

Virtualization

Mengjia Yan

Computer Science & Artificial Intelligence Lab
M.I.T.

Abstractions

Software



Instruction set + memory

Computer architecture

Processors, caches, pipelining



Digital circuits

Digital design

Combinational and sequential circuits



Bits, Logic gates

Devices

Materials

Atoms

Abstractions

Computer programs



Virtual machines

Computer systems

Operating systems, virtual memory, I/O



Instruction set + memory

Computer architecture

Processors, caches, pipelining



Digital circuits

Digital design

Combinational and sequential circuits



Bits, Logic gates

Devices

Materials

Atoms

Evolution in Number of Users

IBM 1620
1959



Single User

Runtime
loaded with
program

IBM 360
1960s



Multiple Users

OS for
sharing
resources

IBM PC
1980s



Single User

OS for
sharing
resources

Cloud Servers
1990s

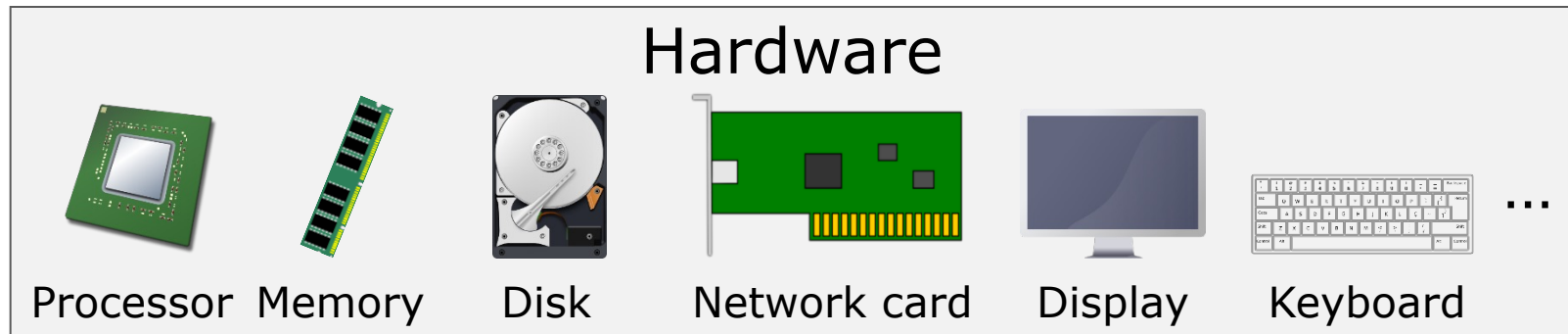


Multiple Users

Multiple OSs

Single-Program Machine

Program



ISA

- Hardware executes a single program and has direct and complete access to all hardware resources
- The ISA is the interface between software and hardware:
 - Program counter
 - General purpose registers
 - Memory

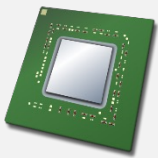
Single-Program Machine (with RTL)

Program

Runtime Library

RTL
API
ISA

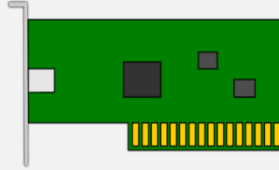
Hardware



Processor Memory



Disk



Network card



Display



Keyboard

...

- Runtime library added to save programming effort and provided *an abstraction to create uniform interface to devices*.

Multi-Program Machine (1st attempt)

Program

Program

RTL

Runtime Library

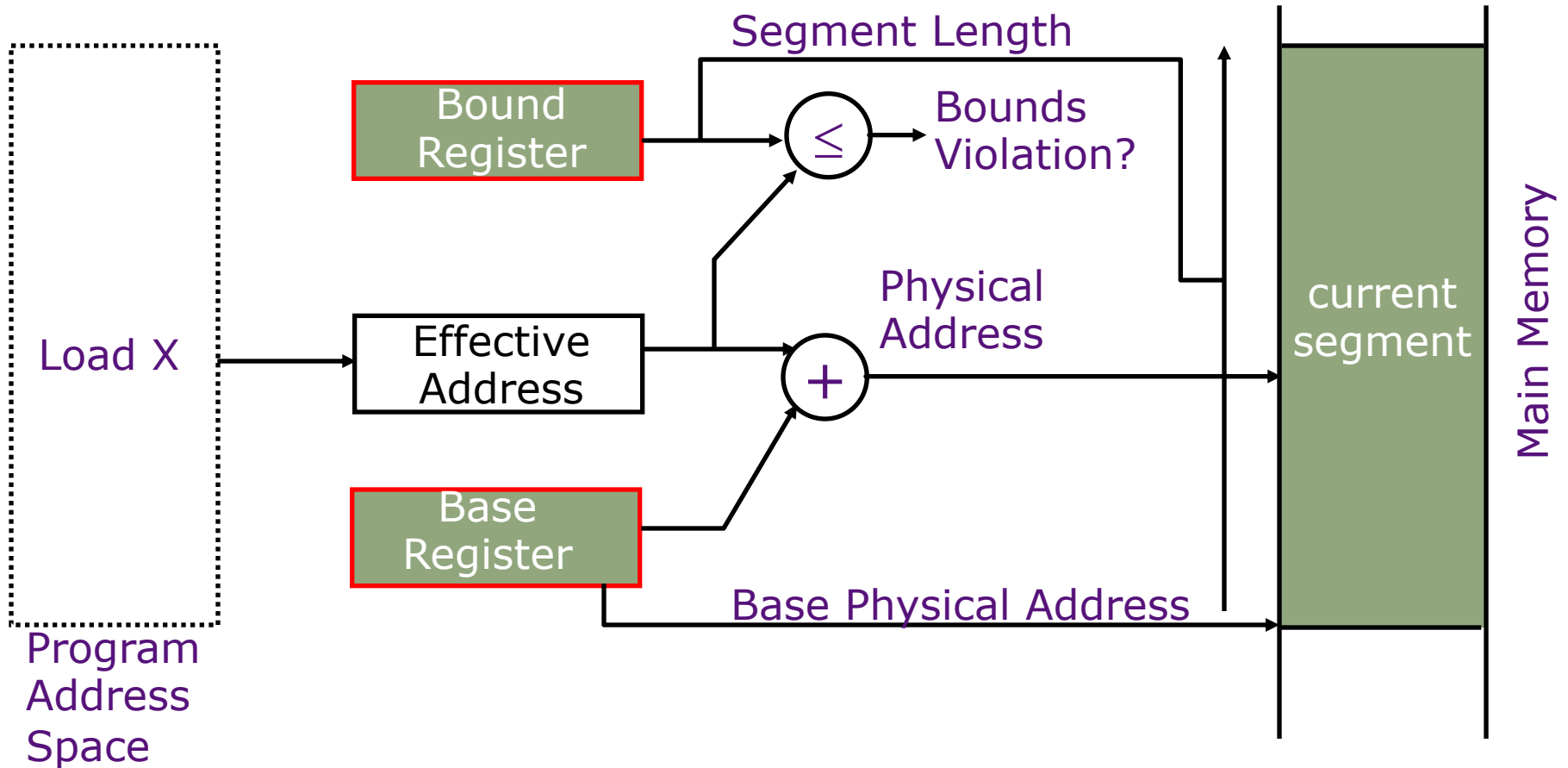
API

ISA

Hardware

Any problems?

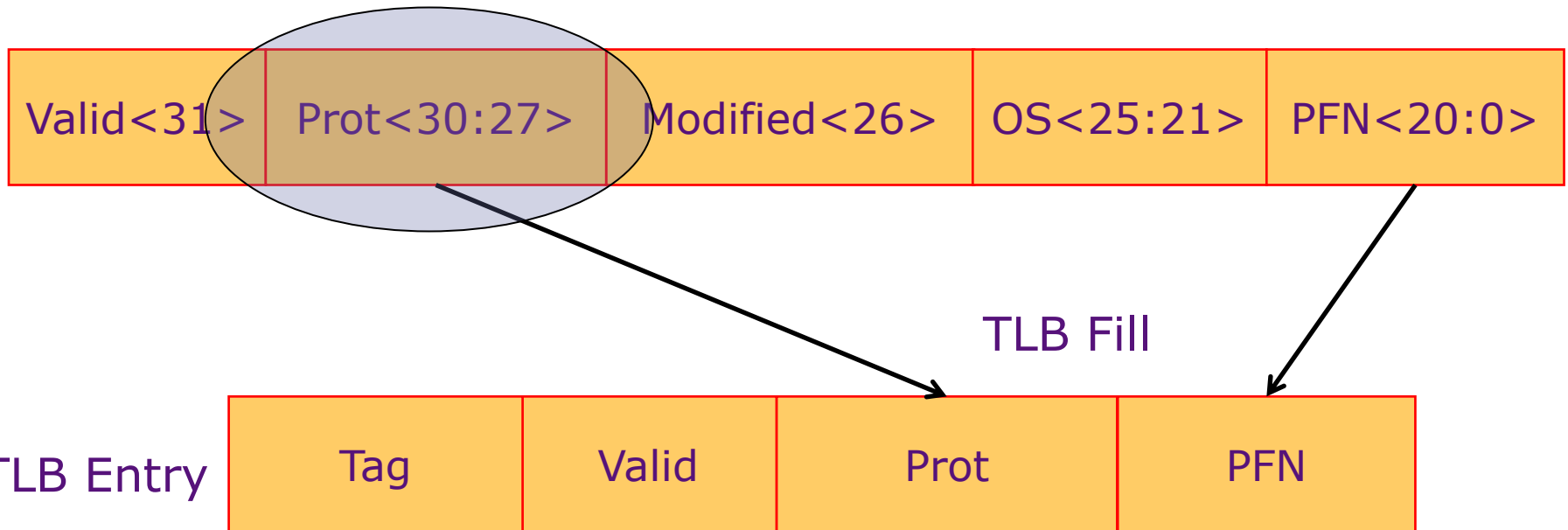
Simple Base and Bound Translation



Introduce **a new privileged mode** in which the base and bounds registers are visible/accessible.

