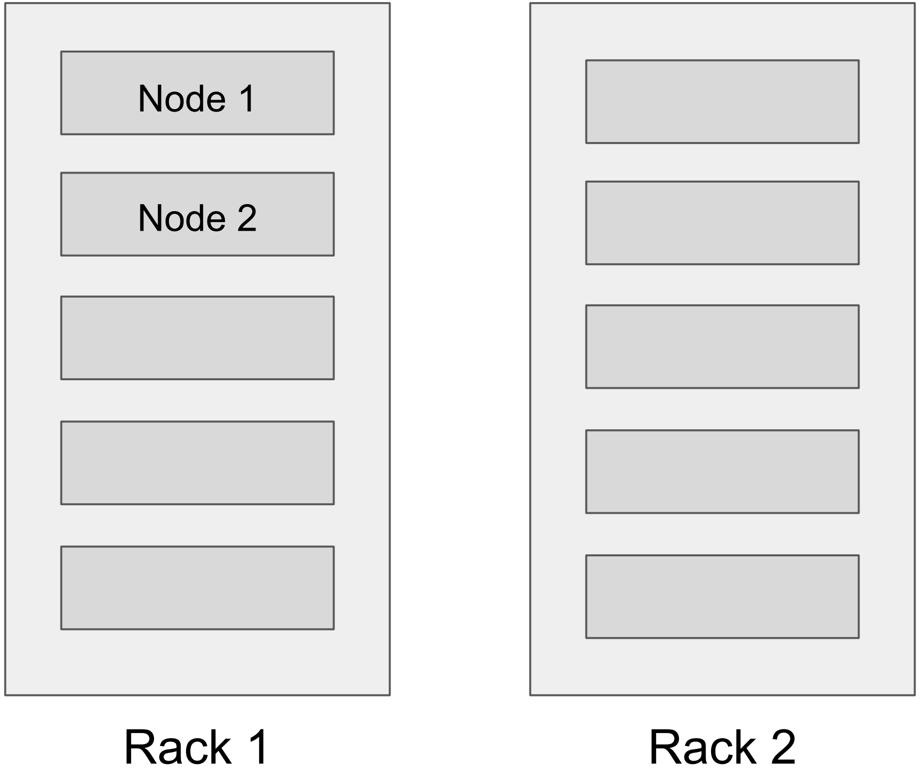
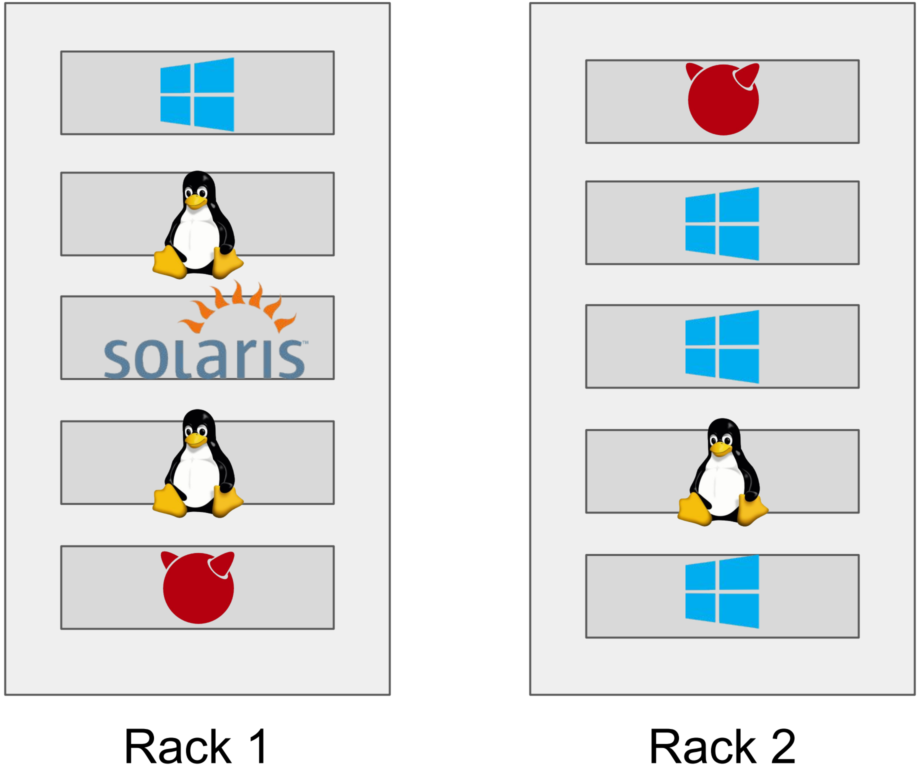
Consistency Demo

