Lecture 14
Virtualization

Prof. Yinqian Zhang

Summer 2022

What is Virtualization

- Virtualization leverages a software component (i.e., the hypervisor) and some hardware support to create an abstraction layer over computer hardware
- It enables the hardware resources of a single computer (processors, memory, storage) to be multiplexed by multiple virtual machines (VMs).
- VMs run their own operating systems (OS) and operate as if they are independent computers, though they are using only a portion of the underlying computer hardware.

Application Scenarios of Virtualization

- **Workload isolation:** Virtualization can improve overall system **security** and **reliability** by isolating multiple software stacks in their own VMs.
  - Security: intrusions can be confined to the VM in which they occur
  - Reliability: software failures in one VM do not affect the other VMs.
- **Workload consolidation**: Corporate data centers are challenged by the proliferation of large numbers of heterogeneous and underutilized servers that run single-OS and single-application workloads
  - Virtualization makes it possible to consolidate individual workloads onto a single physical platform, reducing the total cost of ownership.
- **Workload migration:** By encapsulating a guest's state within a VM, it is possible to decouple the guest from the hardware on which it is currently running and to migrate it to a different platform.

Types of Virtualization

- Type-I hypervisor
  - Type 1 hypervisors are also known as bare-metal hypervisors, because they **run directly on the host's physical hardware**
- Type-II hypervisor
  - Type 2 hypervisors are also known as hosted hypervisors, because they are **installed on existing OSs**, and rely on them for virtualization and resource management

Popular VMM/Hypervisor

- Xen (type-I)
  - An open-source project originate from University of Cambridge Computer Laboratory
- KVM (type-I or type-II)
  - Kernel-based Virtual Machine is an open-source virtualization technology built into Linux
- VMware (type-I or type-II)
  - Products of an American cloud computing and virtualization technology company
  - VMware vSphere/ESXi is a type-I hypervisor, Vmware Player is type-II
- VirtualBox (Type-II)
  - x86 virtualization developed by Oracle Corporation.
- Hyper-V (Type-I)
  - Windows Server Virtualization developed by MS

Paravirtualization v.s. Full Virtualization

- Paravirtualization:
  - The guest OS kernel is aware that it runs in a virtual machine. Instead of simulating the CPU, physical memory and hardware devices, the guest kernel is modified to work with the hypervisor to virtualize the physical machine.
- Full virtualization:
  - Hardware is simulated to allow an unmodified guest OS to run in virtual machines.
  - Full virtualization includes software-based full virtualization and hardware-assisted full virtualization.

Hybrid Virtualization

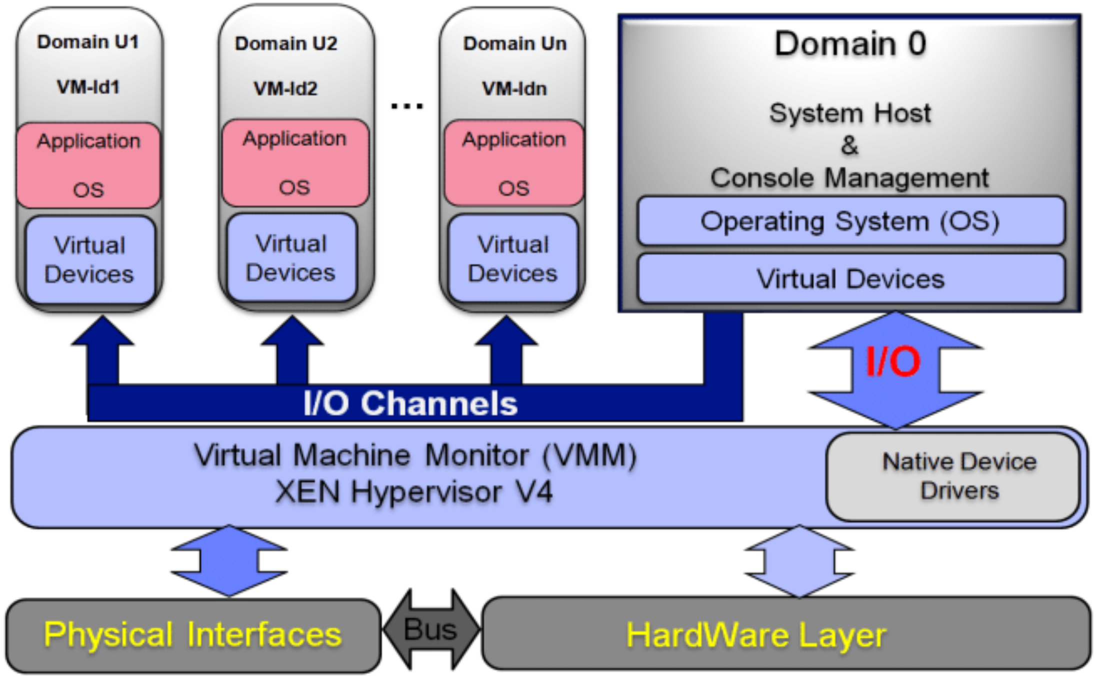
- Hybrid virtualization combines paravirtualization and hardware-assisted virtualization.
  - It can achieve equivalent or better performance than software-only para-virtualization, taking the full advantage of each technology.
  - The hybrid-virtualization employs para-virtualization for I/O, interrupt controllers, and timer to simplify the system and optimize performance.
  - For CPU virtualization, it allows one to use the same code as in the original kernel.

