# On virtualization

**Runtimes for concurrency and distribution**

Tullio Vardanega, tullio.vardanega@unipd.it

Academic year 2021/2022

# Abstraction (what is)

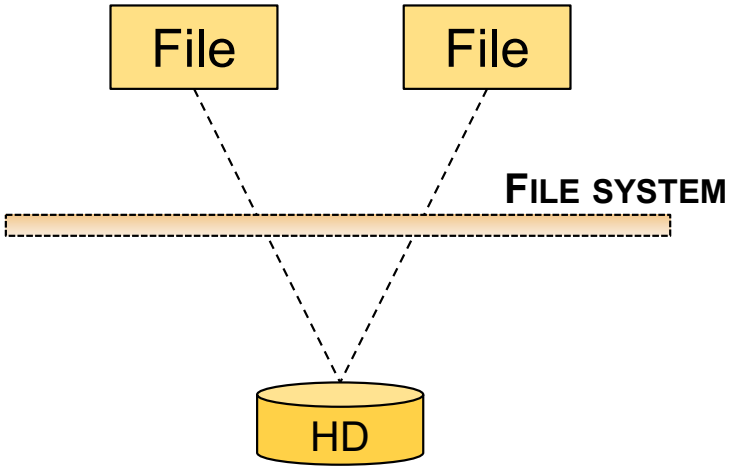
- Hiding details of an entity's implementation to simplify the view of it as offered to the user
  - **Example 1**: exposing a procedure instead of the stack required to make it "live", realizes an abstraction
  - **Example 2**: in UNIX (and then in Linux), everything is a file, specialized as needed, with a common interface
- Keywords
  - ***Information hiding***, ***well-defined interface***
- Weakness
  - The public interface of the abstraction is fragile in the face of changes that break its implementation

# Virtualization (what is)

- Providing a logical view (abstract interface) of an entity, **and** preserving it across changes in the underlying execution environment
- Virtualization adds all of the "adaptation harness" needed to preserve the base abstraction over variations in the underlying substrate
  - **Example**: exposing UNIX-like files in an NTFS environment
- Keyword
  - ***Encapsulation***
- Strength
  - Virtualization sits above abstraction, adding value to it by always preserving its interface contract

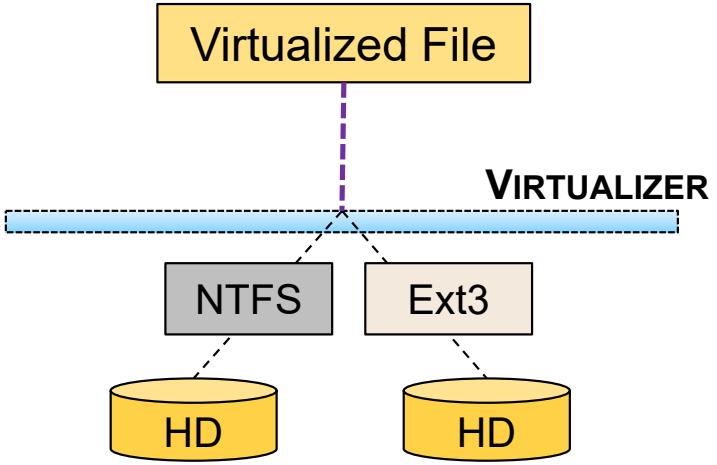
# Example /1

## Abstraction




One logical abstraction provides a simple view of some functionality while hiding its concrete implementation

