Virtualization

D. Duggan
CS526 Enterprise & Cloud Computing
Stevens Institute of Technology

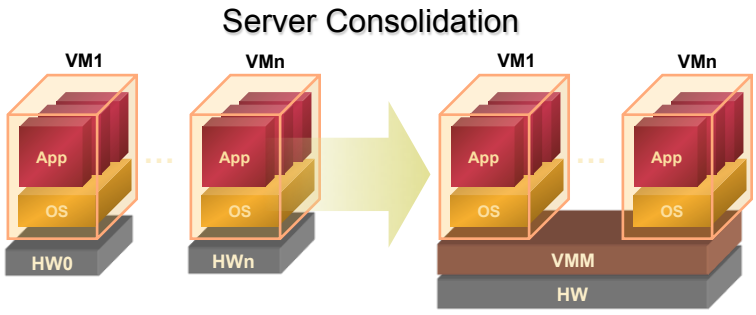
REVIEW: MOTIVATION FOR VIRTUALIZATION

Cost of Distributed Servers

- Energy costs
  - Cooling costs
- Staffing costs
- Data silos and data synchronization

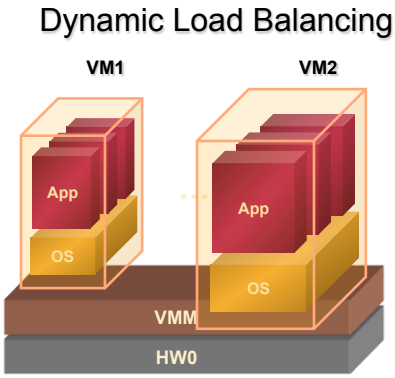
3

Server Consolidation via Virtualization



4

Load Balancing via Virtualization



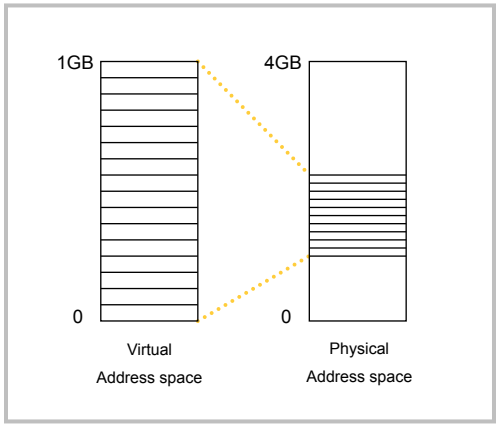
Load balancers in distributed systems cannot take resources from idle servers.

5

MEMORY VIRTUALIZATION

6

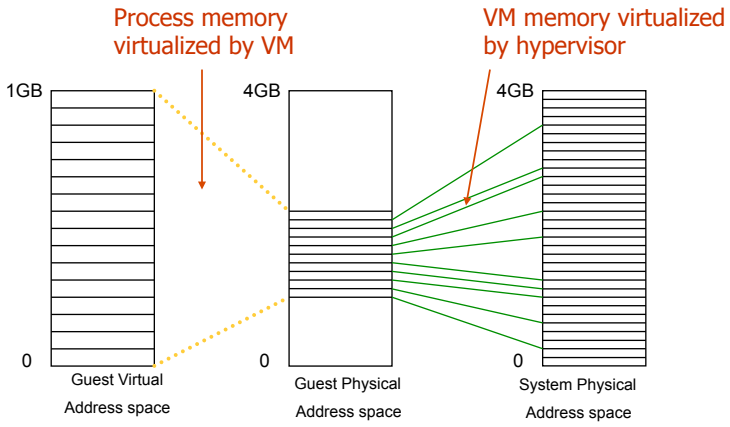
Traditional Memory Map



- Virtual to physical translation (page table) maintained by OSS
- CPU walks page tables automatically during address translation
- Page fault when page is not present or access violation
- CPU uses TLB to cache lookups

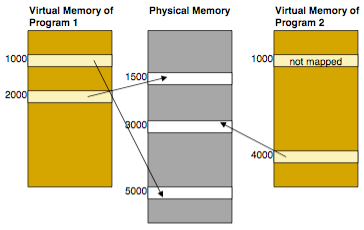
7

Virtualized Memory Map



8

Traditional Memory Map



Page Table for Program 1

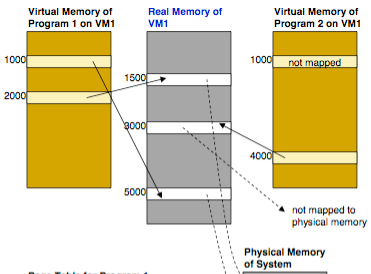
Virtual Page	Real Page
...	...
1000	5000
...	...
2000	1500
...	...

Page Table for Program 2

Virtual Page	Real Page
...	...
1000	not mapped
...	...
4000	3000
...	...

9

Virtualized Memory Map



Page Table for Program 1

Virtual Page	Real Page
...	...
1000	5000
...	...
2000	1500
...	...

Page Table for Program 2

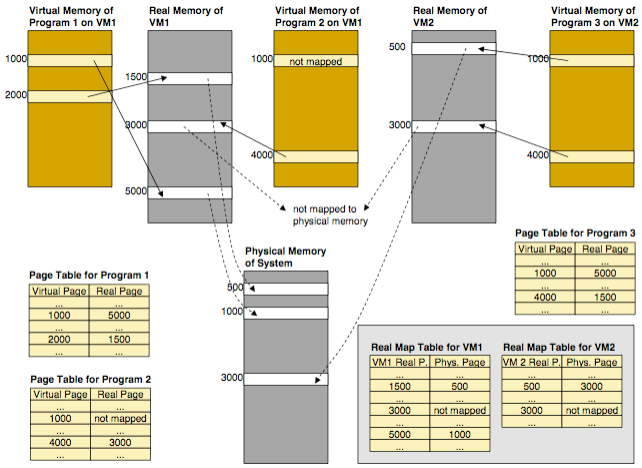
Virtual Page	Real Page
...	...
1000	not mapped
...	...
4000	3000
...	...

Real Map Table for VM1

VM1 Real P.	Phys. Page
...	...
1500	500
...	...
3000	not mapped
...	...
5000	1000
...	...

10

Virtualized Memory Map



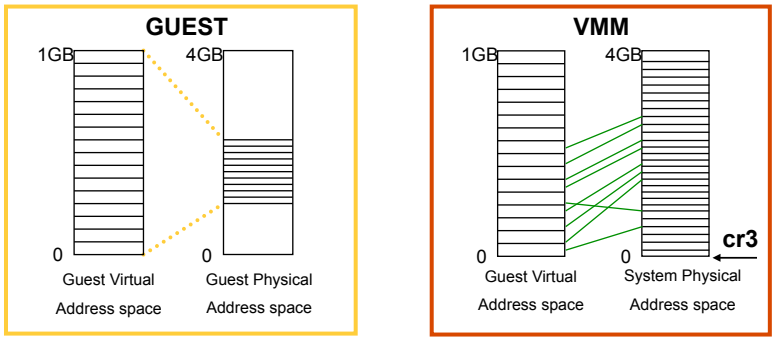
Approaches to Memory Virtualization

- If page table is architected:
  - Shadow page table
  - Extended page table
- If TLB is architected:
  - Copy in/out TLBs during guest switch
  - Use ASID in TLB to identify entries for different guests

