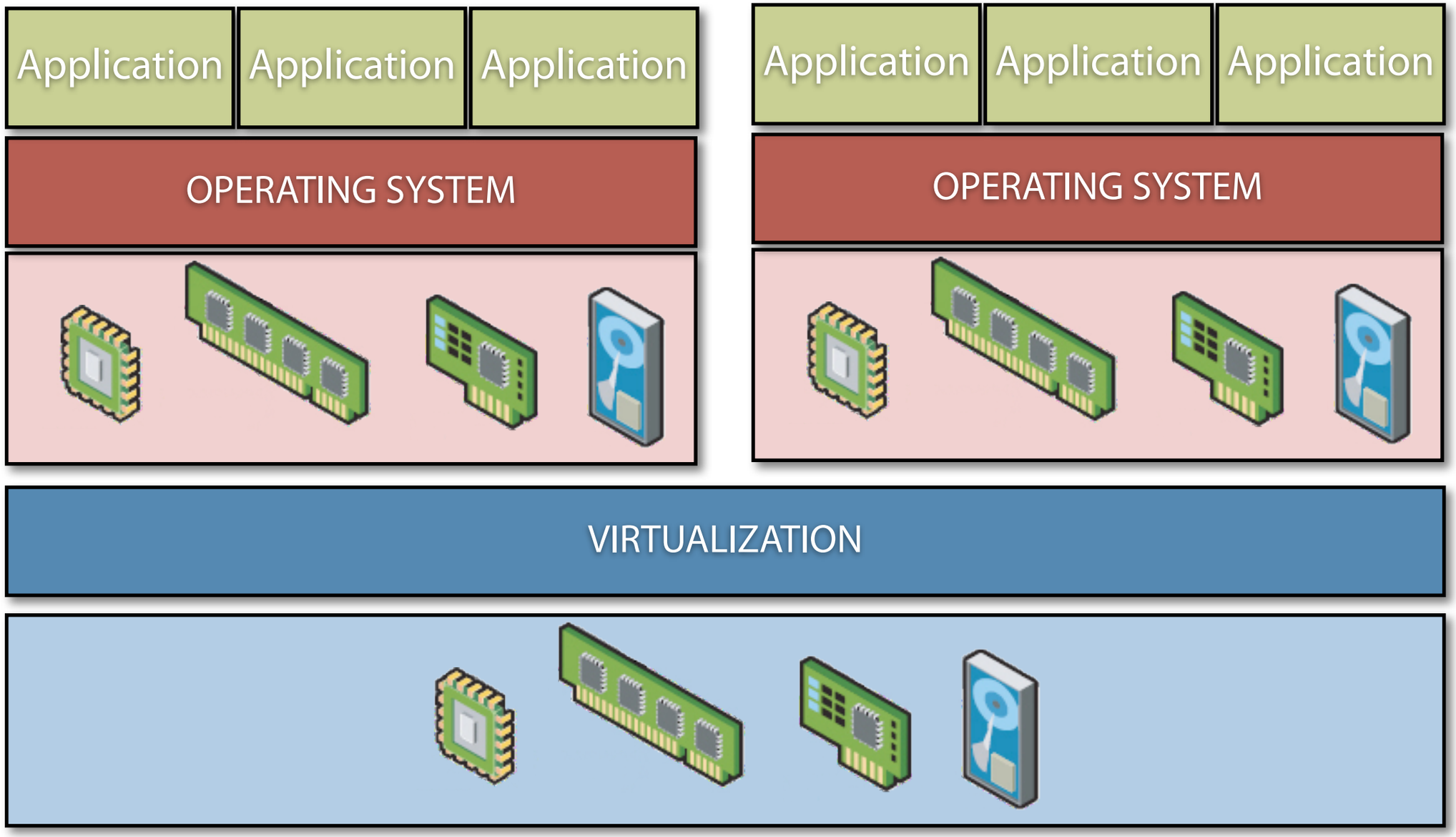
Virtualization



giovedì 29 novembre 12

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

Consolidate Resources

- **Server consolidation**
  - Reduce number of servers
  - Reduce space, power and cooling
  - 70%-80% reduction in numbers
- **Client consolidation**
  - Developers: test multiple OS versions, distributed application configurations on a single machine
  - End users: Windows on Linux, Windows on Apple
  - Reduce physical desktop space, avoid managing of multiple physical computers

Improve System Management

- **Datacenter 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 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

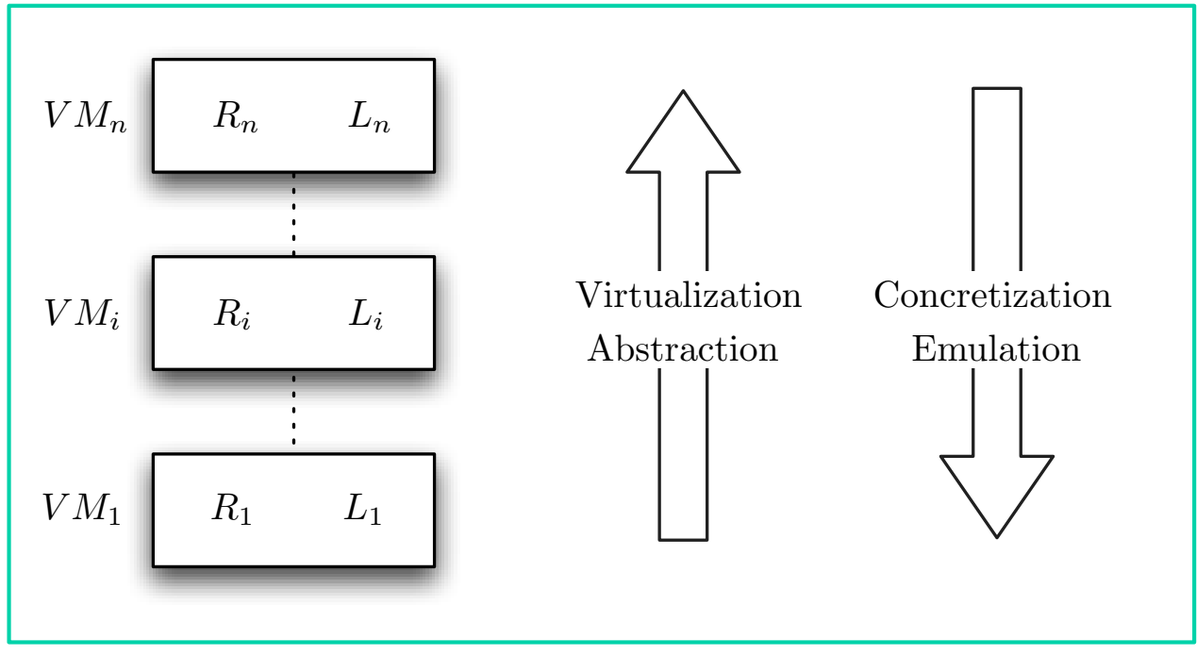
- **Fast, automated recovery**
  - automated failover/restart within a cluster
  - disaster recovery across sites
  - VM portability enables this to work reliably across potentially different hardware configurations
- **Fault tolerance**
  - hypervisor-based fault tolerance against hardware failures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

