CS695 Topics in Virtualization and Cloud Computing

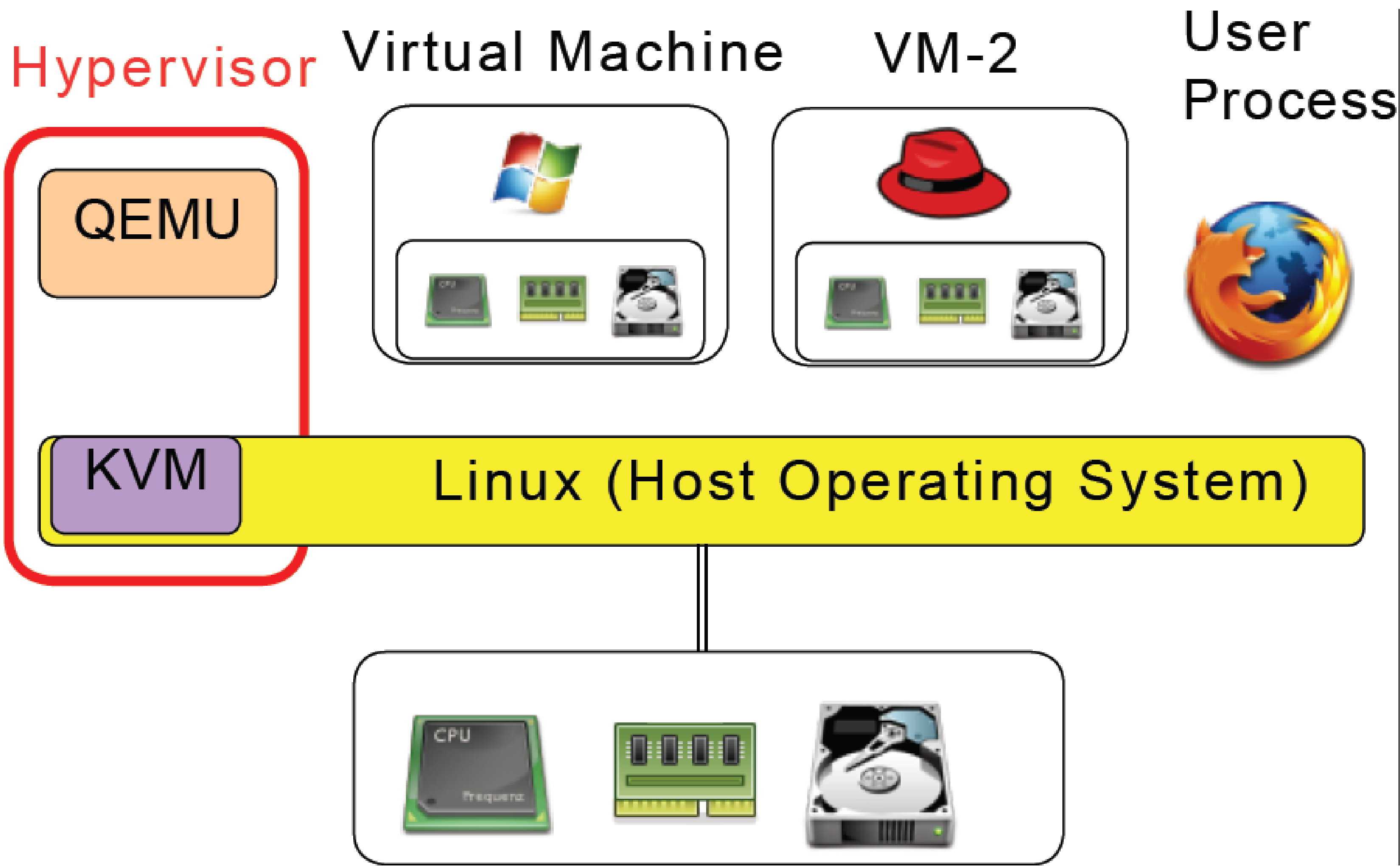
Virtualization in Linux KVM + QEMU

Senthil, Puru, Prateek and Shashank

Topics covered

- KVM and QEMU Architecture
  - VTx support
  - CPU virtualization in KMV
  - Memory virtualization techniques
    - shadow page table
    - EPT/NPT page table
  - IO virtualization in QEMU
- KVM and QEMU usage
  - Virtual disk creation
  - Creating virtual machines
  - Copy-on-write disks

