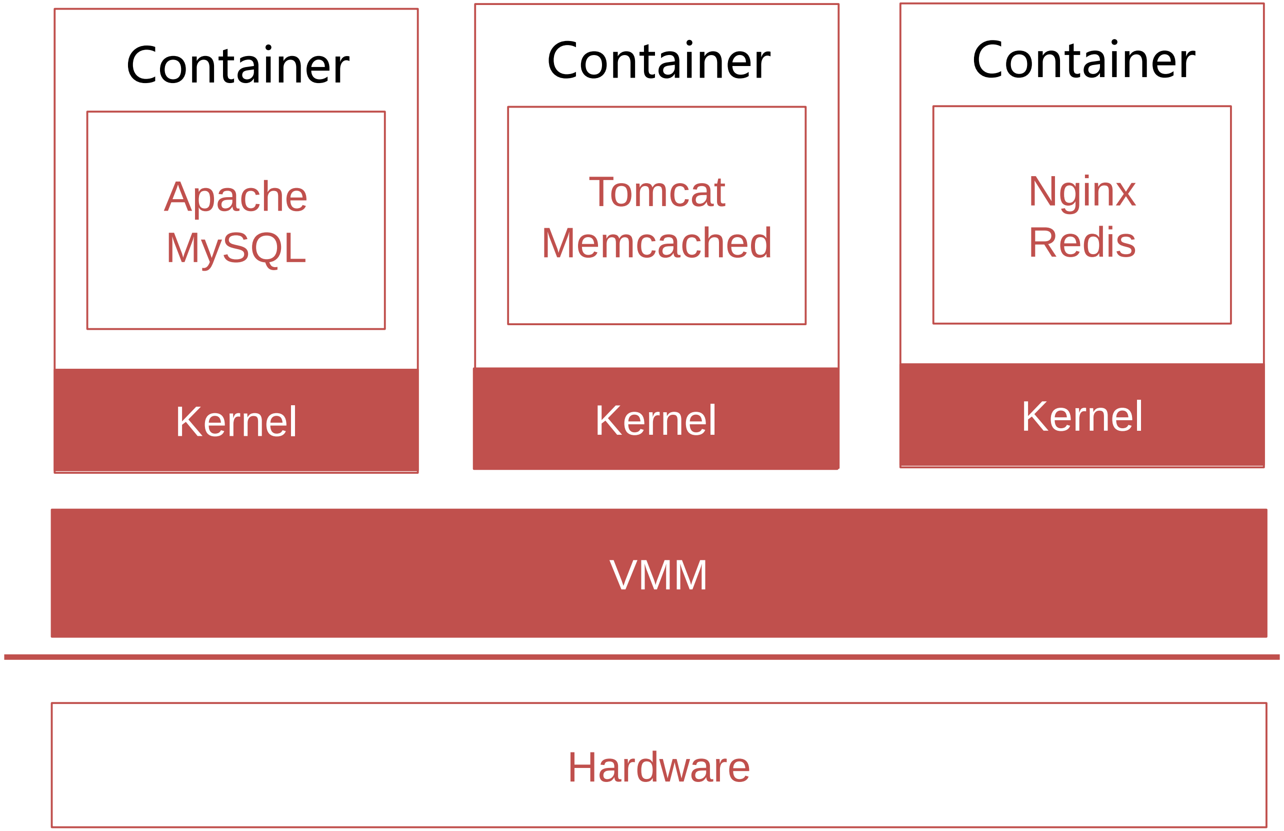
Linux Kernel

Hardware

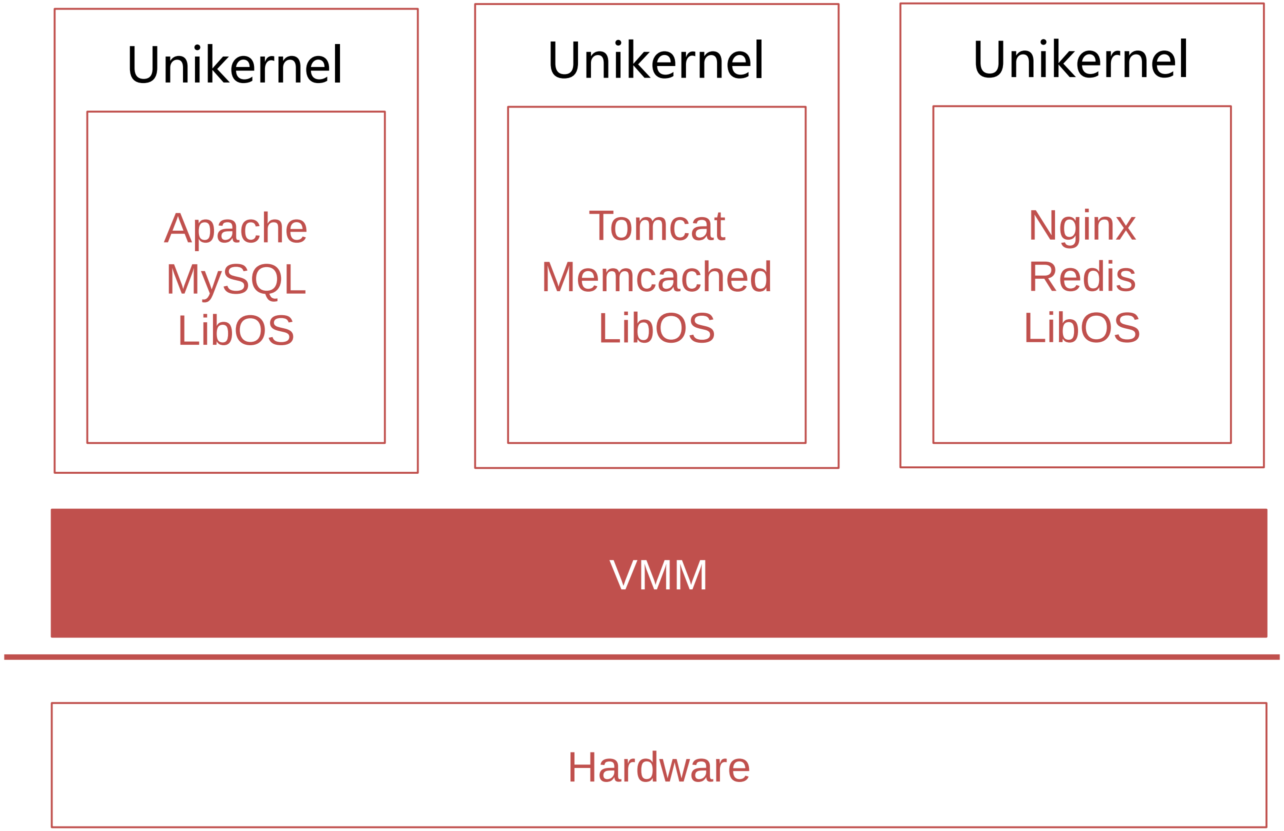
# Container: Weak Isolation



# Hyper Container

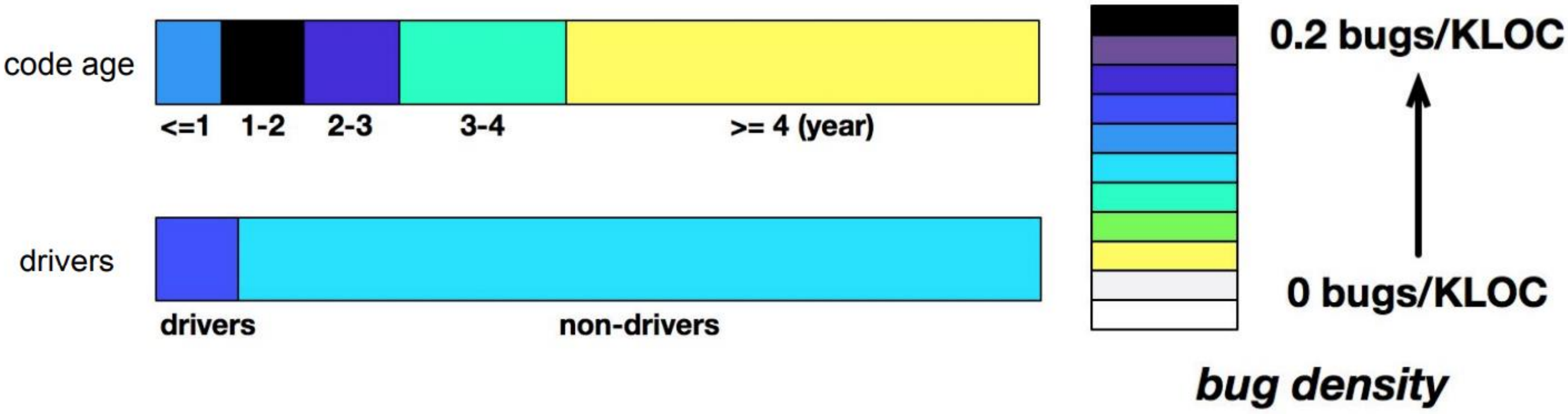


# Unikernel


Unikernel
