

Platform Virtualization

Florent Glück - florent.gluck@hesge.ch

March 03, 2025

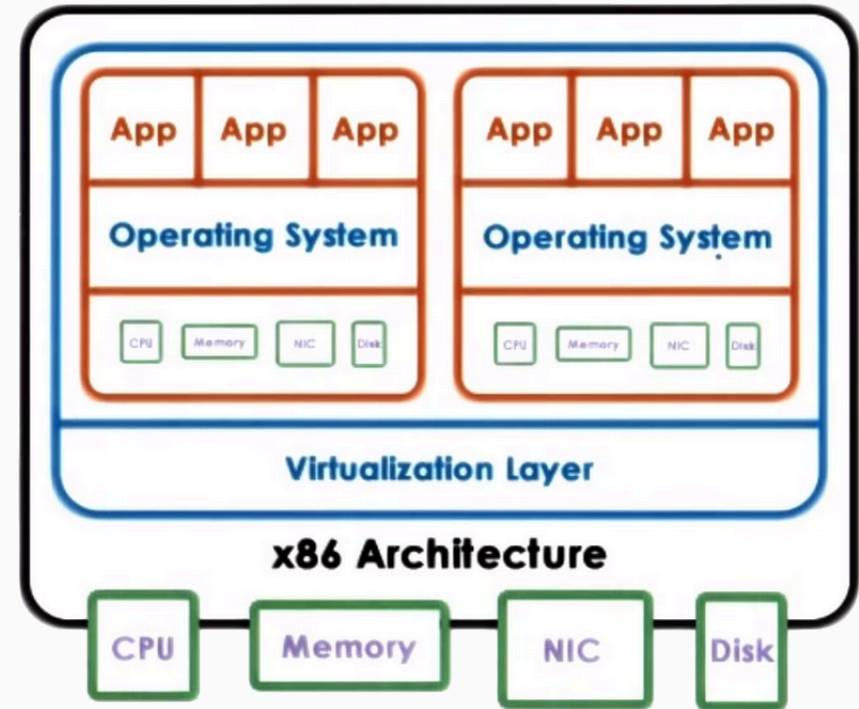
ISC - HEPIA

Thank you note

Thanks to Prof. Ada Gavrilovska at Georgia Institute of Technology for allowing me to use some of her “Introduction to Operating Systems” course’s content

What is Platform virtualization?

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



The OS running in the VM is called the **Guest OS**

The term “**Virtual Machine**” (VM) was originally defined by Popek and Goldberg in 1974:

“An efficient, isolated duplicate of a real computer machine”

Formal Requirements for Virtualizable Third Generation Architectures

Gerald J. Popek
University of California, Los Angeles
and
Robert P. Goldberg
Honeywell Information Systems and
Harvard University

Popek and Goldberg requirements for a VMM

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

- 1.

Popek and Goldberg requirements for a VMM

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

1. **Equivalence**: provide an environment essentially identical to the original machine
 - software on the VMM **executes nearly identically** to how it would on the **real hardware**
- 2.

Popek and Goldberg requirements for a VMM

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

1. **Equivalence**: provide an environment essentially identical to the original machine
 - software on the VMM **executes nearly identically** to how it would on the **real hardware**
2. **Efficiency**: most machine instructions must be executed by the guest OS without VMM intervention
 - **machine instructions run directly** on the underlying hardware
- 3.

Popek and Goldberg requirements for a VMM

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

1. **Equivalence**: provide an environment essentially identical to the original machine
 - software on the VMM **executes nearly identically** to how it would on the **real hardware**
2. **Efficiency**: most machine instructions must be executed by the guest OS without VMM intervention
 - **machine instructions run directly** on the underlying hardware
3. **Isolation**: VMM is in complete control of the virtualized resources (hardware)
 - provides **isolation** and **security**

Quiz: VMM

Based on the **historical** definition of Popek & Goldberg, which of the following is a VMM?

We consider the following host: CPU Intel Core i7, VGA card, Intel Pro 1000 network card, IDE PIIX disk controller

- Java Virtual Machine (JVM)



Quiz: VMM

Based on the **historical** definition of Popek & Goldberg, which of the following is a VMM?

We consider the following host: CPU Intel Core i7, VGA card, Intel Pro 1000 network card, IDE PIIX disk controller

- Java Virtual Machine (JVM)
 - no: equivalence, efficiency and isolation not satisfied
- QEMU running a Raspberry PI VM
 -

Quiz: VMM

Based on the **historical** definition of Popek & Goldberg, which of the following is a VMM?

We consider the following host: CPU Intel Core i7, VGA card, Intel Pro 1000 network card, IDE PIIX disk controller

- Java Virtual Machine (JVM)
 - no: equivalence, efficiency and isolation not satisfied
- QEMU running a Raspberry PI VM
 - no: equivalence and efficiency not satisfied
- Oracle VirtualBox running a VM with: Intel Core i7, VGA card, Intel Pro 1000 network card, IDE PIIX disk controller
 -

Quiz: VMM

Based on the **historical** definition of Popek & Goldberg, which of the following is a VMM?

We consider the following host: CPU Intel Core i7, VGA card, Intel Pro 1000 network card, IDE PIIX disk controller

- Java Virtual Machine (JVM)
 - no: equivalence, efficiency and isolation not satisfied
- QEMU running a Raspberry PI VM
 - no: equivalence and efficiency not satisfied
- Oracle VirtualBox running a VM with: Intel Core i7, VGA card, Intel Pro 1000 network card, IDE PIIX disk controller
 - yes
- Oracle VirtualBox running a VM with: Intel Core i7, VGA card, Realtek RTL8111E network card, SCSI Adaptec disk controller
 -

Quiz: VMM

Based on the **historical** definition of Popek & Goldberg, which of the following is a VMM?

We consider the following host: CPU Intel Core i7, VGA card, Intel Pro 1000 network card, IDE PIIX disk controller

- Java Virtual Machine (JVM)
 - no: equivalence, efficiency and isolation not satisfied
- QEMU running a Raspberry PI VM
 - no: equivalence and efficiency not satisfied
- Oracle VirtualBox running a VM with: Intel Core i7, VGA card, Intel Pro 1000 network card, IDE PIIX disk controller
 - yes
- Oracle VirtualBox running a VM with: Intel Core i7, VGA card, Realtek RTL8111E network card, SCSI Adaptec disk controller
 - no: equivalence not satisfied

Modern VMM definition

- Popek and Goldberg's **efficiency** and **isolation** requirements for a VMM remain true to this day
- However, **equivalence** is not a requirement anymore
- Nowadays, VMMs might expose **different hardware devices** (excluding the CPU architecture) than the physical hardware present on the host and still be called a VMM
 - most VMMs fall in this category
 - in fact, for performance reasons, most VMMs expose virtual hardware devices instead

Platform virtualization

- Sometimes called “hardware virtualization”
- Type of virtualization that **virtualizes a whole machine**
- Three main components must be virtualized:
 - CPU
 - memory (MMU - Memory Managing Unit)
 - devices (also called Input/Output or I/O): hard drive, disk controllers, display, mouse, keyboard, etc.

