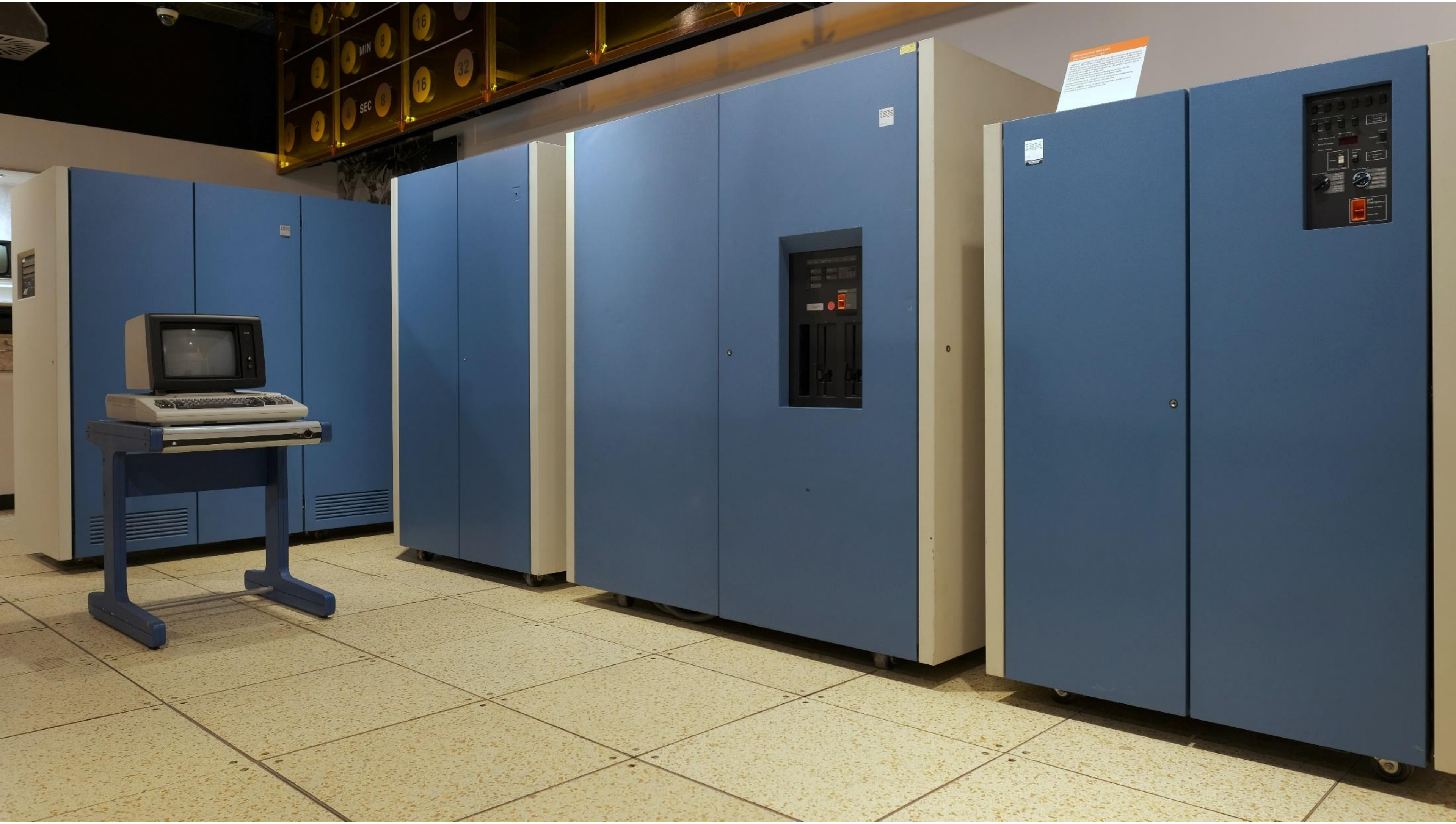
Virtual Machines - history

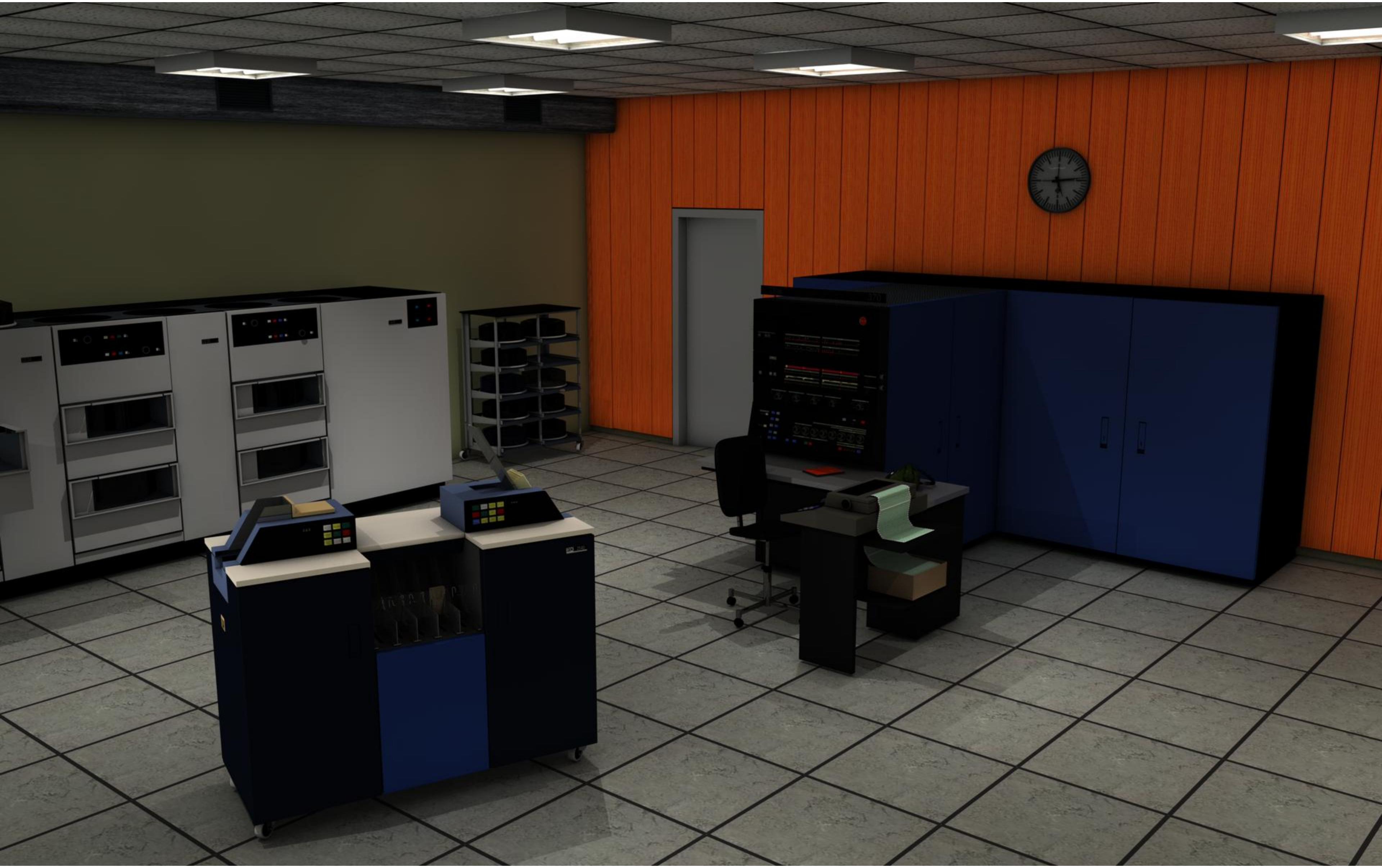
IBM S/370 – 4381 (1983-1992)



4381 Model Group 24 (1988): 20 MHz dual CPU, 128KB cache, 64MB RAM, 890K USD (2M USD@2020)

(Technické muzeum v Brně, 2018)

IBM S/370 - 145 (1971, up to 512 KB RAM, ~1M USD)

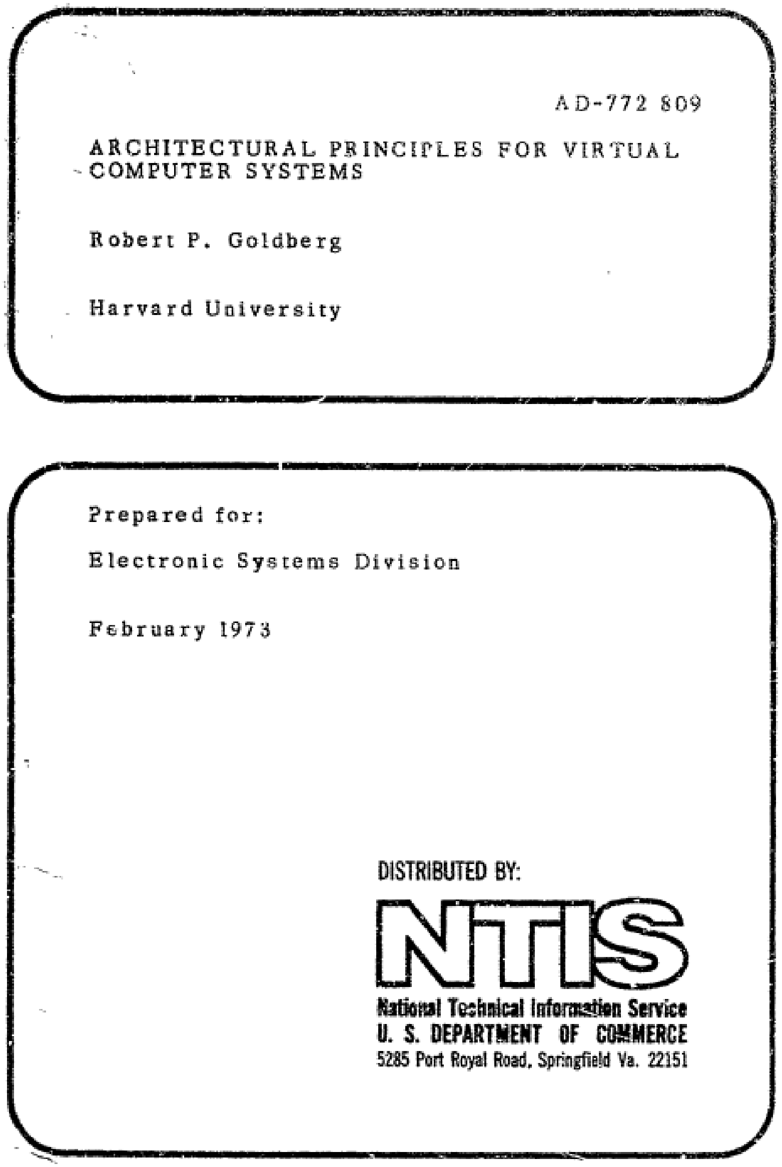


(3D Rendering, Oliver.obi, 2013)

IBM S/370 - 145 (1971)

(3D Rendering, Oliver.obi, 2013)

Virtual Machines - history



AD-772 809

ARCHITECTURAL PRINCIPLES FOR VIRTUAL COMPUTER SYSTEMS

Robert P. Goldberg

Harvard University

Prepared for:

Electronic Systems Division

February 1973

DISTRIBUTED BY:

NTIS

National Technical Information Service
U. S. DEPARTMENT OF COMMERCE
5285 Port Royal Road, Springfield Va. 22151

- The first era of VM
  - 1972 – IBM VM for S/370
    - Coexistence of different OSes
    - Time-sharing and virtual memory for OSes not implementing them
    - Debugging of OSes
      - Including a VM under a VM
  - Every tenth S/370 used a VM
  - 1980... – Gradual decline
    - Mainframes overcome by cheaper architectures (minicomputers, PC)
      - New hardware did not support VM
    - The growing dominance of Unix
      - VM is a complication for inter-process communication

VM - requirements

- Formal Requirements
for Virtualizable Third Generation Architectures
  - Gerald J. Popek and Robert P. Goldberg, 1974

  - Equivalence / Fidelity
    - A program running under the VMM should exhibit a behavior essentially identical to that demonstrated when running on an equivalent machine directly.

