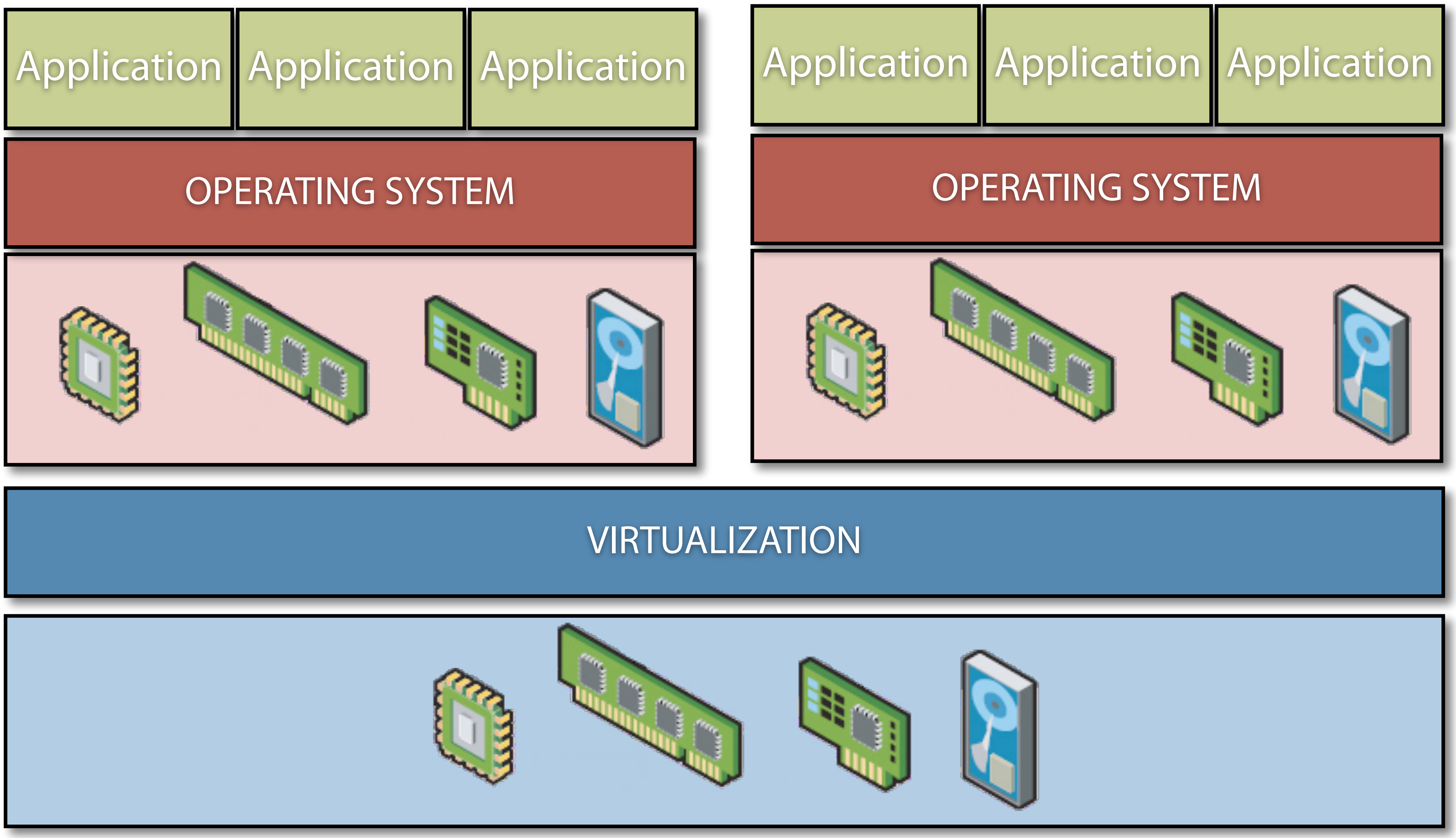
Virtualization



Basic Idea

- **Observation**
  - Hardware resources are typically under-utilized
  - Hardware resources directly relate to cost
- **Goal**
  - Improve hardware utilization
- **How**
  - Share hardware resources across multiple machines
  - May make sense for network attached storage, but what about processor, memory, etc.?
- **Approach**
  - Decouple machine from hardware
- **Virtual Machine** (VM)
  - A machine decoupled from the hardware, i.e. does not necessarily correspond to the hardware
  - Multiple "Virtual Machines" on the same physical host could share the underlying hardware
  - First VM: IBM System/360 Model 40 VM [1965]

Why?

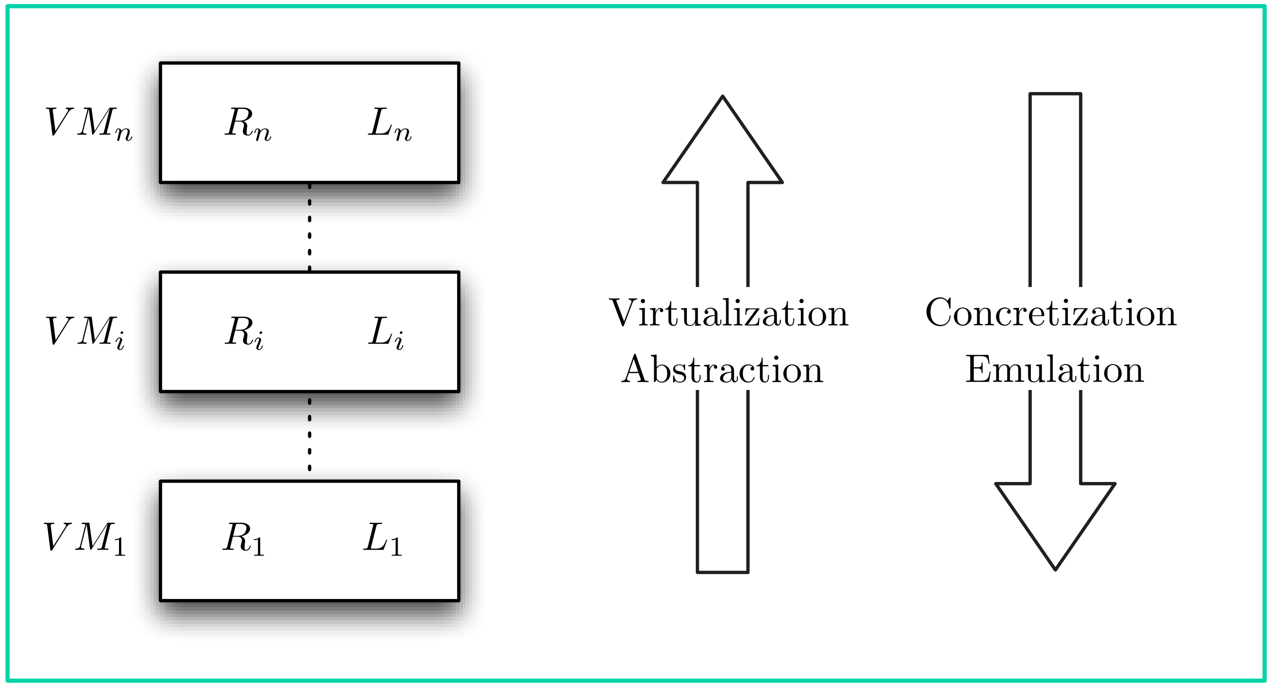
- **Consolidate resources**
  - Server consolidation
  - Client consolidation
- **Improve system management**
  - For both hardware and software
  - From the desktop to the datacenter
- **Improve software lifecycle**
  - Develop, debug, deploy and maintain applications in virtual machines
- **Increase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

Modern Computer Systems

- **Modern computer system is very complex**
  - Hundreds of millions of transistors
  - Interconnected high-speed I/O devices
  - Networking infrastructures
  - Operating systems, libraries, applications
  - Graphics and networking software
- **To manage this complexity: Levels of Abstractions**
  - Allows implementation details at lower levels of design to be ignored or simplified
  - Each level is separated by well-defined interfaces, so that the design of a higher level can be decoupled from the lower levels

Layers of Abstraction

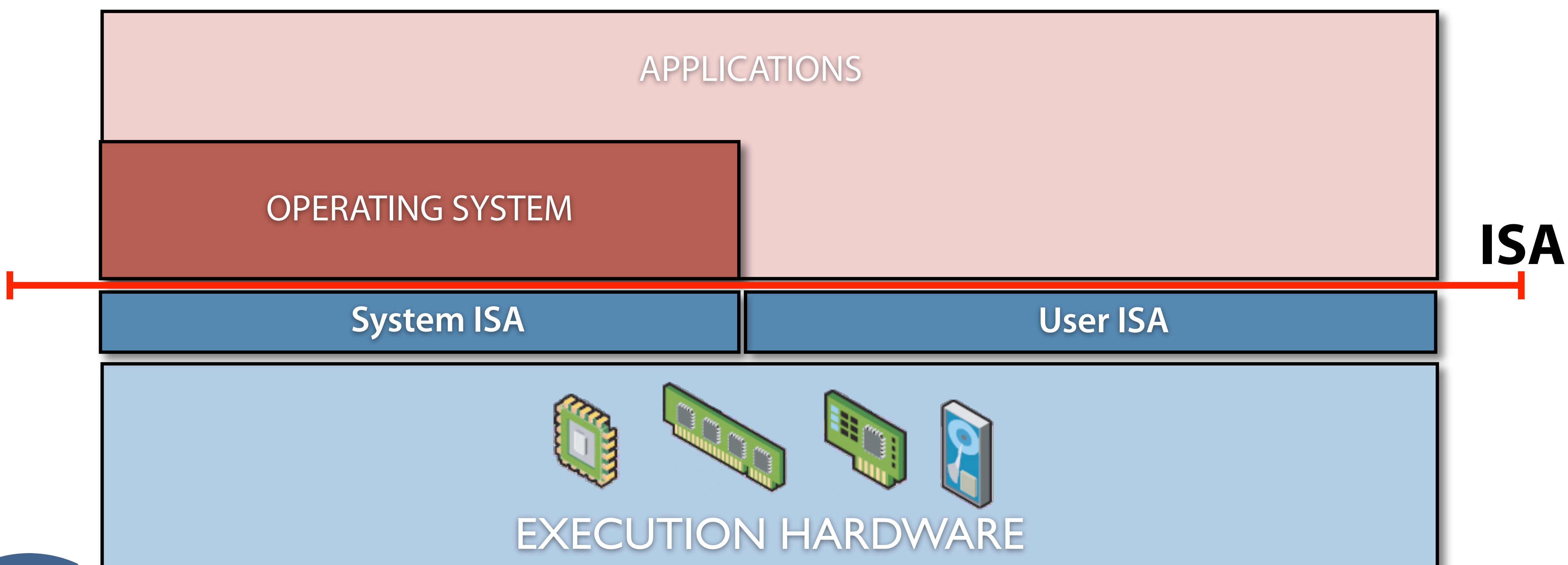
- **Abstraction**
  - used to manage complexity
  - typically defined in layers (VMi)
  - each layer has its own language (Li) and data structures (Ri)
  - lowest layers implemented in hardware
  - higher layers implemented in software
- **Machine: denotes the system on which software is executed.**
  - to an operating system this is generally the physical system
  - to an application program a machine is defined by the combination of hardware and OS-implemented abstractions



- **Typical Layers**
  - VM4: Applications
  - VM3: Operating System
  - VM2: Assembler Machine
  - VM1: Firmware Machine
  - VM0: Hardware Machine

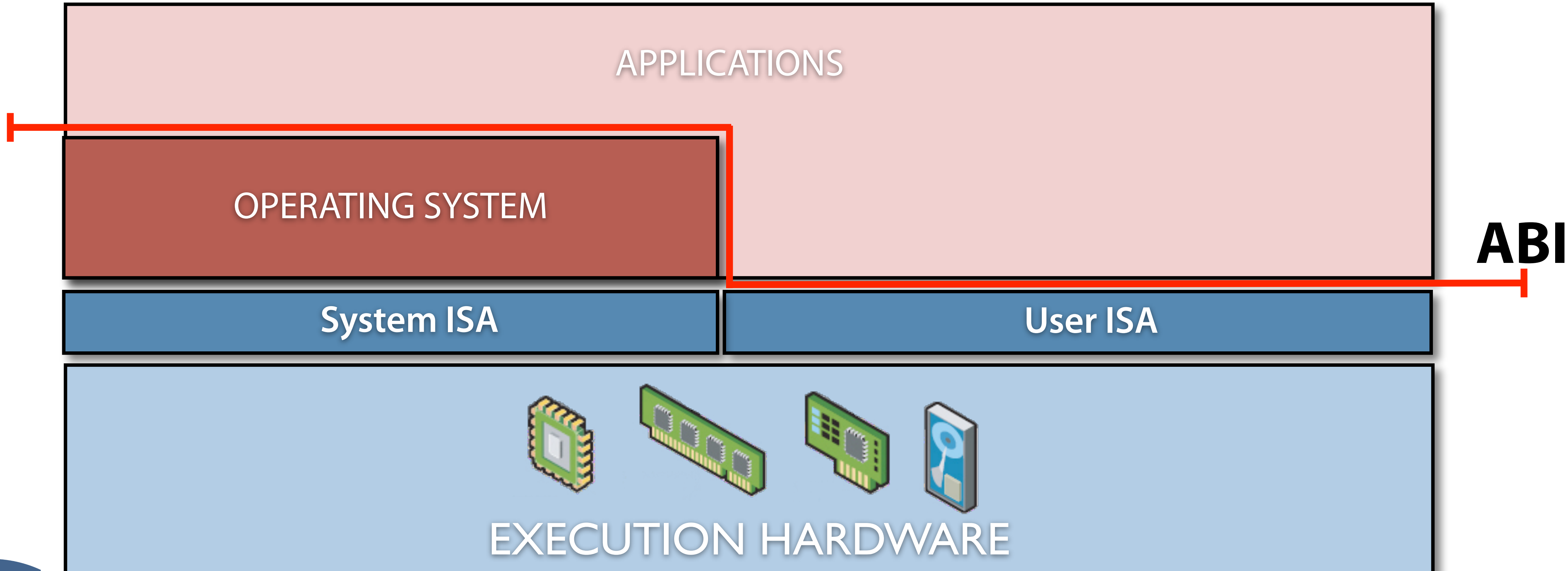
Machine Level Abstraction

- **Instruction Set Architectures (ISA)**
  - Defines hardware/software boundary
  - **User ISA**: portion of architecture visible to an application programmer
  - **System ISA**: portion of architecture visible to the supervisor software (i.e., OS)
- **For OS developers, a machine is defined by ISA**



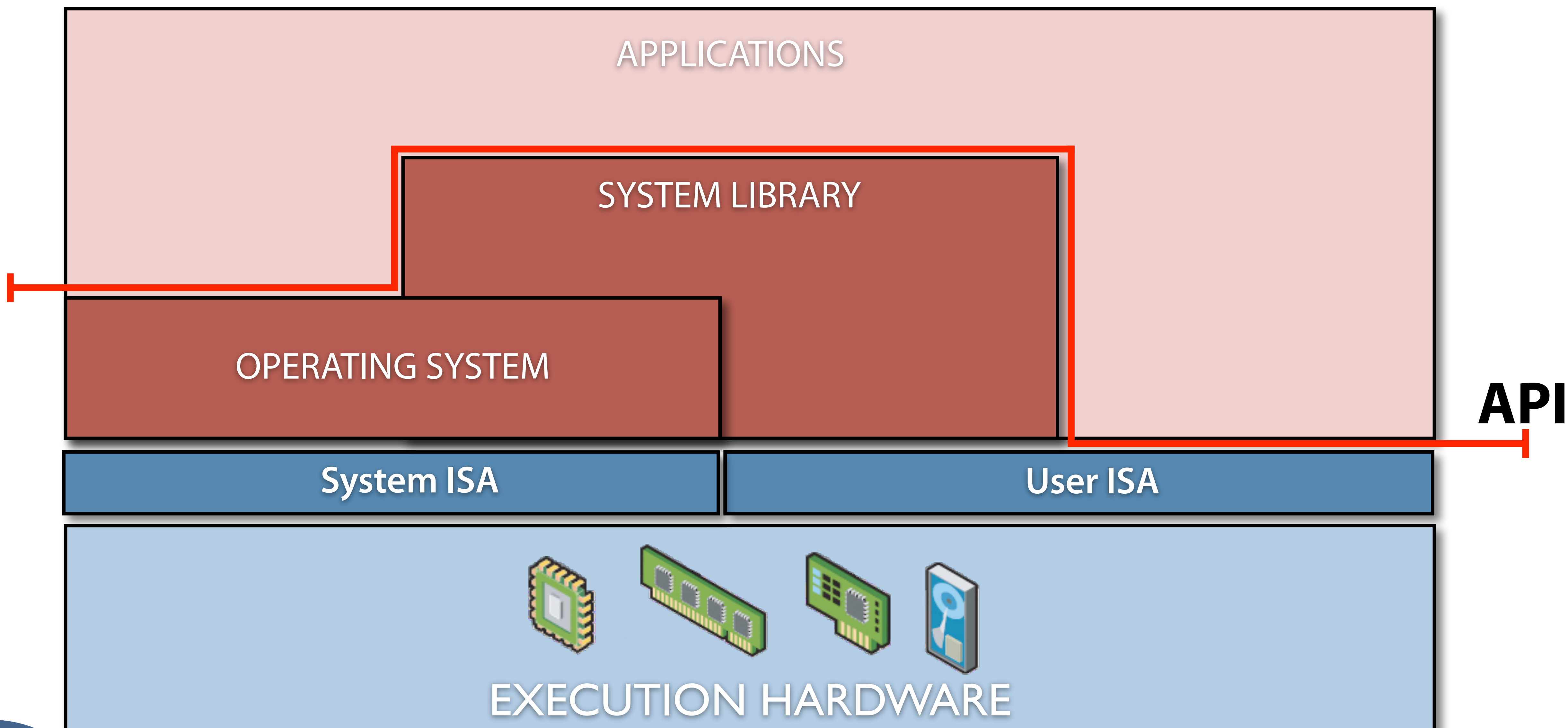
OS Level Abstraction

- **Application Binary Interface (ABI)**
  - Defines program interface to hardware resources and services
  - **User ISA**
    - System instructions are not included in the ABI
    - User instructions allow program to direct access hardware
  - **System calls**
    - Indirect interface for accessing shared system resources and services
    - implemented by the supervisor software
- **For compiler/library developers, a machine is defined by ABI**



Application Level Abstraction

- **Application Programming Interface (API)**
  - Defined in terms of an high level language (e.g., C)
  - Typically implemented as a system library (e.g., libc)
- **For application developers , a machine is defined by API**



What is a machine?

- A machine is an entity that provides an interface
  - **Program** view:

    Machine = Entity that provides the API
  - **Process** view:

    Machine = Entity that provides the ABI
  - **Operating system** view:

    Machine = Entity that provides the ISA

What is a virtual machine?

- Virtual machine is an entity that emulates a guest interface on top of a host machine
  - **Program** view:

    Virtual machine = Entity that emulates an API on top of another

    Virtualizing software = compiler/interpreter
  - **Process** view:

    Virtual Machine = Entity that emulates an ABI on top of another

    Virtualizing software = runtime
  - **Operating system** view:

    Virtual Machine = Entity that emulates an ISA

    Virtualizing software = virtual machine monitor (VMM)

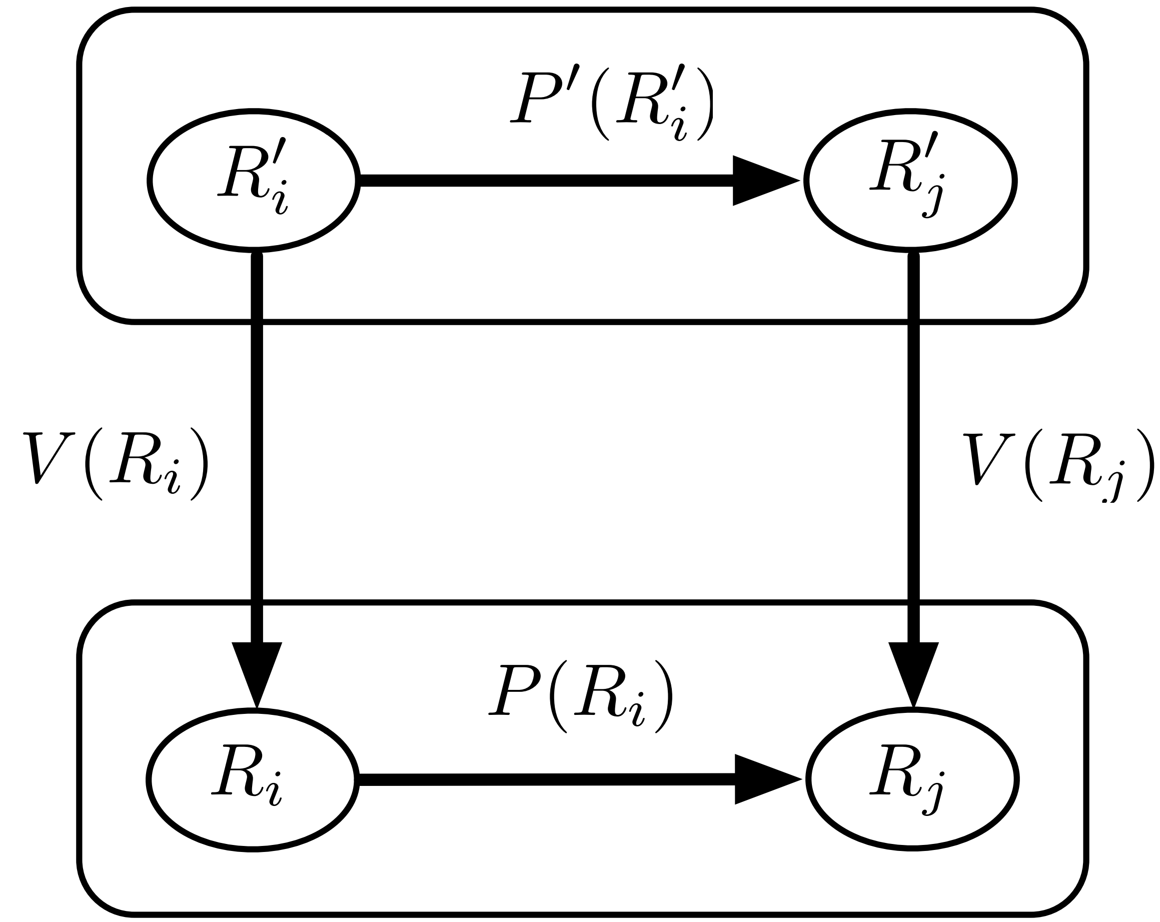
Formal Definition

- Virtualization is defined as the construction of an isomorphism that maps a **guest** machine to an existing **host** machine such that:
  - maps the guest state R_i (collection of guest virtualization objects) onto the host state R_i' through some function V() such that

$$V(R_i) = R_i'$$

  - for every policy P() transforming the state R_i in state R_j in the guest, there is a corresponding policy P'() in the host that performs an equivalent modification of the host state

