



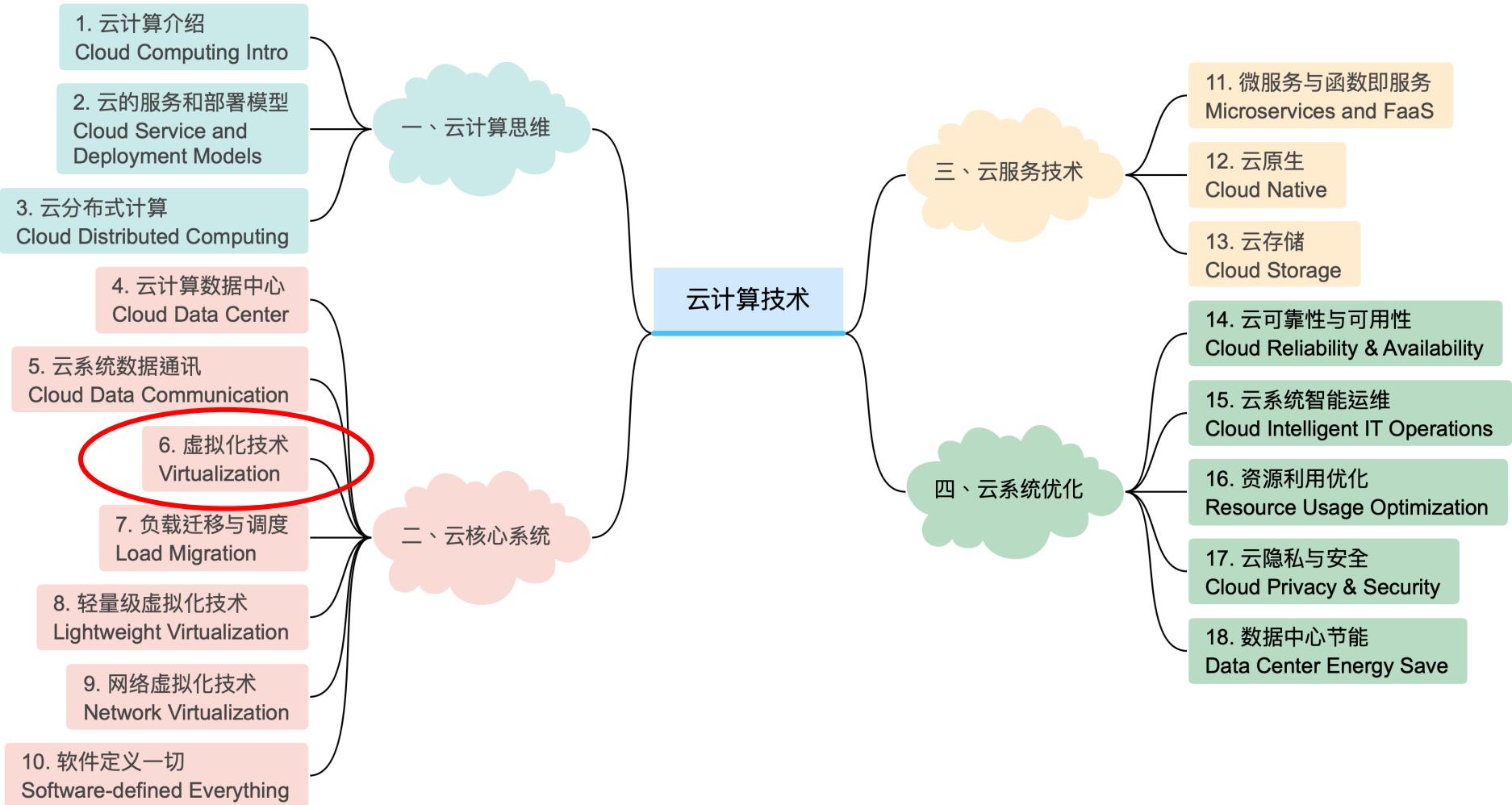
Lecture 06: 虚拟化技术

SSE316: 云计算技术
Cloud Computing Technologies

陈壮彬

软件工程学院

chenzhb36@mail.sysu.edu.cn



Today's topics

□为什么需要虚拟化技术?

□如何实现虚拟化

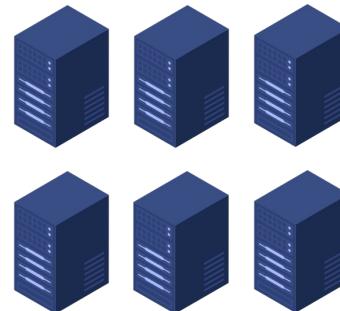
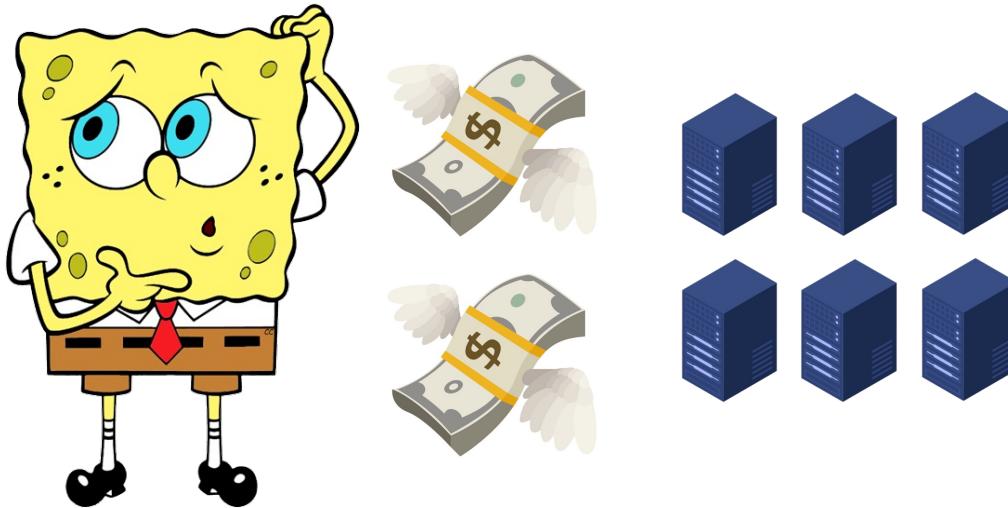
- 操作系统中的接口层次
- 可虚拟化原则

□虚拟化技术

为什么需要虚拟化技术？

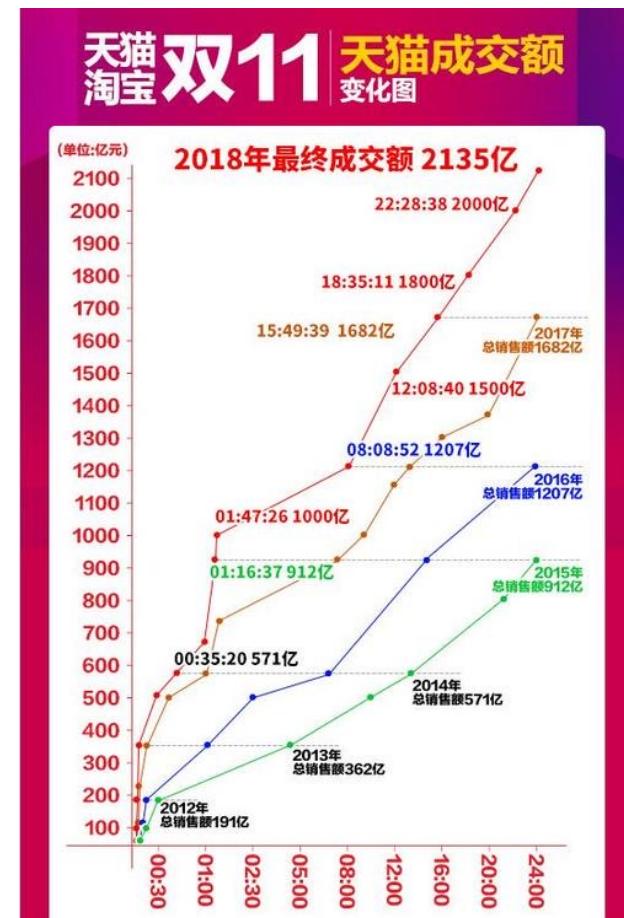
企业上云

口初创企业对外提供服务，若自己购买设备，**前期成本巨大**



口与此同时，难以应对**多变的用户需求**
口因此，**上云是一个更好的解决方案**

那云计算是如何解决这个问题的？



如何应对海量且复杂的用户需求？



分时系统

- 在同一个硬件平台上**切换不同任务**，服务不同用户
- 用户需求增加则**创建更多进程**

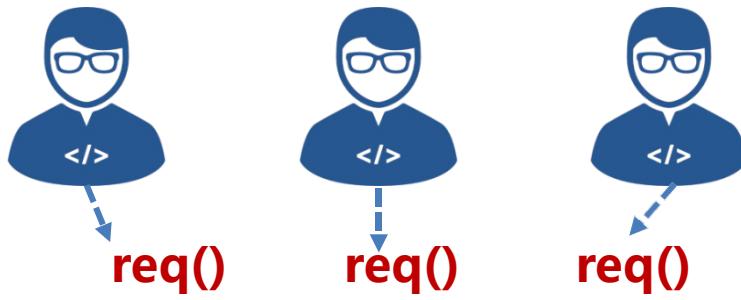
□ 缺点

- 任务、资源调度复杂
- 硬件资源不能充分利用
- 隔离性和安全性不足
- 规模和性能受硬件平台限制



虚拟化技术

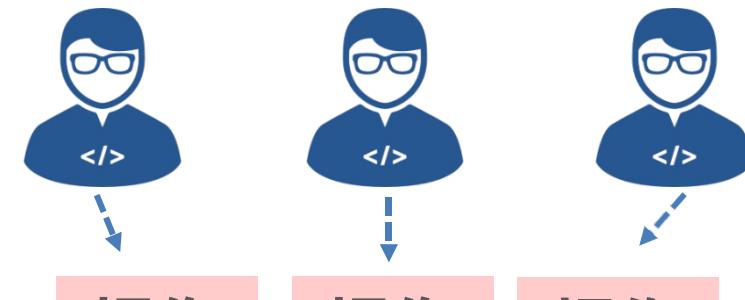
从并发任务处理到并发执行环境



操作系统

系统硬件

一台计算机可以为多个用户提供并发任务处理



操作系统

操作系统

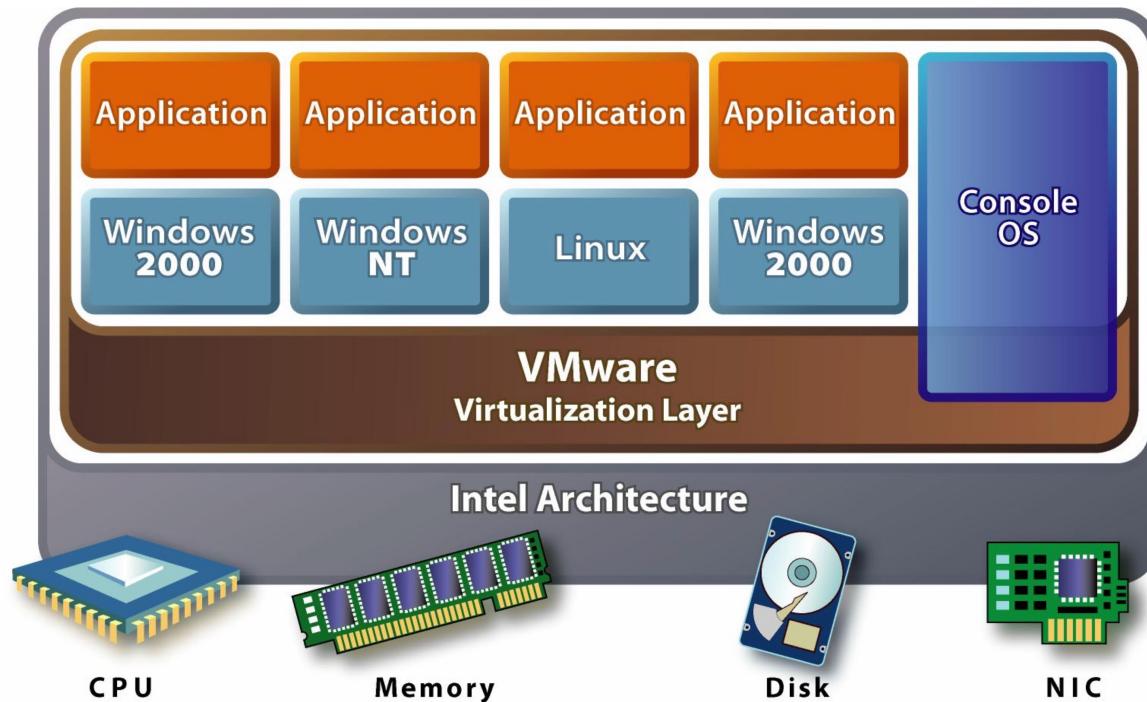
操作系统

系统硬件

一台计算机为多个用户提供独立的操作环境

什么是虚拟化?

系统虚拟化技术能在一台物理主机上创建多个虚拟机
从程序的角度来看，虚拟机和真实的物理主机没有区别



随意点击即可
添加任意硬件!

用软件模拟硬件!

什么是虚拟化?

