Virtual Machines
虛擬機器
CSIE 5310

Prof. Shih-Wei Li

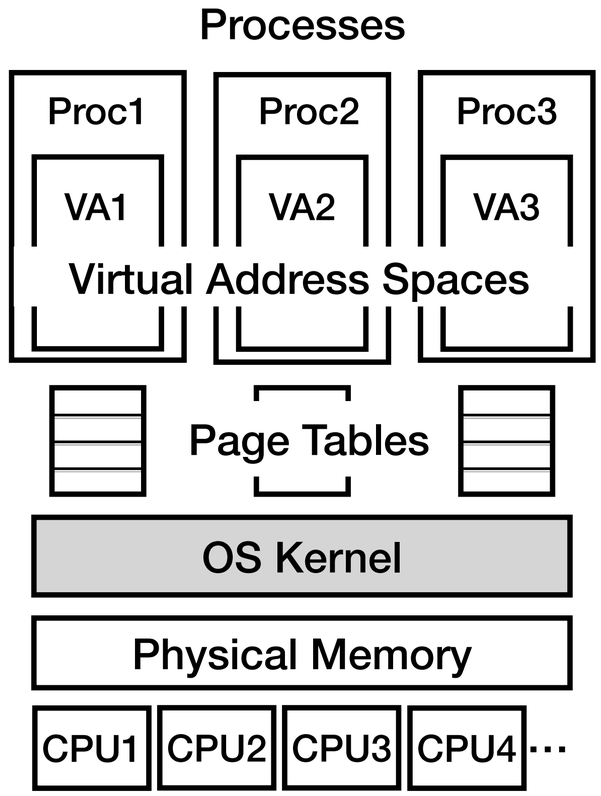
Department of Computer Science and Information Engineering
National Taiwan University

Agenda

- **Introduction of Virtualization**
- Types of Virtual Machine

Virtual v.s. Real

- Virtual Memory
  - Ex: OS kernel manages virtual address (VA) spaces
- Virtual CPU resources
  - Ex: OS kernel manages processes that share CPUs
- Virtual I/O devices
  - Ex: Linux virtual network kernel devices (TUN/TAP)
- Virtual Private Network (VPN)
  - Ex: Secure network communication over shared/public physical network

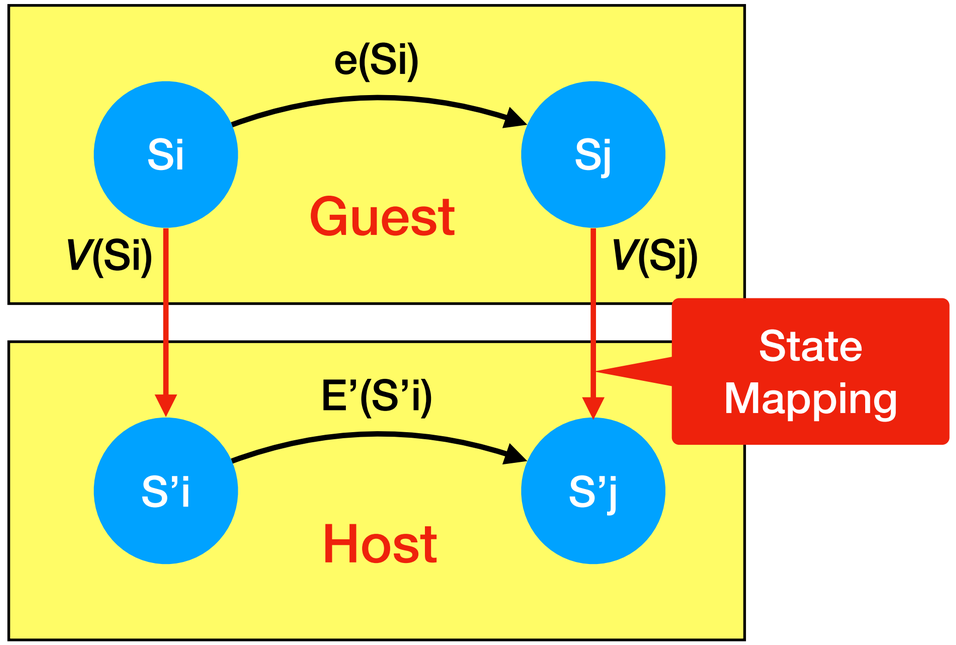


What is Virtualization?

- Map some **virtual** interface/resource to some **real** interface/resource
  - Can support more virtual copies than copies available in real
- The virtual and real interfaces can be identical or different
  - Ex: Virtualizing same architecture — ex: x86 on x86 (faster)
  - Ex: Virtualizing different architectures — ex: arm on x86 (slower but better flexibility)
- Software built for the (virtual) interface can run as intended
  - An OS Kernel that runs on real machines also runs on virtual machines

Virtualization is Isomorphism

- By Popek and Goldberg 1974:
  - Virtualization involves the construction of an isomorphism that maps a virtual ***guest*** system to a real ***host*** system

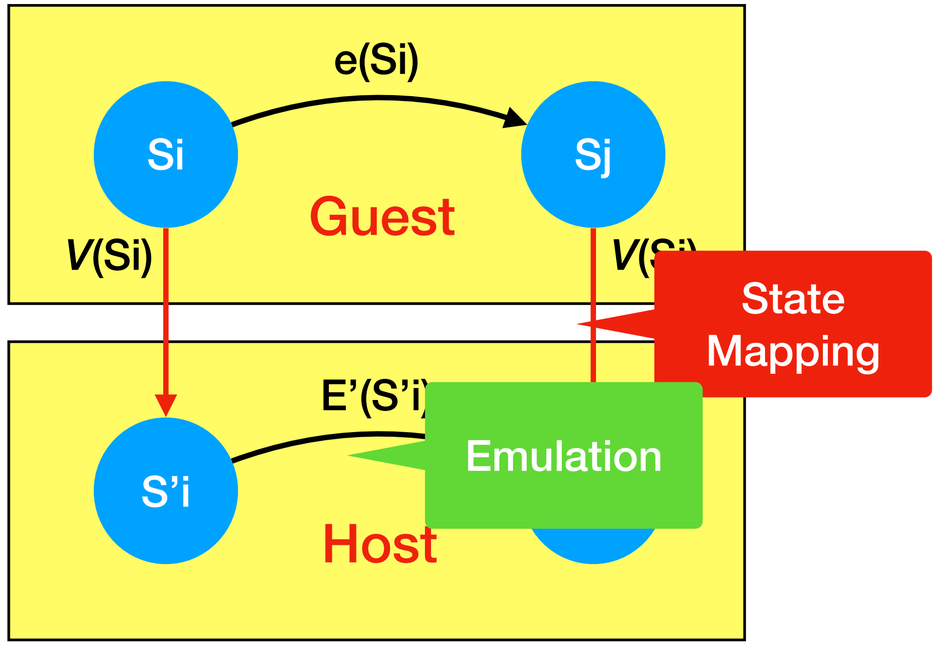


Mapping Virtual to Real

- ***Virtual Memory***
  - Read/Write virtual memory -> Read/Write physical memory
- ***Virtual CPU Resources***
  - Per-process execution context -> hardware execution context
- ***Virtual I/O Resources***
  - Traffic to the TUN/TAP interface -> traffic to the hardware ethernet interface
- ***Virtual Private Network***
  - Private network -> Public network

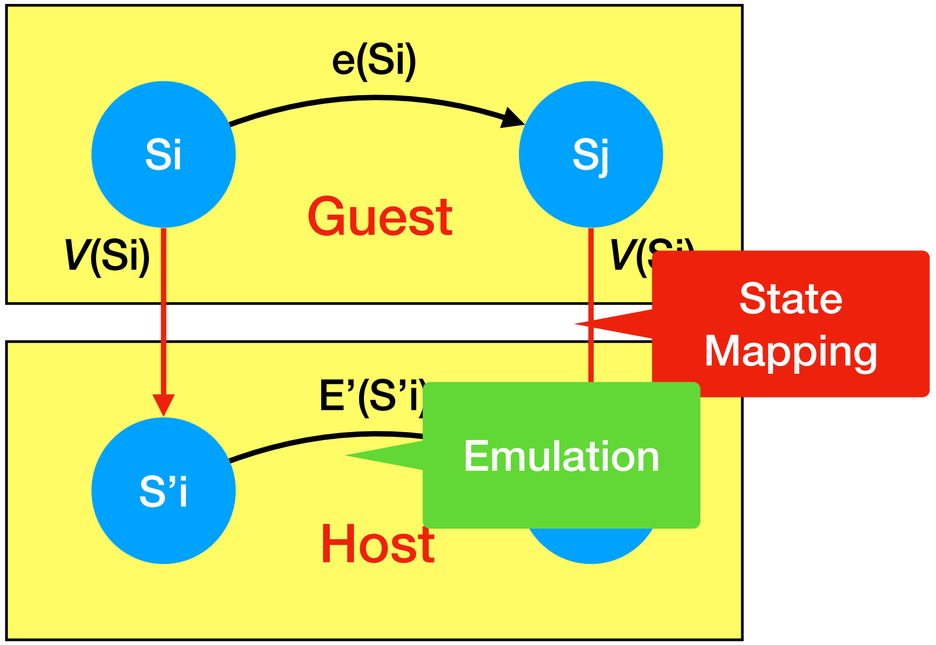
Steps of Virtualization

- Virtualization consists of two steps: ***state mapping*** and ***emulation***
- Goal is to faithfully emulate the interface and do so efficiently (fast)

