Virtual Machines

Part 1: 61 years ago

It’s 1964 …

- The Beatles appear on the Ed Sullivan show

It's 1964 …

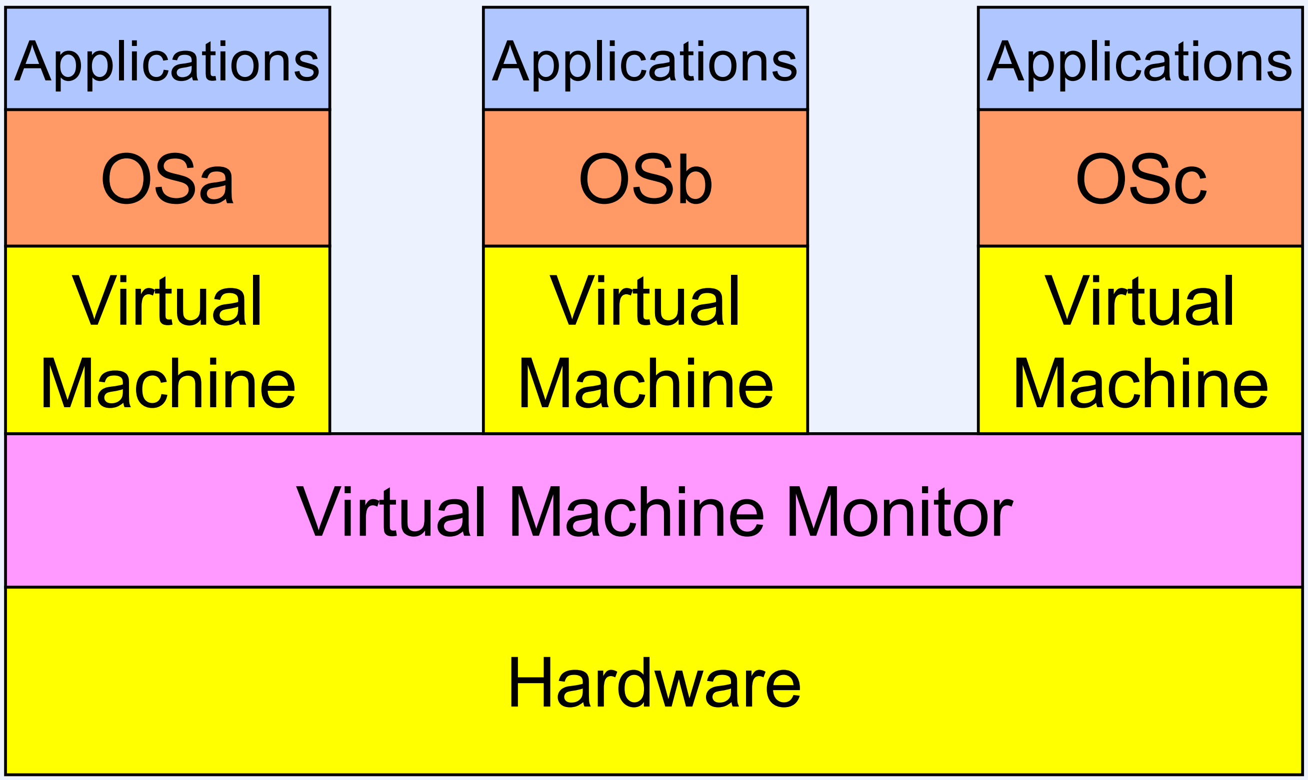
- The Beatles appear on the Ed Sullivan show

It's 1964 …

- The Beatles appear on the Ed Sullivan show
- IBM wants a multiuser time-sharing system

- TSS project
  - large, monolithic system
  - lots of people working on it
  - for years
  - total, complete flop
- CMS
  - single-user time-sharing system for IBM 360
- CP67
  - virtual machine monitor (VMM)
  - supports multiple virtual IBM 360s
- Put the two together …
  - a (working) multiuser time-sharing system

Virtual Machines



Why?

- Structuring technique for a multi-user system
- OS debugging and testing
- Multiple OSes on one machine
- Adapt to hardware changes in software
- Server consolidation and service isolation
- It's cool

User vs. Privileged Mode

- **Privileged mode**
  - may run all instructions, access all registers and memory
  - for example:
    - modify address translation for virtual memory
    - access and control I/O devices
    - mask and unmask interrupts
    - start and stop system clock
- **User mode**
  - may run only "innocuous" instructions
  - may access only normal registers and certain designated memory

How?

- **Approach 1**
  - system has “normal” scheduler and virtual memory
  - its processes run in privileged mode

How?

