2020-2021学年春季研究生选修课程

云计算技术原理

Cloud Computing: Principles and Technologies

教学组：胡春明,沃天宇,林学练,李建欣,李博

2021年3月9日下午14:00-15:35

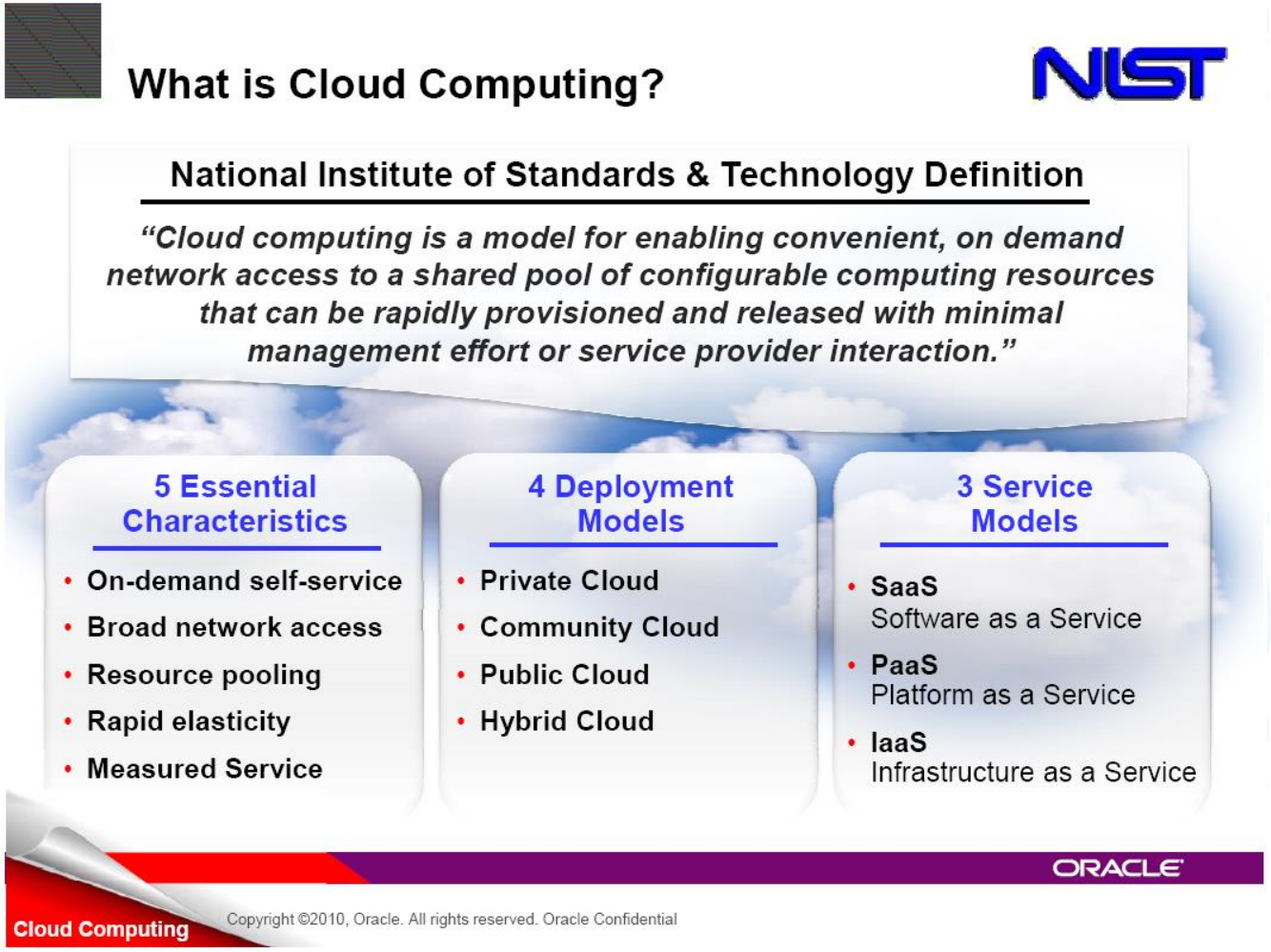
第二讲 系统虚拟化技术

回顾：云计算技术的核心

维护一个计算资源池

通过网络远程访问

快速弹性



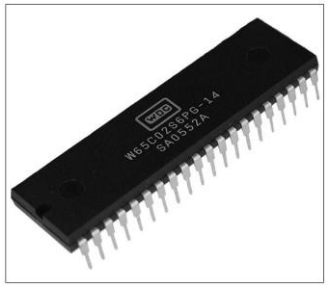
提供一种“机器”的抽象

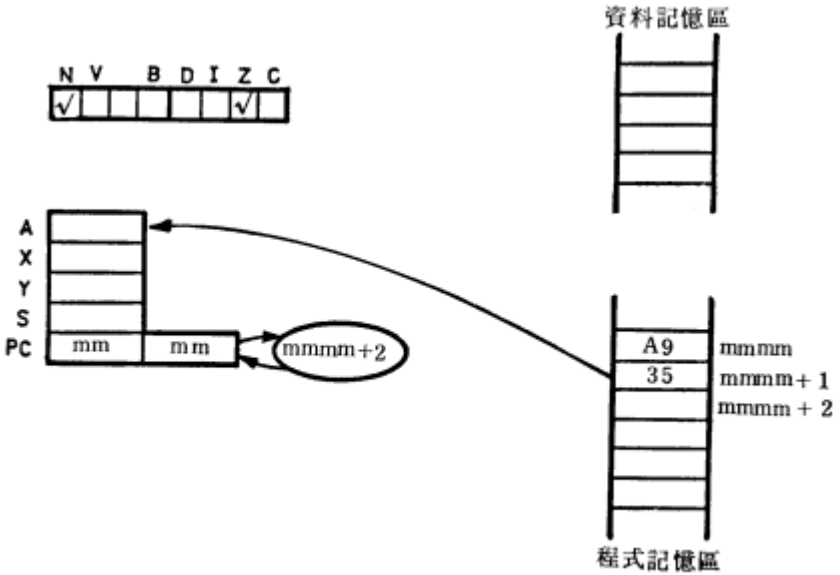
- “机器”的远程访问形式
  - Shell：
  - 远程桌面：RDP、RFB …
- 提供一个“抽象”的机器
  - 一虚多：如何将一台服务器虚拟成多台服务器
  - 多虚一：如何让多台服务器构建Single System Image，形成一台服务器

- CPU：Ricoh 6502 (8位），主频 1.8 MHz
  - Motorola 6502指令集
  - 累加器A、索引寄存器X、Y, 堆栈寄存器S
    状态寄存器F、程序计数器PC
- 内存：2 KB,
- 显存：2 KB
- 显示处理：图像控制器PPU，同时最多显示16色
  - 支持2层卷轴（1 kB）、5个页面（1 KB）
- 声音发生器：
- ROM (8-256KB)
- Motorola 6502指令集

我们来写一个模拟器吧

- 如何在 Linux/Windows操作系统上运行一个ROM





提纲

- **虚拟化原理**
- 主流虚拟机分析（Xen和KVM）
- 虚拟化关键技术及其在数据中心中的应用

为什么会出现“虚拟机”？

Butler Lampson (1992)

All problems in Computer Science can be solved by another level of indirection

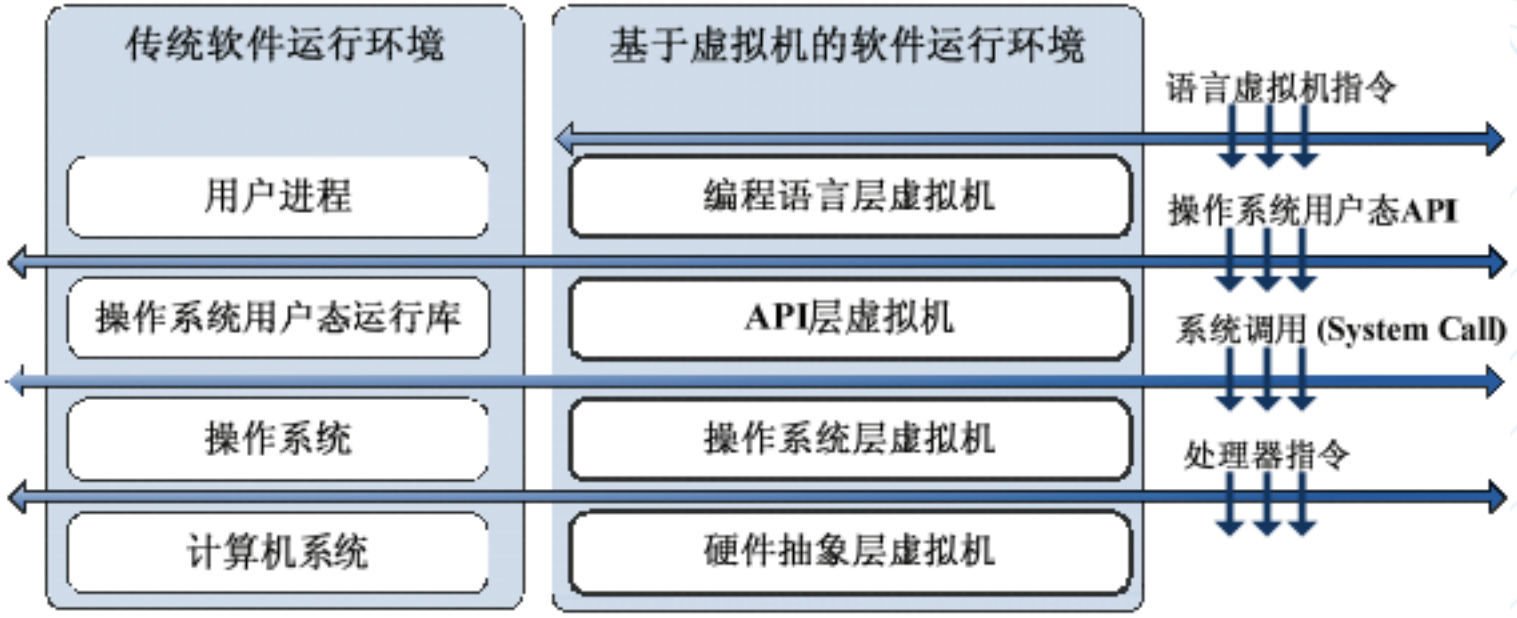
Run Windows 7 in VM under Mac OS X

虚拟化的几个例子

- 模拟器
- Java语言与JVM
- VM

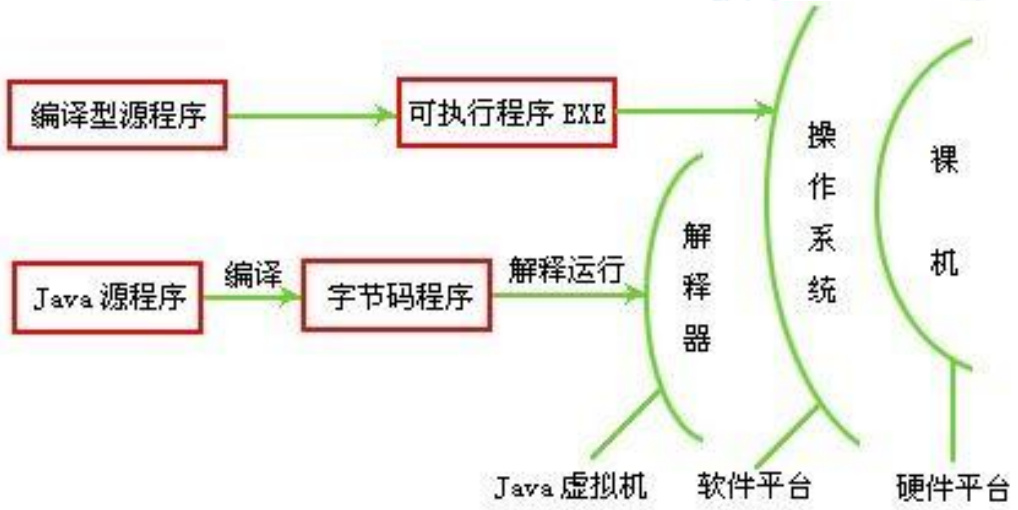
虚拟化分类

- 编程语言层虚拟机(Java)
- API层虚拟机(Cygwin,在Win32系统下实现了POSIX系统调用的API)
- 操作系统层虚拟机(Linux VServer, UML)
- 硬件层虚拟机(XEN,KVM,VMWare,Qemu)

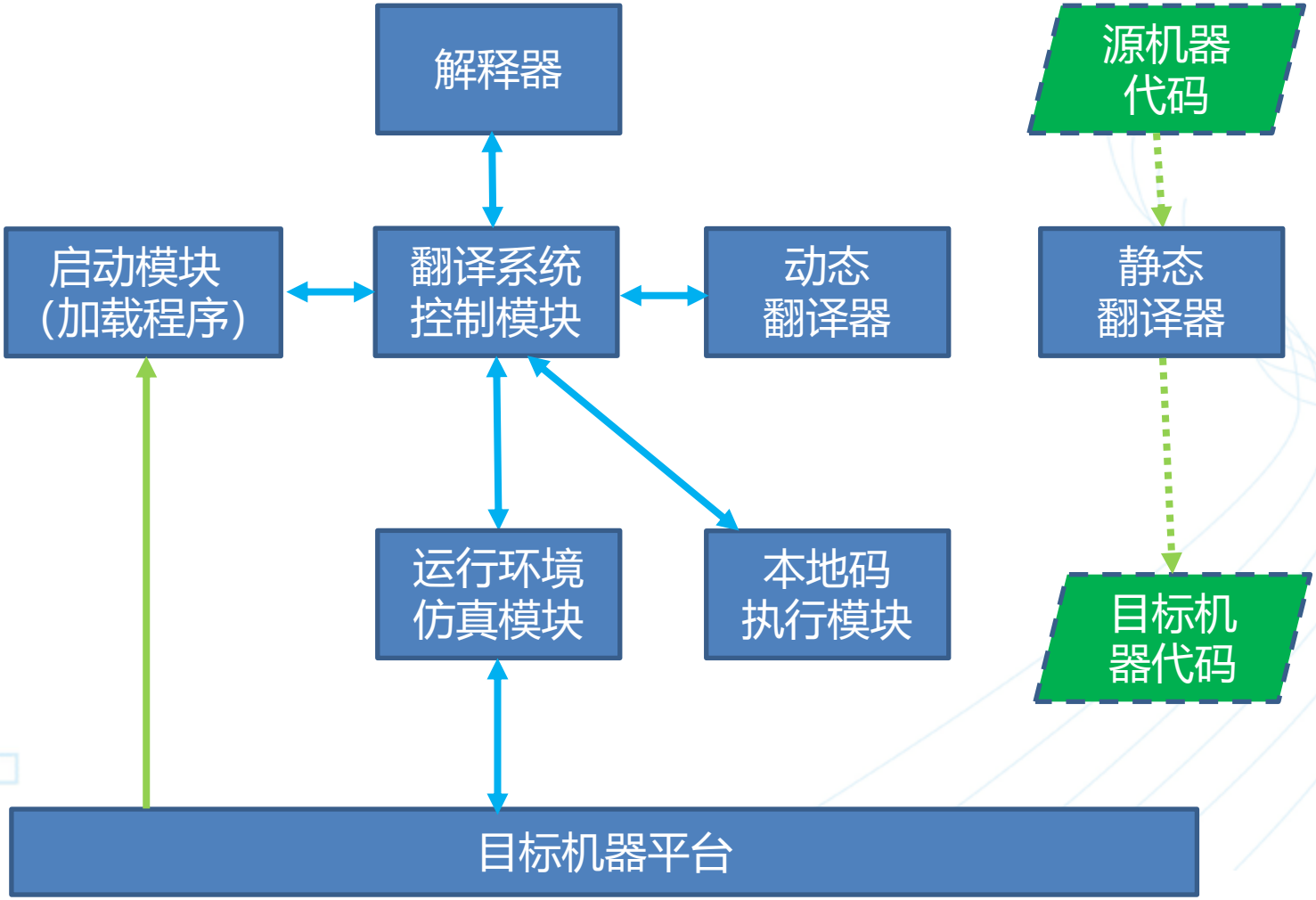


指令级虚拟化

- 二进制翻译：软件方法
  - 解释执行（解释器）
  - 静态二进制翻译
  - 动态二进制翻译



指令级虚拟化



二进制翻译的效率问题

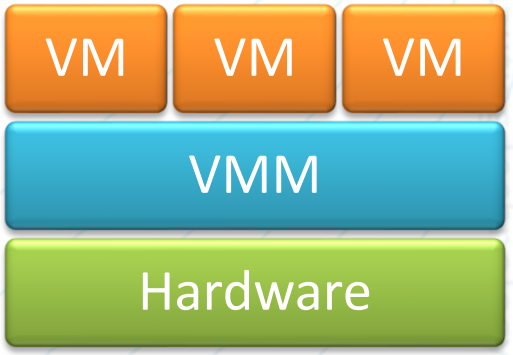
- 磁盘访问开销：磁盘加载到内存
- 存储访问开销：很难充分获得Cache的能力
- 翻译和优化开销：每翻译一条x86指令，平均需要1000条目标机指令
- 目标代码的执行开销：

什么是系统虚拟机

- *Popek and Goldberg(1974)*
  - *虚拟化典型特征*
    - *Fidelity. A VMM must provides an execution environment almost identical to the original machine*
    - *Performance. Large percentage of the virtual processor's instructions must be executed by the machine's real processor, without VMM intervention.*
    - *Safety. A VMM* must be in control of real system resources

VM: a hardware-software duplicate of a real existing computer system in which a statistically dominant subset of the virtual processor's instructions execute on the host processor in native mode

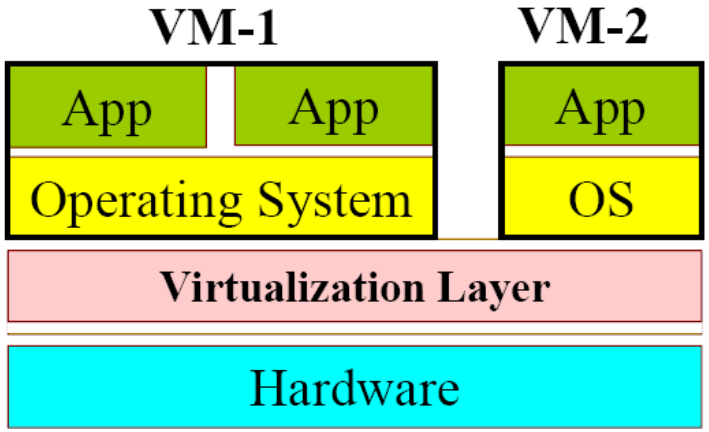
- What a VMM do
  - 管理硬件设备
  - 对上层VM呈现出虚拟的硬件平台



经典虚拟机模型

系统虚拟化技术是

- 虚拟化是在“硬件”和“软件”之间的一种抽象



- 主要的优势：封装、隔离、灵活、便于迁移
  - 有效提高系统可控可管能力
  - 增强系统可信保障能力
- 核心：虚拟机监控器(VMM)
  - 由于硬件提供的虚拟化支持，虚拟化后的性能开销可控
  - 追求的目标：轻载、高效

