Cloud Computing
CS 15-319

Virtualization- Part III

Lecture 19, April 2, 2012

Majd F. Sakr and Mohammad Hammoud

Today…

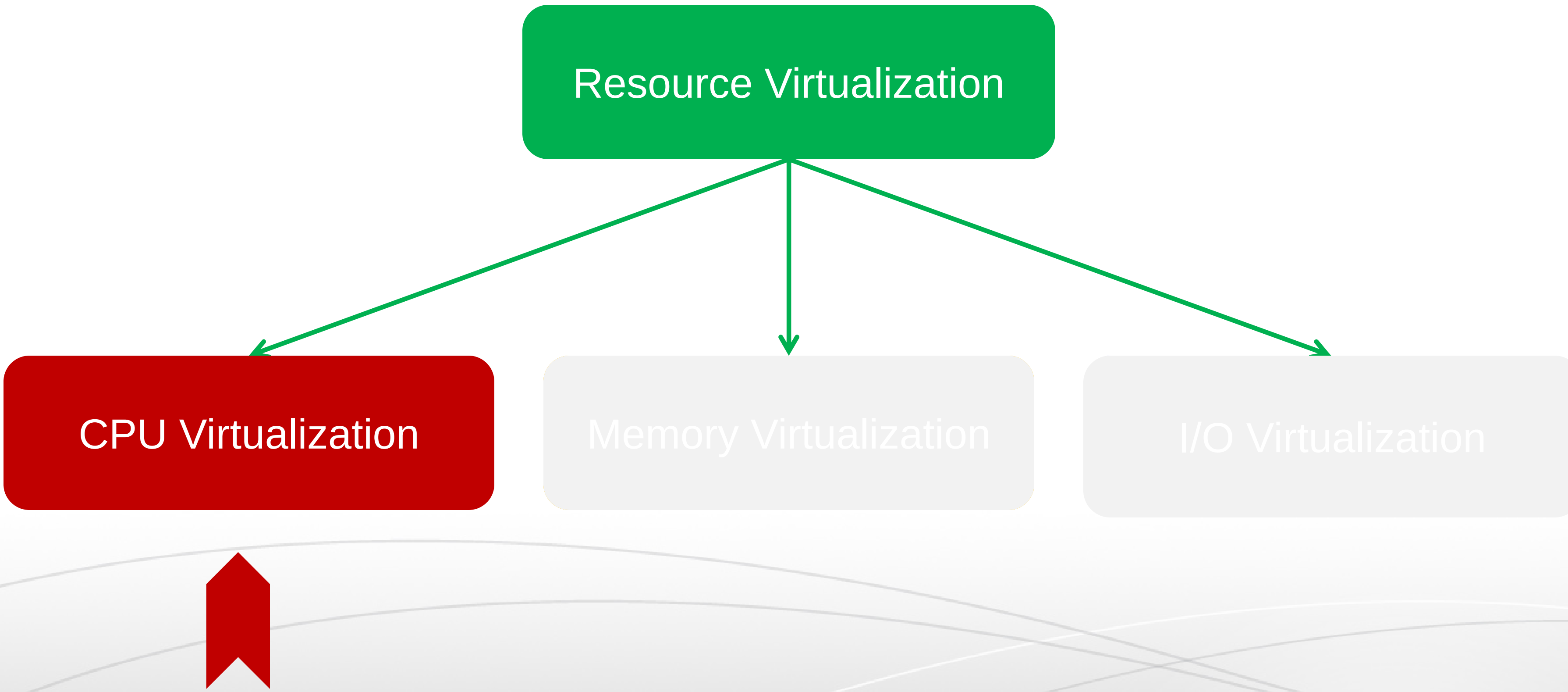
- Last session
  - Virtualization Part II
- Today's session
  - Virtualization – *Part III*
- Announcement:
  - Project update/discussion is due on Wed April, 4

Objectives

Discussion on Virtualization

- Why virtualization, and virtualization properties
- Virtualization, para-virtualization, virtual machines and hypervisors
- Virtual machine types
- Partitioning and Multiprocessor virtualization
- Resource virtualization

Resource Virtualization



CPU Virtualization

- Interpretation and Binary Translation
- Virtualizable ISAs

CPU Virtualization

- Interpretation and Binary Translation
- Virtualizable ISAs

Binary Translation

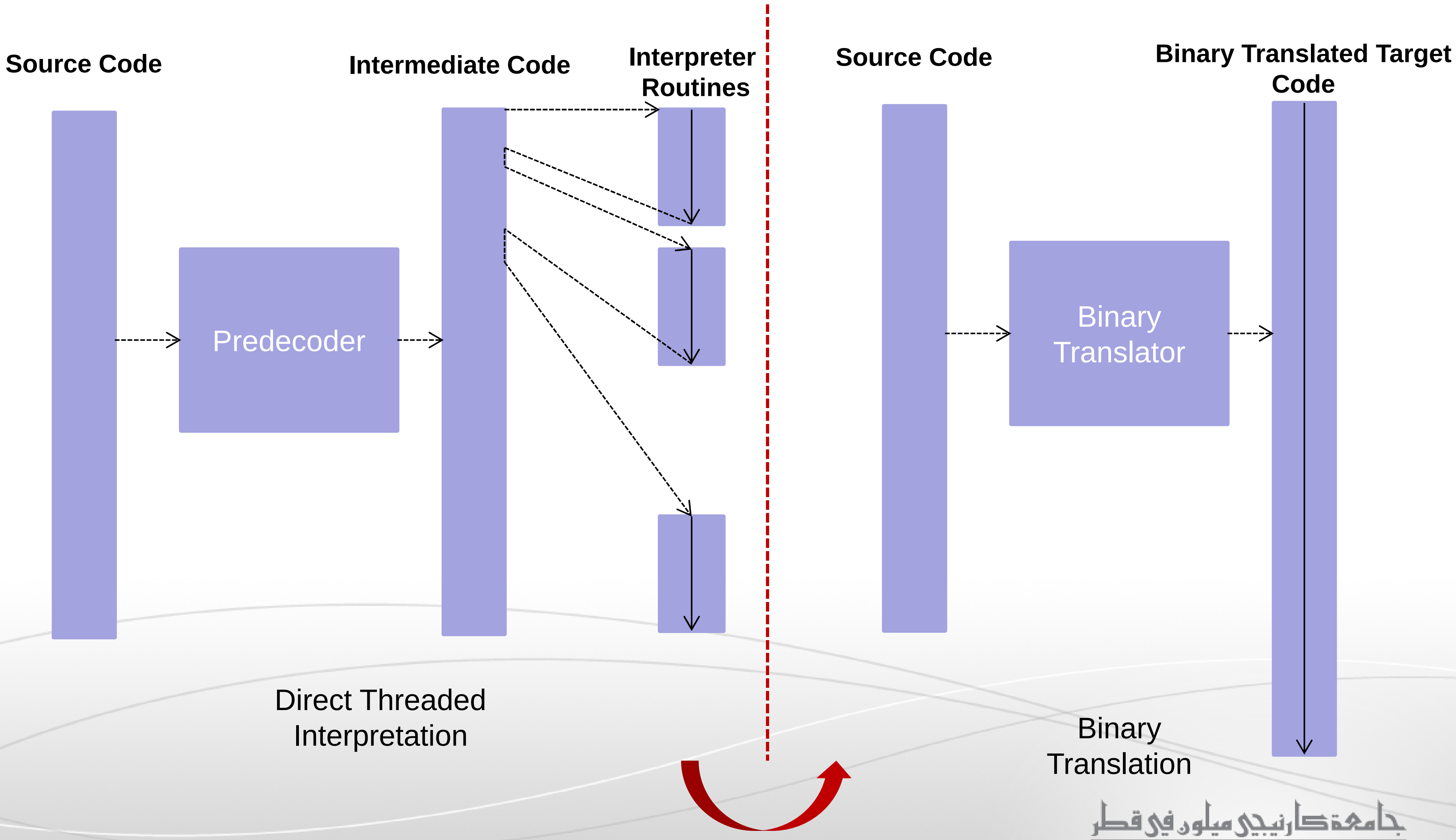
- Performance can be significantly enhanced by mapping each individual source binary instruction to its own customized target code