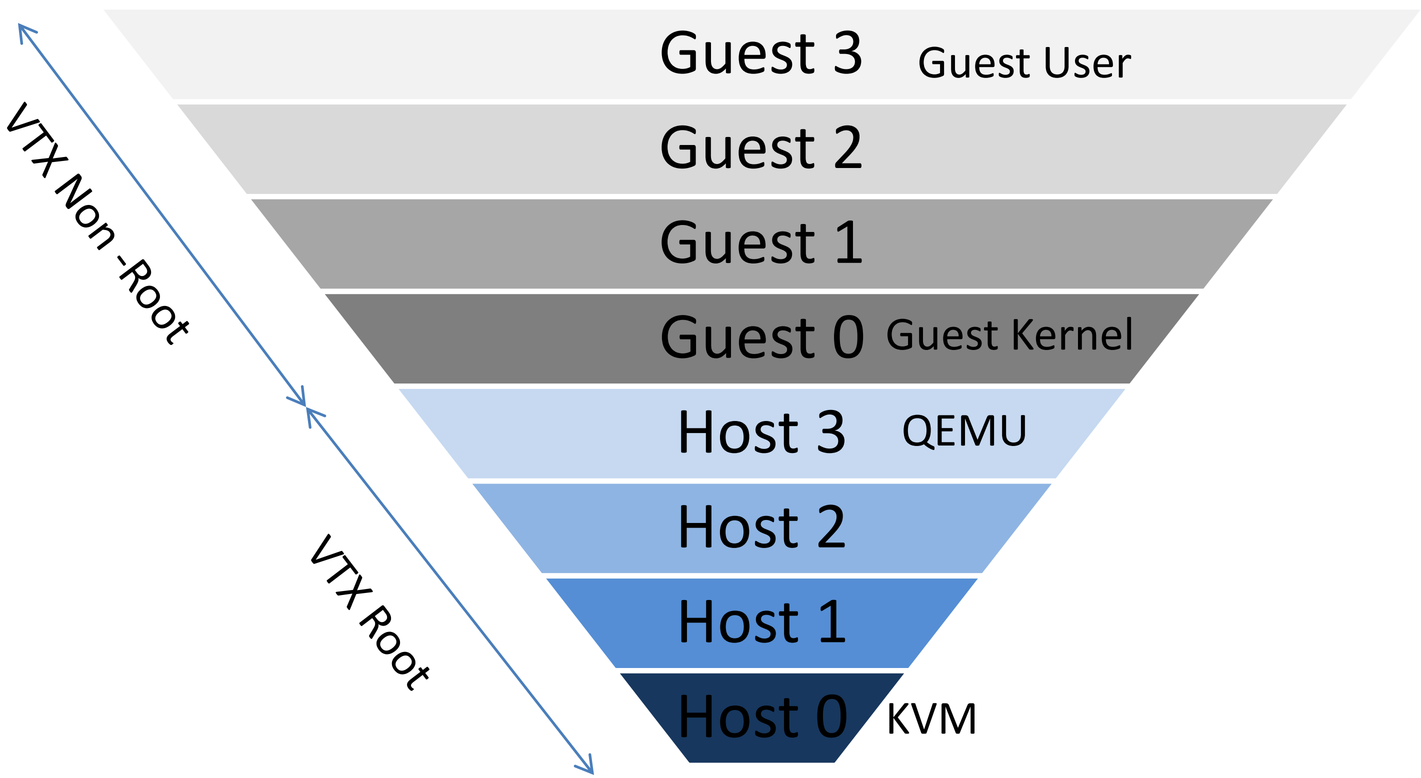
KVM + QEMU - Architecture



KVM + QEMU – Architecture

- Need for hardware support
  - less privileged rings (rings > 0) are not sufficient to run guest – sensitive unprivileged instructions
  - Should go for
    - Binary instrumentation/ patching
    - paravirtualization
    - VTx and AMD-V
  - 4 different address spaces - host physical, host virtual, guest physical and guest virtual

X86 VTx support



Communication Channels

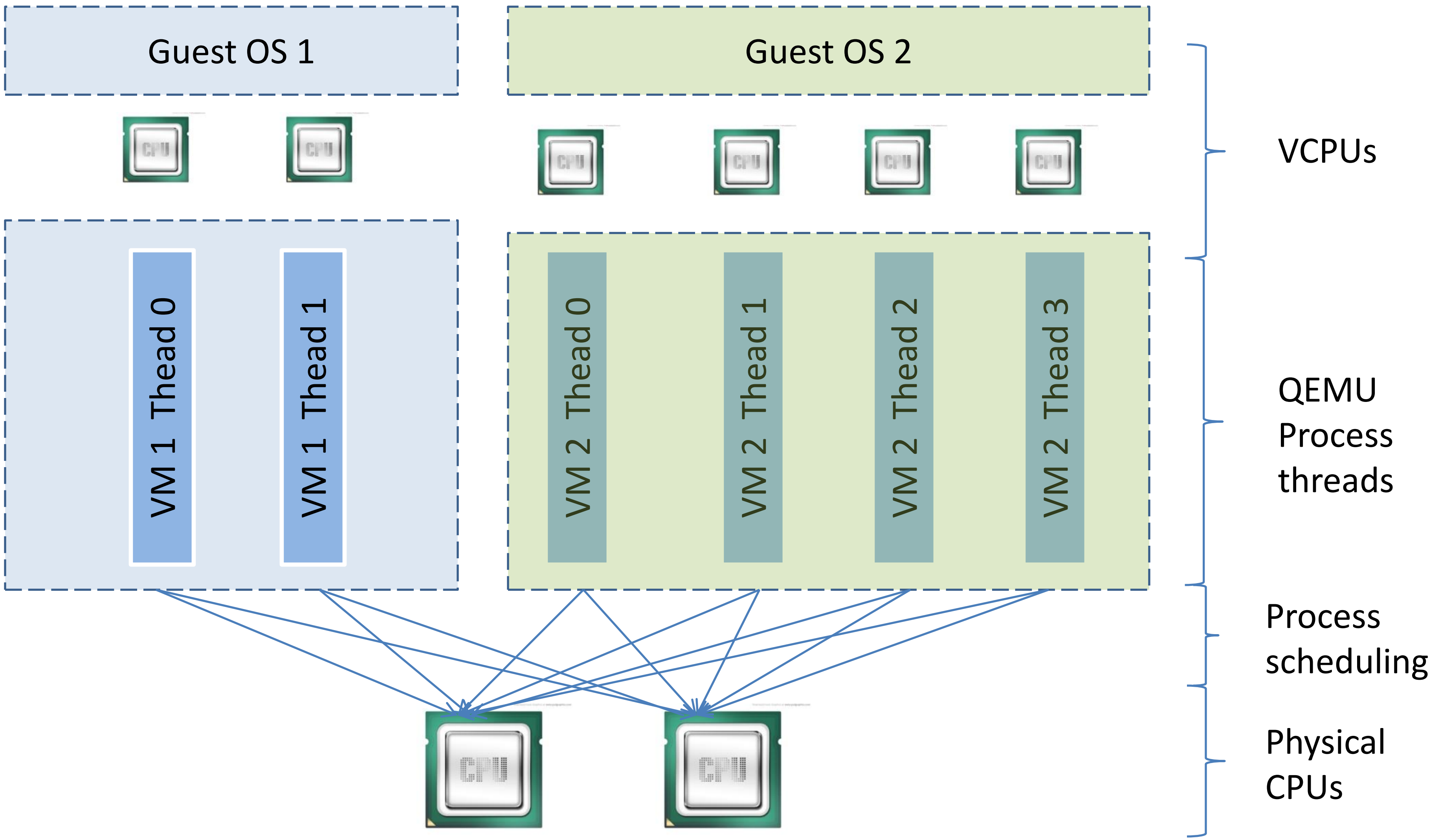
X86 VMX Instructions

- Controls transition between VMX root and VMX non-root
- VMX root -> VMX non-root - **VM Entry**
- VMX non-root -> VMX root – **VM Exit**
- Example instructions
  - VMXON – enables VMX Operation
  - VMXOFF – disable VMX Operation
  - VMLAUNCH – VM Entry
  - VMRESUME – VM Entry
  - VMREAD – read from VMCS
  - VMWRITE – write to VMCS

X86 VMCS Structure

- Controls CPU behavior in VTx non root mode
- 4KB structure – configured by KVM
- Also provides space for guest and host register save & restore
- Example fields
  - HLT exiting – if 1 VM Exit on HLT
  - CR3-load exiting – if 1 VM Exit on CR3 load
  - Exception Bitmap – if bit i is set, VM Exits on exception i
  - VM-entry interrupt – To deliver interrupts during VM Entry