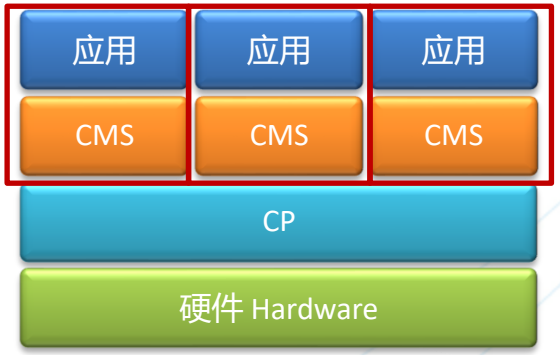
从操作系统说起

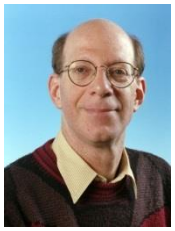
- 批处理系统(1950-1960)
  - 在没有人工参与的情况下，顺序执行一系列的程序
  - 不支持多用户、多程序
- 分时系统(Bob Bemer在1957年首次提出)
  - 批处理系统存在问题
    - 一次执行一个程序，I/O过程CPU空转
    - 同一时刻只能供单一用户使用
  - 将CPU处理时间分割为多个时间片，多个程序"同时"运行
    - 暂停当前程序、进入另一个程序运行 → 中断Interrupt、时间片
    - 一个程序独占资源 vs 多个程序共享资源
      - 应用软件不能访问所有硬件功能，要通过操作系统
  - 代表系统
    - Multics/Unix (1968/1970)
    - IBM VM 360/370 (1966/1972)

两类典型的分时系统

- Multics/Unix
  - 进程
  - 虚拟内存
  - 中断、系统调用

- IBM VM 360/370
  - “虚拟机”在VM/360操作系统中首次提出
  - CP/CMS
  - CMS:单用户单进程操作系统
  - CP: 管理多个CMS





Tanenbaum (1985)

An Operating System is a program that
1. controls the resources of a computer
2. provides its users with an interface or VM

虚拟机发展历程

X86服务器虚拟化

X86虚拟化比小型机虚拟化技术具有更好的通用性，因此X86服务器虚拟化成为云计算时代的关键技术。

2008年，VMware基于X86虚拟化技术推出企业级云计算vCloud计划

2006年，亚马逊发布基于Xen的EC2服务

2003年，剑桥大学发布第一个开源虚拟化软件Xen

2001年，VMware推出ESX Server，开启X86服务器虚拟化时代。

个人计算机虚拟化

1999年，VMware推出面向X86 PC的虚拟化产品Workstation

1997年，Connectix推出了Virtual PC（Mac平台）。

大型机/小型机虚拟化

60年代，IBM最早在7044大型机中引入了虚拟化技术。

1999年，IBM 公司在AS/400 上推出逻辑分区(LPAR)技术

2002年，IBM 公司在AIX5L上推出动态逻辑分区(DLPAR)技术

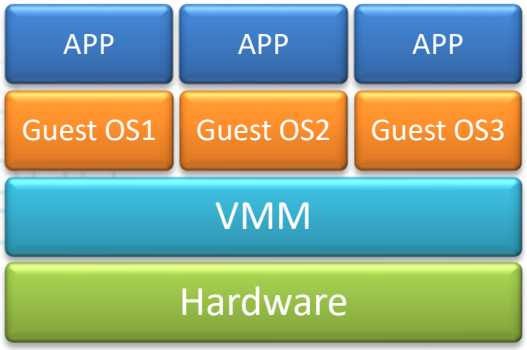
60年代 90年代 2000年代

虚拟机发展历程

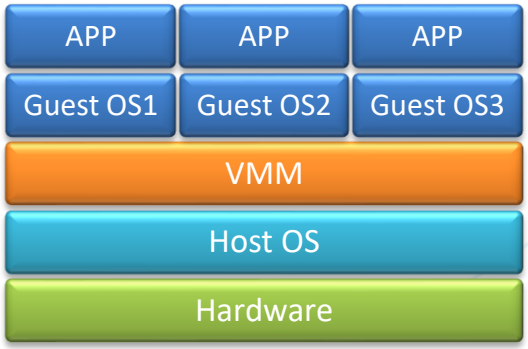
- 20世纪60-80年代：IBM VM 360/370
  - 大型机数量有限，为满足多用户同时访问
  - VM为每个用户提供隔离的运行环境
- 20世纪90年代后期
  - 个人台式机性能足够满足多系统运行，虚拟机又开始升温
  - 1997年，斯坦福大学的Disco系统
  - 1998年，Vmware公司的诞生
  - 1999年, VMware introduced the first x86 virtualization product
- 21世纪
  - 2000: IBM announces z/VM, new version of VM for IBM's 64-bit z/Architecture
  - 2001: Connectix launchs its first version of Virtual PC for Windows
  - 2003:First release of first open-source x86 hypervisor, Xen
  - 2005: Intel and AMD announced CPU extensions to support virtualization
  - 2006: kvm: kernel virtual machine. Entering linux kernel

系统虚拟机分类

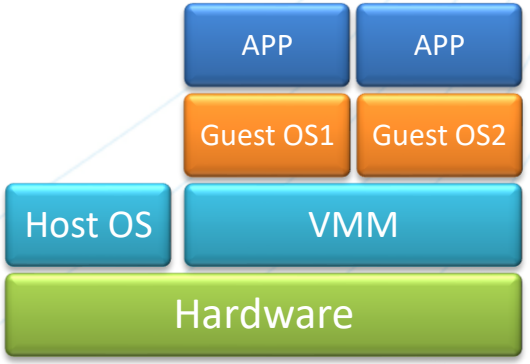
- 按实现技术分类 (主要针对x86)
  - 半虚拟化技术(Xen …)
    - 需要修改客户机操作系统, Hypercall机制
  - 完全虚拟化技术
    - 硬件辅助(hardware-assist)虚拟化 (Xen hvm, KVM …)
      - Add ring -1
    - 动态指令转换虚拟化(VMware)
      - 扫描指令流，识别敏感指令，跳转到等价的模拟指令
- 按体系结构分类



Type I



Type II



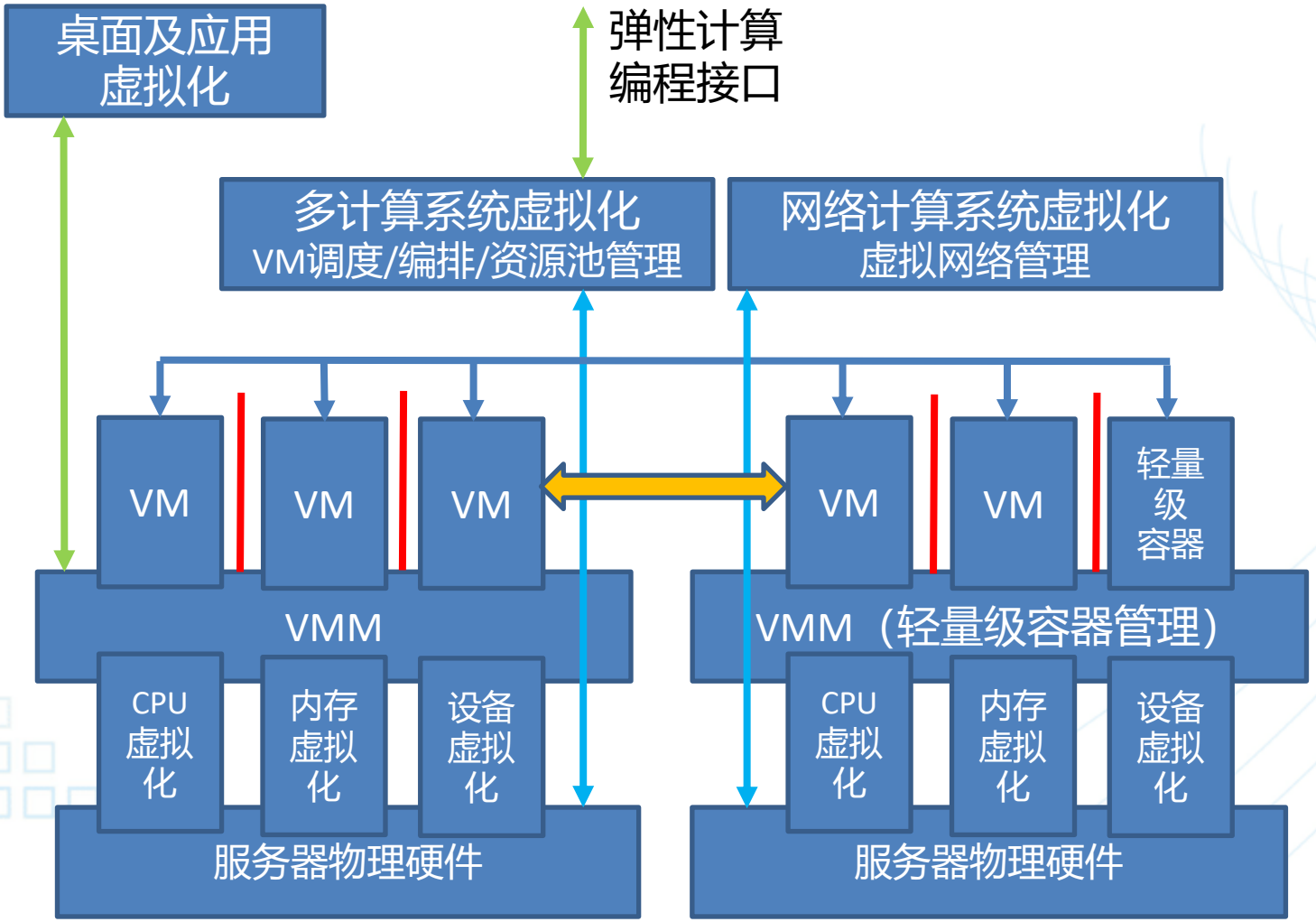
Hybrid

硬件辅助的虚拟化

- 把纯软件虚拟化技术的功能，（全部或部分）用硬件实现，提高虚拟化的性能

- CPU： Intel VT、 AMD SVM
  - 提供新的特权层，专门运行VMM
  - 优化指令集，减少二进制转换的需求
  - 为内存映射提供专门的加速电路
- 网卡、显卡等加速器支持：
  - 提供多组共享单元，支持多虚拟机上下文切换

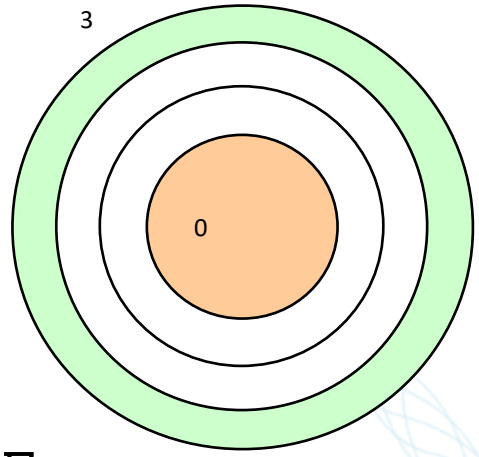
计算系统虚拟化研究



小结

- 到这里，希望你能够对“虚拟化”建立起概念
  - 不同层次的虚拟化
  - 为什么指令集翻译“效率比较低”
  - 好的虚拟化“三原则”
  - 什么是VMM？什么是VM？
  - 半虚拟化 vs 完全虚拟化
  - 硬件辅助的虚拟化

CPU虚拟化原理



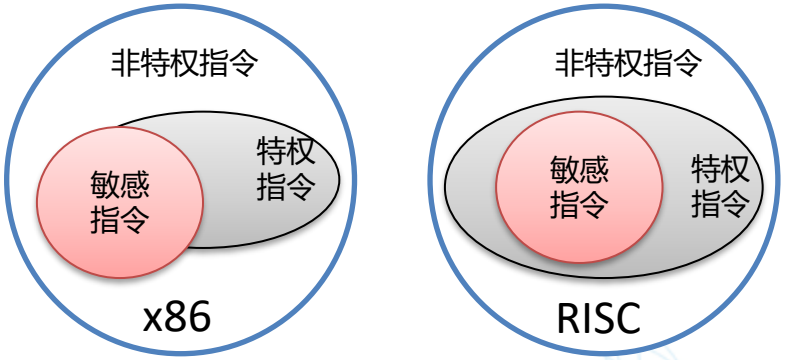
- 传统操作系统中
  - OS运行在特权级别
  - OS负责管理和控制硬件
  - 用户程序和代码运行在非特权级别
- 虚拟机监控器
  - VMM出于隔离和性能的目的要求最高控制权
    - Protect virtual machines from each other
    - Protect VMM from virtual machines
- 虚拟机（虚拟化解除）
  - 将客户操作系统（guest OS）运行在用户态
  - 整个虚拟机运行于非特权级别，与虚拟机监控器区别开
  - 手段：特权环压缩（Ring compression）或虚拟化解除
- 传统VMM的做法(陷入-模拟/Trap and Emulate)
  - 特权指令陷入，并由VMM模拟

CPU虚拟化原理

- 如何确保VMM接管所有“状态和控制”相关指令
  - 特权解除：
  - 陷入-模拟：
- 要求目标平台ISA：
  - CPU必须支持多个特权级（才能“特权解除”）
  - 如果非特权指令（导致改变特权级的指令）都不依赖于CPU的特权级，则不需要额外处理（即执行集支持“特权解除”）
  - 如果敏感指令（访问共享资源或改变共享资源的指令）都是特权指令，则该ISA的计算机支持虚拟化

敏感指令 vs 特权指令

- X86演进
  - 16bit: 8086/8088 (1978)
  - 32bit:80386 (1985)
  - 64bit:x86-64 (2003)



- 指令 （Instructions）
  - 特权指令 vs 非特权指令（Privileged / non-privileged）
  - 敏感指令： 需要处理敏感资源（这些资源可能破坏进程上下文）
    - 例如：POPF、SGDT等
  - 敏感非特权指令： Sensitive but not privileged instructions (SNPI)

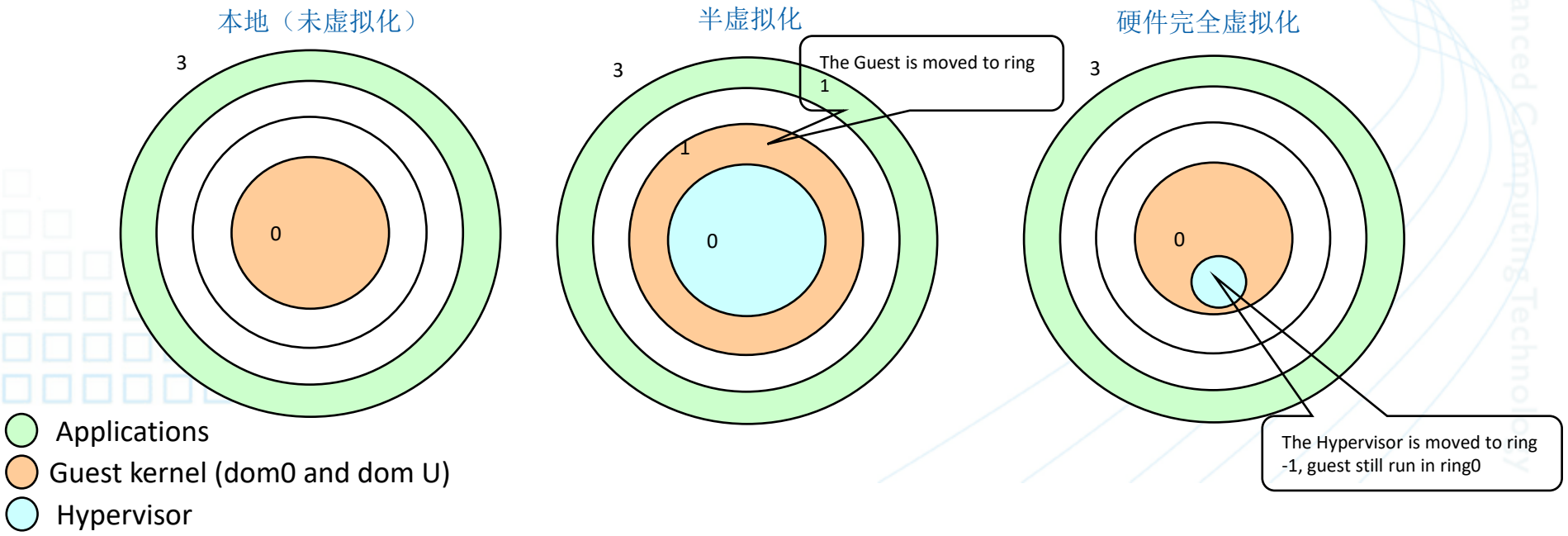
虚拟化漏洞

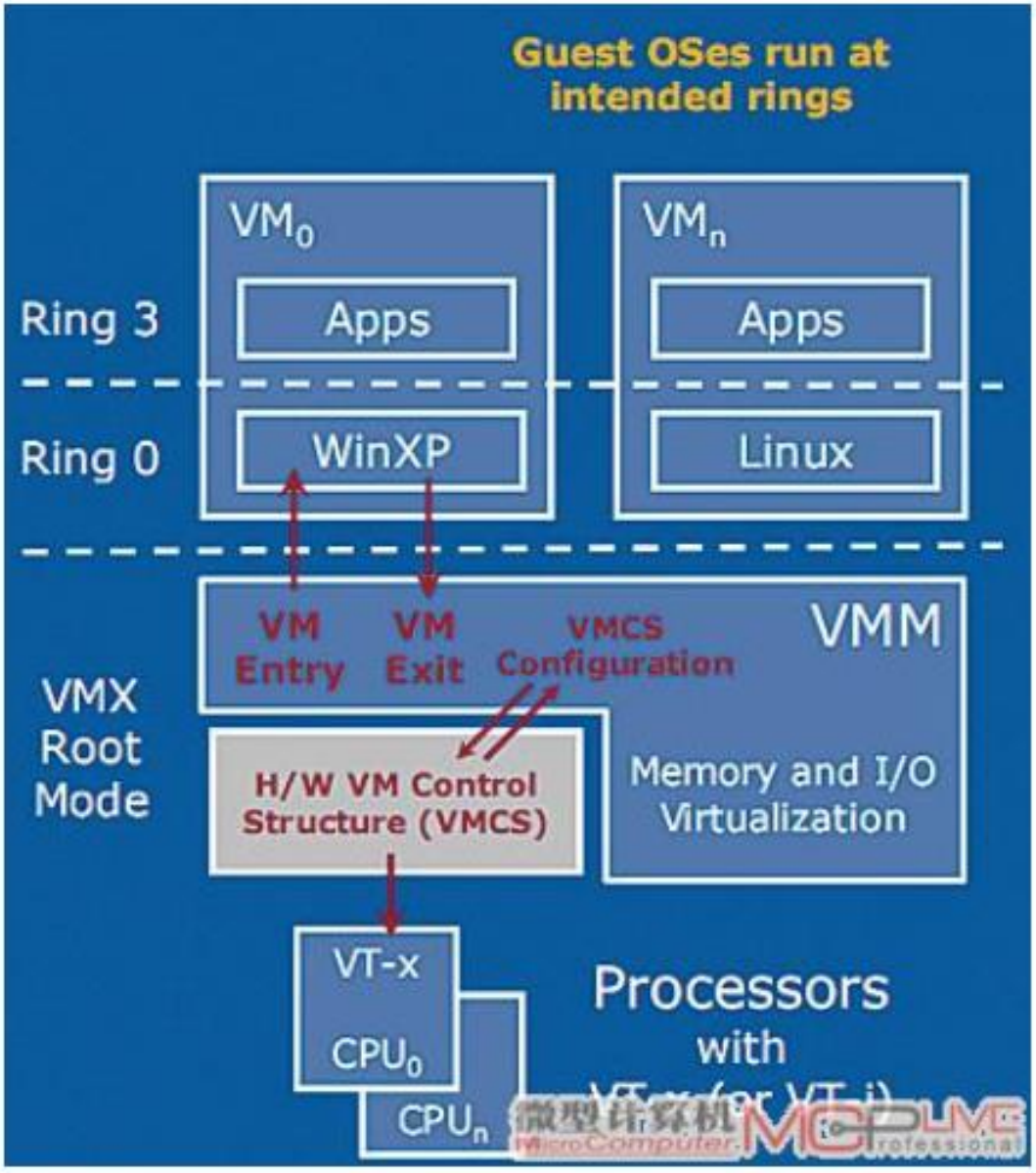
- 虚拟化漏洞 (SNPI not trap in De-privileging)
  - 一部分指令运行结果受到当前特权级的影响：如POPF
  - 一部分访问敏感资源的指令不是特权指令（即敏感非特权指令SNPI）：
  - SNPI 会影响 VMM 或 宿主OS的内部状态处理器，在执行 SNPI 时并不会产生异常
    - SGDT SIDT SLDT SMSW PUSHF POPF LAR LSL VERR VERW POP PUSH CALL JMP INT RET STR MOV
    - SMSW: store machine status VM: real mode VMM: protection mode
    - POPF 允许修改当前标志寄存器EFLAGS register的某些位

