

CSE 120

Operating Systems Principles

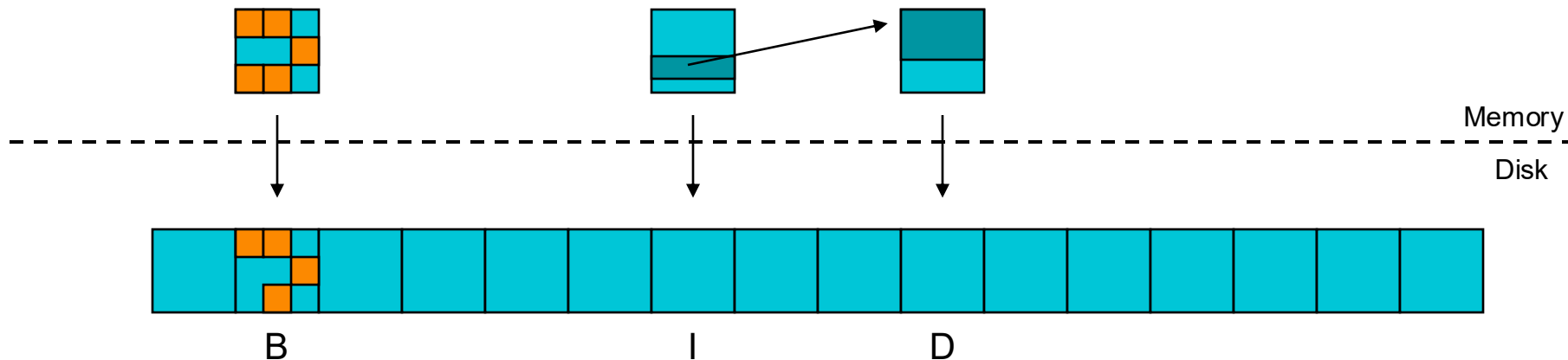
Spring 2025

Lecture 17: Virtual Machines

Amy Ousterhout

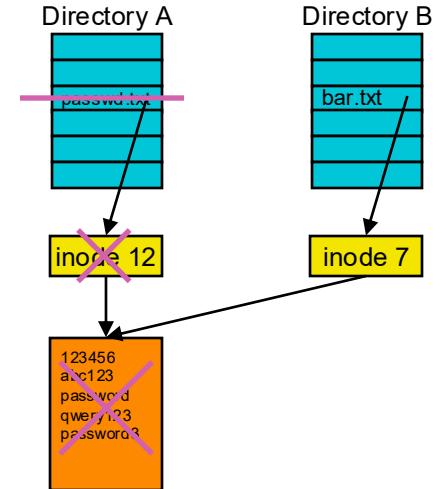
Crash Consistency Problem

- File system operations may involve writing multiple blocks
- Crashes can happen at any time
 - The file system may be left in an **inconsistent state**
- Goal: ensure that updates to the file system occur **atomically**