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

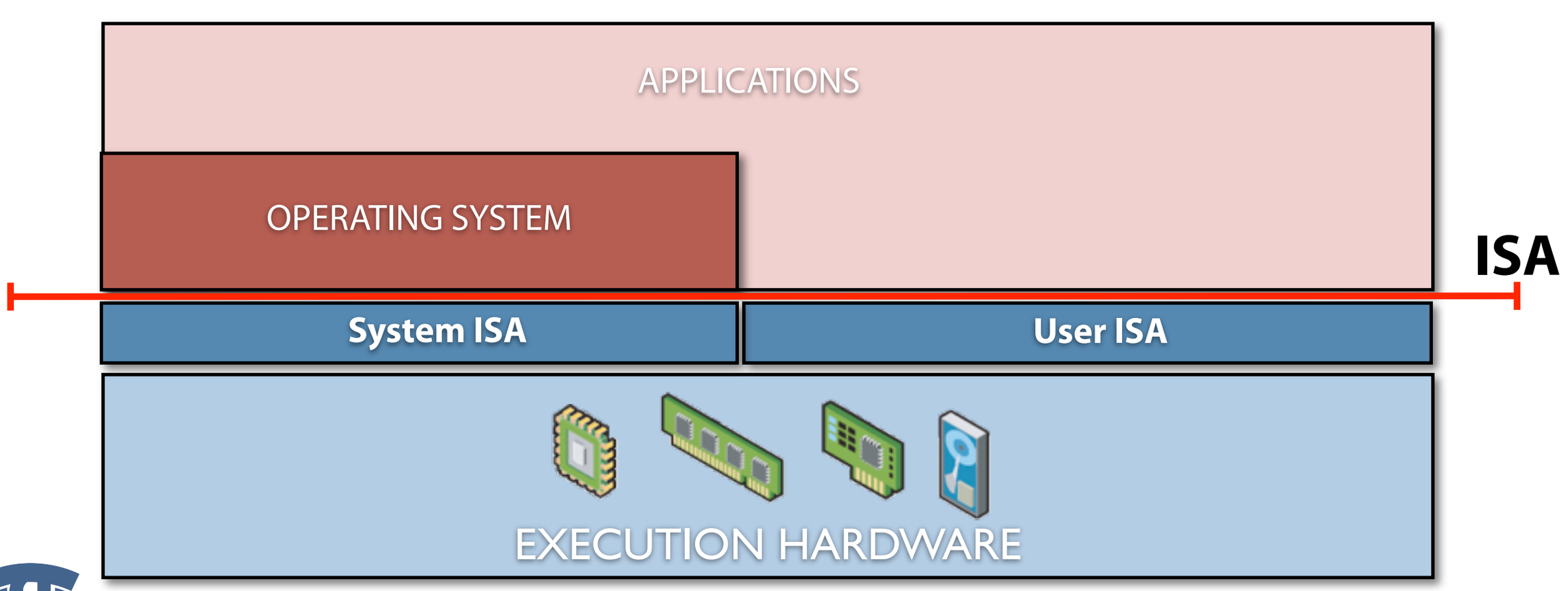


- **Machine: denotes the system on which software is executed.**
  - to an operating system this is generally the physical system
  - to an application program a machine is defined by the combination of hardware and OS-implemented abstractions

- **Typical Layers**
  - VM_4: Applications
  - VM_3: Operating System
  - VM_2: Assembler Machine
  - VM_1: Firmware Machine
  - VM_0: Hardware Machine