- 特权解除带来的问题
  - Ring aliasing, nonfaulting access to privileges state, ring compression
  - Adverse impacts on guest transitions (sysenter)

CPU 虚拟化

- Native: Ring 0运行特权指令； Ring 3运行非特权指令
- 二进制翻译：扫描指令流，识别敏感指令，跳转到等价的模拟指令
- 半虚拟化（Para-Virtualization）：将Guest kernel移到 ring 1 (Xen)
- 硬件辅助虚拟化（Hardware-Assist）： Kvm Hypervisor运行在ring -1（需要CPU硬件支持)





实现“陷入-模拟”的技术手段

- 二进制代码动态翻译
  - 如VMWare，在敏感指令前，插入“陷入”指令
  - 模拟完敏感指令后，插入“退出”指令
- 半虚拟化（Para-Virtualization）
  - 半：修改GuestOS源码
  - Xen：将特权操作替换为一个系统调用Hypercall，强制将guestOS控制权转移给VMM，模拟完成再返回
  - 提供异步消息机制，取代中断，对系统事件进行屏蔽
- 预虚拟化：
  - 将敏感指令在编译时替换为相关调用

实现“陷入-模拟”的技术手段

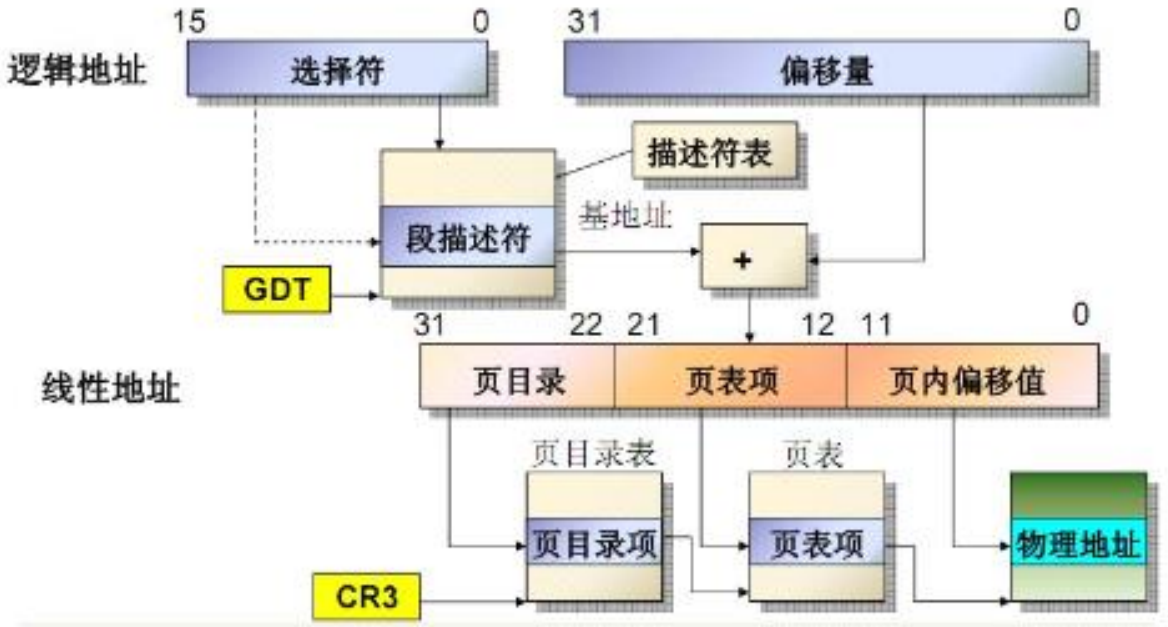
- 硬件辅助的CPU虚拟化
  - Intel VT-x, Intel VT-I （Vanderpool）
  - 增加两种工作模式：root, non-root
  - 增加VMXON指令、VMXOFF指令：进入和退出虚拟化状态
  - 增加VMEntry、VMResume指令，VMExit指令：进入和退出VM
  - 增加VMCS（虚拟机控制结构），管理并保存VM状态
    - Guest-state area, host-state area
    - VM-exit, VM-entry控制域、信息域

虚拟CPU的调度

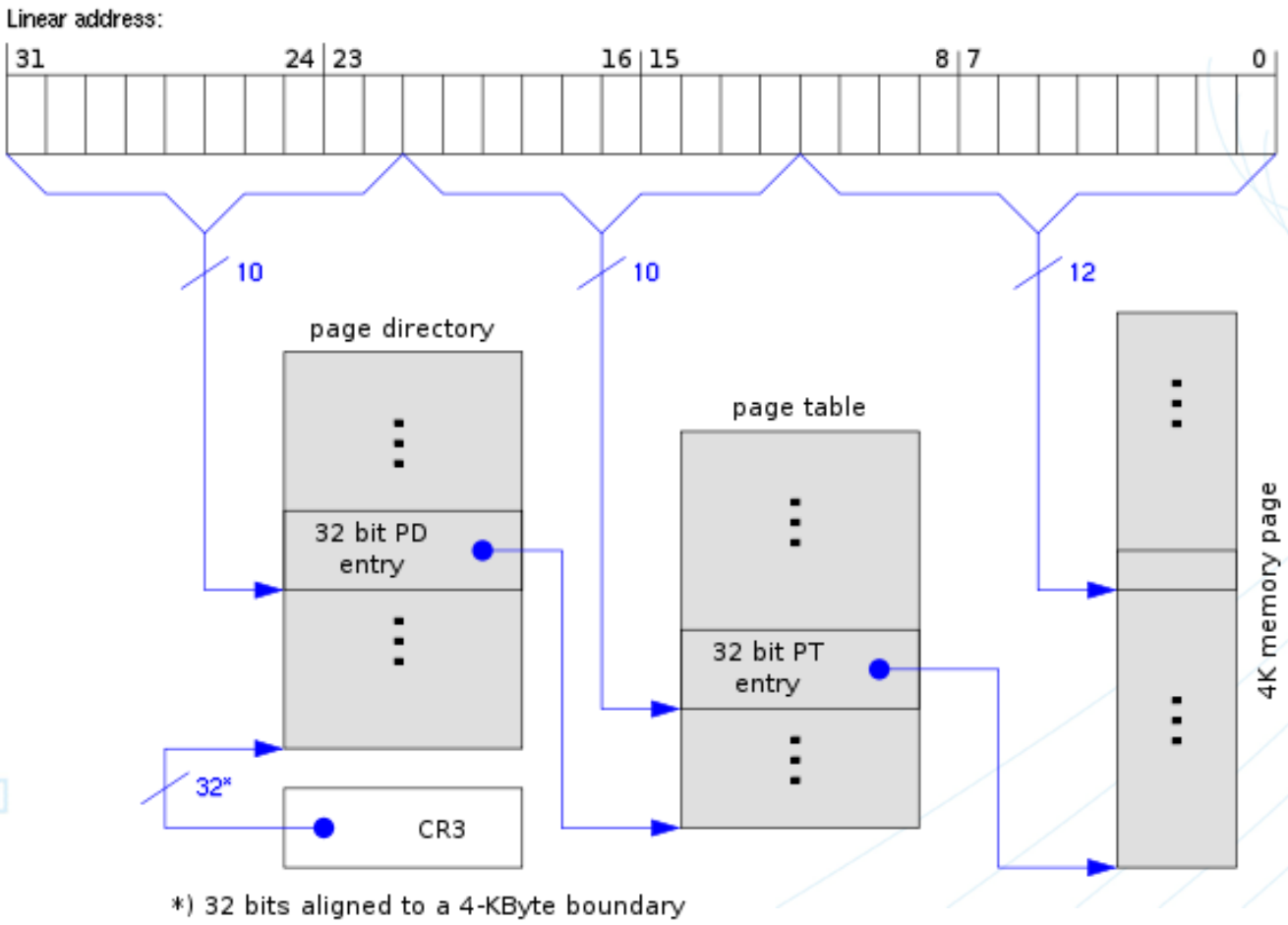
- 需求：
  - 充分利用CPU资源
  - 精确在多个VM间分配资源（时间片）
  - 性能隔离（一个VCPU不能影响其他的VCPU）
  - 虚拟机的公平
  - 虚拟机之间的依赖关系
- 常见算法：
  - BVT（1999）
  - sEDF（1970s，Xen 3.0使用）
  - Credit based调度（比例分配）

内存虚拟化

- 现代操作系统的基本内存管理（段页式内存管理）
  - 段+段偏移量
  - 页表+页内偏移量
  - 实模式到保护模式的进步

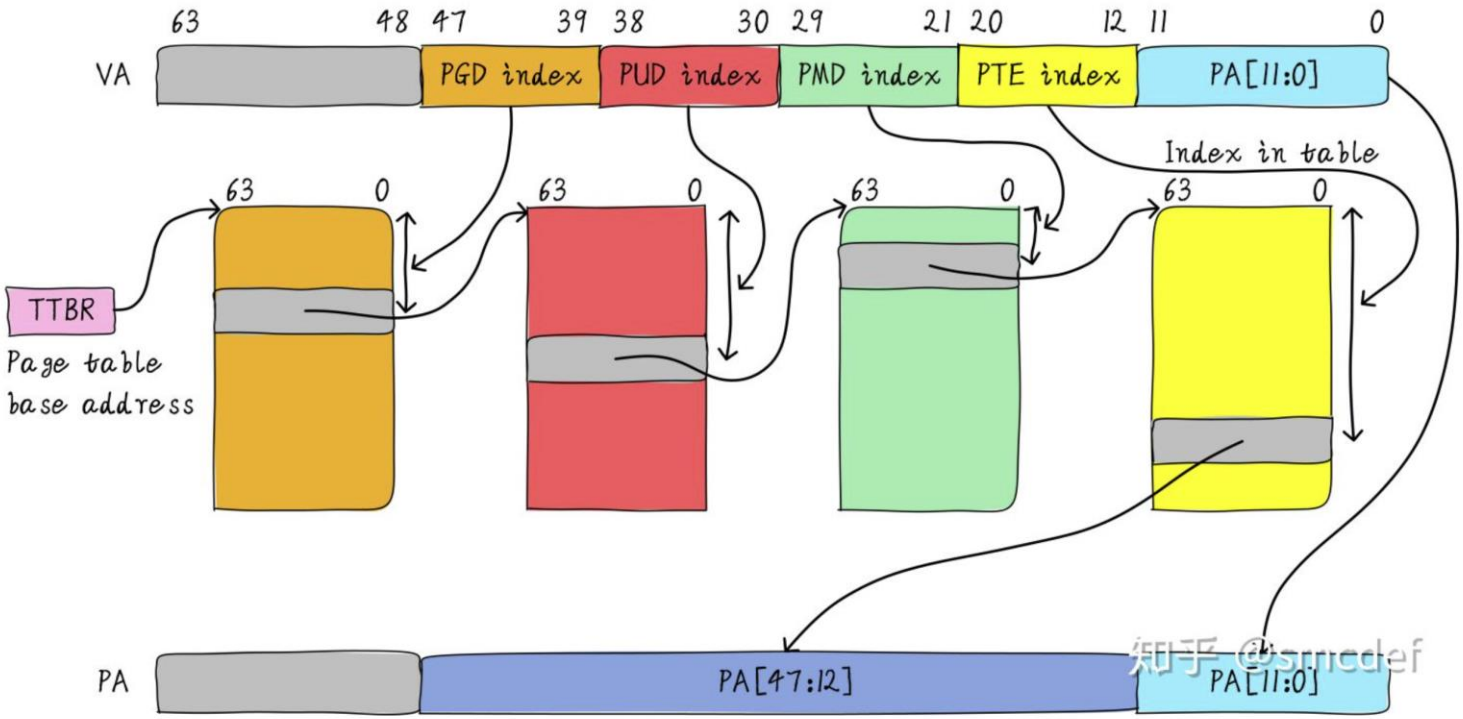


页表 (x86-32 non-PAE)



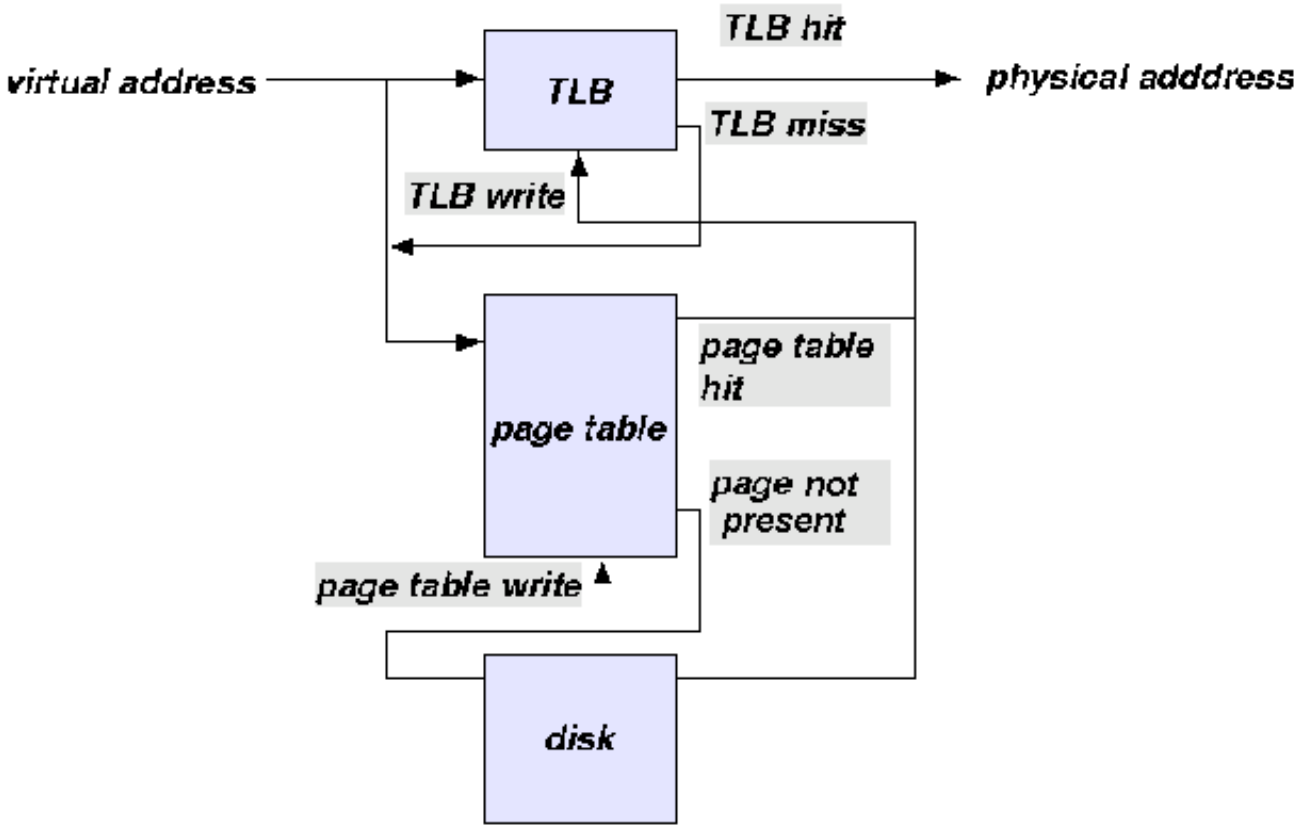
OS的内存管理

- MMU：把物理地址转换为虚拟地址
  - 虚拟地址（VA）：应用程序看到的连续地址空间
  - 物理地址（PA）：操作系统看到的物理地址



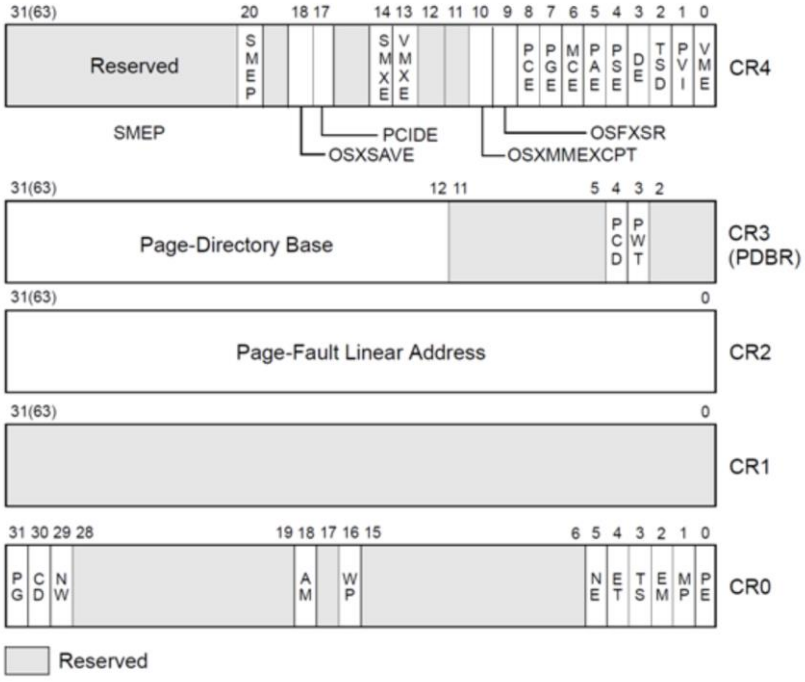
内存虚拟化

- TLB：硬件的高速缓存



内存虚拟化

- TLB：硬件的高速缓存
- 进程切换时：OS会将待切换的进程对应的页表基地址写入CR3寄存器（从而触发进程上下文切换）



切换页目录基地址（从而将进程空间映射到一组新的物理地址空间）

切换内核态堆栈和硬件上下文（包括CPU寄存器等）

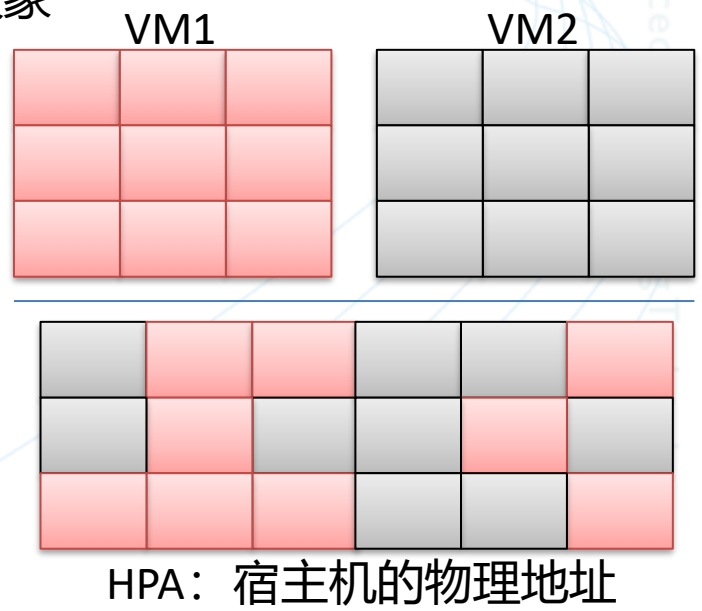
内存虚拟化

- 现代操作系统的基本内存管理（段页式内存管理）
- 内存分页保护机制/虚拟内存

- 虚拟化下面的内存管理
  - HPA：机器地址，或宿主机物理地址
  - GPA：物理地址，经过VMM抽象的、虚拟机看到的伪物理地址
  - GVA：GuestOS给应用程序提供的线性空间

