

Platform Virtualization

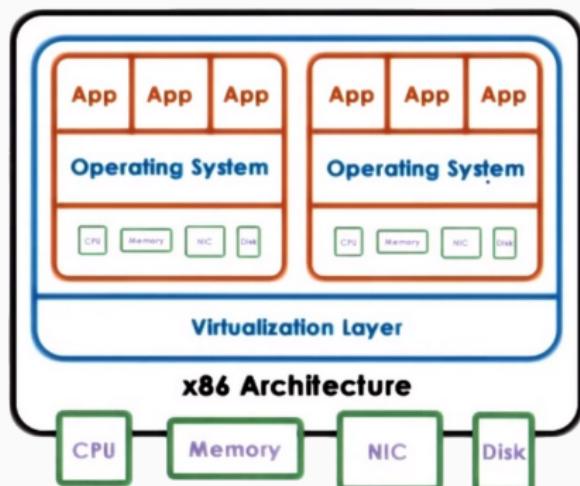
Florent Gluck - Florent.Gluck@hesge.ch

November 28, 2020

* Many thanks to Prof. Ada Gavrilovska for letting me use some of the contents of her course "Introduction to Operating Systems" thought at Georgia Institute of Technology

What is Platform virtualization?

- Virtualization of a **whole platform** → allows concurrent execution of multiple OSes on the same physical machine
- **Virtual machine (VM)** or guest domain = efficient, isolated duplicate of the real physical machine
- A VM is supported by a virtualization layer = **virtual machine monitor (VMM)** or **hypervisor**



Popek and Goldberg requirements for a VMM

In 1974, Popek and Goldberg provide three major requirements for a VMM:

1. **Fidelity:** provide an environment essentially identical to the original machine; **software on the VMM executes nearly identically to how it would on real hardware**

Popek and Goldberg requirements for a VMM

In 1974, Popek and Goldberg provide three major requirements for a VMM:

1. **Fidelity**: provide an environment essentially identical to the original machine; software on the VMM executes nearly identically to how it would on real hardware
2. **Performance**: to achieve good performance most instructions executed by the guest OS should be run directly on the underlying physical hardware → programs show at worst only minor decrease in speed

Popek and Goldberg requirements for a VMM

In 1974, Popek and Goldberg provide three major requirements for a VMM:

1. **Fidelity**: provide an environment essentially identical to the original machine; software on the VMM executes nearly identically to how it would on real hardware
2. **Performance**: to achieve good performance most instructions executed by the guest OS should be run directly on the underlying physical hardware → programs show at worst only minor decrease in speed
3. **Safety & isolation**: VMM is in complete control of system resources (hardware)

Quiz: virtualization technology

Based on the classical definition of virtualization by Popek & Goldberg, which of the following are virtualization technologies?

- Java Virtual Machine (JVM)
- Virtual GameBoy
- Oracle VirtualBox

Quiz: virtualization technology

Based on the classical definition of virtualization by Popek & Goldberg, which of the following are virtualization technologies?

- Java Virtual Machine (JVM)
- Virtual GameBoy
- Oracle VirtualBox

How to virtualize the machine?

- VMM must transparently virtualize the machine:
 - CPU
 - Memory
 - devices (Input/Output – I/O)
- Techniques to implement platform (hardware) virtualization:
 - **Full virtualization**
 - Trap-and-Emulate
 - Dynamic Binary Translation
 - Hardware-assisted
 - **Paravirtualization**

Platform virtualization: Full virtualization

- Guest OS completely **abstracted** from underlying hardware by the virtualization layer
- Guest OS **unaware** it's running on top of a hypervisor
→ Guest OS is **unmodified**
- Hypervisor translate all OS calls (syscalls) on-the-fly

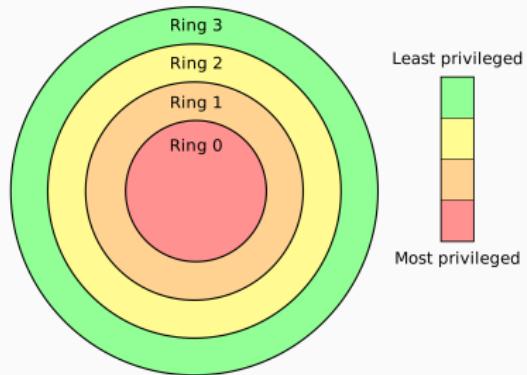
Platform virtualization: Paravirtualization