- This process of converting the *source binary program* into a *target binary program* is referred to as binary translation

- Binary translation attempts to amortize the fetch and analysis costs by:

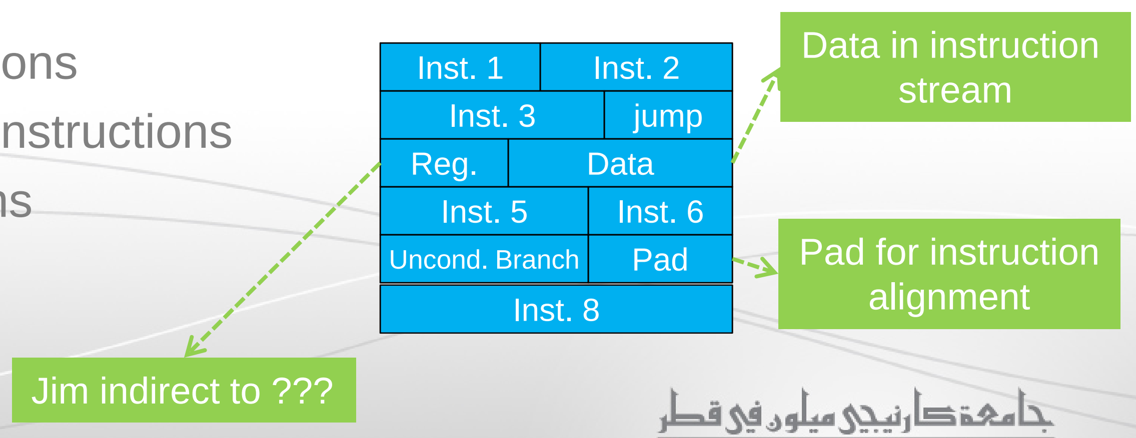
  1. Translating a block of source instructions to a block of target instructions
  2. Caching the translated code for repeated use

Binary Translation



Static Binary Translation

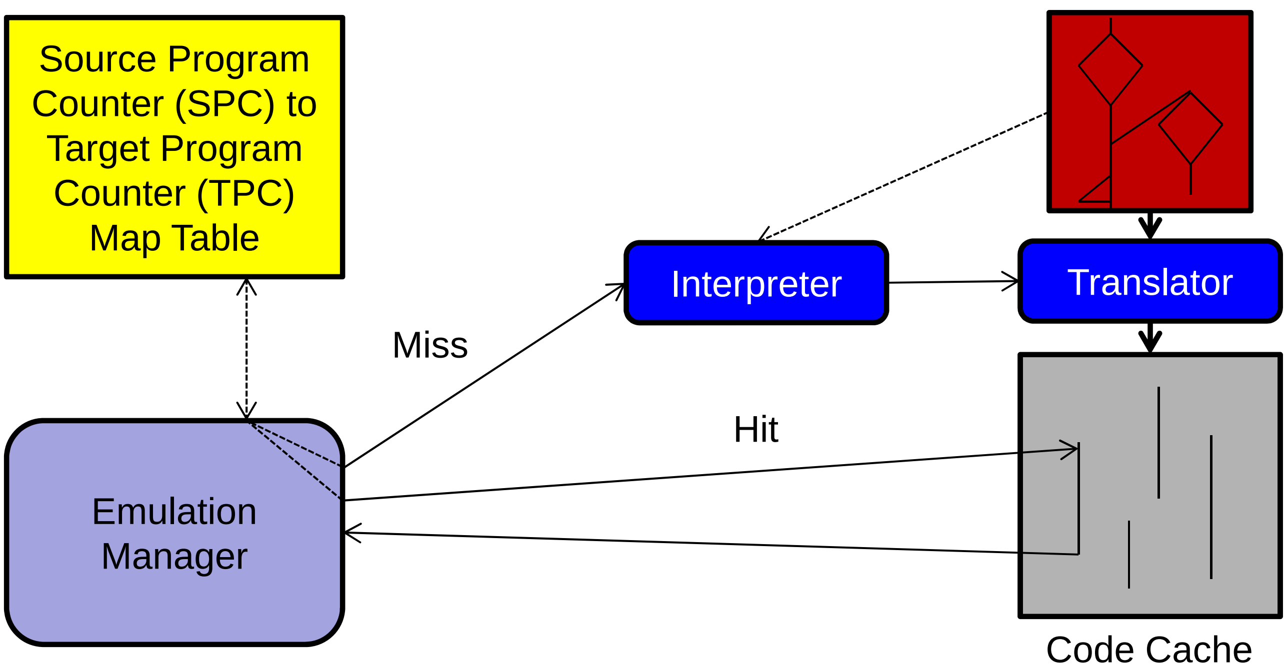
- It is possible to binary translate a program in its entirety before executing the program
- This approach is referred to as static binary translation
- However, in real code using conventional ISAs, especially CISC ISAs, such a static approach can cause problems due to:
  - Variable-length instructions
  - Data interspersed with instructions
  - Pads to align instructions
  - Register indirect jumps



