CSCI315 – Operating Systems Design

Department of Computer Science
Bucknell University

Introduction to Virtual Machines

Ch 18.1-18.3

This set of notes is based on notes from the textbook authors, as well as L. Felipe Perrone, Joshua Stough, and other instructors.
Xiannong Meng, Fall 2021.

Chapter 18: Virtual Machines

- Overview
- History
- Benefits and Features
- Building Blocks
- Types of Virtual Machines and Their Implementations
- Virtualization and Operating-System Components
- Examples

Chapter Objectives

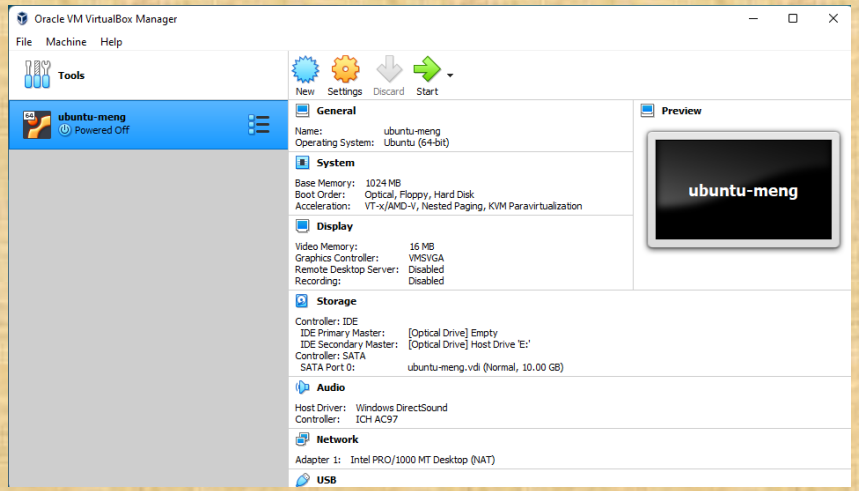
- Explore the history and benefits of virtual machines
- Discuss the various virtual machine technologies
- Describe the methods used to implement virtualization
- Show the most common hardware features that support virtualization and explain how they are used by operating-system modules
- Discuss current virtualization research areas

