System Virtualization

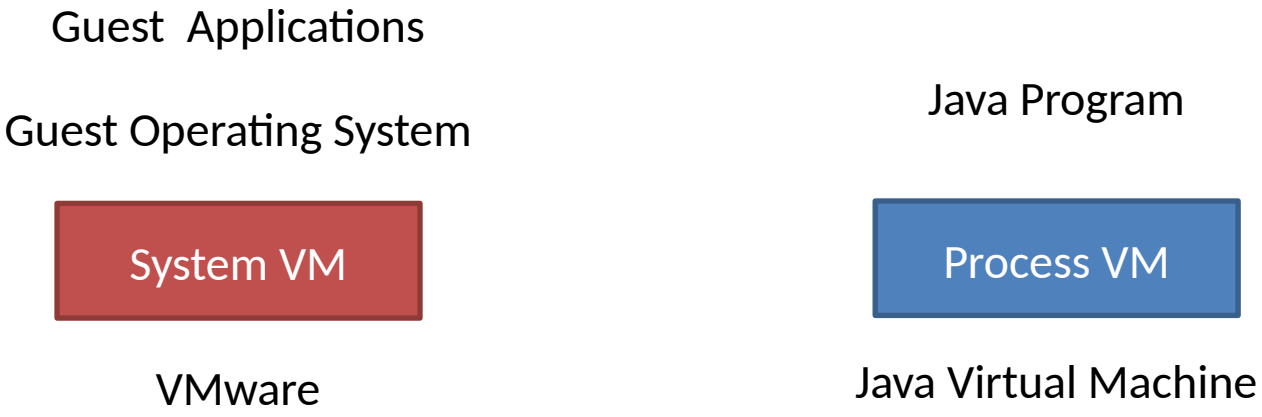
Definition of Virtualization

In computing, virtualization means to create a virtual version of a device or resource, such as a server, stor age device, network or even an operating system wh ere **the framework divides the resource into one or more execution environments**.

Virtual Machine (1/2)

- A virtual machine (VM) is a software implementation of a machine that executes programs like a physical machine. Virtual machines are separated into two major classifications:
  - A system virtual machine
    - Which provides a complete system platform which supports the execution of a complete operating system (OS)
  - A process virtual machine
    - Which is designed to run a single program, which means that it supports a single process.

Virtual Machine (2/2)



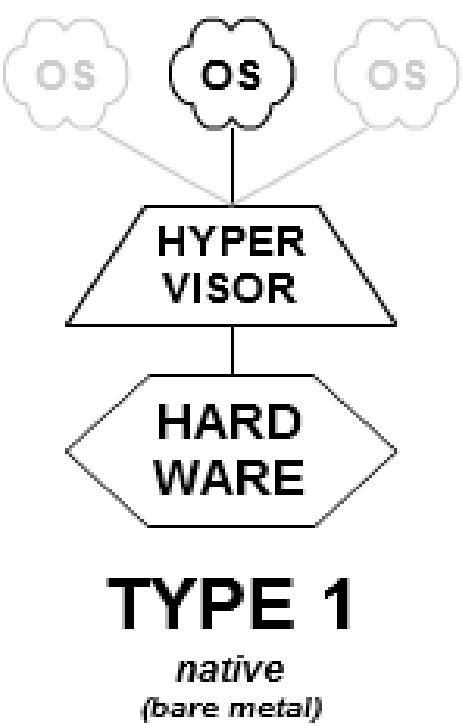
System Virtual Machine

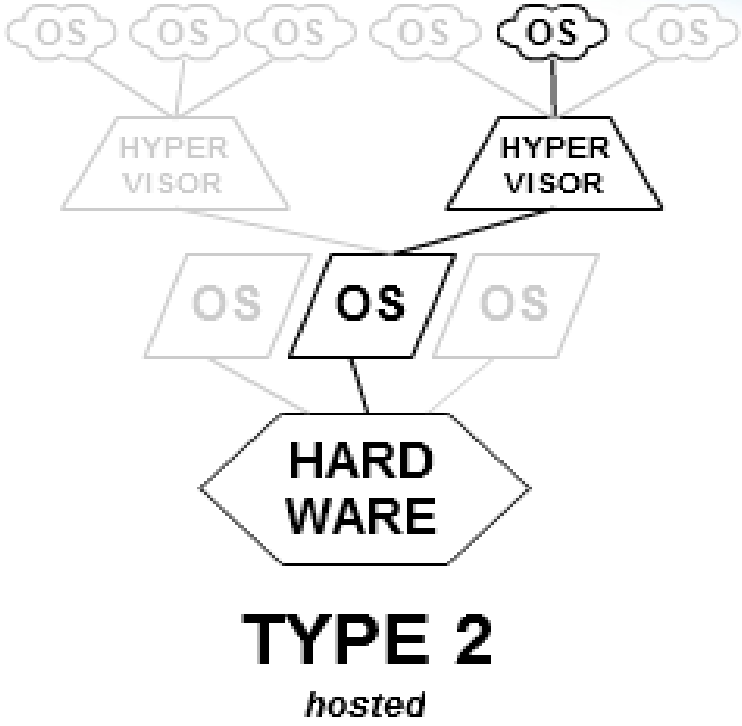
- System virtual machine is controlled by a hypervis or or VMM (Virtual Machine Monitor)
- A hypervisor or VMM is a software to provide a ha rdware emulation interface including CPU, memor y, I/O by multiplexing host resources

Two Types of Hypervisor (1/2)

- In their 1974 article "Formal Requirements for Virtualizable Third Generation Architectures" Gerald J. Popek and Robert P. Goldberg classified two types of hypervisor:
  - Type 1 hypervisor : bare metal type
  - Type 2 hypervisor : hosted type

Two Types of Hypervisor (2/2)





Type 1 (or *native*, bare metal) hypervisors

Type 2 (or *hosted*) hypervisors

http://en.wikipedia.org/wiki/Hypervisor

Purposes of Hypervisor

- CPU Virtualization
  - Handle all sensitive instructions by emulation
- Memory Virtualization
  - Allocate guest physical memory
  - Translate guest virtual address to host virtual address
- I/O Virtualization
  - Emulate I/O devices for guest
  - Ex: Keyboard, UART, Storage and Network

CPU Virtualization

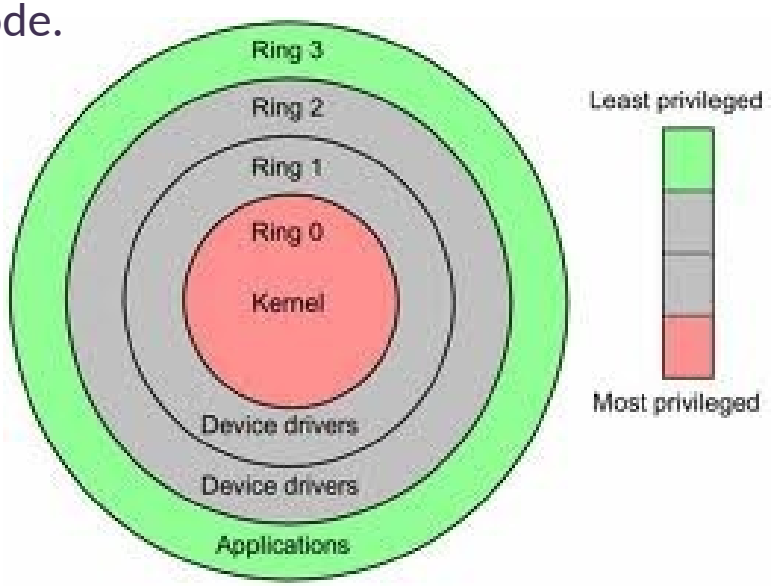
Category of instructions

- In architecture field, the CPU designers separate instructions into different categories.
  - Privilege instruction
    - Those instructions are trapped if the machine is in user mode and are not trapped if the machine is in kernel mode.
    - ex: Instruction to modify page table base register
  - Non-Privilege instruction
    - All other instructions
    - ex: Software interrupt, Normal arithmetic operation

