Cloud Computing

Virtualization

Minchen Yu

SDS@CUHK-SZ

Fall 2024

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

SCHOOL OF DATA SCIENCE
數據科學學院

Outline

- Introduction and Concepts
- History
- How does virtualization work?
- State-of-the-art implementations
- Cloud infrastructures

Suppose that an IaaS provider owns a large datacenter and wants to provision cloud services for its users

Users demand…

Different machines with diverse computing capabilities, e.g., CPU, memory, networking, storage, etc.

Different OSs, e.g., CentOS, Ubuntu, Windows, etc.

Different softwares and libraries pre-installed, e.g., Python, Java, vim, git, etc.

Different networking requirement (topology, security, firewalls, etc.)

Can any of these be easily provisioned with "bare metal?"

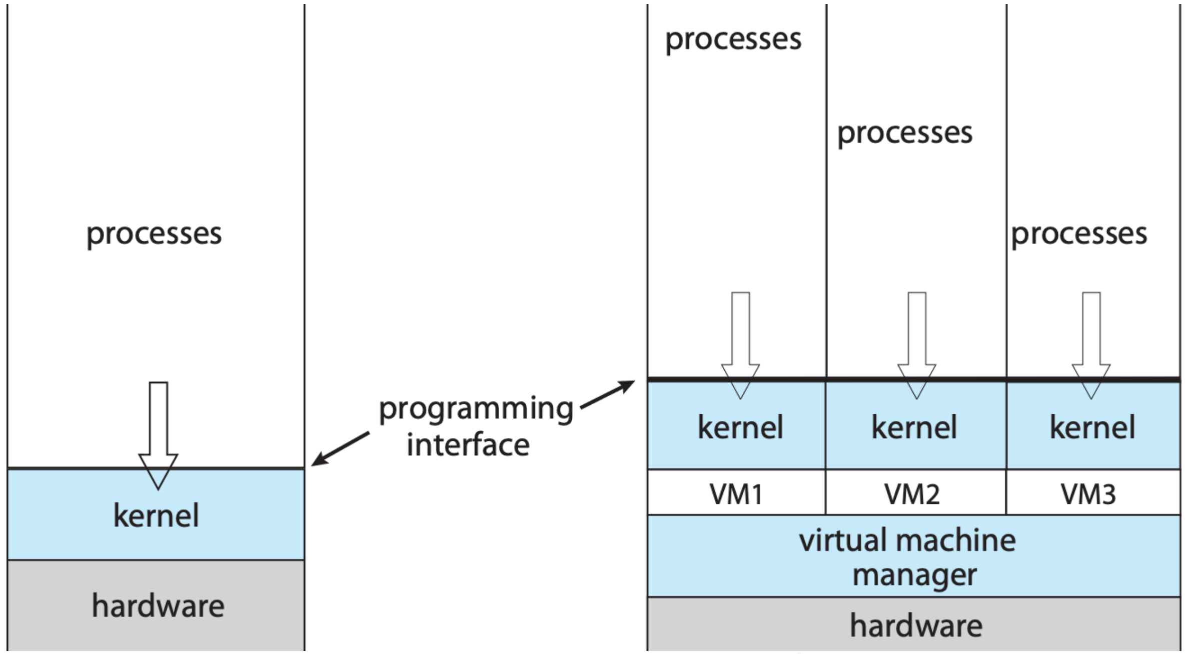
Virtualization is an enabling technology for IaaS Cloud

What is virtualization?

Virtualization is a broad term. It can be applied to all types of resources (CPU, memory, network, etc.)

Allows one computer to “look like” multiple computers, doing multiple jobs, by sharing the resources of a single machine across multiple environments.

Virtualization



'Nonvirtualized' system
A single OS controls all hardware platform resources

Virtualized system
It makes it possible to run multiple Virtual Machines on a single physical platform

Several components

Host

- the underlying hardware system

Virtual Machine Manger (VMM) or **hypervisor**

- creates and runs virtual machines by providing interface that is ***identical*** to the host (except in the case of paravirtualization)
- a VMM is essentially a simple operating system

Several components

A **virtual machine (VM)**

- a software-based implementation of some real (hardware-based) computer
- in its pure form, supports booting and execution of unmodified OSs and apps

Guest

- usually an operating system

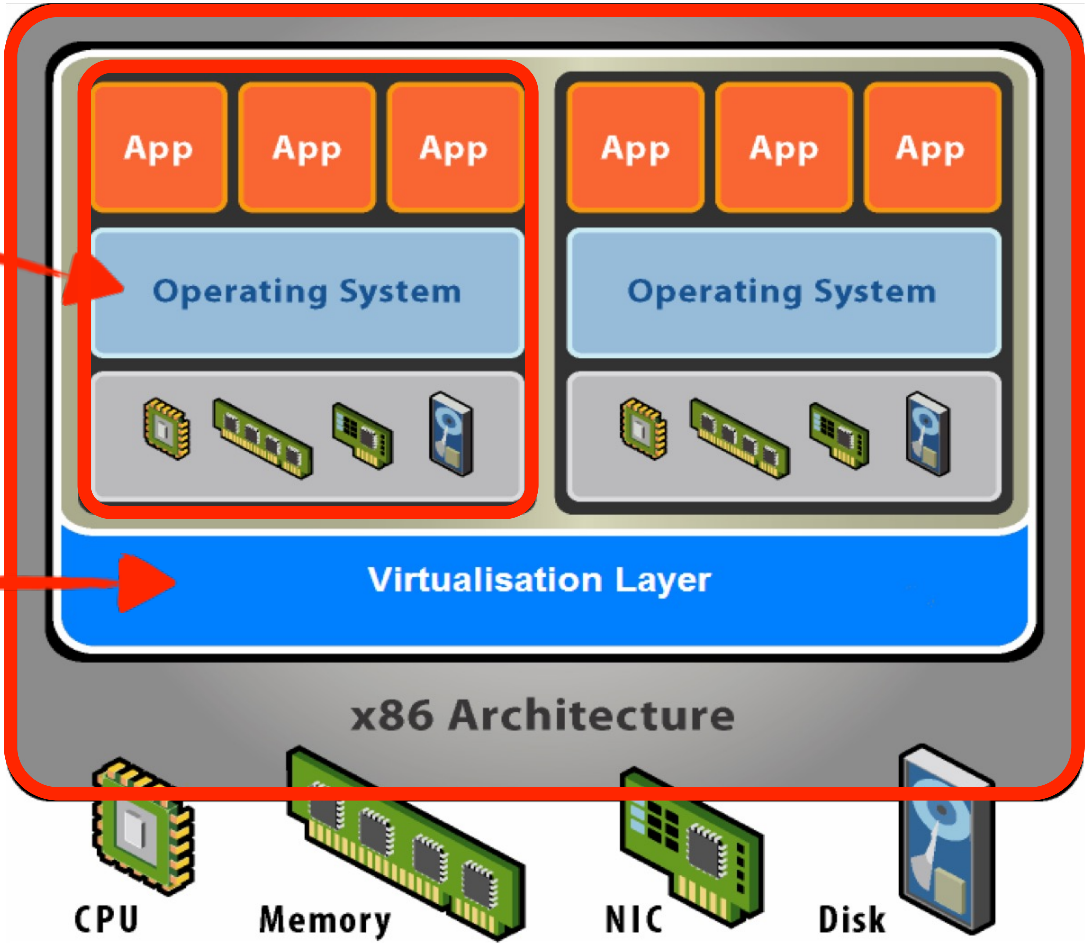
Several components

Guest OS

Guest VM

VMM

Host



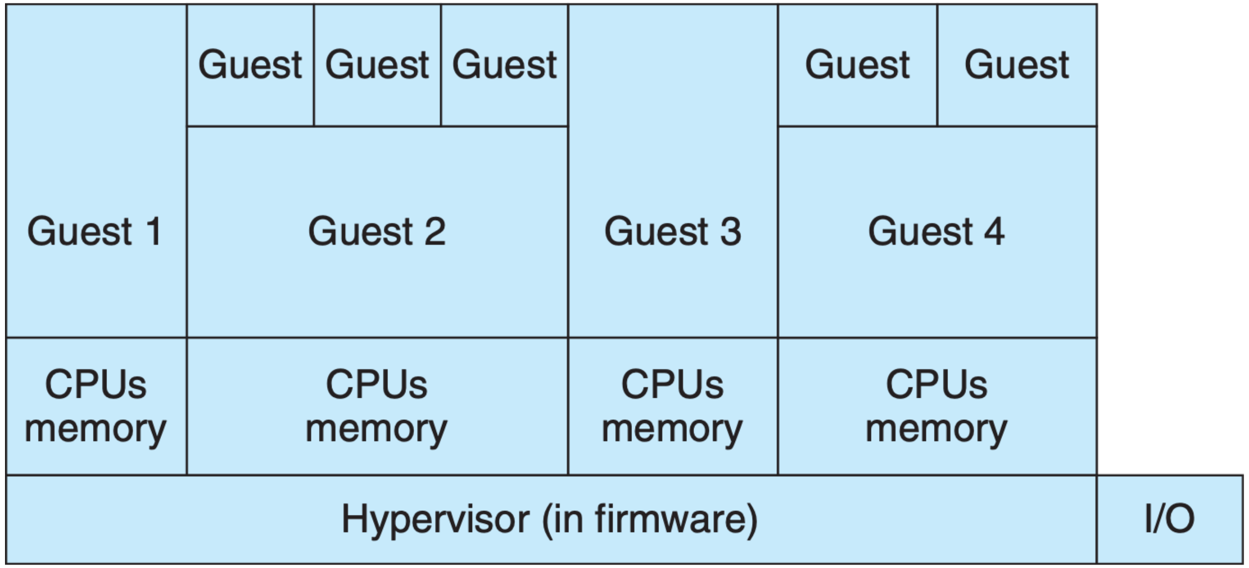
Implementation of VMMs

Varies greatly

Type-0 hypervisors

Hardware-based solutions that provides support for virtual machine creation and management via firmware

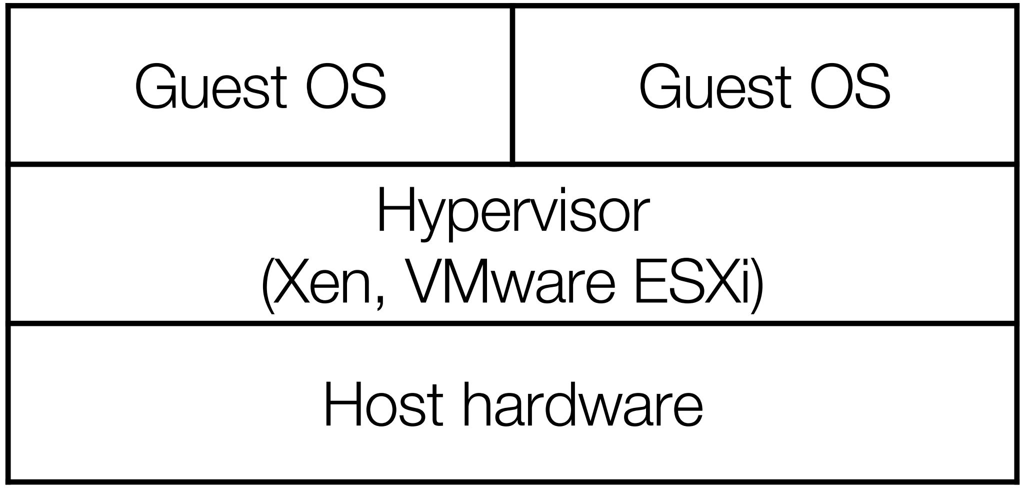
- commonly found in mainframes and large- to medium-sized servers, e.g., IBM LPARs and Oracle LDOMs



Type-1 hypervisors

The OS-like software providing a virtualization layer, directly on a clean x86-based system

- e.g., VMware ESXi, Citrix XenServer
- widely deployed in production clouds



Type-1 hypervisors

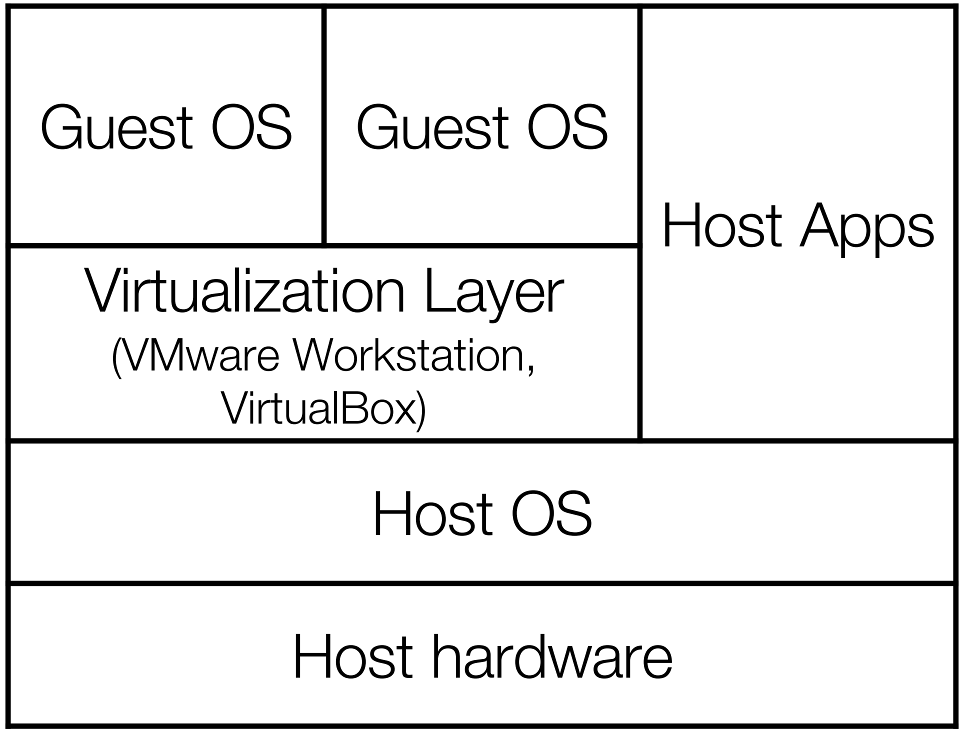
Also include general-purpose operating systems that provide standard functions as well as VMM functions

- e.g., Microsoft Windows Server w/ HyperV and RedHat Linux w/ KVM, Oracle Solaris
- typically less feature rich than dedicated type-1 hypervisors

Type-2 hypervisors

VMM is simply another process, run and managed by host

- even the host doesn't know they are a VMM running guests



Type-2 hypervisors

Indirect access to hardware through the host OS

- performance not good
- need administration privilege granted by the host OS
- usually for desktops and personal use, but not the production cloud environments

Other variations

Para-virtualization

- technique in which the guest OS is modified to work in cooperation with the VMM to optimize performance

Programming-environment virtualization

- VMMs do not virtualize real hardware but instead create an optimized virtual system, e.g., JVM and Microsoft.Net

Emulators

- allow applications written for one hardware environment to run in a different hardware environment, e.g., iOS emulator

Type-1 hypervisors and para-virtualization are the most popular choices on the cloud

History

Before there were datacenters

Early commercial computers were mainframes

IBM 704 (1954): $250K - millions

Issues with early mainframes

Early mainframe families had some disadvantages

- successive (or even competing) models were NOT architecturally compatible!
  - massive headache to update HW: gotta port software
- the systems were primarily ***batch-oriented***

In the mean time…

Project MAC (**M**ultiple **A**ccess **C**omputer) at MIT was kicking off

- responsible for developing Multics, a time-sharing OS
- invented many of the modern ideas behind time-sharing OS
- the computer was becoming a multiplexed tool for a community of users, rather than a batch tool for programmers

The mainframe companies, e.g., IBM, were about to be left in the dust!

