# Virtual Machines

COMP 3361: Operating Systems I

Winter 2015

http://www.cs.du.edu/3361

# 1 Virtualization

- Create illusion of multiple machines on the same physical hardware
- Single computer hosts multiple virtual machines (computers)
- Virtualization software is also called **hypervisors**

- Each virtual machine is managed by a **Virtual Machine Monitor (VMM)**
- **Host:** the OS that runs the hypervisor, or the hypervisor itself
- **Guest:** the OS that runs in the virtual machine

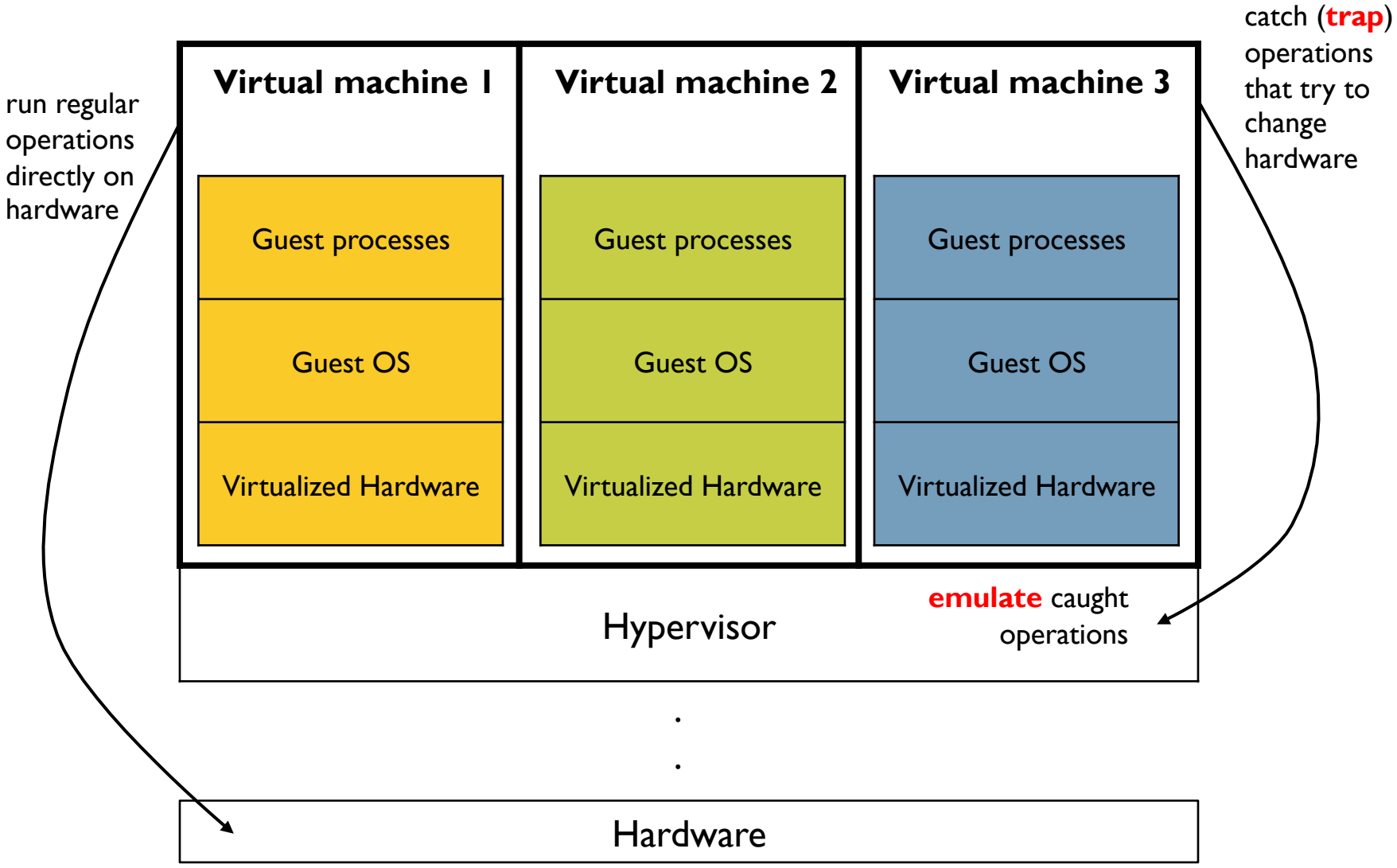
# 2 Why Virtualization?

