# System Virtual Machines

**(same ISA)**

ISTITUTO DI SCIENZA E TECNOLOGIE
DELL'INFORMAZIONE "A. FAEDO"

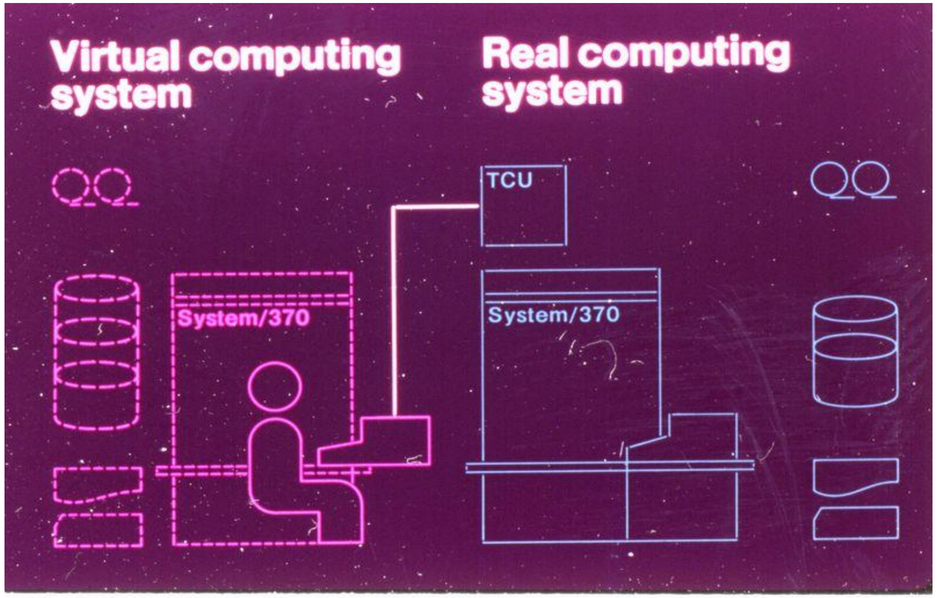
UNIVERSITÀ DI PISA

# Classical Virtualization

- Popek & Goldberg, 1974

A virtual machine is taken to be *an efficient, isolated duplicate* of the real machine. We explain these notions through the idea of a *virtual machine monitor* (VMM). See Figure 1. As a piece of software a VMM has three essential characteristics. First, the VMM provides an environment for programs which is essentially identical with the original machine; second, programs run in this environment show at worst only minor decreases in speed; and last, the VMM is in complete control of system resources.

- Fidelity
  - Run any software
- Performance
  - Run it fast
- Safety and Isolation
  - VMM manages all hardware



# Resource Control

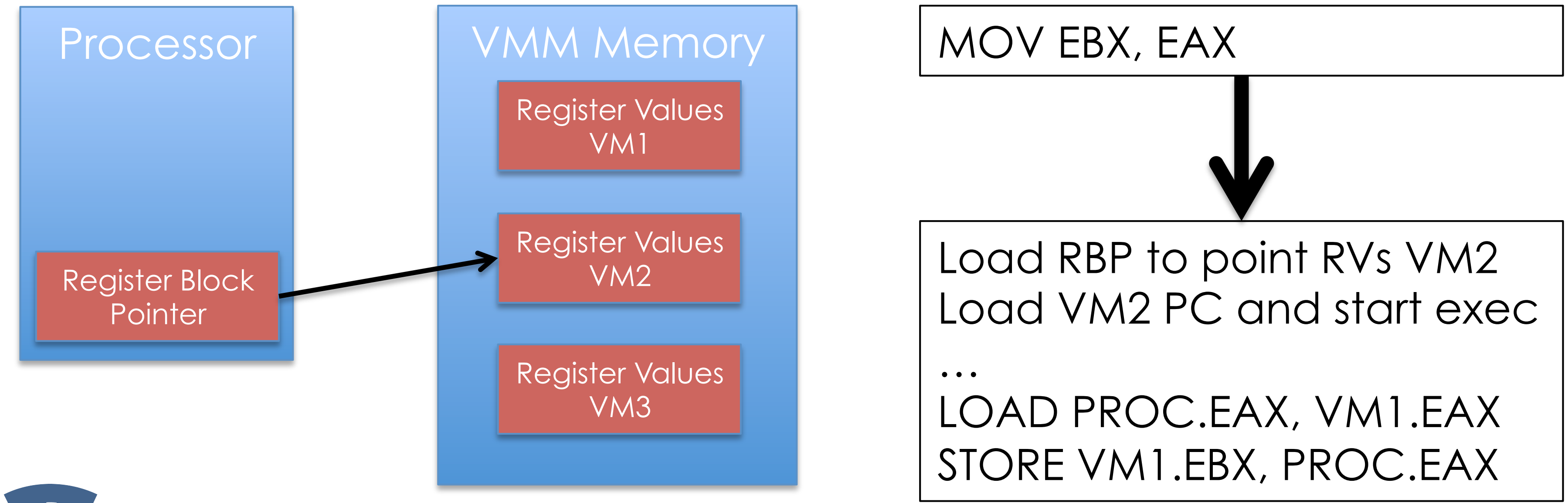
- VMM must maintain *overall control* of the hardware resources
  - Hardware resources are assigned to VMs when they are created/ executed
  - Should have a way to get them back when they need to assigned to a different VM
  - Similar to multi-programming in OS
- Privileged Resources
  - Certain resources are accessible only to and managed by VMM
  - Interrupts relating to such resources must then be handled by VMM
  - Privileged resources are emulated by VMM for the VM
- All resource that could help maintain control are marked privileged
  - "Interval timer" is used to decide VM scheduling
  - "Page table base register" (CR3 on x86) is used to isolate VM memory