Full Virtualization

- Binary translation:
  - Binary translation is a technique of software-based full virtualization. Instructions in the guest OS that cannot be virtualized are translated into instructions of the host OS at runtime.
- Hardware-assisted virtualization:
  - Hardware-assisted virtualization eliminates the need of binary translation and paravirtualization.
  - It allows guest OS to run directly on the processor with a privilege level dedicated to guest kernels.
  - Techniques of hardware-assisted virtualization includes Intel VT-x and AMD-V.

Xen and the Art of Virtualization

- An x86 virtual machine monitor (VMM) that allows multiple commodity operating systems to share conventional hardware

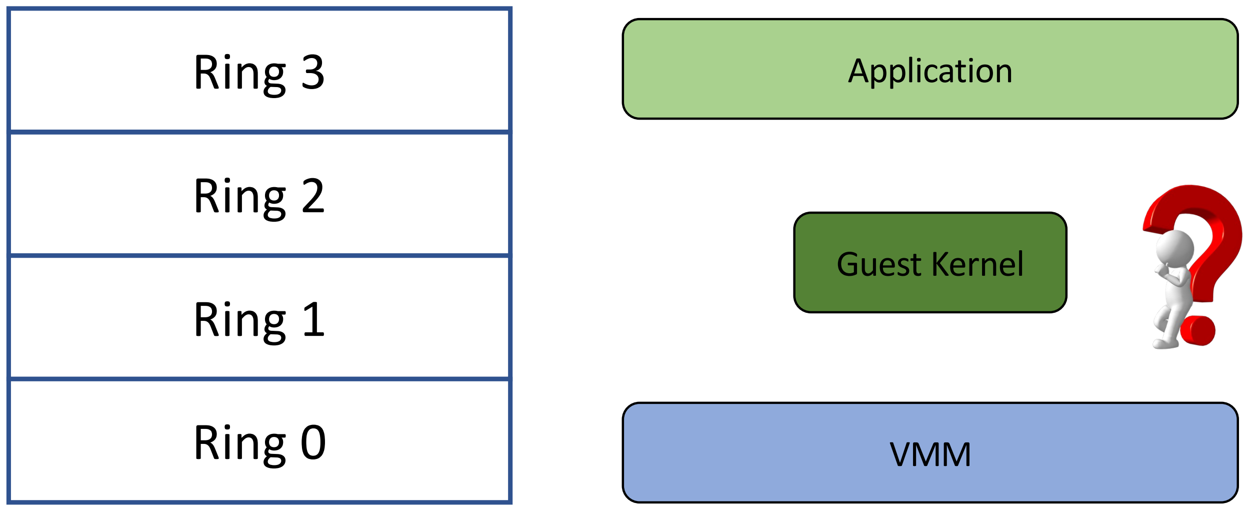


Privilege Levels

- X86 provides four privilege levels (rings).
  - Ring 0 is the highest privilege level
- OS runs in ring 0 and processes run in ring 1.
  - Instructions to access devices (e.g., IN/OUT) must run at a higher privilege level.
  - Instructions to update page table pointers (or other activities related to memory management) must run at a higher privilege level.
  - Instructions to register exception handlers (or other activities related to interrupt/exception handlers) must run at a higher privilege level.
  - Context switch must be performed at high privileged levels.

