THE UNIVERSITY *of* EDINBURGH
informatics

Operating Systems (INFR10079)

2023/2024 Semester 2

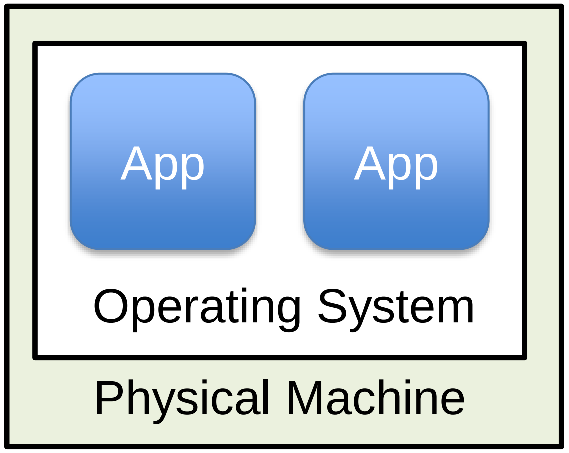
Virtualization

Kim Stonehouse kstoneho@ed.ac.uk,
Antonio Barbalace abarbala@ed.ac.uk
Luo Mai luo.mai@ed.ac.uk

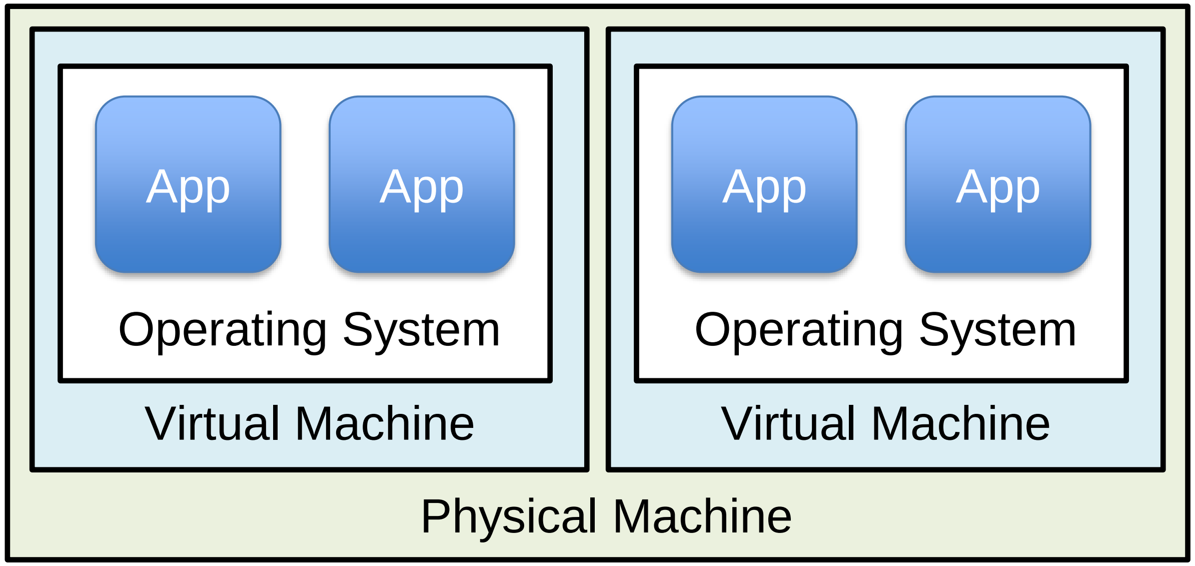
Chapter 18

Introduction

- **Virtualisation** is the process of creating a **virtual** version of a **physical** object
- In computing, **hardware virtualisation** is the process of creating a **virtual version** of real hardware
- Like real hardware, this virtual hardware may run a complete operating system and its applications