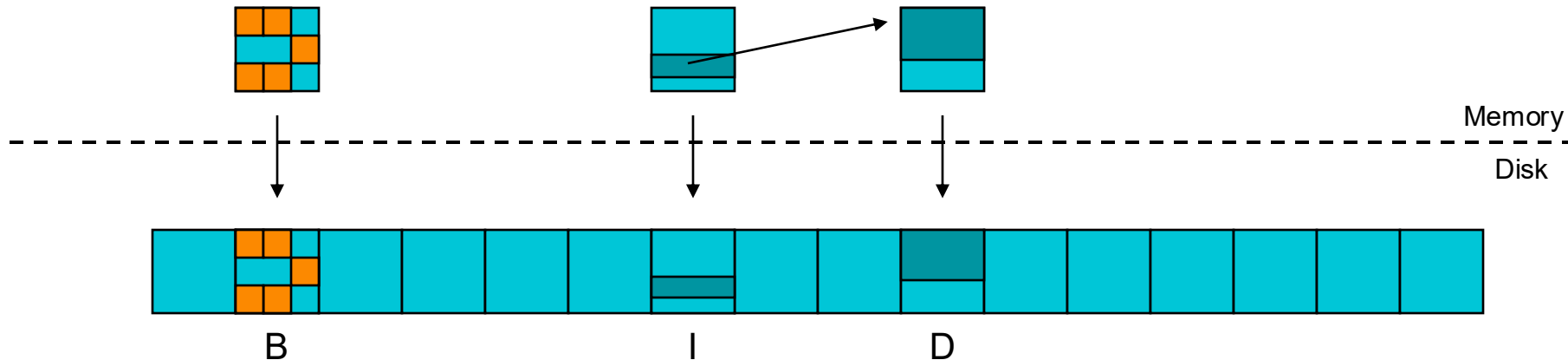
Check and Repair with fsck

- **fsck**: the file system checker in Unix
- When system boots:
 - Scan the file system for inconsistencies
 - Repair inconsistencies or notify an admin
 - Example repairs:
 - » Inode points to a data block that is marked as free – update the bitmap
 - » Inode reference count differs from number of links – update reference count
- Cons:
 - Very slow! Can take hours to run on large disk volumes
 - May not restore data to a valid state
 - May pose security issues



Ordered Writes

- Goal: prevent inconsistencies by **writing the blocks to disk in a safe order**
- Rules for how to order writes:
 - Initialize blocks before writing pointers that point to them
 - Nullify existing pointers to a block before reusing it
 - Set a new pointer to a resource before clearing the last one (e.g., with `mv`)



Ordered Writes

- Pro:
 - No need to wait for fsck on reboot
- Cons:
 - Can leak resources (run fsck in the background)
 - Must wait for writes to complete, slows down operations
 - Difficult to find a safe interruptible ordering for all operations

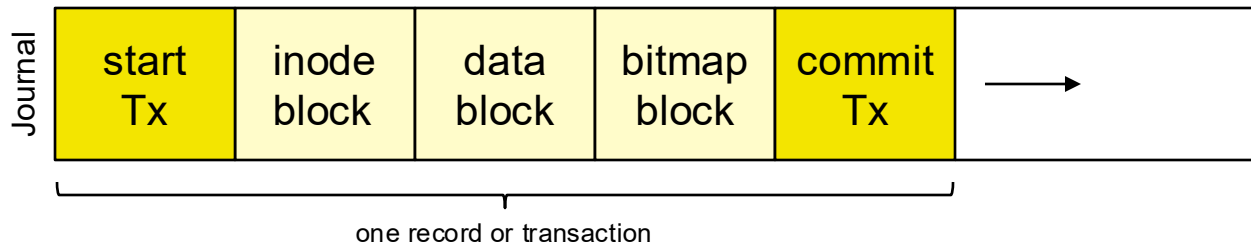
Journaling

- Also referred to as **write-ahead logging**
- Idea: write down what you are going to do before you do it
 - Record this information in a special append-only log file on disk
 - Flush this log to disk before modifying other blocks
 - When a crash occurs, replay the log to make sure all updates completed

Add data block #2953 to inode #438 at index #7	Remove data block #125 from inode #215 at index #9	...
--	--	-----

