General Idea

- Multi-threading:
  - We have virtual CPUs that give the illusion that enables threads to persist their computation on the CPU as switching happens – without clobbering each other
- Multi-processing:
  - Adds memory virtualization to multi-threading. Each process has its own memory view.
- File system is shared and so is the kernel. However, kernel is a protected resource. In a perfect scenario, we don’t need to worry about this sharing.
- What is beyond..? Virtual Machines

It's 1964 …

- IBM wants a multiuser time-sharing system

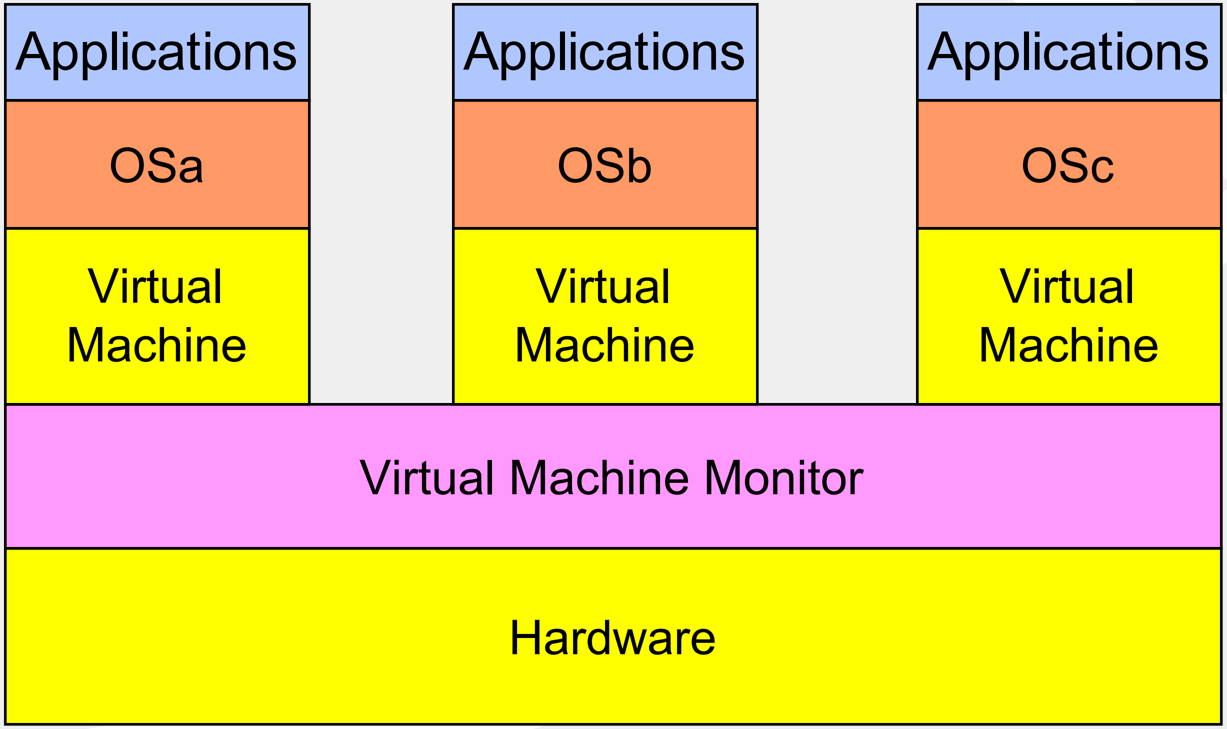
- TSS (Time sharing system) project
  - large, monolithic system
  - lots of people working on it
  - for years
  - total, complete flop

- CMS (Conversational monitor system)
  - single-user time-sharing system for IBM 360
- CP67 (Control program)
  - virtual machine monitor (VMM)
  - supports multiple virtual IBM 360s
- Put the two together …
  - a (working) multiuser time-sharing system

