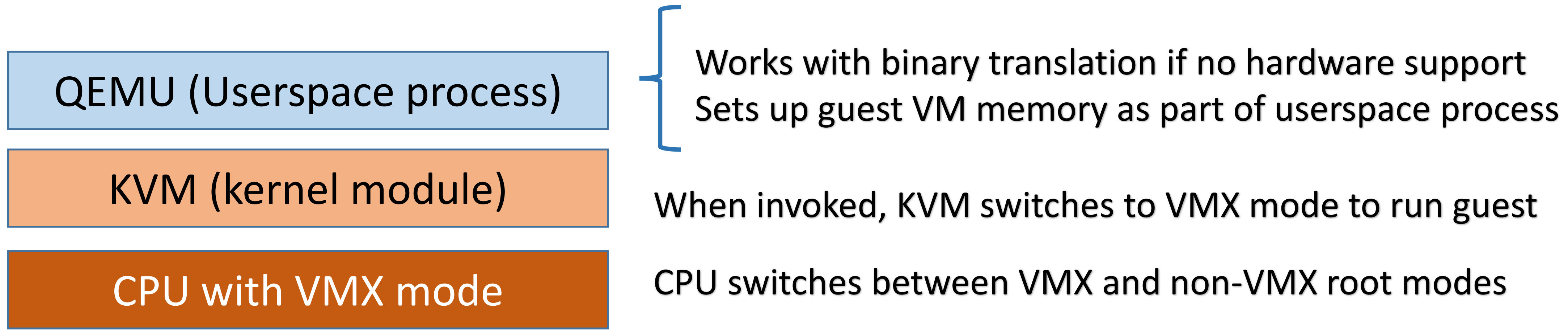
Hardware-assisted CPU Virtualization in KVM/QEMU

Hardware-assisted Virtualization

- Modern technique, after hardware support for virtualization introduced in CPUs
  - Original x86 CPUs did not support virtualization
  - Intel VT-X or AMD-V support is widely available in modern systems
  - Special CPU mode of operation called VMX mode for running VMs
- Many hypervisors use this H/W feature, e.g., QEMU/KVM in Linux



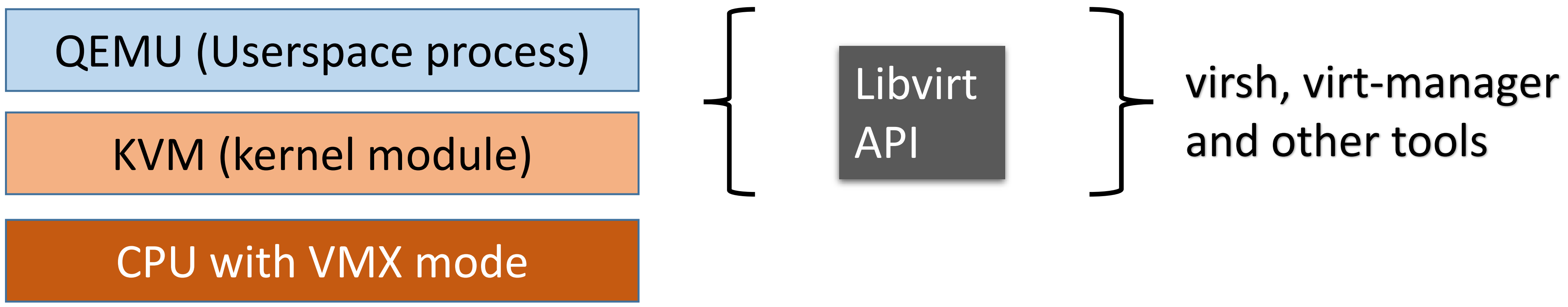
Works with binary translation if no hardware support
Sets up guest VM memory as part of userspace process