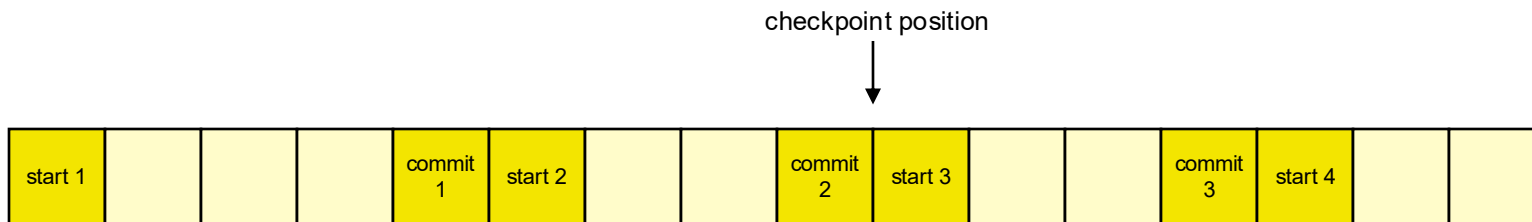
Journal

- Journal: an append-only file containing log records
 - Start transaction
 - » Transaction ID, block addresses, etc.
 - Blocks
 - Commit transaction
 - » Transaction has committed – updates will survive a crash
 - » The transaction is only final after the commit block is written



Journaling

- Checkpointing
 - Record the current position in the log
 - Copy all modified blocks to final destinations (“home locations”) on disk
 - Can clear the log before the recorded position
- Crash recovery
 - Replay all transactions that have committed but are not checkpointed
 - Discard uncommitted transactions



Journaling

- Pros:
 - Recovery is much faster than running fsck
 - Eliminates inconsistencies
 - Log is written sequentially so writes are fast (no seeks)
 - Can delay writes to improve performance
- Cons:
 - Have to write the data twice

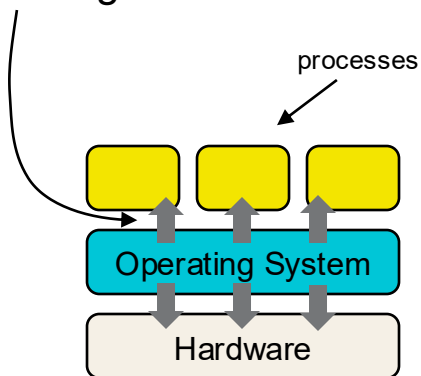
Abstractions for Virtualization

- Processes
- Virtual machines
 - Types of Virtual Machine Monitors
 - Virtualization components
 - » Processor
 - » I/O
 - » Memory
- Containers

Processes vs. Virtual Machines

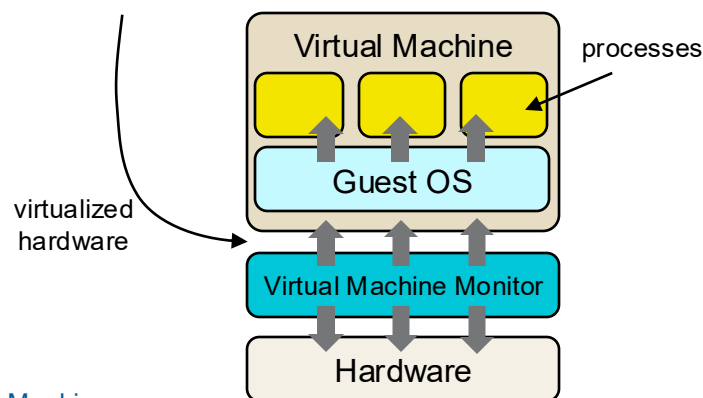
- Abstractions for processes:

- Virtual memory
- System calls
- Most instructions in the ISA
- Most registers



- Abstractions for virtual machines:

- Physical memory
- Interrupts
- All instructions in the ISA
- All registers
- I/O devices

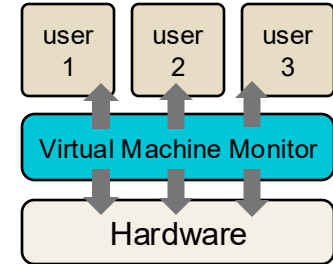
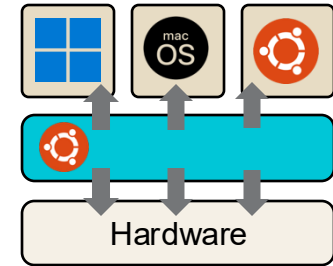
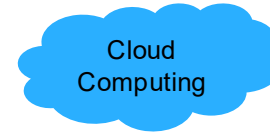


Virtual Machine Monitor (VMM)

- A **VMM** virtualizes an entire physical machine
- Presents a hardware interface to the guest OSes above
- Provides the **illusion** to each guest OS that it has full control of the hardware
 - Actually, the VMM controls the hardware
- Manages resources among multiple **virtual machines** (VMs)
- Isolates VMs from each other
- Also called a **hypervisor**

Why Virtual Machines?

