IPADS INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

系统虚拟化

陈海波 / 夏虞斌

上海交通大学并行与分布式系统研究所

https://ipads.se.sjtu.edu.cn

版权声明

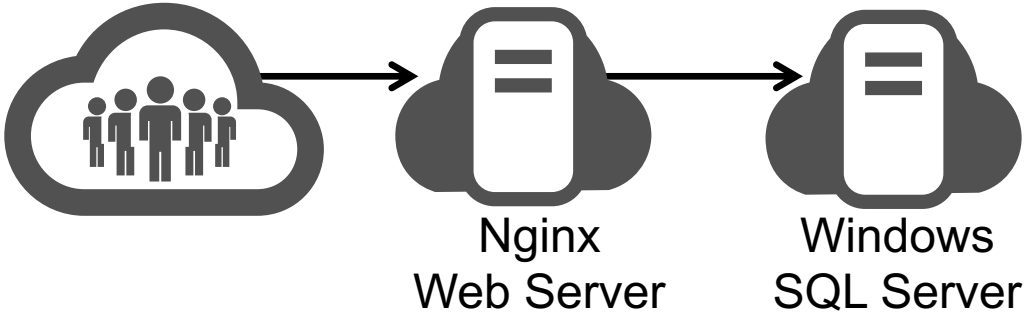
- 本内容版权归**上海交通大学并行与分布式系统研究所**所有
- 使用者可以将全部或部分本内容免费用于非商业用途
- 使用者在使用全部或部分本内容时请注明来源：
  - 内容来自：上海交通大学并行与分布式系统研究所+材料名字
- 对于不遵守此声明或者其他违法使用本内容者，将依法保留追究权
- 本内容的发布采用 Creative Commons Attribution 4.0 License
  - 完整文本：https://creativecommons.org/licenses/by/4.0/legalcode

计算设备集中与分散的变化

- **大型机时代**
  - 集中式计算资源，所有人通过网络连接，共享计算资源
  - 20世纪70年代，虚拟化技术兴起(!)
- **PC时代**
  - 分布式计算资源，每个PC用户独占计算资源
  - 20世纪80-90年代，虚拟化技术沉寂
- **云时代**
  - 集中式计算资源，所有人通过网络连接，共享计算资源
  - 21世纪，虚拟化技术再次兴起