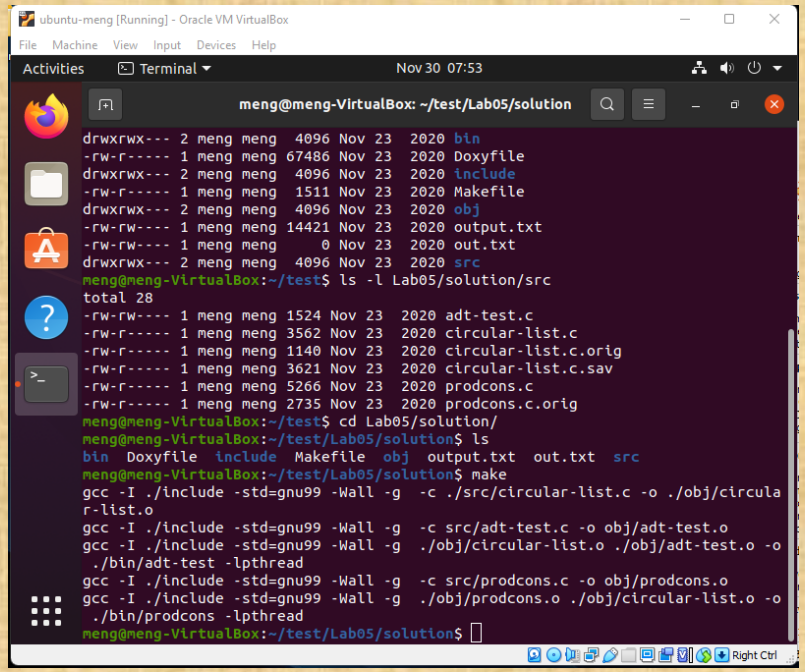
Overview

- Fundamental idea – abstract hardware of a single computer into several different execution environments
  - Similar to layered approach
  - But layer creates virtual system (**virtual machine**, or **VM**) on which operation systems or applications can run
- Several components
  - **Host** – underlying hardware system
  - **Virtual machine manager** (**VMM**) or **hypervisor** – creates and runs virtual machines by providing interface that is ***identical*** to the host
    - (Except in the case of paravirtualization)
  - **Guest** – process provided with virtual copy of the host
    - Usually an operating system
- Single physical machine can run multiple operating systems concurrently, each in its own virtual machine

Quick Examples of VM

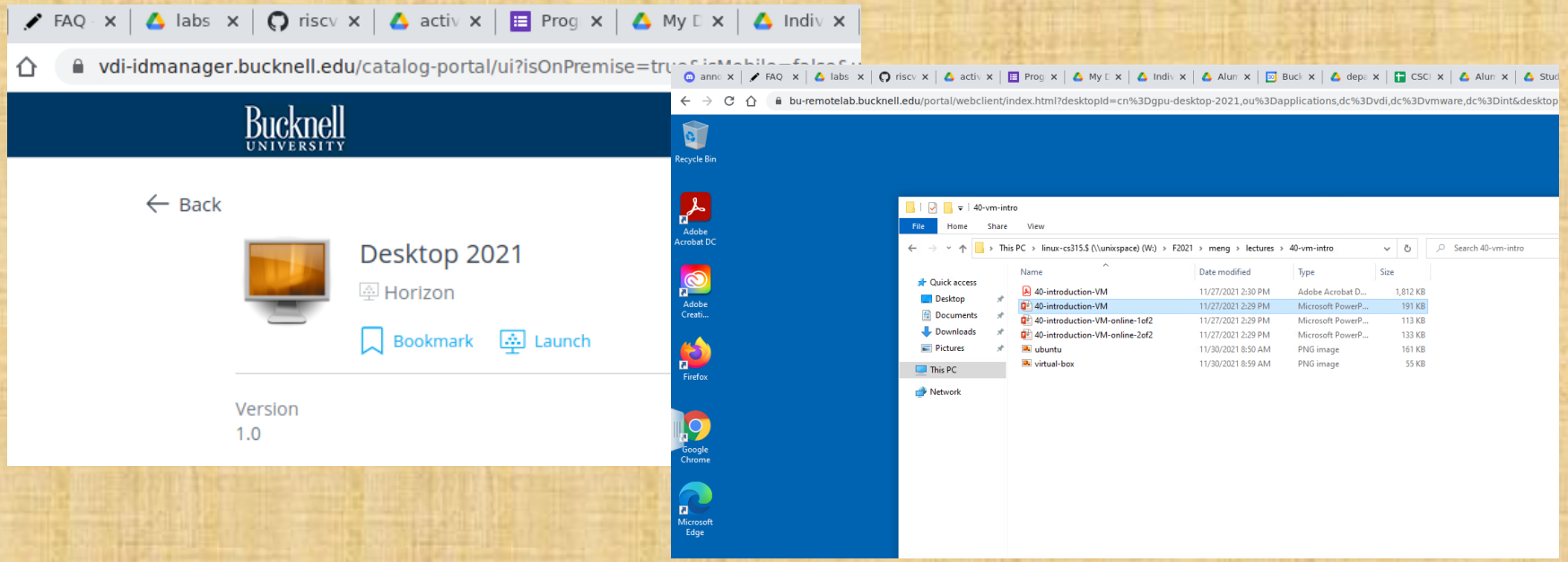
- Oracle VirtualBox: Running Linux on Windows