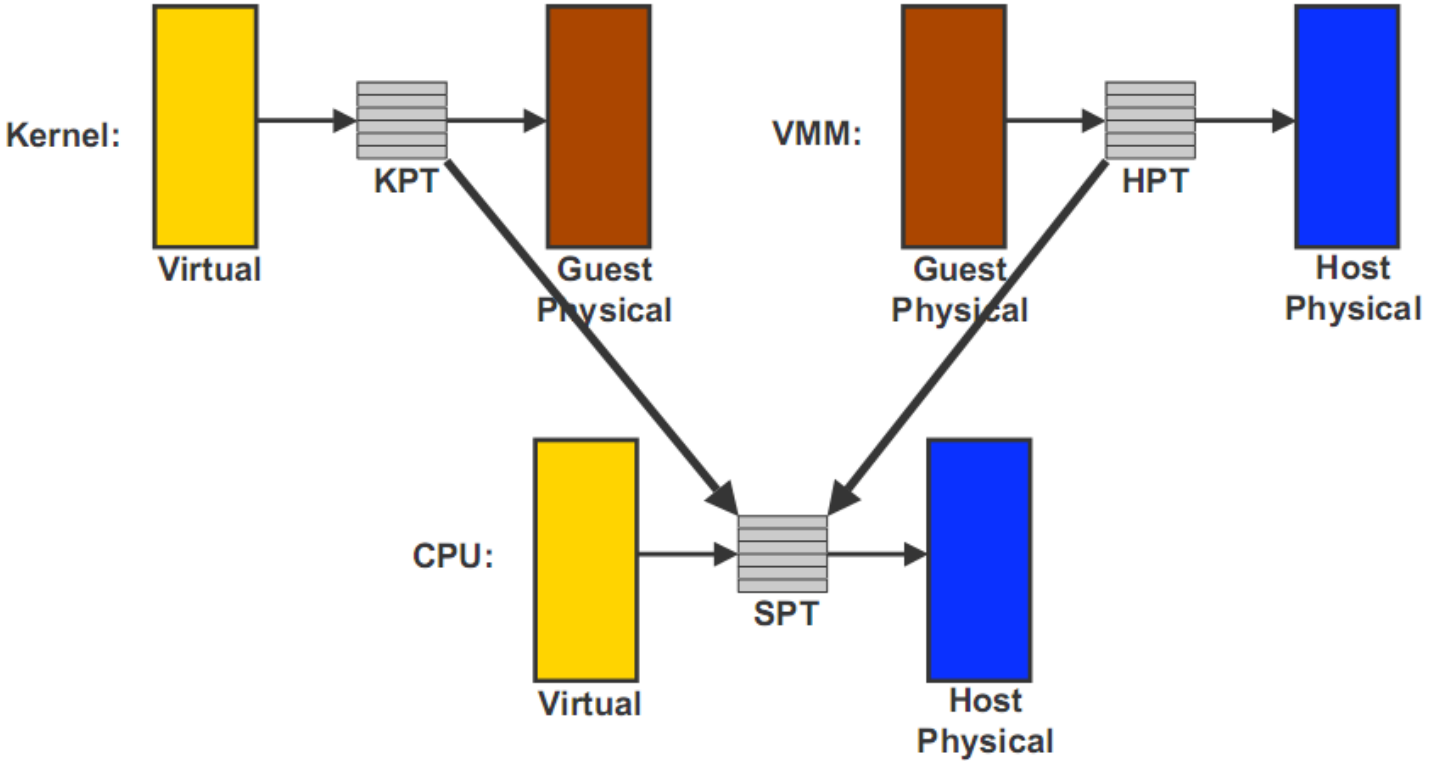
内存虚拟化

- 对于虚拟机：
  - 物理地址起始位置都是0，且是连续的
  - VM对所有内存区域有控制权
- 对于VMM（虚拟机监控器）
  - VMM可以看到全部物理地址
  - VMM需要给VM制造“掌控全部地址空间”的假象
  - VMM需要保护自己的数据不被VM访问
  - 需要分配和管理内存
- VM内存地址的映射
  - GVA -> GPA -> HPA
- 关键问题
  - 维护从 GPA 到 HPA mapping
  - 截获VM中的进程对GPA的访问，并做转换



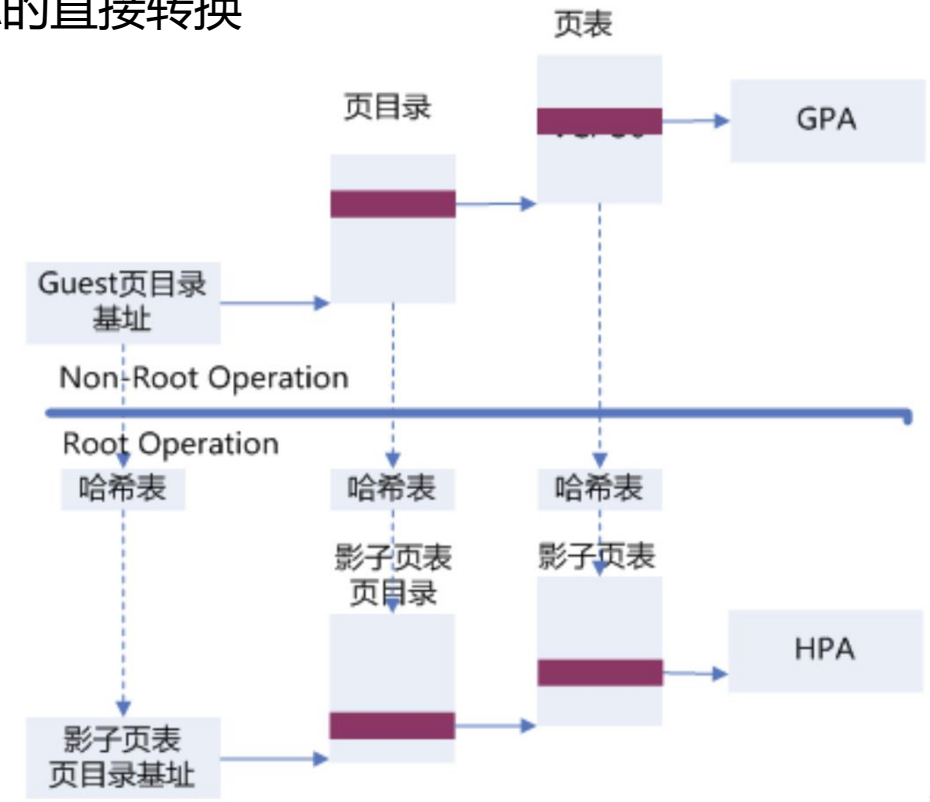
影子页表（Shadow Page Table）

- Virtual MMU for guest VM
  - 为GustVM中的每个进程维护一个影子页表SPT
  - GuestVM访问内存时，VMM在物理MMU中加载对应的影子页表，实现GVA到HPA的直接转换



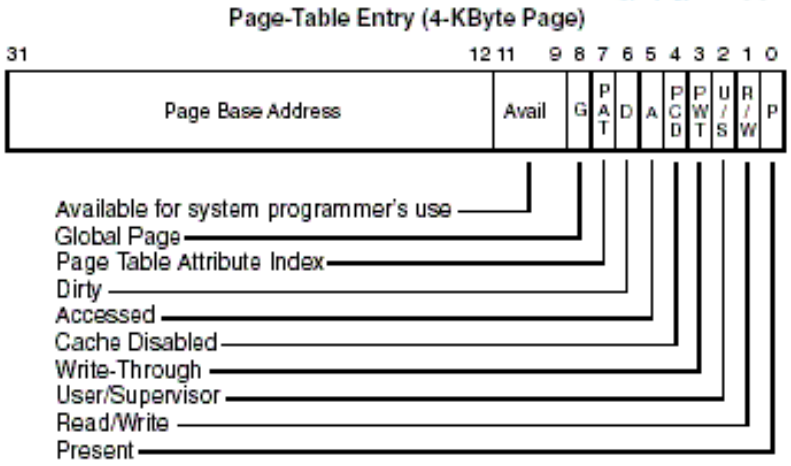
影子页表（Shadow Page Table）

- Virtual MMU for guest VM
  - 为GustVM中的每个进程维护一个影子页表SPT
  - GuestVM访问内存时，VMM在物理MMU中加载对应的影子页表，实现GVA到HPA的直接转换



Sync SPT and GPT

- Guest 的页表（GPT）会因GuestVM的运行过程而动态改变
  - VMM 需要维护影子页表SPT与GPT的一致性
    - 如何截获对Guest 页表（GPT）的访问
    - 如何同步 SPT 与 GPT
  - 硬件TLB 的更新
    - Write CR3: (1) identical value (2)not identical
    - Invlpg: 页表项修改
  - 客户OS修改内存模式
    - Mov CR0
- 例如：页表的脏页位
  - Guest VM 更新一个页，同时更新 GPTE
  - GPTE.dirty=1 -> SPTE.dirty=1
- TLB 刷新
  - CR3 和 invlpg 都是特权指令
  - 这一过程可以被VMM截获

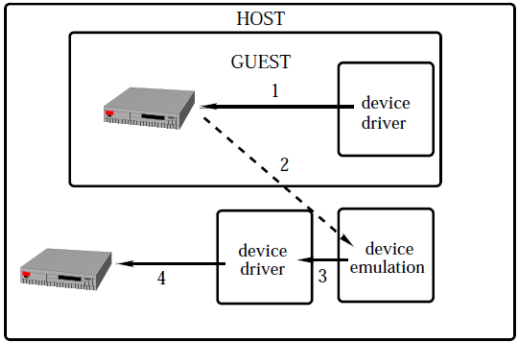


同步模式

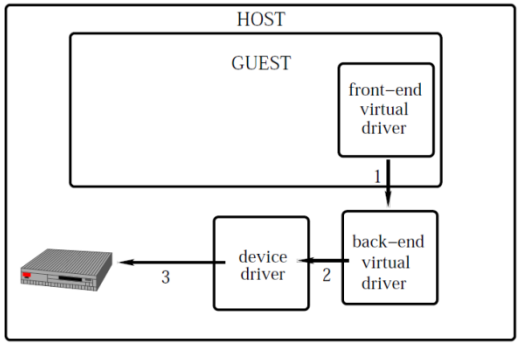
- Lazy方式:
  - 当Guest OS修改了客户页表时,不进行模拟:
    - 1) 访问权限下降W->R：客户操作系统会使用INVLPG进行TLB刷新,hypervisor通过捕获INVLPG指令进行SPT/GPT同步
    - 2) 访问权限上升R->W：客户操作系统再次访问该页面时会引发#PF，VMM通过捕获#PF实现SPT/GPT同步
- Eager方式:
  - Hypervisor监视客户操作系统页表:
    - 通过扫描客户页表层次,使客户操作页表只读
  - 当Guest OS试图修改客户页表GPT时,引发#PF:
    - VMM捕获#PF,对指令进行译码,模拟指令的执行进行SPT同步,确保SPT/GPT总是一致的.
- 同步开销大

I/O 虚拟化

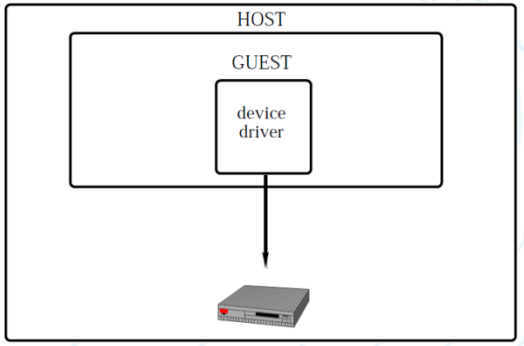
- 虚拟设备（Virtual Device）
  - 设备模型：Device Model
    - logic module in VMM to emulate device and handle dev request and response
    - Can be different from real device. For example, VM->scsi disk, host->ide disk
  - 设备访问接口：
    - A PCI Device: (1)config;(2)PIO; (3)MMIO; (4) DMA; (5) Interrupt
  - 访问的截获和模拟：



Emulation
(1)Guest writes to register 0x789 of emulated dev
(2) Trap to VMM: guest want to send a packet
(3) Device model sends packet to HDD
(4) Host device driver write register 0x789



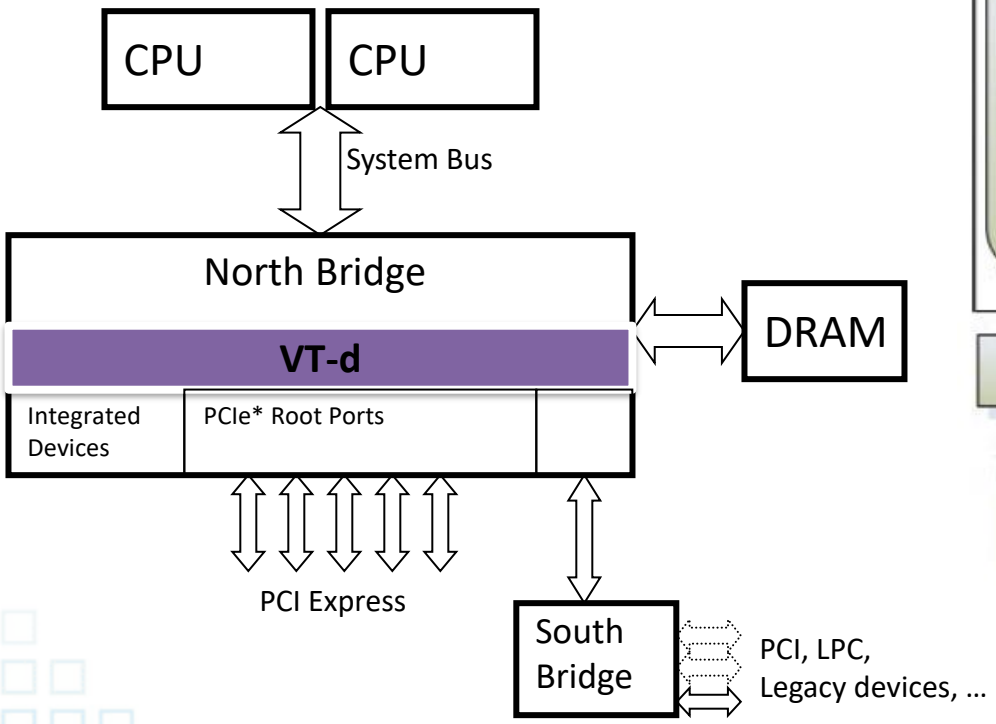
Virtual Split Driver
(1)Guest send a packet
(2)Host transfer the packet to HDD
(3) HDD write to register of real dev

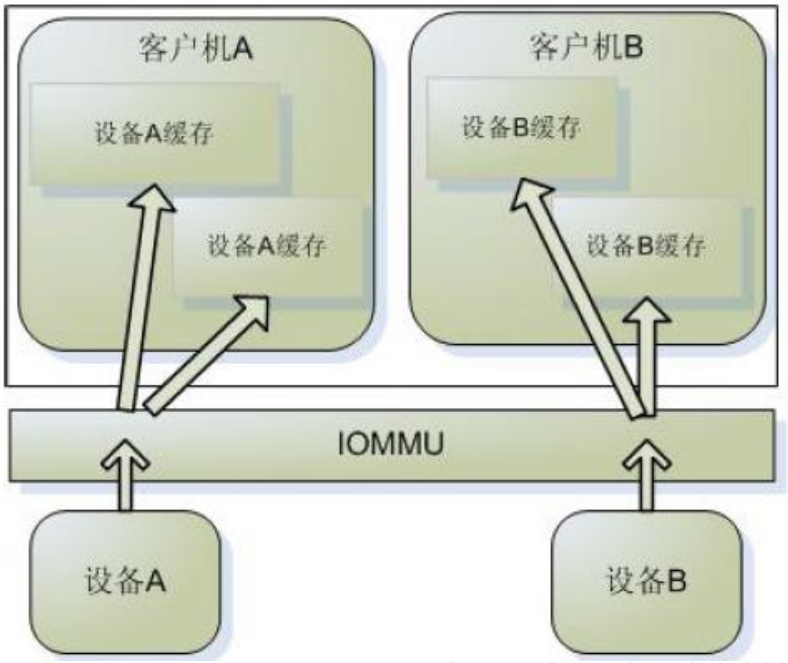


Direct I/O or Pass-through
(1)Guest write register 0x789 directly

硬件支持的DMA重映射

- VT-d





请参考“Intel Virtualization Technology for Directed I/O Architecture Specification”

小结

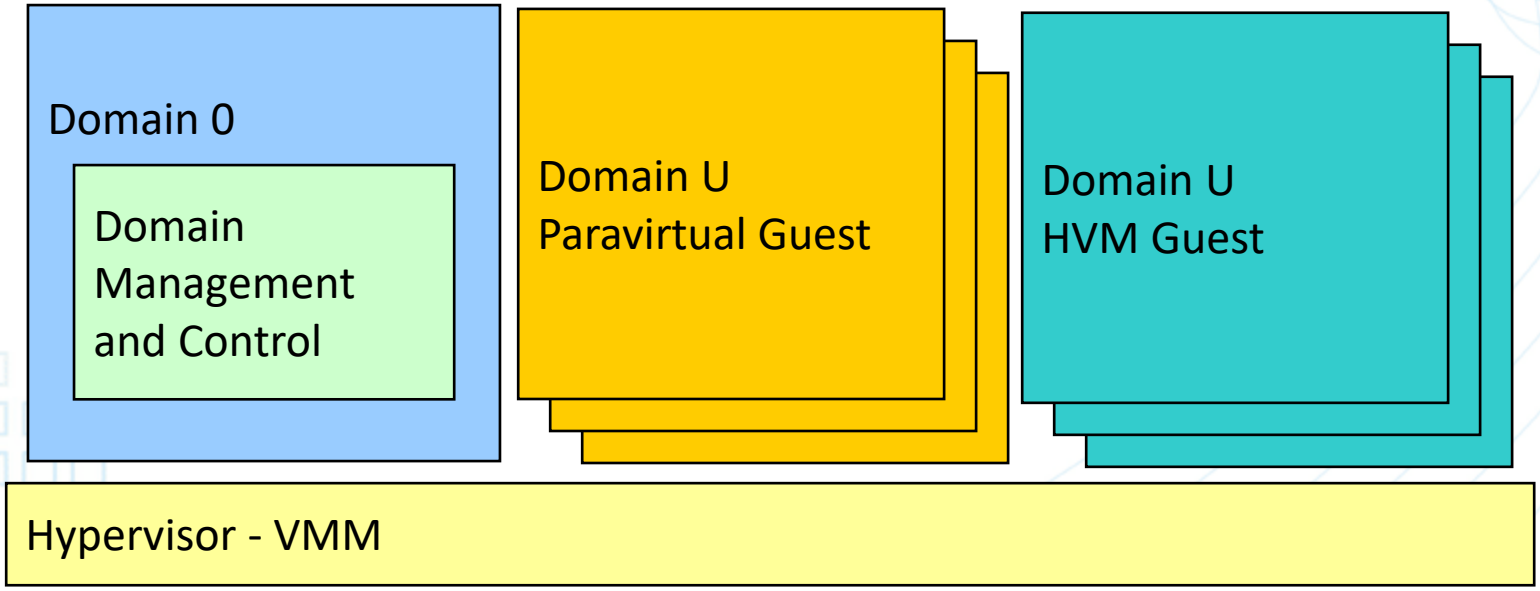
- CPU虚拟化
  - 指令的模拟（虚拟化解除+陷入再模拟）
  - 重点处理什么样的指令（敏感？特权？）
- 内存虚拟化
  - 理解地址的二层映射： GVA -> GPA -> HPA
  - 影子页表 + 硬件实现 (如 Intel的： EPT)
- 设备虚拟化
  - 三种模型

