CS 60038: Advances in Operating Systems Design

Virtualization

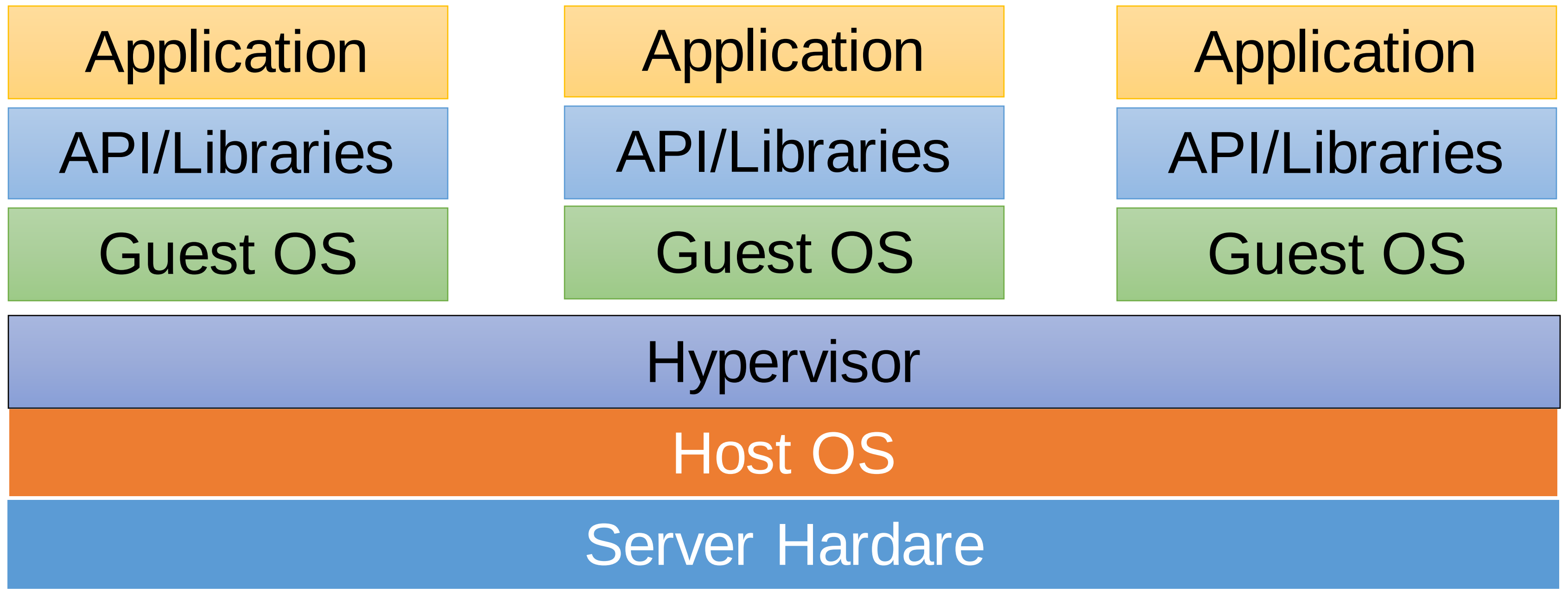
Department of Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

What is Virtualization?

- A technology to create virtual representations of servers, storage, networks, or other physical machines
  - Mimics the function of physical hardware to run multiple virtual machines simultaneously on a single physical machine
  - Efficient use of hardware resources by dividing and distributing the resources across multiple users while maintaining security and confidentiality of individual user's information
  - Provides flexibility of resource usage – creates an abstraction of the physical resources available

Virtualization on top of Traditional OS



Types of Virtualization

- **Process virtualization**
  - The virtual machine supports an application binary interface (ABI) that contains user instructions and system calls – virtualization of a single process
  - Examples: Java Virtual Machines, Dynamic translators

- **Namespace Virtualization**
  - Lightweight virtualization, multiple logical VMs that share the same OS kernel
  - Example: Chroot++ that isolates VMs by partitioning all objects into namespaces, Linux containers

Types of Virtualization

- **System Virtualization**
  - Virtualizes the OS and the apps
  - The VMs support a complete instruction set architecture including user and system instructions
  - Example: Classic VMs with a guest OS

Popek and Goldberg Virtualization Requirements

- Set of conditions sufficient for a computer architecture to support system virtualization efficiently
- Introduced by Gerald J. Popek and Robert P. Goldberg in their 1974 article "Formal Requirements for Virtualizable Third Generation Architectures"
- **Virtual Machine Monitors (VMM):** Software that creates and runs virtual machines (VMs)
    - Also called Hypervisors; we'll discuss in more details later

Popek and Goldberg Virtualization Requirements

- **Equivalence / Fidelity**
  - A program running under the VMM should exhibit a behavior essentially identical to that demonstrated when running on an equivalent machine directly.
- **Resource control / Safety**
  - The VMM must be in complete control of the virtualized resources.
- **Efficiency / Performance**
  - A statistically dominant fraction of machine instructions must be executed without VMM intervention.

The Core of Virtualization

- A process that allows a computer to share its hardware resources with multiple digitally separated environments
  - Each virtualized environment runs within its allocated memory, processing power and storage
  - Each virtual environment is logically separated from the others, and cannot access the resources allocated to other virtual environments – thus, provides an **isolated environment** from other virtual instances (called **sandboxing**)
  - Sandboxing helps in providing the security supports for each such virtual environments (we'll later discuss this in detail)

Core Concepts

- **Virtual Machines (VMs)**
  - A software-defined computational environment that runs on a physical computational environment (desktop computers or servers) with a separate operating system and computing resources
  - The physical computational environment is called a host machine
  - The virtual machines are called the guest machines
  - Multiple VMs can run on a single physical machine

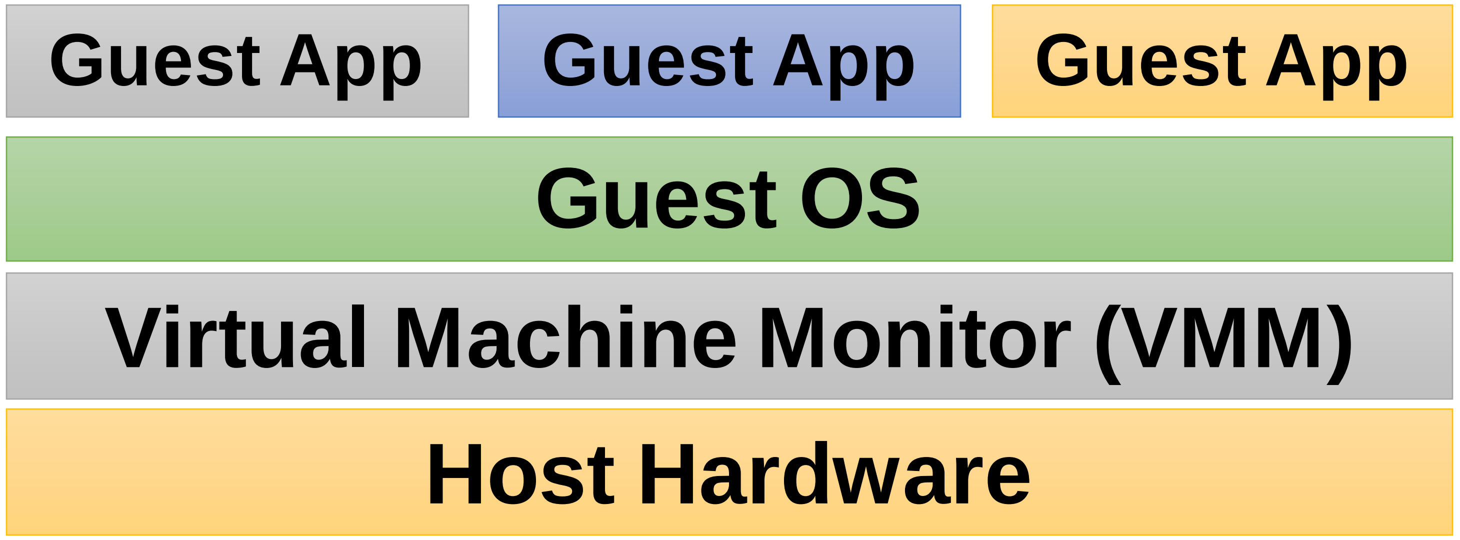
Core Concepts

- **Hypervisors (also called the Virtual Machine Monitors)**
  - A software component or middleware that provides an abstraction of the physical hardware to the virtual machines.
  - Hypervisor manages multiple VMs on a single physical machine
  - Ensures that each VM gets the allocated resources and does not interfere with the other VMs running on the same physical machine (**provides sandboxing across the VMs**)

