DSC516: Cloud Computing

Part II: Cloud Building Blocks

Module 4: Virtualization, Containers and Resource Management

University of Cyprus
Department of Computer Science

Lecture 7

Virtualization Fundamentals

University of Cyprus
Department of Computer Science

Learning Objectives

- Examine, understand, assess, and describe the concepts of virtualization, virtual machines, and virtual machine monitors hypervisors.
- Understand and explain basic Operating Systems' concepts of relevance to virtualization: core abstractions, layering, libraries, application binary interface, security and privilege management, protection rings, running in kernel vs. user mode.
- Understand and explain different virtualization types and the concepts of server virtualization, virtual machines, and hypervisors.
- Understand and explain techniques for implementing virtual machines: de-privileging, primary and shadow structures, memory tracing.
- Explain the key features of VMM
- Understand and explain alternative techniques for implementing hypervisors

Readings

Required Readings

- **"Beyond server consolidation."** Vogels, W. In Queue (Vol. 6, Issue 1, pp. 20–26). https://doi.org/10.1145/1348583.1348590 (2008).
- **"The architecture of virtual machines,"** J. E. Smith and Ravi Nair, in Computer, vol. 38, no. 5, pp. 32-38 (May 2005) doi: 10.1109/MC.2005.173.
- "**Understanding Full Virtualization, Paravirtualization, and Hardware Assist,"** VMWare White Paper (2008)
- **Chapters 4.10, 10.2, 10.3, "Cloud Computing: Theory and Practice,"** Dan Marinescu (2017)
- **"Understanding virtualization,"** RedHat (2018), **https://www.redhat.com/en/topics/virtualization/what-is-virtualization**

Additional Readings

- **"A comparison of software and hardware techniques for x86 virtualization,"** Adams and Agesen, (2006) ACM SIGPLAN Notices, vol. 41, issue 11. **https://doi.org/10.1145/1168918.1168860**
- **"Xen and the art of virtualization,"** P. Barham et al., Proc. Nineteenth ACM Symp. Oper. Syst. Princ. - SOSP '03, p. 164, 2003.

Virtualization: Definition

- Refers to the act of creating a virtual (rather than actual) version of some computing resource
- Virtualization:
  - abstracts the underlying resources;
  - simplifies their use;
  - isolates users from one another; and
  - supports replication which increases the elasticity of a system

Virtualization Fundamentals

Introduction and Basic Concepts

University of Cyprus
Department of Computer Science

HOW DO WE TACKLE THE INCREDIBLE COMPLEXITY OF COMPUTER SYSTEMS TO ALLOW FOR THEIR DEVELOPMENT, DEBUGGING AND EVOLUTION?

Introduction and Basic Concepts

Interfaces, Abstraction, Layering

University of Cyprus
Department of Computer Science

"The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise."

Edsger Dijkstra

Interfaces

- Well-defined interfaces:
  - facilitate independent subsystem development of both hardware and software
  - permit development of interacting computer subsystems in different organizations and at different times

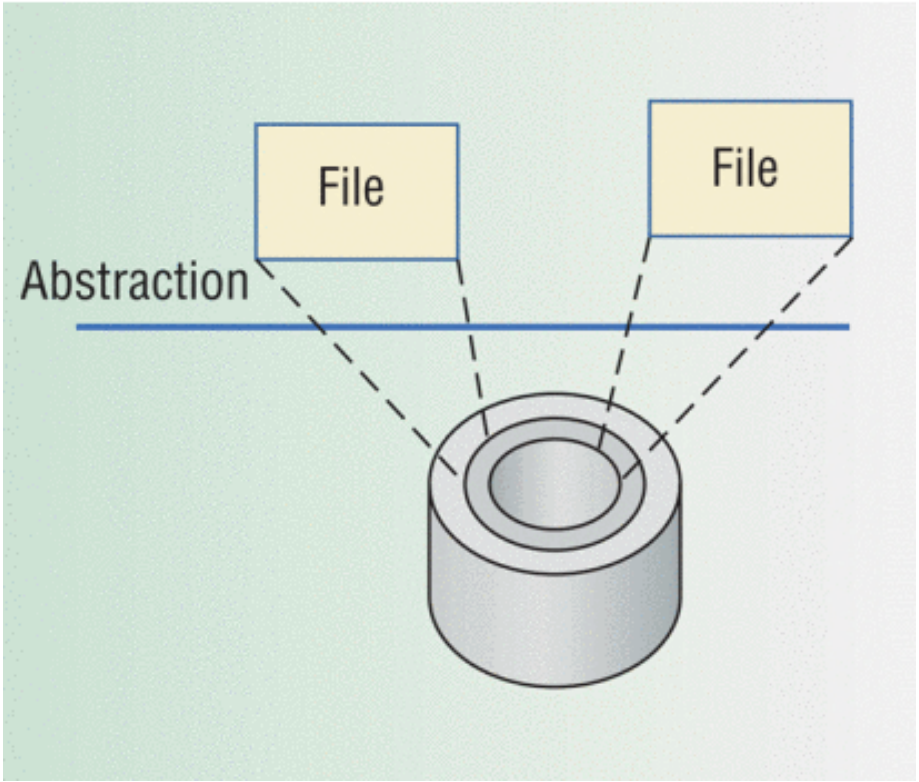
Architecture & Implementation

- **Architecture**, as applied to computer systems, refers to a formal specification of an interface in the system, including the logical behavior of resources managed via the interface.
- **Implementation** describes the actual embodiment of an architecture.
- **Abstraction levels** correspond to implementation layers, whether in hardware or software, each associated with its own interface or architecture.

Layering

- A common approach to manage system complexity:
  - Minimizes interactions among subsystems of a complex system
  - With layering we are able to design, implement, and modify individual subsystems independently.

Abstraction Example



- Operating system abstracts hard-disk addressing details (sectors, tracks) so that the disk appears to application software as a set of variable-sized files.
- Application programmers can then create, write, and read files without knowing the hard disk's construction and physical organization.

Interface Example: ISA

- Intel and AMD designers develop microprocessors that implement the Intel IA-32 (×86) instruction set architecture (ISA)
- Microsoft developers write software that is compiled to the same instruction set.
- Because both groups satisfy the ISA specification, the software can be expected to execute correctly on any PC built with an IA-32 microprocessor.

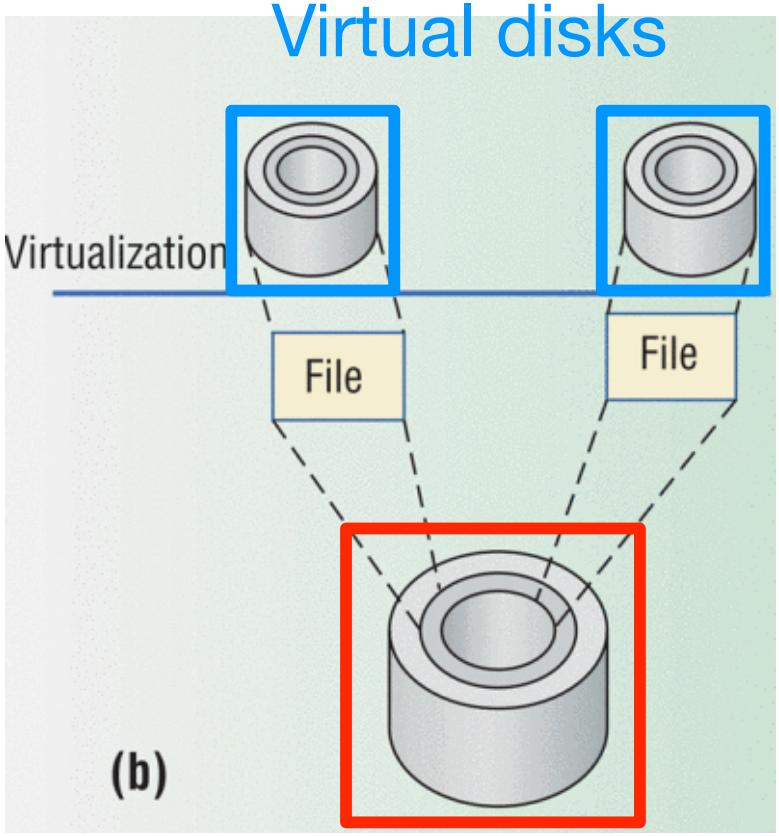
Interfaces' Limitations

- Subsystems and components designed based on specifications for one interface will not work with those designed for another.
  - For example: application programs distributed as compiled binaries, are tied to a specific ISA and depend on a specific operating system interface.
- Lack of interoperability can be confining, especially in a world of networked computers where it is advantageous to move software as freely as data.

Virtualization

- Virtualizing a system or component (processor, memory, or I/O device) at a given abstraction level:
  - maps its interface and visible resources onto the interface and resources of an underlying, possibly different, real system.
  - Makes the real system appear as a different virtual system or even as multiple virtual systems.

Virtualization vs Abstraction



Real disk

- Virtualization transforms a single large disk into two smaller virtual disks.
- Virtualizing software uses the file abstraction as an intermediate step to provide a mapping between the virtual and real disks.
- A write to a virtual disk is converted to a file write, which is converted to a real disk write.

"The architecture of virtual machines," J. E. Smith and Ravi Nair, in Computer, vol. 38, no. 5, pp. 32-38, May 2005, doi: 10.1109/MC.2005.173.

Virtualization: remarks

- Provides a way of getting around interoperability constraints of different interfaces.
- Does not necessarily aim to simplify or hide implementation details. E.g.:
  - In the previous example, the level of detail provided at the virtual disk interface—the sector/track addressing—is no different from that for a real disk; no abstraction takes place.
- Virtual Machines: Virtualization applied to an entire machine.

Introduction and Basic Concepts

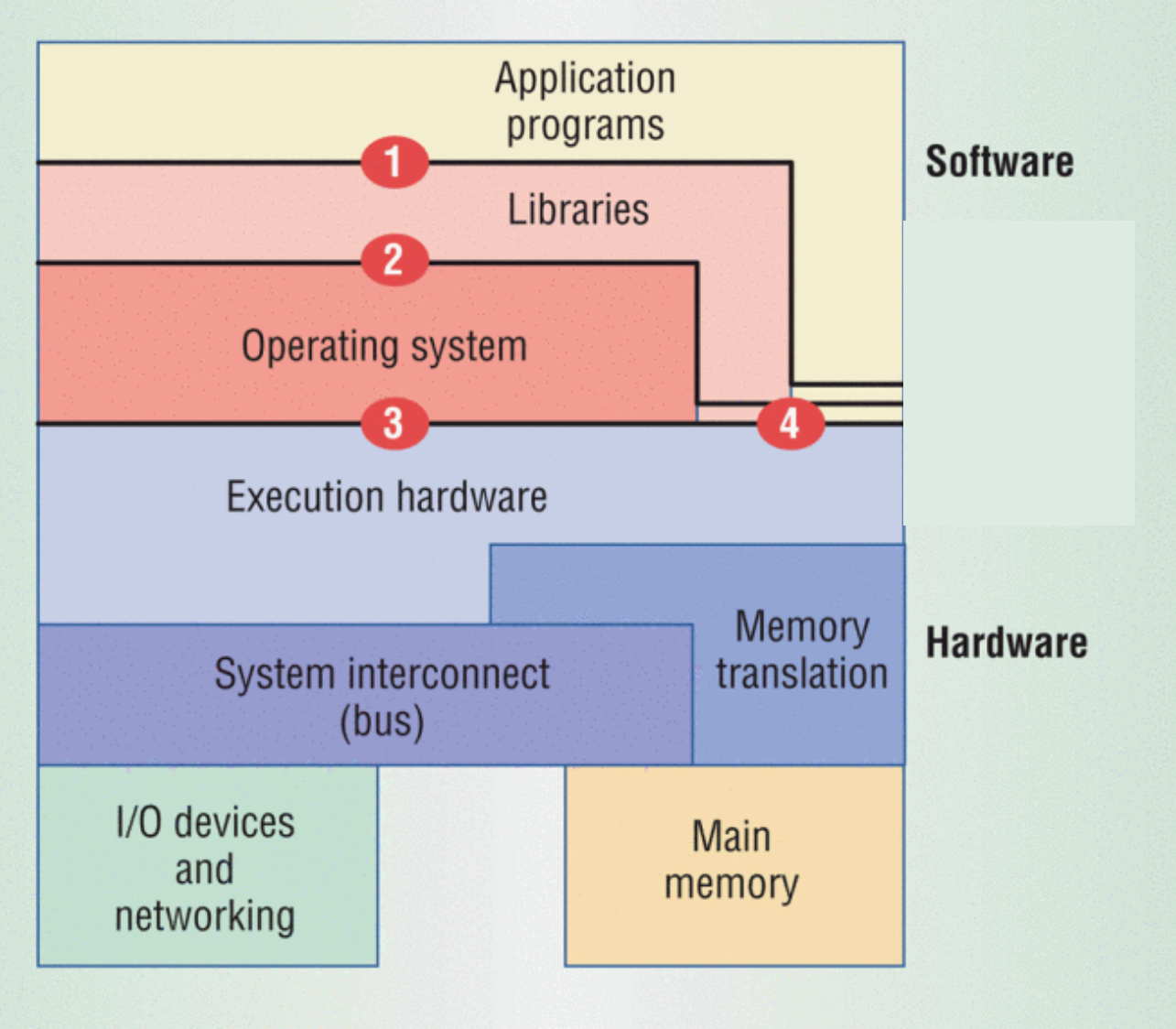
Computer Systems Layering

University of Cyprus
Department of Computer Science

IDENTIFY THE INTERFACES AND LAYERS COMMONLY FOUND IN A COMPUTER SYSTEM

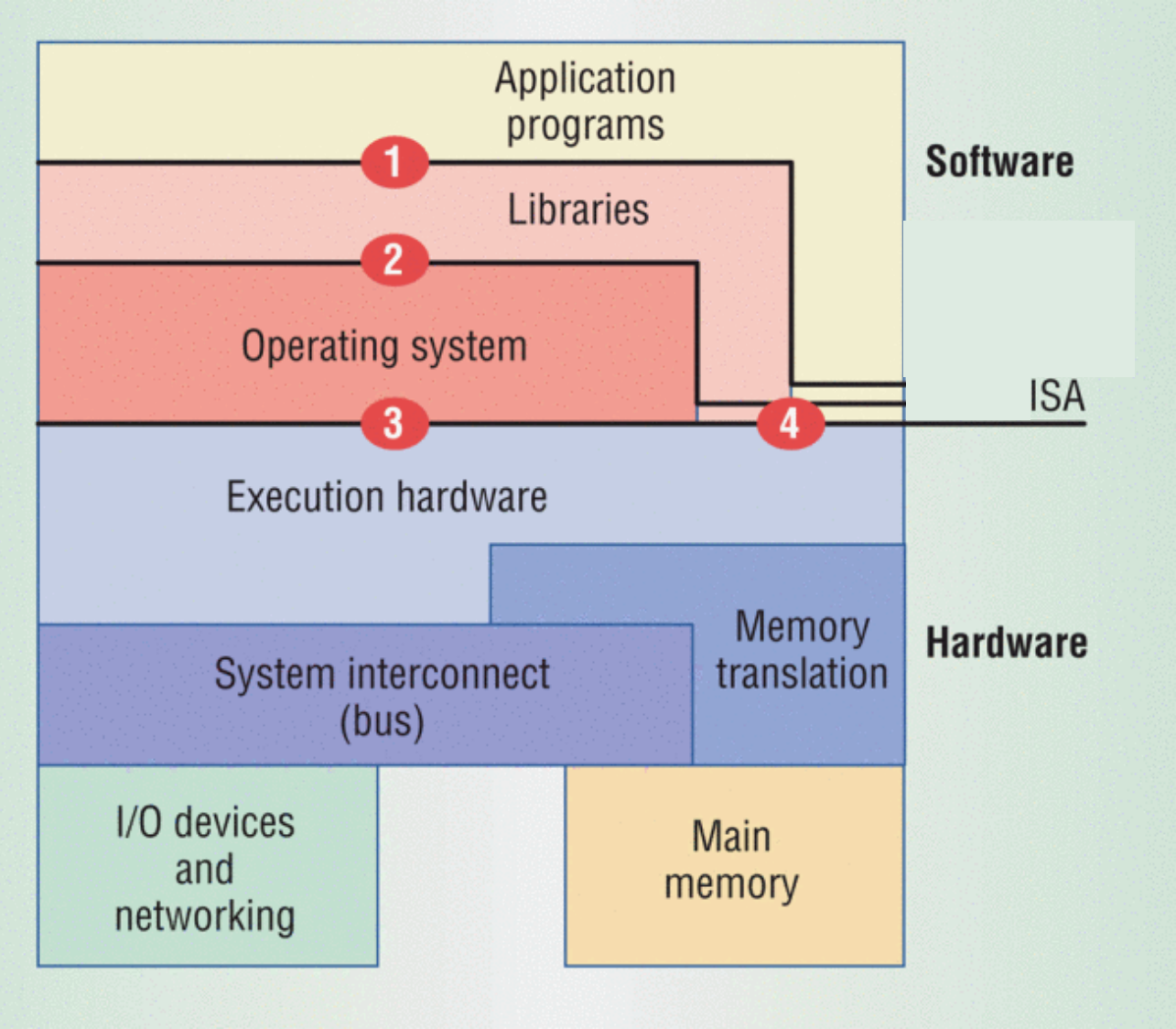
University of Cyprus
Department of Computer Science

Computer System Architecture: Layers and Interfaces



"The architecture of virtual machines," J. E. Smith and Ravi Nair, in Computer, vol. 38, no. 5, pp. 32-38, May 2005, doi: 10.1109/MC.2005.173.

