# CS 444 Operating Systems

## Chapter 7 Virtualization and the Cloud, pp477-524

J.H.DeBlois

November 18, 2025

# Introduction

- Organizations sometimes use separate machines for email, Web server, etc.
- They may be in the same server rack, connected by a high-speed network, so a multicomputer.
- One reason for separate machines is load, but another is reliability, and another is security.
- It's also possible that multiple OS are needed.
- But all these machines are a large expense.
- To save money, a company could use virtual machine technology

# History, 7.1 (p480)

- As early as 1960, IBM experimented with hypervisors
- In 1974, two computer scientists at UCLA, Gerald Popek and Robert Goldberg, published a paper that listed conditions for a computer architecture to be able to support virtualization efficiently (Popek and Goldberg, 1974)
- This paper was mentioned in Section 1.7.5 Virtual Machines (pp 69-73), so you know it is important!
- "Famously" (p480) the well-known x86 architecture that also originated in the 1970s did not meet these requirements for decades

# History, Continued

- In 1999, VMware introduced its first virtualization solution for x86 (p481)
- OS-level virtualization traces back to 1979, so there is a separate environment on disk to which the process is confined
- Popularity of containers exploded with the launch of Docker in 2013, which uses OS-level virtualization to allow software to be packaged in containers that hold all the code, libraries and config files needed to run it

# Definitions

- Virtual Machine Monitor (VMM): creates the illusion of multiple virtual machines on the same hardware
- A VMM is also known as a Hypervisor
- There are type 1 and type2 hypervisors
- Type 1 hypervisor runs on bare metal
- Type 2 hypervisor runs on and uses all the services of an underlying OS

# Definitions, Continued

- Virtual Machine Monitor (VMM): creates the illusion of multiple virtual machines on the same hardware
- Pure Type 2 and Practical Type 2 are shown in Figure 1-29 (p71)
- Cloud: outsource your computation or storage needs
- OS-level virtualization: one OS offers multiple virtual environments, containers or jails

# Goldberg (1972) paper described Type 1 and Type 2 Hypervisors

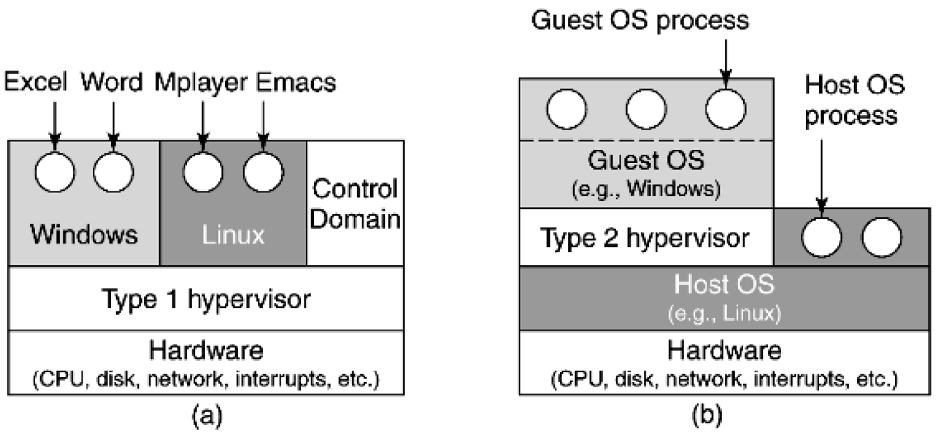


**Figure 7-1.** Location of type 1 and type 2 hypervisors.

- Type 1 Hypervisor, shown in Fig. 7-1a), is like an OS
- Type 2 Hypervisor, shown in Fig. 7-1(b), relies on a Host OS
- Both types must execute machine instructions in a safe manner.
- For example, an OS running on top of either hypervisor may change and even mess up its own page tables, but not those of others. (p485)

# Goldberg (1972) paper described Type 1 and Type 2 Hypervisors - larger

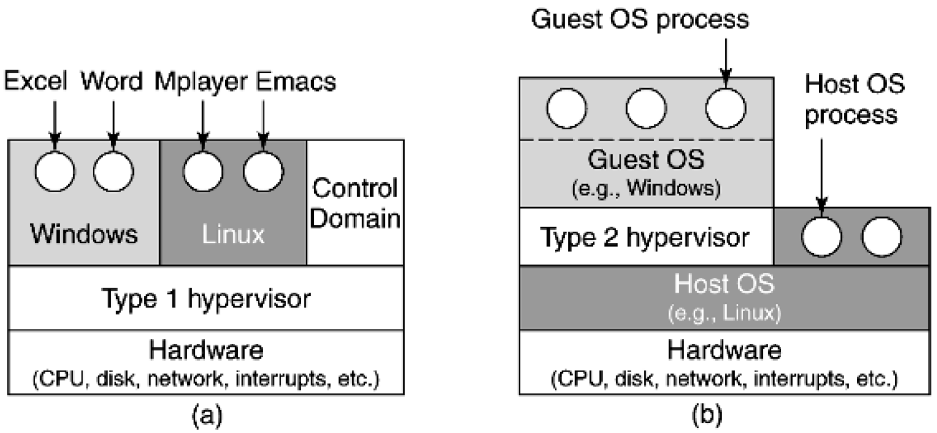


**Figure 7-1.** Location of type 1 and type 2 hypervisors.

# Popek and Goldberg (1974) paper described the problem with the instruction set of some CPUs

- When an OS running on a virtual machine (in user mode) executes a privileged instruction, such as modifying the PSW or doing I/O, it is essential that the hardware trap to the virtual-machine monitor so the instruction can be emulated in software. (p71)
- On some CPUs - notably the Pentium, its predecessors and its clones - attempts to execute privileged instructions in user mode are just ignored.
- This property made it impossible to have virtual machines on this hardware