Virtualisation

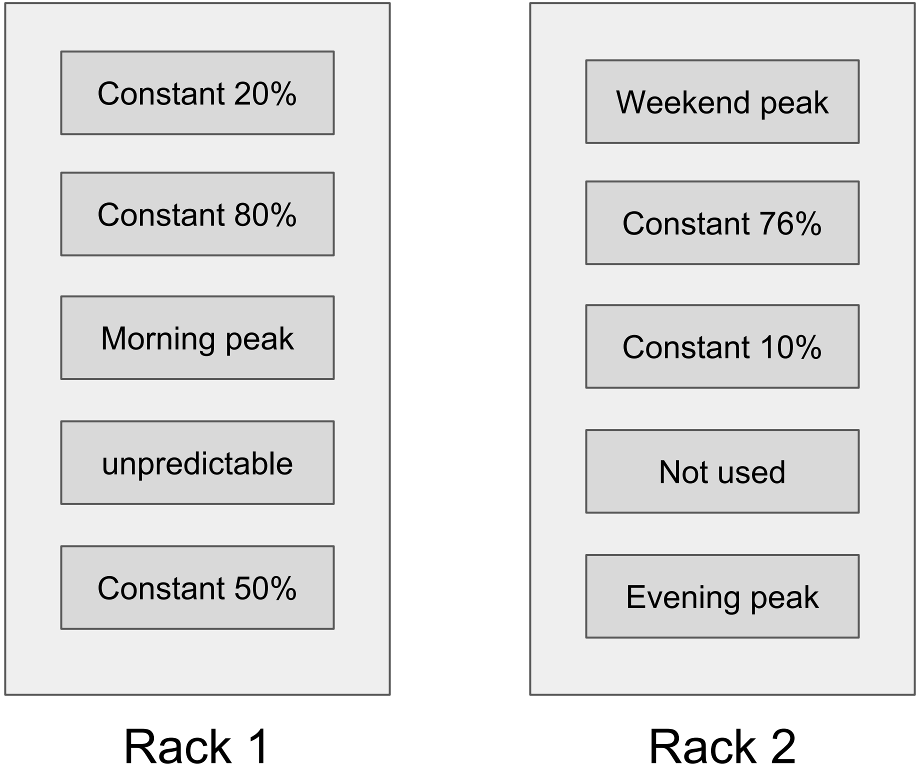
Based on Operating Systems Lecture by **Tom Spink**