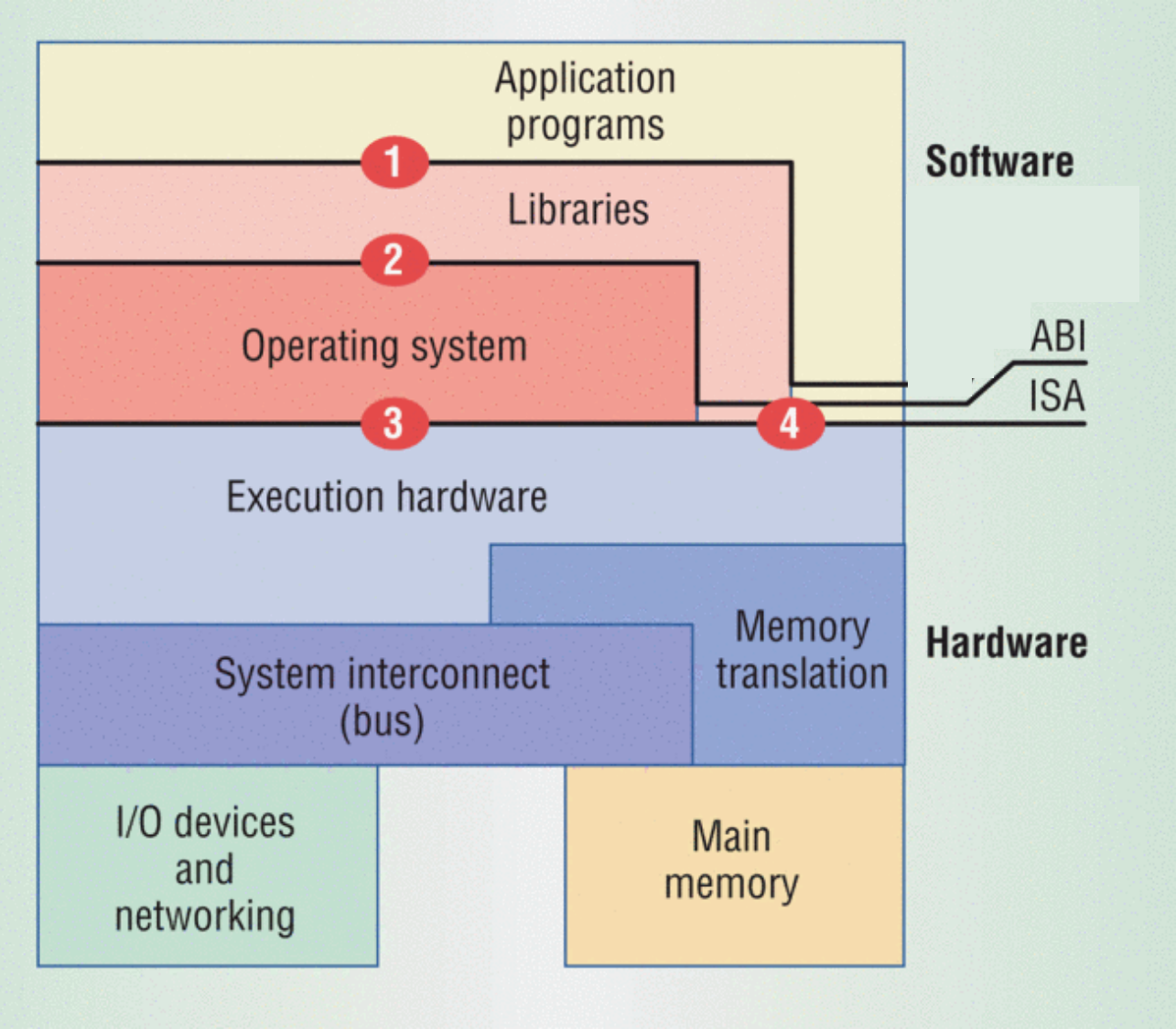
Computer System Architecture: Layers and Interfaces



1. Instruction set architecture (ISA)

"The architecture of virtual machines," J. E. Smith and Ravi Nair, in Computer, vol. 38, no. 5, pp. 32-38, May 2005, doi: 10.1109/MC.2005.173.

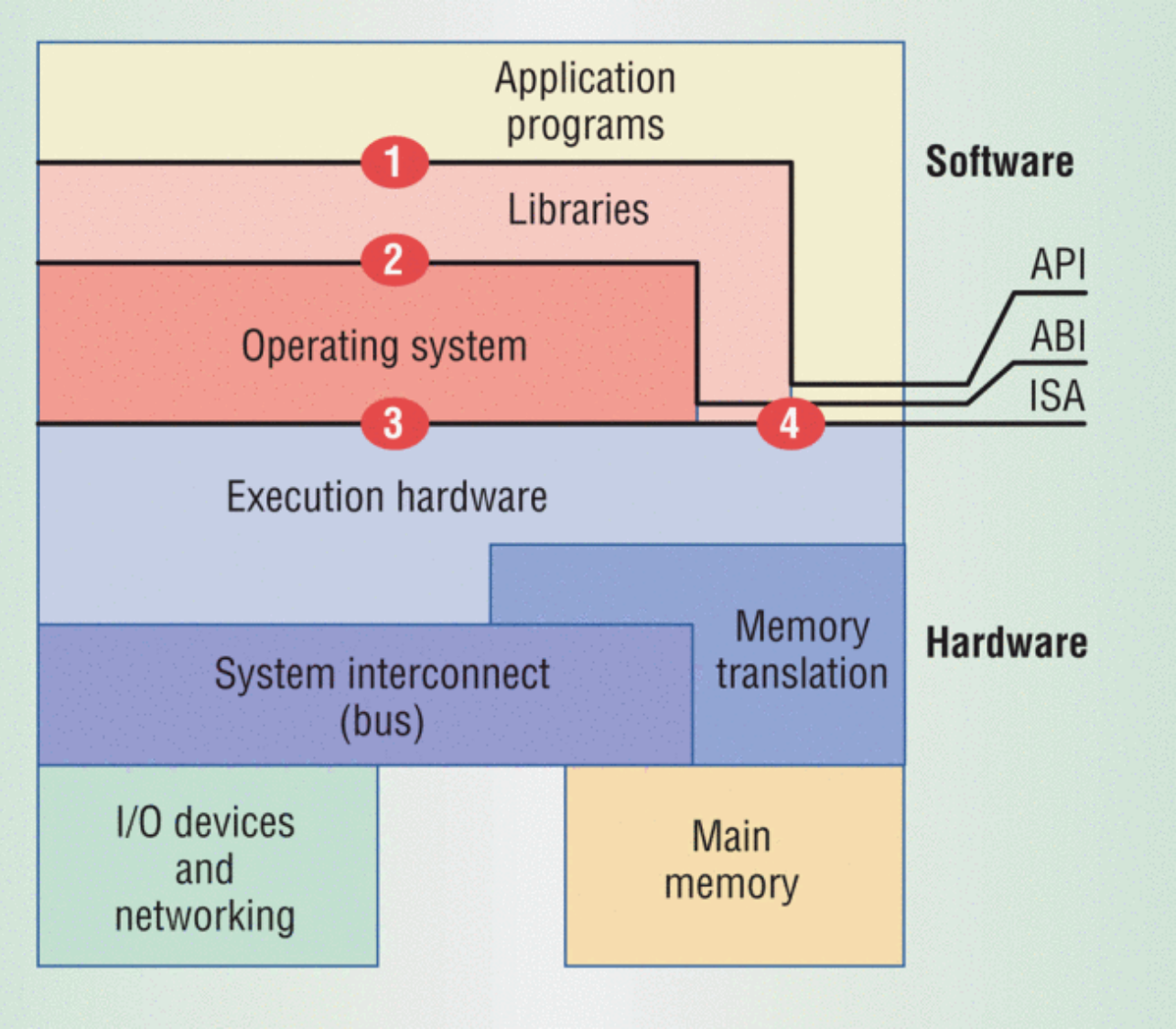
Computer System Architecture: Layers and Interfaces



1. Instruction set architecture (ISA)
2. Application binary interface (ABI)

"The architecture of virtual machines," J. E. Smith and Ravi Nair, in Computer, vol. 38, no. 5, pp. 32-38, May 2005, doi: 10.1109/MC.2005.173.

Computer System Architecture: Layers and Interfaces

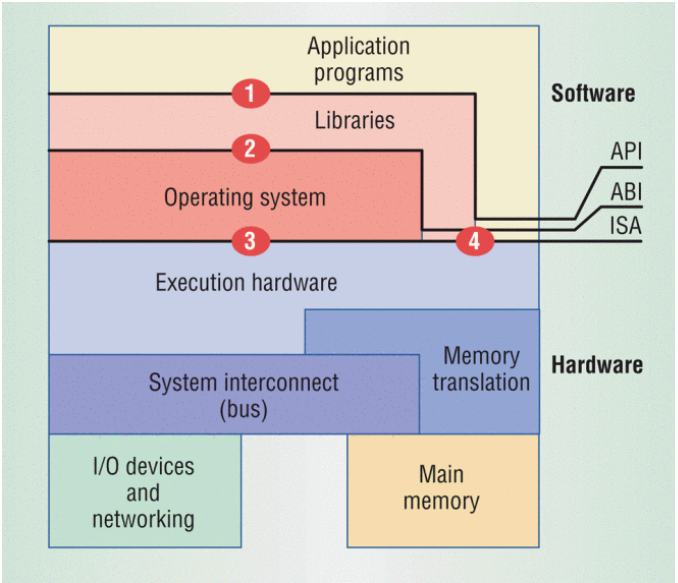


1. Instruction set architecture (ISA)
2. Application binary interface (ABI)
3. Application programming interface (API)

"The architecture of virtual machines," J. E. Smith and Ravi Nair, in Computer, vol. 38, no. 5, pp. 32-38, May 2005, doi: 10.1109/MC.2005.173.

Instruction Set Architecture

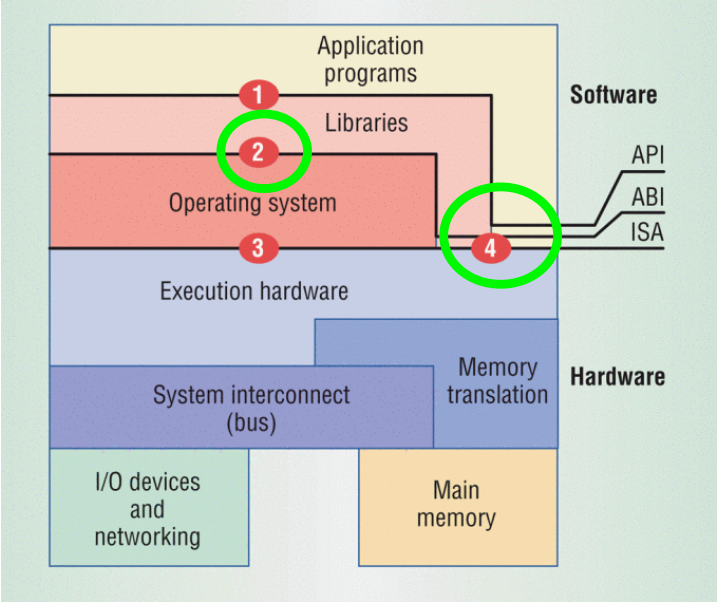
- Marks the division between hardware and software
- Consists of interfaces 3 and 4:
  - Interface 4 represents the **user ISA** and includes those aspects visible to an application program.
  - Interface 3, the **system ISA**, is a superset of the user ISA and includes those aspects visible only to operating system software responsible for managing hardware resources.



"The architecture of virtual machines," J. E. Smith and Ravi Nair, in Computer, vol. 38, no. 5, pp. 32-38, May 2005, doi: 10.1109/MC.2005.173.

Application Binary Interface

- Gives a program access to the hardware resources and services available in a system through the user ISA (interface 4) and the system call interface (interface 2).
- Is an interface between two binary program modules. Usually:
  - One is a library or operating system facility,
  - The other is a program that is being run by a user.
- Defines how data structures or computational routines are accessed in machine code.
- Does **not** include system (privileged) instructions;

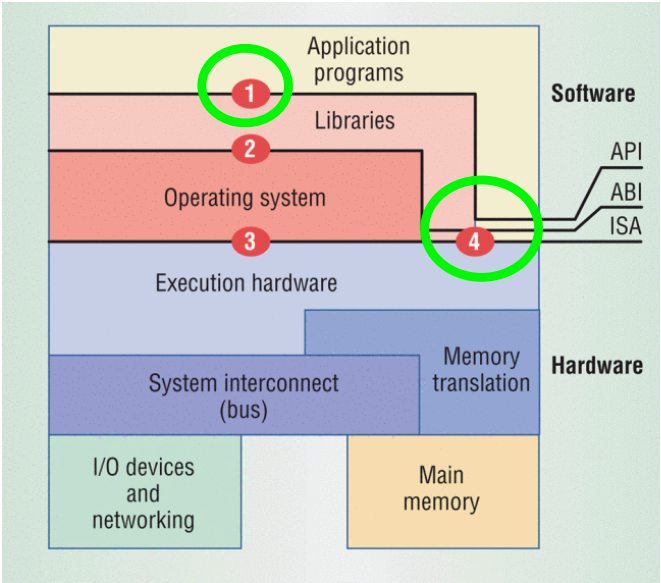


- System calls provide a way for an operating system to perform operations on behalf of a user program after validating their authenticity and safety.
  - All application programs interact with the hardware resources indirectly by invoking the operating system's services via the system call interface.

"The architecture of virtual machines," J. E. Smith and Ravi Nair, in Computer, vol. 38, no. 5, pp. 32-38, May 2005, doi: 10.1109/MC.2005.173.

