CPU Virtualization

Yu Ke <ke.yu@intel.com>

Jiang Yunhong <yunhong.jiang@intel.com>

Legal Disclaimer

Agenda

- CPU Virtualization Overview
- CPU Virtualization Hardware Support (VT-x)
- CPU Virtualization Software Implementation
- Summary

Summary of Last Session

- What is Virtualization
- Virtualization Challenge
- Virtualization Technologies

Agenda

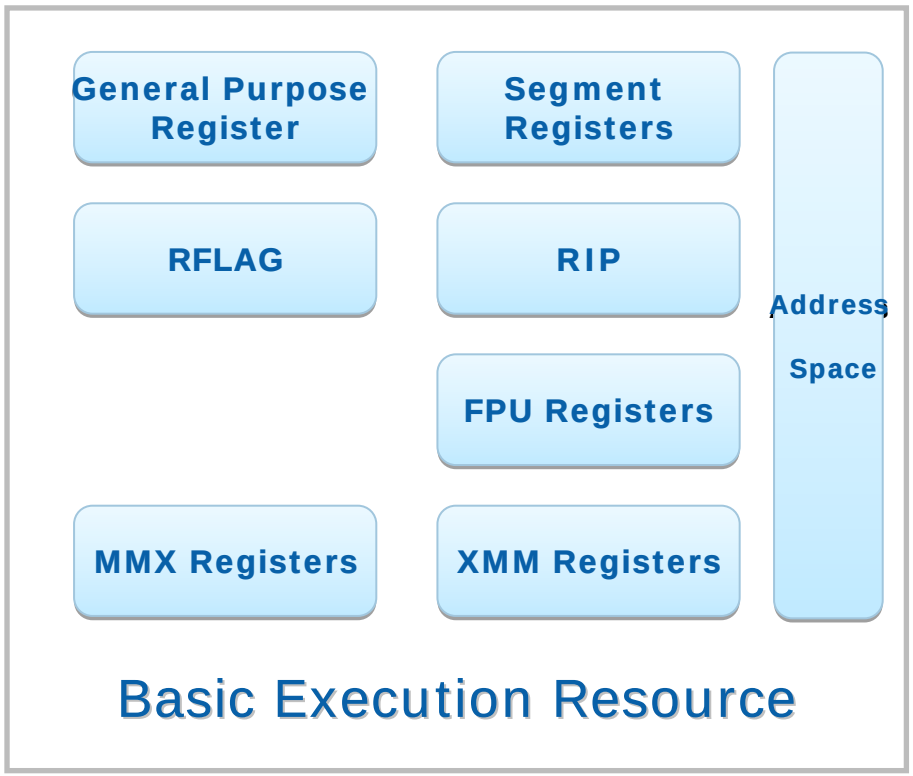
- **CPU Virtualization Overview**
  - CPU Virtualization Goal
  - What is CPU: Software's View
  - CPU Virtualization Approach
- **CPU Virtualization Hardware Support (VT-x)**
- **CPU Virtualization Software Implementation**
- **Summary**

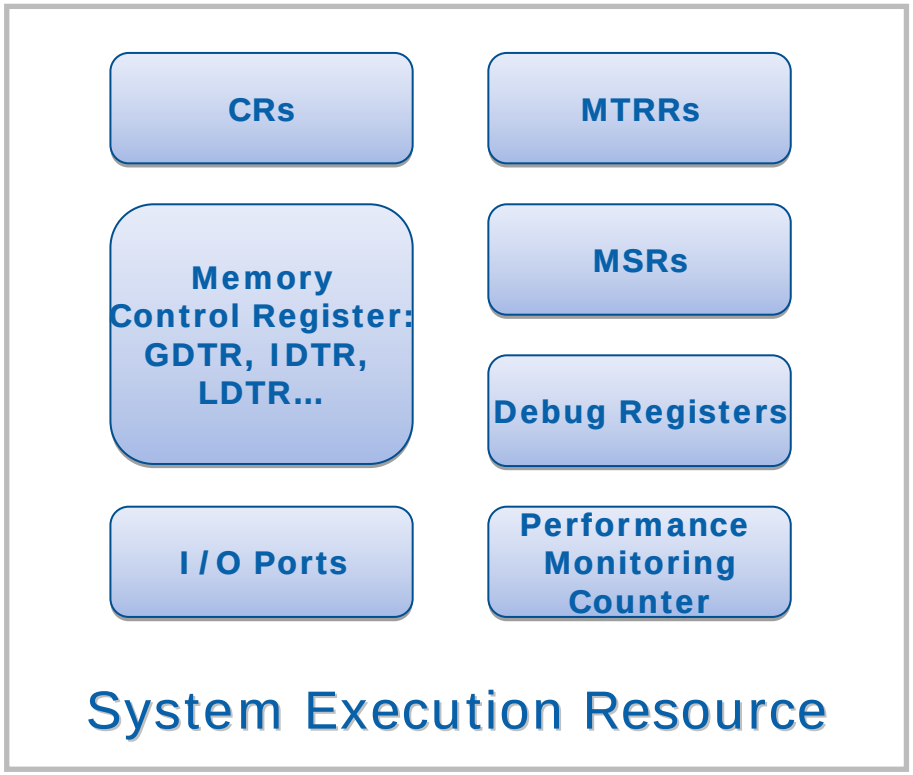
CPU Virtualization Goal

- **Goal: provide guest software the same ISA (Instruction Set Architecture) as physical CPU**

- **ISA defines**
  - the state visible to the software
    - Registers and memory
  - the instruction that operate on the state

