Advanced Operating Systems (CS 202)

Virtualization

Virtualization

- One of the natural consequences of the extensibility research we discussed

- What is virtualization and what are the benefits?

Virtualization motivation

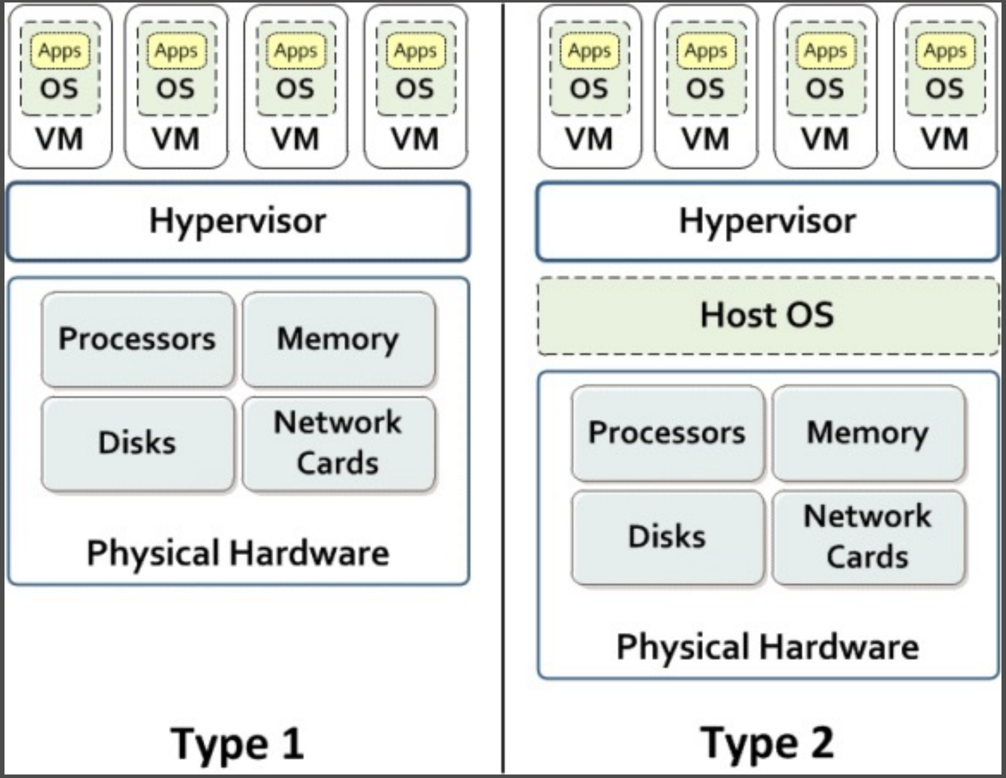
- Cost: multiplex multiple virtual machines on one hardware machine
  - Cloud computing, data center virtualization
  - Why not processes?
  - Why not containers?
- Heterogeneity:
  - Allow one machine to support multiple OS's
  - Maintaining compatibility
- Other: security, migration, energy optimization, customization, …

How do we virtualize?

- Create an operating system to multiplex resources among operating systems!
  - Exports a virtual machine to the Operating systems
  - Called a hypervisor or Virtual Machine Monitor (VMM)

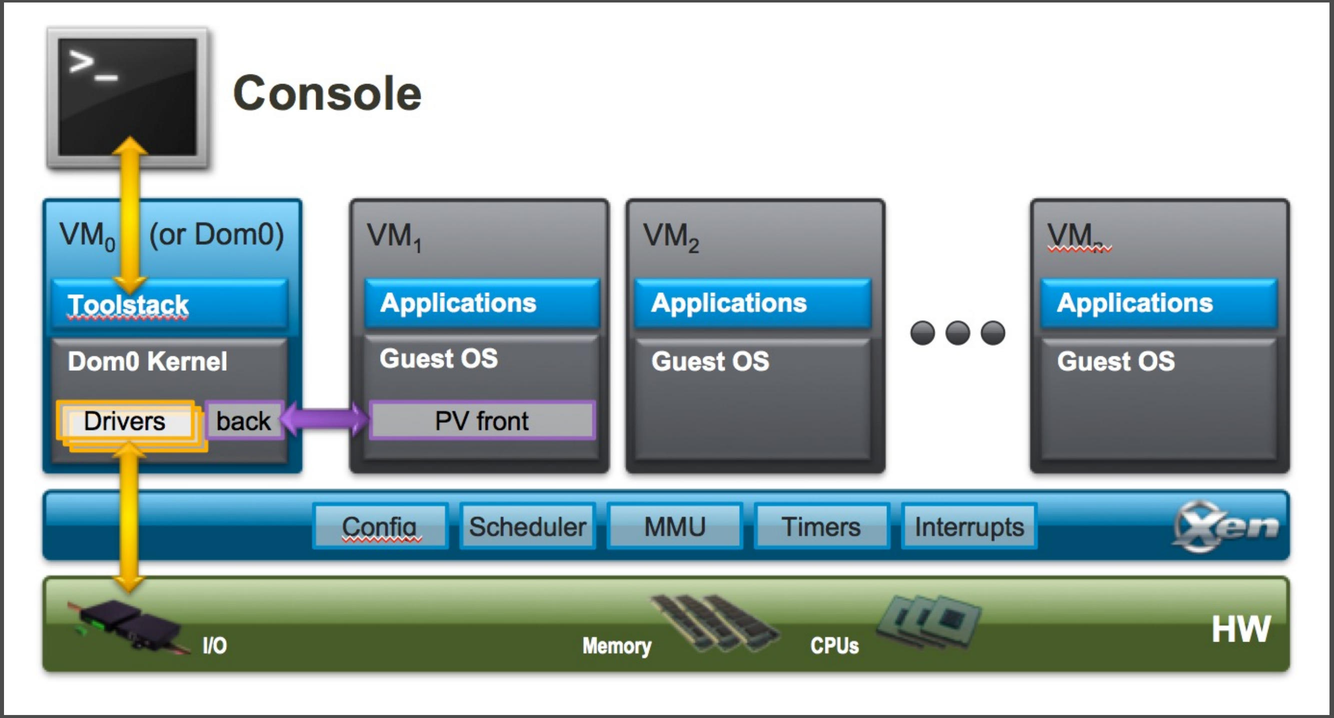
VIRTUALIZATION MODELS

Two types of hypervisors



- Type 1: Native (bare metal)
  - Hypervisor runs on top of the bare metal machine
  - e.g., KVM
- Type 2: Hosted
  - Hypervisor is an emulator
  - e.g., VMWare, virtual box, QEMU

Hybrid organizations



- Some hybrids exist, e.g., Xen
  - Mostly bare metal
  - VM0/Dom0 to keep device drivers out of VMM

Stepping back – some history

- IBM VM 370 (1970s)
- Microkernels (late 80s/90s)
- Extensibility (90s)
- SIMOS (late 90s)
  - Eventually became VMWare (2000)
- Xen, Vmware, others (2000s)
- Ubiquitous use, Cloud computing, data centers, …
  - Makes computing a utility

Full virtualization

- Idea: run guest operating systems unmodified
- However, supervisor is the real privileged software
- When OS executes privileged instruction, trap to hypervisor who executes it for the OS
- This can be very expensive
- Also, subject to quirks of the architecture
  - Example, x86 fails silently if some privileged instructions execute without privilege
  - E.g., popf

Example: Disable Interrupts

- Guest OS tries to disable interrupts
  - the instruction is trapped by the VMM which makes a note that interrupts are disabled for that virtual machine

- Interrupts arrive for that machine
  - Buffered at the VMM layer until the guest OS enables interrupts.

- Other interrupts are directed to VMs that have not disabled them

Binary translation--making full virtualization practical

- Use binary translation to modify OS to rewrite silent failure instructions
- More aggressive translation can be used
  - Translate OS mode instructions to equivalent VMM instructions
    - Some operations still expensive
    - Cache for future use
    - Used by VMWare ESXi and Microsoft Virtual Server
- Performance on x86 typically ~80-95% of native

Binary Translation Example

Guest OS Assembly

```
do_atomic_operation:
    cli
    mov eax, 1
    xchg eax, [lock_addr]
    test eax, eax
    jnz spinlock
    …
    …
    mov [lock_addr], 0
    sti
    ret
```

Translated Assembly

```
do_atomic_operation:
    call [vmm_disable_interrupts]
    mov eax, 1
    xchg eax, [lock_addr]
    test eax, eax
    jnz spinlock
    …
    …
    mov [lock_addr], 0
    call [vmm_enable_interrupts]
    ret
```

Paravirtualization

- Modify the OS to make it aware of the hypervisor
  - Can avoid the tricky features
  - Aware of the fact it is virtualized
    - Can implement optimizations
- Comparison to binary translation?
- Amount of code change?
  - 1.36% of Linux, 0.04% for Windows

Hardware supported virtualization (Intel VT-x, AMD-V)

- Hardware support for virtualization
- Makes implementing VMMs much simpler
- Streamlines communication between VM and OS
- Removes the need for paravirtualization/binary translation
- EPT: Support for shadow page tables
- More later…

NUTS AND BOLTS

What needs to be done?