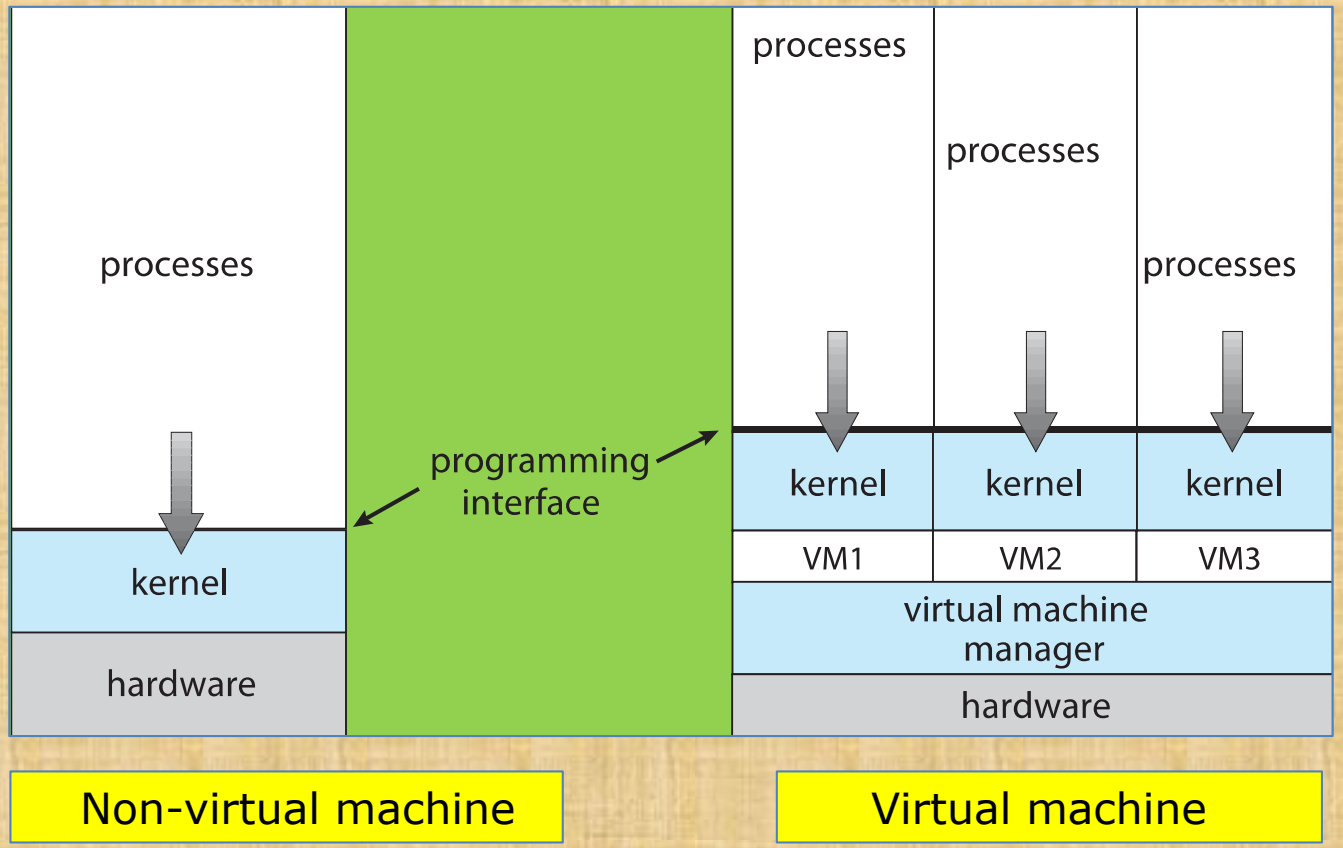


Quick Examples of VM

- Remote Lab: Running Windows over the web



System Models



Implementation of VMMs

- Vary greatly, with options including:
  - **Type 0 hypervisors** - Hardware-based solutions that provide support for virtual machine creation and management via firmware
    - IBM LPARs and Oracle LDOMs are examples
  - **Type 1 hypervisors** - Operating-system-like software built to provide virtualization
    - Including VMware ESX, Joyent SmartOS, and Citrix XenServer
  - **Type 1 hypervisors** – Also includes general-purpose operating systems that provide standard functions as well as VMM functions
    - Including Microsoft Windows Server with HyperV and RedHat Linux with KVM
  - **Type 2 hypervisors** - Applications that run on standard operating systems but provide VMM features to guest operating systems
    - **Including** VMware Workstation and Fusion, Parallels Desktop, and Oracle VirtualBox

Implementation of VMMs (Cont.)