MEMORY VIRTUALIZATION: SHADOW PAGE TABLES

13

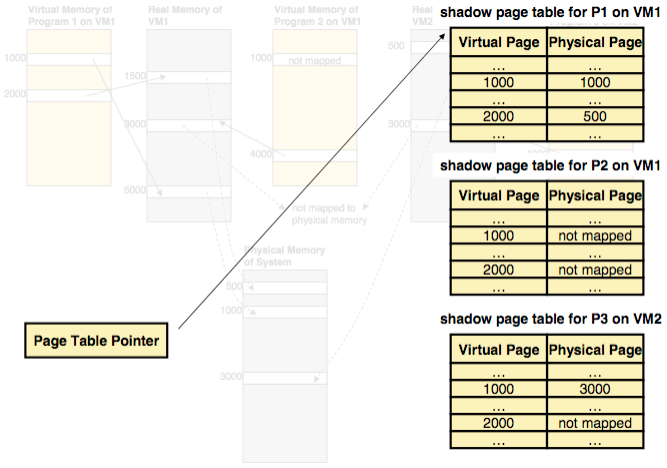
Shadow Page Tables



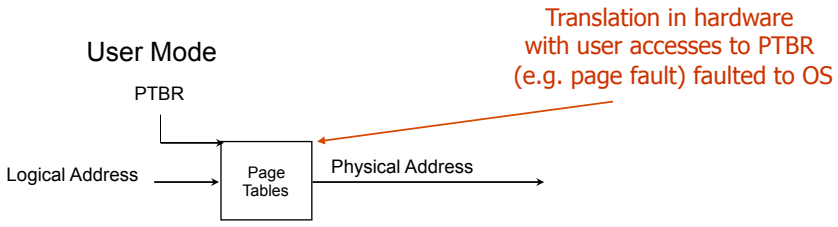
VMM maintains a shadow "copy" of the guest page table
- Hardware (address translation & TLB) sees the shadow page table
- Changes to guest PT must be reflected in shadow PT
- Virtualize the PTBR (page table base register)
  - When a guest tries to access the PTBR, the instruction traps to VMM

14

Shadow Page Tables

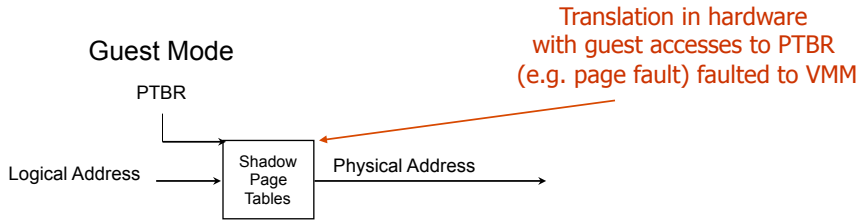


Normal Page Table Usage



- Hardware translates logical address to physical address *on every memory access*
- Page fault handled by OS
  - Trap to OS
  - OS has privileged access to PTBR, modifies page tables

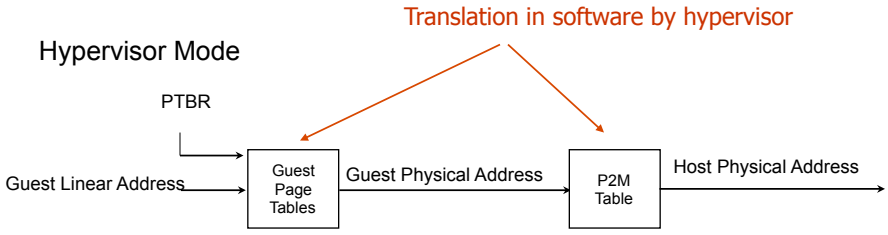
Shadow Page Table Usage



- Hardware translates logical address to physical address *on every memory access*
- Page fault handled by VMM

17

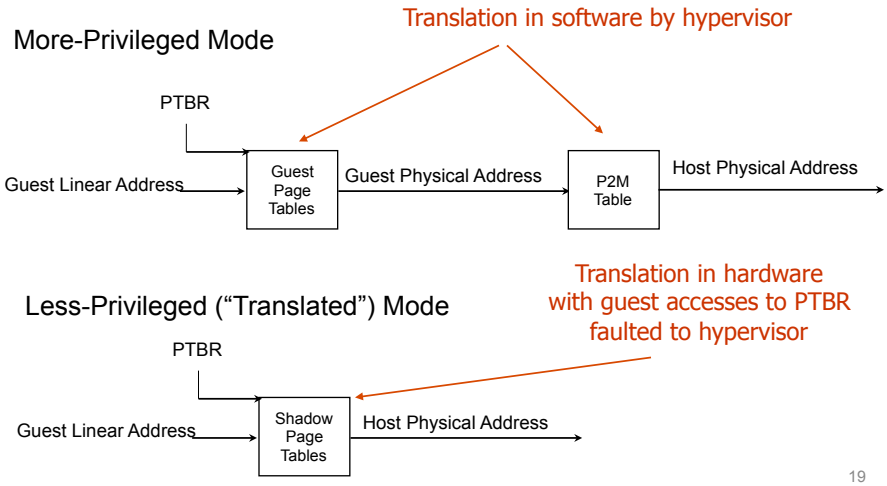
Shadow Page Table Usage



- Hardware translates logical address to physical address on every memory access
- Page fault handled by VMM
  - VMM (not OS!) has privileged access to PTBR
  - VMM ensures shadow page table consistent with guest page tables and P2M table

18

Page Table Shadowing



19

Shadow Table Issues

- All page faults are handled by the VMM
  - VMM has to walk the guest page tables, and instantiate a shadow entry
- VMM needs to propagate access and modify bits
  - A&M bits are used by the demand paging algorithms
    - *Access bits:* restrict read/write accesses to pages e.g. access violation if a write is attempted on a read-only page
    - *Modify bit:* a page being paged out does not need to be rewritten to disk if it has not been modified since last write to disk
  - The hardware modifies the shadow page table entry
  - VMM needs to emulate A&M behavior for the guest page table
  - May take up to 3 actual page faults per one guest page fault

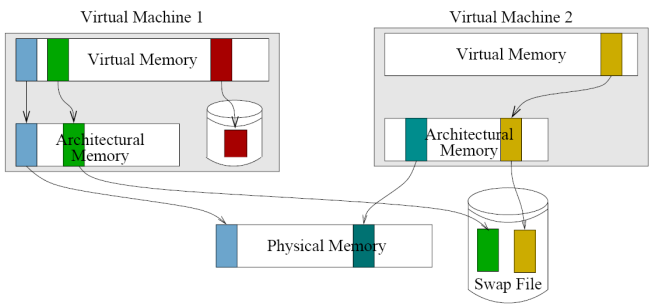
20

MEMORY VIRTUALIZATION: PAGE REPLACEMENT

21

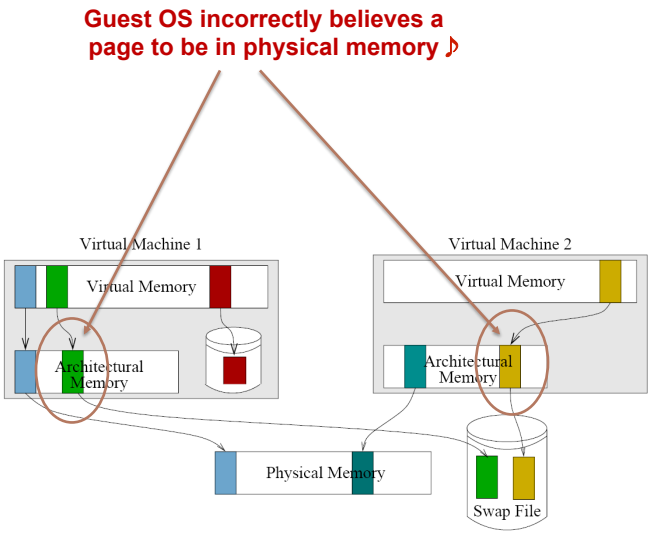
Page Replacement and Double Paging

- Two levels of memory mapping
  - Guest process logical memory to VM memory
  - VM memory to physical memory
- Two independent layers of paging will interact, perform poorly.