- Virtualize hardware
  - Memory hierarchy
  - CPUs
  - Devices
- Implement data and control transfer between guests and hypervisor
- We'll cover this by example – Xen paper
  - Slides modified from presentation by Jianmin Chen

Xen

- Design principles:
  - Unmodified applications: essential
  - Full-blown multi-task O/Ss: essential
  - Paravirtualization: necessary for performance and isolation

Xen

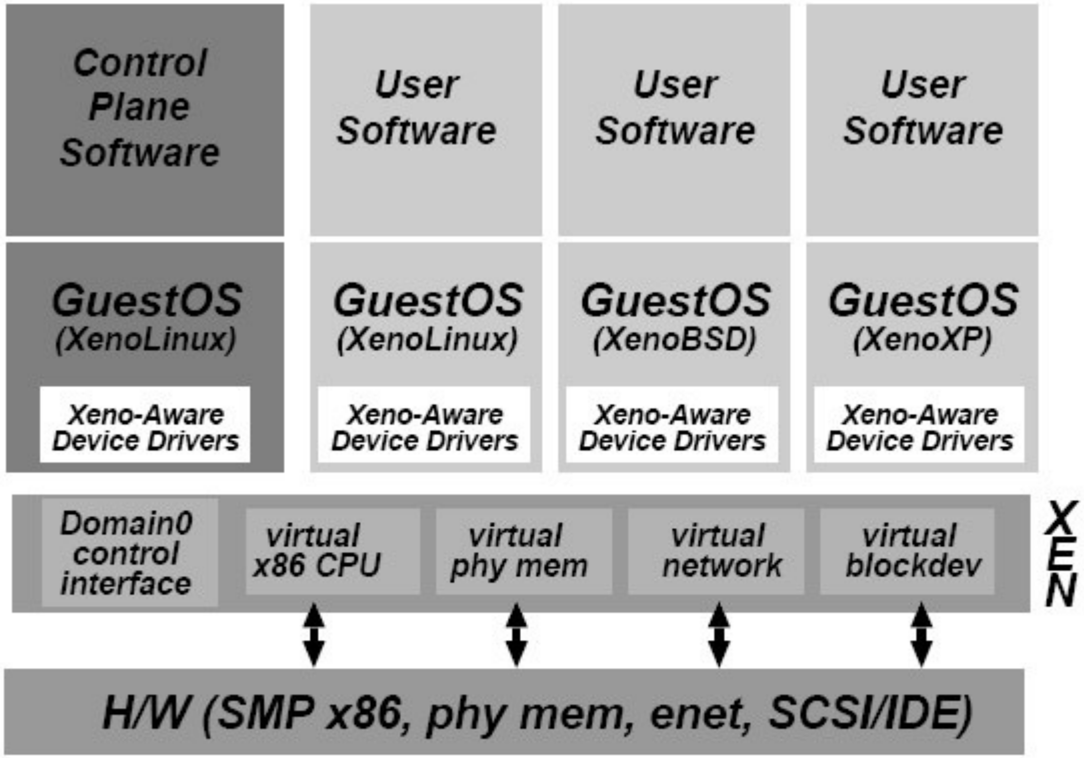
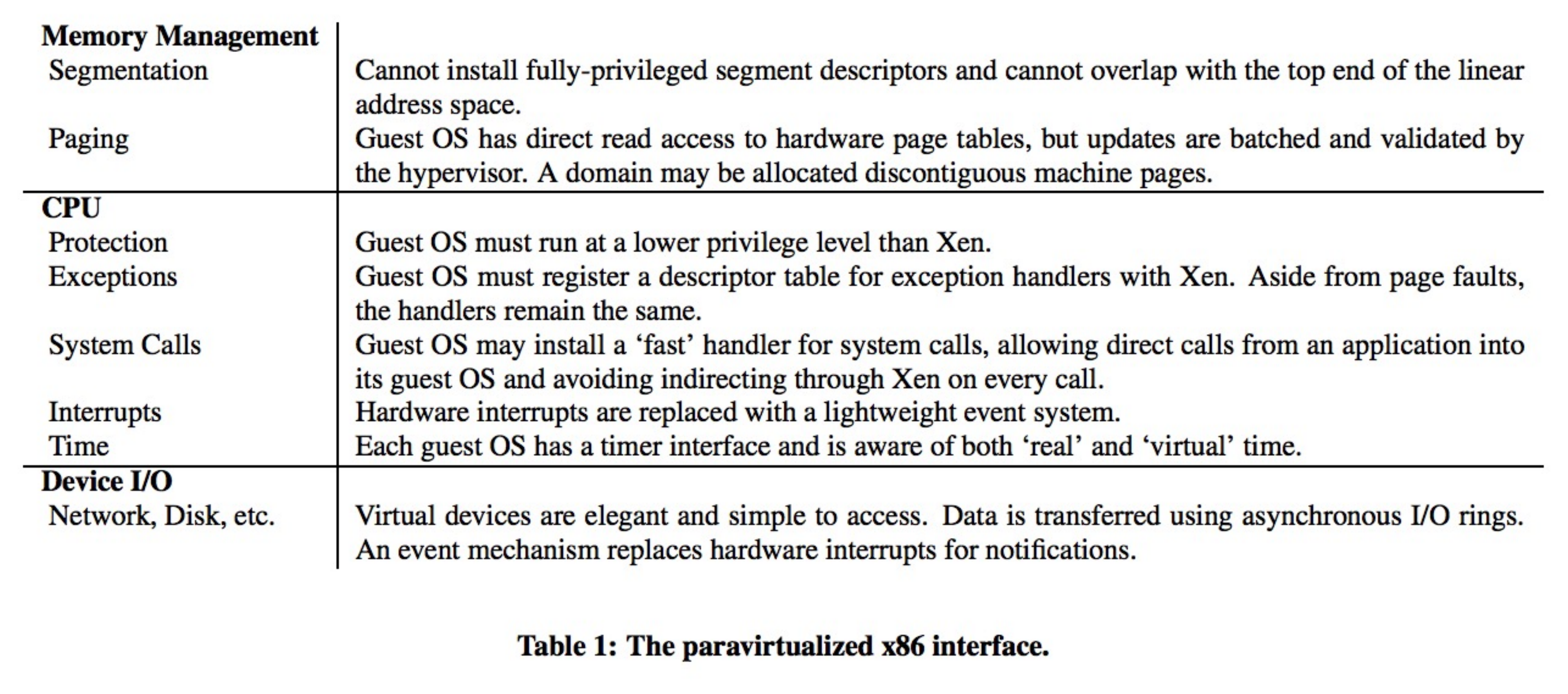


Figure 1: The structure of a machine running the Xen hypervisor, hosting a number of different guest operating systems, including *Domain0* running control software in a XenoLinux environment.