- Other variations include:
  - **Paravirtualization** - Technique in which the guest operating system is modified to work in cooperation with the VMM to optimize performance
  - **Programming-environment virtualization** - VMMs do not virtualize real hardware but instead create an optimized virtual system
    - Used by Oracle Java and Microsoft.Net
  - **Emulators** – Allow applications written for one hardware environment to run on a very different hardware environment, such as a different type of CPU

Implementation of VMMs (Cont.)

- **Application containment** - Not virtualization at all but rather provides virtualization-like features by segregating applications from the operating system, making them more secure, manageable
  - Including Oracle Solaris Zones, BSD Jails, and IBM AIX WPARs
- Much variation due to breadth, depth and importance of virtualization in modern computing

History

- First appeared in IBM mainframes in 1972
- Allowed multiple users to share a batch-oriented system
- Formal definition of virtualization helped move it beyond IBM
  1. A **VMM** provides an environment for programs that is essentially identical to the original machine
  2. Programs running within that environment show only minor performance decreases
  3. The **VMM** is in complete control of system resources
- In late 1990s Intel CPUs fast enough for researchers to try virtualizing on general purpose PCs
  - **Xen** and **VMware** created technologies, still used today
  - Virtualization has expanded to many OSes, CPUs, VMMs

Benefits and Features

- Host system protected from VMs, VMs protected from each other
  - i.e., A virus less likely to spread
  - Sharing is provided via shared file system volume, network communication
- Freeze, **suspend**, running VM
  - Then can move or copy somewhere else and **resume**
  - Snapshot of a given state, able to restore back to that state
    - Some VMMs allow multiple snapshots per VM
  - **Clone** by creating copy and running both original and copy
- Great for OS research, better system development efficiency
- Run multiple, different OSes on a single machine
  - **Consolidation**, app dev, …

Benefits and Features (Cont.)

- **Templating** – create an OS + application VM, provide it to customers, use it to create multiple instances of that combination
- **Live migration** – move a running VM from one host to another!
  - No interruption of user access
- All those features taken together -> **cloud computing**
  - Using APIs, programs tell cloud infrastructure (servers, networking, storage) to create new guests, VMs, virtual desktops

Building Blocks

- Generally difficult to provide an ***exact*** duplicate of underlying machine
  - Especially if only dual-mode operation available on CPU
  - But getting easier over time as CPU features and support for VMM improves
  - Most VMMs implement **virtual CPU** (**VCPU**) to represent state of CPU per guest as guest believes it to be
    - When guest context switched onto CPU by VMM, information from VCPU loaded and stored
  - Several techniques, as described in next slides