Application Programming Interface

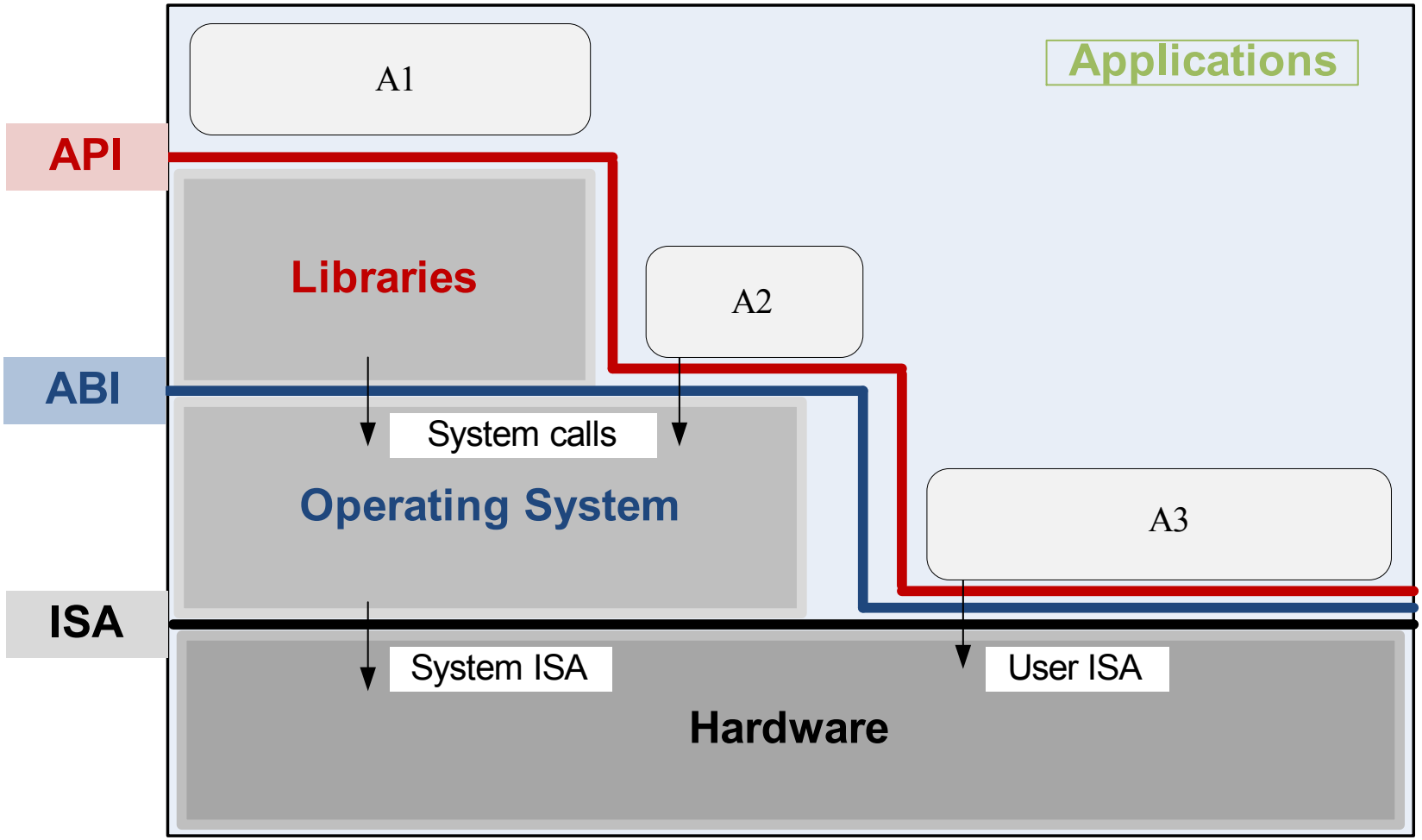
- Gives a program access to the hardware resources and services available in a system through:
  - the user ISA (interface 4) supplemented with
  - high-level language (HLL) library calls (interface 1).



- Any system calls are usually performed through libraries.
- Using an API enables application software to be ported easily, through recompilation, to other systems that support the same API.

"The architecture of virtual machines," J. E. Smith and Ravi Nair, in Computer, vol. 38, no. 5, pp. 32-38, May 2005, doi: 10.1109/MC.2005.173.

Layering and Interfaces



Application Programming Interface (**API**), Application Binary Interface (**ABI**), and Instruction Set Architecture (**ISA**).

An application uses library functions (**A1**), makes system calls (**A2**), and executes machine instructions (**A3**)

Introduction and Basic Concepts

Virtual Machines

University of Cyprus
Department of Computer Science

Virtual Machines

- A Virtual Machine (VM) can be:
  - An execution environment that runs an OS
  - An isolated environment that appears to be a whole computer, but actually only has access to a portion of the computer resources
- VM is encapsulated in a single data file. It can be:
  - moved from one computer to another,
  - opened in either one, and
  - be expected to work the same.

VMs vs. Simulators & Emulators

- Virtual machine
  - Models a machine exactly and efficiently
  - Minimal showdown
  - Needs to be run on the physical machine it virtualizes (more or less)
- Simulator
  - Provides a functionally accurate software model of a machine
  - May run on any hardware
  - Is typically slow (order of 1000 slowdown)
- Emulator
  - Provides a behavioral model of hardware (and possibly S/W)
  - Not fully accurate
  - Reasonably fast (order of 10 slowdown)

Courtesy of Gernot Heiser, UNSW, Australia

Implementing Virtual Machines

- We need to add a software layer to a real machine to support the desired architecture.
  - By doing so, a VM can circumvent real machine compatibility and hardware resource constraints.
- VM implementations lie at architected interfaces:
  - The fidelity with which a VM implements these interfaces is a major consideration.
- The **process** or **system** that runs on a VM is called the **guest**
- The underlying **platform** that supports the VM is called the **host**.
- The virtualizing software that implement VMs are termed runtime ("runtime software"), Virtual Machine Monitor (VMM) or hypervisor (depending on the VM type).

Virtualization Fundamentals

Modern Virtualization Technologies

University of Cyprus
Department of Computer Science

Virtualization Technologies

History and Evolution

University of Cyprus
Department of Computer Science

Virtualization: The History

- Virtualization technology was developed in the late **1960s** to make more efficient use of hardware.
- On a single **IBM System/360**, one could run in parallel, several environments that maintained full isolation and gave each of its customers the illusion of owning the hardware.
- IBM System/360 Virtualization: time sharing implemented at a coarse-grained level
- Achievements:
  - Consolidation.
  - Isolation.
  - Efficient resource management.

Virtualization: The History

- **1960's, IBM: CP/CMS** control program: a virtual machine operating system for the IBM System/360 Model 67
- **2000, IBM: z-series** with 64-bit virtual address spaces and backward compatible with the System/360
- **1974: Popek and Golberg** from UCLA published "Formal Requirements for Virtualizable Third Generation Architectures" where they listed the conditions a computer architecture should satisfy to support virtualization efficiently. *The popular x86 architecture that originated in the 1970s did not support these requirements for decades.*
- **1990's, Stanford & VMware:** Researchers developed a new hypervisor and founded VMware, the biggest virtualization company of today's. First virtualization solution was is 1999 for x86.
- IBM was the first to produce and sell virtualization for the mainframe. But, VMware popularized virtualization for the masses.
- Today many virtualization solutions: Xen from Cambridge, KVM, Hyper-V, …

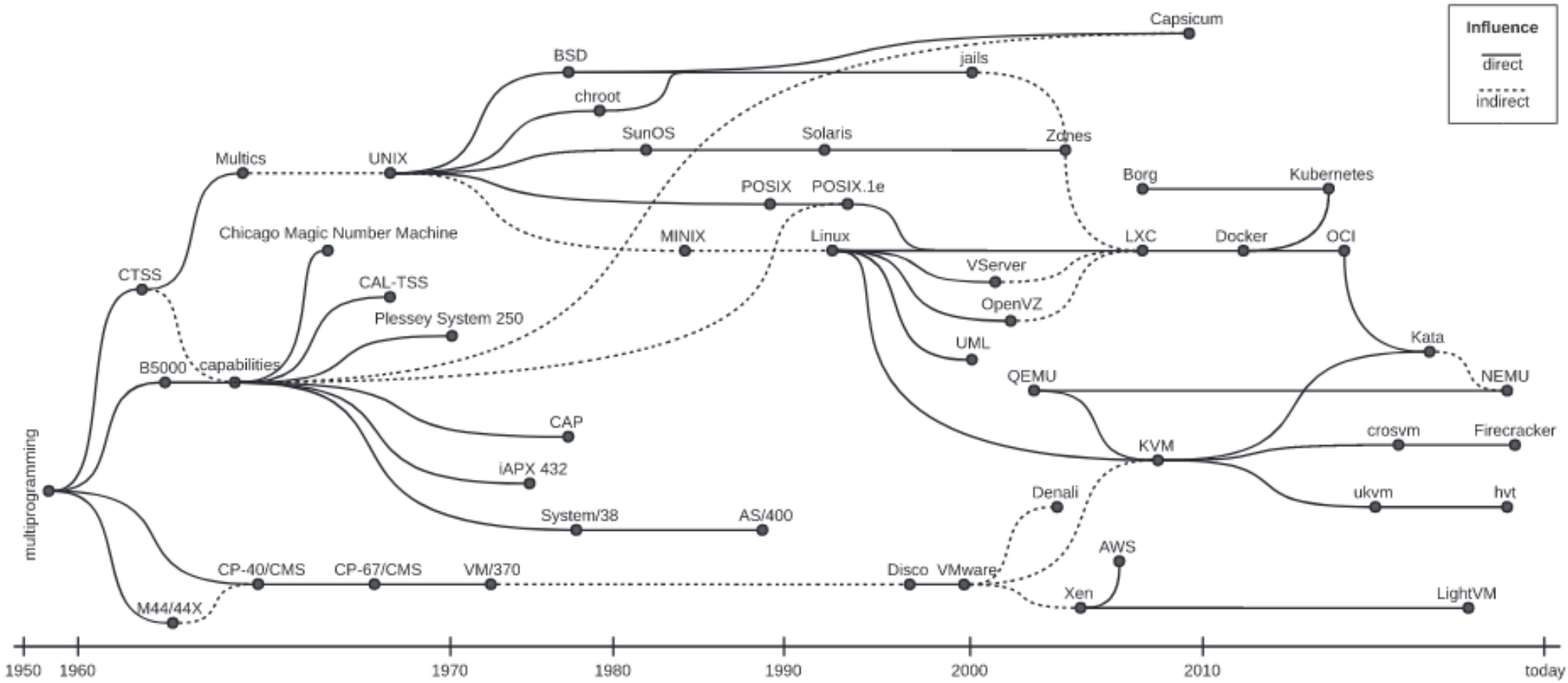


Fig. 1. The evolution of virtual machines and containers.

Source: Randal, ACM Computing Surveys, 2/2020

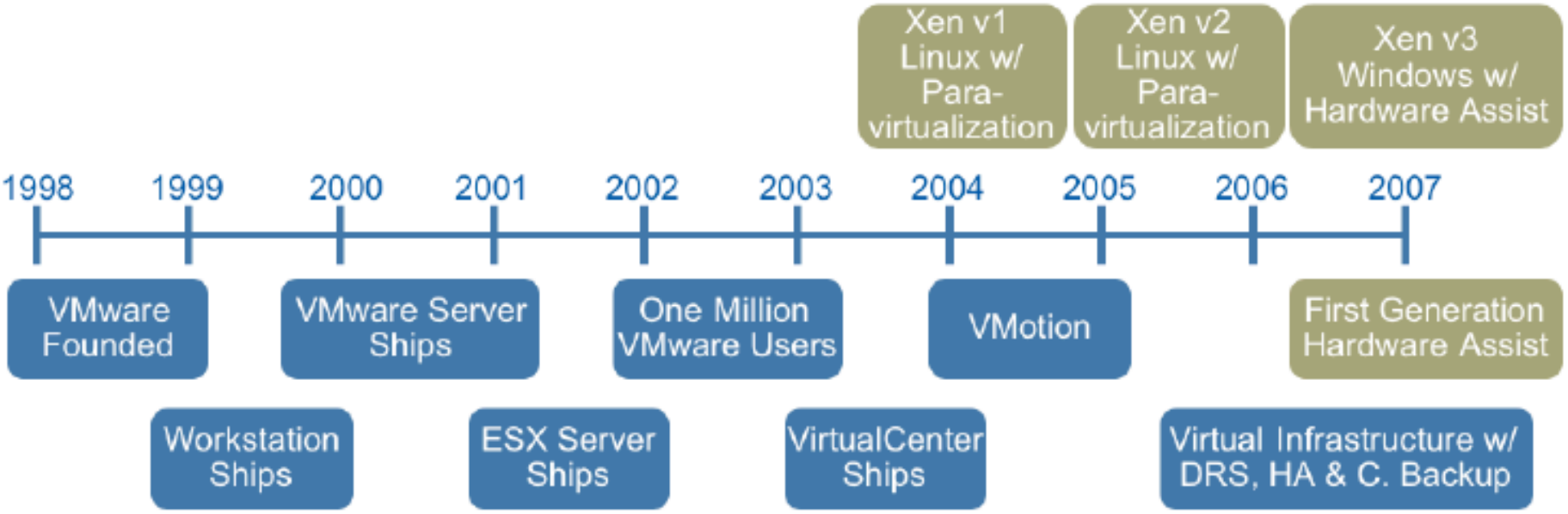


Figure 1 – Summary timeline of x86 virtualization technologies

Source: VMWare

Virtualization Requirements

- **Sufficient conditions** for a computer architecture to support virtualization and allow a hypervisor to operate efficiently:
  - A program running under the hypervisor should exhibit a behavior essentially identical to that demonstrated when running on an equivalent machine directly.
  - The hypervisor should be in complete control of the virtualized resources.
  - A statistically significant fraction of machine instructions must be executed **without** the intervention of the hypervisor, directly by real hardware.

Popek & Goldberg, "Formal requirements for virtualizable third generation architectures", CACM July1974

Virtualization Requirements

"A virtual machine (VM) is an efficient, isolated duplicate of a real machine"

- VM should behave identically to the real machine
  - Programs cannot distinguish between execution on real or virtual hardware
  - Except for:
    - Fewer resources available (potentially different between executions)
    - Some timing differences (when dealing with devices)
- Isolated: Several VMs execute without interfering with each other
- Efficient: VM should execute at a speed close to that of real hardware

Virtualization: Why?

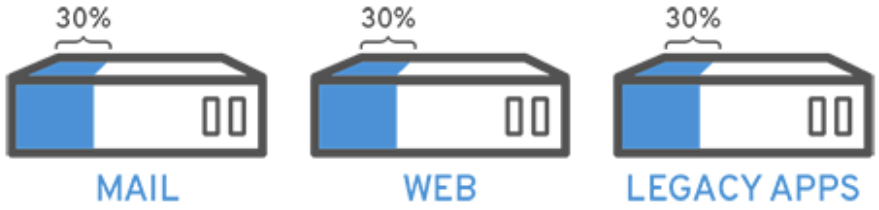
- As the scale of systems and the size of user base grows, it becomes very challenging to manage computing recourses.
- E.g., in data centers:
  - provision for peak demands: overprovisioning
  - heterogeneity of hardware and software
  - machine failures
- We need improved ways of managing resources at scale

Virtualization: Why?

- Simplifies the abstract management of physical resources, focusing on key computing abstractions:
  1. processors
  2. memory
  3. communications links
- Examples:
  - The state of a virtual machine (VM) running under a virtual machine monitor (VMM) can de **saved** and **migrated** to another server to *balance the load*
  - Virtualization allows users to operate in environments they are familiar with, rather than forcing them to specific ones

Virtualization: Why?

- Server sprawl in enterprise environments:
  - Vendors want to run their applications in isolation.
  - Underline OS heterogeneity for different enterprise applications.
  - Complexity of integration projects.
- Many servers are typically underutilized:
  - Large analyst firms estimate that resource utilization of **15-20%** is common
  - Often this could be in the **5-12%** range (Vogels, Amazon, 2008)



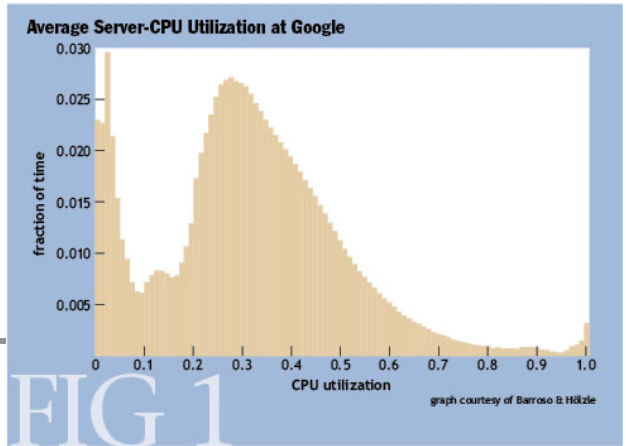


FIG 1

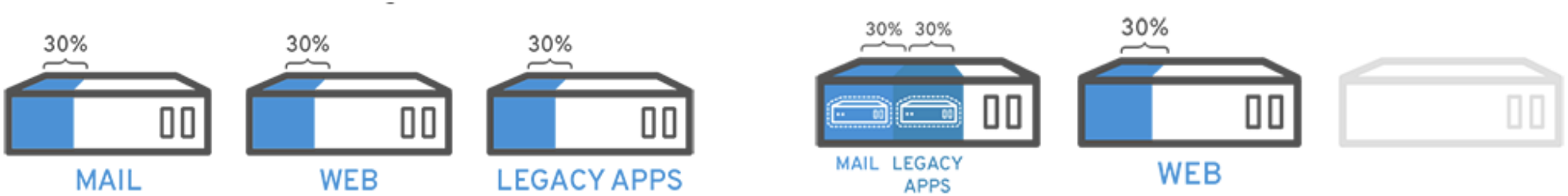
Beyond Server Consolidation, W. Vogels, Amazon.com, ACM Queue, 2008.