Big blue's bold move

IBM bet the company on **System/360** [1964]

- first to clearly distinguish architecture and implementation
- its architecture was **virtualizable**

Unexpectedly, the **CP/CMS** system software is a hit [1968]

- CP: a "control program" that creates and manages virtual S/360 machines
- CMS: the "Cambridge monitor system" — a lightweight, single-user OS
  - run several different OSs *concurrently* on the same HW

Thus began the family tree of IBM mainframes… (type-0 hypervisors)

S/360 (1964-1970)

S/370 (1970-1988)

S/390 (1990-2000)

zSeries (2000-present)

Huge moneymaker for IBM, and many business still depend on these!

In the meantime…

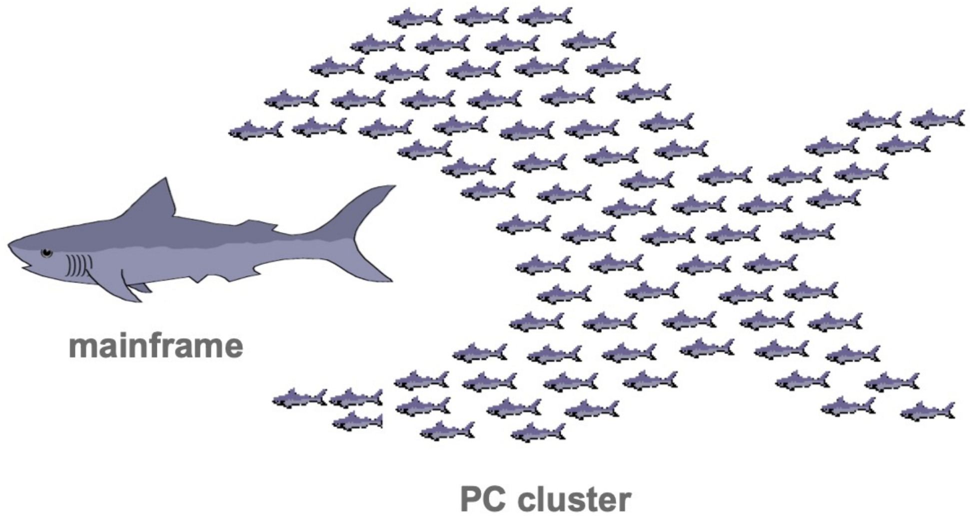
The PC revolution began

- much less powerful, but enjoy massive **economies of scale**

Cluster computing (1990s)

- build a cheap mainframe or supercomputer out of a cluster of commodity PCs

- use clever software to get fault tolerance



Mendel Rosenblum makes it BIG

VMware spun up from Stanford DISCO project in 1998

- brought CP/CMS-style virtualization to PC

Initial market was software developers

- often need to develop and test software on multiple OSs (windows, Linux, MacOS, …)
- can afford multiple PCs, or could dual-boot, but inconvenient
- instead, run multiple OSs simultaneously in separate VMs

Similar to mainframe VM motivation, but for opposite reason — too many computers now, not too few!

The real PC virtualization moneymaker

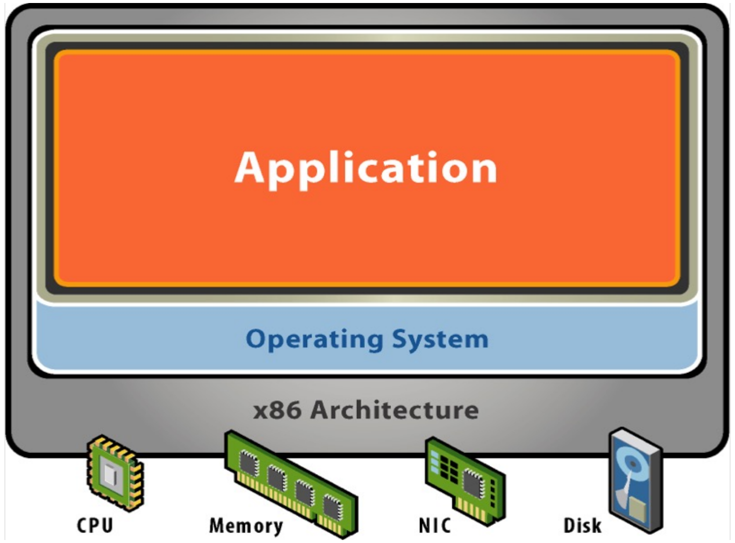
Enterprise consolidation

- big companies usually have their own clusters or datacenters
  - operate many services: mails, Webs, files, remote cycles
  - want to run **one service per machine** (best admin practice)
  - leads to **low utilization**!
- instead, run **one service per VM**

The old model

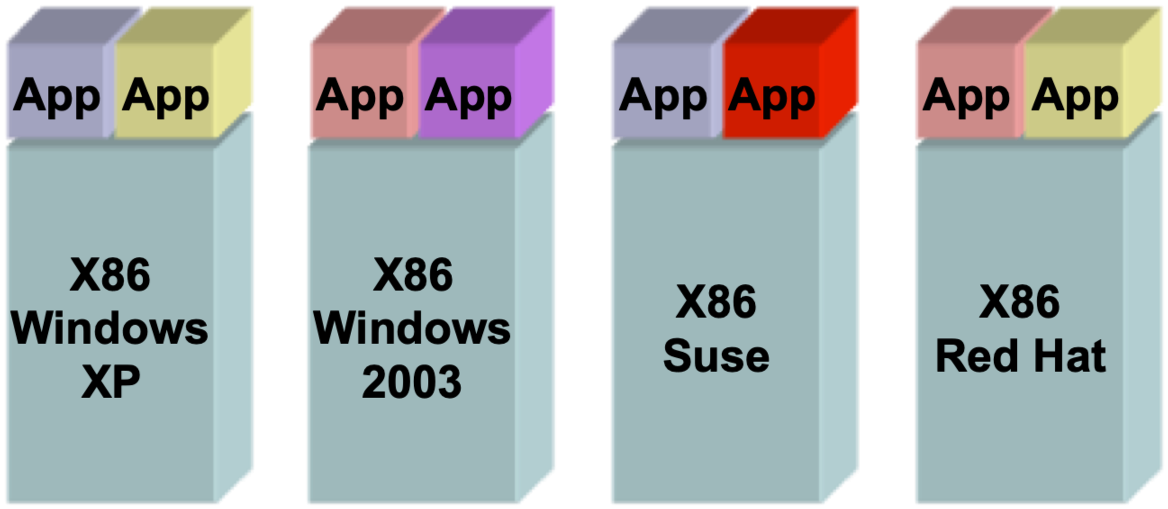
A server for every application

Software and hardware are tightly coupled



The old model

Big disadvantage: **low utilization**

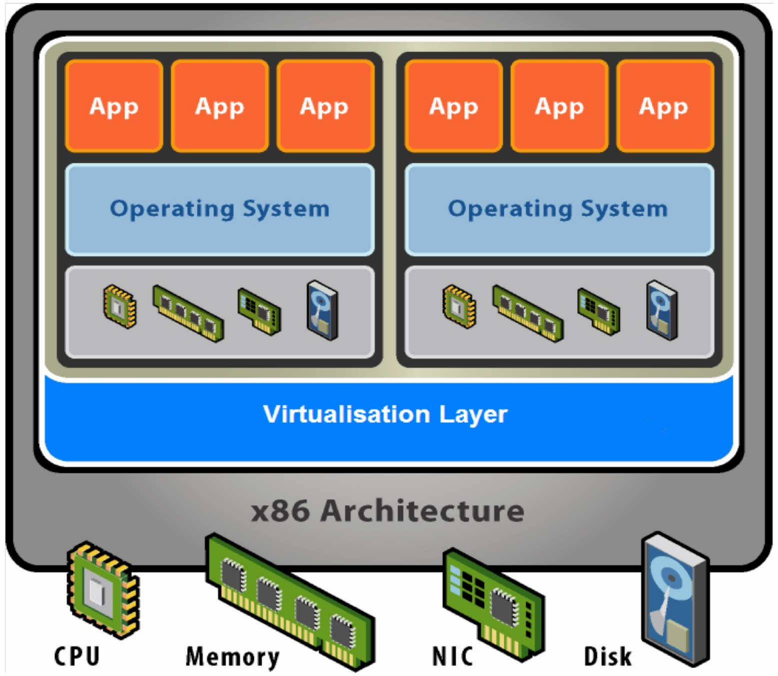


Util. 10% 12% 18% 20%

The new model: Consolidation

Physical resources are virtualized. OS and applications as a single unit by encapsulating them into **VMs**

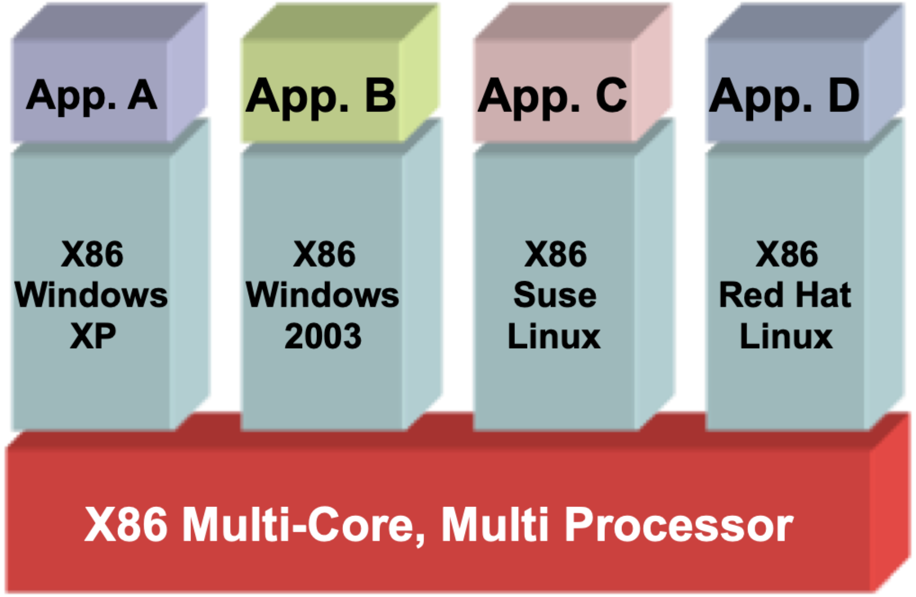
Separate applications and hardware



The new model: Consolidation

Big advantage: consolidation improves utilization

Individual Util. 10% 12% 18% 20%



Overall Util. 60%

Other benefits and features

Isolation: Host system protected from VMs; VMs protected from each other

Freeze, suspend, running VM

- they can move or copy somewhere else and resume

Great for OS research and better system development efficiency

Templating

Live migration

The forefront of virtualization

amazon web services™

Windows Azure™

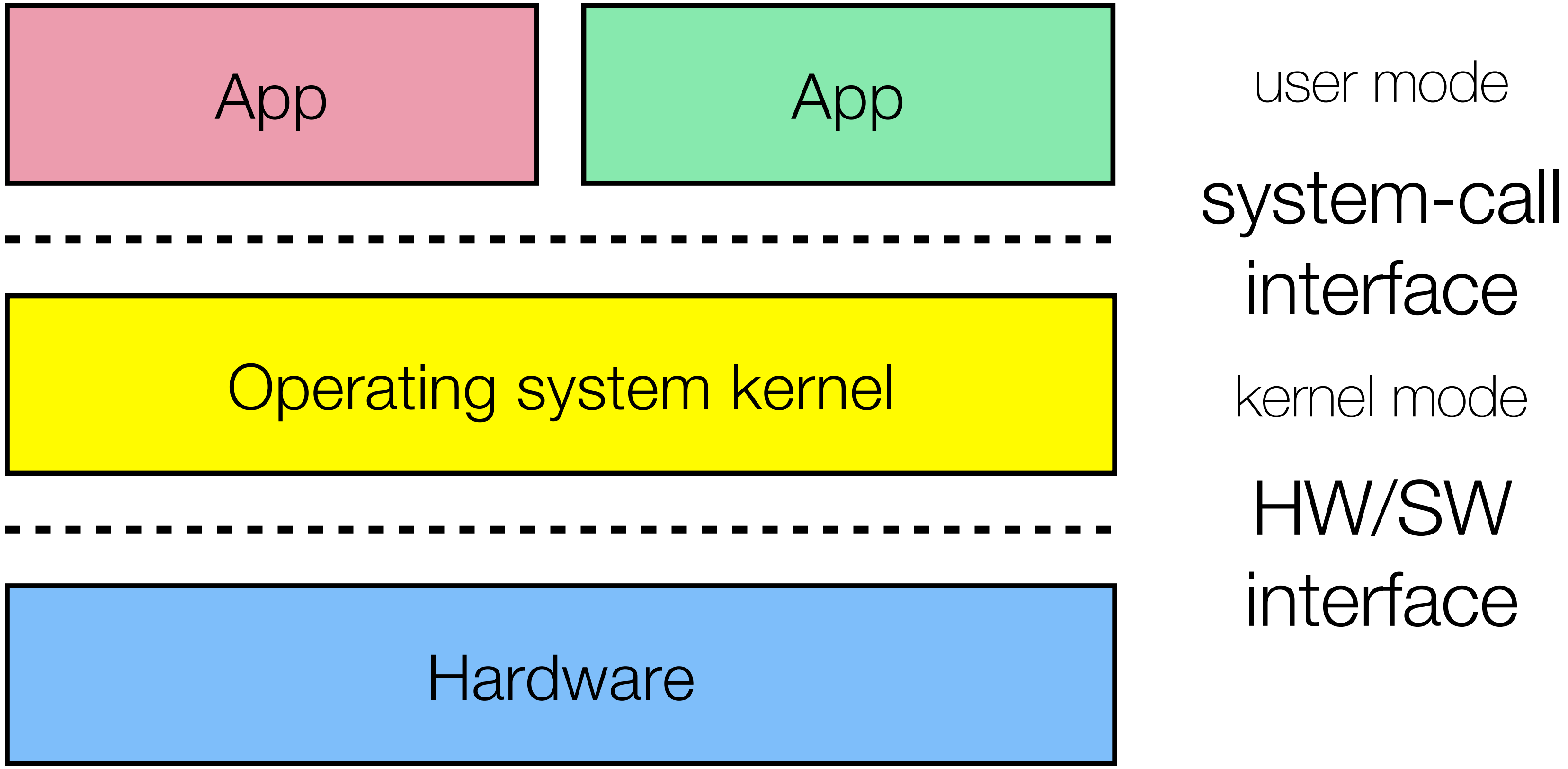
rackspace® the open cloud company

Google Cloud Platform

GOGRID

How does virtualization work?