When invoked, KVM switches to VMX mode to run guest

CPU switches between VMX and non-VMX root modes

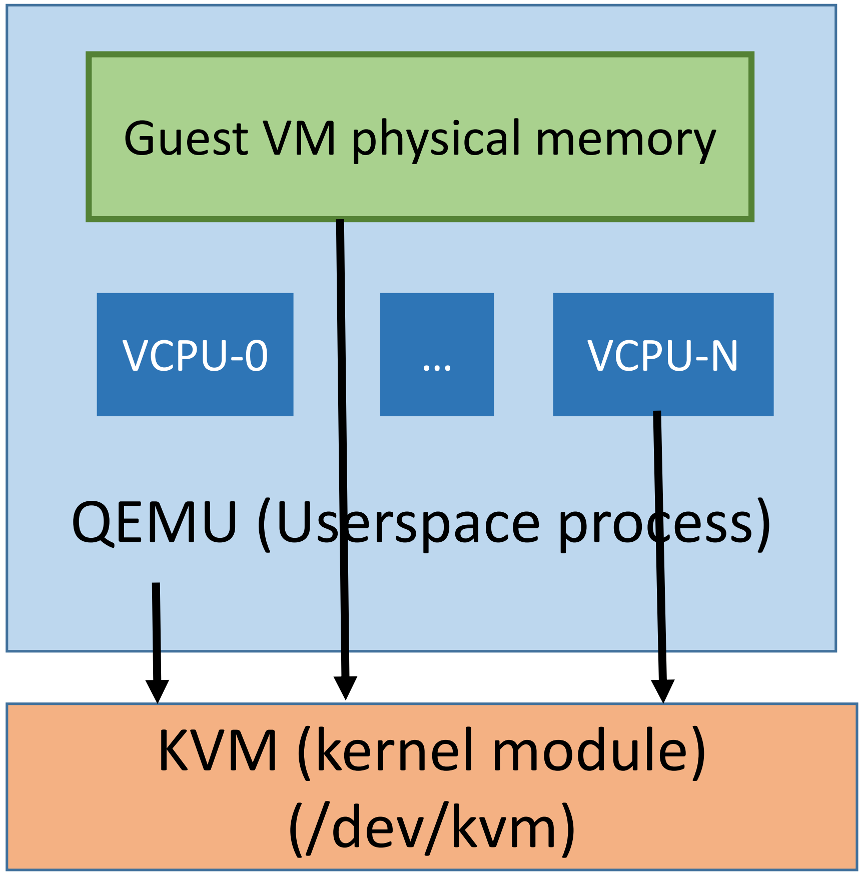
Libvirt and QEMU/KVM

- When you install QEMU/KVM on Linux, libvirt is also installed
  - A set of tools manage hypervisors, including QEMU/KVM
  - A daemon runs on the system and communicates with hypervisors
  - Exposes an API using which hypervisors can be managed, VM created etc.
  - Commandline tool (virsh) and GUI (virt-manager) use this API to manage VMs



QEMU architecture

- QEMU is userspace process
- KVM exposes a dummy device
  - QEMU talks to KVM via open/ioctl syscalls
- Allocates memory via mmap for guest VM physical memory
- Creates one thread for each virtual CPU (VCPU) in guest
- Multiple file descriptors to /dev/kvm (one for QEMU, one for VM, one for VCPU and so on)
  - ioctl on fds to talk to KVM
- Host OS sees QEMU as a regular multi-threaded process



