System Virtual Machines

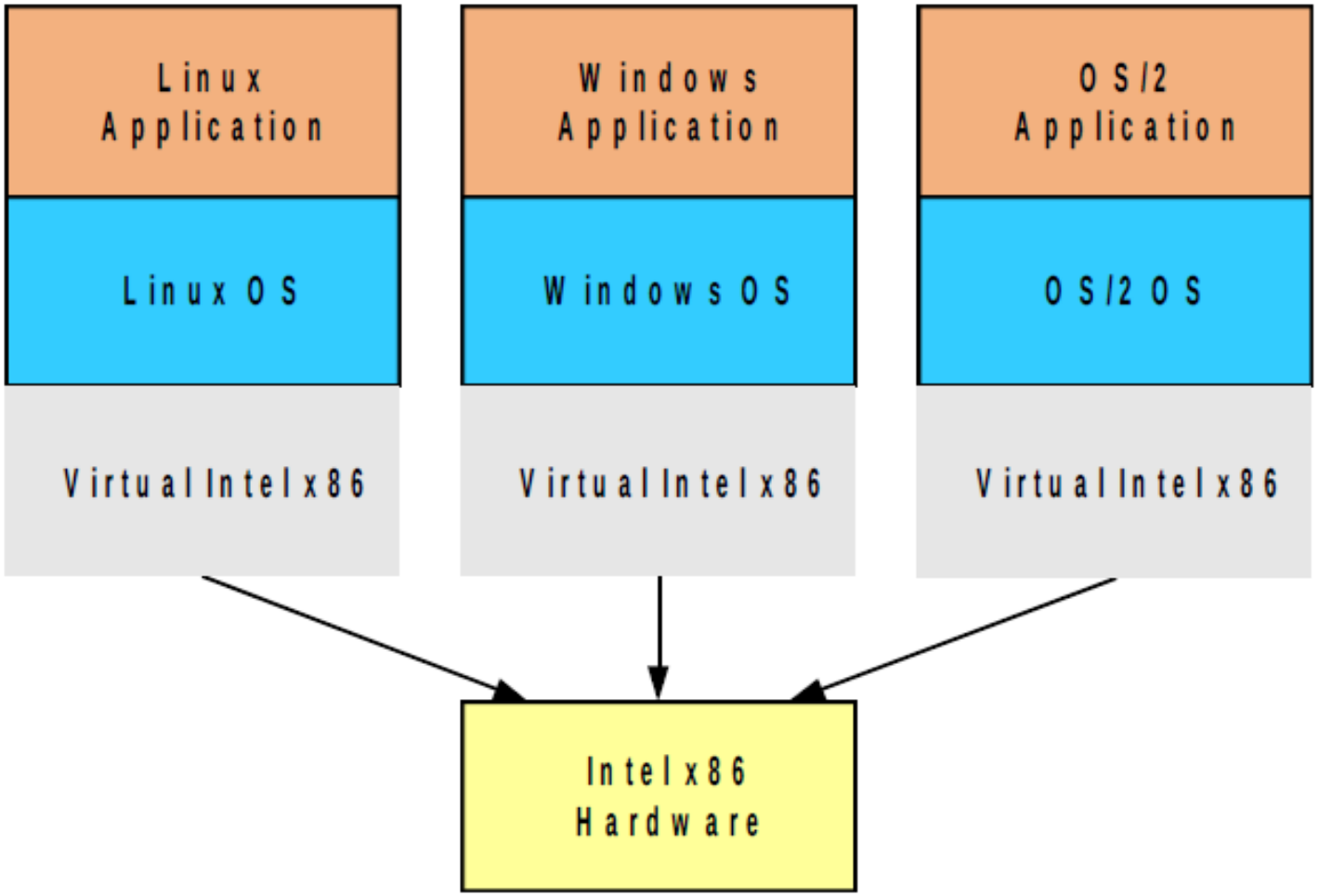
Outline

- Need and genesis of system Virtual Machines
- Basic concepts
  - User Interface and Appearance
  - State Management
  - Resource Control
  - Bare Metal and Hosted Virtual Machines
  - Co-designed Virtual Machines
- Case-Studies
  - Native Virtual Machines: Xen, Vmware-Esxi
  - Hosted Virtual Machines: Vmware-Workstation, Palazzo, Linux-KVM

Need for System VMs

- Multiple single-application Virtual Machines
- Multiple secure environments
- Managed application environments
- Mixed-OS environments
- Legacy applications
- Multi-platform application development and testing environment
- System Encapsulation

Conceptual view of System VMs



Key concepts of System VMs

- Outward appearance
- State Management
- Resource Control
- VM realization:
  - Native VMs
  - Hosted VMs

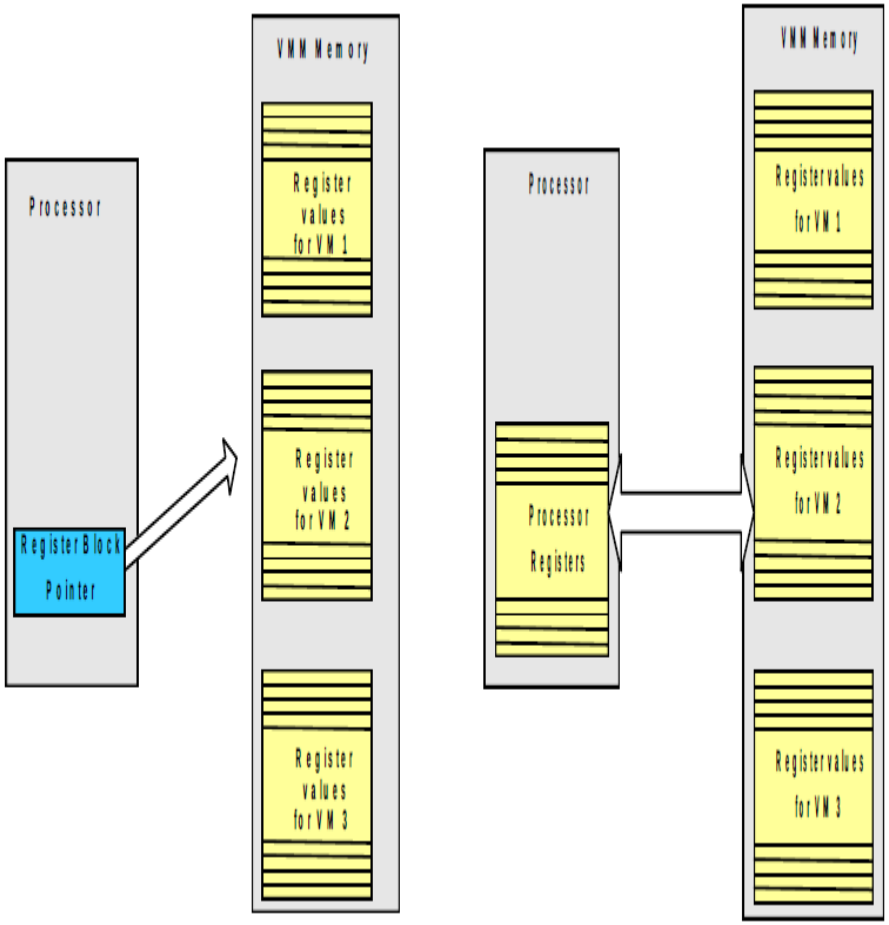
Assumptions: Uni-processor machine supporting same ISA VMs

Outward Appearance

- System VMs give the illusion of hosting multiple systems on a single machine
- Using software virtualizers:
  - Time-multiplexing of resources from one VM to another
- Using Hardware replication:
  - Independent sets of replicated hardware resources dedicated to specific VMs
  - Common resources are time-shared
- Hosted VMs:
  - One of the VM is more privileged than others
  - Other VMs are manifested in special environments of the hostOS.

State Management

- The VM state is composed of collective information from processor registers, memory pages, files etc.
- Single machine hardware may not be sufficient to hold multiple VMs state information.
- Store each VM state in fixed locations in the VMM's memory and identify using a pointer to the present active VM.
- VM State contents can be accessed through redirection or copy mode.