How do regular machines work?



What is computer hardware?

Just a bag of devices…

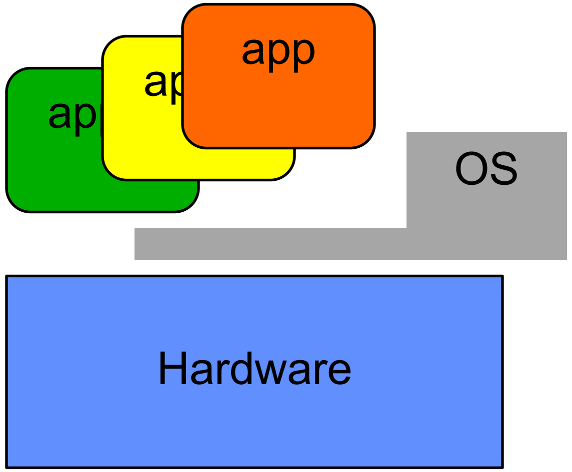
- **CPU:** instruction set, registers, interrupts, privilege modes
- **Memory:** physical memory words accessible via load/store
    - MMU provides paging/segmentation, and virtual memory
- **I/O:** disks, NICs, etc., controlled by programmed I/O or DMA
    - events delivered to software via polling or interrupts
- **Other devices:** graphic cards, clocks, USB controllers, etc.

What is an OS?

Special layer of software that provides application software access to hardware resources

- runs like any other program, but in a **privileged** (kernel) CPU mode
    - protects itself from user programs
    - can interact with HW devices using “sensitive” instructions

What is an OS?



gives apps a high-level programming interface (system-call interface)

OS implements this interface using low-level HW devices

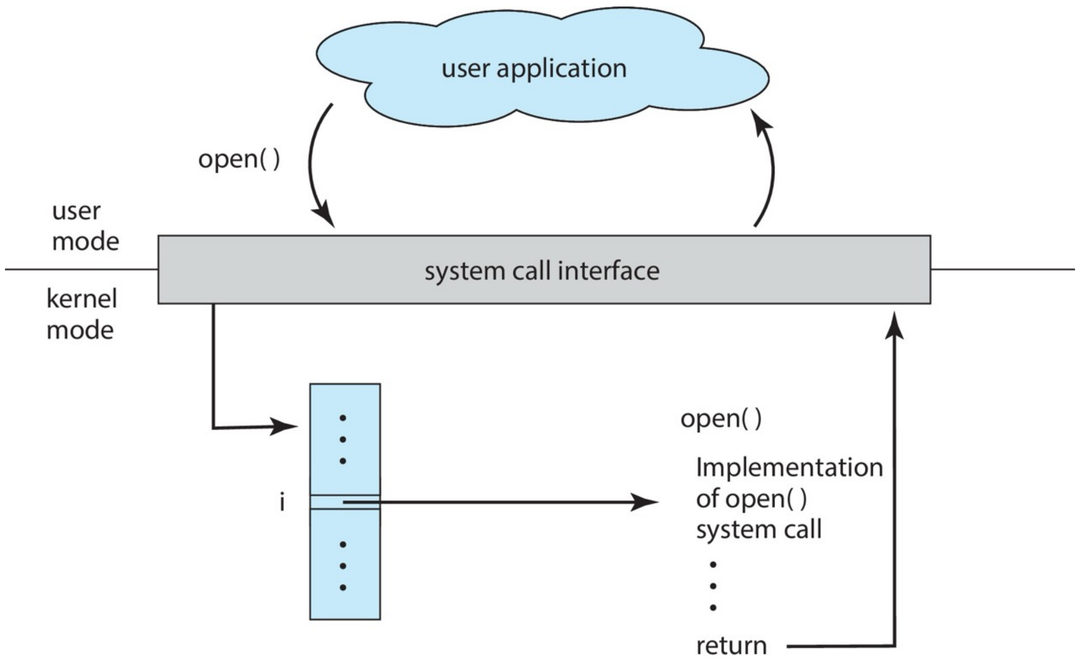
- file open/read/write vs disk block read/write

issues instructions to control HW on behalf of programs

What is an application?

A program that relies on the system-call interface

- While executing, the CPU runs in **unprivileged** (user) mode
- a special instruction (`int` on x86) lets a program call into OS



User program calls into OS

```c
#include <unistd.h>

int main(int argc, char *argv[])
{
    write(1, "Hello World\n", 12);
    _exit(0);
}
```

```
_start:
    movl $4, %eax    ; use the write syscall
    movl $1, %ebx    ; write to stdout
    movl $msg, %ecx  ; use string "Hello World"
    movl $12, %edx   ; write 12 characters
    int $0x80        ; make syscall

    movl $1, %eax    ; use the _exit syscall
    movl $0, %ebx    ; error code 0
    int $0x80        ; make syscall
```

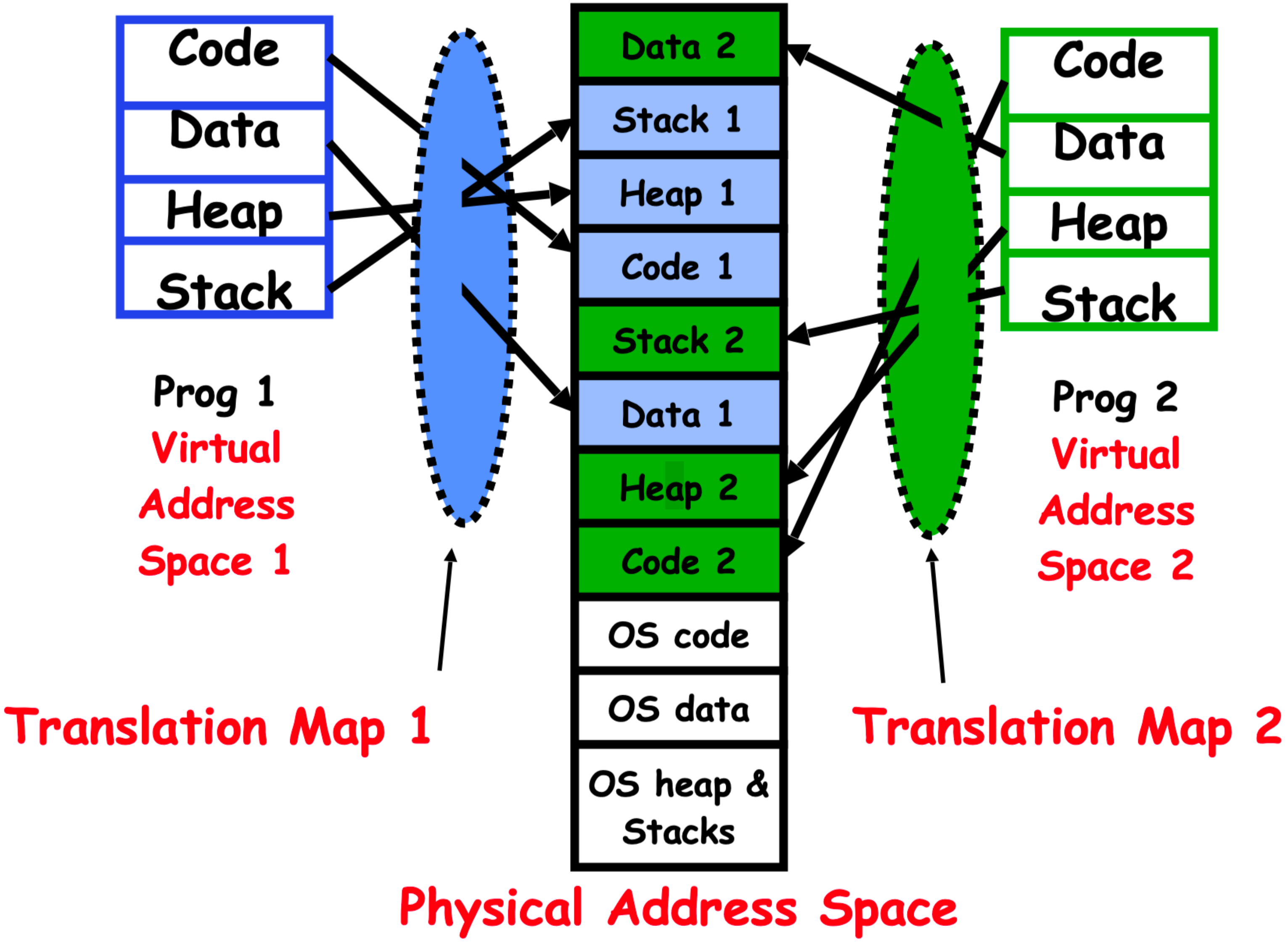
Traps to kernel

Traps to kernel

What is an application?

A program that relies on the system call interface

- While executing, the CPU runs in **unprivileged** (user) mode
- a special instruction (`int` on x86) lets a program call into OS
- OS provides a program with the illusion of its own memory
  - MMU hardware lets the OS define the "**virtual address space**" of the program



Credit: Prof. John Kubiatowicz's slides for CS162, Spring 2015, UC Berkeley

Is this safe?

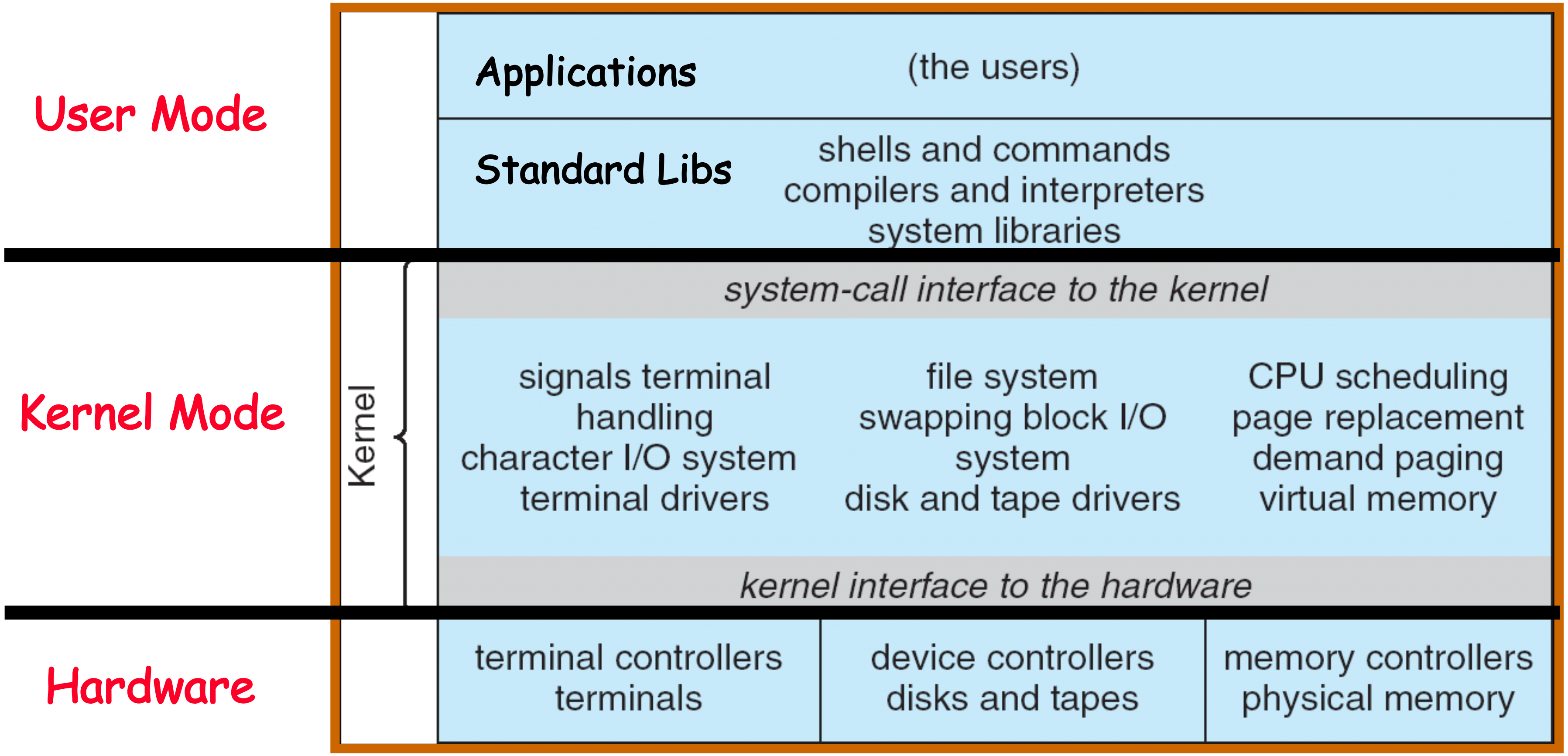
Most instructions run **directly** on the CPU (fast)

- but sensitive instructions cause the CPU to throw an exception to the OS

Address spaces prevent program from stomping on OS memory, each other

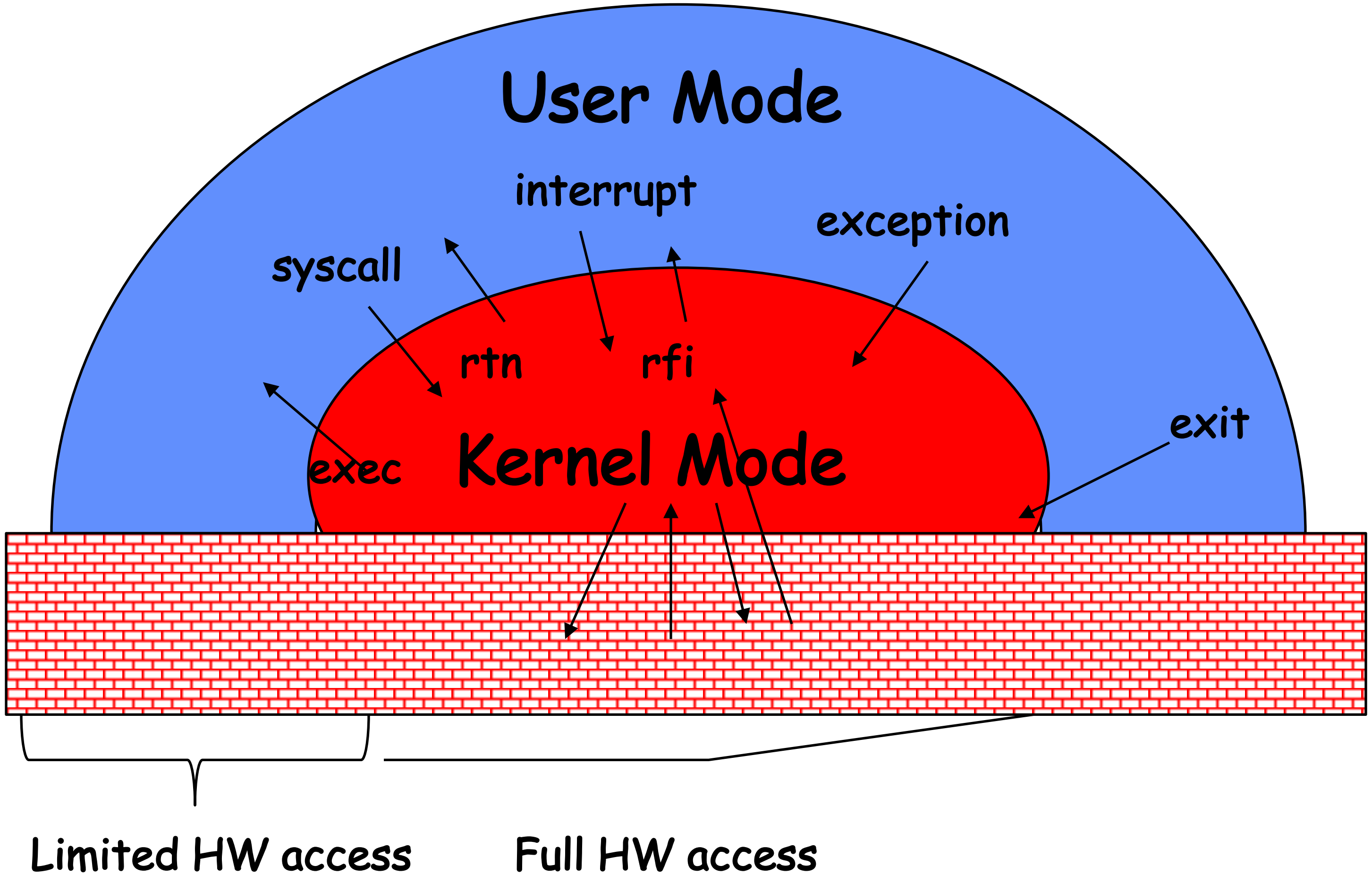
It's as though each program runs in its own, private machine (the "process")