Implementation summary



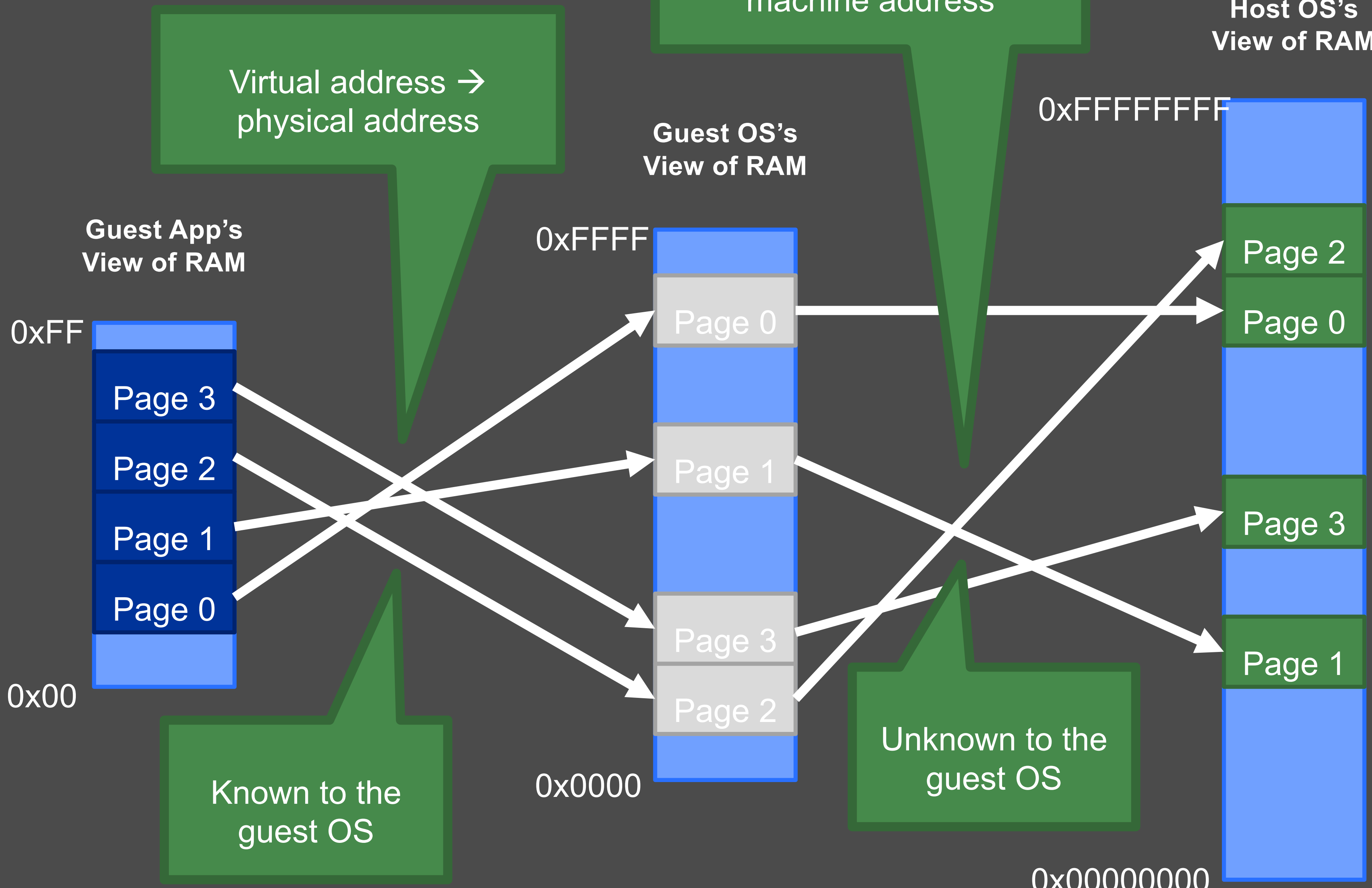
Memory Management	
Segmentation	Cannot install fully-privileged segment descriptors and cannot overlap with the top end of the linear address space.
Paging	Guest OS has direct read access to hardware page tables, but updates are batched and validated by the hypervisor. A domain may be allocated discontiguous machine pages.
CPU	
Protection	Guest OS must run at a lower privilege level than Xen.
Exceptions	Guest OS must register a descriptor table for exception handlers with Xen. Aside from page faults, the handlers remain the same.
System Calls	Guest OS may install a 'fast' handler for system calls, allowing direct calls from an application into its guest OS and avoiding indirecting through Xen on every call.
Interrupts	Hardware interrupts are replaced with a lightweight event system.
Time	Each guest OS has a timer interface and is aware of both 'real' and 'virtual' time.
Device I/O	
Network, Disk, etc.	Virtual devices are elegant and simple to access. Data is transferred using asynchronous I/O rings. An event mechanism replaces hardware interrupts for notifications.

Table 1: The paravirtualized x86 interface.

Xen VM interface: Memory

- Memory management
  - Guest cannot install highest privilege level segment descriptors; top end of linear address space is not accessible
  - Guest has direct (not trapped) read access to hardware page tables; writes are trapped and handled by the VMM
  - Physical memory presented to guest is not necessarily contiguous

Two Layers of mory



Physical address → machine address

Virtual address → physical address

Known to the guest OS

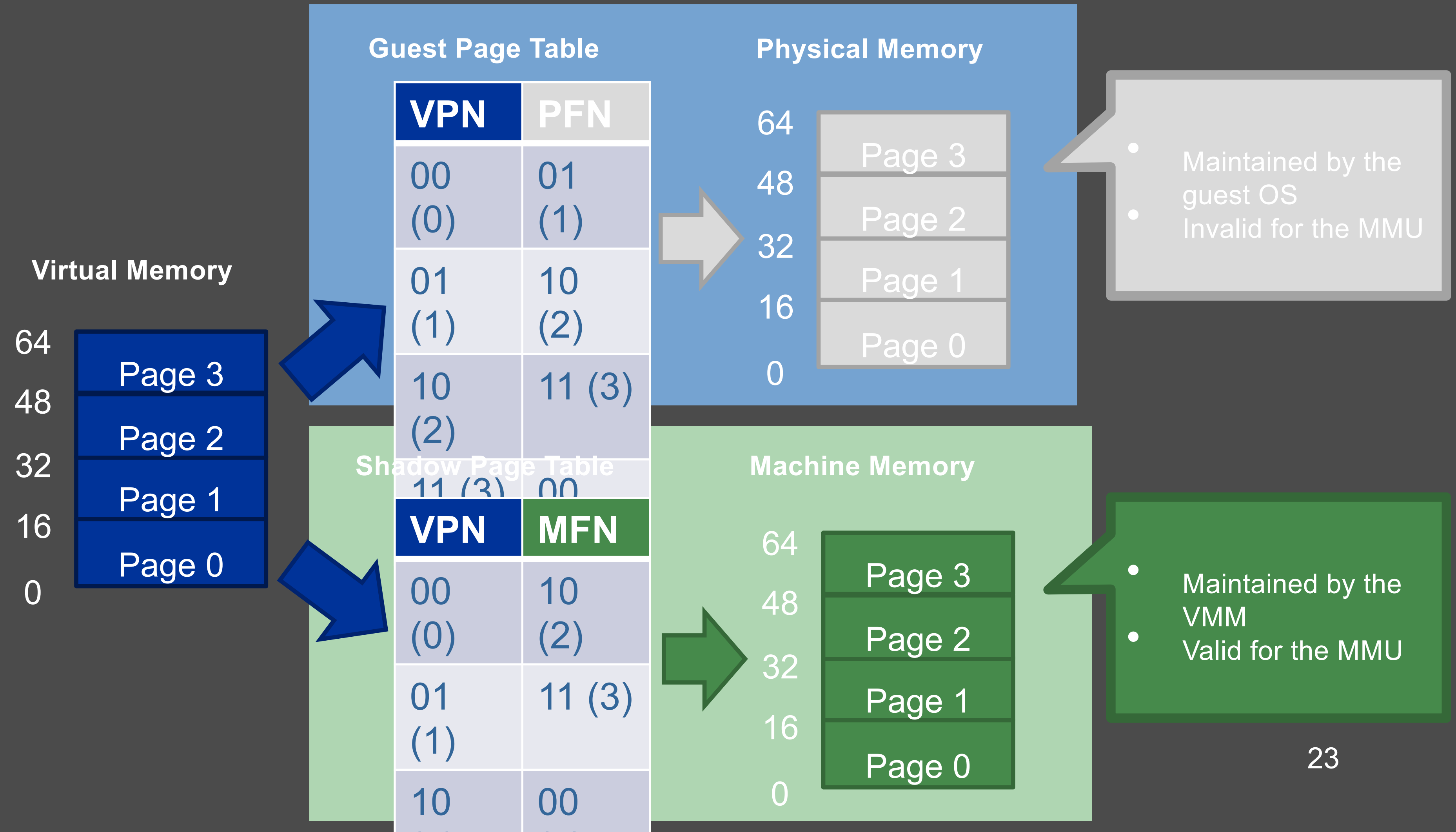
Unknown to the guest OS

Guest's Page Tables Are Invalid

- Guest OS page tables map virtual page numbers (VPNs) to physical frame numbers (PFNs)
- Problem: the guest is virtualized, doesn't actually know the true PFNs
  - The true location is the machine frame number (MFN)
  - MFNs are known to the VMM and the host OS
- Guest page tables cannot be installed in *cr3*
  - Map VPNs to PFNs, but the PFNs are incorrect
- How can the MMU translate addresses used by the guest (VPNs) to MFNs?