1

引入一个新的软件层

2

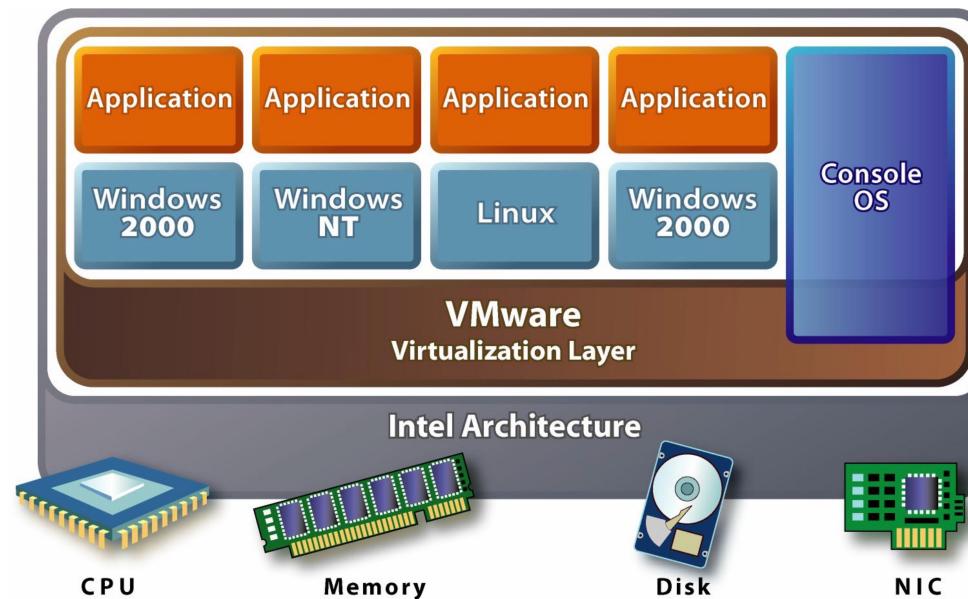
上层是操作系统（虚拟机）

3

底层硬件与上层软件解耦

4

上层软件可以在不同硬件之间切换



虚拟化的优势：整合服务器

□ 传统数据中心服务器资源利用率低

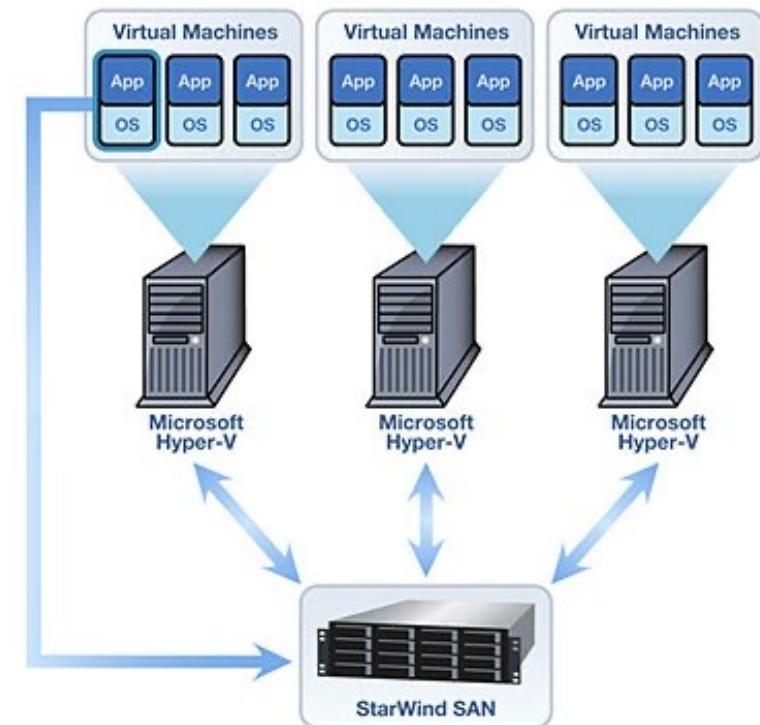
- CPU 平均利用率仅为 **约 20%**

□ 提高物理机资源利用率

- 一个物理机上整合多个虚拟机

□ 降低云提供商成本

- 基于用户**错峰使用**特性，"超售"系统资源



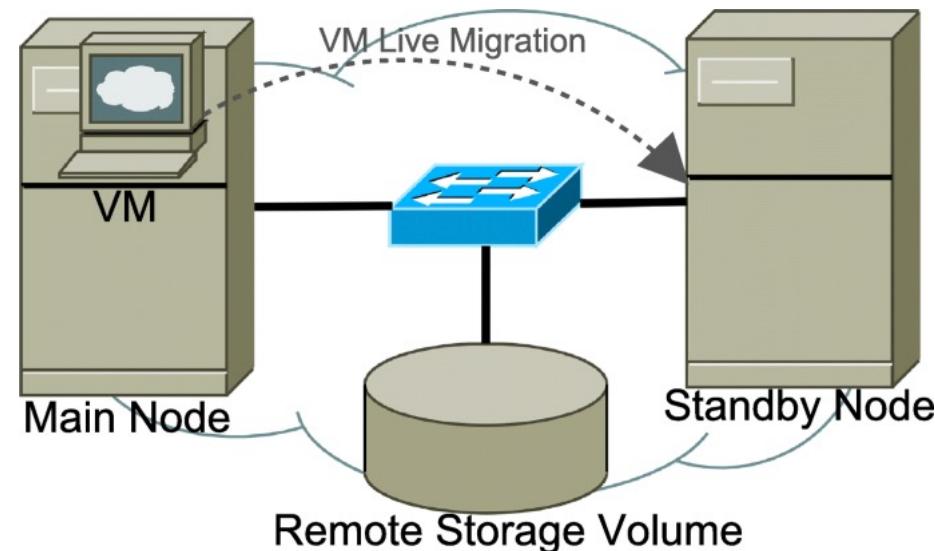
虚拟化的优势：简化服务器管理

□ 通过软件接口管理虚拟机

- 创建、复制、备份、销毁

□ 虚拟机热迁移

- 无须停机或重启
- 便于虚拟机升级和维护
- 实现全局**负载均衡**

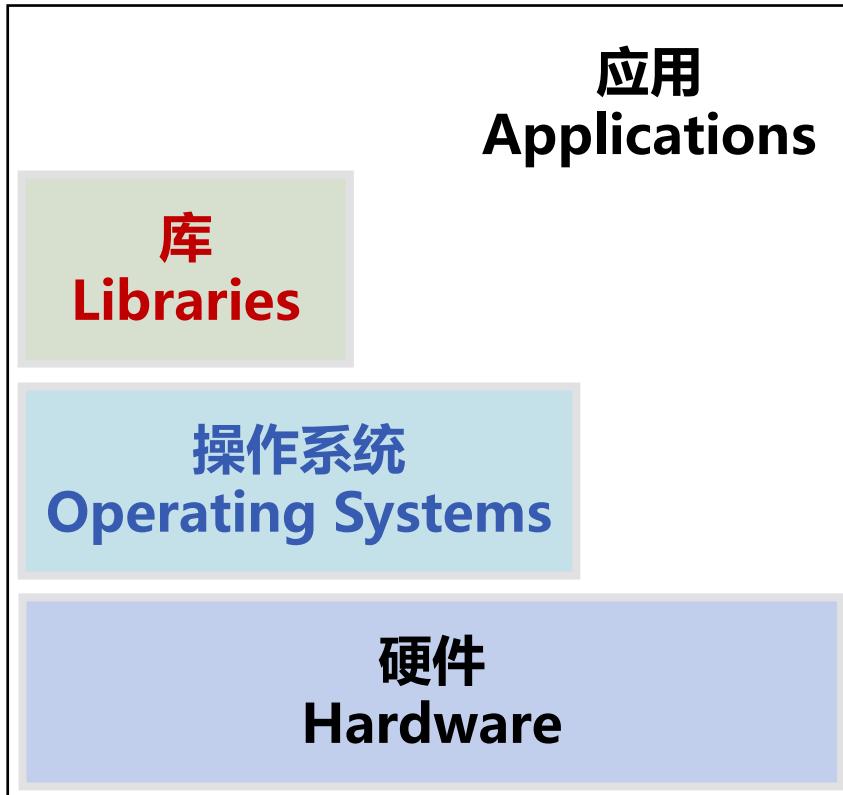


如何实现虚拟化？

操作系统中的接口层次 – 提供抽象

口定义了如何在计算机系统上高效开发和运行软件

- 硬件和软件
- 操作系统和应用程序
- 库/服务和程序员

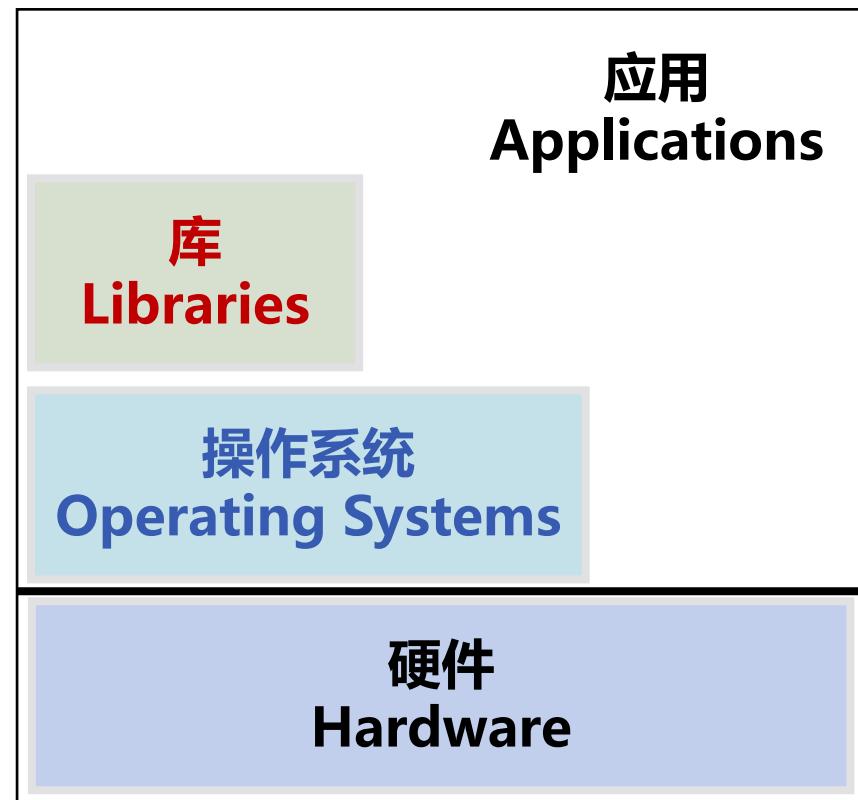


通过一个“代理”实现
软件和硬件之间的交
互和通信

操作系统中的接口层次

ISA (Instruction Set Architecture) 指令集架构层

- 硬件和软件之间的接口，无需关心硬件实现，如汇编语言
- 定义了处理器支持的
 - 数据类型
 - 寄存器
 - 指令集
 - 地址模式等
- 不同架构 CPU 的指令集不同
 - x86
 - ARM
 - RISC
 - MIPS



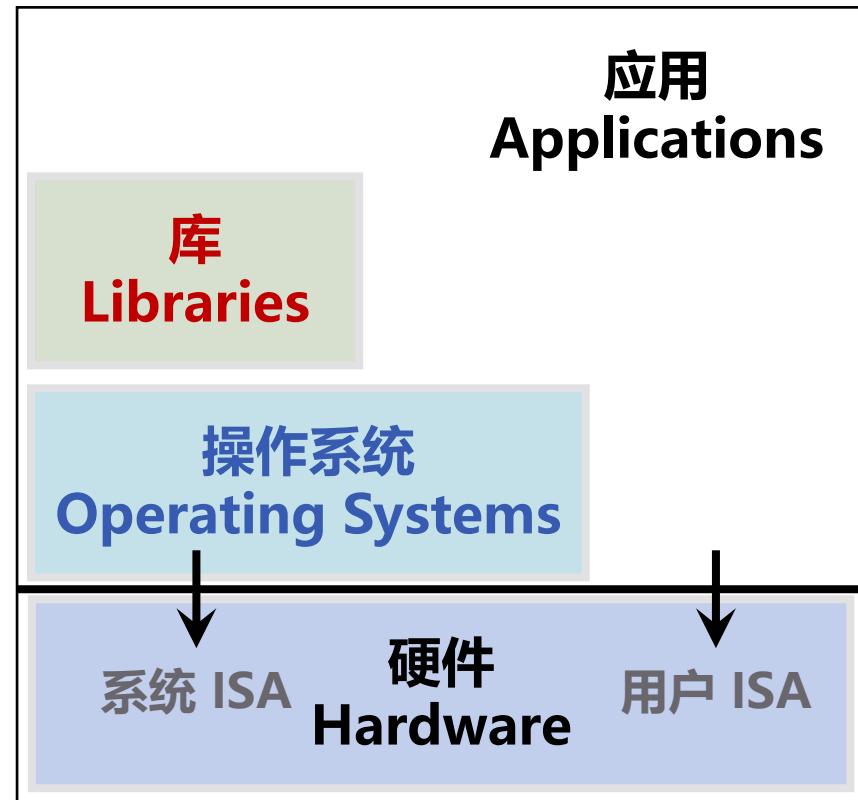
操作系统中的接口层次

□ 用户 ISA (处理器大部分指令集)

- 决定如何编写程序
- 用户态和内核态程序均可用
- 算数运算、逻辑运算、数据传输
- `mov x0, sp`
- `add x0, x0, #1`

□ 系统 ISA (处理器小部分指令集)

