POLITECNICO DI MILANO



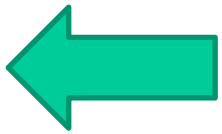


Software Infrastructures (i.e., from Datacenter to Cloud)

The topics of the course: what are we going to see today?

A. HW Infrastructures:
- **System-level**: Computing Infrastructures and Data Center Architectures, Rack/Structure;
- **Node-level**: Server (computation, HW accelerators), Storage (Type, technology), Networking (architecture and technology);
- **Building-level**: Cooling systems, power supply, failure recovery

B. SW Infrastructures:
- Virtualization: Process/System VM, Virtualization Mechanisms (Hypervisor, Para/Full virtualization)
- **Computing Architectures**: Cloud Computing (types, characteristics), Edge/Fog Computing, X-as-a service
- **Machine and deep learning-as-a-service**

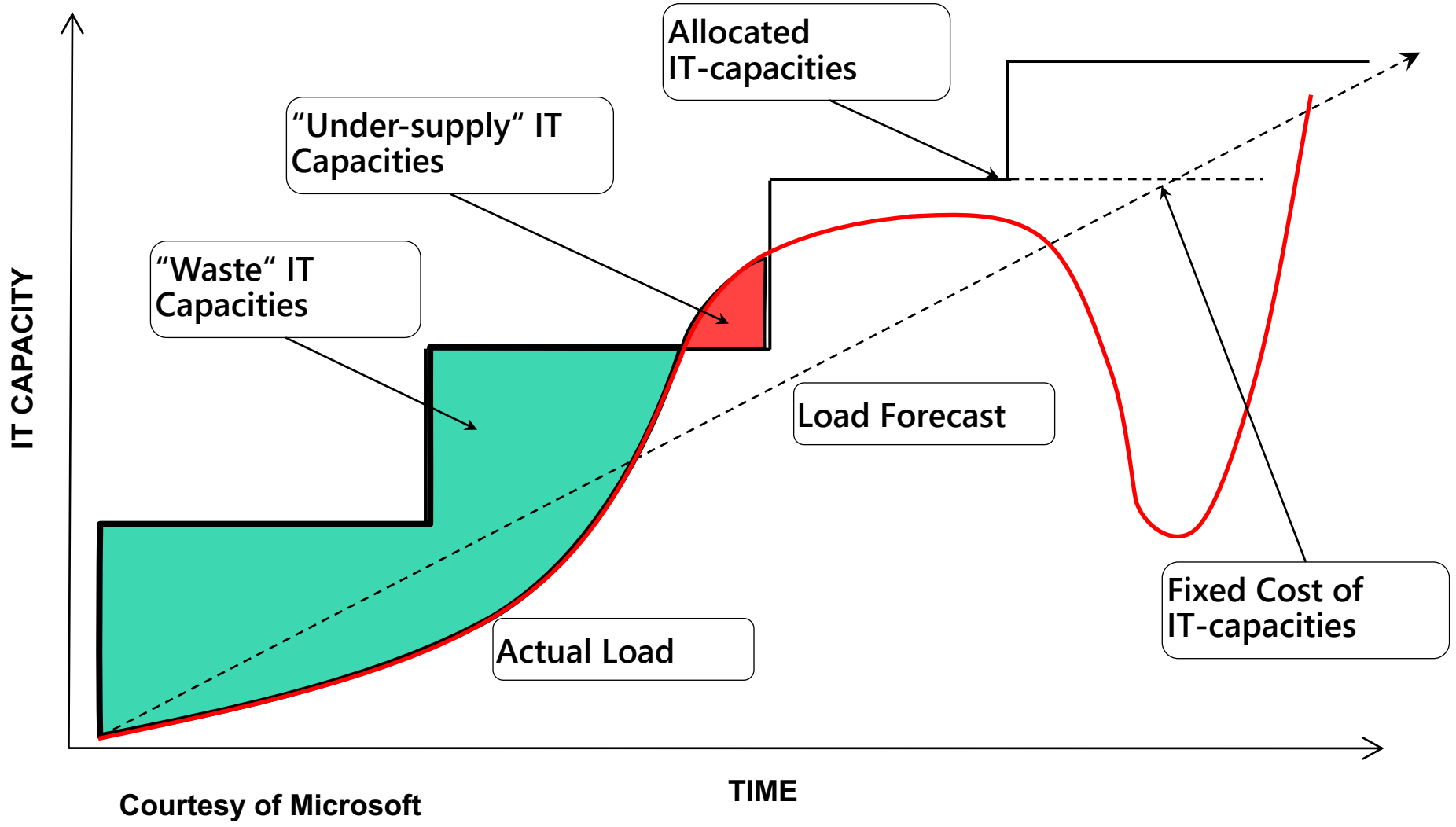


C. Methods:
- **Reliability and availability of datacenters** (definition, fundamental laws, RBDs)
- **Disk performance** (Type, Performance, RAID)
- **Scalability and performance of datacenters** (definitions, fundamental laws, queuing network theory)

What is Cloud Computing?

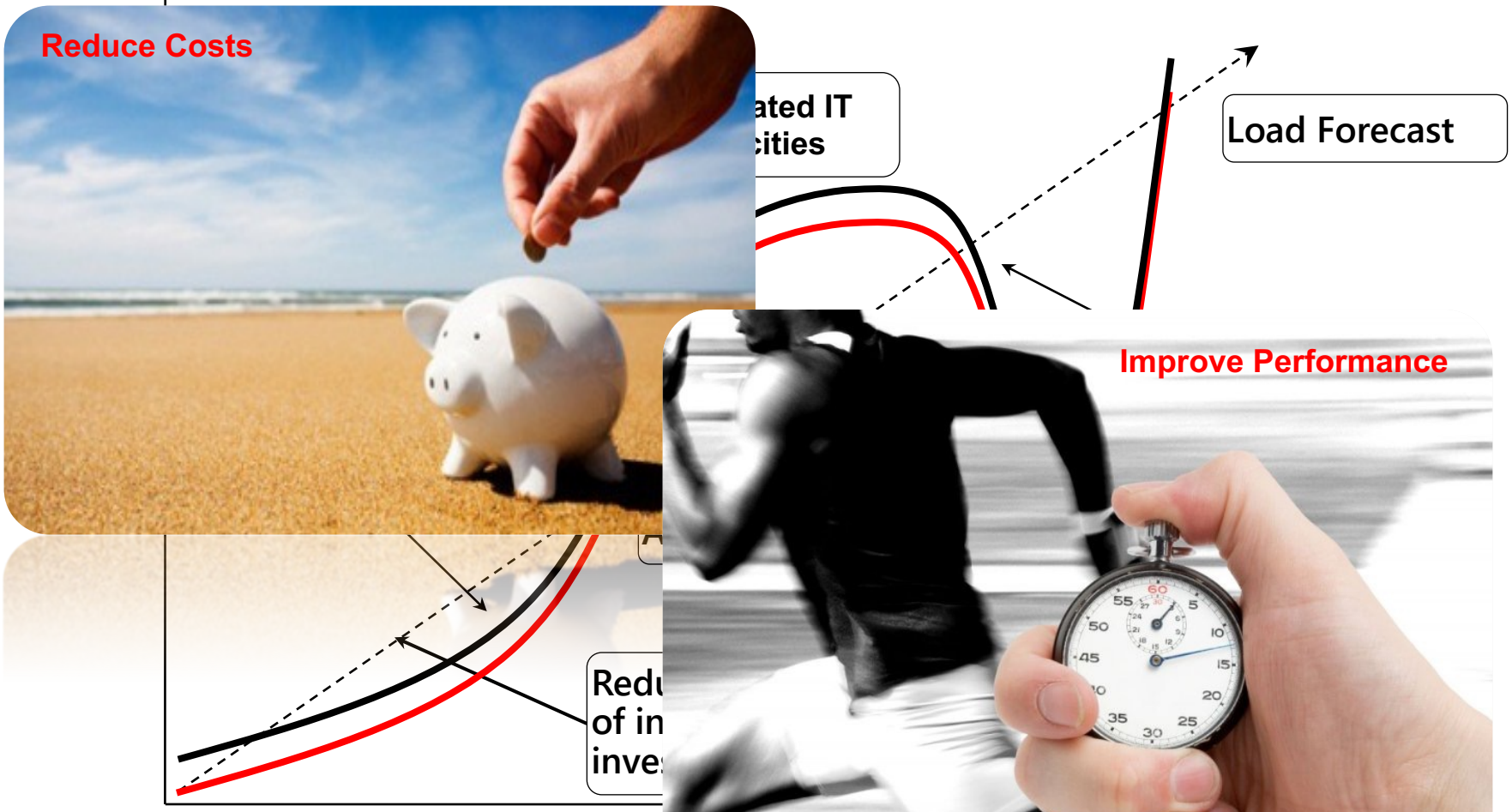
- A coherent, large-scale, publicly accessible collection of computing, storage, and networking resources
- Available via Web service calls through the Internet
- Short- or long-term access on a pay-per-use basis

Over-provisioning - Out of Cloud



Courtesy of Microsoft

Cloud-provisioning

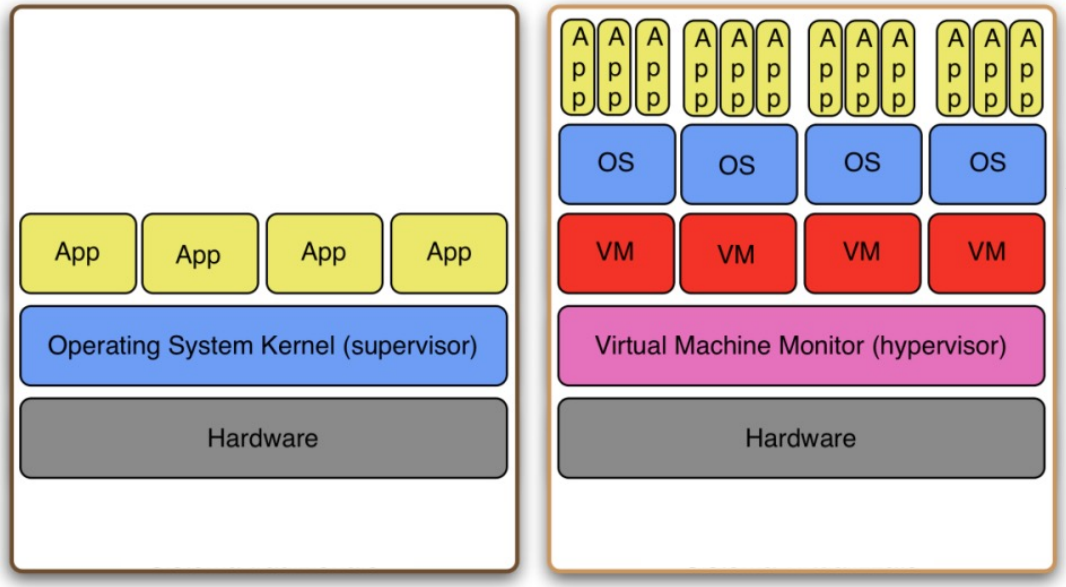


Courtesy of Microsoft

How is Cloud implemented? Virtualization

- Hardware resources (CPU, RAM, ecc...) are partitioned and shared among multiple **virtual machines** (VMs)
- The virtual machine monitor (VMM) governs the access to the physical resources among running VMs
- Performance, isolation and security

A single OS



VMs run possibly different OSs

Computing Infrastructures



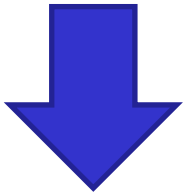


Virtual Machines

Machine

A *Machine* is an execution environment capable of running a program.

What's the difference between a physical machine and a virtual machine?

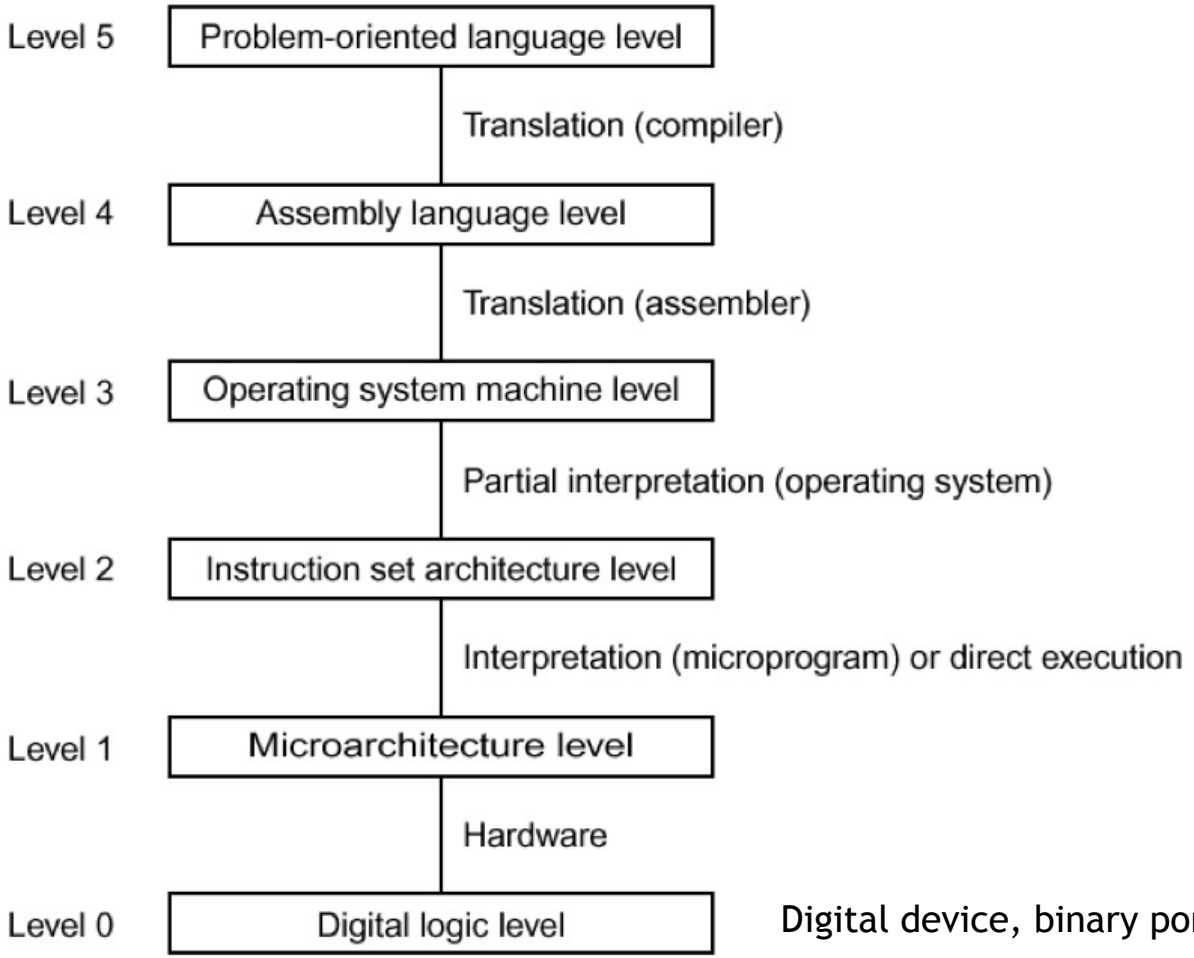


We have to go back to the **computer architecture**:

- Sets of instructions characterized by the levels at which are considered
- OS creates new instructions for programs to access devices/hw

Machine levels

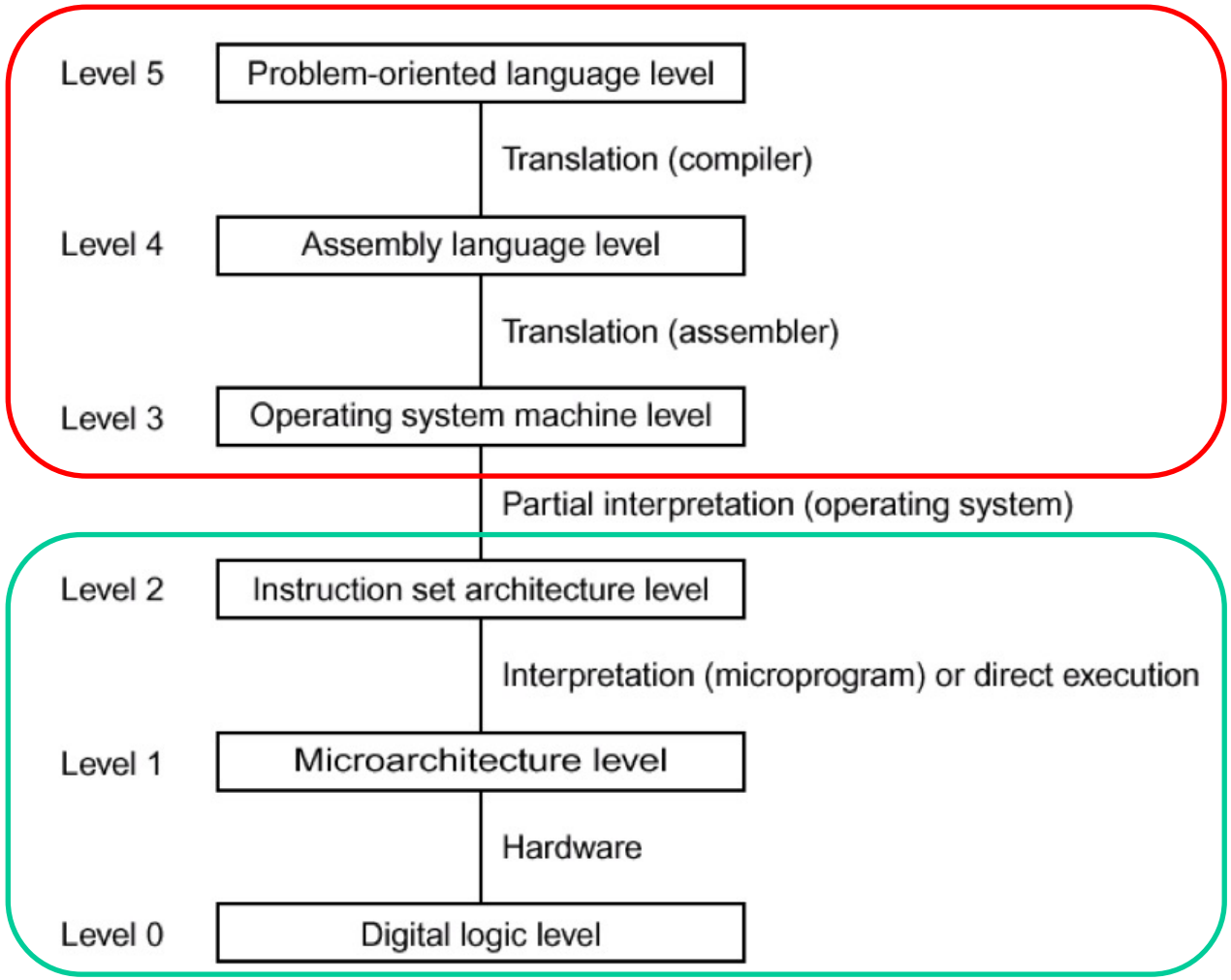
In computer architecture, the set of instructions that a program can use might be structured at different levels.



Digital device, binary ports,…

Machine levels

We usually program the software part, exploiting the hardware.

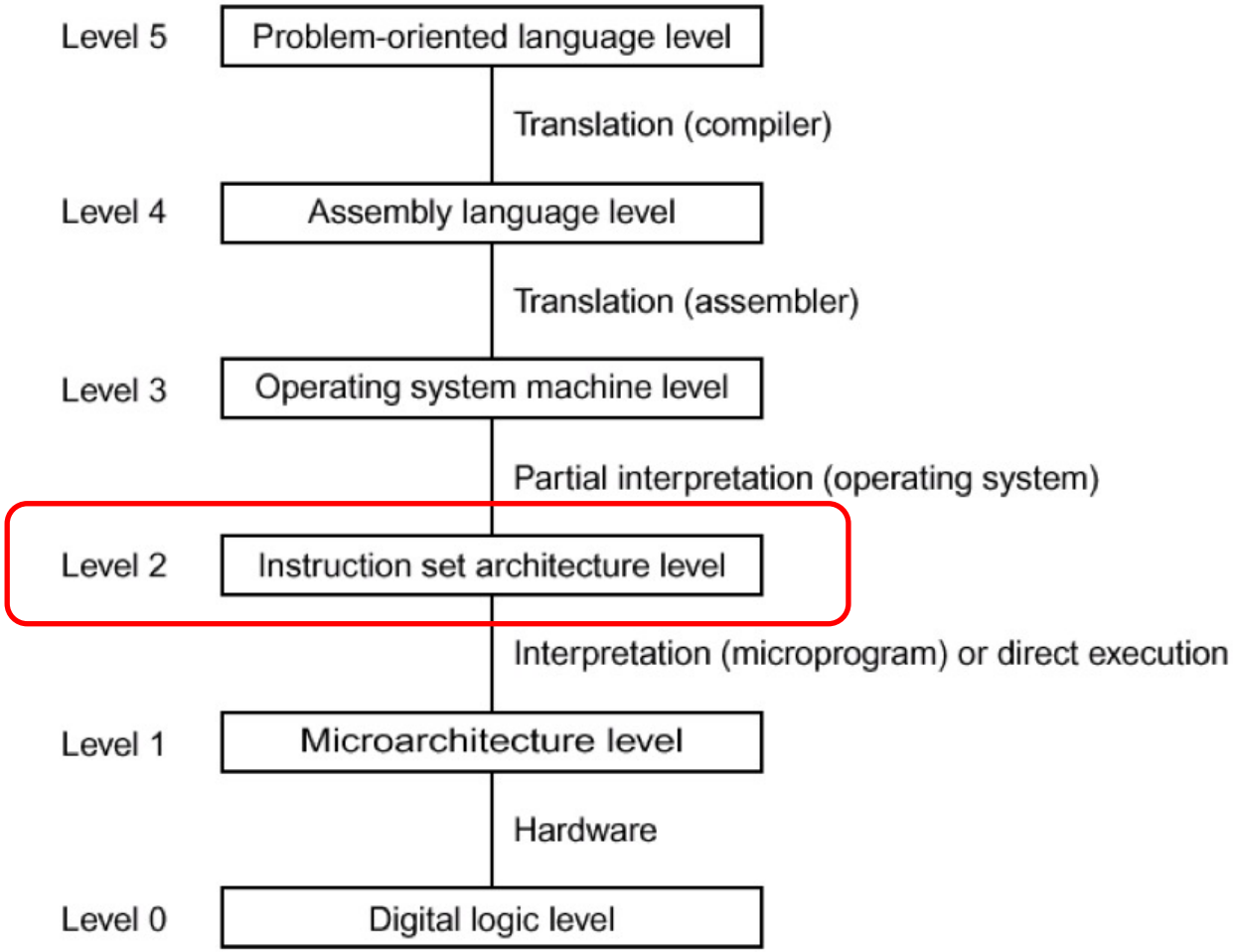


SOFTWARE

HARDWARE

Instruction Set Architecture

- The ISA corresponds to Level 2 in the layered execution model.
- ISA marks the division between hardware and software.



Instruction Set Architecture

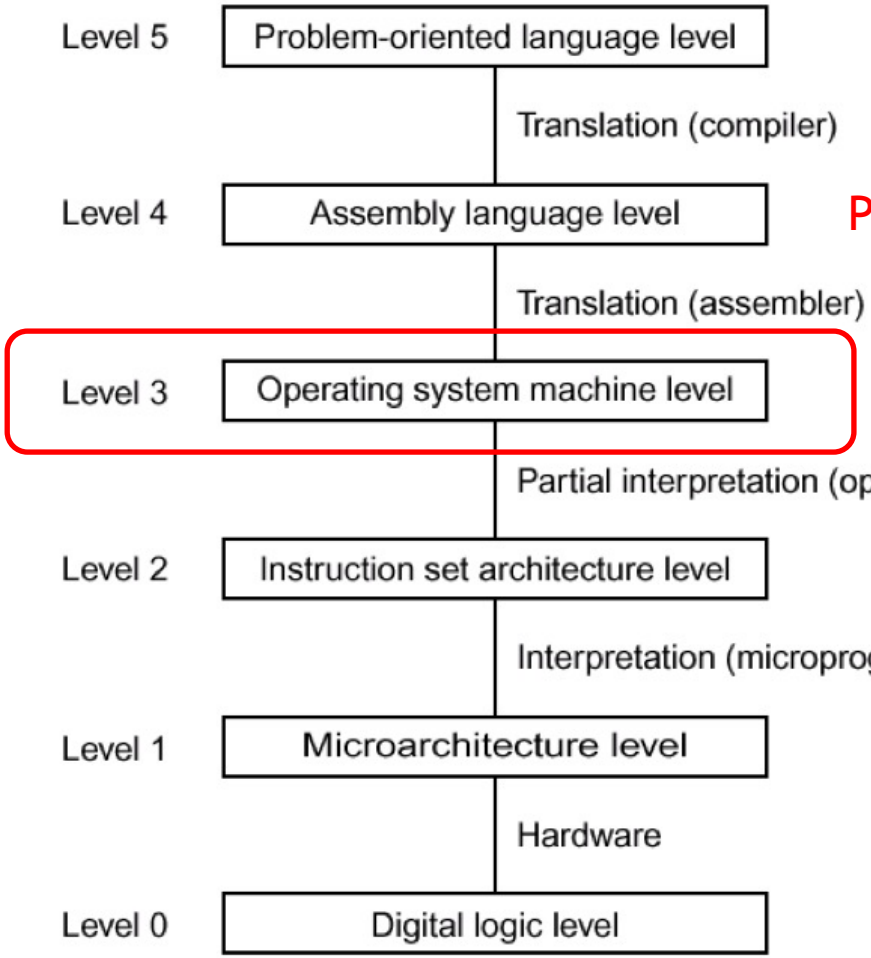
- **The ISA corresponds to Level 2 in the layered execution model.**
- **ISA marks the division between hardware and software**

- **User ISA**: aspects of the ISA that are visible to an application program (sum, multiplication,…, logical operations, branches, etc...). When application interacts with the HW, User ISA is used.

- **System ISA**: aspects visible to supervisor software (i.e., the OS) which is responsible for managing hardware resources (e.g., hide the complexity of CPUs, define how app access memory, communicate with the HW). When the OS interacts with the HW (Drivers, MM, Sched.), System ISA is used.

Application binary interface

The ABI corresponds to Level 3 in the layered execution model.