# Virtualizability and Performance - virtual kernel mode

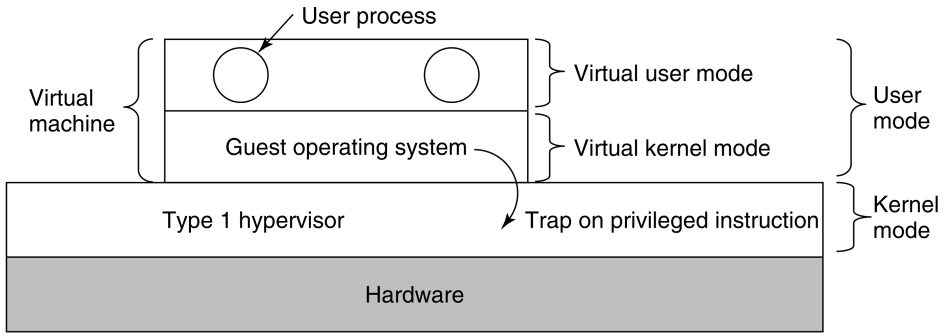


**Figure 7-3.** When the operating system in a virtual machine executes a kernel-only instruction, it traps to the hypervisor if virtualization technology is present.

- Type 1 Hypervisor runs on bare metal
- Virtual machine runs as a user process
- Virtual machine runs a guest OS that thinks it is in kernel mode (it's not)
- Virtual machine also runs user processes (in user mode)
- What happens when guest OS executes an instruction only allowed in kernel mode?

# Virtualizability and Performance - virtual kernel mode (2)

- Normally, on CPU without Virtual Technology (VT), the instruction fails and OS crashes
- On CPUs with VT, when the guest OS executes a sensitive instruction, a trap to the hypervisor does occur (see Fig. 7-3)
- The hypervisor inspects the instruction to see if it were issued by the guest OS in the virtual machine or by a user program in the virtual machine
- In the former case, it arranges for the instruction to be carried out
- In the latter case, it emulates what the real hardware would do when confronted with a sensitive instruction executed in user mode

# Virtualizability and Performance - What did people do before VT?

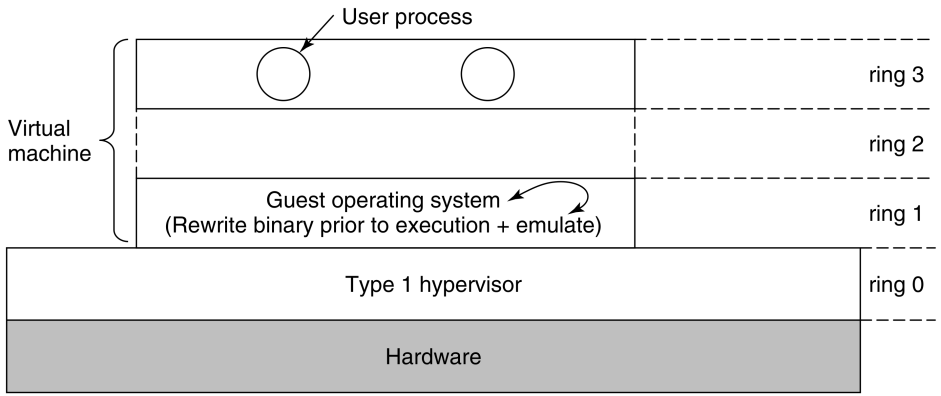


**Figure 7-4.** The binary translator rewrites the guest operating system running in ring 1, while the hypervisor runs in ring 0.

- VMware released a hypervisor well before the arrival of virtualization extensions on the x86
- Answer: software engineers made clever use of binary translations and hardware features that did exist on the x86, such as the processor's protection rings
- The x86 had four rings:
- Ring 3 is least privileged; ring 0 is most privileged
- Rings 1 and 2 were not used by any operating system

# Virtualizability and Performance, Continued

- As shown in Fig. 7-4 below, many virtualization solutions therefore kept the hypervisor in kernel mode (ring 0)
- And kept the applications in user mode (ring 3)
- But put the guest operating system in a layer of intermediate privilege (ring 1)
- Thus, the kernel is privileged relative to the user processes
- And any attempt to access kernel memory from a user program leads to an access violation
- At the same time, the guest OS's privileged instructions trap to the hypervisor
- The hypervisor does sanity checks and performs the instructions on the guest's behalf (p487)

# Virtualizability and Performance Figure 7-4 (p487)

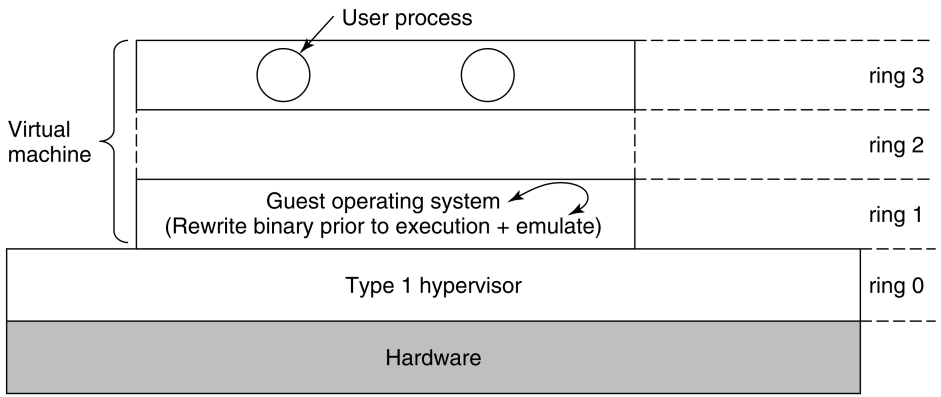


**Figure 7-4.** The binary translator rewrites the guest operating system running in ring 1, while the hypervisor runs in ring 0.