Motivation

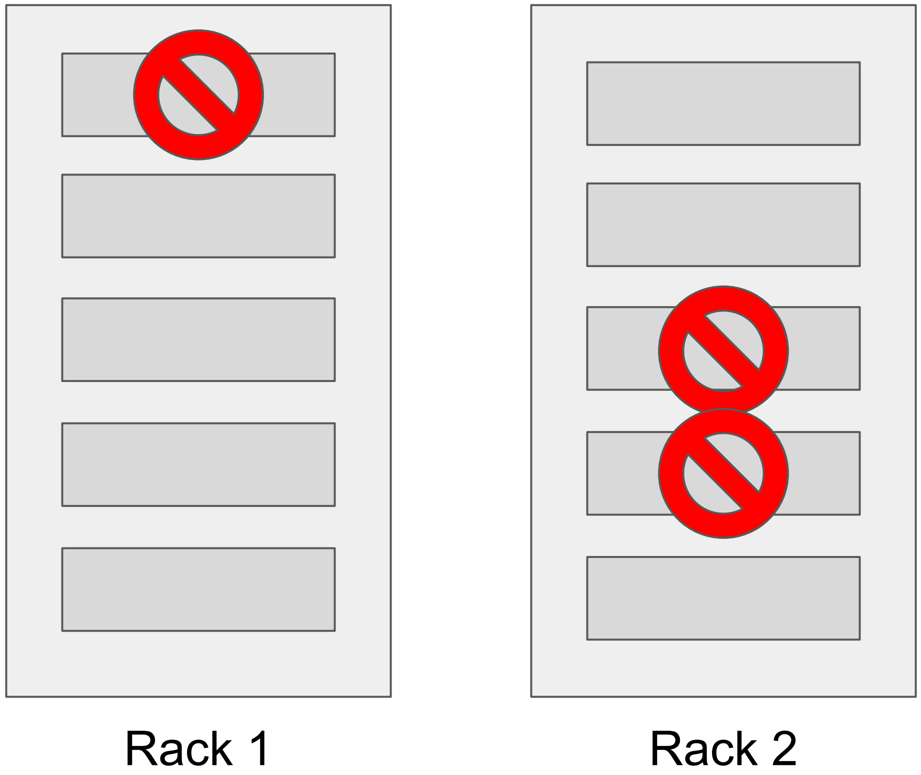




Support for broad customer base



Resource Utilisation



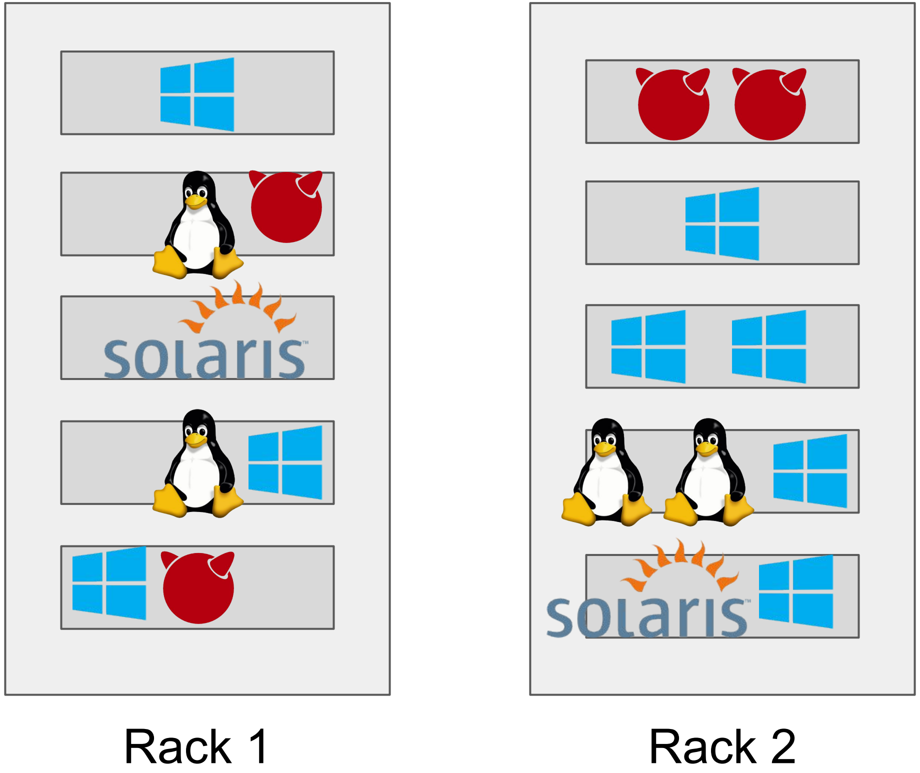
Node Failure

Problem:

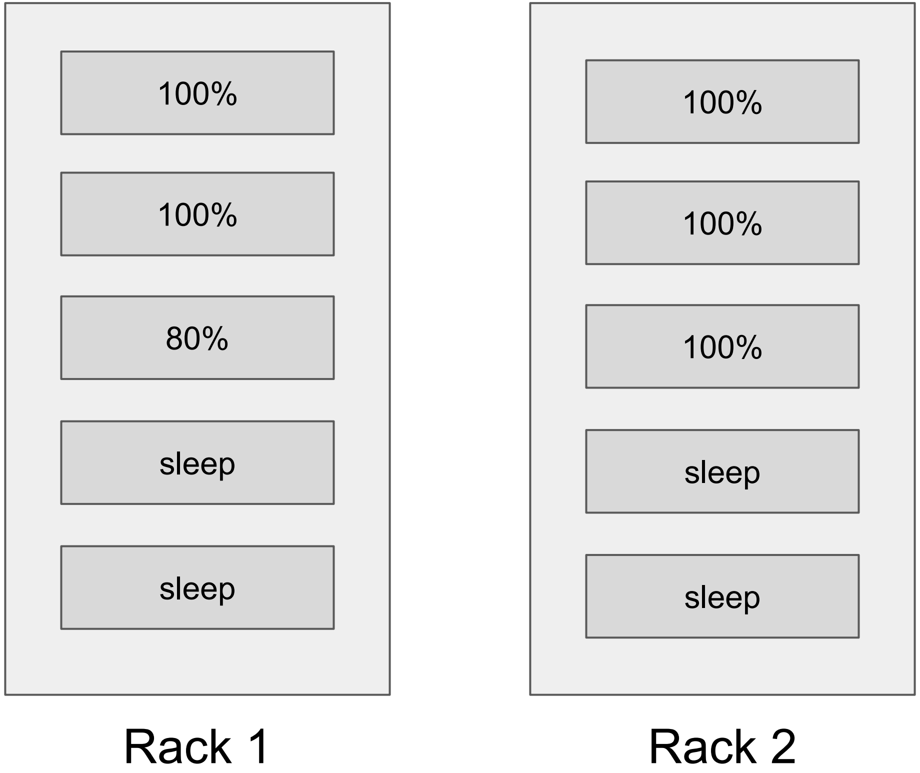
Operating System is tied to underlying Hardware

Solution?