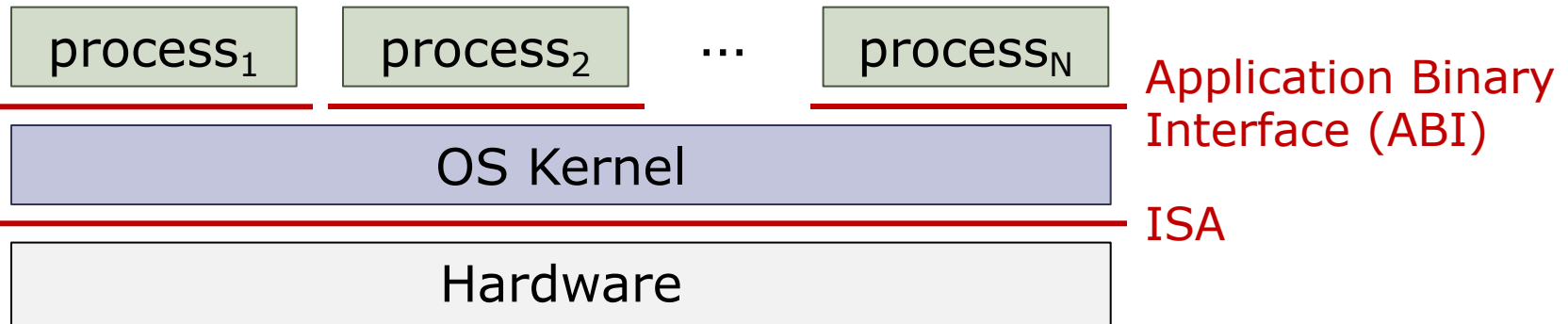
Protecting Memory

Page Table Entry



- TLB access checks if protection allows access for current mode
- TLB fills require read/copy page table data -> security sensitive

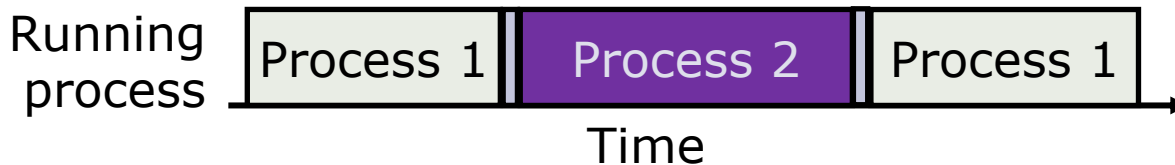
Operating Systems



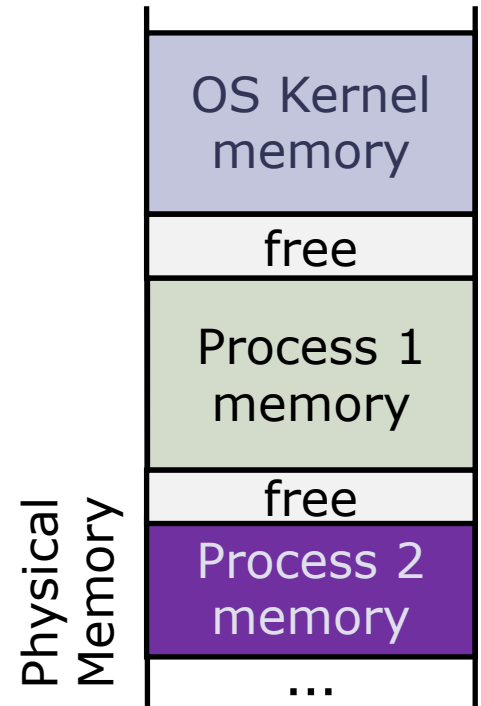
- Operating System (OS) goals:
 - **Abstraction**: OS hides details of underlying hardware
 - e.g., a process can open and access files instead of issuing raw commands to the disk
 - **Resource management**: OS controls how processes share hardware (CPU, memory, disk, etc.)
 - **Protection and privacy**: Processes cannot access each other's data

Operating System Mechanisms

- The OS kernel lets processes invoke system services (e.g., access files or network sockets) via **system calls**
- The OS kernel **schedules processes** into cores
 - Each process is given a fraction of CPU time
 - A process cannot use more CPU time than allowed



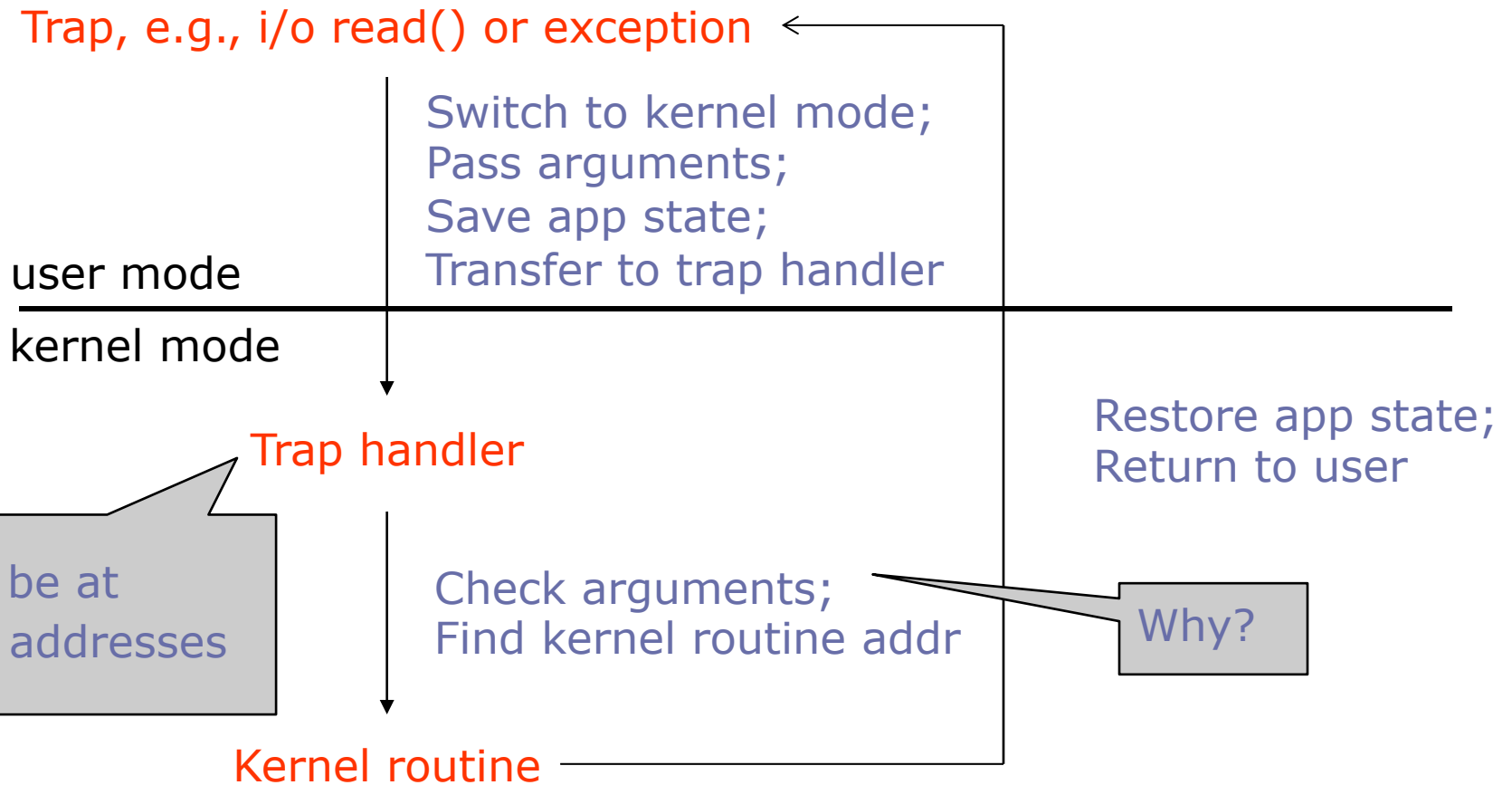
- The OS kernel provides a **private address space** to each process
 - Each process is allocated space in physical memory by the OS
 - A process is not allowed to access the memory of other processes



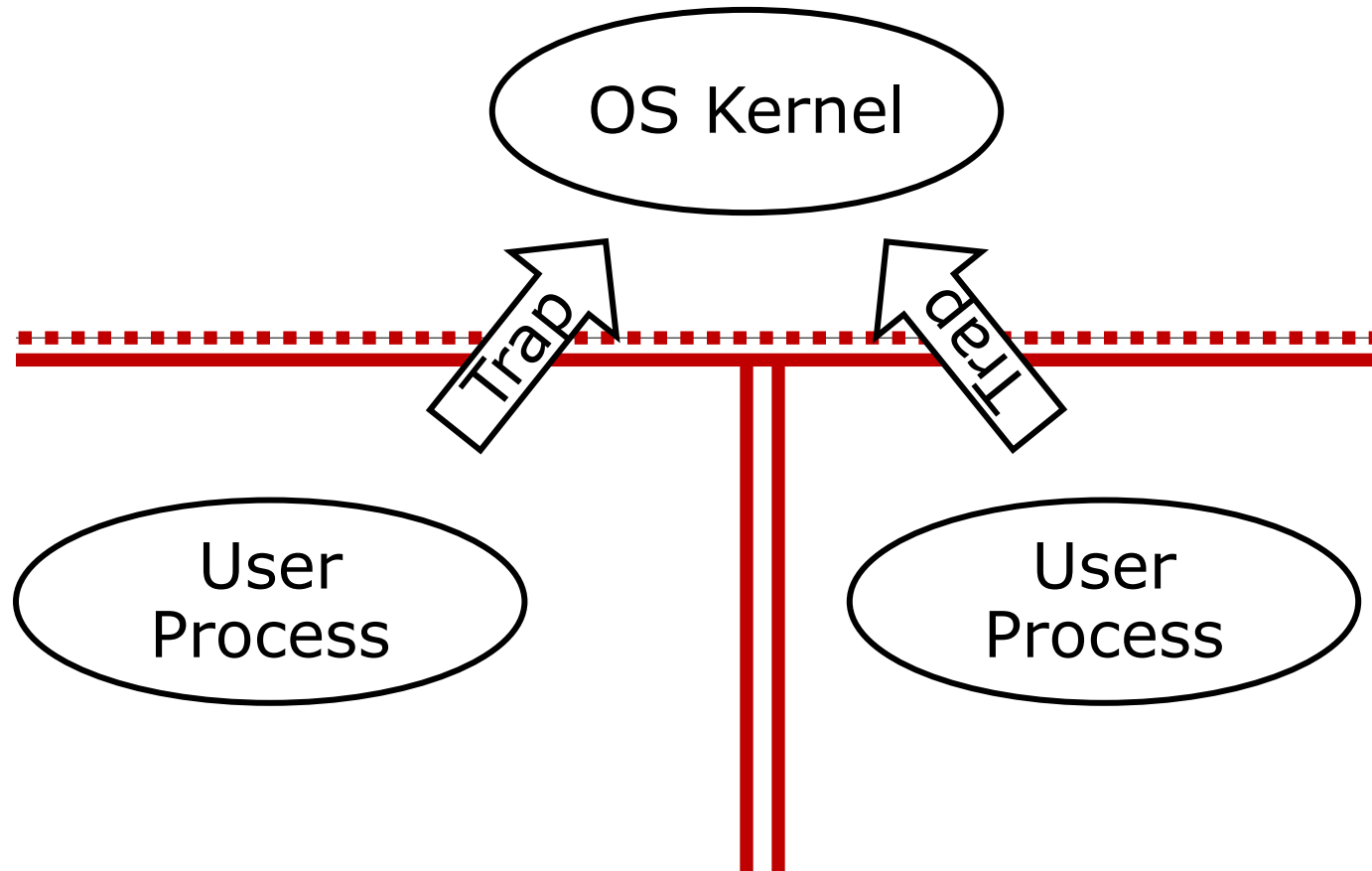
ISA Extensions to Support OS

- Two modes of execution: **user** and **supervisor**
 - OS kernel runs in supervisor mode
 - All other processes run in user mode
- **Privileged instructions and registers** that are only available in supervisor mode
- How to transition from user mode to supervisor mode?
 - **Traps (exceptions)** to safely transition from user to supervisor mode

