
Virtual Machine Assists

From:

“Virtual Machines” JE.Smith, R.Navir

Uhlig, Rich, et al. "Intel virtualization technology." *Computer* 38.5 (2005): 48-56.



Performance: The fee for VM?

- Not only “maximize” server utilization but, how to achieve near-native performance?
 - Early usages with VM reports 20-30% performance degradation
- The flexibility provide by the VMM has to be paid somehow
 - Apparently, always there will a performance hit of using VM
- (next)... The components of such “fee” and how to minimize each one (with emphasis in x86)

Reasons for VM Slowdown

- Setup a Save state
 - Each time the VM is activated/deactivated the state of the processor/MMU should be copied from/to memory
 - Copy of registers, timing facilities and perhaps MMU structures
- Emulation
 - Not all insn can be executed natively
 - Some insn should be emulated (mostly through interpretation)
 - Trap to VMM
 - Interpretation by VMM
 - Return from VMM to guest
- Interrupt handling
 - Reflect through VMM before getting to Guest/OS
- Virtual Memory Management
 - Shadow page faults when page is already mapped
- Duplicated effort between VMM and Guest/OS
 - Memory management done by both
- Interval timer
 - Notion of time of guest/OS can be lost (because all the previous reasons)

Virtual Machine Assists

- Ways of making application on VM run faster
 - Have no performance effect if run in native mode
 - i.e. Minimize VMM interventions
1. Instruction Emulation Assist
 2. System Call intervention
 3. Shadow Table Management
 4. Non-architected TLB management
 5. Virtual interval timer
 6. Exclusive hardware access
 7. ...

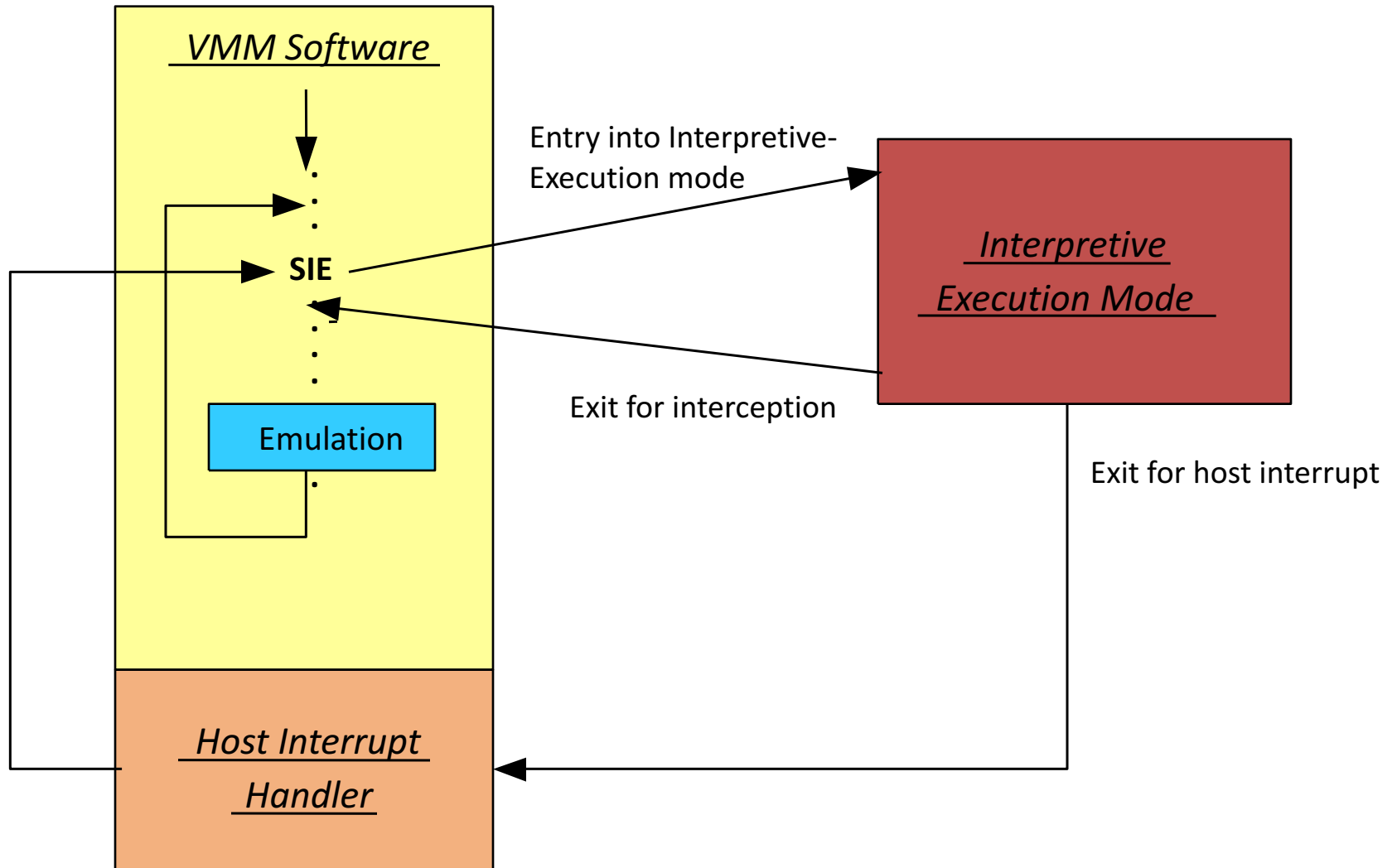
Instruction Emulation Assist (IEA)

