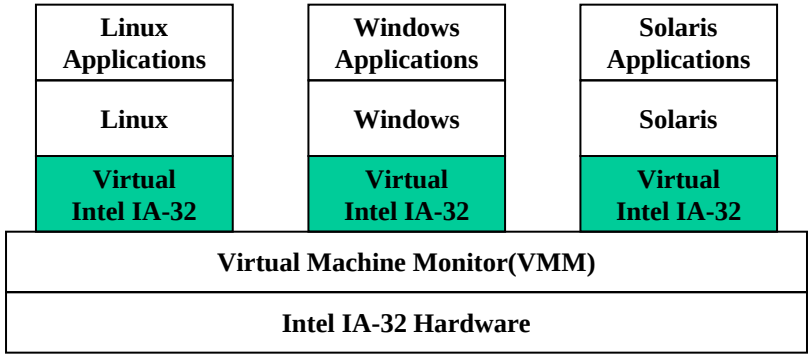
System Virtual Machines (1)

Contents

- Key Concepts of system VMs
- Resource virtualization – processors
- Resource virtualization – memory

System Virtual Machines

- A system VM environment can support multiple system images simultaneously, each running its own OS and application programs
- Real resources of the host platform are shared among the guest VM with the virtual machine monitor (VMM)
  - The VMM manages the allocation and access of hardware resources



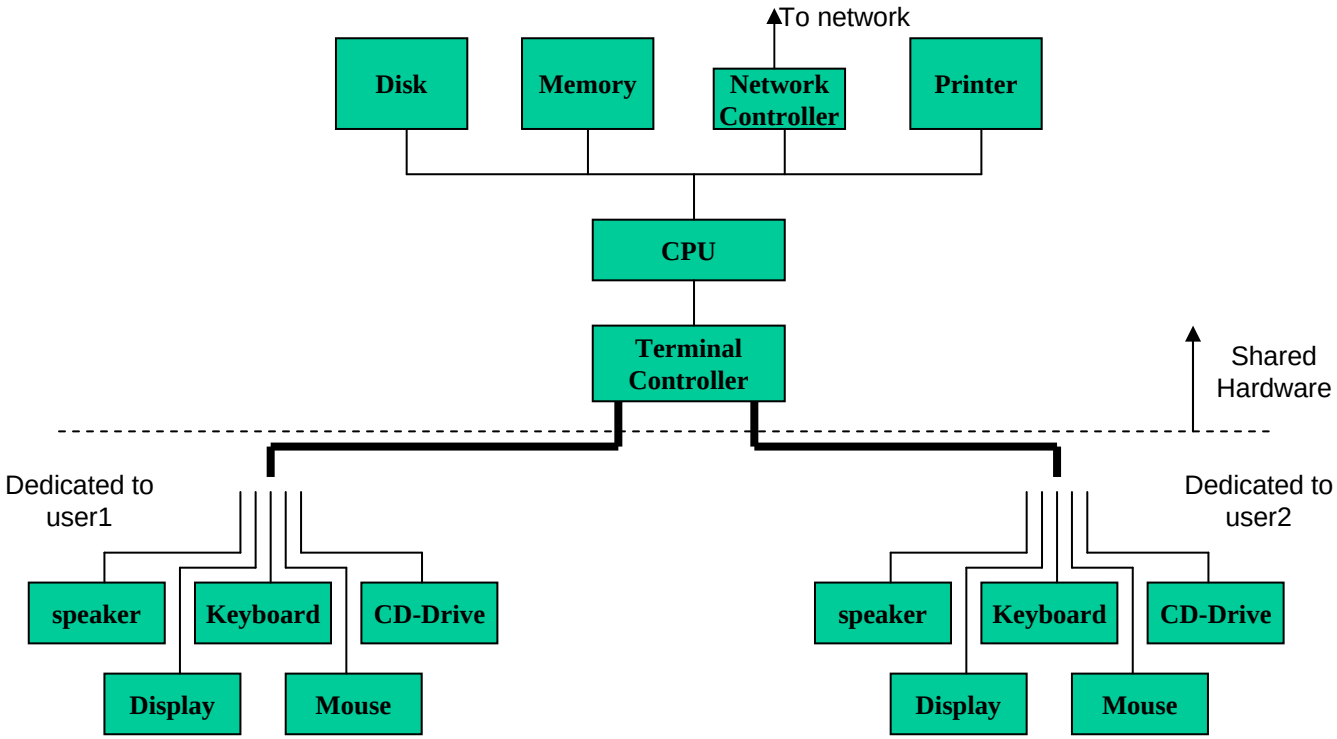
Advantage of System VM

- Implementing multiprogramming
- Multiple single-application virtual machines
- Multiple secure environment
- Managed application environment
- Mixed-OS environment
- Legacy application
- Multiplatform application development
- New system transition
- System software development
- Operating system training
- Help desk support
- Operating system instrumentation
- Even monitoring
- System encapsulation

Outward Appearance

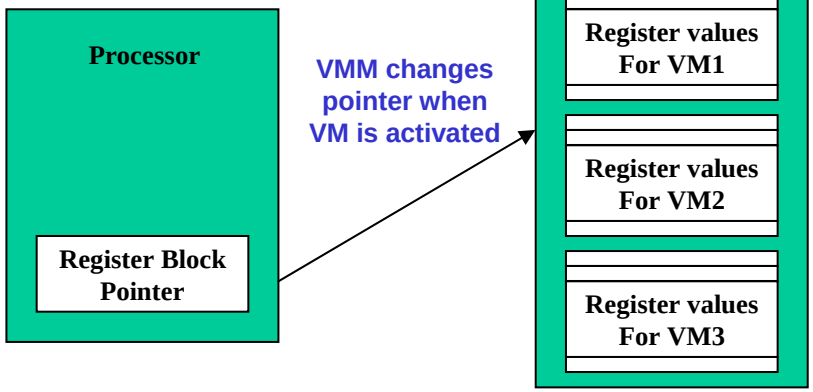
- Outward appearance:
  - Illusion of multiple machines
- Two ways to do so:
  - Replication:
    - Replicate a subset of the hardware resources and share rest of H/W
  - Switch:
    - Use a H/W switch or enter a special key sequence on the key board to switch a subset of the hardware and share the rest of the hardware