Process Mode Switching



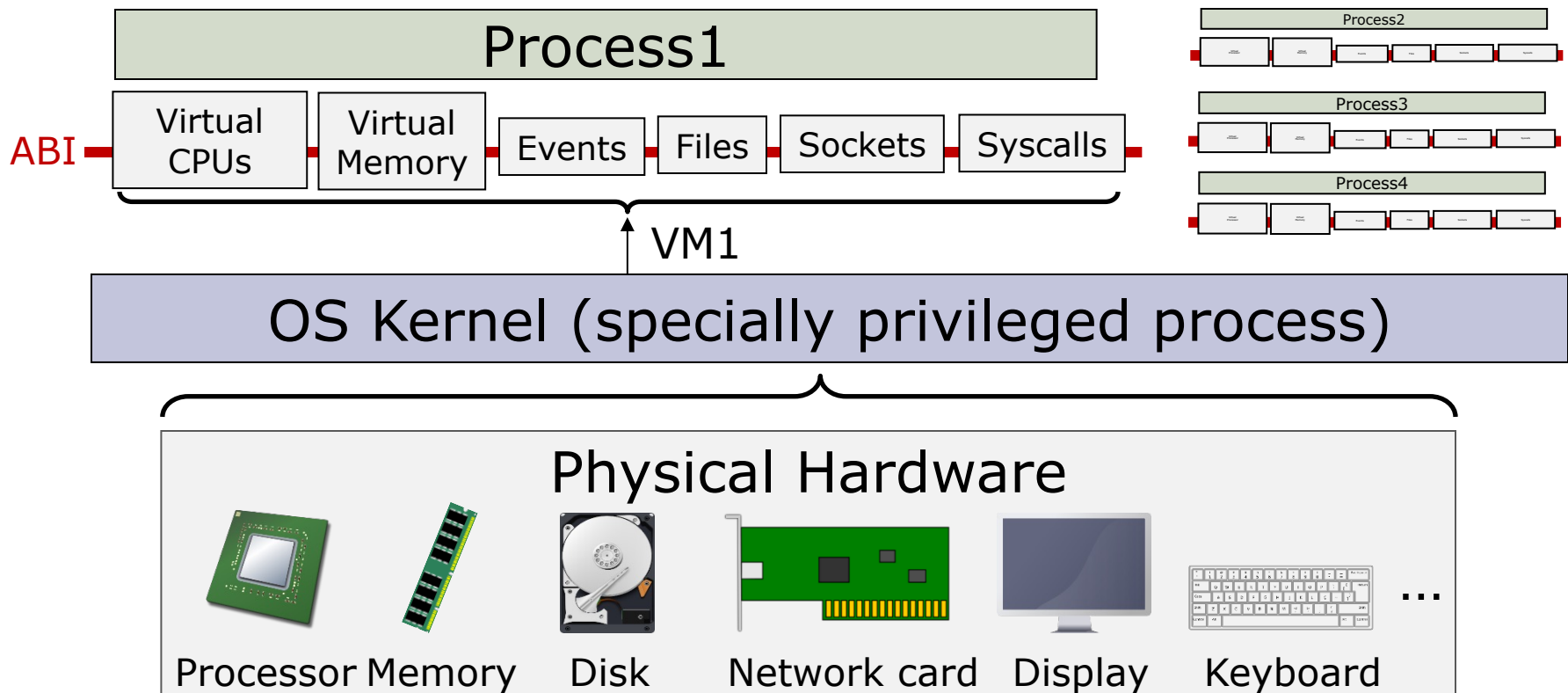
Protection – Single OS



Key idea: Provides a strong abstraction that cannot be escaped

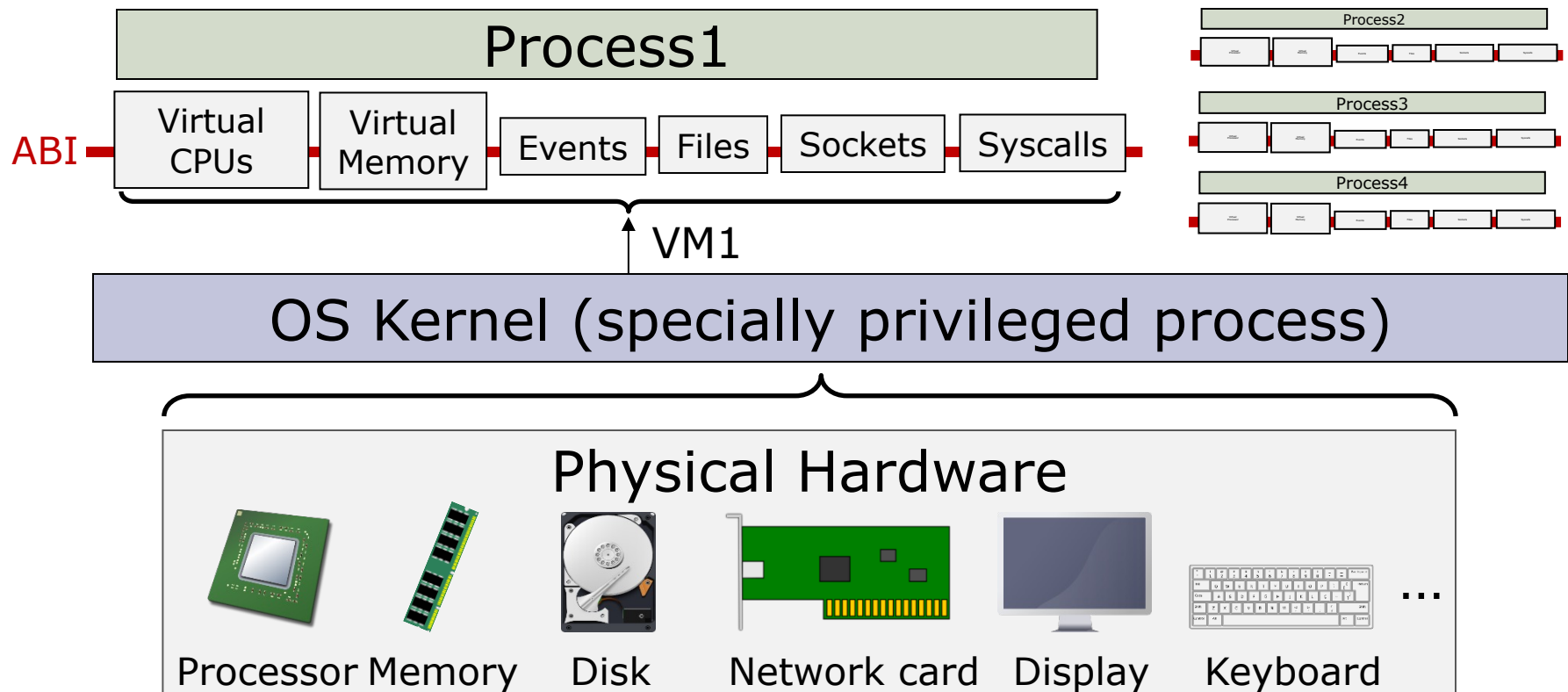
Virtual Machines

- The OS gives a **Virtual Machine (VM)** to each process
 - Each process believes it runs on its own machine...
 - ...but this machine does not exist in physical hardware



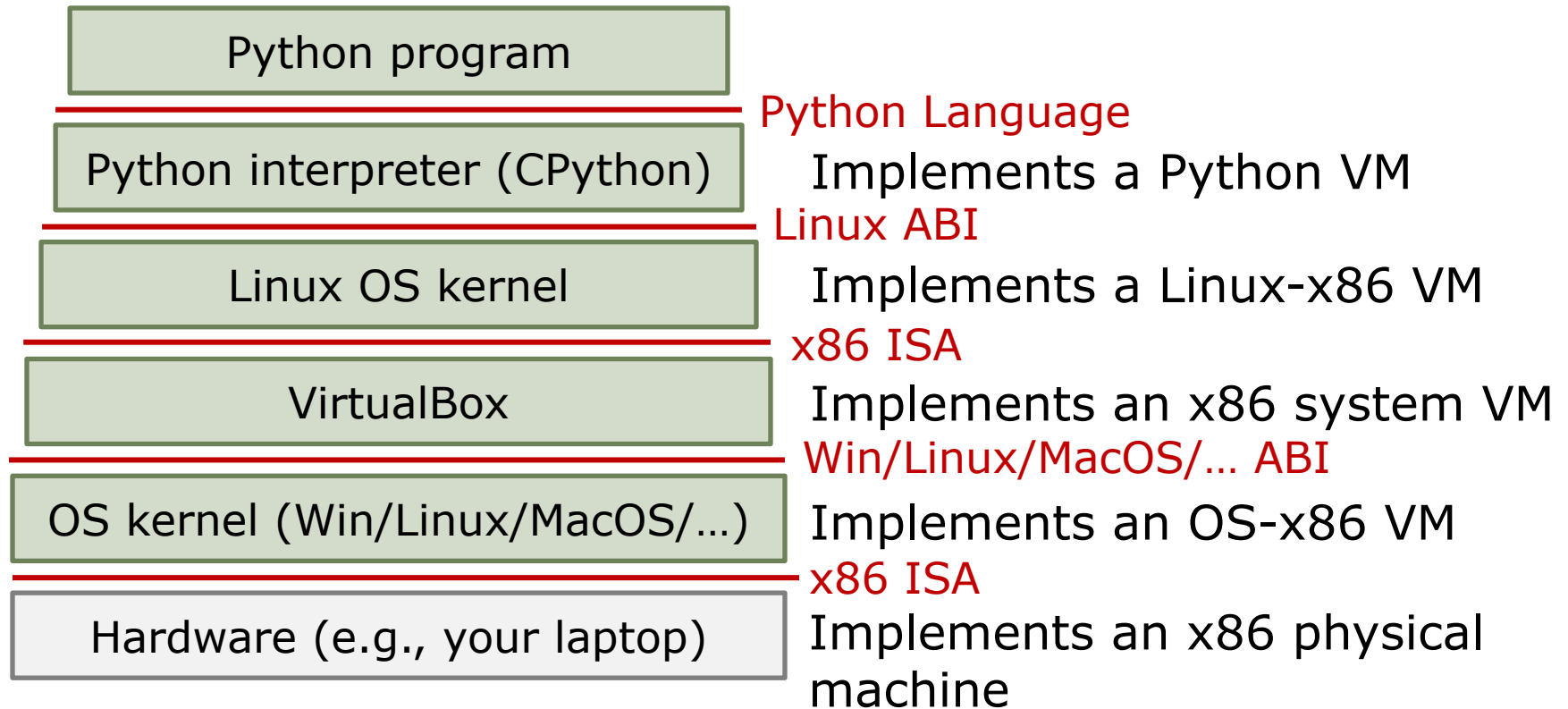
Virtual Machines

- A Virtual Machine (VM) is an **emulation** of a computer system
 - Very general concept, used beyond operating systems



Virtual Machines Are Everywhere

- Example: Consider a Python program running on a Linux Virtual Machine



Implementing Virtual Machines

- Virtual machines can be implemented entirely in software, but at a performance cost
 - e.g., Python programs are 10-100x slower than native Linux programs due to Python interpreter overheads
- We want to support virtual machines with minimal overheads → often need hardware support!

