Heavy Virtualization

Guillaume Urvoy-Keller

January 15, 2021

Source documents

- Bugnion, Edouard, Jason Nieh, and Dan Tsafrir. "Hardware and software support for virtualization." Synthesis Lectures on Computer Architecture 12.1 (2017): 1-206.
- https://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html
- https://en.wikipedia.org/wiki/X86_virtualization
- https://www.hardwaresecrets.com/everything-you-need-to-know-about-the-intel-virtualization-technology/3/

Heavy Virtualization dates back to the mainframes era...

> "Virtual machines have finally arrived. Dismissed for a number of years as merely academic curiosities, they are now seen as cost-effective techniques for organizing computer systems resources to provide extraordinary system flexibility and support for certain unique applications".
>
> Robert. P. Goldberg, *IEEE Computer*, 1974 [78]

.... and made a come-back at the very end of the 90s:

- In 1999, VMware released VMware Workstation 1.0 the first commercial virtualization solution for x86 processors.
- Intel introduced its first-generation hardware support for virtual machines in 2004.

Heavy Virtualization ages

- Age 1: Virtualization without Architectural Support
- Age 2: Virtualization with Architectural Support
  - KVM was built on top of the hardware support provided by Intel

What we need to cover first:

- Some details about what virtualization is ... as virtualization is not limited to VM
- The role of the hypervisor
- The Popek and Goldberg Theorem

Virtualization Reloaded

Virtualization : definition from Bugnion et al.

Virtualization is the application of the layering principle through enforced modularity, whereby the exposed virtual resource is identical to the underlying physical resource being virtualized.

Virtualization : definition from Bugnion et al.

Definition is grounded in two fundamental principles of computer systems.

- Layering : presentation of a single abstraction
- Enforced modularity: guarantees that the clients of the layer cannot bypass the abstraction layer, for example :
  - to access the physical resource directly
  - or have visibility into the usage of the underlying physical resource.

Virtualization

An instance of layering for which the exposed abstraction is equivalent to the underlying physical resource.

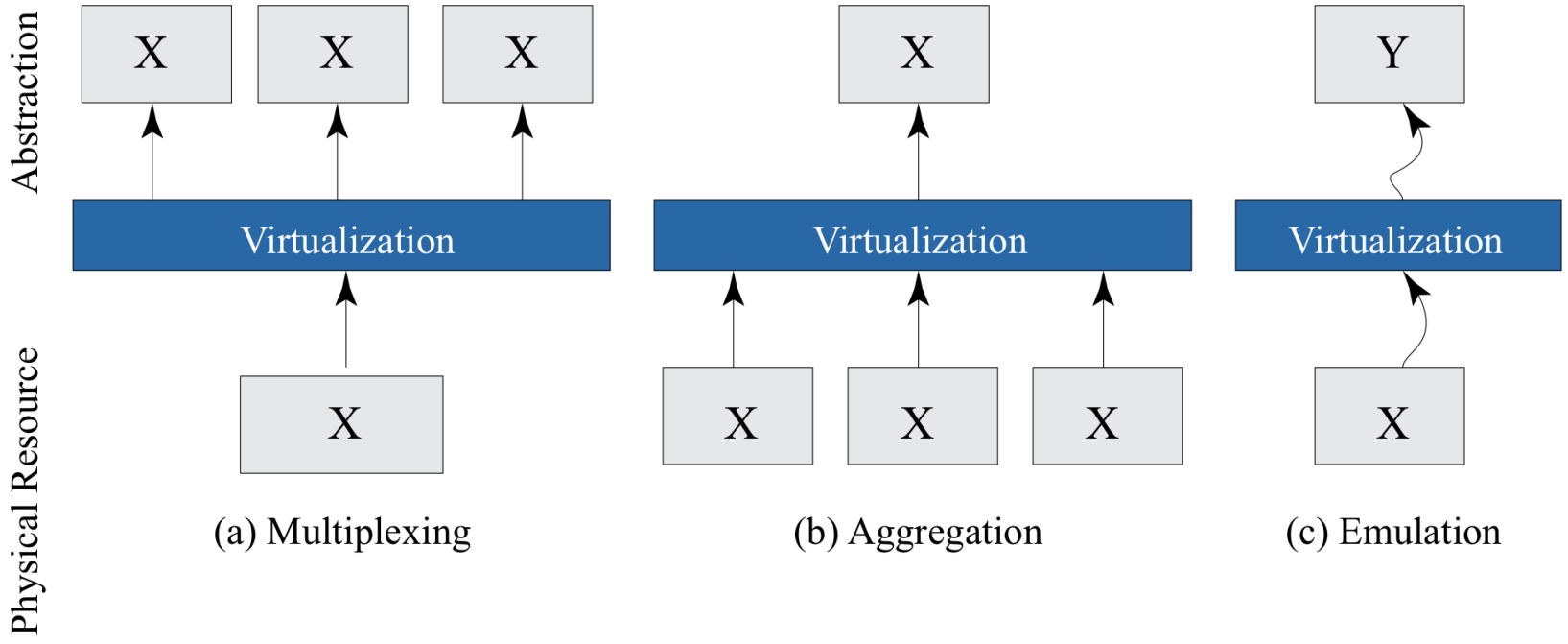
Virtualization : Definition is not limited to virtual machines

- RAID: a *redundant array of inexpensive disk* is aggregated to form a single, virtual disk. Two key advantages:
  - A compatible interface (a block device for both the virtual and physical disks) $\rightarrow$ a filesystem can be deployed identically, with or without the RAID !!
  - RAID hides the physical addresses from the abstraction $\rightarrow$ physical disks can be swapped into the virtual disk transparently from the filesystem using it.

