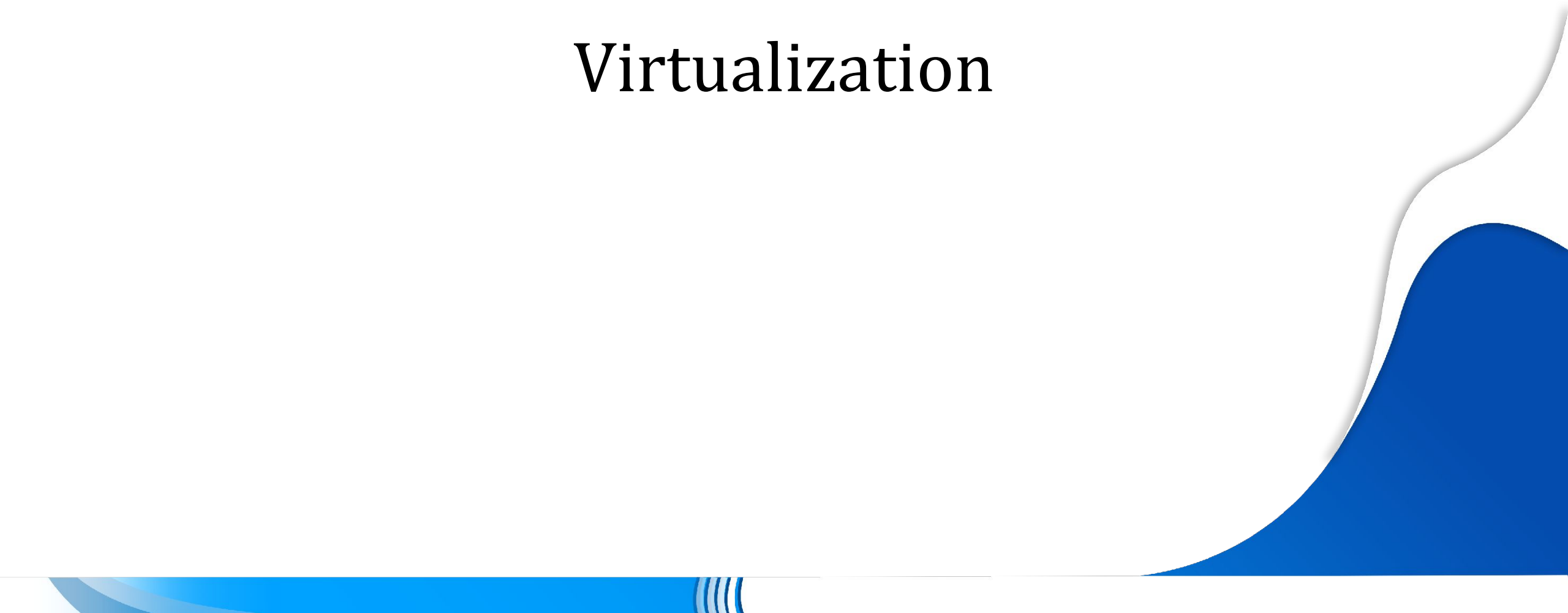




Virtualization

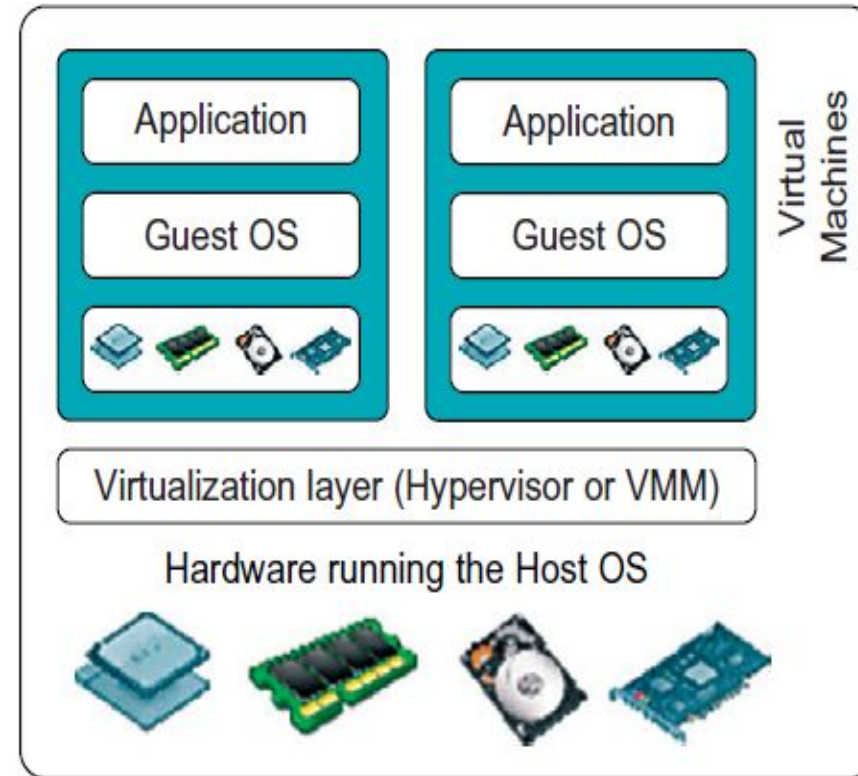


Preface

- In this lecture, we will discuss:
 - Virtualization technology its importance and benefits
 - Different models and key approaches to virtualization
 - CPU, Memory and Device Virtualization.
- 
- 

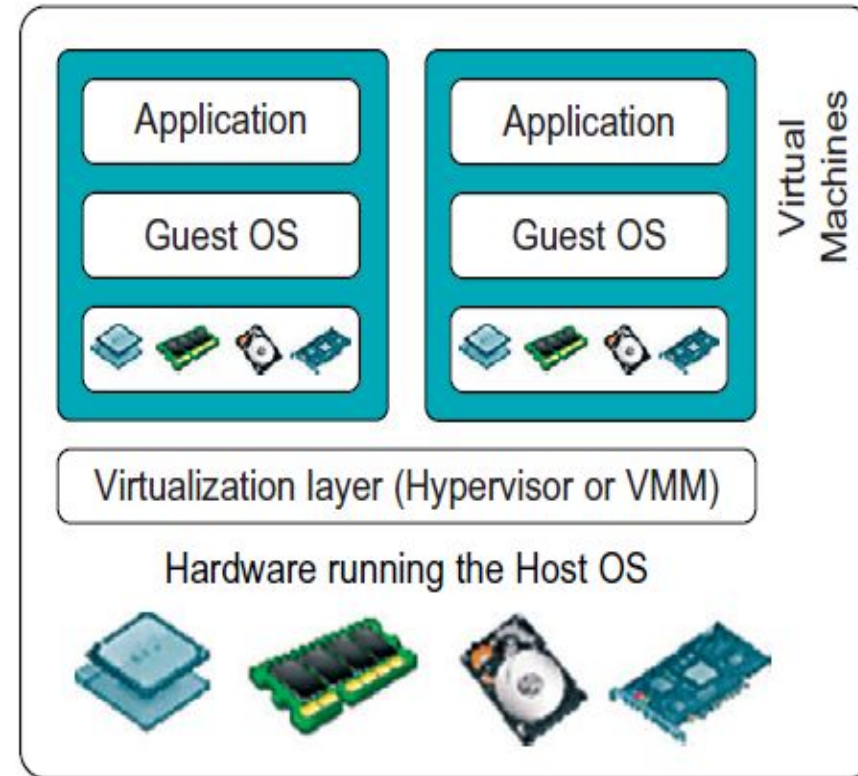
What is Virtualization ?

