Virtual Machine Monitors

Operating Systems

Baochun Li
University of Toronto

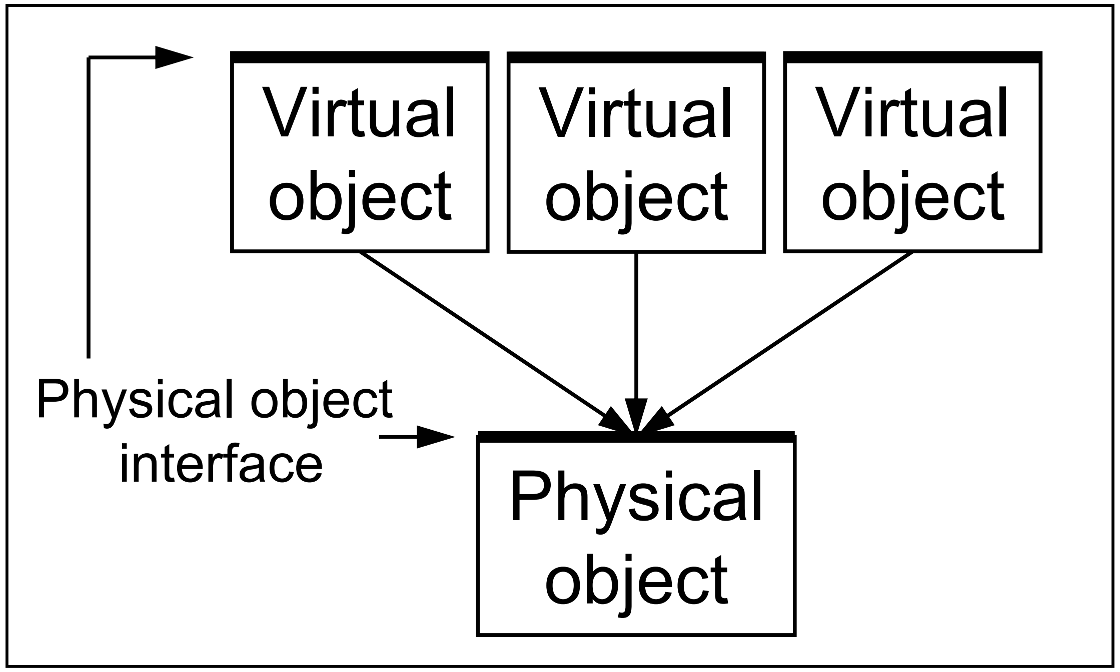
The role of virtualization

Virtualization of a physical object refers to simulating the interface of the physical object, while allowing

