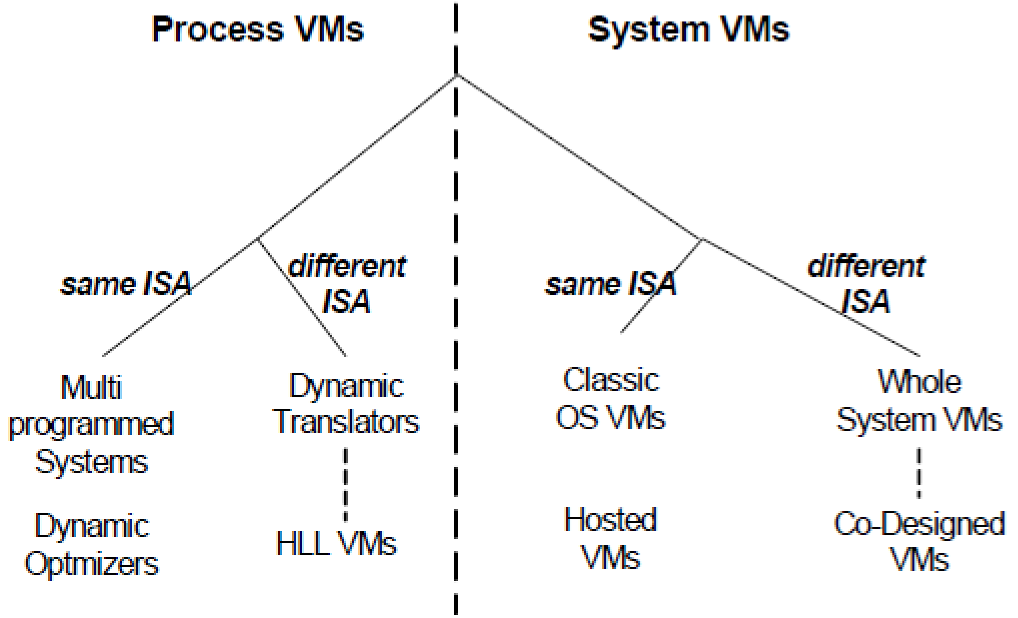
*Overview of Virtual Machines **

*This presentation are based on the slides from Vmware
http://labs.vmware.com/academic/introduction-to-virtualization

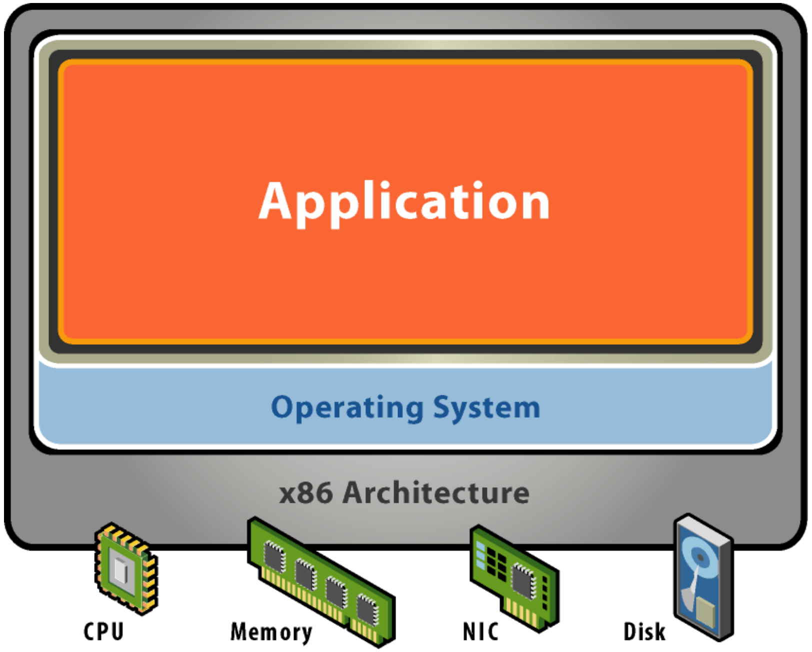
Types of Virtualization

- Process Virtualization
  - Language-level Java, .NET, Smalltalk
  - OS-level processes, Solaris Zones, BSD Jails, Virtuozzo
  - 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

Another taxonomy of virtual machine architectures

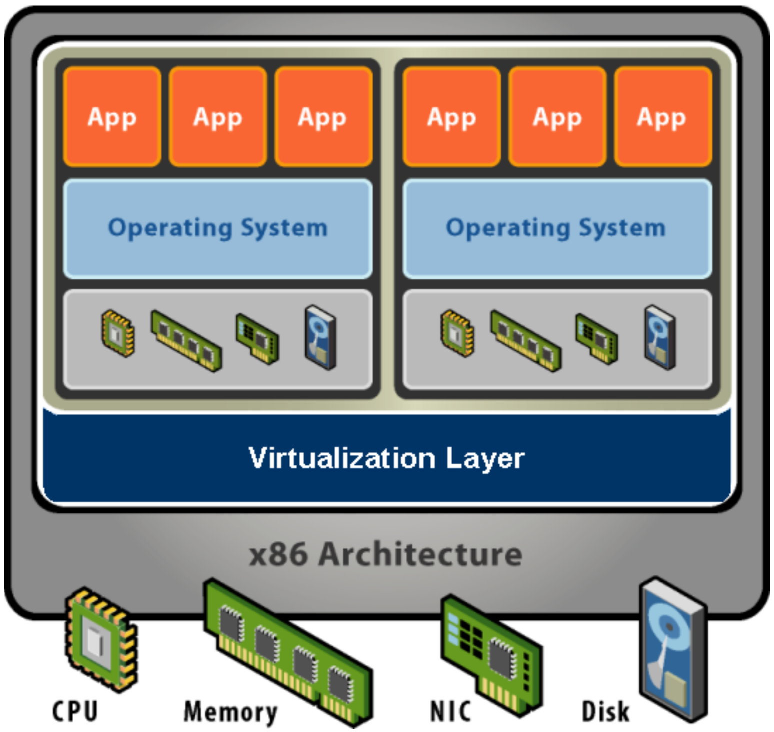


Starting Point: A Physical Machine



- 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

Why Virtualize?

- Consolidate resources
  - Server consolidation
  - Client consolidation
- Improve system management
  - For both hardware and software
  - From the desktop to the data center
- Improve the software lifecycle
  - Develop, debug, deploy and maintain applications in virtual machines
- Increase application availability
  - Fast, automated recovery

Consolidate resources

- Server consolidation
  - reduce number of servers
  - reduce space, power and cooling
  - 70-80% reduction numbers cited in industry
- Client consolidation
  - developers: test multiple OS versions, distributed application configurations on a single machine
  - end user: Windows on Linux, Windows on Mac
  - reduce physical desktop space, avoid managing multiple physical computers

Improve system management

- Data center management
  - VM portability and live migration a key enabler
  - automate resource scheduling across a pool of servers
  - optimize for performance and/or power consumption
  - allocate resources for new applications on the fly
  - add/remove servers without application downtime
- Desktop management
  - centralize management of desktop VM images
  - automate deployment and patching of desktop VMs
  - run desktop VMs on servers or on client machines
- Industry-cited 10x increase in sysadmin efficiency

Improve the software lifecycle

- Develop, debug, deploy and maintain applications in virtual machines
- Power tool for software developers
  - record/replay application execution deterministically
  - trace application behavior online and offline
  - model distributed hardware for multi-tier applications
- Application and OS flexibility
  - run any application or operating system
- Virtual appliances
  - a complete, portable application execution environment

Increase application availability

- Fast, automated recovery
  - automated failover/restart within a cluster
  - disaster recovery across sites
  - VM portability enables this to work reliably across potentially different hardware configurations
- Fault tolerance
  - hypervisor-based fault tolerance against hardware failures [Bressoud and Schneider, SOSP 1995]
  - run two identical VMs on two different machines, backup VM takes over if primary VM's hardware crashes
  - commercial prototypes beginning to emerge (2008)

Why virtualize?

- Virtualization makes hardware and software more flexible and efficient
- Virtualization improves the way people use and manage computers

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 …

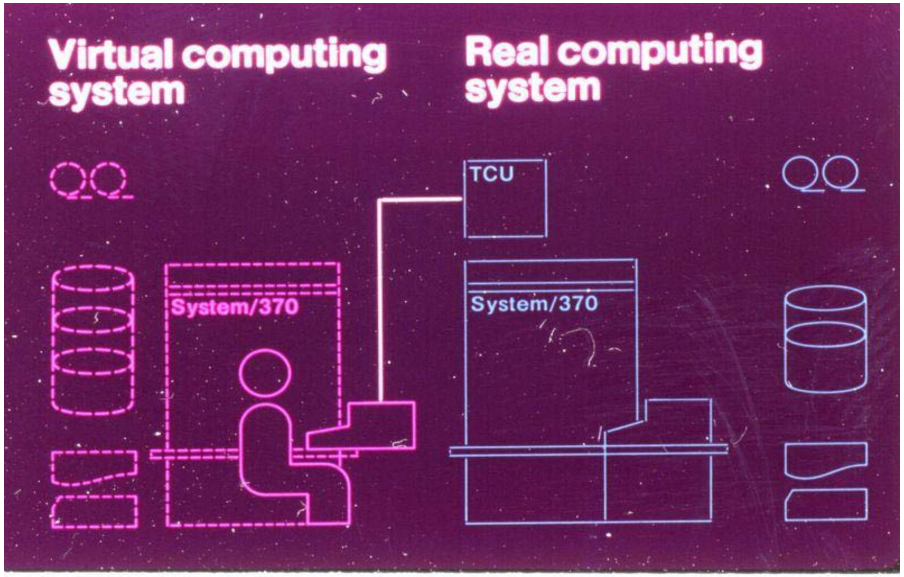
What is a Virtual Machine Monitor?

- 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

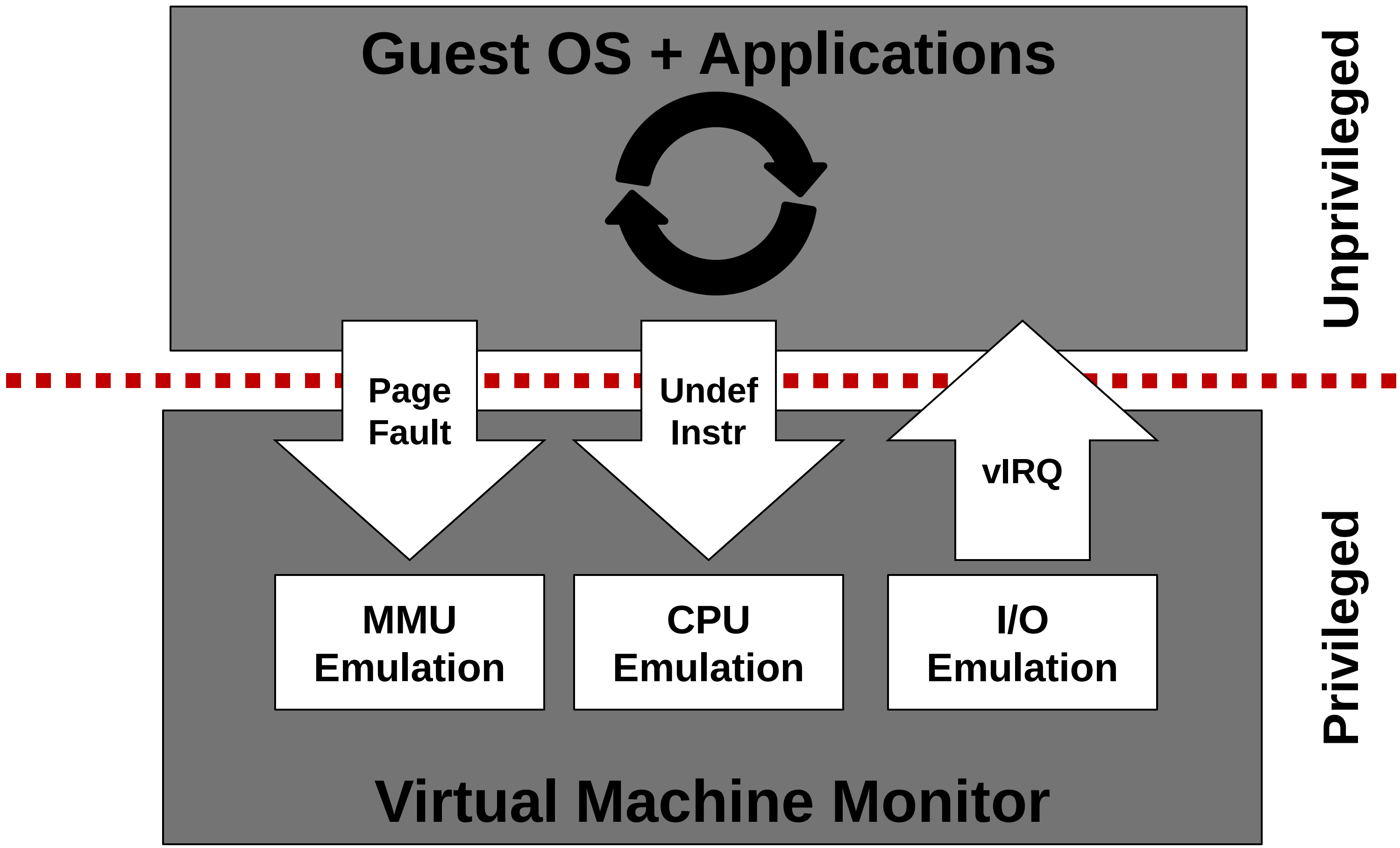
Modern Virtualization Applications

- 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"

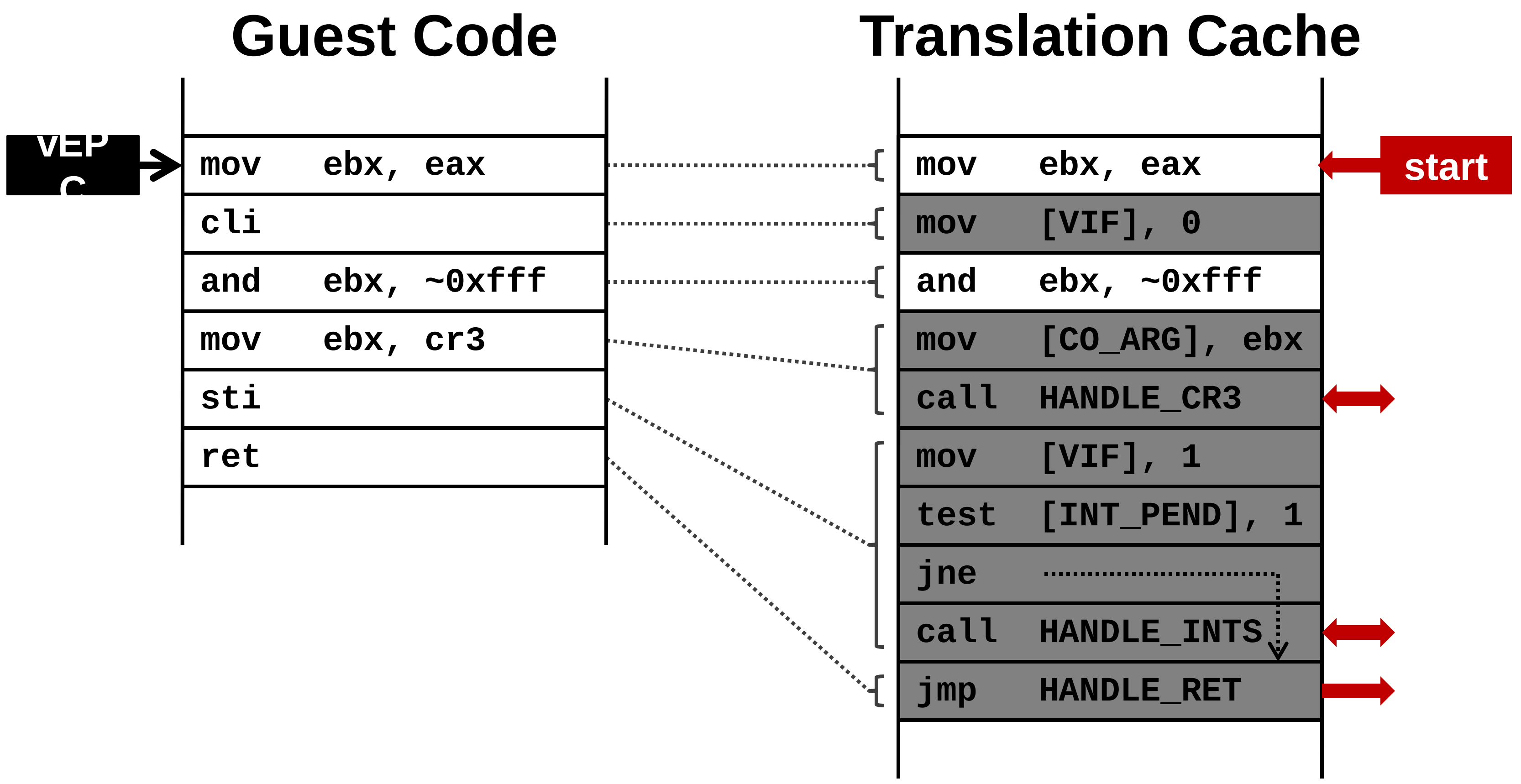
A processor or mode of a processor is *strictly virtualizable* if, when executed in a lesser privileged mode:

- 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



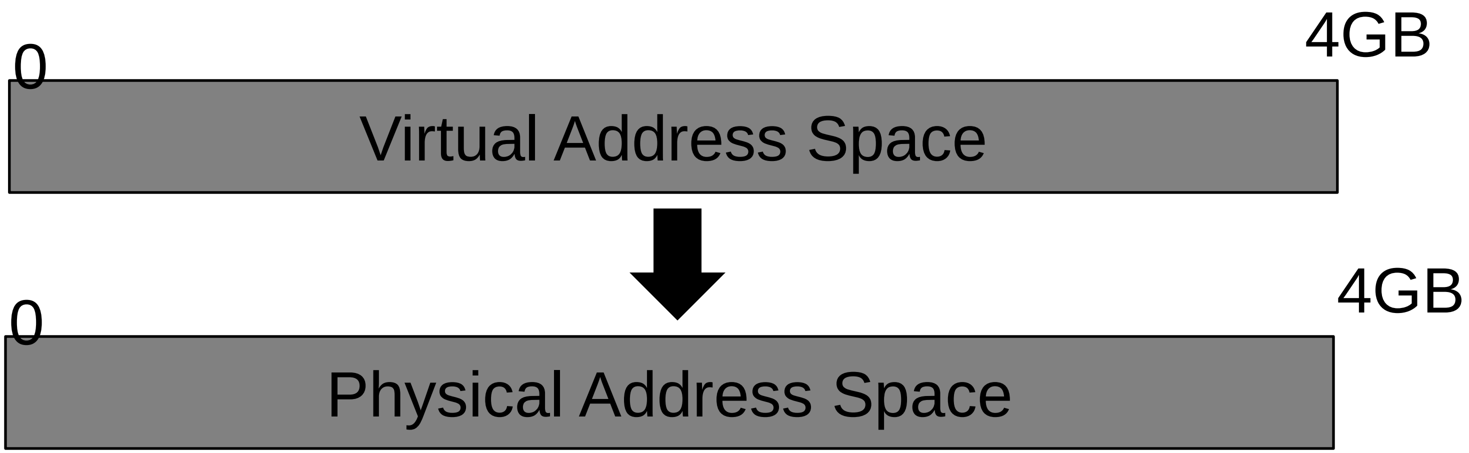
Issues with Binary Translation

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

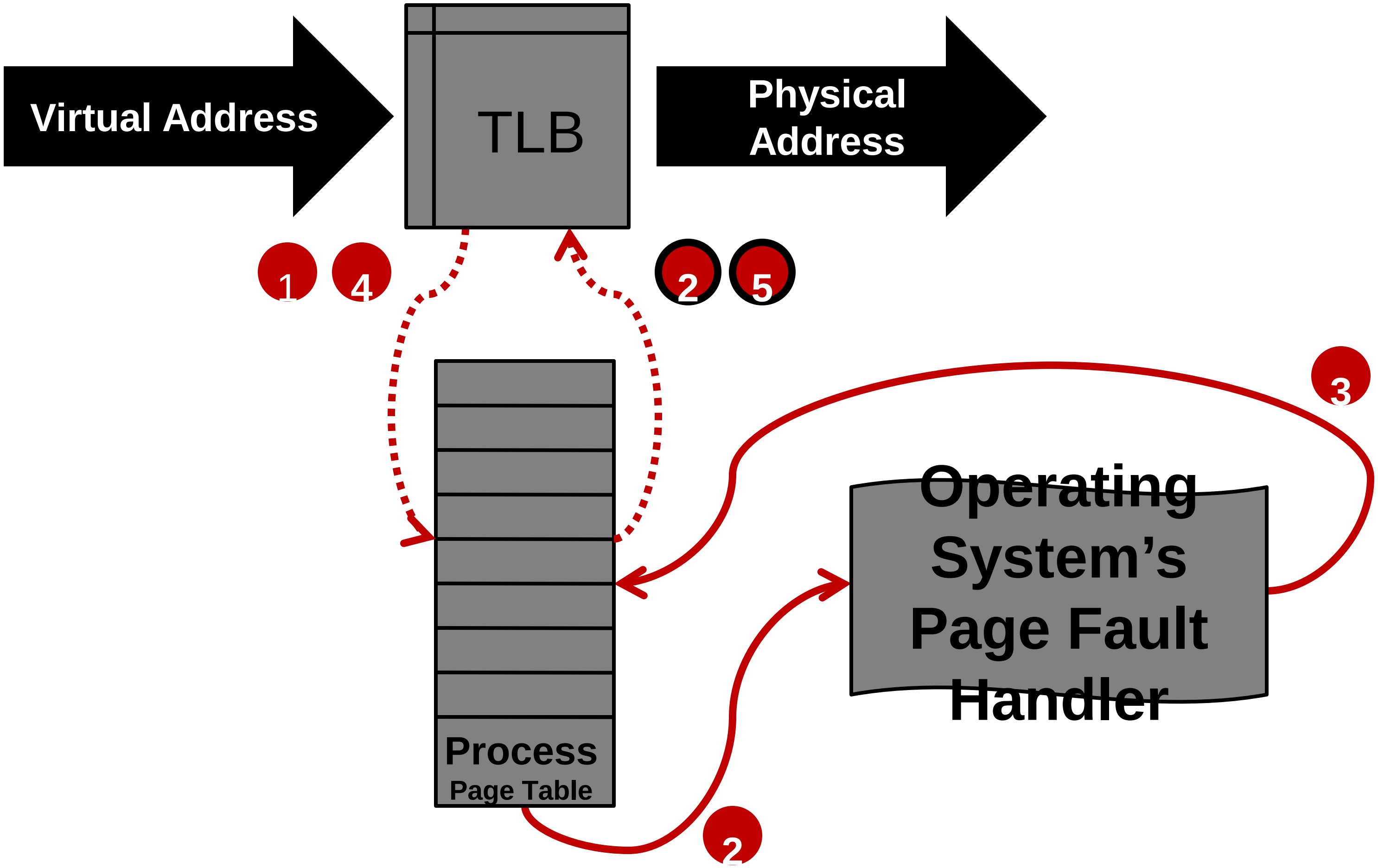
Memory Virtualization

- Shadow Page Tables
- Nested Page Tables

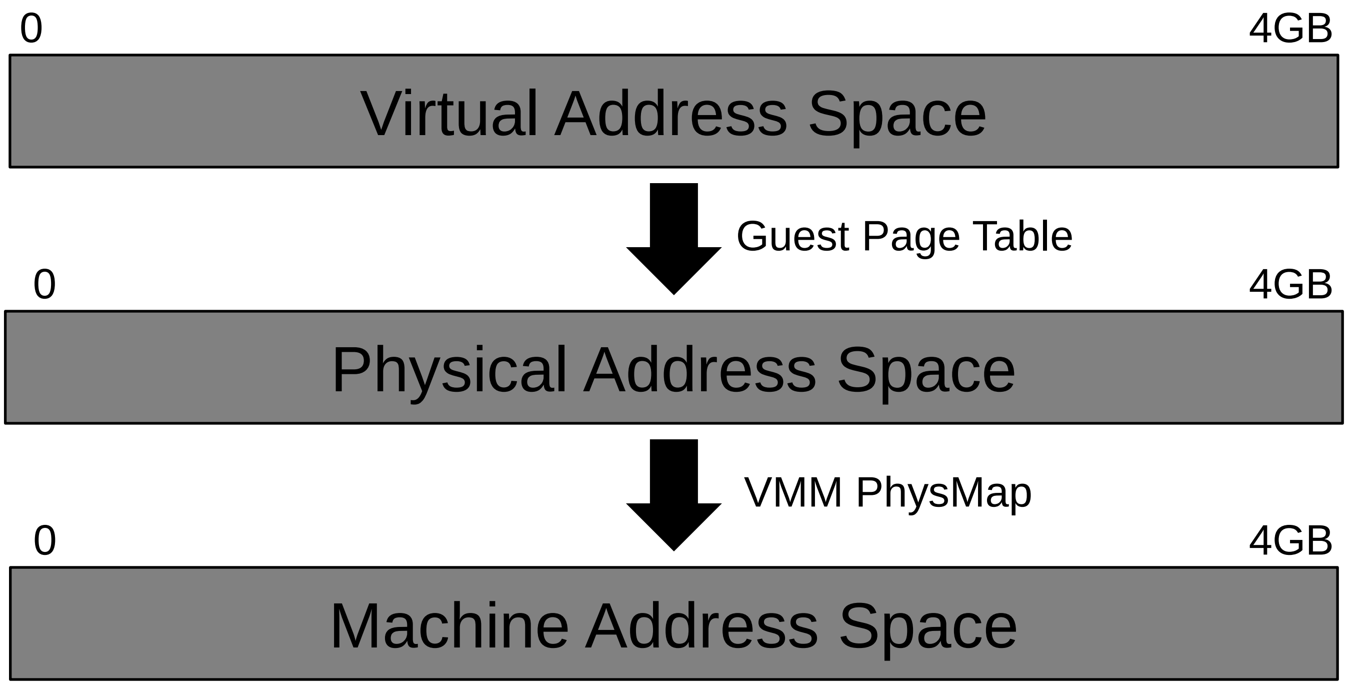
Traditional Address Spaces



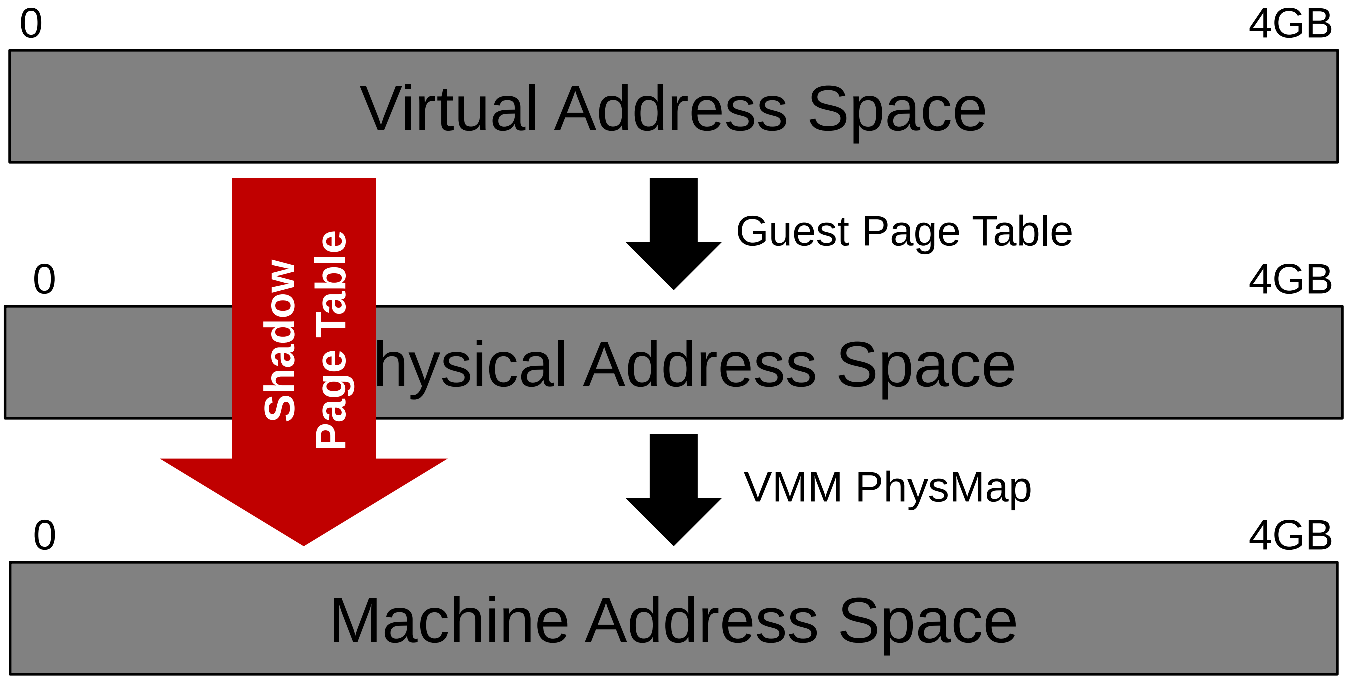
Traditional Address Translation



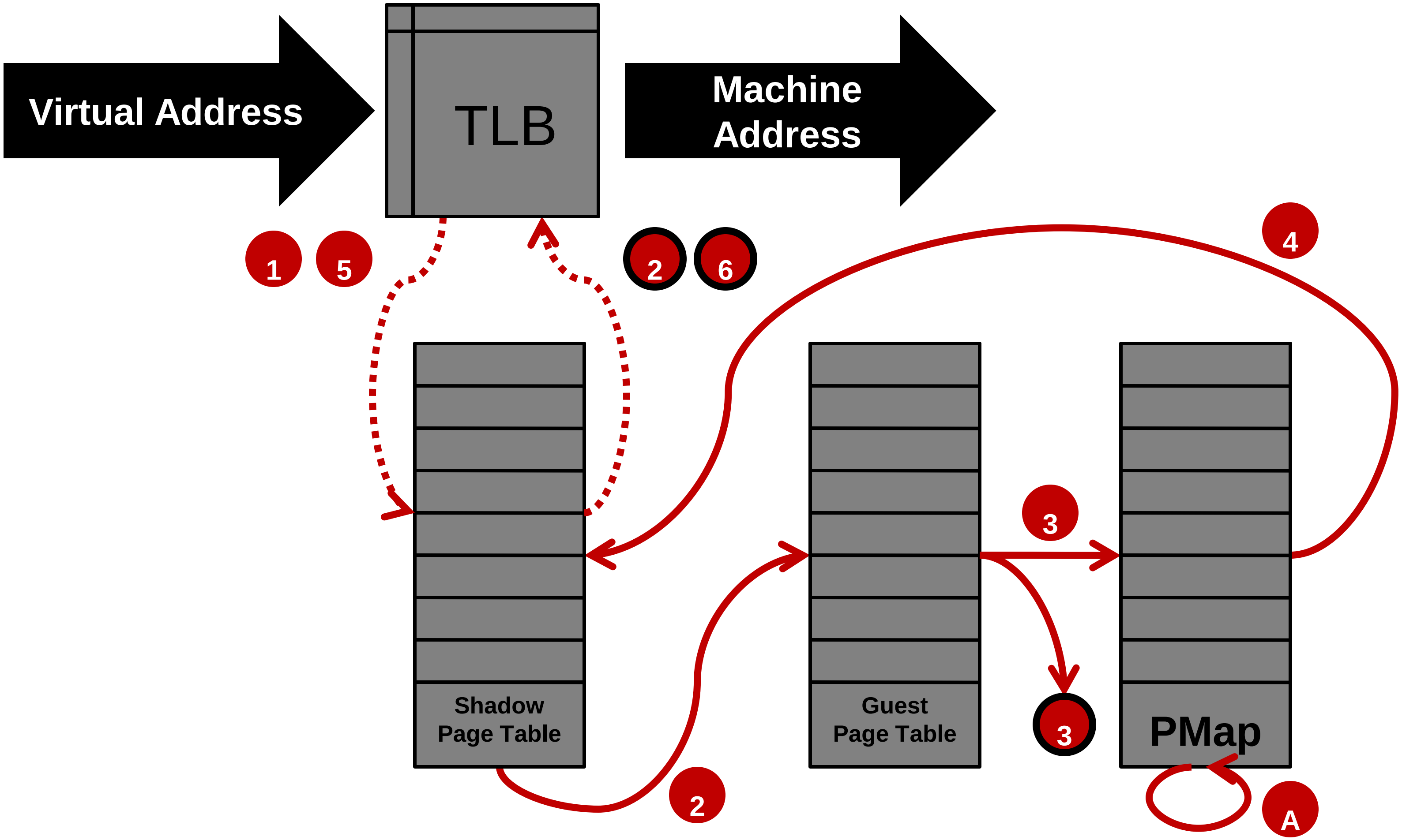
Virtualized Address Spaces



Virtualized Address Spaces w/ Shadow Page Tables



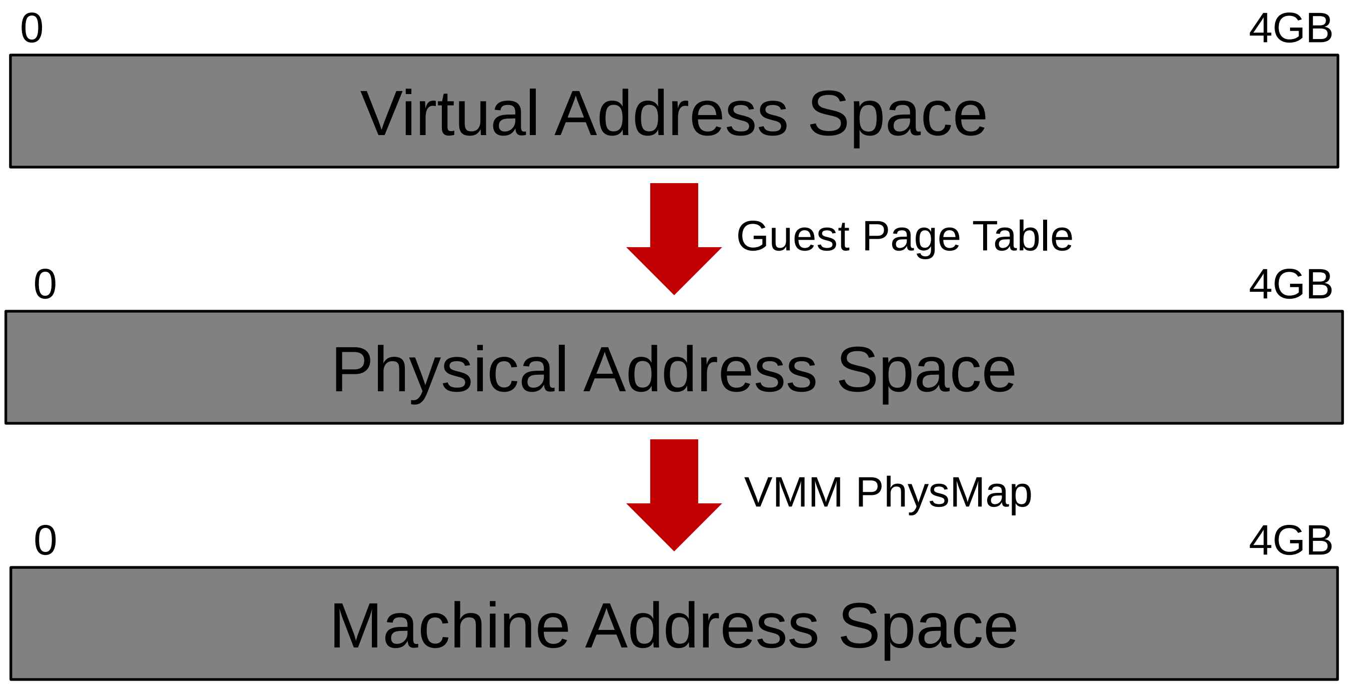
Virtualized Address Translation w/ Shadow Page Tables