Provides a program (via the OS):

- the hardware resources
- services available in a system

Application binary interface

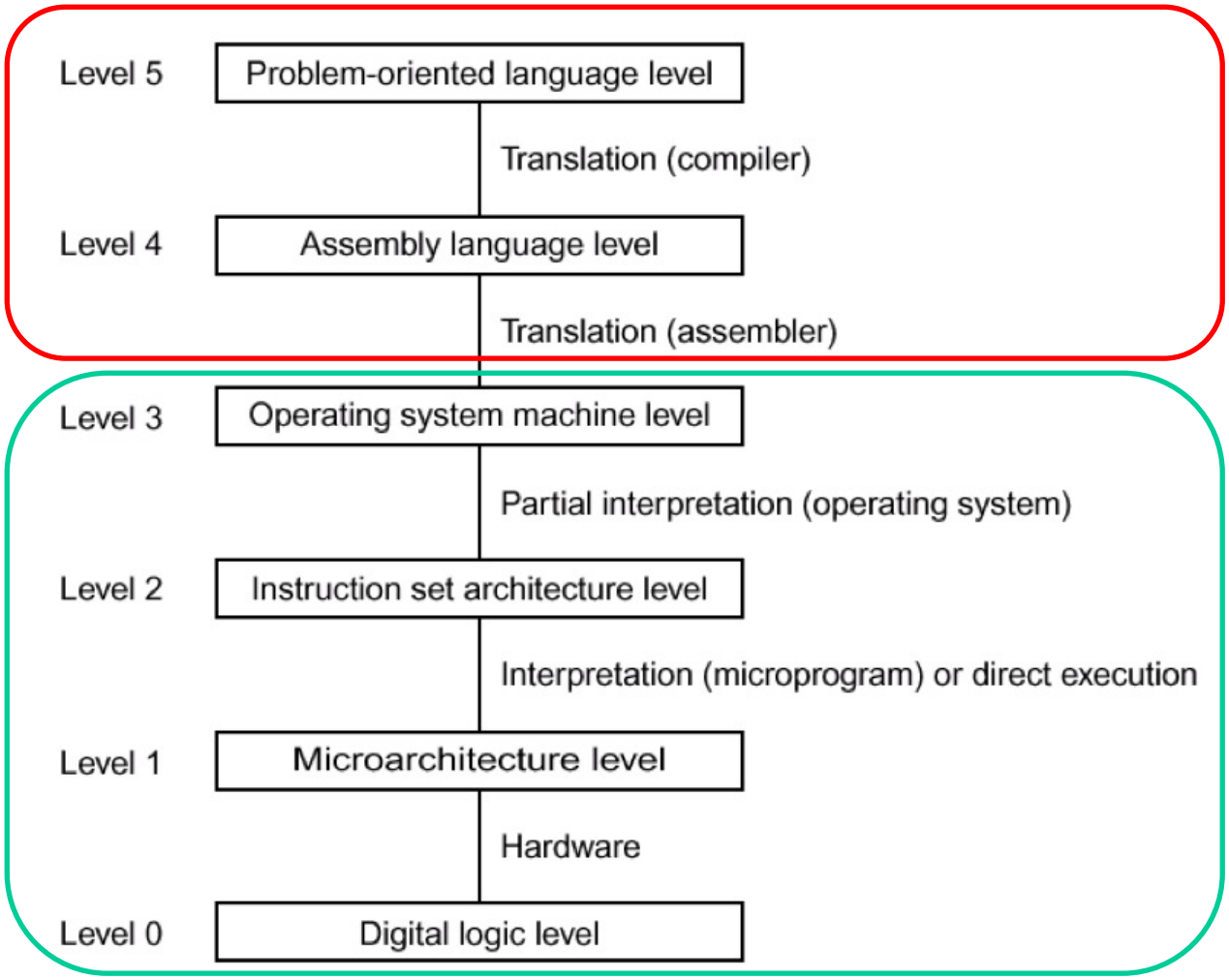
The ABI corresponds to Level 3 in the layered execution model.

- **User ISA**: aspects of the ISA that are visible to an application program (sum, multiplication,…, logical operations, branches, etc...).

- **System Calls**: calls that allow programs to interact with shared hardware resources indirectly by OS

Machine levels

One machine level can only run instructions that were meant for it



Machine Virtualization

A **Virtual Machine** (**VM**) is a logical abstraction able to provide a virtualized *execution environment*. *More specifically, a VM*:

- (provides) Identical Software behavior
- (consists in a) Combination of physical machine and virtualizing software
- (may appear as) Different resources than physical machine
- (may result in) Different level of performance

Its tasks are:

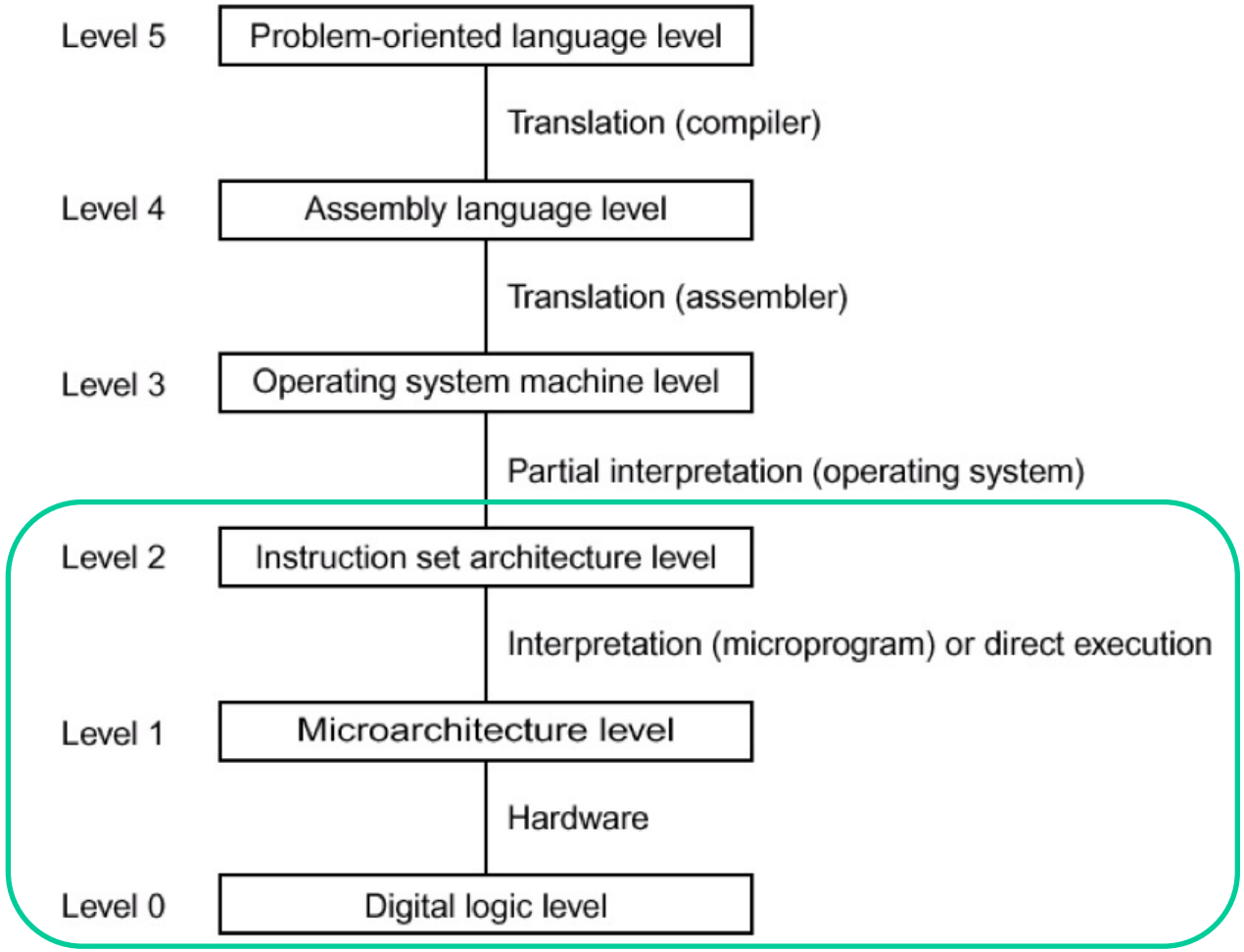
- To map virtual resources or states to corresponding physical ones
- To use physical machine instructions/calls to execute the virtual ones.

Two types of Virtual Machines:

- System VMs
- Process VMs

System VMs

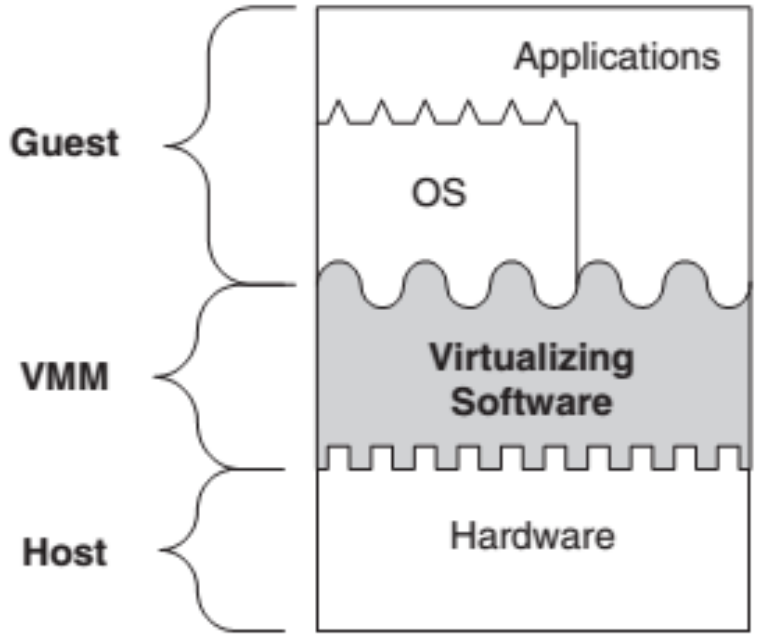
The VMM supports the levels 0-2 of the architecture



System Virtual Machine

System VMs

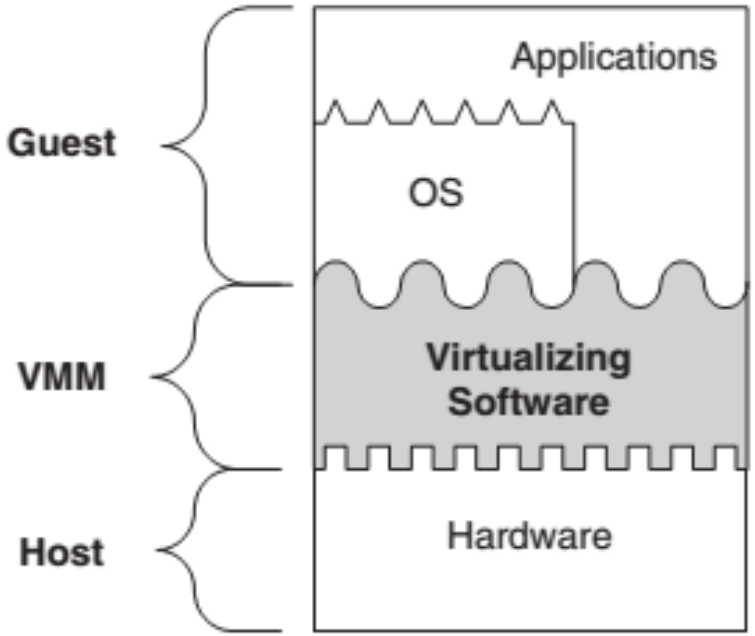
System Virtual Machine

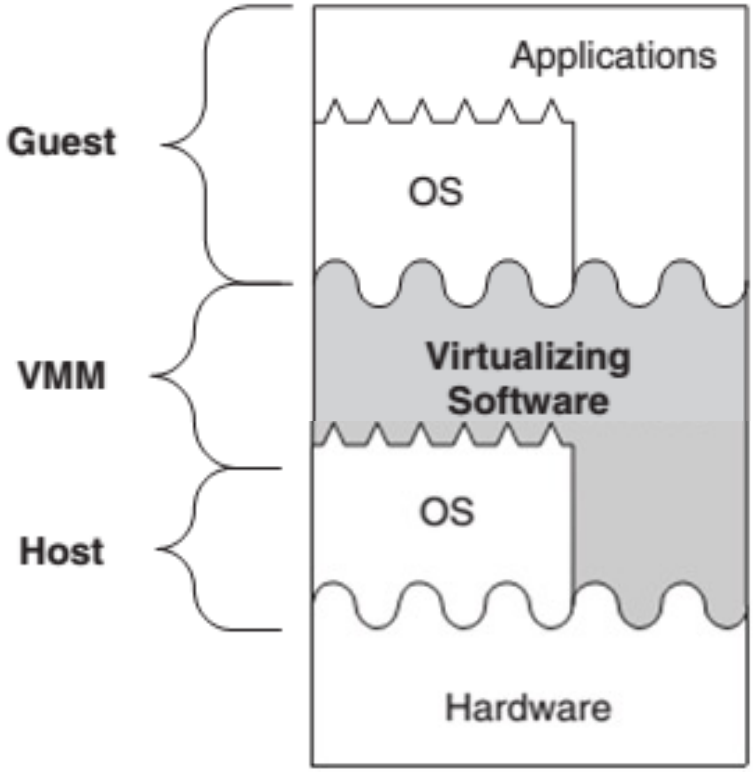


- **Provide a complete system environment** that can support an operating system (potentially with many user processes)
- It provides **operating system running in it access to underlying hardware resources** (networking, I/O, a GUI).
- The VM supports the operating system as long as the system environment is alive.
- Virtualizing software placed between hardware and software (**emulates the ISA interface seen by software**)
- The virtualization software is called **VMM (Virtual Machine Monitor)**

