# Recap

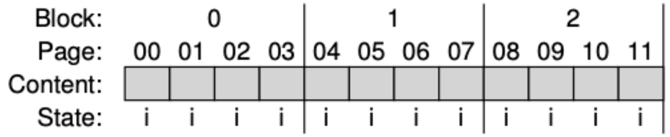
SSD

- Storage hierarchical organization: bank-block-page-bit
- Operations: read(page), erase(block), program(page)
- Erase before write
- Wear out issue; wear leveling
- Log-structured FS

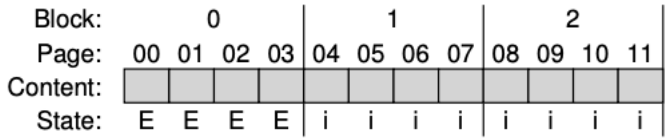
# Log-Structured Example

Write(100) with contents a1
Write(101) with contents a2
Write(2000) with contents b1
Write(2001) with contents b2

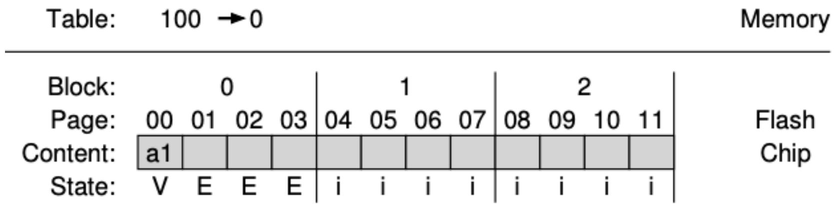
Block starts with all invalid



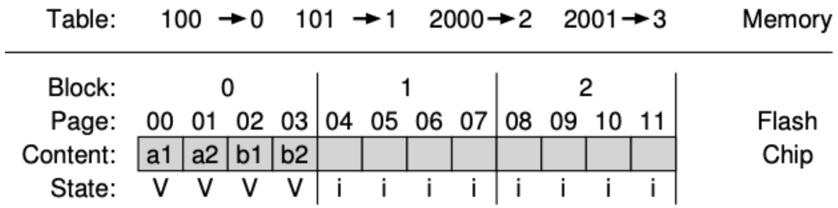

Erase block




Write a1 to first free physical
Page and log mapping

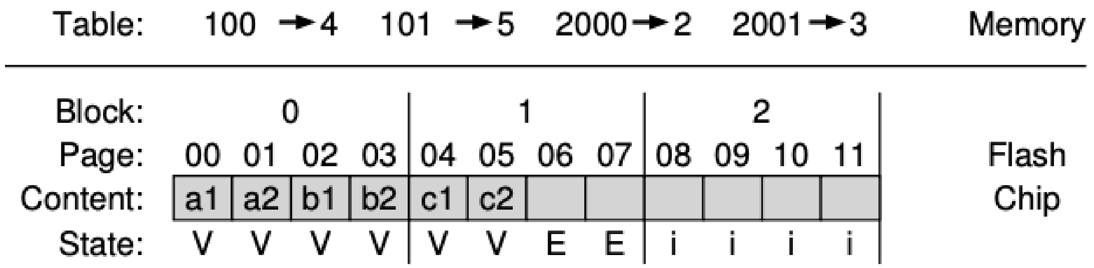



Write other pages to free
Physical pages and log
mappings



# Garbage Collection

If the same logical page is written multiple times, the old versions of the page will remain in physical memory as garbage (unusable)




**Garbage collection** is the reclamation of dead blocks

In example above, b1 and b2 can be moved to physical pages 6 and 7, then block 0 can be erased for reuse

# Mapping Table Size

Mapping table can be very large

Assume 1TB SSD and 4 byte entry for each 4KB page, then map is 1GB

A **hybrid mapping** approach can map at either the page or block level, far fewer blocks so less mapping required

# Example Performance of HDD vs SSD

| Device | Random Reads (MB/s) | Random Writes (MB/s) | Sequential Reads (MB/s) | Sequential Writes (MB/s) |
|---|---|---|---|---|
| Samsung 840 Pro SSD | 103 | 287 | 421 | 384 |
| Seagate 600 SSD | 84 | 252 | 424 | 374 |
| Intel SSD 335 SSD | 39 | 222 | 344 | 354 |
| Seagate Savvio 15K.3 HDD | 2 | 2 | 223 | 223 |

- For random I/O, SSD significantly outperforms HDD
- For sequential I/O, SSD still outperforms HDD
- SSD random write outperforms SSD random read; why?
- Sequential access always outperforms random access

# Virtual Machine

(Based on Appendix B)

