Introduction to Virtual Machines

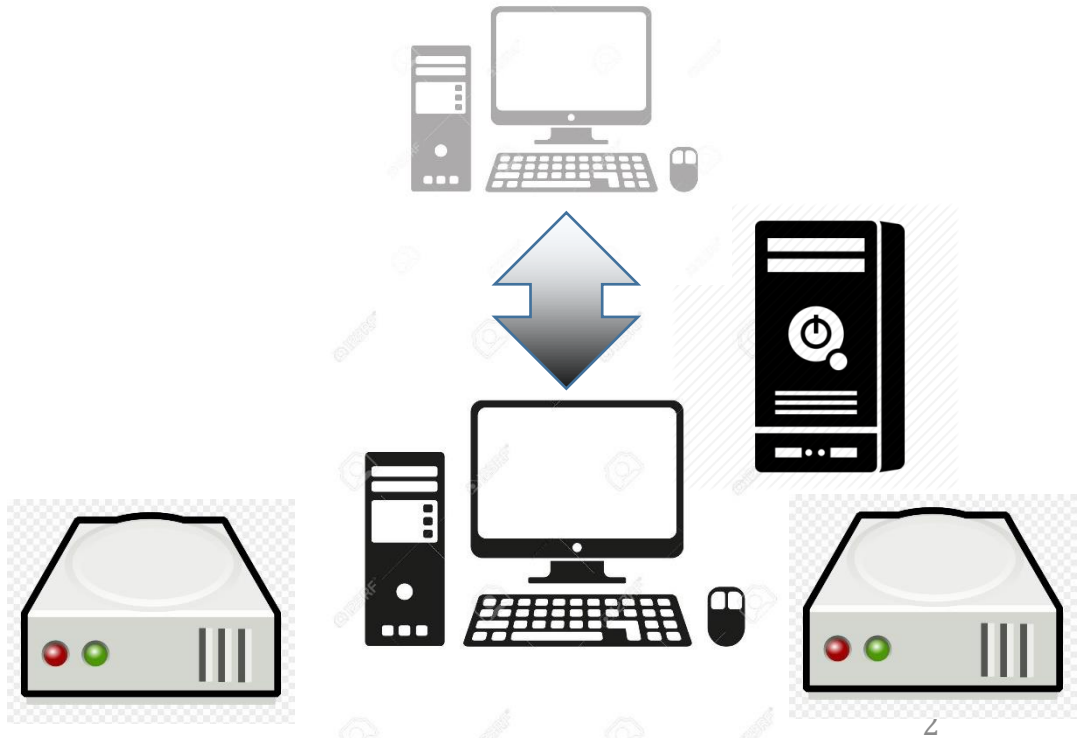
Modern Operating Systems, by Andrew Tanenbaum — Chap. 7