现代IT公司的部署方式：云

- **云服务器代替物理服务器**
  - 云服务器配置与物理服务器一致
  - 所有云服务器维护由服务商提供

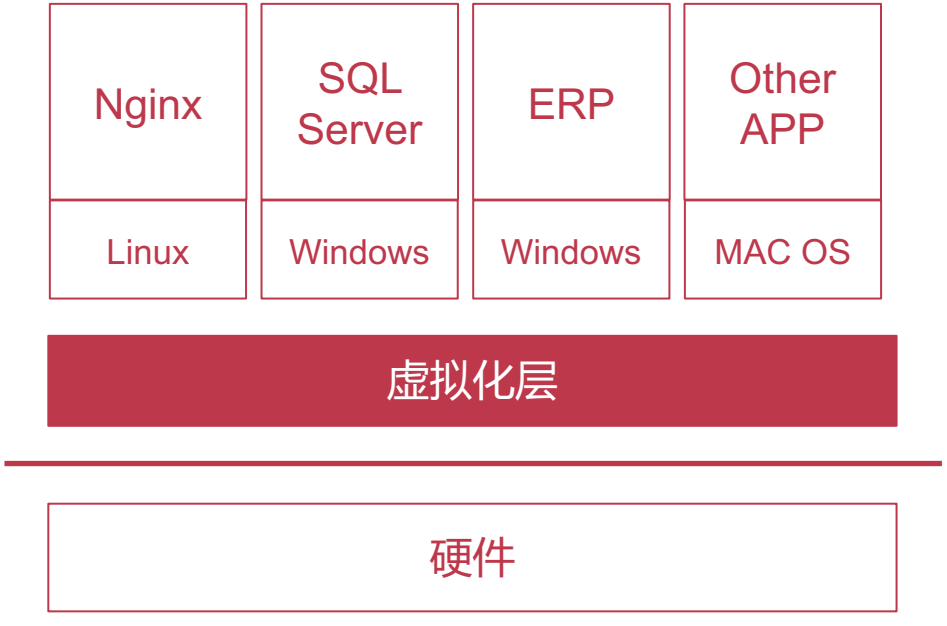


云计算为云租户带来的优势

- 按需租赁、无需机房租赁费
- 无需雇佣物理服务器管理人员
- 可以快速低成本地升级服务器
- …

系统虚拟化是云计算的核心支撑技术

- **新引入的一个软件层**
  - 上层是操作系统（虚拟机）
  - 底层硬件与上层软件解耦
  - 上层软件可在不同硬件之间切换



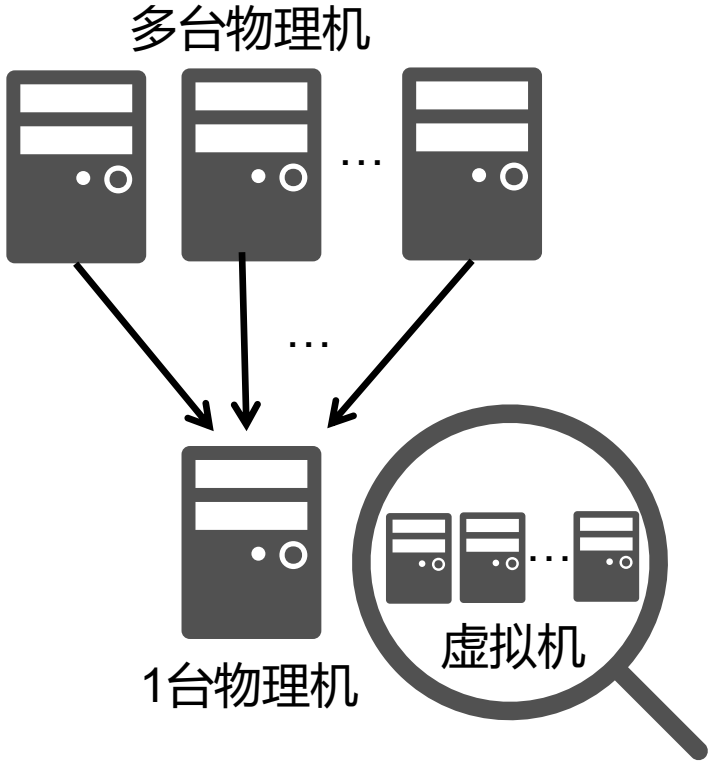
虚拟化带来的优势

- ”*Any problem in computer science can be solved by another level of indirection*" **--- David Wheeler**

- **1、服务器整合：提高资源利用率**
- **2、方便程序开发**
- **3、简化服务器管理**
- **…**

虚拟化优势-1：服务器整合

- **单个物理机资源利用率低**
  - CPU利用率通常仅20%
- **利用系统虚拟化进行资源整合**
  - 一台物理机同时运行多台虚拟机
- **提升物理机资源利用率**
- **降低云服务提供商的成本**



虚拟化优势-2：方便程序开发

- **调试操作系统**
  - 单步调试操作系统
  - 查看当前虚拟硬件的状态
    - 寄存器中的值是否正确
    - 内存映射是否正确
  - 随时修改虚拟硬件的状态
- **测试应用程序的兼容性**
  - 可以在一台物理机上同时运行在不同的操作系统
  - 测试应用程序在不同操作系统上的兼容性