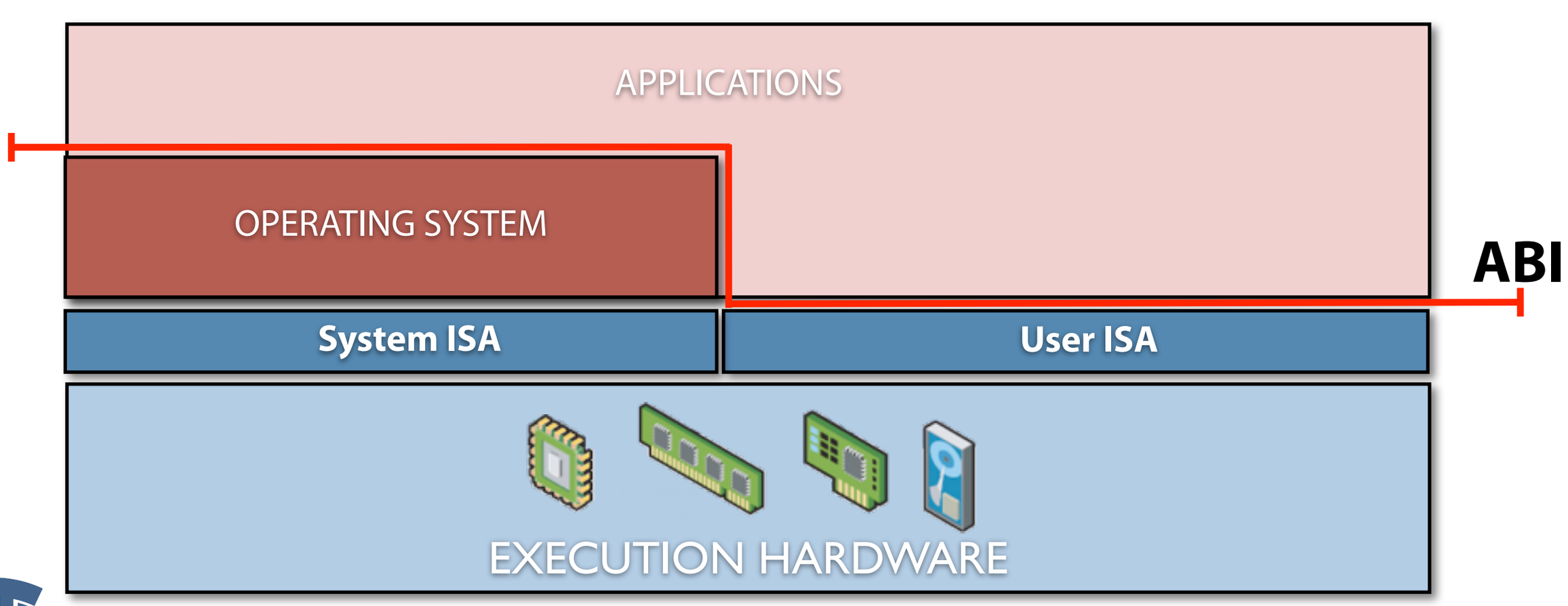
Machine Level Abstraction

- **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**



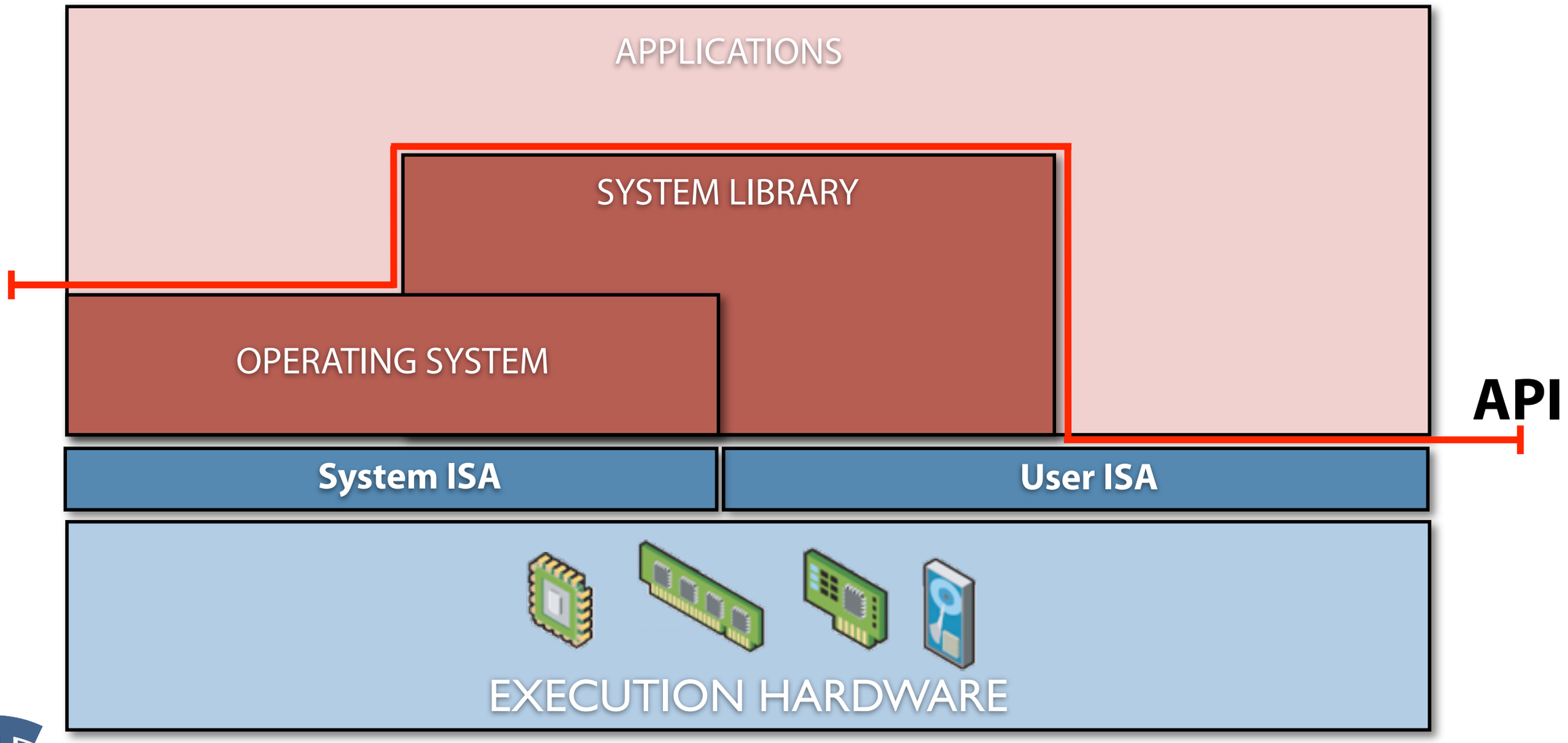
OS Level Abstraction

- **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

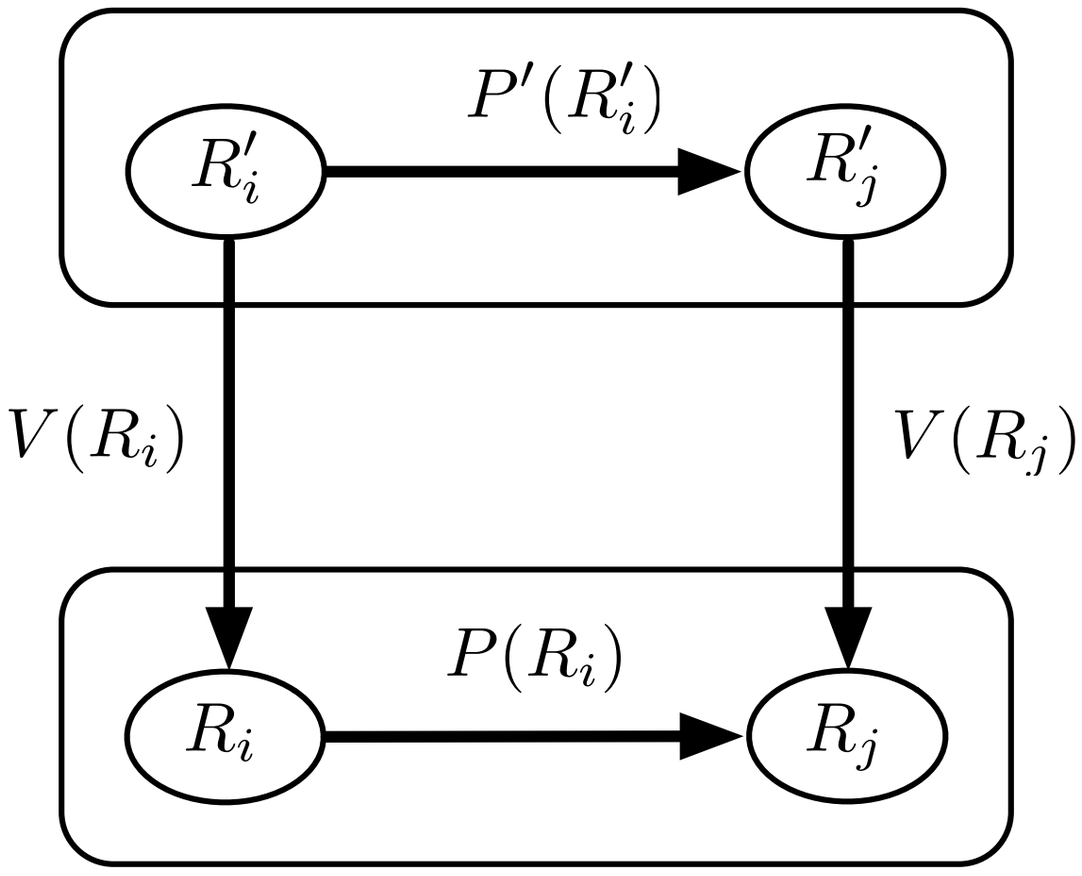
- Virtualization is defined as the construction of an isomorphism that maps a **guest** machine to an existing **host** machine such that:

- 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

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



Properties

- **Isolation**
  - Fault Isolation
    - Fundamental property of virtualization
  - Software Isolation
    - Software versioning
    - DLL Hell
  - Performance Isolation
    - Accomplished through scheduling and resource allocation
- **Encapsulation**
  - All VM state can be captured into a file
    - Operate on VM by operating on file
    - mv, cp, rm
  - Complexity
    - Proportional to virtual HW model
    - Independent of guest software configuration
- **Interposition**
  - All guest actions go through monitor
  - Monitor can inspect, modify, deny operations

Perspectives

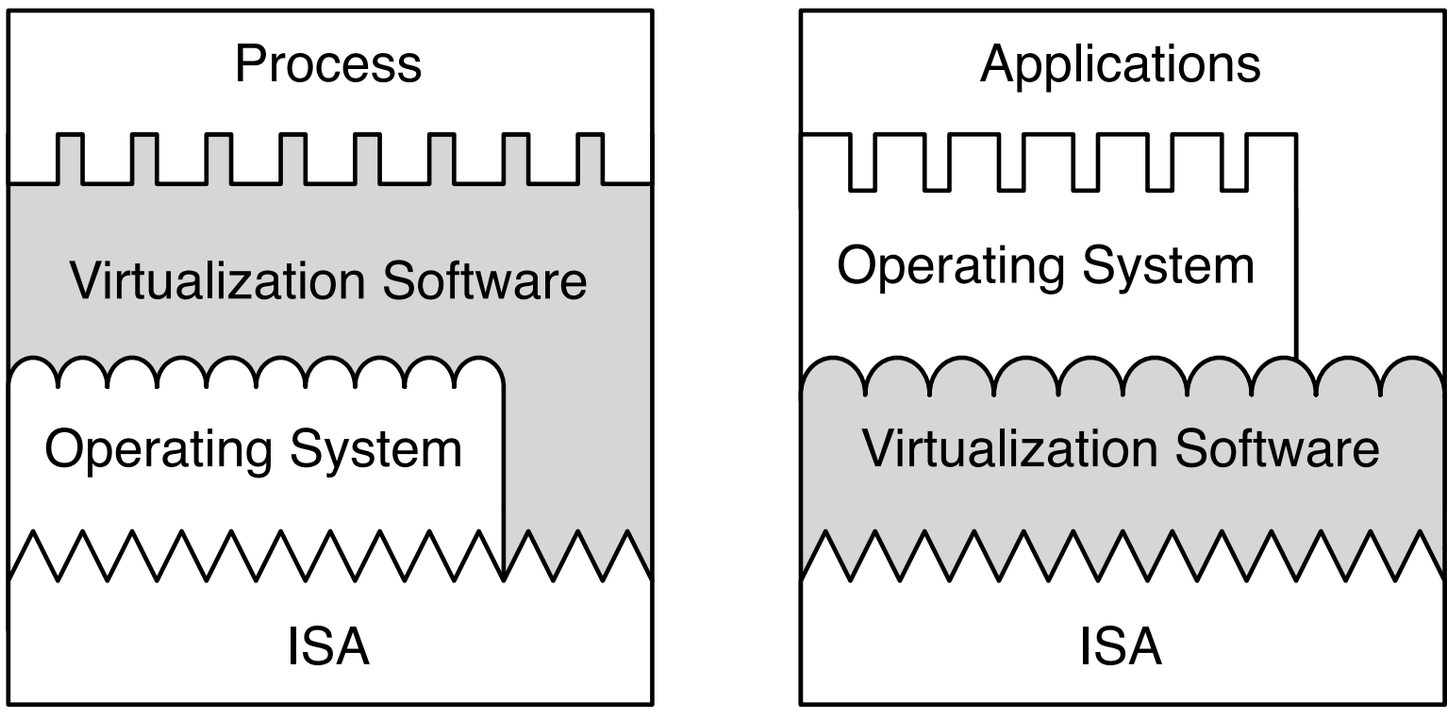
- ***Process perspective*:** the system ABI defines the interface between the process and machine
  - user-level hardware access: logical memory space, user-level registers and instructions
  - OS mediated: Machine I/O or any shared resource or operations requiring system privilege.