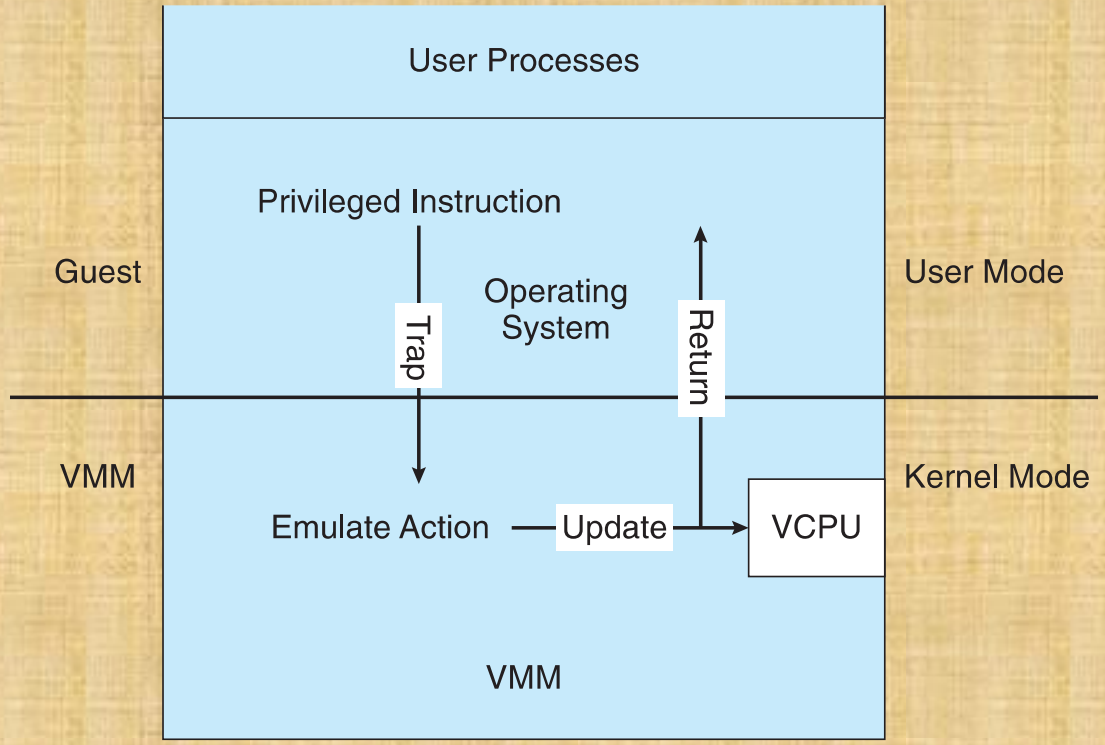
Building Block – Trap and Emulate

- Dual mode CPU means guest executes in user mode
  - Kernel runs in kernel mode
  - Not safe to let guest kernel run in kernel mode too
  - So VM needs two modes – virtual user mode and virtual kernel mode
    - Both of which run in real user mode
  - Actions in guest that usually cause switch to kernel mode must cause switch to virtual kernel mode

Trap-and-Emulate (Cont.)