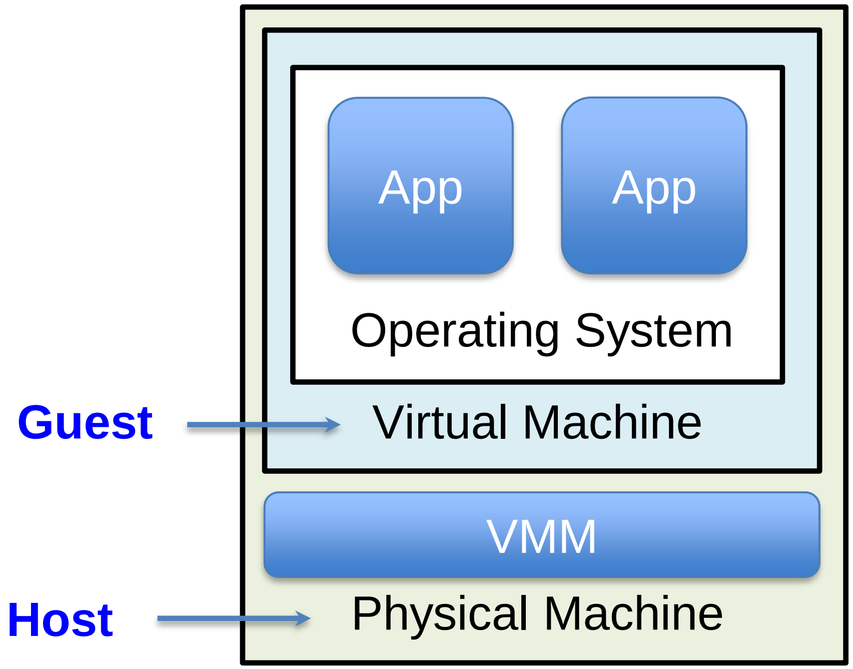
Without virtualisation



With virtualisation

Terminology

- **Virtual Machine**: a virtual instance of a physical machine
- **Virtual Machine Monitor (VMM),** or **Hypervisor**: monitors and manages running virtual machines
- **Host Machine**: the physical machine that a virtual machine is running on
- **Guest Machine**: the virtual machine, running on the host machine