Virtualization & Cloud Computing

- A basic enabler of Cloud Computing
- Cloud resource virtualization is important for:
  - Performance isolation: we can dynamically assign and account for resources across different applications
  - System security: allows isolation of services running on the same hardware
  - Performance and reliability: allows applications to migrate from one platform to another
  - Development and management of services offered by a provider
  - Server **consolidation**: we can use same physical server for multiple applications

Server consolidation

- Server consolidation can decrease IT cost by multiplexing physical resources over a number of virtualized environments:
  - reduce hardware required
  - reduce data-center footprint
  - indirectly reduce power consumption



Modern Virtualization Technologies

Virtualization approaches

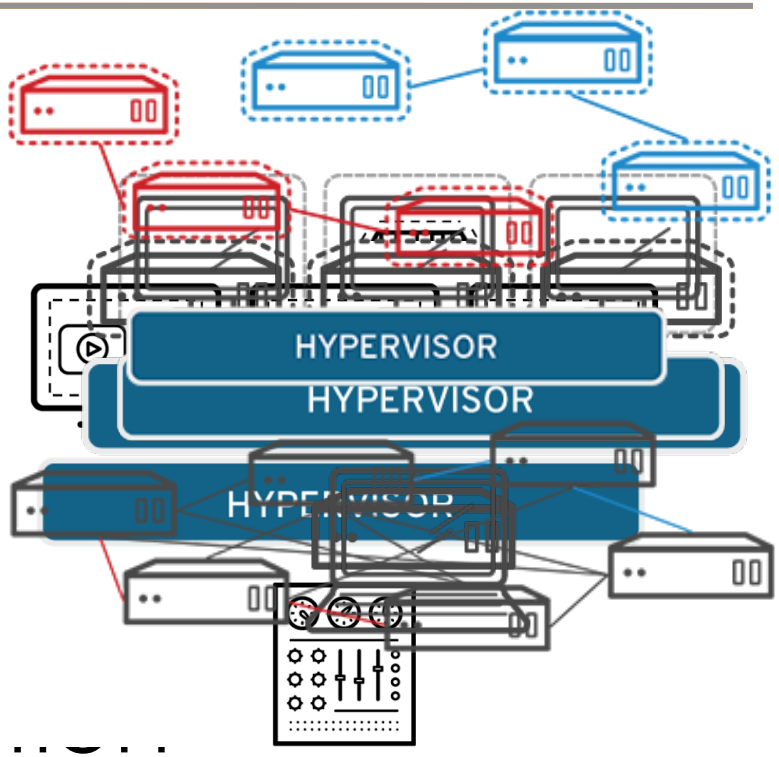
University of Cyprus
Department of Computer Science

Virtualization: High-Level Tasks

Virtualization simulates the interface to a physical object by:

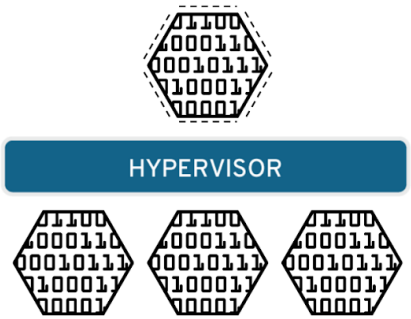
- Multiplexing: creates **multiple virtual objects** from **one instance of a physical object**.
  - Maps many virtual objects to one physical.
  - Eg - a processor is multiplexed among a number of processes or threads.
- Aggregation: creates **one virtual object** from **multiple physical objects**.
  - Maps one virtual object to many physical objects.
  - Example - a number of physical disks are aggregated into a RAID disk.
- Emulation: constructs a **virtual object of a certain type** from a **different type of a physical object**.
  - Example - a physical disk emulates a Random Access Memory (RAM).
- Multiplexing and emulation.
  - Examples - **virtual memory with paging** multiplexes **real memory** and **disk**; a virtual address emulates a real address.

Virtualization Approaches



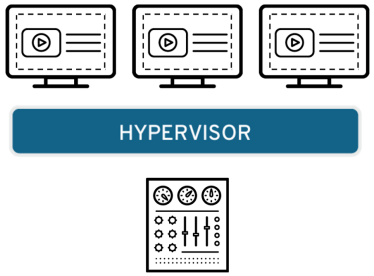
- Data virtualization
- Desktop virtualization
- Server virtualization
- Operating system virtualiz… (containers)
- Network functions virtualization

Data Virtualization



- Data virtualization:
  - brings together data from multiple sources
  - easily accommodates new data sources
  - transforms data according to user needs
- Data virtualization tools sit in front of multiple data sources and allows them to be treated as single source.

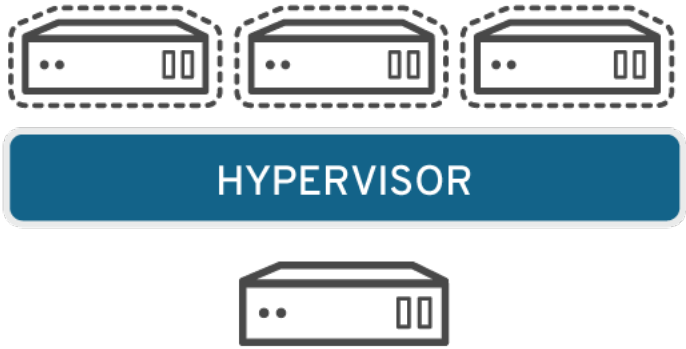
Desktop Virtualization



- Allows a central administrator (or automated administration tool) to deploy simulated desktop environments to hundreds of physical machines at once.
- Unlike traditional desktop environments that are physically installed, configured, and updated on each machine, desktop virtualization allows admins to perform mass configurations, updates, and security checks on all virtual desktops.

Server virtualization

- Server virtualization converts one physical server into multiple virtual machines. Each virtual server acts like a unique physical device, capable of running its own operating system (OS).
  - The physical server is called the **host**.
  - The virtual servers are called **guests** and behave like physical machines.
- Each virtualization system uses a different approach to allocate physical server resources to virtual server needs.
- Approaches:
  - **Full virtualization**
  - **Para-virtualization**
  - **OS-level virtualization**



Network-function virtualisation

- NFV separates a network's key functions (like directory services, file sharing, and IP configuration) so they can be distributed among environments.
- Once software functions are independent of the physical machines they once lived on, specific functions can be packaged together into a new network and assigned to an environment.
- Virtualizing networks reduces the number of physical components—like switches, routers, servers, cables, and hubs—that are needed to create multiple, independent networks, and it' s particularly popular in the telecommunications industry.

Modern Virtualization Technologies

Virtual Machine Taxonomies

University of Cyprus
Department of Computer Science

WHAT IS A MACHINE ?

Consider the meaning of “machine” from

the perspective of the **process** and the **operating system**

WHAT IS A PROCESS?

A running program with its allocated resources and execution state

Process Perspective

- The "machine" where a program runs consists of:
  - a logical memory address space assigned to the process
  - user-level instructions and registers that allow the execution of code belonging to the process.
- Machine's I/O is visible only through the operating system:
  - the only way the process can interact with the I/O system is through operating system calls.
- ABI defines the machine as seen by the process.
- API specifies the machine characteristics as seen by an application HLL program.

System Perspective

- The Operating System is a full execution environment that can support numerous processes simultaneously.
  - These processes share a file system and other I/O resources.
- OS environment persists over time as processes come and go.
- OS allocates real memory and I/O resources to the processes, and allows the processes to interact with their resources.
- The machine where the OS runs is defined by the underlying hardware's characteristics;
  - Therefore, it is the **ISA** that provides the interface between the system and machine.

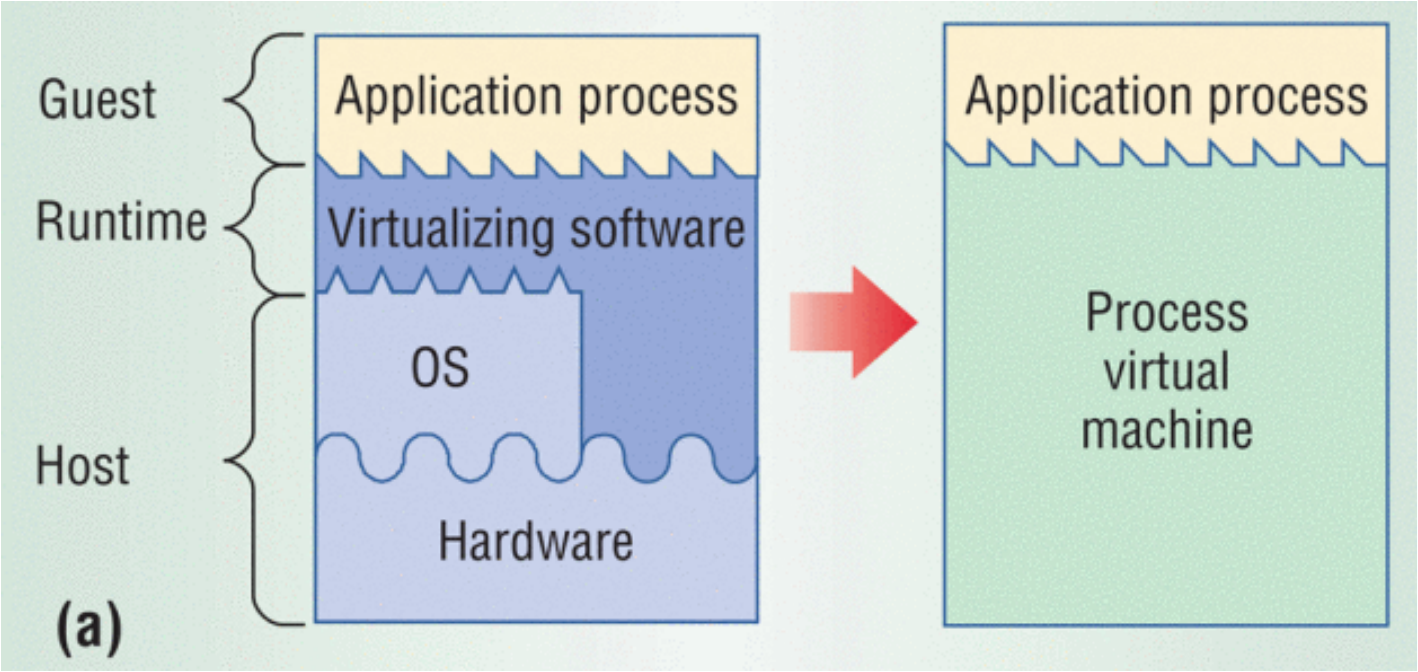
Process and System VMs

- A process VM is a virtual platform that executes an individual process.
  - Exists solely to support the process;
  - Created when the process is created and terminates when the process terminates.
- A system VM provides a complete, persistent system environment that supports an operating system along with its many user processes. Provides:
  - The guest operating system
  - Access to virtual hardware resources, including networking, I/O, along with a processor and memory.
  - Perhaps a graphical user interface.

"The architecture of virtual machines," J. E. Smith and Ravi Nair, in Computer, vol. 38, no. 5, pp. 32-38, May 2005, doi: 10.1109/MC.2005.173.

Process VMs

- The Virtualizing software that implements a process VM is often termed the runtime ("runtime software").
- The runtime is at the **ABI** or **API level**, atop the OS/HW combination: emulates both user-level instructions and either operating system or library calls.

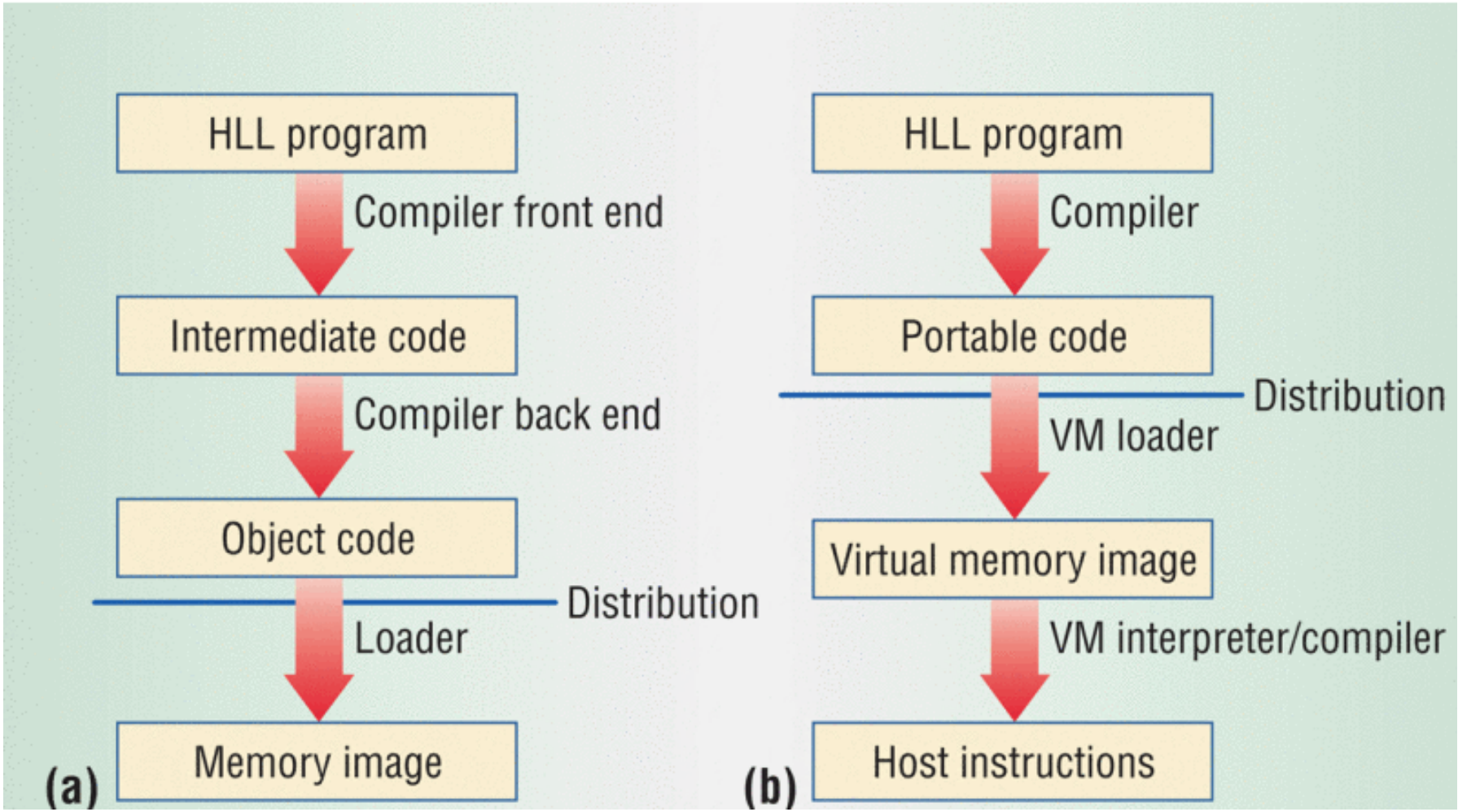


"The architecture of virtual machines," J. E. Smith and Ravi Nair, in Computer, vol. 38, no. 5, pp. 32-38, May 2005, doi: 10.1109/MC.2005.173.

