Cloud Computing
CS 15-319

Virtualization Case Studies : Xen and VMware

Lecture 20

Majd F. Sakr, Mohammad Hammoud and
Suhail Rehman

جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

Today…

- Last session
  - Resource Virtualization

- Today's session
  - Virtualization Case Studies

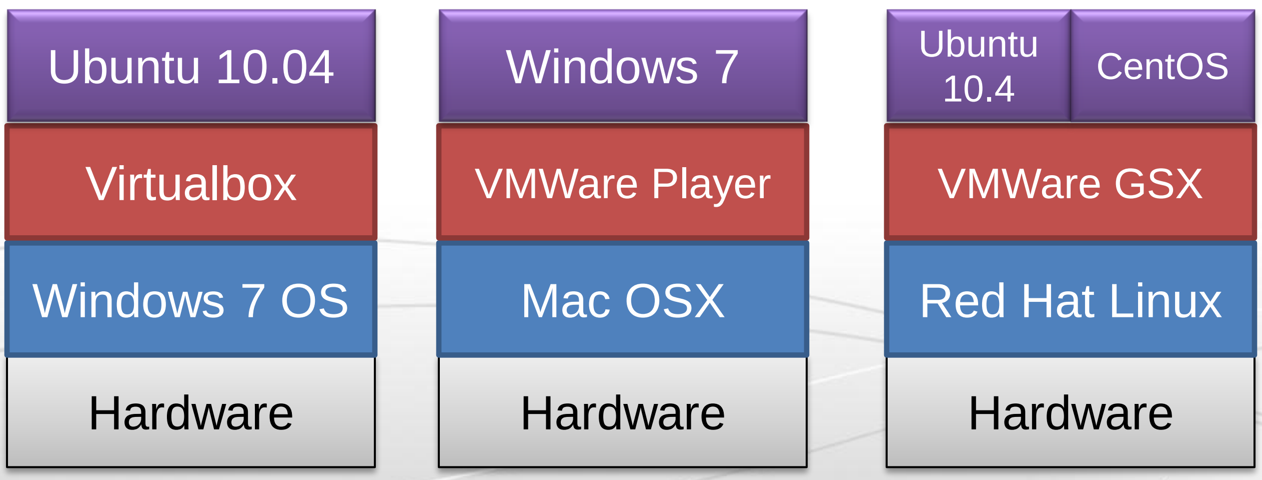
Virtualization Case Studies

- In this lecture, we shall explore popular virtualization environments

- Virtualization environments can be classified as:
  - *Hosted/Dual-Mode Virtual Machine Environments*
    - VirtualBox
    - VMWare GSX/Player/Workstation
  - *System Virtual Machine Environments*
    - Xen
    - VMWare ESX, ESXi

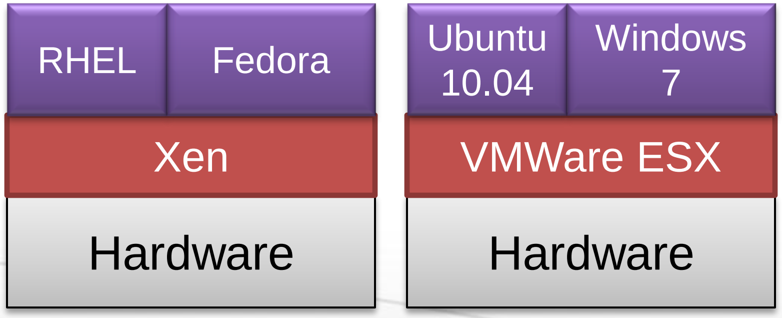
Hosted Virtual Machines

- Hosted Virtual Machines
  - The virtualizing software is installed on top of a traditional operating system.
- Dual-Mode Virtual Machines
  - Portion of the virtualization software runs in the privilege level as the Host operating system
- Examples:
  - VMWare GSX, Player, Workstation



System Virtual Machines

- In a system virtual machine, the virtualizing software (hypervisor) is installed in the place of a traditional operating system.
  - Also known as a “bare-metal” hypervisor
- Guest OSes are installed on top of the hypervisor
  - Eg: Xen, VMWare ESX



Case Study: Xen

Xen

- Xen is a virtual machine monitor that provides services to allow multiple computer operating systems to execute on the same hardware simultaneously.

