On virtualization

Runtimes for concurrency and distribution

Tullio Vardanega, tullio.vardanega@unipd.it

Academic year 2020/2021

Abstraction (what is)

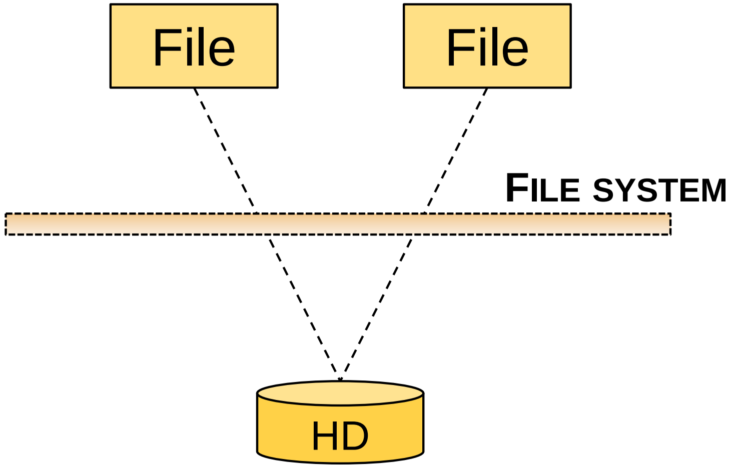
- Hiding details of an entity’s implementation to “purify” the view of it offered to the user
  - Expose an abstract data type instead of the (complex) machinery that realizes it
  - **Example**: in UNIX/Linux, every entity is represented as a file, so that they all have the same public interface
- Keywords
  - ***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, is preserving it across changes in the underlying execution machinery
- Virtualization adds to the abstraction all of the “adaptation layer” necessary to preserve the original interface stipulations over variations in the underlying substrate
  - **Example**: exposing a UNIX-like file system over an NTFS file system
- Keyword
  - ***Encapsulation***
- Strength
  - Virtualization sits above abstraction, adding value to it by always preserving its interface contract

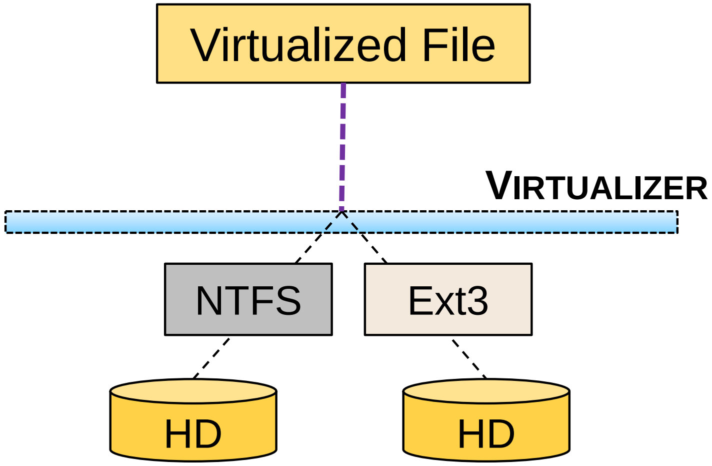
Example /1

Abstraction



One and the same logical abstraction allows for multiple uses by hiding its concrete implementation