Shadow Page Tables

- Solution: VMM creates shadow page tables that map VPN → MFN (as opposed to VPN→PFN)



Virtual Memory

64
Page 3
48
Page 2
32
Page 1
16
Page 0
0

Guest Page Table

VPN	PFN
00 (0)	01 (1)
01 (1)	10 (2)
10 (2)	11 (3)
11 (3)	00

Physical Memory

64
Page 3
48
Page 2
32
Page 1
16
Page 0
0

- Maintained by the guest OS
- Invalid for the MMU

Shadow Page Table

VPN	MFN
00 (0)	10 (2)
01 (1)	11 (3)
10	00

Machine Memory

64
Page 3
48
Page 2
32
Page 1
16
Page 0
0

- Maintained by the VMM
- Valid for the MMU

Building Shadow Tables

- Problem: how can the VMM maintain consistent shadow pages tables?
  - The guest OS may modify its page tables at any time
  - Modifying the tables is a simple memory write, not a privileged instruction
    - Thus, no helpful CPU exceptions :(
- Solution: mark the hardware pages containing the guest's tables as read-only
  - If the guest updates a table, an exception is generated
  - VMM catches the exception, examines the faulting write, updates the shadow table

More VMM Tricks

- The VMM can play tricks with virtual memory just like an OS can
- Balooning:
  - The VMM can page parts of a guest, or even an entire guest, to disk
  - A guest can be written to disk and brought back online on a different machine!
- Deduplication:
  - The VMM can share read-only pages between guests
  - Example: two guests both running Windows XP

Xen VM interface: CPU

- CPU
  - Guest runs at lower privilege than VMM
  - Exception handlers must be registered with VMM
  - Fast system call handler can be serviced without trapping to VMM
  - Hardware interrupts replaced by lightweight event notification system
  - Timer interface: both real and virtual time

Details: CPU

- Frequent exceptions:
  - Software interrupts for system calls
  - Page faults
- Allow "guest" to register a 'fast' exception handler for system calls that can be accessed directly by CPU in ring 1, without switching to ring-0/Xen
  - Handler is validated before installing in hardware exception table: To make sure nothing executed in Ring 0 privilege.
  - Doesn't work for Page Fault

Xen VM interface: I/O

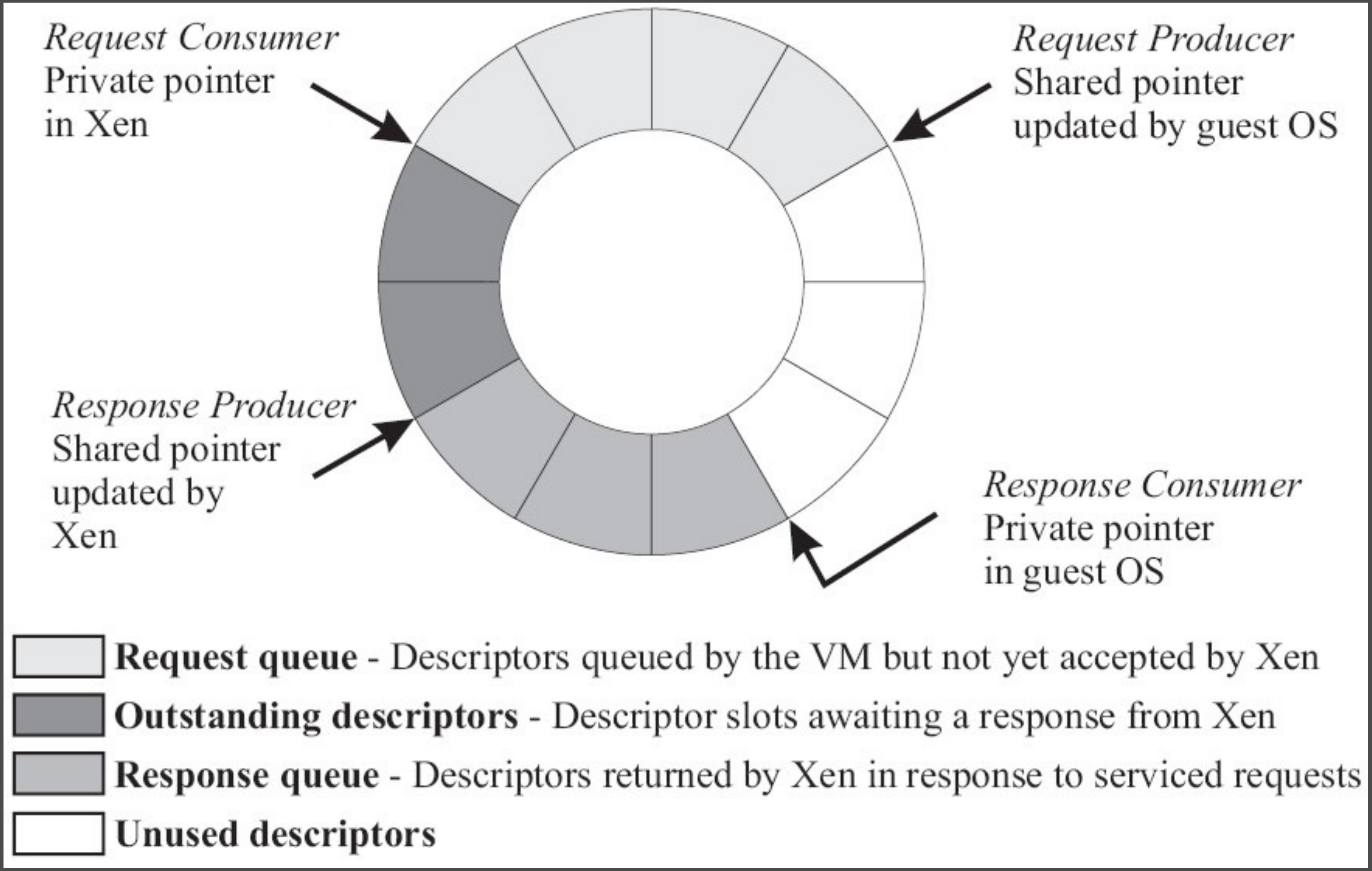
- I/O
  - Virtual devices exposed as asynchronous I/O rings to guests
  - Event notification replaces interrupts

Details: I/O 1

- Xen does not emulate hardware devices
  - Exposes device abstractions for simplicity and performance
  - I/O data transferred to/from guest via Xen using shared-memory buffers
  - Virtualized interrupts: light-weight event delivery mechanism from Xen-guest
    - Update a bitmap in shared memory
    - Optional call-back handlers registered by O/S

Details: I/O 2

- I/O Descriptor Ring:



Request queue - Descriptors queued by the VM but not yet accepted by Xen

Outstanding descriptors - Descriptor slots awaiting a response from Xen

Response queue - Descriptors returned by Xen in response to serviced requests

Unused descriptors

OS Porting Cost

- Number of lines of code modified or added compared with original x86 code base (excluding device drivers)
  - Linux: 2995 (1.36%)
  - Windows XP: 4620 (0.04%)
- Re-writing of privileged routines;
- Removing low-level system initialization code

Control Transfer

- Guest synchronously call into VMM
  - Explicit control transfer from guest O/S to monitor
  - "hypercalls"
- VMM delivers notifications to guest O/S
  - E.g. data from an I/O device ready
  - Asynchronous event mechanism; guest O/S does not see hardware interrupts, only Xen notifications

Event notification

- Pending events stored in per-domain bitmask
  - E.g. incoming network packet received
  - Updated by Xen before invoking guest OS handler
  - Xen-readable flag may be set by a domain
    - To defer handling, based on time or number of pending requests
    - Analogous to interrupt disabling

Data Transfer: Descriptor Ring

- Descriptors are allocated by a domain (guest) and accessible from Xen
- Descriptors do not contain I/O data; instead, point to data buffers also allocated by domain (guest)
  - Facilitate zero-copy transfers of I/O data into a domain

Network Virtualization

- Each domain has 1+ network interfaces (VIFs)
  - Each VIF has 2 I/O rings (send, receive)
  - Each direction also has rules of the form (<pattern>,<action>) that are inserted by domain 0 (management)
- Xen models a virtual firewall+router (VFR) to which all domain VIFs connect

Network Virtualization

- Packet transmission:
  - Guest adds request to I/O ring
  - Xen copies packet header, applies matching filter rules
    - E.g. change header IP source address for NAT
    - No change to payload; pages with payload must be pinned to physical memory until DMA to physical NIC for transmission is complete
  - Round-robin packet scheduler

Network Virtualization

- Packet reception:
  - Xen applies pattern-matching rules to determine destination VIF
  - Guest O/S required to exchange unused page frame for each packet received
    - Xen exchanges packet buffer for page frame in VIF's receive ring
    - If no receive frame is available, the packet is dropped
    - Avoids Xen-guest copies; requires pagealigned receive buffers to be queued at VIF's receive ring

Disk virtualization

- Domain0 has access to physical disks
  - Currently: SCSI and IDE
- All other domains: virtual block device (VBD)
  - Created & configured by management software at domain0
  - Accessed via I/O ring mechanism
  - Possible reordering by Xen based on knowledge about disk layout

Disk virtualization

- Xen maintains translation tables for each VBD
  - Used to map requests for VBD (ID,offset) to corresponding physical device and sector address
  - Zero-copy data transfers take place using DMA between memory pages pinned by requesting domain
- Scheduling: batches of requests in round-robin fashion across domains