  - Resource control / Safety
    - The VMM must be in complete control of the virtualized resources

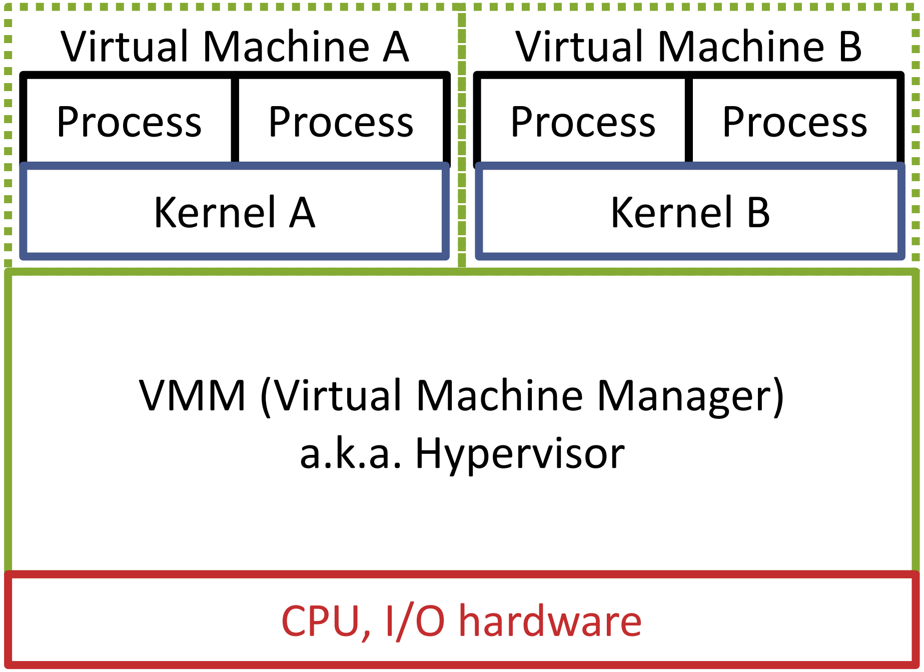
  - Efficiency / Performance
    - A statistically dominant fraction of machine instructions must be executed without VMM intervention

CPU virtualization

Types of Virtual Machine Systems

Type 1 (Bare Metal) Hypervisor

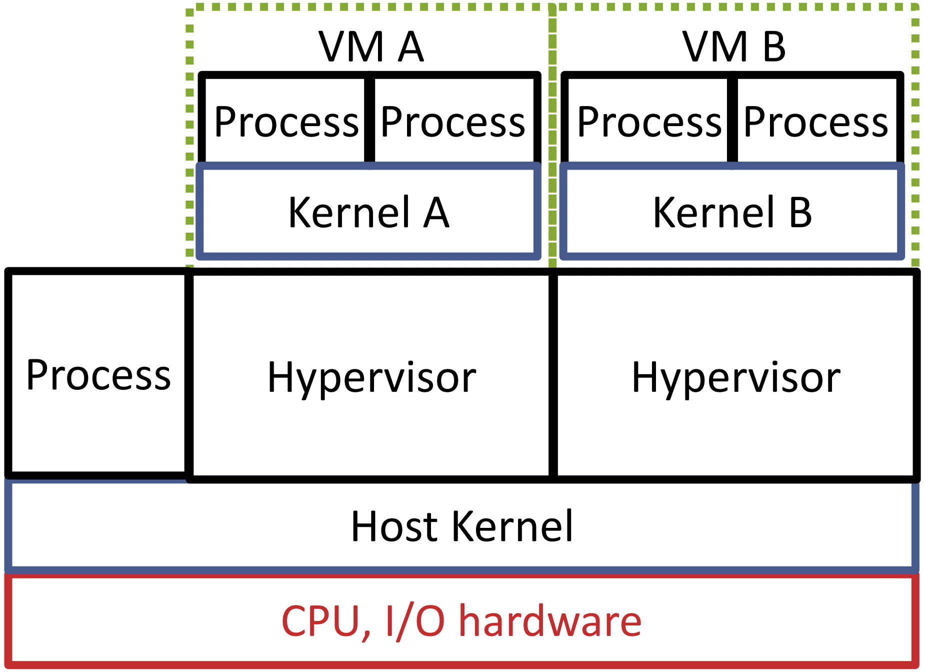
Example: VMWare ESXi



- Hypervisor on bare metal
  - Hypervisor directly performs all hardware access (CPU configuration, I/O)
    - Requires device drivers
    - Complex but fast

Type 2 (Hosted) Hypervisor

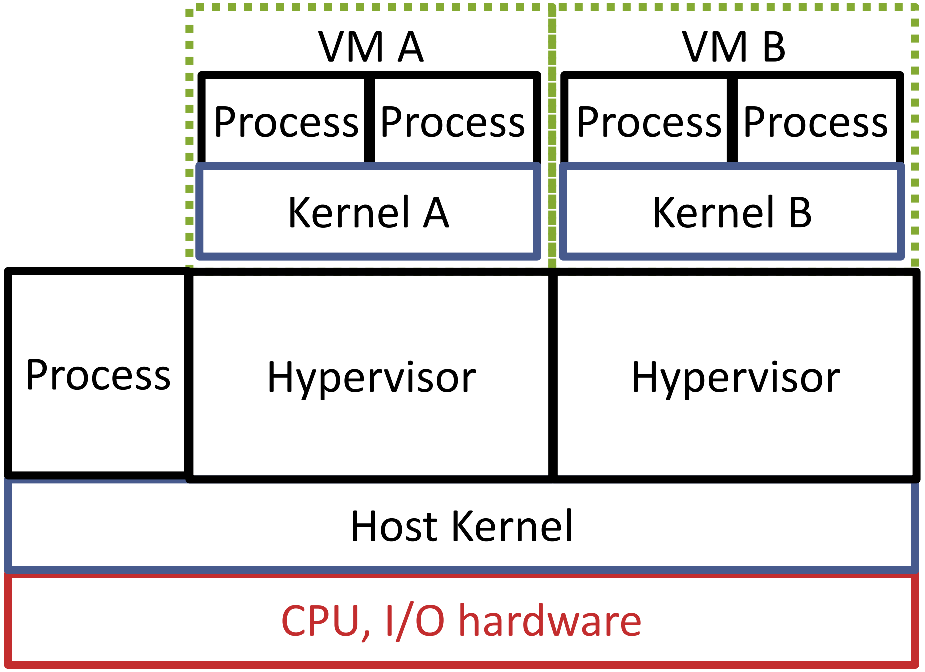
Example: VMWare Workstation Player



- Hypervisor above an host OS
  - Hypervisor is a (privileged) process
    - Often one per VM
    - I/O access performed by host kernel
    - CPU control requires support from the host kernel (debugging services)

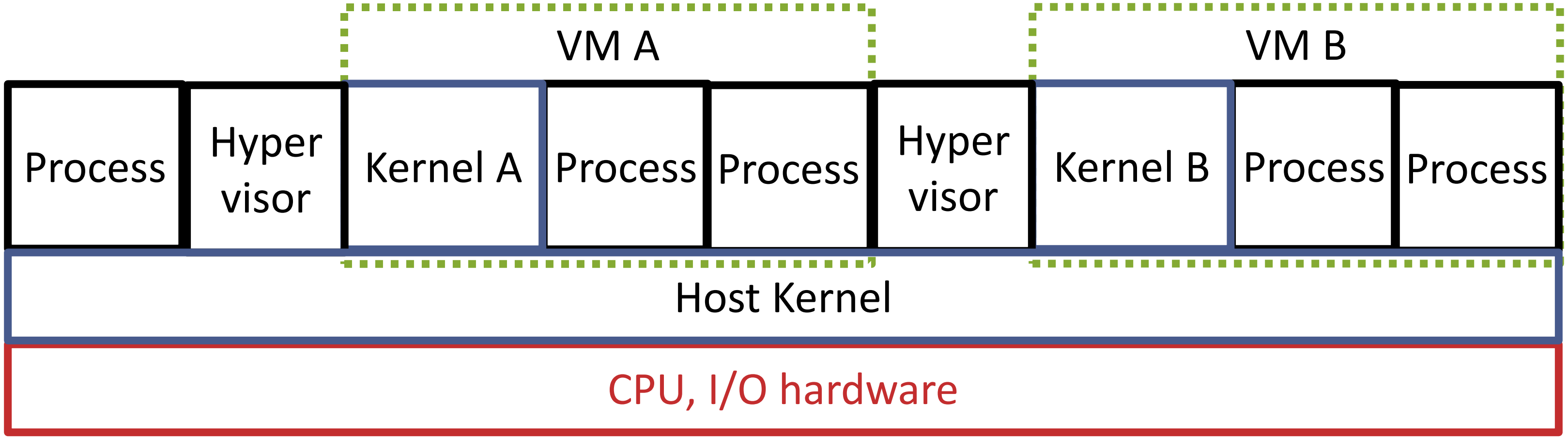
Beware

- Pictures like this are misleading

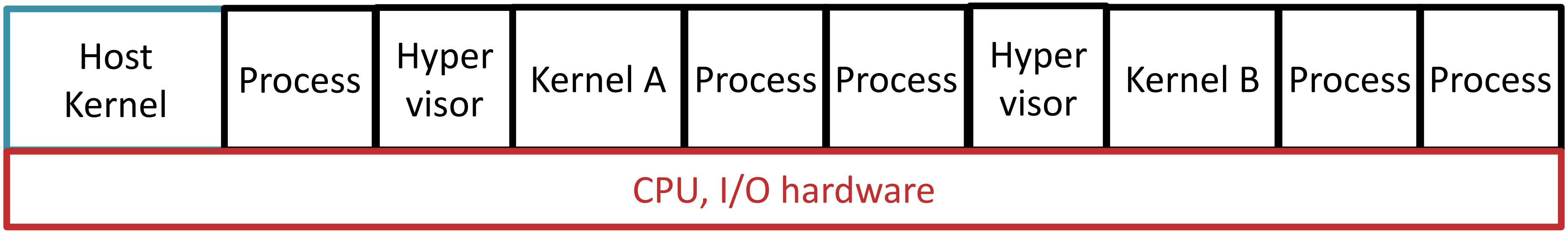


Beware

- The host kernel actually sees this:



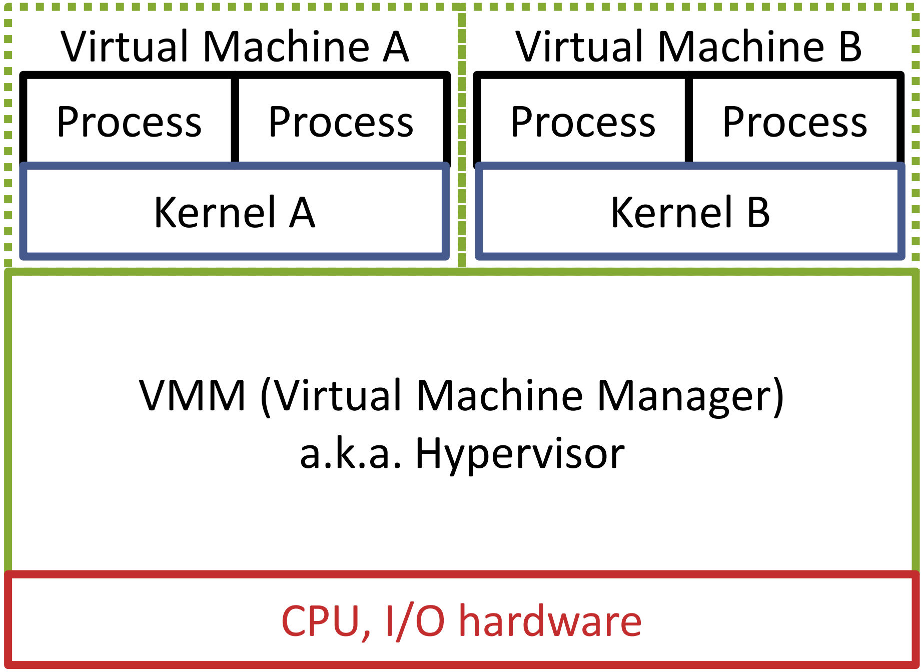
- The CPU sees this:



Flavors of Type 1 Hypervisors

Traditional

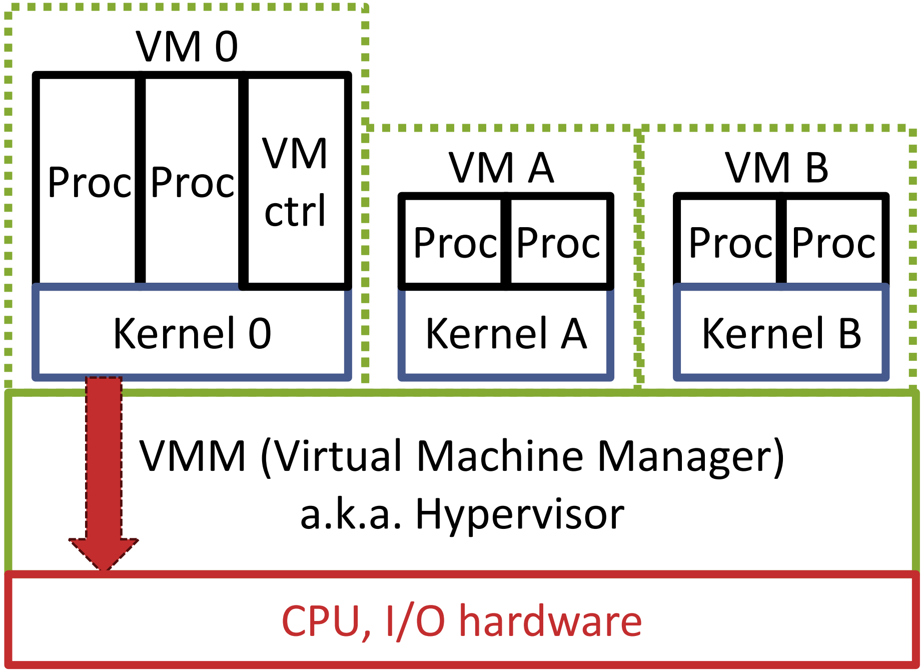
Example: VMWare ESXi



- Hypervisor performs I/O
  - Requires device drivers tailored for the hypervisor
  - Too costly development

With root partition (Microsoft terminology)