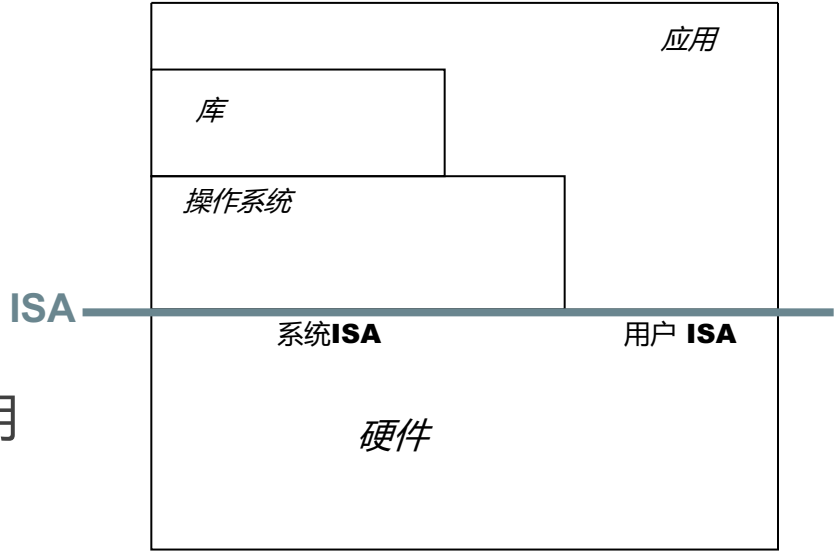
虚拟化优势-3：简化服务器管理

- **通过软件接口管理虚拟机**
  - 创建、开机、关机、销毁
  - 方便高效
- **虚拟机热迁移**
  - 方便物理机器的维护和升级

什么是系统虚拟化？

操作系统中的接口层次: ISA

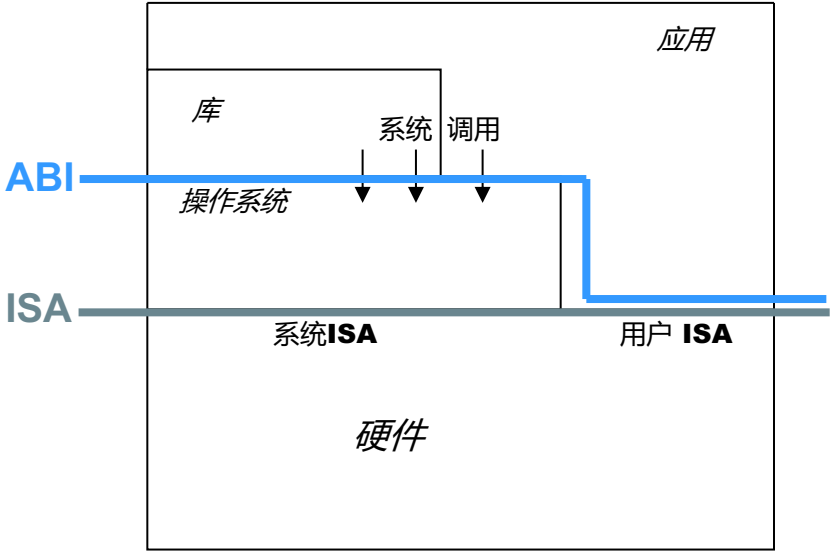
- **ISA层**
  - Instruction Set Architecture
  - 区分硬件和软件
  - 用户ISA
    - 用户态和内核态程序都可以使用
    - mov x0, sp
    - add x0, x0, #1
  - 系统ISA
    - 只有内核态程序可以使用
    - msr vbar_el1, x0



ISA – instruction set architecture

操作系统中的接口层次: ABI

- **ABI**
  - Application Binary Interface
  - 提供操作系统服务或硬件功能
  - 包含用户ISA和系统调用

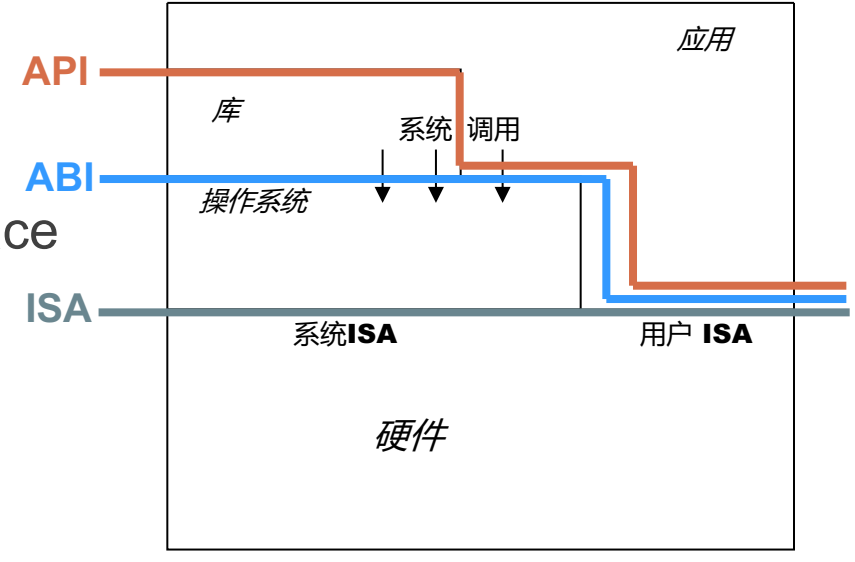


ABI – application binary interface

ISA – instruction set architecture

操作系统中的接口层次: API

- **API**
  - Application Programming Interface
  - 不同用户态库提供的接口
  - 包含库的接口和用户ISA
  - UNIX环境中的**clib**:
    - 支持UNIX/C编程语言



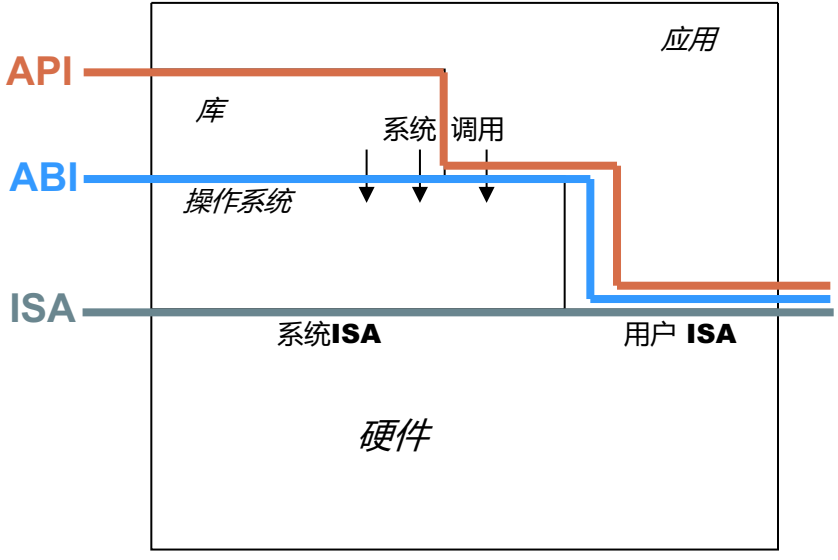
API – application programming interface

ABI – application binary interface

ISA – instruction set architecture

思考：这些程序用了哪层接口？

- Hello world
- Web game
- Dota
- Office 2016
- Windows 10
- Java applications
- ChCore