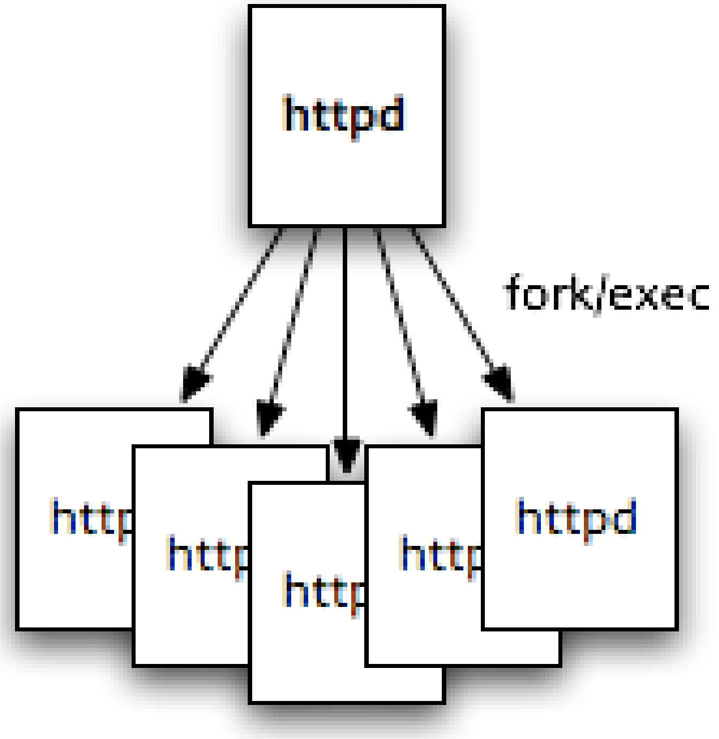
Virtualization



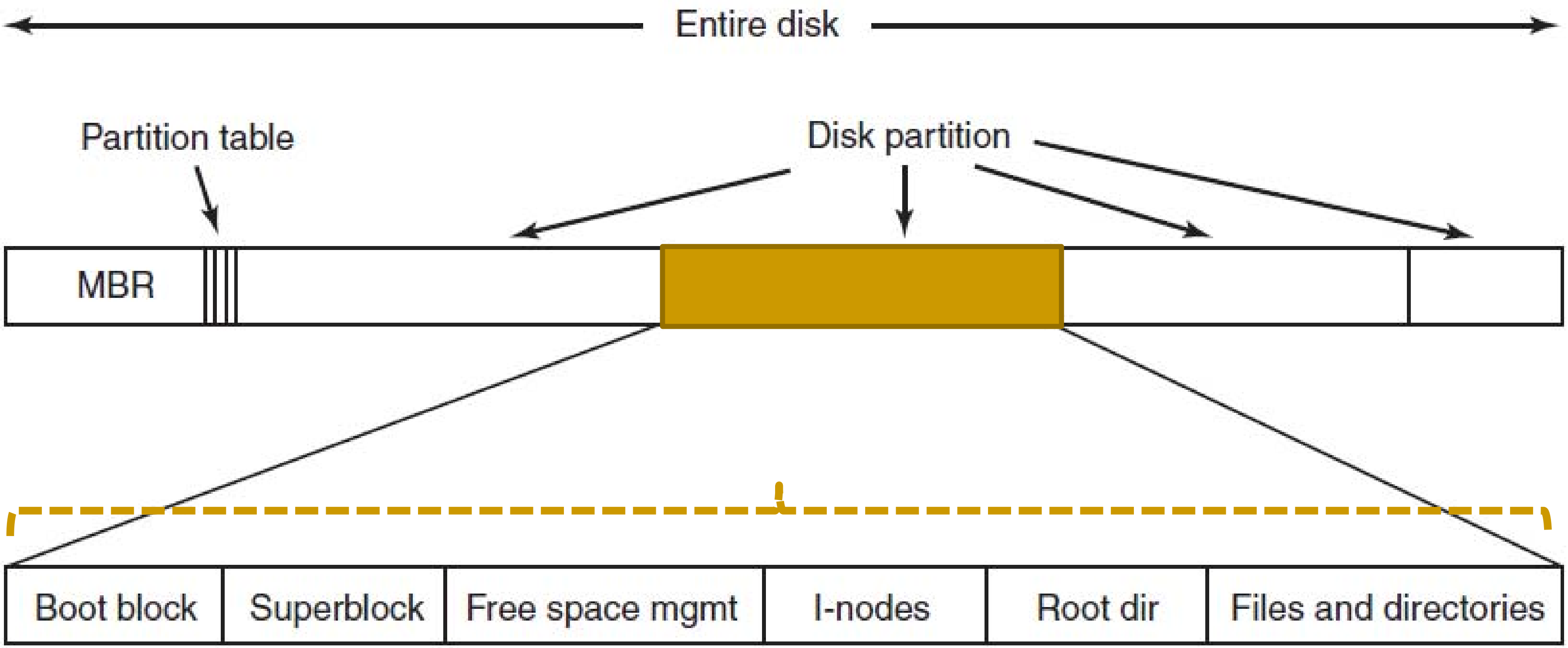
One and the same interface is provided regardless of what the underlying infrastructure has to offer

Example /2

- The UNIX abstraction of “process” lends itself to virtualize into 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*
- The HW resources are scarce and very 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 ’80s, from mainframes to minicomputers and PCs
- The scarcity of hardware resources is alleviated for all users by general-purpose multi-programmed OSs
- Everybody is satisfied and the urge to push virtualization further fades away

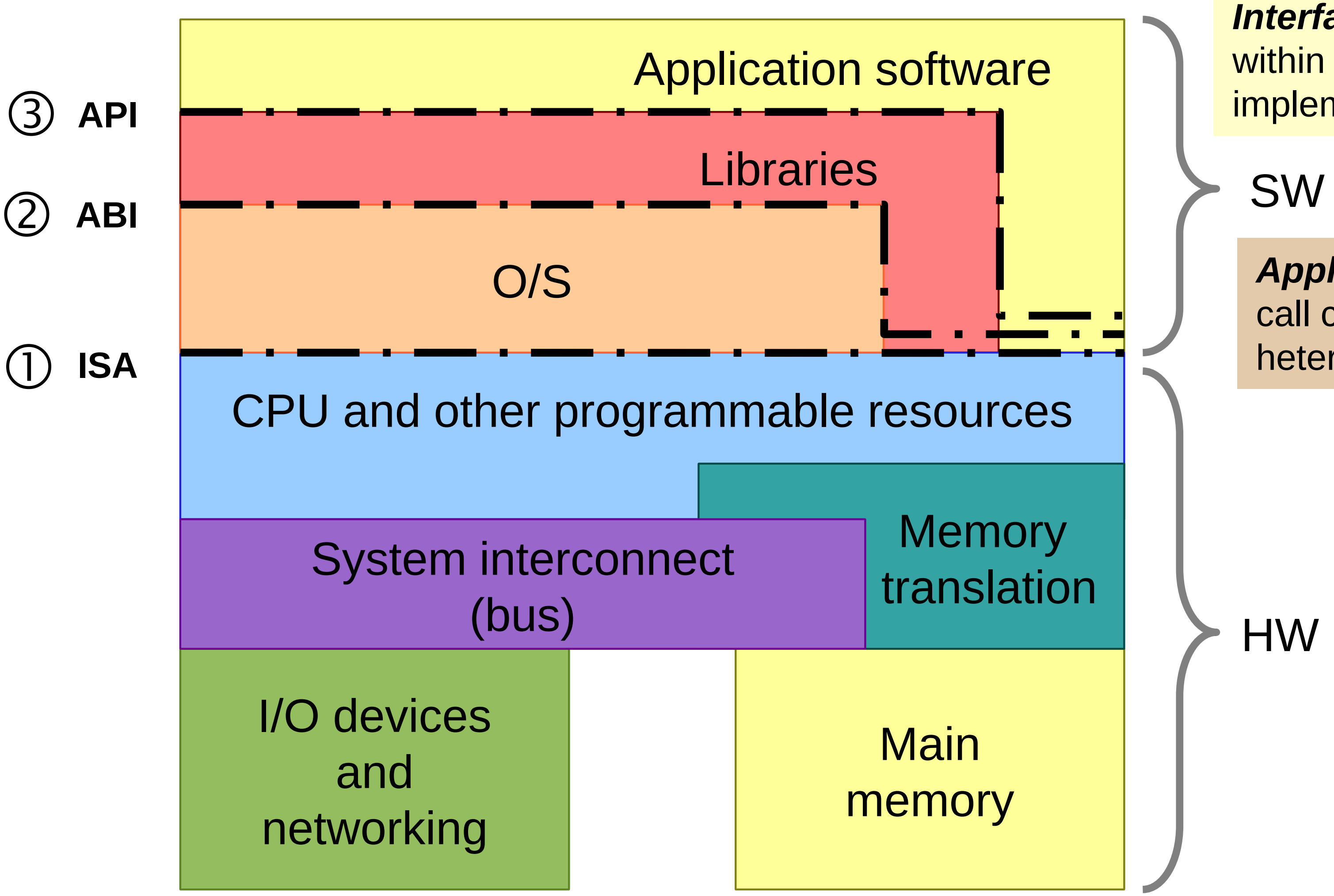
Some history /3

- The early ’90s, commercial and scientific interest for massively parallel computing
  - **Example**: weather prediction ☺
- This needs specialized hardware, 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, industry begins to run on an array of digital services
  - The simultaneous decrease in the unitary cost of computer hardware yields a surge in heterogeneity (the classic law of demand)
- The increasing (vertical) industry needs 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 support cost-reducing “consolidation” (aka rationalization)
  - Sharing HW across application servers