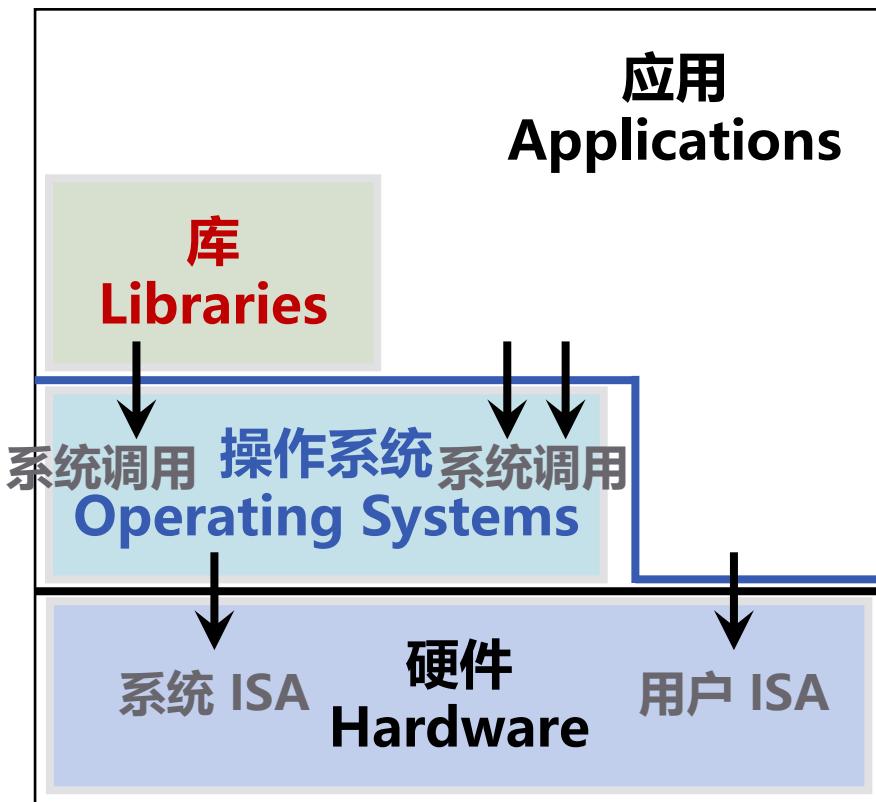
- 决定如何编写操作系统
- 仅内核态程序可用
- 中断处理、内存管理、设备控制
- `msr vbar_el1, x0`



操作系统中的接口层次

□ ABI (Application Binary Interface) 应用二进制接口层

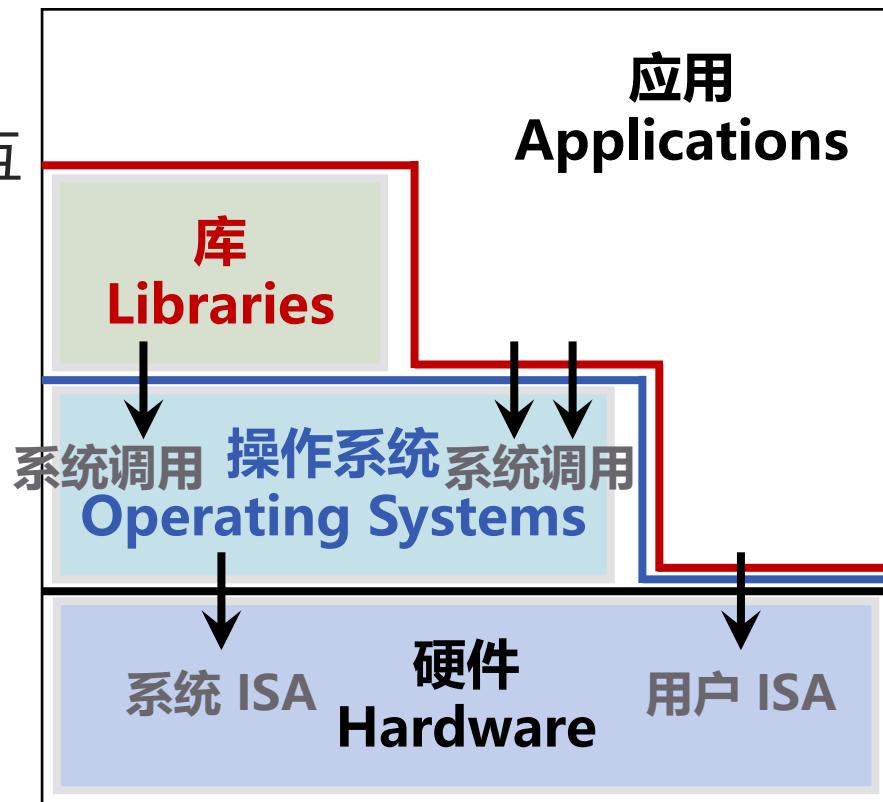
- 操作系统和应用程序之间的接口，无需关心操作系统细节
- 包含用户 ISA 和系统调用
- 定义了
 - 数据类型的大小和对齐
 - 函数调用的约定
 - 系统库和对象文件的格式
 - ...



操作系统中的接口层次

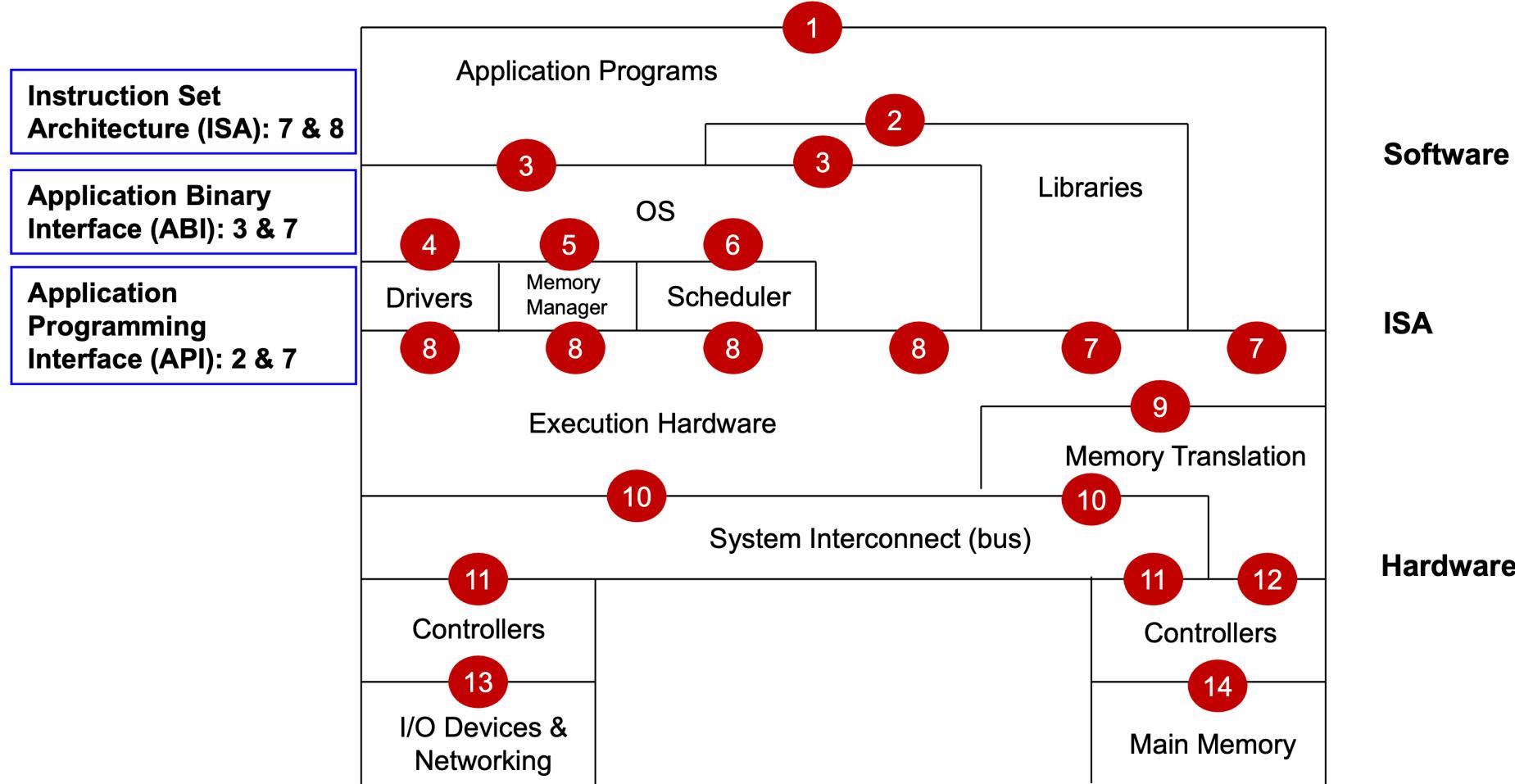
□ API (Application Programming Interface) 层

- 用户态库或服务提供给程序员的接口，无需关心函数实现细节
- 用于让不同的软件系统、应用程序或组件之间进行通信和交互
- 包含库的接口和用户 ISA
- 便于程序员快速开发软件