Outward Appearance Model



State Management (e.g., register copy)

Indirection



Load register block pointer
To point to VM's registers
In VMM memory

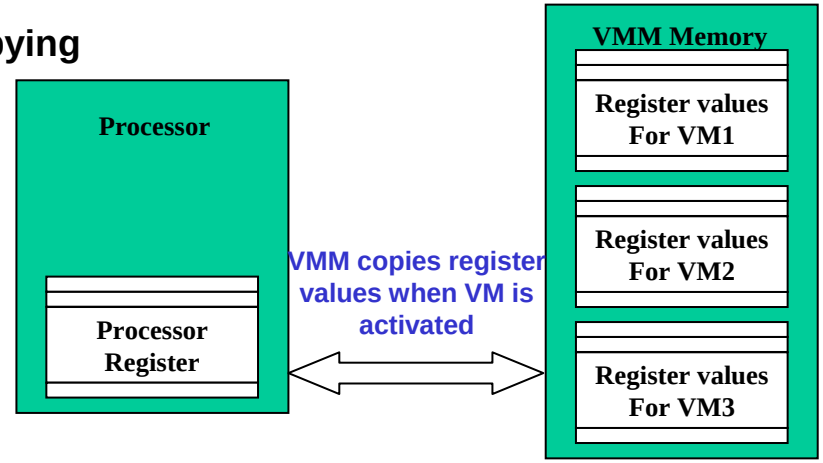
Load program counter to
Point to VM program and
Start execution

.
.

Load temp <- reg_pointer, index(A)
Store reg_pointer, index(B) <- temp

.
.

Copying



Copy register state from
VMM memory

Load program counter to
Point to VM program and
Start execution

.
.

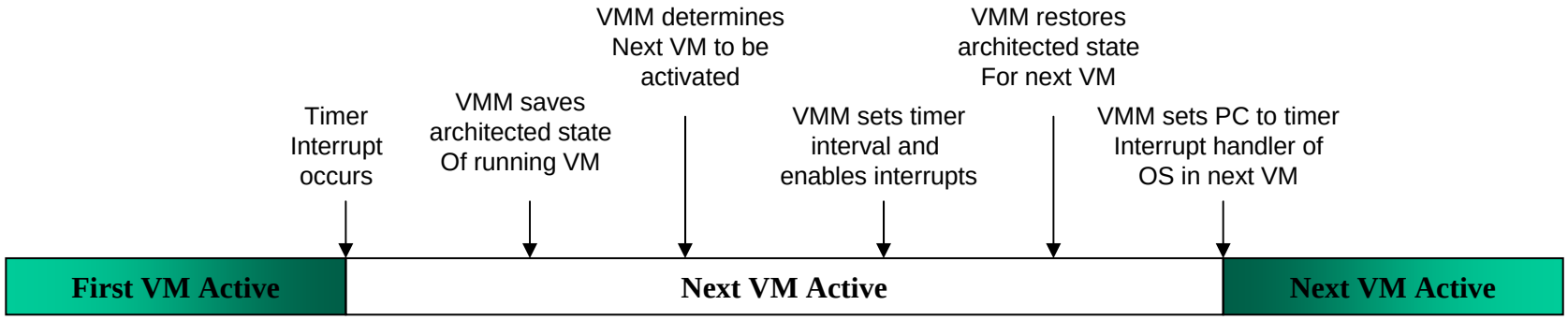
Mov reg A -> reg B

.
.

Copy register state from
Processor back to system memory

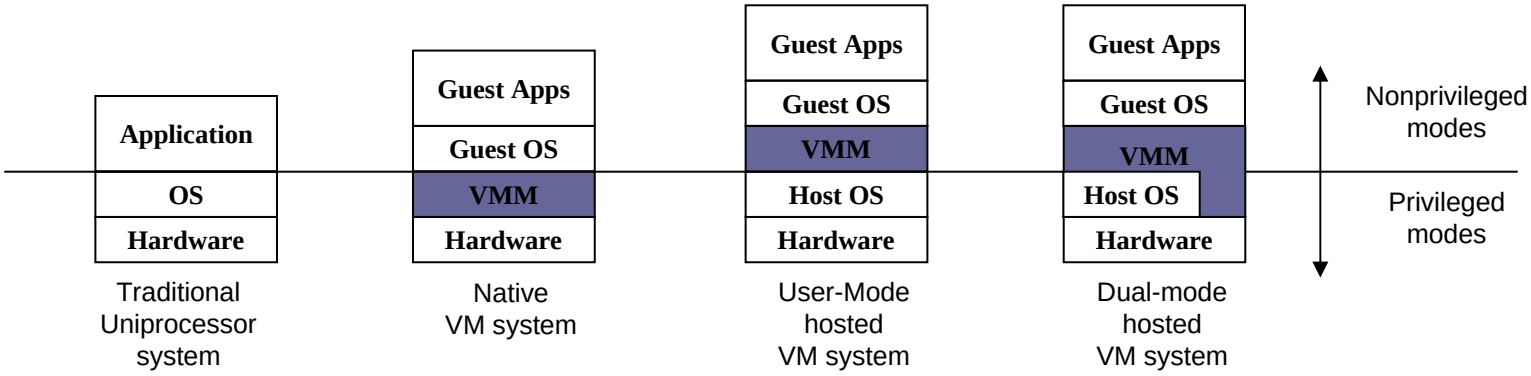
Resource Control

- The VMM maintain overall control of all the hardware resources
- Interval timer interrupt
  - Instead of allowing the OS in a VM to handle the timer interrupt, the VMM first handles the interrupt itself



