# Introduction to Virtual Machines

Carl Waldspurger (SB SM ’89 PhD ’95)

VMware R&D

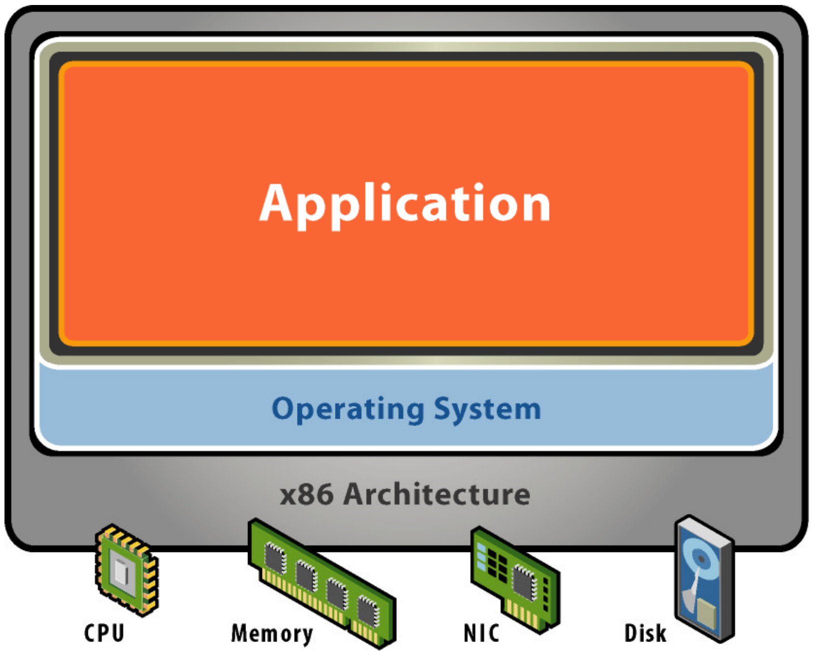
# Overview

- Virtualization and VMs
- Processor Virtualization
- Memory Virtualization
- I/O Virtualization

# Types of Virtualization

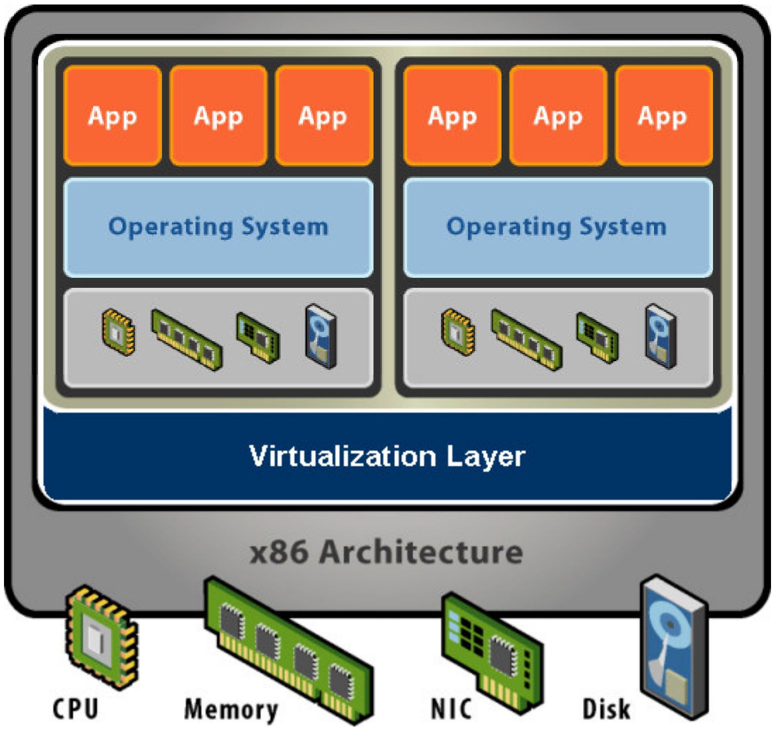
- Process Virtualization
  - OS-level  processes, Solaris Zones, BSD Jails, Virtuozzo
  - Language-level  Java, .NET, Smalltalk
  - Cross-ISA emulation  Apple 68K-PPC-x86, Digital FX!32
- Device Virtualization
  - Logical vs. physical  VLAN, VPN, NPIV, LUN, RAID
- **System Virtualization**
  - "Hosted"  VMware Workstation, Microsoft VPC, Parallels
  - "Bare metal"  VMware ESX, Xen, Microsoft Hyper-V

# Starting Point: A Physical Machine




- Physical Hardware
  - Processors, memory, chipset, I/O devices, etc.
  - Resources often grossly underutilized
- Software
  - Tightly coupled to physical hardware
  - Single active OS instance
  - OS controls hardware

# What is a Virtual Machine?




- Software Abstraction
  - Behaves like hardware
  - Encapsulates all OS and application state
- Virtualization Layer
  - Extra level of indirection
  - Decouples hardware, OS
  - Enforces isolation
  - Multiplexes physical hardware across VMs

# Virtualization Properties

- Isolation
  - Fault isolation
  - Performance isolation
- Encapsulation
  - Cleanly capture all VM state
  - Enables VM snapshots, clones
- Portability
  - Independent of physical hardware
  - Enables migration of live, running VMs
- Interposition
  - Transformations on instructions, memory, I/O
  - Enables transparent resource overcommitment, encryption, compression, replication …

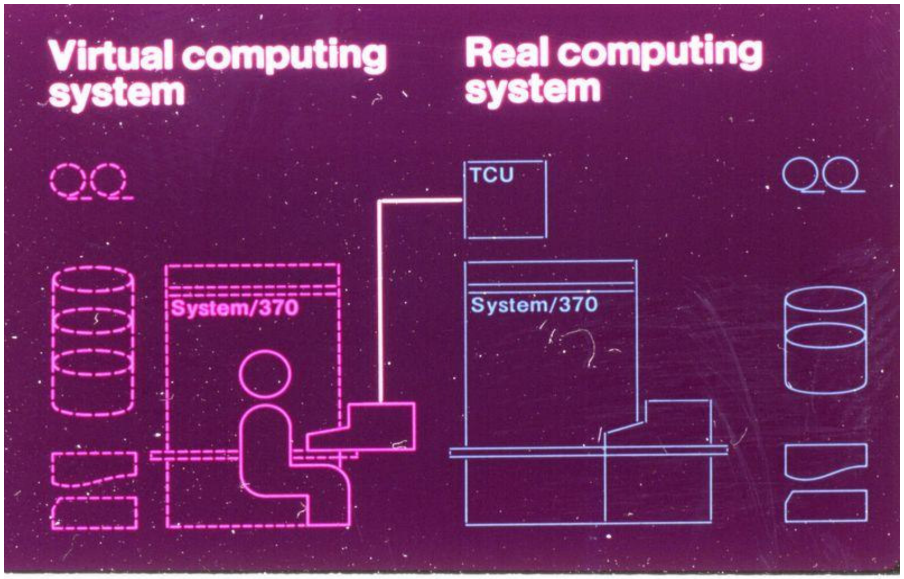
# What is a Virtual Machine Monitor?

- Classic Definition (Popek and Goldberg ’74)

> 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.