Into 1990s…

- 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

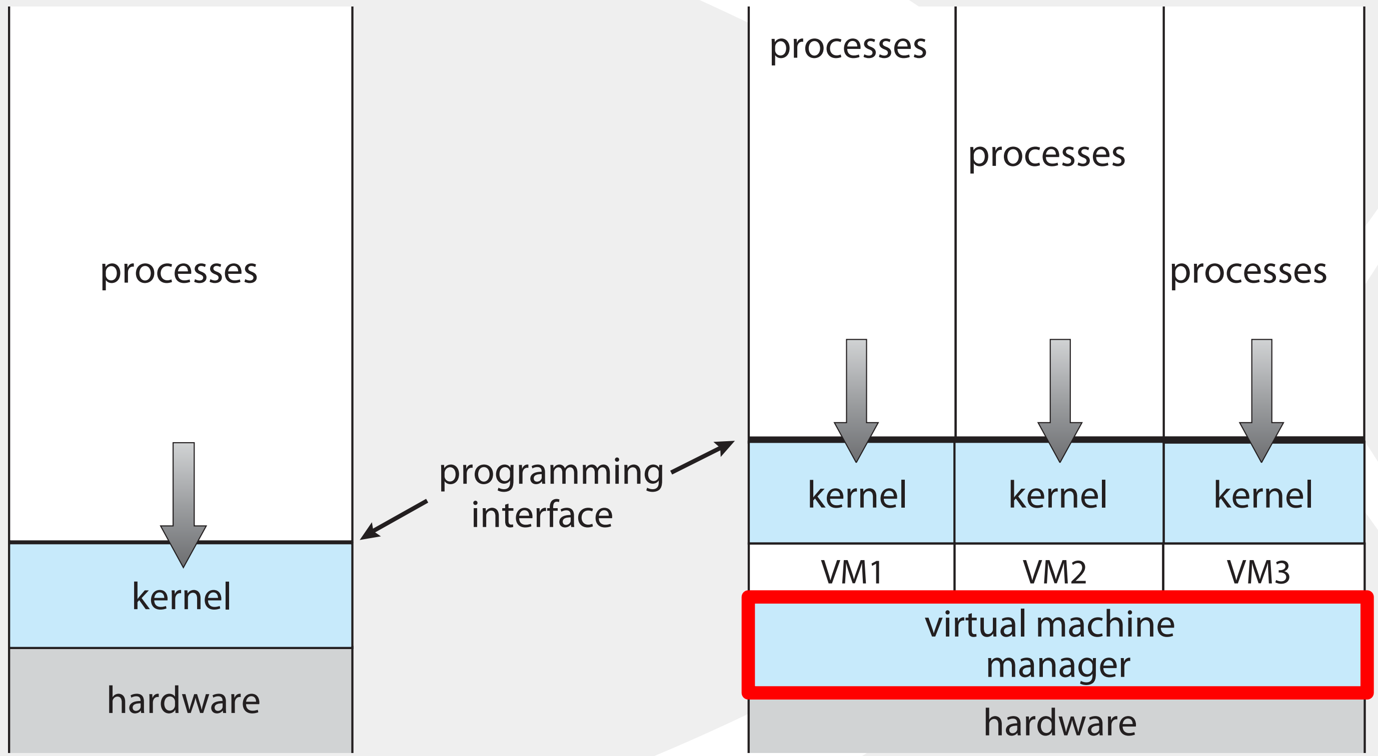
Virtual Machines



Virtual Machines

- 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 operating 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

Physical vs. Virtual Machines



Non-virtual machine

Virtual machine

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
  - **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

