Chapter 16: Virtual Machines

CSCI 3753 Operating Systems

Prof. Rick Han

University of Colorado Boulder

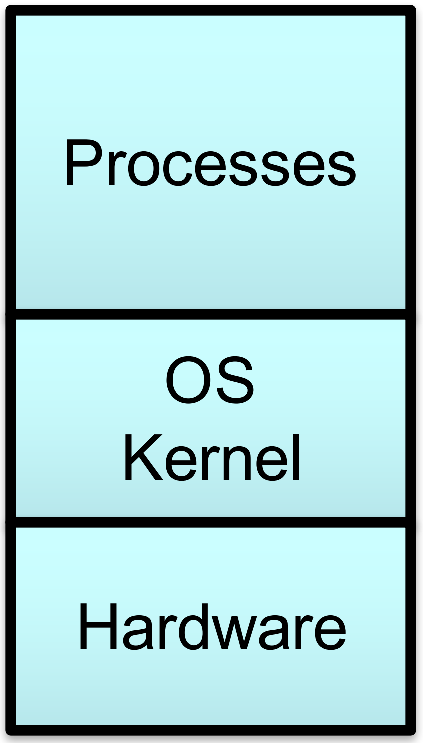
Virtual Machines

- A process already is given the illusion that it has its
  - Own memory, via virtual memory
  - Own CPU, via time slicing
- Virtual machine extends this idea to give a process the illusion that it also has its own hardware
  - Moreover, extend the concept from a process to an entire OS being given the illusion that it has its own memory, CPU, and I/O devices