Apache
MySQL
LibOS
Unikernel
Tomcat
Memcached
LibOS
Unikernel
Nginx
Redis
LibOS
VMM
Hardware

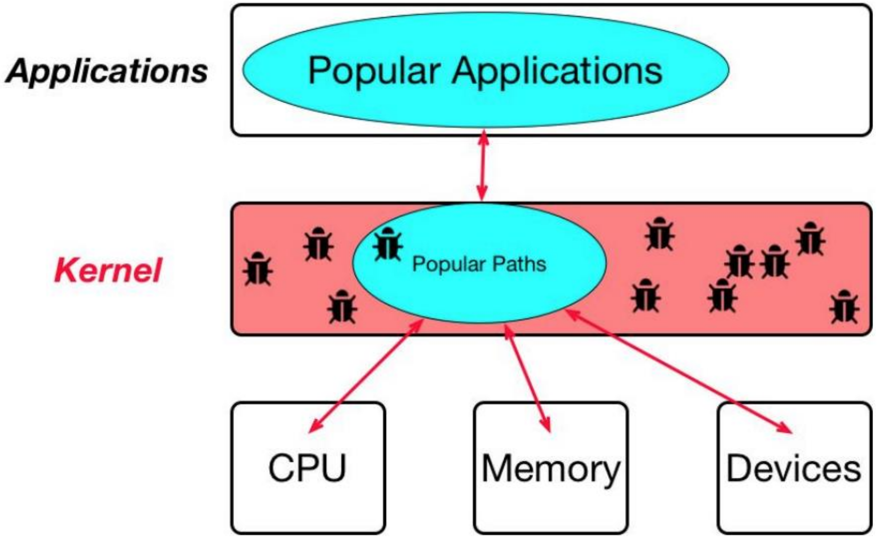
# Bug Density in Linux Kernel

- Bug density in Linux kernel



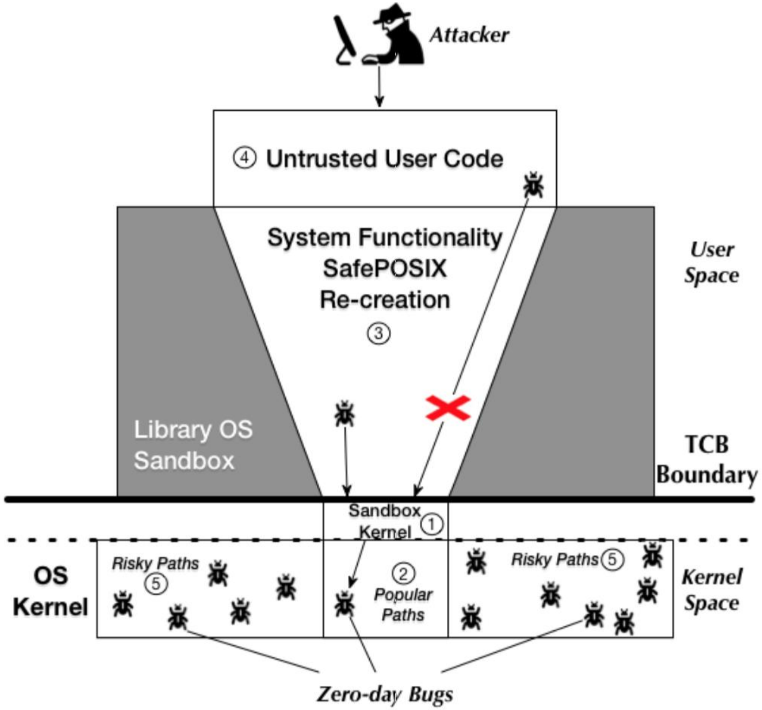
# Popular Path

- **Definition**
  - lines of code in the kernel source files, which are commonly executed in the system’ s normal workload
- **Key insight**
  - the popular paths contain many fewer bugs!