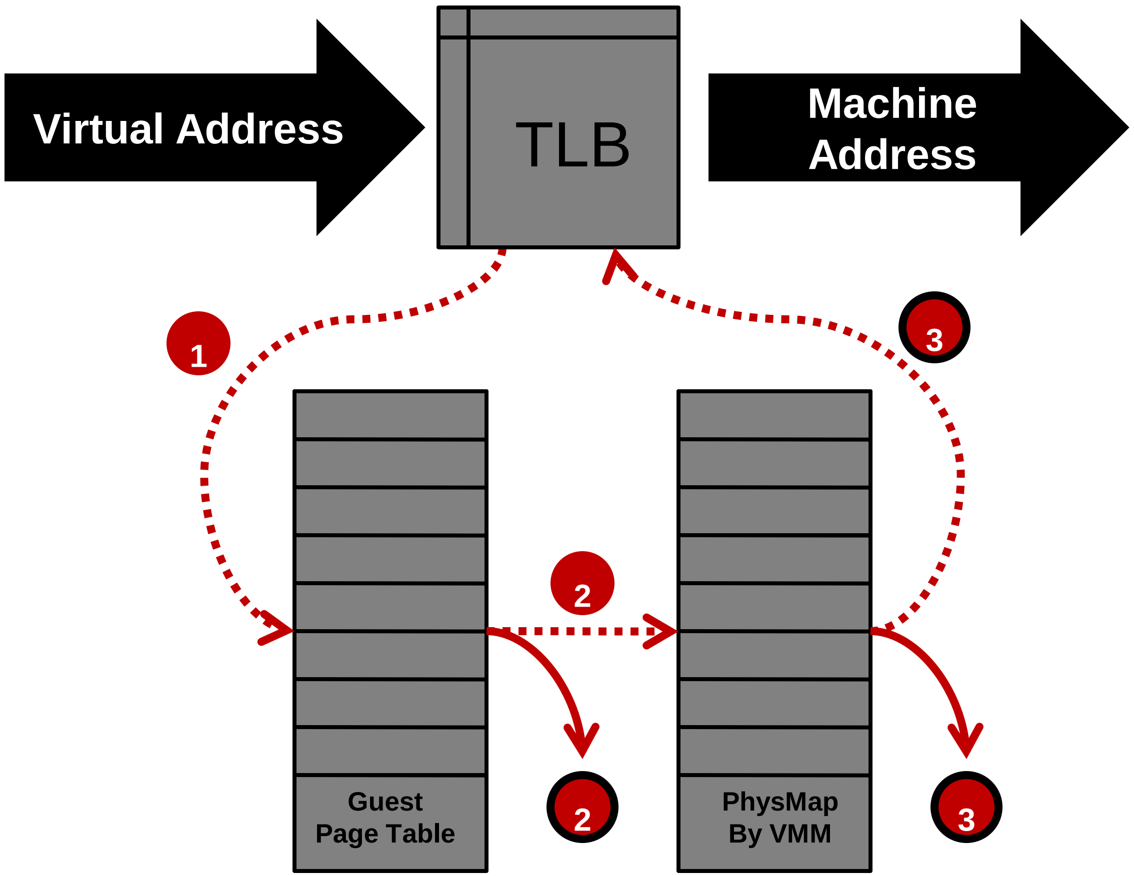
Issues with Shadow Page Tables

- 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



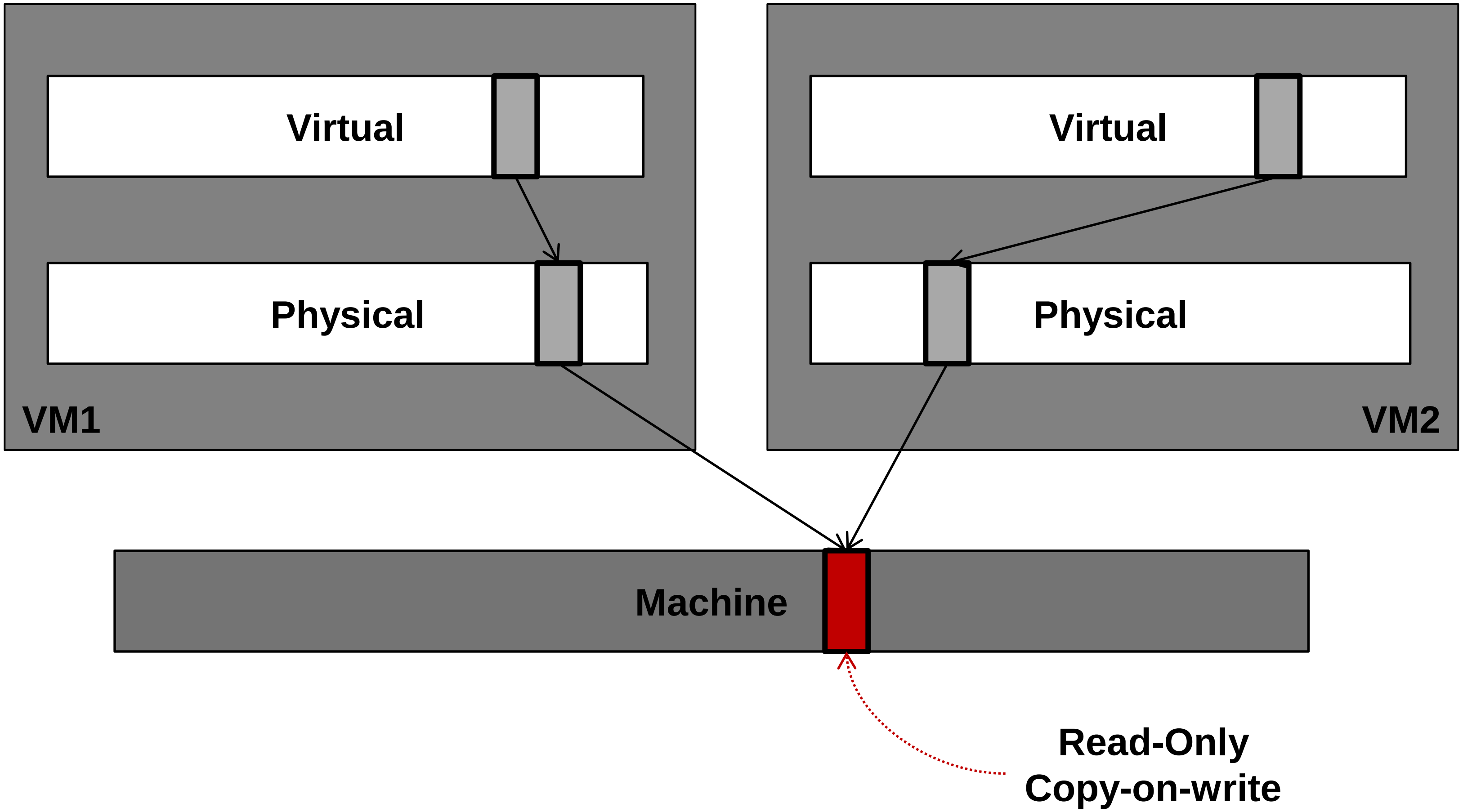
Virtualized Address Translation w/ Nested Page Tables



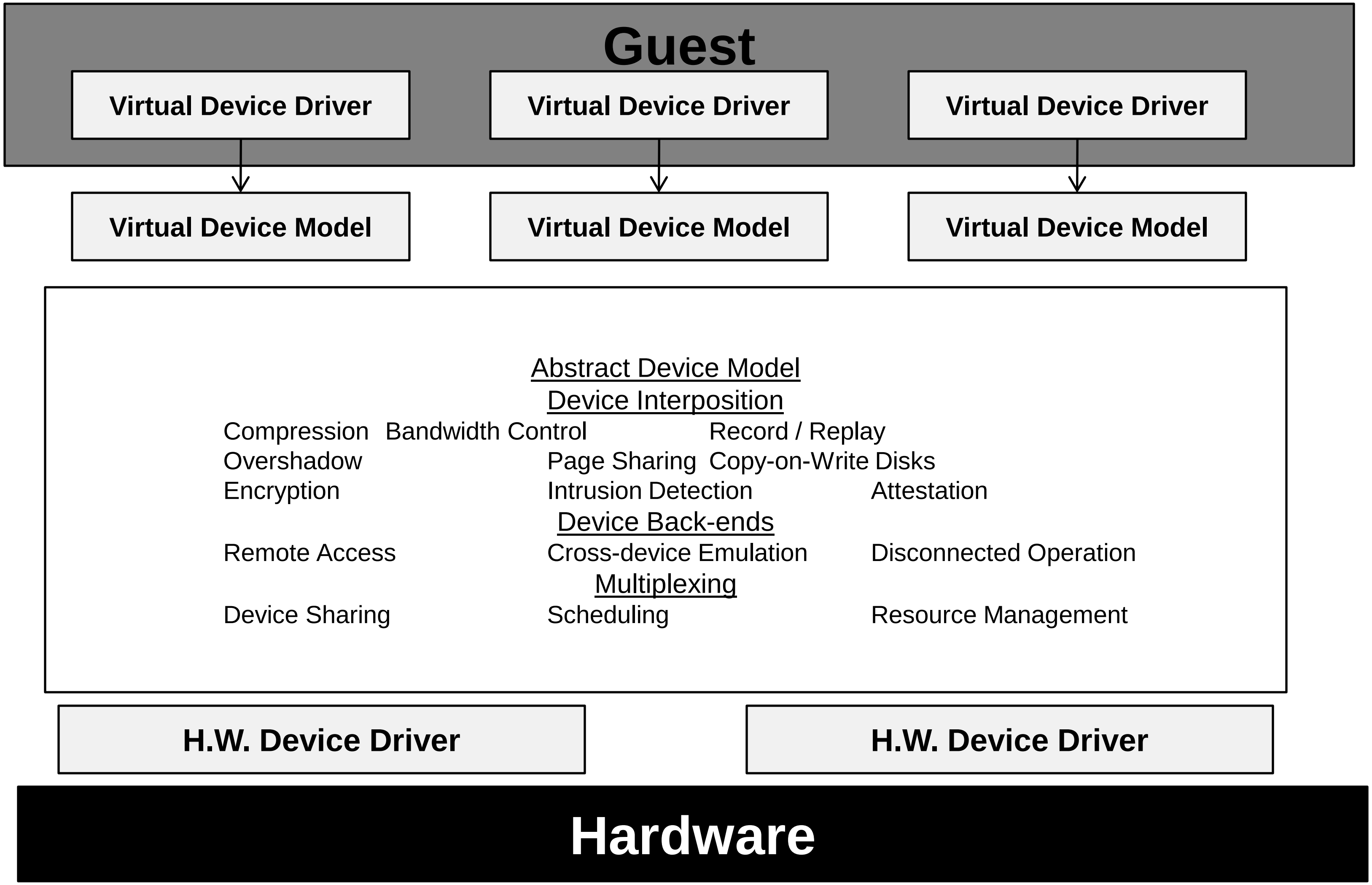
Issues with Nested Page Tables

- 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



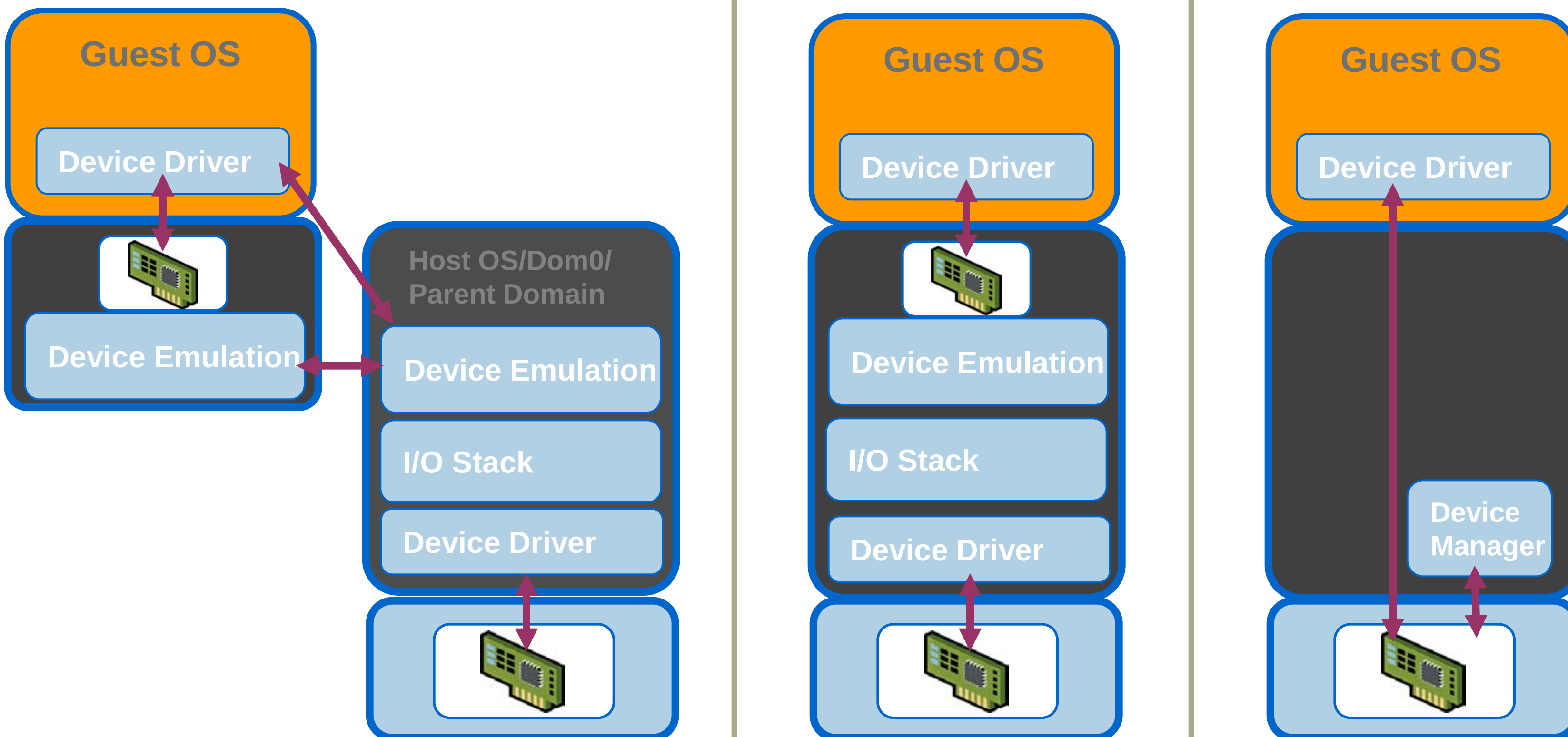
I/O Virtualization Implementations

Emulated I/O

Hosted or Split

Hypervisor Direct

Passthrough I/O



VMware Workstation, VMware Server, Xen, Microsoft Hyper-V, Virtual Server

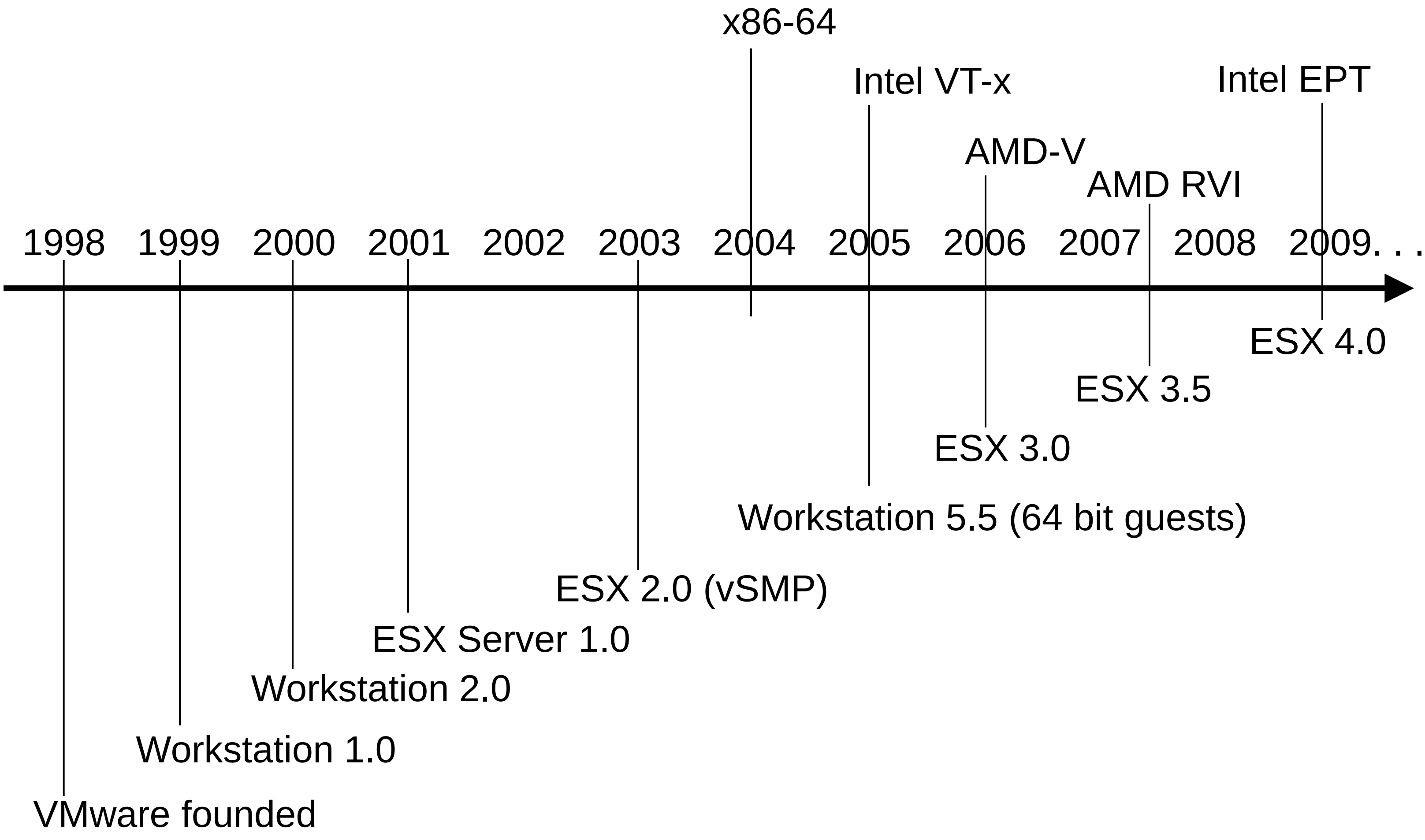
VMware ESX

VMware ESX (FPT)