Slides revised from
Matthew Tancreti
Iowa State University

# Motivations for Virtual Machine

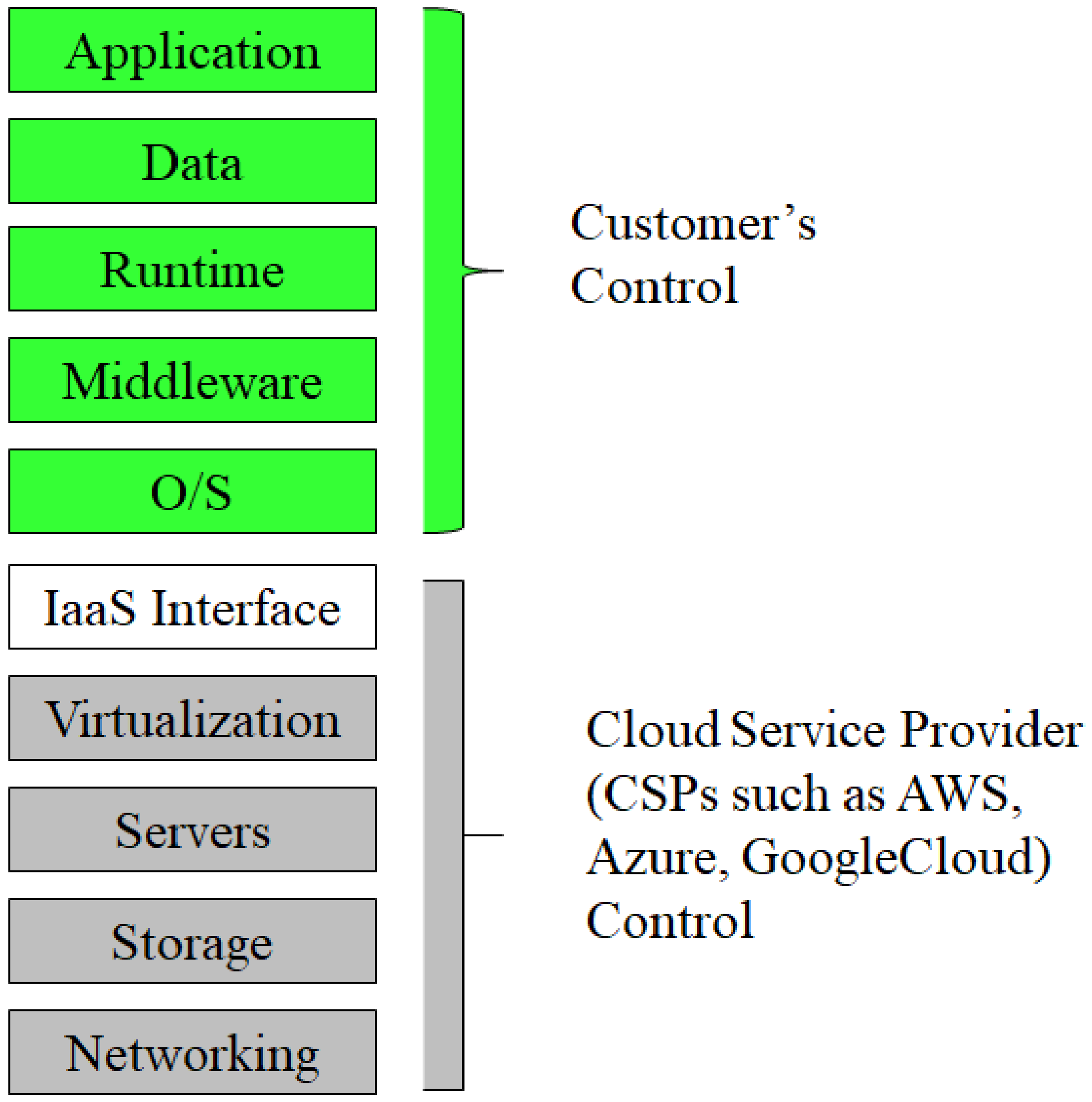
Consolidate multiple OSes onto fewer hardware platforms, lowering cost and ease administration

Accesses to applications that only run on a different platform, e.g., run Linux application on Windows host

Isolated and reproducible environment for testing and debugging

More flexible and secure sharing of resources (e.g., cloud computing)