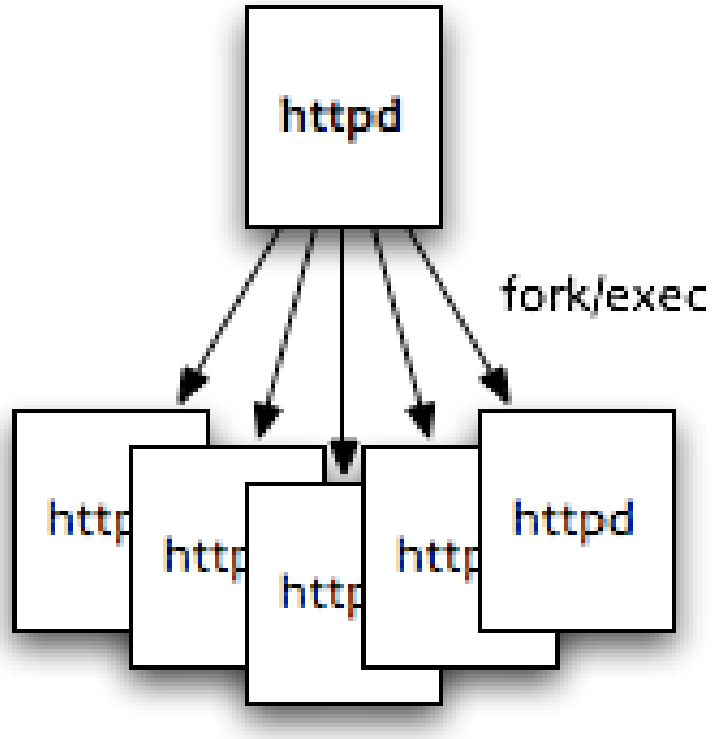
## Virtualization



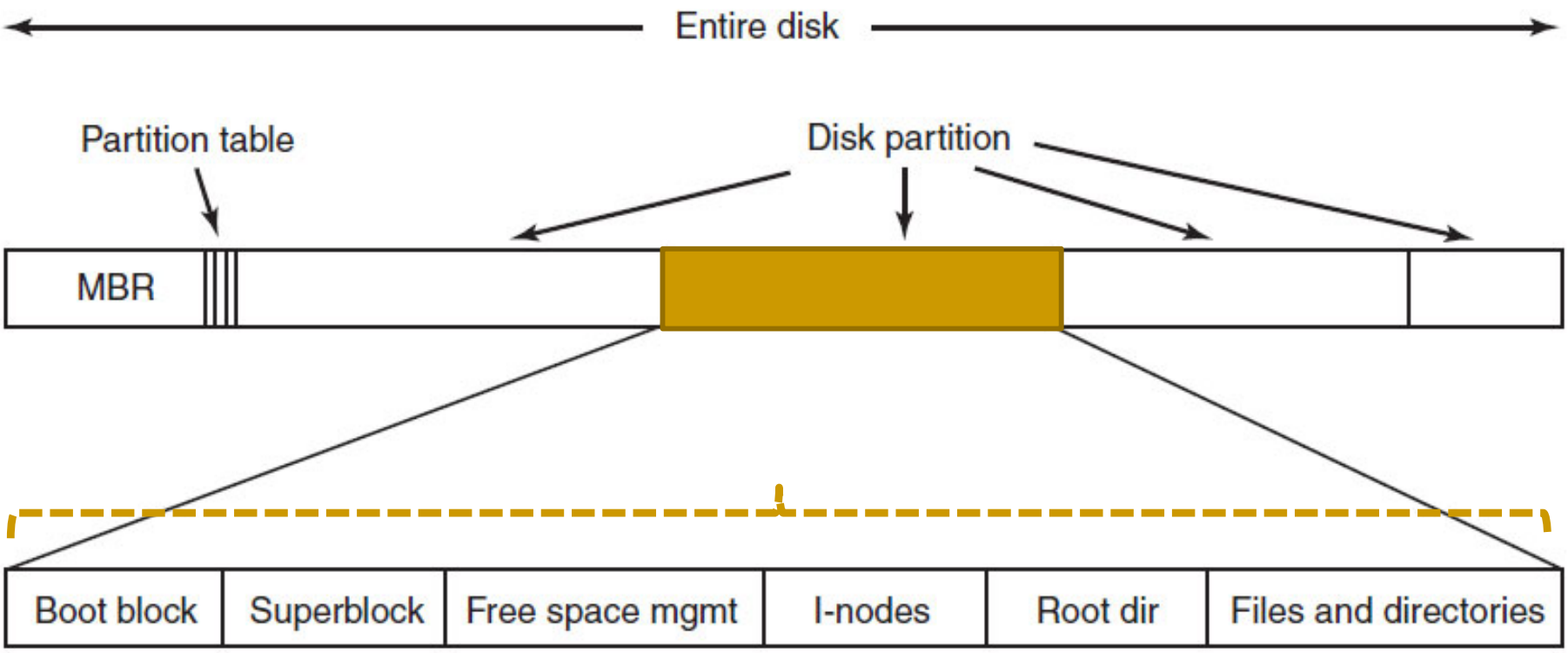

One and the same interface is provided regardless of what the underlying infrastructure has to offer

# Example /2

- The UNIX abstraction of “process” lends itself to virtualize multi-programming



# Abstracting the Operating System /1




- ***Boot block***: procedure to initialize the OS (make it "live")
- ***Superblock***: descriptor of the whole partition (in the form of a file system)
- ***I-nodes***: list of all file-system-object descriptors (*i-node*)

# Abstracting the Operating System /2

- Knowing the abstraction of a specific OS (its implementation at run time) allows treating it as an entity “from the outside of it”
  - Copying it
  - Moving it
  - Deleting it
  - Stopping and resuming its execution at will
- All that this requires is a way to “understand” its descriptors and their life cycle

# Some history /1

- The ‘60s, the time of the *mainframe*
- HW resources are scarce and costly
- Virtualization allows transparent sharing of them across multiple competing processes
  - *Time sharing* virtualizes access to the CPU
  - *Virtual memory* overcomes the size limitations of the RAM
- Virtualization becomes one of the founding principles of computing

# Some history /2

- The ’70s, from mainframes to minicomputers
- The scarcity of HW resources is alleviated by general-purpose multi-programmed Oss
- The arrival of Personal Computers makes HW plentiful
- Everybody is satisfied and the urge to push virtualization further fades away
- Good read: “The Game”, A. Baricco, 2018

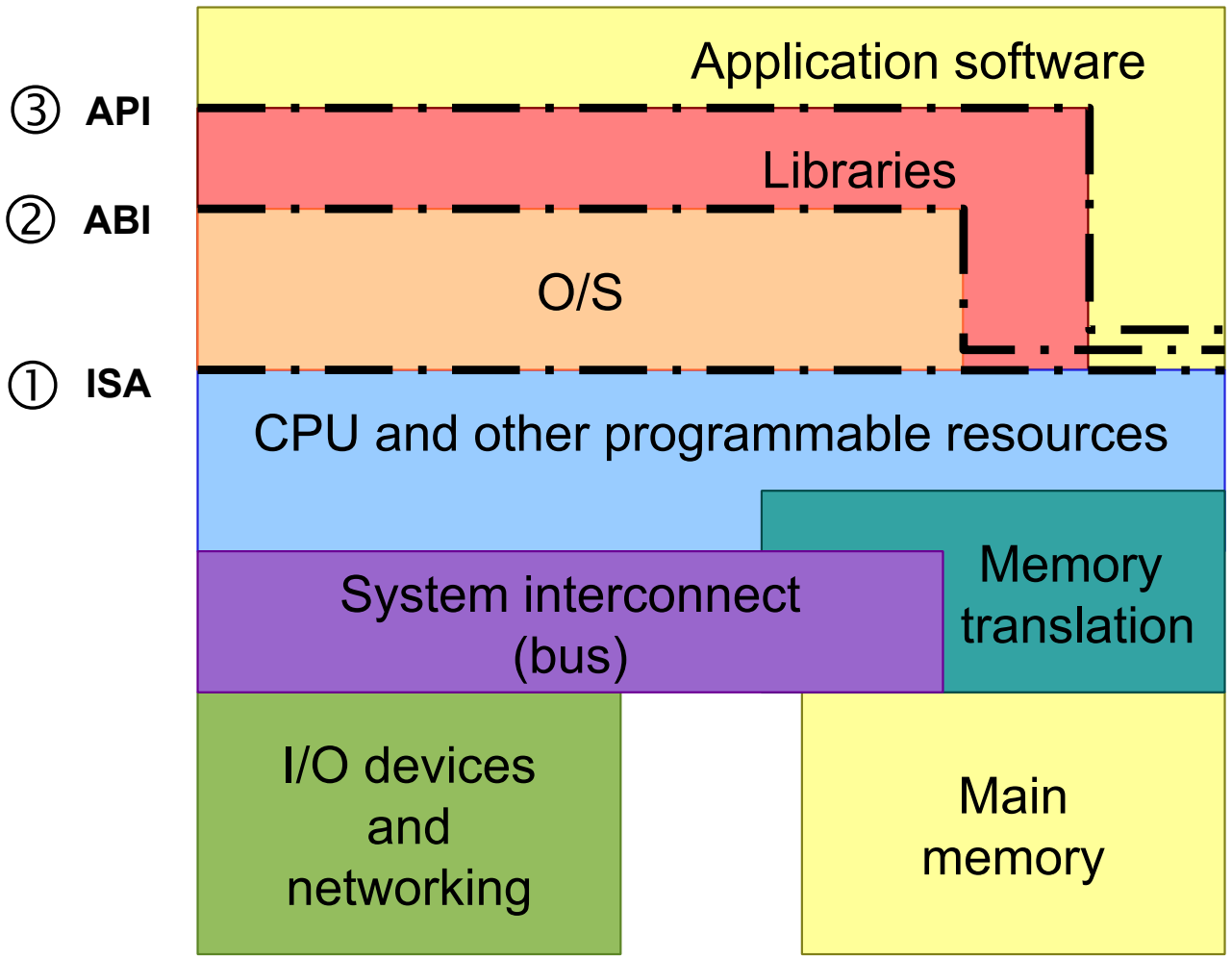
# Some history /3

- The early ’90s, commercial and scientific interest for massively parallel computing
  - **Example**: weather prediction ☺
- This needs specialized HW, made short-lived by commercial competition
  - **Example**: the Transputer (DOI: 10.1145/255129.255192), the building block of a highly composable general-purpose massively parallel processor
- Interest in virtualization resurrects, to ease the porting of applications across hardware evolutions
  - 10/02/1998: VMware Inc. is founded (https://www.vmware.com/timeline.html)