Type of Hypervisors

- **Type 1 Hypervisor (Bare Metal Hypervisor)**
  - A hypervisor program installed directly on the computer's hardware instead of the operating system
  - Have better performance, are typically used by enterprise applications on the servers or data centers

Type 1 Hypervisors



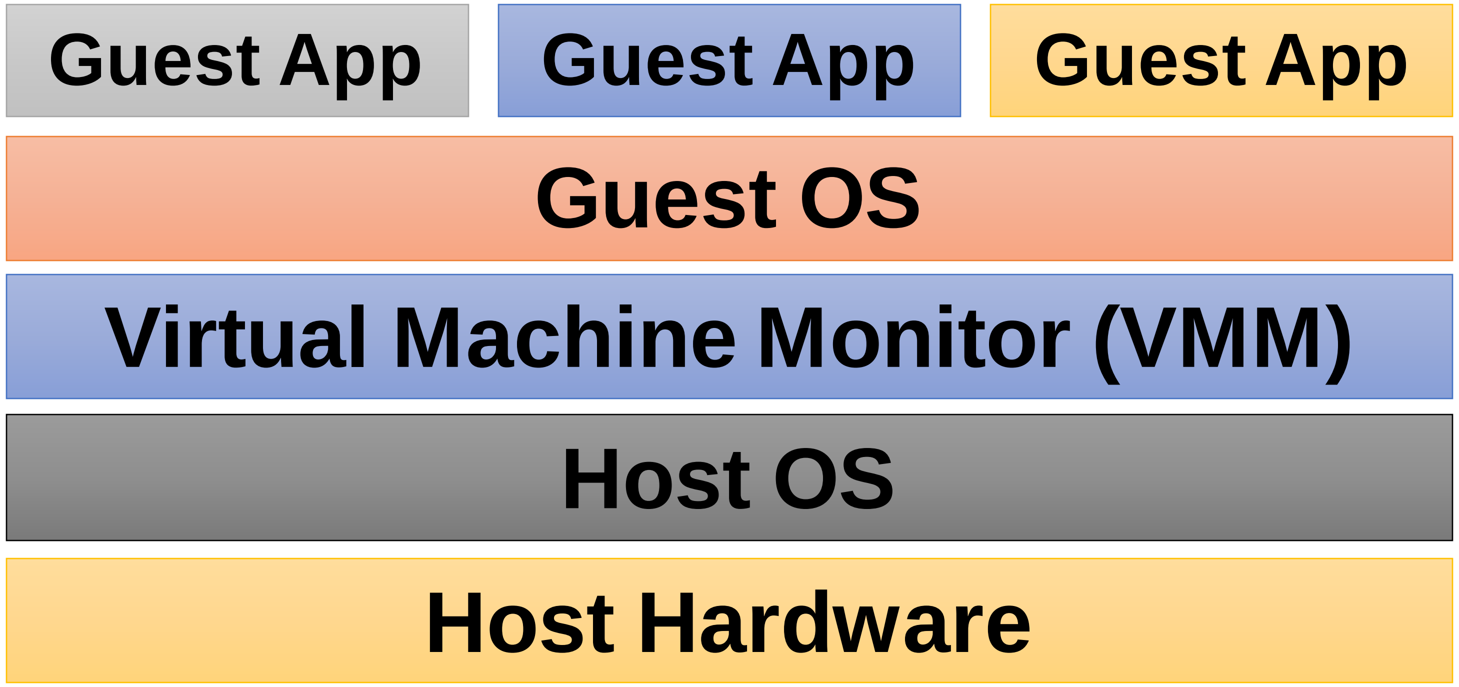
Examples:

- VMware hypervisors like vSphere, ESXi and ESX
- Microsoft Hyper-V
- Oracle VM Server
- Citrix Hypervisor

Type of Hypervisoprs

- **Type 1 Hypervisor (Bare Metal Hypervisor)**
  - A hypervisor program installed directly on the computer's hardware instead of the operating system
  - Have better performance, are typically used by enterprise applications on the servers or data centers
- **Type 2 Hypervisor (Hosted Hypervisor)**
  - A hyperviosr program installed on a host OS
  - Suitable for end-user computing

Type 2 Hypervisors



Examples:

- VMware Fusion
- Oracle VirtualBox

Application Binary Interface (ABI)

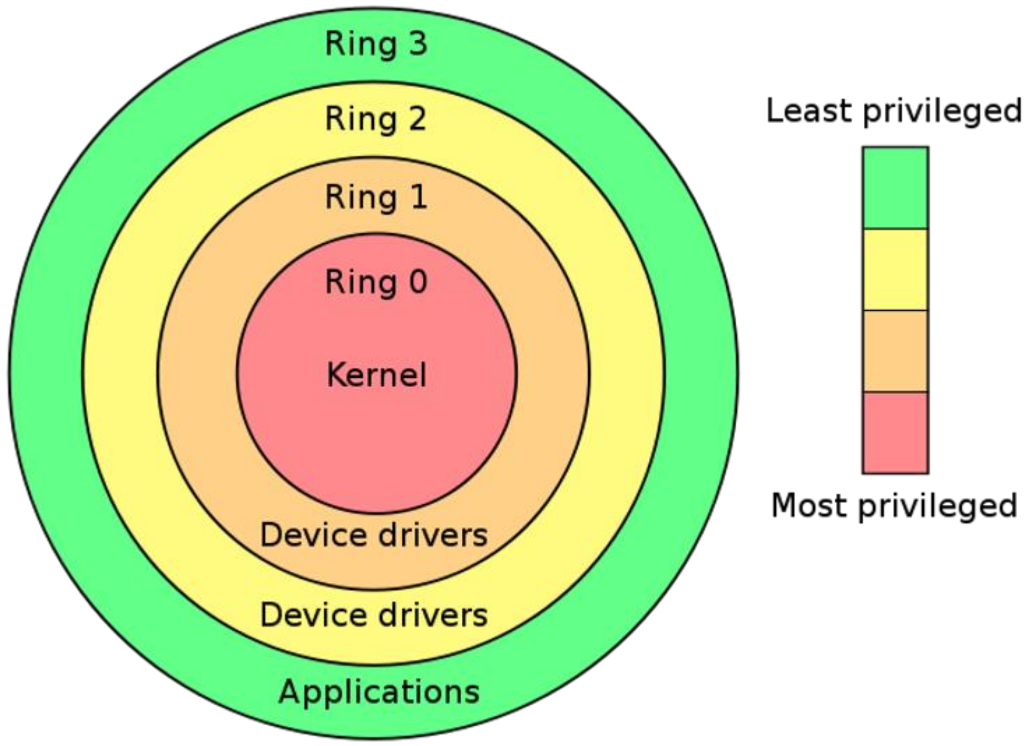
- An interface between two binary program modules
  - One of these modules is often a library or the OS facility
- Defines how data structures or computational routines are accessed in machine code
  - API defines this access in source code, which is a relatively high-level, hardware-independent, often human-readable format

Application Binary Interface (ABI)

- Details covered by an ABI include the following:
  - Processor instruction set, with details like register file structure, stack organization, memory access types, etc.
  - Sizes, layouts, and alignments of basic data types that the processor can directly access
  - Calling convention, which controls how the arguments of functions are passed, and return values retrieved; for example, it controls the following:
  - Whether all parameters are passed on the stack, or some are passed in registers
  - Which registers are used for which function parameters
  - Whether the first function parameter passed on the stack is pushed first or last
  - Whether the caller or callee is responsible for cleaning up the stack after the function call
  - How an application should make system calls to the operating system, and if the ABI specifies direct system calls rather than procedure calls to system call stubs, the system call numbers
  - In the case of a complete operating system ABI, the binary format of object files, program libraries, etc.