- VMM Properties
  - Fidelity
  - Performance
  - Safety and Isolation

# Classic Virtualization and Applications




From IBM VM/370 product announcement, *ca*. 1972

- Classical VMM
  - IBM mainframes: IBM S/360, IBM VM/370
  - Co-designed proprietary hardware, OS, VMM
  - "Trap and emulate" model
- Applications
  - Timeshare several single-user OS instances on expensive hardware
  - Compatibility

# Modern Virtualization Renaissance

- Recent Proliferation of VMs
  - Considered exotic mainframe technology in 90s
  - Now pervasive in datacenters and clouds
  - Huge commercial success
- Why?
  - Introduction on commodity x86 hardware
  - Ability to “do more with less” saves $$$
  - Innovative new capabilities
  - Extremely versatile technology

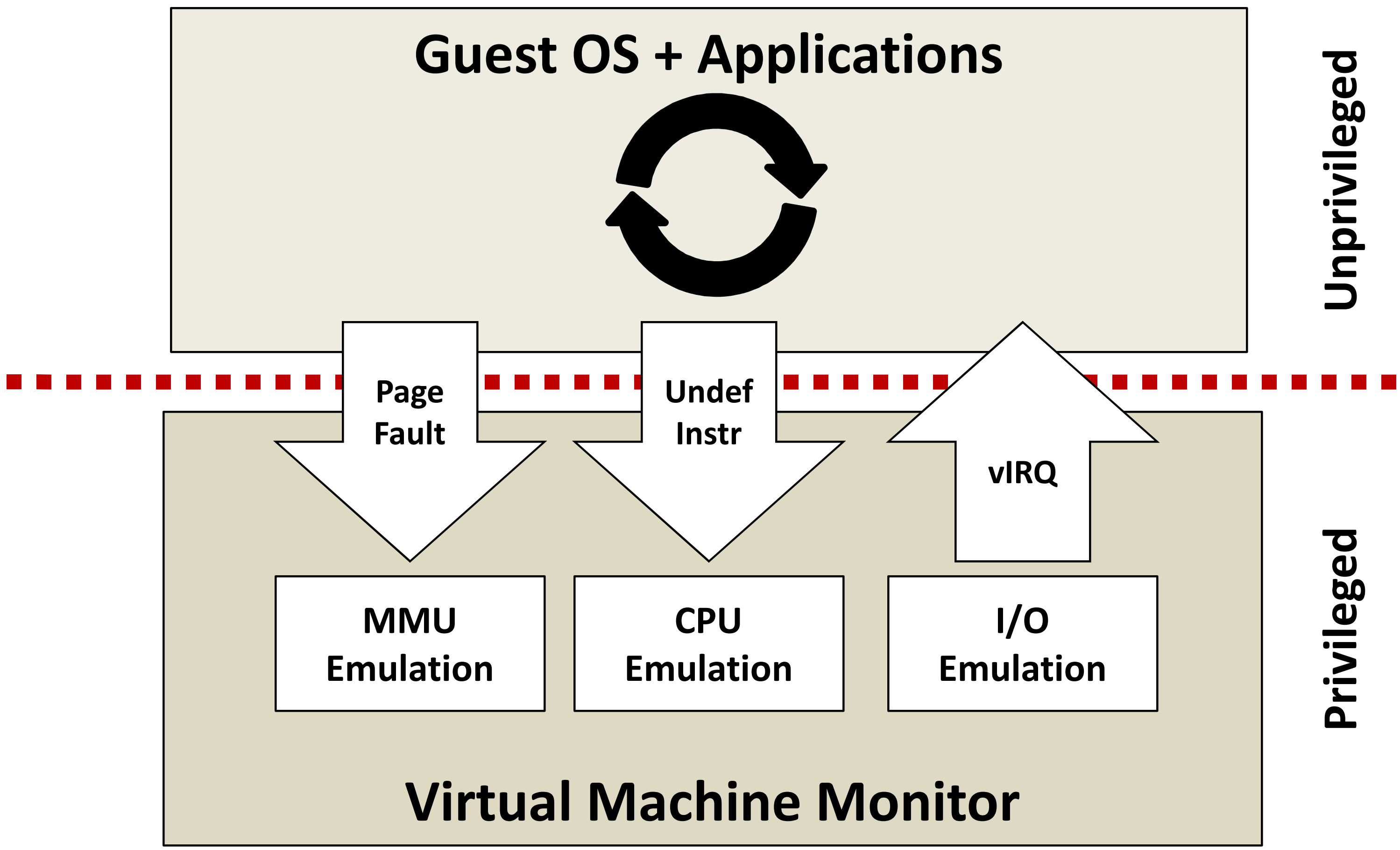
# Modern Virtualization Applications

- Server Consolidation
  - Convert underutilized servers to VMs
  - Significant cost savings (equipment, space, power)
  - Increasingly used for virtual desktops
- Simplified Management
  - Datacenter provisioning and monitoring
  - Dynamic load balancing
- Improved Availability
  - Automatic restart
  - Fault tolerance
  - Disaster recovery
- Test and Development

# Processor Virtualization

- Trap and Emulate
- Binary Translation

# Trap and Emulate



# "Strictly Virtualizable"

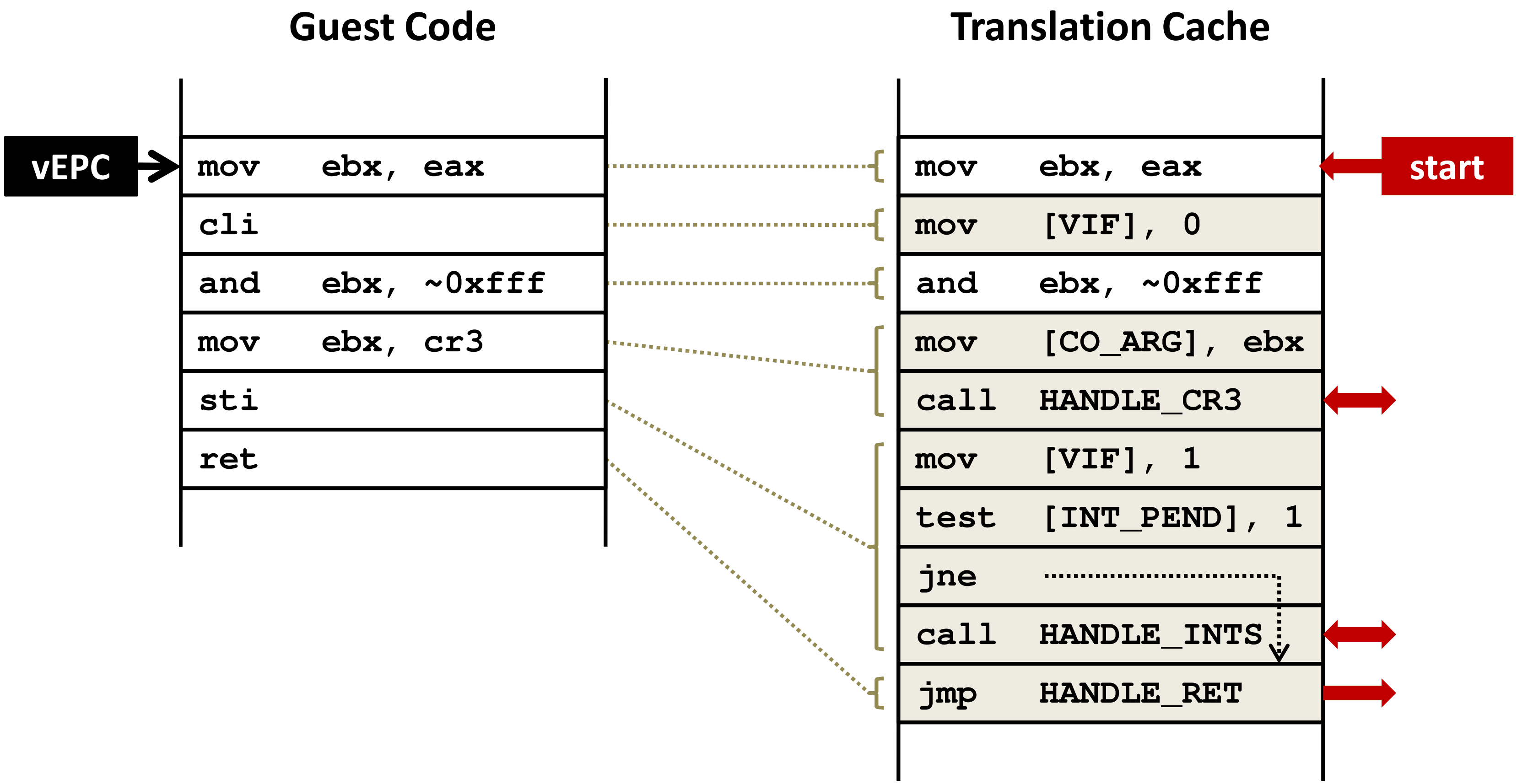
A processor or mode of a processor is *strictly virtualizable* if, when executed in a lesser privileged mode:

- all instructions that access privileged state trap
- all instructions either trap or execute identically

# Issues with Trap and Emulate

- Not all architectures support it
- Trap costs may be high
- VMM consumes a privilege level
  - Need to virtualize the protection levels

# Binary Translation



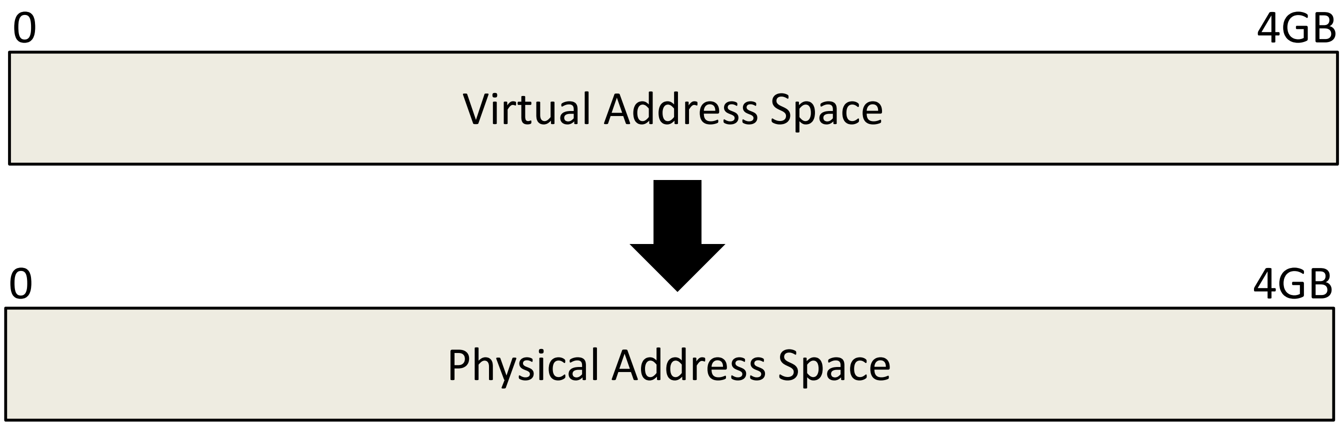
# Issues with Binary Translation

- Translation cache management
- PC synchronization on interrupts
- Self-modifying code
  - Notified on writes to translated guest code
- Protecting VMM from guest