# Some history /4

- Late ‘90s, all industry begins to run on an array of digital services
  - The simultaneous decrease in the unitary cost of computer HW yields a surge in *heterogeneity* (classic law of demand)
- The increasing (vertical) needs of industry are met by an increasing number of *dedicated* servers
  - More independent heterogenous servers means higher maintenance cost for less average use of HW resources
- Interest in virtualization resurrects, to seek cost-reducing “consolidation” (aka rationalization)
  - Sharing HW across application servers

# Architecture and interfaces /1

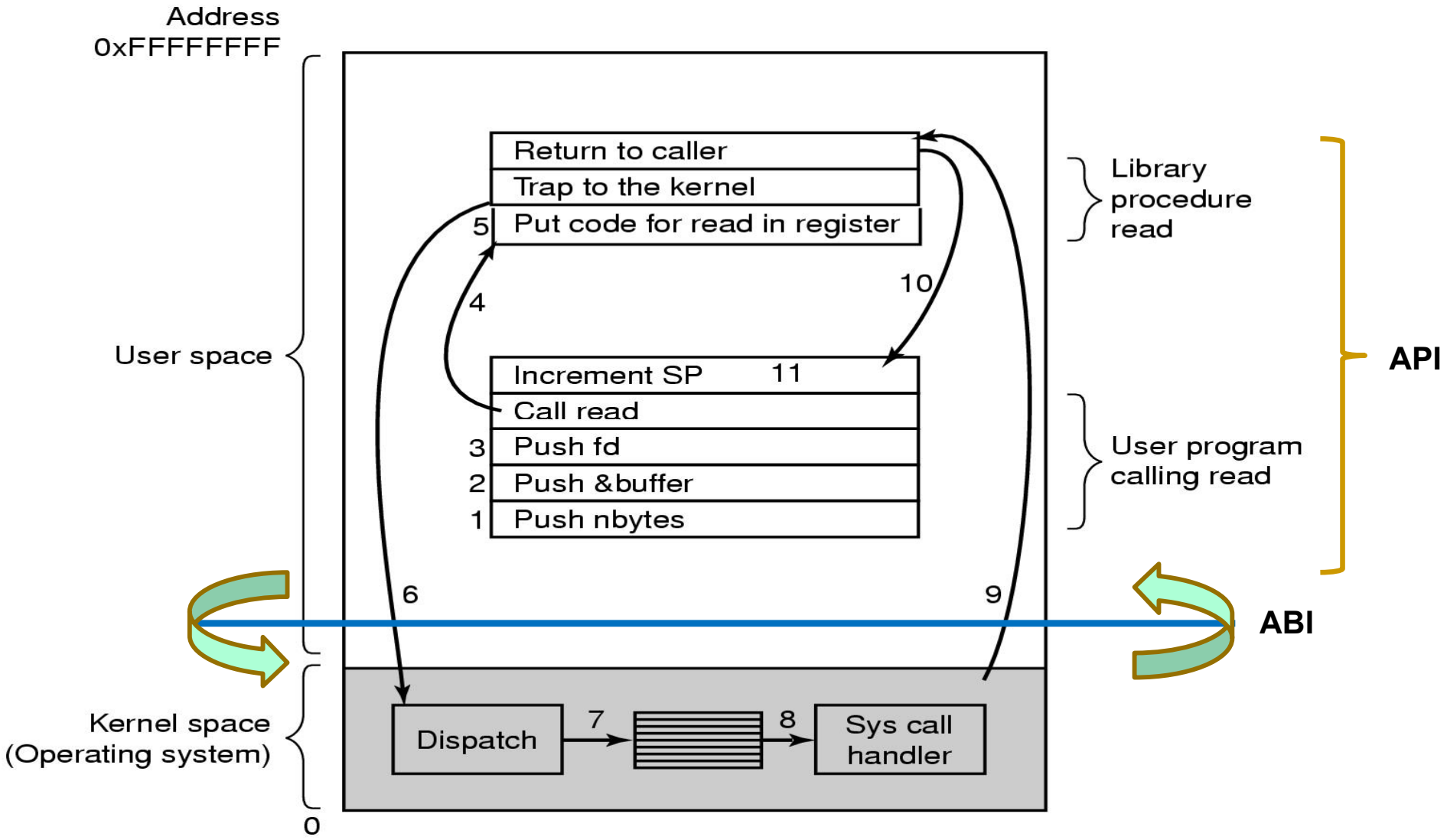



***Application Programming Interface***: call conventions within one and the same implementation language (how the compiler wants data to be presented in function calls)

***Application Binary Interface***: call conventions across heterogeneous object files (how the execution may use the processor memory and registers)

HW

# Where does the ABI operate?

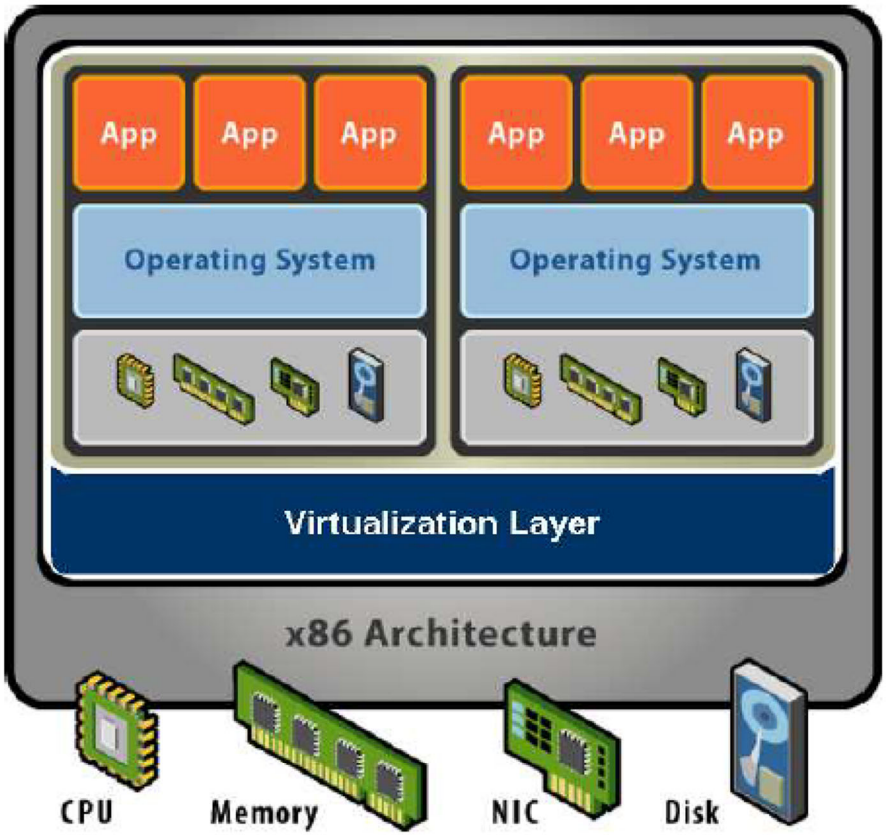


# Architecture and interfaces /2

- Changes in the processor HW may cause the ISA to change too
- Changes in the ISA affect the OS
  - And of course all compiler backends that target it
- The extent of the change may also affect the ABI
  - And possibly the API as well
- To preserve the value of applications we need to augment abstraction with virtualization
  - At which level should we act?