- Virtualization is originated in the 1960s at IBM.
- Virtualization **allows** concurrent execution of multiple OSs (and their applications) on the same physical machine.
- Virtual resources = **each OS thinks that it “owns” hardware resources**
- Virtual machine (VM) = **OS + applications + virtual resources** (guest domain)
- Virtualization layer = **management of physical hardware** (virtual machine monitor, hypervisor)



Defining Virtualization

- A virtual machine is an **efficient, isolated duplicate of the real machine.**
- Supported by a **virtual machine monitor (VMM)**:
 1. Provides environment essentially identical with the original machine
 2. Programs show at worst only minor decrease in speed.
 3. VMM is in complete control of system resources.
(This means that the virtual machine monitor has full control to make decisions, who accesses which resources and when)



VMM Goals: Fidelity
Performance
Safety & isolation

Benefits of Virtualization

- **Consolidation:**

- It is this ability to run multiple virtual machines, with their operating systems and applications on a single physical platform.
- Decrease cost, improve manageability (with fewer admins and with lower electrical bills)

- **Migration:**

- Migrate the OS in the applications from one physical machine to another physical machine.
- Greater availability of the services, improve reliability

- **Security:** As the OS and the applications are nicely encapsulated in a virtual machine. It becomes more easy to contain any kinds of bugs, or any kinds of malicious behavior, to those resources that are available to the virtual machine only, and not to potentially affect the entire hardware system.

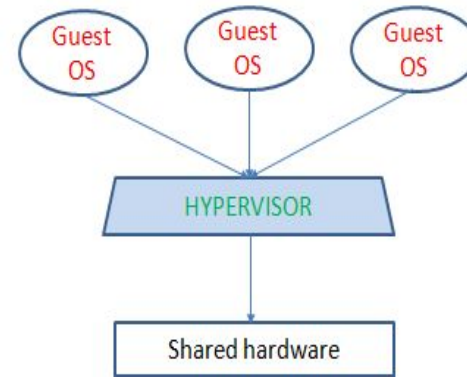
- **Some other benefits:** Debugging, Provide affordable Support for legacy OSs

Virtualization Models

The two popular models for virtualization are called:

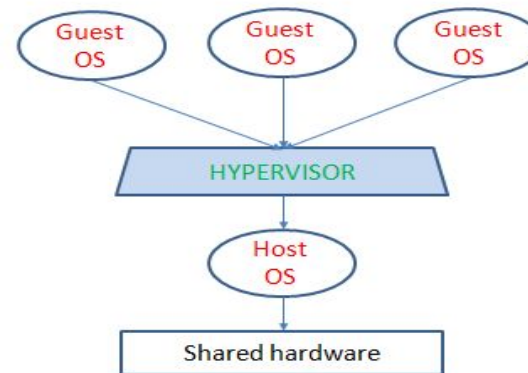
1. Bare-metal hypervisor or Native Hypervisor (Type 1)

Native Hypervisor (bare metal hypervisor)



2. Hosted Hypervisor (Type 2)

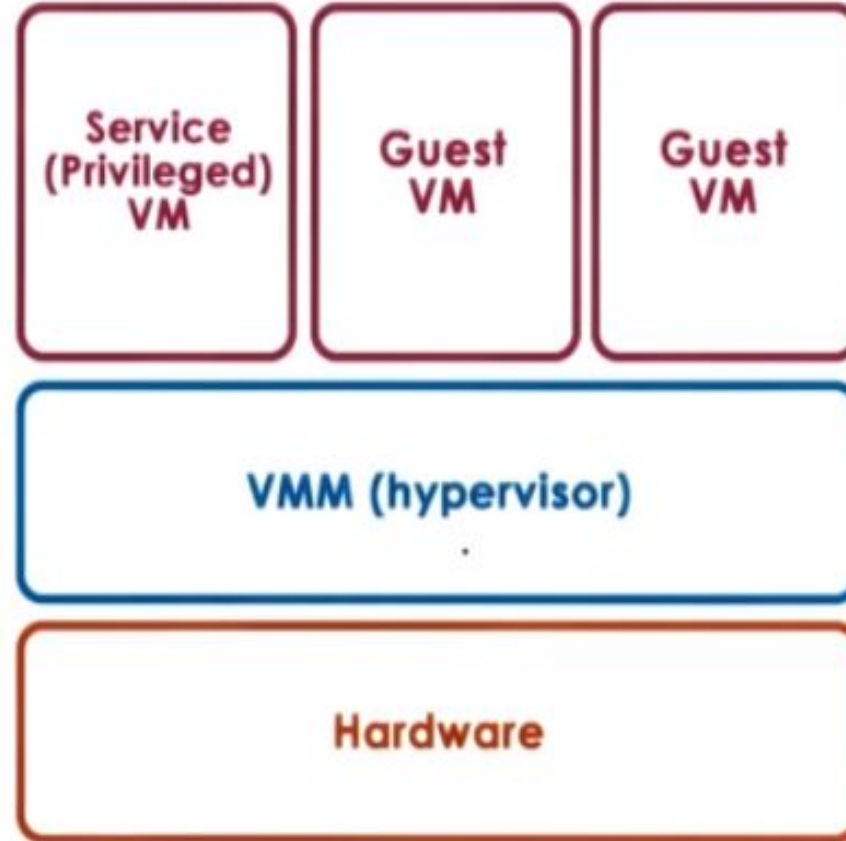
Hosted Hypervisor



Bare-metal virtualization model

Bare-metal hypervisor (Type 1)

- **VMM (hypervisor)** manages all hardware resources and supports execution of entire VMs.
- **Privileged, service VM** to deal with devices (and other configuration and management task)