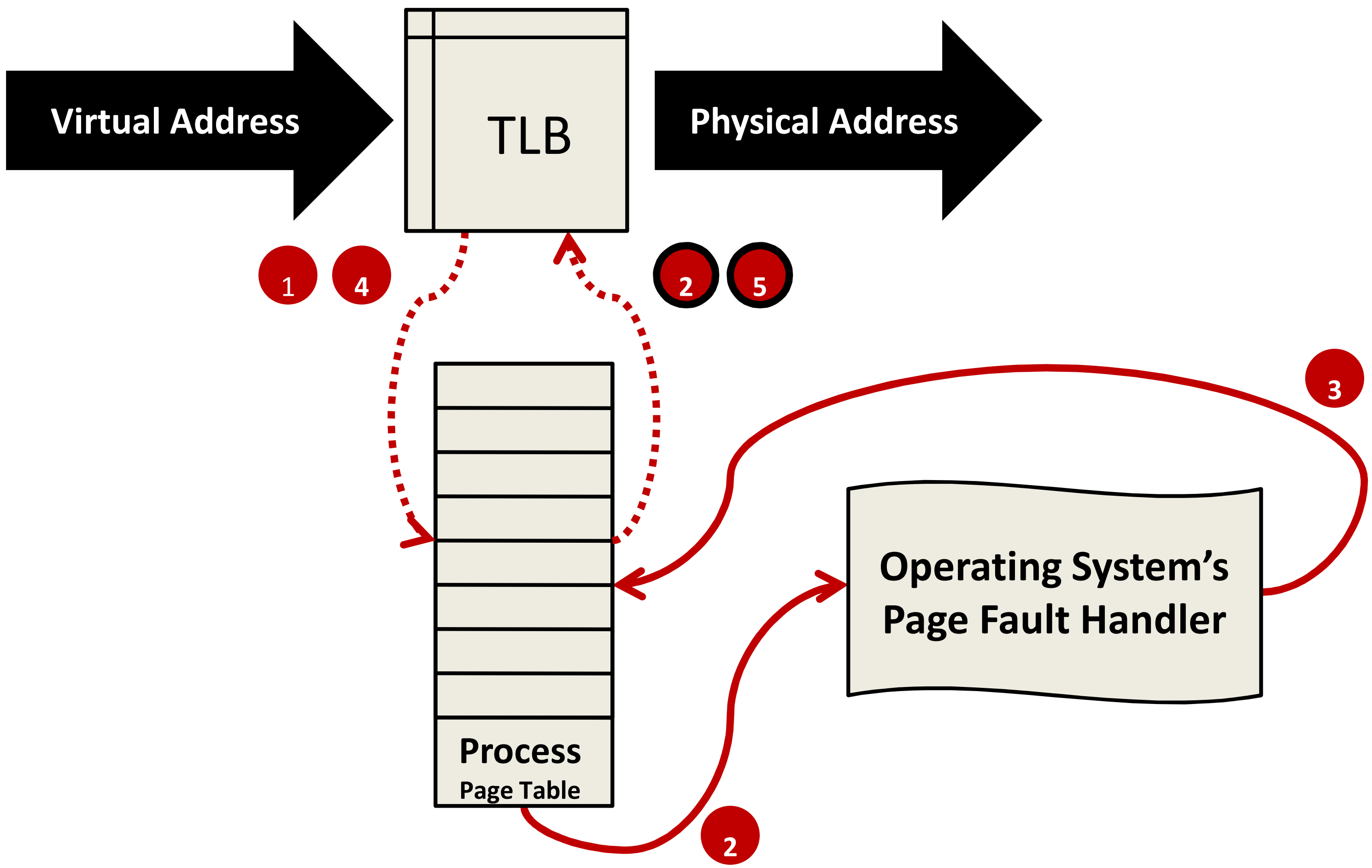
# Memory Virtualization

- Shadow Page Tables
- Nested Page Tables

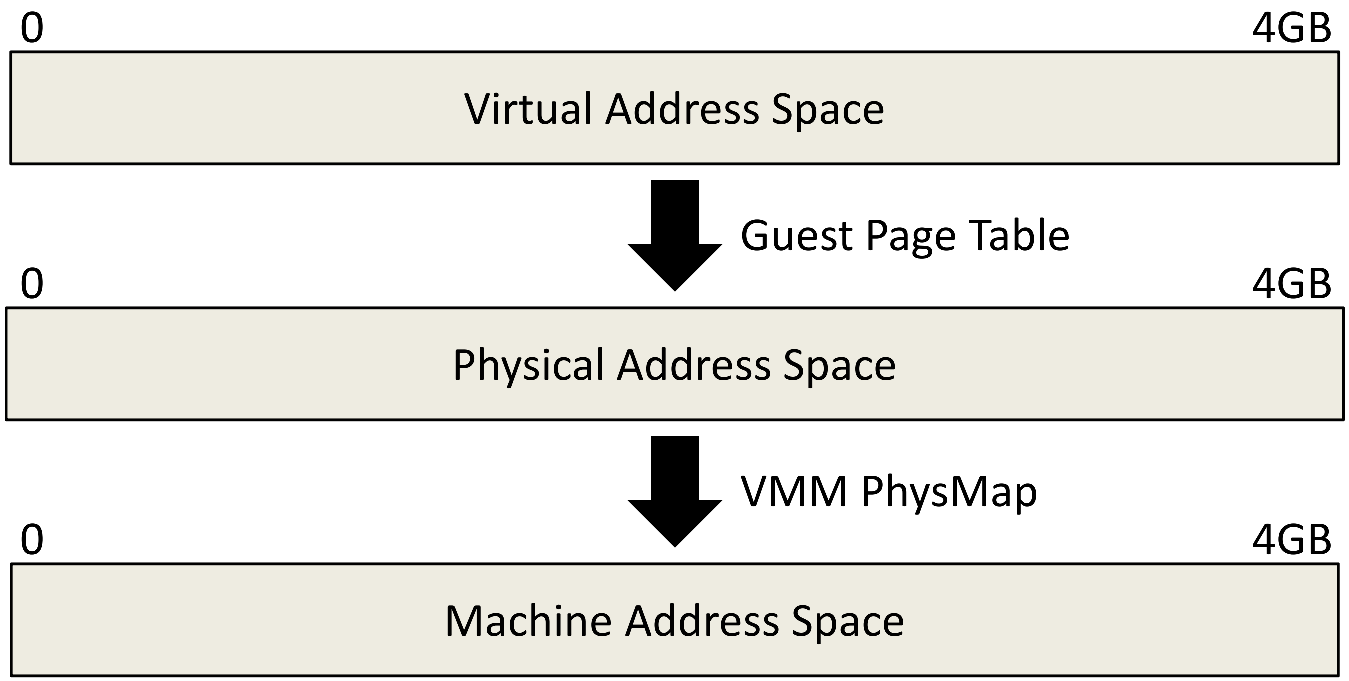
# Traditional Address Spaces



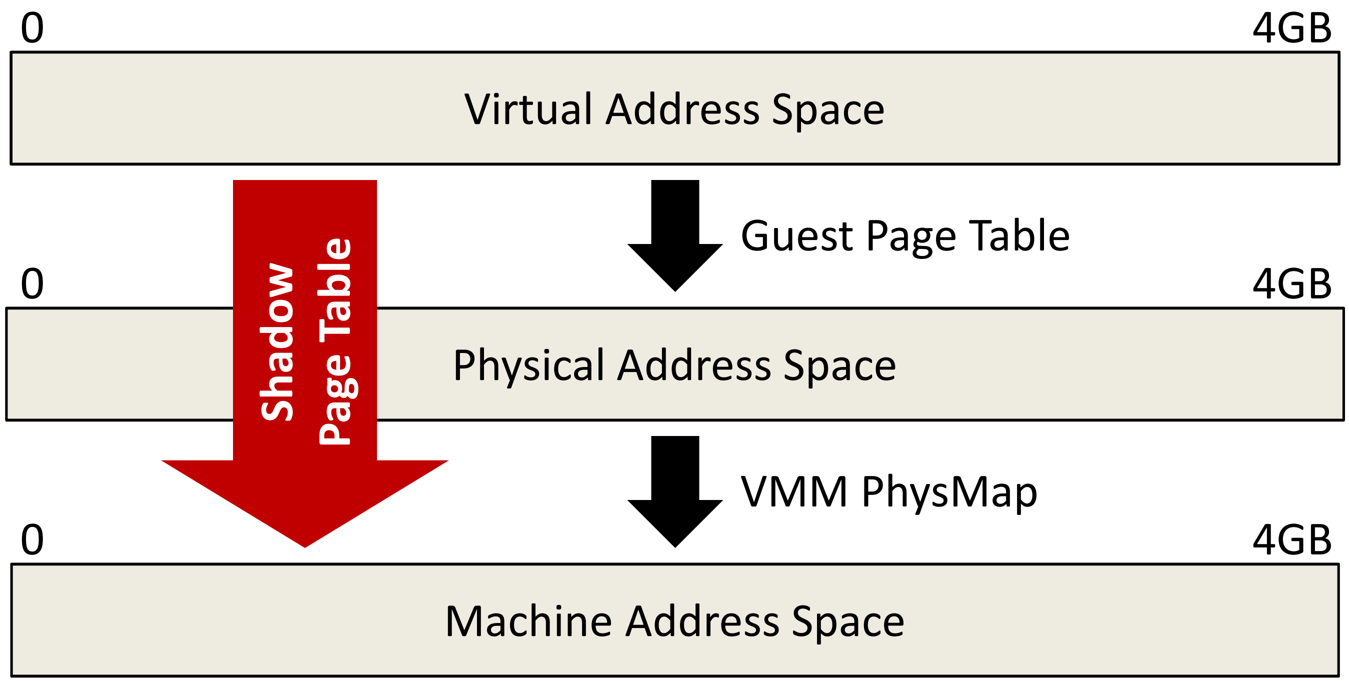
