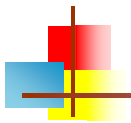


Cloud Computing

Virtual Machines and Virtualization

Dr P Victer Paul, IIIT Kottayam

contents

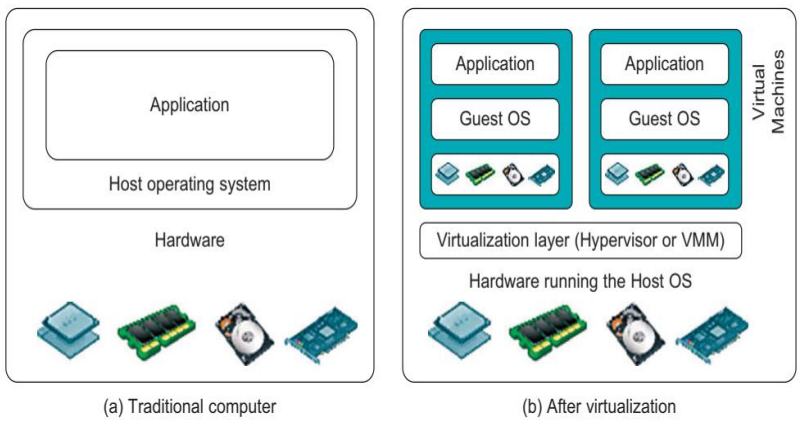
- Virtualization Technology
- Levels of Virtualization
- Relative Merits of Virtualization
- VMM & Design Requirements
- Virtualization Structures And Mechanisms

Dr P Victer Paul, IIIT Kottayam

Virtualization Technology

- enabling users *to share expensive hardware resources* by multiplexing VMs on the same set of hardware hosts
- *allows a single machine to act as if it were many machines*
  - multiple virtual machines (VMs) are multiplexed in the same hardware machine
- implemented using **hypervisors**
- Purpose of a VM
  - to enhance resource sharing by many users and improve computer performance in terms of *resource utilization and application flexibility*
- According to a 2009 Gartner Report, *virtualization* was the top strategic technology poised to change the computer industry

Virtualization Technology



(a) Traditional computer (b) After virtualization

Virtualization Technology

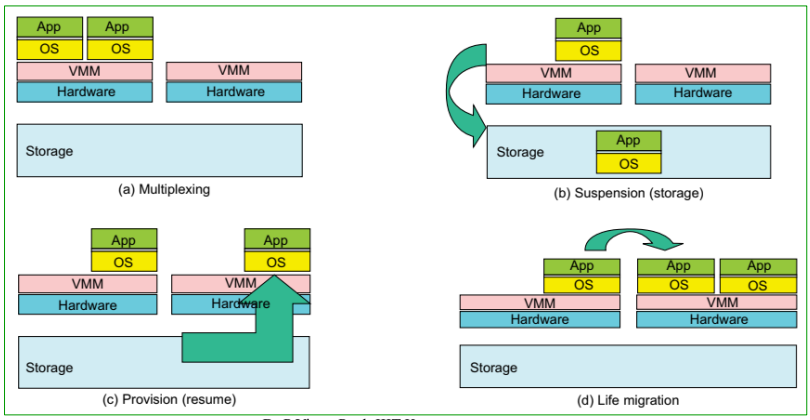
- After virtualization, different user applications managed by their own operating systems (guest OS) can *run on the same hardware, independent of the host OS*.
- This is often done by adding additional software, called a virtualization layer.
  - known as hypervisor or virtual machine monitor (VMM)
- function of the software layer for virtualization
  - *virtualize the physical hardware of a host machine into virtual resources*
  - *Allocates these resources exclusively to each virtual machine*
  - *Ensures isolation and security between VMs*

Virtualization and Virtual Machines

- A conventional computer has a single OS image
  - architecture that tightly couples application software to a specific hardware platform
- Virtual machines (VMs) offer solutions to underutilized resources, application inflexibility, software manageability, and security concerns in existing physical machines

VM Primitive Operations

- VMM provides the VM abstraction to the guest OS
- Low-level VMM operations and enable a VM to be provisioned to any available hardware platform

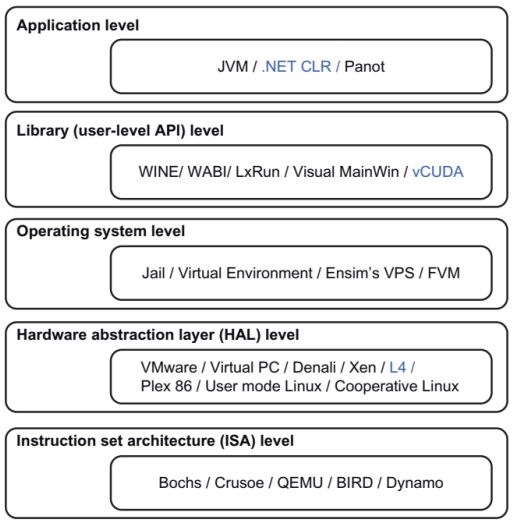


Implementation Levels of Virtualization

- Hardware resources (CPU, memory, I/O devices, etc.) or software resources (operating system and software libraries) can be virtualized in various functional layers
- virtualization layers include the
  - *instruction set architecture (ISA) level,*
  - *hardware level,*
  - *operating system level,*
  - *library support level, and*
  - *application level*

Levels of Virtualization

Virtualization ranging from hardware to applications in five abstraction levels.



Levels of Virtualization

- Lower levels → more flexibility, but lower performance
- Higher levels → better performance, but less flexibility
- each layer:
  - ***ISA Level:*** Virtualizes the processor instruction set
  - ***Hardware Level:*** Virtualizes physical hardware using hypervisors
  - ***OS Level:*** Multiple isolated environments share the same kernel
  - ***Library Level:*** Replaces or intercepts API calls
  - ***Application Level:*** Virtualizes the runtime environment

Instruction Set Architecture Level

- Instruction Set Architecture (ISA) is *the fundamental design of a computer's processor.*
  - It defines the types of instructions the processor can execute, the size and format of the instructions
- MIPS (Microprocessor without Interlocked Pipeline Stages) and x86 are two different instruction set architectures (ISA) used by processors.
- MIPS and x86 processors have different instruction sets, memory management, and other hardware features, *making it difficult or impossible for software written for one architecture to run on the other* without modification or translation.

Instruction Set Architecture Level

- virtualization is performed *by emulating a given ISA by the ISA of the host machine*
- Instruction set virtualization is a processor virtualization technique that *enables emulation of the instruction set of one processor on a different processor.*
  - MIPS binary code can run on an x86-based host machine with the help of ISA emulation.
- The basic emulation method is through *code interpretation*.
  - An interpreter program *interprets the source instructions to target instructions one by one*.
  - One source instruction may require tens or hundreds of native target instructions to perform its function.
  - Obviously, this process is relatively slow
- *Dynamic binary translation* - approach translates basic blocks of dynamic source instructions to target instructions

Instruction Set Architecture Level

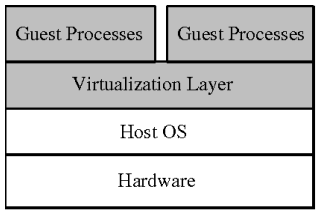
- Typical systems: Bochs, Crusoe, Quemu, BIRD, Dynamo
- Advantage:
  - It can run a large amount of legacy binary codes written for various processors on any given new hardware host machines
  - best application flexibility
- Limitation:
  - One source instruction may require tens or hundreds of native target instructions to perform its function, which is relatively slow.

Hardware Abstraction Level

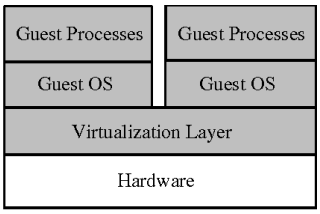
- Hardware-level virtualization is performed right on top of the bare hardware.
  - On the one hand, this *approach generates a virtual hardware environment for a VM*.
  - On the other hand, *the process manages the underlying hardware through virtualization*.
- idea is to virtualize a computer's resources, such as its processors, memory, and I/O devices
  - Typical systems: VMware vSphere with ESX/ESXi, KVM, Microsoft Hyper-V, Oracle VM (Xen), Denali, Xen
- Advantage:
  - Has higher performance and good application isolation
  - *upgrade the hardware utilization rate by multiple users concurrently*
- Limitation:
  - Very expensive to implement (complexity)

Operating System Level

- refers to an *abstraction layer between traditional OS and user applications*
- OS-level virtualization creates *isolated containers* on a single physical server and the OS instances to utilize the hardware and software in data centers
- OS-level virtualization is commonly used in creating *virtual hosting environments to allocate hardware resources among a large number of mutually distrusting users*.