- By opposition to full virtualization, paravirtualization requires **modification** of the guest OS!
- **Better performance** by avoiding overhead/complexity required to support unmodified OSes
- Guest OS is **slightly modified**
- Guest OS **knows** it's running on top of a hypervisor
- For privilege operations, guest OS makes explicit calls to hypervisor → **hypercalls**

Platform virtualization: CPU virtualization

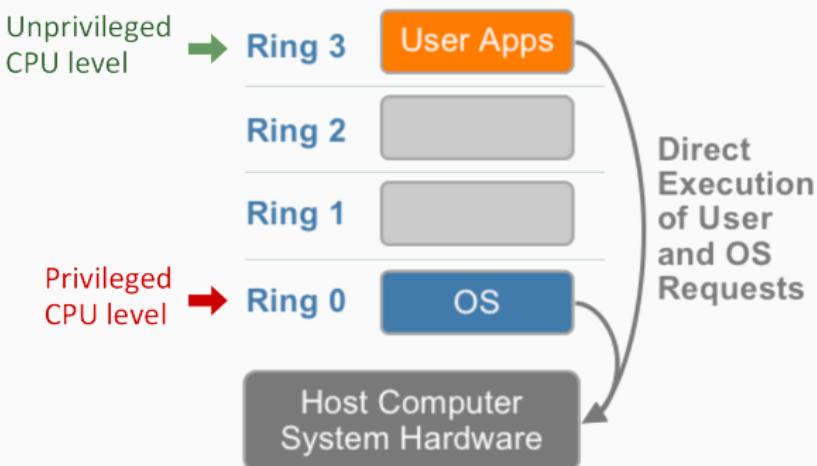
Hardware protection levels

- Commodity hardware has more than two **protection levels**
- For example, the original Intel 32-bit architecture has **four protection levels**, called **rings**
- **OS (kernel)** runs in **ring 0**
- **Applications** runs in **ring 3**



Direct execution without virtualization

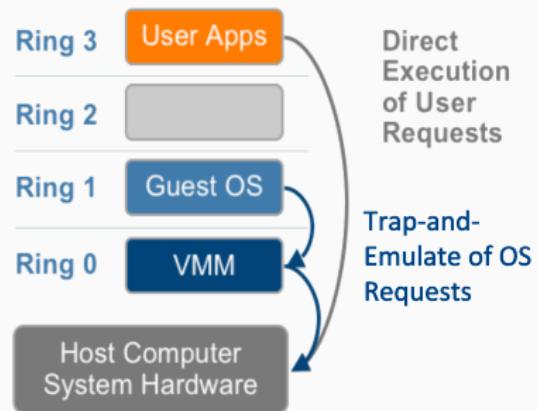
Application execution when executed on top of an OS (x86) without any virtualization:



Full virtualization: Trap-and-Emulate

Guest instructions are executed **directly** by the hardware:

- VMM does not interfere with every instruction/memory access issued by guest OS or applications
- For non-privileged operations → runs at hardware speed = **efficient**
- For privileged operations: **trap** to VMM
 - **If illegal operation:** **terminate** VM
 - **If legal operation:** **emulate** the behavior the guest OS was expecting from the hardware



Full virtualization: issue with Trap-and-Emulate

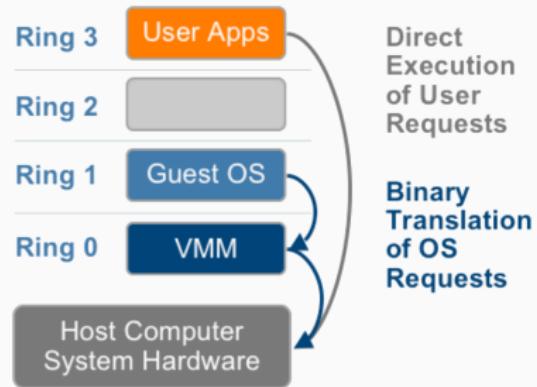
- Trap-and-Emulate worked well on mainframes
- In the 90s, when the need to apply virtualization to x86 architecture arose → Trap-and-Emulate model did NOT work!
- x86 pre-2005:
 - 17 privileged instructions do not trap → **Fail silently!** (doesn't pass control back to hypervisor)
 - E.g. interrupt enable/disable bit in privilege register; `pushf/popf` instructions that access it from ring3 fail silently
 - Hypervisor doesn't know, so doesn't try to change settings
 - Guest OS doesn't know, so assume change was successful!