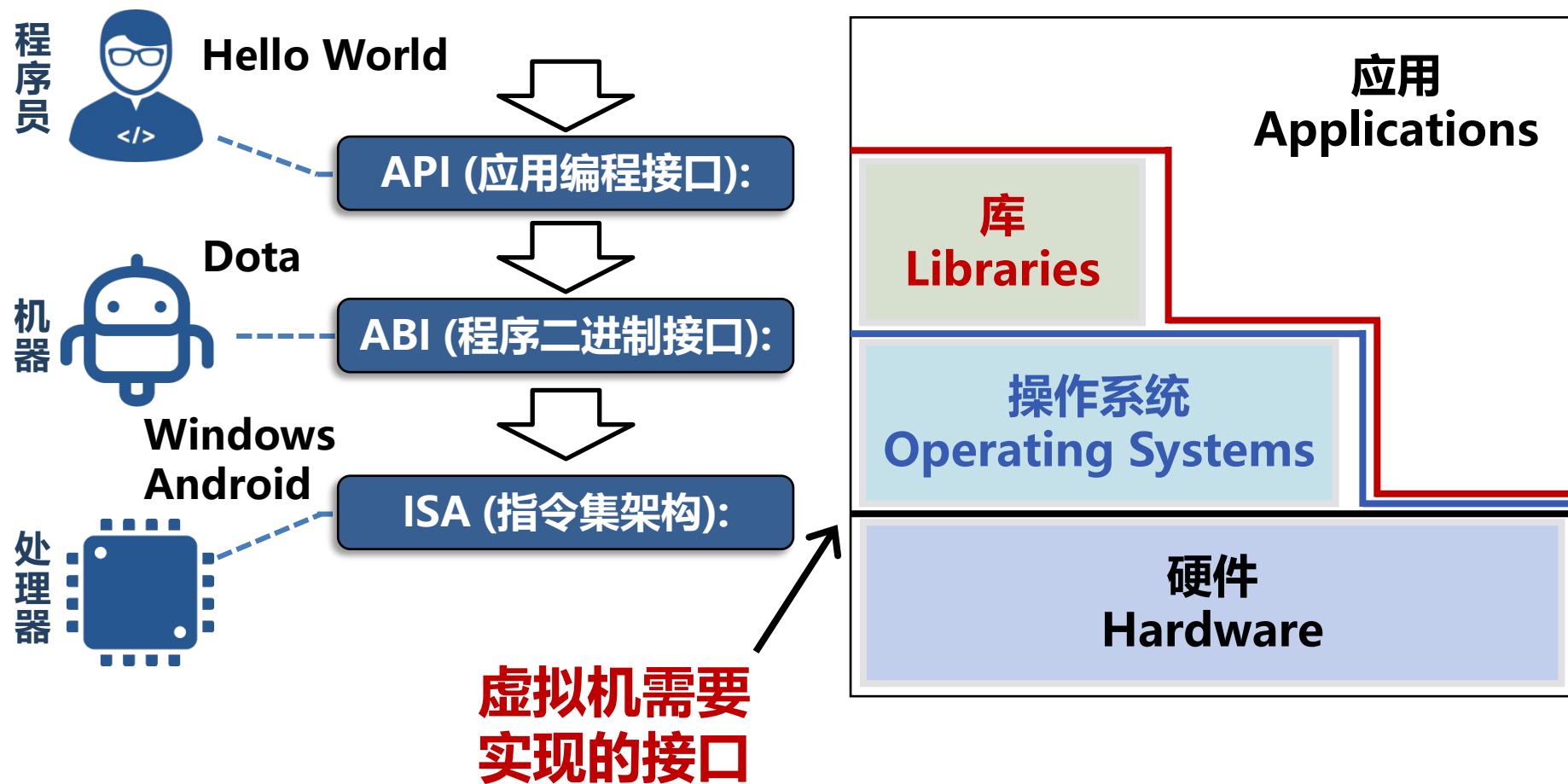
操作系统中的接口层次

多样化的接口提供了不同的执行环境视角，丰富了对“机器”的定义



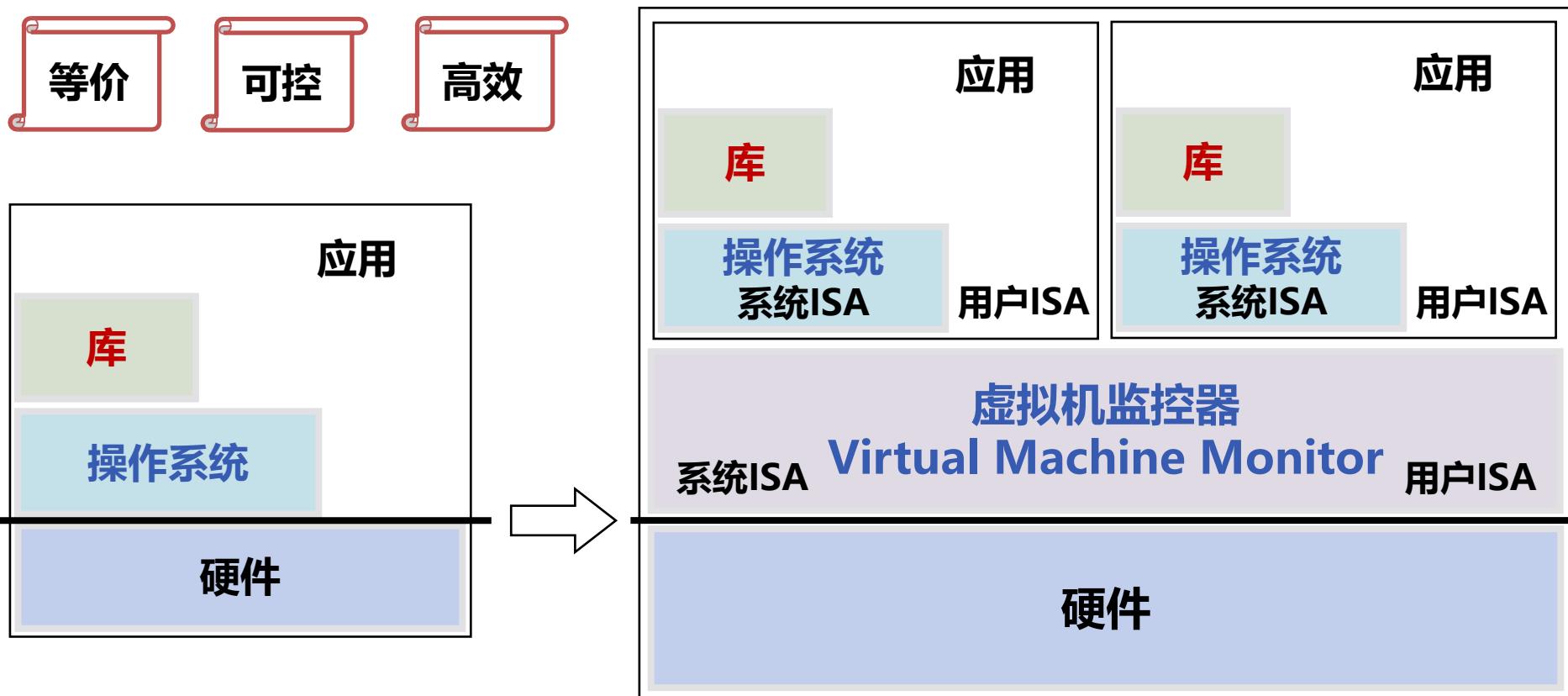
不同接口层次面对的对象

关键资源下沉，逐层通过 **接口（Interface）** 交互



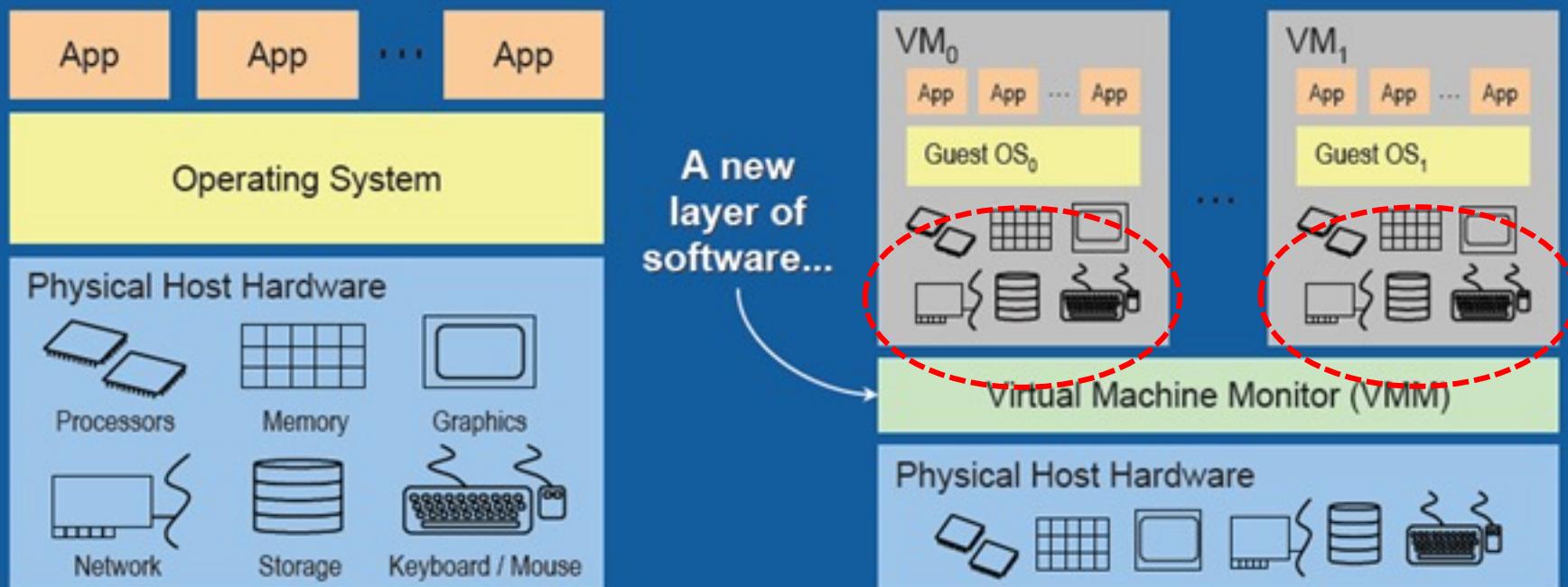
高效虚拟化的三个特征

- 通过模拟 ISA，为虚拟机内程序提供与硬件完全一致的接口
- 虚拟机监控器控制所有物理资源：CPU、内存、设备
- 高效 虚拟化仅比硬件的性能略差 (大多数指令避免VMM干预)



虚拟机监控器 VMM

- 引入一个新的软件层，虚拟机监控器（帮助隔离，帮助翻译）
- 每个虚拟机拥有完整的“硬件资源”



Without VMs: Single OS owns
all hardware resources

With VMs: Multiple OSes
share hardware resources

虚拟机监控器的类型

□ Type-1 型 VMM

- 直接运行在硬件之上，控制硬件资源
- 充当操作系统的角色
- 实现调度、内存管理、驱动的功能
- 性能损失较小



虚拟机监控器的类型

□ Type-2 型 VMM

- 依托于主机操作系统，由主机操作系统管理物理资源
- VMM 以进程/内核模块的形态运行
- 复用 OS 大部分功能：文件系统、处理器调度、内存管理
- 性能损失相对较大



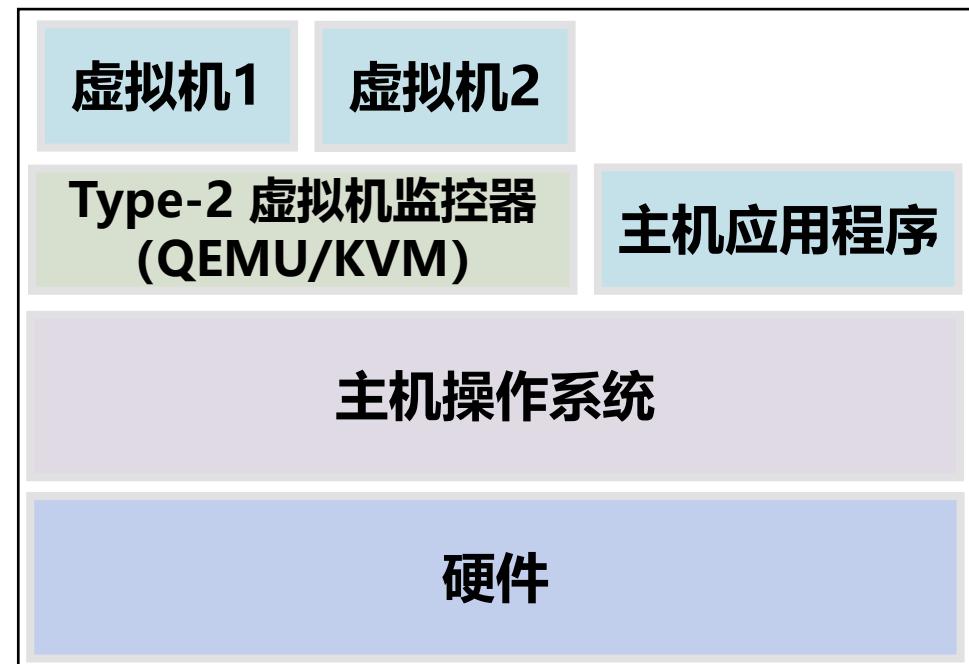
VMware Fusion



Parallel Desktop



VirtualBox



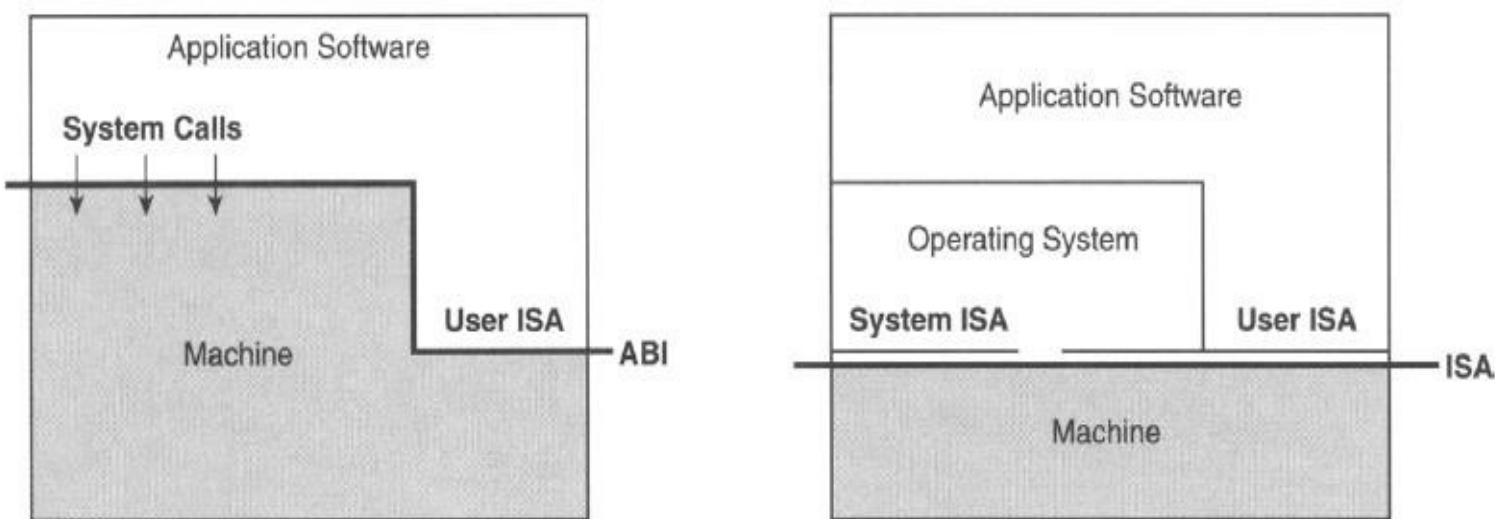
虚拟机基础类型

□ 进程级虚拟机 (Process VMs)

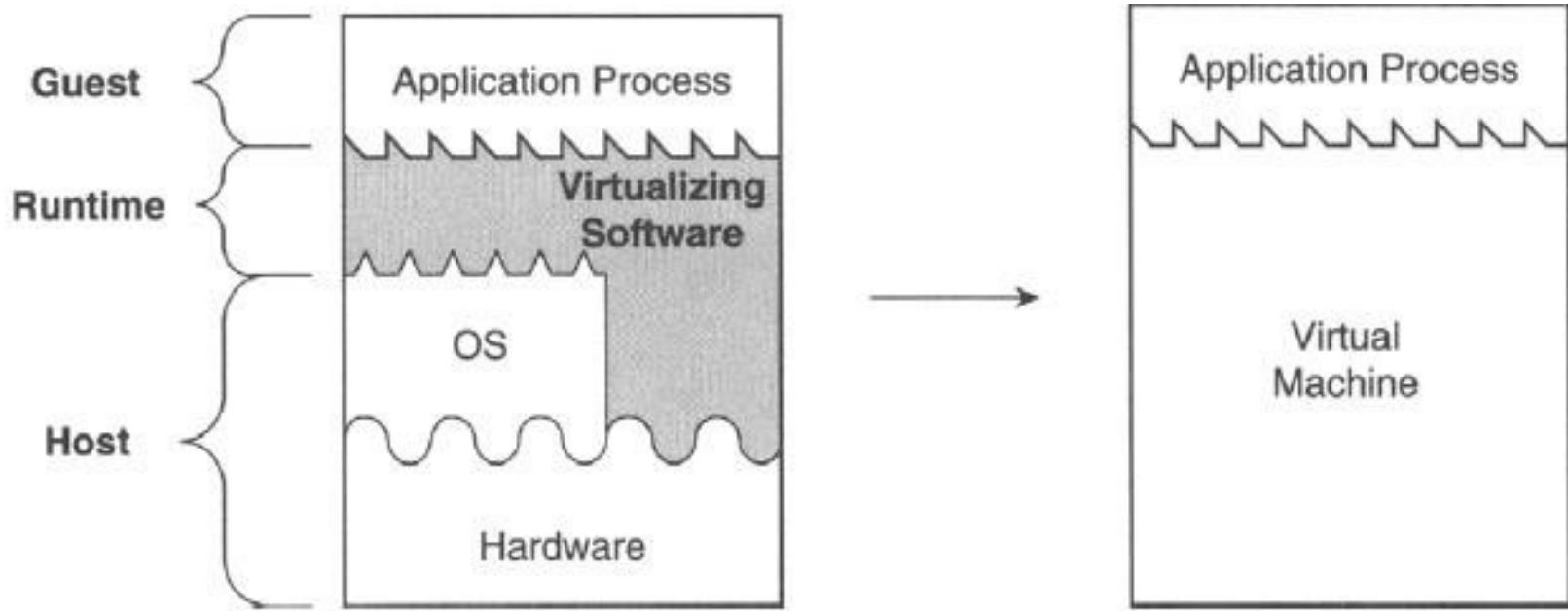
- 虚拟化层置于 ABI 接口层 (ABI VMs)

□ 系统级虚拟机 (System VMs)

- 虚拟化层置于 ISA 接口层 (ISA VMs)