Ring Compression

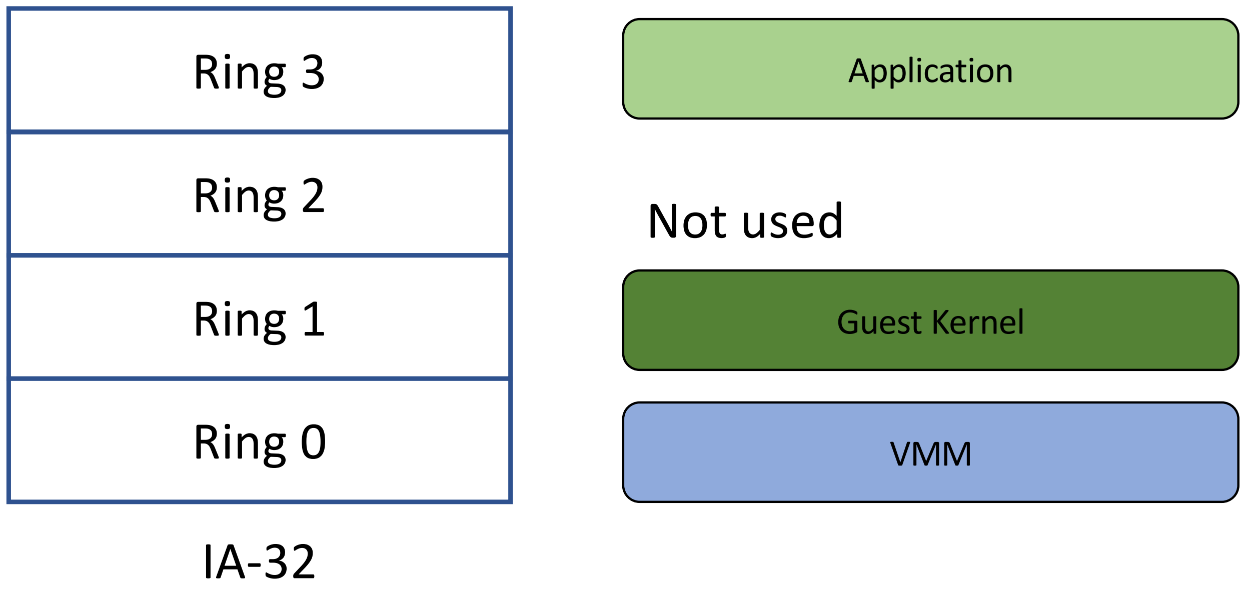
- If VMM runs in ring 0, where do we run the guest kernel?
- If the guest OS run at privilege level 3, the guest OS will run at the same privilege level as guest applications and will not be protected from them.