# State Management

- Each VM would have its *own* architected state information
  - Example: registers/memory/disks, page table/TLB
- Not always possible to map all architected states to its natural level in the host
  - Insufficient/Unavailable host resources
  - Example: Registers of a VM may be architected using main memory in the host
- VMs keep getting switched in/out by the VMM
  - "Isomorphism" requires all state transitions to be performed on the VM states
- State Management: *Indirection* Vs. *Copying*

# Indirection

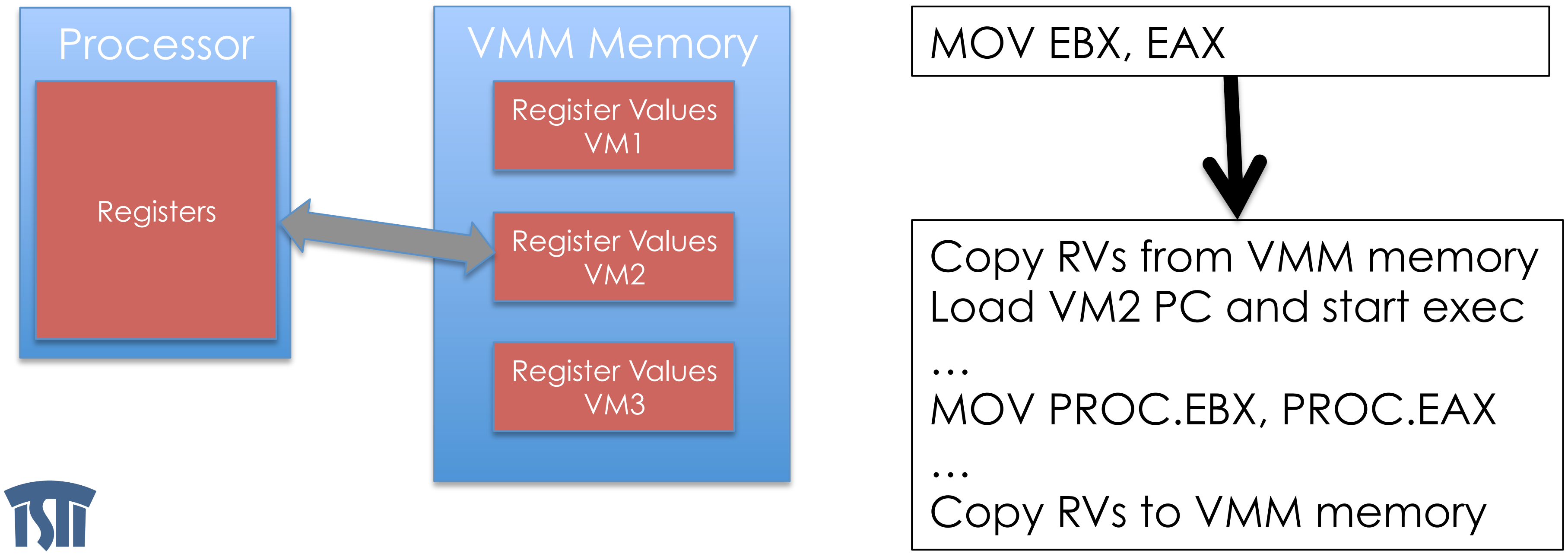
- Indirection
  - Hold *state* for each VM in fixed locations in the host's memory hierarchy
  - A *pointer* managed by VMM indicating the guest state that is currently *active*
  - Analogous to page table pointer in virtual memory systems
  - Pros: Ease of management
  - Cons: Inefficient (*mov eax ebx* requires 2 inst)




ISTITUTO DI SCIENZA E TECNOLOGIE DELL'INFORMAZIONE "A. FAEDO"

# Copying

- Copying
  - Copy VM's state information to its natural level in memory hierarchy when *switched in*
  - Copy them back to the original place when *switched out*
  - Example: Copy all the VM registers to the processor registers
  - Pros: Efficient (most instructions are executed natively)
  - Cons: Copying overhead