- ***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

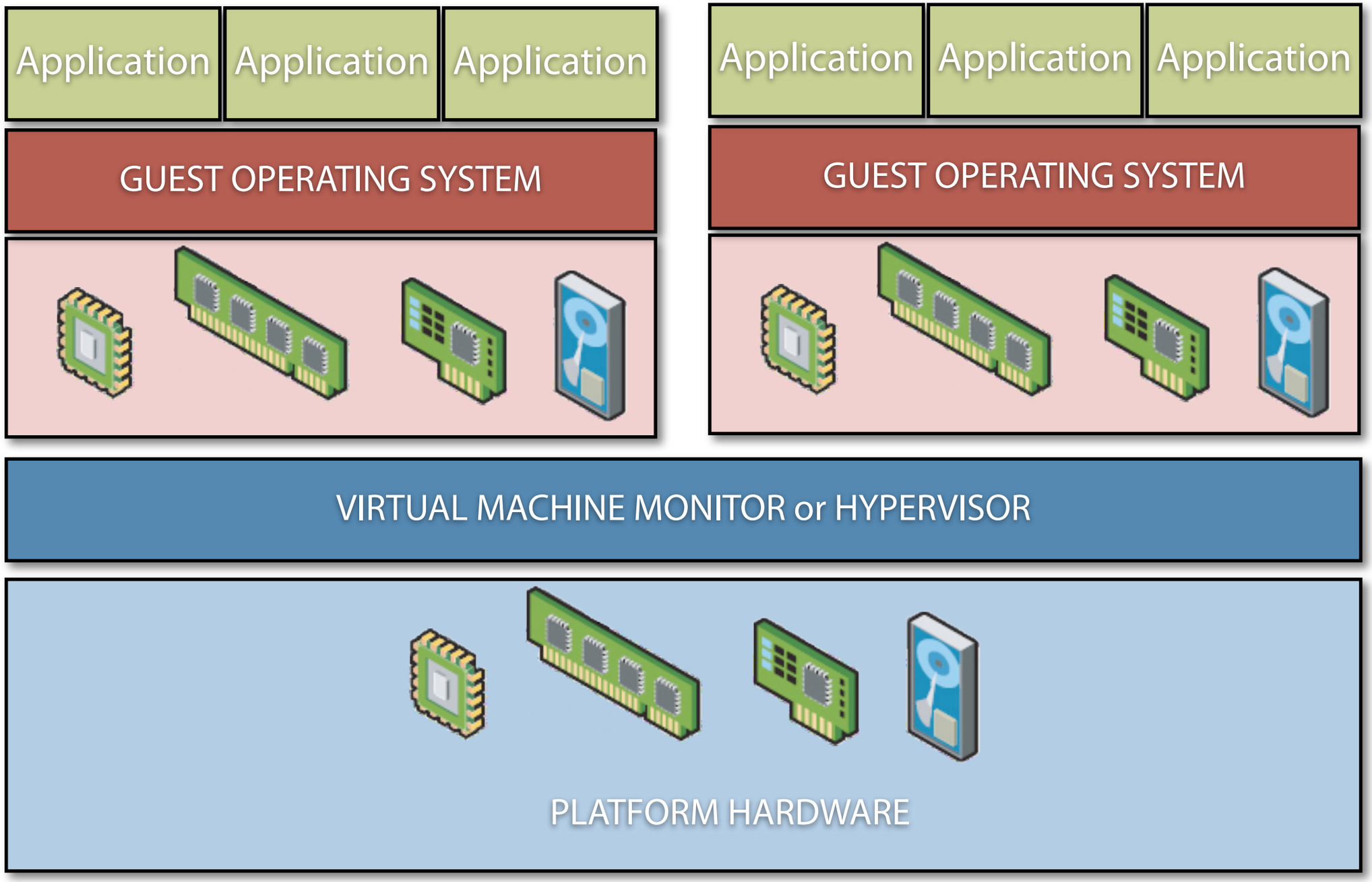
- Virtual machine executes software (process or operating system) in the same manner as target machine
  - Implemented with both hardware and software
  - VM resources may differ from that of the physical machine
  - Generally not necessary for VM to have equivalent performance

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



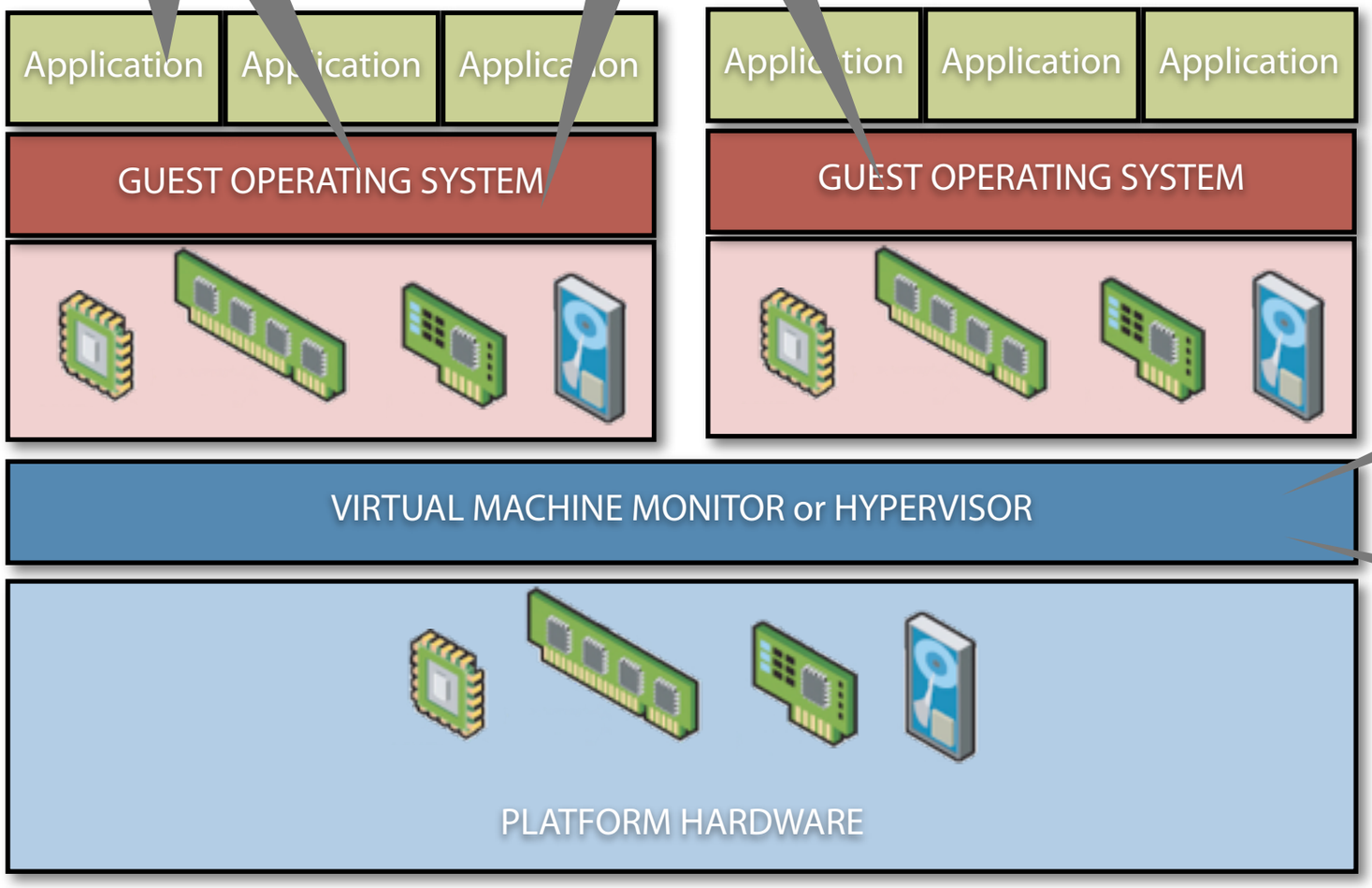
Virtual Machine Monitor



VMM Challanges

OS and Apps in a VM don't know that the VMM exists or that they share CPU resources with other VMs