- **Software use**
 - Can run software developed for different OSes
- **Development and testing**
 - Can test apps on different OSes, or test different OSes
- **Isolation**
 - Bugs or compromises in one VM cannot affect another VM
- **Efficiency and cost reduction**
 - Can share one machine among multiple users
 - Can migrate VMs from one machine to another



VMM Goals

- **Fidelity**
 - Guest OSes and applications work the same without modification (although we may modify the guest OS a bit)
- **Manageability**
 - Creation, maintenance, administration, provisioning
- **Isolation**, like separate physical machines
 - VMM protects resources and VMs from each other
- **Performance**
 - Minimize the overhead of running in a VM
- **Scalability**
 - Support many VMs at once

Virtual Machine Monitors



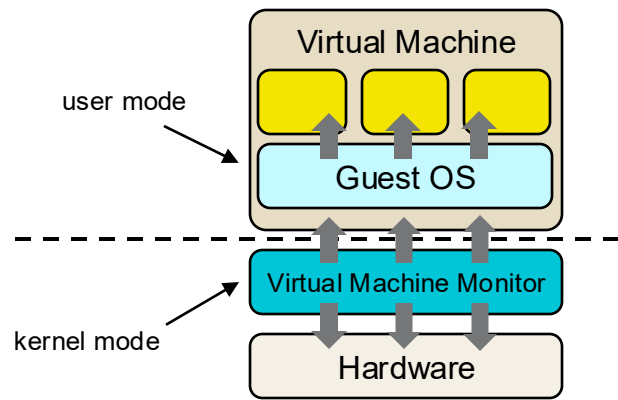
Virtualization via Simulation

- VMM can simulate instruction execution
 - Simulate memory, I/O, etc.
- Examples:
 - Bochs
 - Nachos – simulates a MIPS processor
- Con:
 - Very slow



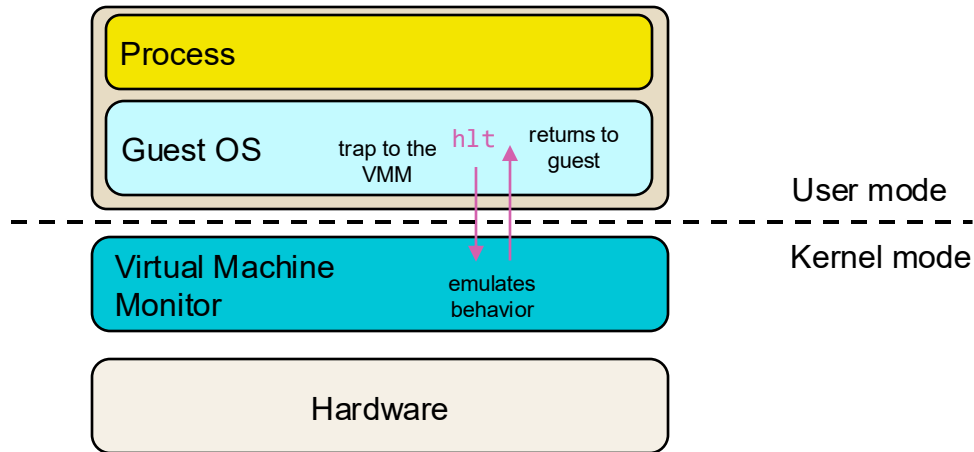
VMM Approach

- Run the VMM in kernel mode
- Run the guest OS in user mode
 - Most instructions execute at regular CPU speed
- What happens when the guest OS takes privileged actions?
 - E.g., issues I/O operations, halts the CPU
 - Causes a trap to the VMM!
- VMM simulates the appropriate behavior
 - Known as **trap-and-emulate**