# Processor Virtualization

# Classes of instructions

- PRIVILEGED instructions trap if executed in user mode and do not trap if executed in kernel mode
- SENSITIVE instructions interact with hardware
  - CONTROL-sensitive instructions attempt to change the configuration of resources in the system
  - BEHAVIOR-sensitive instructions have their result depending on the configuration of resources (e.g. mode of operation)
- INNOCUOUS instructions are not sensitive

# Popek & Goldberg Theorem (1974)

For any conventional third-generation computer a virtual machine monitor with the following properties:

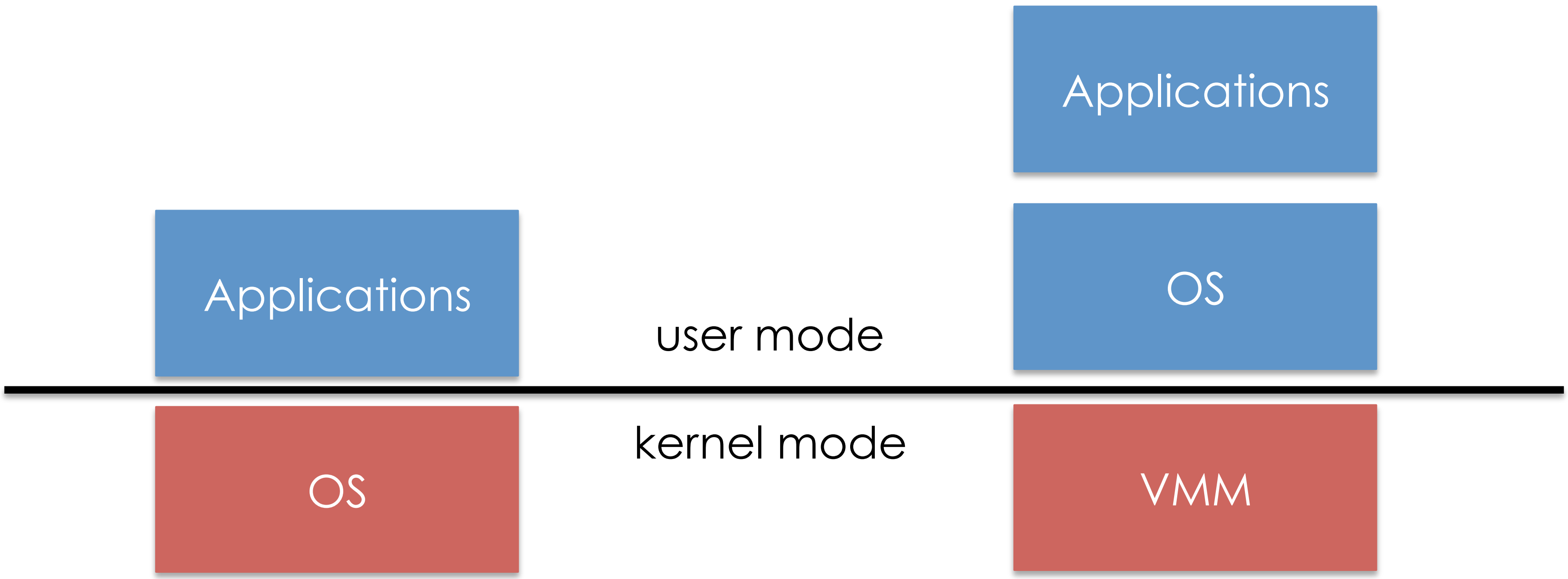
1. Efficiency: innocuous instruction must be executed natively
2. Resource Control: guest can not directly change host resources
3. Equivalence: app behavior in guest must be identical to app behavior in host