- Xen was a research project at the University of Cambridge Computer Laboratory in association with Microsoft and Intel research in Cambridge, UK.

Xen approach to Virtualization

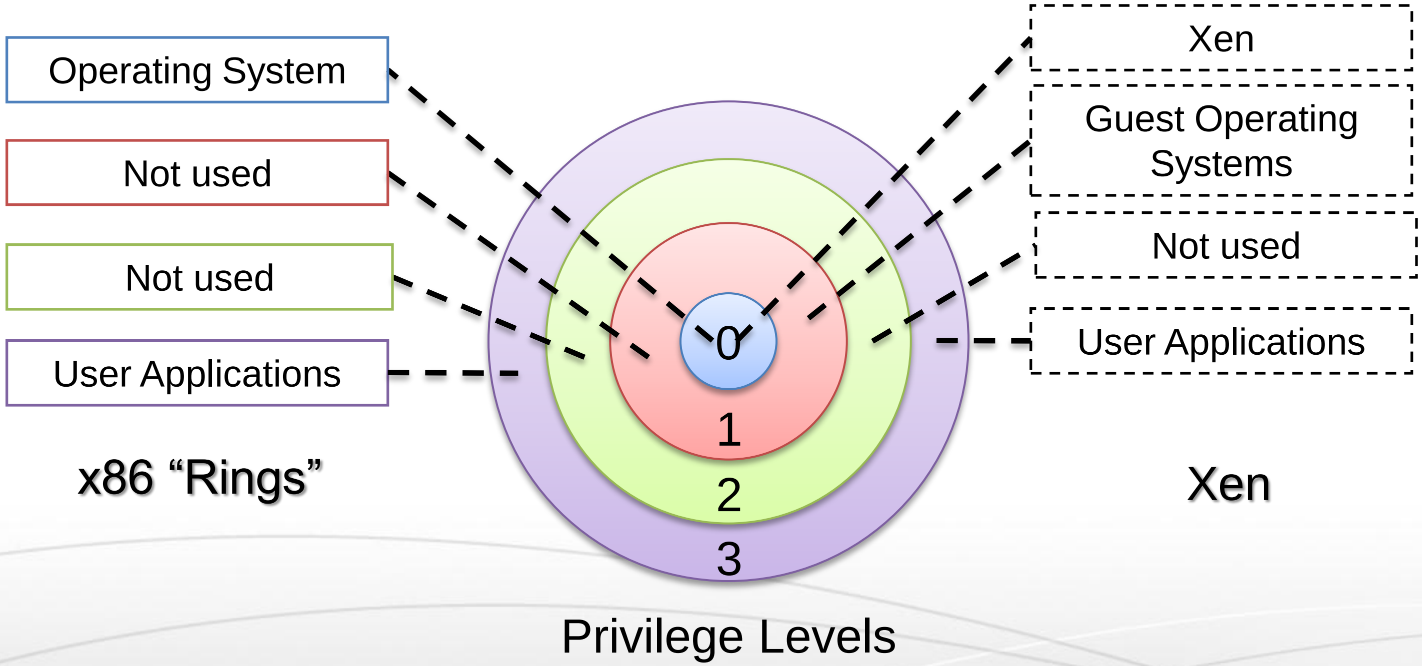
- Classically, Xen uses a *paravirtualization* architecture (PVM)
  - Guest OS'es need to be modified in order to work with Xen
  - Guest OS is aware of virtualization – allows better performance and simplifies the hypervisor design.

- As of version 3.0, Xen also supports Hardware-Assisted Virtualization (HVM)
  - Virtualization support was added to the x86 ISA in new processors (such at Intel VT-x or AMD-V)
  - This enables full virtualization without the need for modifying the Guest OS.