Example: Microsoft Windows + Hyper-V

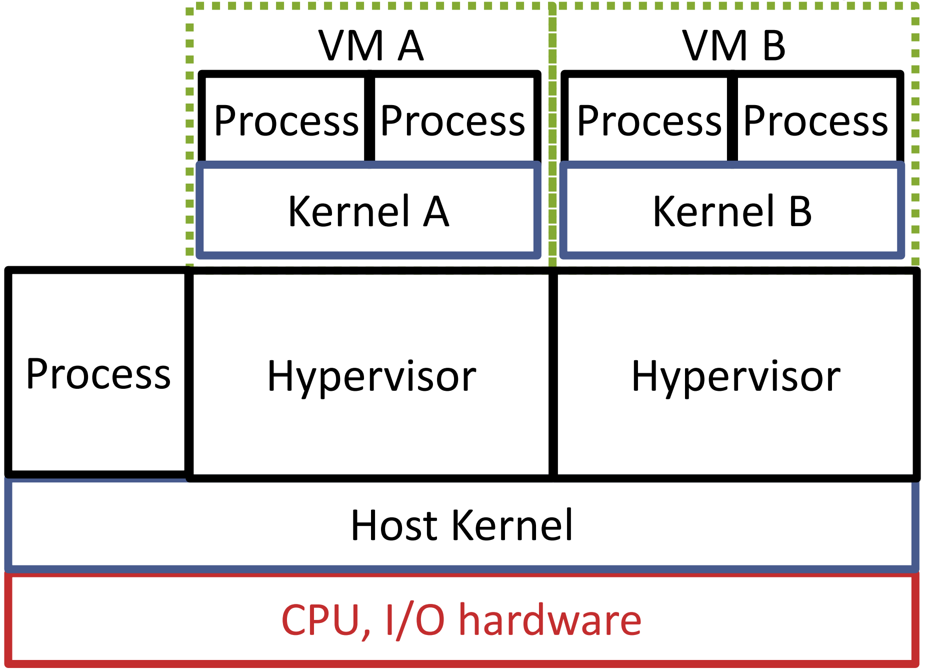


- Hypervisor only controls CPU
  - VM 0 aka Root partition
    - Allowed to directly access I/O hardware
    - Standard OS with device drivers
  - Hypervisor forwards I/O requests

Flavors of Type 2 Hypervisors

Implemented in user-space

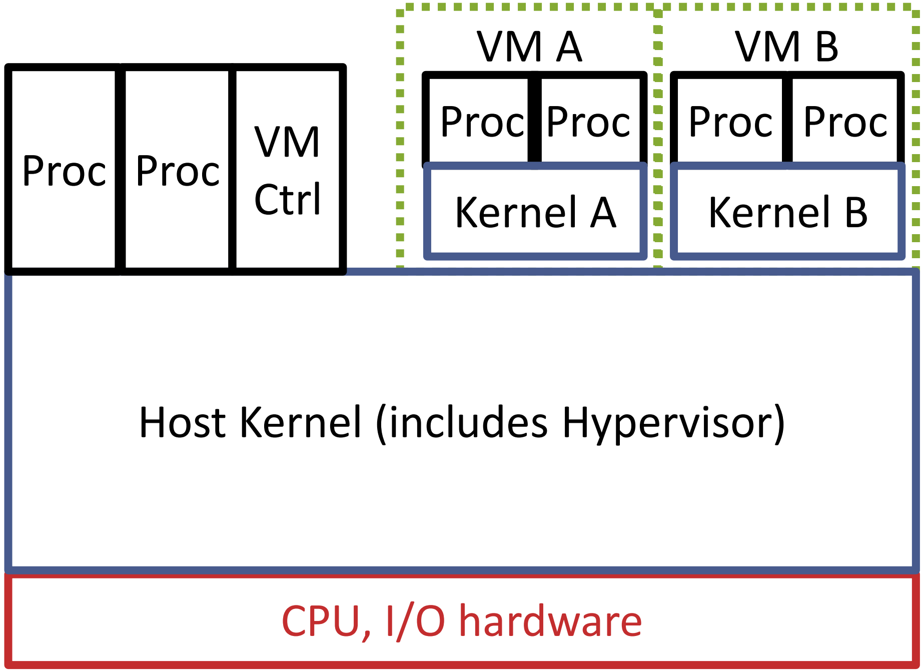
Example: VMWare Workstation Player



- Hypervisor above an host OS
  - Hypervisor is a (privileged) process
    - Often one per VM
    - I/O access performed by host kernel
    - CPU control requires support from the host kernel (debugging services)

Implemented in a kernel

Example: Linux KVM

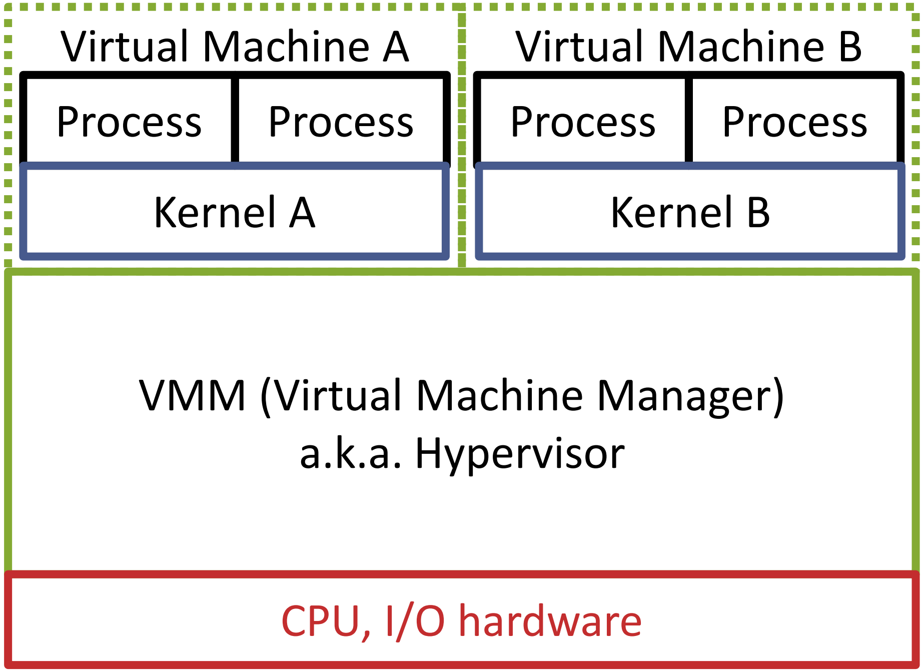


- Hypervisor integrated in kernel
  - Fast
    - No need to indirect CPU control via kernel service
  - Complex and dangerous
    - Kernels were not designed for this

Where is the difference? Only in the history.

Traditional type 1 hypervisor

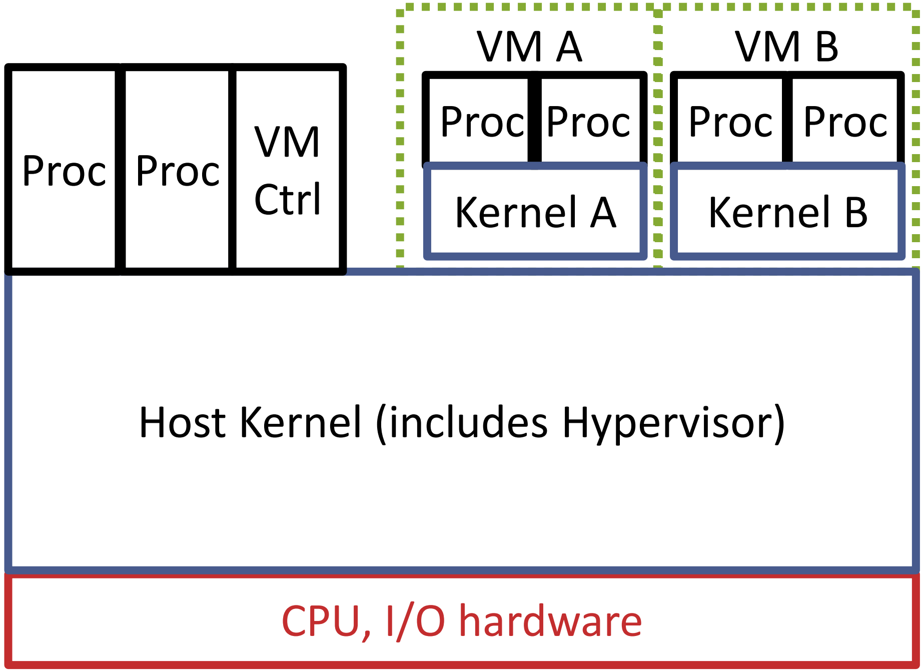
Example: VMWare ESXi



- Hypervisor does everything
  - CPU control, time sharing, and I/O in the same project
  - Complex and dangerous

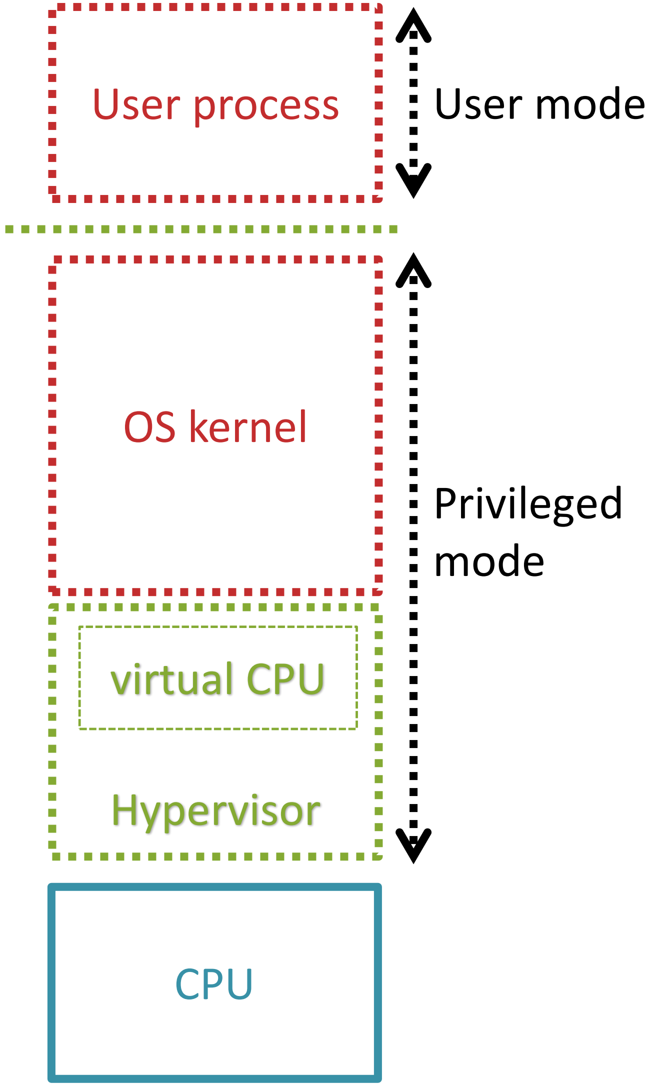
Type 2 implemented in a kernel

Example: Linux KVM



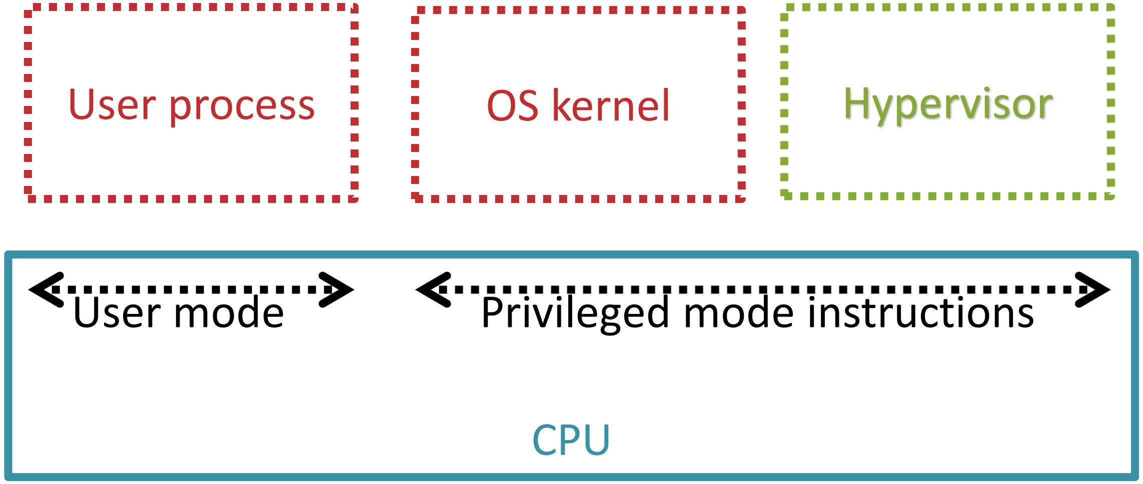
- Hypervisor implanted in kernel
  - CPU control, time sharing, and I/O in the same project
  - Complex and dangerous