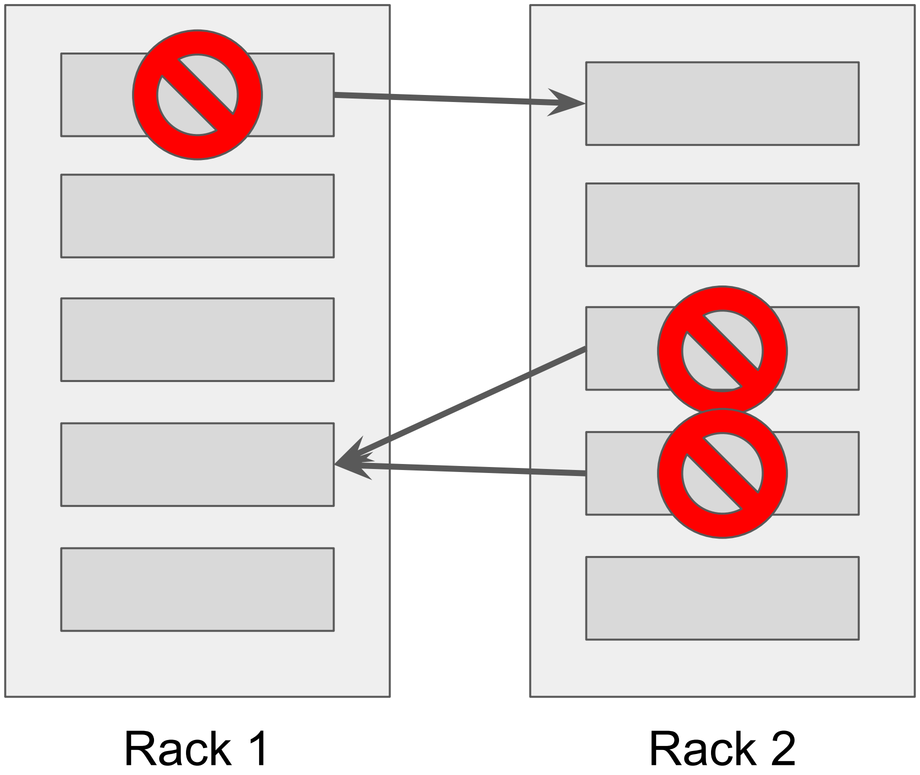
Virtuatlisation!



Virtualisation allows **flexibility**



Virtualisation allows **server consolidation**



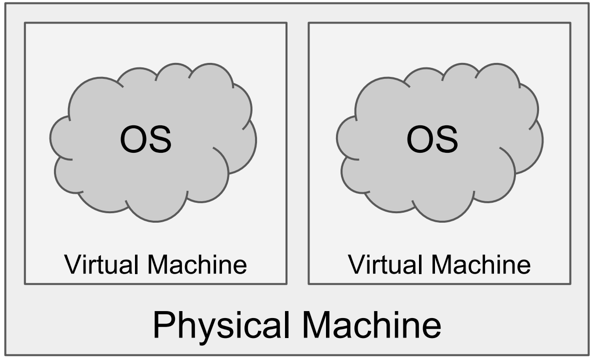
Virtualisation allows **fault tolerance**