$$P' \circ V(R_i) = V \circ P(R_i)$$

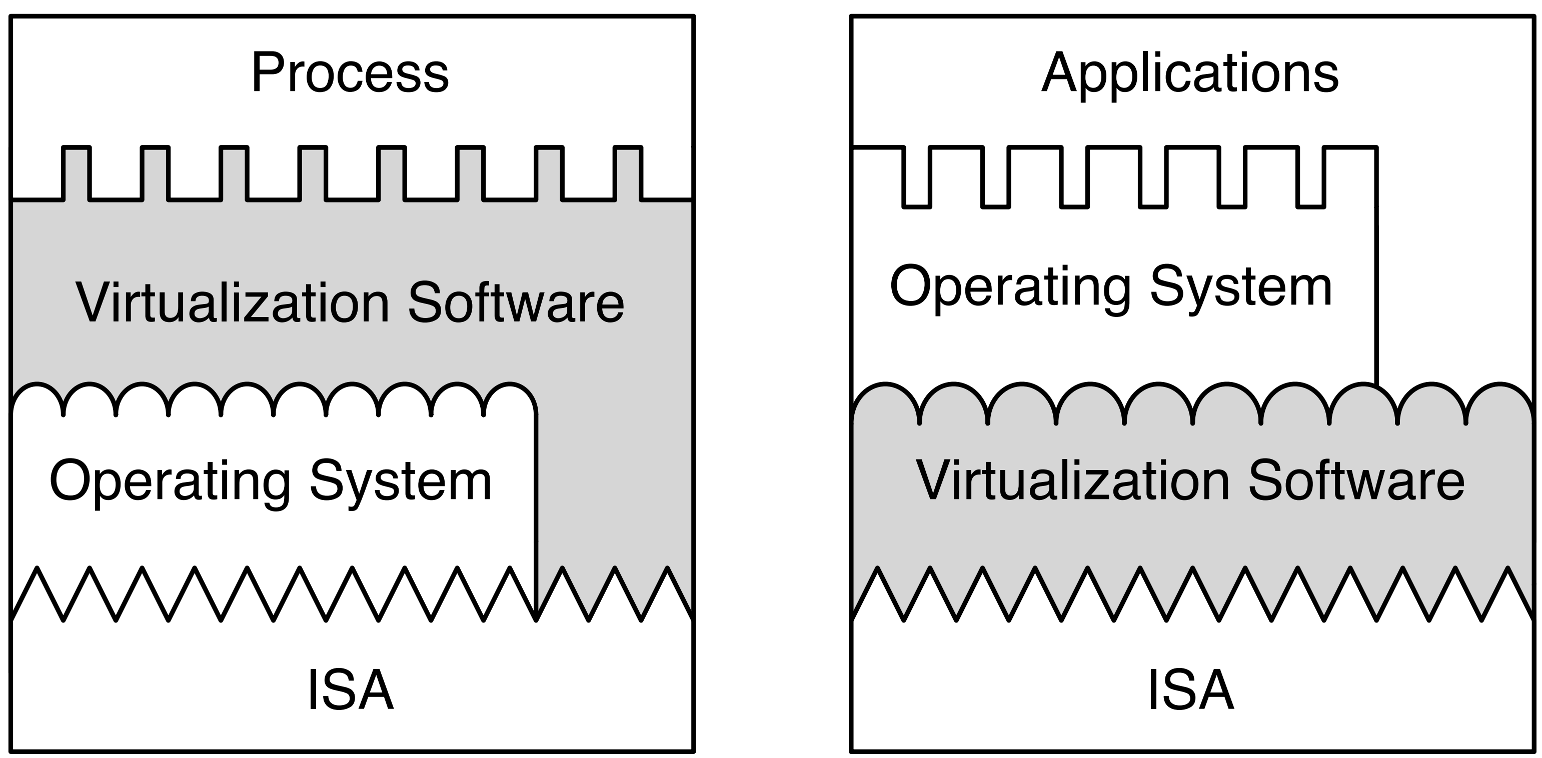


Perspectives

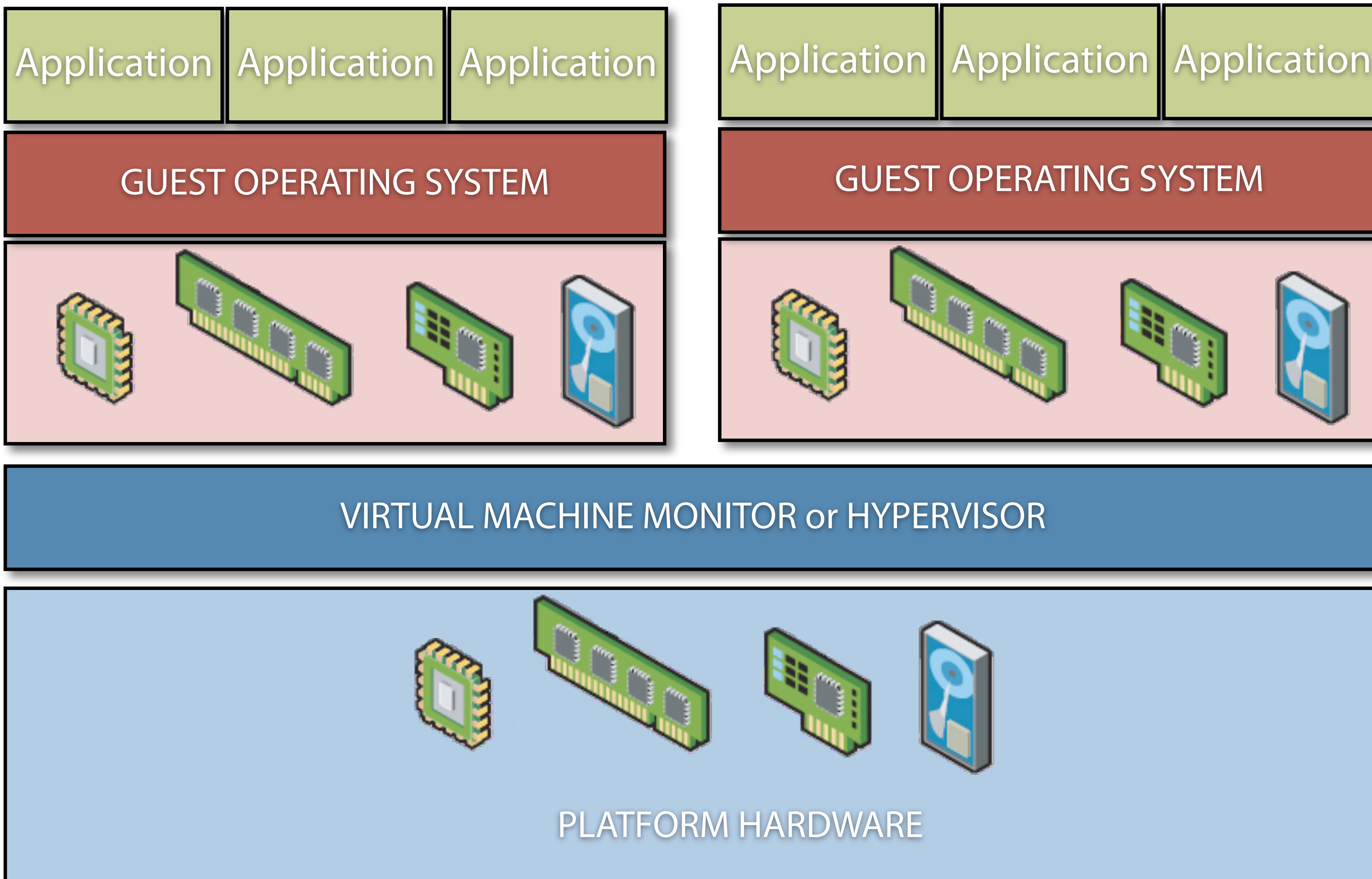
- ***Process perspective*:** the system ABI defines the interface between the process and machine
  - user-level hardware access: logical memory space, user-level registers and instructions
  - OS mediated: Machine I/O or any shared resource or operations requiring system privilege.
- ***Operating system perspective*:** ISA defines the interface between OS and machine
  - system is defined by the underlying machine
  - direct access to all resources
  - manage sharing
- Virtual machine executes software (process or operating system) in the same manner as target machine
  - Implemented with both hardware and software
  - VM resources may differ from that of the physical machine
  - Generally not necessary for VM to have equivalent performance

Where is the VM?

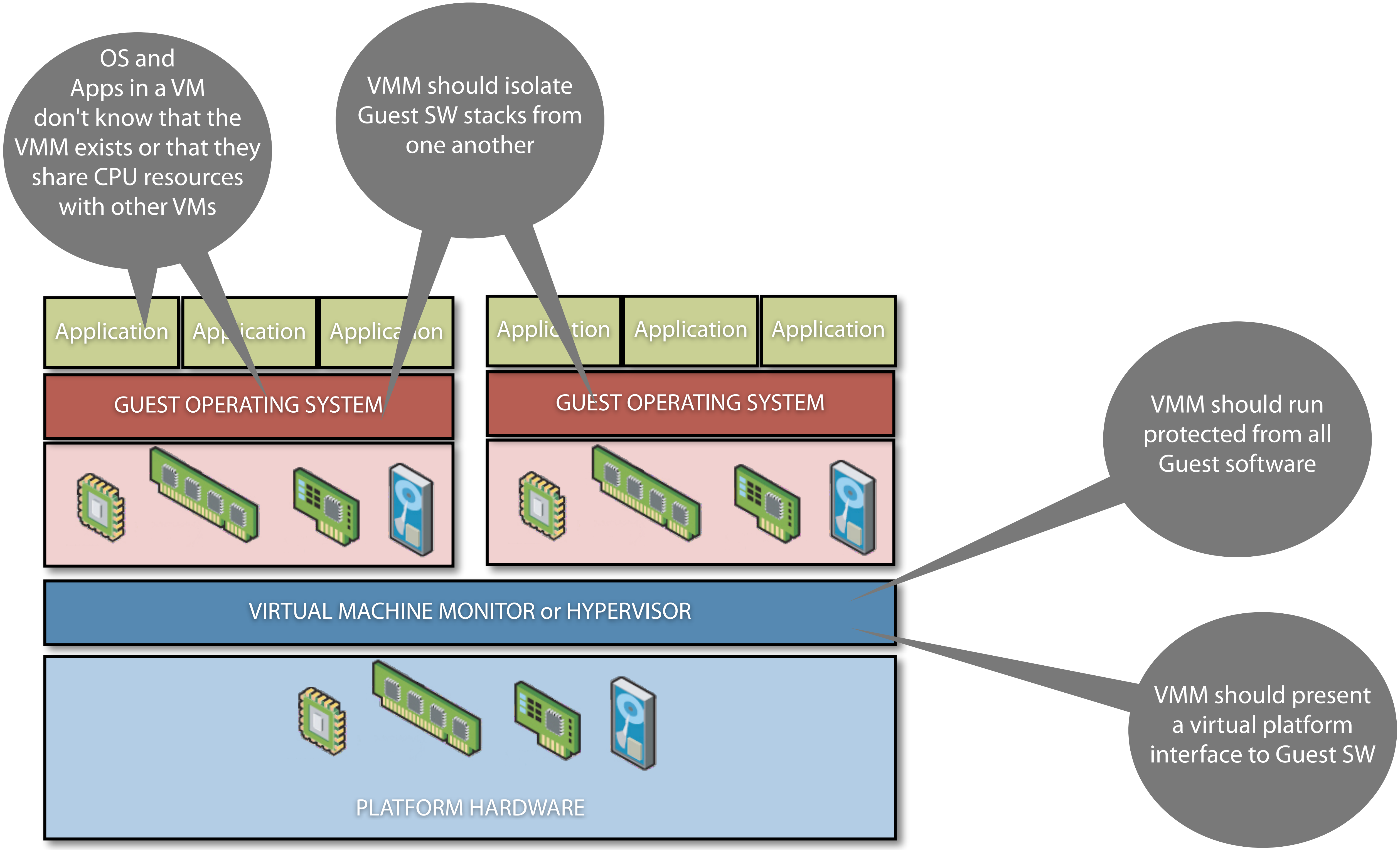
- ***Process virtual machine*:** supports an individual process
  - Emulates user-level instructions and operating system calls
  - Virtualizing software placed at the ABI layer
- ***System Virtual Machines*:** emulates the target hardware ISA
  - guest and host environment may use the same ISA
- Virtual Machines are implemented as combination of
  - Real hardware
  - Virtualizing software



Virtual Machine Monitor

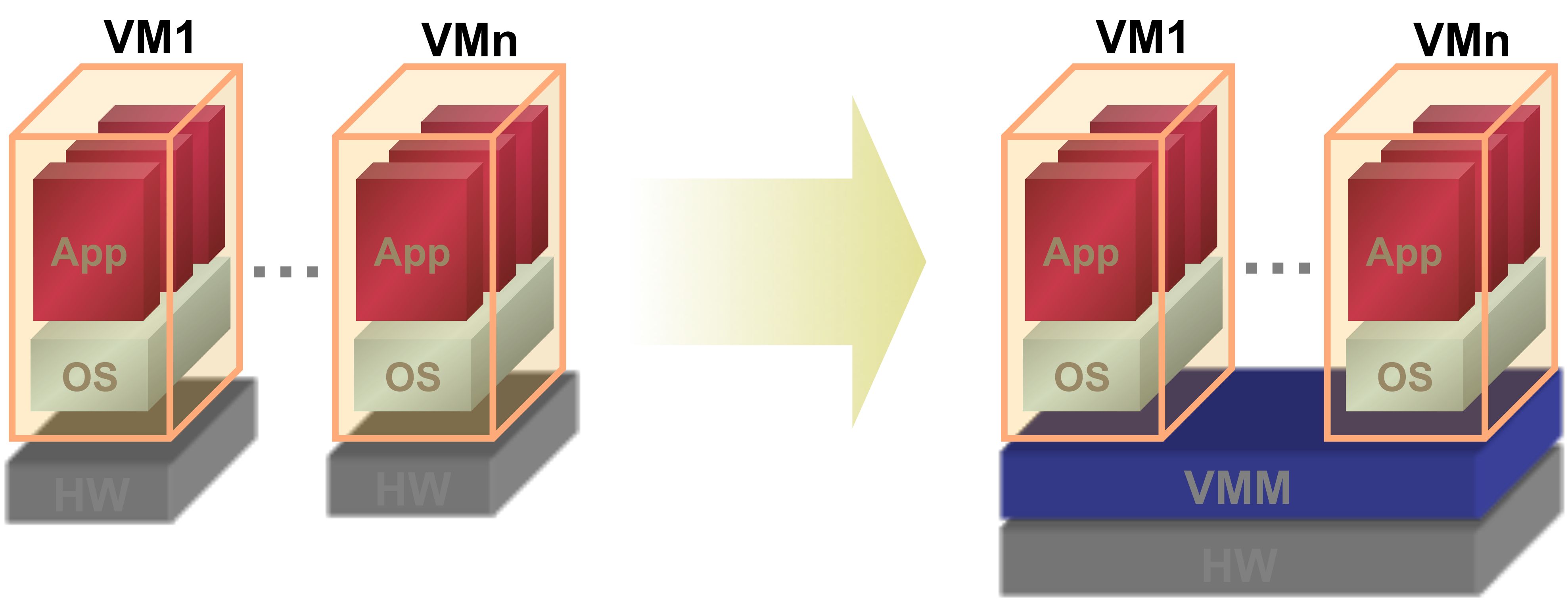


VMM Challanges



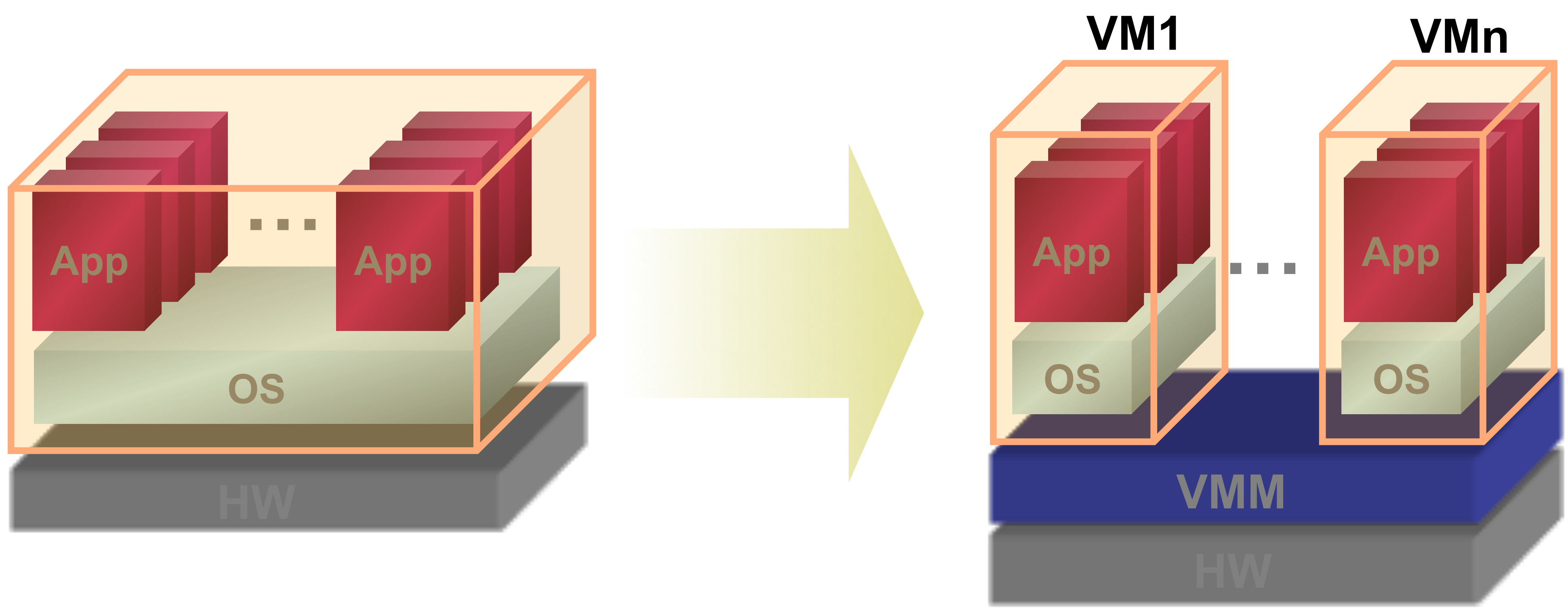
VM Usage: Workload Consolidation

- Server virtualization consolidates many systems onto one physical platform
- Pros
  - Each application can run in a separate environment delivering true isolation
  - Cost Savings: power, space, cooling, hardware, software and management
  - Ability to run legacy applications in legacy OSs
  - Ability to run through emulation legacy applications in legacy HW
- Cons
  - Disk and memory footprint increase due to multiples OSs
  - Performance penalty caused by resource sharing management



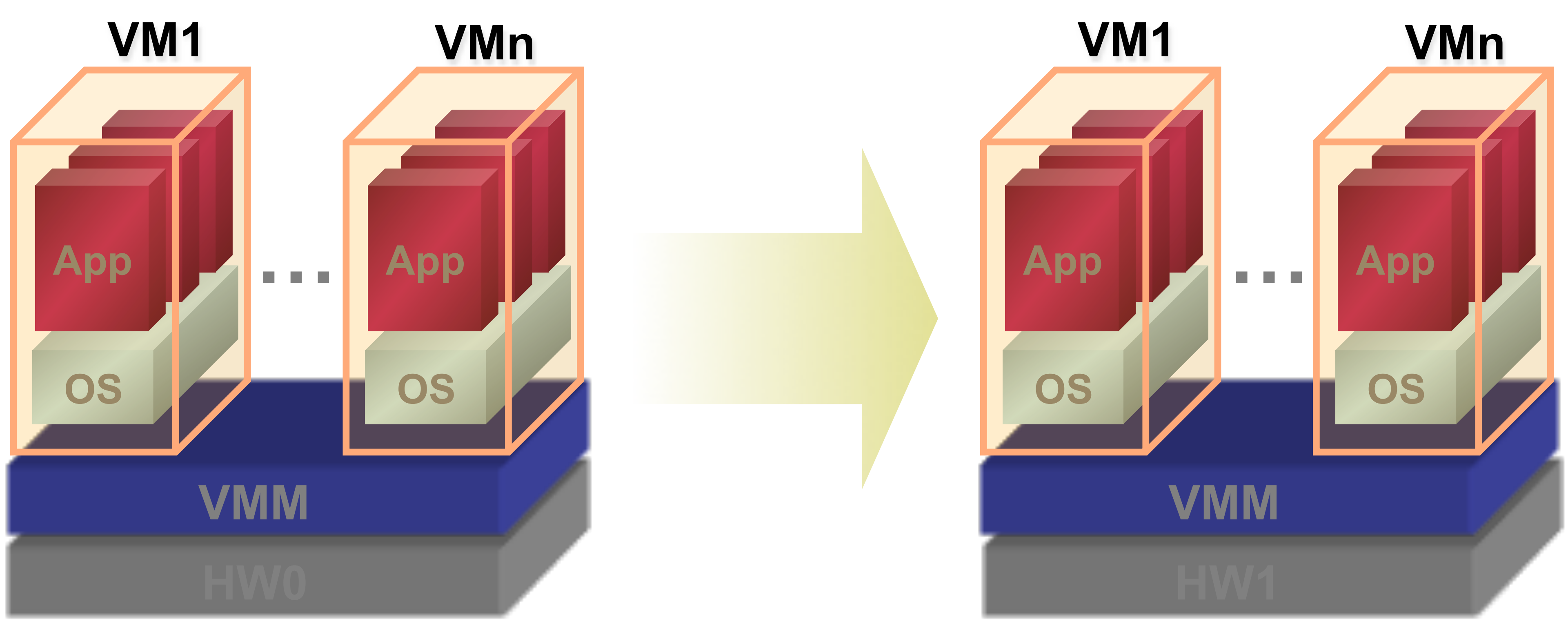
VM Usage: Workload Isolation

- Virtualization can improve overall system security and reliability by isolating multiple software stacks in their own VMs
  - Security: intrusions can be confined to the VM in which they occur
  - Reliability: software failures in one VM do not affect the other VMs
  - As a side effect, if the hypervisor or drivers are compromised, the whole VMs can be compromised (equivalent to BIOS attack)



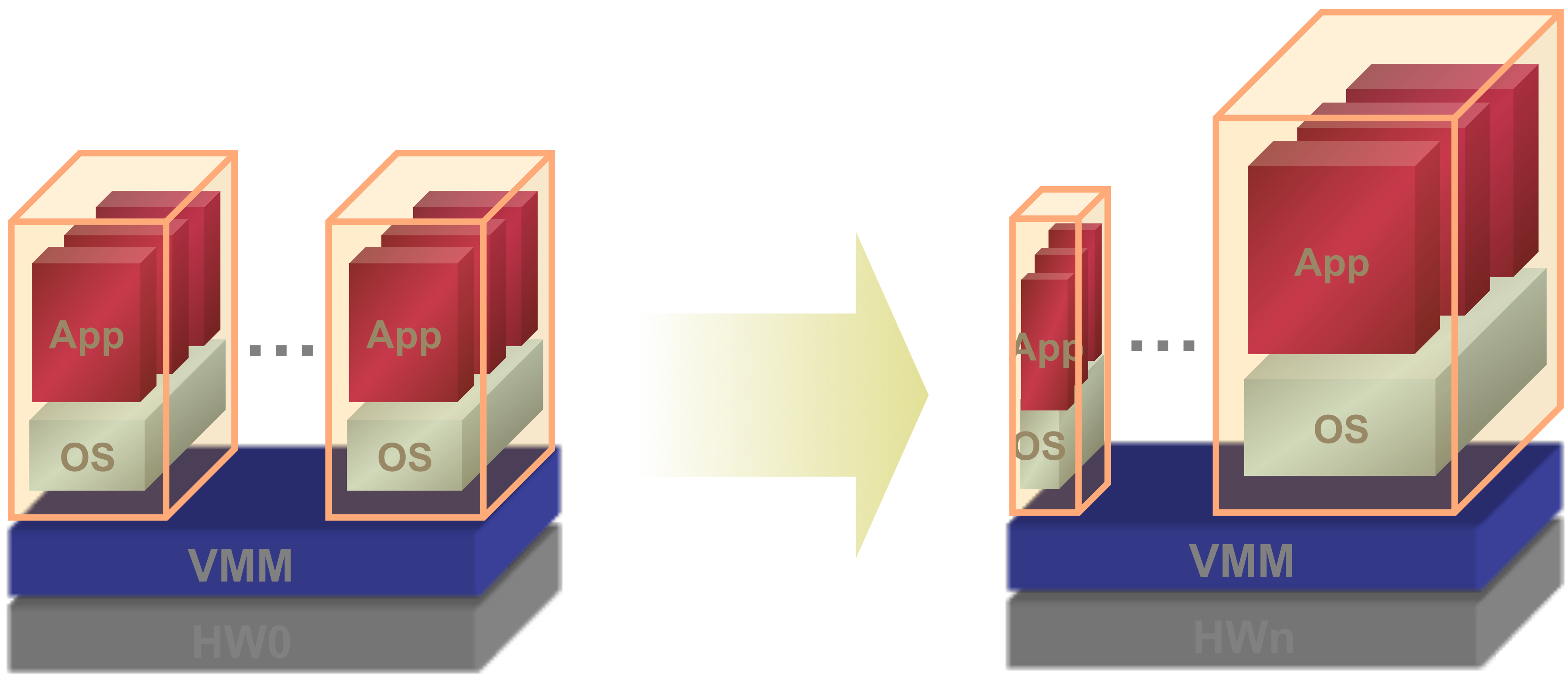
VM Usage: Workload Isolation (I)

- Migrate (move) running VMs to a different platform
  - It facilitates hardware maintenance operations
  - Both at server and data-center level
  - High Availability: if an application goes down, it is not necessary to wait for the reboot of the operating system/application



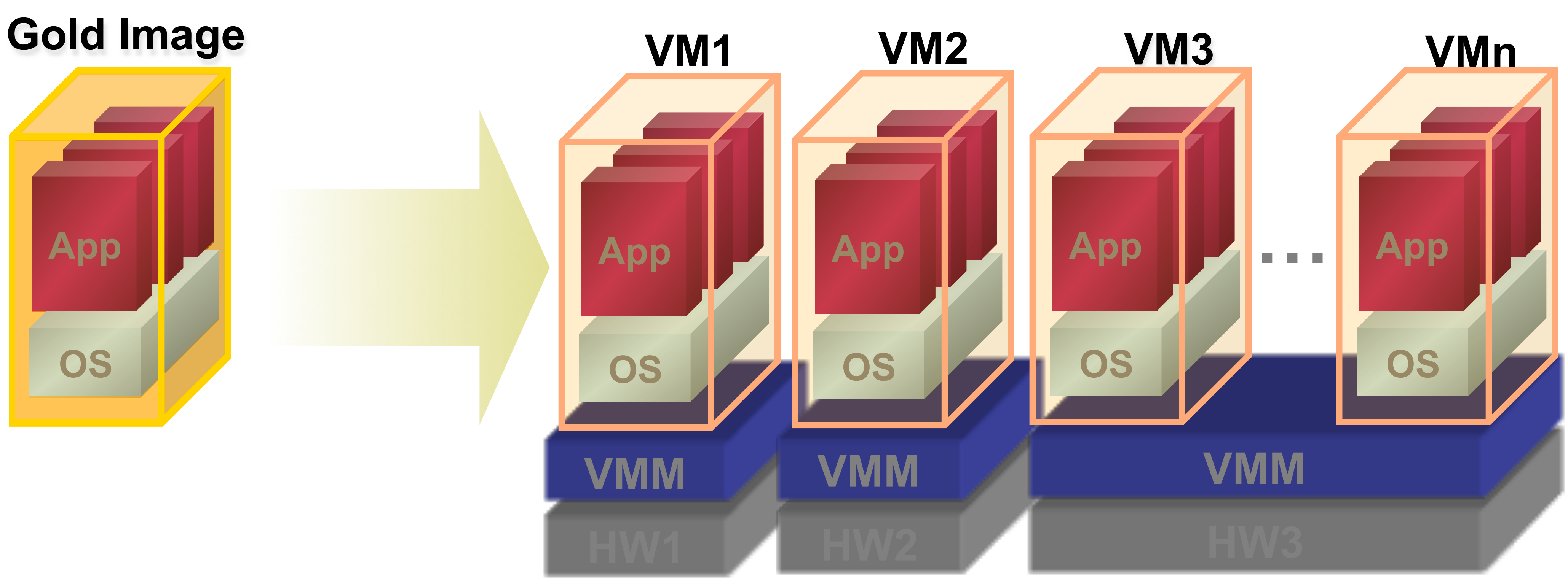
VM Usage: Workload Isolation (II)