True virtualization



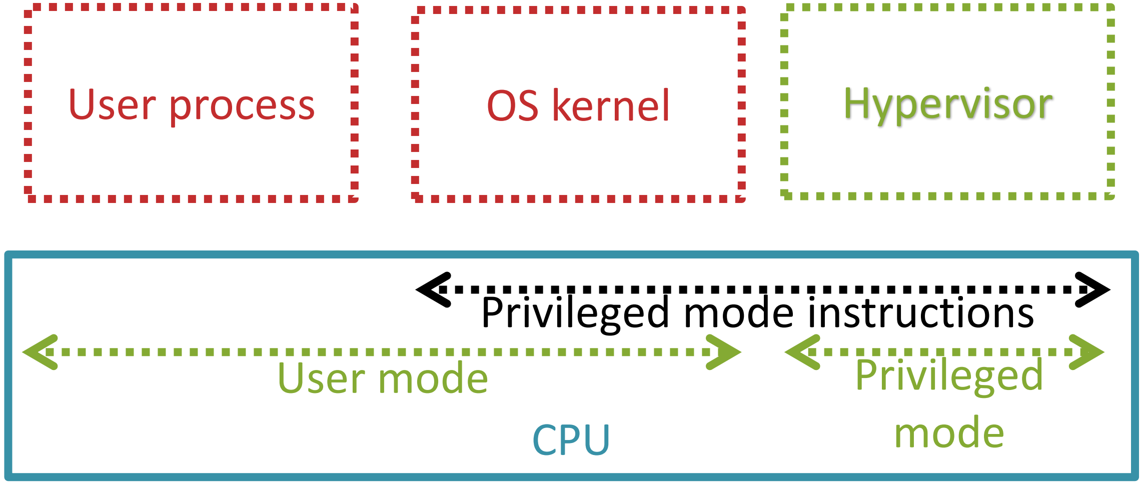
- True virtualization
- Three layers of software
  - Guest user processes
    - Only user-mode CPU instructions
  - Guest OS kernel
    - All CPU instructions
    - Privileged mode expected
      - But shall not be granted
  - Hypervisor
    - All CPU instructions
    - Exclusive control over the hardware
- This picture is misleading

True virtualization



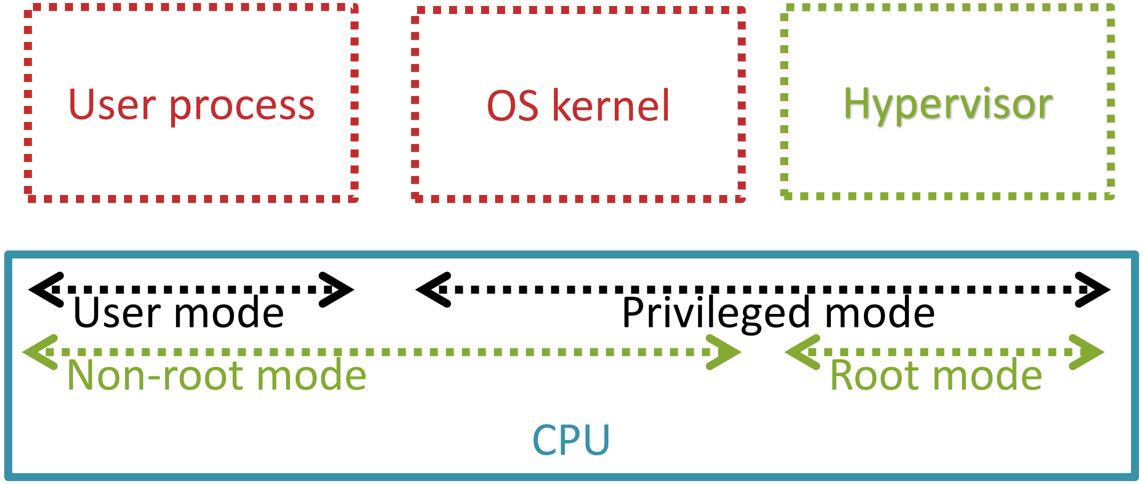
- The correct picture
  - All levels of the software directly interact with CPU by executing instructions
  - OS kernels use privileged instructions
    - We must not allow their direct execution (Popek-Goldberg: Safety)
    - We must allow direct execution of the other instructions (Performance)
    - OS kernel must run with a different privilege setting than the Hypervisor
  - *Compression of privileges*
    - Mapping of 3 privilege levels onto the 2 levels available in typical CPU

True virtualization



- Trap-and-emulate (IBM 1972)
  - OS kernels use privileged instructions but run in the user mode
    - Every privileged instruction in the kernel causes a *trap* (synchronous interrupt)
    - The hypervisor emulates the instruction
    - The emulation allows verification of access rights, virtualization etc.
  - Performance considerations
    - Every syscall goes through hypervisor
    - Every I/O instruction in kernel is emulated
      - S/370 had "channel programs" = single I/O instruction started the whole I/O operation

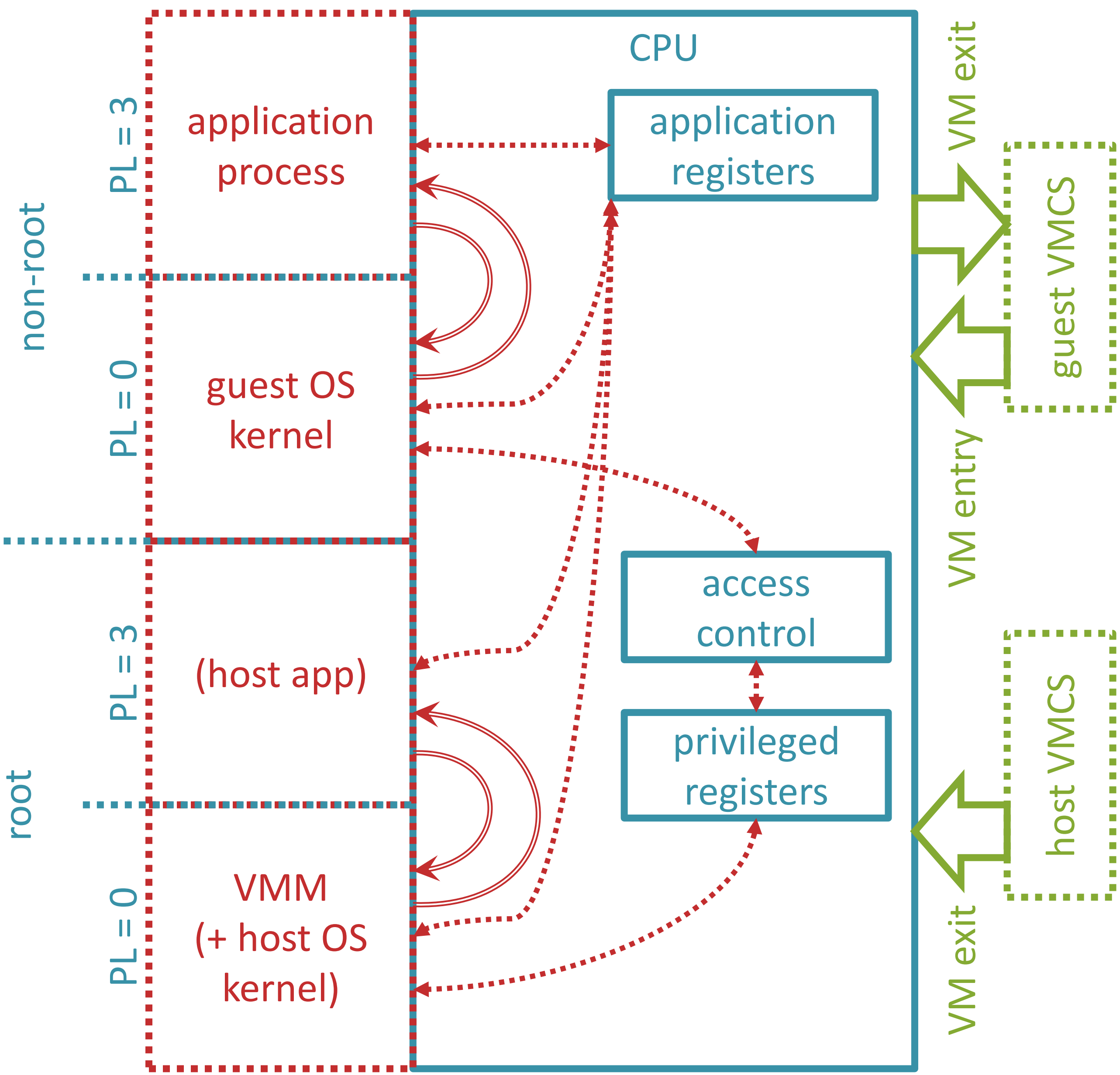
True virtualization



Intel/AMD root/non-root modes

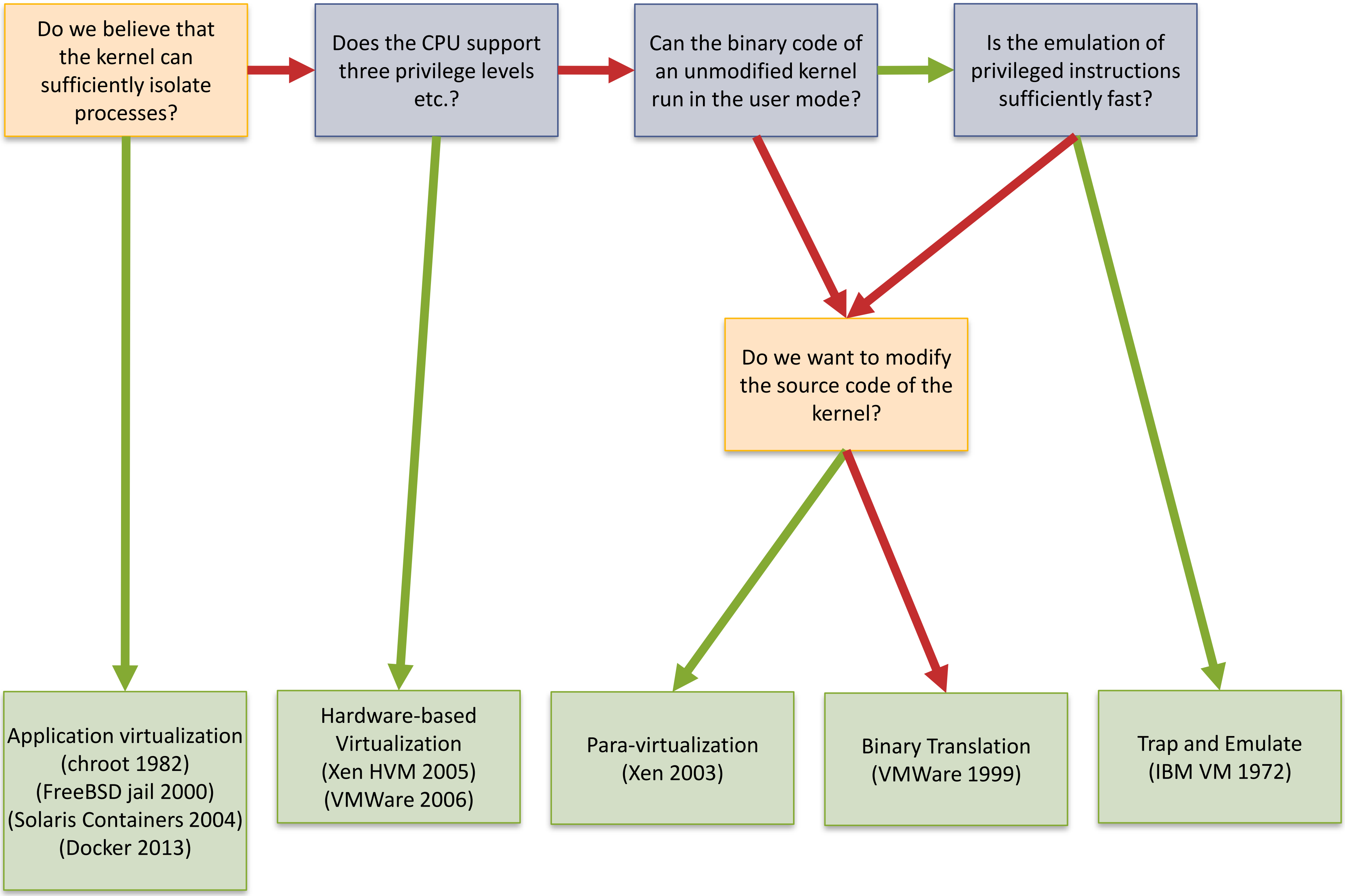
- Control transferred between levels
  - User -> Kernel: Simultaneously with switching CPU to the privileged mode
    - SYSCALL, some synchronous (software) interrupts
  - User/Kernel -> Hypervisor: VM Exit event = switch to hypervisor mode
    - Asynchronous (hardware) interrupts
    - Some synchronous (software) interrupts (e.g. page faults)

Hardware support for virtualization – a new dimension of privilege



- Intel VT-x / AMD-V
  - Details differ
- „Root“ mode
  - Like a CPU without virtualization
  - Usable to run the host OS
- „Non-root“ mode
  - Limited access to the privileged of the CPU state
  - Unwanted actions cause „VM exit“
- Mode switch
  - A part of the CPU state is read from/stored to memory
  - Address-space switch included

Approach to virtualization



Do we believe that the kernel can sufficiently isolate processes?

Does the CPU support three privilege levels etc.?

Can the binary code of an unmodified kernel run in the user mode?

Is the emulation of privileged instructions sufficiently fast?

Do we want to modify the source code of the kernel?