# Virtualizability and Performance, Continued again

- As for the sensitive instructions in the guest's kernel code: the hypervisor makes sure they no longer exist!
- To do so, it rewrites the code, one basic block at a time
- So if the block includes sensitive instructions they are replaced by a call to a hypervisor procedure that handles them
- Most code blocks do not contain sensitive or privileged instructions and thus can execute natively
- This can be done pretty fast

# Virtualizability and Performance, Continued again and again

- Same techniques are used for Type 2 hypervisors
- Most Type 2 hypervisors have a kernel module operating in ring 0 so they can manipulate the hardware with privileged instructions
- Thus: now it's clear how these hypervisors work, even on unvirtualizable hardware
- Sensitive instructions in the guest kernel are replaced by calls to procedures that emulate these instructions

# Difference between true virtualization and paravirtualization (p491)

- Paravirtualization is defined on p72 - "modify the [guest] OS by removing the control instructions" but this is not true virtualization
- Two virtual machines supported on VT hardware
- On the left, an unmodified version of Windows as guest OS
- On the right, a modified versionn of Linux that no longer has sensitive instructions, so it makes a hypervisor call to get the work done

# Difference between true virtualization and paravirtualization - Continued)

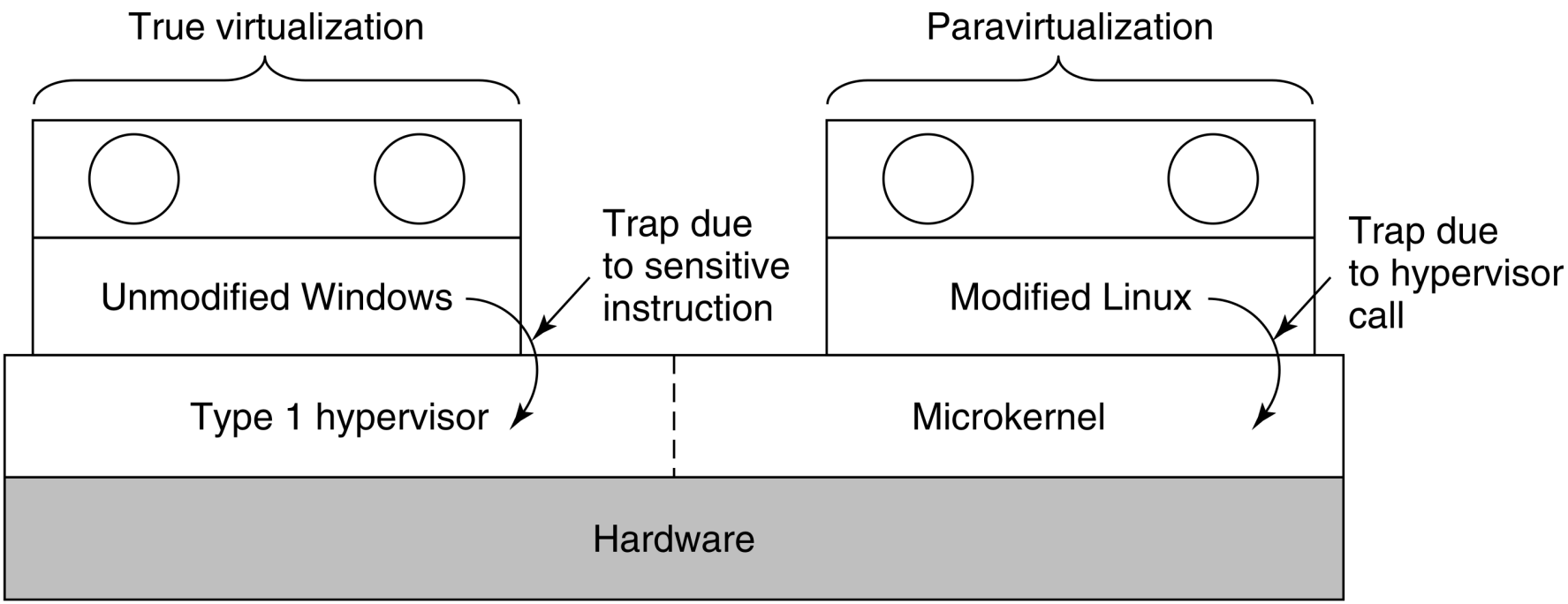


**Figure 7-5.** True virtualization and paravirtualization.

# Virtual Machine Interface (VMI) can help

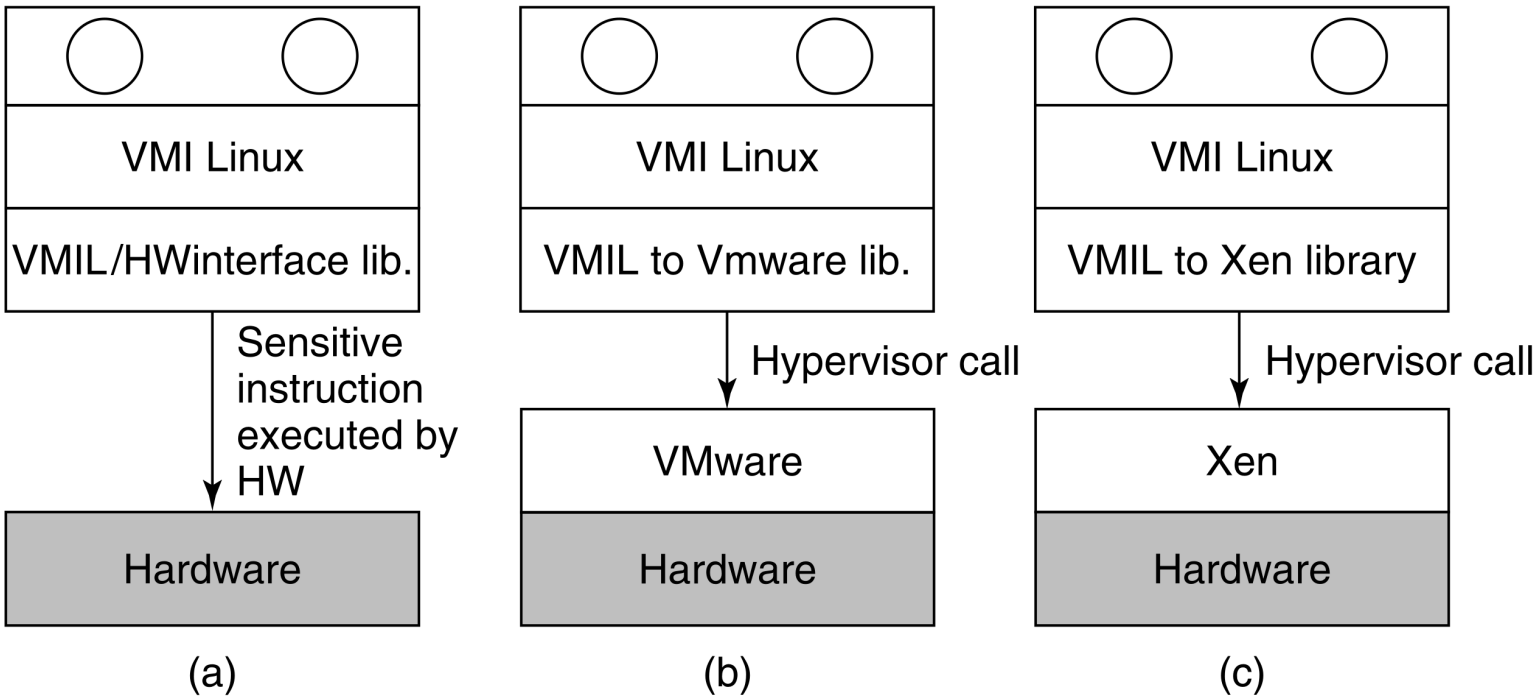


**Figure 7-6.** VMI Linux running on (a) the bare hardware, (b) VMware, and (c) Xen.

- When VMI Linux runs on bare hardware, it is linked to a library
- (a) The library issues the sensitive instructions
- When running on the hypervisor, the guest OS is linked with different libraries

# Memory Virtualization, 7.6 (p493)

- So far we have talked about how to virtualize the CPU
- But a computer has memory and I/O devices which also have to be virtualized
- Modern OSs nearly all support virtual memory - the mapping of pages in the virtual address space onto pages of physical memory