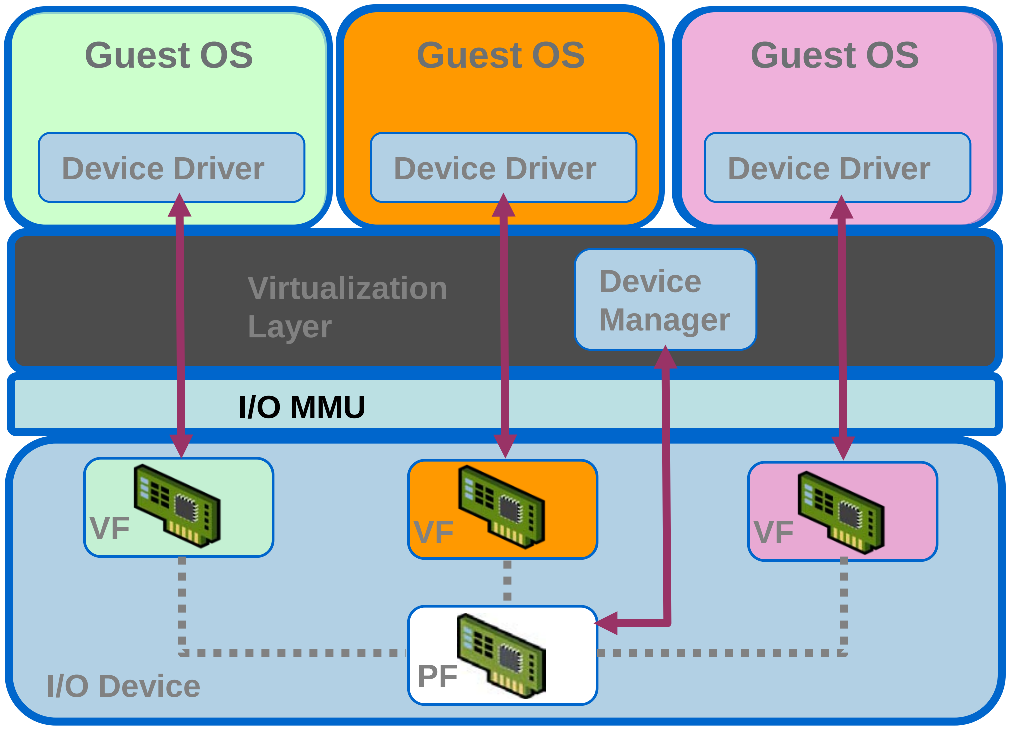
Issues with I/O Virtualization

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

Brief History of VMware x86 Virtualization



Passthrough I/O Virtualization



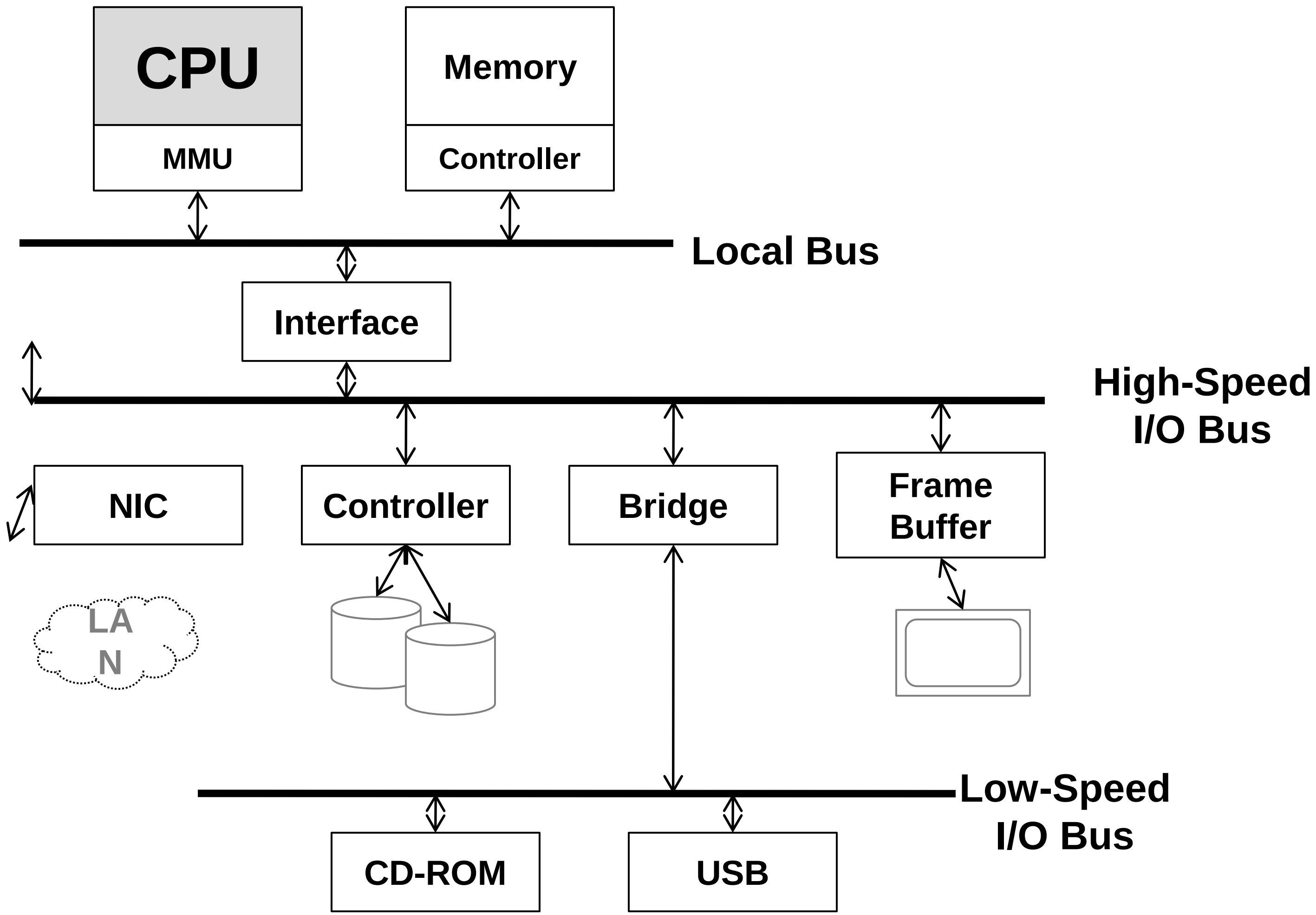
PF = Physical Function, VF = Virtual Function

- 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

*CPU Virtualization Basics **

*This presentation are based on the slides from Vmware
http://labs.vmware.com/academic/introduction-to-virtualization

Computer System Organization



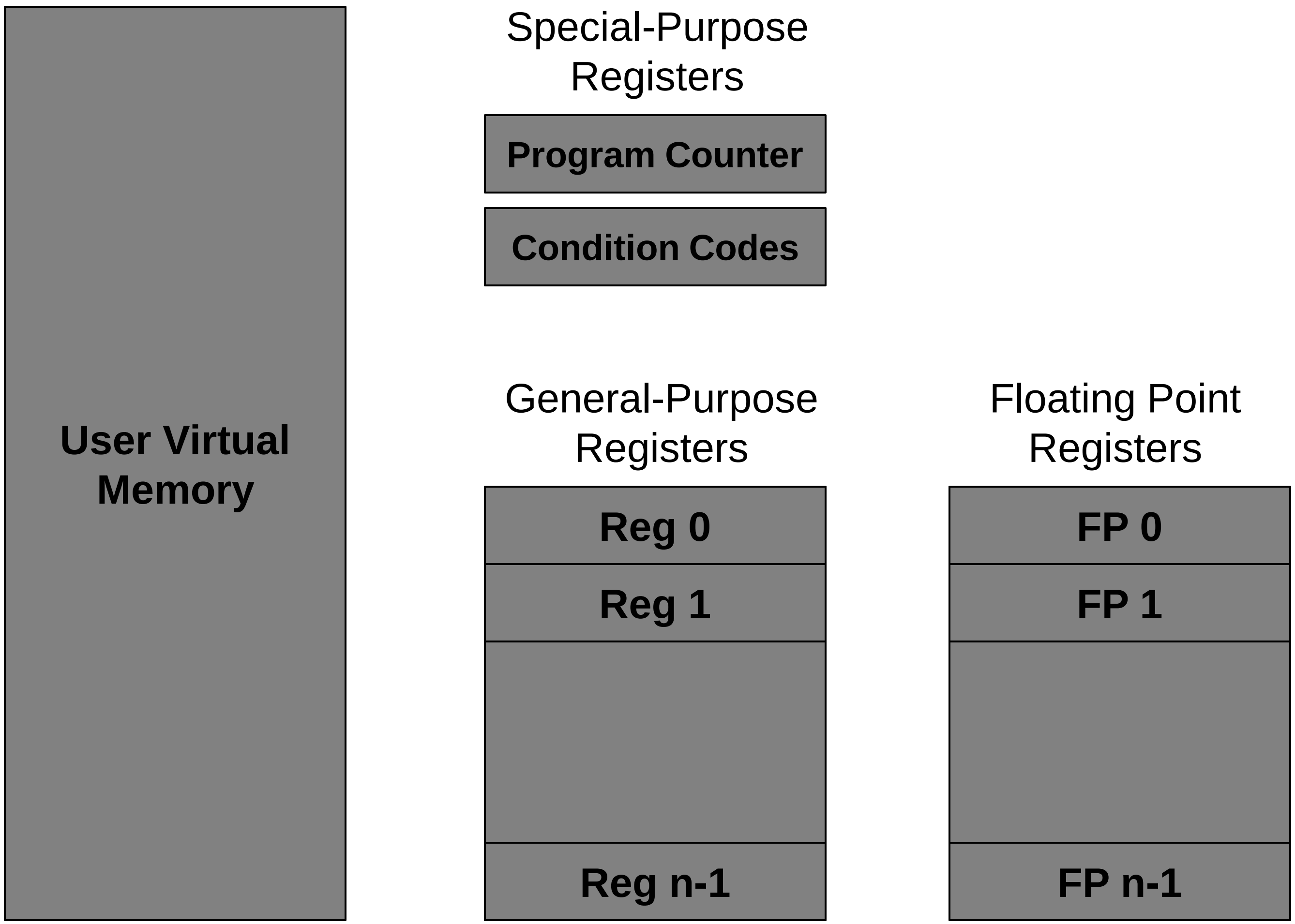
CPU Organization

- Instruction Set Architecture (ISA)

  Defines:

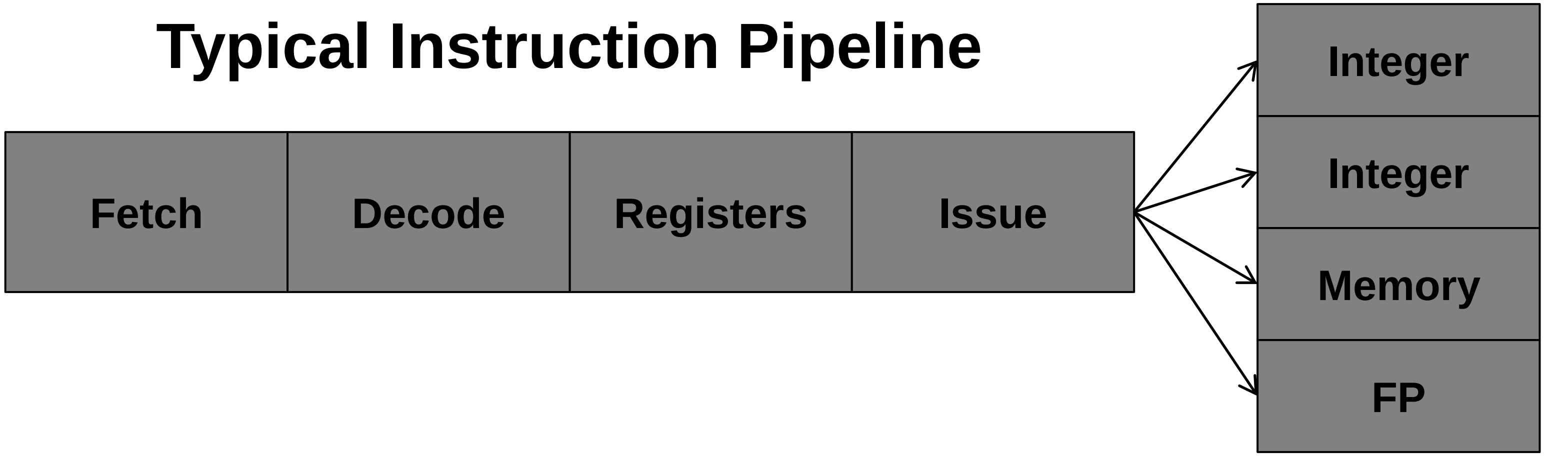
  – the state visible to the programmer
    - registers and memory
  – the instruction that operate on the state
- ISA typically divided into 2 parts
  – User ISA
    - Primarily for computation
  – System ISA
    - Primarily for system resource management

User ISA - State



User ISA – Instructions

Typical Instruction Pipeline

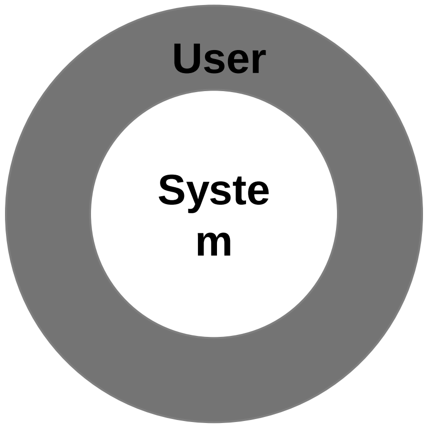


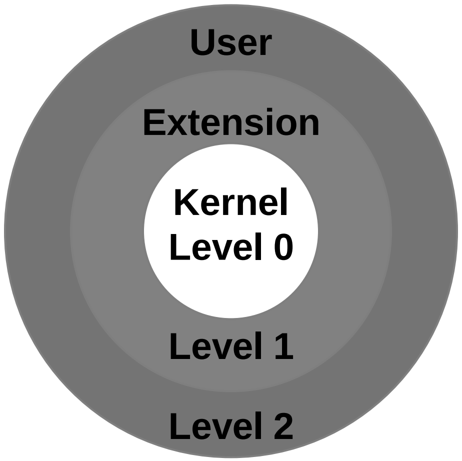
Integer	Memory	Control Flow	Floating Point
Add Sub And Compare …	Load byte Load Word Store Multiple Push …	Jump Jump equal Call Return …	Add single Mult. double Sqrt double …

Instruction Groupings

System ISA

- Privilege Levels
- Control Registers
- Traps and Interrupts
  - Hardcoded Vectors
  - Dispatch Table
- System Clock
- MMU
  - Page Tables
  - TLB
- I/O Device Access



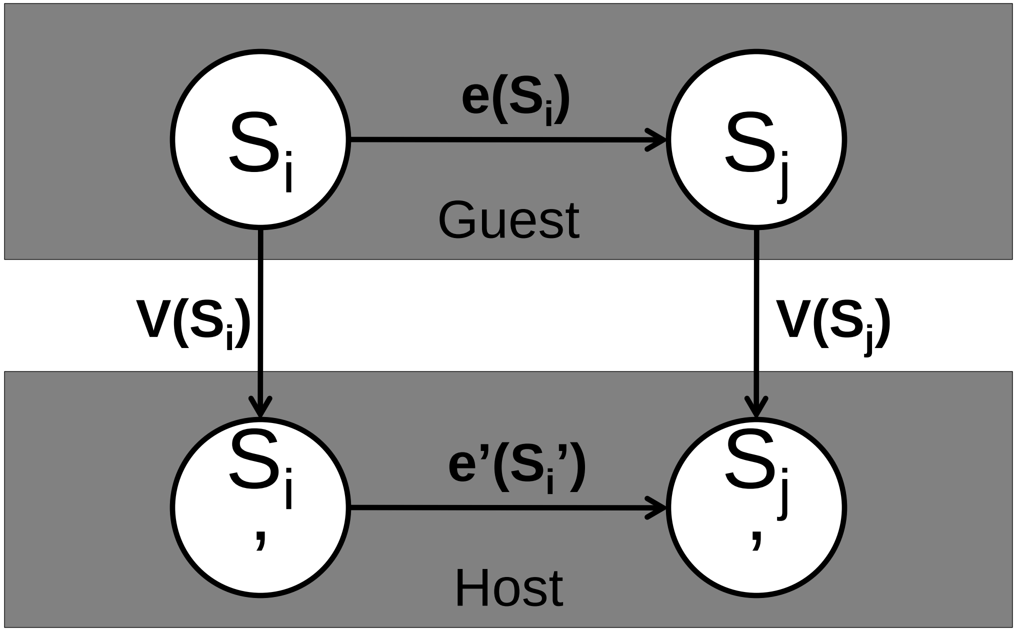


Outline

- CPU Background
- Virtualization Techniques
  - System ISA Virtualization
  - Instruction Interpretation
  - Trap and Emulate
  - Binary Translation
  - Hybrid Models

Isomorphism

- Formally, virtualization involves the construction of an isomorphism from guest state to host state.



Virtualizing the System ISA

- Hardware needed by monitor
  - Ex: monitor must control real hardware interrupts
- Access to hardware would allow VM to compromise isolation boundaries
  - Ex: access to MMU would allow VM to write any page
- So…
  - All access to the virtual System ISA by the guest must be emulated by the monitor in software.
  - System state kept in memory.
  - System instructions are implemented as functions in the monitor.

Example: CPUState

- Goal for CPU virtualization techniques
  - Process normal instructions as fast as possible
  - Forward privileged instructions to emulation routines

```
static struct {
   uint32  GPR[16];
   uint32  LR;
   uint32  PC;
   int     IE;
   int     IRQ;
} CPUState;
```

```
void CPU_CLI(void)
{
   CPUState.IE = 0;
}

void CPU_STI(void)
{
   CPUState.IE = 1;
}
```

Instruction Interpretation

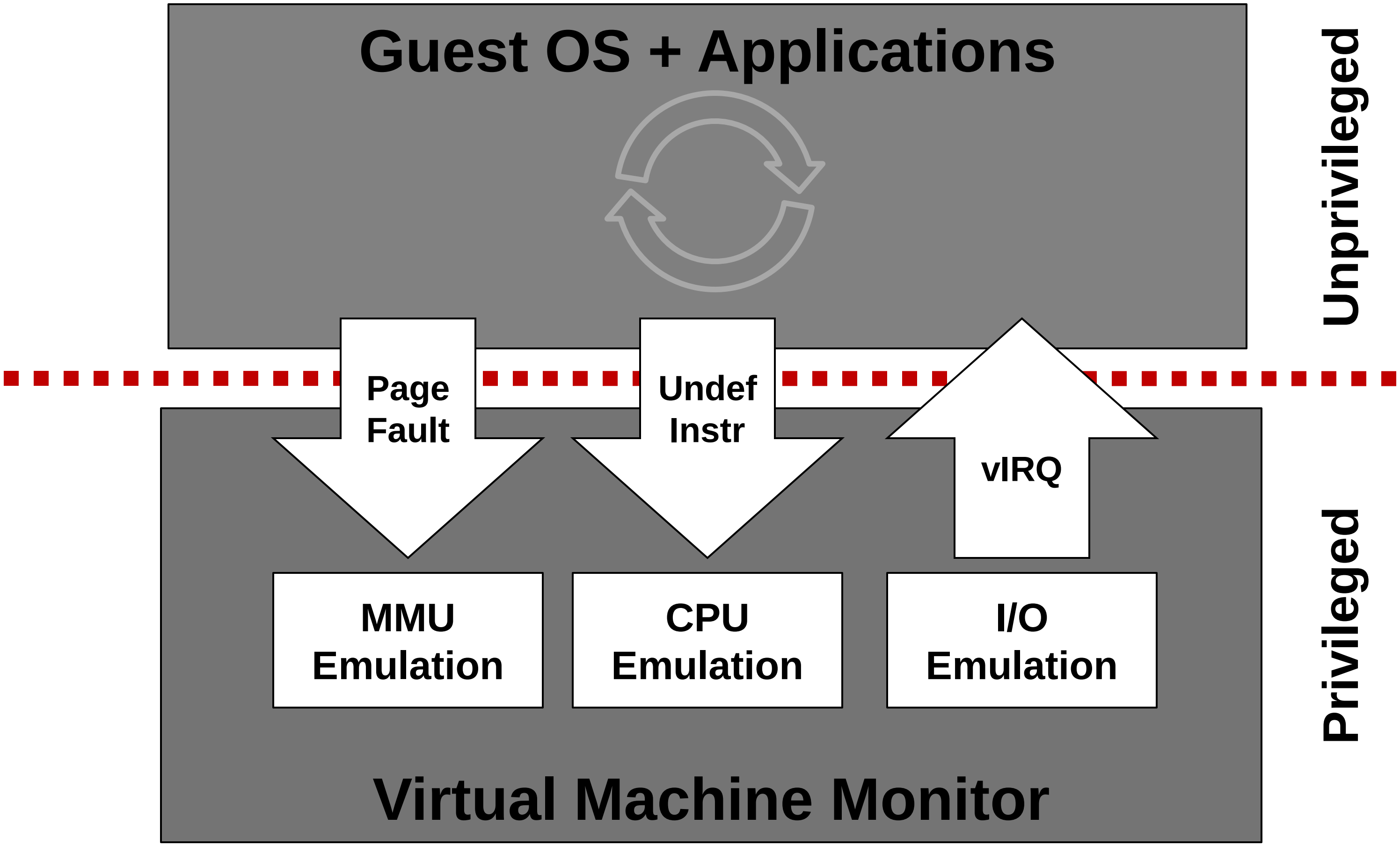
- Emulate Fetch/Decode/Execute pipeline in software
- Postives
  - Easy to implement
  - Minimal complexity
- Negatives
  - Slow!