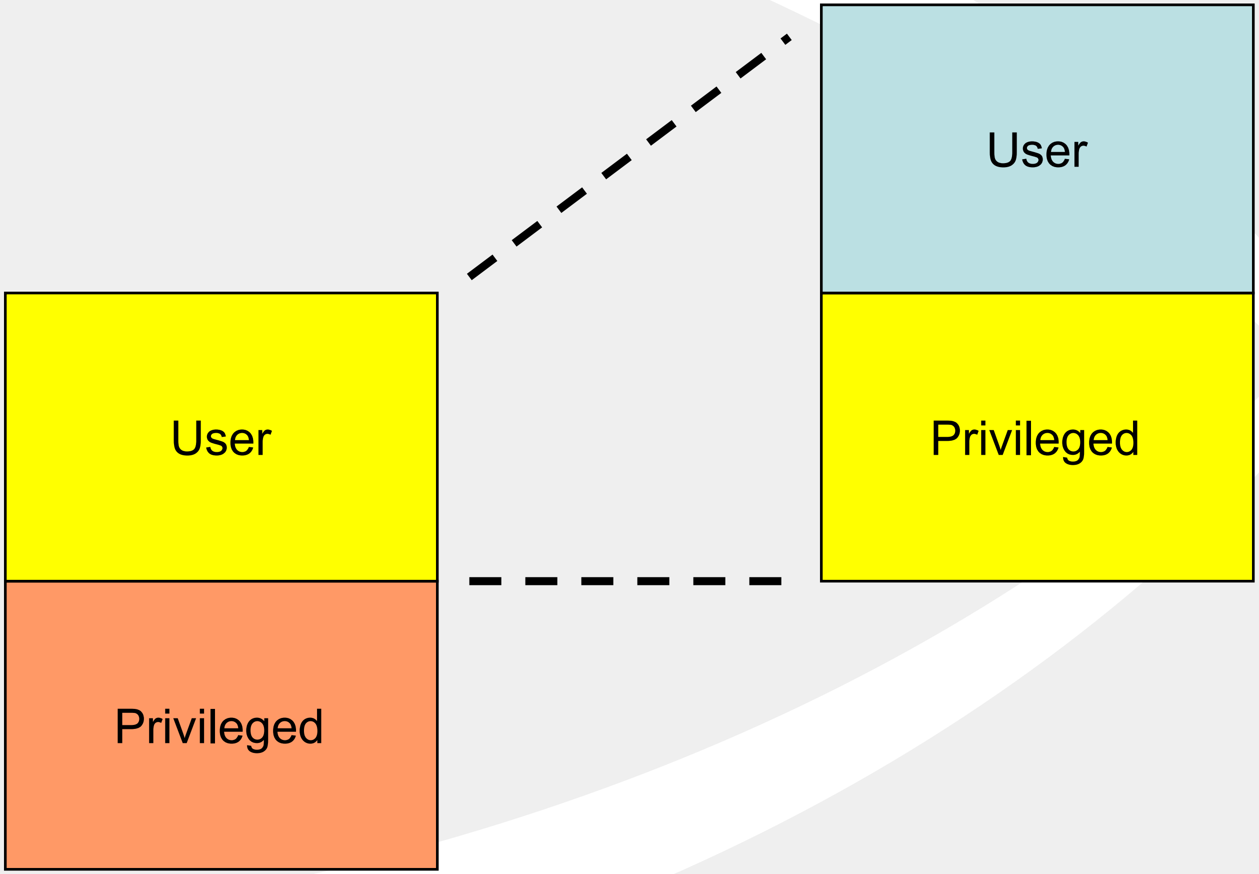
Benefits and Features

- Host system protected from VMs, VMs protected from each other
  - I.e. A virus less likely to spread
  - Sharing is provided though 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

How?



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

Requirements

- A virtual machine is an efficient, isolated duplicate of real machine

Sensitive Instructions

- Control-sensitive instructions
  - affect the allocation of resources available to the virtual machine
  - change processor mode without causing a trap
- Behavior-sensitive instructions
  - effect of execution depends upon location in real memory or on processor mode

Privileged Instructions

- Cause a fault in user mode
- Work fine in privileged mode

Theorem (!)

- For any conventional third-generation computer, a virtual machine monitor may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions.