- Migrate (move) running VMs to a different platform
  - Resources can be adjusted dynamically
  - VM migration can be triggered automatically by workload balancing or failure-prediction agents
  - If a given application needs more resources, it could be easily moved to other physical host with more power



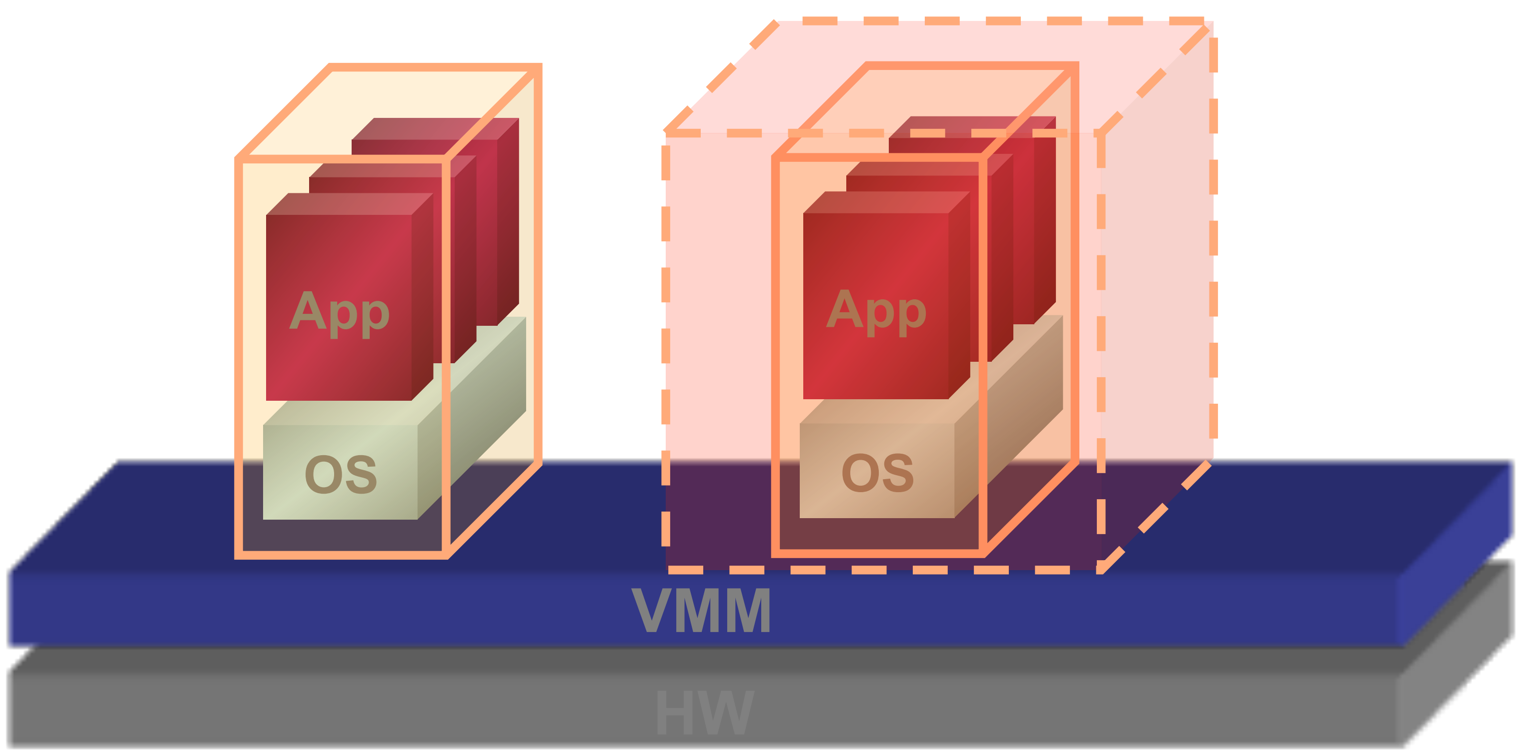
VM Usage: Fast Deployment

- Service providers usually offer some standard services
  - Standard images can be provided instantaneously
  - Simplifies deployment procedures: everything is stored in a file that represents the VM
  - Easier backward compatibility (Gold Image 1, 2, 3, etc)

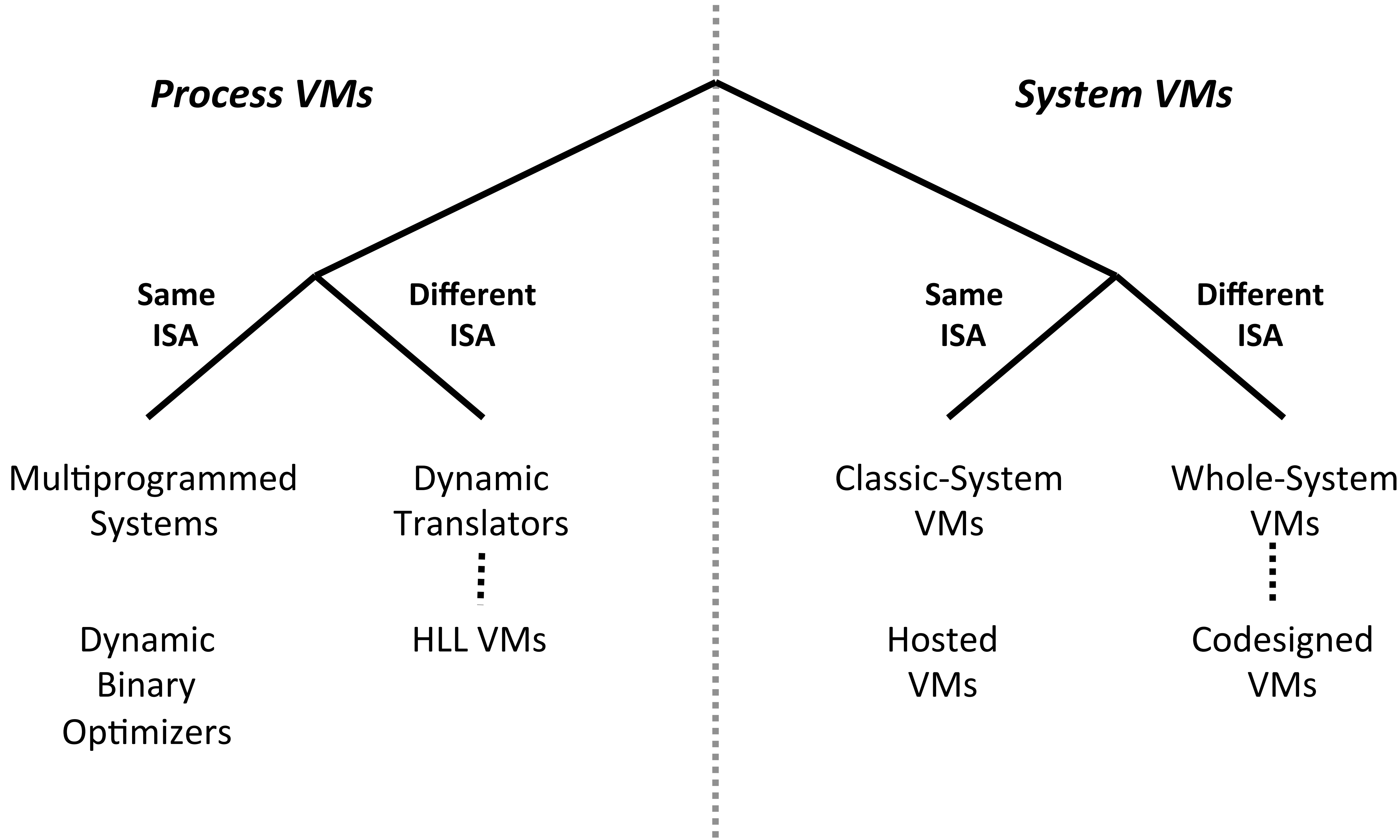


VM Usage: Testing and Develpment

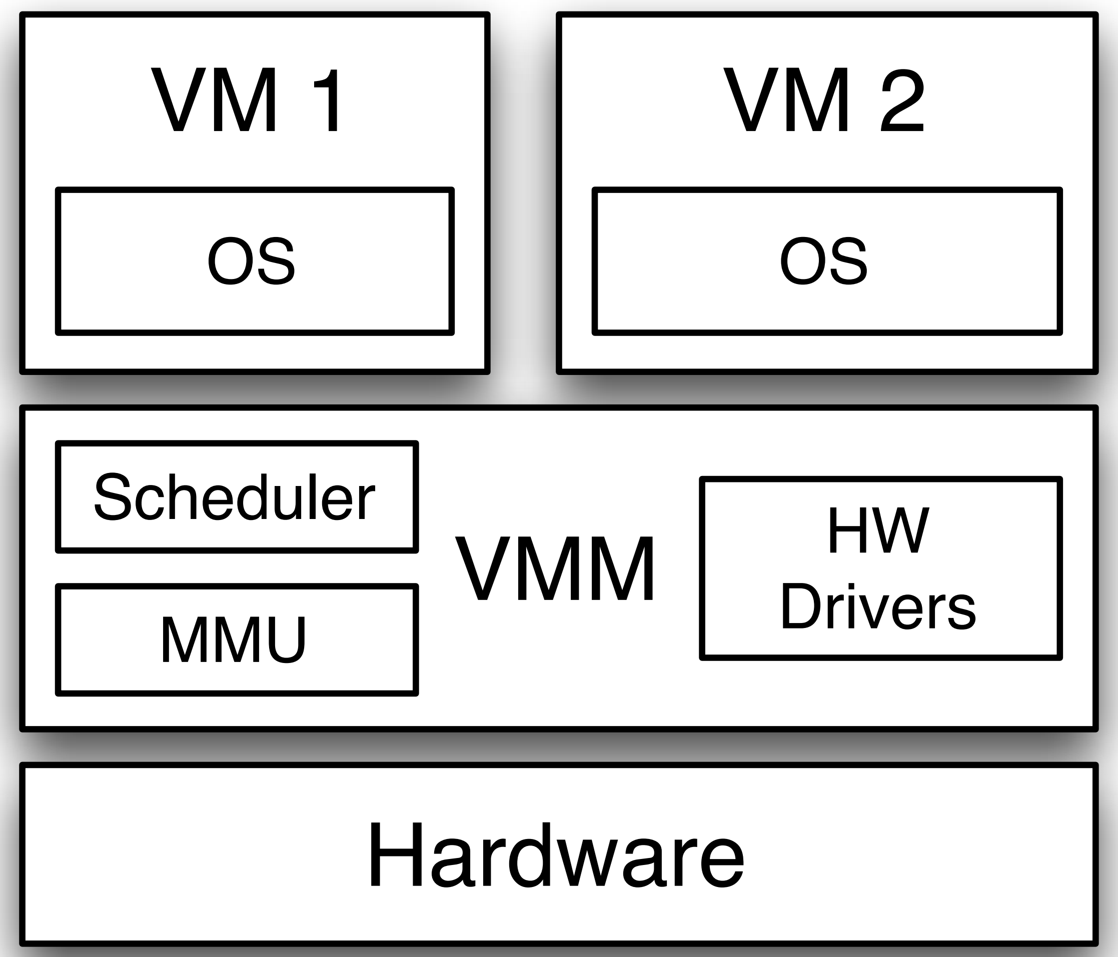
- Development and testing environments
  - A VM with standard tools is distributed amongst developers
  - Releasing new revisions of tools, patches, etc. is very simple
- Business Agility and Productivity
  - It allows to easily transform environments (Development to test, back to development, etc)
- Deployment of Patches in controlled environments
- Allows for testing in production hardware before official activation



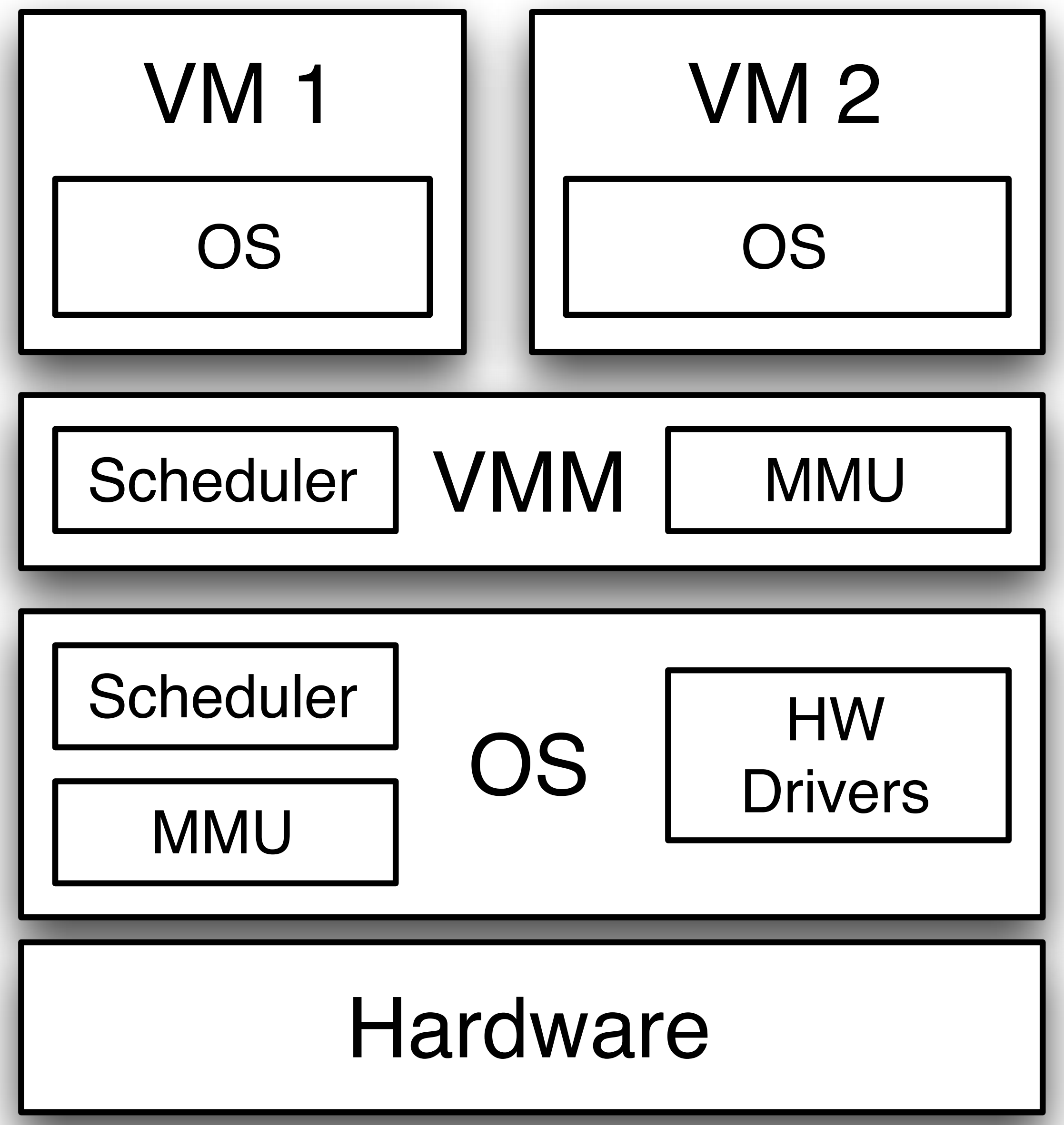
VM Taxonomy


Process VMs
System VMs
Same ISA
Different ISA
Same ISA
Different ISA
Multiprogrammed Systems
Dynamic Translators
Classic-System VMs
Whole-System VMs
Dynamic Binary Optimizers
HLL VMs
Hosted VMs
Codesigned VMs

System Virtualization



Native Virtualization



Hosted Virtualization

Virtualization Implementations

- ***Full Virtualization***
  - Software Based
  - VMware and Microsoft
- ***Para Virtualization***
  - Cooperative virtualization
  - Modified guest OS
  - VMware, Xen
- ***Hardware-assisted Virtualization***
  - Unmodified guest OS
  - VMware and Xen on virtualization-aware hardware platforms

Resource Control

- VMM must maintain *overall control* of the hardware resources
  - Hardware resources are assigned to VMs when they are created/ executed
  - Should have a way to get them back when they need to assigned to a different VM
  - Similar to multi-programming in OS
- Privileged Resources
  - Certain resources are accessible only to and managed by VMM
  - Interrupts relating to such resources must then be handled by VMM
  - Privileged resources are emulated by VMM for the VM
- All resource that could help maintain control are marked privileged
  - "Interval timer" is used to decide VM scheduling
  - "Page table base register" (CR3 on x86) is used to isolate VM memory

Classes of instructions

- PRIVILEGED instructions trap if executed in user mode and do not trap if executed in kernel mode
- SENSITIVE instructions interact with hardware
  - CONTROL-sensitive instructions attempt to change the configuration of resources in the system
  - BEHAVIOR-sensitive instructions have their result depending on the configuration of resources (e.g. mode of operation)
- INNOCUOUS instructions are not sensitive

Popek & Goldberg Theorem (1974)

For any conventional third-generation computer a virtual machine monitor with the following properties:

1. Efficiency: innocuous instruction must be executed natively

2. Resource Control: guest can not directly change host resources

3. Equivalence: app behavior in guest must be identical to app behavior in host

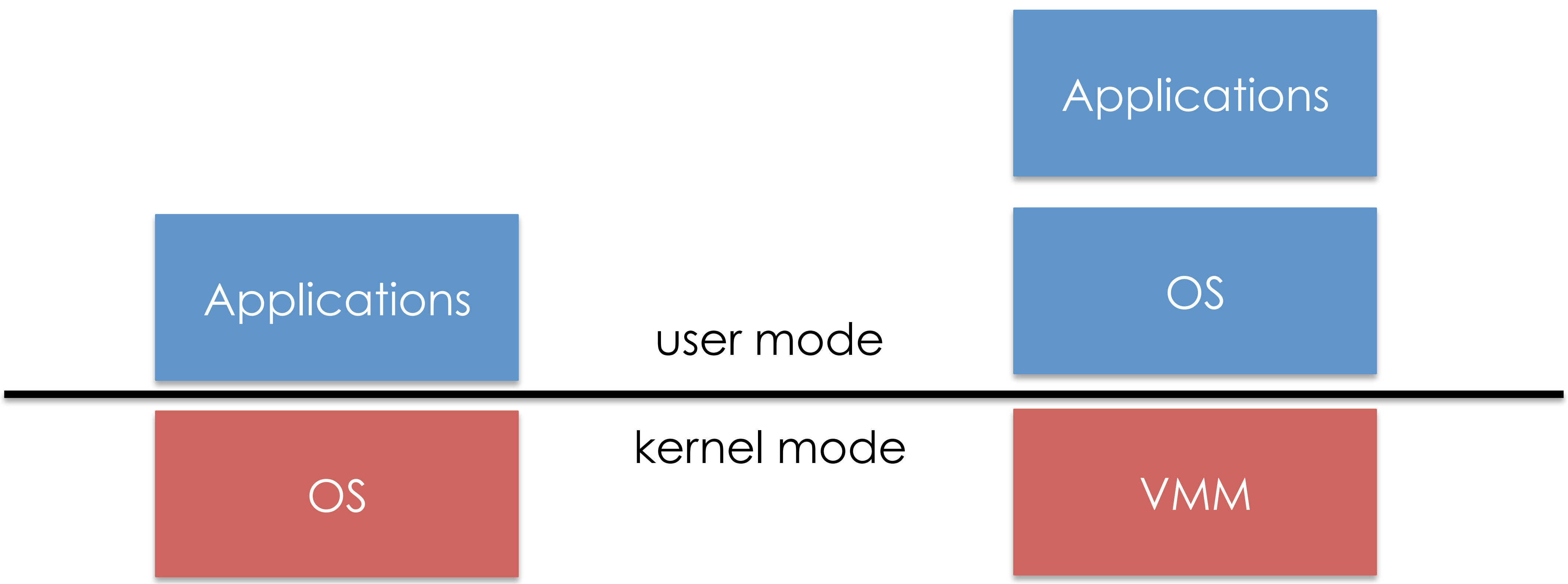
may be constructed if the set of sensitive instructions for that computer is a subset of the set of privileged instructions

Full Virtualization

ISTITUTO DI SCIENZA E TECNOLOGIE DELL'INFORMAZIONE "A. FAEDO"

Trap & Emulate

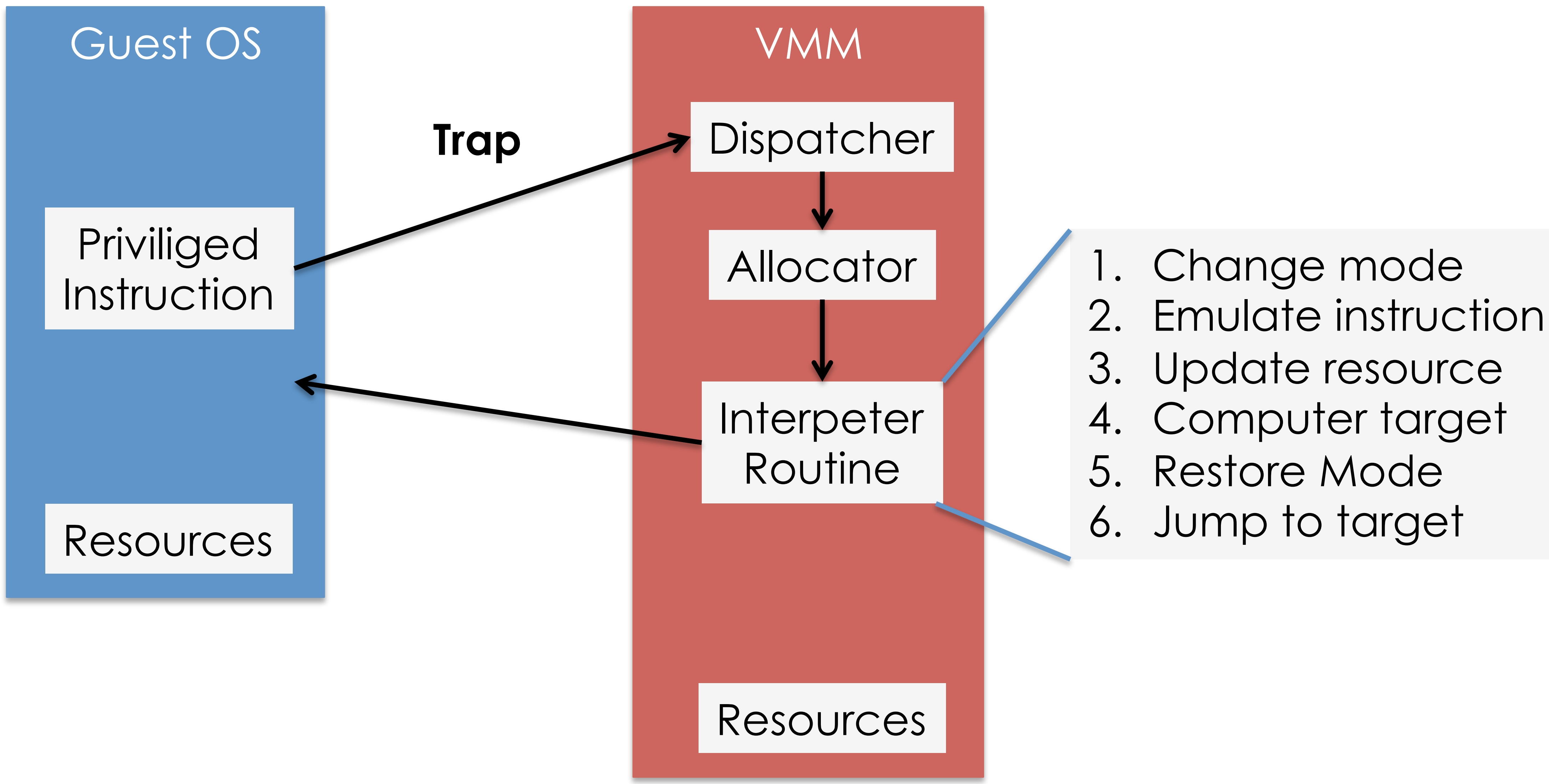
- Must be able to "detect" when VMM must intervene
- Some ISA instructions must be "trapped" and "emulated"
- Must De-Privilege OS
- Very similar to the way programs transfer control to the OS kernel during a system call



Privileged Resources

- Each VM's privileged state differs from that of the underlying HW.
- Guest-level **primary structures** reflect the state that a guest sees.
- VMM-level **shadow structures** are copies of primary structures.
- Traps occur when **on-chip privileged state** is accessed/modified.
- HW page protection schemes are employed to "detect" when **off-chip privileged** state is accessed/modified

Handling of Privileged Instructions

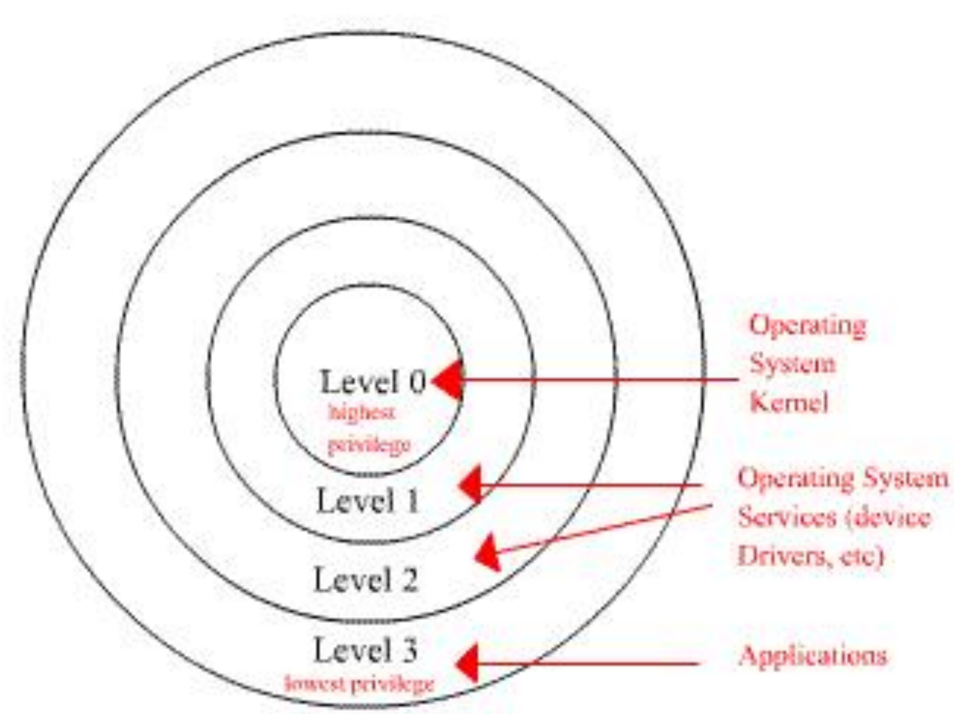


Traps are expensive!