Example: Virtualizing the Interrupt Flag w/ Instruction Interpreter

```
void CPU_Run(void)
{
   while (1) {
      inst = Fetch(CPUState.PC);

      CPUState.PC += 4;

      switch (inst) {
      case ADD:
         CPUState.GPR[rd]
            = GPR[rn] + GPR[rm];
         break;
      …
      case CLI:
         CPU_CLI();
         break;
      case STI:
         CPU_STI();
         break;
      }

      if (CPUState.IRQ
          && CPUState.IE) {
         CPUState.IE = 0;
         CPU_Vector(EXC_INT);
      }
   }
}
void CPU_CLI(void)
{
   CPUState.IE = 0;
}
```

```
{
   CPUState.IE = 1;
}

void CPU_Vector(int exc)
{
   CPUState.LR = CPUState.PC;
   CPUState.PC = disTab[exc];
}
```

Trap and Emulate



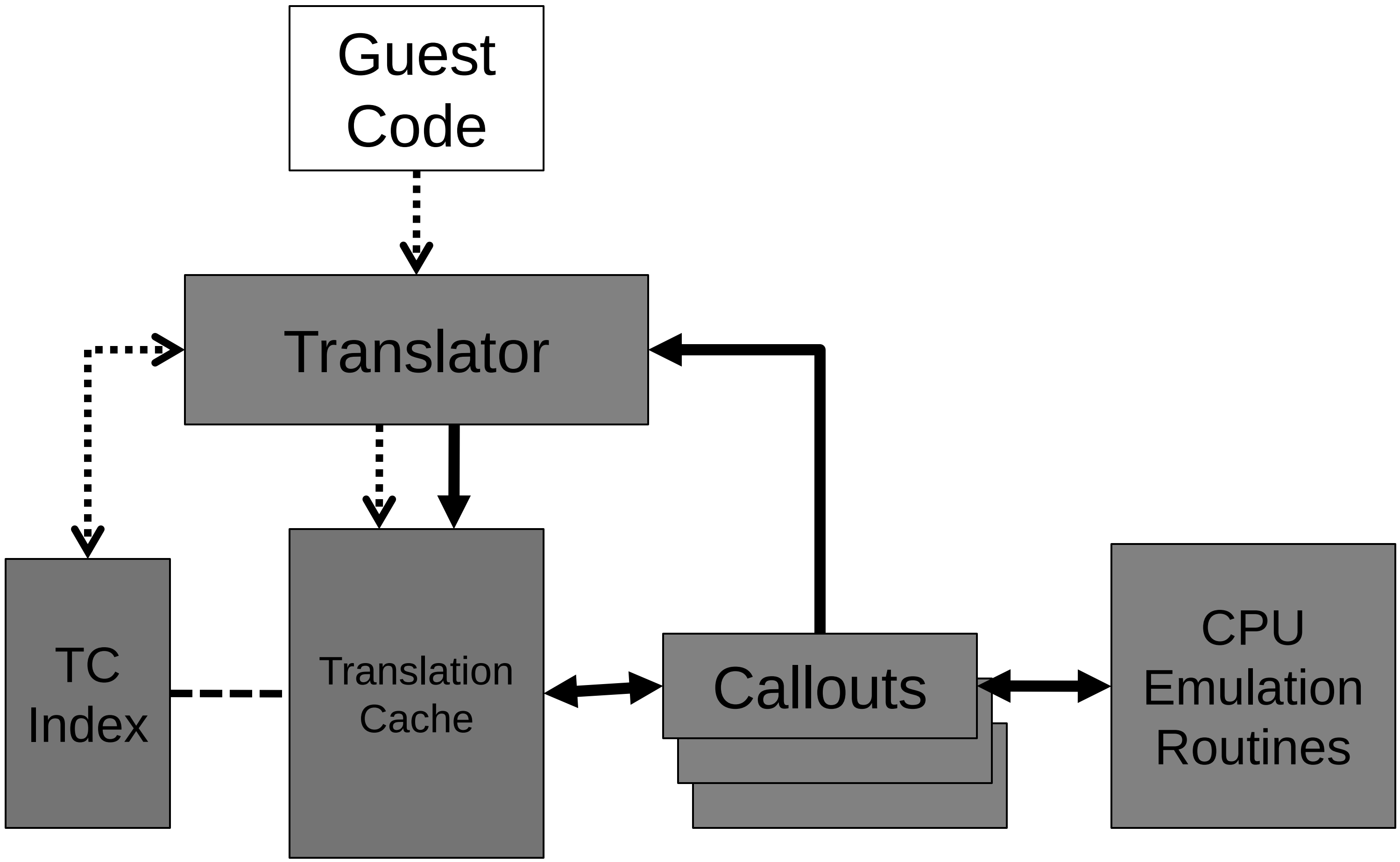
"Strictly Virtualizable"

- 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
  - …

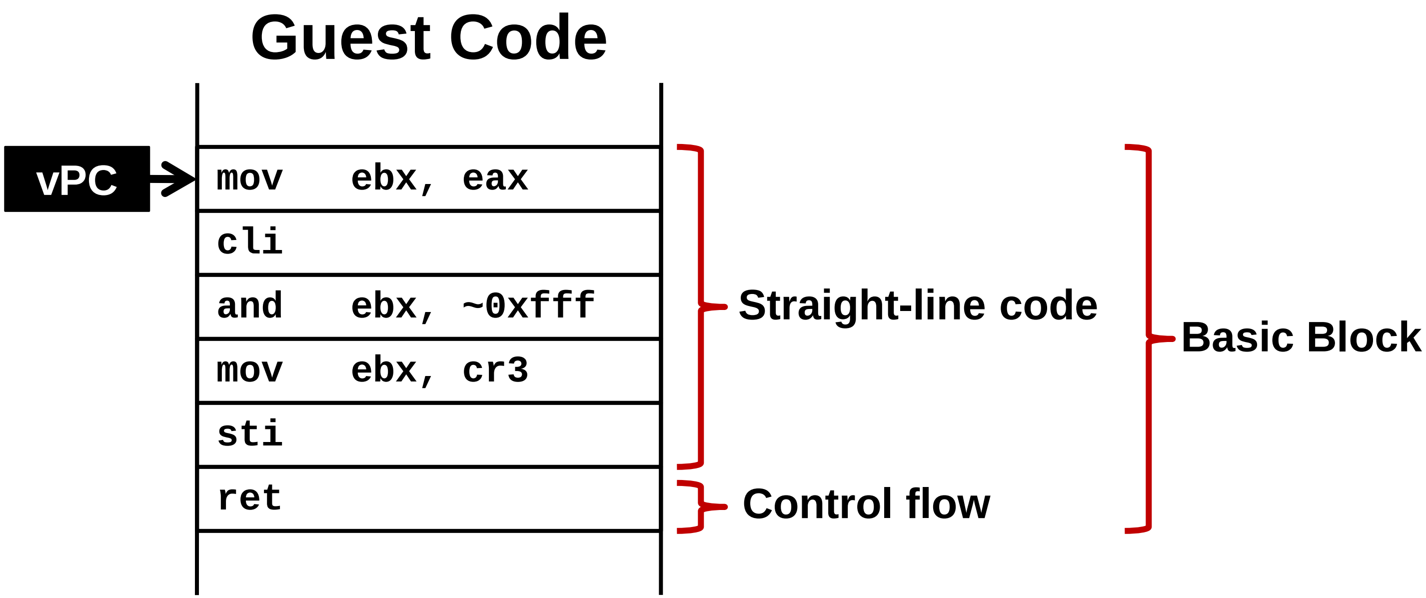
Issues with Trap and Emulate

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

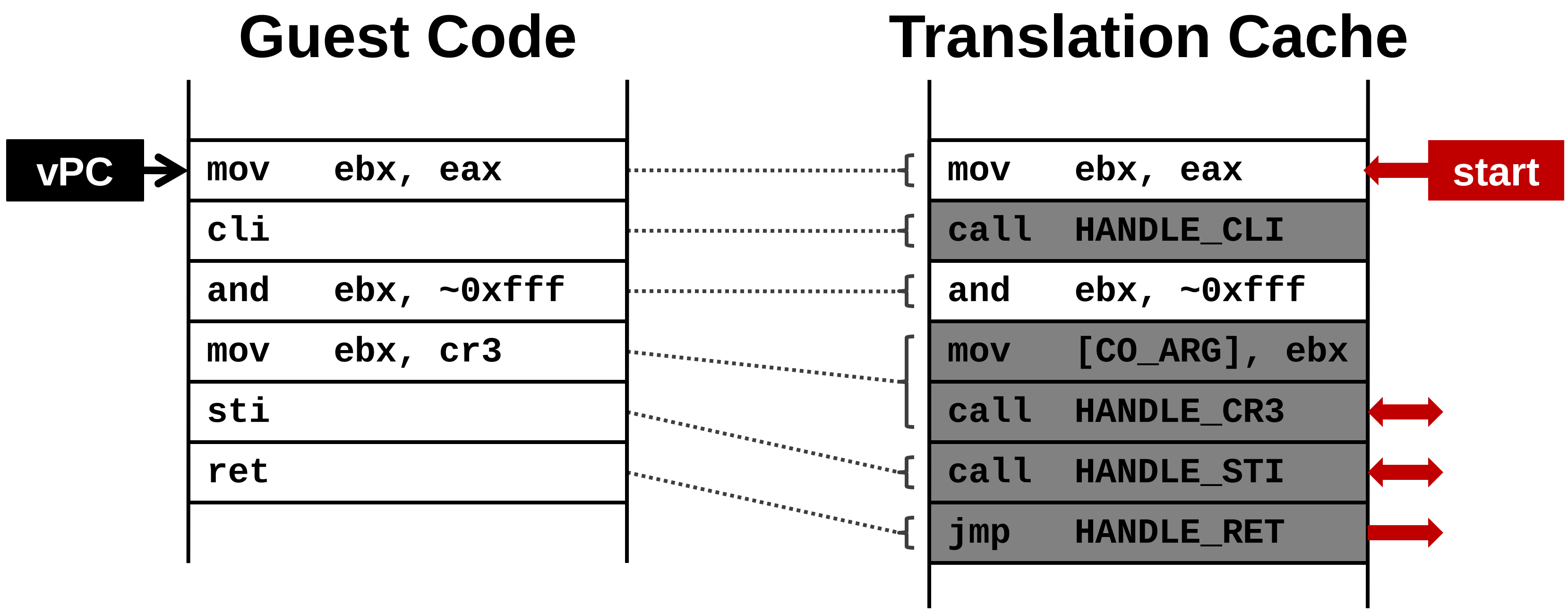
Binary Translator



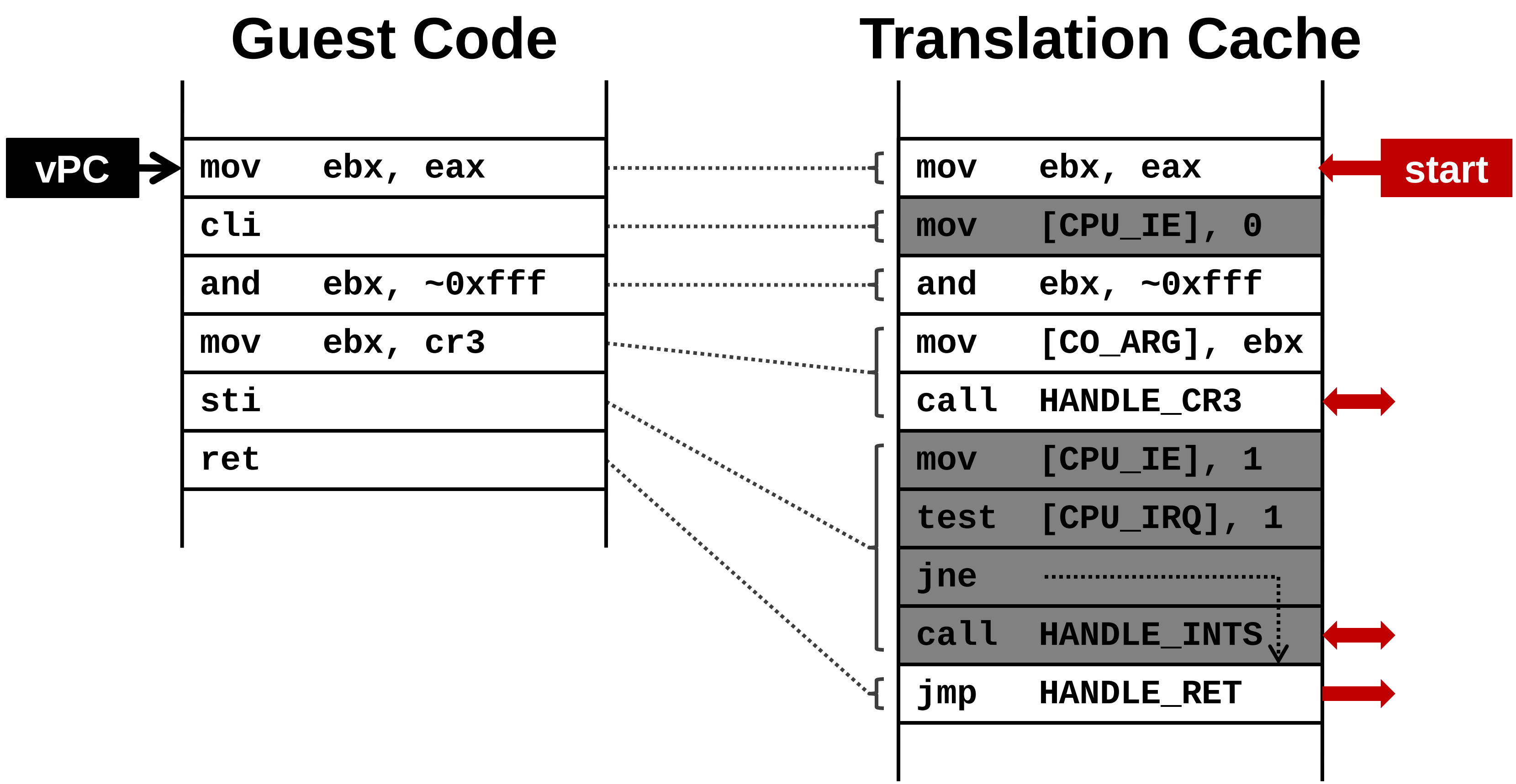
Basic Blocks



Binary Translation



Binary Translation



Basic Binary Translator

```
void BT_Run(void)
{
   CPUState.PC = _start;
   BT_Continue();
}

void BT_Continue(void)
{
   void *tcpc;

   tcpc = BTFindBB(CPUState.PC);

   if (!tcpc) {
      tcpc = BTTranslate(CPUState.PC);
   }

   RestoreRegsAndJump(tcpc);
}
```

```
void *BTTranslate(uint32 pc)
{
   void *start = TCTop;
   uint32 TCPC = pc;

   while (1) {
      inst = Fetch(TCPC);
      TCPC += 4;

      if (IsPrivileged(inst)) {
         EmitCallout();
      } else if (IsControlFlow(inst)) {
         EmitEndBB();
         break;
      } else {
         /* ident translation */
         EmitInst(inst);
      }
   }

   return start;
}
```

Basic Binary Translator – Part 2

```
void BT_CalloutSTI(BTSavedRegs regs)
{
   CPUState.PC = BTFindPC(regs.tcpc);
   CPUState.GPR[] = regs.GPR[];

   CPU_STI();

   CPUState.PC += 4;

   if (CPUState.IRQ
         && CPUState.IE) {
      CPUVector();
      BT_Continue();
      /* NOT_REACHED */
   }

   return;
}
```