System VMs

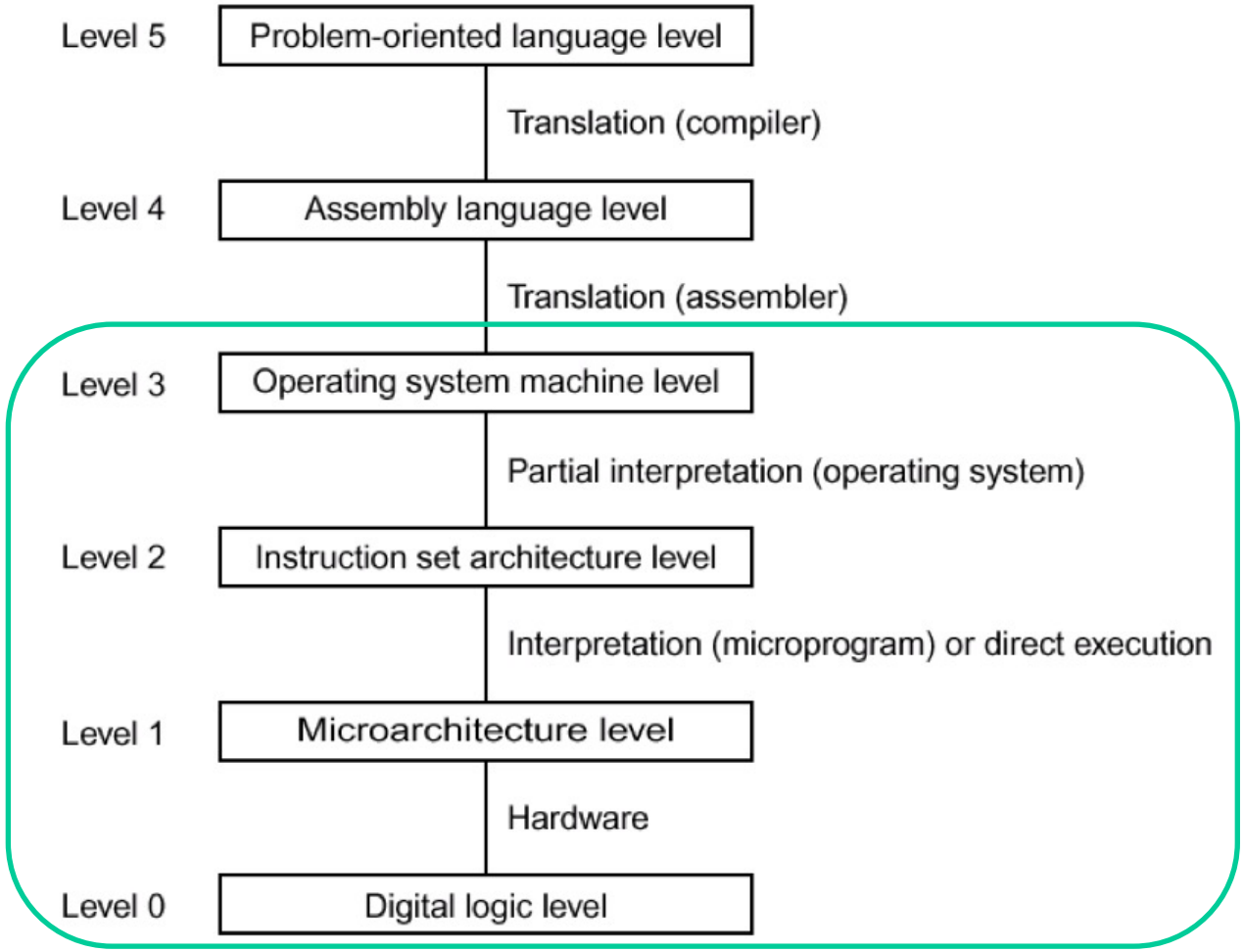
The VMM can provide its functionality either **working directly on the hardware**, or **running on another OS**.





Process VMs

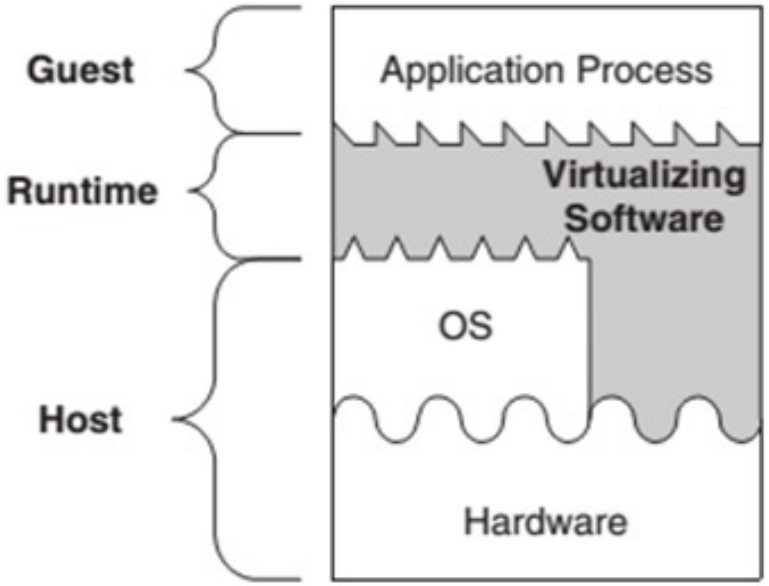
The runtime software supports the levels 0-3 of the architecture



Process Virtual Machine

Process VMs

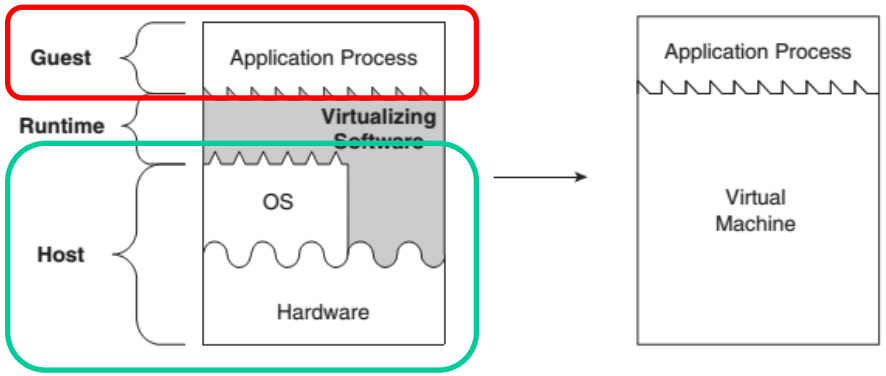
Process Virtual Machine



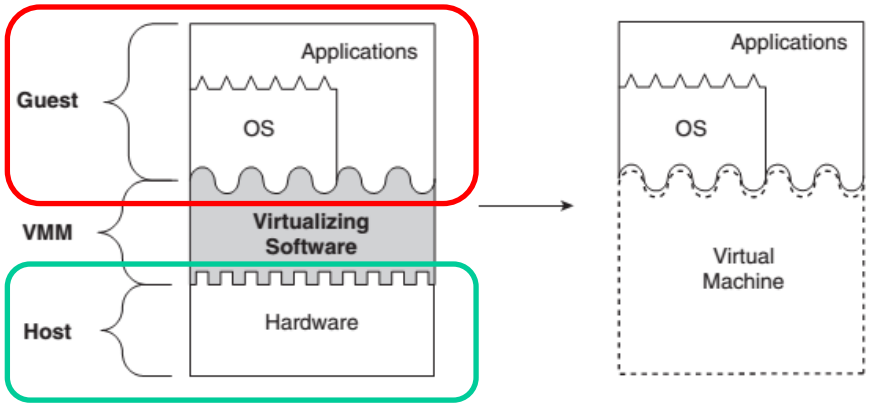
- Able to support an individual process
- The virtualizing software is placed at the ABI interface, on top of the OS/hardware combination.
- The virtualizing software emulates both user-level instructions and operating system calls.
- The virtualization software is usually called **Runtime Software.**

Terminology (the *host* and the *guest*)

Process Virtual Machine



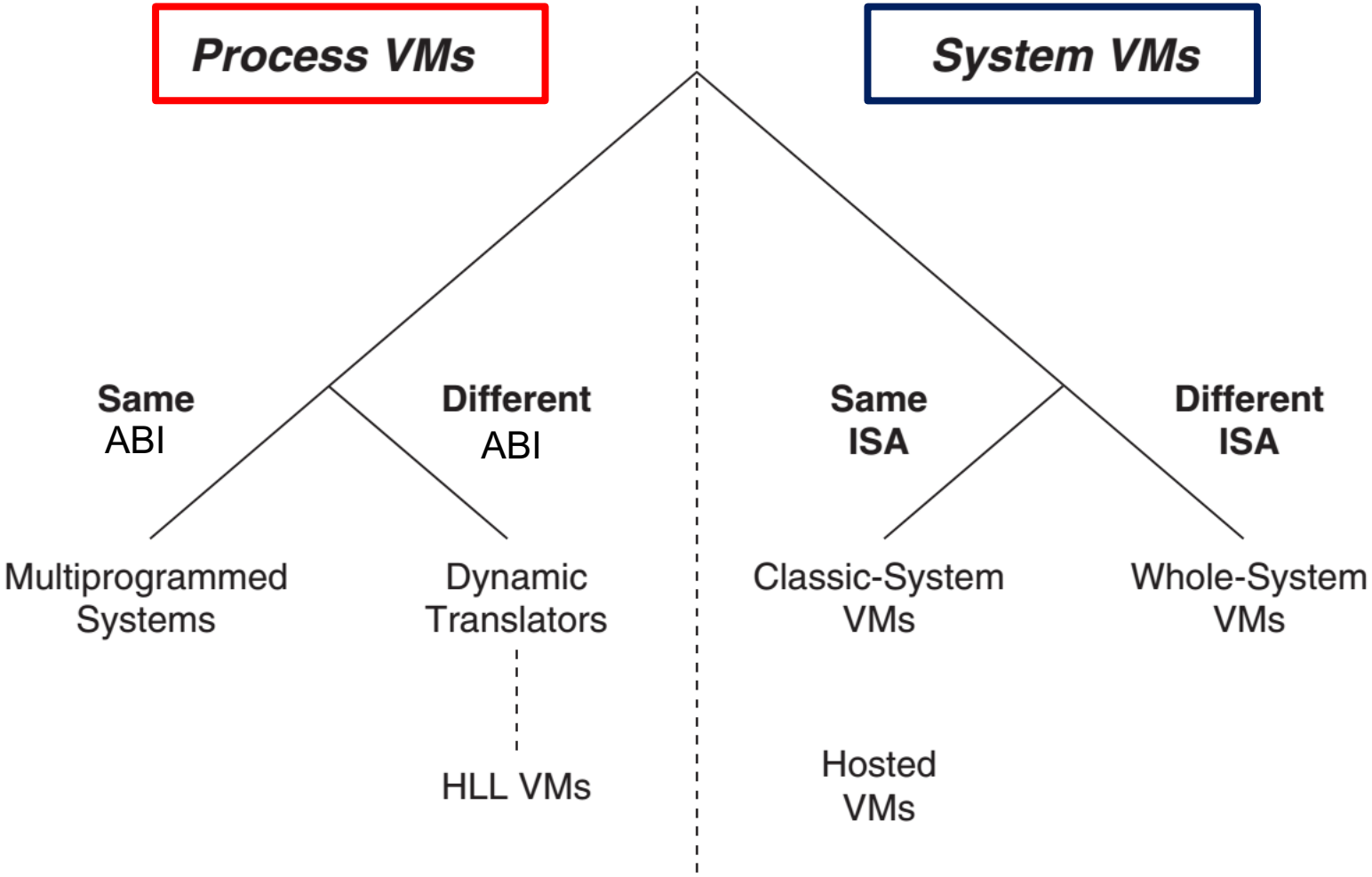
System Virtual Machine



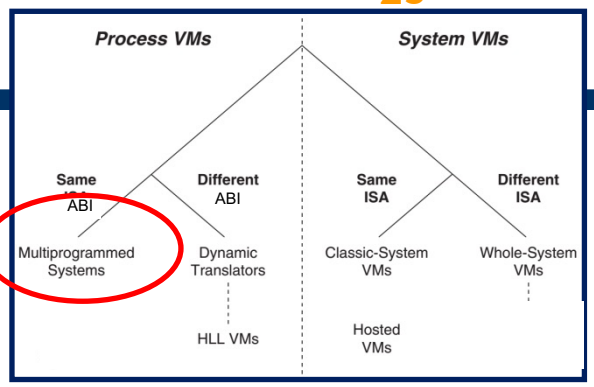
Host: the underlying platform supporting the environment/system

Guest: the software that runs in the VM environment as the guest.

Different types of Virtualization



Multiprogrammed systems

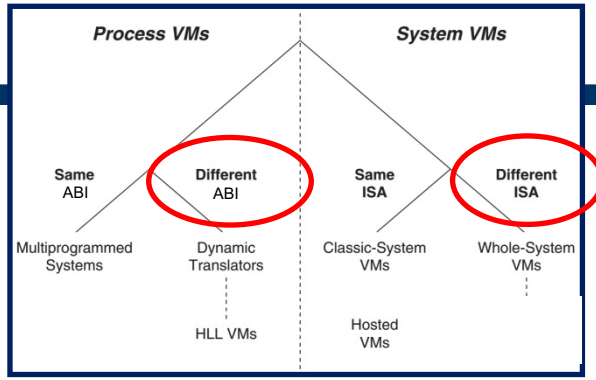


Same ABI / OS:

- VM formed by OS call interface + user ISA
- Common approach of all modern OS for **multi user** support
  - Task/Process Manager
- Each user process is given:
  - the illusion of having a complete machine to itself.
  - its own address space and is given access to a file structure
- OS timeshares and manages HW resources to permit this

- *Arguable is a real VM*

Emulation



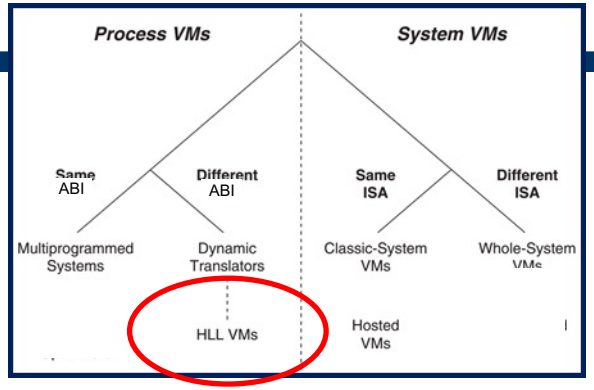
Emulation refers to those **software technologies developed to allow an application (or OS) to run in an environment different from that originally intended.**

It is required when the VMs have a different ISA / ABI from the architecture where they are running on.