Process VM Categories

- Multiprogrammed Systems: most OS can simultaneously support multiple used processes through through multiprogramming, which gives each process the illusion of having a complete machine to itself.
- Emulators and Dynamic Binary Translators: support program binaries compiled to an instruction set different from the one the host executes: implemented through **interpretation** or **dynamic binary translation**.
- Same-ISA Binary Optimizers: perform code optimizations during translation, where instruction sets of host and guest are the same, and the purpose of the VM is to optimize execution performance of a program binary.
- High-Level-Language VMS: instead of emulating one conventional architecture on another, these are designed to match the features of given HLL(s) and facilitate cross-platform application portability (e.g., JVM)
  - In a HLL VM, a compiler front end generates abstract machine code in a virtual ISA that specifies the VM's interface.
  - This virtual ISA code, along with associated data structure information (metadata), is distributed for execution on different platforms.
  - Each host platform implements a VM capable of loading and executing the virtual ISA and a set of library routines specified by a standardized API. In its simplest form, the VM contains an **interpreter**. More sophisticated, higher-performance VMs **compile** the abstract machine code into **host machine code** for direct execution on the host platform.

"The architecture of virtual machines," J. E. Smith and Ravi Nair, in Computer, vol. 38, no. 5, pp. 32-38, May 2005, doi: 10.1109/MC.2005.173.

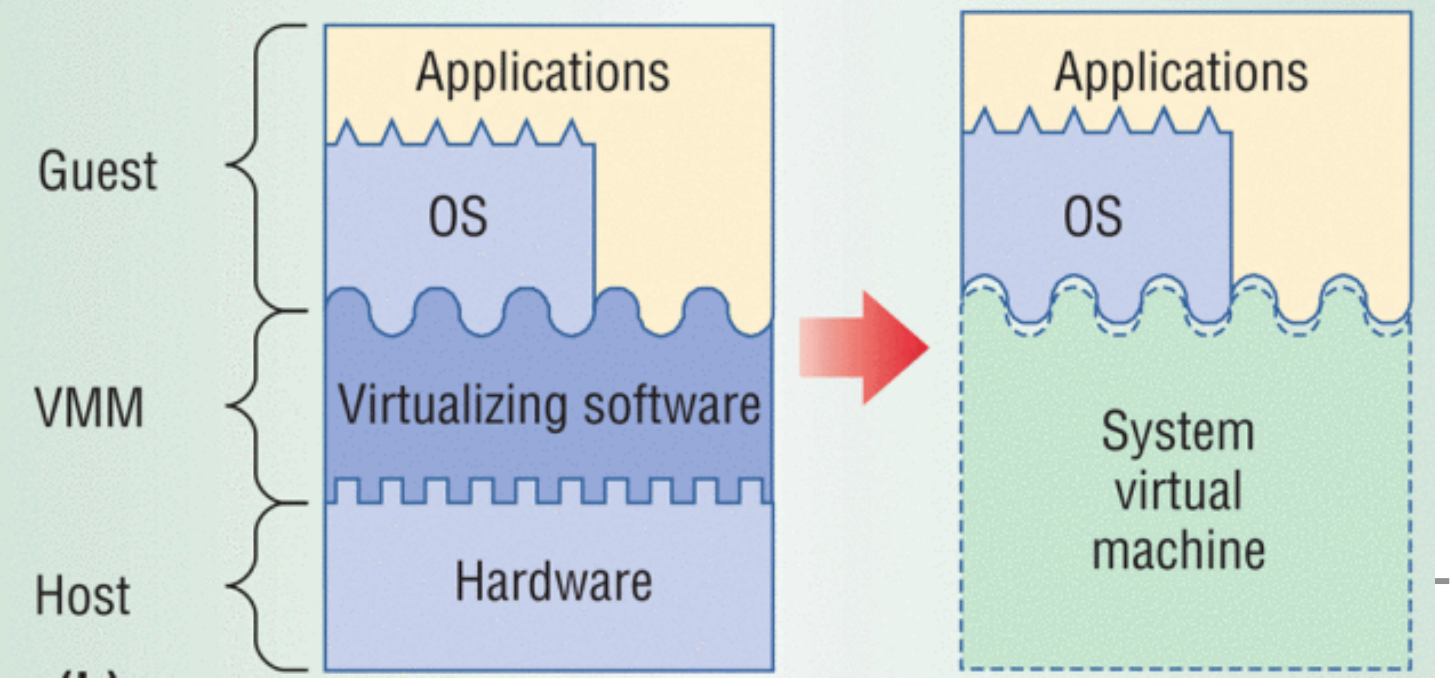


High-level-language environments
(a) conventional environment where platform-dependent object code is distributed.
(b) hll vm environment where a platform-dependent vm executes portable intermediate code.

"The architecture of virtual machines," J. E. Smith and Ravi Nair, in Computer, vol. 38, no. 5, pp. 32-38, May 2005, doi: 10.1109/MC.2005.173.

System VMs and Hypervisors

- Virtualizing software is typically referred to as the virtual machine monitor (VMM) or hypervisor.
- A **Hypervisor**:
  - Provides complete environment in which an OS and many processes of possibly multiple users, can coexist.
  - Has access to and manages all hardware resources, dividing them among multiple guest OSs
  - Maintains hidden control of a guest operating system and its application processes.
  - When a guest operating system performs a privileged instruction or operation that directly interacts with shared hardware resources, the VMM intercepts the operation, checks it for correctness, and performs it on behalf of the guest. Guest software is unaware of this behind-the-scenes work.



System VM Categories

- From the user perspective most system VMs provide essentially the same functionality (differences in implementation exist)
- Classic System VMs: place the VMM on bare hardware and the VMs fit on top. The VMM runs in the most highly privileged mode, while all guest systems run with reduced privileges so that the VMM can intercept and emulate all guest operating system actions that would normally access or manipulate critical hardware resources.
- Hosted VMs: virtualizing software is built on top of an existing host operating system, resulting in a hosted VM (e.g. VMware):
  - Users can install VMM just like a typical application program.
  - Virtualizing software can rely on host OS to provide device drivers and other lower-level services.
- Whole-System VMs: address the case where guest and host have different ISA (e.g. Windows PCs and Apple PowerPC):
  - VM must virtualize and emulate both application and operating system code
  - VMM executes as application supported by host OS; uses no system ISA ops.

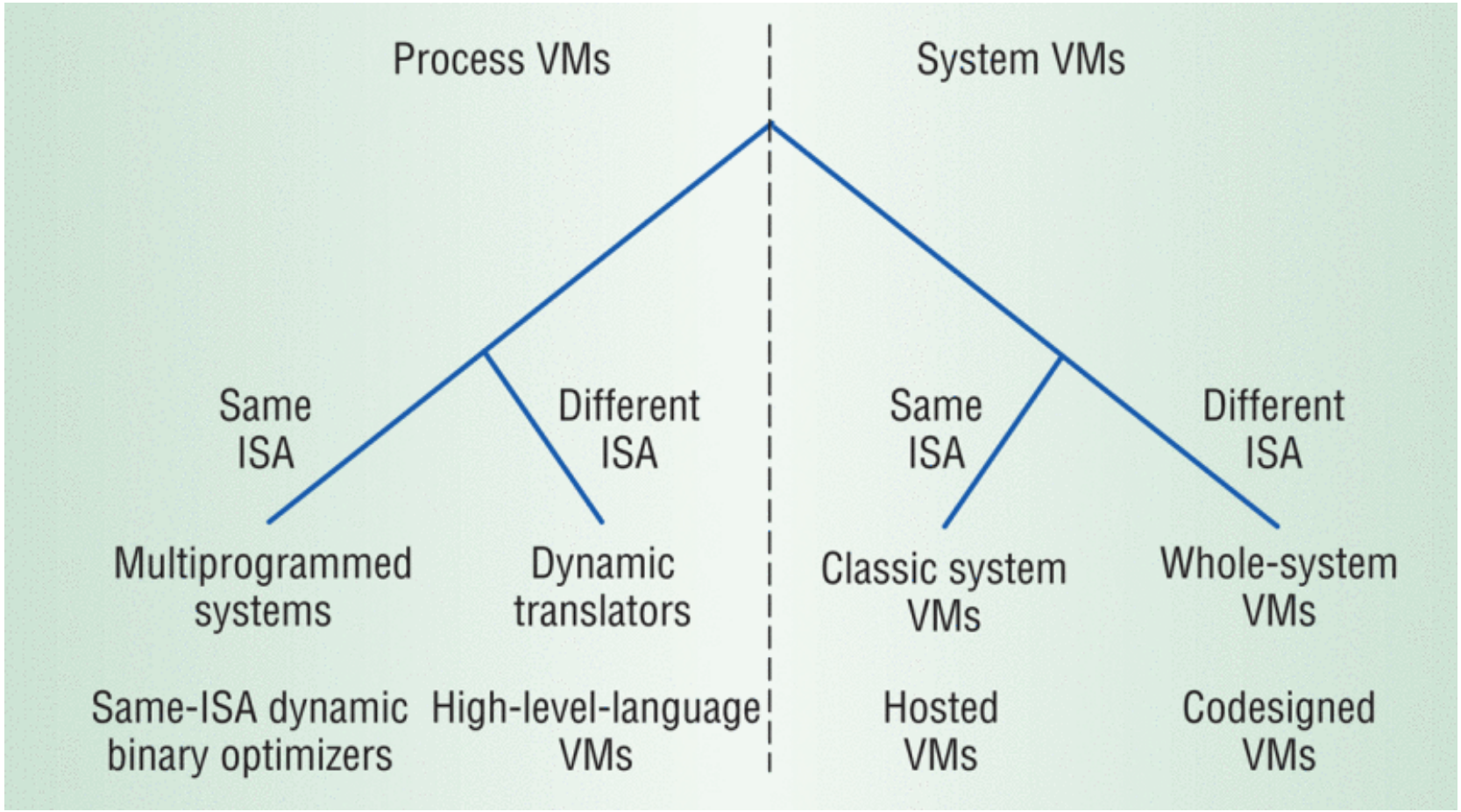
"The architecture of virtual machines," J. E. Smith and Ravi Nair, in Computer, vol. 38, no. 5, pp. 32-38, May 2005, doi: 10.1109/MC.2005.173.

System VM Categories

- Multiprocessor Virtualization: Underlying host platform is a large shared-memory multiprocessor that needs to be partitioned into multiple smaller multiprocessor systems:
  - Physical partitioning: physical resources that one virtual system uses are disjoint from those used by other virtual systems
  - Logical partitioning: underlying hardware resources are time-multiplexed between the different partitions, thereby improving system resource utilization.
  - Both techniques typically use special software or firmware support based on underlying hardware modifications specifically targeted at partitioning.
- Co-designed VMs: implement new, proprietary (host) ISAs targeted at improving performance, power efficiency, or both.
  - The host's ISA may be completely new, or an extension of existing ISA.
  - The VMM appears to be part of the hardware implementation and resides in a region of memory concealed from all conventional software.
  - VMM includes binary translator that converts guest instructions into optimized sequences of host ISA instructions and caches them in the concealed memory region.

"The architecture of virtual machines," J. E. Smith and Ravi Nair, in Computer, vol. 38, no. 5, pp. 32-38, May 2005, doi: 10.1109/MC.2005.173.

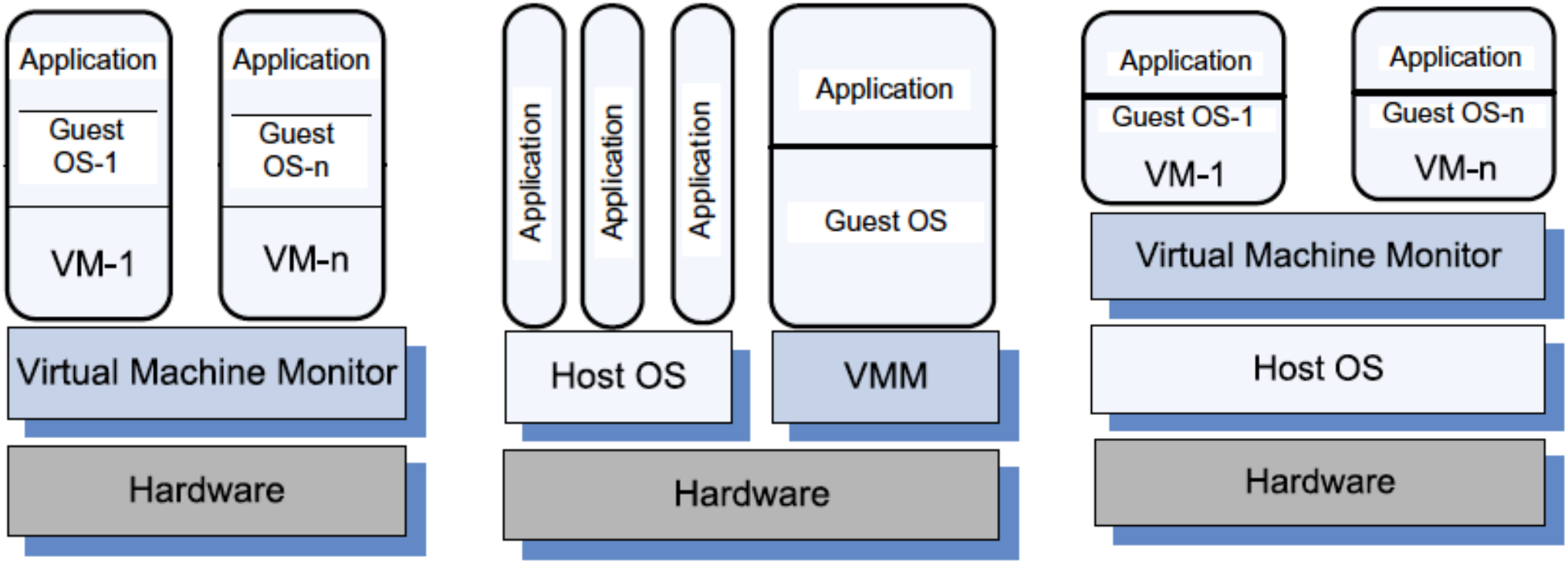
Taxonomy of Process & System VMs



Within the general categories of process and system VMs, **ISA simulation is the major basis of differentiation**.

"The architecture of virtual machines," J. E. Smith and Ravi Nair, in Computer, vol. 38, no. 5, pp. 32-38, May 2005, doi: 10.1109/MC.2005.173.

Same-ISA VM Classes



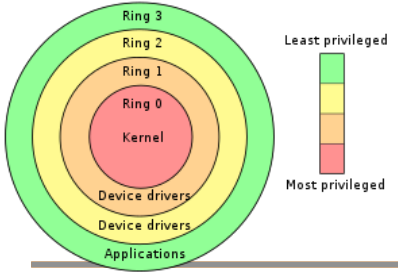
Classes of VM for systems with the same ISA:

Traditional, **hybrid**, and **hosted**

Modern Virtualization Technologies

Virtualization Mechanisms

University of Cyprus
Department of Computer Science



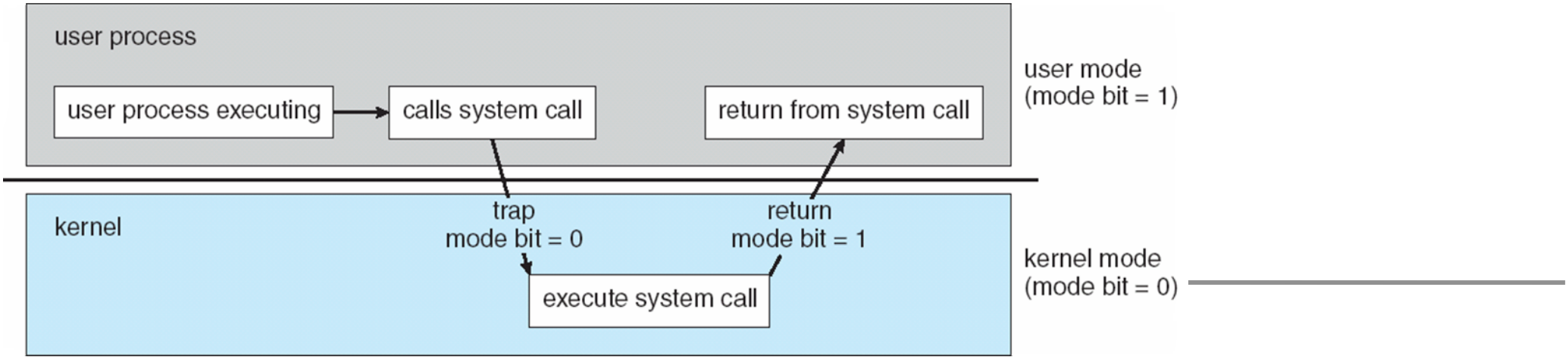
Protection Rings

- Hierarchical protection domains, often called protection rings, are mechanisms in CPU architectures and Operating Systems aiming at protecting data and functionality from faults and malicious behavior.
- A protection ring is one of two or more hierarchical levels or layers of privilege within the architecture of a computer system:
  - Ring hierarchy: from most privileged (most trusted, usually numbered **zero**) to least privileged (least trusted).
- Hardware-enforced by some CPU architectures that provide different CPU modes at the hardware or microcode level.
- On most operating systems, Ring 0 interacts most directly with the physical hardware such as the CPU and memory.
- Special gates between rings are provided to allow an outer ring to access an inner ring's resources in a predefined manner, as opposed to allowing arbitrary usage.