报告提纲

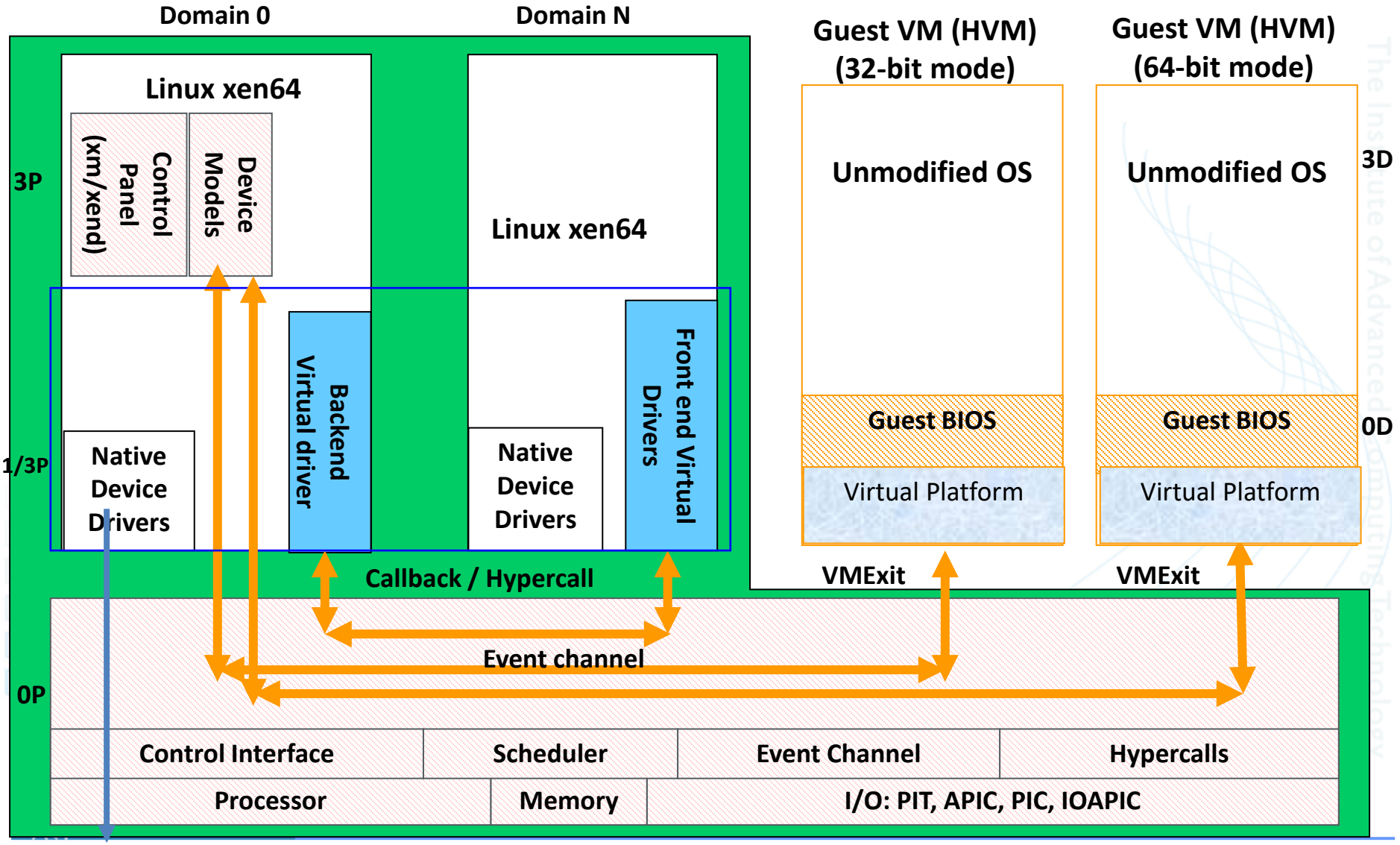
- 虚拟化原理
- **主流虚拟机分析（Xen和KVM）**
- 虚拟化关键技术及其在数据中心中的应用

XEN Overview

- Xen虚拟环境由以下几部分组成:
  - Xen Hypervisor – 硬件启动时首先加载
  - Domain 0 – 虚拟机管理与设备访问
  - Domain User
    - Paravirtual guest (需要修改操作系统)
    - HVM guest (需要硬件虚拟化支持)



XEN Architecture

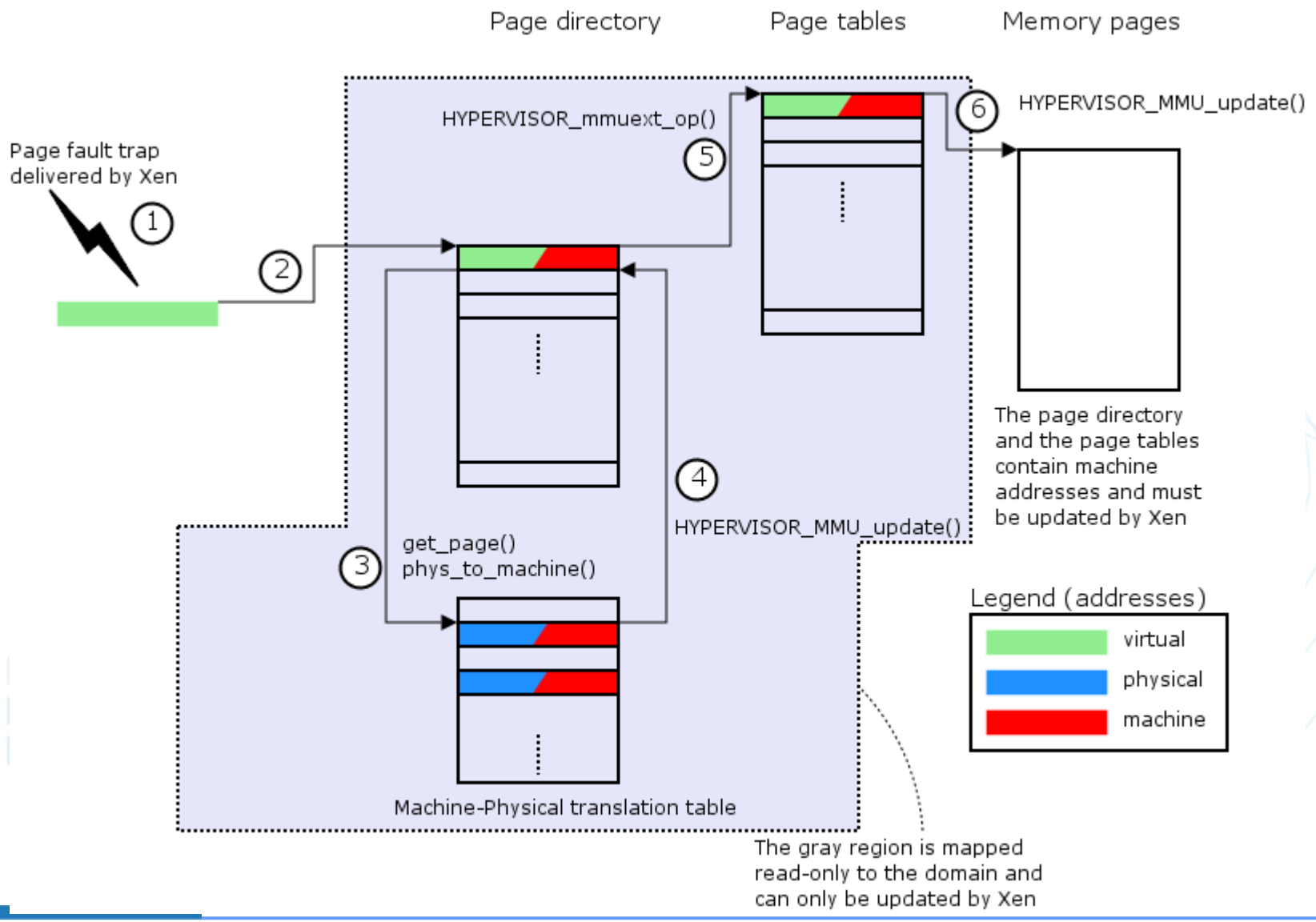


Xen Management and Control

The Domain Management and Control is composed of Linux daemons and tools:

- Xm
  - Command line tool and passes user input to Xend through XML RPC
- Xend
  - Python application that is considered the system manager for the Xen environment
- Libxenctrl
  - A C library that allows Xend to talk with the Xen hypervisor via Domain 0 (privcmd driver delivers the request to the hypervisor)
- Xenstored
  - Maintains a registry of information including memory and event channel links between Domain 0 and all other Domains
- Qemu-dm
  - Supports HVM Guests for networking and disk access requests

Xen Memory Management



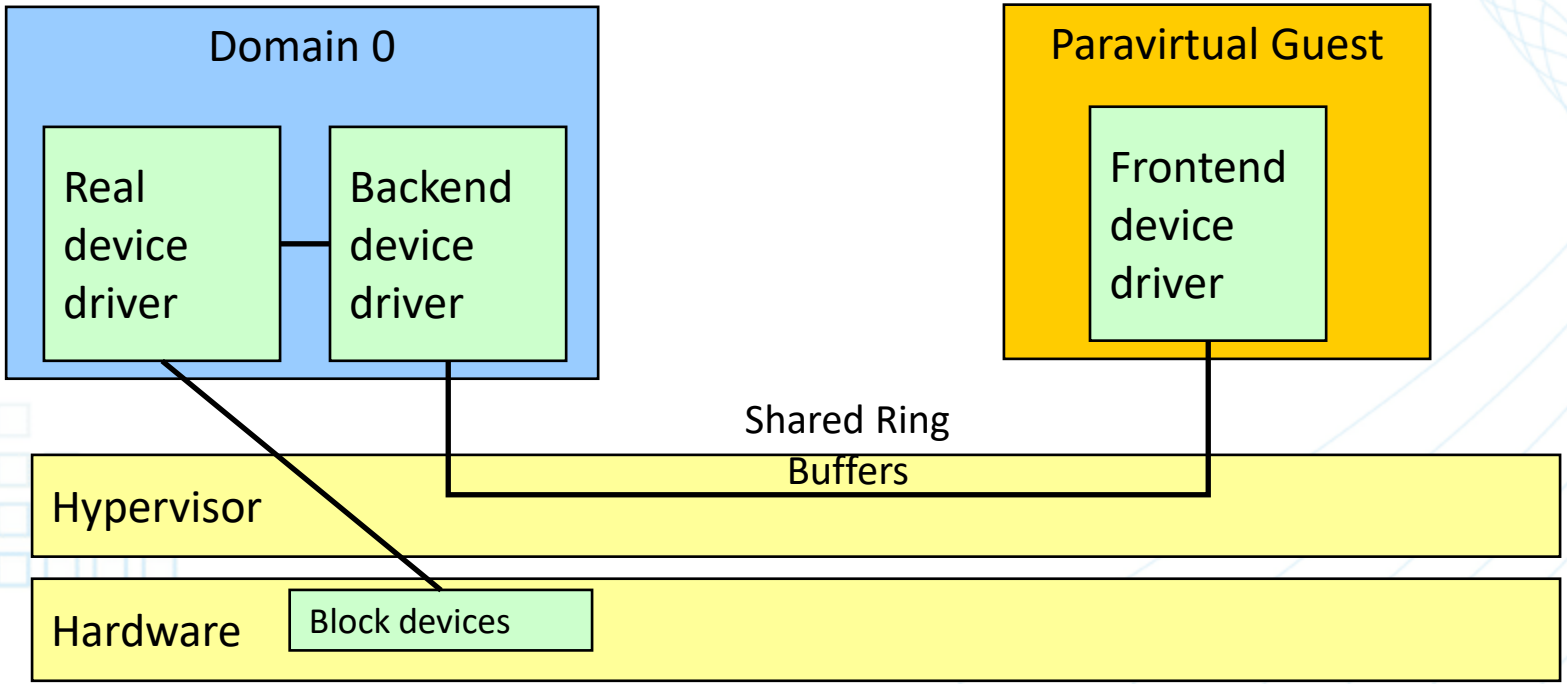
Event Channel

- Event指包括中断在内的异步消息的抽象
- Event Type
  - pIRQ: interrupt from physical device，如网络包到达
  - vIRQ: interrupt from VMM，如VMM模拟生成的虚拟机时间中断
  - IDI: inter-domain interrupt，双向跨guest VM通信
  - IADI: ***i***ntr***a***-***d***omain ***i***nterrupt (vIPI)，同一VM不同vCPU间同步
- 中断处理流程
  - 中断发生，硬件根据IDT表调用Xen注册的中断处理函数
  - Xen通过event channel将中断事件通知给客户机，由其中断处理函数处理
  - 在该event channel的事件向量中pending位置1
    - 该VM正在执行，则打断执行，注入终端
    - 该VM没在执行，则xen调度该VM，执行VM注册的终端处理程序

Xen Split Device Driver Model (for PV guests)

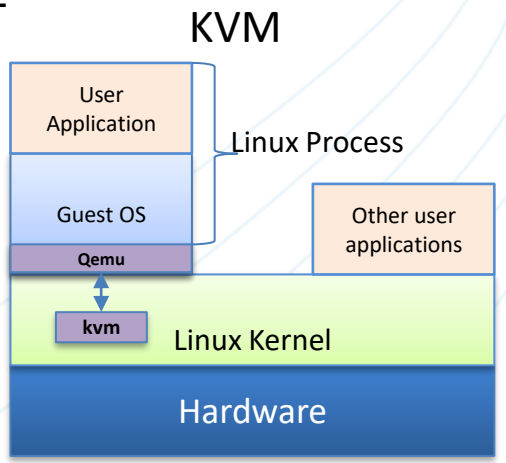
Xen delegates hardware support typically to Domain 0, and device drivers typically consist of four main components:

- The real driver
- The back end split driver
- A shared ring buffer (shared memory pages and events notification)
- The front end split driver



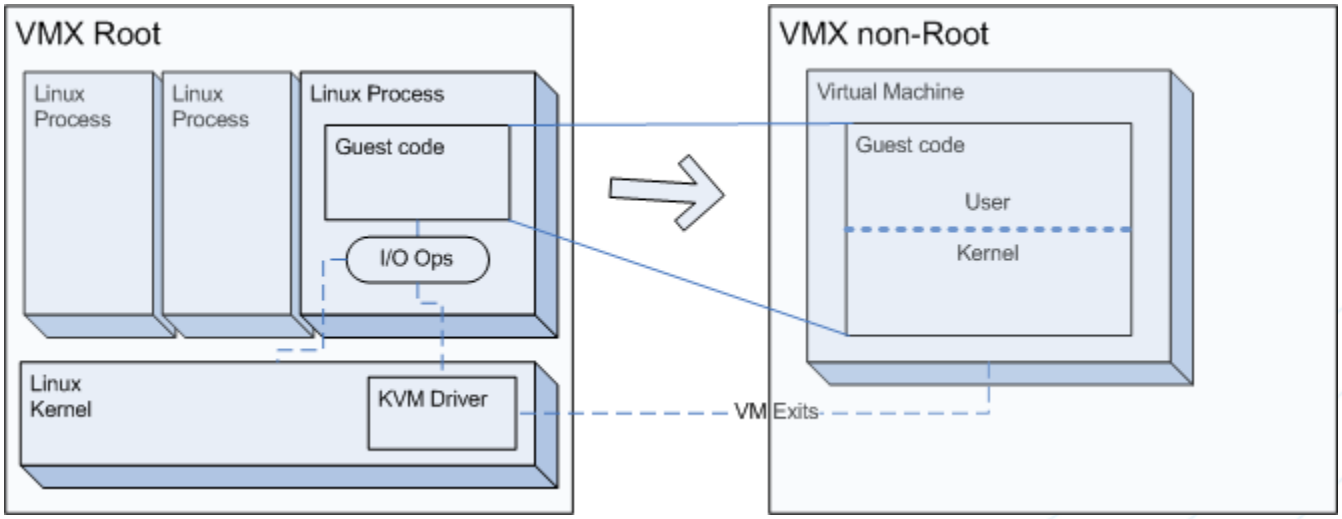
KVM Overview

- KVM虚拟环境由以下几部分组成：
  - Kvm内核模块
    - 执行硬件虚拟化相关操作
    - 虚拟机执行监控
  - Kvm用户程序（大部分代码来自Qemu）
    - 虚拟机管理接口（启动、停止、休眠等）
    - 外设模拟：模拟guest虚拟机的I/O操作等
  - Guest虚拟机
    - 运行在客户模式下



How it works

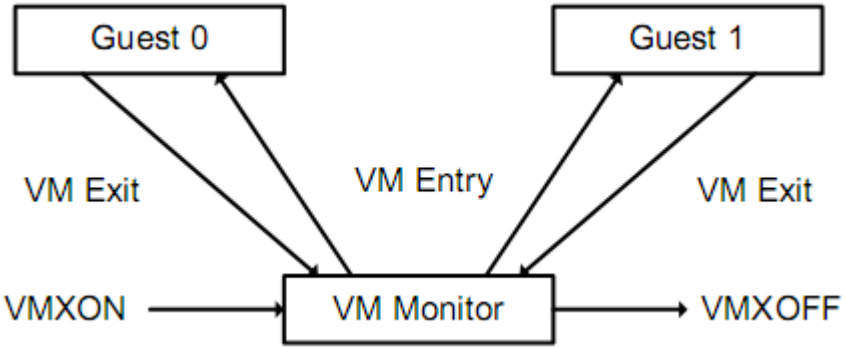
- A normal Linux process has two modes of execution: kernel and user
  - KVM adds a third mode: guest mode
- A virtual machine in KVM will be “seen” as a normal Linux process
  - A portion of code will run in user mode: performs I/O on behalf of the guest
  - A portion of code will run in guest mode: performs non-I/O guest code



KVM运行态切换

CPU Virtualization

- 基于Intel VT
  - Root mode
  - Non-root mode
- VMCS(虚拟机控制结构)
  - 保存vCPU需要的相关状态
  - 存放在内存中，每个VMCS对应一个vCPU
  - 使用时与物理CPU绑定
  - CPU发生vm_exit和vm_entry时会自动查询和更新vmcs
  - VMM可以配置VMCS，从而影响CPU行为



- VM exit
  - Sensitive instructions running in non-root trap into VMM
  - Saves Guest state in VMCS
  - Loads VMM state from VMCS
- VM entry (VMLAUCH VMRESUME)
  - CPU change from root to non-root
  - Loads Guest state and Exit info from VMCS

KVM Memory Virtualization

A day in the life of a KVM guest physical page

Memory allocation:
kvm-user call malloc to alloc 1GB mem

Memory register:
kvm-user register memory to kvm-kernel

Guest access:
Quest access GPA, will trap into kvm-kernel

Kvm-kernel handle page fault:
Kvm-kernel see a page fault, call do_page_fault() and allocate some memory to back it

Kvm-kernel create pte and spte:
kvm -kernel creates a pte_t to connect the malloc()'d virtual address to a host physical address