Full virtualization: Binary Translation

- **Main idea:** rewrite the guest VM binary to never use those 17 instructions
- Pioneered by Mendel Rosenblum's group at Stanford, commercialized as VMWare in 1998
- **Goal:** to **not modify** the guest OS!

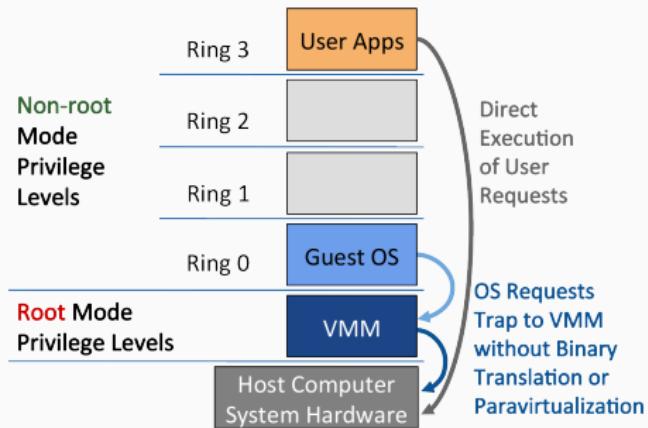
Full virtualization: Dynamic Binary Translation

- Dynamic because a sequence may depend on the runtime state
- Input dependent → can't be done statically
- Dynamically capture code blocks
- Inspect code blocks to be executed for the 17 instructions
- If needed, translate to alternate instruction sequence (to emulate desired behavior and avoiding traps)
- Otherwise → run at hardware speed
- Cache translated blocks to amortize translation costs



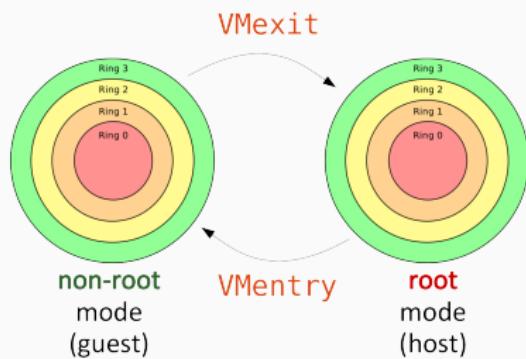
Full virtualization: Hardware-assisted

- Also called “Accelerated Virtualization” and “Hardware Virtual Machine” (HVM)
- Exists since the release of Intel-VT & AMD Pacifica (2005):
 - Close “holes” in x86 ISA (Instruction Set Architecture)
 - Adds new modes:
root/non-root = **hypervisor/guest**



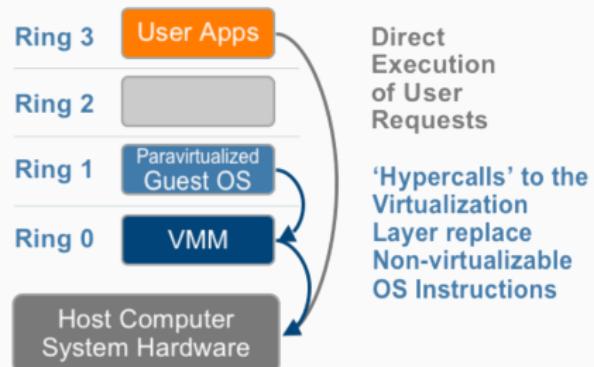
Hardware-assisted, root/non-root modes

- In **non-root mode**, certain not permitted operations cause traps (**VMExits**) → trigger switch to **root mode** (hypervisor)
- An hypervisor could use the following:
 - **Non-root** (guest): VMs
 - Ring 3: applications
 - Ring 0: OS
 - **Root** (host):
 - Ring 0: hypervisor



Paravirtualization

- Goal: better performance by avoiding overhead/complexity required to support unmodified OSes:
 - Guest OS is slightly modified
 - Guest OS knows it's running on top of a hypervisor
 - For privilege operations, guest OS makes explicit calls to hypervisor through hypercalls
 - Hypercalls from guest OS to hypervisor ≈ system calls from application to OS
- Paravirtualization popularized by Xen hypervisor from Citrix