# A possible ultimate goal

Focus shifts on isolation

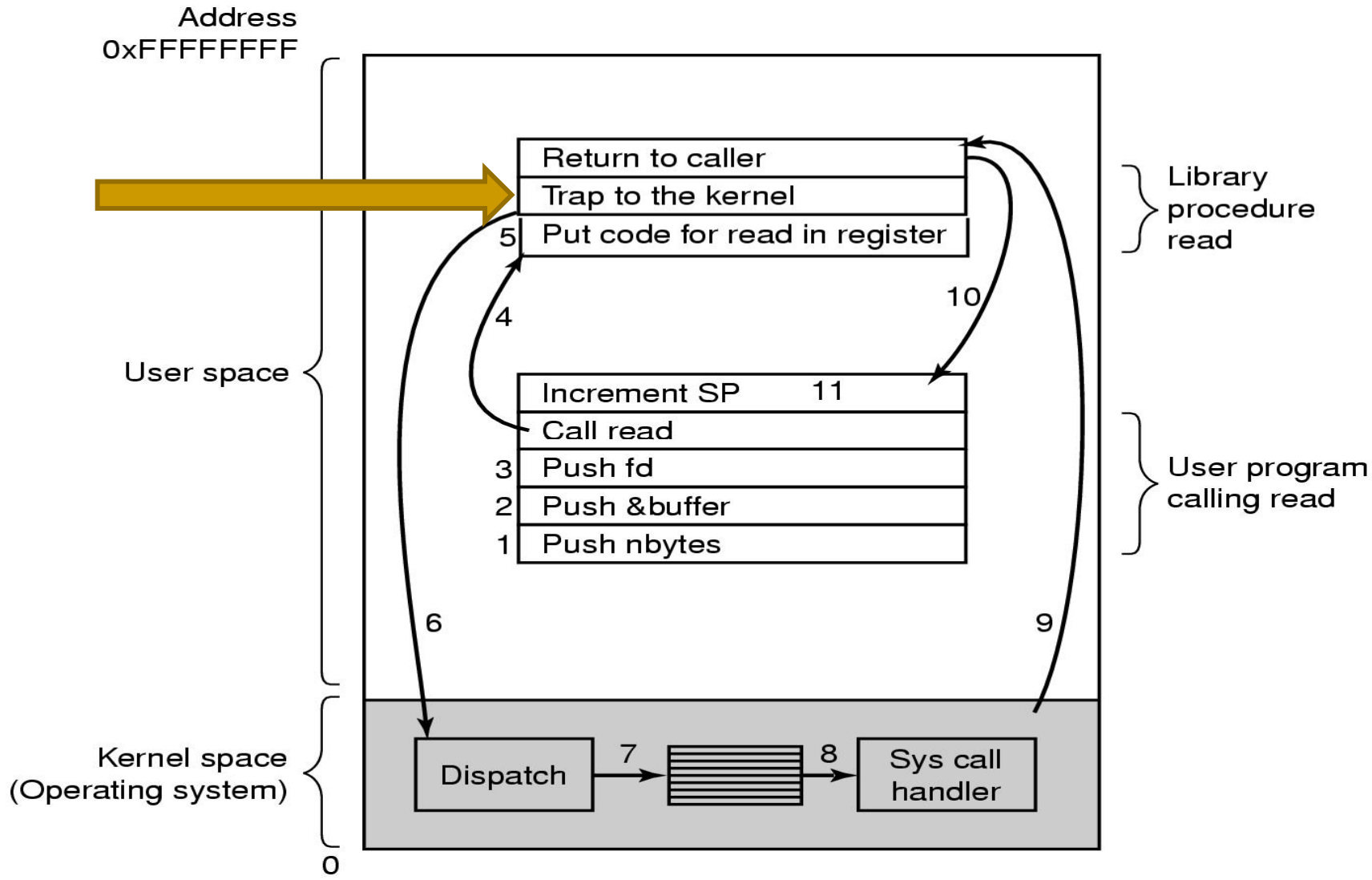



- Hardware-Level Process Abstraction: CPU, memory, chipset, I/O devices, etc.
  - Virtual NIC instead of sockets
  - Virtual disk instead of file system
  - Hardware state becomes software state
- Virtualization Software
  - Hardware and software decoupled

# The basics of virtualization /1

- Since the end of the ’60s, processor execution was associated with *levels of privilege*
  - The ISA was accessible to the executing program in subsets (aka “**protection rings**”)
  - The inner the ring the greater the privilege
- Any attempt to execute outside of the assigned level of privilege is trapped by the processor HW
  - HW trap, a form of predefined exception
- The raising of the program’s level of privilege may be requested by specialized instructions
  - SW trap (and the associated “return from trap”)