Evaluation

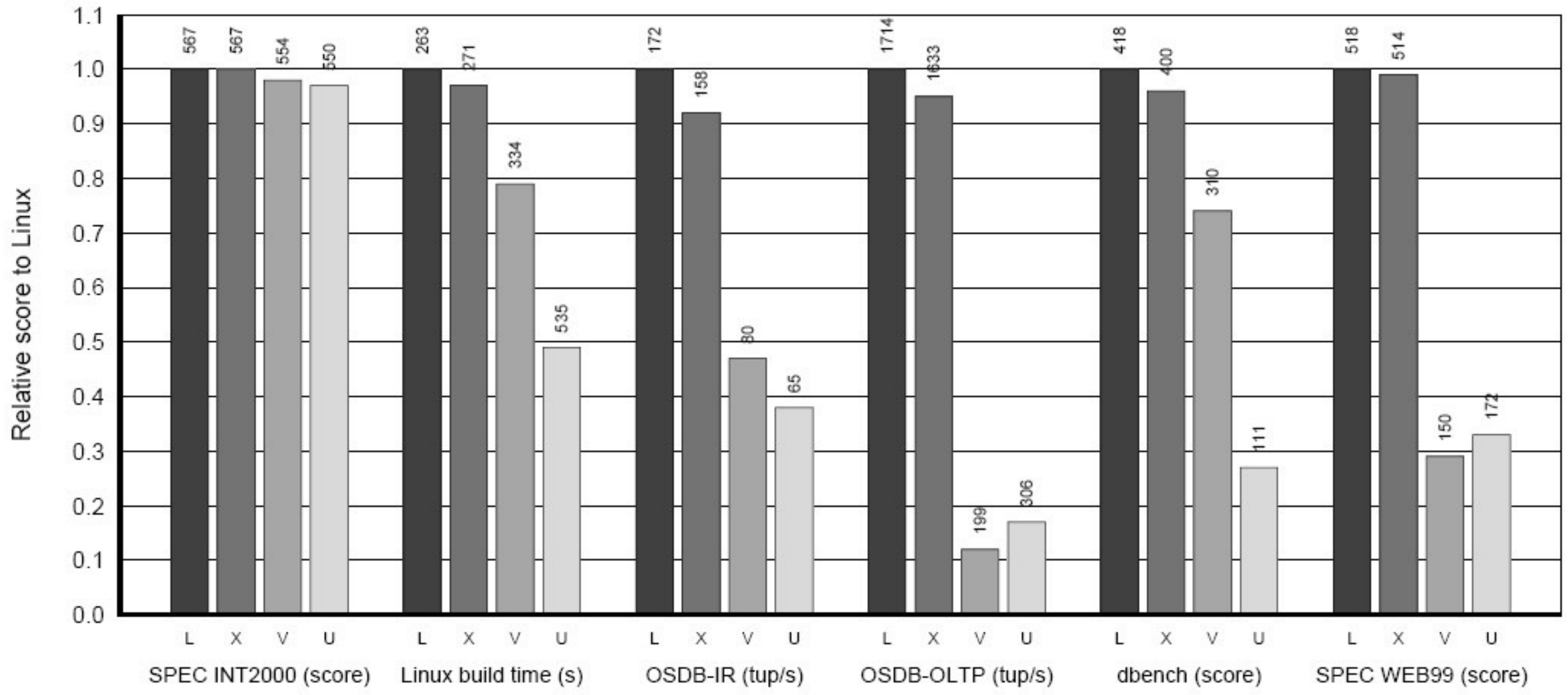


Figure 3: Relative performance of native Linux (L), XenoLinux (X), VMware workstation 3.2 (V) and User-Mode Linux (U).

Microbenchmarks

- Stat, open, close, fork, exec, etc
- Xen shows overheads of up to 2x with respect to native Linux
  - (context switch across 16 processes; mmap latency)
- VMware shows up to 20x overheads
  - (context switch; mmap latencies)
- UML shows up to 200x overheads
  - Fork, exec, mmap; better than VMware in context switches

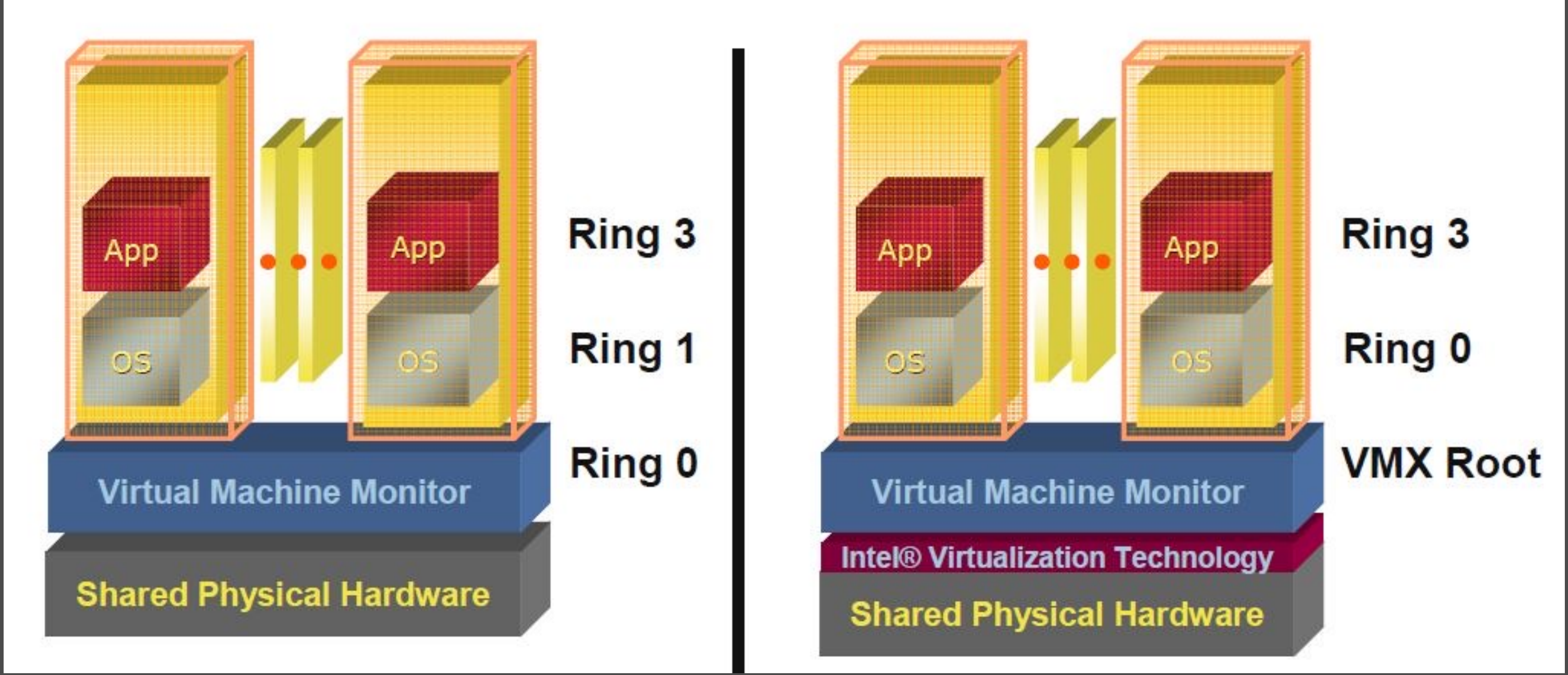
VT-x : Motivation

- To solve the problem that the x86 instructions architecture cannot be virtualized.
- Simplify VMM software by closing virtualization holes by design.
  - Ring Compression
  - Non-trapping instructions
  - Excessive trapping
- Eliminate need for software virtualization (i.e paravirtualization, binary translation).

CPU Virtualization with VT-x

VMX

- Virtual Machine Extensions define processor-level support for virtual machines on the x86 platform by a new form of operation called VMX operation.
- Kinds of VMX operation:
  - **root:** VMM runs in VMX root operation
  - **non-root:** Guest runs in VMX non-root operation
- Eliminate de-privileging of Ring for guest OS.