Putting them together



Credit: Prof. John Kubiatowicz's slides for CS162, Spring 2015, UC Berkeley

User/kernel (privileged) mode

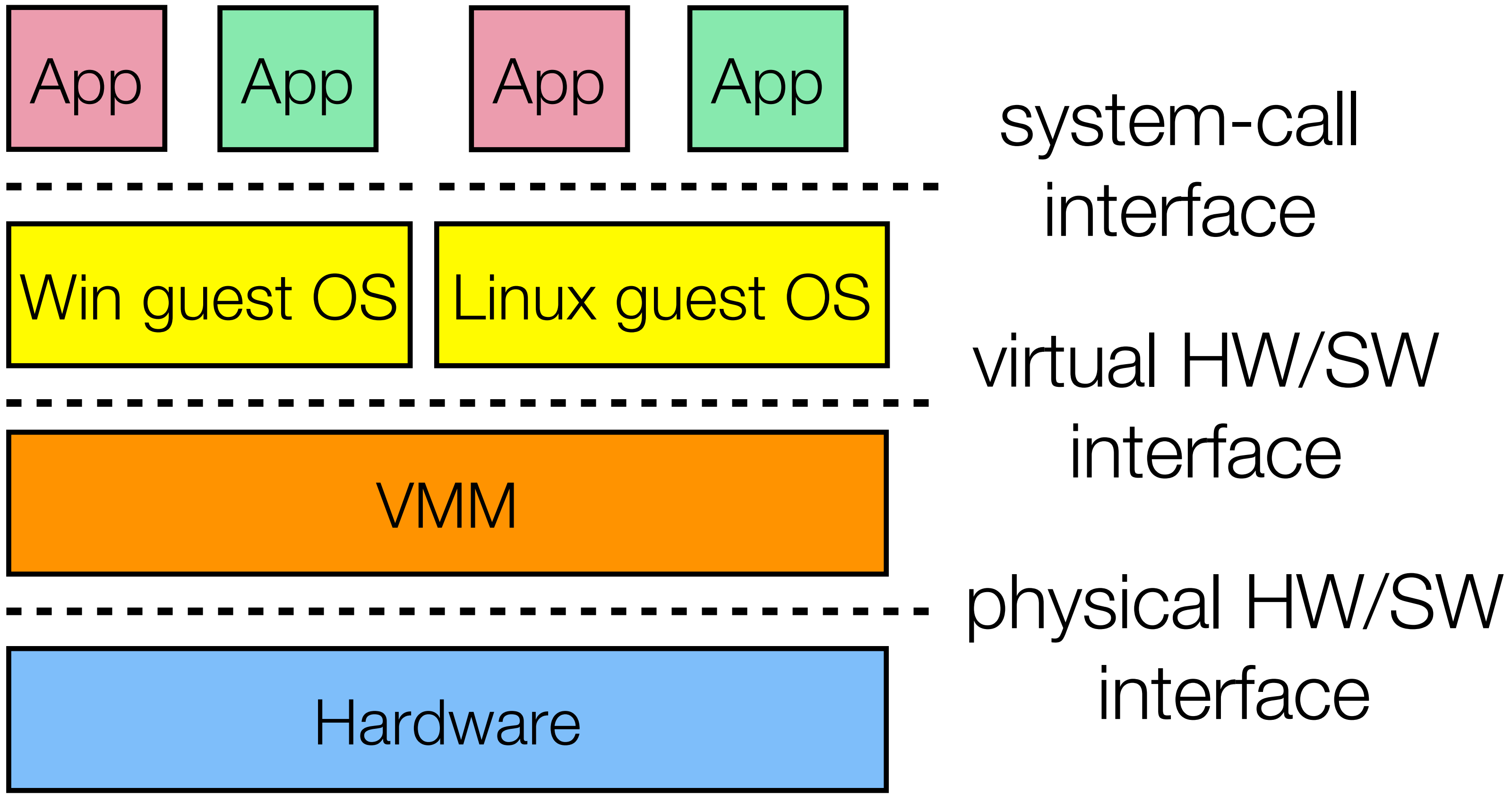


Credit: Prof. John Kubiatowicz's slides for CS162, Spring 2015, UC Berkeley

How does virtualization work?

A goofy idea…

What if we run Windows as a **user-level** program?



It almost works, but…

What happens when Windows issues a sensitive instruction in kernel mode?

What (virtual) hardware devices should Windows use?

How do we prevent apps running on Windows from hurting Windows?

- or apps from hurting the VMM…
- or Windows from hurting Linux… or the VMM…

Trap-and-emulate

Trap and emulate

Guest VM needs two modes — **virtual user mode** and **virtual kernel mode**

- both of which run in real user mode, as it is not safe to let guest kernel run in kernel mode
- only VMM runs in kernel mode

Actions in guest OS that cause switch to kernel mode must cause the VM switch to virtual kernel mode

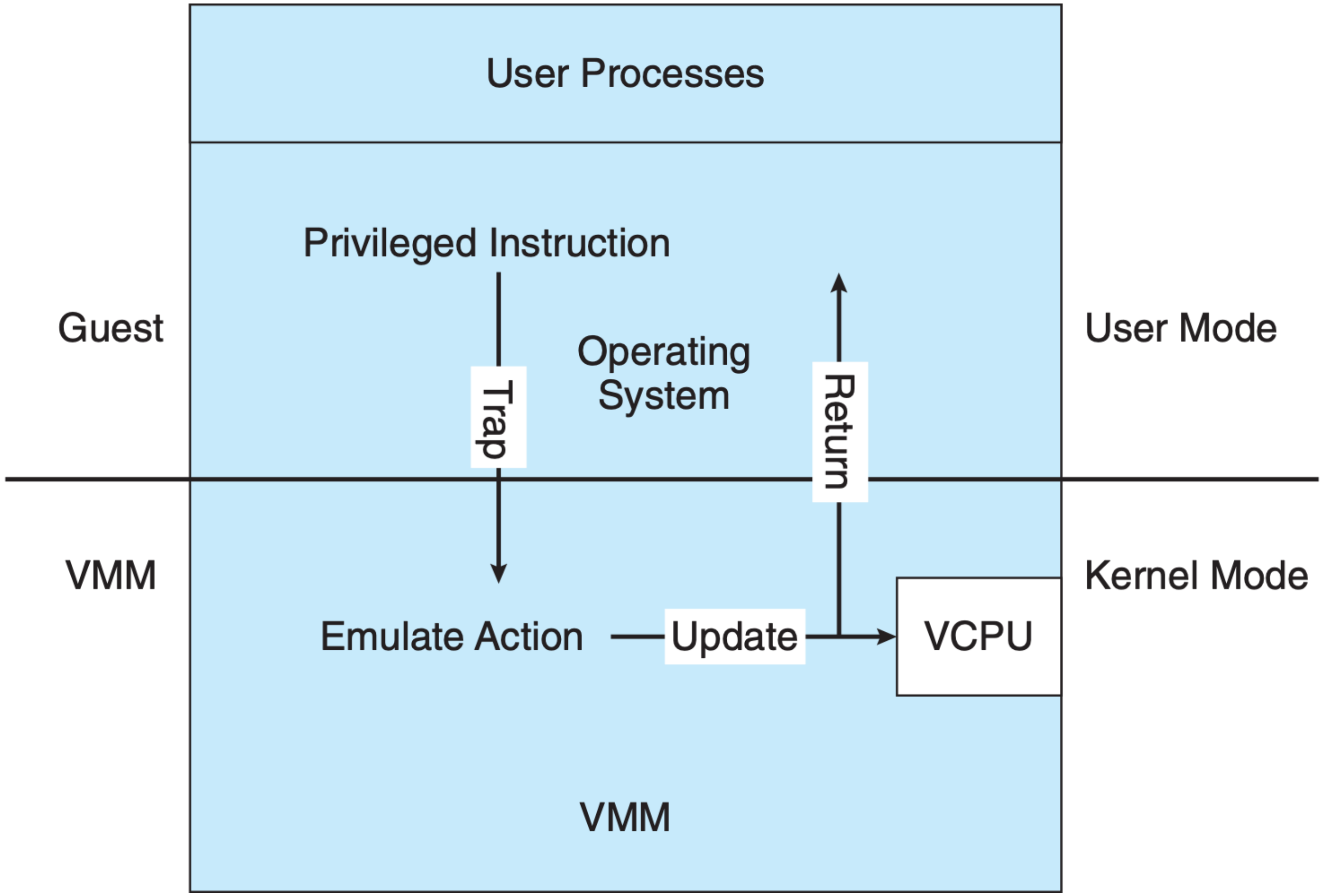
- but how?

Trap and emulate

How does switch from virtual user mode to virtual kernel mode occur?

- **Trap**: guest attempting a privileged instruction in user mode causes an error -> host traps to kernel mode
- **Emulate**: VMM gains control, analyzes the error, emulates the effect of instruction attempted by guest
  - VMM provides a virtual HW/SW interface to guest
- **Return**: VMM returns control to guest in user mode

Trap and emulate



Most virtualization products use this at least in part

Correctness requirement

Two classes of instructions

Privileged instructions

- those that trap when CPU is in user-mode, i.e., requiring the kernel mode

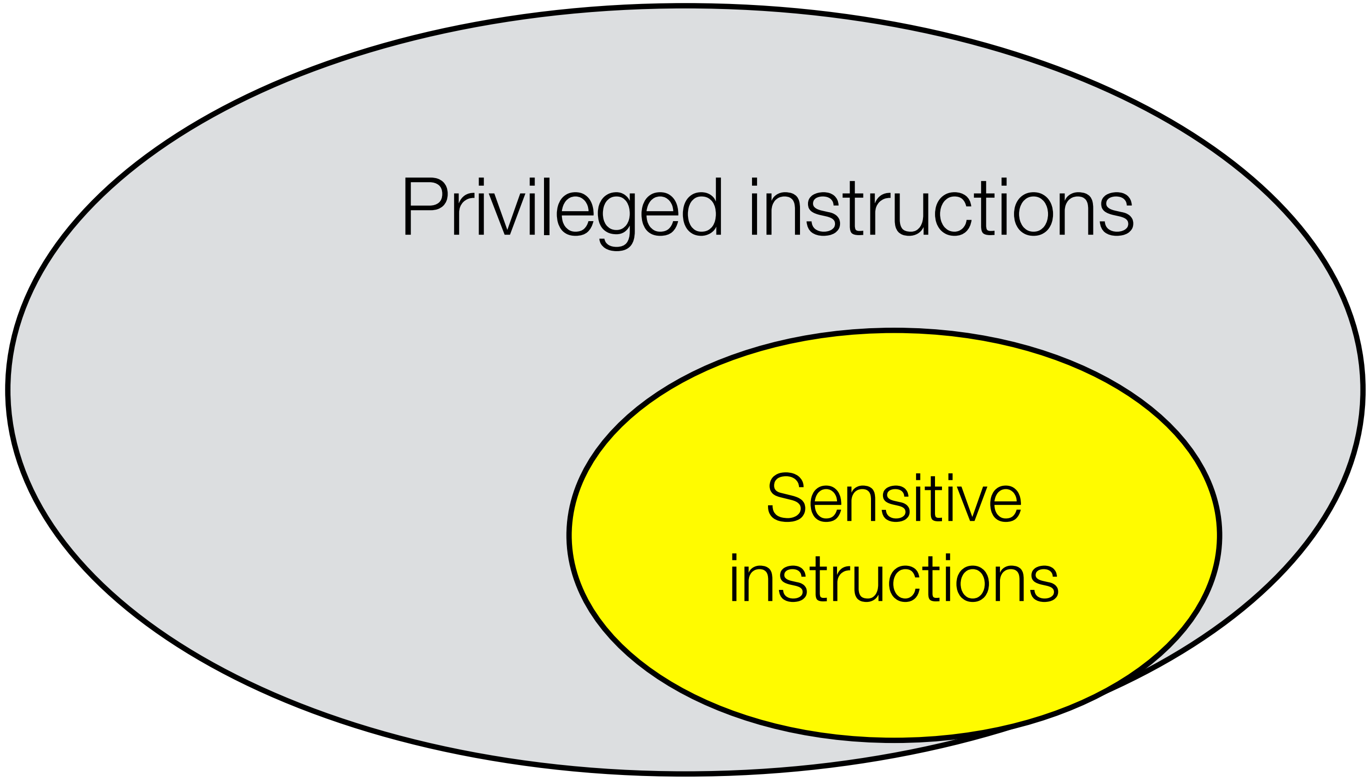
Sensitive instructions

- those that modify (virtual) HW configuration or resources, and those whose behaviors depend on (virtual) HW configuration
- e.g., read, write, CPU register setting

Emulation is only needed for sensitive instructions

Popek & Goldberg (1974)

A VMM can be constructed efficiently and safely if the set of sensitive instructions is a **subset** of privileged instructions.



How about the performance?

Non-sensitive instructions

Almost no overhead

- they execute directly on CPU
- CPU-bound code execute at the same speed on a VM as on a bare metal
  - e.g., scientific simulations

Sensitive instructions

Significant performance hit!