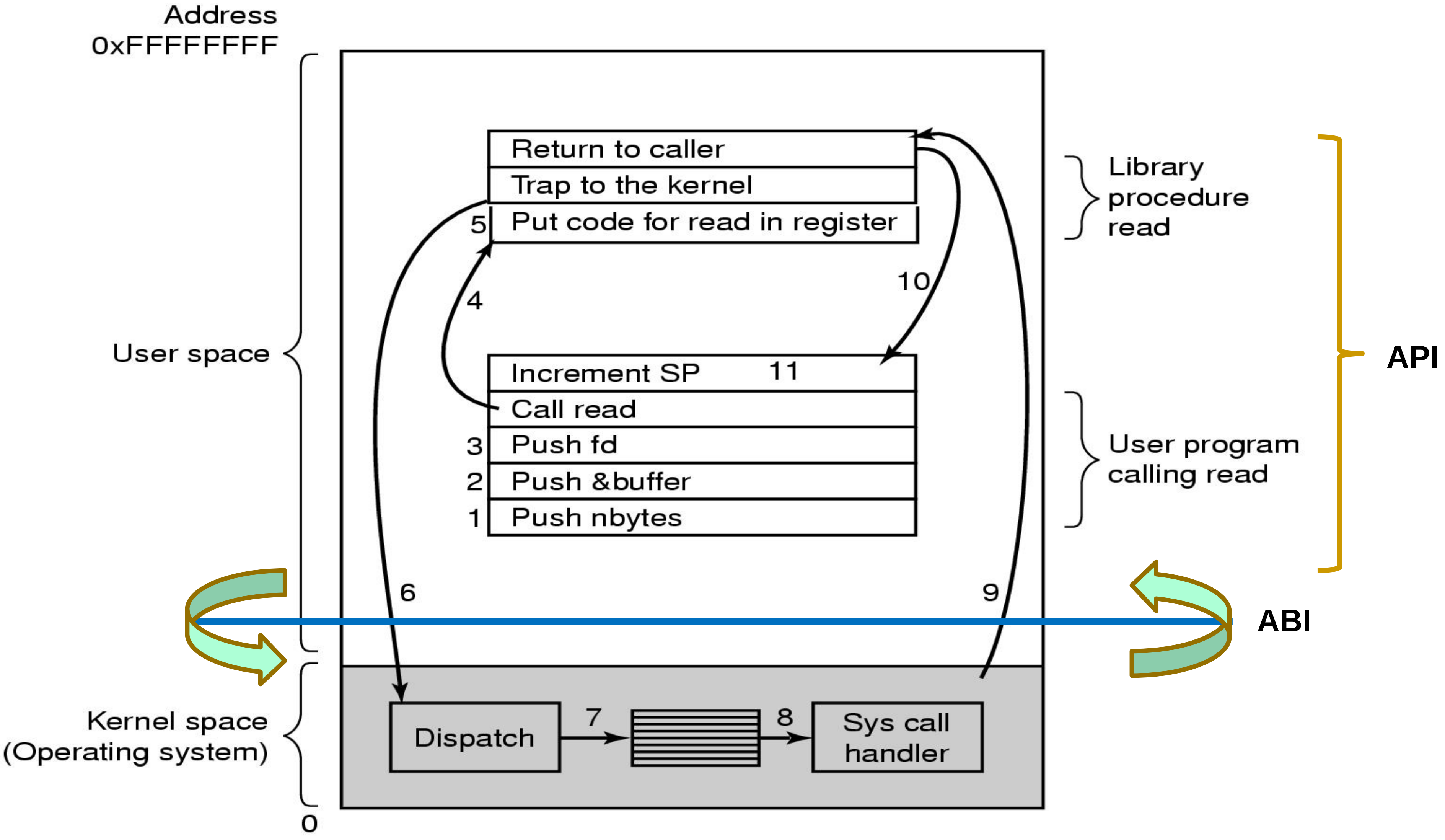
Architecture and interfaces /1


③ API
② ABI
① ISA
Application software
Libraries
O/S
CPU and other programmable resources
Memory translation
System interconnect (bus)
I/O devices and networking
Main memory
SW
HW

Application Programming Interface: call conventions within one and the same implementation language

Application Binary Interface: call conventions across heterogenous executables

Where does the ABI operate?

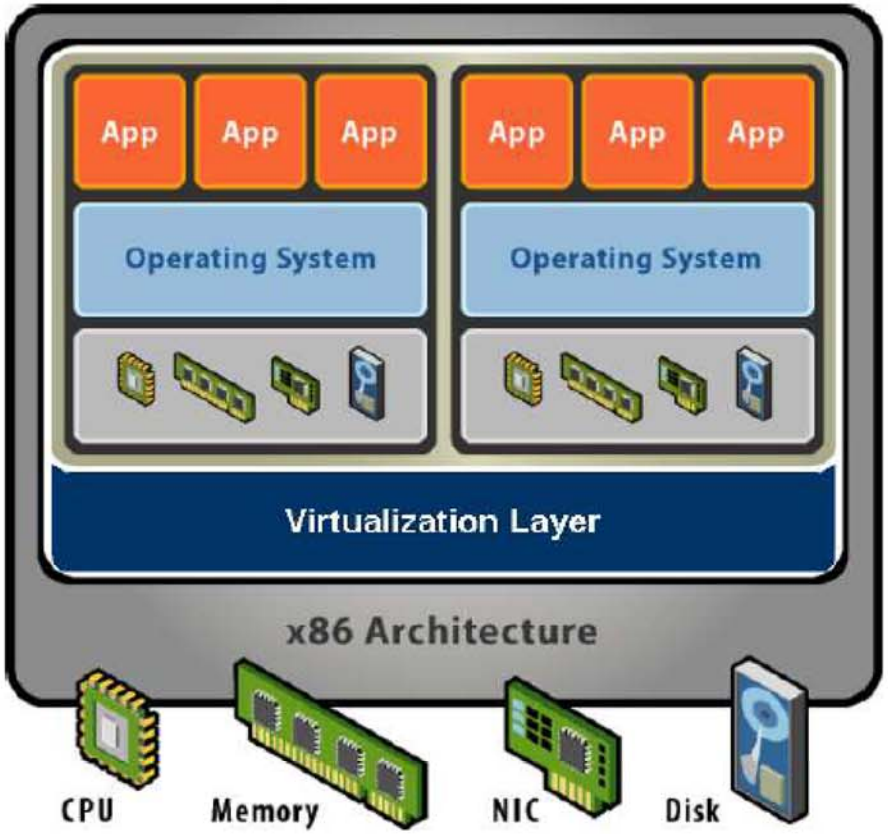


Architecture and interfaces /2

- Changes in the processor hardware may change the ISA
- Changes in the ISA affect the operating system
  - 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