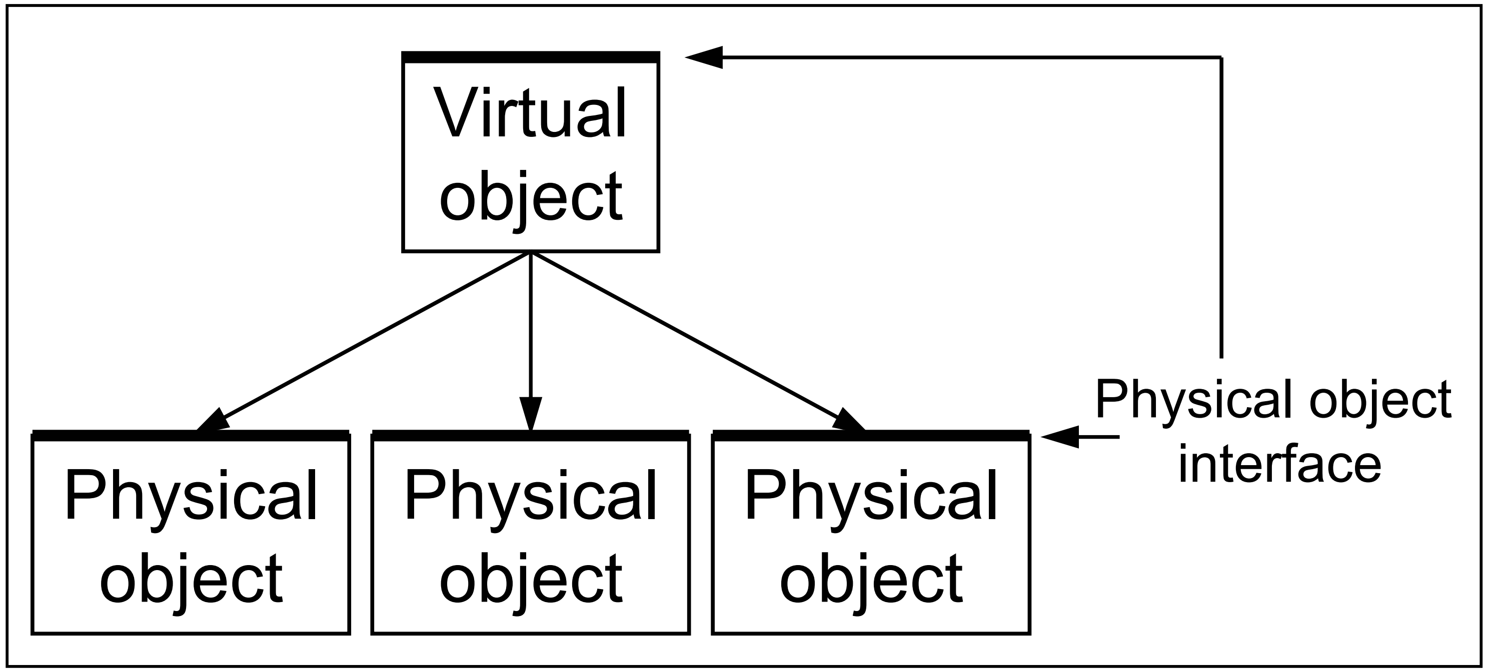
- **Multiplexing** of one physical object across many virtual objects
- **Aggregation** of multiple physical objects into one virtual object

Virtualization aims to preserve physical object interface

- Virtual object behaves the same as the physical object



Multiplexing



Aggregation

Virtual Machines — Intuitive idea

The Operating System provides two functions —

- Multiplexing: managing multiple programs sharing a common pool of resources (processor, memory, disk space)
- Convenient interface to hardware: a common API — called system calls — to all applications

What if these two functions can be cleanly separated?

- So that a bug in a device driver will not affect the entire OS

Virtual Machine Monitors (VMMs)

VMMs are software, similar to operating systems

Provide an interface that is an exact replica of the underlying physical machine, called virtual machine (VM)

Each VM has CPU, memory, disk and network like a physical machine, runs its own OS

OS thinks it is running directly on hardware!

Benefits

VMM software is simpler than OS, is bug-free software (we hope), only provides multiplexing, and protects VMs from each other

Bug in an OS affects applications running on one VM only

Motivation: Why Virtual Machine Monitors?