Virtualization : Definition is not limited to virtual machines:

- OS safely exposes the resources of a computer—CPU, memory, and I/O—to multiple, concurrent applications.
- For example, an operating system controls the MMU to expose the abstraction of isolated address spaces to pro- cesses;
  - MMU: Memory Management Unit controls the mapping between virtual addresses (seen by appli) and physical addresses (seen by kernel only)

Virtualization Techniques



- Multiplexing ex: OS with CPU or memory
- Aggregation ex: RAID disk
- Emulation ex: a RAM disk emulates a disk in RAM (e.g. /proc in Linux)

Hypervisor

Hypervisor

Definition

A software (a simple soft like Virtualbox or complete OS like VMware ESX) runs virtual machines with the goal of minimizing execution overheads.

Popek and Goldberg in 1974 specifies 3 key features of hypervisor:

- Equivalence: exposed resource (i.e., the virtual machine) is equivalent with the underlying computer.
- Safety: VMs are isolated from each other as well as from the hypervisor.
- Performance: VM must show at worst a minor decrease in speed.
  - The latter separates hypervisors from machine simulators, e.g. to simulate ARM processor on Intel (ex: Android simulator)
  - Simulator means that no native code of VM is sent to processor while this is the case of hypervisor $\rightarrow$ Direct Execution

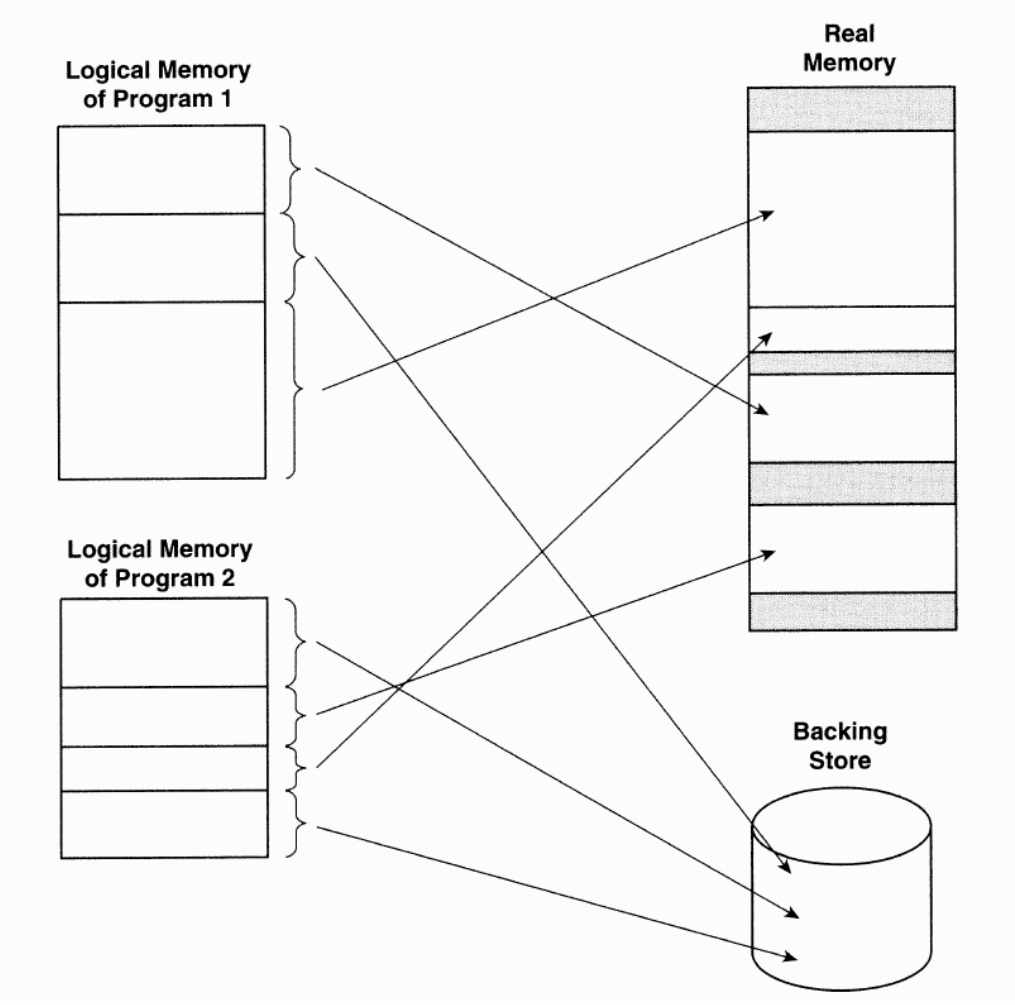
Ideal Hypervisor

- To achieve high performance $\rightarrow$ direct code execution of VM, ie. from virtual CPU to physical
- As guest OS is unprivileged $\rightarrow$ frequent traps.
- Hypervisor will emulate action for guest OS and give it back control of CPU

