Hardware Virtualization

Virtualizable Architectures

Non-virtualizable Architectures

- x86 architecture is non-virtualizable
- What is the fundamental assumption that must hold if trap and emulate virtualization is to work?

Virtualizable Architectures

- The fundamental assumption for trap and emulate virtualization is
  - All *sensitive* instructions can be trapped
  - Sensitive instructions = instructions that *must* be emulated
- *Behavior sensitive*: result of instruction depends upon privilege level
  - If guest OS executes these instructions at lower privilege levels, result will be wrong
- *Control sensitive*: results that could change processor privilege level or change processor state
  - Like modify the processor registers like page tables.
- *Formal Requirements for Virtualizable Third-Generation Architectures* by *Gerald Popek* and *Robert Goldberg, Comm. ACM, 1974*

Exercise 1 (5 minutes)

- Consider an imaginary OS that executes the following imaginary instructions
  - MASK ..... # turn off all interrupts before doing I/O
  - IO ..... # do I/O
- MASK instruction works if executed in privileged mode
- If executed at non-privileged level, fails silently
  - Will fail if executed by imaginary OS in VM

- Would the imaginary OS work if run in a VM?
- Is MASK a (i) control sensitive (ii) behavior sensitive instruction
- Is the imaginary architecture virtualizable?

Non-virtualizable Aspects of x86 Architecture

Pops a doubleword (POPFD) from the top of the stack (if the current operand-size attribute is 32) and stores the value in the EFLAGS register, or pops a word from the top of the stack (if the operand-size attribute is 16) and stores it in the lower 16 bits of the EFLAGS register (that is, the FLAGS register). These instructions reverse the operation of the PUSHF/PUSHFD instructions.

as privileged as the IOPL. If a POPF/POPFD instruction is executed with insufficient privilege, an exception does not occur but privileged bits do not change.

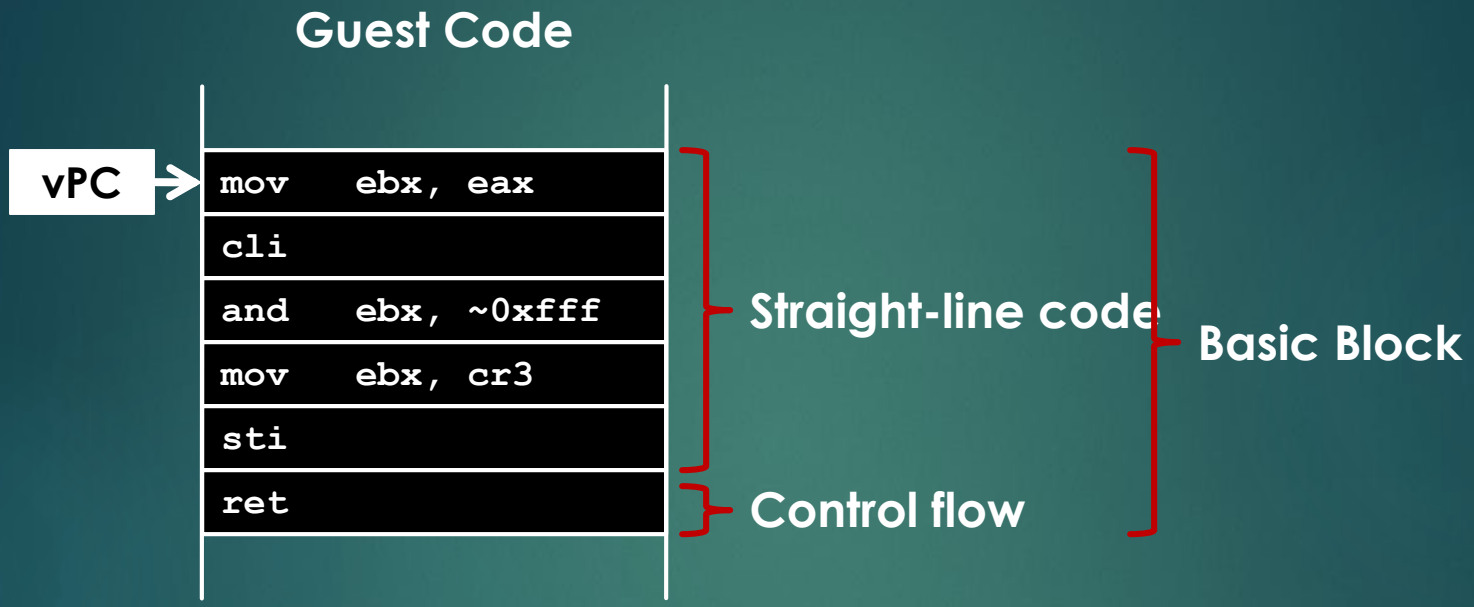
- *popf* instruction
- Fails silently if guest executes it in non-privileged mode
- Cannot be handled by Trap-and-Emulate virtualization
  - There are about 17 instructions in x86 that are non-virtualizable

Binary Translation

Binary Translation

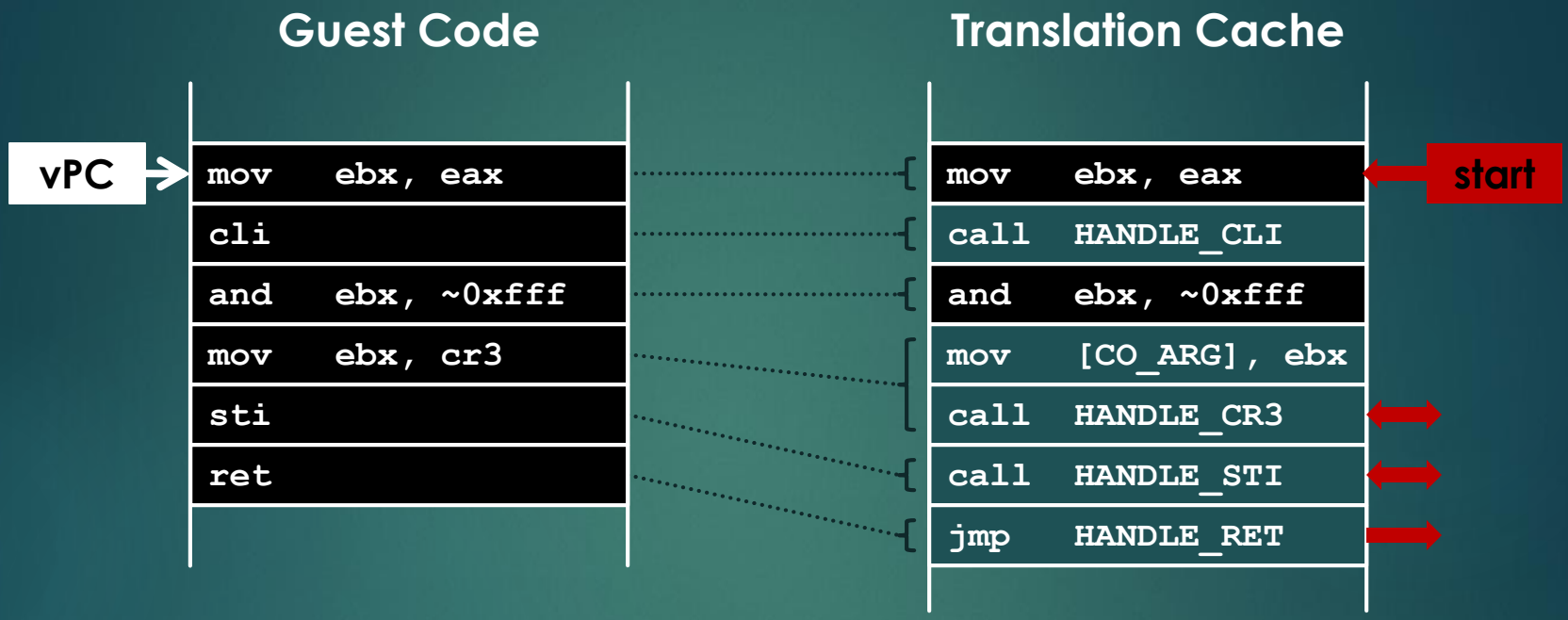
- Transparent virtualization technique
- Complements trap and emulate virtualization
  - Allows virtualization of x86 architecture
- Invented by VMWare
  - Mainframe architectures are virtualizable
- Conceptually simple
  - Replace *popf* instruction with trap to kernel and emulation
  - Replace simple instructions with equivalent instructions (details in paper)
  - Replacement is not done at run time; done before OS is run to create a modified OS image

What are Basic Blocks?



