BITS Pilani
Pilani Campus

Cloud Computing

Dr.P.Chinnasamy
Department of CS/IS

innovate achieve lead

BITS Pilani
Pilani Campus

CSI ZG527/ SE ZG527 – Hardware Virtualization, Memory Virtualization Problem – L4 & L5

Recap

- ✓ Comparison of Type I and Type II Hypervisor
- ✓ Resource Sharing VM-CPU
- ✓ Resource Sharing VM-Memory
- ✓ Resource Sharing VM-IO
- ✓ X86 Architecture
- ✓ Evolutions of Virtual Machine
- ✓ Emulations

Agenda

- ✓ Hardware Assisted Virtualization
- ✓ Libvirt and QEMU/KVM
- ✓ Full Virtualization VMM Architecture
- ✓ Xen and Paravirtualization
- ✓ Xen Architecture
- ✓ Memory Virtualization Problem
- ✓ Memory Reclamation Techniques

Hardware Assisted Virtualization

- ✓ Modern technique after hardware **virtualization supports in CPU**
- ✓ **Traditional x86 hardware** not supported Virtualization
- ✓ **Intel VT-x or AMD-v** support is widely used in modern computer
- ✓ CPU has a special mode of operation called **VMX mode** for running a VMs
- ✓ Many hypervisors used this mode ex. **QEMU/KVM in Linux system**

Hardware Assisted Virtualization

QEMU (User Modes)

Works with binary translation if no hardware support

KVM (Kernal Modes)