VMM should isolate Guest SW stacks from one another

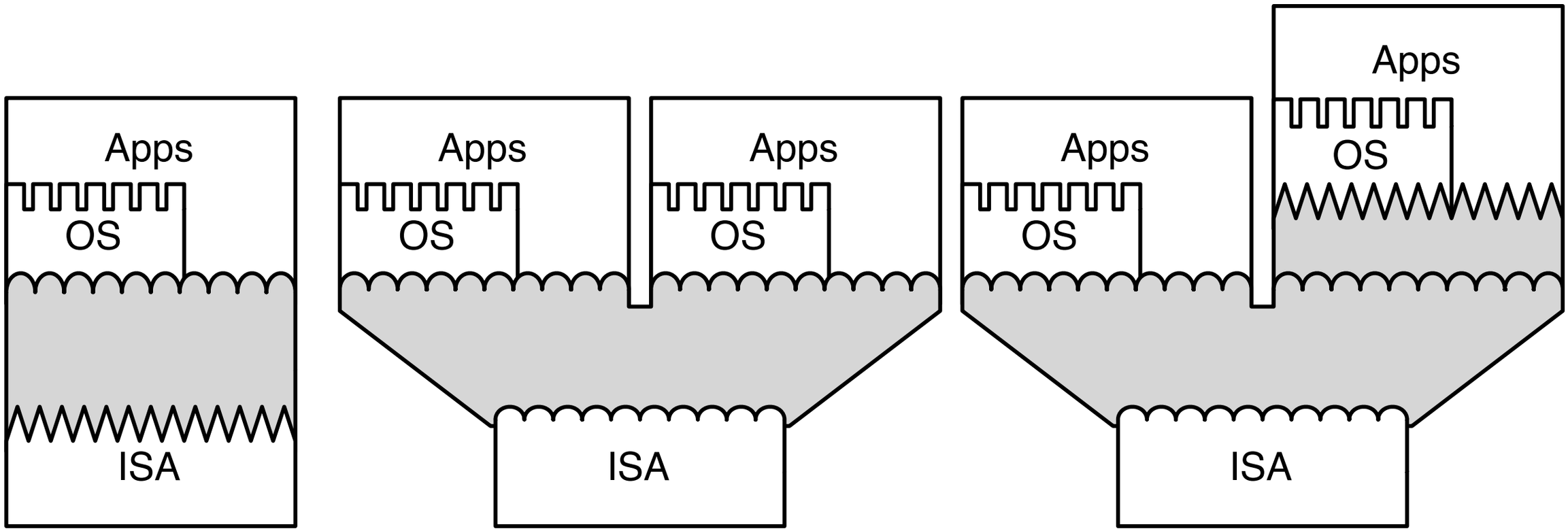


VMM should run protected from all Guest software

VMM should present a virtual platform interface to Guest SW

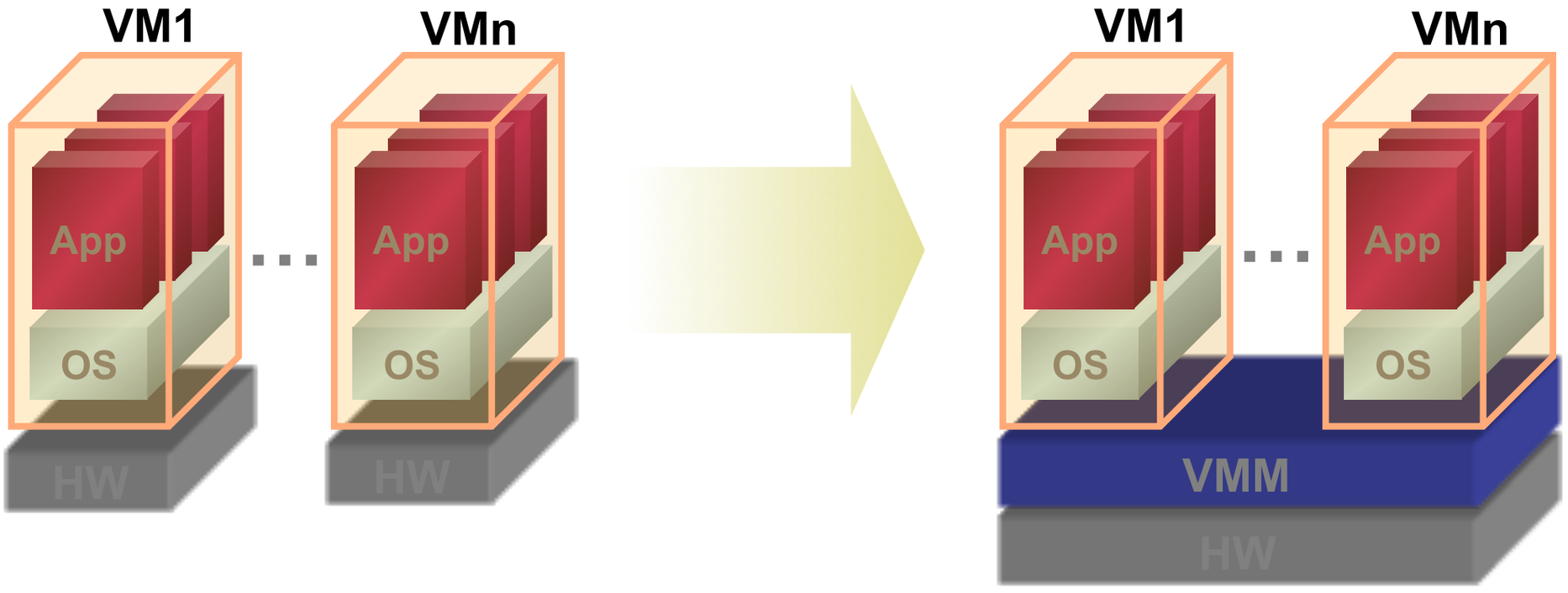
VM Capabilities

- **Emulation**: cross platform compatibility
- **Optimization**: by considering implementation specific information
- **Replication**: making a single resource or platform appear as many