- The mapping is by (multi-level) page tables
- The OS sets a control register in the CPU that points to the top-level page table
- Memory Virtualization greatly complicates memory management!

# Memory Virtualization - Example

- Suppose a virtual machine is running, and the guest OS in it decides to map its virtual pages 7, 4 and 3 onto physical pages 10, 11 and 12 respectively
- It builds the page tables with this mapping and loads a hardware register to point to the top-level page table
- This instruction is sensitive: On a VT CPU, it will trap; with dynamic translation it will cause a call to a hypervisor procedure; on a paravirtualized OS, it will generate a hypercall
- Let us assume it traps into a Type 1 hypervisor, but the problem is the same in all three cases
- What does the hypervisor do now?

# Memory Virtualization - Example - (2)

- One solution is to actually allocate physical pages 10, 11 and 12 to this virtual machine and set up the actual page tables to map the virtual machines's virtual pages 7, 4 and 3 to use them
- Now suppose a second virtual machine starts and maps its virtual pages 4, 5 and 6 onto physical pages 10, 11 and 12 and load the control register to point to its page tables
- The hypervisor catches the trap, but what should it do?

# Memory Virtualization - Example - (3)

- It cannot use this mapping because physical pages 10, 11 and 12 are already in use
- It can find some free pages, say 20, 21 and 22, but first has to create new page tables mapping the virtual pages 4, 5 and 6 of virtual machine2 onto 20, 21 and 22
- So, for each virtual machine, the hypervisor needs to create a shadow page table that maps the virtual pages used by the virtual machine onto the actual pages the hypervisor gave it

# Memory Virtualization - Example - (4)

- AND every time the guest OS changes its page tables, the hypervisor must change the shadow page tables as well (p494)
- The guest OS can change page tables by just writing to memory so the hypervisor does not know about the change
- Many page faults ensue - which is costly
- In paravirtualized OS, the guest OS informs the hypervisor when done