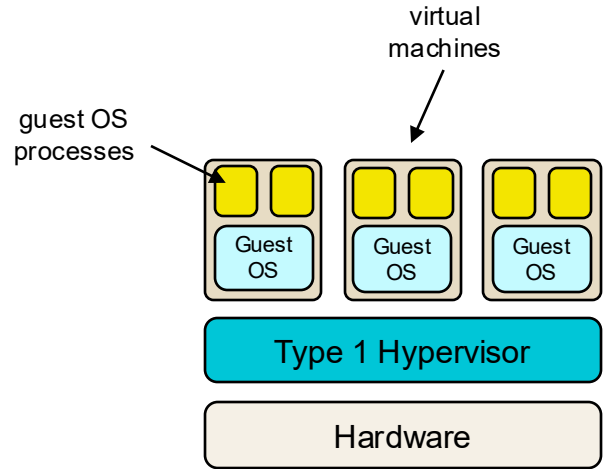


Trap and Emulate Example: Halt Instruction

- Guest OS issues a `hlt` instruction
 - Puts the CPU in idle mode

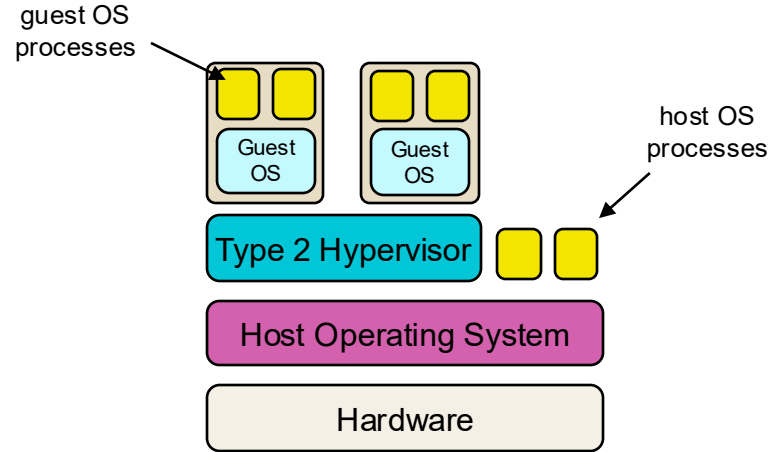


Type 1 and Type 2 Hypervisors



Type 1

(bare metal hypervisor)



Type 2

(hosted hypervisor)



Virtualizing the x86 Architecture

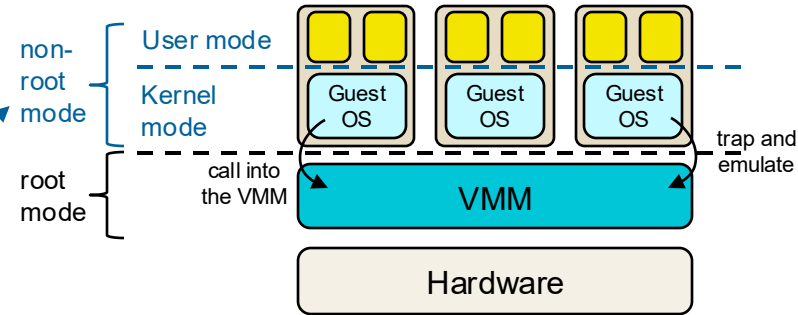
- Challenge of the trap-and-emulate approach:
 - x86 architecture was not fully virtualizable
- Problems:
 - Some privileged instructions behave differently when run in unprivileged mode
 - » popf does not trap when it cannot modify system flags
 - Hardware-managed TLB
 - » VMM cannot easily interpose on a TLB miss