CPU Virtualization in KVM

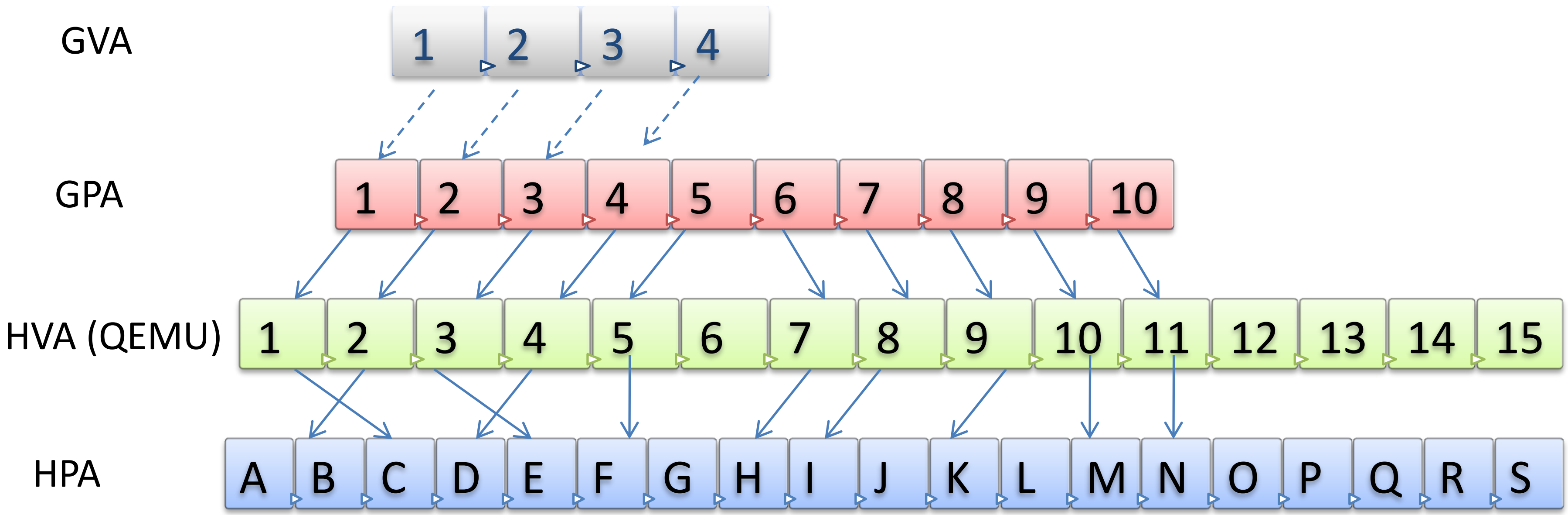


Shadow page table

- Problems in memory virtualization
  - 3 levels of indirection, MMU can translate 1 level
  - GVA -> GPA -> HVA -> HPA must be achieved

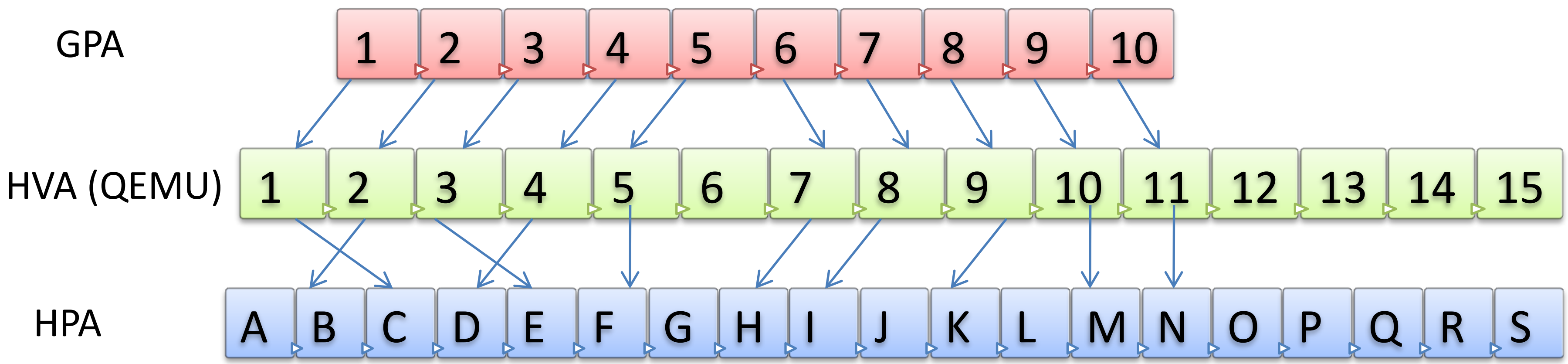
- Solution1 - Shadow page table
  - Contains GVA -> HPA. MMU will use this instead of guest page table
  - One shadow table for each guest page table
  - Incrementally build

Shadow page table building



- Guest wants to create a linear mapping for a process
- Guest does pure demand
- QEMU knows GPA-> HVA mapping (malloc())

Shadow page table building



Shadow page table
GVA -> HPA

1	
2	
3	

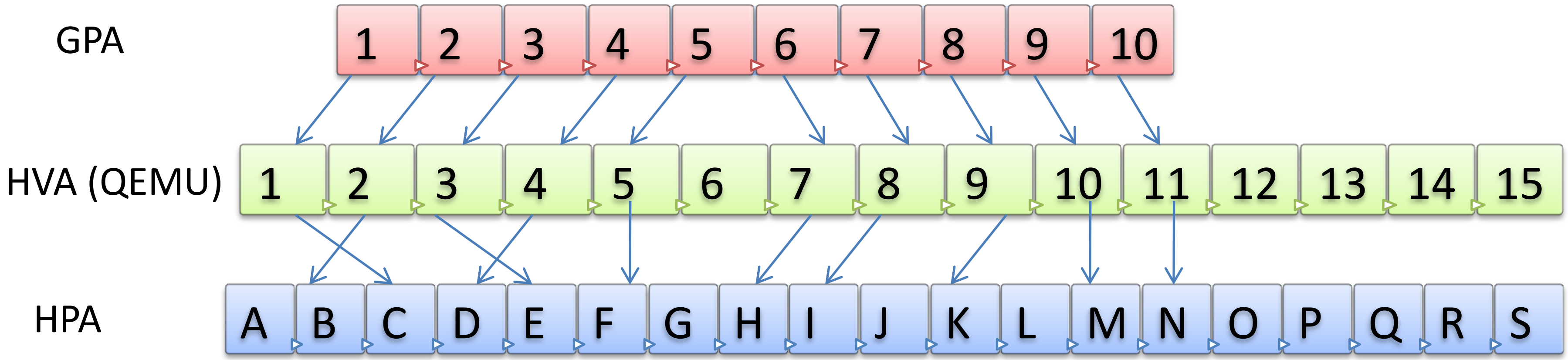
Guest process page table
GVA -> GPA (Read only)