Hypervisor models

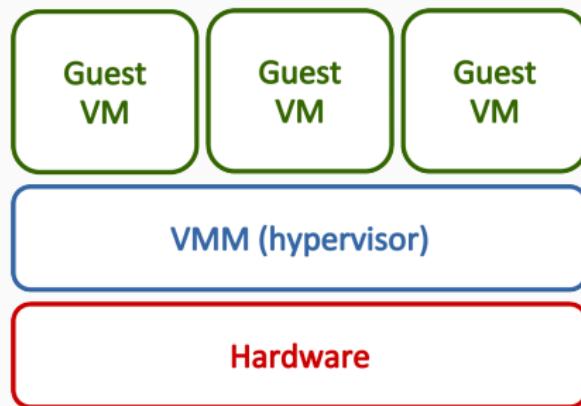
Two hypervisor models:

- **Bare-metal** (type 1)
- **Hosted** (type 2)

Bare-metal (type 1) virtualization model

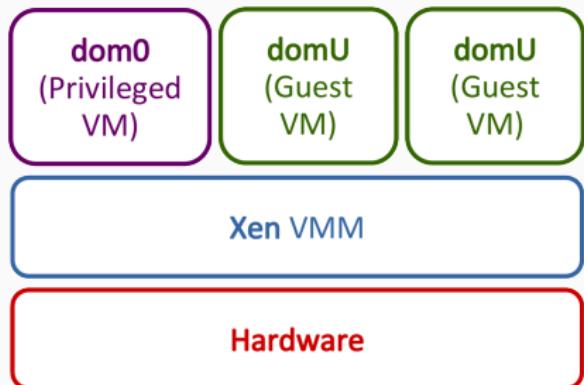
Bare-metal = hypervisor runs above hardware (metal)

- VMM manages all hardware resources and supports execution of VMs → hypervisor must manage all possible devices



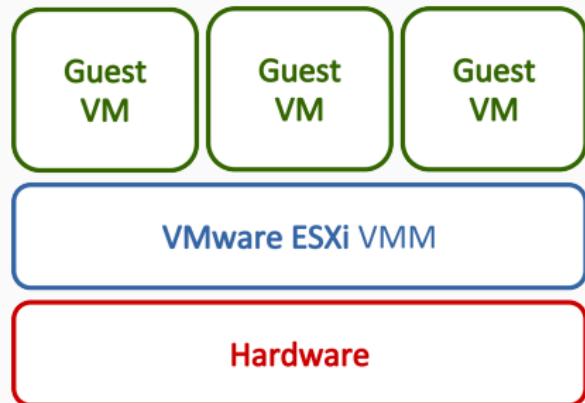
Bare-metal model, variant 1: Xen

- VMM manages all hardware resources and supports execution of VMs
- Integrates a **privileged, service VM**, that runs a standard OS with full hardware privileges
 - Run all device drivers
 - Configuration and management task (how VMM share resources across VMs)



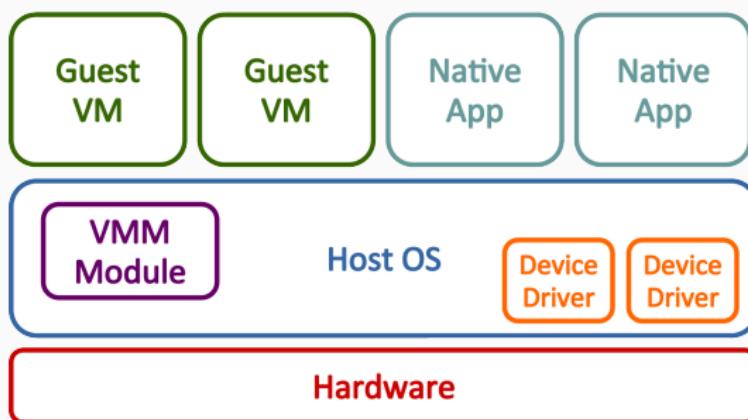
Bare-metal model, variant 2: VMware ESXi

- VMM manages all hardware resources and supports execution of VMs
- Device manufacturers provide device drivers for the hypervisor