may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions

**Full Virtualization**

# Trap & Emulate

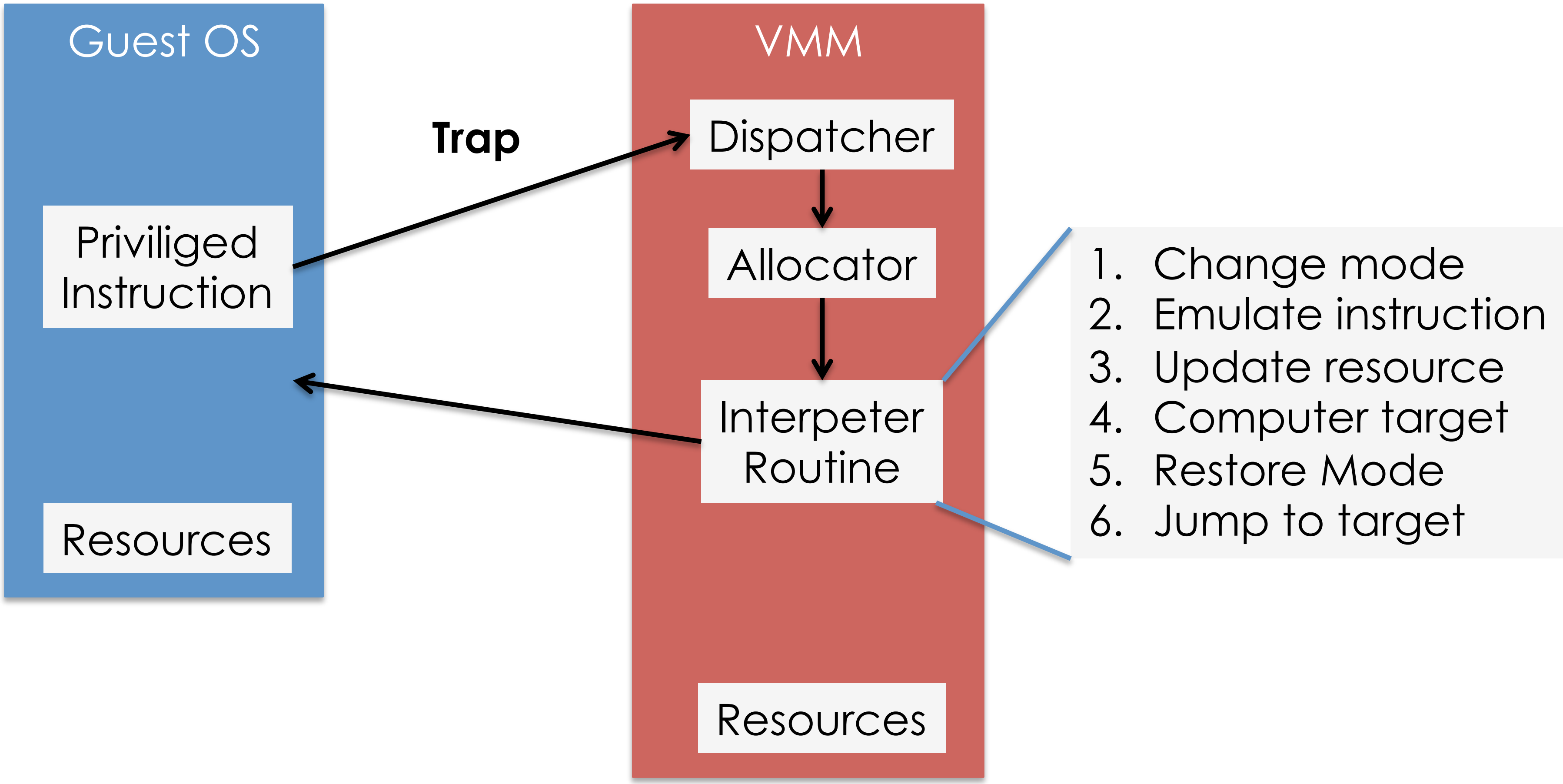
- Must be able to “detect” when VMM must intervene
- Some ISA instructions must be “trapped” and “emulated”
- Must De-Privilege OS
- Very similar to the way programs transfer control to the OS kernel during a system call



# Privileged Resources

- Each VM's privileged state differs from that of the underlying HW.
- Guest-level **primary structures** reflect the state that a guest sees.
- VMM-level **shadow structures** are copies of primary structures.
- Traps occur when **on-chip privileged state** is accessed/modified.
- HW page protection schemes are employed to "detect" when **off-chip privileged** state is accessed/modified

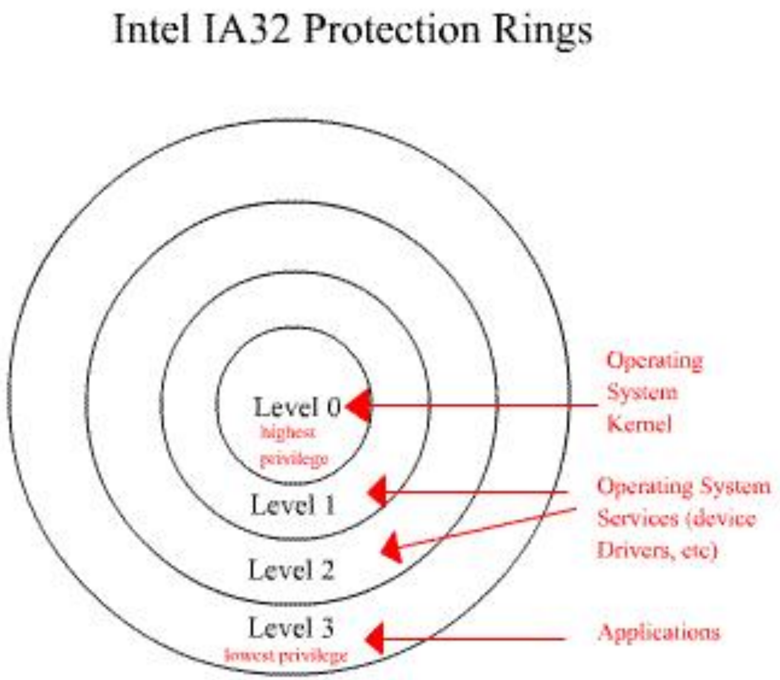
# Handling of Privileged Instructions




**Traps are expensive!**

# Is X86 (fully) virtualizable?

- Lack of trap when priviliged instructions run at user level
- Some privileged instructions execute only in ring 0 but do not fault when executed outside ring 0
- Masking interrupts can only be done in ring 0