IBM 360

The (Real) 360 Architecture

- Two execution modes
  - supervisor and problem (user)
  - **all sensitive instructions are privileged instructions**
- Memory is protectable: 2k-byte granularity
- All interrupt vectors and the clock are in first 512 bytes of memory
- I/O done via channel programs in memory, initiated with privileged instructions
- Dynamic address translation (virtual memory) added for Model 67

Actions on Real 360

	User mode	Privileged mode
non-sensitive instruction	executes fine	executes fine
errant instruction	traps to kernel	traps to kernel
sensitive instruction	traps to kernel	executes fine

Actions on Virtual 360

	User mode	Privileged mode
non-sensitive instruction	executes fine	executes fine
errant instruction	traps to VMM; VMM causes trap to occur on guest OS	traps to VMM; VMM causes trap to occur on guest OS
sensitive instruction	traps to VMM; VMM causes trap to occur on guest OS	traps to VMM; VMM verifies and emulates instruction

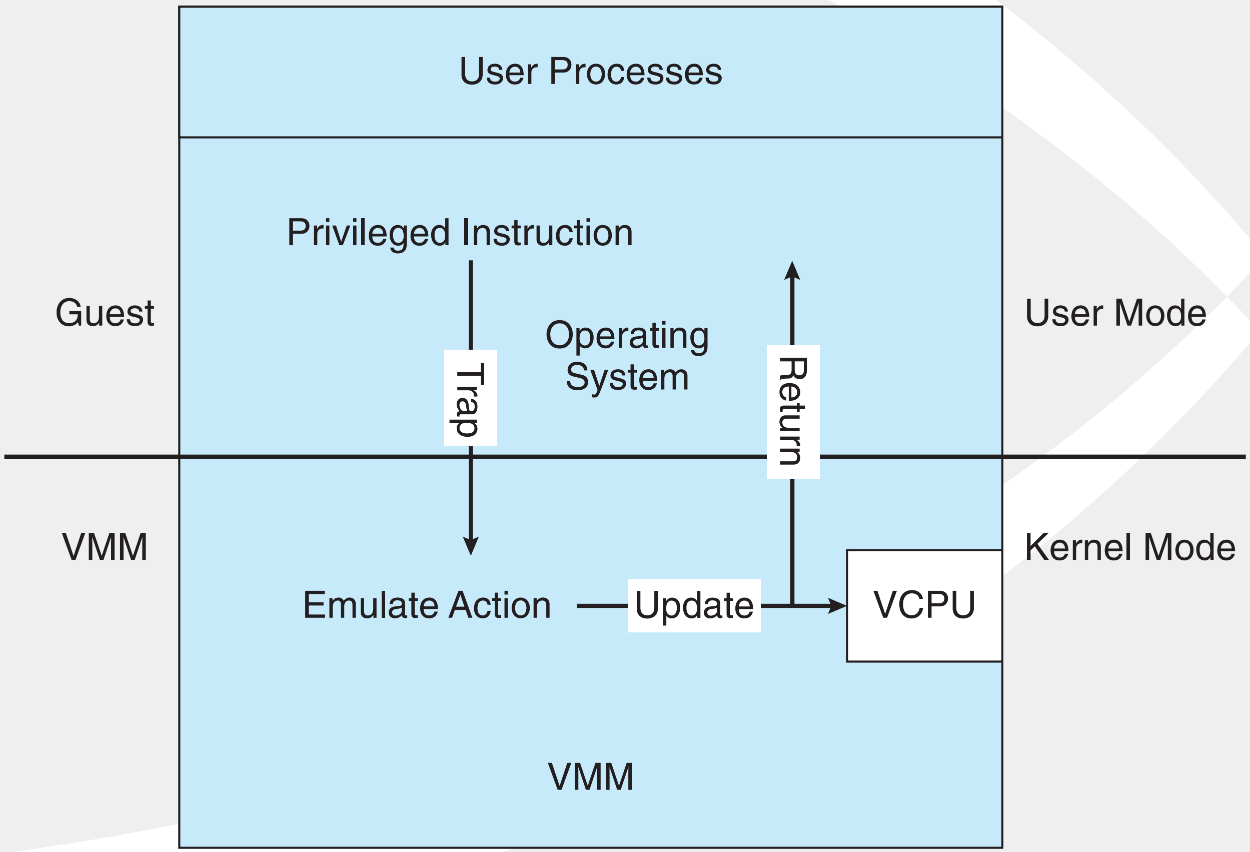
Building Block – Trap and Emulate

- 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.)