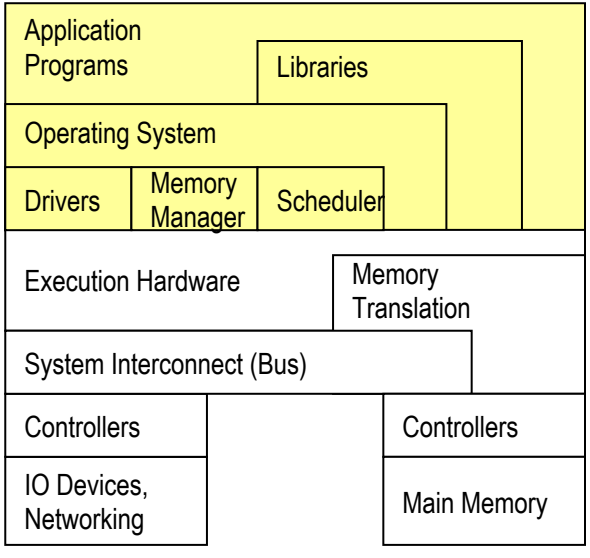
API – application programming interface

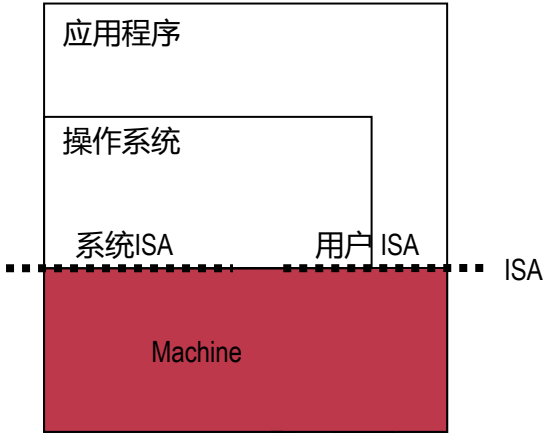
ABI – application binary interface

ISA – instruction set architecture

如何定义虚拟机？

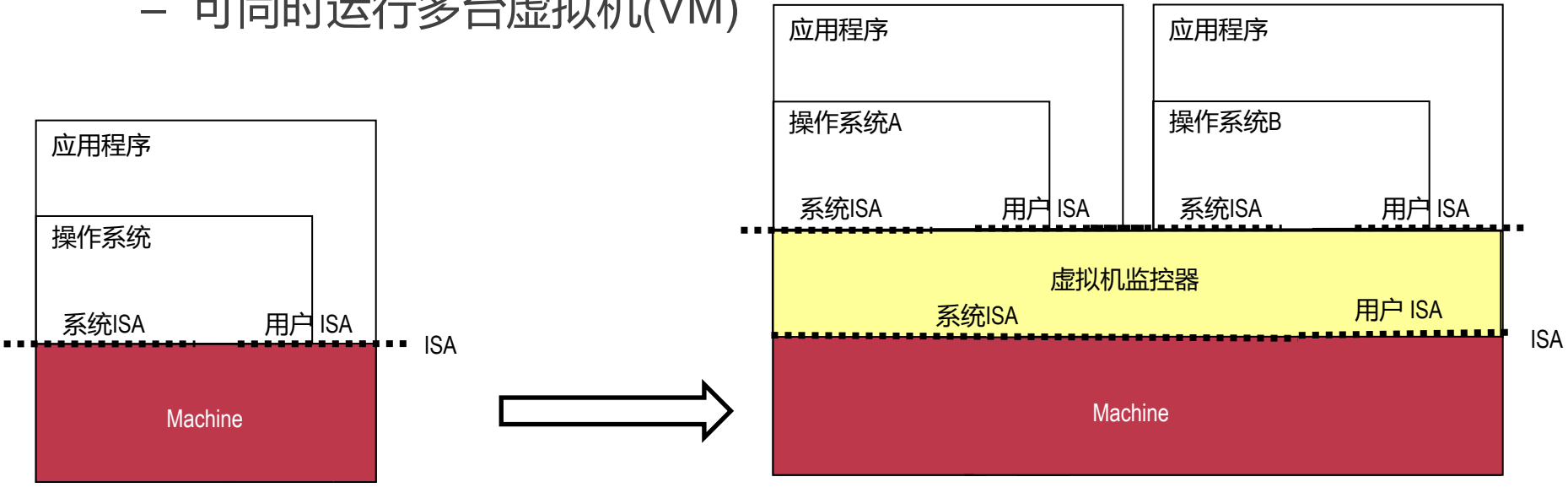
- **从操作系统角度看"Machine"**
  - ISA 提供了操作系统和Machine之间的界限





虚拟机和虚拟机监控器

- **虚拟机监控器(VMM/Hypervisor)**
  - 向上层虚拟机暴露其所需要的ISA
  - 可同时运行多台虚拟机(VM)