# Lock-in-Pop

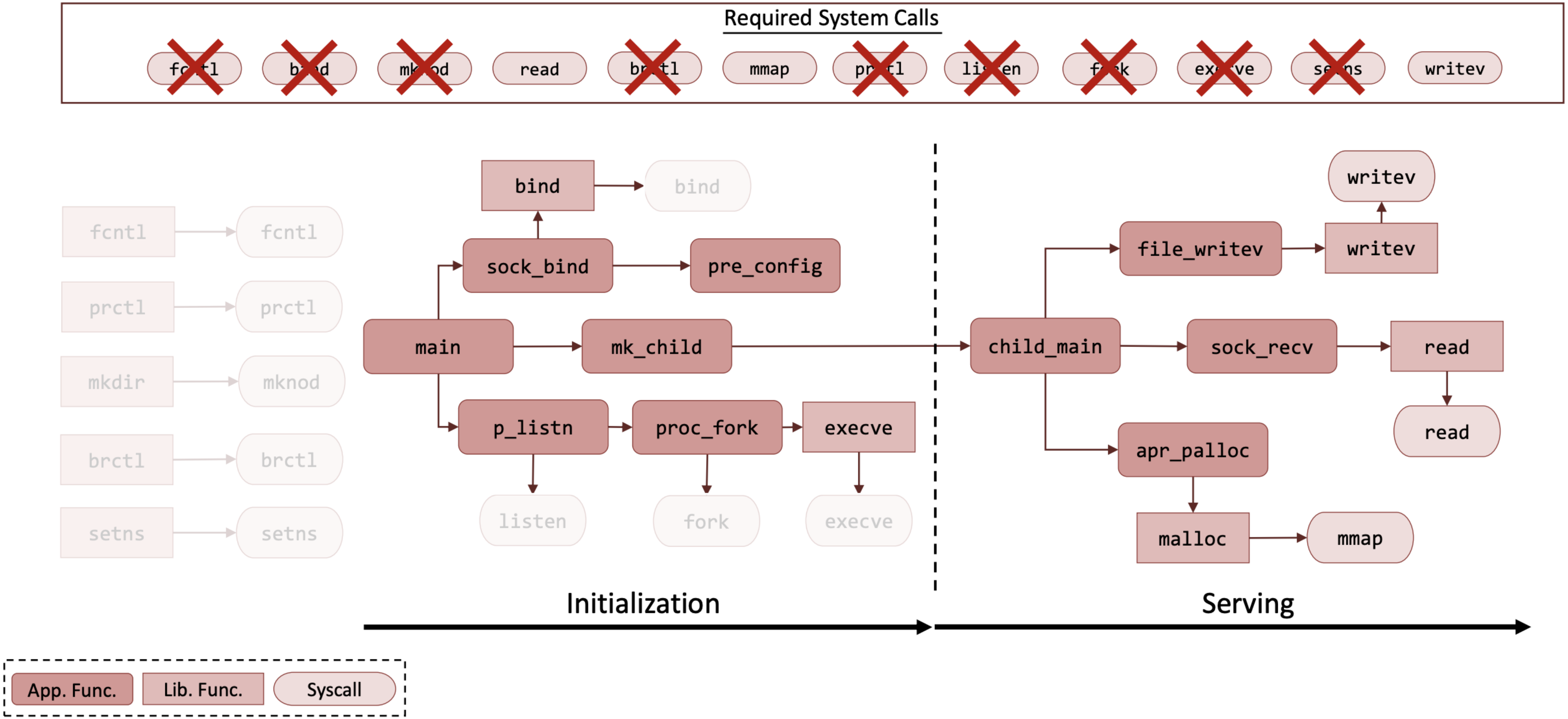
- Lock applications into using only popular paths
- Safely re-create file directories with basic calls like open(), read(), write(), close() to avoid using unpopular paths



# Considering Timeline of a Process

- **Initialization phase**
  - Read configuration files
  - Fork worker processes
  - Execute other programs
  - Create files and set their permissions
- **Serving phase**
  - Handle client requests
  - Establish connections
  - …

