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 最先进的实现
- Cloud infrastructures 云基础设施

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

Users demand…

具有不同计算能力的不同机器

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

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

预装不同的软件和库

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

不同的网络要求（拓扑、安全性、防火墙等）

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

Can any of these be easily provisioned with “bare metal?”

支持技术

Virtualization is an enabling technology for IaaS Cloud

What is virtualization?

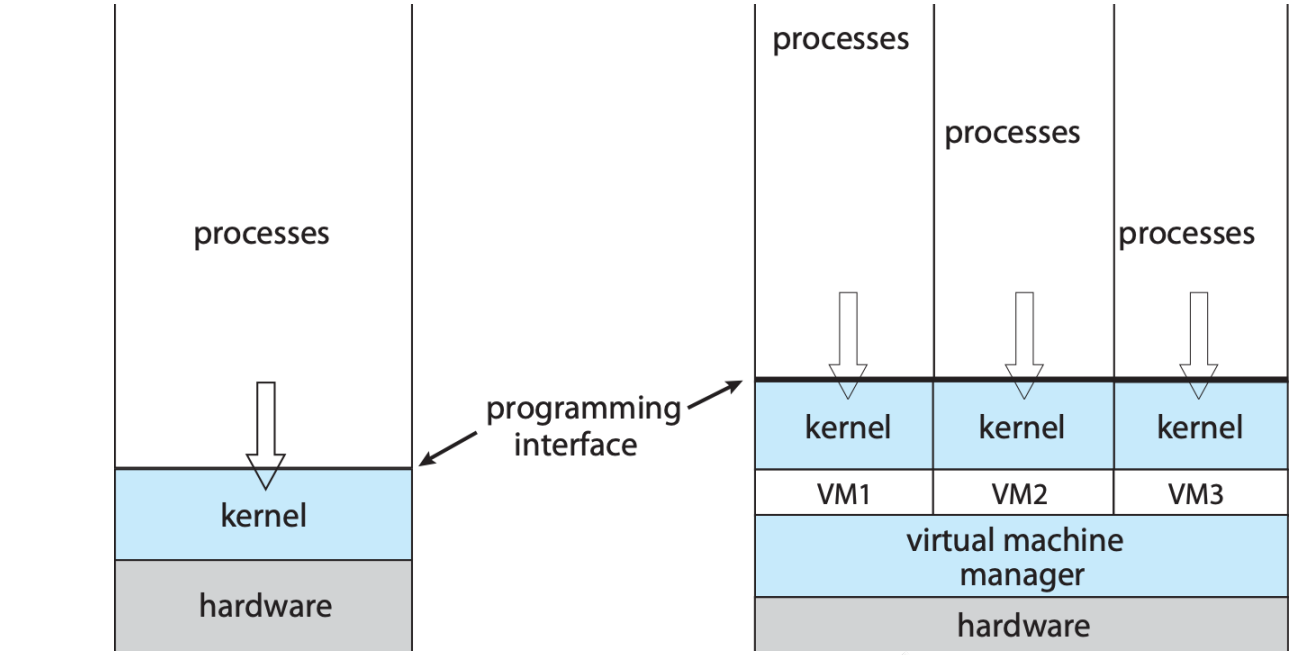
虚拟化是一个广义的术语。它可以应用于所有类型的资源(CPU、内存、网络等)。

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.

通过在多个环境之间共享单台机器的资源,使一台计算机 “ 看起来像 ” 多台计算机,执行多项任务

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 拥有者：底层硬件系统

- the underlying hardware system

Virtual Machine Manger (VMM) or **hypervisor** 虚拟机管理器(VMM)或虚拟机管理程序

- creates and runs virtual machines by providing interface that is ***identical*** to the host (except in the case of paravirtualization)
  通过提供与虚拟机相同的接口来创建和运行虚拟机主机(半虚拟化情况除外)
- a VMM is essentially a simple operating system
  VMM 本质上是一个简单的操作系统

Several components

A **virtual machine (VM)**

虚拟机(VM)

- a software-based implementation of some real (hardware-based) computer

一些真的(基于硬件)计算机的基于软件的实现

- in its pure form, supports booting and execution of unmodified OSs and apps

以纯净形式支持启动和执行未修改的操作系统和应用程序

Guest

访问者
通常是操作系统

- usually an operating system

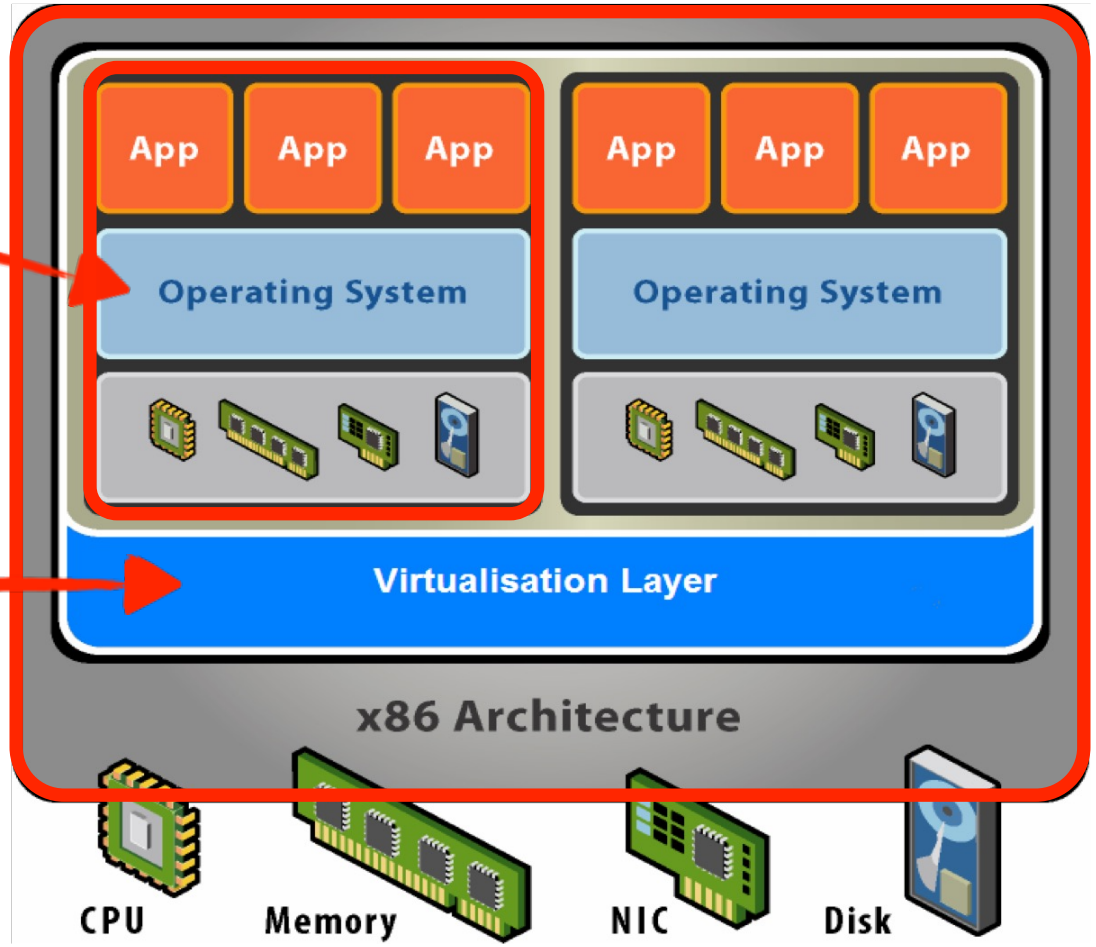
Several components

Guest OS

Guest VM

VMM

Host



Implementation of VMMs

实现：差异很大

Varies greatly

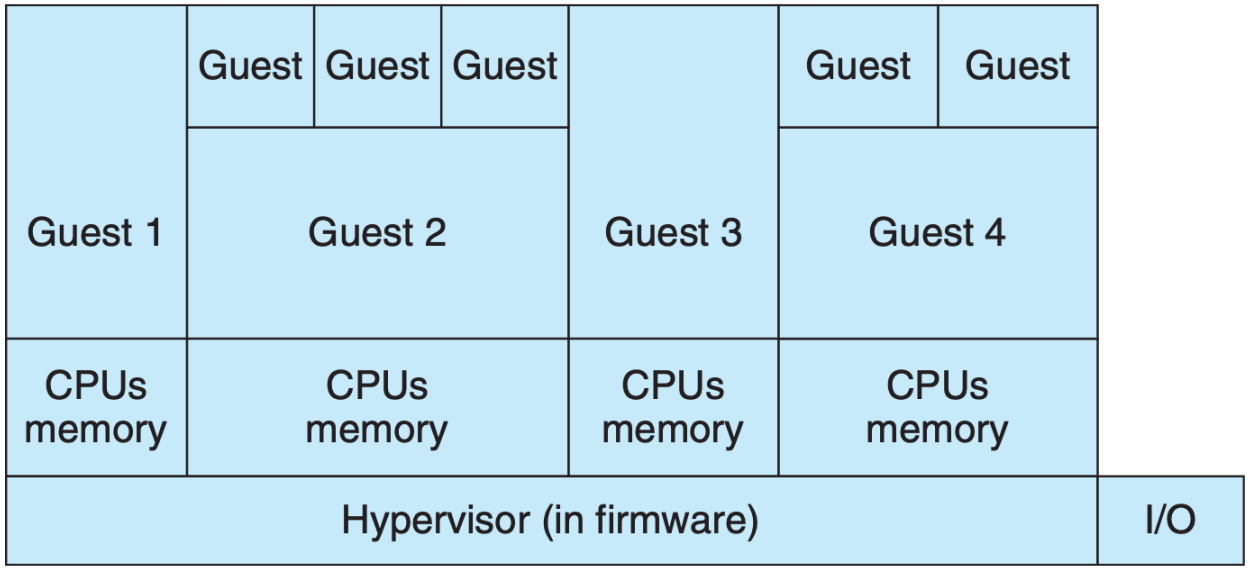
Type-0 hypervisors

基于硬件的解决方案,通过固件支持虚拟机创建和管理

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

常见于大型和大中型服务器，例如IBM

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

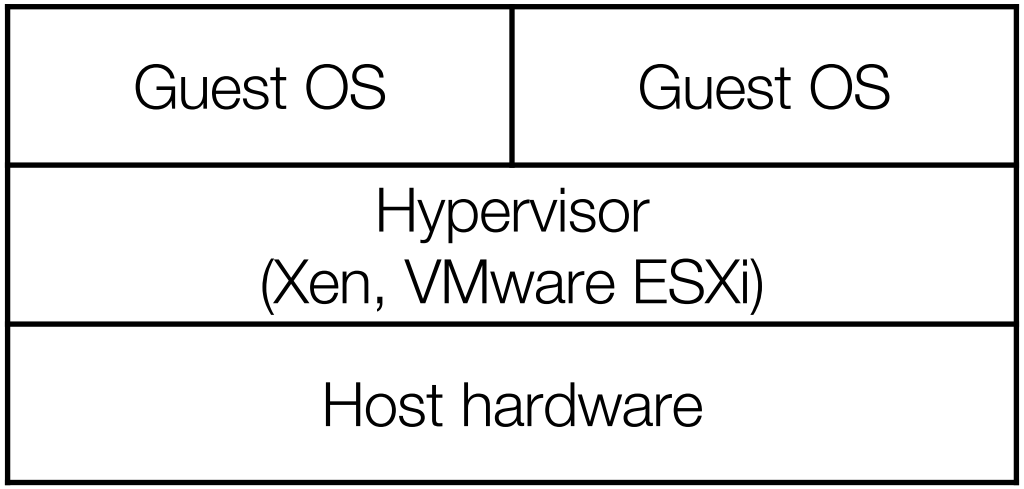


Type-1 hypervisors

类似操作系统的软件上提供虚拟化层，基于 x86 系统

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 生产云适用



Type-1 hypervisors

Also include general-purpose operating systems that provide standard functions as well as VMM functions 还包括提供标准功能和 VMM 功能的通用操作系统

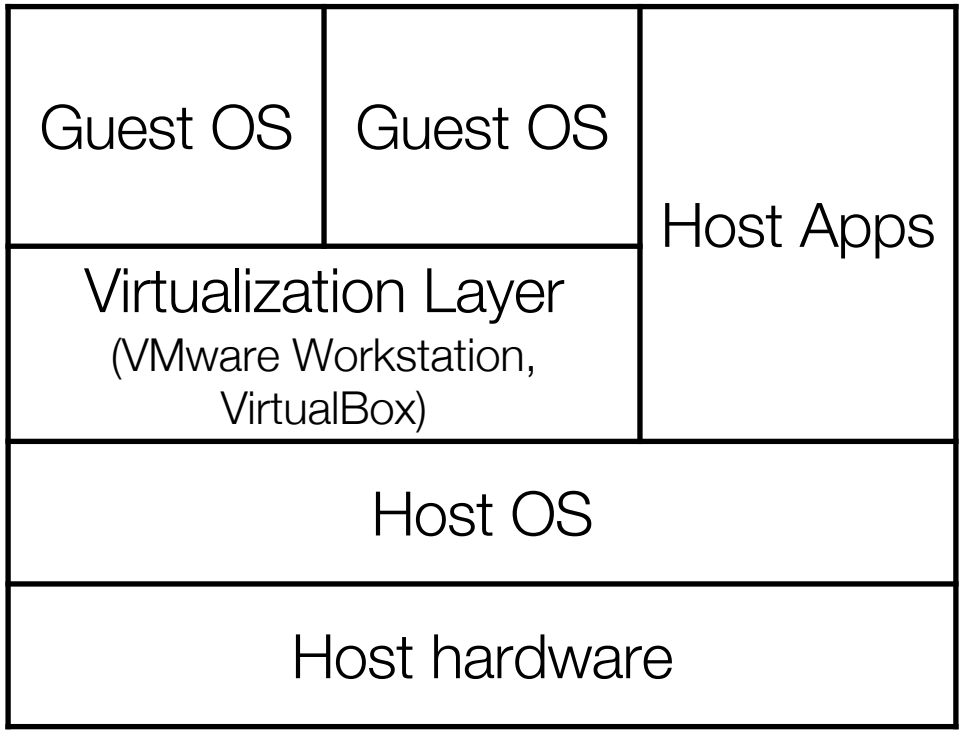
- e.g., Microsoft Windows Server w/ HyperV and RedHat Linux w/ KVM, Oracle Solaris
- typically less feature rich than dedicated type-1 hypervisors

通常功能不如专用的 1型虚拟机管理程序丰富

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



VMM 只是另一个进程,由主机运行和管理

甚至主机都不知道他们是运行 VMM 的客户机

Type-2 hypervisors

Indirect access to hardware through the host OS

通过主机操作系统间接访问硬件

- performance not good

性能不佳

- need administration privilege granted by the host OS

需要主机操作系统授予的管理权限

- usually for desktops and personal use, but not the production cloud environments

通常用于桌面和个人用途,但不用于生产云环境

Other variations

Para-virtualization

半虚拟化
修改客户操作系统,使其与 VMM 协同工作,以优化性能的技术

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

Programming-environment virtualization

编程环境虚拟化

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

VMM 并不虚拟化真实硬件,而是创建一个优化的虚拟系统,例如 JVM 和Microsoft.Net

Emulators

模拟器
允许为一种硬件环境编写的应用程序在不同的硬件环境,例如 iOS 模拟器

- 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

1型虚拟拟机管理程序和半虚拟化是云端最受欢迎的选择

History

Before there were datacenters

Early commercial computers were mainframes

早期的商用计算机是大型机

IBM 704 (1954): $250K - millions

Issues with early mainframes

Early mainframe families had some disadvantages 早期的大型机系列有一些缺点

- successive (or even competing) models were NOT architecturally compatible!
  连续的(甚至是竞争的)模型在架构上并不兼容
  - massive headache to update HW: gotta port software 更新硬件非常头疼:必须移植软件
- the systems were primarily *batch-oriented* 该系统主要以批处理为主

In the mean time…

MAC(多路访问计算机)项目启动

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

- responsible for developing Multics, a time-sharing OS
  负责开发分时操作系统Multics
- invented many of the modern ideas behind time-sharing OS
  发明了分时操作系统背后的许多现代理念计算机
- the computer was becoming a multiplexed tool for a community of users, rather than a batch tool for programmers
  正在成为用户社区的多路复用工具,而不是程序员的批处理工具

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

IBM 等大型机公司即将被远远地抛在后面

Big blue's bold move

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

IBM 将整个公司押注于System/360

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

首先明确区分架构和实现
其架构可虚拟化

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

CP/CMS系统软件意外大获成功

- 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

CP:创建和管理虚拟 S/360 机器的 " 控制程序

CMS: " 剑桥监控系统 " 轻量级单用户操作系统

在同一硬件上同时运行多个不同的操作系统

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

S/360 (1964-1970)