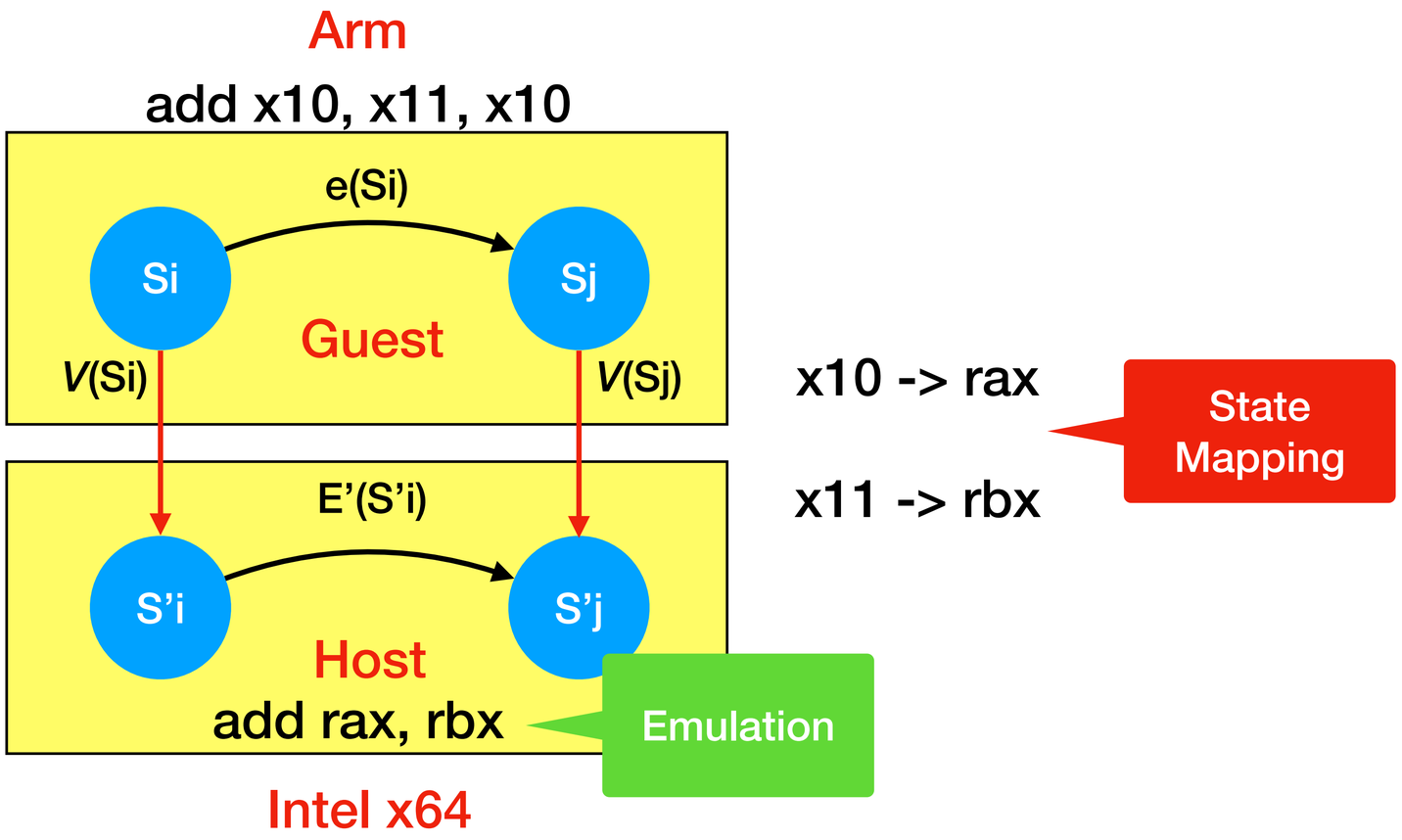


Steps of Virtualization

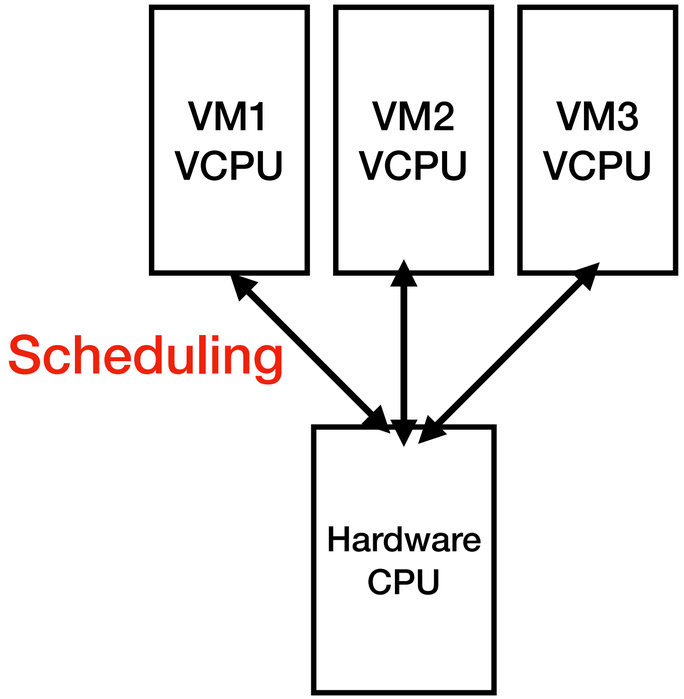
- What is emulation?
  - Think of it as a function that performs the state transition
  - It can be a no-op, why?



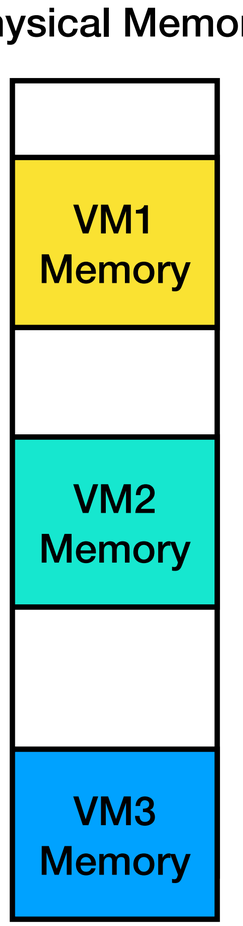
Example: Virtualizing Arm instructions on x86 Host



Example: Virtualizing VM resources



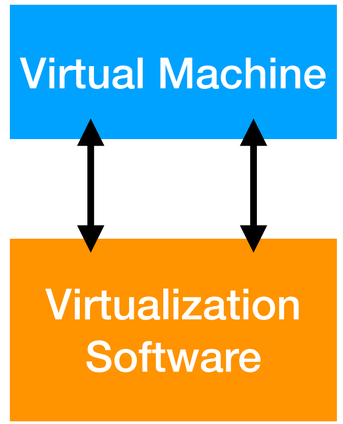
Schedule VM's virtual CPUs on hardware CPUs to virtualize CPUs



Virtualize VM memory

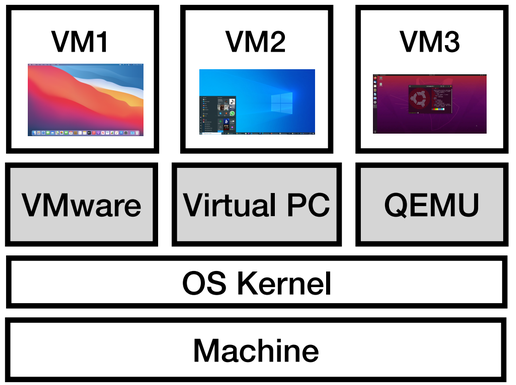
Virtualization Properties

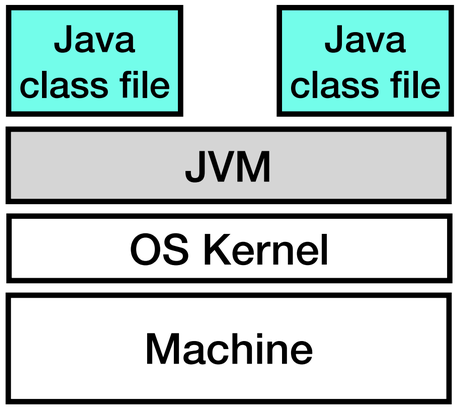
- **Isolation**
  - Fault and Data isolation
- **Encapsulation**
  - Define an interface that cleanly captures all (VM) states
- **Migration**
  - Support migrating VM states to different environment
  - Ex: ship VMs with preinstalled OS, container images with preinstalled applications
- **Interposition**
  - Transparent resource allocation, scheduling, and protection



Virtual Machines

- Add a new software layer to a real machine to support virtualization — virtualizing the desired virtual machine interface
  - VM architecture can be the same or different from the host
- Virtualize machine interface
  - VMware Fusion, Virtual PC, QEMU, KVM, Xen
- Virtualizing user programs: Java Virtual Machine (JVM)
  - Run compiled Java byte code on various host platforms





Real use case of Virtual Machines (1)

- **Improve Resource Sharing**

  - Abstract one large multiprocessor servers into multiple smaller virtual servers
  - Virtual machines serve as backbone for cloud computing platforms!
- **Security through isolation**
  - Isolate/confine compromised VMs
  - Ensure an attacker from one VM cannot access other VMs' data

Real use case of Virtual Machines (2)

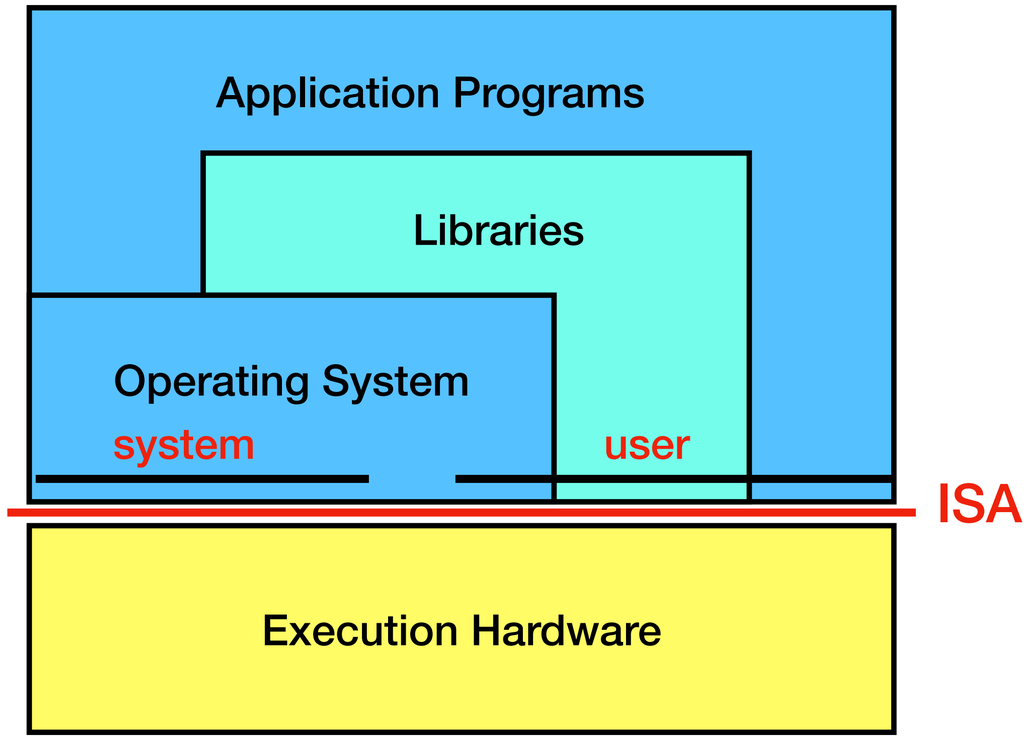
- **Flexibility & Interoperability & Portability**
  - Support heterogeneous OS (VMware Fusion, Virtual PC)
  - Support multi-architecture (Run Arm binaries on x86 machines using QEMU)
  - JAVA programs run on various OSes/architectures (Java VM)
- **Encapsulation**
  - Schedule based on the VM abstractions
  - Snapshots, backups, and restores
  - Migrate Ubuntu VMs across different machines