Is X86 (fully) virtualizable?

- Lack of trap when priviliged instructions run at user level
- Some privileged instructions execute only in ring 0 but do not fault when executed outside ring 0
- Masking interrupts can only be done in ring 0

Intel IA32 Protection Rings



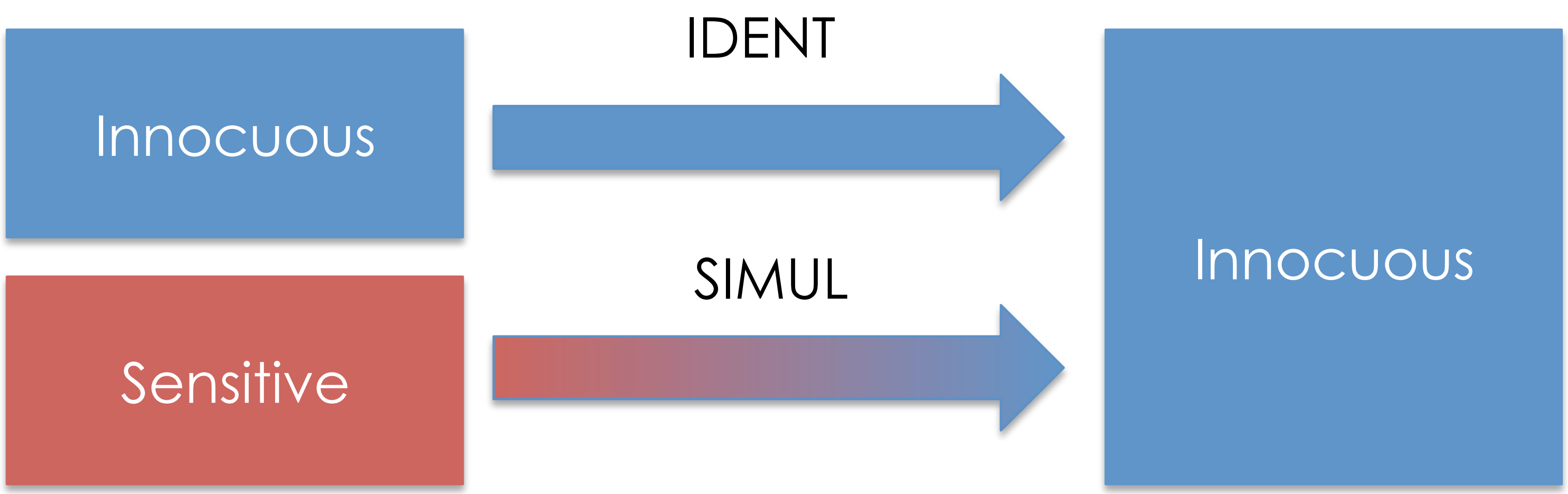
Example: POPF

- Same instruction **behaves differently** depending on execution mode
- **User Mode**: changes ALU flags
- **Kernel Mode**: changes ALU and system flags
- Does not generate a trap in user mode

The IA-32 instruction set contains 17 sensitive, unprivileged instructions

ISTITUTO DI SCIENZA E TECNOLOGIE
DELL'INFORMAZIONE "A. FAEDO"

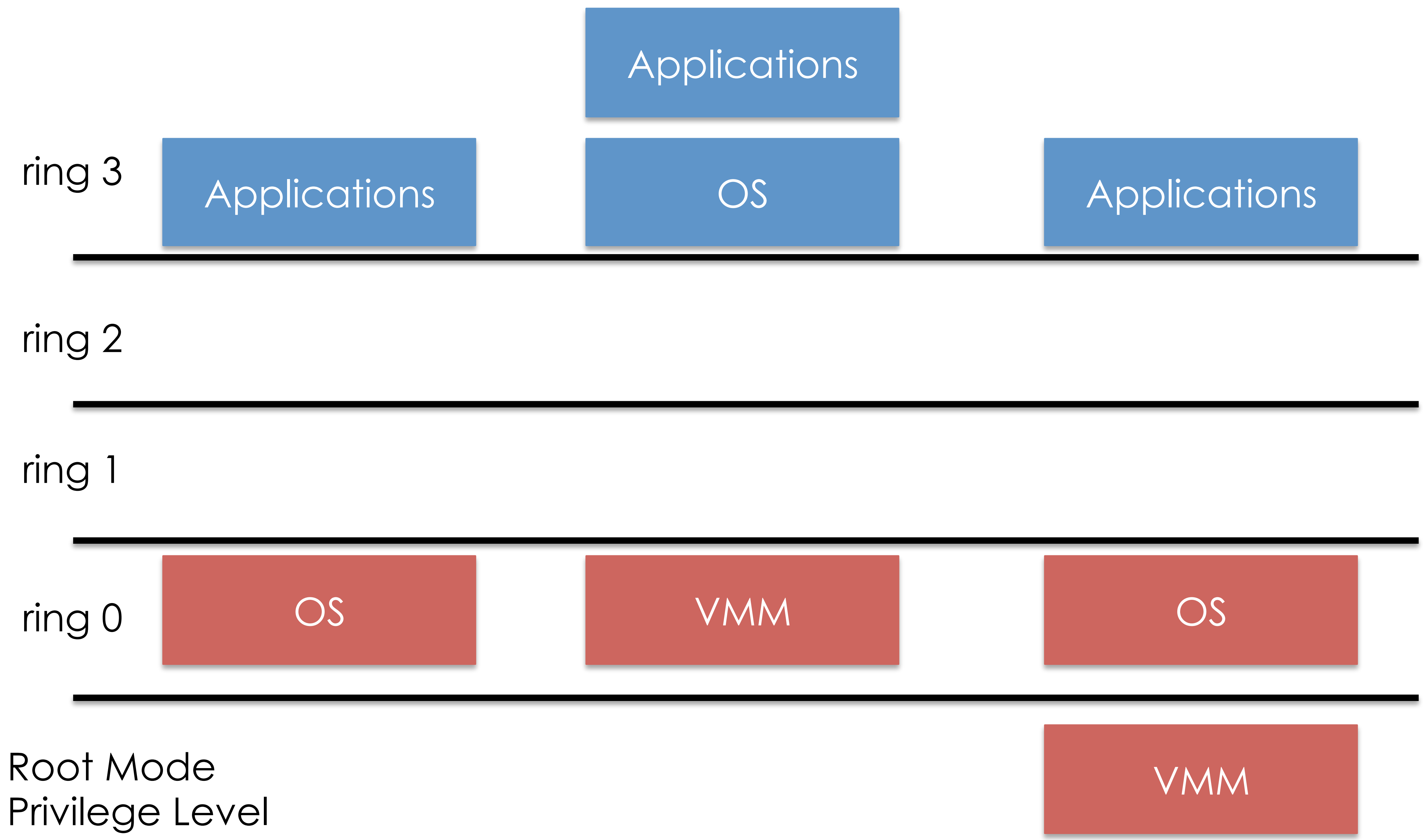
Binay Translation



- **Binary** – input is machine-level code
- **Dynamic** – occurs at runtime
- **On demand** – code translated when needed for execution
- **System level** – makes no assumption about guest code
- **Subsetting** – translates from full instruction set to safe subset
- **Adaptive** – adjust code based on guest behavior to achieve efficiency

ISTITUTO DI SCIENZA E TECNOLOGIE DELL'INFORMAZIONE "A. FAEDO"

Hardware-assisted Virtualization

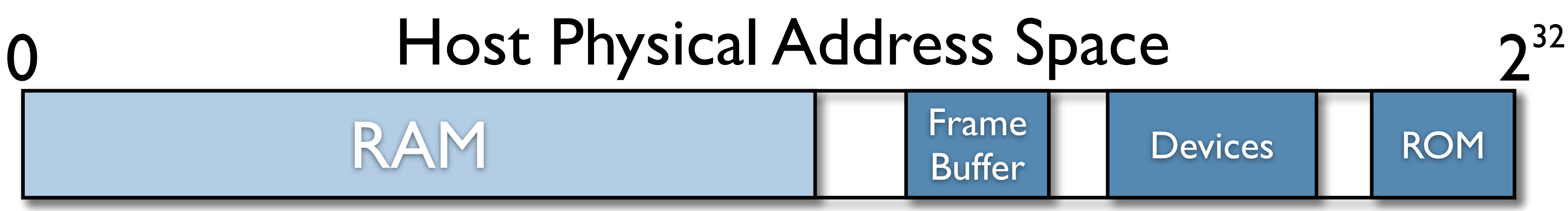


ISTITUTO DI SCIENZA E TECNOLOGIE
DELL'INFORMAZIONE "A. FAEDO"

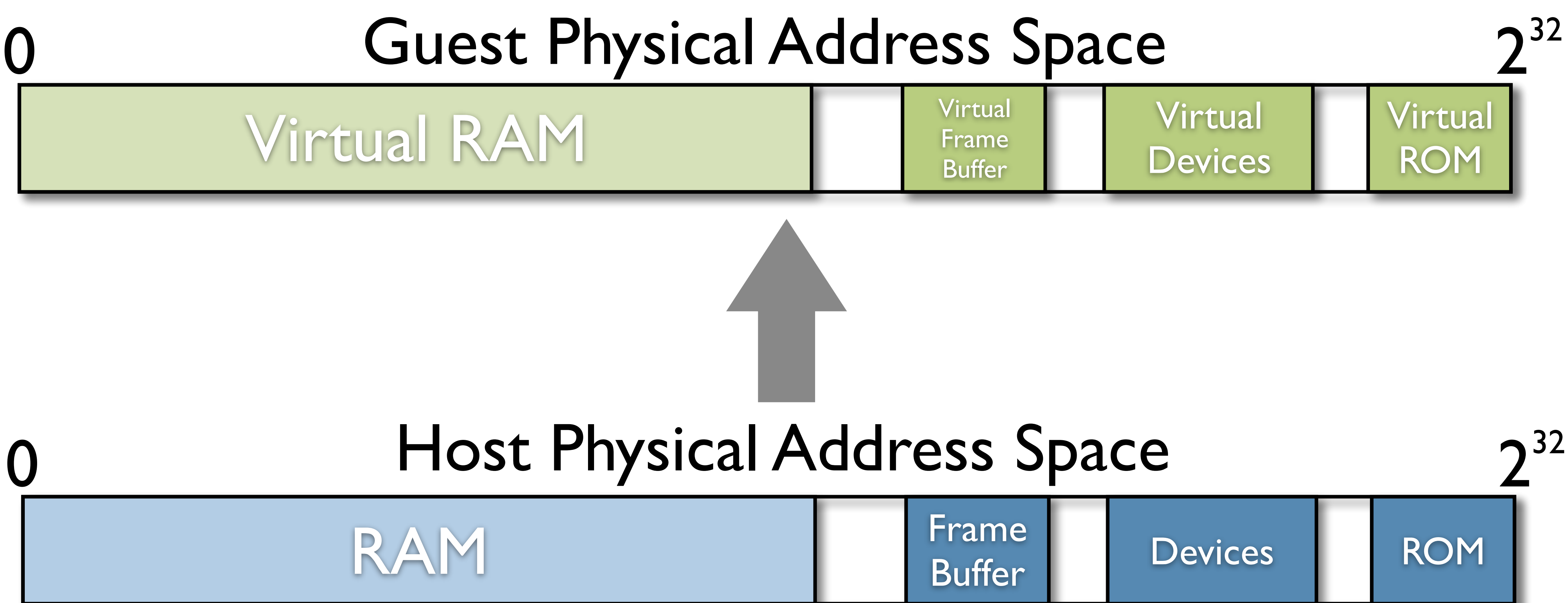
New Hardware Features

- Virtual Machine Control Blocks (VMCBs)
- Root mode privilege level
- Ability to transfer control to/from guest mode.
  - *vmrun* - host to guest.
  - *exit* - guest to host.
- VMM executes *vmrun* to start a guest.
  - Guest state is loaded into HW from in-memory VMCB.
  - Guest mode is resumed and guest continues execution.
- Guests execute until they “toy” with control bits of the VMCB.
  - An *exit* operation occurs.
  - Guest saves data to VMCB.
  - VMM state is loaded into HW - switches to host mode.
  - VMM begins executing.

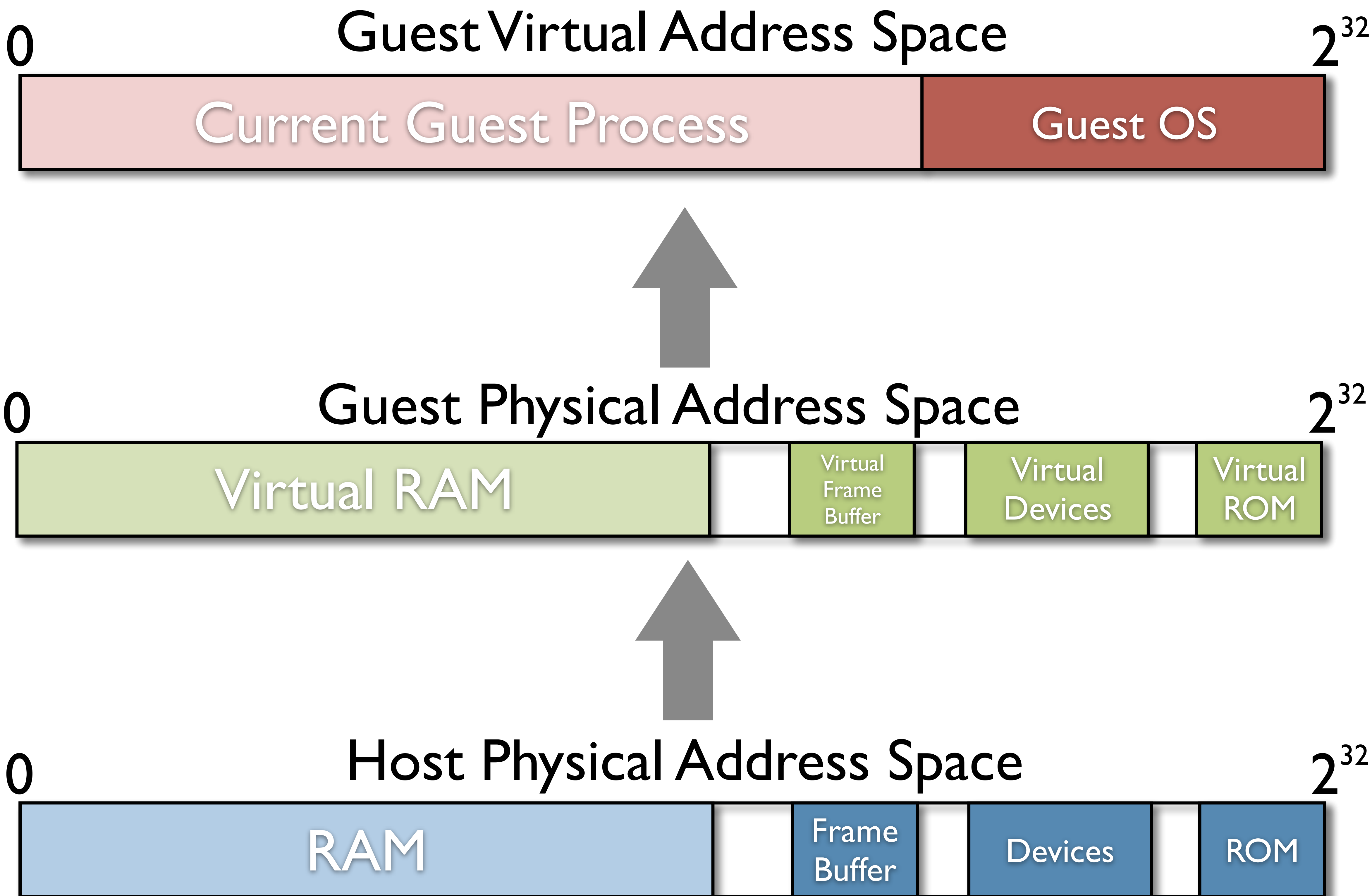
Virtualizing Virtual Memory



Virtualizing Virtual Memory



Virtualizing Virtual Memory



Shadow Page Table

Shadow Page Table

- The VMM must map guest virtual address to host physical address

Shadow Page Table

- The VMM must map guest virtual address to host physical address
- Guest OS maintains its own virtual memory page table in the guest physical memory

Shadow Page Table

- The VMM must map guest virtual address to host physical address
- Guest OS maintains its own virtual memory page table in the guest physical memory
- The VMM maintains a mapping from each guest physical memory page to the host physical memory page
  - PMAP data structure

Shadow Page Table

- The VMM must map guest virtual address to host physical address
- Guest OS maintains its own virtual memory page table in the guest physical memory
- The VMM maintains a mapping from each guest physical memory page to the host physical memory page
  - PMAP data structure
- The VMM seems able to intercept MMU hardware requests to translate GVAs
  - Monitoring PTBR
  - Two memory accesses, to guest virtual memory page table and PMAP

Shadow Page Table

- The VMM must map guest virtual address to host physical address
- Guest OS maintains its own virtual memory page table in the guest physical memory
- The VMM maintains a mapping from each guest physical memory page to the host physical memory page
  - PMAP data structure
- The VMM seems able to intercept MMU hardware requests to translate GVAs
  - Monitoring PTBR
  - Two memory accesses, to guest virtual memory page table and PMAP
- So what the hell is a shadow page table?

Shadow Page Table

- The VMM must map guest virtual address to host physical address
- Guest OS maintains its own virtual memory page table in the guest physical memory
- The VMM m… emory page to the host
  - PMAP data
- The VMM s… s to translate G
  - Monitoring
  - Two memory accesses, to guest virtual memory page table and PMAP
- So what the hell is a shadow page table?

What about the TLB?

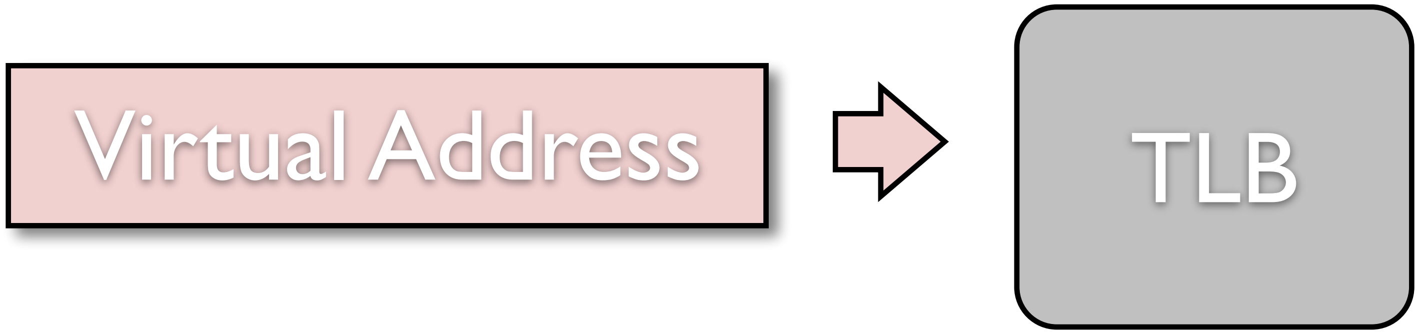
Shadow Page Table

- The VMM must intercept all VM instructions that manipulate:
  - The hardware TLB contents
  - Guest OS page table
- The actual hardware TLB is updated based on the separate shadow page tables
  - They contain the guest virtual to host physical address mapping
- The VMM must protect the host frames containing the guest page tables!