系统虚拟化的标准

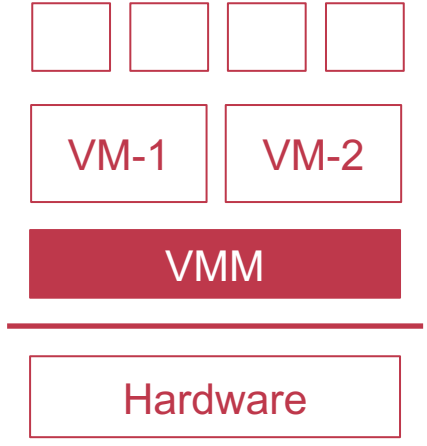
- **Popek & Goldberg, 1974 “Formal Requirements for Virtualizable Third Generation Architectures”**

- **高效系统虚拟化的三个特性**
  - 为虚拟机内程序提供与该程序原先执行的硬件完全一样的接口
  - 虚拟机只比在无虚拟化的情况下性能略差一点
  - 虚拟机监控器控制所有物理资源

虚拟机监控器的分类

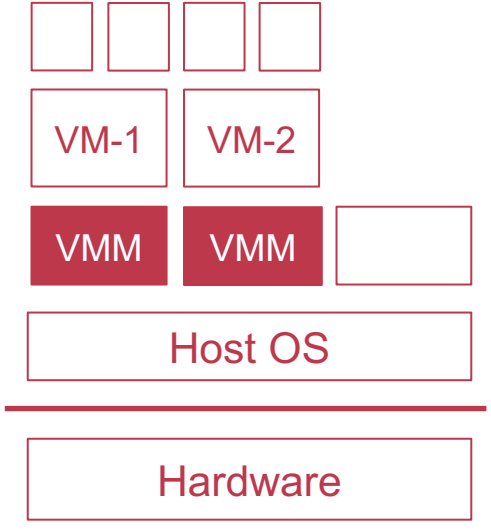
Type-1虚拟机监控器

- **直接运行在硬件之上**
  - 充当操作系统的角色
  - 直接管理所有物理资源
    - 实现调度、内存管理、驱动等功能
- **性能损失较少**
- **例如Xen, VMware ESX Server**



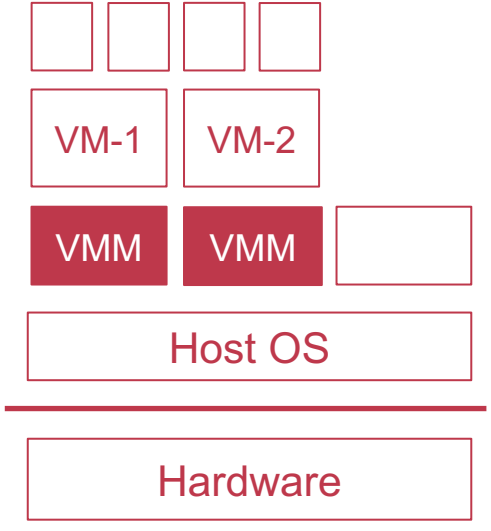
Type-2虚拟机监控器

- **依托于主机操作系统**
  - 主机操作系统管理物理资源
  - 虚拟机监控器以进程/内核模块的形态运行
- **易于实现和安装**
- **例如QEMU/KVM**
- **思考：**
  - Type-2类型有什么优势？



Type-2的优势

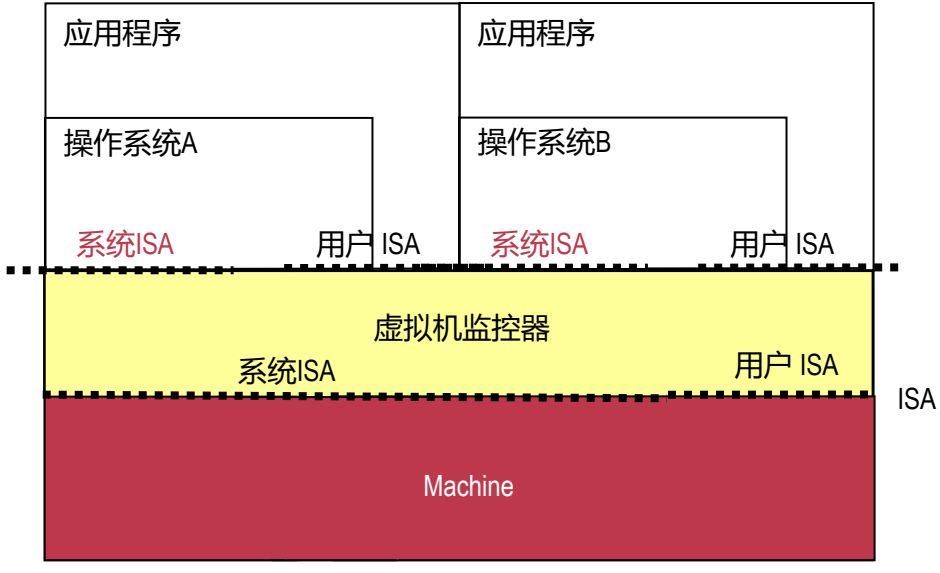
- **在已有的操作系统之上将虚拟机当做应用运行**
- **复用主机操作系统的大部分功能**
  - 文件系统
  - 驱动程序
  - 处理器调度
  - 物理内存管理