QEMU operation

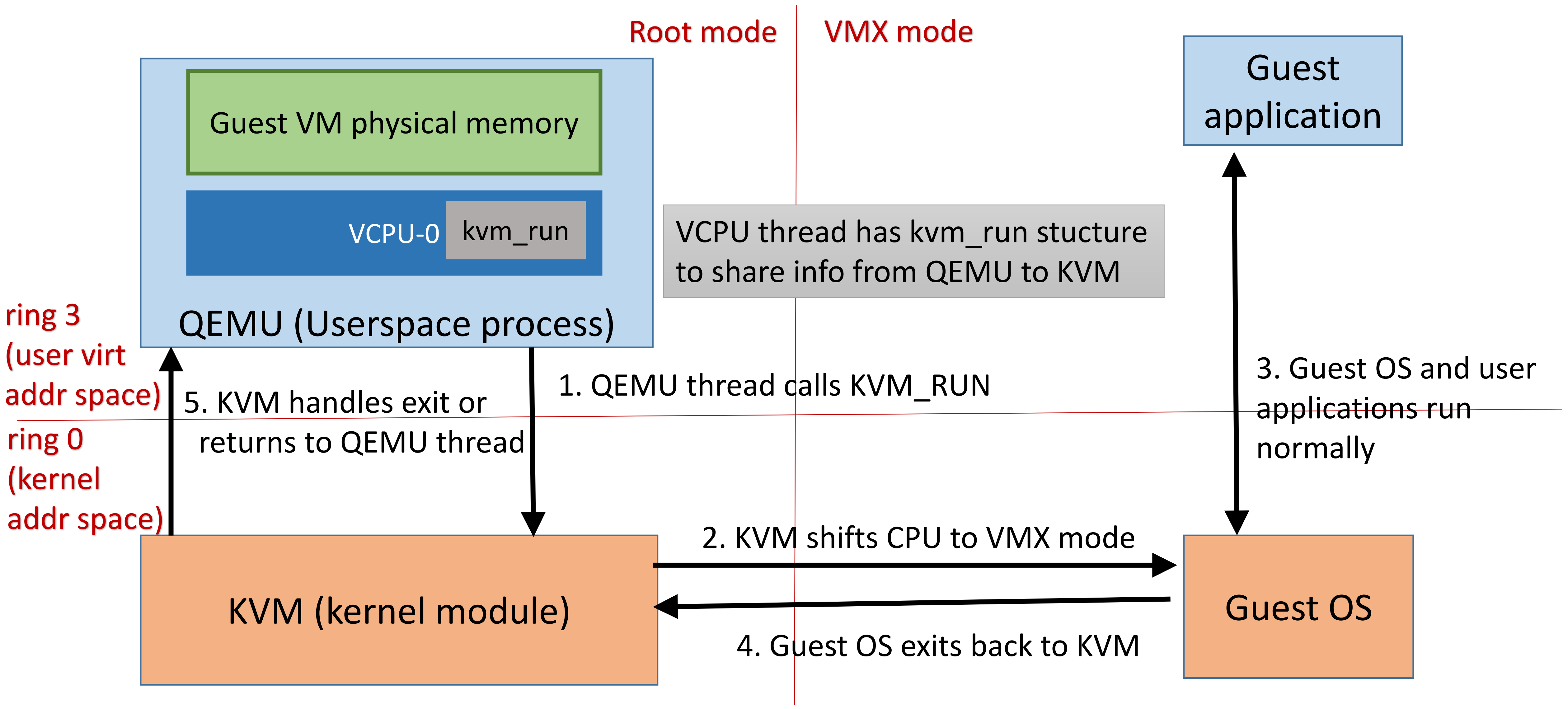
```
open(/dev/kvm)
ioctl(qemu_fd, KVM_CREATE_VM)
ioctl(vm_fd, KVM_CREATE_VCPU)

for(;;) { //each VCPU runs this loop
  ioctl(vcpu_fd, KVM_RUN)
  switch(exit_reason) {
    case KVM_EXIT_IO: //do I/O
    case KVM_EXIT_HLT:
  }
}
```

This ioctl system call blocks this thread, KVM switches to VMX mode, runs guest VM

Returns to QEMU on host when VM exits from VMX mode.