Application-level virtualization

- Programs are usually distributed in *a binary format*:
 - Encodes the program's instructions and initial values of data segments.
 - Conforms to the application binary interface (ABI).
- ABI specifications include
 - Which instructions are available (the ISA)
 - What system calls are possible (I/O, or the *environment*)
 - What state is available at process creation
- Operating system implements the virtual environment
 - At process startup, OS reads the binary program, creates an environment for it, then begins to execute the code, handling traps for I/O calls, emulation, etc.

Full ISA-Level Virtualization

Run programs for one ISA on hardware with different ISA (for compatibility, platform-independent):

- Run-time Hardware Emulation
 - IBM System 360 had IBM 1401 emulator in microcode
 - Intel Itanium converted x86 to native VLIW (two software-visible ISAs)
 - ARM cores support 64-bit ARM, 32-bit ARM, 16-bit Thumb
- Run-time Software Emulation (*OS software interprets instructions*)
 - E.g., OS for PowerPC Macs had emulator for 68000 code
- Static Binary Translation (*convert at install time, load time, or offline*)
 - IBM AS/400 to modified PowerPC cores
 - DEC tools for VAX->Alpha and MIPS->Alpha
- Dynamic Binary Translation (*non-native to native ISA at run-time*)
 - Sun's HotSpot Java JIT (just-in-time) compiler
 - Transmeta Crusoe, x86->VLIW code morphing

Partial ISA-level virtualization

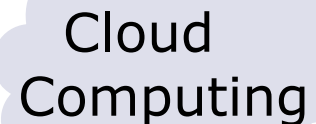
Implement part of ISA in software to trade-off between performance and cost (make the common things fast):

- Expensive but rarely used instructions can cause trap to OS emulation routine:
 - e.g., decimal arithmetic in μ Vax implementation of VAX ISA
- Infrequent but difficult operand values can cause trap
 - e.g., IEEE floating-point denormals cause traps in almost all floating-point unit implementations
- Old machine can trap unused opcodes, allows binaries for *new* ISA to run on *old* hardware
 - e.g., Sun SPARC v8 added integer multiply instructions, older v7 CPUs trap and emulate

Motivation for Multiple OSs

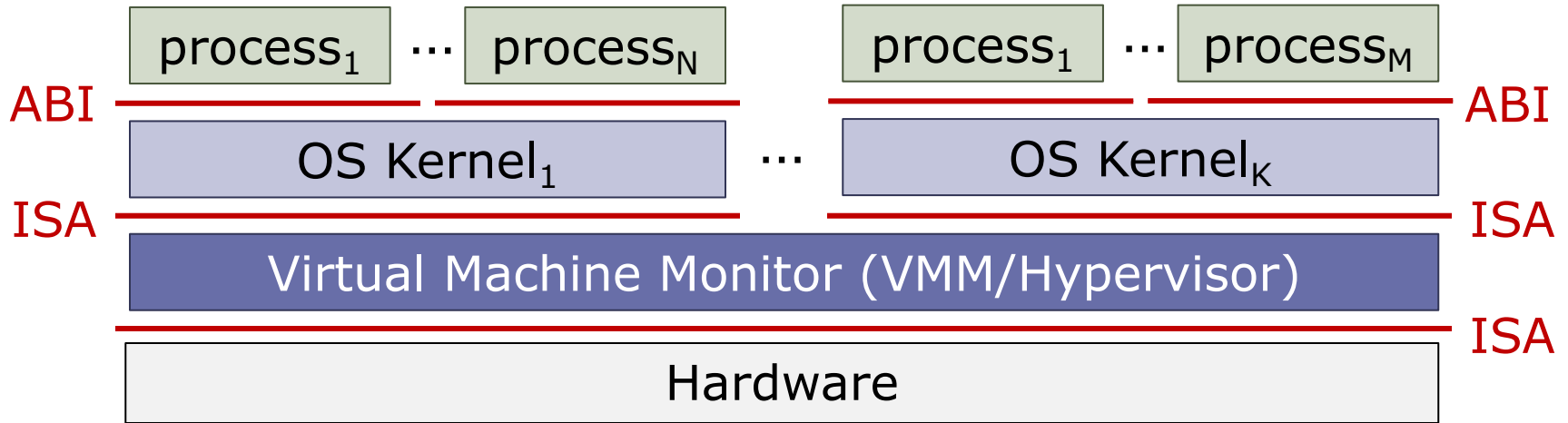
Some motivations for using multiple operating systems on a single computer:

- Allows use of capabilities of multiple distinct operating systems
- Allows different users to share a system while using completely independent software stacks
- Allows for load balancing and migration across multiple machines
- Allows operating system development without making entire machine unstable or unusable



Cloud
Computing

Supporting Multiple OSs



- A VMM (aka Hypervisor) provides a **system virtual machine** to each OS
- VMM can run directly on hardware (as above) or on another OS
 - Precisely, VMM can be implemented against an ISA (as above) or a process-level ABI. Who knows what lays below the interface...

Virtualization Nomenclature

From (Machine we are attempting to execute)

- Guest
- Client
- Foreign ISA

To (Machine that is doing the real execution)

- Host
- Target
- Native ISA

Virtual Machine Requirements

[Popek and Goldberg, 1974]

- Equivalence/Fidelity: A program running on the VMM should exhibit a behavior essentially *identical* to that demonstrated when running on an equivalent machine directly.
- Resource control/Safety: The VMM must be in complete control of the *virtualized resources*.
- Efficiency/Performance: A statistically dominant fraction of machine instructions must be executed without VMM intervention.

Virtual Machine Requirements

[Popek and Goldberg, 1974]

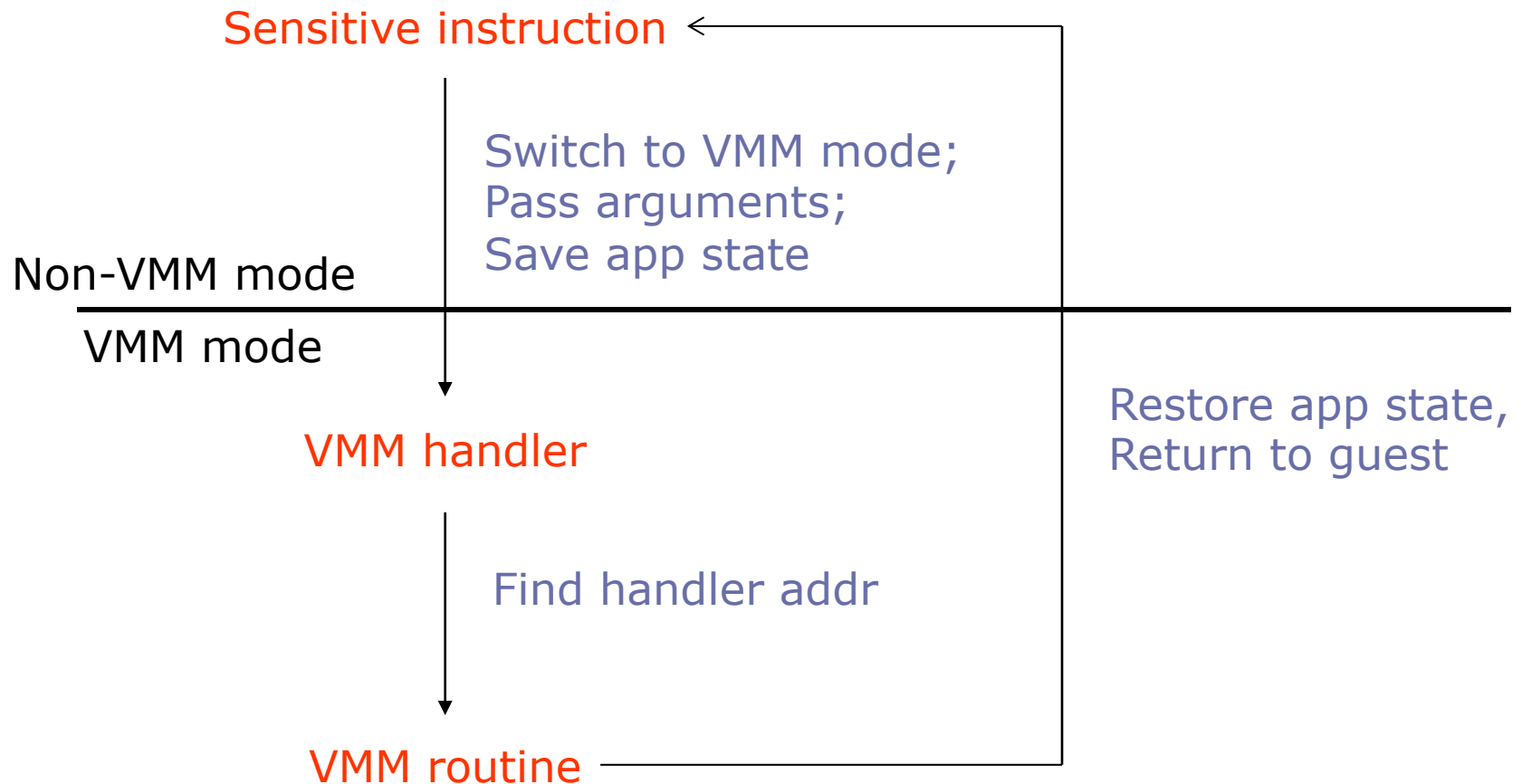
Classification of instructions into 3 groups:

- Privileged instructions: Instructions that **trap** if the processor is in **user mode** and do not trap if it is in a more privileged mode. (*previously defined*)
- Control-sensitive instructions: Instructions that attempt to change the configuration of resources in the system.
- Behavior-sensitive instructions: Those whose behavior depends on the configuration of resources, e.g., mode