Example:

- Obsolete hardware emulators
- Other architectures emulators

High-Level Language VM



Goal: to have an isolated execution environment for each application (or for different instances of the same application)

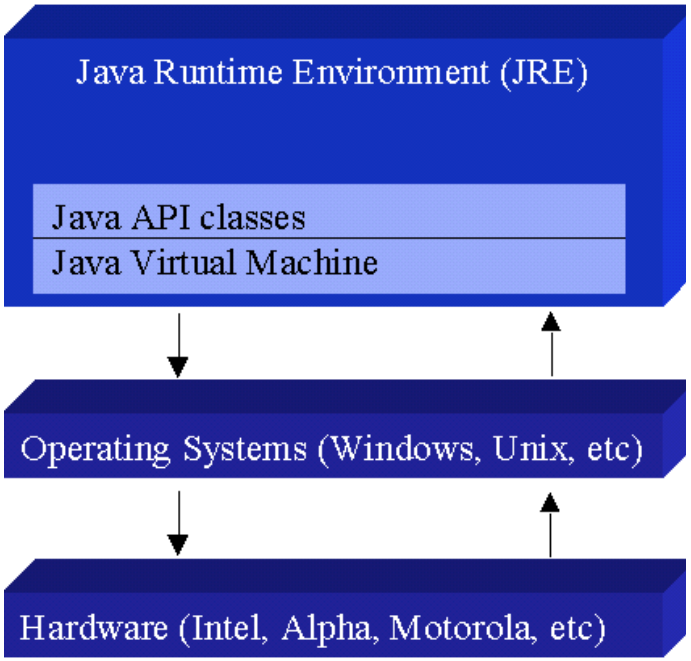
VM Task:

- Translates application byte code to OS-specific executable.
- Minimize HW/OS-specific feature for platform independence

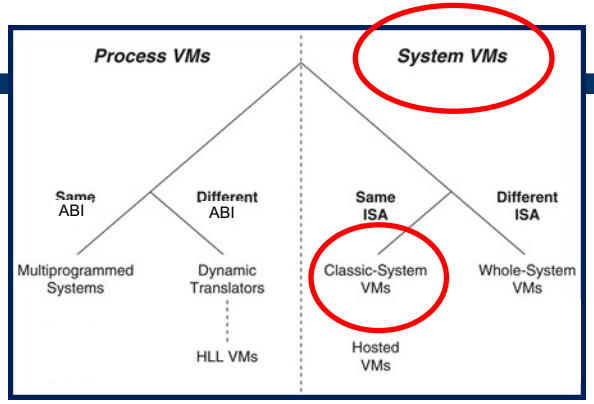
Applications run normally but

- Sandboxed
- Can "migrate"
- Are not conflicting one another
- Are not "installed" in a strict sense

Example: Java Virtual Machine

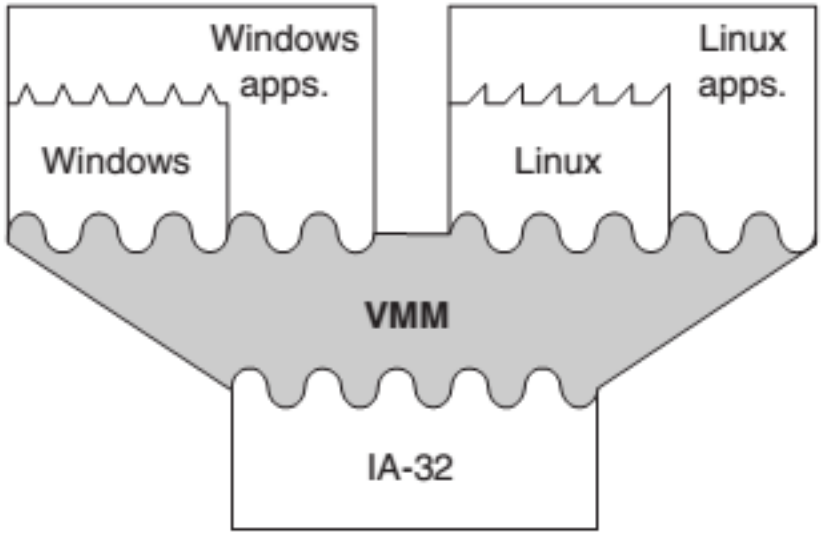


System VM

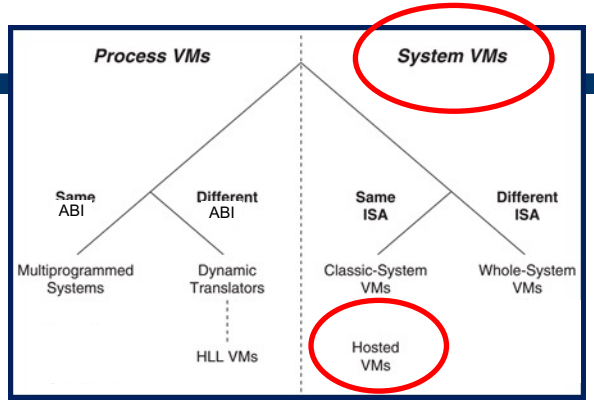


System VM for same ISA:

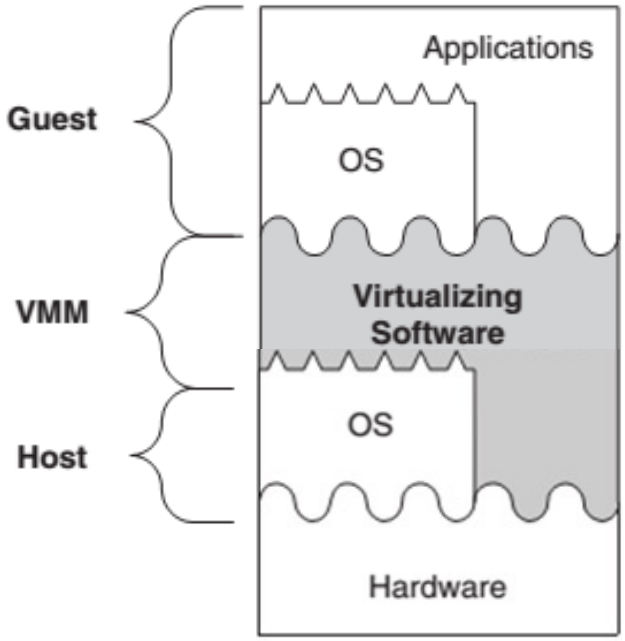
- The VMM is on bare hardware, and virtual machines fit on top
- The VMM can intercept guest OS's interaction with hardware resources
- The most efficient VM architecture (HW executes the same instructions of VMs)
- Two different OSs on the same on the same HW



System VM

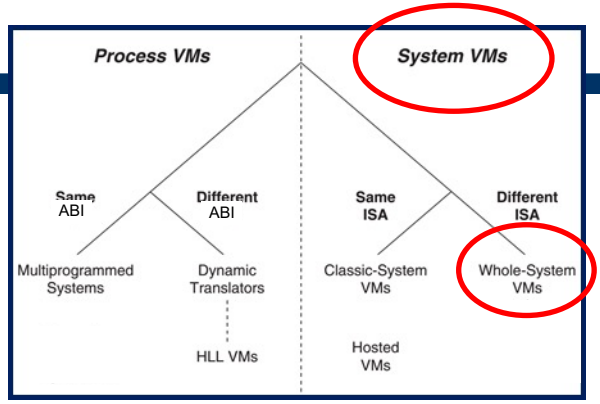


- **Hosted VM:** virtualizing software is on top of an existing host operating system.



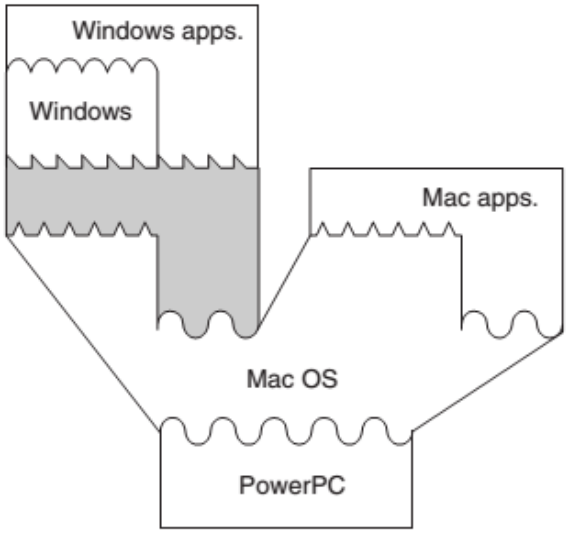
This will be the main type of VMs seen in this course!

System VM



Whole-system VMs:

- Virtualize all software: ISAs are different so both application and OS code require emulation, e.g., via binary translation. (*no native execution possible*)
- Usually implement the VMM and guest software on top of a conventional host OS running on the hardware.
- the VM software must emulate the entire hardware environment and all the guest ISA operations to equivalent OS call to the Host.
- Example: MS Word and Windows running on a Power PC (not x86 family)

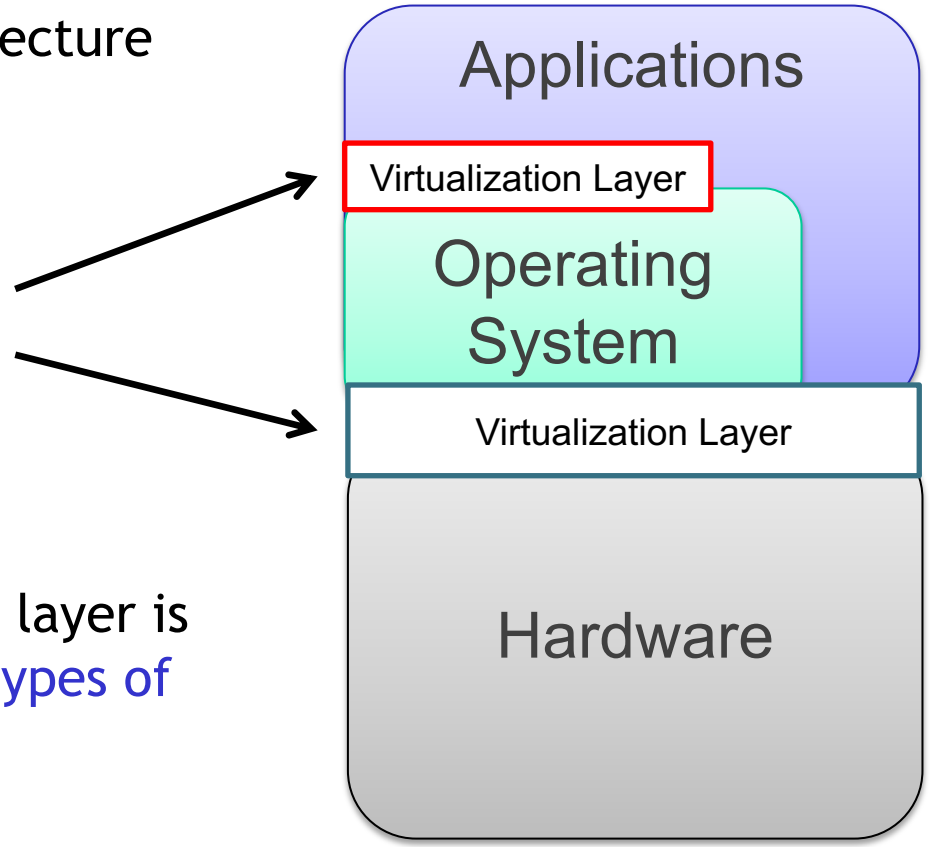


How is Virtualization Implemented?

How is Virtualization implemented?

Given a typical layered architecture of a system…

… by adding layers between execution stack layers.

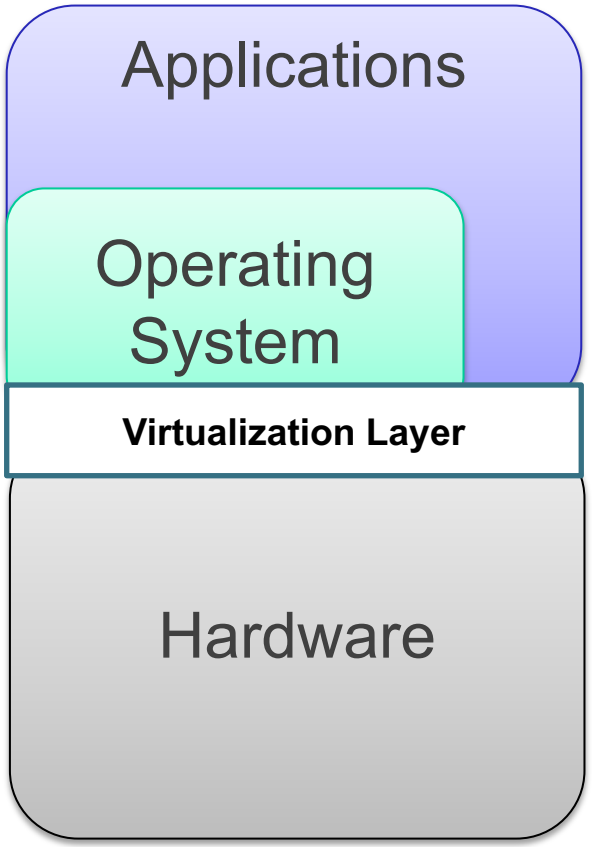


Depending on where the new layer is placed, we obtain different types of Virtualization.

How is Virtualization implemented?