S/370 (1970-1988)

S/390 (1990-2000)

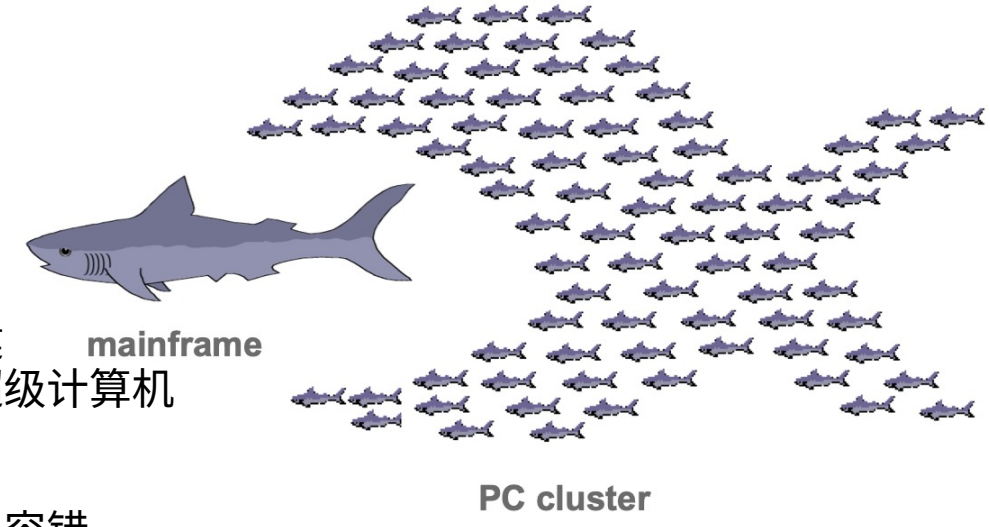
zSeries (2000-present)

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

In the meantime…

The PC revolution began 与此同时，PC 革命开始了

- much less powerful, but enjoy massive **economies of scale**
  Cluster computing (1990s) 实力弱得多,但享有巨大的规模经济集群计算(20 世纪 90 年代)
  - build a cheap mainframe or supercomputer out of a cluster of commodity PCs
    利用商品集群构建廉价的大型机或超级计算机
  - use clever software to get fault tolerance
    使用智能软件实现容错



Mendel Rosenblum makes it BIG

VMware 于 1998 年从斯坦福 DISCO 项目中分离

VMware spun up from Stanford DISCO project in 1998

将 CP/CMS 风格的虚拟化引入 PC

- brought CP/CMS-style virtualization to PC

Initial market was software developers

最初的市场是软件开发商

- often need to develop and test software on multiple OSs (windows, Linux, MacOS, …)

经常需要在多个操作系统(Windows、Linux、MacOS…

- can afford multiple PCs, or could dual-boot, but inconvenient

可以买得起多台电脑,或者可以双启动,但不方便

- instead, run multiple OSs simultaneously in separate VMs

因此,在单独的虚拟机中同时运行多个操作系统

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

与大型机VM 的动机类似,但原因相反：
现在计算机太多了,而不是太少!

The real PC virtualization moneymaker

Enterprise consolidation

企业整合

- big companies usually have their own clusters or datacenters

  大公司通常有自己的集群或数据中心

  - operate many services: mails, Webs, files, remote cycles

    运营多种服务 :邮件、网站、文件、远程循环

  - want to run **one service per machine** (best admin practice)

    希望每台机器运行一项服务(最佳管理实践)

  - leads to **low utilization**!

    这导致利用率低下

- instead, run **one service per VM**

  相反,每个虚拟机运行一个服务

The old model

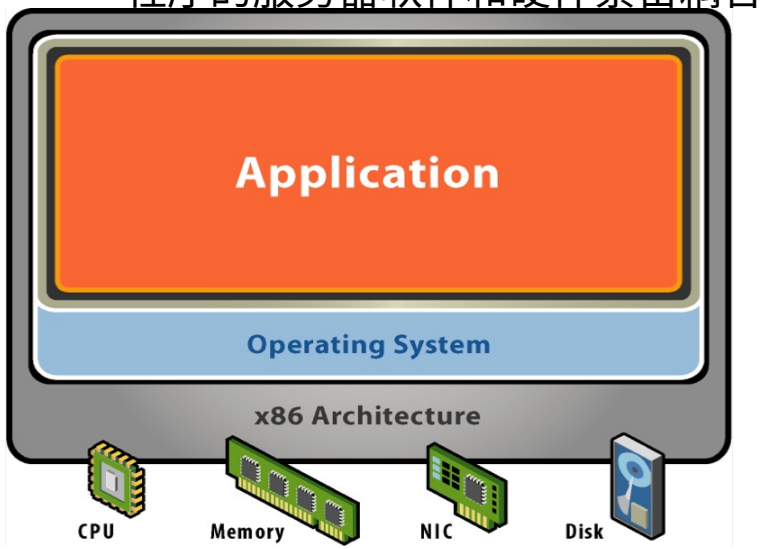
旧模式

A server for every application

适用于每个应用

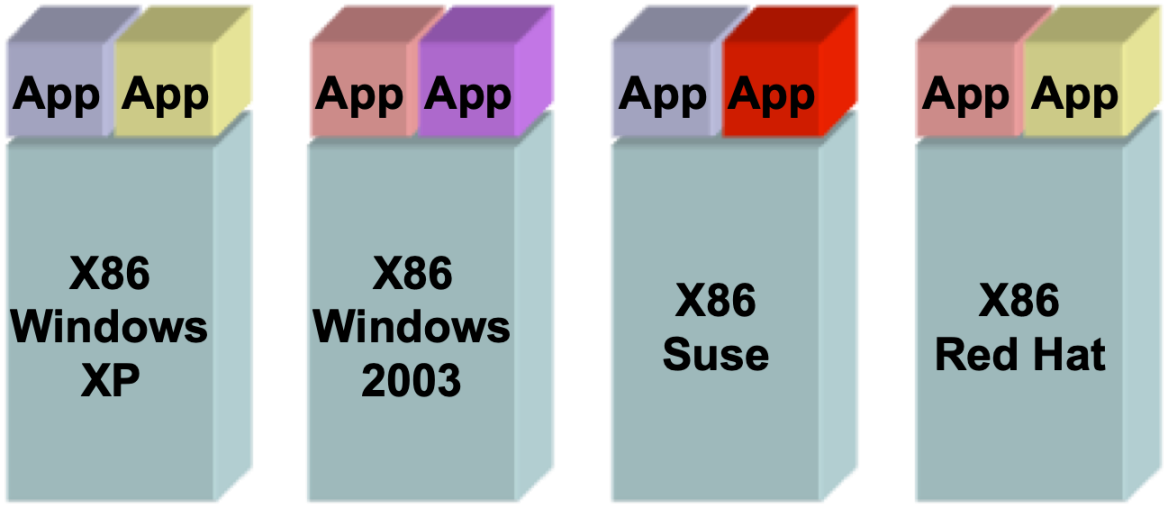
Software and hardware are tightly coupled

程序的服务器软件和硬件紧密耦合



The old model

Big disadvantage: **low utilization** 利用率低



Util. 10% 12% 18% 20%

The new model: Consolidation

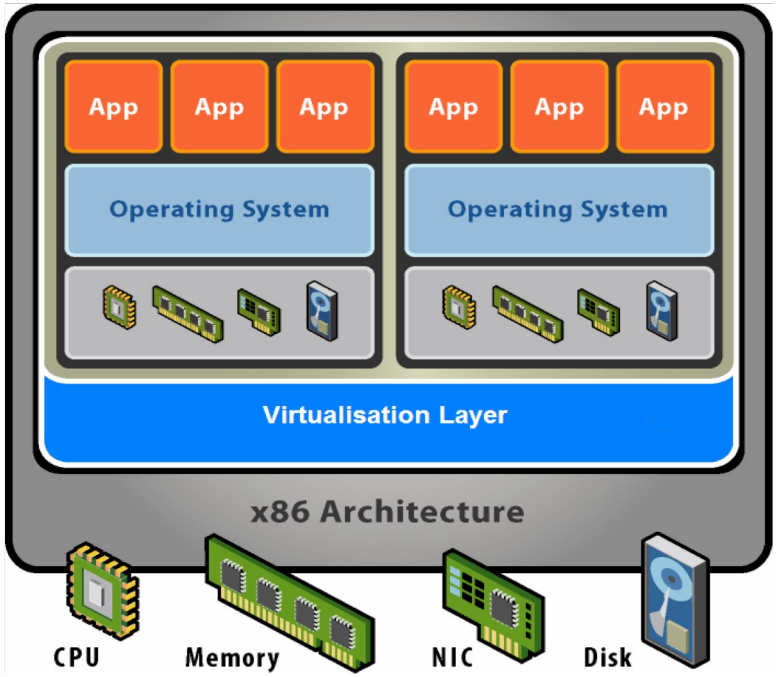
新模式:整合

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

通过将物理资源虚拟化。操作系统和应用程序封装到虚拟机中,将其作为一个单元

Separate applications and hardware

分离应用程序和硬件

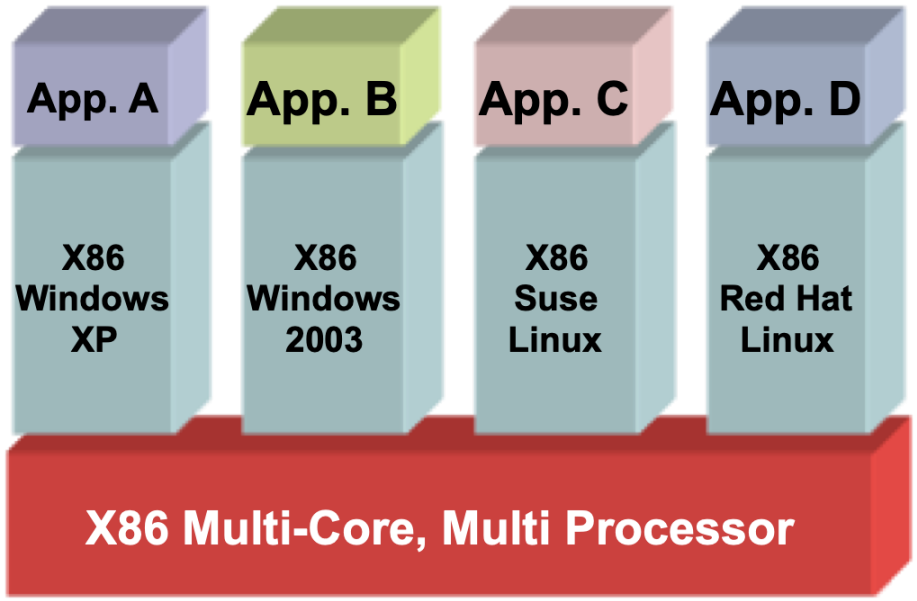


The new model: Consolidation

Big advantage: consolidation improves utilization

整合提高利用率

Individual Util. 10% 12% 18% 20%



Overall Util. 60%

Other benefits and features

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

隔离:主机系统受到保护,不受虚拟机影响:虚拟机之间相互保护

Freeze, suspend, running VM

冻结、暂停、正在运行的虚拟机

- they can move or copy somewhere else and resume

他们可以移动或复制到其他地方并继续

Great for OS research and better system development efficiency

非常适合操作系统研究并提高系统开发效率

Templating

模板

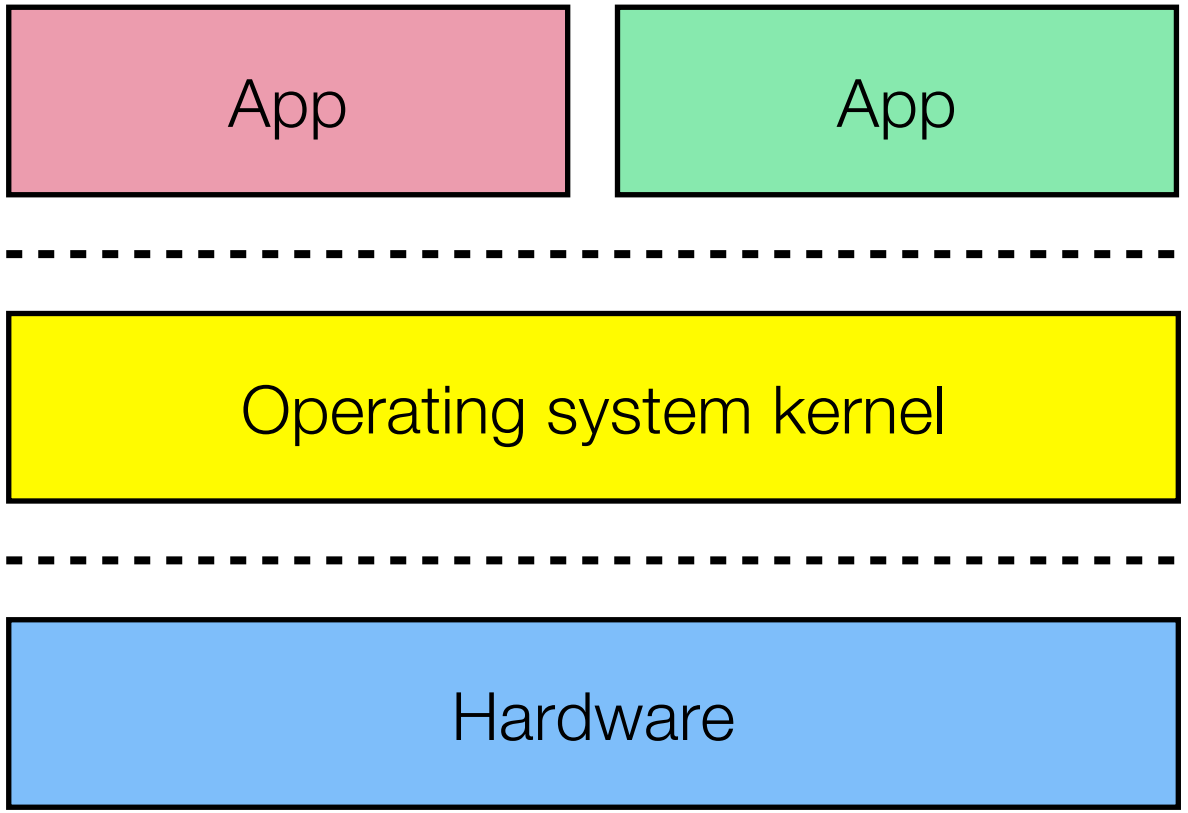
Live migration

实时迁移

The forefront of virtualization

How does virtualization work?

How do regular machines work?



user mode

system-call interface

系统调用接口

kernel mode

HW/SW interface

软硬件接口

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.

CPU:指令集、寄存器、中断、特权模式

存:可通过加载/存储访问的物理内存字
MMU 提供分页/分段和虚拟内存

I/O :磁盘网卡等,由程序化I/O或DMA控制

通过轮询或中断传递给软件的事件

其他设备:显卡、时钟、USB 控制器等

What is an OS?

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

为应用软件提供访问硬件资源的特殊软件层

- runs like any other program, but in a **privileged** (kernel) CPU mode

  像其他程序一样运行,但处于特权(内核)CPU 模式

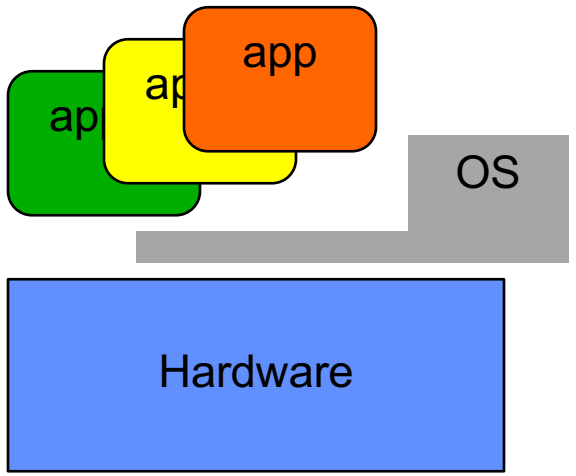
  - protects itself from user programs

    保护自己免受用户程序的侵害

  - can interact with HW devices using “sensitive” instructions

    可以使用“敏感”指令与硬件设备交互

What is an OS?



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

为应用程序提供高级编程接口(系统调用接口)

OS implements this interface using low-level HW devices

操作系统使用低级硬件设备实现此接口

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

文件打开/读/写 vs 磁盘块读/写

issues instructions to control HW on behalf of programs

代表程序发出控制 HW 的指令

What is an application?

