Lecture 3 : Hypervisors / Virtual Machine Monitors

Dr. Bibhas Ghoshal

Assistant Professor

Department of Information Technology

Indian Institute of Information Technology Allahabad

Virtual Machine (VM)

- VM - Isolated environment that appears to be a whole computer, but actually only has access to a portion of the computer resources
- Each VM appears to be running on the bare hardware
  - **Appearing as multiple instances of the same hardware**
- Process VM- a virtual platform created for an individual process and destroyed once the process terminates
- System VM - supports an OS along with many user processes

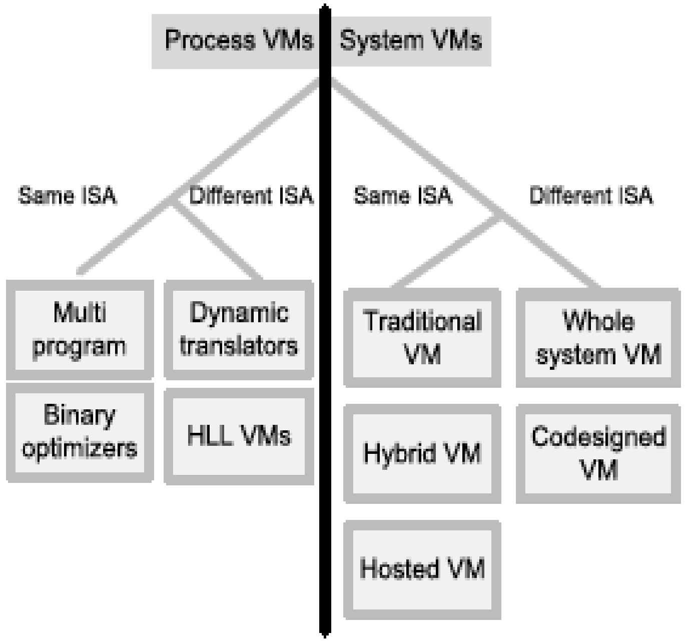
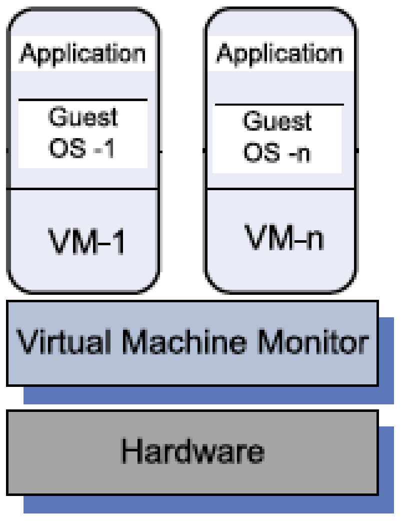
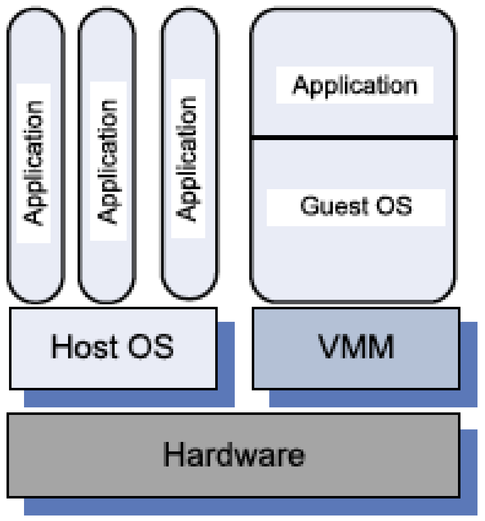


Figure Source : Cloud Computing, Theory and Practice, Dan Marinescu, 2nd edition, 2018