# Traditional Address Translation



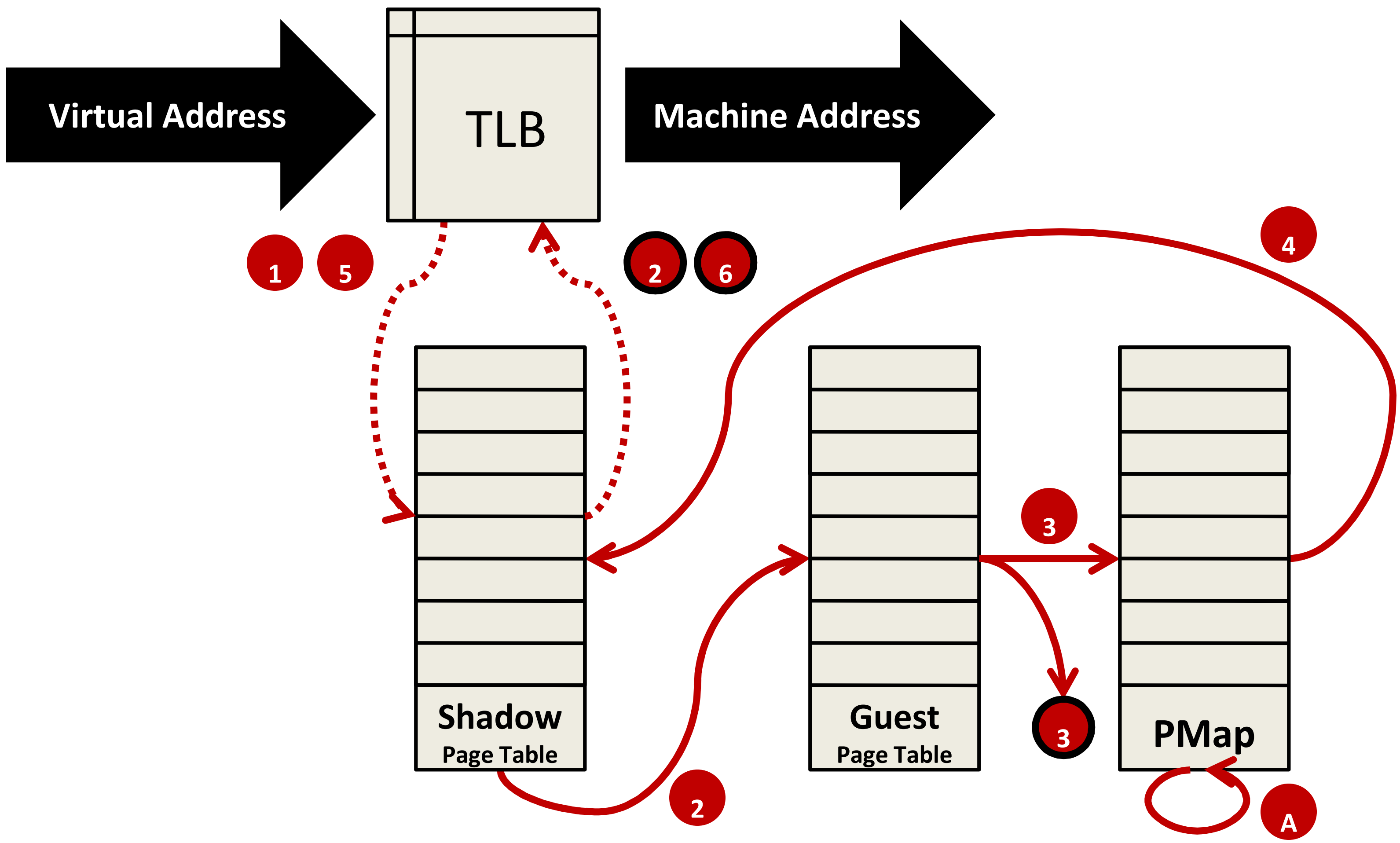
# Virtualized Address Spaces



# Virtualized Address Spaces w/ Shadow Page Tables



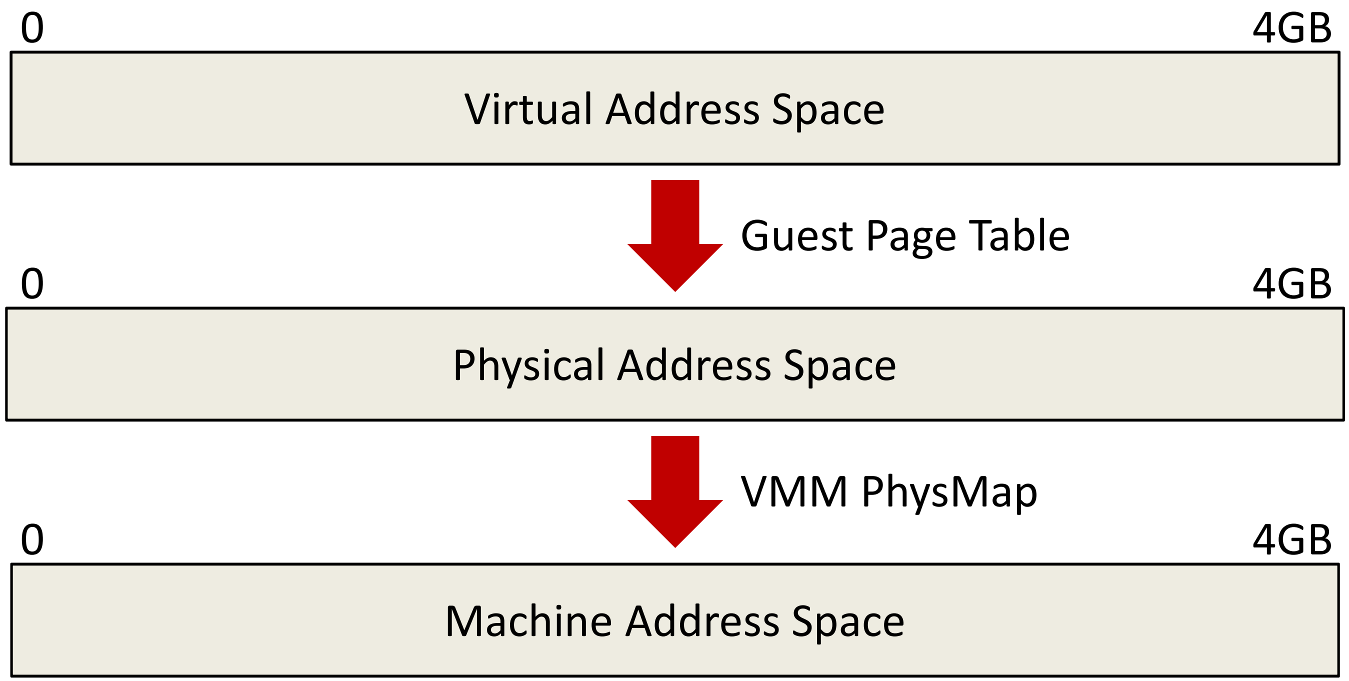
# Virtualized Address Translation w/ Shadow Page Tables