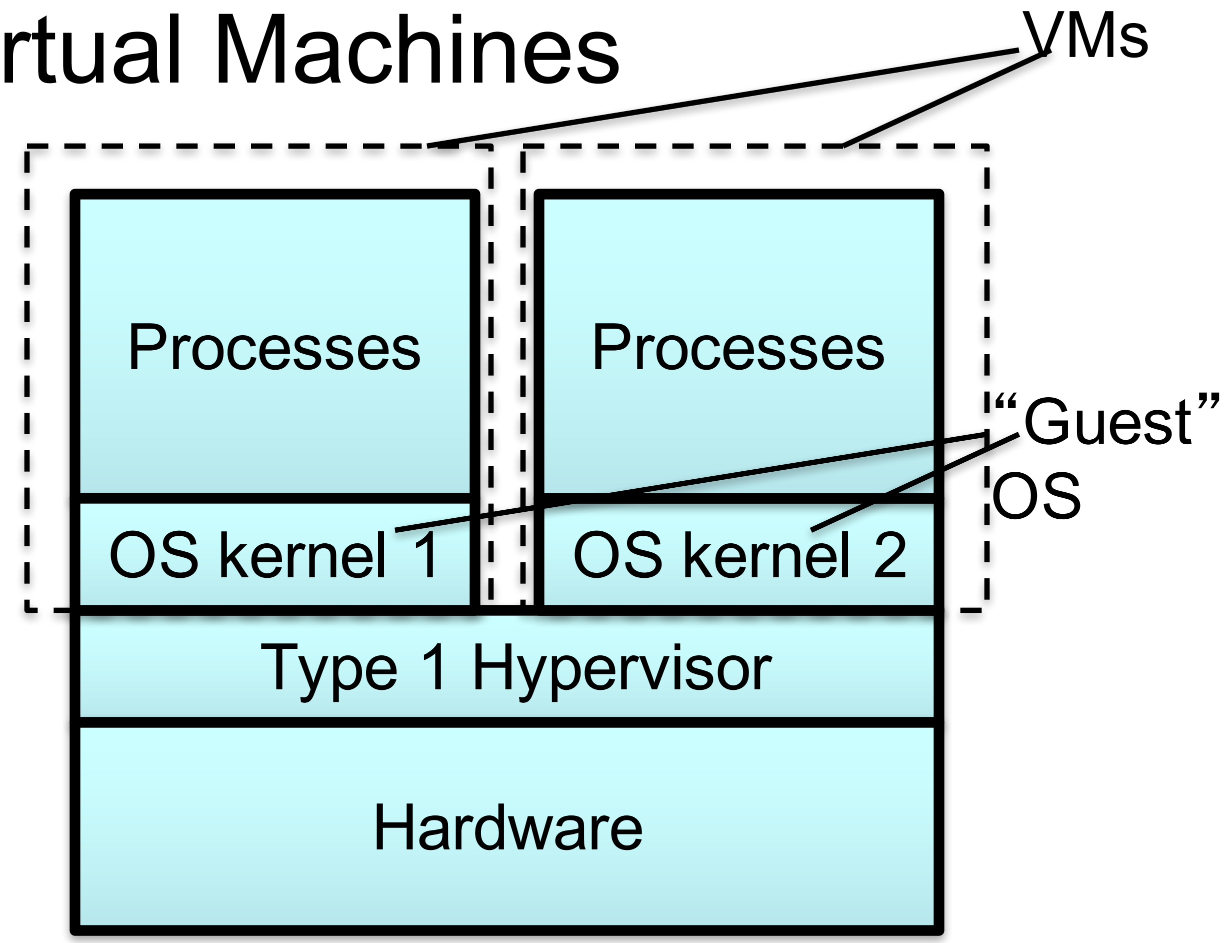
Virtual Machines

- Benefits include:
  - Can run multiple OS's simultaneously on the same host
  - Fault isolation if an OS fails – doesn't crash another VM. This is also useful for debugging a new OS.
  - Easier to deploy applications – can deploy an app within a VM instance that is customized for the app, rather than directly deploying the app itself and worrying about compatibility with the target OS – useful for cloud server deployments

Virtual Machines



Traditional OS



A Type 1 *Hypervisor* provides a virtualization layer for guest OSs and resides just above the hardware. A hypervisor is also called a *virtual machine monitor* (VMM).

University of Colorado **Boulder**

Virtual Machines

- How it basically works:
  - Goal: want to create a virtual machine that executes at close to native speeds on a CPU, so emulation and interpreting instruction by instruction are not good options – too much software overhead
  - Solution: have the guest OS execute normally, directly on the CPU, except that it is not in kernel mode. Therefore, any special privileged instructions invoked by the guest OS will be trapped to the hypervisor, which is in kernel mode.
  - The hypervisor then emulates only these privileged instructions and when done passes control back to the guest OS, also known as a "VM entry"
  - This way, most ordinary (non-privileged) instructions operate at full speed, and only privileged instructions incur the overhead of a trap, also known as a "VM exit", to the hypervisor/VMM.
  - This approach to VMs is called *trap-and-emulate*

Virtual Machines

- How it basically works:
  - Intel CPUs support 4 modes. In this case, the hypervisor is typically configured to execute in kernel mode (ring 0), the guest OS to execute in ring 1 (or 3), and user-mode processes to execute in ring 3.
    - In CPUs with only kernel and user modes, the VMM executes in kernel mode, and both the guest OS and its processes would execute in user mode.
  - Trap-and-emulate gives the impression to the guest OS that it commands the machine, but in reality underneath it the VMM is in charge

Virtual Machines

- Handling paging and virtual memory is tricky:
  - Want address translation virtual->physical to be handled at native speed by the hardware MMU, rather than in software for each instruction
  - Also want to preserve VMM's role as the controller of all memory allocation, so guest OS's are isolated from each other in memory

Virtual Machines

- Solution: Shadow page tables
  - In order to execute natively on the MMU, the guest OS's virtual addresses (VA) must be translated to actual physical pages in memory controlled by the VMM (we'll call these machine addresses MA), but the page tables of the guest OS contain the mapping of VA -> virtualized guest OS "physical" addresses (PA)

Virtual Machines

- Solution: Shadow page tables
  - VMM creates a shadow page table for each of the guest OS’s page tables. It is a composite of the mappings:
    - VA->PA (obtained from guest OS)
    - PA->MA (this mapping represents the VMM knowing which pages have been given to which guest OSs, and having to remap the guest OS’s “physical” pages to real machine pages so that the guest OS doesn’t grab memory pages given to other guest OSs)
    - The composite is VA->MA, which is stored in the shadow page table used by the MMU

Virtual Machines

- Solution: Shadow page tables
  - slows down memory management operations with extra software overhead
  - When your guest OS or process needs a new page, causing a page fault, both the shadow page table as well as the guest OS's page table need to be updated when a new page is chosen and/or evicted
  - Also, "When the virtual machine switches context from one process to another, the VMM must intervene to switch the physical MMU to the new process' shadow page table root."