- Isolation: failure of one guest does not bring down the entire system
- Cost: fewer physical machines mean less money to maintain hardware, or pay for running costs (e.g. electricity)
- Migration: moving a system is equivalent to moving the memory and disk images (files) in software
- Legacy support: run legacy applications without the need for legacy hardware
- Software development: test software written for different operating systems in a single machine
- Class assignments!

# 3 Requirements for Virtualization

- **Safety**: hypervisor should have full control of virtualized resources
- **Fidelity**: behavior of a program on a virtual machine should be identical to same program running on bare hardware
- **Efficiency**: much of the code in a virtual machine should run without intervention by the hypervisor

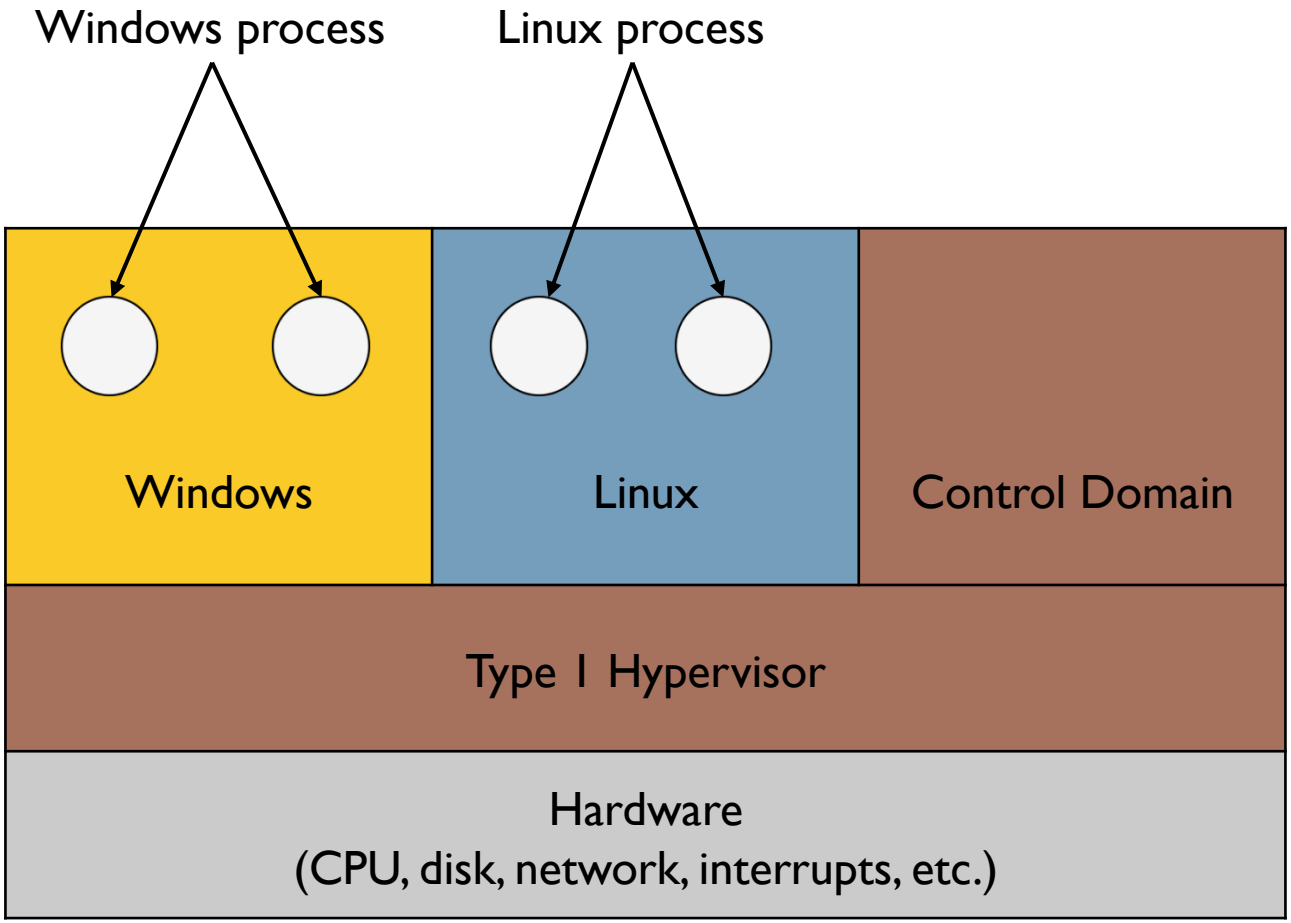
# 4 Trap-and-Emulate



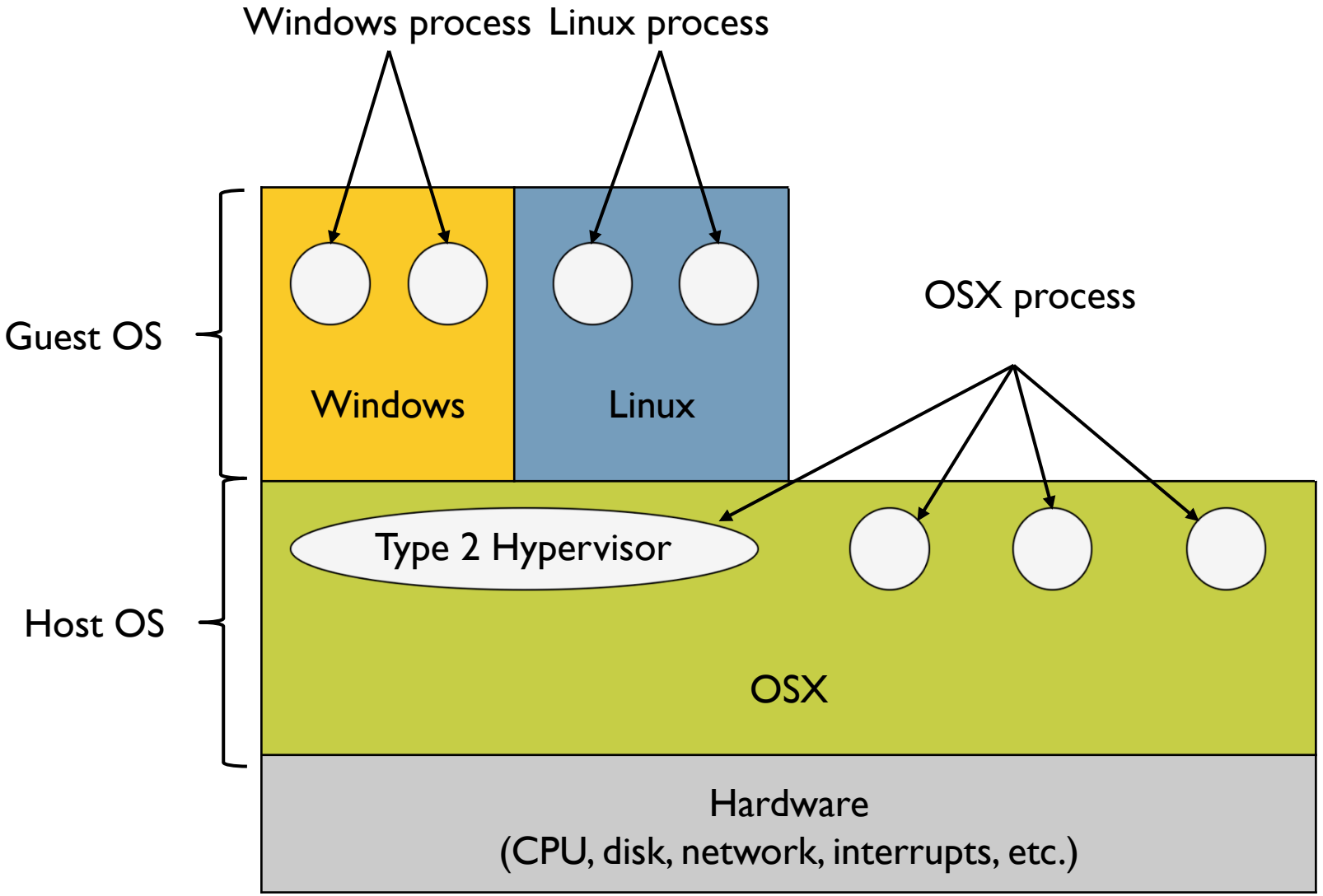
# Instruction Types

- **Sensitive:** instruction works differently in kernel mode and in user mode
  - e.g. POPF, SGDT, SIDT, …
- **Privileged:** instruction that is only allowed in kernel mode
  - e.g. CLI, MOV to CR3, …
  - executing a privileged instruction in user mode will result in an exception

