KVM

2012-05

OSS Technology Section II
OSS Platform Technology Center
Business Strategy Group
SCSK

Linux

- a UNIX-like kernel
  - Process,Thread
  - Signal,TTY
  - Pipe
  - BSD Socket,TCP/IP
  - Filesystem
  - ...

qemu

- “FAST! processor emulator” by Fabrice Bellard
- An ordinary process from the host OS's POV
- Dynamic translator
- Emulate many of misc peripherals
  - PCI, ISA, ...
  - IDE, NIC, ...
  - Keyboard, Mouse
  - Video
  - ...

x86

- Intel i386 and compatible processors
- AMD introduced 64-bit mode
  - “amd64”,”x86-64”,“long mode”
  - Intel followed; “IA32e”,”Intel64”
- Virtualization unfriendly
  - CPUID
- Recent virtualization support featues
  - Intel VMX
  - AMD SVM

KVM

- Modified version of qemu (“qemu-kvm”), accelerated by “kvm” kernel module
- “kvm” kernel module requires hardware virtualization features provided processors
  - eg. Intel's virtual-machine extension “VMX”

KVM

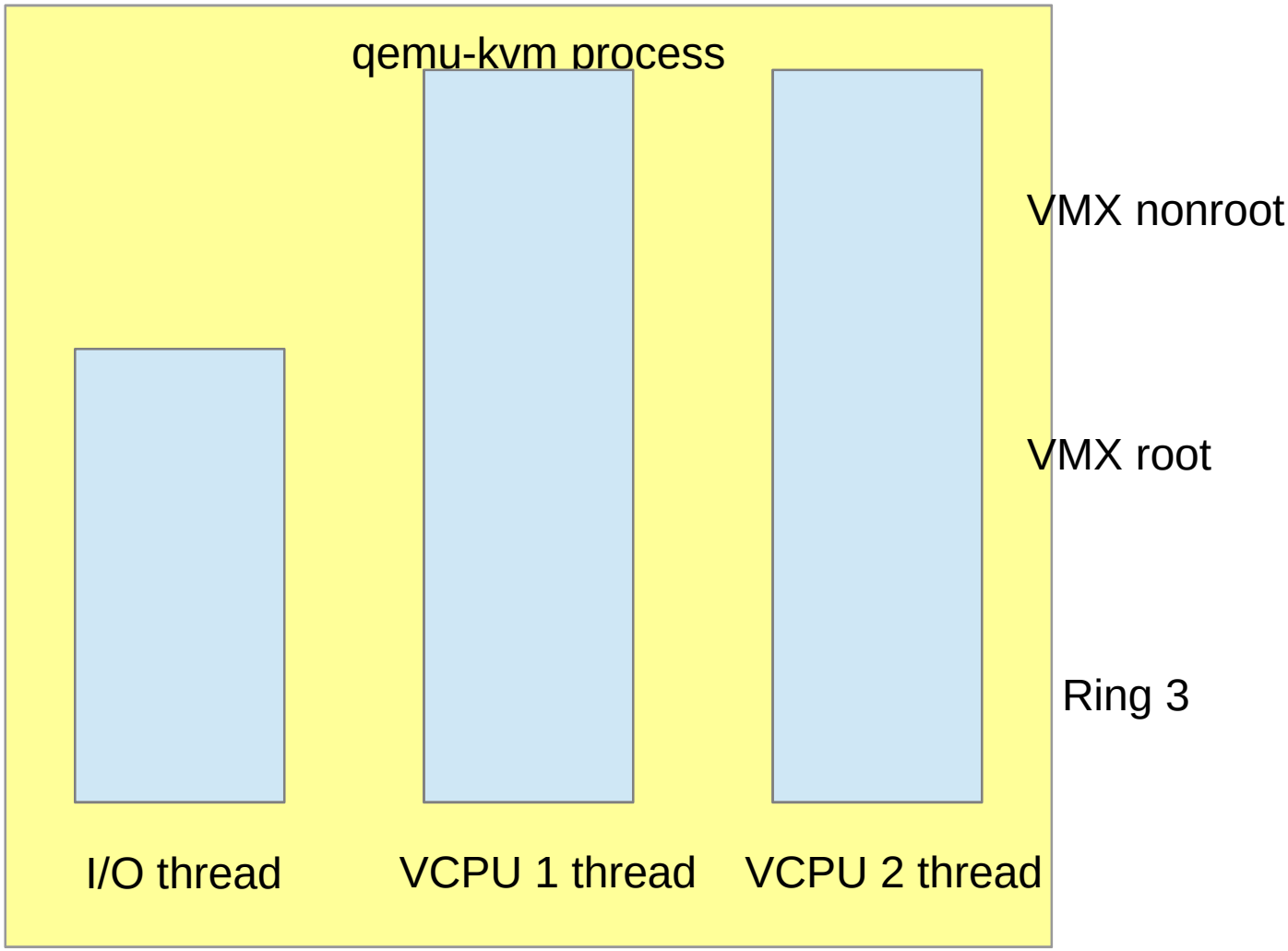
- qemu options

  --enable-kvm

  --no-kvm

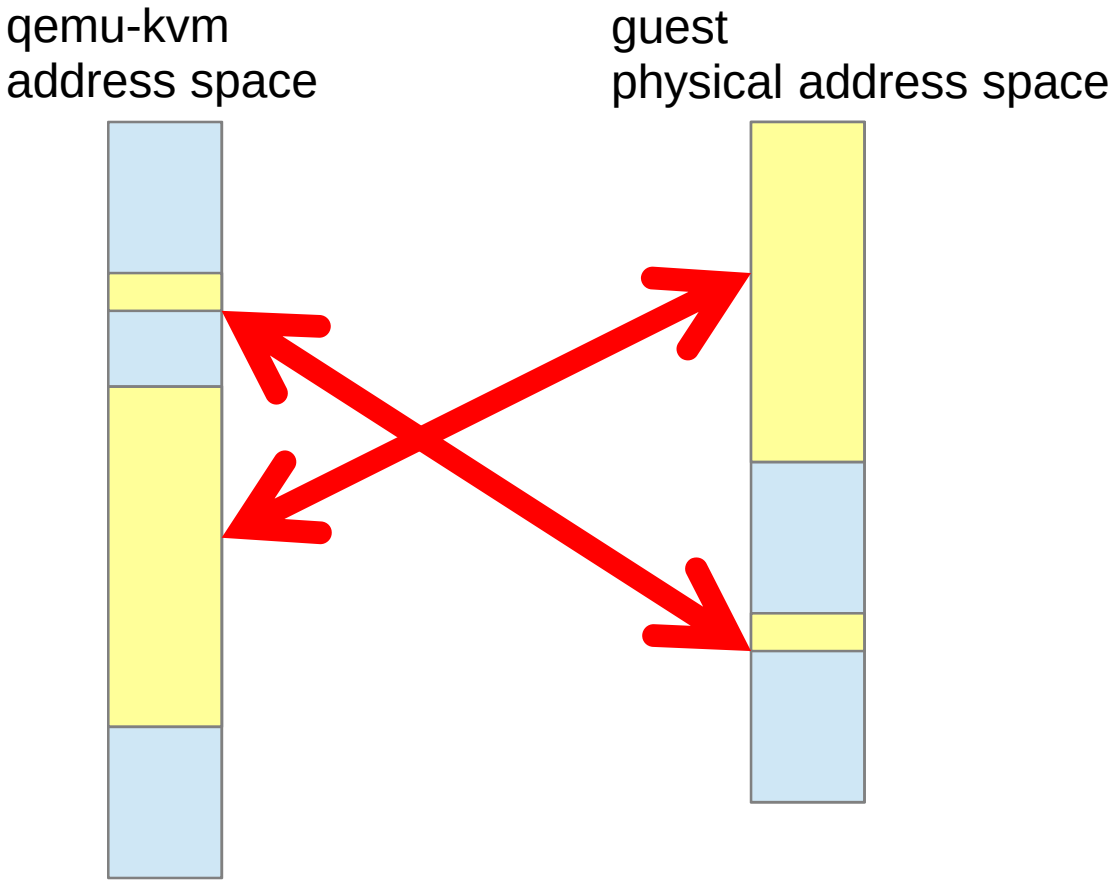
qemu-kvm VCPU model

- Spawns threads for each VCPUs



qemu-kvm memory model

- Use some parts of qemu-kvm process' virtual memory as its guest's physical memory



VMX

- Extensions for VMM implementations
- Special instructions
  - VMXON, VMLAUNCH, VMREAD, ...
- Virtual-machine control data structures (“VMCS”)
- VMX non-root operation
  - “Guest mode”
- VMX root operation
  - “Host mode”

VMCS

- Per logical processer (VCPU) structure
- Maintain VCPU state
  - Guest-State
    - Guest registers
    - non-register state (eg. “Blocking by STI”)
  - Host-State
    - Host processor state used for VM Exit
  - VM-Exit Information
    - Why VM exit happened?
      - Interrupt, Page fault, ...
  - etc

VM Enter, VM Exit

- VM Enter; Transition from “VMX root” to “VMX non-root”
- VM Exit; Transition from “VMX non-root” to “VMX root”
- Expensive and should be avoided for better performance

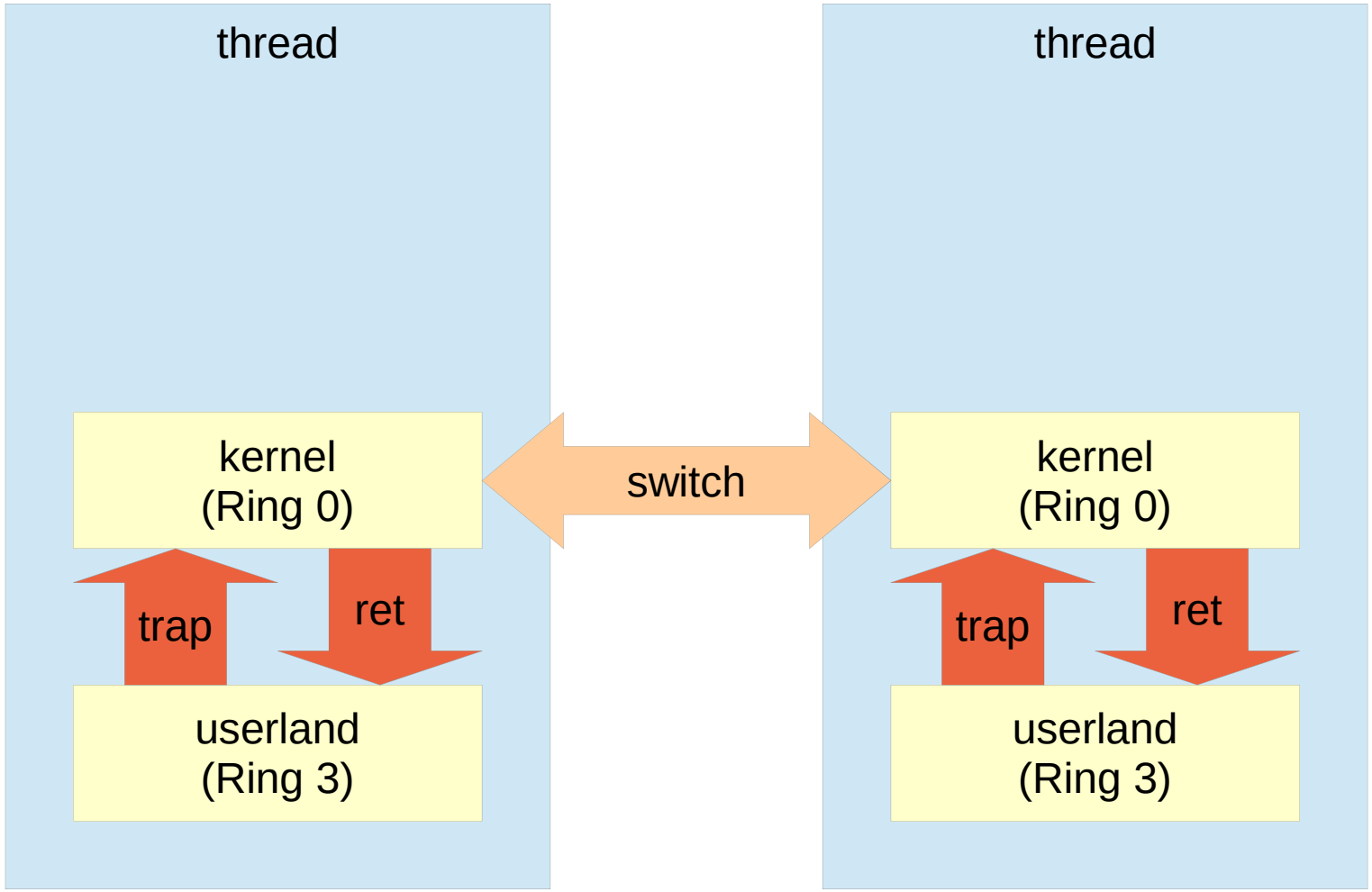
VM Exit reasons

- Interrupts
- Page faults
- Instructions
  - I/O
    - inb, outb, ...
  - HLT
  - CPUID
  - ...
- ...

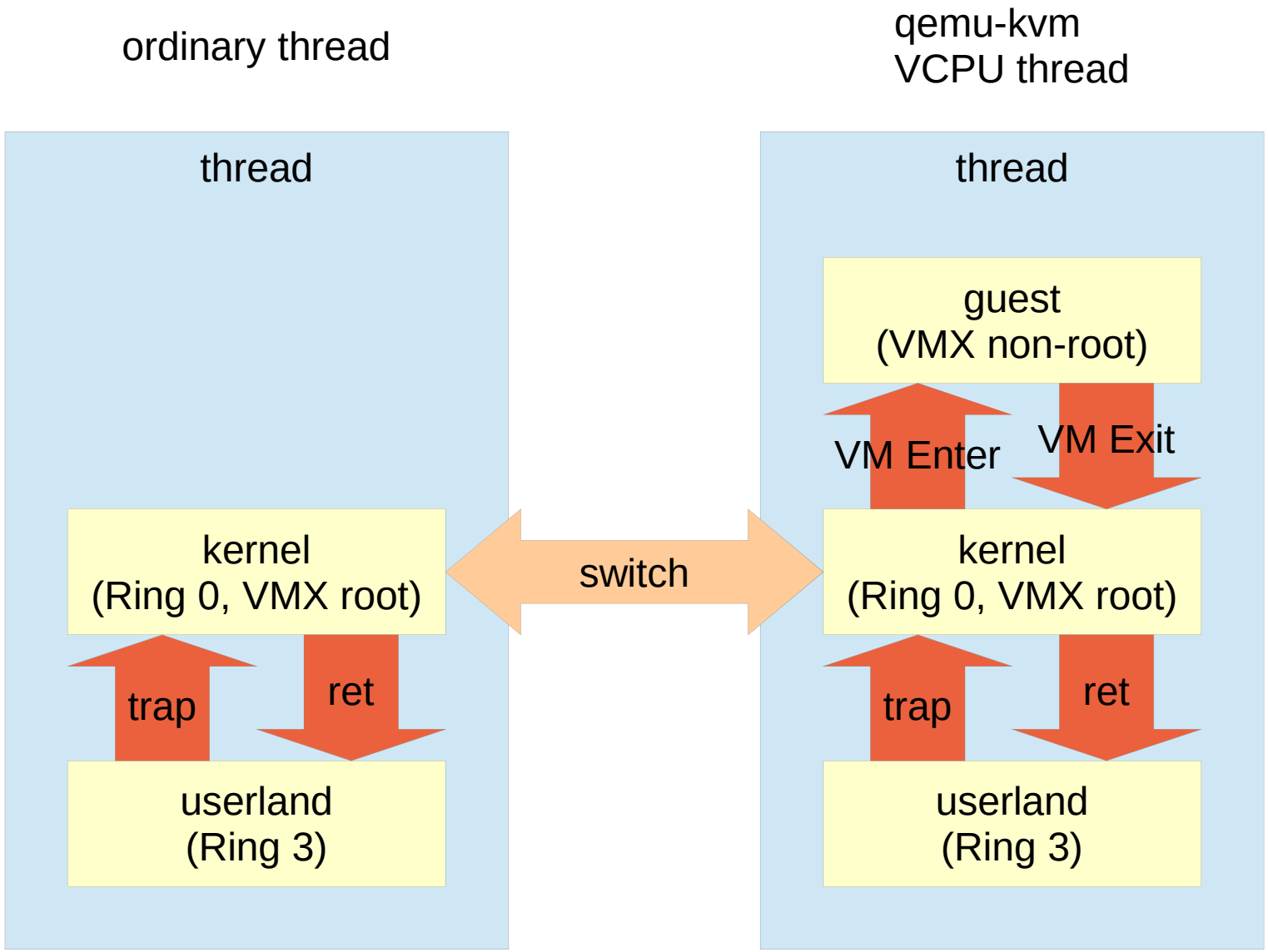
Who emulates what?