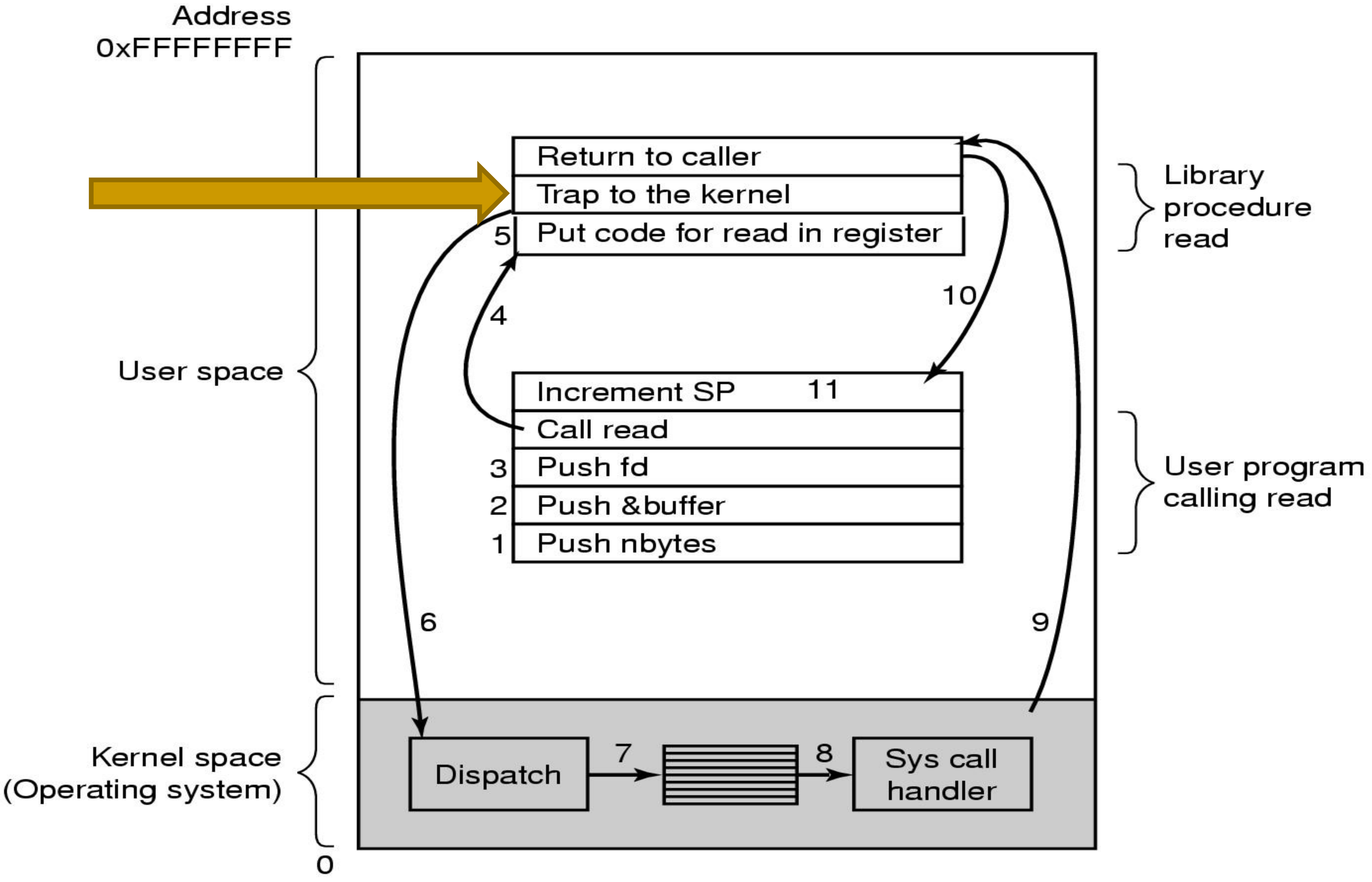


- Hardware-Level Process Abstraction: CPU, memory, chipset, I/O devices, etc.
  - Virtual NIC instead of sockets
  - Virtual disk instead of file system
  - Hardware state becomes software state
- Virtualization Software
  - Hardware and software decoupled

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 outer the ring the greater the ring privilege
- Any attempt to execute outside of the assigned level of privilege is trapped by the processor hardware
  - Hardware trap, a form of predefined exception
- The raising of the program’s level of privilege may be requested by specialized instructions
  - Software trap (and the associated “return from trap”)