如何实现系统虚拟化？

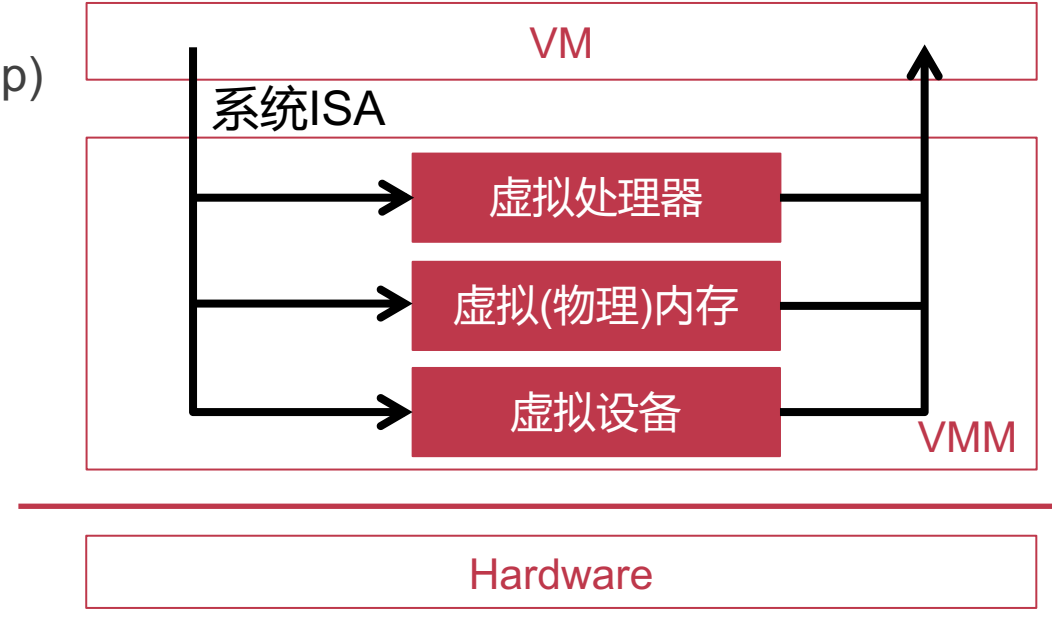
系统ISA

- **读写敏感寄存器**
  - sctrl_el1、ttbr0_el1/ttbr1_el1…
- **控制处理器行为**
  - 例如: WFI(陷入低功耗状态)
- **控制虚拟/物理内存**
  - 打开、配置、安装页表
- **控制外设**
  - DMA、中断



系统虚拟化的流程

- **第一步**
  - 捕捉所有系统ISA并陷入(Trap)
- **第二步**
  - 由具体指令实现相应虚拟化
    - 控制虚拟处理器行为
    - 控制虚拟内存行为
    - 控制虚拟设备行为
- **第三步**
  - 回到虚拟机继续执行



系统虚拟化技术

- **处理器虚拟化**
  - 捕捉系统ISA
  - 控制虚拟处理器的行为
- **内存虚拟化**
  - 提供“假”物理内存的抽象
- **设备虚拟化**
  - 提供虚拟的I/O设备

CPU Virtualization

处理器虚拟化

回顾：ARM的特权级

- EL0: 用户态进程
- EL1: 操作系统内核

处理器虚拟化：一种直接的实现方法

- 将虚拟机监控器运行在EL1
- 将客户操作系统和其上的进程都运行在EL0
- 当操作系统执行系统ISA指令时下陷
  - 写入TTBR0_EL1
  - 执行WFI指令
  - …

Trap & Emulate

- Trap: 在用户态EL0执行特权指令将陷入EL1的VMM中
- Emulate：这些指令的功能都由VMM内的函数实现

ARM不是严格的可虚拟化架构

- **敏感指令**
  - 读写特殊寄存器或更改处理器状态
  - 读写敏感内存：例如访问未映射内存、写入只读内存
  - I/O指令
- **特权指令**
  - 在用户态执行会触发异常，并陷入内核态

ARM不是严格的可虚拟化架构

- **在ARM中：不是所有敏感指令都属于特权指令**
- **例子: CPSID/CPSIE指令**
  - CPSID和CPSIE分别可以关闭和打开中断
  - 内核态执行：PSTATE.{A, I, F} 可以被CPS指令修改
  - 在用户态执行：CPS 被当做NOP指令，不产生任何效果
    - 不是特权指令