Virtual Machines, by Smith and Nair — Chap. 1&6

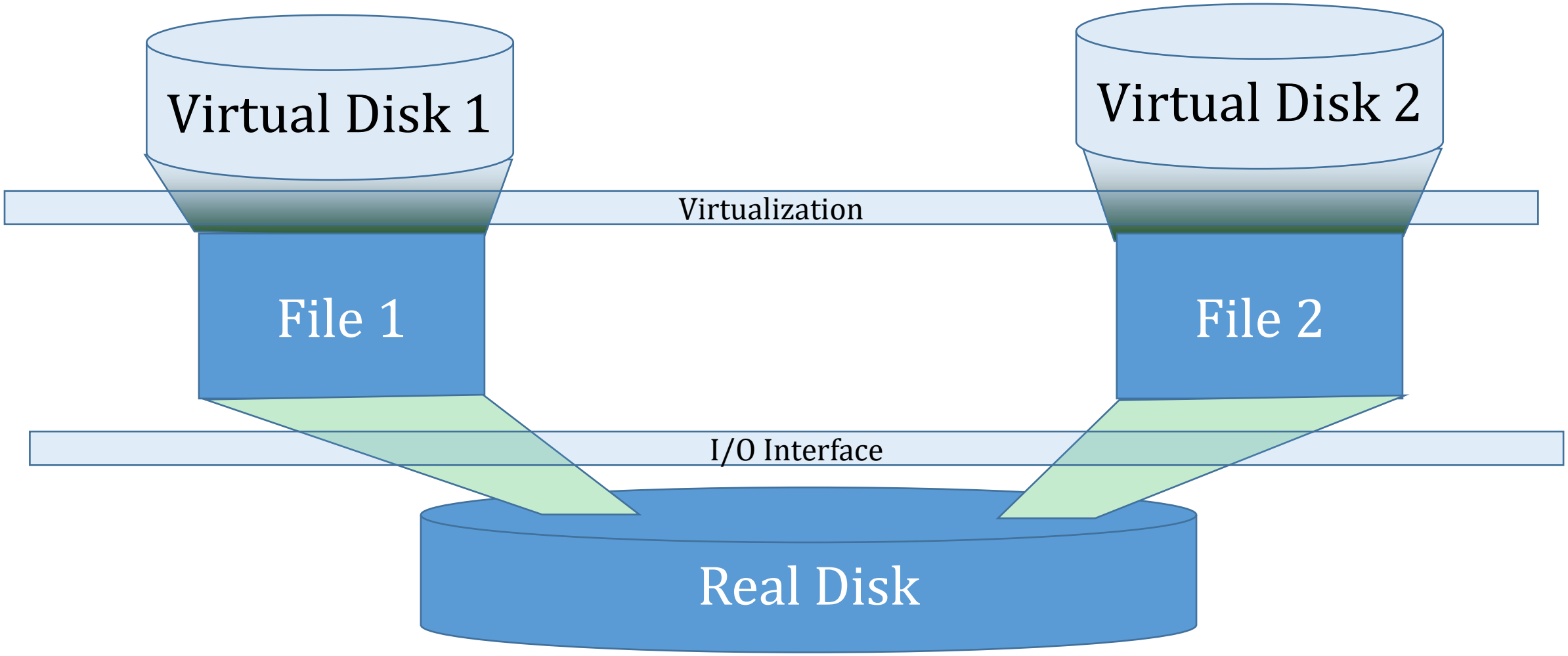
Virtualization

One to Many

Many to One



Example: Disk Virtualization

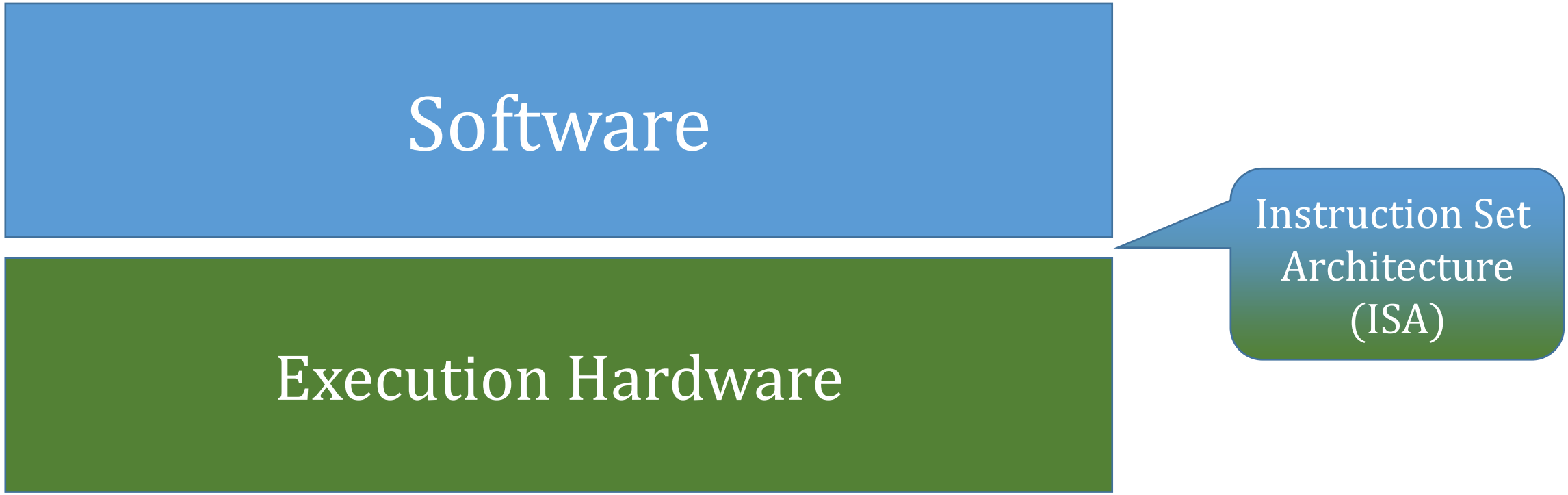


Virtual Machines

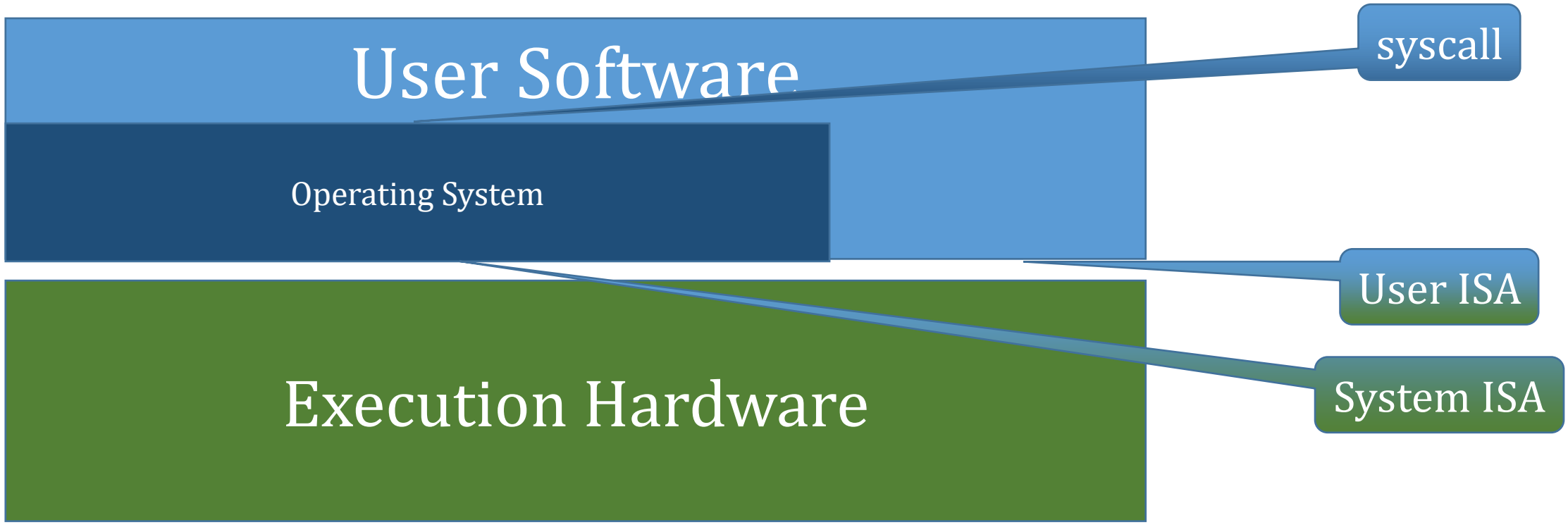
- Logical / Emulated representations of full computing environment
  - CPU + Memory + I/O
  - Implemented by adding virtualization layers of software

- Uses:
  - Multiple OS's on one machine (including legacy OS's)
  - Isolation (e.g. crash protection) & Enhanced security
  - Live migration of servers
  - Testing and Development
  - On-the-fly optimization
  - Platform Emulation including realization of ISA's not found in physical machines

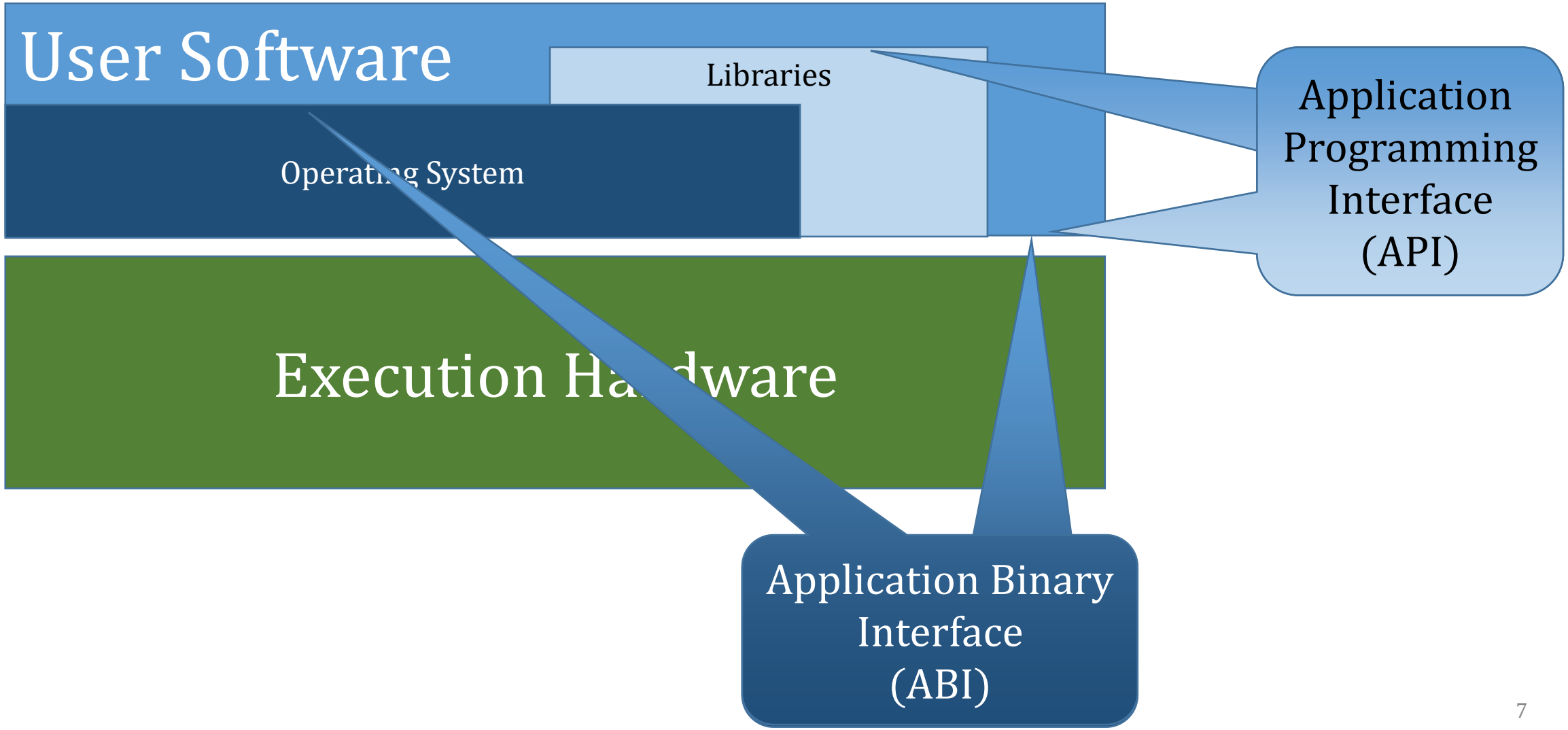
Interfaces: Instruction Set Architecture