Bare-metal virtualization model

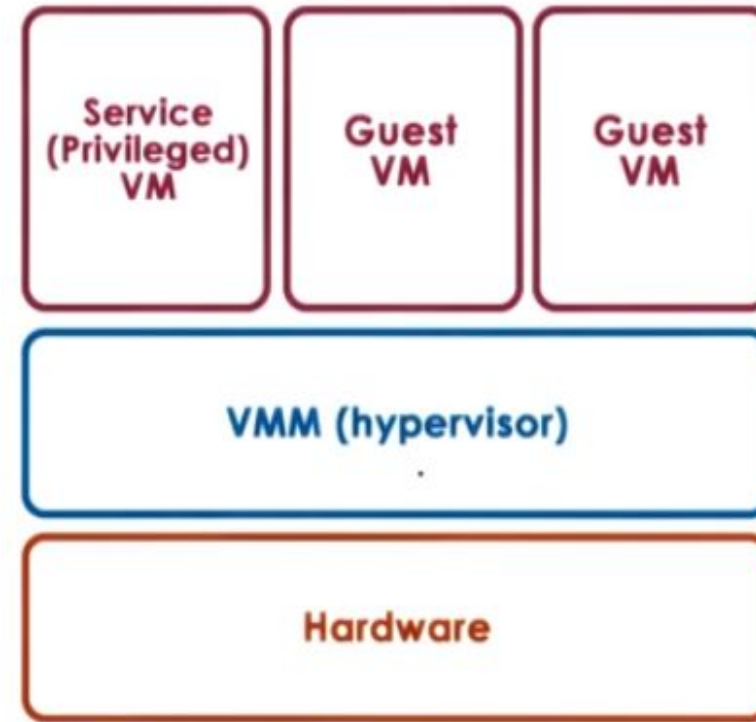
- This model is adapted by the **Xen** virtualization solution (open source or Citrix Xen Server) and also by the VMware's hypervisor, the **ESX hypervisor**.

(i) Xen (Open source or Citrix Xen Server)

- The VMs that are run in the virtualized environment are referred to as domains.
- The privileged domain is called dom-0, and the guest VMs are referred to as dom-U's.
- Xen is the actual hypervisor and all of the drivers are running in the privileged domain, in dom-0.

(ii) ESX (VMware)

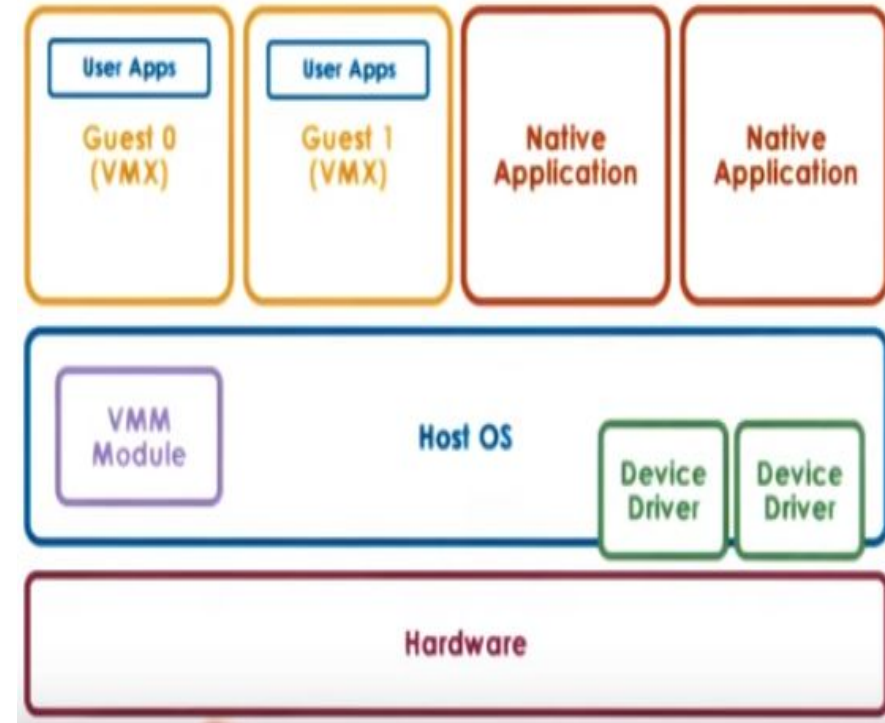
- Given that VMware and its hypervisors were first to market, VMware still owns the largest percentage of virtualized server cores. So these server cores run the ESX hypervisor and also provide the drivers for the different devices. That are going to be part of the hypervisor. To support a third party community of developers VMware exports a number of APIs.



Hosted virtualization model

Hosted Hypervisor (Type 2)

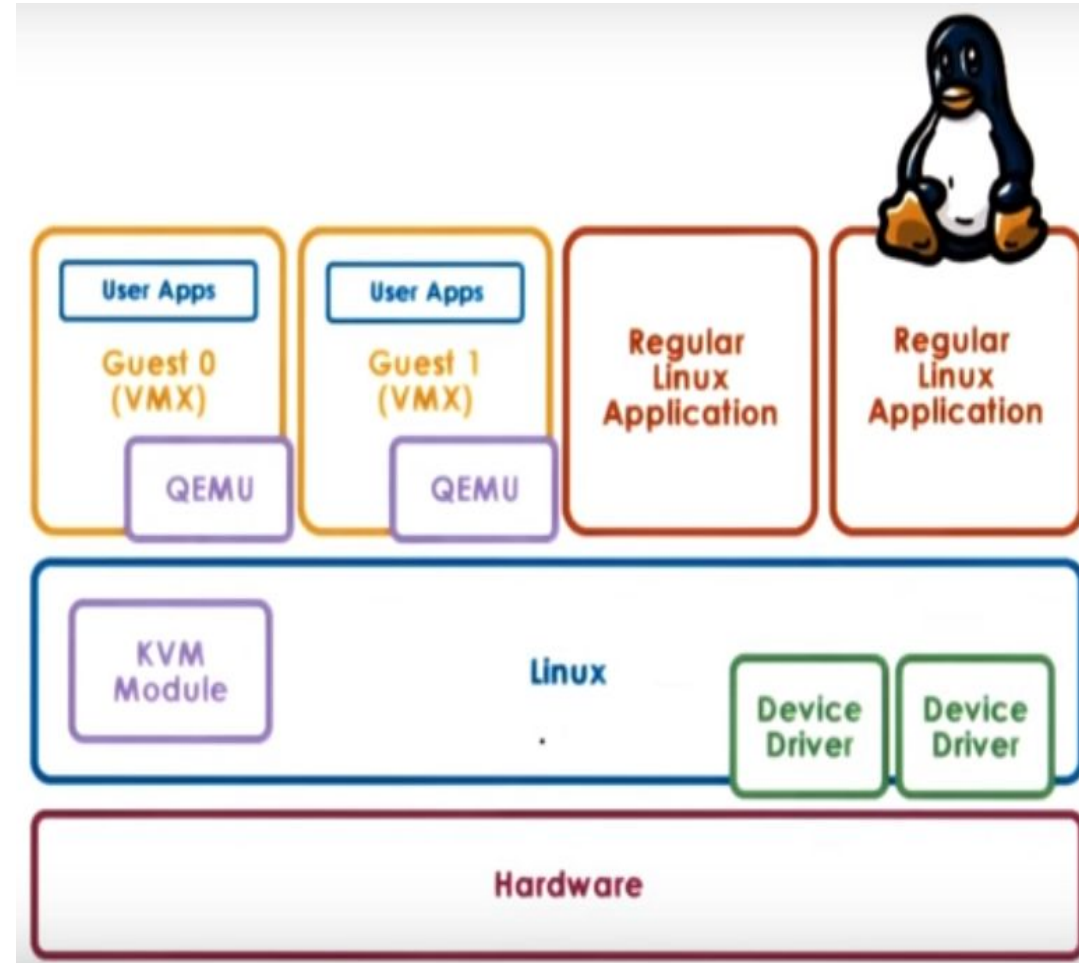
- In this model, at the lowest level, there is a **full fledged host OS that manages all of the hardware resources.**
- The **Host OS integrates a VMM module**, that's responsible for providing the virtual machines with their virtual platform interface and for managing all of the context switching scheduling, etc.