Hardware-level virtualization:

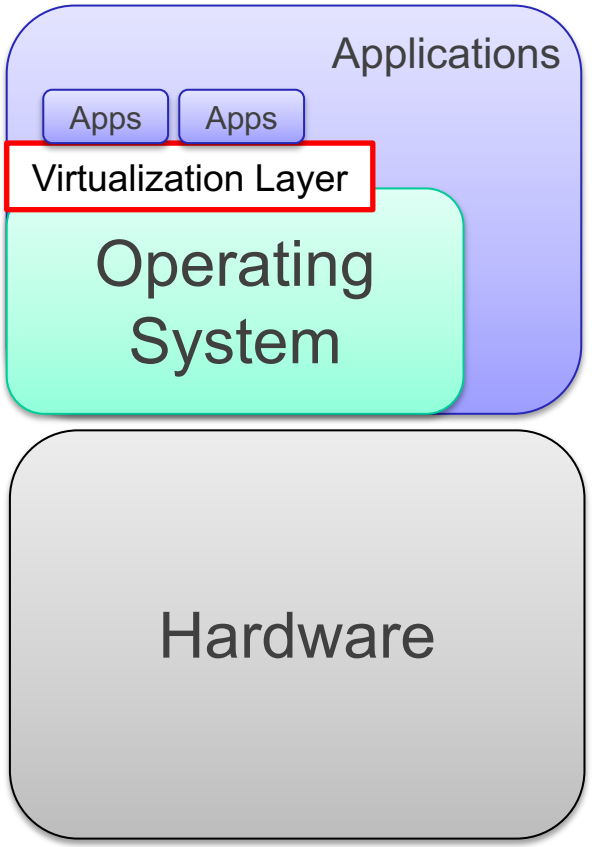
- Virtualization layer is placed between hardware and OS.
- The interface seen by OS and application might be different from the physical one



How is Virtualization implemented?

Application-level virtualization:

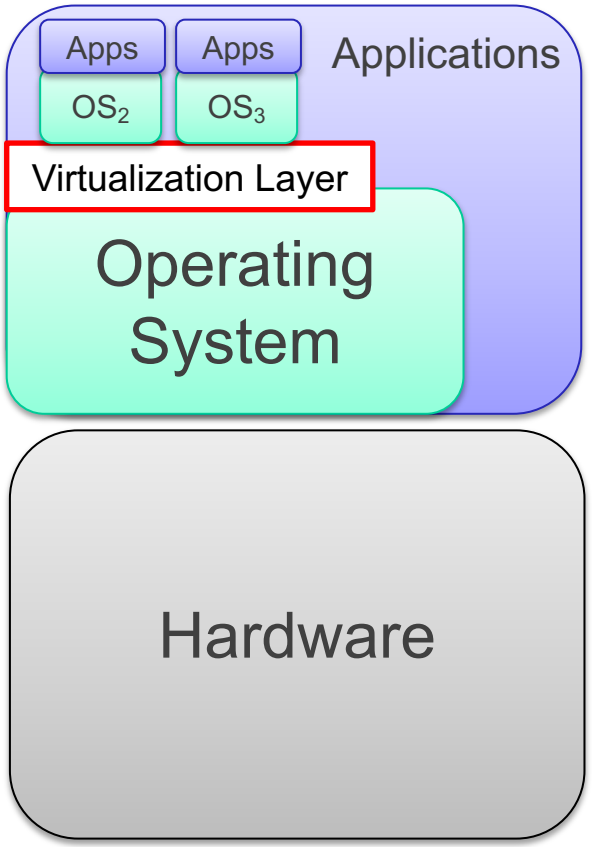
- A virtualization layer is placed between the OS and some applications
  - E.g.: JVM (Java Virtual Machine)
- Provides the same interface to the applications.
- Applications run in their environment, independently from OS



How is Virtualization implemented?

System-level virtualization:

- The virtualization layer provides the interface of a physical machine to a secondary OS and a set of application running in it, allowing them to run on top of an existing OS.
- Placed between the system's OS and other OS
  - E.g.: VMware Wks/Player, VirtualBox
- Enable several OSs to run on a single HW



Virtualization technologies: properties

Partitioning

- Execution of multiple OSs on a single physical machine
- Partitioning of resources between the different VMs

Isolation

- Fault tolerance and security (as at the hardware level)
- Advanced resource control to guarantee performance (managed by the *hypervisor*)

Encapsulation

- The entire state of a VM can be saved in a file (e.g., freeze and restart the execution)
- Because a VM is a file, can be copied/moved as a file

HW-independence

- Provisioning/migration of a given VM on a given physical server.

Virtual Machine Managers (System VMs – Same ISA)

Virtual Machine Manager

An application that:

- manages the virtual machines
- mediates access to the hardware resources on the physical host system
- intercepts and handles any privileged or protected instructions issued by the virtual machines

- This type of virtualization typically runs virtual machines whose operating system, libraries, and utilities have been compiled for the same type of processor and instruction set as the physical machine on which the virtual systems are running.
- Crucial to support Cloud Computing

VMM and Hypervisor

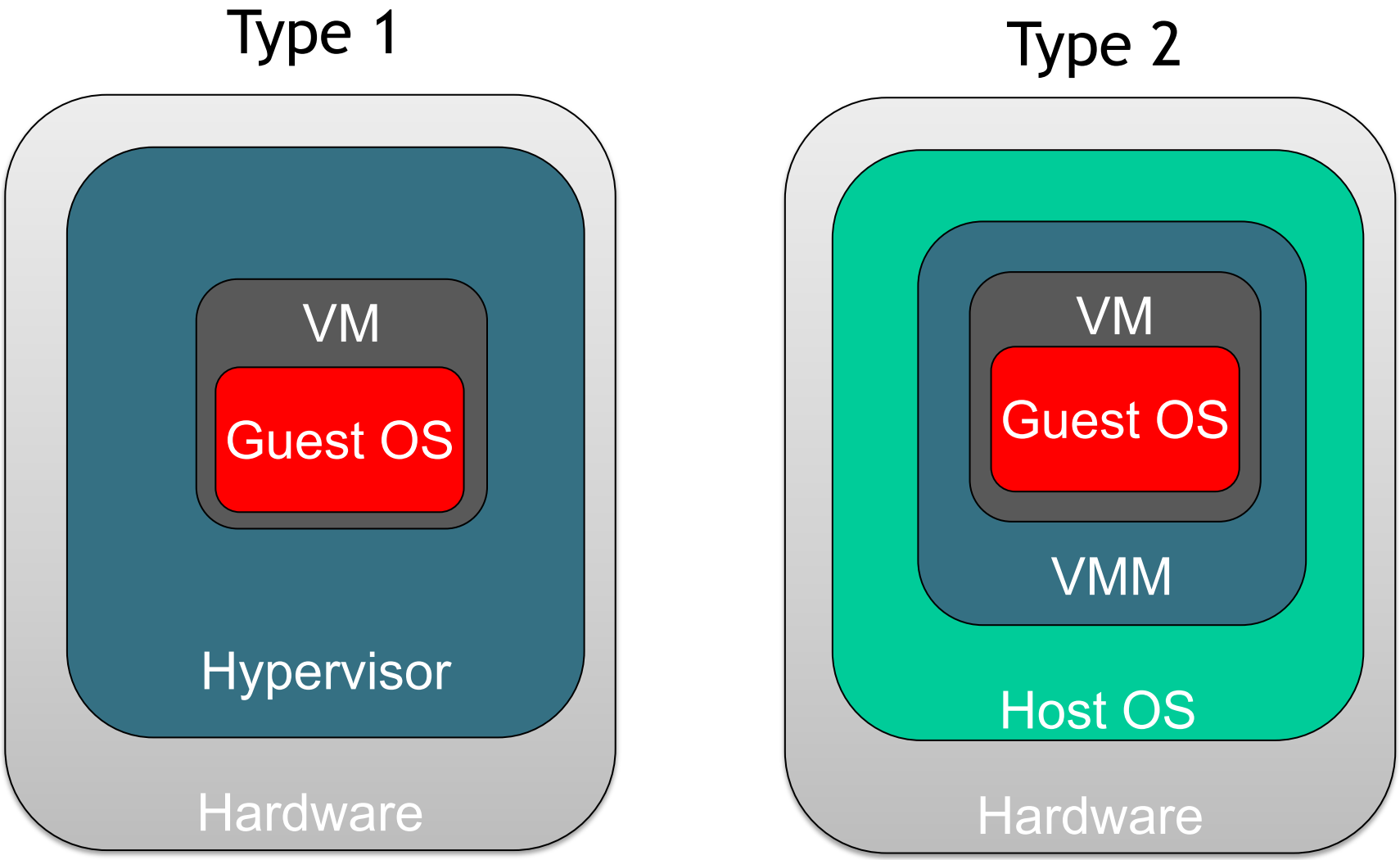
Three terms are used to identify the same thing:

- Virtual Machine Manager
- Virtual Machine Monitor
- Hypervisor

Some authors gives slightly different meanings to the three:

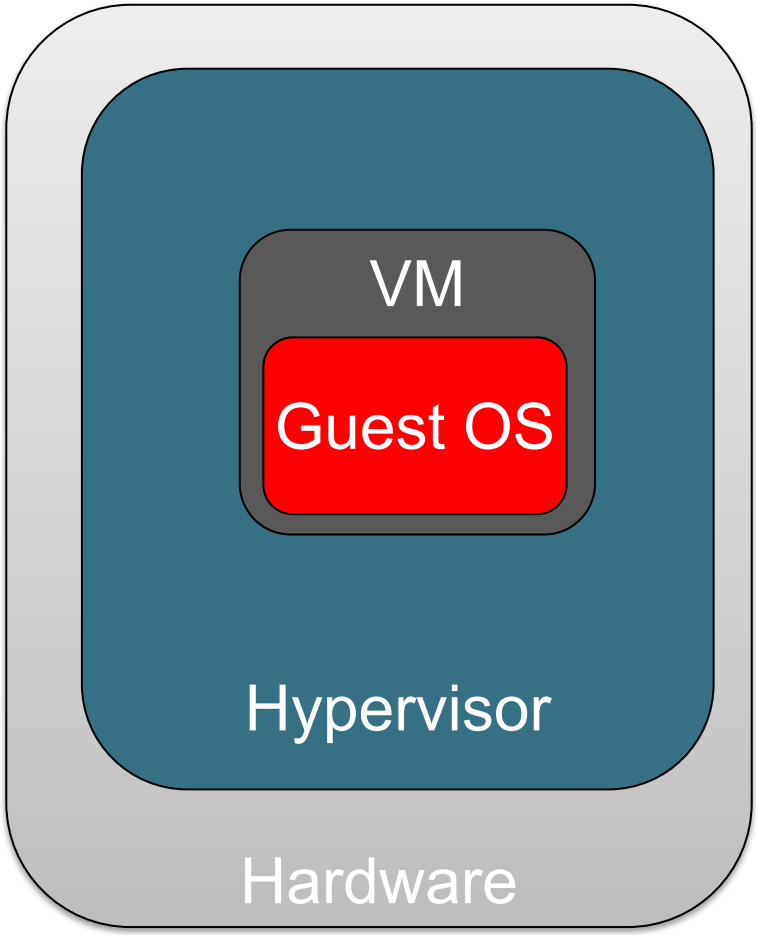
- *Virtual Machine Monitor*:
  - The virtualization software
- *Hypervisor*:
  - A virtualization software that runs directly on the hardware
- *Virtual Machine Manager*:
  - A VMM or Hypervisor that is also used to create, configure and maintain virtualized resources.
  - It provides a user-friendly interface to the underlying virtualization software