- they raise a trap and must be vectored to and emulated by VMM
  - each instruction could require tens of native instructions to emulate
- I/O or system-call intensive applications get hit hard

Trap-and-emulate not always works

The Intel architecture did not meet Popek & Goldberg's requirement

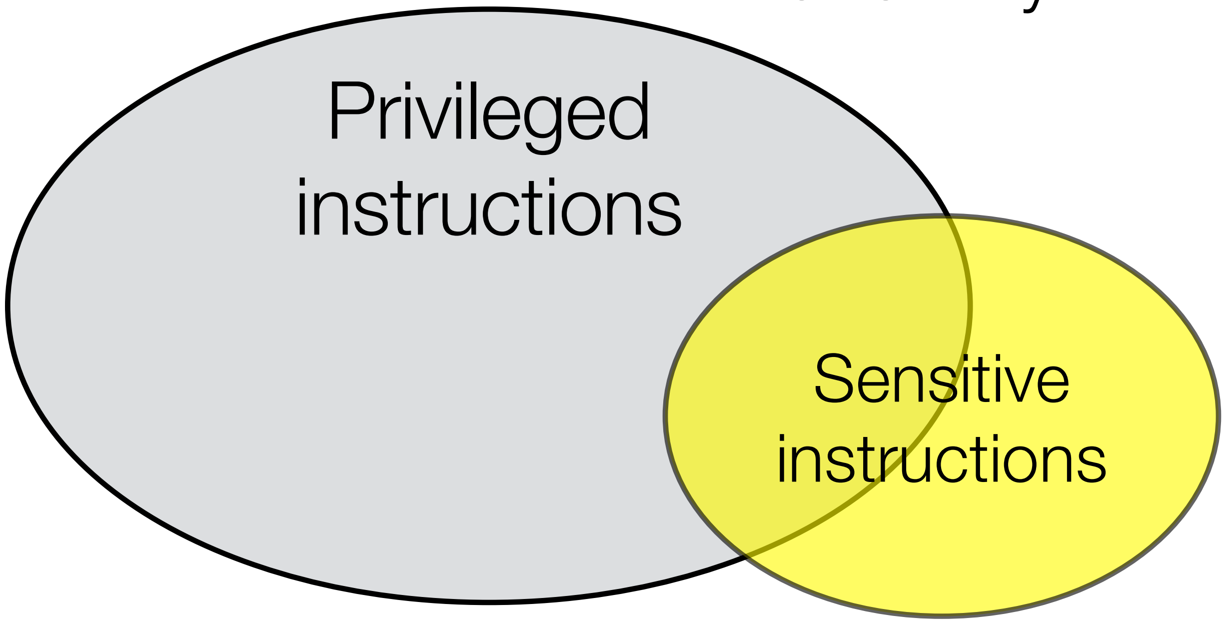
- consider Intel x86 **popf** instruction, which loads CPU flags register from contents of the stack
  - if CPU in kernel mode -> all flags replaced
  - if CPU in user mode -> only some flags replaced, without trapping to kernel mode
  - **popf** is sensitive but not privileged, i.e., not virtualizable using trap-and-emulate!

A hard problem

Some CPUs don’t have a clean separation between privileged and non-privileged instructions

- **special instructions** not virtualizable

That’s why VMware made $$



Intel CPUs considered not virtualizable until 1998

Three solutions

Full virtualization

- Emulate + binary translation: this is rocket science and what VMware did!

Para-virtualization

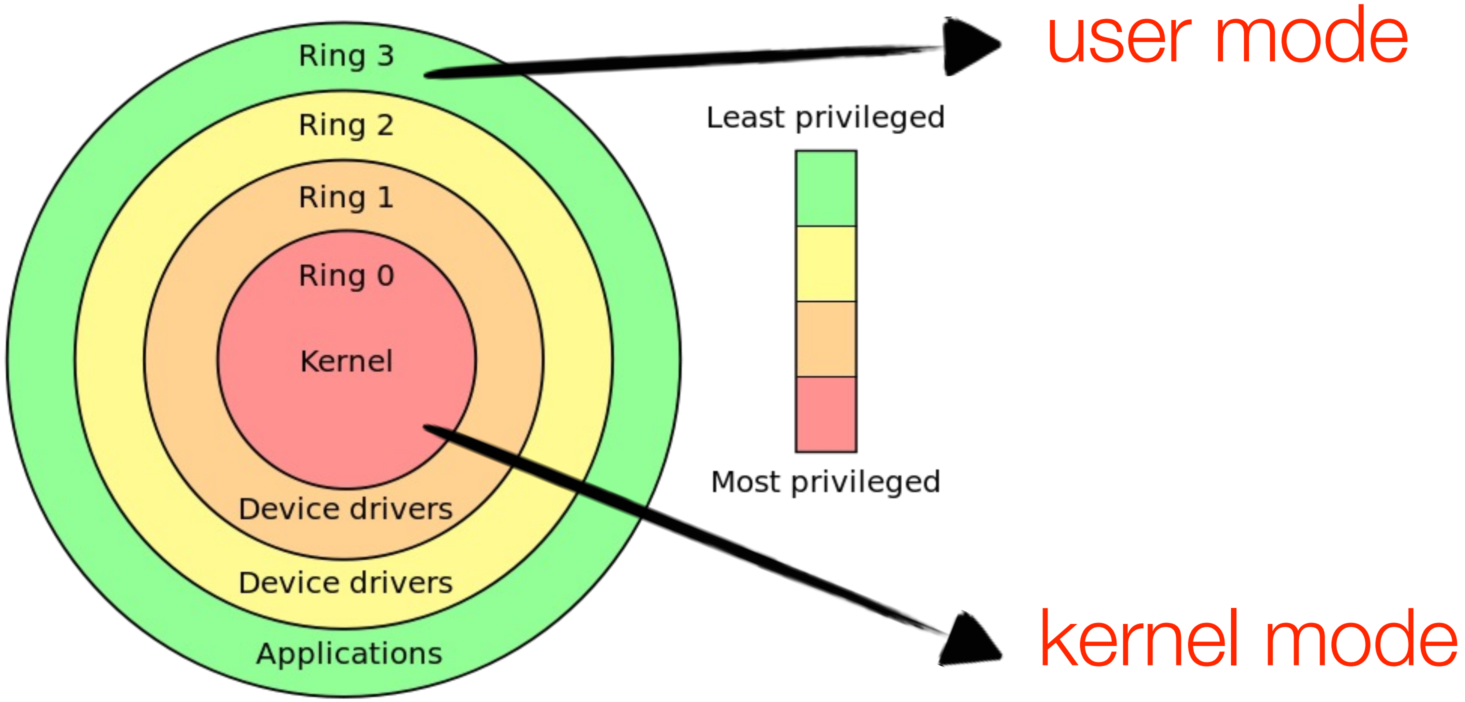
- modify the guest OS to avoid non-virtualizable instructions

Hardware-assisted virtualization

- fix the CPUs

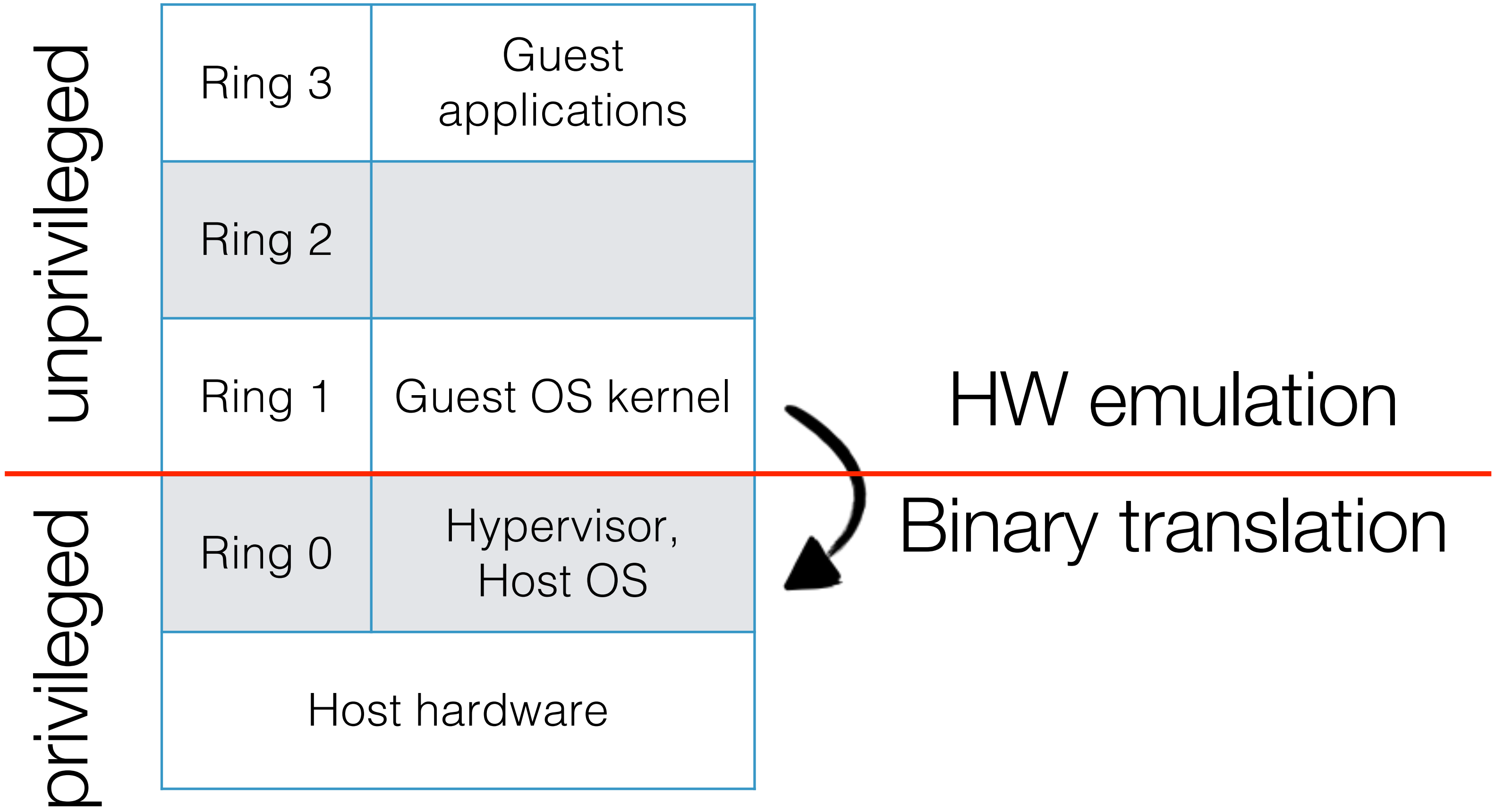
x86 protection rings

Enforced in hardware in Intel x86 architectures



Source: Wikipedia

Full virtualization



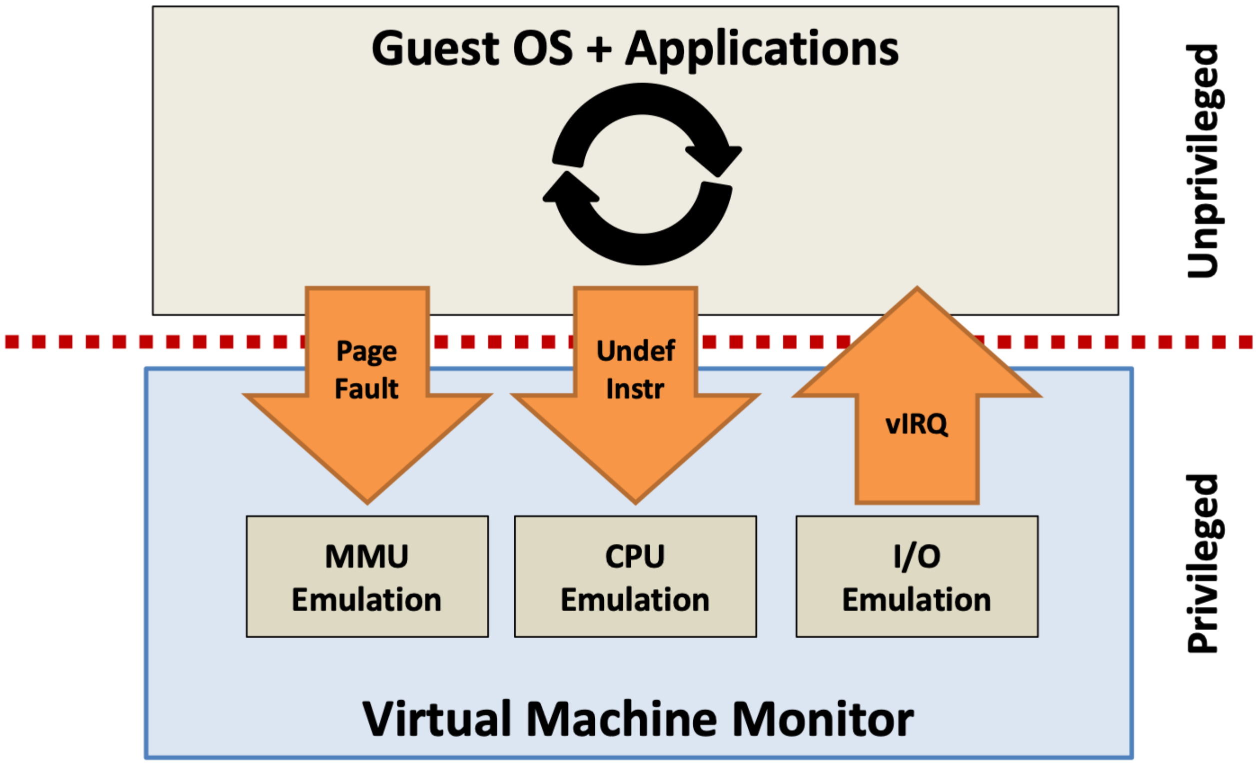
Full virtualization

Key technique: **binary translation**

- basics are simple, but implementation rather complex
- if guest VCPU is in user mode, guest can run instructions natively
- if guest VCPU is in (virtual) kernel mode, hypervisor examines every instruction guest is about to execute
    - non-special instructions run natively
    - special instructions translated into new set of instructions that perform equivalent task in emulated hardware