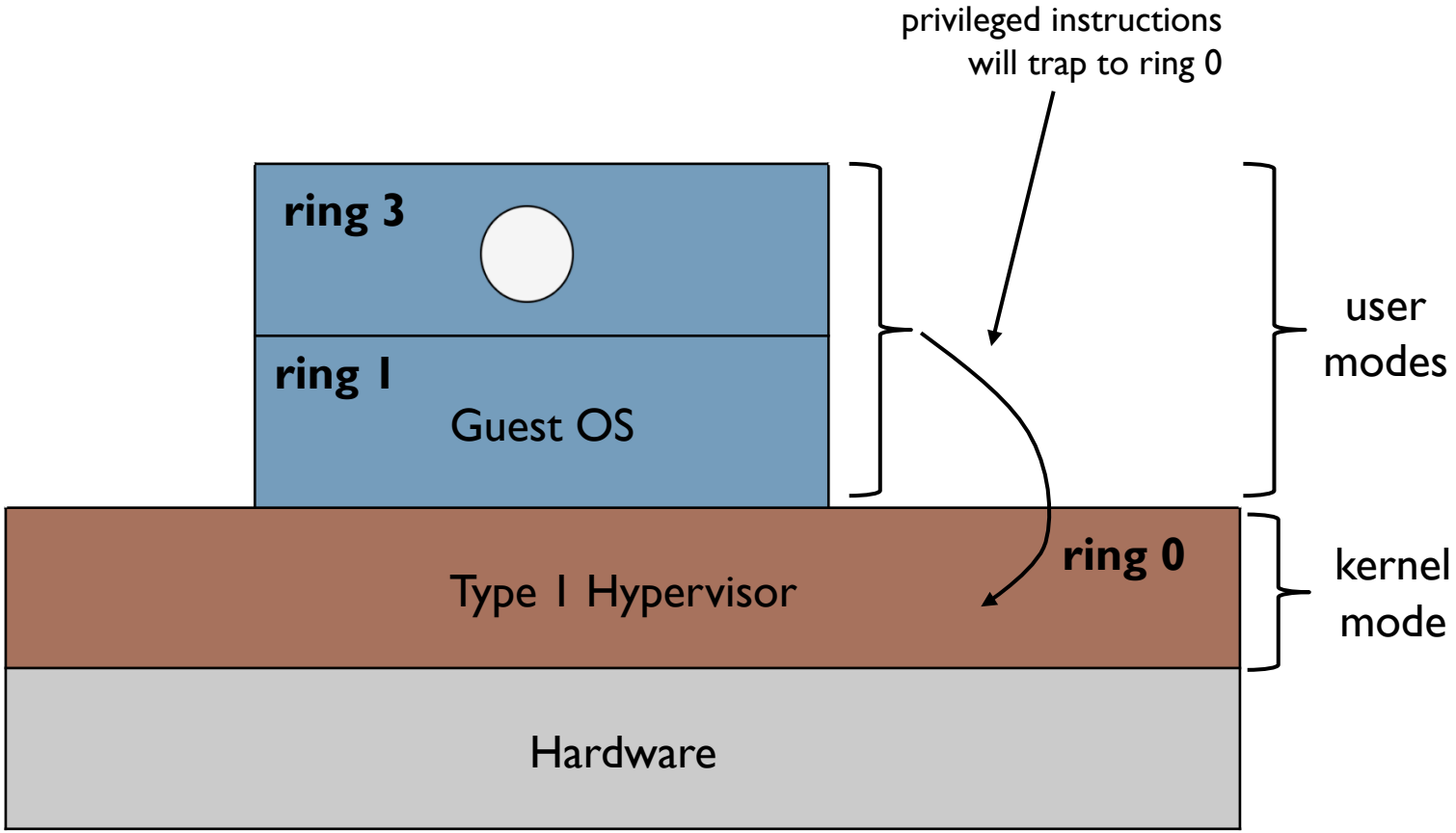
# Type 1 Hypervisors



# Type 2 Hypervisors



# 8 Virtualizing Sensitive Instructions




- Privileged instruction in user modes
  - from ring 1: emulate the instruction
  - from ring 3: transfer to exception handler in guest OS

**What happens for sensitive operations performed in guest OS?**

# 9 Full Virtualization With Binary Translation

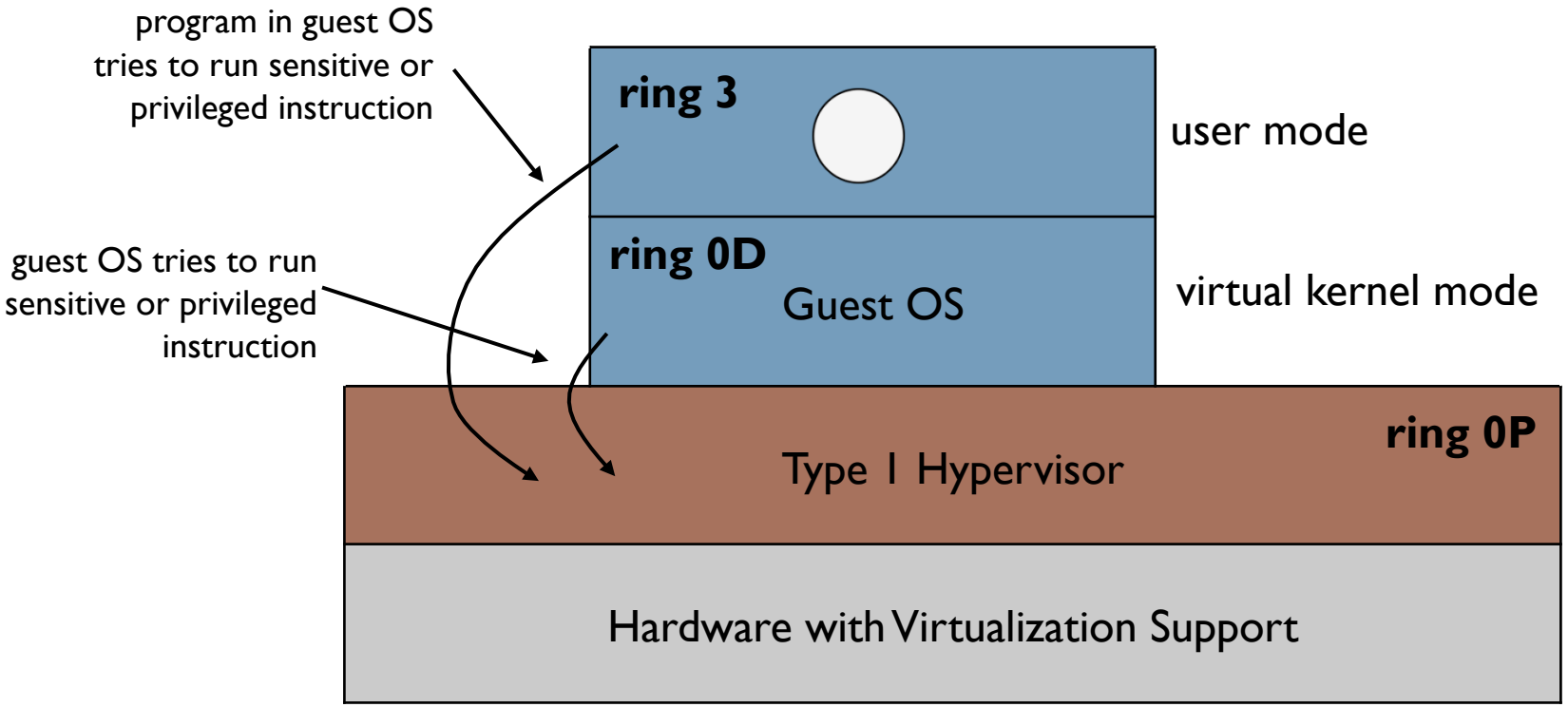
- Hardware will run sensitive instructions in guest OS processes as they should be since they are issued from ring 3
- Sensitive instructions executed in guest OS should be run as if they are issued in kernel mode
  - ring 1 is not kernel mode: hardware will not do this by default
- **Binary translation**: sensitive instructions in kernel code are converted to calls into hypervisor code
  - hypervisor code emulates the instruction as if it is executed in kernel mode
- Converted code fragments are cached