- Depends on versions and configurations, but...
- VMX emulates performance critical stuffs
  - Most of CPU instructions
    - Including the infamous “CPUID”
- kvm kernel module emulates some of the rest
  - PTE walker (“shadow paging”)
  - “HLT” instruction
  - APIC
- qemu-kvm (userland) handles the rest
  - Many of devices, including disks

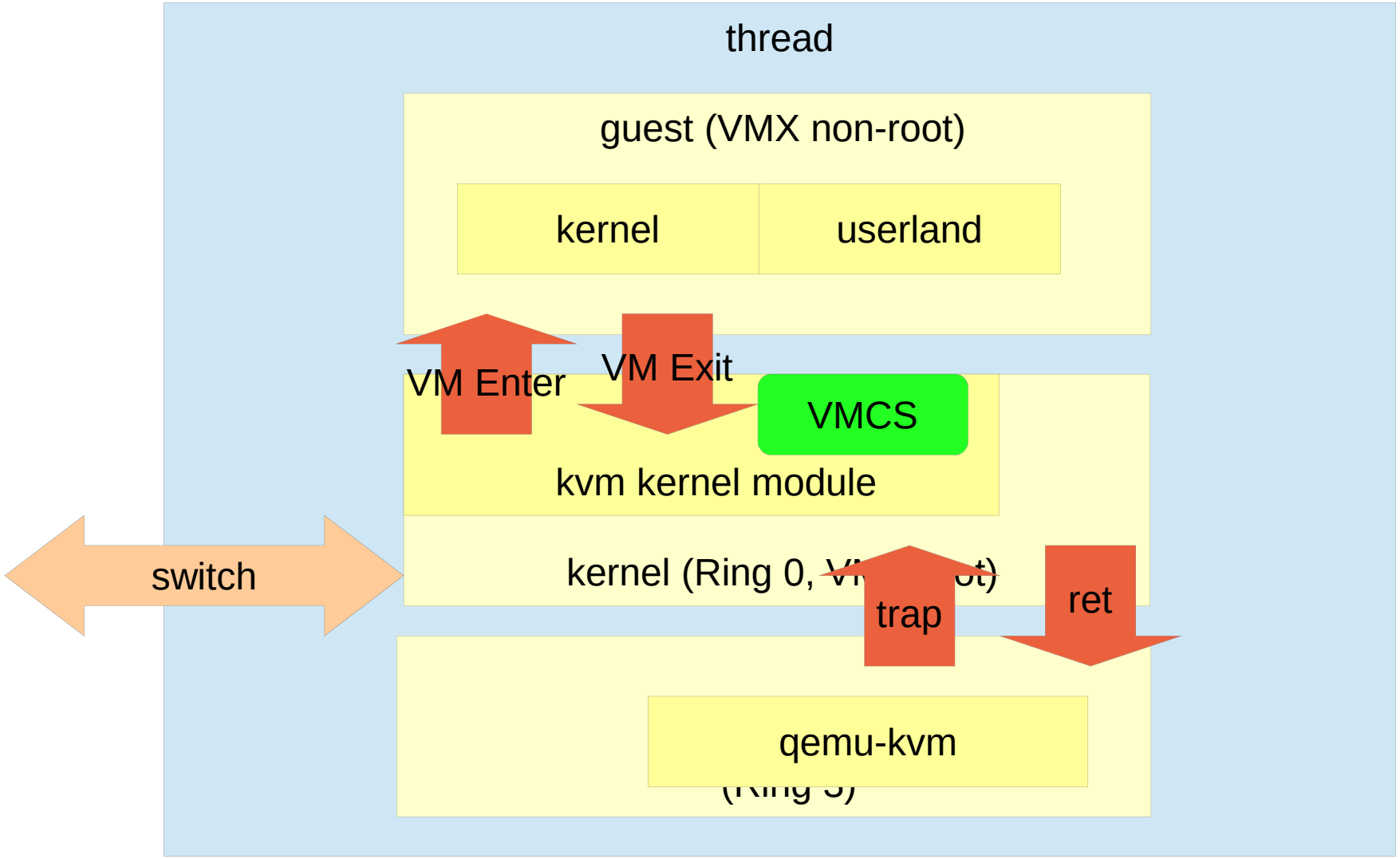
Ordinary threads



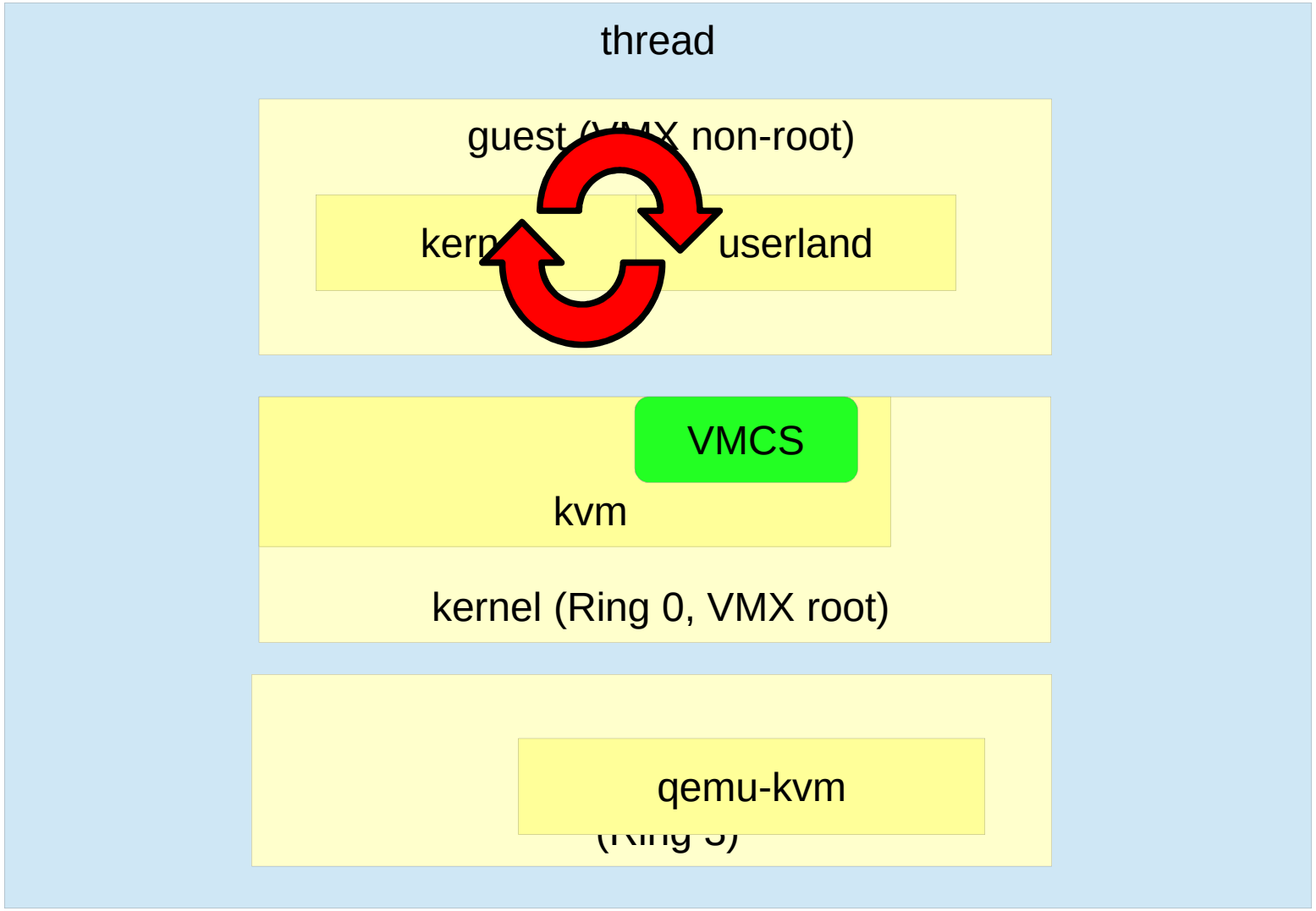
qemu-kvm VCPU thread



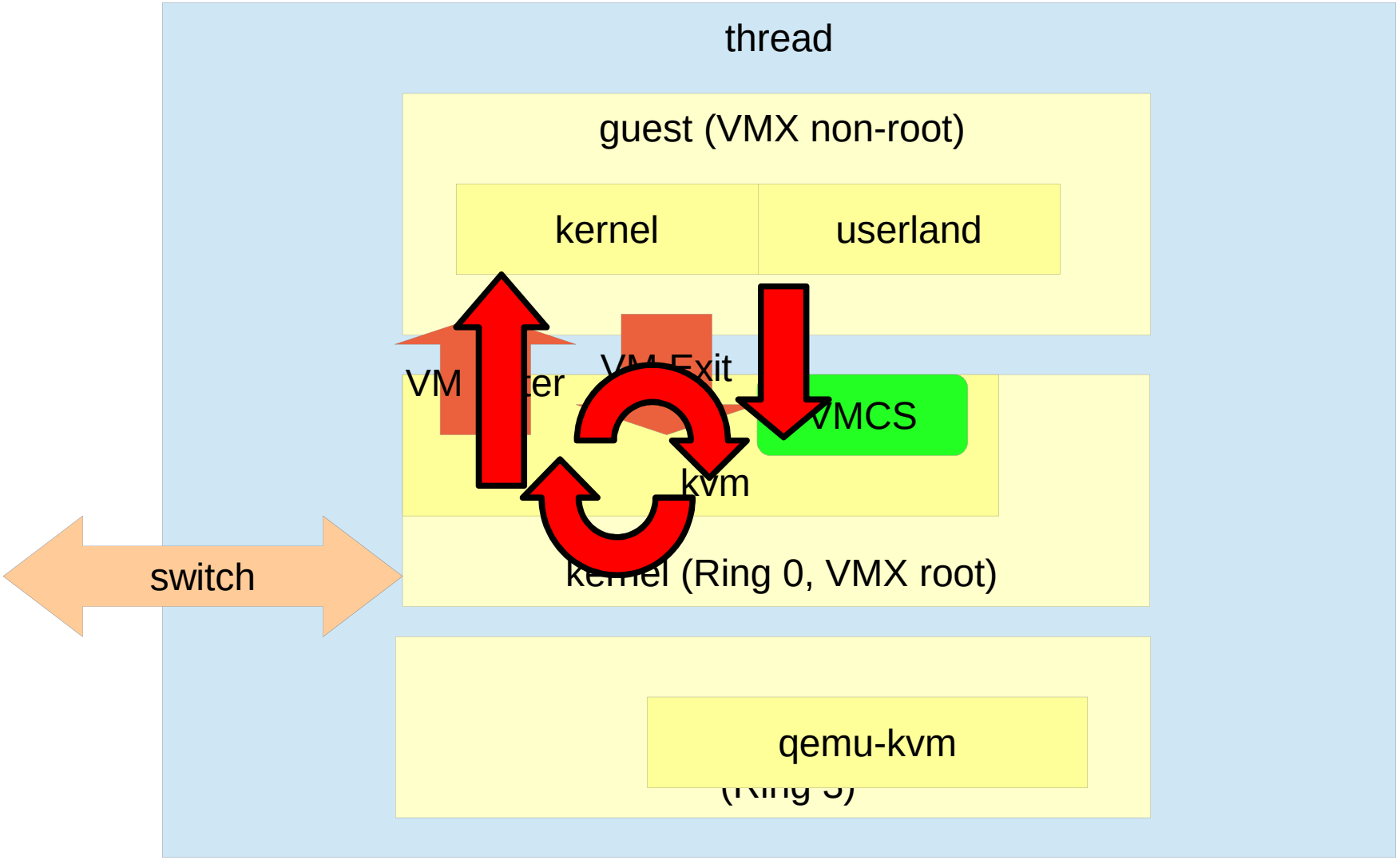
qemu-kvm VCPU thread



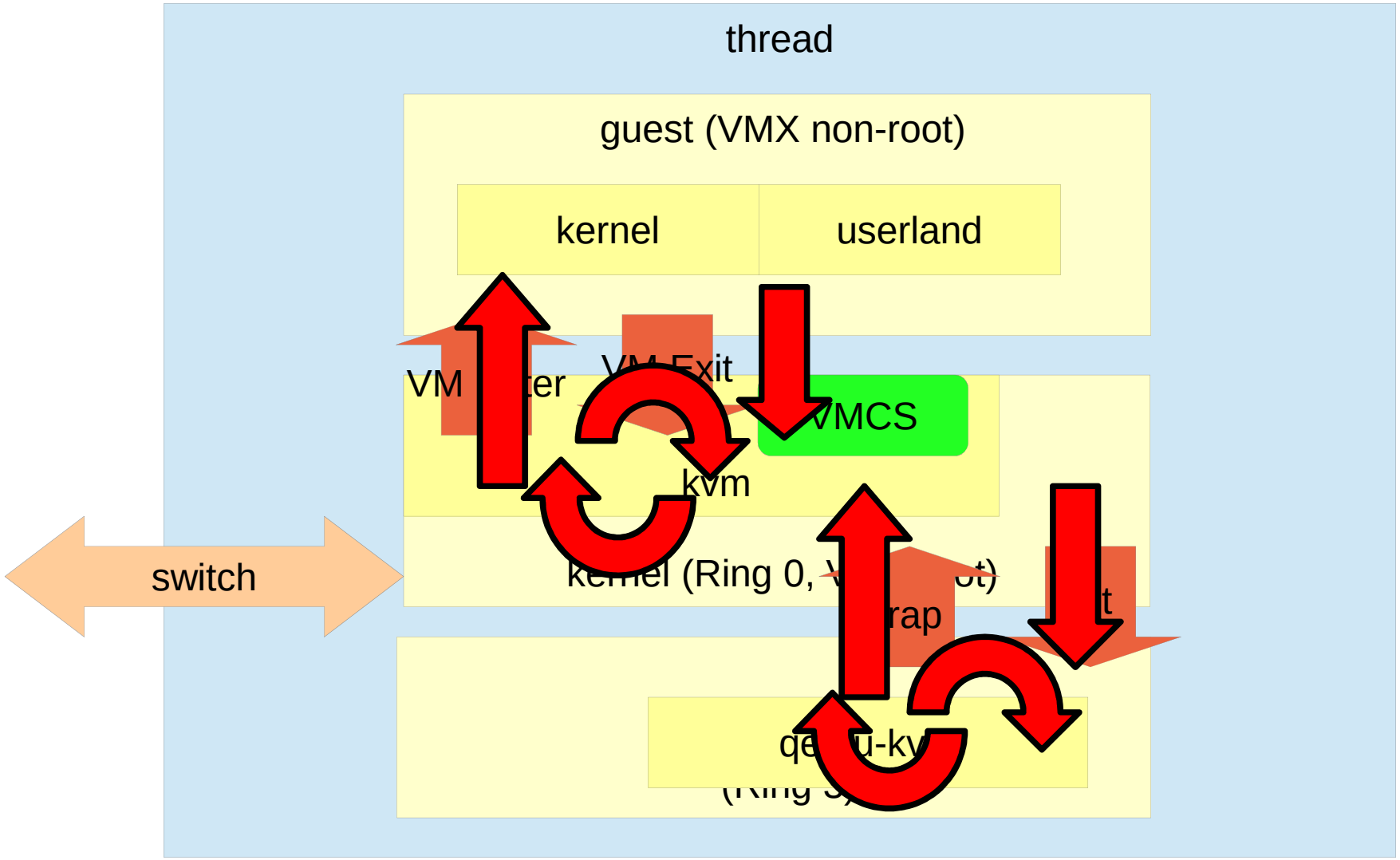
VMX emulated stuff



kvm emulated stuff



qemu emulated stuff

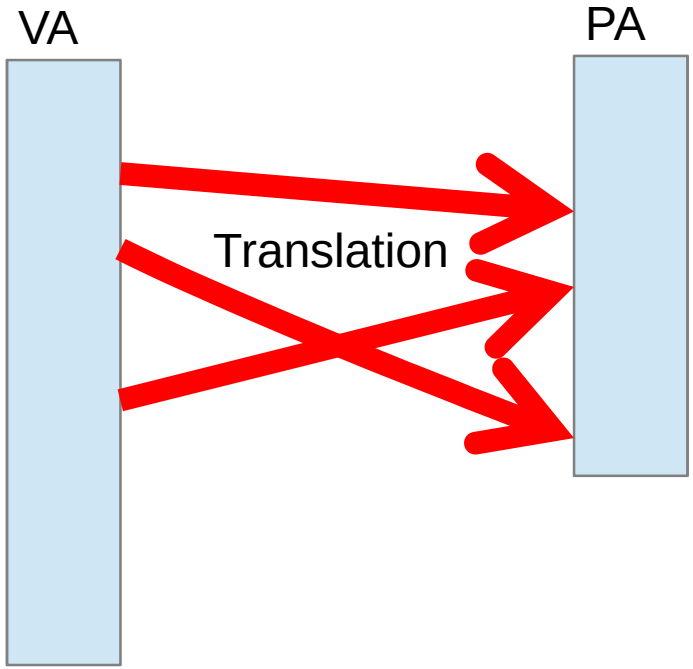


Features of recent processors

- EPT (Extended Page Table)
  - Nested paging
  - PTE walk w/o VM Exit
- VPID (Virtual Processor Identifier)
  - 16-bit tag for TLBs and caches
- VT-d
- PAUSE-Loop Exit
  - Detect busy loop in guest
- TPR shadow

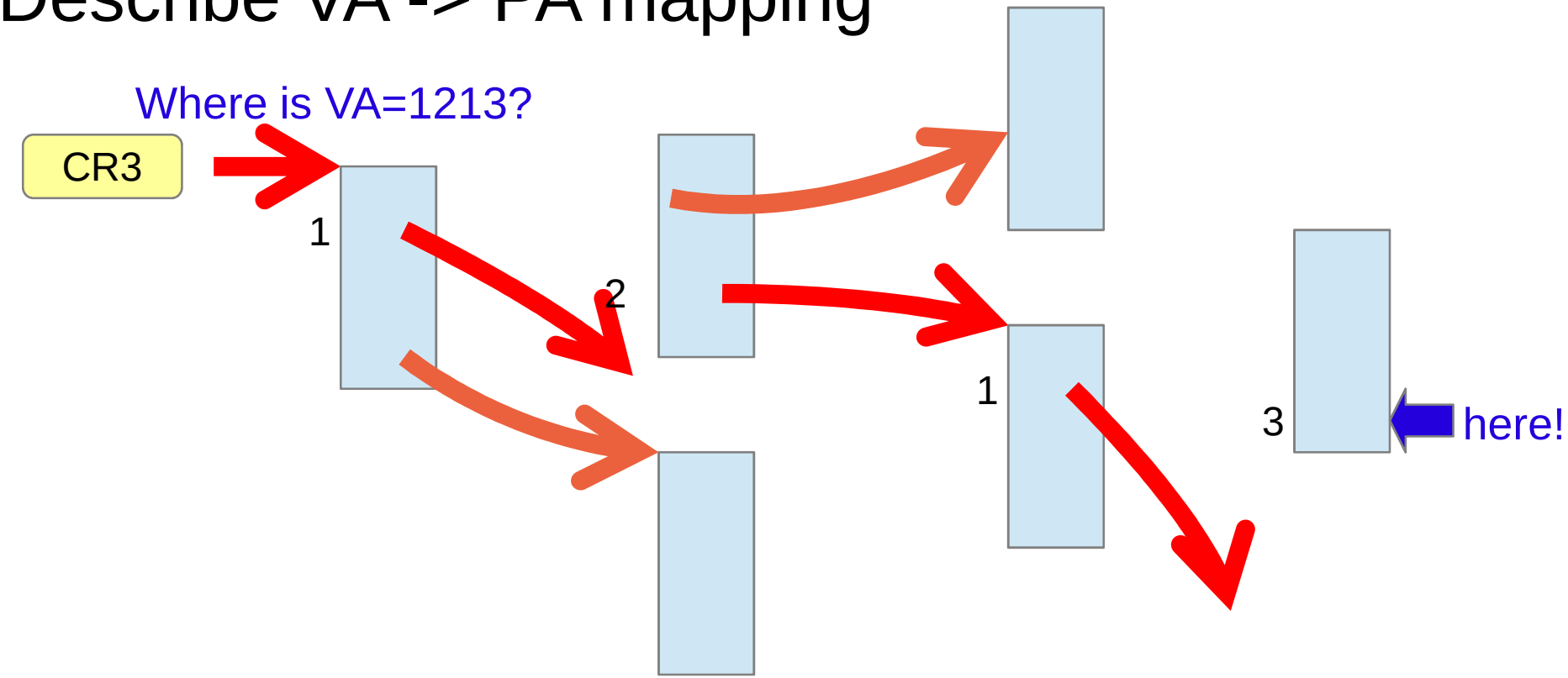
Address translation

- Software uses virtual address (VA)
- Physical memory is located by by physical address (PA)
- The translation is often cached (TLB)