More: https://en.wikipedia.org/wiki/Protection_ring

OS Dual-Mode Operation

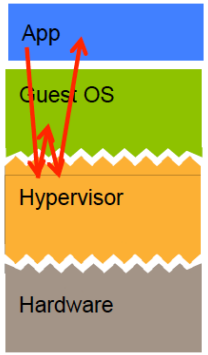
- Dual-mode operation allows OS to protect itself and other system components
  - User mode and kernel mode
  - Mode bit provided by hardware
    - Ability to distinguish when system is running user or kernel code
    - Some instructions are privileged: only executable in kernel mode
    - System call changes mode to kernel, return resets it to user



User-mode vs Kernel-mode

- Kernel-code (in particular, interrupt handlers) runs in kernel mode
  - the hardware allows all machine instructions to be executed and allows unrestricted access to memory and I/O ports
- Everything else runs in user mode
- OS relies very heavily on this hardware-enforced protection mechanism

De-privileging



- A classical VMM executes the guest operating systems directly, but **at a reduced privilege level**.
- All instructions that read or write privileged state can be made to trap when executed in the unprivileged context.
- Traps can result from:
  - The instruction type itself (e.g., an out instruction)
  - The VMM protecting structures that the instructions access (e.g., the address range of a memory-mapped I/O device)
- The VMM intercepts traps from the de-privileged guest, and emulates the trapping instruction against the virtual machine state.

Classical VMM Approach

- Traditional approach: "trap and emulate"
  - Guest software executes in unprivileged mode; attempts to access physical resource.
  - Hardware raises exception (trap), invoking hypervisor's exception handler.
  - Hypervisor emulates the result, based on access to virtual resource
- Most instructions do not trap. This:
  - Makes efficient virtualization possible
  - Requires that VM ISA is (almost) same as physical processor ISA

Courtesy of Gernot Heiser, UNSW"

x86 Architecture

- x86 is the most popular processor architecture, implemented in processors from Intel, Cyrix, AMD, etc. As of 2018, the majority of personal computers and laptops sold are based on the x86 architecture.
- x86 OS designed to run directly on bare-metal hardware: assumes full 'ownership' of computer hardware.
- x86 architecture offers four levels of privilege: Ring 0, 1, 2 and 3
  - User level applications typically run in Ring 3.
  - The OS needs direct access to memory and hardware: must execute privileged instructions in Ring 0.

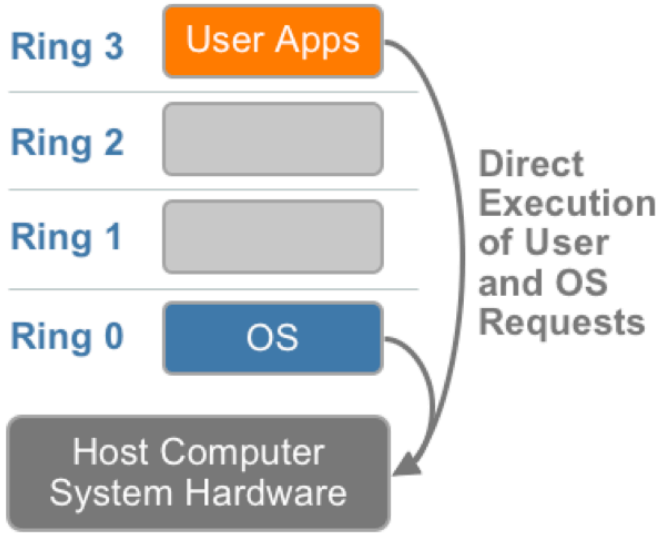


Figure 4 – x86 privilege level architecture without virtualization

x86 Machine Instruction Types

- **Non-privileged** instructions can be executed in user mode.
- **Privileged** instructions can be executed in kernel mode. When attempted to be executed in user mode, they cause a trap and so are executed in kernel mode.
- **Sensitive** instructions can be:
  - Control sensitive: they attempt to change the configuration of system resources (either the memory allocation or privileged mode).
  - Mode sensitive: behavior differs when running in the privileged or in non-privileged mode.
- Some sensitive instructions are hard to virtualize: they have different semantics when executed outside **Ring 0**.
- The difficulty in trapping and translating these sensitive instructions at runtime was the challenge that originally made x86 architecture virtualization look impossible.

"Trap-and-emulate" Requirements

- An architecture is virtualizable if all sensitive instructions are privileged:
  - Can achieve accurate, efficient guest execution by simply running guest binary on hypervisor
- Virtualized execution is indistinguishable from native, except:
  - Resources are more limited (running on smaller machine)
  - Timing is different (if there is an observable time source)
- VMM controls resources

Courtesy of Gernot Heiser, UNSW

"Trap-and-emulate" Overheads

- VMM needs to maintain virtualized privileged machine state
  - processor status
  - addressing context
  - device state...
- VMM needs to emulate privileged instructions
  - translate between virtual and real privileged state, e.g. guest ↔ real page tables
- Virtualization traps are expensive on modern hardware (can be 100s of cycles - x86)
- Some OS operations involve frequent traps

Courtesy of Gernot Heiser, UNSW

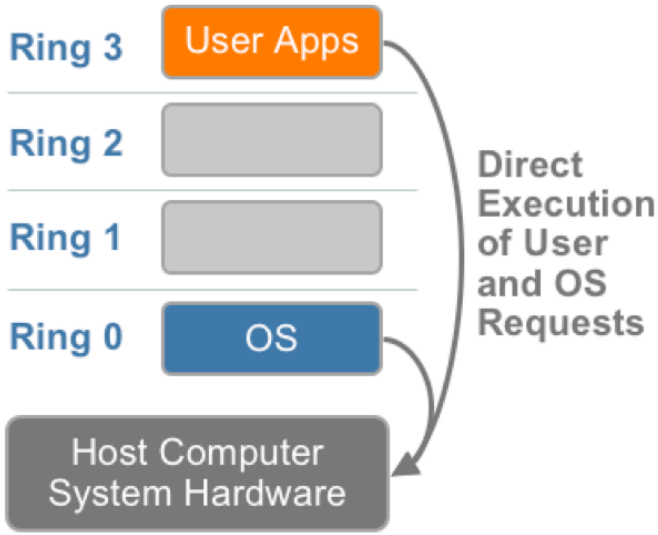
Modern Virtualization Technologies

x86 Virtualization

University of Cyprus
Department of Computer Science

Virtualizing the x86 Architecture

- Need to place a VMM under the OS (which expects to be in Ring 0) so as to create and manage the VMs that deliver shared resources.
- However:
  - Running the VMM in ring 0, gives it equal privileges to the OS: wrong
  - Running VMM in rings 1,2,3, gives the OS higher privileges than the VMM: wrong
- How can this be be addressed?
  - Move the guest OS to ring 1 and place VMM to ring 0?



Challenges of x86 CPU Virtualization

- Even the later x86 architected 32- and 64-bit CPUs with protected modes, were considered to be not classically virtualizable due to:
  - Visibility of privilege state: the guest OS can observe that it runs at a reduced privilege level (leakage of privileged state).
  - Mode-sensitive instructions behave differently in de-privileged mode (e.g. not causing a trap in a de-privileged mode): cannot guarantee correctness
- Possible Approaches:
  - Software Virtualization
  - Binary Translation

Software virtualization

- Guest executes on an interpreter instead of directly on a physical CPU.
- Interpreter can prevent leakage of privileged state and it can correctly implement non-trapping instructions.
- In essence, the interpreter separates virtual state (the VCPU) from physical state (the CPU).
- Interpretation ensures Fidelity and Safety, but fails to meet *Popek and Goldberg's Performance bar*:
  - the fetch- decode-execute cycle of the interpreter may burn hundreds of physical instructions per guest instruction.

Binary Translation

- VMware introduced in 1998 binary translation to allow the VMM to run in Ring 0 for isolation and performance, while moving the operating system to a user level ring with greater privilege than applications in Ring 3 but less privilege than the virtual machine monitor in Ring 0.
- Binary translation:
  - The VMM monitors the execution of guest OS;
  - Non-virtualizable instructions executed by a guest OS are replaced with (**translated to**) other instructions.
  - The OS runs unchanged, and this ensures that this direct execution mode is efficient.
- Binary translation can combine the semantic precision of interpretation with high performance, yielding an execution engine that meets all of Popek and Goldberg's criteria.
- VMMs built around a suitable binary translator can virtualize the x86 architecture and it is a VMM according to Popek and Goldberg.

Binary Translation

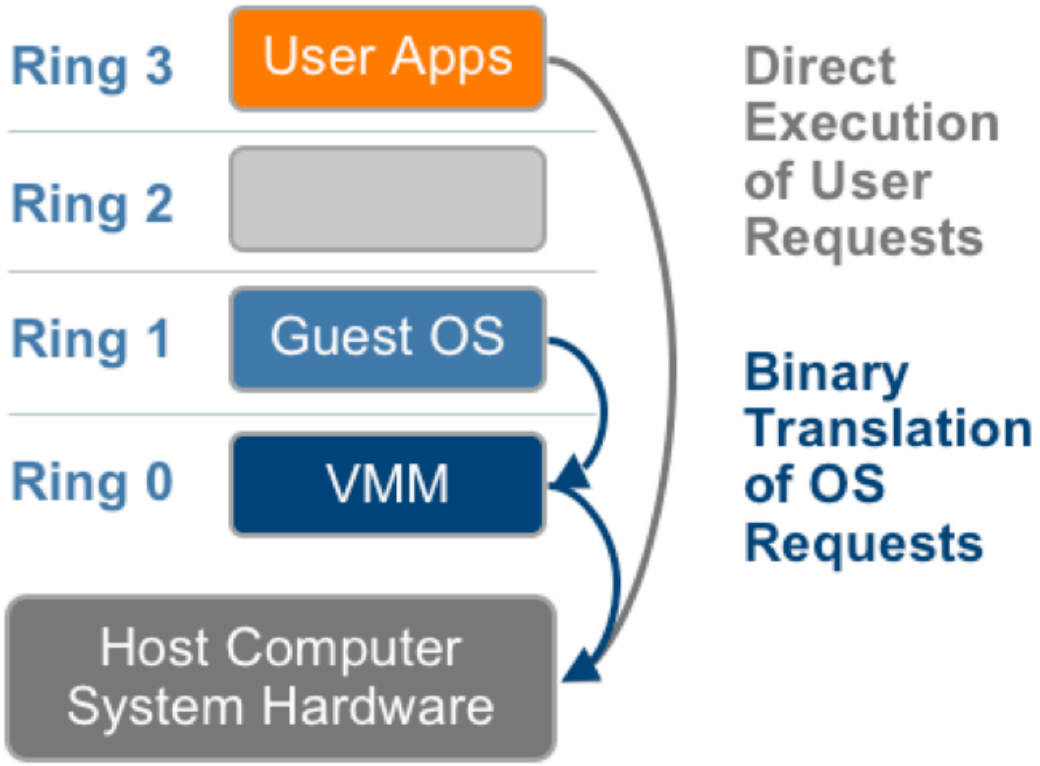


Figure 5 – The binary translation approach to x86 virtualization

"Understanding Full Virtualization, Paravirtualization, and Hardware Assist," VMWare White Paper (2008)

Binary Translation Properties

- ***Binary***: Input is binary x86 code, not source code.
  - A guest OS can run unchanged on the VMM as if it was running directly on the hardware platform. Each VM runs on an exact copy of the actual hardware.
- ***Dynamic***: Translation happens at runtime, interleaved with execution of the generated code.
  - "The hypervisor translates all OS instructions on the fly and caches the results for future use, while user level instructions run unmodified at native speed."
- ***On demand***: Code is translated only when it is about to execute.
- ***System level***: Translator makes no assumptions about the guest code. Rules are set by the x86 ISA.
- ***Subsetting***: the translator's input is the full x86 instruction set, including all privileged instructions. The output is a safe subset (mostly user mode instructions)
  - Binary translation rewrites parts of the code on the fly to replace sensitive but not privileged instructions with safe code to emulate the original instruction
- ***Adaptive***: translated code is adjusted in response to guest behavior changes to improve overall efficiency.

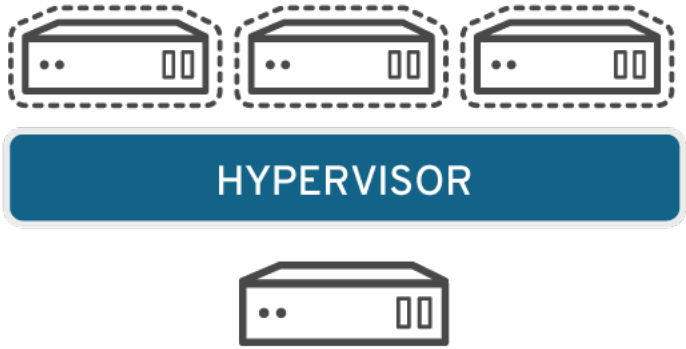
Modern Virtualization Technologies

Server Virtualization

University of Cyprus
Department of Computer Science

Server virtualization

- Server virtualization converts one physical server into multiple virtual machines. Each virtual server acts like a unique physical device, capable of running its own operating system (OS).
  - The physical server is called the **host**.
  - The virtual servers are called **guests** and behave like physical machines.
- Each virtualization system uses a different approach to allocate physical server resources to virtual server needs.
- Approaches:
  - **Full virtualization**
  - **Para-virtualization**
  - **OS-level virtualization**



Modern Virtualization Technologies: Server Virtualization

Full Virtualization

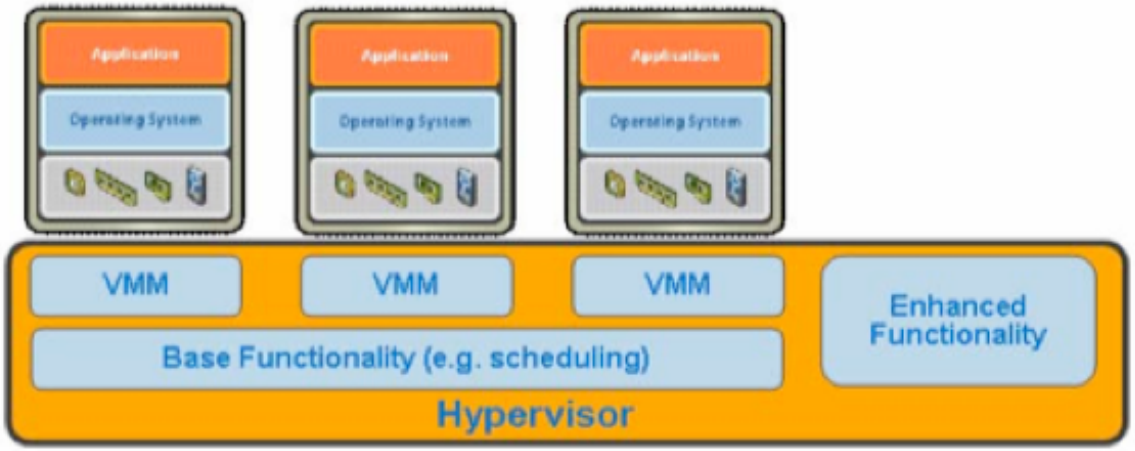
University of Cyprus
Department of Computer Science

Full virtualization

- Each virtual machine runs on an exact copy of the actual hardware.
- The guest OS:
  - is fully abstracted (completely decoupled) from the underlying hardware by the virtualization layer (VMM)
  - is not aware it is being virtualized and requires no modification.