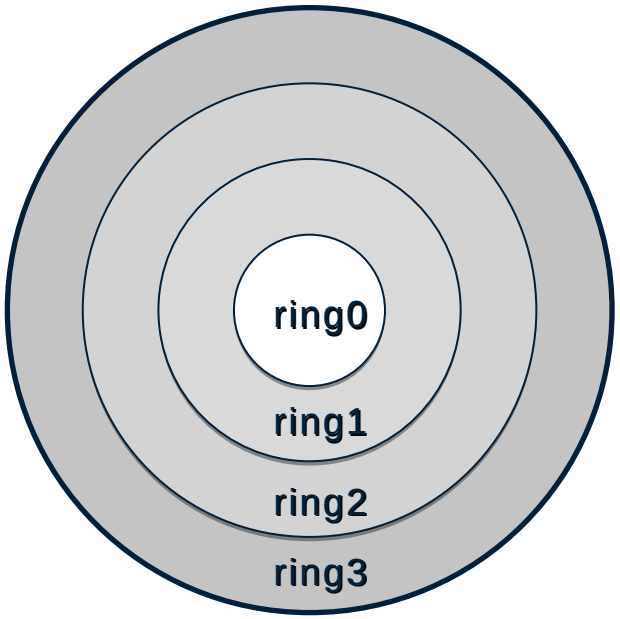
CPU Software's View: Resource



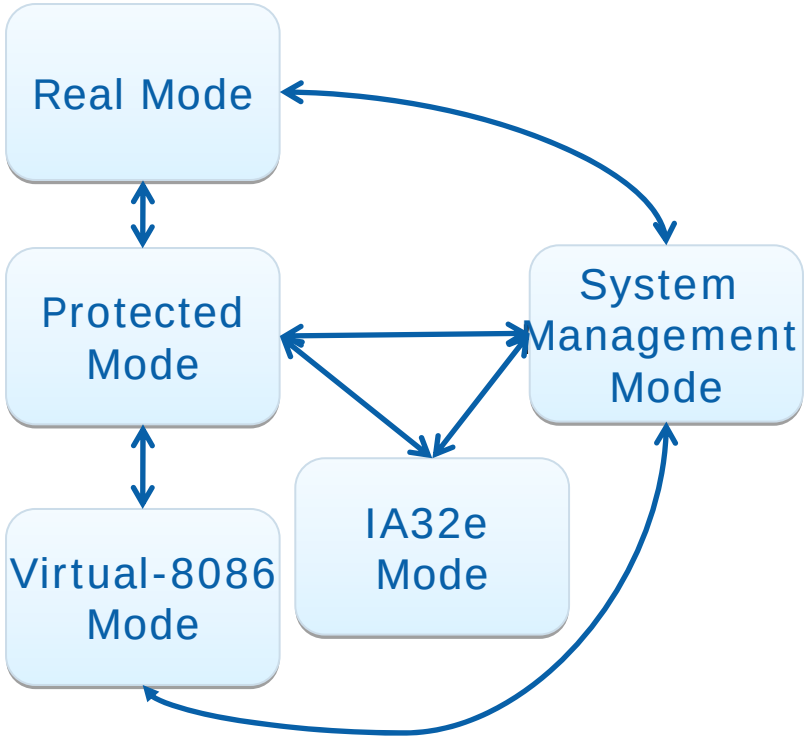


View 1: Set of Execution Resource

CPU Software's View: Mode



Ring

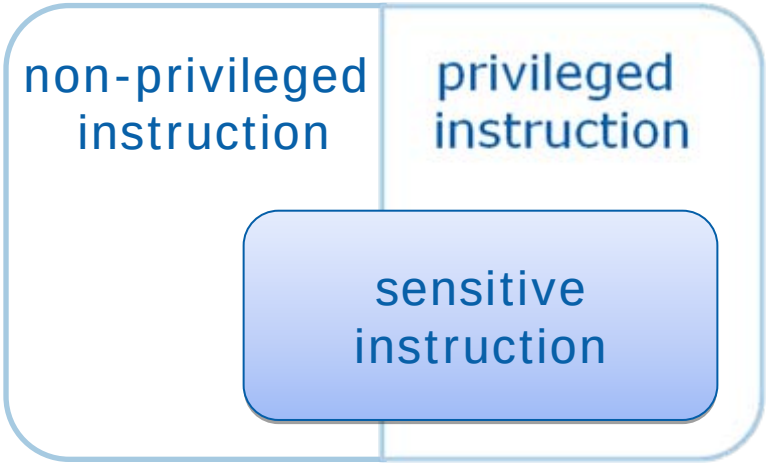


Mode of Operation

View 2: Predefined Mode - Ring and Mode of Operation

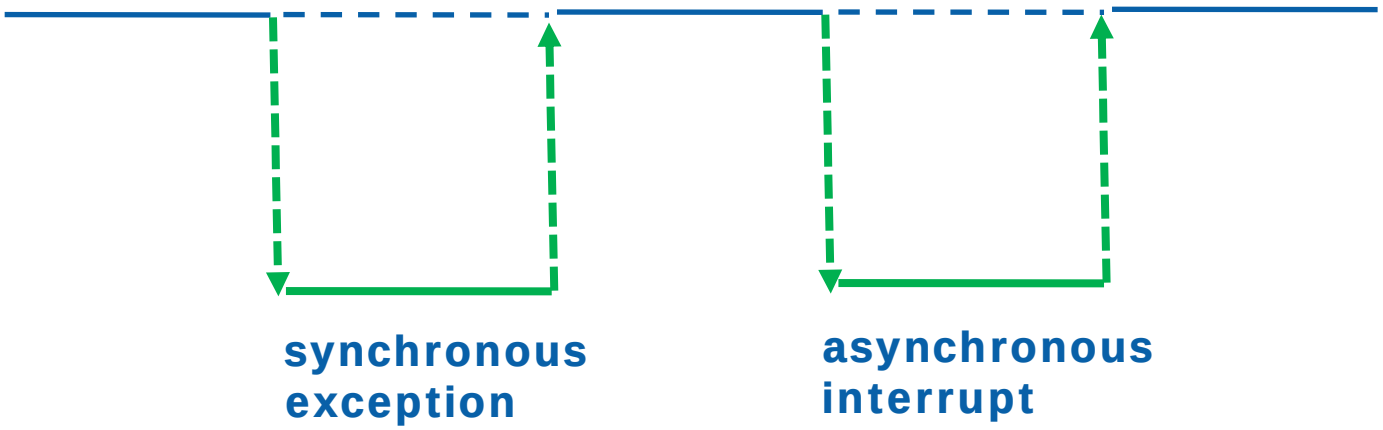
CPU Software's View: Instruction

- **Privileged Instruction:**
  - Those that trap if the processor is in user mode and do not trap if it is in system mode. Non-privileged Instruction
- **Sensitive Instruction:**
  - Control sensitive instructions
    - Those that attempt to change the configuration of resources in the system.
  - Behavior sensitive instructions
    - Those whose behavior or result depends on the configuration of resources (the content of the relocation register or the processor's mode).



View 3: Execute Instruction Set with Predefined Semantic

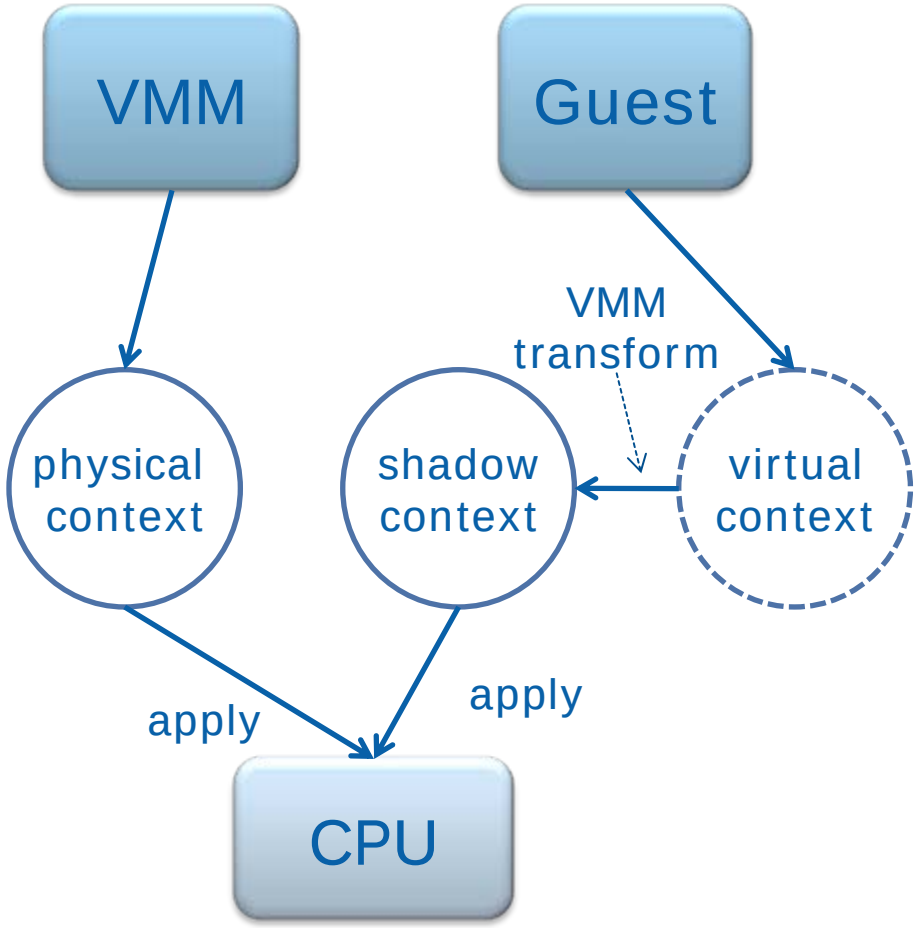
CPU Software's View: interruption



View 4: Response to Interruption

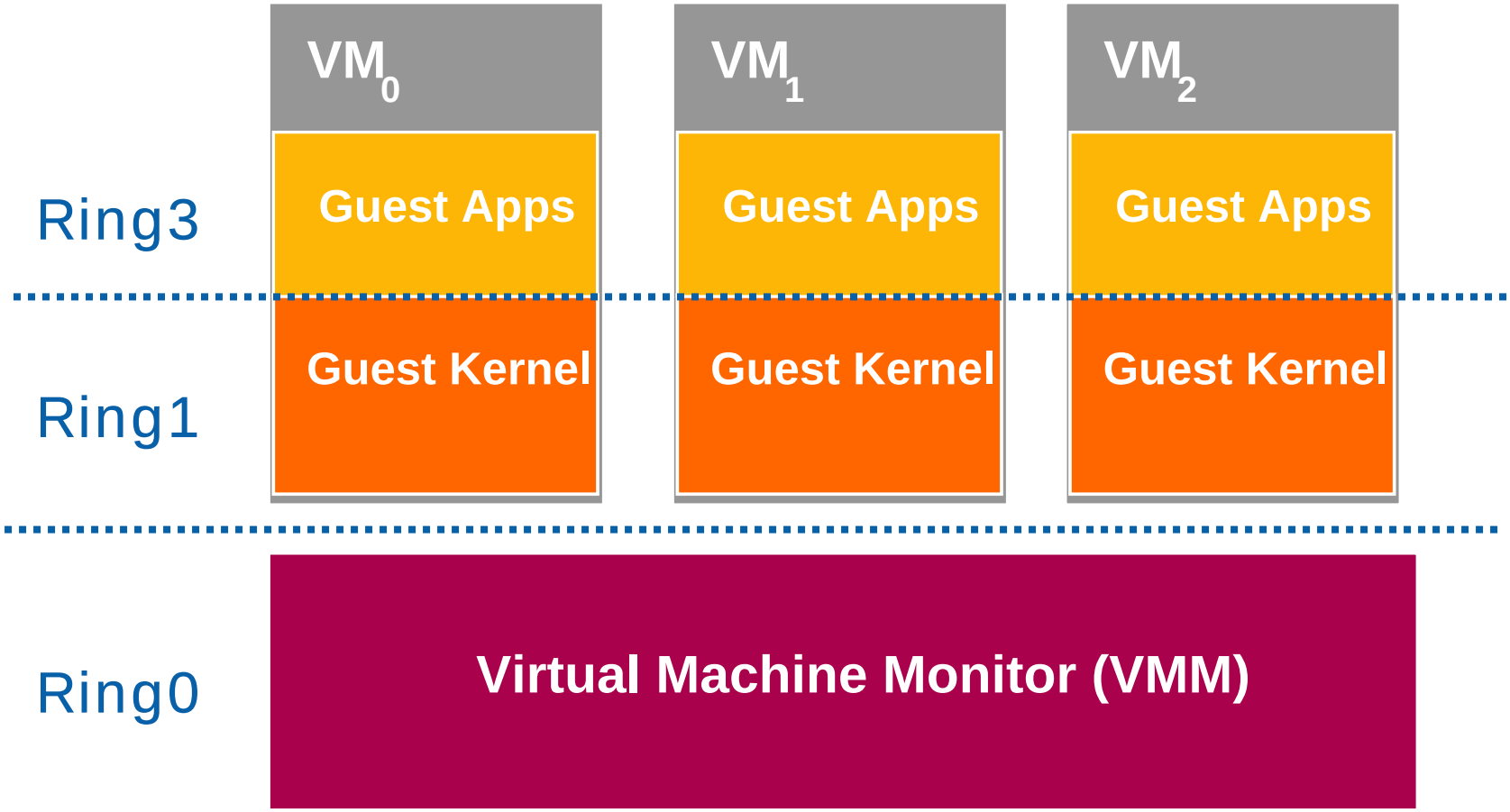
Resource Virtualization

- **Basic Execution Resource: context switch approach**
  - Context save/restore during virtual CPU switch
- **System Execution Resource: "Physical / Virtual / Shadow" approach**
  - Host Mode
    - Physical context
  - Guest mode
    - Virtual context
    - Shadow context