Motivation for Virtualisation in a compute cluster

- flexibility
- consolidation
- fault tolerance

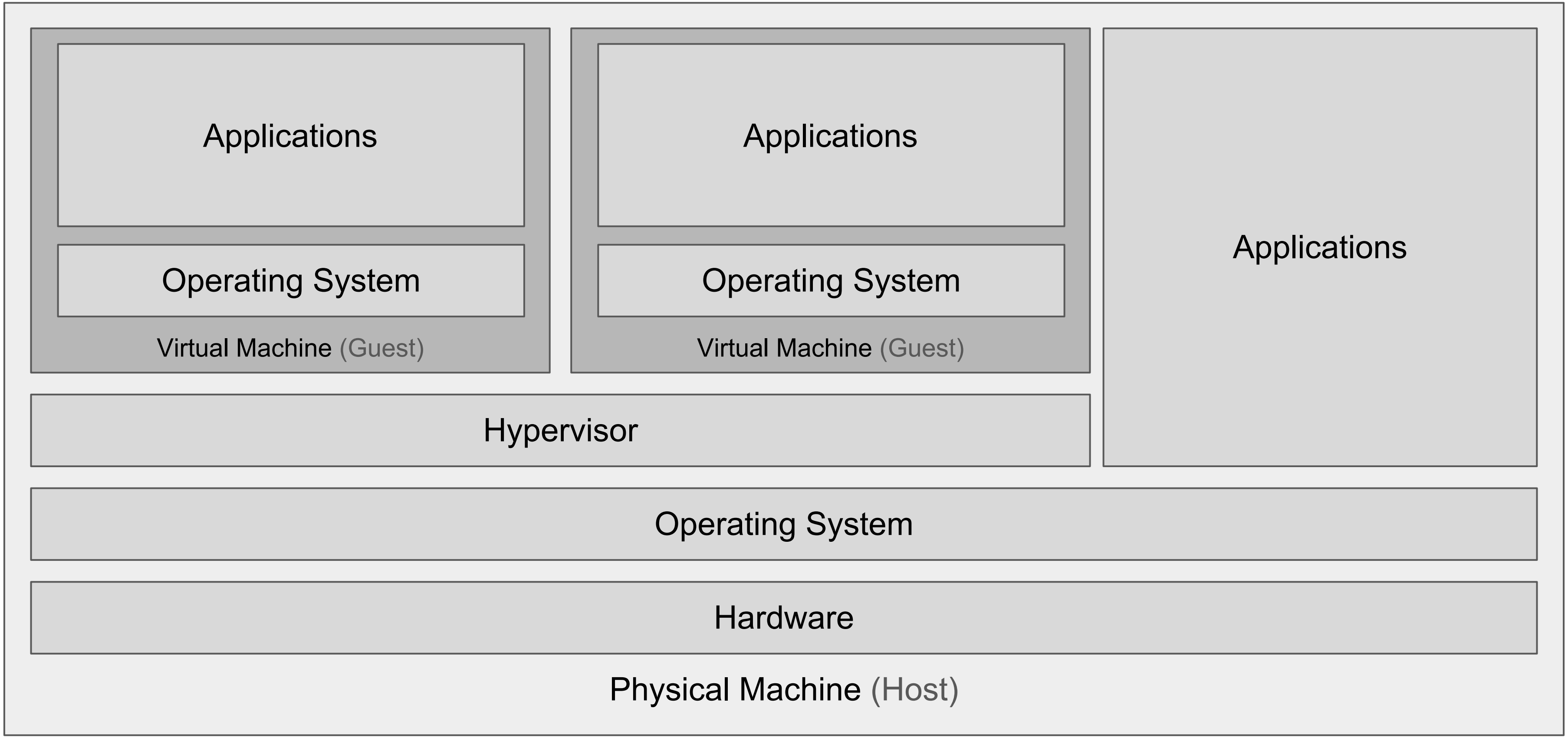
Introduction

- 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.
- This virtual hardware can be used to run a complete operating system.



Terminology

- **Virtual Machine:** A virtual representation of a physical machine.
  - Not to be confused with a **Java Virtual Machine** or the **CLR** (.NET)
- **Virtual Machine Monitor** or **Hypervisor:** A software application that 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.