Native and Hosted Virtual Machine

- A native VM system
  - VMM operates in a privilege mode higher than the mode of the guest VMs
  - The privilege level of the guest OS is emulated by the VMM
- A Hosted VM system
  - A VM is installed on a host platform that is already running an existing OS
  - The VMM utilizes the functions already available on the host OS to control and manage resources desired by each of the virtual machine



The Case of IBM VM/370

- The virtual machine monitor of VM/370
  - Control program (CP)
- A single-user operating system
  - Conversational monitor system (CMS)

Resource Virtualization - Processor

- Key idea is the execution of the guest instructions
  - Both system-level instructions and user-level instructions
- Two virtualization method
  - Emulation
    - Interpretation or binary translation which are slow
  - Direct native execution
    - Fast but only if the host ISA is the same as the guest ISA
    - Most instructions are directly executed, and the remainders are emulated
- Well-virtualizable ISA
  - A trap occurs naturally when an instruction needs to be emulated
    - the trap handler jumps to an appropriate interpreter routine, interprets the single instruction, and returns control back to the original program
  - For other ISA we cannot isolate only those instructions, thus we have to emulate more instructions

Conditions for ISA Virtualizability(1)

- Based on a paper by Popek and Goldberg
- We restrict the discussion here to native system VMs
- In a native system VM, the **VMM** runs in the **system mode**, and all other software including the **OS** runs in the **user mode**
- Assumptions:
  - The hardware consists of a processor and a uniformly addressable memory
  - Processor can operate in one of two modes, system mode or user mode
  - Some subset of the instruction set is available only in the system mode
  - Memory addressing is done relative to the contents of a relocation register

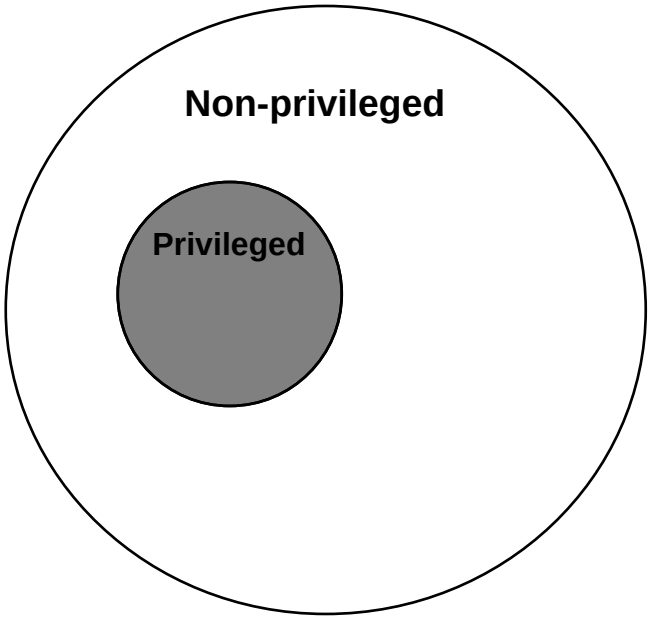
Conditions for ISA Virtualizability (2)

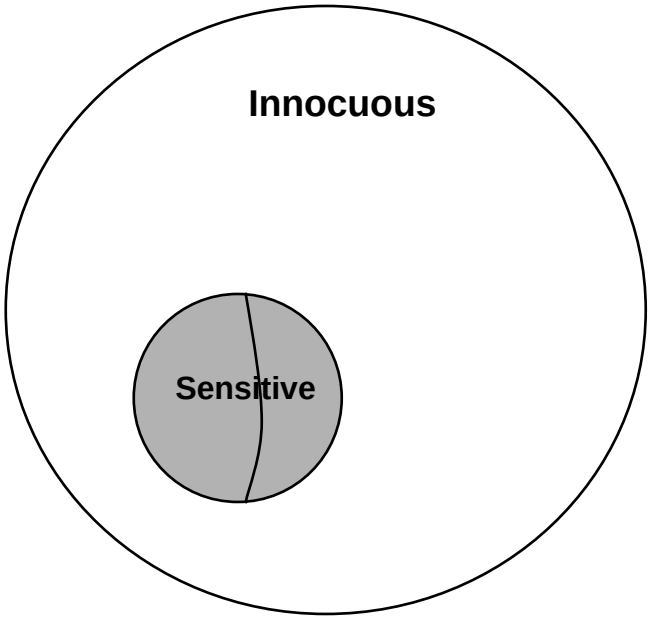
- Privileged instruction: those that trap if the machine is in user mode and do not trap if the machine is in system mode
- Examples of privileged instructions
  - Load PSW (LPSW, IBM System/370)
    - Load the processor status word (PSW) from a location in memory if the processor is in system mode. If it is not in system mode, the machine traps
  - Set CPU Timer (SPT, IBM System/370)
    - Replaces the CPU interval timer with the contents of a location in memory if the CPU is in system mode and traps if it is not
  - Not privileged if the instruction just behaves differently in the two modes
    - E.g., a system-level instruction which becomes no-op in user mode but does not trap
- Why privileged instructions are important? An OS in a VM should not affect other VMs by changing H/W resources directly, so OS should execute these instructions in user mode and must trap
- Are instructions interacting with H/W all privileged?