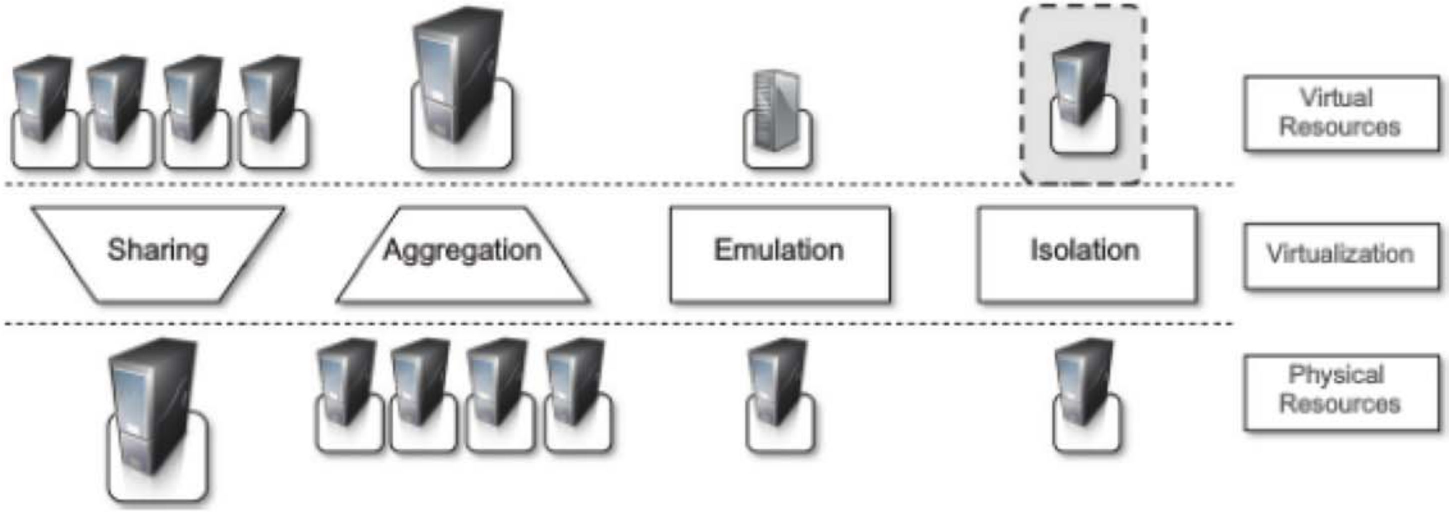
The basics of virtualization /2



A taxonomy of virtualization /1

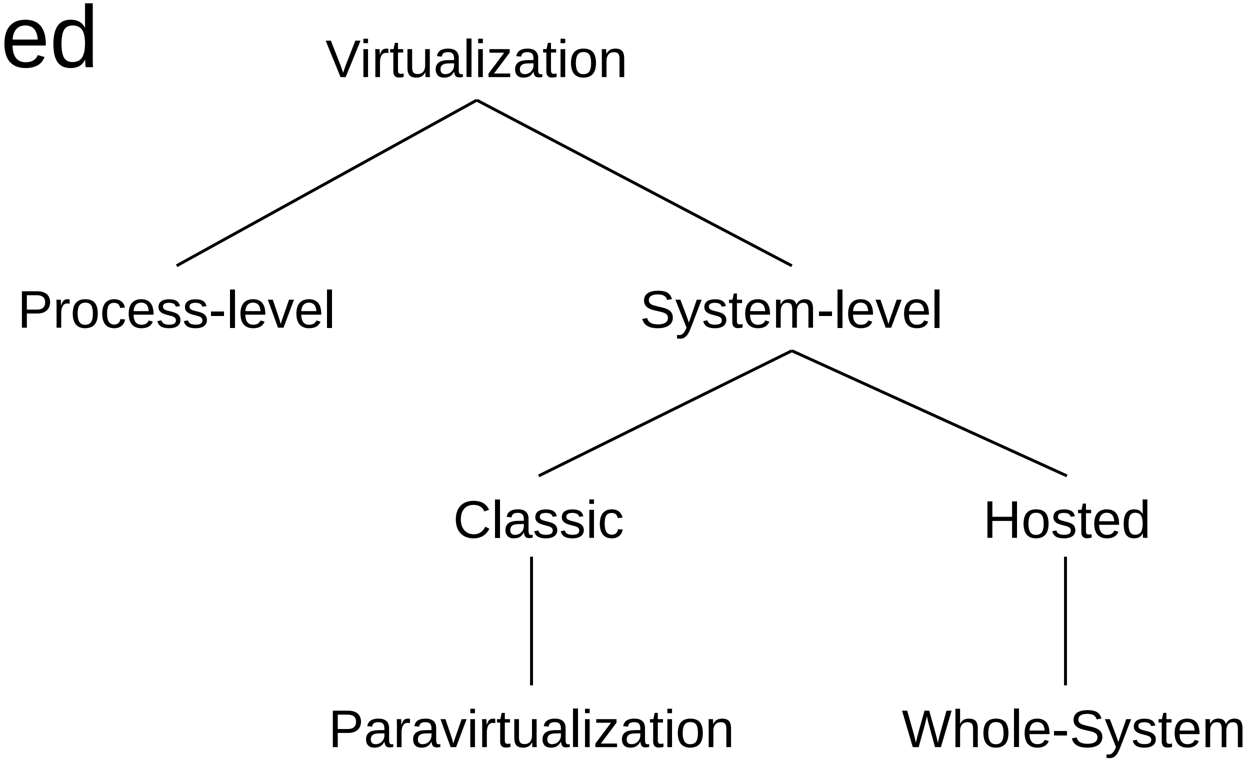
Virtualization allows the creation of a secure, customizable, and isolated execution environment for running applications without affecting other users' applications