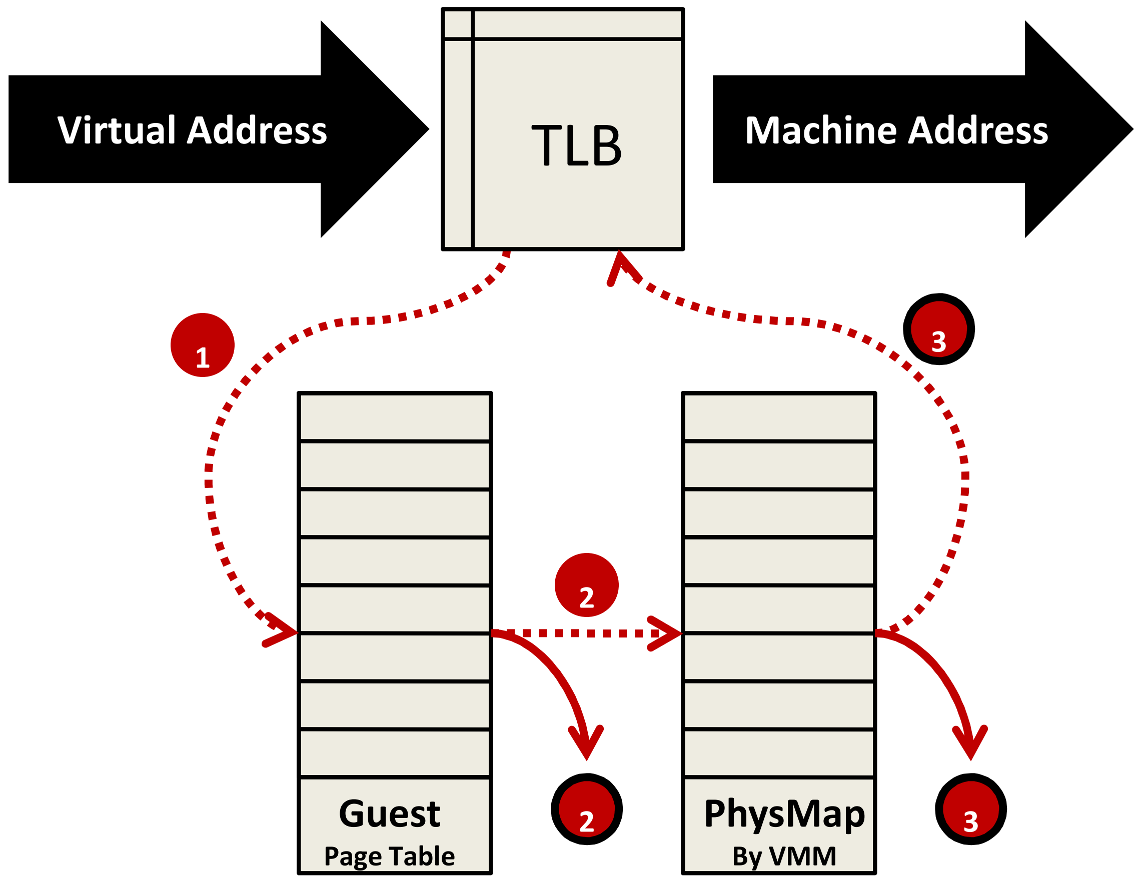
# Issues with Shadow Page Tables

- Guest page table consistency
  - Rely on guest's need to invalidate TLB
- Performance considerations
  - Aggressive shadow page table caching necessary
  - Need to trace writes to cached page tables

# Virtualized Address Spaces w/ Nested Page Tables



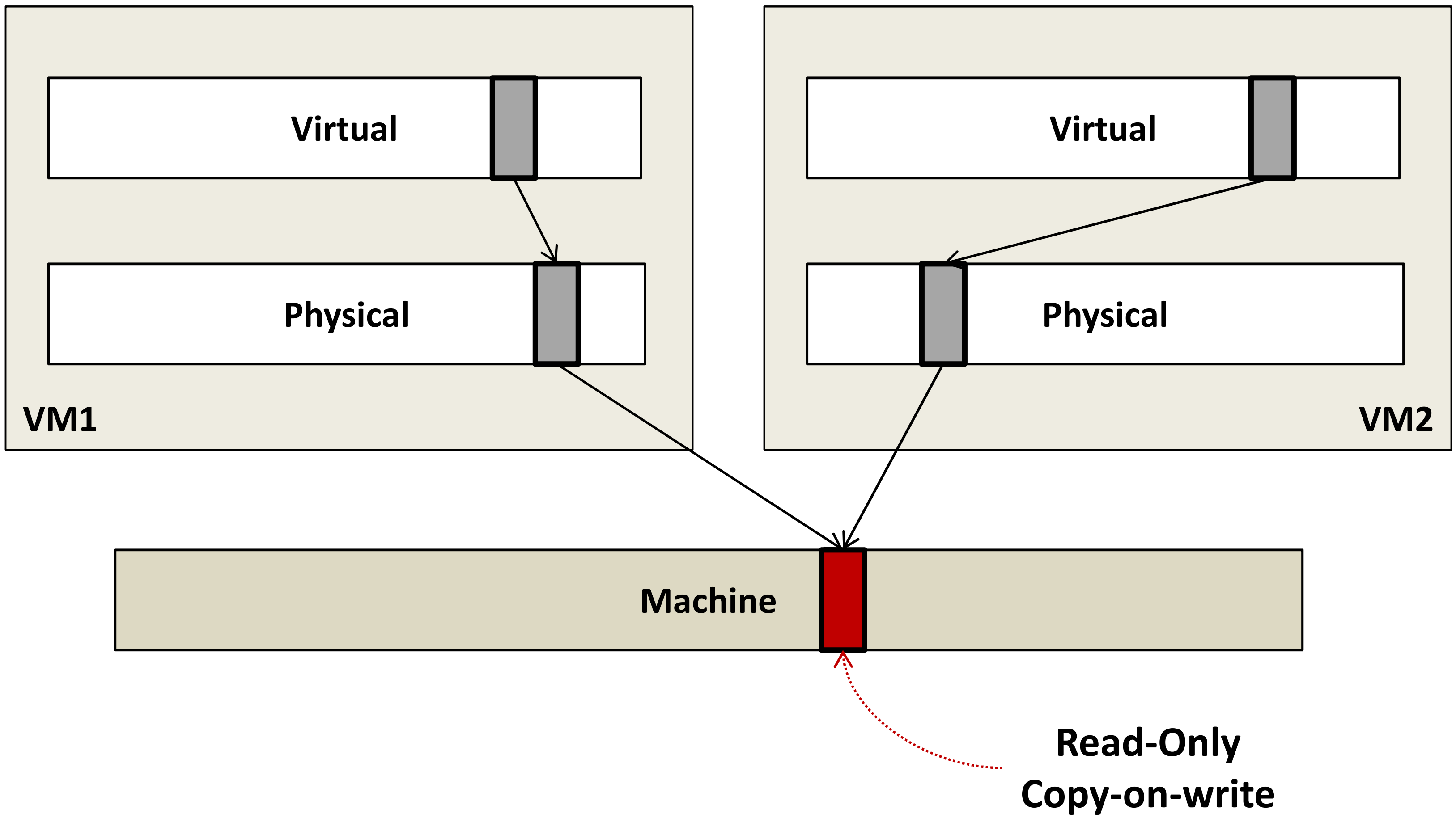
# Virtualized Address Translation w/ Nested Page Tables