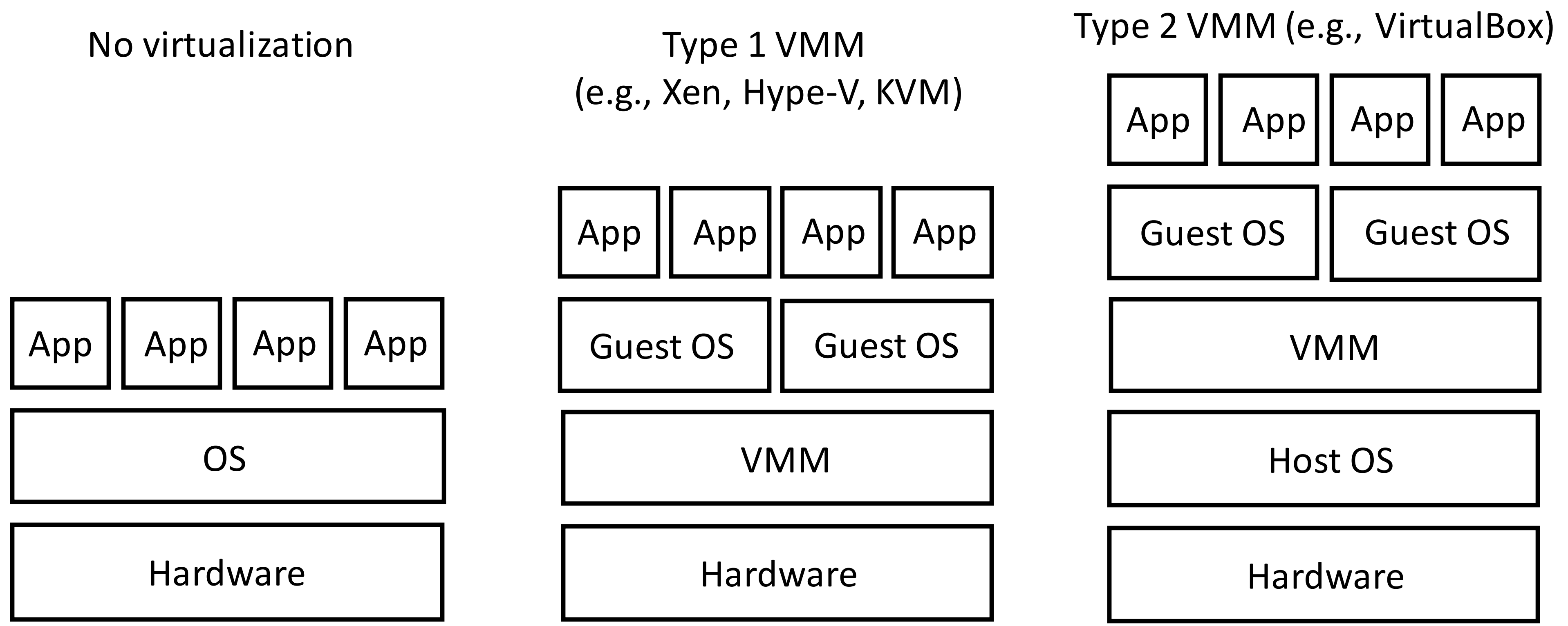
# Cloud Computing



# Virtual Machine Monitor

**Virtual Machine Monitor** (**VMM**) (also called **hypervisor**) creates illusion of being hardware to a OS above it

Don’t want to modify the OS to run on VMM, **transparency** is major goal of VMM



# Limited Direct Execution

OS achieves virtualization through limited direct execution, e.g., context switching between processes

VMM also operates on limited direct execution, need to context switch (or “machine switch”) between virtual machines

To achieve “machine switch”, VMM must save entire machine state of one OS and restore state for state being switched into

# Problem: OS Privilege

A problem is that OS normally operates in kernel mode allowing it to execute all privileged instructions

# Standard System Call

| Process | Hardware | Operating System |
|---|---|---|
| **1.** Execute instructions (add, load, etc.) | | |
| **2.** System call: Trap to OS | | |
| | **3.** Switch to kernel mode; Jump to trap handler | |
| | | **4.** In kernel mode; Handle system call; Return from trap |
| | **5.** Switch to user mode; Return to user code | |
| **6.** Resume execution (@PC after trap) | | |

# Trap Handler

Standard system call:

application in user mode executes privileged instruction (e.g., int 80)
which causes trap
which switches processor to kernel mode