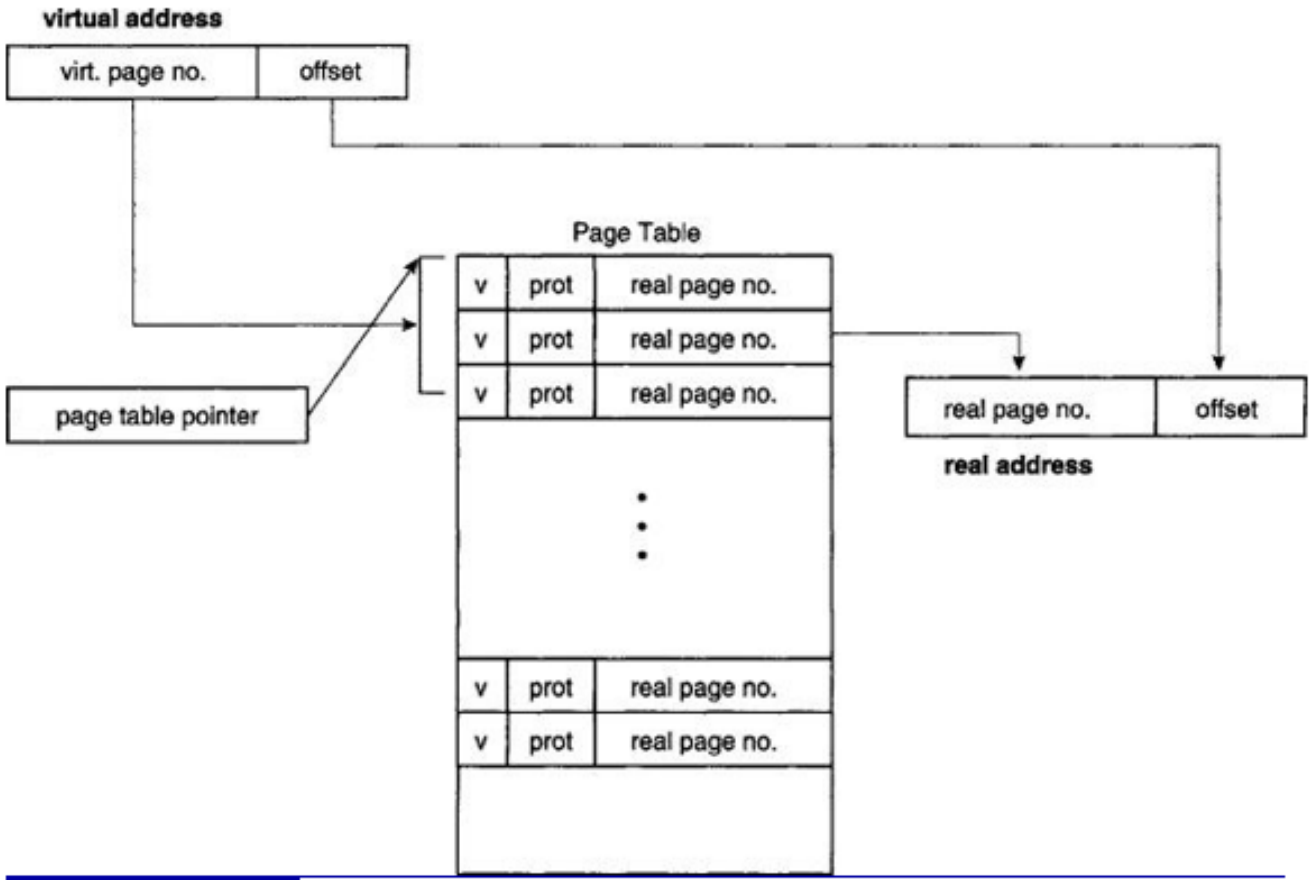
  $\rightarrow$ Trap-and-Emulate approach

  Ex: guest OS wants to manipulate page table. Hypervisor will take over after trap, update the shadow page table and the real page table ...and resrtart the VM

Classical Virtual Memory: an OS hides the physical memory to the processes via page table (next slide)