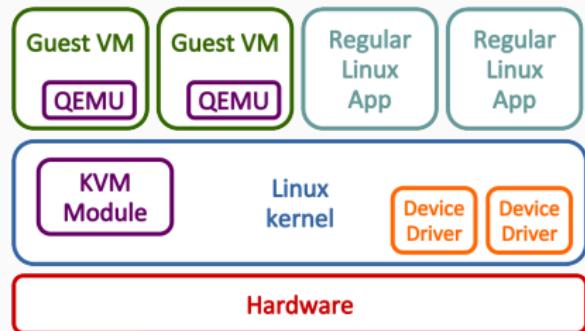
Hosted (type 2) virtualization model

- Host OS owns all hardware
- Special VMM module provides hardware interfaces to VMs and deals with VM context switching



Hosted virtualization example: KVM

- KVM (Kernel Virtual Machine) = Linux kernel module
- Linux host → provides hardware management + runs regular Linux applications
- Guest VMs support through KVM + QEMU (platform emulator + virtualizer) for hardware virtualization
- Massive Linux kernel re-use
- Leverages advances in Linux open-source community



Quiz: bare-metal or hosted

Among the following virtualization products, which ones are **bare-metal** (type-1) hypervisors and which ones are **hosted** (type-2)?

- KVM
- VMWare Fusion
- VirtualBox
- VMWare Player
- VMWare ESXi
- Citrix XenServer
- Microsoft Hyper-V
- Amazon Web Services

Quiz: bare-metal or hosted

Among the following virtualization products, which ones are **bare-metal** (type-1) hypervisors and which ones are **hosted** (type-2)?

- KVM
- VMWare Fusion
- VirtualBox
- VMWare Player
- VMWare ESXi
- Citrix XenServer
- Microsoft Hyper-V
- Amazon Web Services

Platform virtualization: Device virtualization

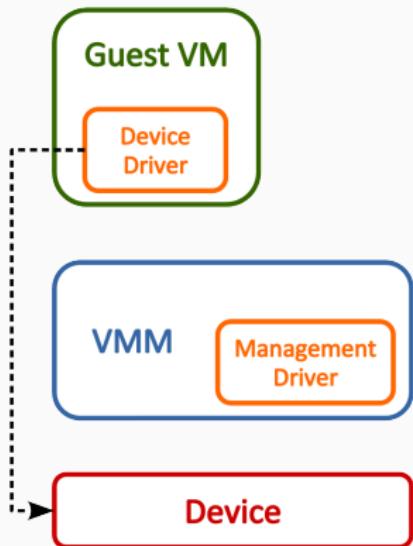
Device virtualization

- For CPU and memory → little diversity
 - ISA¹-level interface standardization
- For devices → great diversity
 - Lack of standardization
 - Specific device interface
 - Specific device behavior (semantic)
- Three key models for device virtualization (pre VT hardware)
 1. Passthrough
 2. Emulated (also called “hypervisor - direct”)
 3. Paravirtualized (also called “split - device”)

¹Instruction Set Architecture

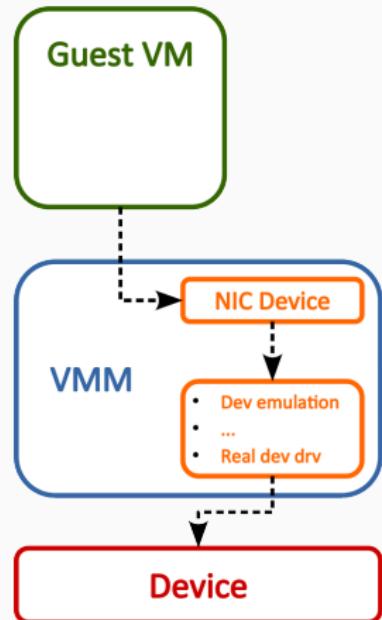
Device virtualization: Passthrough model

- VMM-level driver (management) configures device access permissions
- **Pros**
 - VM provided with exclusive access to the device
 - VM can directly access the device (VMM bypass)
- **Cons**
 - Device sharing is difficult
 - VMM must have exact type of device as what VM expects
 - VM migration difficult