Agenda

- Introduction of the course
- Introduction of Virtualization
- **Types of Virtual Machines**
  - Process Virtual Machines
  - System Virtual Machines

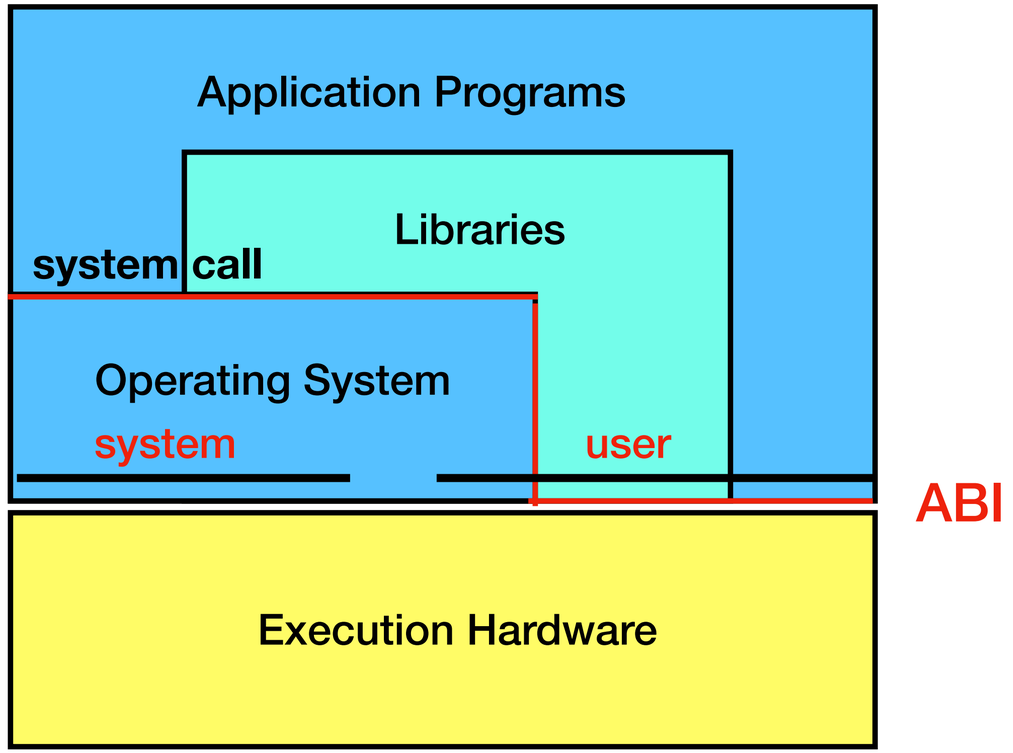
Computer Systems Architecture (1)

- Instruction Set Architecture (ISA)
  - The division between SW/HW
  - An ISA specifies the registers and instructions, and how they interact
  - Includes ***user*** and ***system*** ISA
    - Examples?
- Different CPU architectures (Arm, x86, MIPS, RISC-V) define different ISAs



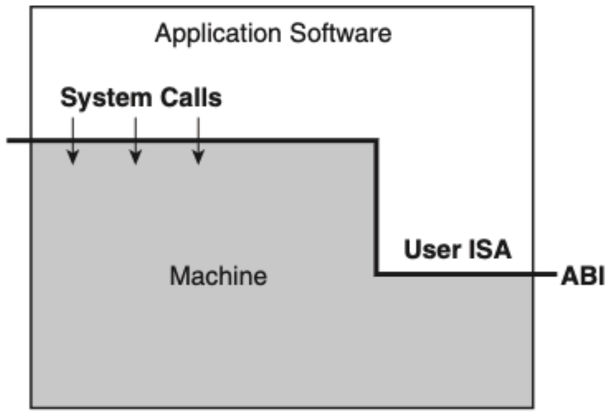
Computer Systems Architecture (2)

- Application Binary Interface (ABI)
  - Defines how compiled code interacts with the system and other compiled code
    - The calling convention: e.g., how to pass arguments system calls (via the stack or registers)
    - Sizes, layouts, and alignments of basic data types that the processor can directly access
  - Most typically include the user space ISA and system call interface
    - The latter defines how system calls are implemented, how?
- Application Programming Interface (API) v.s. ABI?

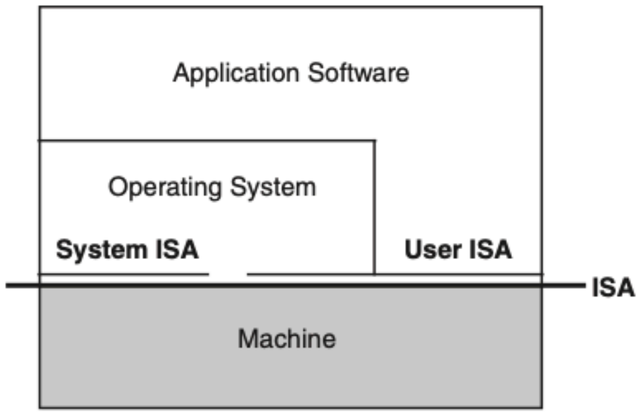


Two Types of Virtual Machines

- Just as machines, there are also two types of virtual machines!
  - ***Process-level*** and ***System-level***
- It depends on what interface is virtualized



Process VM



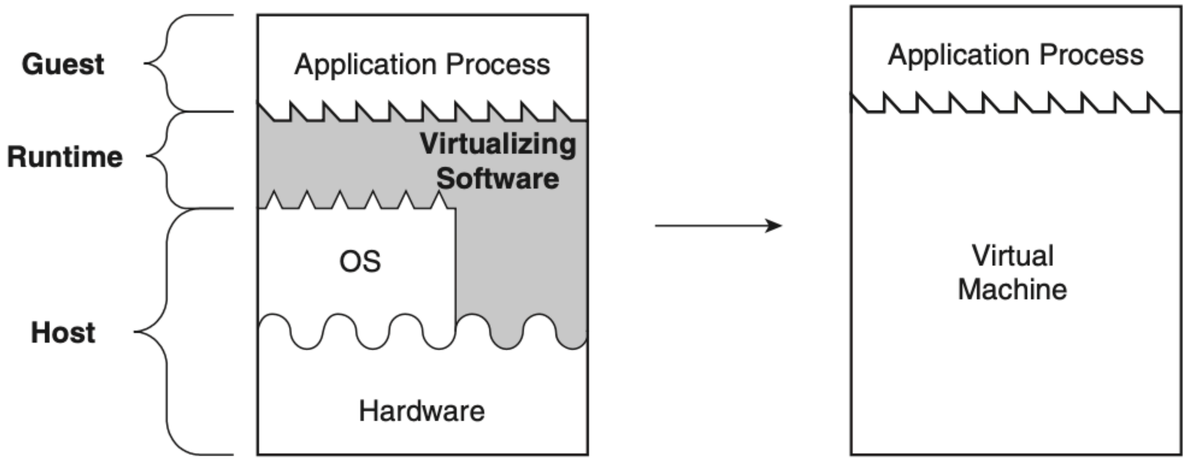
System VM

Process Virtual Machines Overview (1)

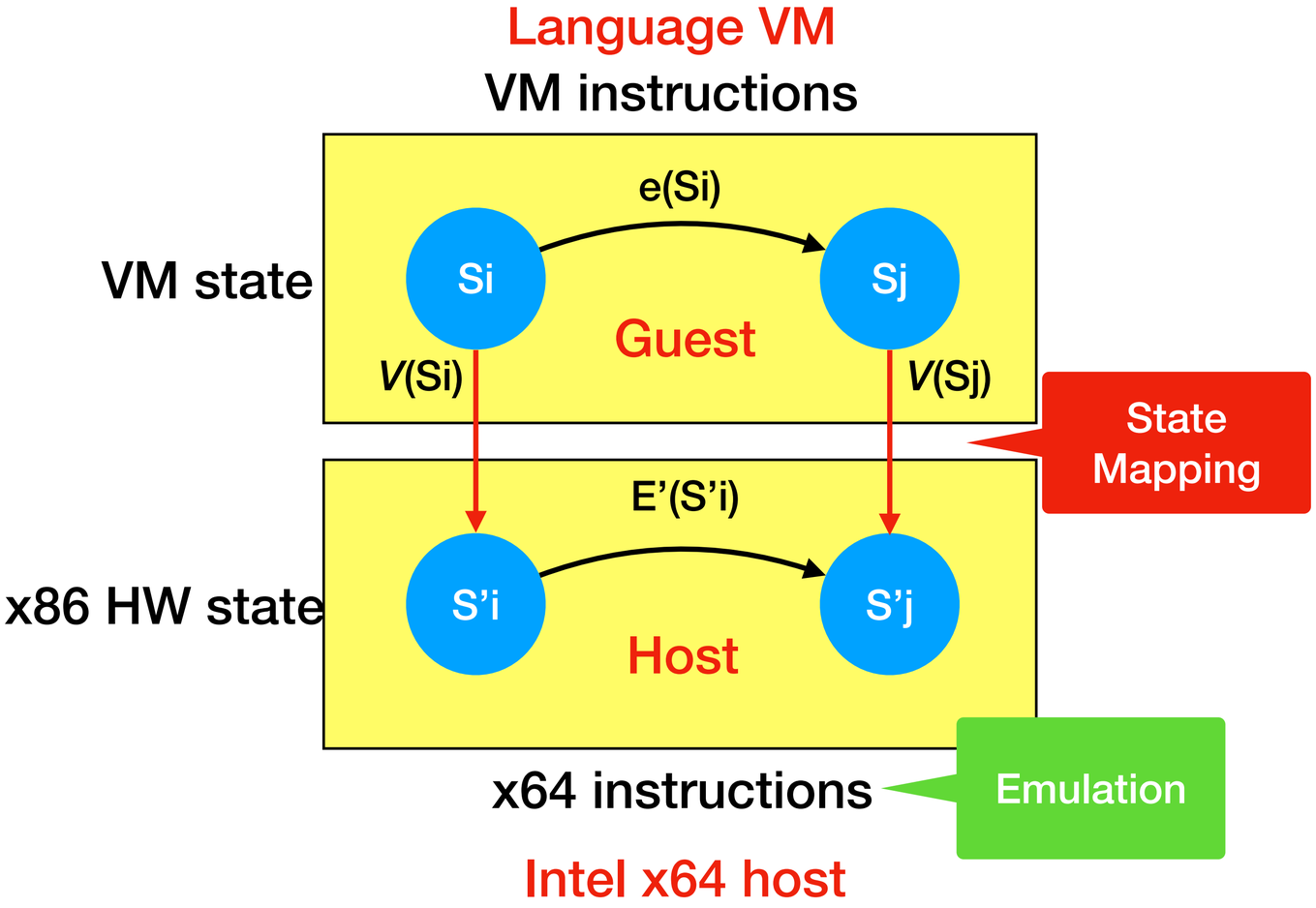
- Support an individual process
- Placed at the ABI interface
  - Virtualize user-level instructions and OS system calls
  - Leverage host ABI to virtualize
- Refer to the virtualizing software for process VMs as the ***runtime***
- Examples: Apple's Rosetta, Java VM, Microsoft .NET