Protection Rings

- A mechanism to protect data from faults and malicious behavior
  - Thus supports both fault-tolerance and security
  - Supports the layer of privilege within the architecture of an OS



Modes of Protection Ring

- **Supervisor Mode**
  - Allows execution of all instructions (including priviledged instructions)
  - Give access to different address space, memory management hardware and other peripherals
  - OS runs in this mode
- **Hypervisor Mode**
  - Modern CPUs offer x86 virtualization instructions for hypervisor to control Ring 0 hardware access.
  - Insert new privilege level below Ring 0, and new machine code instructions at Ring 1, intended to be executed by the hypervisor (**hardware-assisted virtualization**)

CPU Virtualization – Key Issues

- **Protection Levels**: Ring 0 (Most privileged, privileged instructions cannot be executed outside Ring 0), Ring 3 (User mode)
- **Requirements for efficient virtualization:**
  - Privileged Instructions: Trap if executed in user mode
  - Sensitive instructions: Those affects important system states
  - If a sensitive instruction is privileged, we can support efficient "trap and emulate" approach
  - Virtualized instructions can be emulated as native execution, plus exception handling code that emulates privileged instructions

CPU Virtualization: Key Issues

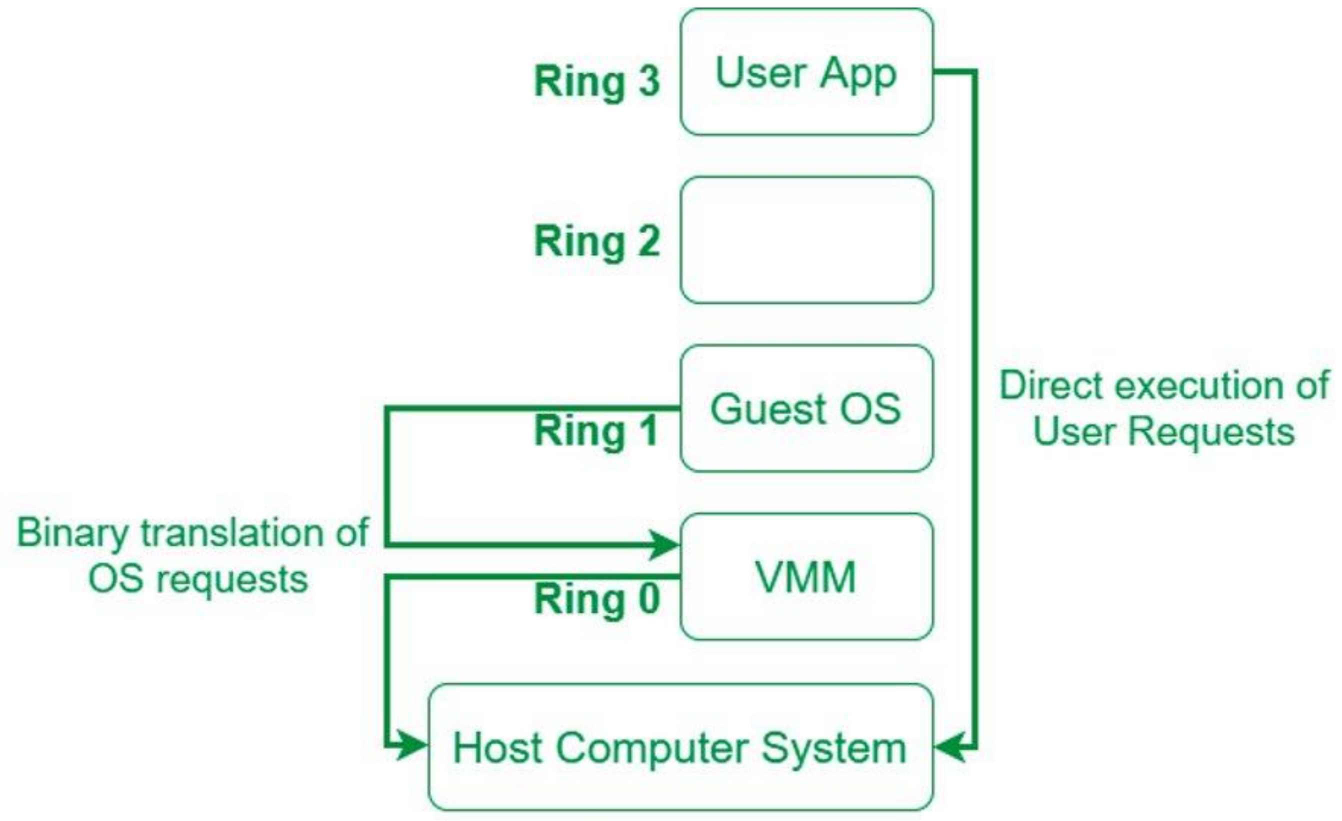
- For x86 architecture, not all sensitive instructions are privileged
  - Some instructions exhibit different behaviors in user and privileged modes

- For example, If `CR4.UMIP=1` is set, then the `SGDT`, `SIDT`, `SLDT`, `SMSW` and `STR` instructions can only run in Ring 0.

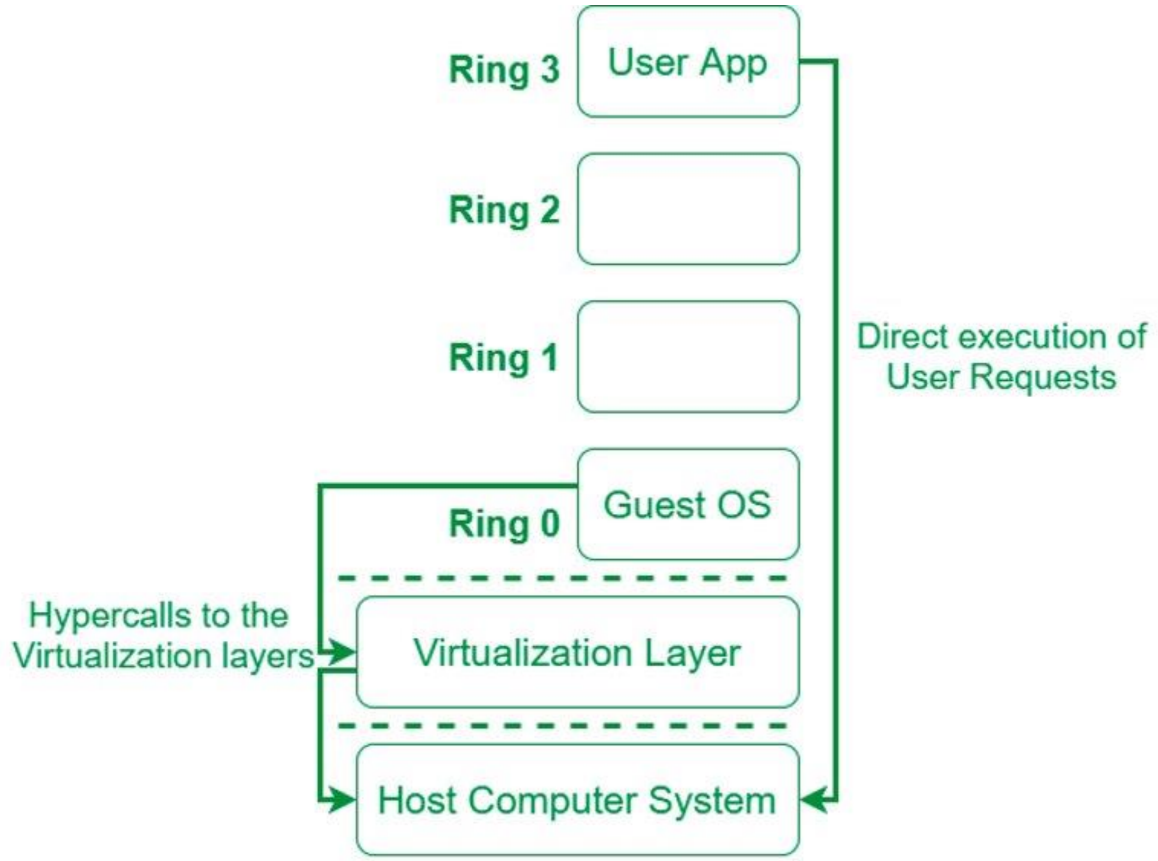
Full vs Para Virtualization

- **Full Virtualization**
  - Application running on the guest OS is completely abstracted from the underlying hardware
  - OS is unaware that it is a guest, and the hypervisor translates all OS calls to the actual hardware access
- **Para-virtualization**
  - The guest OS is recompiled prior to installation inside a VM
  - Does not implement full isolation of OS but rather provides a different API (hypercalls) which is utilized for actual hardware access