- Avoid VMM intervention when a privileged instruction has to be executed
 - Translate to the hardware the functionality of the VMM
 - Do it for the most frequent and “annoying” critical instructions
- Example LPSW (IBM 370) modify state register
 - With IEA, the hardware (through microcode) perform the required actions
 - Check if the guest has privilege (i.e., is OS code), the user accessible portion and the memory copy (of the VM) of the state register
 - A control register has to enable and disable this behavior (to differentiate VM from native mode). ISA adds a Control Register (CR6)
 - *Bit 0 VM Assist On/Off (do LPSW from microcode)*
 - *Bit 1 Virtual user/supervisor state (change the “virtual” PSR or “virtually” trap)*
 - Bit 4 SVC handling On/Off
 - Bit 5 Shadow table fixup On/Off
 - Bit 7 Virtual interval timer assist
 - Bits 8-28 address of VM pointer list
 - CR6 Set by VMM when Guest is dispatched
- VMM and ISA should be co-designed. No OS knowledge is required

Virtual Machine Monitor Assists

- Context switch
 - Use hardware storage to store and restore in VM switches
- Decoding privileged instructions
 - Avoid VMM soft-“decode” of a privileged instruction
 - Not instruction Emulation Assist (i.e. VMM intervention occurs), just ease the VMM overhead
- Virtual Interval Timer
 - OS is designed with time in mind: OS ticks, scheduler, responsiveness, etc...
 - VMM can “emulate” to the guest-time using soft-counters: imprecise and resource consuming
 - Hardware assisted: use a real timer when the guest is running and fake it when not
- Adding new instructions to the ISA
 - According usage, some “common” operations of the VMM can be mapped to hardware
 - IBM/370: Page lock, page unlock, Translate VA and test for shared page, invalidate segment/page table, etc...

Entry and Exit from IE mode



IBM Interpretive Execution Facility

- Provides a way to execute most of the VMM functions in hardware
- Function of VMM separated between hardware and software
 - Cleaner separation compared to earlier VM assists
- Advantages of interpretive execution
 - Better performance
 - Better predictability of performance
 - Applicable for all types of guest operating systems
- Key instruction: SIE (Start Interpretive Execution)
 - Used by VMM to give control to hardware
 - Architectural state of VM in table accessible to hardware
 - Privileged instructions interpreted in hardware
 - Occasionally need to get back to the software part of the VMM (I/O or infrequent instructions)

Improving performance of the Guest/OS

- The *classical* concept is guest/OS shouldn't be aware of the presence of a VMM
 - It is VMM task to handle interrupts and to ensure that privileged insn do the same than in a privileged environment
 - Perfect from the point of view of abstraction
- But, if the guest/OS is aware of the situation, it might be useful:
 - To relegate some functionality to VMM
 - To provide VMM with some hints to improve performance: this exchange of information is called **Handshaking**
 - IBM/370 provides the insn DIAGNOSE to do it

Handshaking: examples of use

- Non paged mode
 - Run the guest/OS without dynamic address translation: just virtual addresses
 - Translation is handled by VMM. *“Just”* shadow PT
- Pseduo-page-fault handling
 - Prevent VMM to replace silently a page allocated by a VM
 - Match real page faults with virtual page faults
- Spool Files
 - Connect spool files in the VMM with spool files in guest/OS
- Inter-VM- communication
 - Avoid “I/O” path when two VM in the same host want to communicate
 - Not only reduce VMM overhead but even speed-up communication over a native implementation