Guest Resume:
Host returns from page fault, guest execution resumes

缺页异常处理

Host memory pressure:
A page is found in inactive list, kernel

Reverse Mapping:
Host kernel uses rmap to find the host page table entry

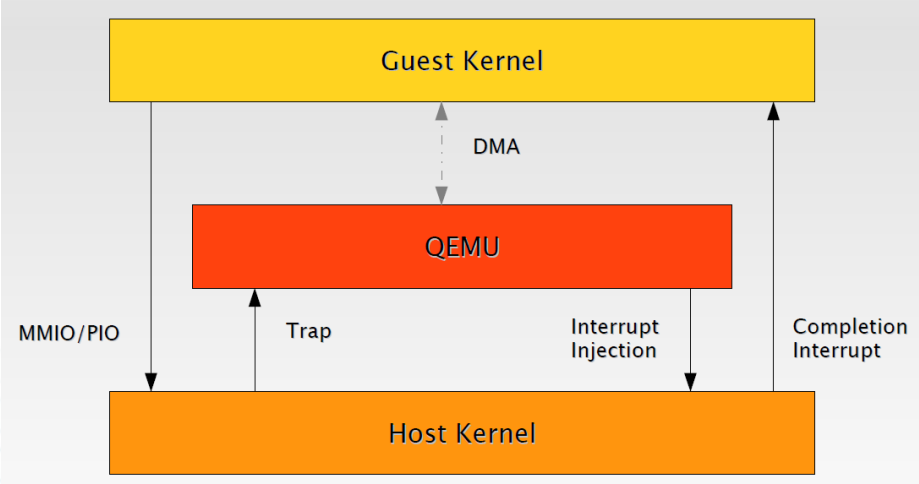
Host kernel swap the page:
swaps out the page and clears the pte_t

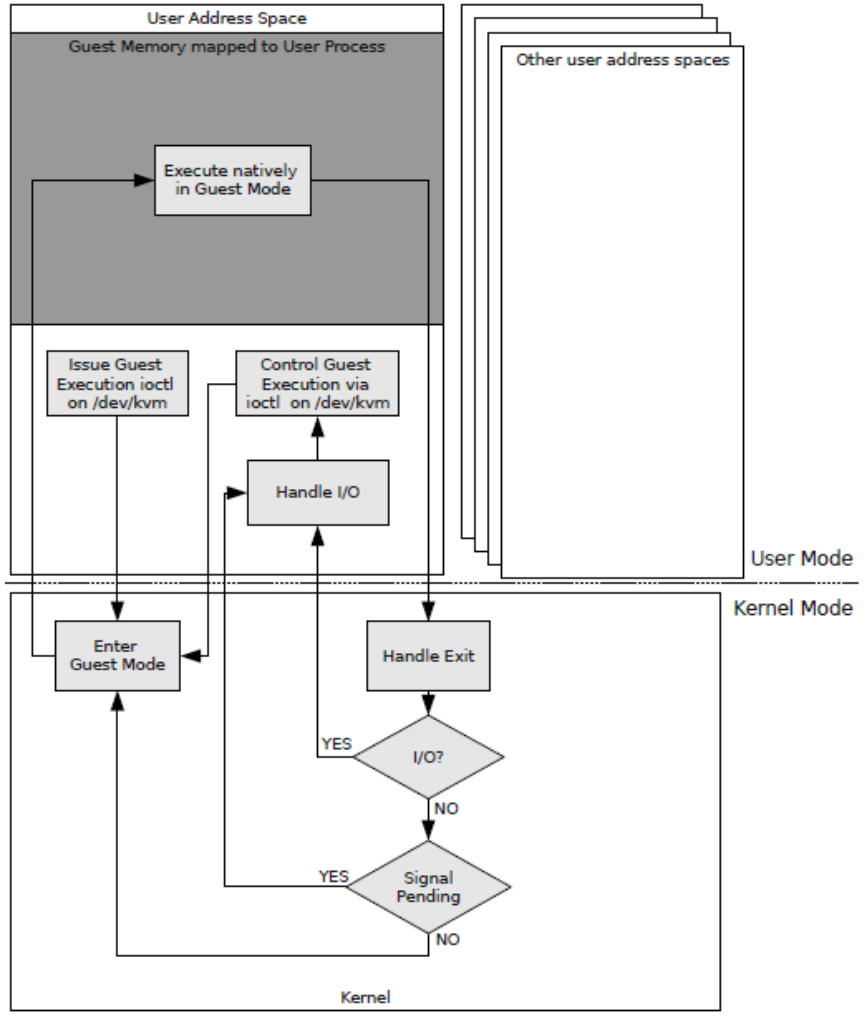
Invalidate the page:
mmu_notifier invalidate_page(), looks up the pte in the NPT/EPT structures and removes it

Subsequent Access:
will trap into the host in the fault-in path address

页置换处理流程

KVM I/O Emulation



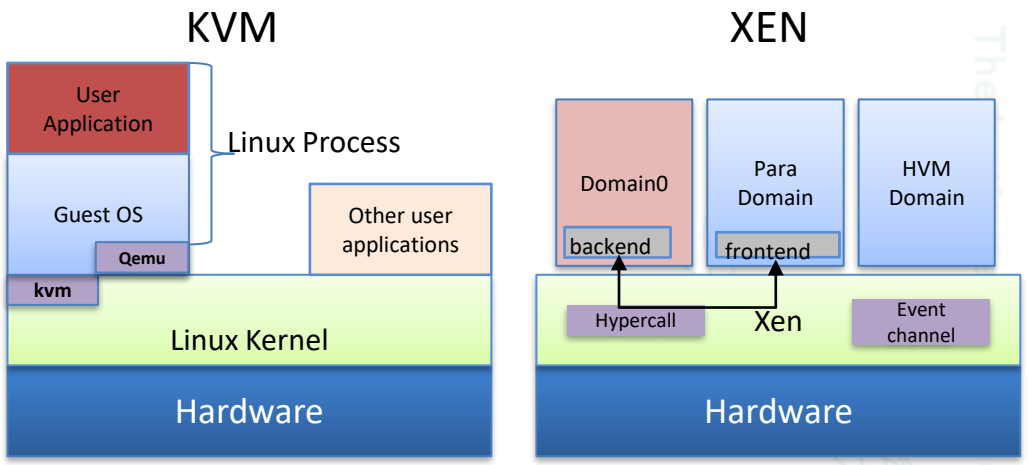


小结1：半虚拟化存在的问题

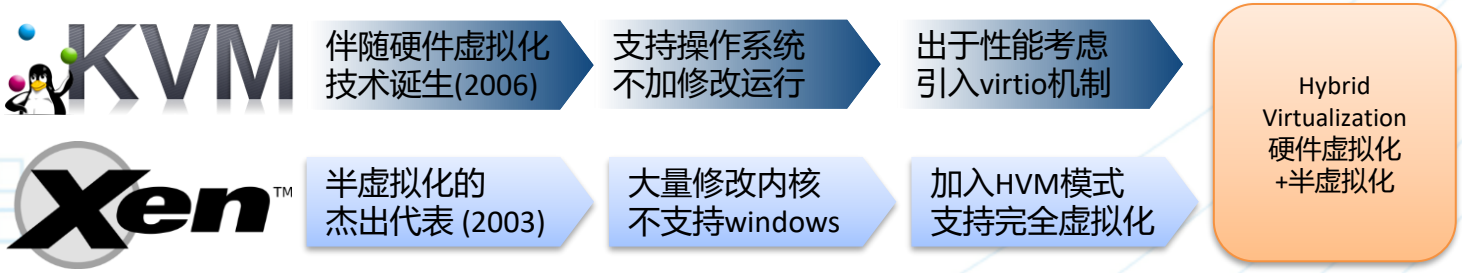
- 以下问题影响Xen Para成为企业级服务
  - 无法支持商用和非开源操作系统
    - 例如，Xen para不支持windows
  - Xen需要修改linux kernel
    - 对内核改动过大
    - 升级、维护困难
  - Guest OS和Host OS的kernel又不相同
    - 需要维护两套内核
  - Guest OS与VMM紧耦合，破坏了兼容性
    - VMM更新，需要重新编译Guest kernel

小结2：KVM vs XEN

- 系统实现方式不同
  - Kvm
    - 进入虚拟化模式时，kernel成为VMM
    - Linux功能强大，无需重复发明轮子
    - Guest虚拟机为Linux进程
  - Xen
    - Hypervisor位于最底层，运行和调度虚拟机
    - 需要修改Domain 0的操作系统
      - Code size大，维护成本高
      - 内核升级和移植困难
    - 基于Hypercall机制, I/O性能较高



- 技术路线
  - “殊途同归”



工业界及开源社区

- Xen
  - 优势: 工业界应用广泛，稳定性高、管理功能丰富、优异的性能表现
  - 劣势: 对内核改动过大，难以融入主流linux，维护升级困难
- KVM
  - 优势: Linux和Redhat重点支持，随内核升级而升级
  - 劣势: 不够稳定且功能尚不完善
- 开源云计算平台 (如OpenNebula, Eucalyptus等)
  - 通常Xen和KVM都支持

Redhat 6内置虚拟化技术

Redhat

Redhat 6放弃

CITRIX

amazon web services

redhat.

OpenNebula.org
The Open Source Toolkit for Cloud Computing

OpenNebula.org
The Open Source Toolkit for Cloud Computing

KVM

Xen™

redhat.

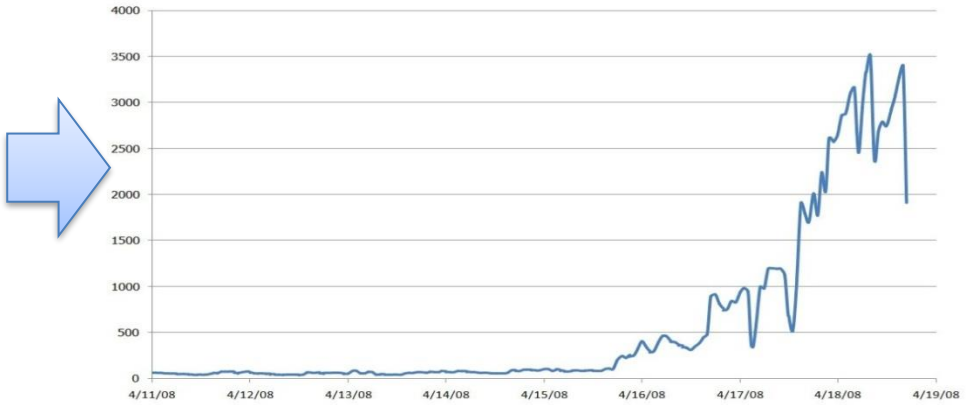
提纲

- 虚拟化原理
- 主流虚拟机分析（Xen和KVM）
- **虚拟化关键技术及其在数据中心中的应用**

互联网应用对资源的需求

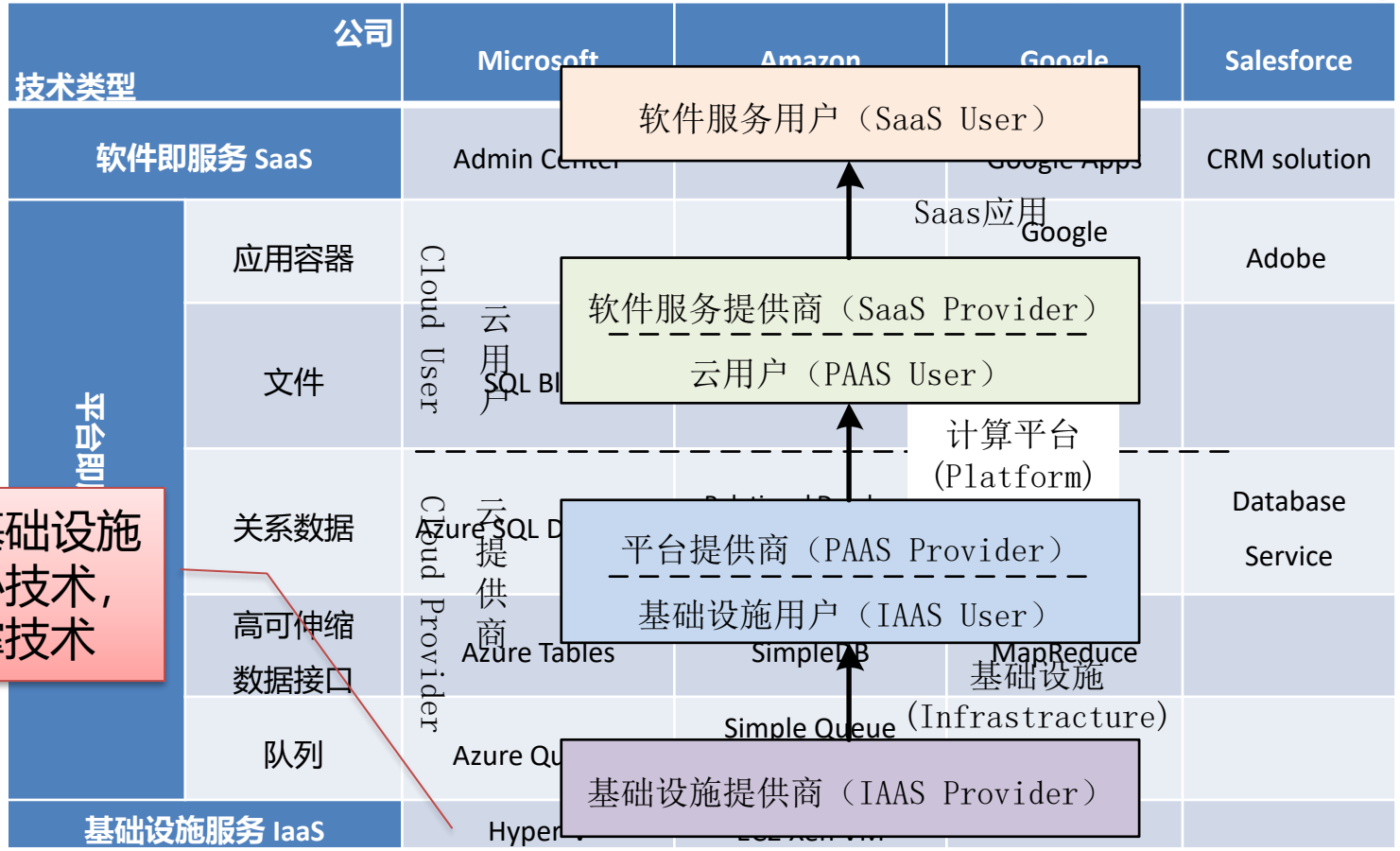
- 资源按需使用(On-Demand)
  - 1 支持资源的可伸缩性使用，无需对计算资源进行超前规划
  - 2 消除计算资源的进入门槛，允许从无到有，根据资源实际需求量的逐步增加计算资源的投入和消耗
  - 3 提倡资源的按需付费模式（Pay per Use），支持资源临时短期使用模式，及时释放多余资源