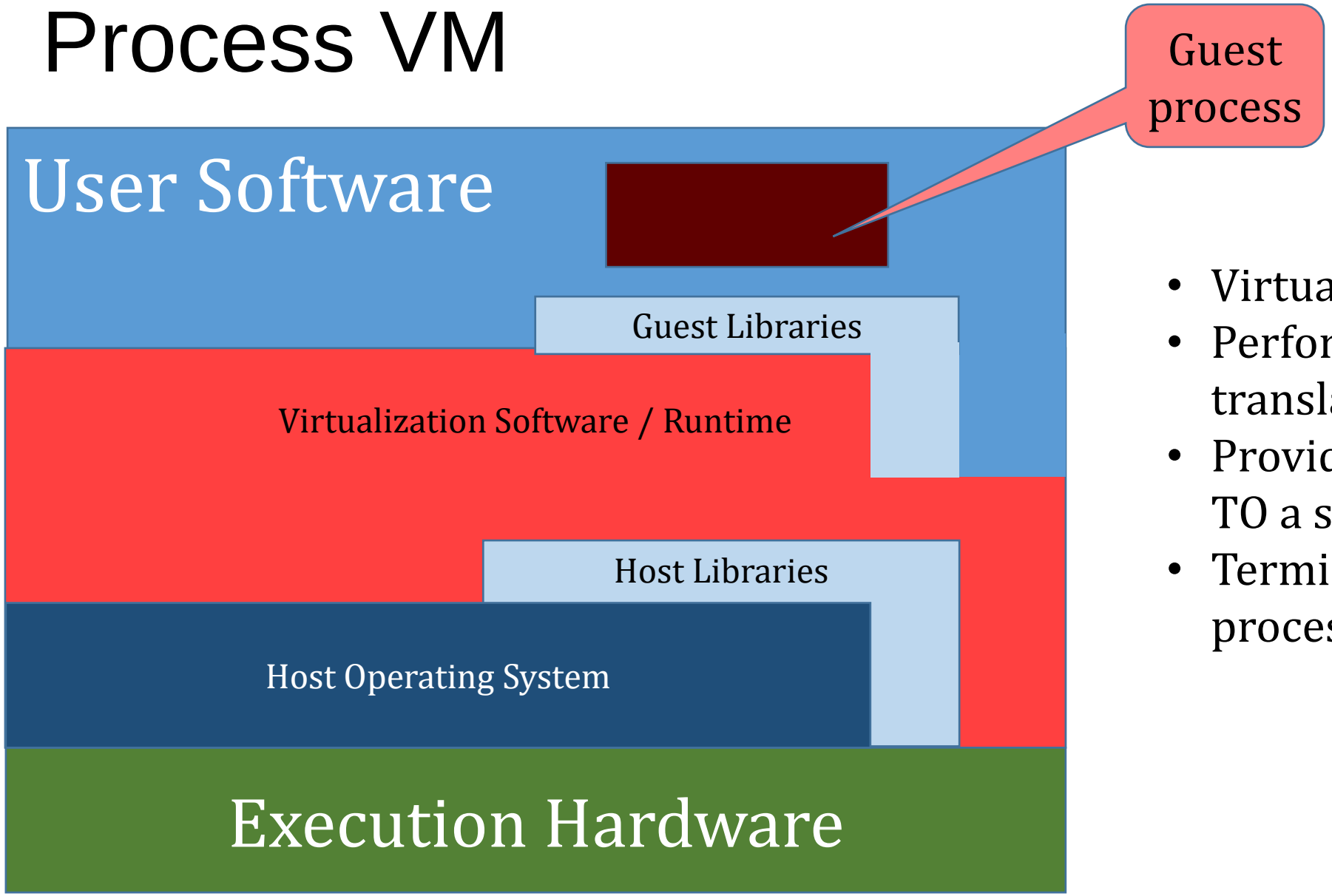
Interfaces: Operating System



Interfaces: Libraries

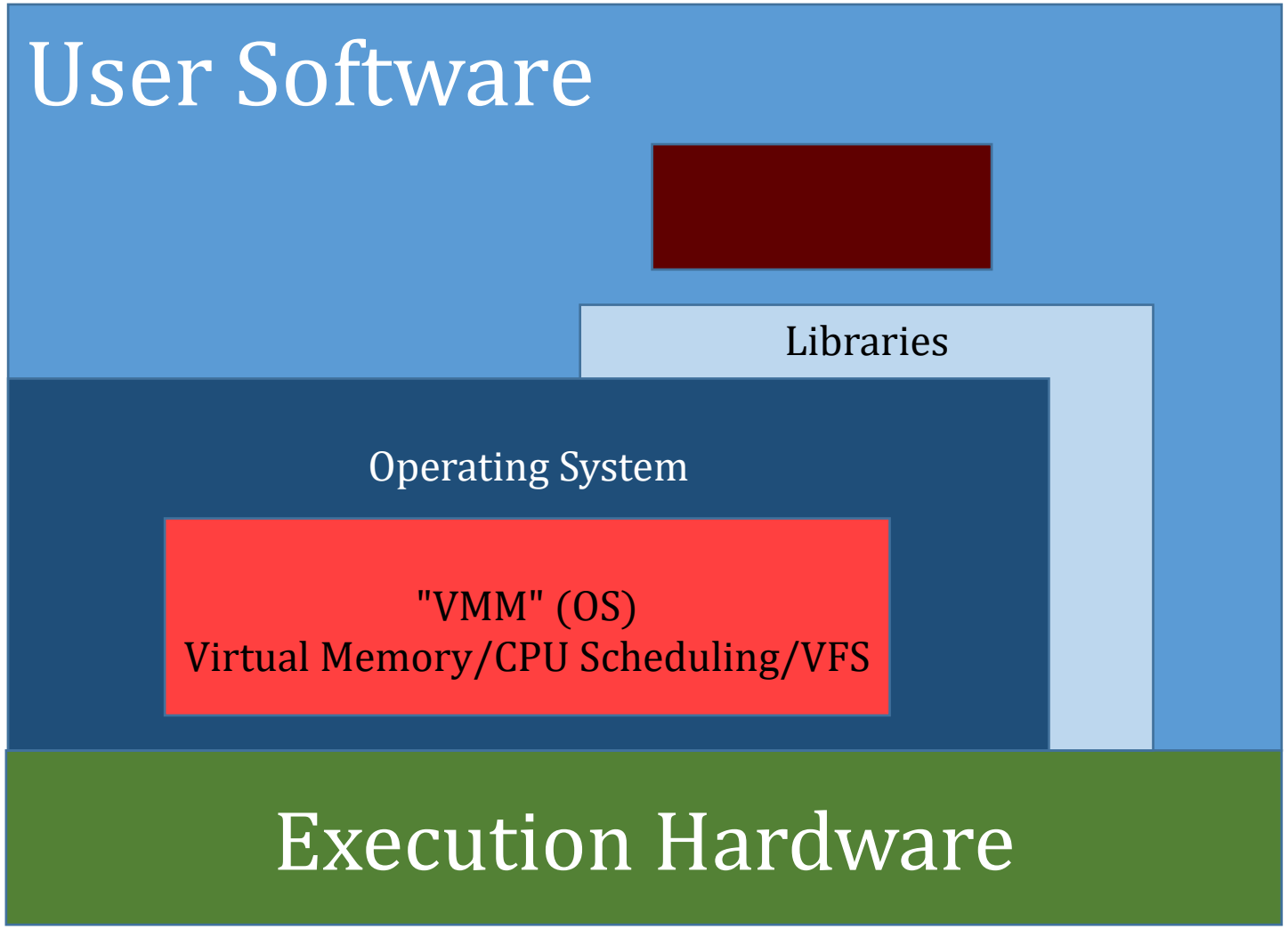


Process VM



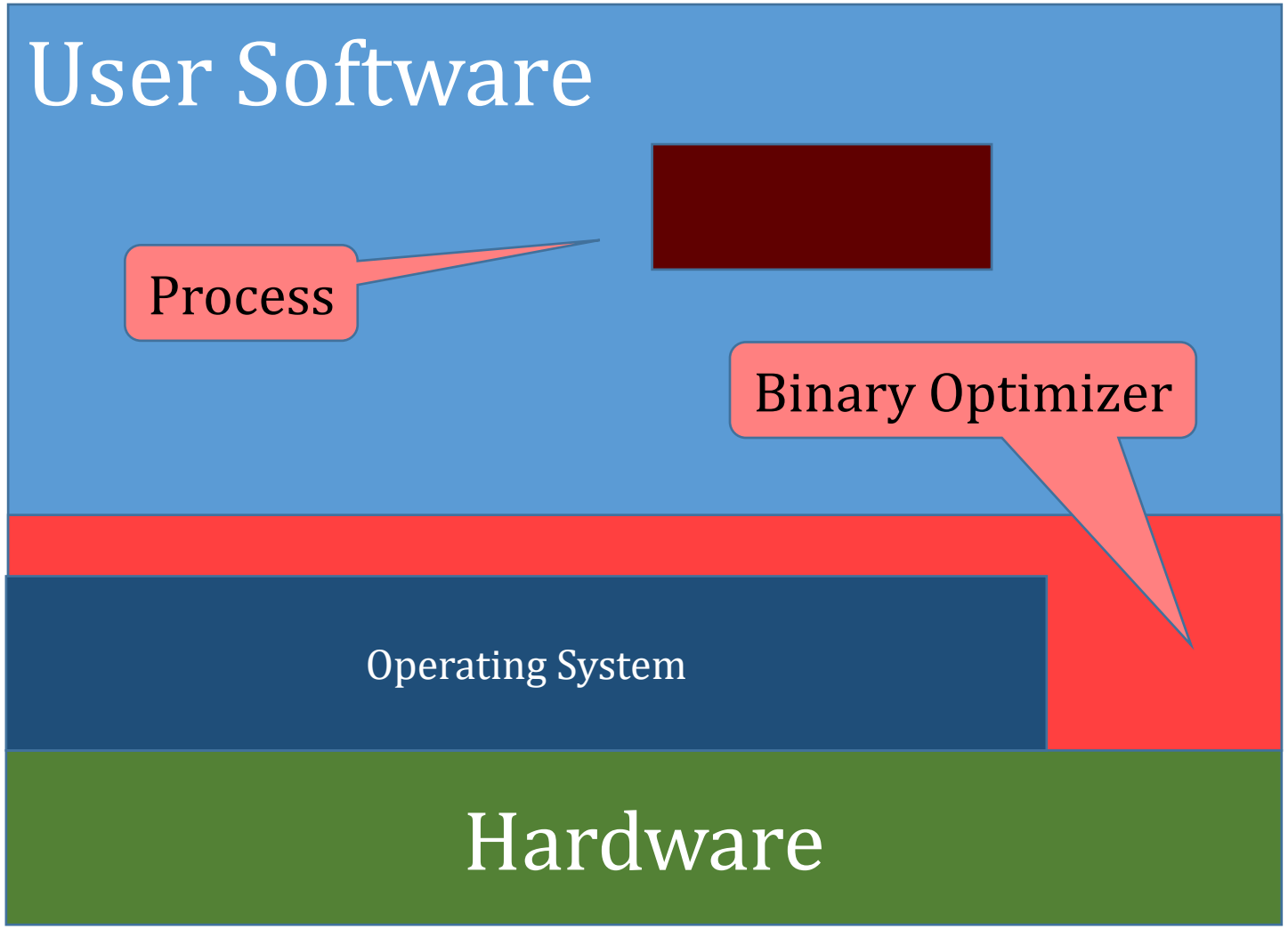
- Virtualizes the ABI
- Performs binary translation
- Provides a virtual view TO a single process
- Terminates when guest process terminates