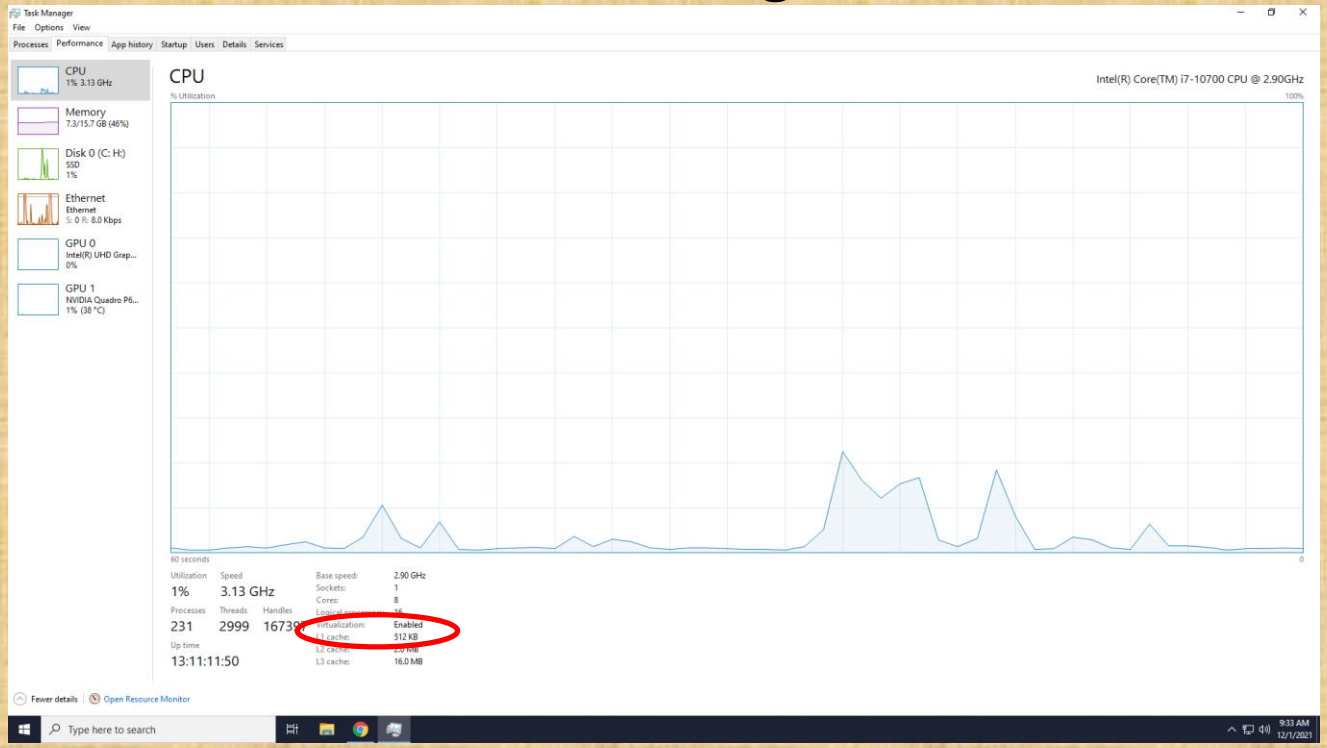
- How does switch from virtual user mode to virtual kernel mode occur?
  - Attempting a privileged instruction in user mode causes an error -> trap
  - VMM gains control, analyzes error, executes operation as attempted by guest
  - Returns control to guest in user mode
  - Known as **trap-and-emulate**
  - Most virtualization products use this at least in part
- User mode code in guest runs at same speed as if not a guest
- But kernel mode privilege mode code runs slower due to trap-and-emulate
  - Especially a problem when multiple guests running, each needing trap-and-emulate
- CPUs adding hardware support, CPU modes to improve virtualization performance

Trap-and-Emulate Virtualization Implementation



Is Virtualization Supported?

- On Windows, use TaskManager to find out



Is Virtualization Supported?

- On Linux, use /proc/cpuinfo to find out
- grep vmx /proc/cpuinfo
- If vmx is present, virtualization is supported

```
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2
tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf ea
pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc
ne_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch epb invpcid_single intel_pt ssbd ibrs ibpb stibp tpr_shadow vnm
riority ept vpid fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm mpx rdseed adx smap clflushopt xsaveopt xsavec
1 dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp spec_ctrl intel_stibp flush_l1d
```

Building Block – Binary Translation

- Some CPUs don't have clean separation between privileged and nonprivileged instructions
  - Earlier Intel x86 CPUs are among them
    - Earliest Intel CPU designed for a calculator
  - Backward compatibility means difficult to improve
  - Consider Intel x86 **`popf`** instruction
    - Loads CPU flags register from contents of the stack
    - If CPU in privileged mode -> all flags replaced
    - If CPU in user mode -> on some flags replaced
      - No trap is generated

Binary Translation (Cont.)