Source: https://labs.vmware.com/download/45, Scott Devine, VMWare

Binary Translation



Source: https://labs.vmware.com/download/45, Scott Devine, VMWare

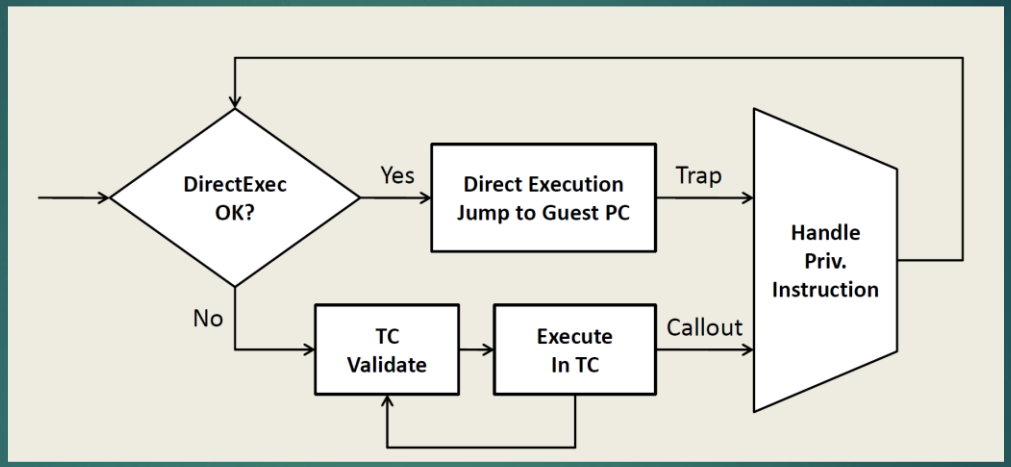
Binary Translation Complexities

- Control flow changes
  - Translation can change addresses of instructions
  - Branches have to be re-translated
  - Keep track of branch addresses

```
isPrime:  mov    %ecx, %edi ; %ecx = %edi (a)
          mov    %esi, $2   ; i = 2
          cmp    %esi, %ecx ; is i >= a?
          jge    prime      ; jump if yes
```

```
isPrime':   mov %ecx, %edi    ; IDENT
            mov %esi, $2
            cmp %esi, %ecx
            jge [takenAddr]   ; JCC
            jmp [fallthrAddr]
```

x86-32 workaround in VMWare Workstation



Picture from Sorav Bansal/Scott Devine

- **Idea**: Direct execution + Dynamic Binary translation
  - Direct execution of most user level instruction
  - Dynamic binary translation for most system code
  - Dynamically decide whether to use direct execution or

Workaround for x86-32: Paravirtualization

- Co-design of guest OS and hypervisor
  - Advantage: Simplicity, work around corner cases
  - Disadvantage: Need **modified** guest OS
  - Example: Xen hypervisor
- Trick to virtualize x86-32:
  - Block all 17 instructions that are sensitive but not privileged
    - Just modify the guest OS to #undef them
  - Instead provide ***hypercalls*** from guest OS to VMM
    - Hypercalls are like system calls but from guest OS to VMM

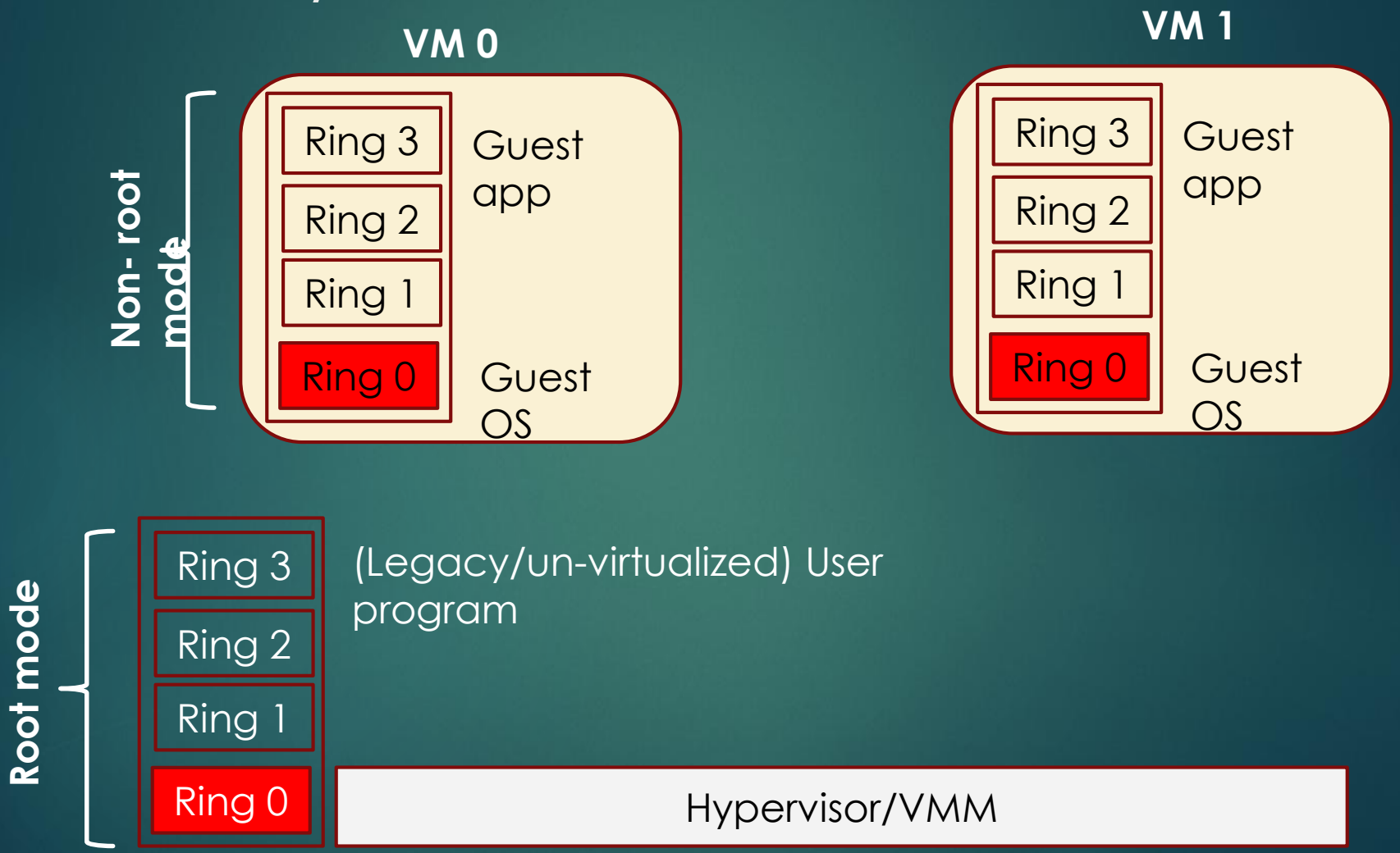
Slide: Arkaprava Basu

Hardware assisted virtualization: Intel VT-x/AMD-v for x86-64

- Solution idea:
  - Two *new* modes of operation: root and non-root
    - Each mode has complete set of execution rings (0-3)
    - New instructions to switch between modes
  - H/W state is duplicated for each operation mode
  - Hypervisor runs in root mode and VMs in non-root mode
  - When any "sensitive" instruction executed in non-root mode it either (1) executed by processor on duplicated state or (2) trap to hypervisor