Virtual Machines

- When there are multiple guest OSs:
  - VMM round-robins between them. When the timer interrupt fires, the VMM is invoked, and it chooses the next virtual machine (housing a guest OS & its processes) to run.
  - A challenge for the VMM is multiplexing the I/O requests from different guest OSs. VMM intercepts all interrupts from hardware. Decides which virtual machine to send an interrupt to, then starts the guest OS at the correct instruction to handle the interrupt, as if the guest OS had received the interrupt directly.
    - Each node is given the appearance of having its own virtual I/O devices

Virtual Machines

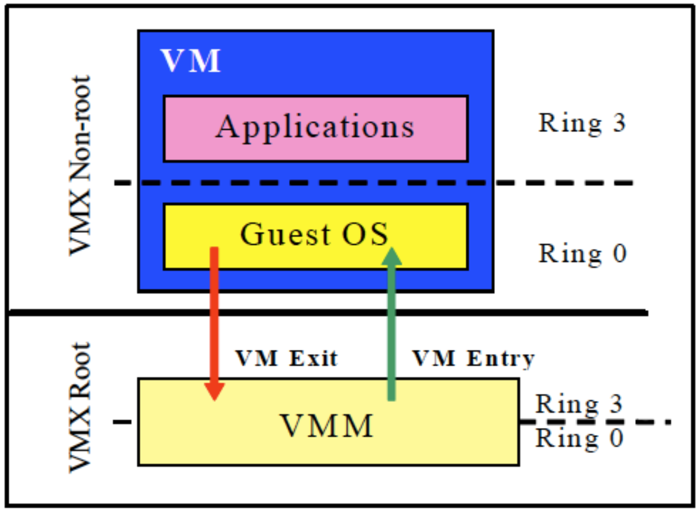
- *Full virtualization* is when the hypervisor provides illusion of identical hardware as the real system
  - Guest OS doesn't have to be recompiled, and doesn't even realize it's executing on a VM
  - Example VMs: Microsoft Virtual Server, VMWare ESX Server
  - Disadvantage:
    - Virtualized applications can run much slower, because of the many layers of fully virtualized abstraction between them and the hardware

Virtual Machines

- Demoting the privilege of the guest OS to non-kernel mode can cause excessive faulting (by privileged instructions), so VMWare ESX Server does just-in-time binary translation. VirtualBox does something similar.
  - "the binary translator replaces privileged instructions with sequences of instructions that perform the privileged operations in the virtual machine rather than on the physical machine."
    - For example, the privileged instruction would be replaced a jump instruction to a snippet of code that executes a sequence of safe instructions that emulate the desired privileged instruction but are safe to run at that ring/mode
  - JIT binary translation only occurs the first time the privileged instruction is called, and the snippet is cached thereafter, thus incurring translation overhead only once.

Virtual Machines

- Hardware assistance:
  - Intel and AMD came up with hardware support for virtualization, e.g. Intel's VT-x (and VT-i) and AMD's AMD-V
    - "VT-x consists of a set of virtual machine extensions (VMX) that support virtualization of processor hardware for multiple software environments using virtual machine."
    - e.g. can configure VirtualBox to run with hardware assistance or not.
  - Intel's VT-x introduced two new modes for virtualization: root mode and non-root mode.
    - The hypervisor executes in root mode, & guest OS executes in non-root mode in ring 0.
      - Benefit: this allows the guest OS to remain unchanged, because it typically runs in ring 0, unlike before where guest OS was modified to execute in ring 1



Virtual Machines

- Hardware assistance:
  - The VT-x enhancements ultimately mean that the guest OS trapped/faulted less to the VMM, and could execute more instructions safely inside of its virtual machine. This means faster performance.
    - Only those instructions that affected VMM state such as modifying page tables trap to the VMM
    - No longer need to do JIT binary translation, which rewrote those privileged instructions that it could to not fault. With the new hardware modes root and non-root, those privileged instructions that were faulting because they had been de-privileged will now no longer fault (they' re in ring 0, albeit non-root), so no need to rewrite them.