Virtualizing the x86 Architecture

- **Paravirtualization**
 - Change the guest OS to better cooperate with the VMM
 - E.g., VMM can provide a “hypervisor API” so guest can perform certain functions
 - Sacrifices transparency for better performance
- **Binary translation**
 - Run guest OS code under control of a **binary translator**
 - Rewrites privileged instructions with emulation at runtime (may trap to VMM)
 - Incurs overhead, but can be kept small
- **Hardware support**
 - Intel and AMD added virtualization support in 2005 (Intel VT-x, AMD-V)

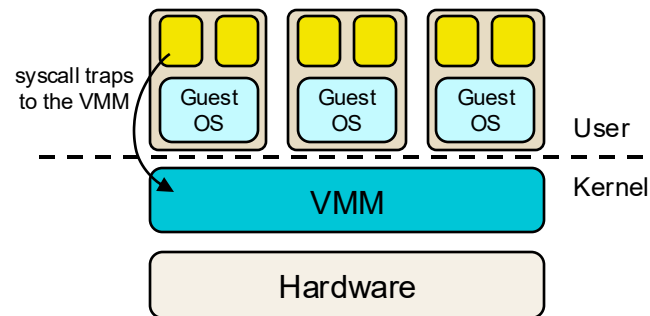
Virtualizing Privileged Instructions

- For privileged instructions that cause a trap (when executed at user level):
 - Trap to VMM, handle the instruction, return to guest OS or process (trap and emulate)
- For privileged instructions that do not trap:
 - With paravirtualization – modify guest OS to call into the VMM
 - With binary translation – rewrite guest OS instructions to emulate or call into VMM
 - With hardware support – add a new CPU mode and instructions to support trap-and-emulate



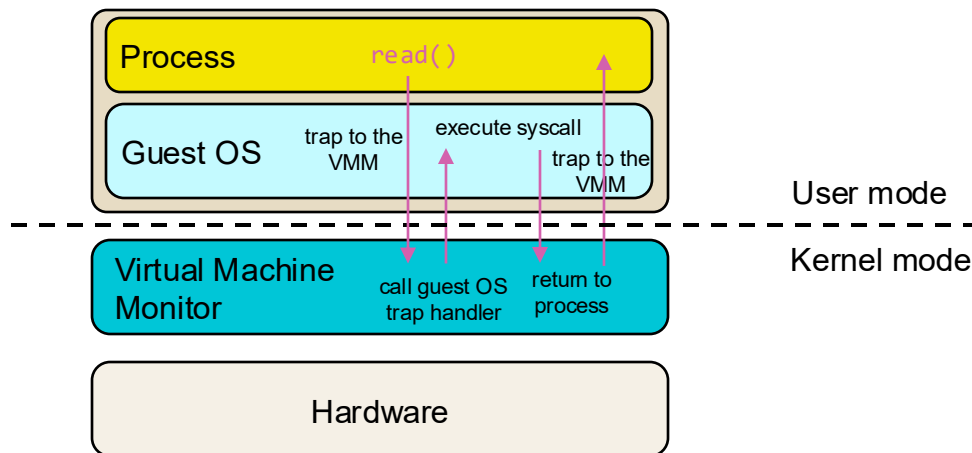
Virtualizing Events

- VMM receives interrupts and exceptions (faults and syscalls)
- Need to vector events to the correct VM
 - With paravirtualization – VMM notifies guest OS using an event queue
 - With full virtualization – call into the guest OS from VMM
 - With hardware support – hardware delivers events directly to the guest OS



Virtualizing Events Example: System Call

- Process invokes a system call (e.g., `read()`)
- Assume full virtualization (no paravirtualization or hardware support)