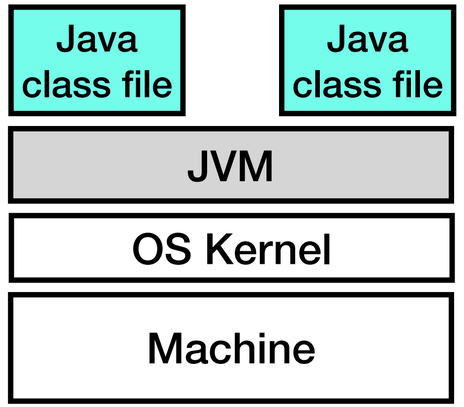
Process Virtual Machines Overview (2)

- Process VMs usually run in a different architecture from the host
  - Run the same ISA for optimization
- Intermingle with processes on the host — scheduled by the host OS



Process Virtual Machines Example: Language runtime



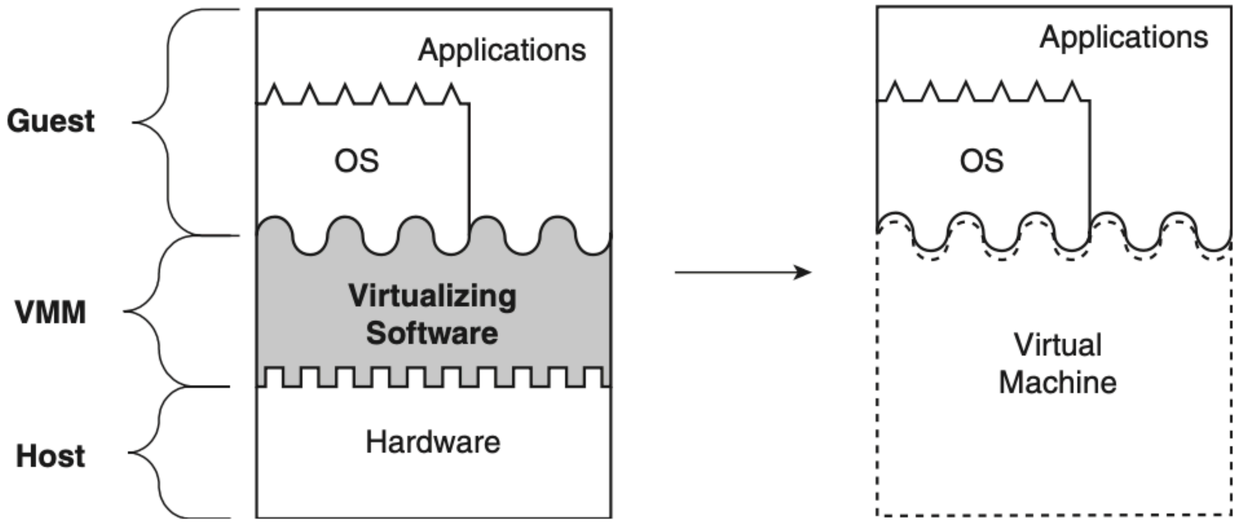


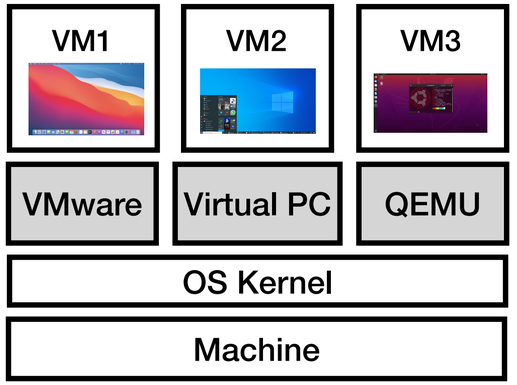
System Virtual Machines Overview (1)

- Provide a complete system environment that can support a full OS kernel
  - Provide virtual HW: CPU, Memory, I/O devices (network/disk/graphics)
- Placed at the ISA interface
- The virtualizing software usually virtualizes the same ISA as the host ISA
  - That is not always the case — QEMU is an example
- Examples: Linux/KVM, VMware, etc.

System Virtual Machines Overview (2)

- Refer the virtualizing software for system VMs as the ***virtual machine monitor (VMM)*** or ***hypervisor***
  - Note: the term will be use interchangeably throughout the course





Case Study: QEMU

- A generic and open-source machine emulator
- Supports both process and system VMs
  - Includes a dynamic binary translator (DBT) that supports cross-ISA virtualization
    - The guest runs in user space when using DBT
  - Integrates with a hypervisor (KVM and Xen)
  - https://www.qemu.org/docs/master/about/emulation.html
- Process VM
  - Supports user mode virtualization (ex: qemu-i386): for a specific OS and CPU architecture
- System VM
  - Supports full system virtualization based on either DBT or native execution via a hypervisor
  - In the latter case, the DBT is disabled; VMs run at the ISA level

Agenda

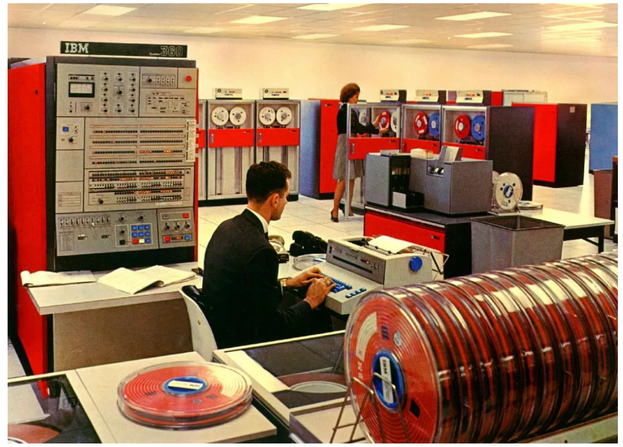
- Introduction of the course
- Introduction of Virtualization
- **Types of Virtual Machines**
  - Process Virtual Machines
  - **System Virtual Machines**

In the 1960-70s

- Mainframe computer systems were very large and expensive in 1960-70s
  - IBM System/360: the first mainframe computer that supports wide-range of applications — was designed to run batch jobs

In the 1960-70s

- IBM's CP/CMS (Control Program/Cambridge Monitor System) or CP-67 was released to the public in 1968
  - IBM needs a robust time-sharing solution for a multi-user scenario
    - Time-share a single-user OS on the CPU
- IBM VM/370 reimplements CP/CMS (released in 1972)
  - First hypervisor to support multiple OSes

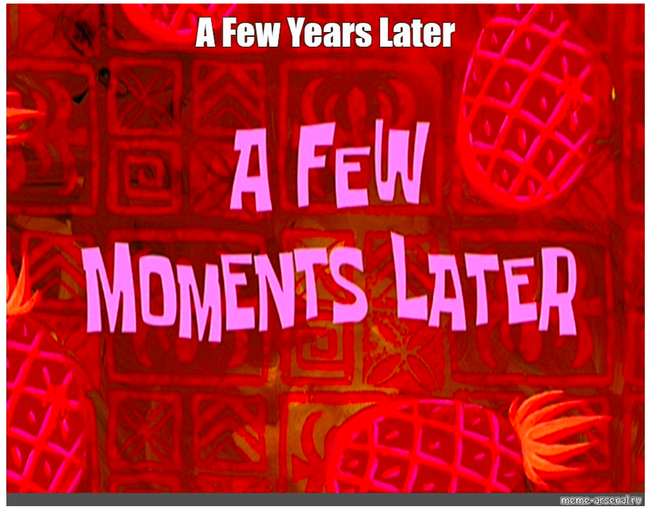


Why System VMs in 60-70s?

- Better utilize resources of an expensive machine — very few machines are available
- Support multiple OSes for different workloads
- Run various OS versions (new and stable) to ensure app compatibility
- Was a hot research topic — dedicated conferences for system VMs

VMs in the 80s

- Research interests faded out
  - Machines become much cheaper; shift to desktop computers
- More advanced OSes emerges
  - UNIX running on workstations supports multi-user
  - Do not have to run single-user VMs

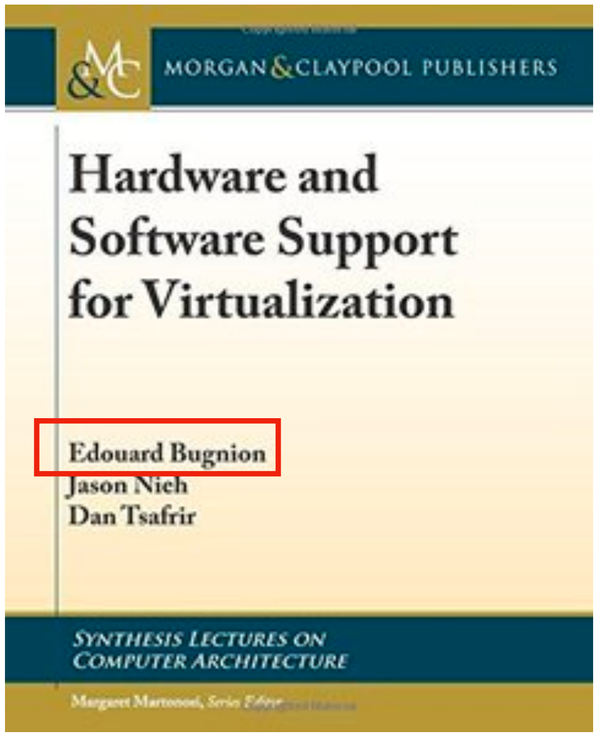


VMs in the 90s

- Fast advancement in hardware development (e.g. multiprocessor support)
  - Commodity OSes back then did not scale (guess why?) and cannot isolate faults
  - OS development cannot catch up — customized OSes are incompatible
- ***Disco*** published in SOSP 1997
  - Rely on the simple Disco hypervisor (with 13K LOC) to manage resources of multiprocessor hardware
  - Support commodity OSes in virtual machines
  - The predecessor of VMware ESXi
- ***Xen*** published in SOSP 2003 to support better performance in VMs
- More about Disco and Xen will be discussed in the class

Fun Fact

- The authors of the Disco paper co-founded VMware in 1998
  - Disco's first author is also the first author of our text book!