Hosted virtualization model

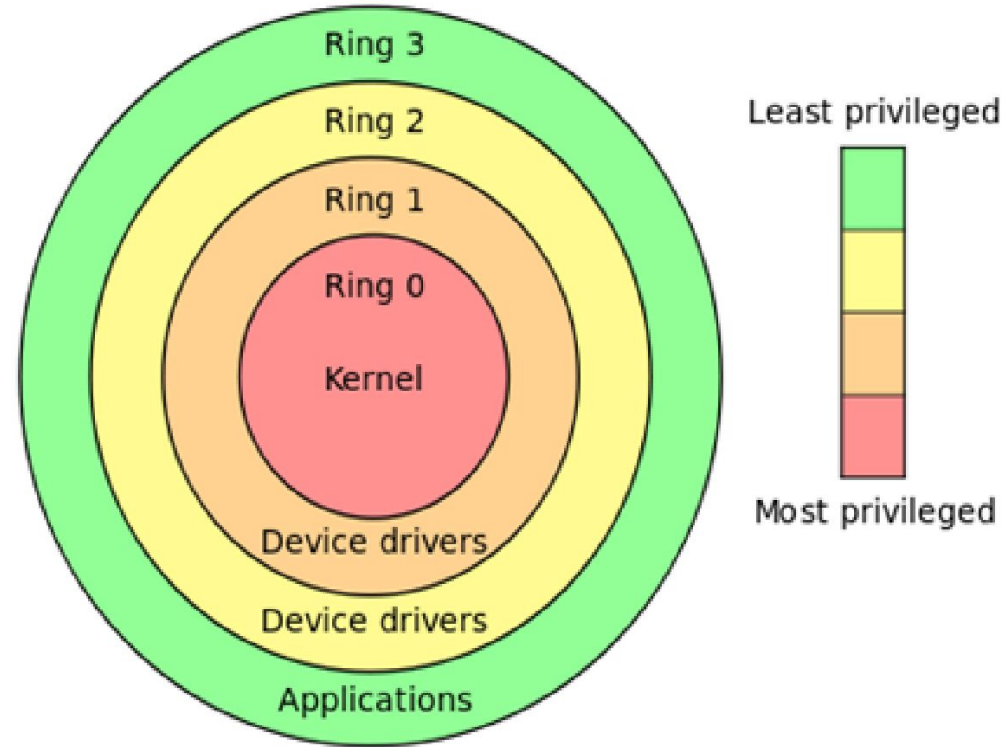
Example:

- **KVM** (Kernel-based VM)
- Based on Linux
- KVM kernel module + hardware emulator (**QEMU**) for hardware virtualization
- Leverages large Linux open-source community



Hardware Protection Levels

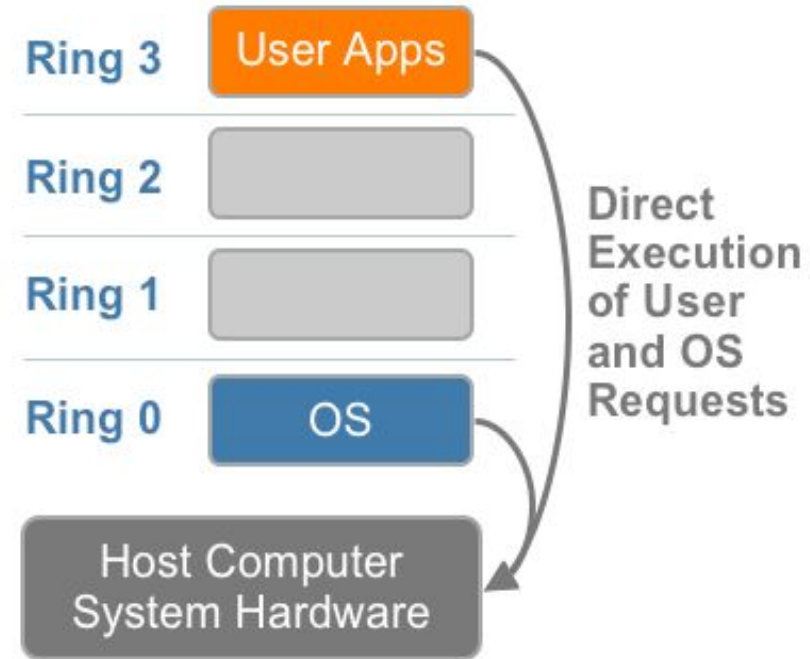
- Commodity hardware actually has more than two protection levels.
- E.g. x86 architecture has four protection levels called **rings**.
- **Ring 3**: In contrast, ring 3 has the least level of privilege, so this is where the applications would reside.
- **Ring 0**: has the highest privilege and can access all of the resources and execute all hardware-supported instructions.



x86 Hardware without Virtualization

- The **x86** architecture offers **four levels** of privilege known as **Ring 0, 1, 2 and 3** to operating systems and applications to manage access to the computer hardware.
- While user level applications typically run in Ring 3, the operating system needs to have direct access to the memory and hardware and must execute its privileged instructions in Ring 0.