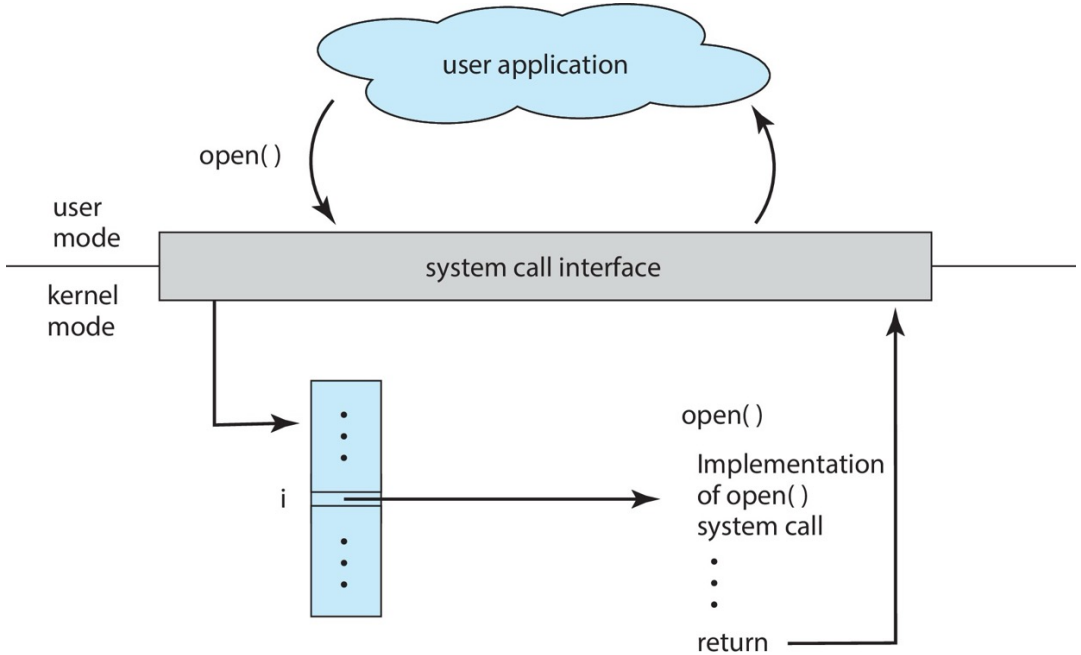
A program that relies on the system-call interface

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

依赖系统调用接口的程序

执行时,CPU 以非特权(用户)模式运行 - 个特殊指令(x86 上的 int)