# Para-Virtualization

- Guest OS is modified so that it makes a system call to the hypervisor when privileged operations are to be performed
  - these calls are also called **hypercalls**
- Poor portability, but easier than full virtualization

# 11 Hardware Assisted Virtualization

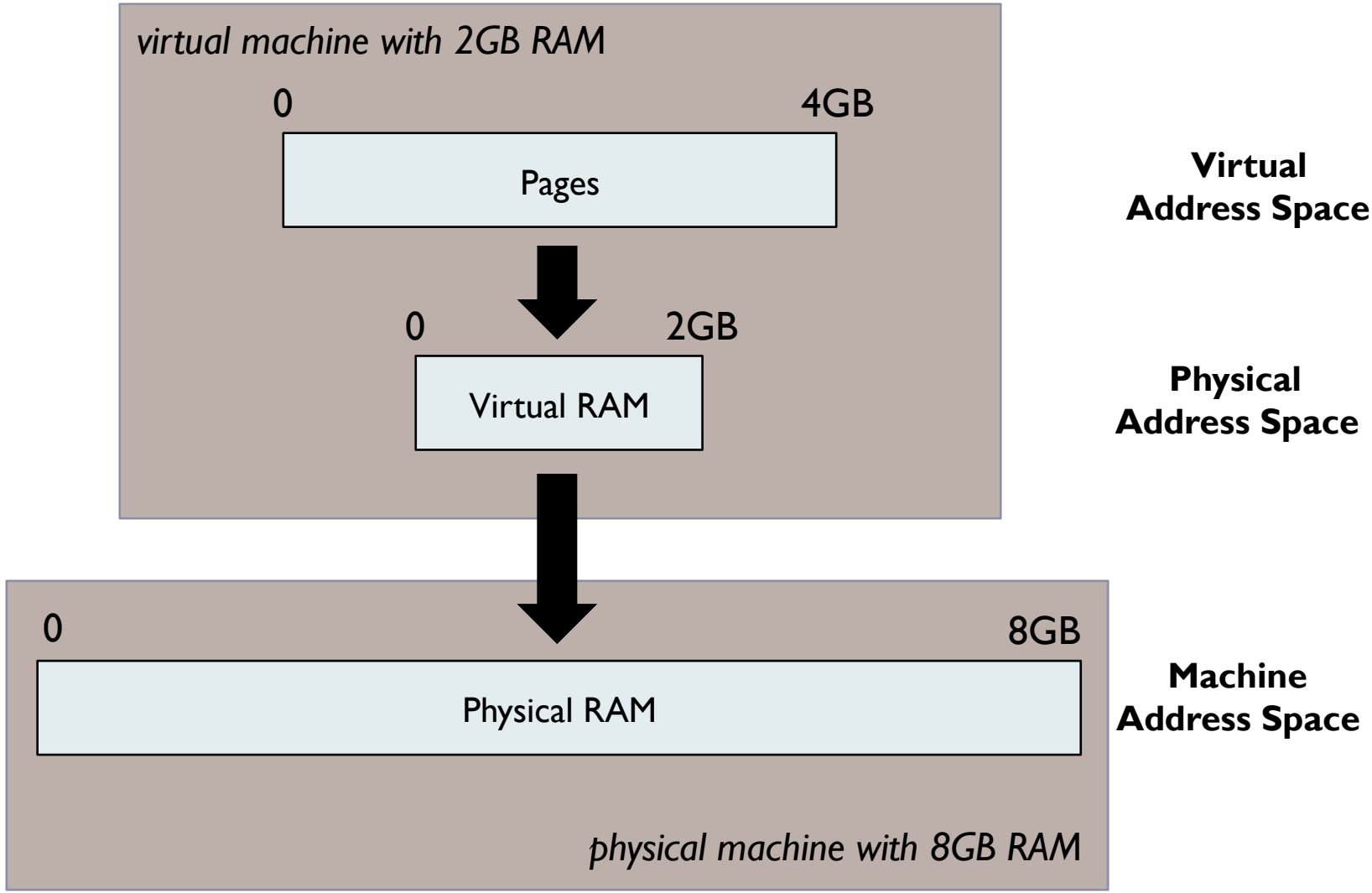
Technology: *Intel VT-X, AMD-V*



# Type 2 Hypervisors

- Type 2 hypervisors run as user processes
- Must perform binary translation of all sensitive and privileged code that run in the hypervisor
- Or, install a module in ring 0 that takes care of loading virtual machines
  - set up to run similar to full virtualization in type 1 hypervisors
  - ring 0 module must take care to "clean" the CPU state when other host OS processes are running

# Memory in Virtual Machines



# Memory Virtualization

- Efficiency: the lesser we involve the hypervisor, the faster the code runs
- Consider that the guest page tables map page *x* to frame *y*, and we use these page tables directly in hardware for efficiency
- Could be problematic:
  - frame *y* could be in use by the hypervisor or the host
  - frame *y* could be in use by another guest

# Shadow Page Tables

- Hypervisor maintains another set of page tables for each virtual machine
  - **shadow page tables**: the actual mapping between pages and frames
- When guest OS tries to load page tables, it traps to the hypervisor
  - loading page tables is a privileged operation
- Hypervisor makes MMU use the shadow page tables instead of the guest page tables

# Issue with Shadow Page Tables

- Guest OS removes an existing mapping
  - guest OS needs to invalidate the entry in the TLB
  - invalidating TLB is a privileged operation
    - hypervisor will know and update the shadow page table as well
- Guest OS remaps a page to a different frame
  - TLB invalidation should be performed here too
- Guest OS maps a new page to a frame
  - a simple update in memory that belongs to the guest OS
  - no sensitive/privileged operation is necessary to do this