Fig - x86 privilege level architecture without virtualization



Processor Virtualization (Trap-and-Emulate)

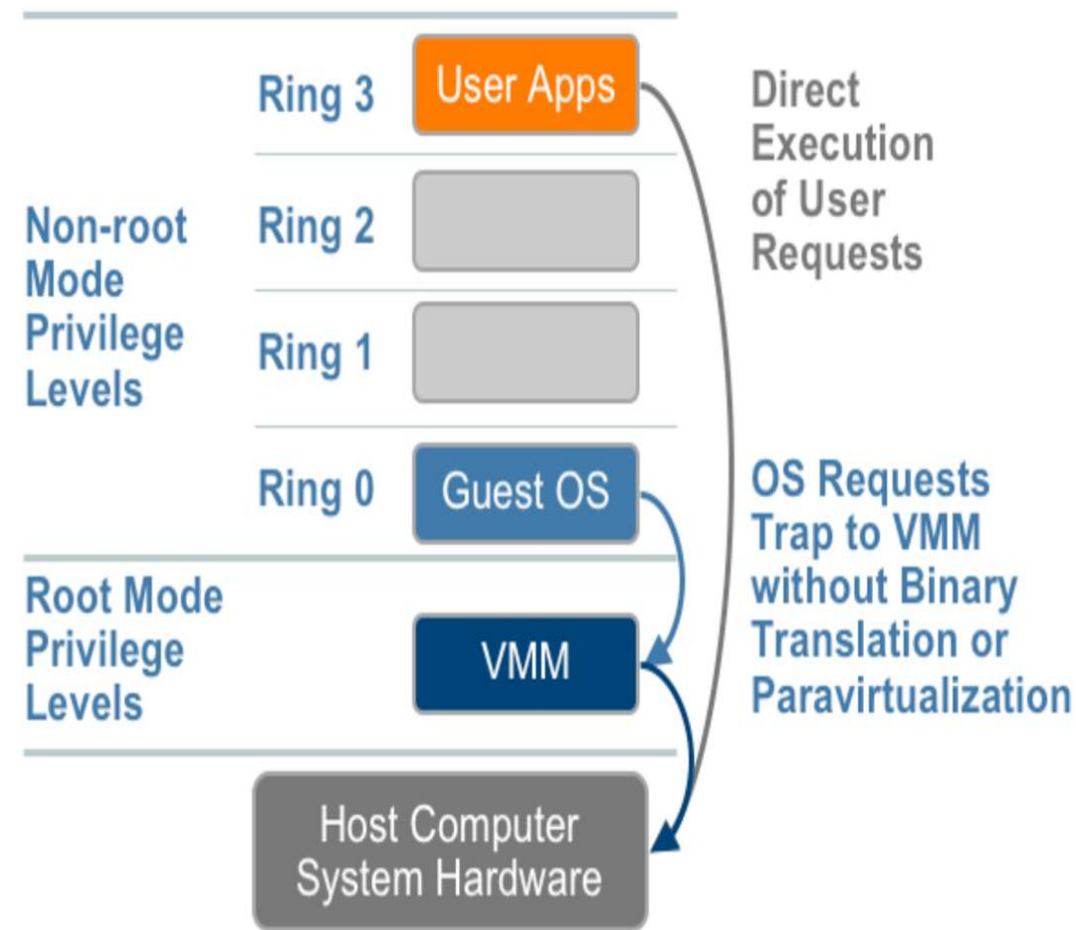
- Guest instructions
 - executed **directly by hardware**
 - for non-privileged operations:
 - hardware speeds=> efficiency
 - for privileged operations: trap to hypervisor
- Hypervisor determines what needs to be done:
 - **If illegal operation:** terminate VM
 - **If legal operation:** emulate the behavior the guest OS was expecting from the hardware

Problems with Trap-and-Emulate

- X85 Pre 2005
- 4 rings, no root/non-root modes yet
- hypervisor in ring 0, guest OS in ring 1
- BUT: 17 privileged instructions do not trap!
Fail silently!

E.g. interrupt enable/disable bit in privileged register; POPF/PUSHF instructions that access it from ring 1 fail silently

- Hypervisor doesn't know, so it doesn't try to change settings
- OS doesn't know, so assumes change was successful.



Binary Translation

- **Main idea:** rewrite the VM binary to never issue those 17 instructions
- Pioneered by Mendel Rosenblum's group at Stanford, commercialized as *Vmware*
- Rosenblum awarded ACM Fellow for “reinventing virtualization”

Binary Translation

Binary translation:

- **Goal:** full virtualization: guest OS not modified
- **Approach:** dynamic binary translation

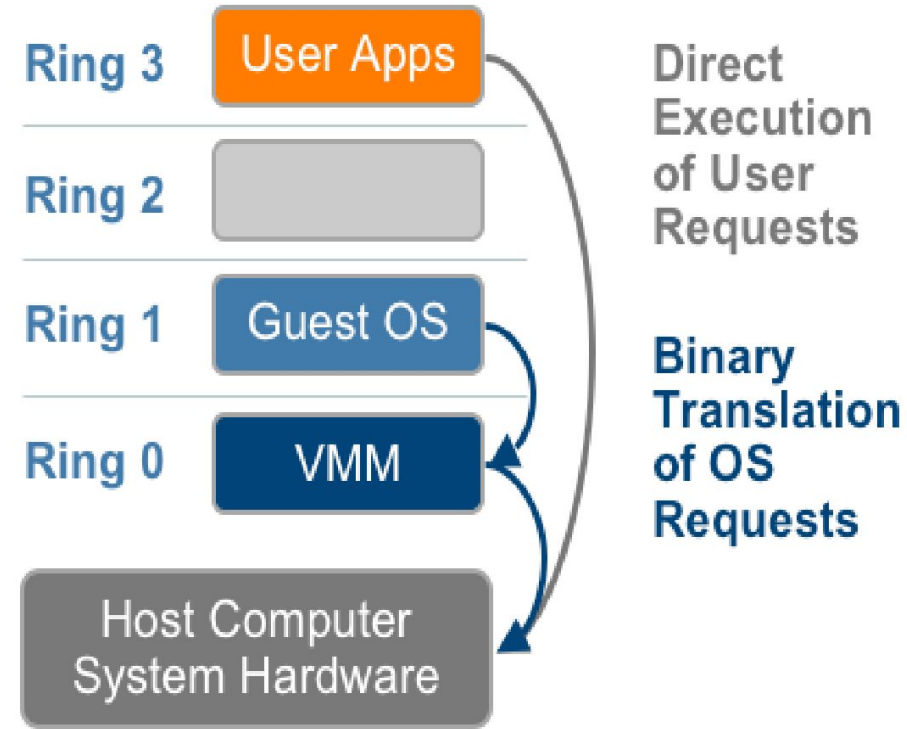
1. Inspect code blocks to be executed
2. If needed, translate to alternate instruction sequence