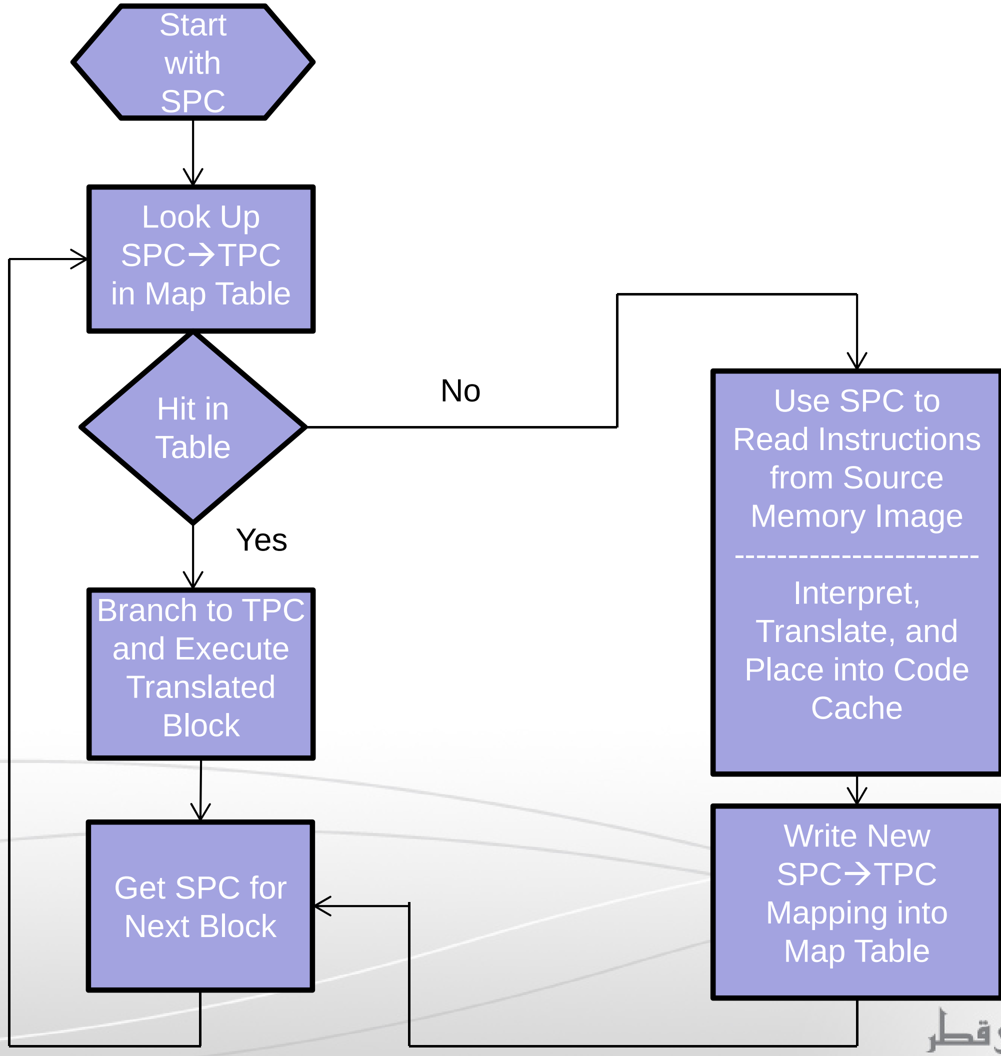
Dynamic Binary Translation

- A general solution is to translate the binary while the program is operating on actual input data (i.e., *dynamically*) and interpret new sections of code *incrementally* as the program reaches them

- This scheme is referred to as dynamic binary translation



Dynamic Binary Translation



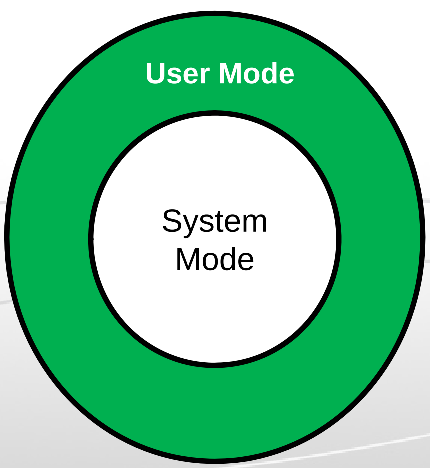
CPU Virtualization

- Interpretation and Binary Translation
- Virtualizable ISAs

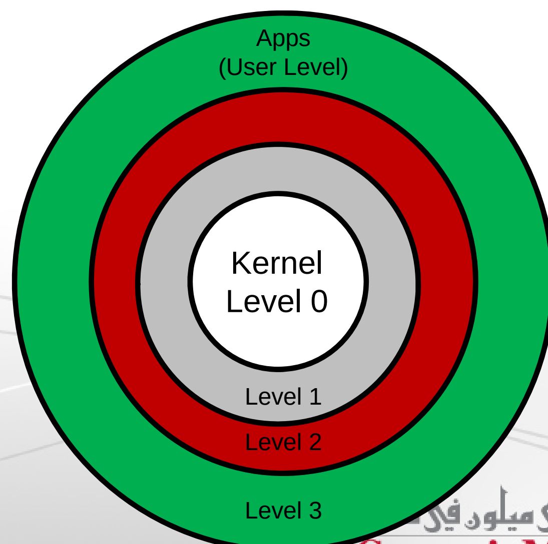
Privilege Rings in a System

- In the ISA, special privileges to system resources are permitted by defining modes of operations

- Usually an ISA specifies at least two modes of operation:
  1. System (also called supervisor, kernel, or privileged) mode: all resources are accessible to software
  2. User mode: only certain resources are accessible to software



Simple systems have 2 rings



Intel's IA-32 allows 4 rings

Privileged Instructions

- In a native system VM, the VMM runs in system mode, and all “other” (e.g., guest OS) software run in user mode

- A privileged instruction is defined as one that *traps* if the machine is in user mode and does not trap if the machine is in system mode

- Examples of Privileged Instructions are:

  - Load PSW: If it can be accessed in user mode, a malicious user program can put itself in system mode and get control of the system

  - Set CPU Timer: If it can be accessed in user mode, a malicious user program can change the amount of time allocated to it before getting context switched

جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

Types of Instructions

- Instructions that interact with hardware can be classified into three categories:

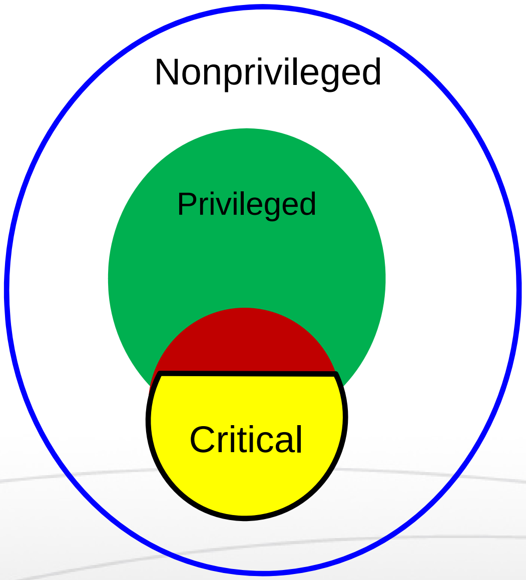
1. Control-sensitive: Instructions that attempt to change the configuration of resources in the system (e.g., memory assigned to a program)

2. Behavior-sensitive: Instructions whose behaviors or results depend on the configuration of resources

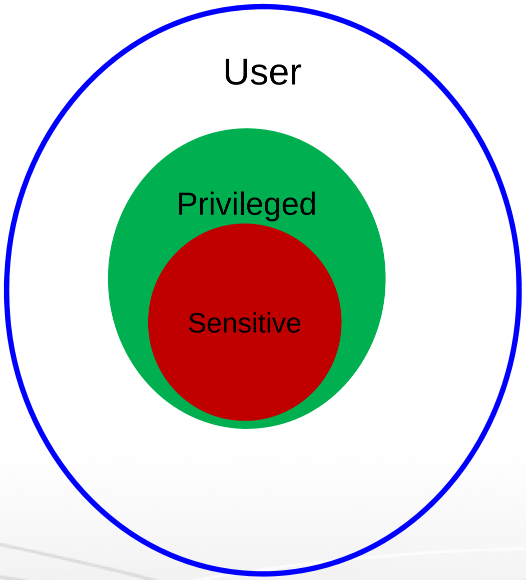
3. Innocuous: Instructions that are neither control-sensitive nor behavior-sensitive

Virtualization Theorm

- **Virtualization 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 [Popek and Goldberg, 1974]