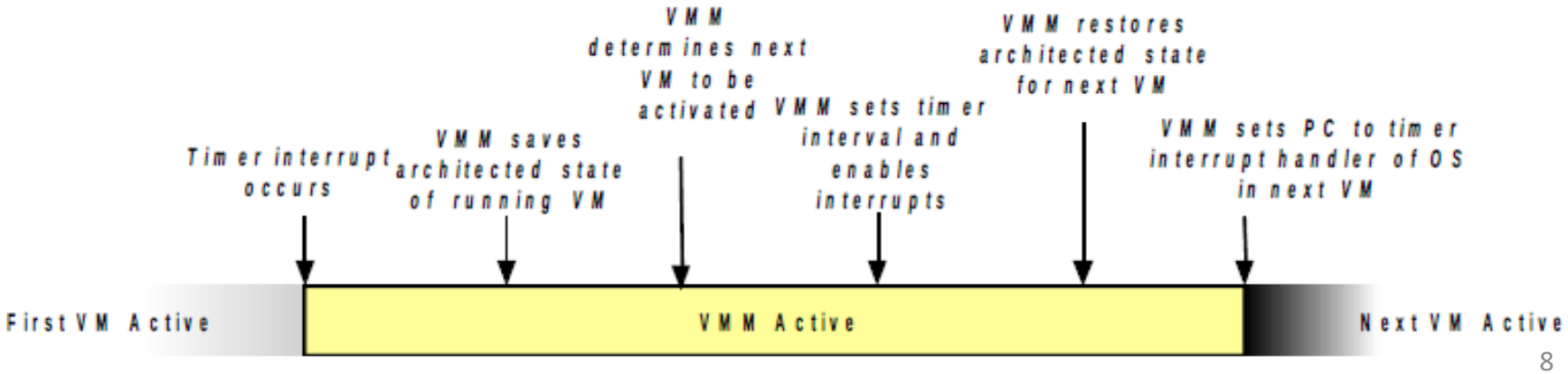


Using Indirection

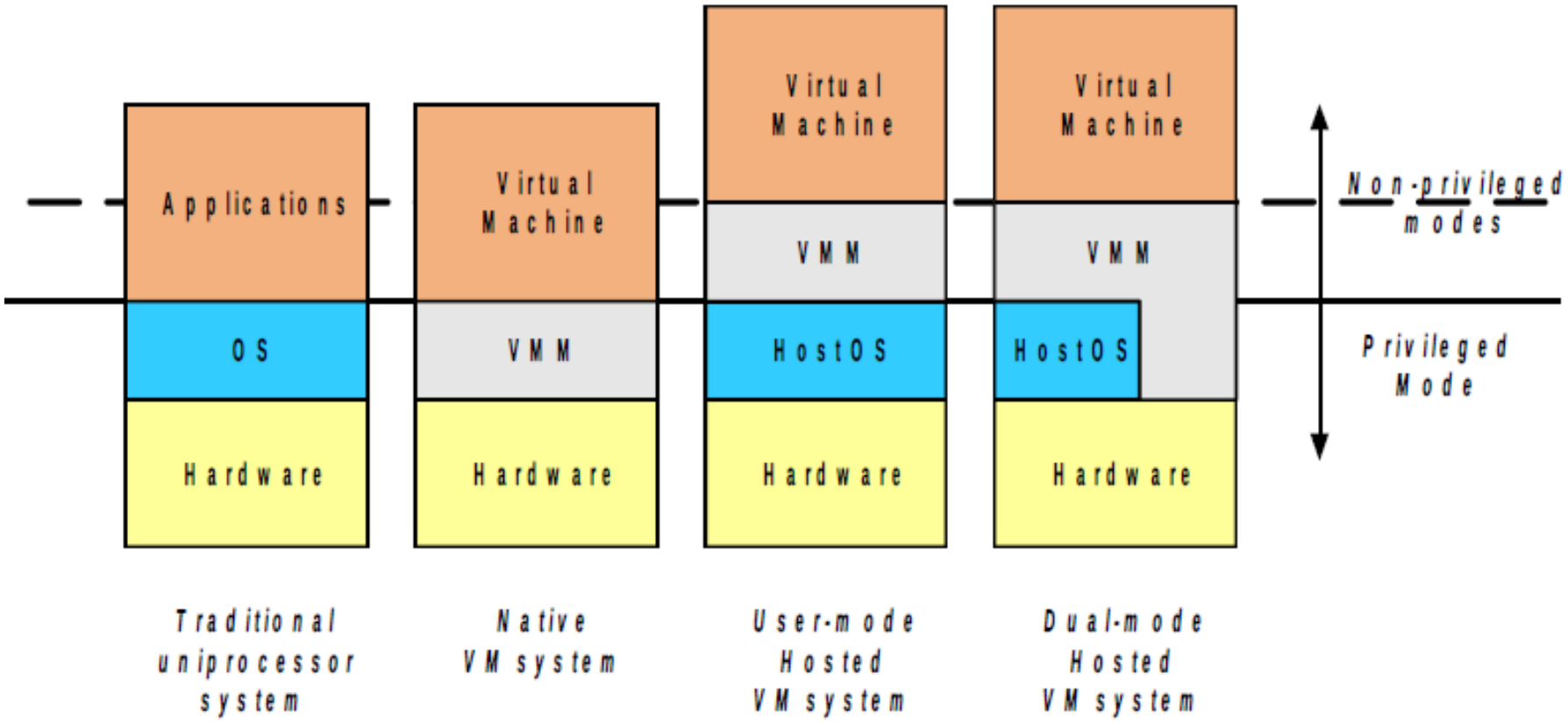
Using Copy

Resource Control

- The VMM maintains overall control of all hardware resources.
- Each VM is allocated virtual resources according to configuration specifications.
- VMM allows for direct execution of non-privileged instructions from within VM's environment.
- All privileged instructions trap to the VMM for processing.
- VMM handles the timer interrupts
- All shared resources are time-shared across contesting VMs using fair-shared or credit share scheduling.



VM Realization: Native & Hosted VMs



Resource Virtualization - Processors

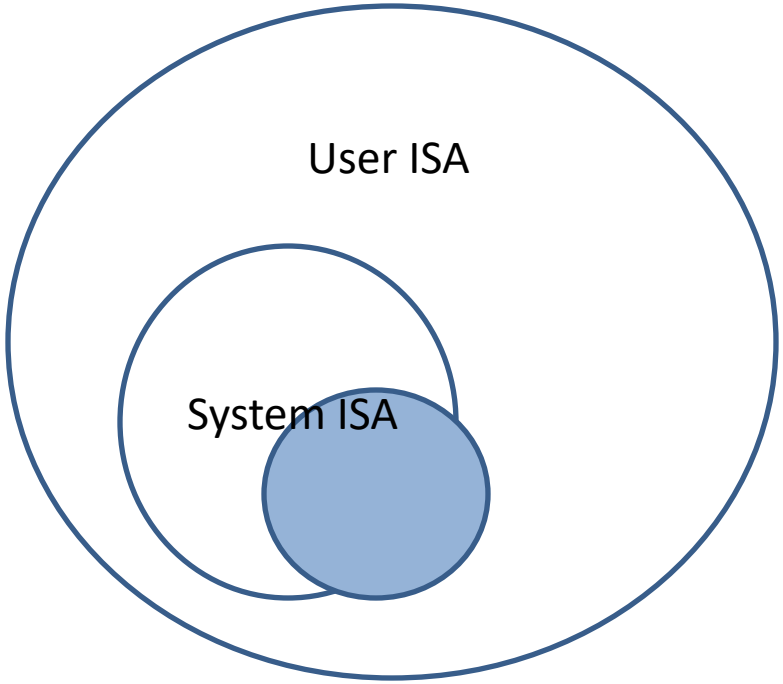
- Processors:
  - Same ISA VMs
    - Direct execution through VM
    - Emulation or para-virtualization of privileged instructions
  - Different ISA VMs
    - Emulation or binary translation

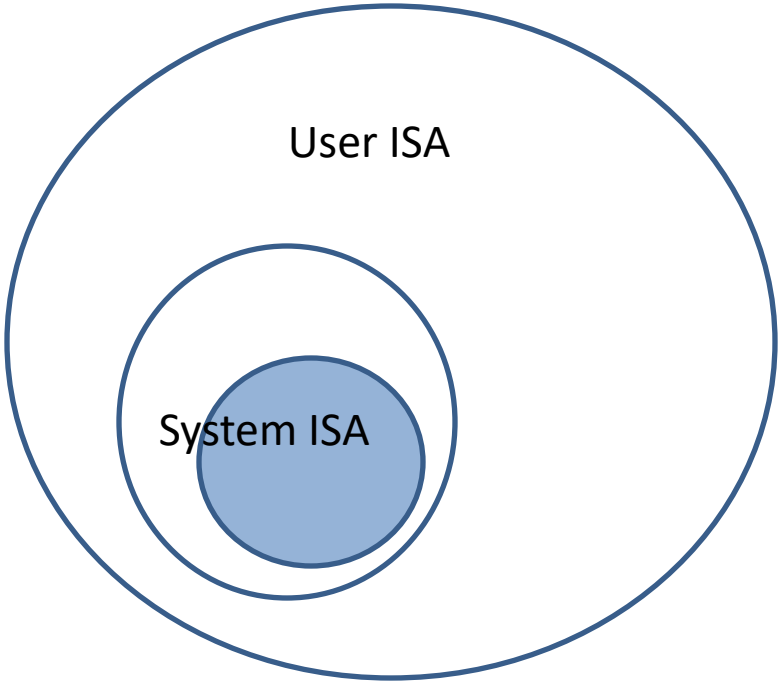
Prerequisite come prepared for the rest of classes by reading Appendix A – Real Machines from Smith & Nair's Virtual Machine text book.

ISA Virtualizability

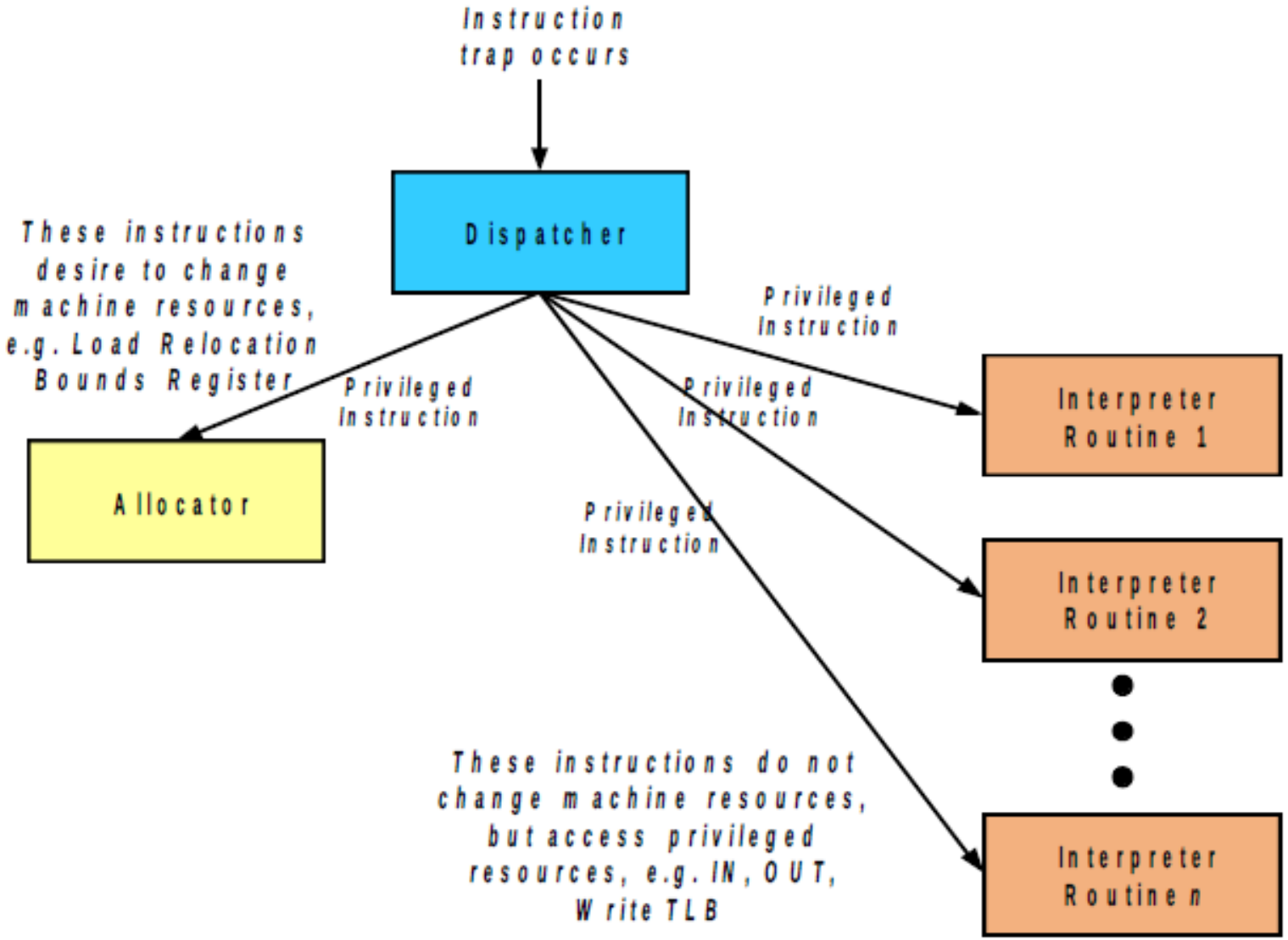
- User ISA: Instructions that enable use of hardware (innocuous instructions)
- System ISA: Instructions that enable management of hardware and access to nonconcurrent devices (sensitive and privileged instructions)
- ISA must allow multiple modes of operation
  - User mode (for innocuous instrs)
  - Privileged mode (for privileged instrs)
- All privileged instructions when executed in user mode must trap to OS.
  - Control-sensitive instructions are those that change the configuration of the system resources
  - Behavior-sensitive instructions are those whose behavior or results change with the configuration of resources