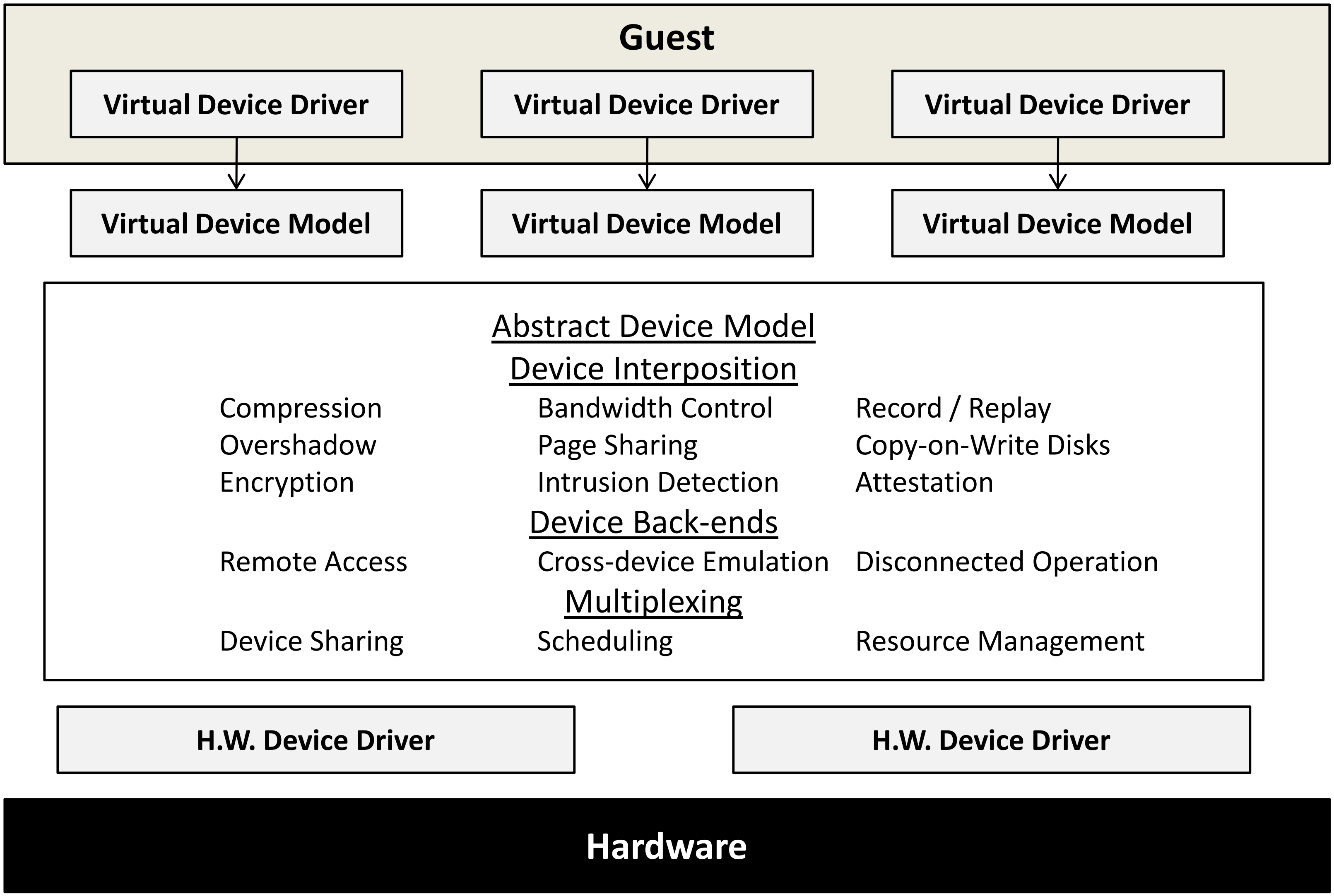
# Issues with Nested Page Tables

- Positives
  - Simplifies monitor design
  - No need for page protection calculus
- Negatives
  - Guest page table is in physical address space
  - Need to walk PhysMap multiple times
    - Need physical-to-machine mapping to walk guest page table
    - Need physical-to-machine mapping for original virtual address
- Other Memory Virtualization Hardware Assists
  - Monitor Mode has its own address space
    - No need to hide the VMM

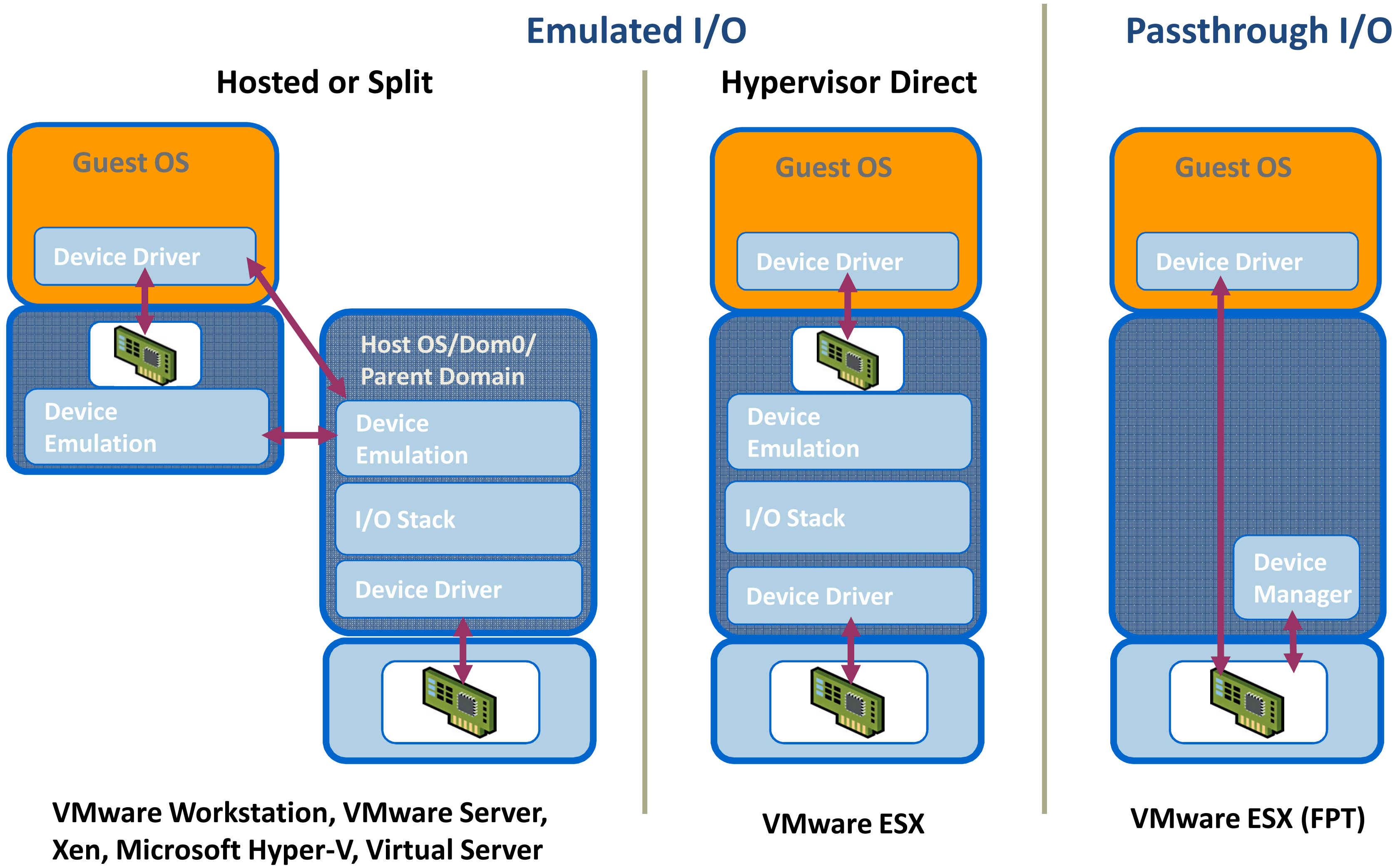
# Interposition with Memory Virtualization
# Page Sharing