Revival of System Virtual Machines

- Deploy hypervisors in cloud server farms to run system VMs
  - Provide an efficient and secure way to partition software on shared hardware platforms

VMs in 2000s

- Growing demand for efficient hypervisor implementation
- CPU architectures include hardware support for virtualization
  - Intel released VT-x in 2005 on Pentium 4 processors
  - AMD released AMD-V on Orleans and Windsor processors
- More hypervisors came out
  - KVM was merged into the Linux kernel mainline (v2.6.20) in 2005

VMs in 2010s and beyond (1)

- VMs are used everywhere — more functionality and better performance
- VMs enable efficient resource sharing, but what about security?

VMs in 2010s and beyond

- Growing concerns about running VMs on shared hardware
  - An attacker that owns the hypervisor can wiretap your VM's data
  - Hardware-based solutions:
    - Have you heard of Intel SGX/TDX, AMD SEV, or Arm CCA?

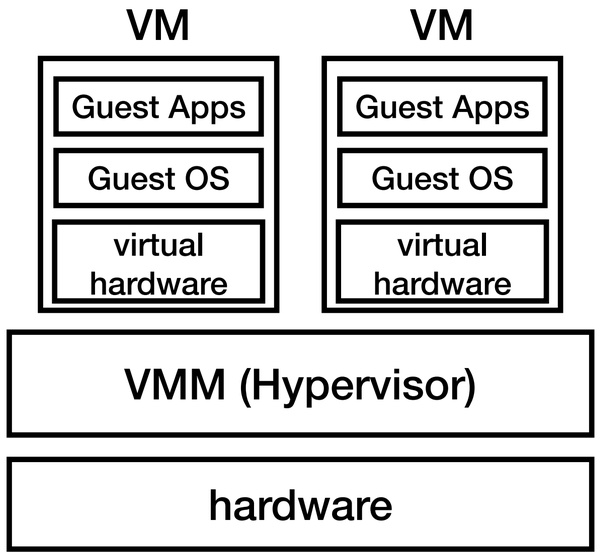
Properties of Hypervisors

- From “Formal Requirements for Virtualizable Third Generation Architectures” — VMM Properties by Popek and Goldberg 1974:
  - Equivalence
  - Performance
  - Complete Control

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.

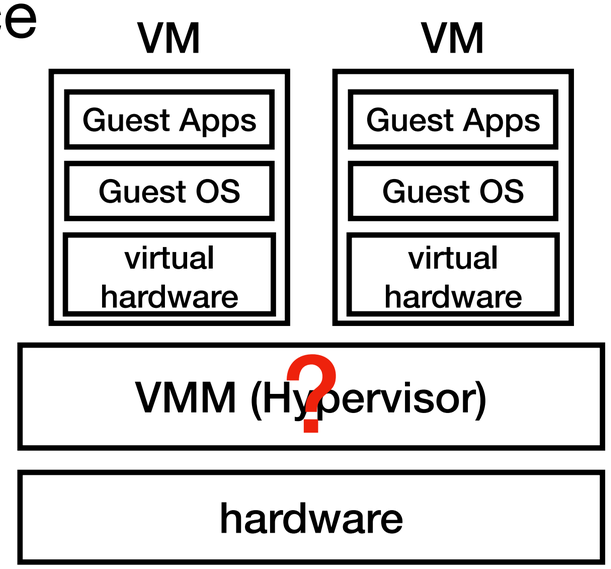
Designing VMMs/Hypervisors

- **Equivalence**:
  - Provide multiple VM abstractions (virtual HW interface)
  - Programs running in VMs should behave the same as on real hardware
- **Performance**:
  - Multiplex shared hardware resources for different VMs
- **Complete Control**
  - Fully control the HW to isolate VMs for security



Designing VMMs/Hypervisors: what to virtualize?

- Hypervisors should provide the machine/ISA interface
  - CPU Virtualization
  - Memory Virtualization
  - I/O Virtualization



- To do so efficiently is challenging:
  - Many SW/HW techniques — will be covered later in the course

Types of Virtualization for System VMs

- ***Full Virtualization***
  - Support unmodified OS kernels in VMs
  - Pioneered in 1966 with IBM CP40/67
- ***Para Virtualization***
  - Support modified OS kernels in VMs
  - Modified guest OS for performance

State of Modern Hypervisors/VMMs

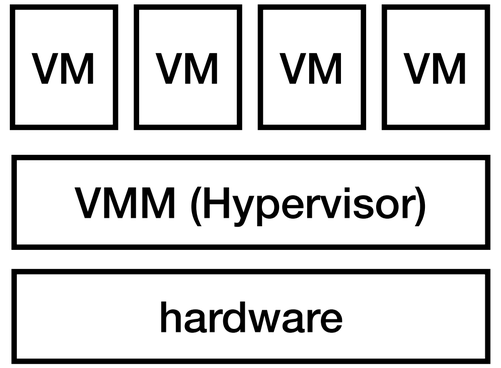
- Popular VMMs include Linux/KVM, VMware Workstation/ESXi/Fusion, Microsoft's Hyper-V, Xen, Virtual PC, VirtualBox, Parallels
- Modern VMMs support full virtualization with hardware assistance
- Use para virtualization as techniques for performance optimization
  - Example: Virtio framework for para virtual I/O virtualization

VMM Classification: Type 1 and 2

- Type 1 hypervisors:
  - Also called bare-metal hypervisors
  - Example: IBM VM/370, Disco, VMware ESXi, Xen, Hyper-V

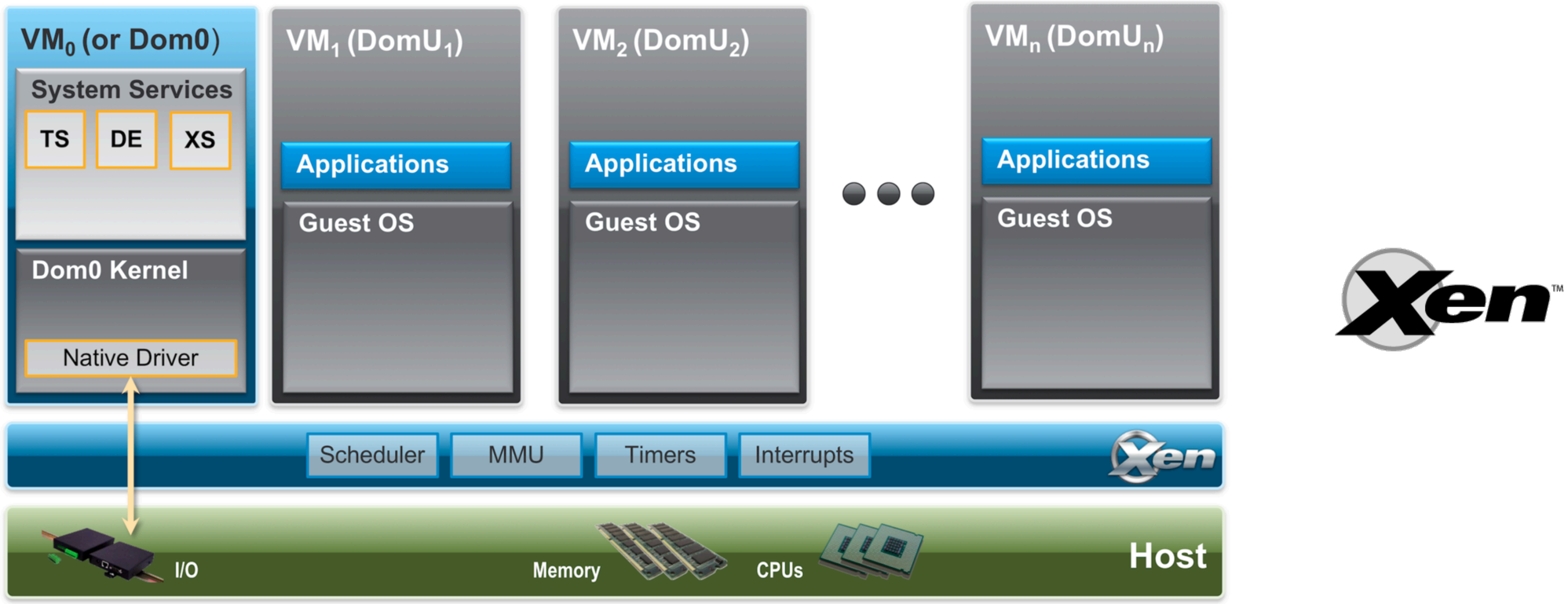
- Type 2 hypervisors:
  - Also called hosted hypervisors
  - Example: KVM, VMware Workstation/Fusion, VirtualBox

Type 1 Hypervisors (1)

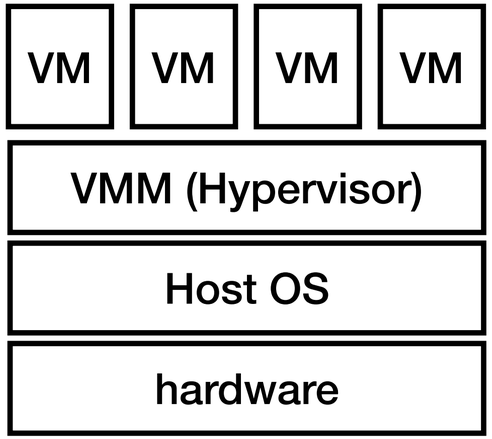


- The hypervisor directly runs on the physical hardware
  - Provide standalone support for virtualization features:
    - Machine bootstrapping, scheduling, and resource allocation
  - The hypervisor codebase is usually smaller than an OS
- Could rely on a privileged VM and its OS for I/O virtualization

Type 1 Hypervisors (2)