Servers — consolidate multiple OSes onto fewer hardware platforms

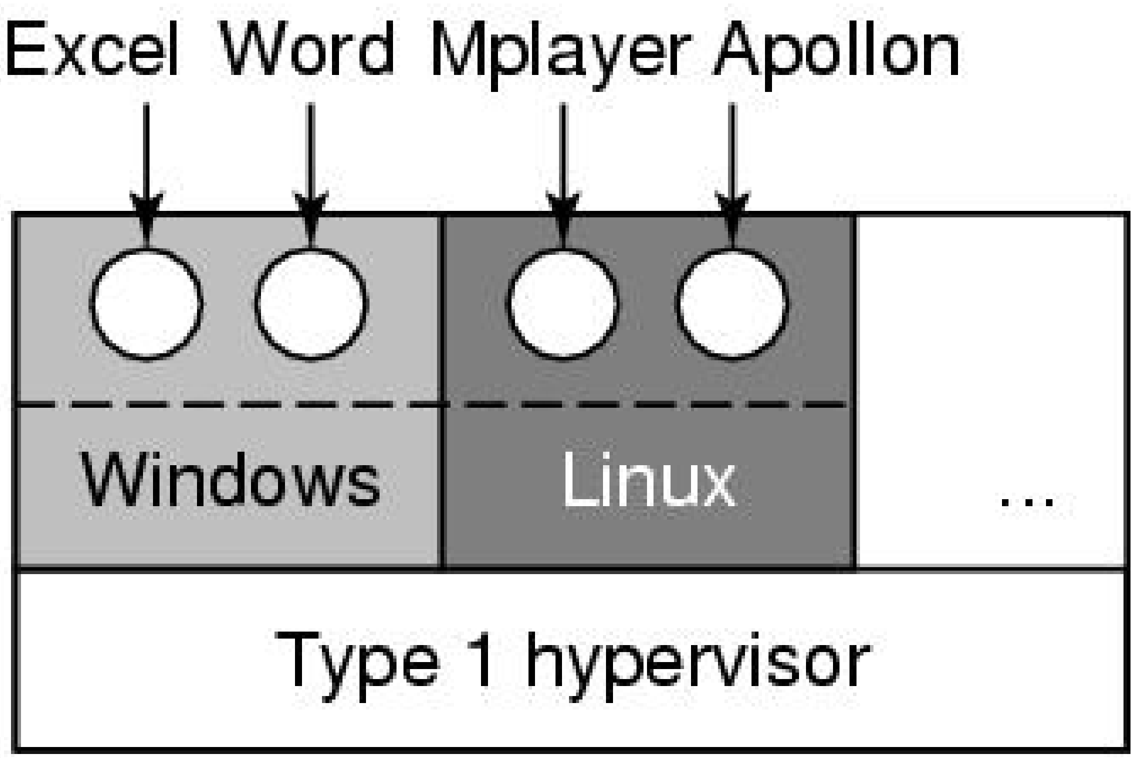
- Easier to manage and lower hardware costs
- Each of these virtual machines are lightly utilized

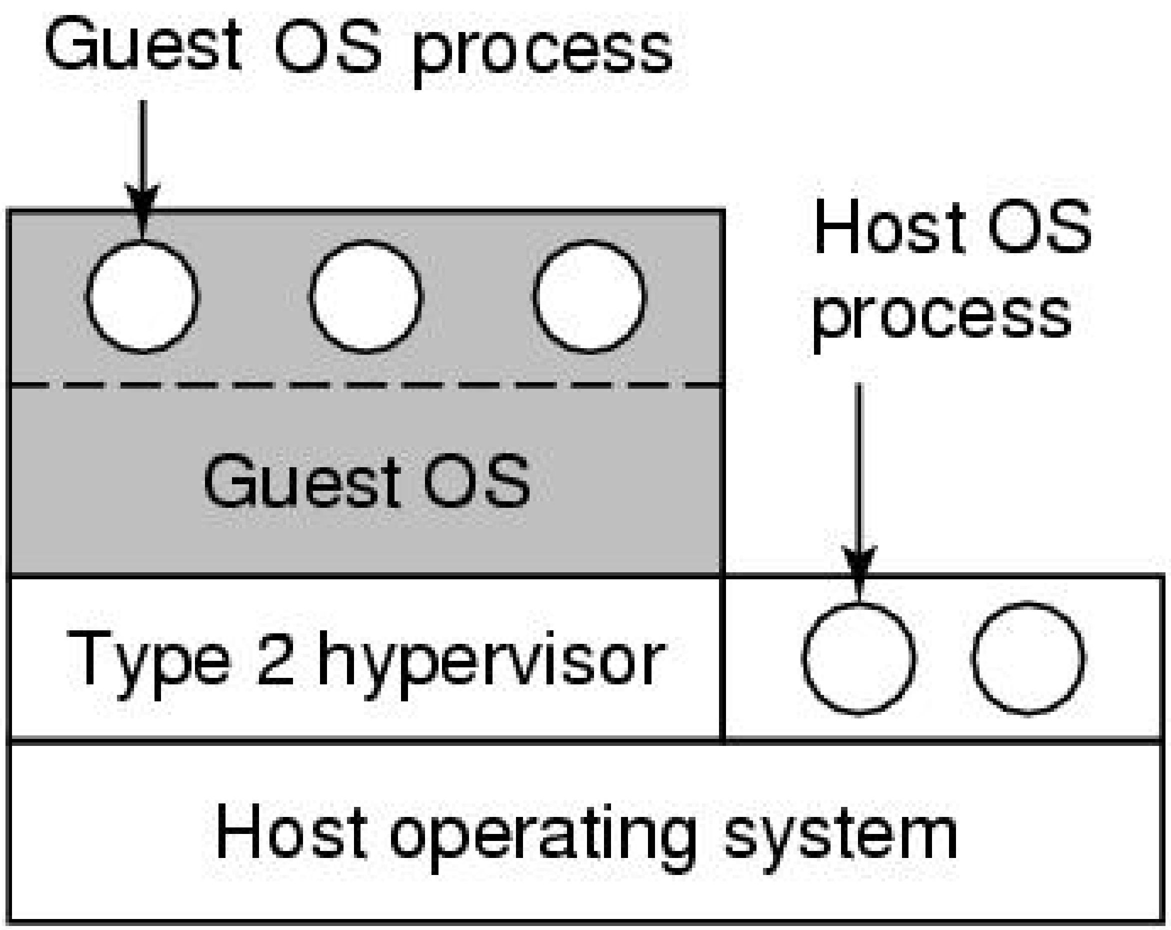
Desktops — Convenience of running Windows applications on a Mac