允许程序调用操作系统



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
```

陷阱？

Traps to kernel

Traps to kernel

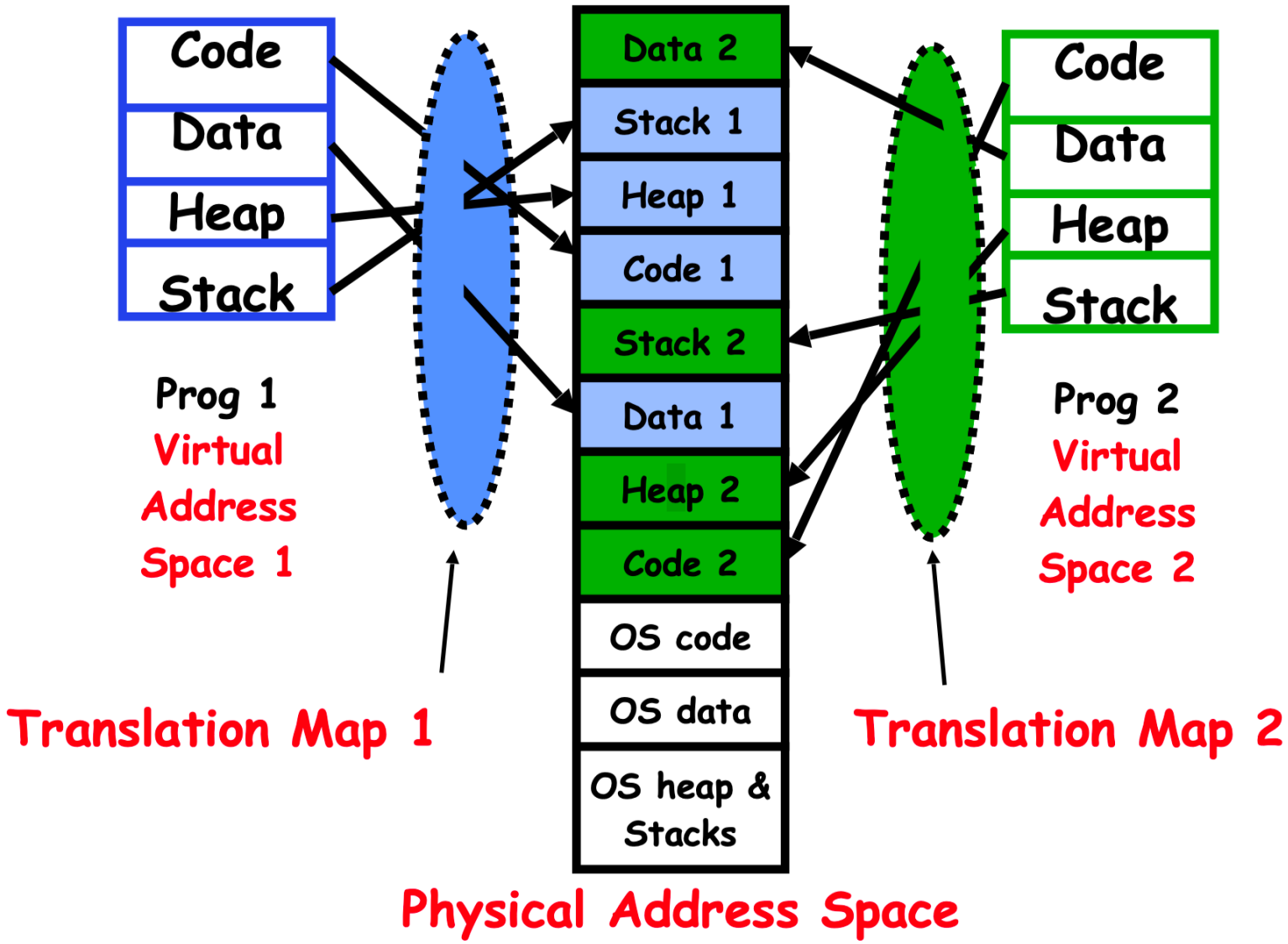
What is an application?

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

  操作系统为程序提供了自己内存的幻觉
  - MMU hardware lets the OS define the "**virtual address space**" of the program

    MMU 硬件让操作系统定义程序的 " 虚拟地址空间



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

Is this safe? 安全性

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

大多数指令直接在 CPU 上运行(快速
但敏感指令会导致 CPU 向操作系统抛出异常地址

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

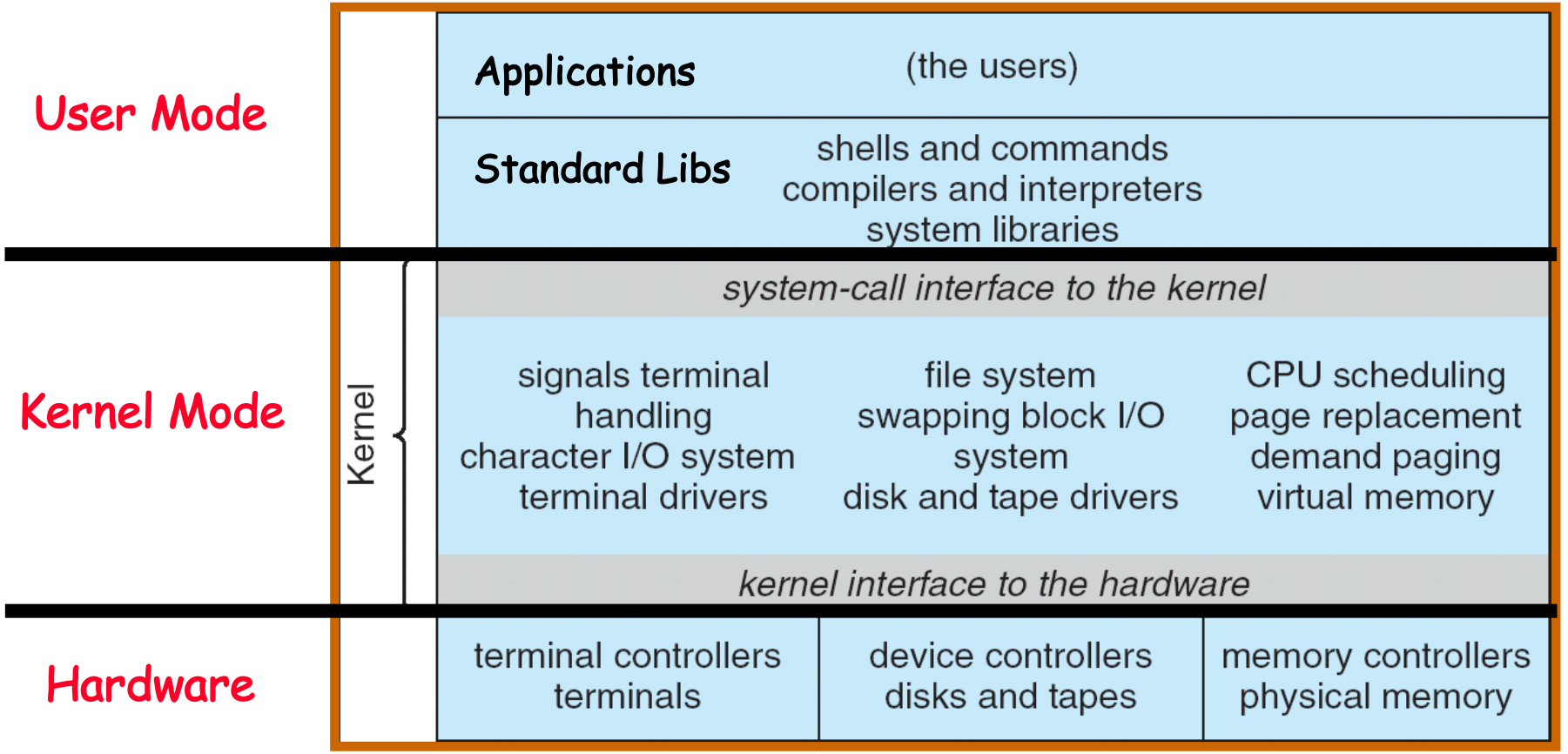
Address spaces prevent program from stomping on OS memory, each other

空间可防止程序相互侵犯操作系统内存

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

就好像每个程序都在自己的私有机器(“ 进程 ”)中运行

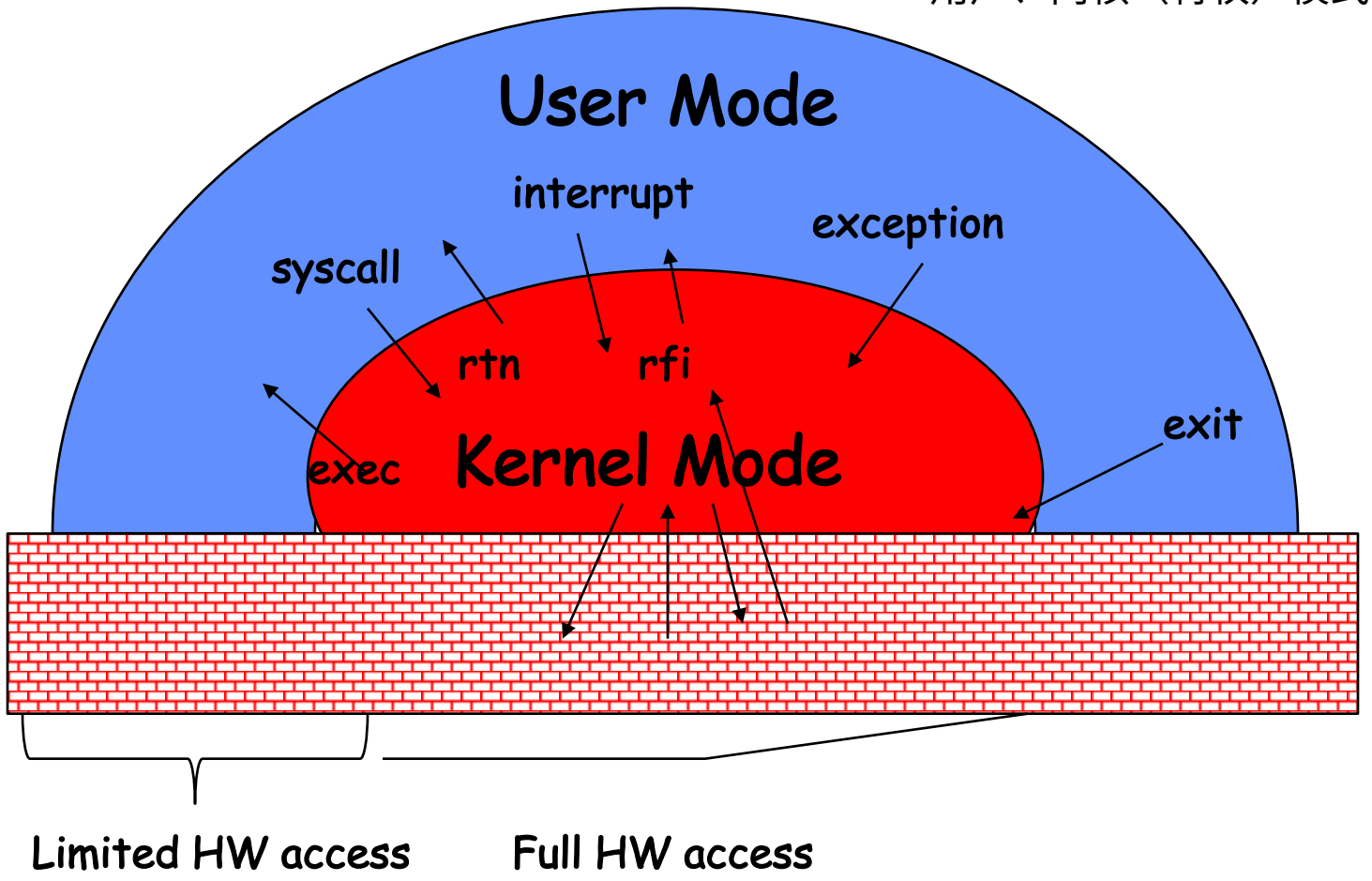
Putting them together



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

User/kernel (privileged) mode

用户、内核（特权）模式



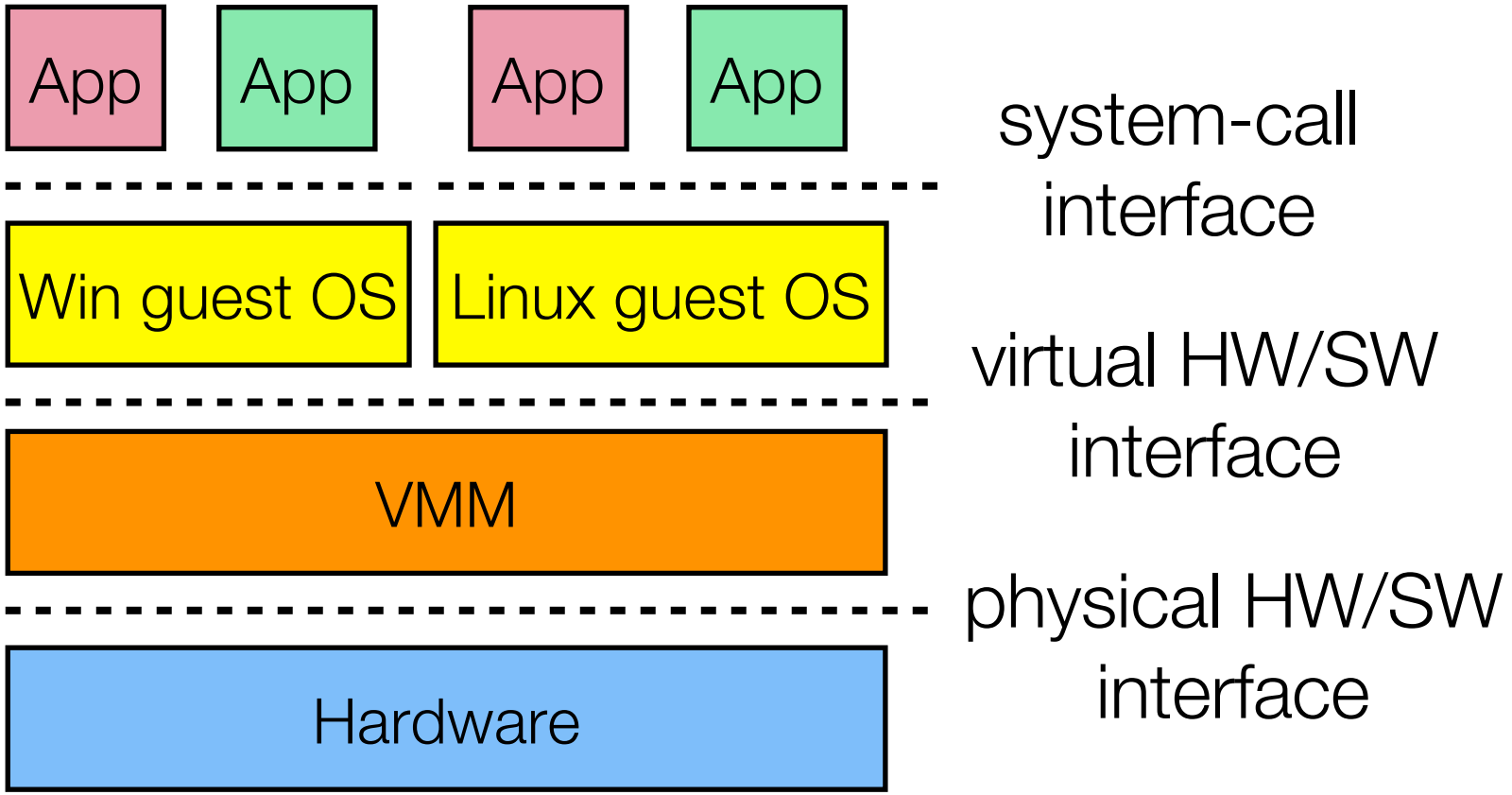
Credit: Prof. John Kubiatowicz's slides for CS162, Spring 2015, UC Berkeley

How does virtualization work?

A goofy idea…

只给操作系统用户权限？

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



It almost works, but…

当 Windows 在内核模式下发出敏感指令时会发生什么?W

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

What (virtual) hardware devices should Windows use?

indows 应该使用什么(虚拟)硬件设备?

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

我们如何防止在 Windows 上运行的应用程序损害 Windows?

- or apps from hurting the VMM…

或者应用程序损害 VMM……

- or Windows from hurting Linux… or the VMM…

或者 Windows 损害 Linux……或者 VMM……

Trap-and-emulate

设陷阱而后模仿

Trap and emulate

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

客户虚拟机需要两种模式：虚拟用户模式和虚拟内核模式

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

  两者都在真实用户模式下运行,因为让客户内核在内核模式上跑不安全

- only VMM runs in kernel mode

  只有 VMM 运行在内核模式下

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

客户操作系统中，导致切换到内核模式的操作，将导致虚拟机切换到虚拟内核模式

- but how?

Trap and emulate

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

从虚拟用户模式切换到虚拟内核模式是如何发生的?

- **Trap**: guest attempting a privileged instruction in user mode causes an error -> host traps to kernel mode

  陷阱:客户机在用户模式下尝试执行特权指令会导致错误 ->主机陷入内核模式

- **Emulate**: VMM gains control, analyzes the error, emulates the effect of instruction attempted by guest