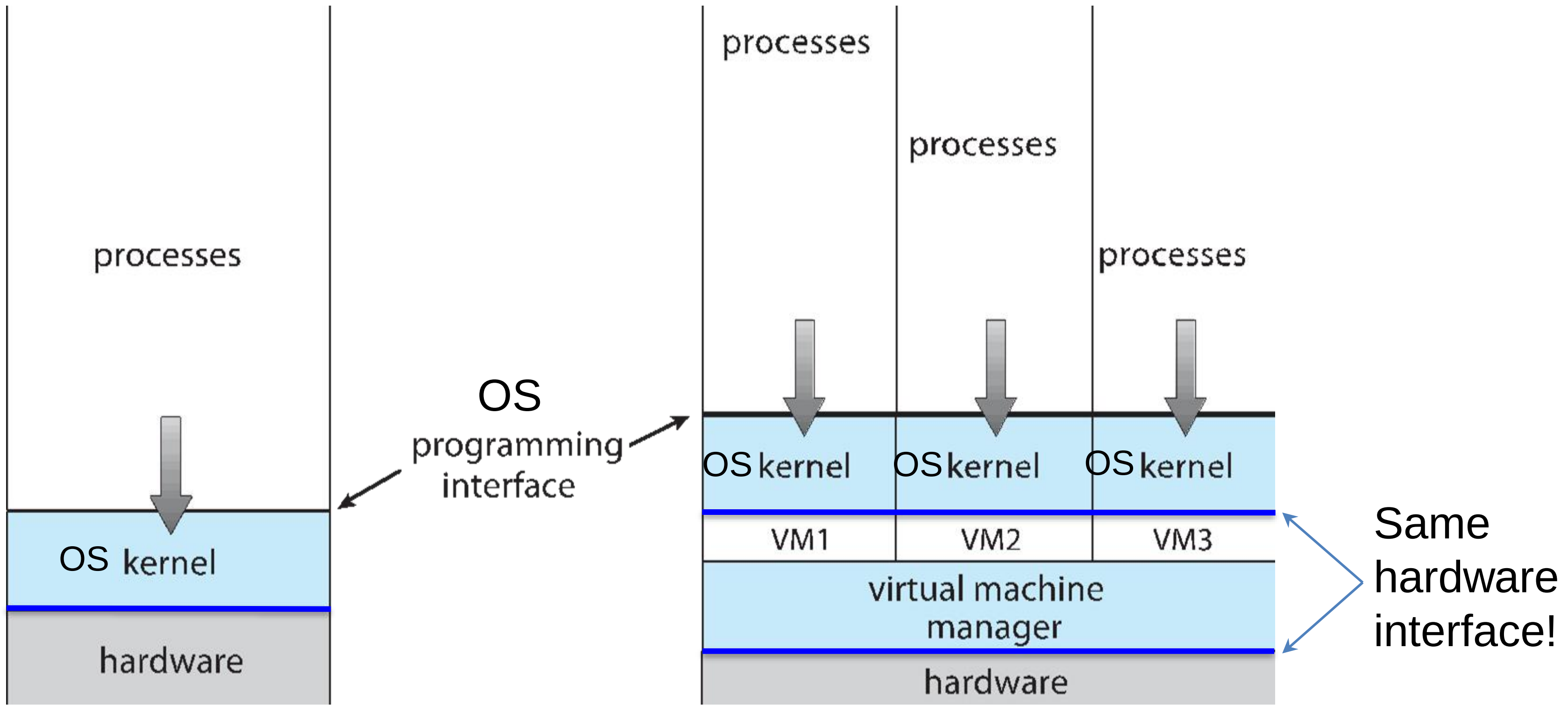
Brief History of the Idea

- **First developed in the 1960s**
  - Mainframes underutilized
  - Virtualization was developed to facilitate time-sharing
- **IBM led the way with CP/CMS**
  - Control Program/Cambridge Monitor System
  - CP was essentially the VMM
  - CMS was a lightweight operating system
- **Resurgence with cloud computing**
  - Servers are too powerful for any one workload to exploit
  - Virtualization techniques used heavily in datacentres
  - Server virtualization is ubiquitous, used by 92% of businesses (pre-COVID report)
    - https://www.spiceworks.com/marketing/reports/state-of-virtualization/

How is this different from previous material?

- Sharing resources
  - Both operating systems and virtual machine managers share resources between competing entities
  - So how do they differ?
- OS decouples
  - OS provides abstraction for file, address space, process/thread, etc, providing a **programming interface different to the real hardware interface**
- VMM does not
  - Whereas the VMM uses real hardware interface, and guests use the **exact same hardware interface as before**

How is this different from previous material?

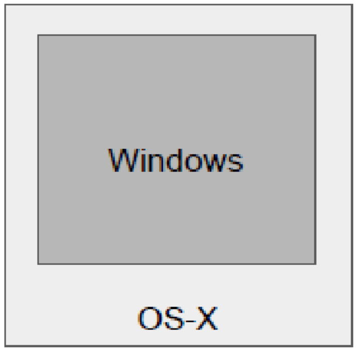


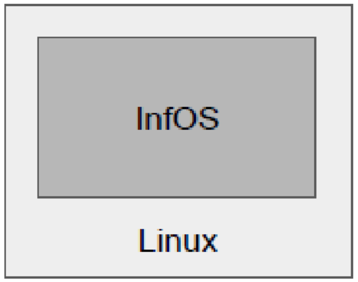
Without virtualisation

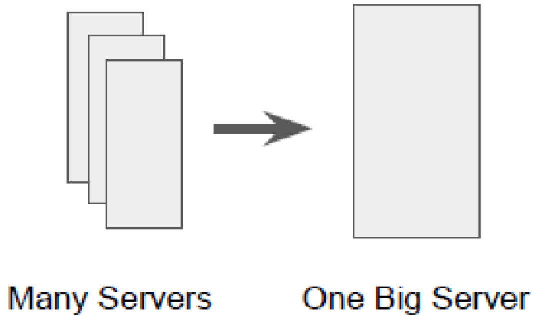
With virtualisation

How can virtualisation be used?