Process VM: Multi-programming



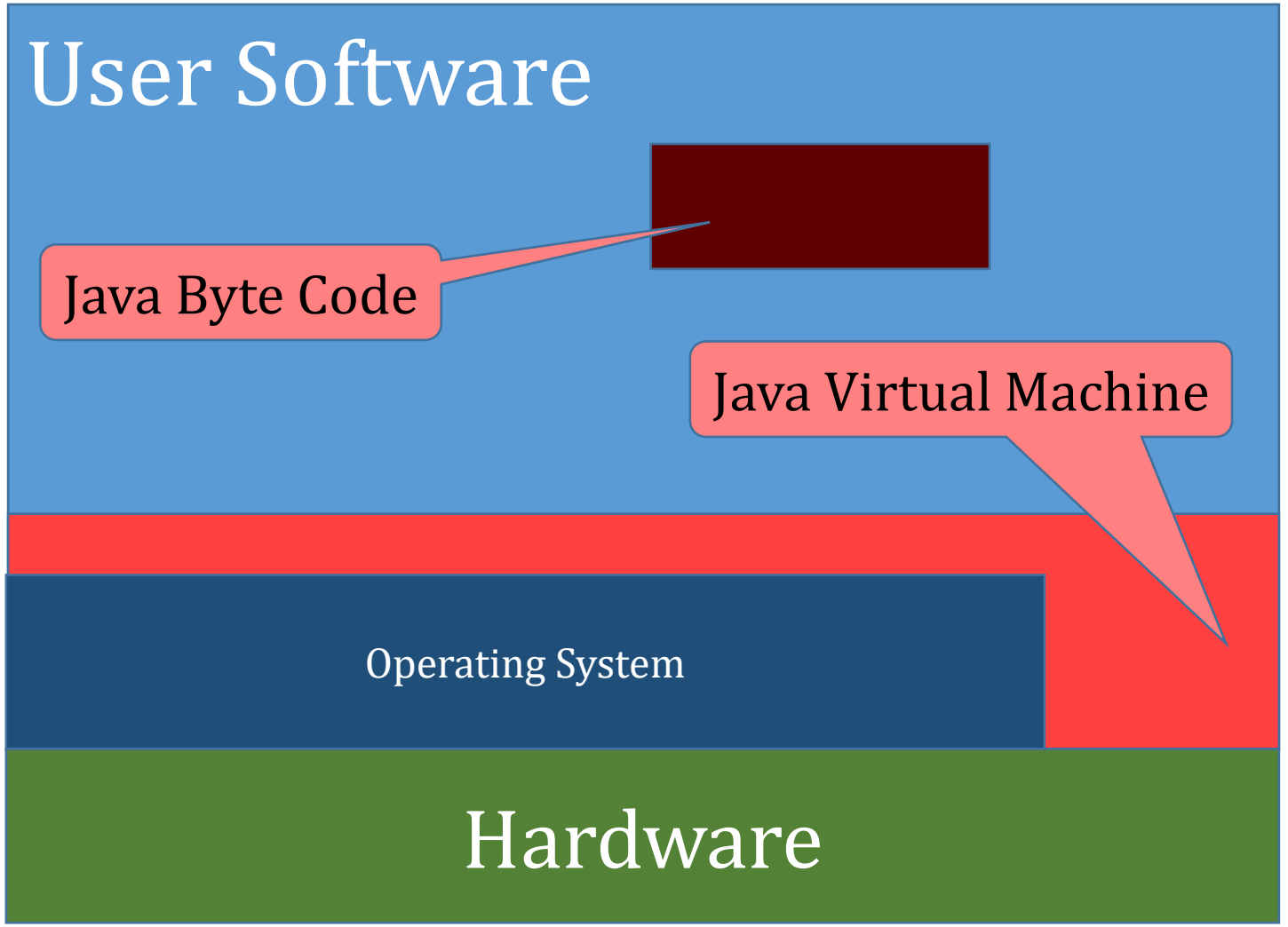
- Each Process operates on a virtual "runtime" provided by the Operating System
  - virtual memory
  - virtual cpu
  - virtual file system

Process VM : Binary Optimizers



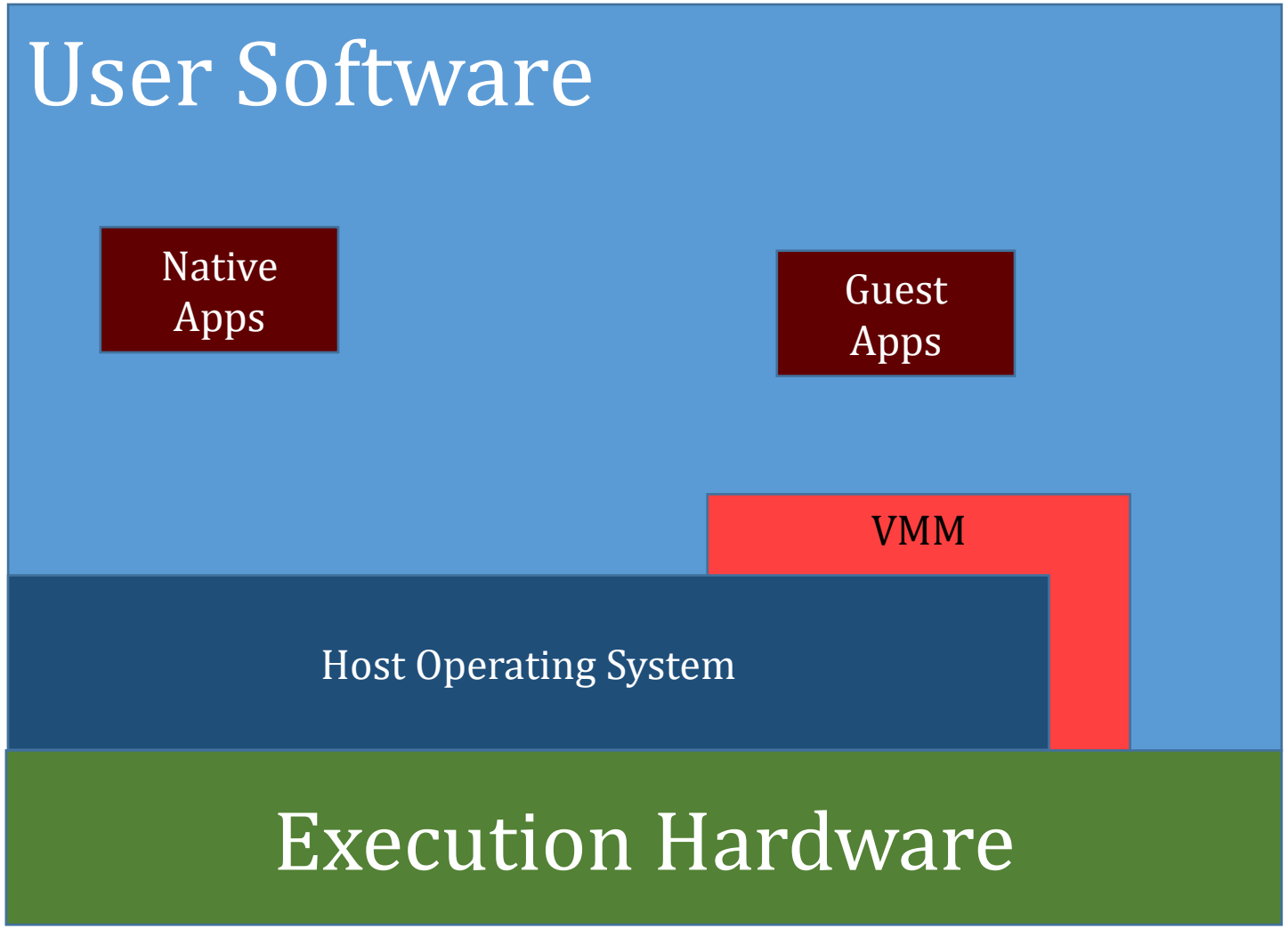
- Debug like VALGRIND, GDB
- Fast MALLOC/FREE
- …

Process VM: High Level Language



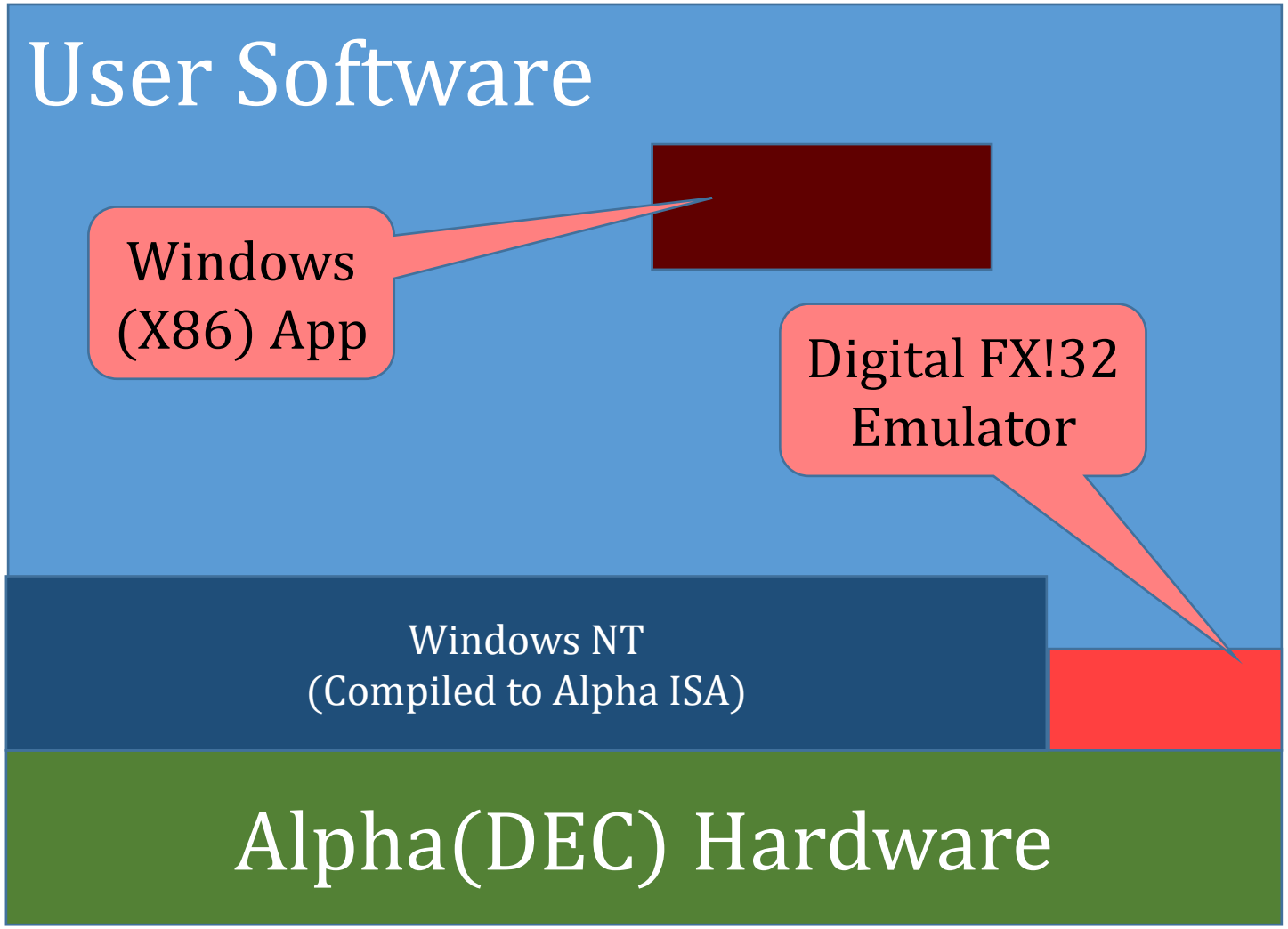
- Virtual ISA
  - e.g. Java Byte Code
  - For platform independence
- Platform Dependent VM executes virtual ISA
- e.g. Java JVM or .NET CLI