1	
2	
3	

Step 1:

- Guest tries to map GVA 1 -> GPA 1
- Page fault (because of RO) causes VM exit
- KVM sees GPA as 1 by instruction emulation /using register contents

Shadow page table building



Shadow page table
GVA -> HPA

1	
2	
3	

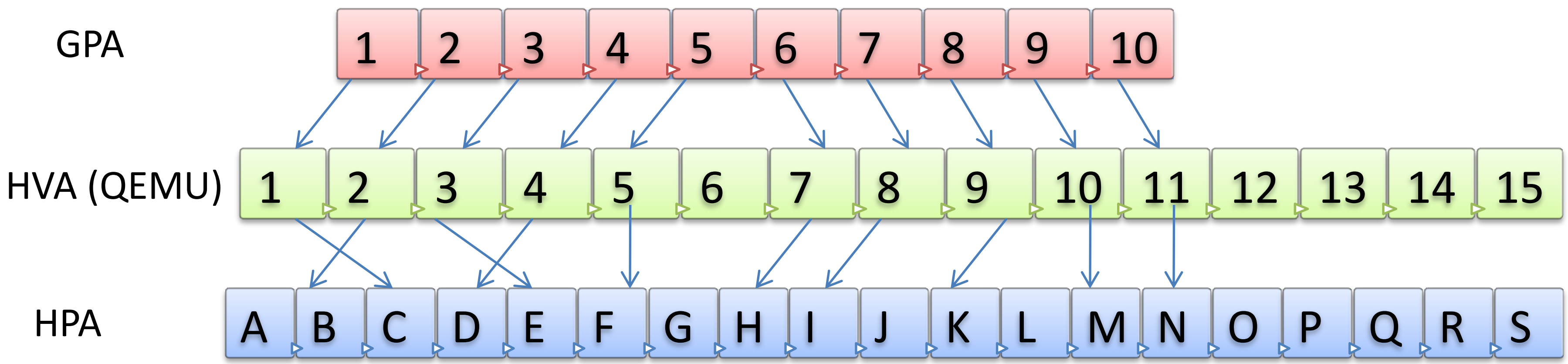
Guest process page table
GVA -> GPA (Read only)

1	
2	
3	

Step 2:

- GPA 1 -> HVA 1 is obtained
- This possible because GPA -> HVA mapping is known to QEMU/KMV

Shadow page table building



Shadow page table
GVA -> HPA

1	
2	
3	

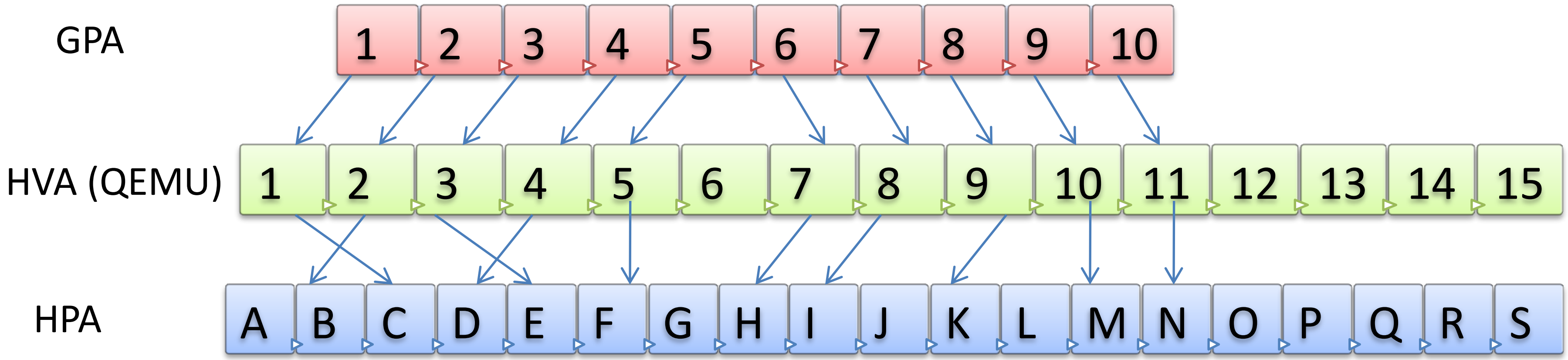
Guest process page table
GVA -> GPA (Read only)

1	
2	
3	

Step 3:

- KVM does lookup on QEMU's page table to find out HVA->HPA
- KVM finds out HVA 1 -> HPA C

Shadow page table building



Shadow page table
GVA -> HPA

1	C
2	
3	

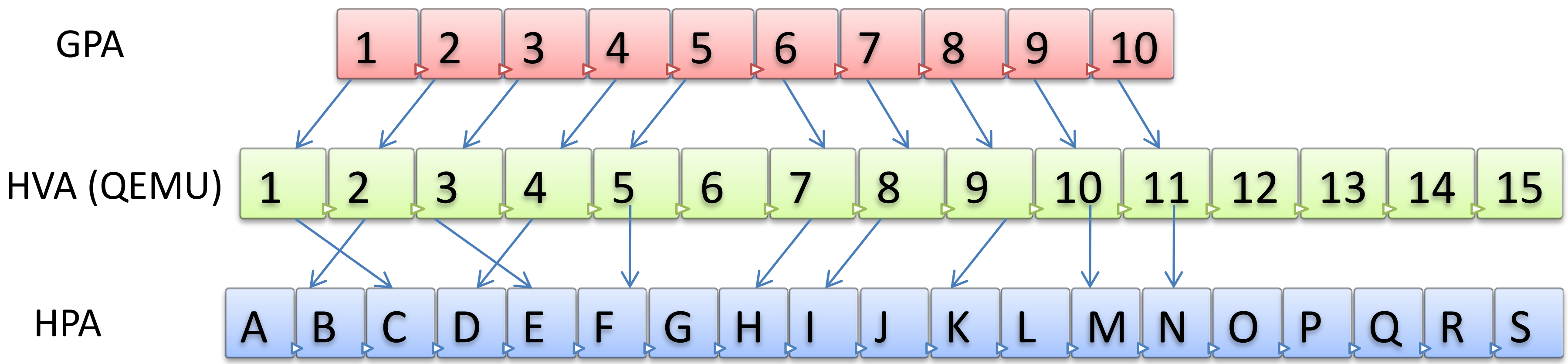
Guest process page table
GVA -> GPA (Read only)

1	1
2	
3	

Step 4:

- KVM updates shadow page table with GVA 1 -> HPA C
- KVM also updates guest page table – by emulating the instruction which tried to map GVA 1 -> GPA 1
- GVA -> GPA -> HVA -> HPA is done

Shadow page table building



Shadow page table
GVA -> HPA

1	**C**
2	**B**
3	**E**

Guest process page table
GVA -> GPA (Read only)

1	**1**
2	**2**
3	**3**

Step 5:

- Similarly other entries are update as and when page fault happens
- GVA 2 -> GPA 2 -> HVA 2 -> HPA B
- GVA 3 -> GPA 3 -> HVA 3 -> HPA E

Shadow page table (additional info)

- Additional questions
  - How to identify pages used in page tables to write protect them ?
  - How to remove write protection when a page is not used in any page table ?
  - What happens when pure demand paging is not used i.e. (guest builds the page table before loading on CR3) ?