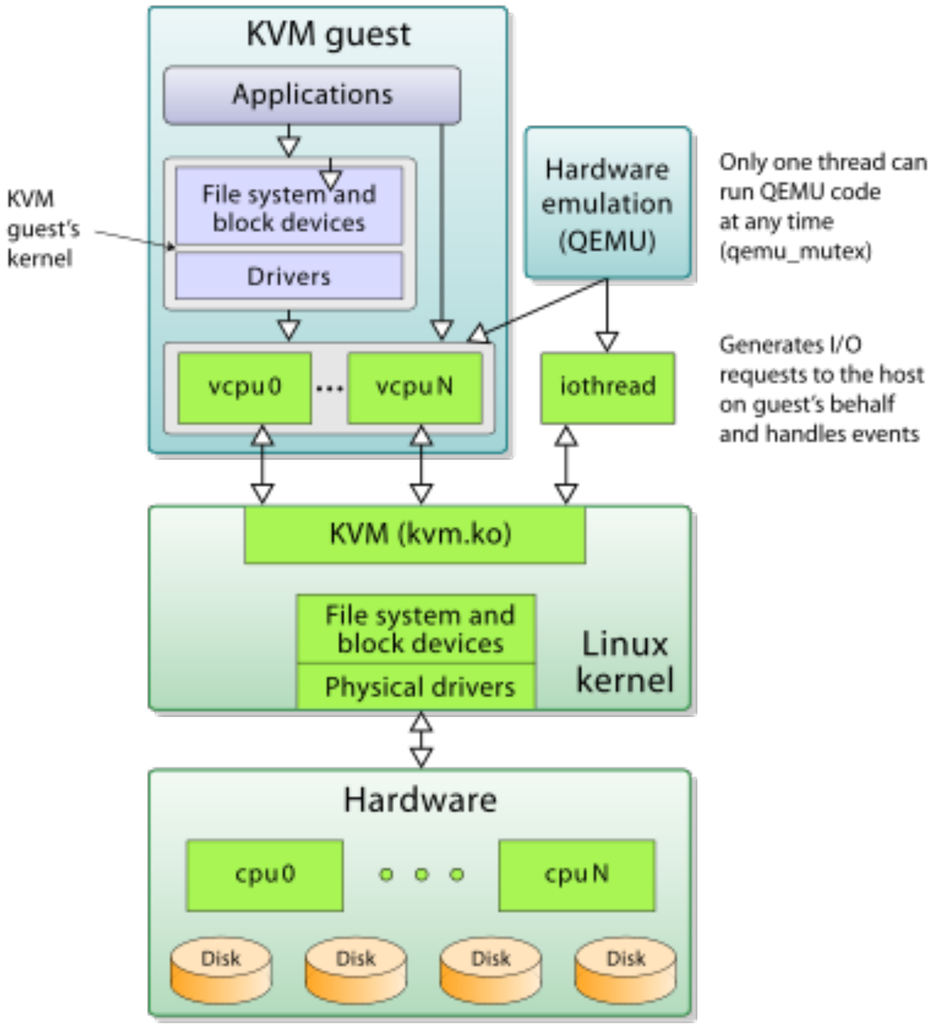


Type 2 Hypervisors (1)



- Built on top of (or integrates with) an existing OS (*host OS*)
  - Why? Reuse functionality from the integrated host OS directly for machine bootstrapping, scheduling, resource allocation, device I/Os
- Larger hypervisor codebase compared to type-1

Type 2 Hypervisors (2)



Agenda

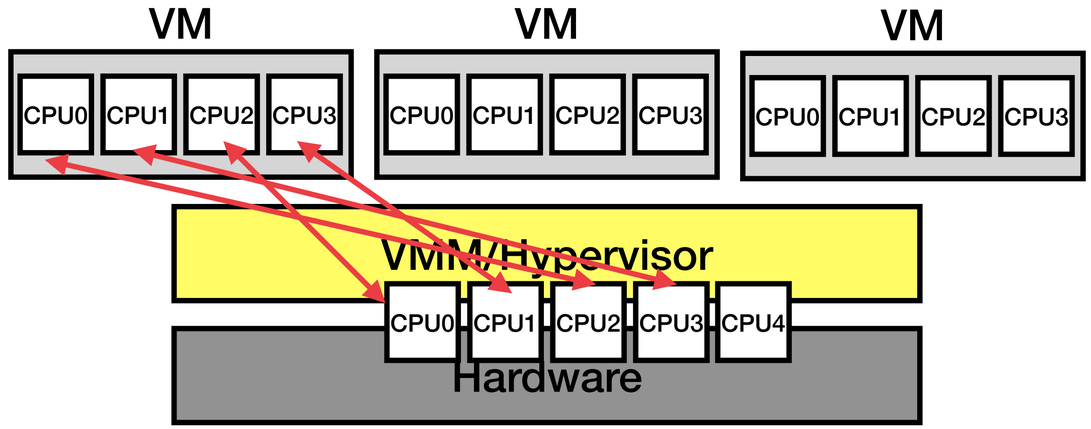
- Introduction of the course
- Introduction of Virtualization
- **Types of Virtual Machines**
  - Process Virtual Machines
  - **System Virtual Machines**
    - **CPU Virtualization**

CPU Virtualization

- Goal: guest systems run in the VM to manage CPUs
- The hypervisor must:
  - Virtualizes CPU resources; a VM has multiple CPUs
  - Virtualizes the ISA; supports execution of system and user instructions, and a CPU has multiple different registers

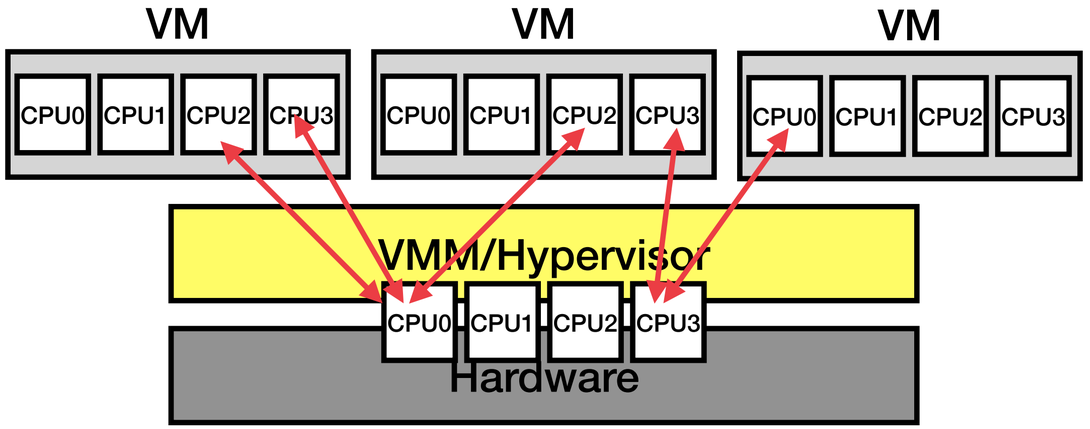
Virtualizing CPU Resources (1)

- Goal: map virtual CPUs (vCPUs) to physical CPUs (pCPUs)
- Simple solution: statically assign a VCPU to a specific PCPU
  - Any issues?



Virtualizing CPU Resources (2)

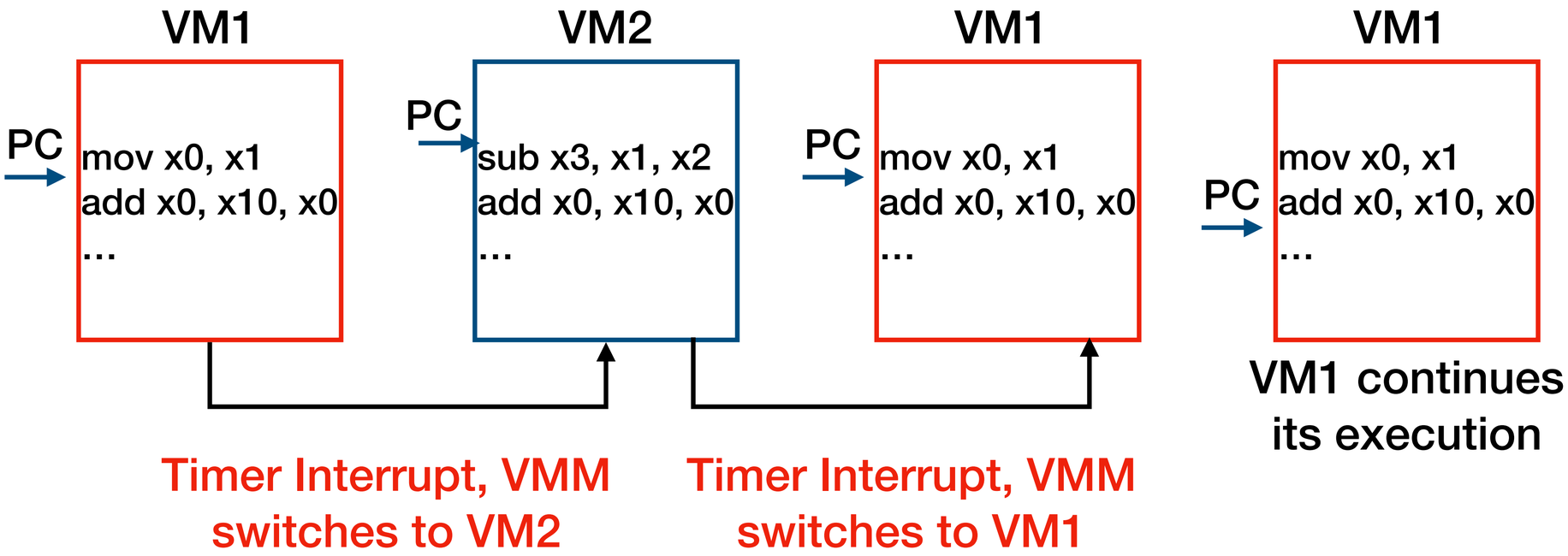
- Goal: map virtual CPUs (vCPUs) to physical CPUs (pCPUs)
- Better solution: time-sharing pCPUs between different vCPUs
- Give the illusion to VMs that they exclusively own the physical CPUs



Time-sharing CPUs (1)

- Technique to virtualize a few CPU resources to many more
  - OS schedules multiple processes on a set of real CPUs — each process has the illusion that it exclusively owns the CPU
  - Similarly, the VMM time-shares real CPUs among VM's **virtual CPUs** — each multiprocessor VM thought it owned the CPUs

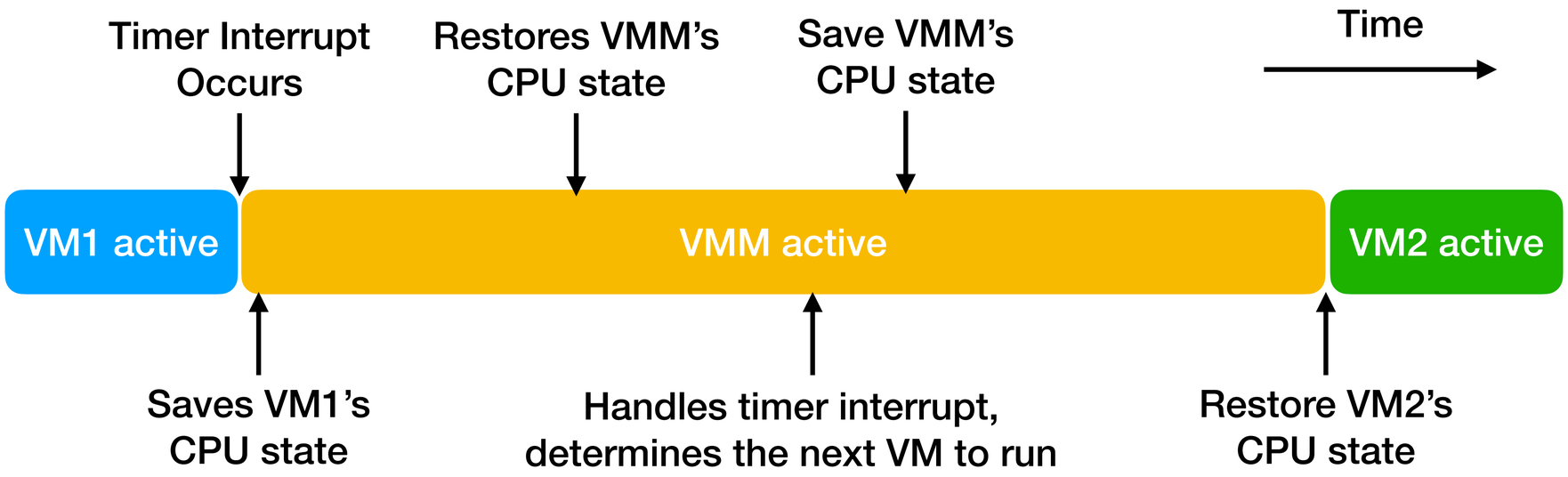
Time-sharing CPUs (2)