- In virtualization field, the hypervisor designers separate instructions into two categories.
  - Sensitive instruction
    - Those instructions that interact with hardware, which include control-sensitive and behavior-sensitive instructions.
    - ex: Instruction to modify page table base register, Software Interrupt, …
  - Non-sensitive instruction
    - All other instructions
    - ex: Normal arithmetic operation, …

Privilege instruction

- Take x86 architecture for example:
  - Kernel mode (Ring 0)
    - CPU may perform any operation allowed by its architecture, including any instruction execution, IO operation, area of memory access, and so on.
    - Traditional OS kernel runs in Ring 0 mode.
  - User mode (Ring 1 ~ 3)
    - CPU can typically only execute a subset of those available instructions in kernel mode.
    - Traditional application runs in Ring 3 mode.



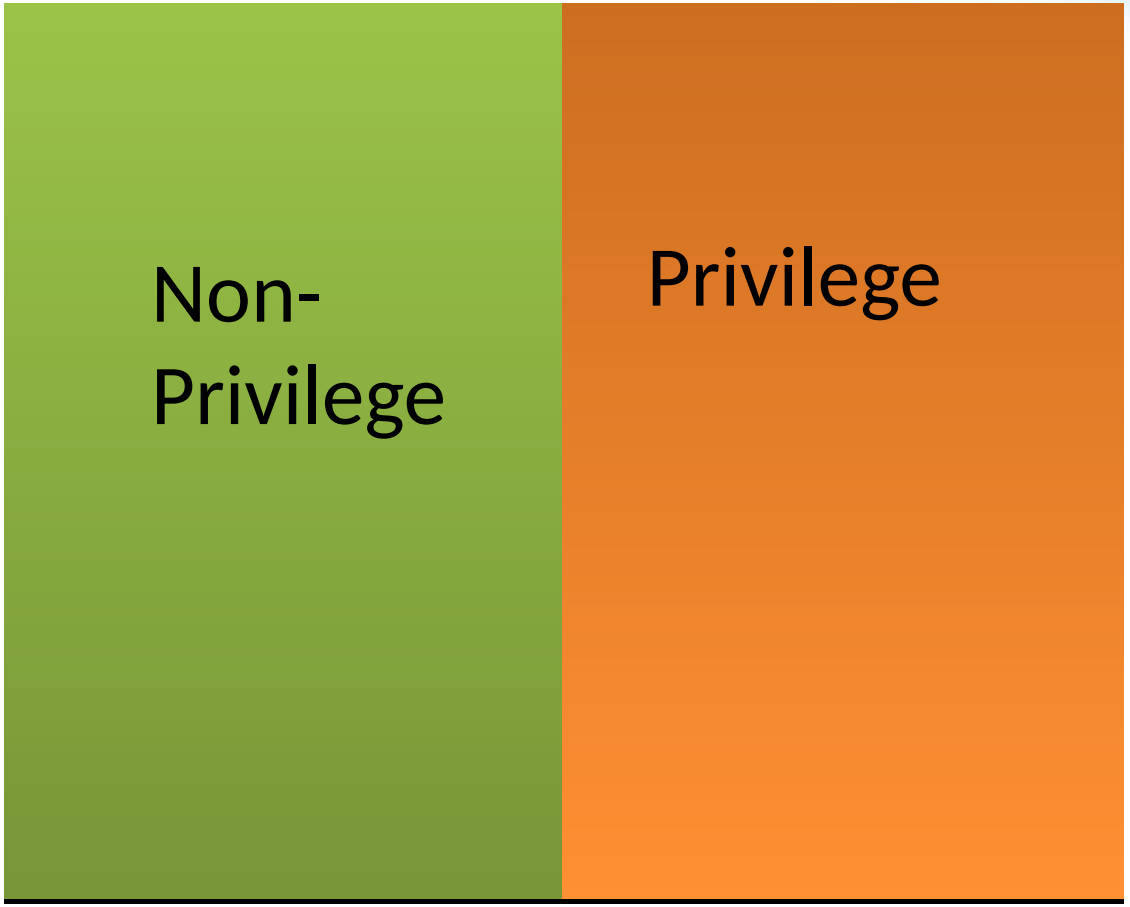
Sensitive Instruction

- Those instructions that interact with hardware, which include control-sensitive and behavior-sensitive instructions
- 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).