Full vs Para Virtualization


Ring 3
User App
Ring 2
Ring 1
Guest OS
Ring 0
VMM
Binary translation of OS requests
Direct execution of User Requests
Host Computer System

Full Virtualization



Paravirtualization

Image Source: https://www.geeksforgeeks.org/difference-between-full-virtualization-and-paravirtualization/

Para Virtualization

- Not fully interface compatible, but provides better performance
  - Guest OSes must be modified to use VMM's interface
  - ABI remains unchanged
  - Therefore, applications need not to be modified

- Guest OS is aware of virtualization (not in case of full virtualization)
  - Privileged instructions are replaced by hypervisor calls (hypercalls)
  - No need for binary translations

Hardware Assisted Virtualization

- Use of a computer's physical components (hardware) to support the software that creates and manages VMs.
- Both Intel and AMD support virtualization for their modern processors
  - Intel supports virtualization through the VT-x technology; the CPU flag for the VT-x capability is **V**irtual **M**achine e**X**tension (VMX)
  - AMD developed its first generation of virtualization extensions under the codename “*Pacifica*” and initially published them as *AMD Secure Virtual Machine (SVM);* later marketed them under *AMD Virtualization*, abbreviated ***AMD-V***.
- Notably, Intel and AMD use different instruction sets for virtualization

Hardware Assisted Virtualization

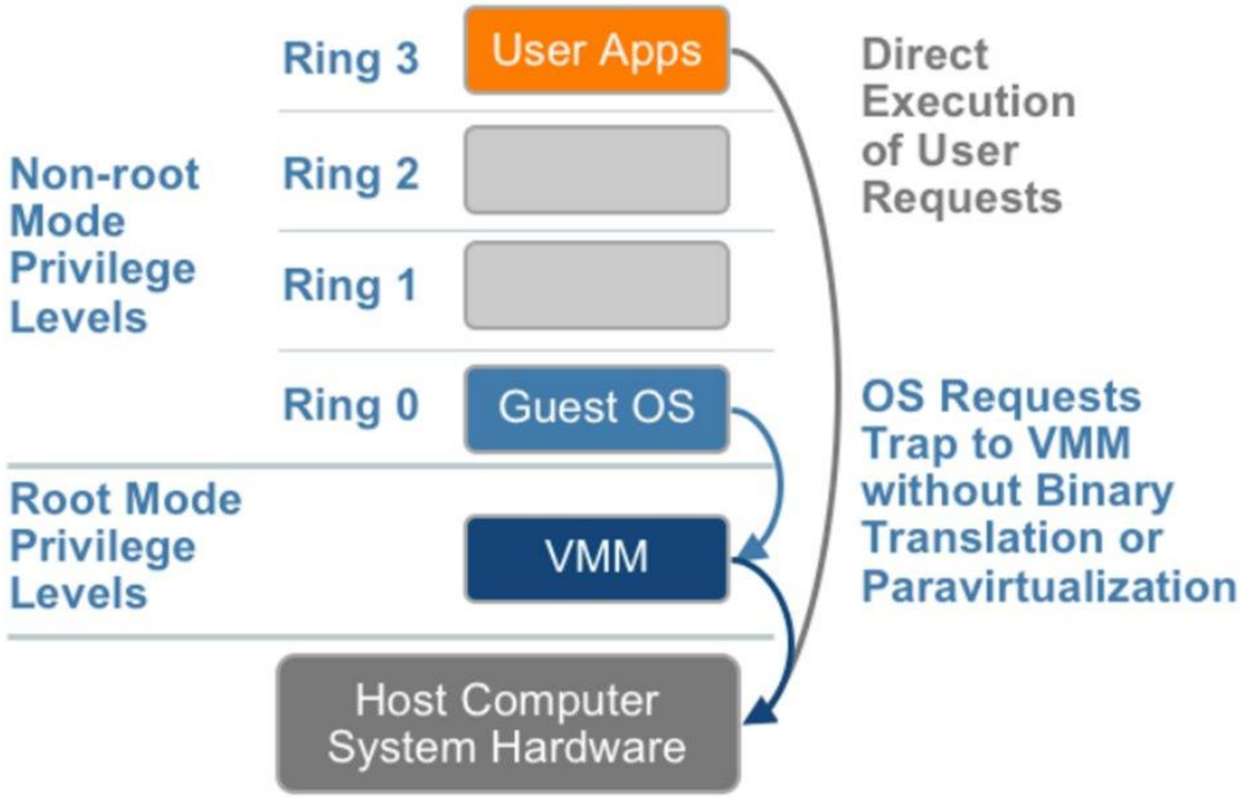

Ring 3
User Apps
Direct Execution of User Requests
Non-root Mode Privilege Levels
Ring 2
Ring 1
Ring 0
Guest OS
OS Requests Trap to VMM without Binary Translation or Paravirtualization
Root Mode Privilege Levels
VMM
Host Computer System Hardware

Image Source: https://thecustomizewindows.com/2014/09/hardware-assisted-virtualization/

Comparing Virtualization Techniques

Parameter	Full Virtualization	Para-virtualization	Hardware Assisted Virtualization
Generation	1st	2nd	3rd
Performance	Good	Better in certain cases	Fair
Used By	VMWare, Microsoft, KVM	VMWare, Xen	VMware, Xen, Microsoft
Guest OS Modification	Not needed	Updated to issue hypercalls	Not needed
Guest OS hypervisor independent?	Yes	XenLinux runs only on hypervisors	Yes
Technique	Direct execution	Hypercalls	Exit to root mode on privileged instructions
Compatibility	Excellent	Poor	Excellent

Intel VT-x

- Intel virtualization technology supports the followings:
  - **CPU Virtualization**
    - Enables abstraction of the CPU to a VM; supports live migration from one CPU to another, as well as nested virtualization
  - **Memory Virtualization**
    - Allows abstraction isolation and monitoring of memory on a per virtual machine (VM) basis
    - Allows live migration of VMs, fault tolerance
    - Supports feature like direct memory access (DMA) remapping and extended page tables

Intel VT-x

- Intel virtualization technology supports the followings:
  - **I/O Virtualization**
    - Facilitate offloading of multi-core packet processing to network adapters as well as direct assignment of virtual machines to virtual functions, including disk I/O
    - Examples: Intel® Virtualization Technology for Directed I/O (VT-d), Virtual Machine Device Queues (VMDQ), Single Root I/O Virtualization (SR-IOV, a PCI-SIG standard), and Intel® Data Direct I/O Technology (Intel® DDIO) enhancements, etc.
  - **Graphics Virtualization**
    - Allows VMs to have full and/or shared assignment of the graphics processing units (GPU) and the video transcode accelerator engines integrated in Intel system-on-chip products.
    - Enables usages such as workstation remoting, desktop-as-a-service, media streaming, and online gaming.