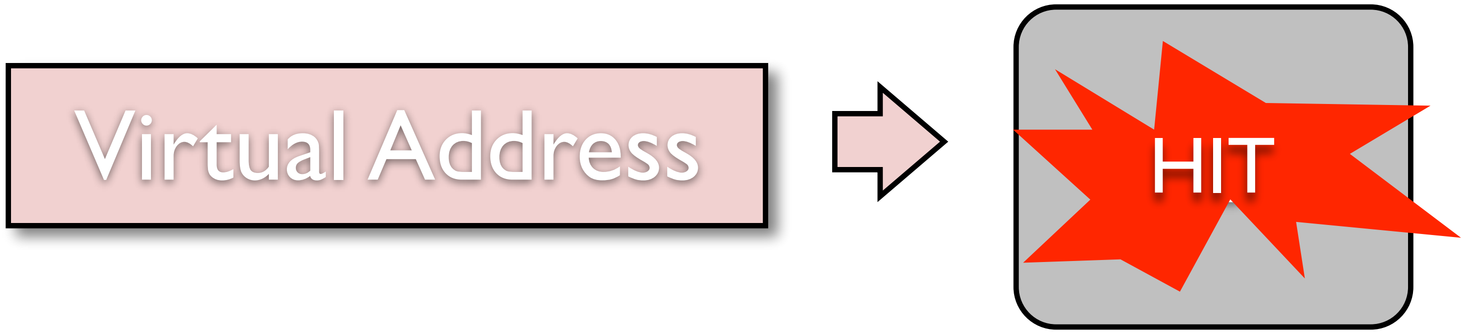
Shadow Page Tables with TLB

Virtual Address

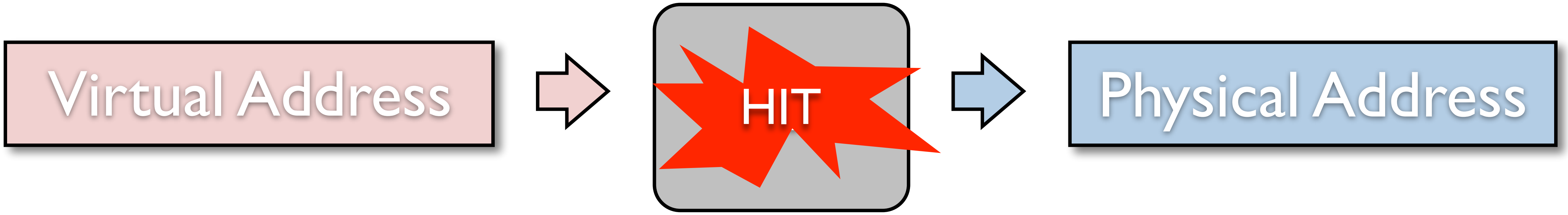
Shadow Page Tables with TLB



Shadow Page Tables with TLB



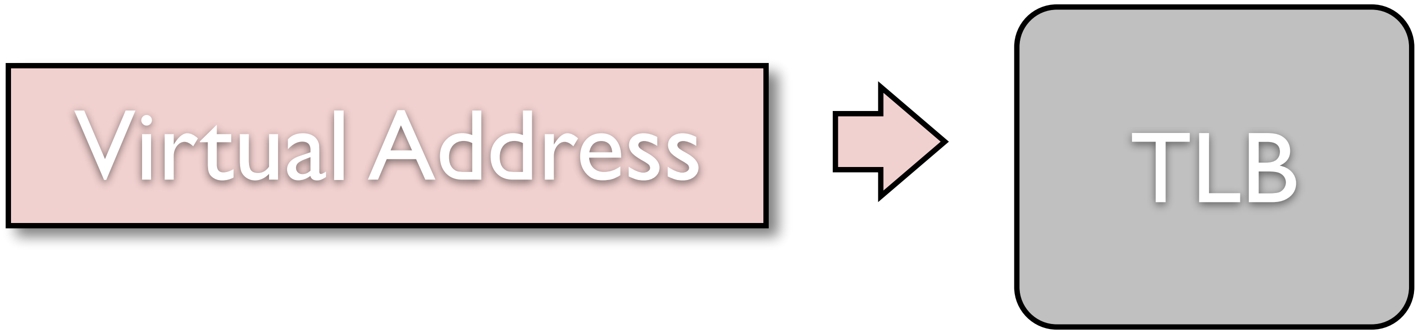
Shadow Page Tables with TLB



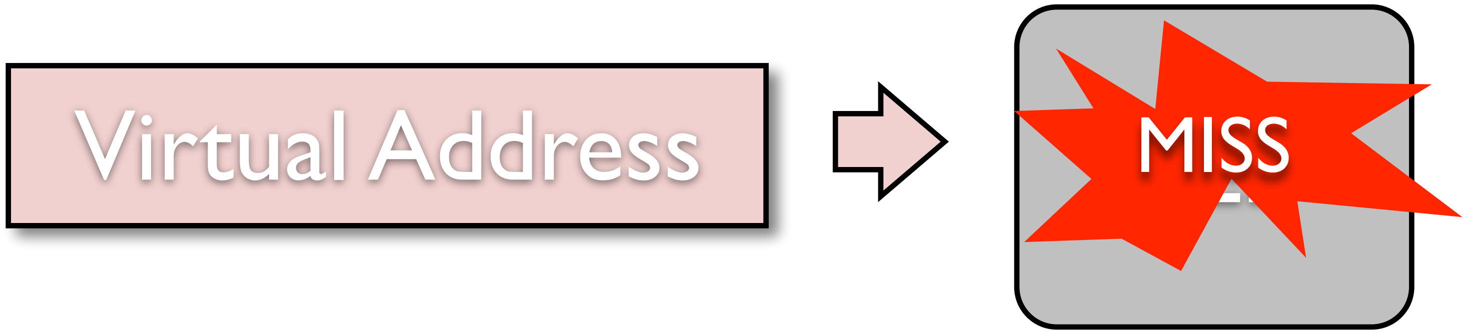
Shadow Page Tables with TLB

Virtual Address

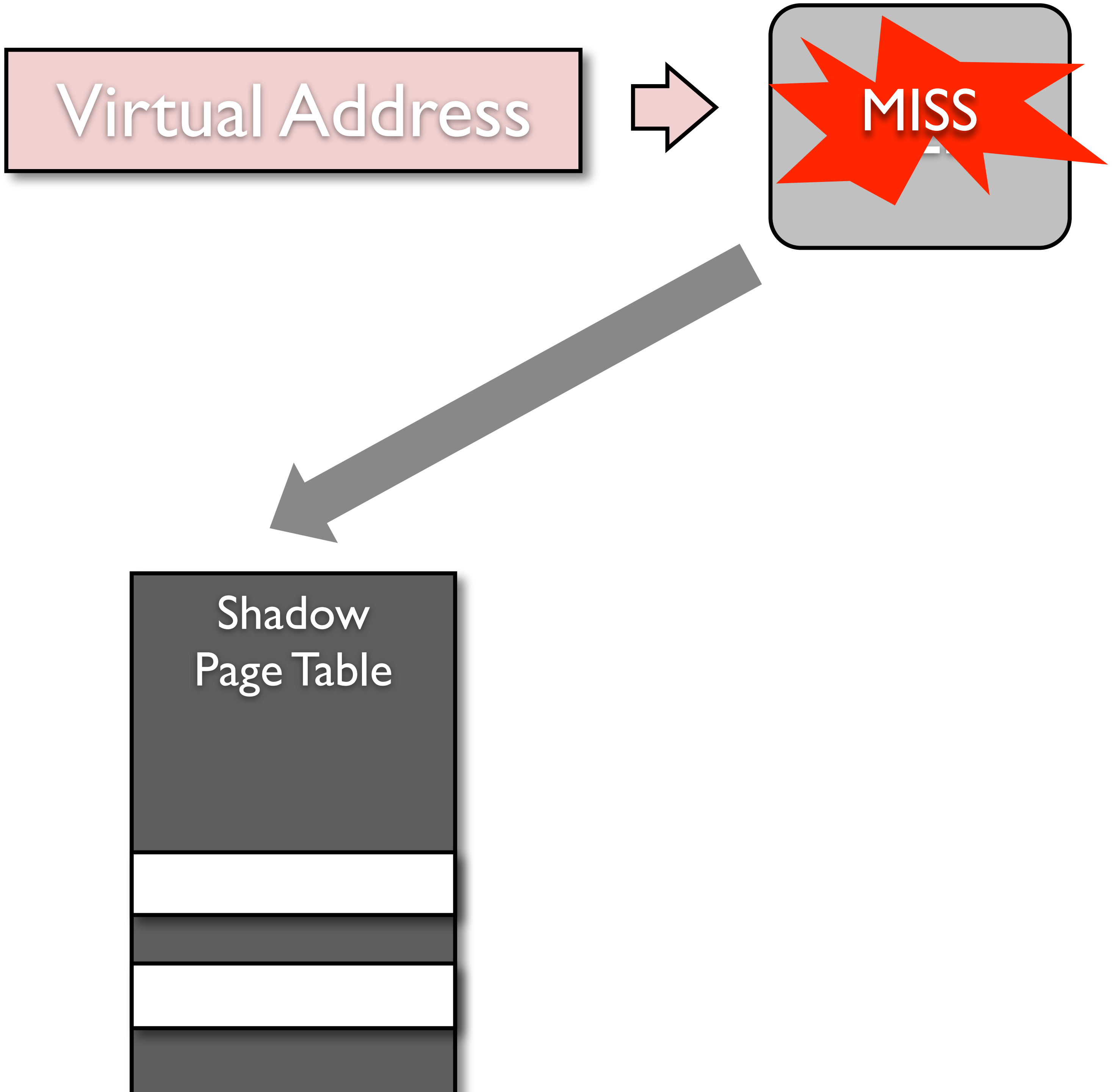
Shadow Page Tables with TLB



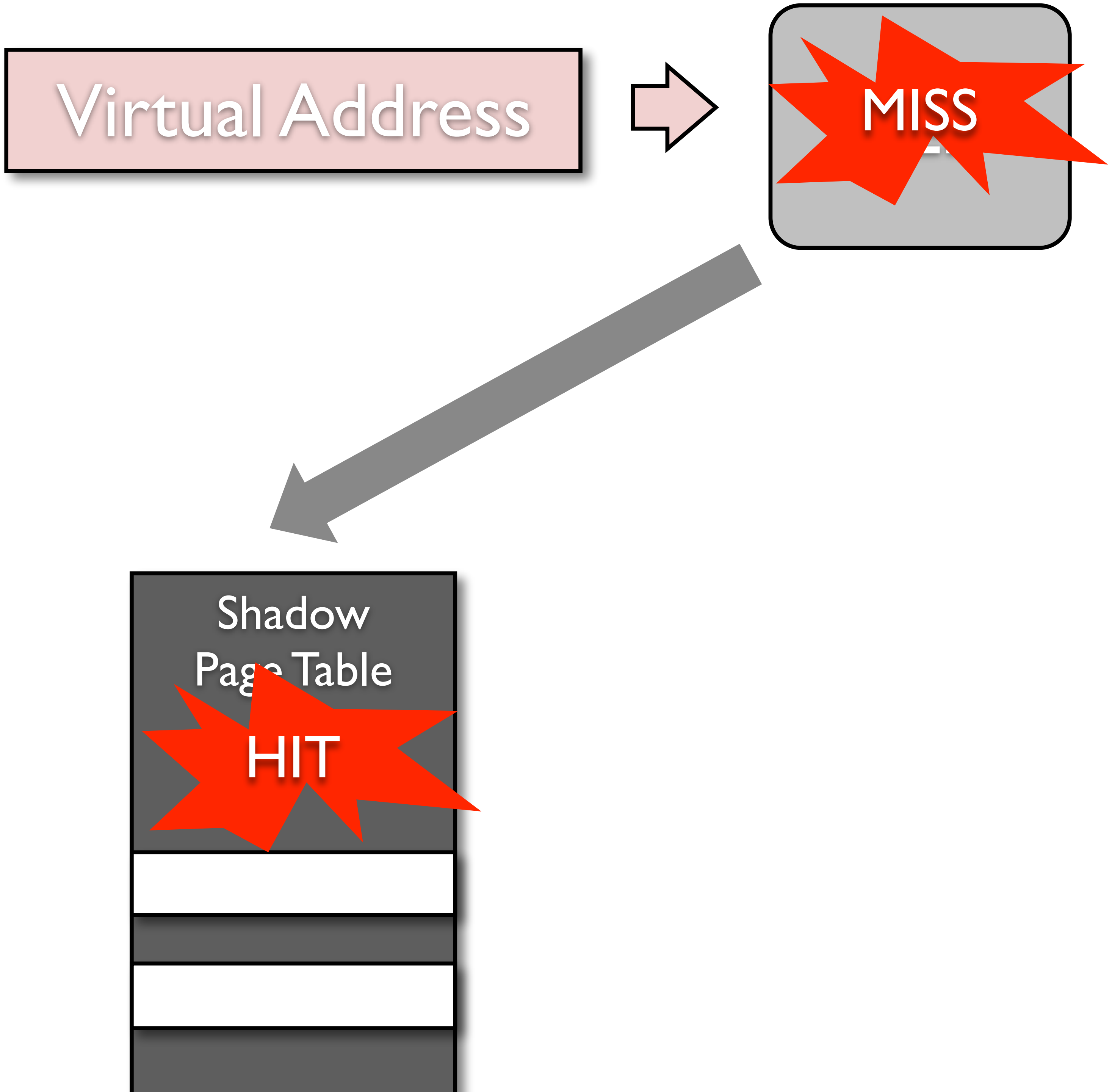
Shadow Page Tables with TLB



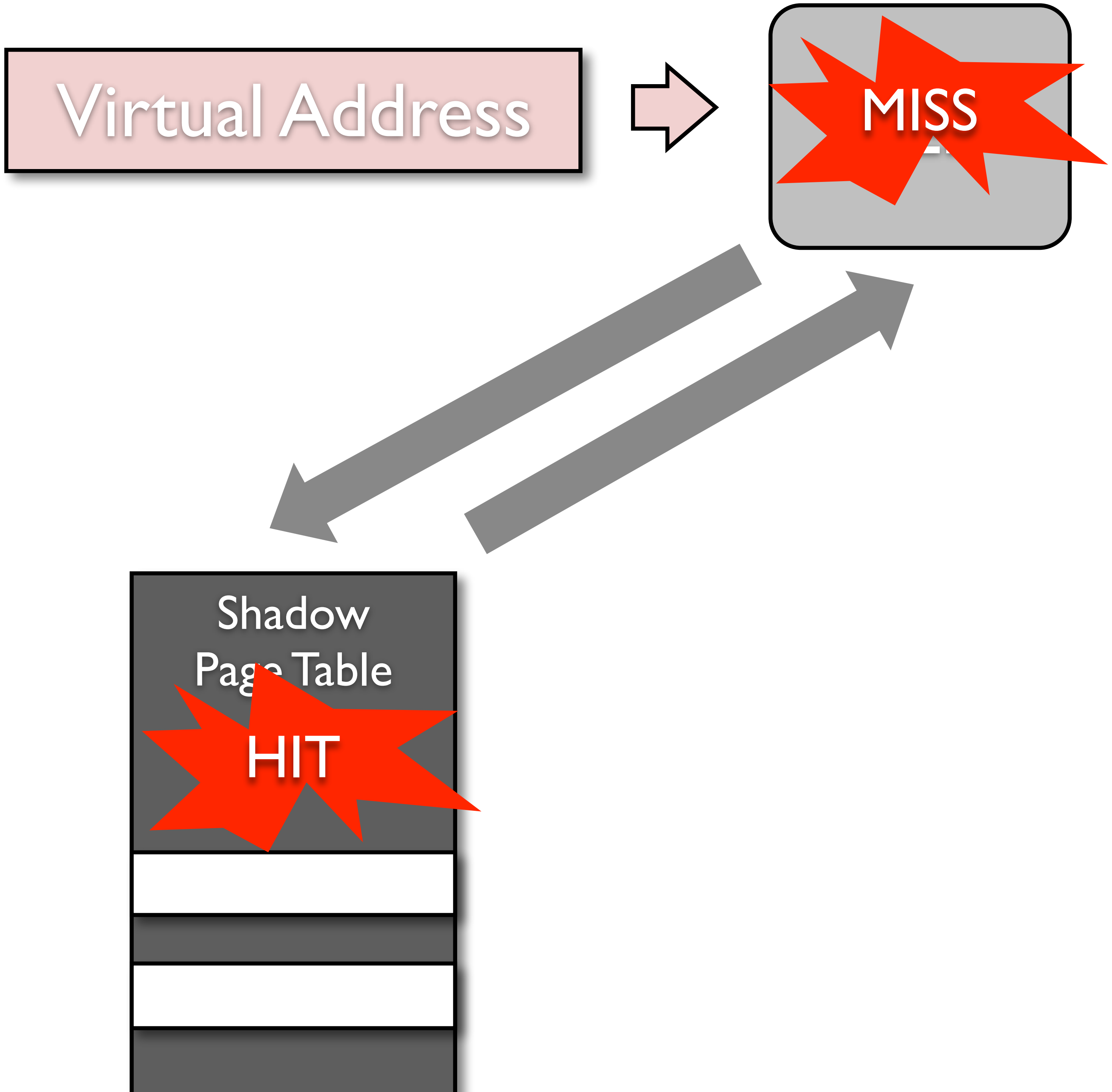
Shadow Page Tables with TLB



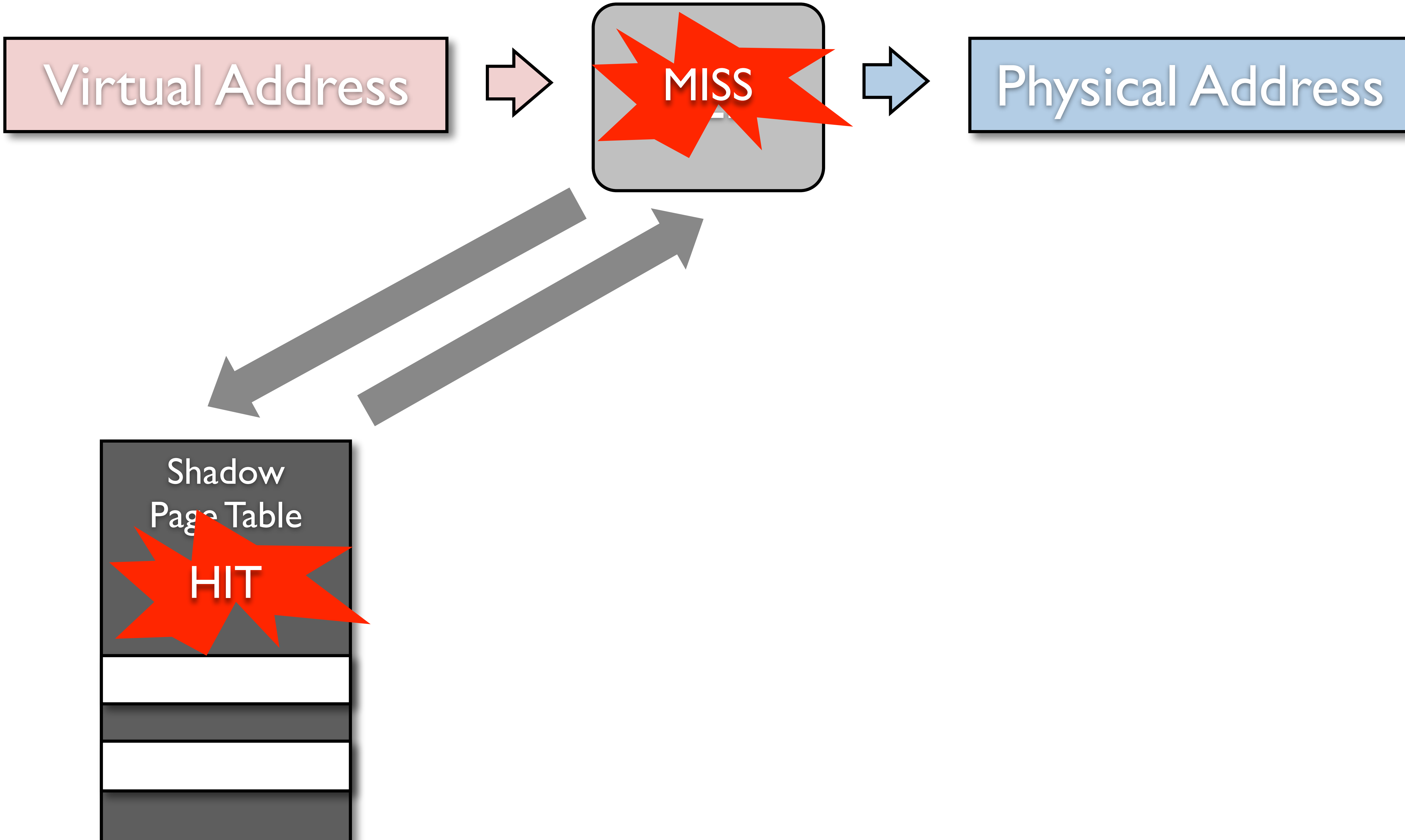
Shadow Page Tables with TLB



Shadow Page Tables with TLB