Virtualizable CPU

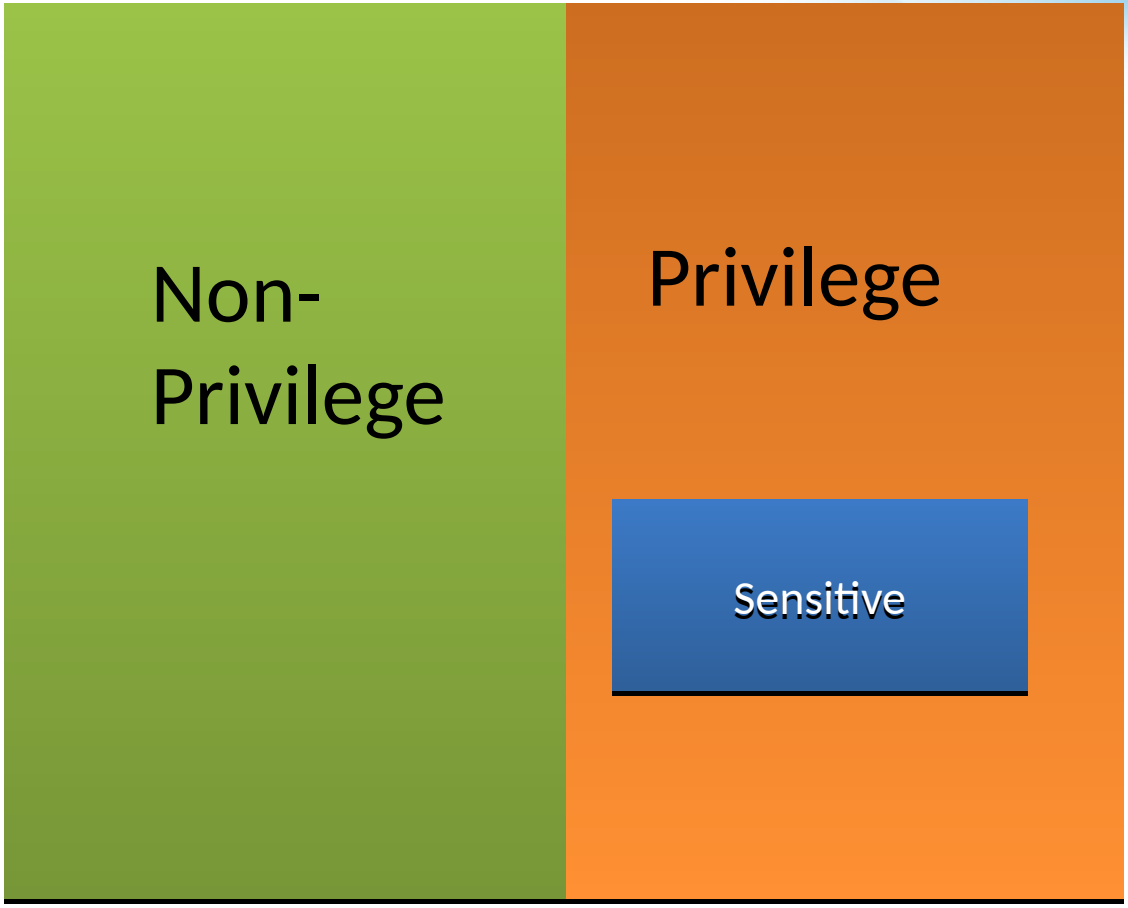
Privilege and Non-Privilege

ps: whole circle is a set of all instructions



Privilege and Sensitive

ps: whole circle is a set of all instructions



All sensitive are privilege: Virtualizable CPU

Virtualizable CPU

- All of sensitive instructions are privilege instructions.
- We call this kind of CPU as “Virtualizable CPU”
- For “Virtualizable CPU”, it is quite easy to implement hypervisor. All you have to do is to put hypervisor in privilege mode and Guest OS in non-privilege mode.
- When Guest OS wants to execute sensitive instructions, the execution will be trapped to hypervisor which is running on privilege mode automatically.
  - By this way, we can make sure that there is no chance for Guest OS to change any important hardware resource directly. All important hardware resource is under management by hypervisor.

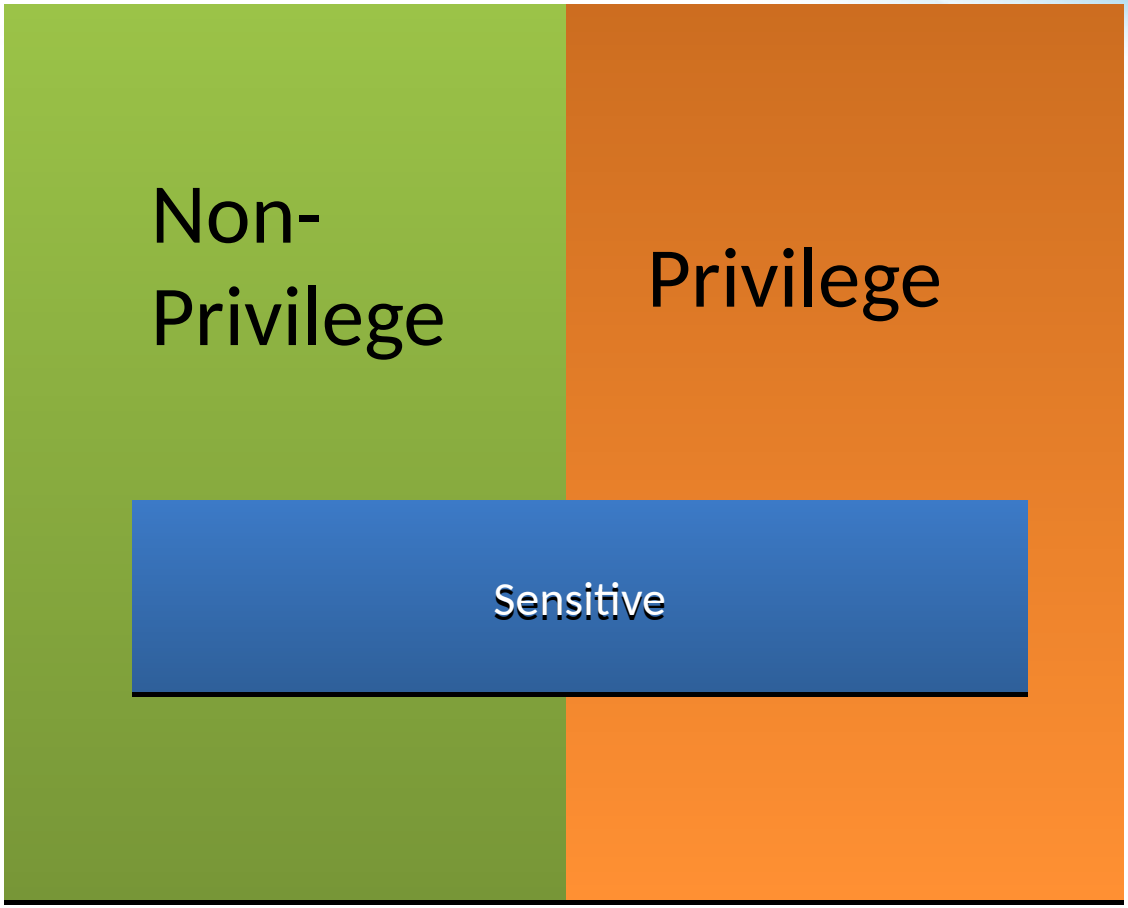
Virtualizable CPU

- Example:
  - IBM PowerPC
  - IBM S/390
    - CPU for IBM mainframe

Non-Virtualizable CPU

Privilege and Sensitive

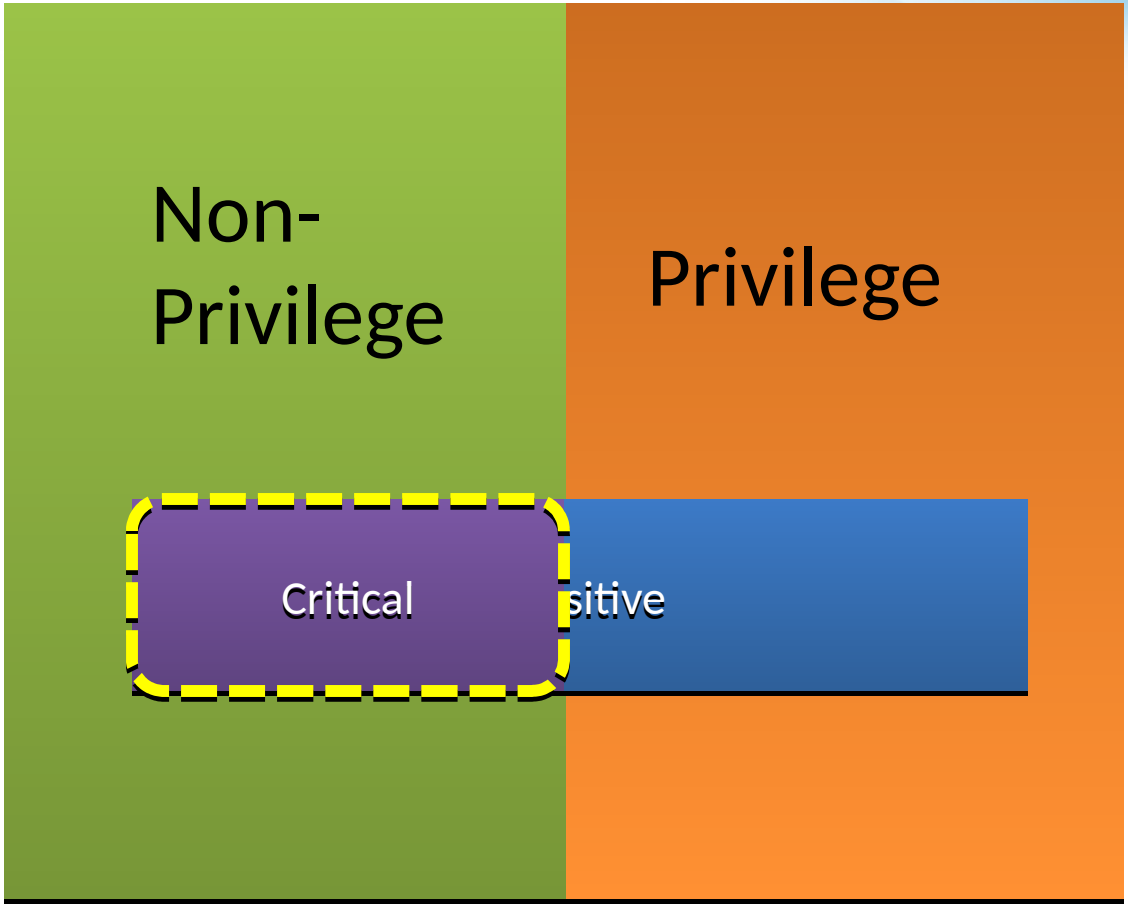
ps: whole circle is a set of all instructions