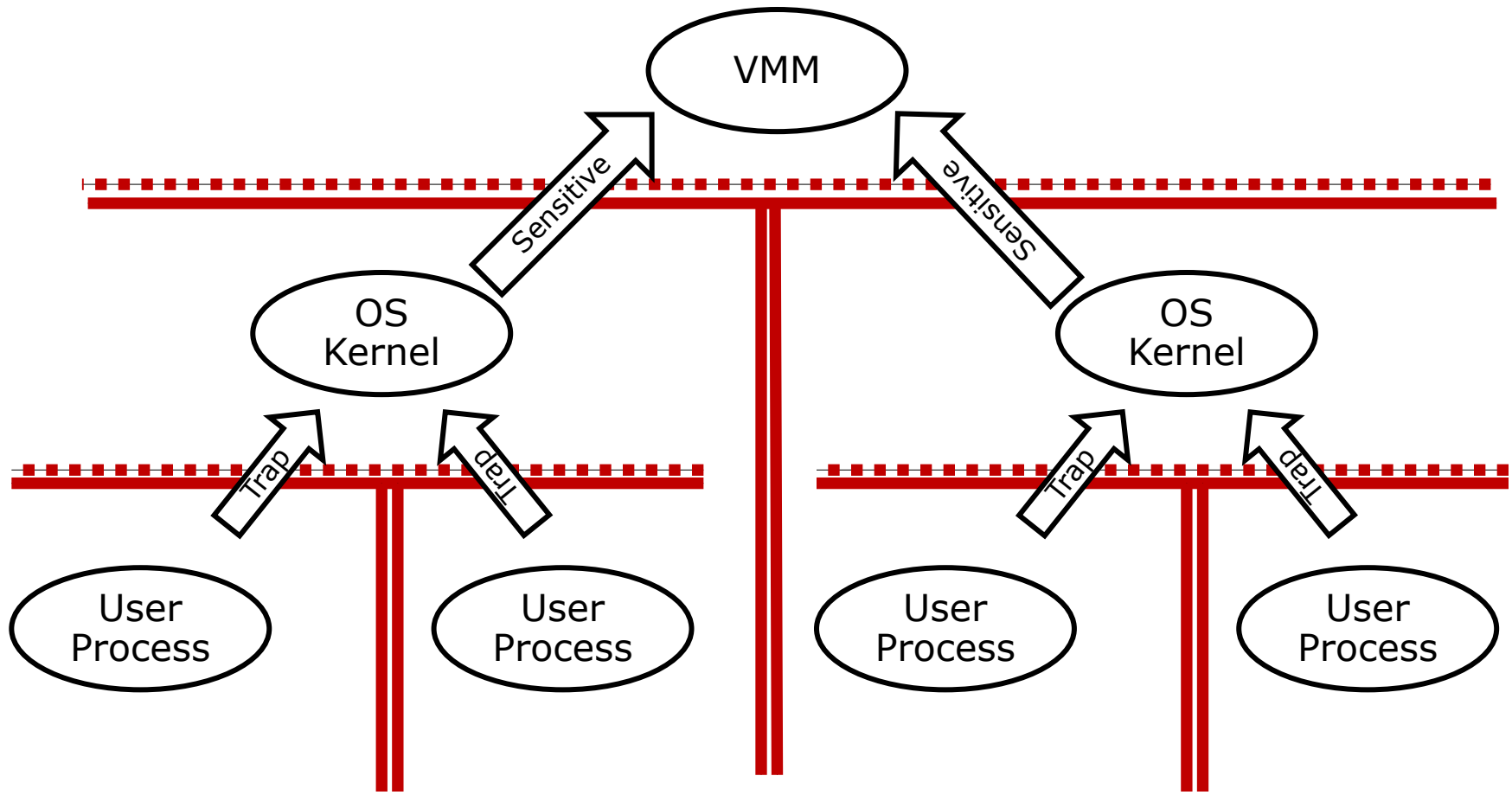
Building an *effective* VMM for an architecture is possible if the set of sensitive instructions is a subset of the set of privileged instructions.

Run guest-OS code using the *trap-and-emulate* strategy.

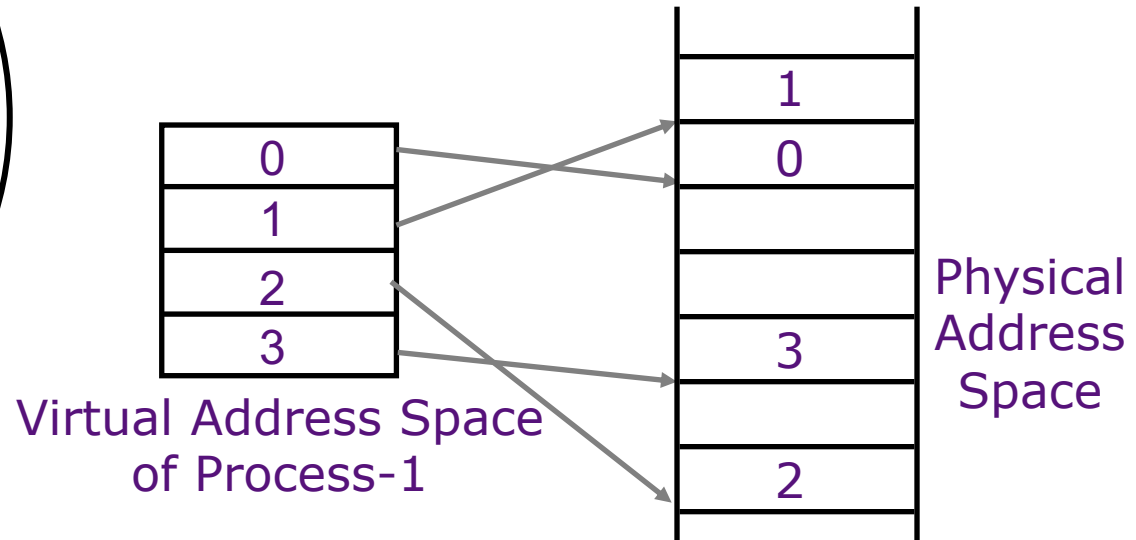
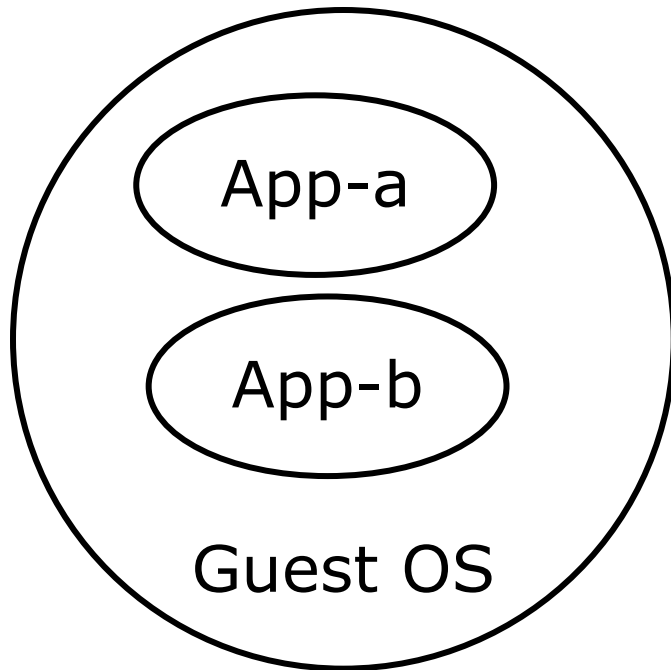
Sensitive instruction handling



Protection – Multiple OS



Virtual Memory in VMs



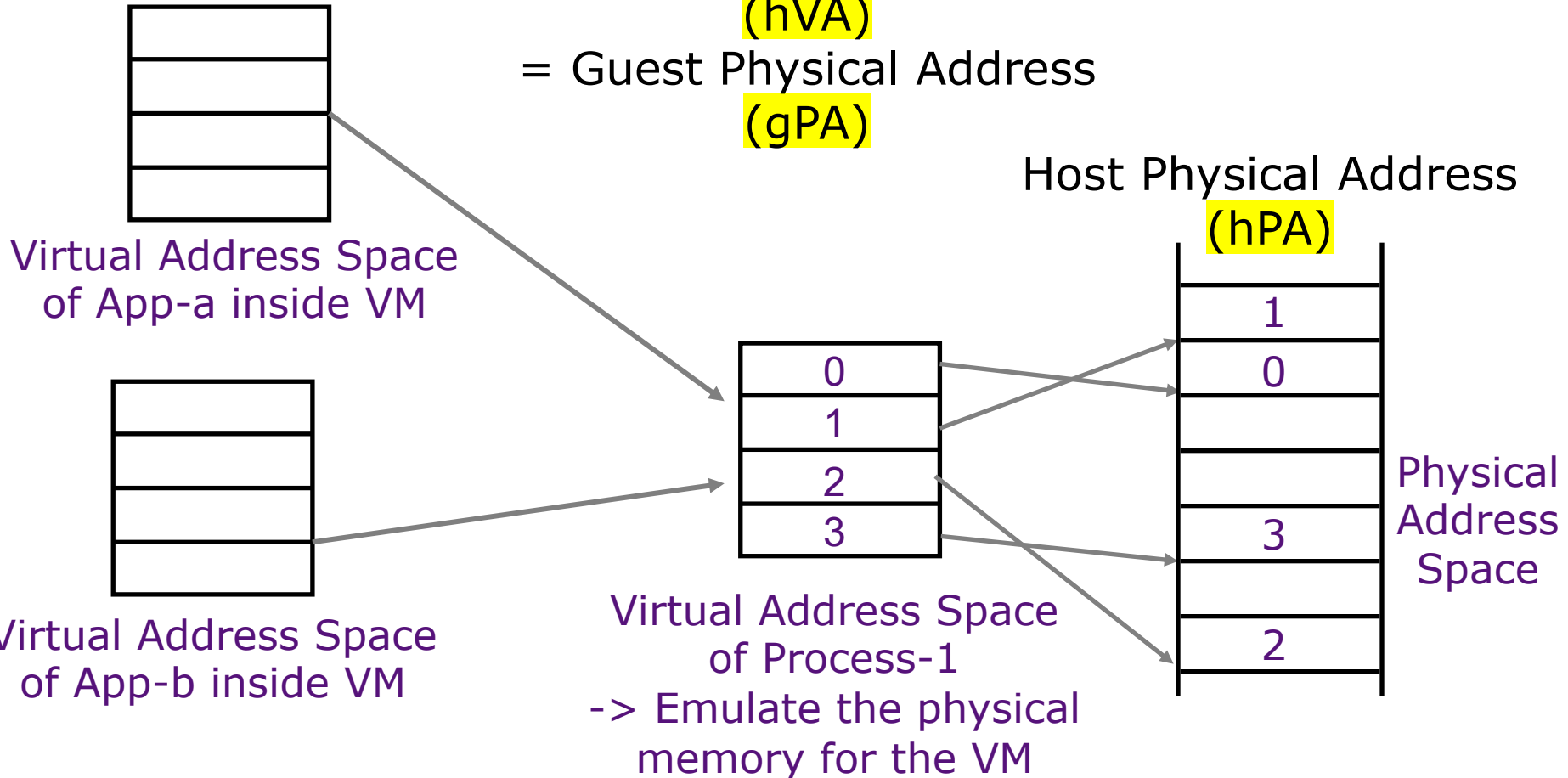
Virtual Memory in VMs

Guest Virtual Address
(gVA)