OS-level Virtualization



Hardware-level Virtualization

Operating System Level

- allows multiple isolated user-space instances, often referred to as containers or virtual environments, *to run on a single operating system kernel*
- host operating system's kernel includes features to manage
  - *Namespaces – isolate process IDs, networking, file systems*
  - *cgroups (control groups) – control CPU, memory, and I/O usage*
  - *File system isolation*
- Key Features
  - *Shared Kernel, Lightweight, Isolation, Portability*
- *Linux Containers (LXC)*, Linux-based OS-level virtualization platform for creating and managing container

Library Support Level

- Most applications use APIs exported by user-level libraries rather than using lengthy system calls by the OS.
- *User-level libraries are software libraries that are available to user-level applications*,
  - as opposed to system-level libraries which are used by the operating system itself.
- Since most systems provide well-documented APIs, such an interface becomes another candidate for virtualization
- It creates *execution environments for running alien programs on a platform rather than creating VM to run the entire operating system*.
  - It is done by *API call interception and remapping*

- Typical systems: Wine, WAB, LxRun, VisualMainWin, Dockers
  - approach to support Windows applications on top of UNIX hosts
  - vCUDA - applications within VMs to leverage GPU hardware acceleration

Library Support Level

- Library-level virtualization refers to the *abstraction of an operating system's resources such as file systems, process management, and memory allocation through libraries*.
- This approach provides a layer of abstraction between the application and the underlying system, allowing the application to run on a different operating system without modification.
  - POSIX (Portable Operating System Interface) standard, which provides a uniform interface to operating system services across different platforms, and
  - the Wine project, which allows Windows applications to run on Linux and other Unix-like systems.

cygwin

- Library-level virtualization is also known as user-level Application Binary Interface (ABI) or API emulation

Library Support Level

- abstracting *system calls and application requests through a virtualized library layer*
- Layer provides an *interface between the application and the operating system*, allowing the application to function as though it is running in its native environment
  - even if the underlying OS or hardware is different
- *Applications interact with a virtualized library layer* instead of directly communicating with the OS
- System calls from the application are mapped to equivalent calls on the host operating system
- virtualized library hides differences between the application's expected environment and the host system

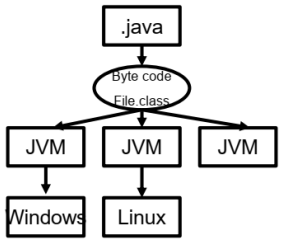
Wine and Cygwin

User-Application Level

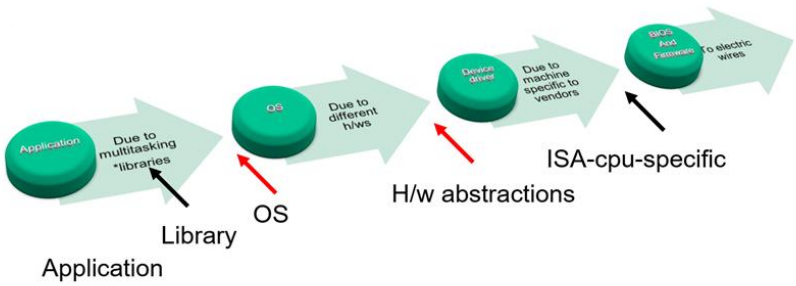
- *Virtualization at the application level virtualizes an application as a VM*
  - application often runs as a process also known as process-level virtualization
- virtualization layer sits as an application program on top of the operating system
  - exports an abstraction of a VM that can run programs written and compiled to a particular abstract machine definition.
- Any program written in the HLL and compiled for this VM will be able to run on it.
  - The Microsoft .NET CLR and Java Virtual Machine (JVM) are two good examples of this class of VM

User-Application Level

- application-level virtualization involves the *virtualization of an entire application or a portion of an application* into a self-contained environment.
- The *virtual environment includes all the dependencies and configuration required to run the application, making it portable* and easy to deploy on different systems.
- an application written for an architecture can be executed on the other architect
- VMware ThinApp, Citrix XenApp



Levels of Virtualization



Merits of Different Approaches

- Relative Merits of Virtualization at Various Levels (More "X"'s Means Higher Merit, with a Maximum of 5 X's)

Level of Implementation	Higher Performance	Application Flexibility	Implementation Complexity	Application Isolation
ISA	X	XXXXX	XXX	XXX
Hardware-level virtualization	XXXXX	XXX	XXXXX	XXXX
OS-level virtualization	XXXXX	XX	XXX	XX
Runtime library support	XXX	XX	XX	XX
User application level	XX	XX	XXXXX	XXXXX

VMM & Design Requirements

- Virtual Machine Monitor (VMM) *manages the hardware resources of a computing system*
- There are three requirements for a VMM.
  - provide *an environment for programs which is essentially identical to the original machine*
  - *programs run in this environment should show, at worst, only minor decreases in speed*
  - should be in *complete control of the system resources*
- A VMM should demonstrate *efficiency in using the VMs*
- Complete control of resources by a VMM includes :
  - responsible for *allocating hardware resources for programs*
  - a *program cant access any resource not explicitly allocated*
  - under certain circumstances for a VMM *to regain control of resources already allocated*

Popek and Goldberg virtualization requirements

VMM & Design Requirements

- A VMM is tightly related to the architectures of processors
- It is *difficult to implement a VMM for some types of processors*, such as the x86.
  - Specific limitations include the inability to trap on some privileged instructions
- If a processor is not designed to support virtualization primarily, it is *necessary to modify the hardware to satisfy the three requirements for a VMM*.
- This is known as *hardware-assisted virtualization*

VMM & Design Requirements

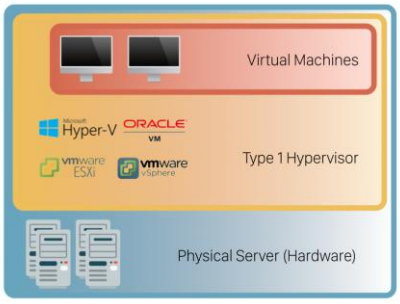
Provider and References	Host CPU	Host OS	Guest OS	Architecture
VMware Workstation [71]	x86, x86-64	Windows, Linux	Windows, Linux, Solaris, FreeBSD, Netware, OS/2, SCO, BeOS, Darwin	Full Virtualization
VMware ESX Server [71]	x86, x86-64	No host OS	The same as VMware Workstation	Para-Virtualization
Xen [7,13,42]	x86, x86-64, IA-64	NetBSD, Linux, Solaris	FreeBSD, NetBSD, Linux, Solaris, Windows XP and 2003 Server	Hypervisor
KVM [31]	x86, x86-64, IA-64, S390, PowerPC	Linux	Linux, Windows, FreeBSD, Solaris	Para-Virtualization

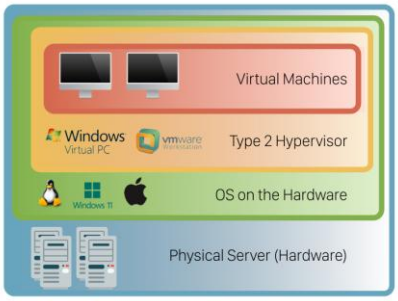
Comparison of Four VMM

Virtualization Structures And Mechanisms

- Depending on the position of the virtualization layer, there are several *classes of VM architectures*, namely the hypervisor architecture,
  - Type 1 Hypervisors, also known as bare-metal or native
  - Type 2 Hypervisors, also known as hosted hypervisors
- *Type 1 hypervisors* are mainly found in enterprise environments
  - VMware vSphere with ESX/ESXi, KVM, Microsoft Hyper-V, Oracle VM (Xen)
- *Type 2 hypervisors* are typically found in environments with a small number of servers
  - Oracle VM VirtualBox, VMware Workstation Pro/VMware Fusion, Windows Virtual PC, Parallels Desktop (macOS)