# Extended/Nested Page Tables

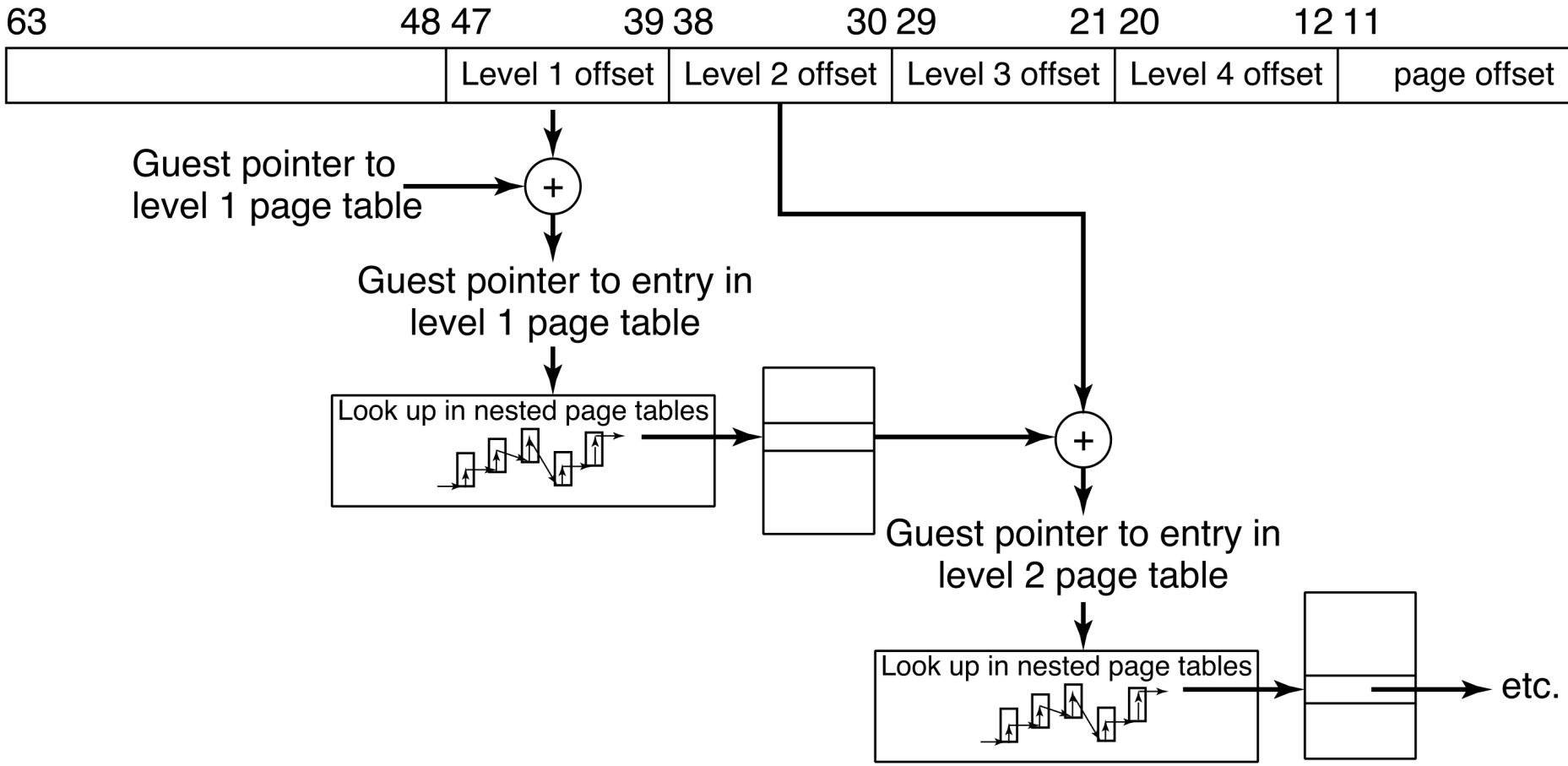


**Figure 7-7.** Extended/nested page tables are walked every time a guest physical address is accessed—including the accesses for each level of the guest's page tables.

# Hardware Support for Nested Page Tables

- The cost of handling shadow page tables led chip makers to add hardware support for nested page tables
- AMD calls them Nested Page Tables; Intel refers to them as Extended Page Tables
- Even without virtualization, the OS maintains a mapping between virtual pages and physical pages - we saw these in Ch3
- The hardware walks these page tables to find the physical address that corresponds to a virtual address

# Hardware Support for Nested Page Tables (2)

- In addition to guest virtual addresses, we now also have guest physical addresses
- And subsequently host physical addresses (called machine physical addresses)
- Without Extended Page Tables, the hypervisor is responsible for maintaining shadow page tables
- With EPT, the hypervisor still has an addiitonal set of page tables, but not the CPU is able to handle much of the intermediate level in hardware also
- The walking happens more often than you think: if the guest virtual address was not cached, it requires a full page-table lookup

# I/O Virtualization

- Now we look at how to virtualize the I/O
- The guest OS will probe the hardware to find I/O devices
- The probes will trap to the hypervisor
- What should the hypervisor do?