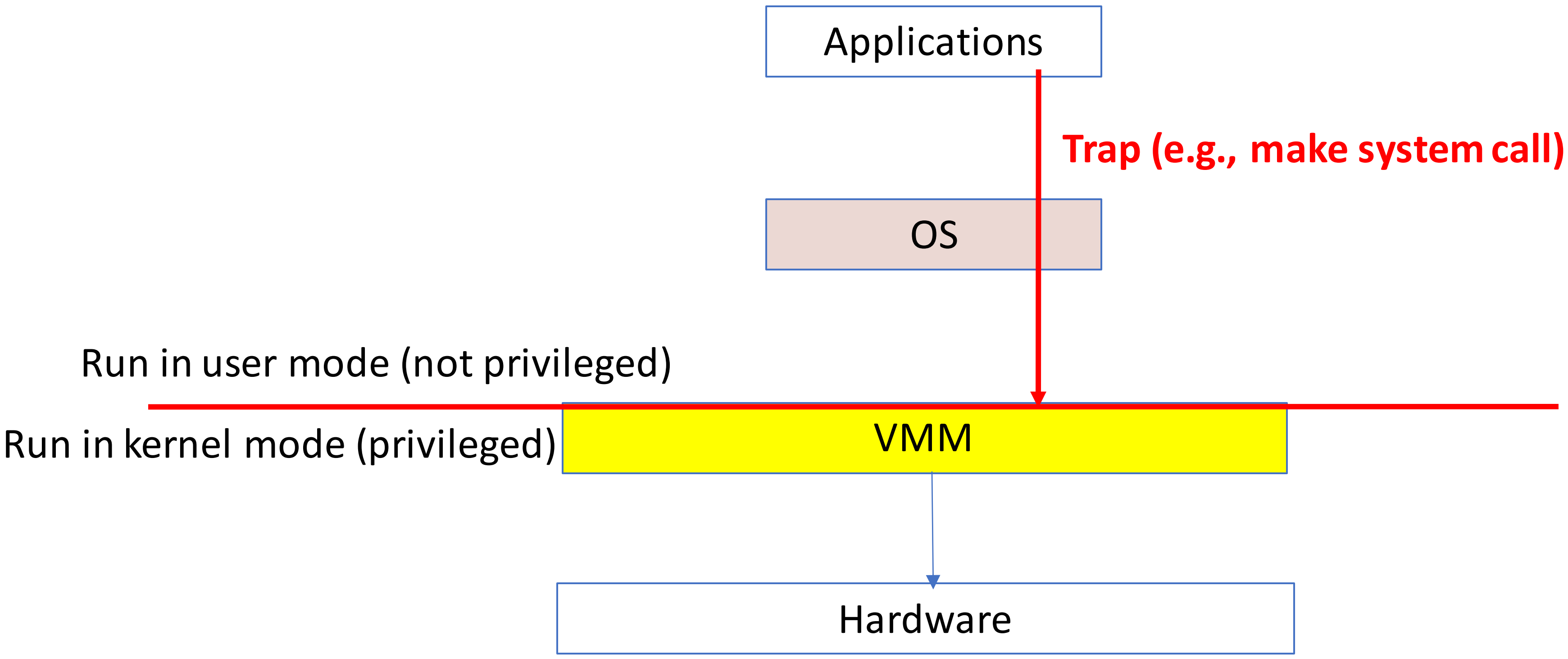
In standard configuration, it is safe for trap to execute in kernel mode because the **OS installed the trap handlers at boot time**

- There is only one OS who controls the whole system

The difference on virtual machine is the **VMM installed the trap handlers**

- There could be many OSes, none of which should control the whole system
- Only the VMM should control the whole system

# Trap Handling with Virtual Machine



# How does VMM know what to do with trap?

When the guest OS boots it tries to install trap handlers by writing them into specified memory locations

VMM know where in memory the guest OS trap handers are located

When VMM trap handler executes it calls the guest OS trap handler

When guest OS trap handler returns from trap, it returns into the VMM trap handler which performs the actual return-from-trap into the application

# Reduced Privilege

But what stops the guest OS trap handler from executing instructions that overwrite the VMM instructions?

Before calling guest OS trap handler, VMM reduces the privilege mode (some processors have an additional **supervisor mode**)