x86 page table

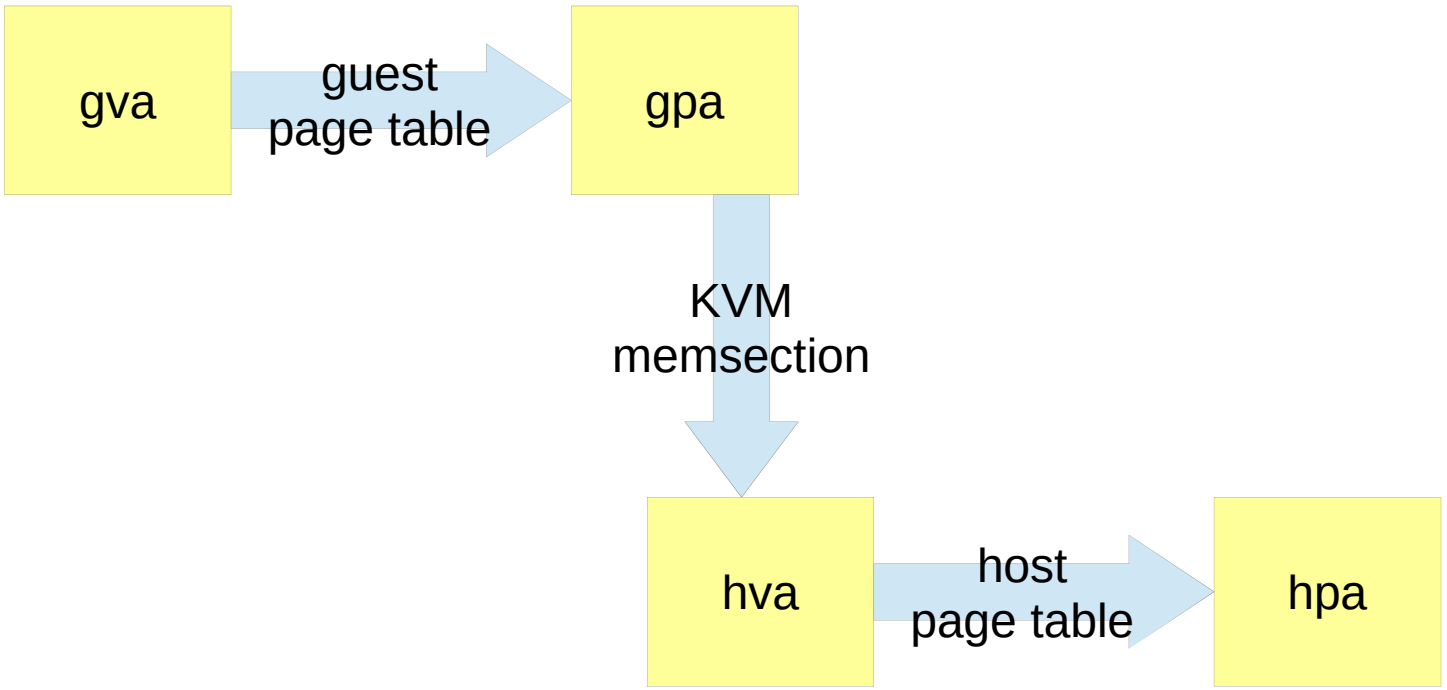
- Processor-defined in-core structure
- Radix tree
- Describe VA -> PA mapping



KVM address spaces

- 4 different address spaces
  - guest virtual address (gva)
    - used by guest software
  - guest physical address (gpa)
  - host virtual address (hva)
    - qemu-kvm process' address space
  - host physical address (hpa)

Address translation



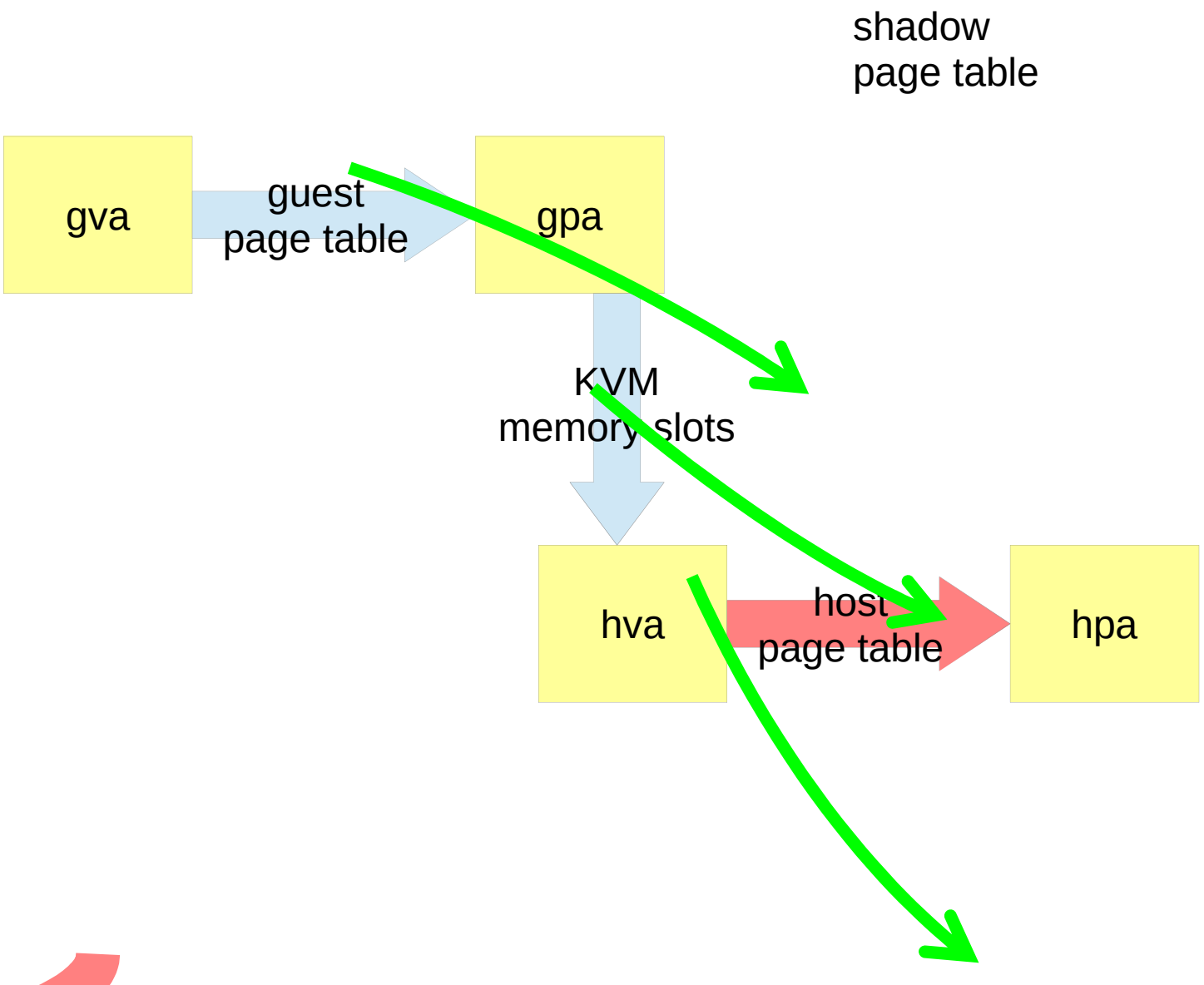
Address translation

- gva -> gpa
  - Need to walk page table in guest
  - Complicated because guest page table itself uses gpa
- gpa -> hva
  - KVM maintains the mapping (memory slots)
- hva -> hpa
  - Same as normal processes

Shadow page table

- Software technique to emulate guest page table
- Host software walks guest page table and build the corresponding “shadow” page table
- CPU actually walks the “shadow” one
- Complicated

Address translation (shadow)

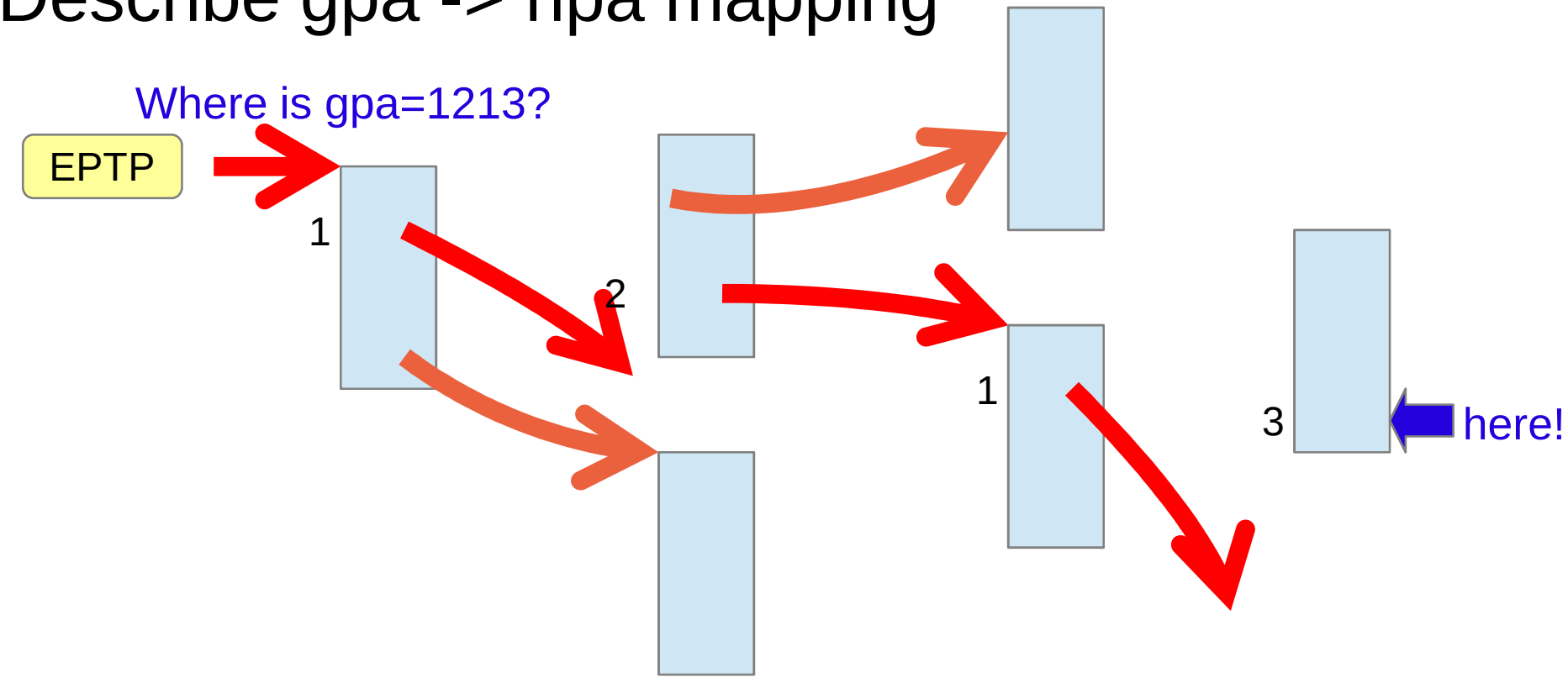


EPT

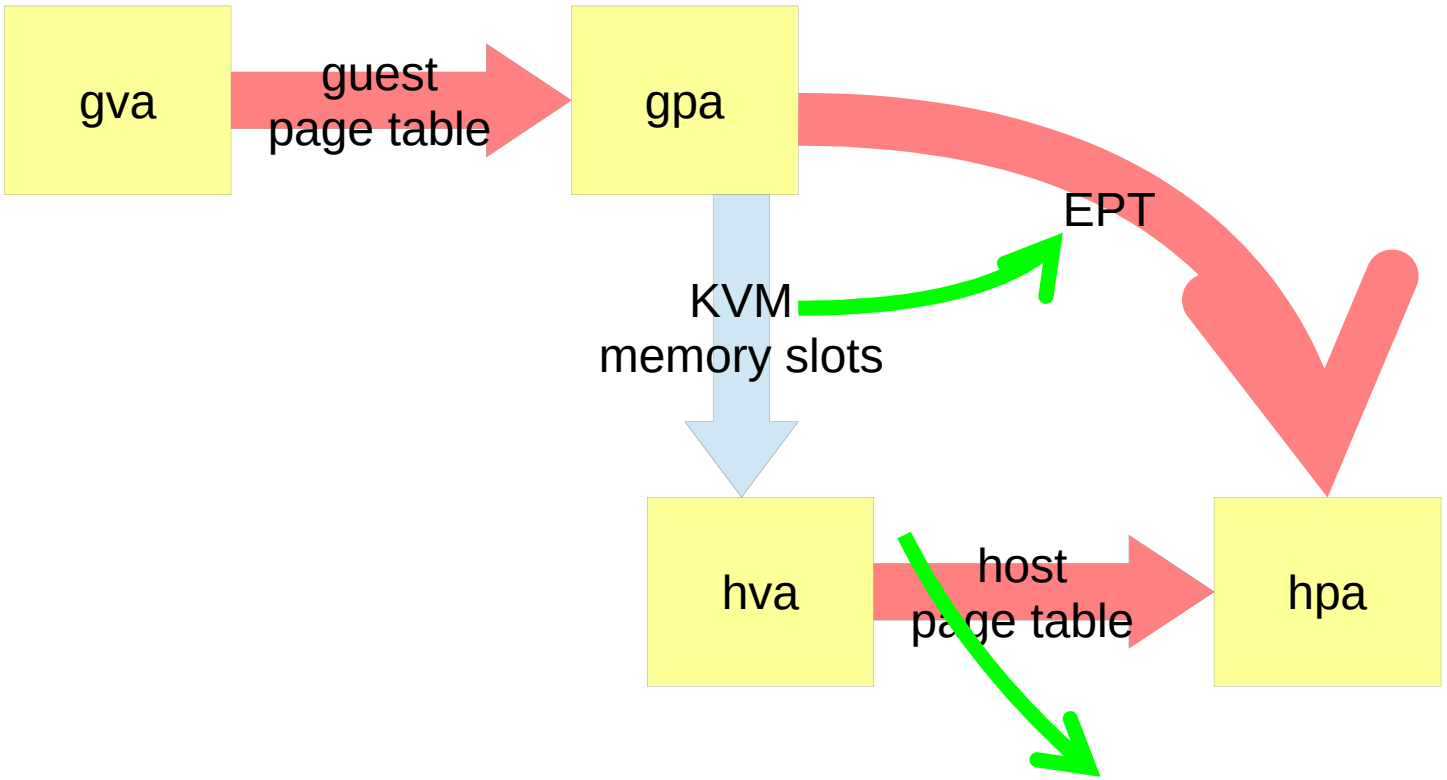
- gpa -> hpa translation table
  - In-core tree-ish structure similar to page table
- CPU automatically traverses guest page table and EPT w/o software intervention
- Top level pointer (EPTP) is stored in VMCS
- A new instruction to invalidate translation
  - INVEPT

EPT

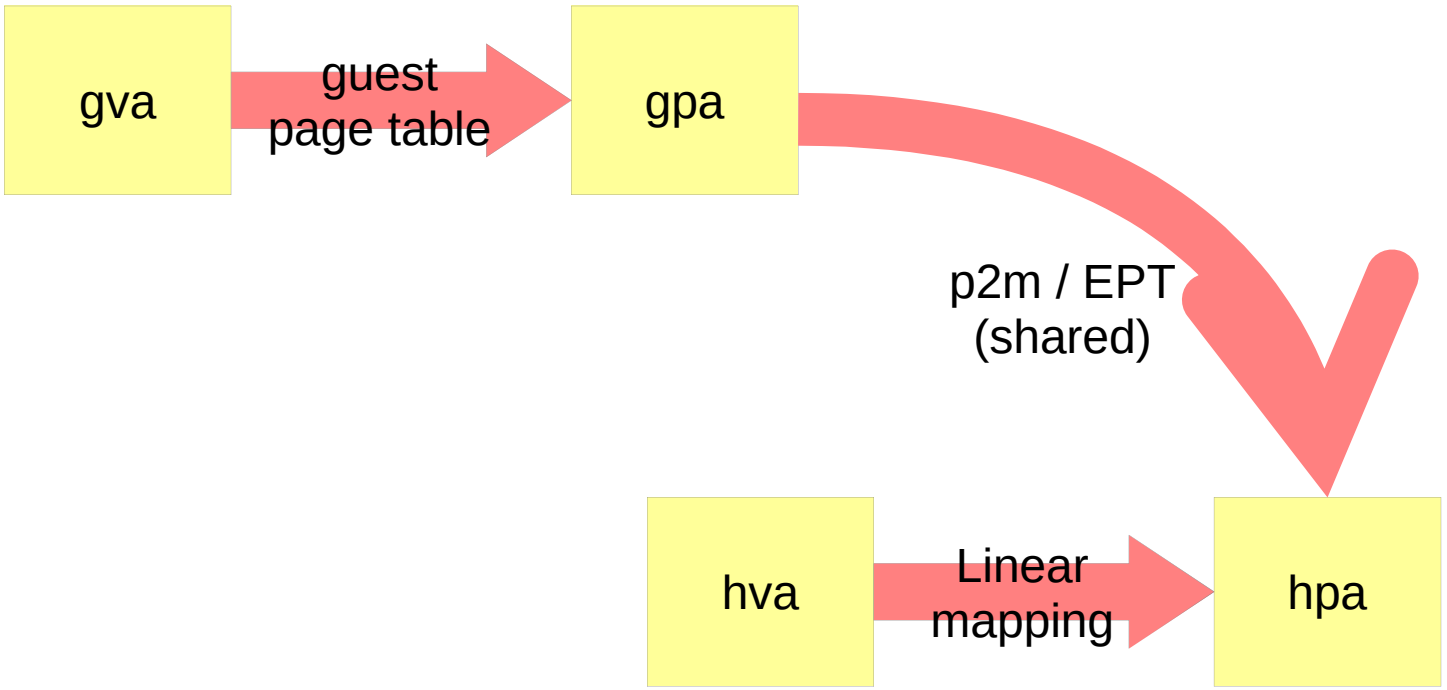
- Processor-defined in-core structure
- Radix tree
- Describe gpa -> hpa mapping



Address translation (EPT)



Address translation (Xen, FV)



Example: COW (native)

- memory write -> fault
- kernel: update page table
- memory write -> OK!