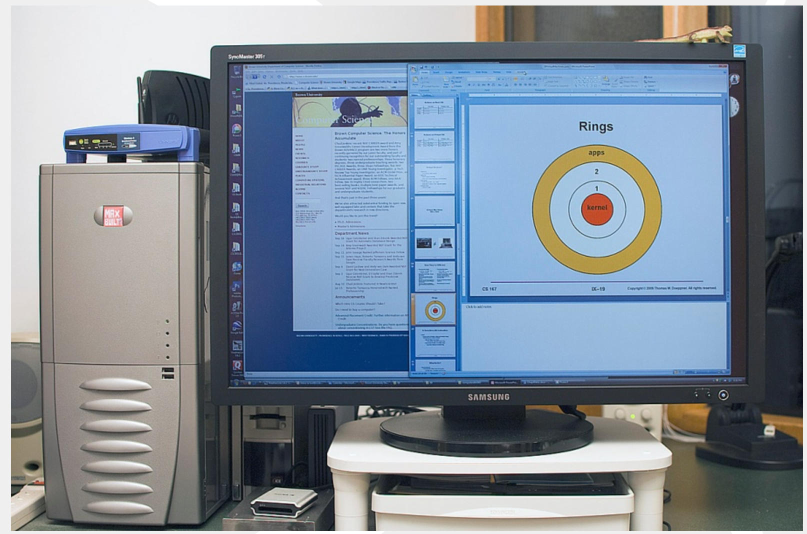
- How does switch from virtual user mode to virtual kernel mode occur?
  - Attempting a privileged instruction in user mode causes an error -> trap
  - VMM gains control, analyzes error, executes operation as attempted by guest
  - Returns control to guest in user mode
  - Known as **trap-and-emulate**
  - 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, mode CPU modes to improve virtualization performance