IA-32

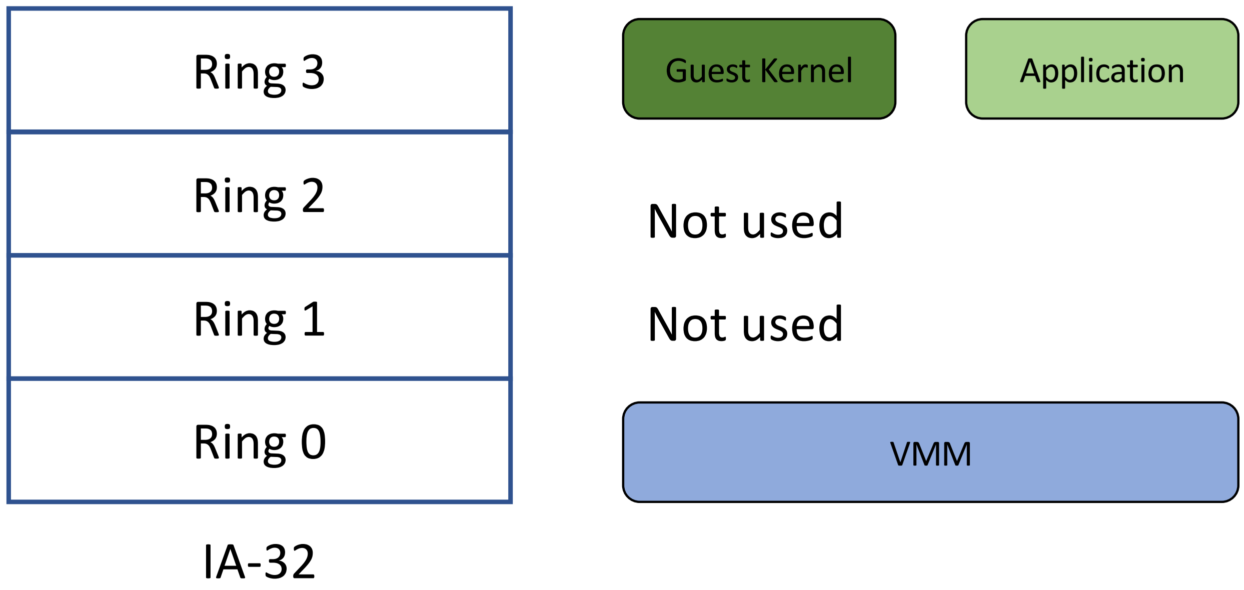
Xen's Solution for IA-32

- Guest OS kernel runs in ring 1.
  - Guest OS cannot directly execute privileged instructions
  - Safely isolated from applications running in ring 3.
- IA-32 and x86-64 do not protect ring 0 from ring 1 and ring 2
- VMM protected from guest kernel by segmentation



Xen's Solution for x86-64

- x86/64 provides very limited segment-level protection
- Both the guest kernel and applications in ring 3
- Run the guest kernel in a different address space (i.e., on different page tables) from its applications.



Xen's Solution for x86-64 (Cont'd)

- When forking a new process, the guest kernel creates two new page tables: one that is used in application context, and the other in kernel context.
- The kernel page table contains all the same mappings as the application page table but also include a mapping of the kernel address space.
- All transitions between application and guest-kernel contexts must pass via Xen, which automatically switches between the two page tables.

Hypercall

- Guest kernel runs in unprivileged level
- The hypercall interface allows domains to perform a synchronous software trap into the hypervisor to perform a privileged operation, analogous to the use of system calls in conventional operating systems.
- An example use of a hypercall is to request a set of page-table updates, in which Xen validates and applies a list of updates, returning control to the calling domain when this is completed.

Virtualizing Exceptions

- A table describing the handler for each type of exception is registered with Xen for validation.
  - Every exception will be checked by Xen to make sure that the handler's code segment does not specify execution in ring 0.
  - Since no guest OS can create such a segment, it suffices to compare the specified segment selector to a small number of static values which are reserved by Xen.
- Not many changes needed to the guest OS

Virtualizing Exceptions (Cont'd)

- The only modification is to the page fault handler
  - Page fault handler needs to read the faulting address from a privileged processor register (CR2)
    - Not possible when guest kernel is de-privileged
  - CR2 is written into an extended stack frame.
  - When an exception occurs while executing outside ring 0, Xen's handler creates a copy of the exception stack frame on the guest OS stack and returns control to the appropriate registered handler.

Virtualizing Interrupts

- IA-32 architectures provide mechanisms for masking external interrupts, preventing their delivery when the OS is not ready for them.
  - Interrupt flag (IF) in the EFLAGS register to control interrupt masking
- Xen VMM manages external interrupts and denies guest software the ability to control interrupt masking
  - As guest kernel is de-privileged, this can be done
- Such faulting can cause problems because some operating systems frequently mask and unmask interrupts
- Intercepting every guest attempt to do so could significantly affect system performance.