# I/O Virtualization (2)

- One approach is to report back that the disks, printers and so on are the ones that the hardware actually has
- Then the guest OS will try to use them
- But there is a problem: Each guest OS thinks it owns an entire disk partition and there may be more virtual machines than disk partitions

# I/O Virtualization (3)

- The hypervisor could then do translations of block numbers
- Or the disk to use could be different from the real one
- Maybe a high performance RAID
- So virtual machines can remap hardware devices (we saw this during our field trip to the server room)
- Hypervisor could also act as a switch and use MAC address of each virtual machine and switch frames from one virtual machine to another
- Use of DMA, which uses absolute memory addresses, must also have addresses remapped

# Case Study: VMWare, Inc., 7.11 (p507)

- VMware Workstation, released in 1999, was the first virtualization product for 32-bit x86 computers.
- Mainframes are vertically integrated so a single vendor (like IBM) engineers the whole stack.
- But the x86 industry is disaggregated: a) Intel and AMD make the processors; b) Microsoft offers Windows and the open source community offers Linux, c) other companies build I/O devices, peripherals and device drivers; and system integrators such as HP and Dell put together computer systems for retail sales.

# VMware Virtual Machine Monitor - high-level components

- Figure 7-8 (p513) shows the modular building blocks of the original VMware VMM.

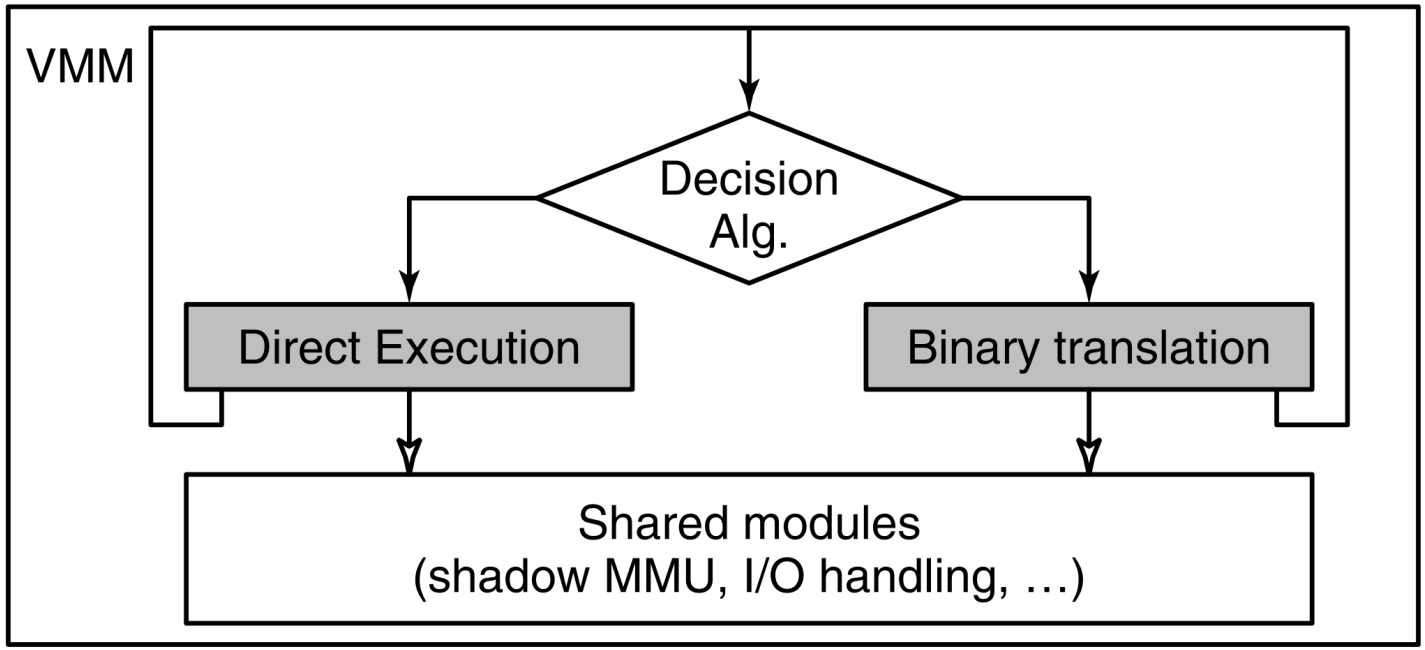


**Figure 7-8.** High-level components of the VMware virtual machine monitor (in the absence of hardware support).

- Both subsystems rely on some shared modules
- Direct execution is preferred
- Binary translation is the fallback

# VMware Virtual Machine Monitor - when binary translation is used

- Binary translation must be used if any of the following were true:
- The virtual machine is currently running in kernel mode (ring 0 in the x86 architecture).
- The virtual machine can disable interrupts and issue I/O instructions (in the x86 architecture, when the I/O privilege level is set to the ring level).
- The virtual machine is currently running in real mode, a legacy 16-bit execution mode used by the BIOS among other things.

# VMware virtual hardware configuration options (p516)

| | Virtual Hardware (front end) | Back end |
|---|---|---|
| Multiplexed | 1 virtual x86 CPU, with the same instruction set extensions as the underlying hardware CPU | Scheduled by the host operating system on either a uniprocessor or multiprocessor host |
| | 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 |
| | 4x IDE disks<br>7x Buslogic SCSI Disks | Virtual disks (stored as files) or direct access to a given raw device |
| | 1x IDE CD-ROM | ISO image or emulated access to the real CD-ROM |
| | 2x 1.44 MB floppy drives | Physical floppy or floppy image |
| | 1x VMware graphics card with VGA and SVGA support | Ran in a window and in full-screen mode. SVGA required VMware SVGA guest driver |
| | 2x serial ports COM1 and COM2 | Connect to host serial port or a file |
| | 1x printer (LPT) | Can connect to host LPT port |
| | 1x keyboard (104-key) | Fully emulated; keycode events are generated when they are received by the VMware |