QEMU handles exit and returns to guest VM

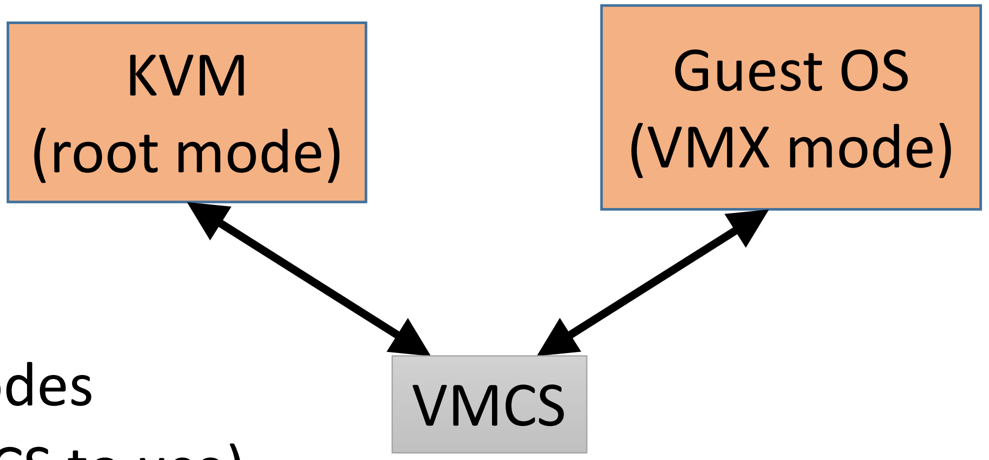
QEMU/KVM operation


Root mode
VMX mode
Guest VM physical memory
VCPU-0
kvm_run
VCPU thread has kvm_run stucture to share info from QEMU to KVM
QEMU (Userspace process)
Guest application
ring 3 (user virt addr space)
ring 0 (kernel addr space)
1. QEMU thread calls KVM_RUN
2. KVM shifts CPU to VMX mode
3. Guest OS and user applications run normally
4. Guest OS exits back to KVM
5. KVM handles exit or returns to QEMU thread
KVM (kernel module)
Guest OS

VMX mode

- Special CPU instructions to enter and exit VMX mode
  - VMLAUNCH, VMRESUME invoked by KVM to enter VMX mode
  - VMEXIT invoked by guest OS to exit VMX mode
- On VMX entry/exit instructions, CPU switches context between host OS to guest OS
  - Page tables (address space), CPU register values etc switched
  - Hardware manages the mode switch
- Where is CPU context stored during mode switch?
  - Cannot be stored in host OS or guest OS data structures alone (why?)
  - VMCS (VM control structure), also called VMCB (VM control block)

VM control structure (VMCS)