Virtualization Structures And Mechanisms

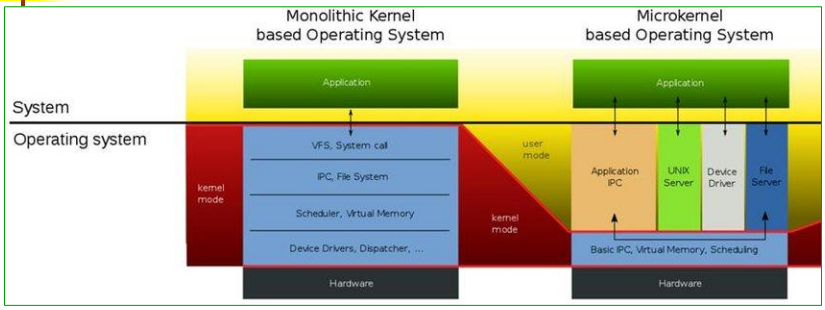


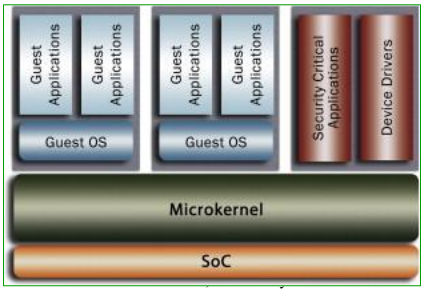


Hypervisor Functionality

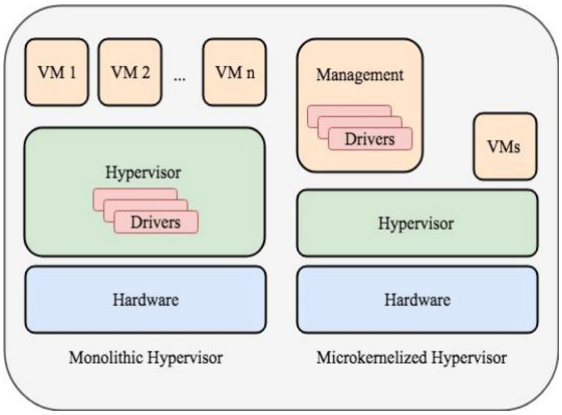
- provides *hypercalls for the guest OSes and applications*
- *Depending on the functionality*, a hypervisor can
  - *micro-kernel architecture*
  - *monolithic hypervisor architecture*
- micro-kernel architecture – MS Hyper V
  - only the basic and unchanging functions (such as physical memory management and processor scheduling)
  - device drivers and other changeable components are outside the hypervisor
- monolithic hypervisor architecture - ESX
  - *implements all the functions*, including those of the device driver
- the size of the hypervisor code of a micro-kernel hypervisor is smaller than that of a monolithic hypervisor

Hypervisor





Hypervisor



Xen - *An open-source hypervisor with a microkernel-style design*

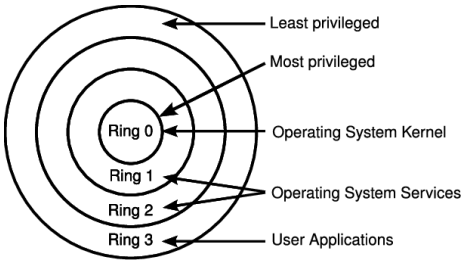
Virtualization types

- Depending on implementation technologies, virtualization can be classified into two categories
  - *full virtualization and para virtualization*
- Full virtualization *does not need to modify the host OS*
  - binary translation to trap and to virtualize instructions
  - In a host-based system, *both a host OS and a guest OS are used*
- Para-virtualization *needs to modify the guest operating systems*

special *privileges to system resources* are permitted by defining modes of operations. two modes of operation:

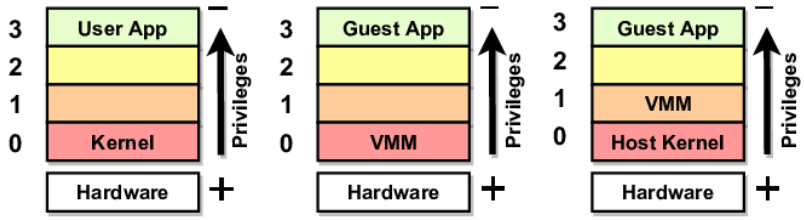
1. System (also called supervisor, kernel, or privileged) mode: *all resources are accessible to software*
2. User mode: *only certain resources are accessible to software*

Protection/Privilege Rings



to implement different levels of access control and privilege separation

Security, Isolation, Stability, Efficiency



Protection/Privilege Rings

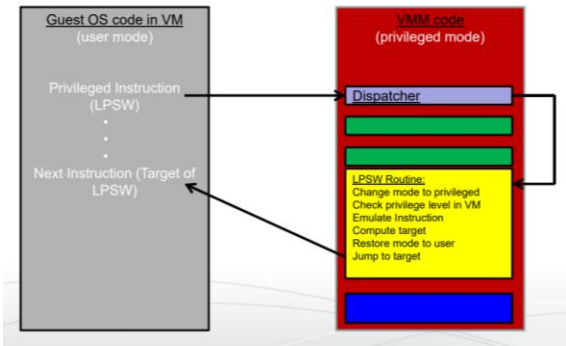
- The kernel, which is at the heart of the operating system and has access to everything, can access *Ring 0*
  - kernel mode
- *Ring 3* doesn't have any access to the CPU or memory
  - any instructions involving these must be passed to ring 0
- *rings 1 and ring 2* offer unique advantages that ring 3 lacks
  - In ring 2, for example, loading an Excel file document from storage.
  - ring 3 will be responsible for editing and saving the data.

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.
Ex. Load PSW, Set CPU Timer, mov eax, cr0

mov, add, and sub

Protection/Privilege Rings

- In a native system VM, the *VMM runs in system mode*, and all "other" (e.g., guest OS) software run in user mode
- An *OS running on a guest VM should not be allowed to change hardware resources*
- guest OSs are all forced to run in user mode



ensures isolation, security, and control

Full Virtualization

- *noncritical instructions run on the hardware directly*
- critical instructions are *discovered and replaced with traps into the VMM* to be emulated by software.

- Why are only critical instructions trapped into the VMM?
  - binary translation can incur a large performance overhead

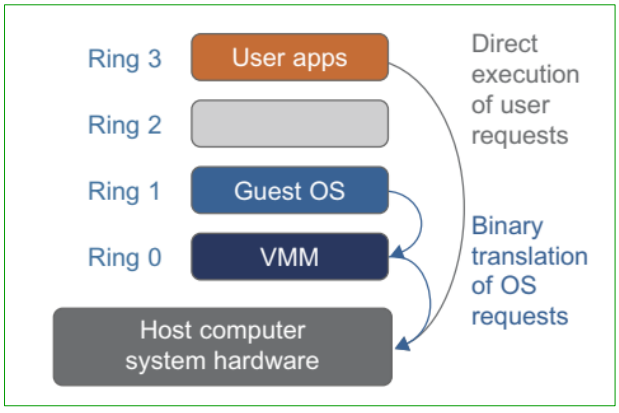
operations that require exclusive access (atomicity) to certain resources or that have a significant impact on the system's state
CLI, STI, PUSH, POP, LOCK, FORK, EXEC

Binary Translation of Guest OS Requests

- VMware puts the *VMM at Ring 0 and the guest OS at Ring 1*
- The VMM scans the instruction stream and *identifies the privileged instructions* (*control- and behavior-sensitive*)
  - If so they are trapped into the VMM, which emulates the behavior of these instructions
  - this emulation is called binary translation
- full virtualization combines *binary translation and direct execution*
  - *guest OS is completely decoupled from the underlying hardware*
- binary translation which is rather time-consuming
  - the full virtualization of *I/O-intensive applications is a really a big challenge*
- employs a *code cache* to store translated hot instructions to *improve performance*
  - increases the cost of memory usage

balances performance and safety.