Basic Concepts in Intel VT-x

- **Virtual Machine Monitor (VMM)**:
  - VMM acts as a host and has full control of the processor(s) and other platform hardware.
  - A VMM is able to retain selective control of processor resources, physical memory, interrupt management, and I/O.

- **Guest Software:**
  - Each virtual machine (VM) is a guest software environment.

Basic Concepts in Intel VT-x

- **VMX Root/Non-root Operations:**
  - VMM runs in VMX root operation (Similar to the traditional OS with no virtualization – CPU executes priviledged instructions from supervisor mode)
  - Guest software/OS runs in VMX non-root operation
- **VMX Transitions:**
  - Transitions between VMX root operations and VMX non-root operations
- **VM Entries**
  - Transition from VMX root operations to VMX non-root operations
- **VM Exits:**
  - Transition from VMX non-root operation to VMX root operation

Basic Concepts in Intel VT-x

- **Virtual Machine Control Structure (VMCS):**
  - A data structure in memory that exists exactly once per VM (or more precisely one per each VCPU -- Virtual CPU)
  - The VMM manages the VMCS for each VM.
  - With every change of the execution context between different VMs, VMCS is restored for the current VM
  - Defining the state of the VM’s virtual processor and VMM control Guest software using VMCS
- **Extended Page Table:**
  - A mechanism that uses a second layer to convert the guest's physical address to the host's physical address

Virtual Machine Control Structure (VMCS)

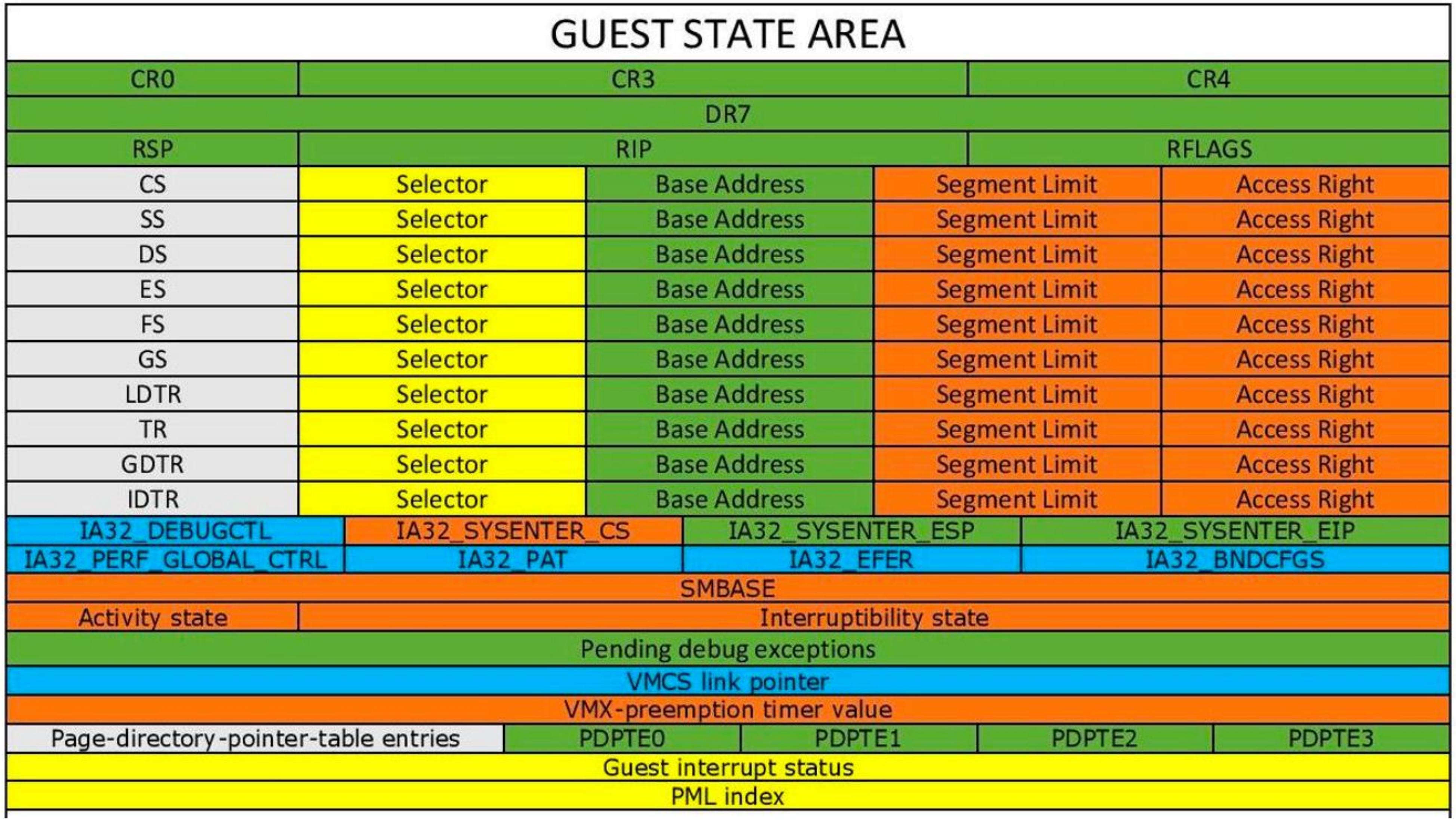
- Instead of fixing the x86 architecture to (optionally) make it Popek-Goldberg compliant and have all critical instructions trap if not run in Ring 0, Intel added the *non-root mode* in the VT-x architecture
  - Allows the system to switch the CPU state completely to that of the guest and switches back to the original host state on a certain event in the guest.

- VMCS facilitates this switching in the CPU state from guest to host and vice versa
  - 4 KB block in memory that holds the complete CPU state of both the host and the guest (segment registers, GDT and IDT pointers, certain MSRs etc.) as well as some control bits (for example, when to exit).

Virtual Machine Control Structure (VMCS)

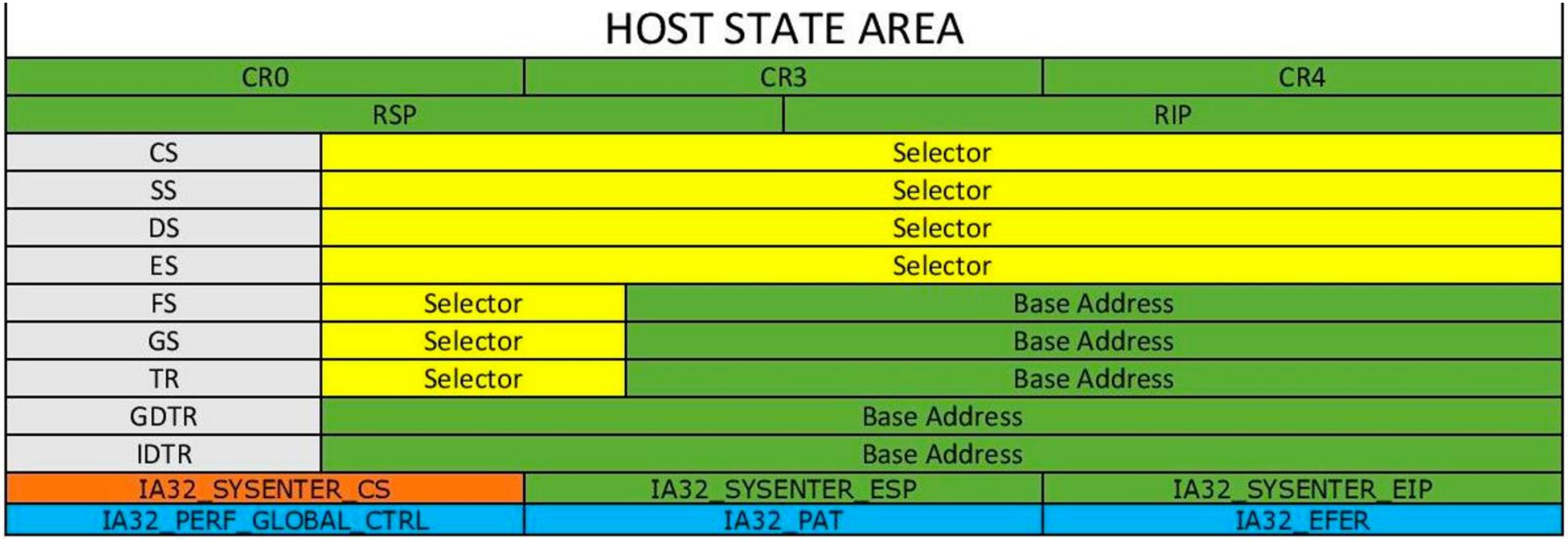
- VMCS consists of six logical groups:
  - **Guest-state area**: Processor state saved into the guest state area on VM exits and loaded on VM entries.
  - **Host-state area**: Processor state loaded from the host state area on VM exits.
  - **VM-execution control fields**: Fields controlling processor 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.