某Facebook的应用一周内资源使用量曲线。在三天之内该应用所用的虚拟机从不到50激增到3500



云计算技术分层

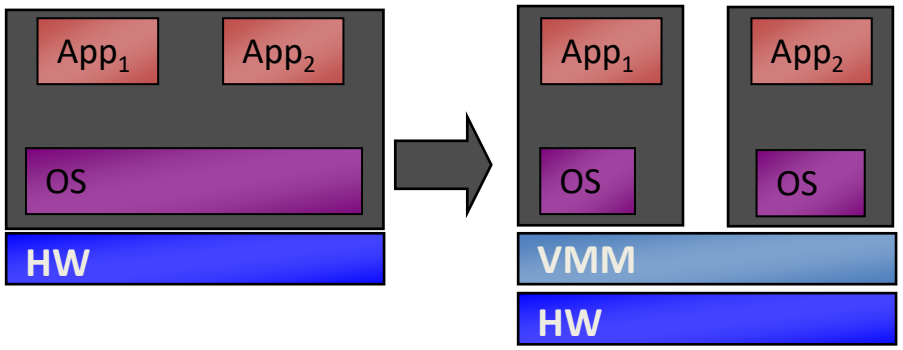
- 云计算技术包括：SaaS、PaaS、IaaS三个层次



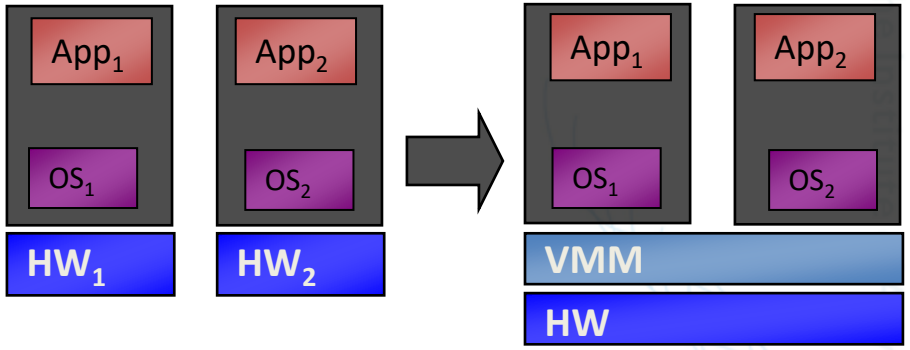
虚拟机技术是基础设施即服务层的核心技术，是上层的支撑技术

虚拟化的用途

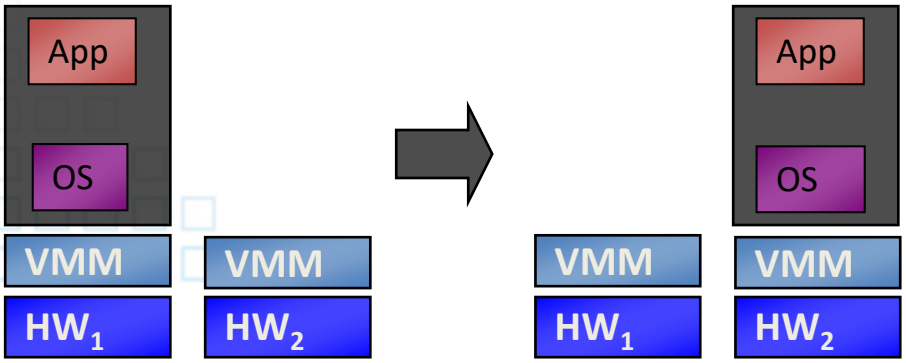
Workload Isolation



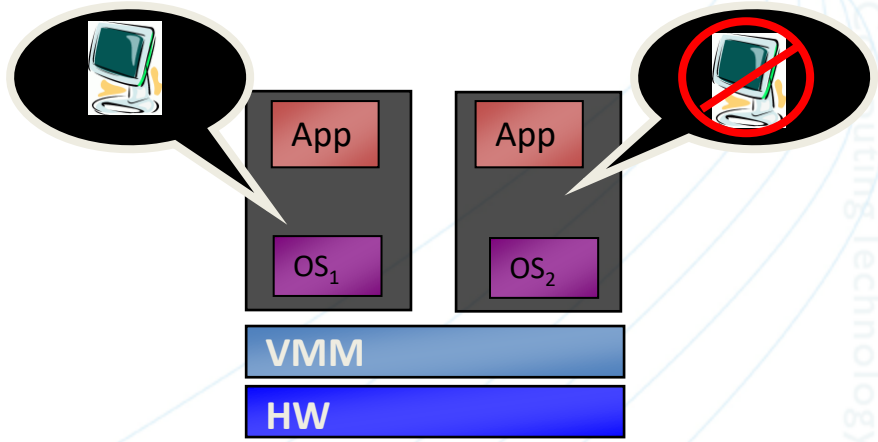
Workload Consolidation



Workload Migration

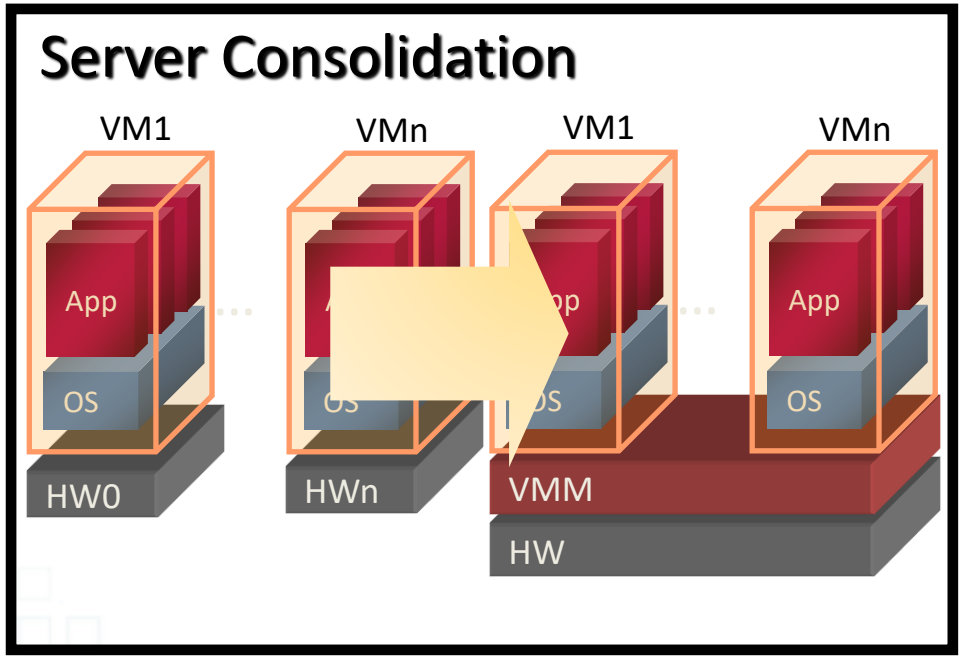


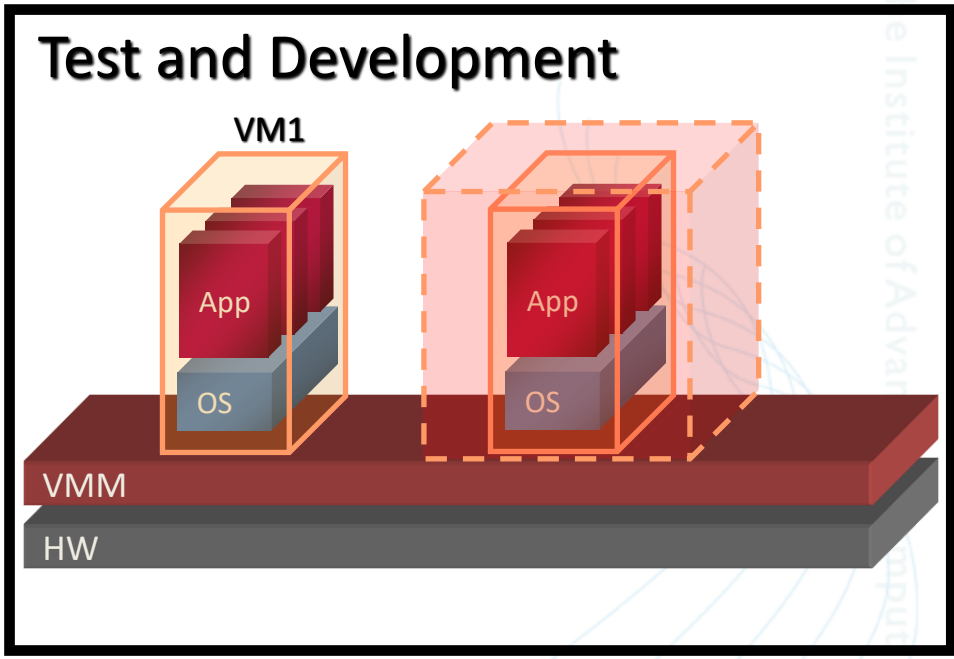
Workload Embedding



目前虚拟化的应用模式

Virtualization addresses today's IT concerns

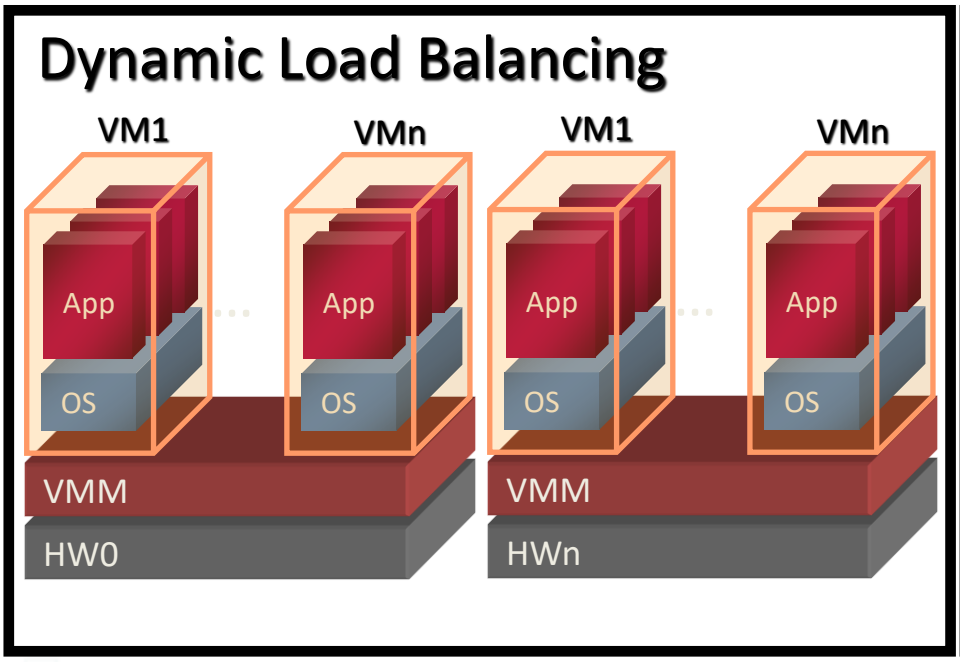


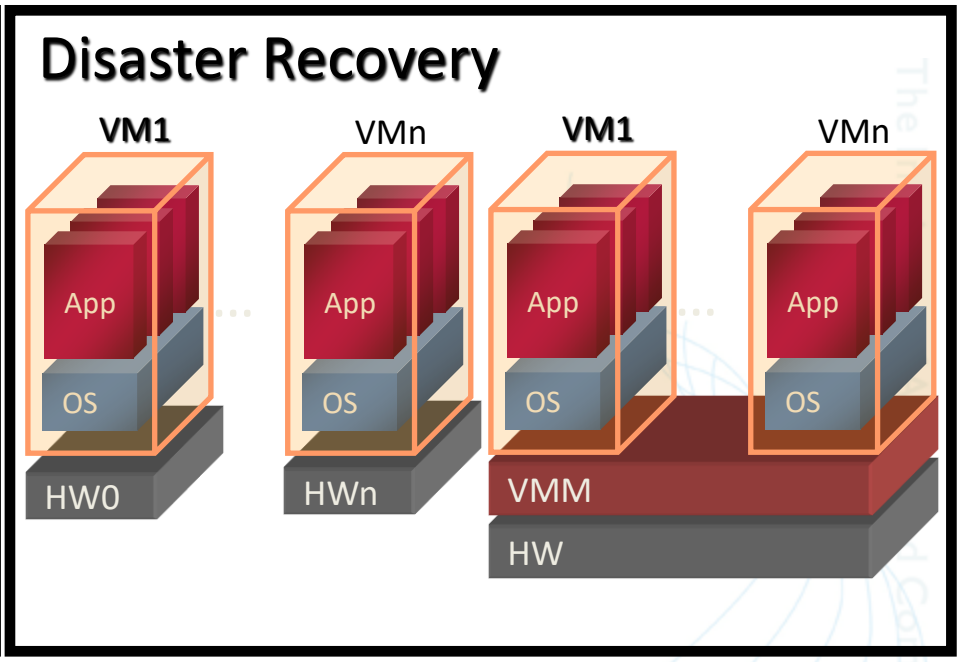


10:1 in many cases

Enables rapid deployment

其他潜在的应用模式





Goal: True “Lights Out” Datacenter

Instantaneous failover

Dynamic load balancing

Autonomics

Self healing

虚拟机迁移

- 无需停机
- 不影响上层应用
- 需要共享磁盘
- 对带宽要求较高

⇩

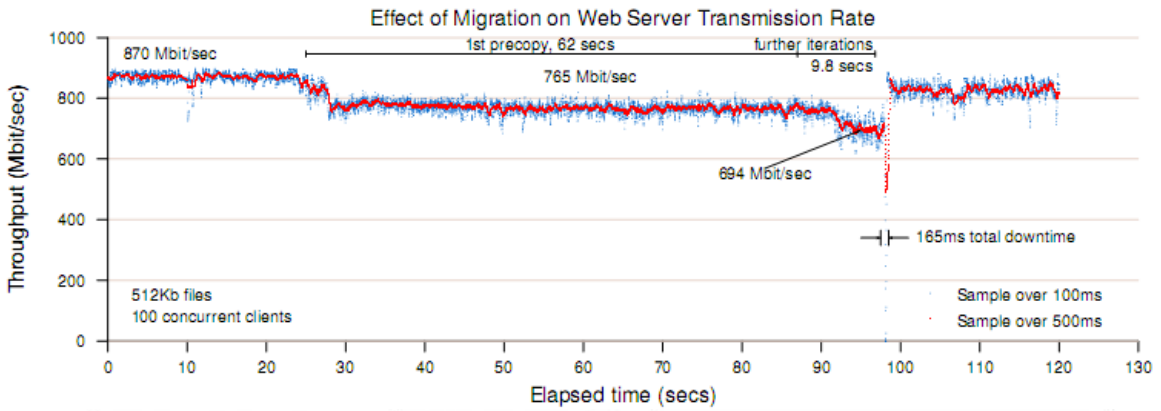
数据中心
DataCenter

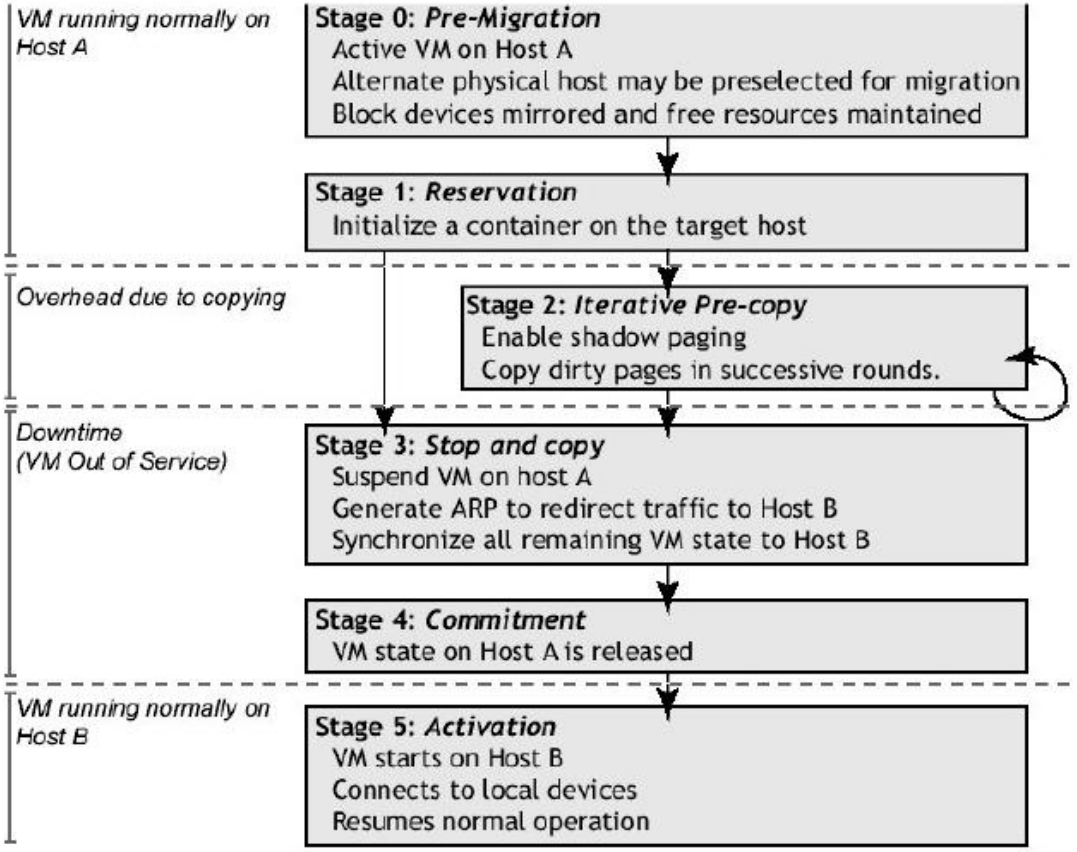
虚拟机迁移

- 无需停机
- 不影响上层应用
- 需要共享磁盘
- 对带宽要求较高

⇩

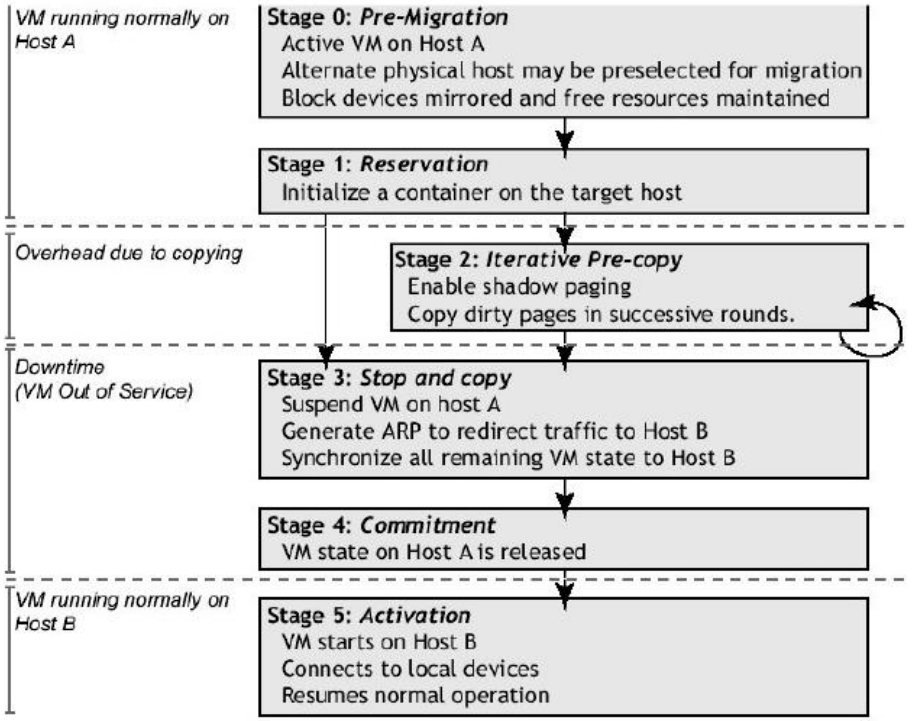
数据中心
DataCenter





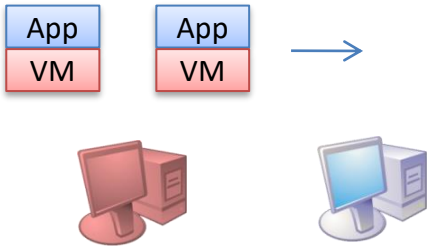
迁移技术

- 在线迁移技术
  - 预拷贝技术(pre-copy)
  - 后拷贝技术(post-copy)
  - 基于记录重放技术(log/replay)
- 迁移场景
  - 内存、磁盘迁移
  - 整个虚拟机迁移[Luo]
  - 虚拟机集群迁移[Deshpande]
  - 广域网、跨云平台迁移[CloudNet]

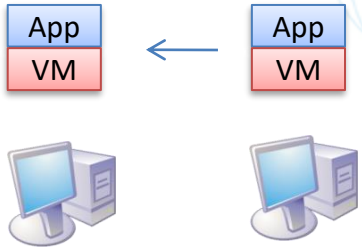


基于在线迁移的数据中心管理

- 为了提高资源利用率，基于虚拟机的数据中心资源管理需要应对以下两种情况：



负载均衡
平衡负载以提高应用响应速度



负载汇聚
将低负载任务汇聚以节约能耗

- 虚拟机迁移是其中的关键

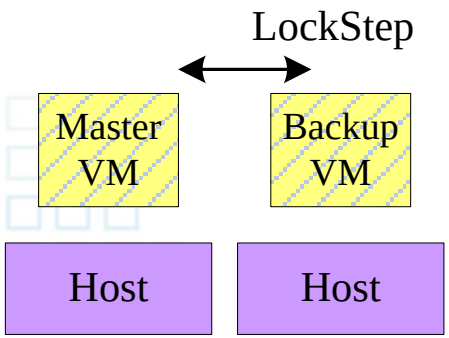
故障恢复

记录虚拟CPU的指令流和非确定性事件的插入时间，能完整地重现虚拟机运行生命周期中的任意时刻
既能重现执行过程，支持故障诊断、入侵检测；也可以保存运行状态，并在虚拟机发生故障时进行恢复

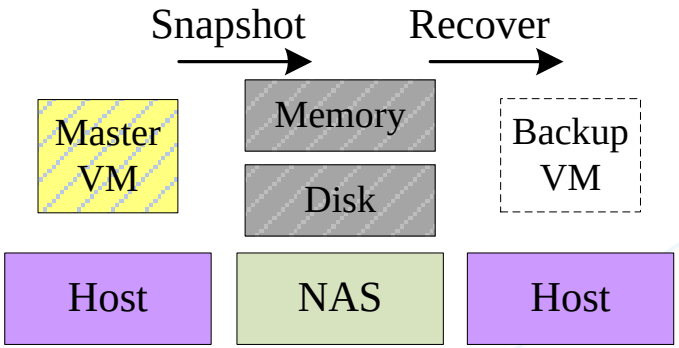
- 虚拟机快照是执行状态的持久化表示
- 包括内存、处理器、虚拟设备等信息
- 虚拟机失效后通过快照，能够恢复虚拟机的运行

通过增量方式实现快照同步，能提高快照创建频率，降低故障恢复代价，实现非预期故障的快速恢复
原宿主机和备份宿主机之间具有较强耦合关系，只能实现一对一的备份

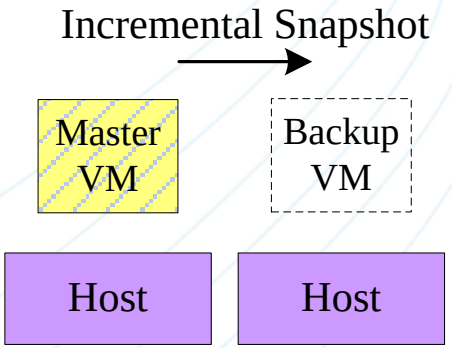
指令回放



快照恢复



持续备份

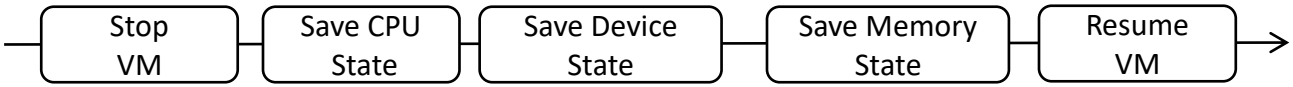


虚拟机快照（Snapshot）

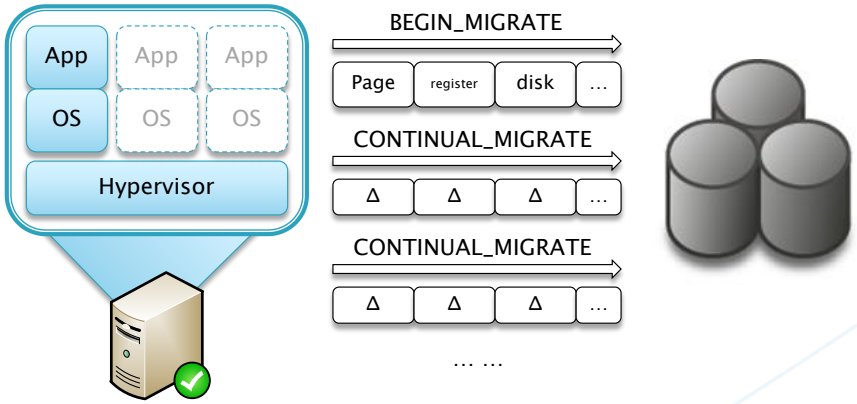
- What is a snapshot
  - the state of a system at a particular point in time
- Why uses snapshot
  - Fault-tolerance（故障恢复，备份还原）
  - Integrity recovery and intrusion analysis
  - Rollback debugging and Testing
    - Time travel
  - Snapshot种类
    - 进程
    - 数据（数据库、磁盘、逻辑卷和文件系统）
    - 操作系统
    - ***虚拟机***
    - 分布式系统（网络）