VMMs are also called “Hypervisors”

Type 1 hypervisor: VMM software runs in kernel mode on the physical machine

Type 2 hypervisor: VMM is a just a user-level program running on another OS (called the **host OS**)





Requirements for Virtualization

A **sensitive instruction** can only be executed in kernel mode

A **privileged instruction** causes a trap if executed in user mode

An architecture is **virtualizable** when all sensitive instructions are privileged instructions

- Cause traps when executed at lower privilege levels
- Essentially, allows running an OS in user mode

Non-sensitive instructions do not reveal or modify privilege level and can be run directly on physical machine at full speed

- Much faster than full emulation (e.g., BLITZ)
- However, virtualized program, i.e., OS, must use the same instruction set as physical machine

Bad news about Intel x86

Traditionally, Intel x86 does not meet virtualizability requirements

Certain sensitive instructions do not cause trap when executed in the user mode

Only rectified since December 2005

Virtualization Technology (VT-x) was introduced with Intel Pentium 4 CPUs 662/672 and later architectures

Implementing Type 1 Hypervisors

Limited direct execution

Sensitive instructions will be handled by VMM on behalf of the OS inside each VM

VM executes sensitive instruction

Causes trap, transfers control to VMM

VMM determines whether trap was issued by one of

Guest OS: Implements the OS request

Program running on Guest OS: emulates behaviour of sensitive instruction executing in user mode — jump to guest OS

Virtualizing the CPU

Without virtualization: OS will perform context switch between user processes

With virtualization: VMM will perform “machine switch” across VMs (each with its own OS)

Implementing System Calls

System calls without virtualization —

Process	Operating System
1. System call: Trap to OS	
	2. OS trap handler: Decode trap and execute appropriate syscall routine; When done: return from trap
3. Resume execution (@PC after trap)	

Implementing System Calls

System calls with virtualization —

Process	Operating System	VMM
1. System call: Trap to OS		
		2. Process trapped: Call OS trap handler (at reduced privilege)
	3. OS trap handler: Decode trap and execute syscall; When done: issue return-from-trap	
		4. OS tried return from trap: Do real return from trap
5. Resume execution (@PC after trap)		