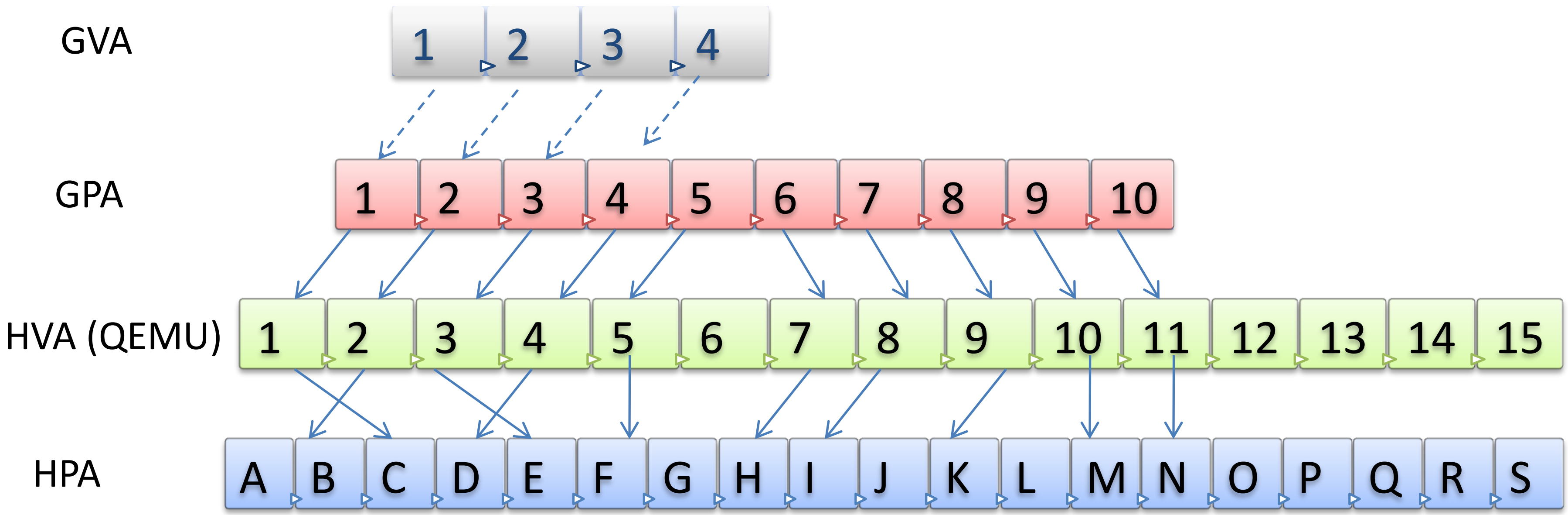
- Advantages
  - No guest OS change is required
  - Any OS can be guest
  - No special hardware is required

- Disadvantages
  - For every page table used by guest.. Shadow version has to be kept.
  - Shadow page table must be consistent with guest and host
  - Caching shadow page table needs considerable memory

EPT/NPT Basics

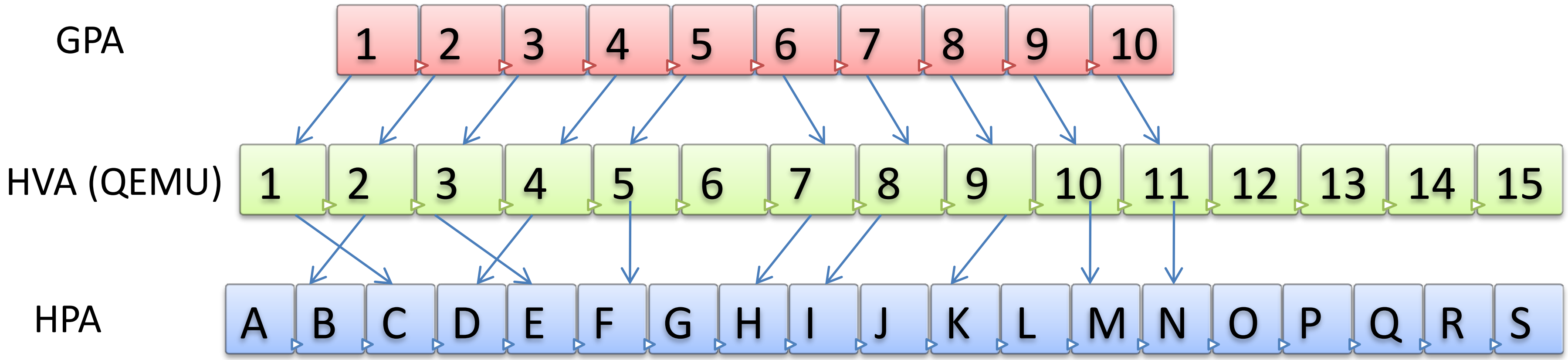
- Solution2 – EPT/NPT hardware support
  - EPT/NTP enabled MMU can translate two levels of indirection.
  - First one from GVA -> GPA and second from GPA -> HPA
  - GVA -> GPA is maintained by guest and GPA -> HPA is maintained by KVM
  - KVM does GPA -> HVA translation - because malloc()
  - MMU walks EPT table for every GPA

EPT/NPT Building



- EPT solution consists of two tables
  - GPA -> HPA - EPT table
  - GVA -> GPA – guest process page table
- MMU accesses these two tables to complete address translation
- Guest has full rights on its page table