Privilege levels and traps

Privilege levels

For a traditional OS¹ to provide security and control over user applications, a CPU must provide at least 2 privilege levels:

- **Privileged** (or *supervisor*): the execution level used by the OS' kernel
 - the **kernel** has **access** to **all instructions** and fully controls the CPU
- **Unprivileged** (or *user*): the execution level used by user applications
 - **only a subset** of instructions is **allowed**; **executing a privileged instruction raises a trap** which can then be intercepted by the kernel, which then decides what to do, e.g. *segmentation fault* in UNIX

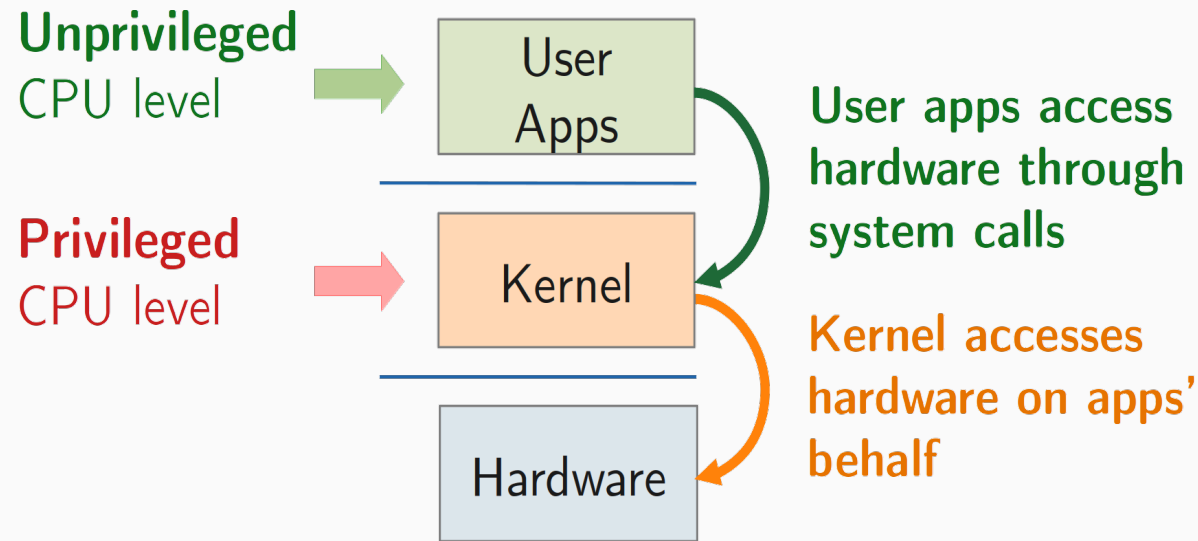
¹By opposition to other types of OSes, such as realtime OSes for embedded systems for instance

Traps

- A **trap** is **raised** when an instruction **cannot be fulfilled** by the CPU
- Typical trap examples:
 - a **unprivileged code** attempts to execute a **privileged instruction**, for instance:
 - disable hardware interrupts
 - modify the CPU flags register
 - halt the CPU
 - an **unprivileged code** attempts to read or write from a device (I/O)
 - a code attempts to address (read or write) a memory address that has **no mapping**

Reminder: privilege levels, kernel and applications

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



Without system calls, apps accessing hardware would trigger **traps** and be likely terminated (*killed*) by kernel

Platform virtualization: CPU virtualization

CPU virtualization techniques

The CPU can be virtualized using 4 different techniques:

- Full virtualization using Trap-and-Emulate (historical)

CPU virtualization techniques

The CPU can be virtualized using 4 different techniques:

- Full virtualization using Trap-and-Emulate (historical)
- Full virtualization using Dynamic Binary Translation

CPU virtualization techniques

The CPU can be virtualized using 4 different techniques:

- Full virtualization using Trap-and-Emulate (historical)
- Full virtualization using Dynamic Binary Translation
- Hardware-assisted full virtualization

CPU virtualization techniques

The CPU can be virtualized using 4 different techniques:

- Full virtualization using Trap-and-Emulate (historical)
- Full virtualization using Dynamic Binary Translation
- Hardware-assisted full virtualization
- Paravirtualization

CPU virtualization techniques

The CPU can be virtualized using 4 different techniques:

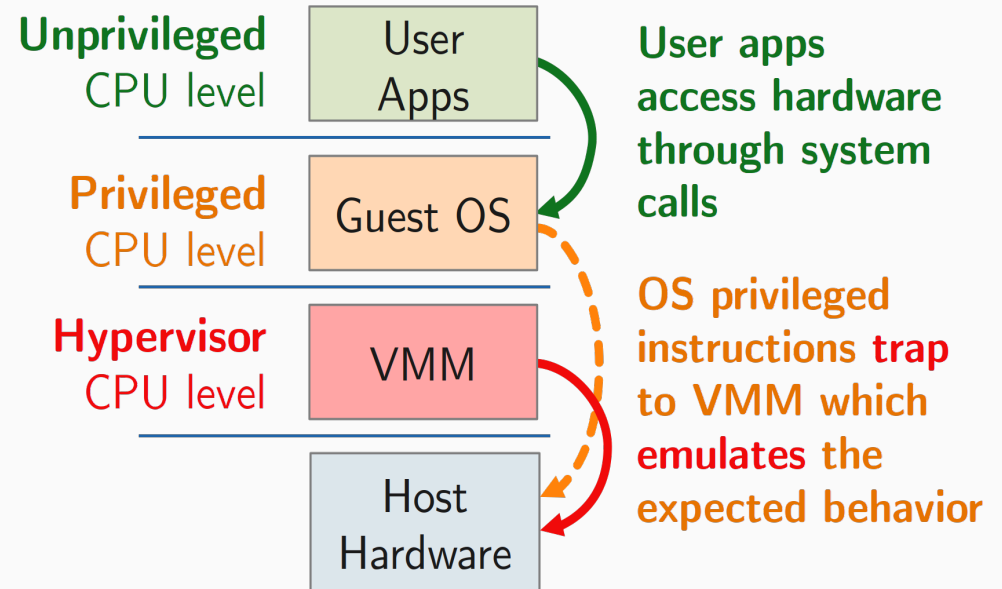
- Full virtualization using Trap-and-Emulate (historical)
- Full virtualization using Dynamic Binary Translation
- Hardware-assisted full virtualization
- Paravirtualization

Here, we present these techniques and their **history**, starting with the first platform virtualization implementation with IBM System/360 CP-67 mainframes, then by VMware's implementation for the PC architecture

CPU virtualization: full virtualization using Trap-and-Emulate

IBM CP-67 added a new **hypervisor** CPU privilege level (more privileged than the traditional **privileged** level) used by the VMM:

- **Most** guest OS instructions (**non-privileged**) are executed **directly**
 - they run at native speed → **efficient**
- **Privileged** instructions trigger a **trap** to the VMM
 - VMM then **emulates** the behavior the guest OS is expecting from the hardware



Used on IBM mainframes since the 1970s

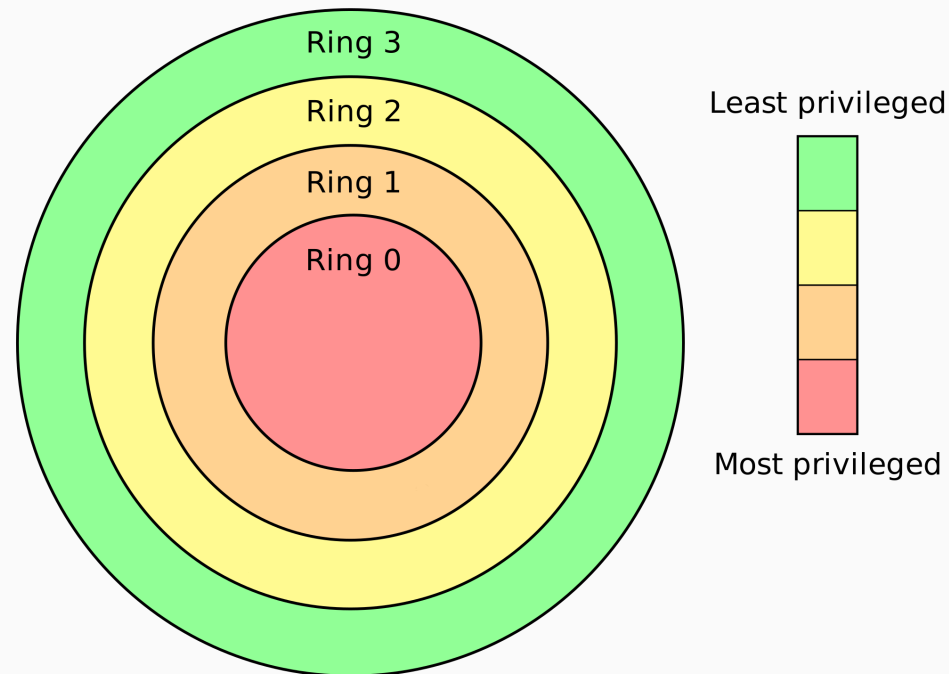
Market changes

- From the 1970s until the mid-1990s, platform virtualization was almost **exclusively used on mainframes**
- Platform virtualization through Trap-and-Emulate worked well on mainframes because their CPUs were **designed** to support it
- Starting in the mid-1990s, as **PC** (x86 architecture) became more **powerful**, platform virtualization became a technology of interest
- Attempts were made to implement Trap-and-Emulate for the PC
- However...