"Physical / Virtual / Shadow" Approach

Ring Virtualization



Traditional Ring De-privileging: virtualization hole issue!
Hardware-assisted approach can cleanly solve this issue

Instruction & Interrupt Virtualization

- **Run instruction without intervening**
  - Apply to basic instruction
- **Trap and Emulation**
  - Apply to sensitive instructions
  - Apply to interrupt

Agenda

- CPU Virtualization Overview
- **CPU Virtualization Hardware Support (VT-x)**
- CPU Virtualization Software Implementation
- Summary

Why need H/W support

- **S/W approach has virtualization holes**
  - Ring Aliasing
  - Non-trapping instructions
  - Excessive Faulting
  - Interrupt Virtualization Issues
  - CPU state context switching
  - Address Space Compression

- **S/W approach lead to complex workaround under traditional IA architecture**
  - Guest OS level source code modification (para-virtualization)
  - Guest OS binary level patching

H/W support can eliminate virtualization holes and simplify VMM

VT-x: Key Features

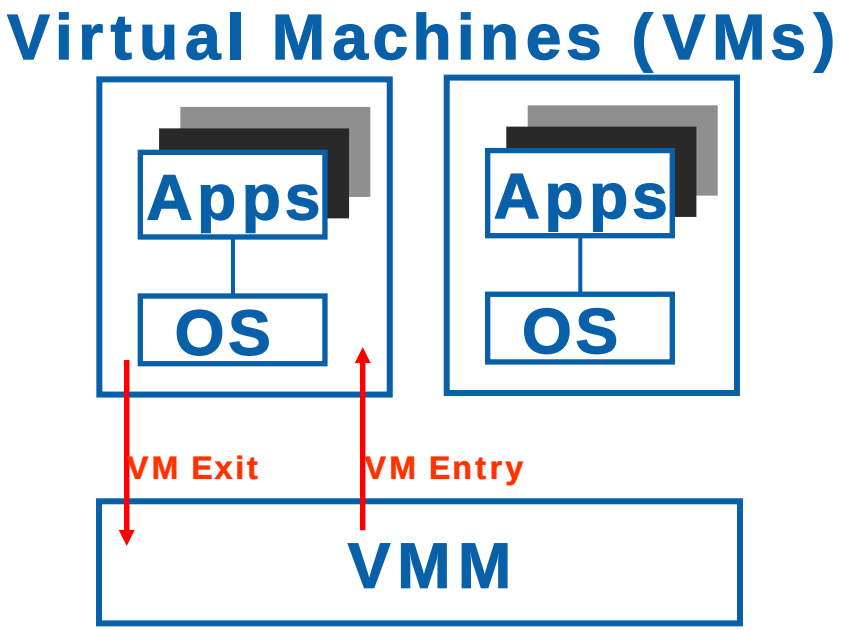
- **New mode of operation for guest**
  - Allows VMM control of guest operation
  - Need not use segmentation to control guest
  - Guest can run at its intended ring
- **New structure controls CPU operation**
  - VMCS: virtual-machine control structure
  - Resides in physical-address space
  - Need not be in guest's linear-address space

VMX Operation

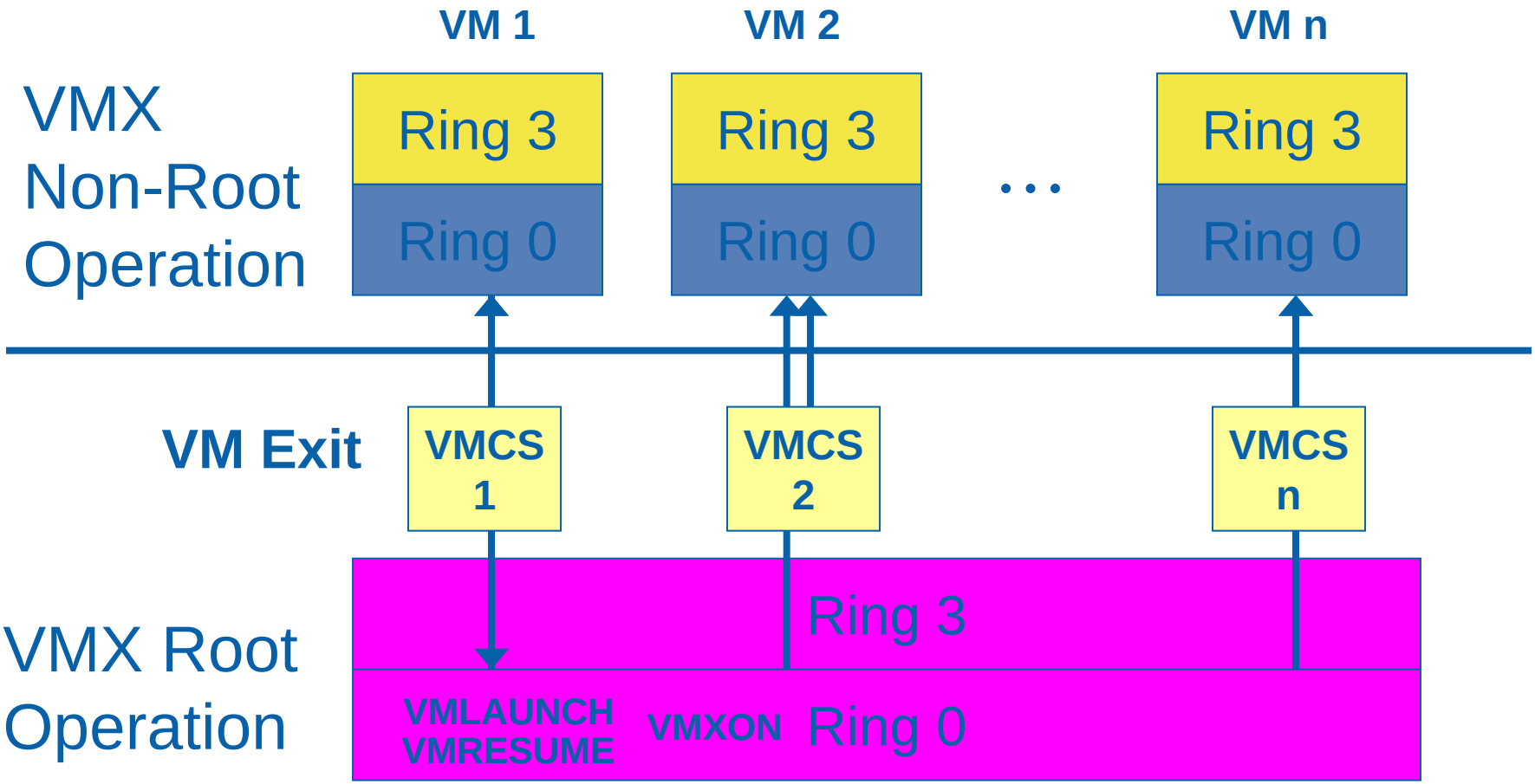
- **New mode of operation for VMM and VMs**
- **Entered with new VMXON instruction**
- **VMX root operation:**
  - Fully privileged, intended for VM monitor
  - Entered on exits from guest software
- **VMX non-root operation:**
  - Not fully privileged, intended for guest
  - Explicitly entered by VMM (new instructions)

VMX Transitions: Overview

- **VM entry**
  - Transition from VMM to guest
  - Enters VMX non-root operation
  - VMLAUNCH used for initial entry
  - VMRESUME used subsequently
- **VM exit**
  - Guest-to-VMM transition
  - Enters VMX root operation
  - Caused by external events, exceptions, some instructions