Application virtualization
(chroot 1982)
(FreeBSD jail 2000)
(Solaris Containers 2004)
(Docker 2013)

Hardware-based Virtualization
(Xen HVM 2005)
(VMWare 2006)

Para-virtualization
(Xen 2003)

Binary Translation
(VMWare 1999)

Trap and Emulate
(IBM VM 1972)

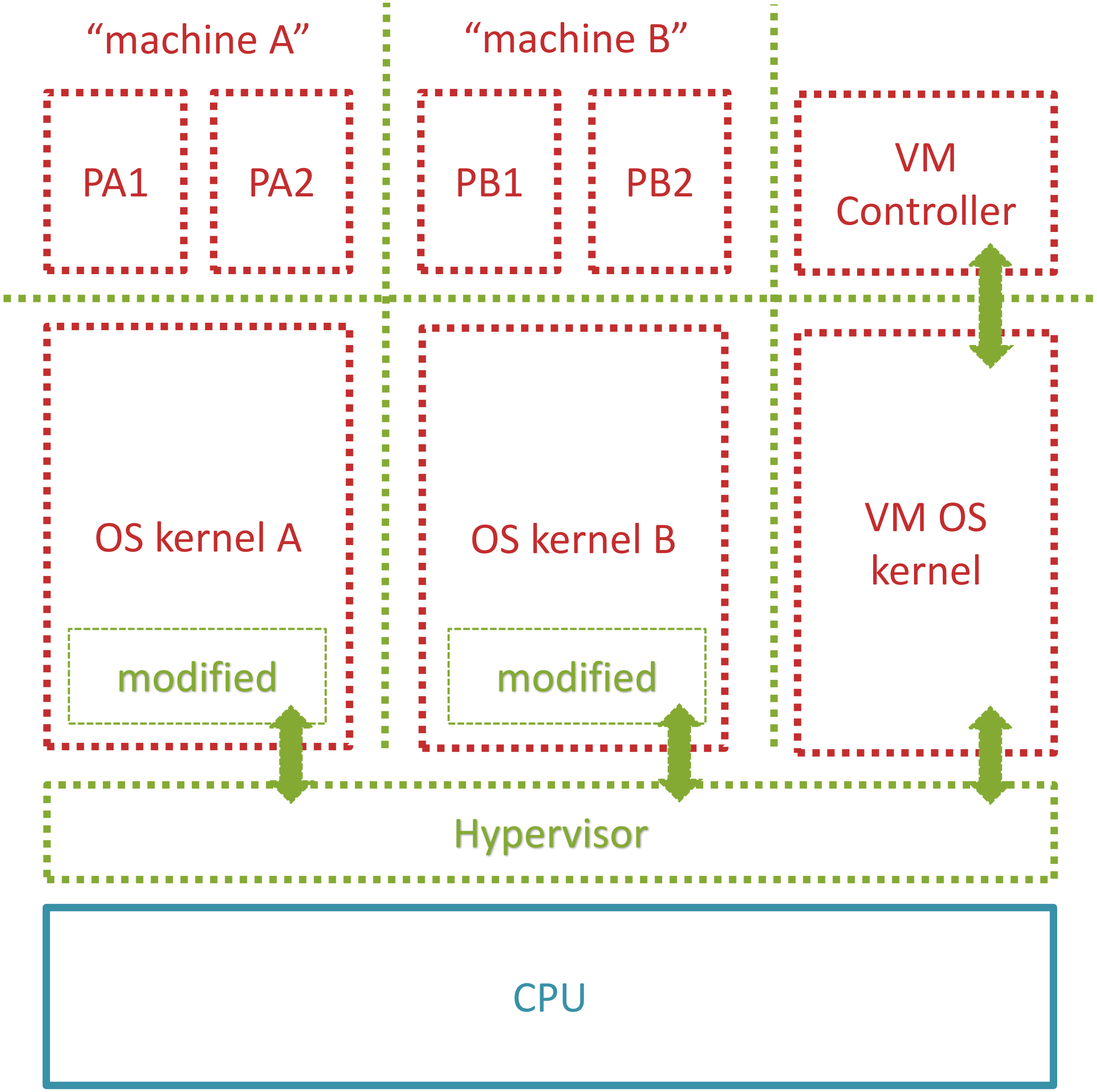
Virtualization - history

- The second era of VM
  - 1999 – VMWare Workstation
    - Software virtualization (Binary Translation)
    - VMM as an app in Windows NT
  - 2002 – VMWare ESX Server
    - VMM replaces the host OS
  - 2003 – Xen
    - Para-virtualization
      - Host OS modified (in source code)
  - 2007 – Linux KVM
    - VMM integrated into the OS kernel
  - 2008 – Microsoft Hyper-V
    - VMM cooperates with the host OS
      - Non-cooperating guest OS possible

- The x86 is unsuitable for VM
  - Legacy of the Intel 80286 CPU
    - 1982 – still in the first era of VM
  - The first mitigation attempts
    - 2005 – Intel VT-x
    - 2006 – AMD-V
  - Gradually improved performance
    - Improved HW support
    - Para-virtualization in critical OS parts
  - Performance loss is now insignificant for most applications
    - Variations in performance too big for Real-Time applications and performance measurement

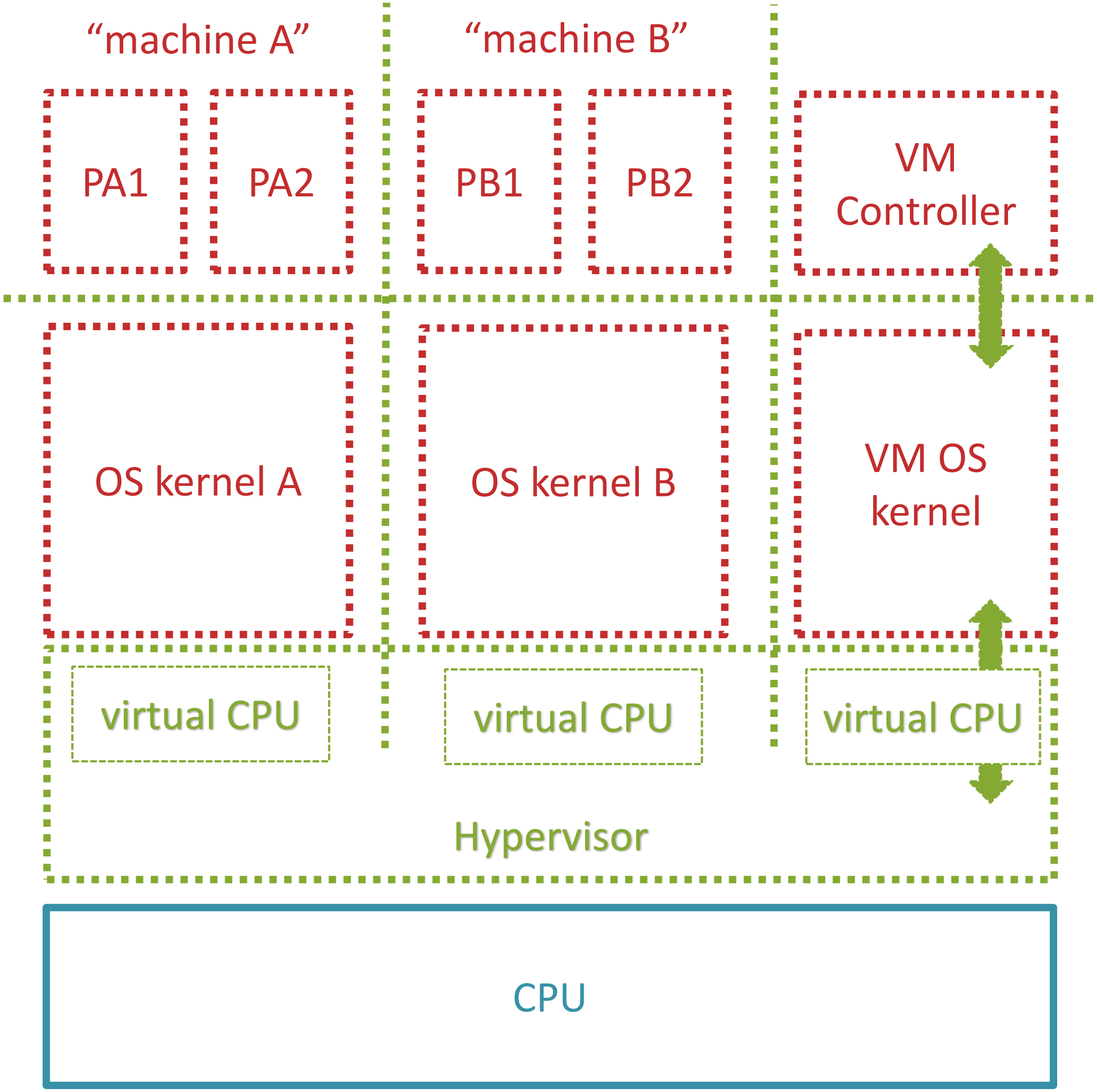
VMM implementation

Para-virtualization



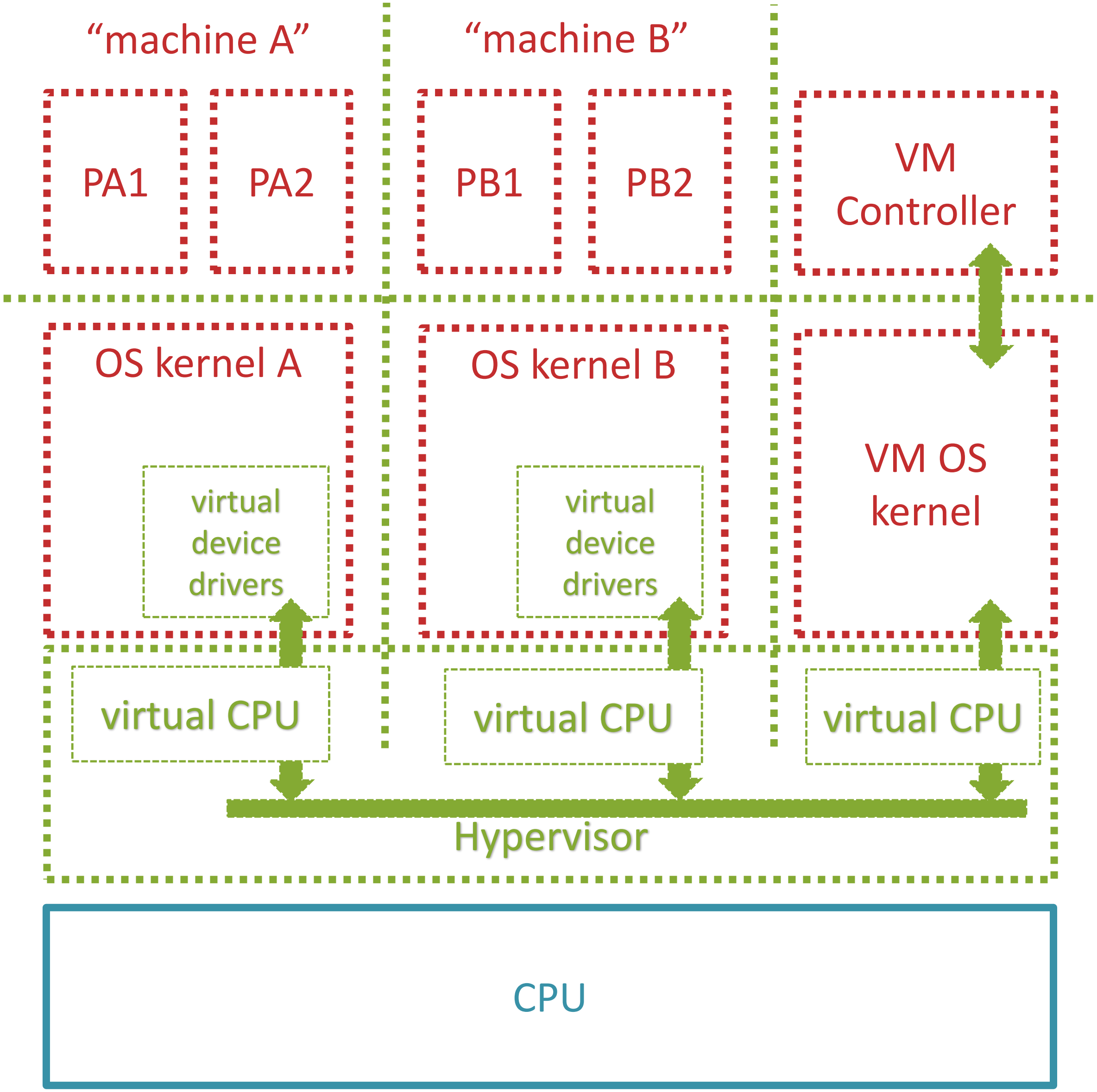
- Para-virtualization
  - Lower layers of OS kernels are modified
  - Instead of controlling hardware, these layers call the hypervisor
- Hypervisor (VMM)
  - Creates an illusion of a machine dedicated for each kernel
  - The illusion is not perfect; difficult parts replaced by cooperation of the modified kernel
- VM controller
  - Provides administrator control
- VM OS kernel
  - Provides file and network services for the controller and hypervisor

True virtualization



- True virtualization
  - OS kernels directly work with virtual CPUs and other HW
- Hypervisor (VMM)
  - Creates an illusion of a machine dedicated for each kernel
  - The illusion is perfect, emulating every bit of CPU and other HW
  - Modern physical CPUs help creating this illusion
- VM controller
  - Provides administrator control
- VM OS kernel
  - Provides file and network services for the controller and hypervisor