Full virtualization VMMs

- Each VMM running on the hypervisor implements the virtual machine hardware abstraction and is responsible for running a guest OS:
  - Each guest server runs on its own OS
  - You can even have one guest running on Linux and another on Windows.
- As virtual servers run applications, the hypervisor relays resources from the physical machine to the appropriate virtual server:
  - Each VMM has to partition and share the CPU, memory and I/O devices to successfully virtualize the system



HOW CAN THIS WORK?

The **hypervisor** translates all operating system instructions on the fly and caches the results for future use, while user level instructions run unmodified at native speed.

WHAT HAPPENS BEHIND THE SCENES?

University of Cyprus
Department of Computer Science

Binary Translation of non-virtualizable (sensitive) instructions, direct execution of nonsensitive instructions and provision by each VMM of virtual services available on the physical system (BIOS, virtual devices, virtual memory management)

Requires no hardware assist or OS assist to virtualize sensitive and primitive instructions.

Full virtualization

- Relies on the **hypervisor**, which:
  - Interacts directly with the physical server's CPU and disk space.
  - Monitors the physical server's resources.
  - Serves as a platform for the virtual servers' operating systems.
  - Keeps each virtual server completely independent and unaware of the other virtual servers running on the physical machine.
- Hypervisors have their own processing needs, which means that the physical server must reserve some processing power and resources to run the hypervisor application.
  - This can impact overall server performance and slow down applications.

Full virtualization: Pros and Cons

- Advantages:
  - No hardware assistance,
  - No modifications of the guest OS
  - Best isolation and security for virtual machines,
  - Simplifies migration and portability as the same guest OS instance can run virtualized or on native hardware.
- Disadvantages:
  - Speed of execution

Modern Virtualization Technologies: Server Virtualization

Paravirtualization

University of Cyprus
Department of Computer Science

Paravirtualization

- Operating System-assisted Virtualization or Paravirtualization proposed to:
  - Cope with hardware architectures that cannot be virtualized easily.
  - To improve performance of virtualization.
  - To present a simpler VMM interface.
- Demands that:
  - The guest OS uses only instructions that can be virtualized, and run on the VMM.
  - Guest OS code is ported for individual hardware platforms.
- The term is used to describe the Denali, Xen, L4, TRANGO, VMware, Wind River and XtratuM hypervisors.

Paravirtualization

- Presents a software interface to the VMs which is similar, yet not identical, to the underlying hardware–software interface.
  - Intent of the modified interface: reduce portion of the guest's execution time on operations which are substantially more difficult to run in a virtual environment compared to a non-virtualized environment.
- A conventional OS distribution that is not paravirtualization-aware cannot be run on top of a para-virtualizing VMM.
- If the OS cannot be modified, components may be available that enable many of the significant performance advantages of paravirtualization:
  - Xen Windows GPLPV project provides paravirtualization-aware device drivers, intended to be installed into a Microsoft Windows virtual guest running on the Xen hypervisor.

Para-virtualization implementation

- Manually port guest OS to a modified (higher level) ISA, by:
  - Replacing non-virtualizable instructions with explicit hypervisor calls (hypercalls) that communicate directly with the hypervisor.
  - Providing specially defined 'hooks' to 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)
- The "higher-level" ISA results in:
  - Reduction of number of traps
  - Removal of non-virtualizable instructions
  - Removal of "messy" ISA features
  - Simplicity: easier to modify the guest OS to enable paravirtualization than to implement binary translation

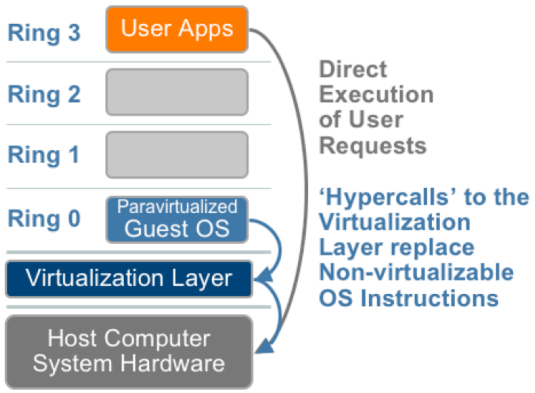


Figure 6 – The Paravirtualization approach to x86 Virtualization

Para-virtualization: Pros & Cons

- Advantages:
  - Generally outperforms pure virtualisation & binary re-writing
  - A successful paravirtualized platform may allow the VMM to be simpler (by relocating execution of critical tasks from the virtual domain to the host domain), and/or reduce the overall performance degradation of machine execution inside the virtual guest: lower virtualization overhead; para-virtualization hypervisor doesn't need as much processing power to manage the guest operating systems
  - Each guest OS is already aware of the demands the other operating systems are placing on the physical server.
  - The entire system works together as a cohesive unit.
  - Faster execution - performance is generally very close to running bare-metal, non-virtualized operating systems.
- Drawbacks:
  - Poor portability and compatibility (cannot support unmodified OS)
  - Significant engineering effort
  - Needs to be repeated for each guest-ISA-hypervisor combination
  - Para-virtualised guests must be kept in sync with native evolution
  - Requires source

Modern Virtualization Technologies: Server Virtualization

Hardware-assisted virtualization

University of Cyprus
Department of Computer Science

Hardware Assisted Virtualization

- A new CPU execution mode feature that allows the VMM to run in a new root mode below ring 0.
- Privileged and sensitive calls are set to automatically trap to the hypervisor, removing the need for either binary translation or paravirtualization“
- Advantage: even faster execution
- Examples: Intel VT-x, Xen 3.x

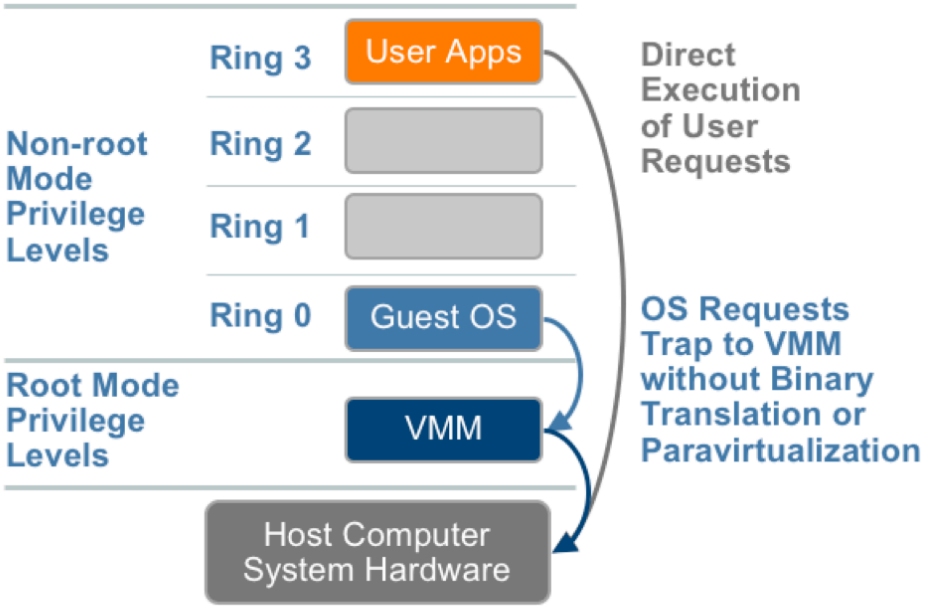


Figure 7 – The hardware assist approach to x86 virtualization

Summary of x86 Virtualization Techniques

	Full Virtualization with Binary Translation	Hardware Assisted Virtualization	OS Assisted Virtualization / Paravirtualization
Technique	Binary Translation and Direct Execution	Exit to Root Mode on Privileged Instructions	Hypercalls
Guest Modification / Compatibility	Unmodified Guest OS Excellent compatibility	Unmodified Guest OS Excellent compatibility	Guest OS codified to issue Hypercalls so it can't run on Native Hardware or other Hypervisors Poor compatibility; Not available on Windows OSes
Performance	Good	Fair Current performance lags Binary Translation virtualization on various workloads but will improve over time	Better in certain cases
Used By	VMware, Microsoft, Parallels	VMware, Microsoft, Parallels, Xen	VMware, Xen
Guest OS Hypervisor Independent?	Yes	Yes	XenLinux runs only on Xen Hypervisor VMI-Linux is Hypervisor agnostic

2007

Modern Virtualization Technologies: Server Virtualization

Operating System-level virtualization

University of Cyprus
Department of Computer Science

OS-level virtualization

- An OS-level virtualization approach doesn't use a hypervisor at all.
- Instead, the virtualization capability is part of the host OS, which performs all the functions of a fully virtualized hypervisor.
- The biggest limitation of this approach is that all the guest servers must run the same OS.
- Each virtual server remains independent from all the others, but you can't mix and match operating systems among them.
- Because all the guest operating systems must be the same, this is called a homogeneous environment.

OS-level virtualization

- Happens at the OS kernel: the kernel allows the existence of multiple isolated user space instances.
- Such instances, called containers (LXC, Solaris containers, Docker), Zones (Solaris containers), virtual private servers (OpenVZ), partitions, virtual environments (VEs), virtual kernels (DragonFly BSD), or jails (FreeBSD jail or chroot jail).
- Containers may look like real computers from the point of view of programs running in them, but:
  - A computer program running on an ordinary OS can see all resources (connected devices, files and folders, network shares, CPU power, quantifiable hardware capabilities) of that computer.
  - However, programs running inside of a container can only see the container's contents and devices assigned to the container.

Modern Virtualization Technologies

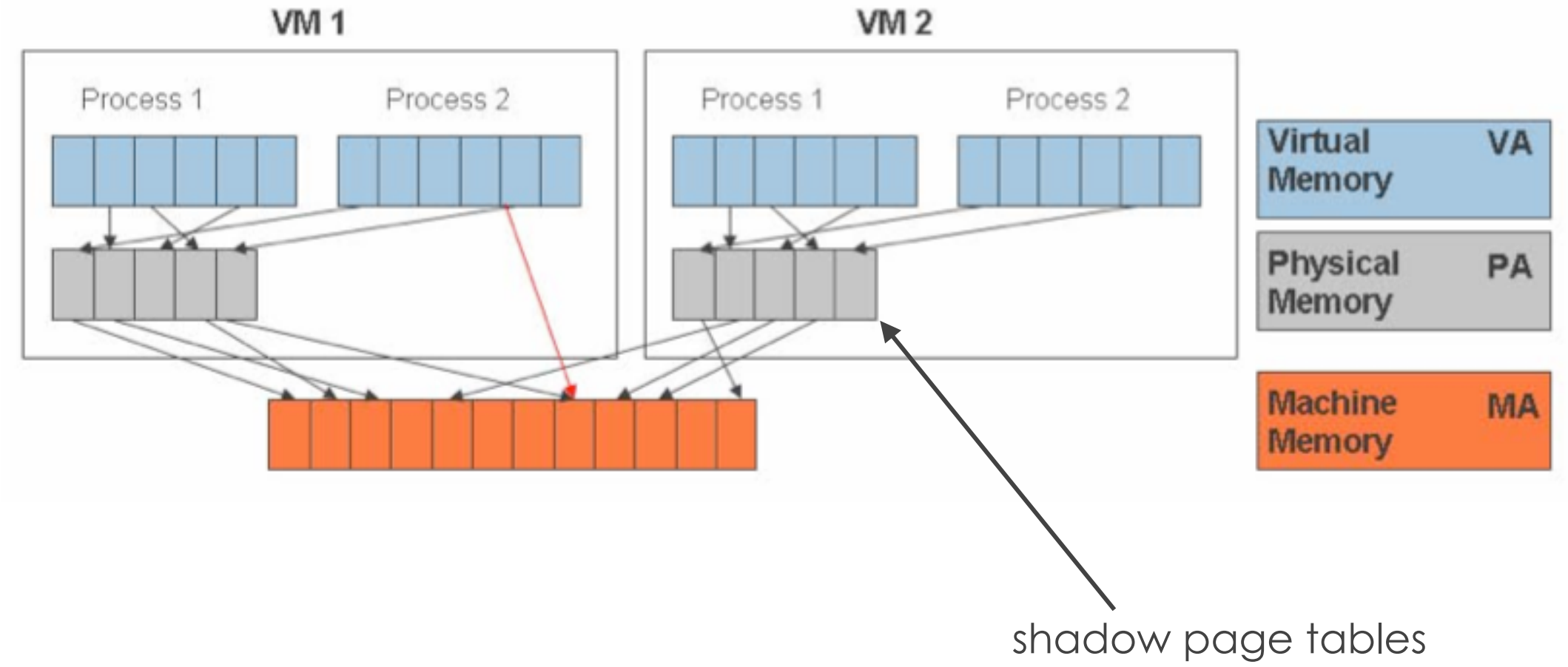
Other Virtualization Types

University of Cyprus
Department of Computer Science

Memory Virtualization

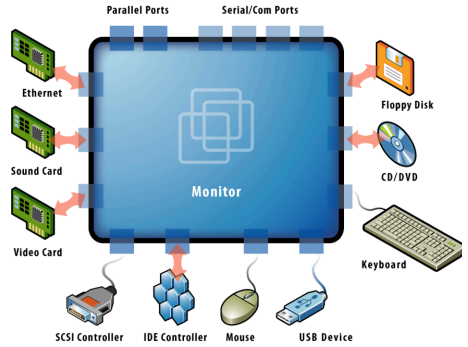
- Sharing the physical system memory and dynamically allocating it to virtual machines.
- Virtual machine memory virtualization is very similar to the virtual memory support provided by modern OSs:
  - Applications see a contiguous address space that is not necessarily tied to the underlying physical memory in the system.
  - The OS keeps mappings of virtual page numbers to physical page numbers stored in page tables.
- All modern x86 CPUs include a memory management unit (MMU) and a translation lookaside buffer (TLB) to optimize virtual memory performance.
- To run multiple VMs on a single system, another level of memory virtualization is required:
  - Virtualize the MMU to support the guest OS.
  - Guest OS continues to control the mapping of virtual addresses to the guest memory physical addresses, but the guest OS cannot have direct access to the actual machine memory.
  - The VMM is responsible for mapping guest physical memory to the actual machine memory, and it uses shadow page tables to accelerate the mappings.

Memory Virtualization



Device & I/O Virtualization

- Manages the routing of I/O requests between virtual devices and the shared physical hardware.
- Software based I/O virtualization and management, enables a rich set of features and simplified management.
  - Virtual NICs and switches create virtual networks between virtual machines without the network traffic consuming bandwidth on the physical network.
  - The hypervisor virtualizes the physical hardware and presents each virtual machine with a standardized set of virtual devices as seen in the Figure.
  - These virtual devices effectively emulate well-known hardware and translate the virtual machine requests to the system hardware



Modern Virtualization Technologies

Hypervisors Recap

University of Cyprus
Department of Computer Science

Hypervisors: Key Requirements

- Fidelity. Software on the VMM executes identically to its execution on hardware, except for timing effects.
- Performance. An overwhelming majority of guest instructions are executed by the hardware without the intervention of the VMM.
- Safety. The VMM manages all hardware resources.

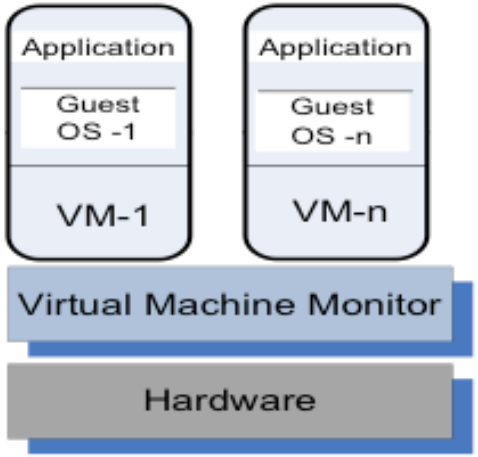
Popek & Goldberg, "Formal requirements for virtualizable third generation architectures", CACM July1974

Code portability and Binary Translation

- Binaries created by a compiler for a specific ISA and a specific operating systems are not portable
- It is possible, though, to compile a HLL program for a virtual machine (VM) environment where portable code is produced and distributed and then converted by *binary translators* to the ISA of the host system
- A dynamic binary translation converts blocks of guest instructions from the portable code to the host instruction set and leads to a significant performance improvement, as such blocks are cached and reused

Virtual Machine Monitors - Hypervisors

- **Virtual Machine Monitor** (VMM) or **Hypervisor**: software layer that implements virtualization
  - Separates the physical resources from the virtual environments running upon them
  - Translates calls to the interfaces of the guest environment to the interfaces of the host environment.
  - Divides physical resc manages their map simultaneous use by environments.