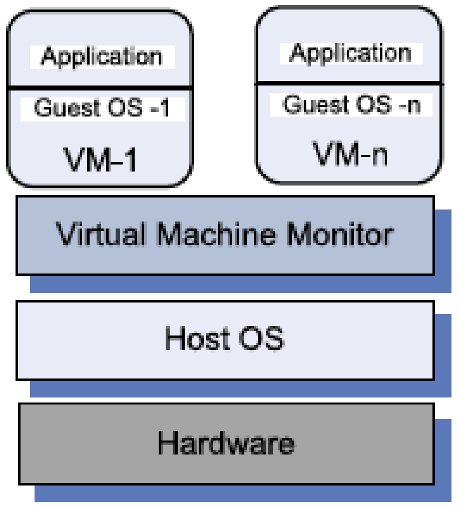
Classes of VM with same ISA



Traditional VM



Hybrid VM



Hosted VM

- Traditional VM - supports multiple virtual machines and runs directly on the hardware.
- Hybrid VM - shares the hardware with a host operating system and supports multiple virtual machines.
- Hosted VM - runs under a host operating system.

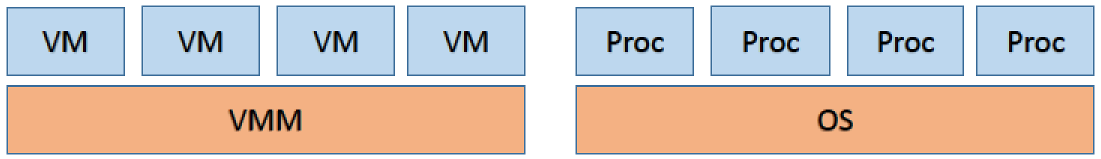
Figure Source : Cloud Computing, Theory and Practice, Dan Marinescu, 2nd edition, 2018

VMM Virtualizes CPU and Memory

- Traps privileged instructions executed by a guest OS and enforces operation correctness and safety.
- Traps interrupts and dispatches them to the individual guest operating systems.
- Controls virtual memory management.
- Maintains a shadow page table for each guest OS and replicates any modification made by the guest OS in its own shadow page table. This shadow page table points to the actual page frame and it is used by the Memory Management Unit (MMU) for dynamic address translation.
- Monitors the system performance and takes corrective actions to avoid performance degradation. For example, the VMM may swap out a Virtual Machine to avoid thrashing

How does the VMM Work?

- Multiple VMs running on a PM –multiplex the underlying machine
- Similar to how OS multiplexes processes on CPU



VMM performs machine switch (much like context switch)- run a VM for a bit, save context and switch to another VM, and so on…

What is the problem?

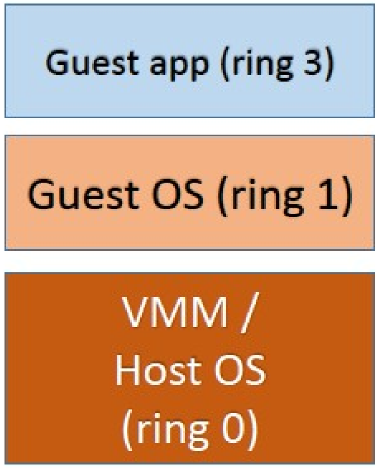
- Guest OS expects to have unrestricted access to hardware, runs privileged instructions, unlike user processes but one guest cannot get access. It must be isolated from other guests

Slide Author : Mythilli Vutukuru, IIT Bombay

VMM Working Principle : Trap and Emulate

All CPUs have multiple privilege levels Ring 0,1,2,3 in x86 CPUs

- User process - Ring 3 while OS in Ring 0
- Privileged instructions only run in ring 0
- Now, User process in Ring 3, VMM/host OS in ring 0



- Guest OS must be protected from guest apps but not fully privileged like host OS/VMM – Guest OS runs in Ring 1