Issues with Binary Translation

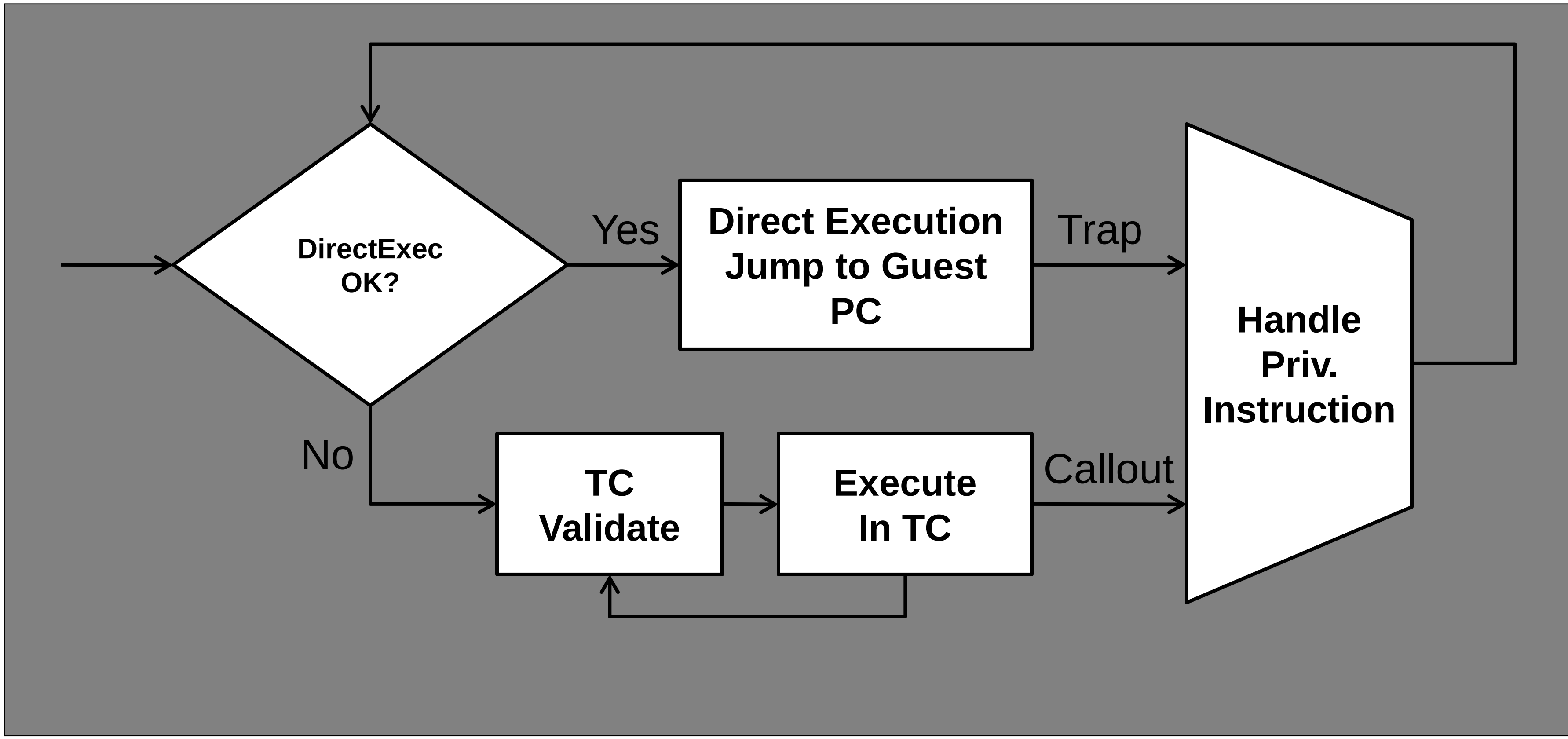
- Translation cache index data structure
- PC Synchronization on interrupts
- Self-modifying code
  - Notified on writes to translated guest code

Other Uses for Binary Translation

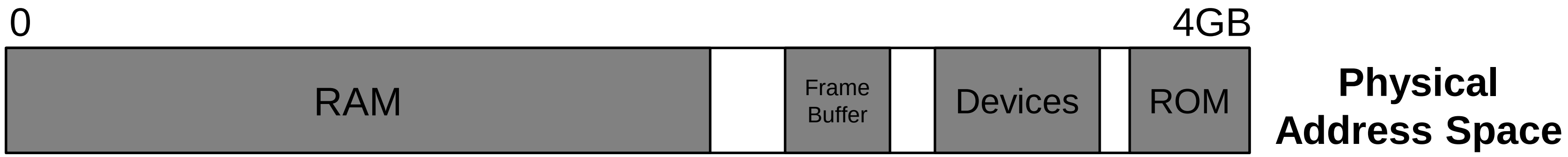
- Cross ISA translators
  - Digital FX!32
- Optimizing translators
  - H.P. Dynamo
- High level language byte code translators
  - Java
  - .NET/CLI

Hybrid Approach

- Binary Translation for the Kernel
- Direct Execution (Trap-and-emulate) for the User
- U.S. Patent 6,397,242



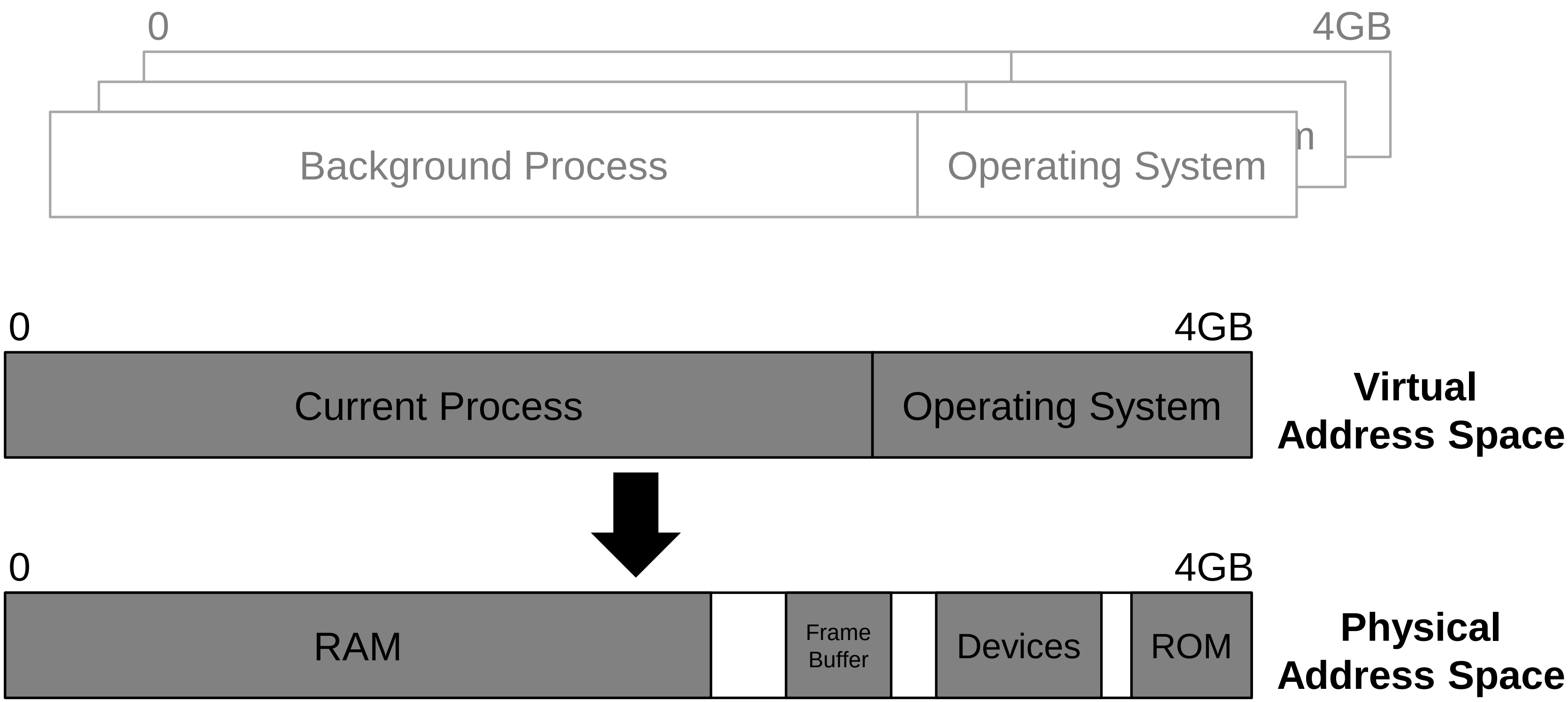
Traditional Address Spaces



*Memory Virtualization Basics **

*This presentation are based on the slides from Vmware
http://labs.vmware.com/academic/introduction-to-virtualization

Traditional Address Spaces



Memory Management Unit (MMU)

- Virtual Address to Physical Address Translation
  - Works in fixed-sized pages
  - Page Protection
- Translation Look-aside Buffer
  - TLB caches recently used Virtual to Physical mappings
- Control registers
  - Page Table location
  - Current ASID
  - Alignment checking

Types of MMUs

- Architected Page Tables

  x86, x86-64, ARM, IBM System/370, PowerPC

  – Hardware defines page table layout
  – Hardware walks page table on TLB miss

- Architected TLBs

  MIPS, SPARC, Alpha

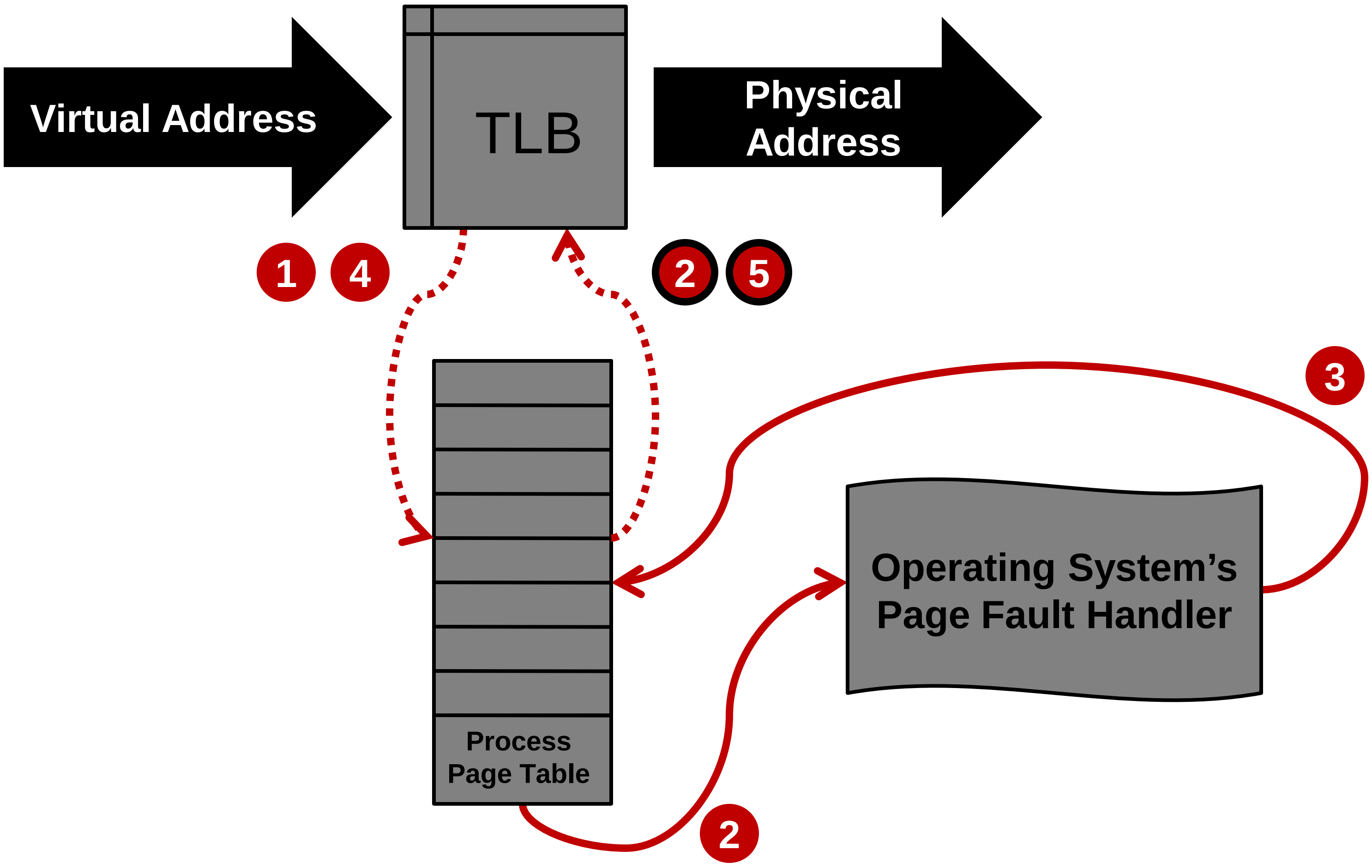
  – Hardware defines the interface to TLB
  – Software reloads TLB on misses
  – Page table layout free to software

- Segmentation / No MMU

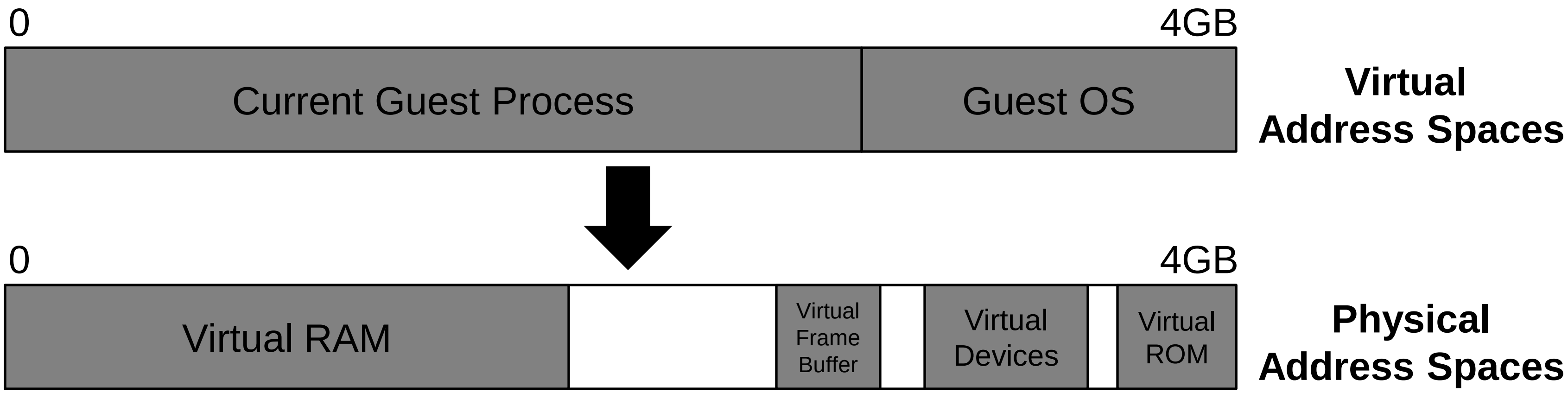
  Low-end ARMs, micro-controllers

  – Para-virtualization required

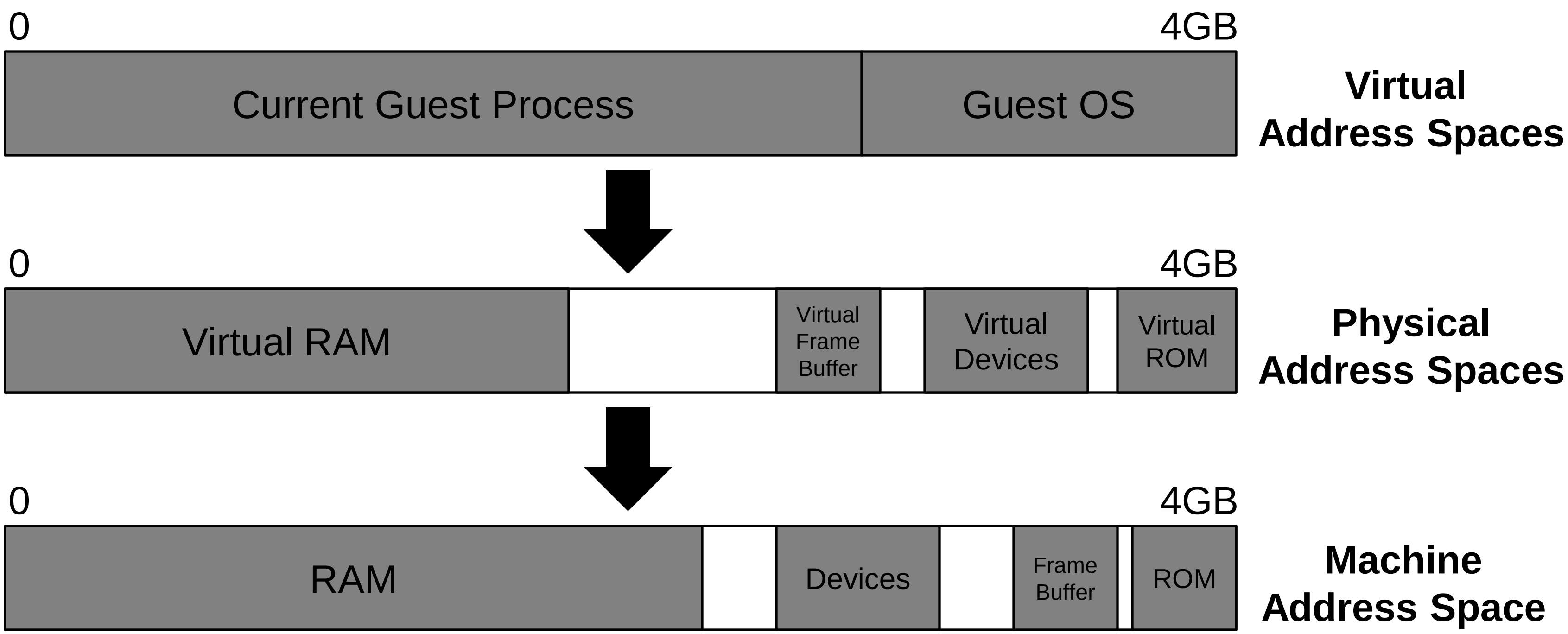
Traditional Address Translation w/Architected Page Tables