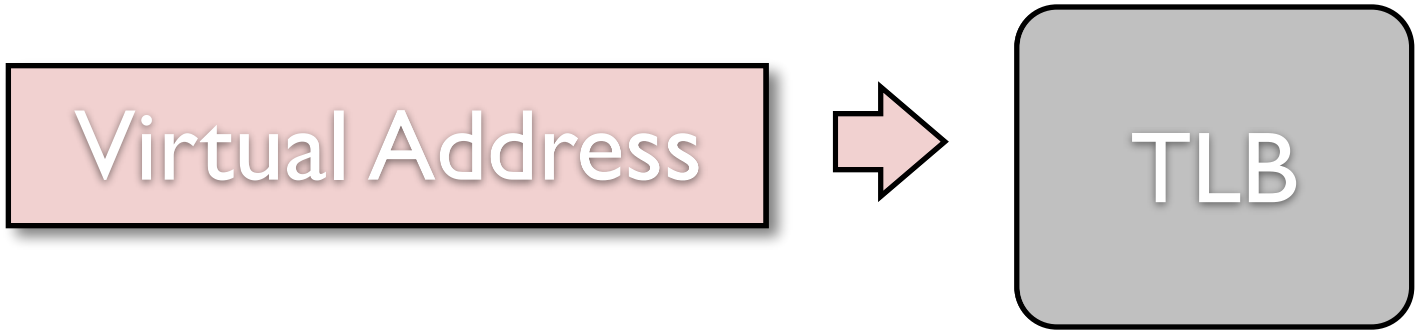
Shadow Page Tables with TLB



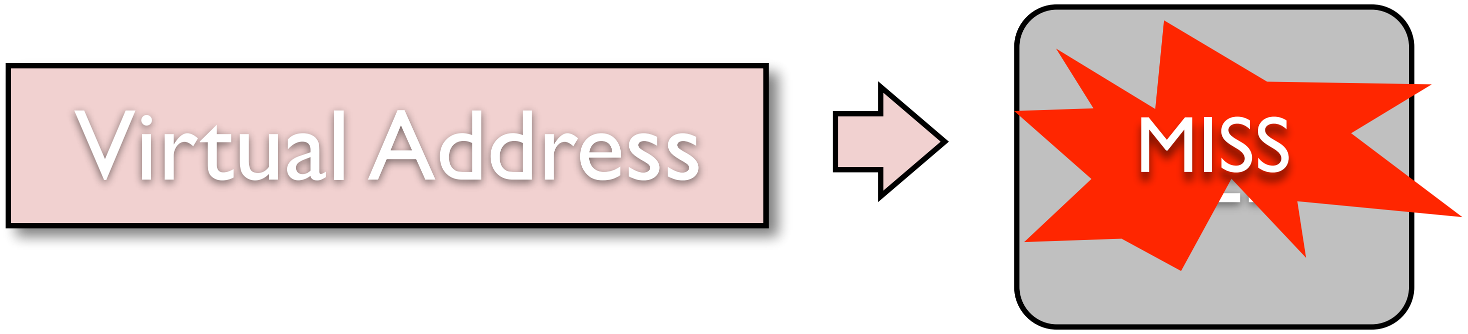
Shadow Page Tables with TLB

Virtual Address

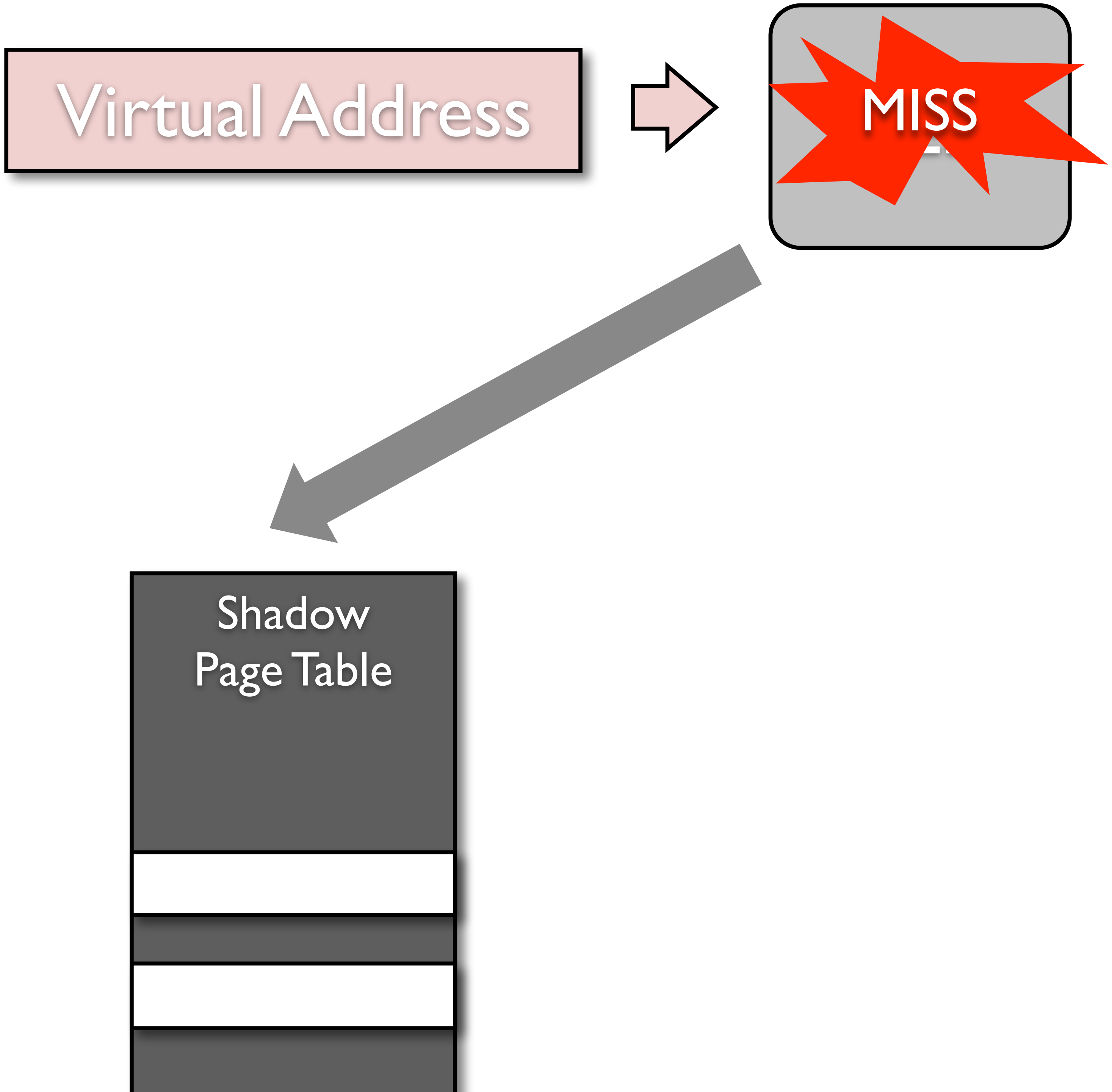
Shadow Page Tables with TLB



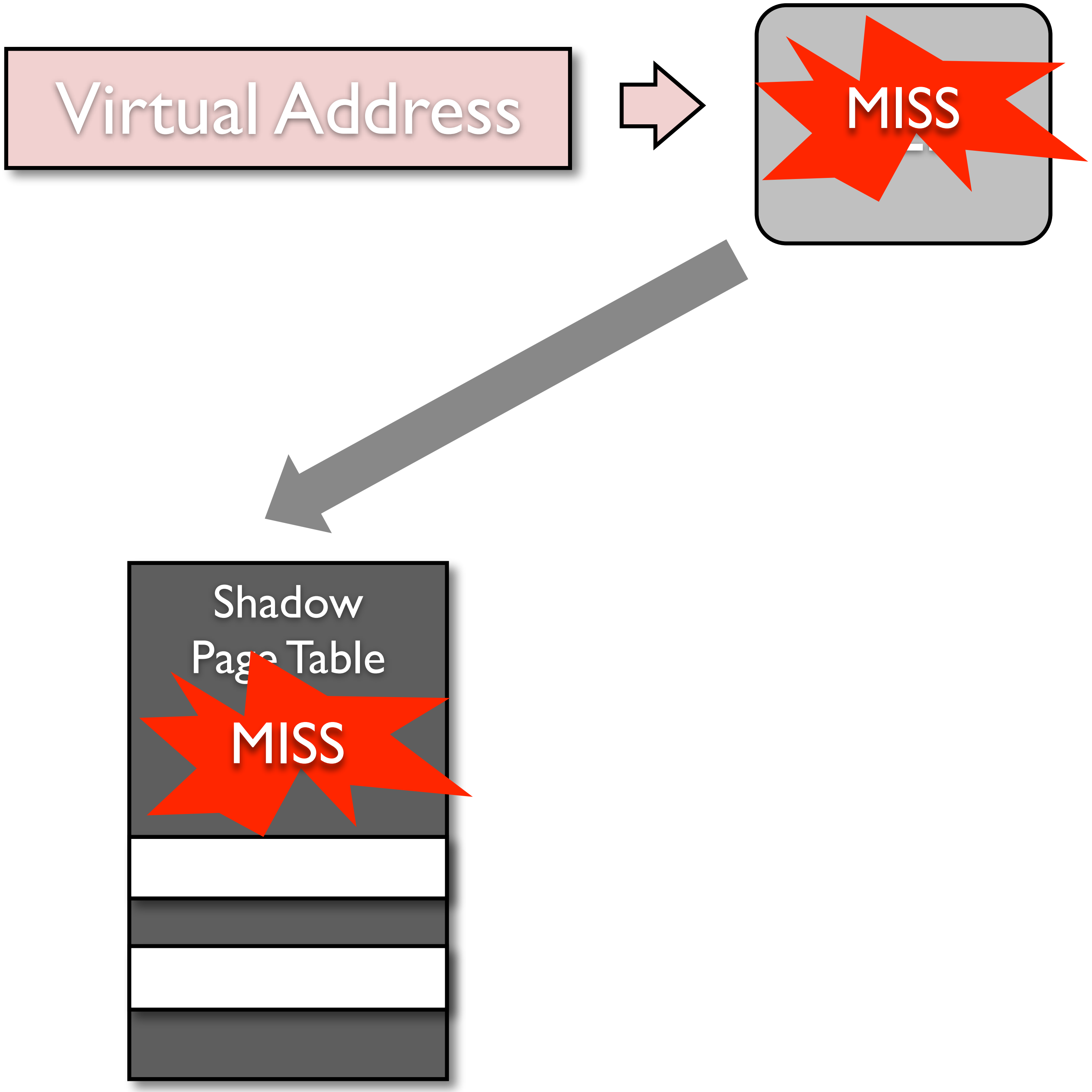
Shadow Page Tables with TLB



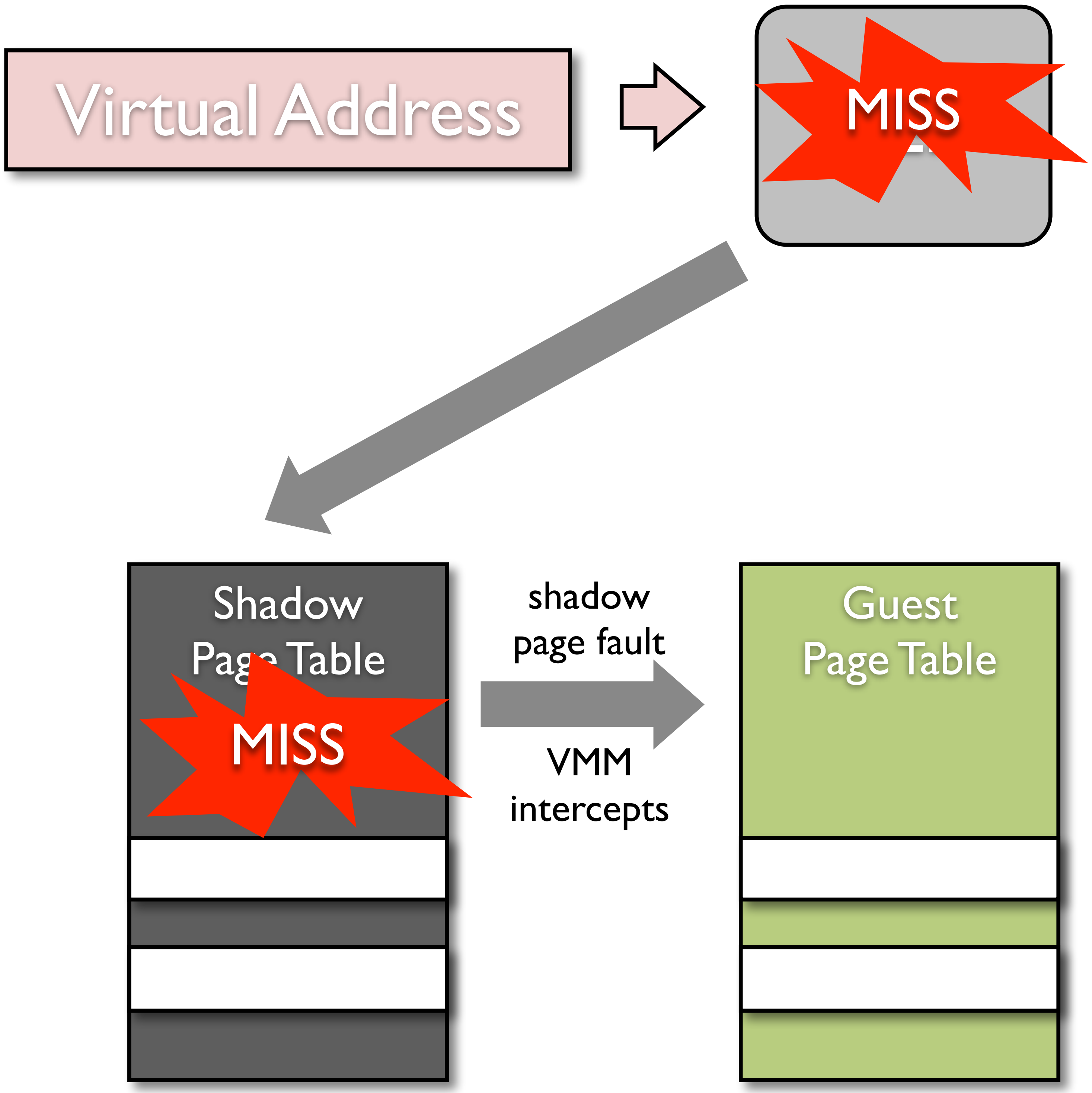
Shadow Page Tables with TLB



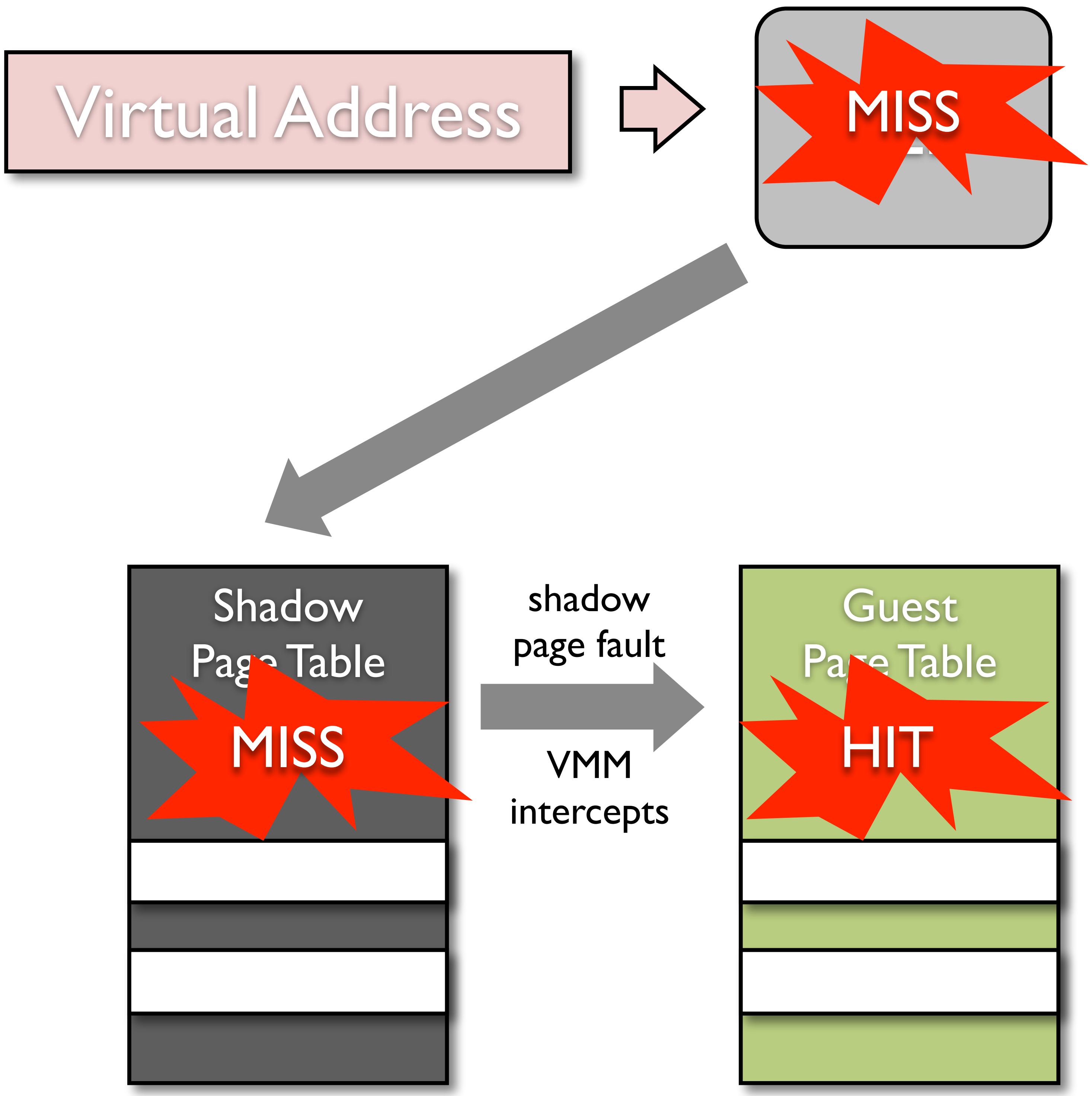
Shadow Page Tables with TLB



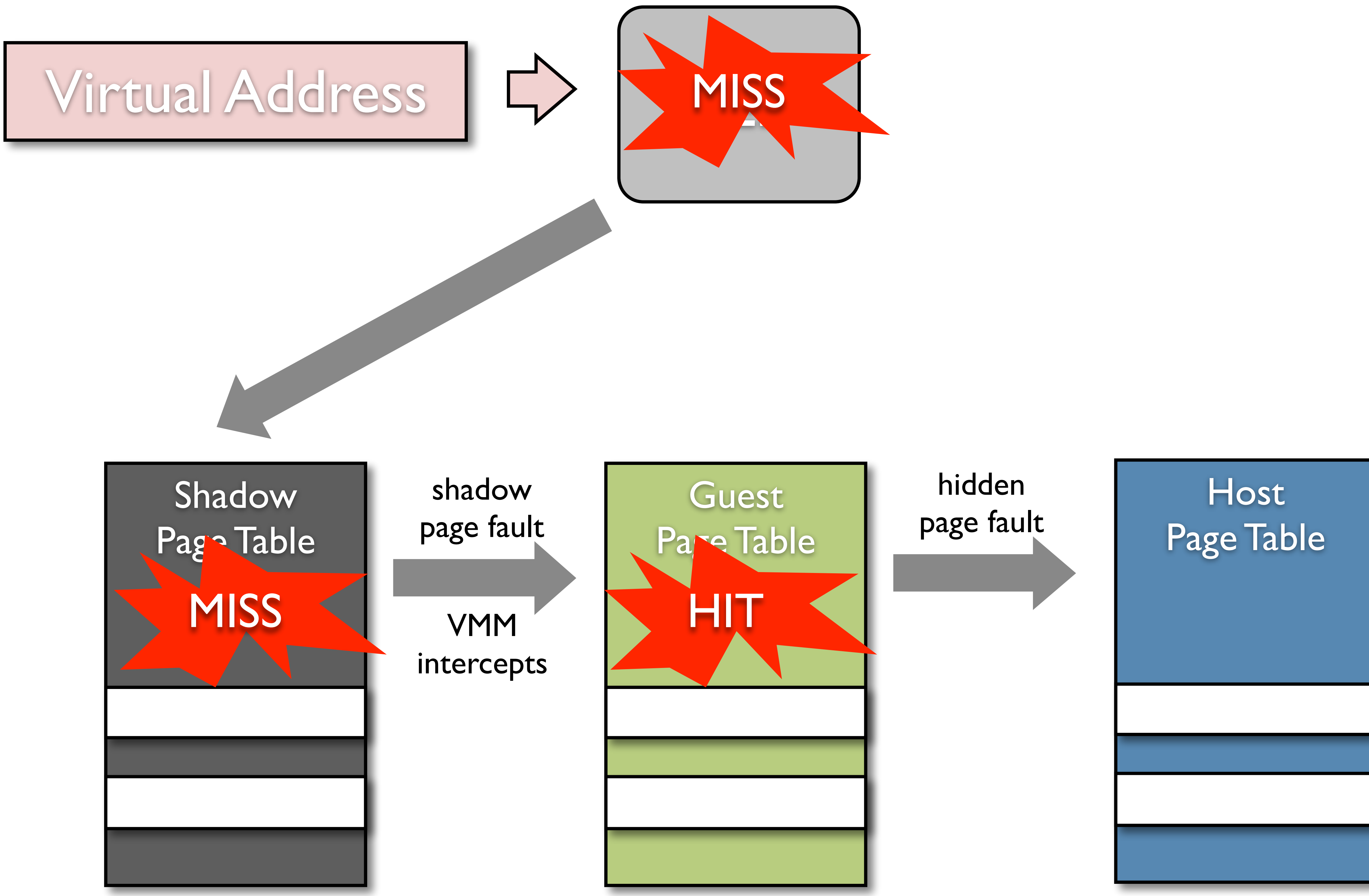
Shadow Page Tables with TLB



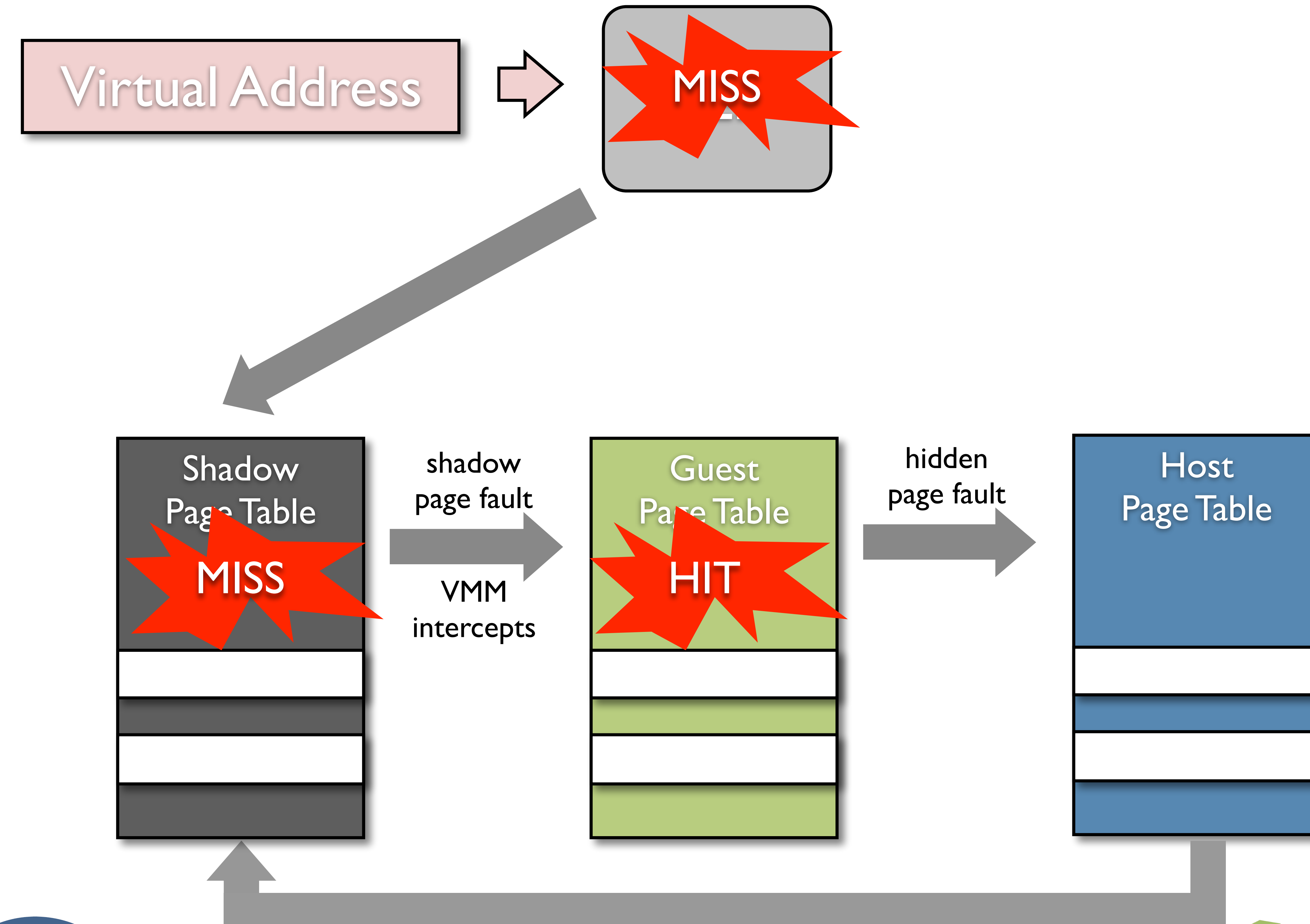
Shadow Page Tables with TLB



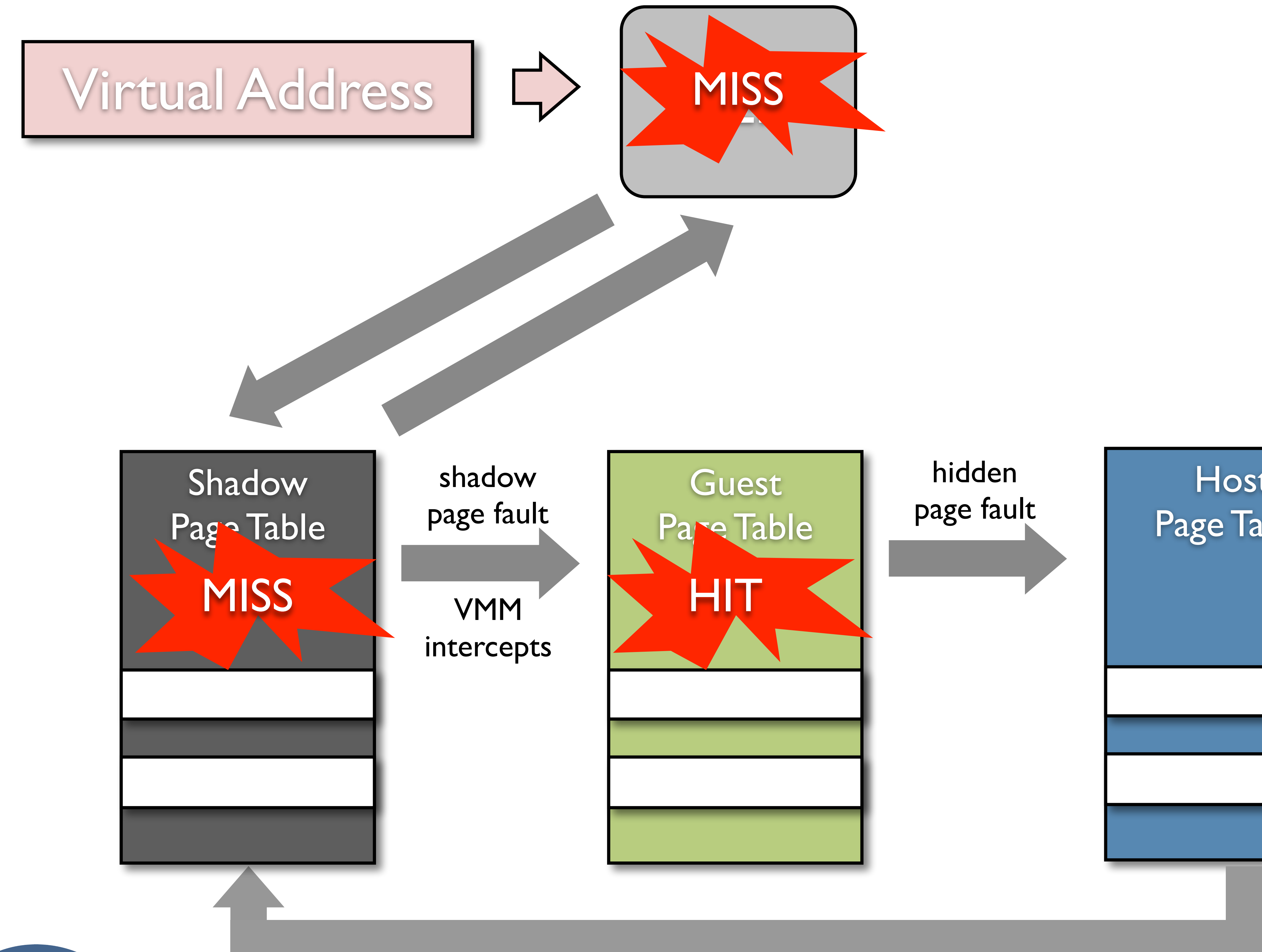
Shadow Page Tables with TLB