Some sensitive are non-privilege: Non-Virtualizable CPU

Privilege and Sensitive

ps: whole circle is a set of all instructions



Sensitive but Non-Privilege instruction is the problem.

We call “Sensitive but Non-Privilege instruction” as “Critical Instruction”.

Non-Virtualizable CPU

- Some of sensitive instructions are privilege instructions. But some of sensitive instructions are non-privilege instructions.
- We call this kind of CPU as “Non-Virtualizable CPU”
- For “Non-Virtualizable CPU”, it is HARD to implement hypervisor.
- If you put hypervisor in privilege mode and Guest OS in non-privilege mode, when Guest OS wants to execute sensitive instructions,
  - For those privilege and sensitive instructions, they (which is) will be trapped into hypervisor which is running on privilege mode automatically.
  - For those critical instructions, they will NOT be trapped into hypervisor automatically.

Critical instruction annoy us!

- Critical instruction can be executed in privilege mode and non-privilege mode.
- The behaviors of critical instructions in privilege mode and non-privilege mode are different. It will cause problems.
- As a result, hypervisor designers have to let critical instructions be trapped to hypervisor, and let hypervisor emulate their behaviors.

Non-virtualizable CPU

- Example:
  - x86
  - ARM

What is sensitive instruction?

How to “trap and emulate”?

SENSITIVE INSTRUCTION

CPU Architecture

- What is trap ?
  - When CPU is running in user mode, some internal or external events, which need to be handled in kernel mode, take place.
  - Then CPU will jump to hardware exception handler vector, and execute system operations in kernel mode.
- Trap types :
  - System Call
    - Invoked by application in user mode.
    - For example, application ask OS for system IO.
  - Hardware Interrupts
    - Invoked by some hardware events in any mode.
    - For example, hardware clock timer trigger event.
  - Exception
    - Invoked when unexpected error or system malfunction occur.
    - For example, execute privilege instructions in user mode.

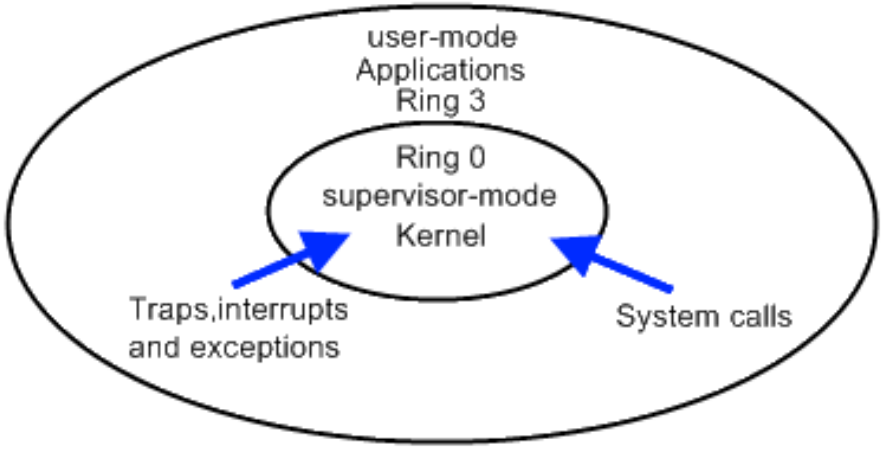
Trap and Emulate Model

- If we want CPU virtualization to be efficient, how should we implement the VMM ?
  - We should make guest binaries run on CPU as fast as possible.
  - Theoretically speaking, if we can run all guest binaries natively, there will NO overhead at all.
  - But we cannot let guest OS handle everything, VMM should be able to control all hardware resources.

- Solution :
  - Ring Compression
    - Shift traditional OS from kernel mode(Ring 0) to user mode(Ring 1), and run VMM in kernel mode.
    - Then VMM will be able to intercept all trapping event.