22

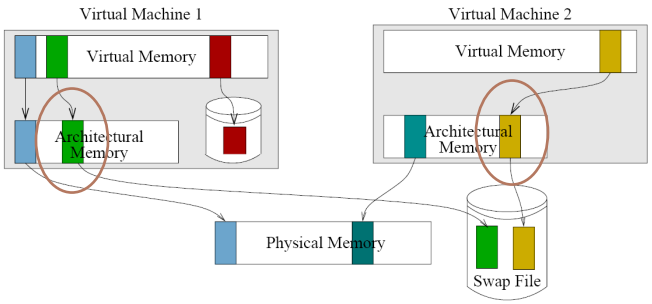
Page Replacement and Double Paging



23

Page Replacement and Double Paging

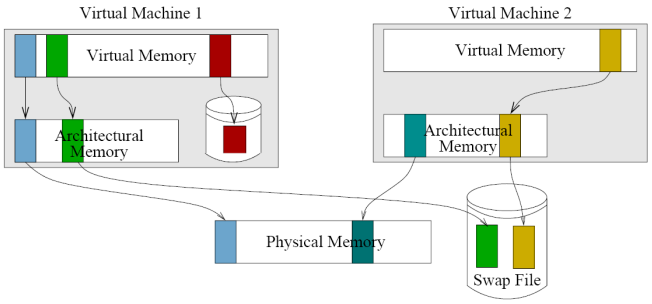
- Page fault traps to VMM
  - Brings in physical pages from VMM swap space
  - Guest OS never learns of the page fault



24

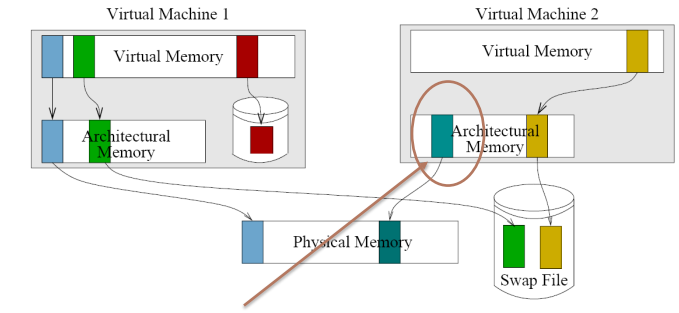
Page Replacement and Double Paging

- Suppose fault on page **not** in physical memory
  - Handled by guest OS, bring in page from swap
  - VMM intercepts accesses to PTBR, page table



25

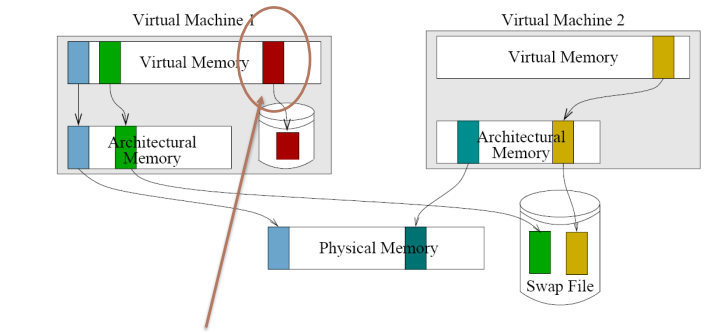
Page Replacement and Double Paging



VMM believes an unneeded page is still in use♪

26

Page Replacement and Double Paging



Guest evicts a page despite available physical memory♪

27

Dealing with Double Paging

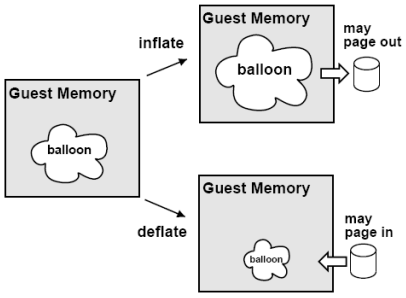
- z/VM: Let VMM handle paging
  - Small swap space for guests
- VMWare ESX Server: Ballooning
  - Driver in guest OS to force paging out

28

Ballooning

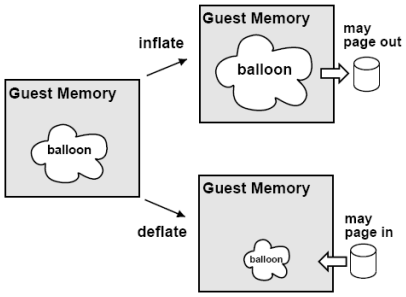
- Reclaims the pages considered least valuable by the operating system running in a virtual machine.
- Small balloon module loaded into the guest OS as a pseudo-device driver or kernel service.
- Module communicates with VMM via a private channel
- Used in VMWare ESX Server

Ballooning



- Inflating a balloon
  - When the VMM wants to reclaim memory
  - Driver allocates pinned physical pages within the VM
  - Increase memory pressure in the guest OS, **reclaim space** to satisfy the driver allocation request
  - Driver communicates the physical page number for each allocated page to VMM

Ballooning



- Deflating
  - Frees up memory for general use within the guest OS

MEMORY VIRTUALIZATION: VIRTUALIZING THE TLB

32

Virtualizing the TLB

- Recall TLB: cache to speed up logical physical address translation
- The TLB must be rewritten whenever a guest is activated
  - copy virtual TLB of guest VM into physical TLB
  - problem: overhead at each switch to another VM and also to VMM

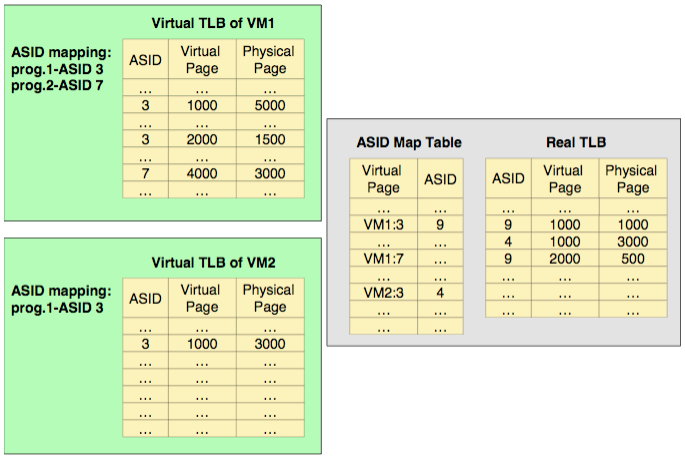
33

Virtualizing the TLB

- Use the address space identifier (ASID) to share the TLB
  - an ASID register indicates address space of active process
- TLB entry is required to have a matching ASID to the register
  - each guest VM has a virtual ASID register
  - VMM manages real ASID register
- Since TLB is shared between guests, it should be very large
- IBM S/370 did not tag the TLB