  模拟: VMM 获得控制权,分析错误,模拟客户机尝试执行的指令的效果!

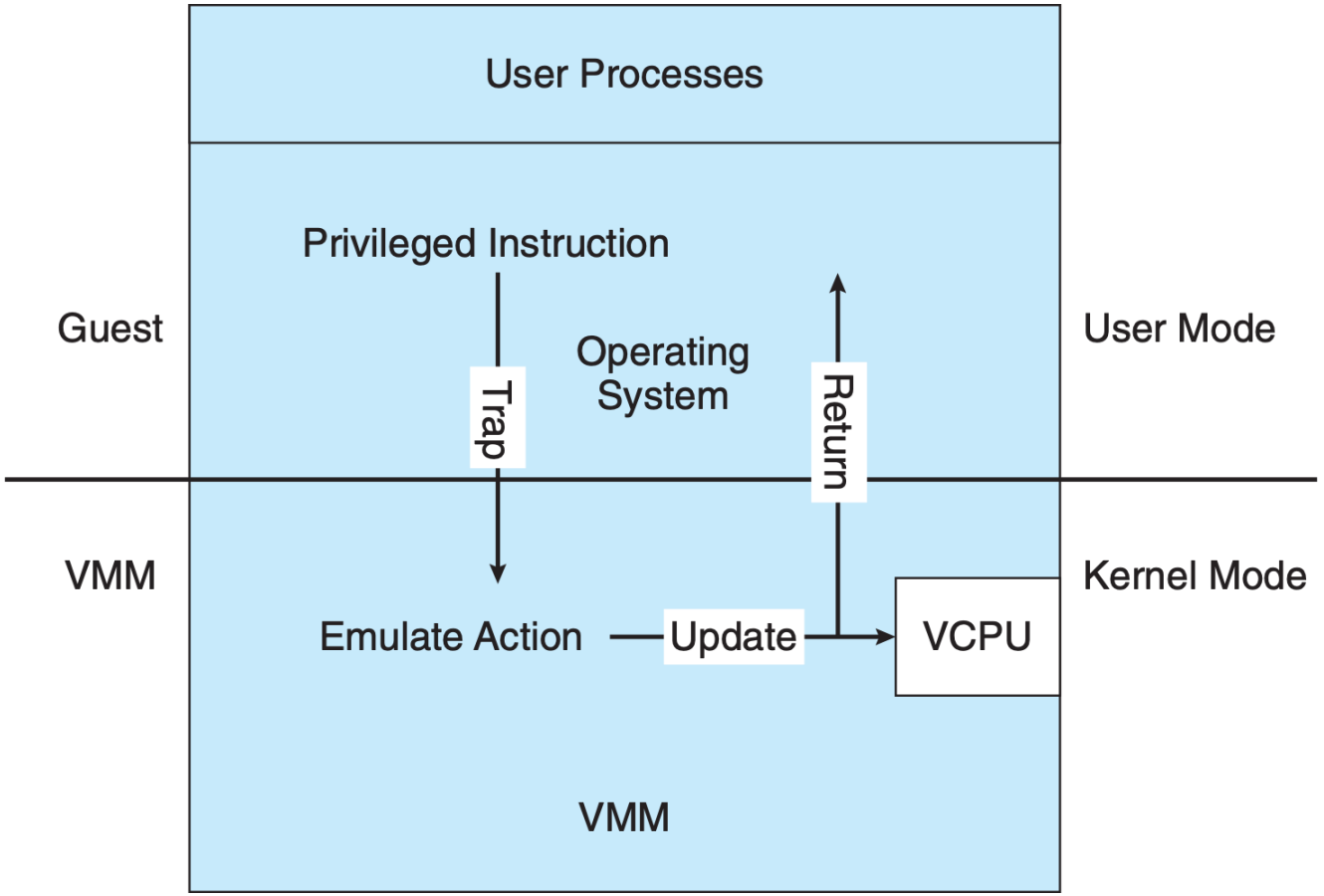
  - VMM provides a virtual HW/SW interface to guest

    VMM 为客户机提供虚拟硬件/软件接口

- **Return**: VMM returns control to guest in user mode

  返回: VMM 在用户模式下将控制权返回给 Guest

Trap and emulate



Most virtualization products use this at least in part

大多数虚拟化产品至少部分地使用了这一点

Correctness requirement

正确性要求

Two classes of instructions

Privileged instructions

特权指令
当 CPU 处于用户模式时,即需要内核模式时,捕获的

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

Sensitive instructions

敏感指令
修改(虚拟)硬件配置或资源的,以及其行为依赖于(虚拟)硬件配置的

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

例如 ,读取、写入、CPU寄存器设置

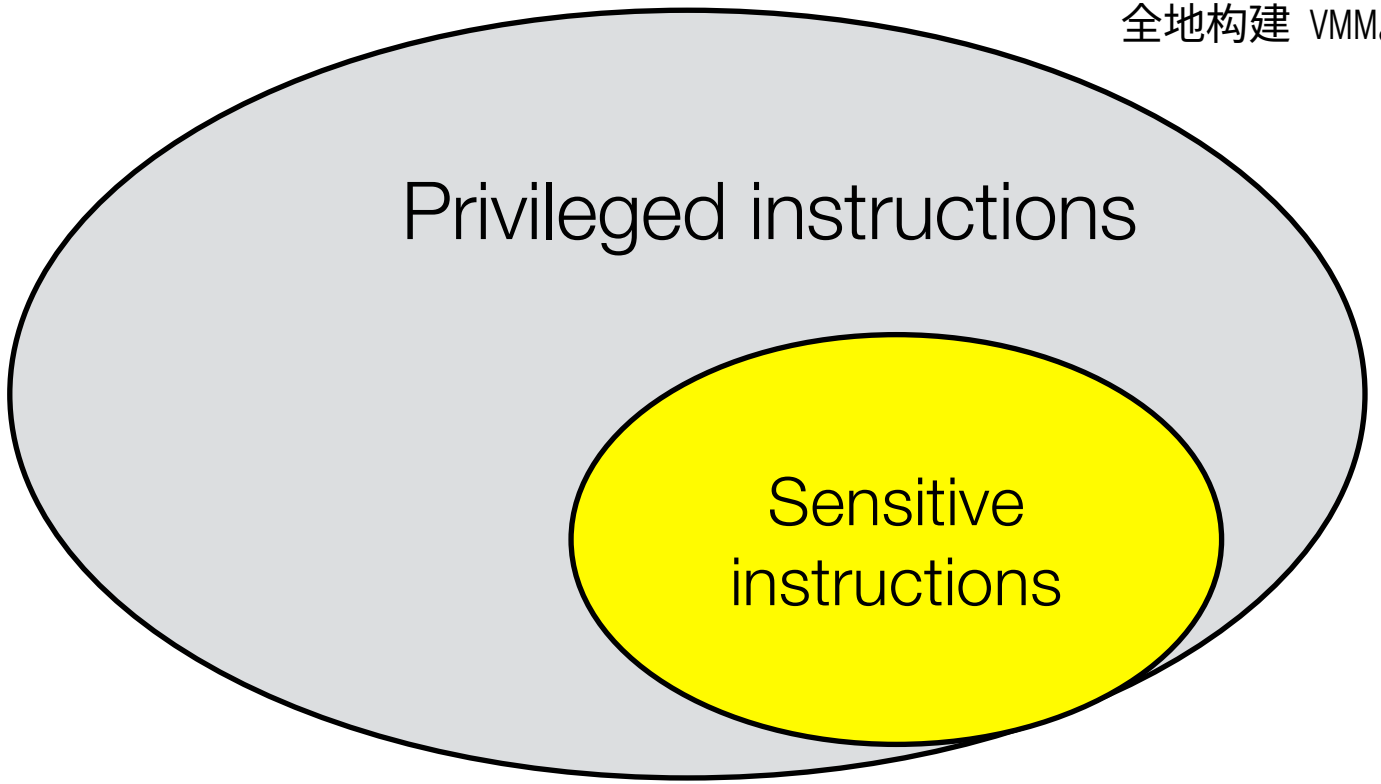
Emulation is only needed for sensitive instructions

仅敏感指令才需要模拟

Popek & Goldberg (1974)

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

如果敏感指令集是特权指令的子集,则可以高效、安全地构建 VMM。



How about the performance?

Non-sensitive instructions

非敏感指令

Almost no overhead

几乎没有开销

- they execute directly on CPU

它们直接在 CPU 上执行

- CPU-bound code execute at the same speed on a VM as on a bare metal

CPU 绑定代码在虚拟机上的执行速度与在裸机上的执行速度相同

  - e.g., scientific simulations

例如科学模拟

Sensitive instructions

敏感指令

Significant performance hit! 性能受到显著影响

- they raise a trap and must be vectored to and emulated by VMM

  它们会引发陷阱,必须由 VMM 引导和模拟

  - each instruction could require tens of native instructions to emulate

    每条指令可能需要数十条本机指令来模拟

- I/O or system-call intensive applications get hit hard

  I/O 或系统调用密集型应用程序受到严重影响

Trap-and-emulate not always works

诱捕和模仿并不总是有效
英特尔架构不符合 Popek和 Goldberg 的要求

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

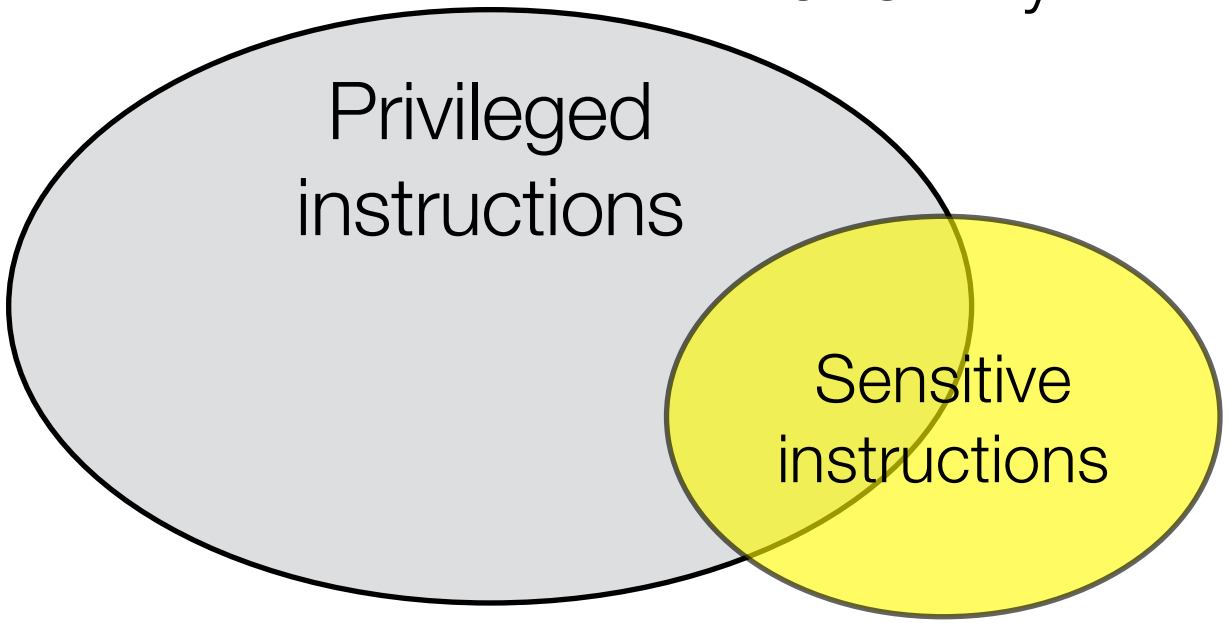
- consider Intel x86 **popf** instruction, which loads CPU flags register from contents of the stack
  考虑 Intel x86 popf 指令，它从堆栈内容加载 CPU 标志寄存器
  - if CPU in kernel mode -> all flags replaced
    如果 CPU 处于内核模式 ->所有标志均被替换
  - if CPU in user mode -> only some flags replaced, without trapping to kernel mode
    如果 CPU 处于用户模式 ->仅替换一些标志,而不会捕获到内核模式
  - **popf** is sensitive but not privileged, i.e., not virtualizable using trap-and-emulate!
    popf 是敏感的但没有特权,即不能使用 trap-and-emulate 进行虚拟化!

A hard problem

Some CPUs don't have a clean separation between privileged and non-privileged instructions 有些 CPU 没有明确区分特权指令和非特权指令

- **special instructions** not virtualizable