# Example: POPF

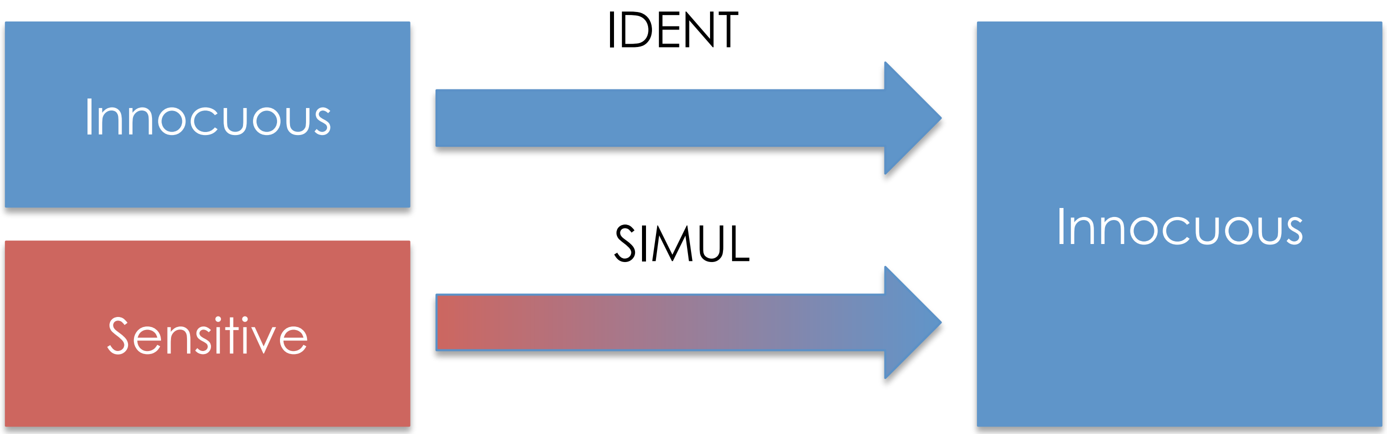
- Same instruction **behaves differently** depending on execution mode
- **User Mode**: changes ALU flags
- **Kernel Mode**: changes ALU and system flags
- Does not generate a trap in user mode

**The IA-32 instruction set contains 17 sensitive, unprivileged instructions**

# Solution

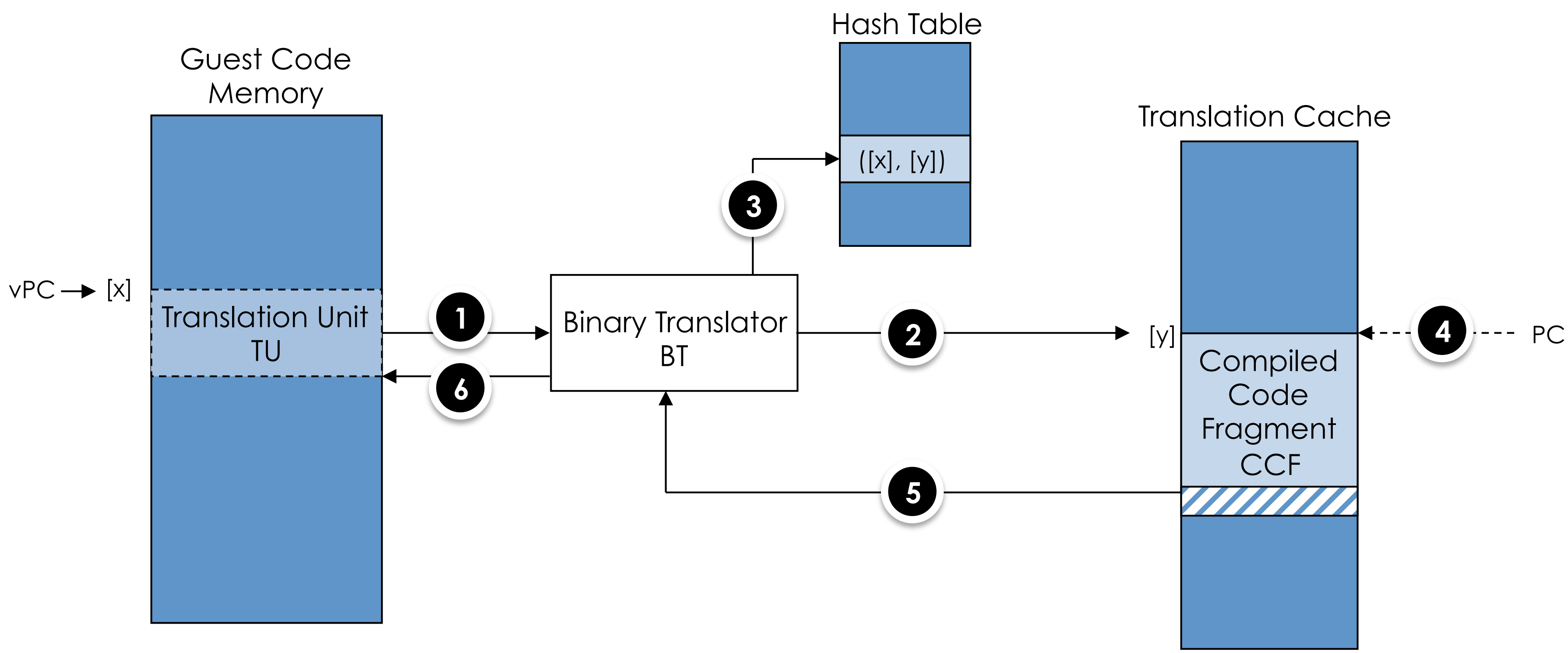
- How can x86's faults be overcome?
- What if guests execute on an interpreter?
- The interpreter can…
  - Prevent leakage of privileged state.
  - Ensure that all sensitive instructions are correctly detected.
- Therefore it can provide…
  - Fidelity
  - Safety
  - Performance??