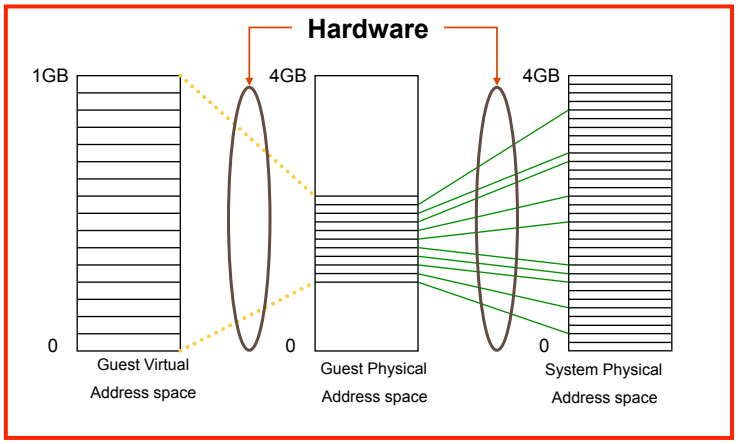
34

Virtualized TLB using ASID



MEMORY VIRTUALIZATION: EXTENDED PAGE TABLES

Hardware-Assisted Virtual Memory



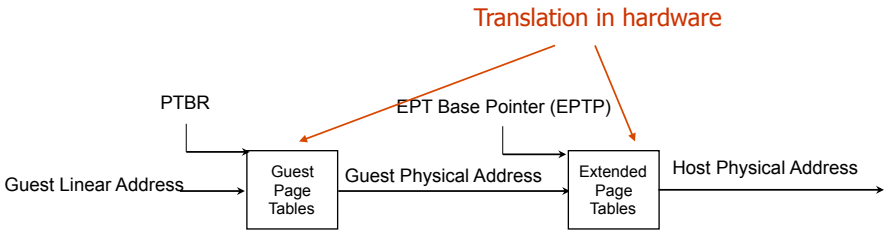
37

Hardware-Assisted Virtual Memory

- Examples
  - IBM S/370
  - Intel VT-x Extended Page Tables
  - AMD Rapid Virtualization Index

38

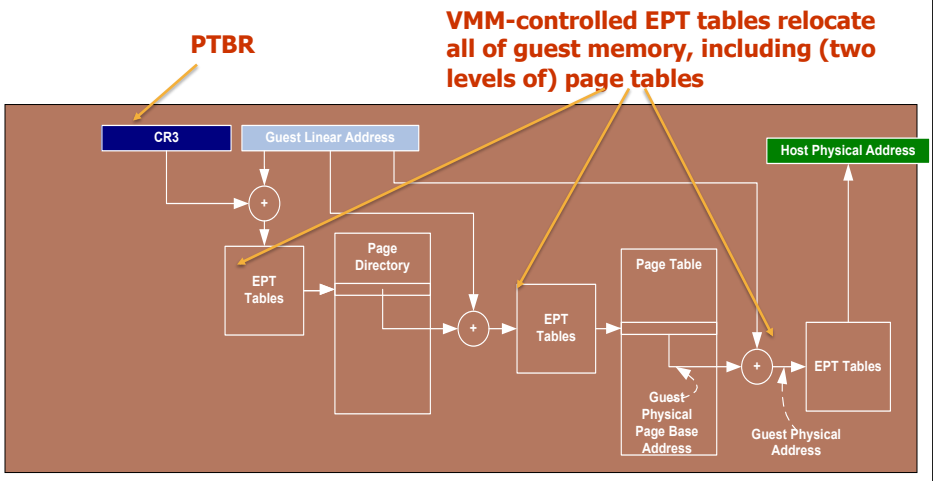
Example: Intel VT-x Extended Page Tables



- Page-table structure under the control of the VMM
  - Defines mapping between guest- and host-physical addresses
  - EPT (optionally) activated on VM entry, deactivated on VM exit
- Guest has full control over its own page tables
  - No VM exits due to e.g. guest page faults or PTBR changes

39

EPT Translation: Details

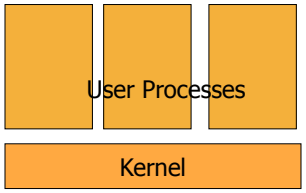


40

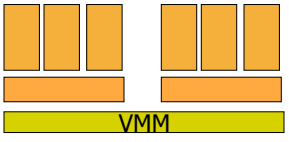
PROCESSOR VIRTUALIZATION

41

Recall: User vs Kernel Mode

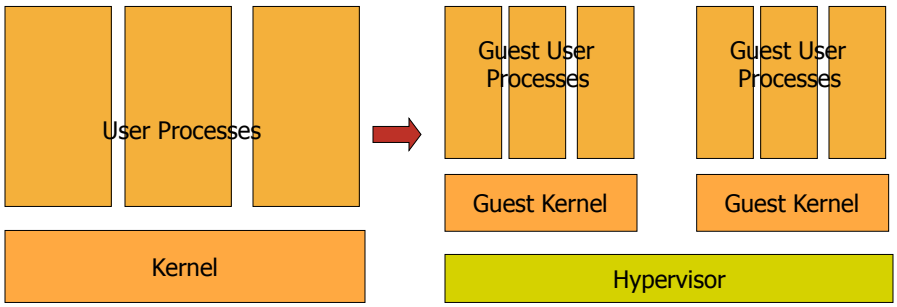


- Normal processing
  - User mode for applications
  - Kernel mode for OS
  - App traps to kernel to execute privileged instructions
- Virtualized processing
  - User mode for apps
  - User mode for OS
  - Guest OS traps to VMM to execute privileged instructions



42

User/Kernel Mode in Virtualization



Hypervisor runs in kernel mode.

Guest kernels may now be running in user mode!

43

Privileged vs Sensitive Instructions

- Privileged instructions
  - Can only be executed in system mode
  - Must trap if executed in user mode

44

Privileged vs Sensitive Instructions

- Sensitive instructions
  - Control "sensitive" resources
  - Does **not** imply that they are always privileged
  - **Control-sensitive:** change resource configurations
    - Example: set CPU timer
  - **Behavior-sensitive:** result depends on resource configurations
    - Example: POPF, Pop stack into flag register (enable/ disable interrupts) on Intel-32 has no effect if in user mode

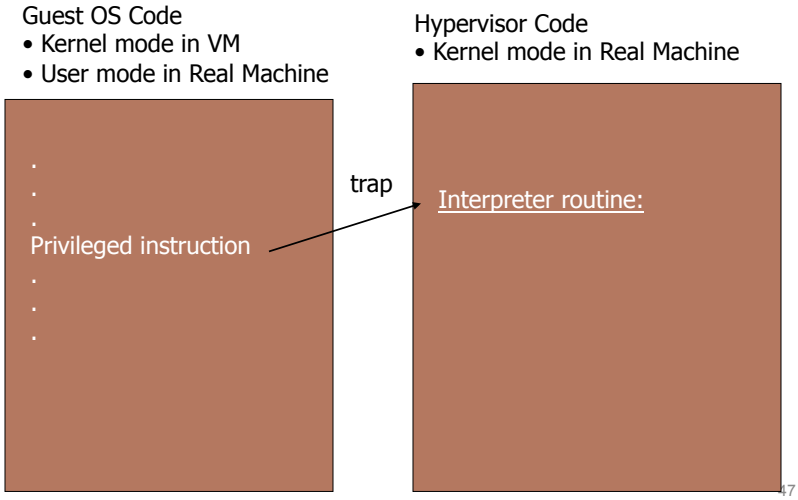
45

Hypervisor implementation 1: Trap and Emulate

- Hardware requirement (Popek and Goldberg): All sensitive instructions should be privileged
  - All non-privileged instructions are executed natively
  - All sensitive instructions trap to the VMM
- This prevents tampering with hardware
  - Guest kernel executes in user mode
  - Privileged instructions, although not available in user mode, are emulated by trapping to VMM
- Cost: Trapping to VMM is expensive
- Example: VM/370