Xen Para-virtualization

- Use handshaking approach (i.e. let the guest/OS aware) to avoid “critical” IA-32 insns
 - Rewrite the guest kernel (xenderized kernels) and transform system calls in “hypercalls”
- Also runs the guest/OS code mode in ring-1 (32-bits)
- The changes are restricted to the kernel: user level applications will run natively
- Initially only 3000 lines of Linux kernel where changed
- Since virtualization extensions in x86, this in no longer strictly true/necessary

Other optimizations

- Shadow table bypass assist
 - Use inside the guest/OS pointers to shadow-PT
 - No “real” pages: just virtual and physical pages
 - Hardware is intercepting all PT and TLB guest manipulation to reflect such changes in shadow-PT
 - Hardware pays special care with “special pages” (such as page0 in IBM/370)
- Preferred-machine assist
 - Allow the guest/OS to run in system mode
 - Hardware protection to isolate VMM from guest
- Segment sharing
 - Allow share “read-only” segments within virtual machines (reentrant code)
 - Less pressure over TLB and memory
 - Can be emulated in software by the VMM (but with expensive checking and controlling). If hardware support it

Example machine: IBM 360/370/390

- CP-67 on 360/67 in 1960s
 - First production VM implementation
 - Provided means for supporting timesharing via Multiple guest versions of CMS – single user OS
 - Used basic virtualization concepts described by Goldberg
- VM/370 (1972) led to widespread use of VMs
- Virtual Machine Assist (1974)
 - Enhancements to support VMs
- Extended Control Program Support (1978)
 - Further enhances VM support
- Handshaking
 - Lets guest/OS in on the secret
- Interpretive Execution Facility (IEF) (1991)
- Still in use in Z-Series (see <http://www.vm.ibm.com>)

X86 VIRTUALIZATION EXTENSIONS

Four Generations

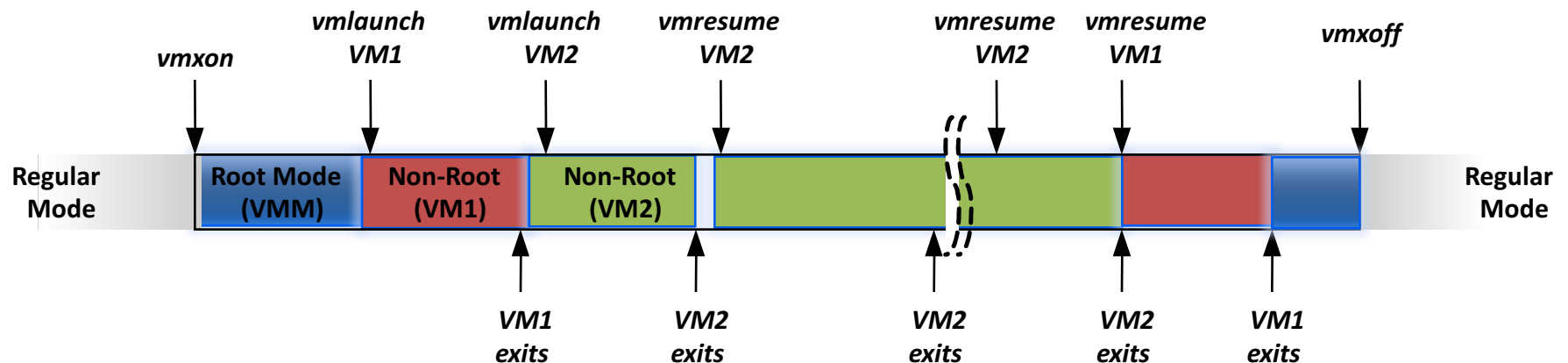
- 1st Generation: VT-x
 - Processor virtualization
- 2nd Generation: EPT
 - MMU virtualization
- 1^{st'} Generation: VT-d and VT-c
 - I/O virtualization & network virtualization (SDN)
- 3th Generation:
 - Virtualization Nesting
- Details about what is supported in each processor model in <http://ark.intel.com/Products/VirtualizationTechnology>

1st Generation: Intel VT-x (Vanderpool)