- **Approach 2**
  - system has “normal” scheduler and virtual memory
  - its processes run an emulator of the real machine

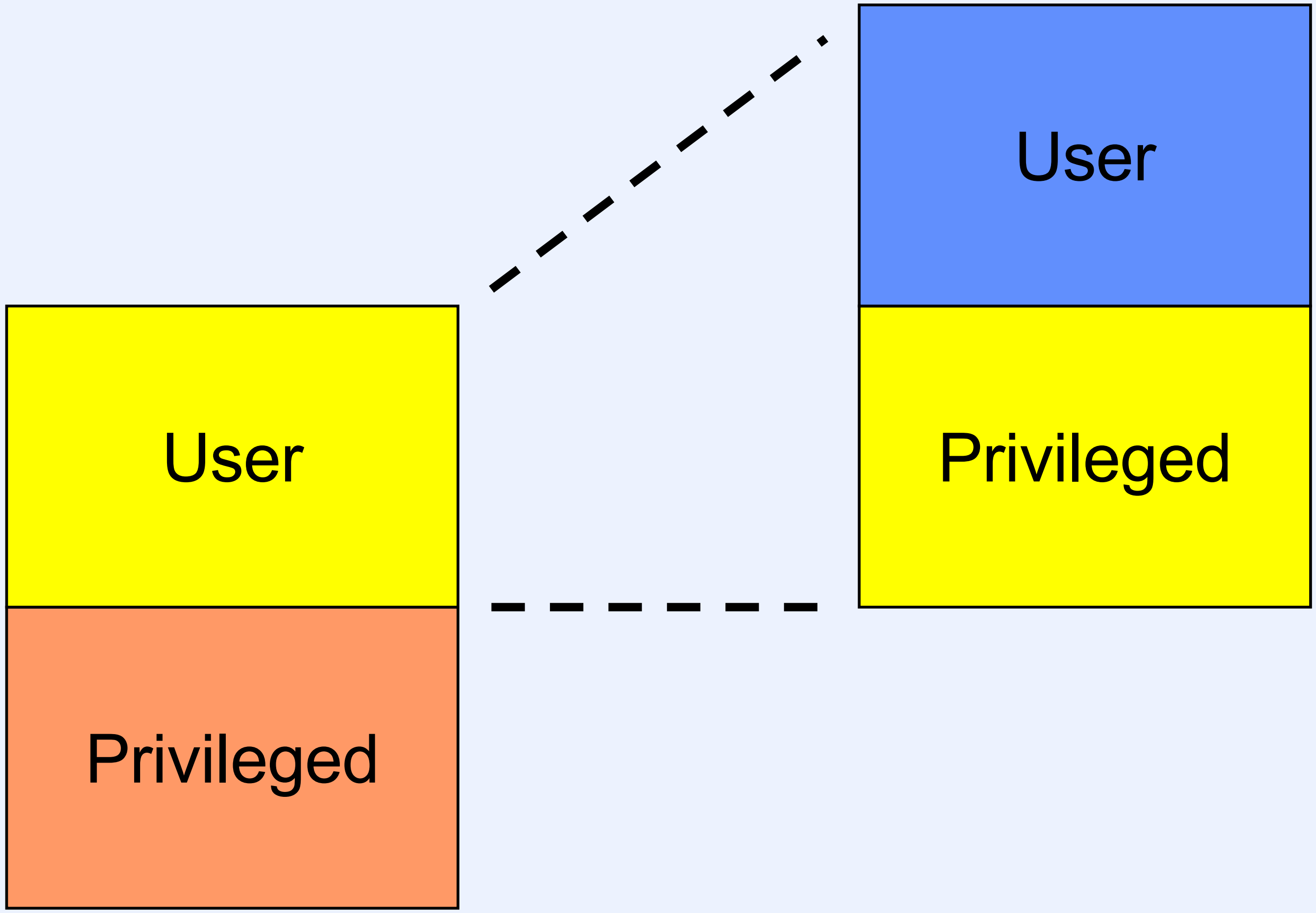
How?

- **Approach 3**
  - system has “normal” scheduler and virtual memory
  - its processes execute user-mode code directly, but run the emulator when going into privileged mode

How?

- Approach 4
  - system has “normal” scheduler and virtual memory
  - its processes execute non-privileged instructions directly, but emulate privileged instructions

How?



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.