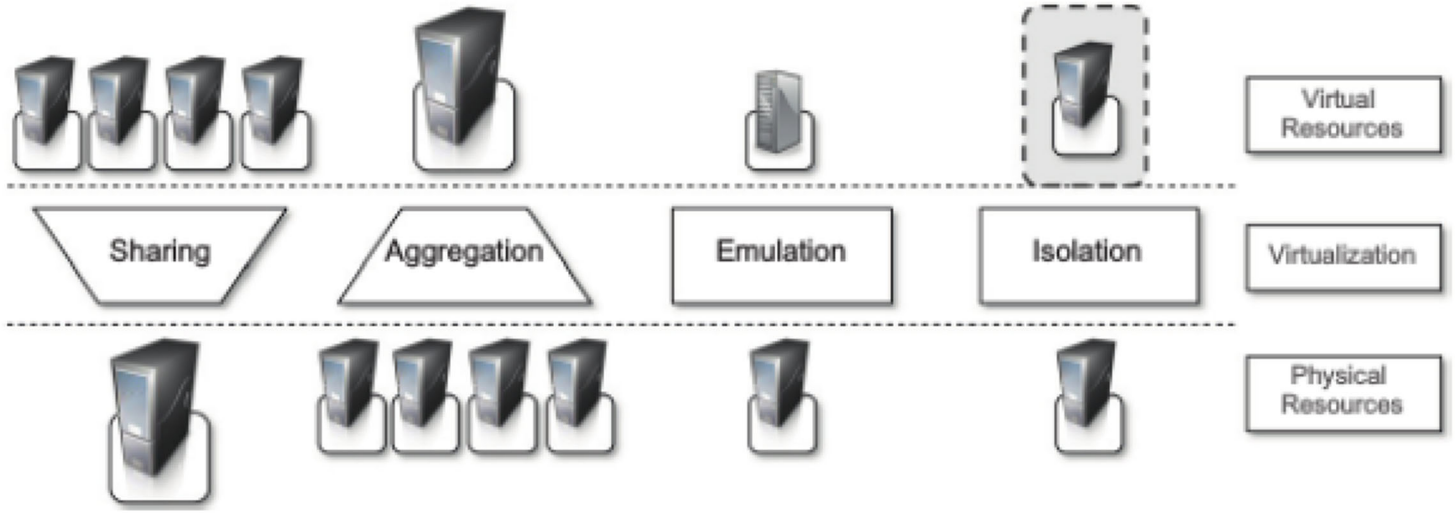
# The basics of virtualization /2



# A taxonomy of virtualization /1

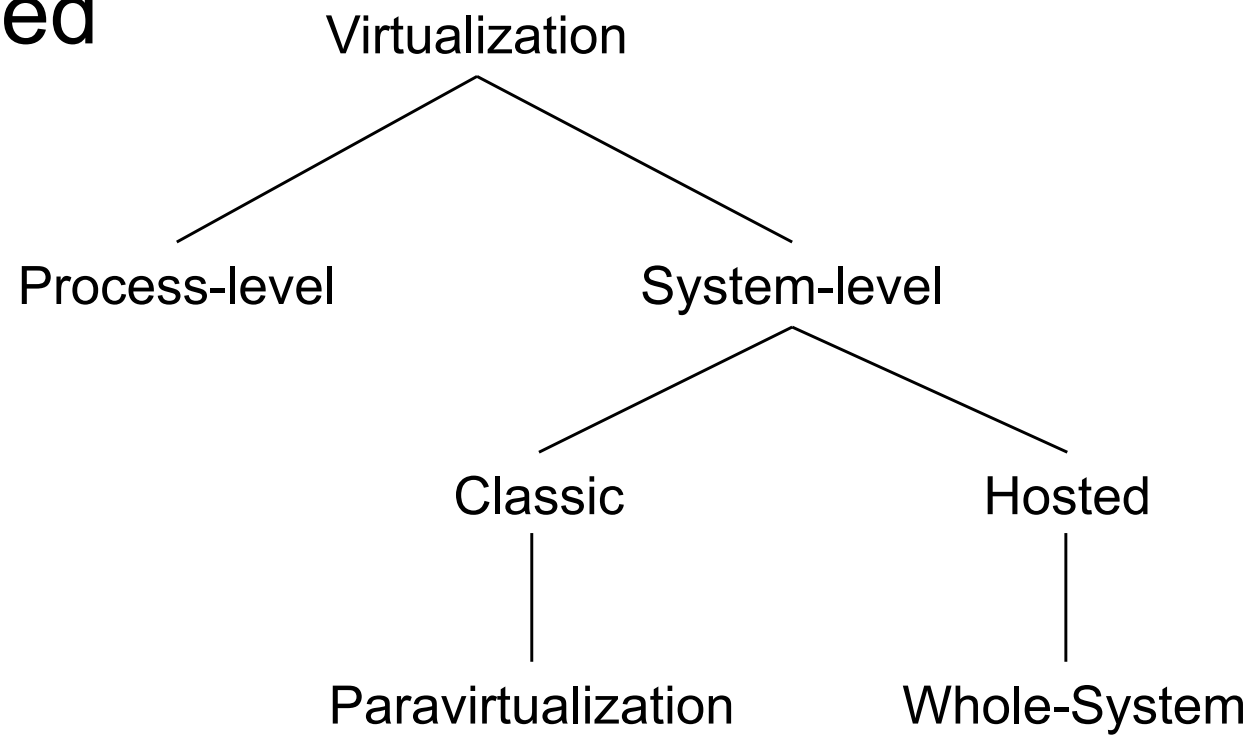
Virtualization allows the creation of a secure, customizable, and isolated execution environment for running applications without affecting other users' applications

- various functions enabled by managed execution
  - sharing (e.g. server consolidation)
  - aggregation (e.g. cluster management software)
  - emulation (e.g arcade-game emulator)
  - isolation ⇒ no interference between multiple guest

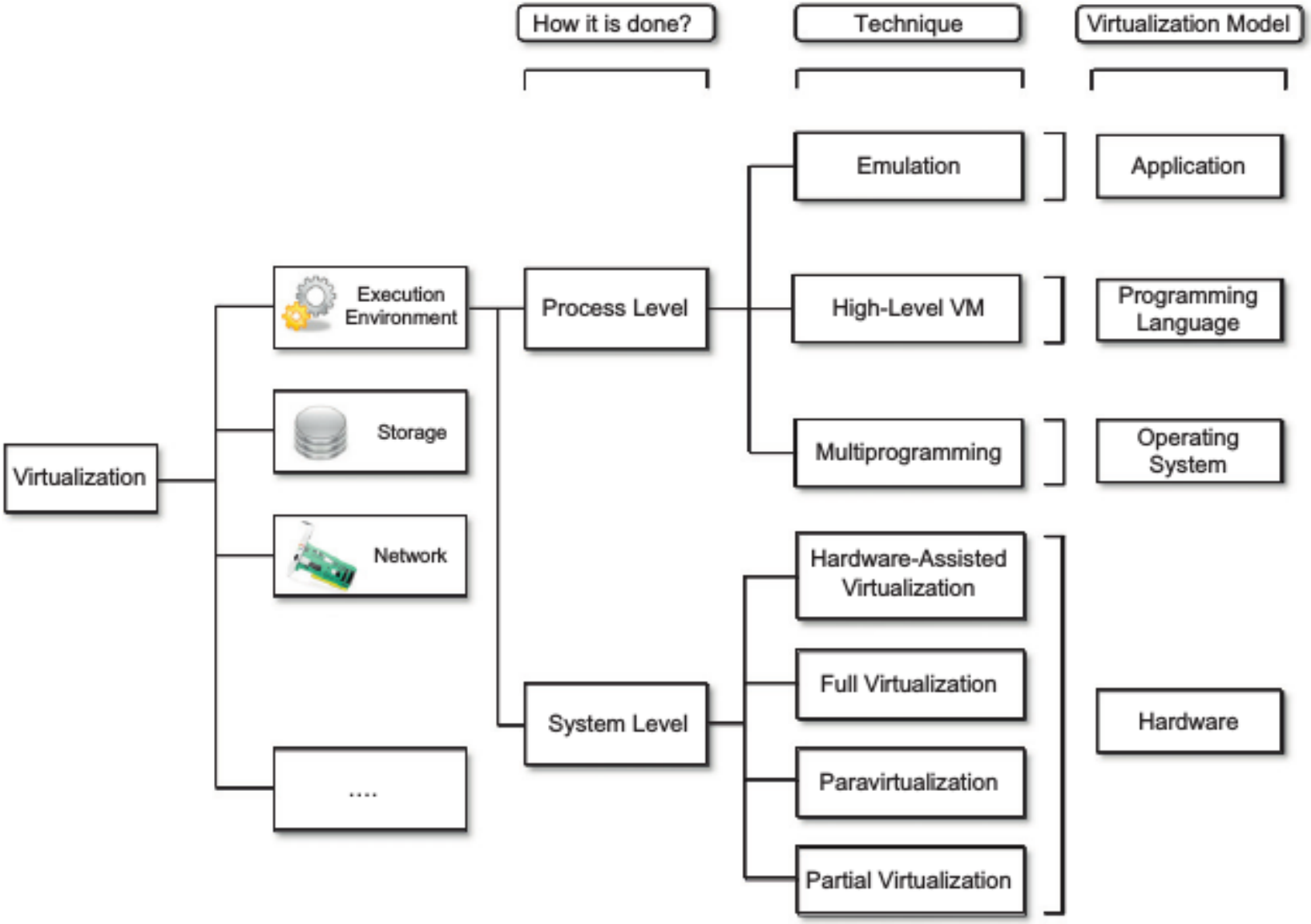


# A taxonomy of virtualization /2

- Another important classification follows the level of abstraction under which virtualization is realized