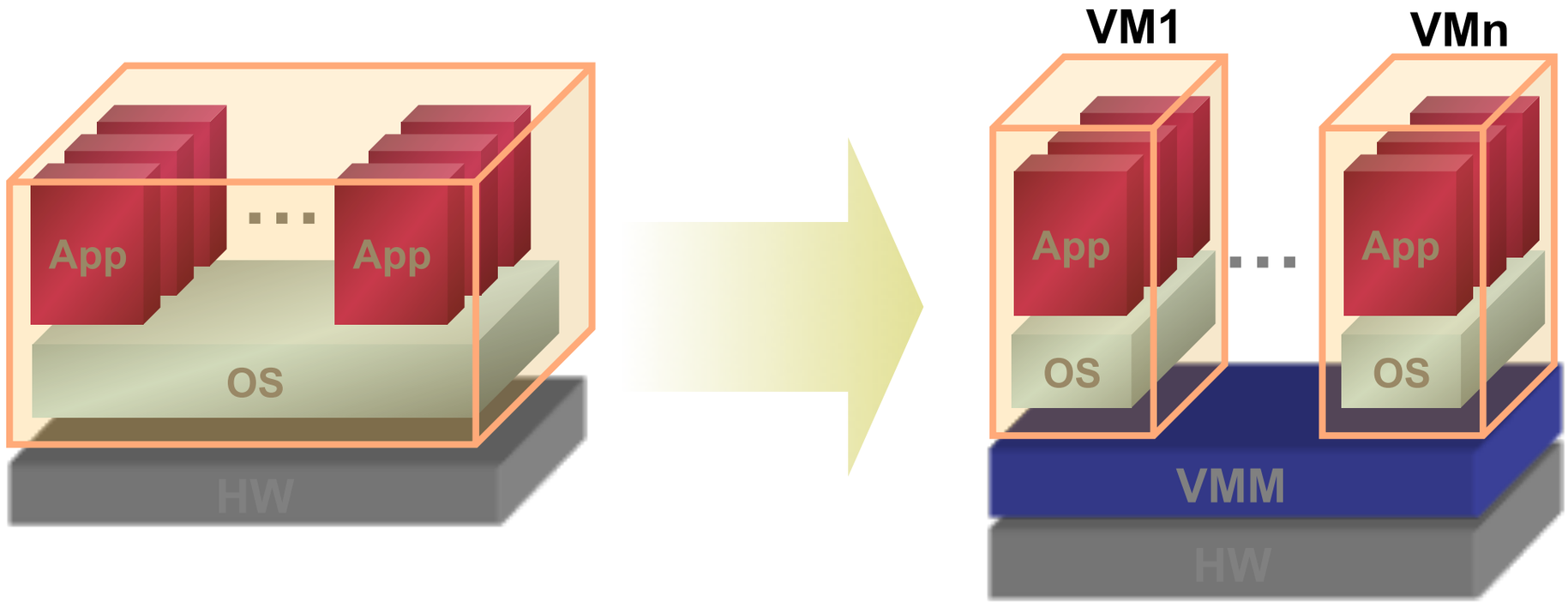
VM Usage: Workload Consolidation

- 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



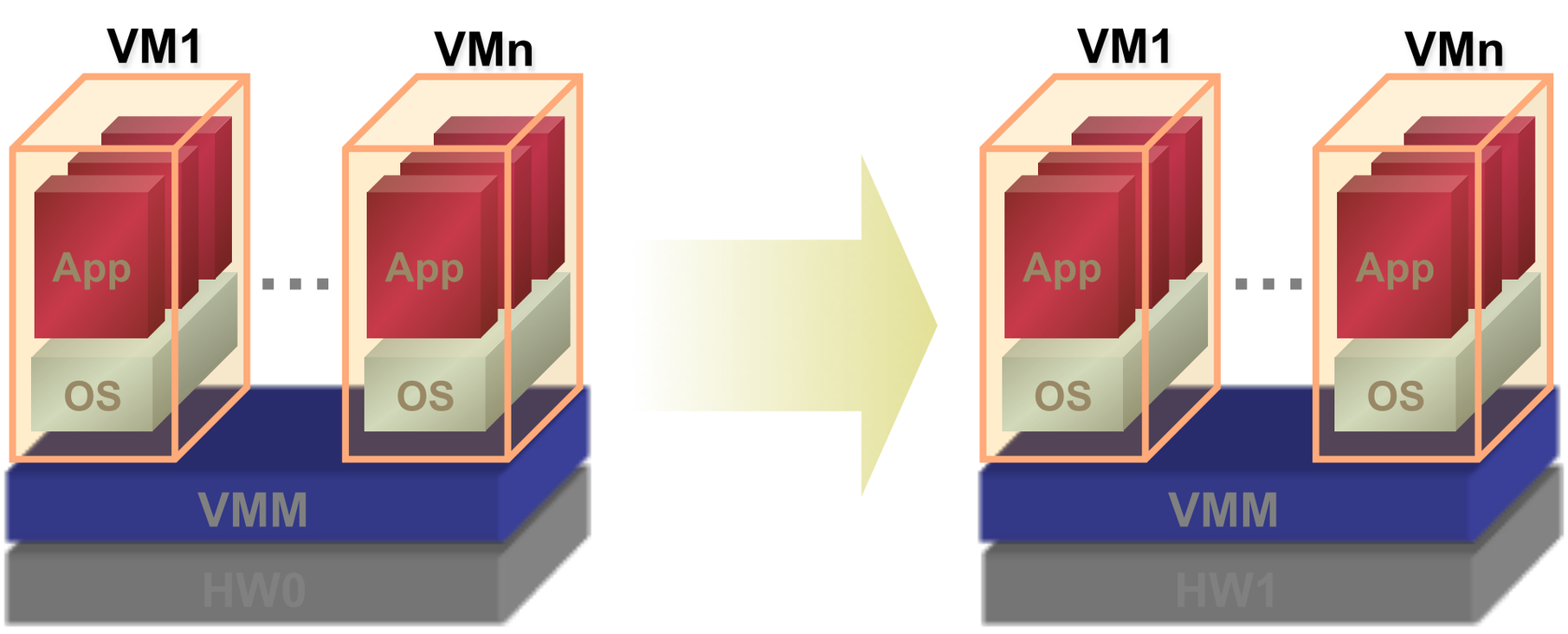
VM Usage: Workload Isolation

- 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)



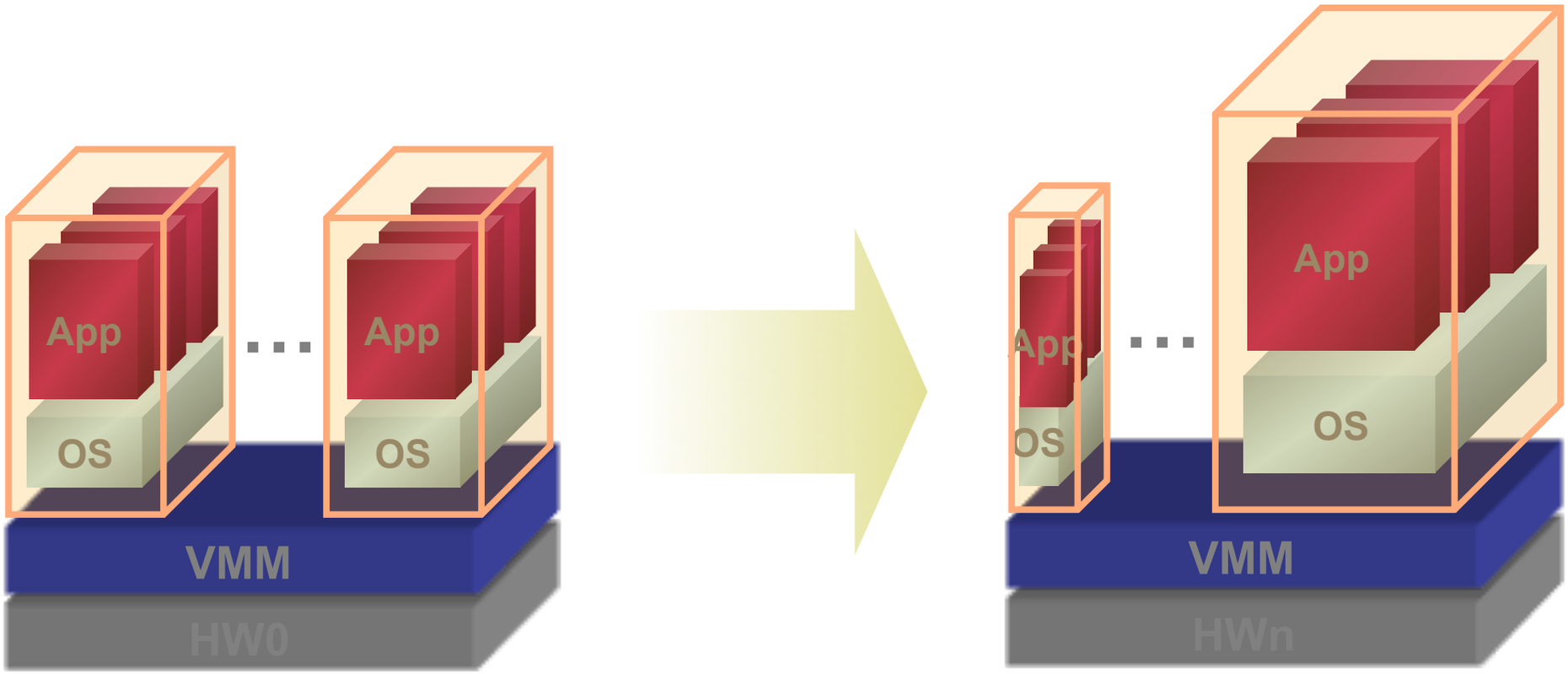
VM Usage: Workload Isolation (I)

- 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



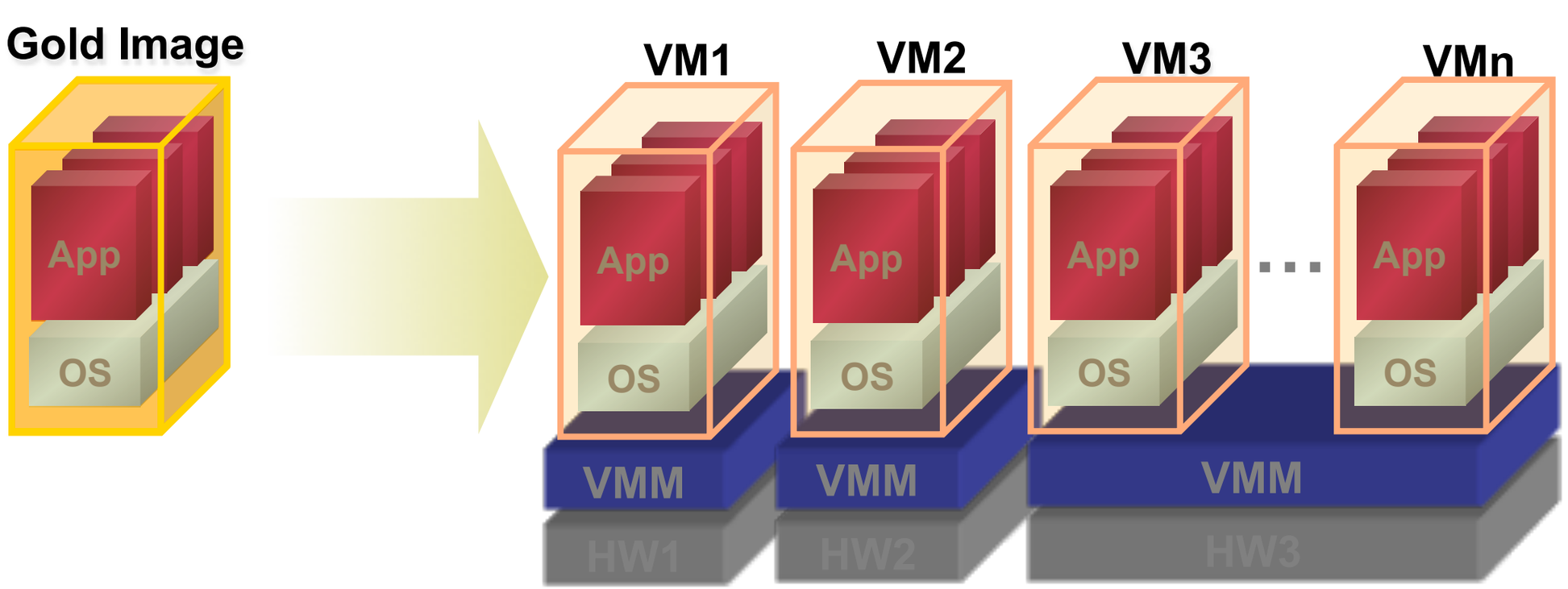
VM Usage: Workload Isolation (II)