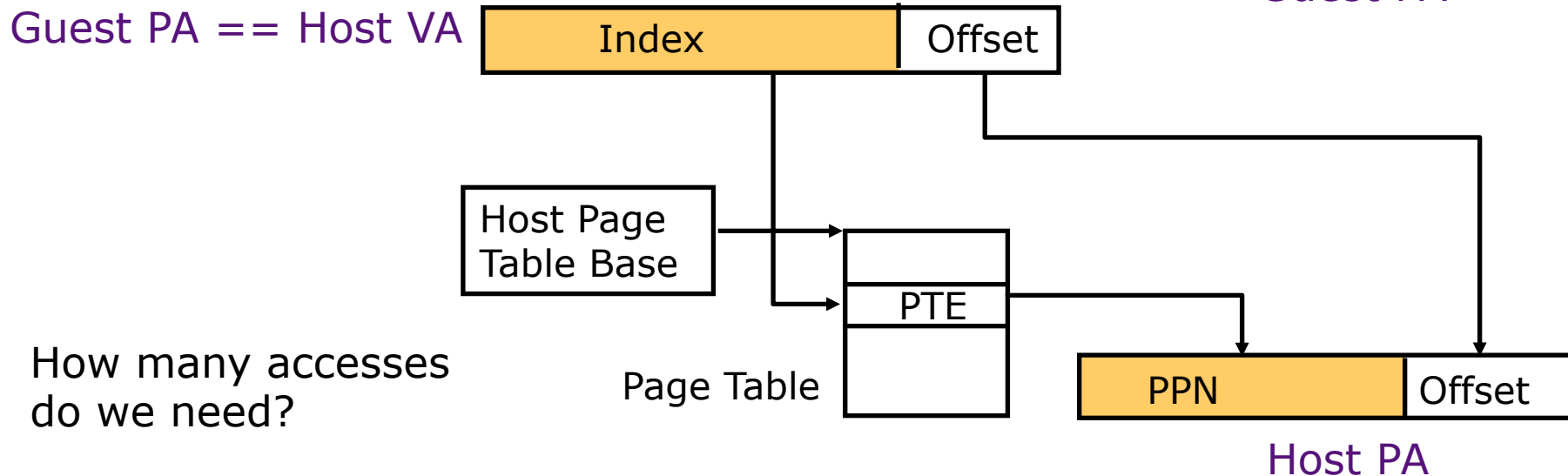
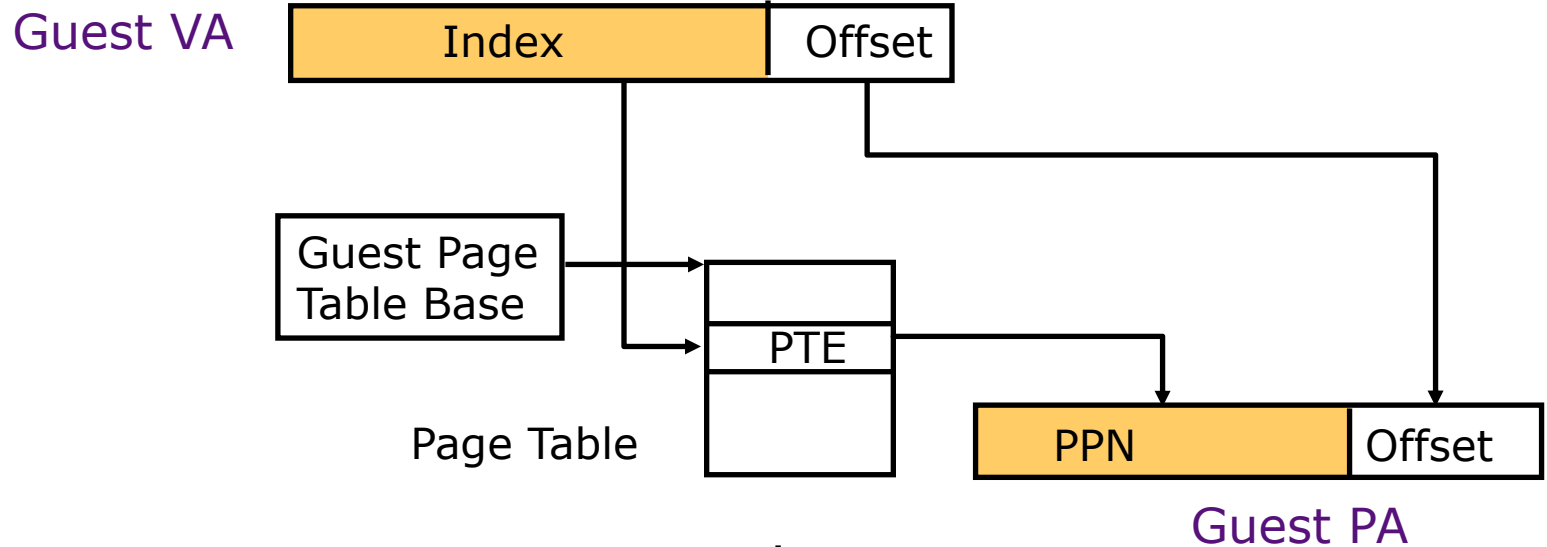
Host Virtual Address
(hVA)

= Guest Physical Address
(gPA)

Host Physical Address
(hPA)

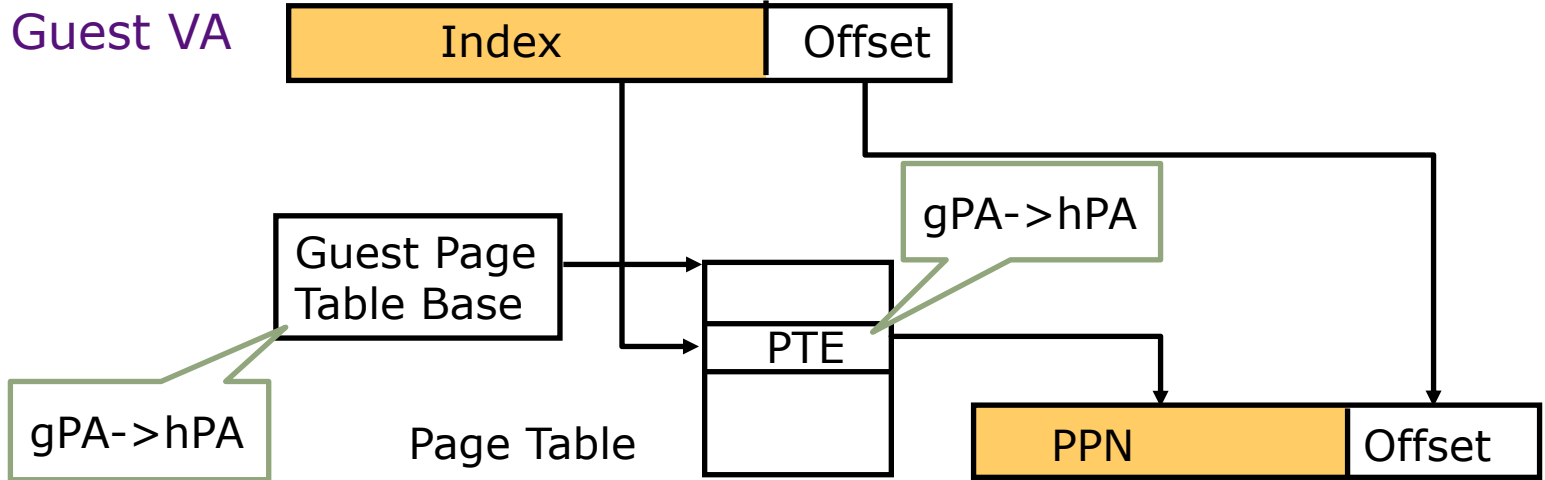


Nested Page Tables



How many accesses do we need?