Quiz 1

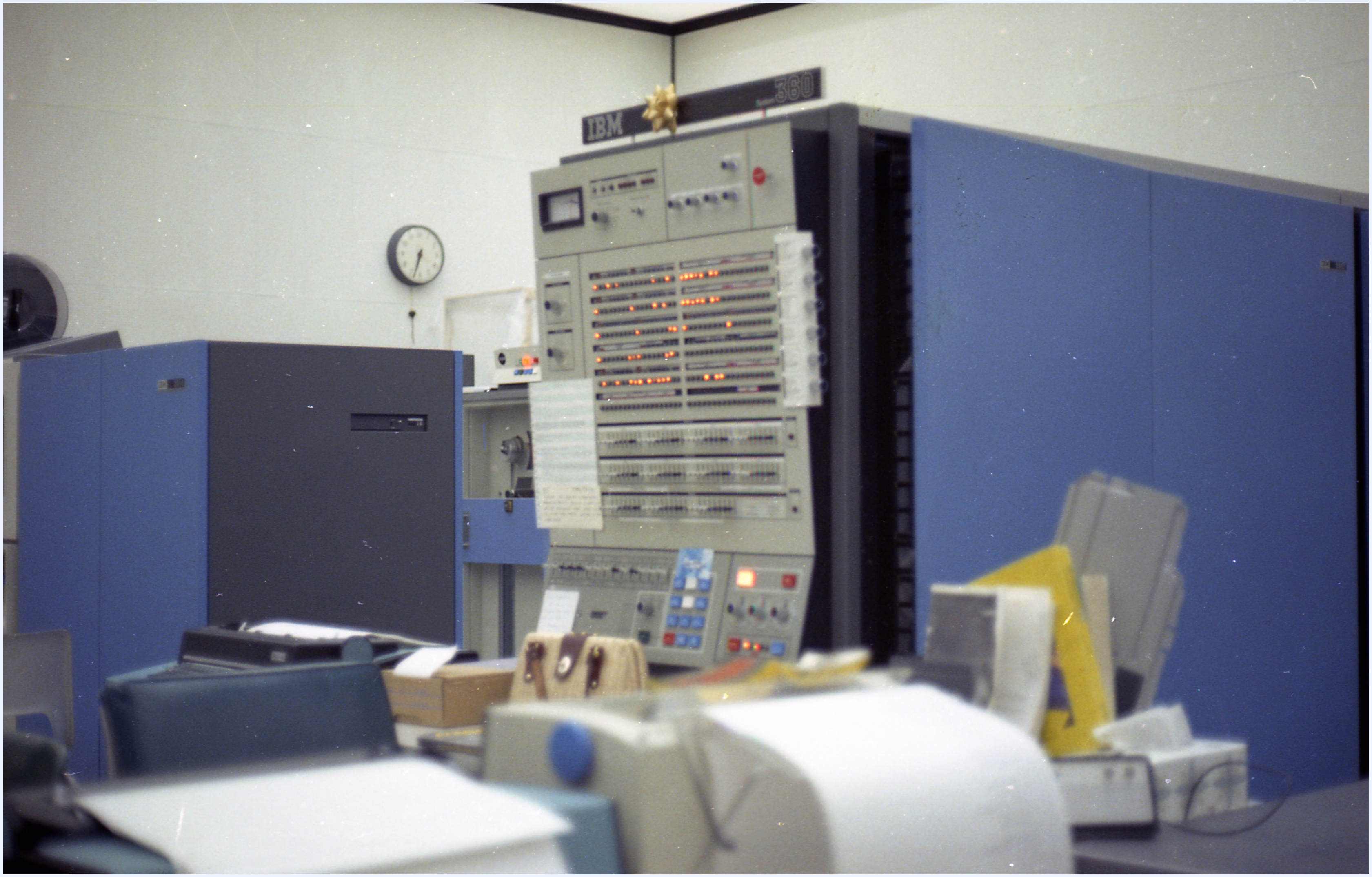
A certain computer has an instruction that alters the values of certain bits in a control register, including the interrupt-enable bit, when executed in privileged mode, but, when executed in user mode, it does everything except for altering the interrupt-enable bit (and causes no trap). Does it satisfy the premise of the theorem? (Assume the computer is “conventional and third-generation”)