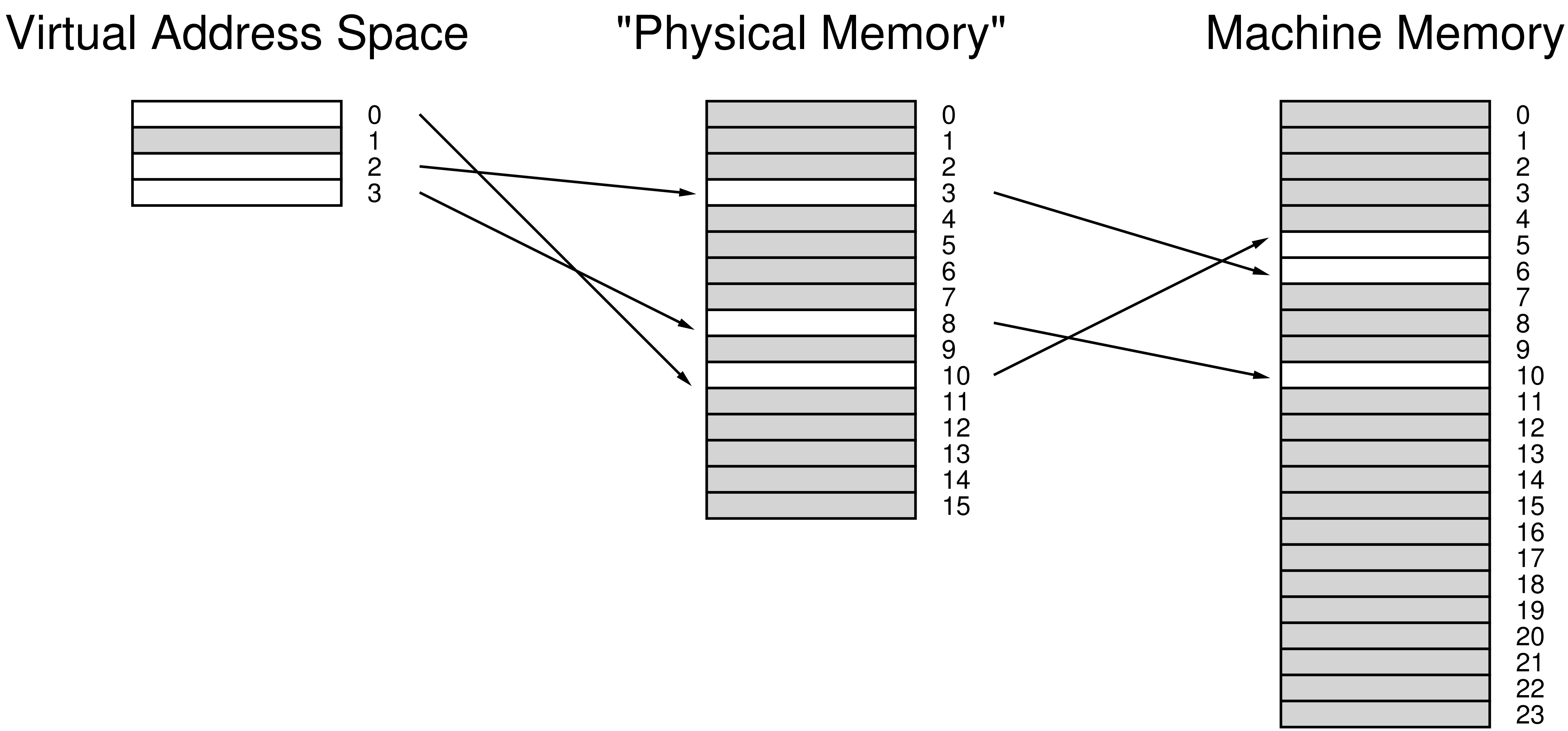
Virtualizing Memory

OS Page Table

VPN 0 to PFN 10
VPN 2 to PFN 03
VPN 3 to PFN 08

VMM Page Table

PFN 03 to MFN 06
PFN 08 to MFN 10
PFN 10 to MFN 05



TLB Miss without Virtualization

Process	Operating System
1. Load from memory: TLB miss: Trap	
	2. OS TLB miss handler: Extract VPN from VA; Do page table lookup; If present and valid: get PFN, update TLB; Return from trap
3. Resume execution (@PC of trapping instruction); Instruction is retried; Results in TLB hit	

TLB Miss with Virtualization

Process	Operating System	Virtual Machine Monitor
1. Load from memory TLB miss: Trap		
		2. VMM TLB miss handler: Call into OS TLB handler (reducing privilege)
	3. OS TLB miss handler: Extract VPN from VA; Do page table lookup; If present and valid, get PFN, update TLB	
		4. Trap handler: Unprivileged code trying to update the TLB; OS is trying to install VPN-to-PFN mapping; Update TLB instead with VPN-to-MFN (privileged); Jump back to OS (reducing privilege)
	5. Return from trap	
		6. Trap handler: Unprivileged code trying to return from a trap; Return from trap
7. Resume execution (@PC of instruction); Instruction is retried; Results in TLB hit		

What we've covered so far

Three Easy Pieces: Appendix B (Virtual Machine Monitors)