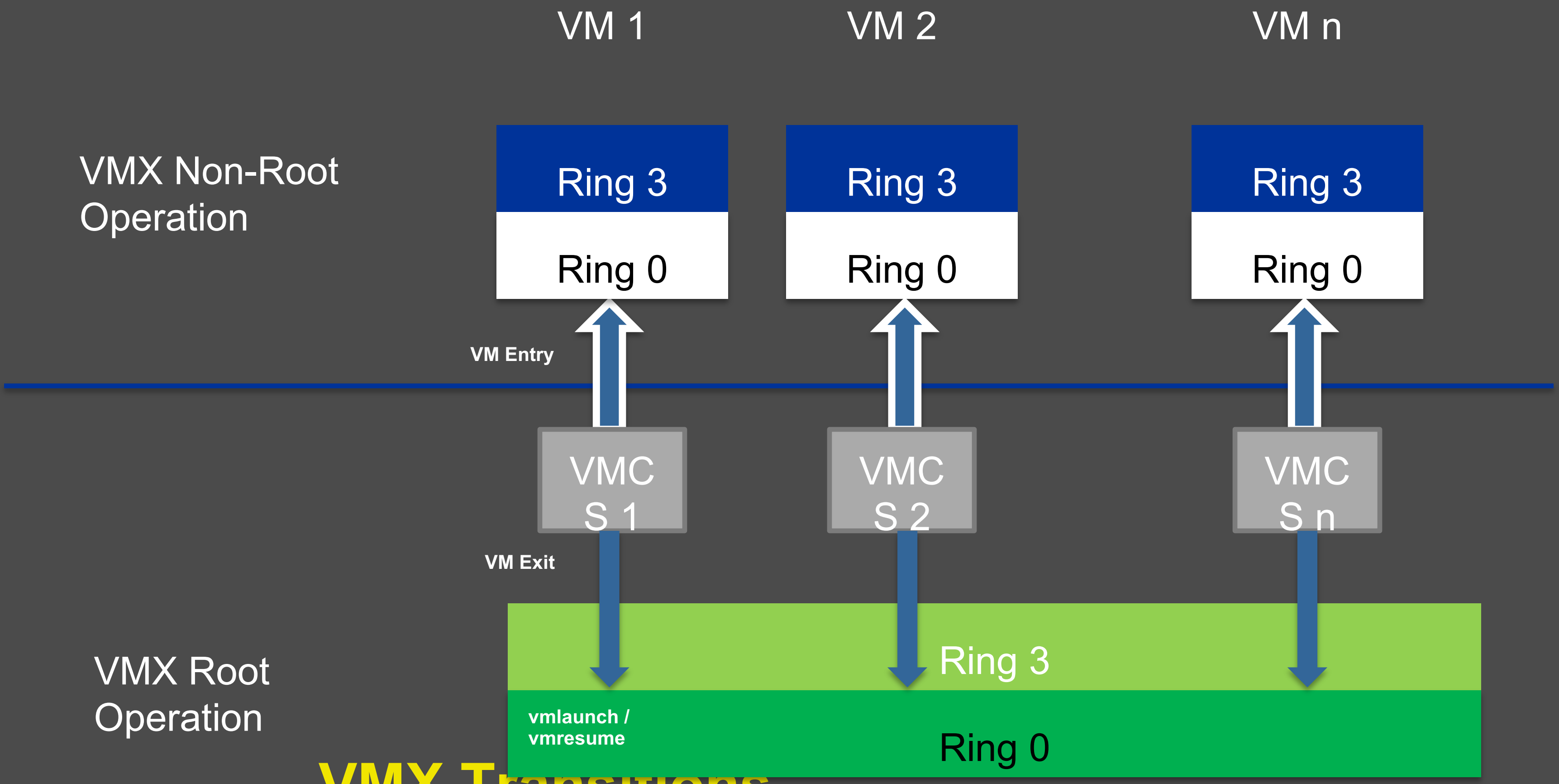
VT-x

VMM ring de-privileging of guest OS	VMM executes in VMX root-mode
Guest OS aware its not at Ring 0	Guest OS de-privileging eliminated
	Guest OS runs directly on hardware

VMX Transitions

- Transitions between VMX root operation and VMX non-root operation.
- Kinds of VMX transitions:
  - **VM Entry:** Transitions into VMX non-root operation.
  - **VM Exit:** Transitions from VMX non-root operation to VMX root operation.
- Registers and address space swapped in one atomic operation.

VMX Transitions



VM 1

VM 2

VM n

VMX Non-Root Operation

Ring 3

Ring 0

Ring 3

Ring 0

Ring 3

Ring 0

VM Entry

VMCS 1

VMCS 2

VMCS n

VM Exit

VMX Root Operation

Ring 3

vmlaunch /
vmresume

Ring 0

VMCS: VM Control Structure

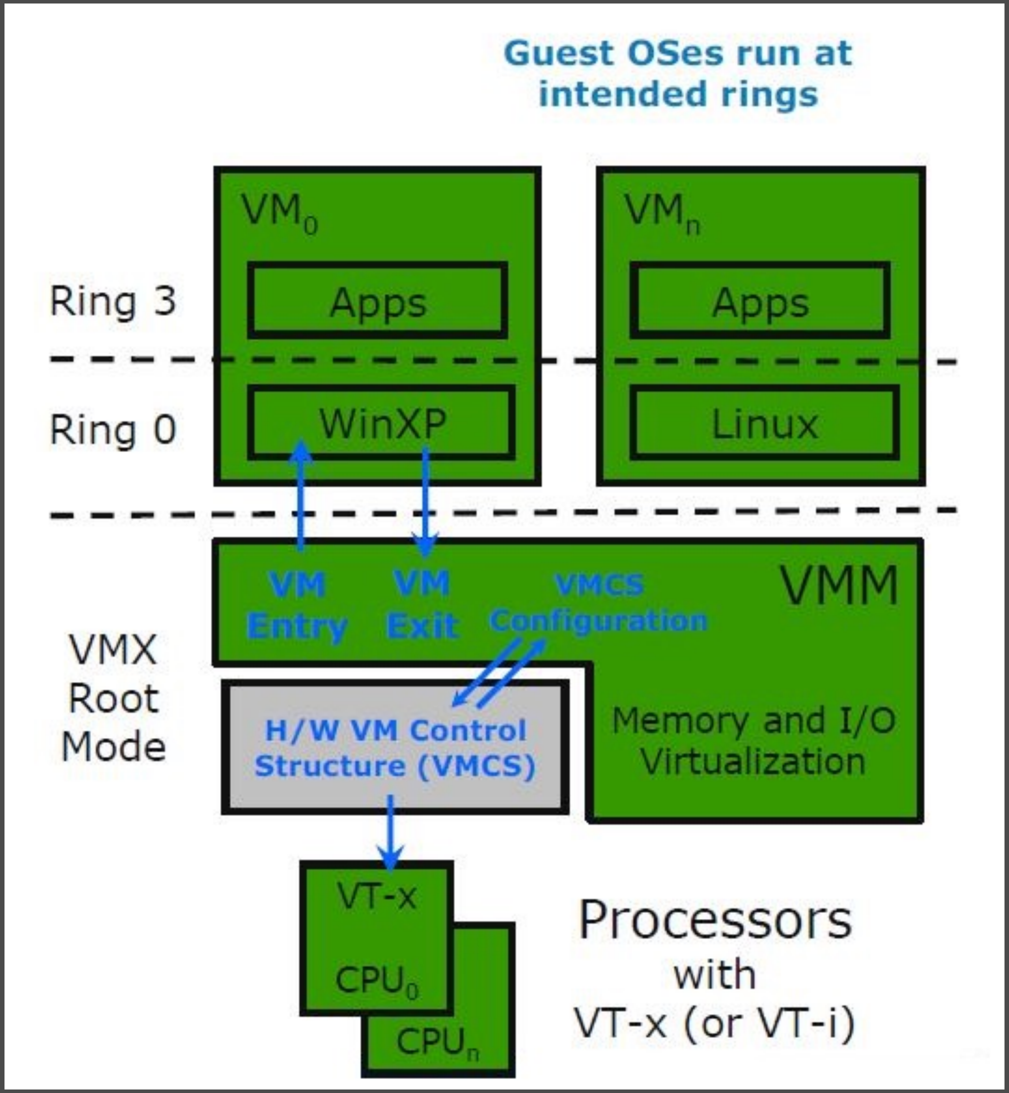
- Data structure to manage VMX non-root operation and VMX transitions.
- Specifies guest OS state.
- Configured by VMM.
- Controls when VM exits occur.

VMCS: VM Control Structure

The VMCS consists of six logical groups:

- **Guest-state area:** Processor state saved into the guest-state area on VM exits and loaded on VM entries.
- **Host-state area:** Processor state loaded from the host-state area on VM exits.
- **VM-execution control fields:** Fields controlling processor operation in VMX non-root operation.
- **VM-exit control fields:** Fields that control VM exits.
- **VM-entry control fields:** Fields that control VM entries.
- **VM-exit information fields:** Read-only fields to receive information on VM exits describing the cause and the nature of the VM exit.