a) no, and thus the theorem doesn’t apply

b) yes, and thus the theorem holds in this case

c) yes, but the theorem doesn’t hold and is thus falsified

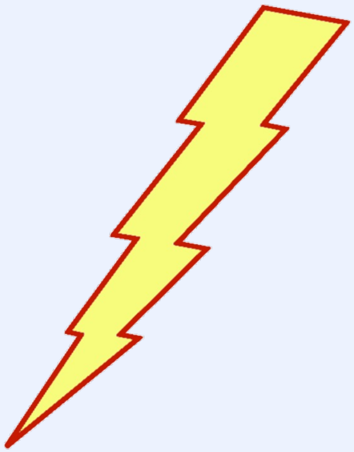
IBM 360/67



The (Real) 360 Architecture

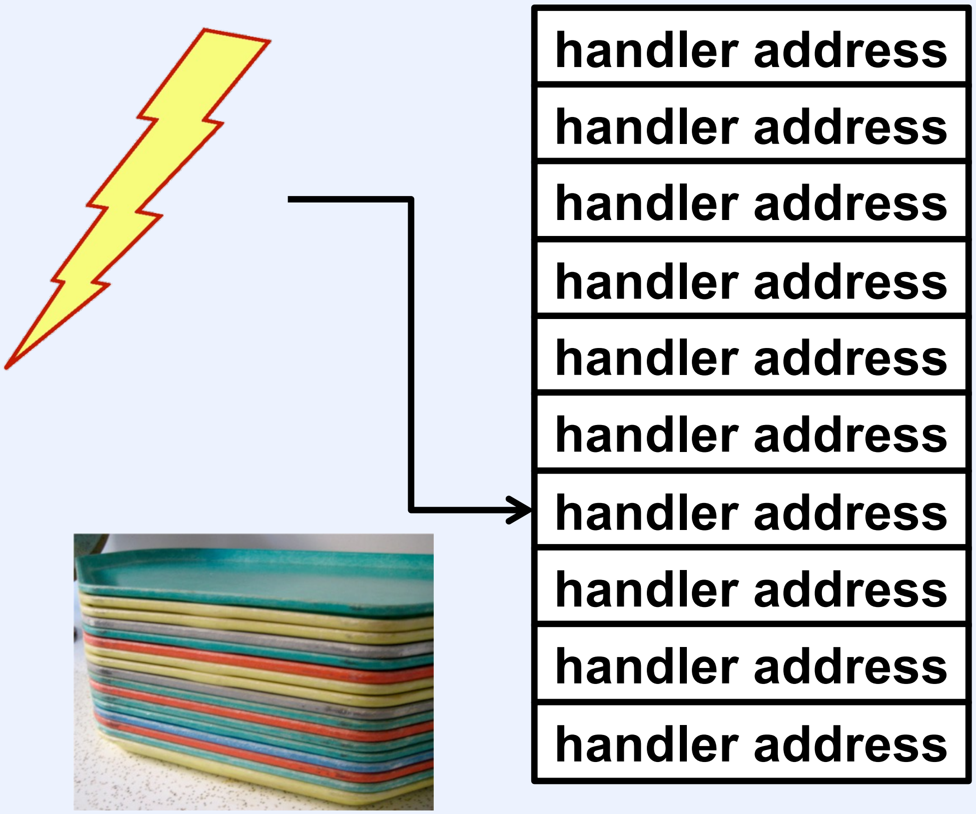
- Two execution modes
  - supervisor and problem (user)
  - all sensitive instructions are privileged instructions
- Memory is protectable: 2k-byte granularity
- All interrupt vectors and the clock are in first 512 bytes of memory
- I/O done via channel programs in memory, initiated with privileged instructions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

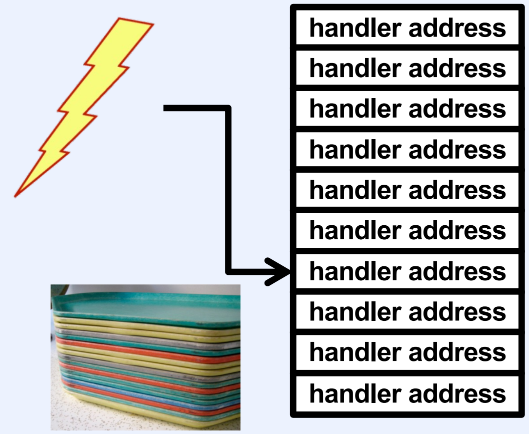
Real Interrupts and Traps

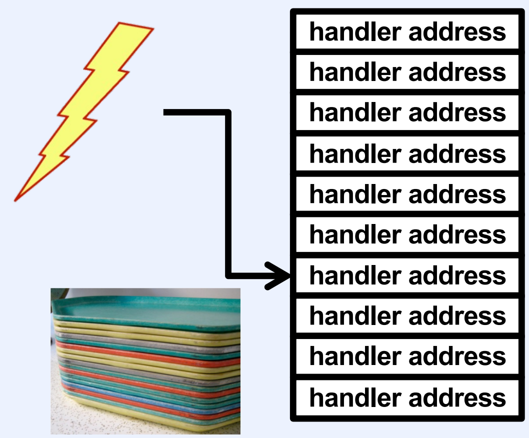


handler address
handler address
handler address
handler address
handler address
handler address
handler address
handler address
handler address
handler address

Virtual Interrupts and Traps







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
access low memory	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
access low memory	traps to VMM; VMM causes trap to occur on guest OS	traps to VMM; VMM verifies and emulates/translates access

Quiz 2

Can a VMM (supporting other virtual machines) run on a virtual machine?

a) it requires some changes to a VMM for it to run on a virtual machine

b) yes, no problem

c) no, can’t be done

Virtual Devices?