- 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



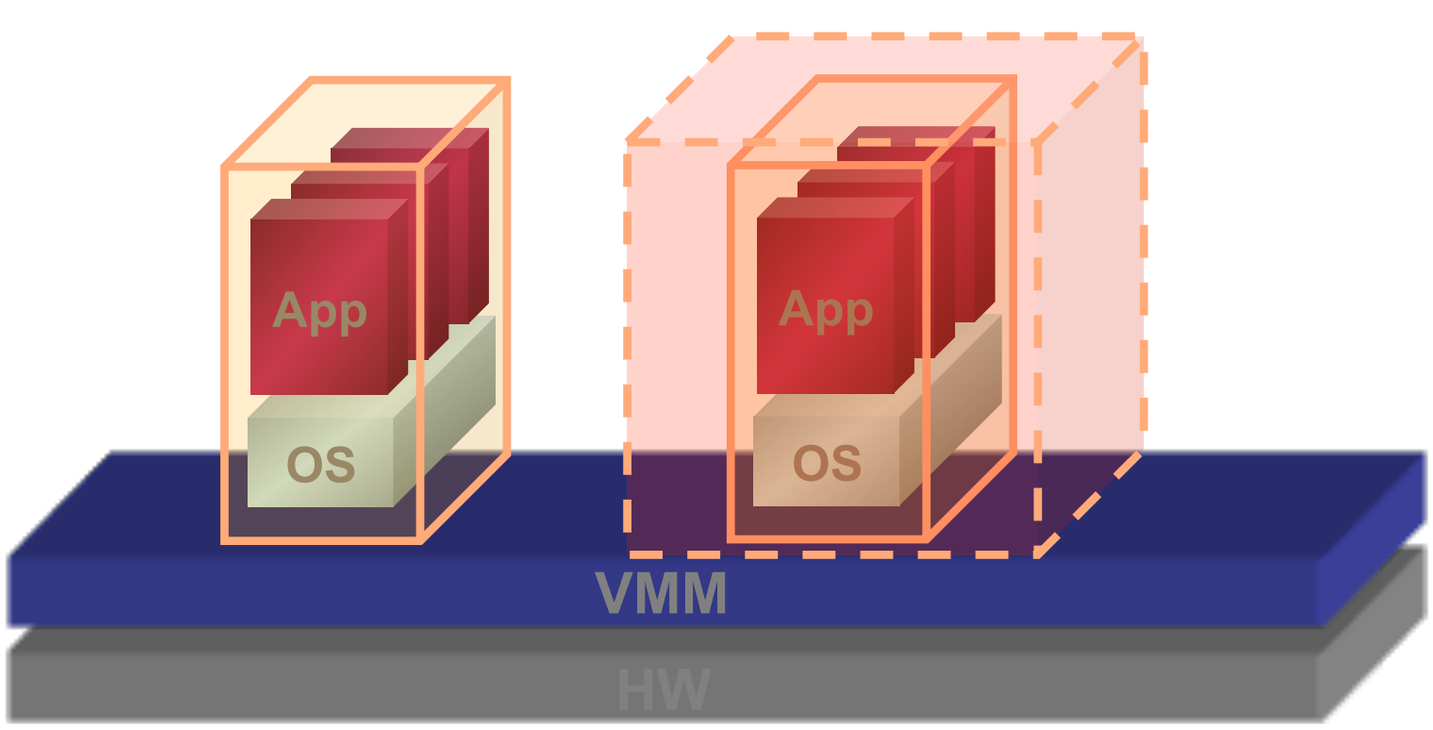
VM Usage: Fast Deployment

- 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)