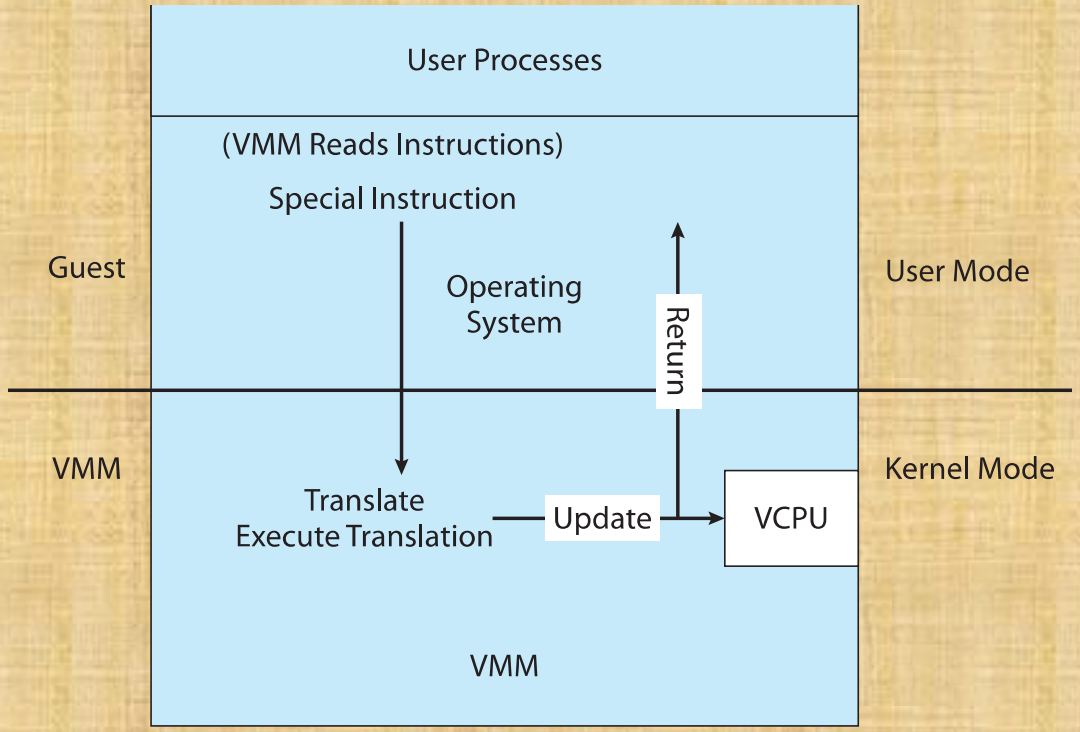
- Other similar problem instructions we will call ***special instructions***
  - Caused trap-and-emulate method considered impossible until 1998
- Binary translation solves the problem

1. Basics are simple, but implementation very complex
2. If guest VCPU is in user mode, guest can run instructions natively
3. If guest VCPU in kernel mode (guest believes it is in kernel mode)
   a) VMM examines every instruction guest is about to execute by reading a few instructions ahead of program counter
   b) Non-special-instructions run natively
   c) Special instructions translated into new set of instructions that perform equivalent task (for example changing the flags in the VCPU)

Binary Translation (Cont.)

- Implemented by translation of code within VMM
- Code reads native instructions dynamically from guest, on demand, generates native binary code that executes in place of original code
- Performance of this method would be poor without optimizations
  - Products like VMware use caching
    - Translate once, and when guest executes code containing special instruction cached translation used instead of translating again
    - Testing showed booting Windows XP as guest caused 950,000 translations, at 3 microseconds each, or 3 second (5 %) slowdown over native

Binary Translation Virtualization Implementation



Nested Page Tables

- Memory management is another general challenge to VMM implementations
- How can VMM keep page-table state for both guests believing they control the page tables and VMM that does control the tables?
- Common method (for trap-and-emulate and binary translation) is **nested page tables** (**NPTs**)
  - Each guest maintains page tables to translate virtual to physical addresses
  - VMM maintains per guest NPTs to represent guest's page-table state
    - Just as VCPU stores guest CPU state
  - When guest on CPU -> VMM makes that guest's NPTs the active system page tables
  - Guest tries to change page table -> VMM makes equivalent change to NPTs and its own page tables
  - Can cause many more TLB misses -> much slower performance