Process VM: Emulators



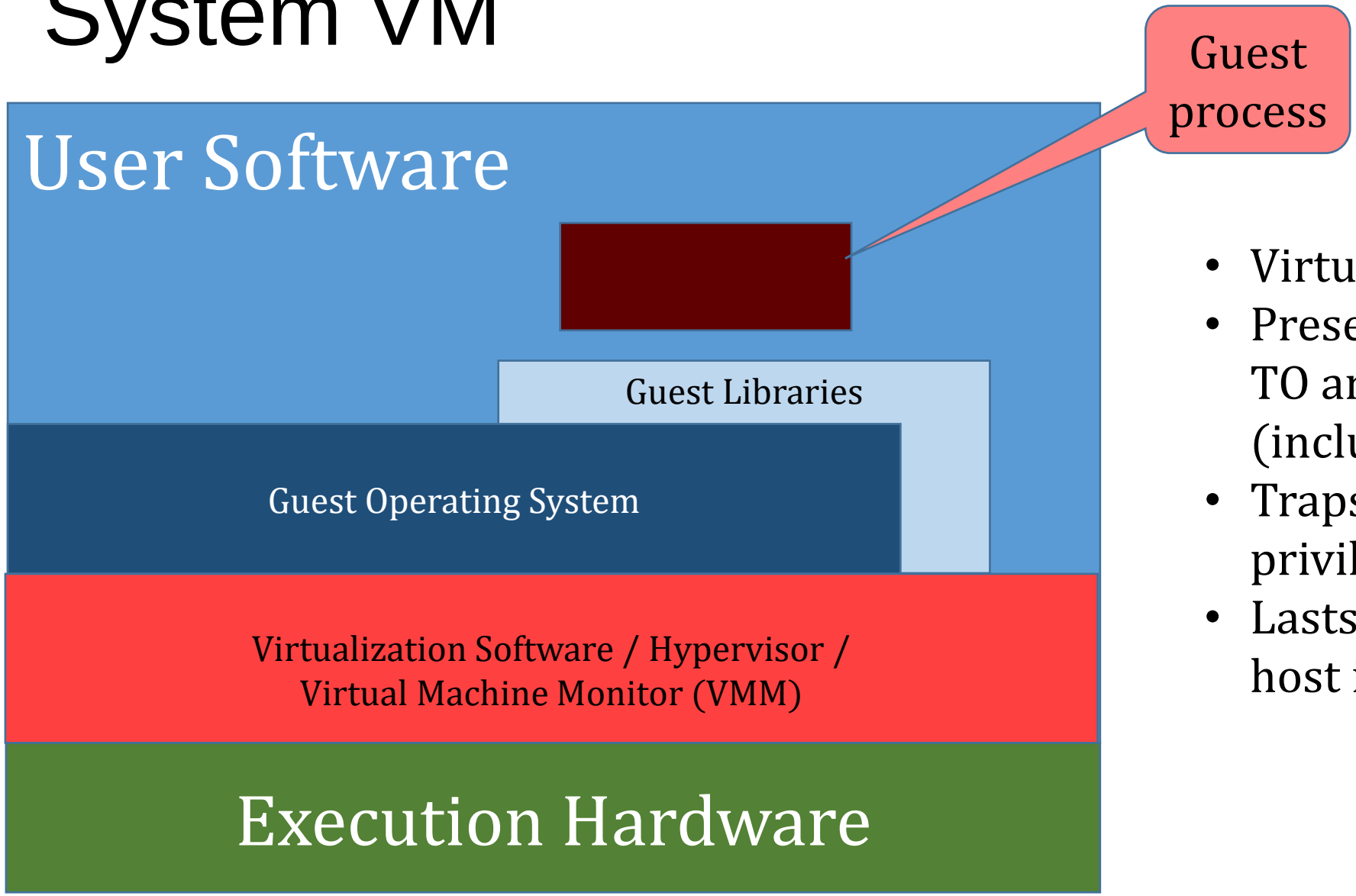
- Host & Guest ABI are different – emulation required
- Hosted VM + emulation
- E.g. Cygwin, Virtual PC (Windows on MAC)

Process VM: Emulator w/ Guest OS



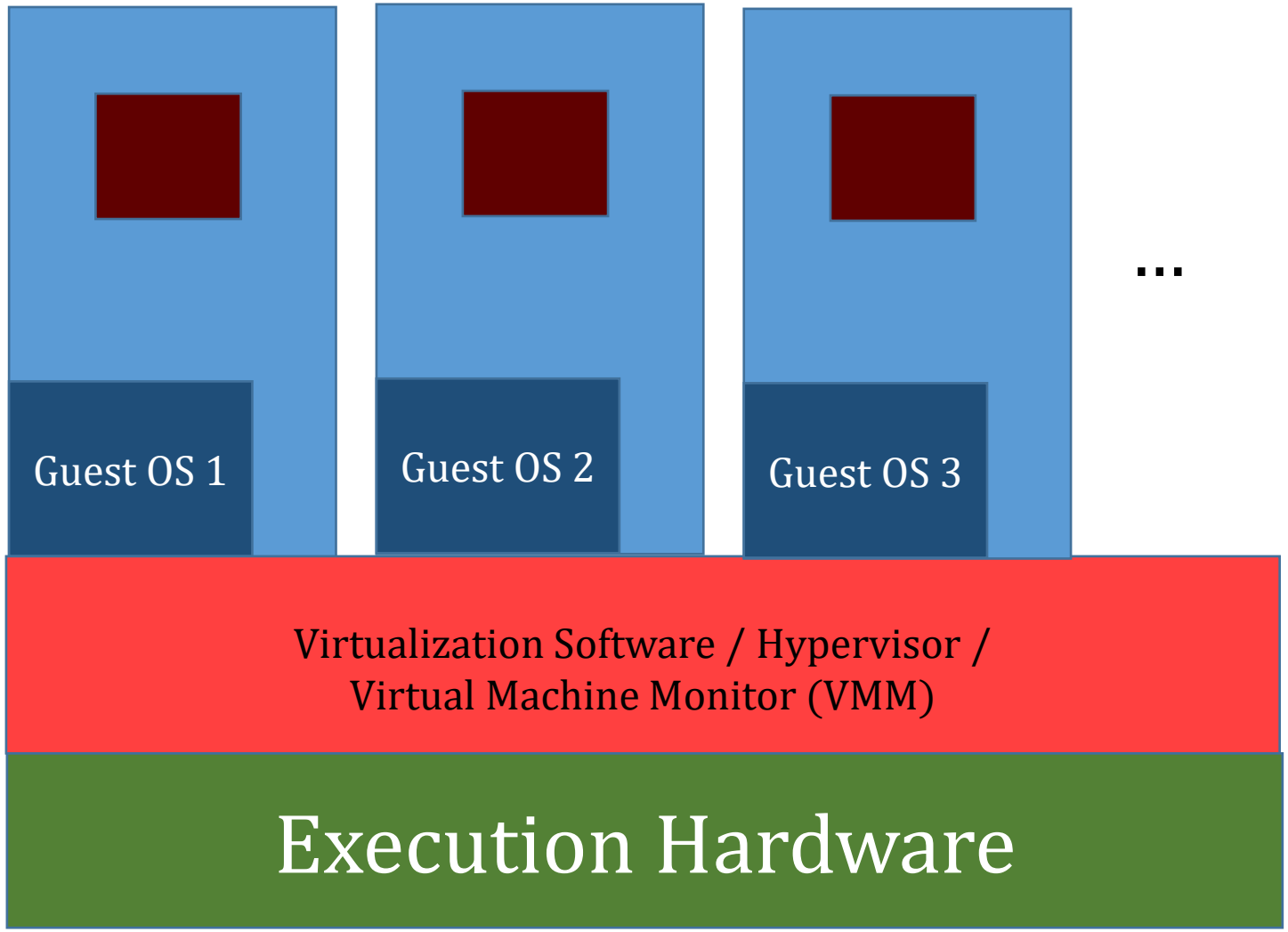
- Support an app ABI on hardware running a different target ABI
- Interpreter
  - Fetch, decode, convert
  - SLOW
- Dynamic Binary Translator
  - Blocks of code translated
  - Cached to exploit locality