When Involved, KVM switches to VMX mode to run guest

CPU with VMX modes

CPU Switches VMX modes and Non VMX root modes

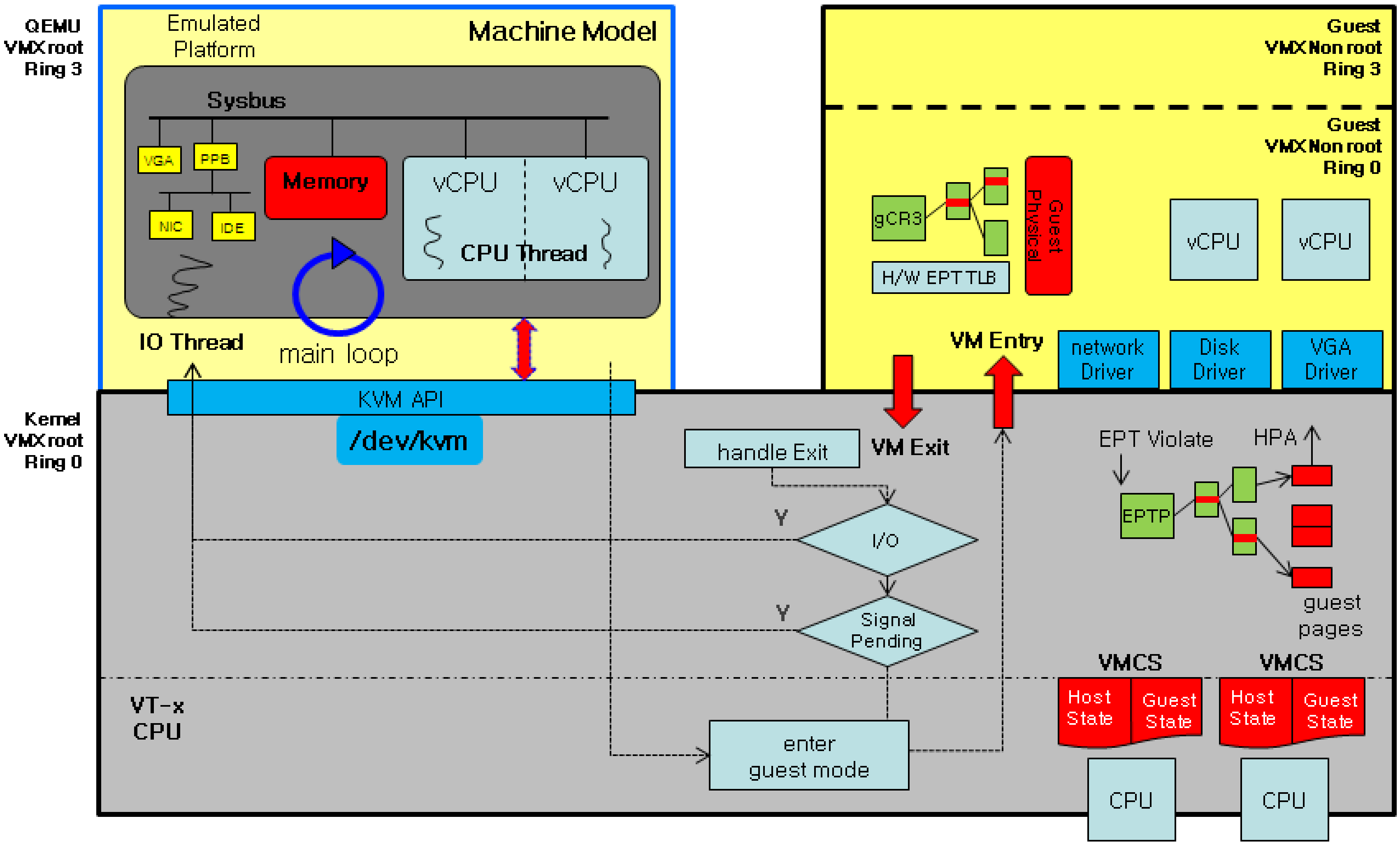
Libvirt and QEMU/KVM

- ✓ **Libvirt** is a toolkit used for **managing virtualization platforms.**
- ✓ It provides a unified interface to interact with **different hypervisors like KVM, Xen, and VirtualBox.**
- ✓ It simplifies the **management of virtual machines (VMs) by providing a common set of commands.**
- ✓ **QEMU (Quick Emulator)**: It is an open-source machine **emulator and virtualizer**. It can emulate different CPU architectures and run operating systems independently.
- ✓ **KVM (Kernel-based Virtual Machine):** It is a Linux kernel module that turns the **Linux OS into a hypervisor**, allowing it to run multiple VMs efficiently.

Libvirt and QEMU/KVM

- ✓ **sudo apt update**
- ✓ **sudo apt install qemu-kvm libvirt-daemon-system virt-manager**

QEMU Architecture



QEMU Architecture

- ✓ **QEMU (Quick Emulator)** is an open-source **machine emulator and virtualizer.**
- ✓ It allows running **operating systems and applications designed for one architecture** on a different architecture. QEMU can be used as:
- ✓ **Emulator** – It **simulates hardware to run software** compiled for different CPU architectures.
- ✓ **Virtualizer** – When used with **KVM (Kernel-based Virtual Machine)**, it provides near-native performance by using CPU virtualization extensions.

User Mode Emulation

- ✓ QEMU can run **user-mode applications** compiled for one architecture on a different architecture.
- ✓ It translates **system calls between different operating systems.**
- ✓ **Example**: **Running ARM applications on an x86 machine.**

System Mode Emulation

- ✓ Emulates a **full machine, including CPU, memory, and peripherals.**
- ✓ Allows **running an entire guest OS inside a virtual machine**.
- ✓ **Example: Running a virtual Linux instance on a Windows machine.**

Dynamic Binary Translation (DBT) Engine

- ✓ Converts **guest machine code into host machine code at runtime.**
- ✓ Ensures the emulated system runs efficiently.
- ✓ QEMU uses **TCG (Tiny Code Generator)** for translation.
- ✓ If running on the same architecture, it can use **KVM for better performance.**