VMCS Layout – Guest State Area



GUEST STATE AREA				
CR0	CR3		CR4	
DR7				
RSP	RIP		RFLAGS	
CS	Selector	Base Address	Segment Limit	Access Right
SS	Selector	Base Address	Segment Limit	Access Right
DS	Selector	Base Address	Segment Limit	Access Right
ES	Selector	Base Address	Segment Limit	Access Right
FS	Selector	Base Address	Segment Limit	Access Right
GS	Selector	Base Address	Segment Limit	Access Right
LDTR	Selector	Base Address	Segment Limit	Access Right
TR	Selector	Base Address	Segment Limit	Access Right
GDTR	Selector	Base Address	Segment Limit	Access Right
IDTR	Selector	Base Address	Segment Limit	Access Right
IA32_DEBUGCTL	IA32_SYSENTER_CS	IA32_SYSENTER_ESP	IA32_SYSENTER_EIP	
IA32_PERF_GLOBAL_CTRL	IA32_PAT	IA32_EFER	IA32_BNDCFGS	
SMBASE				
Activity state	Interruptibility state			
Pending debug exceptions				
VMCS link pointer				
VMX-preemption timer value				
Page-directory-pointer-table entries	PDPTE0	PDPTE1	PDPTE2	PDPTE3
Guest interrupt status				
PML index				

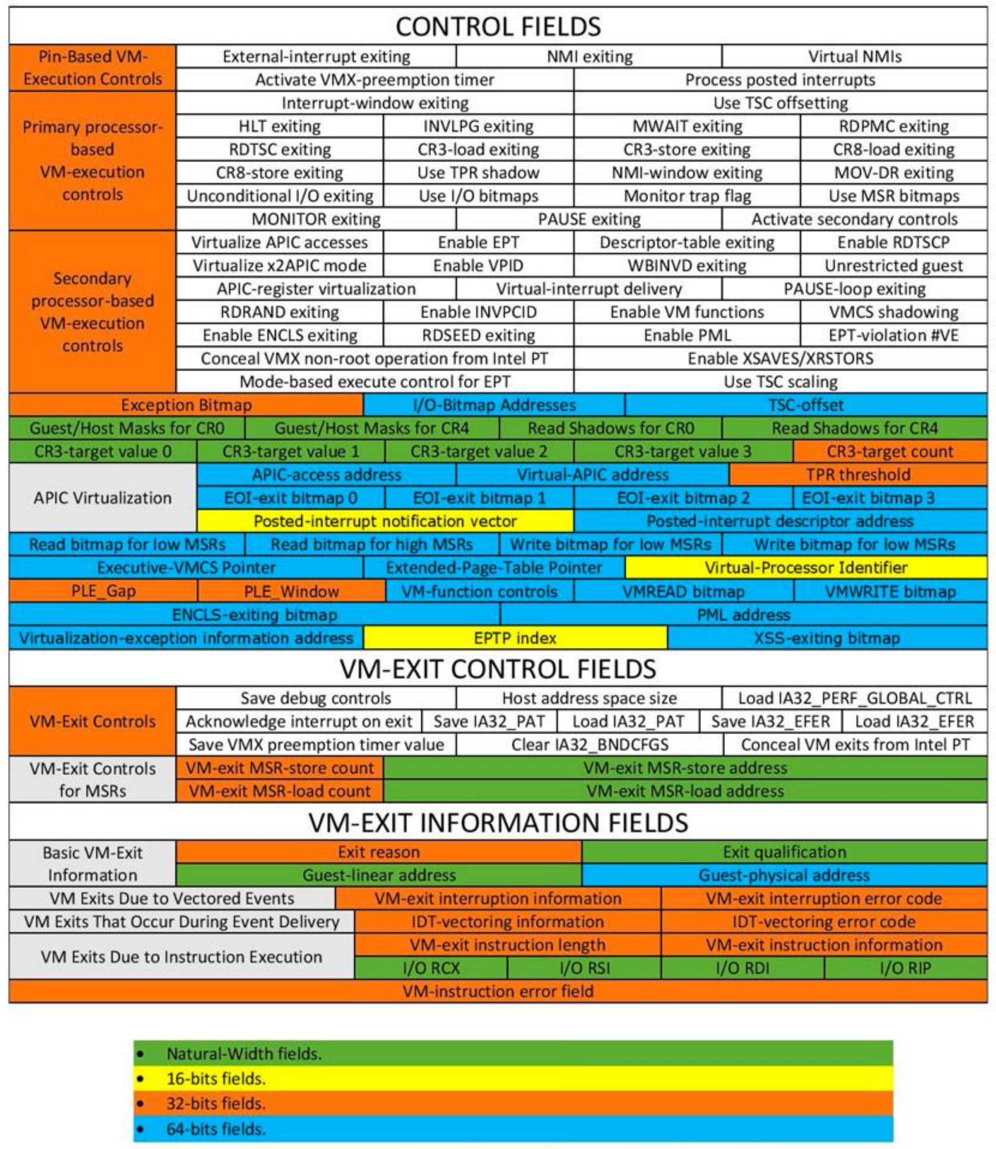
VMCS Layout – Host State Area



HOST STATE AREA

CR0	CR3	CR4
RSP	RIP	
CS	Selector	
SS	Selector	
DS	Selector	
ES	Selector	
FS	Selector	Base Address
GS	Selector	Base Address
TR	Selector	Base Address
GDTR	Base Address	
IDTR	Base Address	
IA32_SYSENTER_CS	IA32_SYSENTER_ESP	IA32_SYSENTER_EIP
IA32_PERF_GLOBAL_CTRL	IA32_PAT	IA32_EFER