University of Cyprus
Department of Computer Science

Hypervisors: What do they do?

- Control resources:
  - Partition hardware
  - Schedule guests
  - Mediate access to shared resources
  - Allow several operating systems to run concurrently on a single hardware platform
- Allow:
  - Live migration - the movement of a virtual server from one platform to another
  - System modification while maintaining backward compatibility with the original system
  - Enforce isolation among the systems, thus security

Courtesy of Gernot Heiser, UNSW, Australia

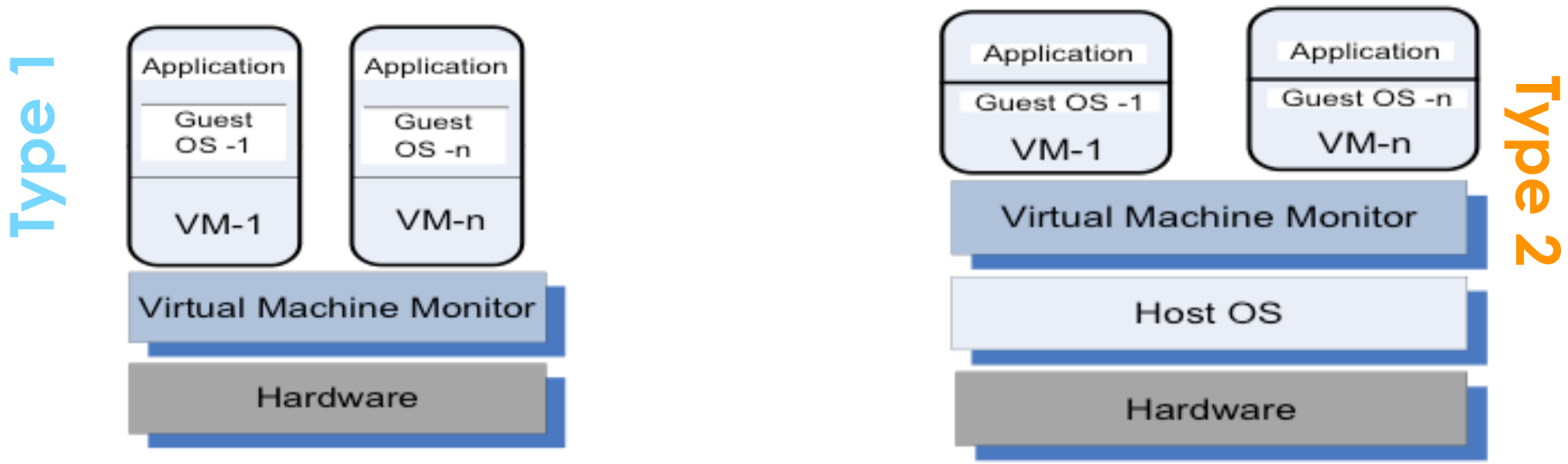
VMM Virtualizes the CPU & Memory

- A VMM:
  - Traps the privileged instructions executed by a guest OS and enforces the correctness and safety of the operation
  - Traps the interrupts and dispatches them to the individual guest operating systems
  - Controls the virtual memory management
  - Maintains a shadow page table for each guest OS and replicates any modification made by the guest OS in its own shadow page table. This shadow page table points to the actual page frame and it is used by the Memory Management Unit (MMU) for dynamic address translation.
  - Monitors the system performance and takes corrective actions to avoid performance degradation. For example, the VMM may swap out a VM to avoid thrashing.

VMM Implications

- Hypervisors execute in privileged mode
- Guest software executes in unprivileged mode
- Privileged instructions in guest cause a trap into hypervisor
  - Hypervisor interprets/emulates them
- VMM can have extra instructions for hypercalls *(hypervisor calls)*
- When a program running on a virtual environment issues an instruction that requires additional resources from the physical environment, the hypervisor relays the request to the physical system and caches the changes— at close to native speed.

Hypervisor Types



- **Type 1** (**bare metal**, native): run directly on the hardware (as a lightweight OS) and support multiple virtual machines and OS.
  - Most popular in production environments due to the low overhead.
  - Citrix/Xen Server, Denali, VMware ESXi and Microsoft Hyper-V
- **Type 2** (**hosted**) VM - runs the virtualization layer as an application on top of a host operating system (e.g., user-mode Linux)
  - Microsoft Virtual PC, Oracle Virtual Box, VMware Workstation, Oracle Solaris Zones, VMware Fusion, Oracle VM Server for x86.
  - Ideal option for personal use due to low cost and ease of installation .

Examples of Hypervisors

Name	Host ISA	Guest ISA	Host OS	guest OS	Company
Integrity VM	*x86*-64	*x86*-64	HP-Unix	Linux,Windows HP Unix	HP
Power VM	Power	Power	No host OS	Linux, AIX	IBM
z/VM	z-ISA	z-ISA	No host OS	Linux on z-ISA	IBM
Lynx Secure	*x86*	*x86*	No host OS	Linux, Windows	LinuxWorks
Hyper-V Server	*x86*-64	*x86*-64	Windows	Windows	Microsoft
Oracle VM	*x86*, *x86*-64	*x86*, *x86*-64	No host OS	Linux, Windows	Oracle
RTS Hypervisor	*x86*	*x86*	No host OS	Linux, Windows	Real Time Systems
SUN xVM	*x86*, SPARC	same as host	No host OS	Linux, Windows	SUN
VMware EX Server	*x86*, *x86*-64	*x86*, *x86*-64	No host OS	Linux, Windows Solaris, FreeBSD	VMware
VMware Fusion	*x86*, *x86*-64	*x86*, *x86*-64	MAC OS *x86*	Linux, Windows Solaris, FreeBSD	VMware
VMware Server	*x86*, *x86*-64	*x86*, *x86*-64	Linux, Windows	Linux, Windows Solaris, FreeBSD	VMware
VMware Workstation	*x86*, *x86*-64	*x86*, *x86*-64	Linux, Windows	Linux, Windows Solaris, FreeBSD	VMware
VMware Player	*x86*, *x86*-64	*x86*, *x86*-64	Linux Windows	Linux, Windows Solaris, FreeBSD	VMware
Denali	*x86*	*x86*	Denali	ILVACO, NetBSD	University of Washington
Xen	*x86*, *x86*-64	*x86*, *x86*-64	Linux Solaris	Linux, Solaris NetBSD	University of Cambridge

Type-1 VMM Characteristics

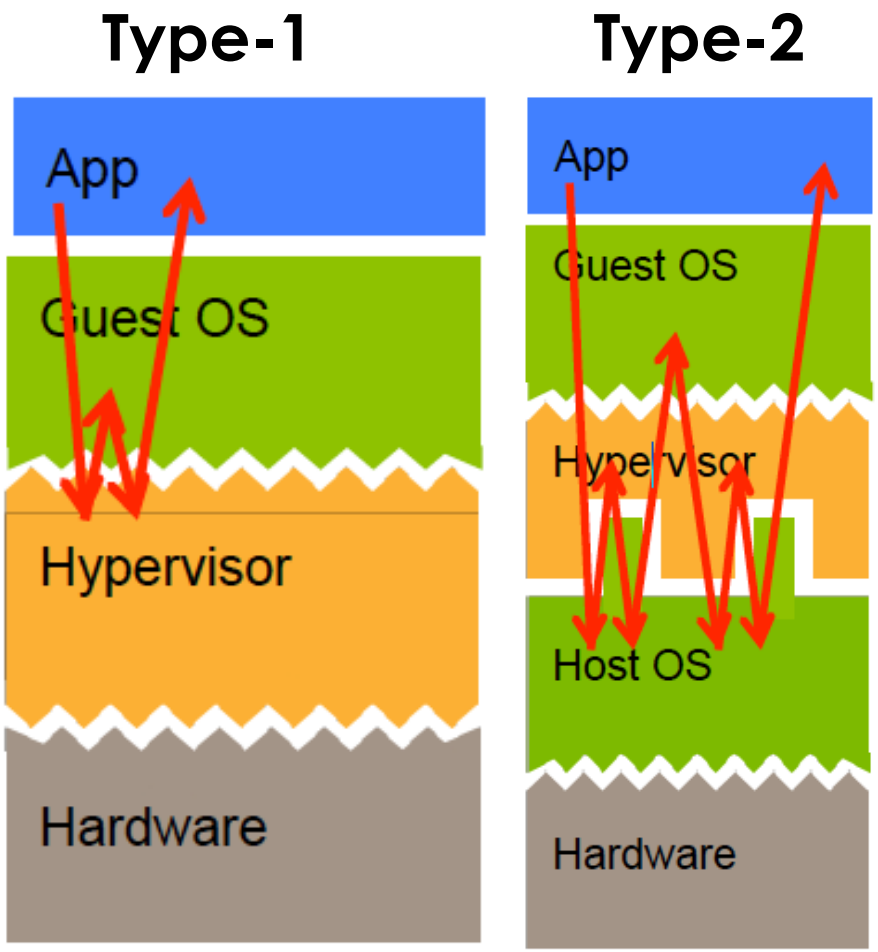
- Efficient, good performance:
  - Benefits from hardware support for virtualization (VT-x, VT-A), which helps the hypervisor perform the intensive tasks required to manage the virtual resources of the computer.
  - Without hardware support, the hypervisor would have to handle the intensive tasks required for virtualization on its own resulting to:
    - Overall performance drop
    - Restricted the number of guest VMs that could be hosted on a computer
- Very secure because:
  - Are much simpler and better specified than traditional operating systems. Example - Xen has approximately 60,000 lines of code; Denali has only about half: 30,000
  - Have considerably reduced security vulnerabilities as they expose a much smaller number of privileged functions. For example, Xen VMM has 28 hypercalls while Linux has 100s of system calls

Type-2 VMM Characteristics

- Typically installed on an existing Operating System and support a wide range of hardware.
- Rely on the host OS to undertake operations like managing calls to the CPU, managing network resources, managing memory and storage.
- Make use of hardware acceleration technologies, when available.
- Fall back on software emulation if the support isn't available on the physical host system.

Type-2 vs Type-1 VMM

- You can host the VMM beside native apps
  - Sandbox untrusted apps
  - Convenient for running alternative OS on desktop
  - Leverage host drivers
- Less efficient
  - Double node switches
  - Double context switches
  - Host not optimised for exception forwarding



Courtesy of Gernot Heiser, UNSW, Australia

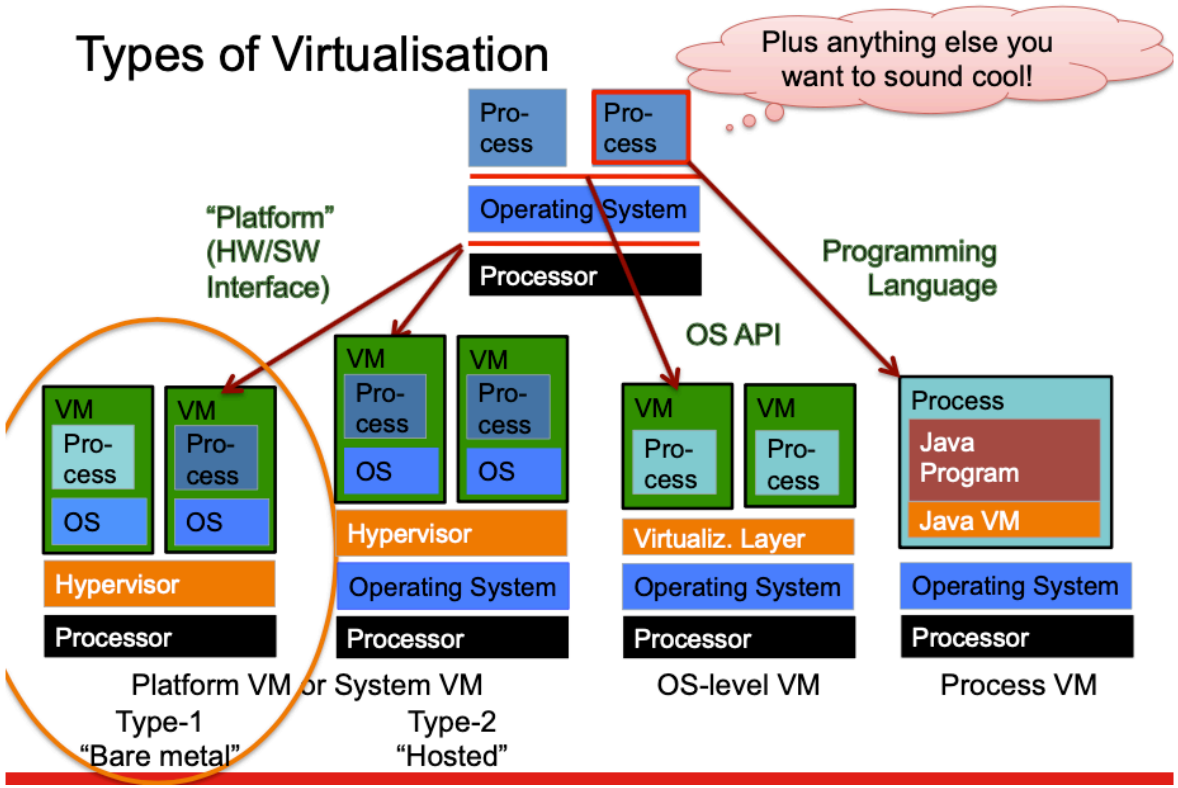
Kernel-based Virtual Machine-KVM

- Type-2 hypervisor and open source virtualization technology built into Linux®.
- KVM lets you turn Linux into a type-1 (bare-metal) hypervisor that allows a host machine to run multiple, isolated virtual machines (guests).
- All hypervisors need some operating system-level components—such as a memory manager, process scheduler, input/output (I/O) stack, device drivers, security manager, a network stack, and more—to run VMs.
  - KVM has all these components because it's part of the Linux kernel.
  - Every VM on KVM is implemented as a **regular Linux process**, scheduled by the standard Linux scheduler, with dedicated virtual hardware like a network card, graphics adapter, CPU(s), memory, and disks.

VM Management Frameworks

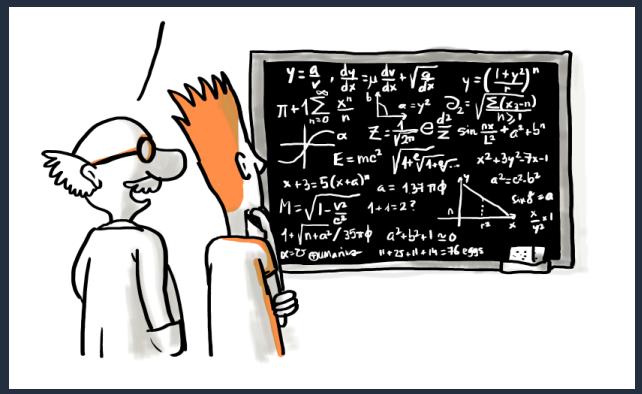
- Different types of virtualization technologies have management frameworks that enable VMs and applications to be deployed and managed at data center scale:
  - Commercial offerings like vCenter
  - Open source frameworks like OpenStack, CloudStack.
  - Kubernetes and Docker Swarm are recent container management frameworks.

Virtualization Types



Courtesy of Gernot Heiser, UNSW, Australia

Summary



- Review of basic Operating Systems' concepts of relevance to virtualization: core abstractions, layering, libraries, application binary interface, security and privilege management, protection rings, running in kernel vs. user mode.
- Introduction to Virtualization and discussion of different virtualization types.
- Discussed the concepts of server virtualization, virtual machines, and virtual machine monitors/hypervisors.
- Explained differences between Type-1 (bare metal) and Type-2 (hosted) VMMs.
- Discussed security issues and concerns with server virtualization.
- Discussed the problem of mapping VMs to physical machines.