Conditions for ISA Virtualizability (3)

- Sensitive instructions: those that interact with hardware
  - Control-sensitive instructions
    - Attempt to change the resources in the system
    - Ex) Load PSW, Set CPU Timer
  - Behavior-sensitive instructions
    - Behavior or results produced depend on the configuration of resource
    - Ex) Load Real Address (LRA, System/370)
      - ✓ Takes a virtual address, translates it, saves the corresponding real address in a specified general-purpose register, whose behavior depends on the mapping of the real memory
    - Ex) Pop Stack into Flags Register (POPF, IA-32)
      - ✓ Pops from a stack held in memory into flag registers
- Other instructions are called Innocuous instruction
- **Is sensitive a subset of privileged? Not necessarily. See POPF**
  - In user mode, this instruction can overwrite all flags except the interrupt-enable flag
  - For the interrupt-enable flag, the instruction acts as a no-op when executed in user mode
  - **Sensitive but not privileged**

Classification of Instructions



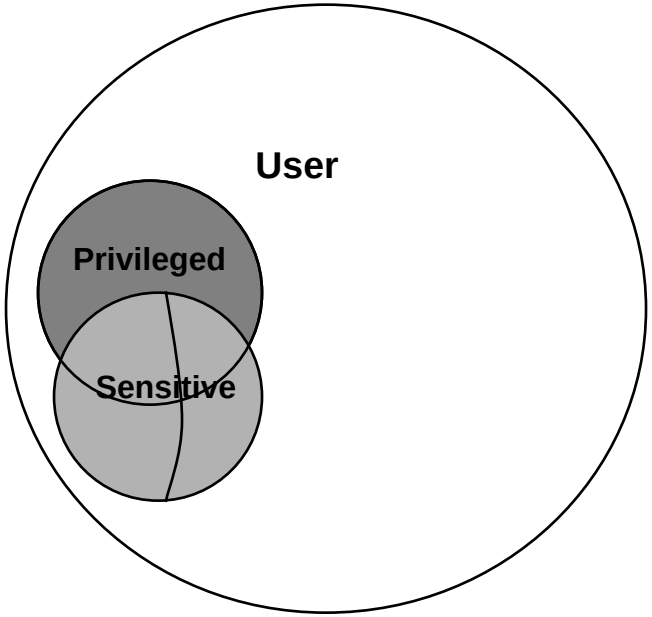


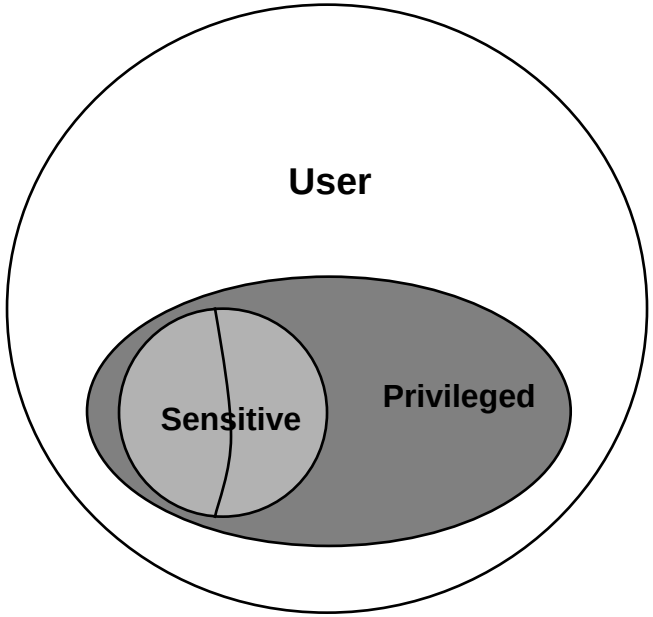
Is sensitive a subset of privileged? Maybe or maybe not

Theorem for an Efficient VMM

- A theorem for an efficient VMM construction
  - **A virtual machine monitor may be constructed if the set of sensitive instruction is a subset of the set of privileged instructions**

- An efficient virtual machine can be constructed if instructions that could interfere with the functioning of the VMM always trap in the user mode