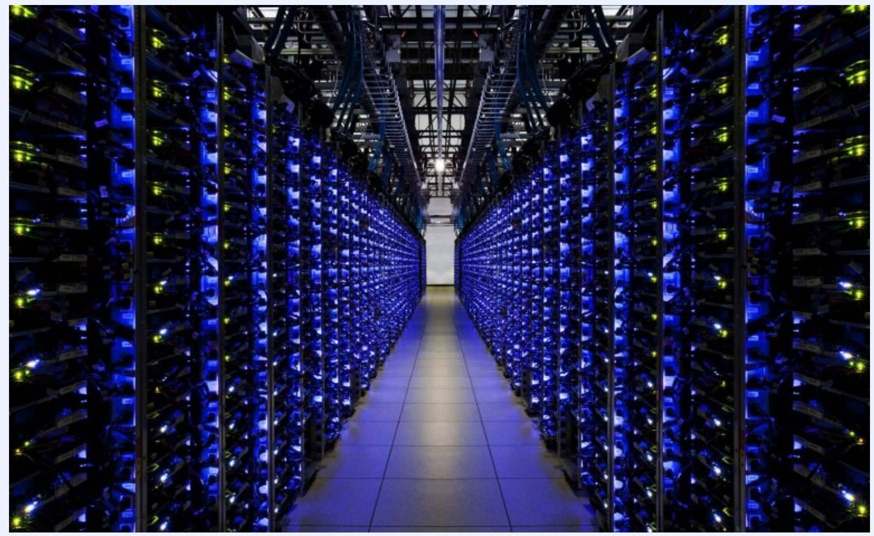
- Terminals
  - connecting (real) people
- Networks
  - didn’t exist in the 60s
  - (how did virtual machines communicate?)
- Disk drives
  - CP67 supported “mini disks”
  - extended at Brown into “segment system”
- Interval timer
  - virtual or real?

Coping

- Invent new devices
  - recognized by VMM as not real, but referring to additional functionality
    - e.g., mini disks
- Provide new VM facilities not present on real machine
  - e.g., Brown segment system
  - special instructions on VM to request service from VMM
    - sort of like system calls (supervisor calls on 360), but ...
      - hypervisor calls
        - 360 had an extra, unused privileged instruction
          - the *diagnose* instruction

Virtual Machines
Part 2: starting ~20 years ago

$\neq$



How They're Different

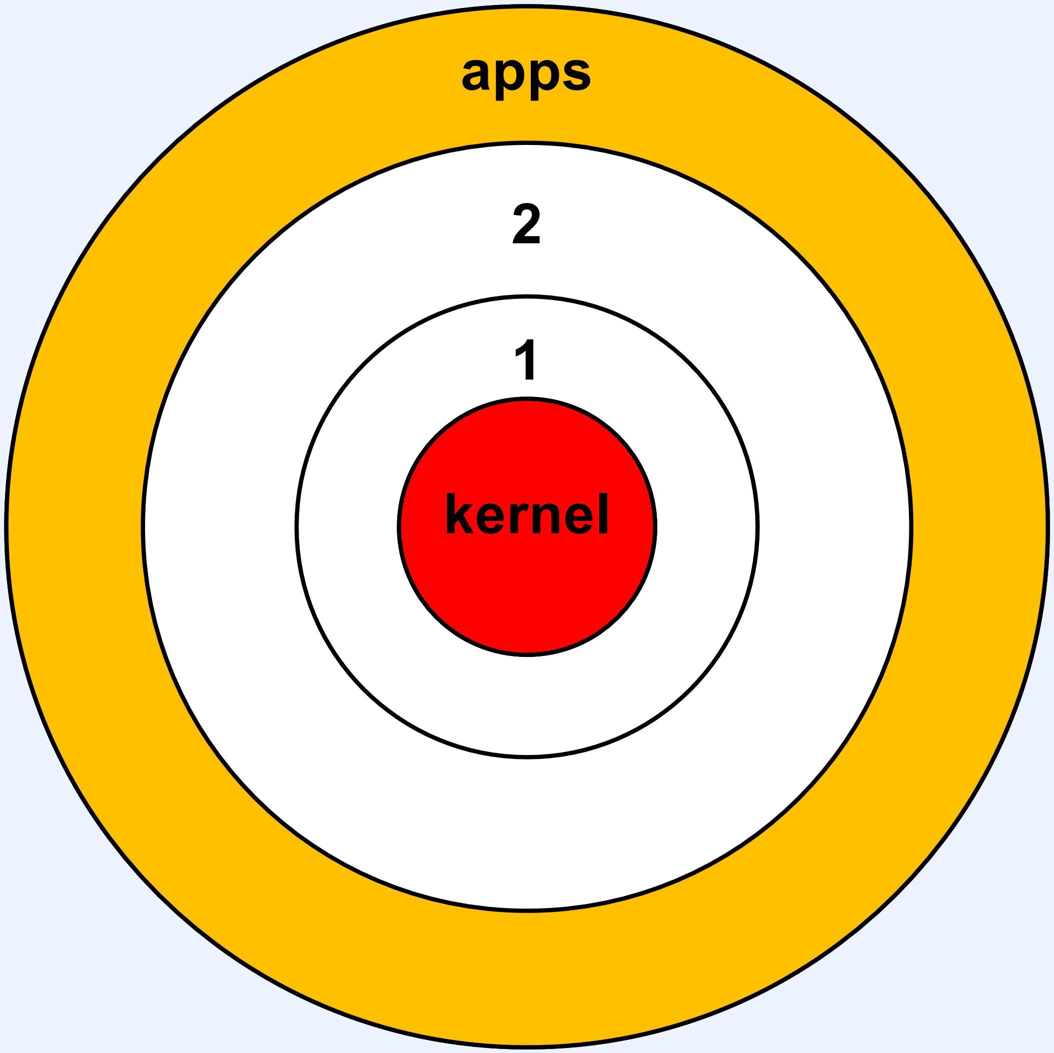
IBM 360

- 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 vector
- 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