- Design Goal: Keep the hypervisor layer as small and as simple as possible.
  - Management tools, device drivers etc. run in a privileged VM
  - This enhances security, resource isolation

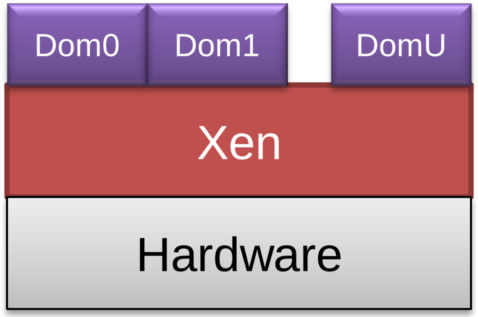
Levels of Protection in x86 and Xen

- Intel's x86 architecture provides levels of privilege for code executing on the processor



Basics of Xen

- Xen hypervisor runs on hardware
  - “Classic” system VM
  - The hypervisor is a thin layer with minimal functionality
- Guest Operating Systems run “on top” of Xen
  - Each virtual machine is known as a **domain**
- Domain 0 is a privileged Virtual Machine
  - Typically some version of NetBSD / Solaris / Linux
  - Privileged access to resources
  - Often hosts device drivers
  - Contains tools to manage other domains on the physical machine



CPU Virtualization in Xen

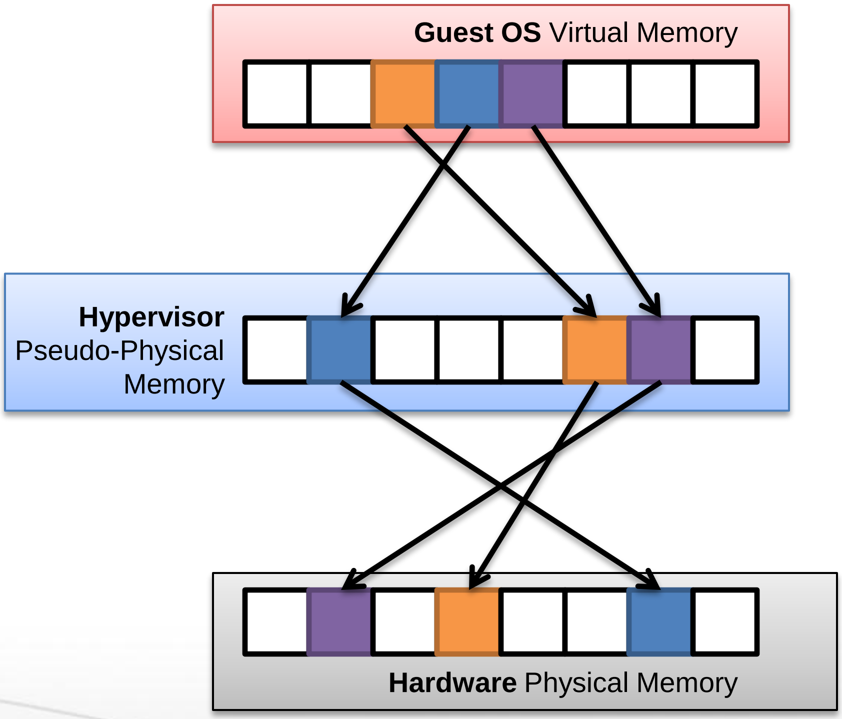
- x86 instruction set was hard to virtualize
  - Certain instructions can be executed only in privileged mode
  - PVM Approach: modify the operating system to replace privileged instructions

- OS code containing privileged instructions are replaced by *hypercalls*
  - The hypercall (similar to a system call) is used to interact with hardware resources
  - The hypervisor will handle a hypercall and will manage the resources between all of the virtual machines

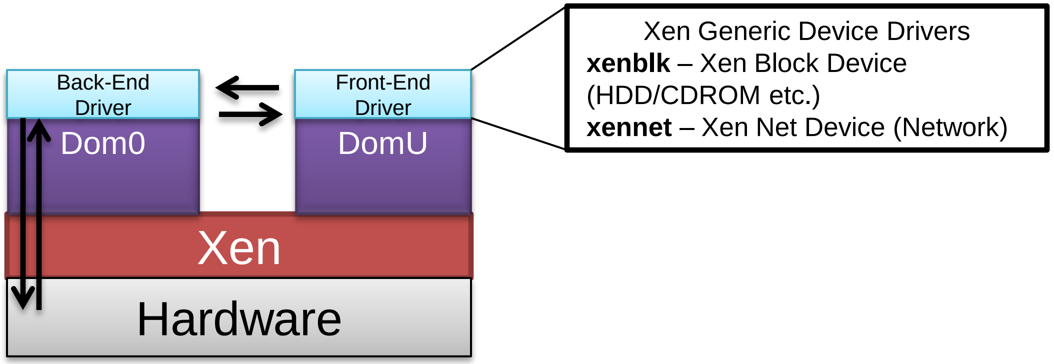
- The modification required to the OS is not very large
  - The OS may contain code that is specific to an particular architecture – which might be privileged instructions. These need to be modified in the PVM approach.
  - Less than 5% of the code to be modified (in case of linux)