Example: COW (w/o EPT)

- guest: memory write -> fault
- VM Exit
- host: inspect guest page table and inject page fault
- VM Enter
- guest kernel: update page table
- guest: memory write -> fault
- VM Exit
- host: inspect guest page table and update shadow
- VM Enter
- guest: memory write -> OK!

Example: COW (w/ EPT)

- guest: memory write -> fault
- guest kernel: update page table
- guest: memory write -> OK!

Q: how many memory fetches can be necessary for a translation?

- Hint
  - Native
    - CR3
    - 4 level page directories
  - EPT
    - Guest CR3
    - 4 level guest page directories
    - All of the above are gpa-based
      - Need EPT walk for gpa->hpa
      - 4 level EPT directories

EPT switching

- Allows a guest switch EPT
  - Select from listed EPTPs
- What to use?

EPT

- kvm_intel module option

  ept=1

VPID

- Additional 16-bit tag for TLB entries
- Stored in VMCS
- A new instruction to invalidate translations
  - INVVPID

VPID

- kvm_intel module option

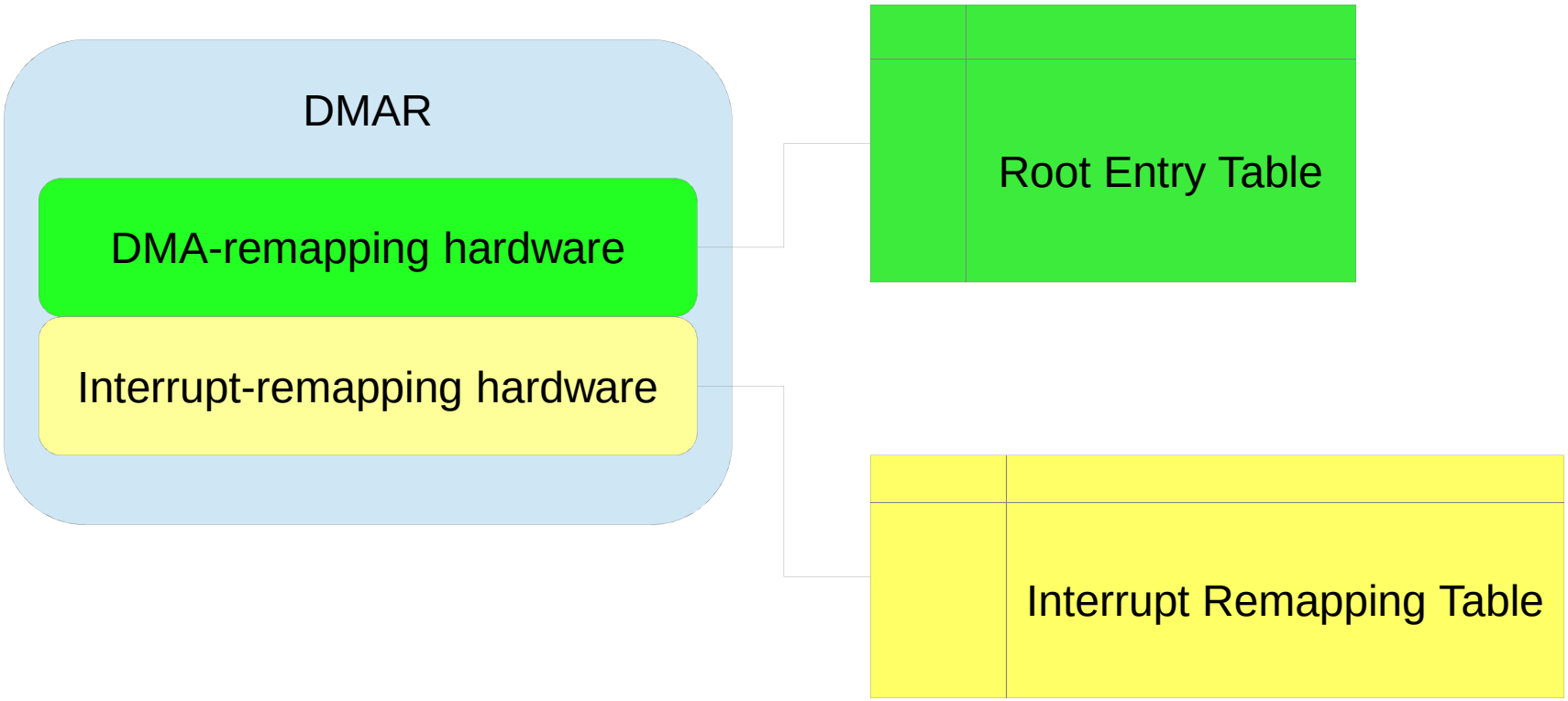
  vpid=1

VT-d

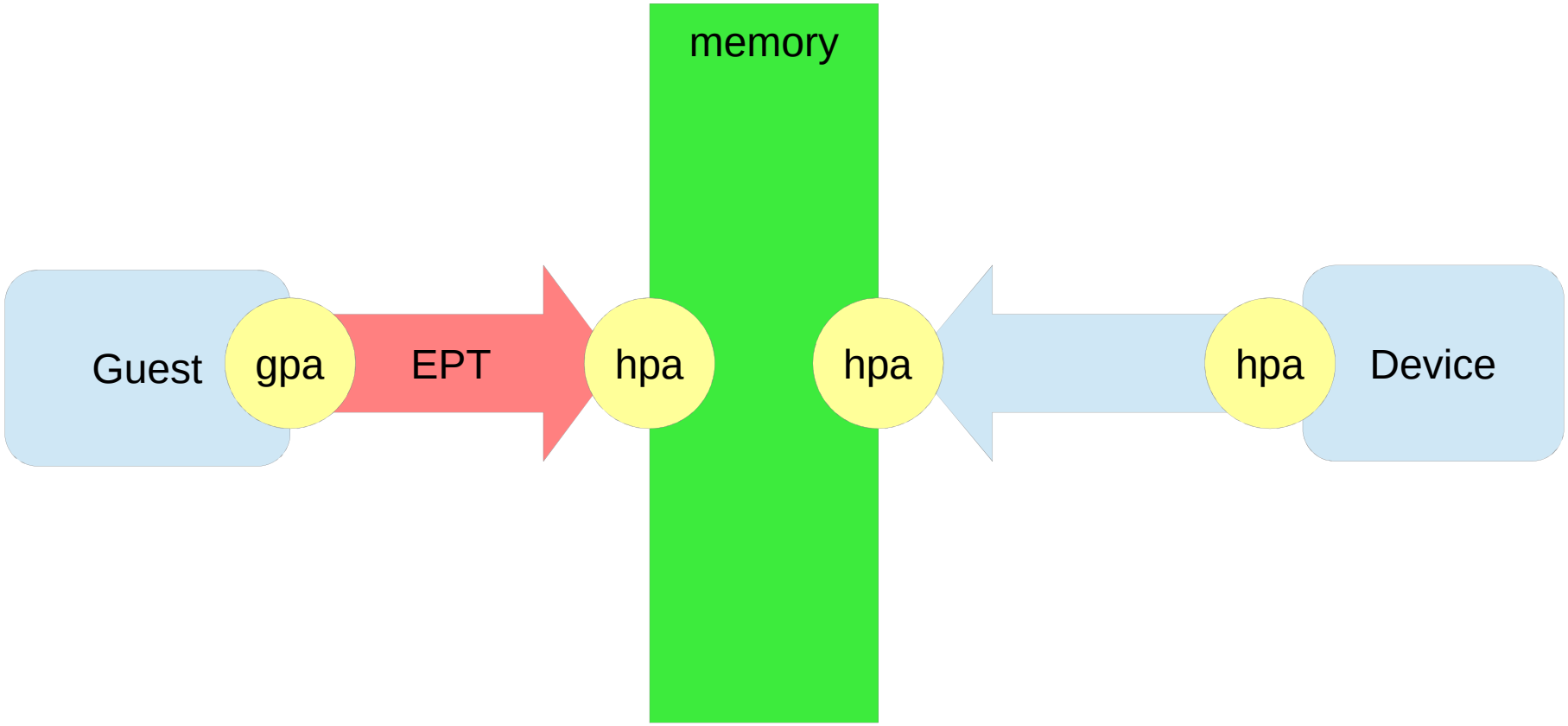
- DMA remapping
- Interrupt remapping
- Allows device pass-through

DMA/Interrupt remap hardware unit

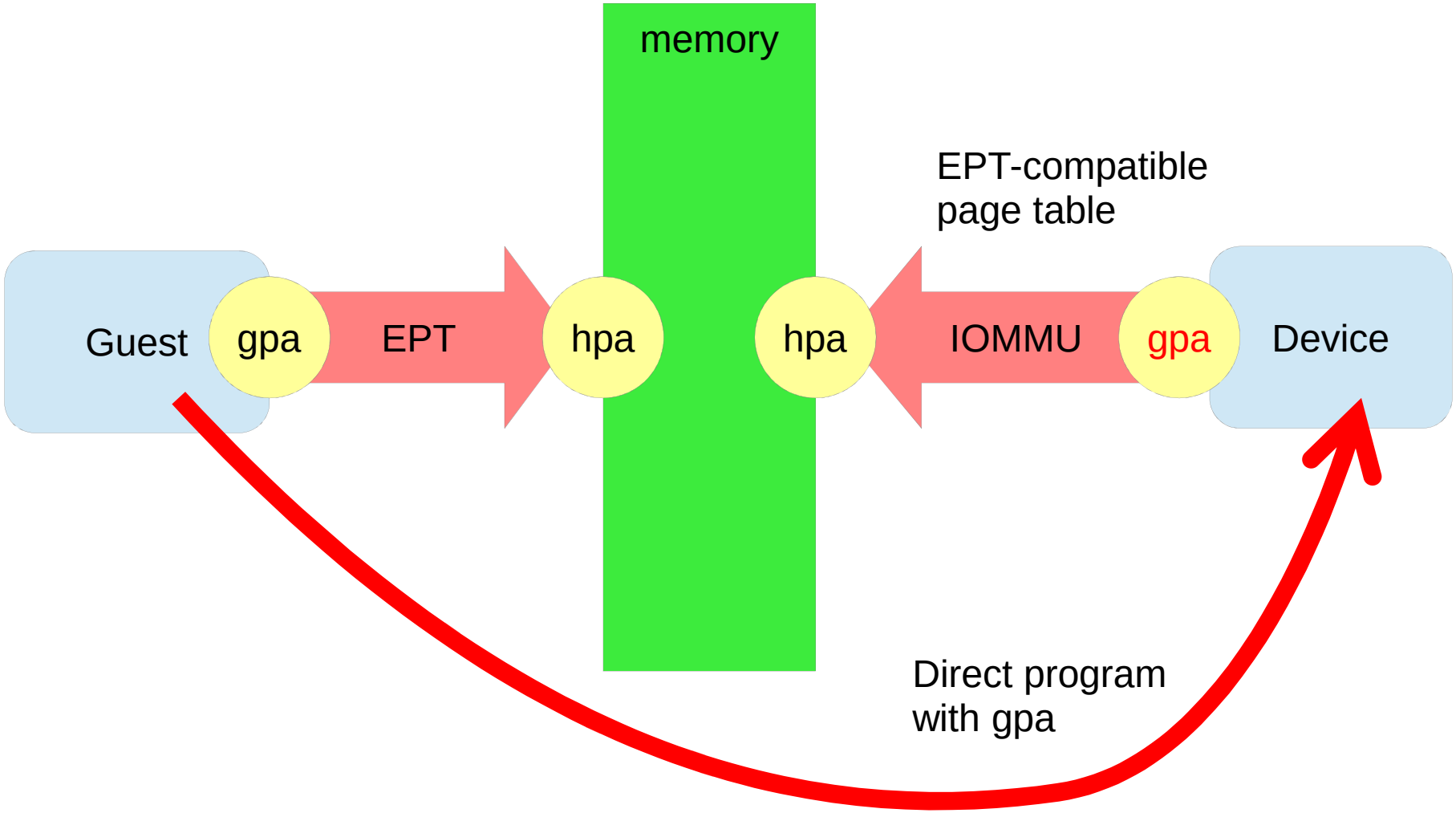
- At least one for a PCI segment
- Described by ACPI “DMAR”



w/o DMA remapping



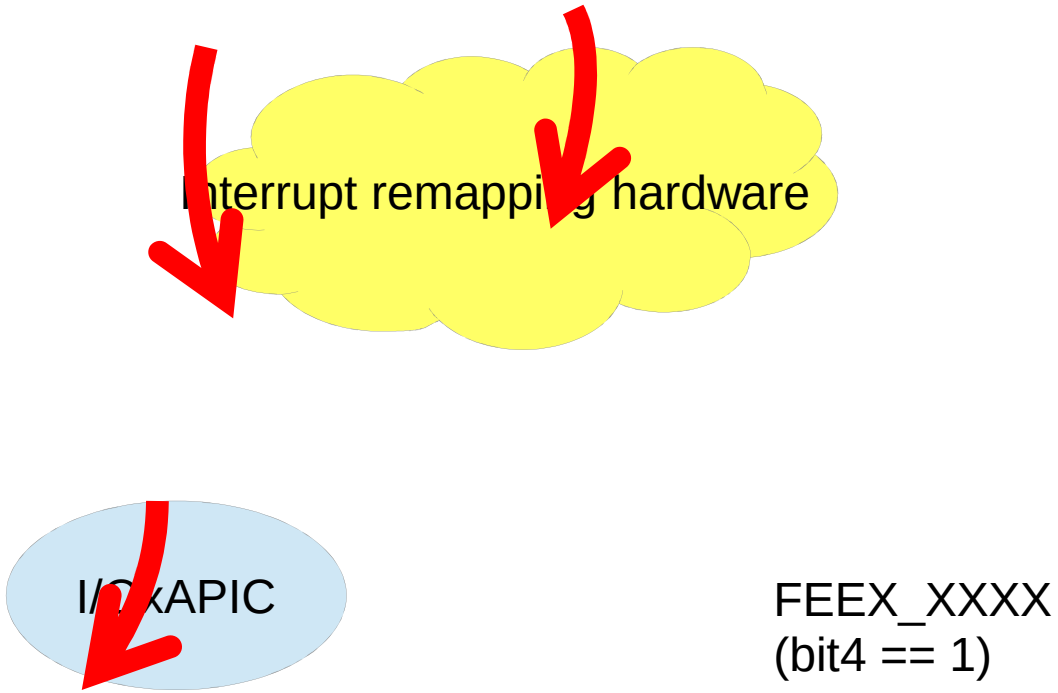
w/ DMA remapping



DMA remapping (IOMMU)

- Bus/Device/Function -> Address space
  - 2 level tree
    - Root-entry table
      - Indexed by Bus#
    - Context-entry table
      - Indexed by Device# and Function#
      - Contains
        - Domain ID
        - Address space root
- DMA Virtual Address (dva) -> hpa
  - EPT-like multi-level page table

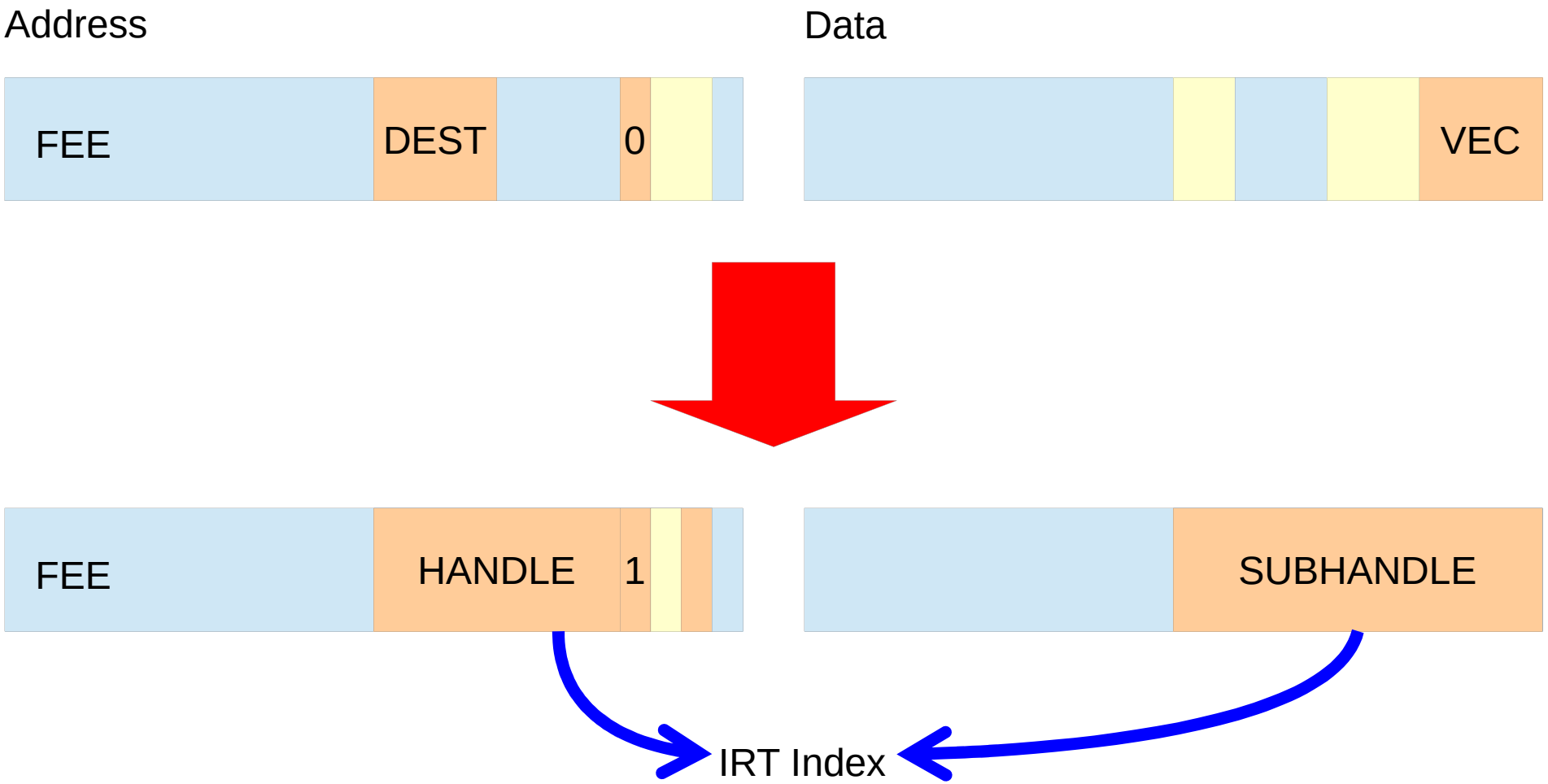
Interrupt remapping



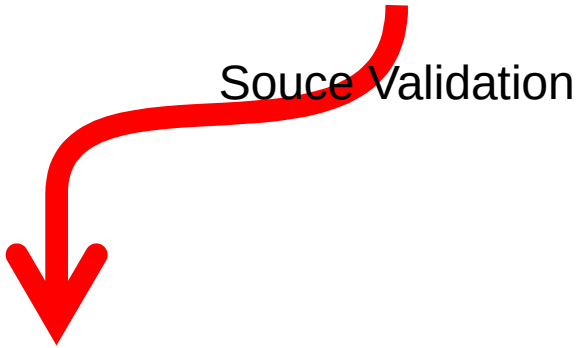
Interrupt remapping

- New interrupt request format
  - Compatibility format (OLD)
    - Address contains Destination ID
    - Data contains Vector
  - Remappable format (NEW)
    - Address contains HANDLE
    - Data contains SUBHANDLE
- Interrupt Remapping Table (IRT)
  - Indexed by HANDLE+SUBHANDLE
  - Entry (IRTE) contains Destination ID, Vector, ...