CPU Virtualization with VT-x



Source: [2]

MMU Virtualization with VT-x

VPID: Motivation

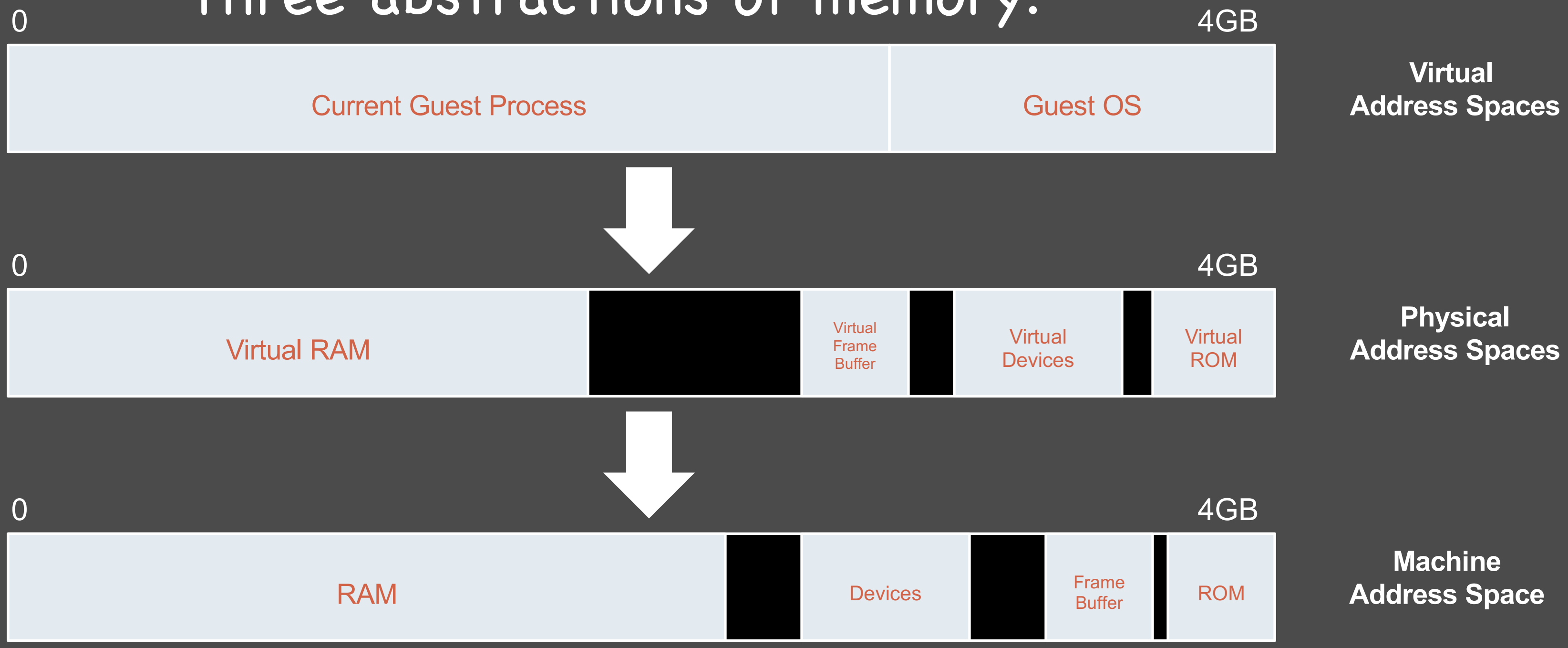
- First generation VT-x forces TLB flush on each VMX transition.
- Performance loss on all VM exits.
- Performance loss on most VM entries
  - Guest page tables not modified always
- Better VMM software control of TLB flushes is beneficial.

VPID: Virtual Processor Identifier

- 16-bit virtual-processor-ID field in the VMCS.
- Cached linear translations tagged with VPID value.
- No flush of TLBs on VM entry or VM exit if VPID active.
- TLB entries of different virtual machines can all co-exist in the TLB.

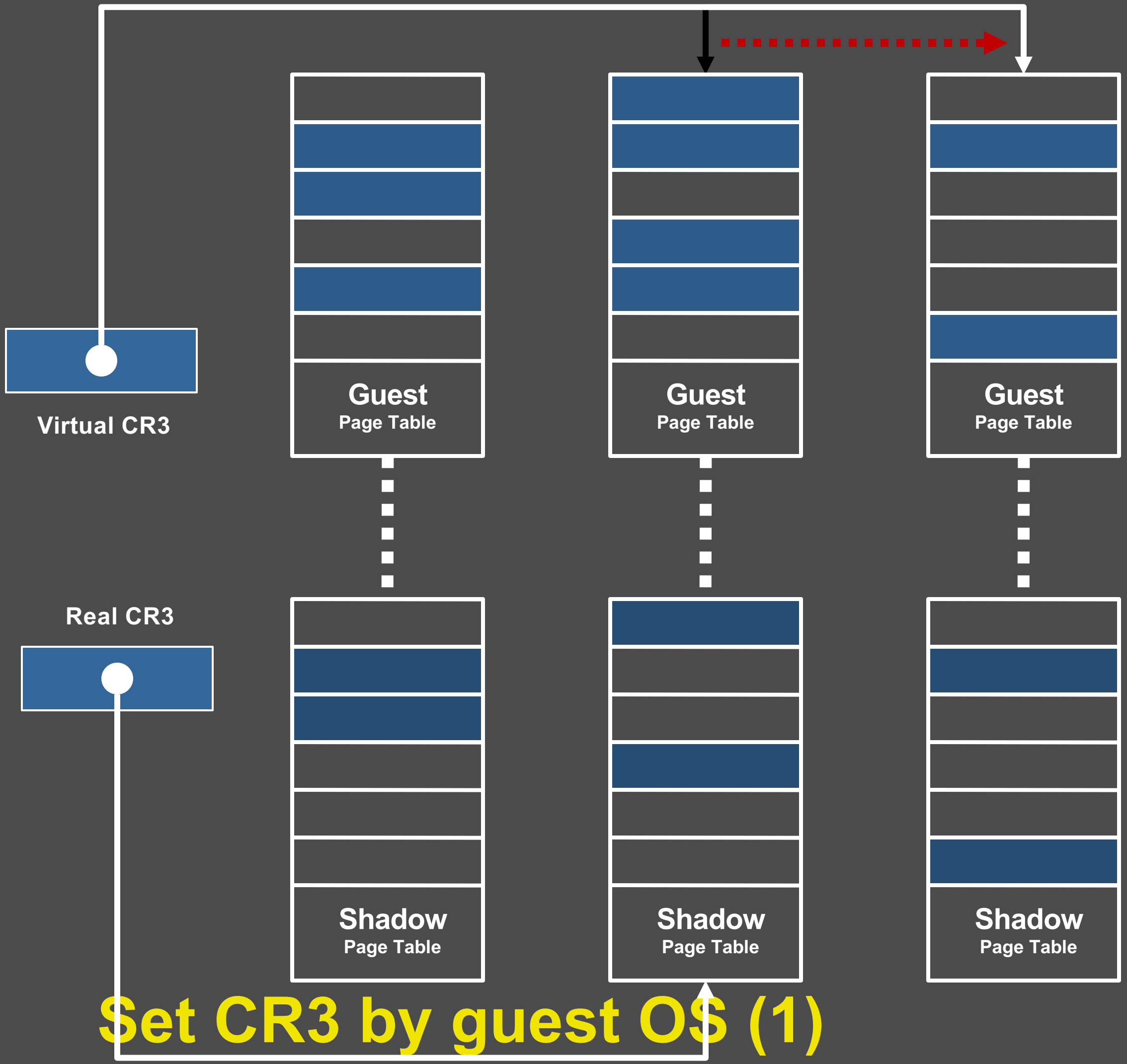
Virtualizing Memory in Software

- Three abstractions of memory:



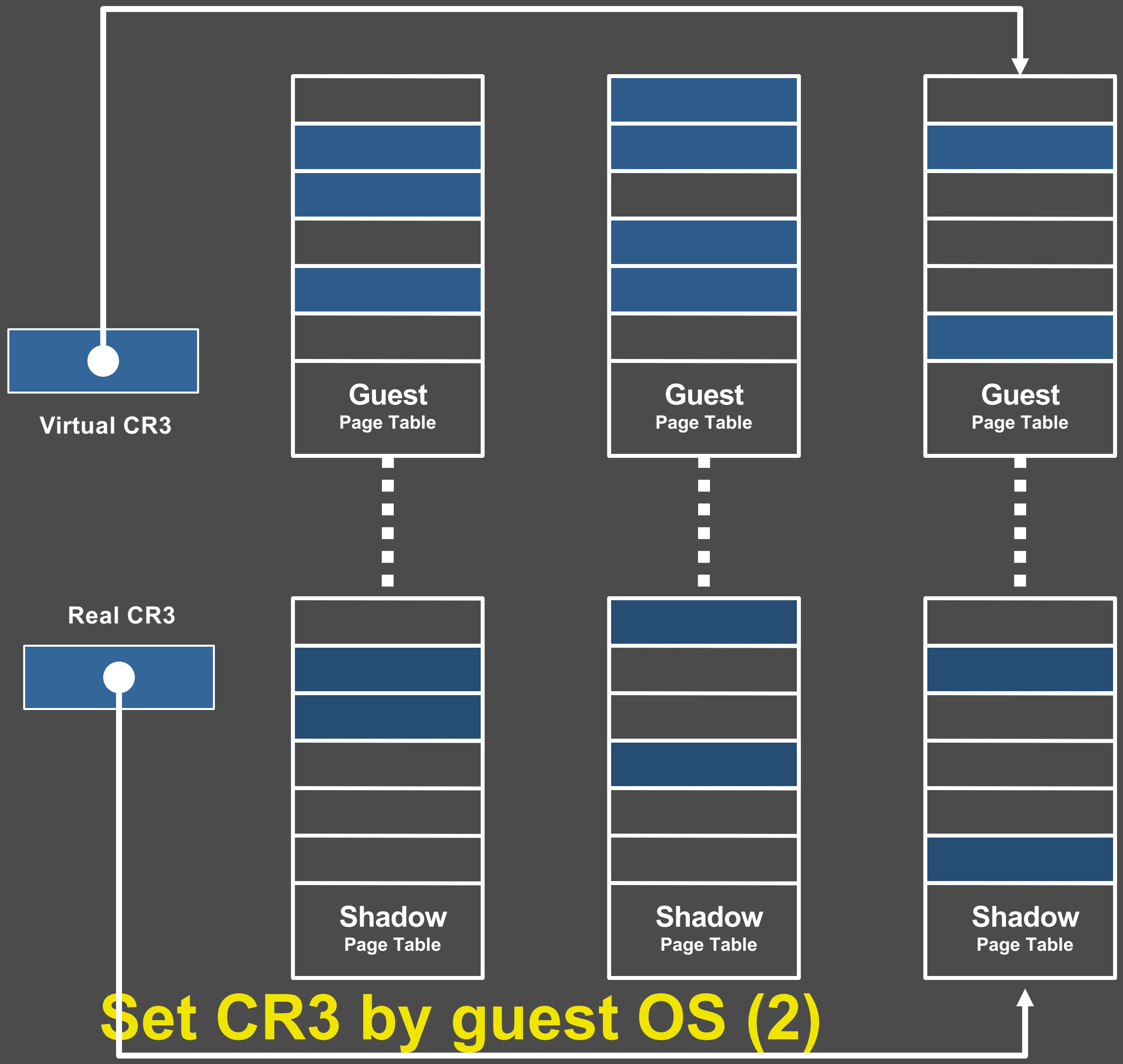
Shadow Page Tables

- 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)

Set CR3 by guest OS (2)



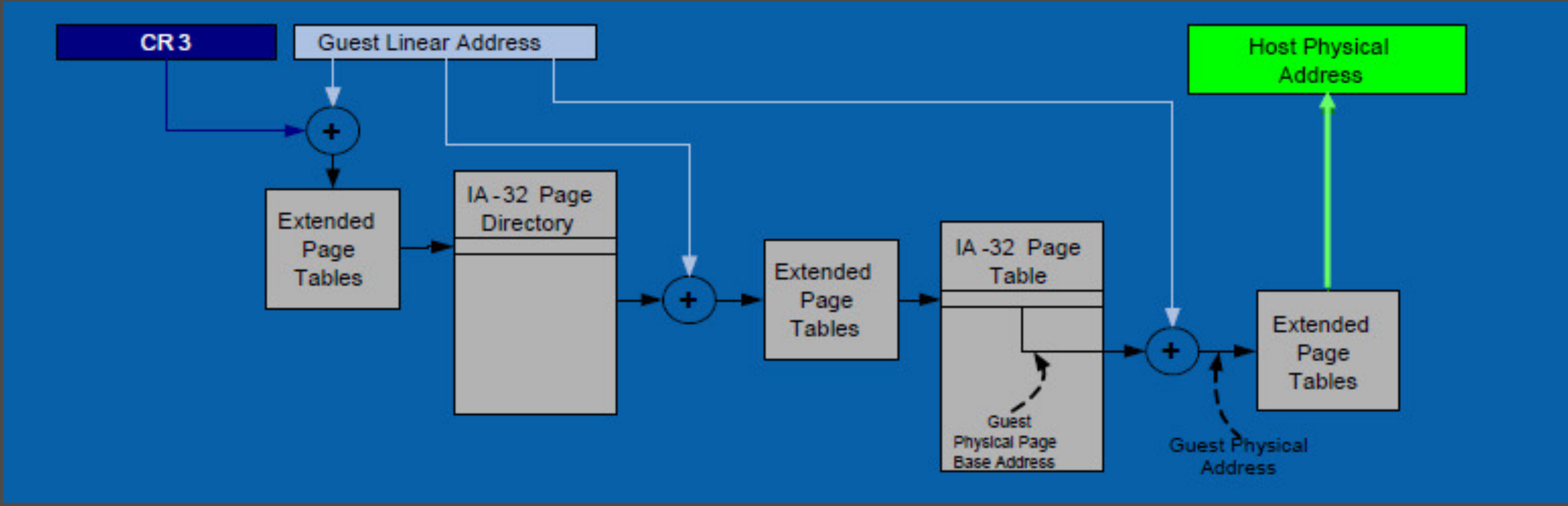
Drawbacks: Shadow Page Tables

- 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.

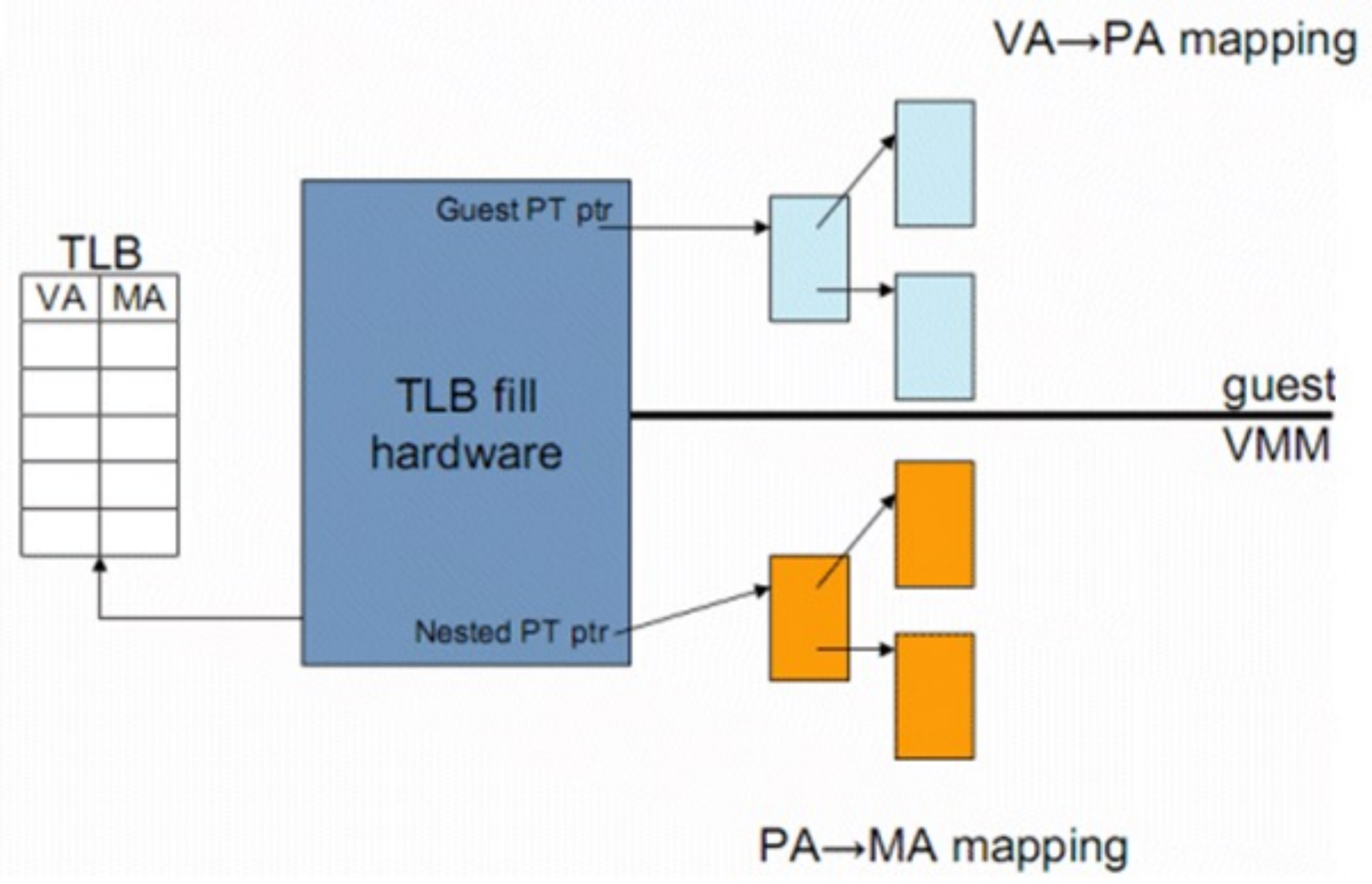
Nested / Extended Page Tables

- Extended page-table mechanism (EPT) used to support the virtualization of physical memory.
- Translates the guest-physical addresses used in VMX non-root operation.
- Guest-physical addresses are translated by traversing a set of EPT paging structures to produce physical addresses that are used to access memory.

Nested / Extended Page Tables



Nested / Extended Page Tables



Source: [4]

Advantages: EPT

- Simplified VMM design.
- Guest page table modifications need not be trapped, hence VM exits reduced.
- Reduced memory footprint compared to shadow page table algorithms.

Disadvantages: EPT

- TLB miss is very costly since guest-physical address to machine address needs an extra EPT walk for each stage of guest-virtual address translation.