Binary Translation of Guest OS Requests



Host-Based Virtualization

- An alternative VM architecture is to install *a virtualization layer on top of the host OS*
  - guest OSes are installed and run on top of the virtualization layer
- some applications can *run with the host OS directly*
- *virtualizing software can rely on the host OS* to provide device drivers and other low-level services
- When an application requests hardware access
  - *it involves four layers of mapping which downgrades performance significantly.*
- When the *ISA of a guest OS is different from the ISA of the underlying hardware*
  - binary translation must be adopted
  - architecture has flexibility, the performance is too low

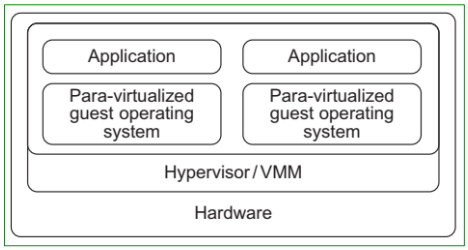
Para-Virtualization

- needs to *modify the guest operating systems*
- performance degradation is a critical issue of a virtualized system
  - para-virtualized VM *provides special APIs requiring substantial OS modifications*
- para-virtualization attempts to *reduce the virtualization overhead*
  - thus improve performance by modifying only the guest OS kernel
- guest operating systems are para-virtualized
  - assisted by an intelligent compiler to replace the *non-virtualizable OS instructions by hypercalls*
- *Instead of executing non-virtualizable instructions:*
  - *The guest OS uses special APIs called hypercalls*
- *This significantly improves performance by:*
  - *Avoiding binary translation*
  - *Reducing trap-and-emulate operations*

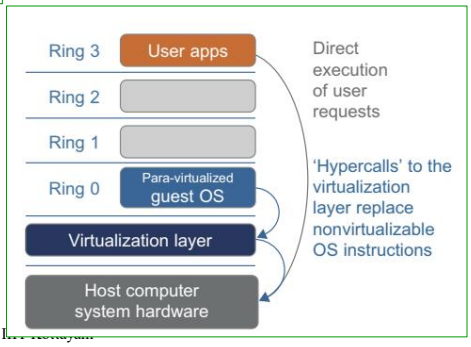
Hypercalls directly invoke VMM services

Para-Virtualization

Only supported operating systems can be used



para-virtualized VM architecture



Para-Virtualization Architecture

- para-virtualization replaces *nonvirtualizable instructions with hypercalls* that communicate directly with the hypervisor or VMM
- Although para-virtualization reduces the overhead, it has incurred other problems
- Its *compatibility and portability* may be in doubt
  - because it must support the unmodified OS as well
- the *cost of maintaining* para-virtualized OSes is high
  - because they may require deep OS kernel modifications
- Xen, KVM and ESXi are examples

https://app.datacamp.com/learn/courses/introduction-to-gcp

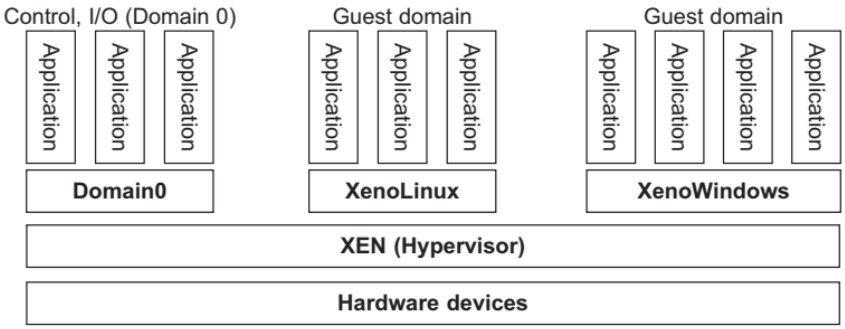
Full vs Para-Virtualization

- the full virtualization architecture which *intercepts and emulates* privileged and sensitive instructions at runtime
- para-virtualization handles these instructions at compile time
  - The guest OS kernel is *modified to replace the privileged and sensitive instructions with hypercalls* to the hypervisor or VMM
  - a dedicated service routine

The Xen Architecture

- Xen is an *open source hypervisor program developed at Cambridge University*
- a microkernel hypervisor - does not include any device drivers natively
- provides a mechanism by which a guest OS can have direct access to the physical devices
  - commercial Xen hypervisors, are *Citrix XenServer and Oracle VM*
- core components of a Xen system are the *hypervisor, kernel, and applications*
- not all guest OSes are created equal, and *one in particular controls the others*
  - Xen introduces domain-based isolation
  - guest OS with control ability is called *Domain 0*, and the others are called *Domain U*

The Xen Architecture



Xen separates mechanism (in the hypervisor) from policy (in Domain 0).

The Xen Architecture

- Xen hypervisor implements all the mechanisms, leaving the policy to be handled by Domain 0
- Domain 0 is a *privileged guest OS of Xen*
  - *first loaded when Xen boots* without any file system drivers being available
  - designed to *access hardware directly and manage devices*
- responsibilities of Domain 0 is to
  - *allocate and map hardware resources for the guest domains*
  - *Creating, copying, saving, reading, modifying, and deleting VMs*
  - *Migrating and rolling back virtual machines*
- If Domain 0 is compromised, the hacker can control the entire system
- Domain 0 allows users to *all primitive operations with VMs*

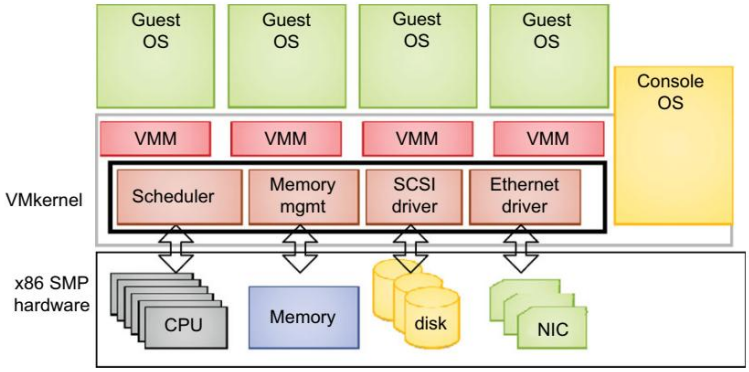
VMware ESX Server for Para-Virtualization

- *VMware* pioneered the software market for virtualization
- ESX is a VMM or a hypervisor for *bare-metal x86 symmetric multiprocessing (SMP) servers*
- ESX-enabled server consists of four components:
  - a virtualization layer,
  - a resource manager,
  - hardware interface components, and
  - a service console

Elastic Sky X

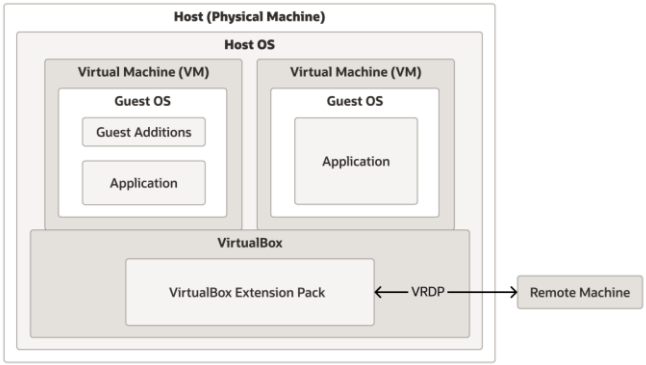
- ESX server employs a para-virtualization architecture in which the VM kernel interacts directly with the hardware without involving the host OS
  - physical hardware resources such as *CPU, memory, network and disk controllers, and human interface devices*

VMware ESX Server for Para-Virtualization



- *VMM layer* virtualizes the physical hardware resources
- *resource manager* allocates CPU, memory disk, and network bandwidth and maps them to the virtual hardware resource
- *Hardware interface components* are the device drivers and the Server File System
- *Service console* is responsible for booting the system, initiating the execution of the VMM and resource manager

Oracle VirtualBox Components



- *cross-platform virtualization application*
- host is the physical computer, such as a laptop or server, where you run VirtualBox and the virtual machines created
- Oracle VirtualBox Extension Pack is an optional, separately licensed –
  - *VirtualBox Remote Desktop Protocol (VRDP) server* - allows remote clients to connect to a VM using RDP

Oracle VirtualBox Components

- The Host OS (Windows / Linux / macOS) is responsible for:
  - Hardware drivers, Device management, System calls, File systems
- The VirtualBox layer is the virtualization engine
- Its responsibilities include:
  - Creating and managing virtual machines, CPU instruction virtualization, Memory virtualization
- The Extension Pack is installed on the host, not inside the VM.