VMX Operation

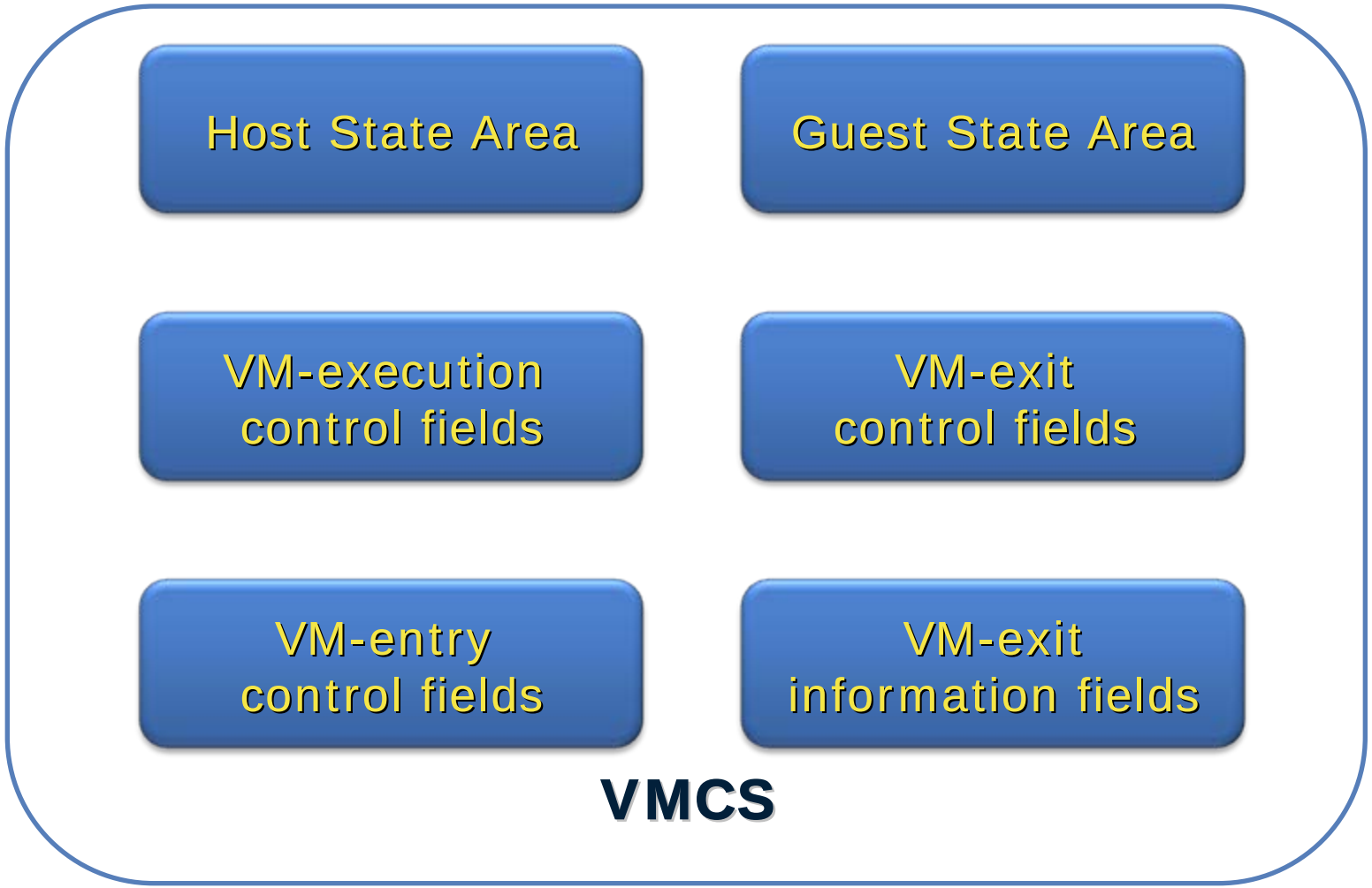


Virtual Machine Control Structure (VMCS)

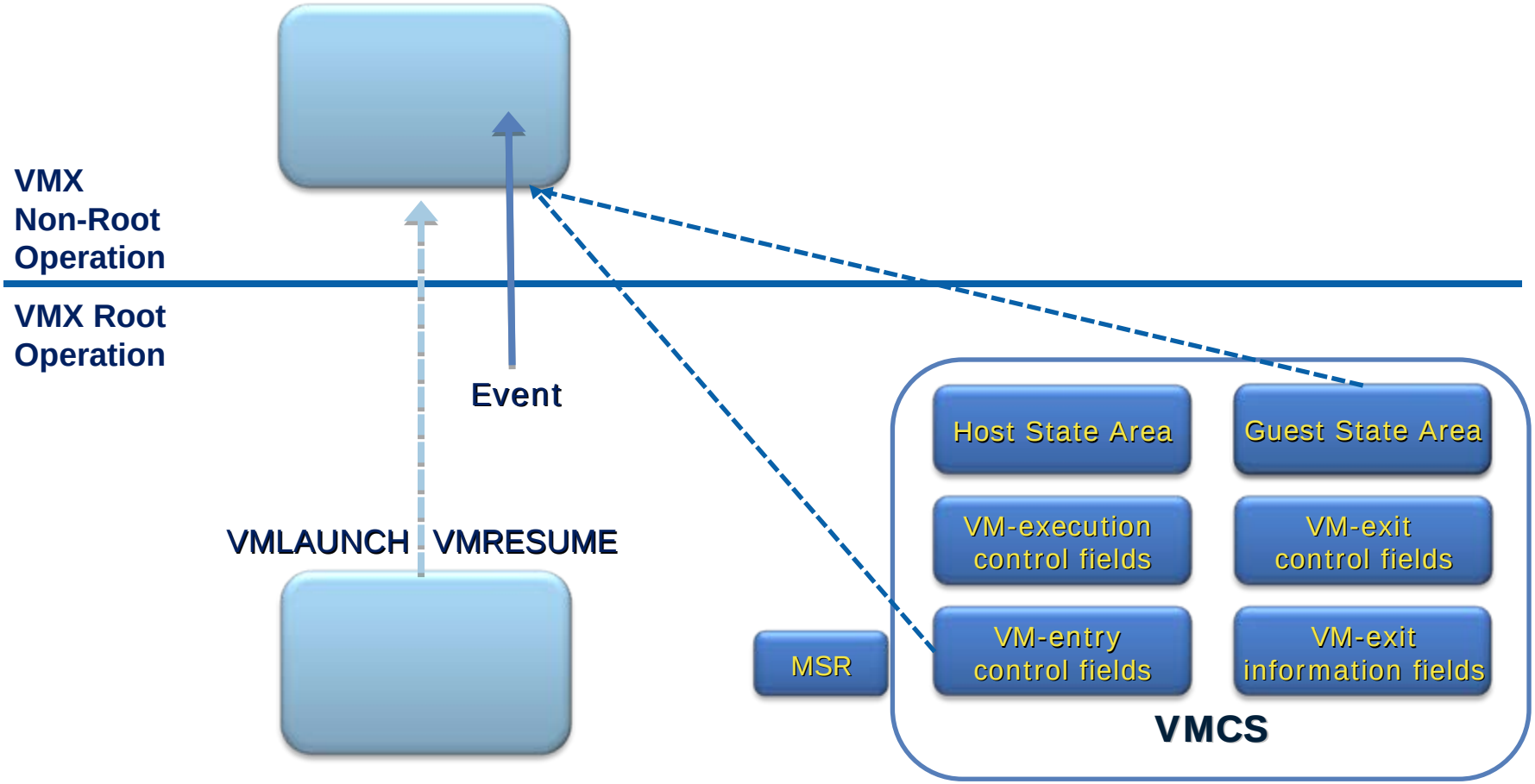
- **Control structures representing virtual processor**
  - Each virtual CPU should have its own VMCS
  - Only one VMCS active at a time on a CPU

- **A memory block used by VMM and VT-x hardware**
  - VMM: allocate and initialize the memory block, use “VMPTRLD” to make it active, use “VMREAD/VMWRITE” to access @ runtime
  - VT-x hardware: use the VMCS to control virtual processor behavior, update VMCS according to guest behavior.

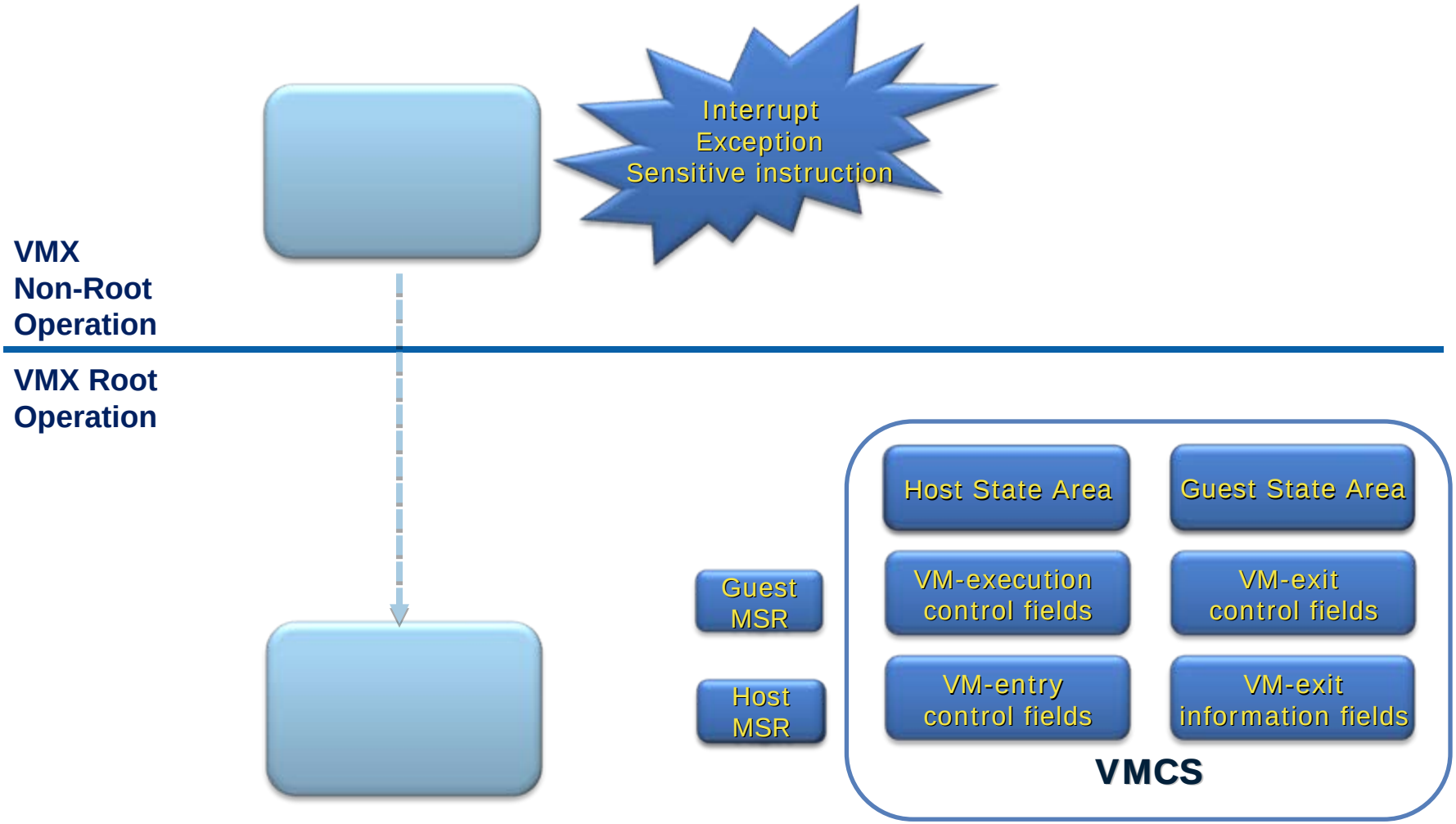
VMCS Content



VM Entry


VMX
Non-Root
Operation
VMX Root
Operation
Event
VMLAUNCH
VMRESUME
Host State Area
Guest State Area
VM-execution
control fields
VM-exit
control fields
MSR
VM-entry
control fields
VM-exit
information fields
VMCS