x86 CPU privilege levels (protection)

- Intel 32-bit architecture released in 1985 with the Intel 80386¹ CPU features **4 protection levels**², called **rings**:

- **OS (kernel) runs in ring 0**
- **Applications run in ring 3**



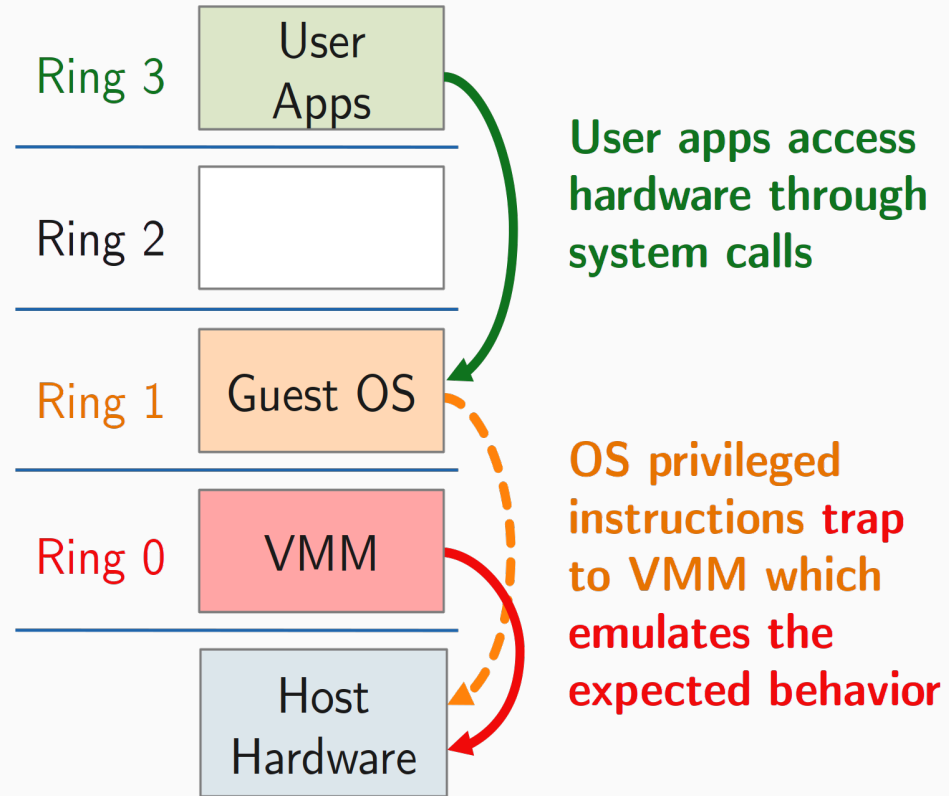
¹AMD released the AM386 in 1991, featuring the same architecture; this led to the creation of the “IA-32” name for the same architecture, which is vendor-agnostic

²Still the case today with the latest Intel/AMD CPUs!

CPU virtualization: attempt at Trap-and-Emulate on x86

Basic idea:

- Guest OS applications run in **ring 3**
- Guest OS kernel runs in **ring 1**
- VMM runs in **ring 0**



Issue with Trap-and-Emulate on x86

- The Trap-and-Emulate model did **not work on x86!**
- On 80386 and later Intel/AMD CPUs (Pentium, etc.):
 - 17 privileged instructions do not trigger a trap when they should
 - instead, they **fail silently!** (don't pass control back to the VMM)
 - VMM: doesn't know → cannot emulate the expected behavior!
 - guest OS: doesn't know → believes operation was **successful!**
 - Example: interrupt enable/disable bit in eflags register; pushf/popf instructions that access it from **ring 1** fail silently

Trap-and-Emulate on x86: what to do?

- The Trap-and-Emulate model **does not work on x86**

Trap-and-Emulate on x86: what to do?

- The Trap-and-Emulate model **does not work on x86**
- Therefore, the x86 architecture cannot be virtualized!

Trap-and-Emulate on x86: what to do?

- The Trap-and-Emulate model **does not work on x86**
- Therefore, the x86 architecture cannot be virtualized!
- What to do?

Trap-and-Emulate on x86: what to do?

- The Trap-and-Emulate model **does not work on x86**
- Therefore, the x86 architecture cannot be virtualized!
- What to do?
- Mendel Rosenblum's research group at Stanford University came-up with a novel idea to implement CPU virtualization:

Dynamic Binary Transation

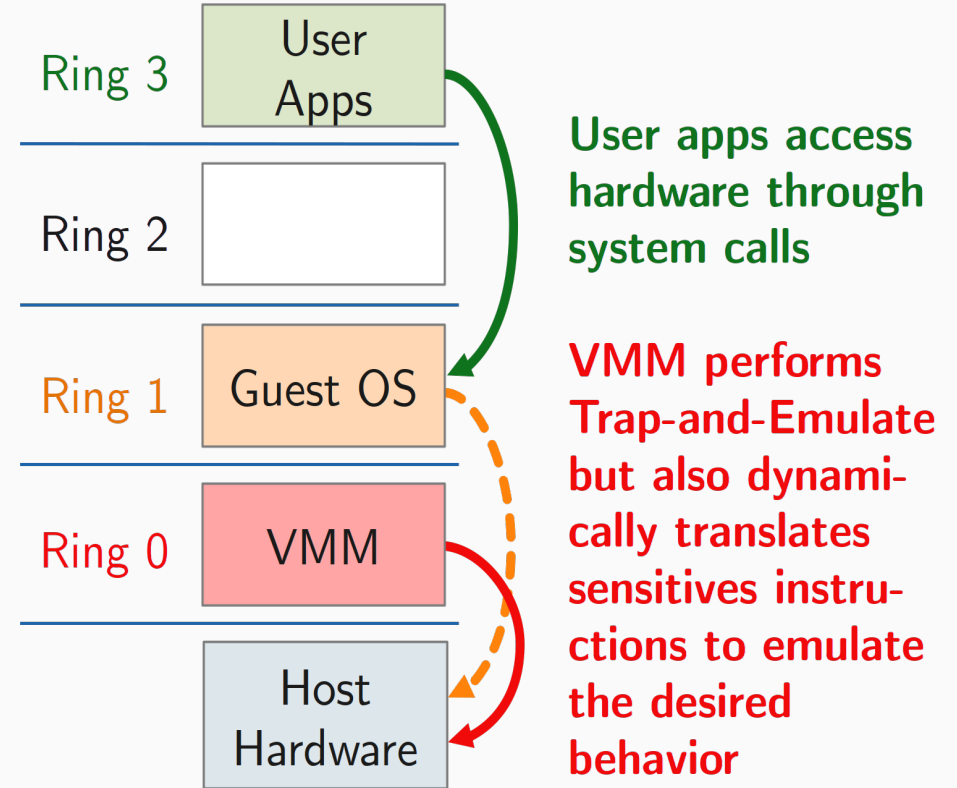
CPU full virtualization: Dynamic Binary Translation

- Mendel Rosenblum's idea was commercialized as VMware¹ in 1998
- **Basic idea:** VMM **modifies** the guest OS' **kernel code** at **runtime** to avoid using the 17 “problem” instructions
- The guest OS is **unaware** it's being modified!
- **Goal:** **avoid modifying** the guest OS kernel' source code!
 - required in order to work with any guest OS kernel (e.g. Windows)