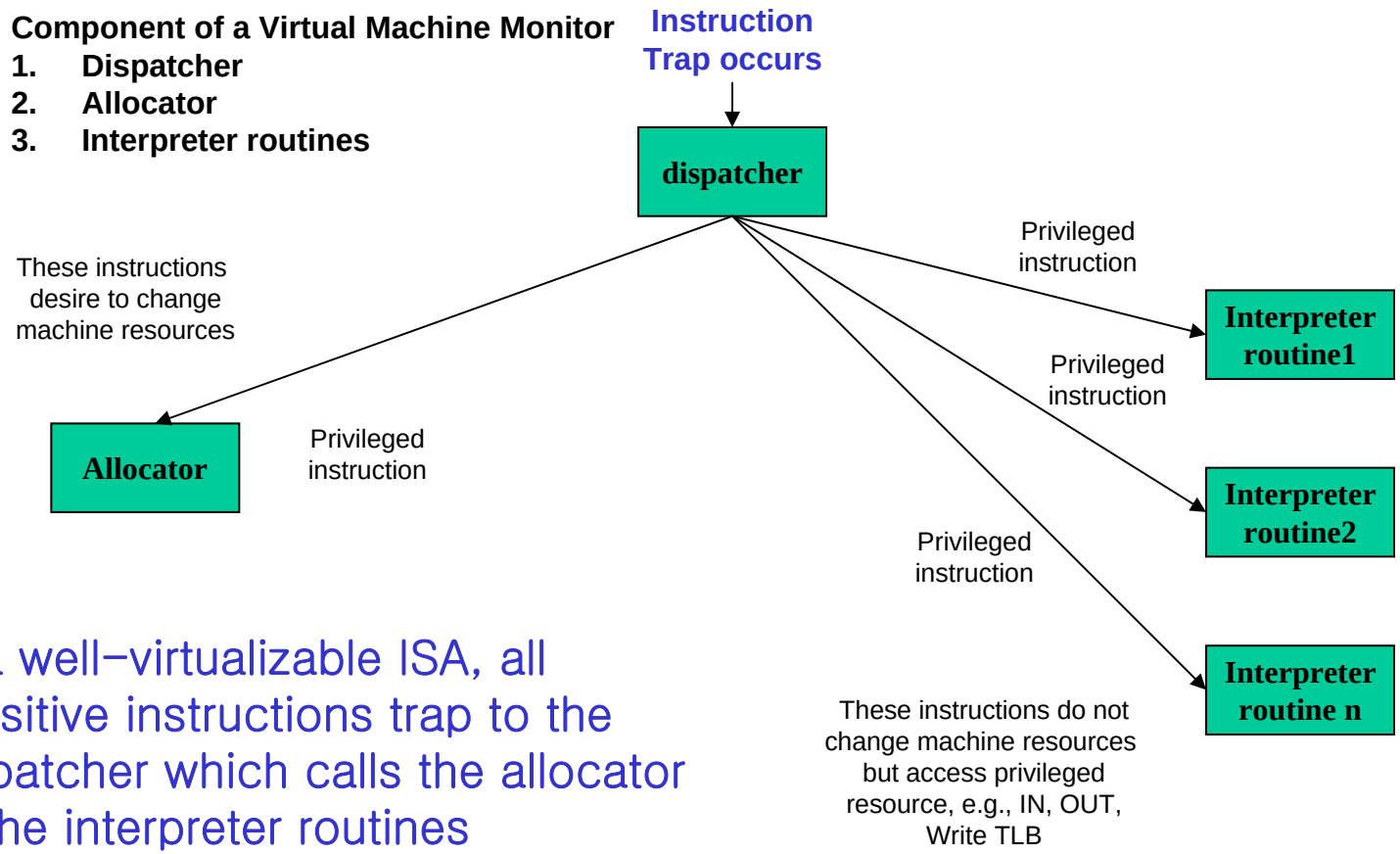




Components of a VMM

Component of a Virtual Machine Monitor

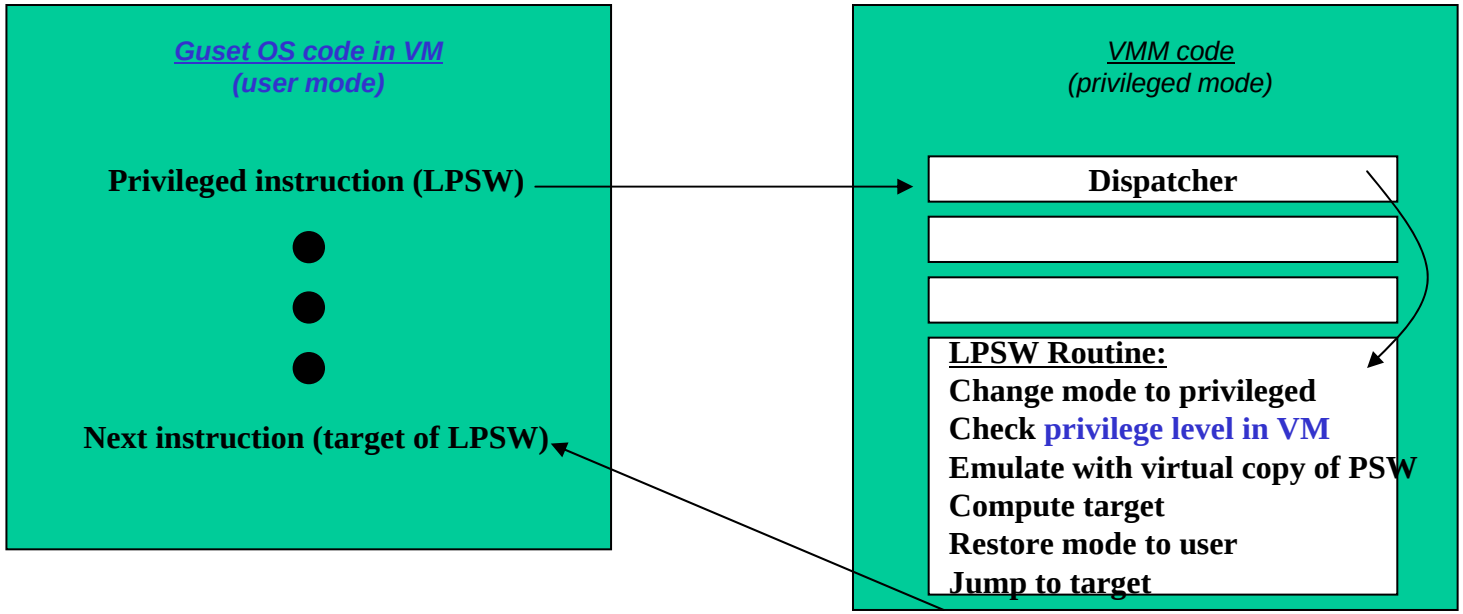
1. **Dispatcher**
2. **Allocator**
3. **Interpreter routines**



In a well-virtualizable ISA, all sensitive instructions trap to the dispatcher which calls the allocator or the interpreter routines

Emulation of Sensitive Instructions

- The VMM interprets a sensitive instruction according to state of the VM
- If LPSW is executed by the OS



- If LPSW is executed in the user mode, trap handler will generate a virtual trap which is passed to the VM and handled by the guest OS