# Example: Apache Web Server



# Temporal System Call Specialization

- **Disable additional system calls that are needed only during the initialization phase, after entering the serving phase**

- Disables **51%** more security-critical system calls, breaking **218** more shellcodes and ROP payloads
- Mitigates **13** more Linux kernel CVEs

# Principles for System Isolation

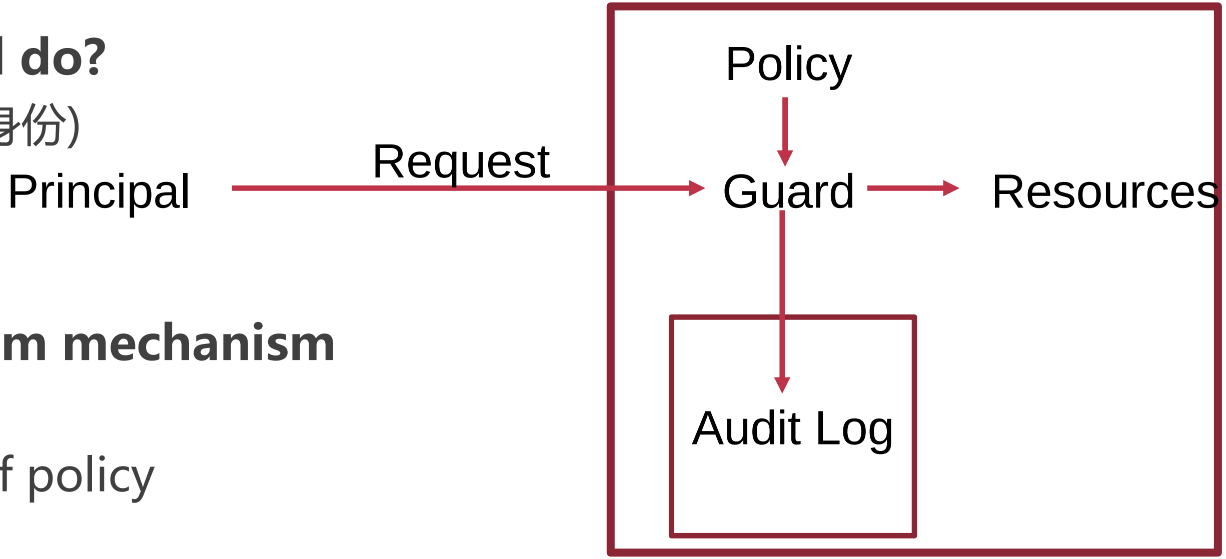
- Fine-grained isolation
- Reduced attack surface
- **Small TCB and Reference monitor**
- Defense in depth

# TCB

- **The Trusted Computing Base (TCB)**
  - A trusted component is a part of the system that we rely upon to operate correctly
- **TCB lets us separate the system into two parts**
  - The part that is security-critical (the TCB)
  - Everything else.