Necessary condition for ISA Virtualizability

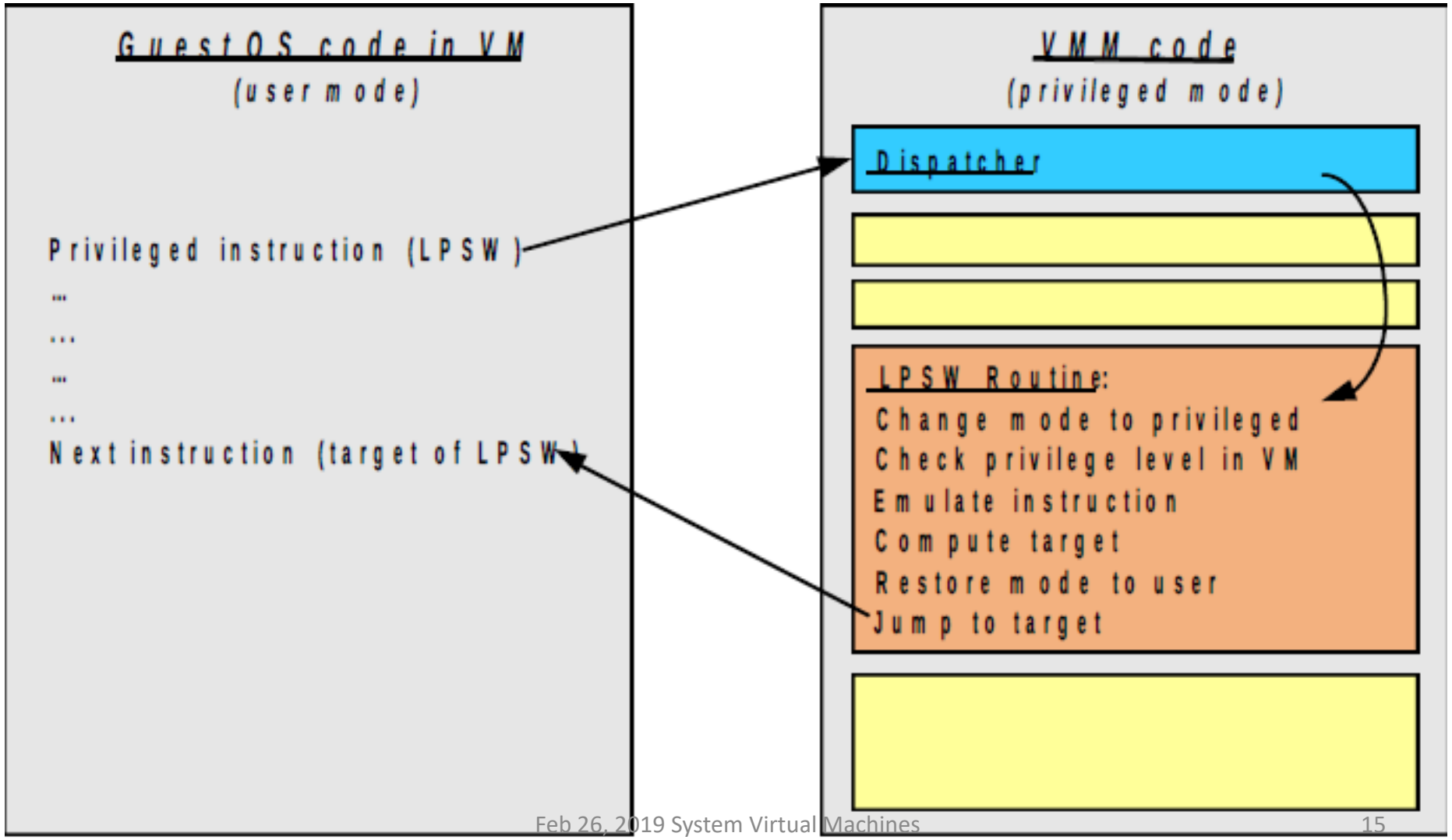




Components of a VMM



Handling of Privileged Instruction in GuestOS

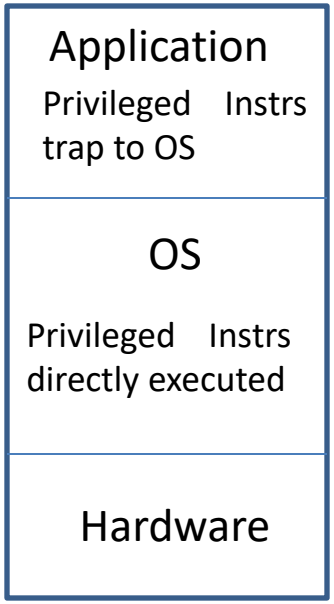


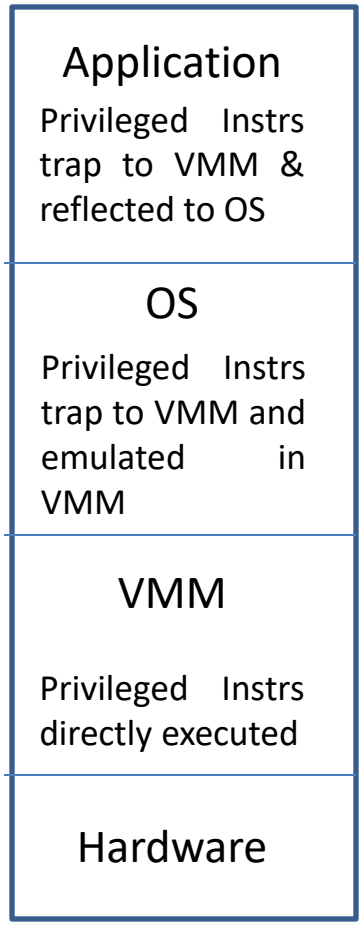
Difficulty in i86 Virtualization

- Some of the sensitive instructions in Intel IA-32 are not privileged (POPF instruction)
- Handling problem instructions
  - Interpret the guest software
  - Scan and patch before execution (standard techniques of binary translation are applied)
  - Every sensitive instruction leads to a change in privilege level since GuestOS is executing in non-privileged mode.

Execution of Privileged instructions in VM/370

- System VM/370 is ISA virtualizable
  - All sensitive and privileged instructions trap to OS when executed in user mode.
  - Supports two CPU modes – user and system





Concept of Virtual CPUs

- Implementation specifics of virtualizing a CPU
- Hypervisor schedulers
  - What are they?
    - Multiple VMs share the same CPU
    - Variants of time-sharing CPU schedulers are mostly used
  - How do they impact the application performance when executed inside the VM?

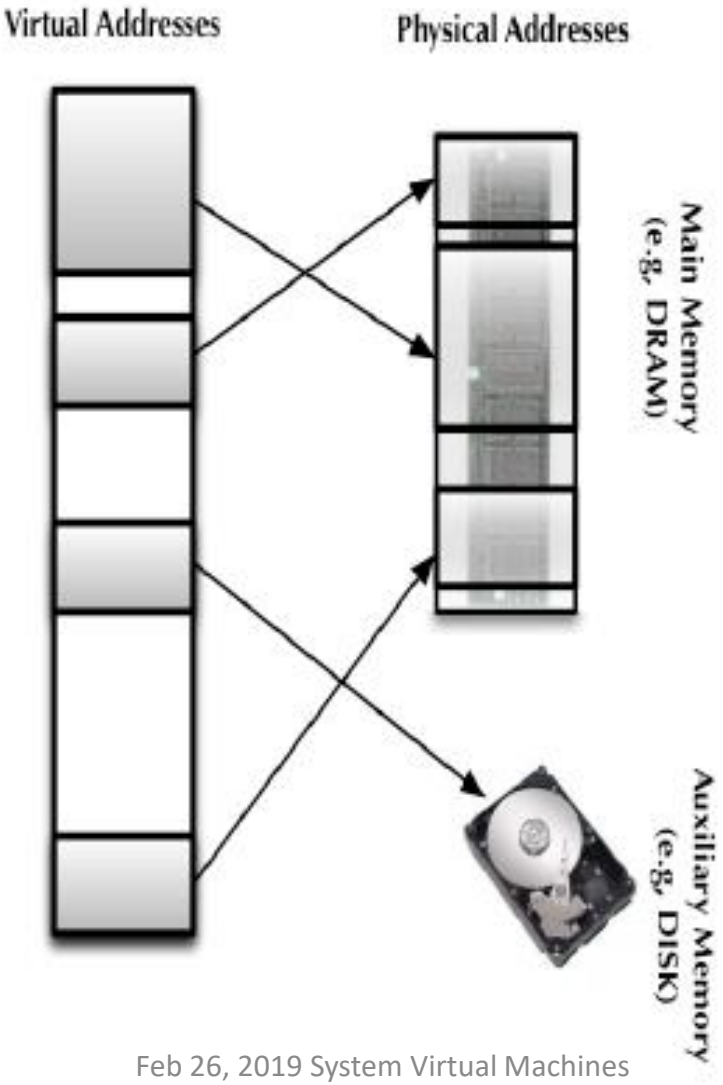
Resource Virtualization - Memory

- Most virtualization technologies extend the prevailing Virtual Memory concept to support memory virtualization for VMs.
- Each VM is given a logical view of its real memory using the virtual memory in VMM.
- Each VM can support virtual memory in its address space on its real memory.
- The real memory of any VM is further mapped to the physical memory by the VMM.
- Every virtual address in application process undergoes two translations:
  - Application virtual address to VM's real memory
  - VM's real memory to Host's physical memory

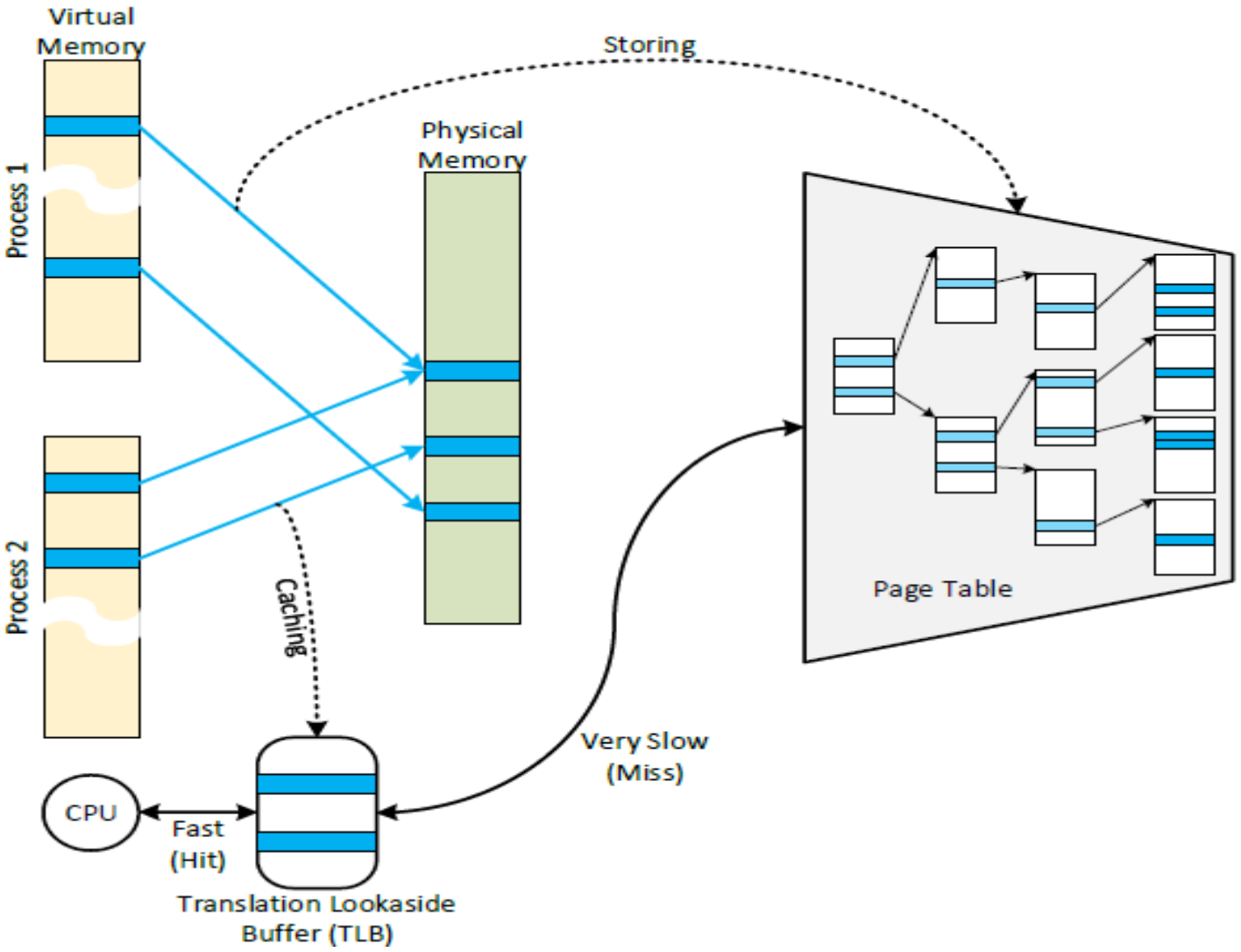
Support for Virtual Memory

- Virtual Memory basics:
  - Application is provided with a logical view of memory in the form of virtual address space.
  - Underlying OS manages the real memory (hardware memory)
  - Per process Page-Tables used to map logical (virtual address) to real (physical address) memory.
  - TLBs cache the most commonly used mappings
  - ISA defines the support for page table walks and page sizes.
- Segment registers or Relocation registers to support virtual addresses
- Address translation hardware
- Page-tables and TLBs to map physical to virtual addresses
- Page table cache structures to fasten the address translation lookups.