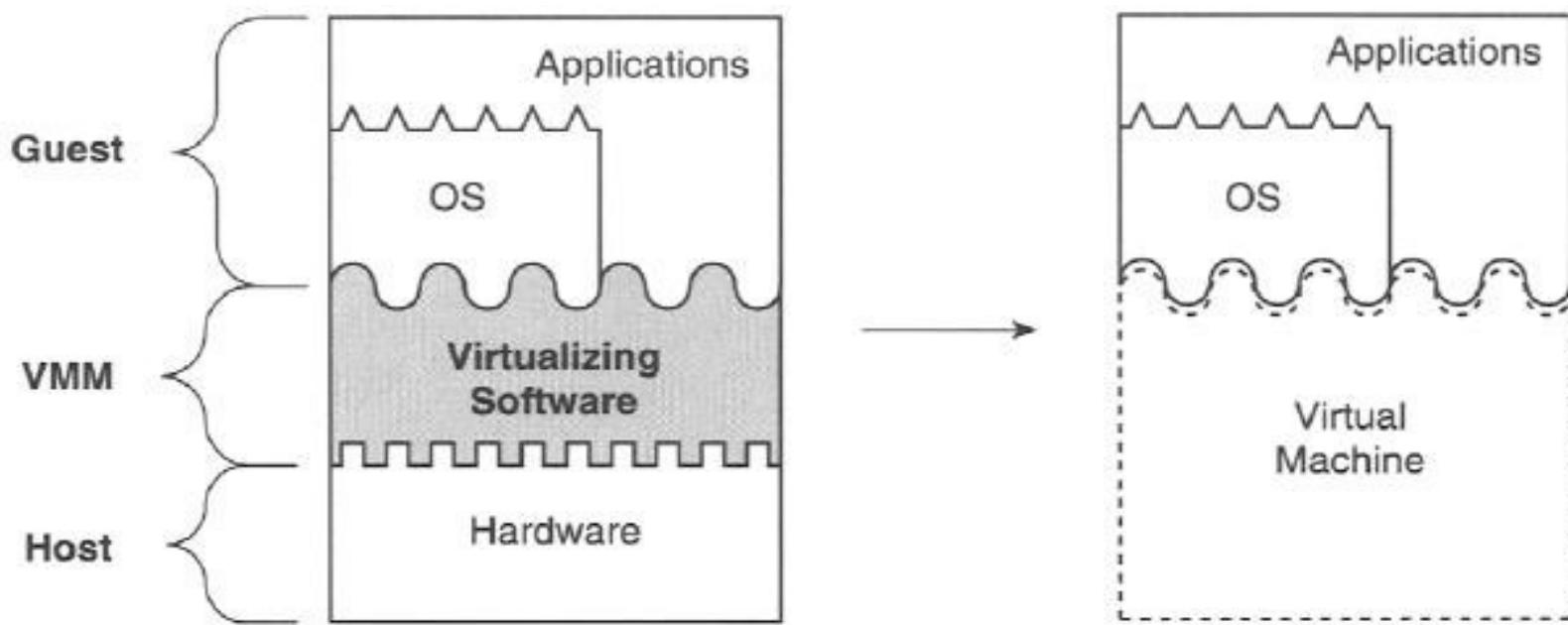


虚拟机基础类型



Process Virtual Machine. Virtualization software translates a set of OS and user-level instructions composing one platform to another, forming a process VM capable of executing programs developed for a different OS and a different ISA.

虚拟机基础类型



System Virtual Machine. Virtualization software translates the ISA used by one hardware platform to another, forming a system VM, capable of executing a system software environment developed for a different set of hardware.

如何实现虚拟化?

□ 用户 ISA 虚拟化

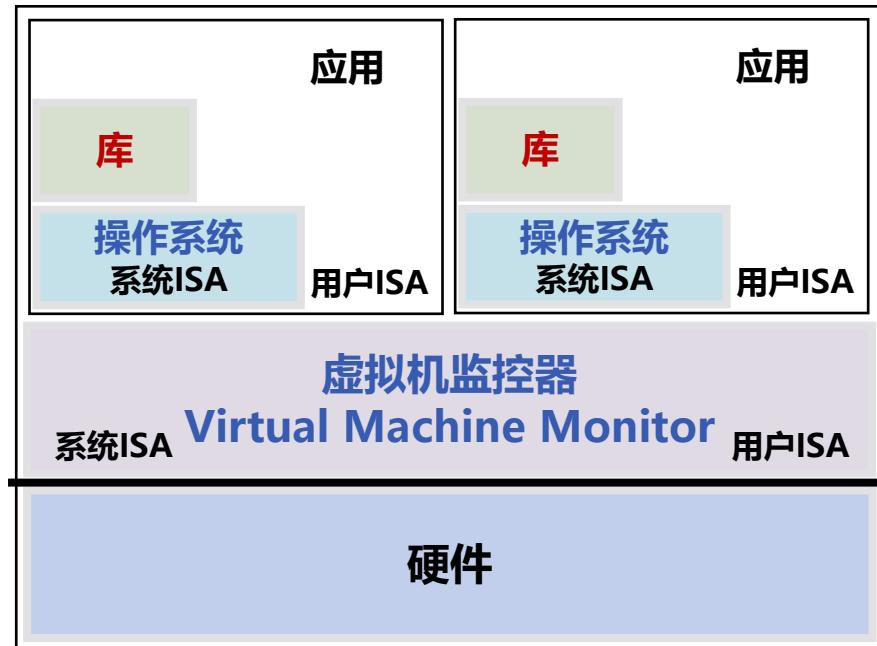
- 操作系统通过进程这一抽象，支持多个应用运行在同一个 CPU
- MOV, ADD, SUB

□ 系统 ISA 虚拟化（关键步骤！）

- 系统 ISA 会影响整个系统，进而影响其他进程或虚拟机
- 读写敏感寄存器
- 控制处理器行为
- 控制虚拟/物理内存
- 控制外设



原先仅针对单一操作系统，若多个操作系统同时控制上述资源则会造成混乱



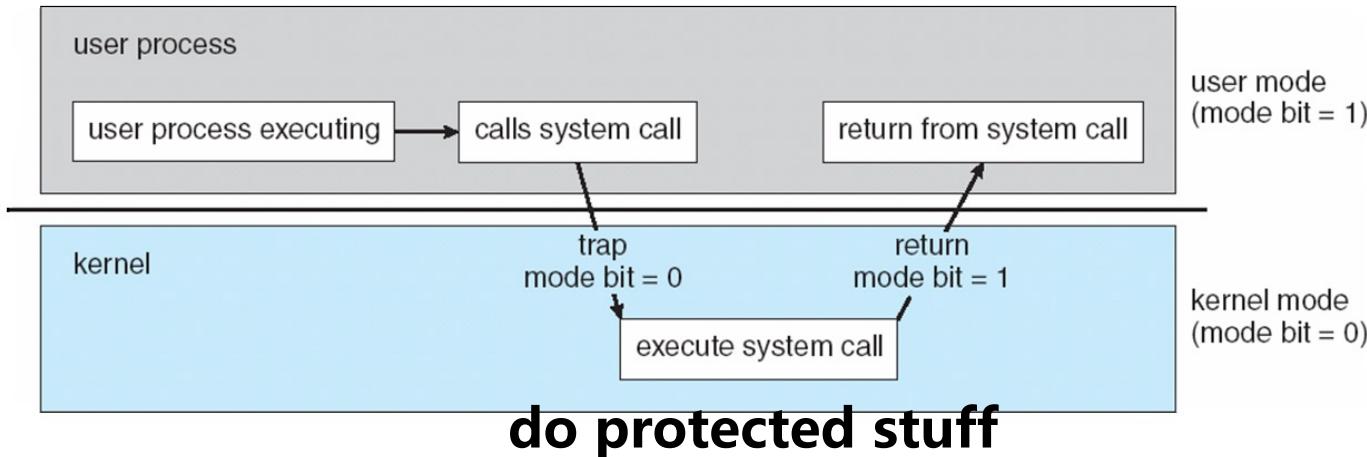
如何捕捉系统 ISA?

- CPU 在用户态执行系统 ISA 时，便会**下陷 (Trap)** 到内核处理
- 在**内核态**，操作系统对计算机的硬件和软件资源拥有完全的控制权，可执行任何系统级任务
- 典型下陷场景

- 执行系统调用 (文件操作/网络通信/进程控制): svc (supervisor call)
- 指令触发异常，如除以零、缺页
- 外设中断信息，如键盘、鼠标

**system call/page
fault/enter on interrupt**

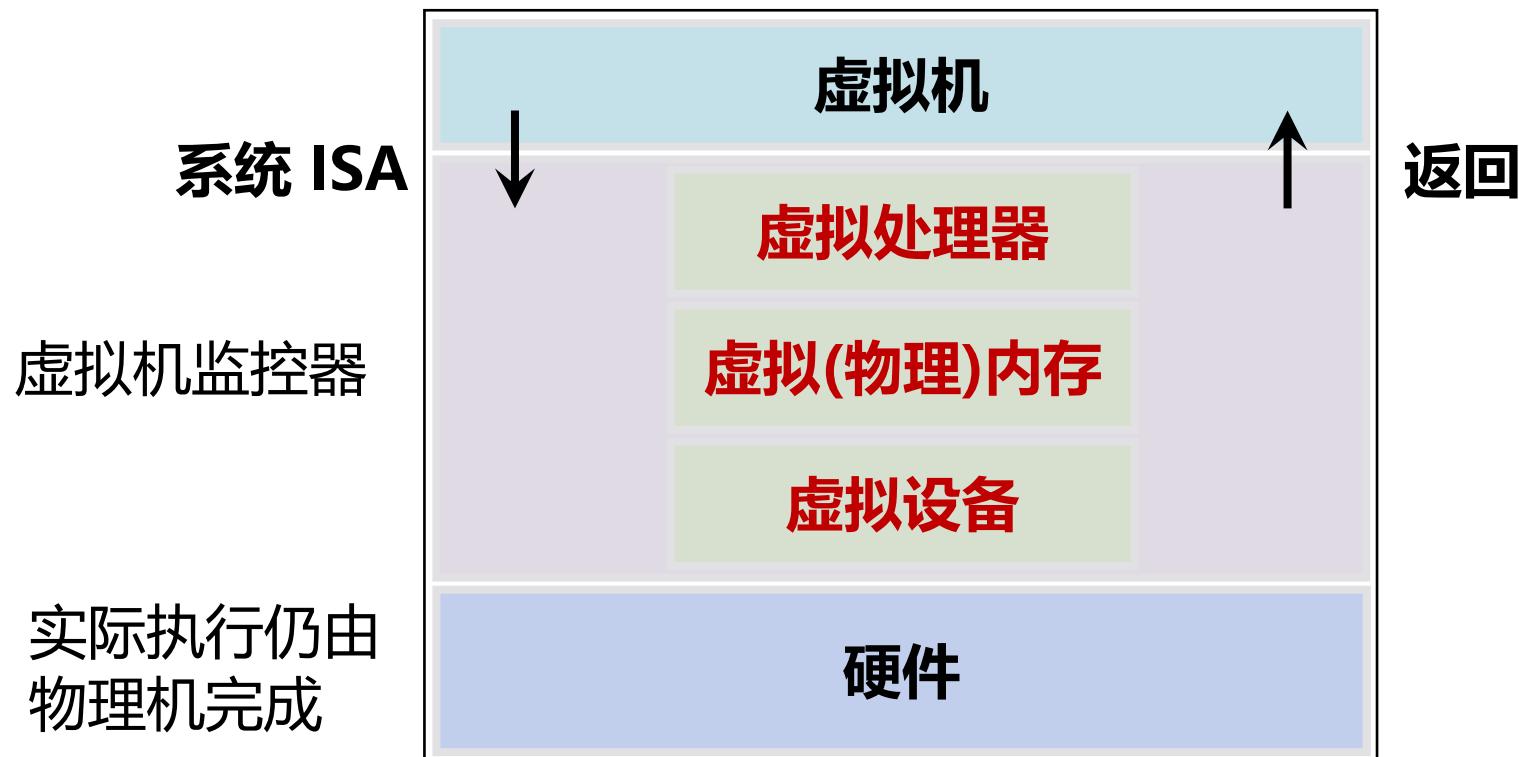
exit back to user program



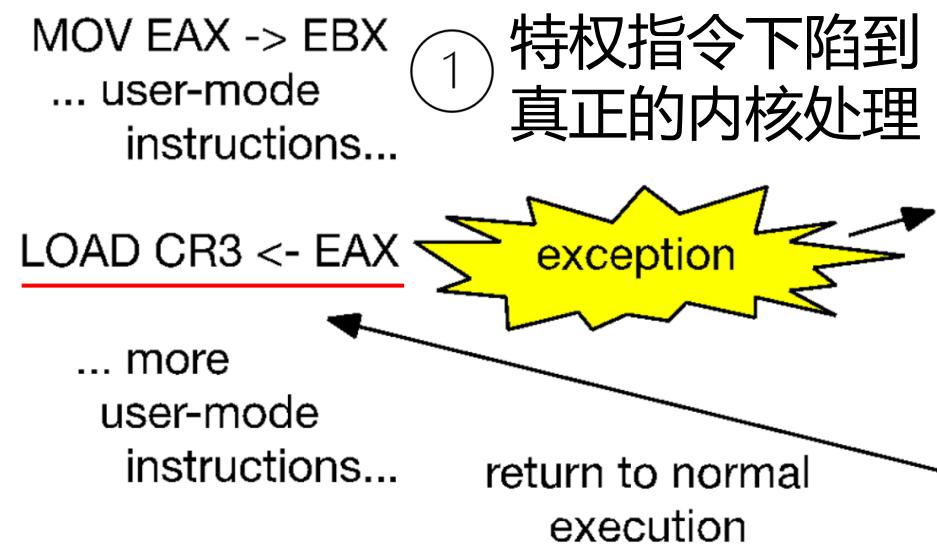
系统虚拟化流程

口下陷 – 模拟 (Trap-and-Emulate)