Hypervisor types



Hypervisor types

Type 1



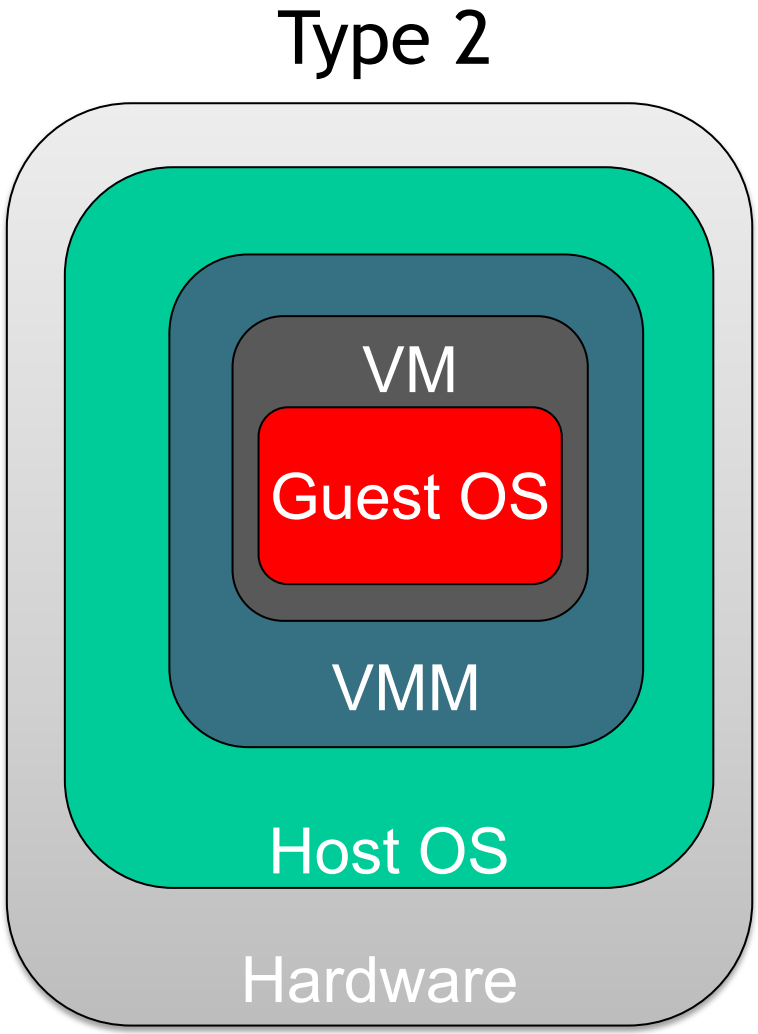
Bare-metal or type 1 hypervisors

- Takes direct control of the hardware

Hypervisor types

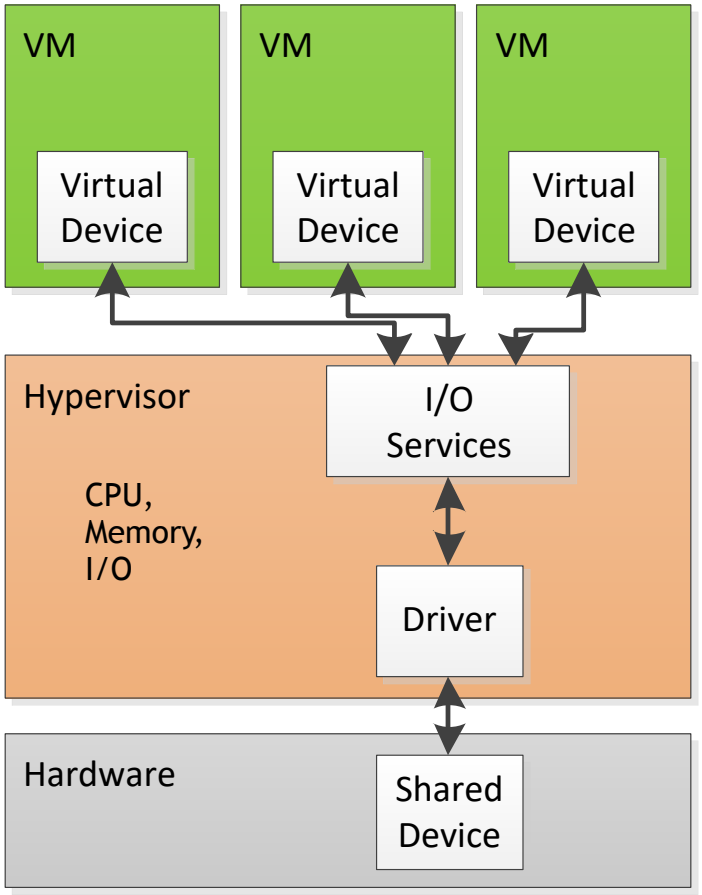
Hosted or type 2 hypervisors (VMM)

- Reside within a host operating system
- Leverage code of the host OS
  - E.g. CPU scheduling, memory management

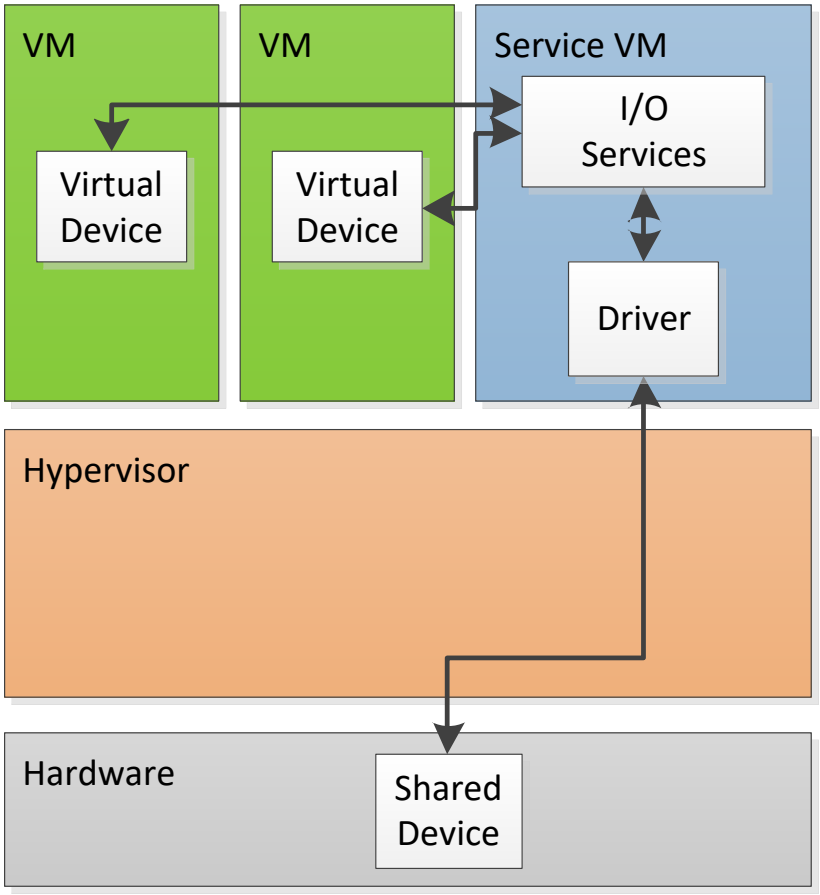


Type 1 hypervisor architecture

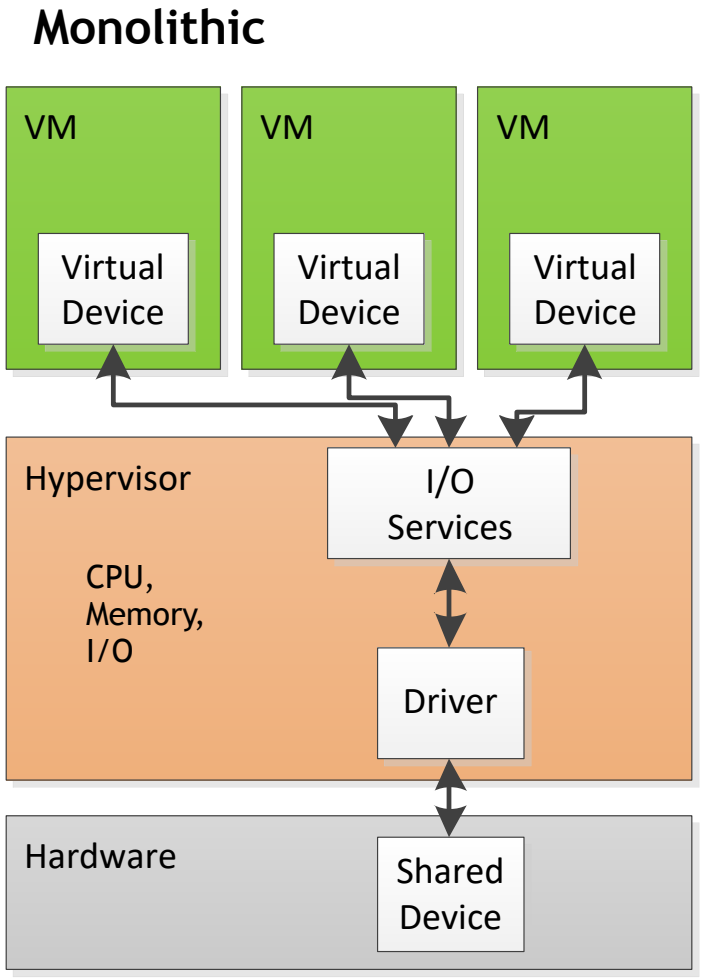
Monolithic



Microkernel



Type 1 hypervisor architecture: Monolithic

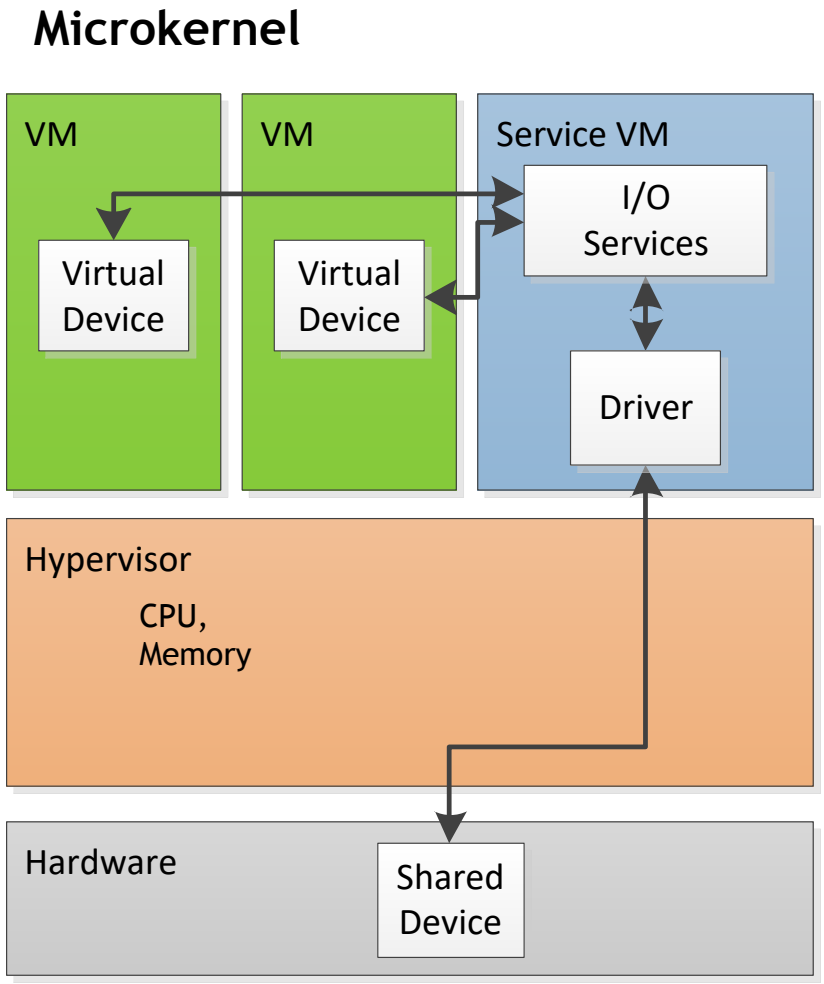


- Device drivers run within the hypervisor
  + Better performance
  + Better performance isolation
  - Can run only on hardware for which the hypervisor has drivers

Type 1 hypervisor architecture

Device drivers run within a service VM

+ Smaller hypervisor
+ Leverages driver ecosystem of an existing OS
+ Can use 3rd party driver (even if not always easy, recompiling might be required)



Hypervisor Type I - Hacks

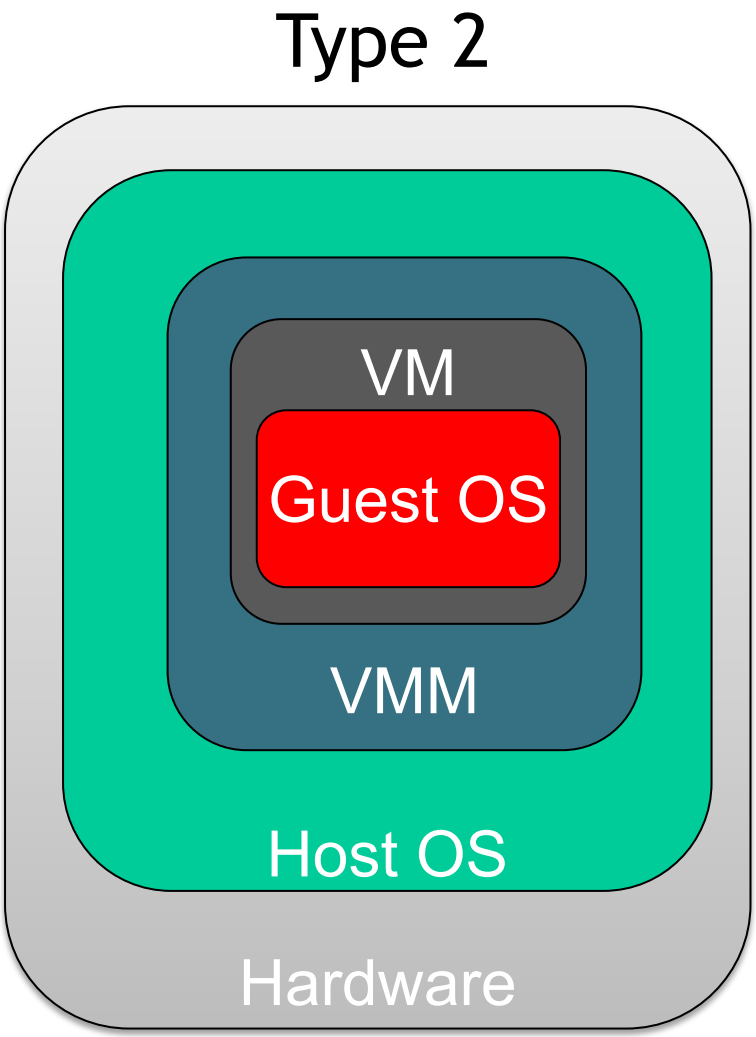
Sometimes Type I Hypervisor can be used to provide hacks to some OSs.

Two well-known examples on PC hardware:

1) Windows 7 genuine without an official license: the Hypervisor mimics vendor specific hardware on different machines to allow the installation of pre-authorized S.O. copies (e.g., Pre-activated version of Windows on HP machines);

2) Mac OS on not-Apple hardware: the Hypervisor emulates the EFI (Extended Firmware Interface) of an Apple hardware on a standard PC.

Type 2 hypervisor architecture



They are characterized by at least two OSs running on the same hardware:

- The *Host OS* controls the hardware of the system
- The *Guest OS* is the one the runs in the VM.

The *VMM* runs in the *Host OS*, while *applications* run in the *Guest OS*.

Type 2 hypervisor architecture

Type 2 hypervisors

- There is a Host OS that manages the hardware where the VMM runs.
- + More flexible in terms of underlying hardware.
- + Simpler to manage and configure (VMM can use the Host OS to provide GUI, not only BIOS).
- - Special care must be taken to avoid conflict between Host OS and Guest OS (e.g., Virtual Memory).
- - The Host OS might consume a non-negligible set of physical resources (e.g., 1 core for the Host OS).