- **Personal** (e.g. Parallels Desktop/VirtualBox)
  - Running multiple operating systems on one host, without the inconvenience of rebooting
  - e.g. Running Windows inside OS X
  - Some hypervisors support “seamless integration”
- **Technical** (e.g. QEMU as used in the coursework)
  - Operating System/Hardware Design
  - Kernel Debugging/Testing
  - Prototyping new architectures/architectural features
- **Commercial** (e.g. XEN/VMWare)
  - Data centre server consolidation
  - High availability/Migration



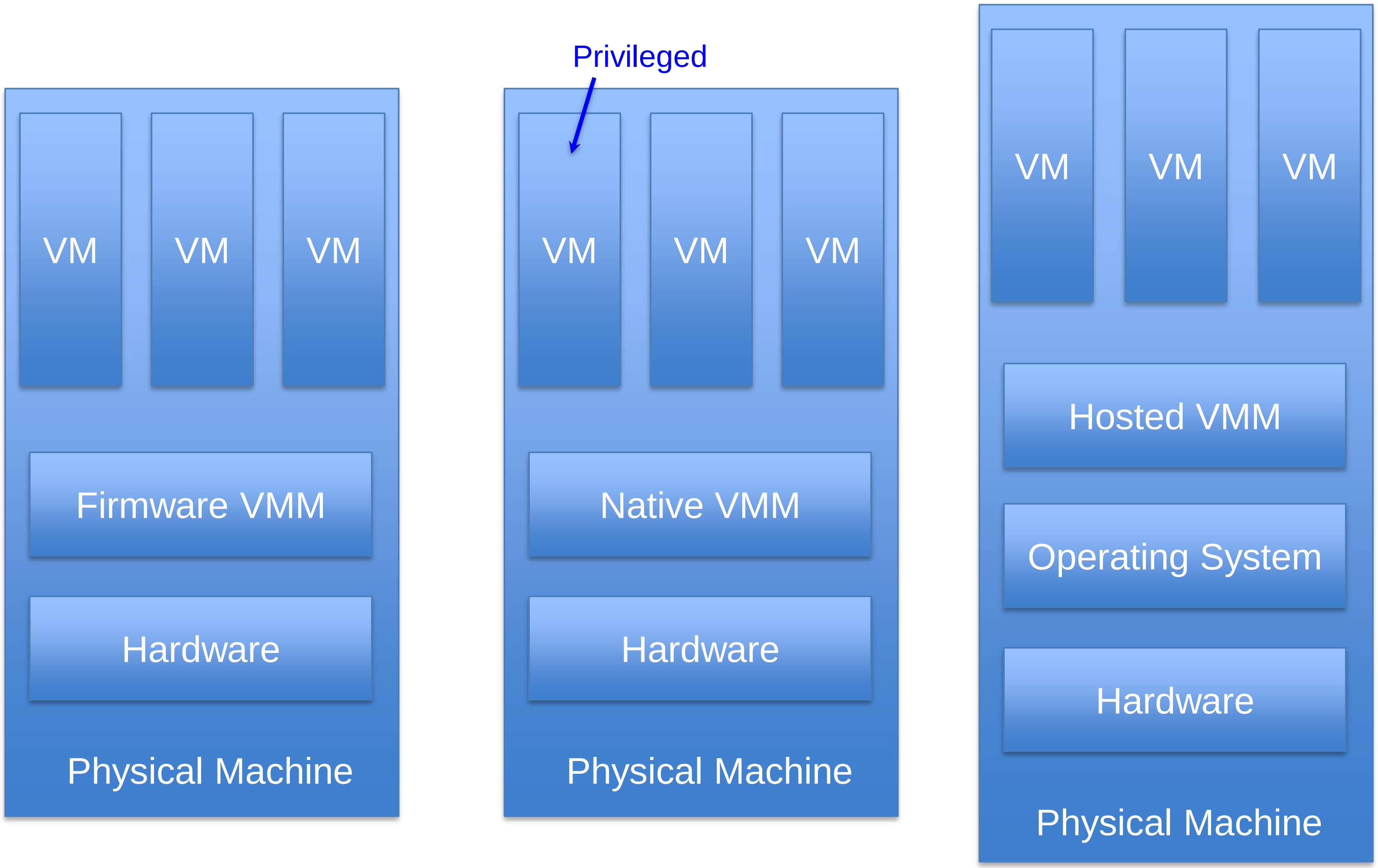




Virtual Machine Monitor (VMM)

- The **VMM** is in charge of running the **virtual machines**
- There are several types of VMM:
  - **Type 0: Firmware**
    - Provides support for virtual machine creation and management via firmware, unmanaged by software
    - Often found in mainframes and large servers
    - E.g. IBM LPARs, Oracle LDOMs
  - **Type 1: Native**
    - Software that runs directly on the physical machine
    - Shares out system resources between virtual machines
    - E.g. Xen, Oracle VM Server
  - **Type 2: Hosted**
    - Runs as an application inside an operating system
    - Support virtual machines running as individual processes
    - E.g., virtualbox, Parallels Desktop, QEMU

VMM Types



Type 0 **Type 1** **Type 2**

Accessing Resources

- VMs still need to access **physical** resources, such as
  - Memory
  - Storage
  - Networking
  - Graphics