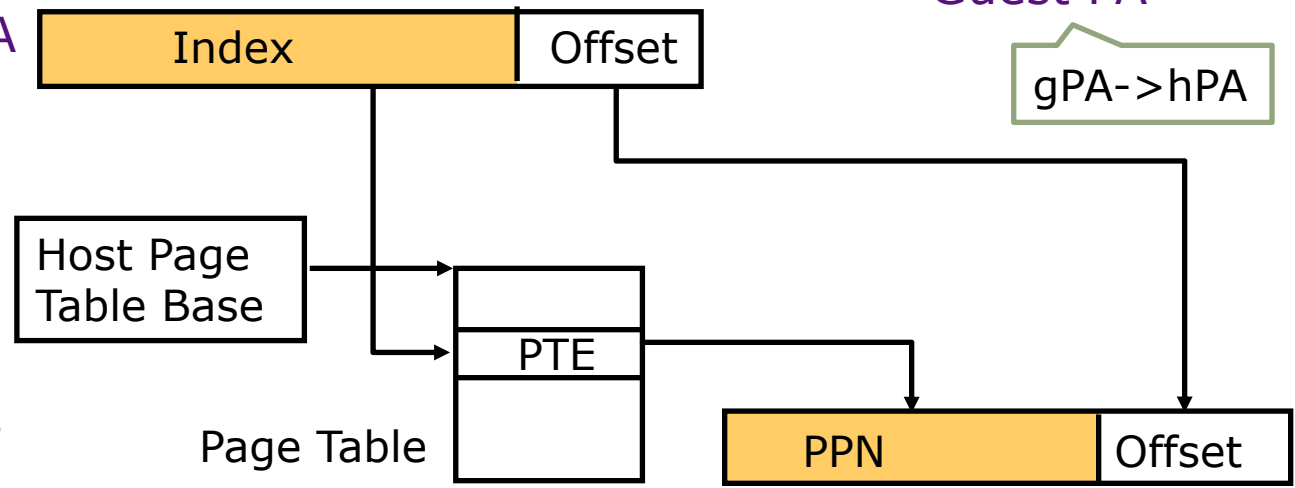
Nested Page Tables



Guest PA

gPA->hPA

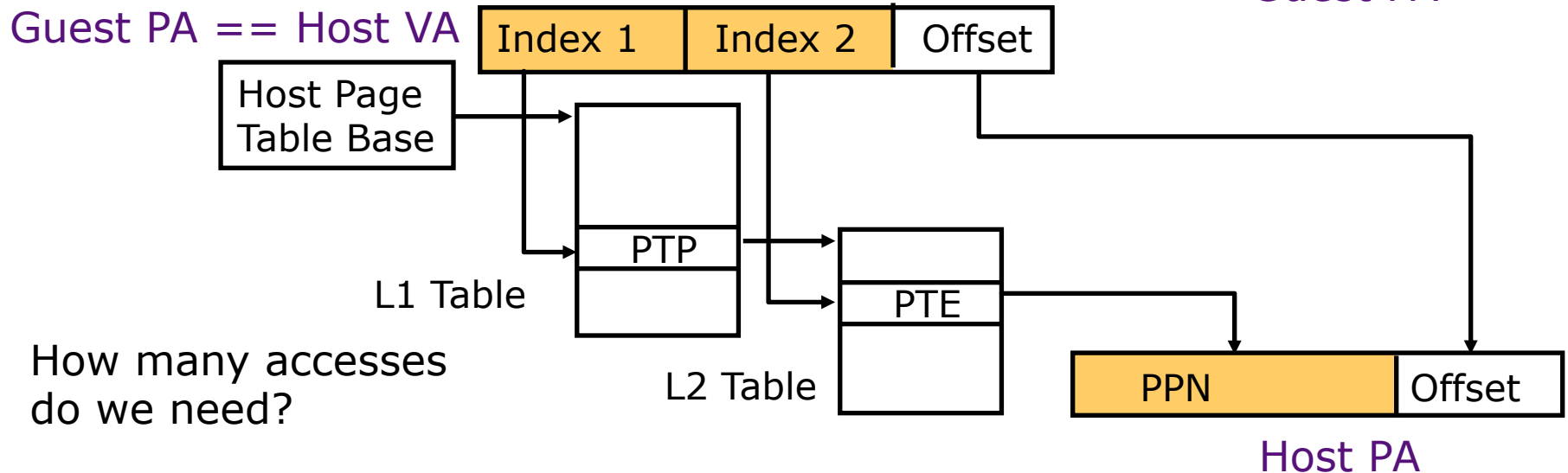
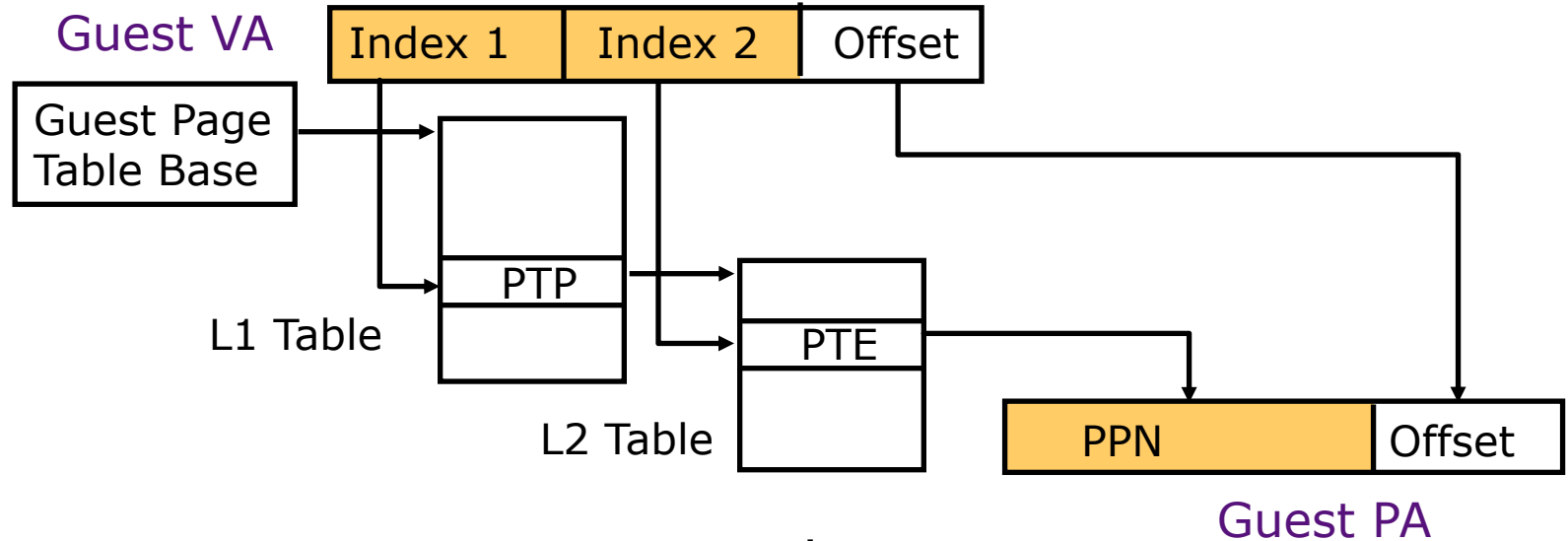
Guest PA == Host VA



Host PA

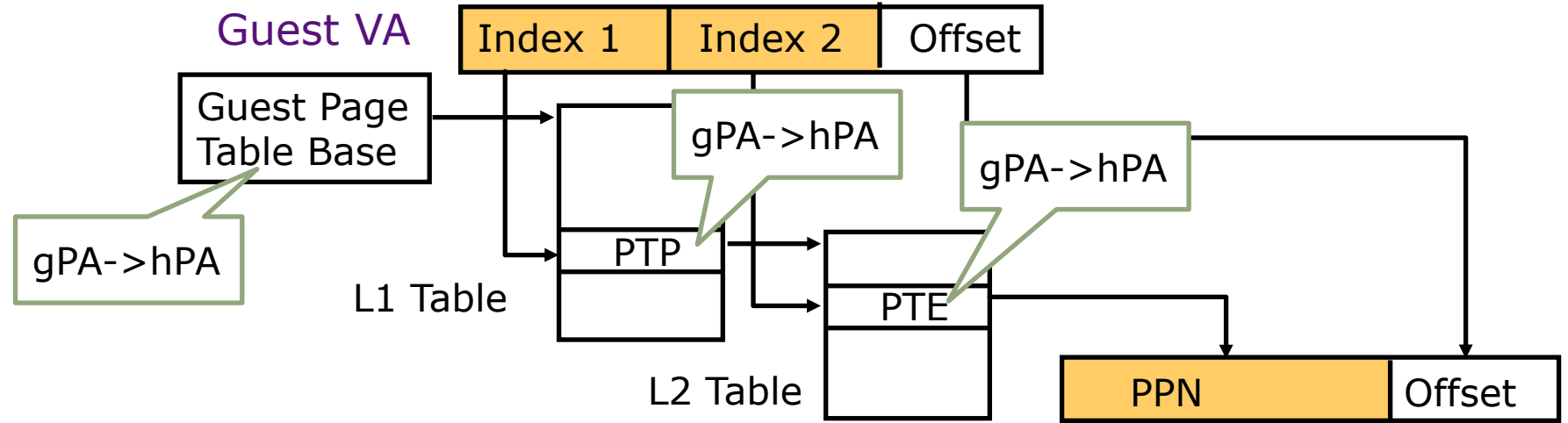
How many accesses do we need?

Nested Page Tables (Hierarchical)



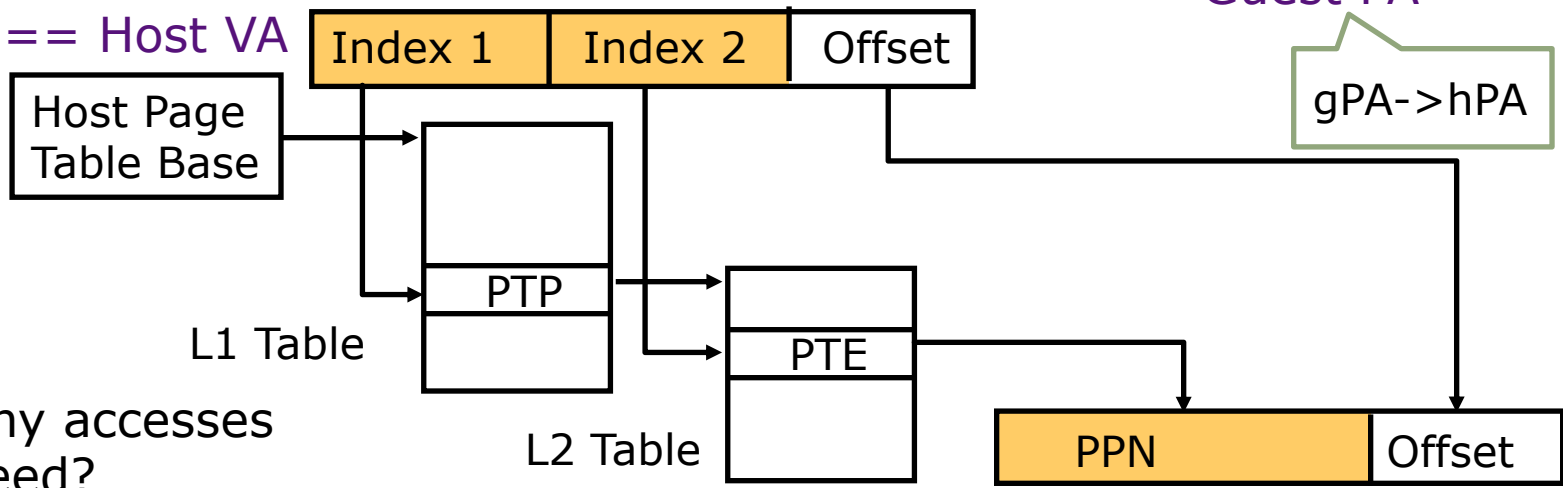
How many accesses do we need?