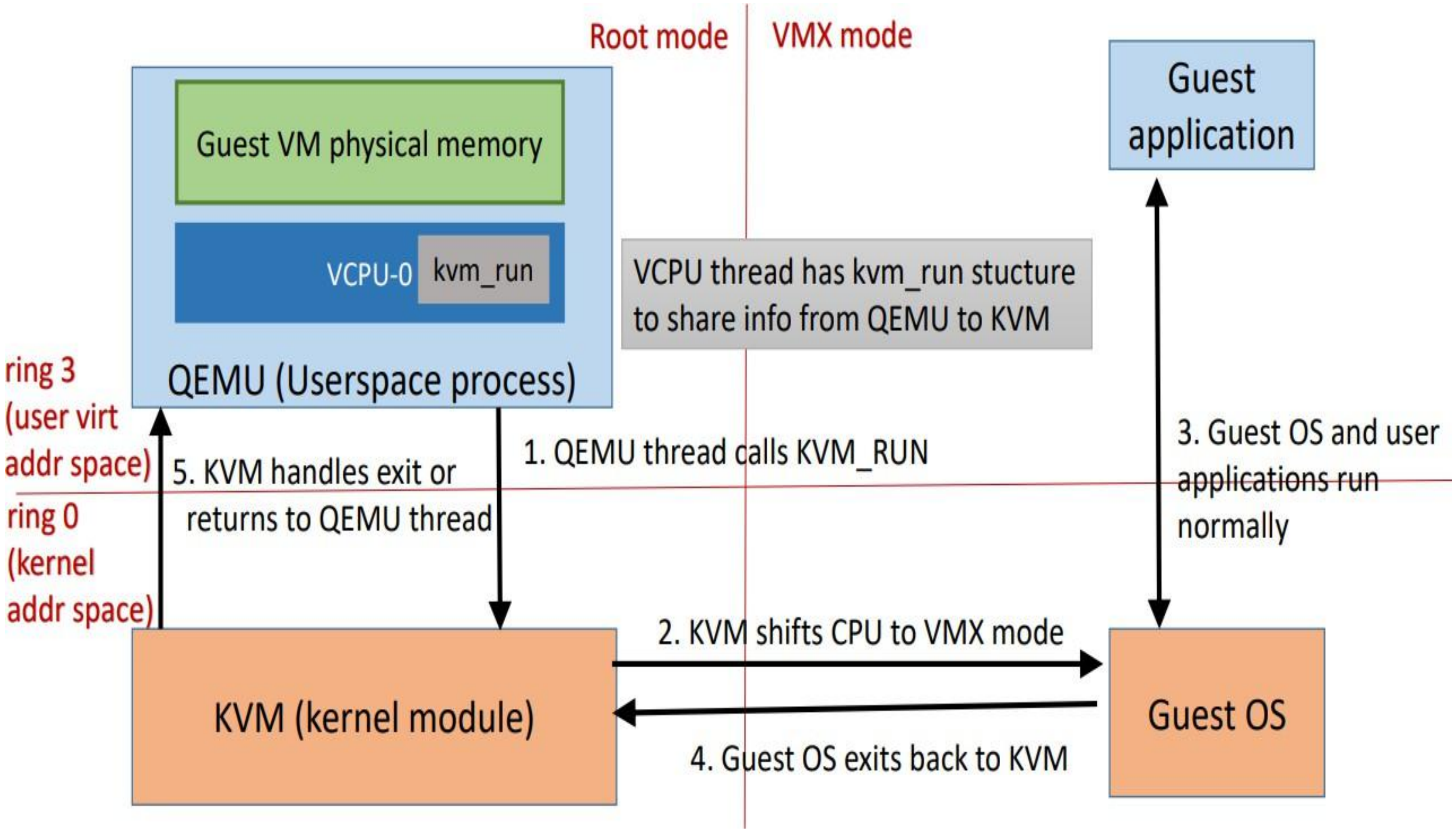
Block Device & Storage Management

- ✓ Supports various storage formats like **raw, qcow2, vmdk, vdi.**
- ✓ Provides **disk snapshotting, compression, and encryption.**

Network Emulation

- ✓ Supports multiple network backends like **user-mode networking (SLIRP), TAP, and bridge mode.**
- ✓ Allows the guest **OS to communicate with external networks.**

QEMU/KVM Operation



VM Control Structure (VMCS)

- ✓ The VM Control Structure (VMCS) is a **key data structure** used in **Intel VT-x (Virtualization Technology for x86 processors)**.
- ✓ It is responsible for **managing transitions between the Virtual Machine Monitor (VMM) (or hypervisor)** and the **Guest Virtual Machine (VM).**

Guest State Area

- ✓ Stores the **state of the virtual machine before a VM exit occurs**.
- ✓ **CPU registers (RIP, RSP, RFLAGS)**
- ✓ **Control registers (CR0, CR3, CR4)**
- ✓ **Segment registers (CS, DS, SS, ES)**
- ✓ **Debug registers (DR7)**

Host State Area

- ✓ Stores the **state of the hypervisor (VMM) when switching from guest to host.**
- ✓ **Host’s RIP and RSP (to resume execution in the VMM)**
- ✓ **Control registers (CR0, CR3, CR4)**
- ✓ **Segment selectors for the host OS**

VM Execution Control Fields

- ✓ Controls how the **VM runs and interacts with hardware.**
- ✓ **Example controls:**
- ✓ **I/O bitmaps:** Determines if **I/O instructions cause a VM exit.**
- ✓ **Memory protection:** Configures **extended page tables (EPT).**
- ✓ **Interrupt handling:** Defines **how interrupts are handled.**

VM Exit Control Fields

- ✓ Determines **when and how the guest exits to the VMM.**
- ✓ **Example exit conditions:**
- ✓ **CPUID instruction execution**.
- ✓ Accessing **specific memory addresses.**
- ✓ Triple **faults or hardware exceptions**.