如何处理这些不会下陷的敏感指令？

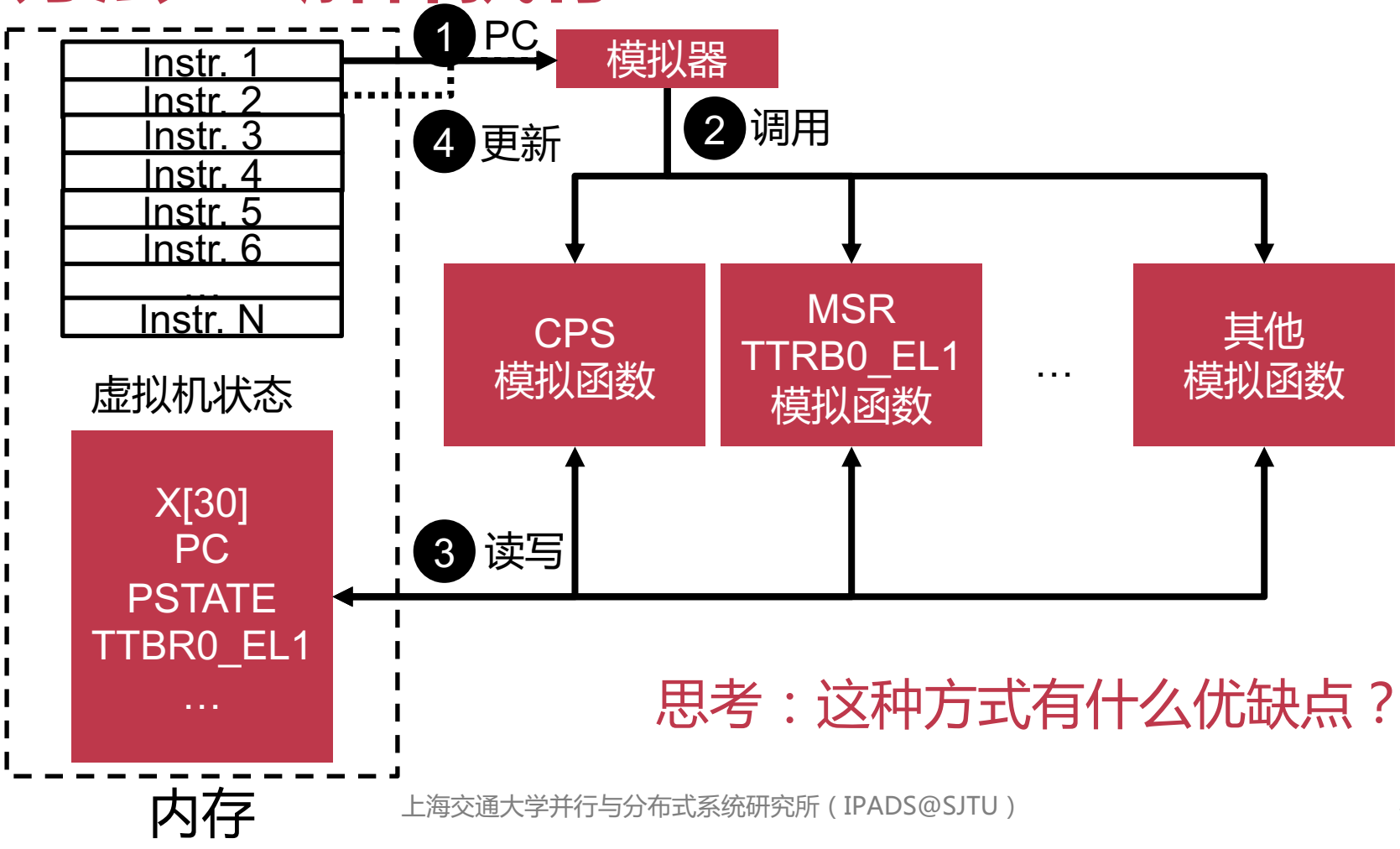
处理这些不会下陷的敏感指令，使得虚拟机中的操作系统能够运行在用户态（EL-0）

- **方法1：解释执行**
- **方法2：二进制翻译**
- **方法3：半虚拟化**
- **方法4：硬件虚拟化（改硬件）**

方法1：解释执行

- **使用软件方法一条条对虚拟机代码进行模拟**
  - 不区分敏感指令还是其他指令
  - 没有虚拟机指令直接在硬件上执行
- **使用内存维护虚拟机状态**
  - 例如：使用uint64_t x[30]数组保存所有通用寄存器的值