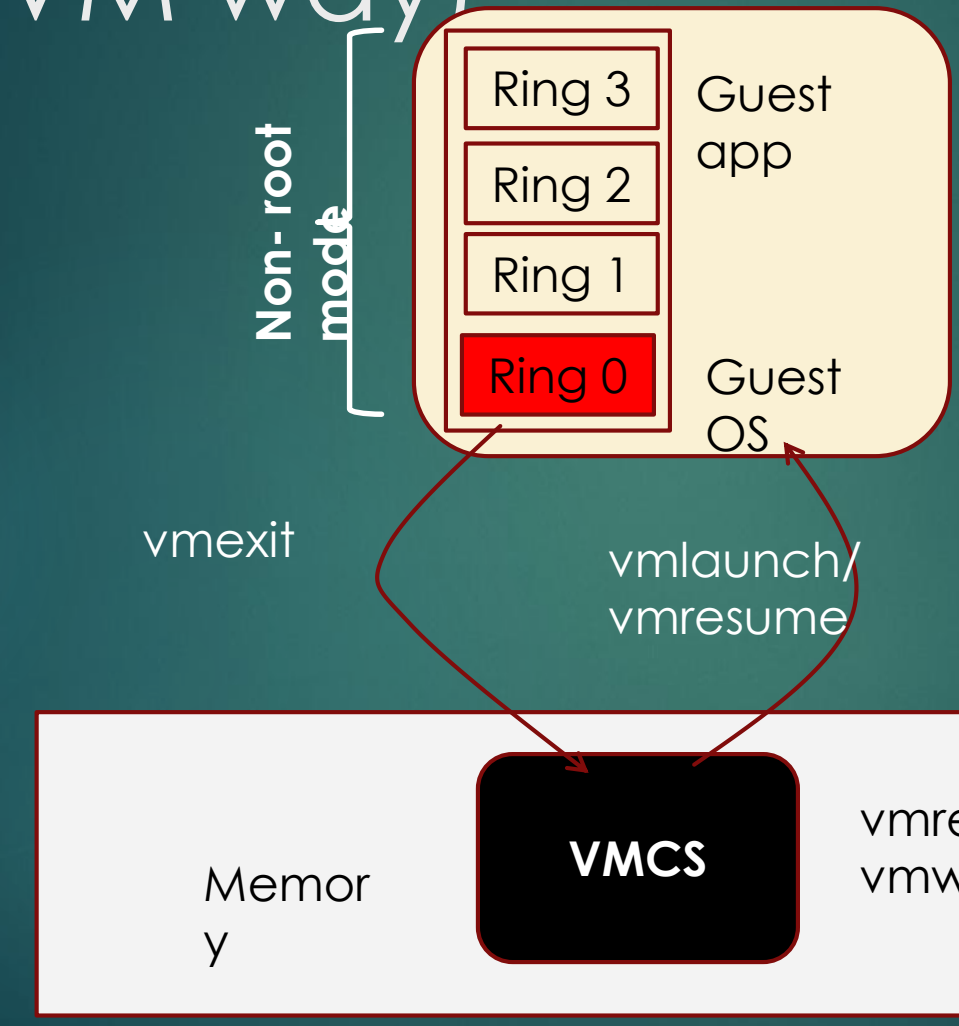
Slide: Arkaprava Basu

Hardware assisted virtualization: Intel VT-x/AMD-v for x86-64



Slide: Arkaprava Basu

Using Intel VT-x/AMD-v (KVM way)



(1) VMM initializes/modifies VMCS
(2) VMM executes vmlaunch/vmres ume instr.
(3) The VM runs until it hits a vmexit condition
(4) VMM analyzes and takes action on vmexit

Slide: Arkaprava Basu

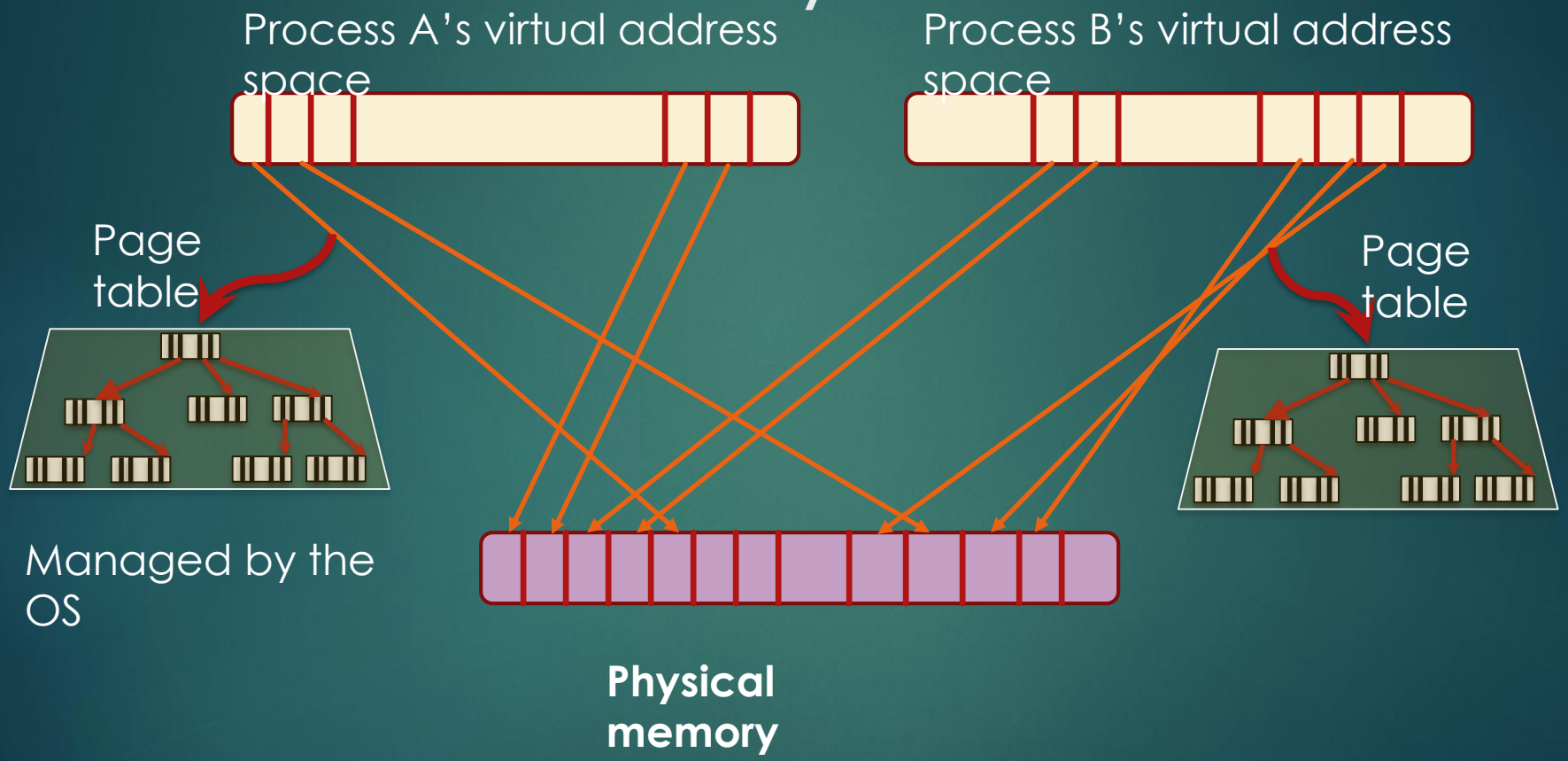
Current status of H/W assist

- Almost all VMMs make use of h/w assist today
- KVM is integrated part of Linux and makes heavy use of h/w assist
- Even Xen and VMWare workstation uses it