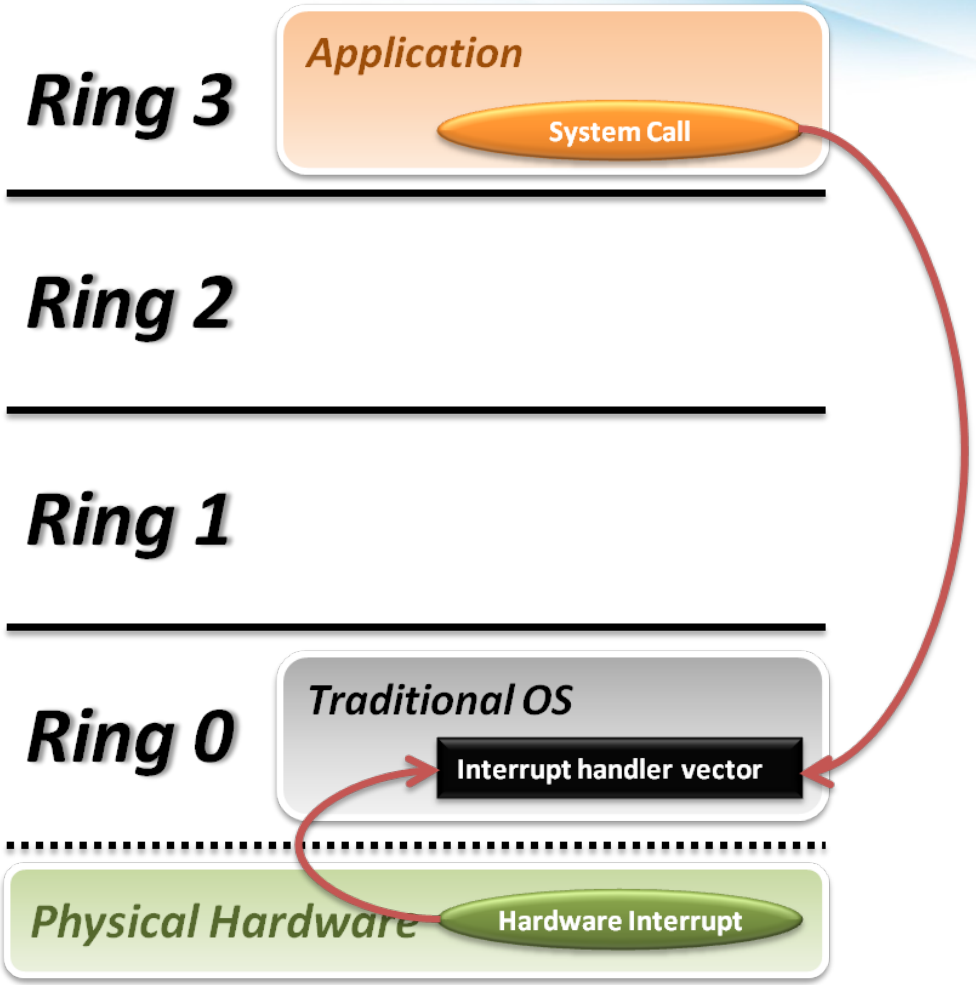
Trap and Emulate Model

- VMM virtualization paradigm *(trap and emulate)* :
  1. Let normal instructions of guest OS run directly on processor in user mode.
  2. When executing privilege instructions, hardware will make processor trap into the VMM.
  3. The VMM emulates the effect of the privilege instructions for the guest OS and return to guest.



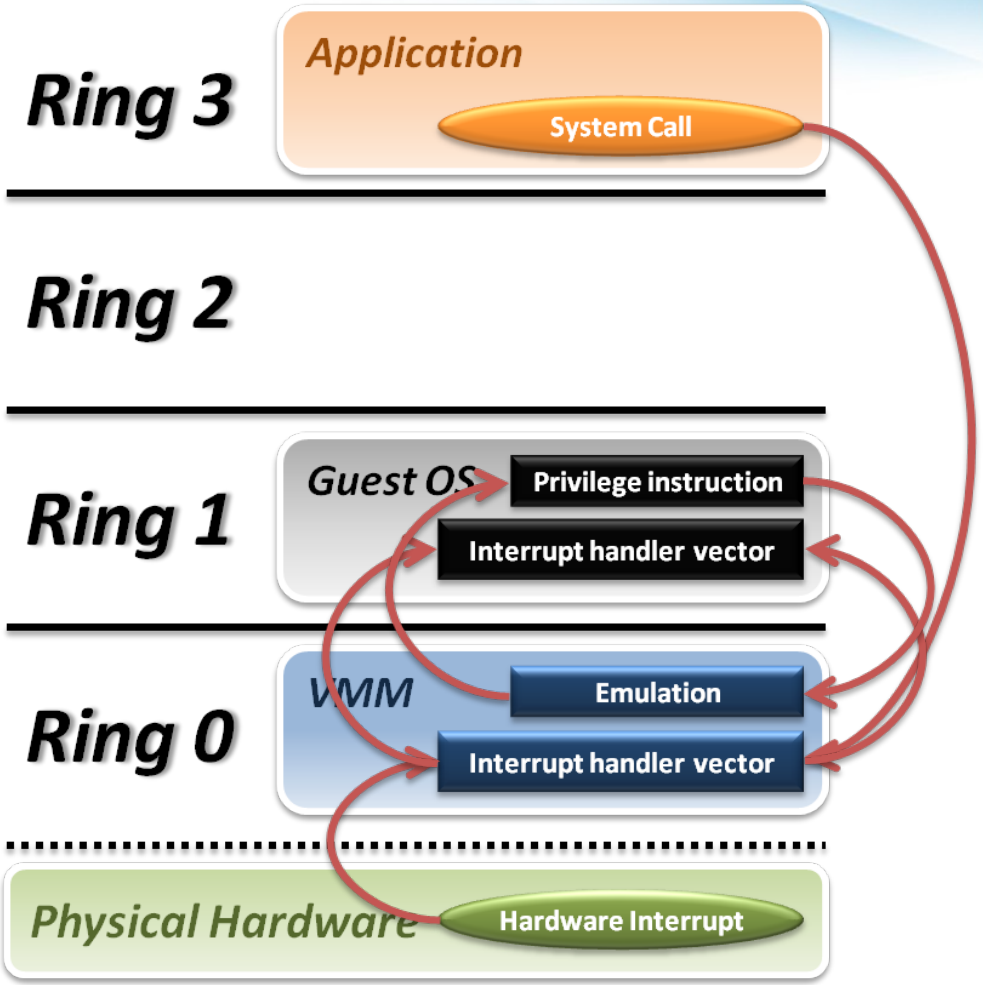
Trap and Emulate Model

- Traditional OS :
  - When application invoke a system call :
    - CPU will trap to interrupt handler vector in OS.
    - CPU will switch to kernel mode (Ring 0) and execute OS instructions.
  - When hardware event :
    - Hardware will interrupt CPU execution, and jump to interrupt handler in OS.



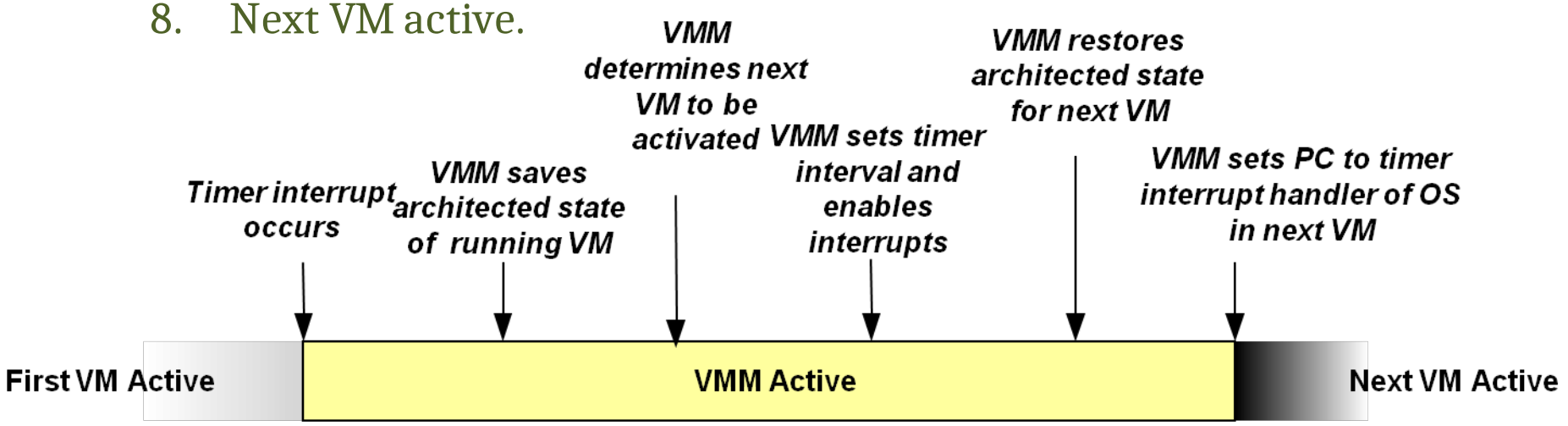
Trap and Emulate Model

- VMM and Guest OS :
  - System Call
    - CPU will trap to interrupt handler vector of VMM.
    - VMM jump back into guest OS.
  - Hardware Interrupt
    - Hardware make CPU trap to interrupt handler of VMM.
    - VMM jump to corresponding interrupt handler of guest OS.
  - Privilege Instruction
    - Running privilege instructions in guest OS will be trapped to VMM for instruction emulation.
    - After emulation, VMM jump back to guest OS.