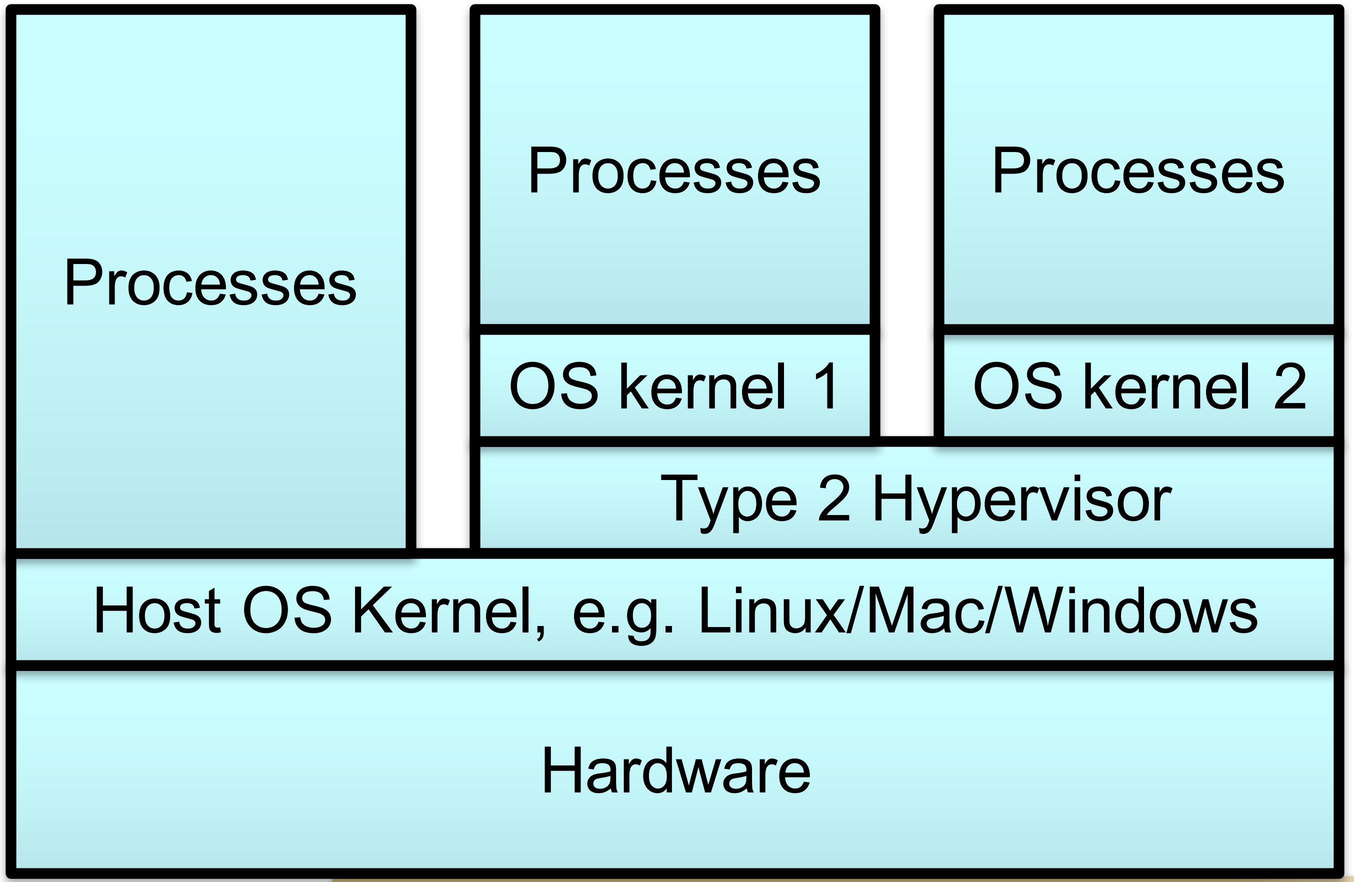
Virtual Machines

- Hardware assistance:
  - To address the software overhead of shadow page tables, Intel added hardware *Extended page tables* (AMD has an equivalent called Rapid Virtualization Indexing (RVI, formerly known as Nested Page Tables)) – each guest OS gets its own hardware support for its page tables:
    - "When this feature is active, the ordinary IA-32 page tables (referenced by control register CR3) translate from linear addresses to guest-physical addresses"
    - "A separate set of page tables (the EPT tables) translate from guest-physical addresses to the host-physical addresses that are used to access memory. *As a result, guest software can be allowed to modify its own IA-32 page tables and directly handle page faults*."
    - "This allows a virtual machine monitor (VMM) to avoid the VM exits associated with page-table virtualization, which are a major source of virtualization overhead without EPT."