Slide: Arkaprava Basu

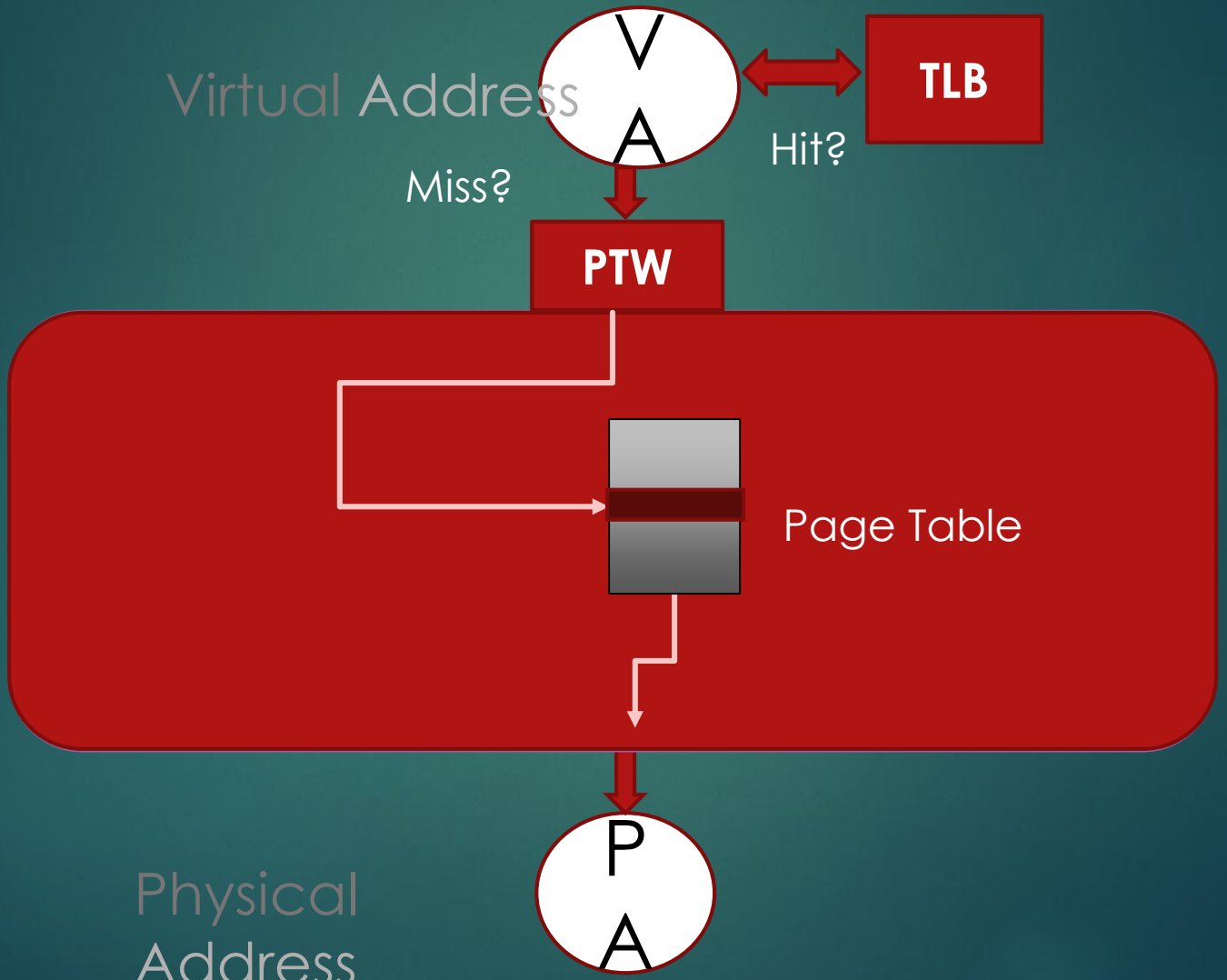
Memory Virtualization

Recap: Virtual Memory in un-virtualized system



Slide: Arkaprava Basu

Recap: Performing address translation in unvirtualized machine



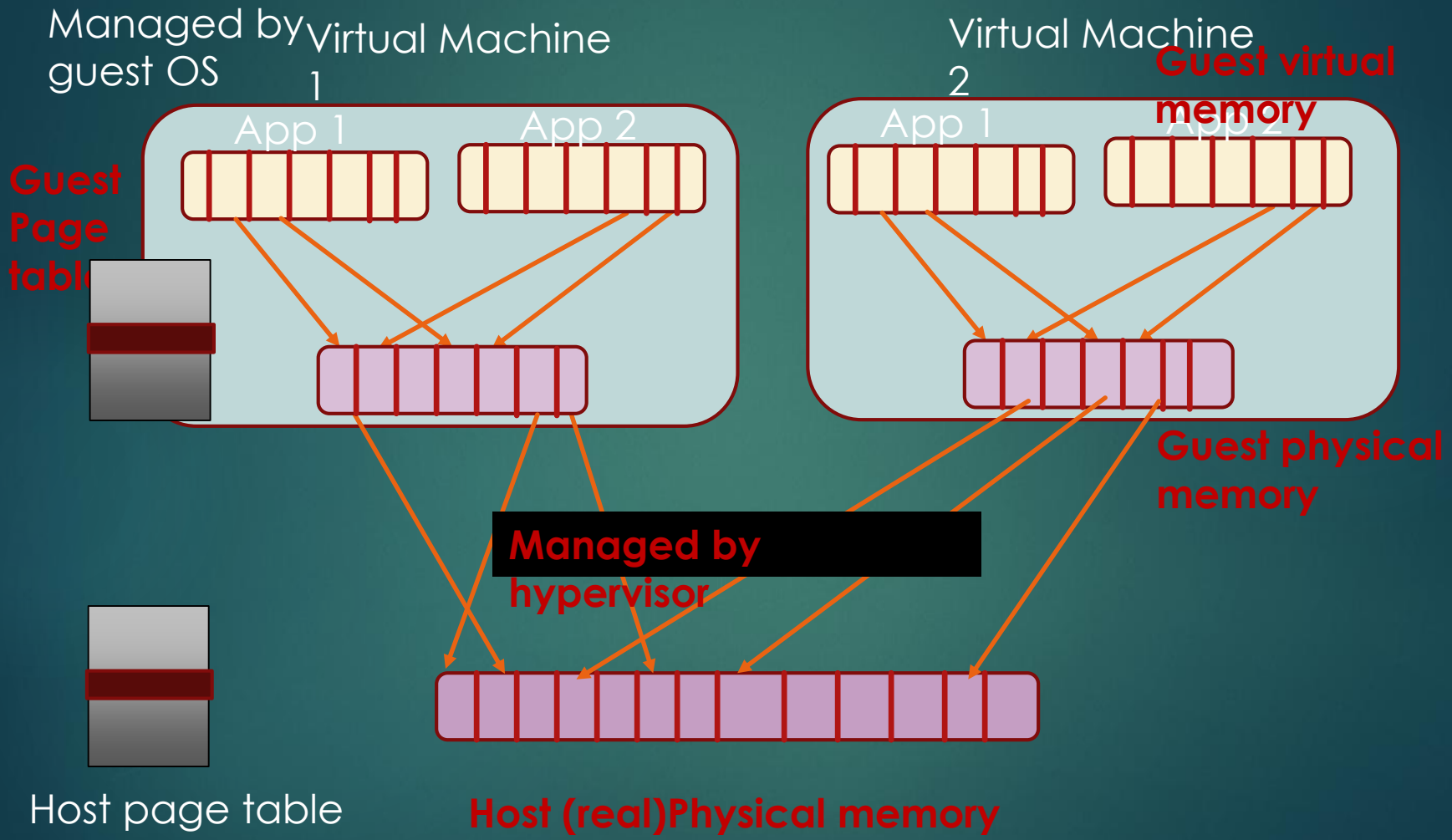
Slide: Arkaprava Basu

Requirement for memory virtualization

- **No** direct access to physical memory from VM
- **Only** hypervisor should manage physical memory
  - Why?
- But, fool the guest OS to think that it is accessing physical memory