Emulation of Sensitive Instructions

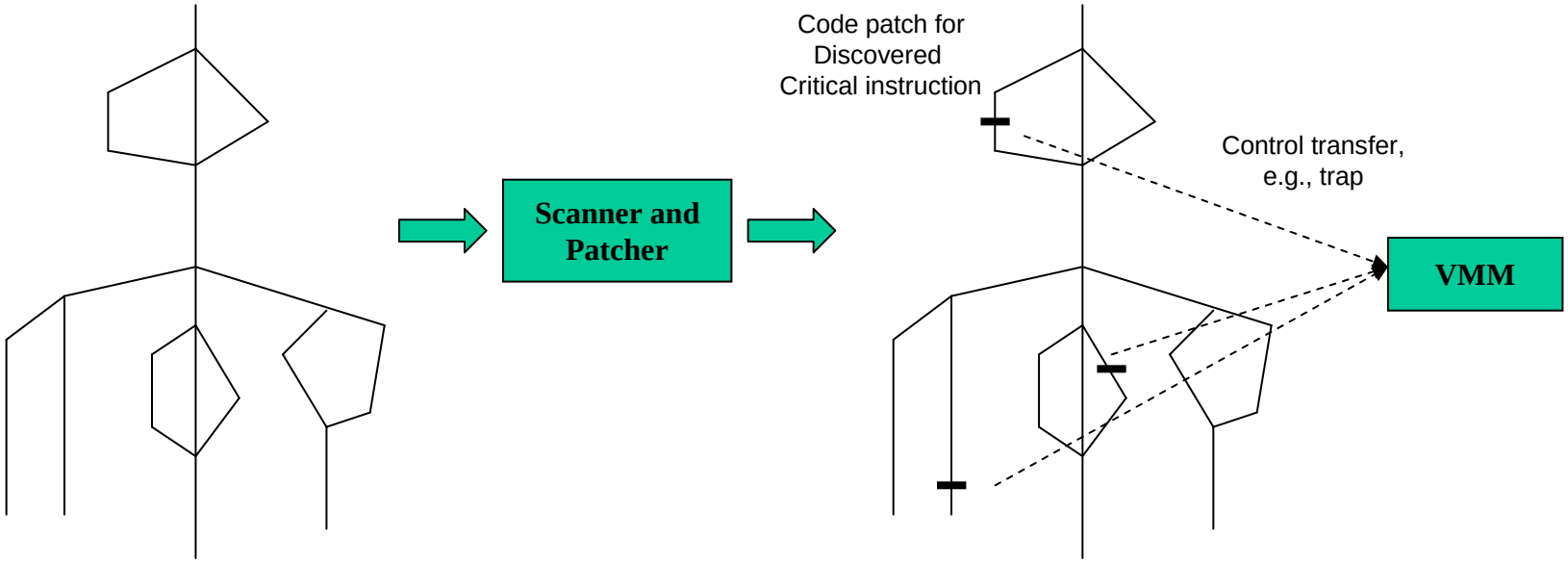
- Interpreting the SPT instruction
  - VMM examines the time value (*t*) to be loaded into the CPU timer
    - **If(*t* < T) t is loaded, else T is loaded**
      - ✓ ***t* : the content of the location**
      - ✓ **T : the time remaining from the allocated time for the virtual machine itself**
  - Meanwhile, it keeps the time difference(*t* - T) in an internal table so that this time can be restored when the guest VM is again activated

Handling Problem Instructions

- The POPF instruction is sensitive but not privileged
  - Called critical instructions
  - It does not generate a trap in user mode
  - IA-32 includes several (17) critical instructions

- Additional steps are needed with some loss of efficiency
  - It is possible for a VMM intercepts POPF and other critical instructions if all guest software were interpreted, but very inefficient
  - VMM scan the guest code before execution, discover all critical instructions, replace them by a trap or a jump to the VMM