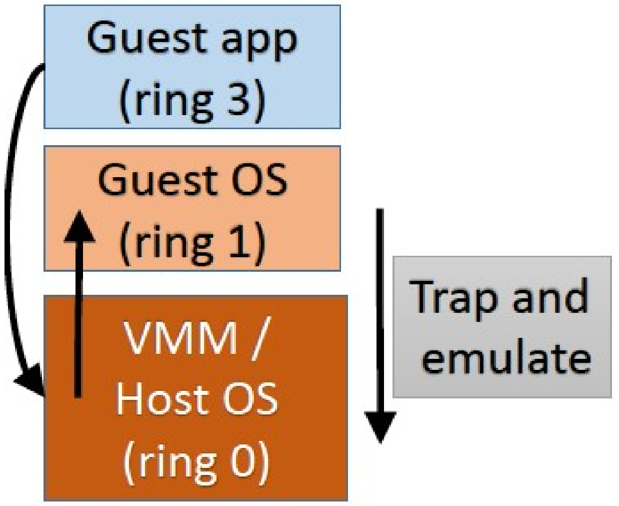
- Trap-and-emulate VMM: Guest OS runs at lower privilege level than VMM, Traps to VMM for privileged operation

Slide Author : Mythilli Vutukuru

Trap and Emulate

Guest app has to handle syscall/interrupt

- Special trap instr(intn), traps to VMM
- VMM doesn't know how to handle trap
- VMM jumps to guest OS trap handler
- Trap handled by guest OS normally
- Guest OS performs return from trap



Privileged instruction traps to VMM

- VMM jumps to corresponding user process
- Any privileged action by guest OS traps to VMM, emulated by VMM

  Example: set IDT, set CR3, access hardware
- Sensitive data structures like IDT must be managed by VMM, not guest OS

Slide Author : Mythilli Vutukuru

Limitations of Trap and Emulate Procedure

OS behaves incorrectly in ring1, will not trap to VMM

Why ???

- Some x86 instructions which change hardware state (sensitive instructions) run in both privileged and unprivileged modes - Will behave differently when guest OS is in ring 0 vs in less privileged ring 1

How does the know OS realize it is running at lower privilege level

- Some registers in x86 reflect CPU privilege level (code segment/CS)
- Guest OS can read these values and get offended!

Why these problems?

- OSes not developed to run at a lower privilege level
- Instruction set architecture of x86 is not easily virtualizable(x86 wasn't designed with virtualization in mind)

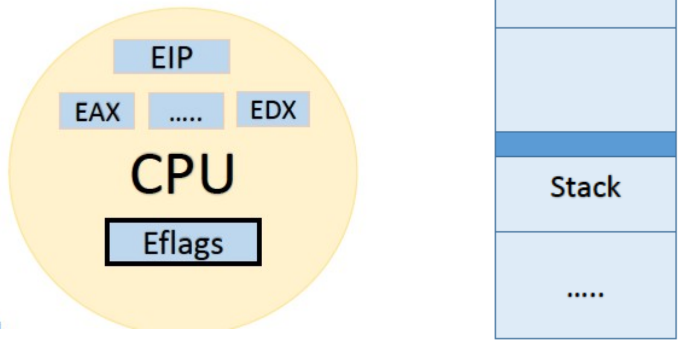
Slide Author : Mythilli Vutukuru

Limitations of Trap and Emulate
Example of Sensitive Instruction

- Eflags register is a set of CPU flags - IF (interrupt flag) indicates if interrupts on/off
- Consider the popf instruction in x86 - Pops values on top of stack and sets eflags
- popf executed in ring 0 - all flags set normally
- popf executed in ring 1 - only some flags set - IF is not set as it is privileged flag
- popf is a sensitive instruction, not privileged, does not trap,