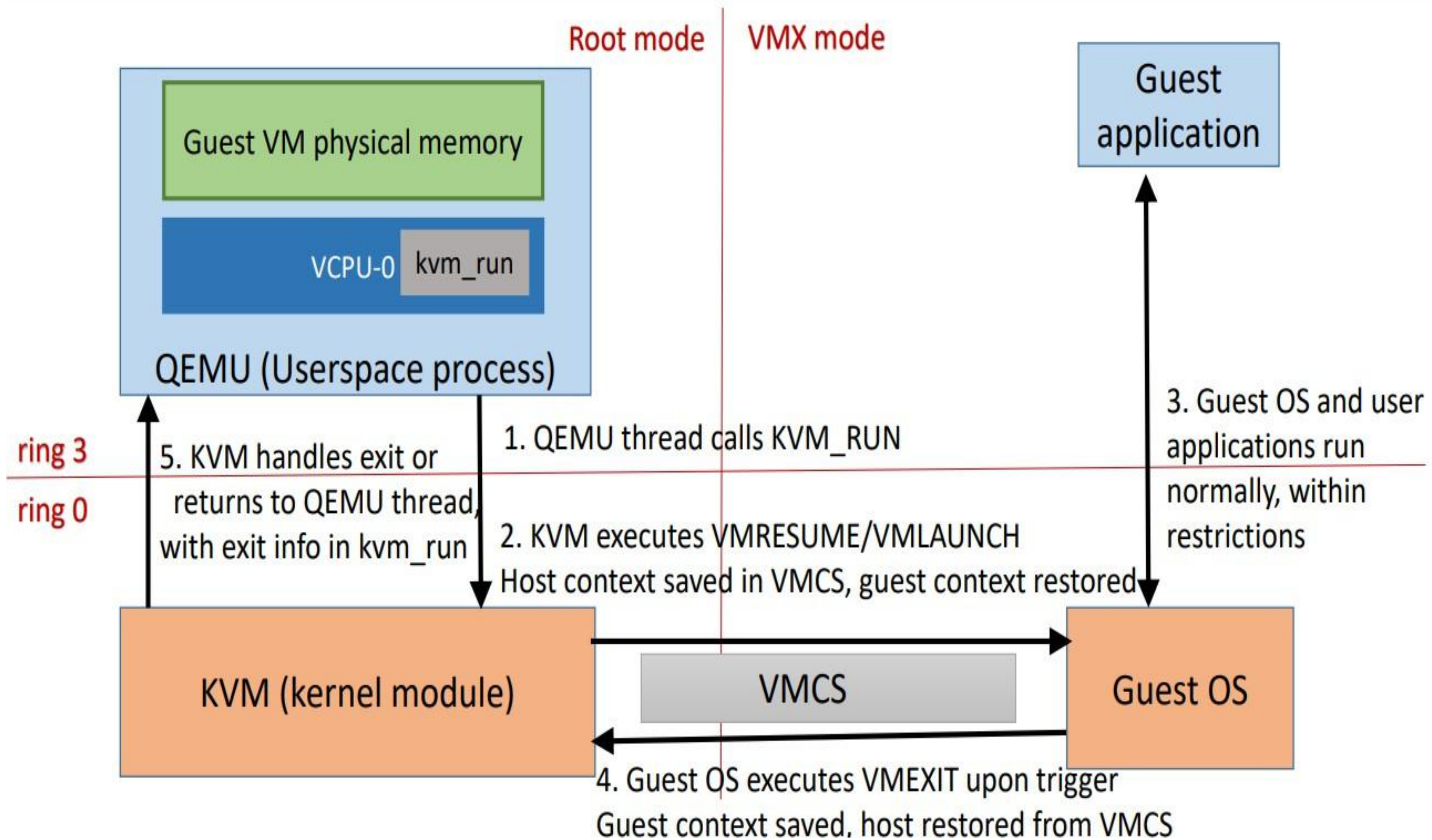
VM Entry Control Fields

- ✓ Controls the **process of entering the guest VM from the hypervisor.**
- ✓ Defines:
- ✓ **Guest execution mode (protected mode, real mode, etc.)**
- ✓ **Event injection (forcing the guest to handle certain interrupts).**
- ✓ **Register loading policies.**

How VMCS works?

- **VM Entry:** The VMM (hypervisor) loads the guest state from VMCS and starts the VM.
- **Guest Execution:** The guest VM runs its OS and applications.
- **VM Exit:** If a privileged instruction or specific event occurs, the CPU switches to the VMM, storing the guest's state in VMCS.
- **VMM Execution:** The hypervisor handles the event and updates the VMCS.
- **VM Resume:** The hypervisor switches back to the guest using the stored VMCS state.