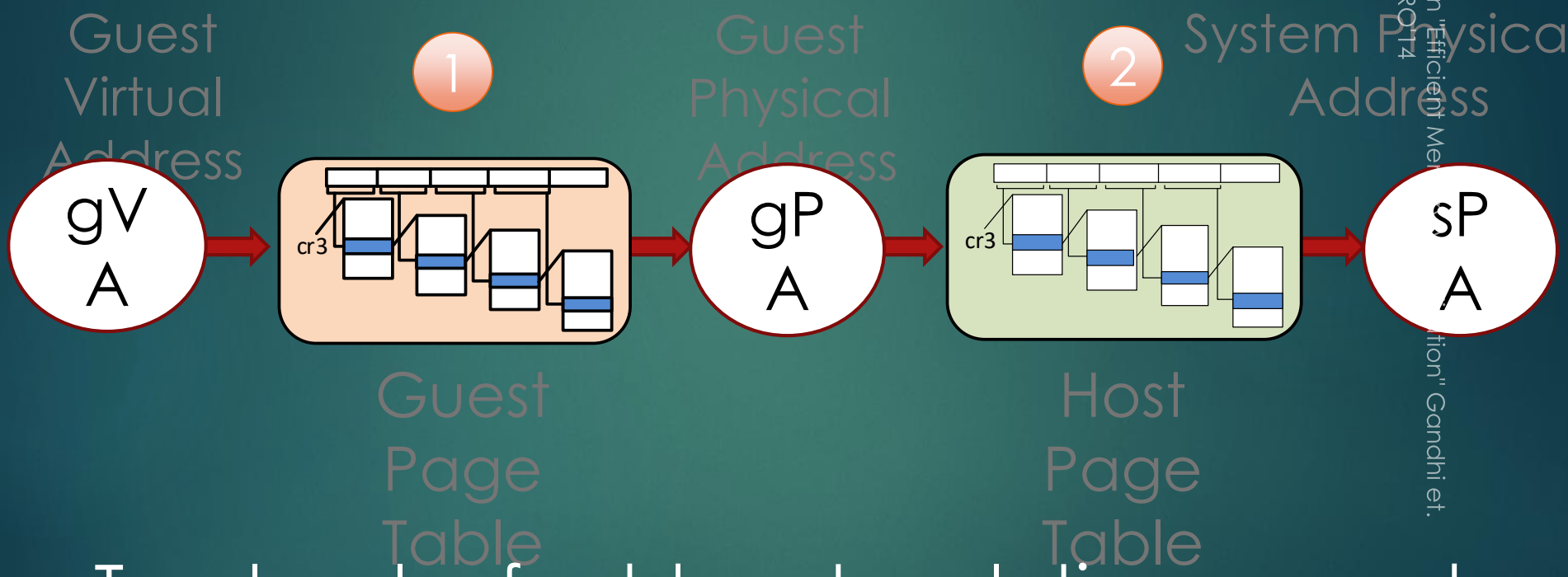
Slide: Arkaprava Basu

Virtualizing Virtual Memory



Slide: Arkaprava Basu

Virtualizing Virtual Memory



Based on "Efficient Memory Virtualization" Gandhi et. al., MICRO'14

Two levels of address translation on each memory access by an application running inside a VM

Implementing two level address translation

1

- Shadow page table
  - Software only technique

2

- Nested/Extended page table
  - Hardware support

Slide: Arkaprava Basu

Shadow page table

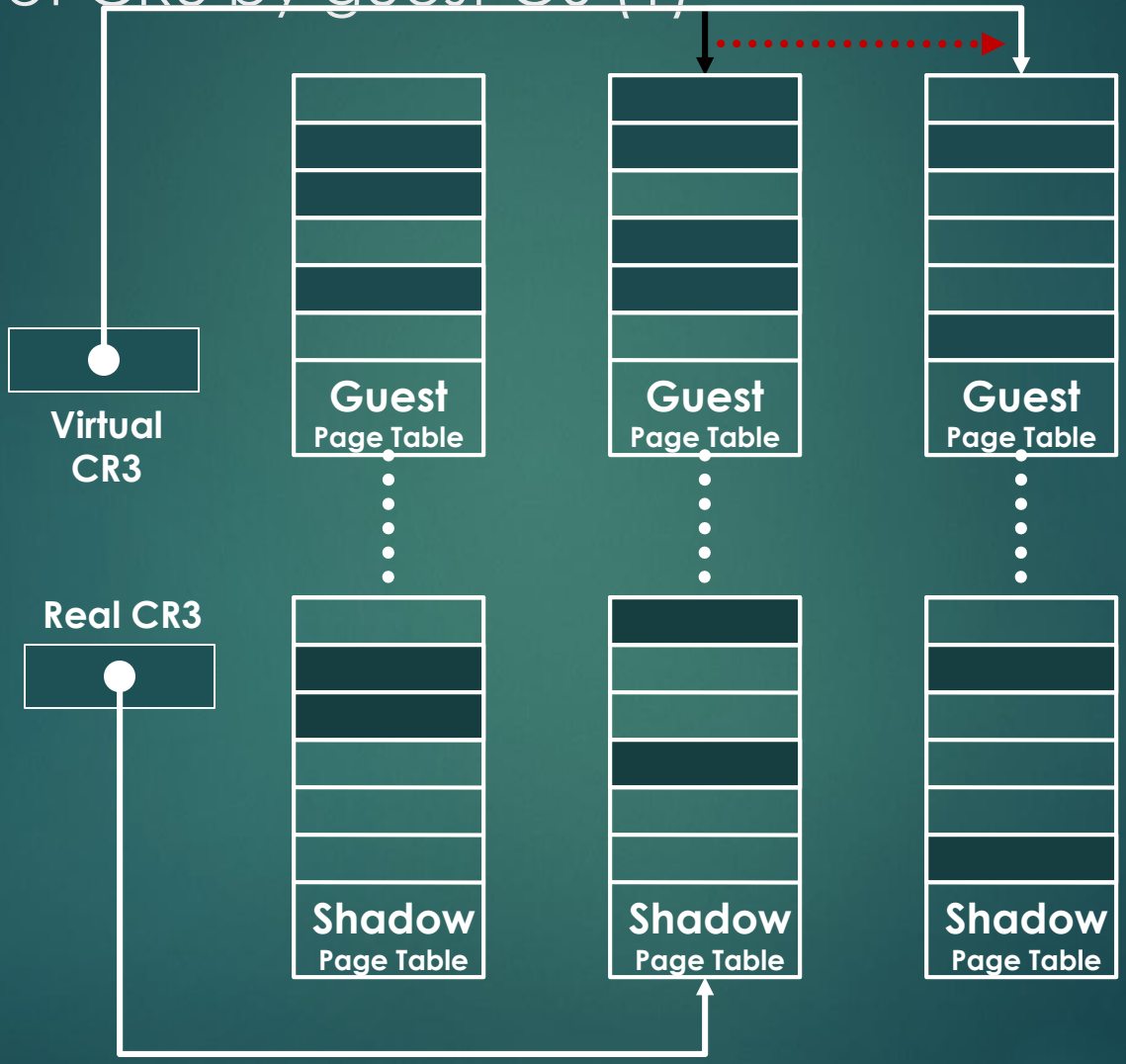
- **Idea**: Let hypervisor "create" a shadow page table that maps guest VA to host PA directly
  - Made by combining guest page table w/ system page table
  - Hypervisor makes the *cr3* point to the shadow page table