Does not satisfy the theorem



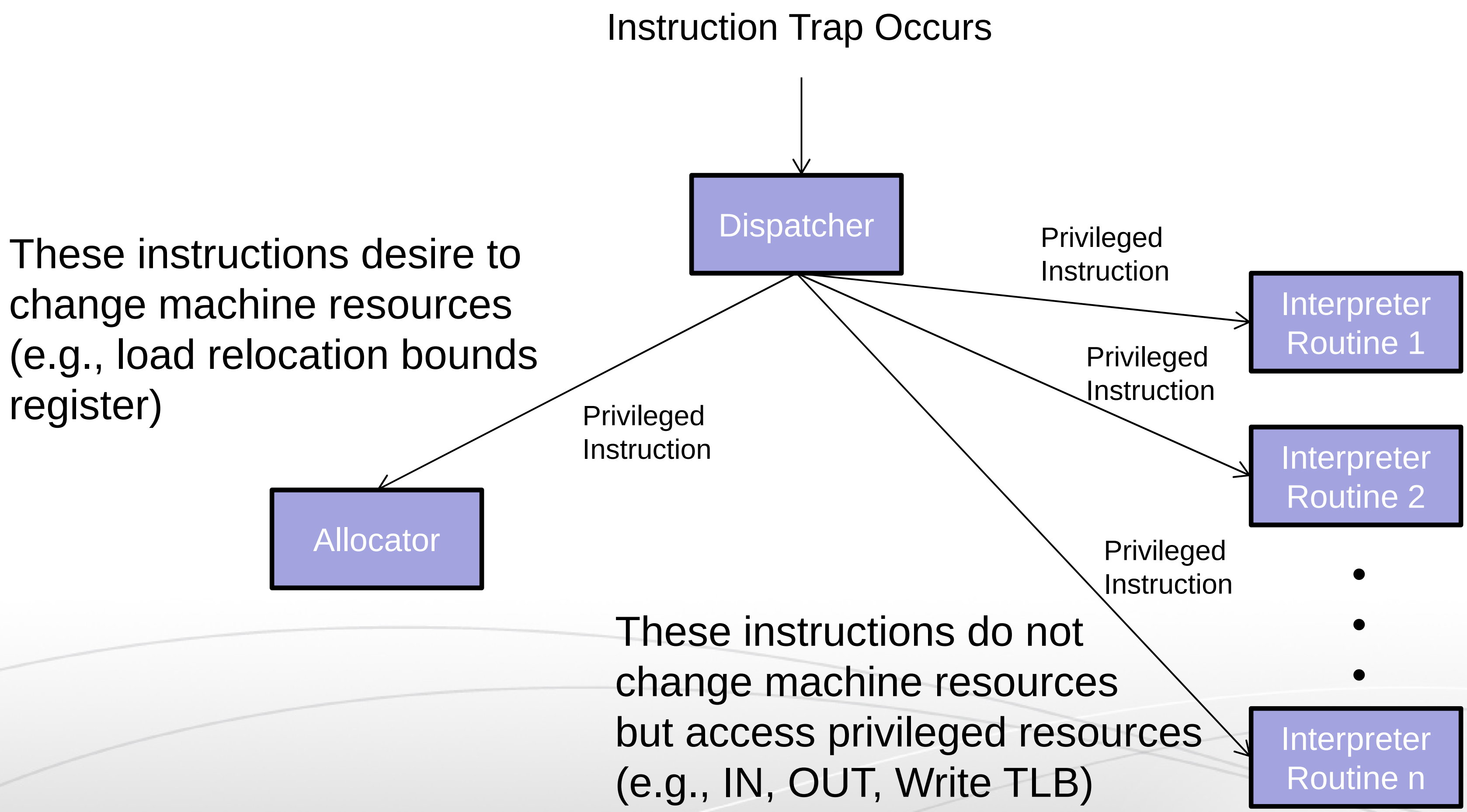
Satisfies the theorem

Efficient VM Implementation

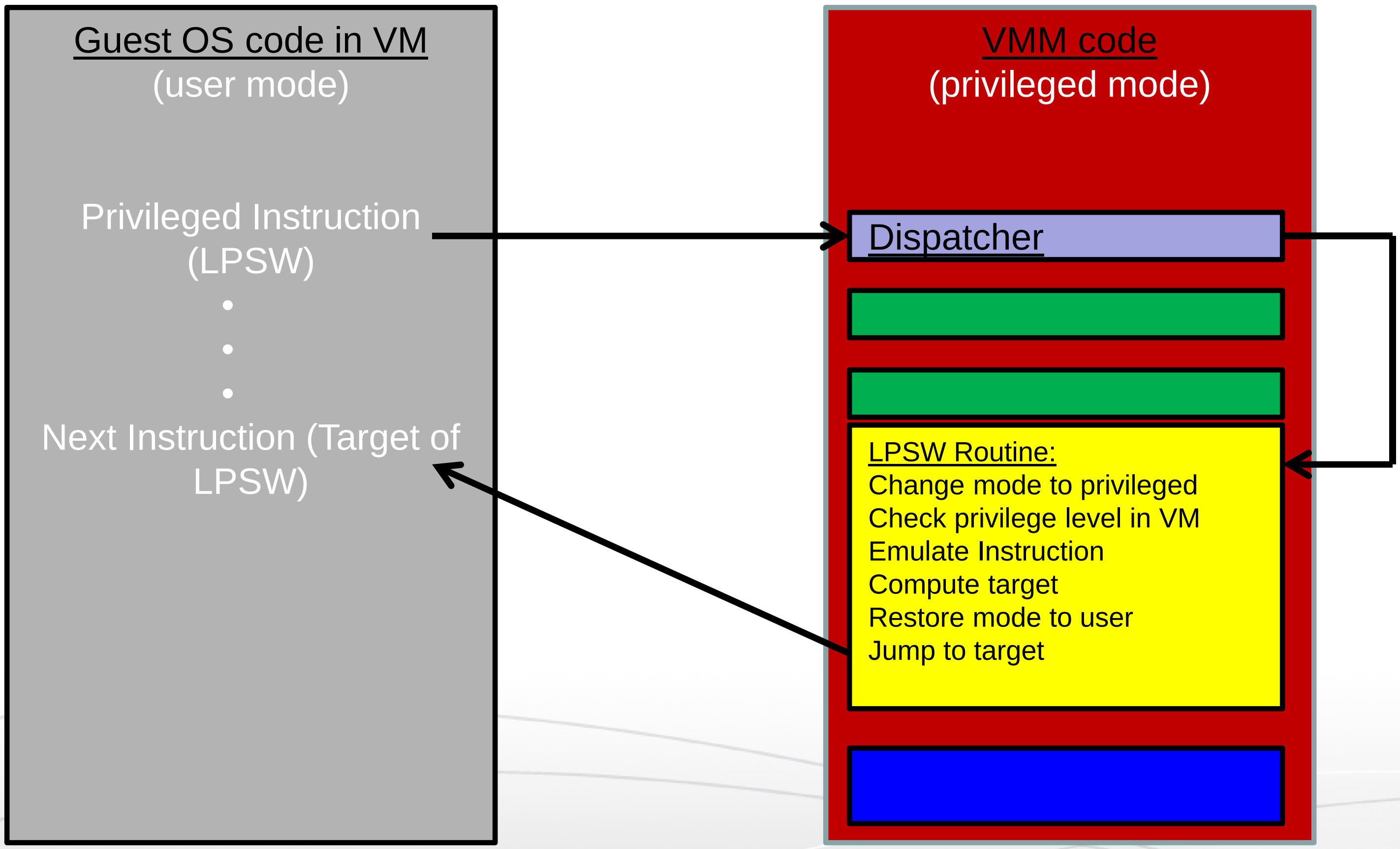
- An OS running on a guest VM should not be allowed to change hardware resources (e.g., executing PSW and set CPU timer)
- Therefore, guest OSs are all forced to run in user mode

An ***efficient*** VM implementation can be constructed if instructions that could interfere with the correct or efficient functioning of the VMM always trap in the user mode