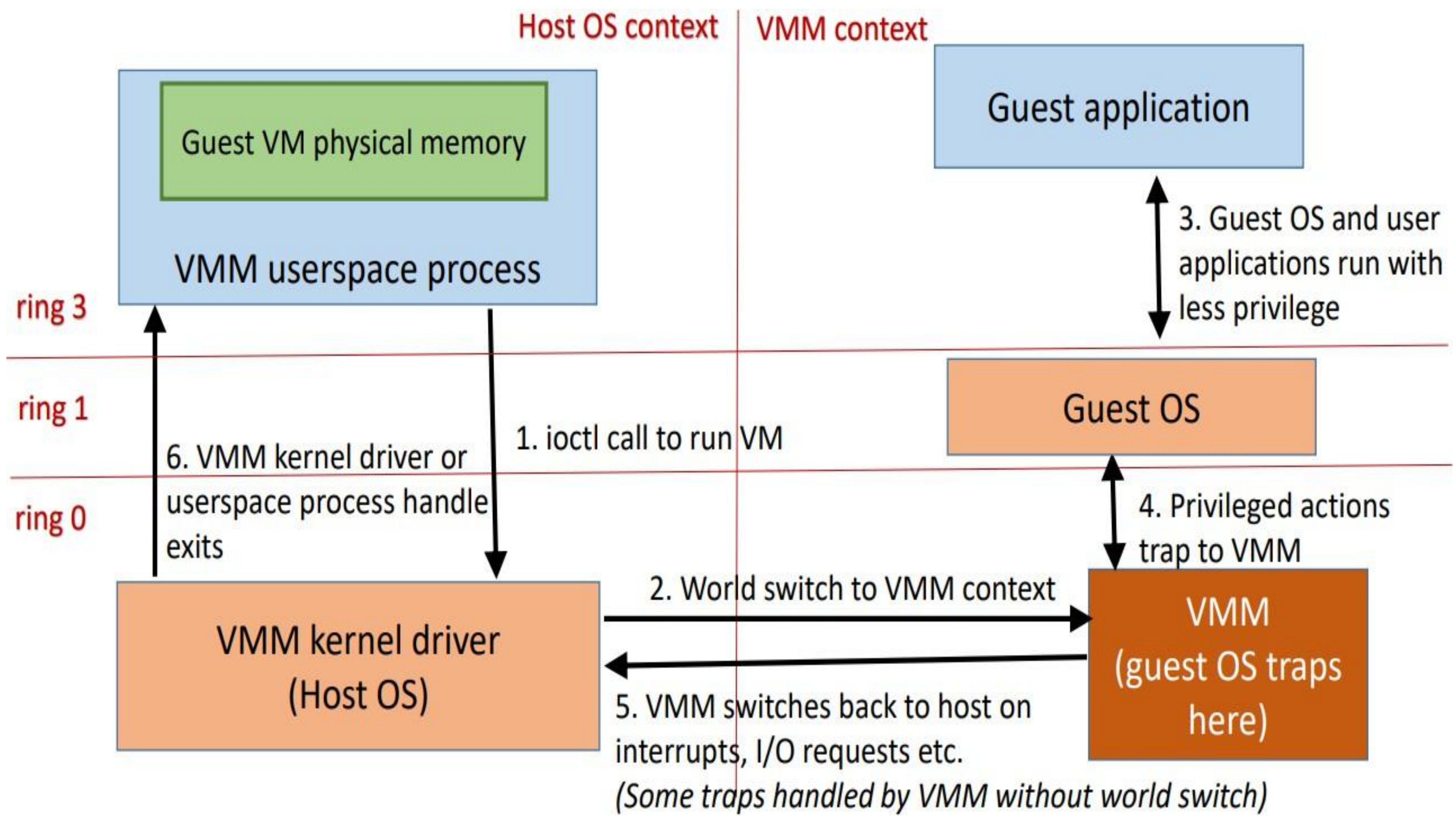
QEMU/KVM Operation revisited



Full Virtualization

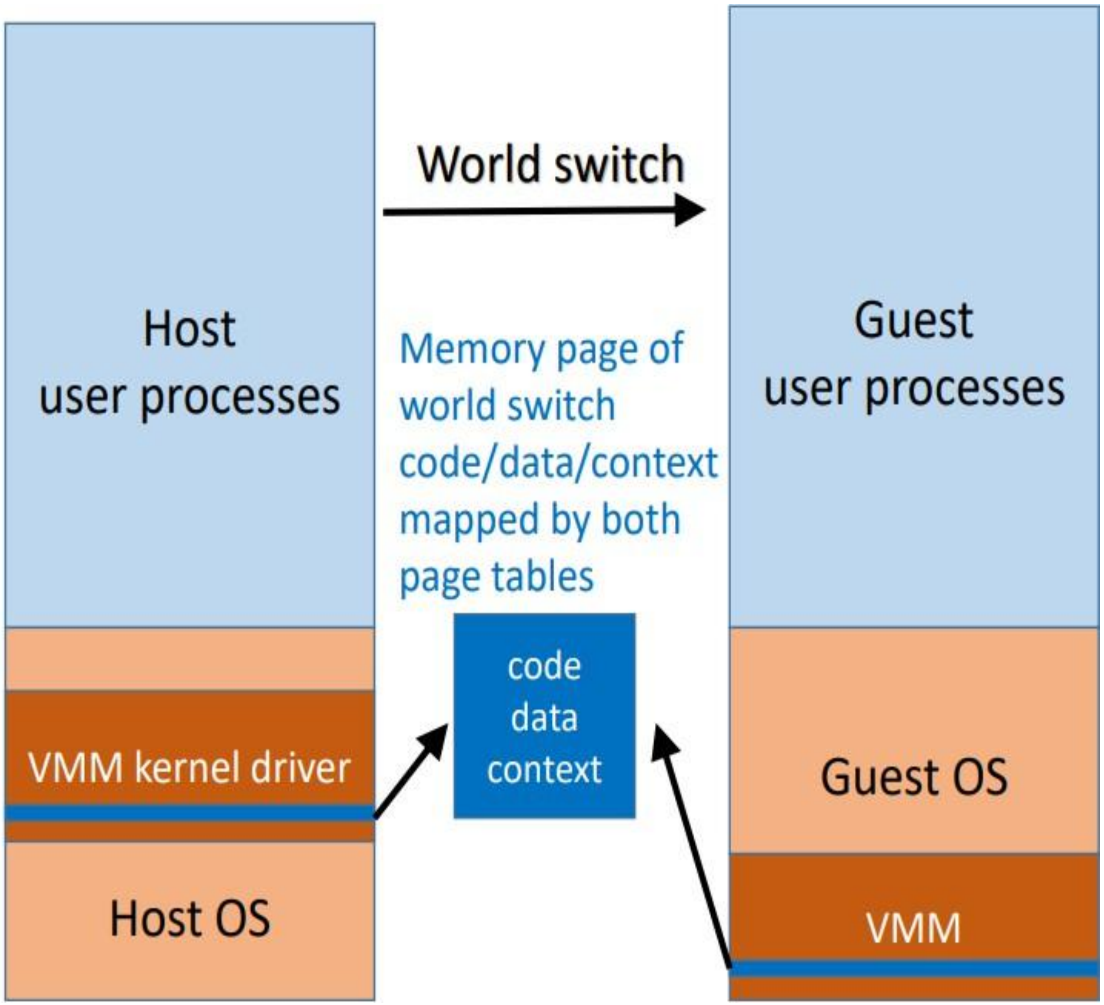
- ✓ Full Virtualization is a virtualization technique that allows an **unmodified guest operating system (OS) to run in a virtual machine (VM) without requiring changes to the OS kernel.**
- ✓ It is achieved by emulating all **hardware resources and managing privileged CPU instructions efficiently**.
- ✓ On x86 architecture, full virtualization is implemented using **software-based techniques (like Binary Translation) or hardware-assisted virtualization (like Intel VT-x and AMD-V).**

Full Virtualization VMM Architecture



Hosts and VMM Contexts

- Each context has separate page tables, CPU registers, IDTs and so on
- VMM context: VMM occupies top 4MB of address space
- Memory page containing code/data of world switch mapped in both contexts
  - Host/VMM context saved/restored in this special "cross" page by VMM



Difference between QEMU Vs KVM

Feature	QEMU (Quick Emulator)	KVM (Kernel-based Virtual Machine)
Type	**Emulator & Virtualizer**	**Hypervisor (Kernel Module)**
Functionality	Emulates full hardware, including CPU, memory, and peripherals	Uses CPU virtualization to run VMs at near-native speed
Performance	**Slower (due to full emulation)**	**Faster (hardware-assisted virtualization)**
Virtualization Method	Software-based (**binary translation)**	Hardware-assisted **(Intel VT-x, AMD-V)**
Guest OS Support	Supports multiple architectures **(x86, ARM, PowerPC, etc.)**	Supports only **x86-based OSes**
Use Case	Used when **hardware virtualization is not available**	Used for **high-performance virtualization**