Trap-and-Emulate Virtualization Implementation



IBM 360 versus Intel x86

≠



How They're Different

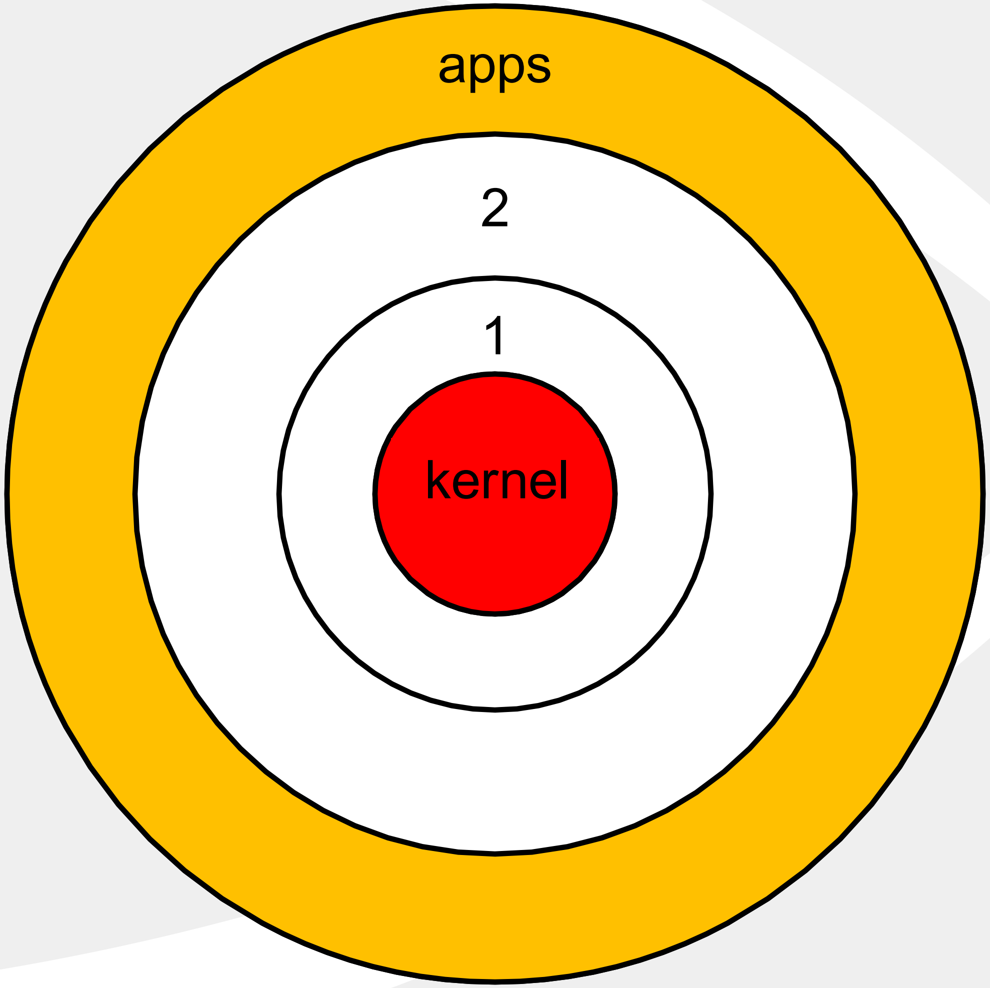
IBM 360

- Two execution modes
  - supervisor and problem (user)
  - all sensitive instructions are privileged instructions
- Memory is protectable: 2k-byte granularity
- All interrupt vectors and the clock are in first 512 bytes of memory
- I/O done via channel programs in memory, initiated with privileged instructions
- Dynamic address translation (virtual memory) added for Model 67

Intel x86

- Four execution modes
  - rings 0 through 3
  - not all sensitive instructions are privileged instructions
- Memory is protectable: segment system + virtual memory
- Special register points to interrupt table
- I/O done via memory-mapped registers
- Virtual memory is standard

Rings



A Sensitive x86 Instruction

- popf
  - pops word off stack, setting processor flags according to word's content
    - sets all flags if in ring 0
      - including interrupt-disable flag
    - just some of them if in other rings
      - ignores interrupt-disable flag

What to Do?

- Binary rewriting
  - rewrite kernel binaries of guest OSes
    - replace sensitive instructions with hypercalls
    - do so dynamically
- Hardware virtualization
  - fix the hardware so it's virtualizable
- Paravirtualization
  - virtual machine differs from real machine
    - provides more convenient interfaces for virtualization
    - *hypervisor* interface between virtual and real machines
    - guest OS source code is modified

Binary Rewriting

- Privilege-mode code run via binary translator
  - replaces sensitive instructions with hypercalls
  - translated code is cached
    - usually translated just once
  - VMWare
  - U.S. patent 6,397,242