- The VMM must **share out** these resources
  - However, **unmodified** VMs do not know they are being virtualised
  - So the VMM must **emulate** the interface that the VMs expect
  - Some devices, like **dedicated network cards**, may be passed straight through to the VMs

Methods of Virtualisation

- **Trap-and-Emulate**
  - The guest operating system runs in user mode
  - Attempting to execute **privileged** instructions traps to the VMM
  - The VMM **emulates** the attempted instruction on behalf of the guest
  - Control is returned to the guest, who resumes executing
- **Binary Translation**
  - Software examines each instruction and provides a translation
  - Very slow, but necessary for **x86** until recently
  - Still necessary for **cross-architecture virtualisation**

Hardware Acceleration (Full Virtualization)

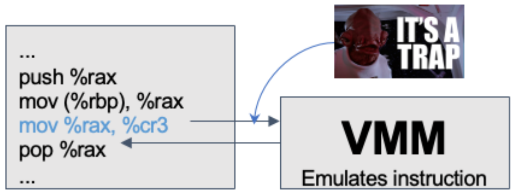
- **Virtualising x86 was problematic**
  - Software solutions were much too slow
  - Violated the **efficiency** principle
- **VTX instructions**
  - In 2005, **Intel-VT** and **AMD-V** introduced new **VTX** instructions
  - VMM can create a **control structure** for each guest, defining what instructions should trap to the VMM using a new **exit instruction**
  - Now, the VMM does not need to examine all instructions
- **What about other architectures?**
  - ARM introduced new EL2 and EL3 execution modes to support

Extra materials [Luo]

How to handle privileged instructions?