- x86 Virtualization Extensions used since 2005 (Pentium 663/667)
 - AMD did the same with AMD-v (Pacifica)
 - Conceptually similar to IBM Interpretive Execution z/VM in zSeries systems
- New VMX mode
 - Two privilege levels: root and non-root
- Root level
 - Similar to conventional x86
 - Plus new VMX instructions
 - VMM runs in root level
- Non-root level
 - Limited control of resources
 - Including when in ring 0
 - Guest OS plus apps runs in non-root level

VT-x Operation

- Transition from normal mode to VMX root mode via **vmxon** instruction
- VMM in root level, sets up the environment for each VM and initiates the virtual machine via **vmlaunch** instruction
- Attempts to modify resource cause return to root level
- Explicit **vmcall** causes return to root mode
- **vmresume** instructions causes return to guest in non-root mode
- **vmxoff** instruction causes exit from VMX mode



VT-x Capabilities

- Root mode eliminates need to run all guest code in user mode (no emulation needed)
 - VMM runs in root mode
 - For code regions with no critical instructions, HW “can be” as fast as in regular state (not really)
- VT-x HW maps state-holding data elements directly to native structures during VM execution.
 - VMCS (virtual machine control structure) encapsulates VM state
 - HW implementation can take over loading and unloading state
 - No need for VMM to perform data manipulation of state info from memory (the data is directly accessed by the processor)
- Eliminates the need for paravirtualization, code patching, etc...
 - Allows standard versions of OSes to be used as guests
 - Still `vmcall` instruction, can be used to pass hints and data to the VMM (if required)

VMCS (Virtual Machine Control Structure)

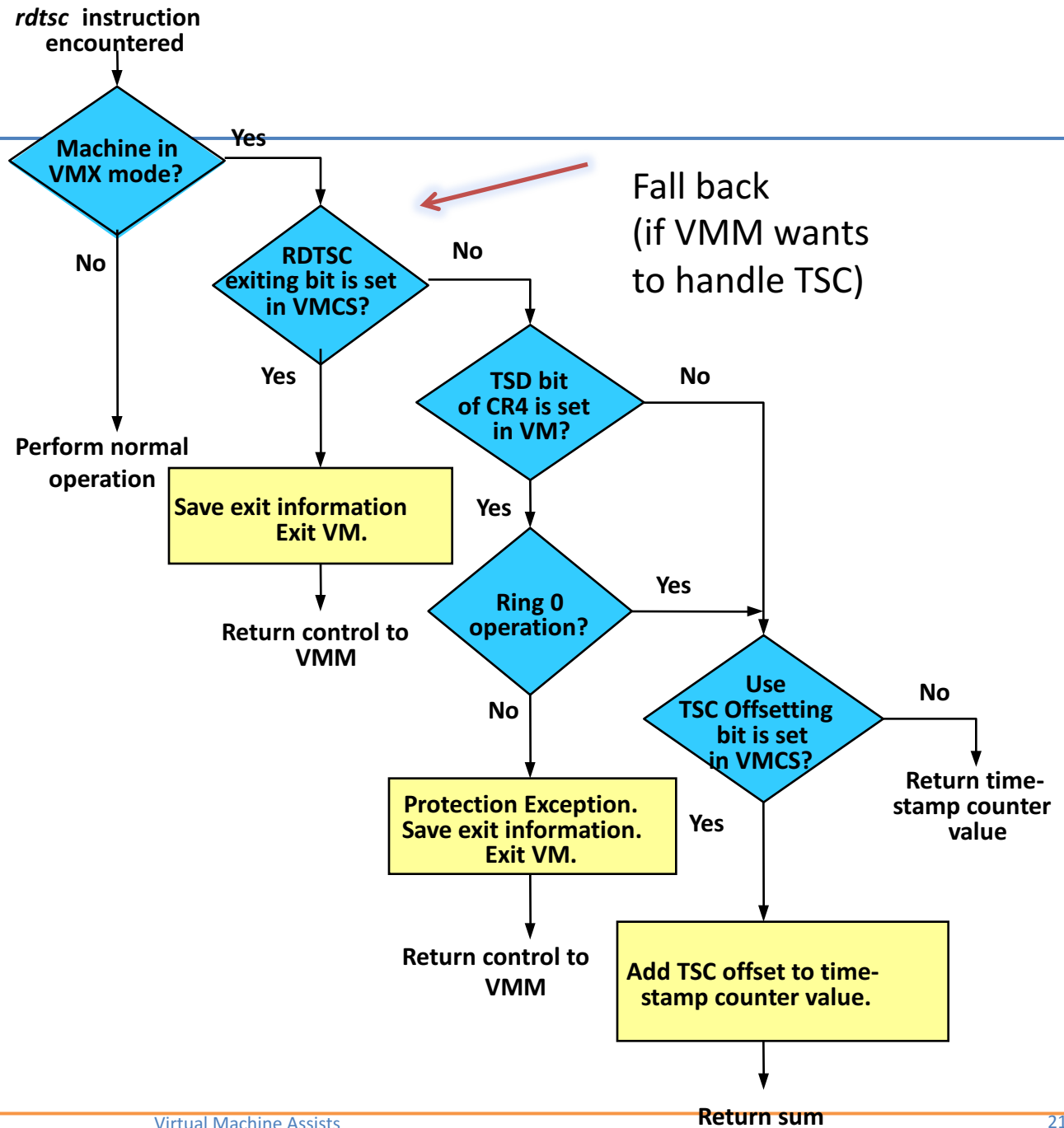
- Can be manipulated by HW or SW in root mode
 - VMM is implementation-dependent
- Aligned on 4KB boundary identifies the current running VM
- Each VM has a pointer to its VMCS through the VMPTR register (one active per logical processor)
 - Load VMPTR with `vmptwld` instruction
 - Read VMCS with `vmread` ; Write VMCS with `vmwrite`
- Keeps architected state of the VM: “shadow” copies of control register, segment register, etc...