e.g., to emulate desired behavior, possibly even avoiding trap

3. Otherwise, run at hardware speeds

Cache translated blocks to amortize translation costs

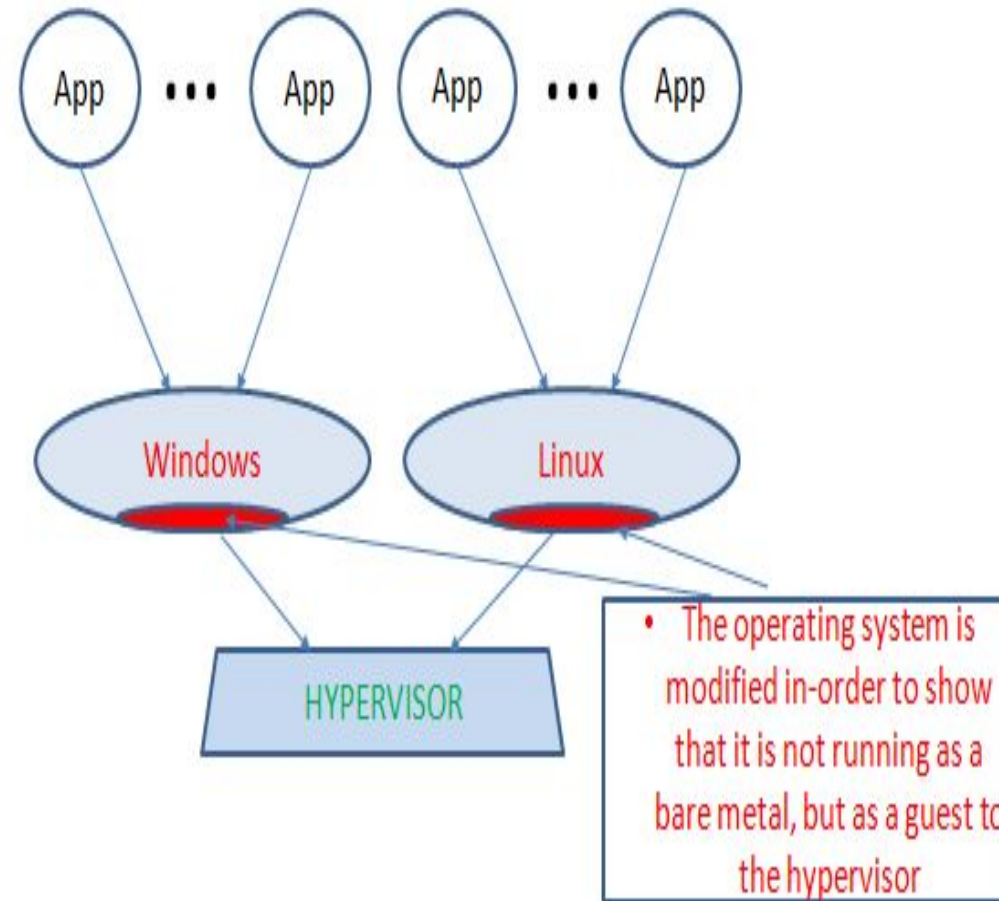
Fig.: Binary translation approach to x86 virtualization



Para virtualization

- **Goal:** performance; give up on unmodified guests
- **Approach:** Para virtualization => modify guest so that
 - It knows it's running virtualized
 - It makes explicit calls to the hypervisor (hypercalls)
- **Hypercall** (~system calls)
 - Package context info
 - Specify desired hypercall
 - Trap to VMM

*e.g. Xen= open source hypervisor
(XenSource -> Citrix)*



Para virtualization: Review !

- What percentage of Guest OS code may need modification with para Virtualization?

- ☐ 10 %
- ☐ 50%
- ☐ 30%
- ☐ 2%

Para virtualization: Review !

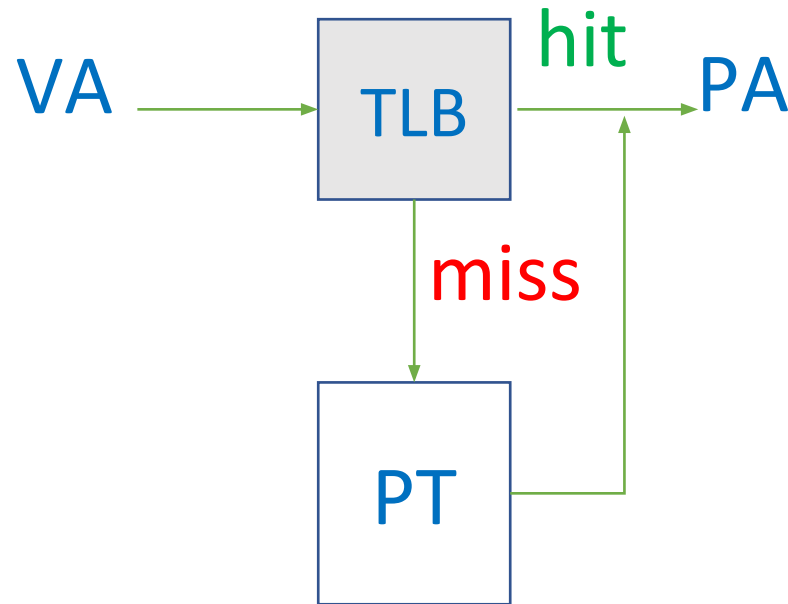
- What percentage of Guest OS code may need modification with para Virtualization?
 - ☐ 10 %
 - ☐ 50%
 - ☐ 30%
 - ☐ 2%
- Answer: less than 2%.
 - This can be shown by a proof-of-construction by **XEN**
 - **Xen** is a para virtualized hypervisor.

Memory Virtualization

- To run multiple virtual machines on a single system, one has to virtualize the **MMU**(memory management unit) to support the guest OS.
- The **VMM** is responsible for mapping guest physical memory to the actual machine memory, and it uses **shadow page tables** to accelerate the mappings.
- The VMM uses **TLB** (translation lookaside buffer) hardware to map the virtual memory directly to the machine memory to avoid the two levels of translation on every access.