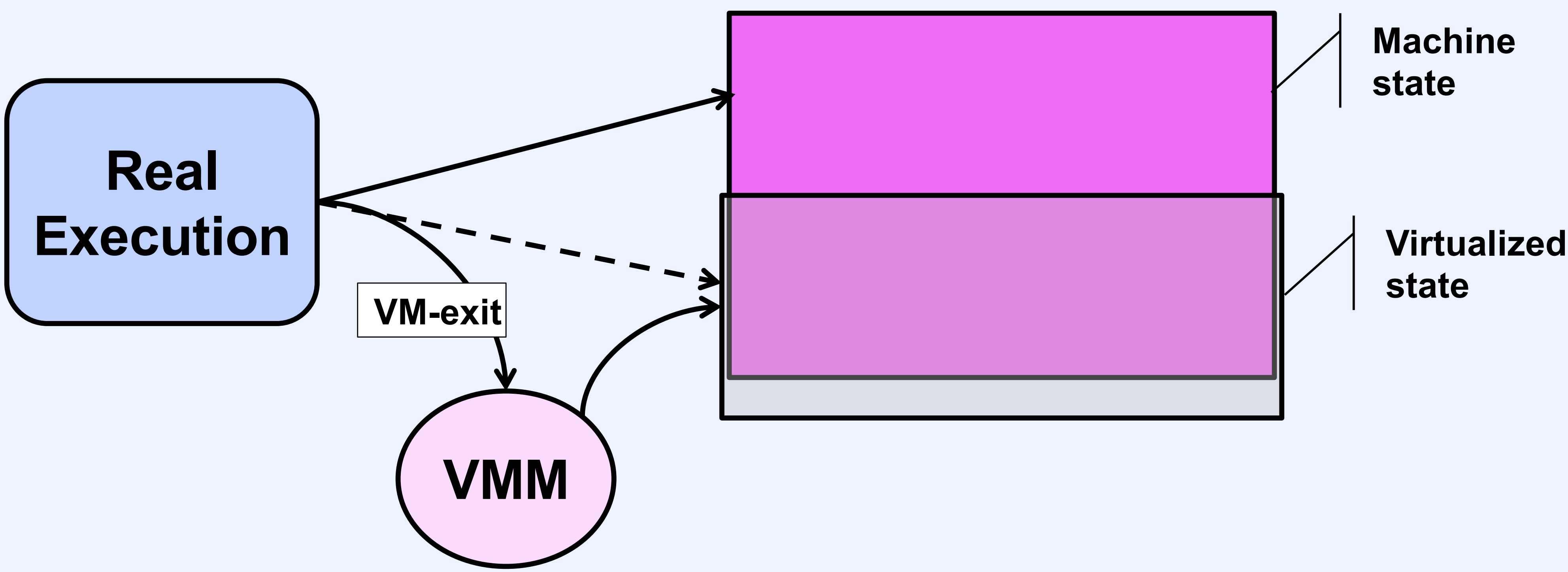
Binary Rewriting

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

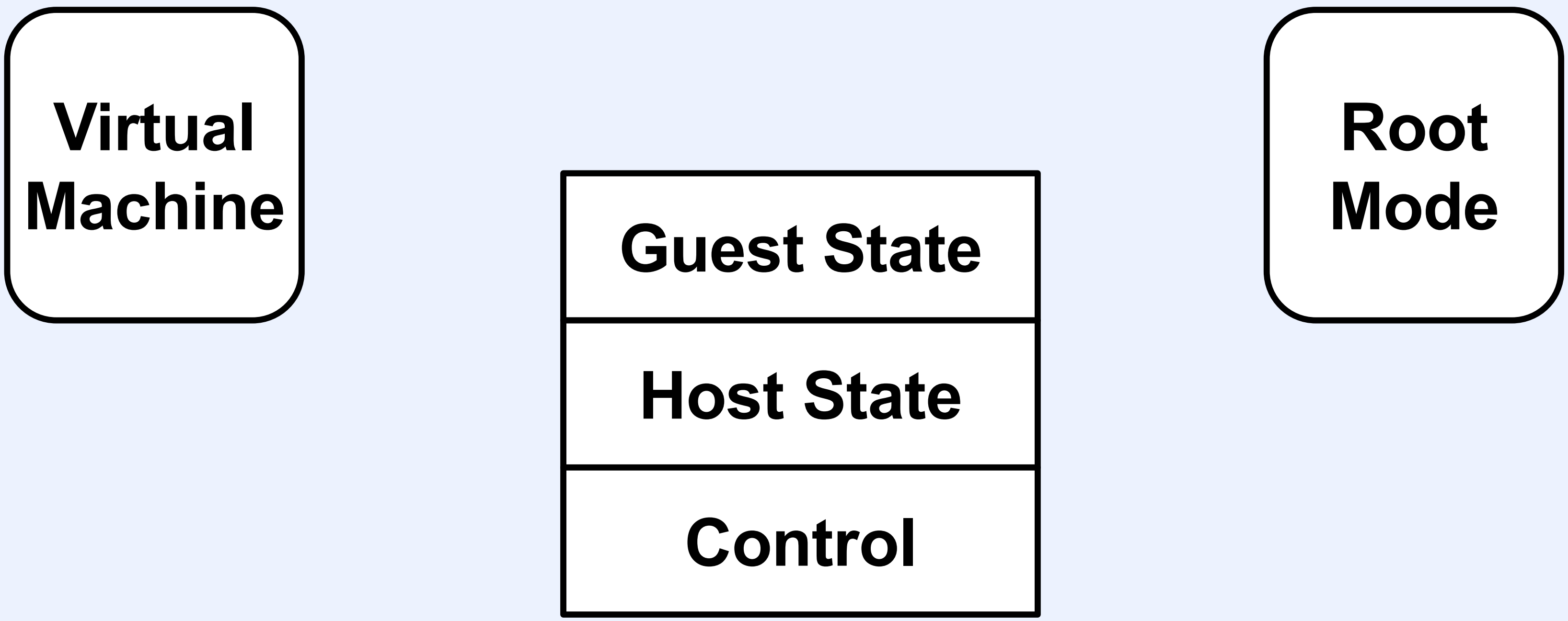
Fixing the Hardware

- Intel Vanderpool technology: VT-x
  - also known as VMX (virtual-machine extensions)
  - 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
    - data structure in root mode specifies which events cause VM-exits
  - non-VMM OSes must be written not to use root mode!