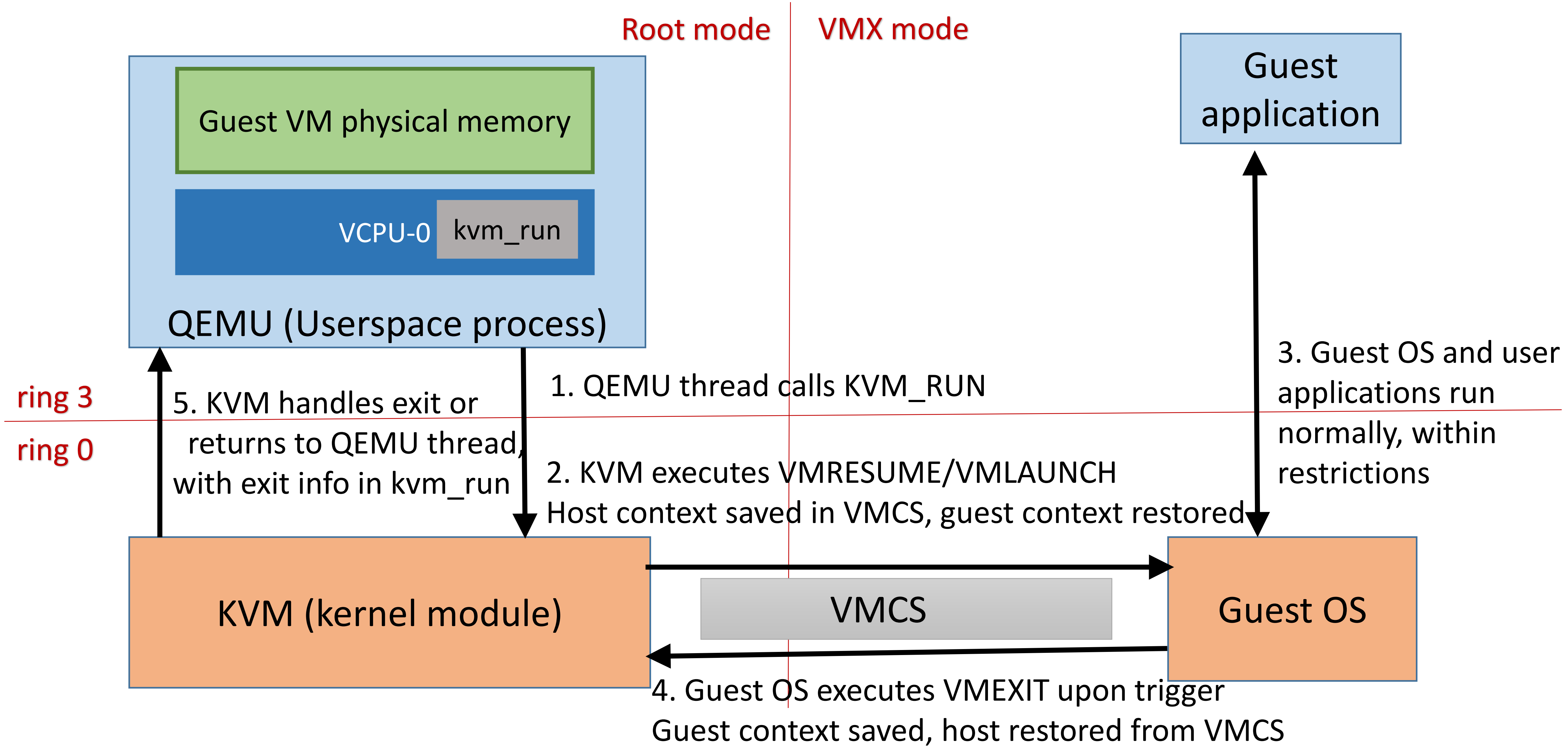


- What is VMCS?
  - Common memory area accessible in both modes
  - One VMCS per VM (KVM tells CPU which VMCS to use)
- What is stored in VMCS?
  - Host CPU context: Stored when launching VM, restored on VM exit
  - Guest CPU context: Stored on VM exit, restored when VM is run
  - Guest entry/execution/exit control area: KVM can configure guest memory and CPU context, which instructions and events should cause VM to exit
  - Exit information: Exit reason and any other exit-related information
- VMCS information (e.g., exit reason) exchanged with QEMU via kvm_run structure
  - VMCS only accessible to KVM in kernel mode, not to QEMU userspace