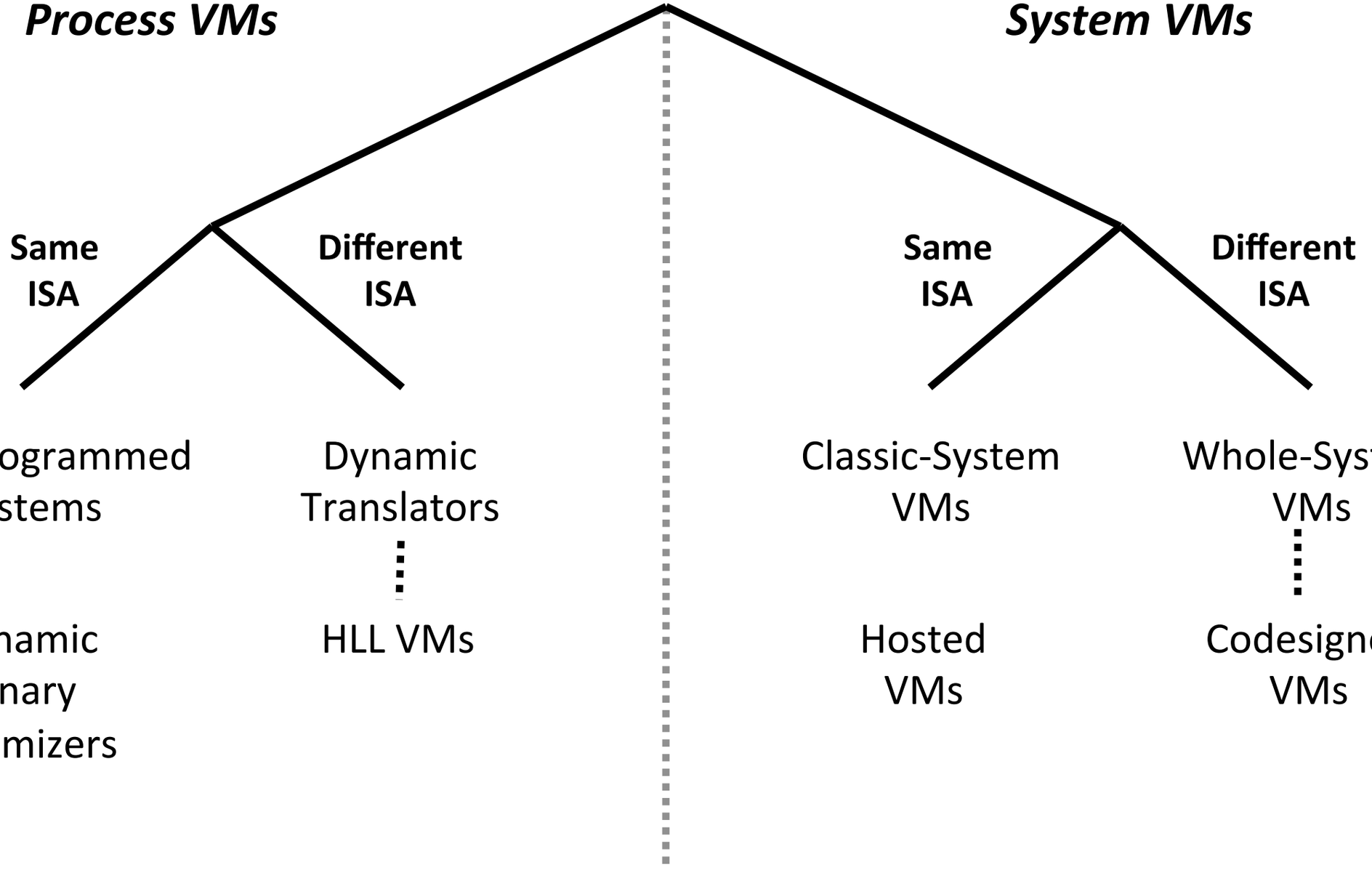


VM Usage: Testing and Develpment

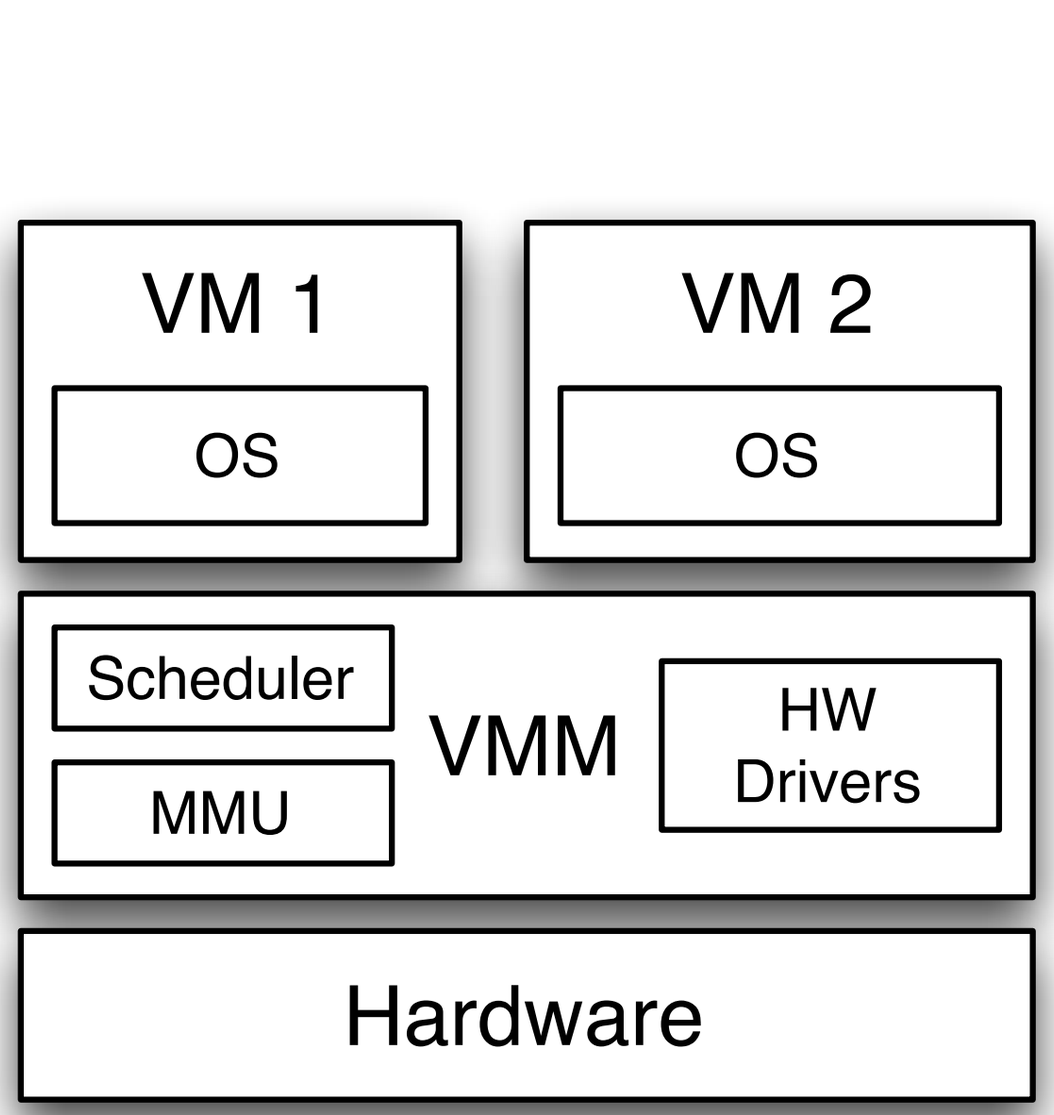
- Development and testing environments
  - A VM with standard tools is distributed amongst developers
  - Releasing new revisions of tools, patches, etc. is very simple
- Business Agility and Productivity
  - It allows to easily transform environments (Development to test, back to development, etc)
-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



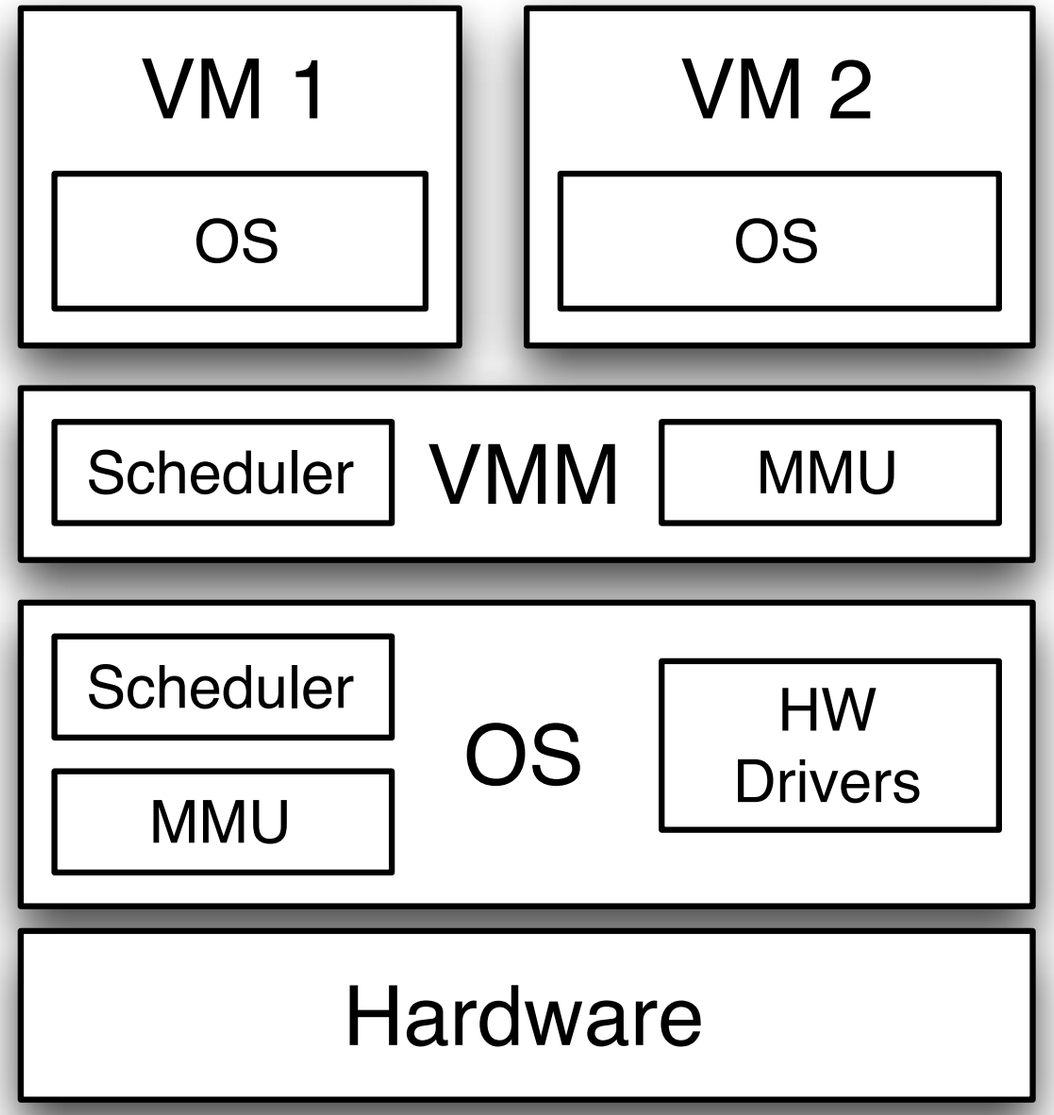
VM Taxonomy


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