Memory Management in Xen

- The Xen hypervisor controls memory management of the system
  - Responsible for physical memory allocation to guest operating systems
- Guest OS has view of "*Real*" (*pseudo-physical*) memory pages
  - Xen hypervisor maps pseudo-physical pages to physical memory on the hardware
- Interaction with Memory depends on OS type
  - For PVM (paravirtualized) – guest OS uses hypercalls to interact with memory
  - For HVM (hardware-assisted) – Xen uses shadow page tables that are accessed through hardware instructions (in Intel VT-x or AMD-V supported CPUs)



Device Management in Xen

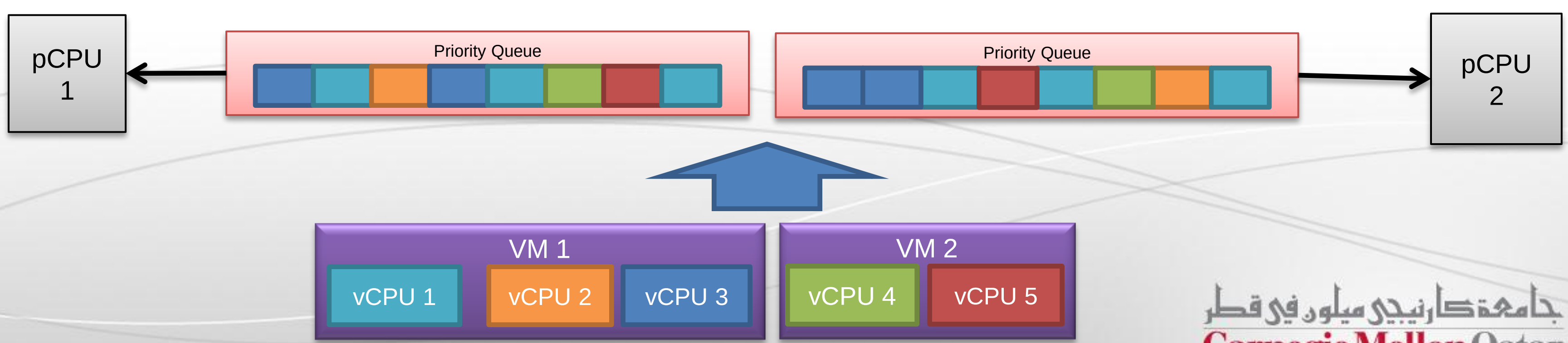


- In a paravirtualized environment, modifying the Guest OS means that existing drivers will not work
  - It is tedious to re-write all device drivers for all the modified guest operating systems
- Instead, Xen uses a *split-driver* model
  - Front-end drivers in DomU, back-end drivers in Dom0
  - The front-end drivers are generic, one type of driver required for each class of device
  - Back-end drivers are device specific and can reuse existing drivers written for Linux

Multiprocessor Virtualization in Xen

- Xen schedules virtual CPUs (vCPUs) against a pool of physical CPUs (pCPUs)
  - Different schedulers available for allocation and scheduling
  - Default scheduler is the Credit Scheduler.

- *Credit Scheduler*
  - Proportional fair-share algorithm
  - Users can assign a weight and a cap for each VM, this influences the scheduling decision and affects the CPU scheduling priority of VMs
  - Assigns credits for all VMs running on the host and debits credits from VMs periodically for each running vCPU.