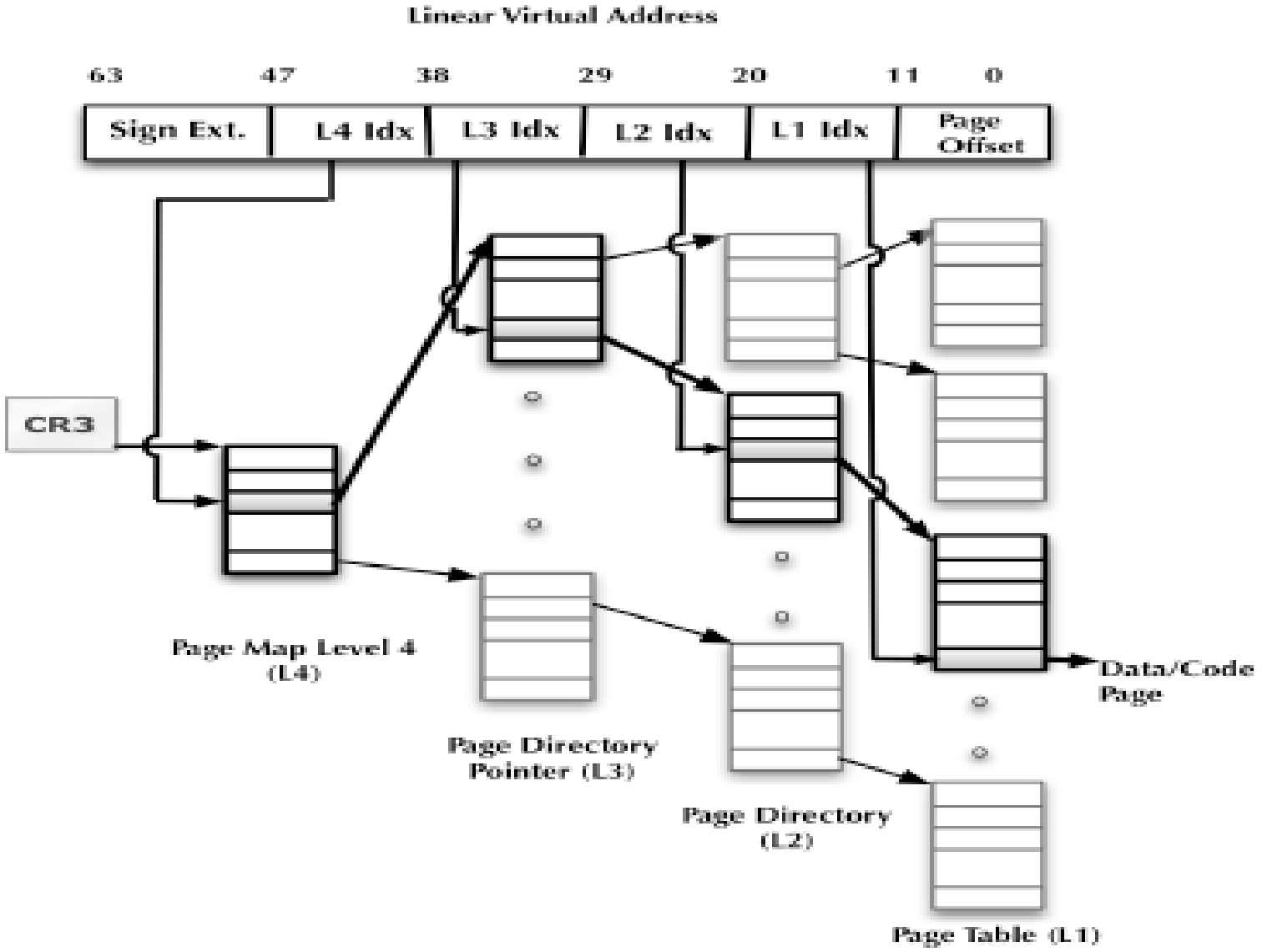
Virtual Memory Abstraction

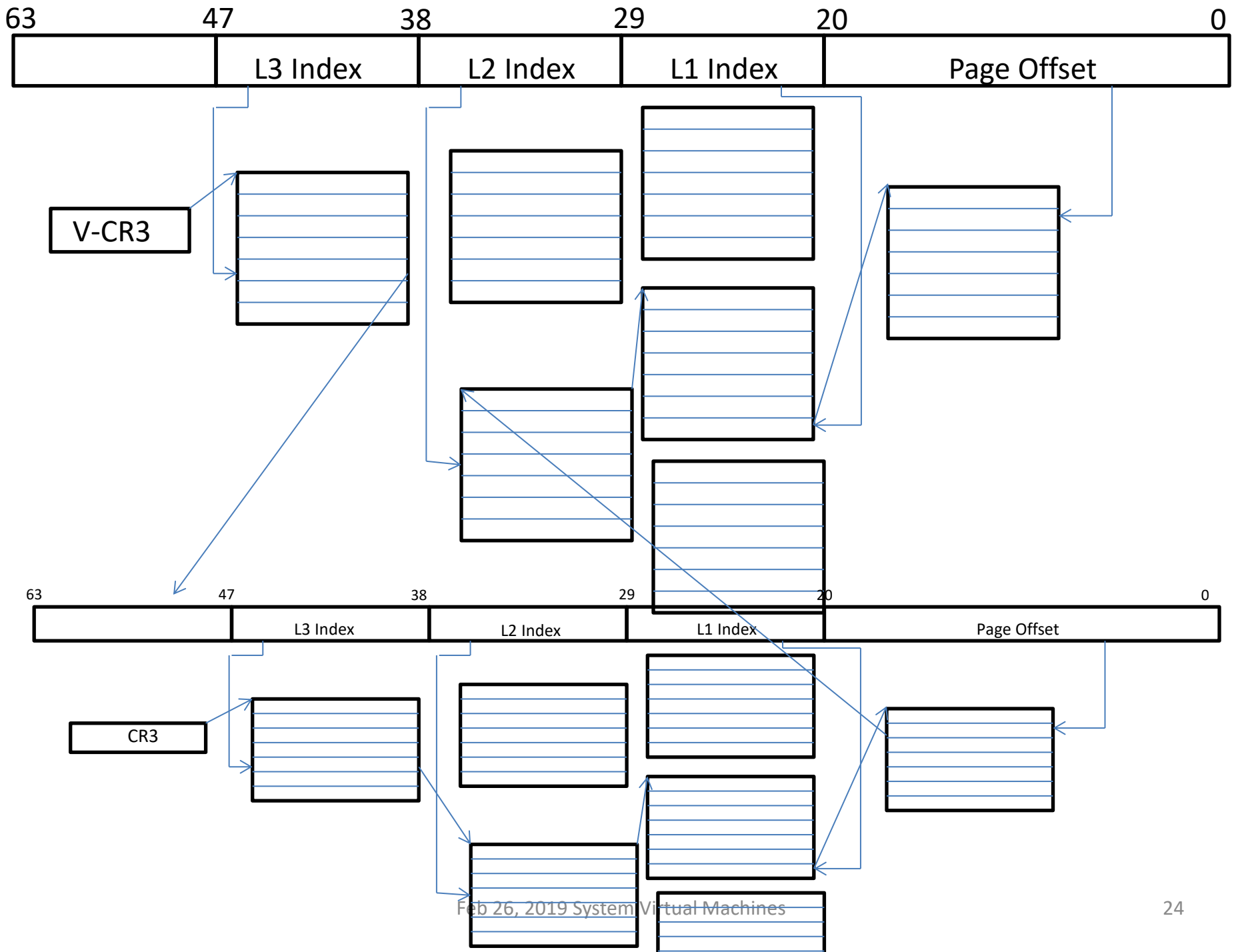


Virtual to Physical Address Translation



Page Table walk in x86-64 with 4KB Pages

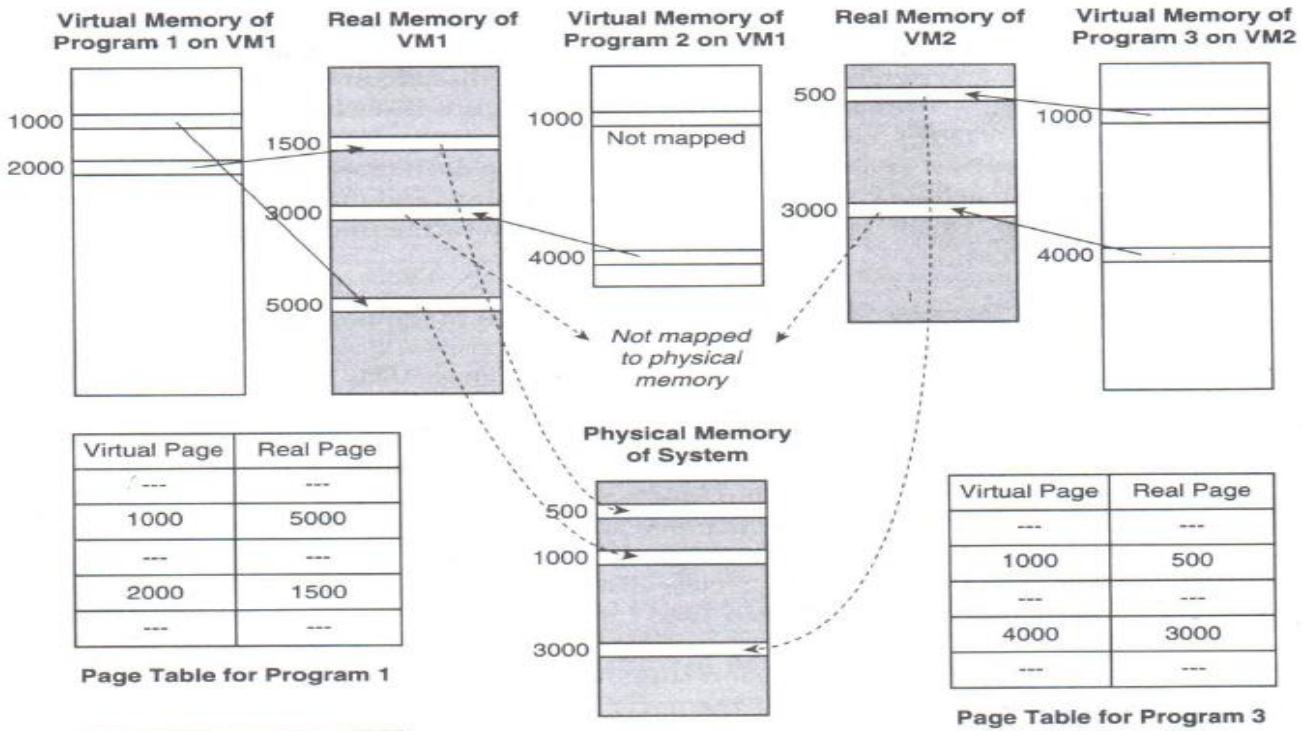



63
47
38
29
20
0
L3 Index
L2 Index
L1 Index
Page Offset
V-CR3
63
47
38
29
20
0
L3 Index
L2 Index
L1 Index
Page Offset
CR3

Memory Virtualization Issues with page based structures

- Hypervisors perform two-level address translation to support memory virtualization:
  - Guest Virtual Address to Guest Physical Address
  - Guest Physical Address to Host Physical Address
- Every TLB/Cache miss introduces this penalty.
- Virtualization using this technique makes it unattractive for workloads that have significant memory reads and writes!