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

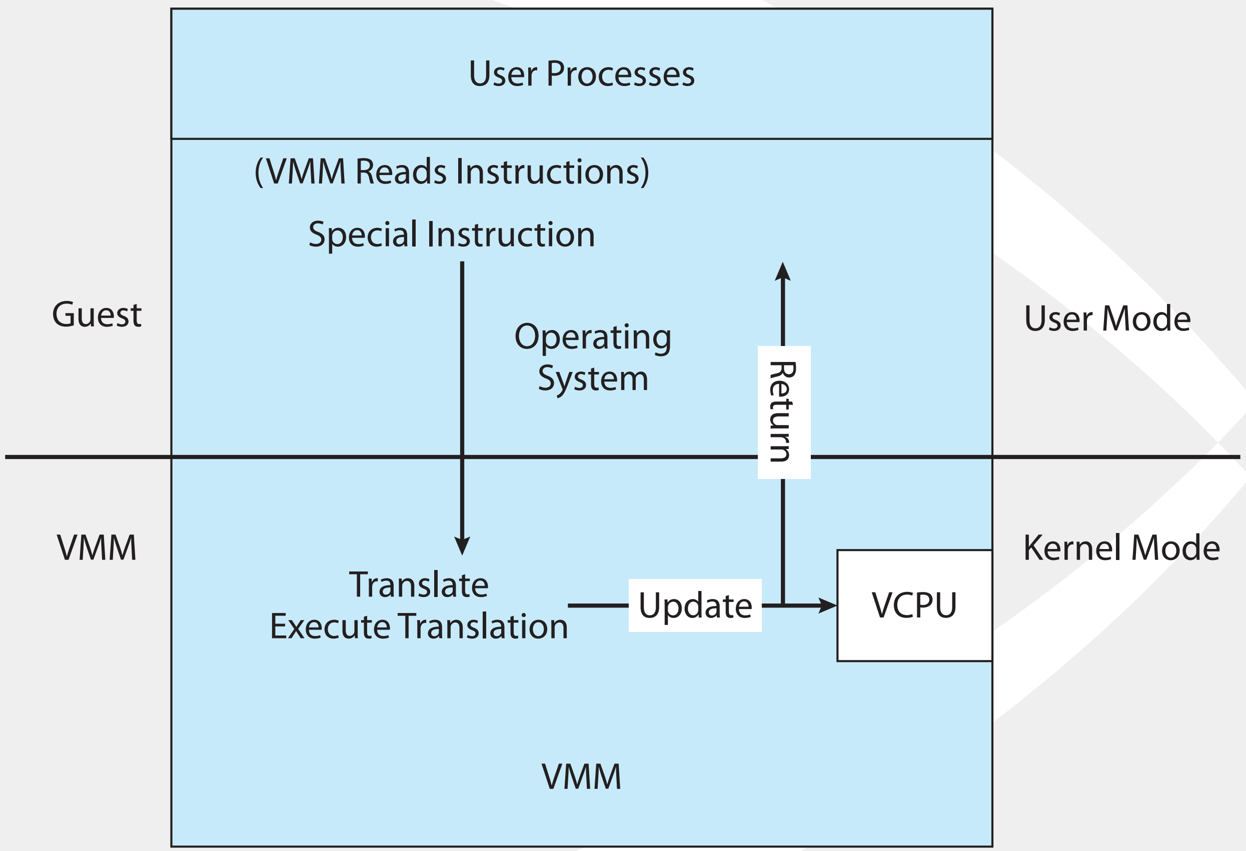
Binary Translation (cont.)

- Other similar problem instructions we will call ***special instructions***
  - Caused trap-and-emulate method considered impossible until 1998
- Binary translation solves the problem
  - Basics are simple, but implementation very complex
  1. If guest VCPU is in user mode, guest can run instructions natively
  2. If guest VCPU in kernel mode (guest believes it is in kernel mode)
     1. VMM examines every instruction guest is about to execute by reading a few instructions ahead of program counter
     2. Non-special-instructions run natively
     3. 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



Fixing the Hardware

- Intel Vanderpool technology: VT-x
  - new processor mode
    - “ring -1”
      - *root* mode
      - other modes are *non-root*
  - certain events in non-root mode cause *VM-exit* to root mode
    - essentially a hypercall
    - code in root mode specifies which events cause VM-exits
  - non-VMM OSes must not be written to use root mode!

Paravirtualization

- Sensitive instructions replaced with hypervisor calls
  - traps to VMM
- Virtual machine provides higher-level device interface
  - guest machine has no device drivers