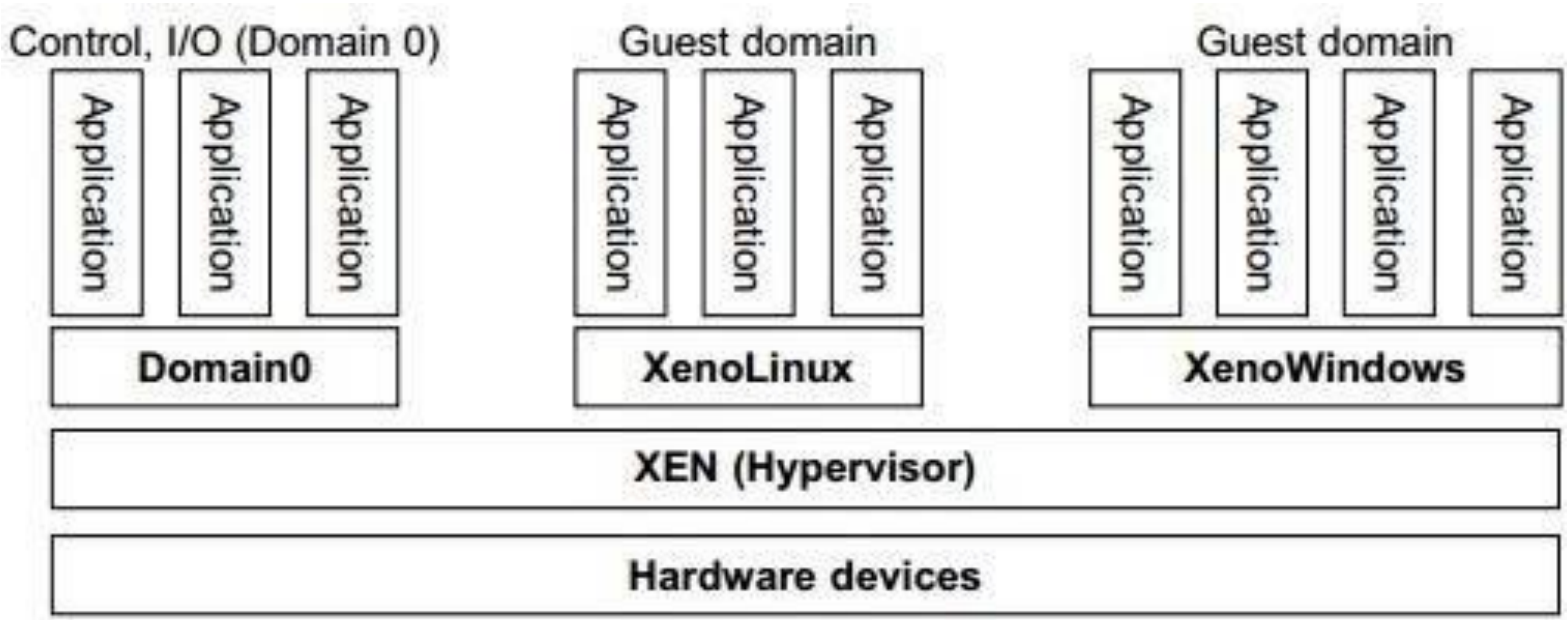
Difference between QEMU Vs KVM

Feature	QEMU (Quick Emulator)	KVM (Kernel-based Virtual Machine)
Dependency on Host OS	Works independently, does not require a specific kernel	Works as a **Linux kernel module**
Networking & Storage	**Provides virtual networking & disk I/O emulation**	Uses **Linux networking stack & storage drivers**
Snapshot & Migration	Supports live snapshots & migration	Supports live migration with better performance
Integration	Can work standalone or with KVM	Needs **QEMU for device emulation**
Best Use Case	Running different CPU architectures (e.g., ARM on x86)	Running Linux VMs with high efficiency

Xen and Para Virtualization

- ✓ Para-virtualization is a **virtualization technique** where the guest OS is **aware that it is running in a virtualized environment**

Xen Architecture



Trap and Emulate Architecture

- ✓ The **Trap and Emulate** architecture is a fundamental technique used in **CPU virtualization**.
- ✓ It enables a **guest operating system (OS)** to run inside a **virtual machine (VM)** without modification, even if it executes privileged instructions.
- ✓ This method ensures that **guest OS operations that require direct hardware access** are intercepted (**trap**) by the hypervisor, which then simulates (**emulate**) their execution in a safe manner.

CPU Virtualization in Xen

- Guest OS code modified to not invoke any privileged instruction
  - Any privileged operation traps to Xen in ring 0
- Hypercalls: guest OS voluntarily invokes Xen to perform privileged ops
  - Much like system calls from user process to kernel
  - Synchronous: guest pauses while Xen services the hypercall
- Asynchronous event mechanism: communication from Xen to domain
  - Much like interrupts from hardware to kernel
  - Used to deliver hardware interrupts and other notifications to domain
  - Domain registers event handler callback functions

Trap Handling in Xen

innovate achieve lead

- When trap/interrupt occurs, Xen copies the trap frame onto the guest OS kernel stack, invokes guest interrupt handler
- Guest registers an interrupt descriptor table with Xen to handle traps
  - Interrupt handlers validated by Xen (check that no privileged segments loaded)
- Guest trap handlers work off information on kernel stack, no modifications needed to guest OS code
  - Except page fault handler, which needs to read CR2 register to find faulting address (privileged operation)
  - Page fault handler modified to read faulting address from kernel stack (address placed on stack by Xen)
- What if interrupt handler still invokes privileged operations?
  - Traps to Xen again and Xen detects this "double fault" (trap followed by another trap from interrupt handler code) and terminates misbehaving guest