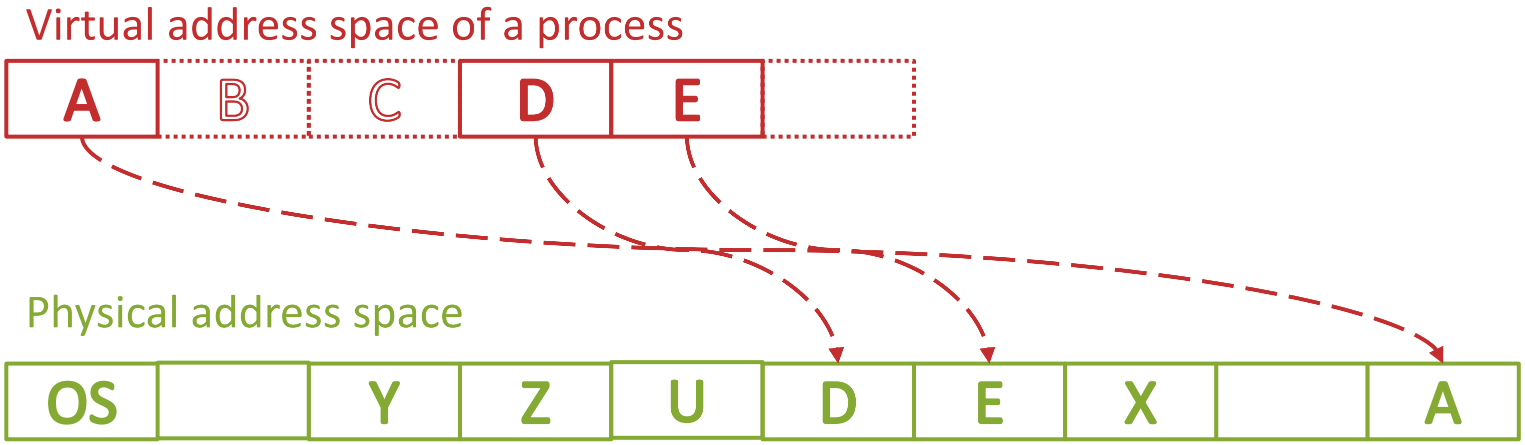
Reality: Mixed true and para-virtualization



- Hardware support makes CPU virtualization "easy"
  - Negligible overhead
  - Implementing a hypervisor is still a tremendous task
- This does not apply to most other HW
  - OS kernels "modified" by the means of device drivers
  - Actions forwarded to the VM OS
- Fast communication infrastructure
  - Implemented in the Hypervisor

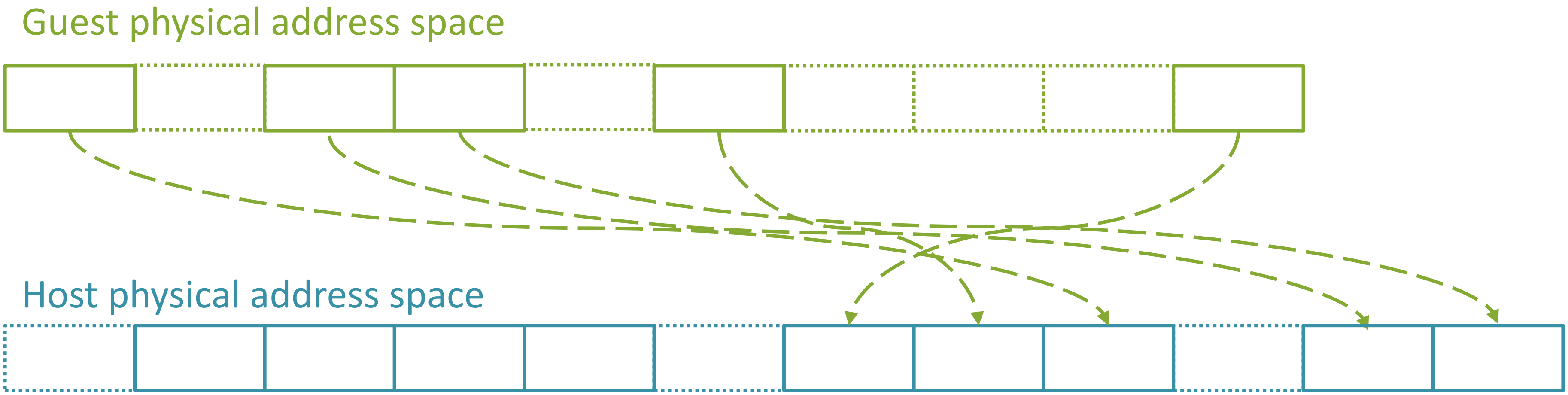
Virtualization of Virtual Memory

Virtual memory in a physical computer



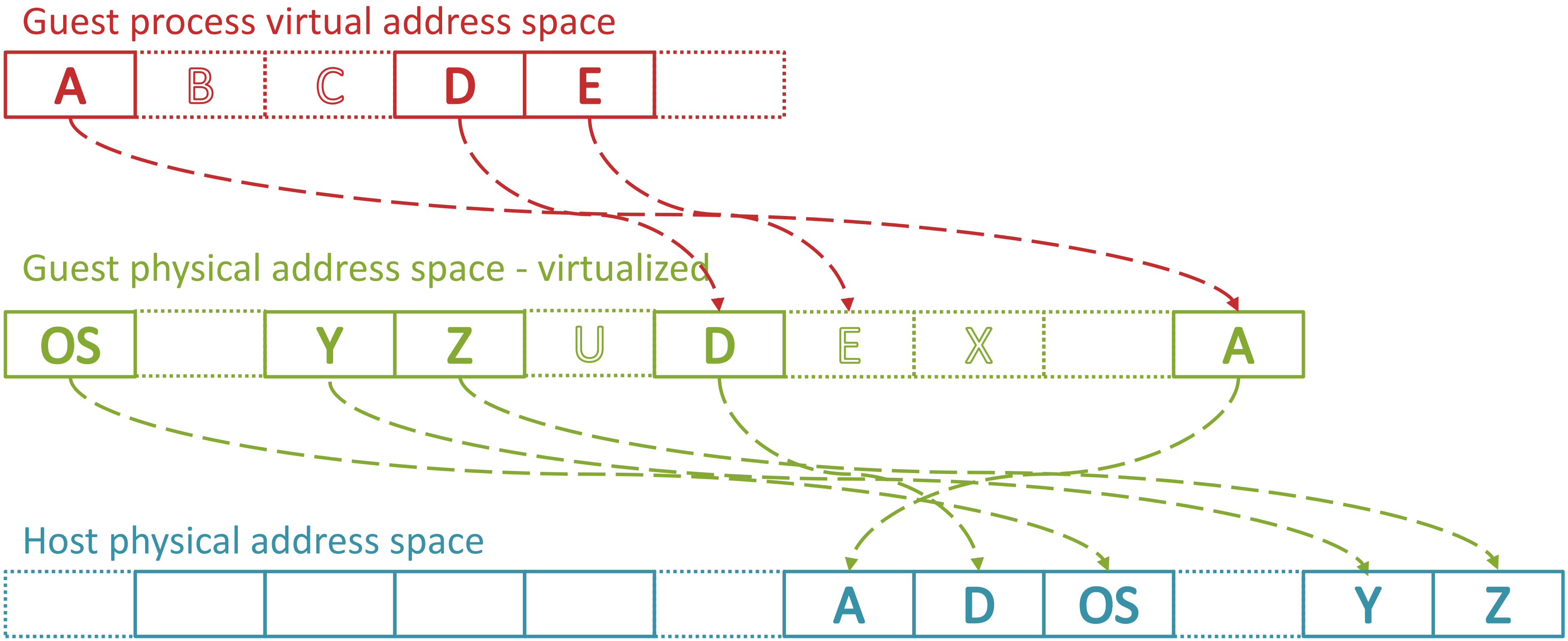
- All code works in a virtual address space, including the OS kernel
- OS defines the mapping of virtual to physical addresses (including for itself)
  - Intel/AMD: 2 to 5 levels of page tables, stored in the physical memory
    - CPU translates addresses using the TLB in most cases
    - On a TLB miss, the CPU will read the page tables to fill the TLB
    - On a page-table miss, the CPU will wake-up the OS by executing a synchronous interrupt

Memory as seen by the hypervisor



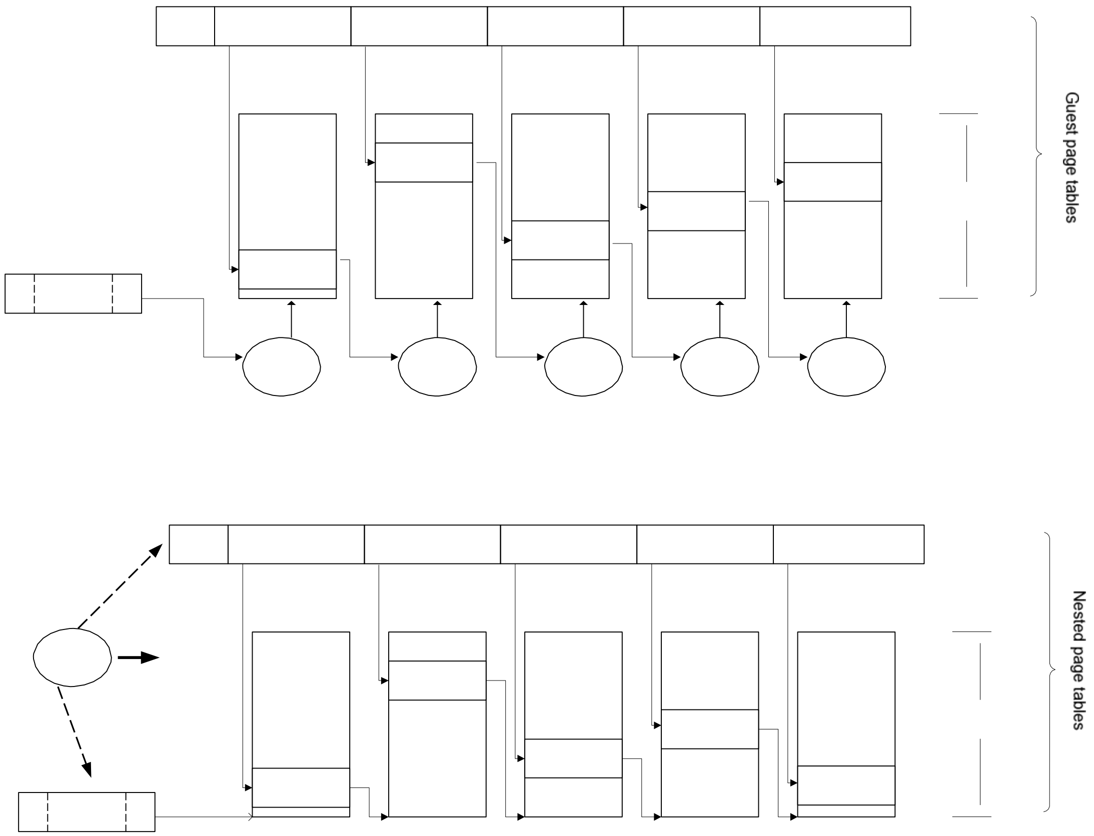
- The hypervisor must allow co-existence of several VMs
- The physical address space of each VM is virtualized
  - The mapping is defined by the hypervisor
  - An equivalent of page mapping by an OS

Virtual memory in a VM


Guest process virtual address space
A
B
C
D
E
Guest physical address space - virtualized
OS
Y
Z
U
D
E
X
A
Host physical address space
A
D
OS
Y
Z

- A composition of two mappings
  - The mapping defined by the guest OS for a process
  - The guest-host mapping defined by the hypervizor

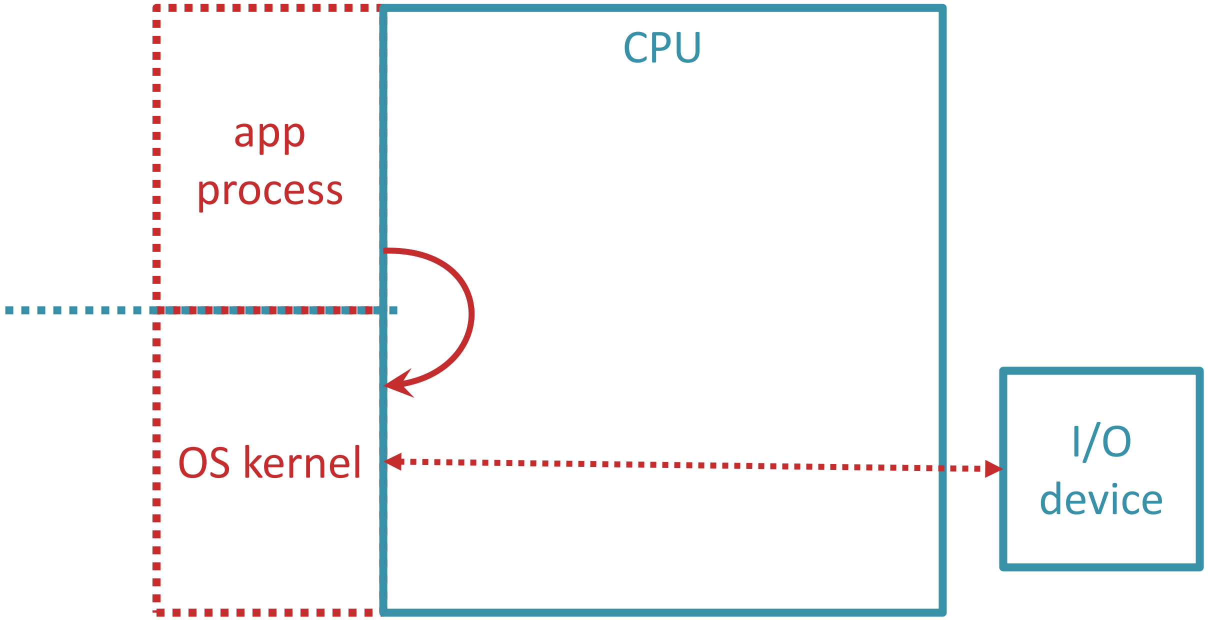
Virtual memory in a VM – EPT (Intel) / NPT (AMD)



- The CPU holds two mappings and performs the composition
  - Each (guest-physical) address in the GPT (starting with the CR3) must be translated by the NPT (to a host-physical address)
    - For 5 levels, 25 memory-accesses required!
  - The TLB stores the composed mapping