Interrupt request format



IRTE



VT-d

- kernel boot parameters

  iommu=

  intel_iommu=

  intremap=

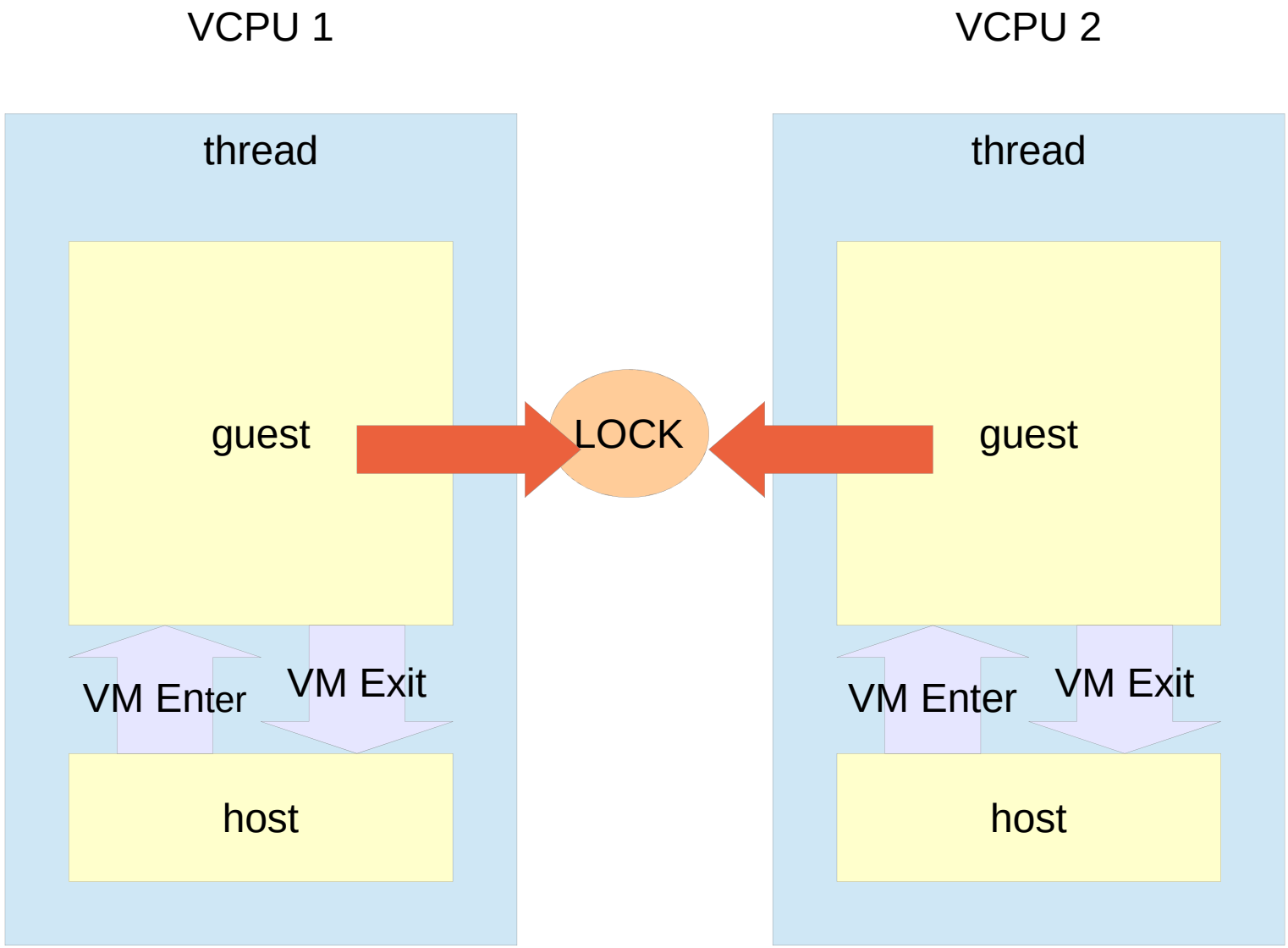
- qemu

  “device assignment”

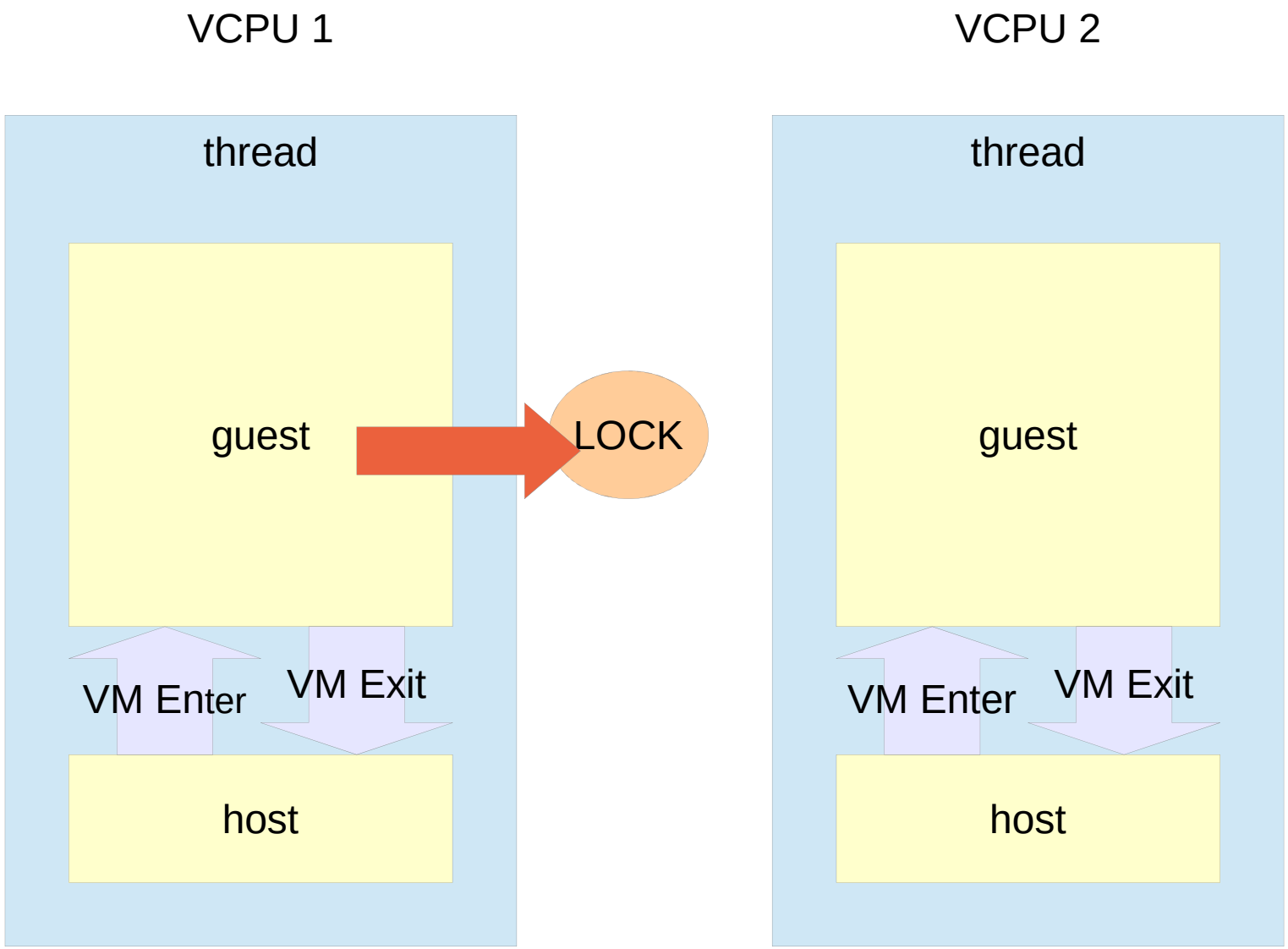
PAUSE Loop-Exit

- PAUSE instruction is used to “yield” processor resources to sibling threads (Hyper Threading, SMT)
- Detect tight loop with PAUSE and causes VM Exit to notify host OS
  - Avoid wasting processor cycles

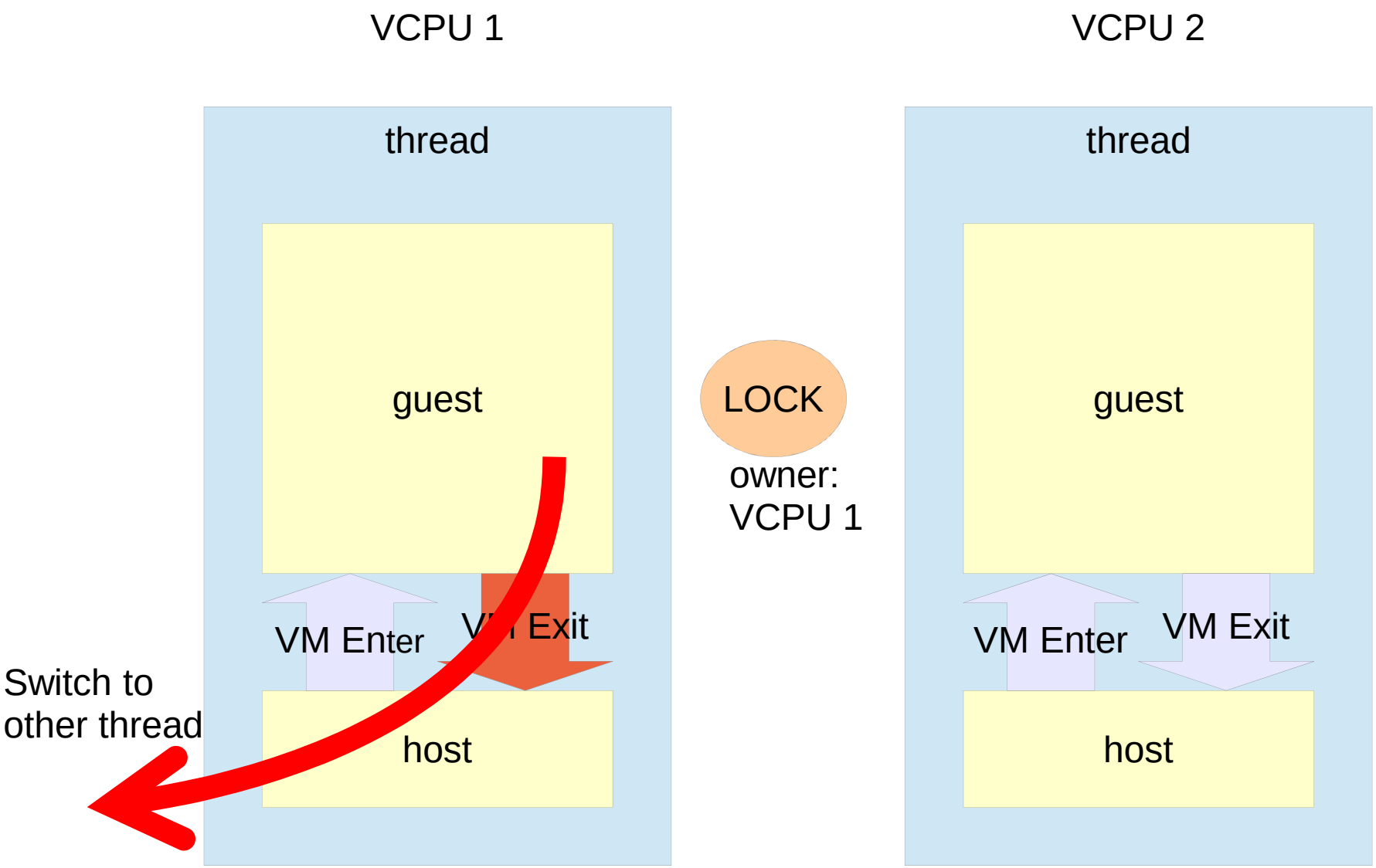
Lock contention in a guest OS



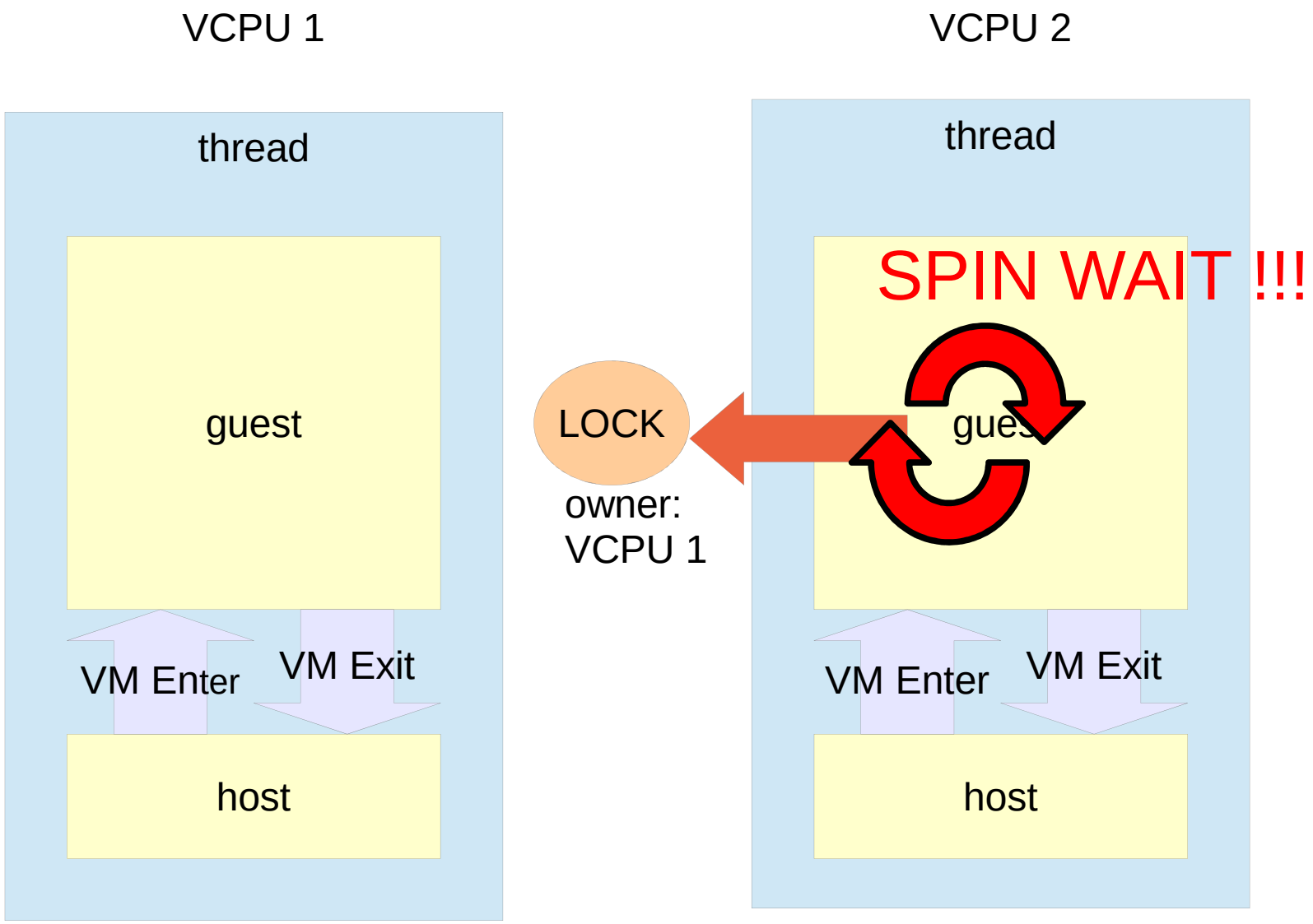
VCPU 1 acquires the lock



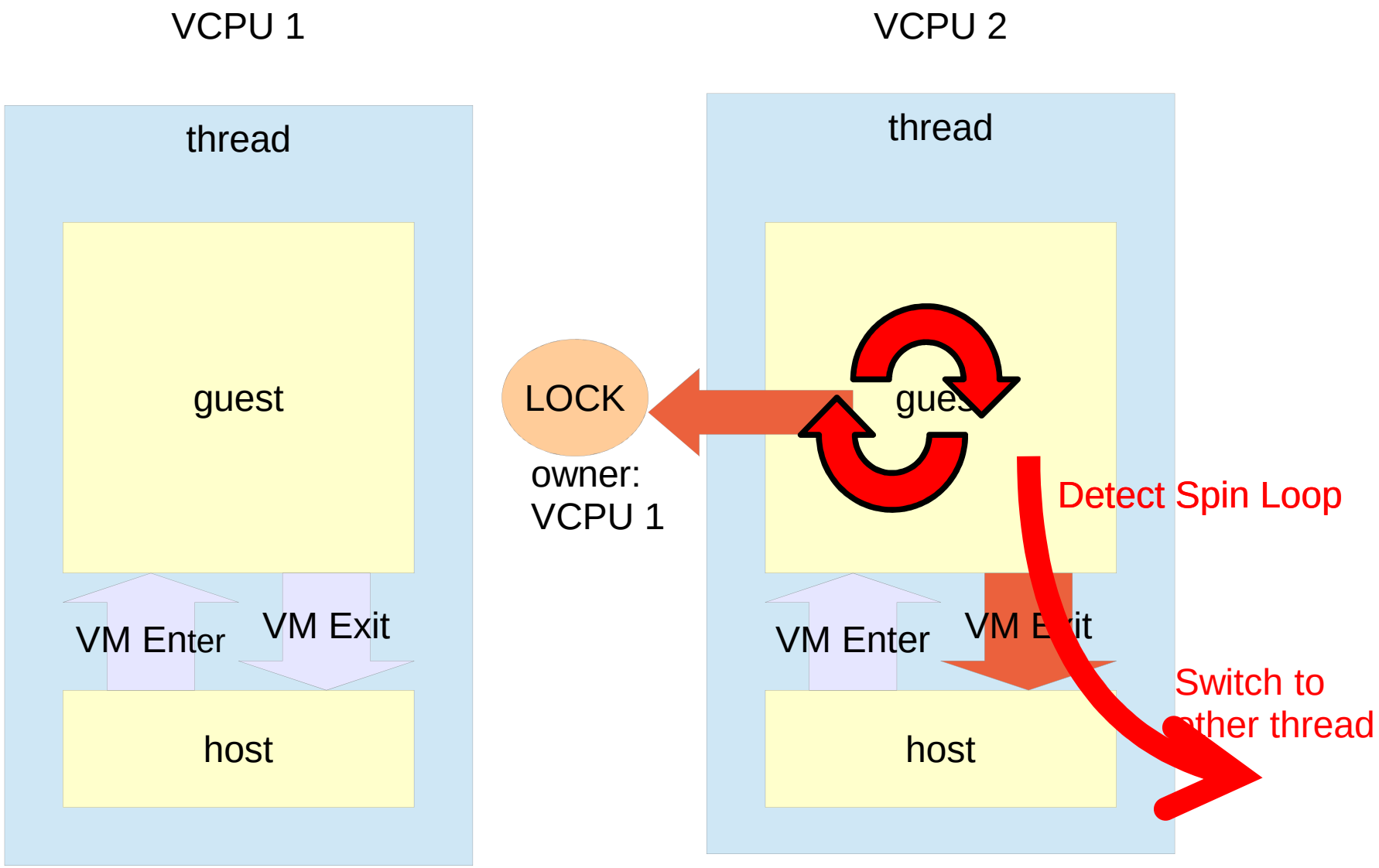
... but preempted by the host OS



Now, VCPU 2 wants the lock



w/ PAUSE-Loop Exit



HvNotifyLongSpinWait

- HYPER-V hypercall API
- Explicit scheduler hint from virtualization-aware guest OS
- cf. “Hypervisor Top Level Functional Specification v1.0.docx”
- KVM handles this in the same way as PAUSE-Loop Exit