State Area	Guest State	Register State
	Host State	Interruptibility State
Control Area	VM Execution Controls	Register State
		Pin-based Execution Controls
		Processor-based Execution Controls
		Bitmap Fields
		etc.
	VM Exit Controls	Control Bitmap
		MSR Controls
	VM Entry Controls	Control Bitmap
		MSR Controls
		Controls for Event Injection
VM Exit Information	Basic Information	VM-Exit Information
		Vectoring Event Information
	Other Exit Information	Due to Event Delivery
		Due to Instruction Execution

Critical Instructions

- Programmable VM exit conditions given in VMCS
 - E.g., which instructions should cause exit to VMM
- **Example:** Read Time Stamp Counter (RDTSC)
 - Reads *Time-stamp register* IA32_TIME_STAMP_COUNTER (a MSR or Model-specific Register) in a GP registers
 - Works in any mode if TSD (*Time-stamp Disable*) bit in control register 4 (CR4) is off
 - If TSD is on, RDTSC does:
 - If ring == 0, ignores TSD
 - If ring != 0, traps (*protection mode exception*)

RDST



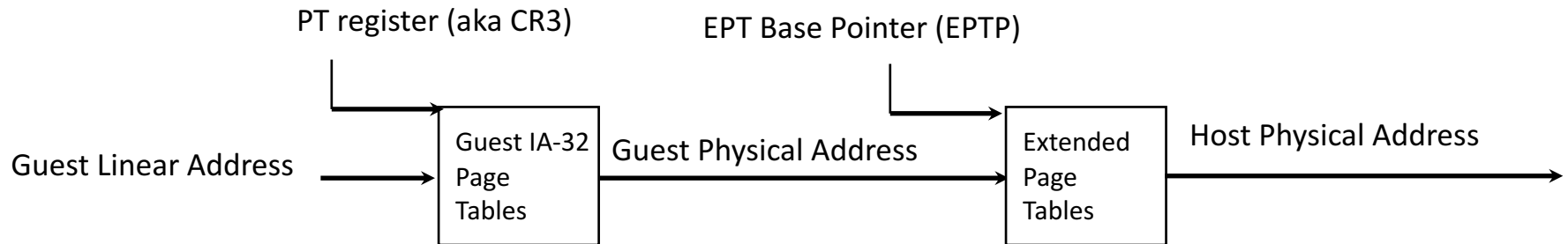
2nd Generation: MMU virtualization

- Intel EPT or AMD NPT
 - Intel Nehalem, AMD Barcelona
- A VMM must protect host physical memory at all cost!
 - Multiple guest operating systems share the same host physical memory
 - VMM typically implements protections through “page-table shadowing” in software
- Page-table shadowing might accounts for a large portion of virtualization overheads
 - VM exits due to: page faults, invalidate TLB entry, Changes in CR3,....