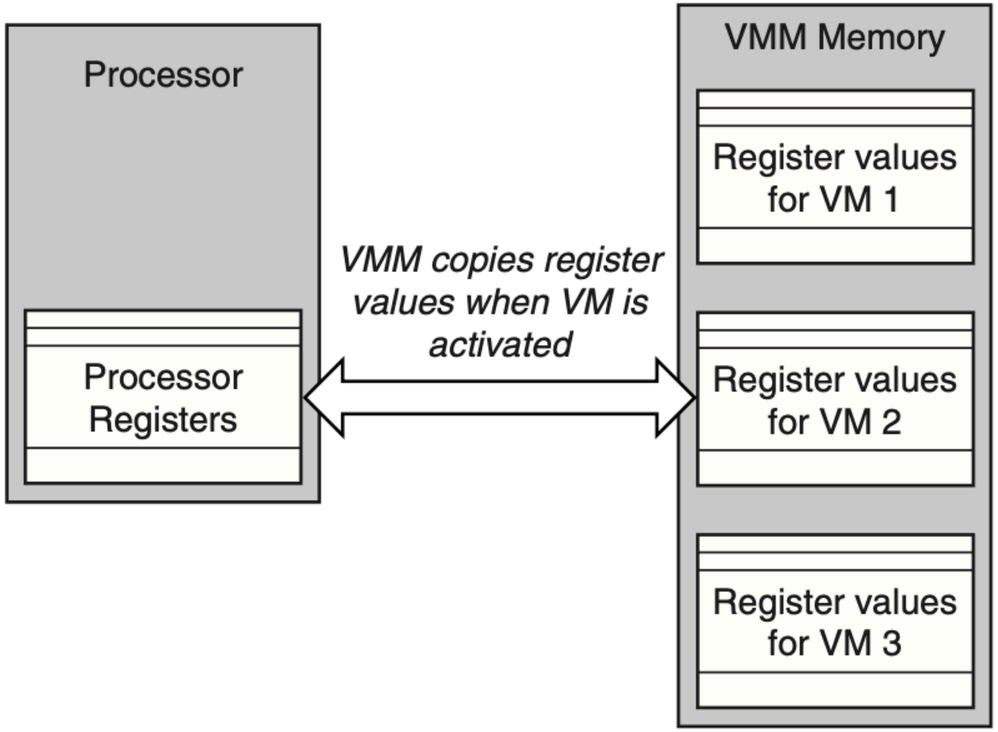
Example: a hypervisor context switches two VMs on CPU0

Time-sharing CPUs (3)

- Example: a hypervisor schedules (time-shares) two VMs on a single CPU
  - For a multiprocessor system, a VMM/hypervisor does this for all CPUs



Maintaining virtual CPU state



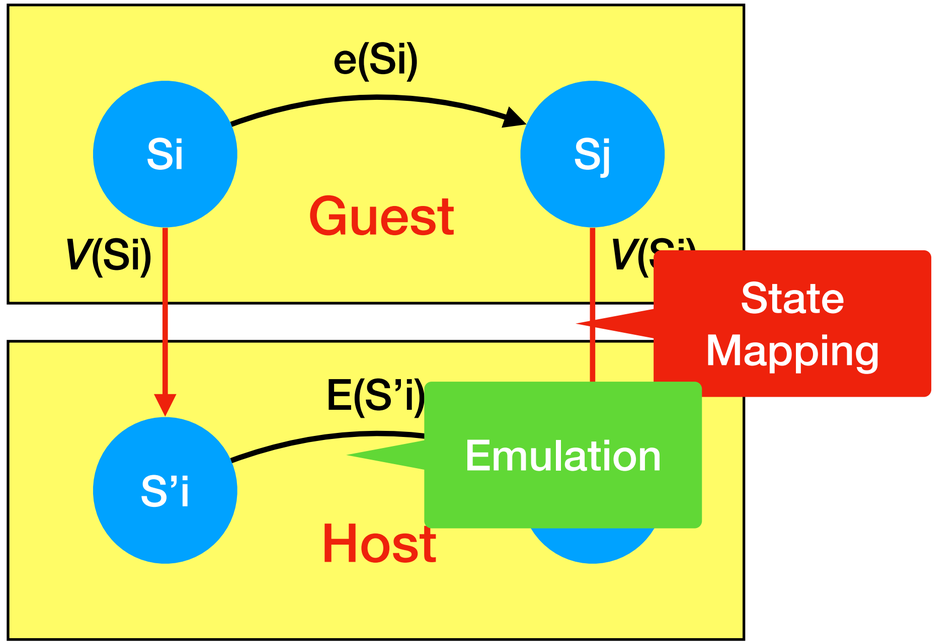

```
Copy register state from
VMM memory

Load program counter to
point to VM program and
start execution
                •
                •
                •
mov reg A -> reg B
                •
                •
                •
Copy register state from
processor back to system
memory
```

```
//for ia32
struct cpu_reg_state {
    u32 eax;
    u32 ebx;
    u32 ecx;
    u32 edx;
    ....
    u32 eflags;
    u32 eip;
    u32 sp;
}
```

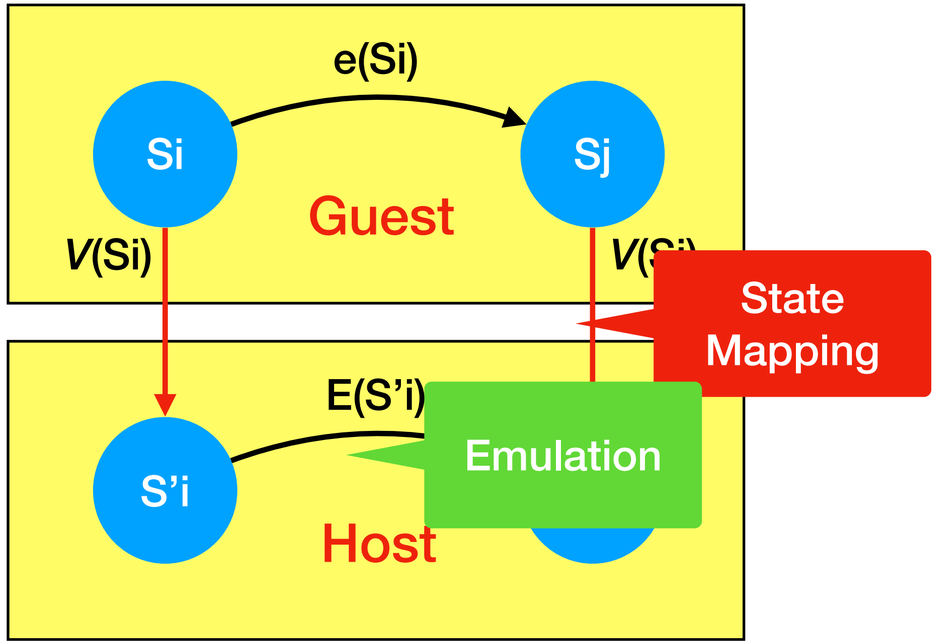
Virtualizing ISA (1)

- Guest ISA != host ISA:
  - The guest and host instructions/registers do not match
  - Guest programs cannot execute directly on the host hardware
  - Translate guest instructions (binary translation) or interpretation



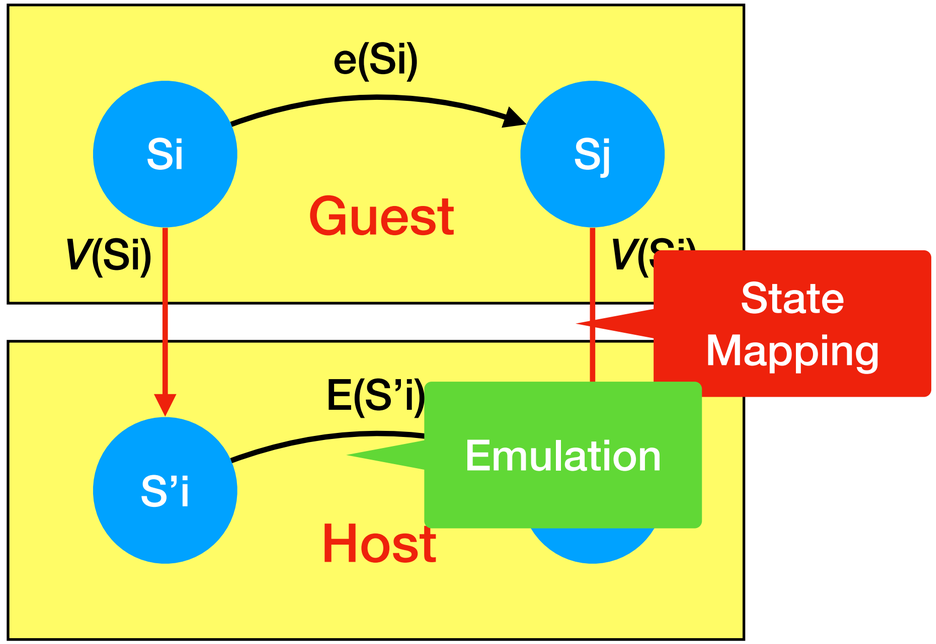
Virtualizing ISA (2)

- Guest ISA = host ISA:
- Intuition: native guest execution — this might not work; why?