PAUSE Loop-Exit

- kvm_intel module options

  ple_window=

  ple_gap=

TPR

- Task Priority Register
- Resides in Local APIC
- Controls interrupt acceptance
  - Larger value blocks more interrupts
- Many ways to access
  - Local APIC
  - RDMSR/WRMSR
  - MOV CR8

TPR

- Some OSes updates TPR very frequently
  - Windows
    - A workaround: disable ACPI
- Others don't use TPR at all
  - Linux
  - NetBSD

TPR shadow

- Redirect TPR traffic to virtual APIC memory w/o VM Exit
- VM Exit only if TPR value drops below the threshold in VMCS
- aka FlexPriority

TPR shadow

- kvm_intel module option

  flexpriority=1

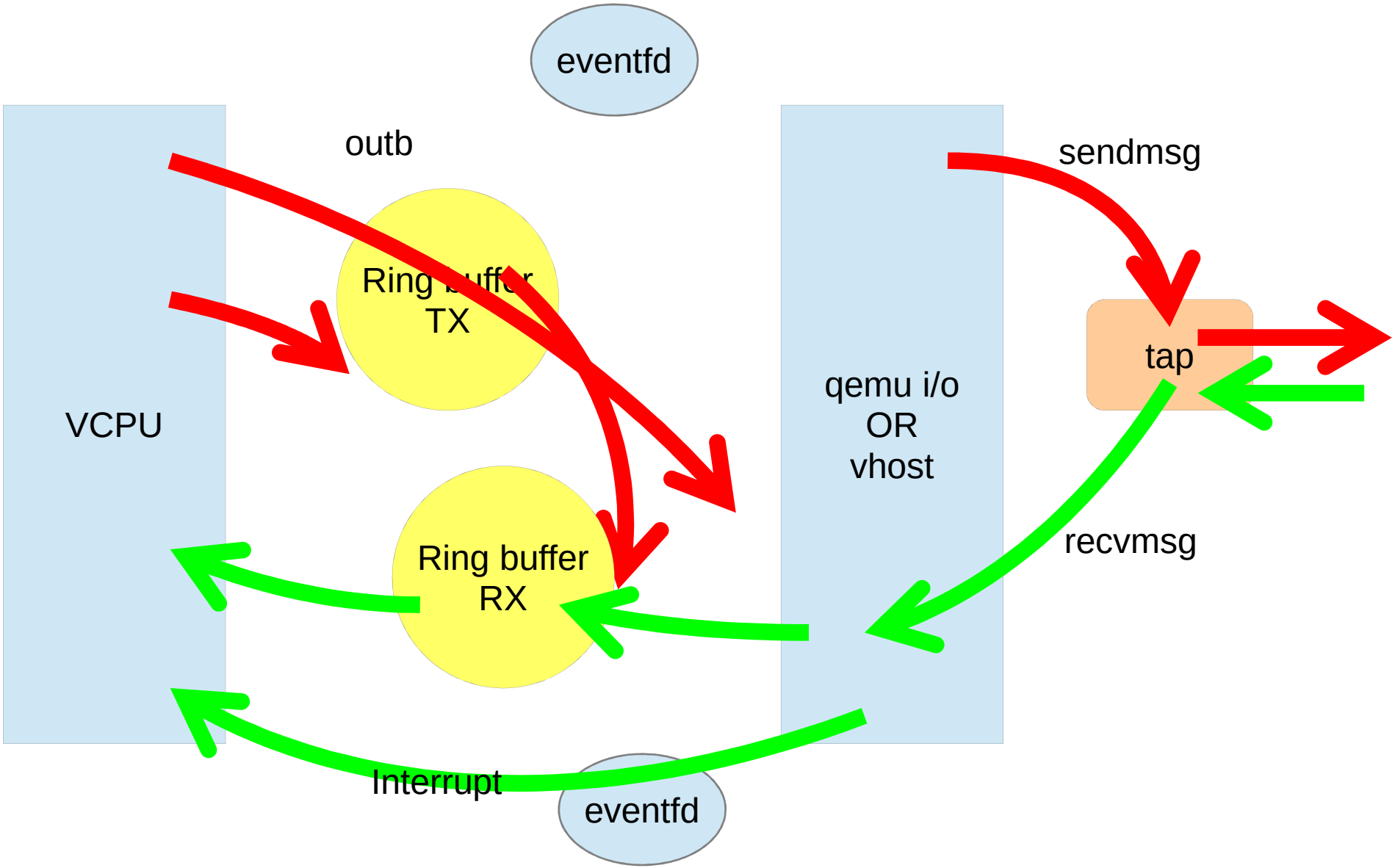
PV devices, PV drivers

- Emulation of “real” devices is complex, and often inefficient
- Virtual devices for virtualization-aware guests
  - virtio
    - net
    - blk
  - PV clock
  - balloon
  - PV ticket lock
  - ...

virtio

- “Virtio PCI Card Specification v0.9.4 DRAFT”
- Virtual PCI devices for virtual environments
  - Vendor ID 1AF4 Qumranet
  - Device ID 1000 - 103F
  - Subsystem Vendor ID
    - 1 Network card
    - 2 Block device
    - ...
- Not specific to KVM

virtio-net


eventfd
outb
sendmsg
Ring buffer
TX
tap
VCPU
qemu i/o
OR
vhost
Ring buffer
RX
recvmsg
Interrupt
eventfd

virtio

- qemu options

  -device virtio-net-pci,.....

  -device virtio-blk-pci,.....

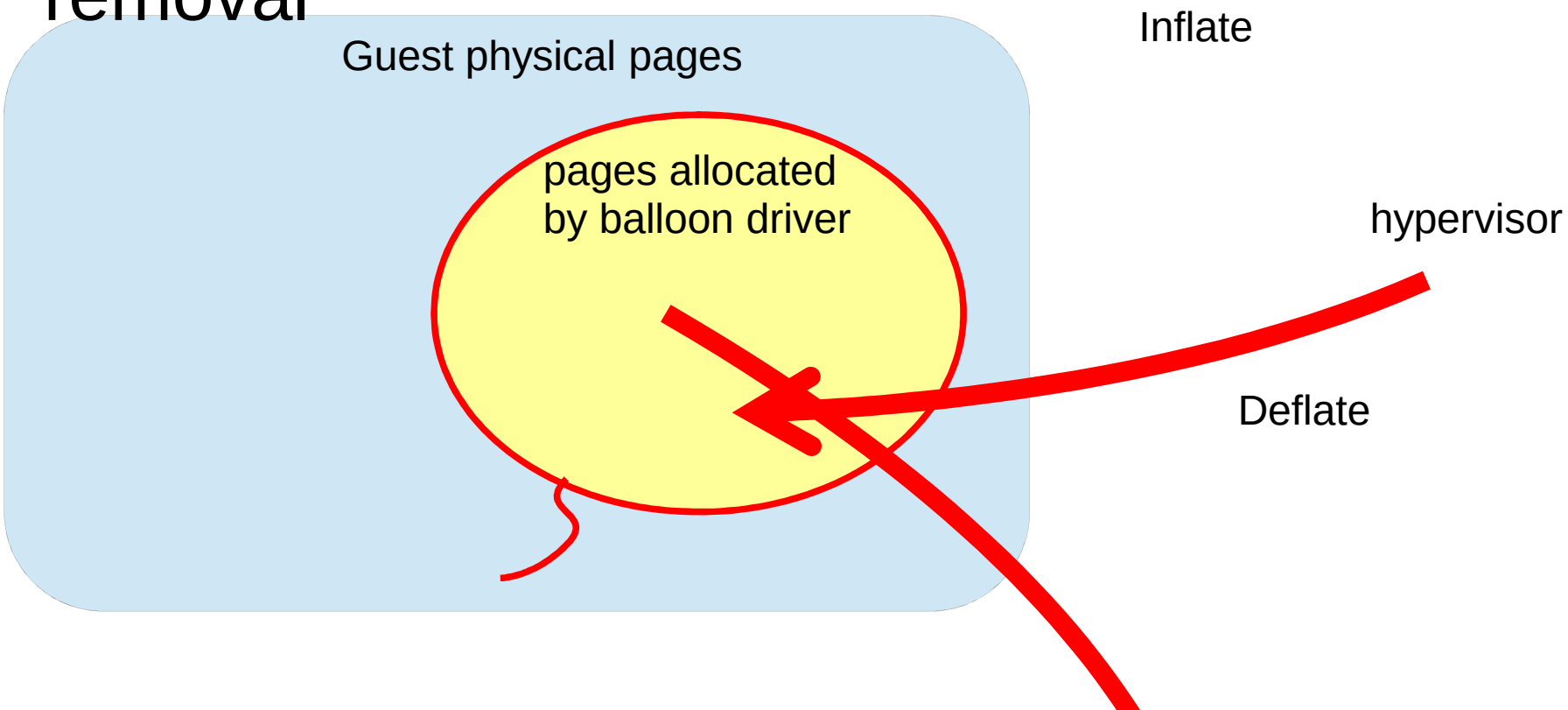
  -device virtio-balloon-pci,.....

PV Clock

- Before VM Enter, host writes:
  - TSC at last update (tsc_timestamp)
  - ns since boot (system_timestamp)
  - TSC rate (tsc_to_system_mul, tsc_shift)
- Guest reads the above and calculates:
  - system_timestamp +
  - (((rdtsc() - tsc_timestamp) * tsc_to_system_mul)
  - >> tsc_shift)

Ballooning

- Thin-provisioning, Overcommit
- Reduce the amount of guest memory w/o requiring the guest OS to support memory hot removal



virtio balloon

- Balloon operations are translated to madvise on qemu-kvm process space
  - Inflate -> madvise(MADV_DONTNEED)
    - NOTE: on Linux, DONTNEED discards data
  - Deflate -> madvise(MADV_WILLNEED)

virtio balloon

- qemu options

  -device virtio-balloon-pci,.....

- qemu monitor commands

  balloon

  info balloon

Ticket locks

- A lock consists of 2 counters
  - TAIL
  - HEAD

Ticket locks

- Initialize
  - TAIL = HEAD = 0
- Acquire
  - LOCAL_COPY_OF_TAIL = TAIL
    - “ticket”
  - TAIL += 1
  - Wait until HEAD == LOCAL_COPY_OF_TAIL
- Release
  - HEAD += 1

Ticket locks

- FIFO behaviour is desirable for fairness
- But horrible worst-case performance for virtualized environment
  - Hypervisor doesn't know the FIFO order
- Disabled for KVM guests

PV ticket locks

- Used for Xen
  - KVM version is still under development
- HALT instead of spin
- Upon unlock, issue an explicit hypercall to wake up waiters

PV ticket locks

- Acquire
  - LOCAL_COPY_OF_TAIL = TAIL
    - “ticket”
  - TAIL += 1
  - HALT until HEAD == LOCAL_COPY_OF_TAIL
- Release
  - HEAD += 1
  - Hypercall to unHALT waiters

Async PF (problem)

- Guest memory can be swapped out in host OS
- Access to the memory makes VCPU block
- During swap-in, the VCPU can't do anything useful

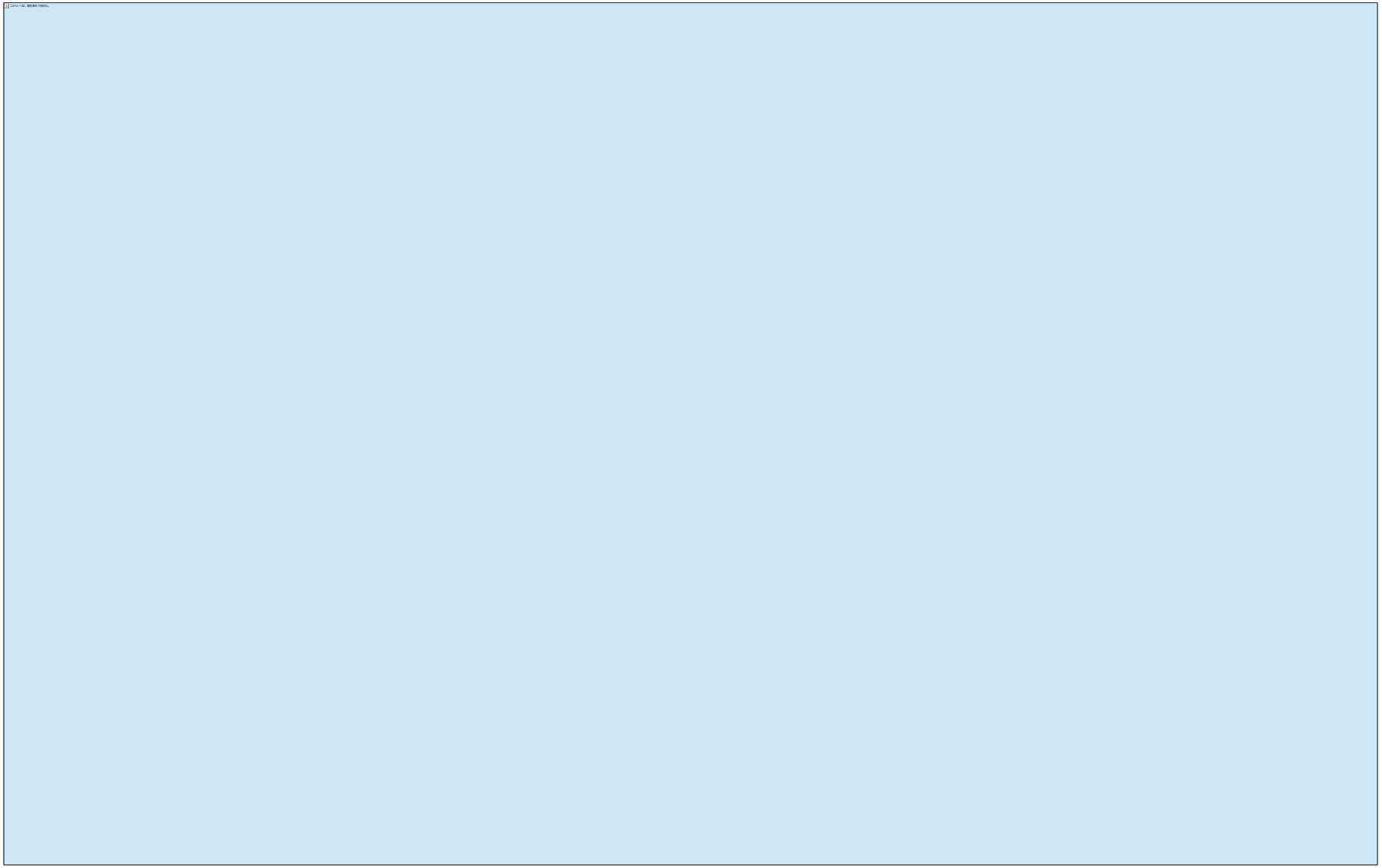
Async PF (FV guest)

- Perform swap-in in a separate worker thread in host OS and make VCPU block as if it “HLT”
  - KVM_REQ_APF_HALT
- A halted VCPU can serve virtual interrupts
  - Thus, if lucky enough, can switch to another guest thread, which might be able to run without the swapped-out memory

Async PF (PV guest)

- If supported by a guest
  - MSR_KVM_ASYNC_PF_EN
- Explicitly notify guest
  - Per-VCPU mailbox; apf_reason
    - KVM_PV_REASON_PAGE_NOT_PRESENT
    - KVM_PV_REASON_PAGE_READY
  - Exception #14 (page fault)
- Allows PV-aware-guest block and unblock its threads

Guest OS PV support



Misc Linux features used by KVM

- vhostnet
- eventfd
- Linux native AIO (libaio)
- signalfd

vhostnet

- Move virtio queue handling from userland (qemu) to kernel thread (vhost)
- Improve perfomance, mainly latencies