VM Exit


Interrupt
Exception
Sensitive instruction
VMX
Non-Root
Operation
VMX Root
Operation
Guest
MSR
Host
MSR
Host State Area
Guest State Area
VM-execution
control fields
VM-exit
control fields
VM-entry
control fields
VM-exit
information fields
VMCS

Agenda

- CPU Virtualization Overview
- CPU Virtualization Hardware Support (VT-x)
- **CPU Virtualization Software Implementation**
- Summary

VCPU Concept

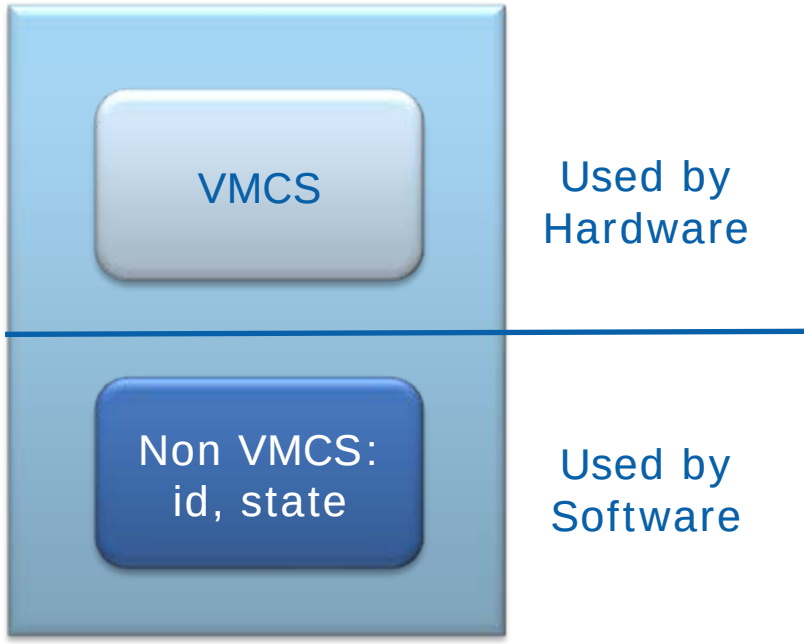
- **VCPU is the central of CPU virtualization software implementation**

- **VCPU Data**
  - A software entity
    - representing virtual processor
    - Containing all the context info for CPU virtualization (similar as "process descriptor" in OS)
  - Scheduler entity

- **VCPU Operations**
  - VCPU creation
  - VCPU running
  - VCPU migration
  - VCPU destroy

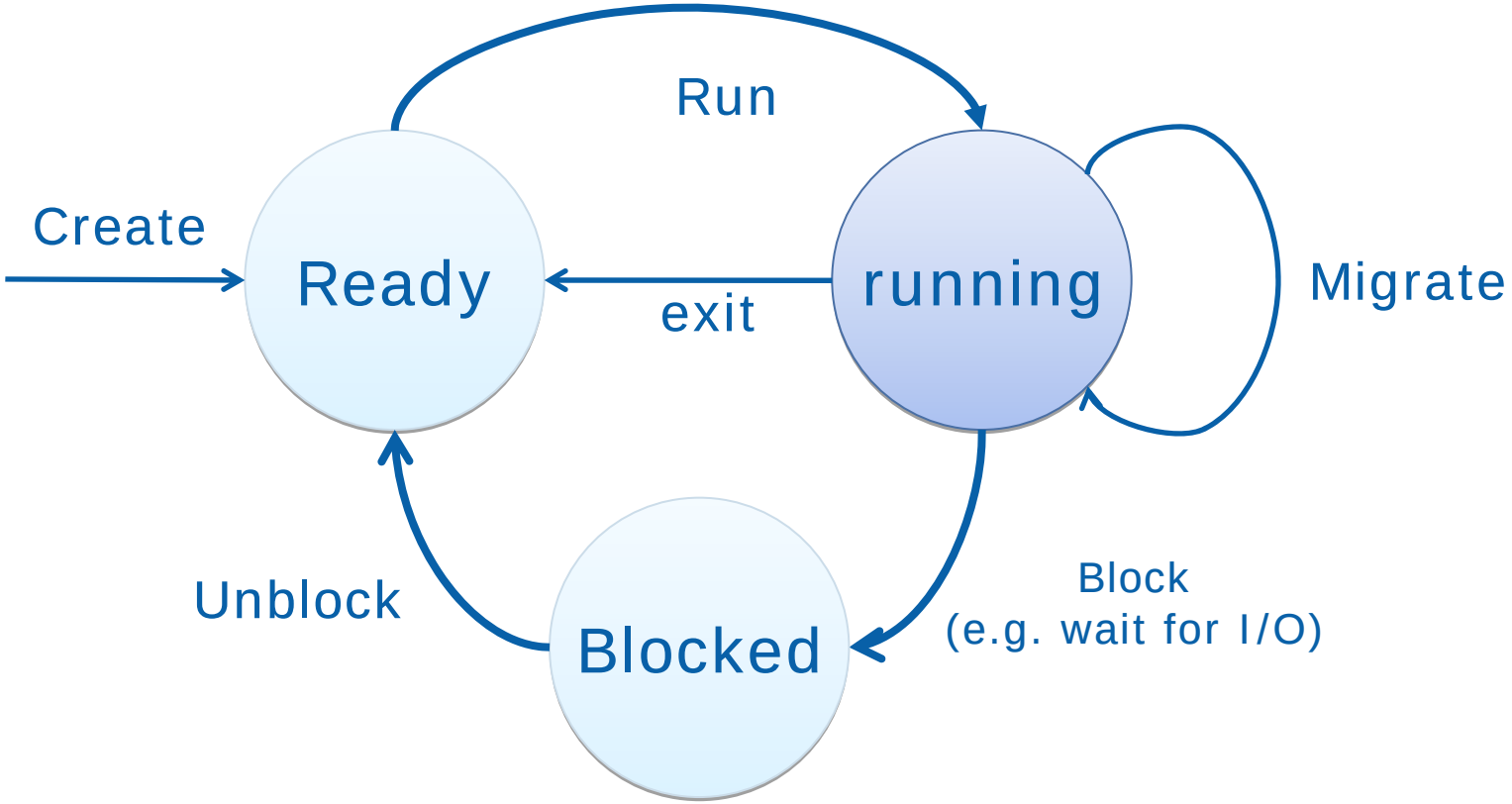
VCPU Structure

- **VCPU is consisted of**
  - VCPU id
  - Virtual register
    - VMCS
    - Registers not in VMCS
  - VCPU booking info for scheduler: e.g. running/sleeping state
  - Misc.