内存快照

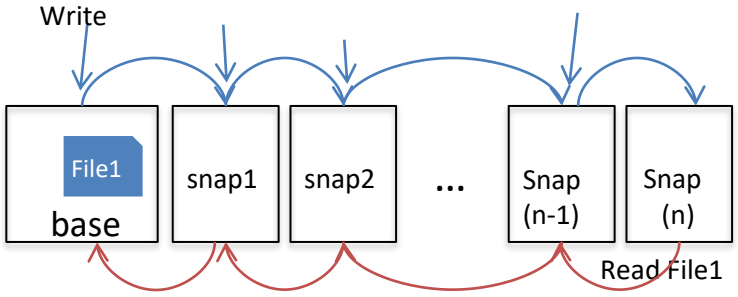
- Cold snapshot



- Live snapshot

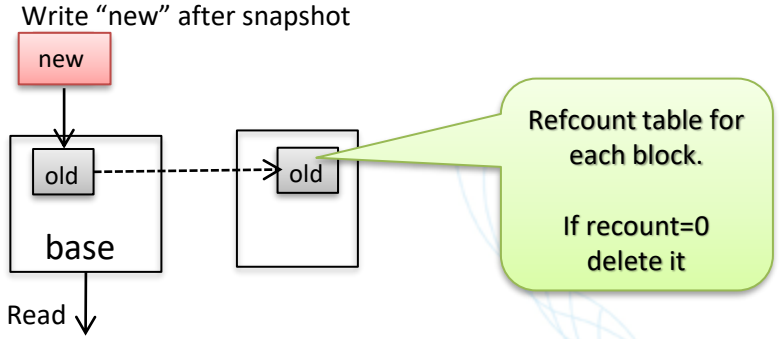


磁盘快照



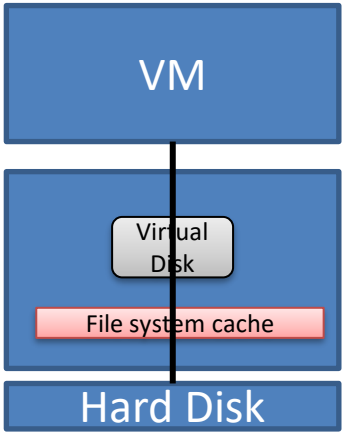
Redirect on Write

$Snap_x$=base+(snap(1)+…+snap(x))



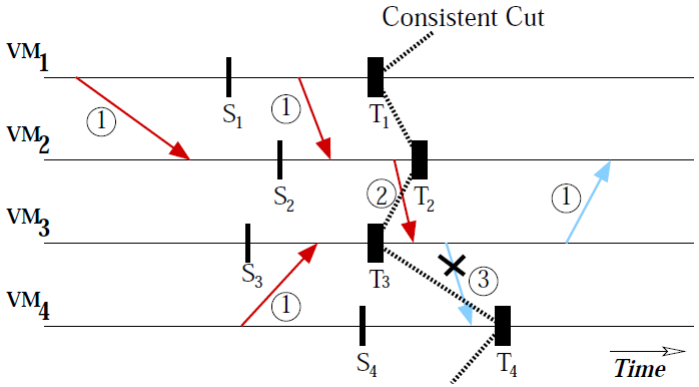
Copy on Write

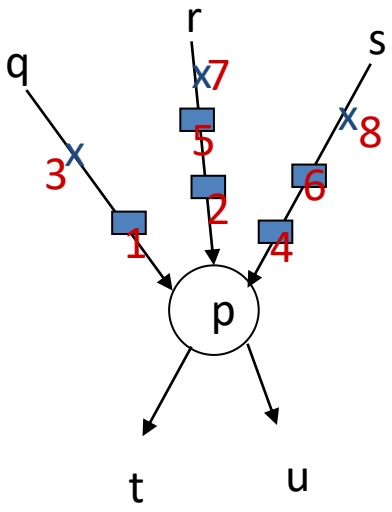
Snap=Current-new+old



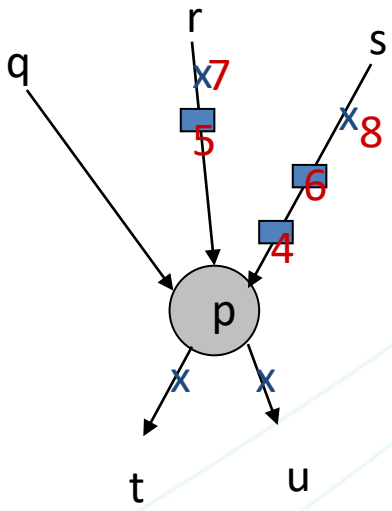
type	Flag	details	default
Cache None	O_DIRECT	No cache for file data; has cache for metadata	
Write through	O_SYNC	读写会通过cache，但对于写操作，只有真正写到存储上才会返回写完	raw
Write back		Return as soon as the data is in memory	qcow2

虚拟网络的一致性网络状态快照



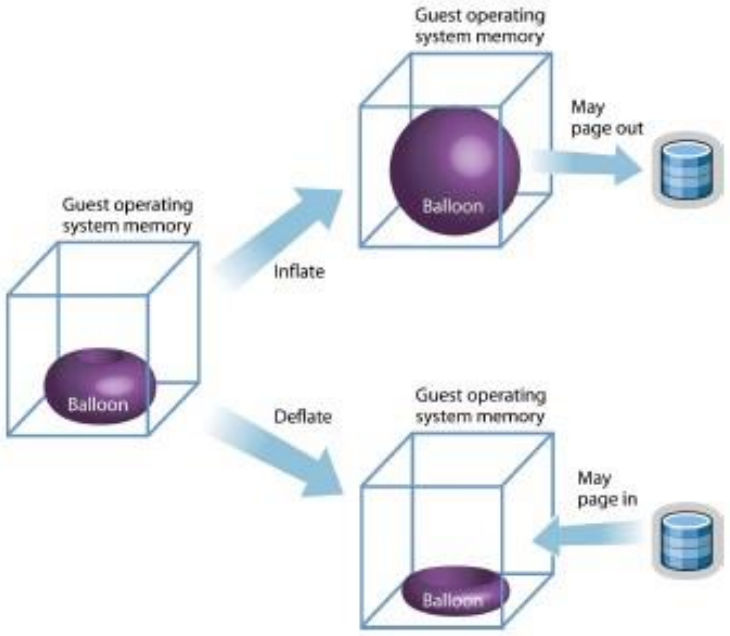


(1) p is receiving messages

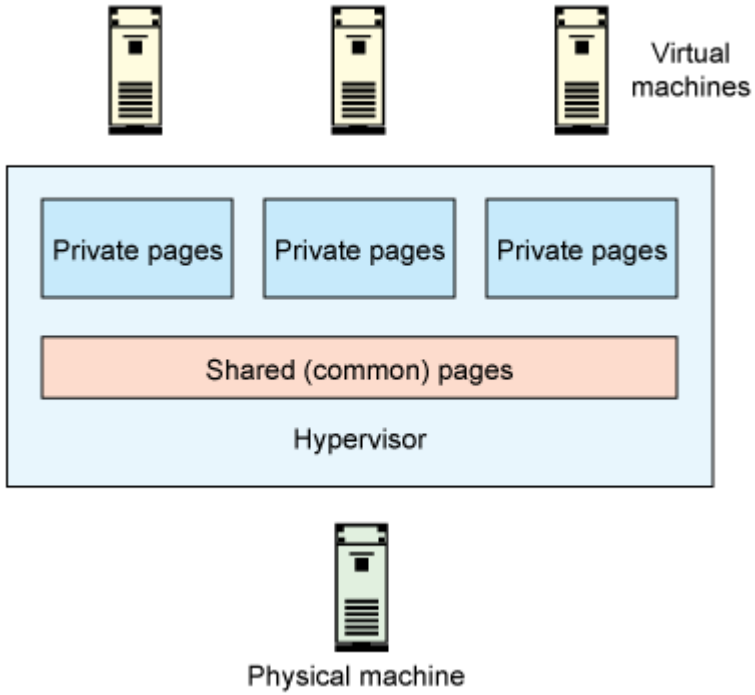


(2) p has just saved its state

内存使用的优化：弹性内存



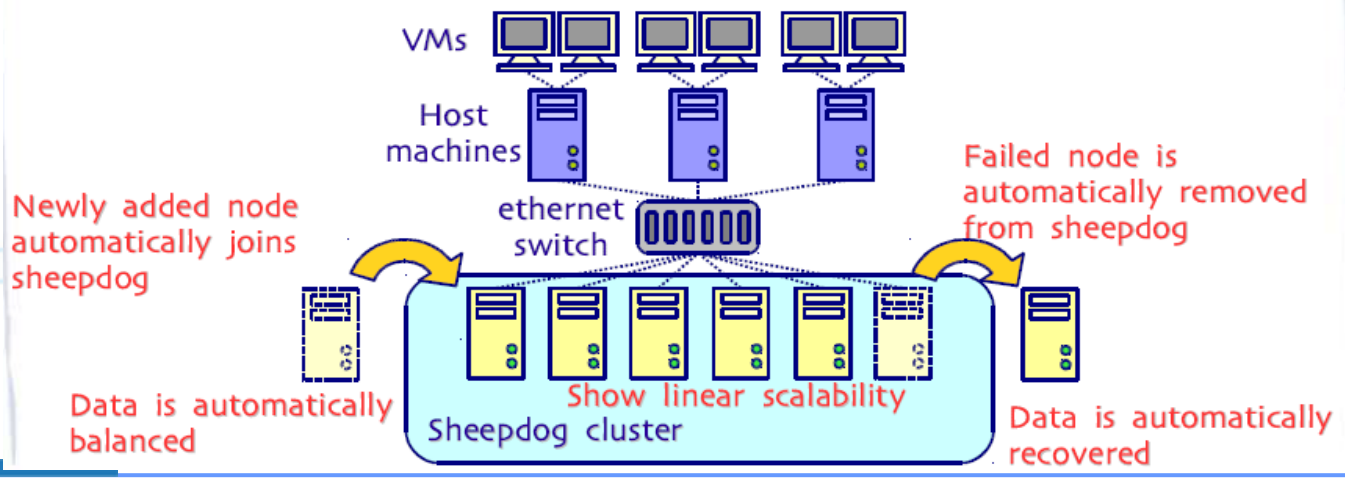
Memory Ballooning



KSM in kvm

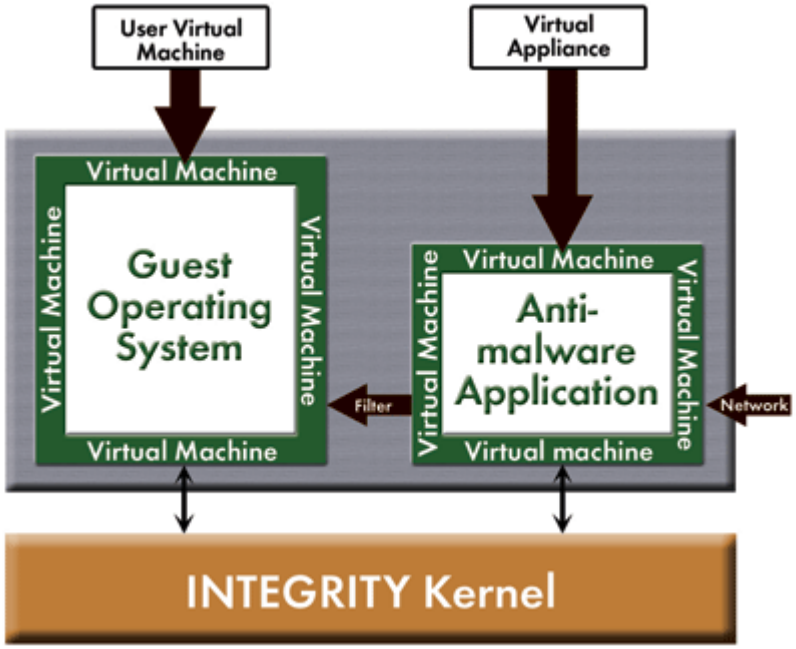
虚拟磁盘优化

- Copy-on-write virtual disk
  - One base image, multiple cow images
- Advanced virtual disk format
  - 支持cow方式
  - 支持快照与回滚
  - 支持加密和压缩
- Virtual Disks over Distributed file system



虚拟电器（Virtual Appliance）

- A **virtual appliance** is a *pre-integrated, self contained* system that is made by combining a software application (e.g., server software) with just enough operating system for it to run optimally on industry standard hardware or a virtual machine (e.g., VMWare, VirtualBox, Xen HVM, KVM).
- Simplified deployment: A software appliance encapsulates an application's dependencies in a pre-integrated, self-contained unit. This can dramatically simplify software deployment by freeing users from having to worry about resolving potentially complex OS compatibility issues, library dependencies or undesirable interactions with other applications.
- Improved isolation: virtual appliances are typically used to run applications in isolation from one another. If the security of a virtual appliance is compromised, or if the virtual appliance crashes, other isolated virtual appliances will not be affected.

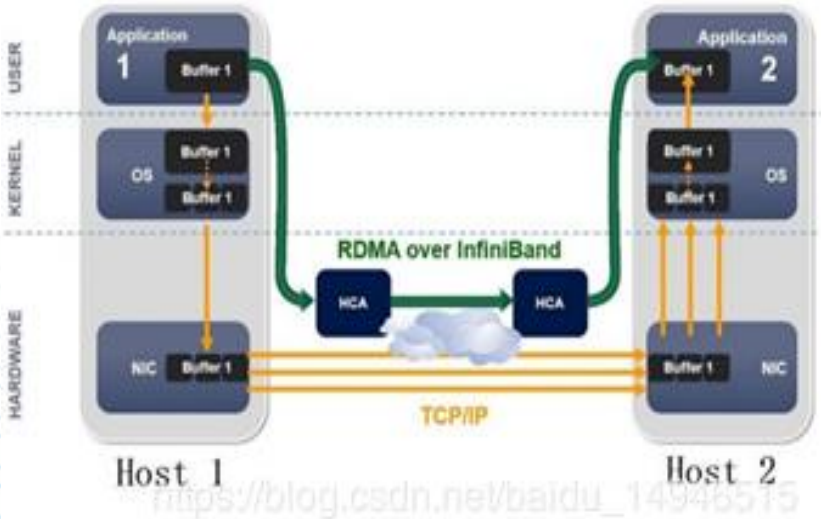


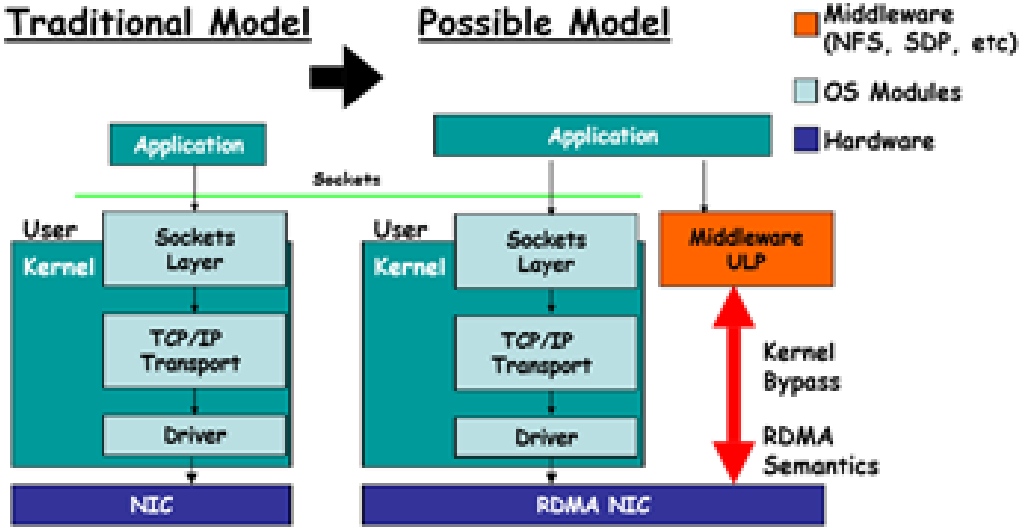
小结及研究点

- **知识点**
  - 虚拟化的基本原理
  - 基本资源的虚拟化方法（CPU、内存、外设）
  - 两个代表性开源系统：Xen, KVM
  - 虚拟机的应用：热迁移、安全、容错、快照
- **研究思路和热点**
  - 更加高效：轻量化、实时性、提升性能（和不虚拟化比较）
  - 采用新的硬件优化：VT-d，多端口网卡，RDMA
  - 新指令集的CPU虚拟化：如ARM虚拟化
  - 新型设备虚拟化：GPU、FGPA、机器学习加速器……
  - 广域网远程虚拟机迁移
  - 从“一虚多” 到“多虚一”

RDMA：远程直接内存访问

- 数据通过DMA方式拷贝到NIC(网卡)，再通过网络传输到远端NIC，然后直接到达远端内存，而无需操作系统多次在缓冲区指间拷贝数据和CPU参与





- 图片来源：https://blog.csdn.net/baidu_14946515/article/details/84201240

参考文献

- 1.Xen and the ART of Virtualization
- 2. Clark C., Fraser K., Hand S., et al. Live migration of virtual machines[A]. Proceedings of the 2nd conference on Symposium on Networked Systems Design and Implementation - Volume 2[C]. USENIX Association, 2005: 273-286.
- 3.Jianbao Ren,Yong Qi, Yuehua Dai, Xiaoguang Wang,Yi Shi. AppSec: A Safe Execution Environment for Security Sensitive Applications. In VEE '15 Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments
- 4.D. Lie, C. A. Thekkath, and M. Horowitz. Implementing an untrusted operating system on trusted hardware.In SOSP, pages 178–192, 2003.
- 5.H. Lee, H. Moon, D. Jang, K. Kim, J. Lee, Y. Paek, and B. B. Kang. Ki-mon: a hardwareassisted eventtriggered monitoring platform for mutable kernel object.In USENIX Security, pages 511–526, 2013.
- 6.O. S. Hofmann, S. Kim, A. M. Dunn, M. Z. Lee, and E. Witchel. Inktag: secure applications on an untrusted operating system. In ASPLOS, pages 265–278, 2013.
- 7.Y. Fu and Z. Lin. Exterior: using a dual-vm based external shell for guest os introspection, configuration, and recovery. In VEE, pages 97–110, 2013.
- 8.John C, Nanthan D. Virtual Ghost: Protecting Applications from Hostile Operating Systems. In ASPLOS 2014

谢谢！