EPT/NPT Building



EPT/NTP page table

GPA -> HPA

1	
2	
3	

Guest process page table

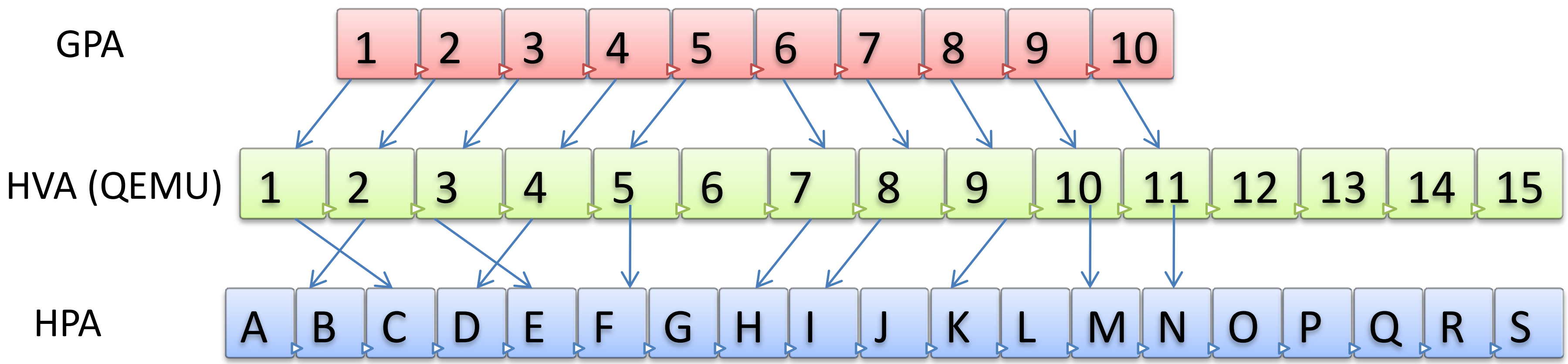
GVA -> GPA

1	1
2	
3	

Step 1:

- Guest tries to access linear address 1
- Will not cause page fault, because VMCS is configured not to cause page fault VM exits
- Guest OS will handle this and fill GVA 1 -> GPA 1

EPT/NPT Building



EPT/NTP page table
GPA -> HPA

1	
2	
3	

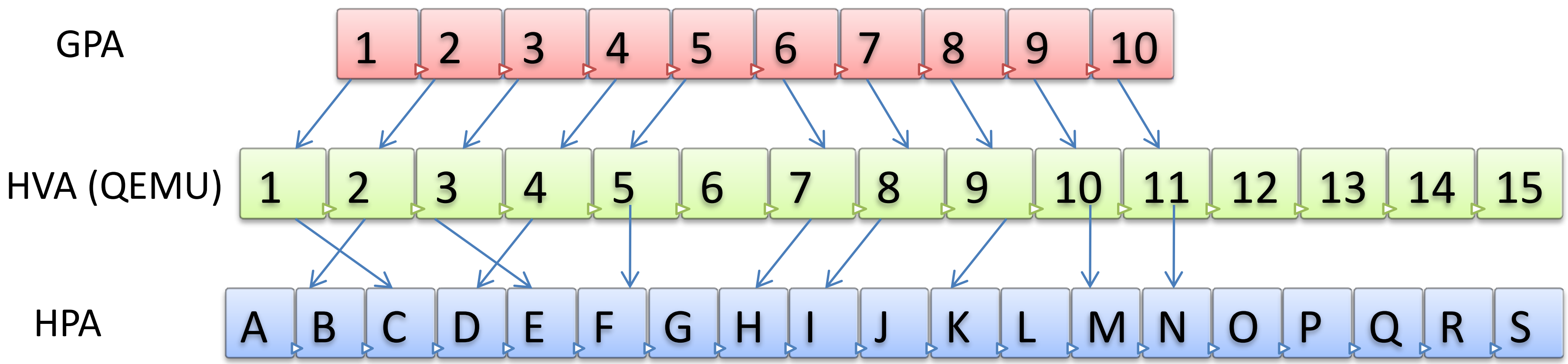
Guest process page table
GVA -> GPA

1	**1**
2	
3	

Step 2:

- When guest access the linear memory address GVA 1, the hardware gets GPA as 1 using guest page table
- And tries to figure out corresponding HPA using EPT table and cause EPT violation

EPT/NPT Building



EPT/NTP page table
GPA -> HPA

1	C
2	
3	

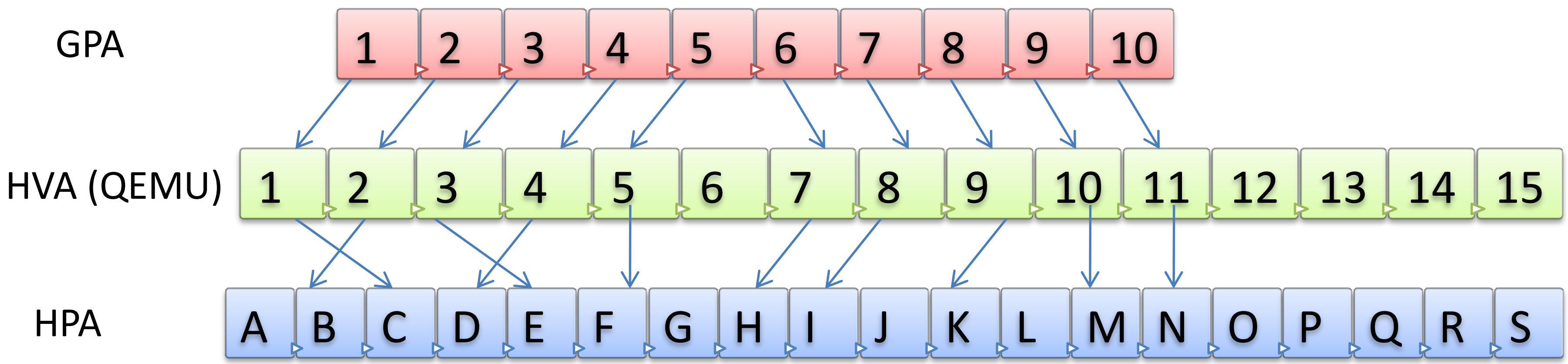
Guest process page table
GVA -> GPA

1	1
2	
3	

Step 3:

- EPT violation occurred because corresponding HPA is not mapped.
- KVM will fill this entry using GPA 1-> HVA 1 -> HPA C