VCPU Structure

VCPU Operation and Life-cycle



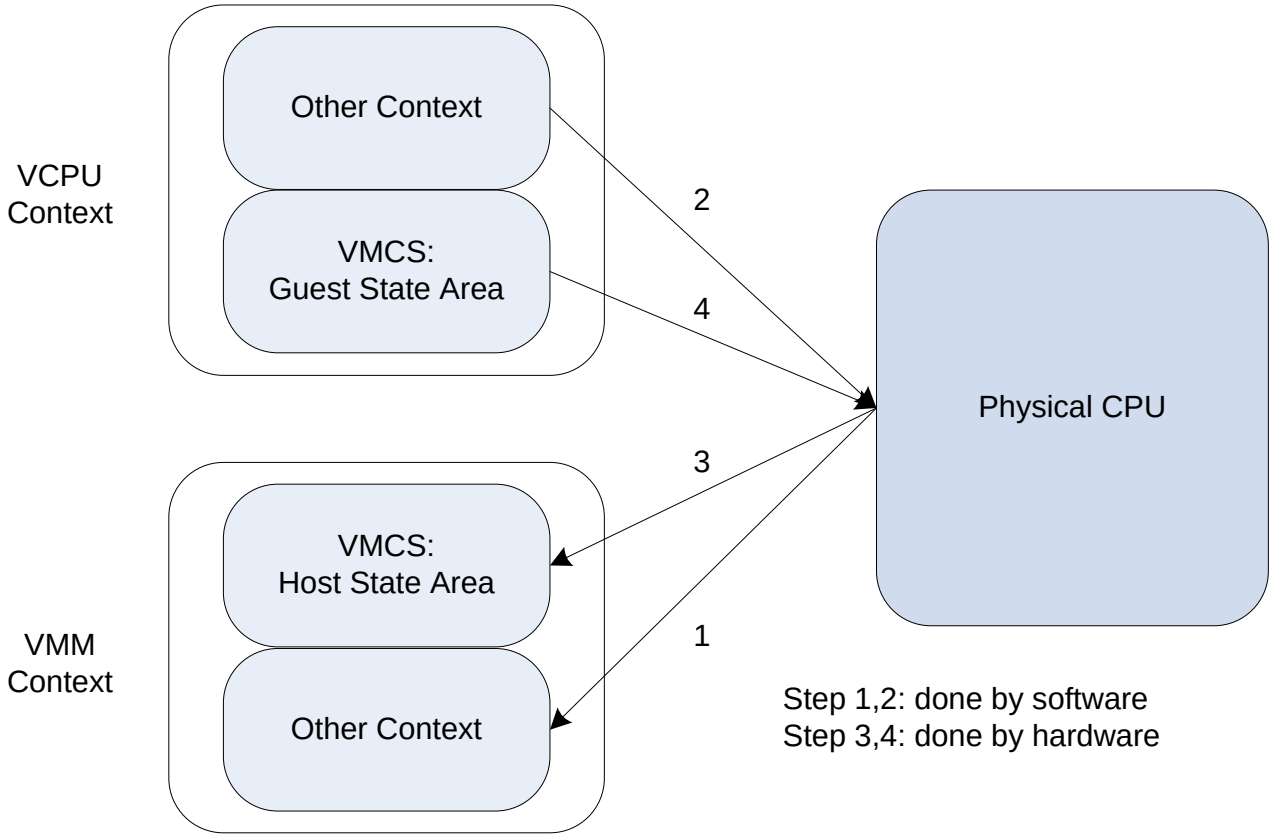
VCPU Creation

- **Task: create and initialized the VCPU structure**

- **Allocate and initialize VMCS**
  - VMCS is 4KB aligned memory block
  - Guest State Area: set to the similar state as physical CPU POST state. e.g. RIP can be the entry point of guest BIOS
  - Host State Area: set to VMM host CPU state.
  - Other area: set according to VMM policy

- **Allocate and initialize non-VMCS content**
  - VCPU id
  - VCPU booking info
  - Misc.

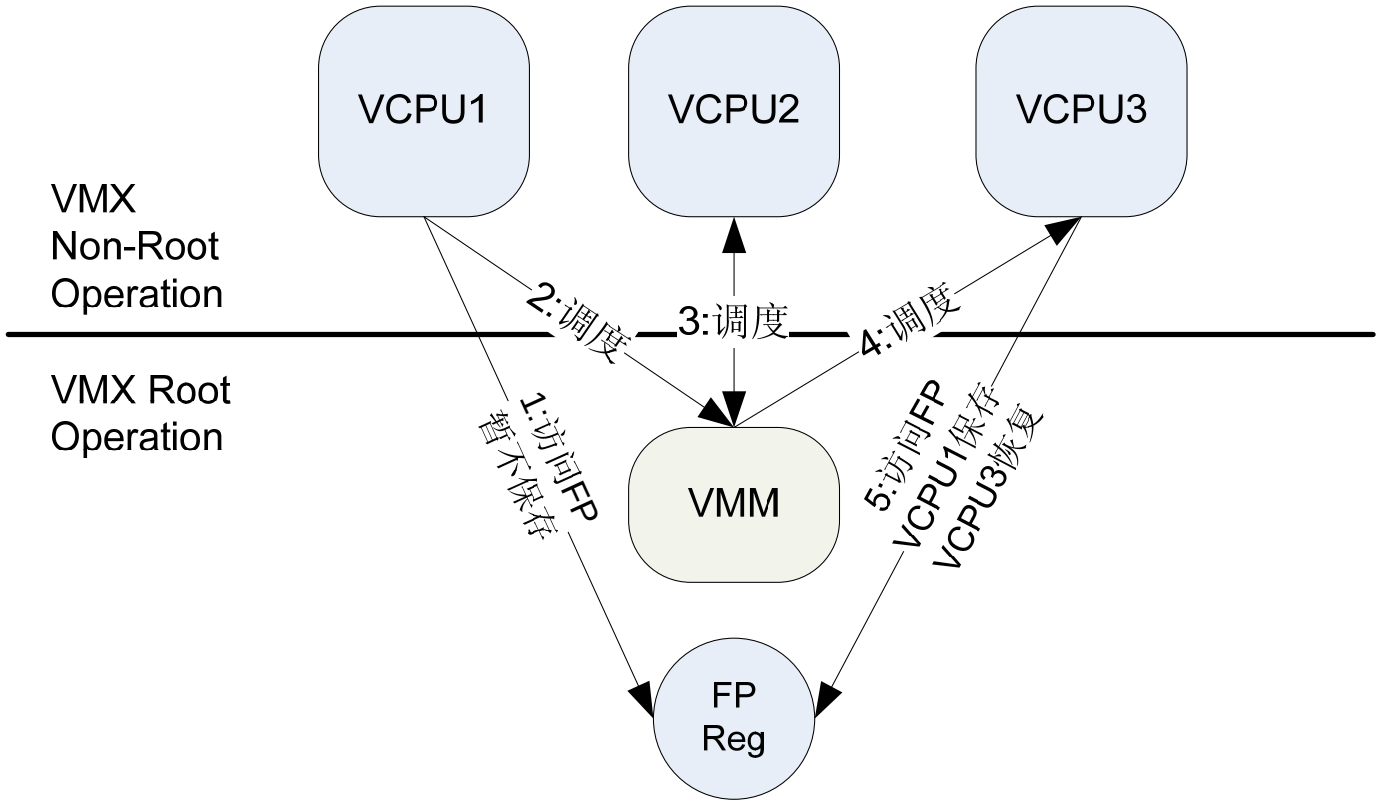
VCPU Run – Context Switch for VM entry



- **VMM**
  - save VMM context
  - restore VCPU context
  - vmlaunch / vmresume
- **Hardware (VM Entry)**
  - Restore VMCS guest state area