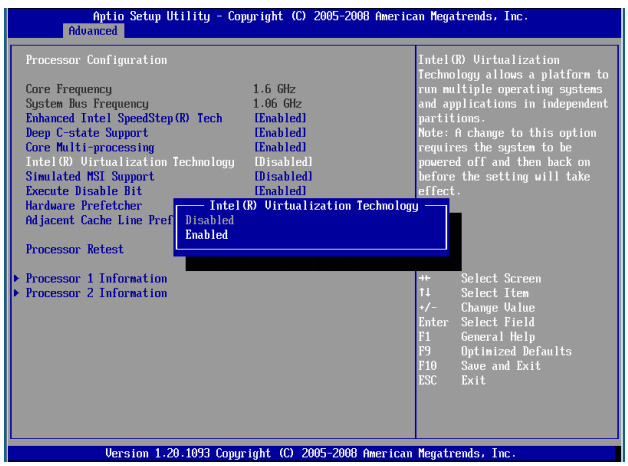
Virtualization of CPU, Memory, and I/O Devices

Hardware Support for Virtualization

- In general, *user mode and supervisor mode*, to ensure controlled access of critical hardware
  - supervisor mode are called *privileged instructions*
  - In a virtualized environment, it is more *difficult to make OSes and applications run correctly* because there are more layers
- To support virtualization, processors such as the *x86 employ a special running mode and instructions*, known as hardware-assisted virtualization
  - *save processor states, mode switching is completed by hardware*
- *Intel and AMD* have technologies for hardware-assisted virtualization (*VT-x and AMD-v*)
- Hardware virtualization products
  - VMware Workstation, Xen, KVM

Hardware Support for Virtualization in the Intel x86

Intel Virtualization Technology (VT) formerly known as Vanderpool, *enables a CPU to act as if you have several independent computers*, in order to enable several operating systems to run at the same time on the same machine.

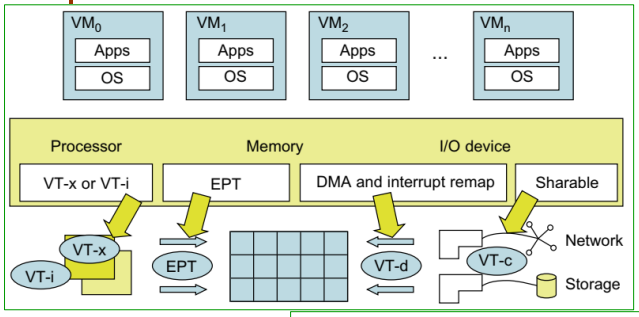


H/w virtualization support should be enabled in the BIOS setup

Hardware Support for Virtualization in the Intel x86

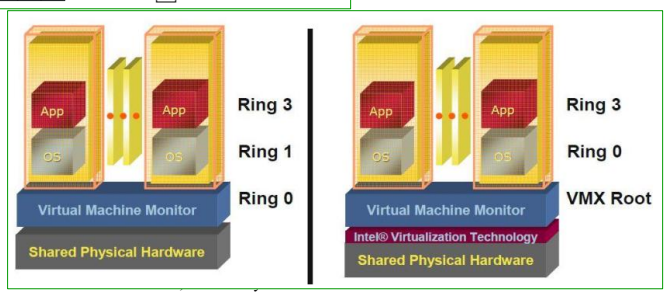
- Since *software-based virtualization techniques are complicated and incur performance overhead*
  - Intel provides a *hardware-assist technique* to make virtualization easy and improve performance
- *For processor virtualization*,
  - Intel offers the VT-x or VT-i technique (x86 or Itanium) (AMD-V)
  - VT-x adds a privileged mode (*VMX Root Mode*) - *high priority layer was introduced at the hardware level*
  - enhancement traps all sensitive instructions in the VMM automatically
- *For memory virtualization* - Extended Page Tables
  - Intel offers the EPT, which translates the virtual address to the machine's physical addresses to improve performance (Core i3, i5 and i7)
- *For I/O virtualization*
  - Intel implements VT-d and VT-c

Hardware Support for Virtualization in the Intel x86



The main idea was to quickly identify the privileged instructions and to efficiently execute them

Intel's full virtualization techniques - Intel hardware support for virtualization of processor, memory, and I/O devices



CPU Virtualization

- In VMs, instructions are executed on the host processor in native mode
  - unprivileged instructions of VMs run directly on the host machine for higher efficiency
  - Other critical instructions should be handled carefully for *correctness and stability*
- *critical instructions are divided into three categories*:
  - *Privileged instructions* execute in a privileged mode and will be trapped if executed outside this mode
  - *Control-sensitive instructions* attempt to change the configuration of resources used (modify the system registers, POPF PUSHF)
  - *Behavior-sensitive instructions* have *different behaviors depending on the configuration of resources*, including the load and store operations over the virtual memory (*behavior depends on the processor privilege level*)

    - CLI (Clear Interrupt Flag) - a guest OS trying to disable interrupts should only disable them for its own virtual machine

CPU Virtualization

- *Privileged Instructions*
  - I/O Instructions: IN, OUT (direct hardware access)
  - Control Register Access: Modifying CPU control registers (e.g., MOV CR3, RAX to change page tables)
  - Interrupt Management: CLI, STI (enable/disable interrupts)
  - System Call Instructions: INT 0x80 (used for OS-level interactions)
- *Sensitive Instructions*
  - Reading CPU State: CPUID (can expose that a system is virtualized)
  - Modifying Segment Registers: LGDT, LIDT (loading Global/Interrupt Descriptor Tables)
  - Memory Management Instructions: MOV to CR3 (changing page tables affects memory mapping)
- *Un-privileged Instructions*

  - General Arithmetic and Logical Operations: ADD, SUB, MUL, XOR.
  - Memory Operations: MOV, PUSH, POP.
  - Branching and Control Instructions: JMP, CALL, RET.

CPU Virtualization

- the *VMM acts as a unified mediator for hardware access from different VMs* to guarantee the correctness and stability
- *not all CPU architectures are virtualizable*
- *RISC CPU architectures can be naturally virtualized*
  - because all control- and behavior-sensitive instructions are privileged instructions
- *x86 CPU architectures are not* designed to support virtualization
  - because about 10 sensitive instructions are not privileged instructions
  - When these instructions execute in virtualization, they cannot be trapped in the VMM - Executed silently in user mode
  - Hypervisor could **not detect** these instructions reliably

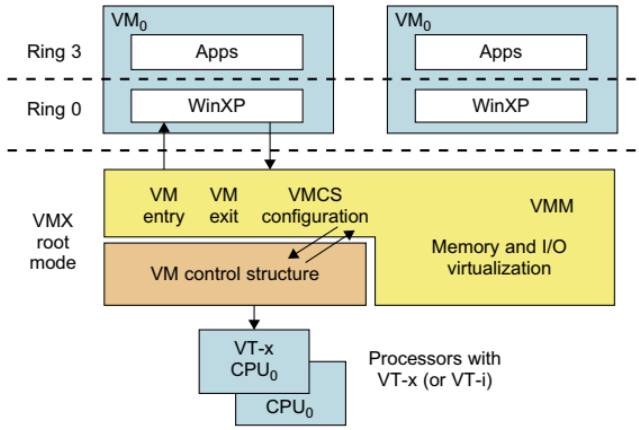
Hardware-Assisted CPU Virtualization

- *Intel and AMD add an additional mode called privilege mode level* to x86 processors
  - technique removes the *difficulty of implementing binary translation* of full virtualization
  - also lets the operating system run in *VMs without modification*
- *The CPU itself becomes virtualization-aware.*
- Intel VT-x / AMD-V modify the processor so that:
  - *All sensitive and privileged instructions are automatically intercepted by hardware*
  - *Hypervisor no longer needs to rewrite guest code*

Hardware-Assisted CPU Virtualization

- With Intel VT-x, there are two distinct modes of CPU operation:
  - *VMX root mode and non-root mode*
- In root mode,
  - Used by the hypervisor (VMM)
  - Has full control over the system
  - *root mode is what a host operating system without virtualization uses*
- In non-root mode,
  - Used by guest operating systems
  - Guest OS can run at Ring 0 without accessing real hardware
  - a new structure called VMCS (Virtual Machine Control Structure) now controls the *CPU operation and determines how certain instructions behave*

Hardware-Assisted CPU Virtualization



- VMCS – memory processor uses to store and keep track of data requires during VMX transition (ie) during switching from VM non root to VM root operation
- The VMCS is a processor-defined data structure that stores: Guest CPU state, Host CPU state, Control flags

Hardware-Assisted CPU Virtualization

- *Working Process*
- A guest OS (WinXP) runs inside a VM at Ring 0, while applications execute in Ring 3.
- The VMM (Hypervisor) manages multiple VMs by leveraging Intel VT-x.
- When a guest OS executes a privileged instruction, the VM exit mechanism transfers control to the VMM, which processes the request and then re-enters the VM
  - *VM Entry:* When the CPU switches execution to a VM, it enters non-root mode, allowing the guest OS (WinXP) to run.
  - *VM Exit:* When a VM executes a privileged instruction that requires hypervisor intervention, the CPU exits VM execution and returns control to the VMM.