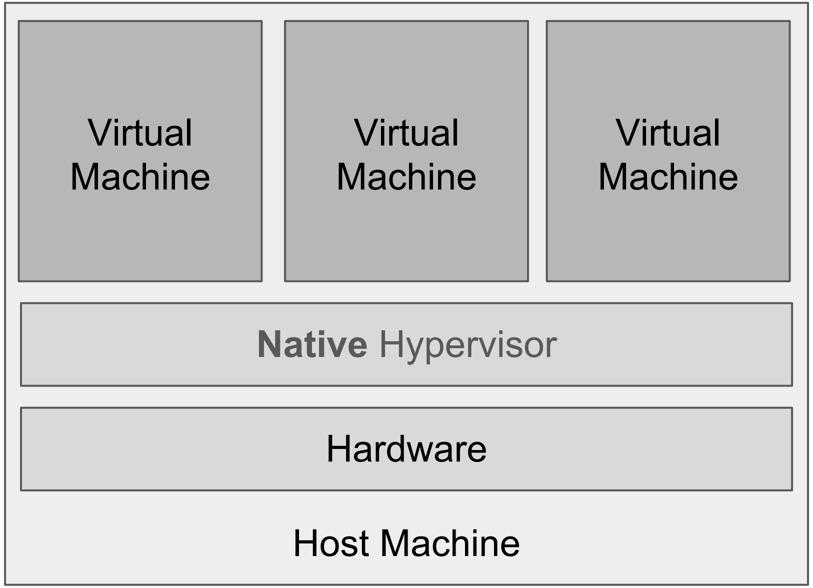


Virtual Machine Diagram

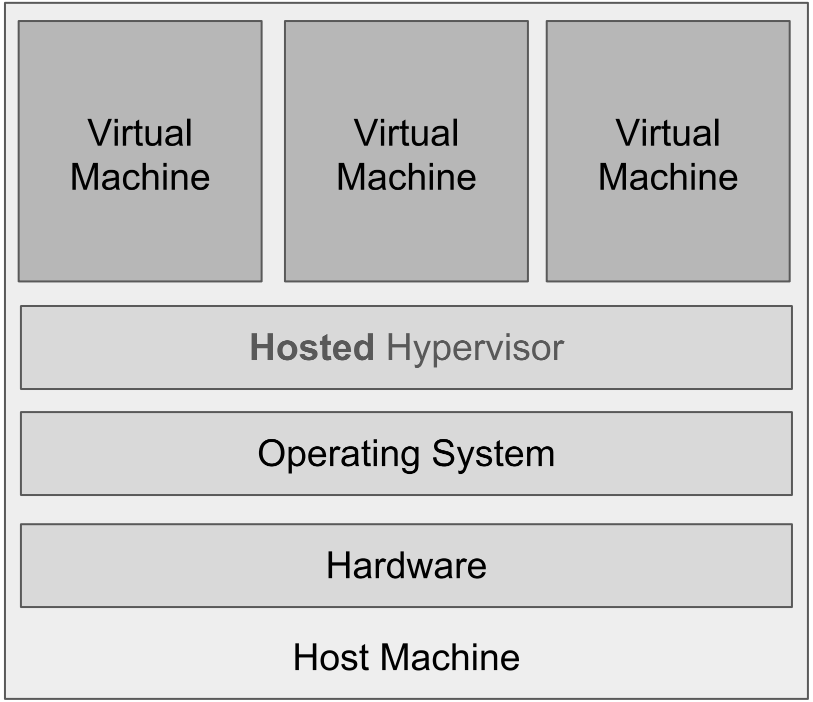
Virtual Machine Monitor (Hypervisor)

- The VMM is in charge of running the virtual machines.
- There are two main types of VMM:
  - **Type 1:** Native
  - **Type 2:** Hosted
- **Type 1: Native Hypervisors** run directly on the host machine, and share out resources (such as memory and devices) between guest machines.
  - e.g. XEN, Oracle VM Server
- **Type 2: Hosted Hypervisors** run as an application inside an operating system, and support virtual machines running as individual processes.
  - e.g. VirtualBox, Parallels Desktop, QEMU

Type 1 - Native



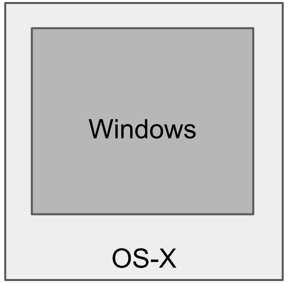
Type 2 - Hosted

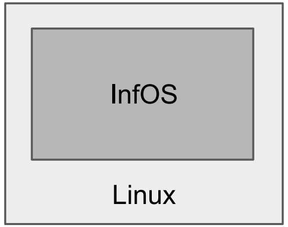


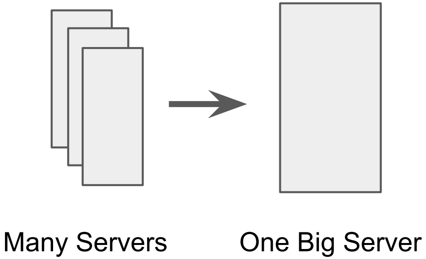
Hypervisor Types

Uses of Virtualisation

- **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)
  - Operating System/Hardware Design.
  - Kernel Debugging/Testing.
  - Prototyping new architectures/architectural features.
- **Commercial** (e.g. XEN/VMWare)
  - Data centre server consolidation.
  - High availability/Migration.