# The VMware Hosted Architecture and Components (p518)

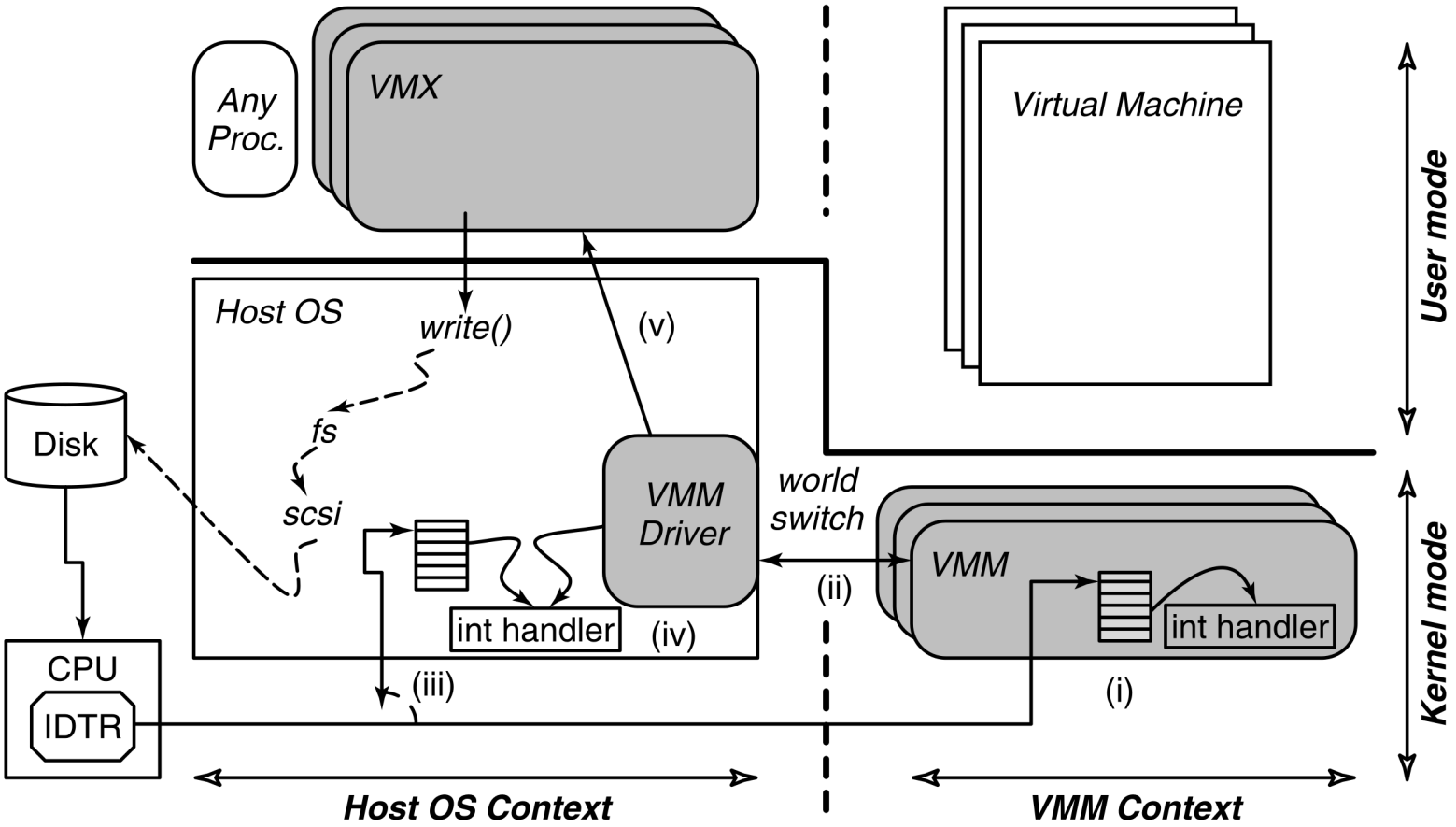


**Figure 7-10.** The VMware Hosted Architecture and its three components: VMX, VMM driver, and VMM.

# Normal Context Switch vs. VMware World Switch (p520)

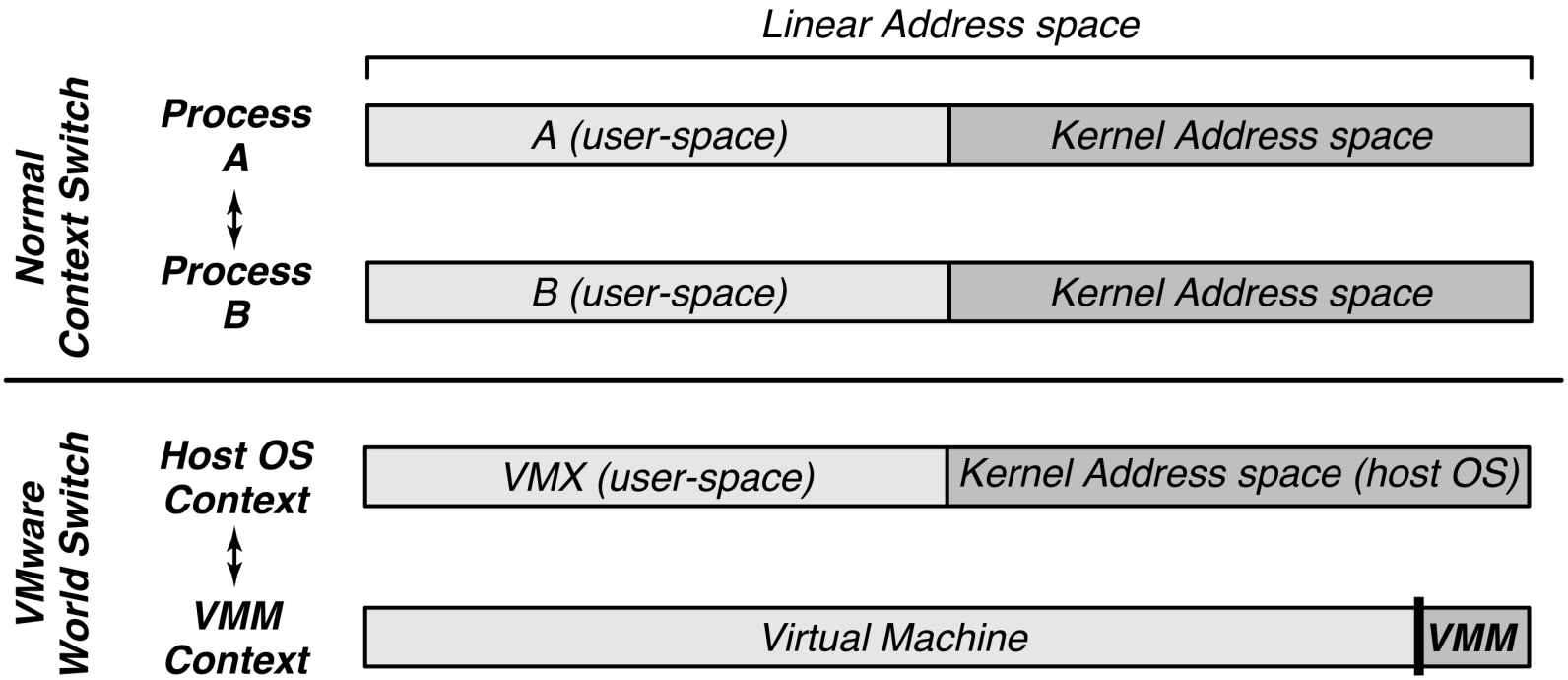


**Figure 7-11.** Difference between a normal context switch and a world switch.

- A regular context switch between processes swaps user portion of the address space and registers of the two processes, but leaves some system resources unmodified - like kernel portion of addr space
- World switch changes entire address space, exception handlers, privi. registers, etc.

# ESX Server: VMware's type 1 Hypervisor

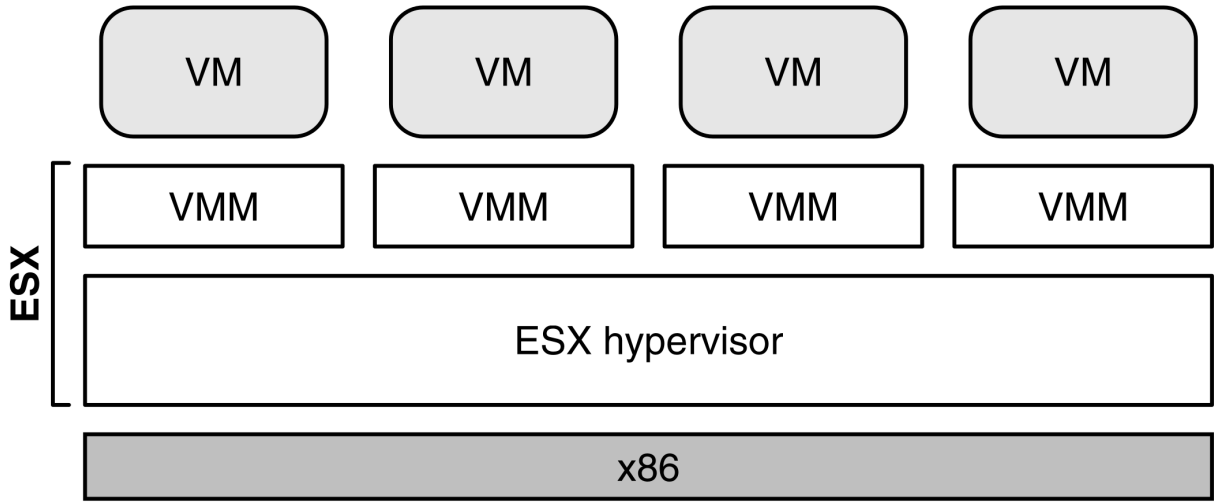


**Figure 7-12.** ESX Server: VMware's type 1 hypervisor.

- Released in 2001
- Aimed at the server marketplace
- The type 1 solution ran directly on the hardware (p522)

# Research on Virtualization and the Cloud

- Virtualization technology and cloud computing are both extremely active research areas. (p523)
- Hardware support for virtualization is now present in nearly all relevant CPU architectures.
- Cloud is seeing a shift from being a platform for tenants to run virtual machines (specified by a virtual disk image) to a platform used by tenants to run containers specified as Dockerfiles and coordinated by orchestrators such as Kubernetes. Often the container runs within a virtual machine but the guest operating system is now run by the cloud provider.

# Summary

- "Virtualization is the technique of simulating a computer, but with high performance."
- "Typically, one computer runs many virtual machines at the same time."
- "This technique is widely used in data centers to provide cloud computing."
- "In this chapter, we looked at how virtualization works, especially for paging, I/O and multicore systems."
- "We also studied an example: VMWare."