Extended Page Table

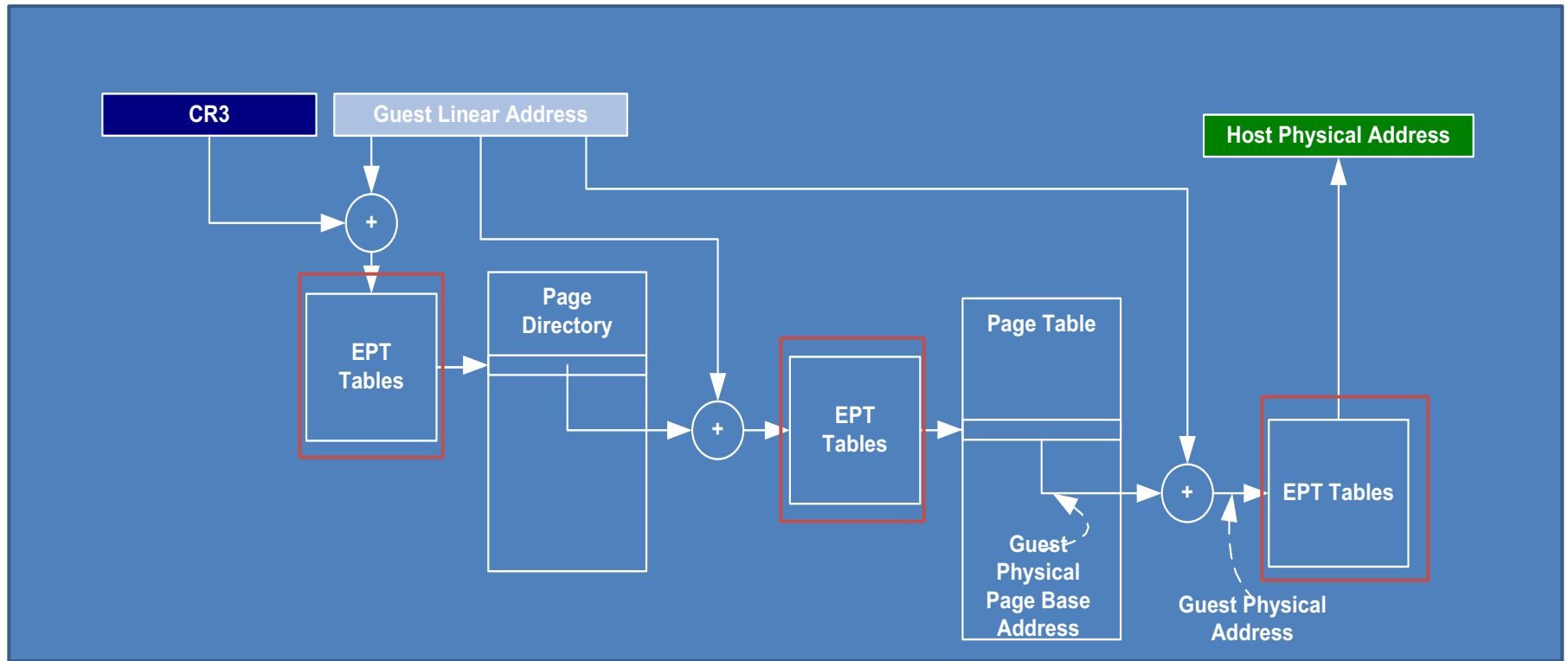
- Concept of Extended Page Table (EPT) :
 - Instead of walking along with only one page table hierarchy, EPT technique implement one more page table hierarchy.
 - One page table is maintained by guest/OS, which is used to generate guest physical address.
 - The other page table is maintained by VMM, which is used to keep guest physical address under “control”
 - For each PT manipulation, EPT will transparently “translate” guest physical address into host physical address

What Is EPT?



- Extended Page Table
- A new page-table structure, under the control of the VMM
 - Defines mapping between guest- and host-physical addresses
 - EPT base pointer (new VMCS field) points to the EPT page tables
 - EPT (optionally) activated on VM entry, deactivated on VM exit
- Guest has full control over its own page tables
 - No VM exits due to guest page faults, TLB manipulation, etc...