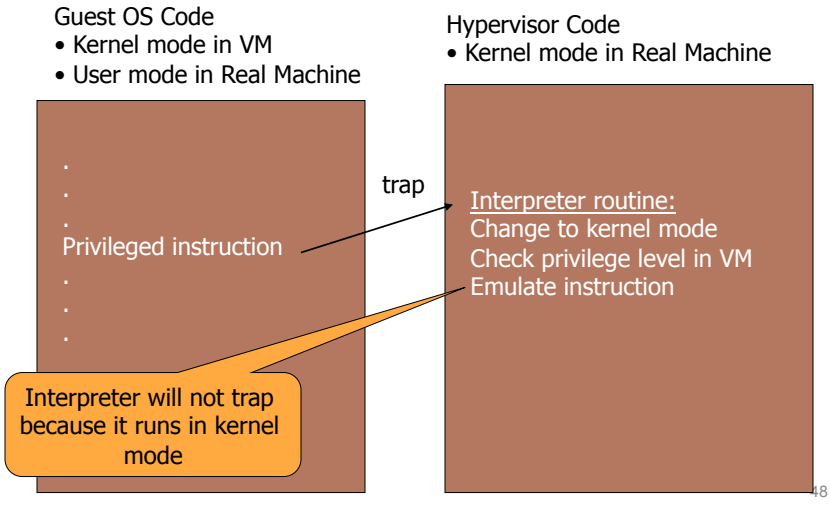
46

Emulating Privileged Instructions



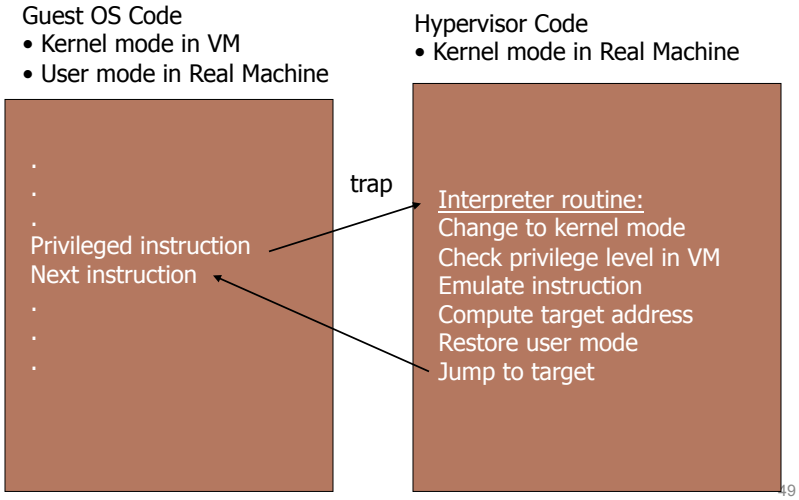
47

Emulating Privileged Instructions



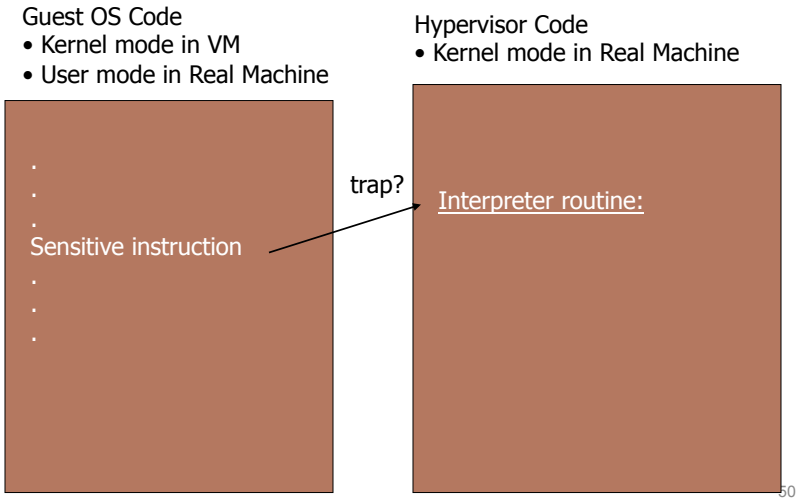
48

Emulating Privileged Instructions



49

What if not all sensitive instructions are privileged?



50

What if not all sensitive instructions are privileged?

- Intel-32 instruction set architecture (ISA) contains 17 non-privileged but sensitive instructions
- Example: POPF instruction
  - behavior sensitive
    - IA-32 POPF instruction: pops the flag registers from a stack held in memory.
    - One of the registers is the interrupt-enable flag, which can be modified only in privileged mode.
    - In user mode, this instruction overwrites all flags except the interrupt-enable flag (i.e. no-op)
  - but is not privileged (does not trap in user mode)

51

Trap & Emulate Summary

- Can be used to virtualize if all sensitive instructions are privileged
- Not true on IA-32 architecture
- Trap & emulate not efficient
- Alternatives:
  - Binary translation
  - Paravirtualization
  - Hardware assists (esp. interpretive execution)

52

PROCESSOR VIRTUALIZATION: BINARY TRANSLATION

53

Hypervisor Implementation 2: Binary Translation

- At run-time, replace sensitive instructions with calls to hypervisor
  - Translation while executing guest code
  - Cache translations of sensitive instructions
    - One translation of an instruction even if in a loop
- Example: VMWare ESX Server

54

Hypervisor Implementation 2: Binary Translation

- Replace traps to hypervisor with callouts to interpreter routines
  - Some traps may be due to page faults
    - hypervisor must manage shadow page tables
  - Adaptively decide whether to translate instructions that access memory
    - based on how often they page fault.

55

Comparison of Hardware and Software Approaches

- Software
  - Trap elimination
    - Replace with callouts to hypervisor
  - Emulation speed
    - No need to figure out which operation caused the trap; operation is known during translation
- Hardware
  - Code density
    - No code bloat from translation
  - Precise exceptions
    - Since code is unchanged
  - System calls to guest OS
    - Guest trap & system call vectors already installed
    - No VMM intervention

56

PROCESSOR VIRTUALIZATION: PARAVIRTUALIZATION

57

Hypervisor Implementation 3: Paravirtualization

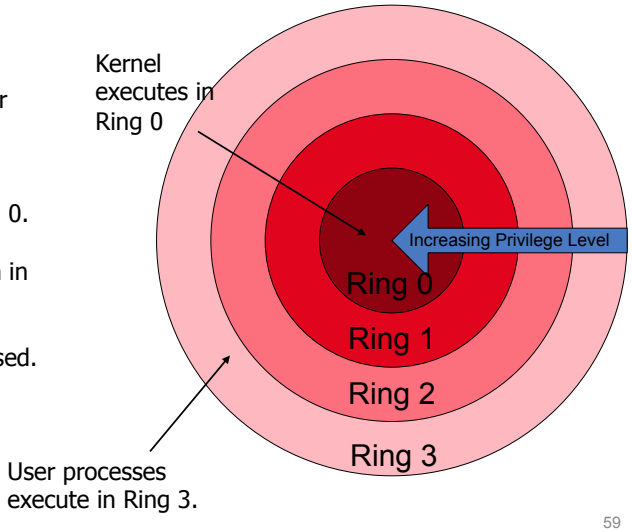
- Guest OS must be modified to understand virtualization
  - **Hypercall**: synchronous calls from guest to hypervisor
    - Analogous to system calls
  - **Events**: asynchronous notifications from hypervisor to guests
    - Replace device interrupts
  - Hypervisor validates use of privileged instructions
  - Resources available reflect virtualization
    - Xen: timers for both wall-clock time and virtual time
- User applications are not modified
- Example: Xen