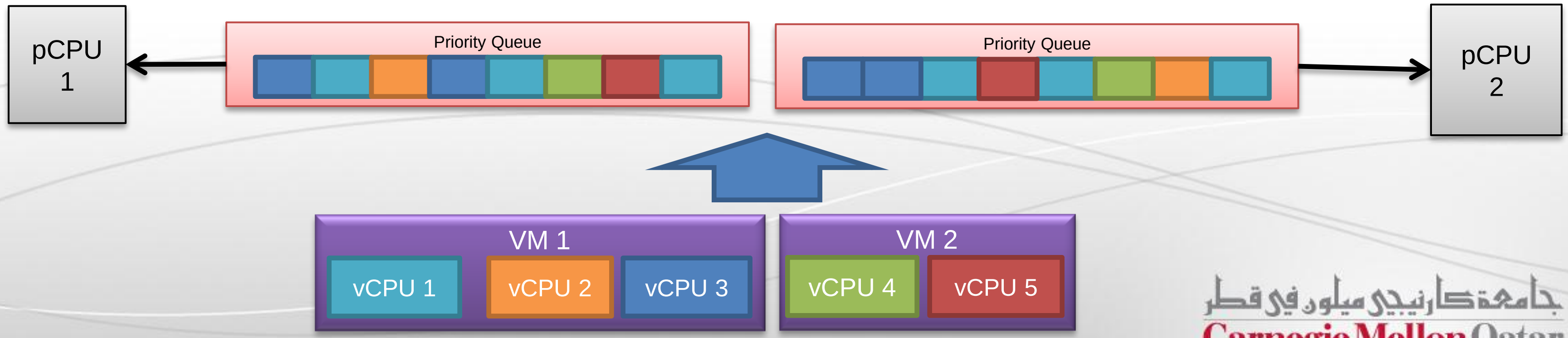
جامعة كارنيجي ميلون في قطر
Carnegie Mellon Qatar

Credit Scheduler (contd.)

- vCPU priority is calculated based on the credits remaining for each vCPU and is refreshed periodically.

- Positive credit VMs are given status of *OVER* and negative credit vCPUs are given status of *UNDER*.

- Scheduling priority determined in the following order:
  - UNDER VM in the run queue of the local CPU
  - UNDER VM in a run queue of another CPU
  - OVER VM in the run queue of the local CPU
  - OVER VM in a run queue of another CPU



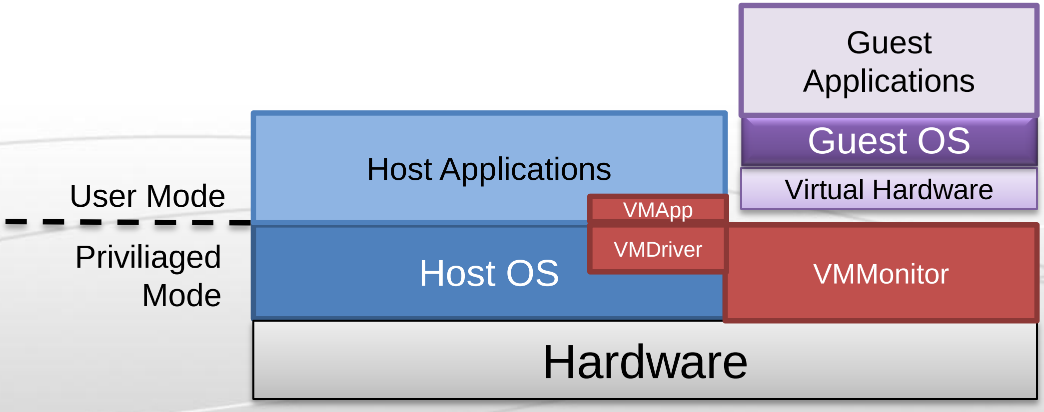
Case Study: VMWare

VMWare Products

- VMWare has two types of Products:
- Hosted VMMs
  - VMWare Workstation, VMWare GSX, VMWare Player, VMWare Fusion etc.
- Bare-Metal Operating Systems
  - VMWare ESX. ESXi

VMWare Dual-Mode Hosted Architecture

- Original VMWare virtual platform approach
  - Host Operating System is loaded first
  - VMWare is installed after the Host OS is installed.
- The virtualization platform contains 3 components:
  - *VMMonitor* (System Level VMM)
    - Runs in privileged mode and has access to hardware resources
  - *VMApp* (User Level VMM)
    - Runs in user mode
    - Makes I/O system calls and has access to the Host OS device drivers
  - *VMDriver* (Pseudo-Driver)
    - Co-ordinates the communication between VMMonitor and VMApp



VMware Bare Metal Architecture

- VMware's ESX/ESXi are system virtual machines
  - Also known as "bare-metal" hypervisors.
- The hypervisor consists of the Virtual Machine Monitor (VMM), a Hardware interface layer and VMKernel.
- A service console allows administrators to manage the hypervisor, virtual machines and hardware on the system.