Classical Virtual Memory: OS controls the Page Table

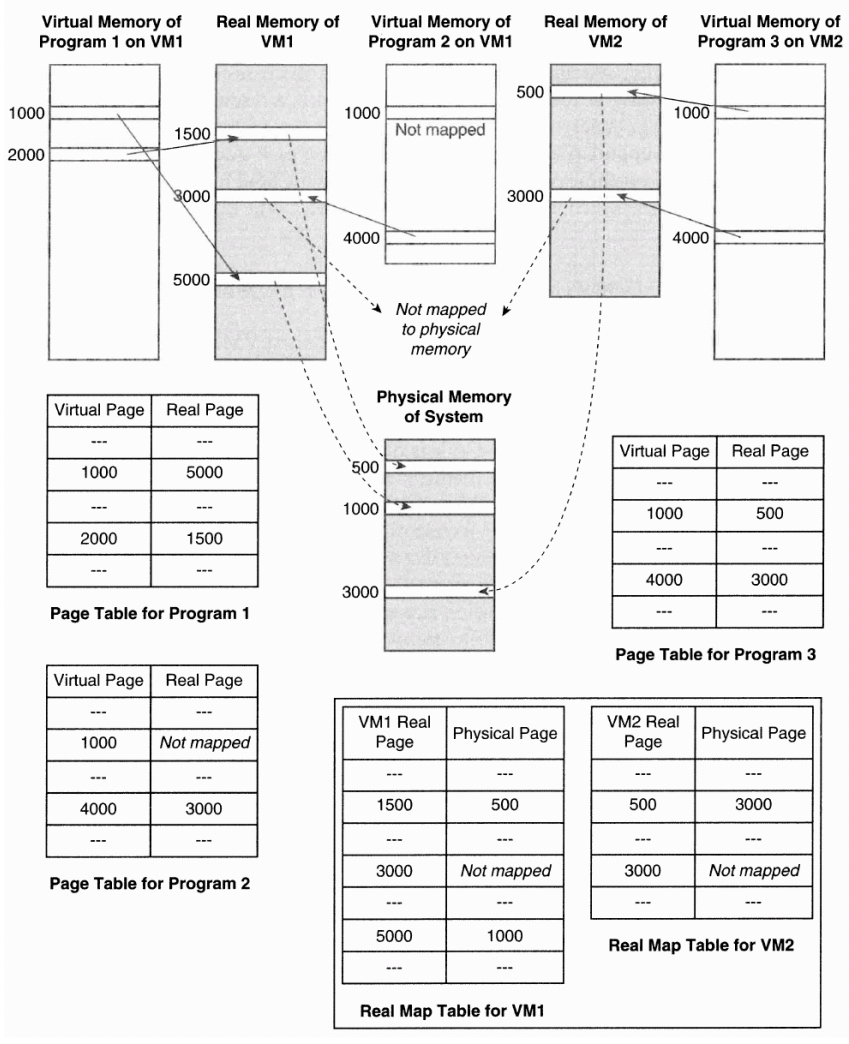


Virtual Memory with VMs: hypervisor controls the Real Page Table and Emulate it for guest OS

Tow levels of indirections

Two levels of traductions:

- Guest program $\rightarrow$ guest OS
- Guest OS $\rightarrow$ host OS



Virtualization of CPU, RAM, I/O

- CPU and RAM virtualization is achieved via temporal and spatial multiplexing
  - RAM $\rightarrow$ spatial multiplexing
  - CPU $\rightarrow$ temporal multiplexing
- I/O virtualization is done via emulation
  - Key advantage: portability $\rightarrow$ a VM sees the same virtual hardware on different physical platform (when migrated or in case of change of physical parts)
  - Hardware support now exists - see age 2 of virtualization

Example with an early type 2 hypervisor : VMware workstation

One sees a mix of multiplexing and emulation

Table 1.1: Virtual Hardware of early VMware Workstation [45]

	Virtual Hardware (front-end)	Back-end
Multiplexed	1 virtual x86-32 CPU	Scheduled by the host operating system with one or more x86 CPUs
	Up to 512 MB of contiguous DRAM	Allocated and managed by the host OS (page-by-page)
Emulated	PCI Bus	Fully emulated compliant PCI bus with B/D/F addressing for all virtual mother-board and slot devices
	4 x 4IDE disks 7 x Buslogic SCSI Disks	Either virtual disks (stored as files) or direct access to a given raw device
	1 x IDE CD-ROM	ISO image or real CD-ROM
	2 x 1.44 MB floppy drives	Physical floppy or floppy image
	1 x VGA/SVGA graphics card	Appears as a Window or in full-screen mode
	2 x serial ports COM1 and COM2	Connect to Host serial port or a file
	1 x printer (LPT)	Can connect to host LPT port
	1 x keyboard (104-key) and mouse	Fully emulated
	AMD PCnet NIC (AM79C970A)	Via virtual switch of the host

The different types of virtualization

Full (software) virtualization Runs unmodified guest OS on architectures (hardware) lacking support for it $\rightarrow$ Age 1
Hardware Virtualization: Hypervisors benefits from support from CPU and possibly peripherals $\rightarrow$ Age 2
Paravirtualization: favors simplicity and efficiency over full compatibility $\rightarrow$ modified guest OS.

- Popularized by Xen
- Still used nowadays in the form of platform specific extensions often implemented as driver to **unmodified** guest OS, e.g. to implement high performance front-end device. Ex: Virtualbox guest additions

A last problem...is an ISA virtualizable?

What does it mean?

It means that the trap-and-emulate approach is doable $\iff$ the hypervisor can exclusively rely on direct execution

Indeed, imagine that for some instructions the guest OS ends up in an undesirable state as no trap occurred $\rightarrow$ Problem!!

Popek and Golberg theorem

- Theorem states if an ISA is virtualizable
- Assumptions:
  - One processor
  - Processor has two modes of executions
  - Support of virtual memory via segmentation where physical memory consists of set of segments of size L and a process sees a virtual address $x \in [0, L]$ mapped to a segment with offset B (ie physical address $\in [B, B + L]$)
  - Processor state, called processor status word (psw) consitsts of:
    - execution level (root,user)
    - segment register (B,L)
    - current program counter (PC), a virtual address
  - Trap saves psw in a well-known memory and load in psw another well-known address (so that OS can take over control and check psw of stopped process)

Popek and Golberg theorem

> Theorem 1 [143]: For any conventional third-generation computer, a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.