58

Protection Rings in X86

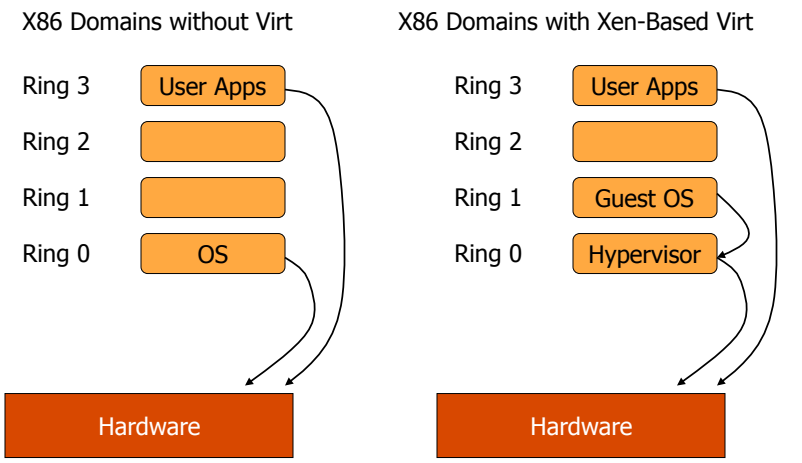
X86 generalizes user and kernel mode to "rings" of privilege.

- Kernel runs in Ring 0.
- User processes run in Ring 3.
- Rings 1 and 2 unused.



59

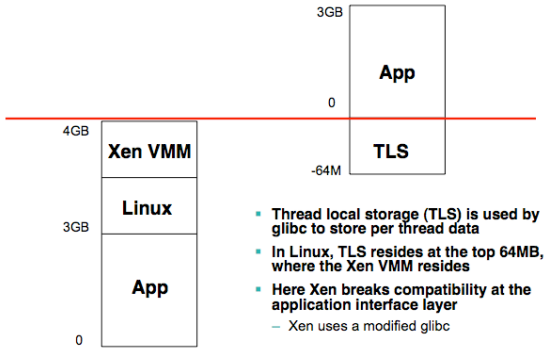
Rings and Virtualization



60

Trapping to Hypervisor

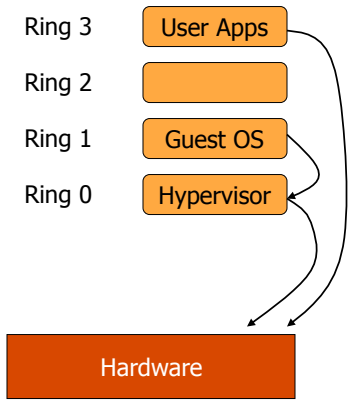
- Xen lives on the top of every address space, so getting in and out of hypervisor doesn't require TLB flush



61

Xen Hypervisor

- Uses X86 protection rings
  - Xen runs in ring 0
  - Hypercalls jump to Xen in ring 0
  - Ring 1,2 for guest OS
  - Ring 3 for user-space
- A guest OS may install a "fast trap" handler so user processes do not have to go through the hypervisor
- Hypervisor administrator runs as domain 0 guest



62

PROCESSOR VIRTUALIZATION: INTERPRETIVE EXECUTION

63

Hypervisor Implementation 4: Interpretive Execution

- Hardware assist: add new instructions as assists for hardware-based virtualization
- Interpretive execution: add a new mode for guest OS execution

- Privileged instructions emulated by VMM
  - Is guest in user or in system mode?
  - VMM must track aspects of guest state

64

Hypervisor Implementation 4: Interpretive Execution

- Instruction Emulation Assist:
  - Replace VMM emulation with virtualized instruction
  - Hardware checks guest VM state
    - performs either the entire action in guest kernel mode
    - or the restricted action in guest user mode

65

Hypervisor Implementation 4: Interpretive Execution

- Example: System/370, load processor status word (LPSW) privileged instruction
  - P bit of PSW, indicates if guest in kernel mode
  - Hardware compares with current *virtual* PSW
  - No more emulation and context switch to VMM
  - Unless LPSW traps for user code

66

Hypervisor Assists

- VMM Assists added in hardware to improve performance
  - **Context switch:** Save and restore guest state
  - **Decoding** of privileged instructions
  - **Virtual interval timer:** virtual timer is decremented every time the real timer is decremented
  - **New instructions**
    - Page locking/unlocking
    - Page table in

67

Interpretive Execution

- Idea: The hypervisor can allow **guest OS** to run with the right to execute some privileged instructions
  - Rather than trap to hypervisor to emulate those instructions, virtualize those instructions in the hardware
  - **Guest OS** is allowed to execute those virtualized instructions
  - **Guest user** is stopped by the virtualized instructions

68

Interpretive Execution

- Idea: The hypervisor can allow **guest OS** to run with the right to execute some privileged instructions
- Guest has state that is of interest to hypervisor
  - State is shadowed in the hypervisor
  - Virtualized instructions keep the shadow state in sync with guest state, in hardware
  - No need to trap to hypervisor

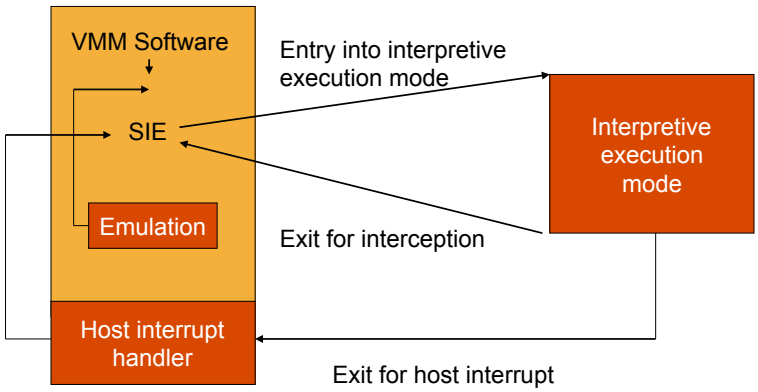
69

Example: IBM S/370 Interpretive Execution Facility (IEF)

- The processor directly executes most of the functions of the virtual machine in hardware.
- Interpretive Execution Entry and Exit
  - Entry
    - Start Interpretive Execution (SIE) : The software gives up control to the hardware IEF part and the processor enters the interpretive execution mode.
  - Exit
    - Host Interrupt
    - Interception
      - Unsupported hardware instructions.
      - Exception during the execution of interpreted instruction.
      - Some special cases…

70

Interpretive Execution Entry and Exit



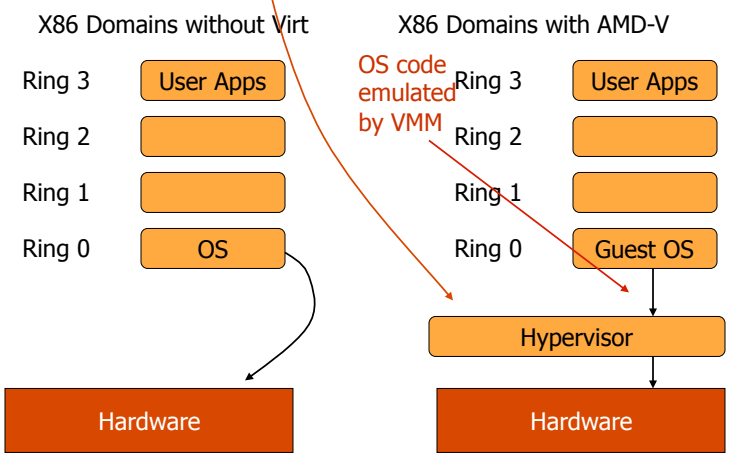
71

PROCESSOR VIRTUALIZATION: AMD-V

72

AMD-V distinguishes "guest" and "host" mode.
Hypervisor runs in "host" mode.