# Reference Monitor

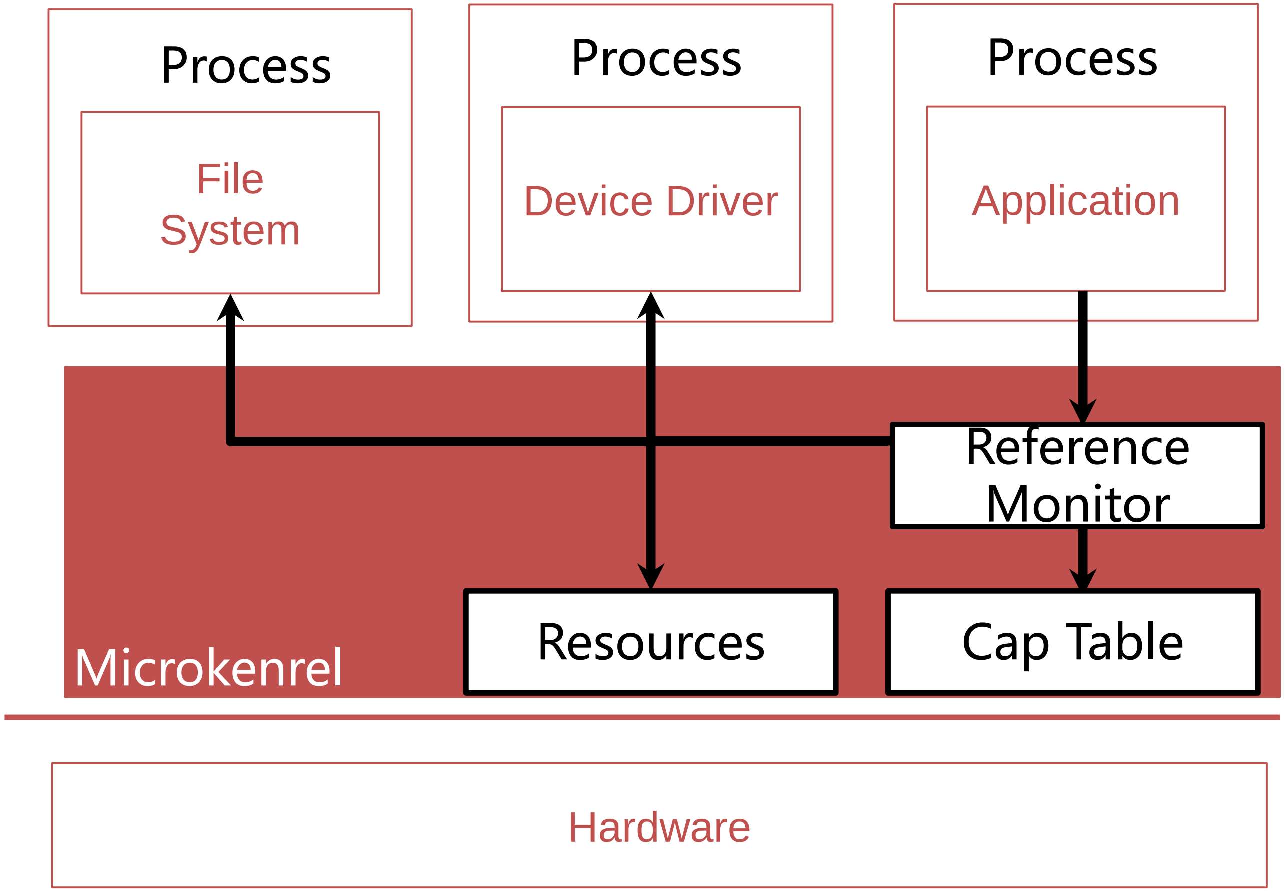
- **100% isolation is usually not what we want**
- **We need controlled sharing/interaction as well**
- **What does the guard do?**
  - Authenticate (验证身份)
  - Authorize (授权)
  - Audit (审计)
- **Separation policy from mechanism**
  - To ease reasoning
  - To ease evolution of policy



# Reference Monitor

- **Reference monitor concept was defined in 1972 by James Anderson to describe design requirements on a "reference validation mechanism"**
  - The reference validation mechanism must always be invoked (complete mediation).
  - The reference validation mechanism must be tamperproof (tamperproof).
  - The reference validation mechanism must be small enough to be subject to analysis and tests, the completeness of which can be assured (verifiable).