Types of Virtualisation

- **Software Emulation**
  - Maximum flexibility for virtualisation, but very slow to run (high overhead).
  - Each guest instruction is emulated (can use binary translation for speed-up)
- **Containers/Namespaces**
  - Isolate processes/groups of processes within a single operating system, e.g. Docker.
- **Full System** or **Hardware Virtualisation**
  - Isolate multiple operating systems from each other, within a single physical machine.
- **Same-architecture Virtualisation**
  - Guest Machine is the **same** architecture as the Host Machine, e.g. Intel x86 on Intel x86.
- **Cross-architecture Virtualisation**
  - Guest Machine has a **different** architecture than the Host Machine, e.g. ARM on Intel x86.
  - Must use software emulation to do this.

Popek and Goldberg Requirements for Virtualisation

Paper published in 1974 [1] that laid the foundations for hardware virtualisation, and formalised the requirements for an architecture to be "virtualisable".

Three main properties for a virtual machine:

1. **Efficiency**
   - The majority of guest instructions are executed directly on the host machine.
2. **Resource Control**
   - The virtual machine monitor must remain in control of all machine resources.
3. **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 be executed directly on the processor. 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 virtual machine should not be able to affect the host machine in any adverse way. The host machine should remain in control of all physical resources, sharing them out to guest machines.

Equivalence

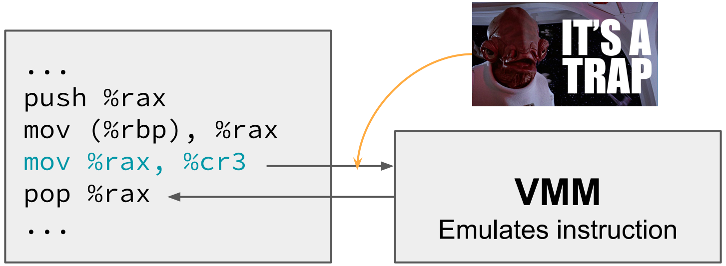
➔ "Any program K executing with a control program resident, with two possible exceptions, performs in a manner indistinguishable from the case when the control program did not exist and K had whatever freedom of access to privileged instructions that the programmer had intended."

A formal way of saying that the operating system running on a virtual machine should believe it is running on a physical machine, i.e. the behaviour of the virtual machine (from the guest OS' point of view) is identical to that of the corresponding physical machine.

The two exceptions mentioned are: **temporal latency** (some instruction sequences will take longer to run) and **resource availability** (physical machine resources are shared between virtual machines).