Shadow Page table

- VA- Virtual Address
- PA- Physical Address



- CPU uses PT for address translation
- Hardware PT is really the S-PT(Shadow Page Table)

Memory Virtualization

- **Full virtualization**

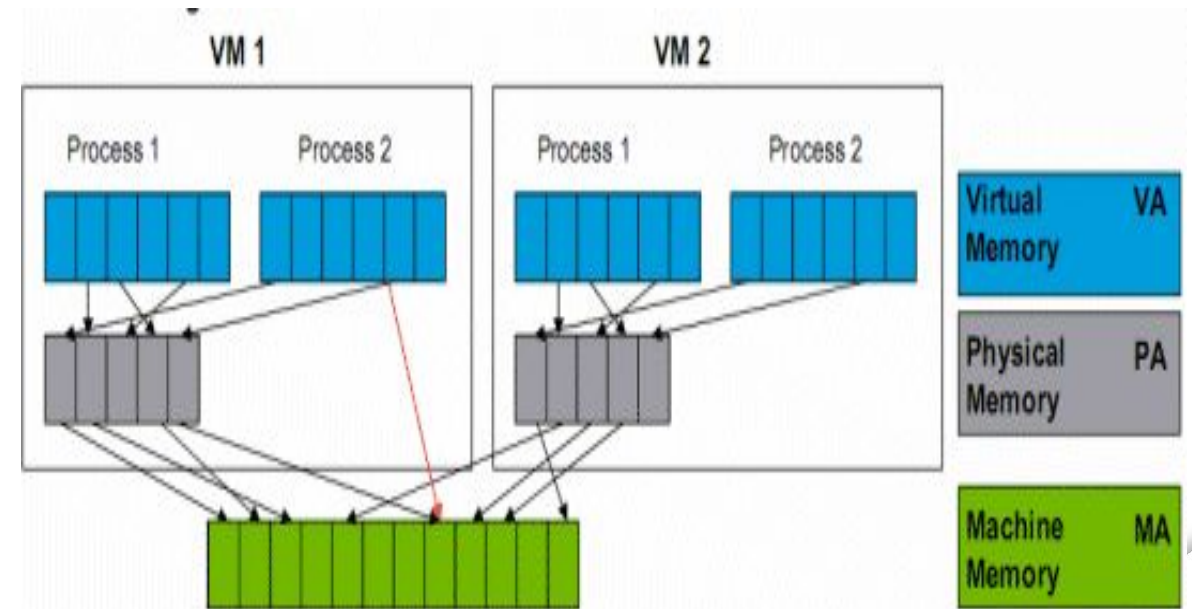
- all guests expect contiguous physical memory, starting at 0
- virtual vs. physical vs. machine addresses and page frame numbers
- Still leverages hardware MMU, TLB

Option 1:

- Guest page table: VM => PA
- Hypervisor: PA=> MA
- Too expensive

Option 2:

- Guest page table: VA => PA
- Hypervisor shadow Page Table: VA=> MA
- Hypervisor maintains consistence
e.g. invalidate on ctx switch, write-protect guest PT to track new mappings

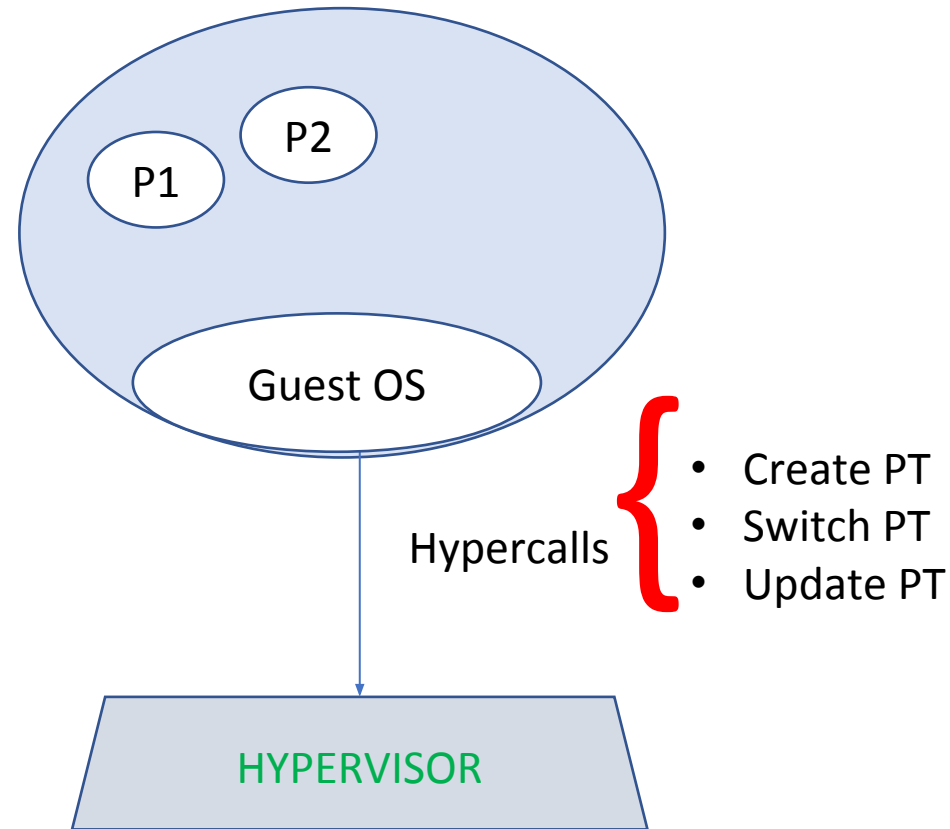


Memory Virtualization

• Para virtualization

- Guest aware of virtualization
- No longer strict requirement on contiguous physical memory starting at 0
- Explicitly registers page tables with hypervisor
- Can “batch” page table updates to reduce VM exists
- Other optimizations

Overheads eliminated or reduced on newer platforms



Device Virtualization

- For CPUs and memory
- Less diversity, ISA (instruction set architectures) level standardization of interface
- For devices
- High diversity
- Lack of standard specification of device interface and behavior
- Three Key models for device virtualization