EPT Translation: Ugly Details



- All guest-physical memory addresses go through EPT tables
 - CR3, PDE, PTE, etc...
- Above example is for 2-level table for 32-bit address space
 - Translation possible for other page-table formats (e.g., PAE)

TLB management

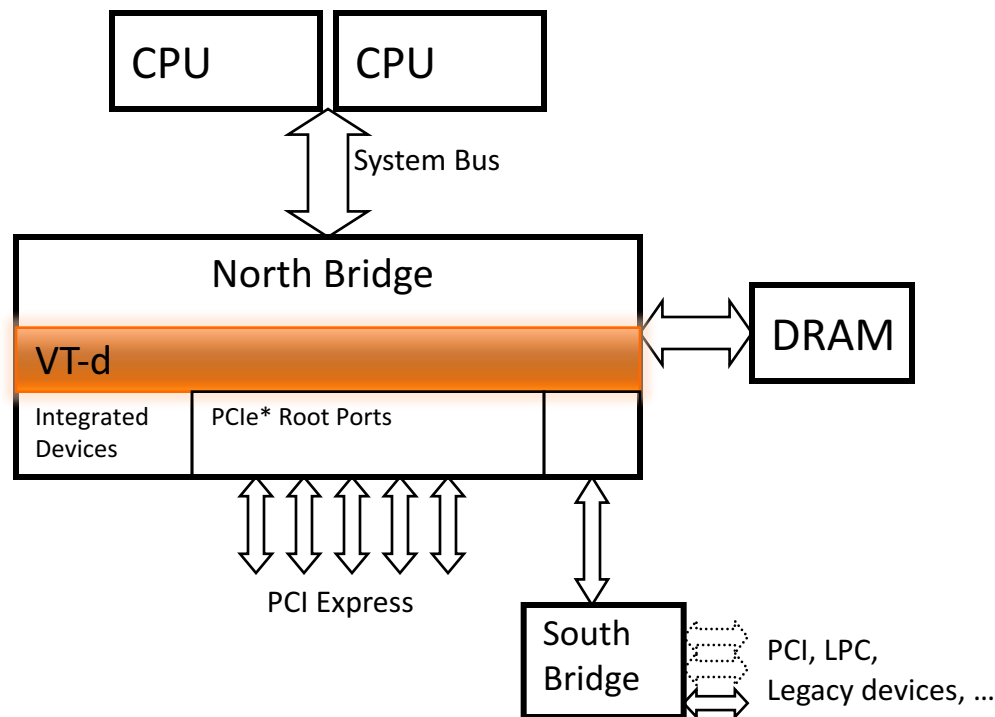
- 1st generation force TLB flush in each VMX transaction (**vmxon**, **vmlaunch**, **vmresume**, etc...)
- 2nd generation introduced the VPID
 - New 16-bit virtual-processor-ID field (VPID) field in VMCS
 - VMM allocates unique value for each guestOS
 - VMM uses VPID of 0x0000, no guest can have this VPID
 - TLB is tagged with VPID (Similar to ASID but at VM level)
 - No need for TLB flush at **vmexit**
 - VPID is managed by VMM
 - Freed at VM termination

1st Generation: I/O virtualization

- Intel VT-d and AMD IOMMU
- I/O device assignment
 - VM owns real device
- DMA remapping
 - Support address translation for DMA
- Interrupt remapping
 - Routing device interrupt