The VM Control Structure (VMCS) maintains VM execution states, including: CPU state, Memory mappings, I/O permissions.

Hardware-Assisted CPU Virtualization

- *VM Exit Mechanism (Hardware Trap)*
- When a sensitive instruction executes:
  - CPU detects it *in hardware*
  - CPU stops guest execution
  - CPU switches to *VMX Root Mode*
  - Control is handed to the hypervisor
  - Hypervisor emulates or approves the operation
  - CPU resumes guest execution via *VM Entry*
- All of this happens *without modifying guest code*.

Advantages

- *Reduces Overhead* – Eliminates the need for software-based binary translation.
- *Better Performance* – Efficiently handles privileged instructions with VM Exits.
- *Improved Isolation* – Keeps the VMM separate from guest OSes, ensuring security.
- *Faster Context Switching* – Uses VMCS to manage execution states quickly.

Hardware-Assisted CPU Virtualization

- hardware-assisted virtualization *should have high efficiency*
- since the transition from the hypervisor to the guest OS incurs *high overhead switches between processor modes*
  - it sometimes cannot outperform binary translation
- virtualization systems such as VMware now use *a hybrid approach*
  - *a few tasks are offloaded to the hardware but the rest is still done in software*
  - *para-virtualization and hardware-assisted virtualization can be combined* to improve the performance

Memory Virtualization

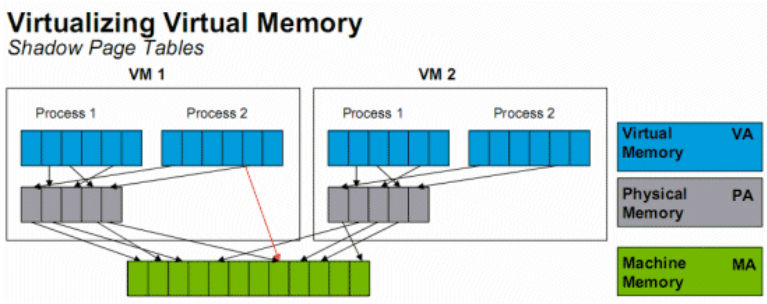
- *Memory virtualization = the virtual memory support provided by OS*
- In a traditional execution environment,
  - OS maintains mappings of virtual memory to machine memory using page tables
  - x86 CPUs include a memory management unit (MMU) and a *translation lookaside buffer (TLB)* to optimize
  - *a one-stage mapping* from virtual memory to machine memory
- in a virtual execution environment
  - virtual memory virtualization involves sharing the physical system memory in RAM
  - dynamically allocating it to the physical memory of the VMs
  - *a two-stage mapping process* should be maintained by the guest OS and the VMM
    - ***virtual memory to physical memory and physical memory to machine memory***

Memory Virtualization

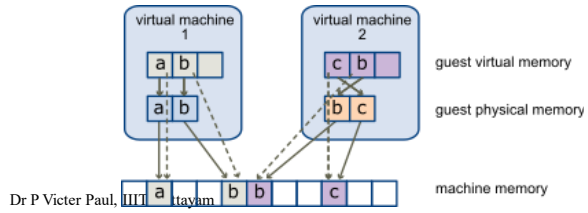
- Since *each page table of the guest OSes has a separate page table in the VMM* corresponding to it (does the mapping of addresses)
  - the VMM page table is called the *shadow page table*
- The *MMU handles virtual-to-physical translations* as defined by the OS
- Then the *physical memory addresses are translated to machine addresses* using another *set of page tables defined by the VMM*
- Since modern operating systems maintain a set of page tables for every process
  - the *shadow page tables will get flooded*
- Consequently, the *performance overhead, cost of memory will be very high and poor scalability*
  - shadow paging process takes *3 to 400 times more cycles than native*

Memory Virtualization

Guest OS cannot directly access the machine memory



Two-level memory mapping procedure



Memory Virtualization Works

- Memory virtualization involves multiple layers of address translation:
- Guest Virtual Address (GVA) → Guest Physical Address (GPA)
  - The guest OS maintains its own virtual memory paging.
  - The guest perceives its memory as a contiguous block, unaware of the actual hardware layout.
- Guest Physical Address (GPA) → Host Physical Address (HPA)
  - The hypervisor (or Virtual Machine Monitor, VMM) translates guest physical addresses to actual host physical addresses.
  - This translation is necessary because multiple VMs share the same host memory.
- Extended Page Table (EPT) - hardware-assisted memory virtualization feature introduced by Intel VT-x as part of Intel's Virtualization Technology.
  - reduces the overhead of memory management in virtualized environments by *eliminating the need for repeated software-based address translations*

Memory Virtualization

EPT enables the processor to perform *two-level address translation in hardware.*

VM Virtual Address
↓ *(guest OS page table)*
Guest Physical Address
↓ *(shadow page table by VMM)*
Host Physical Address

Address Translation Without EPT
(Software-Based)

VM Virtual Address
↓ *(guest page tables)*
Guest Physical Address
↓ *(EPT tables – hardware)*
Host Physical Address

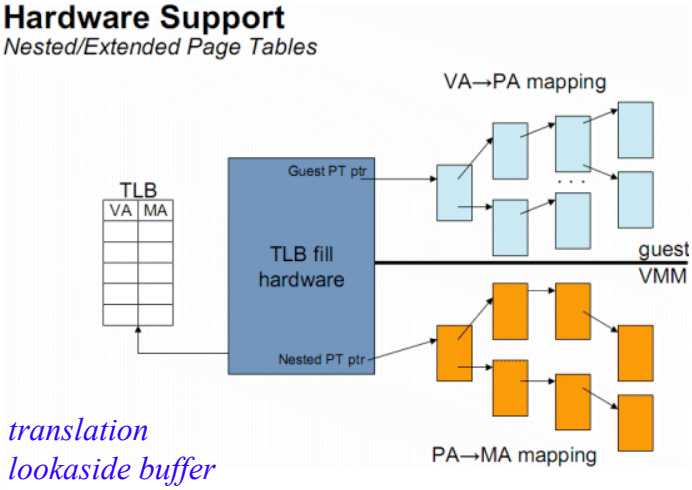
Address Translation With EPT
(Hardware-Assisted)

Memory Virtualization

EPT enables the processor to perform *two-level address translation in hardware.*

Dr P Victer Paul, IIIT Kottayam

Memory Virtualization



translation lookaside buffer (TLB)

 https://www.anandtech.com/show/2480/10

Memory Virtualization

- *VMware uses shadow page tables* to perform virtual-memory-to-machine-memory address translation
- Processors use *TLB hardware to map the virtual memory directly to the machine memory*
  - to avoid the two levels of translation on every access
- When the guest OS changes the virtual memory to a physical memory mapping
  - the *VMM updates the shadow page tables to enable a direct lookup*
- provides hardware assistance to the two-stage address translation in a virtual execution environment
- *Extended Page Table (EPT) by Intel for Memory Virtualization*