Example: AMD-V



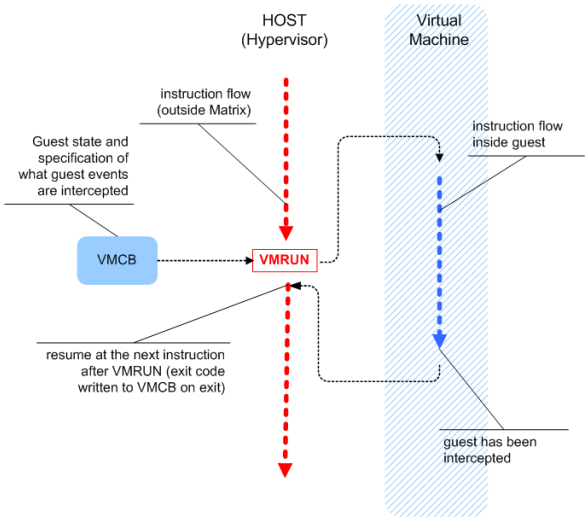
73

Example: AMD-V

- Hypervisor runs guest OS in "guest" mode, with vmrun instruction
  - Guest state of interest to hypervisor kept in VM Control Block (VMCB)
    - Virtualized instructions update VMCB, in hardware
  - Guest can exit vmrun due to page faults, I/O interrupts, etc
  - Hypervisor determines why vmrun exited (by examining VMCB) and acts accordingly

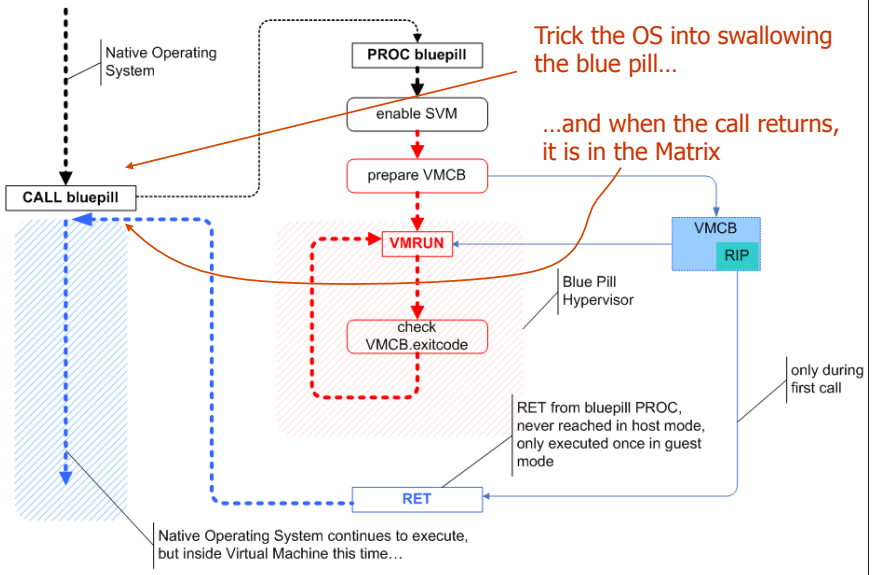
74

AMD-V and VMRUN



75

Aside: The "Blue Pill" Attack



76

PROCESSOR VIRTUALIZATION: VT-X

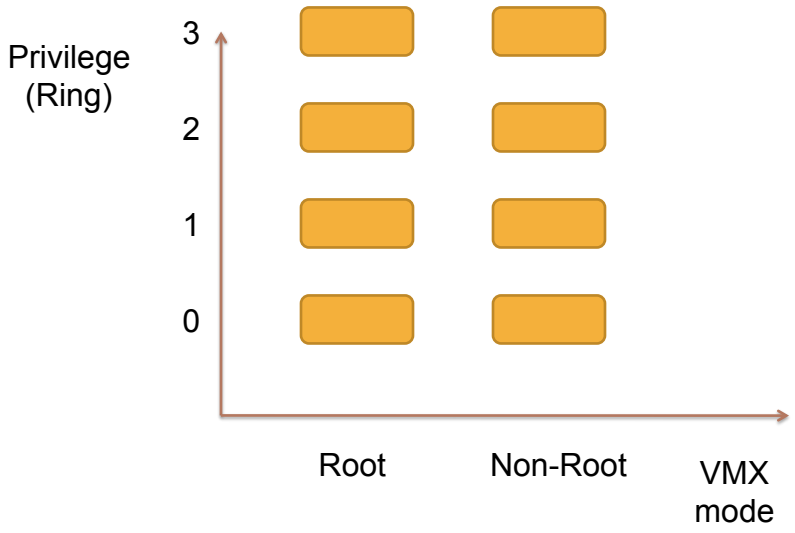
77

Example: Intel VT-x

- Main Feature: VMX mode of operation
  - **VMX root operation**
    - Fully privileged, intended for VM monitor
    - New instructions – VMX instructions
  - **VMX non-root operation**
    - Not fully privileged, intended for guest software
    - Reduces Guest SW privilege w/o relying on rings
  - Similar to AMD-V
    - Operations
      - VMXON, VMXOFF: turn on/off virtualization
      - VMLAUNCH, VMRESUME: enter guest mode