# Binay Translation

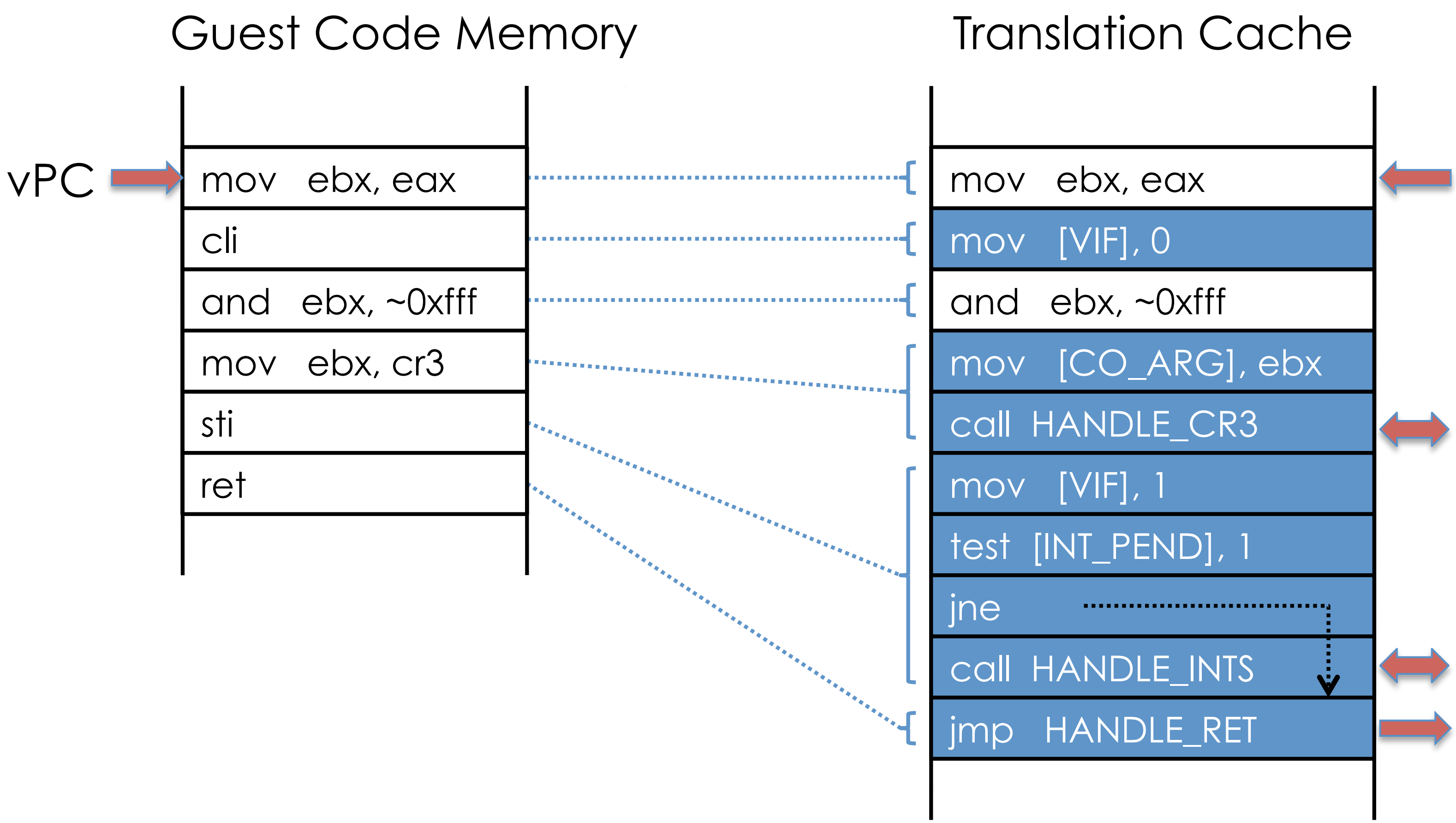



- **Binary** – input is machine-level code
- **Dynamic** – occurs at runtime
- **On demand** – code translated when needed for execution
- **System level** – makes no assumption about guest code
- **Subsetting** – translates from full instruction set to safe subset
- **Adaptive** – adjust code based on guest behavior to achieve efficiency