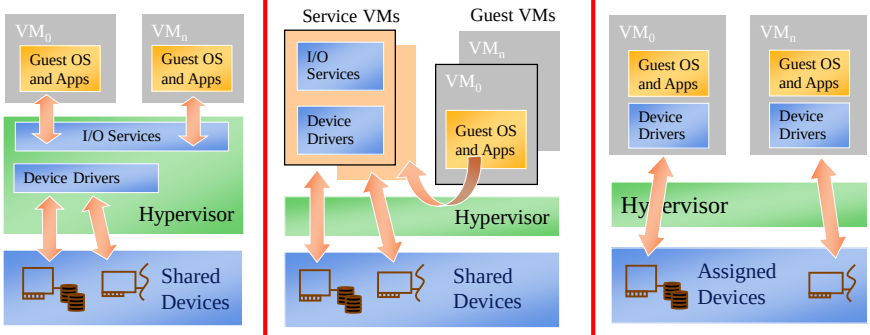
I/O Virtualization

https://www.elby.ch/en/products/vcd.html
https://www.virtualcd-online.com/

I/O Virtualization

- involves managing the *routing of I/O requests between virtual devices and the shared physical hardware*
- three ways to implement I/O virtualization:
  full device emulation, para-virtualization, and direct I/O
- Full Device Emulation
  - *emulates real-world devices*
  - functions like *device enumeration, identification, interrupts, and DMA*, are replicated in software
  - software is *located in the VMM* and acts as a virtual device
  - *I/O access requests of the guest OS are trapped in the VMM which interacts with the I/O devices*
  - *single hardware device can be shared by multiple VMs* that run concurrently
  - Software emulation runs *much slower than the hardware* it emulates

I/O Virtualization

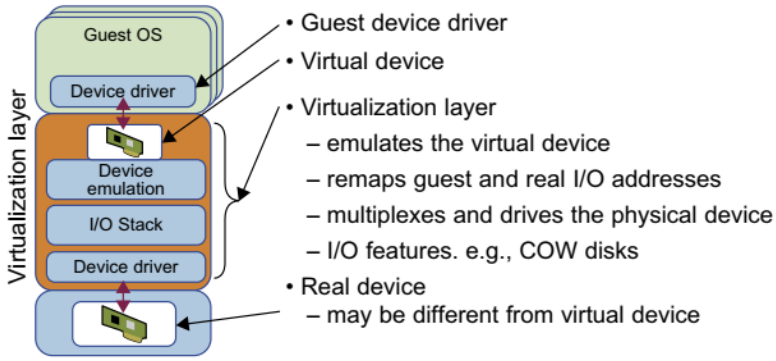


- Pro: Higher Performance
- Pro: I/O Device Sharing
- Pro: VM Migration
- Con: Larger Hypervisor

- Pro: High Security
- Pro: I/O Device Sharing
- Pro: VM Migration
- Con: Lower Performance

- Pro: Highest Performance
- Pro: Smaller Hypervisor
- Pro: Device assisted sharing
- Con: Migration Challenges

I/O Virtualization

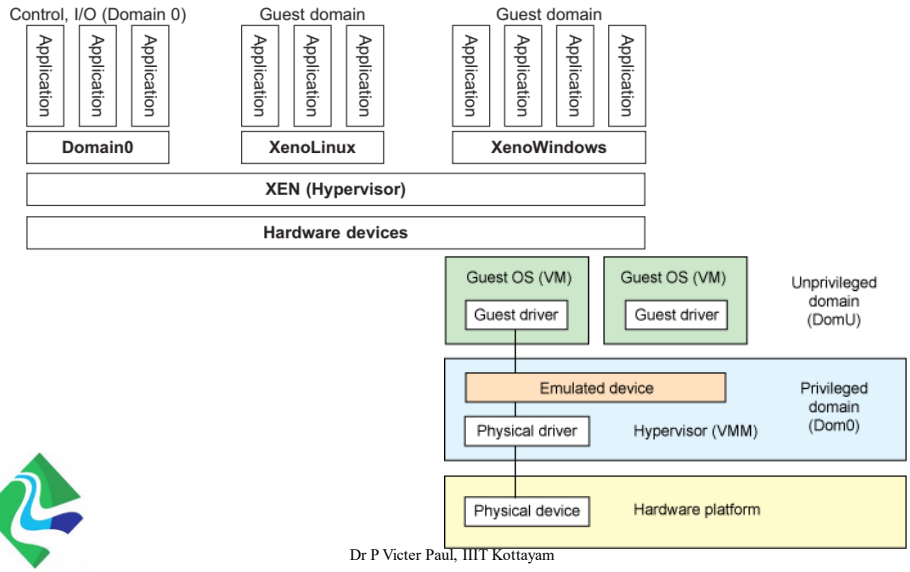


Device emulation for I/O virtualization

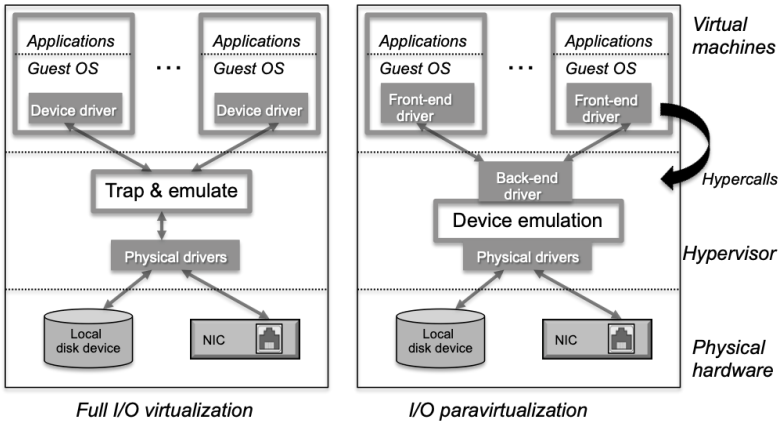
I/O Virtualization

- Para-virtualization
  - *typically used in Xen*
  - also known as the *split driver model* - frontend driver and a backend driver
  - frontend driver is running in Domain U and the backend driver is running in Domain 0
  - interact with *each other via a block of shared memory*
  - Frontend driver manages the I/O requests of the guest OSes
  - backend driver is responsible for *managing the real I/O devices and multiplexing the I/O data of different VMs*
  - achieves better device performance than full device emulation, *but, with a higher CPU overhead*