Virtualized Address Spaces



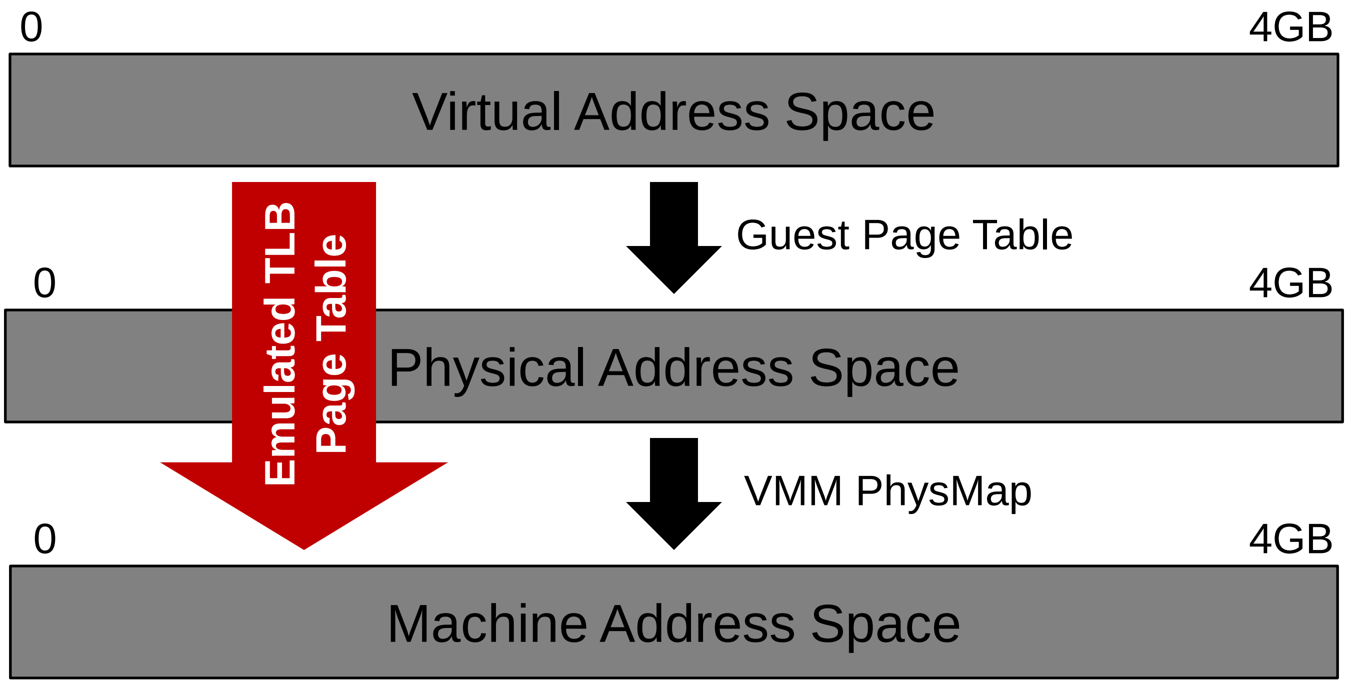
Virtualized Address Spaces



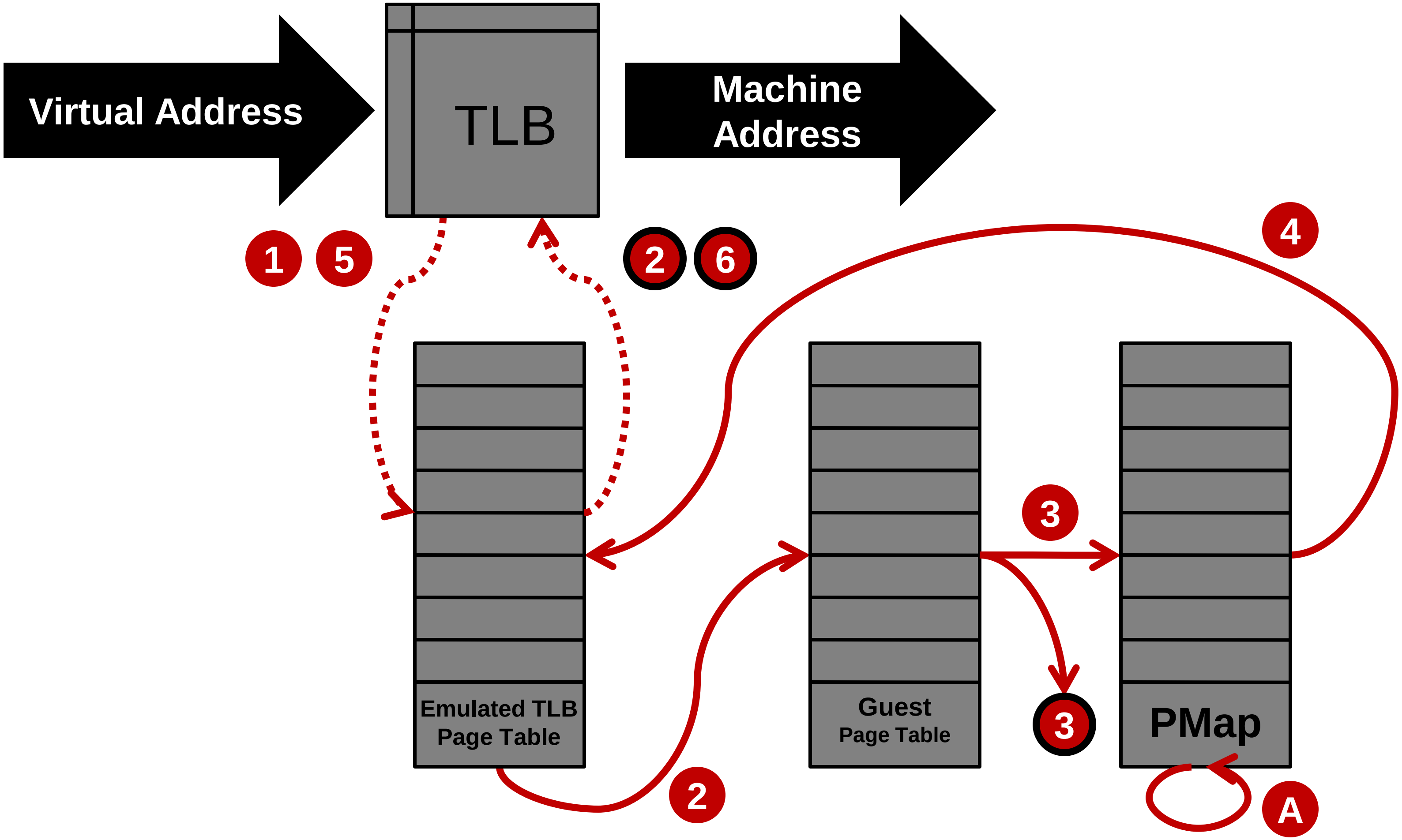
Outline

- Background
- Virtualization Techniques
  - Emulated TLB
  - Shadow Page Tables
- Page Protection
  - Memory Tracing
  - Hiding the Monitor
- Hardware-supported Memory Virtualization
  - Nested Page Tables

Virtualized Address Spaces w/ Emulated TLB



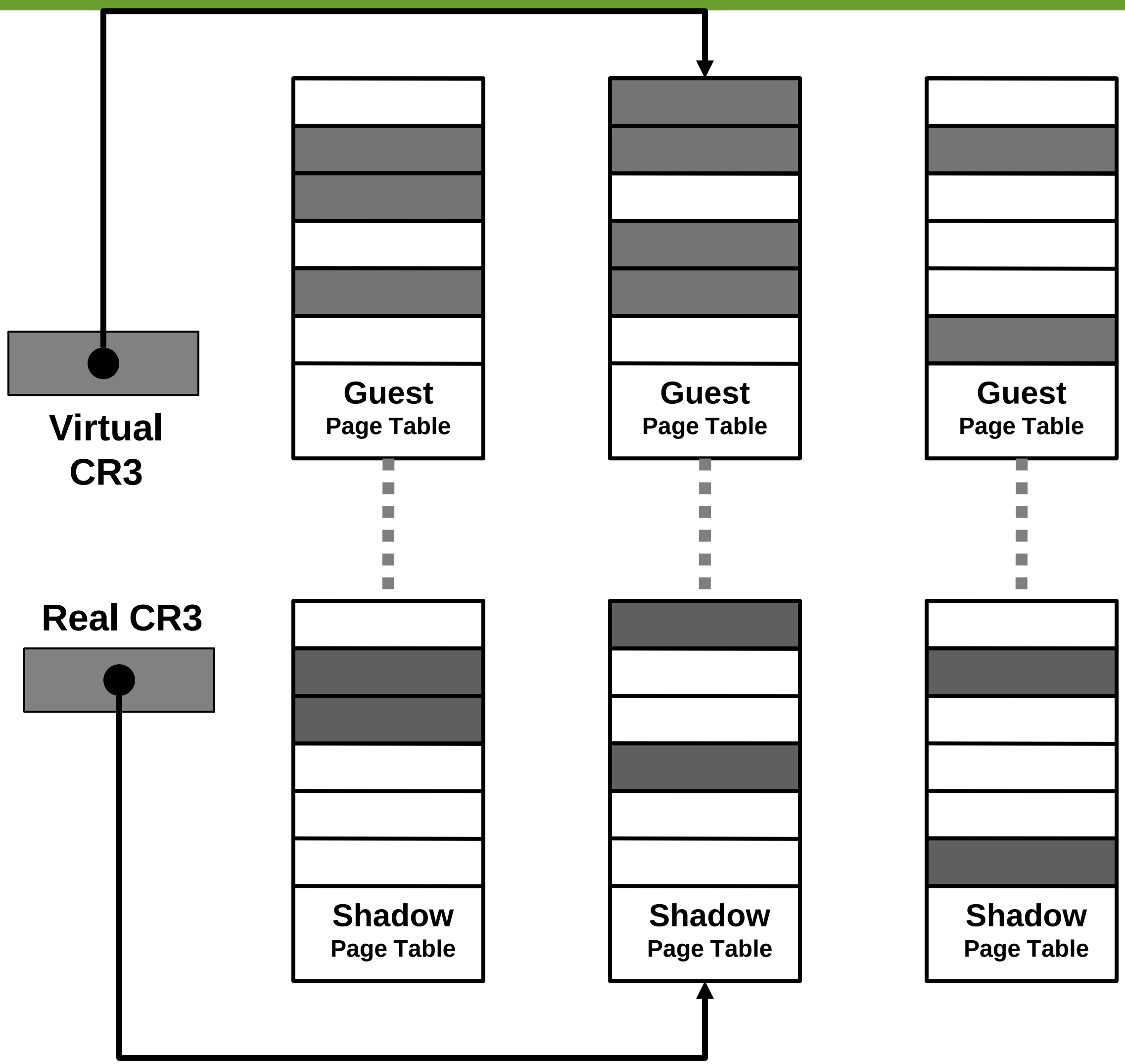
Virtualized Address Translation w/ Emulated TLB



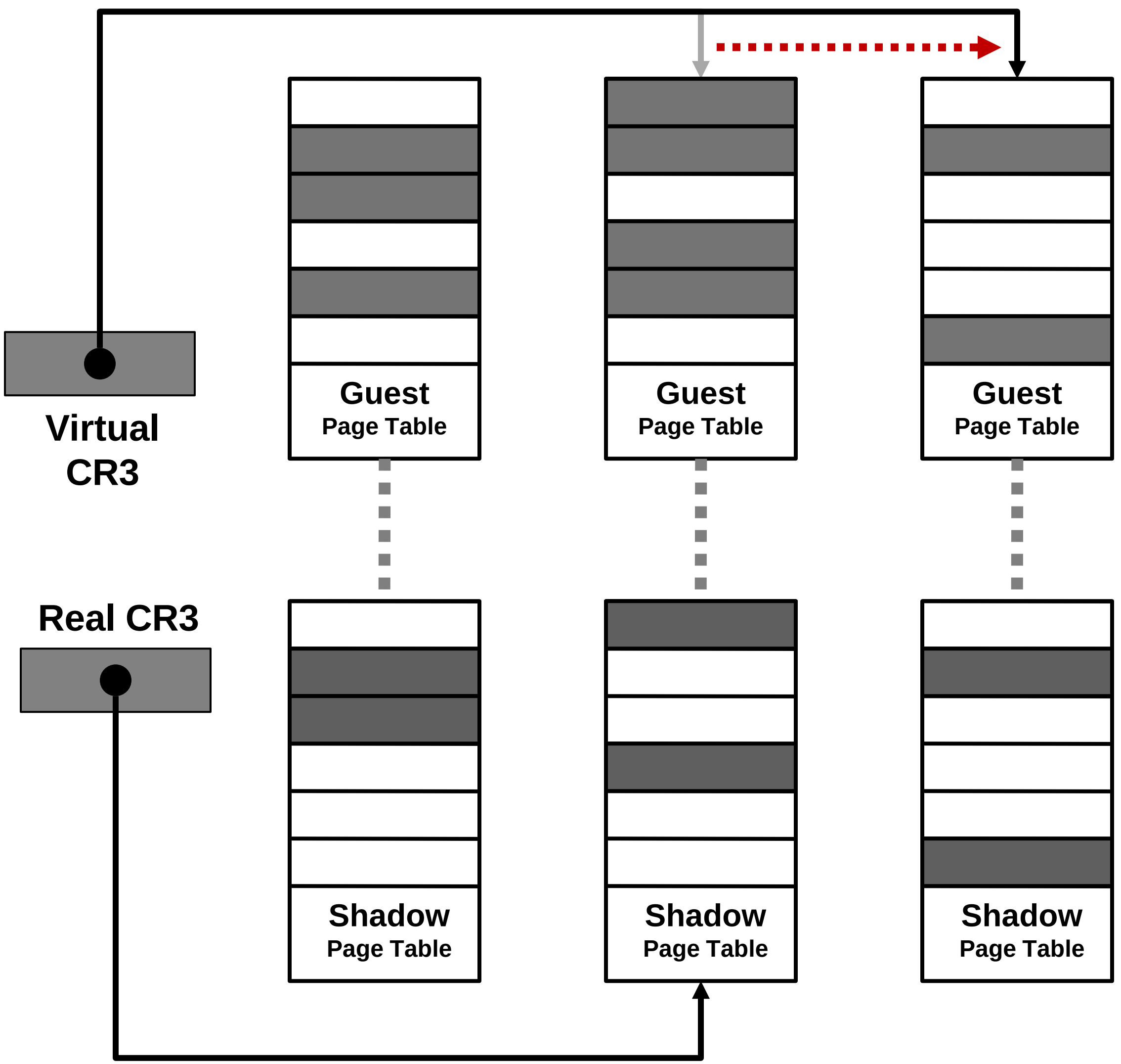
Issues with Emulated TLBs

- Guest page table consistency
  - Rely on Guest's need to invalidate TLB
  - Guest TLB invalidations caught by monitor, emulated
- Performance
  - Guest context switches flush entire software TLB