Device virtualization: Emulated

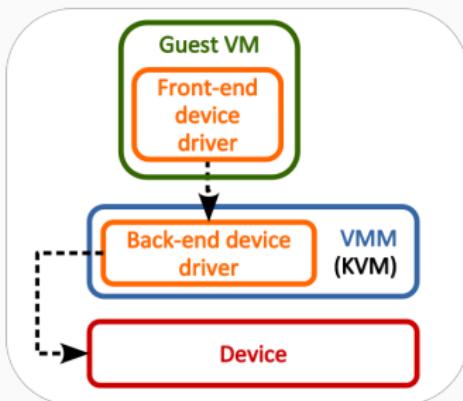
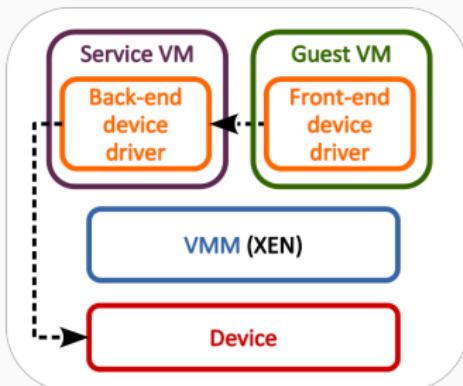
- VMM **intercepts** all device accesses
- **Emulate** device operation:
 - translate to generic I/O operations
 - traverse VMM-resident I/O stack
 - invoke VMM-resident driver
- **Pros**
 - VM decoupled from physical device
 - Sharing, migration, dealing with device specifics
- **Cons**
 - Latency
 - Device driver complexities in VMM



Device virtualization: Paravirtualized

- Also called “Split device driver model”
- Device access control **split** between:
 - **Front-end** driver in guest VM
 - **Back-end** driver in
 - service VM (e.g. XEN)
 - VMM (e.g. KVM)
 - Modified guest drivers

- **Pros**
 - Eliminate emulation overhead
 - Better management of shared devices
- **Cons**
 - Requires drivers on the Guest OS



x86 Intel-VT revolution

Intel® Virtualization Technology Evolution

Vector 3: IO Device Focus

Vector 2: Chipset Focus

Vector 1: Processor Focus

VMM Software Evolution

- Assists for IO sharing:
- PCI IOV compliant devs
 - VMDq: Multi-context IO
 - End-point DMA translation caching
 - IO virtualization assists

Core support for IO robustness & device assignment via DMA remapping

Interrupt filtering & remapping
VT-d extensions to track PCI-SIG IOV

Close basic processor "virtualization holes" in Intel® 64 & Itanium CPUs

Performance extensions VTx/VTi EPT, APIC-TPR, VPID, ECRR, APIC-V

Perf improvements for interrupt intensive env, faster VM boot

Software-only VMs
Binary translation
Paravirtualization
Device emulations

Simpler and more secure VMM through use of hardware VT support

Better IO/CPU perf and functionality via hardware-mediated access to memory

Richer IO-device functionality and IO resource sharing

Yesterday:
No HW Support

2005-2006
With CPU Support

2007-2008
With Chipset Support & IO improvements

x86 Intel-VT revolution

Intel® Virtualization Technology Evolution

Vector 3: IO Device Focus

Vector 2: Chipset Focus

Vector 1: Processor Focus

VMM Software Evolution

Assists for IO sharing:
• PCKIOV-compliant devs
• VMDq Multi-context IO
• End-point DMA translation
caching
• IO virtualization assists

Core support for IO
robustness & device
assignment via DMA
remapping

VT-d

Interrupt filtering &
remapping
VT-d extensions to
track PCI-SIG I/OV

VT-d2

Close basic
processor
"virtualization holes"
in Intel® 64 & Itanium
CPUs

VT-x

Performance
extensions VTx/VTi
EPT, APIC-TPR, VPID,
ECRR, APIC-X

VT-x2

Perf improvements
for interrupt intensive
env, faster VM context
switching

VT-x3

Software-only VMs
Binary translation
Paravirtualization
Device emulations

Simpler and more
secure VMM through
use of hardware VT
support

Better IO/CPU perf
and functionality via
hardware-mediated
access to memory

Richer IO-device
functionality and IO
resource sharing

Yesterday:
No HW Support

2005-2006
With CPU Support

2007-2008
With Chipset Support & IO improvements

Quiz: virtualization requirements

Which of the following are virtualization requirements?

- Present virtual platform interface to VMs
- Provide isolation across VMs
- Protect VMM from guest OS

Quiz: virtualization requirements

Which of the following are virtualization requirements?