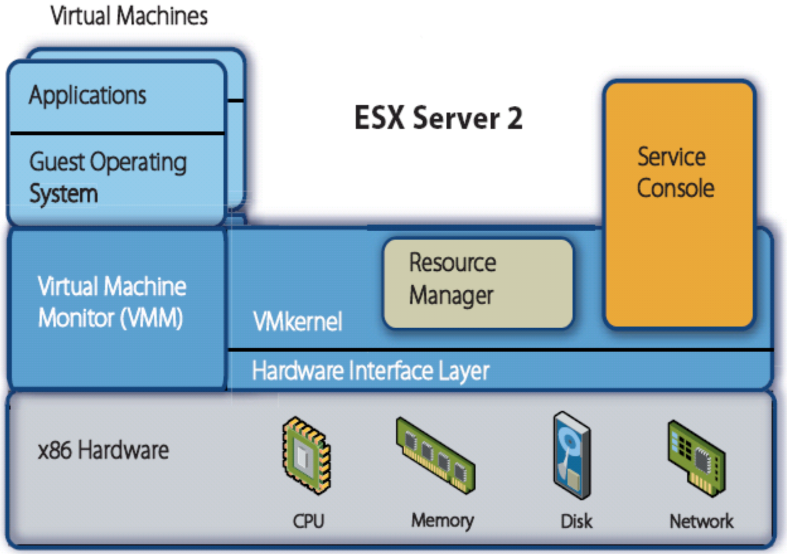
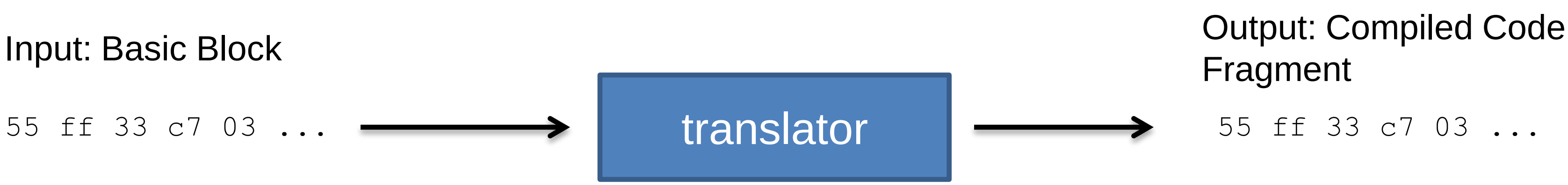


Figure 1: ESX Server architecture

Source: http://www.vmware.com/pdf/esx2_performance_implications.pdf

Processor Virtualization in VMware

- VMWare uses dynamic binary translation
  - Allows code to run on the hardware at near-native speeds.
  - Except for privileged instructions such as traps, interrupts etc.



- Privileged instructions are replaced with a controlled emulation sequence handled by the Hypervisor.
- Further performance improvements
  - Translation of a block of instructions (basic block)
  - Code caching

Multi-Processor Virtualization in VMWare

- VMware features multiple mechanisms to enable efficient scheduling of vCPUs on pCPUs
  - Enables efficient multiplexing of all the vCPUs on a given system, especially if |vCPUs| > |pCPUs|

- *Proportional-share based algorithm*
  - Each VM is allocated *shares*, by default assignment is 1000 shares per vCPU.
  - Each VM also has an associated *reservation* and *limit* which are 0 and unlimited respectively by default.
  - The scheduler calculates if each VM is fully utilizing its resources. VMs which do not fully utilize their allocated resources will be accorded higher priority.
  - This way, the scheduler is also designed for fairness as individual VMs cannot constantly consume the Host CPU, their priority will be dynamically adjusted as they receive more CPU time.

Multi-Processor Virtualization in VMWare (contd.)

- *Co-scheduling*
  - Multi-threaded applications usually require all the vCPUs to be active simultaneously in order to make progress during execution
  - Co-scheduling attempts to schedule all the vCPUs of a VM together, as to maximize the performance of multi-threaded applications within the VM

- *CPU Topology Aware Load-Balancing*
  - Balances the vCPU load among multiple processor systems
  - When a vCPU is migrated to another processor, it does no longer has its working set in cache.
  - The vCPU can continue executing, but will have to fetch its working set from memory and *warm-up* the on-chip cache.
  - Frequent migration of the vCPUs across processors can be detrimental to performance.
  - VMware will migrate vCPUs only if they have not consumed too many CPU resources and warmed up the cache.