Virtual Machines

- *Paravirtualization* tries to fix the performance issue of full virtualization another way:
  - An OS no longer sees full virtualization of hardware, but instead must be specially recompiled to work with this particular hypervisor, exploiting special optimizations to speed performance
  - In paravirtualization, guest OSs are modified to make special system calls (also known as hypercalls) directly into the hypervisor/VM monitor, say for I/O. Thus, you avoid the normal trapping overhead of the VMM. Instead, go directly to a VMM function for say reading/writing a file - it's much faster
  - Example VMs: Xen, Parallels, VMware Workstation

Virtual Machines



A Type 2 Hypervisor

Virtual Machines

- Type II Hypervisors
  - Essentially run like an application process on top of the host OS
  - Example VMs: VirtualBox, VirtualPC, VMWare Workstation/Server
  - More susceptible than type 1 hypervisors to security threats: if you can compromise the host OS, then can compromise the VMM and the guest OSs, whereas type 1 VMMs are typically much smaller and ostensibly more bug-free and secure

Virtual Machines

- Type II Hypervisors
  - VMM is a patched Linux kernel
  - Kernel modules, VirtualBox adds a driver/kernel module talks with user process

Virtual Machines

- Note the Type of a hypervisor describes the relationship between the VMM and what's underneath it, i.e. the hardware or host OS.
  - Full vs. para- virtualization describes the relationship between the VMM and what's above it, namely the guest OSs.
  - Thus, Xen can be both a Type 1 hypervisor and paravirtualized.

Virtual Machines

- Cloud Computing
  - Very easy to provision and deploy VM instances on the cloud
  - Amazon's Elastic Compute Cloud (EC2) uses Xen virtualization
    - There are different types of VMs or instances that can be deployed:
      - Standard, High-Memory, High-CPU
    - Users can create and reboot their own VMs
    - To store data persistently, need to supplement EC2 with an additional cloud service, e.g. Amazon's Simple Storage Service (S3)

Supplementary Slides

Java Virtual Machines

- Process VMs, e.g. Java VMs
  - Differ from System VMs in that the goal is not to try to run multiple OSs on the same host, but to provide portable code execution of a single application across different hosts
- Java applications are compiled into Java byte code that can be run on any Java VM
  - Java VM acts as an *interpreter* of byte code, translating each byte code instruction into a local action on the host OS

Java Virtual Machines

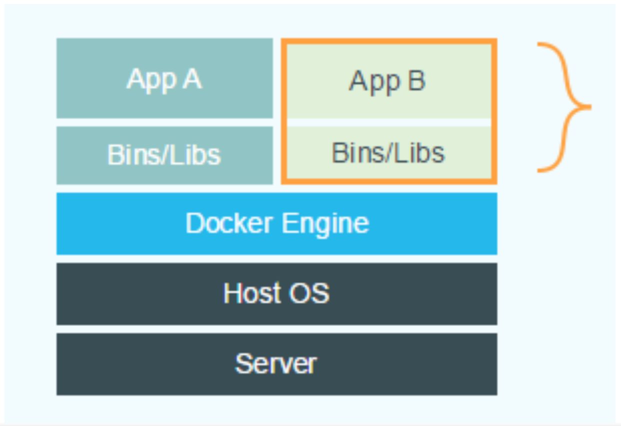
- Just in time compilation can be used to speed up execution of Java code
  - Java byte code is compiled at run time into native machine code that is executed directly on the hardware, rather than being interpreted instruction by instruction
- Note Java VMs virtualize an abstract machine, not actual hardware, unlike system VMs
  - i.e. the target machine that Java byte code is being compiled for is a software specification

Containers

- container virtualization isolates the guests,
- Instead of virtualizing the hardware,containers for each virtual environment.
- a patched kernel and user tools to run the virtual environments.
- The kernel provides process isolation and performs resource management.
- running under the same kernel, they effectively have their own filesystem, processes, memory, devices, etc.

Containers

- Docker
  - just the application and its dependencies.
  - isolated process in user-space on the host OS
  - Shares the kernel with others
  - resource isolation and allocation benefits of VMs portable and efficient.


App A
App B
Bins/Libs
Bins/Libs
Docker Engine
Host OS
Server