Virtual-Machine State

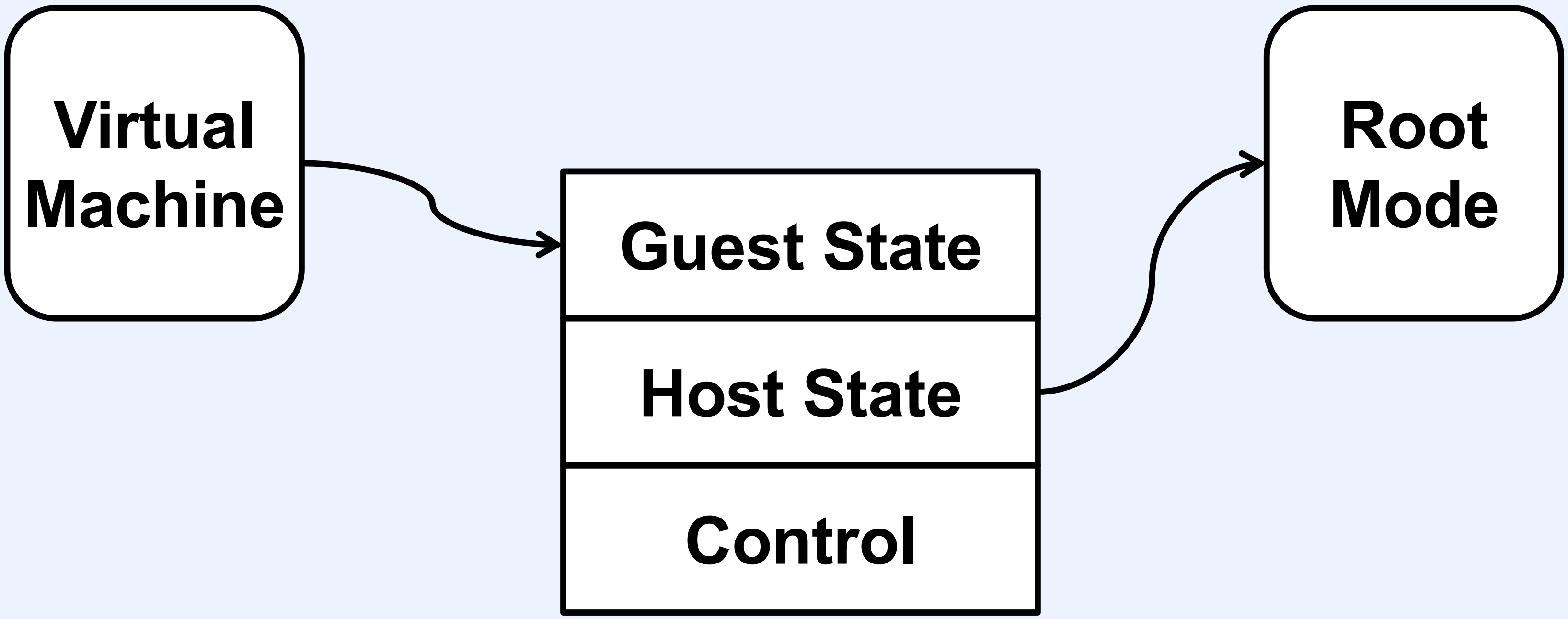


VM Control State



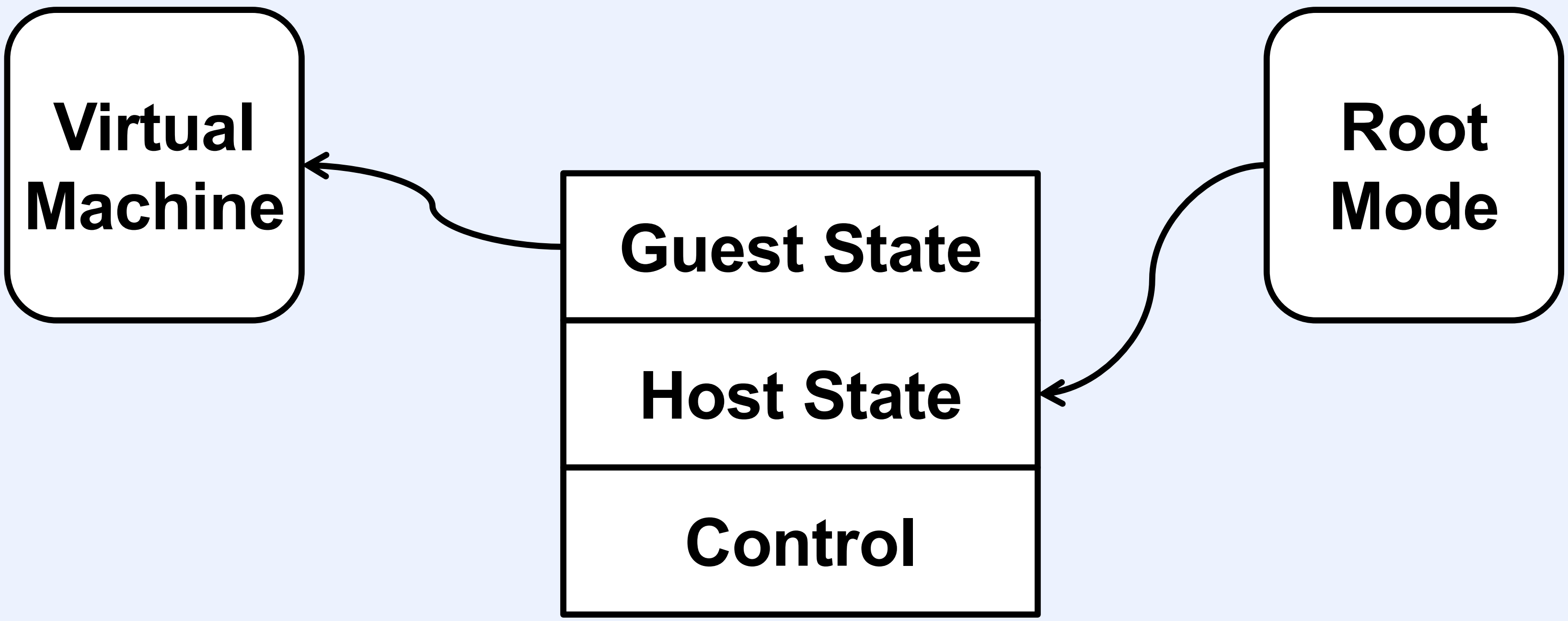
virtual machine
control structure
(VMCS)

VM Control State
VM-Exit



virtual machine control structure (VMCS)

VM Control State
VM-Entry



virtual machine
control structure
(VMCS)

Examples

- mov instruction
  - `mov $2, %rax`
    - no VM-exit
  - `mov $2, %CR3`
    - VM-exit (if desired)
- interrupts
  - interrupt occurs
    - VM-exit (always)
  - `popf` in ring 0
    - affects interrupt-disable flag on guest, no effect on real machine
    - no VM-exit

Quiz 3

We've implemented recursive virtualization: VMM_i runs on a VM supported by VMM_{i-1}, which runs on a VM supported by VMM_{i-2}, …, which runs on a VM supported by VMM_0, which runs on the real hardware. A VM-Exit takes place on a VM running on VMM_i.

a) It's handled first on VMM_0, is then handled on VMM_1, …, and finally on VMM_i.

b) It's handled first on VMM_i, which then VM-Exits to VMM_{i-1}, which the VM-Exits to VMM_{i-2}, …, which VM-exits to VMM_0.