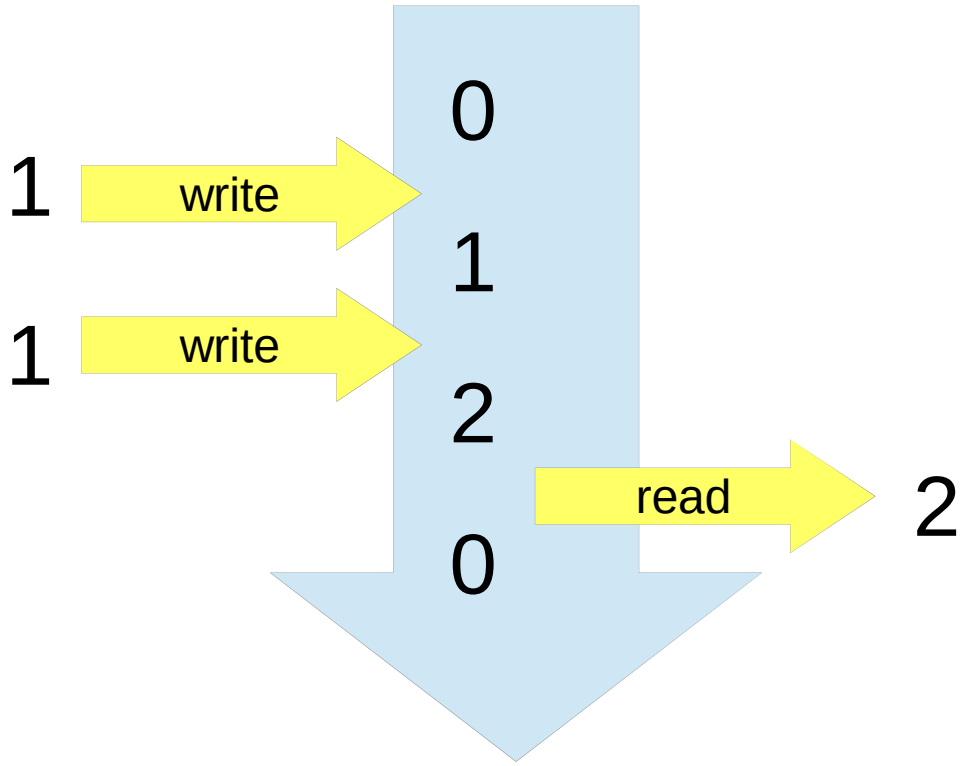
vhostnet

- qemu options

  -netdev .....,vhost=on

eventfd

- int eventfd(unsigned int initval, int flags)
- pollable

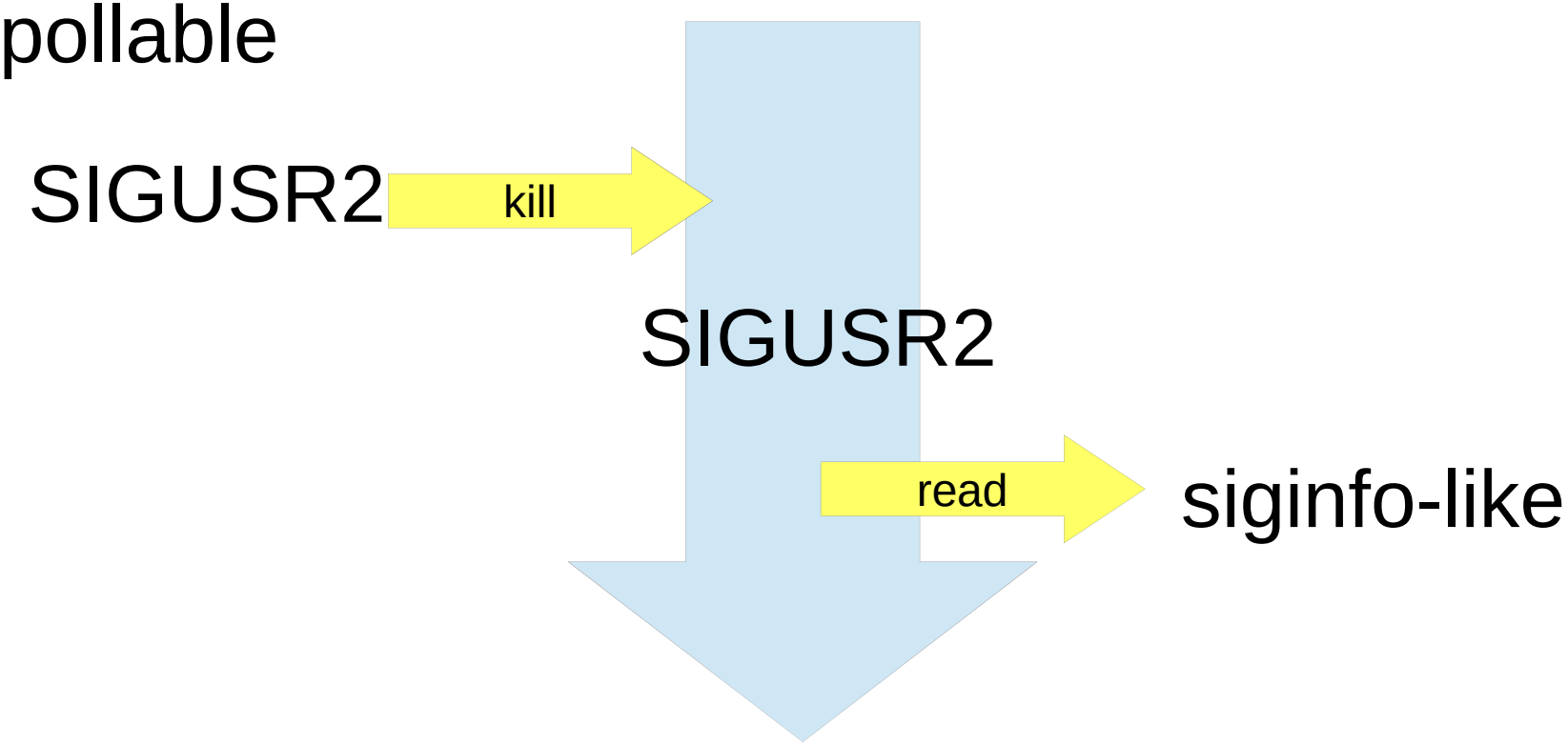


libaio

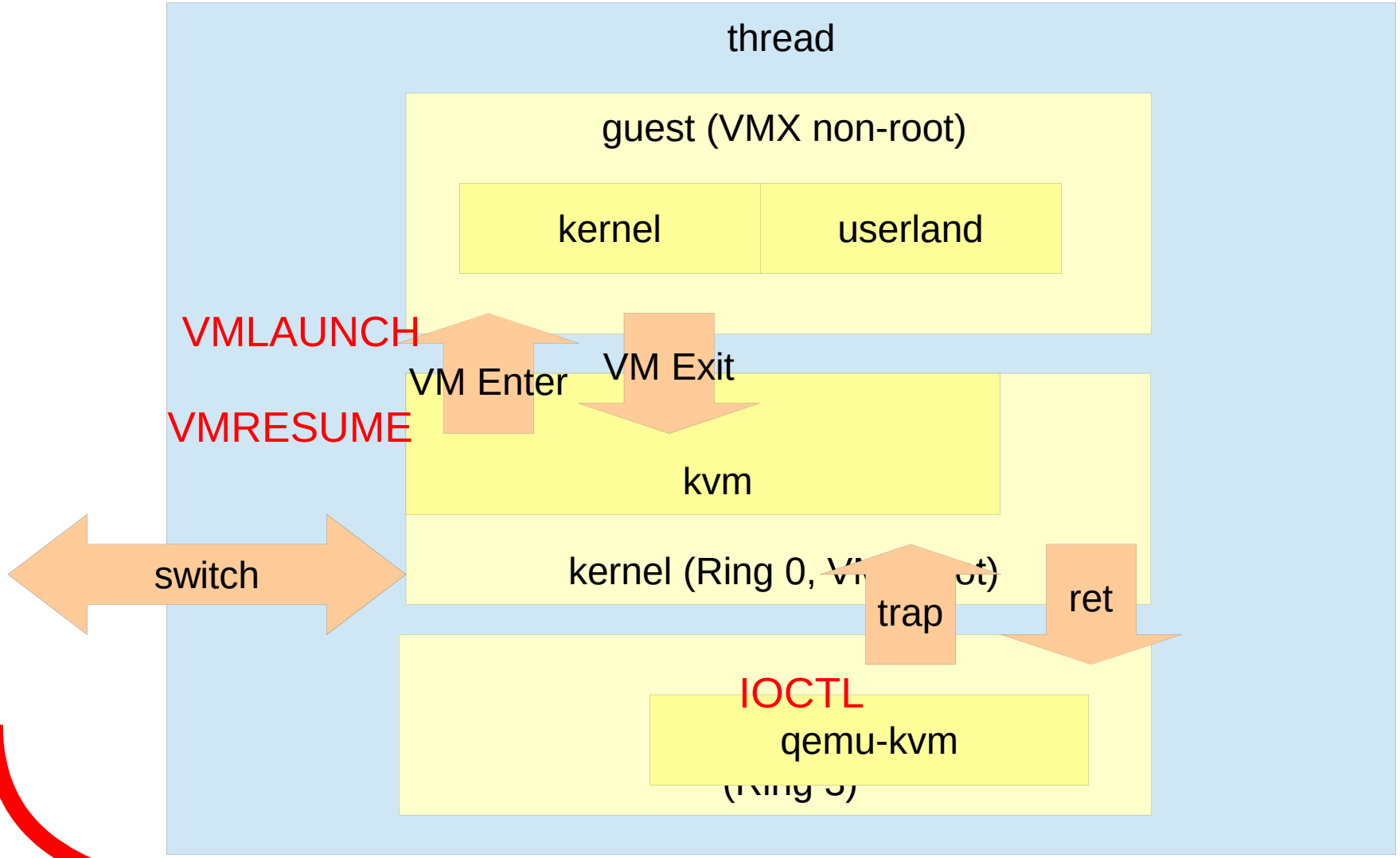
- Kernel-supported AIO
- API different from POSIX AIO

signalfd

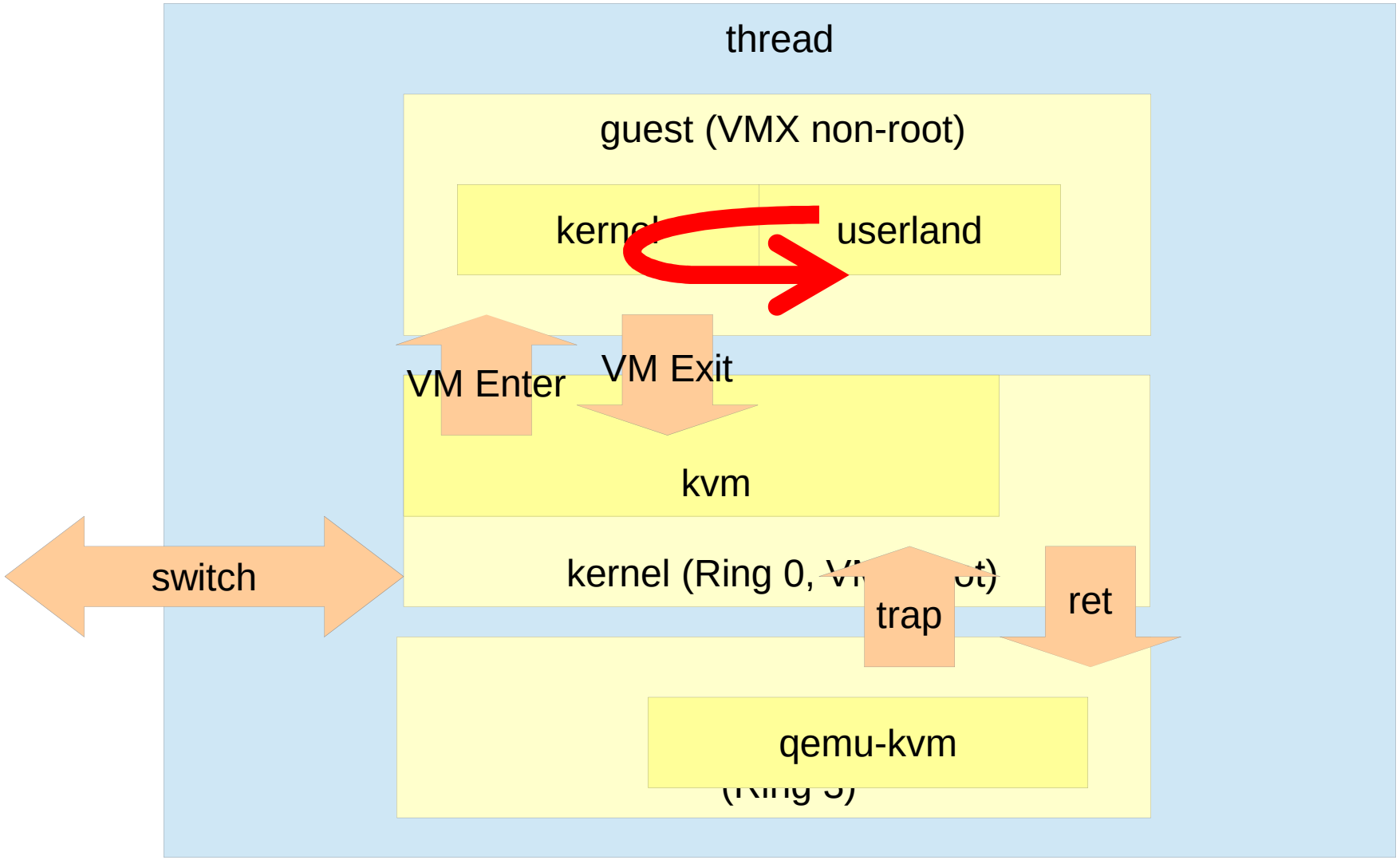
- int signalfd(int fd, const sigset_t *mask, int flags);
- receive signals via a descriptor
- pollable