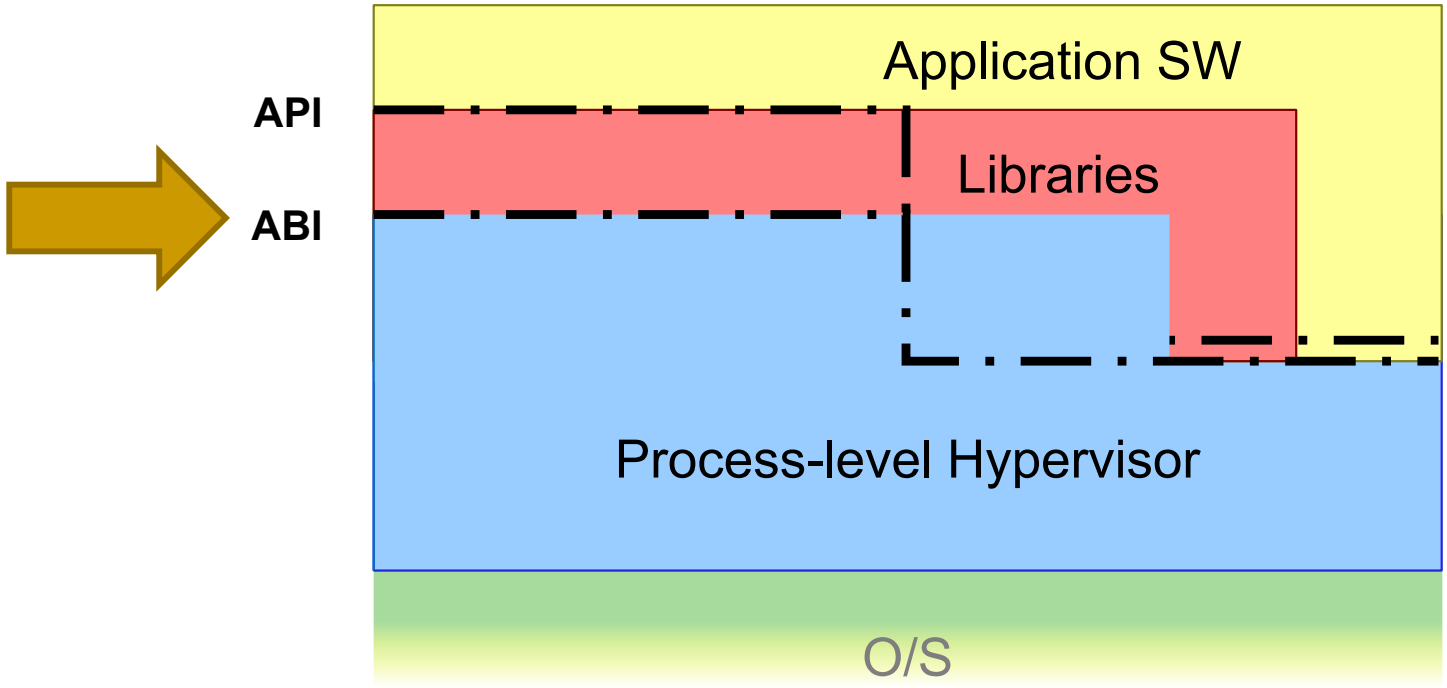


# A taxonomy of virtualization /3



# Process-level Virtualization /1

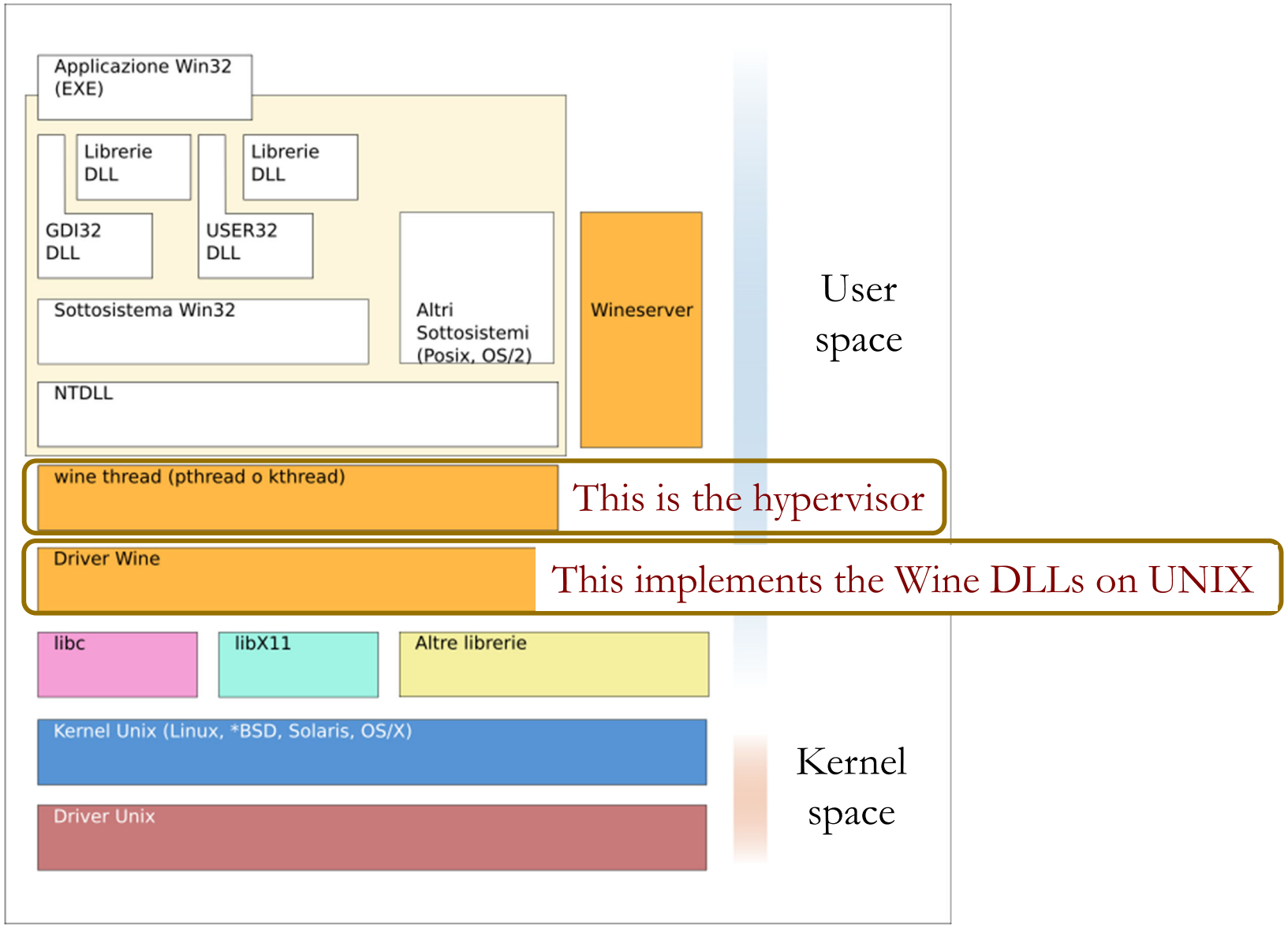
- The hypervisor runs as a process on the host OS, and provides its own ABI for virtualized applications to use
  - Reminiscent of the multiprogramming model of UNIX