# I/O Virtualization



# I/O Virtualization Implementations

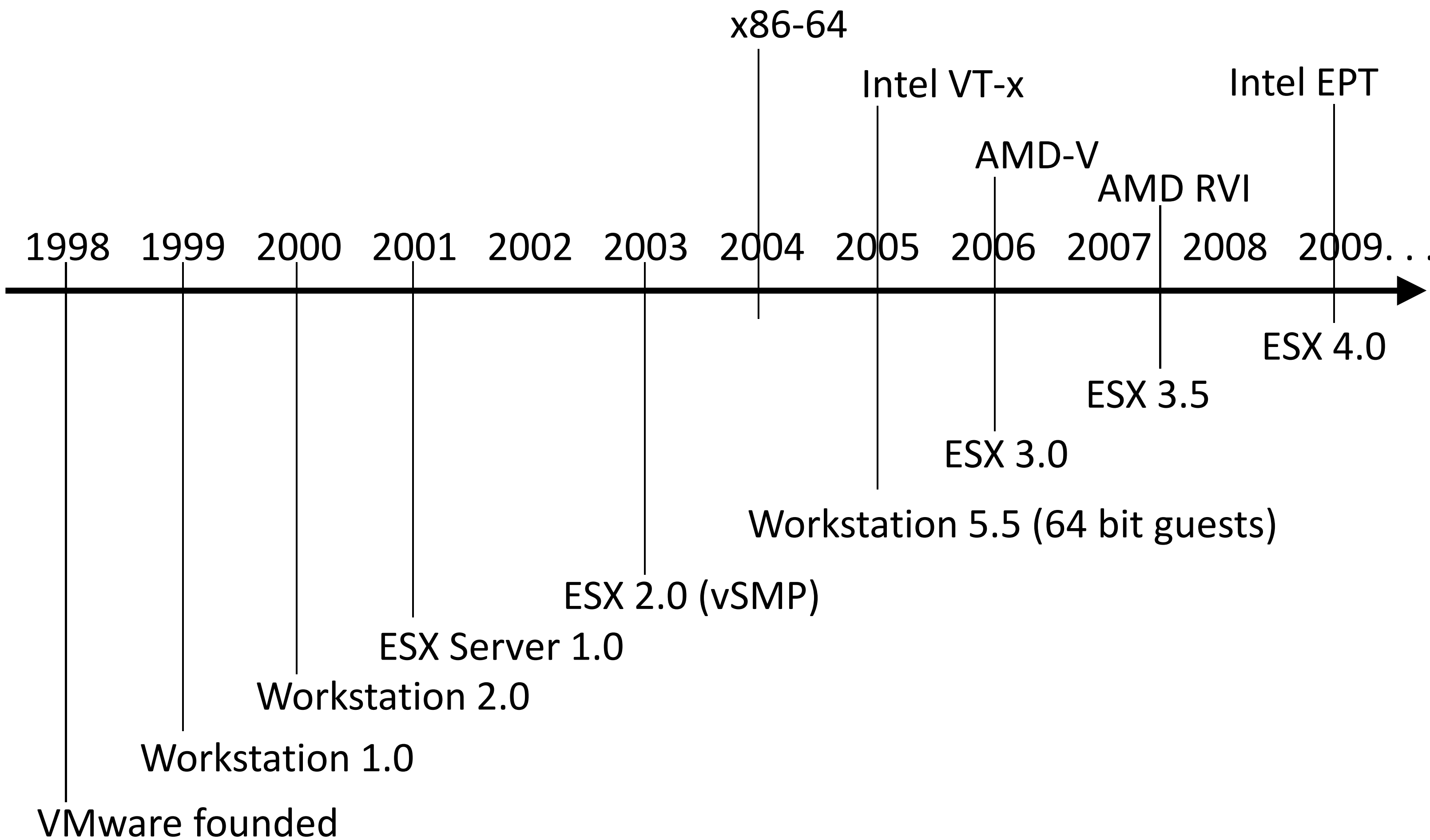


# Issues with I/O Virtualization

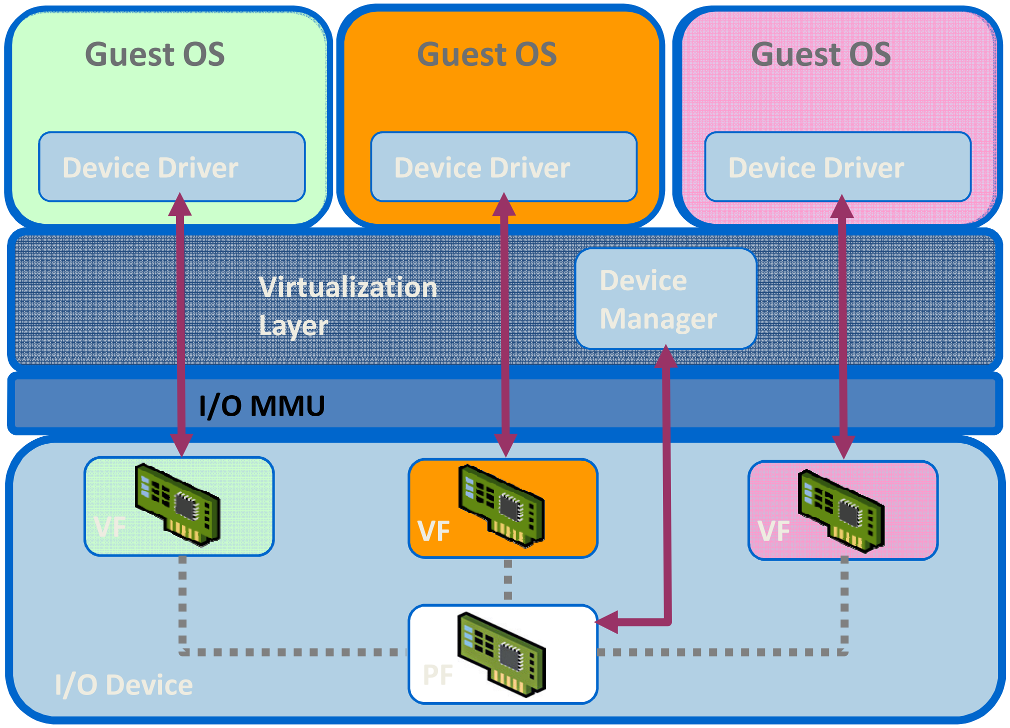
- Need physical memory address translation
  - need to copy
  - need translation
  - need IO MMU
- Need way to dispatch incoming requests

# Backup Slides

# Brief History of VMware x86 Virtualization



# Passthrough I/O Virtualization




PF = Physical Function, VF = Virtual Function

- High Performance
  - Guest drives device directly
  - Minimizes CPU utilization
- Enabled by HW Assists
  - I/O-MMU for DMA isolation *e.g.* Intel VT-d, AMD IOMMU
  - Partitionable I/O device *e.g.* PCI-SIG IOV spec
- Challenges
  - Hardware independence
  - Migration, suspend/resume
  - Memory overcommitment