Slide: Arkaprava Basu

Shadow Page Tables

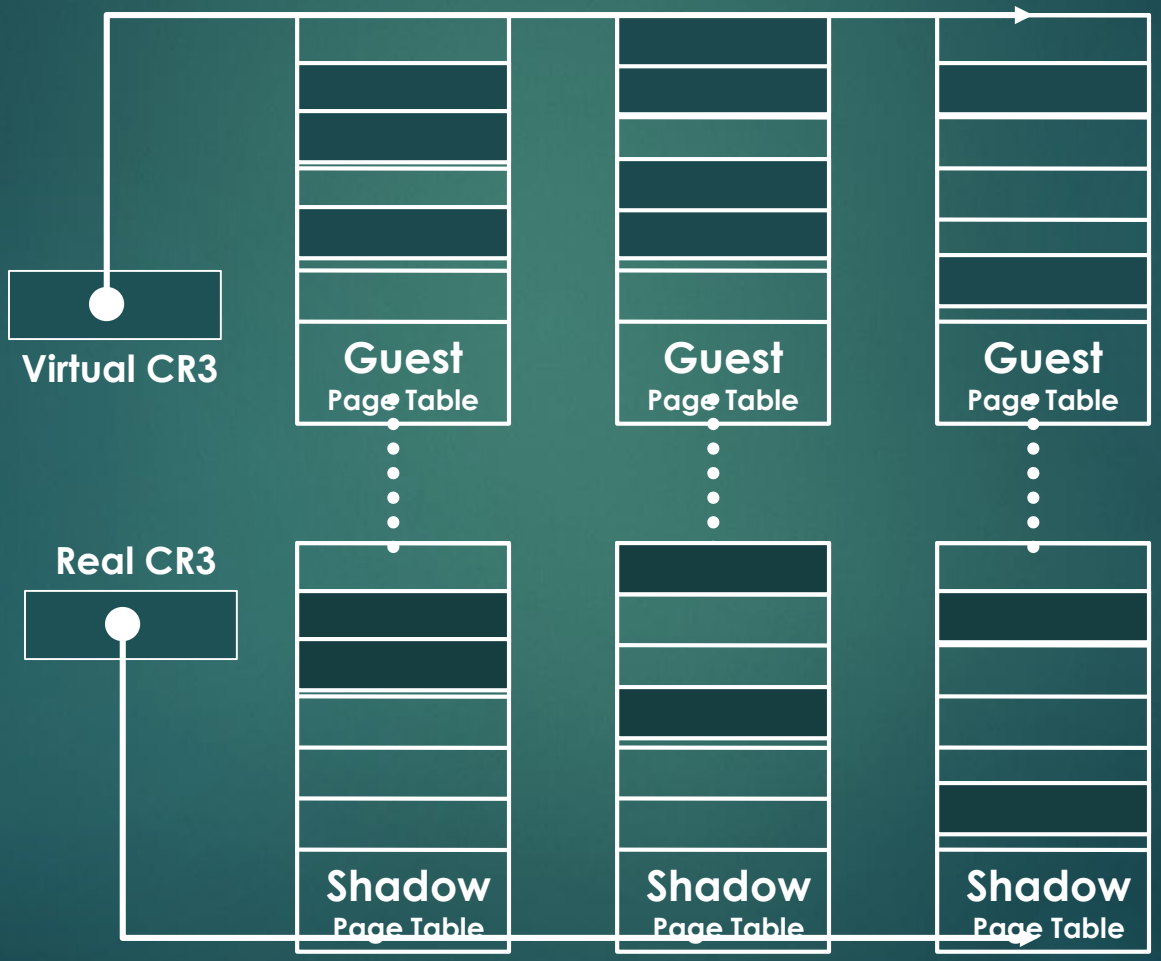
- VMM maintains shadow page tables that map guest-virtual pages directly to machine pages.
- Guest modifications to V->P tables synced to VMM V->M shadow page tables.
  - Guest OS page tables marked as read-only.
  - Modifications of page tables by guest OS -> trapped to VMM.
  - Shadow page tables synced to the guest OS tables

Set CR3 by guest OS (1)



Source: [6]

Set CR3 by guest OS (2)



Source: [6]

Drawbacks: Shadow Page Tables

- Maintaining consistency between guest page tables and shadow page tables leads to an overhead: VMM traps
- Loss of performance due to TLB flush on every "world-switch".
- Memory overhead due to shadow copying of guest page tables.

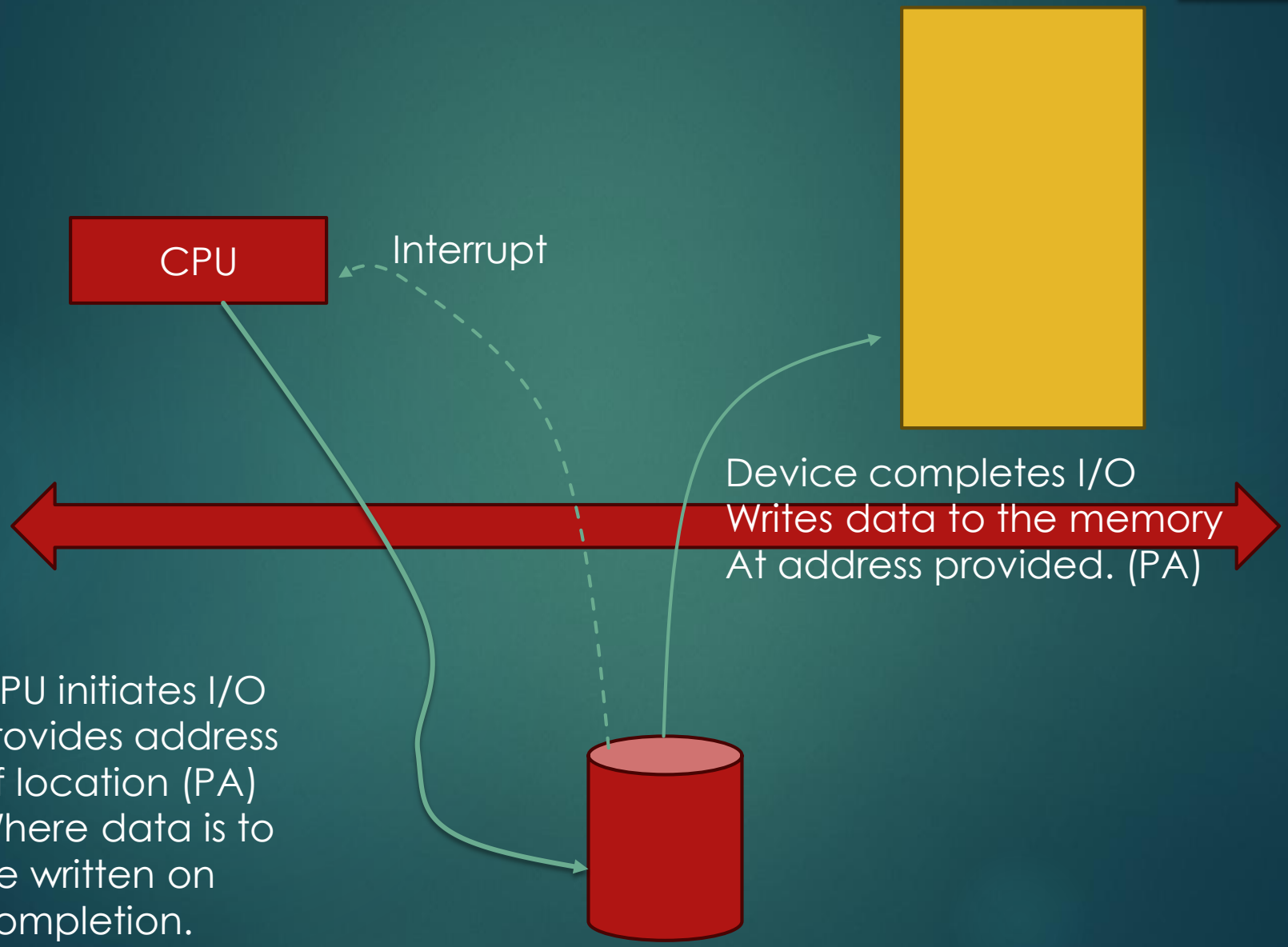
Review

- True or false: x86 architecture is virtualizable
- What is binary translation?