- Step 1: 捕捉所有系统 ISA 下陷
- Step 2: 模拟出指令的效果, 实现**系统 ISA 虚拟化**
 - 虚拟处理器、虚拟内存、虚拟设备
- Step 3: 回到虚拟机继续执行



下陷 – 模拟过程



① 特权指令下陷到
真正的内核处理

② 用软件的方式模拟指令
对“虚拟机进程”的效果

Load user-visible registers
into software CPU

Emulate one instruction

Restore user-visible registers
from software CPU

③ 将结果加载到 CPU 里，然
后返回用户态的下一条指令

EAX (Extended Accumulator Register): x86架构中的一个32位寄存器，用于算数运算

EBX (Extended Base Register): x86架构中的一个32位寄存器，用作基址寄存器，进行内存寻址

CR3 (Control Register 3): x86架构中用于存储页目录的物理地址，**用户态程序不可访问**

这个过程有什么问题？

- 捕捉下陷指令并处理是为了防止影响整个系统
- 然而，并不是所有对系统有影响的指令均会下陷！

特权指令 Privileged instructions

在用户态执行时会触发下陷的指令，包括主动触发下陷的指令和不允许在用户态执行的指令。

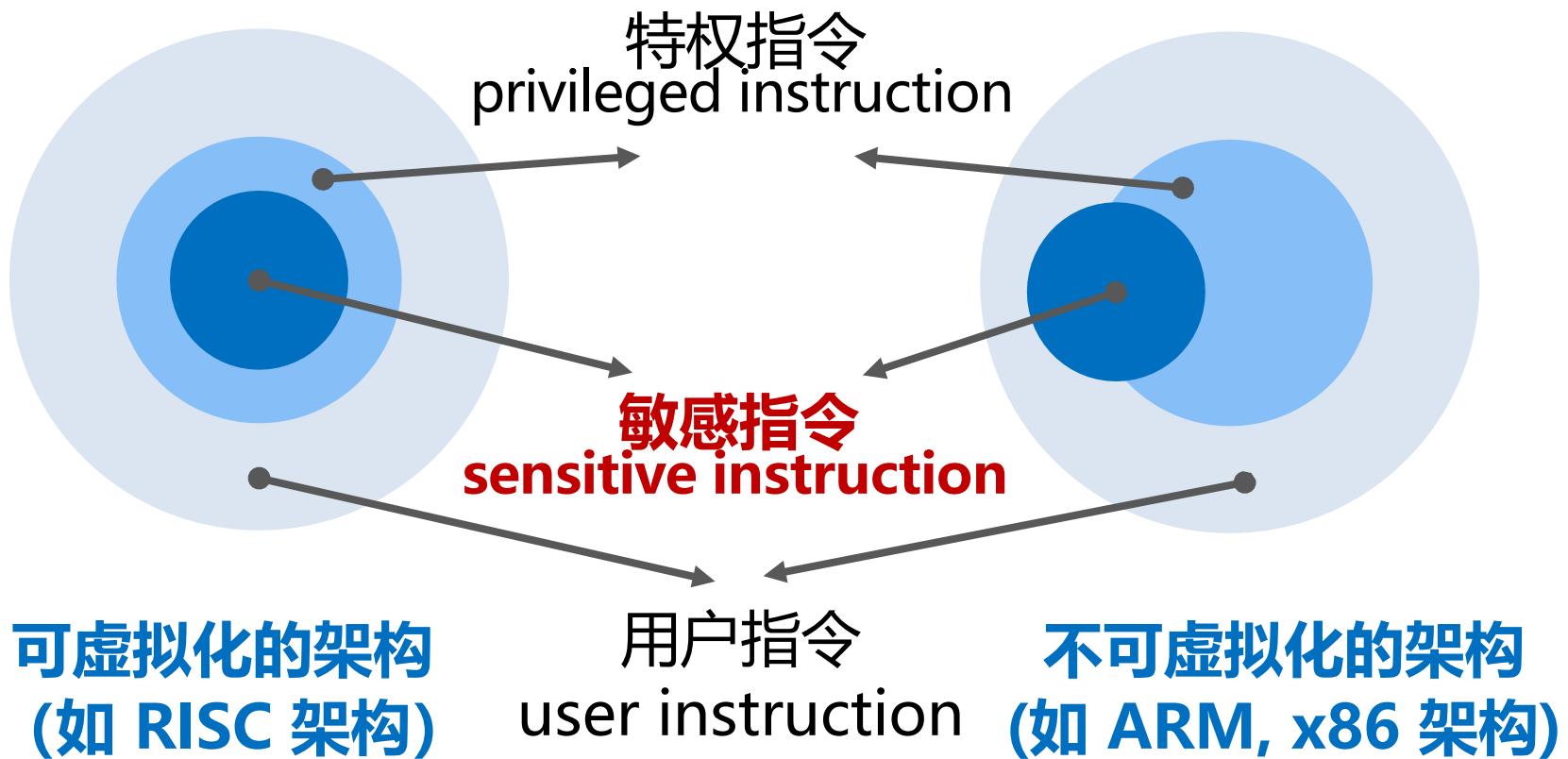
敏感指令 Sensitive instructions

管理系统物理资源或者更改CPU状态的指令：读写特殊寄存器、读写敏感内存、I/O指令。

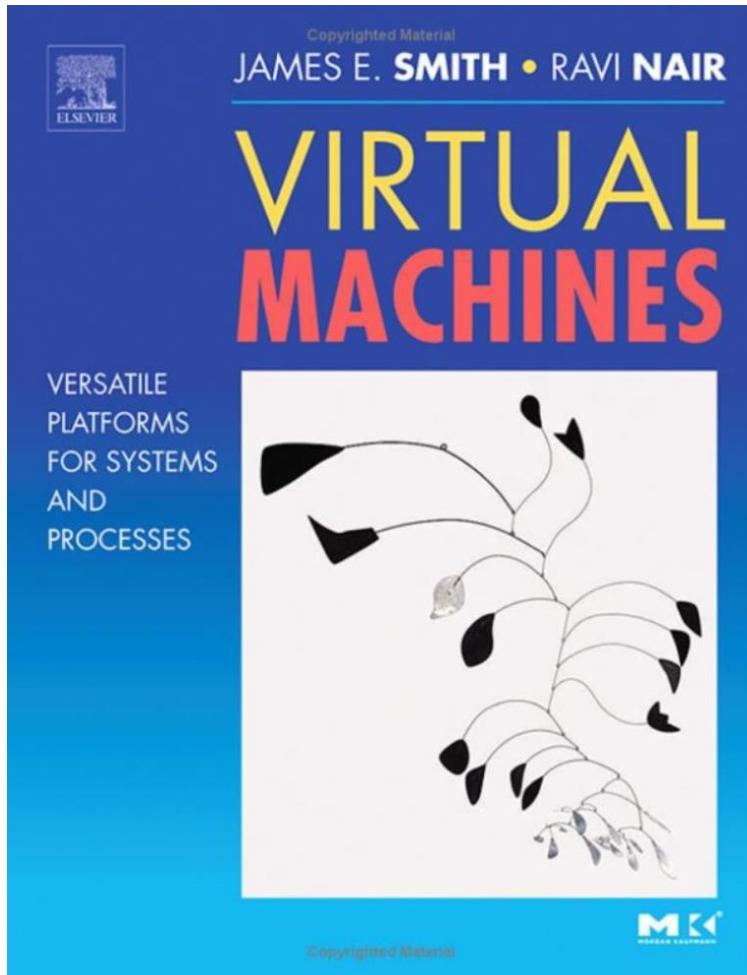
当所有敏感指令都是特殊指令，在非特权级执行时都会发生下陷时，才为可虚拟化架构

虚拟化原则

Conditions for ISA Virtualizability



虚拟化原则



COVER FEATURE

The Architecture of Virtual Machines

A virtual machine can support individual processes or a complete system depending on the abstraction level where virtualization occurs. Some VMs support flexible hardware usage and software isolation, while others translate from one instruction set to another.

James E. Smith
University of Wisconsin-Madison

Ravi Nair
IBM T.J. Watson Research Center

Virtualization has become an important tool in computer system design, and virtual machines are used in a number of subdisciplines ranging from operating systems to programming languages to processes. By moving developers and users from traditional interfaces to constraints, VMs enhance software interoperability, system impregnability, and platform versatility.

Because VMs are the product of diverse groups with different goals, however, there has been relatively little unification of VM concepts. Consequently, it is useful to take a step back, consider the variety of VM architectures, and describe them in a unified way, putting both the notion of virtualization and the types of VMs in perspective.

ABSTRACTION AND VIRTUALIZATION

Despite the multiple applications, some teams exist and continue to evolve because they are designed as hierarchies with *well-defined interfaces* that separate levels of abstraction. Using well-defined interfaces facilitates independent subsystem development by both hardware and software design teams. The simplifying abstractions hide lower-level implementation details, thereby reducing the complexity of the design process.

Figure 1a shows an example of abstraction applied to disk storage. The operating system abstracts the disk as a collection of files—*e.g.*, files that it is composed of sectors and tracks—so that the disk appears to application software as a set of variable-sized files. Application programmers can then create, write, and read files without knowing the hard disk's construction and physical organization.

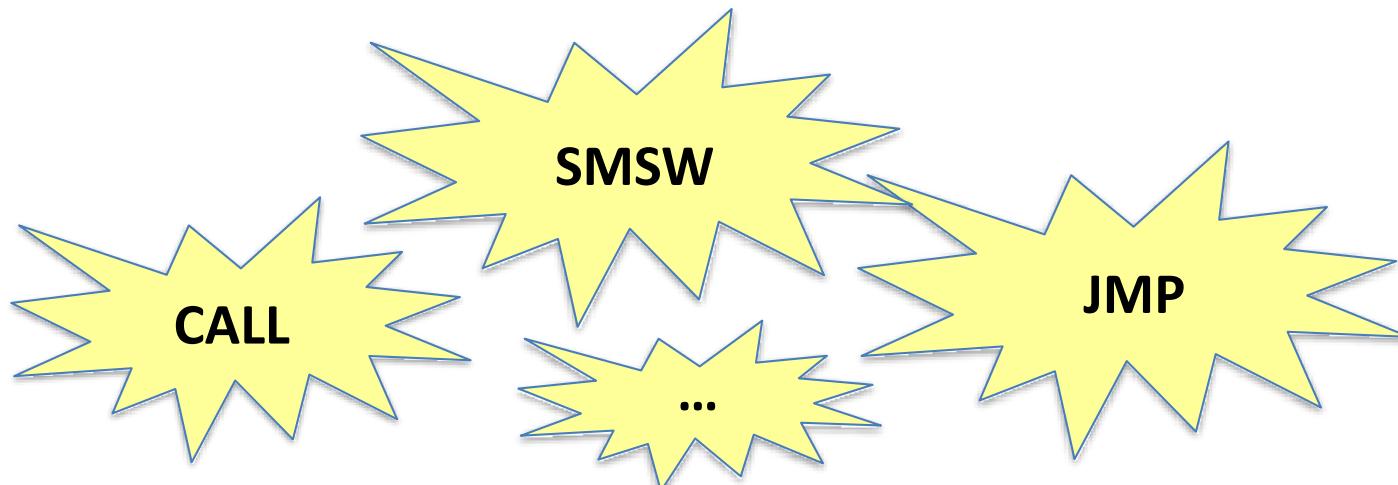
Computer
Published by the IEEE Computer Society
0160-9162/05/\$20.00 © 2005 IEEE

虚拟化原则

□ 原有的 x86 设计架构不满足虚拟化规范要求

- 直至2005年，仍有17条敏感指令不属于特权指令，这导致一些敏感操作可能不会被VMM察觉，e.g., Intel x86 popf

*The x86 processor architecture did not originally meet the **Formal Requirements for Virtualizable Third-Generation Architectures**, a specification for virtualization created in 1974 by Popek and Goldberg.*



虚拟化原则

□ ARM 也不是可虚拟化架构

□ Change Process State (Interrupt Disable, Interrupt Enable)

CPSID

CPSIE

分别用于打开和关闭中断

□ 内核态执行：PSTATE.{A, I, F} 可以被 CPS 指令修改

- PSTATE 是 ARM 架构中的程序状态寄存器
- A (Asynchronous Abort Mask) : 异步异常屏蔽位
- I (IRQ Mask) : 中断请求屏蔽位
- F (FIQ Mask) : 快速中断请求屏蔽位

□ 用户态执行：CPS 被当作 NOP 指令，不产生任何效果

□ 因此**不是特权指令**

虚拟化技术

Emulation (模拟)

Emulation is defined as the process of implementing the interface and functionality of one system or subsystem on a system or subsystem having a different interface and functionality. (J. Smith and R. Nair, Virtual Machines)

Guest
Source ISA

emulated by

Target ISA

Host

两种典型
模拟技术

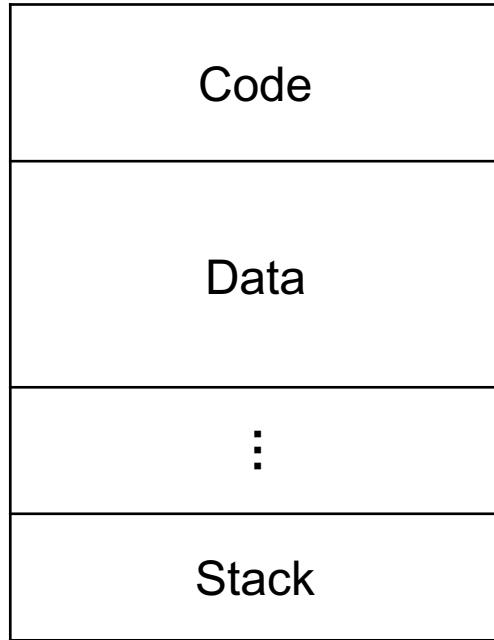
Basic Interpretation
解释执行

Binary Translation
二进制翻译

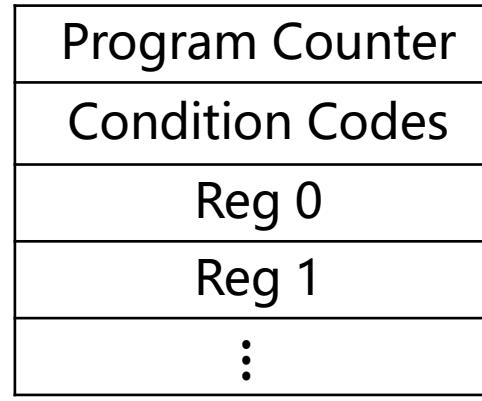
技术一：解释执行

- 口依次取出虚拟机内的每条指令，用软件模拟其执行效果
- 口使用内存维护虚拟机的状态，如使用数组保持寄存器的值
- 口不依赖下陷，能很好处理不同架构间指令集不兼容问题