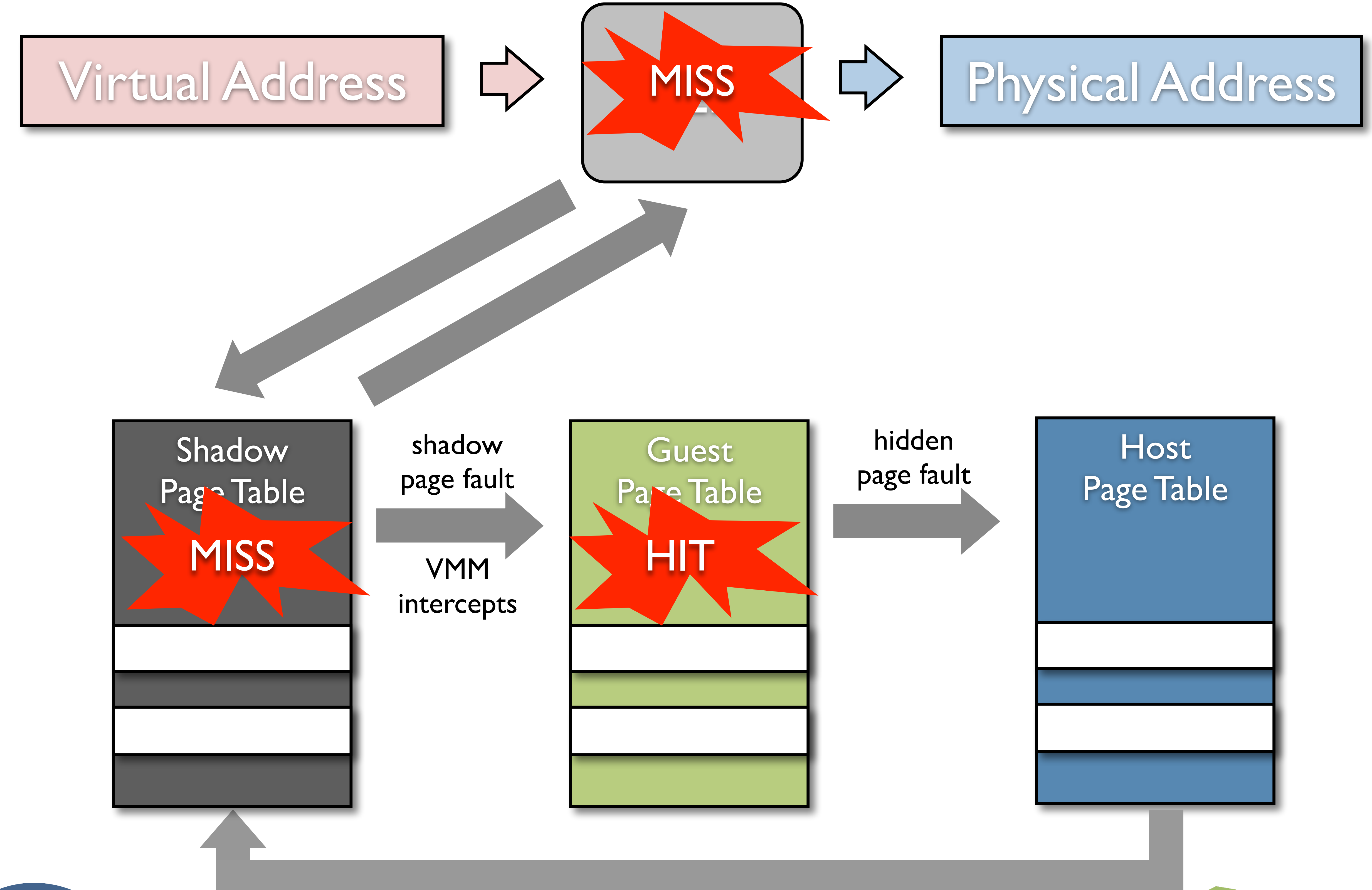
Shadow Page Tables with TLB



Shadow Page Tables with TLB



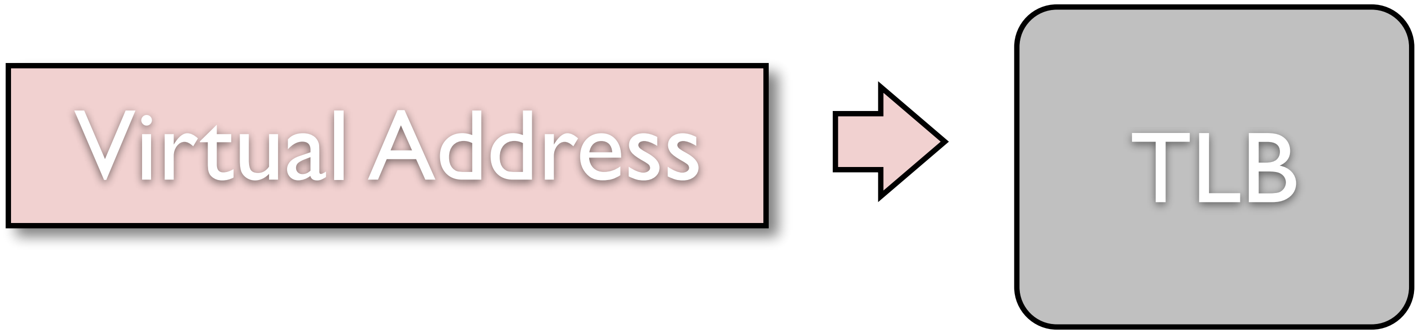
Shadow Page Tables with TLB



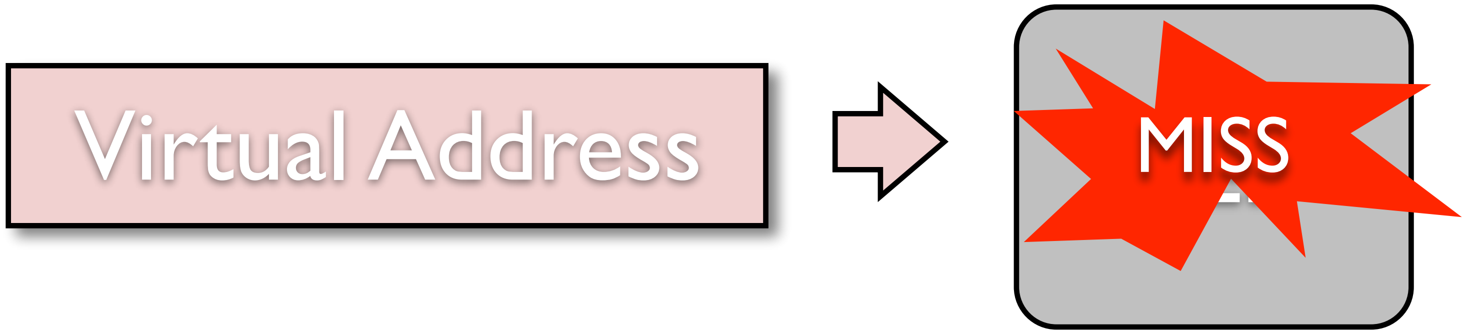
Shadow Page Tables with TLB

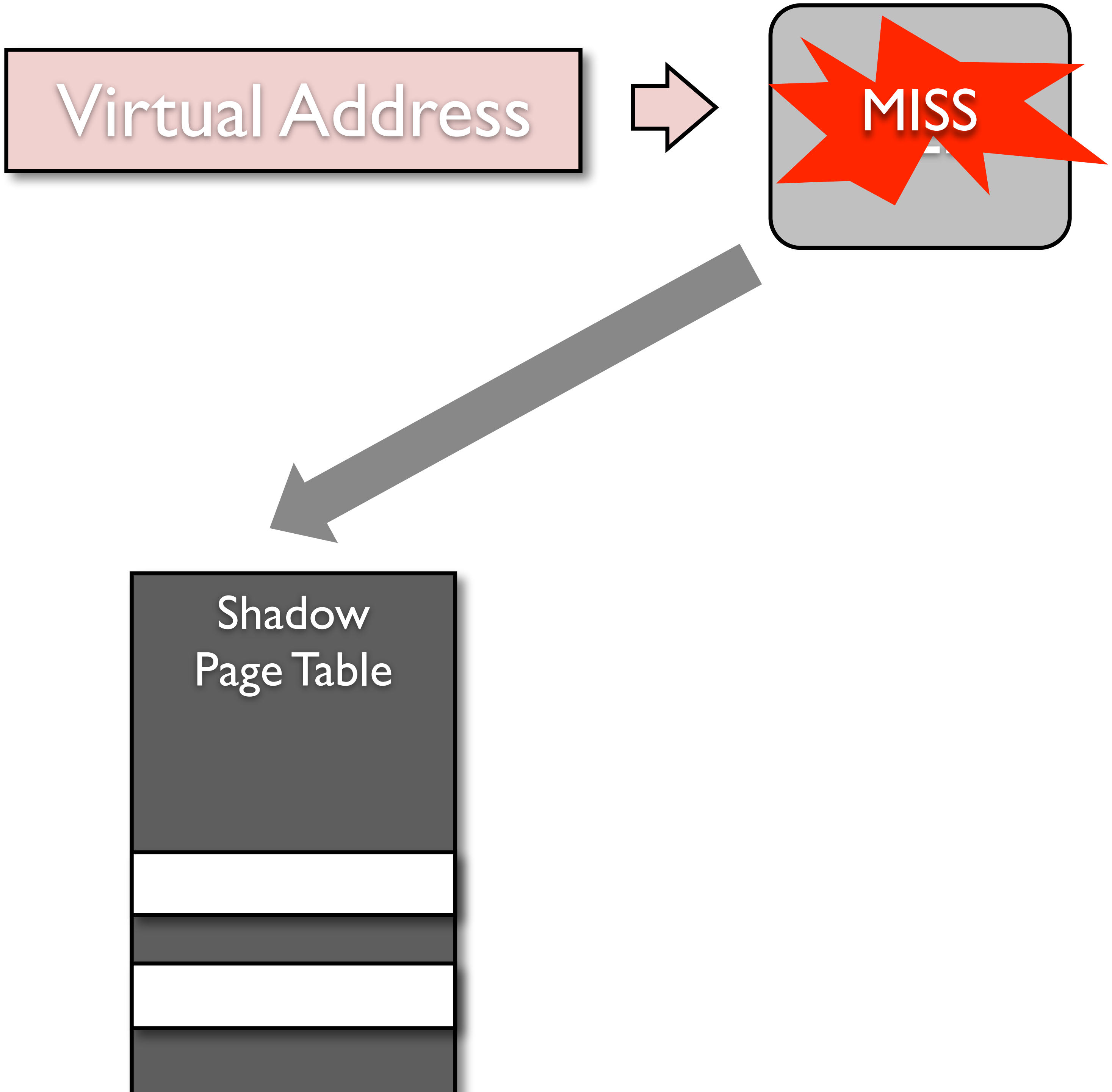
Virtual Address

Shadow Page Tables with TLB

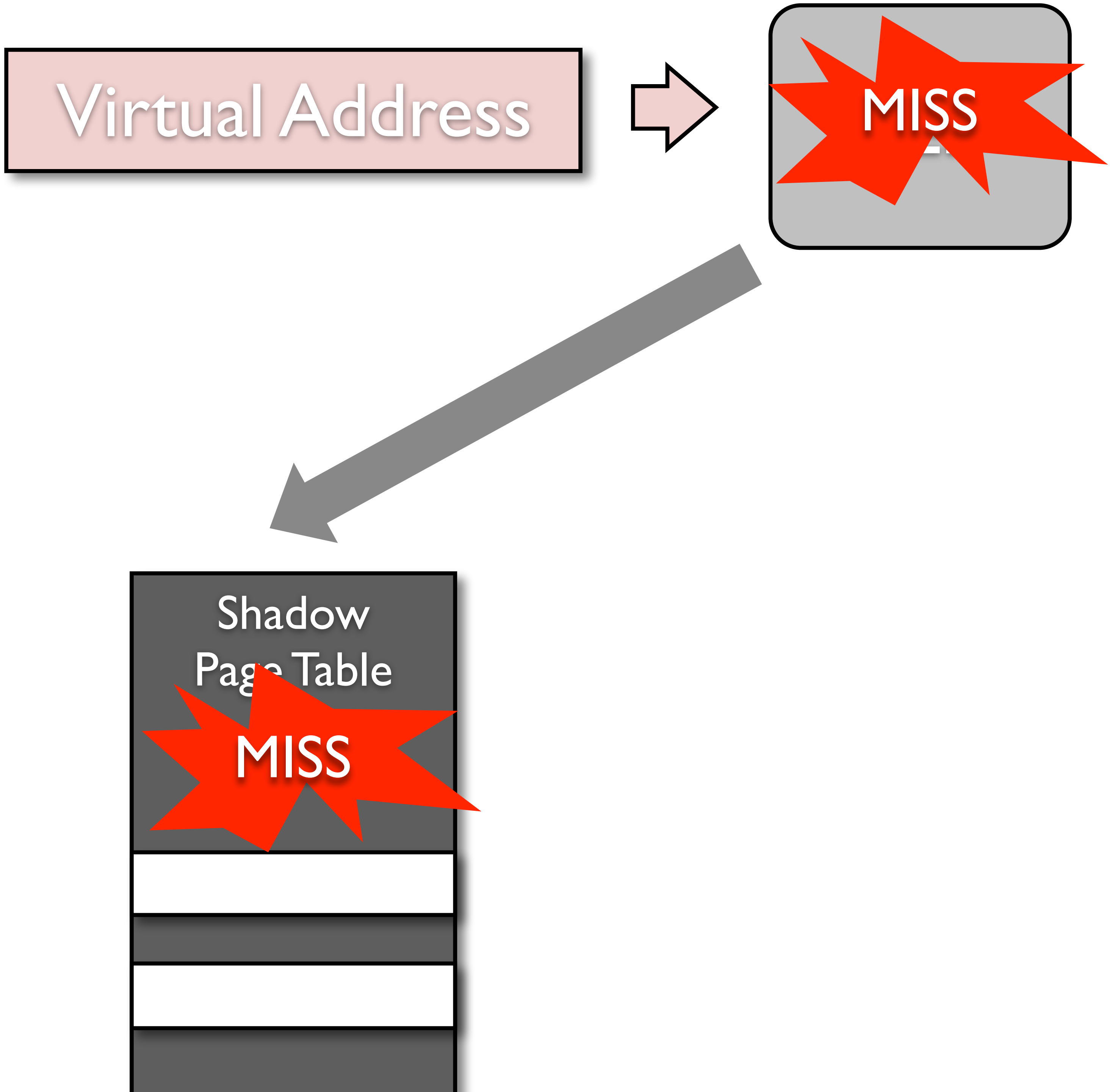


Shadow Page Tables with TLB

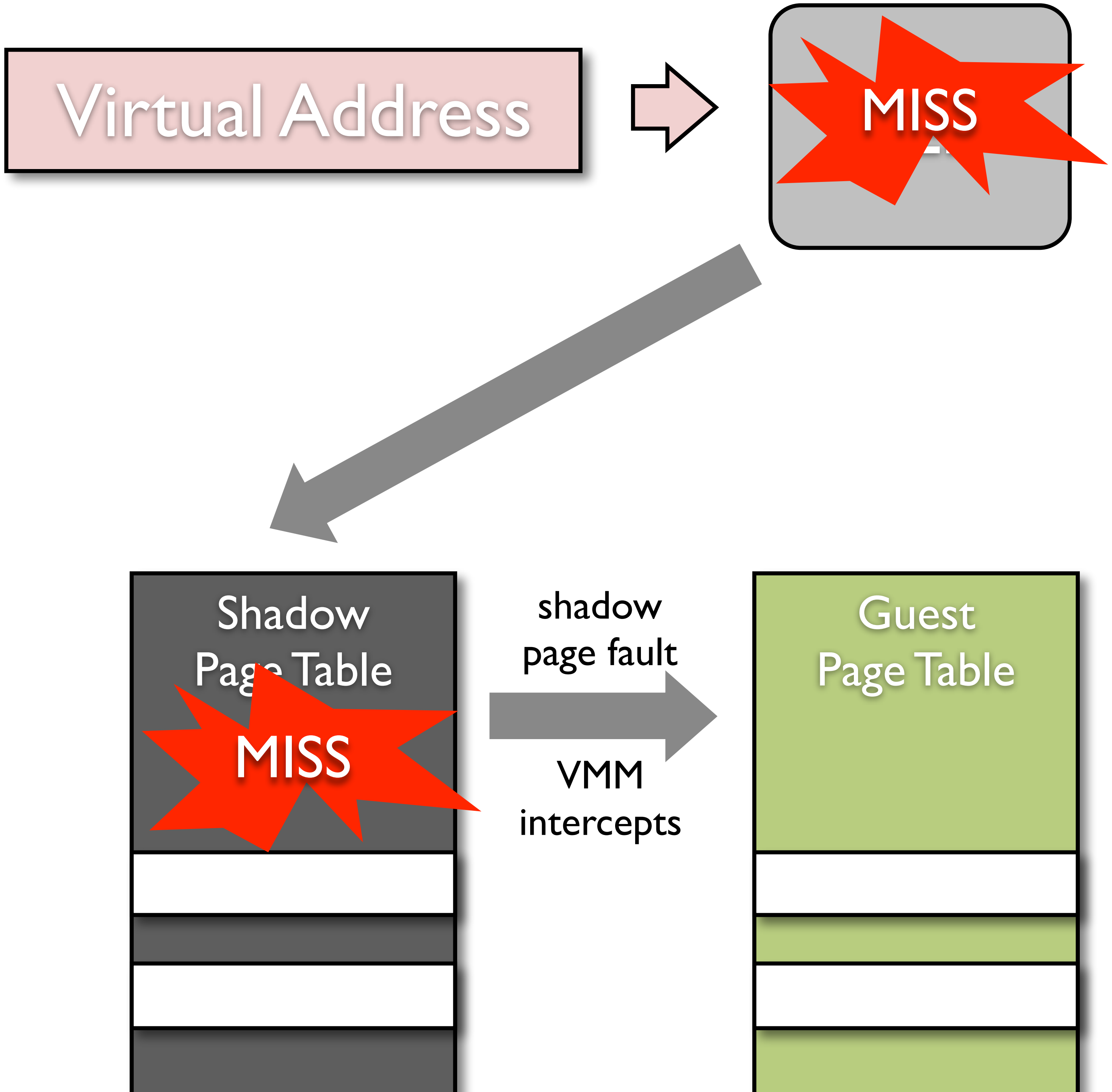



Virtual Address
MISS
Shadow
Page Table

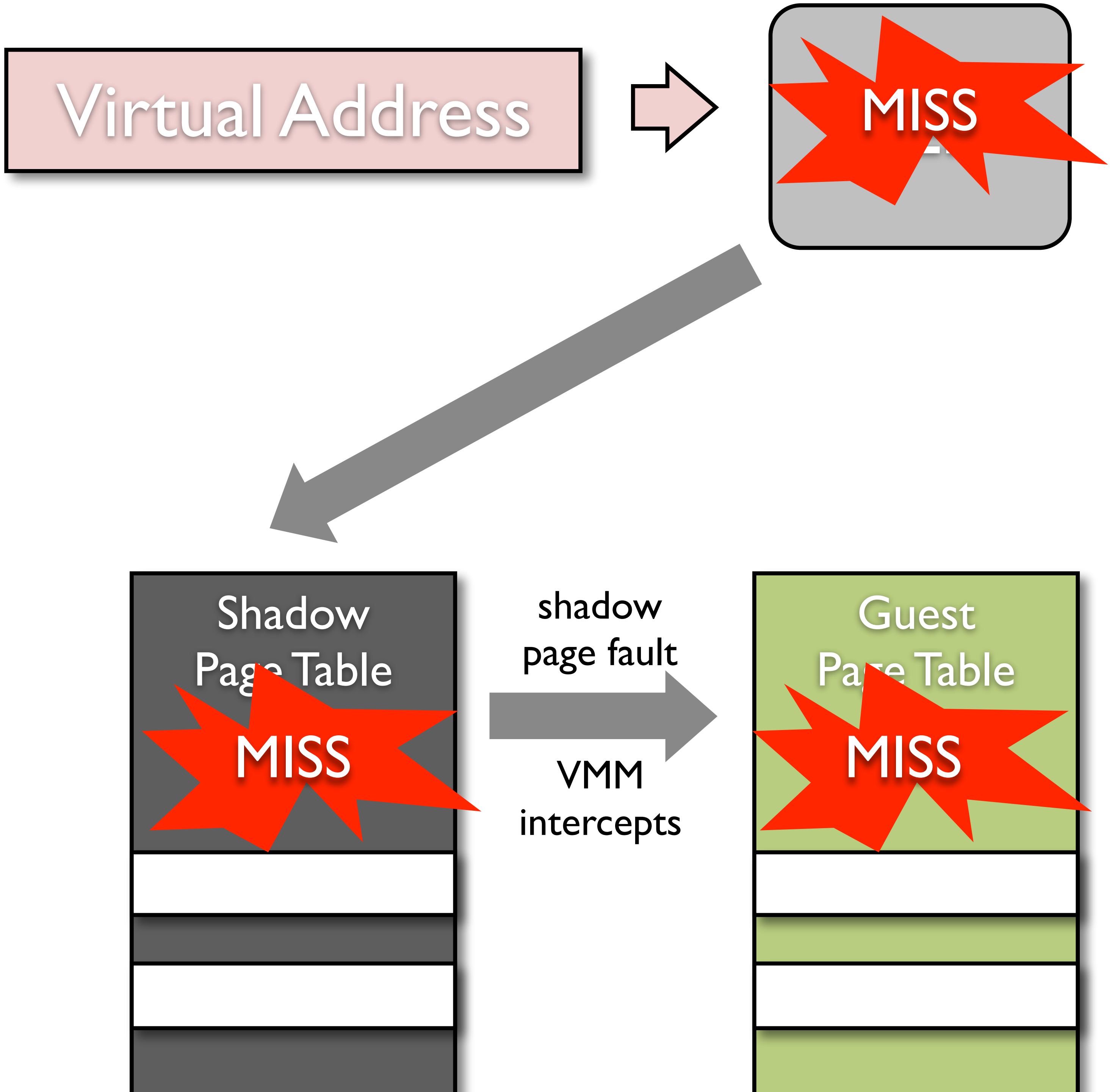
Shadow Page Tables with TLB



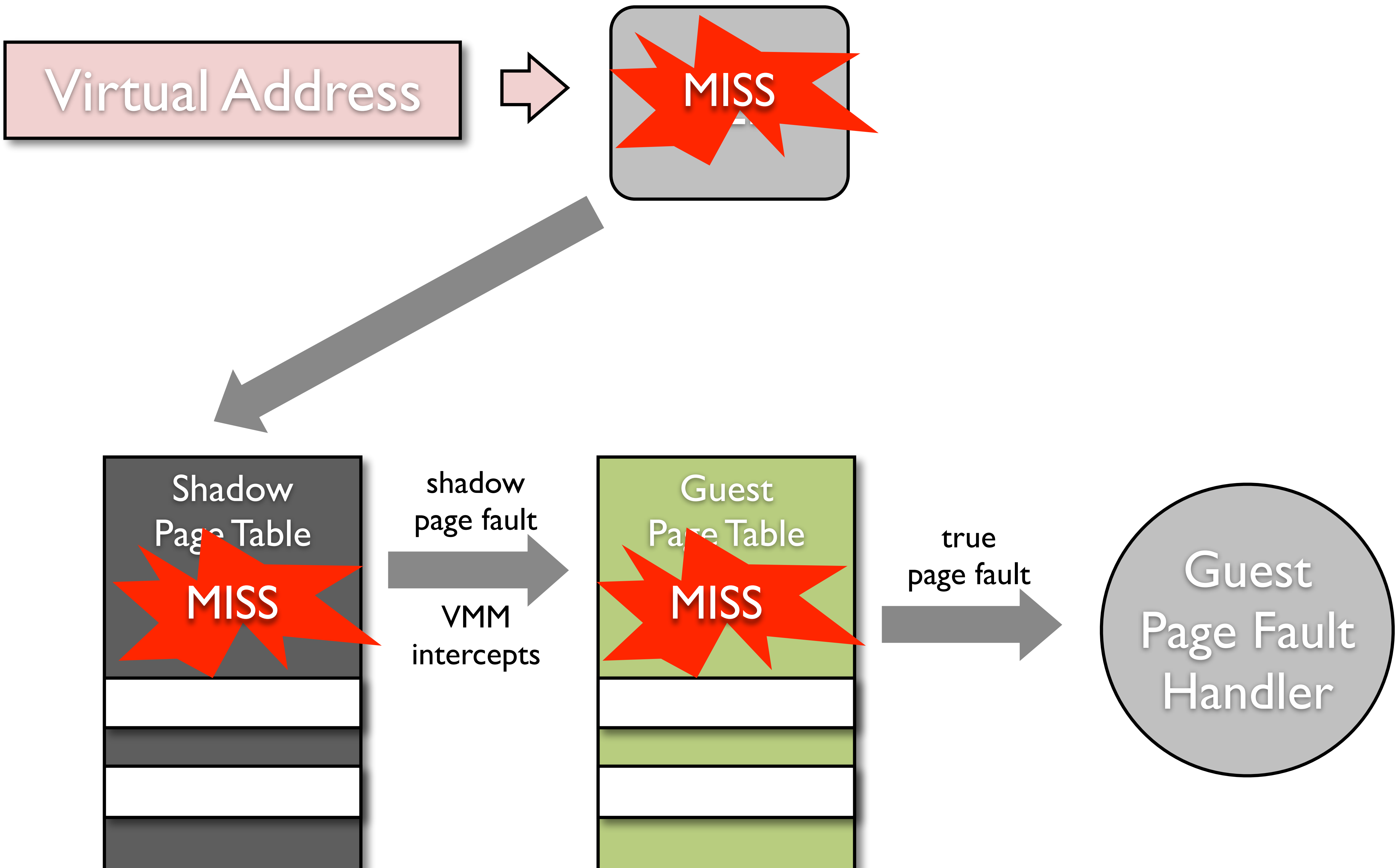
Shadow Page Tables with TLB



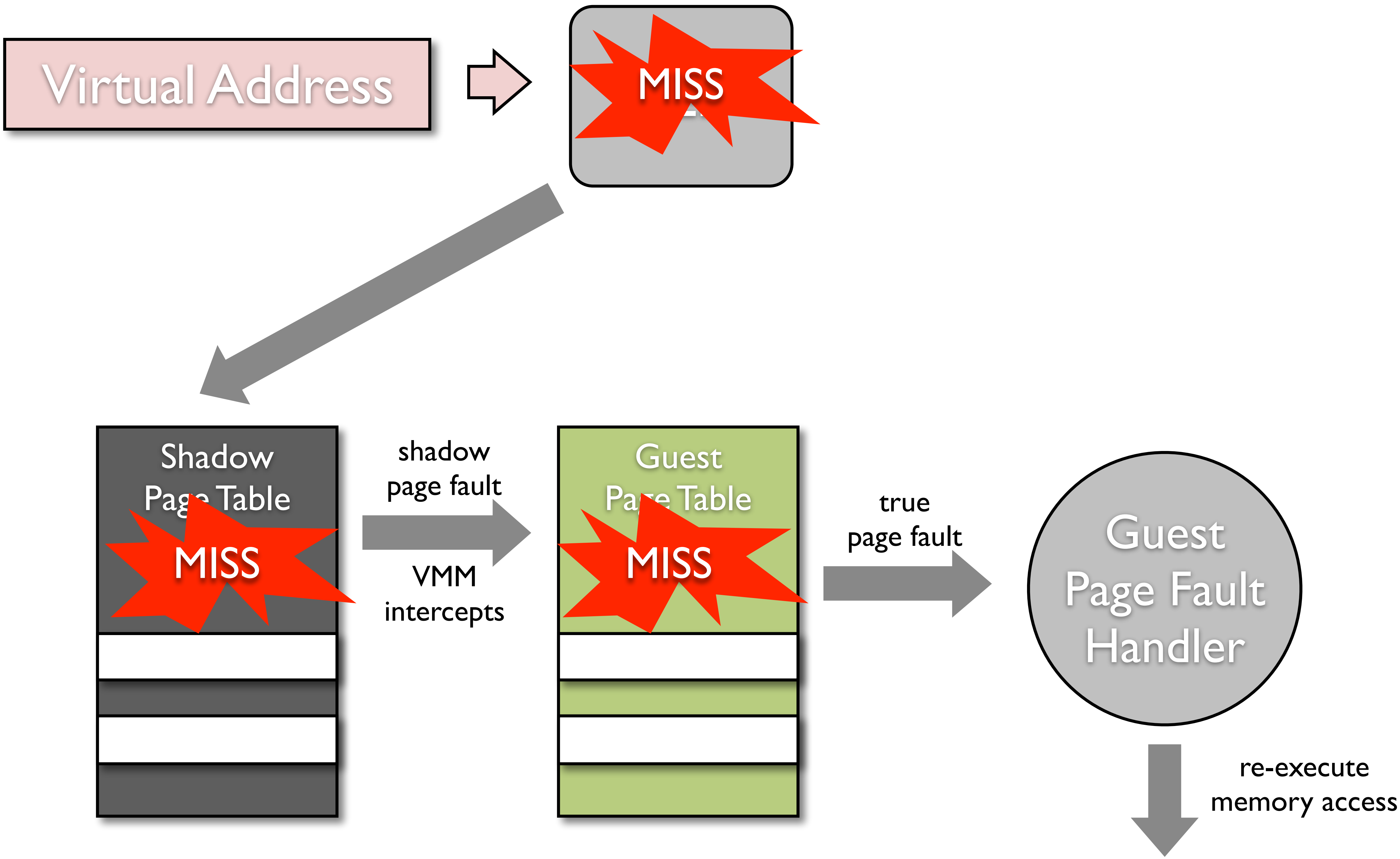