Software methods for memory virtualization – Shadow page tables



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

- GuestOS of each VM maintains its own page tables to map GVA → GPA
- VMM maintains shadow page tables to map GPA → HPA
- Shadow page tables are used by the hardware to translate virtual addresses to keep TLB updated
- The Page Table Pointer is virtualized and VMM manages it.
- This technique is used on hardware where the ISA dictates page table architecture.

Software methods for memory virtualization – Virtualized TLBs

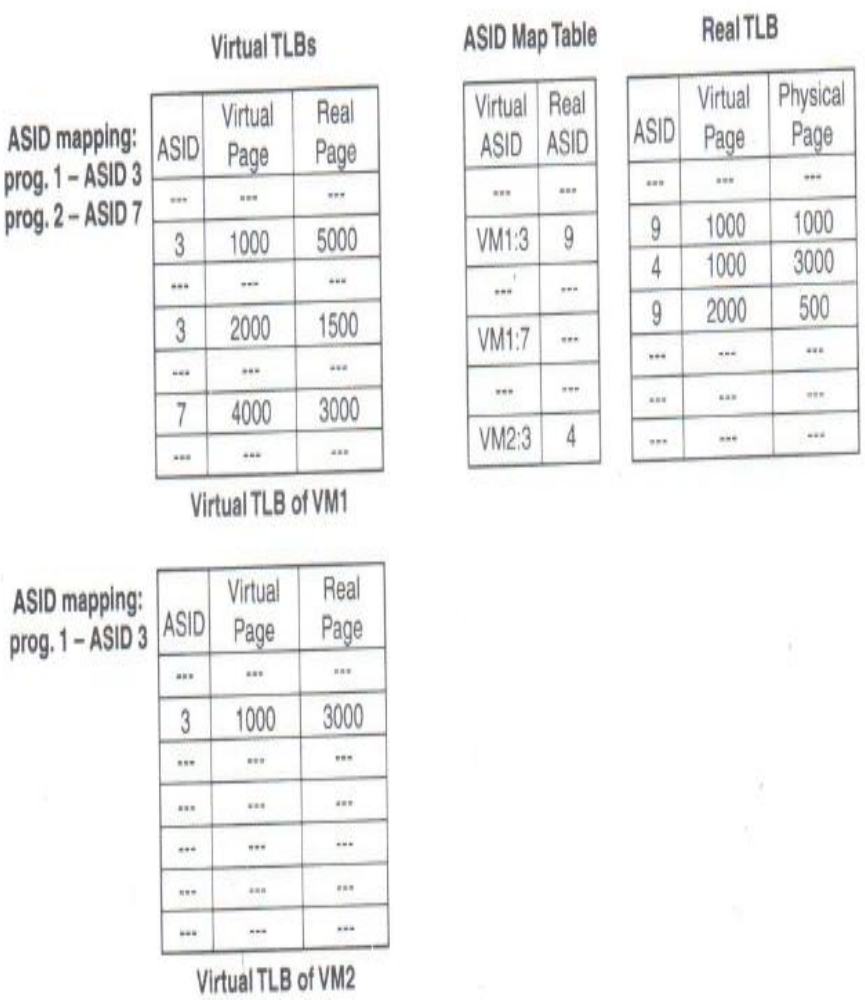
OS based TLB Management

- Some ISAs allow OS to decide on the page table structure and hence page walks in such case are software based.
- TLBs are architected and special instructions are available to update them.
- A TLB miss results in an OS trap and the page table information is used by the OS to update the TLB.
- Recent RISC ISAs use architected TLBs

Virtualizing architected TLBs

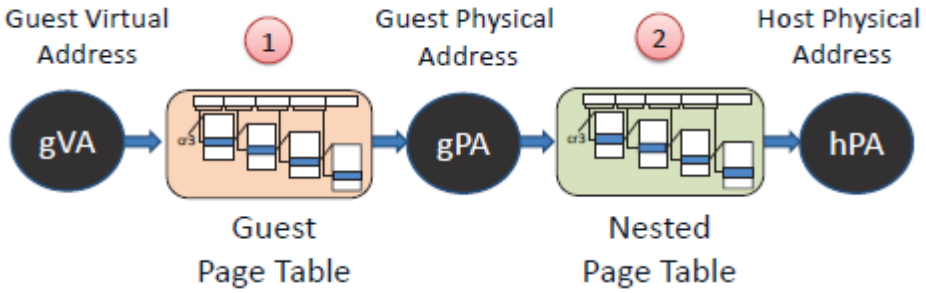
- The VMM manages each VM's TLB by maintaining a copy and also the physical TLB.
- Any instruction that modify the TLB are sensitive and trap to VMM.
- When a VM is activated the VMM copies the virtual TLB's entries into the physical TLB after appropriately translating the VM's PA to Host's PA.

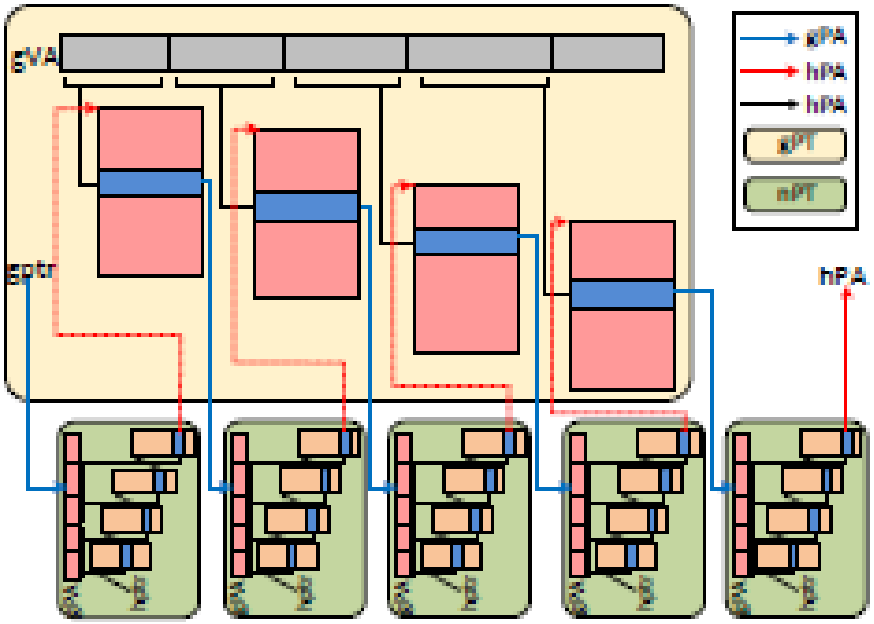
Virtualizing an Architected TLB



- VMM needs to virtualize the TLB when the ISA provides a software managed TLB (SPARC ISA).
- VMM maintains a copy of the VM's TLB contents and manages it.
  - VMM copies the Guest OS specific TLB contents whenever the VM is activated (has an issue?).
  - Each entry in TLB is associated with an ASID (address space identifier) and the TLB can host multiple VM address mappings.
- VMM copies the VM's TLB contents to the real TLB after appropriately translating the GPA->HPA.

Hardware support for memory virtualization – Nested Page Tables





- Nested page tables need hardware support and enables memory virtualization
- Processor has two page table pointers to complete a page table translation
  - One points to the guest page table ***gptr***
  - Other points to the host page table ***hptr***
- Guest page table holds the translation for GVA→GPA
- Host page table holds GPA→HPA translation

I/O Device Virtualization

- The complicated part of system virtualization is the Input/Output device virtualization.
- No standard interface; each device has it's own intricacies with regard to control and access.
- A general purpose OS has support for a large and a variety of I/O devices; number of different I/O devices also is growing.
- Unlike the processor and memory, I/O devices are oblivious of sharing and concurrency; OS supports these features by way of OS level resource abstractions.

Virtualizing I/O Activity

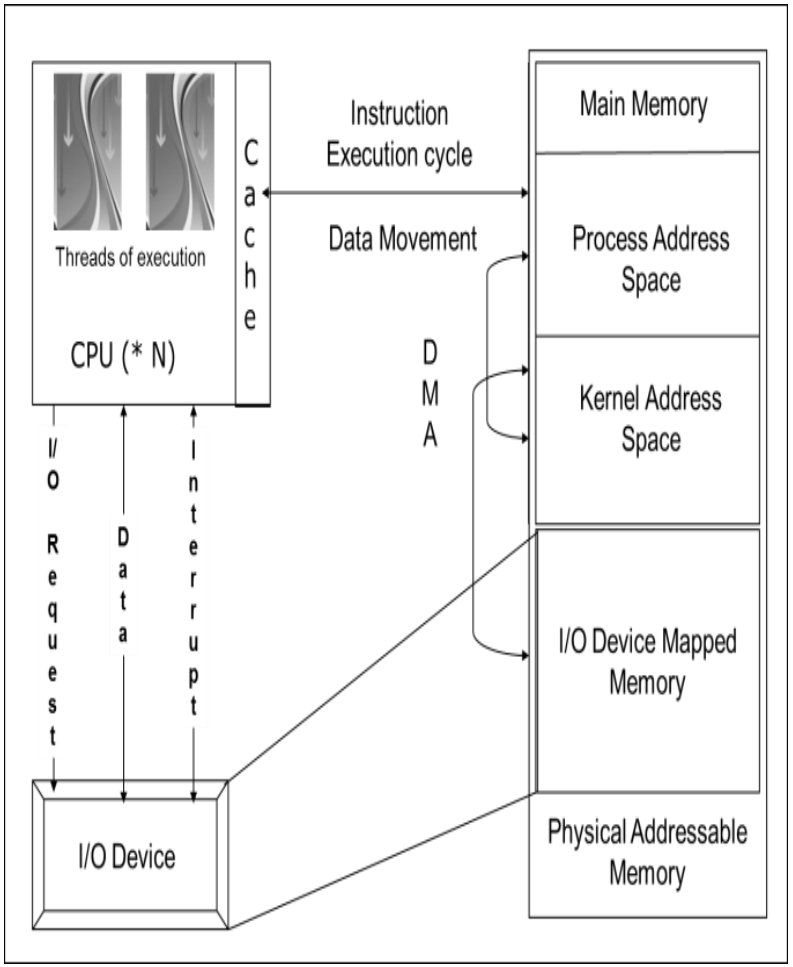
I/O Instruction Level

- I/O devices communicate with the processors using special PIO instructions or using memory mapping feature.
- All PIO instructions are privileged so execute in privileged mode and hence trap to VMM when executed in GuestOS.
- Execution of PIO instruction by the GuestOS needs support at the VMM end in terms of address translation.
- Issues also with bulk execution of I/O requests and from various VMs.
- Normally adopted in device emulation modes of virtualization
- Has high performance overheads.