Two types of sensitive operations:

- Control sensitive can change the system state
- behavior-sensitive if its semantics depends on the actual values of system state

Elements of proof (when ISA virtualizable)

- Hypervisor is the only software in root mode
- Each VM has a fixed and contiguous part of physical memory $\rightarrow$ isolation
- Hypervisor has reserved its own physical memory, different from VMs
- Hypervisor keeps in (its own) memory a copy of the psw of each VM
- ...

And the bad news....

- “Analysis of the Intel Pentiums Ability to Support a Secure Virtual Machine Monitor” USENIX 2000, J. Robin, C. Irvine
  Over the 250 instructions, 17 are sensitive but not privileged (they are called critical)
  Problem is still present in current x86 architecture (common to Intel and AMD)
- ARM processors also feature sensitive but not privileged instructions

Behavior and control sensitive operations: when things go wrong

Behavior sensitive example

Assume that an instruction can read the actual PSW state (root or user) without a trap $\rightarrow$ a guest OS could observe that it executes in... user mode !

Control sensitive example

In x86: popf

- Popf in user mode: change ALU flag
- Popf en kernel mode: change ALU **and system flag**

Problem: if guest OS executes this command, no trap to hypervisor and guest OS in inconsistent state

Age 1: Non Hardware Assisted Virtualization

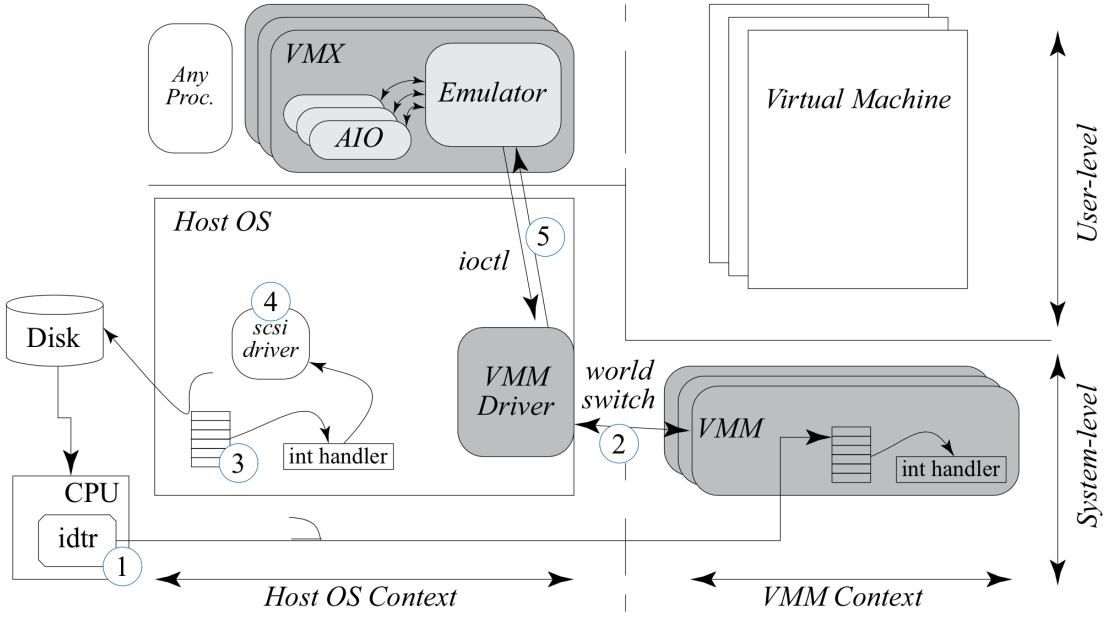
Subtitle: How to deal with a an ISA that does not respect the Popek and Goldberg criterion

The VMware approach

- Combine direct execution with dynamic binary translation which consists in patching the instructions sent by guest OS.
- Key idea: most of the time, e.g. to execute well-behaving applications from guest, direct execution is possible and fast!!!
- Example : a guest application that mostly performs computations on its data should benefit from direct execution
- Note: still to solve the problem of memory access as guest OS does not control MMU: we don't discuss this but VMware optimized the process.

The VMware approach

- VMM runs at the same level as host OS
- VMM delegates I/O operations from VM to VMX that acts as a normal process asking I/O operation to the host OS → no need to take care about drivers
- World switch: when VM traps, provides all the info for VMM to do its job. Saves more than a classical trap to help VMM. → hardware support from Intel/AMD will replace the world switch with info per VM on processor.