# Microkernel



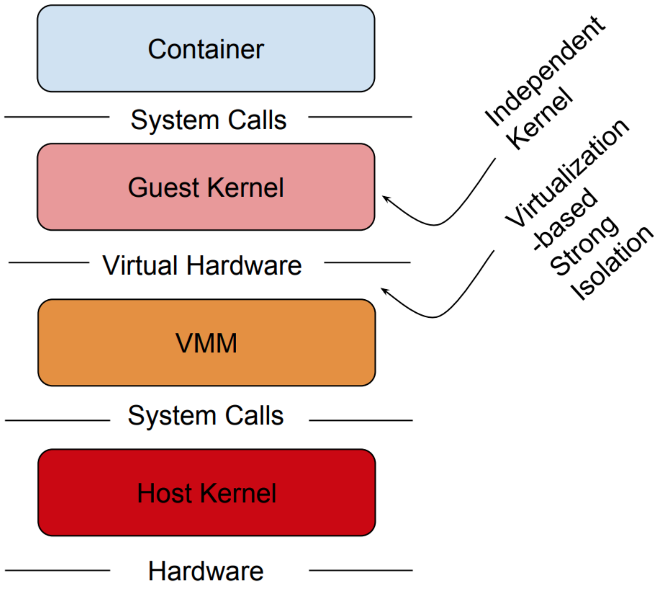
# Principles for System Isolation

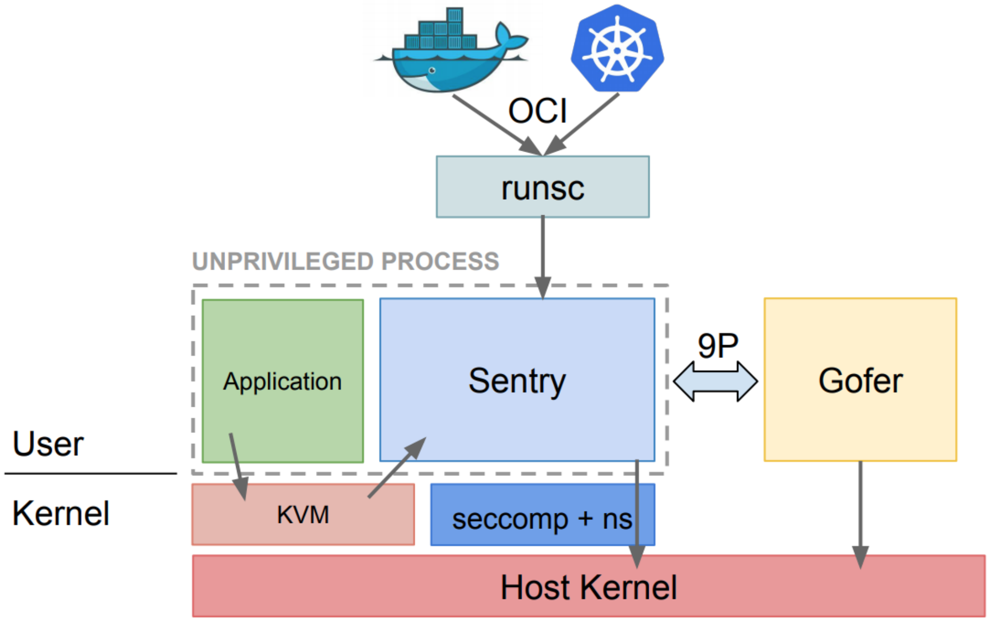
- Fine-grained isolation
- Reduced attack surface
- Small TCB and Reference monitor
- **Defense in depth**

# Defense in Depth

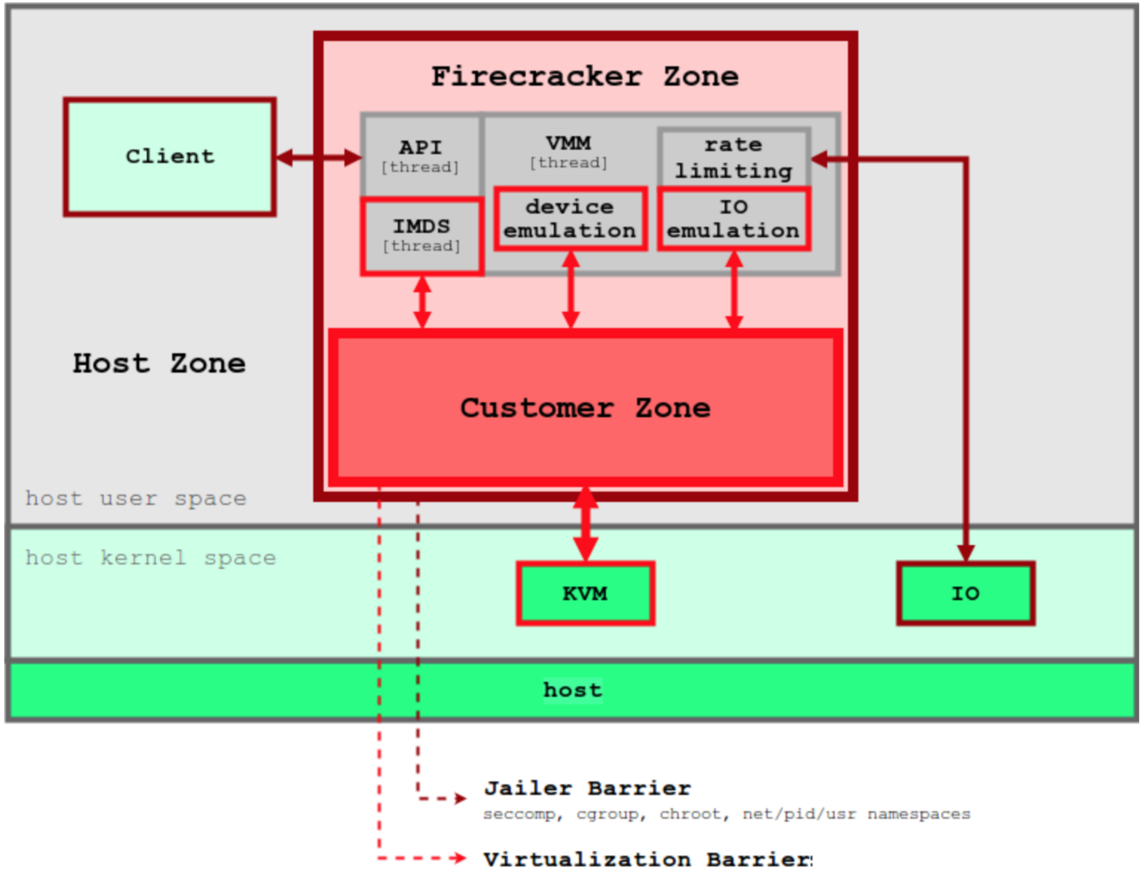
- **The notion of layering multiple types of protection together**
- **Hypothesis is that attacker needs to breech all the defenses**
- **But defense in depth isn't free:**
  - You are throwing more resources at the problem
  - And although it can be better, it is less than the sum of the parts…