Virtualizing Interrupts (Cont'd)

- Whenever a device's interrupt line is asserted, it triggers execution of a stub routine within Xen rather than causing immediate entry into the guest
- Communication from Xen to a domain is provided through an asynchronous event mechanism,
  - Event replaces the usual delivery mechanisms for device interrupts
  - Event allows lightweight notification of important events such as domain-termination requests.
- Pending events are stored in a per-domain bitmask which is updated by Xen before invoking an event-callback handler specified by the guest OS.
- The callback handler is responsible for resetting the set of pending events, and responding to the notifications in an appropriate manner.
- A domain may explicitly defer event handling by setting a Xen-readable software flag: this is analogous to disabling interrupts on a real processor.

Memory Management for Virtualization

- Each OS normally thinks of physical memory as a linear array of pages, and assigns each page to itself or user processes.
  - The OS already virtualizes memory for its running processes, such that each process has the illusion of its own private address space.
- VMM adds another layer of virtualization, so that multiple OSes can share the actual physical memory of the machine.
  - This extra virtualization layer makes "physical" memory a virtualization on top of machine memory---the real physical memory of the system.
- An additional layer of indirection:
  - Each OS maps virtual-to-physical addresses via its per-process page tables;
  - The VMM maps the resulting physical mappings to underlying machine addresses via its per-OS page tables.

Page Tables

- Xen registers guest OS page tables directly with the MMU and restrict guest OSes to read-only access.
- Page table updates are passed to Xen via a hypercall; to ensure safety, requests are validated before being applied.
  - To aid validation, we associate a type and reference count with each machine page frame.
  - A frame may have any one of the following mutually-exclusive types at any point in time: page directory (PD), page table (PT), local descriptor table (LDT), global descriptor table (GDT), or writable (RW).
  - The type system is used to track which frames have already been validated for use in page tables.

Address Space Compression

- Operating systems expect to have access to the processor's full virtual-address space.
- A VMM must reserve for itself some portion of the guest's virtual-address space.
  - The VMM could run entirely within the guest's virtual-address space, which allows it easy access to guest data,
  - The VMM's instructions and data structures might use a substantial amount of the guest's virtual-address space.
  - Alternatively, the VMM could run in a separate address space, but even in that case the VMM must use a minimal amount of the guest's virtual-address space for the control structures that manage transitions between guest software and the VMM
- The VMM must prevent guest access to those portions of the guest's virtual-address space that the VMM is using
  - Guest attempts to access these portions of the address space must generate transitions to the VMM, which can emulate or otherwise support them.
- The term address-space compression refers to the challenges of protecting these portions of the virtual-address space and supporting guest accesses to them.

Xen's Solution to Address Space Compression

- Xen exists in a 64MB section at the top of every address space, thus avoiding a TLB flush when entering and leaving the hypervisor.
  - The top 64MB region of each address space, which is reserved for Xen, is not accessible or remappable by guest OSes.
  - This address region is not used by any of the common x86 ABIs however, so this restriction does not break application compatibility.
- Segmentation enforced by validating updates to hardware segment descriptor tables.
  - they may not allow any access to the Xen-reserved portion of the address space.

Xen's Physical Memory Management

- The initial memory allocation, or reservation, for each domain is specified at the time of its creation
- Memory is thus statically partitioned between domains, providing strong isolation.
- A maximum-allowable reservation may also be specified:
  - If memory pressure within a domain increases, it may then attempt to claim additional memory pages from Xen, up to this reservation limit.
  - If a domain wishes to save resources, it can reduce its memory reservation by releasing memory pages back to Xen.

Intel VT-x

- Intel virtualization technology on the x86 platform
- Virtual Machine Extensions (VMX)
  - Adds 13 new instructions: VMPTRLD, VMPTRST, VMCLEAR, VMREAD, VMWRITE, VMCALL, VMLAUNCH, VMRESUME, VMXOFF, VMXON, INVEPT, INVVPID, and VMFUNC.
  - These instructions permit entering and exiting a virtual execution mode where the guest OS perceives itself as running with full privilege (ring 0), but the host OS remains protected
- Extended Page Tables (EPT), a technology for page-table virtualization

VMX Root Mode

- VT-x augments IA-32 with two new forms of CPU operation: VMX root operation and VMX non-root operation.
  - A VMM runs in VMX root operation; it runs its guests in VMX non-root operation.
- Both forms of operation support all four privilege levels, allowing a guest OS to run at its intended privilege level and a VMM to use multiple privilege levels.
- Software running in VMX non-root operation is deprivileged in certain ways, regardless of privilege level.