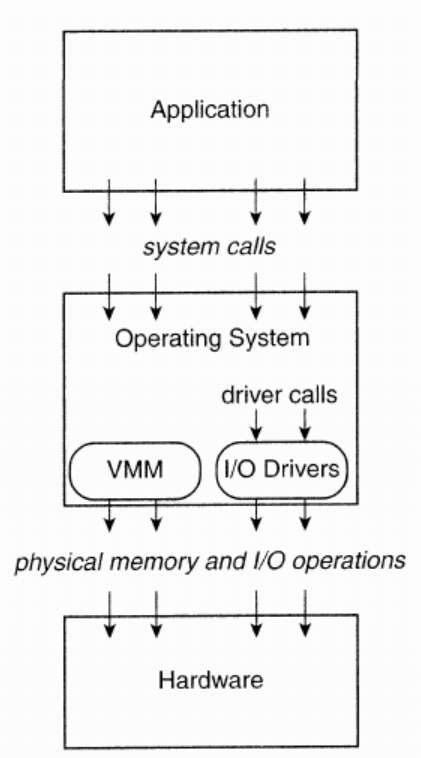


The Xen approach

- Xenserver first released in 2003.
- Prohibit the use the 17 offenging commands
- Replace them with hypercalls: system calls from guest OS to hypervisor $\rightarrow$ need to patch guest OS

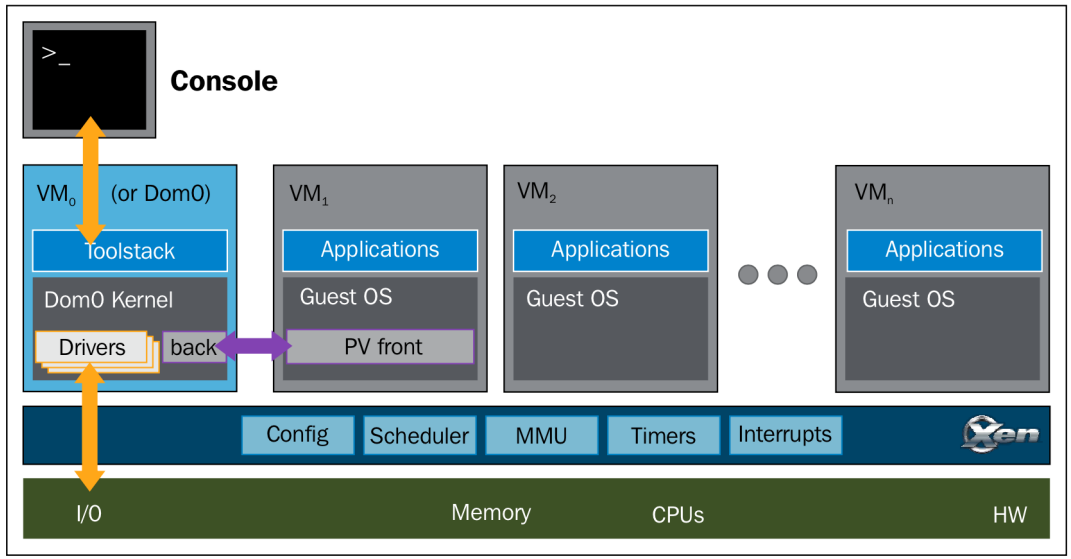
Virtualizing I/O

- Where to virtualize
- System calls? Drivers? I/O operations? $\rightarrow$ too complex
- Solution used: driver level (emulation) $\rightarrow$ offer a normalized interface to device(e.g., always the same Intel Ethernet card, irrespectively of physical one) and provide a specific driver to guest OS to intercept and translate requests



Virtualizing I/O: The Xen approach

- Pack all drivers in a specific VM, called Dom0 (to prevents bugs in drivers from affecting hypervisors)
- Offer a specific drivers to VMs (DomU) so as to simplify info reception at Dom0



Source: `http://www.xenproject.org/`

Age 2: Hardware Assisted Virtualization

Intel and AMD

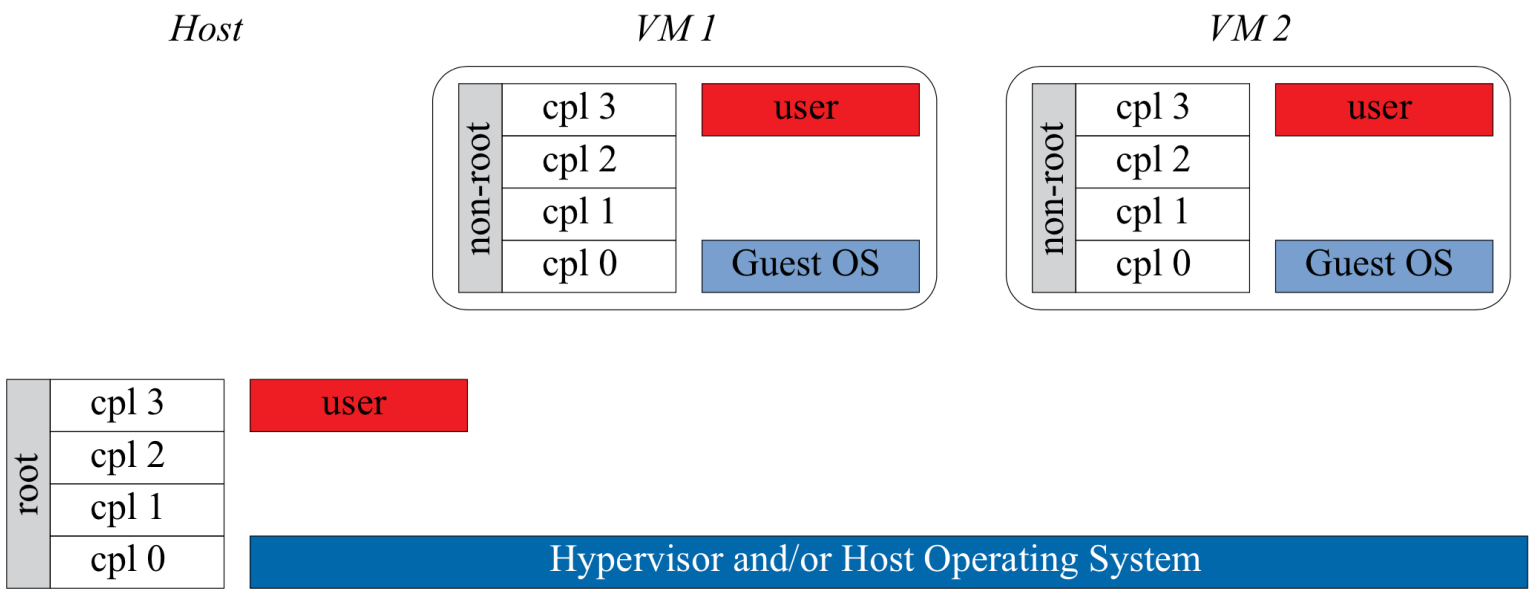
- x86 ISA common to Intel and AMD processors
- Intel Virtualization Technology (Intel VT)
- AMD Virtualization (AMD-V)
- Introduced in 2006...
- ....new functionalities introduced gradually.