Scanning and Patching

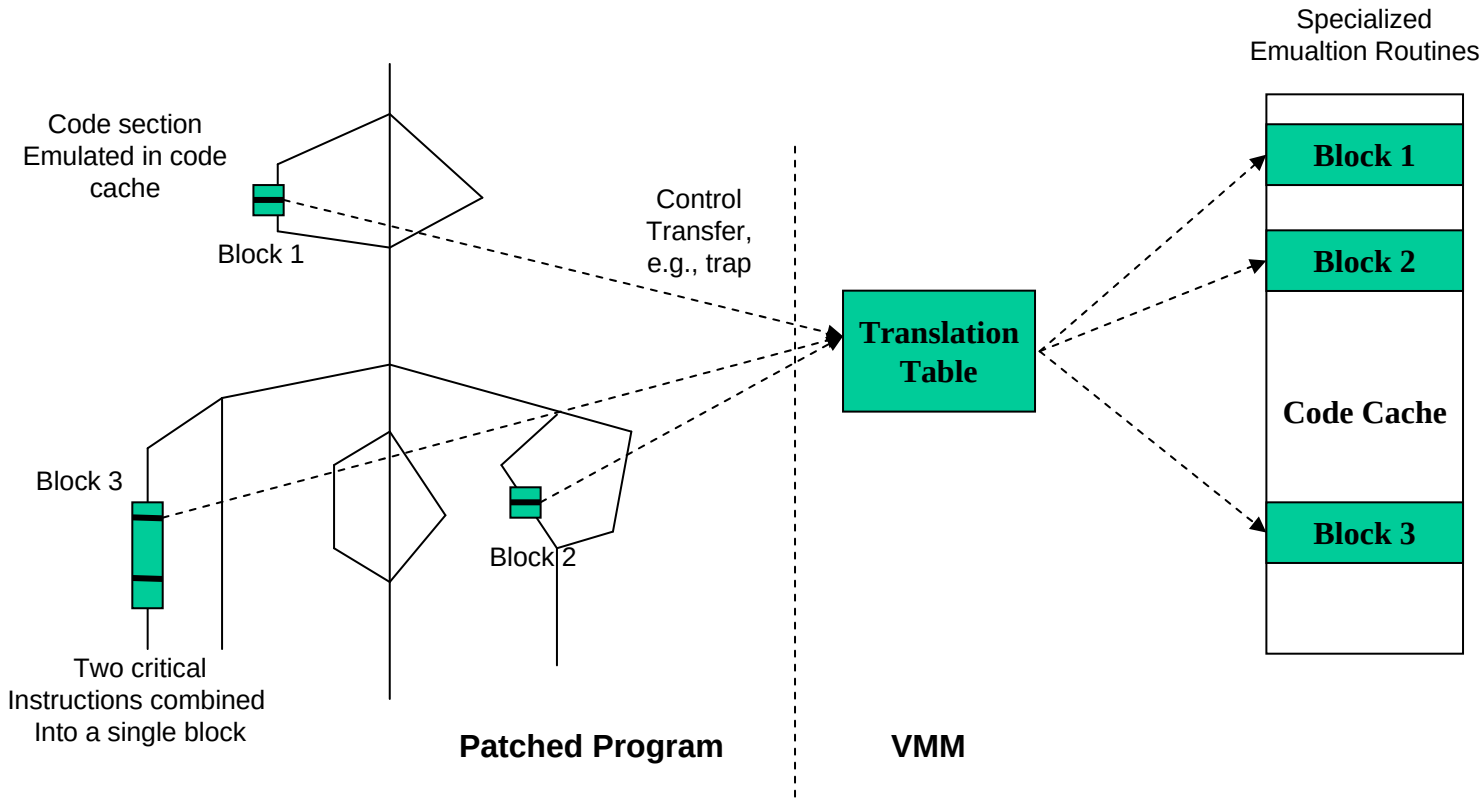


Patching of Critical Instructions

- One way to discover critical instructions
  - The VMM takes control at the head of each guest basic block and scan instructions in sequence until the end of the basic block is reached
    - If a critical instruction is found, it is replaced with a trap to the VMM
    - Another trap back to the VMM is placed at the end of the basic block
- To reduce overhead, the trap at the end of a scanned basic block can be replaced by the original branch or jump instruction after enough patching
- We can also scan beyond basic block boundaries until target is not clear

Caching Emulation Code

- To reduce the overhead of frequent interpretation for the same instance of critical instructions, save the translated block in a code cache
- Control is transferred to the code cache with a trap



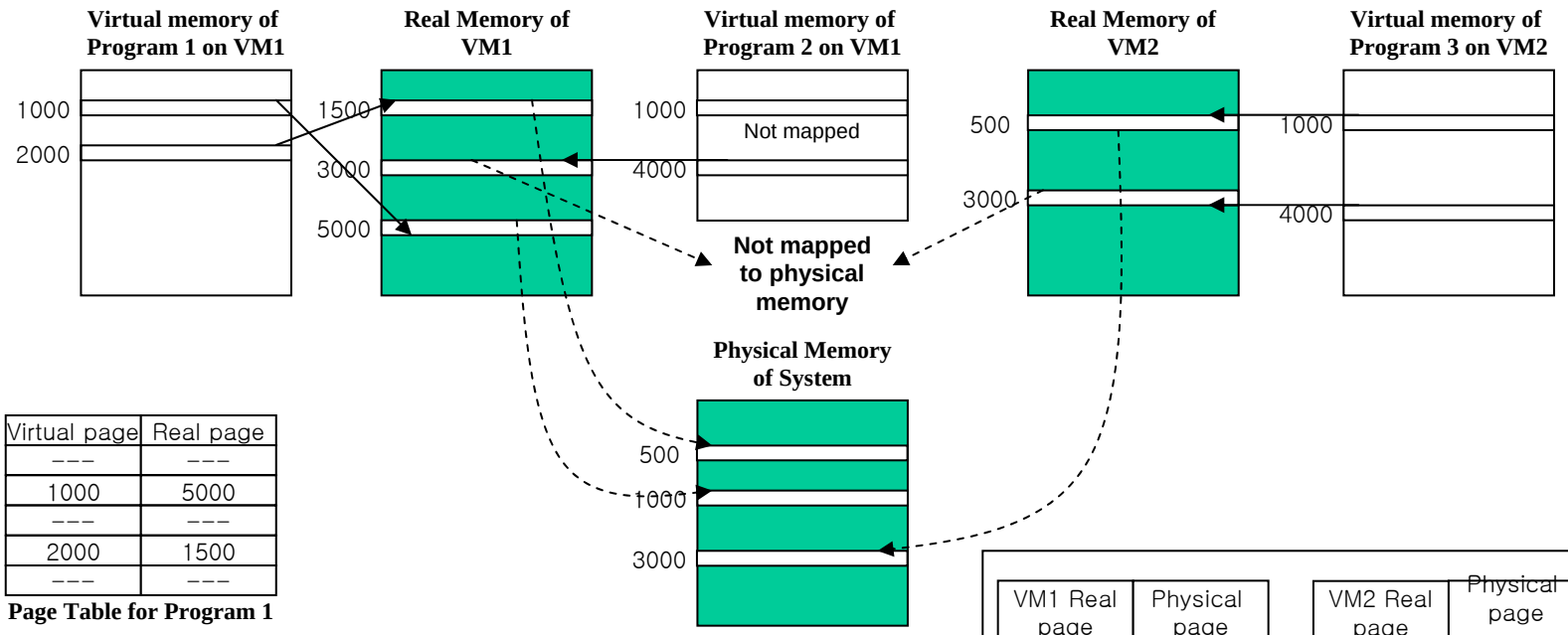
Resource Virtualization – Memory

- Most ISA supports translation of virtual addresses to real addresses
  - **Using the concept of page tables and TLB**
- These real addresses would be physical addresses in native execution
- However, in VM, real addresses should be translated to physical addresses

- In VM, real memory and physical memory is different
  - Real memory: a guest VM's illusion of physical memory
  - Physical memory: the hardware memory

- A guest's real memory address must undergo a further mapping to determine the address in physical memory of the host hardware
- VMM maintains a real map table mapping the real pages to physical pages

Virtual Memory Support in a System VM



Virtual page	Real page
---	---
1000	5000
---	---
2000	1500
---	---

Page Table for Program 1

Virtual page	Real page
---	---
1000	Not mapped
---	---
4000	3000
---	---

Page Table for Program 2

Virtual page	Real page
---	---
1000	500
---	---
4000	3000
---	---

Page Table for Program 3

VM1 Real page	Physical page
---	---
1500	500
---	---
3000	Not mapped
---	---
5000	1000
---	---