Device Driver Level

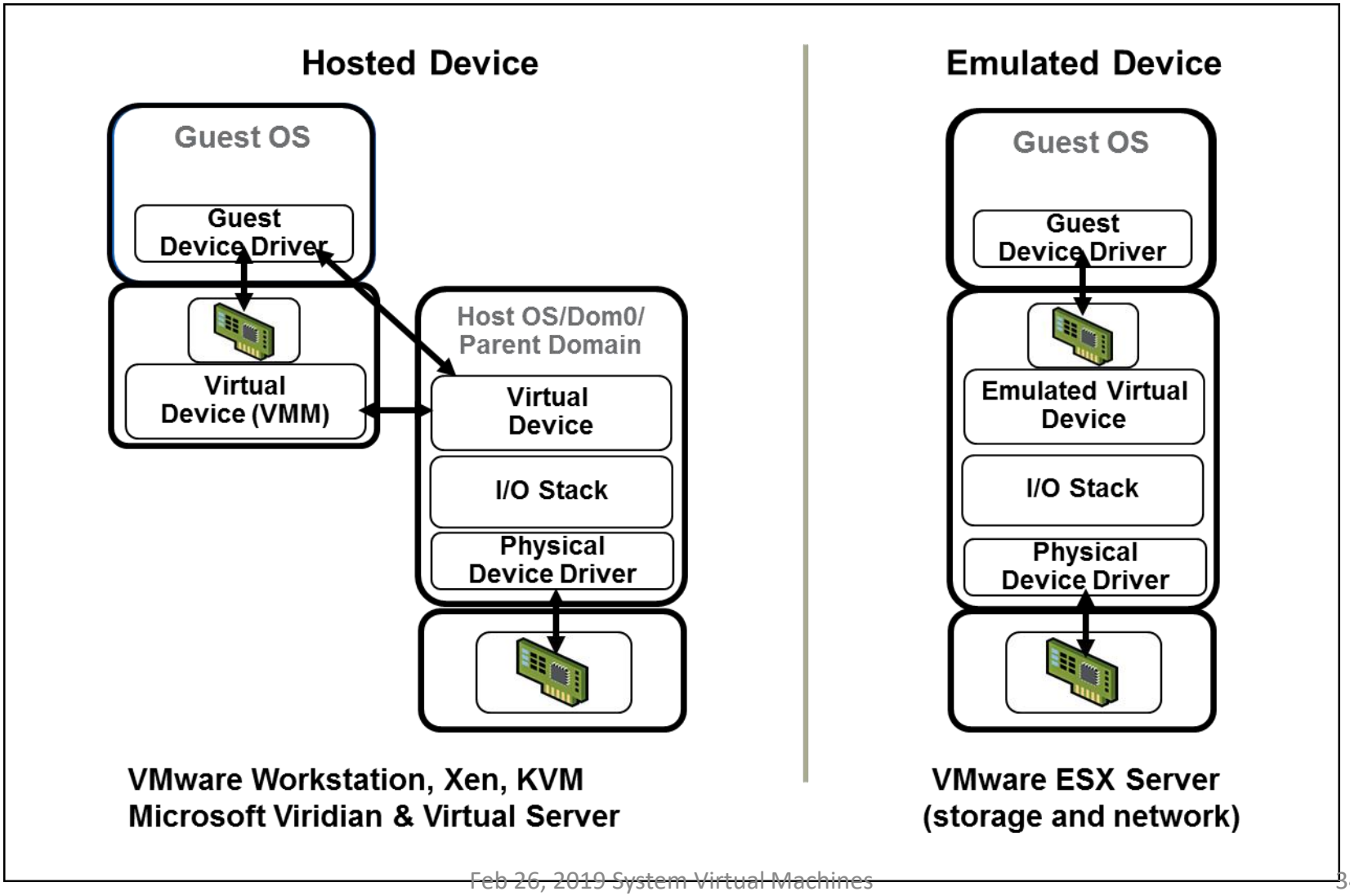
- Device drivers abstract out the device specific I\O instructions and provide the interface to support OS level resource abstractions.
- Virtual device abstractions are created in the GuestOS which then connect to the physical device drivers for execution of PIO instructions.
- Commonly used techniques are emulation and para-virtualization with front-end and backend pairs.
- Front-end drivers are resident in the GuestOS, back-end drivers reside in the hypervisor or the hostOS.

I/O Devices - Recap



- In most general purpose OS's I/O devices are accessed using the system call interface.
- The OS has different abstractions for different devices, viz files for disk storage, sockets for network interfaces, terminals for display devices, etc.
- Most I/O devices today using memory mapping to interface with the system.

Virtualizing Devices



Performance side-effects of VMs

- **VM setup**: Extra time is involved in setting up the appropriate registers, program counters and timing facilities before a VM can be activated.
- **Emulation/para-virtualization**: Privileged instructions need to be emulated or para-virtualized by the VMM. This leads to higher time spent on executing such instructions.
- **Interrupt handling**: All interrupts are intercepted by the VMM before being passed on to the GuestOS of a VM.
- **State saving**: For every VMM entry, a VM's state needs to be saved to enable control transfer.
- **Bookkeeping**: VMM has to perform special operations to reflect equivalent behavior of that of a real machine.
- **Time elongation**: Some instructions, particularly those involved with memory management, require more processing time because of the requirement to walkthrough multiple structures for the same operation.

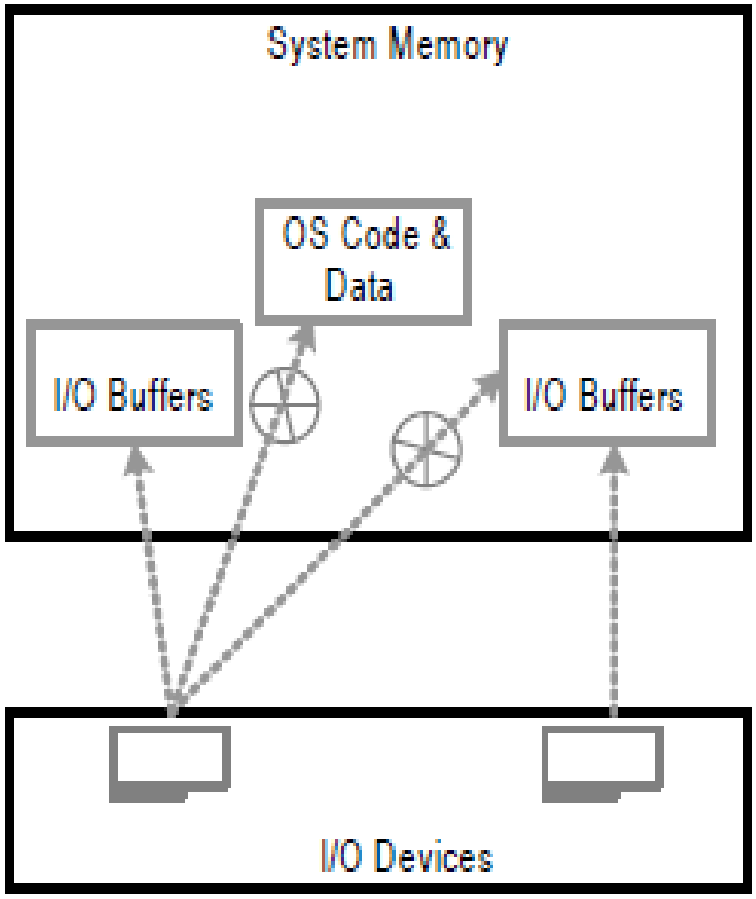
Virtual Machine Assists

- Hardware extensions or support to improve performance of applications when executing inside a VM are called VM assists.
- Instruction emulation assists:
  - VMM emulates an instruction using a routine whose operation depends on whether the VM is executing in a system mode or user mode.
  - IBM system/370 used a hardware assist that would detect the VM's execution mode while performing emulation.
  - Such assists use the knowledge that the hardware is virtualized!

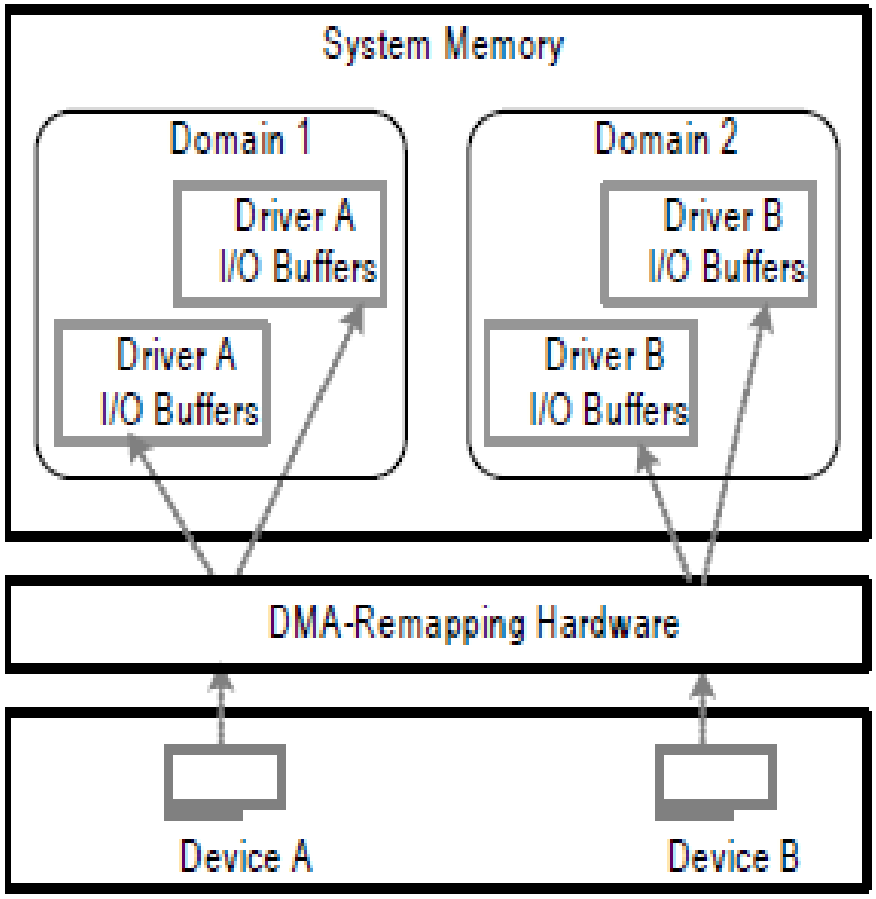
VMM assists

- VMM improvement can potentially benefit all hosted VMs.
  - Hardware assisted context switch between VM and VMM – storing and restoring of machine state registers using hardware
  - Privileged Instructions Decode – Hardware assisted decoding might help improve performance along with software techniques to optimize.
  - Virtual Timers – True implementation of timers in VMs need hardware assistance. Ex. System/370 ISA requires that the virtual timers in a VM be located in a specific place inside the VM's memory and the VMM decrements this timer counter every time a true timer interrupt occurs.
  - Enhanced ISA for VMM support – I/O device assignment, DMA remapping, Interrupt remapping, interrupt posting are some of the evolving list of assists.