Full virtualization

Hardware is emulated by the hypervisor



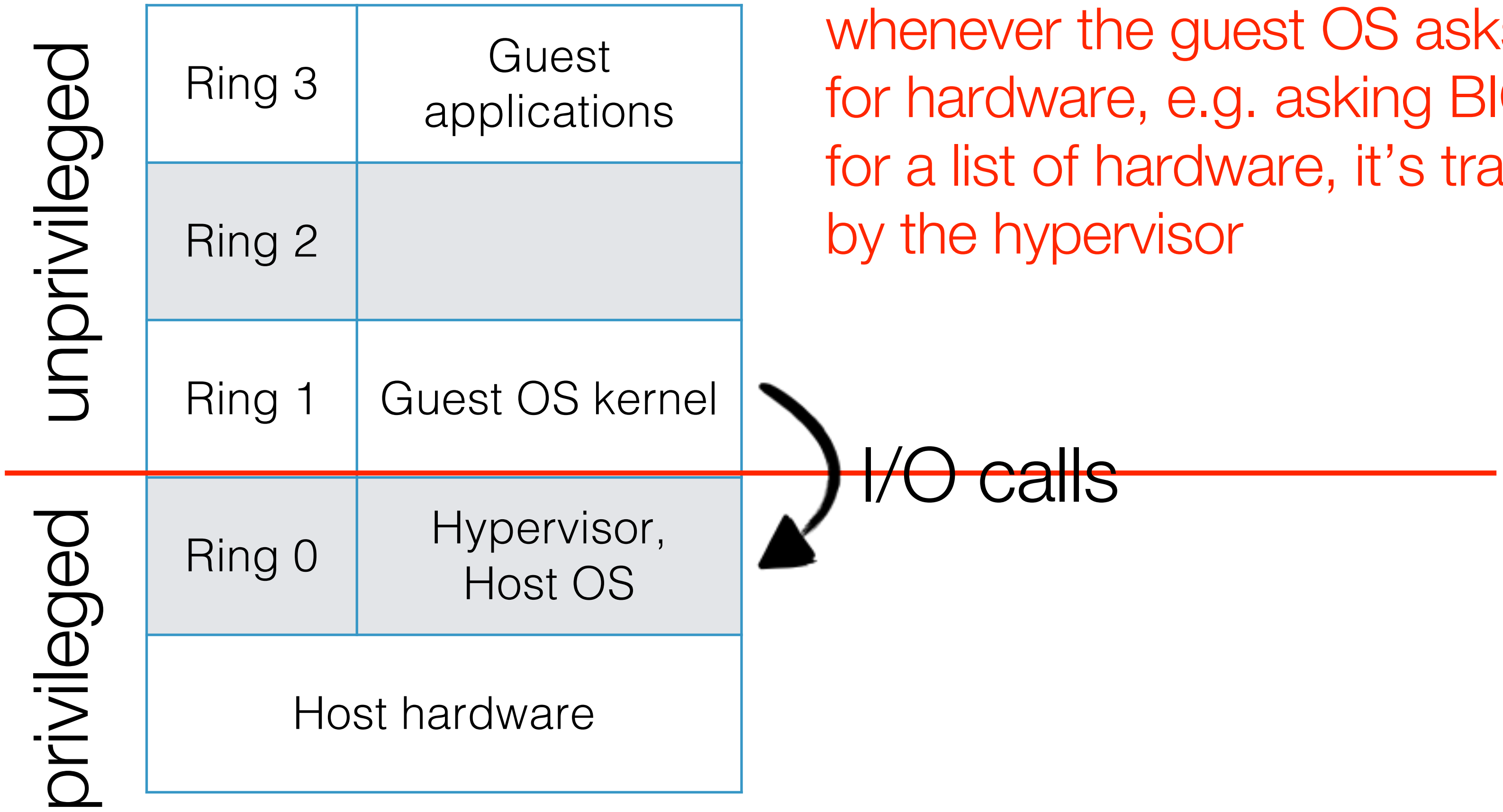
Full virtualization

The hypervisor presents a complete set of emulated hardware to the VM's guest operating system

- e.g., Microsoft Virtual Server 2005 emulates an Intel 21140 NIC card and Intel 440BX chipset.
- Regardless of the actual physical hardware on the host system, the emulated hardware remains the same

Full virtualization

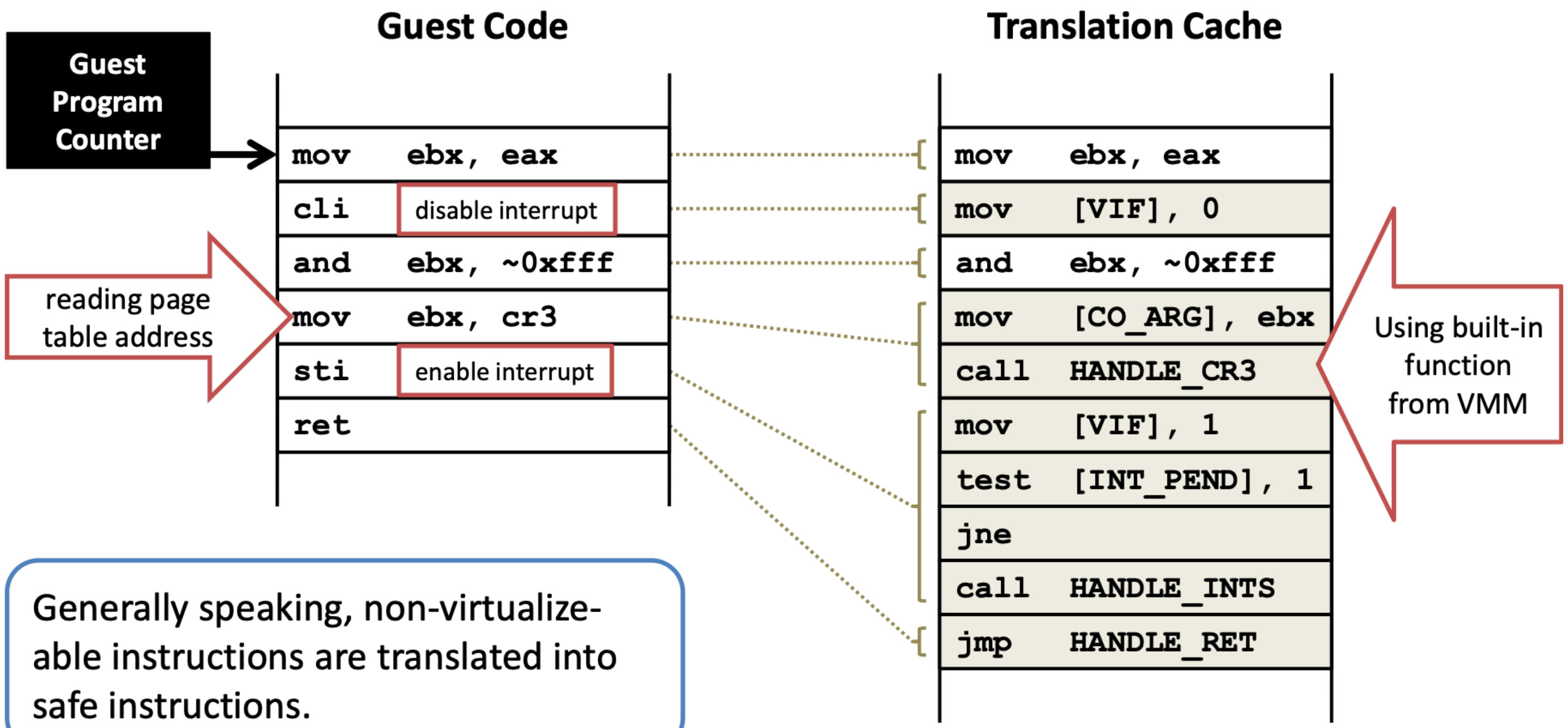
Binary translation – step 1: trapping I/O calls



whenever the guest OS asks for hardware, e.g. asking BIOS for a list of hardware, it's trapped by the hypervisor

Full virtualization

Binary translation – step 2: emulate/translate



Generally speaking, non-virtualize-able instructions are translated into safe instructions.

Full virtualization

The guest OS is tricked to think that it's running privileged code in Ring 0

- it's actually running in Ring 1 of the host with the hypervisor emulating the hardware and trapping privileged code

Unprivileged instructions are directly executed on CPU

Full virtualization

Advantages:

- keeps the guest OS unmodified
- prevents an unstable VMs from impacting system performance
- VM portability

Disadvantages:

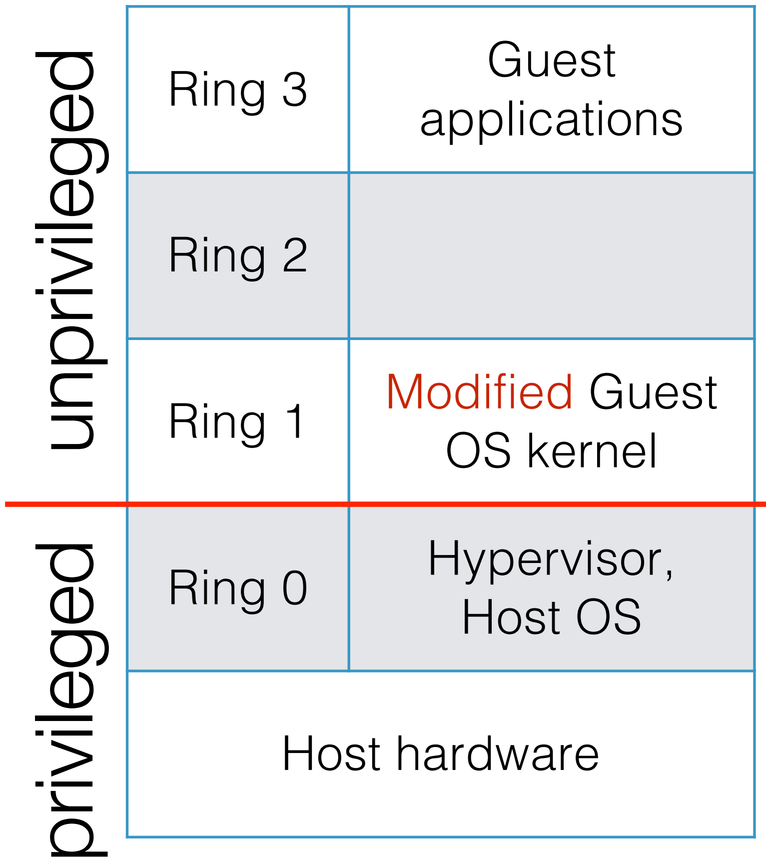
- Performance is not good without optimization
- possible solution: caching the translation of special instructions to avoid translating them again in the future

Para-virtualization

Developed to overcome the performance penalty of full virtualization with hardware emulation

- “Para” means “besides,” “with,” or “alongside.”
- the most well-known implementation is Xen

Para-virtualization



Include virtualization APIs and drivers

No binary translation

Para-virtualization

Can be done in two ways:

- A recompiled OS kernel: easy for Linux, Windows doesn’t support
- Para-virtualization drivers for some hardware, e.g. GPU, NIC

Para-virtualization

Guest OS is aware that it runs in a virtualized environment.

- it talks to the hypervisor through specialized APIs to run privileged instructions.
- These system calls, in the guest OS, are also called “hypercalls.”

Performance is improved. The hypervisor can focus on isolating VMs and coordinating.

Hardware-assisted

Intel introduced "VT" in 2005, and AMD introduced "Pacifica" (AMD-V) in 2006

- re-implemented ideas from VM/370 virtualization support
- basically added a new CPU mode ("guest" and "host") to distinguish VMM from guest/app

Now building a VMM is easy!

- and VMware must make money some other way…

Hardware-assisted

Non-root mode	Ring 3	Guest applications
	Ring 2	
	Ring 1	
	Ring 0	Guest OS kernel
Root mode	Ring -1	Hypervisor
Host hardware		

likely to emerge as the standard for server virtualization into the future

e.g., Intel® VT-x, AMD® V

Hardware-assisted

Originally the machine is executing normally, without any guest OS.

<table>
<tr><td rowspan="4">Non-root mode</td><td>Ring 3</td><td></td></tr>
<tr><td>Ring 2</td><td></td></tr>
<tr><td>Ring 1</td><td></td></tr>
<tr><td>Ring 0</td><td></td></tr>
<tr><td>Root mode</td><td>Ring -1</td><td>Hypervisor</td></tr>
<tr><td colspan="3">Host hardware</td></tr>
</table>

Hardware-assisted

When the hypervisor launches a VM,

Non-root mode	Ring 3	Guest applications
	Ring 2	
	Ring 1	
	Ring 0	Guest OS kernel
Root mode	Ring -1	Hypervisor
Host hardware		

Hardware-assisted

When the guest runs privileged instructions, it traps to hypervisor (VM exit) to exercise system control

<table>
<tr><td rowspan="4">Non-root mode</td><td>Ring 3</td><td>Guest applications</td></tr>
<tr><td>Ring 2</td><td></td></tr>
<tr><td>Ring 1</td><td></td></tr>
<tr><td>Ring 0</td><td>Guest OS kernel</td></tr>
<tr><td>Root mode</td><td>Ring -1</td><td>Hypervisor</td></tr>
<tr><td colspan="3">Host hardware</td></tr>
</table>

VM exit

Hardware-assisted

When the hypervisor finishes, the control switches back to non-root mode, the VM continues

Non-root mode	Ring 3	Guest applications
	Ring 2	
	Ring 1	
	Ring 0	Guest OS kernel
Root mode	Ring -1	Hypervisor
Host hardware		

VM entry

A Summary

	Full	Para-	Hardware-assisted
Handling privileged insturctions	binary translation	hypercalls	non-root / root mode
Guest OS modifications	No	Yes	No
Performance	Good	Best	Better
Examples	VMware, VirtualBox	Xen	Xen, VMware, VirtualBox, KVM