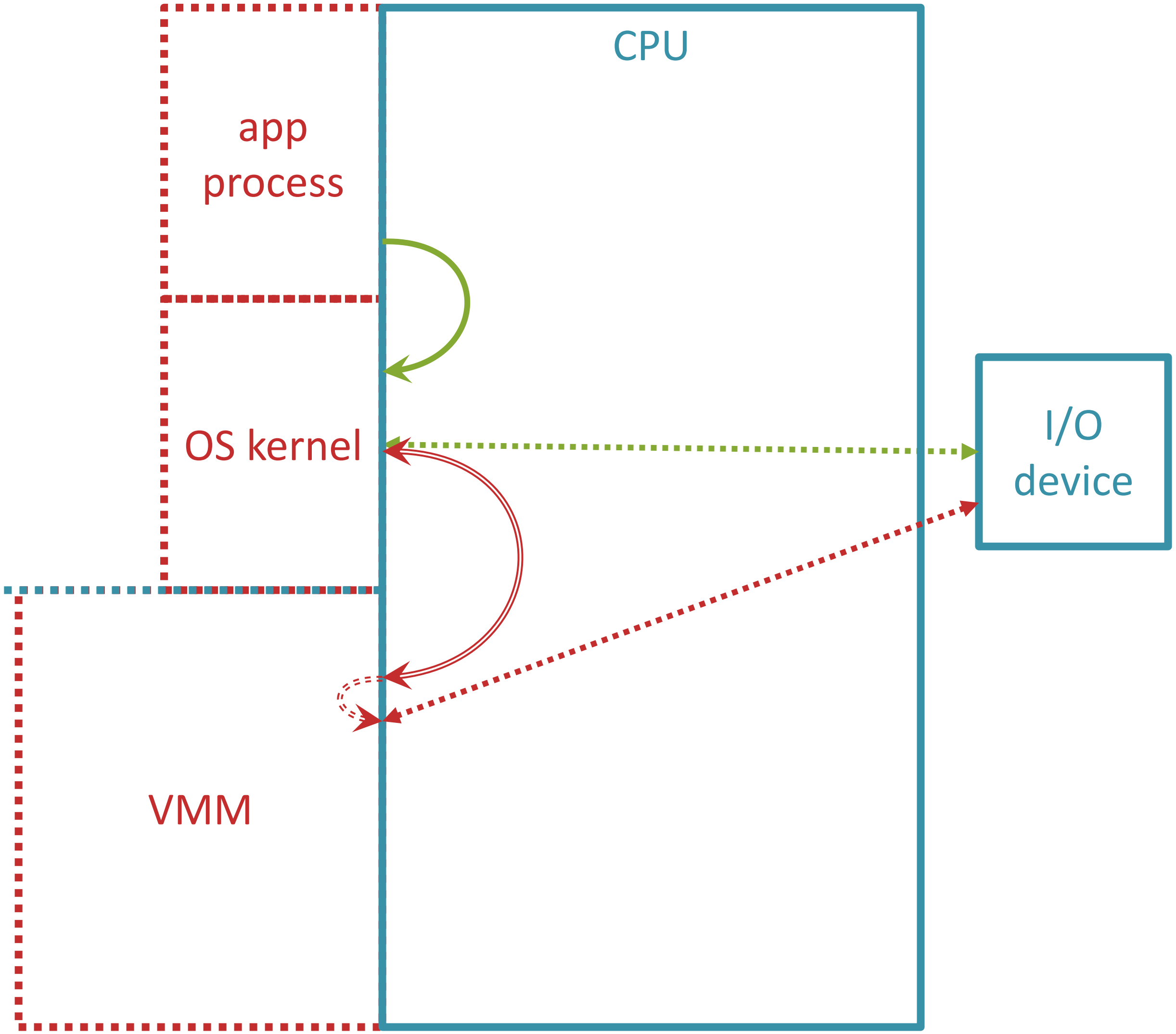
Virtualization of I/O

I/O access in a physical computer



- App processes must perform I/O by invoking an kernel syscall
- The OS kernel communicates with the I/O device
  - I/O instructions (privileged), or
  - Memory-mapped I/O device (protected by virtual memory mapping)

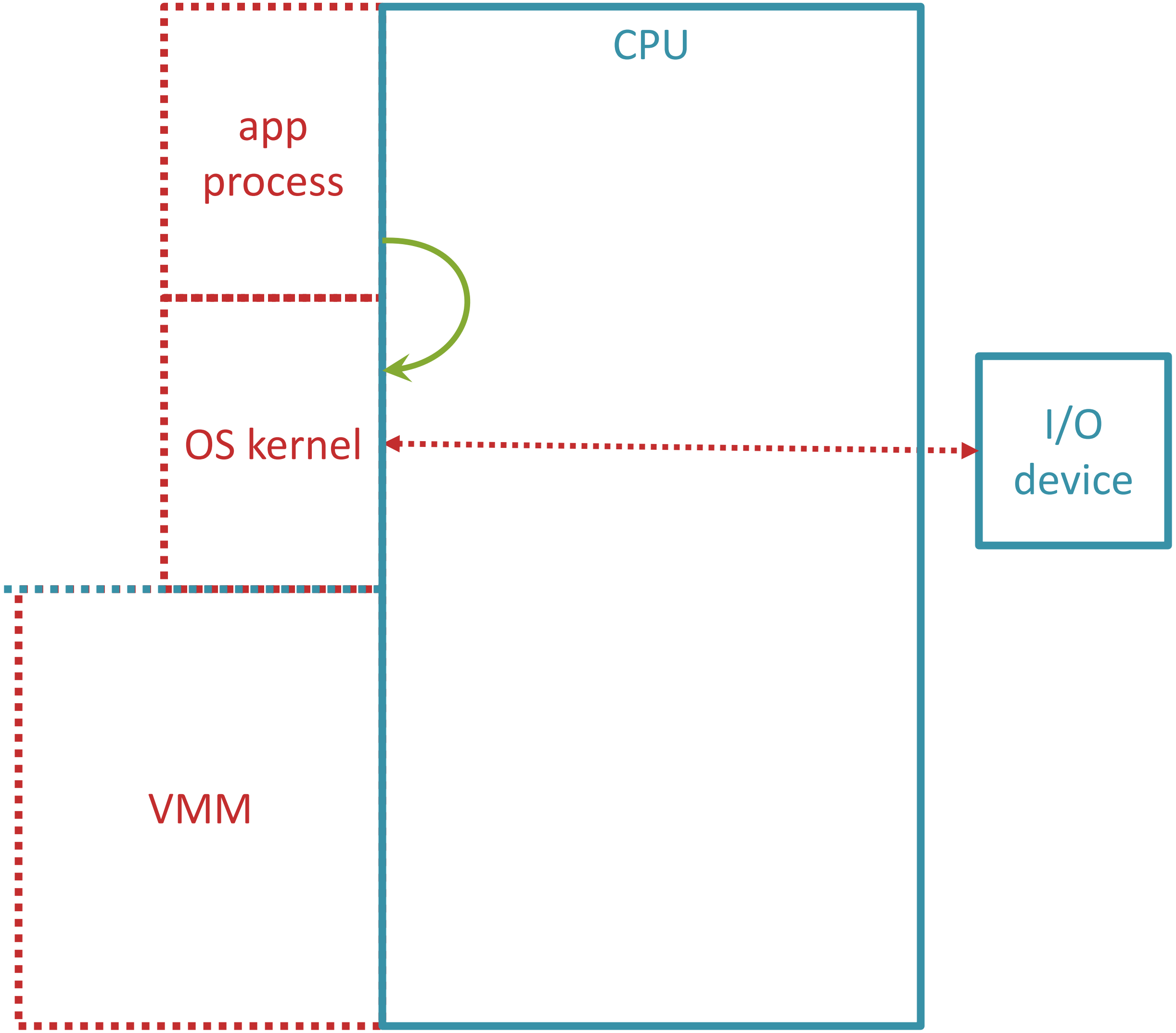
I/O access in a VM – exclusive mode



- Privileged I/O instructions cause synchronous interrupts
- Executed by an instruction emulator in the VMM
- Besides the I/O device, the related interrupt system and/or DMA controller must also be virtualized

- Exclusive mode
  - Only one VM can access the device

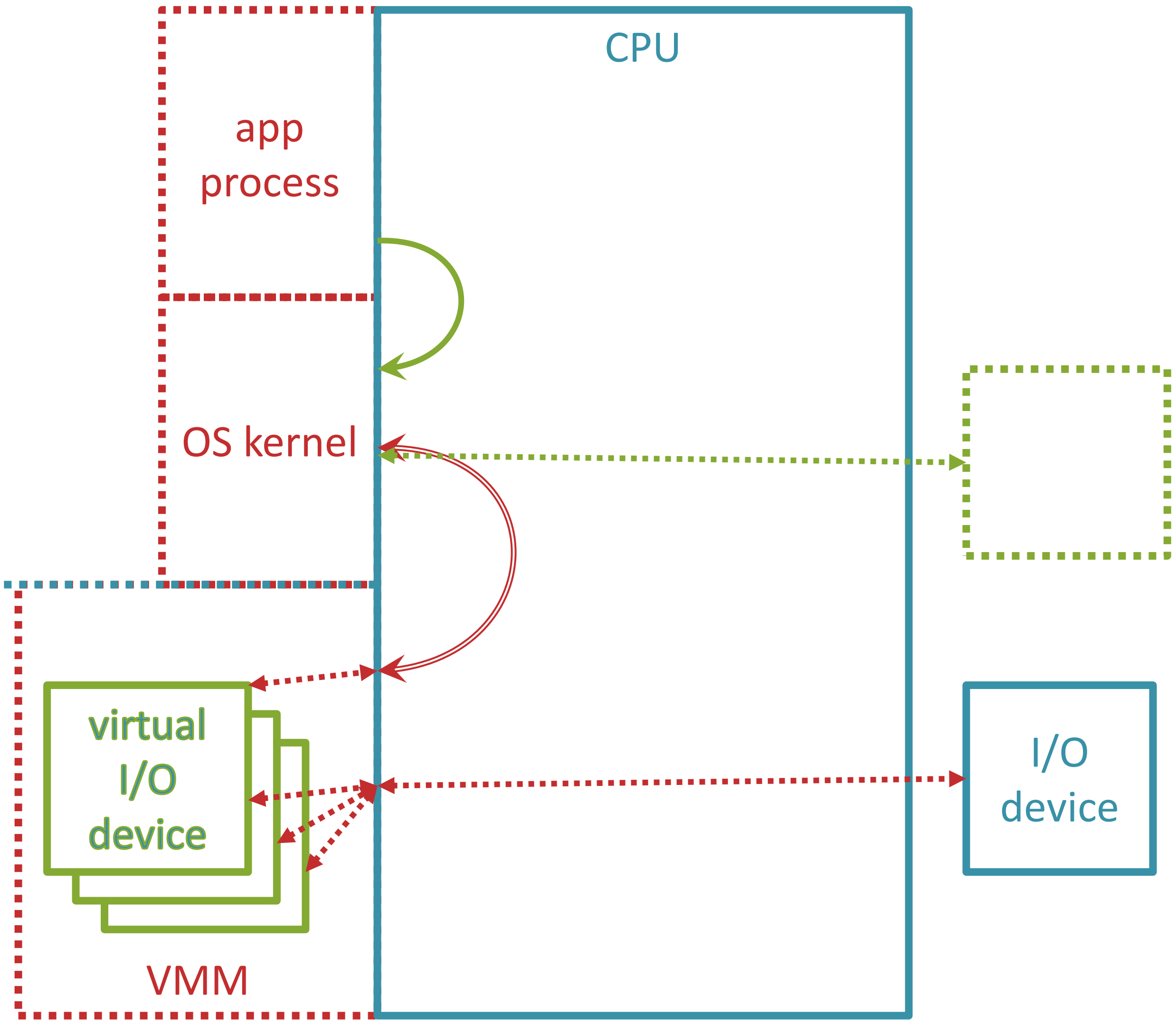
I/O access in a VM – exclusive mode



- When the communication with the device (including the DMA etc) is possible using non-privileged instructions
  - Memory-mapped devices, or
  - configurable access into I/O address space

- Exclusive mode
  - Only one VM can access the device
  - Suitable for the host-OS running in a privileged VM