EPT/NPT Building



EPT/NTP page table
GPA -> HPA

1	C
2	
3	

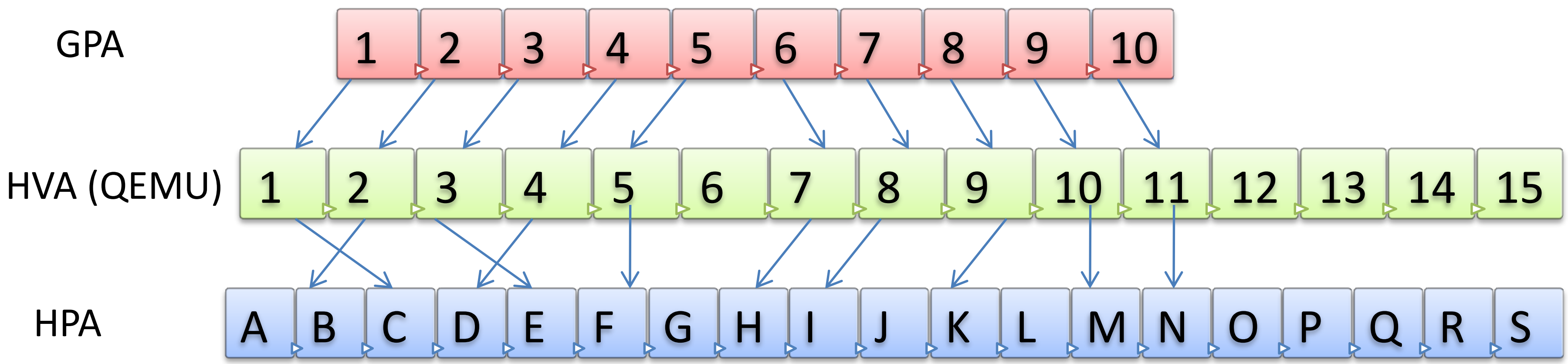
Guest process page table
GVA -> GPA

1	1
2	
3	

Step 4:

- When reexcutes the faulted instruction, MMU will walk two table in nested loop to figure out GVA -> HPA
- i.e. for every guest physical address encountered by MMU, EPT walk will be done to find GPA -> HPA

EPT/NPT Building



EPT/NTP page table
GPA -> HPA

1	**C**
2	
3	

Guest process page table
GVA -> GPA

1	**1**
2	
3	

Step 5:

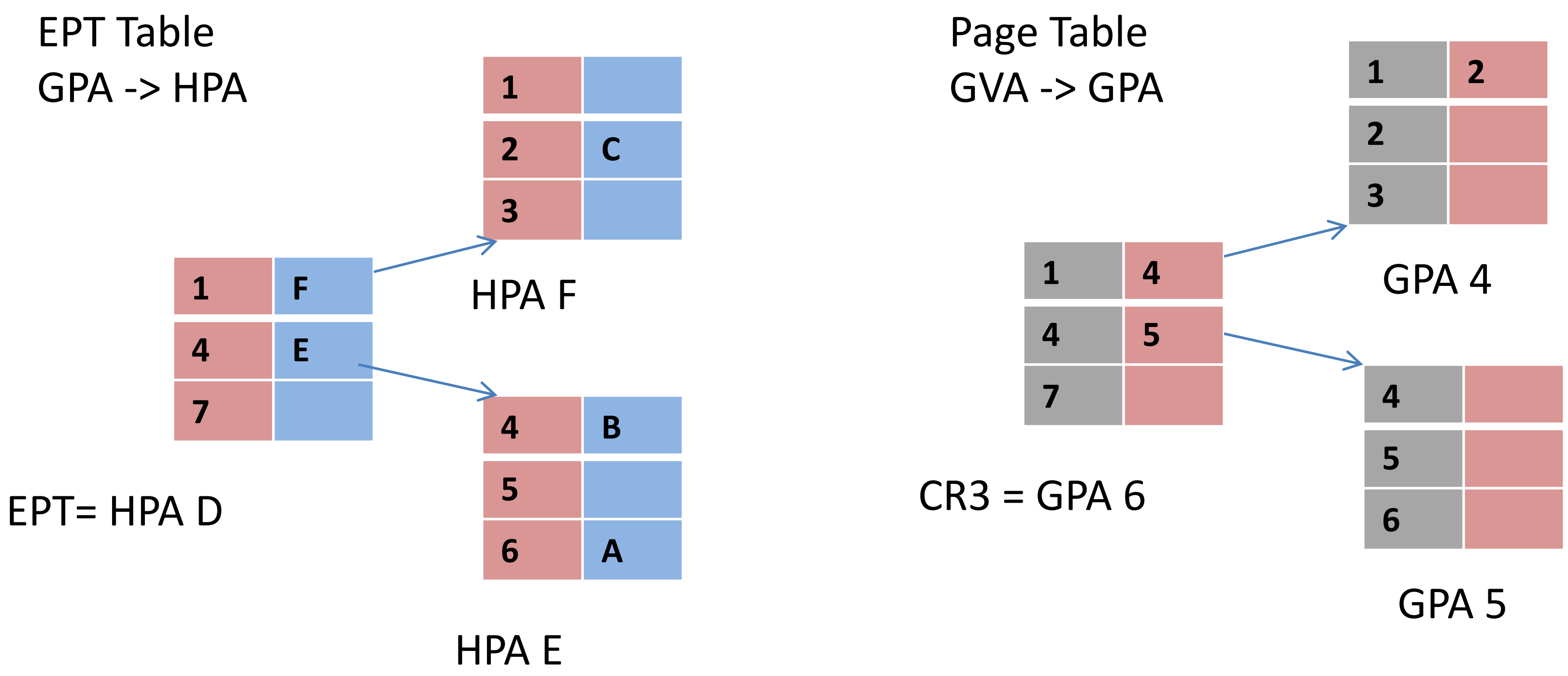
- Similarly every EPT table entry is filled after EPT violation for the corresponding GPA
- since EPT stores GPA -> HPA , the size of EPT table = guest RAM size

EPT/NPT (additional info)

- Advantages
  - No guest OS change is required
  - Any OS can be guest
  - Need not to trap page fault updates
  - Size of EPT table is proportional to guest memory size

- Disadvantages
  - TLB miss would cause considerable overhead in translation – Ex. One level page table would cause 3 page table memory access
  - For m level EPT and n level guest page table, EPT solution access mn + m + n page references
  - Hardware support required

EPT/NPT Scenario of TLB Miss



Resolve GVA 1 -> GPA 4 :
GPA 6 -> HPA A   requires access to HPA D, HPA E = 2
Read GVA 1 = GPA 4 from HPA A = 1
GPA 4 -> HPA B requires access to HPA D, HPA E = 2
Read GVA 1 = GPA 2 from HPA B = 1
GPA 2 -> HPA C requires access to HPA D, HPA F = 2
Total 8 access