Virtualization techniques

System-Level Virtualization Techniques

Different ways to implement (system level / same ISA) virtualization:

Paravirtualization

Full Virtualization

Paravirtualization

Guest OS and VMM collaborate:

- VMM present to VMs an interface similar but not identical to that of the underlying hardware
- Goal: reduce guest's executions of tasks too expensive for the virtualized environment ("hooks" allow the guest(s) and host to request and acknowledge tasks which would otherwise be executed in the virtual domain, where execution performance is worse).

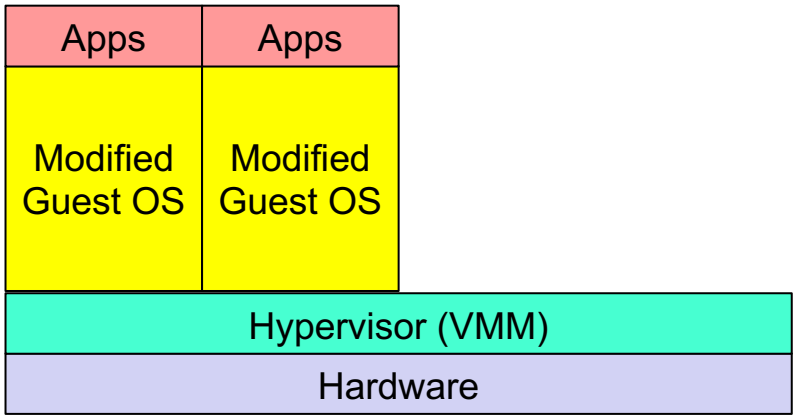
Pros:

- Simpler VMM
- High Performance

Cons:

- Modified Guest OS

(Cannot be used with traditional OSs, e.g., available on some Linux Releases)



Full Virtualization

Provides a **complete simulation of the underlying hardware**.

- the full instruction set
- input/output operations
- Interrupts
- memory access

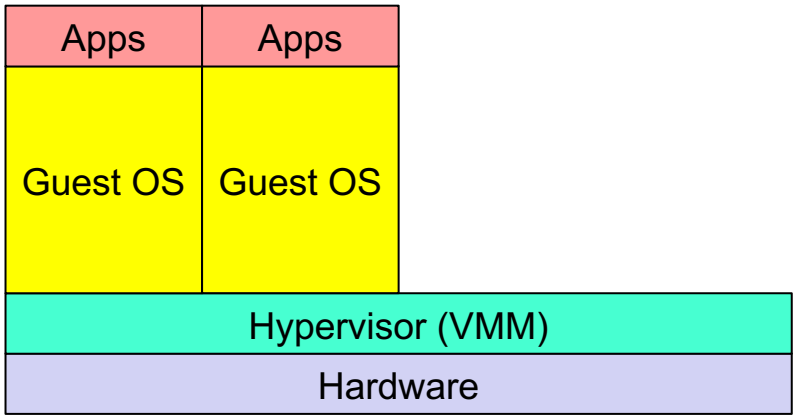
Some protected instructions are trapped and handled by Hypervisor

Pros:

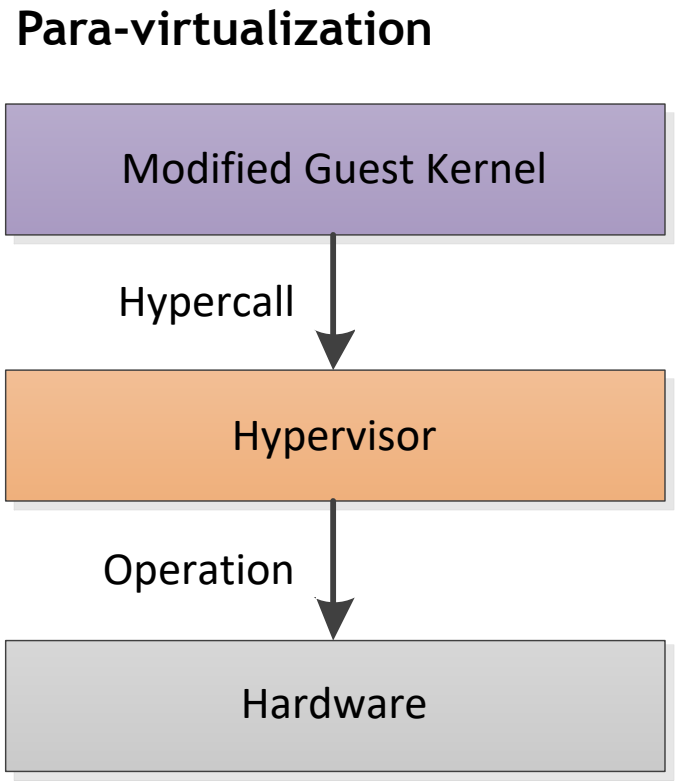
- Running unmodified OS

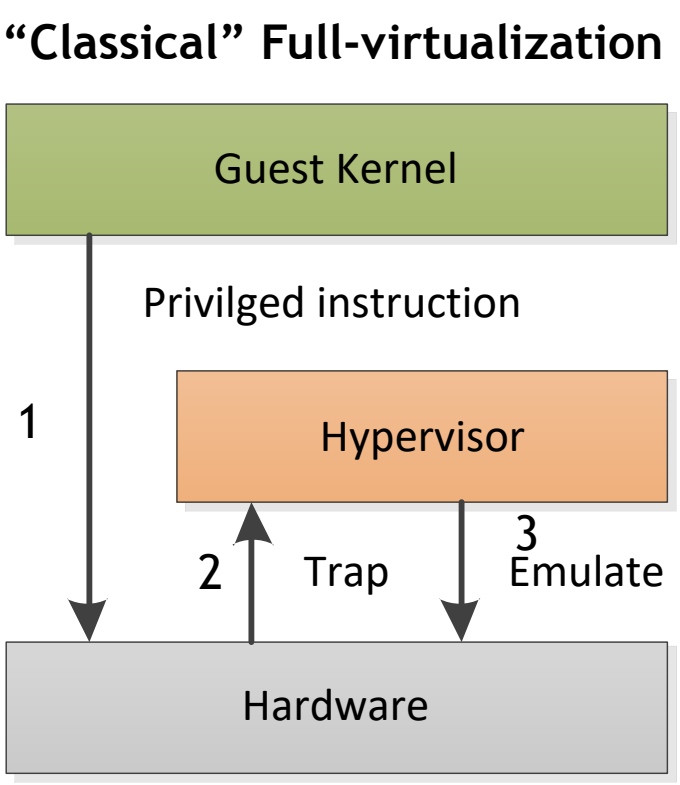
Cons:

- Hypervisor mediation
  (to allow the guest(s) and host to request and acknowledge tasks which would otherwise be executed in the virtual domain)
- Not on every architecture
  (requires some hardware support)



Para VS Full-virtualization





Paravirtualize some devices even in Full Virtualization,
e.g., Guest additions in VMWare or Virtual Box (drivers are replaced)

Containers

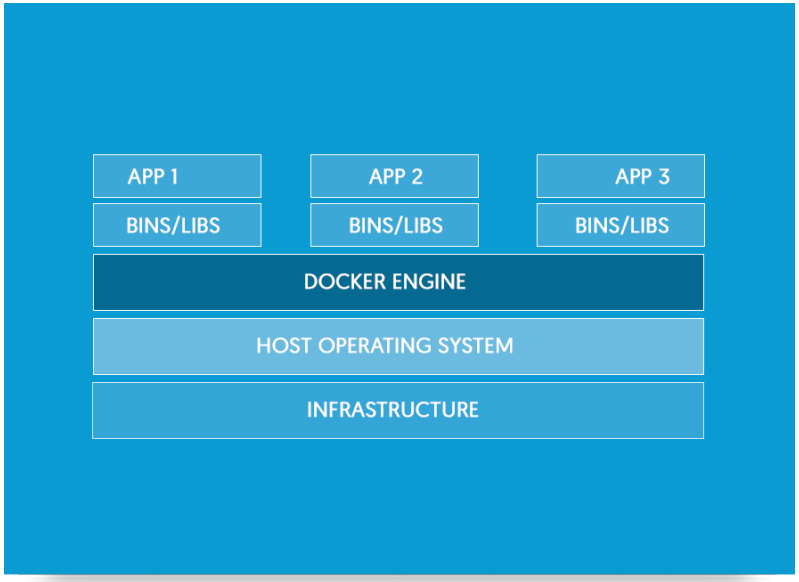
What is a container?

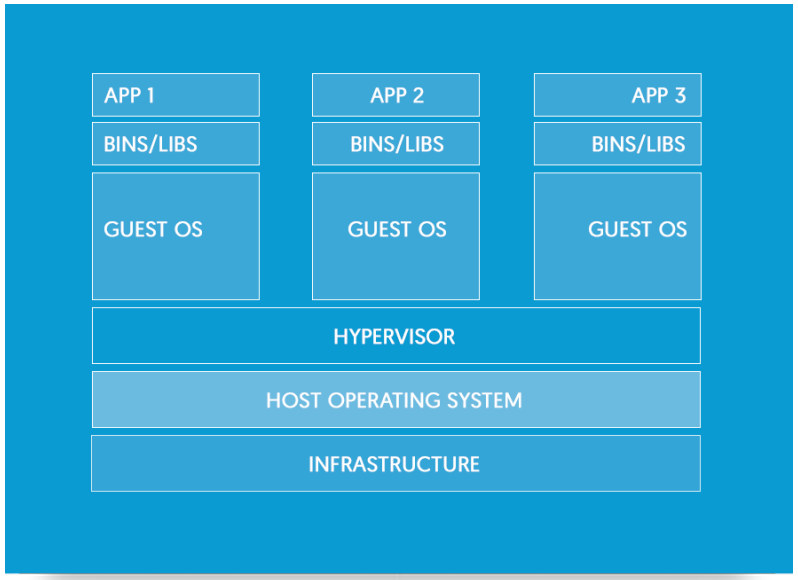
- **Containers** are **pre-configured packages**, with everything you need to execute the code (code, libraries, variables and configurations) in the target machine.
- The **main advantage of containers is that their behavior is predictable, repeatable and immutable:**
  - When I create a "master" container and duplicate it on another server, **I know exactly how it will be executed**
  - There are no unexpected errors when moving it to a new machine or between environments
- **Example: a container for a website**,
  - We are not required to export/import the development/test/production environments,
  - We create a container that contains the site and bring it to the destination environment

Containers and Virtual Machine

- VM provides hardware virtualization, while containers provide virtualization at the operating system level
- The main difference is that the containers share the host system kernel with other containers.

Containers
applications are packaged with all of their dependencies into a standardized unit for software development/deployment.





VMMs
applications depend on guest OS.

Characteristics of containers

Containers are:

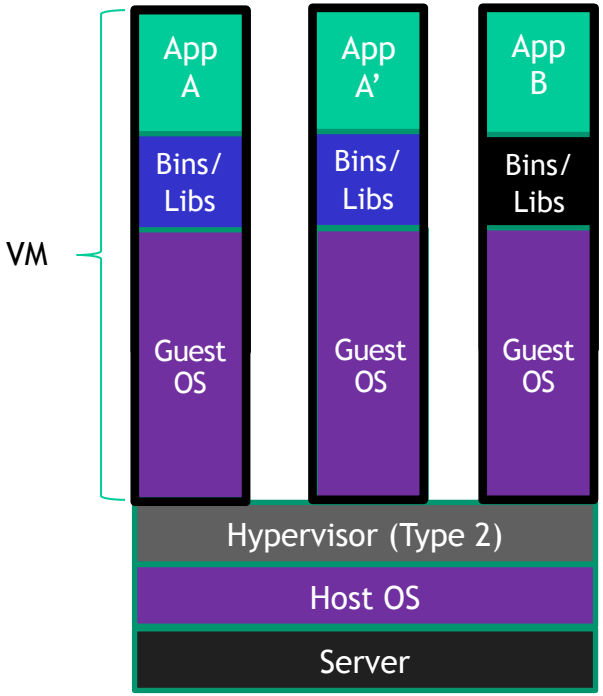
- Flexible: even the most complex applications can be containerized;
- Light: the containers exploit and share the host kernel;
- Interchangeable: updates and updates can be distributed on the fly;
- Portable: you can create locally, deploy in the Cloud and run anywhere;
- Scalable: it is possible to automatically increase and distribute replicas of the container;
- Stackable: containers can be stacked vertically and on the fly.

Containers ease the deployment of applications and increase the scalability but they also impose **a modular application development where the modules are independent and uncoupled**.

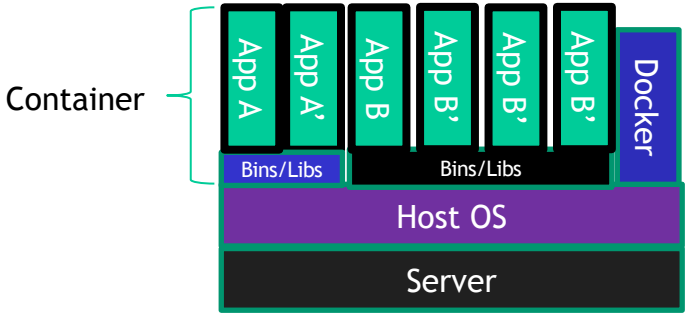
What can you use Containers for?

- Helping make your local development and build workflow faster, more efficient, and more lightweight.
- Running stand-alone services and applications consistently across multiple environments.
- Using Container to create isolated instances to run tests.
- Building and testing complex applications and architectures on a local host prior to deployment into a production environment.
- Building a multi-user Platform-as-a-Service (PAAS) infrastructure.
- Providing lightweight stand-alone sandbox environments for developing, testing, and teaching technologies, such as the Unix shell or a programming language.
- Software as a Service applications.

Container based IaaS - Containers vs. VMs



Containers are isolated, but share OS and, where appropriate, bins/libraries



Software for “Containers”