Nested Page Tables (Hierarchical)



Guest PA

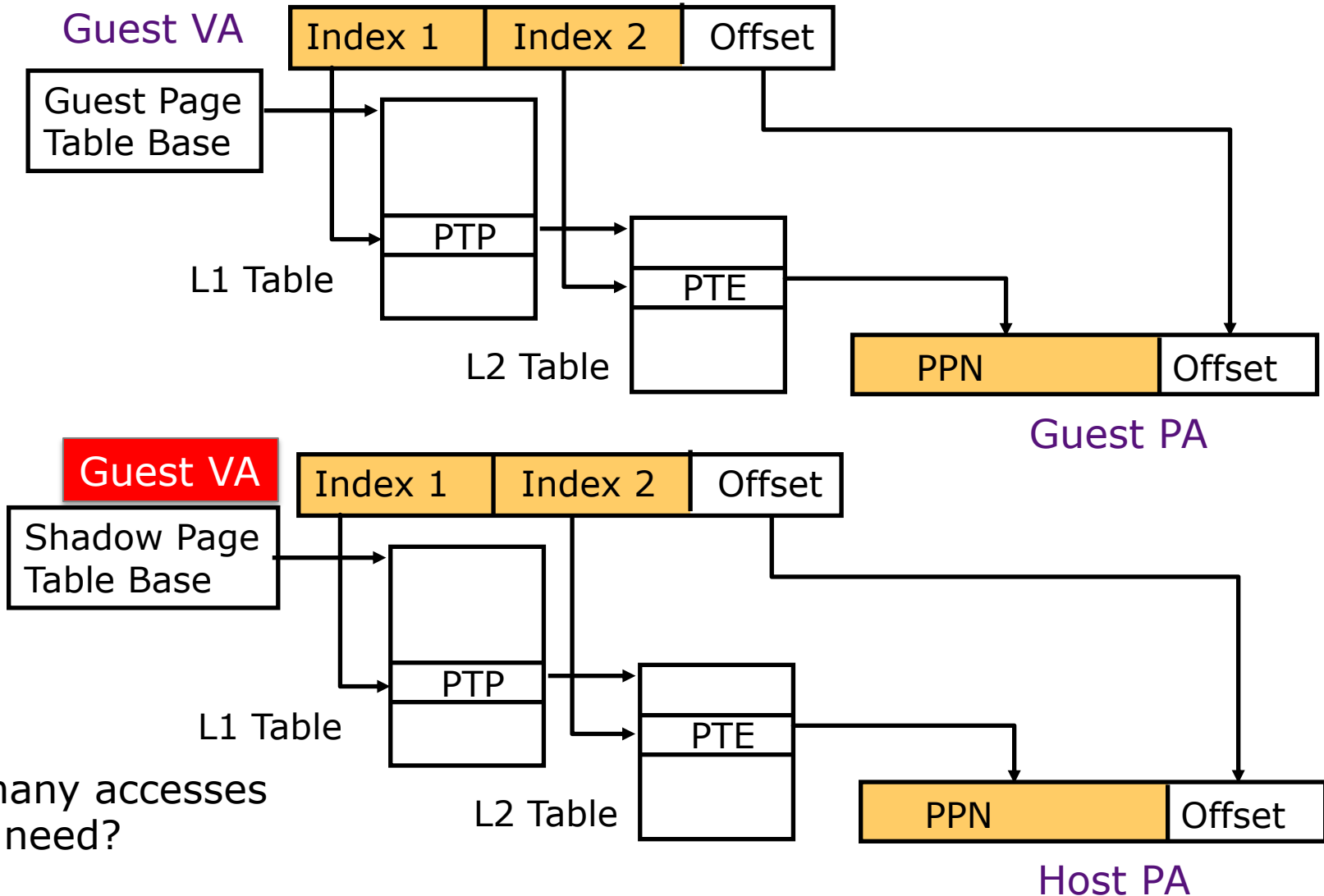
Guest PA == Host VA



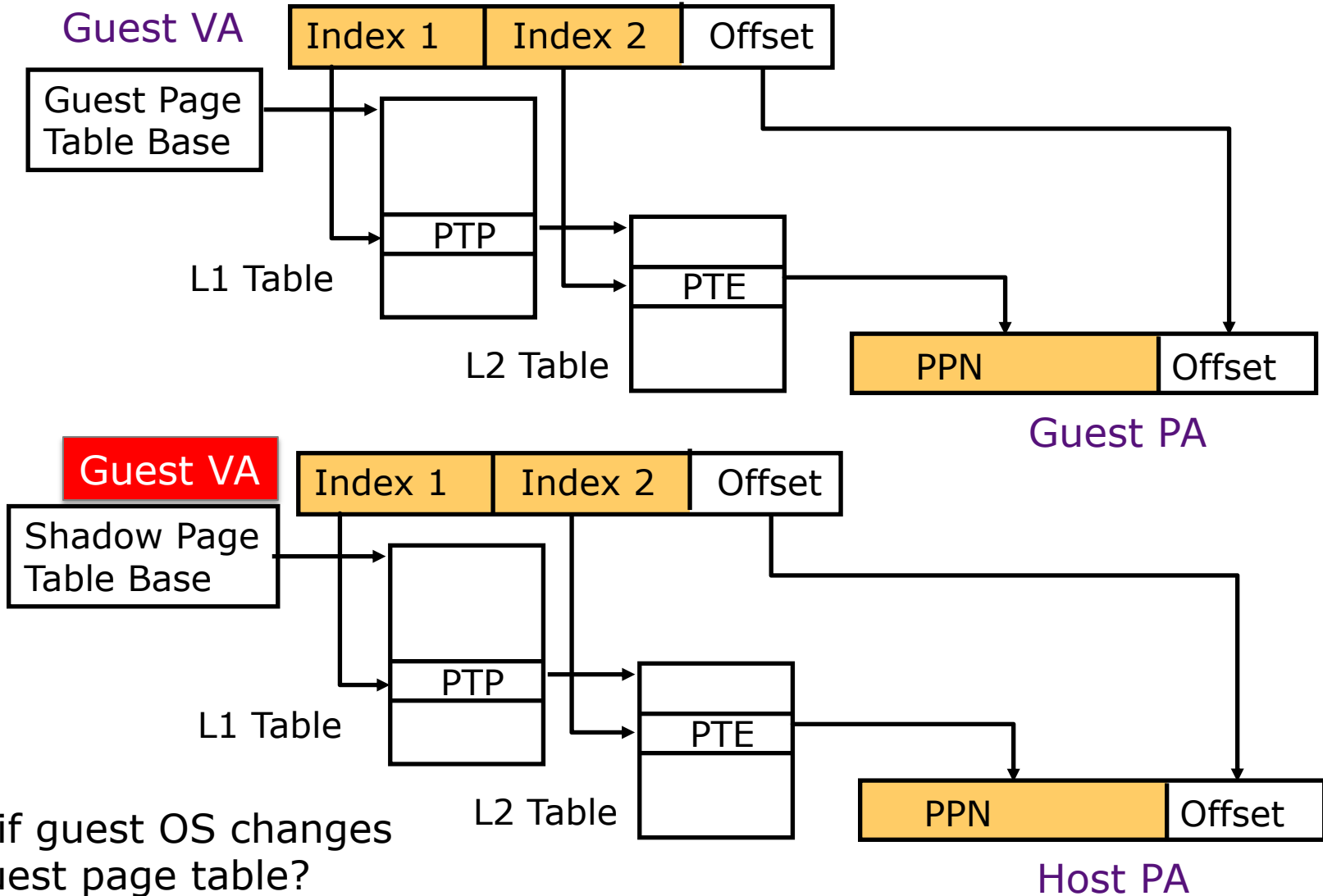
Host PA

How many accesses do we need?

Shadow Page Tables



Shadow Page Tables

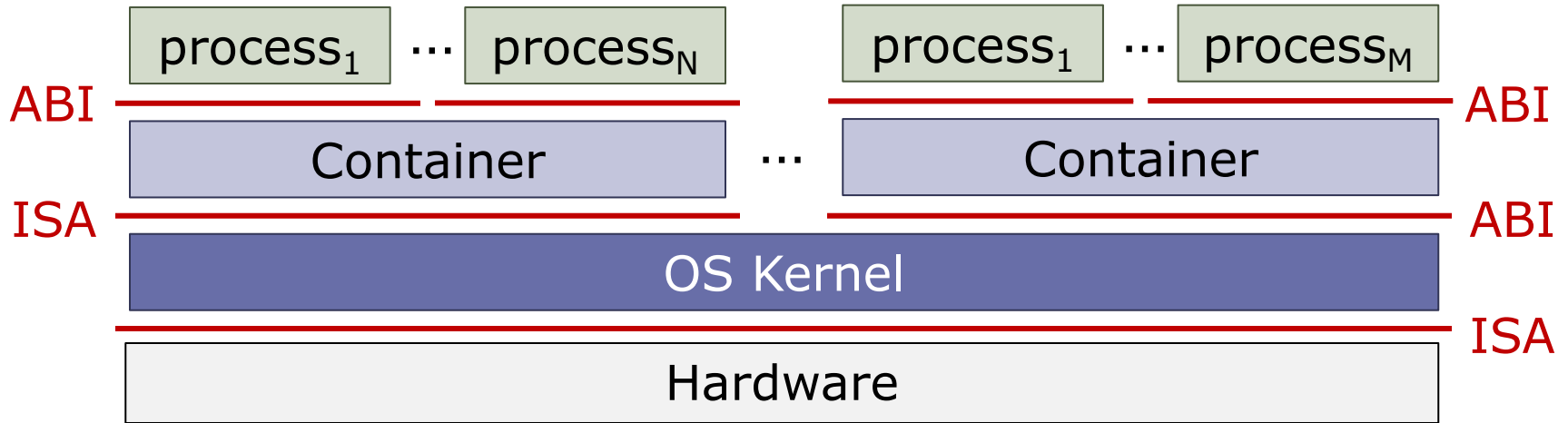


Nested vs Shadow Paging

	Native	Nested Paging	Shadow Paging
TLB Hit	VA->PA	gVA->hPA	gVA->hPA
TLB Miss (max)	4	24	4
PTE Updates	Fast	Fast	Uses VMM

On x86-64

Supporting Multiple Process Groups



- A “container” provides a **process group virtual machine** to each set of processes
- Container can run directly on OS, which provides a specific OS ABI to the processes in container

Container Semantics

- Isolation between containers is maintained by the OS, which supports a virtualized set of kernel calls.
 - Therefore, processes in all containers must target the same OS*
- Per Container Resources
 - Set of processes (each with a virtual memory space)
 - Set of filesystems
 - Set of network interfaces and ports
 - Selected devices

*Or closely related variants

Security and Side Channels

- Hardware isolation mechanisms like virtual memory guarantee that architectural state will not be directly exposed to other processes...and
- ISA and ABI are **timing-independent** interfaces
 - Specify *what* should happen, not *when*
- ...so non-architectural state and other implementation details and timing behaviors (e.g., microarchitectural state, power, etc.) may be used as **side channels** to leak information!

Coming Spring 2023 ...

- 6.S984: Datacenter Computing
- Instructor: Christina Delimitrou
- Short description:
 - Datacenter Computing explores the end-to-end stack of modern datacenters, from hardware and OS all the way to resource managers and programming frameworks.
 - The class will also explore cross-cutting issues, such as ML for systems, energy efficiency, availability, security, and reliability.
 - The main deliverable for the course is a semester-long research project on cloud computing, done in groups of 2-3 students. We will provide a list of suggested projects, but students are also encouraged to suggest their own.
- Lecture time: TR1-2:30

Thank you!