The Xen Architecture



I/O full and paravirtualization

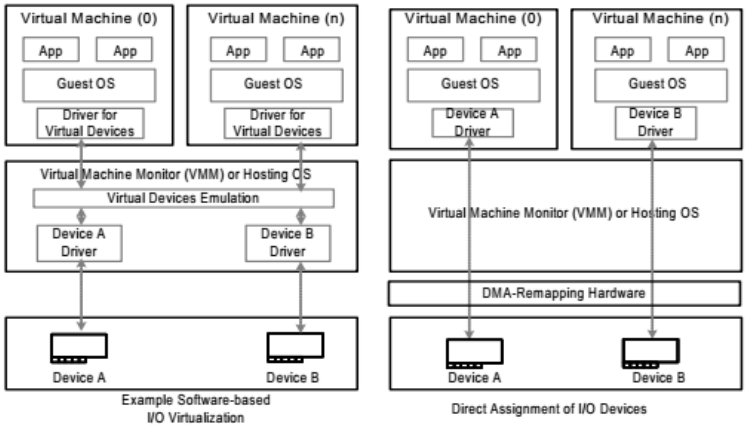


http://journal.auric.kr/kieep/XmlViewer/f387513

I/O Virtualization

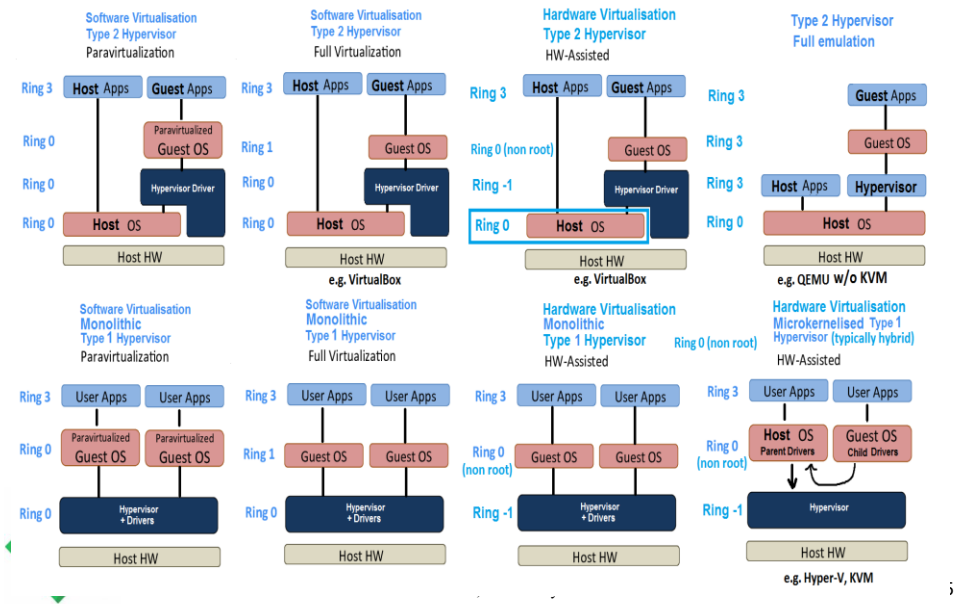
- Direct I/O virtualization
  - *VM access devices directly*
  - achieve close-to-native performance *without high CPU costs*
  - Repeated physical device is reclaimed for later reassignment can function incorrectly or even crash the whole system
- *software-based I/O virtualization requires a very high overhead* of device emulation, hardware-assisted I/O virtualization is critical
- Intel VT-d supports the remapping of I/O DMA transfers and device-generated interrupts.
  - providing hardware support for *isolating and restricting device accesses to the owner* of the partition managing the device.
  - The architecture of VT-d provides the flexibility to support multiple usage models that may run *unmodified, special-purpose, or "virtualization-aware" guest OSes*

I/O Virtualization



I/O Virtualization types

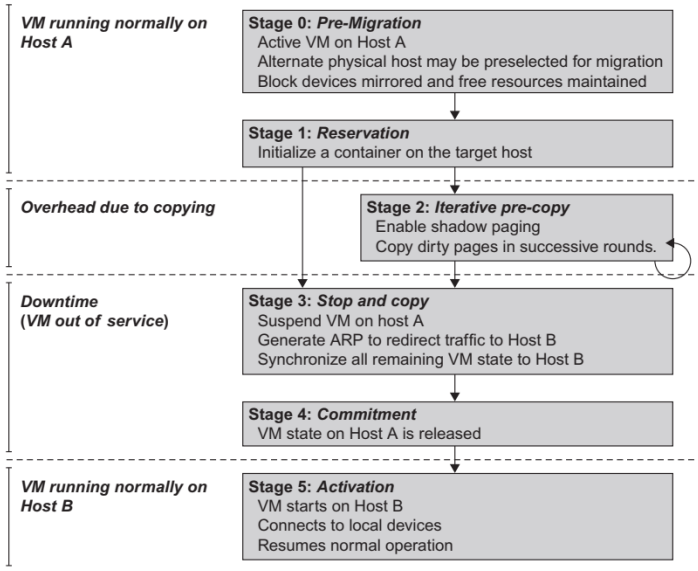
Aspect	*Full Emulation*	*Para-Virtualization*	*Direct I/O*
Guest OS changes	None	PV drivers	None (but hardware support)
Performance	Low	High	Near-native
CPU overhead	High	Medium–Low	Very Low
Compatibility	Very High	High	Limited
Hypervisor involvement	High	Medium	Minimal (control only)
Live migration	Easy	Easy	Hard/No
Examples	IDE/e1000 emulation	virtio-blk/net	PCI passthrough, SR-IOV



Live VM Migration

- VMs can be live-migrated from one physical machine to another
- When a VM runs a live service, it is necessary to make *a trade-off to ensure that the migration occurs minimize three metrics*.
- The motivation is to design a live VM migration
  - *negligible downtime, the lowest network bandwidth consumption possible, and a reasonable total migration time*
- A VM can be in one the following four states.
  - *An inactive state* - the VM is not enabled.
  - *An active state* refers to a VM that has been instantiated at the virtualization platform to perform a real task.
  - *A paused state* corresponds to a VM that has been instantiated but disabled to process a task or paused in *a waiting state*.
  - A VM enters *the suspended state* if its machine file and virtual resources are stored back to the disk.

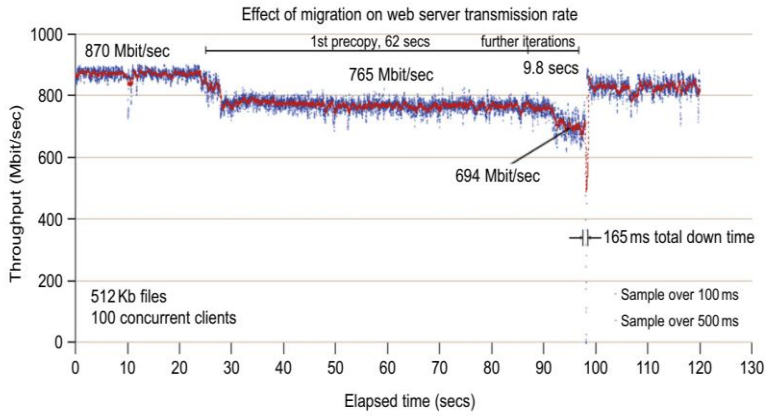
Live VM Migration



Live VM Migration

- live migration of a VM consists of the following six steps
  - *Steps 0 and 1: Start migration*. This step makes preparations for the migration, including determining the migrating VM and the destination host.
  - *Steps 2: Transfer memory*. Since the whole *execution state of the VM is stored in memory*, sending the VM's memory to the destination node ensures continuity of the service provided by the VM.
  - *Step 3: Suspend the VM and copy the last portion of the data*. The migrating VM's execution is suspended when the last round's memory data is transferred. During this step, the VM is stopped and its applications will no longer run. *"service unavailable" time is called the "downtime" of migration*
  - *Steps 4 and 5: Commit and activate the new host.* After all the needed data is copied, on the destination host, the VM reloads the states and recovers the execution of programs in it

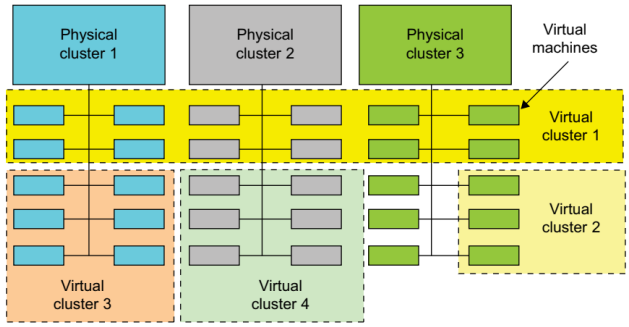
Live VM Migration



live migration of a VM from one host to another - Effect on data transmission rate

Virtual clusters and resource mgmt

- A physical cluster is a *collection of servers (physical machines) interconnected by a physical network such as a LAN*
- Virtual clusters are built with *VMs installed at distributed servers from one or more physical clusters*.
- The VMs in a virtual cluster are interconnected logically by a virtual network across several physical networks.



Virtualization for Data-center Automation

- companies have invested billions of dollars in *datacenter construction and automation*
- Data-center automation means that
  - huge volumes of *hardware, software, and database resources in these data centers can be allocated dynamically* to millions of Internet users simultaneously, with guaranteed QoS and cost-effectiveness
- *highlights high availability (HA), backup services, mobility, workload balancing, reducing planned downtime*

Server Consolidation in Data Centers

- Data centers, a *large number of heterogeneous workloads* can run on servers at various times
- two categories: *chatty workloads and non-interactive workloads*
- *Chatty workloads* burst at some point and silent state at some other point eg. A web video service
- *Non-interactive workloads* do not require people's efforts to make progress after they are submitted eg. High-performance computing
- Need
  - to guarantee that a workload will always be able to cope with all demand levels, the workload is statically allocated enough resources so that peak demand is satisfied
- Server consolidation is an *approach to improve the low utility ratio of hardware resources by reducing the number of physical servers.*

Server Consolidation in Data Centers

- use of VMs increases resource management complexity
- Challenge
  - how to improve resource utilization as well as guarantee QoS in data centers
- server virtualization has the following effects
  - *enhances hardware utilization* - Many underutilized servers are consolidated into fewer servers
  - *enables more agile provisioning and deployment of resources* - guest OSes and their applications are readily cloned and reused
  - *total cost of ownership is reduced* - purchases of new servers, a smaller data-center footprint, lower maintenance costs, and lower power, cooling, and cabling
  - *approach improves availability and business continuity* - crash of a guest OS has no effect on the host OS and virtual servers are unaware of the underlying hardware

The End…