I/O access in a VM – shared mode

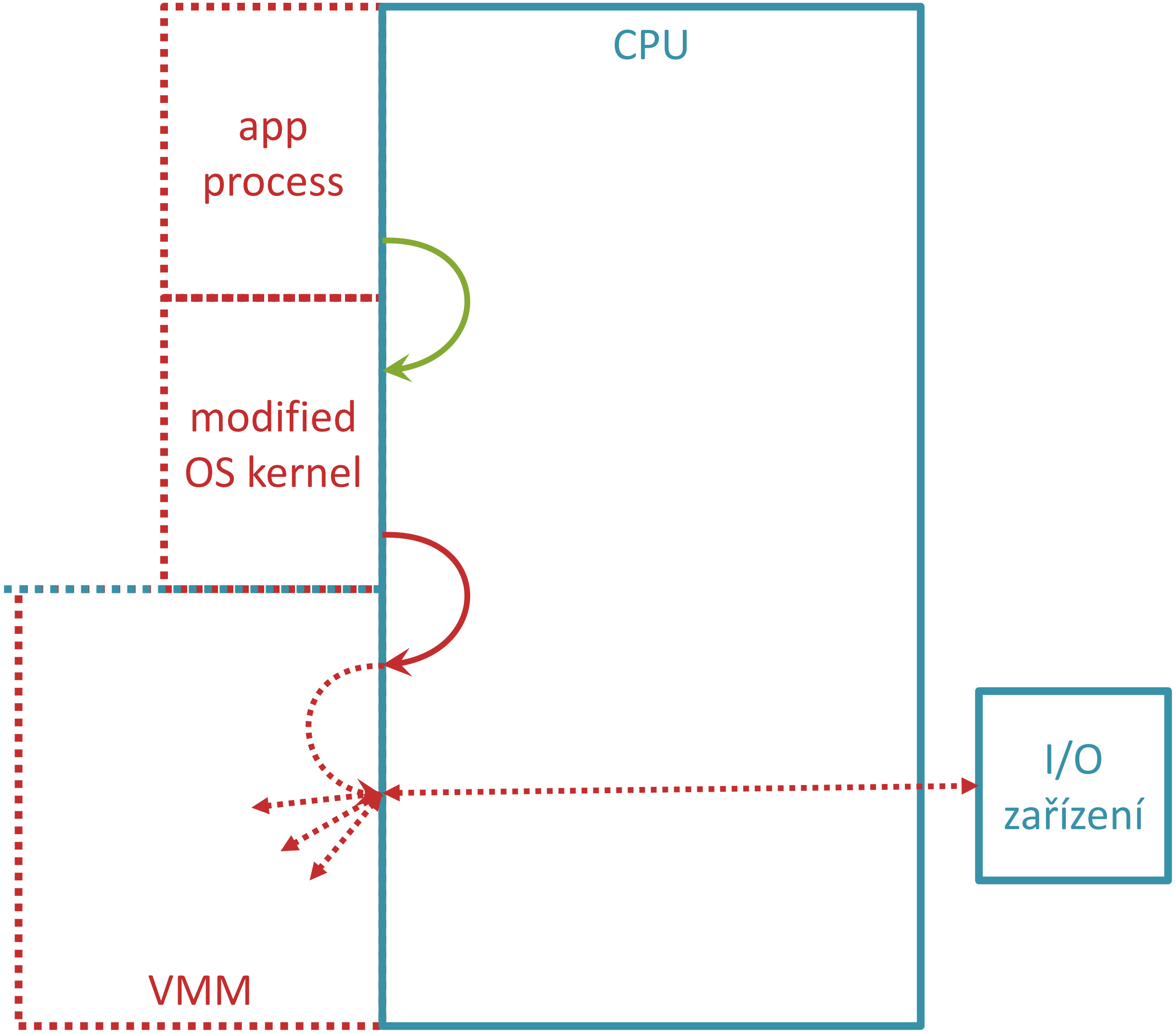


- Access to the I/O device caught and emulated by the VMM
- The emulation state is independent for each VM
- The emulated type of hardware does not have to exactly match the physical device

Shared mode

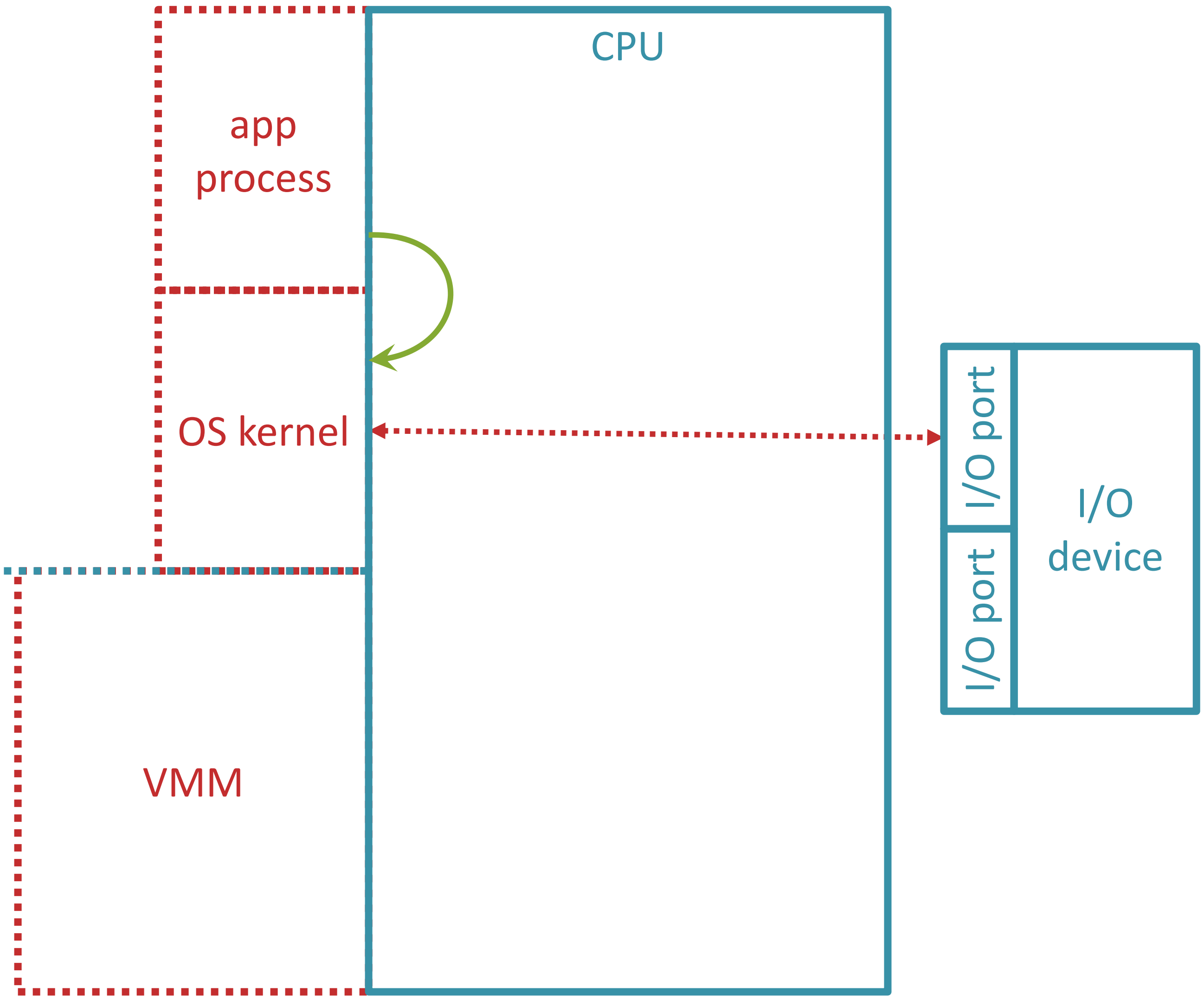
- VMM extracts logical actions from the emulated virtual devices
- The logical actions are performed by the physical device

I/O access in a VM – shared mode, para-virtualization



- Guest OS modified
  - Modified source code, or
  - a device driver for a non-existent device
- Advantages
  - The modified guest OS sends logical commands instead of physical I/O
  - Emulation of I/O instructions not needed
  - Single logical command instead of a sequence of I/O instructions
  - Synchronization of logical commands from different VMs is simpler

I/O access in a VM – multi-port I/O device

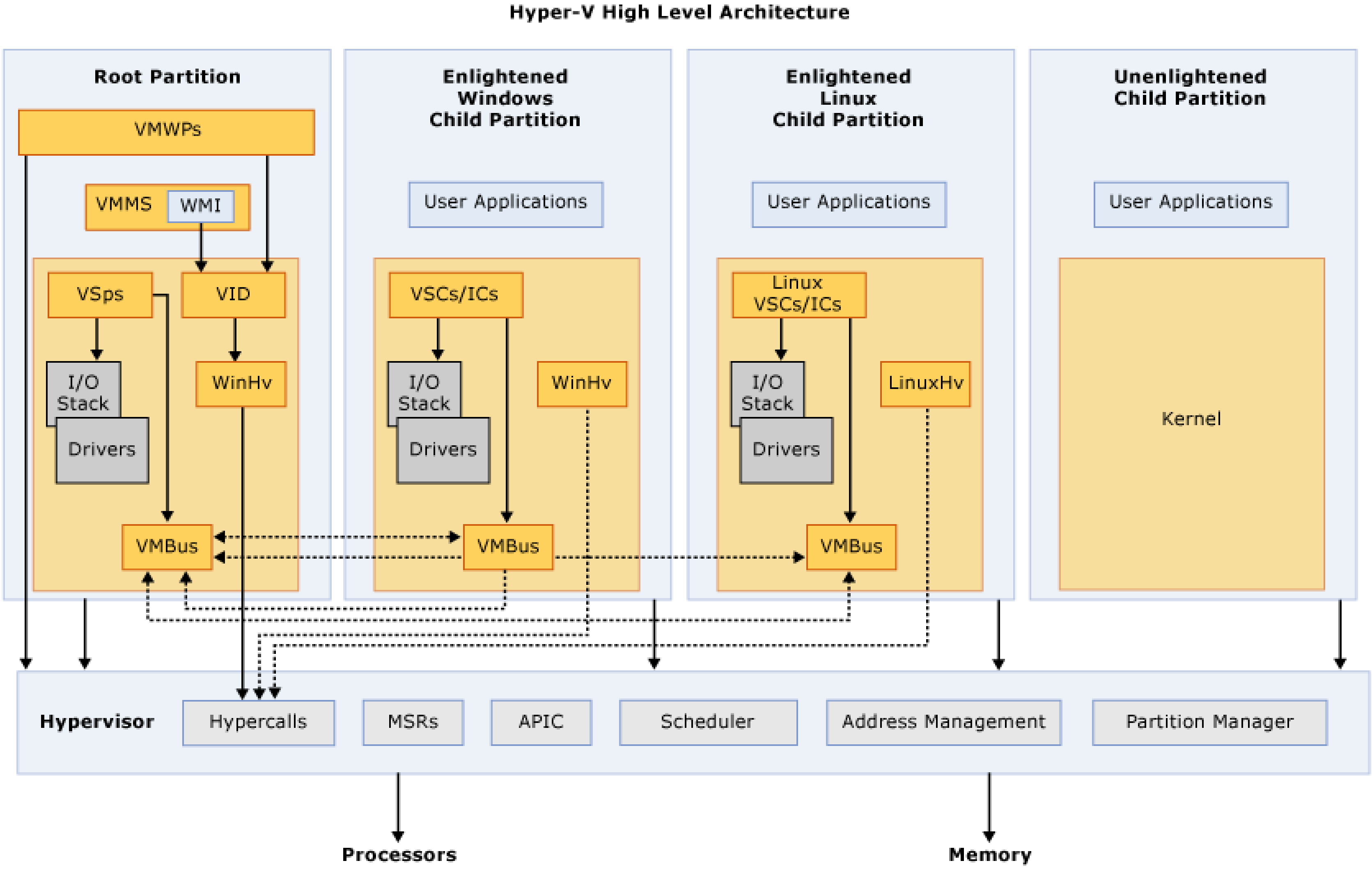


- Non-privileged access to I/O
- Configurable I/O space protection required

- Shared mode
  - The I/O device presents itself more than once in the I/O space - ports
  - The I/O device maintains an independent state for each port
  - The I/O device synchronizes logical commands from the ports on the shared physical device
  - Expensive hardware
    - Mostly NICs

VM-VMM Communication
(Example: Microsoft Hyper-V)

Microsoft Hyper-V



Guest-to-Hypervisor Interface (Microsoft Hyper-V)

- Partition
  - A set of virtual processors and other hardware, plus its configuration
  - Root partition – typically used to run the Host OS and VM Management
- Inter-partition messaging
  - The hypervisor supports a simple message-based inter-partition communication mechanism.
  - Messages can be sent by the hypervisor to a partition or can be sent from one partition to another.
- Guest Physical Address Space
  - The GPA mappings are defined by the partition’s parent.
    - At the time they are mapped, they are specified in terms of the parent’s GPA space.
- Guest Virtual Address Space
  - The hypervisor exposes operations to flush the TLB (on one virtual processor).

Guest-to-Hypervisor Interface (Microsoft Hyper-V)

- Virtual MSRs
    - Physical MSRs used by Kernels to read/alter CPU configuration
  - VMM emulates additional Machine Status Registers (MSR) not present in HW
    - VMM-aware VM Kernel can read/write virtual MSRs to exchange configuration information with VMM
  - Emulation too slow for real communication
- Hypercall
  - Call Hypervisor from Guest (privileged mode)
  - Exposed as procedure call to a special guest-physical page
    - Provided by Hypervisor on request from Guest (via a virtual MSR)
    - VM Kernel must map the guest-physical page to a guest-virtual page
    - The page contains either special instructions or nothing – both cases cause VM exit
  - Arguments passed/returned in registers or VPAP
- Virtual Processor Assist Page (VPAP)
  - Special guest-physical page per virtual processor (core/logical thread)
    - Both Hypervisor and Guest can read/write

Guest-to-Hypervisor Interface (Microsoft Hyper-V)

- Hypercall
  - Call Hypervisor from Guest (privileged mode)
  - Exposed as procedure call to a special guest-physical page
  - Arguments passed/returned in registers or VPAP
  - One Hypercall may serve several logical requests
    - Chained into an array of arguments
  - All Hypercalls return within 50 microseconds
    - Avoids blocking in the Hypervisor (giant lock?)
    - Longer requests serviced in continuation-style
      - The Hypercall return address is set before the instruction that invoked it
      - Arguments adjusted to indicate that part of the request is already done
      - On the next VM Entry, the Hypercall is entered again