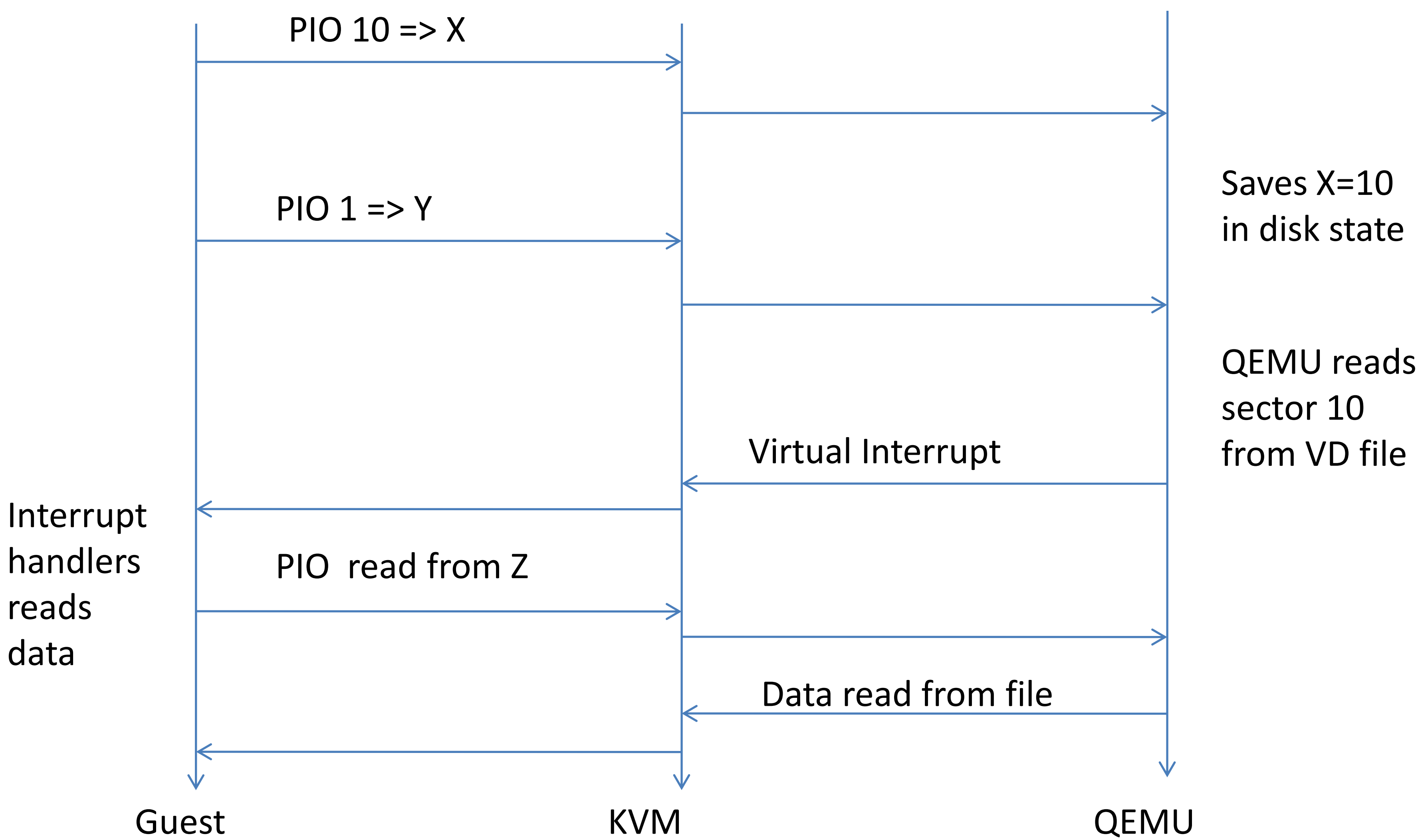
QEMU– IO device emulation

- Basic IO devices are emulated by QEMU
- Example - Keyboard, Mouse, Display, hard drive and NIC
- Device access from guest is trapped (both PIO and MMIO) by KVM
- KVM passes control to QEMU to handle IO
- QEMU injects interrupts from devices through KVM
- To emulate DMA, QEMU uses threads to do the IO

QEMU– IO device emulation- Example

- Assume disk drive having following interface
  - register x to specify sector number
  - register y to receive commands (1 read, 0 write)
  - register z to read/write data
- When guest wants to read sector number 10
  1. Guest does PIO 10 on register x
  2. QEMU saves this information in device state
  3. Guest issues read command using PIO 1 on register y
  4. QEMU maps sector 10 on virtual disk file and reads necessary content
  5. issues an interrupt
  6. Guest reads 512 bytes from register z using PIO
  7. QEMU gives the data it read from VD

QEMU– IO device emulation- Example



KVM + QEMU – Usage

- Prerequisites
  - Linux Distribution
  - Install QEMU packages `yum install qemu*` - in Fedora or rpm based
  - `apt-get install qemu*` - in Ubuntu or deb based
  - Ensure hardware support `grep vmx /proc/cpuinfo`
  - Download some ISO image from IITB FTP server
- Virtual disk (VD)
  - A file at host which acts as disk drive for virtual machine
  - VD can be a file or a raw partition
- Create VD
  - `qemu-img create –f raw disk1.img 10G`
  - Creates disk of 10G in raw format (sectors are directly mapped to file offset)

KVM + QEMU – Usage cont.

- Creating first VM
  - To boot from ISO `qemu-kvm –m 1G –hda disk1.img –cdrom F10-i686-Live.iso`
  - -m says size of RAM, –smp for number of processors
  - -hda primary hard disk, -cdrom for CD rom for guest
  - After this command, you will get the standard installation wizard running in guest. Easy !
  - Once installed on disk1.img, `qemu-kvm –m 1G disk1.img` will boot the guest from disk1.img directly
- QEMU+KVM = host user space process
  - Every virtual machine runs as user space process on the host
  - Can be monitored using standard Linux tools ps, top and kill etc
  - One thread for every CPU in the guest (use –smp option)

KVM + QEMU – Usage Cont.

- Copy-On-Write VD
  - COW disks – versioning / snapshots at disk levels
  - can choose any version without loosing consistency
  - Equivalent to disk level backups
  - Supported only on cow, qcow, qcow2
- Create COW disk
  - `qemu-img create -f qcow2 disk2.cow2 10G`
  - Install the VM
  - Take a snapshot `qemu-img snapshot -c s1 disk2.cow2`
  - Start the VM, create and delete few files inside the VM and shutdown
  - Take another snapshot s2
  - Now to rollback to s1, `qemu-img snapshot -a s1 disk2.cow`

References

1. Intel® 64 and IA-32 Architectures Software Developer's Manual
2. http://www.linux-kvm.org/page/Documents
3. http://wiki.qemu.org/Manual
4. Accelerating Two-Dimensional PageWalks for Virtualized Systems