System VM



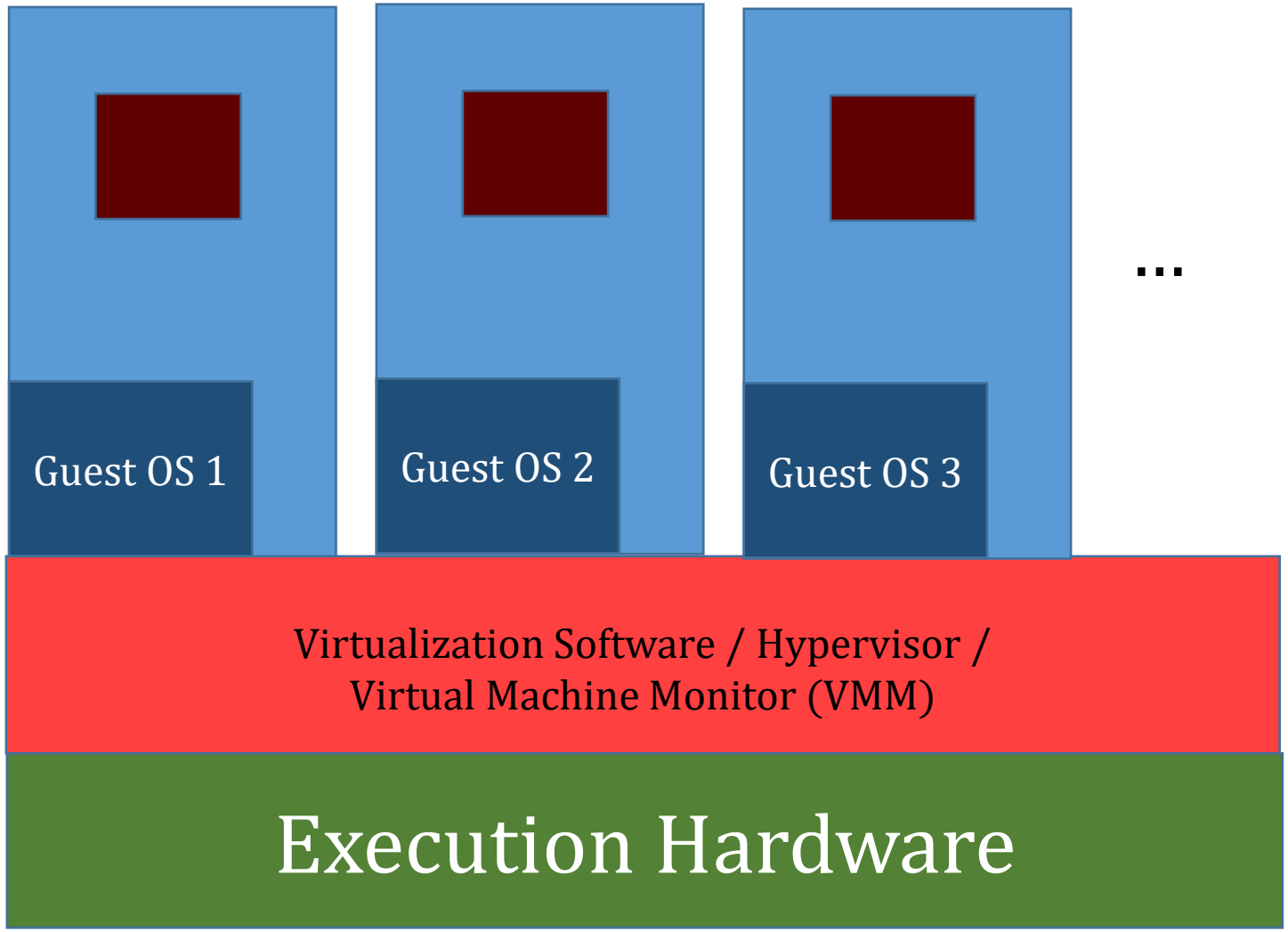
- Virtualizes the ISA
- Presents a Virtual View TO an entire system (including OS)
- Traps and emulates privileged instructions
- Lasts as long as physical host is alive

System VM: Generic Hypervisor



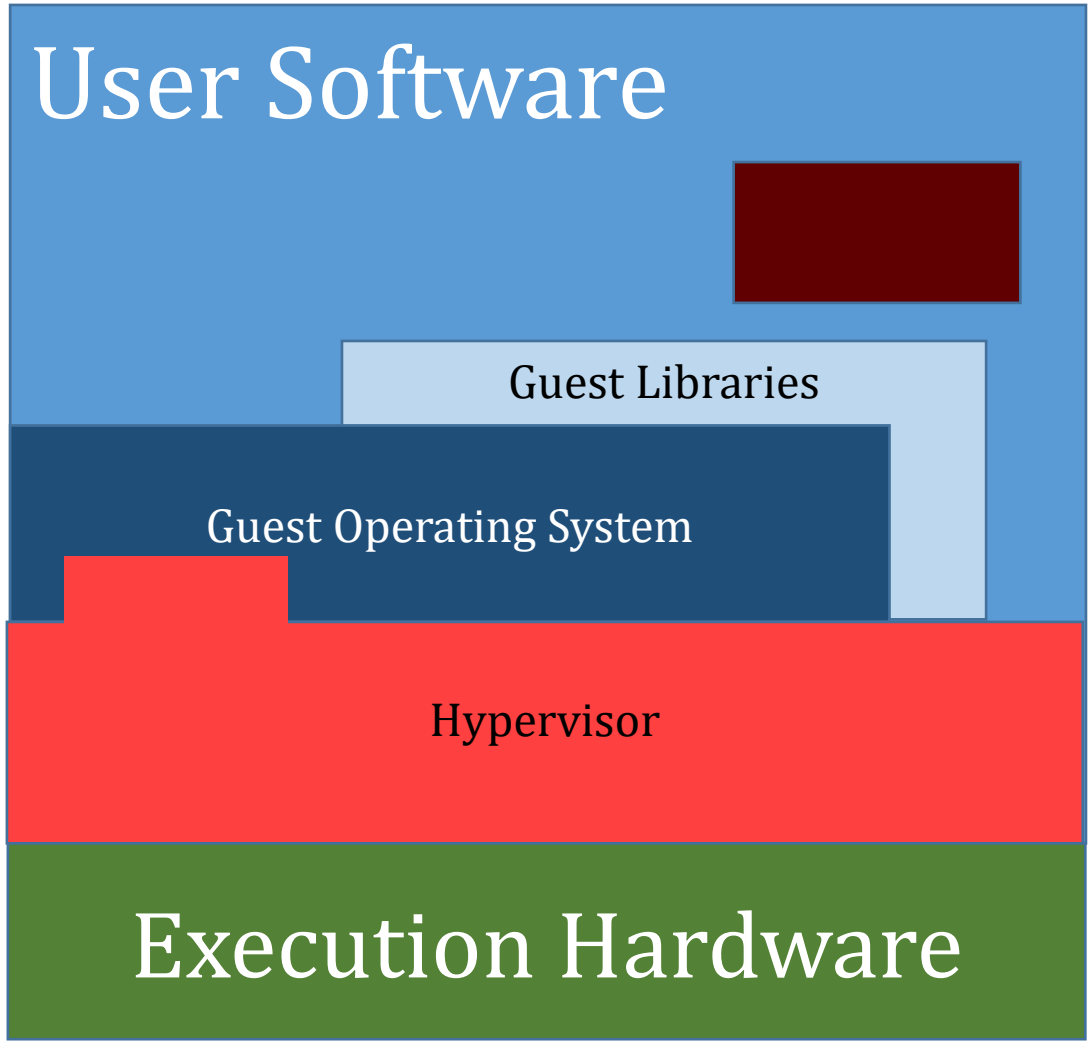
- Operating system for operating systems
- Virtual execution environment for an entire OS and apps
- Controls access to hardware/resources
- When guest OS executes a privileged instruction, Hypervisor intercepts the instruction, checks for correctness, and emulates the instruction