¹One of VMware's five co-founders is Swiss computer scientist [Edouard Bugnion](#)

CPU full virtualization: Dynamic Binary Translation

- **Dynamically** capture code blocks
- **Inspect** code blocks to be executed for the 17 “problem” instructions
- When needed, **translate** to alternate instructions sequence, to **emulate** desired behavior and avoid traps
- Otherwise, which is most of the time → run at **native speed**
- **Speed** optimization: cache translated blocks to amortize translation costs



Dynamic Binary Translation: pros and cons

- Pros

- guest OS kernel' source code does not need to be modified → can run on proprietary kernels (e.g. Windows)

- Cons

- inefficient → low performance
- complex implementation

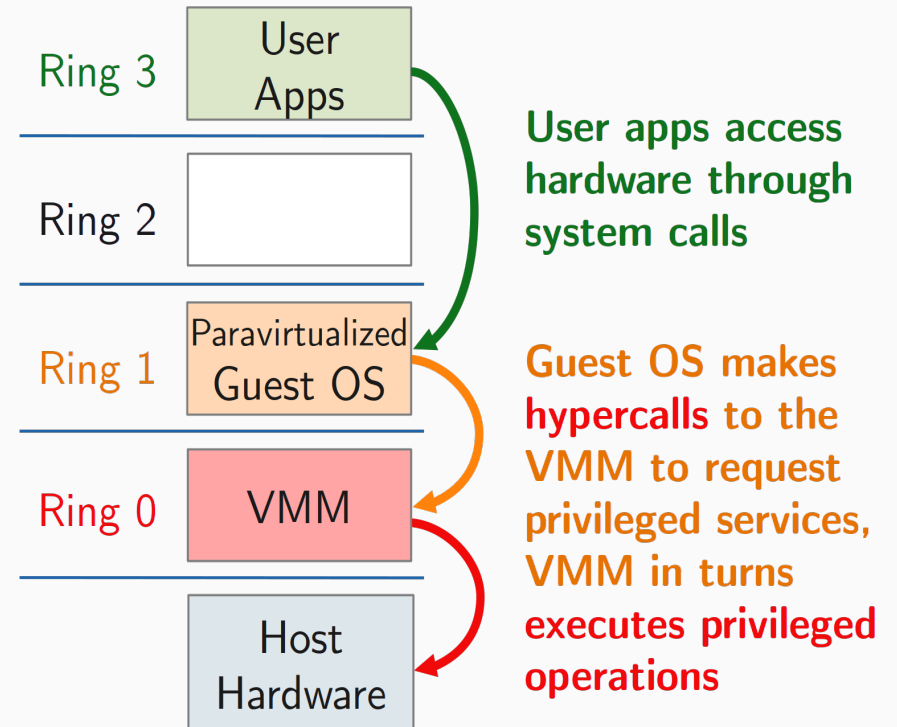
How to improve performance?

CPU paravirtualization: concept

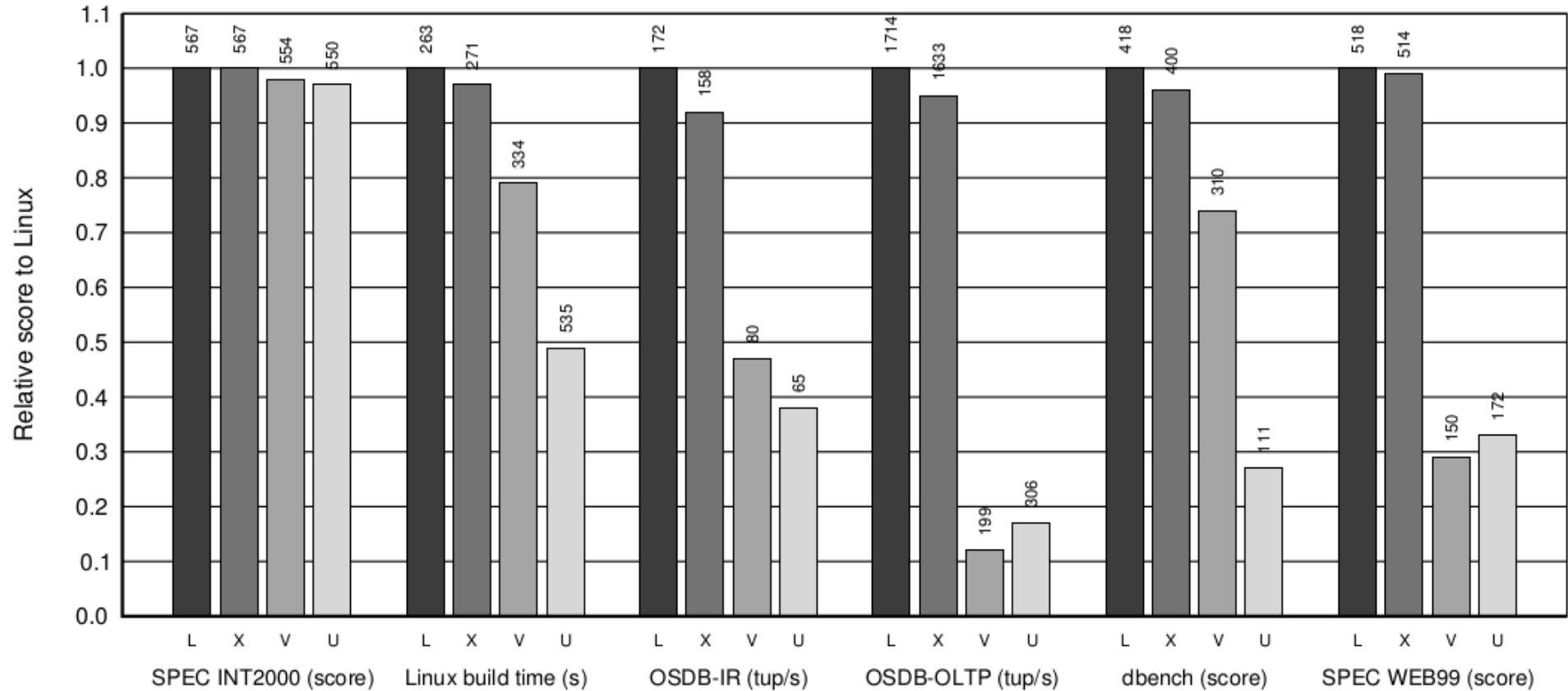
- **Improve performance** by avoiding the overhead/complexity required to support unmodified guest OS' kernel
- **Modify** the guest OS' kernel so that it can **request services** from the VMM
- Pioneered by the Xen hypervisor at the University of Cambridge (UK) in 2003

CPU paravirtualization

- Guest OS' kernel source code is **slightly modified** (only a few %)
- Guest OS **knows** it's running on top of a VMM
- For privilege operations, guest OS **requests services** from the VMM through **hypercalls**
- **Hypercalls** from guest OS to VMM \cong **system calls** from user apps to kernel



Paravirt. vs Dynamic Binary Translation performance comparison



Relative performance of native Linux (L), XenLinux (X), VMware workstation 3.2 (V) and User-Mode Linux (U).