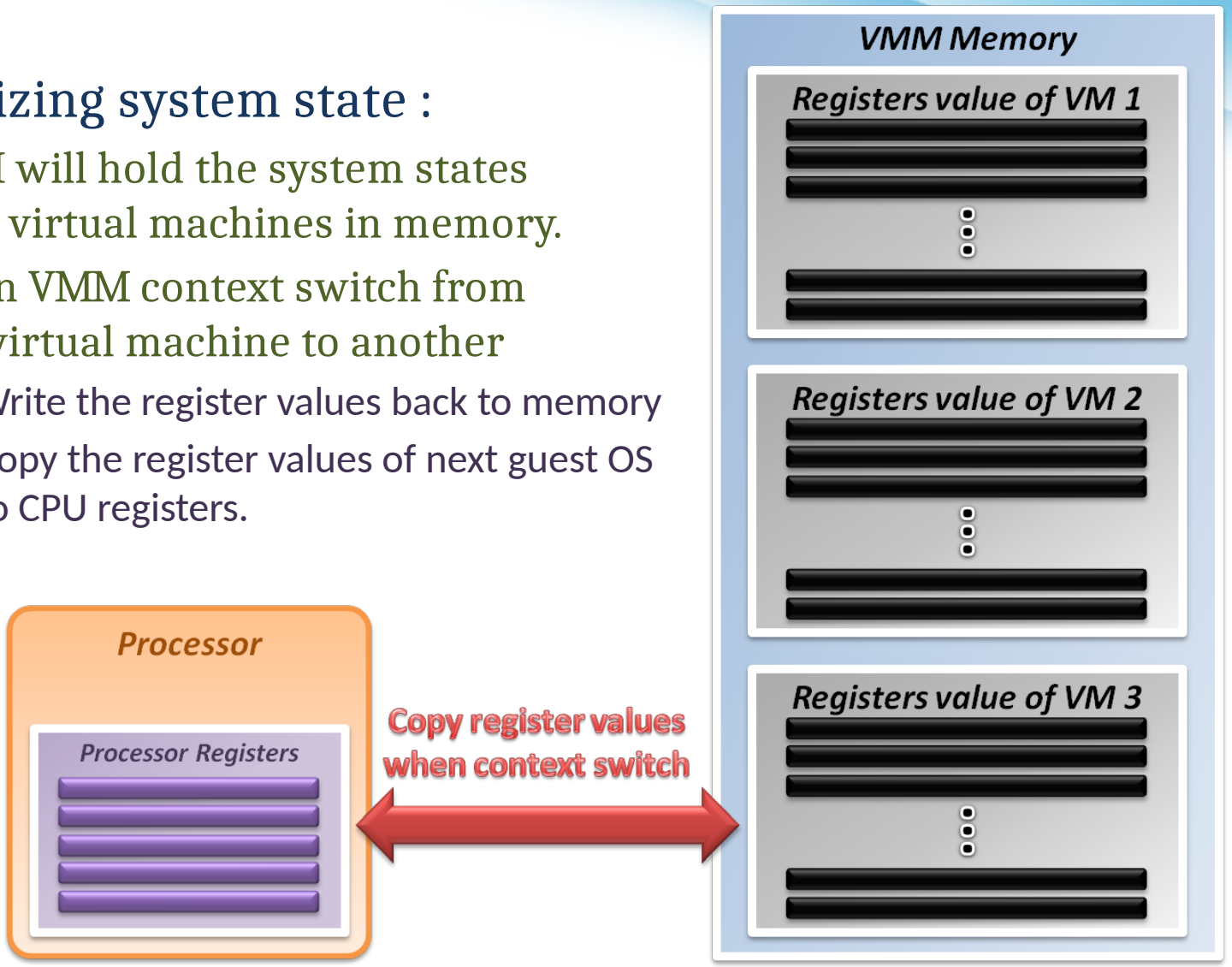
Context Switch

- Steps of VMM switch different virtual machines :
  1. Timer Interrupt occurs in running VM.
  2. Context switch to VMM.
  3. VMM saves state of running VM.
  4. VMM determines next VM to execute.
  5. VMM sets timer interrupt.
  6. VMM restores state of next VM.
  7. VMM sets PC to timer interrupt handler of next VM.
  8. Next VM active.



System State Management

- Virtualizing system state :
  - VMM will hold the system states of all virtual machines in memory.
  - When VMM context switch from one virtual machine to another
    - Write the register values back to memory
    - Copy the register values of next guest OS to CPU registers.



Virtualization Theorem

- Subset theorem :
  - For any conventional third-generation computer, a VMM may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.
- Recursive Emulation :
  - A conventional third-generation computer is recursively virtualizable if
    - It is virtualizable
    - VMM without any timing dependencies can be constructed for it.
- Under this theorem, x86 or ARM architecture cannot be virtualized directly. Other techniques are needed.

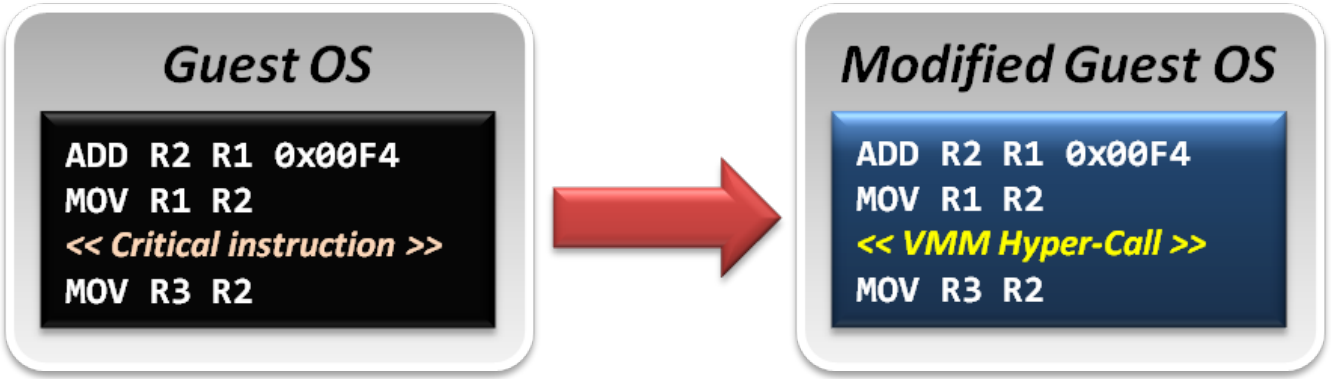
Virtualization Techniques

- How to virtualize non-virtualizable hardware :
  - Para-virtualization
    - Modify guest OS to skip the critical instructions.
    - Implement some hyper-calls to trap guest OS to VMM.
  - Binary translation
    - Use emulation technique to make hardware virtualizable.
    - Skip the critical instructions by means of these translations.
  - Hardware assistance
    - Modify or enhance ISA of hardware to provide virtualizable architecture.
    - Reduce the complexity of VMM implementation.