- False
- Replacement of sensitive instructions by traps

I/O Virtualization

I/O transfers on Baremetal

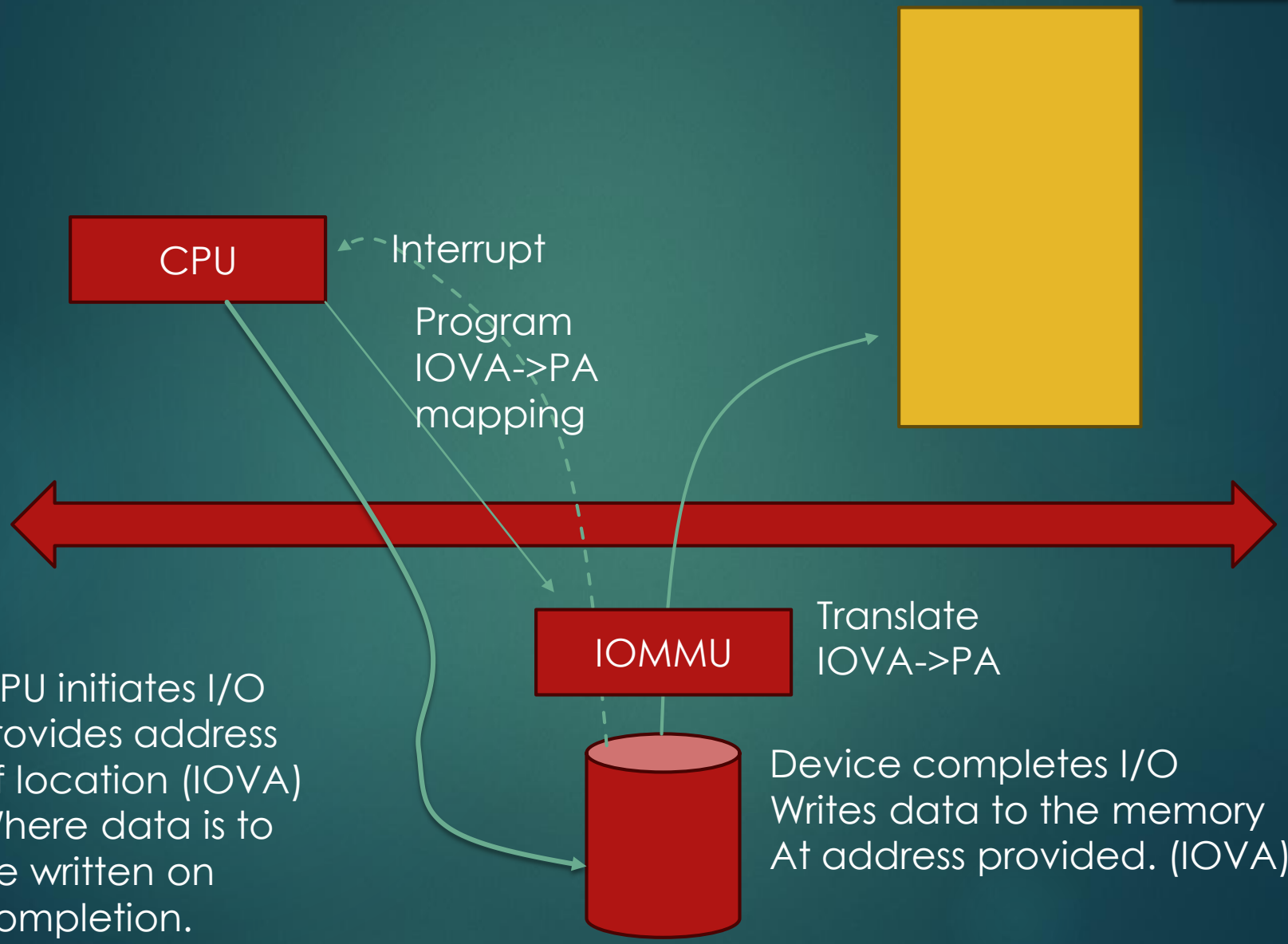


Physical address I/O: Challenges

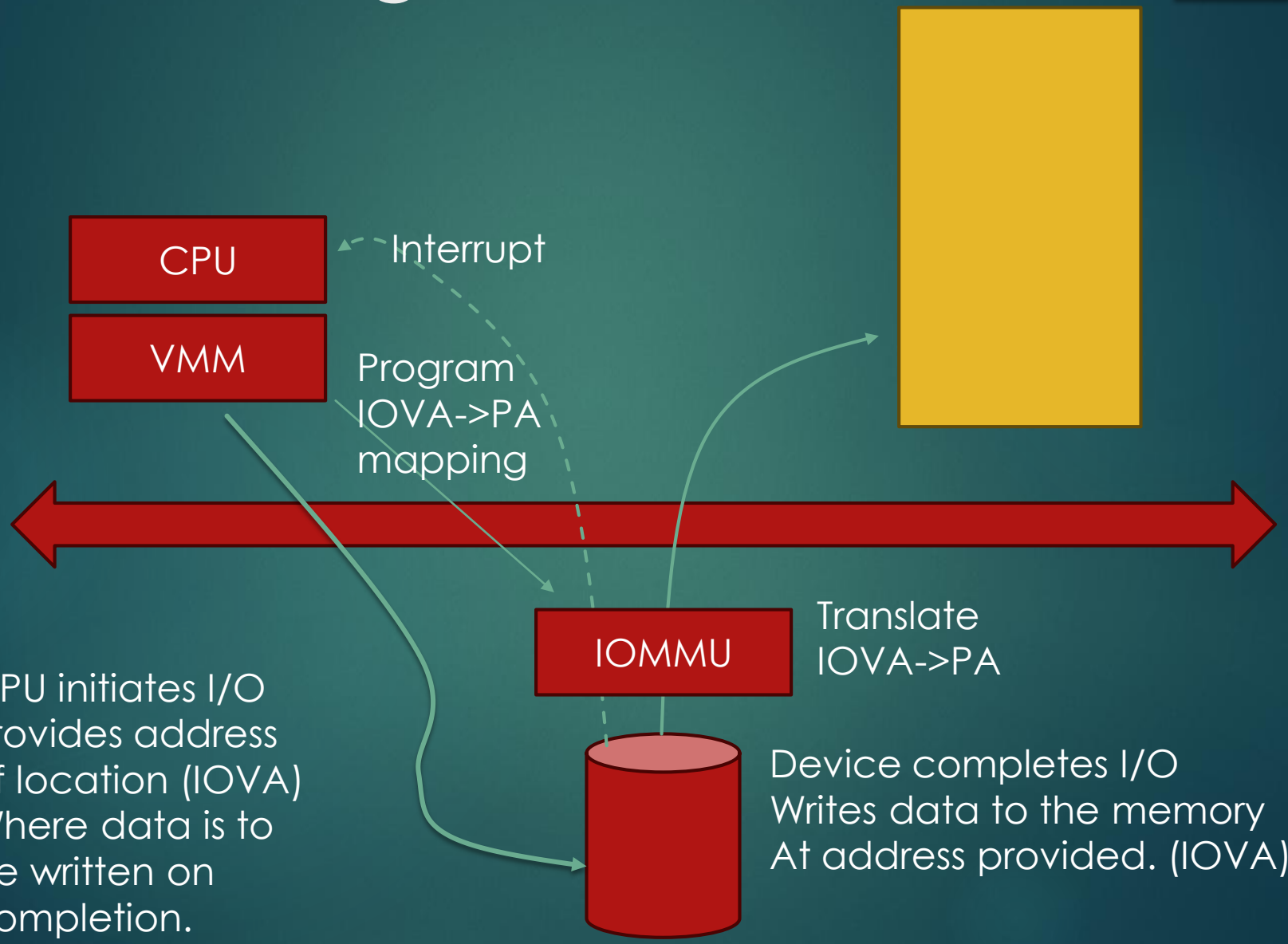
- Can't swap the page out
- External device can write into any memory location – Security concerns

- How will you solve this problem?