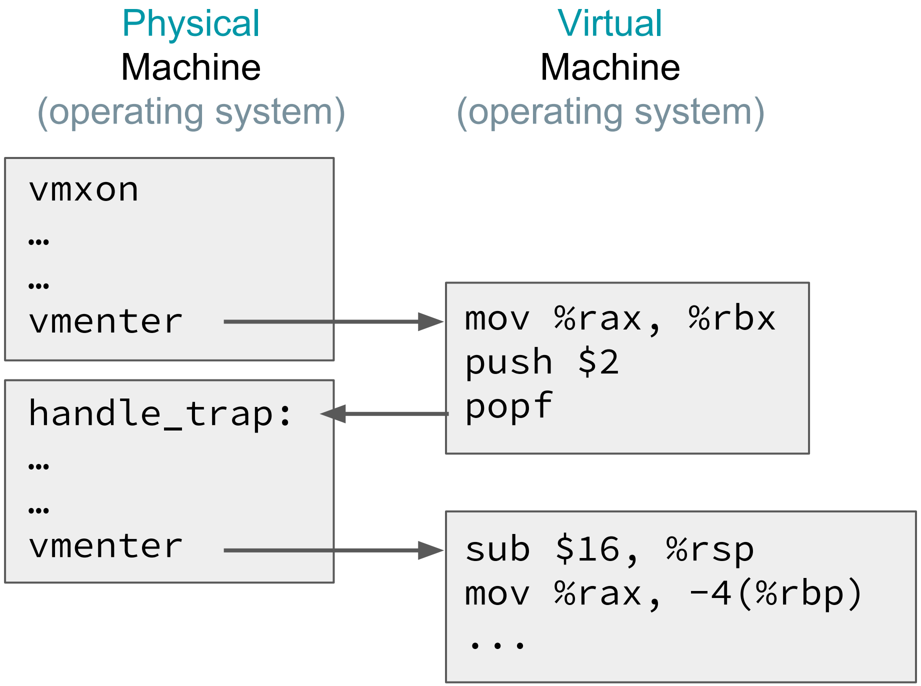
Methods of Virtualisation

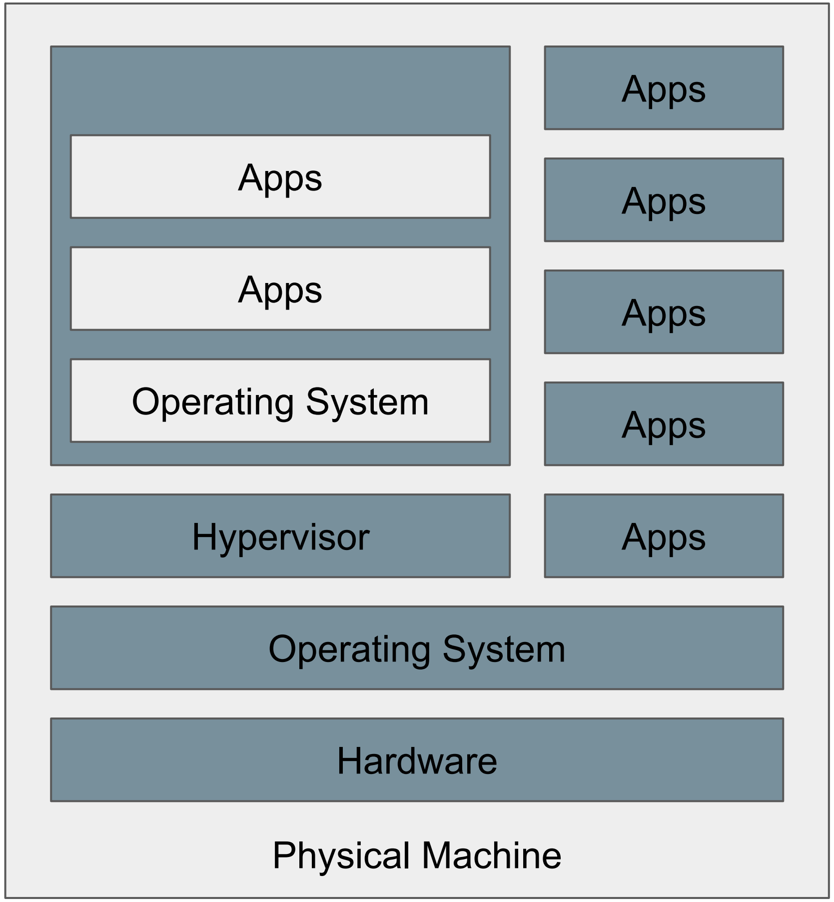
- **Full Software Emulation**
  - Not permitted by Popek and Goldberg because it violates the efficiency property.
    - Although, this no longer holds due to the advent of efficient binary translation.
  - Required for cross-architecture virtualisation, as guest instructions cannot execute natively on the host.
- **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 VMM.
  - VMM emulates these privileged operations.
  - Guest resumes execution after emulation.



Virtualising x86

- Originally x86 was not “classically” virtualisable.
    - Some privileged instructions did not “trap”, and so could not be emulated correctly.
- Interpretation is too slow (violates efficiency)
- Code Patching leaves traces of virtualisation (violates equivalency)
- Binary Translation is better, but still incurs overhead.
- Since 2005, x86 processors now support virtualisation in hardware.
    - Intel-VT
    - AMD-V
- This enables trap-and-emulate style virtualisation.
- Unmodified operating systems can run natively on host machines.





Virtualising x86 on Modern Hardware

Hardware Acceleration for Virtualisation

- Modern processors include hardware support for running virtual machines.
  - Intel VT-X and AMD-V for x86 processors.
  - ARM Virtualization Extensions for ARM processors.
- Hardware extensions allow all guest instructions (including system instructions) to run natively on the processor.
- This works by providing an isolated view of the processor to virtual machines.
- Operating Systems can then run directly on the processor, believing they are running on physical hardware.
- Certain privileged operations “trap” back to the hypervisor.

Virtual Machine Access to Resources

- Virtual Machines need to be given access to resources such as:
  - Memory
  - Storage
  - Networking
  - Graphics
- It is the responsibility of the VMM to share out these resources.
- Access to physical memory is managed by the VMM.
- For an unmodified operating system, expecting a “real” storage device (such as a hard disk), the VMM must provide an emulation of that device.
- Some devices may be passed straight through to the virtual machine, e.g. dedicated network cards.

Paravirtualisation

- Guest operating systems are aware they are being virtualised.
- They co-operate with the hypervisor to enable increased memory and device performance.
- They no longer “trap-and-emulate”, but instead request privileged operations directly from the hypervisor.
- They can co-operate with the hypervisor so that host memory can be more efficiently distributed.
- Instead of providing an emulated storage device, the hypervisor can provided a paravirtualised implementation.