Intel VT-x

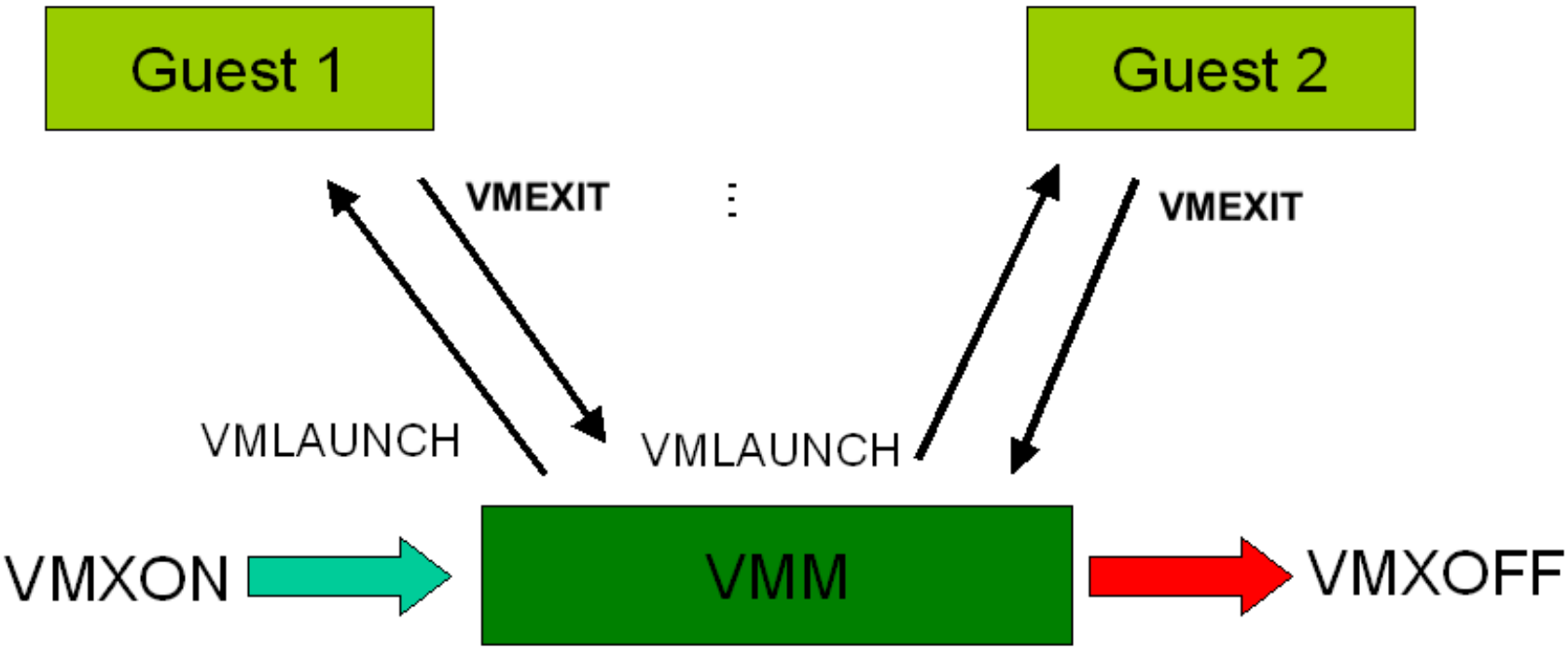
> A central design goal for Intel Virtualization Technology is to eliminate the need for CPU paravirtualization and binary translation techniques, and thereby enable the implementation of VMMs that can support a broad range of unmodified guest operating systems while maintaining high levels of performance.
>
> R. Uhlig et al., *IEEE Computer*, 2005 [171]

Intel VT-x philosophy

- Do not change the semantics of instructions, ie. don't remove the 17 offending instructions
- Duplicates the visible state of the processor and provide a new mode of execution : the root mode
  $\rightarrow$ hypervisor and VM on different parts of the processor.