behaves differently when executed in different privilege levels

- Guest OS is buggy in ring 1

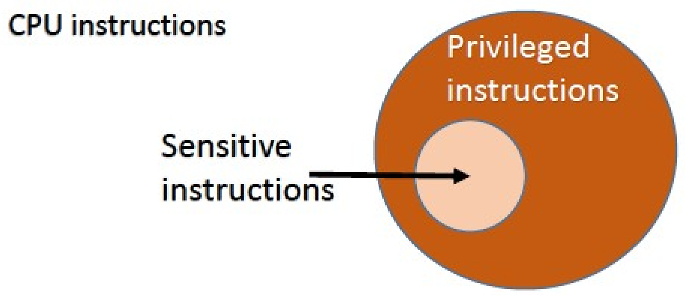


Slide Author : Mythilli Vutukuru

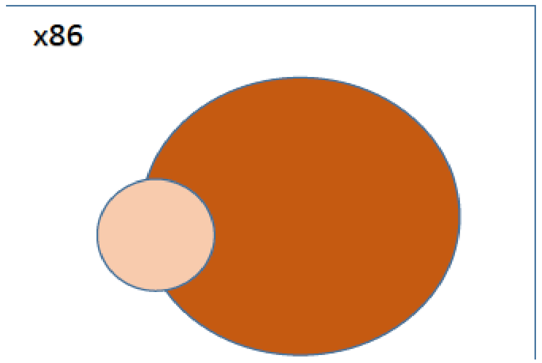
Popek Goldberg Theorem

In order to build a VMM efficiently via trap-and-emulate method, sensitive instructions should be a subset of privileged instructions

[Popek,Goldberg 1974]

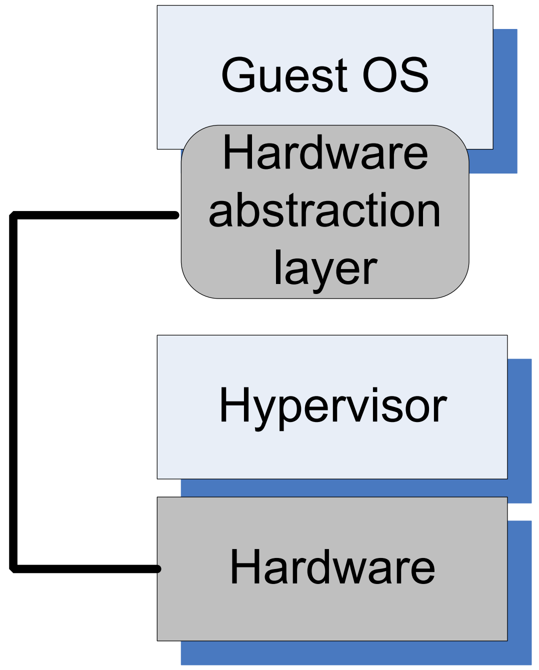


- x86 does not satisfy this criteria, so trap and emulate VMM is not possible

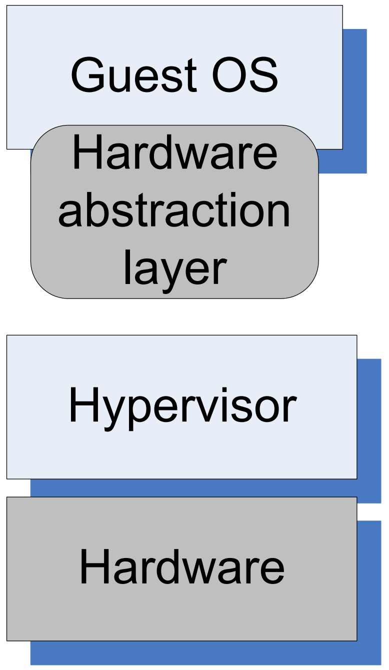


Slide Author : Mythilli Vutukuru

Types of Virtualization



(a) Full virtualization



(b) Paravirtualization

Figure Source : Cloud Computing, Theory and Practice, Dan Marinescu, 2nd edition, 2018