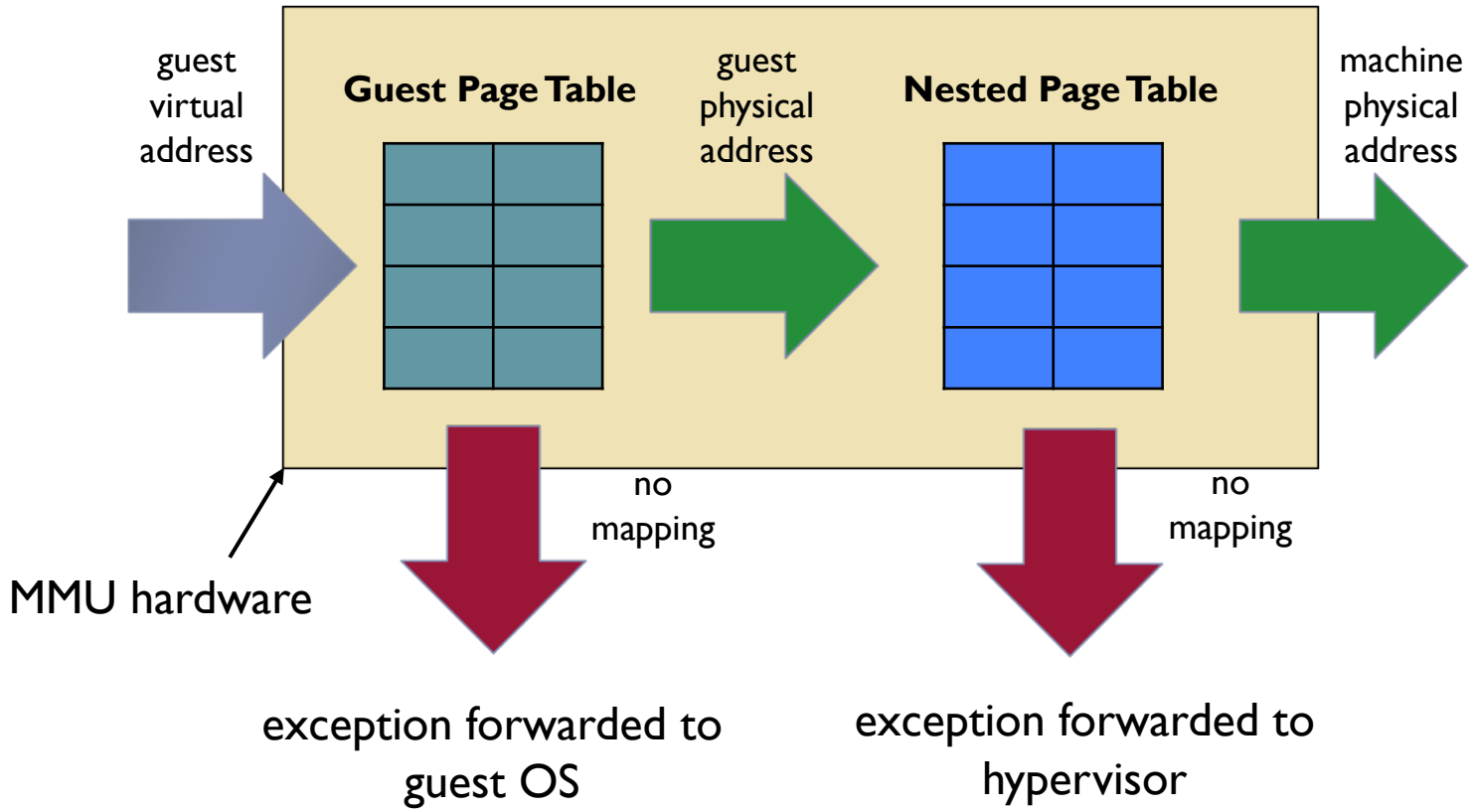
# Shadow Page Table Syncing

- **Solution 1**: For the guest OS, hypervisor marks the pages containing the page directory/tables as read only in the shadow page table
  - hardware will generate an exception when attempting to write to the page directory/tables
  - exception will be handled by the hypervisor
- **Solution 2**: Delay syncing till problem occurs
  - guest OS creates new entry in its page tables
  - guest OS tries to access newly added page
  - no mapping will be found by the hardware for the page
    - since it is using the out-of-sync shadow page tables
  - an exception will be generated
  - hypervisor handles exception by syncing the page tables

# 18 Hardware Based Memory Virtualization

- Shadow page tables work by transferring control to the hypervisor via exceptions
  - too expensive!
- Hardware support for memory virtualization
  - AMD Nested Page Tables
  - Intel Extended Page Tables

# Nested Page Tables



# I/O Virtualization

- I/O operations are sensitive
  - hypervisor gets control when they are performed by guest OS and handles it
- Disk I/O: can read/write to a file instead of an actual disk
- Interrupts: hypervisor receives actual interrupts and propogates them to guest OS by calling the guest handler
- DMA transfers: hypervisor loads IOMMU with mappings such that data is transferred to frames dedicated to guest OS
- Hardware support: separate buffers, registers, queues, etc. for virtual machines

# References

- Chapter 7.1-7.7, Modern Operating Systems, A. Tanenbaum and H. Bos, 4th Edition.