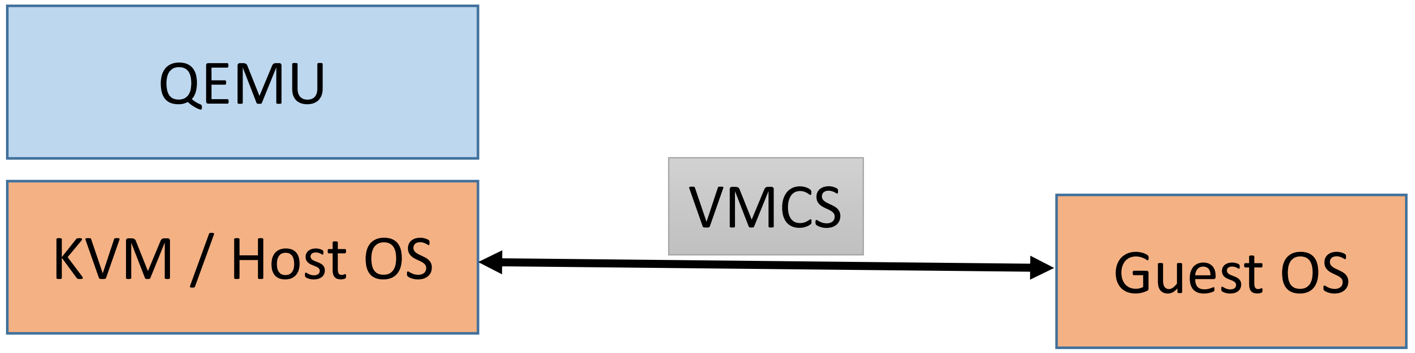
VMX mode execution

- How is guest OS execution in VMX mode different?
- Restrictions on guest OS execution, configurable exits to KVM
  - Guest OS exits to KVM on certain instructions (e.g., I/O device access)
- No hardware access to guest, emulated by KVM
  - Guest OS usually exits on interrupts (interrupts handled by KVM, assigned to the appropriate host or guest OS)
  - KVM can inject virtual interrupts to guest OS during VMX mode entry
- All of the above controlled by KVM via VMCS
- Mimics the trap-and-emulate architecture with hardware support
  - Guest runs in a (special) ring 0, but trap-and-emulate achieved

QEMU/KVM operation revisited

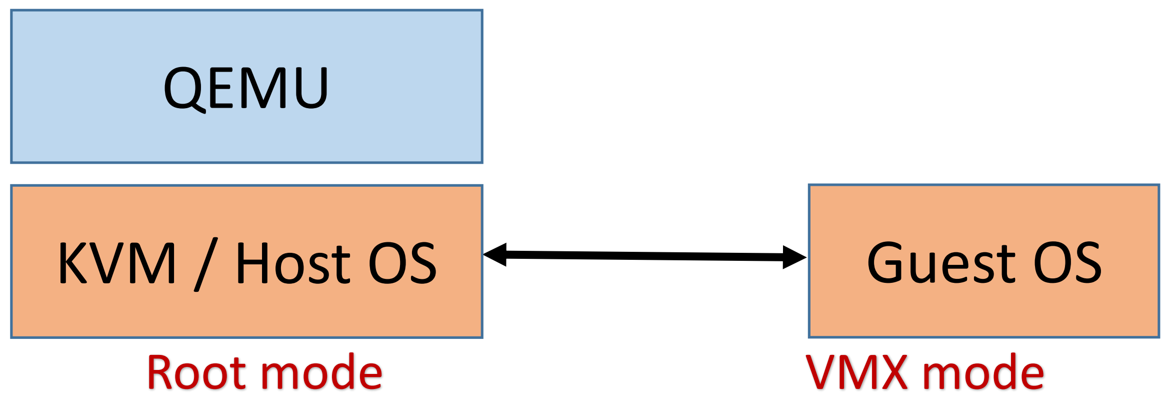

Root mode
VMX mode
Guest VM physical memory
VCPU-0
kvm_run
QEMU (Userspace process)
Guest application
ring 3
ring 0
1. QEMU thread calls KVM_RUN
2. KVM executes VMRESUME/VMLAUNCH
Host context saved in VMCS, guest context restored
3. Guest OS and user applications run normally, within restrictions
4. Guest OS executes VMEXIT upon trigger
Guest context saved, host restored from VMCS
5. KVM handles exit or returns to QEMU thread, with exit info in kvm_run
KVM (kernel module)
VMCS
Guest OS

Host view



- Host sees QEMU as regular multithreaded process
  - Process that has memory-mapped memory, talks to KVM device via ioctl calls
  - Multiple QEMU VCPU threads can be scheduled in parallel on multiple cores
- When KVM launches a VM, host OS context is stored in VMCS
  - Host OS execution is suspended (all host processes stop)
  - CPU loads guest OS context and guest OS starts running
- When guest OS exits, host OS context is restored from VMCS
  - Host OS resumes in KVM, where it stopped execution
  - KVM can return to QEMU, or host can switch to another process
  - Host OS is not aware of guest OS execution

Summary



- Hardware-assisted CPU virtualization in QEMU/KVM
  - QEMU creates guest physical memory, one thread per VPCU
  - QEMU VCPU thread gives KVM_RUN command to KVM kernel module
  - KVM configures VM information in VMCS, launches guest OS in VMX mode
  - Guest OS runs natively on CPU until VM exit happens
  - Control returns to KVM/Host OS on VM exit
  - VM exits handled by KVM or QEMU
  - Host schedules QEMU like any other process, not aware of guest OS