- Present virtual platform interface to VMs
→ virtualize CPU, memory, devices, etc.
- Provide isolation across VMs
→ preemption, MMU for address translation and validation
- Protect VMM from guest OS
→ Cannot run guest OS and VMM at same protection level

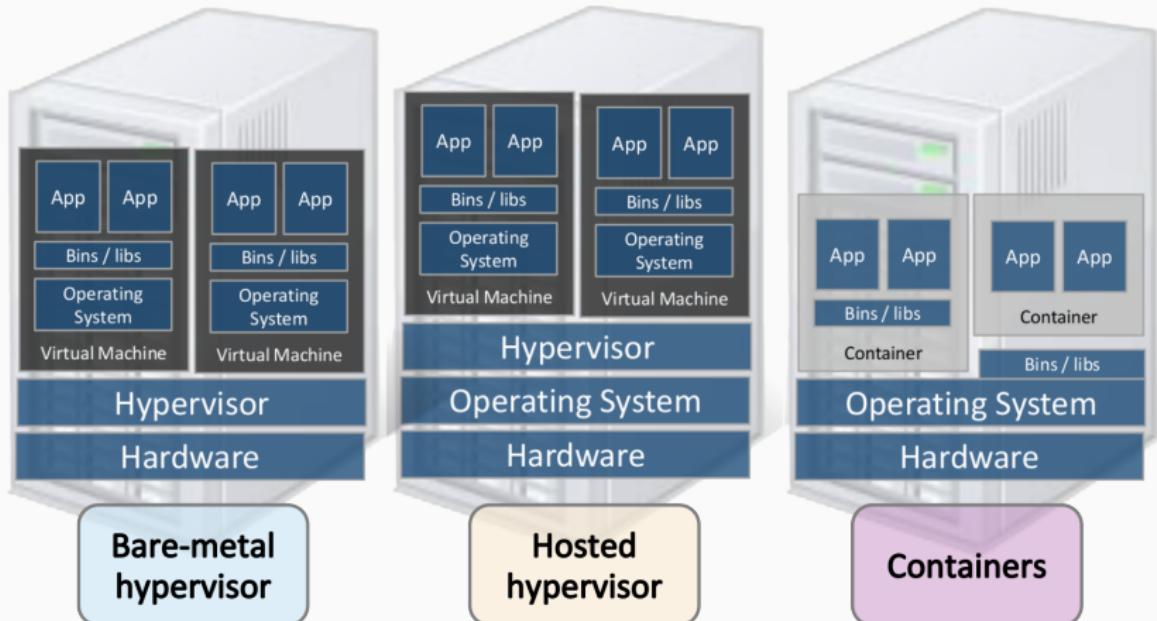
What's used out there?

- Most public hosted services are powered by KVM
 - Amazon AWS & EC2 → Xen → KVM (modified called Nitro)
 - Google Cloud Platform → KVM
 - Microsoft Azure → Microsoft Hyper-V
 - DigitalOcean → KVM
- Everything OpenStack → mostly powered by KVM
- VMware leader in solutions for business/private environments

Operating system virtualization

- OS-level virtualization = containerization
 - OS feature where kernel allows existence of multiple **isolated user-space instances** called containers, partitions, or jails
- Processes inside a container can **only** see the container's contents and devices assigned to the container → isolation
- Kernel provides **resource-management** features to limit impact of one container's activities on other containers
 - Example (Linux kernel): cgroups, namespaces, chroot
- Examples: Linux Containers (LXC), Docker, BSD jails, Solaris Zones, Linux OpenVZ

Platform virtualization vs OS virtualization



- Single-purpose appliances
 - Trend towards each VM = single application (instead of full OS)
 - Unikernel: virtual library OSes, stripped OSes to run a single specialized application
 - Create a specialized VM for a specific application
 - Reduce memory footprint
 - Increase security

Future trends

- Rise of containers
- Hybrid solutions:
 - containers inside hypervisors!



Resources

- “Hardware and Software Support for Virtualization”; E. Bugnion, J. Nieh, D. Tsafrir; Morgan & Claypool Publishers, 2017
- “Virtual Machines: Versatile Platforms for Systems and Processes”; J. Smith, R. Nair; Morgan Kaufmann, 2005
- “Introduction to Operating Systems” by Prof. Ada Gavrilovska, Georgia Institute of Technology
- “Operating Systems: Three Easy Pieces”; Remzi H. et Andrea C. Arpaci-Dusseau; Arpaci-Dusseau Books, 2020
- Virtualization in Xen 3.0:
<https://www.linuxjournal.com/article/8909>