- 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



Virtualization

Process-level

System-level

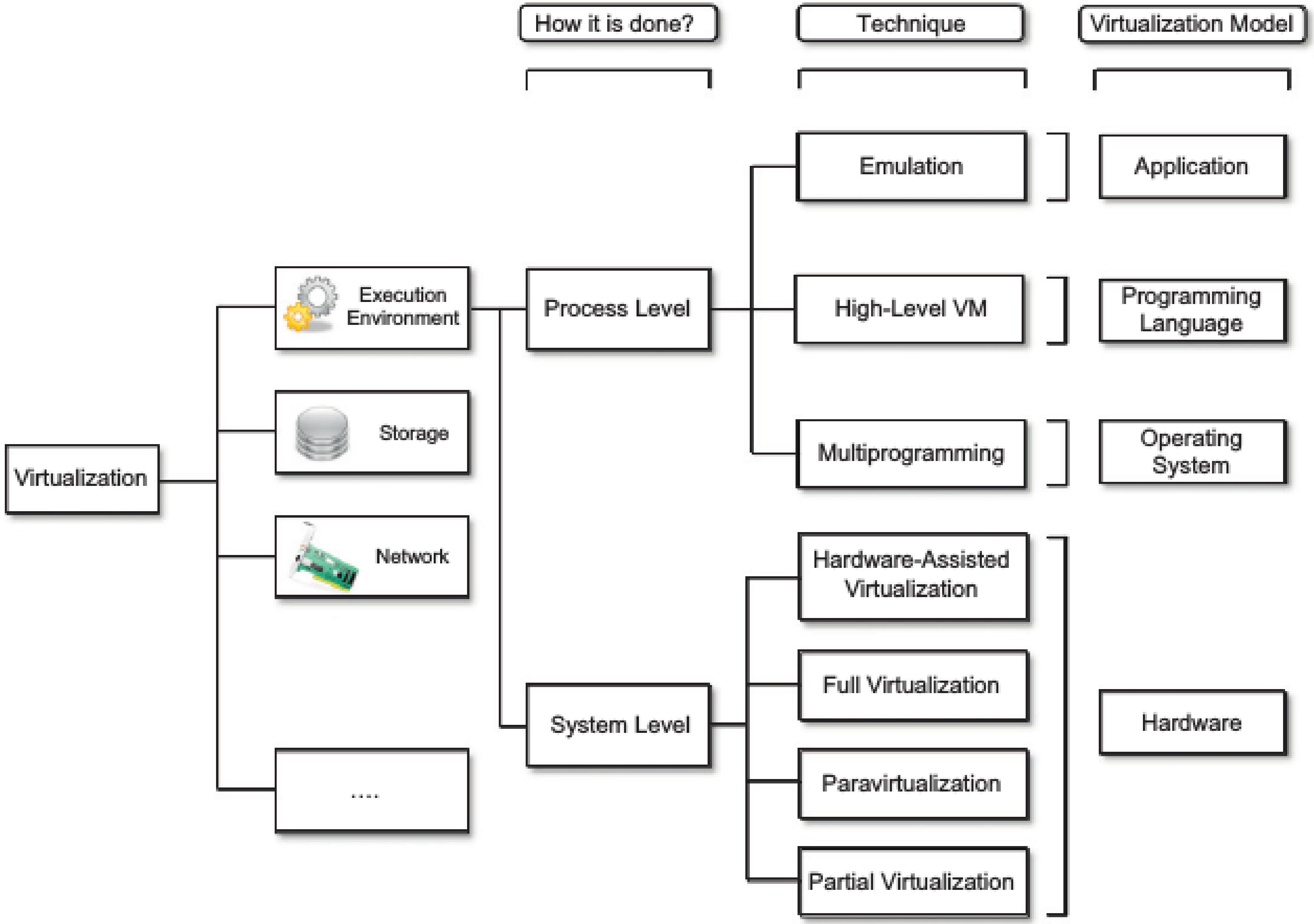
Classic

Hosted

Paravirtualization

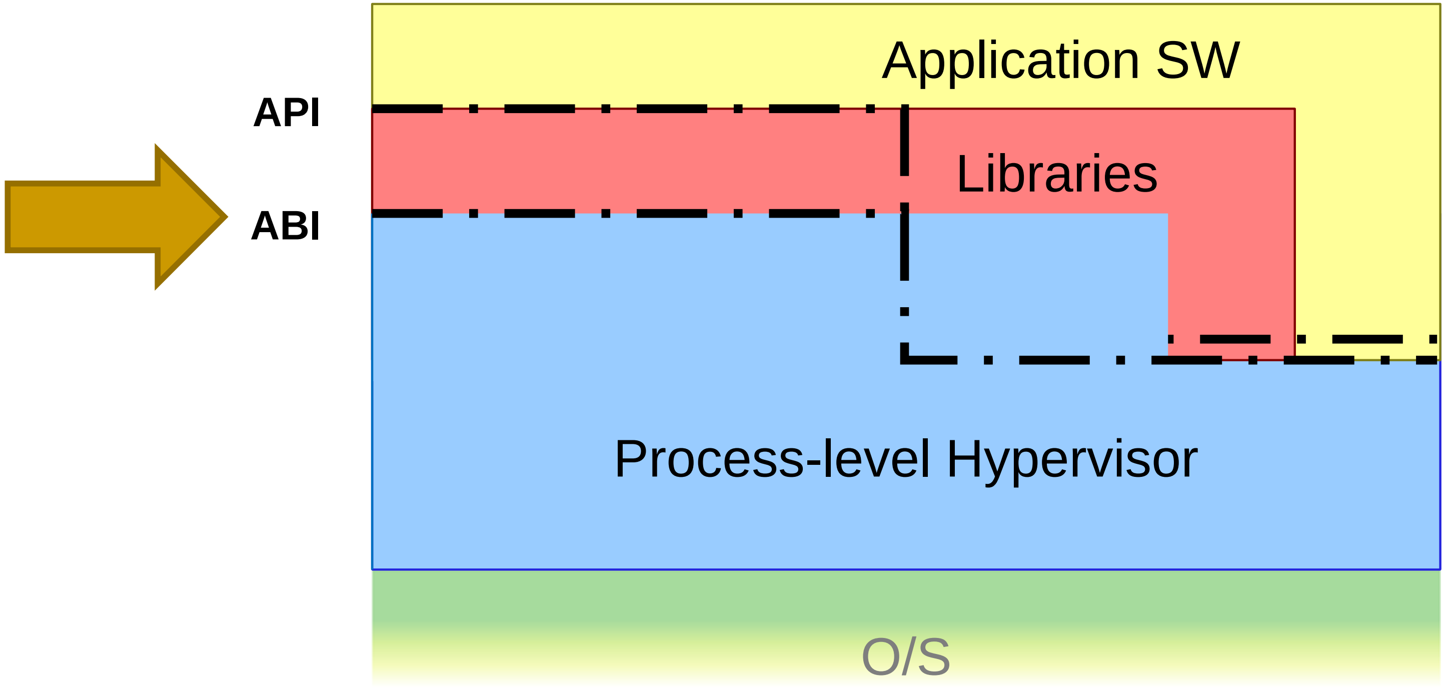
Whole-System

A taxonomy of virtualization /3



Process-level Virtualization /1

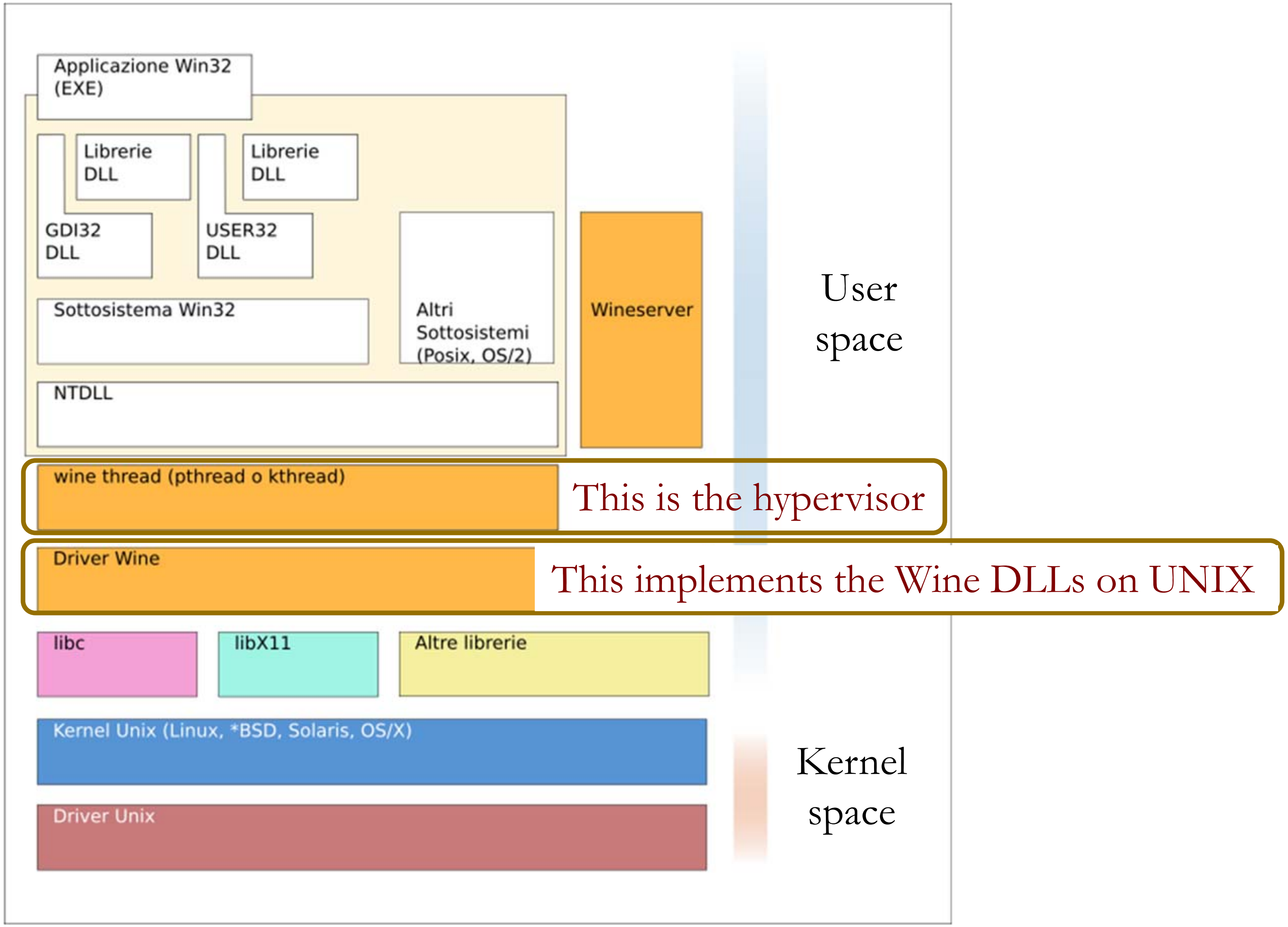
- The hypervisor runs as a process on the host OS, and provides its own ABI for virtualized applications to use