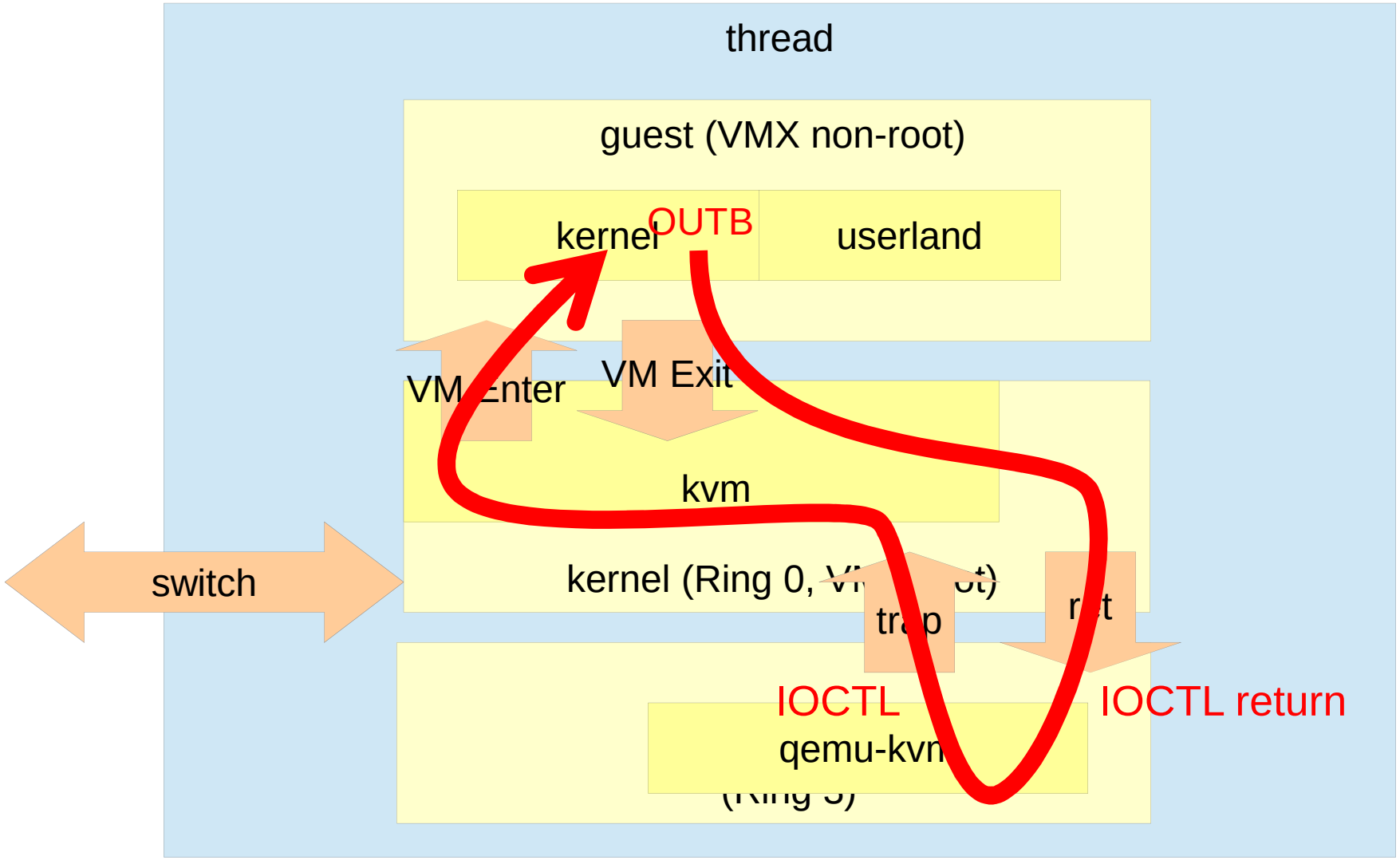
Example: entering guest



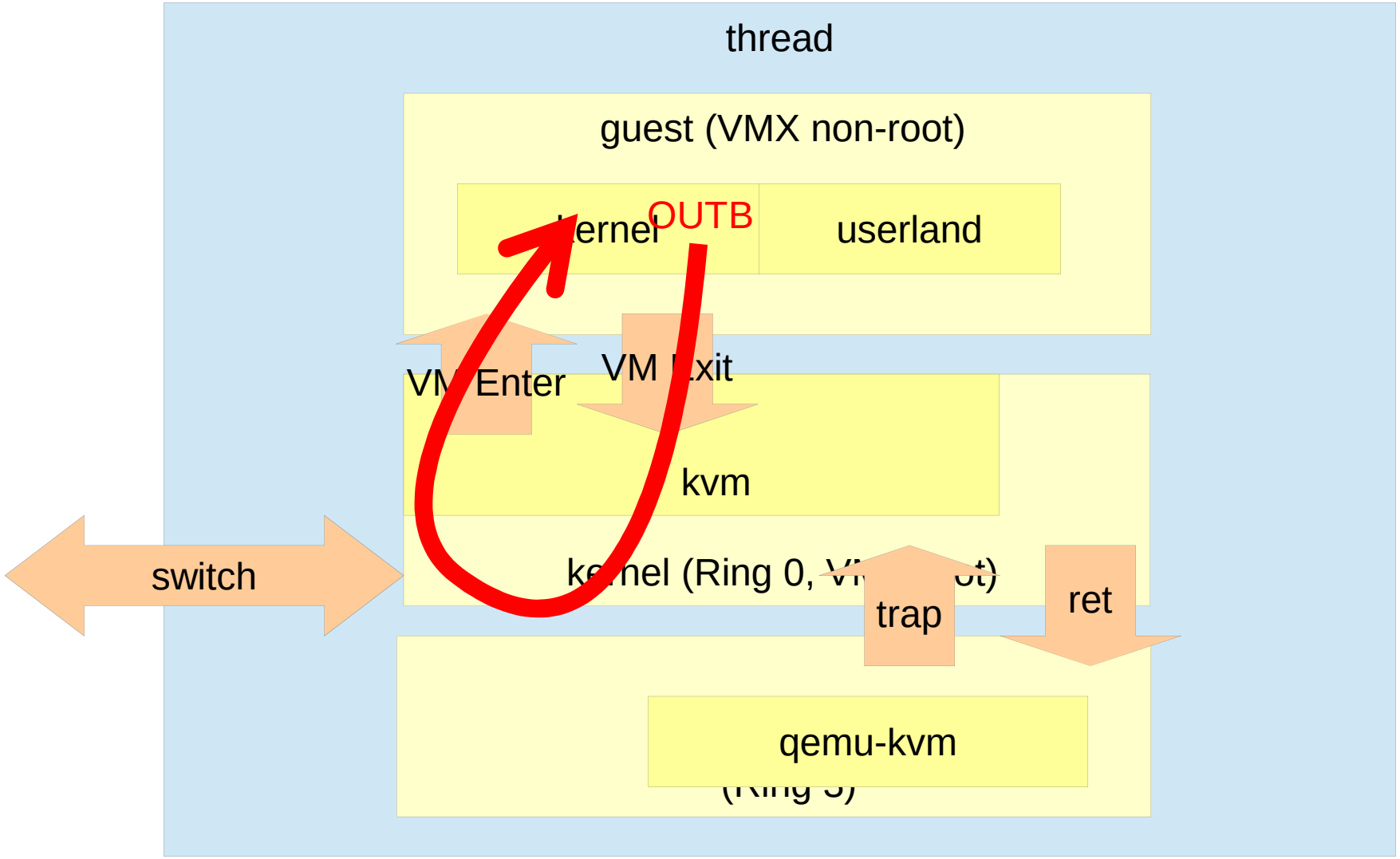
Example: guest system call



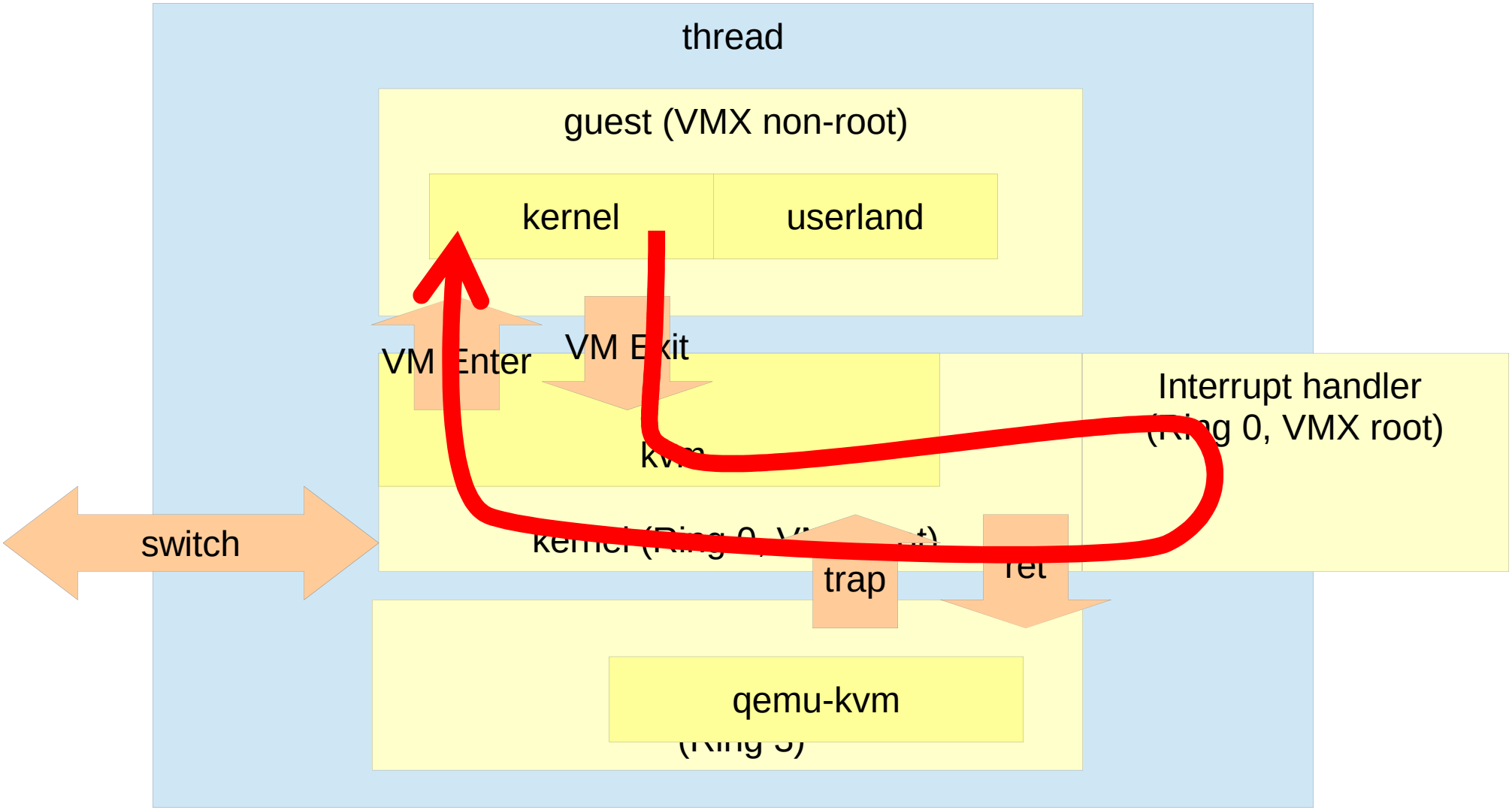
Example: guest I/O (qemu)



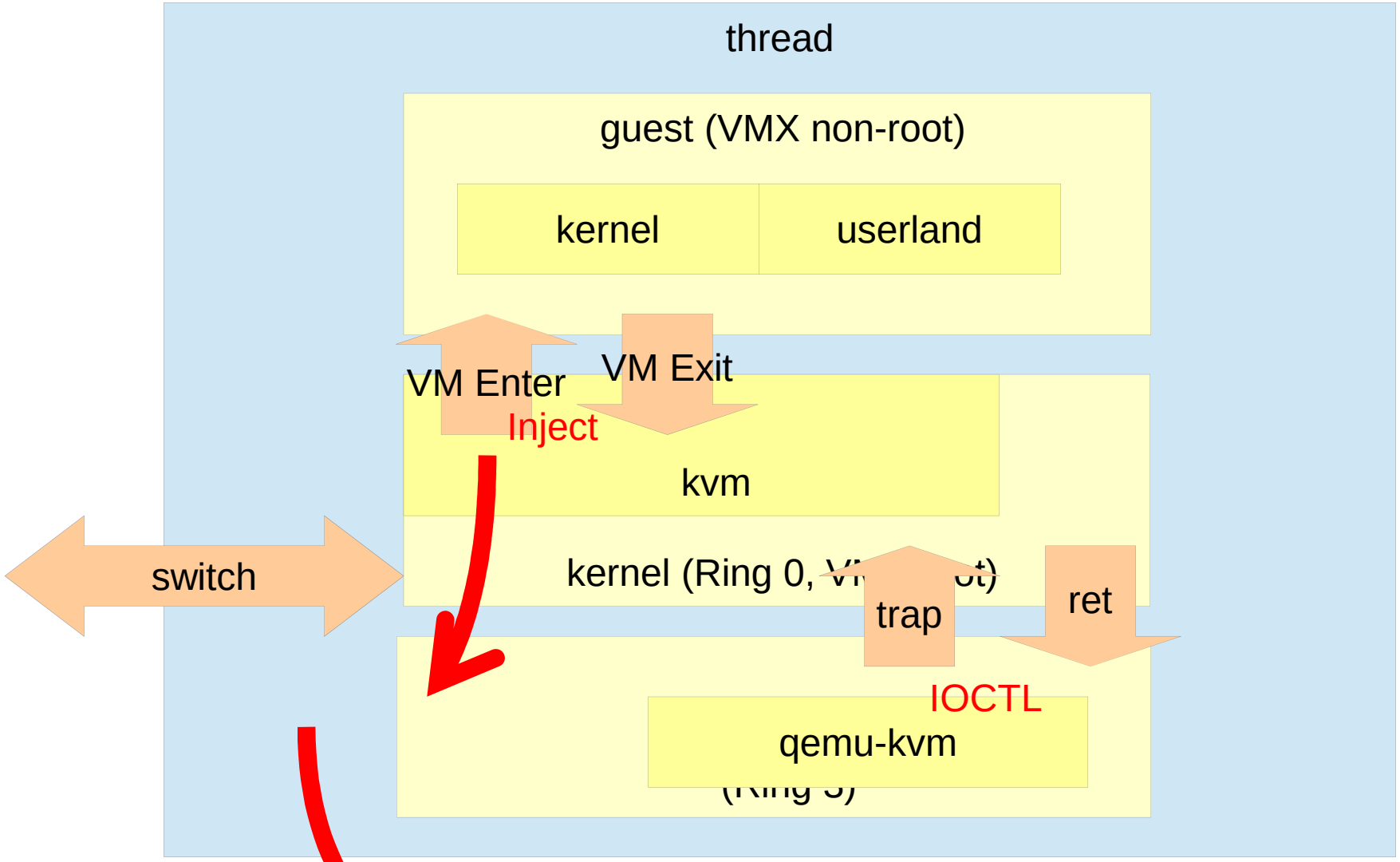
Example: guest I/O (vhost)



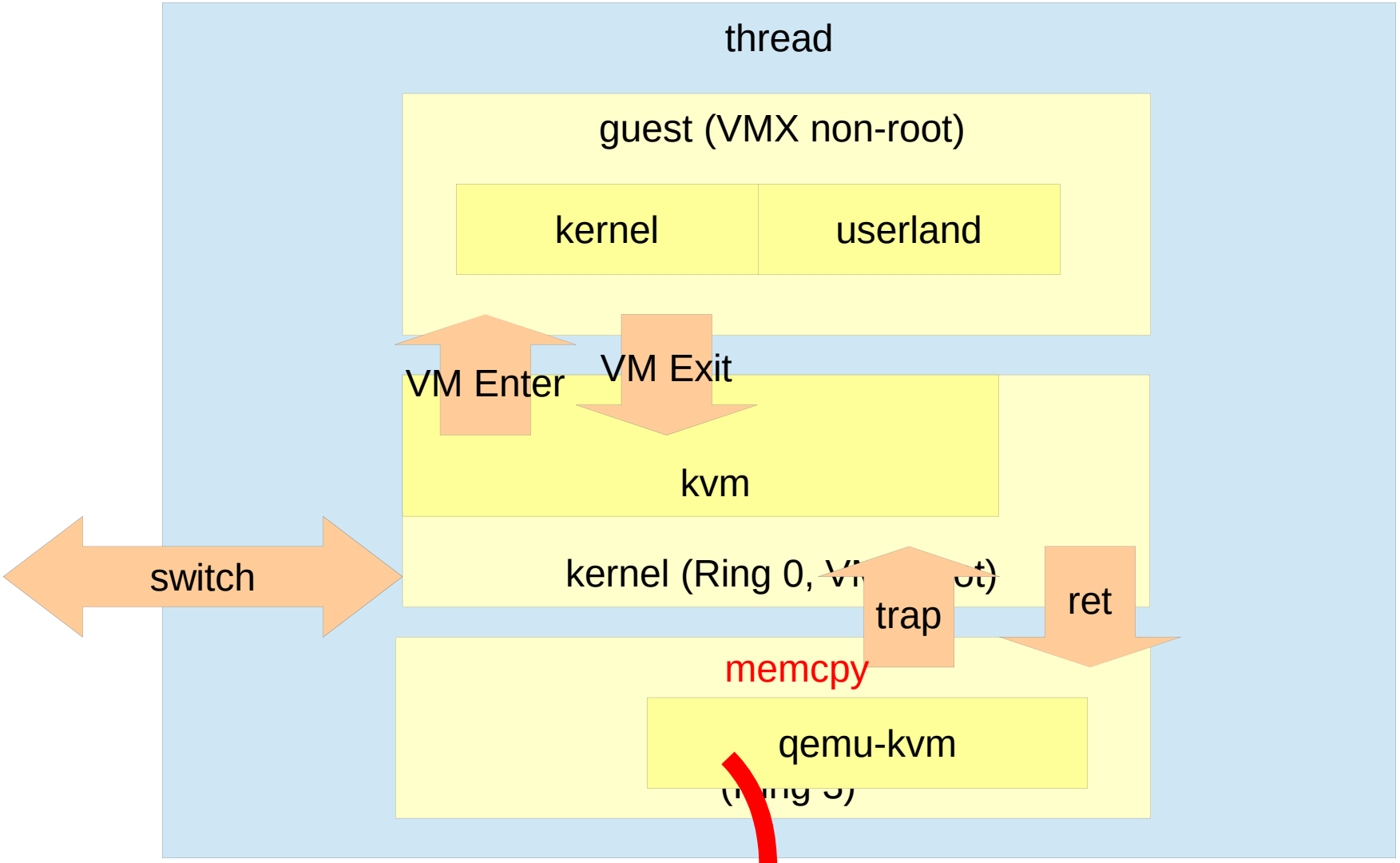
Example: real interrupt



Example: guest interrupt

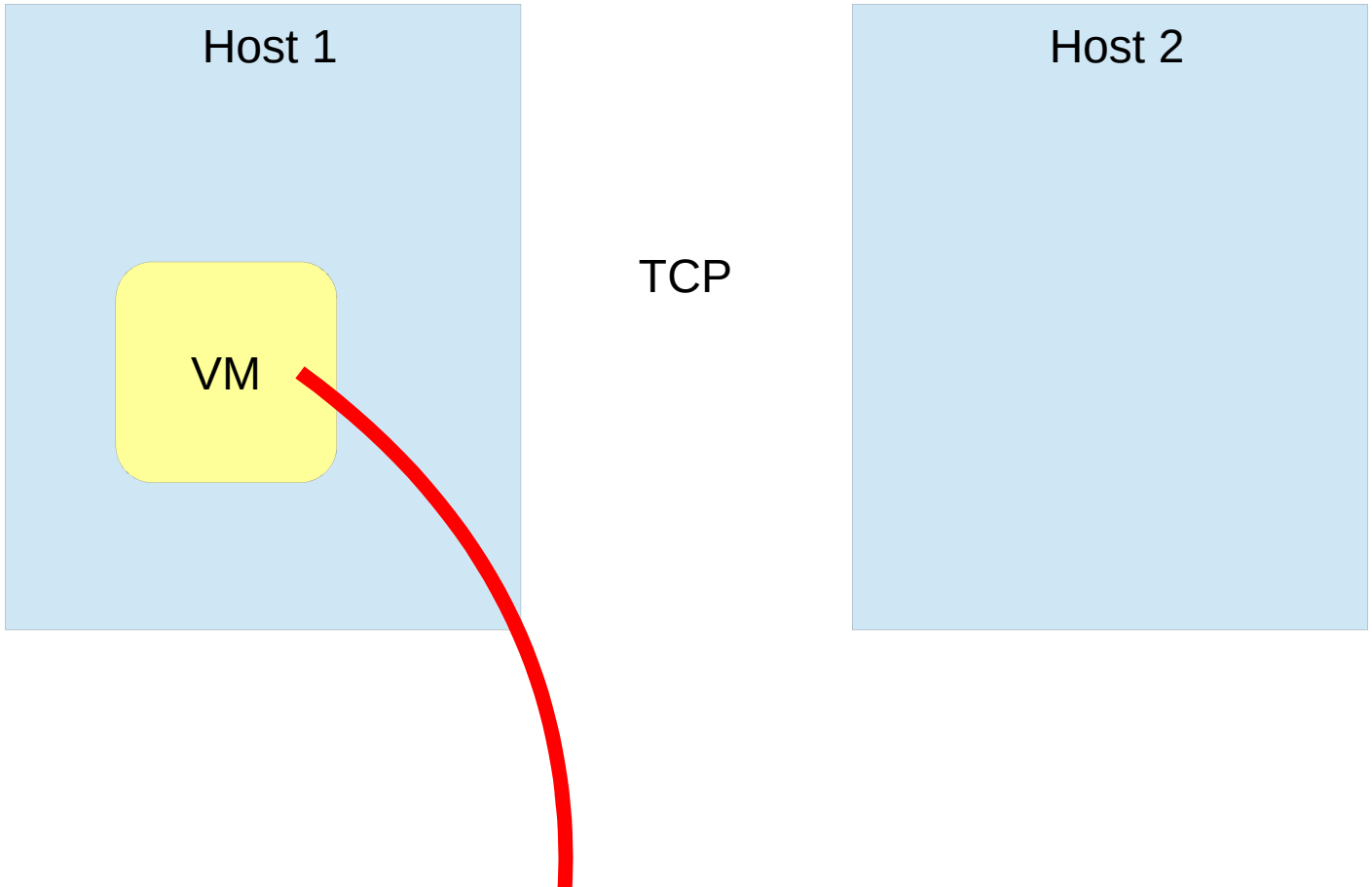


Example: guest DMA



Live migration

- Move a VM to another host over a network link



Live migration

- Naive way
  - Stop VM
  - Transfer VM
    - device state
    - transfer memory <- EXPENSIVE!
  - Start VM

Live migration

- pre-copy (current qemu-kvm implementation)
  - Transfer VM (1)
    - enable dirty page tracking
    - transfer clean memory
  - Stop VM
  - Transfer VM (2)
    - device state
    - transfer dirty memory <- expected to be small
  - Start VM

Dirty page tracking

- Detect and report modification of guest pages
  - Trap modifications by removing write access from shadow page table entries or EPT
  - Record the modified pages in a bitmap
  - IOCTL to query and clear the bitmap
- Used for
  - Live migration
  - Emulation of frame buffer devices

Live migration

- post-copy
  - Stop VM
  - Transfer VM (1)
    - device state
  - Start VM
  - Transfer VM (2)
    - background / on-demand transfer of memory

Live migration (disk)

- Disk is even more expensive to transfer than memory
- Common techniques
  - Share a disk among hosts
    - iSCSI, SAN, NFS, ...
  - Keep disks in-sync
    - NBD, ...
  - Copy disk on migration
    - qemu block migration (migrate -b)

Live migration w/ block migration

- Transfer VM (1)
  - enable dirty page tracking
  - enable dirty disk block tracking
    - in-core dirty bitmap similar to memory
  - transfer clean memory
  - transfer clean disk blocks
- Stop VM
- Transfer VM (2)
  - device state
  - transfer dirty memory <- expected to be small
  - transfer dirty disk blocks <- expected to be small
- Start VM