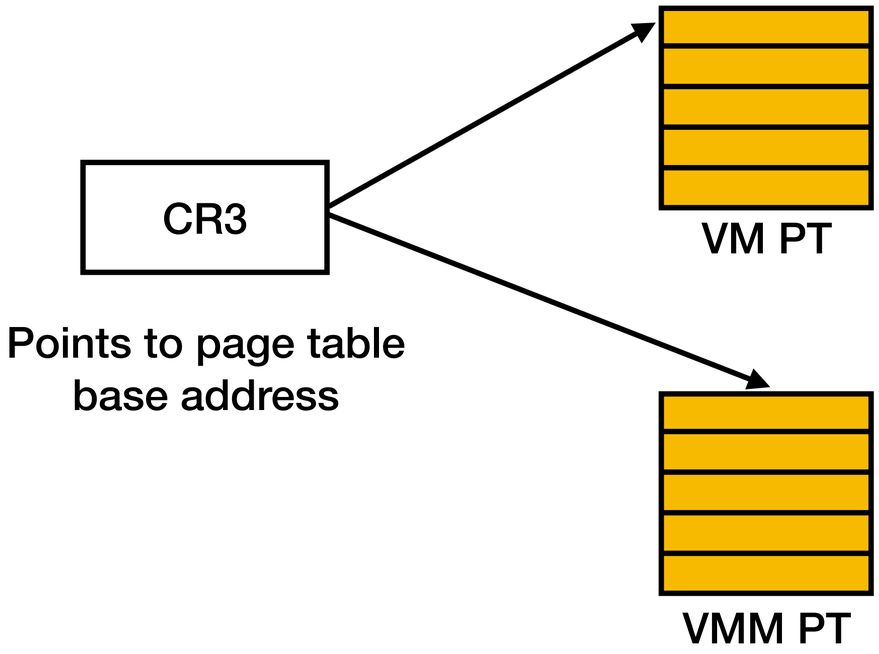


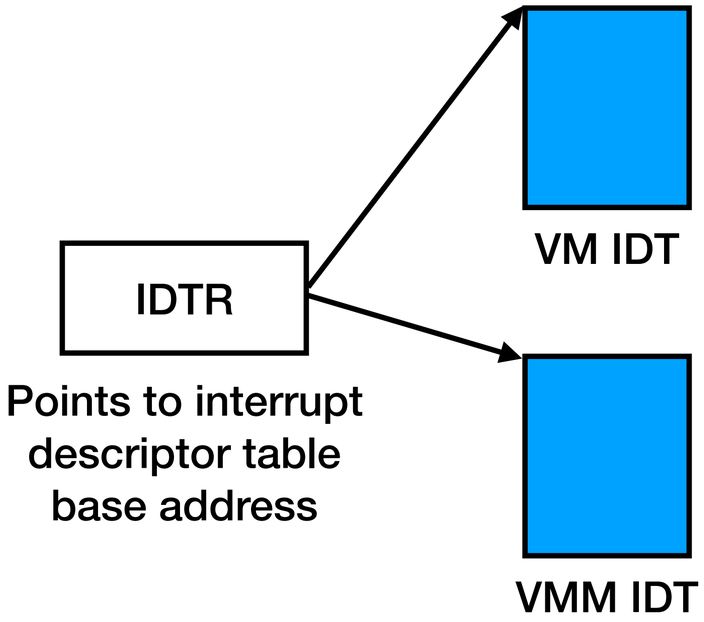
Virtualizing ISA (2)

- Guest ISA = host ISA:
  - Support user ISA is easy; just run it natively
  - Support system ISA is complex; why?



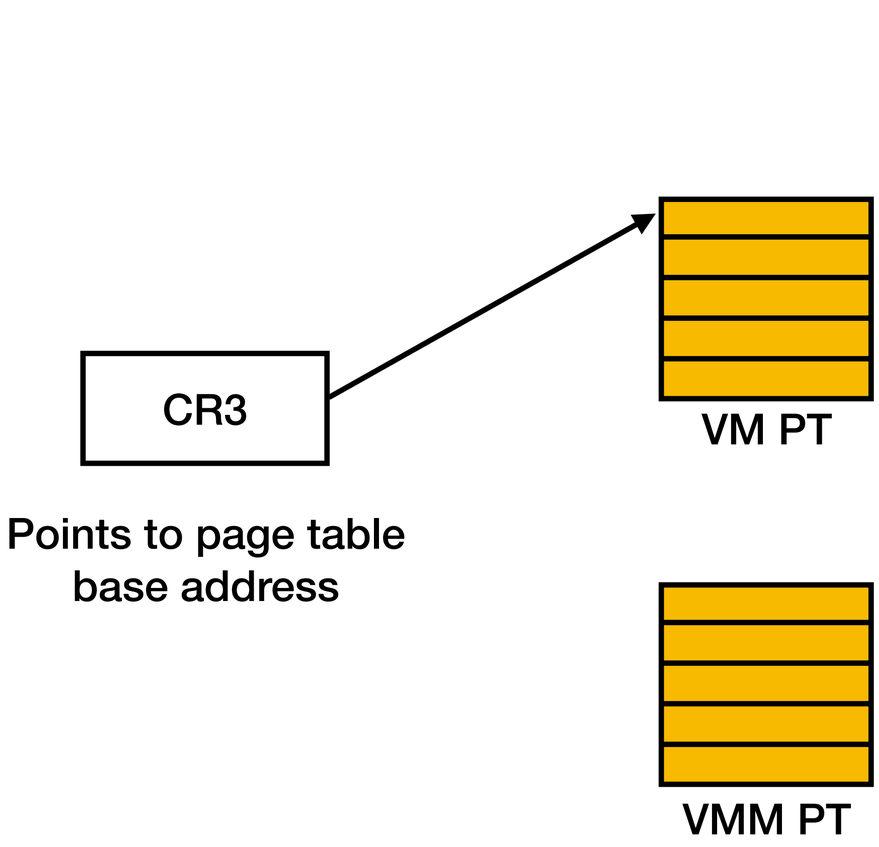
Multiplexing CPU registers between VM/VMM (1)

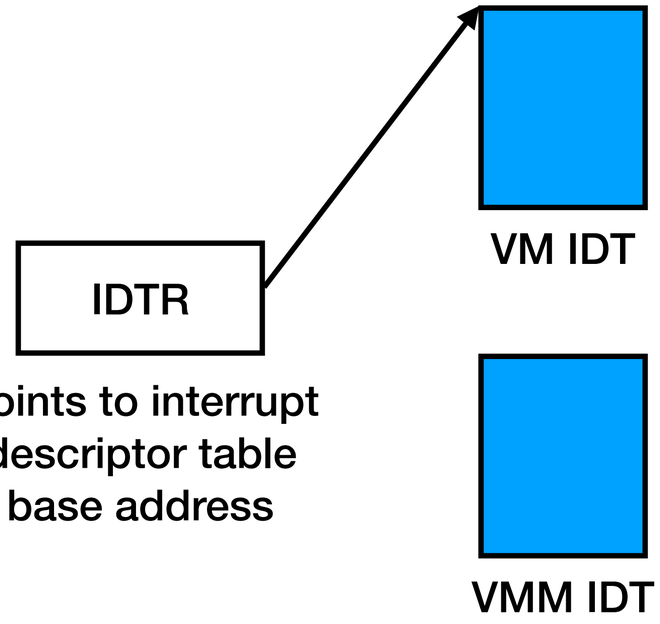




What if the VM can change CR3 or IDTR on the hardware?

Multiplexing CPU registers between VM/VMM (2)

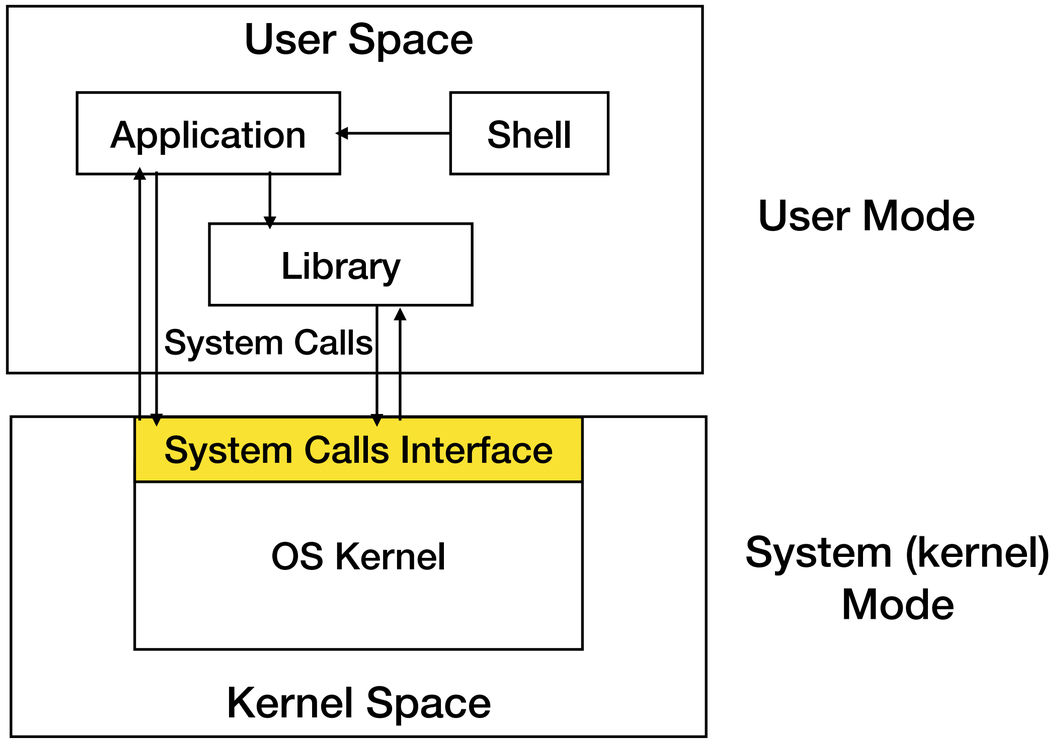




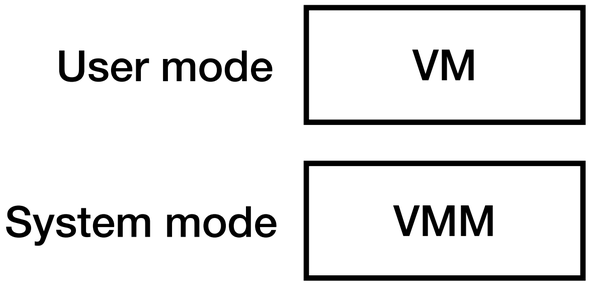
The VM will take over the hardware and never switch back to VMM — why?

Review: User and System (kernel) mode

- User/kernel split: a CPU can execute in the **system (kernel),** and **user mode**
  - System (kernel) mode:
    - Privileged, where the OS kernel runs
    - Where the privileged instructions (defined in the system ISA) can execute
  - User mode:
    - Unprivileged, where the applications run
      - Does the OS run in user mode?
    - Where the user instructions (defined in the user ISA) can execute



Virtualizing ISA: Deprivileging VMs



- Goal: execute guest code on the hardware
  - No (binary) translation or interpretation!
- The VM (essentially the guest OS) should not be allowed to change hardware resources that could affect the VMM or other VMs
- Intuition: run VMM in system mode; VM in user mode

ISA Virtualizability

- Defined by Popek and Goldberg in their 1974's paper about conditions that an ISA can efficiently support virtual machines
  - Made a few assumptions about machines: a CPU includes system/user modes, system ISA, etc.
  - Applicable to modern day CPUs

We will get back to this next week