VMCS Layout



CONTROL FIELDS					
Pin-Based VM-Execution Controls	External-interrupt exiting	NMI exiting	Virtual NMIs		
	Activate VMX-preemption timer	Process posted interrupts			
Primary processor-based VM-execution controls	Interrupt-window exiting	Use TSC offsetting			
	HLT exiting	INVLPG exiting	MWAIT exiting	RDPMC exiting	
	RDTSC exiting	CR3-load exiting	CR3-store exiting	CR8-load exiting	
	CR8-store exiting	Use TPR shadow	NMI-window exiting	MOV-DR exiting	
	Unconditional I/O exiting	Use I/O bitmaps	Monitor trap flag	Use MSR bitmaps	
	MONITOR exiting	PAUSE exiting	Activate secondary controls		
Secondary processor-based VM-execution controls	Virtualize APIC accesses	Enable EPT	Descriptor-table exiting	Enable RDTSCP	
	Virtualize x2APIC mode	Enable VPID	WBINVD exiting	Unrestricted guest	
	APIC-register virtualization	Virtual-interrupt delivery	PAUSE-loop exiting		
	RDRAND exiting	Enable INVPCID	Enable VM functions	VMCS shadowing	
	Enable ENCLS exiting	RDSEED exiting	Enable PML	EPT-violation #VE	
	Conceal VMX non-root operation from Intel PT	Enable XSAVES/XRSTORS			
	Mode-based execute control for EPT	Use TSC scaling			
Exception Bitmap	I/O-Bitmap Addresses	TSC-offset			
Guest/Host Masks for CR0	Guest/Host Masks for CR4	Read Shadows for CR0	Read Shadows for CR4		
CR3-target value 0	CR3-target value 1	CR3-target value 2	CR3-target value 3	CR3-target count	
APIC Virtualization	APIC-access address	Virtual-APIC address	TPR threshold		
	EOI-exit bitmap 0	EOI-exit bitmap 1	EOI-exit bitmap 2	EOI-exit bitmap 3	
	Posted-interrupt notification vector	Posted-interrupt descriptor address			
Read bitmap for low MSRs	Read bitmap for high MSRs	Write bitmap for low MSRs	Write bitmap for low MSRs		
Executive-VMCS Pointer	Extended-Page-Table Pointer	Virtual-Processor Identifier			
PLE_Gap	PLE_Window	VM-function controls	VMREAD bitmap	VMWRITE bitmap	
ENCLS-exiting bitmap	PML address				
Virtualization-exception information address	EPTP index	XSS-exiting bitmap			
VM-EXIT CONTROL FIELDS					
VM-Exit Controls	Save debug controls	Host address space size	Load IA32_PERF_GLOBAL_CTRL		
	Acknowledge interrupt on exit	Save IA32_PAT	Load IA32_PAT	Save IA32_EFER	Load IA32_EFER
	Save VMX preemption timer value	Clear IA32_BNDCFGS	Conceal VM exits from Intel PT		
VM-Exit Controls for MSRs	VM-exit MSR-store count	VM-exit MSR-store address			
	VM-exit MSR-load count	VM-exit MSR-load address			
VM-EXIT INFORMATION FIELDS					
Basic VM-Exit Information	Exit reason	Exit qualification			
	Guest-linear address	Guest-physical address			
VM Exits Due to Vectored Events	VM-exit interruption information	VM-exit interruption error code			
VM Exits That Occur During Event Delivery	IDT-vectoring information	IDT-vectoring error code			
VM Exits Due to Instruction Execution	VM-exit instruction length	VM-exit instruction information			
	I/O RCX	I/O RSI	I/O RDI	I/O RIP	
VM-instruction error field					

- 🟩 Natural-Width fields.
- 🟨 16-bits fields.
- 🟧 32-bits fields.
- 🟦 64-bits fields.

CopyLeft 2017, @Noteworthy (Intel Manual of July 2017)

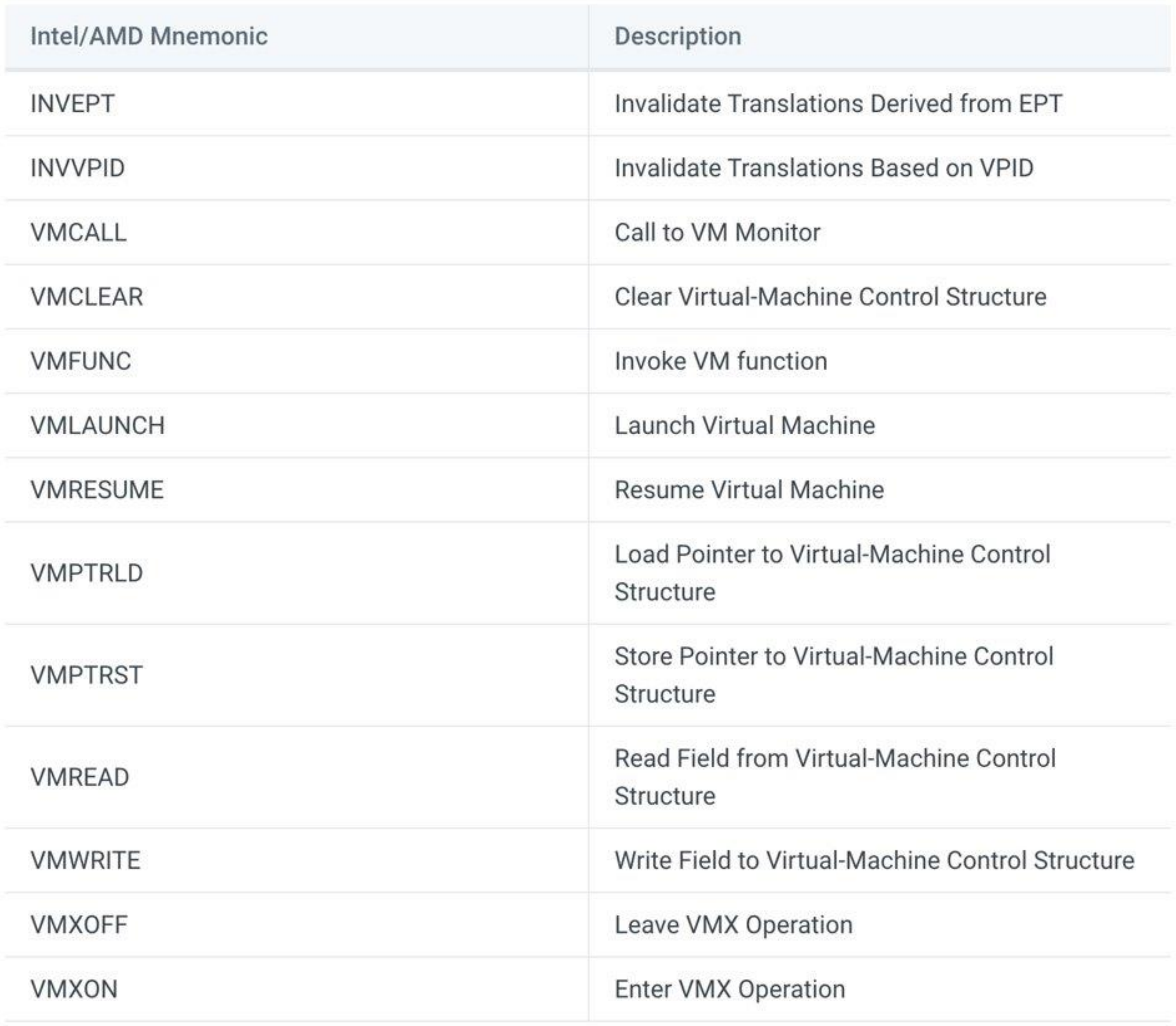
VMCS Operations

- VM entries load processor state from the guest-state area.
  - A VMM can optionally configure VM entry to follow this loading by injecting an interrupt or exception.
  - The CPU effects this injection using the guest *Interrupt Descriptor Table* (IDT), just as if the injected event had occurred immediately after VM entry
  - This feature removes the need for a VMM to emulate delivery of these events