Memory Virtualization in VMWare

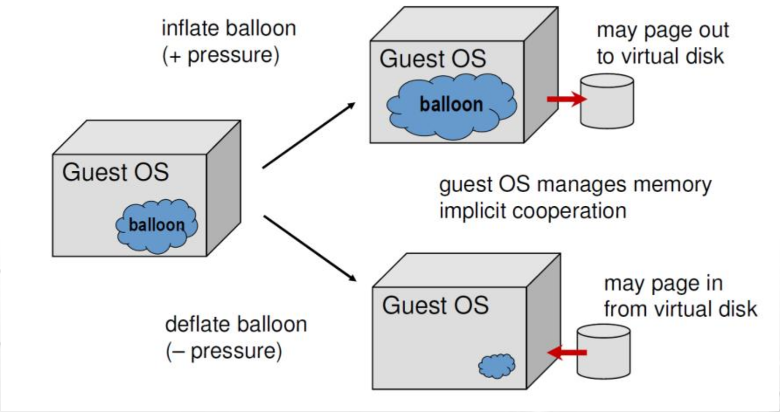
- VMWare virtualizes memory with the following goals:
  - Accurate control over memory resources
  - Efficient over-commitment of memory
  - Exploitation of memory sharing opportunities

- Guest OSes tend to utilize all the memory that is available to them.
  - It's difficult to over-commit memory resources efficiently if the Guest OS does not release memory resources from time to time.

- There are specific mechanisms in VMWare memory management to reclaim memory from guests OSes
  - *Memory Ballooning*
  - *Transparent swapping* of memory pages by the hypervisor
  - *Content-based page sharing* to allow VMs to share identical pages of memory.

Memory Ballooning Technique

- Mechanism to reclaim memory from Guest operating systems
  - A Guest OS *memory balloon driver* is installed on each Guest OS
  - When the hypervisor needs to claim more memory from a guest OS, it signals the driver to *inflate*, it then allocates more memory, forcing the guest OS to send memory pages to disk
  - When the hypervisor decides to give back memory to the Guest, it signals the driver to *deflate* which then de-allocates memory, allowing the Guest to retrieve memory pages from disk

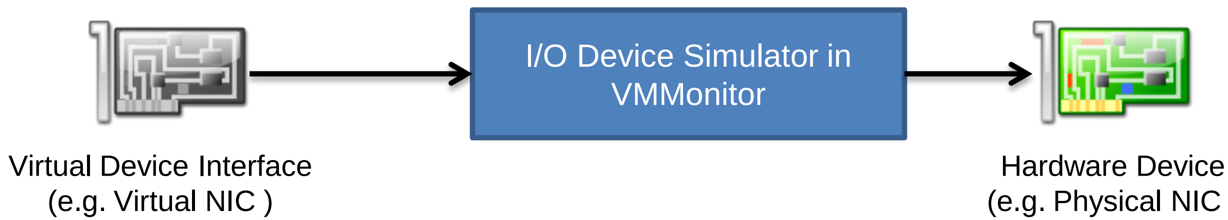


Source: Virtualization 101: http://labs.vmware.com/download/73/

I/O Virtualization in VMWare

- I/O devices are handled through multiple mechanisms in VMWare

- Devices can be supported via the Split Driver Model
  - VMWare uses a *Virtual Device Interface (VDI)*
  - The hypervisor can either directly access the Hardware through privileged mode



  - In addition, the hypervisor can redirect the I/O request through the native OS drivers, this allows the hypervisor to support more devices (but with reduced performance)

Comparison of Xen and VMWare

	Xen	VMware
Architecture	Bare-metal	Hosted (GSX/Workstation/Player) Bare-metal (ESX/ESXi)
Guest OS Modifications	Required if Host CPU does not support virtualization extensions.	Does not require any Guest OS modification.
CPU Virtualization Technique	Hypercalls – all privileged operations are rewritten in PVM	Binary Translation
CPU Scheduling	Credit Scheduler	Proportional-share
Memory Virtualization	Hypervisor managed page translation	Hypervisor managed page translation
I/O Virtualization	Split-Driver Approach	Split Driver Approach with Direct-Mapped in later versions.