Trap-and-Emulate

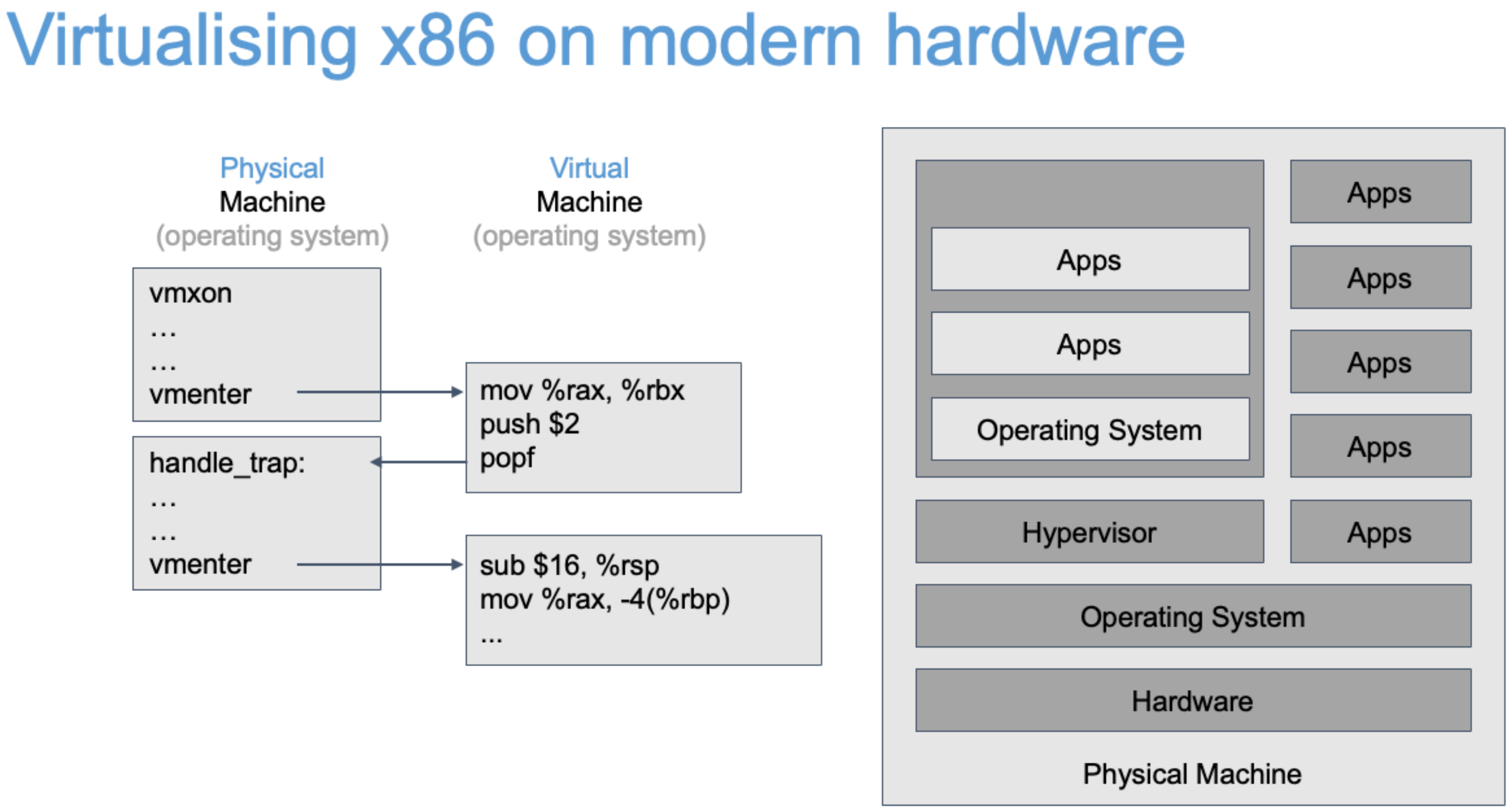
- The guest operating system runs “de-privileged”, all non-privileged instructions execute natively on the host.
- All privileged instructions trap to the Virtual Machine Manager (VMM) which implements the ”Hypervisor”
- VMM emulates these privileged operations.
- Guest resumes execution after emulation.



```
...
push %rax
mov (%rbp), %rax
mov %rax, %cr3
pop %rax
...
```

Problem: Not all privileged x86 instructions trap properly!