OS use of DMA Remapping

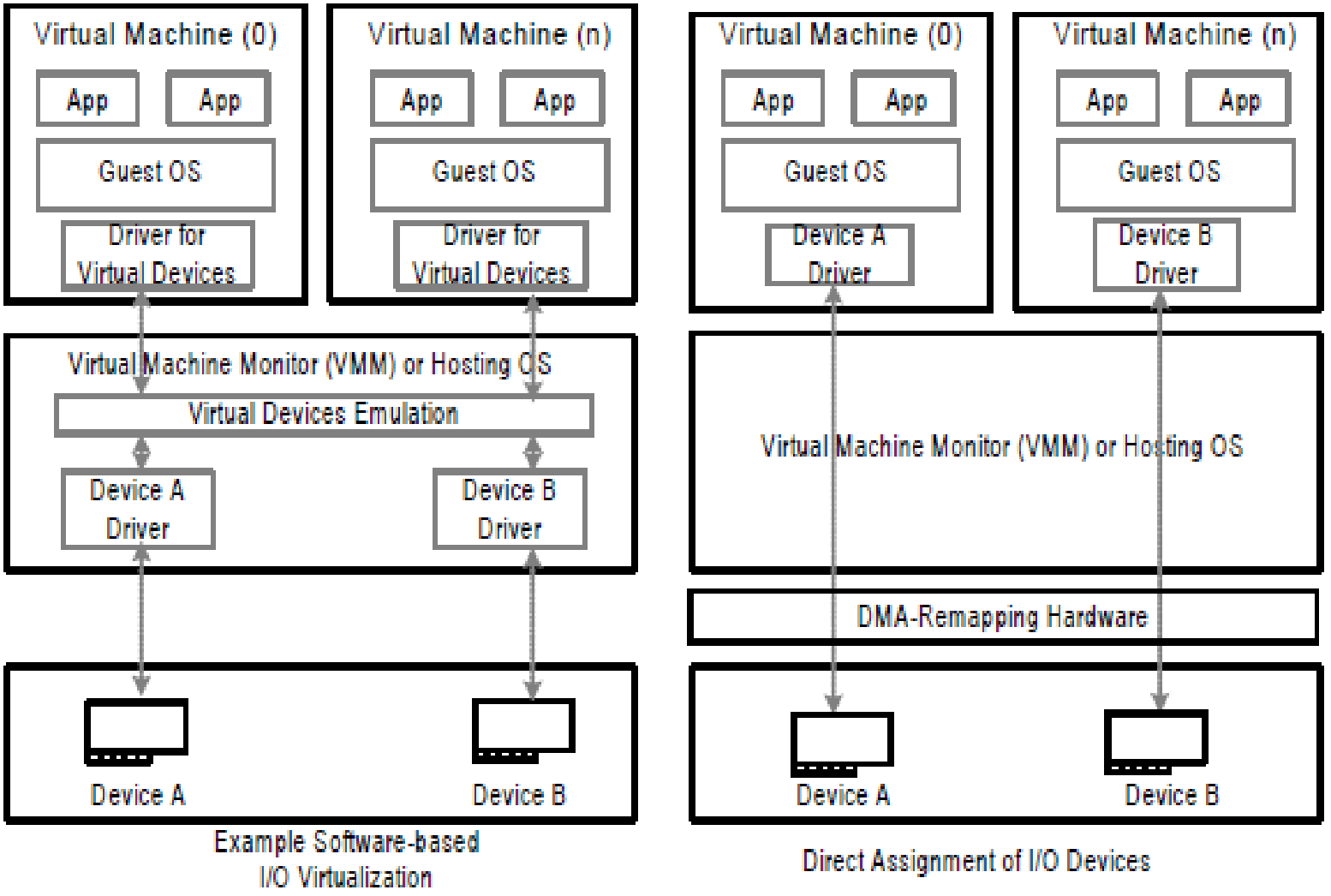


Device DMA without isolation



Device DMA isolated using DMA remapping hardware

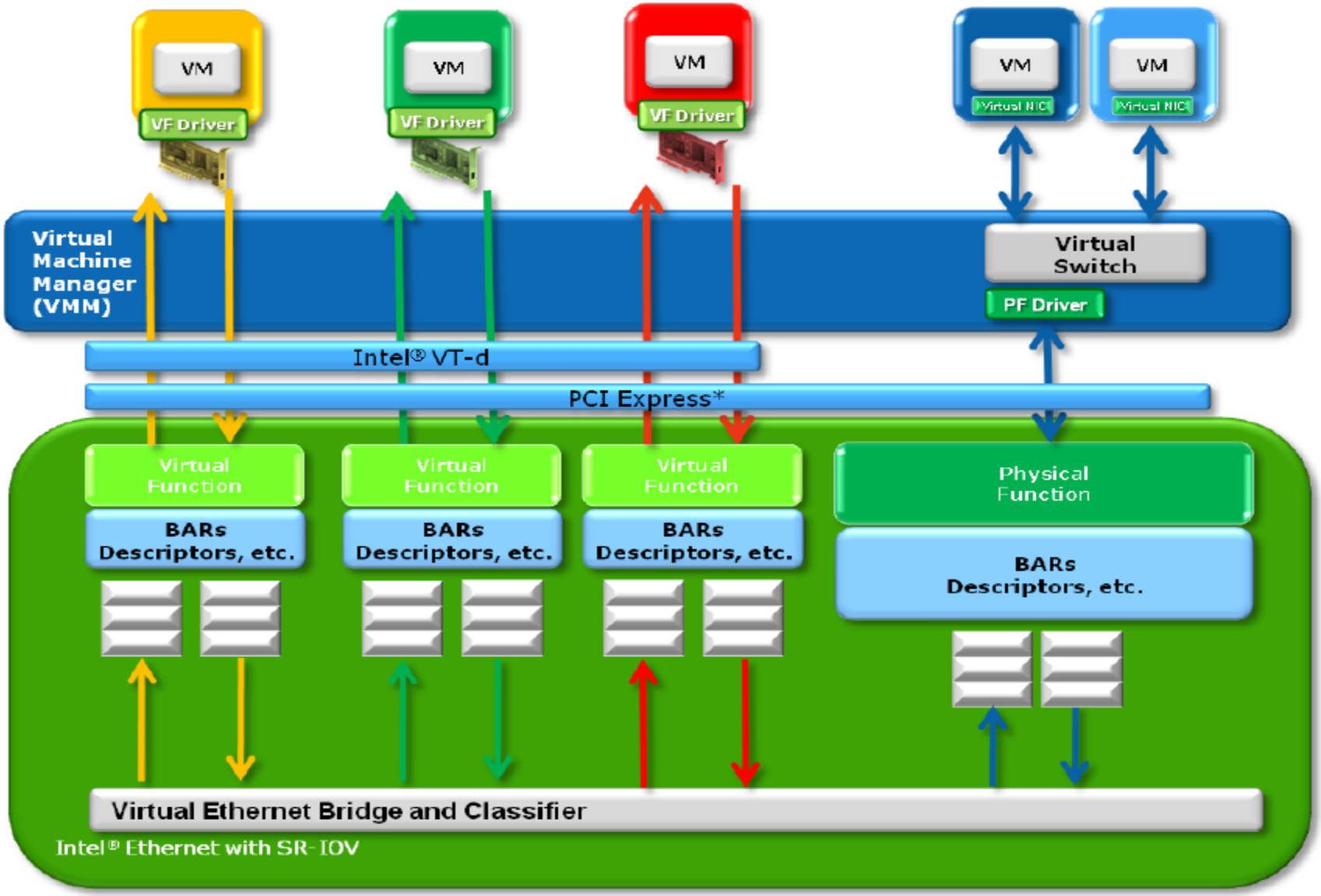
VMM usage of DMA Remapping Direct IO



IO-Virtualization Assists

- Single Root IO Virtualization (SR-IOV)
  - Effort by the PCI-SIG to enable I/O device virtualization ensuring clean isolation interfaces
  - Isolation of
    - Device memory
    - I/O streams
    - I/O interrupts
    - Control and I/O operations
    - Errors

SR-IOV NIC Example



Multi-Processor Virtualization

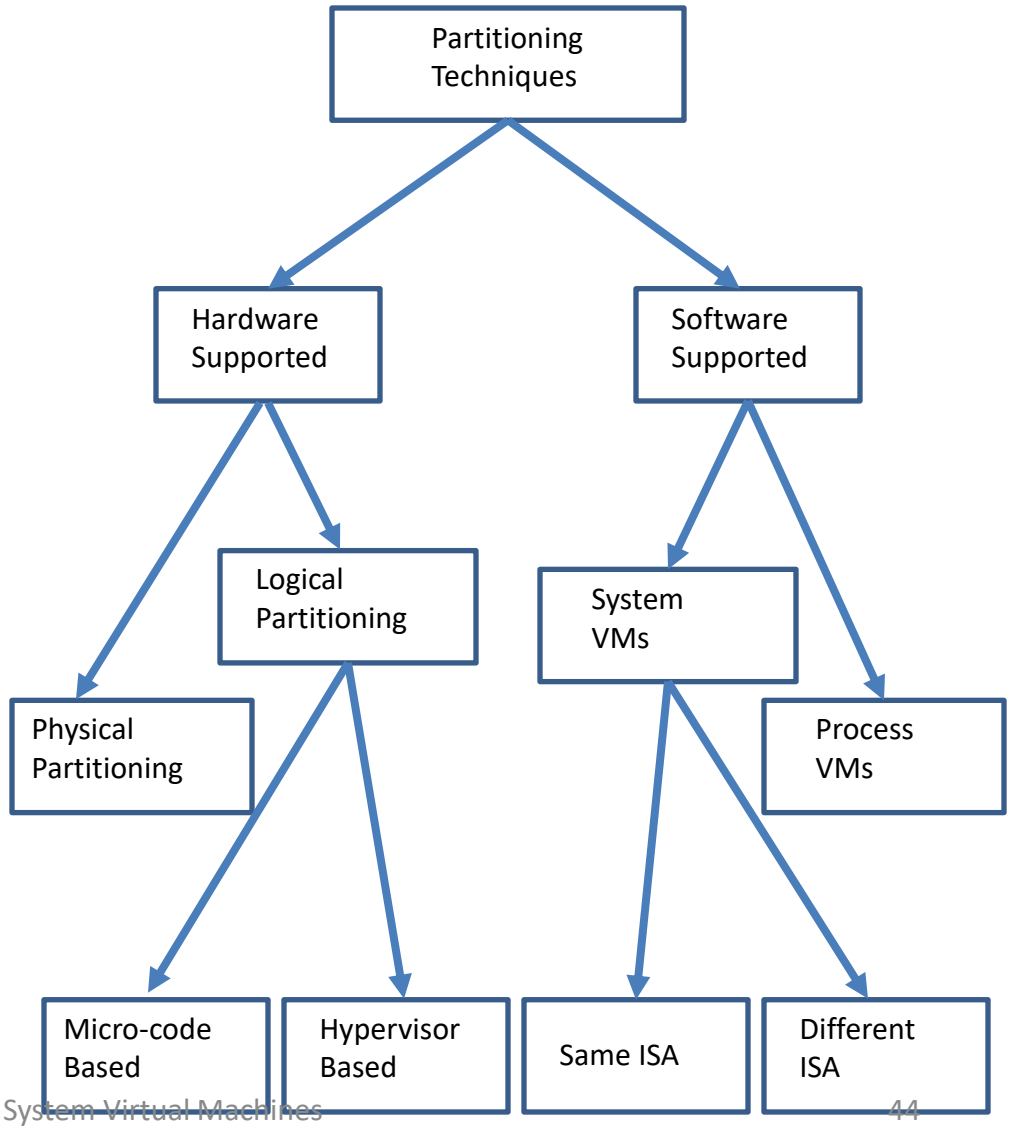
- Multi-core and many-core architectures have higher number of processors that share memory and I/O devices.
- Techniques for system isolation
  - Dynamic partitioning
    - Time-sharing
  - Static partitioning
    - Space-sharing
- When does static partitioning help?