Virtualization Techniques

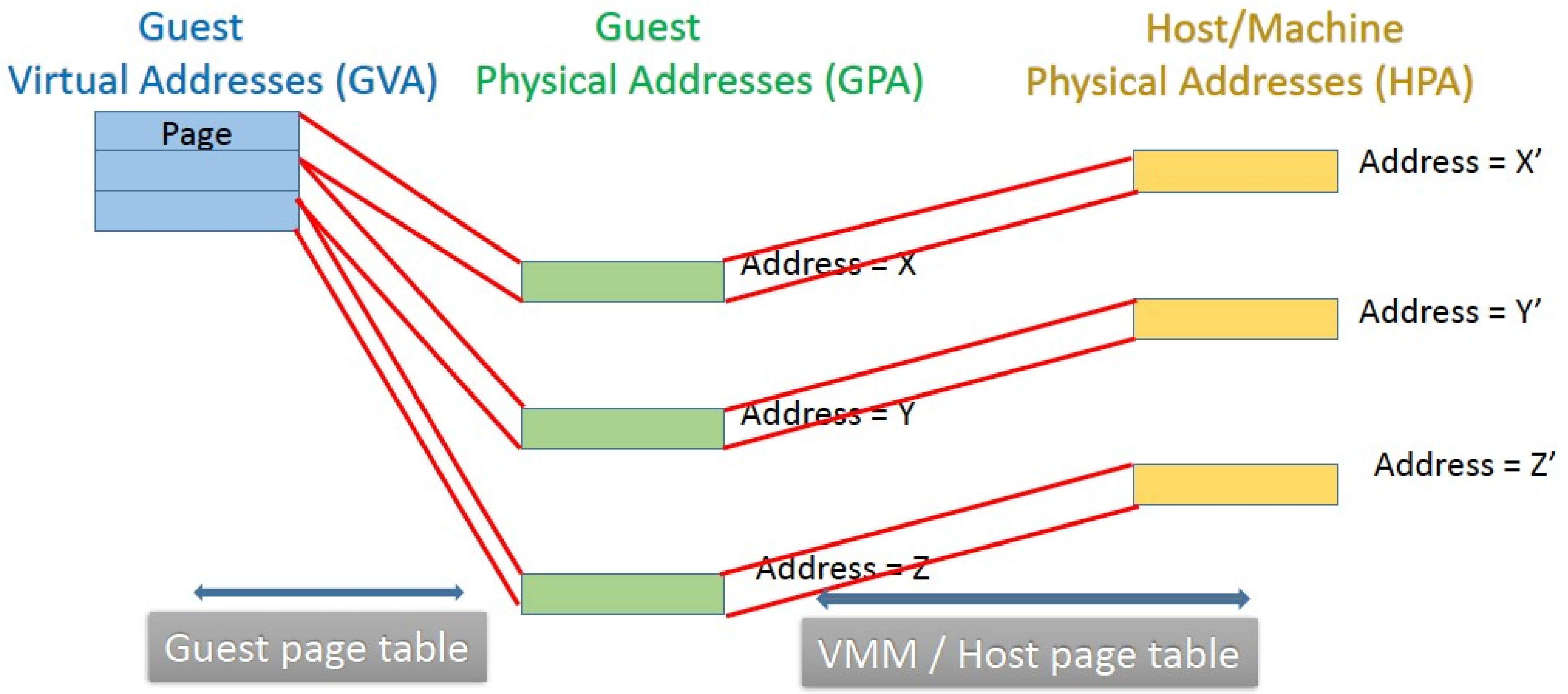
Full virtualization: CPU instructions of guest OS are translated to be virtualizable

- Sensitive instructions translated to trap to VMM
- Dynamic (on the fly) binary translation, so works with unmodified OS
- Higher overhead than para virtualization
- Example: VMWare workstation

Para virtualization: Rewrite guest OS code to be virtualizable

- Guest OS won’t invoke privileged operations, makes “hypercalls” to VMM
- Needs OS source code changes, cannot work with unmodified OS
- Example: Xen hypervisor