Poll – Disk Latency

- Which of the following would be most likely to decrease the latency of disk operations for an application with infrequent disk operations?
 - A: increase the density of data on the disk *improves transfer time, which is already small*
 - B: improve the disk scheduling policy *only helps when there are multiple requests queued*
 - C: increase the time per rotation of the disk *increases rotation time and latency*
 - D: design a disk arm that moves faster *improves seek time*
 - E: use multiple threads to perform multiple disk operations simultaneously
*doesn't improve the latency of an individual operation,
only improves throughput with multiple concurrent requests*

Disk latency = **seek** + **rotation** + transfer

Poll – Soft Links

- Suppose that “/one” is a soft link to “/two”. How many blocks will the file system read in order to open and read the first byte of “/two” via the soft link “/one”? Assume that there is no file buffer cache and that each directory fits in a single data block.

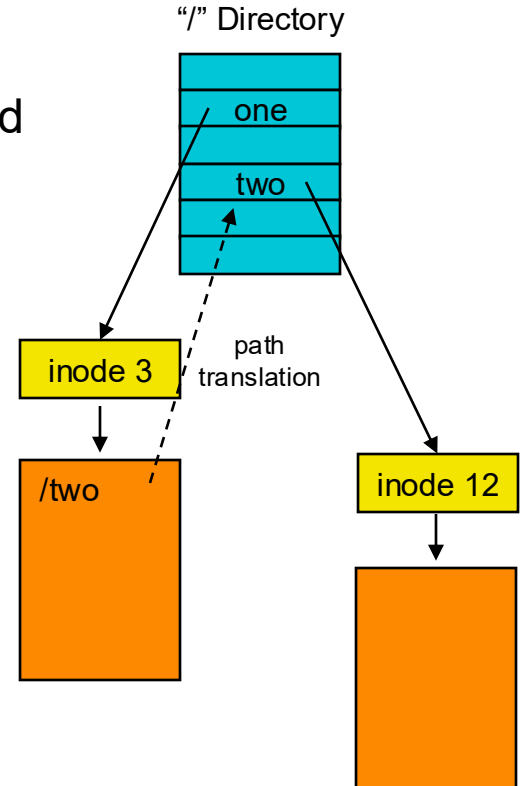
- A: 4
- B: 5
- C: 7
- D: 8
- E: 10
- F: 12

To read “/one”:

- superblock
- inode for “/”
- data block for “/”
- inode for “one”
- data block for “one”

To read “/two”:

- superblock
- inode for “/”
- data block for “/”
- inode for “two”
- data block for “two”

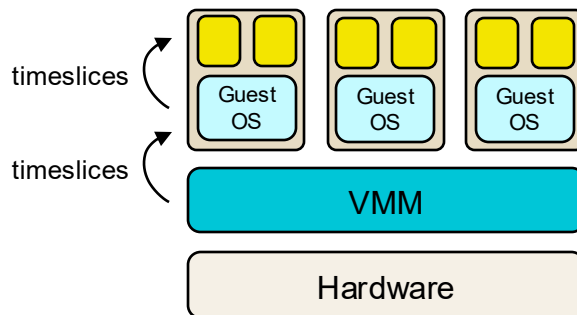


What Needs to be Virtualized?