CPU paravirtualization: pros and cons

- Pros

- much better performance than Dynamic Binary Translation
- much simpler to implement than Dynamic Binary Translation

- Cons

- guest OS kernel' source code must be modified → cannot run on closed-source kernels (e.g. Windows)

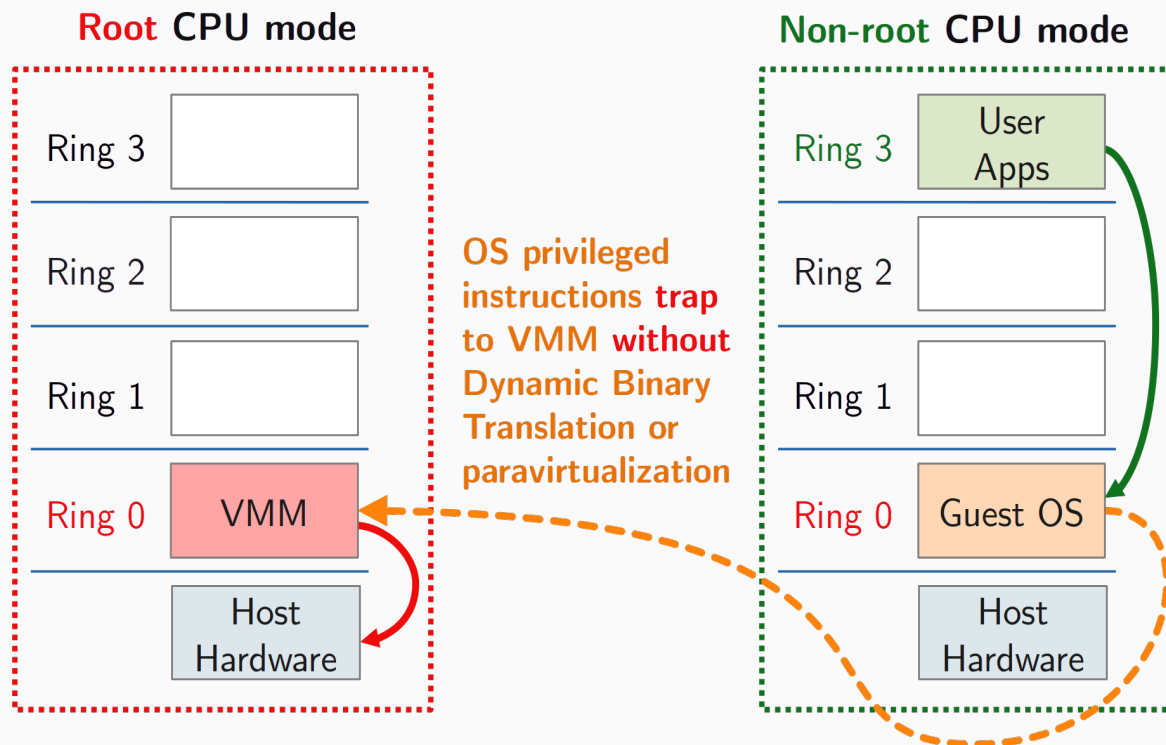
CPU: the need for platform virtualization

- Since 2005, new hardware instructions for virtualization are integrated into Intel/AMD CPUs
- These instructions are called:
 - Intel VT-x for Intel CPUs
 - AMD-V Pacifica for AMD CPUs
- **Goal:** to solve the issue with the 17 “problem” instructions

Starting in 2005, the x86 architecture can finally be virtualized similarly to IBM mainframes in 1970s!

CPU full virtualization: hardware-assisted

- Intel VT-x and AMD-V provide hardware-assisted¹ virtualization instructions
 - add new CPU modes:
root²/**non-root**
 - VMM runs in **root** mode
 - Guest OS runs in **non-root** mode

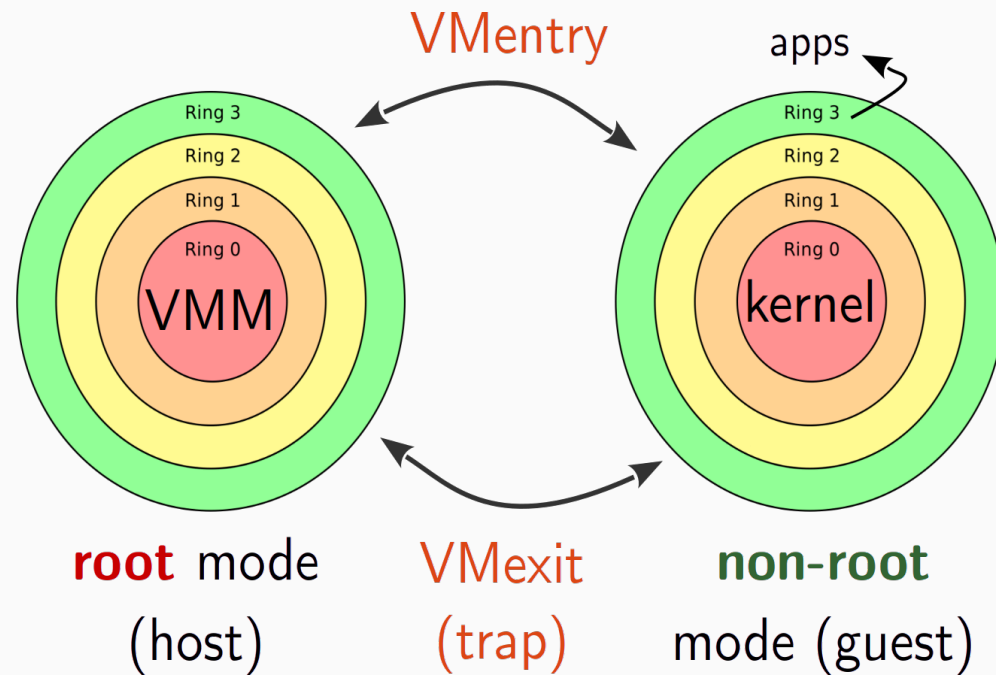


¹Also called “Accelerated Virtualization” and “Hardware Virtual Machine” (HVM)

²Completely unrelated to root user in Linux/UNIX!

CPU hardware-assisted virtualization, root/non-root modes

- VMM runs in **root** mode:
 - **ring 0**: VMM
- Guest OS runs in **non-root** mode:
 - **ring 3**: user applications
 - **ring 0**: kernel
- In **non-root** mode, certain privileged operations trigger **traps** (**VMexits**) → trigger switch to **root** mode (VMM)



CPU hardware-assisted virtualization: pros and cons

- Pros

- guest OS kernel' source code does not need to be modified (by opposition to CPU paravirtualization)
- much more **efficient** than Dynamic Binary Translation, thanks to dedicated hardware instructions

- Cons

- only available if CPU implements the dedicated hardware instructions

Platform virtualization:
device virtualization

Device virtualization techniques

Devices can be virtualized using 4 techniques:

¹We won't present it here

Device virtualization techniques

Devices can be virtualized using 4 techniques:

- Full virtualization using emulation

¹We won't present it here

Device virtualization techniques

Devices can be virtualized using 4 techniques:

- Full virtualization using emulation
- Hardware-assisted full virtualization (using VT-d hardware)¹

¹We won't present it here

Device virtualization techniques

Devices can be virtualized using 4 techniques:

- Full virtualization using emulation
- Hardware-assisted full virtualization (using VT-d hardware)¹
- Paravirtualization

¹We won't present it here

Device virtualization techniques

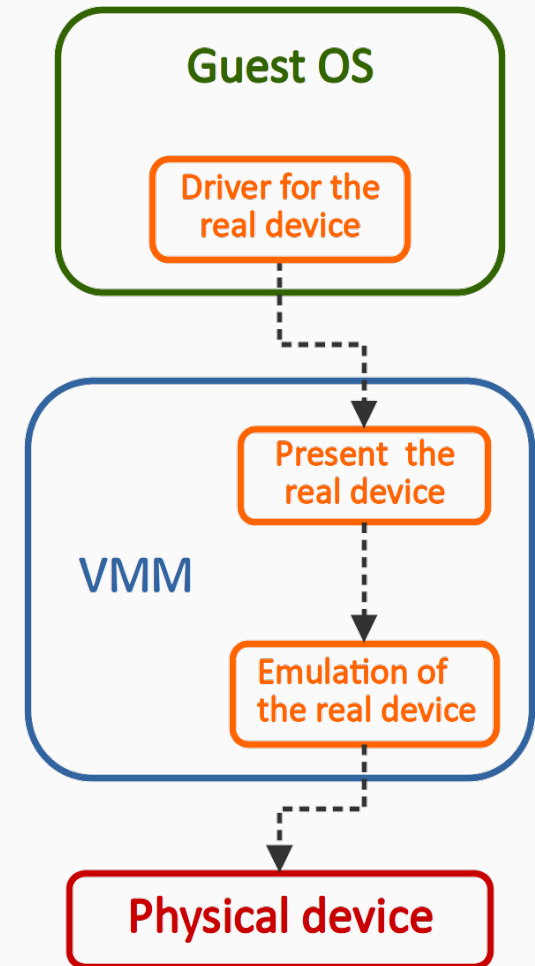
Devices can be virtualized using 4 techniques:

- Full virtualization using emulation
- Hardware-assisted full virtualization (using VT-d hardware)¹
- Paravirtualization
- Passthrough

¹We won't present it here

Device virtualization: full virtualization using emulation

- VM **presents a “real” device** to the guest OS
- Guest OS must have drivers for the real device
- VMM intercepts all device accesses
- VMM **emulates** a real device that’s likely **not physically present** on the host
- **Pros**
 - VM decoupled from physical device
 - guest OS can run on real hardware (provided it has the required drivers)
 - VM migration
 - device sharing
- **Cons**
 - emulating a real device can be complex
 - low performance due to lots of VM exits



Reminder: OS kernel, drivers and user applications

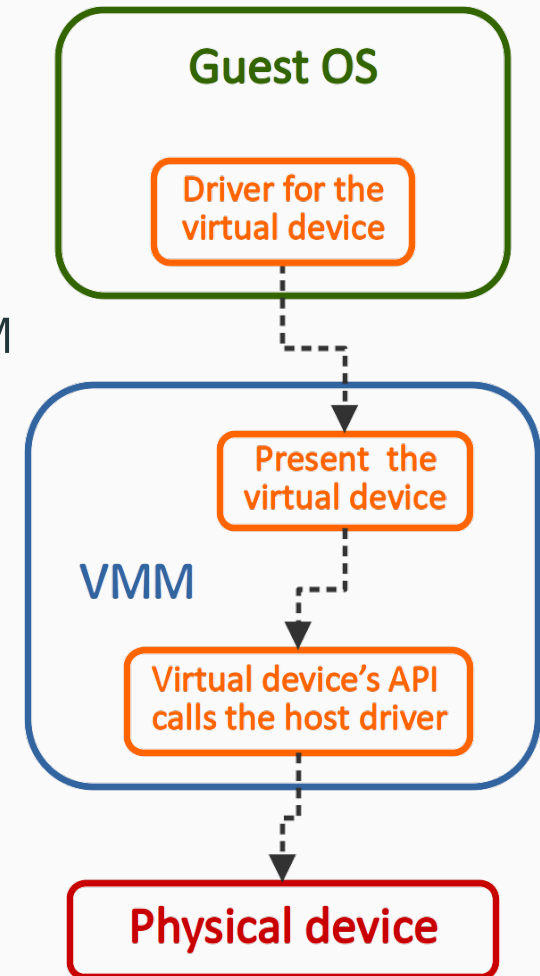
- An OS is composed of: kernel, libraries, and user applications
- The kernel is composed of:
 - the kernel itself
 - drivers (to controls devices)
- **Full kernel source code is not required to write drivers!**
 - anyone can write device drivers for proprietary OSes
 - only the driver SDK¹ is required and available for all OSes

¹Software Development Kit

Device virtualization: paravirtualization

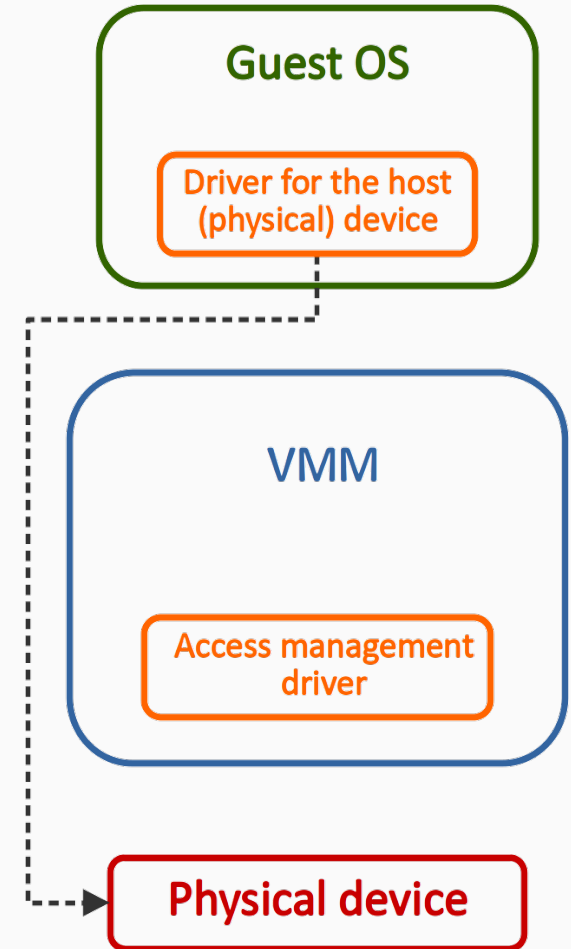
- VM **presents a virtual device** to the guest OS
- Guest OS must have driver for the virtual device
- Driver **much simpler** than for a real device
- Driver uses the virtual device's API to control it
 - uses **hypercalls + shared memory** to communicate with VMM
 - **simple and highly efficient**
- **Pros**
 - VM decoupled from physical device
 - no need to emulate a real device
 - easy to implement & high performance
- **Cons**
 - guest OS requires specific driver
 - guest OS cannot run on real hardware

- VM migration
- device sharing



Device virtualization: passthrough

- VMM gives guest OS **exclusive direct access** to physical device
- **Pros**
 - native (highest) performance
- **Cons**
 - device cannot be shared (or very difficult)
 - VM migration difficult
 - host must have the exact device type expected by the guest OS



Platform virtualization nowadays

Nowadays, VMMs that implement platform virtualization use a combination of virtualization types:

- **Hardware-assisted** full virtualization for CPU and devices
- **Paravirtualization** for devices
 - typically for performance-critical devices, such as disk and network
- **Full virtualization** (emulation) for devices
 - typically used for better compatibility: when guest OS lacks paravirtualized drivers

Hypervisor models

Modern hypervisors

- All hypervisors are based on:

Modern hypervisors

- All hypervisors are based on:
 - **hardware assisted** virtualization for the CPU

Modern hypervisors

- All hypervisors are based on:
 - **hardware assisted** virtualization for the CPU
 - **emulated** real devices through **full virtualization**

Modern hypervisors

- All hypervisors are based on:
 - **hardware assisted** virtualization for the CPU
 - **emulated** real devices through **full virtualization**
 - **paravirtualized** “virtual (non-real)” devices

Hypervisor models

Historically, hypervisors fall into two models:

Hypervisor models

Historically, hypervisors fall into two models:

(a) **hypervisor that runs directly on the hardware**

- minimalistic OS that has only one goal: to manage VMs
- much smaller complexity than a general OS
- historically called **type-1** or **baremetal** hypervisors

(b)

Hypervisor models

Historically, hypervisors fall into two models:

(a) **hypervisor that runs directly on the hardware**

- minimalistic OS that has only one goal: to manage VMs
- much smaller complexity than a general OS
- historically called **type-1** or **baremetal** hypervisors