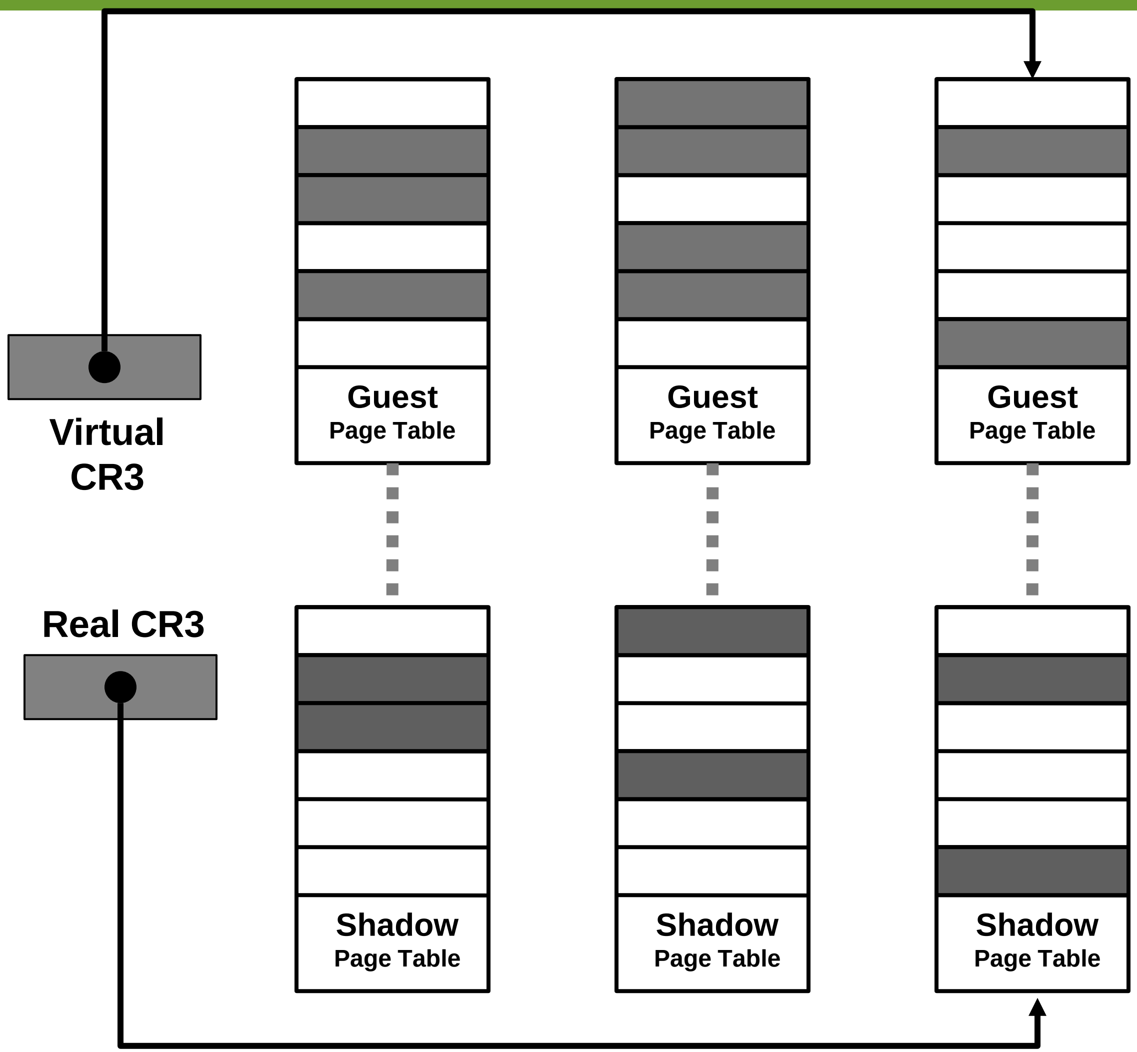
Shadow Page Tables



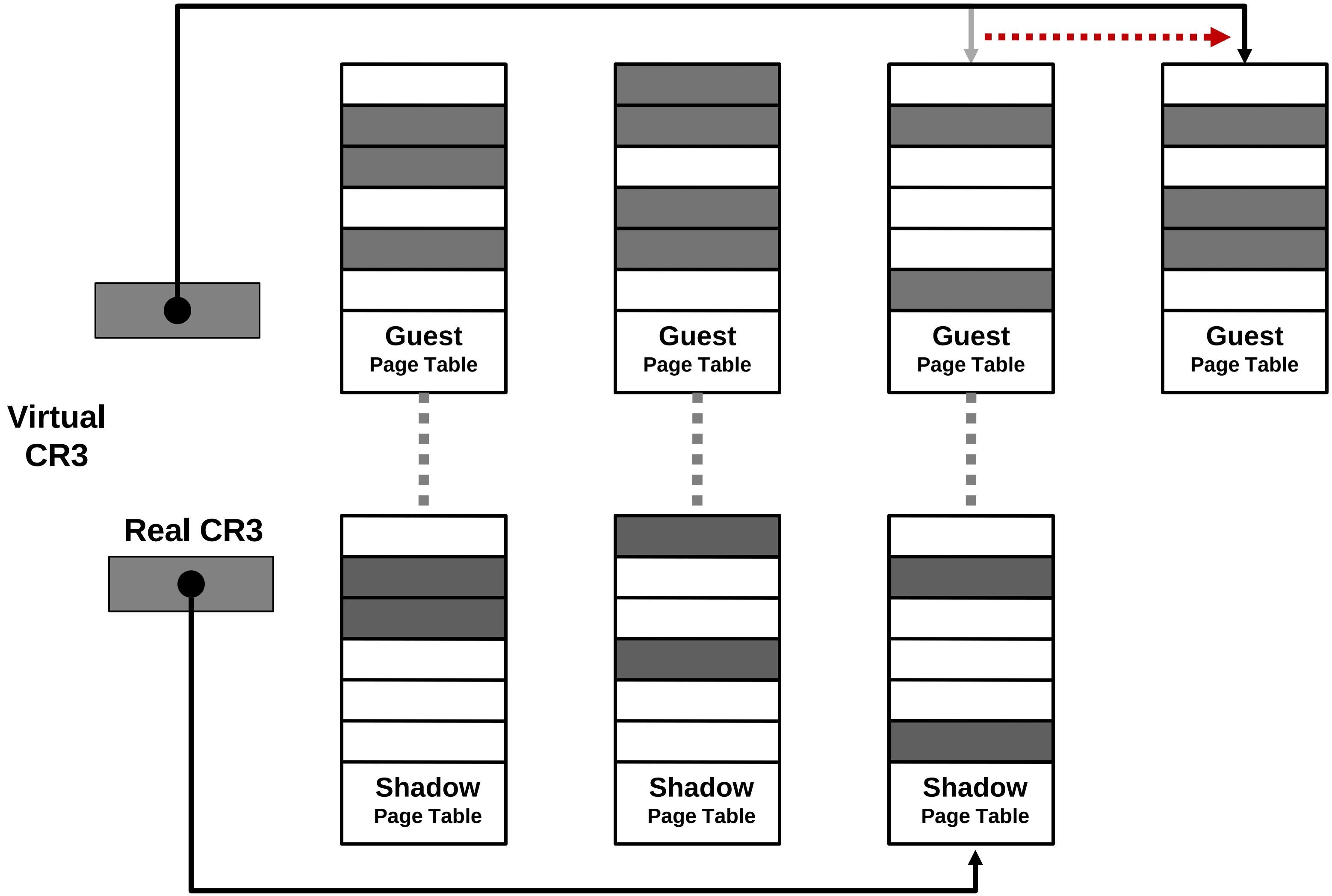
Guest Write to CR3



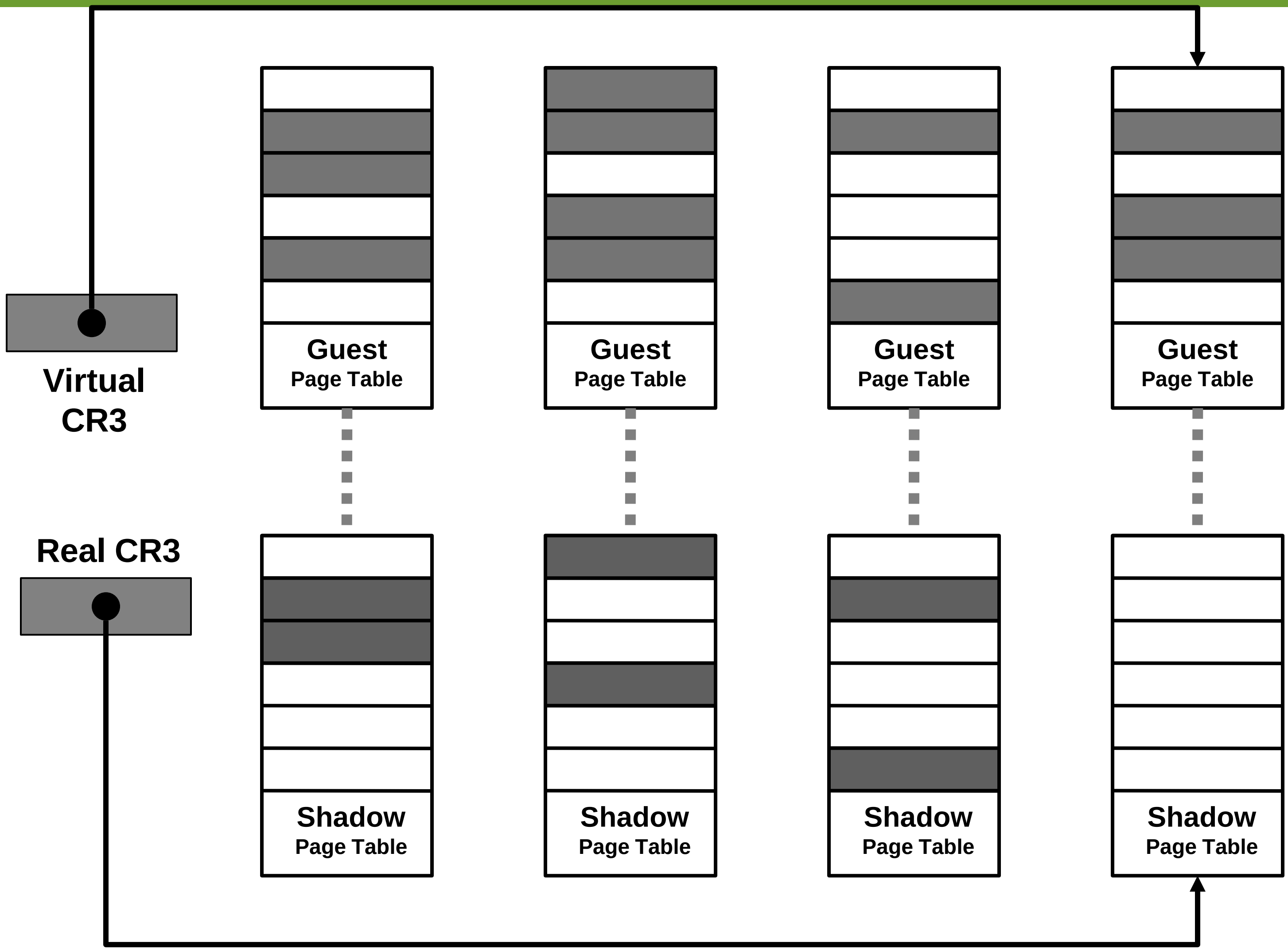
Guest Write to CR3



Undiscovered Guest Page Table



Undiscovered Guest Page Table



Issues with Shadow Page Tables

- Positives
  - Handle page faults in same way as Emulated TLBs
  - Fast guest context switching
- Page Table Consistency
  - Guest may not need invalidate TLB on writes to off-line page tables
  - Need to trace writes to shadow page tables to invalidate entries
- Memory Bloat
  - Caching guest page tables takes memory
  - Need to determine when guest has reused page tables

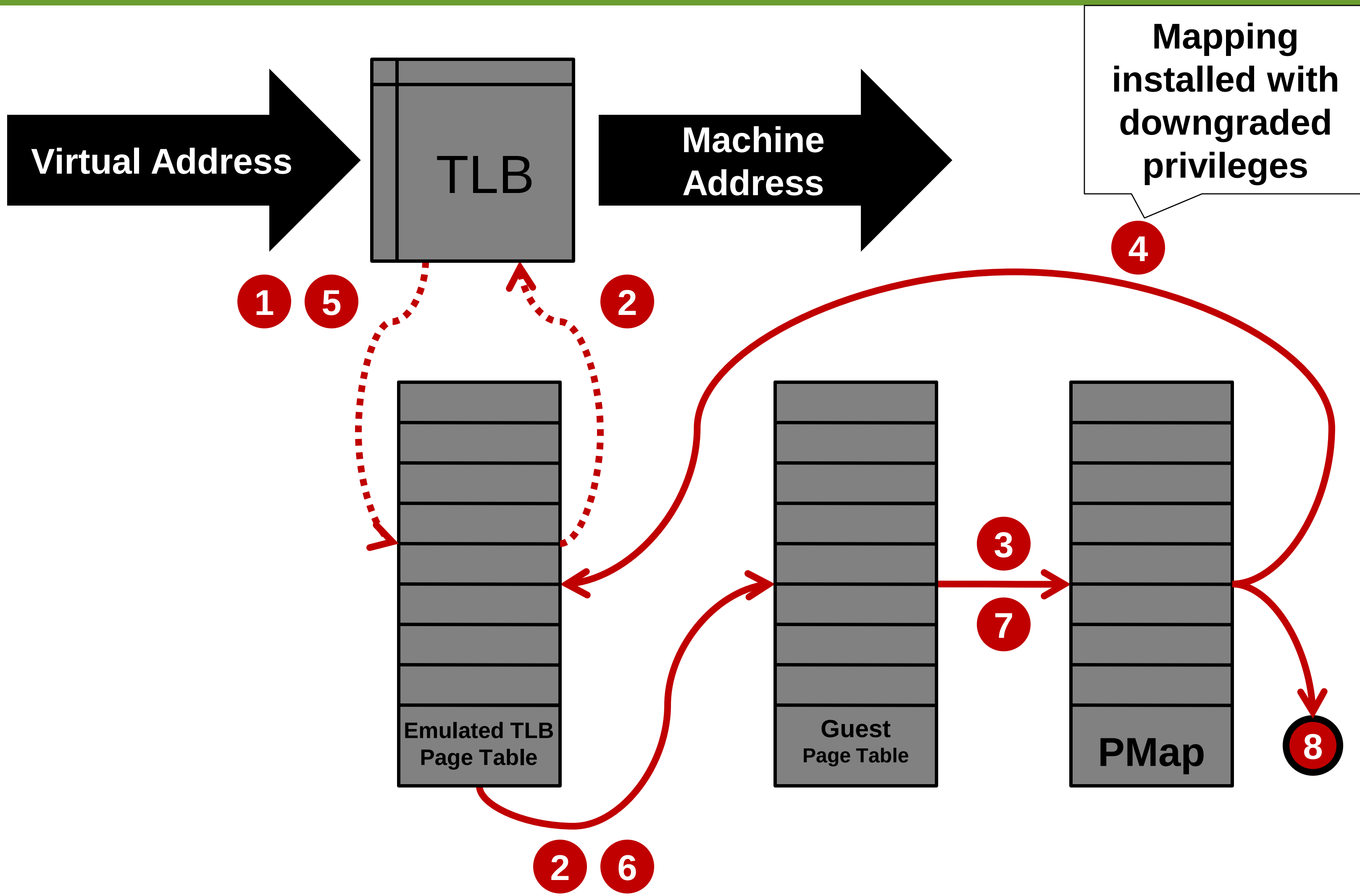
Memory Tracing

- Call a monitor handler on access to a traced page
  - Before guest reads
  - After guest writes
  - Before guest writes
- Modules can install traces and register for callbacks
  - Binary Translator for cache consistency
  - Shadow Page Tables for cache consistency
  - Devices
    - Memory-mapped I/O, Frame buffer
  - ROM
  - COW

Memory Tracing (cont.)

- Traces installed on Physical Pages
  - Need to know if data on page has changed regardless of what virtual address it was written through
- Use Page Protection to cause traps on traced pages
  - Downgrade protection
    - Write traced pages downgrade to read-only
    - Read traced pages downgrade to invalid

Trace Callout Path



Hiding the Monitor

- Monitor must be in the Virtual Address space
  - Exception / Interrupt handlers
  - Binary Translator
    - Translation Cache
    - Callout glue code
    - Register spill / fill locations
    - Emulated control registers

Hiding the Monitor – Options for Trap-and-Emulate

- Address space switch on Exceptions / Interrupts
  - Must be supported by the hardware
- Occupy some space in guest virtual address space
  - Need to protect monitor from guest accesses
    - Use page protection
  - Need to emulate guest accesses to monitor ranges
    - Manually translate guest virtual to machine
    - Emulate instruction
      - Must be able to handle all memory accessing instructions

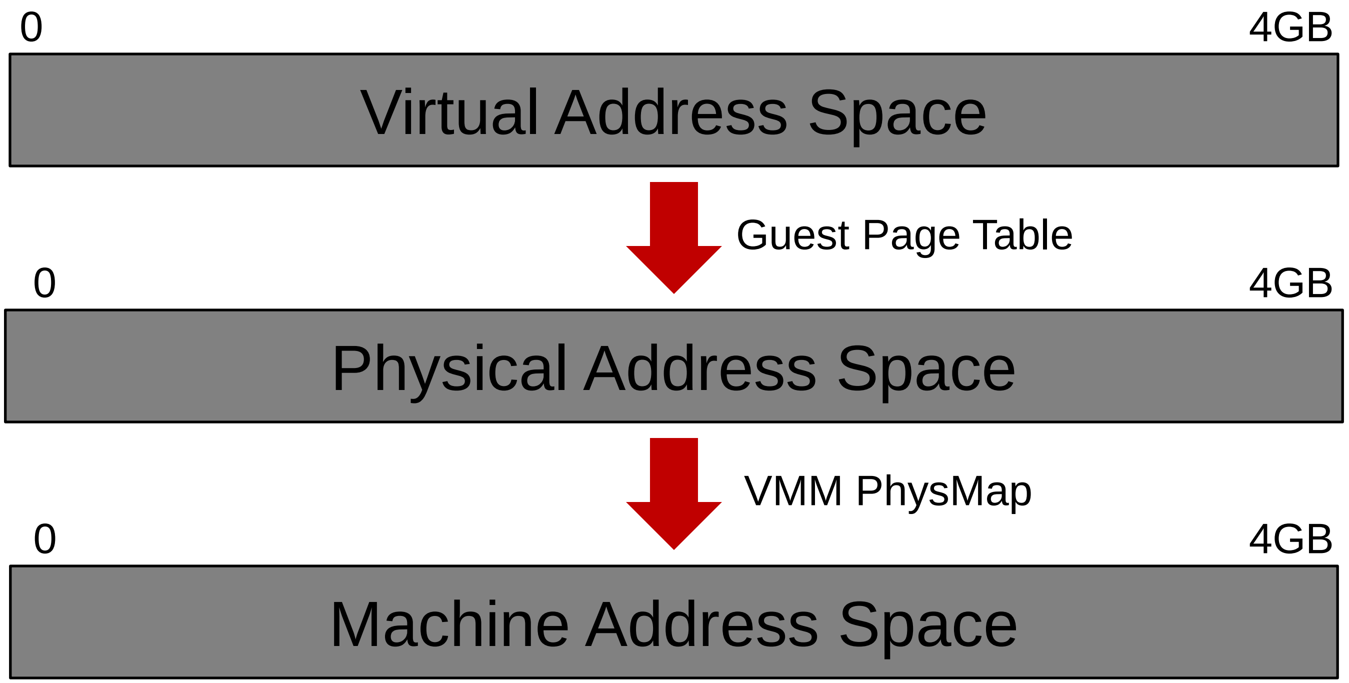
Hiding the Monitor – Options for Binary Translation

- Translation cache intermingles guest and monitor memory accesses
  - Need to distinguish these accesses
  - Monitor accesses have full privileges
  - Guest accesses have lesser privileges
- On x86 can use segmentation
  - Monitor lives in high memory
  - Guest segments truncated to allow no access to monitor
  - Binary translator uses guest segments for guest accesses and monitor segments for monitor accesses

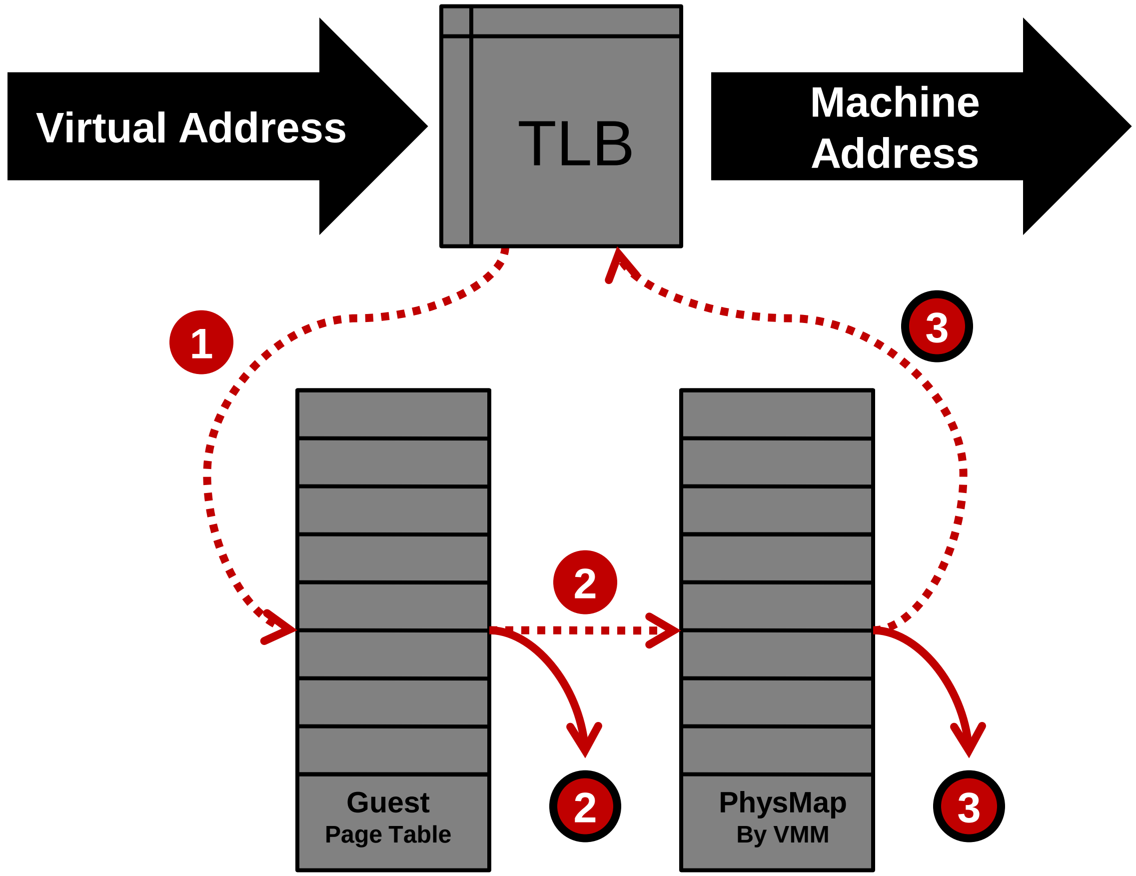
Outline

- Background
- Virtualization Techniques
  - Emulated TLB
  - Shadow Page Tables
- Page Protection
  - Memory Tracing
  - Hiding the Monitor
- Hardware-supported Memory Virtualization
  - Nested 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 monitor

Interposition with Memory Virtualization Page Sharing