Memory Virtualization Techniques

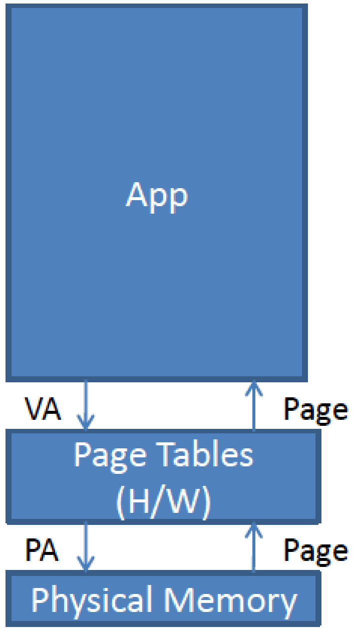


Slide Author : Mythilli Vutukuru

Memory Virtualization

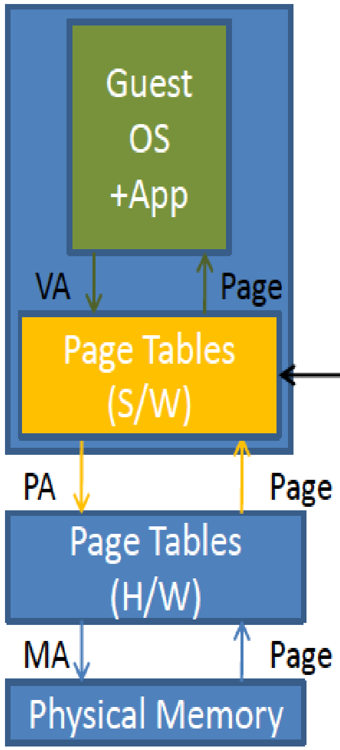
- OS is responsible for setting up and switching between page tables for different processes
- Hardware implements fast table look up using Translational Look Aside Buffer
- Without TLB support, each memory read would require 3-5 extra memory access to look up Page tables
- Two levels of Translation :

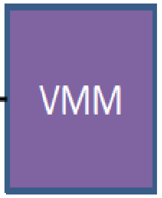
  Guest VA → Guest PA → Host PA
- Guest VA : Guest Virtual Address
- Guest PA : Guest Physical Address
- Host PA : Host Physical Address



Slide Author : Sorav Bansal

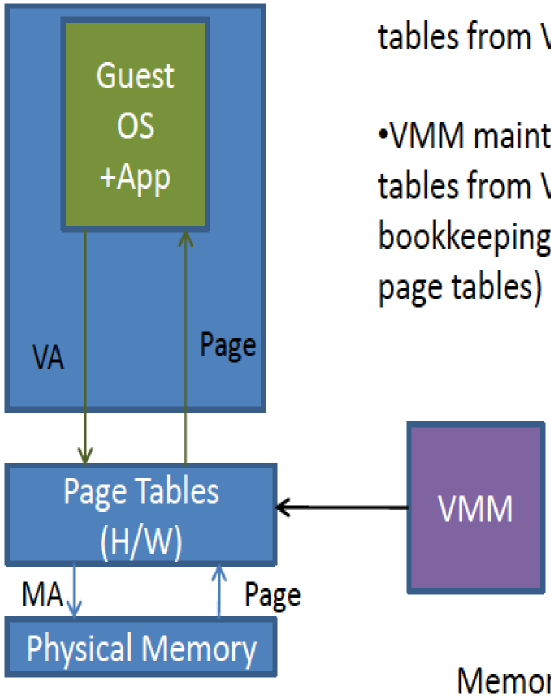
Memory Virtualization





VMM sets up the translation tables from VA to PA

SLOW!!
Each memory access needs to be translated in software ☹



- VMM sets up the translation tables from VA → MA directly.
- VMM maintains separate page tables from VA→PA to perform bookkeeping (also called shadow page tables)

Memory access is at near-native speed ☺

Slide Author : Sorav Bansal

Memory Virtualization Techniques

Guest page table has GVA→GPA mapping

- Each guest OS thinks it has access to all RAM starting at address 0

VMM / Host OS has GPA→HPA mapping

- Guest "RAM" pages are distributed across host memory

Shadow Paging : VMM creates a combined mapping GVA→HPA and MMU is given a pointer to this page table

- VMM tracks changes to guest page table and updates shadow page table

Extended page tables (EPT): MMU hardware is aware of virtualization, takes pointers to two separate page tables

- Address translation walks both page tables

I/O Virtualization Techniques

Guest OS needs to access I/O devices, but cannot give full control of I/O to any one guest OS

Two main techniques for I/O virtualization:

- Emulation : Guest OS I/O operations trap to VMM, emulated by doing I/O in VMM/host OS
- Direct I/O : Assign a slice of a device directly to each VM

- Many optimizations exist, active area of research