Shadow Page Tables with TLB



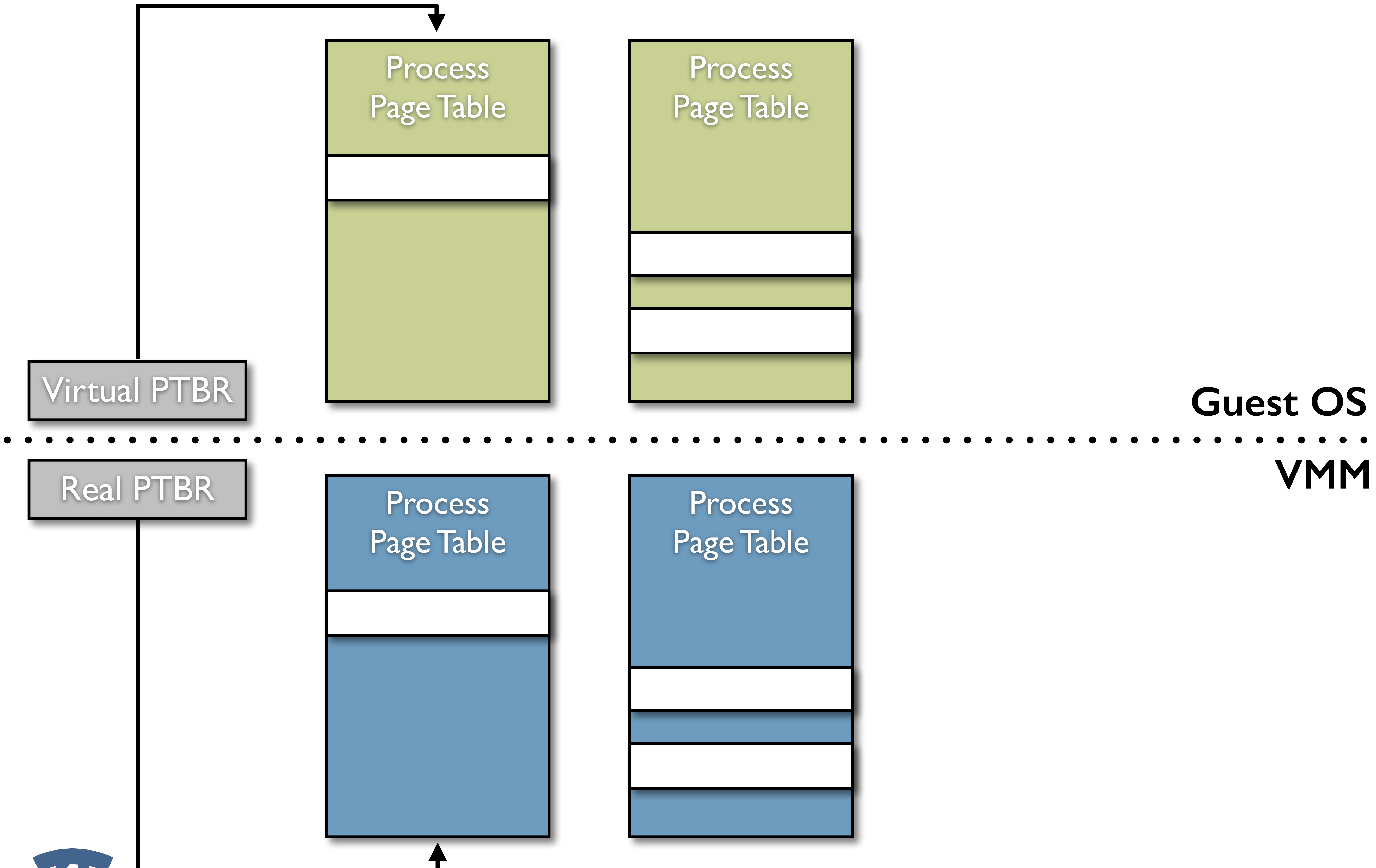
Shadow Page Tables with TLB



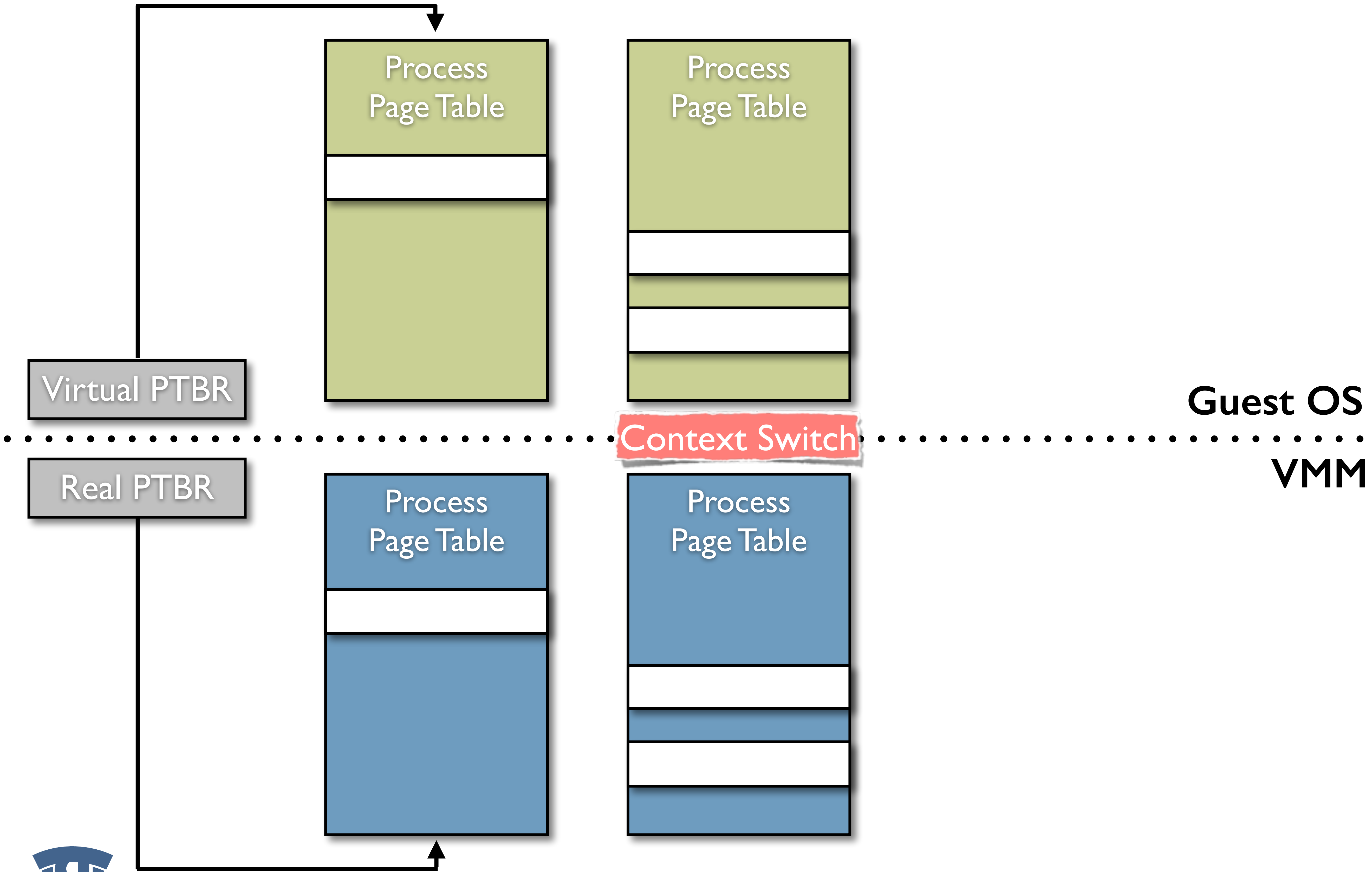
Shadow Page Tables with TLB



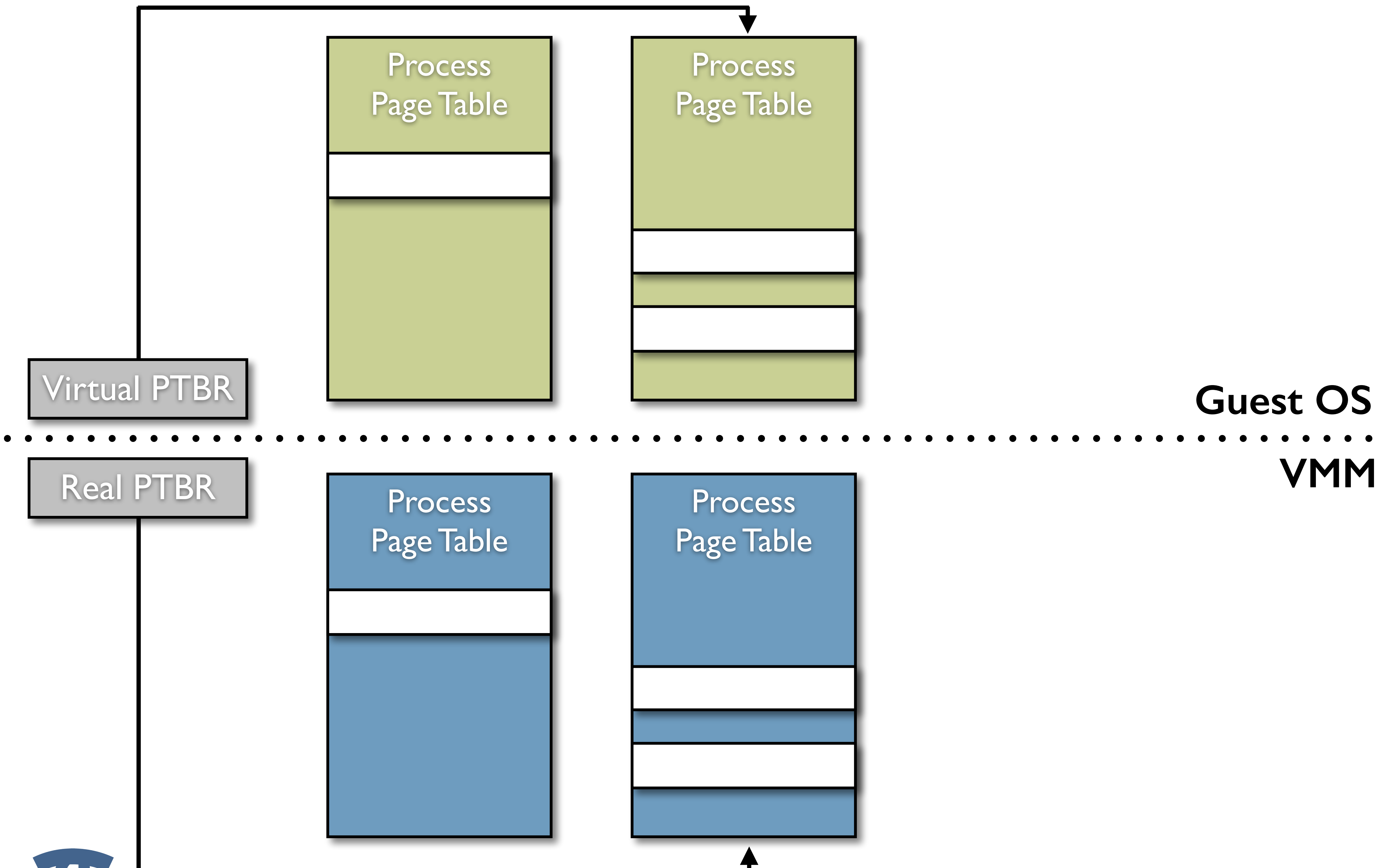
Shadow Page Tables in action...



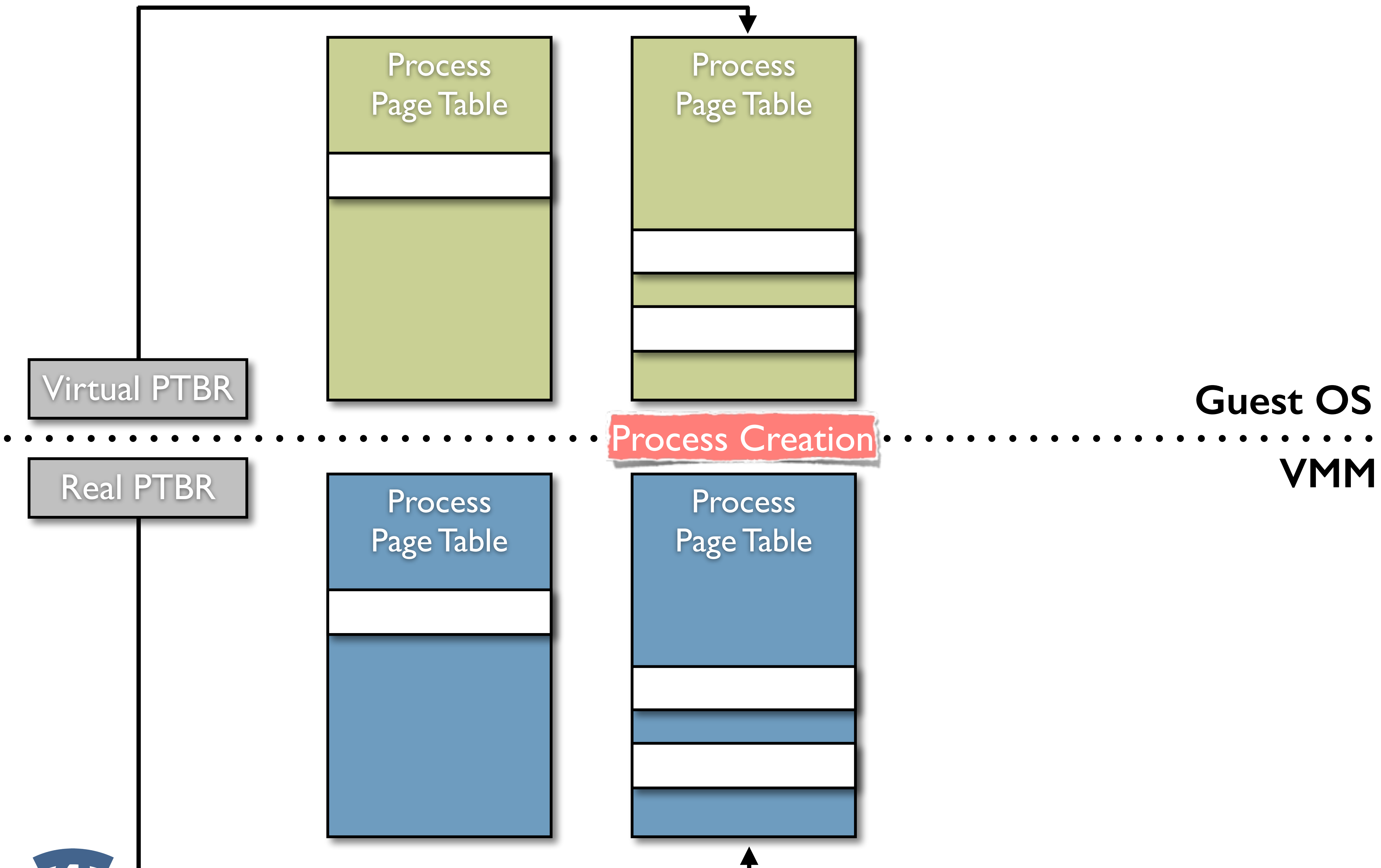
Shadow Page Tables in action...



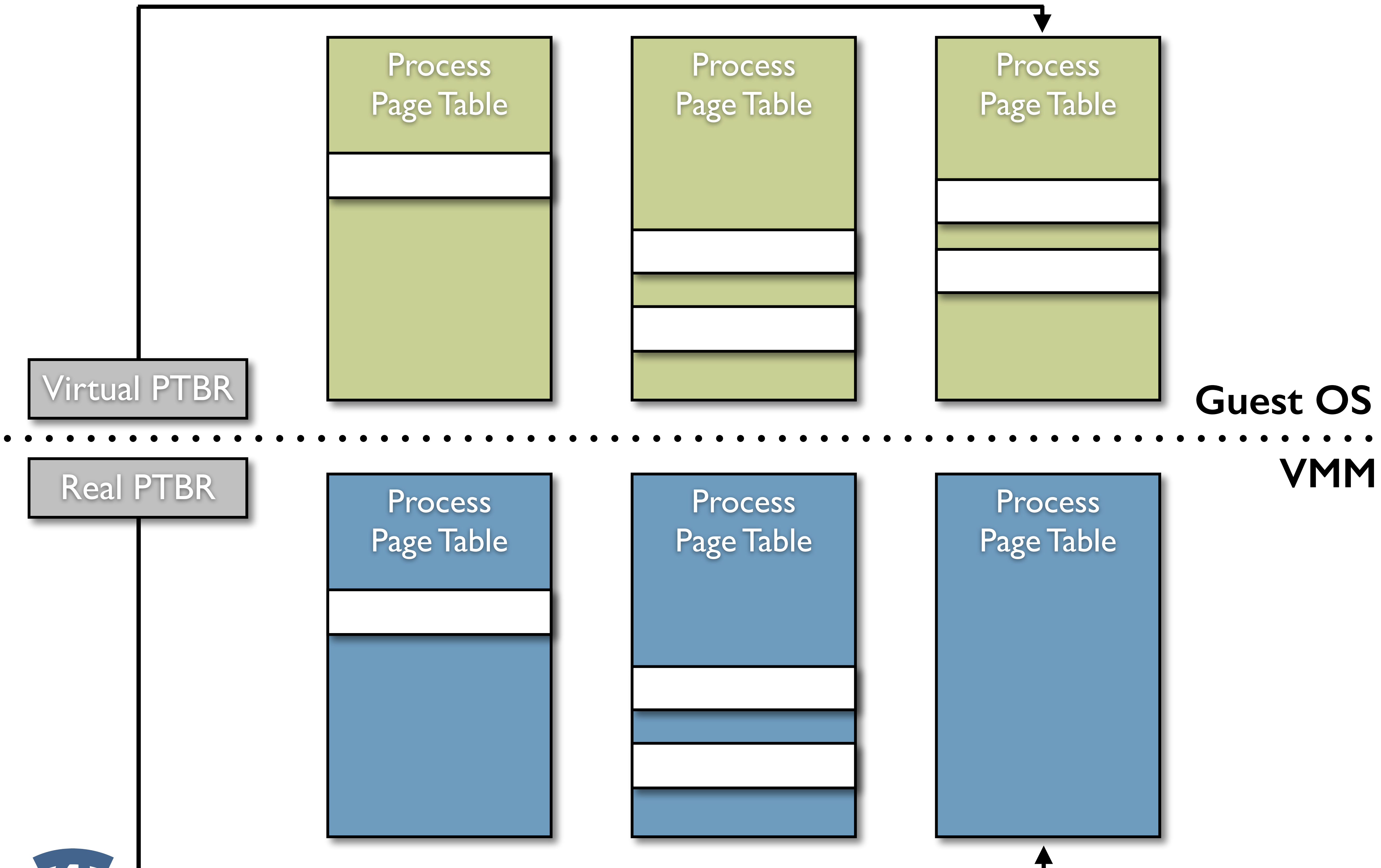
Shadow Page Tables in action...



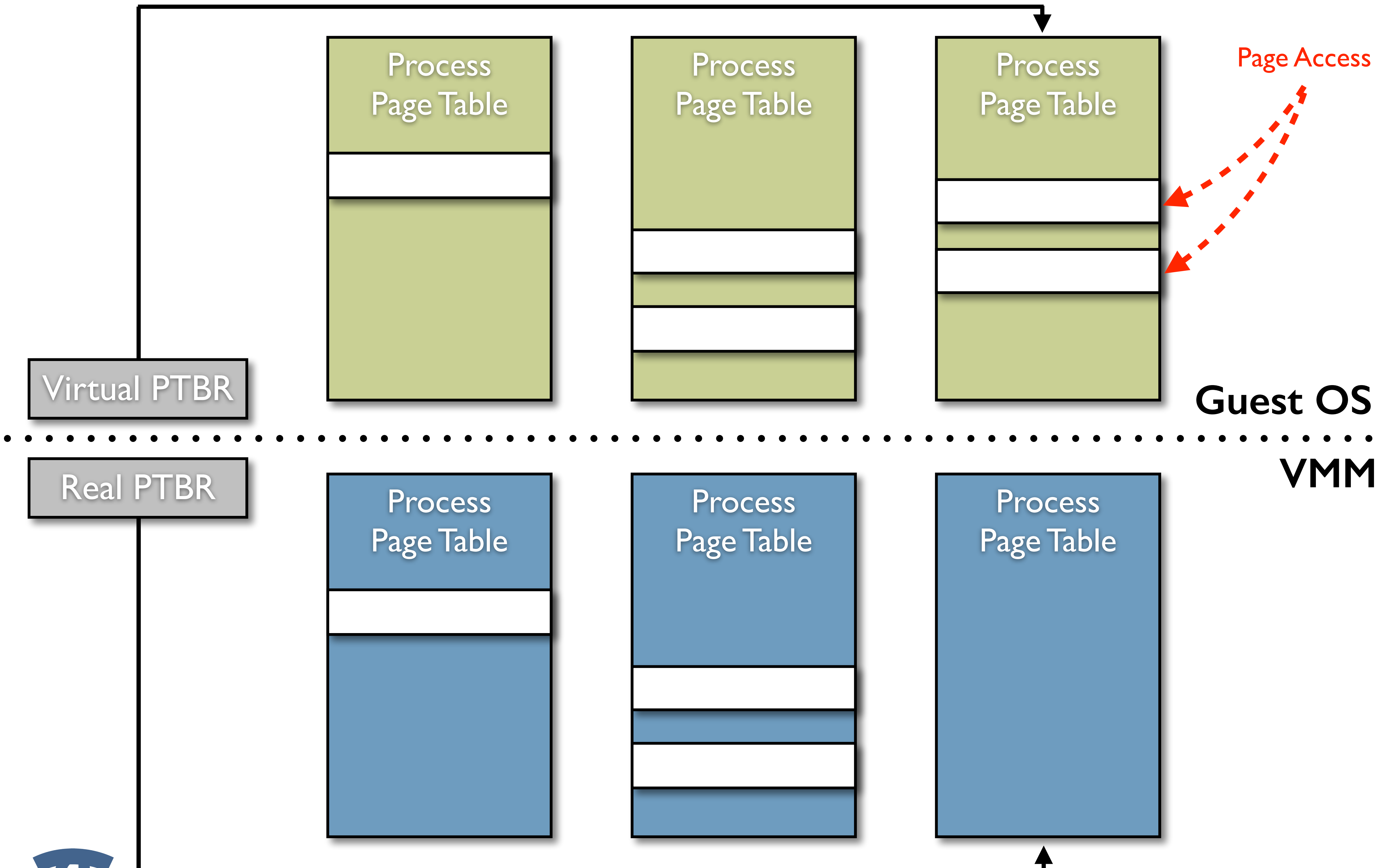
Shadow Page Tables in action...



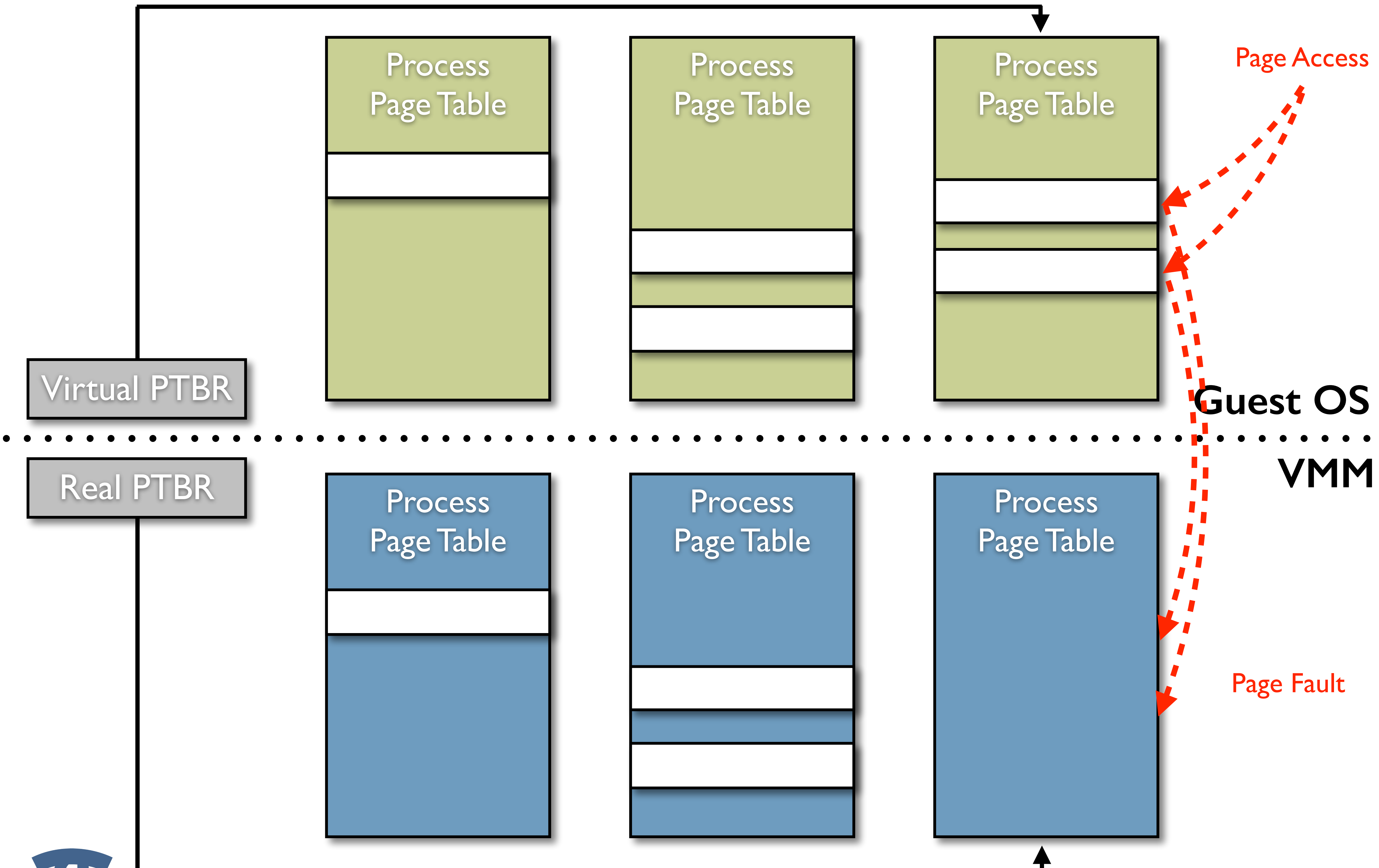
Shadow Page Tables in action...



Shadow Page Tables in action...



Shadow Page Tables in action...



Shadow Page Tables in action...