System VM: Traditional Hypervisor



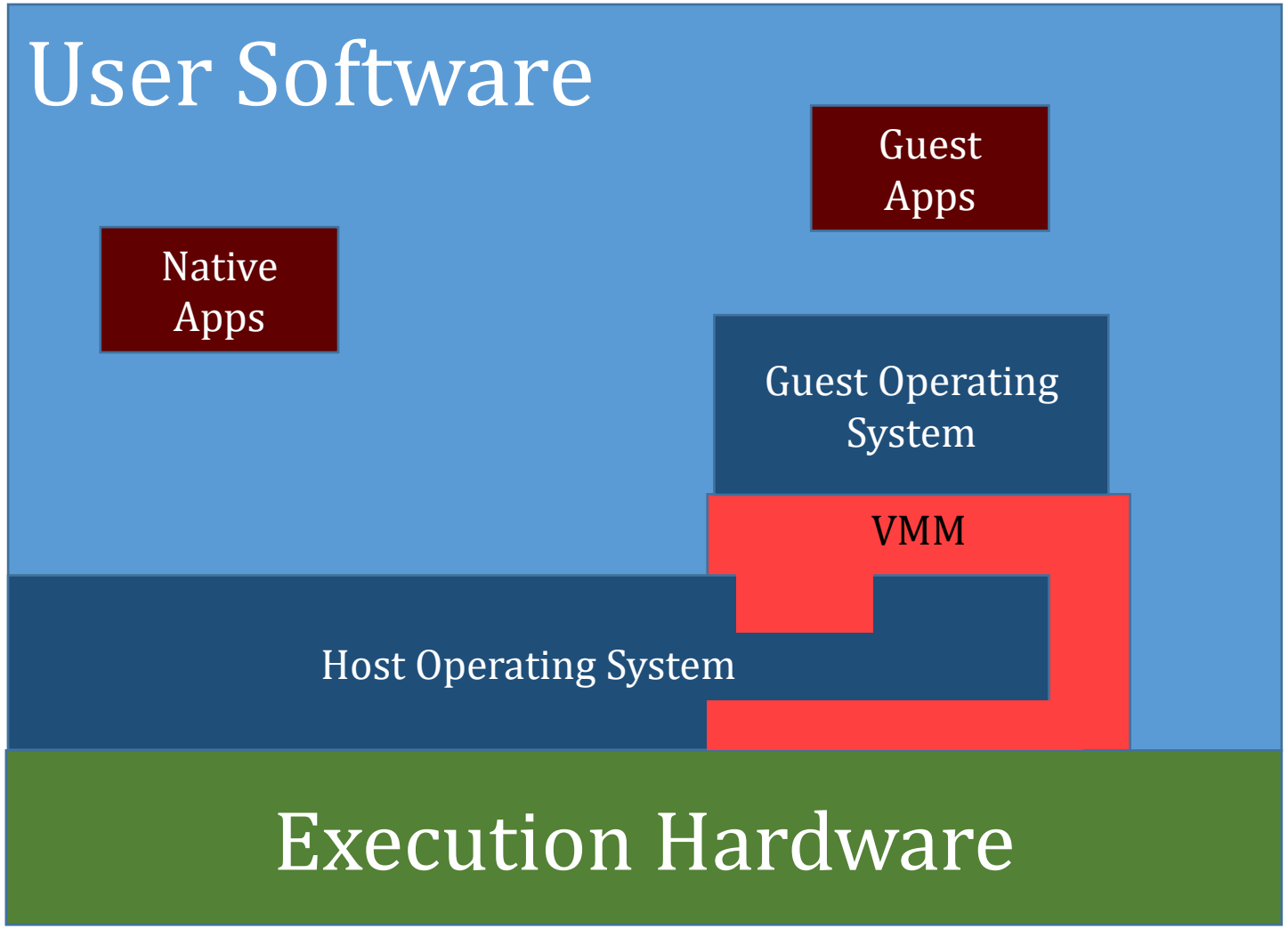
- executes natively on host ISA
- Directly controls hardware – provides all device drivers
- Emulates sensitive instructions executed by the Guest OS
- E.g. KVM, VMWare ESX Server

System VM: Para-Virtualized VM



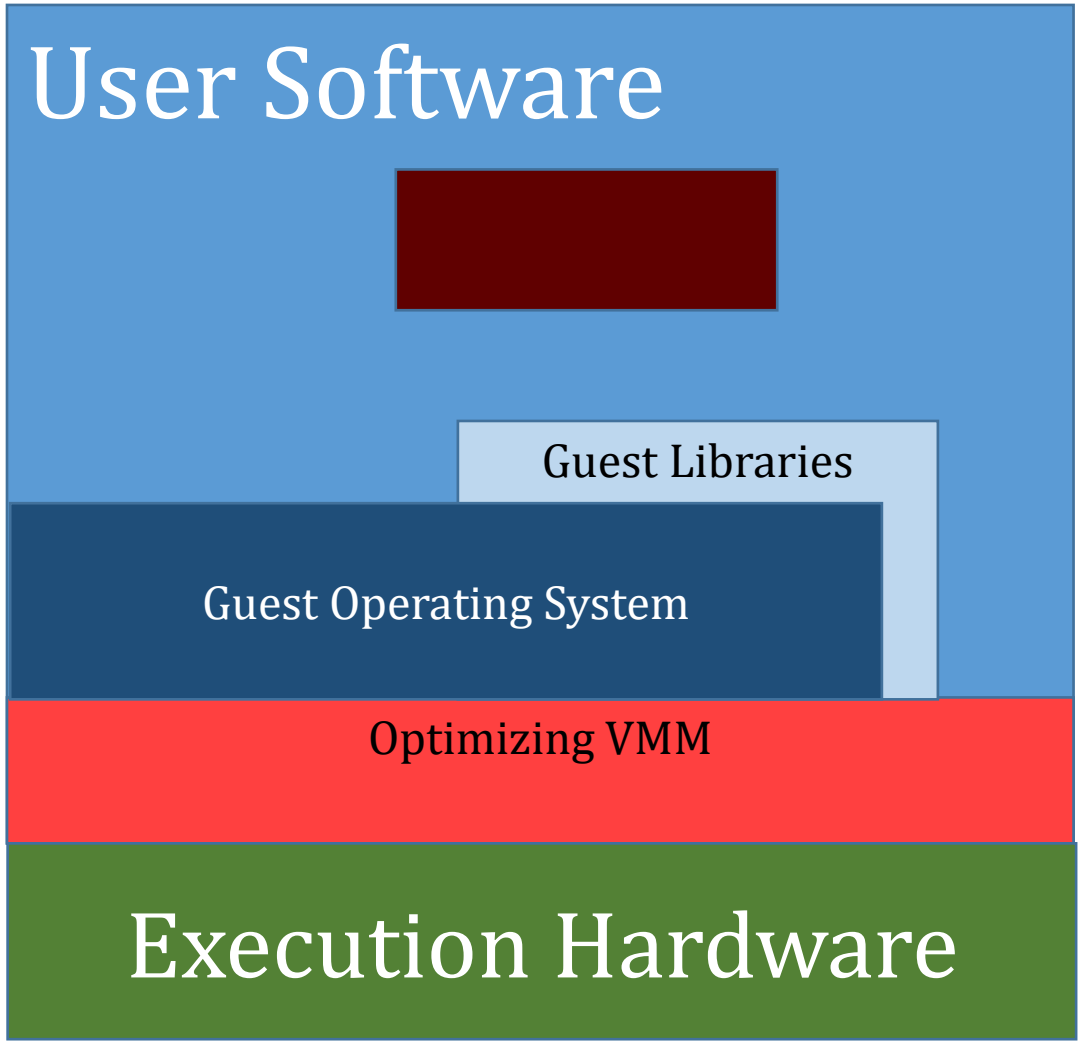
- Modify guest OS for better performance
  - Trap and emulate is expensive
- Virtual hardware is **similar**, but **not identical** to real hardware
- Guest OS replaces some privileged instructions with "hypercalls" to the Hypervisor
- Less overhead = better performance
- Disadvantage: Needs modified guest OS
- Traditional hypervisors are often partially para-virtualized (e.g. device drivers para-virtual, CPU & Memory fully virtual)
- E.g. Xen

VM Flavor B : Hosted VM



- Host OS controls hardware
- Hypervisor runs partly in user space, partly in host kernel
- Relies on host OS to provide drivers
- E.g. VMware Desktop Client