  特殊指令不可虚拟化

That's why VMware made $$



Intel CPUs considered not virtualizable until 1998

Three solutions

Full virtualization

完全虚拟化
模拟 +二进制翻译 :这是火箭科学,也是 VMware 所做的!

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

Para-virtualization

半虚拟化
修改客户操作系统以避免不可虚拟化的指令

- modify the guest OS to avoid non-virtualizable instructions

Hardware-assisted virtualization

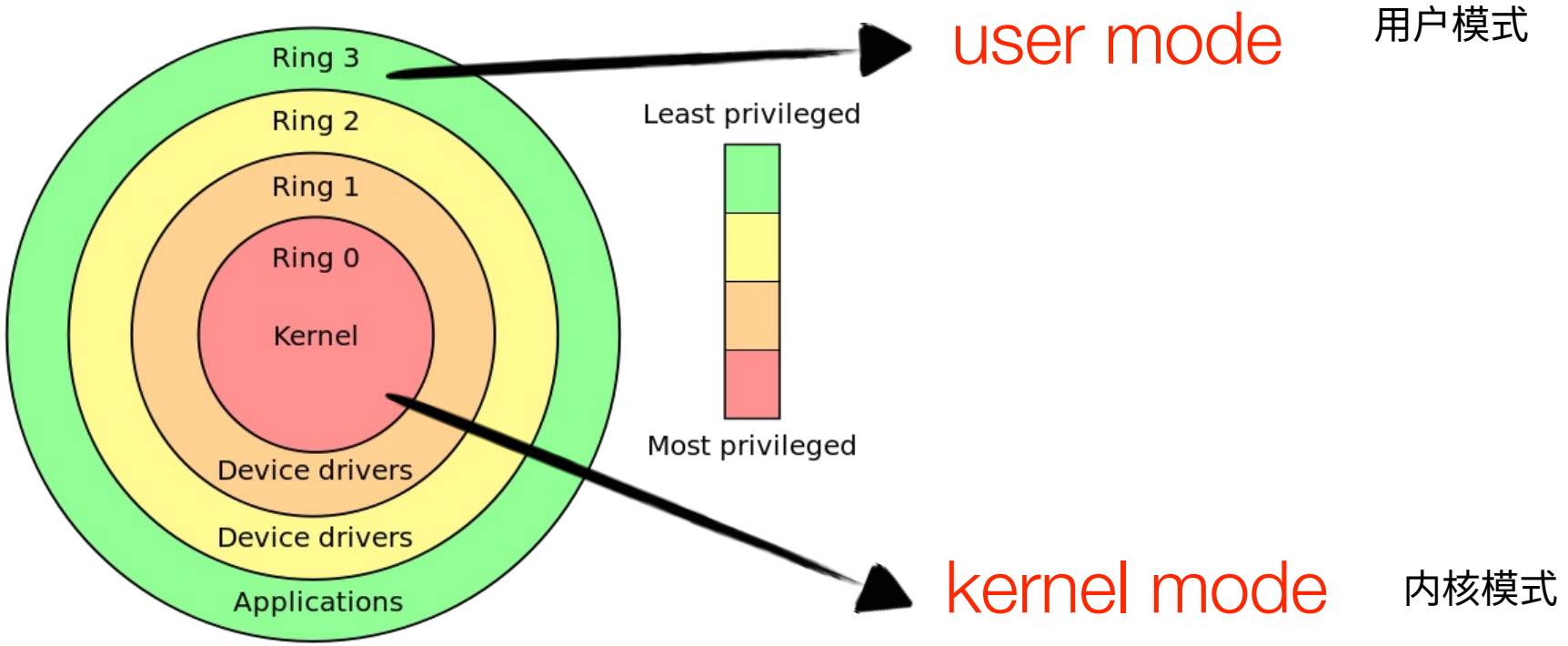
硬件辅助虚拟化
修复 CPU

- fix the CPUs

x86 protection rings

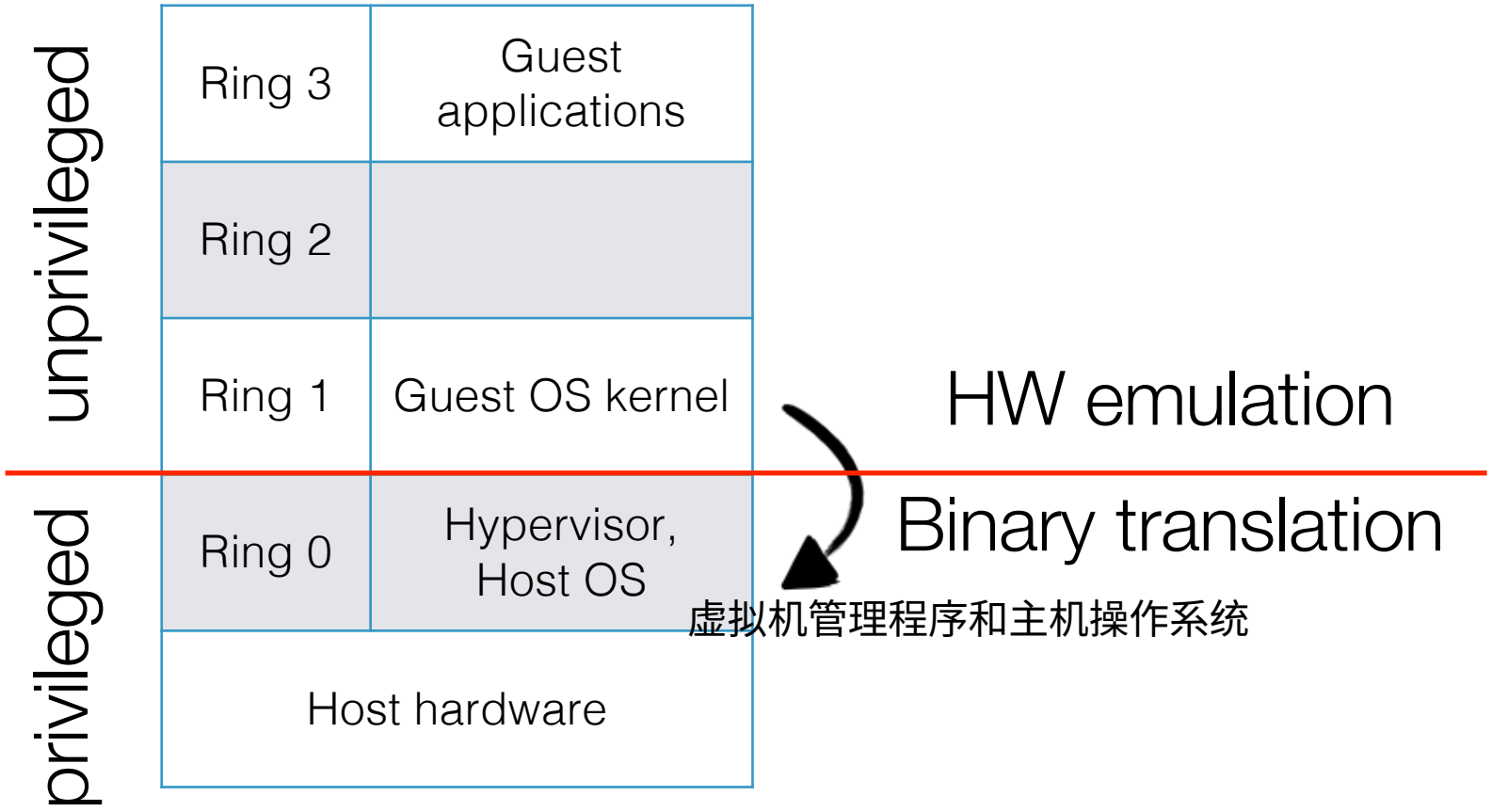
在Intel x86的硬件架构中实施：

Enforced in hardware in Intel x86 architectures



Source: Wikipedia

Full virtualization 完全虚拟化



Full virtualization 完全虚拟化

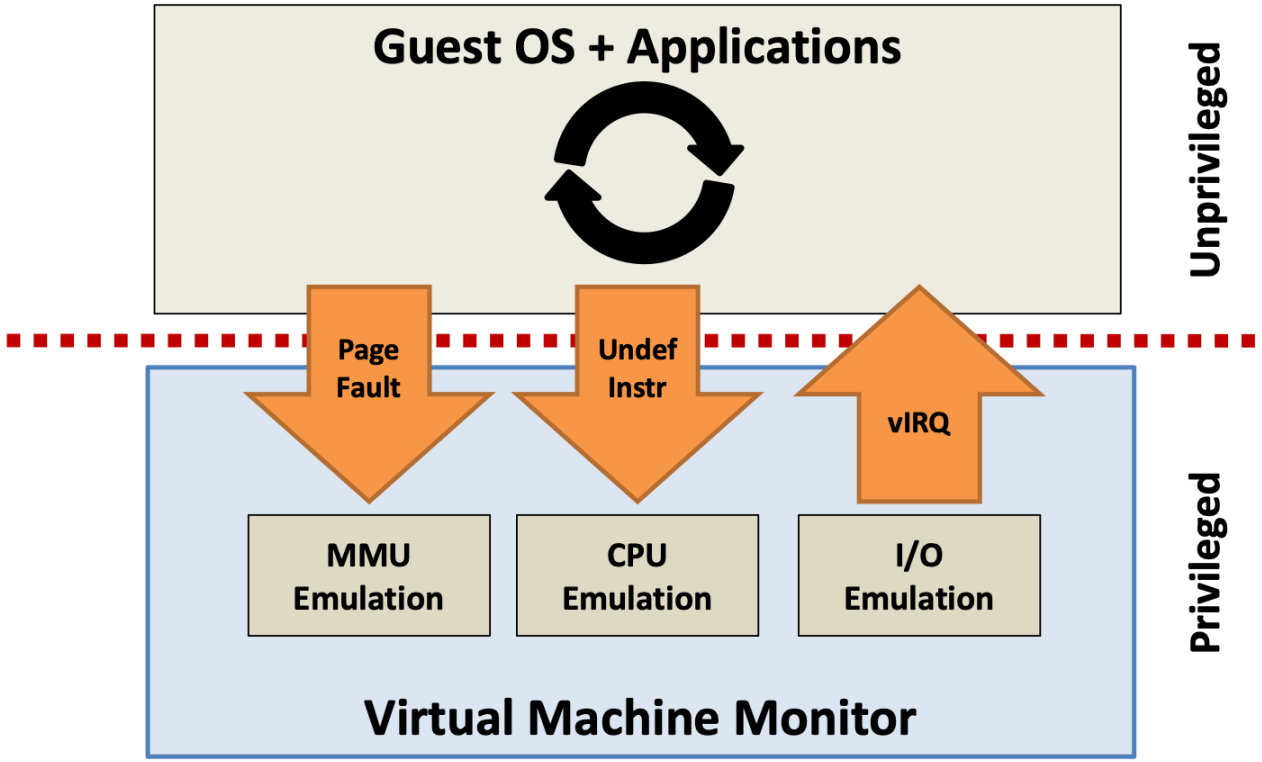
Key technique: **binary translation** 关键技术:二进制翻译

- basics are simple, but implementation rather complex
  基础简单,但实现起来相当复杂
- if guest VCPU is in user mode, guest can run instructions natively
  如果客户机 VCPU 处于用户模式,则客户机可以本地运行指令如果客户机
- if guest VCPU is in (virtual) kernel mode, hypervisor examines every instruction guest is about to execute
  如果VCPU 处于(虚拟)内核模式,则虚拟机管理程序将检查客户机即将执行的每条指令
  - non-special instructions run natively
    非特殊指令本地运行
  - special instructions translated into new set of instructions that perform equivalent task in emulated hardware
    将特殊指令翻译成执行新指令集模拟硬件中的等效任务

Full virtualization

Hardware is emulated by the hypervisor

硬件由虚拟机管理程序模拟



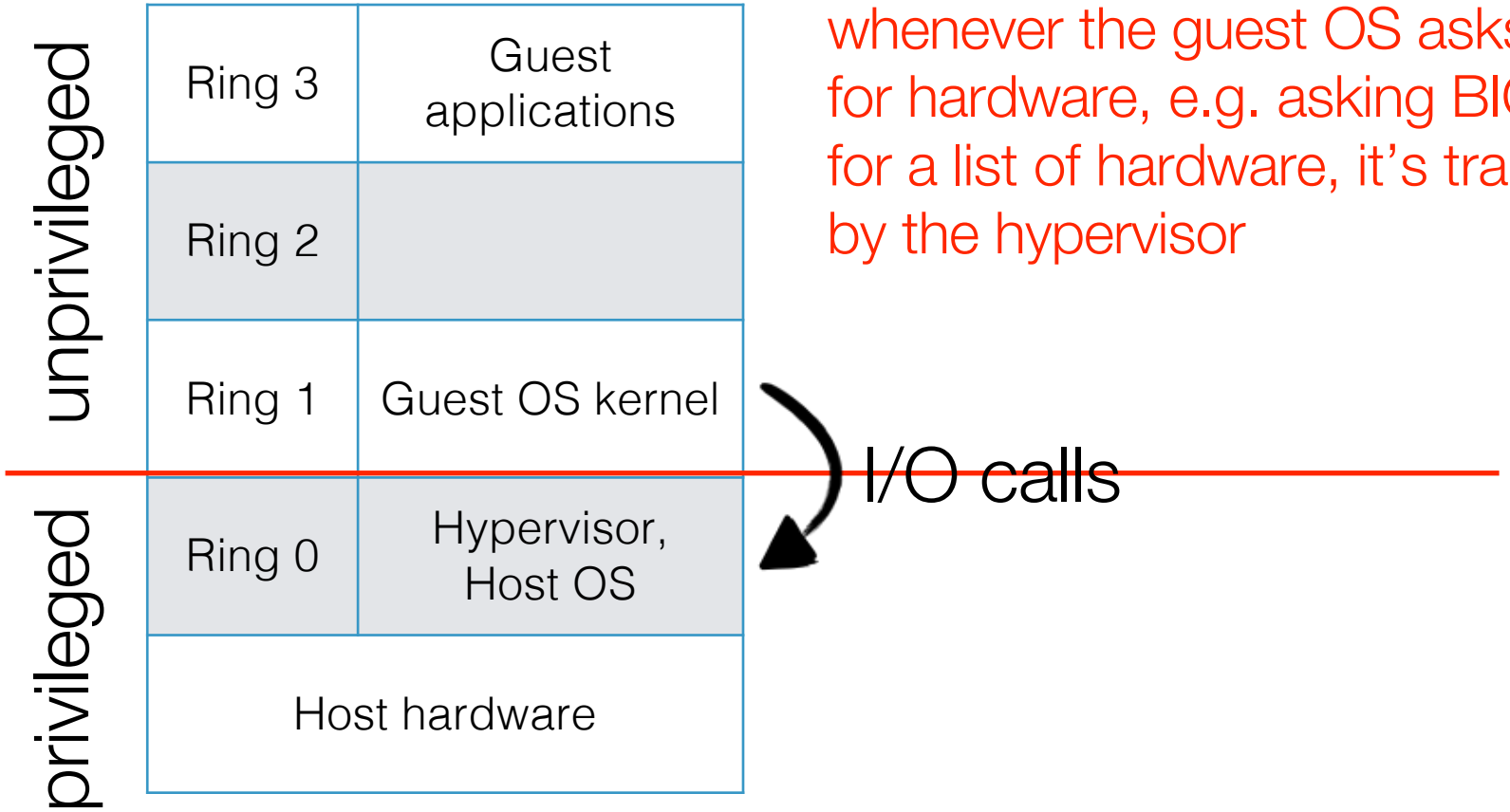
Full virtualization

The hypervisor presents a complete set of emulated hardware to the VM's guest operating system 虚拟机管理程序向虚拟机的客户操作统提供了一整套模拟硬件

- 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 无论主机系统上的实际物理硬件如何,模拟硬件保持不变

Full virtualization

Binary translation – step 1: trapping I/O calls



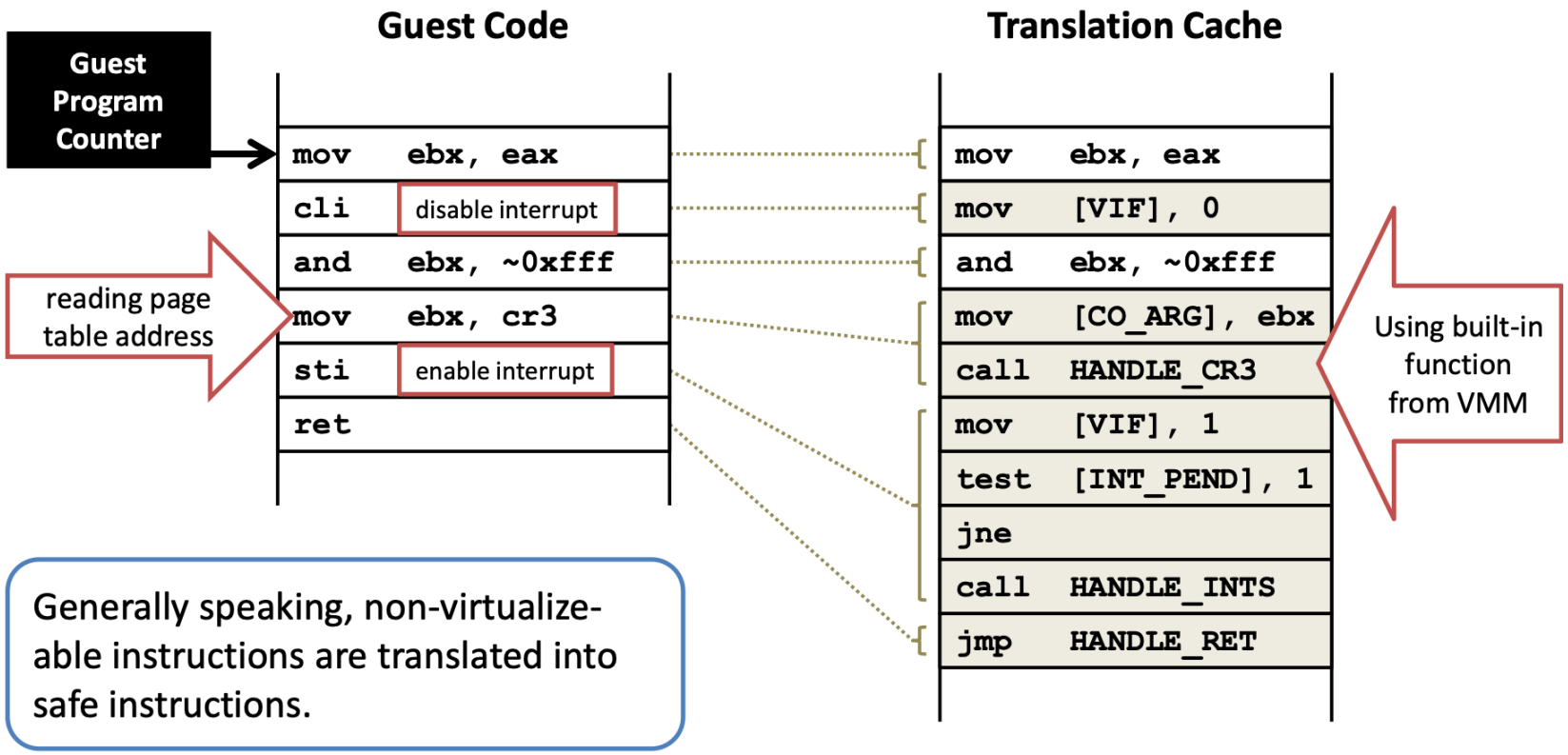
whenever the guest OS asks for hardware, e.g. asking BIOS for a list of hardware, it's trapped by the hypervisor

每当客户操作系统请求硬件时，例如向 BIOS 请求硬件列表，它都会被虚拟机管理程序捕获

Full virtualization

二进制翻译

Binary translation – step 2: emulate/translate



一般来说，不可虚拟化的指令会被翻译成安全指令。

Full virtualization

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

客户操作系统被欺骗,认为它在 Ring0 中运行特权代码

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

它实际上在主机的 Ring1 中运行,其中虚拟机管理程序模拟硬件和捕获特权代码

Unprivileged instructions are directly executed on CPU

非特权指令直接在CPU上执行

Full virtualization

Advantages: 优点:

- keeps the guest OS unmodified
  保持客户操作系统不被修改
- prevents an unstable VMs from impacting system performance
  防止不稳定的虚拟机影响系统性能
- VM portability VM 可移植性

Disadvantages: 缺点:
没有优化，,性能就不会好

- Performance is not good without optimization
  可能的解决方案:缓存特殊指令的翻译,以避免将来再次翻译它们
- possible solution: caching the translation of special instructions to avoid translating them again in the future

Para-virtualization

半虚拟化

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

开发旨在通过硬件模拟克服完全虚拟化的性能损失

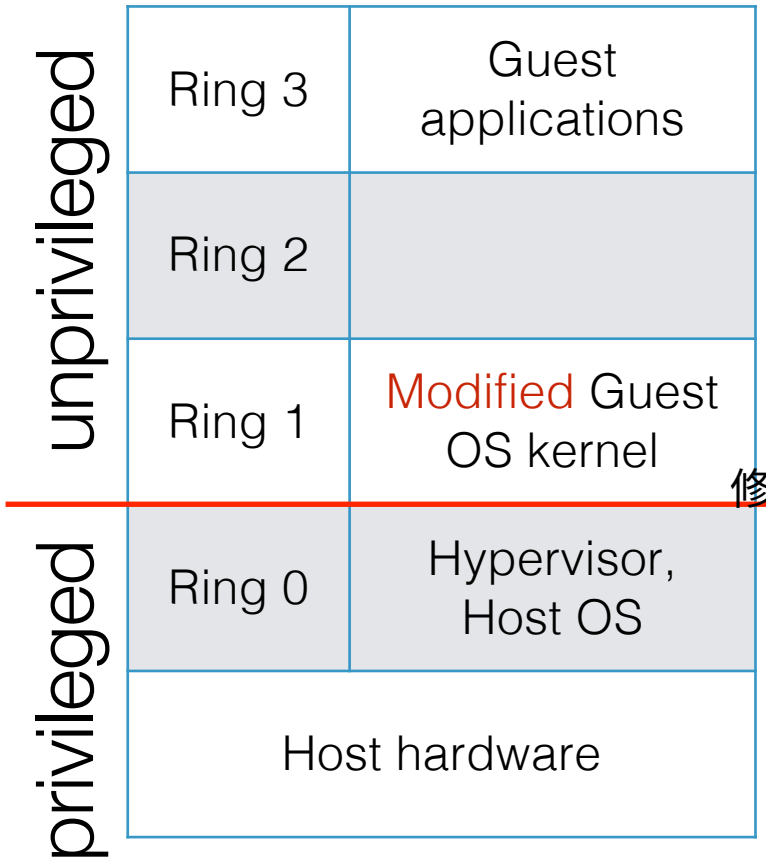
- “Para” means “besides,” “with,” or “alongside.”

“Para”的意思是“此外”、“与”或“旁边”

- the most well-known implementation is Xen

最著名的实现是 Xen

Para-virtualization



Include virtualization APIs and drivers

包括虚拟化API和驱动程序

修改的访客

无二进制翻译

No binary translation

Para-virtualization

Can be done in two ways:

可以通过两种方式完成:

- A recompiled OS kernel: easy for Linux, Windows doesn't support

  重新编译的操作系统内核:适用于 Linux,Windows
- Para-virtualization drivers for some hardware, e.g. GPU, NIC

  不支持某些硬件的半虚拟化驱动程序,例如 GPU、NIC

Para-virtualization

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

客户操作系统知道它在虚拟化环境中运行。

- it talks to the hypervisor through specialized APIs to run privileged instructions.

它通过专门的 API与虚拟机管理程序对话以运行特权指令。

- These system calls, in the guest OS, are also called “hypercalls.”

在客户操作系统中,这些系统调用也称为“ 超级调用 ’ 性能得到提升。

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

虚拟机管理程序可以专注于隔离虚拟机并进行协调。

Hardware-assisted

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

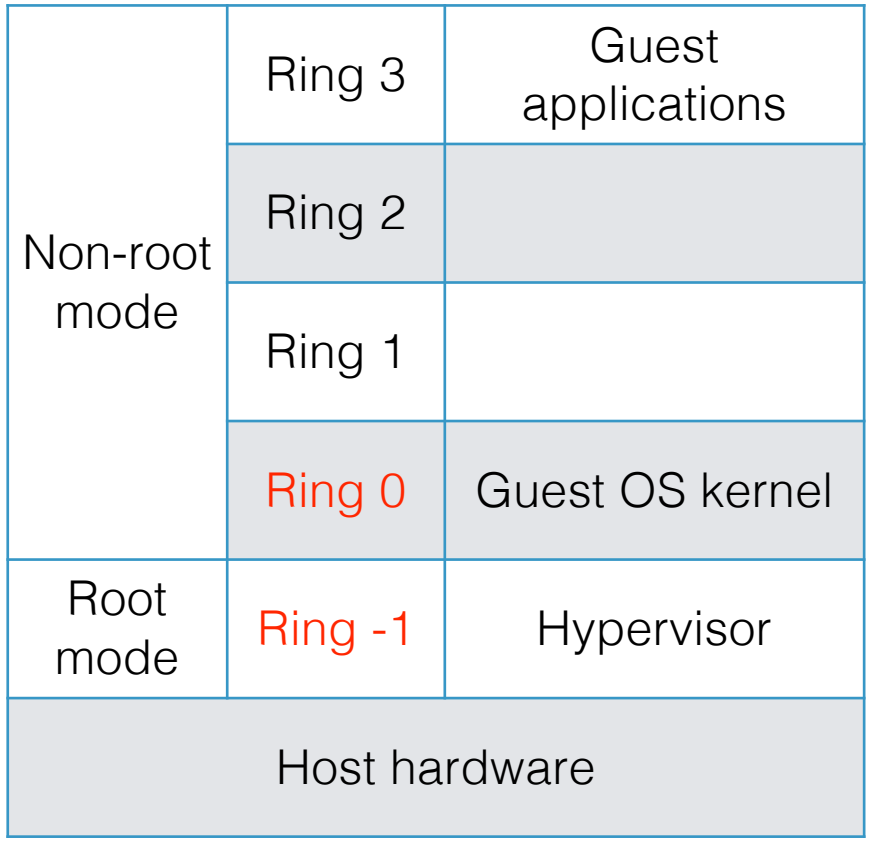
- re-implemented ideas from VM/370 virtualization support
- basically added a new CPU mode (“guest” and “host”) to distinguish VMM from guest/app

基本上增加了一种新的 CPU 模式(“ guest ”和“ host ”)来区分 VMM 和访问/应用程序

Now building a VMM is easy!

- and VMware must make money some other way…

Hardware-assisted



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. 最初,机器运行正常,没有任何客户操作系统。

<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

当虚拟机管理程序启动虚拟机时，

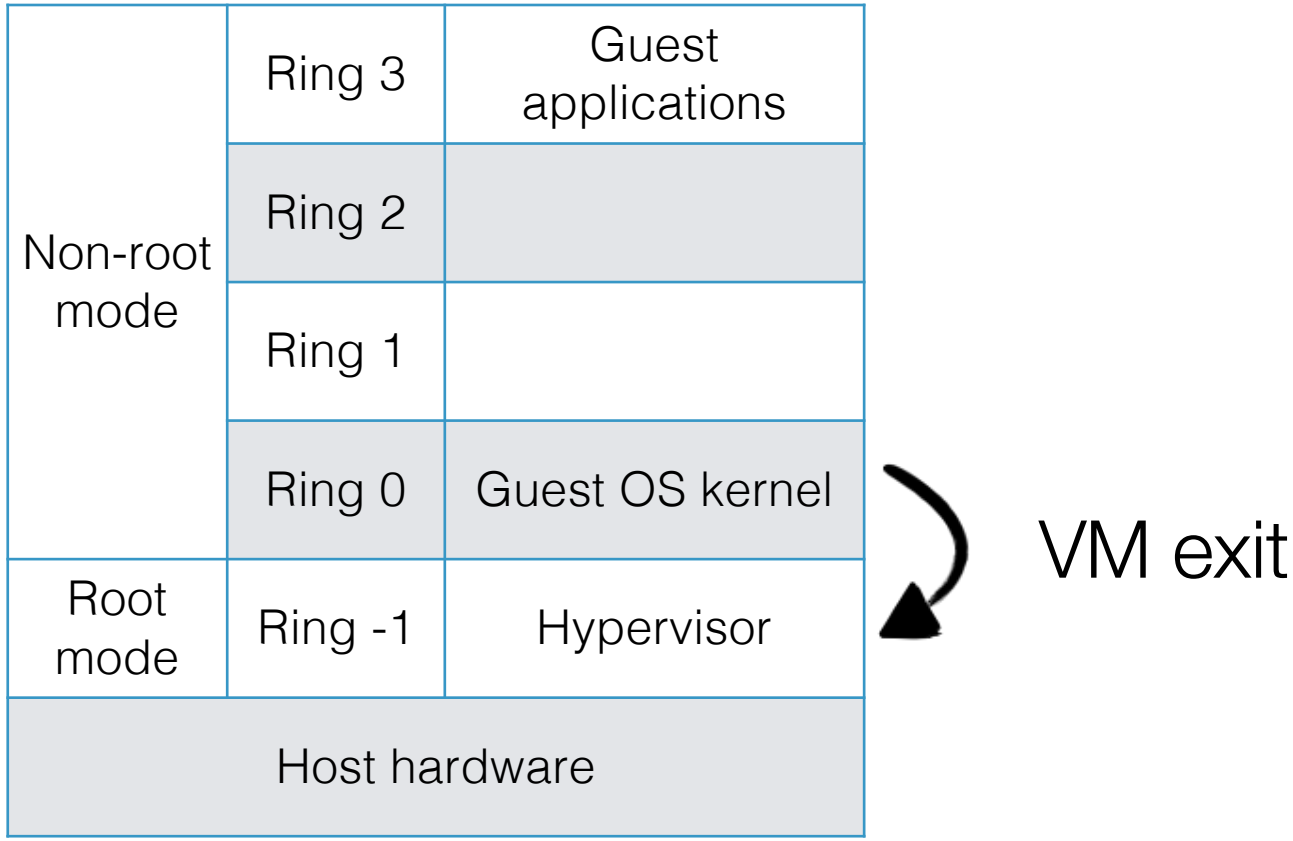
When the hypervisor launches a VM,

<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>

Hardware-assisted

当客户机运行特权指令时,它会陷入虚拟机管理程序(虚拟机退出)以行使系统控制权

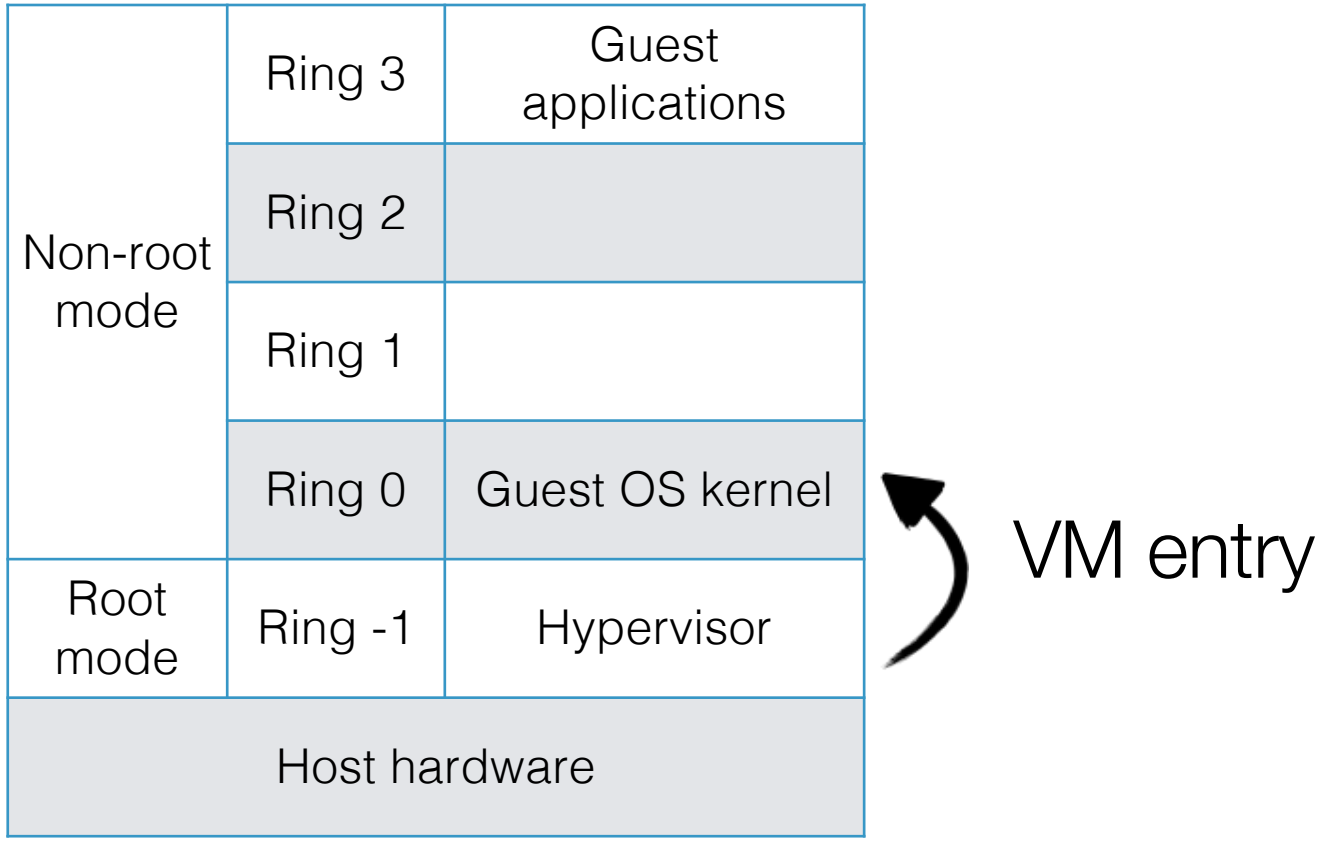
When the guest runs privileged instructions, it traps to hypervisor (VM exit) to exercise system control



Hardware-assisted