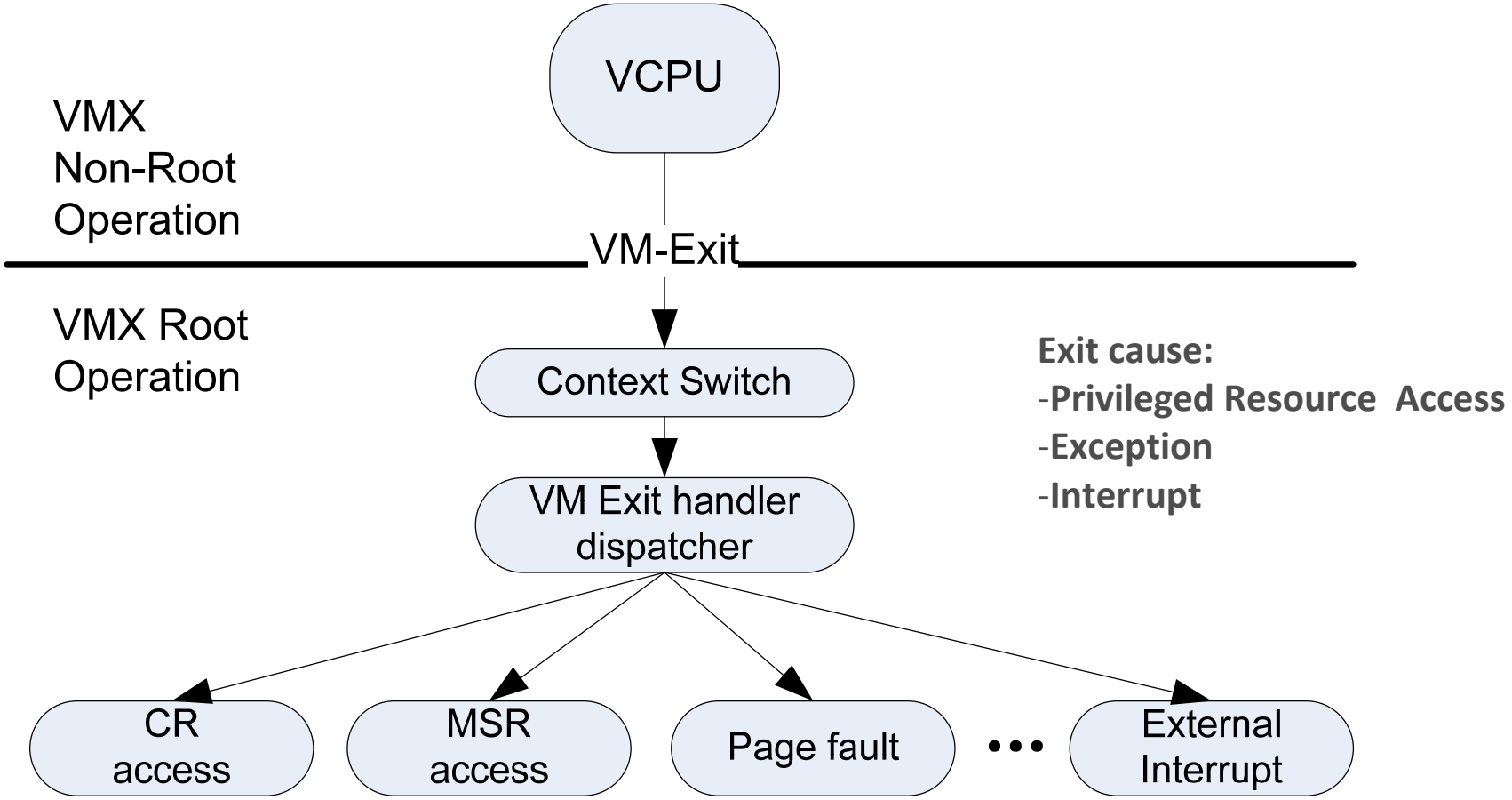
Context switch during VM Entry

VCPU Run – Lazy Save/Restore



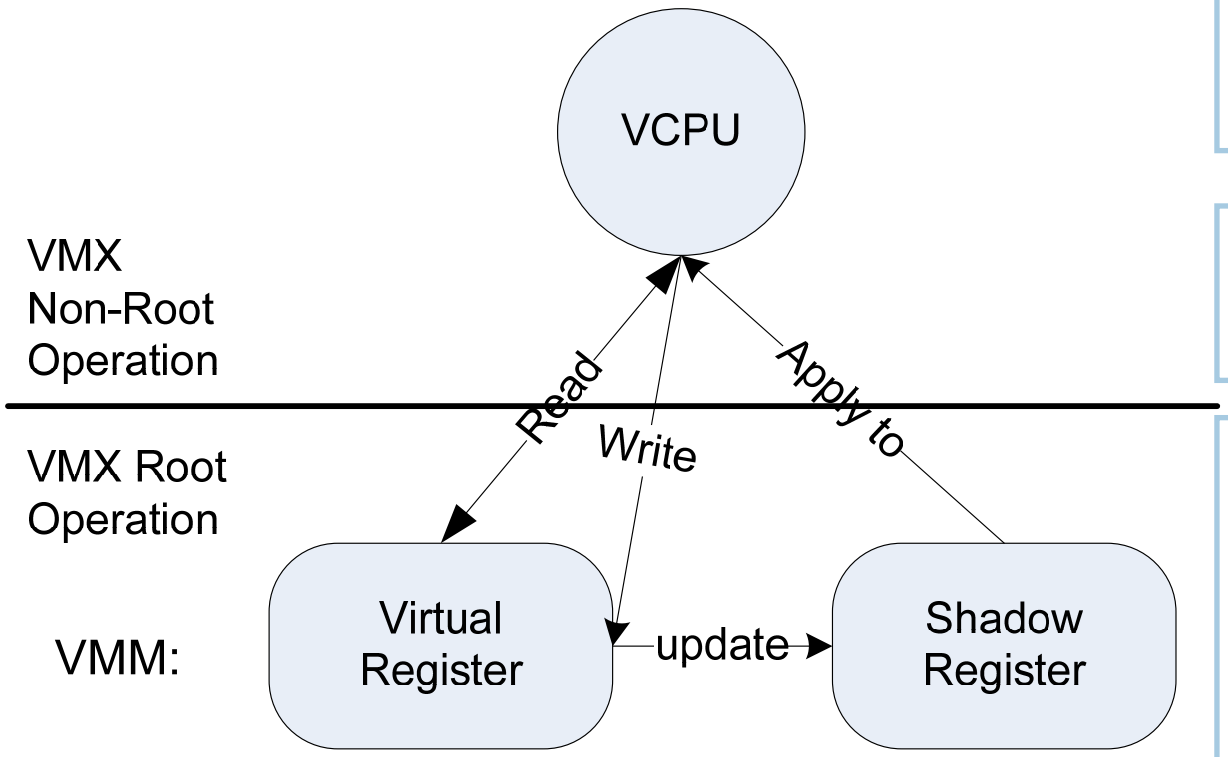
Context Switch Optimization: Lazy Save/Restore

VCPU Exit



VCPU exit - core of CPU virtualization

VCPU Exit – Privileged Resource Access



Privileged Resource Virtualization

e.g. CR0 access
Before Instruction:
virtual CR0=0x80000001
(page enabled, protected mode)

Guest Instruction:

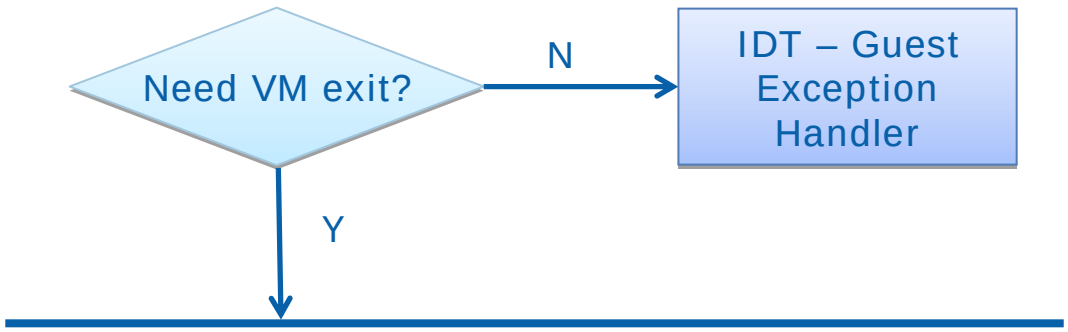
```
MOV EAX, 0X1
MOV CR0, EAX # disable page
```

VMM handler:
- Set virtual CR0 to 0x1
- Shadow CR0 still be 0x80000001
- Notify the change to memory virtualization component

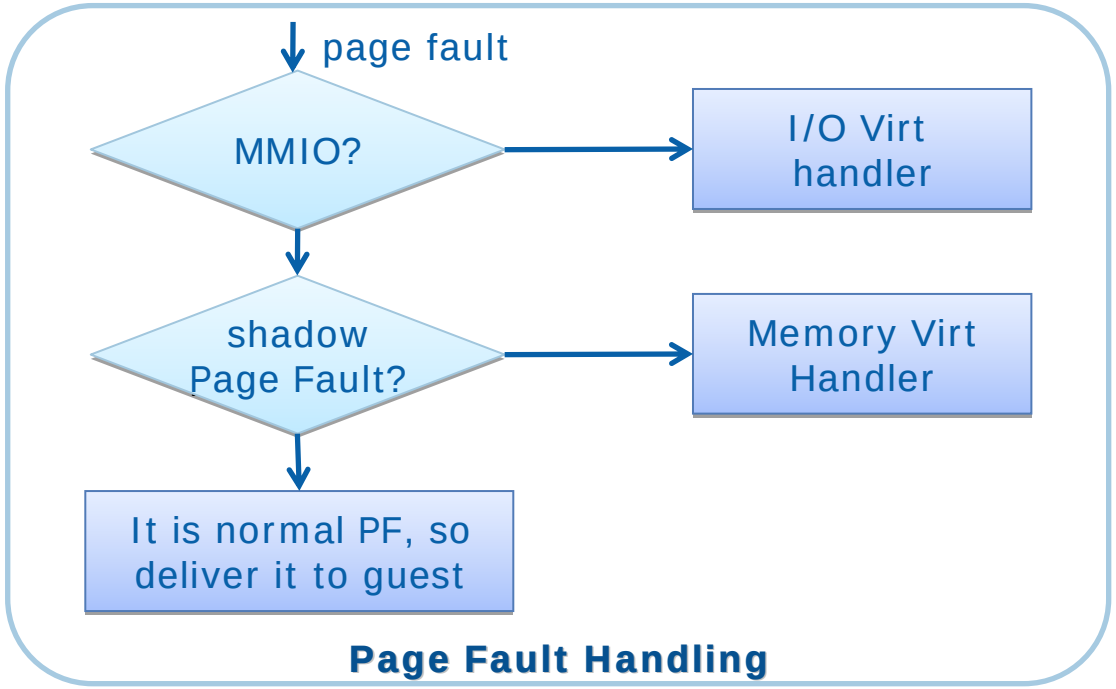
Virtual CR0: VMCS ->VM Execution field-> CR0 read shadow
Shadow CR0: VMCS->Guest State->CR0

VCPU Exit – Exception

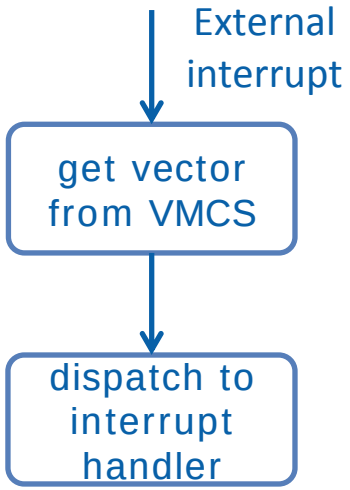
- **Some exception is not necessary to exit, e.g. "Div by 0", VMM can set VMCS exception bit map to deliver it to guest directly**



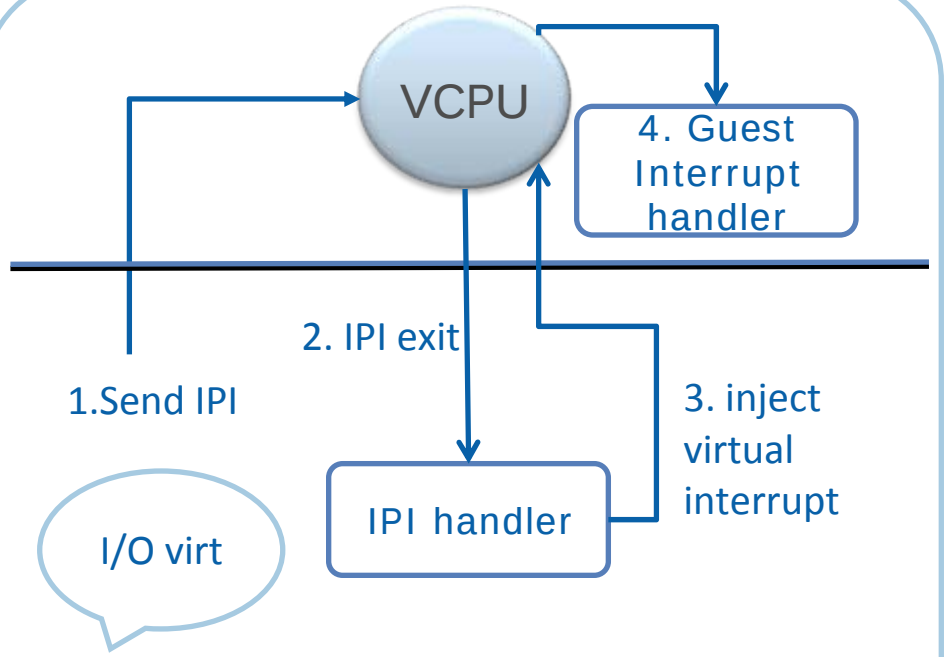
- **For the rest exceptions, VMM will handle case by case. e.g. for page fault:**