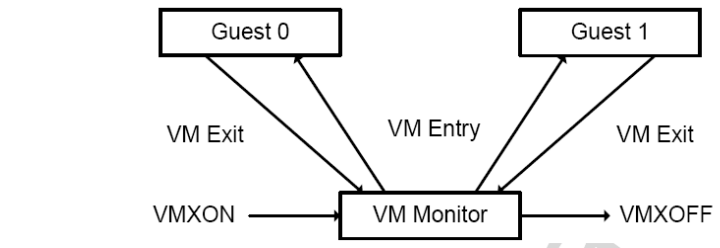
78

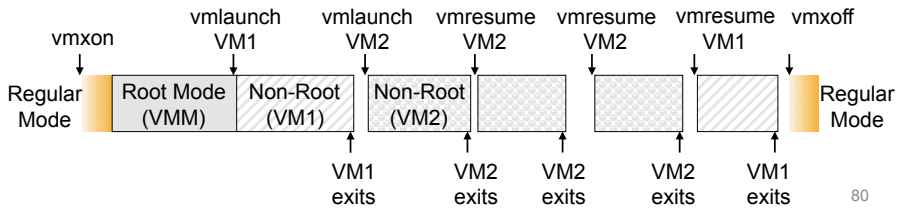
VMX mode versus Privilege Level



79

Lifecycle of VMM in VT-x





80

VT-x Operations

IA-32 Operation

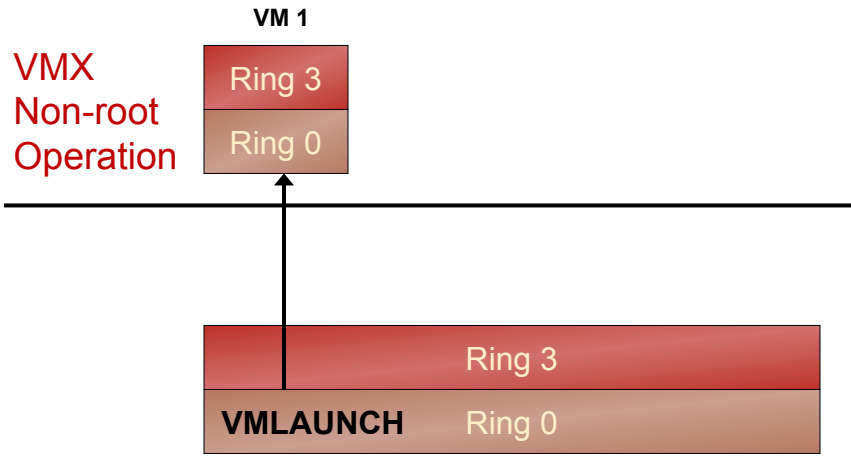
81

VT-x Operations

VMX Root Operation

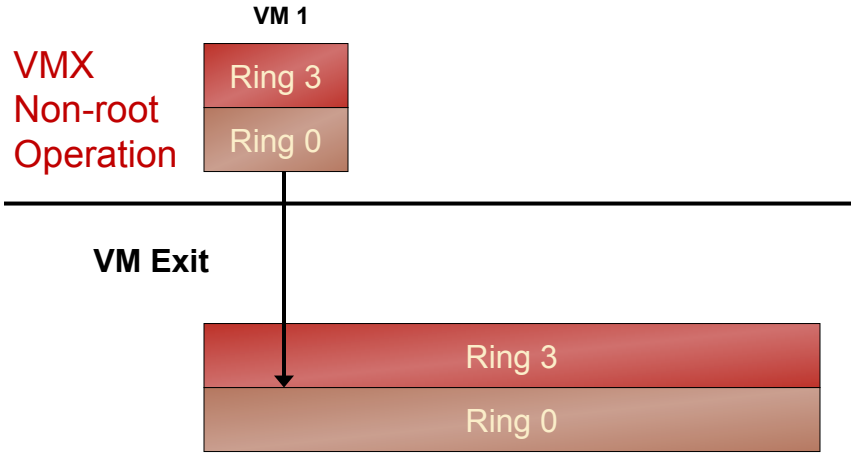
82

VT-x Operations



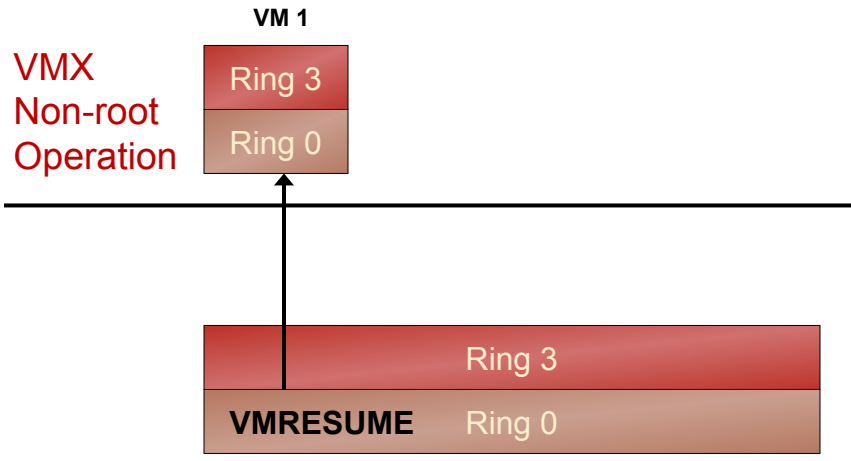
83

VT-x Operations



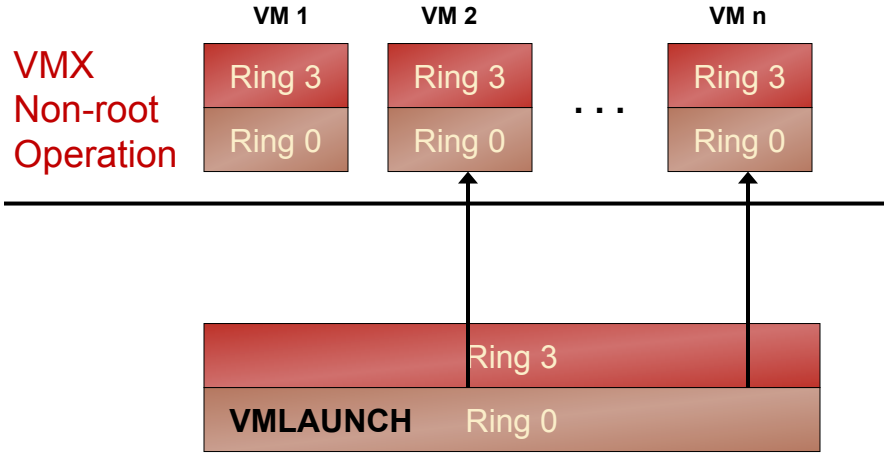
84

VT-x Operations



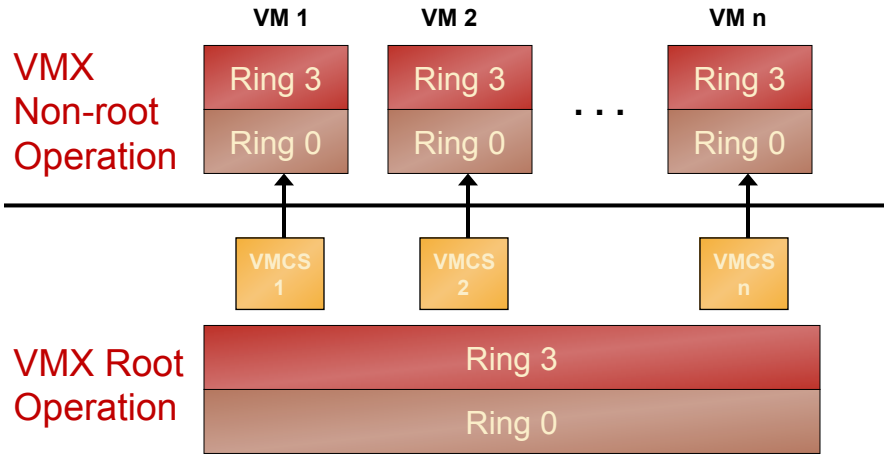
85

VT-x Operations

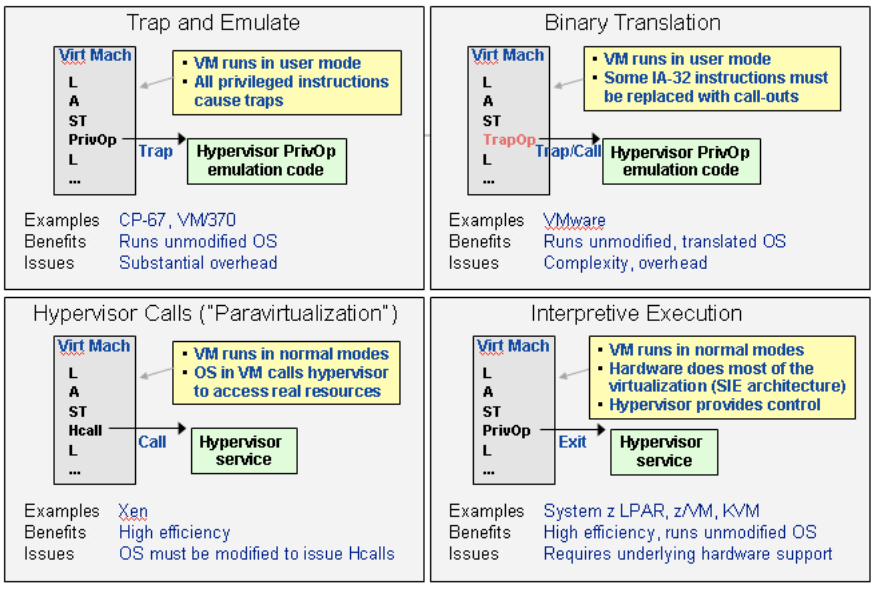


86

VT-x Operations



87



88