当虚拟机管理程序完成时，控制权将切换回非Root模式，VM 将继续

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



	全部	帕拉-	硬件辅助
特权操作的处理	二进制翻译	超调用	非根模式/根模式
客户操作系统修改	不	是	不
性能	好的	最好的	更好的
例子	VMware，虚拟机	辛	Xen，VMware，VirtualBox，KVM

	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

云基础设施

Cloud & Virtualization

云计算与虚拟化

Cloud computing is usually related to virtualization

- highly elastic

云计算通常与虚拟化相关
高弹性

- launching new VMs in a virtualized environment is cheap and fast

在虚拟化环境中启动新虚拟机既便宜又快速

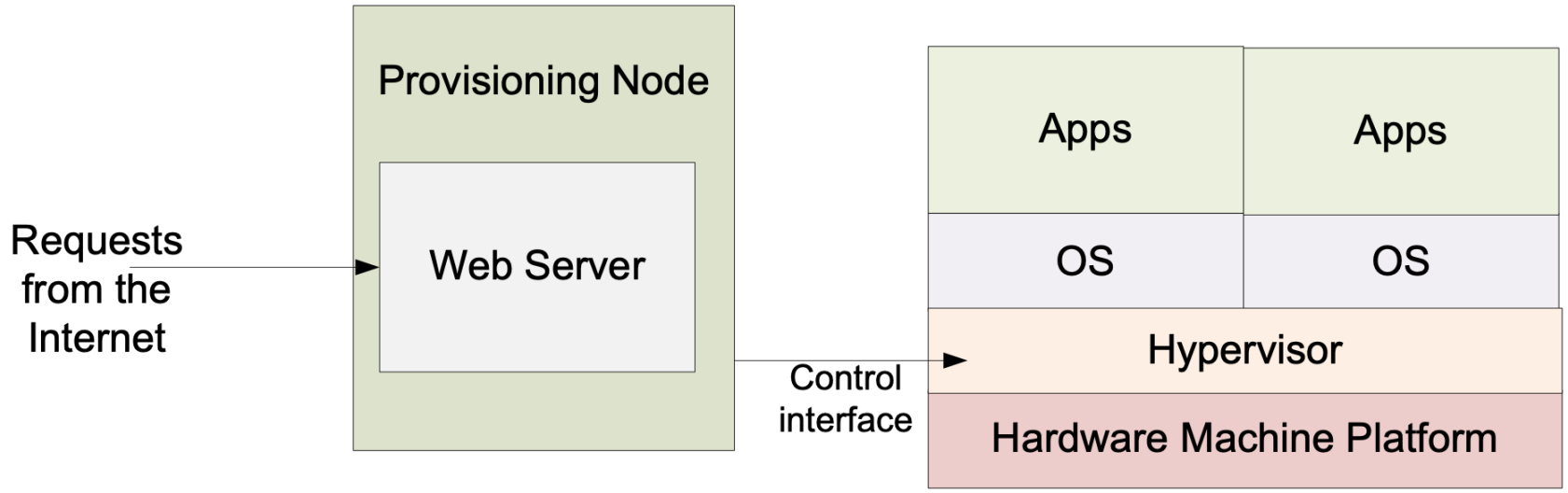
- consolidating multiple VMs onto one physical machine improves the utilization

将多个虚拟机整合到一台物理机上,提高利用率

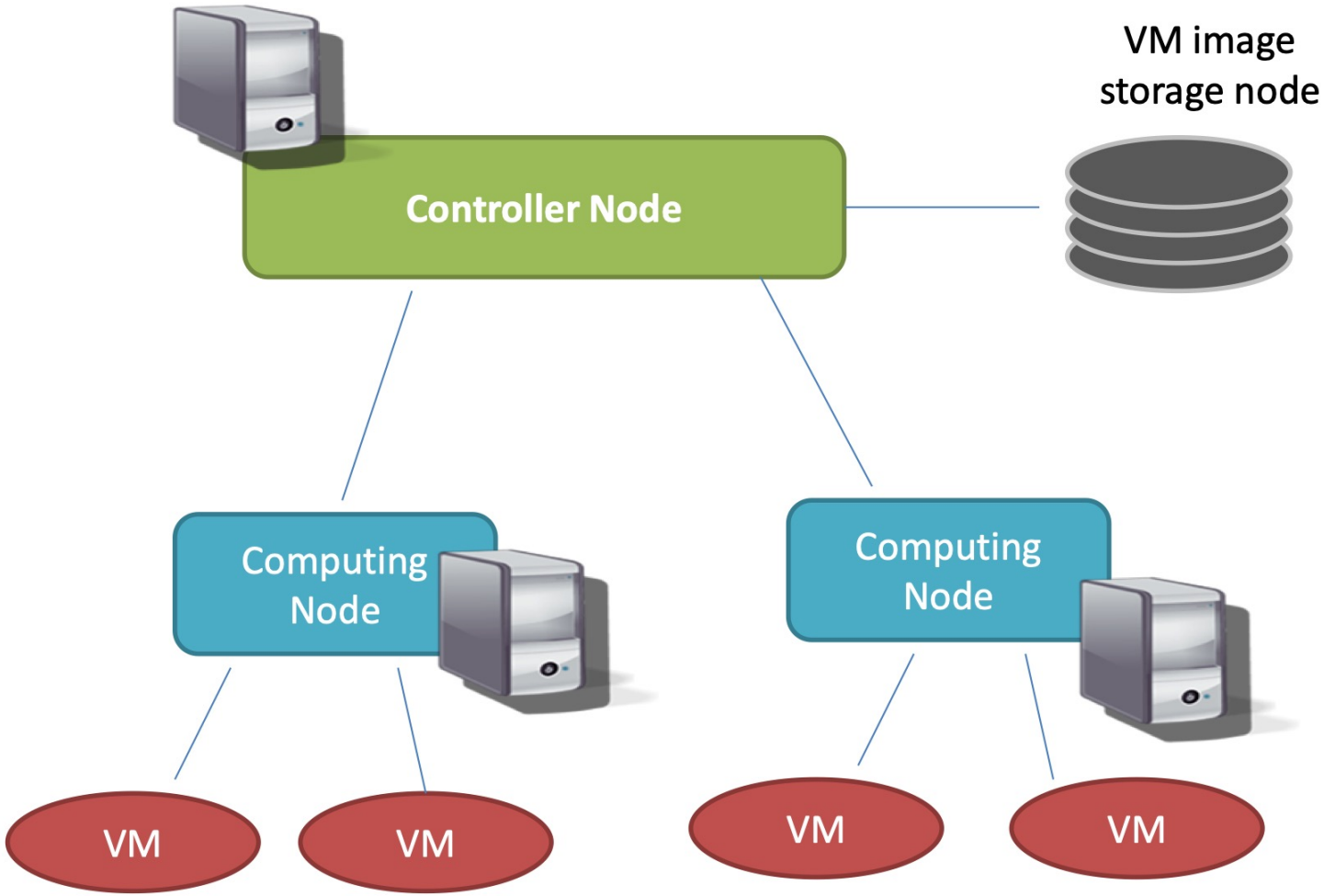
Cloud & Virtualization

云基础设施实际上是虚拟机管理基础设施

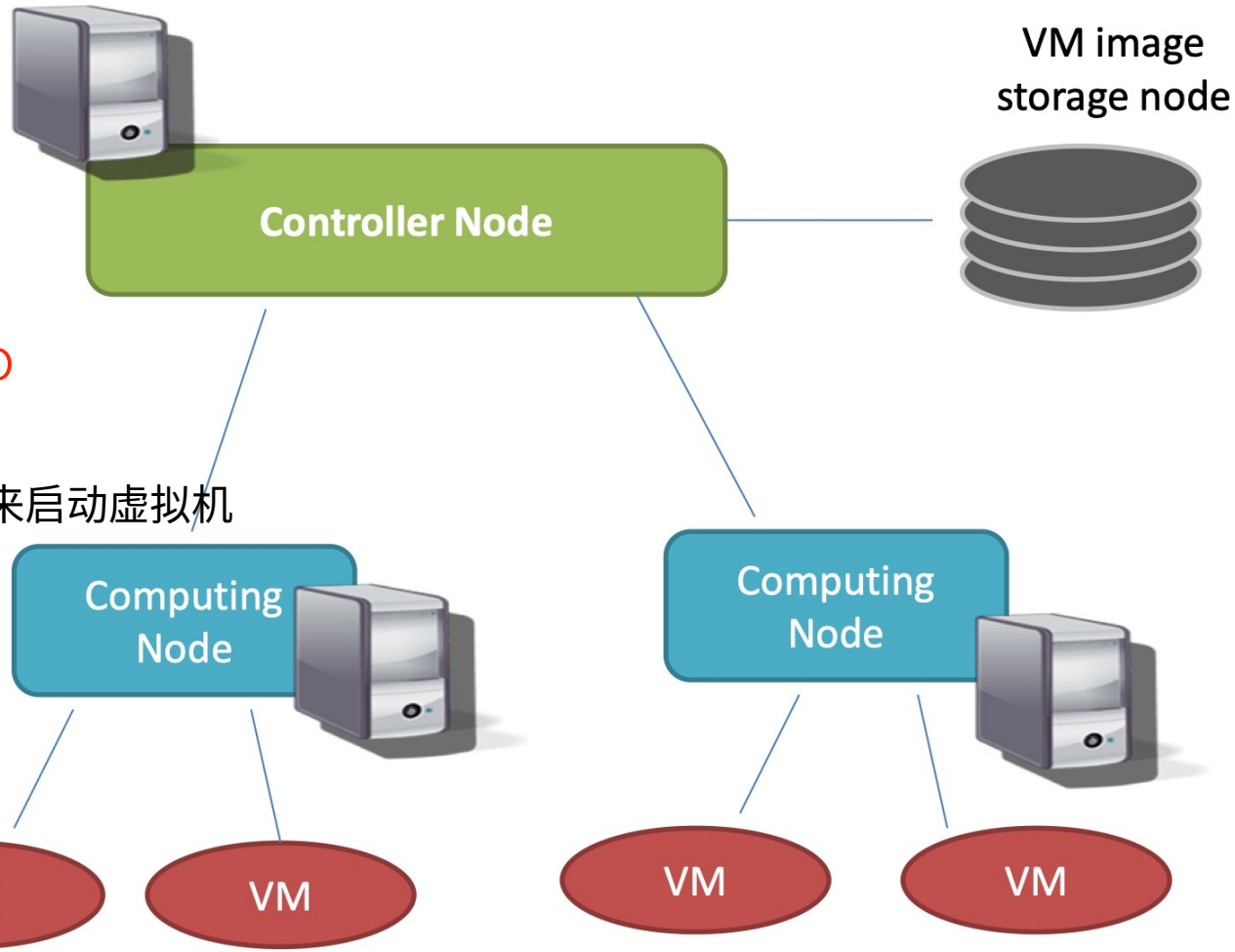
A cloud infrastructure is in fact a **VM management infrastructure**



An IaaS Cloud



IaaS – Control Flow

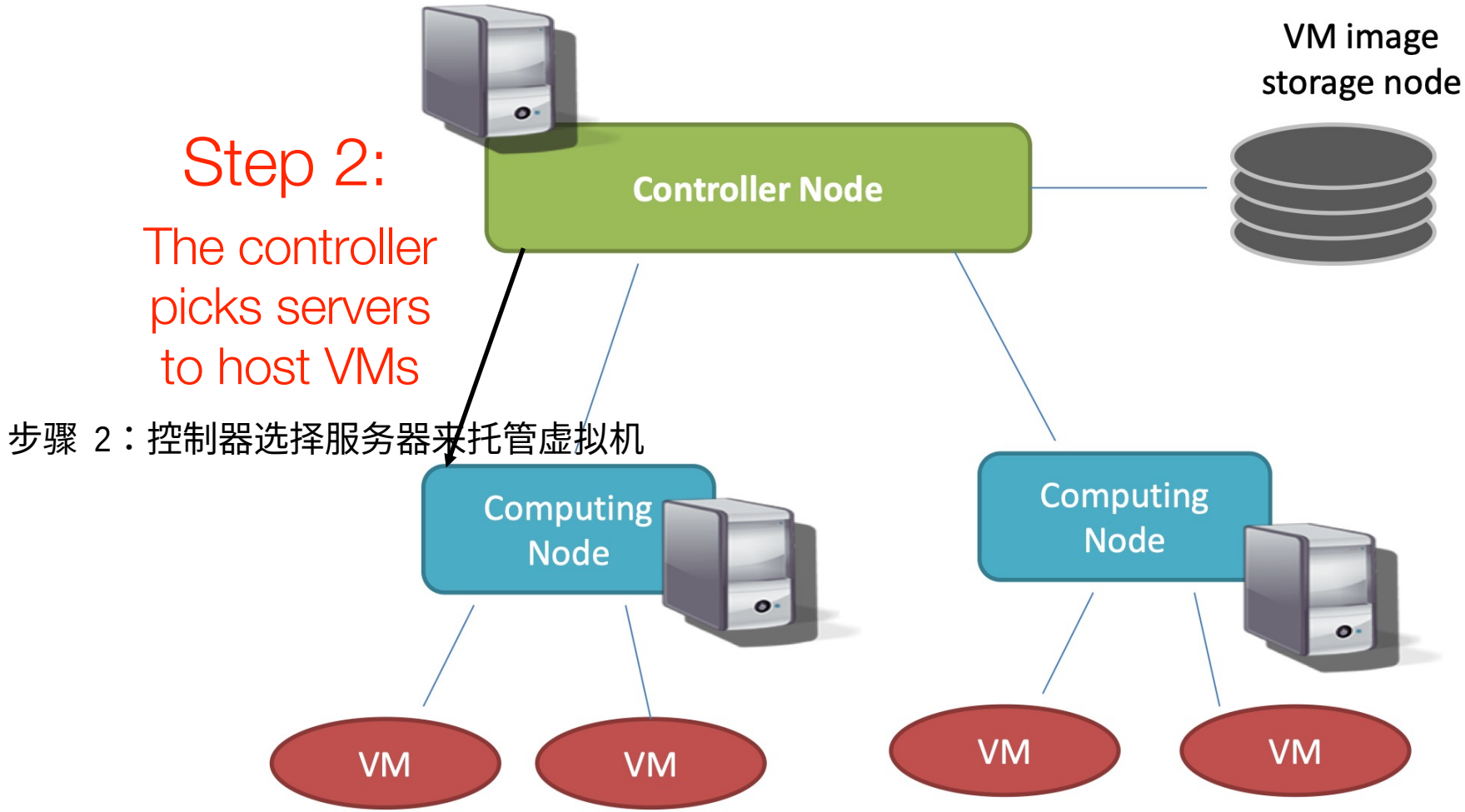


Step 1:

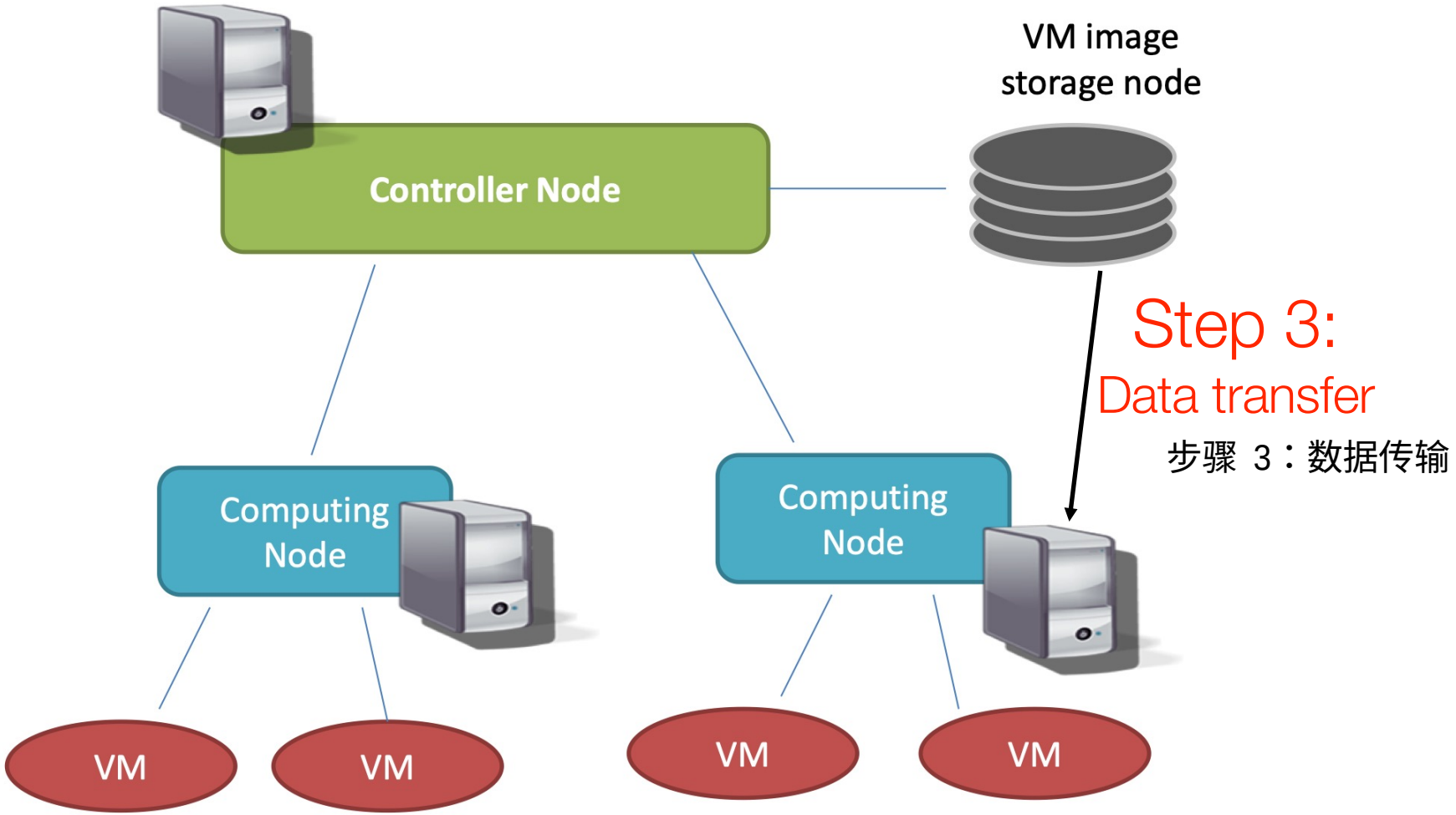
A client picks an image file to start a VM

步骤 1:客户端选择一个映像文件来启动虚拟机

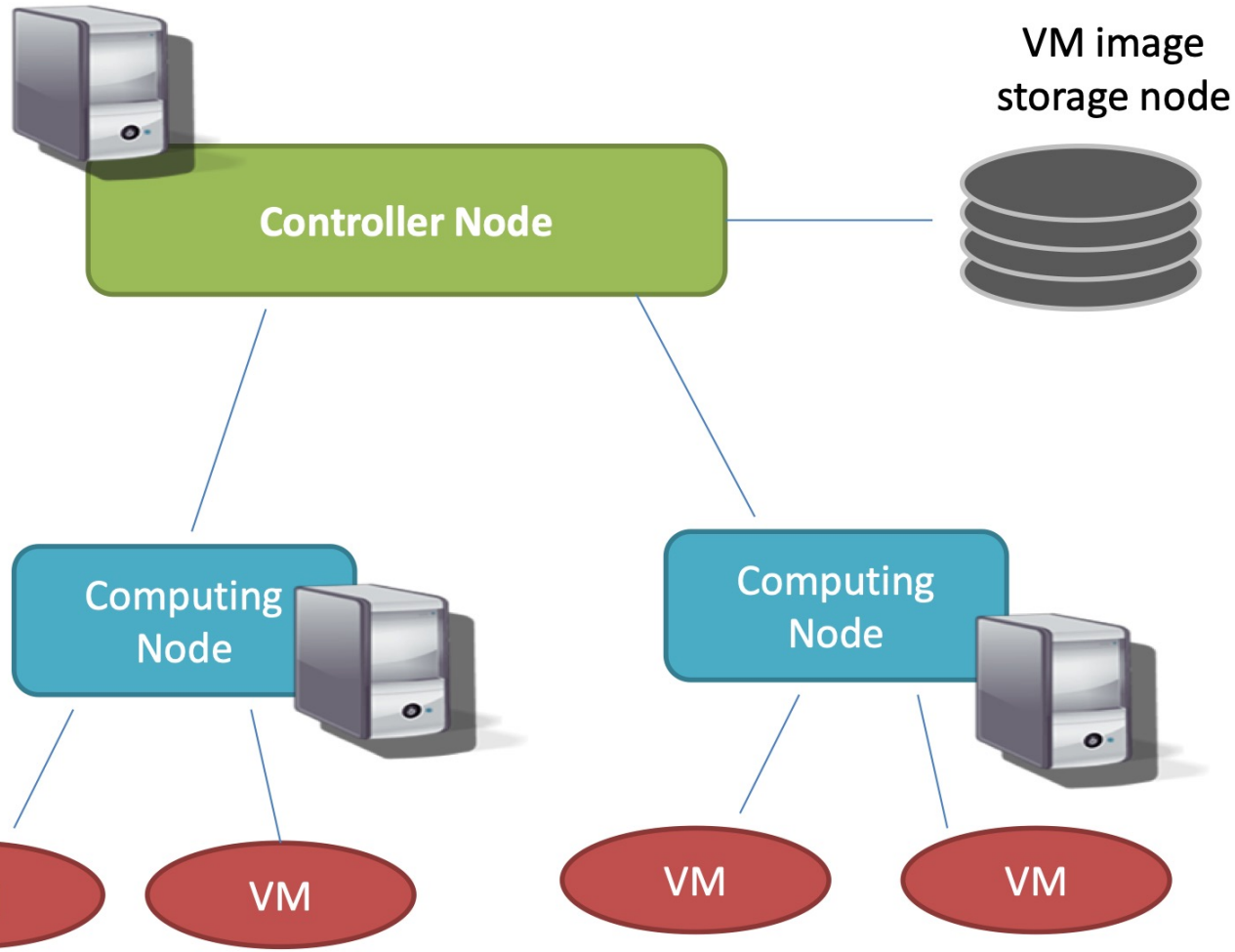
IaaS – Control Flow



IaaS – Control Flow



IaaS – Control Flow

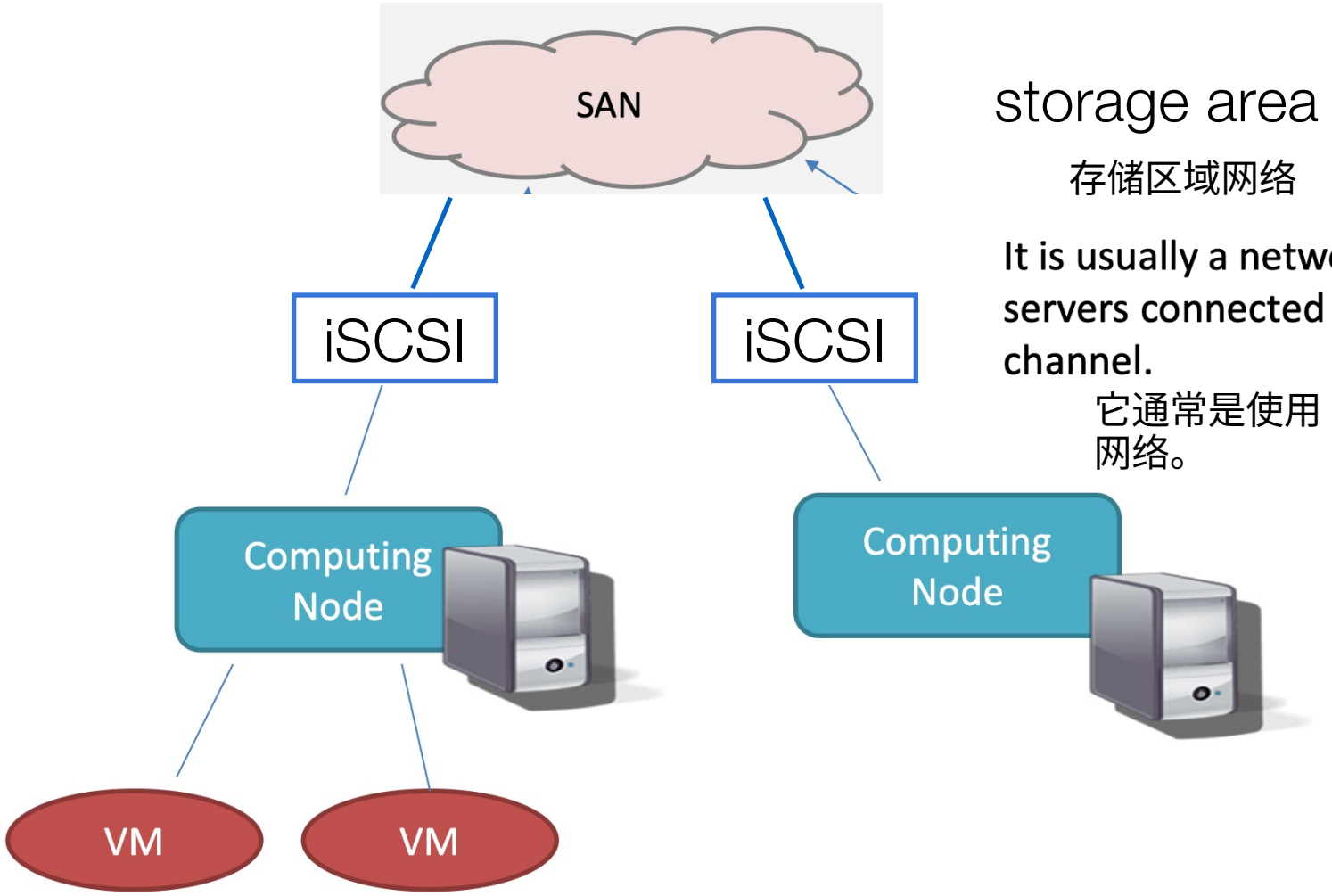


Step 4:

Start VMs

步骤 4：启动虚拟机

Subtleties



storage area network

存储区域网络

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

它通常是使用 fabricchannel 连接的存储服务器网络。

Subtleties

微妙之处在于

Virtualization used in cloud?

云中使用虚拟化吗?

- Yes for **public** cloud

  对于公共云来说,是的

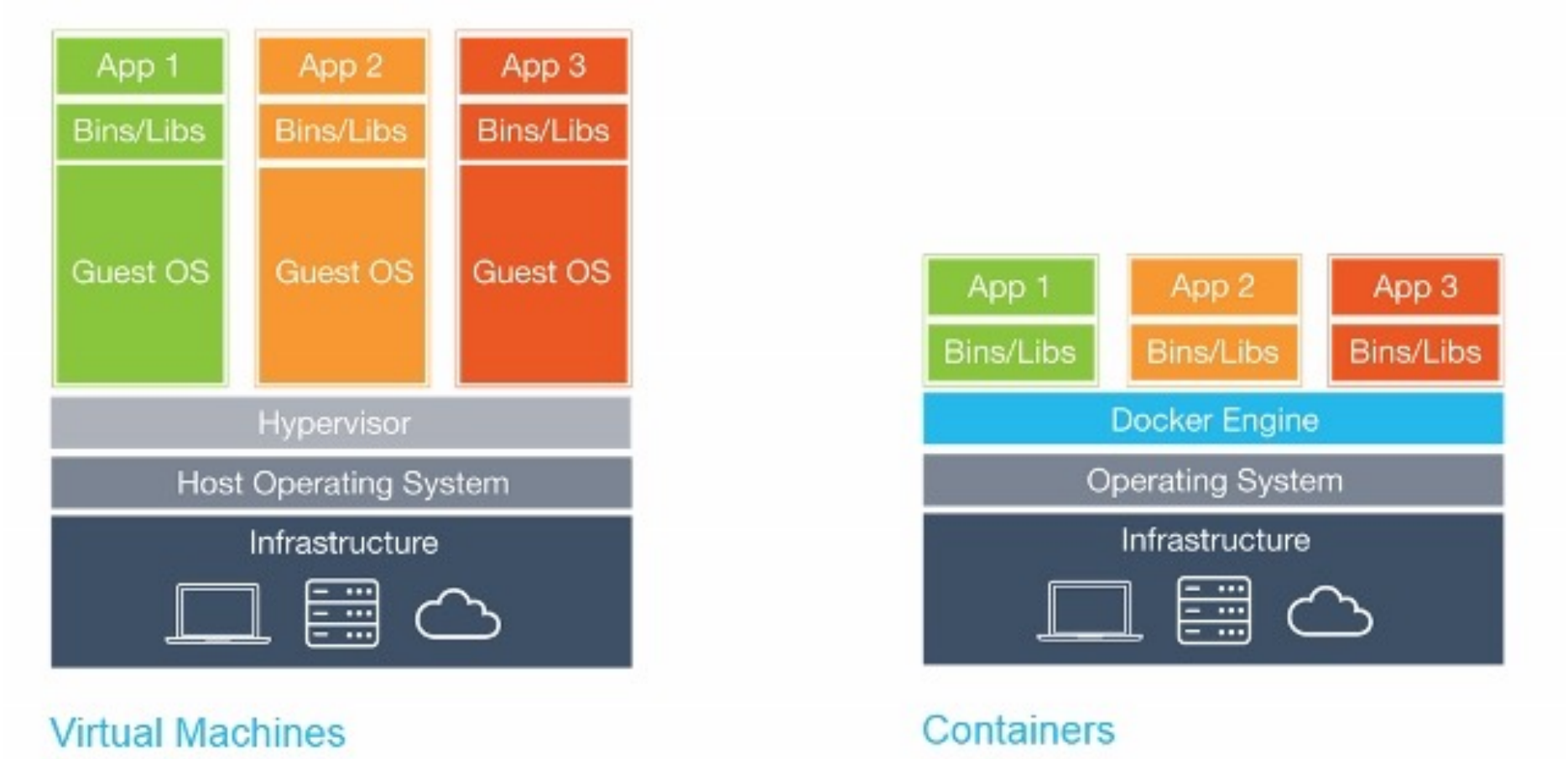
- for **private** cloud, it depends…

  对于私有云,这取决于…

  - Google's clusters are all built on top of **bare metal**: high efficiency without performance penalty

    Google 的集群全部建立在裸机之上:效率高,性能不受影响

The rise of container

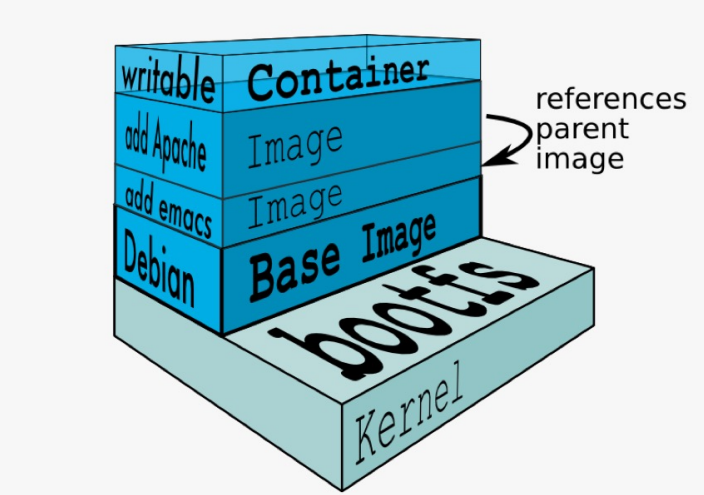


Containers

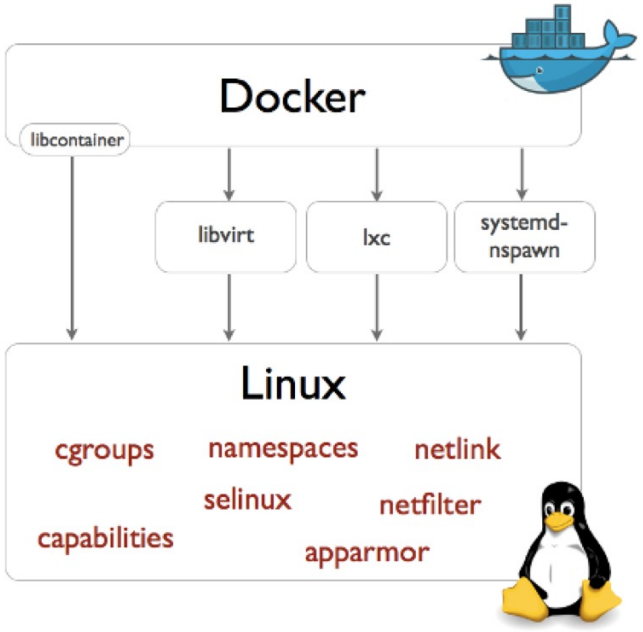
容器
基于 cgroups 和 namespaces 等内核机制的轻量级资源虚拟化
多个容器在同一个内核上运行,并让人误以为它们是唯一使用资源的容器

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

虚拟机与容器

VM system call path

VM系统调用路径

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

虚拟机内部的应用程序进行系统调用陷入虚拟机管理程序(或主机操作系统)

将陷阱返回到客户操作系统

Container virtualization system call path

容器虚拟化系统调用路径

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

容器内的应用程序进行系统调用

进入操作系统

操作系统将结果返回给应用程序

No binary translation,
no emulation

不需要二进制翻译
不需要模拟

Credits

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