# Google gVisor



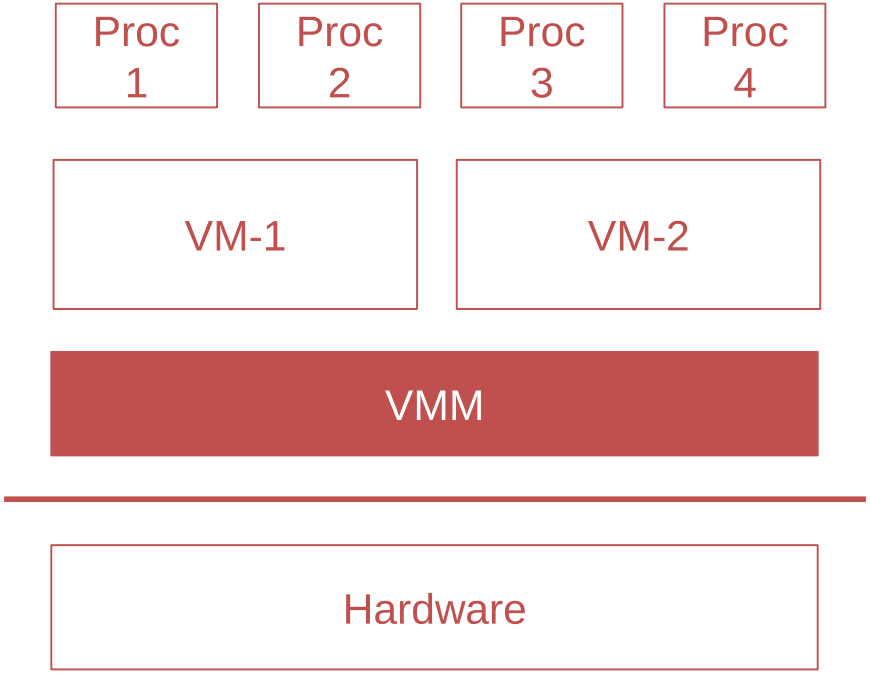


# Amazon Firecracker



# Principles for System Isolation

- **Fine-grained isolation**
  - Light-Weight Context
  - Lord of the x86 Ring
  - SeCage
  - Hodor
- **Reduced attack surface**
  - Hyper Container
  - Unikernel
  - Lock-in-Pop
- **Small TCB and Reference monitor**
  - Microkernel
  - Nested kernel
- **Defense in depth**
  - gVisor
  - Firecracker



# Conclusion

- What is system isolation?
- Operating system and hypervisor
- Isolation tools
- Principles for system isolation

**Thanks!**