Cloud infrastructures

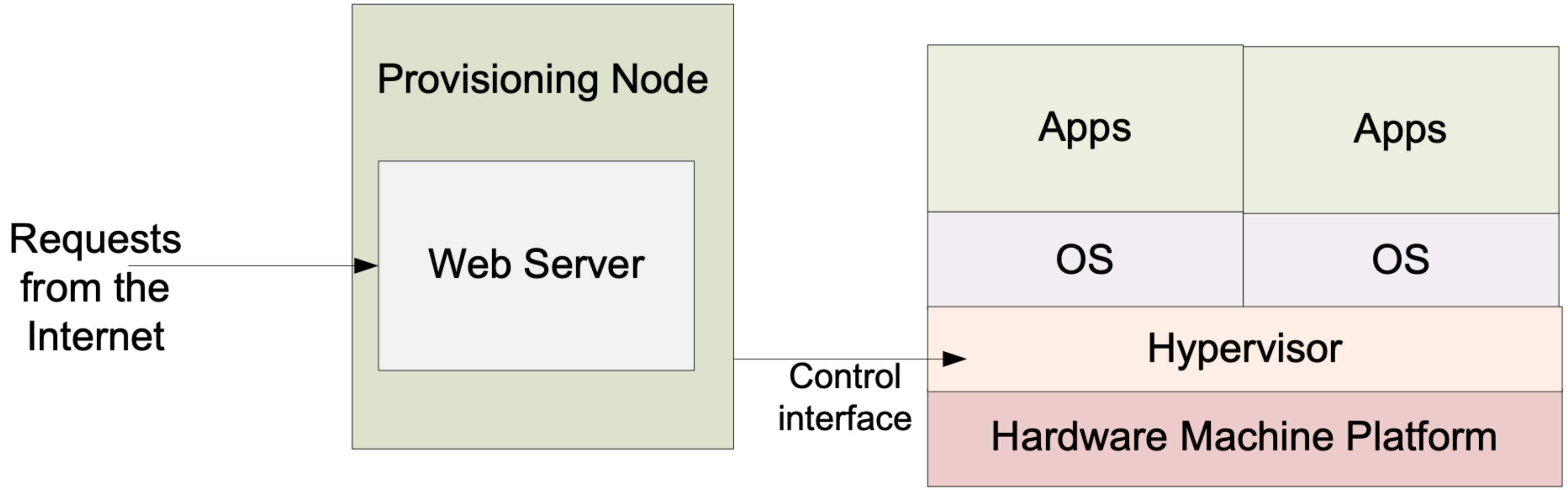
Cloud & Virtualization

Cloud computing is usually related to virtualization

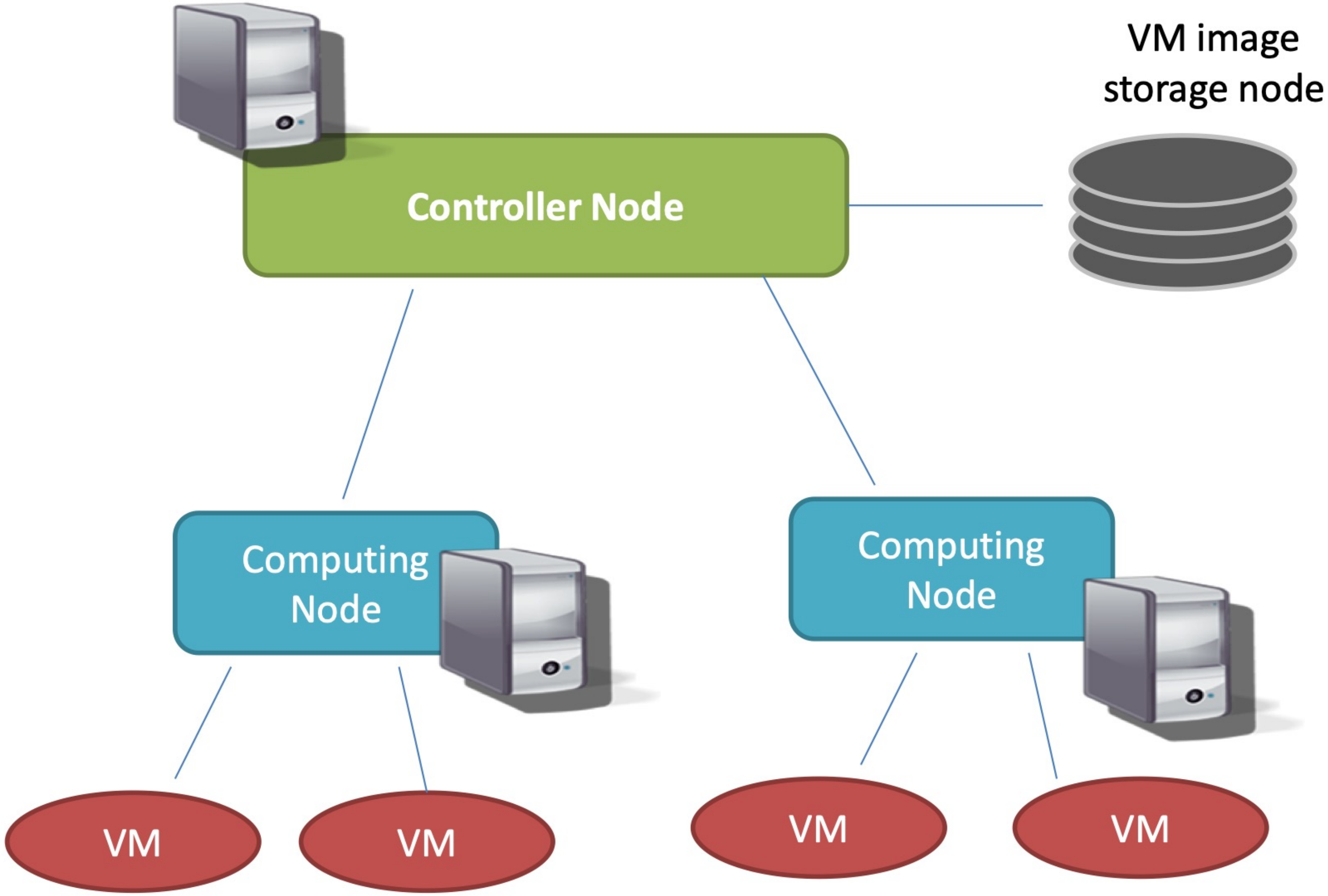
- highly elastic
- launching new VMs in a virtualized environment is cheap and fast
- consolidating multiple VMs onto one physical machine improves the utilization

Cloud & Virtualization

A cloud infrastructure is in fact a **VM management infrastructure**



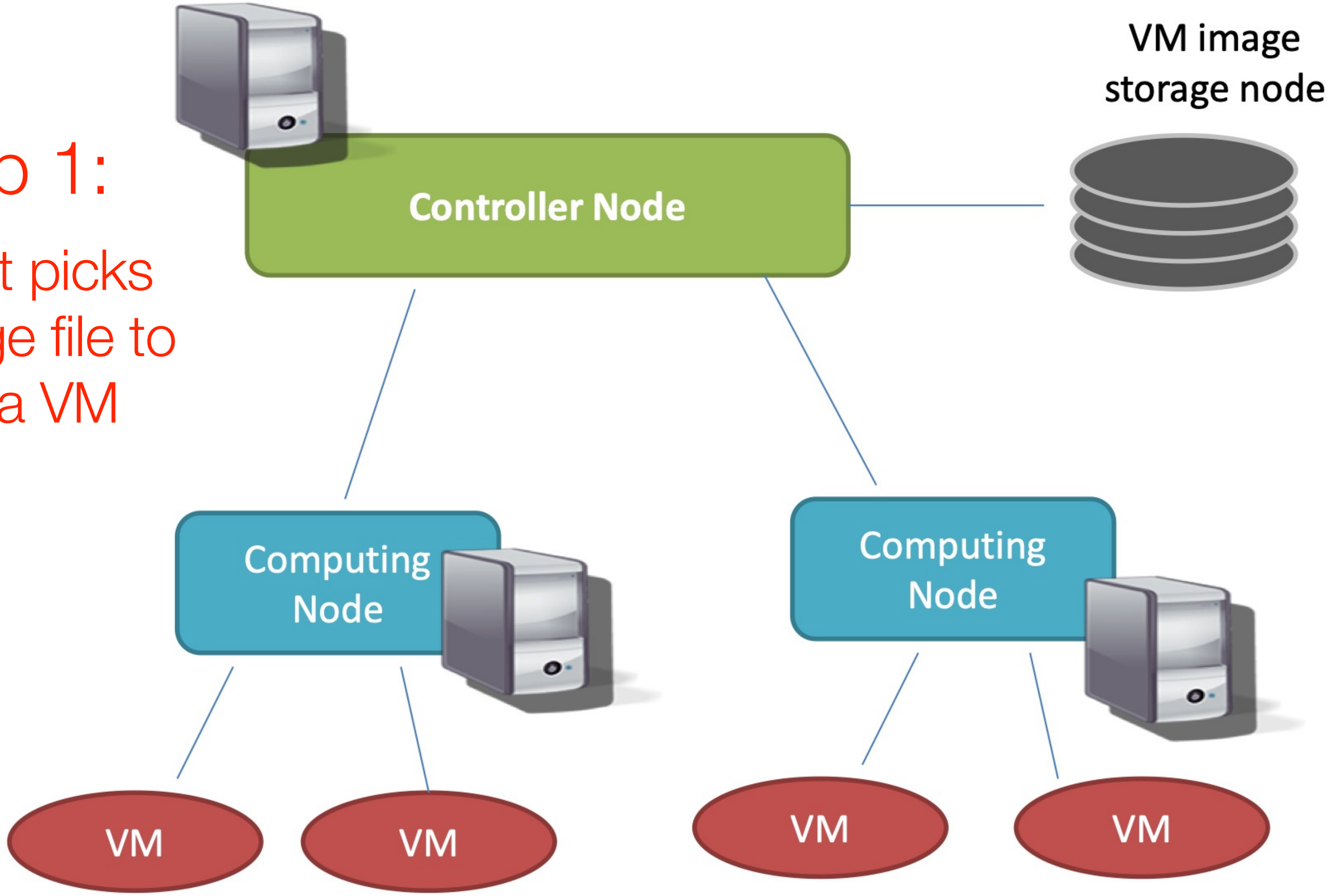
An IaaS Cloud



IaaS – Control Flow

Step 1:

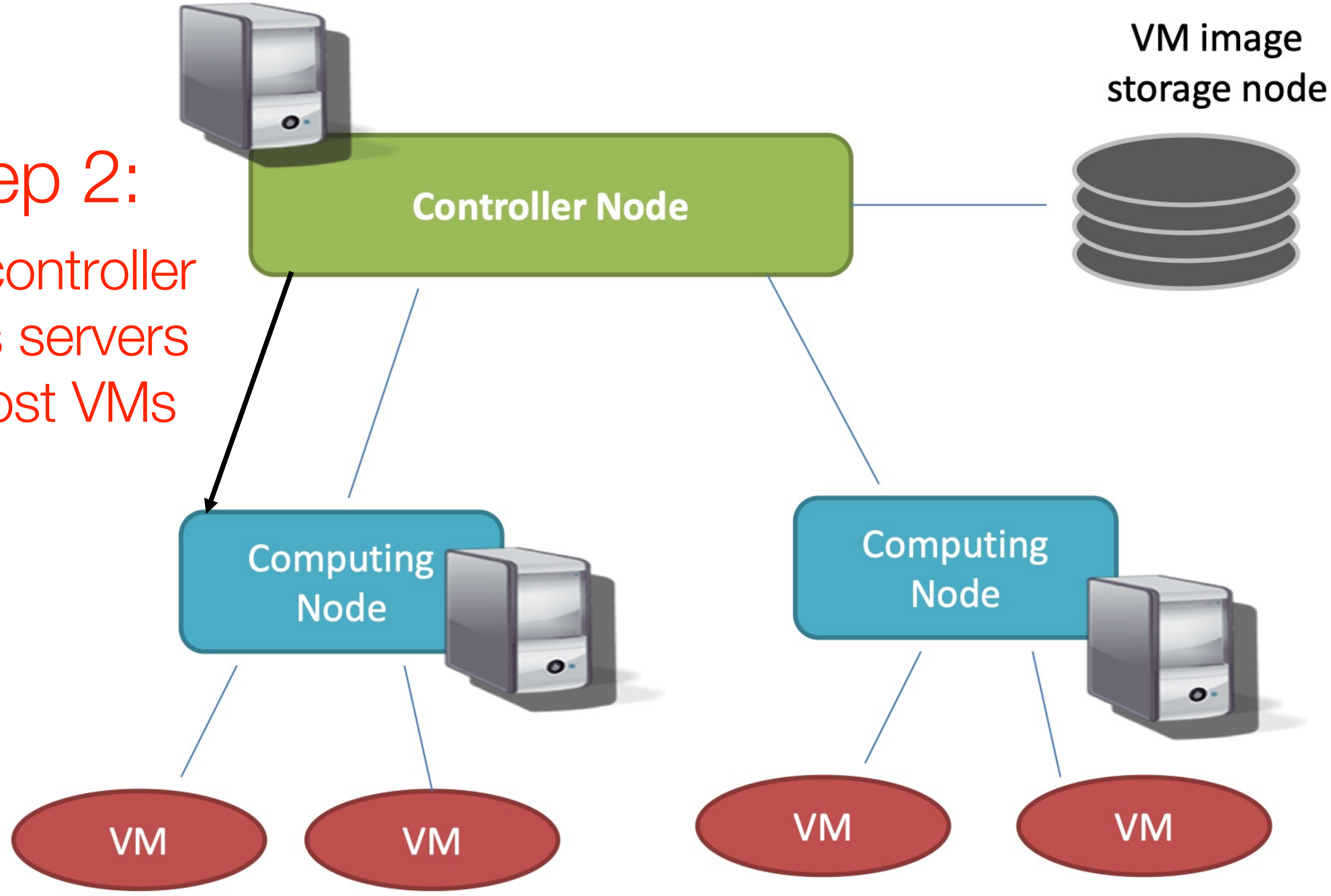
A client picks an image file to start a VM



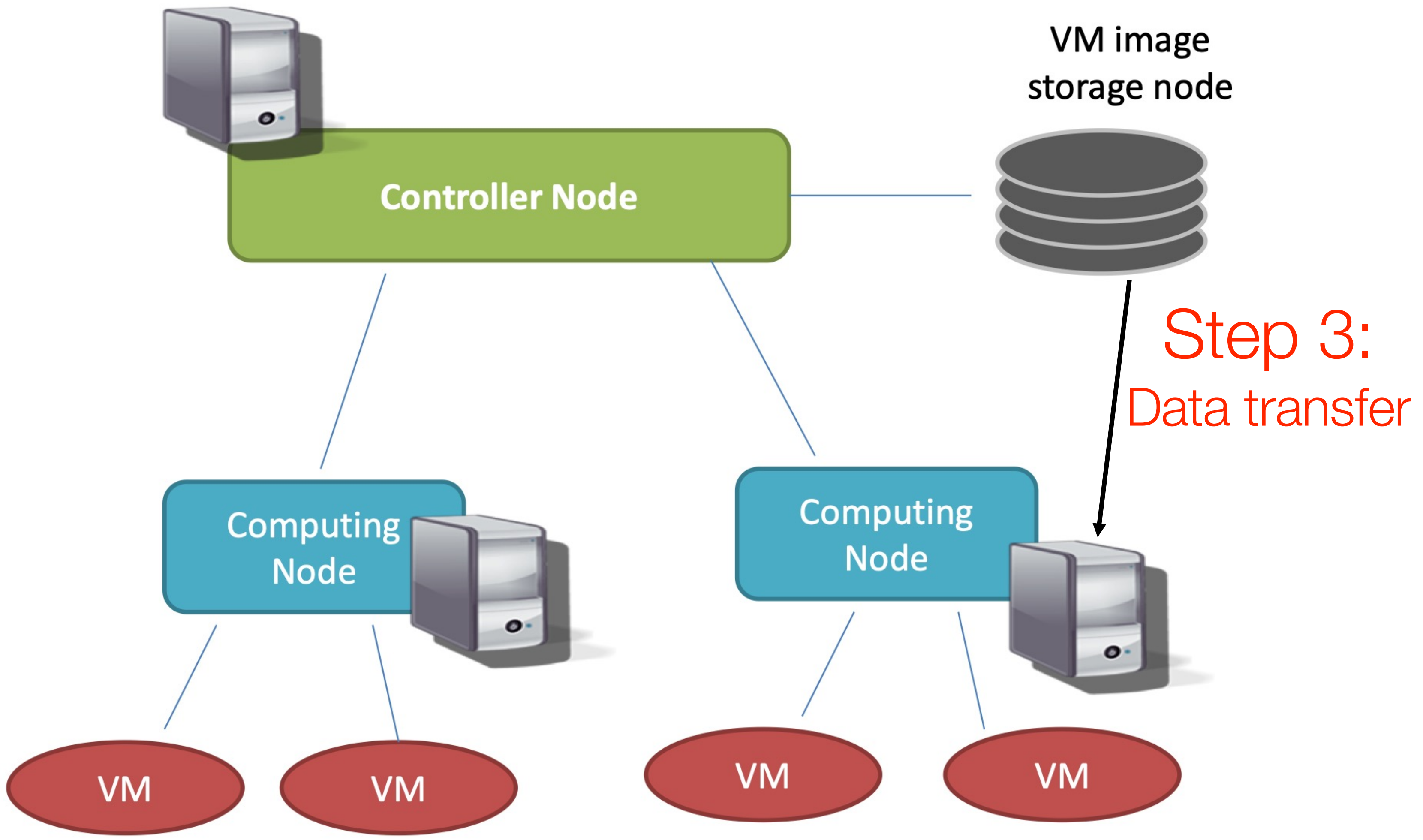
IaaS – Control Flow

Step 2:

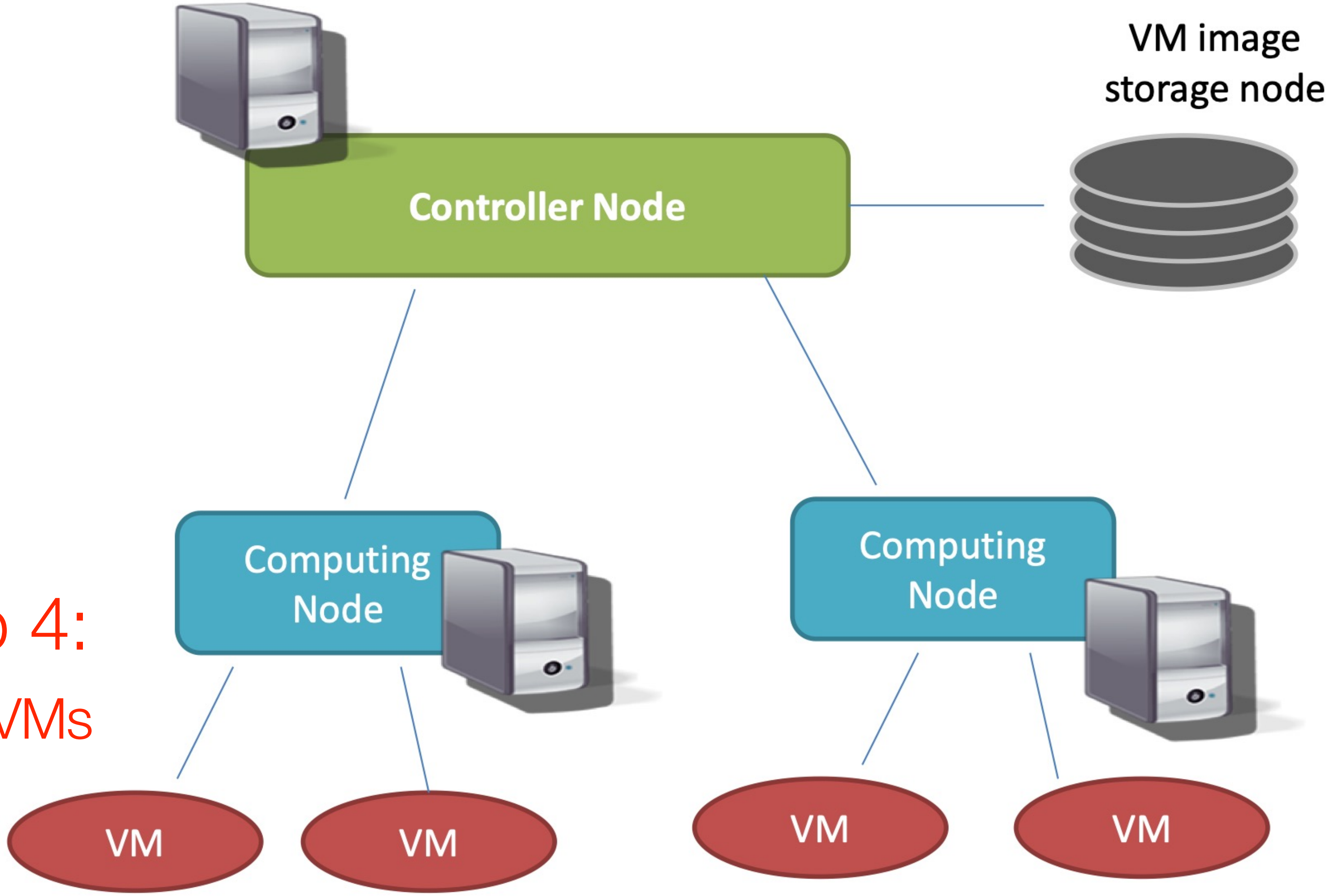
The controller picks servers to host VMs



IaaS – Control Flow



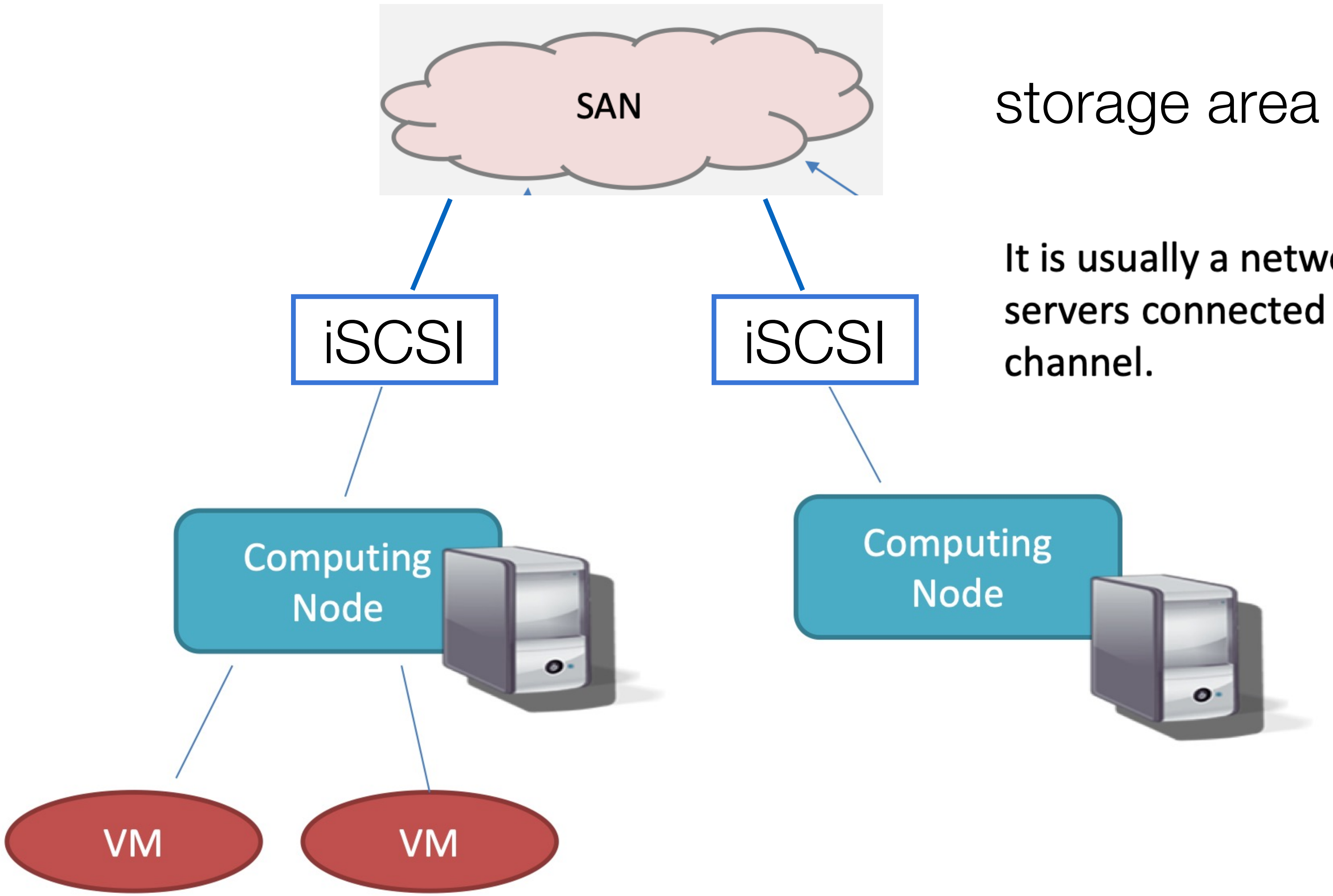
IaaS – Control Flow



Step 4:

Start VMs

Subtleties



storage area network

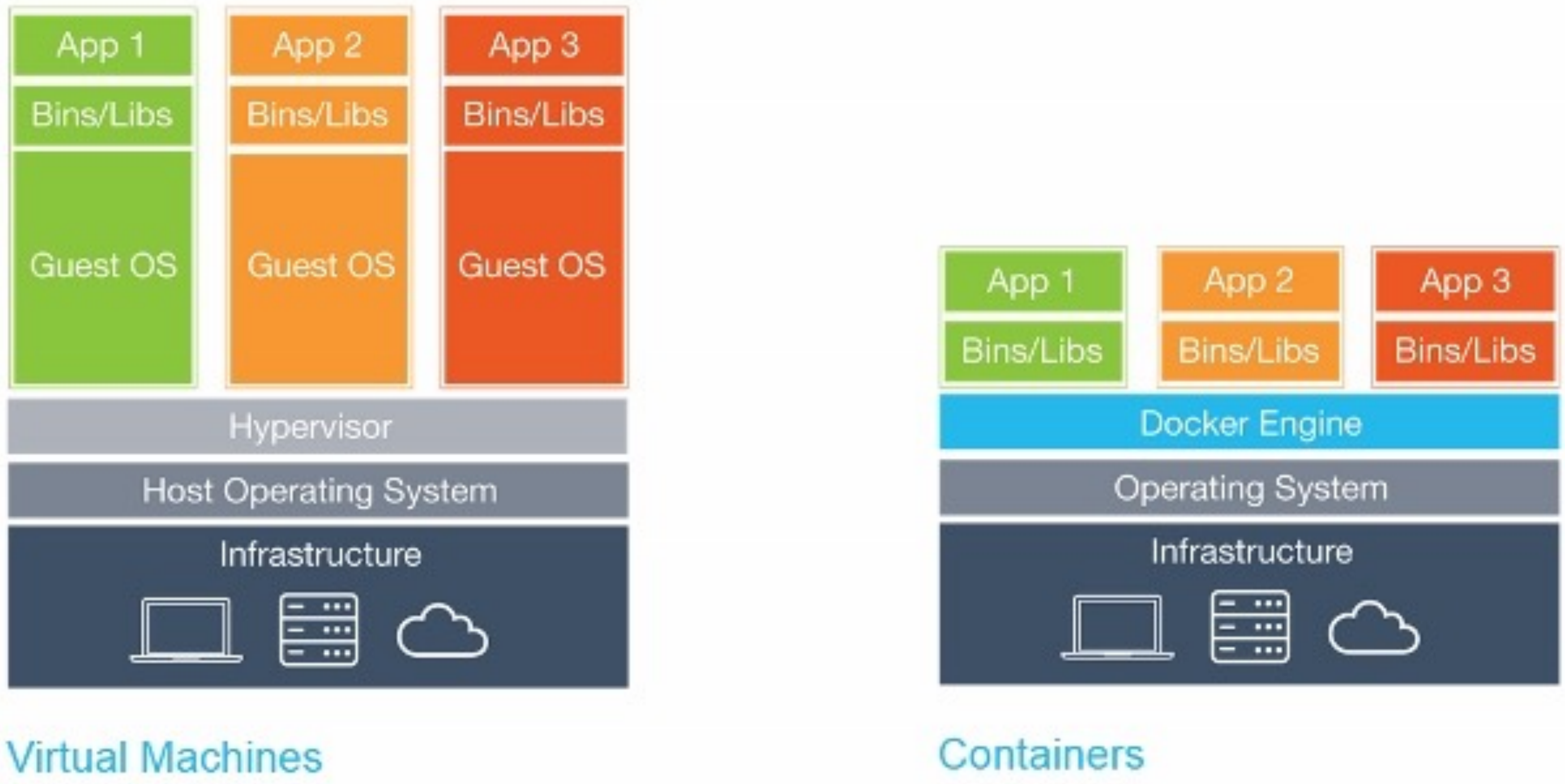
It is usually a network of storage servers connected using fabric channel.

Subtleties

Virtualization used in cloud?

- Yes for **public** cloud
- for **private** cloud, it depends…
  - Google's clusters are all built on top of **bare metal**: high efficiency without performance penalty

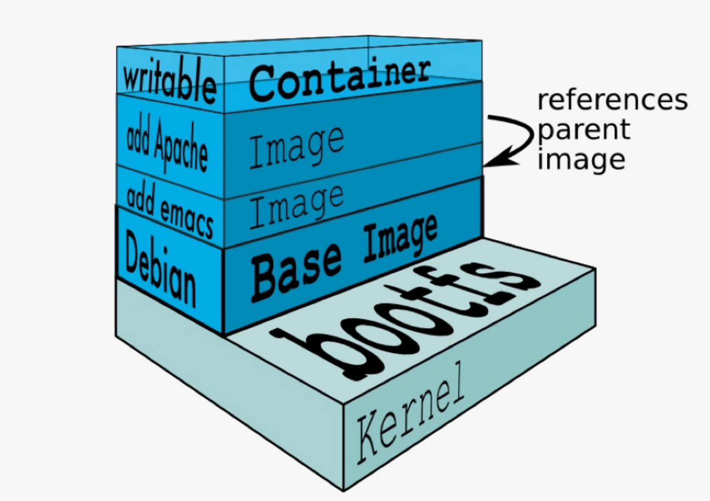
The rise of container



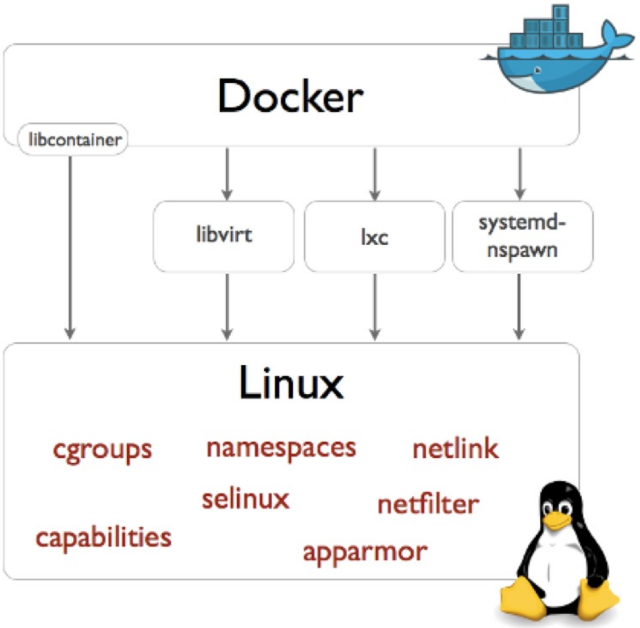
Containers

A light form of resource virtualization based on kernel mechanisms like cgroups and namespaces

- Multiple containers run on the same kernel with the illusion that they are the only one using resources



taken from the Docker documentation



VM vs. Container

VM system call path

- application inside the VM makes a system call
- trap to the hypervisor (or host OS)
- hand trap back to the guest OS

Container virtualization system call path

- application inside the container makes a system call
- trap to the OS
- OS returns the results to application

No binary translation,
no emulation

Credits

- Some slides are adapted from course slides of COMP 4651 in HKUST