Extra materials [Luo]


Virtualising x86 on modern hardware

Paravirtualisation

- **Enhanced virtualisation**
  - The guest is **aware** of the virtualisation
  - The guest can now cooperate with the VMM to optimise performance
- **No need for emulation**
  - Guests no longer “trap-and-emulate”
  - They can now request privileged operations from the VMM
  - Instead of emulating devices, the VMM provides a paravirtualised implementation

Popek and Goldberg Requirements

- **Paper published in 1974 [1]**
  - Formalised the requirements for a virtual machine
- **Efficiency**
  - The majority of guest instructions should be executed directly on the host machine (so **not** binary translation)
- **Resource control**
  - The virtual machine monitor must remain in control of all resources
- **Equivalence**
  - The virtual machine must behave in a way that is indistinguishable from if it was running as a physical machine

[1] Gerald J. Popek and Robert P. Goldberg. 1974. Formal requirements for virtualizable third generation architectures. *Commun. ACM* 17, 7 (July 1974), 412-421. DOI: http://dx.doi.org/10.1145/361011.361073

Efficiency

➔ “All innocuous instructions are executed by the hardware directly, with no intervention at all on the part of the control program.”

- Normal guest machine instructions should execute natively
- System instructions need to be emulated by the VMM

Resource Control

➔ “It must be impossible for that arbitrary program to affect the system resources, i.e. memory, available to it; the allocator of the control program is to be invoked upon any attempt.”

- The guest should not be able to adversely affect the host in any way
- The host should remain in control of all physical resources, sharing them out between guests.

Equivalence

➔ “Any program K executing with a control program resident, with two possible exceptions, performs in a manner indistinguishable from the case when the control program did not exist and K had whatever freedom of access to privileged instructions that the programmer had intended.”

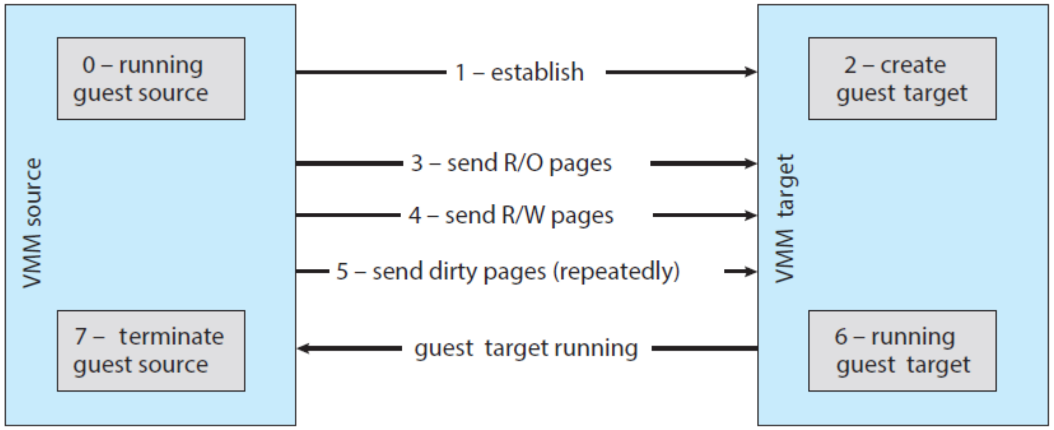
- A formal way of saying that the operating system should not be able to tell that it is being virtualised
- The two exceptions are **temporal latency** and **resource availability**

Characteristics of Virtualisation

- **Resource Optimisation**
  - **Isolation** provides the ability for **different** operating systems to share the same hardware
  - Can use achieve the same things with less physical resources
- **Security**
  - Host machine is **protected** from the virtual machines, just as the virtual machines are protected from each other
  - A **malicious** or infected guest will **not** affect the host or other guests
- **Workload Migration**
  - The VMM is able to **freeze** or take a **snapshot** of a running guest
  - Allows **moving** the guest from one host to another, with the current state intact and **no interruptions**
  - Increases **portability**
  - **Load balancing or fault prevention** in the data center

Migration of a Running Guest VM

- A running guest on one system is copied to another one
  - Running the same VMM
- Type of migrations
  - Stop/restart, or checkpoint/restart
  - Live migration (unnoticeable
    - Pre-copy
    - Post-copy
    - Hybrid-copy



VMM Guest Live-Migration

1) The source VMM establishes a connection with the target VMM and confirms that it is allowed to send a guest.
2) The target creates a new guest by creating a new VCPU, new nested page table, and other state storage.
3) The source sends all read-only memory pages to the target.
4) The source sends all read–write pages to the target, marking them as clean.
5) The source repeats step 4, because during that step some pages were probably modified by the guest and are now dirty. These pages need to be sent again and marked again as clean.
6) When the cycle of steps 4 and 5 becomes very short, the source VMM freezes the guest, sends the VCPU's final state, other state details, and the final dirty pages, and tells the target to start running the guest. Once the target acknowledges that the guest is running, the source terminates the guest.