# Implementation



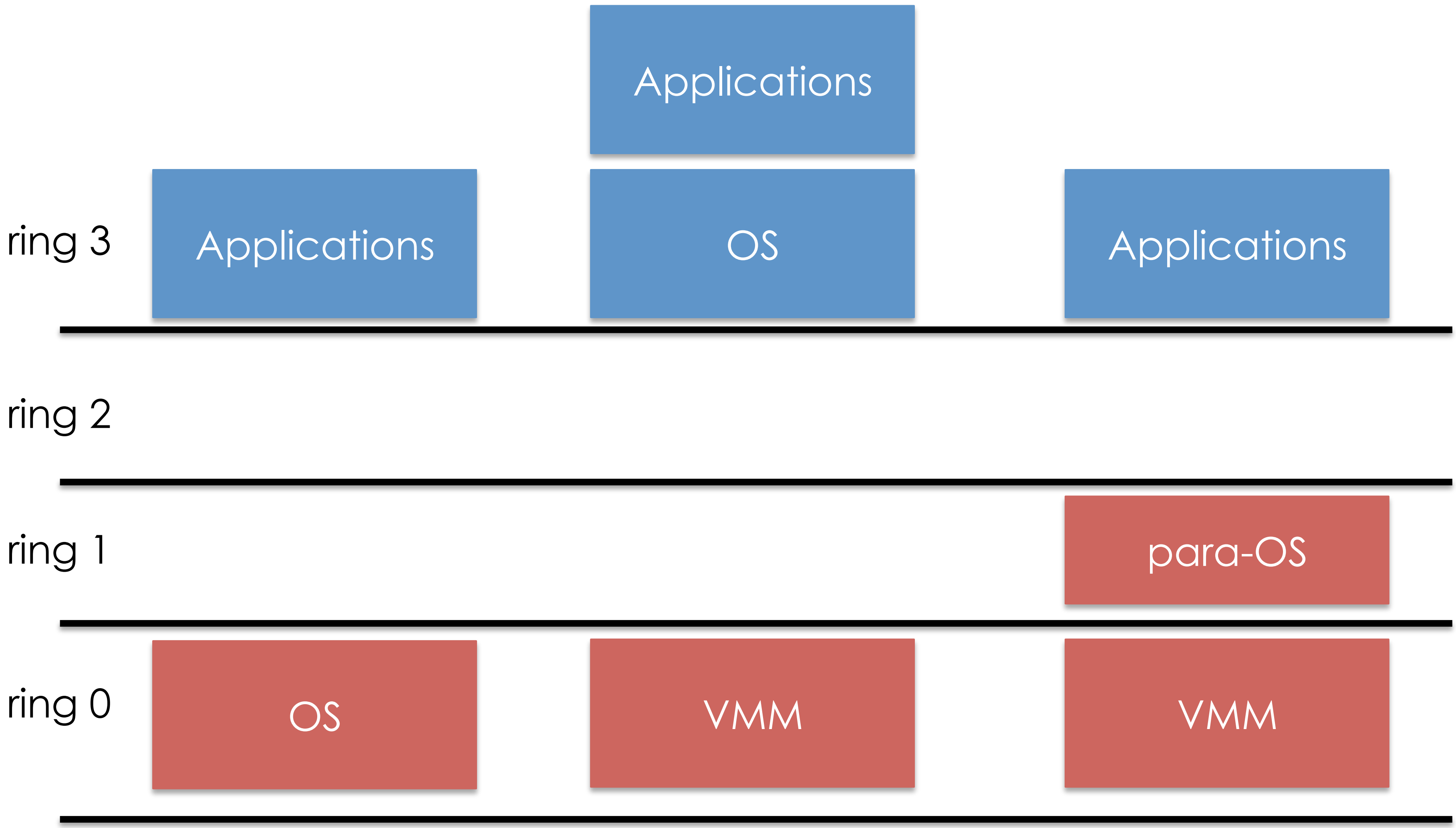
# Example


Guest Code Memory
Translation Cache
vPC
mov ebx, eax
cli
and ebx, ~0xfff
mov ebx, cr3
sti
ret
mov ebx, eax
mov [VIF], 0
and ebx, ~0xfff
mov [CO_ARG], ebx
call HANDLE_CR3
mov [VIF], 1
test [INT_PEND], 1
jne
call HANDLE_INTS
jmp HANDLE_RET

# Issues

- Translation cache index data structure
- Hardware emulation comes with a performance price
- In traditional x86 architectures, OS kernels expect to run privileged code in Ring 0
  - However, because Ring 0 is controlled by the host OS, VMs are forced to execute at Ring 1/3, which requires the VMM to trap and emulate instructions
- Due to these performance limitations, **paravirtualization** and **hardware-assisted virtualization** were developed