Source Memory State

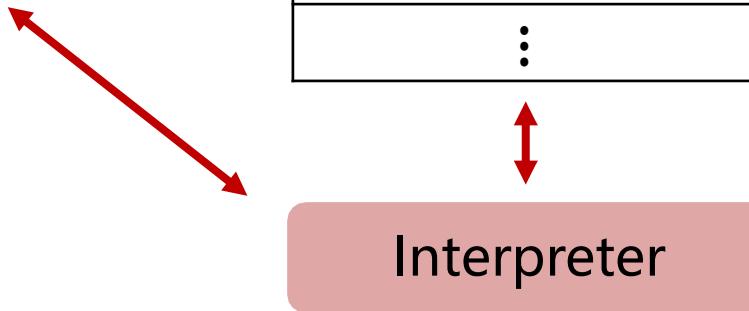


Source Context Block



缺点：开销巨大

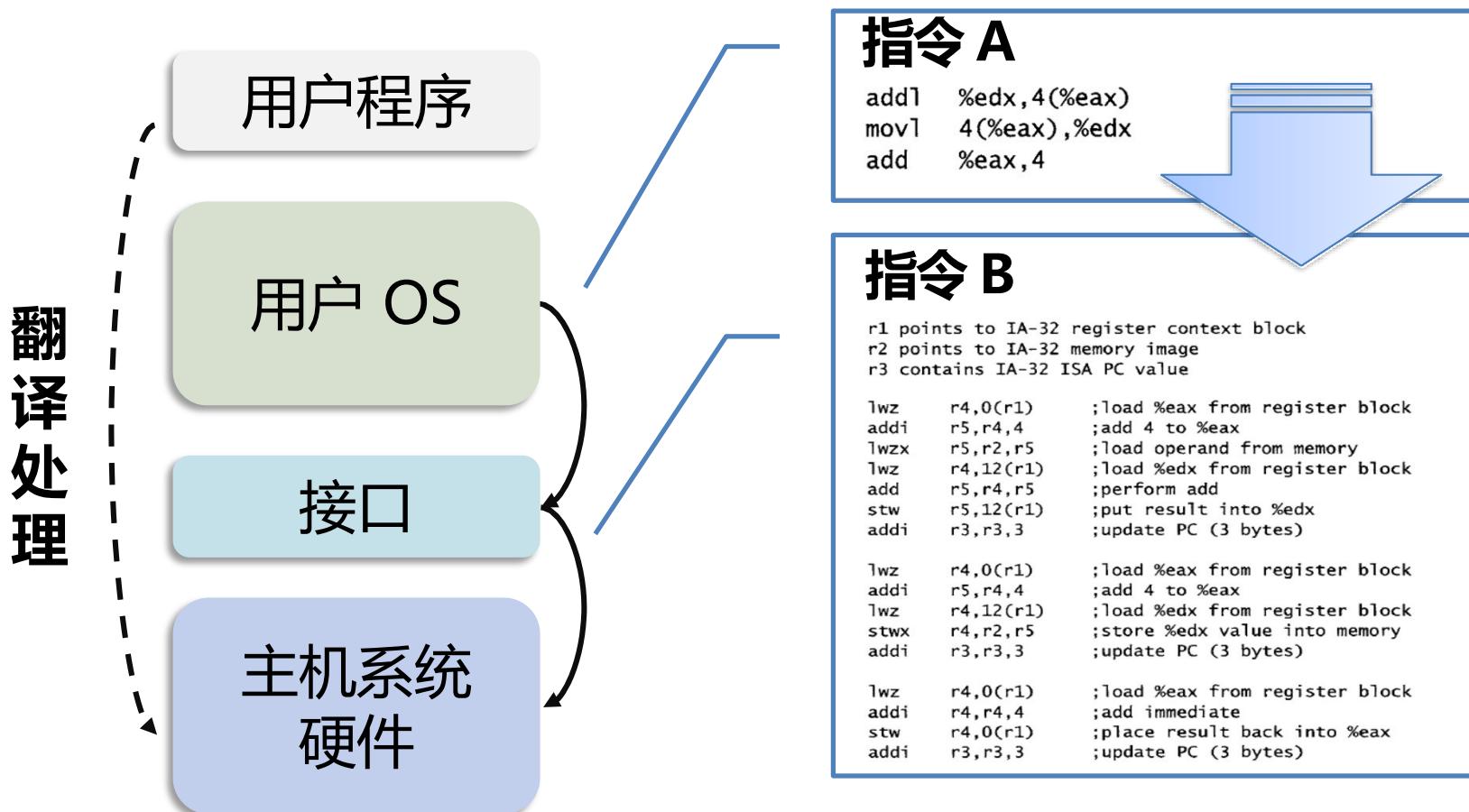
虚拟机指令被转换成多条模拟指令



An interpreter manages the complete architected state of a machine implementing the source ISA

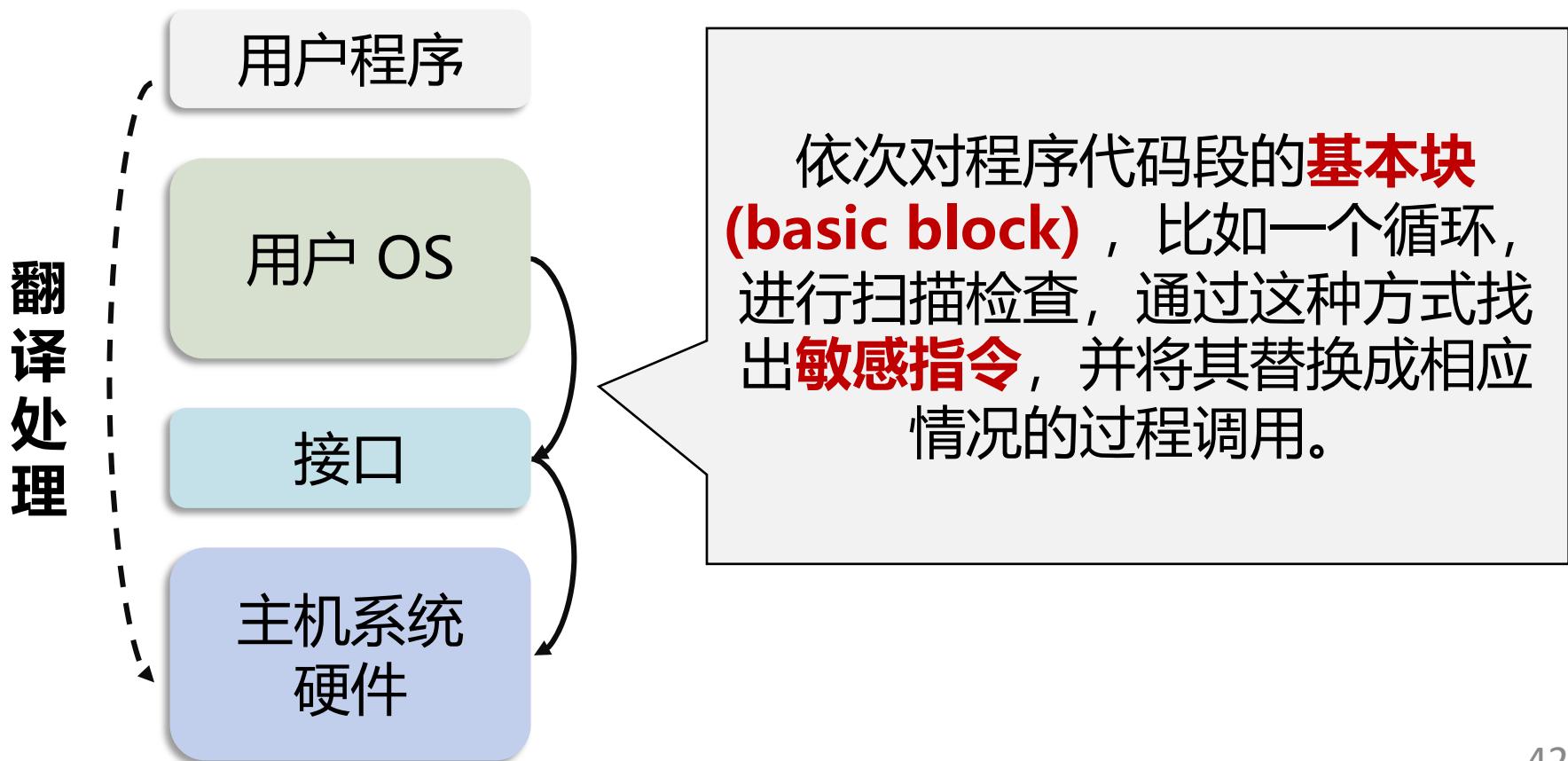
技术二：(动态)二进制翻译

- 将客户机的指令块预先翻译成宿主机的指令
- 这个过程通常会利用一些优化技术，如动态重编译 (JIT编译)



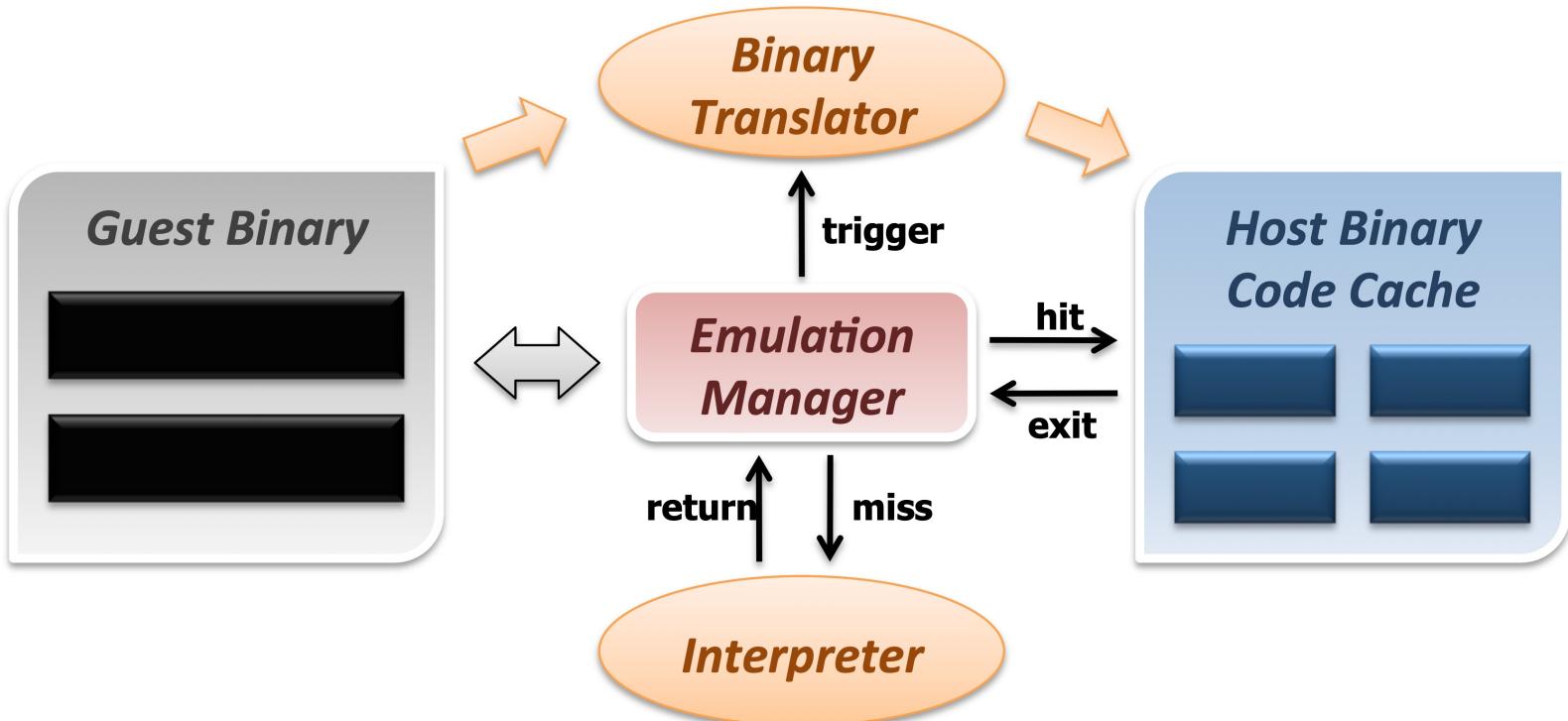
技术二：(动态)二进制翻译

- 缓存代码块（如函数、循环），后续直接执行翻译后的代码
- 结合解释执行和缓存重用（首次执行时翻译，后续直接执行翻译后的代码）



技术二：(动态)二进制翻译

- 相比解释执行，动态二进制翻译采用**批量翻译**与**缓冲**提高性能
- 缓存翻译好的代码块（如循环或热点函数），便于后续重用
- 翻译代码块前先检查其是否在缓存中



技术二：(动态)二进制翻译

口优点

- **跨平台兼容性**: 虚拟机中的程序可以在不同架构的平台运行
- **优化机会**: 翻译器可以在运行过程中进行优化
- **安全和隔离**: 对代码进行细粒度控制，如插入额外的安全检查