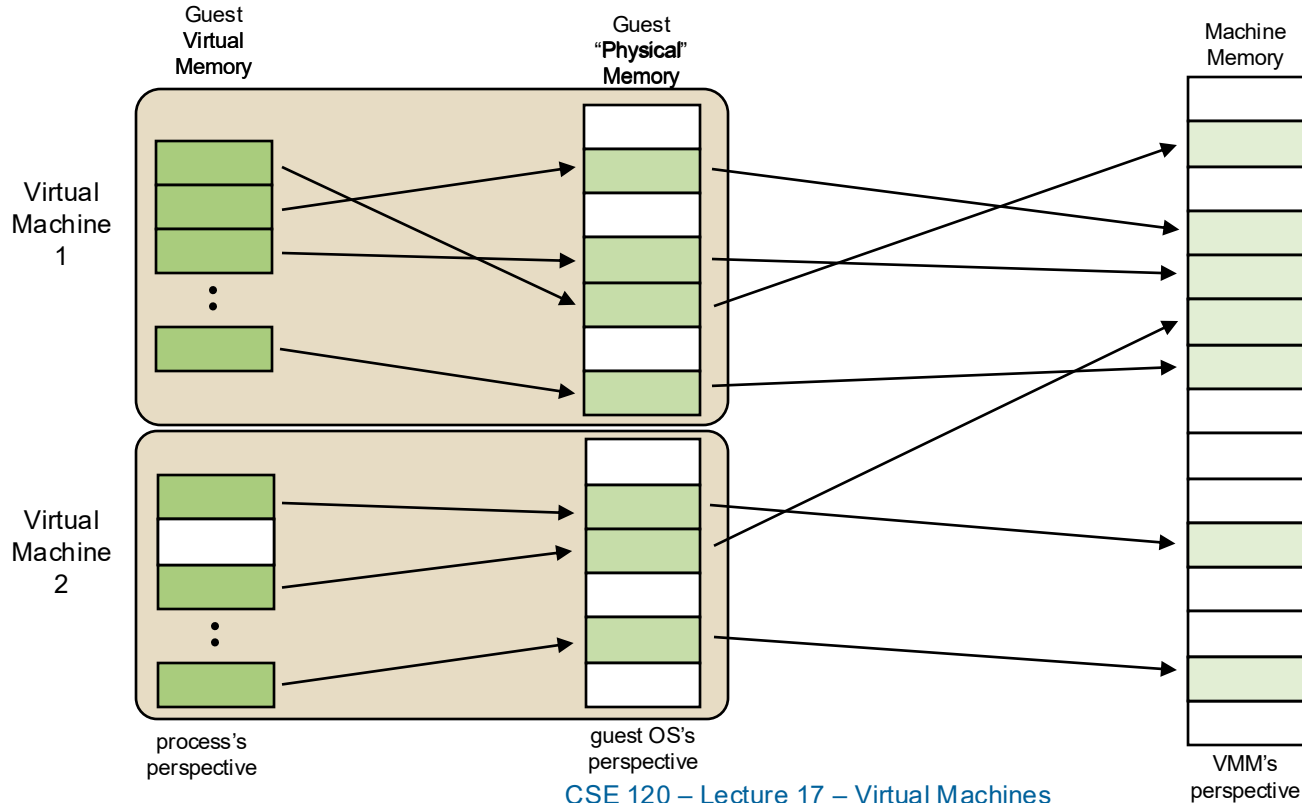
- Privileged instructions
- Events (exceptions and interrupts)
- CPU
- I/O devices
- Memory

Virtualizing the CPU

- VMM needs to schedule multiple VMs on the CPU
- Reuse scheduling techniques
 - Timeslice the VMs
 - Typically a simple scheduler (e.g., round robin)
 - Each VM will timeslice its guest OS/applications during its quantum



Virtualizing Memory



Final Exam

- Monday, June 9th, 3:00-6:00 pm
- Location: REC Gym (not here!)
- Content
 - The final exam is cumulative (covers the entire quarter)
 - It will focus on the content covered since the midterm
- Format
 - Similar to the midterm: multiple choice, short answer, multi-part problems
- Preparation
 - We will review in class on Thursday 6/5
 - Sample exam on the course website
 - In week 10, my Thursday office hours will be on Friday (6/6), 1-2 pm instead

Upcoming Tasks

- Discussion tomorrow: project 3, continued
- Read chapters 53 and 55
- Homework 4
 - Due Tuesday 6/3 at 11:59 pm
- Project 3
 - Due Friday 6/6 at 11:59 pm, no option to submit late