VMCS Operations

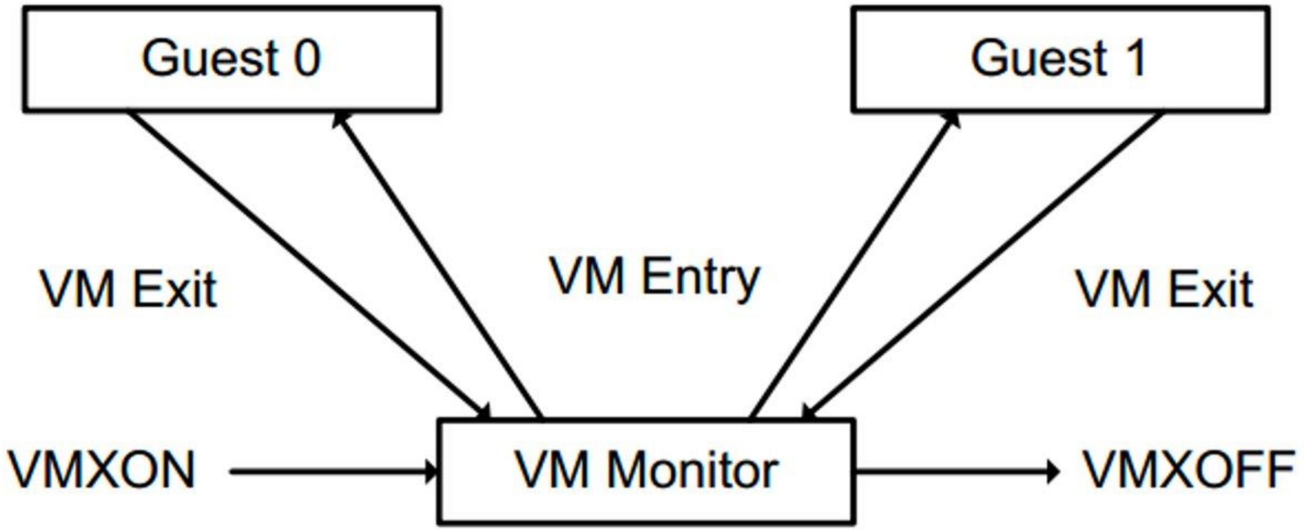
- VM exits save processor state to the guest-state area and then load processor state from the host-state area
  - All VM exits use a common entry point to the VMM.
  - To simplify the design of a VMM, every VM exit saves into the VMCS detailed information specifying the reason for the exit
  - Many exits also record an exit qualification, which provides further details
  - Example: if the MOV CR instruction causes a VM exit, the exit reason would indicate "control-register access"; the exit qualification would indicate (1) the identity of the control register (for example, CR0); (2) whether the MOV was to or from the control register; and (3) which general-purpose register was the source or destination of the instruction.

VMX Instructions



Intel/AMD Mnemonic	Description
INVEPT	Invalidate Translations Derived from EPT
INVVPID	Invalidate Translations Based on VPID
VMCALL	Call to VM Monitor
VMCLEAR	Clear Virtual-Machine Control Structure
VMFUNC	Invoke VM function
VMLAUNCH	Launch Virtual Machine
VMRESUME	Resume Virtual Machine
VMPTRLD	Load Pointer to Virtual-Machine Control Structure
VMPTRST	Store Pointer to Virtual-Machine Control Structure
VMREAD	Read Field from Virtual-Machine Control Structure
VMWRITE	Write Field to Virtual-Machine Control Structure
VMXOFF	Leave VMX Operation
VMXON	Enter VMX Operation

VMM Life Cycle



VMM Life Cycle

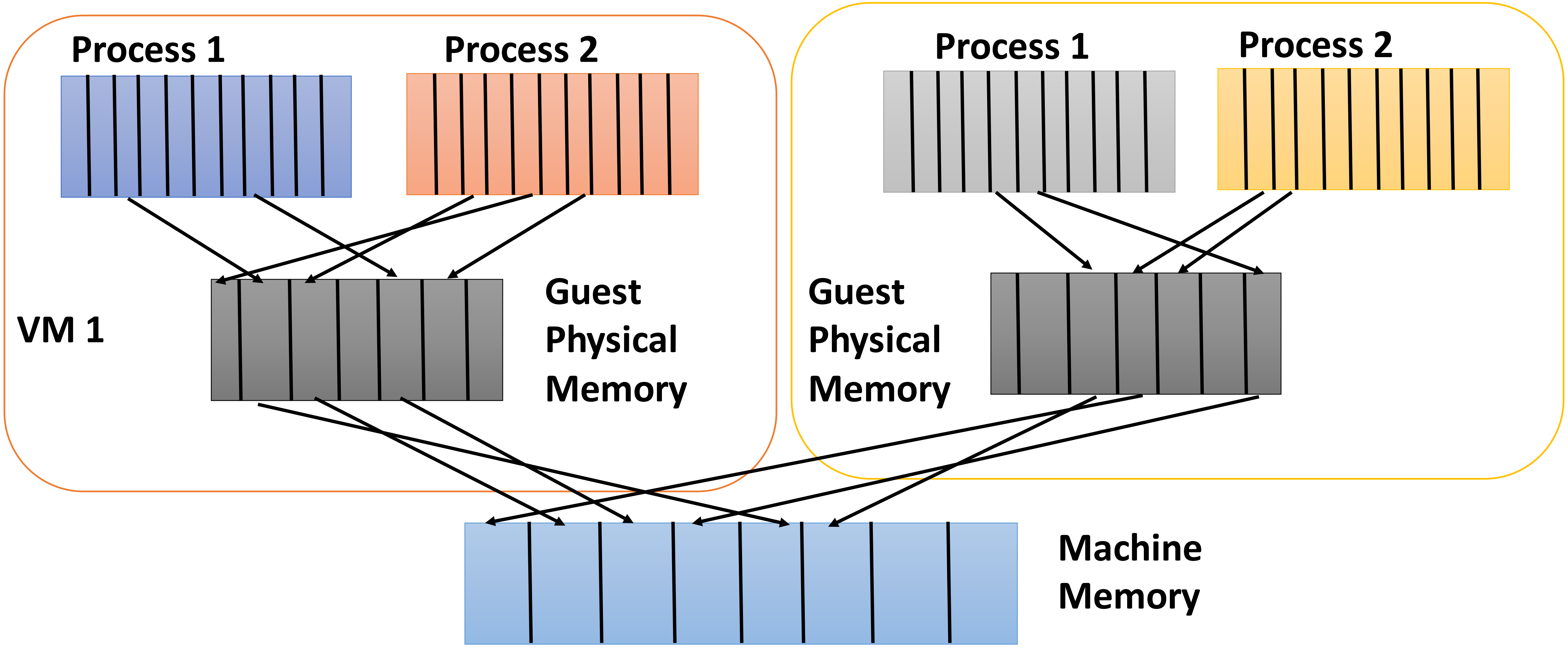
- Software enters VMX operation by executing a VMXON instruction.
- Using VM entries, a VMM can then turn guests into VMs (one at a time).
    - The VMM effects a VM entry using instructions VMLAUNCH and VMRESUME
    - It regains control using VM exits.
- VM exits transfer control to an entry point specified by the VMM.
    - The VMM can act appropriately to the cause of the VM exit and can then return to the VM using a VM entry.
- Eventually, the VMM may decide to shut itself down and leave VMX operation.
    - It does so by executing the VMXOFF instruction.

Memory Virtualization

- Access to the MMU needs to be virtualized
  - Otherwise, the guest OS may directly access the physical memory

- Physical memory is divided among multiple VMs
  - Two levels of translation required
  - **Guest OS:** Guest virtual address -> guest physical address
  - **VMM:** Guest physical address -> Machine address
  - Extended page tables (EPT) in VT-x or Nested page table in AMD-v facilitates this two-level translation

Two Level Paging



Shadow Page Table

- Software-based mechanism to handle two-level paging at the VMM.
- Shadow page tables provide a map between the guest OS's virtual memory pages and the underlying physical machine pages.
  - Deny the guest OS any access to the actual page table entries by trapping access attempts and emulating them in software
  - Guest page tables are read-only (maintained for each VM separately). When guest attempts to update, VMM intercepts (traps) and emulate the effects on the corresponding shadow page table
  - Avoids the mapping from guest virtual address to guest physical address

Second Level Address Translation (SLAT)

- Also known as nested paging, a mechanism for hardware-assisted virtualization
  - Avoid the overhead associated with software-managed shadow page tables
  - AMD supports SLAT through Rapid Virtualization Indexing (RVI); Intel uses Extended Page Tables (EPT).
- Treat each "guest physical address" as the "host virtual address"
  - The host page table can be viewed conceptually as nested within the guest page table
  - A hardware page table walker can treat the additional translation layer almost like adding levels to the page table.

Extended Page Tables (EPT)

- Intel second-generation x86 virtualization technology for MMU
  - Supports in Intel's Core i3, Core i5, Core i7 and Core i9 CPUs
- EPT translates addresses as follows:
  - One page table is maintained by the guest OS, which is used to generate the guest’s physical address
  - The other page table is maintained by VMM, which maps the guest’s physical address to the host’s physical address.
- For each memory access operation, EPT MMU directly gets the guest’s physical address from the guest page table and then use it to get the host’s physical address from the VMM mapping table.