Trapping To VMM



Handling Privileged Instructions



Carnegie Mellon Qatar

Critical Instructions

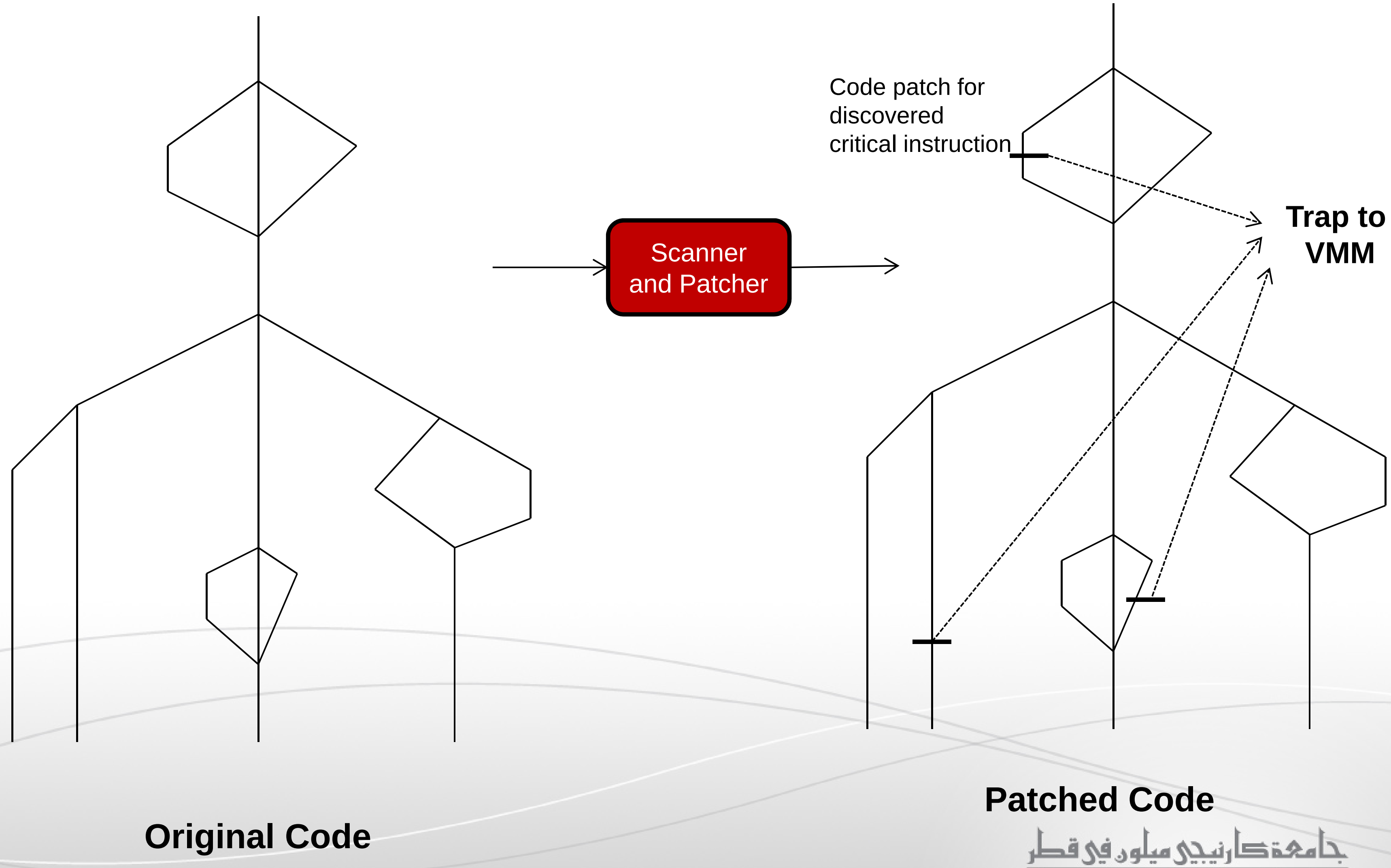
- Critical instructions are sensitive but not privileged– they do not generate traps in user mode
- Intel IA-32 has several critical instructions
- An example is POPF in IA-32 (Pop Stack into Flags Register) which pops the flag registers from a stack held in memory
  - One of the flags is the interrupt-enable flag, which can be modified only in the privileged mode
  - In the user mode, POPF can overwrite all flags except the interrupt-enable flag (for this it acts as no-op)

Can an *efficient VMM* be constructed with the presence of critical instructions?

Handling Critical Instructions

- Critical Instructions are problematic and they inhibit the creation of an efficient VMM
- However, if an ISA is not efficiently virtualizable, this does not mean we cannot create a VMM
- The VMM can scan the guest code before execution, discover all critical instructions, and replace them with traps (system calls) to the VMM
- This replacement process is known as patching
- Even if an ISA contains only ONE critical instruction, patching will be required

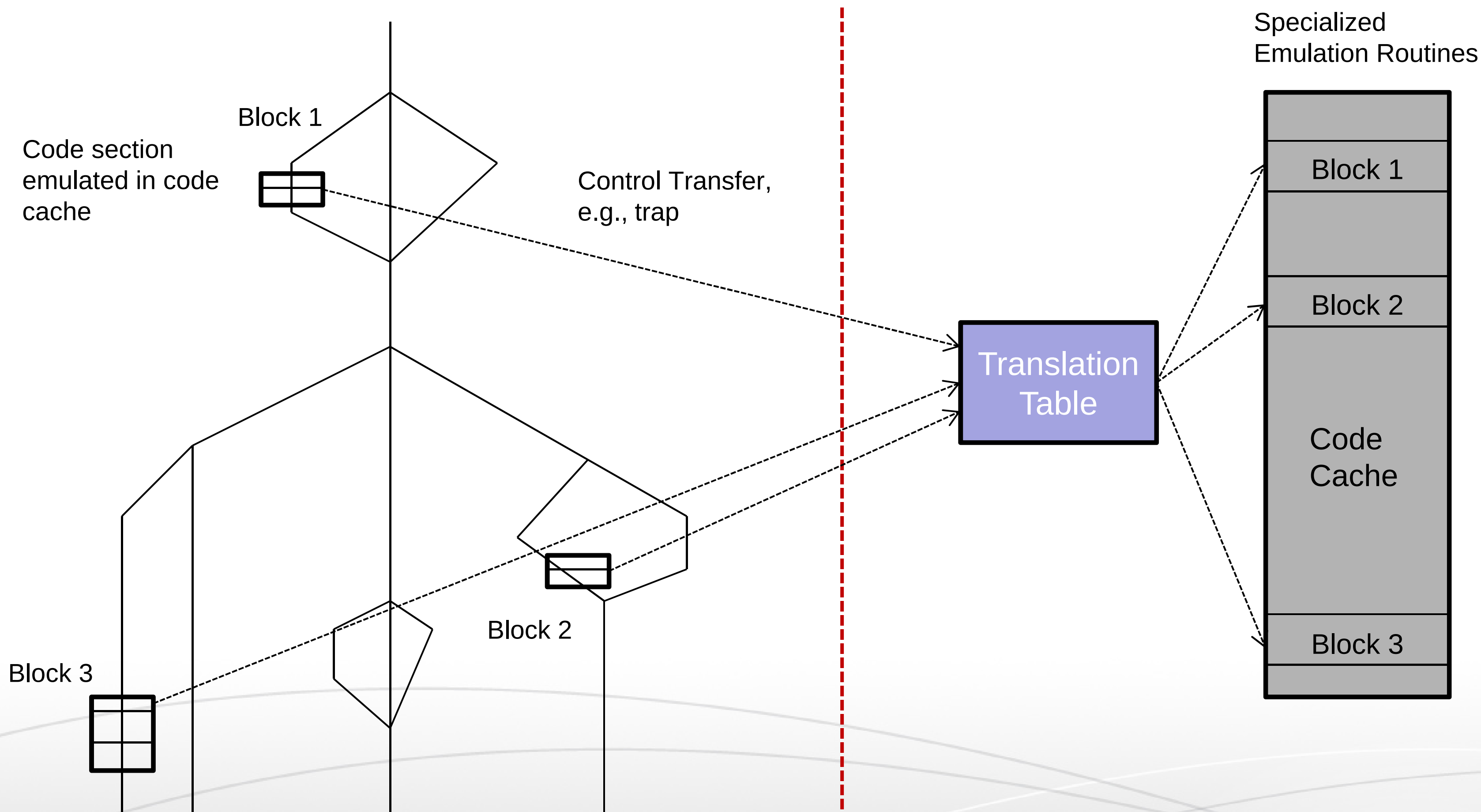
Patching of Critical Instructions



Code Caching

- Some of the critical instructions that trap to the VMM might require interpretation
- Interpretation overhead might slow down the VMM especially if the frequency of critical instructions requiring interpretations increases
- To reduce overhead, interpreted instructions can be cached, using a strategy known as code caching
- Code caching is done on a block of instructions surrounding the critical instruction (larger blocks lend themselves better to optimization)