Types of Virtualisation

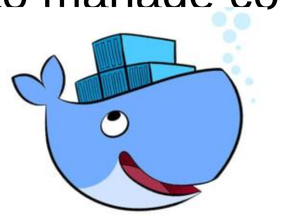
- **Machine Virtualisation** (the content until this slide)
  - Isolate multiple operating systems, within a single physical machine
  - Can be either **same architecture** or **cross architecture**
  - Includes **paravirtualisation** too
- **Container Virtualisation**
  - Isolate processes or groups of processes, within a single operating system, e.g. Docker, Linux containers
  - In Linux, **cgroups** limit and isolate the resource usage for a collection of processes
- **Language Virtualisation**
  - Using VMs to provide a **managed runtime environment**
  - Allows programming environment to be platform independent
  - E.g. **Java Virtual Machine**

Extra materials [Luo]

How can containerisation be achieved?

- provide user space abstraction for each container
  - isolated view at the system for container content

- provide a container management system to manage container instances and standardised access to contents



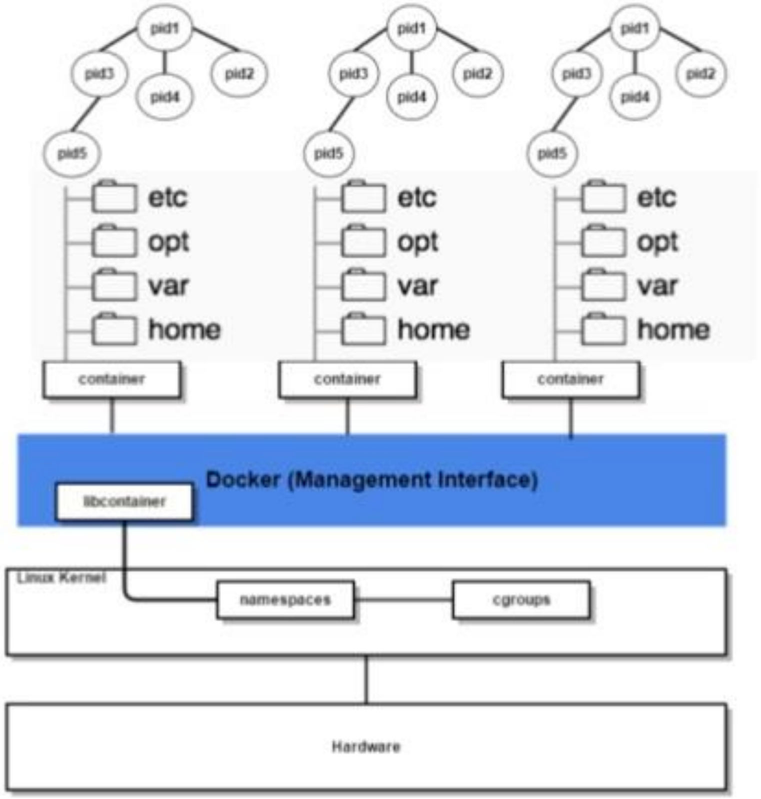
Extra materials [Luo]

How to limit resources and achieve isolation?

- **control groups** - limit what you can use
  - resource control
- **namespaces** - limit what you can see
  - isolated view at system

Extra materials [Luo]

Docker Container under the hood



pid1
pid3
pid4
pid2
pid5
etc
opt
var
home
container
Docker (Management Interface)
libcontainer
Linux Kernel
namespaces
cgroups
Hardware