# Process-level Virtualization /2

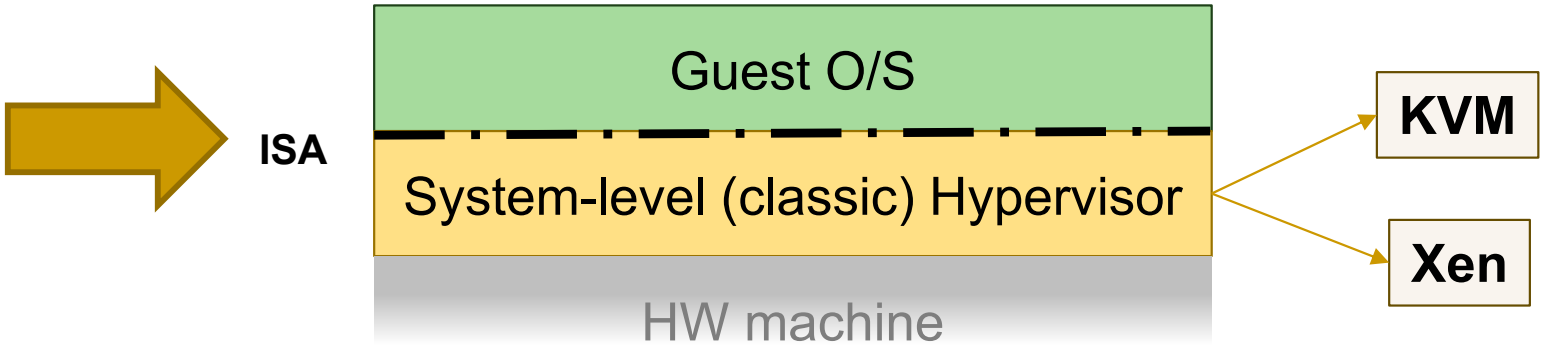
- Process-level virtualized applications enjoy
  - Virtual memory, which they do *not* know is virtual
  - Virtualized IO, which they do *not* know is virtual
  - Access to CPU, multi-programmed by the host OS
  - Exactly like a normal process
- The execution of the application program in this model may be
  - *Direct* if its binary is ISA-conformant (e.g., Wine)
  - *Interpreted*, otherwise (e.g., Java Virtual Machine)

# Process-level Virtualization: Wine



# System-level (classic) Virtualization

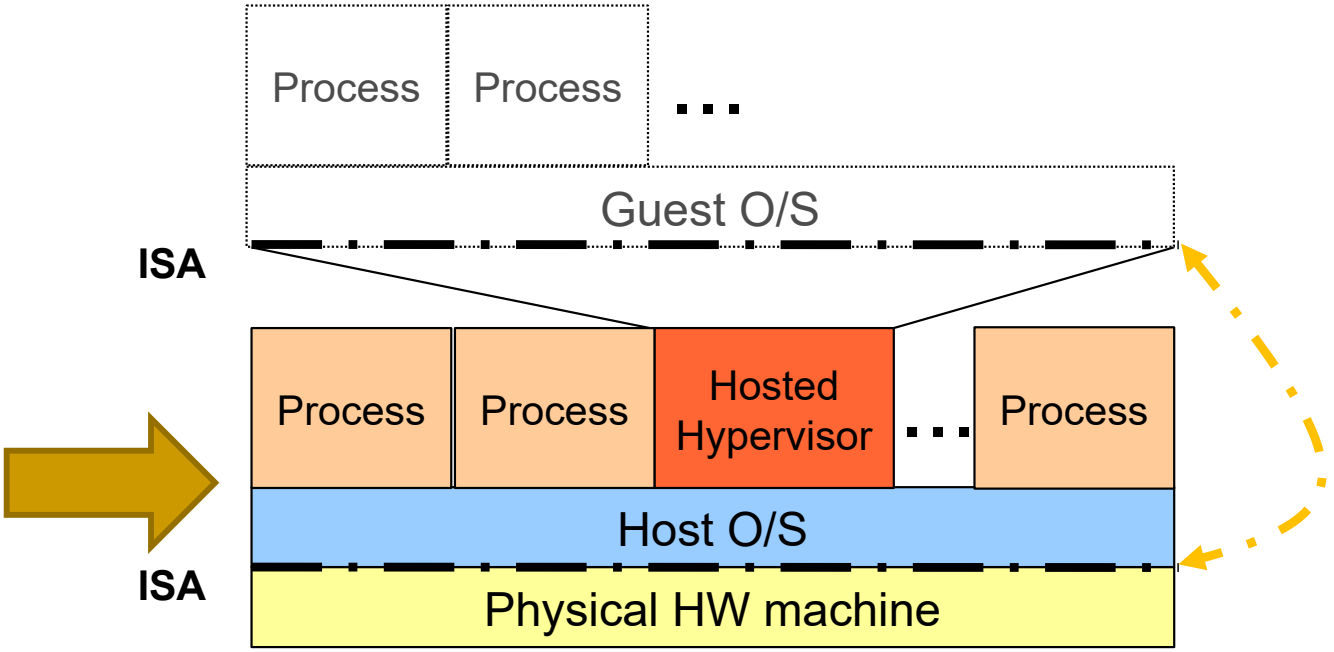
- The hypervisor provides guests with a software ISA
- The guest is a full OS, which however is rendered unable to take control of the processor resources
  - Effectively, the guest OS is stripped of its privileges (**de-privileged**)




http://drsalbertspijkers.blogspot.com/2017/05/kvm-kernel-virtual-machine-or-xen.html

# System-level (hosted) Virtualization

- The hypervisor is a normal process on the host OS
  - As such, it rents the compute resources that it requires
  - The underlying ISA is the same for all executables
- The goal is to preserve the value of (guest) applications, at the cost of inevitable performance decay