Memory Virtualization in Xen

- One copy of combined GVA→HPA page table maintained by guest OS
  - CR3 points to this page table
  - Like shadow page tables, but in guest memory, not in VMM
- Guest is given read-only access to guest "RAM" mappings (GPA→HPA)
  - Using this, guest can construct combined GVA→GPA mapping
- Guest page table is in guest memory, but validated by Xen
  - Guest marks its page table pages as read-only, cannot modify
  - When guest needs to update, it makes a hypercall to Xen to update page table
  - Xen validates updates (is guest accessing its slice of RAM?) and applies them
  - Batched updates for better performance
- Segment descriptor tables are also maintained similarly
  - Read-only copy in guest memory, updates validated and applied by Xen
  - Segments truncated to exclude top 64MB occupied by Xen

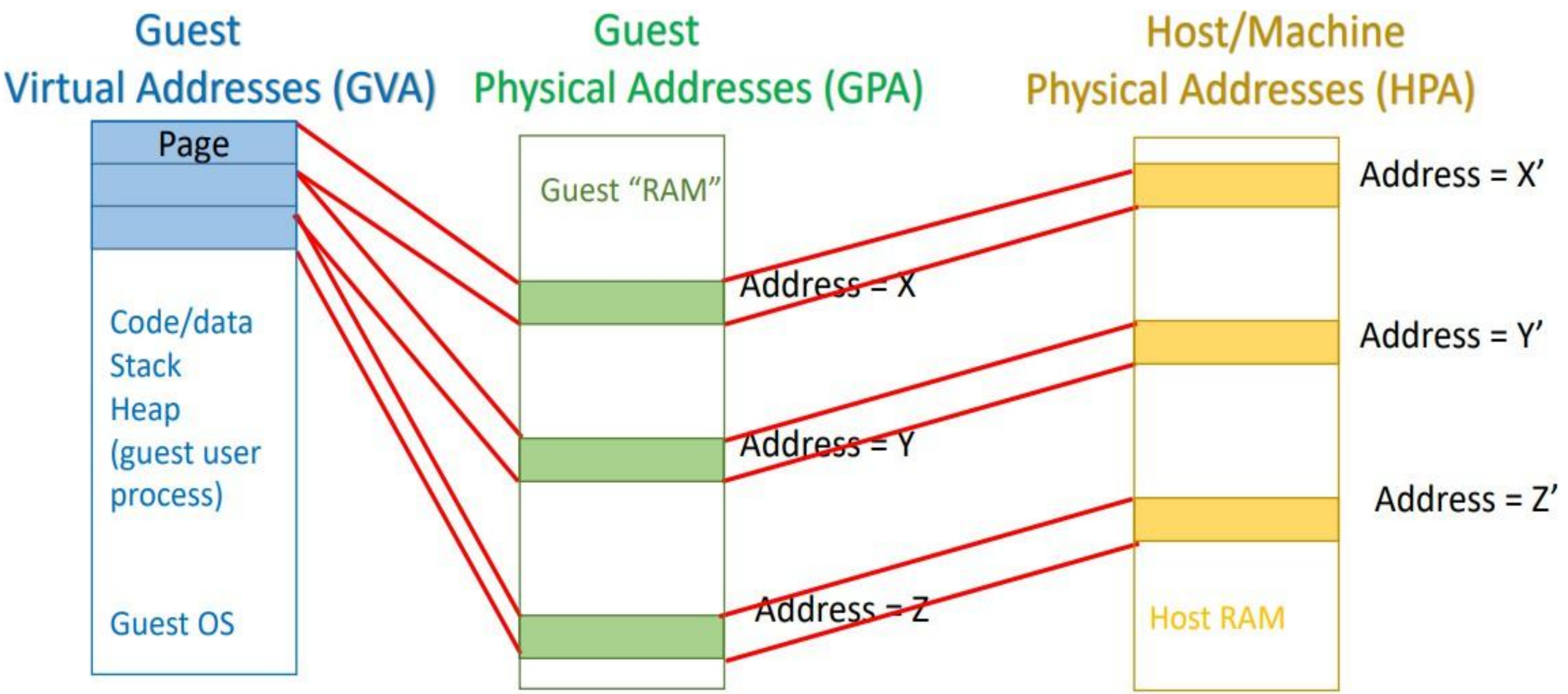
Memory Virtualization

- ✓ Memory virtualization is a critical component of system virtualization, allowing **multiple virtual machines (VMs) to share the physical memory of a host system.**
- ✓ However, it introduces several challenges related to **memory management, performance, and security.**

Virtual Memory Layers

- **Guest Virtual Address (GVA)** – Address used by the guest application.
- **Guest Physical Address (GPA)** – Address used by the guest OS.
- **Host Physical Address (HPA)** – Actual memory address on the host machine.

Virtual Memory Layers



Guest "RAM" is actually memory of the userspace hypervisor process running on the host, which is mapped to host memory by the host's page table

Thank You