方法1：解释执行



思考：这种方式有什么优缺点？

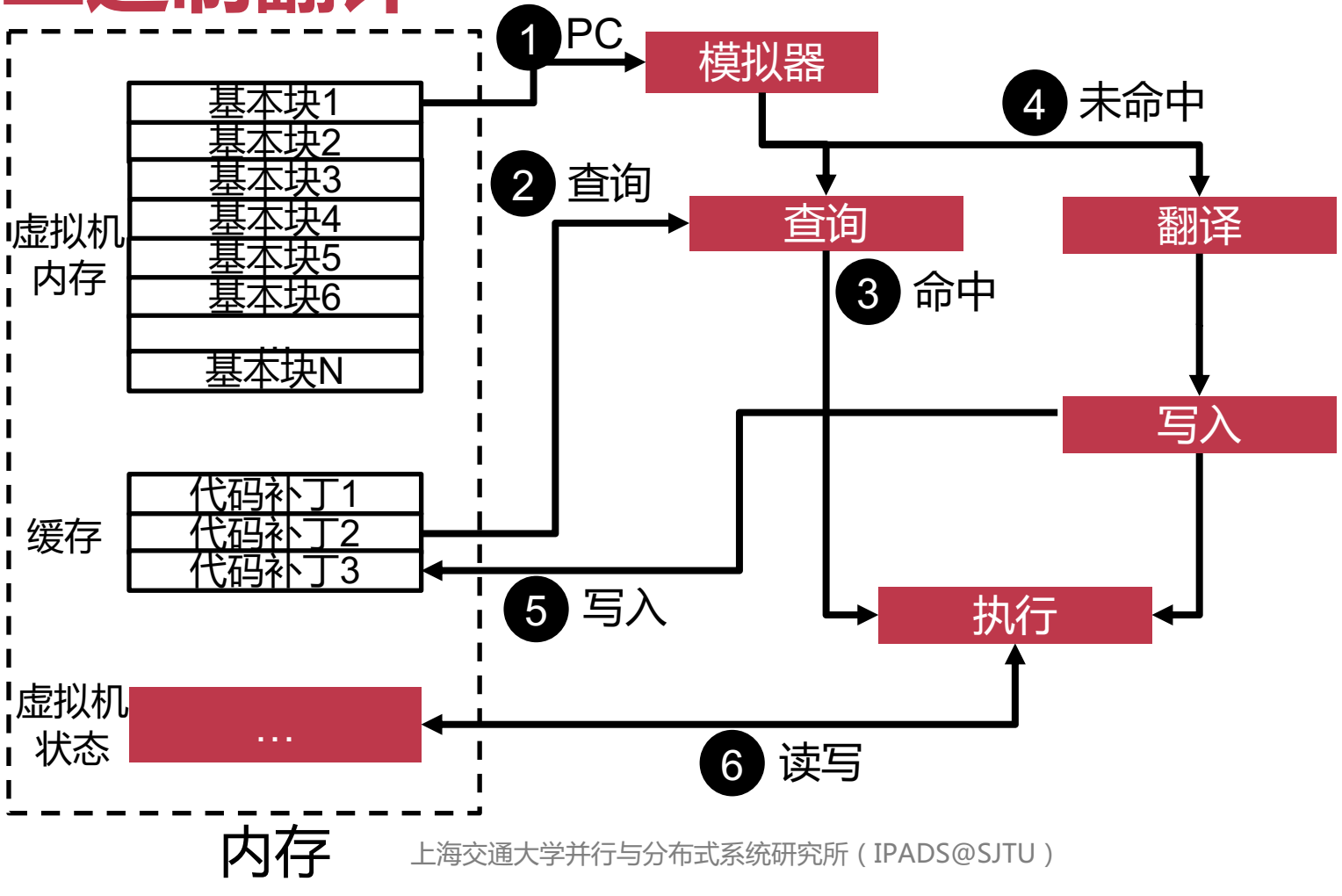
解释执行的优缺点

- **优点：**
  - 解决了敏感函数不下陷的问题
  - 可以模拟不同ISA的虚拟机
  - 易于实现、复杂度低
- **缺点：**
  - 非常慢：任何一条虚拟机指令都会转换成多条模拟指令

方法2：二进制翻译

- **提出两个加速技术**
  - 在执行前批量翻译虚拟机指令
  - 缓存已翻译完成的指令
- **使用基本块(Basic Block)的翻译粒度（为什么？）**
  - 每一个基本块被翻译完后叫代码补丁

二进制翻译


1 PC
模拟器
4 未命中
2 查询
查询
翻译
3 命中
写入
虚拟机
内存
基本块1
基本块2
基本块3
基本块4
基本块5
基本块6
基本块N
缓存
代码补丁1
代码补丁2
代码补丁3
5 写入
执行
虚拟机
状态
…
6 读写
内存

二进制翻译的缺点

- **不能处理自修改的代码(Self-modifying Code)**
- **中断插入粒度变大**
  - 模拟执行可以在任意指令位置插入虚拟中断
  - 二进制翻译时只能在基本块边界插入虚拟中断（为什么？）

方法3：半虚拟化(Para-virtualization)

- **协同设计**
  - 让VMM提供接口给虚拟机，称为**Hypercall**
  - 修改操作系统源码，让其主动调用VMM接口
- **Hypercall可以理解为VMM提供的系统调用**
  - 在ARM中是HVC指令
- **将所有不引起下陷的敏感指令替换成超级调用**
- **思考：这种方式有什么优缺点？**

半虚拟化方法的优缺点

- **优点：**
  - 解决了敏感函数不下陷的问题
  - 协同设计的思想可以提升某些场景下的系统性能
    - I/O等场景
- **缺点：**
  - 需要修改操作系统代码，难以用于闭源系统
  - 即使是开源系统，也难以同时在不同版本中实现

方法4：硬件虚拟化

- **x86和ARM都引入了全新的虚拟化特权级**
- **x86引入了root模式和non-root模式**
  - Intel推出了VT-x硬件虚拟化扩展
  - Root模式是最高特权级别，控制物理资源
  - VMM运行在root模式，虚拟机运行在non-root模式
  - 两个模式内都有4个特权级别：Ring0~Ring3
- **ARM引入了EL2**
  - VMM运行在EL2
  - EL2是最高特权级别，控制物理资源
  - VMM的操作系统和应用程序分别运行在EL1和EL0