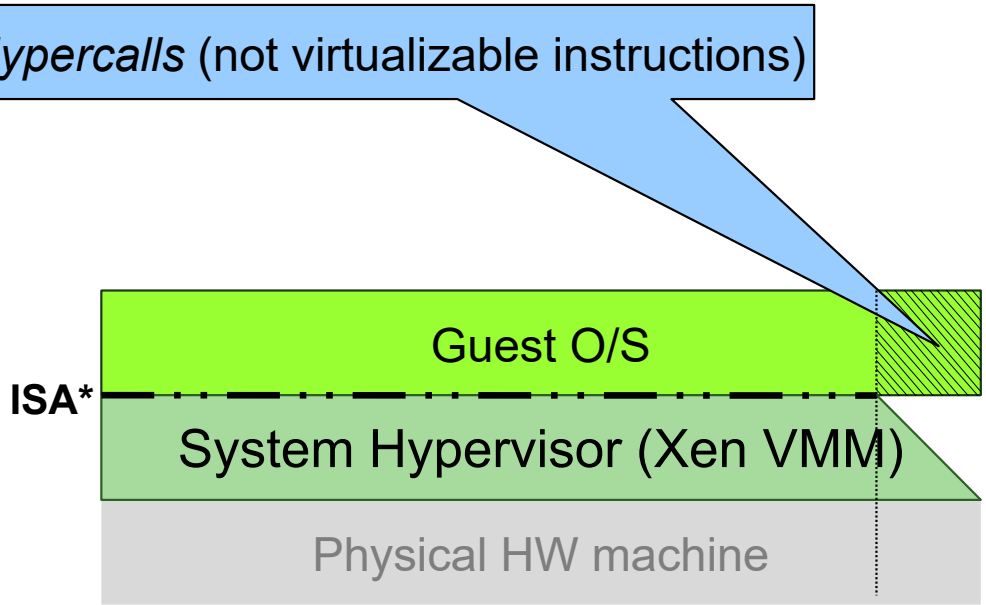


# Para-virtualization /1

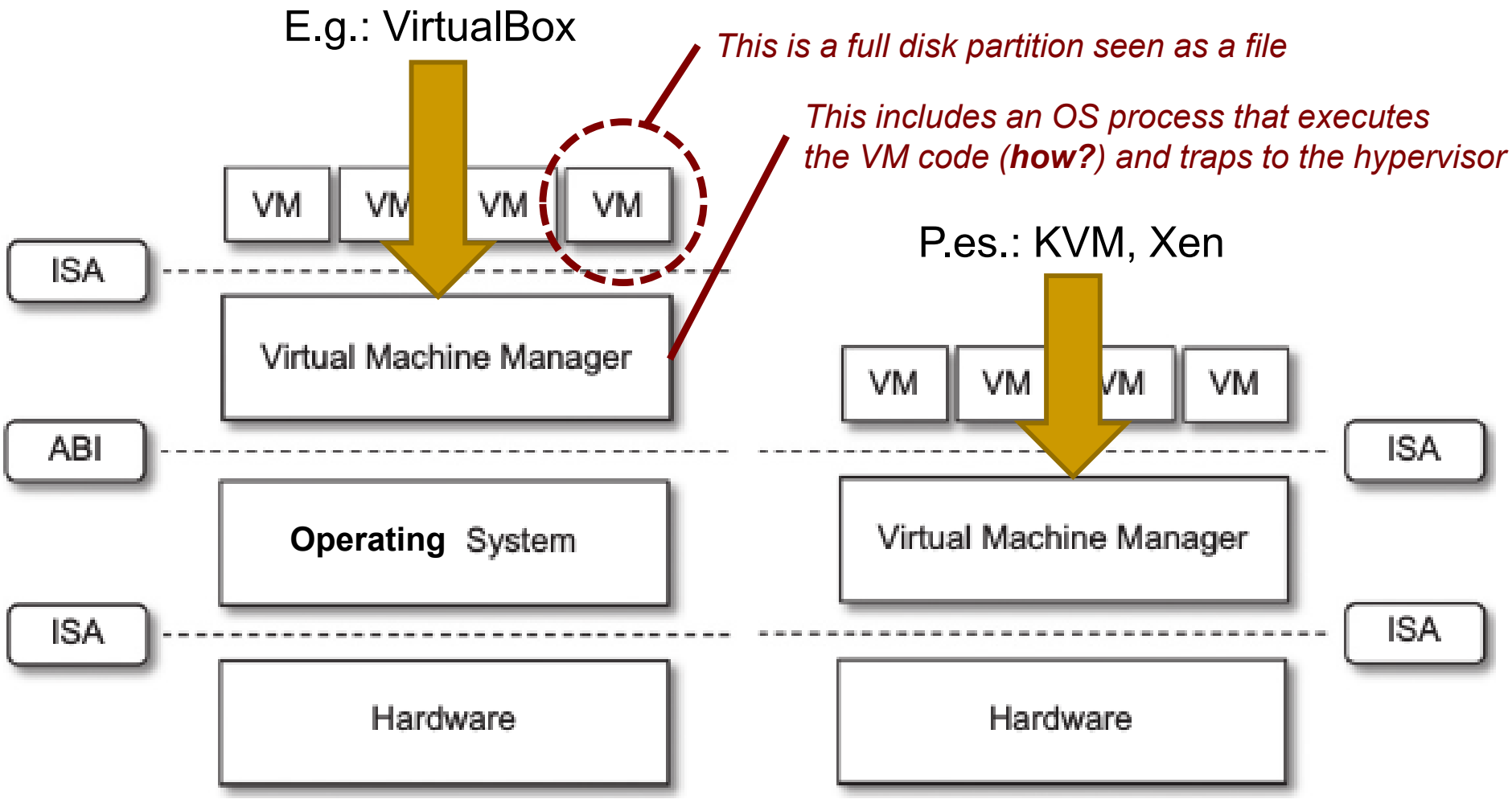
- System-level virtualization rests on the ability to trap trespasses of privilege rings
  - This allows the hypervisor to keep full control of the processor resources against attempts by the guest OS
- HW traps drain performance, which processor makers dislike
  - The support for system-level virtualization ceases
- The Intel architectures begin introducing machine instructions that cannot be virtualized
  - They are “outside” of privilege rings
- This fools traditional hypervisors

# Para-virtualization /2

- The remedy requires extending the ISA with a **hypercall-API** interface that allows hypervisors to retain resource control without the overhead of trapping
  - The resulting performance overhead was proven negligible
  - Guest OSs had to be modified to use those instructions



# Overview


E.g.: VirtualBox
This is a full disk partition seen as a file
This includes an OS process that executes the VM code (**how?**) and traps to the hypervisor
VM
VM
VM
VM
ISA
Virtual Machine Manager
ABI
Operating System
ISA
Hardware
P.es.: KVM, Xen
VM
VM
VM
VM
ISA
Virtual Machine Manager
ISA
Hardware