Caching Interpreted Code

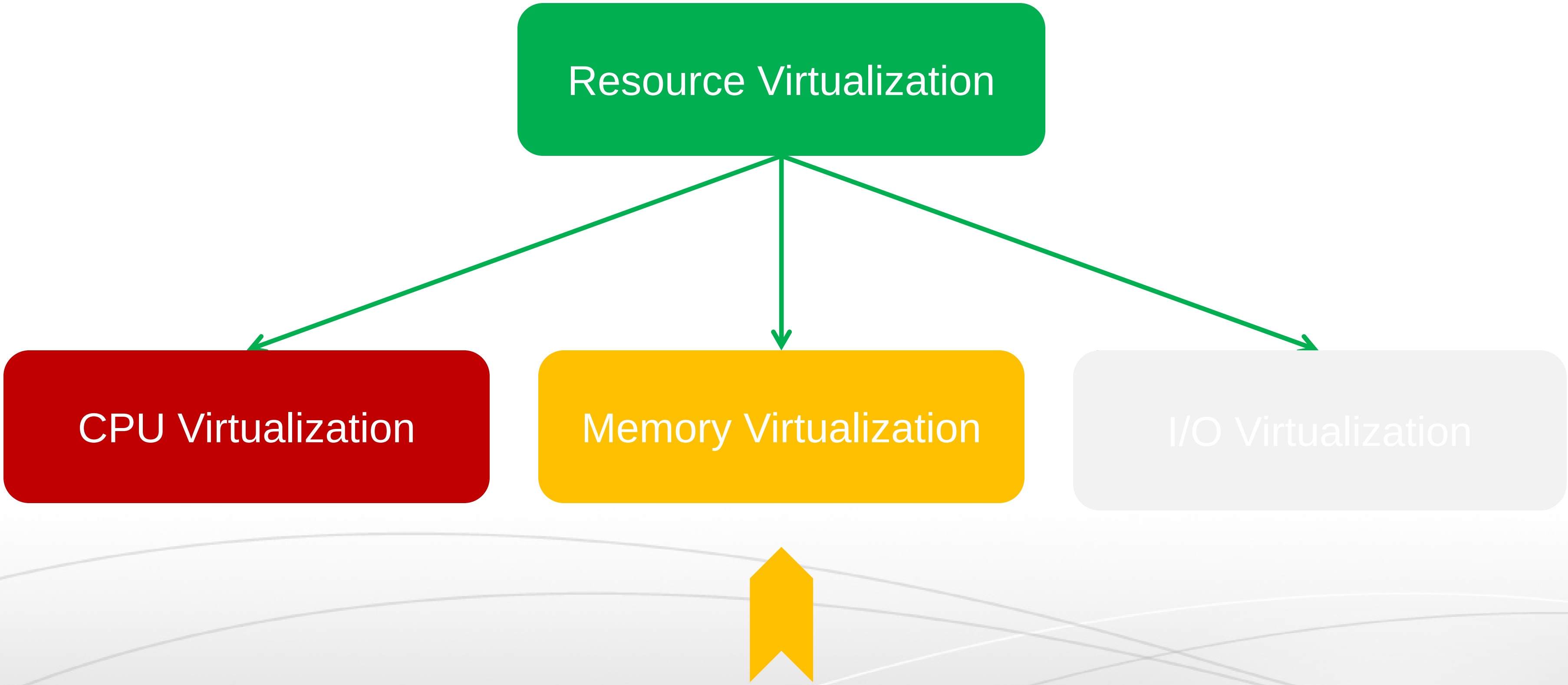


Two critical instructions combined into a single block.

Patched Program

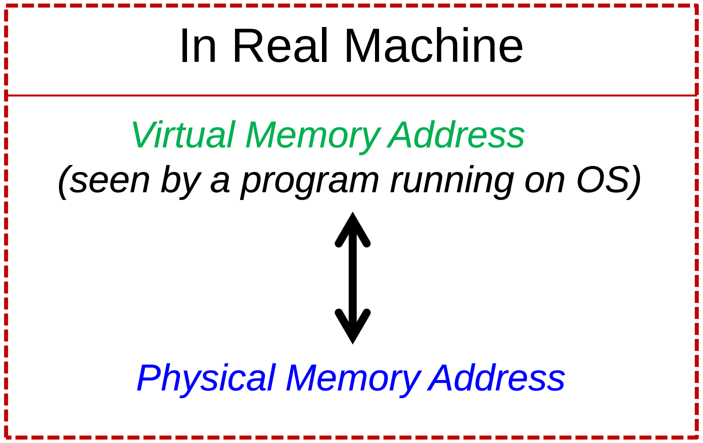
VMM

Resource Virtualization

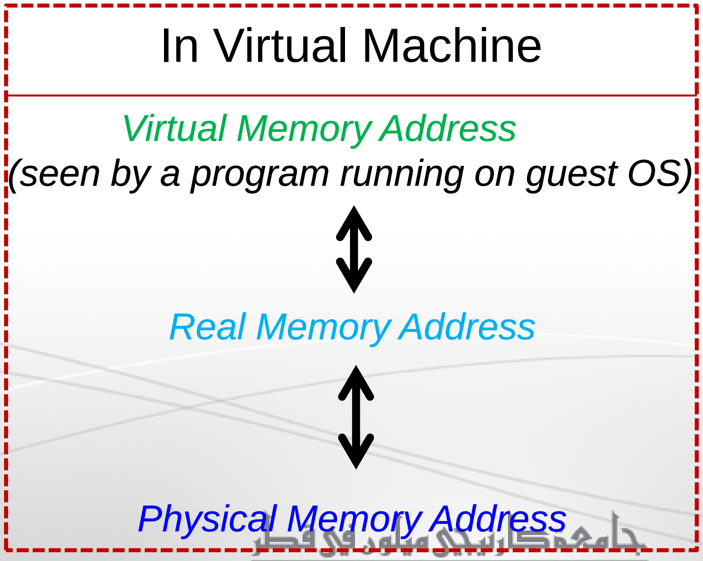


Memory Virtualization

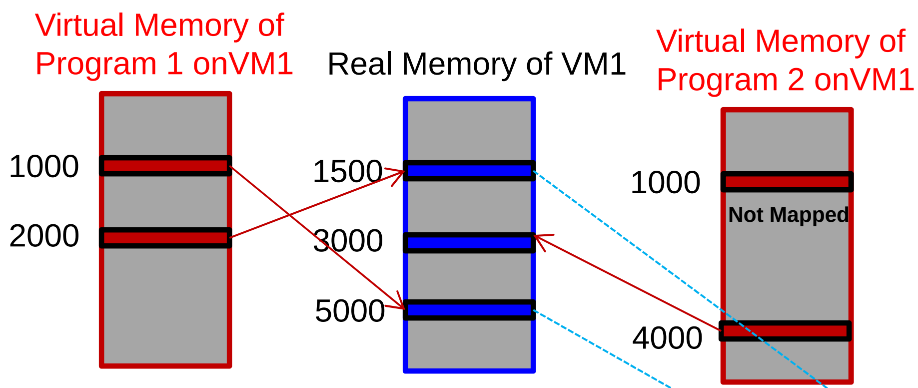
- Virtual memory makes a distinction between the *logical* view of memory as seen by a program and the actual ***hardware memory*** as managed by the OS