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
- Program execution may be
  - *Direct* if its binary is ISA-conformant, as with Wine
  - *Interpreted*, as over the Java Virtual Machine

Process-level Virtualization: Wine

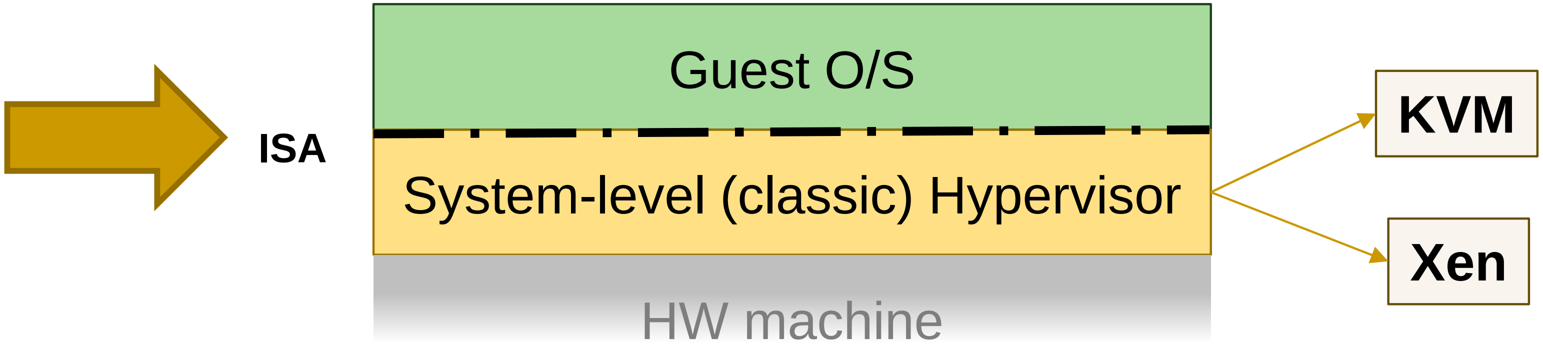


This is the hypervisor

This implements the Wine DLLs on UNIX

System-level (classic) Virtualization

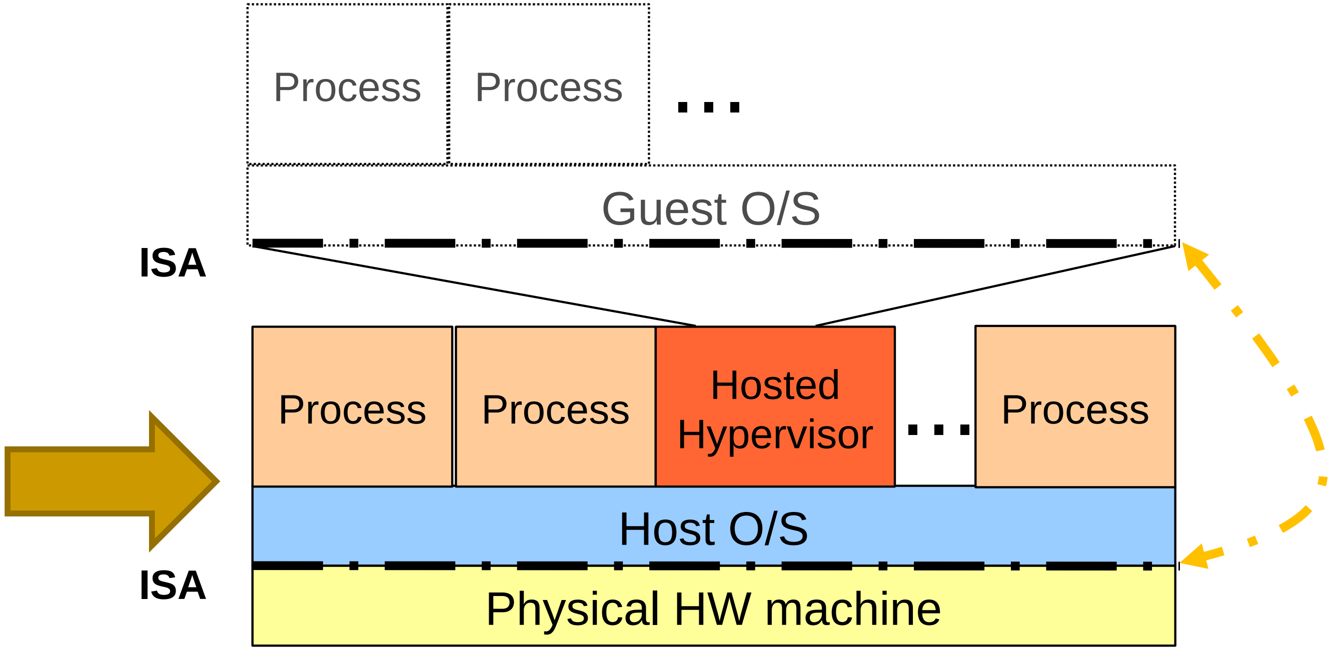
- The hypervisor provides guests with a software ISA
- The guest is a full OS, which however is rendered unable to take control of the processor resources
  - Effectively, the guest OS is stripped of its privileges (**de-privileged**)