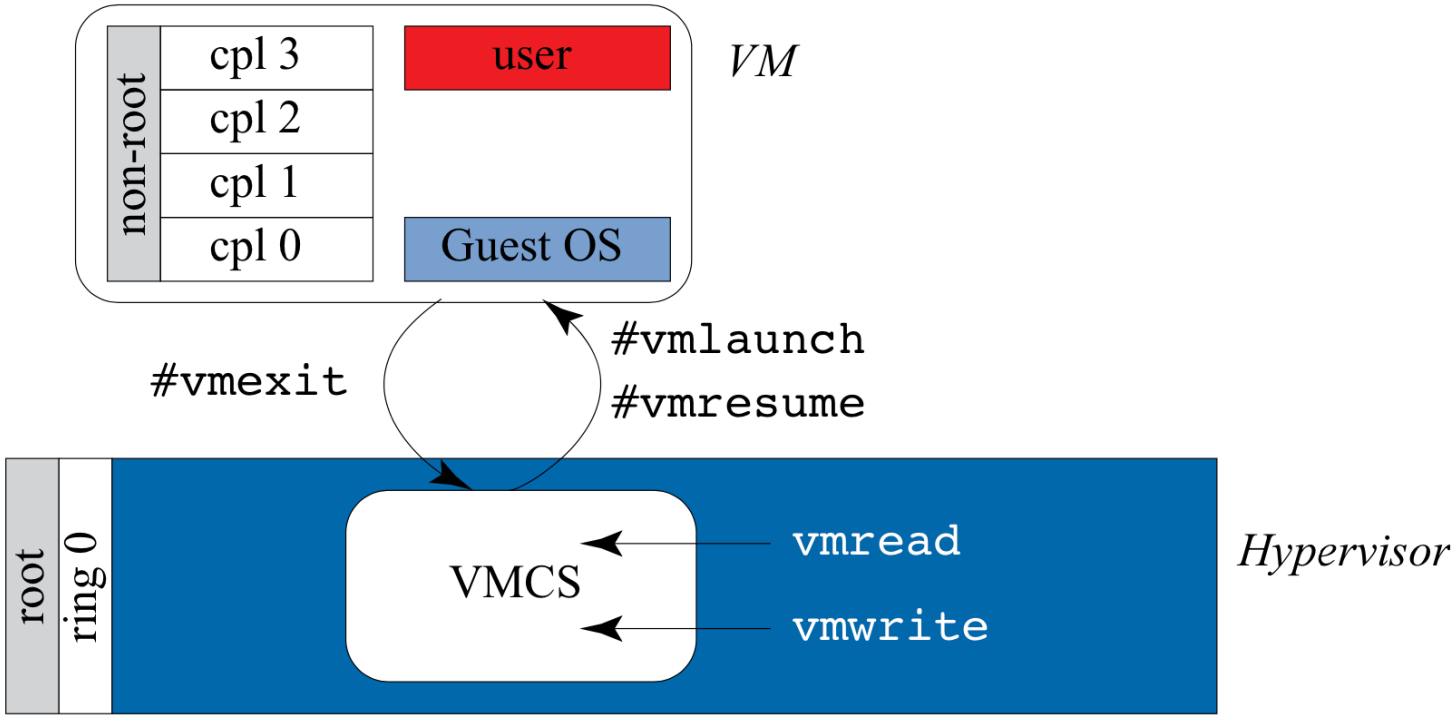


Intel VT CPU virtualization: The Root mode in action



Intel VT-x: provide the hypervisor info on the exited VM

VMCS plays the role (in HW) of the world switch (in SW)



Intel VT portfolio - cont'd

Memory Virtualization

- Non virtualized case: there exists already virtual pages (seen by programs) and real pages seen by OS.
- In virtualized cases, two levels of traductions:
  - Guest program $\rightarrow$ guest OS
  - Guest OS $\rightarrow$ host OS
- Intel offers extended page tables (EPT)
  - known under the name *SLAT, Second-Level Address Translation* in AMD

Intel Extended Page Tables

- Extended Page Tables (EPT) is an Intel x86 virtualization technology for the memory management unit (MMU), ie circuit in charge of translating virtual memory (seen by program) to real memory (on chipset)
- EPT support is found in Intel's Core i3, Core i5, Core i7 and Core i9 CPUs....

Intel VT portfolio - cont'd

I/O Virtualization

- E.g. enable :
  - packet processing offloading to network adapters (to compute checksums - a known performance issue at high speed)
  - Direct disk I/O
- Set of technologies:
  - Intel® Virtualization Technology for Directed I/O (VT-d)
  - Virtual Machine Device Queues (VMDQ),
  - Single Root I/O Virtualization (SR-IOV, a PCI-SIG standard)
  - Intel® Data Direct I/O Technology (Intel® DDIO) enhancements

Intel VT I/O Virtualization: example of DMA

- Guest OS would like to do DMA for its programs
  - DMA: controller of device allowed to directly copy its data to RAM $\rightarrow$ no additional work for CPU
- Problem: guest OS does not see physical pages (only hypervisor does)
- Use of I/O MMU $\rightarrow$ maps guest-physical address to host-physical addresses