Para-Virtualization: Patch Guest OS

Para-Virtualization

- Para-Virtualization implementation :
  - In para-virtualization technique, guest OS should be modified to prevent invoking critical instructions.
  - Instead of knowing nothing about hypervisor, guest OS will be aware of the existence of VMM, and collaborate with VMM smoothly.
  - VMM will provide the hyper-call interfaces, which will be the communication channel between guest and host.



Some Difficulties

- Difficulty of para-virtualization :
  - Guest OS modification
    - User should at least has the source code of guest OS; otherwise, para-virtualization cannot be used.

Full-Virtualization: Dynamic Binary Translation

Binary Translation

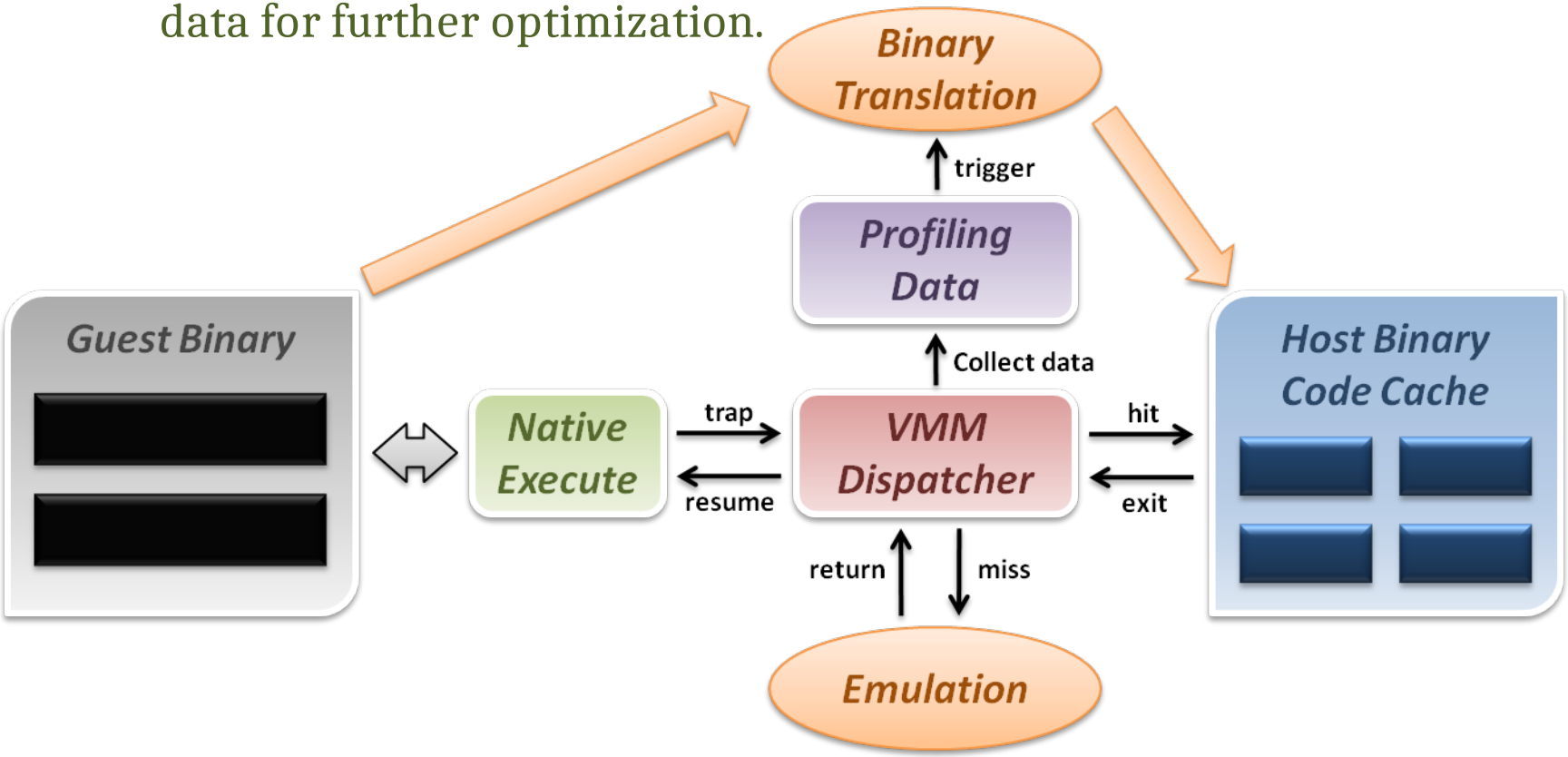
- In emulation techniques :
  - Binary translation module is used to optimize binary code blocks, and translates binaries from guest ISA to host ISA.
- In virtualization techniques :
  - Binary translation module is used to skip or modify the guest OS binary code blocks which include critical instructions.
  - Translate those critical instructions into some privilege instructions which will be trapped to VMM for further emulation.

Binary Translation (revisited)

- Static approach vs. Dynamic approach :
  - Static binary translation
    - The entire executable file is translated into an executable of the target architecture.
    - This is very difficult to do correctly, since not all the code can be discovered by the translator.
  - Dynamic binary translation
    - Looks at a short sequence of code, typically on the order of a single basic block, translates it and caches the resulting sequence.
    - Code is only translated as it is discovered and when possible, branch instructions are made to point to already translated and saved code.

Dynamic Binary Translation (revisited)

- Dynamic binary translation and optimization
  - VMM can dynamically translate binary code and collect profiling data for further optimization.



Some Difficulties

- Difficulties of binary translation :
  - Self-modifying code
    - If guest OS will modify its own binary code in runtime, binary translation needs to flush the corresponding code cache and retranslates the code block.
  - Self-reference code
    - If guest code needs to read its own binary code in runtime, VMM needs to make the referring back to original guest binaries location.
  - Real-time system
    - For some timing critical guest OS, emulation environment will lose precise timing, and this problem cannot be perfectly solved yet.

Full-Virtualization: Hardware Assistant

Hardware Solution

- Why are there so many problems and difficulties ?
  - Critical instructions do not trap in user mode.
  - Even we make those critical instructions trap, their semantic may also be changed; which is not acceptable.
- In short, legacy processors did not design for virtualization purpose at the beginning.
  - If processor can be aware of the different behaviors between guest and host, the VMM design will be more efficient and simple.