(b) **OS kernel modified to add a virtualization layer**

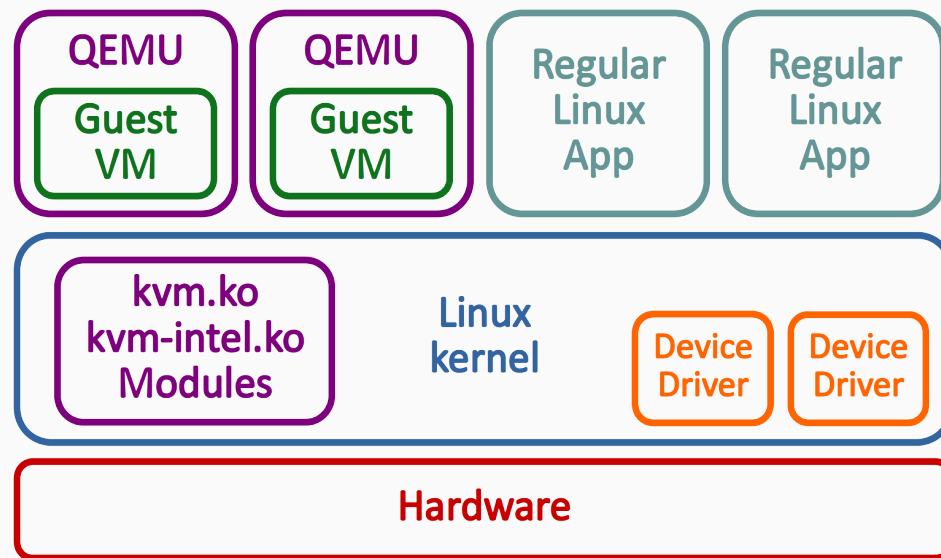
- “transform” a general-purpose OS into a hypervisor
- historically called **type-2** or **hosted** hypervisors

Nowadays, it's not so clear cut, for instance, Linux/KVM is classified as either type-2 or type-1, depending on the source

KVM (“Kernel Virtual Machine”) + QEMU (a)

Linux kernel module that provides hardware-assisted virtualization

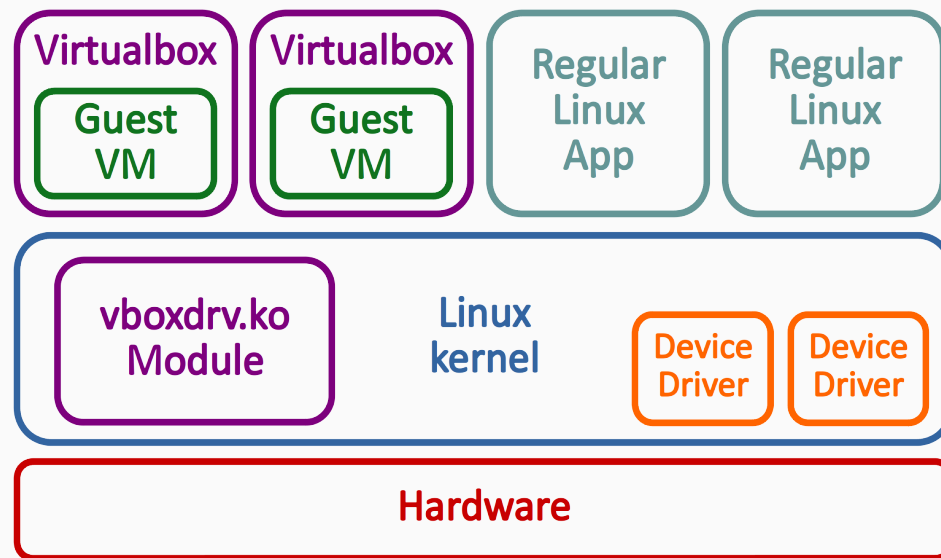
- Exposes a user-space virtualization API
- Requires VT-x or AMD-V
- Linux kernel provides hardware management + runs regular Linux applications
- VMs support through QEMU which uses the KVM API
- First released in 2006



VirtualBox (a)

Similar model to KVM + QEMU

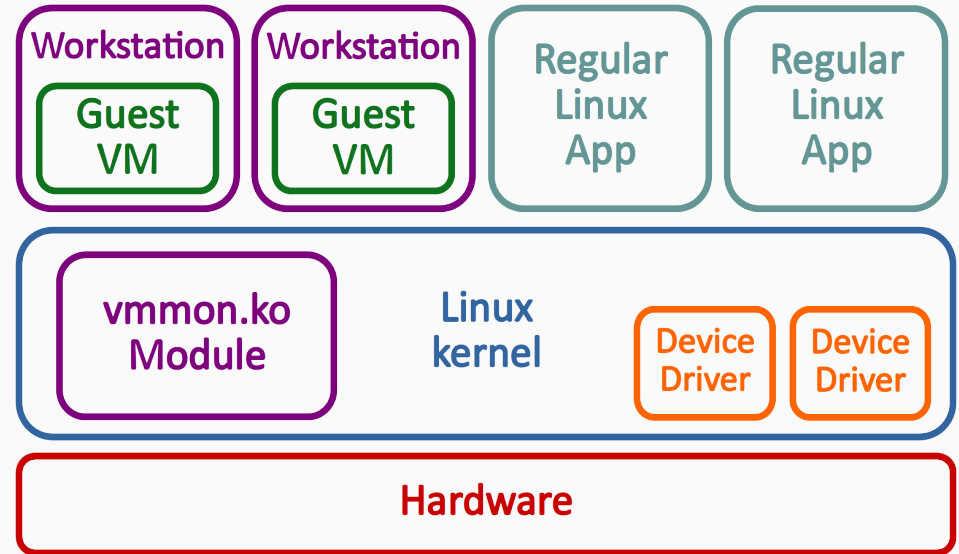
- Uses its own kernel module which provides similar features to KVM
 - however, since early 2024 it can also use KVM
- Developed by Innotek in 2007 (acquired by Sun Microsystems in 2008, in turn acquired by Oracle in 2010)



VMware Workstation (a)

Similar model to KVM + QEMU and VirtualBox

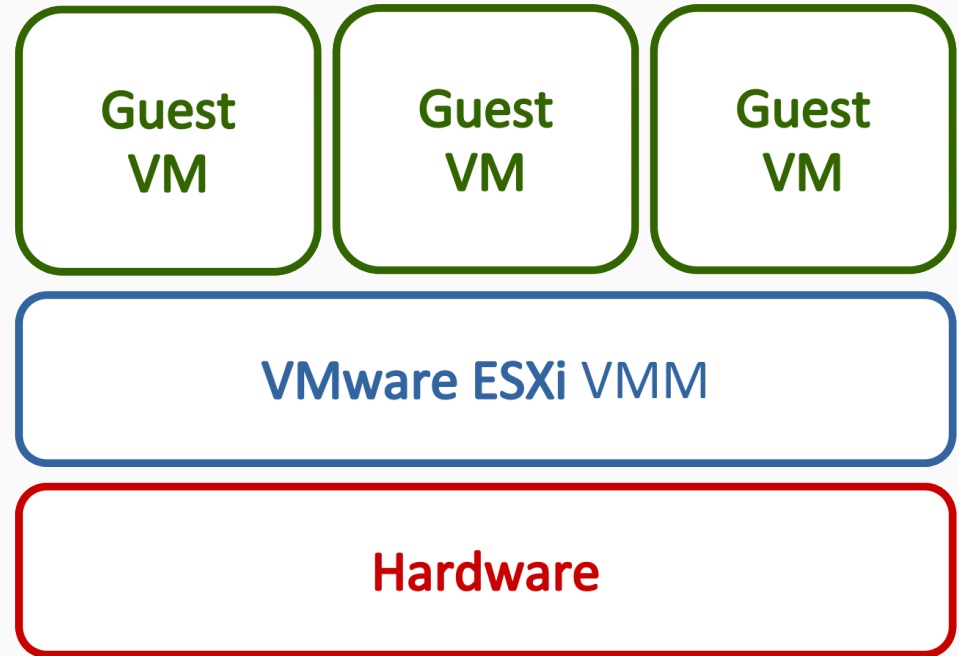
- Uses its own kernel module which provides similar features to KVM
- First released in 1999



VMware ESXi (b)