Real Map Table for VM1

VM2 Real page	Physical page
---	---
500	3000
---	---
3000	Not mapped
---	---

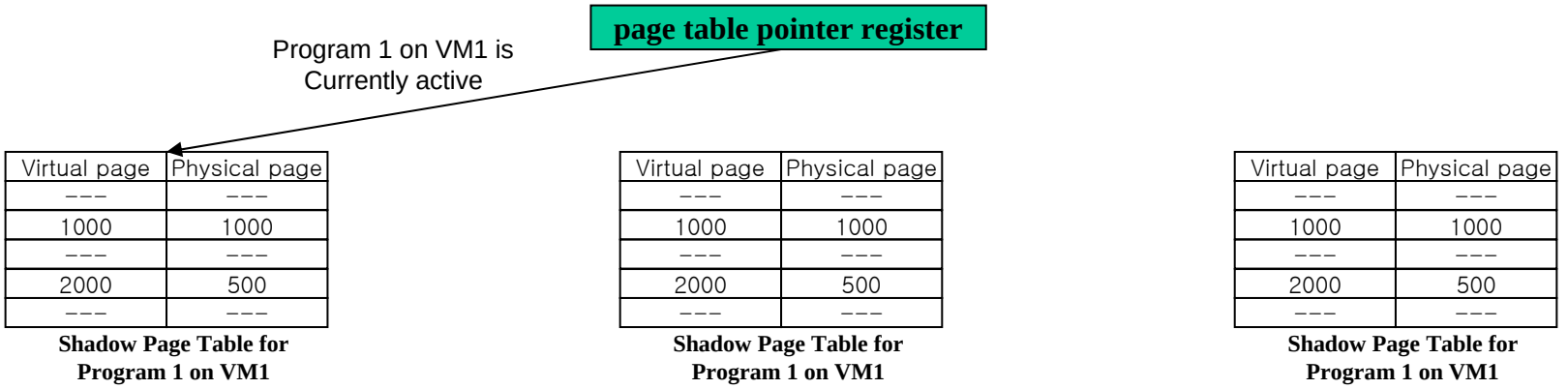
Real Map Table for VM2

Two Virtual Memory Architectures

- Page translation in most ISA is supported by a page table and a TLB
- Two architectures (architected means O/S-managed)
  - Architected page table
    - **Page table is maintained by H/W and OS, with page table pointer register**
    - **TLB is maintained only by H/W and not visible to OS**
    - **For a TLB miss, H/W walks page table to find an entry; if there is none (not mapped to real memory), it is a page fault and OS takes over**
    - **E.g., IA-32**
  - Architected TLB
    - **Page table is maintained by OS only and is not visible to H/W**
    - **TLB is maintained by H/W and OS**
    - **For a TLB miss, there is an immediate trap to the OS**
    - **E.g., RISC CPUs**
  - Memory virtualization differs between the two

The Case of Architected Page Tables

- VMM maintains shadow page tables, one for each guest VM
  - Virtual-to-physical mapping
  - These tables are the ones actually used by hardware to translate virtual addresses and to keep the TLB up-to-date
- To make this method work, page table pointer register (PTPR) is virtualized
  - VMM manages the real PTPR and a virtual PTPR for each VM
  - When a guest VM is activated, the real PTRT points to the shadow page table
  - When a guest attempts to read/write a PTPR, it is intercepted by the VMM
    - VMM reads/writes the virtual PTPR while updating the real PTPR if needed



Page Faults with Architected Page Tables

- When a page fault occurs (TLB miss and no entry in page table)
  - If the page is mapped in the virtual page table of the guest OS
    - When the VMM has moved the accessed real page to its own swap space
    - VMM brings the real page back into physical memory
    - VMM updates the real map table and the affected shadow table(s)
    - Guest OS is not informed of any page fault
  - If the page is not mapped in the virtual page table of the guest OS
    - VMM transfers control to the trap handler of the guest, indicating a page fault
    - Guest OS then issues instruction to modify its page table
    - VMM intercepts these request (privileged or write-protected page table)
    - VMM updates the guest page table and also updates the mapping in the appropriate shadow page table
  - **I/O request with real addresses is converted to physical addresses using the real map table (if swapped out by VMM, must be read back)**

The Case of Architected TLBs

- TLB itself is virtualized by maintaining a copy of each guest's TLB contents and the real TLB by VMM (any instruction modifying TLB is intercepted)
- Two ways of managing the real TLB
  - Rewrite the TLB whenever a guest VM is activated
    - After translating real address in virtual TLB to physical address in physical TLB
    - The VMM copies all VM's virtual TLB entries into the physical TLB, which is costly
  - Uses address space identifier (ASID), originally for multi processes to share TLB
    - There is an architected ASID register which contains the ASID of active process
    - Each guest VM has a virtual ASID register
    - VMM maps guest ASID to real ASID
    - Real TLB is shared by different guests' TLB entries

The Case of Architected TLBs

Real TLB has translations from two VMs are simultaneously present

Virtual TLBs

ASID Mapping:
Prog. 1 – ASID 3
Prog. 2 –ASID 7

ASID	Virtual page	Real page
---	---	---
3	1000	5000
---	---	---
3	2000	1500
---	---	---
7	4000	3000
---	---	---

Virtual TLB of VM1

ASID Mapping:
Prog. 1 – ASID 3

ASID	Virtual page	Real page
---	---	---
3	1000	3000
---	---	---
---	---	---
---	---	---
---	---	---
---	---	---

Virtual TLB of VM2

ASID Map Table

Virtual ASID	Real ASID
---	---
VM1:3	9
---	---
VM1:7	---
---	---
VM2:3	4

Real TLB

ASID	Virtual page	Real page
---	---	---
9	1000	1000
---	---	---
4	1000	3000
---	---	---
9	2000	500
---	---	---