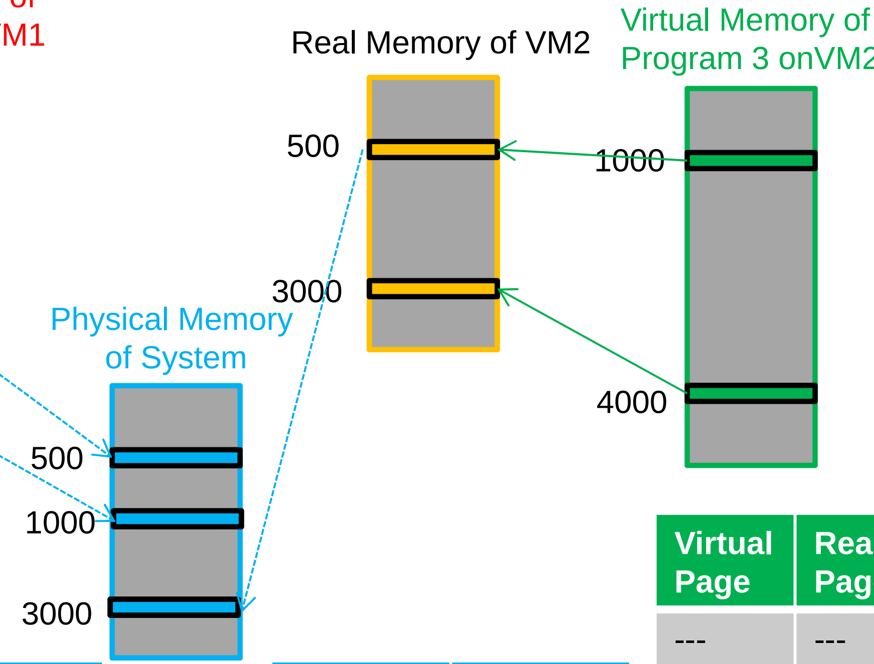


- The virtual memory support in traditional OSs is sufficient for providing guest OSs with the view of having (and managing) their own ***real memories***
  - Such an illusion is created by the underlying VMM



An Example





Virtual Page	Real Page
---	---
1000	5000
---	---
2000	1500
---	---

Page Table for Program 1

Virtual Page	Real Page
---	---
1000	Not mapped
---	---
4000	3000
---	---

Page Table for Program 2

Virtual Page	Real Page
---	---
1000	500
---	---
4000	3000
---	---

Page Table for Program 3

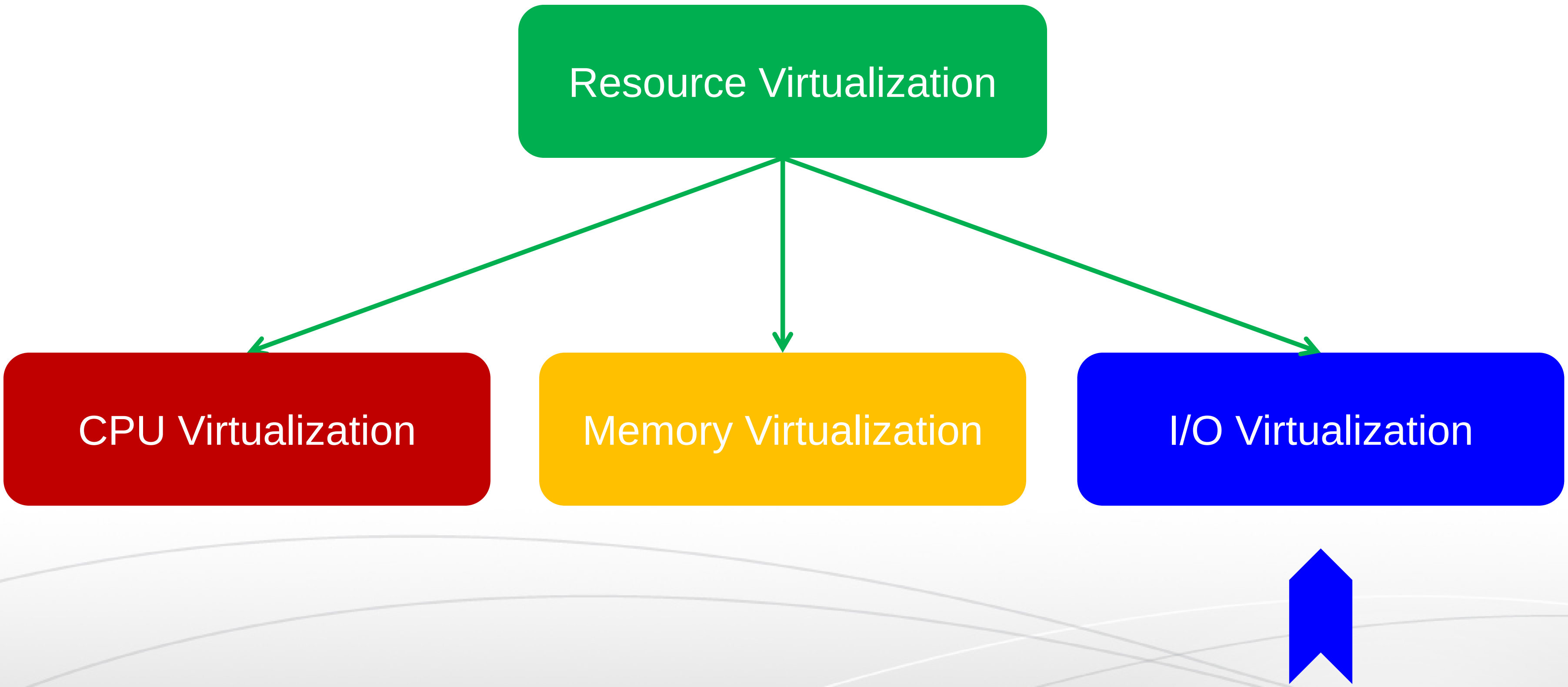
VM1 Real Page	Physical Page
---	---
1500	500
3000	Not mapped
5000	1000
---	---

Real Map Table for VM1 at VMM

VM1 Real Page	Physical Page
---	---
500	3000
---	---
3000	Not mapped
---	---

Real Map Table for VM2 at VMM

Resource Virtualization



I/O Virtualization

- The virtualization strategy for a given I/O device type consists of:

  1. Constructing a virtual version of the device
  2. Virtualizing the I/O activities directed to the device

- A virtual device given to a guest VM is typically (but not necessarily) supported by a similar, underlying physical device

- When a guest VM makes a request to use the virtual device, the request is intercepted by the VMM

- The VMM converts the request to the equivalent request understood by the underlying physical device and sends it out

Virtualizing Devices

- The technique that is used to virtualize an I/O device depends on whether the device is shared and, if so, the ways in which it can be shared

- The common categories of devices are:
  - Dedicated devices
  - Partitioned devices
  - Shared devices
  - Spooled devices