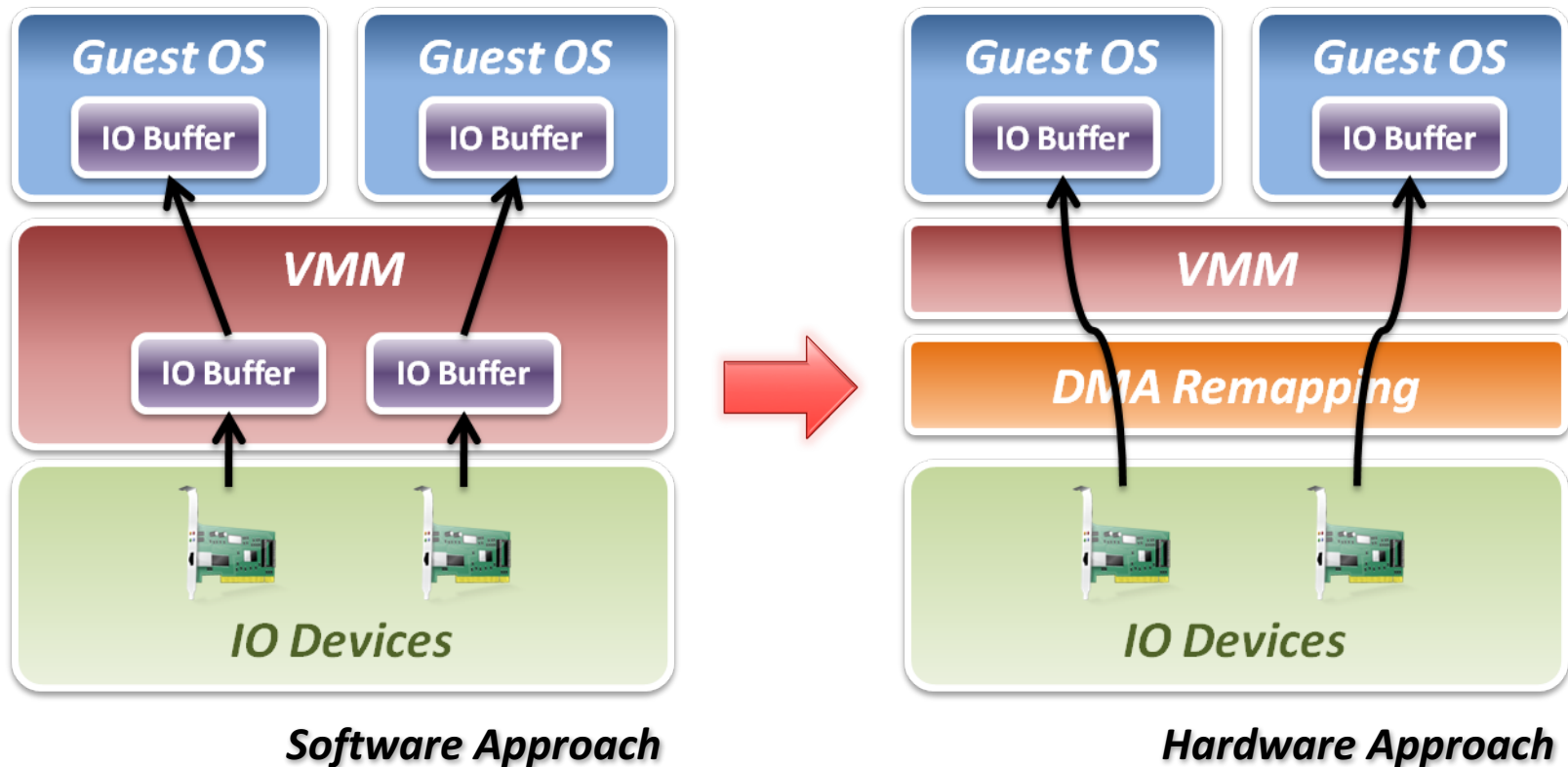
VT-d Overview

- VT-d is platform infrastructure for I/O virtualization
 - Defines architecture for DMA remapping
 - Implemented as part of platform core logic
 - It is supported broadly in Intel server and client chipsets



Intel VT-d

- Add DMA remapping hardware component.



VT-d Usage

- Basic infrastructure for I/O virtualization
 - Enable direct assignment of I/O devices to unmodified or paravirtualized VMs
- Improves system reliability
 - Contain and report errant DMA to software
- Enhances security
 - Support multiple protection domains under SW control
 - Provide foundation for building trusted I/O capabilities
- Other usages
 - Generic facility for DMA scatter/gather
 - Overcome addressability limitations on legacy devices

VT-c (Network)

- VMDq
 - Multiple queue pairs for partitioning
 - Filters a specific VM's unicast packets into individual receive queues
 - Such as MAC filtering, VLAN filtering
 - Ensures transmit fairness between VMs
 - Prevents head-of-line blocking
- PCI-SIG Single Root I/O Virtualization (SR-IOV) allows partitioning of a single Server Adapter port into multiple virtual functions.
 - Use these virtual ports to create with native-performance and isolated connections to virtual machines
- Using switches SDN capable (Software-defined networking), a fully virtualized data-center!
 - SDN is like computing virtualization but at networking level

3th Generation: Virtualization Nesting

- VM nesting is interesting for
 - Cloud development and cloud outsourcing
 - Added flexibility (eg. XP on Win 7/8)
- Since Haswell, hardware partial support for accelerated nesting
 - Virtual EPT
 - VMCS shadowing
- Latest version of Xen supports nesting
 - Still a big penalty in performance
- Interesting topic for research