VMM manages all hardware resources and supports execution of VMs

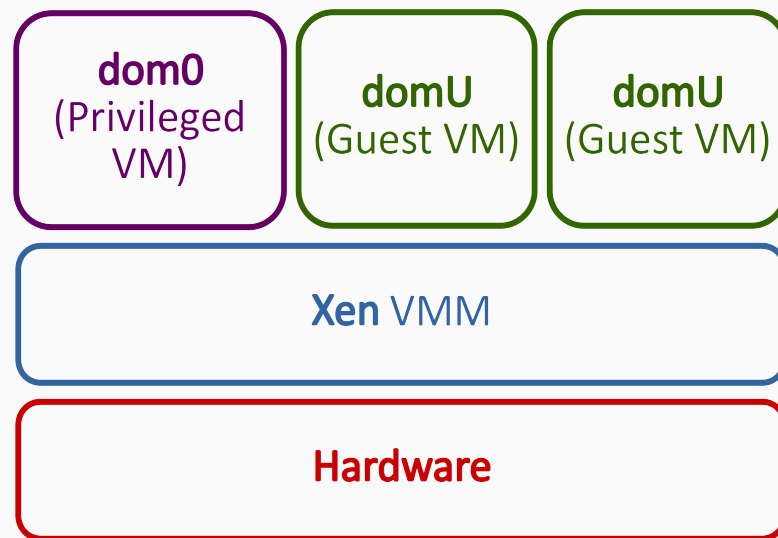
- Device manufacturers provide device drivers for the hypervisor
- **Limited** hardware support: ESXi **only available** for dedicated servers with compatible/supported hardware
- First released in 2001



Xen (b)

VMM manages all hardware resources and supports execution of VMs

- Create a **privileged VM (dom0)** that runs **Linux OS with full hardware privileges**
 - **run all device drivers**
 - manage **unprivileged VMs (domU)**
 - system configuration and management, e.g. how VMM share resources across VMs
 - uses QEMU for emulation of physical devices

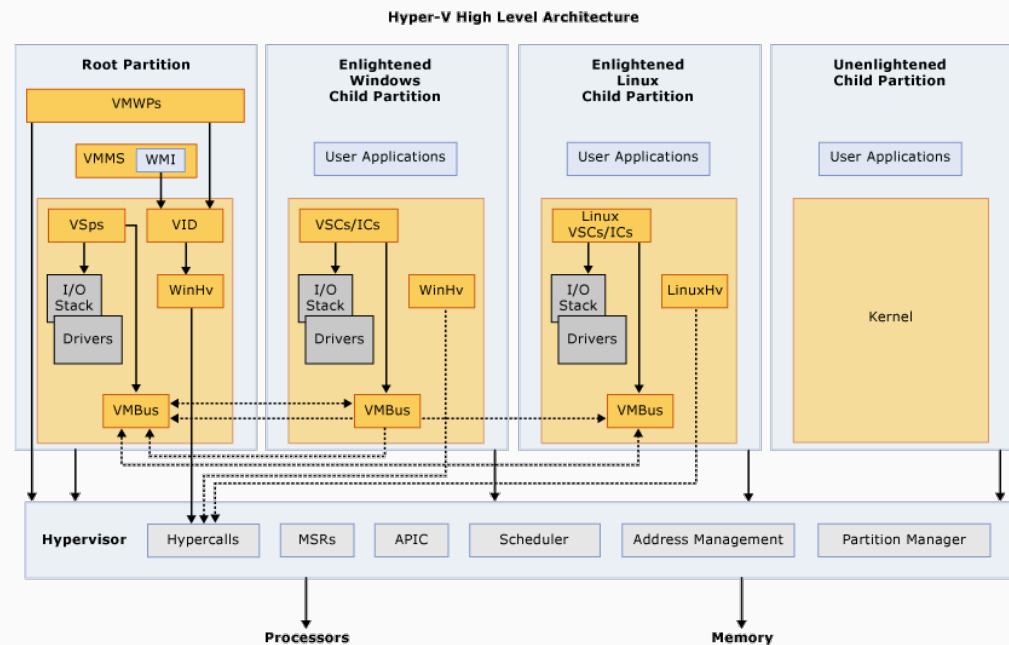


First released in 2003

Microsoft Hyper-V (b)

VMM manages all hardware resources and supports execution of VMs

- Parent partition runs Windows Server OS with full hardware privileges
 - direct access to hardware devices
 - create/manage child partitions which host VMs with guest OS
 - system config and management
- Similar model¹ to XEN
- First released in Windows Server 2008



¹<https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/hyper-v-architecture>

Additional examples

- FreeBSD bhyve → similar model to KVM (a)

Additional examples

- FreeBSD bhyve → similar model to KVM (a)
- VMware Fusion → similar model to VirtualBox (a)

Additional examples

- FreeBSD bhyve → similar model to KVM (a)
- VMware Fusion → similar model to VirtualBox (a)
- Apple Hypervisor for OSX (hvf) → similar model to KVM (a)

Additional examples

- FreeBSD bhyve → similar model to KVM (a)
- VMware Fusion → similar model to VirtualBox (a)
- Apple Hypervisor for OSX (hvf) → similar model to KVM (a)
- Citrix Hypervisor → similar model to Xen (b)

Additional examples

- FreeBSD bhyve → similar model to KVM (a)
- VMware Fusion → similar model to VirtualBox (a)
- Apple Hypervisor for OSX (hvf) → similar model to KVM (a)
- Citrix Hypervisor → similar model to Xen (b)
- VMware Player → similar model to VMware Fusion (a)

What's used out there?

- Amazon AWS & EC2 → from Xen to Nitro which is a stripped-down version of KVM + QEMU

What's used out there?

- Amazon AWS & EC2 → from Xen to Nitro which is a stripped-down version of KVM + QEMU
- Microsoft Azure → Hyper-V

What's used out there?

- Amazon AWS & EC2 → from Xen to Nitro which is a stripped-down version of KVM + QEMU
- Microsoft Azure → Hyper-V
- Google Cloud Platform → KVM + QEMU

What's used out there?

- Amazon AWS & EC2 → from Xen to Nitro which is a stripped-down version of KVM + QEMU
- Microsoft Azure → Hyper-V
- Google Cloud Platform → KVM + QEMU
- Most cloud providers (e.g. DigitalOcean) → KVM + QEMU

What's used out there?

- Amazon AWS & EC2 → from Xen to Nitro which is a stripped-down version of KVM + QEMU
- Microsoft Azure → Hyper-V
- Google Cloud Platform → KVM + QEMU
- Most cloud providers (e.g. DigitalOcean) → KVM + QEMU
- Everything OpenStack → generally powered by KVM + QEMU

What's used out there?

- Amazon AWS & EC2 → from Xen to Nitro which is a stripped-down version of KVM + QEMU
- Microsoft Azure → Hyper-V
- Google Cloud Platform → KVM + QEMU
- Most cloud providers (e.g. DigitalOcean) → KVM + QEMU
- Everything OpenStack → generally powered by KVM + QEMU
- VMware remains leader in solutions for business/private environments

Resources

- [Bringing Virtualization to the x86 Architecture with the Original VMware Workstation](#) E. Bugnion, S. Devine, M. Rosenblum, J. Sugerman, E. Wang; ACM Transactions on Computer Systems, 2012
- [Virtual Machine Monitors from “Operating Systems: Three Easy Pieces”](#) Remzi H. et Andrea C. Arpaci-Dusseau; Arpaci-Dusseau Books
- “Hardware and Software Support for Virtualization”; E. Bugnion, J. Nieh, D. Tsafir; Morgan & Claypool Publishers, 2017
- “Virtual Machines: Versatile Platforms for Systems and Processes”; J. Smith, R. Nair; Morgan Kaufmann, 2005
- [Xen and the Art of Virtualization](#), P. Barham et al., ACM SIGOPS Operating Systems Review, Volume 37, Issue 5, 2003
- [Virtualization in Xen 3.0](#), Rami Rosen, Linux Journal, 2006
- [Understanding Full Virtualization, Paravirtualization, and Hardware Assist](#), VMware White Paper, 2007