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

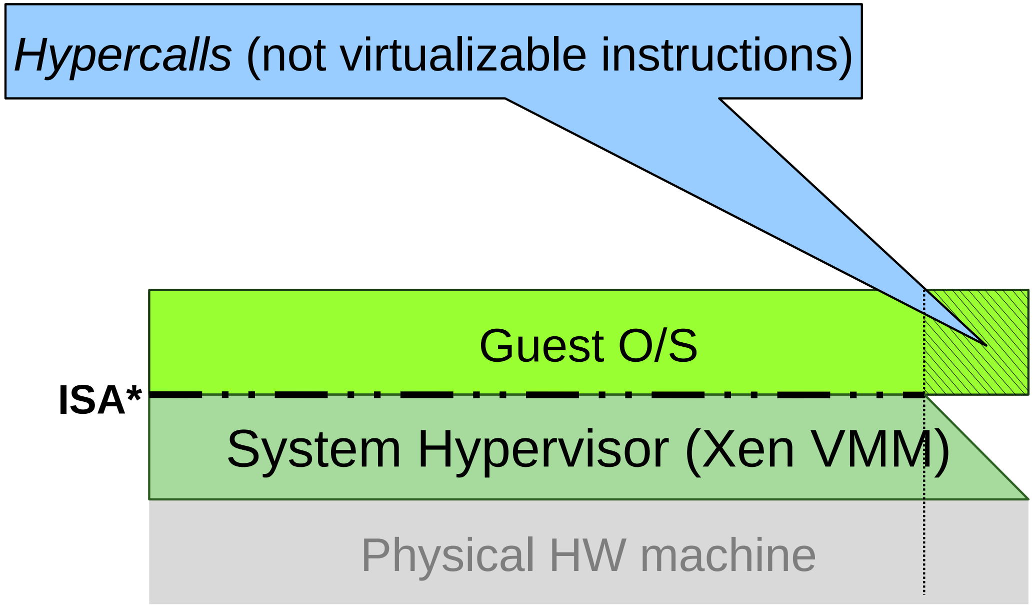


Para-virtualization /1

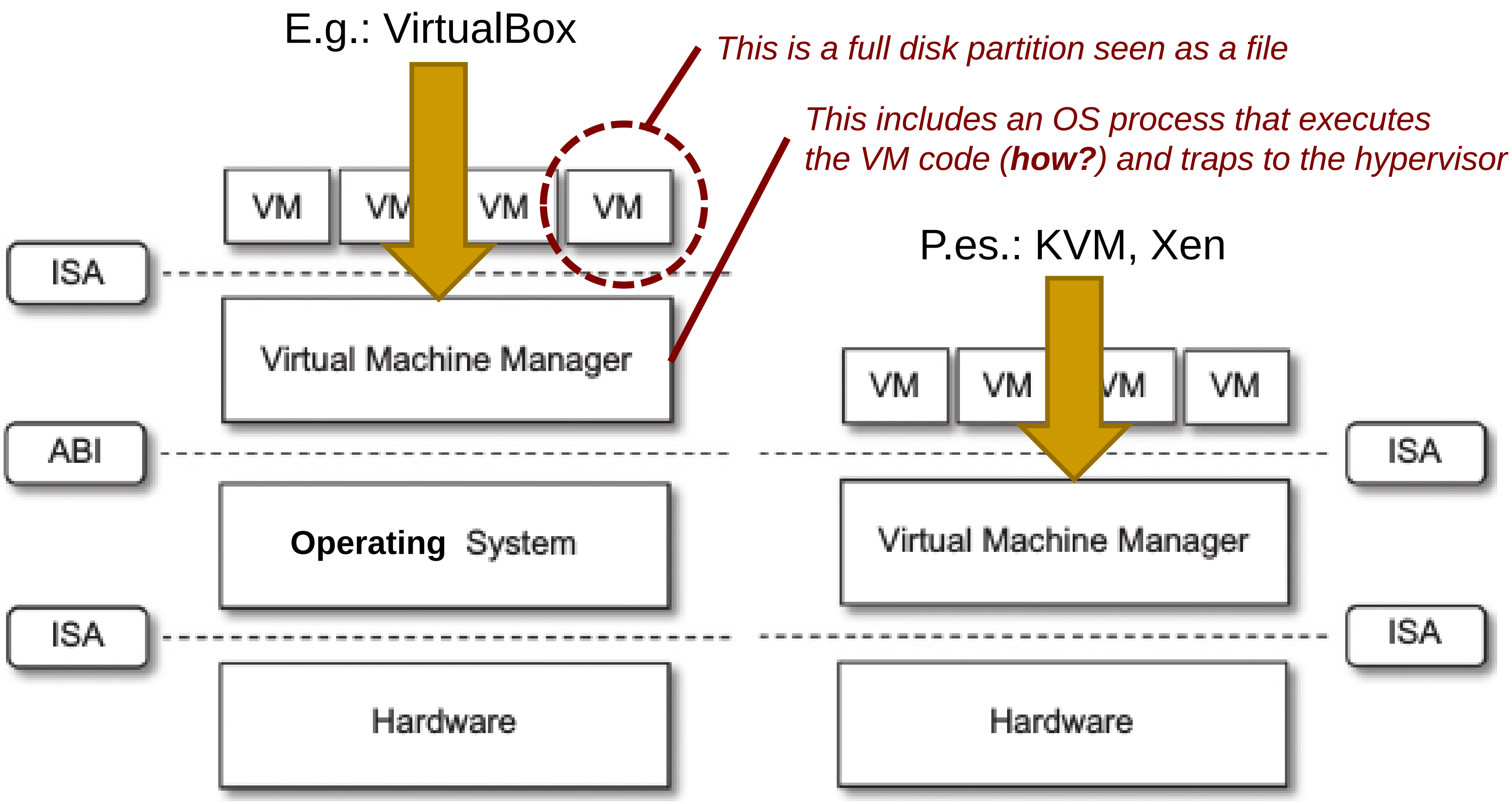
- System-level virtualization rests on the ability to trap trespasses of privilege rings
  - This allows the hypervisor to preserve full control of the processor resources against attempts by the guest OS
- Hardware traps drain performance, which processor makers dislike
  - The support for system-level 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


Hypercalls (not virtualizable instructions)
Guest O/S
ISA*
System Hypervisor (Xen VMM)
Physical HW machine

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*

P.es.: KVM, Xen