口缺点

- **性能开销**: 翻译会导致较大的性能开销，首次翻译开销仍存在
- **复杂性**: 实现一个高效的二进制翻译器非常复杂
- **中断粒度变大**: 只能在基本块边界插入虚拟中断

Advantage



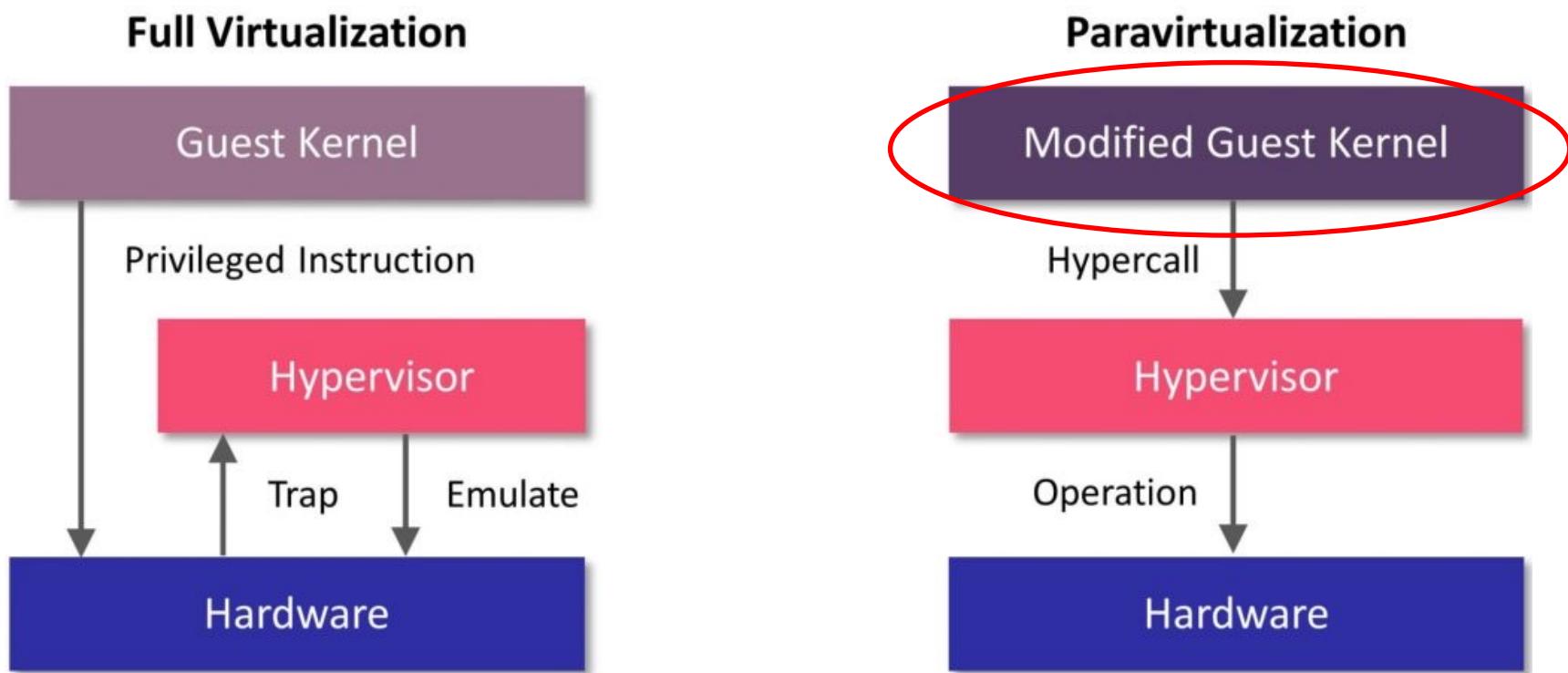
Disadvantage



技术三：半虚拟化技术 Paravirtualization

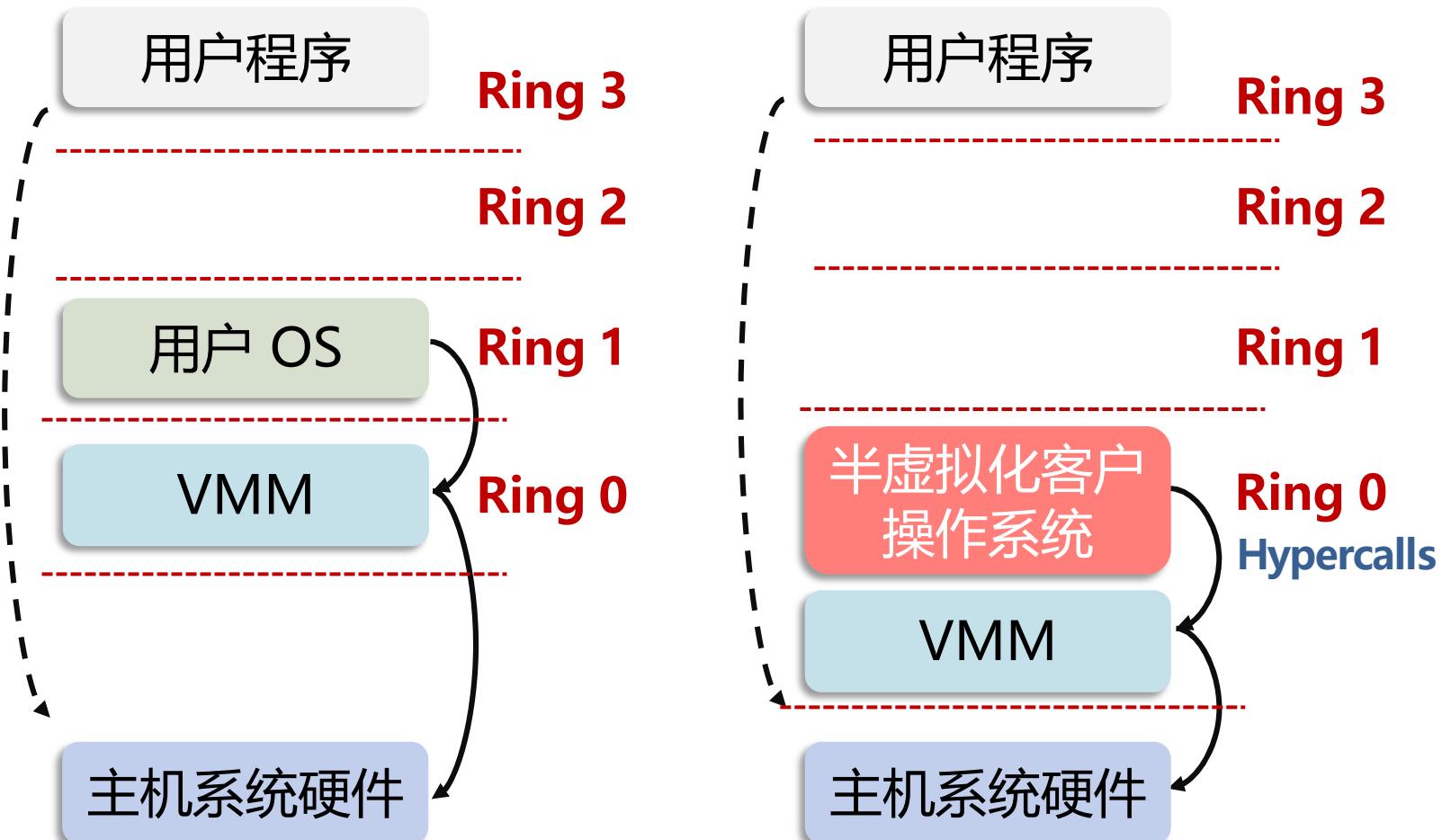
口**全虚拟化技术**：不能修改客户虚拟机的代码，必须在机器指令层面进行模拟或翻译

口**半虚拟化技术**：允许修改客户虚拟机的代码，需要客户操作系统与虚拟机监控器协同设计



技术三：半虚拟化技术

- 口半虚拟化客户操作系统与虚拟监控器密切协作
- 口客户 OS 知道自己运行在虚拟环境中，并依此优化自己



技术三：半虚拟化技术

口虚拟机监控器侧

- 为虚拟机提供**超级调用 (Hypercall)**，客户操作系统主动调用虚拟化层的 Hypercall 来执行特权操作
- 超级调用与系统调用类似，涵盖调度、内存、I/O 等

口客户操作系统侧

- 修改操作系统源代码
- 替换不可虚拟化的指令，改为超级调用

半虚拟化客户
操作系统

VMM

协同设计

将所有不能引起下陷的敏感指令替换为超级调用！

技术三：半虚拟化技术

口优点

- 解决敏感指令不下陷的问题
- 提高系统性能，如减少调度开销
- 缓解**语义鸿沟**，能获得虚拟机内部状态，提高资源分配效率

口缺点

- 需要修改操作系统代码，但如今大多数厂商也愿意拥抱虚拟化
- 维护成本高，如专门的虚拟化接口和驱动、不同操作系统版本

Advantage



Disadvantage



技术四：硬件辅助虚拟化

□ 传统虚拟化技术中，VMM 和客户 OS **共享同一个特权级别**，客户 OS 特权指令直接操作硬件，导致**资源冲突或逃逸漏洞**（如虚拟机破坏宿主机）

□ **硬件辅助虚拟化，由处理器硬件提供架构支持以创建 VMM，因此允许客户机 OS 不加修改直接运行**

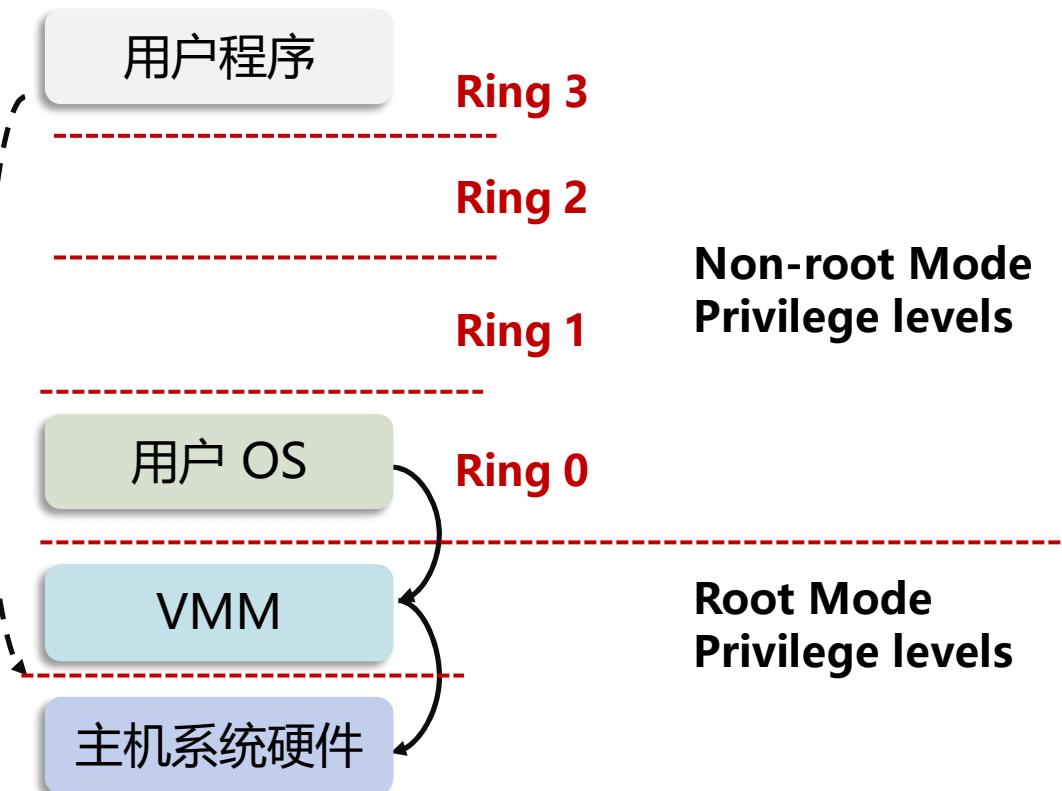
– first introduced on the IBM System/370 in 1972

□ 2005/2006年间，Intel 和 AMD 两家公司各自独立推出了解决方案，即提供**处理器扩展**（CPU extensions）

– Intel VT-x (formerly known as Vanderpool)
– AMD-V (formerly known as Pacifica)

技术四：硬件辅助虚拟化

- ✓ 在已有 CPU 权级下增加根模式 (root mode) 和非根模式 (non-root mode)
- ✓ Intel VT-x 为每一个虚拟机提供虚拟机控制结构 (virtual machine control structure, VMCS)
- ✓ 在硬件层面实现了对敏感指令的监控和拦截，都能被 VMM 捕获



VMM 运行在根模式，拥有对硬件的完全控制权

客户 OS 运行在非根模式，表面上可以执行特权指令，但实际受硬件限制，只能在 VMM 的控制下运行

技术四：硬件辅助虚拟化

口硬件虚拟化的作用

- CPU 在非根模式下维护一个敏感指令列表
- 当客户机尝试执行这些指令时，CPU 硬件会直接拦截，无需依赖软件模拟或修改客户机代码

口示例

- 当客户机执行 MOV CR3 (切换页表) 时，CPU 触发 VM Exit，Hypervisor 会更新扩展页表 (EPT) 以隔离客户机内存
- 当客户机执行 IN EAX, DX (读取端口) 时，CPU 触发 VM Exit，Hypervisor 将 I/O 请求转发给虚拟设备，如 QEMU 模拟的磁盘

各种虚拟化技术的比较

	全虚拟化	硬件辅助	半虚拟化
技术	二进制翻译	虚拟化扩展	Hypercall
兼容性	好	好	差
性能	较好	好	分场合



中山大學 软件工程学院
SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

谢谢

陈壮彬
软件工程学院
chenzhb36@mail.sysu.edu.cn