(i) Passthrough Model

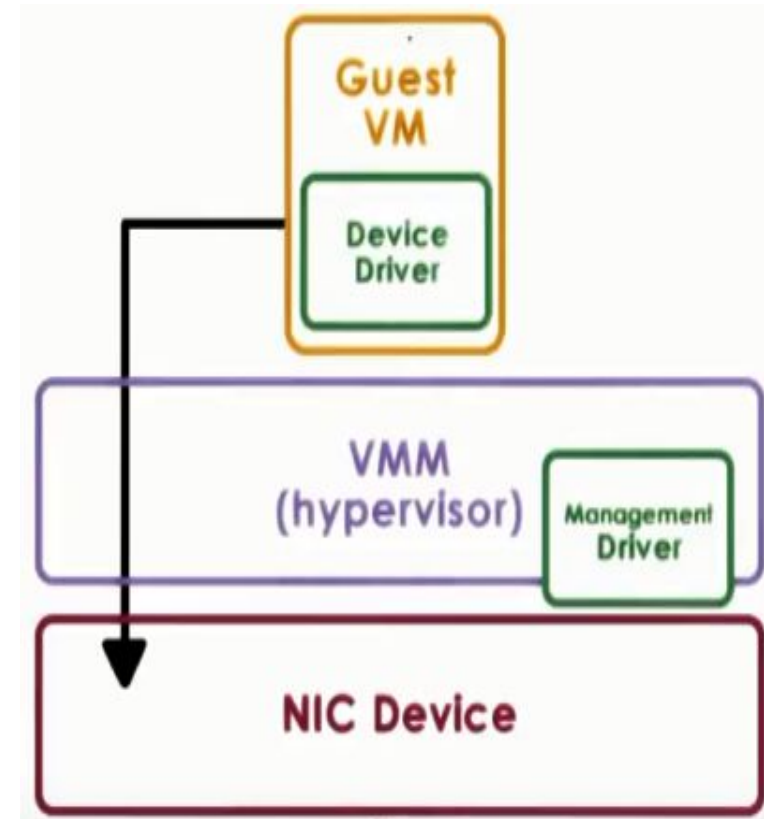
(ii) Hypervisor Direct Model

(iii) Split Device Driver Model

(i) Passthrough Model

Approach: VMM-level driver configures device access permissions

- VM provided with exclusive access to the device
- VM can directly access the device (also called VMM-bypass model)
- Device sharing will become difficult.
- VMM must have exact type of device as what VM expects
- VM migration is tricky



(ii) Hypervisor-Direct Model

Approach:

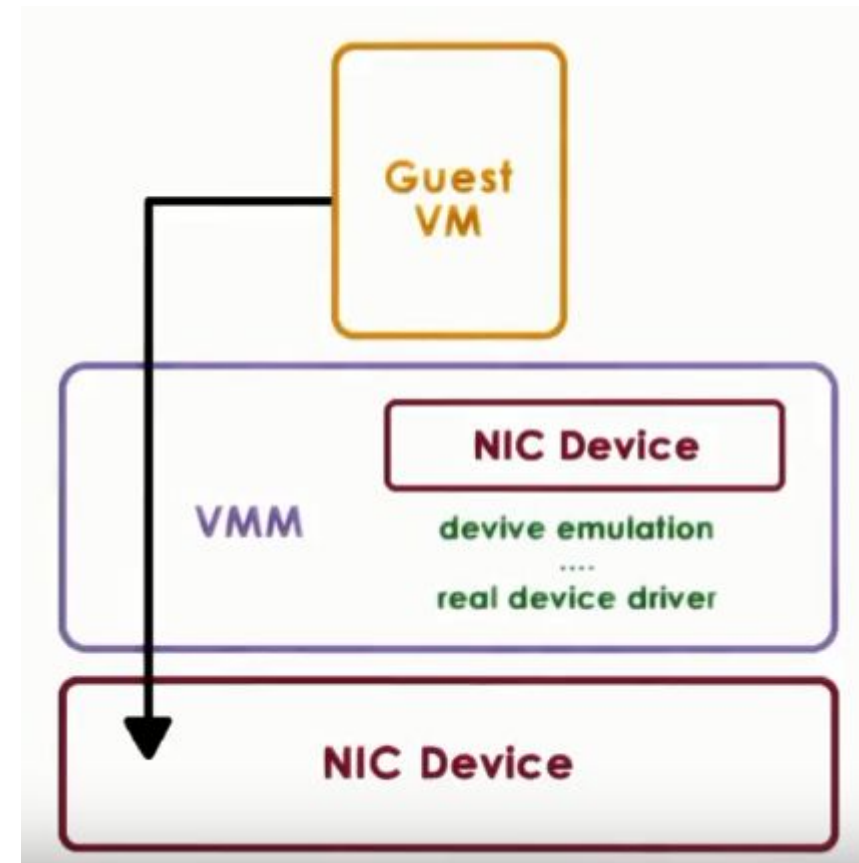
- VMM intercepts all device accesses
- Emulate device operation:
 - translate to generic I/O operation
 - traverse VMM-resident I/O stack
 - invoke VMM-resident driver

Key benefits:

- VM decoupled from physical device
- Sharing, migration, dealing with device specifics

Downside of the model

- Latency of device operations
- Device driver ecosystem complexities in hypervisor



(iii) Split-Device Driver Model

Approach:

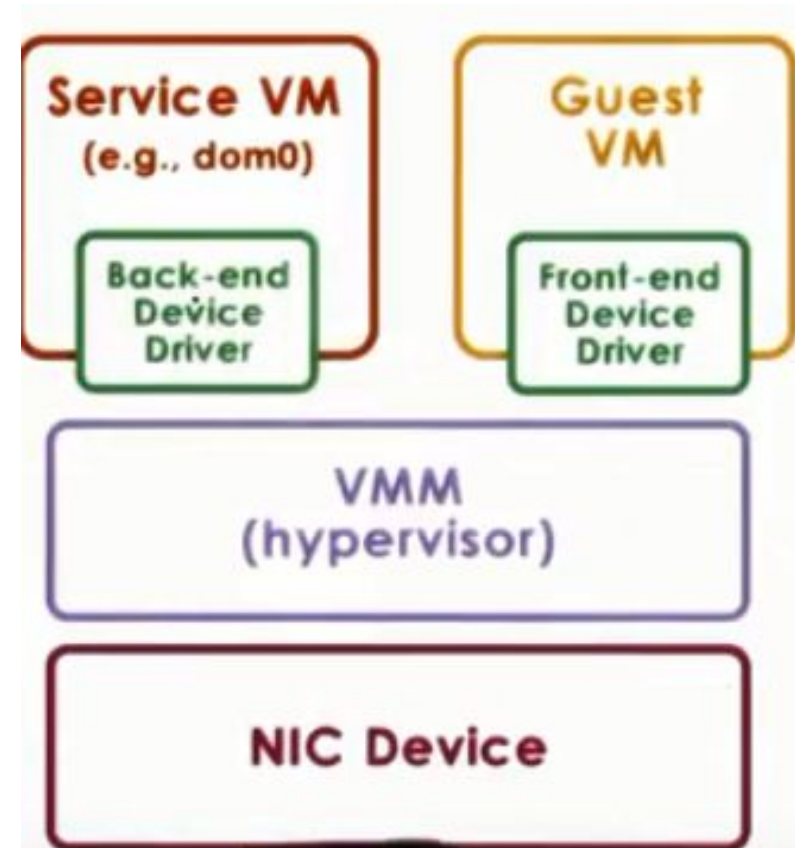
Device access control **split** between

- Front end driver in guest VM (device API)
- Back-end driver in service VM (or host)

Modified guest drivers

i.e. Limited to para-virtualized guests

Eliminate emulation overhead allow for better management of shared devices



Conclusion

- In this lecture, we have
- **defined virtualization** and discussed the main virtualization approaches.
- described **processor virtualization, memory virtualization and device virtualization** used in virtualization solutions, such as Xen, KVM and the VMware.