Virtual Machine Control Structure

- The virtual-machine control structure (VMCS) is a new data structure that manages VM entries and VM exits and processor behavior in VMX non-root operations.
  - The VMCS is logically divided into sections, two of which are the guest-state area and the host-state area.
  - These areas contain fields corresponding to different components of processor state.
- VT-x defines two new transitions: a VM entry that transitions from Xen root operation to guest non-root operation, and a VM exit which does the opposite transition.
- VM entries load processor state from the guest-state area. VM exits save processor state to the guest-state area and then load processor state from the host-state area.

Interrupt Virtualization with Intel VT-x

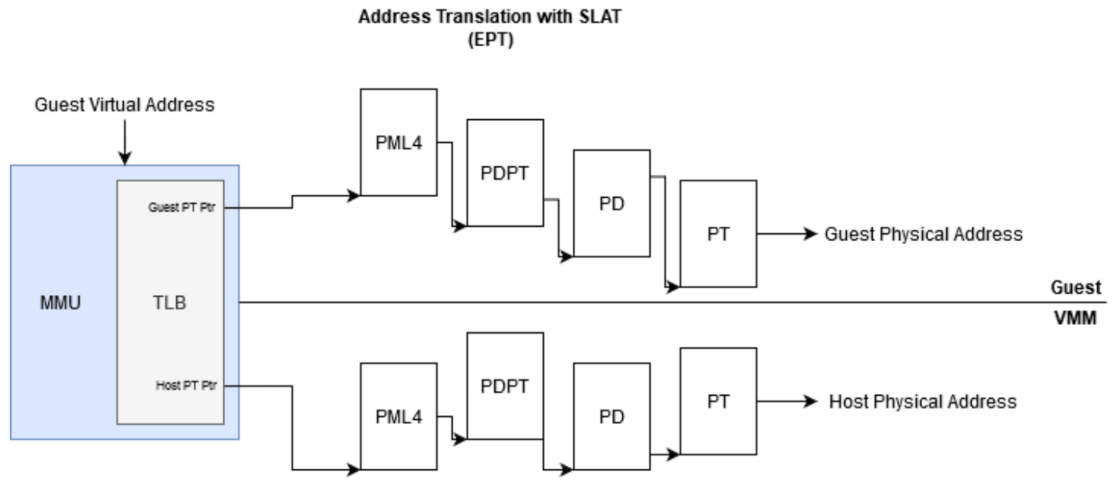
- The real local APICs and I/O APICs are owned and controlled by the Xen hypervisor.
  - All external interrupts will cause VM exits.
  - Interrupts owned by the hypervisor (e.g., the local APIC timer) are handled inside the hypervisor.
  - Otherwise the handler in Dom0 is used if the interrupt is not used by the hypervisor.
- The **HVM guests** only see virtualized external interrupts.
  - The device models can trigger a virtual external interrupt by sending an event to the interrupt controller (PIC or APIC) device model.
  - The interrupt controller device model then injects a virtual external interrupt to the HVM guest on the next VM entry.

Exception Virtualization with Intel VT-x

- Exceptions/faults, such as page fault, are intercepted as VM exits, and virtualized exceptions/faults are injected on VM entry to guests.
- VT-x supports exception bitmap, which contains 32 entries for the IA-32 exceptions.
- It allows a VMM to specify which exceptions should cause VM exits and which should not.
- Another bitmap allows per-port control of I/O instructions.

Intel EPT

- Extended page tables (EPT) was introduced in Intel Architecture based processors with Intel Virtualization technology enabled.
- When EPT is active the EPT based pointer loaded from the VMCS data structure points to extended page tables.
- This mechanism allows the Guest OSs full control of the Intel 64/IA-32 page tables which map guest virtual to guest-physical addresses.



Thank you!