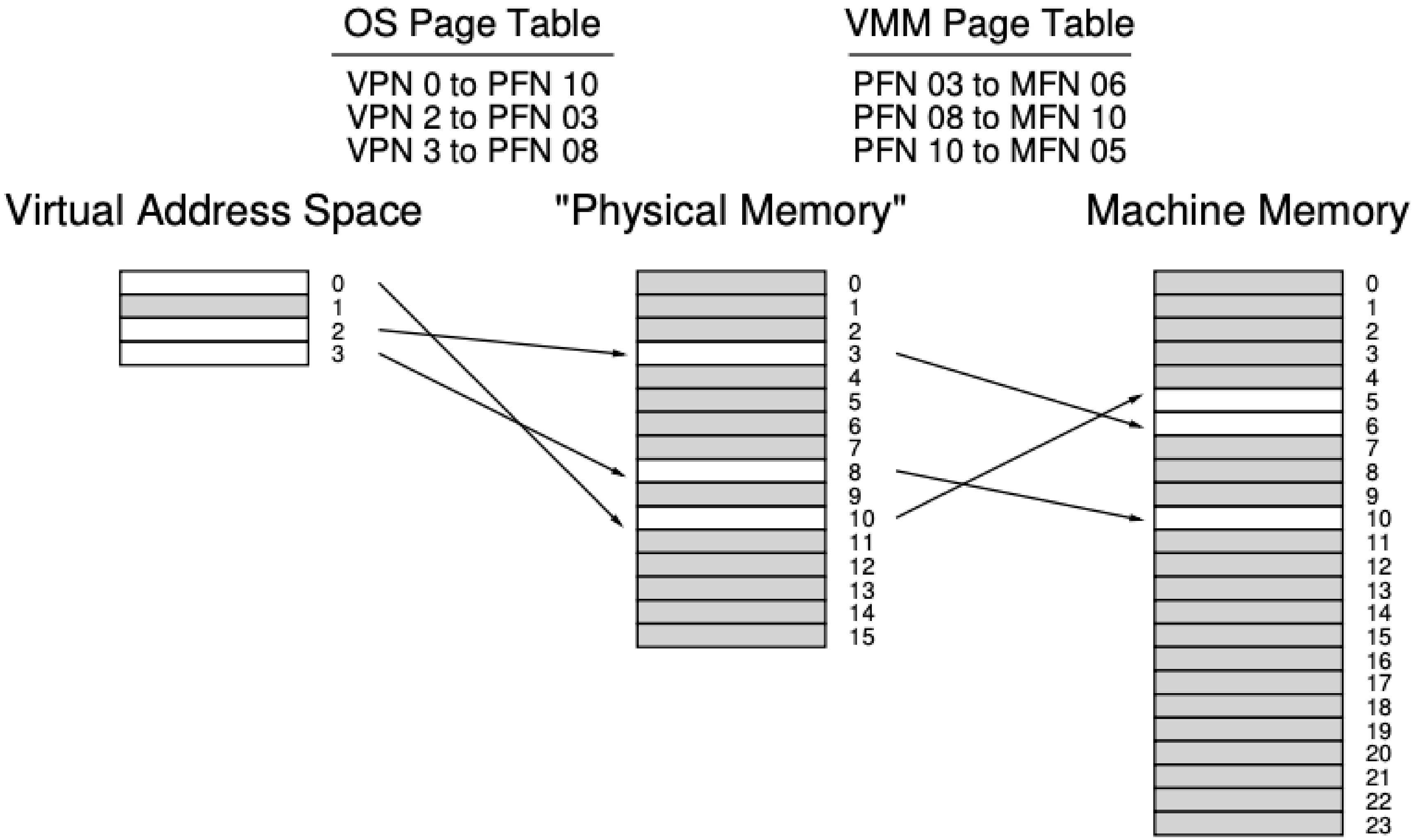
When OS executes privileged instruction, a trap occurs giving VMM control

Guest OS executing return-from-trap is also a privileged instruction, which is how control returns to VMM trap handler

# System Call with VMM

| 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) | | |

# Memory Virtualization Virtualization



# Standard Address Translation Flow on TLB Miss

Assume software managed page tables, i.e., in memory and managed by the OS
VA = Virtual Address

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

# TLB Miss with VMM

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

# Information Gap

Because of transparency, VMM doesn't know what guest OS is trying to achieve

There is an **information gap** between OS and VMM that can lead to significant inefficiency

For example, when OS has no useful work (i.e., no runnable processes) it will spin in its scheduler loop

```
while (1)
  ; // the idle loop
```

Potential solution breaks transparency, in **para-virtualization** guest OS has small modifications to operate more effectively in virtualized environment

# Container

Also known as OS-level virtualization.

Examples: Docker container, Solaris Zone, OpenVZ virtual private server, FreeBSD jail

Container vs Native Computer

- A computer program running on an ordinary OS can see all resources.
- However, programs running inside of a container can only see the container's contents and devices assigned to the container.

# Virtual Machine vs Container

Similar goals:

- To isolate an application and its dependencies into a self-contained unit that can run anywhere;
- For more efficient and secure use of (shared) resources.

Differences:

- VM – package up the virtual hardware, a kernel (i.e., OS) and user space for each VM.
- Container – package up just the user space, and not the kernel or virtual hardware like a VM does; more efficient but less isolated/secure.

# Virtual Machine vs Container