Dedicated Devices

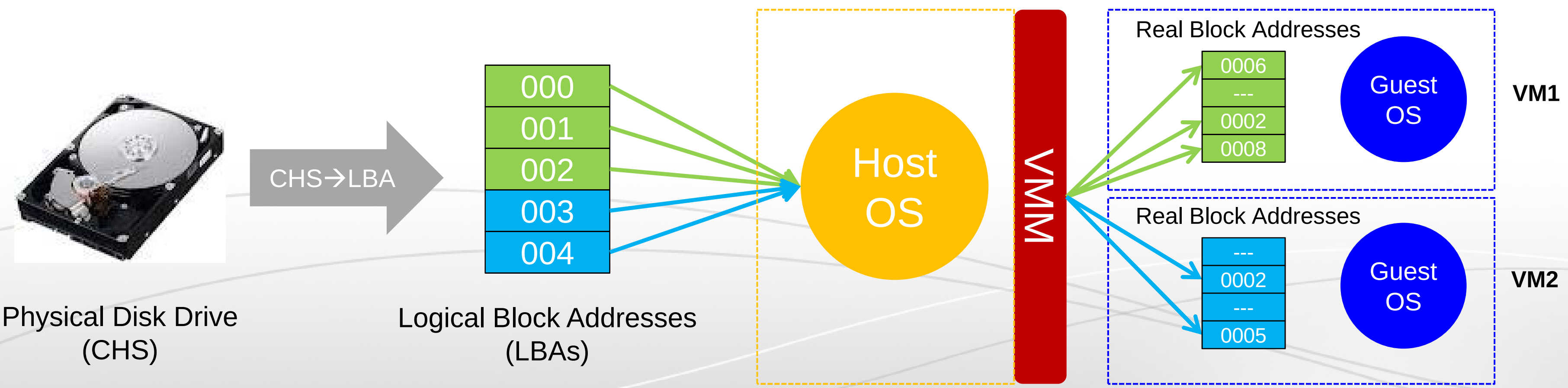
- Some I/O devices must be dedicated to a particular guest VM or at least switched from one guest to another on a very long time scale
- Examples of dedicated devices are: the display, mouse, and speakers of a VM user
- A dedicated device does not necessarily have to be virtualized
- Requests to and from a dedicated device in a VM can theoretically bypass the VMM
- However, in practice these requests go through the VMM because the guest OS runs in a non-privileged user mode

Partitioned Devices

- For some devices it is convenient to partition the available resources among VMs
- For example, a disk can be partitioned into several smaller virtual disks that are then made available to VMs as dedicated devices
- A location on a magnetic disk is defined in terms of cylinders, heads, and sectors (CHS)
- The physical properties of the disk are virtualized by the disk firmware
- The disk firmware transforms the CHS addresses into consecutively numbered logical blocks for use by host and guest OSs

Disk Virtualization

- To emulate an I/O request for a virtual disk:
  - The VMM uses a *map* to translate the virtual parameters into real parameters
  - The VMM then reissues the request to the disk controller



Shared Devices

- Some devices, such as a network adapter, can be shared among a number of guest VMs at a fine time granularity
- For example, every VM can have its own virtual network address maintained by the VMM
- A request by a VM to use the network is translated by the VMM to a request on a physical network port
  - To make this happen, the VMM uses its own physical network address and a virtual device driver
- Similarly, incoming requests through various ports are translated into requests for virtual network addresses associated with different VMs

Network Virtualization- Scenario I

- In this example, we assume that the virtual network interface card (NIC) is of the same type as the physical NIC in the host system

User on VM1

User sends message to external machine (e.g., using send())

→

OS on VM1

OS converts into I/O instructions for virtual NIC, (e.g., OUTS 0xf0…)

→

VMM

VMM sends packet on virtual bridge to device driver of physical NIC (e.g., OUTS 0x280, …)

→

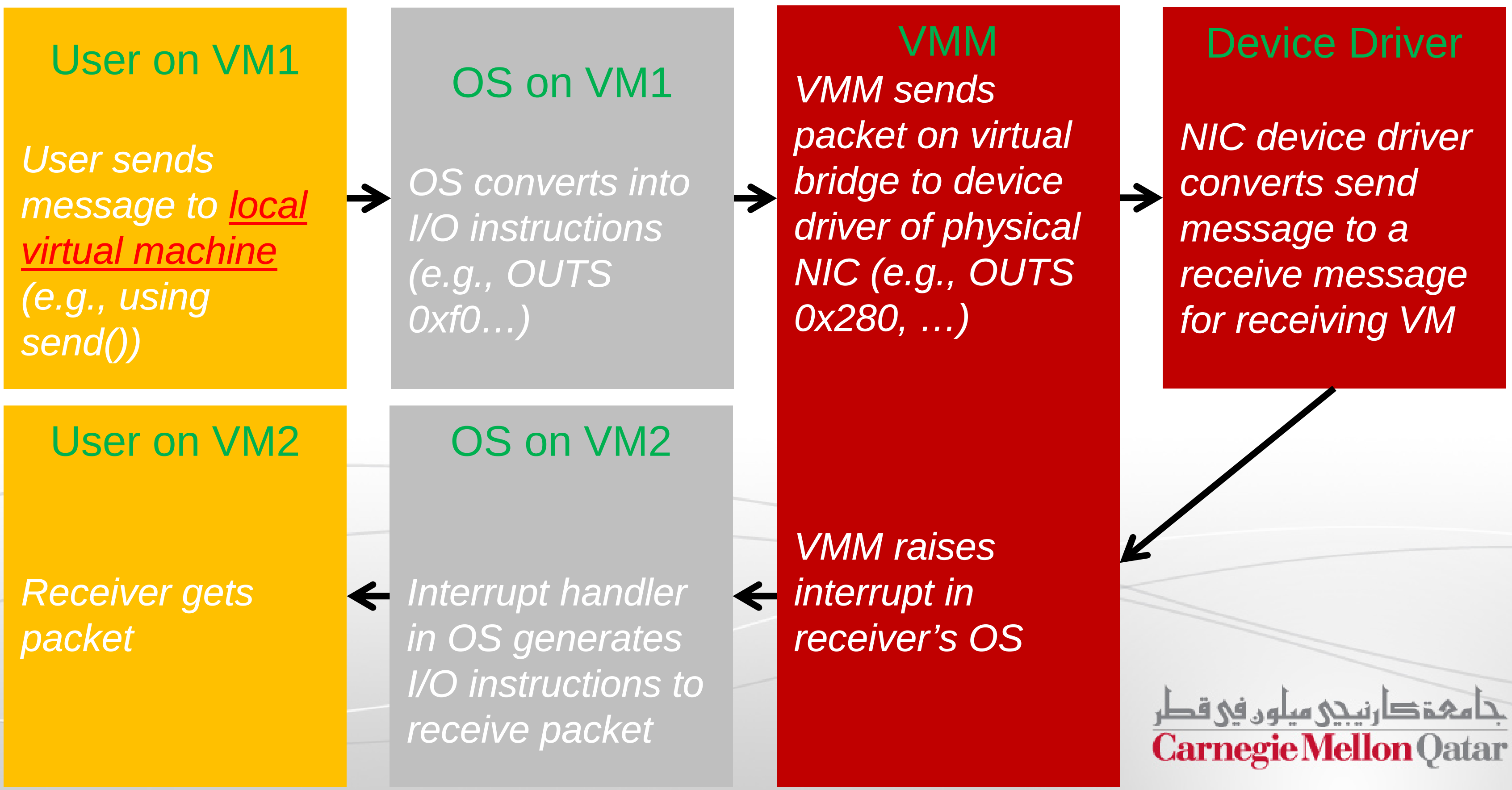
Device Driver

NIC device driver launches packet on network using wire signals

→ ***To Network***

Network Virtualization- Scenario II

- In this scenario, we assume that the desired communication is between two virtual machines on the same platform



جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

Spooled Devices

- A spooled device, such as a printer, is shared, but at a much higher granularity than a device such as a network adapter
- Virtualization of spooled devices can be performed by using a two-level spool table approach:
  - Level 1 is within the guest OS, with one table for each active process
  - Level 2 is within the VMM, with one table for each guest OS
- A request from a guest OS to print a spool buffer is intercepted by the VMM, which copies the buffer into one of its own spool buffers
- This allows the VMM to schedule requests from different guest OSs on the same printer

Thank You!