Hardware Solution

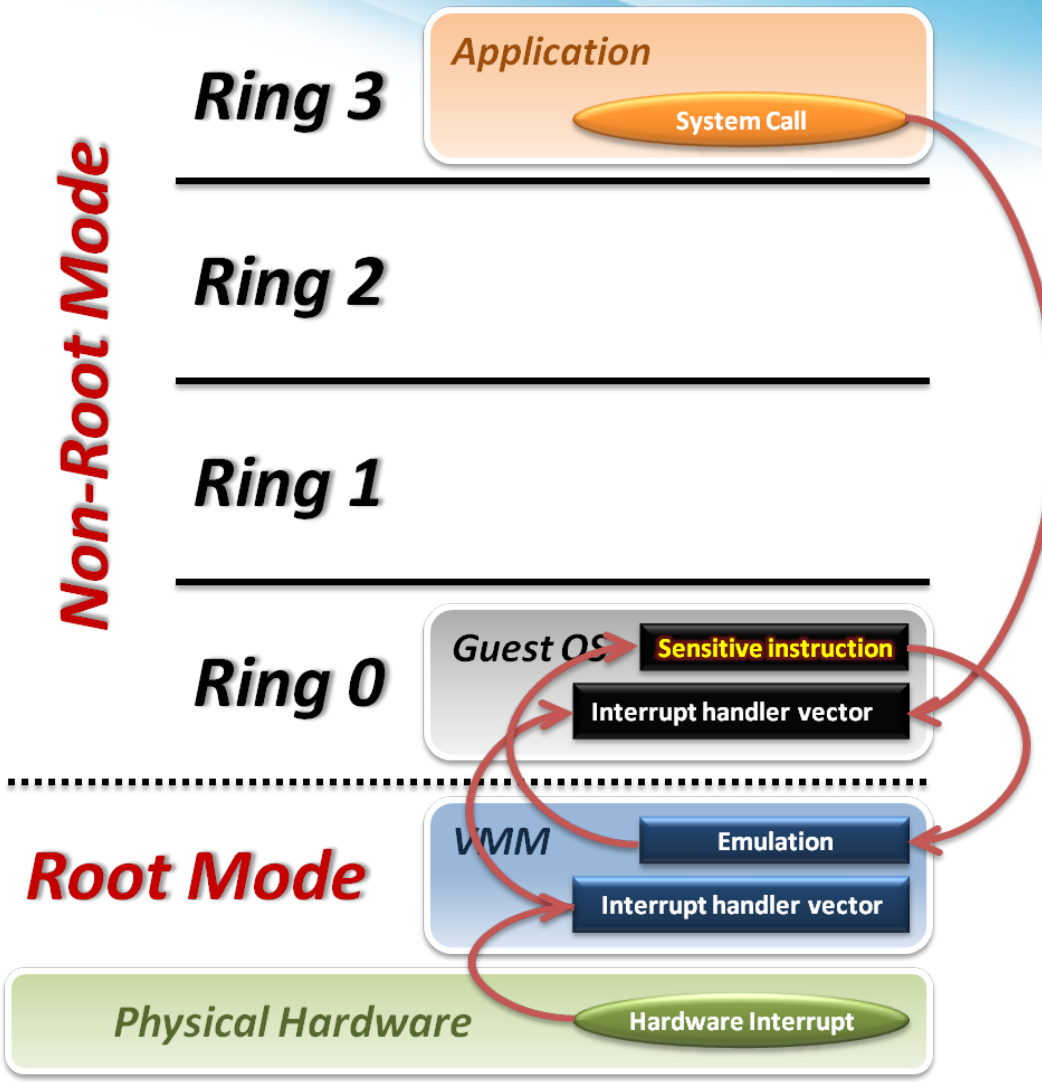
- Let's go back to trap model :
  - Some trap types do not need the VMM involvement.
    - For example, all system calls invoked by applications in Guest OS should be caught by Guest OS only. There is no need to trap to VMM and then forward it back to guest OS, which will introduce context switch overhead.
  - Some critical instructions should not be executed by guest OS.
    - Although we make those critical instructions trap to VMM, VMM cannot identify whether this trapping action is caused by the emulation purpose or the real OS execution exception.
- Solution :
  - We need to redefine the semantic of some instructions.
  - We need to introduce new CPU control paradigm.

Intel VT-x

- In order to straighten those problems out, Intel introduces one more operation mode of x86 architecture.
  - VMX Root Operation (Root Mode)
    - All instructions in this mode are no different to traditional ones.
    - All legacy software can run in this mode correctly.
    - VMM should run in this mode and control all system resources.
  - VMX Non-Root Operation (Non-Root Mode)
    - All sensitive instructions in this mode are redefined.
    - The sensitive instructions will trap to Root Mode.
    - Guest OS should run in this mode and be fully virtualized through typical "*trap and emulation model*".

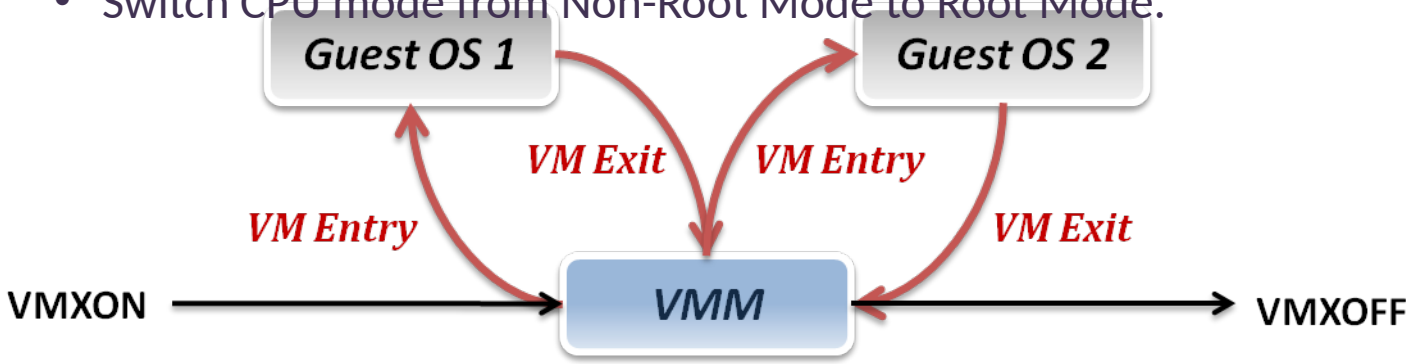
Intel VT-x

- VMM with VT-x :
  - System Call
    - CPU will directly trap to interrupt handler vector of guest OS.
  - Hardware Interrupt
    - Still, hardware events need to be handled by VMM first.
  - Sensitive Instruction
    - Instead of trap all privilege instructions, running guest OS in Non-root mode will trap sensitive instruction only.



Context Switch

- VMM switches different virtual machines with Intel VT-x :
  - **VMXON/VMXOFF**
    - These two instructions are used to turn on/off CPU Root Mode.
  - VM Entry
    - This is usually caused by the execution of **VMLAUNCH/VMRESUME** instructions, which will switch CPU mode from Root Mode to Non-Root Mode.
  - VM Exit
    - This may be caused by many reasons, such as hardware interrupts or sensitive instruction executions.
    - Switch CPU mode from Non-Root Mode to Root Mode.



System State Management

- Intel introduces a more efficient hardware approach for register switching, **VMCS** (*Virtual Machine Control Structure*) :
  - State Area
    - Store Host OS system state when VM-Entry.
    - Store Guest OS system state when VM-Exit.
  - Control Area
    - Control instruction behaviors in Non-Root Mode.
    - Control VM-Entry and VM-Exit process.
  - Exit Information
    - Provide the VM-Exit reason and some hardware information.
- Whenever VM Entry or VM Exit occur, CPU will automatically read or write corresponding information into VMCS.

System State Management

- Binding virtual machine to virtual CPU
  - VCPU (Virtual CPU) contains two parts
    - VMCS maintains virtual system states, which are handled by hardware.
    - Non-VMCS maintains other non-essential system information, which is handled by software.
  - VMM needs to handle Non-VMCS part.