VCPU Exit - Interrupt



case1:
Exit due to physical device external interrupt



case2:
exit due to virtual device interrupt

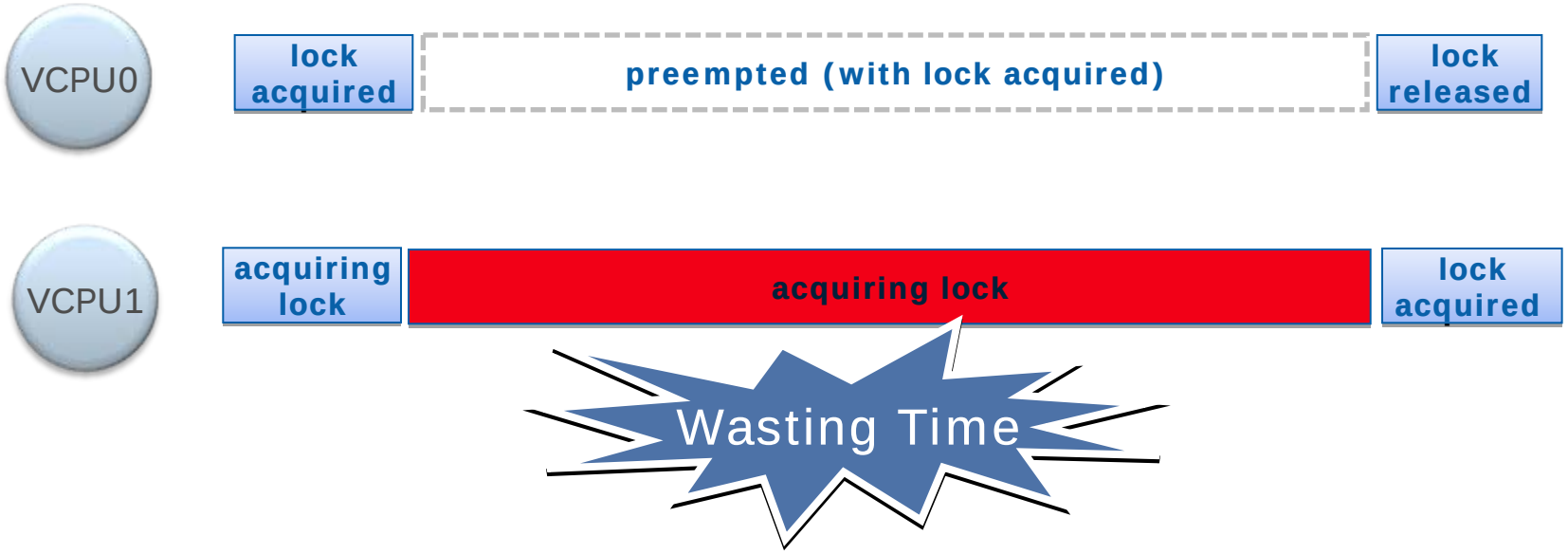
VCPU Optimization

- **VT-x provide hardware optimization to reduce the VM exit and context switch:**

- **Bitmap to control VM-exit**
  - I/O port bitmap
  - Exception bitmap
- **Return shadow value to avoid VM-exit**
  - CR0/CR2/CR4 reading: return shadow value
  - TSC reading: return TSC+offset

- **SYSENTER/SYSEXIT**
  - SYSENTER/SYSEXIT will directly go to system call routine, and no VM exit

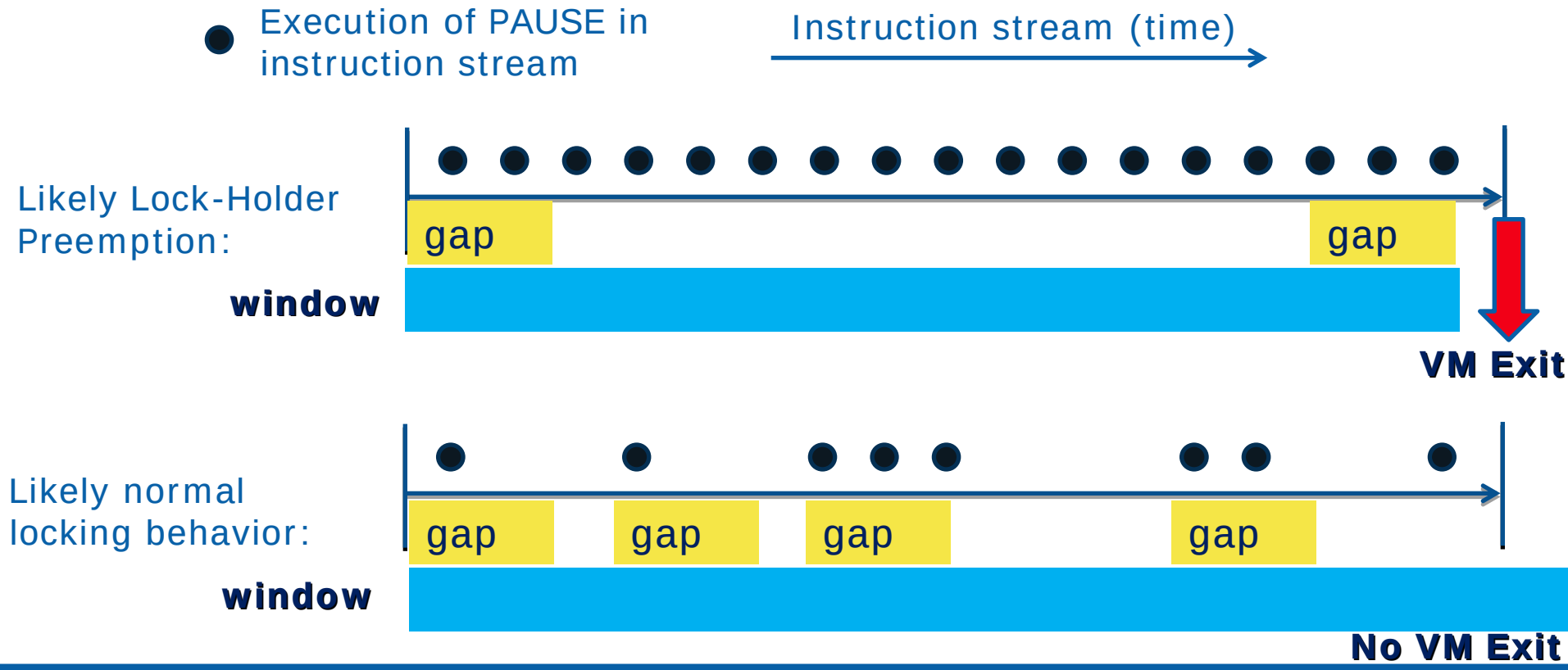
Lock-Holder Preemption



- **Solution for LHP issue**
  - Detect that VCPU is busy acquiring lock for long time and schedule out the VCPU

VT-x Pause Loop Exit

- **VT-x provide "Pause Loop Exit" to detect busy acquiring lock CPU**
  - "PLE_Gap & PLE_Window" field in VMCS



Summary

- CPU virtualization Goal - provide guest the same ISA as physical CPU
- VT-x provide hardware extension for IA32 CPU virtualization and simplify the VMM implementation
- VCPU is the central of CPU virtualization software implementation

Backup

Case Study:
Mode of Operation Virtualization

Why mode of operation virtualization

- **Mode of Operation**
  - Real mode
  - Protected mode
  - VM86 mode
  - IA32e mode
- **VT-x require CR0.PE=CR0.PG=1 (i.e. protected mode with paging) in non-root operation, so if guest require CR0.PG=0 or CR0.PE=0, it need virtualization.**
- **Two guest mode need virtualization:**
  - CR0.PG=0, CR0.PE=1: Guest Protected Mode with Paging disabled
  - CR0.PG=0, CR0.PE=0: Guest Real Mode

Guest Protected with Paging disabled

- **Gap:**
  - Virtual CPU should see flat memory
  - Physical CPU is in protected mode with paging enabled

- **Approach to address the gap:**
  - VMM use identity-mapping page table to provide flat memory

Guest Real Mode

- **Gap:**
  - VCPU is in real mode, has different opcode length, address space, and semantic from protected
- **Approach to address the gap**
  - VMM use protected VM86 mode to execute the real mode instruction
  - for those instructions that can not handle in VM86 mode, VMM will trap and emulate the instruction

Hardware Support

- **In recent VT-x, a new feature called “Unrestricted guest” is introduced.**
  - Feature can be detected by IA32_VMX_MISC MSR bit 5
- **With VMCS “Unrestricted guest” set, guest software can run in unpaged protected mode or in real-address mode**
- **software involvement is not needed in mode of operation virtualization in “Unrestricted guest” case**

VCPU Migration (Backup)

- Scheduler can migrate VCPU from one physical CPU to another physical CPU (e.g. for load balance purpose)

1. De-schedule VCPU from source CPU
2. Migrate VMCS
   - VMCLEAR in source CPU
   - VMPTRLD in target CPU
3. Schedule VCPU in target CPU by VMLAUNCH (not VMRESUME)

PLE in Software

- **Enable the VMCS "Pause-loop exiting" bit**
- **Configure VMCS "PLE_Gap" & "PLE_Window"**
  - The reasonable value is estimated by performance measure
- **How to wake up VCPU0**
  - Key is how to find lock holder
  - Option1: para-virtualization
  - Option2: randomly select one sleep VCPU