System VM: Co-Designed VM

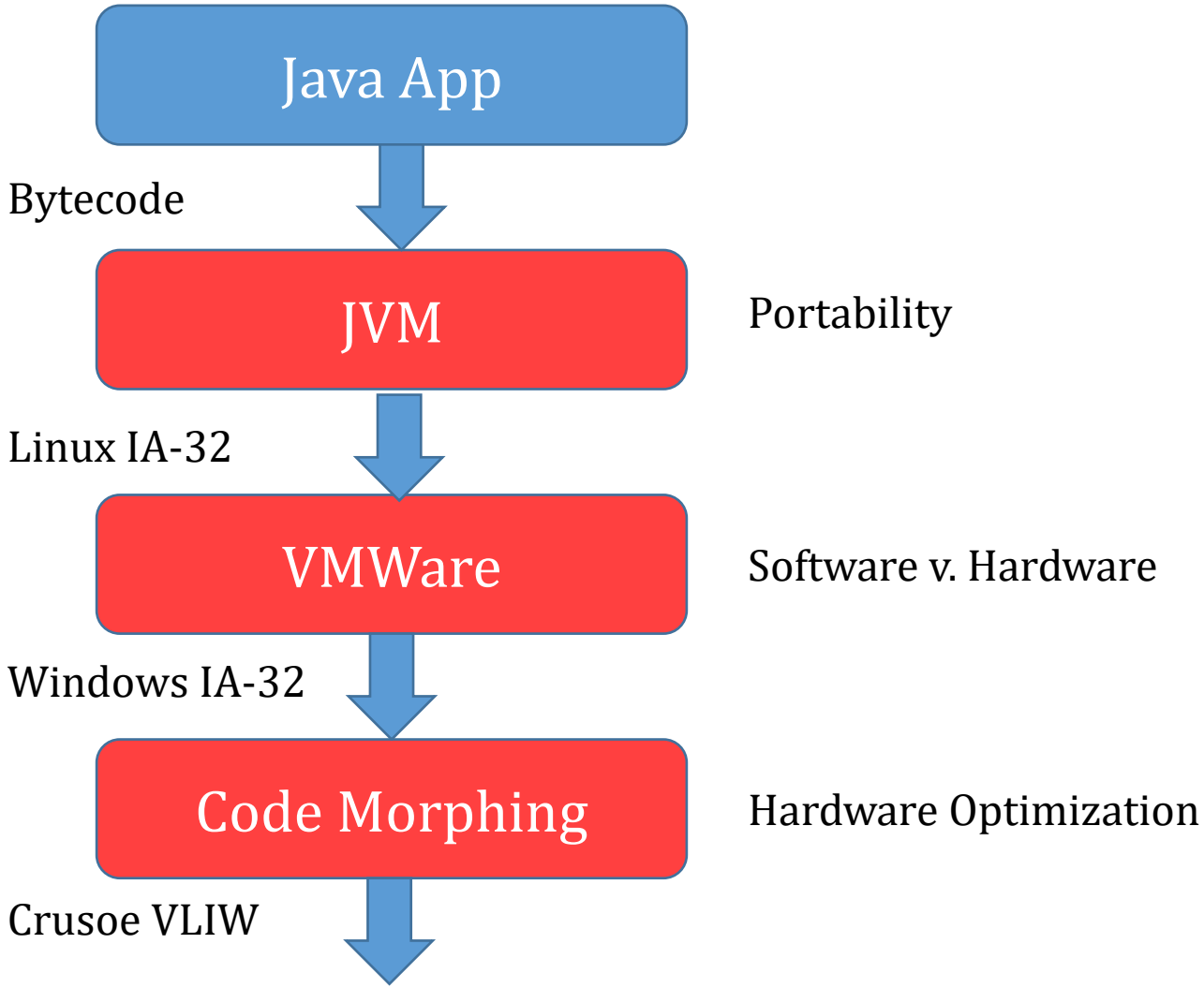


- Hypervisor designed closely with (and possibly built into) hardware ISA
- Goal: Performance improvement of existing ISA (or guest ISA) during runtime
- Hypervisor performs emulation from Guest ISA to native ISA
- E.g. Transmeta Crusoe
  - Native ISA: VLIW; Guest ISA : x86
  - Perform HW operations in VMM
    - e.g. instruction re-ordering
  - Goal: Power savings
  - 700MHz Crusoe runs x86 @ 500MHz
    - Cheaper and less power

Virtual Machine Taxonomy

Process v. System	Guest/Host	Style	Examples
Process VM (ABI)	Same ABI	Multi-Programming	UNIX
		Binary Optimizers	VALGRIND, Fast Malloc
	Different ABI	High Level Language	Java VM
		Emulators	Cygwin
		Emulators w/ Guest OS	DEC FX!32
System VM (ISA)	Either Same ISA or Different ISA	Traditional Hypervisors	KVM, VMware ESX
		Para-Virtualized	Xen
		Hosted	VMware Desktop
	Different ISA	Co-Designed	Transmeta Crusoe

Versatility

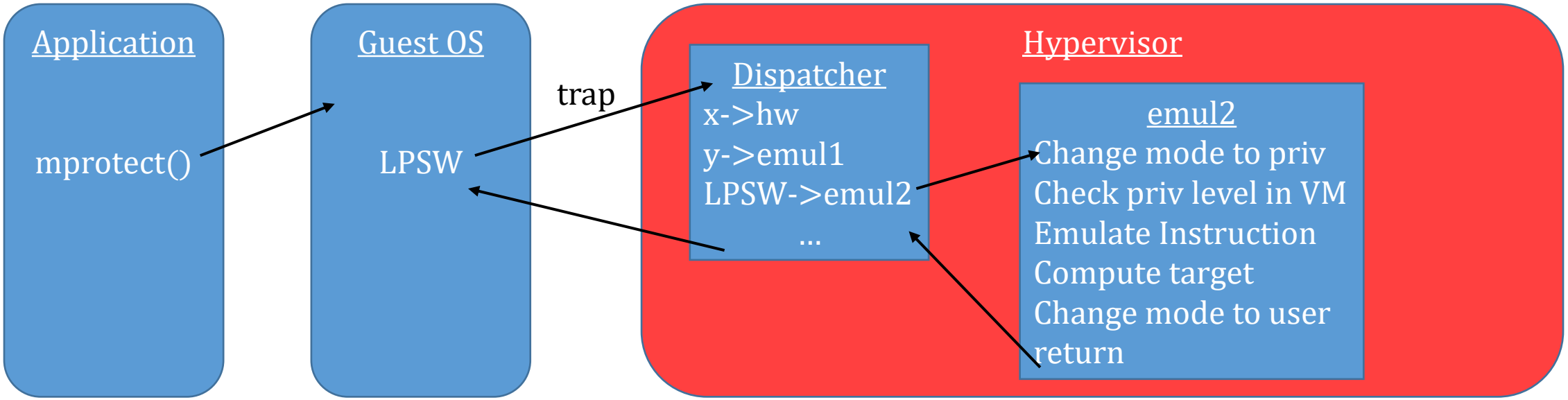


Virtualizing Individual Resources in System VMs

CPU Virtualization

- Each VM sees a set of "virtual CPUs"
- Hypervisors emulate guest OS privileged instructions
  - Hardware traps the instruction to the hypervisor
  - Hypervisor checks – should this instruction be emulated?
    - If so, emulate.
- Modern ISAs provide special interfaces for Hypervisors to run VMs
  - Hardware Assisted Virtualization
  - Intel VTx interface
  - AMD AMD-v interface
- Allow hypervisors to efficiently emulate guest OS privileged instructions

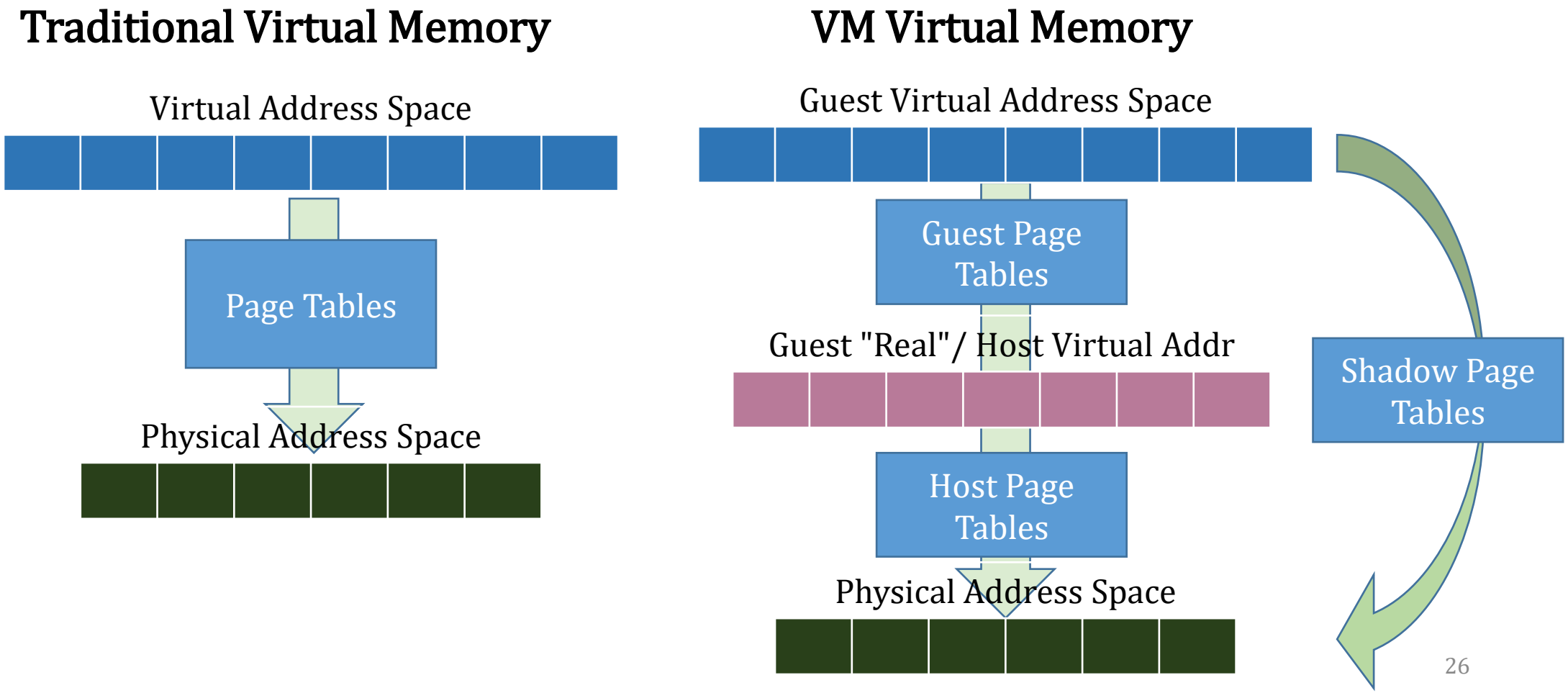
Execution of privileged Instruction



Problem: Scheduling Time Slices

- ISA continues executing user's instructions
- CPU Scheduler needs to intervene
- OS schedules timer interrupt
  - Interrupt handling routine swaps process out, and swaps new process in
- Hypervisor may be managing multiple Virtual OS's
  - Needs to manage VIRTUAL timer interrupt
- Hypervisor needs to run VIRTUAL timer only when Guest OS is active.
- Hypervisor also needs its own timer interrupt to manage swapping guests

Memory Virtualization for VMs



I/O Virtualization for VMs

- Hypervisor manages a virtual version of each physical device
- I/O activity directed at virtual device is converted to physical request

- Option 1: Device Emulation
  - Trap/Emulate each Guest I/O instruction in hypervisor
  - Slow and difficult to emulate the effect of combinations of I/O instructions
- Option 2: Para-Virtual devices (most common)
  - Special device drivers inserted in guest OS to talk to Hypervisor
- Option 3: Direct Device Access (Fast but not scalable, gaining popularity)
  - Allow VM to work on physical device
  - Requires IOMMU and VT-d support from hardware

I/O Memory Management Unit