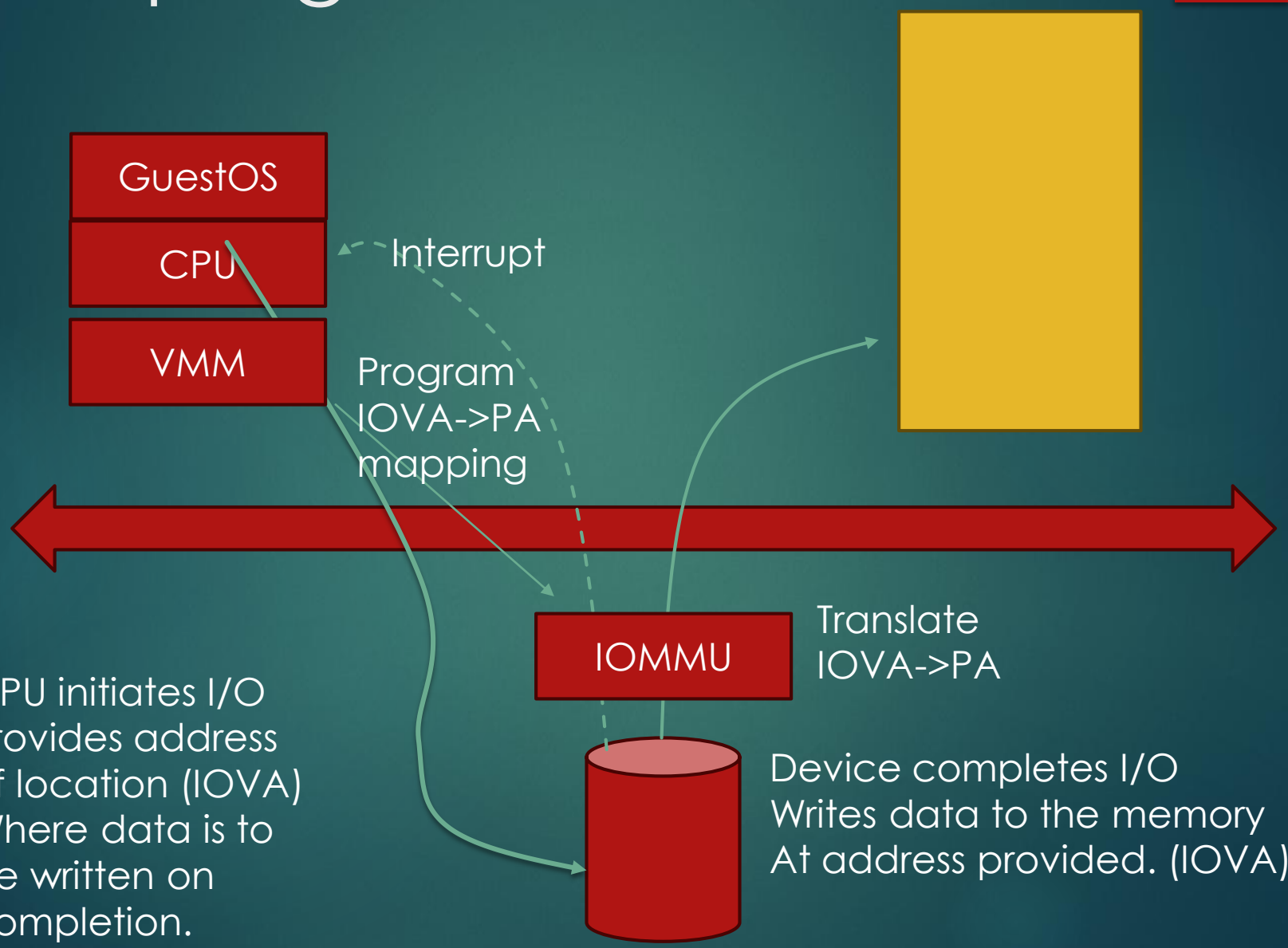
I/O transfers on Baremetal



VMM Programs IOMMU

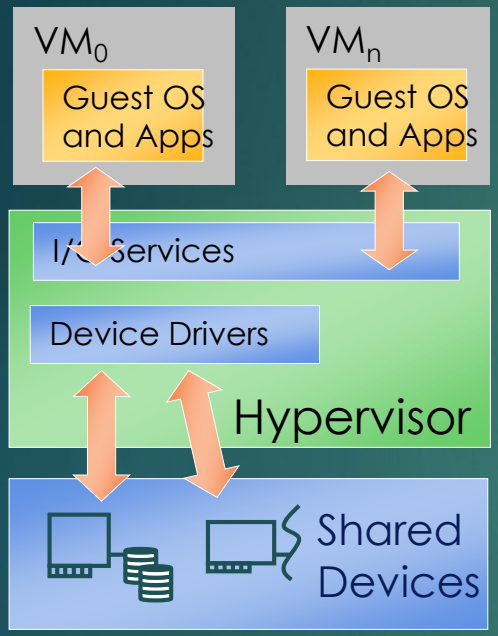


VM programs IOMMU



Options For I/O Virtualization

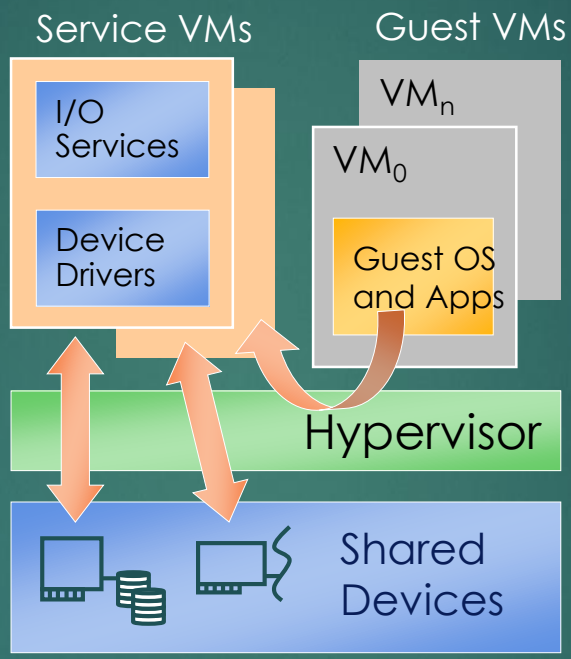
Monolithic Model



- Pro: Higher Performance
- Pro: I/O Device Sharing
- Pro: VM Migration
- Con: Larger Hypervisor

VMWare ESX

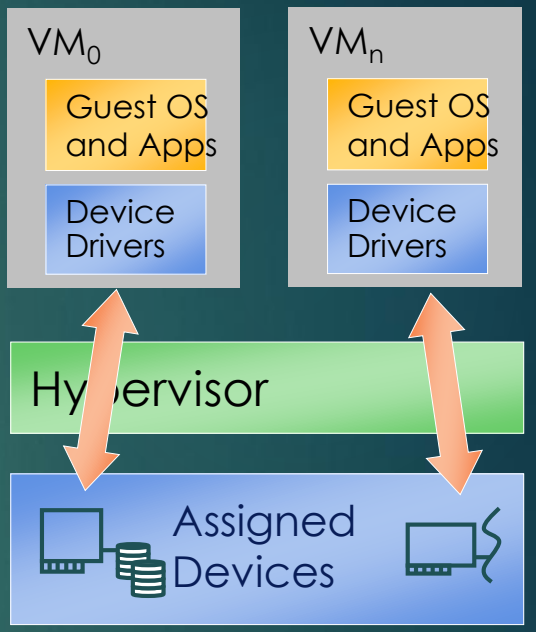
Service VM Model



- Pro: High Security
- Pro: I/O Device Sharing
- Pro: VM Migration
- Con: Lower Performance

Xen

Pass-through Model



- Pro: Highest Performance
- Pro: Smaller Hypervisor
- Pro: Device assisted sharing
- Con: Migration Challenges

Shared Device can be file, disk or disk partition