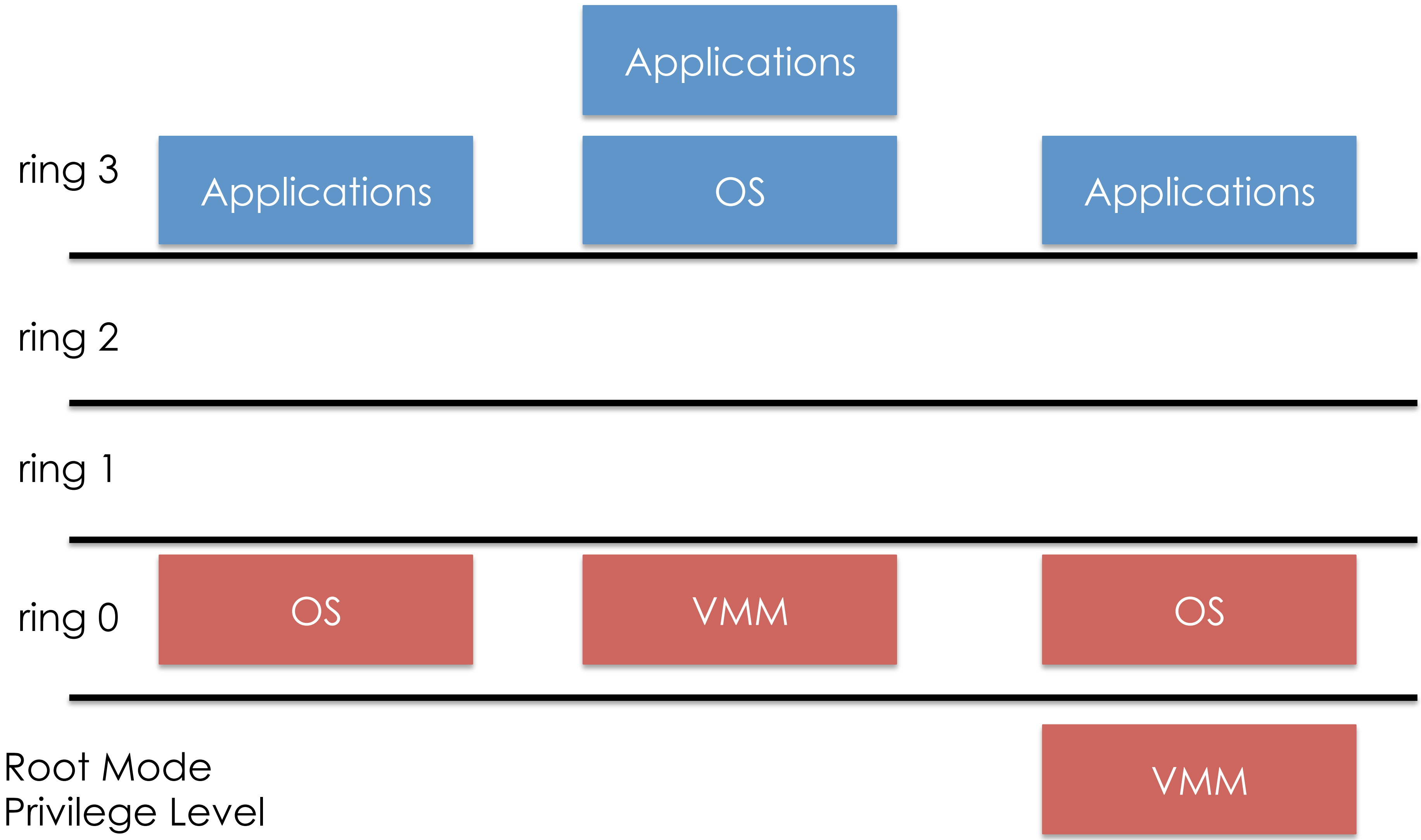
# Paravirtualization



# Drawbacks

- Relies on separate OS kernel for native and in VM
- Tight coupling inhibits compatibility
- Changes to the guest OS are invasive
- Inhibits maintainability and supportability
- Guest kernel must be recompiled when VMM is updated

# Hardware-assisted Virtualization




ISTITUTO DI SCIENZA E TECNOLOGIE
DELL'INFORMAZIONE "A. FAEDO"

# New Hardware Features

- Virtual Machine Control Blocks (VMCBs)
- Root mode privilege level
- Ability to transfer control to/from guest mode.
  - *vmrun* - host to guest.
  - *exit* - guest to host.
- VMM executes *vmrun* to start a guest.
  - Guest state is loaded into HW from in-memory VMCB.
  - Guest mode is resumed and guest continues execution.
- Guests execute until they “toy” with control bits of the VMCB.
  - An *exit* operation occurs.
  - Guest saves data to VMCB.
  - VMM state is loaded into HW - switches to host mode.
  - VMM begins executing.