Prerequisite

- https://classes.soe.ucsc.edu/cmps111/Fall02/Chapter08.pdf
- Appendix A section 7 of Virtual Machines, Smith & Nair

Partitioning Techniques

- Physical Partitioning
  - Failure Isolation
  - Security Isolation
  - Oriented towards specific system level objectives
- Logical Partitioning
  - Flexible resource sharing
  - Improved utilization
  - Fault Tolerant



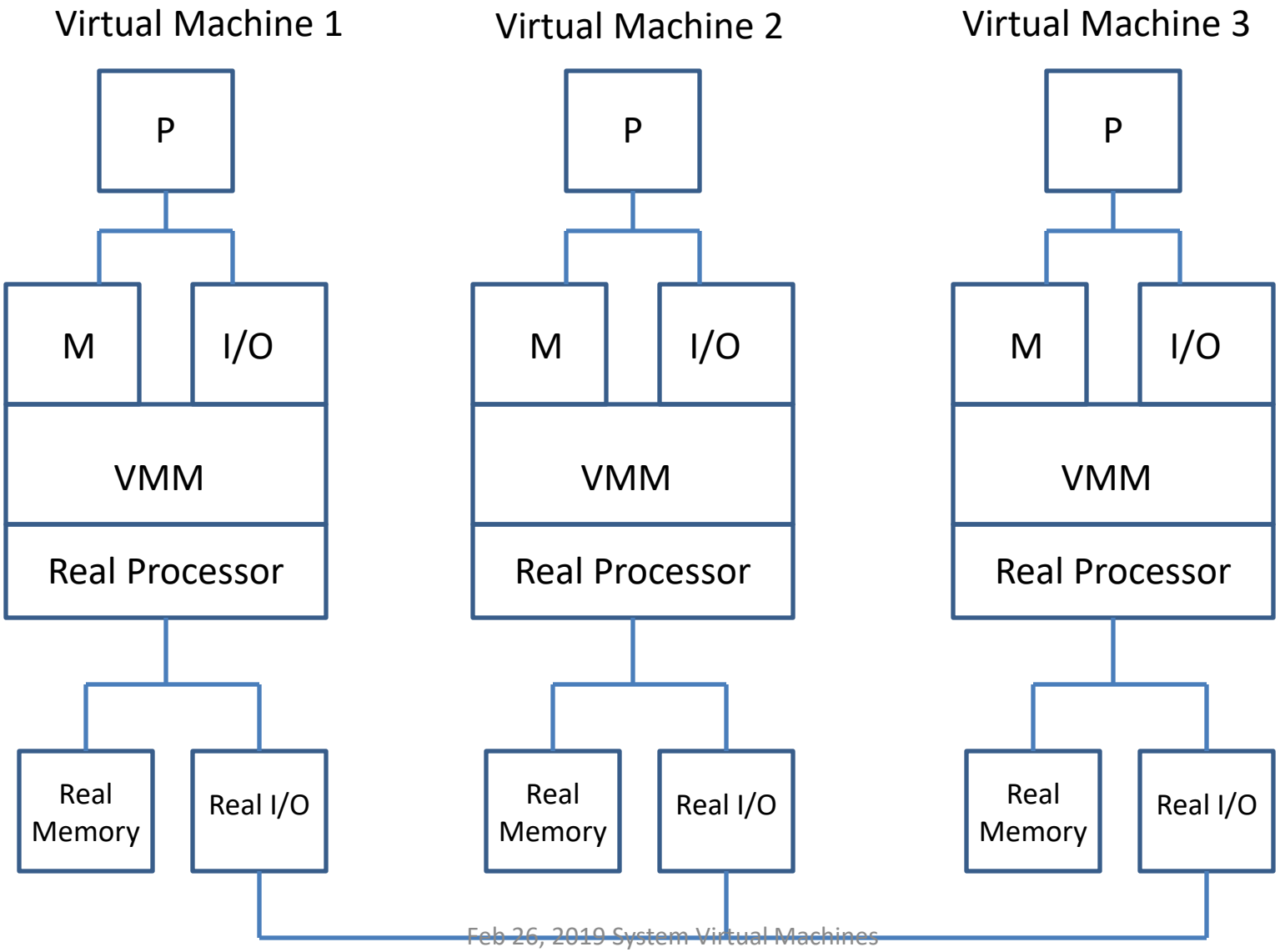
Case Study

- Micro-code based Logical Partitioning:
  - Extended ISA provides the necessary support in hardware for partitioning
    - IBM System/390 LPAR
- Hypervisor based Logical Partitioning
  - Hardware support for extra supervisory level allows hypervisors (system software layer) to execute under privileged mode that is different from privileged mode of the GuestOS. GuestOS still executes under privileged mode and its applications execute in user mode.
    - HP Superdome servers
    - IBM Dynamic LPAR
    - Cellular Disco

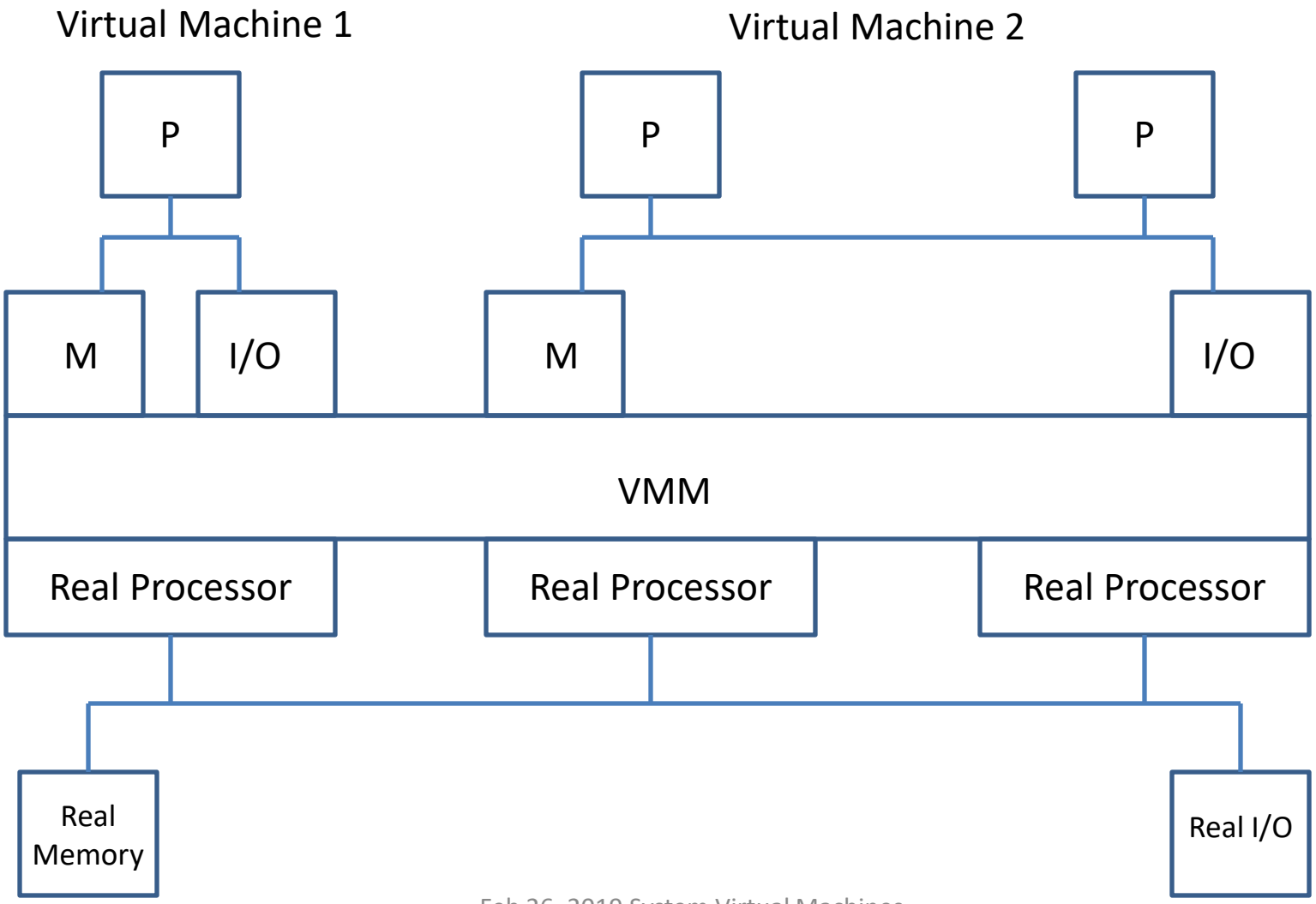
Virtualization: Different Host and Guest ISAs

- What could be the reasons for supporting this model of virtualization?
- Additional complexities to handle:
  - Instructions of target ISA must be transparently and dynamically translated to the host ISA.
  - Memory model of the target ISA must be observed in the VM so the VMM/hypervisor needs to handle the memory ordering and coherence rules of the target ISA

Virtualizing Uniprocessor Cluster



Virtualizing SMP Host



Concerns with multi-core systems for virtualizing

- Concurrency capability of the hypervisor or host-OS can cause bottlenecks for consolidation
  - Serial data structures of the kernel
  - Increased resource usage (CPU cycles/Memory)
  - Serial interrupt and device drivers
- Mitigation strategies
  - Concurrent device design with device interrupt delivery to any core
  - Concurrent kernel design
  - Assigning specific cores to kernel services
  - Scheduling VMs on the same cores even without affinity assignment

Co-designed Virtual Machines

- Co-design exploration space:
  - To maintain code portability and backward compatibility, ISA supported on hardware evolves in restricted sense when compared to hardware that executes its and software that uses it!
  - Virtual Machines offer an opportunity to explore new ISA features using a co-design approach.
  - Host ISA is designed keeping in view the Virtual Machine that runs on it.
  - Target ISA uses a software/hardware approach to support the host ISA dynamically keeping performance, efficiency and power as the target goals.
  - Mostly restricted to processor virtualization.
  - Currently has been restricted to research interest and not explored for workload consolidation.

Examples of Co-Designed VMs

- Transmeta Crusoe TM5000 series Processor: Implements Intel IA-32 to proprietary VLIW ISA mapping using dynamic binary translation and code cache.
- IBM AS/400 Systems:
  - Full system developed with a co-design perspective
  - Host ISA is built using a high-level instruction set called Machine Interface (MI)
  - Implementing the MI is a set of standard libraries called Licensed Internal Code (LIC) that dealt with implementation specific resource management.
  - Initially AS/400 systems were built on a proprietary VLIW ISA and later evolved over to PowerPC ISA.
- Exploring para-virtualization for co-design is a fertile ground for innovation!

Summary

- System Virtual Machines:
  - Requirements for virtualizability
- Basic concepts
  - User Interface and Appearance
  - State Management
  - Resource Control
  - Bare Metal and Hosted Virtual Machines
  - Co-designed Virtual Machines
- Case-Studies (To be part of student seminars)
  - Native Virtual Machines: Xen, Vmware-Esxi
  - Hosted Virtual Machines: Vmware-Workstation, Palazzo, Linux-KVM